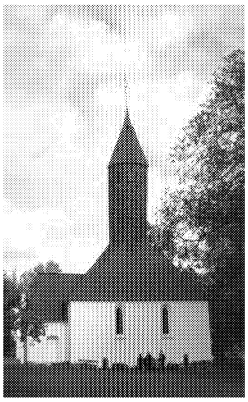
MÄLARDALENS HÖGSKOLA

ESSES 2003
European Summer School on Embedded Systems

Lecture Notes
Part XVIII

Real-Time Systems: Introduction and Overview



European Summer School on
Embedded Systems
Jul 14 - Oct 10 2003
Västerås Sweden

Editors: Ylva Boivie, Hans Hansson, Jane Kim, Sang Lyul Min

Västerås, September 22-25, 2003
ISSN 1404-3041
ISRN MDH-MRTC-111/2003-1-SE

MRTC
MÄLARDALEN REAL-TIME
RESEARCH CENTRE

www.mrtc.mdh.se

Controlling the Performance of Networked Information Servers

Lui Sha
lrs@cs.uiuc.edu

Sept 2003

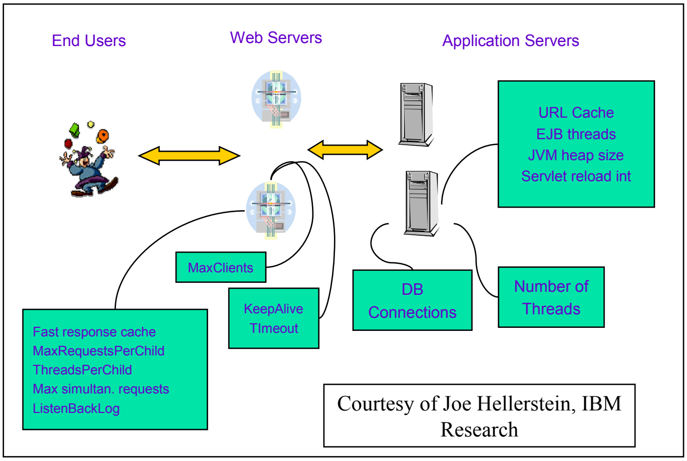
Controlling Computers

- Feedback control was embedded in the TCP protocol in the 70's to solve the congestive failure problems that had brought down the network.

- Since then, we have not experienced system-wide congestive failures again even though the network has grown orders of magnitude. This is a testament of the effectiveness of feedback control in a highly dynamic, decentralized, and fast changing environment.

- However, except under heavy workload conditions that allow effective fluid approximations, the application of control theory to control the performance of computing systems has been slow. But the need of performance control becomes more pressing.

Web Service Performance Control



End Users

Web Servers

Application Servers

URL Cache
EJB threads
JVM heap size
Servlet reload int

MaxClients

KeepAlive
TImeout

DB
Connections

Number of
Threads

Fast response cache
MaxRequestsPerChild
ThreadsPerChild
Max simultan. requests
ListenBackLog

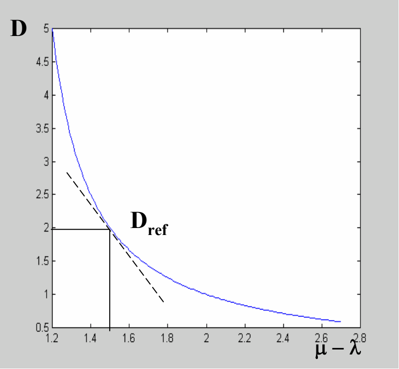
Courtesy of Joe Hellerstein, IBM Research

The Problem

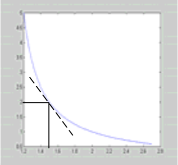
- From a control theory perspective, a networked information server’s behavior is highly non-linear. The parameters of the stochastic process, e.g., request rates, can change abruptly without warning. They are:
  - Not deterministic processes perturbed by random events like LQG
  - Not just the long term equilibrium behavior control like MDP
  - Not just an isolated plant but networked servers
- Yet, we want to tightly control both the long term average and the transient behaviors of these random processes.

The Difficulties

- Linearized controller provides only mediocre performance. A direct application of adaptive control or hybrid control does not provide much improvement
  - Workloads are as fickle as Web servers' attention spans
  - Random fluctuations in key variables makes state estimation time consuming
  - Control action itself pushes the system away from the selected/updated model at runtime.

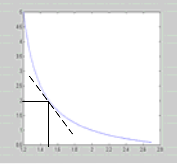


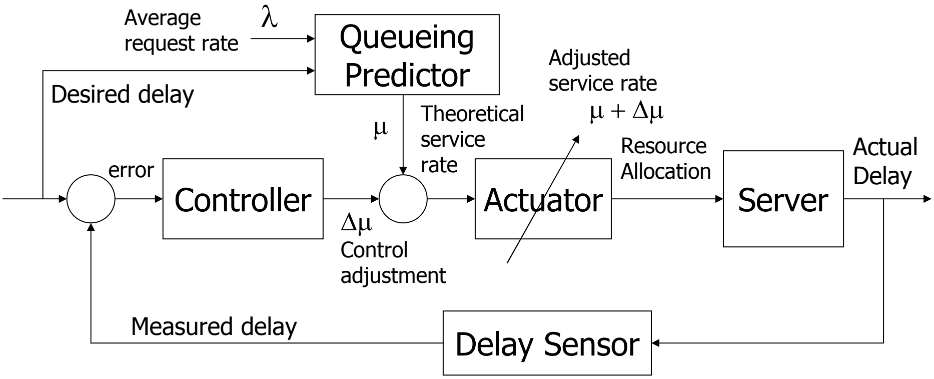
Key Ideas



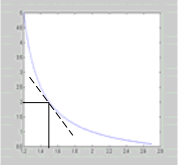
- Model accuracy impacts control performance. And all the difficulties point to a single problem: the actual plant deviates from the plant model used by control.
- Queueing model is a "natural" to model the non-linear behaviors of an information server over a wide range of parameter values
- Why not using a queueing model's solution as a feed forward control to "lock" the system into a desired equilibrium operating point, in spite of workload changes?

Delay Control





“Marriage” in Heaven



- Consider the M/M/1 model where D = **1/(**$\mu - \lambda$**).** The feed forward control from queueing model, $\mu(\tau)$ = 1/D_ref + $\lambda(\tau)$, “locks” the system in an equilibrium state in the neighborhood of linearization.
- This makes the life of a feedback control easier. In return, the feedback controller suppresses approximation errors in the queueing model and the transients that cannot be reduced by queueing modeled based tuning.
- To the best of our knowledge, this is the first framework that has successfully integrates two powerful theories.

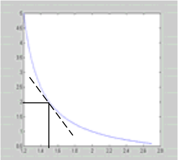
New Perspectives

	Control Physical Plants (Changing Eigenvalues)	Control Web Servers (Changing Probability)
Periods measured in	Units of time	Number of events
Sampling rate for signal processing	Faster than Nyquist rate	Slower than 1/Large_Sample_Size
Observations	High frequency jumps are usually noise	All the fluctuations are part of data
Modeling identification	Large excitation is good as long as it does not cause saturations	Small excitation for many epochs

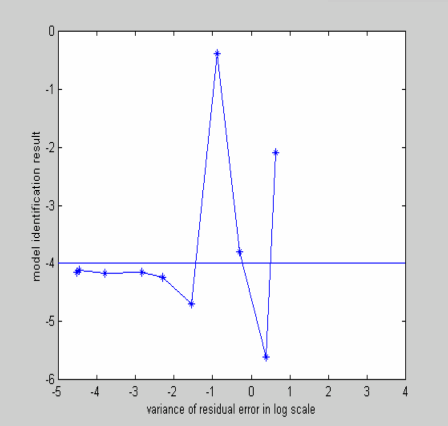
Controlling Probabilities

- Suppose we want to correct a biased coin with prob(head)= 0.4. We begin by soldering a small weight to the side of head of the biased coin, and then do some experiments and adjust the weight…
- How frequently should we adjust the weight? Obviously, we cannot succeed if we change the weight, just flip the coin once, and change the weight again
- The problem here is that we are dealing with *probability*, and not with an instantly measurable quantity such as temperature or pressure.
- From a control perspective, the transfer function (relationship) between the change of weight and the change of probability of head *manifests itself only when the sample variance becomes negligible.*

Small Excitation and Slow Sampling



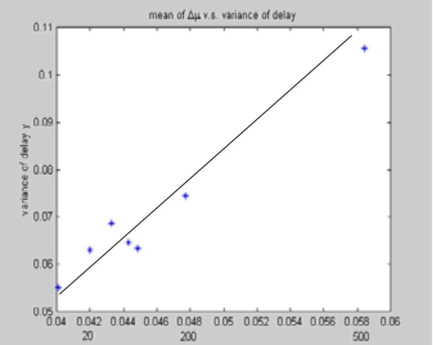
- The horizontal axis is the sample variance in log scale.
- Under a small excitation of *Δμ = ±0.01,* when the sample contains many epochs, the sample variance becomes very small and Matlab's estimation using experimental data converges towards the theoretical value of *–4*.
- Under a large excitation, e.g., *Δμ = ±0.1,* the estimation would converge to a wrong value, because the asymmetry response of queueing system.



Sample Variance and Model Accuracy

Sliding Window Control Action

- Long observation window does not imply slow control action
- For example, a 1000 events window but change control output every 10 events
  - X-axis is step size (also shows avg control effort Δμ)
  - Y-axis is delay variance
- Quick update step reduces variance & control efforts.



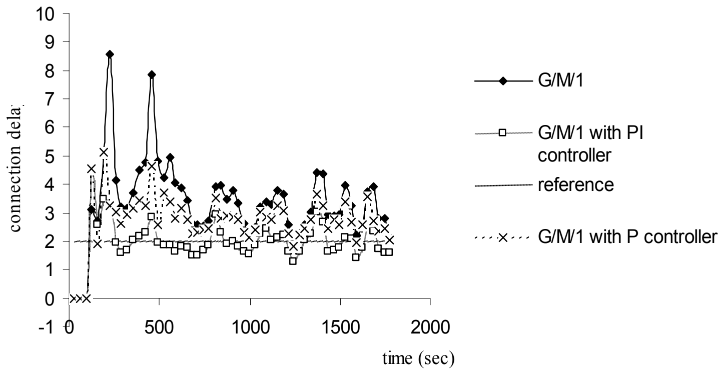
Effect of Control Update Rates with a Suitably Long Observation Window

Experimental Setup

- Objective: Control absolute delay of premium clients on an Apache web server
  - Sensor: averages response time of all requests
  - Actuator: Number of processes (C. Lu, RTAS 2001)
- Load generation: SURGE web benchmark
- Platform: Linux-based PC cluster on Ethernet LAN

Experimental Results

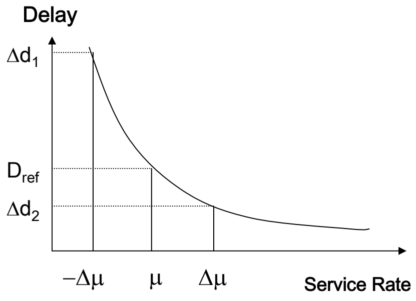
G/M/1 model with PI controller produces much better results



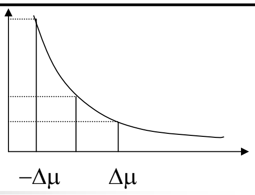
Asymmetric Control

The effect of reducing resource is much more profound than adding resources, especially when workload is heavy.

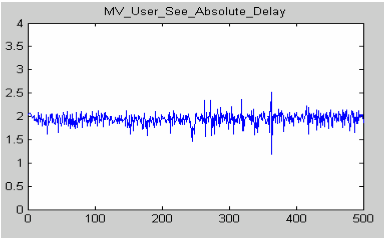
	Asymmetric Control	**Symmetric Control**
Ref Delay	3.0	3.0
Mean	2.9942	3.0850
Variance	0.0821	0.2342

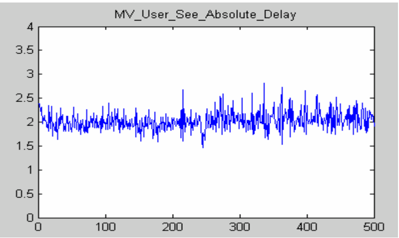


Asymmetric Control



The effect of reducing resource is much more profound than adding resources, especially when workload is heavy.





Asymmetric Control

Symmetric Control

Summary

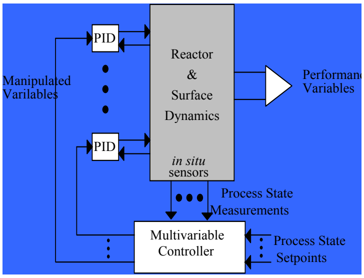
- Providing high quality performance control in a single server is an important first step.
- Looking ahead, we will address issues:
  - Non-linear controllers
  - Adding queue size prediction, measurement and control
  - Performance control of server farms and network of servers
  - Testing results in real system, e.g., servers at IBM Research

1. Sha, L., Li, X., Lu, Y., and; Abdelzaher, T. "Queueing model based network server performance control", the proceedings of IEEE Real-Time Systems Symposium, 2002
2. Lu, Y., Abdelzaher, T., Lu, C., Sha, L., and Liu, Xu, "Feedback Control with Queueing-Theoretic Prediction for Relative Delay Guarantees in Web Servers", to appear in The 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2003)
3. Liu, X., Sha, L., Diao, Y., Froehlich, S., Hellerstein, J., and Parekh, S., "Online Response Time Optimization of Apache Web Server" proceedings of the 11th International Workshop on Quality of Service

Updating RT Embedded Software in the Field

Lui Sha
lrs@cs.uiuc.edu

September, 2003



RT embedded systems have a long life span. How to develop real time systems that can:

- be easily changed in the field, even on the fly?
- maintain stability and controllability in spite of
  - arbitrary errors in the new software?
  - malicious attack by insiders disguised as upgrades?

Job 1 is Robust Against Bugs

- We shall begin with an investigation on the principle of developing software systems that are robust against bugs. Leaving them alone, bugs may destroy:
  - Correctness
  - Performance
  - Reliability
  - Security
  - …
  - any software property that you care.

The Software Reliability Conundrum

- If history is any guide, formal methods can only handle software with moderate complexity in the foreseeable future.

- How about using software tolerance based on diversity?

- But wait. What if the fault tolerance system is itself too complex to verify and have faults?

- For example, the Six Western States Blackout incident in US was
  - triggered by the shorting of 1 power line at Oregon
  - spread by the flawed "self healing" architecture at the time

Complexity, Diversity and Reliability

- To build a robust software system that can tolerant arbitrary application software faults, we must understand the relations between software
  - Complexity: the root cause of software faults
  - Diversity: a necessary condition for software fault tolerance.
  - Reliability: a function of complexity and diversity
- We shall begin with postulates based self-evident facts

Software Development Postulates

- We assert that the following postulates self-evident
  - P1: Complexity Breeds Bugs: Everything else being equal, the more complex the software project is, the harder it is to make it reliable.
  - P2: All Bugs are Not Equal: You fix a bunch of obvious bugs quickly, but finding and fixing the last few bugs is much harder.
  - P3: All Budgets are Finite: There is only a finite amount of effort (budget) that we can spend on any project.
- *How can we model “software complexity”?*

Logical Complexity

- Computational complexity => the number of steps in computation.
- Logical complexity => the number of steps in verification.

- A program can have different logical and computational complexities.
  - Bubble-sort: lower logical complexity but higher computational complexity.
  - Heap sort: the other way around.

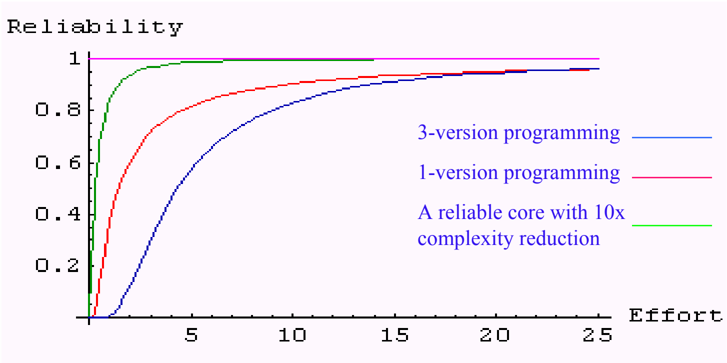
- Residue logical complexity. A program could have high logical complexity initially. However, if it has been verified and can be used as is, then the residue complexity is zero…

The Implications of the 3 Postulates

- P1: Complexity Breeds Bugs: For a given mission duration *t*, the reliability of software decreases as complexity increases.
- P2: All Bugs are Not Equal: for a given degree of complexity, the reliability function has a monotonically decreasing rate of improvement with respect to development effort.
- P3: Budgets are finite: Diversity is not free. That is, if we go for n version diversity, we must divide the available effort n-ways.

- One simple model that satisfies P1, P2 and P3
  - Sum of efforts used in diversity = available effort
  - Reliability function: $e^{-k\,(complexity\,/\,effort)\,t}$

Diversity, Complexity and Reliability



- Analysis shows that what really counts is not the degree of diversity. Rather it is the existence of a simple and reliable core that can guarantee the stability of the system. This result is also robust against change of model assumptions.

  --- *Using Simplicity to Control Complexity, IEEE Software 7/8, 2001, L. Sha*

Putting the Principle to Work

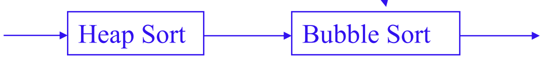
- Complexity is
  - The side effect of features and performance
  - The root cause of software faults
- It is kind of like money … a source of many evils but something we cannot live without.
- So let’s find a way to control complexity, instead of letting it control our systems.

An Example

- Once upon a time, there was an exam on sorting programs. Grades are given as follows:
  - A: Correct and fast: n log (n) in worst case
  - B: Correct but slow
  - F: Incorrect
- Joe can verify his bubble sort, but has only 50% chance to write Heap Sort correctly.
- What is his optimal strategy?

Requirement Decomposition

- Often, requirements can be decomposed into
  - Critical (correctness) requirements
    - Sorting: output numbers in correct order;
    - TSP: visit every city exactly once
    - Control: stable and controllable
  - Performance optimization
    - Sorting: faster
    - TSP: shorter path
    - Control: less time/error/energy
- Joe can exploit software he cannot verify safely …



Stability Control

- Stability control is a mechanism that ensures that errors are bounded in a way that satisfies the preconditions for the recovery operations. Stability control must be simple or it will be self defeating.

- What if the untrusted sorting program alters an item in the input list?
  1. Create a verified simple primitive called “permute”
  2. Untrusted sorting software is not allowed to touch the input list except use the permute primitive.
  3. Enforce the restriction using an object with (only) method “permute”

- Under stability control, the untrusted Heap-sort can only produce “out of order” application errors.

Using Simplicity to Control Complexity

The high assurance control subsystem

- *Application level*: well-understood controllers to keep the control software simple.
- *System software level*: certified OS kernels
- *Hardware level*: well-established and fault tolerant hardware
- *System development:* high assurance process, e.g. DO178B
- *Requirement management*: critical properties and essential services.

The high performance control subsystem

- *Application level:* advanced control technologies,
- *System software level*: COTS OS and middleware
- *Hardware level:* standard industrial hardware
- *System development:* standard industrial development processes.
- *Requirement management*: features, performance & rapid innovation

Stability Control for Control Systems

- Having a reliable controller, we identify the recovery region within which the controller can operate successfully. Recovery region is a subset of the states that are admissible with respect to operational constraints

- The largest recovery region can be found using LMI. This approach is applicable to any linearizable systems. They cover most of the practical control systems.

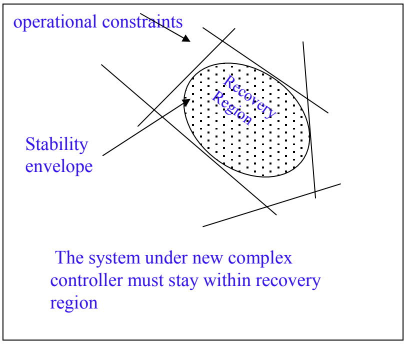
$\dot{X} = AX$

$A^T Q + QA < 0$

min $\log \det Q^{-1}$

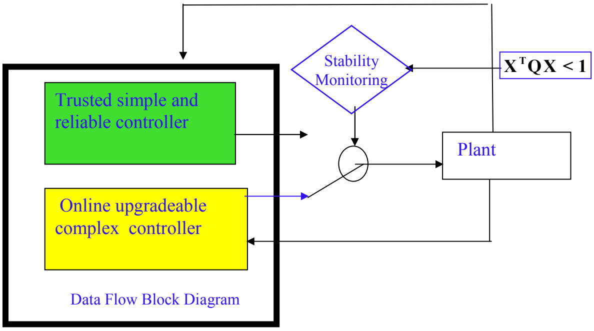
subject to $C^T X < 1$

Safety switching rule: $X^T Q X < 1$



The system under new complex controller must stay within recovery region

Simplex Architecture for Control



Stability Monitoring

$X^T Q X < 1$

Trusted simple and reliable controller

Plant

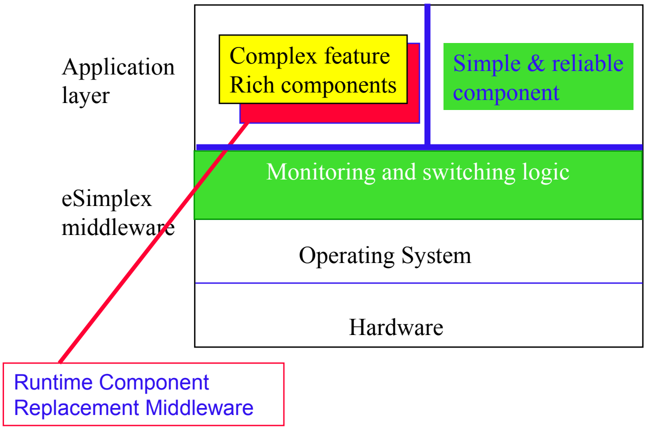
Online upgradeable complex controller

Data Flow Block Diagram

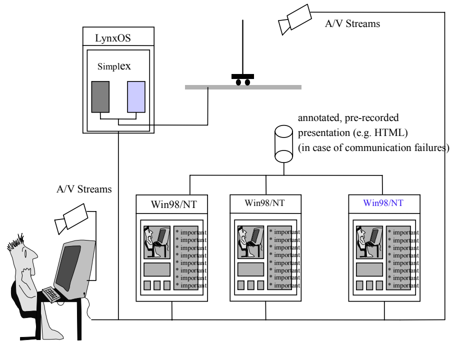
• Simplex architecture for control systems allows the online upgrade of control systems without shutting down the operation.

• It also maintains control in spite of arbitrary application errors in the upgrade process. To try an interactive demonstration, see www-drii.cs.uiuc.edu/download.

Dynamic Component Replacement


Application layer
Complex feature Rich components
Simple & reliable component
eSimplex middleware
Monitoring and switching logic
Operating System
Hardware
Runtime Component Replacement Middleware

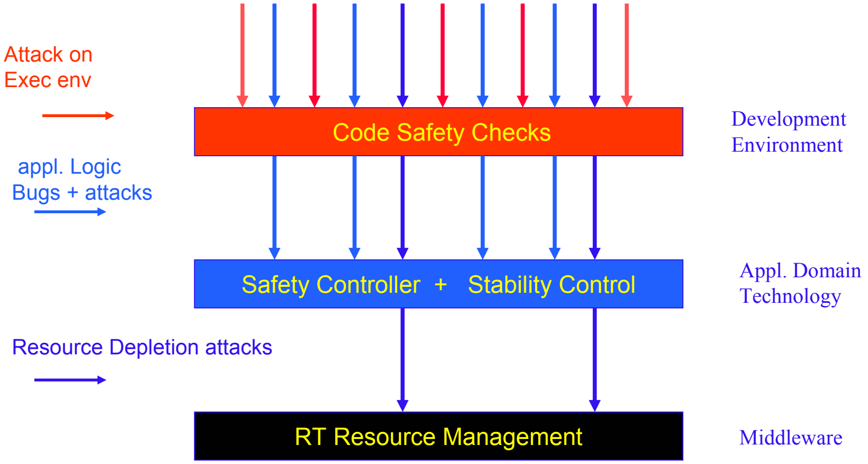
Interactive Demo on the Web



: Telelab Screen Shot

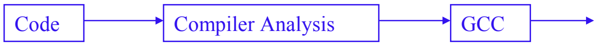
http://www-rtsl.cs.uiuc.edu/ click project, click drii, click telelab download

Protect Against Attacks Disguised as Upgrades

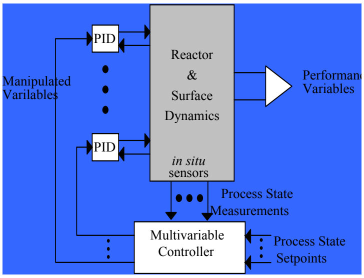


C Code Safety Checks

- Due to the large installed base of C, we working with colleagues to define a subset of C, called Control_C, that can be statically checked for safety and expressive enough for control and signal processing.
  - + { strong-typing }
  - + { Java-style pointers }
  - + { region-based heap *with only 1 region* }
  - + { "bounded" arrays }
  - – { system calls except memory allocation }
  - – {embedded assembly }

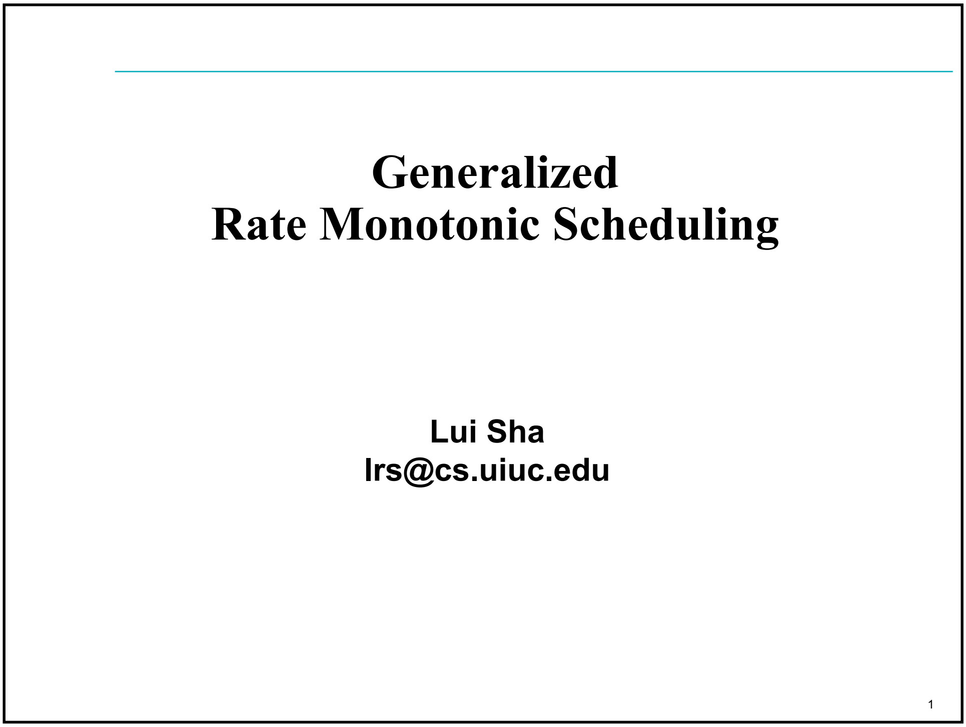


Ensure Code Safety without Runtime Checks for Real Time Control Systems, Kowshik, Dhurjati, & Adve, CASE 2002



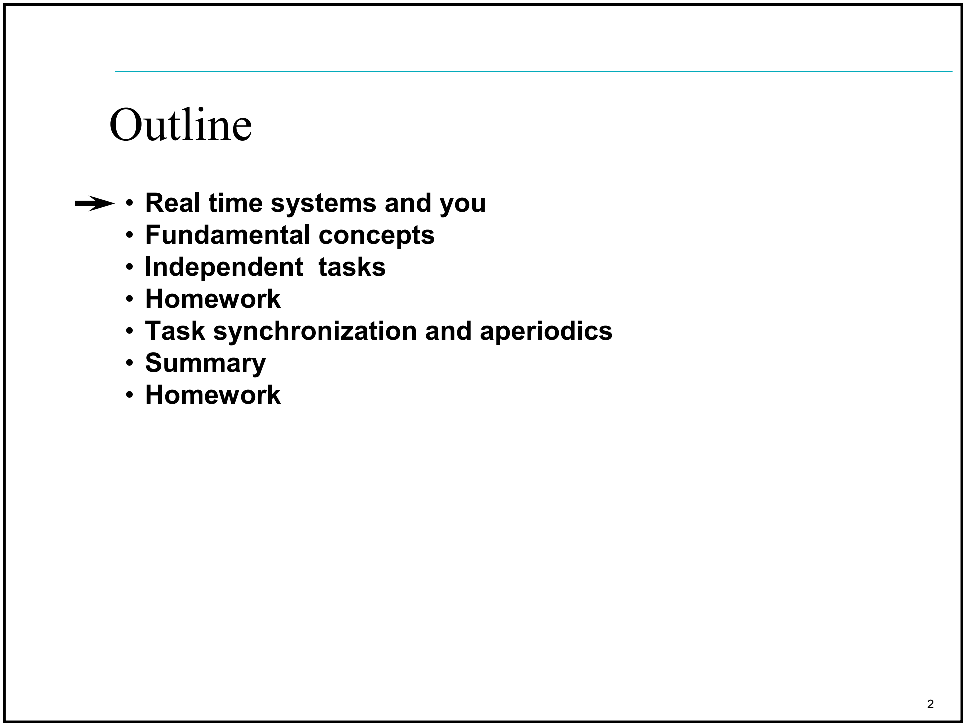
RT embedded systems have a long life span. We are developing technologies that allows a RT system that

- can be easily changed in the field, even on the fly
- maintain stability and controllability in spite of
  - arbitrary errors in the new software
  - malicious attack by insiders disguised as upgrades



Generalized
Rate Monotonic Scheduling

Lui Sha
lrs@cs.uiuc.edu



Outline

- ➔ **Real time systems and you**
- **Fundamental concepts**
- **Independent tasks**
- **Homework**
- **Task synchronization and aperiodics**
- **Summary**
- **Homework**

Prerequisite

The basic operating systems concepts including

- processes, threads and execution priorities
- context switching
- mutual exclusions and locks
- interrupt handling

Commonly used OS scheduling algorithms such as

- FIFO
- Round-robin
- Foreground/background

Real Time Systems and You

Embed real time systems enable us to:

- manage the vast power generation and distribution networks.
- control industrial processes for chemicals, fuel, medicine, and manufactured products.
- control automobiles, ships, trains and airplanes.
- conduct video conferencing over the Internet and interactive electronic commerce.
- send vehicles high into space and deep into the sea to explore new frontiers and to seek new knowledge.

Outline

- Real time systems and you
- ➔ Fundamental concepts
- Independent tasks
- Homework
- Task synchronization and aperiodics
- Summary
- Homework

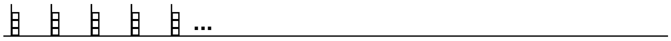
What is Real Time Systems

The correctness of real time computing depends upon not only the correctness of results but also meeting timing constraints:

- deterministically: (hard real time)
- statistically: (soft real time)

Periodic Tasks

A task τ_i is said to be periodic if its inter-arrival time (period), T_i, is a constant.



Periodic tasks are common in real time systems because the sampling actions.

(Can you give some examples?)

The utilization of task τ_i, is the ratio between its execution time C_i and its period T_i: $U_i = C_i / T_i$

The default deadline of a task is the end of period.

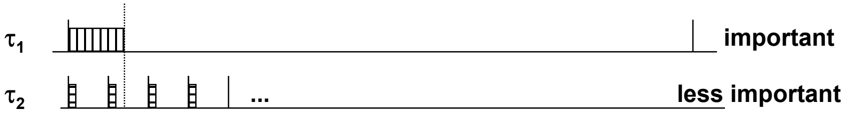
Importance and Priority

Task τ_1 : if it does not get done in time, the world will end.

Task τ_2: if it does not get done in time, you may miss a sweet dream.

Quiz: presume that the world is more important than your dream, should task τ_1 has a higher priority?

Why Ever Faster Hardware is Not Enough



If priorities are assigned according to importance, there is <u>no lower bound of processor utilization, below which tasks deadlines can be guaranteed.</u> Why?

$C_1/T_1 + C_2/T_2 = U$

$U \to 0$, when $C_2 \to 0$ and $T_1 \to \infty$

Task τ_2 will miss its deadline, as long as $C_1 > T_2$

Measure of Merits

	Time-Sharing Systems	Real-Time Systems
Capacity	High throughput	Schedulability
Responsiveness	Fast average response	Ensured worst-case response
Overload	Fairness	Stability

- schedulability is utilization level at or below which tasks can meet their deadlines
- stability in overload means the system meets critical deadlines even if all deadlines cannot be met (critical tasks are assumed to be schedulable.)

Dynamic vs “Static” Priorities

An instance of a task is called a job.

Dynamic priority scheduling adjust priorities in each task job by job.

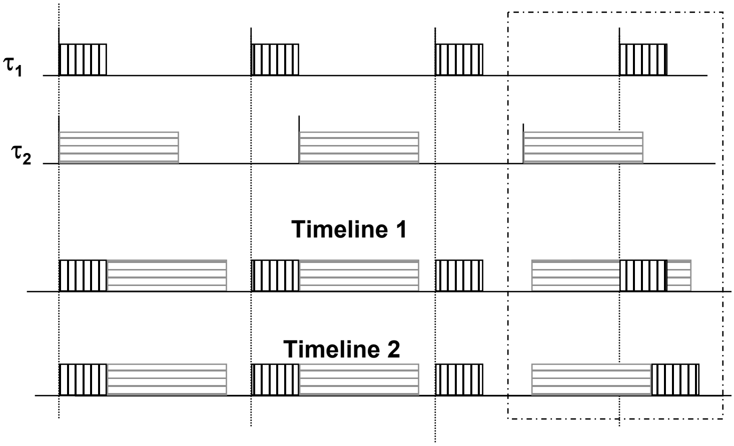
“Static” priority assigns a (base) priority to all the jobs in a task.

Deadline vs Rate Monotonic Scheduling

An optimal dynamic scheduling algorithm is the earlier deadline first (EDF) algorithm. Jobs closer to deadlines will have high priority.

An optimal “static” scheduling algorithm is the rate monotonic scheduling (RMS) algorithm. For a periodic task, the higher the rate (frequency), the higher the priority.

Which One Uses EDF (RMS)?



A Historical Note

For a given set of independent periodic tasks[Liu73],

- **earliest deadline first (EDF) can ensure all tasks' deadlines, if the processor utilization is not greater 1.0.**
- **rate monotonic algorithm can ensure all the tasks' deadlines if processor utilization is not greater than 0.69.**

Since the early 90's, RMS was generalized into GRMS and caught on, but EDF is still used infrequently.

An Open Problem

Under EDF, if a processor has a transient overload, it is not clear which task can ensure its the deadline, since each job of a task can have a different priority.

This problem is solvable. So far, no efficient algorithm has been found to make it worthwhile to implement for majority of the applications. On the other hand,

- RMS has a simple solution to the stability problem.
- The 0.69 worst case number is rarely seen in practice. When encountered, it can be engineered away.
- Processor cycles, which cannot be used by real time tasks under RMS, can be used by non-real time tasks with low background priority.

GRMS in The Real World

“The navigation payload software for the next block of Global Positioning System upgrade recently completed testing. ... This design would have been difficult or impossible prior to the development of rate monotonic theory", Doyle, L., and Elzey, J., , Technical Report, ITT, Aerospace & Communication Division, 1993, p. 1.

"A major payoff...System designers can use this theory to predict whether task deadlines will be met long before the costly implementation phase of a project begins. It also eases the process of making modifications to application software."

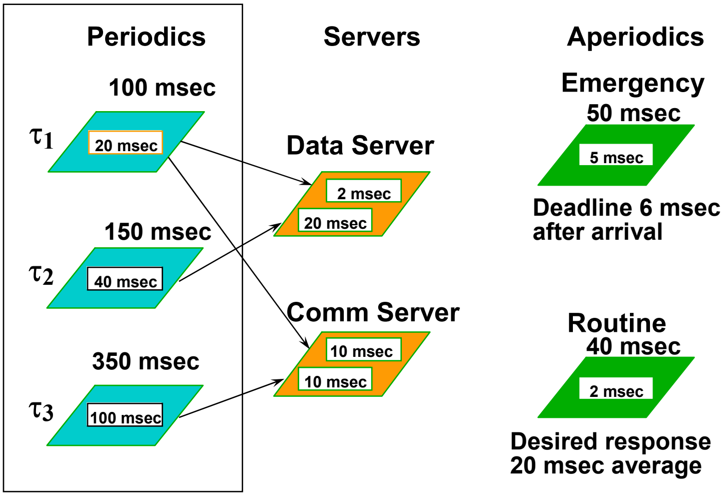
DoD *1991 Software Technology Strategy*. pp. 8-15.

Outline

Class 1:
- Real time systems and you
- Fundamental concepts
- ➔ An Introduction to the GRMS: independent tasks
- Homework

Class 2:
- An introduction to the GRMS: task synchronization and aperiodics
- Summary
- Homework

A Sample Problem



Schedulability: UB Test

Utilization bound(UB) test: a set of n independent periodic tasks scheduled by the rate monotonic algorithm will always meet its deadlines, for all task phasing, if

$$\frac{C_1}{T_1} + ... + \frac{C_n}{T_n} \le U_{(n)} = n(2^{1/n} - 1)$$

U(1) = 1.0	U(4) = 0.756	U(7) = 0.728
U(2) = 0.828	U(5) = 0.743	U(8) = 0.724
U(3) = 0.779	U(6) = 0.734	U(9) = 0.720

For harmonic task sets, the utilization bound is U(n)=1.00 for all n. For large n, the bound converges to *In 2* ~ 0.69.

Conventions, task 1 has shorter period than task 2 and so on.

Sample Problem: Applying UB Test

	C	T	U
Task $\tau 1$:	20	100	0.200
Task $\tau 2$:	40	150	0.267
Task $\tau 3$:	100	350	0.286

Total utilization is .200 + .267 + .286 = .753 < U(3) = .779

The periodic tasks in the sample problem are chedulable according to the UB test.

Toward a More Precise Test

UB test has three possible outcomes:

- **$0 \le U \le U(n)$ ==> *Success***
- **$U(n) < U \le 1.00$ ==> *Inconclusive***
- **$1.00 < U$ ==> *Overload***

UB test is conservative.

Example: Applying Exact Test -1

Taking the sample problem, we increase the compute time of τ_1 from 20 to 40; is the task set still schedulable?

Utilization of first two tasks: $0.667 < U(2) = 0.828$

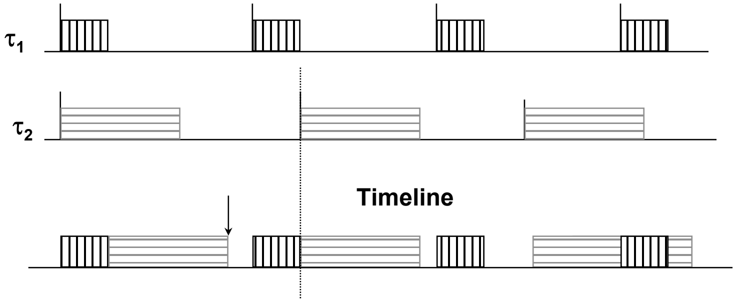
- **first two tasks are schedulable by UB test**

Utilization of all three tasks: $0.953 > U(3) = 0.779$

- **UB test is inconclusive**
- **need to apply exact test**

The Exact Schedulability Test

If a task meets its first deadline when all higher priority tasks are started at the same time, then all this task's future deadlines will always be met[Liu73]. The exact test for a task checks if this task can meet its first deadline.



Schedulability: Exact Test

Intuition: let t = a_0 be the instance at which task τ_i and all higher priority task execute once.

If there is no new arrival from higher priority tasks during a_0, τ_i actually completes its execution at t = a_0 . If there is new arrivals, the compute a_1 and check if there is new arrivals…

The arrivals are counted by the ceiling function.

$$a_{n+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_n}{T_j} \right\rceil C_j \qquad \text{where } a_0 = \sum_{j=1}^{i} C_j$$

Test terminates when $a_{n+1} > T_i$ (not schedulable)
or when $a_{n+1} = a_n \leq T_i$ (schedulable).

Example: Applying Exact Test -2

Use exact test to determine if τ_3 meets its first deadline:

$$a_0 = \sum_{j=1}^{3} C_j = C_1 + C_2 + C_3 = 40 + 40 + 100 = 180$$

$$a_1 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_0}{T_j} \right\rceil C_j$$

$$= 100 + \left\lceil \frac{180}{100} \right\rceil (40) + \left\lceil \frac{180}{150} \right\rceil (40) = 100 + 80 + 80 = 260$$

Example: Applying the Exact Test -3

$$a_2 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_1}{T_j} \right\rceil C_j = 100 + \left\lceil \frac{260}{100} \right\rceil (40) + \left\lceil \frac{260}{150} \right\rceil (40) = 300$$

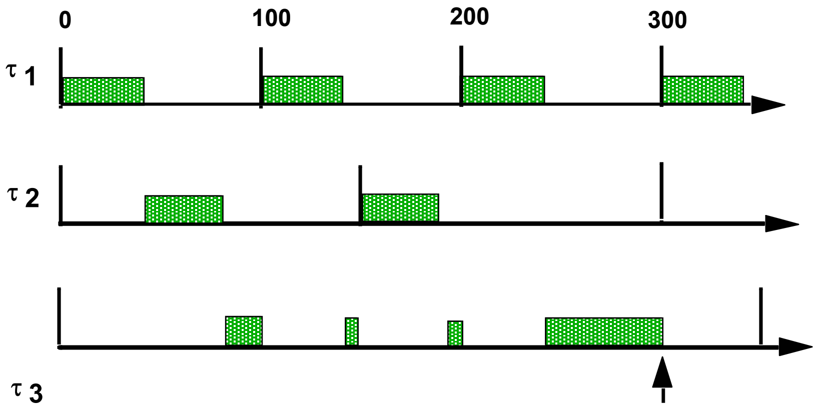
$$a_3 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_2}{T_j} \right\rceil C_j = 100 + \left\lceil \frac{300}{100} \right\rceil (40) + \left\lceil \frac{300}{150} \right\rceil (40) = 300$$

$a_3 = a_2 = 300$ **Done!**

Task τ_3 is schedulable using exact test

$$a_3 = 300 < T = 350$$

Timeline

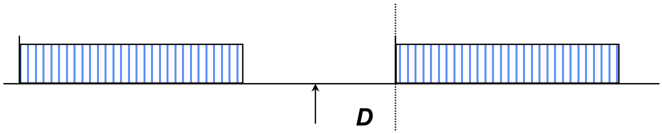


τ 3 completes its work at t = 300

Pre-period Deadline

Note that task τ_3 default deadline is at 350, but its worst case finishing time is 300. Thus, its deadline can be moved earlier by 50 unit before its end of period.

Under GRMS, addressing pre-period deadline is simple, just replace a task deadline from *T to (T - D)* in the exact schedulability analysis.



Stability Under Transient Overload

Rate monotonic scheduling requires assigning task priorities according to periods (rates).

Question: “How does one ensure the deadline of a critical task with a long period, resulting in a low priority.

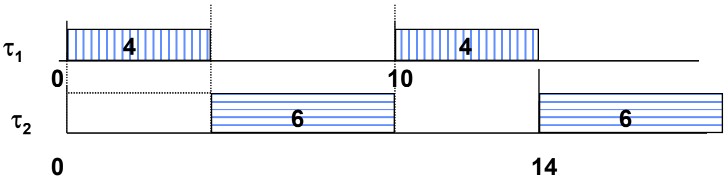
Solution: Period Transformation.

For example, give the task a T/2 period, which increases its priority for RMS, but suspend the task after C/2 worst case execution.

After all, importance and rate monotone priority assignment can be made consistent.

(But don’t buy a knife and slice up the program... It can done invisibly to the program … Stay tuned.)

When Schedulability is low



Home work: task τ_1 has execution time 4 and period 10, while task τ_2 has execution 6 and period 14. Deadline of task τ_2 will be missed if we increase execution time of task τ_2 from 6 to 8.

How can we ensure both tasks’ deadlines without reducing task execution time? (Hint: period transformation.)

Context Switching Overhead

Period transformation is not a free lunch, it increases context switching overhead.

Context switching cost comes in pairs, preemption and resuming.

You need to add the context switching overhead cost, 2S, into the execution of each tasks for more precise schedulability analysis.

The context switching overhead of task τ_i is ($2S / T_i$). The total system context switching overhead is thus the sum of tasks' context overheads.

The impact of context switching time in an OS is inversely related to the periods of application tasks.

Homework

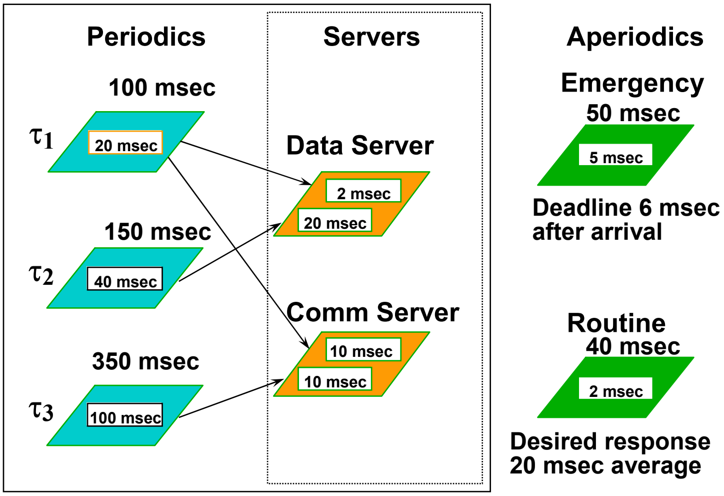
1) Write a simple program to compute schedulability (Hint: to save time, you may want to use a spread sheet program).

2) Change the numbers and tasks in the example and apply the formula.

Outline

- Real time systems and you
- Fundamental concepts
- Independent tasks
- Homework
- ➔ Task synchronization and aperiodics
- Summary
- Homework

A Sample Problem



Periodics

100 msec

τ_1 20 msec

150 msec

τ_2 40 msec

350 msec

τ_3 100 msec

Servers

Data Server

2 msec

20 msec

Comm Server

10 msec

10 msec

Aperiodics

Emergency
50 msec

5 msec

Deadline 6 msec
after arrival

Routine
40 msec

2 msec

Desired response
20 msec average

Priority Inversion

Ideally, under prioritized preemptive scheduling, higher priority tasks should immediately preempt lower priority tasks.

When lower priority tasks causing higher priority tasks to wait due to the locking of shared data, priority inversion is said to occur.

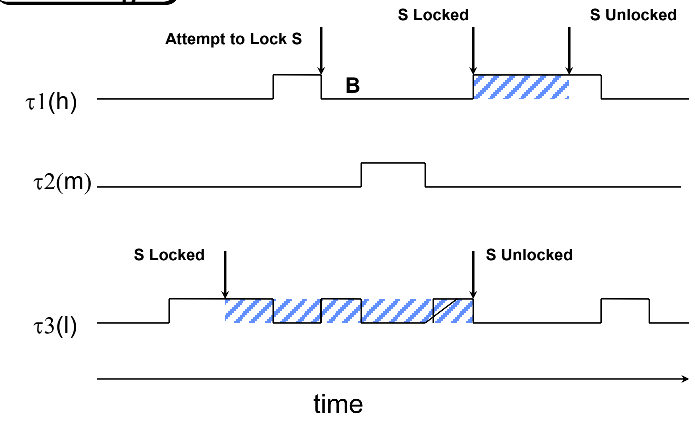
It seems reasonable to expected the duration of priority inversion (also called blocking time), should be a function of the duration of the critical sections.

Critical section: the duration of a task using shared resource.

Unbounded Priority Inversion

Legend
S Locked ▨
Executing □
Blocked B

τ1:{...P(S)...V(S)...}
τ3:{...P(S)...V(S)...}



Basic Priority Inheritance Protocol

Let the lower priority task to use the priority of the blocked higher priority tasks.

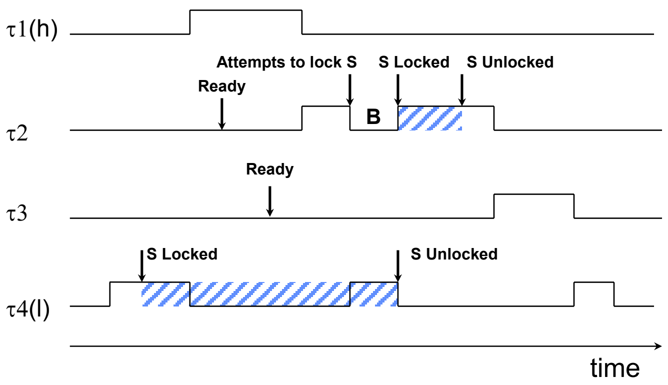
In this way, the medium priority tasks can no longer preempted to low priority task that blocks the high priority tasks.

Priority inheritance is transitive.

Basic Priority Inheritance Protocol

Legend ▨
S Locked □
Executing B
Blocked

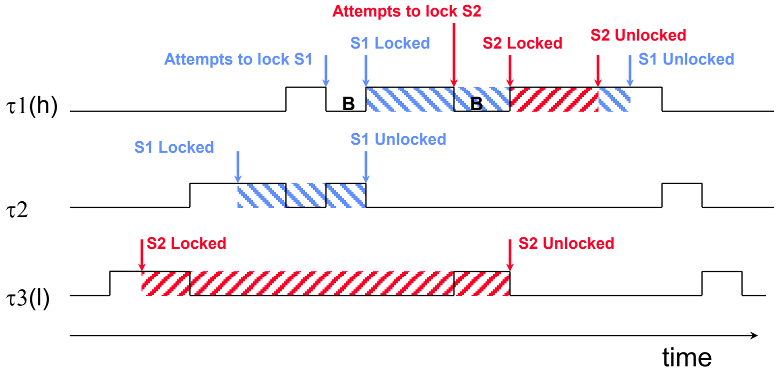
τ2:{...P(S)...V(S)...}
τ4:{...P(S)...V(S)...}



Chained Blocking

Legend
S1 Locked ▨
S2 Locked ▨
Executing ⊓
Blocked B

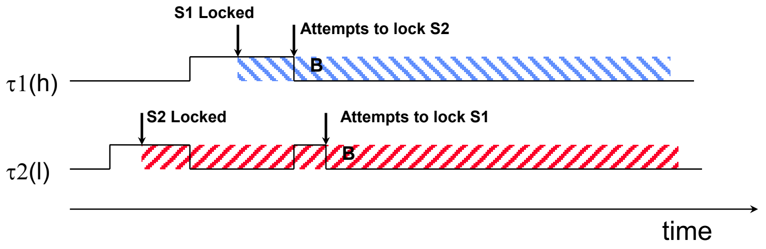
τ1:{…P(S1)…P(S2)…V(S2)…V(S1)…}
τ2:{…P(S1)…V(S1)…}
τ3:{…P(S2)…V(S2)…}



Deadlock Under BIP

Legend
S1 Locked ▨
S2 Locked ▨
Executing ⊓
Blocked B

τ1:{…P(S1)…P(S2)…V(S2)…V(S1)…}
τ2:{…P(S2)…P(S1)…V(S1)…V(S2)…}



Property of Basic Priority Inheritance

OS developers can support it without knowing application priorities.

There will be no deadlock if there is no nested locks, or application level deadlock avoidance scheme such the ordering of resource is used.

Chained priority is fact of life. But a task is blocked at most by n lower priority tasks sharing resources with it, when there is no deadlock.

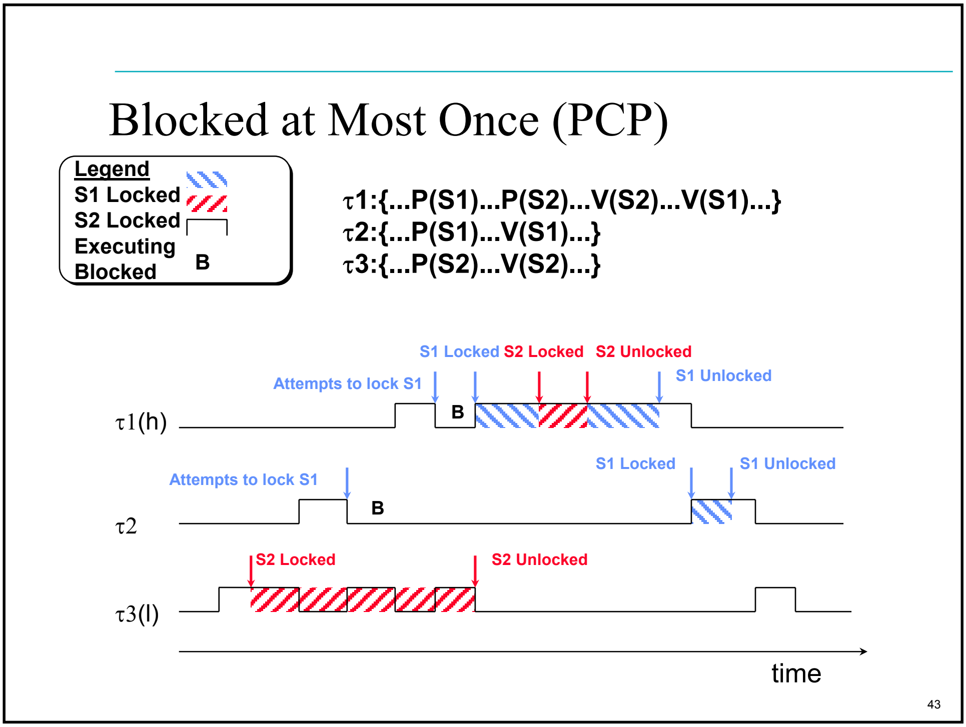
Priority inheritance protocol is supported by almost all of the real time OS and is part of POSIX real time extension.

Priority Ceiling Protocol

A priority ceiling is assigned to each semaphore, which is equal to the highest priority task that may use this semaphore.

A task can lock a semaphore if and only if its priority is higher than the priority ceilings of all locked semaphores.

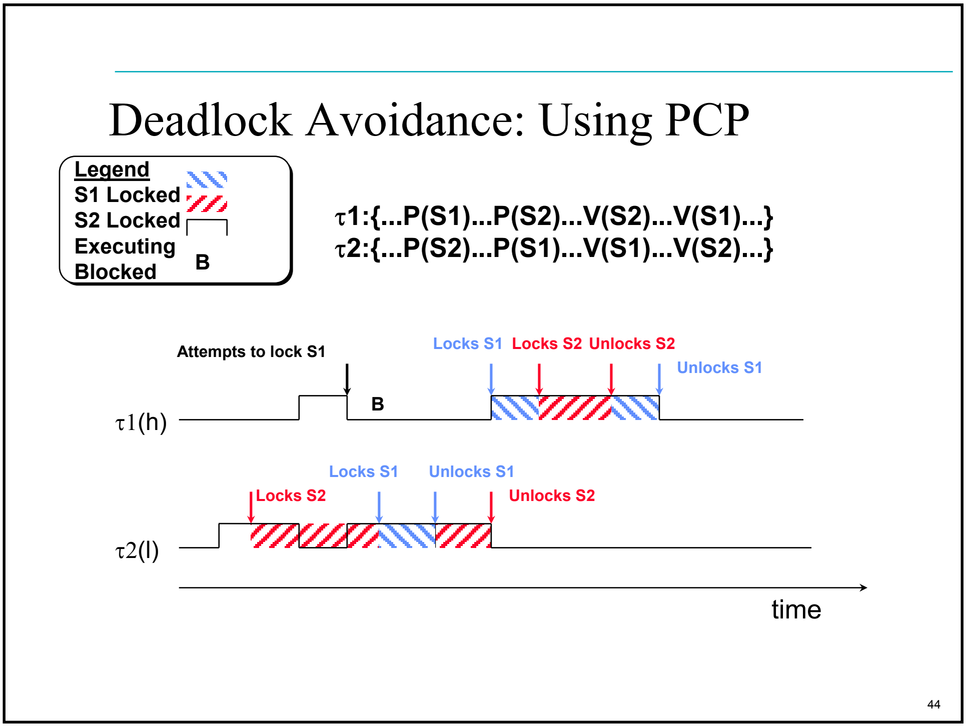
If a task is blocked by lower priority tasks, the lower priority task inherits its priority.



Blocked at Most Once (PCP)

Legend
S1 Locked ▨
S2 Locked ▨
Executing ⊓
Blocked B

τ1:{...P(S1)...P(S2)...V(S2)...V(S1)...}
τ2:{...P(S1)...V(S1)...}
τ3:{...P(S2)...V(S2)...}



Deadlock Avoidance: Using PCP

Legend
S1 Locked ▨
S2 Locked ▨
Executing ⊓
Blocked B

τ1:{...P(S1)...P(S2)...V(S2)...V(S1)...}
τ2:{...P(S2)...P(S1)...V(S1)...V(S2)...}

Schedulability Analysis

A uni-processor equation using BIP

preemption execution blocking

$$\forall i, 1 \le i \le n, \sum_{j=1}^{i-1} \frac{C_j}{T_j} + \frac{C_i + (B_{i+1} + \dots B_n)}{T_i} \le i(2^{1/i} - 1)$$

A uni-processor equation using PCP

preemption execution blocking

$$\forall i, 1 \le i \le n, \sum_{j=1}^{i-1} \frac{C_j}{T_j} + \frac{C_i + \max(B_{i+1} \dots B_n)}{T_i} \le i(2^{1/i} - 1)$$

Sample Problem: Using BIP

	C	T	D	B
τ1	20	100		30
τ2	40	150	20	10
τ3	100	350		

$$W_i(k+1) = B_i + C_i + \sum_{j=1}^{i-1} \left\lceil \frac{W_i(k)}{T_j} \right\rceil C_j$$

Schedulability Model Using BIP

$$\frac{C_1}{T_1} + \frac{B_1}{T_1} \le U(1) \qquad \frac{20}{100} + \frac{30}{100} = 0.50 < 1.0$$

$$\frac{C_1}{T_1} + \frac{C_2 + D_2}{T_2} + \frac{B_2}{T_2} \le U(2) \qquad \frac{20}{100} + \frac{40+20}{150} + \frac{10}{150} = 0.667 < 0.828$$

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} \le U(3) \qquad \frac{20}{100} + \frac{40}{150} + \frac{100}{350} = 0.753 < 0.779$$

Modeling Interrupts

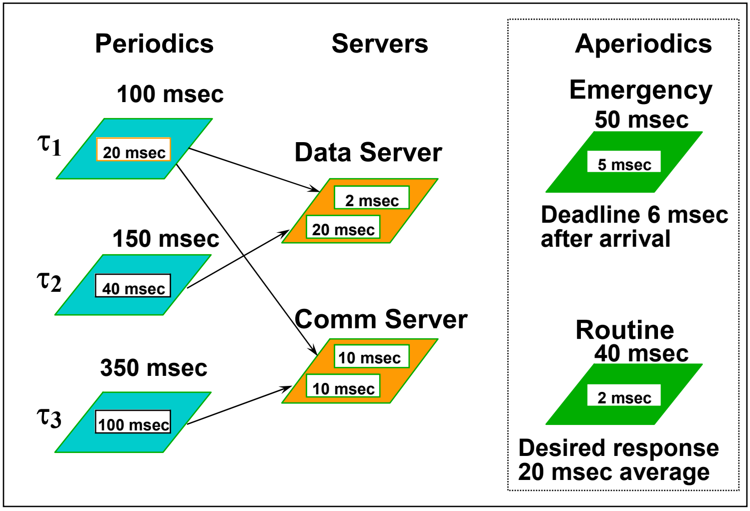
A hardware interrupt can have higher priority than software.

When an interrupt service routine, *R*, is used to capture data for longer period task, it will still preempt the execution of shorter period tasks.

From the perspective of GRMS, the time spent in *R* is a form of priority inversion. Thus, we can add *R* into the blocking time from an analysis perspective.

Quiz: If *R* is long, what should we do in software?

A Sample Problem



Periodics

Servers

Aperiodics

100 msec

τ1

20 msec

150 msec

τ2

40 msec

350 msec

τ3

100 msec

Data Server

2 msec

20 msec

Comm Server

10 msec

10 msec

Emergency
50 msec

5 msec

Deadline 6 msec
after arrival

Routine
40 msec

2 msec

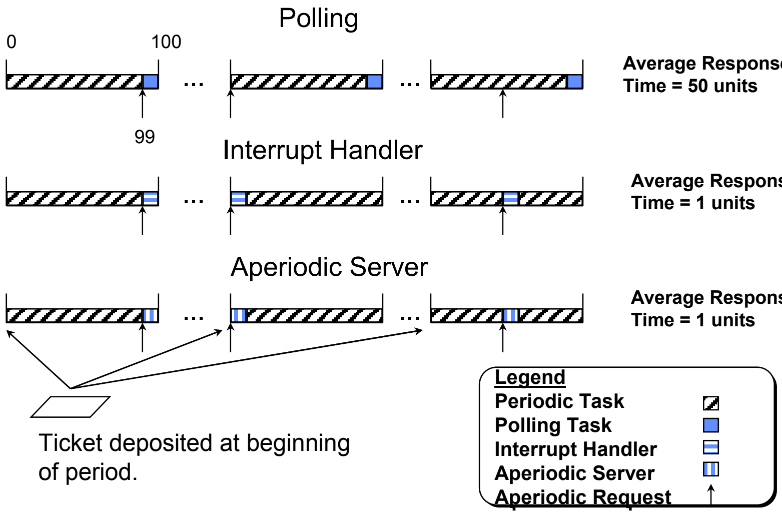
Desired response
20 msec average

Concepts and Definitions

Aperiodic task: runs at irregular intervals.

Aperiodic deadline:
hard, minimum interarrival time
soft, best average response

Scheduling Aperiodic Tasks



Average Response Time = 50 units

Average Response Time = 1 units

Average Response Time = 1 units

Ticket deposited at beginning of period.

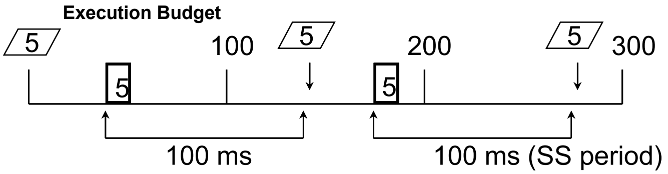
Legend
- **Periodic Task** ▨
- **Polling Task** 🟦
- **Interrupt Handler** ⊟
- **Aperiodic Server** ◫
- **Aperiodic Request** ↑

Sporadic Server (SS)

Modeled as periodic tasks
 Fixed execution budget (C)
 Replenishment interval (T)

Priority is based on T, adjusted to meet requirements

Replenishment occurs one “period” after start of use.



Sample Problems: Aperiodic

Emergency Server (ES)

- Execution Budget, C = 5
- Replenish Interval, T= 50

General Aperiodic Server (GS) Design guideline: Give it as high a priority as possible and as much “tickets” as possible, without causing periodic tasks missing deadlines:

- Execution Budget, C = 10
- Replenish Interval, T = 100

Simulation and queuing theory using M/M1 approximation indicates that the average response time is 2 msec (See Real Time Scheduling Theory and Ada).

Implementing Period Transformation

Recall that period transformation is a useful techniques to ensure:

- stability under transient overload
- improve system schedulability

But it is undesirable to slice up the program codes. (Thou shalt separate timing concerns with functional concerns.)

For example, a task with period T and exception time C, can be transformed as a sporadic task with a budget C/2 and periodic T/2. This is transparent to the applications.

Homework

Try to apply GRMS to your lab work, if you are working a real time computing project.

Summary

We have reviewed

- **the basic concepts of real time computing**
- **the basics of GRMS theory**
  - **Independent tasks**
  - **synchronization**
  - **aperiodic tasks**

"*Through the development of [Generalized] Rate Monotonic Scheduling, we now have a system that will allow [Space Station] Freedom's computers to budget their time, to choose between a variety of tasks, and decide not only which one to do first but how much time to spend in the process*",

--- Aaron Cohen, former deputy administrator of NASA, "Charting The Future: Challenges and Promises Ahead of Space Exploration", October, 28, 1992, p. 3.

Additional Results

In networks, distributed scheduling decision must be made with incomplete information and yet the distributed decisions are coherence - lossless communication of scheduling messages, distributed queue consistency, bounded priority inversion, and preemption control.

From a software engineering perspective, software structures dealing with timing must be separated with construct dealing with functionality.

To deal with re-engineering, real time scheduling abstraction layers (wrapper) are needed so that old software packages and network hardware behavior as if they are designed to support GRMS.

References

Liu, C. and Layland, J., “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,” Journal of the ACM, Vol. 20, No.1, January 1973, pp.46-61. [Classic]

Sha, L. and Goodenough, J., “Real-Time Scheduling Theory and Ada,” Computer, Vol. 23, No.4, April 1990, pp. 53-62. [uni-processors GRMS Tutorial]

M. Klein et al., A Practitioner’s Handbook for Real-Time Analysis: Guide to Rate-Monotonic Analysis for Real-Time Systems, Kluwer Academic Publishers, Boston, July, 1993.

Sha, L., Rajkumar, R., and Sathaye, S., “Generalized Rate Monotonic Scheduling Theory: A Framework of Developing Real-Time Systems”, Proceedings of The IEEE, January, 1994 [Distributed GRMS tutorial].

"It is difficult to say what is impossible, for the dream of yesterday is the hope of today and the reality of tomorrow."

Robert H. Goddard

Elements of Research

Lui Sha

CS, UIUC

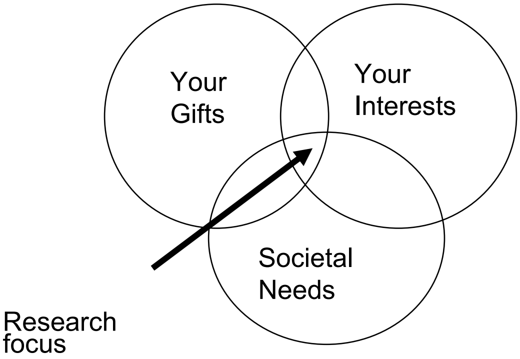
The entrance to graduate school marks a *critical phase of transition* for most graduate students from absorbing knowledge to creating knowledge …

- To excel in research, we must sharpen our skills in
  - positioning R&D strategically
  - identifying and formulating high impact problems
  - communicating ideas and results effectively

Three Basic Elements of Successful R&D

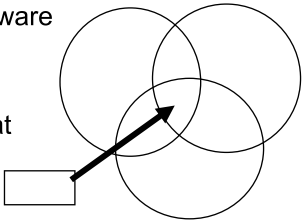
- *Positioning your research*
- *Developing a R&D Roadmap*
- *Getting your ideas across*

> *The path to success consists of three simple elements. Find what interests you that you can do well, and is needed by the people.* *--- Lui Sha*



Understand others is intelligence.
Understand yourself is wisdom.
--- Lao Tze

What is easier for you, writing a complex software program or proving a difficult theorem?



What excites the community at large and what excites you?

Does it play into your strength?

Elements of Successful R&D

- *Positioning your research*
- *Developing a R&D road map*
- *Getting your ideas across*

Creating an Exciting Application Scenario

> *“as a mathematical discipline travels far from its empirical source, or still more, if it is a second and third generation only indirectly inspired by the ideas coming from ‘reality’, it is beset with very grave dangers.*
>
> *… that the stream, so far from its source, will separate into a multitude of insignificant branches, and that the discipline will become a disorganized mass of details and complexities."*
>
> John Von Neumann, "The Mathematician" , 1957

Exciting application scenarios will

- motivate you,
- expose the limitations of existing solutions,
- help you to focus your efforts.

> *Great advancements in science and engineering often are the repudiation of generally accepted beliefs.*
>
> *Anonymous*

Most researches are constrained by models and generally accepted assumptions of the real world. But our knowledge of the nature is never perfect, and the underlining technologies are rapidly changing…

- Velocity of light is constant. … *embrace it as a law of physics and we have the theory of relativity.*
- Clients request and server computes *… Why not send some of the code to client instead? … and we have JAVA & mobile code.*

- *Is TCP appropriate for wireless communication?*
- *Is fairness a good metric for real time computing?*
- *Is load balance is always a good idea?*

Pick the Right Problems to Work on

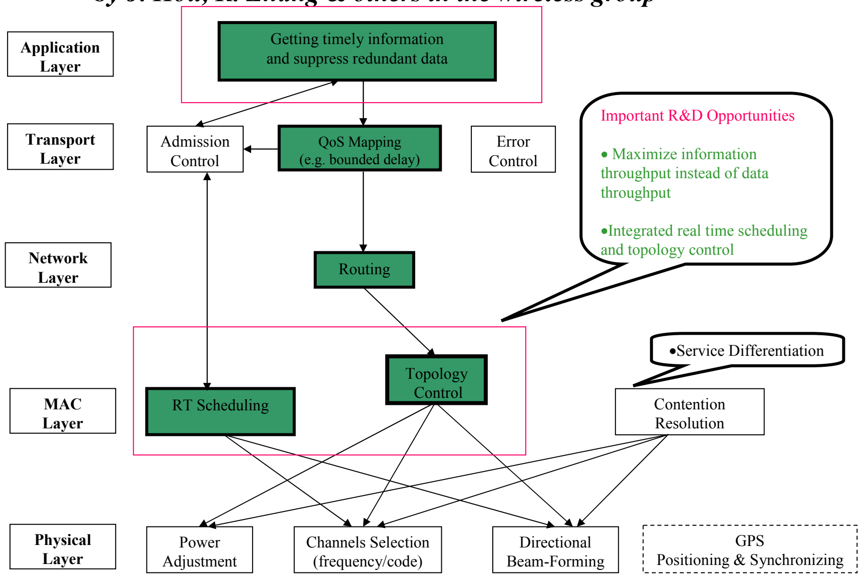
- What is the difference between a theorem and a homework problem to be proven by students?
  - Both were proven to be correct.
  - In fact, some homework problems are harder than some of the theorems.
- If we decide to spend time on a problem, shouldn't we work on a problem with greater potential impacts?

Know what has been Done and Estimate the Impacts

- New directions
  - challenging long-held beliefs and pioneering a new path
- Broad Applicability
  - for the further development of the theory
  - for solving practical problems
- Unification / Integration
  - Proving a unifying structure or theory and give deeper understanding to seemingly diversified approaches
- Advancement along an established line of inquiries
  - You need to significantly improve performance, reliability, or scale

A Recent Example of a R&D Roadmap

by J. Hou, R. Zhang & others in the wireless group



Application Layer

Getting timely information and suppress redundant data

Transport Layer

Admission Control

QoS Mapping (e.g. bounded delay)

Error Control

Important R&D Opportunities

- Maximize information throughput instead of data throughput
- Integrated real time scheduling and topology control

Network Layer

Routing

MAC Layer

RT Scheduling

Topology Control

•Service Differentiation

Contention Resolution

Physical Layer

Power Adjustment

Channels Selection (frequency/code)

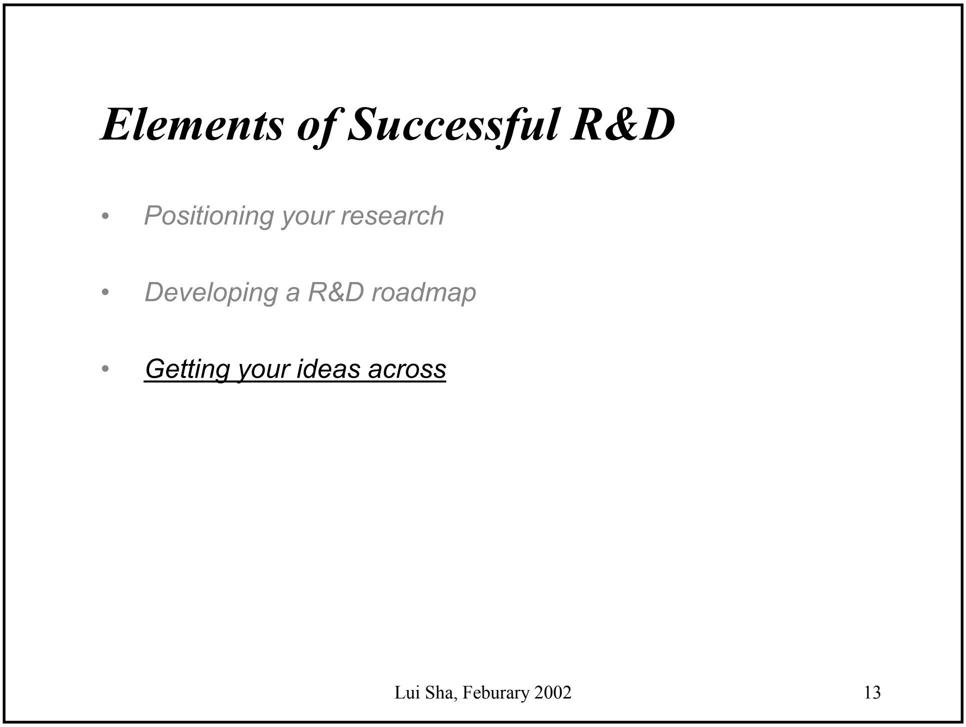
Directional Beam-Forming

GPS Positioning & Synchronizing

Learn what makes classics classic

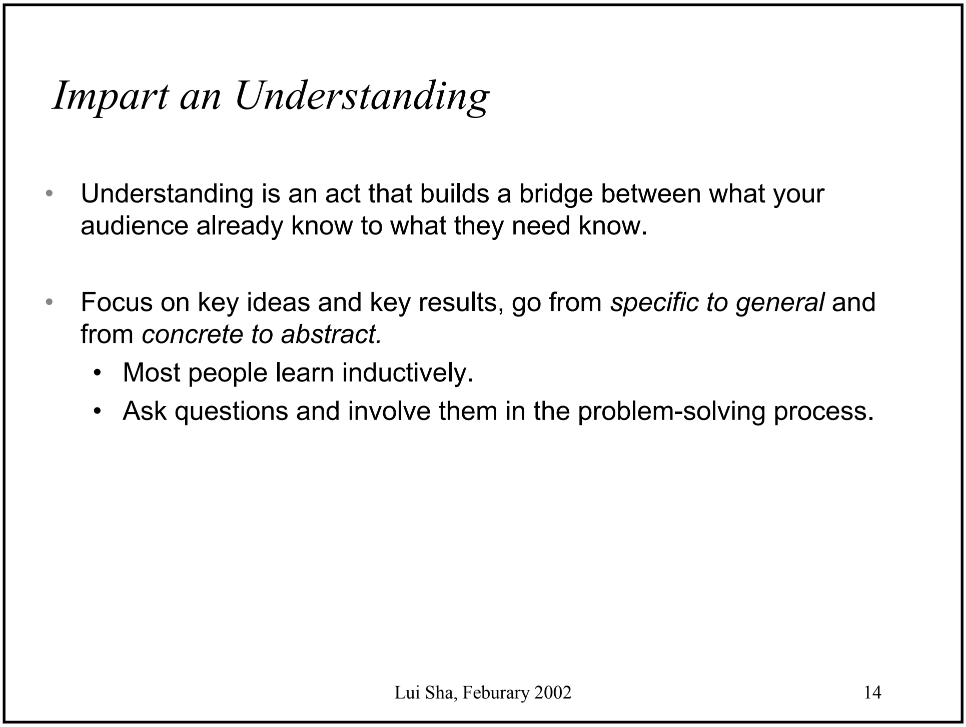
- Learning what makes a classic papers classic
  - The state of research prior to the classic paper
  - The impacts made by the classic paper
  - Compare the classic paper with good papers

- Examples
  - On the Criteria To Be Used in Decomposing Systems into Modules. *David L. Parnas CACM*, Vol. 15, No. 12, December 1972.
  - *Public key cryptography*, R. L. Rivest, A. Shamir and L. Adleman, CACM (21)120--126, 1979. …



Elements of Successful R&D

- *Positioning your research*
- *Developing a R&D roadmap*
- *Getting your ideas across*



Impart an Understanding

- Understanding is an act that builds a bridge between what your audience already know to what they need know.
- Focus on key ideas and key results, go from *specific to general* and from *concrete to abstract.*
  - Most people learn inductively.
  - Ask questions and involve them in the problem-solving process.

Managing Human "cache memory"

- Human short term memory can only hold about 5 *unfamiliar* items
  - Don't load it up with unimportant details
- Suppose you need to present an OS overhead formula unfamiliar to your audience, *(2S + …)*.
  - Don't say we now add "two *S*" to …. This forces others to remember what *S* means. Poor use of human short term memory.
  - Say we add "round trip context switching time to …
- Think carefully about the new ideas you want your audience to absorb.
- Keep them in the "cache" by periodic refreshing during your talk, until your audience "write the new ideas through" into their long term memory.

Techniques that reduce "unfamiliarity" and help "write through".

- *Read out the physical meaning of the terms* .
- Use analogy familiar to your audience

Getting Your Ideas Across: Sha's I^3 Model

An ideal presentation is one that is

- *informative,*
- *interesting and*
- *Insightful*

Being Informative

- Inform: give *new* knowledge...
- "New" is relatively to your audience.
  - what they already know?
  - what they should know after your presentation?
  - what are the steps in-between?

- For example:
  - Managers: the key ideas, expected impacts, and costs
  - Experts: new challenges and new insights/results

Being Interesting

- Interesting: unexpected, counter intuitive, difficult to believe
  - Seemingly unimportant fact that actually holds the key
  - Seemingly true but it is in fact false…
  - A "difficult" problem is solved with ease and elegance.
  - …

Being Insightful

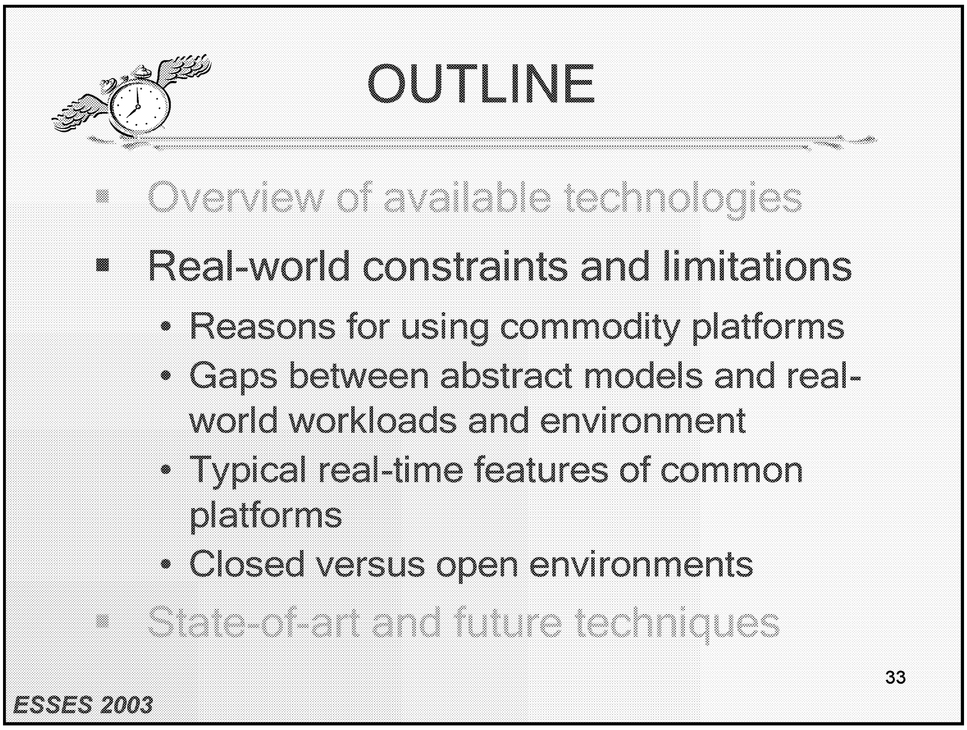
- Insight: impart a deeper understanding…
  - Explain a seemingly complex and confusing problem in a way that is easy to understand.
  - Unearth hidden/unstated assumptions... And quickly put an argument to rest.
  - Show things in new angles, new lights and new forms and gain new understandings.
  - Demonstrate subtle but important connections/inter-dependencies between seemingly unrelated subjects.
  - …

The entrance to graduate school marks a *critical phase of transition* for most graduate students from absorbing knowledge to creating knowledge …

- To excel in research, we must sharpen our skills in
  - positioning R&D strategically
  - identifying and formulating high impact problems
  - communicating ideas and results effectively

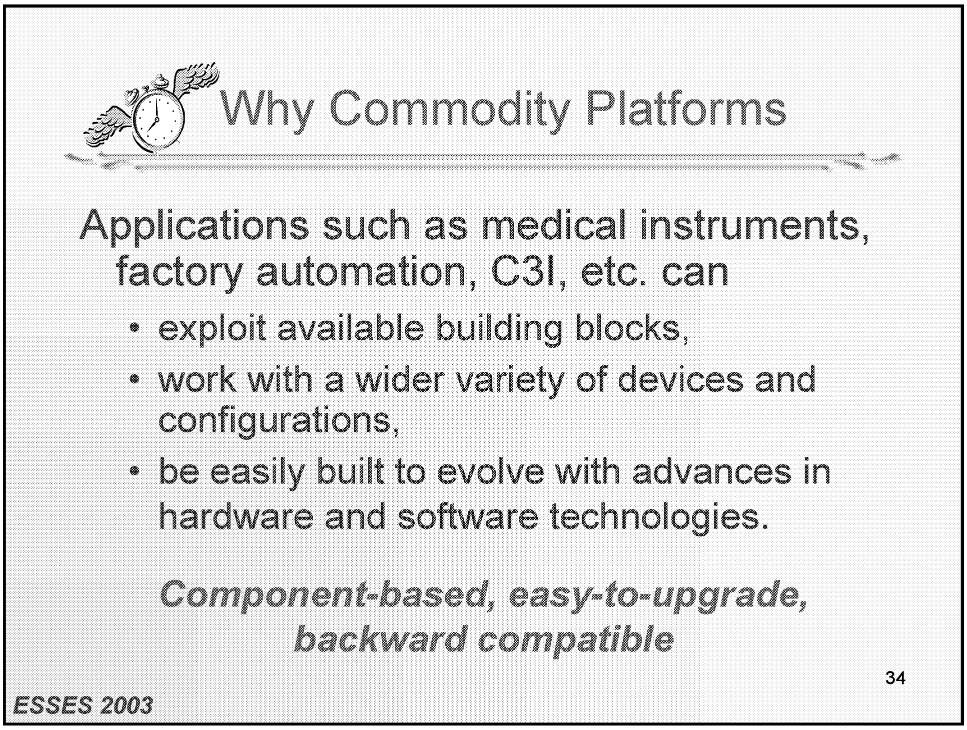
Jane W. S. Liu

janeliu@microsoft.com
Windows OS Core
Microsoft Corporation



OUTLINE

- Overview of available technologies
- Real-world constraints and limitations
  - Reasons for using commodity platforms
  - Gaps between abstract models and real-world workloads and environment
  - Typical real-time features of common platforms
  - Closed versus open environments
- State-of-art and future techniques



Why Commodity Platforms

Applications such as medical instruments, factory automation, C3I, etc. can

- exploit available building blocks,
- work with a wider variety of devices and configurations,
- be easily built to evolve with advances in hardware and software technologies.

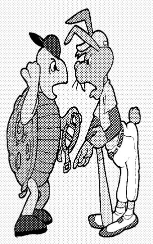
Component-based, easy-to-upgrade, backward compatible

- Most desired attribute of RT OS:

  Predictability ==
  Bounded worst-case response times

- General OS CW:
  - Optimize for average response time and throughput
  - Be greedy and conserve work
  - Allocate resources fairly
  - The faster, the better
  - The more, the better

  Lead to unpredictability !

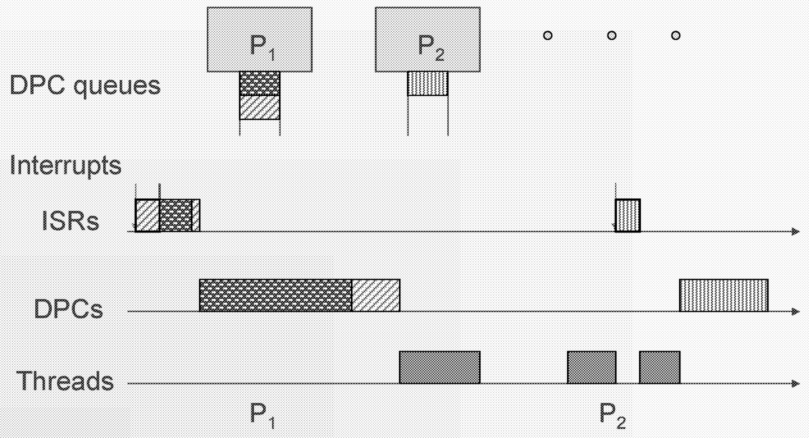


Examples of Shortcomings

- Real-time qualities
- Time granularity
- Multiprocessor scheduling
- Scheduling and synchronization
- Conflicts among components

Questions: Why they are so?
Are they likely to be corrected?
How to live with them?

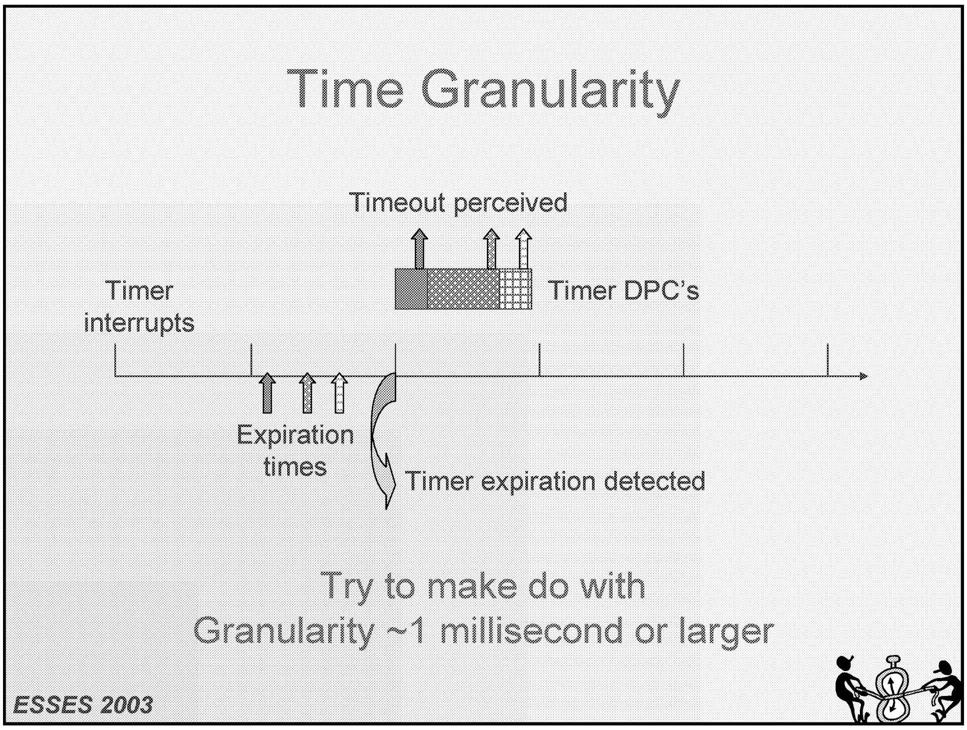
Deferred Procedure Calls
(A Cause of Unbounded Blocking)



Tasklets & bottom-halves are similar

Real-Time Quality Assurance

- Watchdogs for ensuring that
  - Processors are alive
  - DPCs execute within timeout interval
  - No DPC executes too long
  - No run of DPCs executes too long
- Tools for capturing and logging
  - Lateness of time-critical executions
  - Histograms of ISR, processor stall, DPC times, etc.



Time Granularity

Timeout perceived

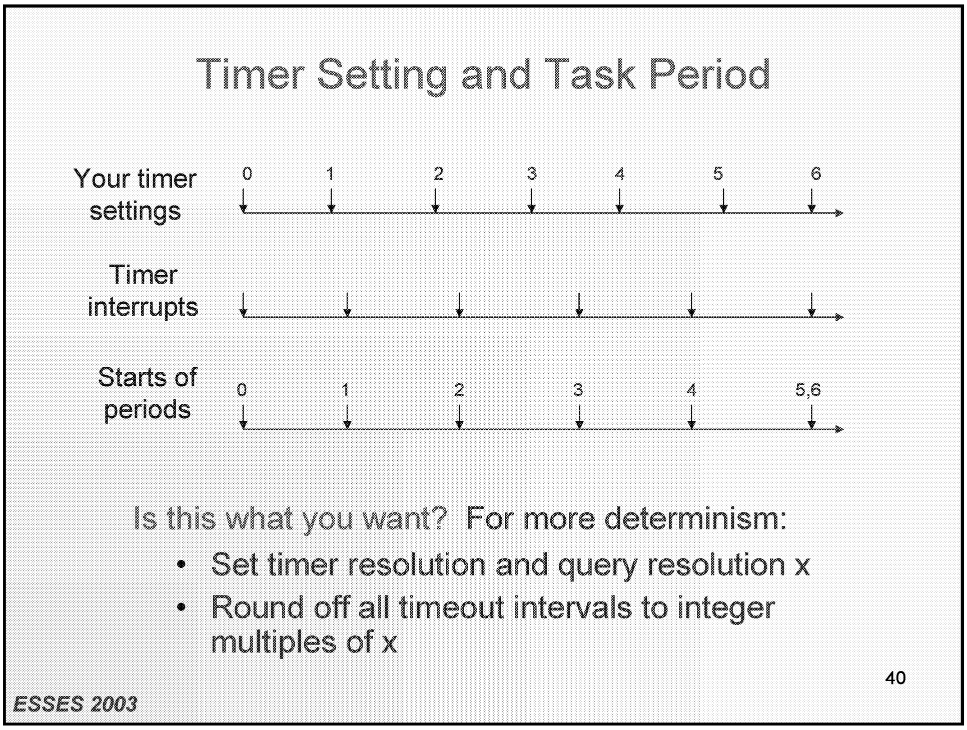
Timer DPC's

Timer interrupts

Expiration times

Timer expiration detected

Try to make do with Granularity ~1 millisecond or larger



Timer Setting and Task Period

Your timer settings: 0 1 2 3 4 5 6

Timer interrupts

Starts of periods: 0 1 2 3 4 5,6

Is this what you want? For more determinism:

- Set timer resolution and query resolution x
- Round off all timeout intervals to integer multiples of x

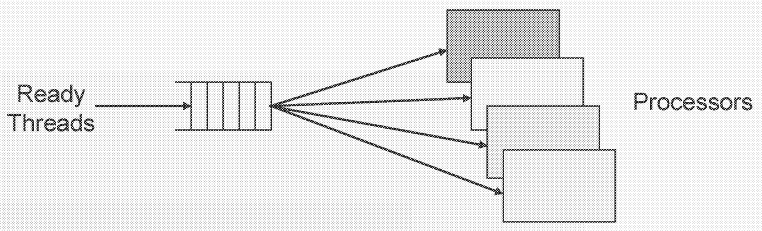
Single-Threaded Avionics (Cycle-executive)

```
#include <windows.h>
#include <stdio.h>
. . .
#define FRAME_LENGTH 34     // ~ 30.1 Hz
#define FRAMES_PER_MAJOR_FRAME 6
. . .
main (int argc, char *argv[])
{
   LARGE_INTEGER CurrentTime;
   ULONG Frame = FRAME_LENGTH;
   ULONG FrameNumber = 0;
   ULONG FrameRatio = FRAMES_PER_MAJOR_FRAME;
   LARGE_INTEGER Phase;
   BOOLEAN Run = TRUE;
   LARGE_INTEGER StartTime;
   HANDLE Timer;
   // Capture StartTime,  initialize, set timer resolution to 1 ms, and
   // create a synchronization timer.
   . . .
   timeBeginPeriod(1);
   Timer = CreateWaitableTimer(NULL, FALSE, NULL);
   NtQuerySystemTime(&CurrentTime);
   if (CurrentTime.QuadPart > StartTime.QuadPart) {
      Phase.QuadPart = CurrentTime.QuadPart;
   } else {
      Phase.QuadPart = SystemTime.QuadPart;
   }
```

Single-Threaded Avionics

```
if ( (NULL == Timer) ||
    !SetWaitableTimer (Timer, Phase, Frame,
                        NULL, NULL, FALSE) ) {
     // Clean up and return
     return;
}
while (Run)  {
      WaitForSingleObject(Timer, INFINITE);
      // At start of every frame, do the following
      ValidationSensorData();
      DetectFailureAndReconfigure();
      // Do the lowest rate task once a major frame
      switch (FrameNumber) {
      case 0:
         SampleKeyboardInputAndModeSelect();
         break;
      case 1:
         NormalizeDataTransformCoordinates();
         break;
      case 2:
         UpdateTrackingReference();
         break;
      case 3:
         OuterPitchControlLaw();
         break;
      case 4:
         OuterRollControlLaw();
         break;
      case 5:
         OuterYawCollectiveControlLaw();
         break;
      default:
         break;
      }
      // Do the medium rate tasks
      if (0 != FrameNumber/2) {
         InterPitchControlLaw();
      } else {
         InterRollCollectiveControlLaw();
      }
      // Do at the end of every frame
      FrameNumber +=1;
      FrameNumber %= FrameRatio;
      InterYawControlLaw();
      OutputCommands();
      BuiltInTest();
      ContinueToRun(&Run);
   }
   // Clean up and return.
   . . . .
   return;
}
```

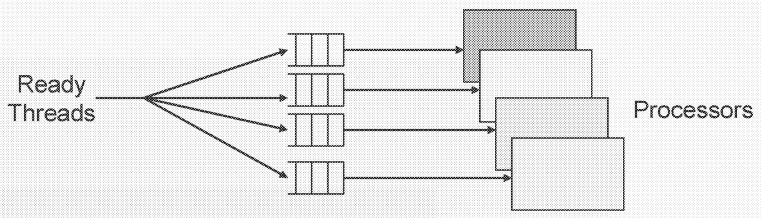
Typical Abstract SMP Model



- When a thread becomes ready, it preempts the lowest priority running thread if it has a higher priority and no processor is idle.
- When a thread stops running, the scheduler replaces it with the highest priority ready thread.

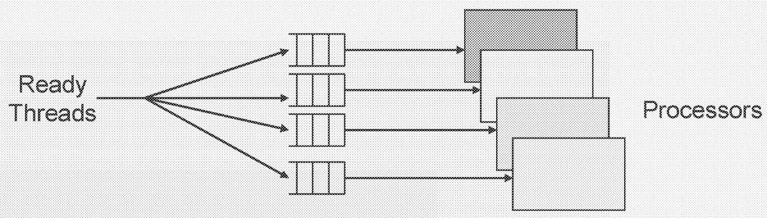
Dynamic dispatch straightforward

Actual SMP Queue Structure



- When a thread becomes ready, the scheduler may not choose to dispatch it on the processor running the lowest priority thread.
- A newly ready thread is put in the local queue of its chosen processor if it cannot preempt the running thread.
- When a thread stops running, the scheduler replaces it by the highest priority thread in the local queue.
- Threads migrate among processors only during load balancing.

Priority Inversion in SMP



Ready Threads

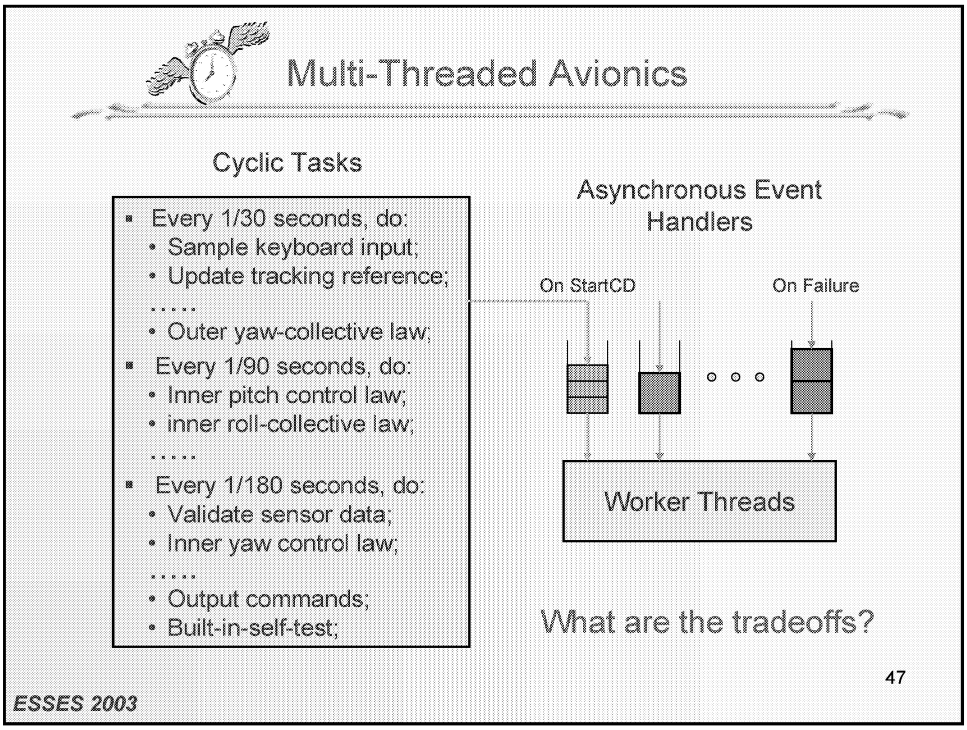
Processors

- Dispatch-thread function may not select the highest priority thread.
- Ready-thread function may not preempt the lowest priority thread.

⇨ ***Uncontrolled priority inversion in dynamic systems***

Means for Static Configuration

- Examples from Windows:
  - IoConnectInterrupt(…,ProcessorEnableMask, …) allows the choice of processors on which interrupts of the device can occur.
  - KeSetTargetProcessorDpc(Dpc, Number) targets a DPC to be queued and executed on a specified processor, other than the processor where the interrupt occurs.
  - KeSetAffinityThread(Thread, Affinity) constraints where a thread executes.
- Similar functions on other platforms.

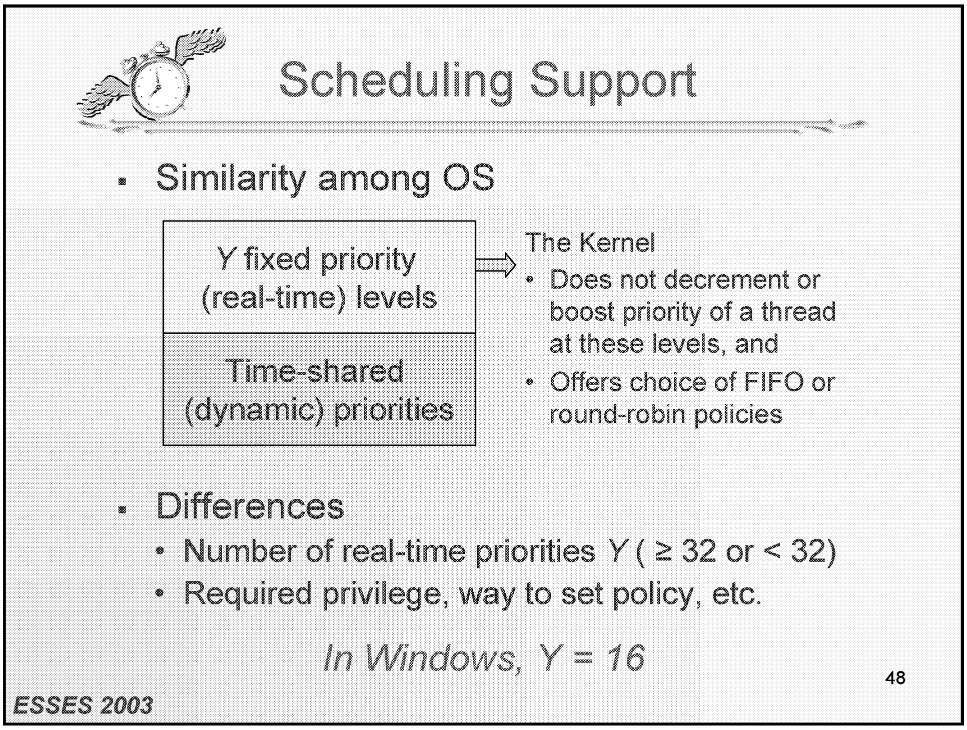


Multi-Threaded Avionics

Cyclic Tasks

- Every 1/30 seconds, do:
  - Sample keyboard input;
  - Update tracking reference;
  - . . . . .
  - Outer yaw-collective law;
- Every 1/90 seconds, do:
  - Inner pitch control law;
  - inner roll-collective law;
  - . . . . .
- Every 1/180 seconds, do:
  - Validate sensor data;
  - Inner yaw control law;
  - . . . . .
  - Output commands;
  - Built-in-self-test;

What are the tradeoffs?



Scheduling Support

- Similarity among OS

Y fixed priority (real-time) levels
Time-shared (dynamic) priorities

The Kernel

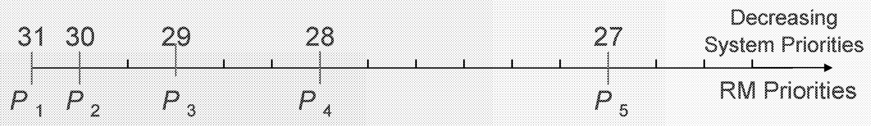
- Does not decrement or boost priority of a thread at these levels, and
- Offers choice of FIFO or round-robin policies

- Differences
  - Number of real-time priorities *Y* (≥ 32 or < 32)
  - Required privilege, way to set policy, etc.

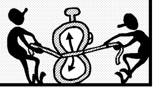
In Windows, Y = 16

How to Live with 16 RT Priorities

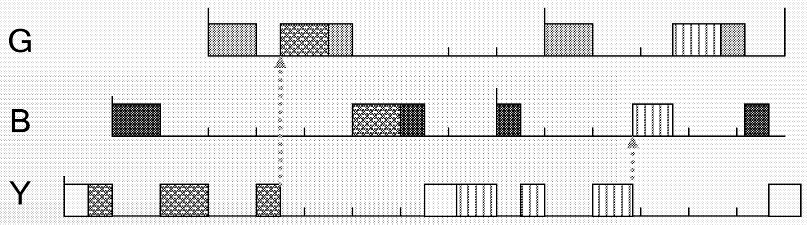
- Make the problem go away:
  - Divide threads into partitions and make priorities across partitions incomparable (with OS support)
  - Divide periodic tasks into rate groups
- Use constant-ratio mapping (Lehoczky and Sha)



- $P_i < P_{i+1}$, $P_i / P_{i+1} = g < 1$ for all $1 \le i \le 16$
- Schedulable utilization = g if $g \le 0.5$
  $\ln(2g) + 1 - g$ $g > 0.5$

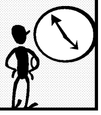


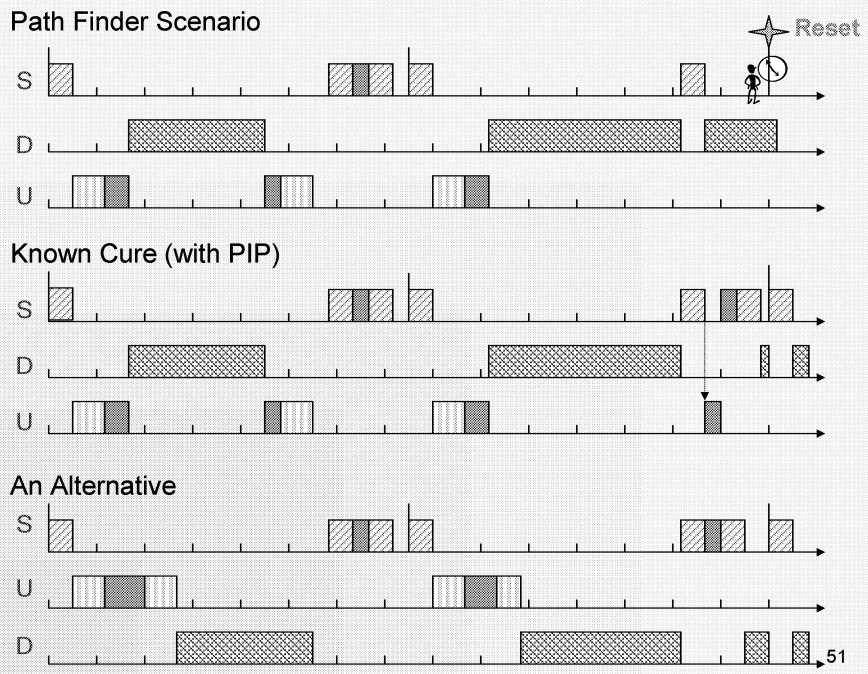
Waitable Objects



- Waitable objects include event, timer, mutant, semaphore, queue, I/O completion port, etc.
- A caller waits for an object in FIFO or priority order depending whether the object is real-time.

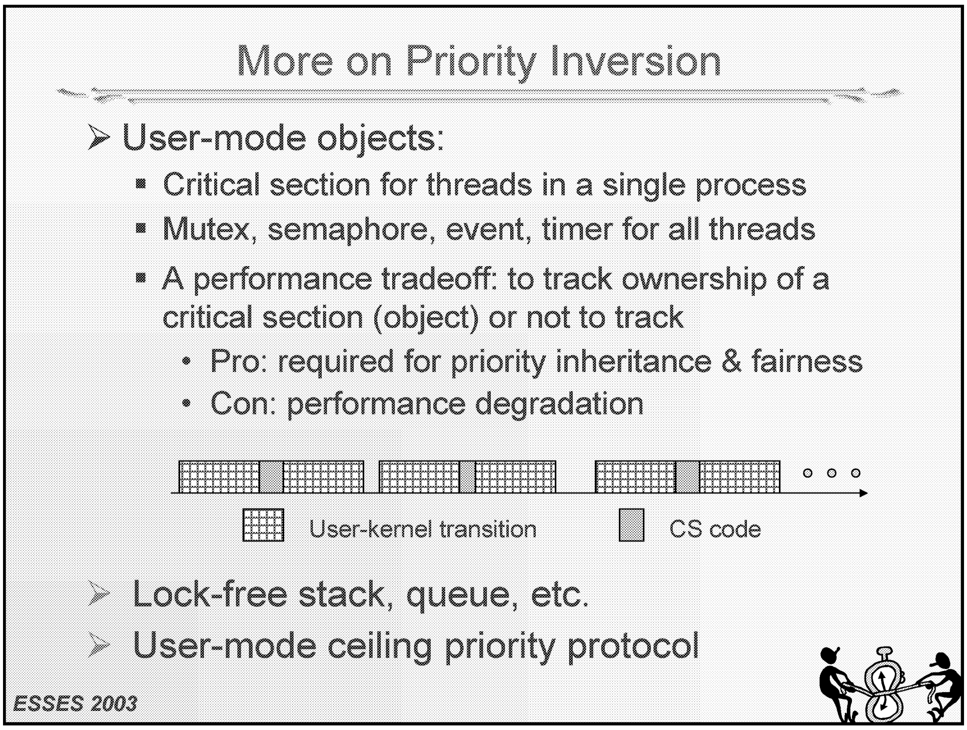
No priority inheritance; why?





On Priority Inversion

- Objects with no owners:
  - They include events, queue, timer, IO completion port, semaphore, process, thread, etc.
  - Priority inheritance & ceiling priority protocols are not applicable even on UP
- Use NPCS protocol for mutant and resources:
  - An implementation by application is:
    - Raise priority to the highest level
    - Wait for and acquire object (or objects)
    - Release the object
    - Restore priority to original level
  - Blocking can be more than one critical section.

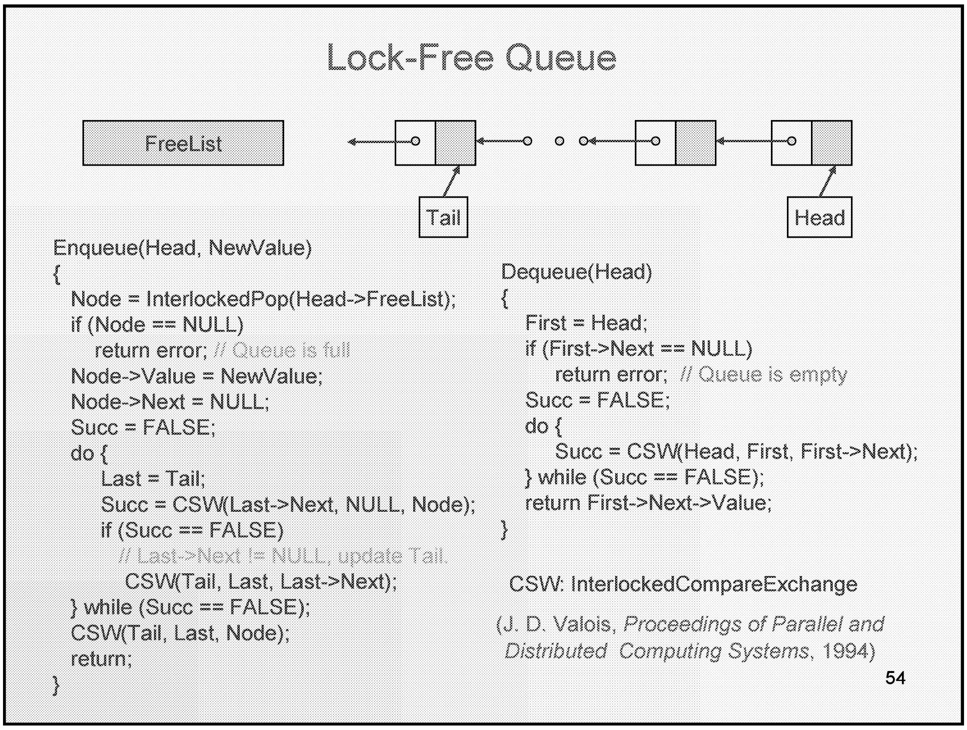


More on Priority Inversion

- User-mode objects:
  - Critical section for threads in a single process
  - Mutex, semaphore, event, timer for all threads
  - A performance tradeoff: to track ownership of a critical section (object) or not to track
    - Pro: required for priority inheritance & fairness
    - Con: performance degradation

▦ User-kernel transition ▨ CS code

- Lock-free stack, queue, etc.
- User-mode ceiling priority protocol



Lock-Free Queue

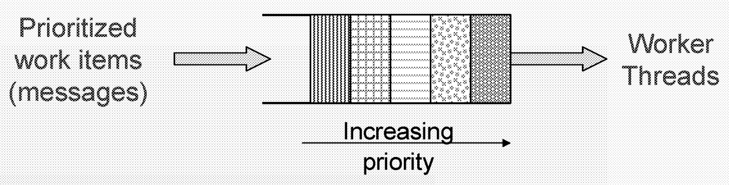
```
Enqueue(Head, NewValue)
{
  Node = InterlockedPop(Head->FreeList);
  if (Node == NULL)
     return error; // Queue is full
  Node->Value = NewValue;
  Node->Next = NULL;
  Succ = FALSE;
  do {
     Last = Tail;
     Succ = CSW(Last->Next, NULL, Node);
     if (Succ == FALSE)
       // Last->Next != NULL, update Tail.
        CSW(Tail, Last, Last->Next);
  } while (Succ == FALSE);
  CSW(Tail, Last, Node);
  return;
}
```

```
Dequeue(Head)
{
   First = Head;
   if (First->Next == NULL)
       return error;  // Queue is empty
   Succ = FALSE;
   do {
       Succ = CSW(Head, First, First->Next);
   } while (Succ == FALSE);
   return First->Next->Value;
}
```

CSW: InterlockedCompareExchange

(J. D. Valois, *Proceedings of Parallel and Distributed Computing Systems*, 1994)

Prioritized Messages



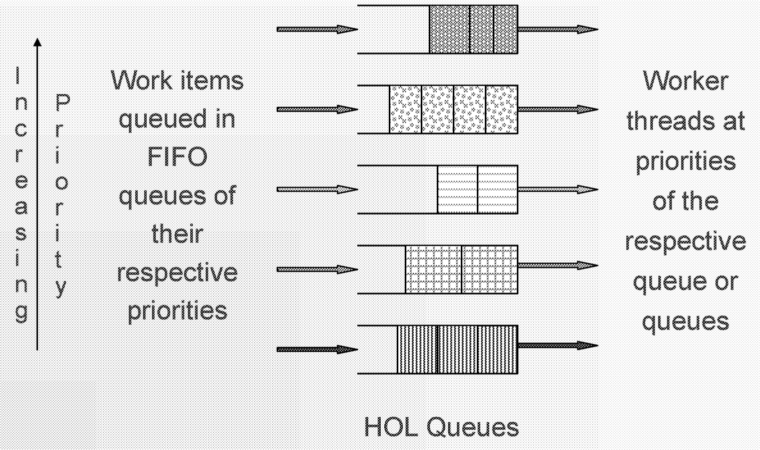
Message-Based Priority Inheritance

Programming for POSIX 4,by B. O. Gallmeister

Each thread do until terminated:

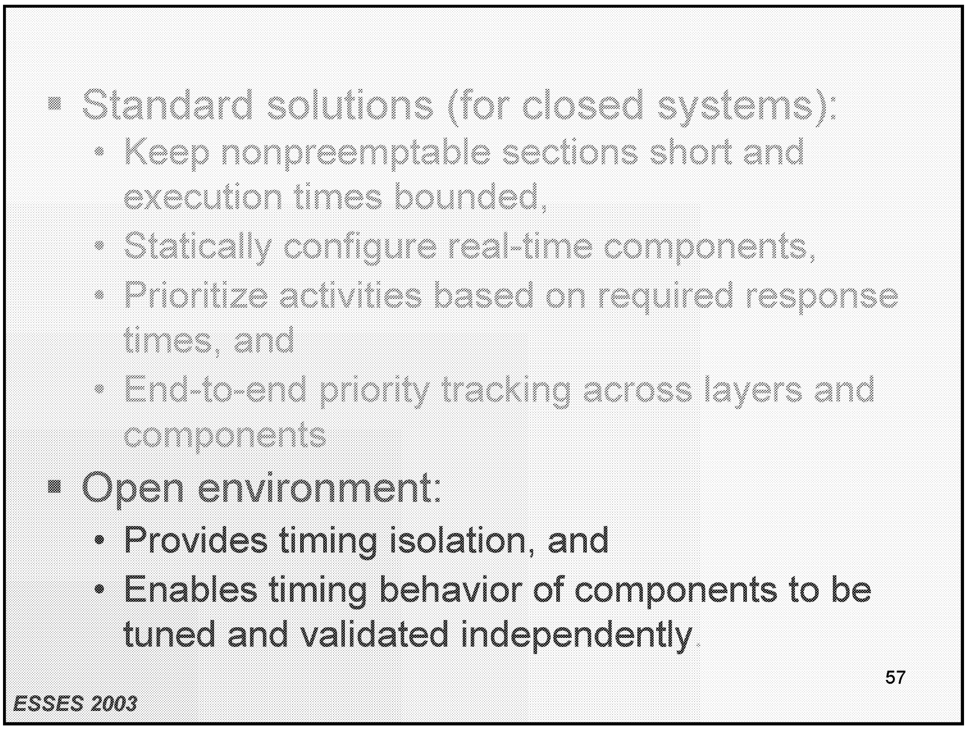
- Waits at the highest priority
- Upon removal of an item, sets its priority to the priority of the item
- Upon completion of the item, raise priority to highest and go wait for the queue

Prioritized Messages without priority queues

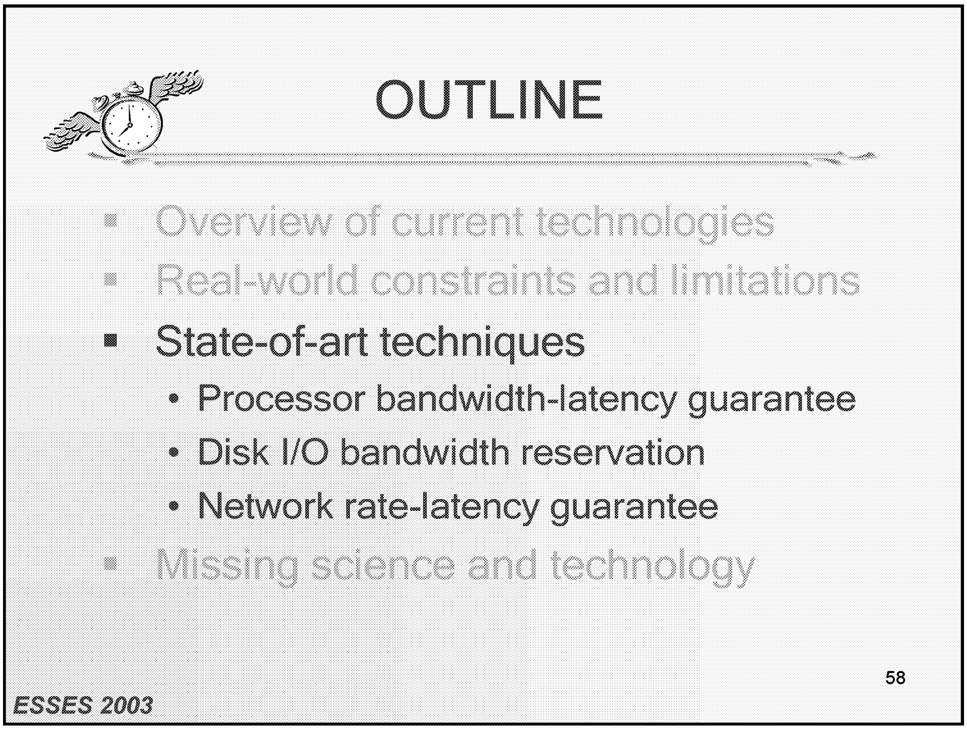


Work items queued in FIFO queues of their respective priorities

Worker threads at priorities of the respective queue or queues

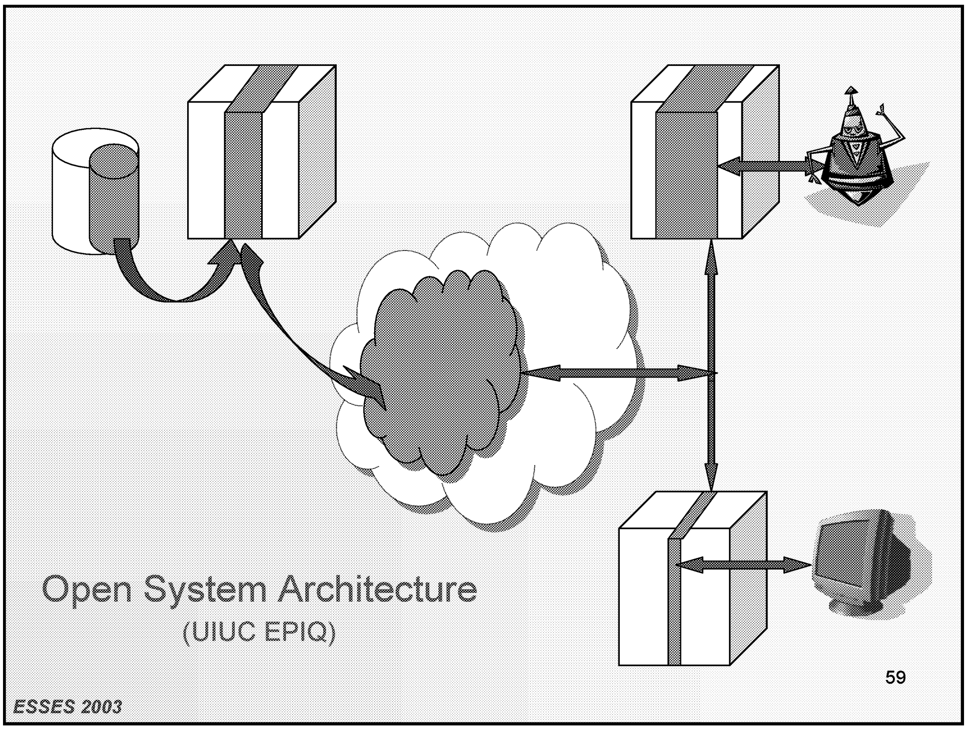


- Standard solutions (for closed systems):
  - Keep nonpreemptable sections short and execution times bounded,
  - Statically configure real-time components,
  - Prioritize activities based on required response times, and
  - End-to-end priority tracking across layers and components
- Open environment:
  - Provides timing isolation, and
  - Enables timing behavior of components to be tuned and validated independently.



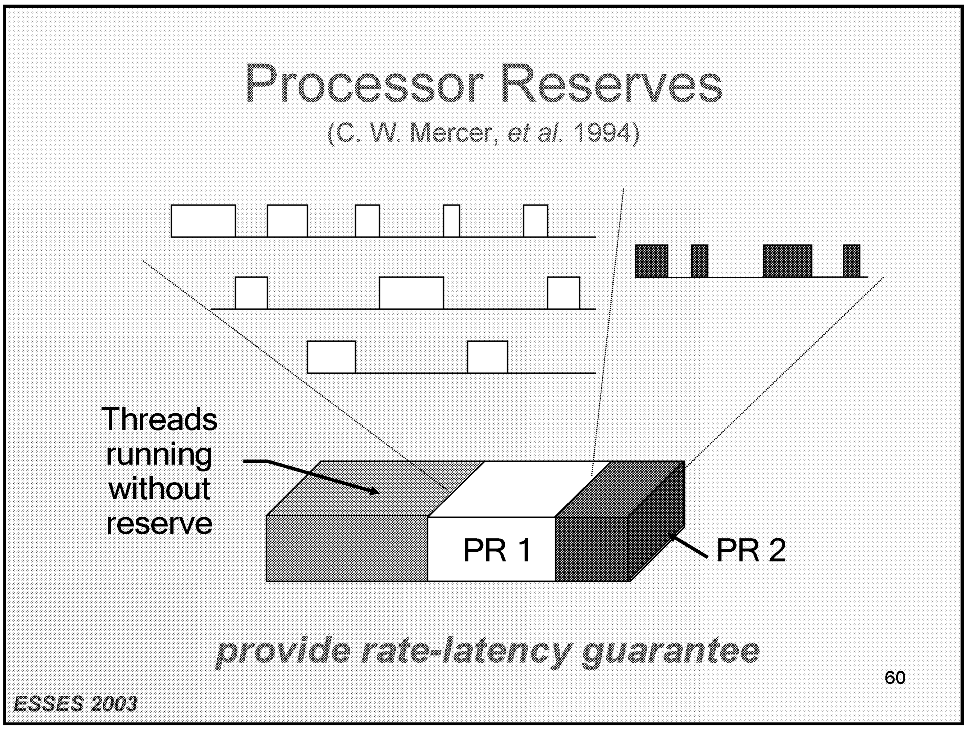
OUTLINE

- Overview of current technologies
- Real-world constraints and limitations
- State-of-art techniques
  - Processor bandwidth-latency guarantee
  - Disk I/O bandwidth reservation
  - Network rate-latency guarantee
- Missing science and technology



Open System Architecture

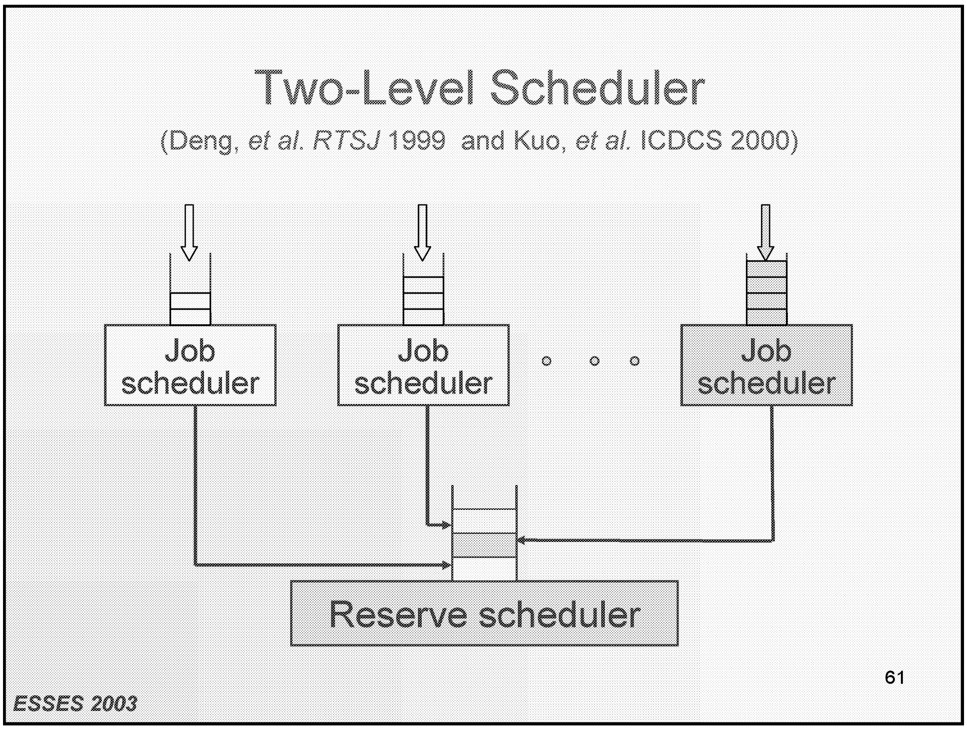
(UIUC EPIQ)



Processor Reserves

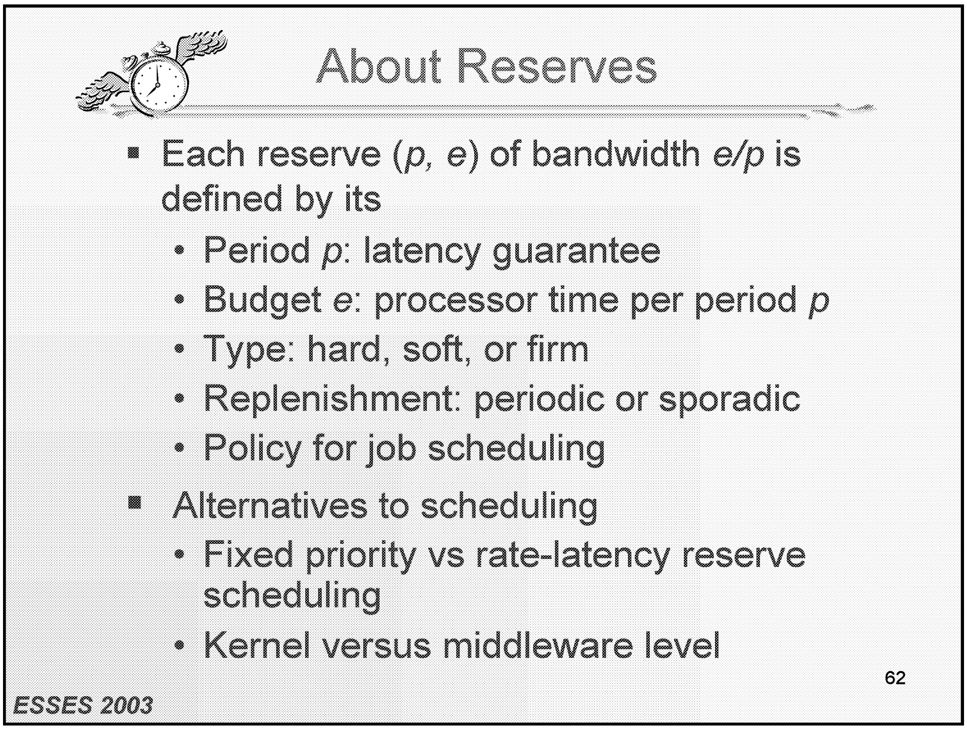
(C. W. Mercer, *et al.* 1994)

provide rate-latency guarantee



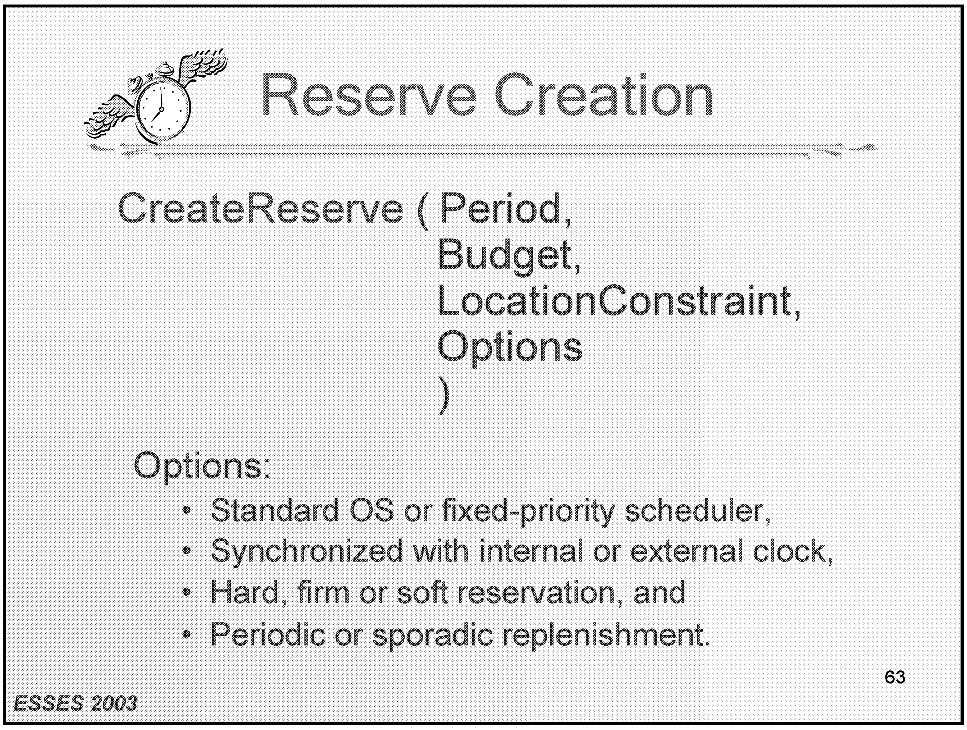
Two-Level Scheduler

(Deng, *et al. RTSJ* 1999 and Kuo, *et al.* ICDCS 2000)



About Reserves

- Each reserve (*p, e*) of bandwidth *e/p* is defined by its
  - Period *p*: latency guarantee
  - Budget *e*: processor time per period *p*
  - Type: hard, soft, or firm
  - Replenishment: periodic or sporadic
  - Policy for job scheduling
- Alternatives to scheduling
  - Fixed priority vs rate-latency reserve scheduling
  - Kernel versus middleware level

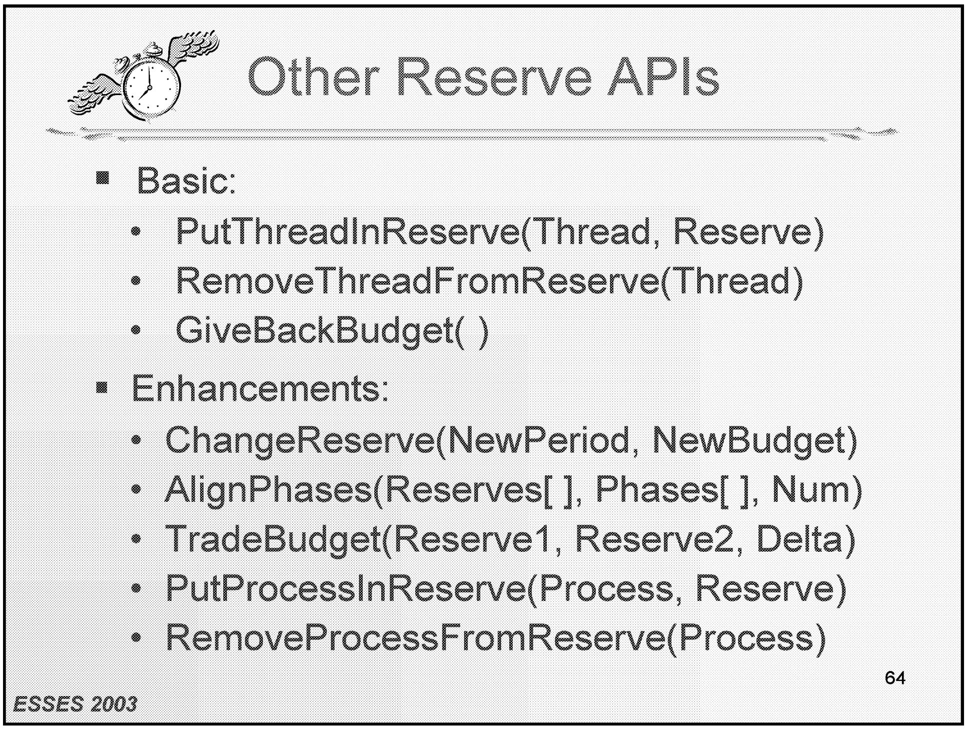


Reserve Creation

CreateReserve (Period,
Budget,
LocationConstraint,
Options
)

Options:

- Standard OS or fixed-priority scheduler,
- Synchronized with internal or external clock,
- Hard, firm or soft reservation, and
- Periodic or sporadic replenishment.



Other Reserve APIs

- Basic:
  - PutThreadInReserve(Thread, Reserve)
  - RemoveThreadFromReserve(Thread)
  - GiveBackBudget( )
- Enhancements:
  - ChangeReserve(NewPeriod, NewBudget)
  - AlignPhases(Reserves[ ], Phases[ ], Num)
  - TradeBudget(Reserve1, Reserve2, Delta)
  - PutProcessInReserve(Process, Reserve)
  - RemoveProcessFromReserve(Process)

Alternatives in scheduling
Hard Reserves

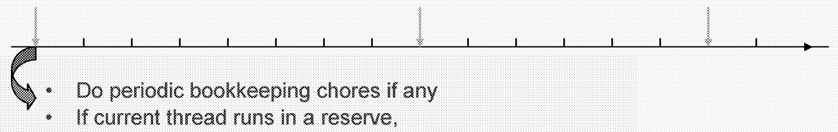
- Earliest-deadline-first approach (Deng, *et al.*)
  - Pros: uniform treatment of periodic and sporadic reserves, simple acceptance test, easy to enhance (budget preservation)
  - Con: higher complexity of EDF queuing
- Fixed-priority approach (Kuo, *et al.*)
  - Pro: uniform reserve and thread queuing
  - Con: high complexity in maintaining sporadic reserves

Instability due to overruns never arises

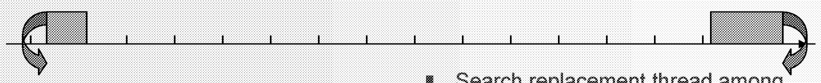
Reserve Scheduling

A simpler version of EDF-sporadic server algorithm (Ghazalie & Baker)

Clock interrupts



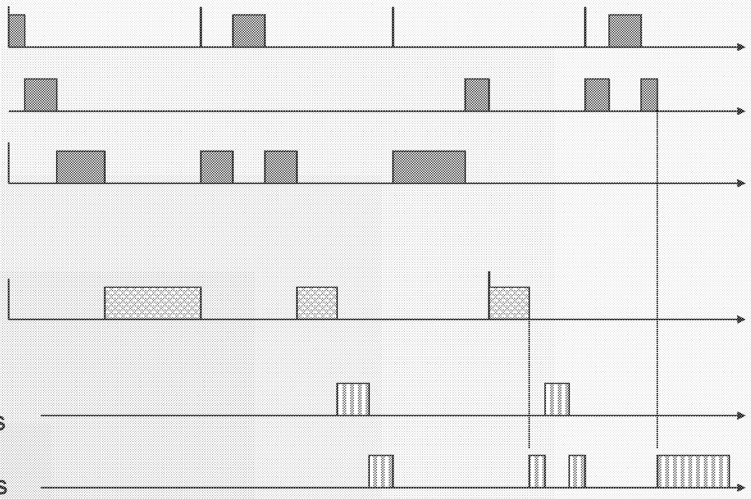
- Do periodic bookkeeping chores if any
- If current thread runs in a reserve,
  - Decrement reserve budget
  - If budget is exhausted, remove the reserve from reserve queue
- Replenish each reserve due for replenishment and insert it in EDF order in reserve queue according to its new deadline
- Swap reserve and thread if head of the reserve queue changed



If the newly ready thread runs in a reserve update reserve budget and queue accordingly

- Search replacement thread among
  - threads in the same reserve
  - threads in some ready reserve
  - threads without reserve, etc.
- Reclaim budget and set replenishment time as needed

Reserve and Thread Scheduling



(2, 1)

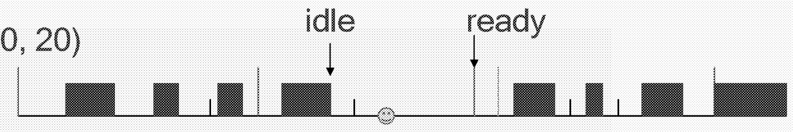
(5, 1.5)

Threads without reserves

Behavior of Reserves

Sporadic

(50, 20)

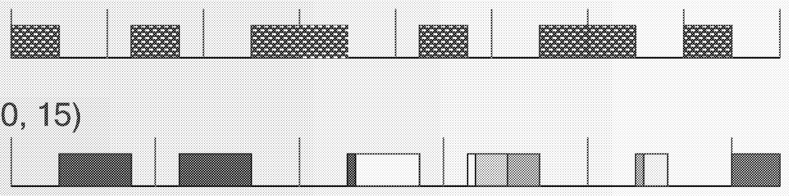


idle

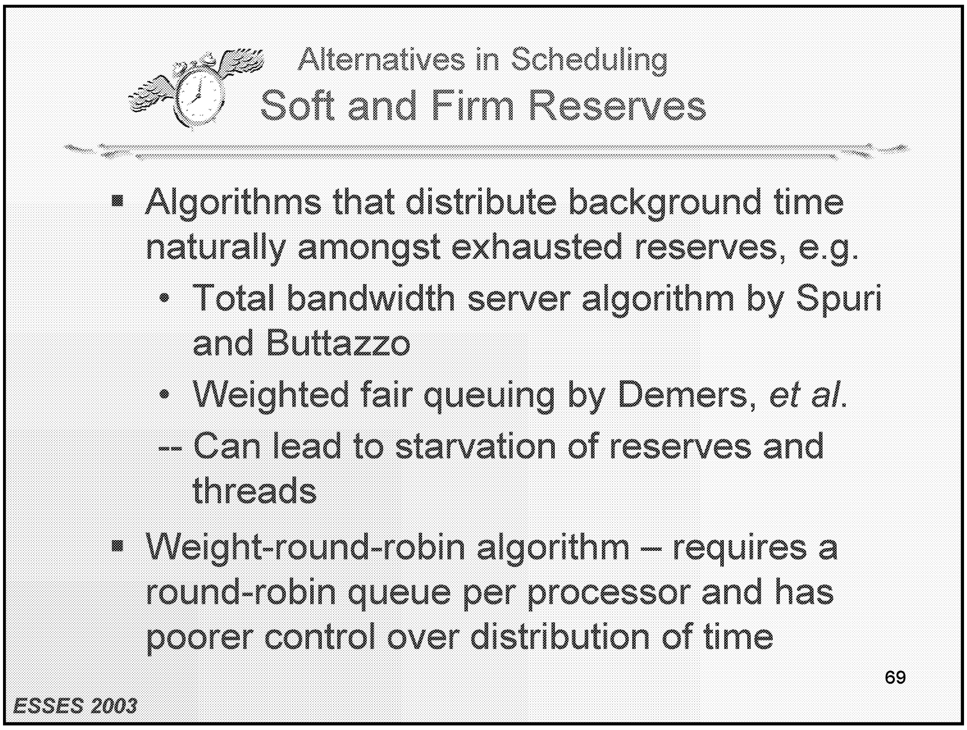
ready

Periodic

(20, 10)



(30, 15)

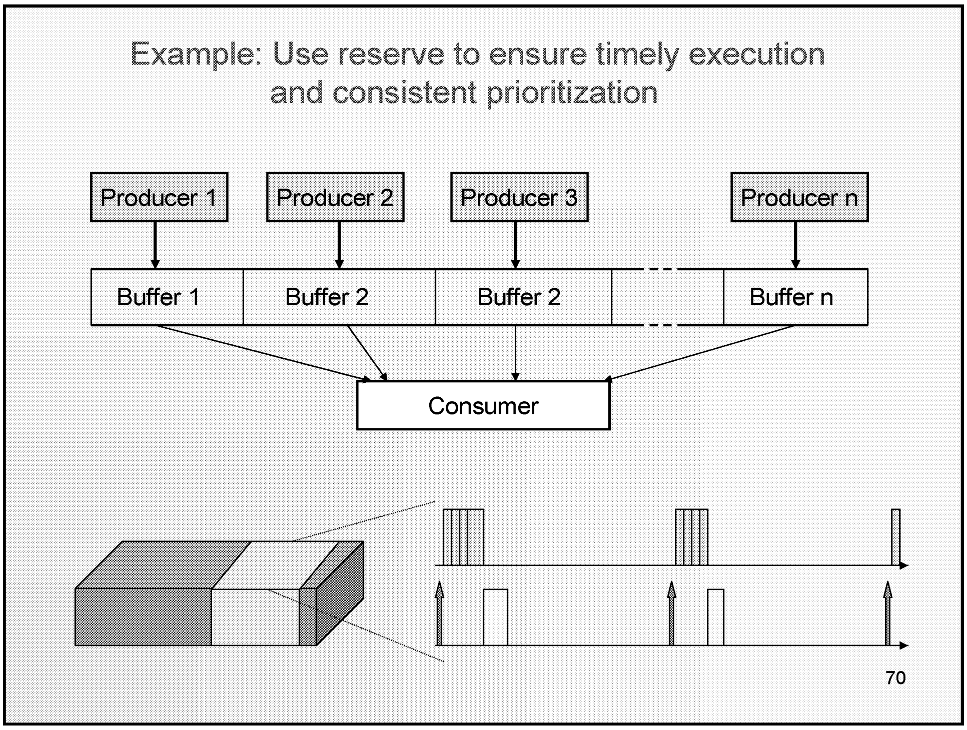


Alternatives in Scheduling

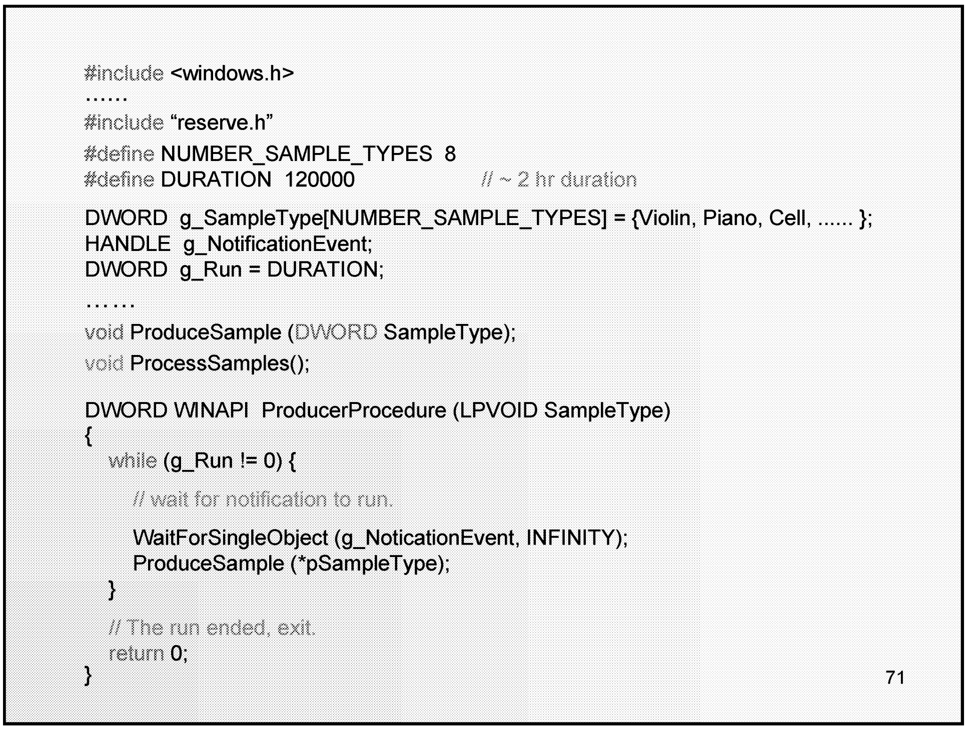
Soft and Firm Reserves

- Algorithms that distribute background time naturally amongst exhausted reserves, e.g.
  - Total bandwidth server algorithm by Spuri and Buttazzo
  - Weighted fair queuing by Demers, *et al*.

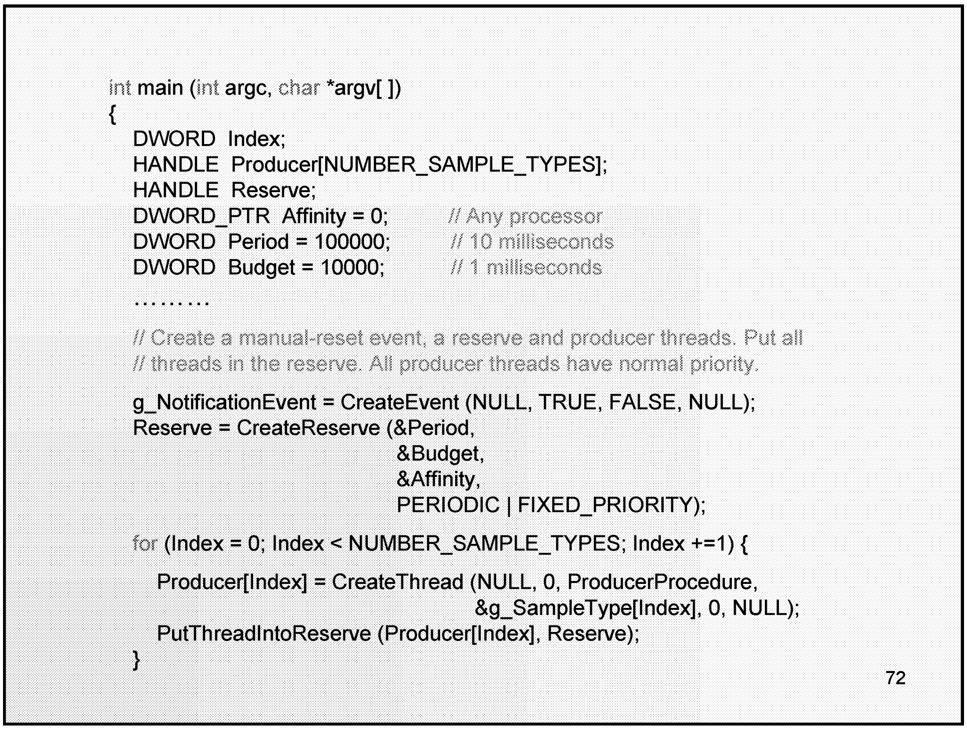
  -- Can lead to starvation of reserves and threads
- Weight-round-robin algorithm – requires a round-robin queue per processor and has poorer control over distribution of time



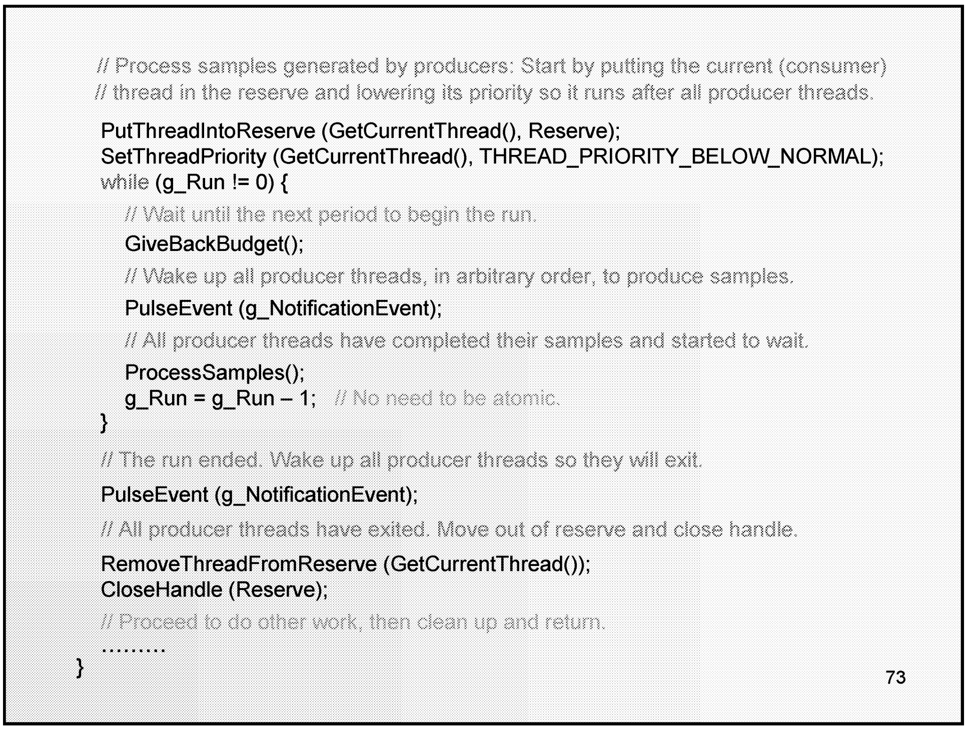
Example: Use reserve to ensure timely execution and consistent prioritization



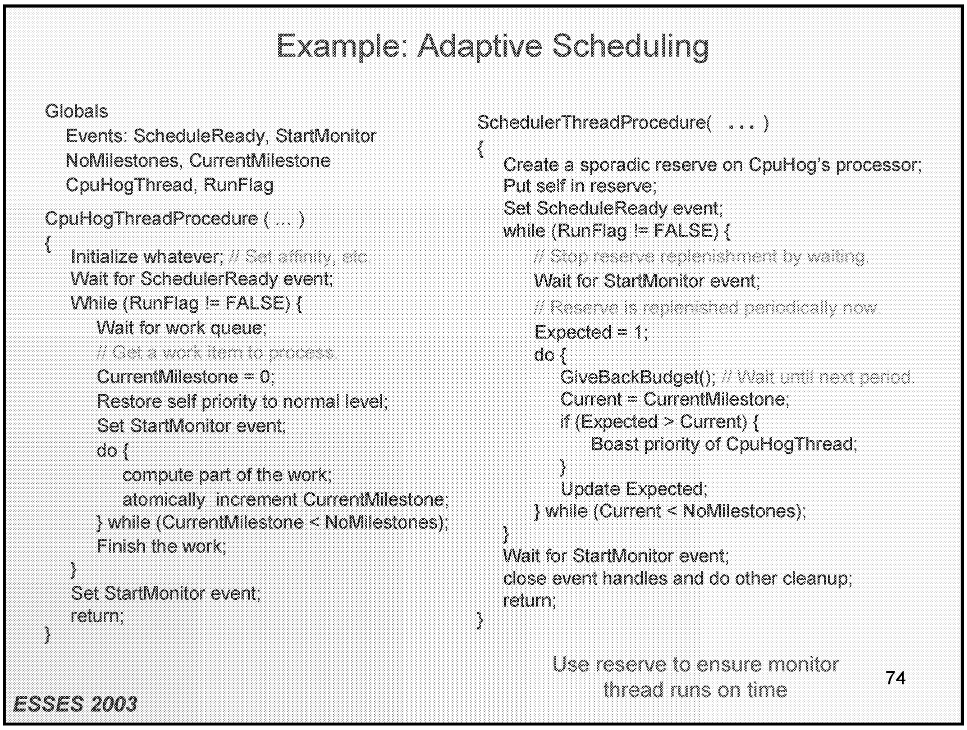
```
#include <windows.h>
……
#include “reserve.h”
#define NUMBER_SAMPLE_TYPES  8
#define DURATION  120000                    // ~ 2 hr duration
DWORD  g_SampleType[NUMBER_SAMPLE_TYPES] = {Violin, Piano, Cell, ...... };
HANDLE  g_NotificationEvent;
DWORD  g_Run = DURATION;
……
void ProduceSample (DWORD SampleType);
void ProcessSamples();
DWORD WINAPI  ProducerProcedure (LPVOID SampleType)
{
   while (g_Run != 0) {
      // wait for notification to run.
      WaitForSingleObject (g_NoticationEvent, INFINITY);
      ProduceSample (*pSampleType);
   }
   // The run ended, exit.
   return 0;
}
```



```
int main (int argc, char *argv[ ])
{
   DWORD  Index;
   HANDLE  Producer[NUMBER_SAMPLE_TYPES];
   HANDLE  Reserve;
   DWORD_PTR  Affinity = 0;         // Any processor
   DWORD  Period = 100000;          // 10 milliseconds
   DWORD  Budget = 10000;           // 1 milliseconds
   ………
   // Create a manual-reset event, a reserve and producer threads. Put all
   // threads in the reserve. All producer threads have normal priority.
   g_NotificationEvent = CreateEvent (NULL, TRUE, FALSE, NULL);
   Reserve = CreateReserve (&Period,
                            &Budget,
                            &Affinity,
                            PERIODIC | FIXED_PRIORITY);
   for (Index = 0; Index < NUMBER_SAMPLE_TYPES; Index +=1) {
      Producer[Index] = CreateThread (NULL, 0, ProducerProcedure,
                                      &g_SampleType[Index], 0, NULL);
      PutThreadIntoReserve (Producer[Index], Reserve);
   }
```



```
    // Process samples generated by producers: Start by putting the current (consumer)
    // thread in the reserve and lowering its priority so it runs after all producer threads.
    PutThreadIntoReserve (GetCurrentThread(), Reserve);
    SetThreadPriority (GetCurrentThread(), THREAD_PRIORITY_BELOW_NORMAL);
    while (g_Run != 0) {
        // Wait until the next period to begin the run.
        GiveBackBudget();
        // Wake up all producer threads, in arbitrary order, to produce samples.
        PulseEvent (g_NotificationEvent);
        // All producer threads have completed their samples and started to wait.
        ProcessSamples();
        g_Run = g_Run – 1;   // No need to be atomic.
    }
    // The run ended. Wake up all producer threads so they will exit.
    PulseEvent (g_NotificationEvent);
    // All producer threads have exited. Move out of reserve and close handle.
    RemoveThreadFromReserve (GetCurrentThread());
    CloseHandle (Reserve);
    // Proceed to do other work, then clean up and return.
    .........
}
```



Example: Adaptive Scheduling

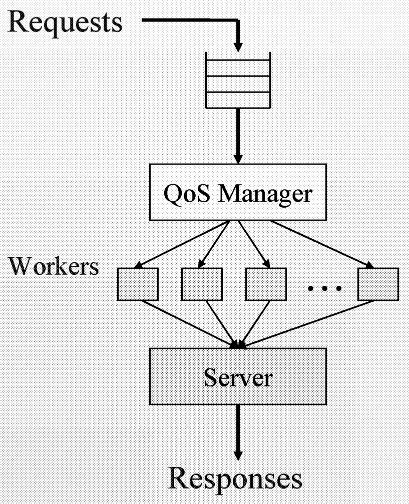
```
Globals
   Events: ScheduleReady, StartMonitor
   NoMilestones, CurrentMilestone
   CpuHogThread, RunFlag

CpuHogThreadProcedure ( ... )
{
   Initialize whatever; // Set affinity, etc.
   Wait for SchedulerReady event;
   While (RunFlag != FALSE) {
      Wait for work queue;
      // Get a work item to process.
      CurrentMilestone = 0;
      Restore self priority to normal level;
      Set StartMonitor event;
      do {
         compute part of the work;
         atomically  increment CurrentMilestone;
      } while (CurrentMilestone < NoMilestones);
      Finish the work;
   }
   Set StartMonitor event;
   return;
}
```

```
SchedulerThreadProcedure( ... )
{
   Create a sporadic reserve on CpuHog's processor;
   Put self in reserve;
   Set ScheduleReady event;
   while (RunFlag != FALSE) {
      // Stop reserve replenishment by waiting.
      Wait for StartMonitor event;
      // Reserve is replenished periodically now.
      Expected = 1;
      do {
         GiveBackBudget(); // Wait until next period.
         Current = CurrentMilestone;
         if (Expected > Current) {
            Boast priority of CpuHogThread;
         }
         Update Expected;
      } while (Current < NoMilestones);
   }
   Wait for StartMonitor event;
   close event handles and do other cleanup;
   return;
}
```

Use reserve to ensure monitor thread runs on time

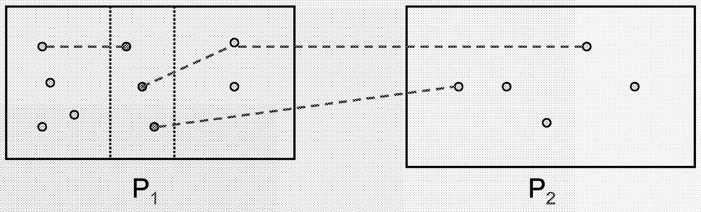
Example: Use reserve to constrain CPU time consumption of long-running threads



```
QoSManager( )
{
  …….
  Create Reserve R (300, 30);
  For each worker thread WT {
    - Get total thread time T;
    - if (T > 10000) {
        Put WT in reserve R;
      }
  }
  …….
}
```

Price of Reserves

- Loss in determinism – Highest priority thread may not run as soon as it becomes ready
- New concern for priority inversion.



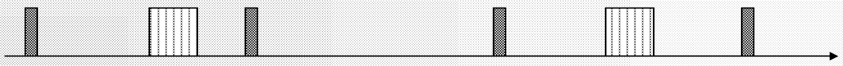
Problem can be overcome with a price.

Reminder: No free lunch

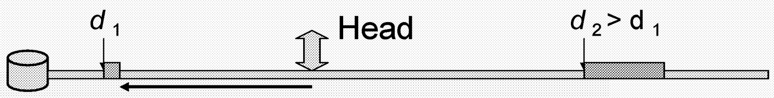
Disk Bandwidth Reservations

- Parameters of a disk bandwidth reserve:
  - Period *p*: minimum interval between requests = guaranteed latency
  - Number of blocks *n*: maximum number of blocks accessed per period

  → Guaranteed bandwidth = *n* / *p*



EDF-Based Disk Scheduling



- Basic strategy: seek to serve the waiting request with the earliest deadline
- Enhancements: serve requests on the way based on whether
  - there is slack (EDF-slack stealing), or
  - current time is within a window (EDF-window)

Pro: Highest schedulable utilization
Cons: Poor throughput, starvation, etc.

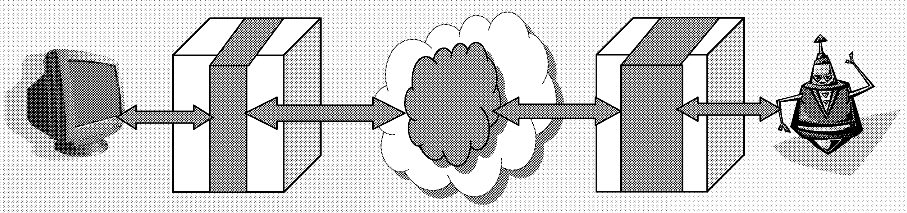
WRR Disk Scheduling

- Weighted-round-robin algorithm:
  - Time is divided into rounds of length ≤ R slots
  - Total slots per round allocated to all jobs is ≤ R
  - Each job is executed for at most its allocated no. of slots each round
  - When used for scheduling periodic tasks:
    - R ≤ the supported minimum period of all tasks
    - If admitted, a task (p, e) is allocated [e / (p / R)] slots per round, rounded up to an integer no.
- WRR disk scheduling:
  - A slot = transfer time of a minimum data block
  - Available slots per round = (R – round trip scan time)

Pros: simple, compatible to c-scan
Con: lower schedulable utilization

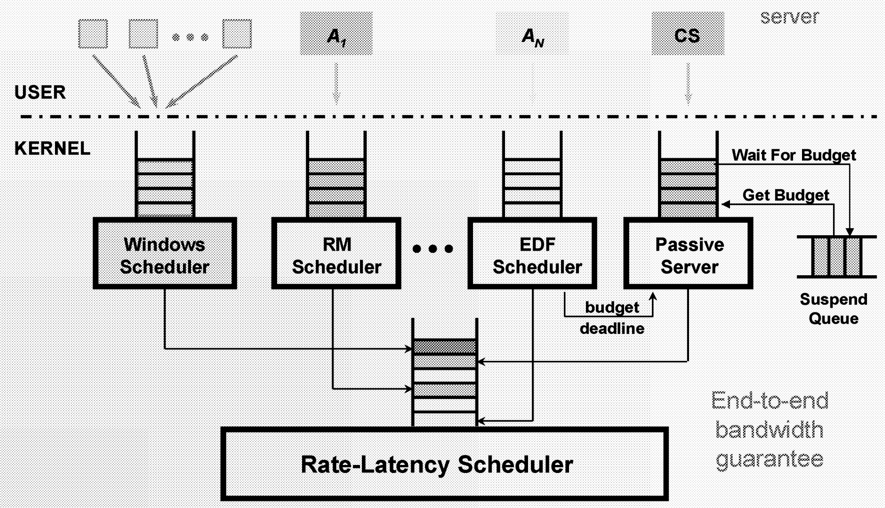
Open Network Environment

(Providing end-to-end rate-latency guarantee)



Good algorithms and protocols exist.
Realization takes integration technology

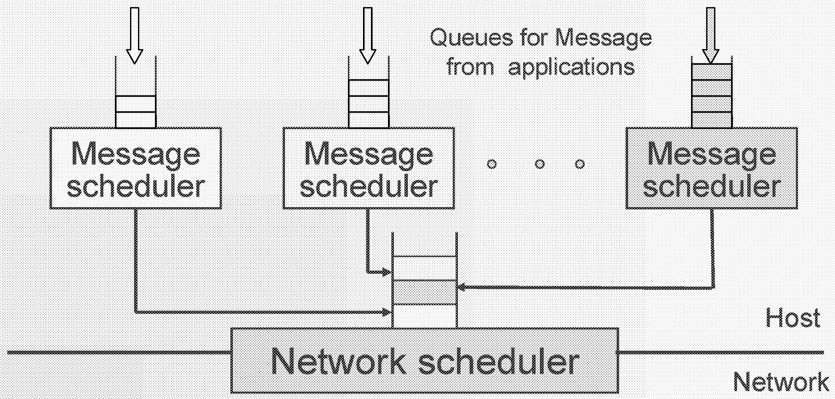
EPIQ Open System Architecture



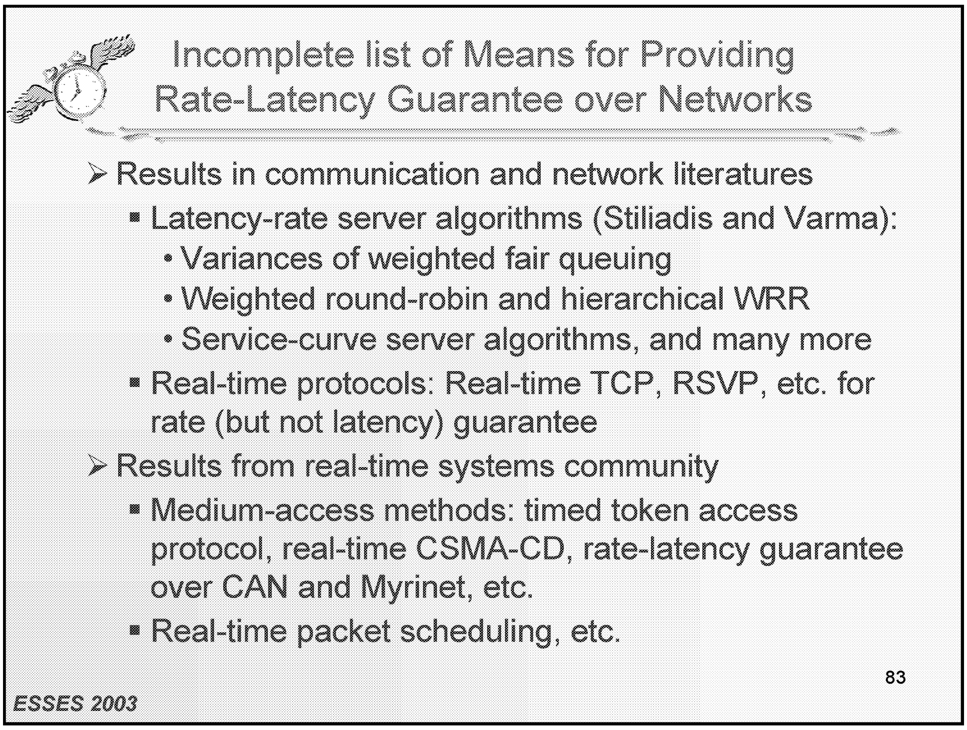
Communication server

End-to-end bandwidth guarantee

Two-Level Scheduler (at end-points)

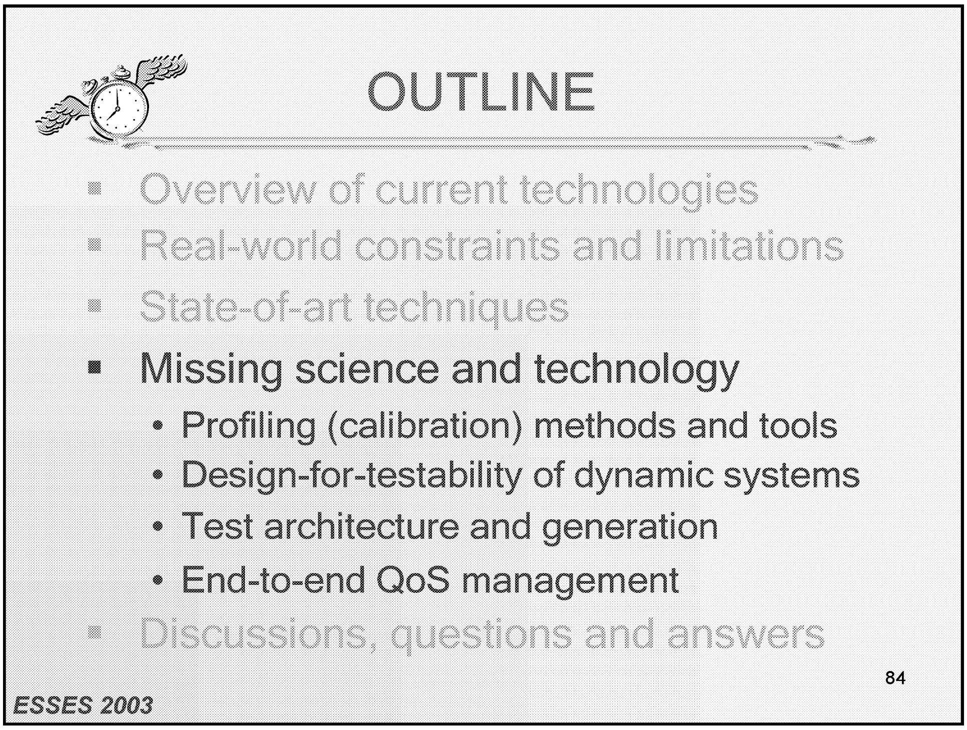


Queues for Message from applications



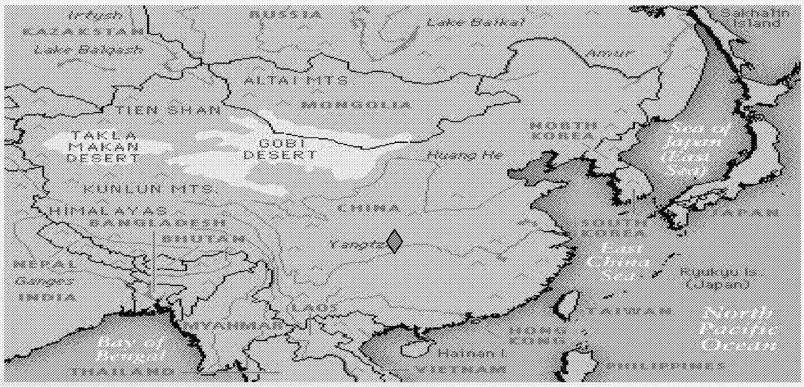
Incomplete list of Means for Providing Rate-Latency Guarantee over Networks

- Results in communication and network literatures
  - Latency-rate server algorithms (Stiliadis and Varma):
    - Variances of weighted fair queuing
    - Weighted round-robin and hierarchical WRR
    - Service-curve server algorithms, and many more
  - Real-time protocols: Real-time TCP, RSVP, etc. for rate (but not latency) guarantee
- Results from real-time systems community
  - Medium-access methods: timed token access protocol, real-time CSMA-CD, rate-latency guarantee over CAN and Myrinet, etc.
  - Real-time packet scheduling, etc.



OUTLINE

- Overview of current technologies
- Real-world constraints and limitations
- State-of-art techniques
- Missing science and technology
  - Profiling (calibration) methods and tools
  - Design-for-testability of dynamic systems
  - Test architecture and generation
  - End-to-end QoS management
- Discussions, questions and answers



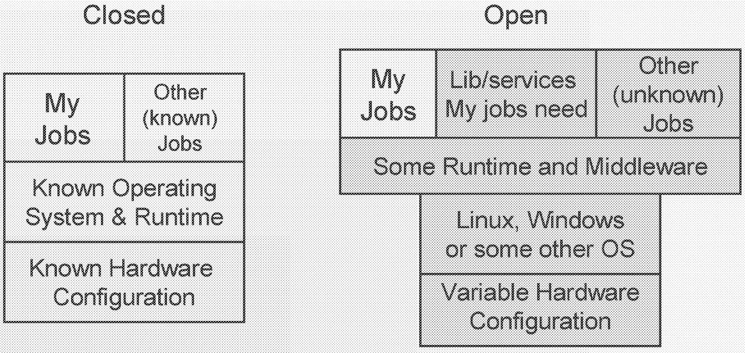
Date: Three Kingdoms   Place: Central China

Battle: Southern alliances against northern kingdom

萬事俱備　只欠東風

Everything is ready, except south-easterly wind

Calibration and Profiling



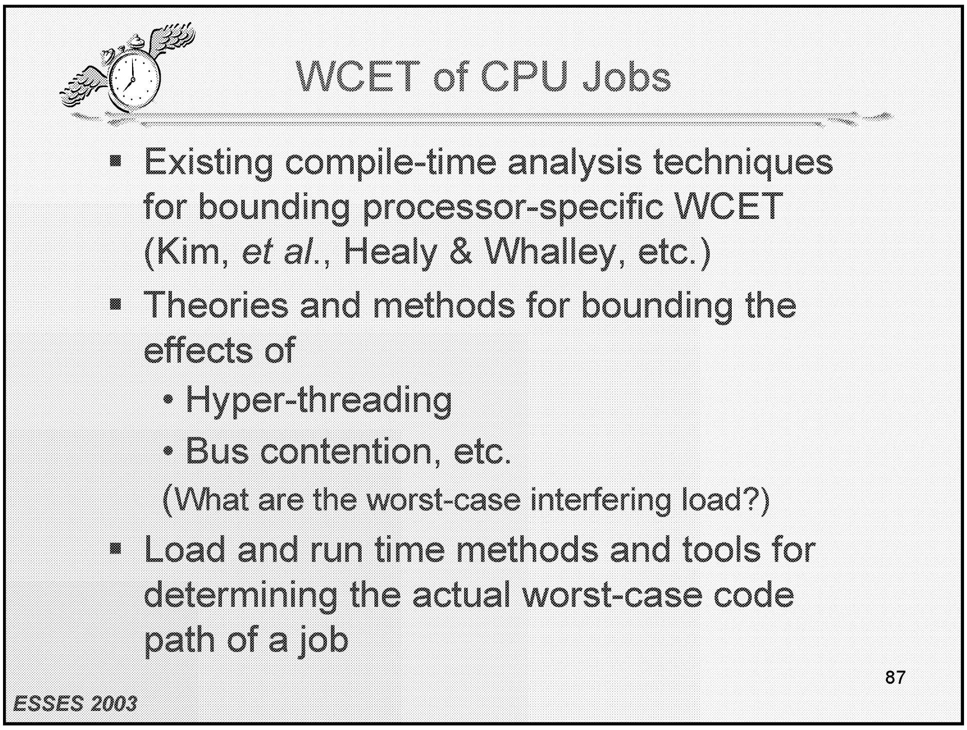
Closed

My Jobs	Other (known) Jobs
Known Operating System & Runtime	
Known Hardware Configuration	

Open

My Jobs	Lib/services My jobs need	Other (unknown) Jobs
Some Runtime and Middleware		
Linux, Windows or some other OS		
Variable Hardware Configuration		

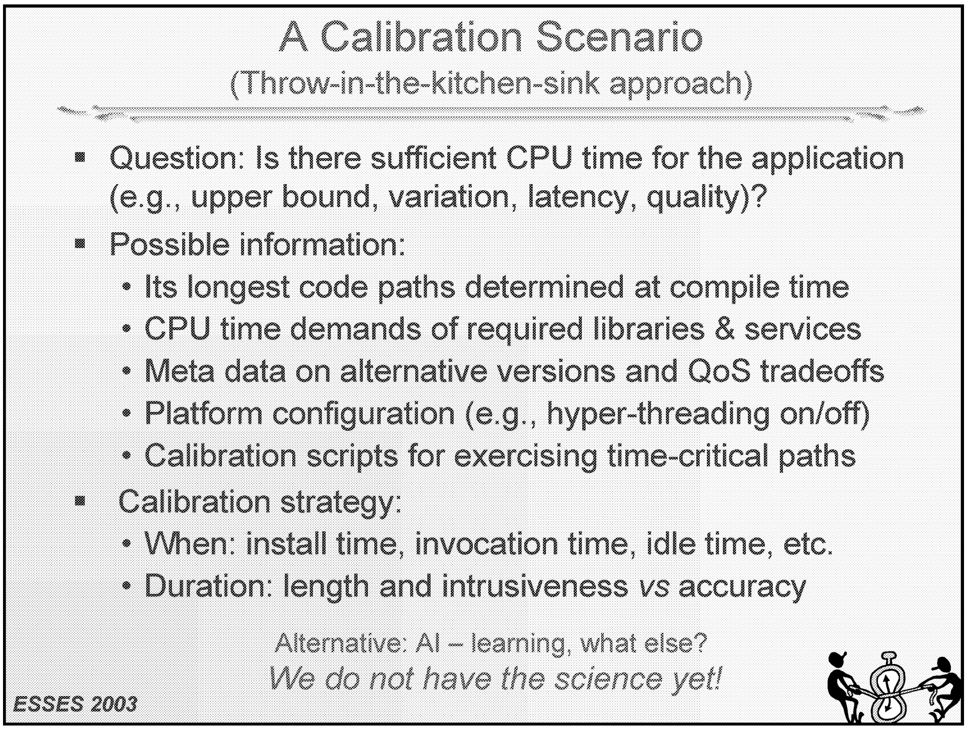
Question: What resources (e.g., memory & CPU cycles) my real-time application needs?



WCET of CPU Jobs

- Existing compile-time analysis techniques for bounding processor-specific WCET (Kim, *et al*., Healy & Whalley, etc.)
- Theories and methods for bounding the effects of
  - Hyper-threading
  - Bus contention, etc.

  (What are the worst-case interfering load?)
- Load and run time methods and tools for determining the actual worst-case code path of a job



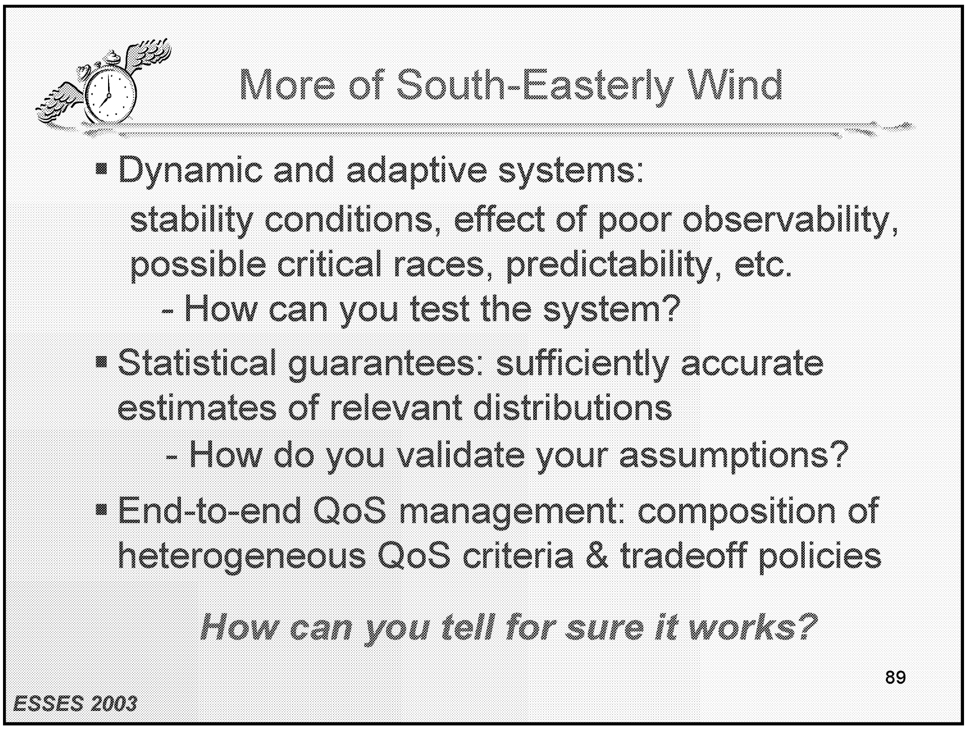
A Calibration Scenario

(Throw-in-the-kitchen-sink approach)

- Question: Is there sufficient CPU time for the application (e.g., upper bound, variation, latency, quality)?
- Possible information:
  - Its longest code paths determined at compile time
  - CPU time demands of required libraries & services
  - Meta data on alternative versions and QoS tradeoffs
  - Platform configuration (e.g., hyper-threading on/off)
  - Calibration scripts for exercising time-critical paths
- Calibration strategy:
  - When: install time, invocation time, idle time, etc.
  - Duration: length and intrusiveness *vs* accuracy

Alternative: AI – learning, what else?

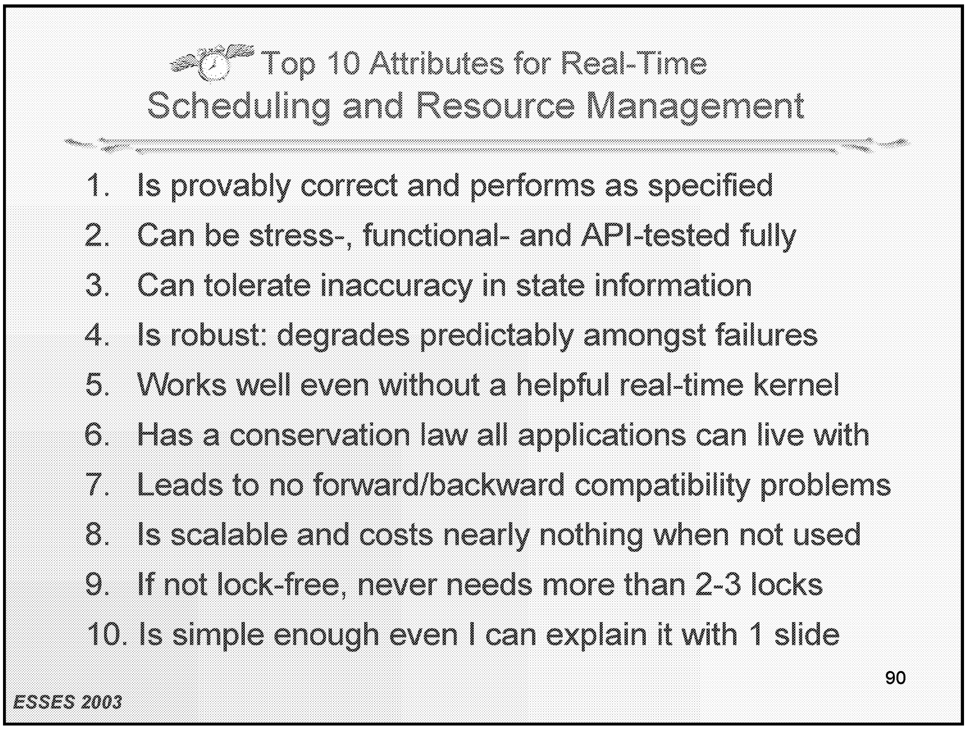
We do not have the science yet!



More of South-Easterly Wind

- Dynamic and adaptive systems:
  stability conditions, effect of poor observability, possible critical races, predictability, etc.
  - How can you test the system?
- Statistical guarantees: sufficiently accurate estimates of relevant distributions
  - How do you validate your assumptions?
- End-to-end QoS management: composition of heterogeneous QoS criteria & tradeoff policies

How can you tell for sure it works?



Top 10 Attributes for Real-Time Scheduling and Resource Management

1. Is provably correct and performs as specified
2. Can be stress-, functional- and API-tested fully
3. Can tolerate inaccuracy in state information
4. Is robust: degrades predictably amongst failures
5. Works well even without a helpful real-time kernel
6. Has a conservation law all applications can live with
7. Leads to no forward/backward compatibility problems
8. Is scalable and costs nearly nothing when not used
9. If not lock-free, never needs more than 2-3 locks
10. Is simple enough even I can explain it with 1 slide

Incomplete List of References

- Books and conference proceedings
  - *Proceedings of Euromicros, IEEE RTSS, and RTAS*
  - *A Practitioner's Handbook for RMA*, by M. H. Klein, *et al*
  - *Real-Time Systems,* by J. W. S. Liu, Prentice Hall
  - *Hard Real-Time Computing Systems*, by G. C. Buttazzo
  - *Real-Time Concepts for Embedded Systems,* by Q. Li & C. Yao
  - *Deadline Scheduling for Real-Time Systems*, by J. Stankovic
  - *Inside Windows 2000*, by D. A. Solomon and M. Russinovich
  - *Understanding the Linux Kernel,* by D. P. Bovet and M. Cesati
- Miscellaneous articles
  - "An open environment for real-time applications," by Z. Deng, *et al. Real-Time Systems Journal*, May 1999
  - "An open real-time environment for parallel and distributed systems," by T. W. Kuo, *et al. Proceedings of ICDCS*, 2000

Discussions

Hermann Kopetz

TU Wien
Austria

TU Wien

Introduction

Outline

- When is a Computer System "Real-Time"
- Classification of Real-Time Systems
- Temporal Requirements
- What is a "System Architecture"?
- Technology Trends
- Architecture Based Design

When is a Computer System 'Real-Time'?

A *real-time computer system* is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time, when these results are produced.

The point in time when a result has to be produced is called a *deadline*.

Deadlines are dictated by the environment of the real-time computer system.

Today's *real-time computer systems* are *distributed* and require a temporal coordination of the actions of the nodes.

Some Definitions

If the result has utility even after the deadline, we call the deadline *soft.* Systems with soft deadlines are not the focus of these lectures.

If the result has no utility after the deadline has passed, the deadline is called *firm.*

If a catastrophe could result if a strict deadline is missed, the deadline is called *hard.*

A real-time computer system that has to meet at least one hard deadline is called a *hard real-time system.*

Hard- and soft real-time system design are fundamentally different.

Classification of RT Systems

On the basis of the external requirements

- ♦ Hard Real-Time versus Soft Real Time
- ♦ Fail-Safe versus Fail-operational

On the basis of the implementation

- ♦ Guaranteed Timeliness versus Best Effort
- ♦ Resource Adequacy-- yes or no?
- ♦ Event Triggered versus Time Triggered

Hard Real Time versus Soft Real Time

Characteristic	Hard Real Time	Soft Real Time
Response time	hard	soft
Pacing	environment	computer
Peak-Load Perform.	predictable	degraded
Error Detection	system	user
Safety	critical	non-critical
Redundancy	active	standby
Time Granularity	millisecond	second
Data Files	small/medium	large
Data Integrity	short term	long term

Fail-Safe versus Fail-Operational

A system is *fail-safe* if there is a safe state in the environment that can be reached in case of a system failure, e.g., ABS, train signaling system.
In a fail-safe application the computer has to have a high *error detection coverage.*
Fail safeness is a characteristic of the application, not the computer system.

A system is *fail operational,* if no safe state can be reached in case of a system failure,e.g., a flight control system aboard an airplane.
In fail-operational applications the computer system has to provide a minimum level of service, even after the occurrence of a fault.

Guaranteed Timeliness versus Best Effort

A system implementation provides *guaranteed timeliness* if, within the specified load- and fault-hypothesis, the temporal correctness can be substantiated by analytical arguments.

A system implementation is *best effort,* if such an analytical argument for the temporal correctness cannot be made.

The temporal verification of best effort systems relies on probabilistic arguments, even within the specified load- and fault hypothesis.

Hard real-time systems should be based on guaranteed timeliness.

Resource Adequacy

If a system has to provide guaranteed timeliness, there must be sufficient computational resources to handle the specified peak load and fault scenario.

In the past, there have been many applications where resource adequacy has been considered *too expensive*. The decreasing cost of hardware makes the implementation of resource adequate designs economically viable. In hard real-time applications, *there is no alternative to resource adequate designs*.

Predictability in Rare Event Situations

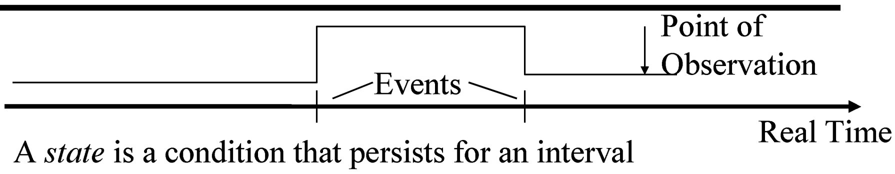
A *rare event* is an important event that occurs very infrequently during the lifetime of a system, e.g., the rupture of a pipe in a nuclear reactor.

A rare event can give rise to many correlated service requests (e.g., an alarm shower).

In a number of applications, the utility of a system depends on the predictable performance in rare event scenarios, e.g. flight control system

In most cases, workload testing will not cover the rare event scenario.

State versus Event



A *state* is a condition that persists for an interval of real time, i.e., along a section of the timeline

An *event* is an occurrence at an instant.

State information informs about the attributes of states at the point of observation (itself an event).

Event information informs about the difference in the attributes of the states immediately before and after the occurrence of the event and an estimation of the point in time of event occurrence

Only the consequences of an event can be observed.

Time Triggered (TT) vs. Event Triggered (ET)

A Real-Time system is *Time Triggered* (TT) if the control signals, such as

- ♦ sending and receiving of messages
- ♦ recognition of an external state change

are derived solely from the progression of a (global) notion of time.

A Real-Time system is *Event Triggered* (ET) if the control signals are derived solely from the occurrence of events, e.g.,

- ♦ termination of a task
- ♦ reception of a message
- ♦ an external interrupt

Temporal Requirements

Temporal accuracy of real-time data: the data elements that are displayed to the operator must be *temporally accurate*.

Maximum response time: The maximum real-time interval between a stimulus and the response must be known and bounded.

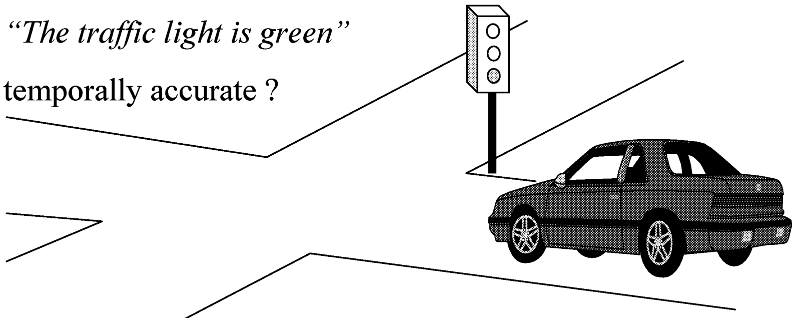
Predictability: The temporal behavior must be predictable, even in a rare event scenario.

Validity of Real-Time Information

How long is the observation:

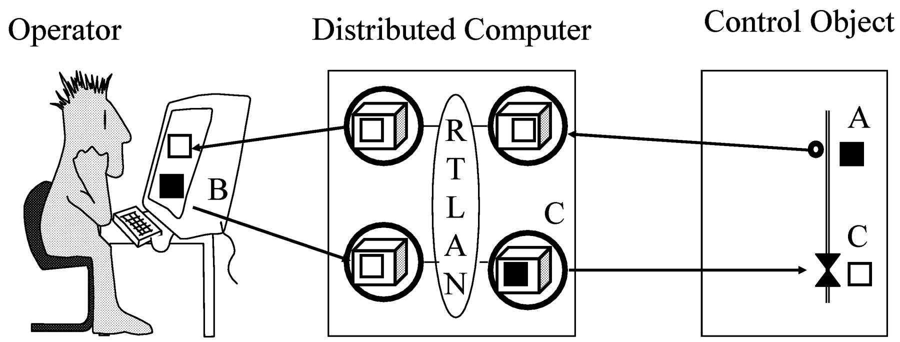
"The traffic light is green"

temporally accurate ?



Temporal parameters are associated with real-time data.

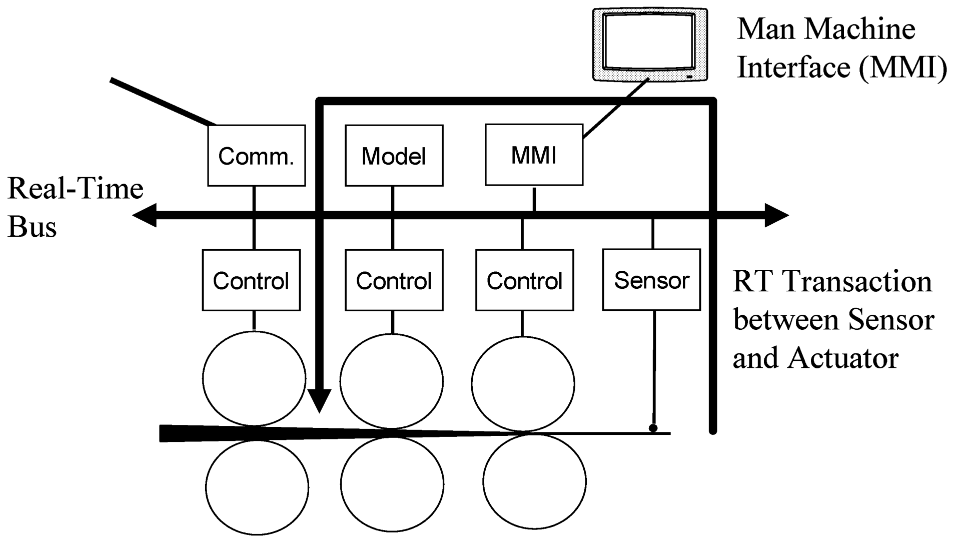
RT Entities, RT Images and RT Objects



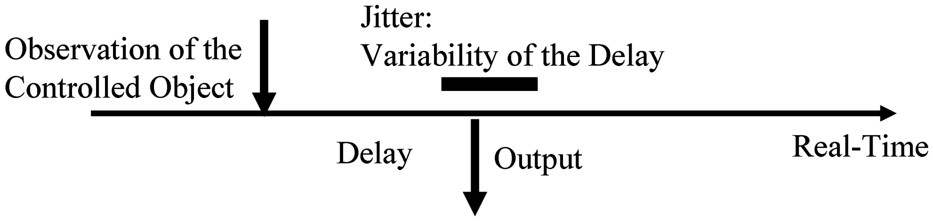
■ RT Entity □ RT Image ⧉ RT Object

A: Measured Value of Flow
B: Setpoint for Flow C: Intended Valve Position

Real-Time Transaction



Jitter at the Application Level



Observation of the Controlled Object

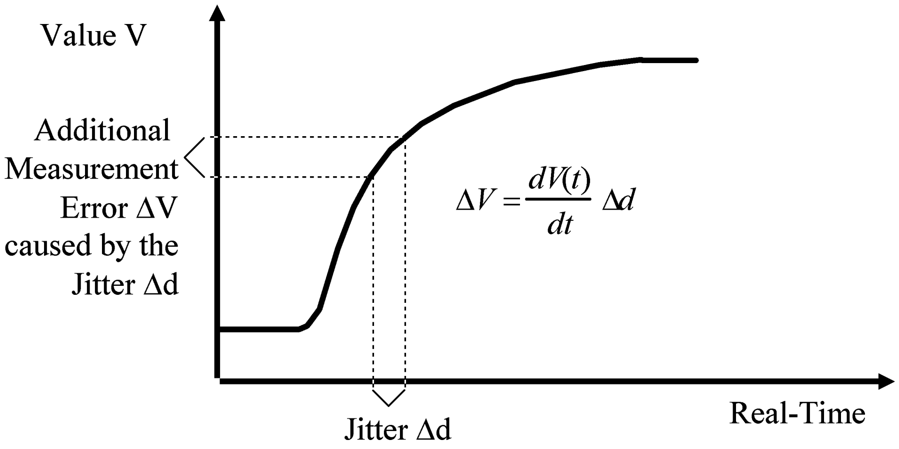
Jitter:
Variability of the Delay

Delay

Output

Real-Time

The Effect of Jitter: Measurement Error



Value V

Additional Measurement Error ΔV caused by the Jitter Δd

$$\Delta V = \frac{dV(t)}{dt} \Delta d$$

Jitter Δd

Real-Time

Jitter in Control Loops causes a degradation of control quality.

Protocol Execution Time in ET Systems

The execution time of an event-triggered protocol between two tasks depends on:

- ♦ the scheduling decisions by the operating system of the sender
- ♦ the buffer management of the sender
- ♦ the data link protocol
- ♦ the media access strategy
- ♦ the buffer management at the receiver
- ♦ the scheduling strategy at the receiver.

We call the maximum protocol execution time d_{max} and the minimum protocol execution time d_{min}.

ET Systems: Jitter at Critical Instant

A *critical instant* is a point in time, when all hosts in the ECUs try to send a message simultaneously. There is no phase control possible in ET system.

The message at the lowest priority level must wait until all higher priority messages have been sent (assume that all message have the same length).

Protocol execution time at critical instant (n ECUs):

$$d_{max} = n\ d_{trans}$$

Protocol execution time if the channel is free:

$$dmin = d_{trans}$$

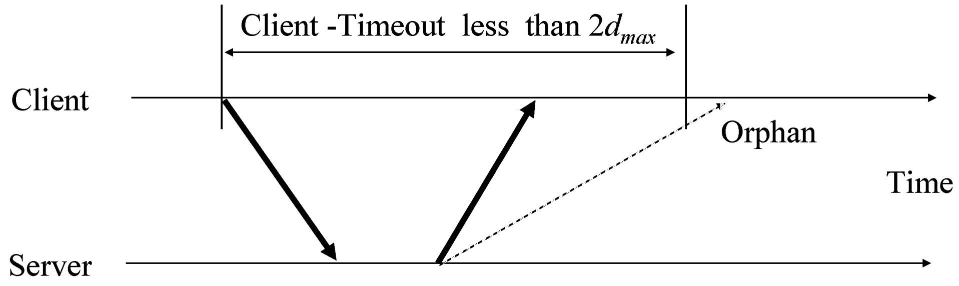
Jitter of the lowest priority message:

$$Jitter = (n-1)\ d_{trans}$$

The jitter depends on the number of ECUs in the system.

The Effect of Jitter: Orphans

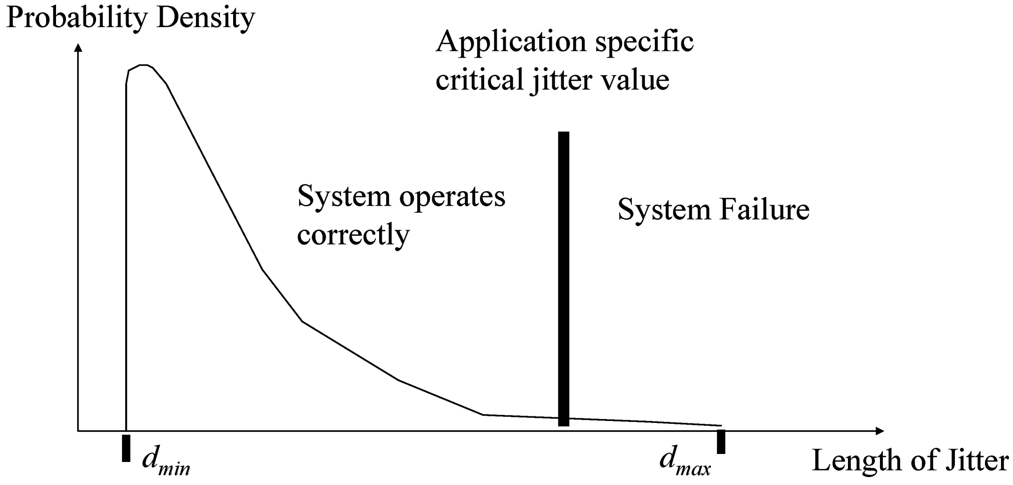
Request Response Transaction between a Client and a Server:



How large is d_{max} ?

It is not contained in the interface specification, available at the sub-supplier.

Probability of “Long” Jitter in ET Systems



Most of the time, the system will operate correctly.

Logical versus Temporal Control

The control scheme determines at what point in time the execution of a selected action will start. In RT systems it is necessary to distinguish between:

- *Logical Control* is concerned with the control flow within a task to realize the specified data transformation
- *Temporal Control* is concerned with the point in time when a task is to be started or when it has to be preempted by a more urgent task. Temporal control is closely related to scheduling

Rolling Mill Example

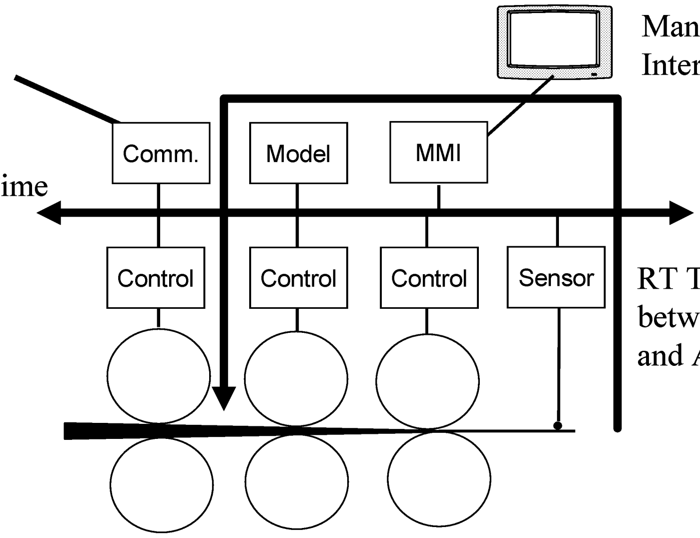
An alarm monitoring component should raise an alarm

WHEN $p_1 < p_2$ THEN raise alarm;

At a first glance, this specification of an alarm condition looks reasonable. A further analysis leads to the following open questions:

- What is the maximum$_1$ and p_2 ?
- At what points in time must the alarm condition be evaluated?

Real-Time Transaction



Man Machine Interface (MMI)

Real-Time Bus

RT Transaction between Sensor and Actuator

What is a Technical System Architecture?

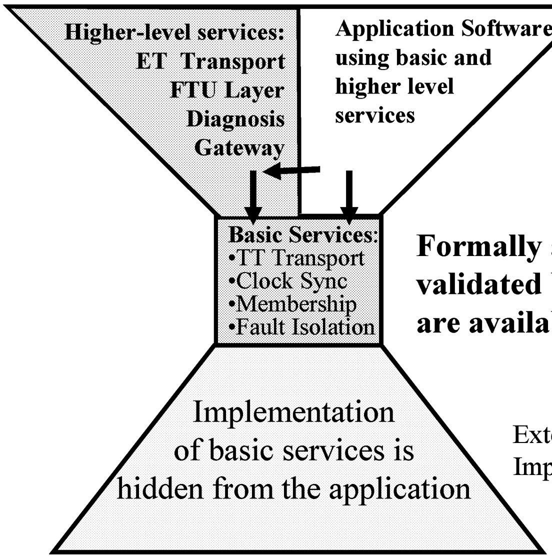
A *technical system architecture* is a framework for the construction of a system that constrains an implementation in such a way that the ensuing system is understandable, maintainable, extensible, and can be built cost-effectively.

Technical System Architecture (II)

- ♦ *Architectural style:* An architecture must provide rules and guidelines for the partitioning of a system into subsystems and for the design of the interactions among the subsystems.
- ♦ *Composability:* An architecture must provide a framework for the systematic construction of a system out of subsystems (components).
- ♦ *Property Match*: Components must comply with the *architectural style* to avoid a *property mismatch* at the component interfaces.
- ♦ *Elegance:* An architecture must *constrain* an implementation in such a way that the ensuing system is understandable, maintainable, extensible, and can be built cost-effectively--in other words, it is *elegant.*

Architecture Design is Interface Design

Waist-line Architecture (e.g., TTA)



Higher-level services:
ET Transport
FTU Layer
Diagnosis
Gateway

Application Software using basic and higher level services

New high-level services to ease application development

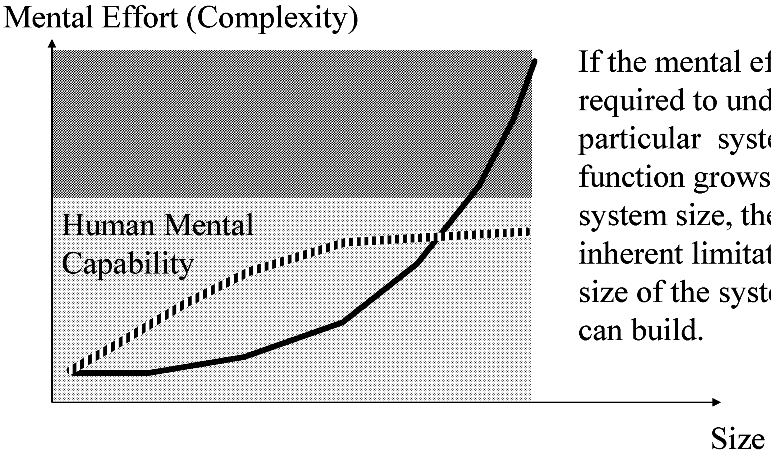
Basic Services:
•TT Transport
•Clock Sync
•Membership
•Fault Isolation

Formally analyzed and validated basic services are available and stable

Implementation of basic services is hidden from the application

Extend the range of Implementation choices

Size versus Mental Effort to Understand



If the mental effort required to understand a particular system function grows with the system size, there is an inherent limitation to the size of the systems we can build.

Complexity and Size

- ♦ Large systems can only be built if the effort required to understand the system operation, i.e, the complexity of the system, remains under control as the system grows.
- ♦ The effort to understand any particular system function should remain constant, and should be *independent* of the system size.
- ♦ A large system contains many more different functions than a small system.
- ♦ The effort needed to understand all functions of a large system grows with the system size.

The design effort must be guided by technical system architecture.

Software Size and Failures

Project Outcome (percent)	Project size in k lines source <10	 <100	 <1000	 >5000
Cancelled	3	7	13	24
Late by >12 months	1	10	12	18
Late by > 6 months	9	24	35	37
Approximately on time	72	53	37	20
Earlier than expected	15	6	3	1
Avg. Schedule(months)				
Planned Schedule	6	12	18	24
Actual Schedule	6	16	24	36
Difference	0	4	6	12

Source: Patterns of large software Systems, Computer, March 1995, p. 86

Grand Challenge

The grand challenge in the design of large distributed real-time systems is the management of the ever-growing complexity

- Partition the system into nearly autonomous subsystems that can be understood, developed and tested independently
- Separate the interactions among the subsystems (system level) from the processing functions within the subsystems (subsystem level).

What is needed is a *distributed architecture* for embedded real-time systems that supports a two level design methodology

- **system level design and the**
- **subsystem level design**

The Interoperability Problem–A Scenario

- A system integrator--e.g., an automotive company--specifies an Drive-by-Wire architecture and generates an **interface specification** for the subsystem (node) suppliers.
- The different nodes are developed by different sub-suppliers with respect to this interface specification.
- The acceptance tests of the nodes, as performed versus the interface specification, are o.k..
- The automotive company integrates the nodes into the system context and observes sporadic failures.

What is the cause of these failures and who is responsible for correcting these failures?

System Integration

Given a set of subsystems that are integrated to form a system.

A subsystem is a self-contained system, including hardware, software, and its own autonomous control, that provides a specified service to its environment.

At the system level, we distinguish between two types of services:

- **Prior Services**: The sum of the services that are provided by the isolated subsystems prior to the integration.
- **Emerging Services**: New services that come into existence by the integration

Principles of Composability

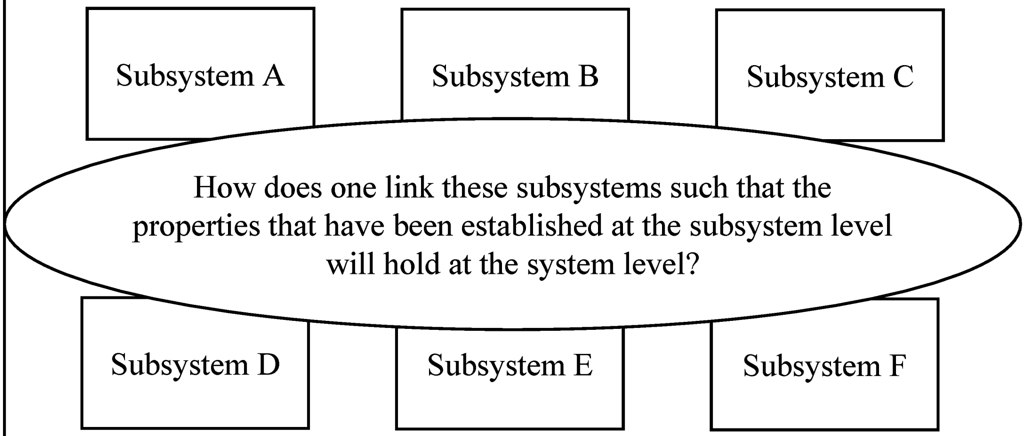
The principles of composability are:

- ♦ **Independent development of components**--relates to the architecture support for a *two-level design process*:
  - • architecture design with precise interface specification
  - • component design, w.r.t these interface specification
- ♦ **Stability of prior services**--relates the components that are used in different system contexts (Solution of the reuse problem).
- ♦ **Constructive integration of components**--component integration should be linear and not circular--relates to the communication system.

Furthermore, if fault-tolerance is to be implemented by component replication, the component must be ***replica deterministic***.

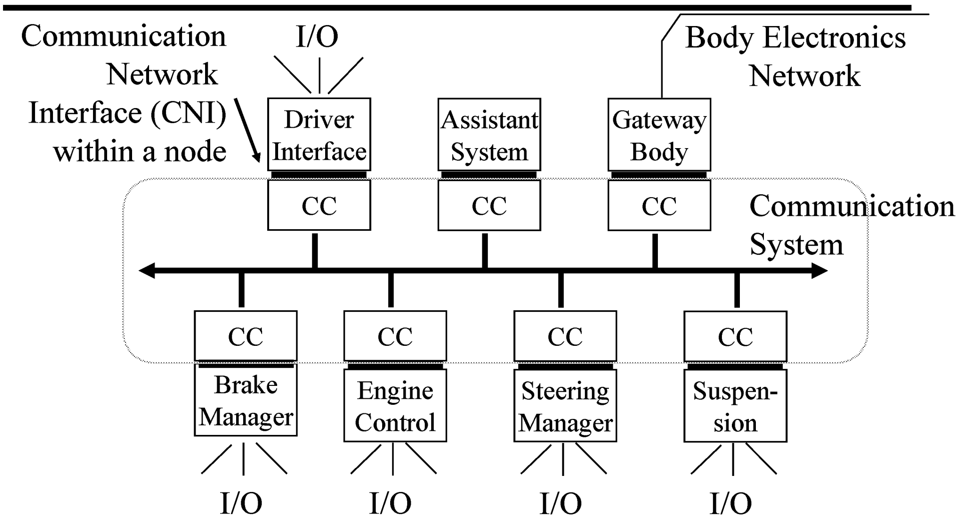
Composability: The Role of the Network

"The network is the only mechanism suitable to enforce and manage real-time operation of distributed systems" (Caro 1998) .



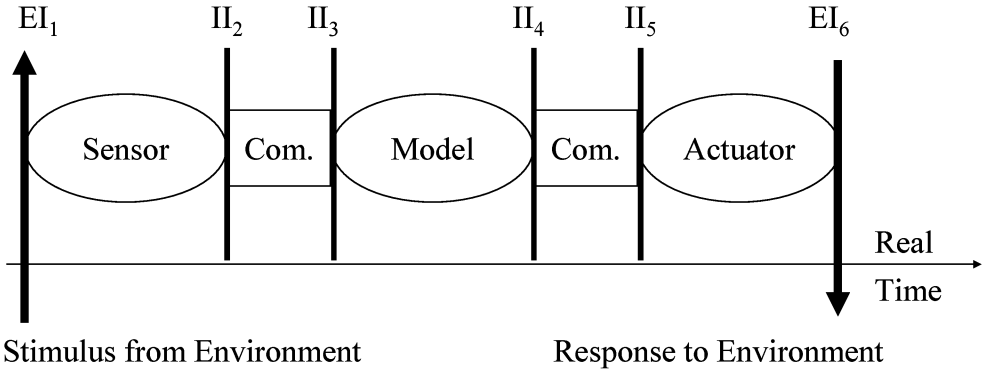
How does one link these subsystems such that the properties that have been established at the subsystem level will hold at the system level?

Example of a Distributed System



CC: Communication Controller

Real-Time Transaction



What Is Required?

An architecture based approach to real-time system design that supports:

- ♦ Composability--*to build systems constructively out of prevalidated components.*
- ♦ Two-level design methodology--*to be able to separate architecture design from component design.*
- ♦ Generic fault-tolerance--*to implement fault-tolerance without* ***any*** *change in the application software.*
- ♦ Flexible configuration--*to support the reuse of existing components*
- ♦ Volume market real-time applications--*efficient use of hardware is a real concern.*

Architecture Design *is* Interface Design

A good interface within a distributed real-time system

- ♦ is *precisely* specified in the value domain and in the temporal domain,
- ♦ provides the relevant abstractions of the interfacing subsystems and hides the irrelevant details,
- ♦ leads to minimal coupling between the interfacing subsystems,
- ♦ limits error propagation across the interface,
- ♦ Conforms to the established architectural style

and thus introduces ***structure*** into a system.

What is a “Component”?

In our context, a *component* is complete computer system that is time aware. It consists of

- ♦ The hardware
- ♦ The system and application software
- ♦ The internal state

The component interacts with its environment by the exchange of messages via interfaces.

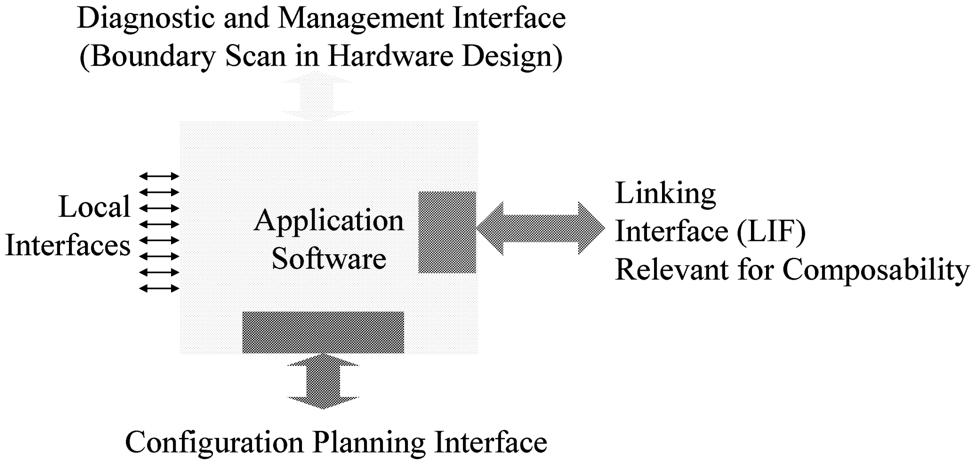
What is a *software* component?

Closed Component vs. Open Component

- ♦ **Closed Component:** Contains no local interface to the *real world*, but can contain local interfaces to other closed components.
  Semi-closed if it is time-aware.
- ♦ **Open Component:** Contains an interface to the *real world.*
  Semi-open if no control signals are accepted from the real-world (e.g., a sampling system).

The real world has an unbounded number of properties.

Interfaces of a Node--Messages


Diagnostic and Management Interface (Boundary Scan in Hardware Design)

Local Interfaces

Application Software

Linking Interface (LIF) Relevant for Composability

Configuration Planning Interface

What is an Interface?

Interface: A common boundary between two or more interacting systems.

Linking Interface (LIF): An interface between two or more existing systems that is introduced in order to realize a system-of-system (SOS) with new emergent services.

Property Mismatch: A disagreement among interfacing ports in one or more of their properties.

Boundary Line (BL): An interface between at least two ports with matching properties.

Interface System (IS): A new system with at least two ports that is introduced between ports of the interfacing legacy systems in order to resolve property mismatches among the legacy systems, to coordinate multicast communication, and/or to introduce emerging services.

Some Further Definitions

System: An autonomous entity that is capable to interact with its environment and is aware of the progression of time.

Port: A point of interaction between a system and its environment. Every port can be characterized by a set of properties

Output action: The production by a system at an output port of a single value change at an instant or of a temporally controlled sequence of value changes during an interval.

Message: An aggregate that comprises the entire information content of an output action, i.e., the control information and the data structure.

Behavior: The sequence of output actions of a system across some or all of its ports.

Service: The intended behavior of a system.

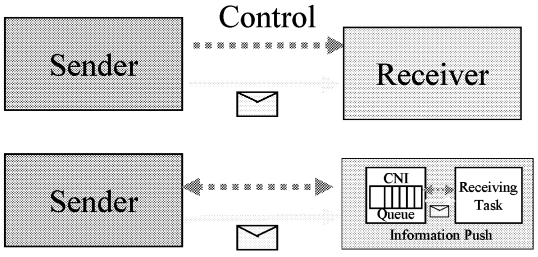
Elementary vs. Composite Interface

Consider a **unidirectional data flow** between two subsystems (e.g., data flow from sensor node to processing node).

We distinguish between:

Elementary Interface:

Composite Interface:



Example: state message in a DPRAM

Queue of event messages

A composite Interface introduces a control dependency between the Sender and Receiver and thus compromises their independence.

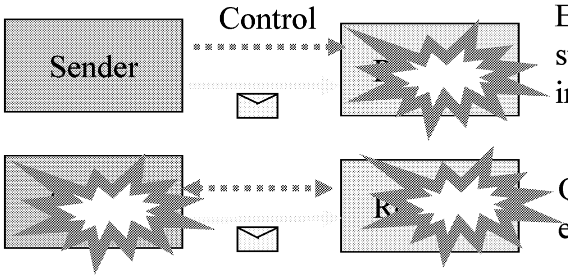
Elementary vs. Composite Interface

Consider a **unidirectional data flow** between two subsystems (e.g., data flow from sensor node to processing node).

We distinguish between:

Elementary Interface:

Composite Interface:



Example: state message in a DPRAM

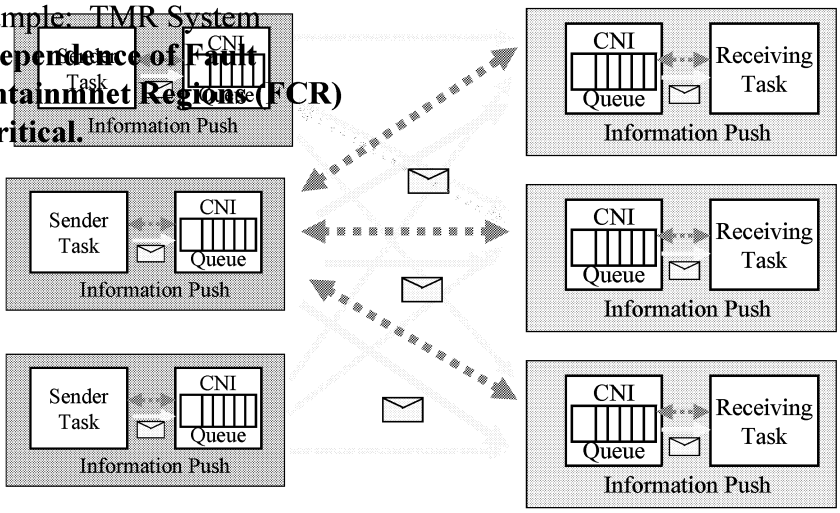
Queue of event messages

A composite Interface introduces a control dependency between the Sender and Receiver and thus compromises their independence.

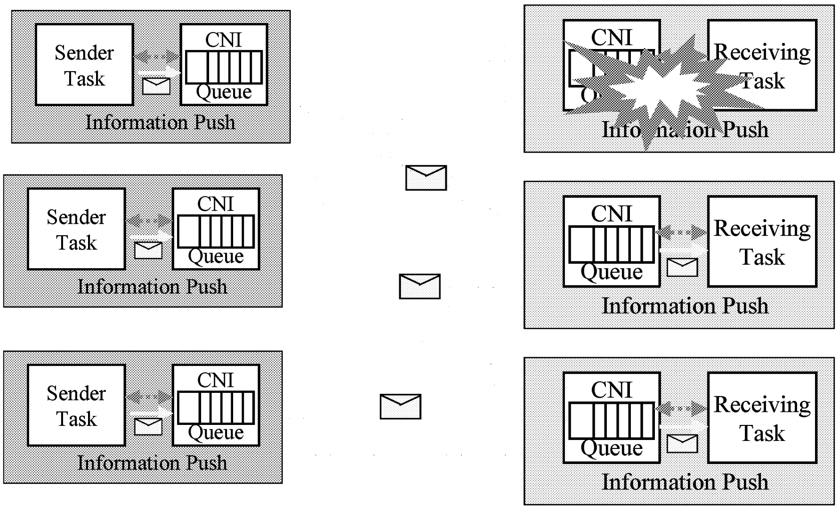
ET in Multicast Communication

Example: TMR System

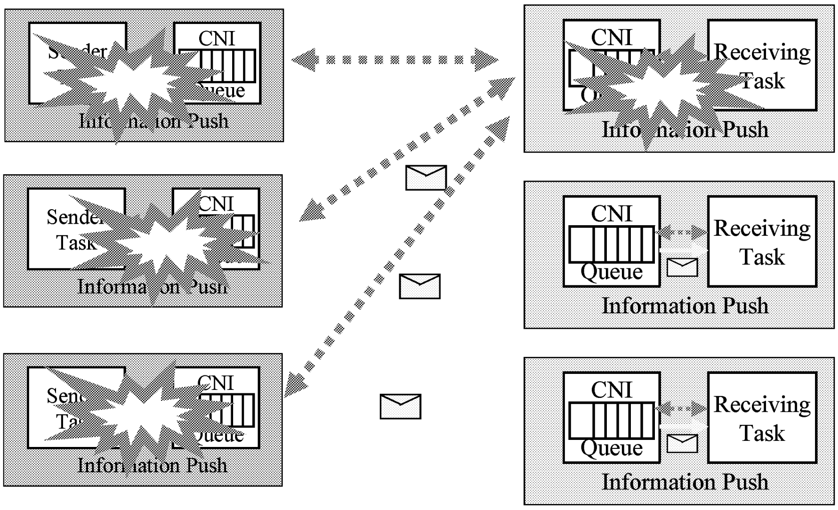
Independence of Fault Containmnet Regions (FCR) is critical.



ET in Multicast Communication



ET in Multicast Communication



Technology Trend to Distributed Systems

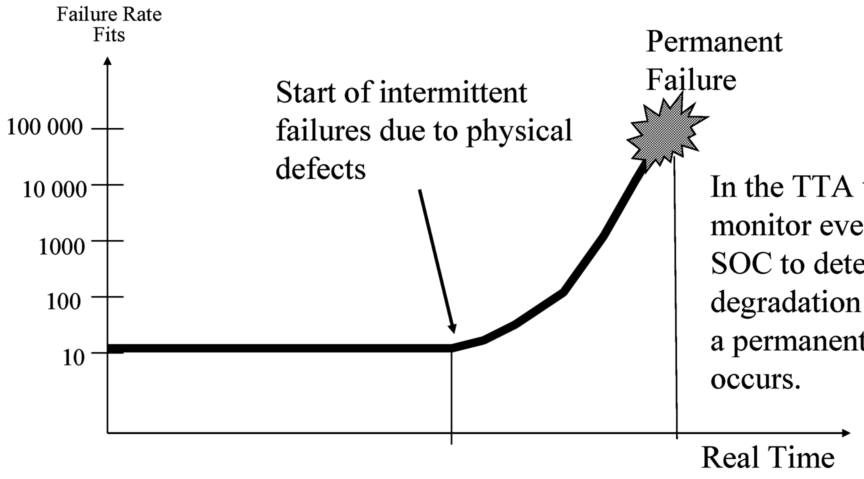
- ♦ **System on a Chip (SOC) is the components**: A complete computer system, including, CPU, Memory, I/O, Communication Controller, Operating Systems, and Application Software can be implemented on a single silicon die: e.g., Motorola "Golden Oak"
- ♦ **Smart Sensors**: Sensing Element, signal processing, calibration, diagnosis, communication control on a single die.
- ♦ **On-Chip Oscillators** for low-cost nodes: cheap, but imprecise
- ♦ **COTS**: Commercial off the shelf components comprising hardware and software
- ♦ **Integrated Fault Tolerance**: to mask faults, e.g. SEU (single event upsets)

Future Failure Modes of SOC

- ♦ The expected further shrinkage of the feature size will cause new failure modes[1] such as, for example:
  - • Transient multi-bit failures caused by a single fault event
  - • Intermittent failures of the interconnect that can affect different functions on the die simultaneously
- ♦ It is expected that in the future the rate for permanent failures will remain unchanged, but that the rate for transient and **intermittent** failures will increase.
- ♦ The assumption that a *fail-silent node* that hosts two independent FCUs can be implemented on a single die is ***not sustainable*** in future high-dependability applications**.**

[1] **Source**: C. Constantinescu, Impact of Deep Submicron Technology on Dependability of VLSI circuits, Proceedings of the IEEE DSN 2002, Washington D.C., p. 205-209

The Issue: Intermittent Failures of Chips



Start of intermittent failures due to physical defects

Permanent Failure

In the TTA we can monitor every single SOC to detect a degradation before a permanent failure occurs.

Systems on Chip:

Current Semiconductor Technology makes it possible to design a selfcontained 32 bit computer system, including 1 Mbyte of memory, Network Access and I/O on a single die, e.g., the Motorola Golden Oak Chip.

Development cost of an SOC: > 10 Mio US $

Production cost: < 10 US $

Number of devices sold (in millions)	0.1	1	10
Development overhead (per device)	1 000%	100%	10 %

Economics of Silicon

Silicon real-estate requirements (today, i.e. in the year 2000):

ARMcore 32 bit CPU: 1 mm^2

Infineon 256 Mbit DRAM: < 100 mm^2 : □
320 kbyte of DRAM: 1 mm^2

- Marginal Production Costs of 1 mm^2 of silicon is in the order of 10 US cent (Cost at silicon foundry TSMC)
- *Cost of packaging, testing, pins, power-supply significant and often dominant.*
- Marginal production costs of 100 mm^2 silicon chip order of 10 US $.

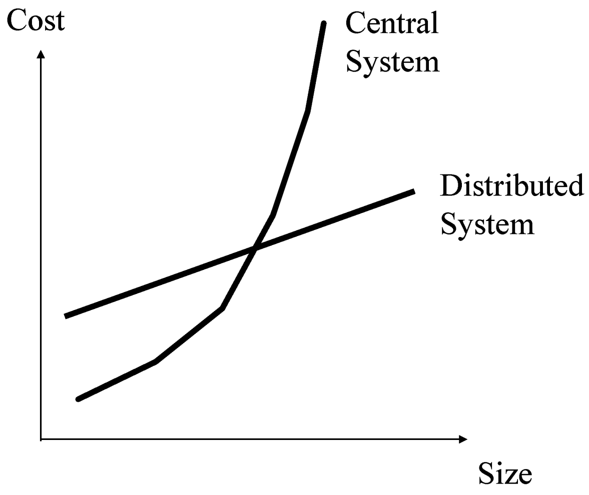
One men minute of work buys how many megabytes of RAM?

Silicon Die Costs

According to Hennesy and Patterson (p.60), the

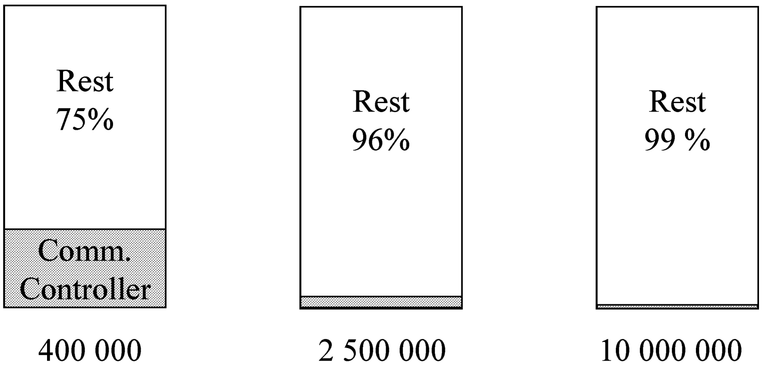
Cost = f (Die-area^3)

On the other hand, a distributed system requires more packaging and an additional die area for the communication controller.

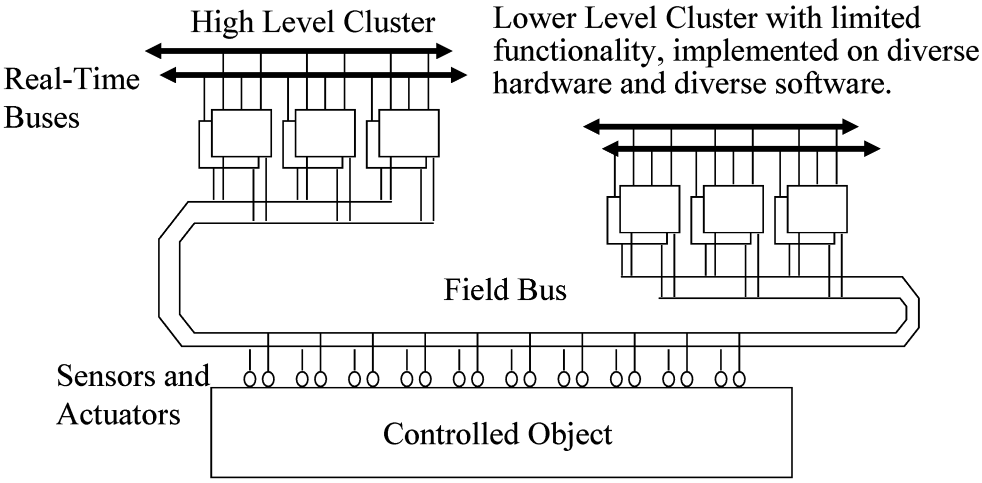


Cost of Distribution

The cost of distribution decreases with increasing VLSI integration. Let us assume, the onchip implementation of a complex communication controller consumes the equivalent of 100 000 transistors:



Multilevel Architecture


High Level Cluster
Lower Level Cluster with limited functionality, implemented on diverse hardware and diverse software.
Real-Time Buses
Field Bus
Sensors and Actuators
Controlled Object

Summary: A Good Distributed Architecture

- ♦ provides a framework and guidelines for the composition of a system out of *nearly autonomous* components (subsystems) without the occurrence of property mismatches.
- ♦ defines an architectural style.
- ♦ specifies the type of interactions among the components across well-defined and small interfaces. It thus builds structure by weak inter-component coupling and strong intra-component coupling.
- ♦ provides interfaces that are flexible enough to support the intended functions, but rigid enough to act as error containment boundaries.
- ♦ is based on already familiar orthogonal concepts that are used recursively.
- ♦ is scalable without limits.

Benefits of a Distributed Architecture

- ♦ Encapsulated software/hardware solutions with fully specified interfaces in the value domain and the temporal domain can be used in differing contexts.
- ♦ Minimal glue code, since property mismatches are avoided by adherence to the established architectural style.
- ♦ Fault containment and fault tolerance can become part of the architecture (outside the application).
- ♦ Composability reduces design efforts, validation efforts and time-to-market.
- ♦ Platform electronics becomes possible.

TU Wien

Time and Order

Outline

- ♦ Time and Clocks
- ♦ Time Measurement
- ♦ Dense Time versus Sparse Time
- ♦ Internal Clock Synchronization
- ♦ External Clock Syncrhonization

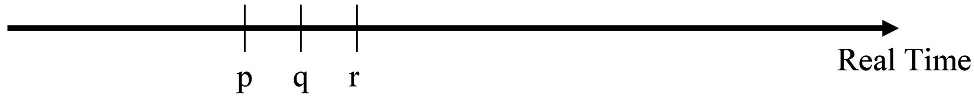
Instants are Temporally Ordered

The continuum of real time can be modeled by a directed timeline consisting of an infinite set {T} of *instants* with the following properties:

(i) {T} is an ordered set, i.e., if p and q are any two instants, then either (1) p is simultaneous with q or (2) p precedes q or (3) q precedes p and these relations are mutually exclusive. We call the order of instants on the timeline the *temporal order*.

(ii) {T} is a dense set. This means that, if p≠r, there is at least one q between p and r.

The order of instants on the timeline is called the *temporal order*.



Durations and Events

A section of the time line is called a *duration.*

An *event* is a happening at an instant of time.

An event does not have a duration. If two events occur at an identical instant, then the two events are said to occur simultaneously. Instants are totally ordered; however, events are only partially ordered, since simultaneous events are not in the order relation. Events can be totally ordered if another criterion is introduced to order events that occur simultaneously, e.g., in a distributed computer system the node numbers where the events occurred can be used to order events that occur simultaneously at different nodes.

Causal Order

Reichenbach [Rei57,p.145] defined *causality* by a mark method without reference to time: "If event e1 is a cause of event e2, then a small variation (a mark) in e1 is associated with small variation in e2, whereas small variations in e2 are not necessarily associated with small variations in e1."

Example: Suppose there are two events e1 and e2:

e1 Somebody enters a room.

e2 The telephone starts to ring.

Consider the following two cases

(i) e2 occurs after e1

(ii) e1 occurs after e2

Alarm Analysis

A *primary alarm event* leads to a shower of *secondary alarm events* (*alarm shower*).

If the (partial) temporal order between alarm events has been established, it is possible to exclude an alarm event that *definitely occurred later* than other alarm events from being the primary event. A precise global time-base helps to determine the event set that is in this *definitely-occurred-later* relation.

Delivery order of messages in computer networks:

Temporal?
Causal?
Consistent?

Clocks and Timestamps

A *clock* is a device that contains a counter and increments this counter periodically according to some law of physics (*microticks*).

Let us assume there exists an external observer with an atomic clock that has a granularity that is much smaller than the duration of any intervals of interest. We call such a clock a *reference clock z* -- Precision, e.g., one femto second (10^{-15}sec)!

The *granularity* of a clock c is the number of microticks of the reference clock between any two consecutive microticks of c.

Given a clock and an event, a *timestamp* of the event is the state of clock immediately after the event occurrence, denoted by
clock (event).

We assume that relativistic effects can be neglected.

Clock Drift

Clock Drift:

$$drift_i^k = \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k}$$

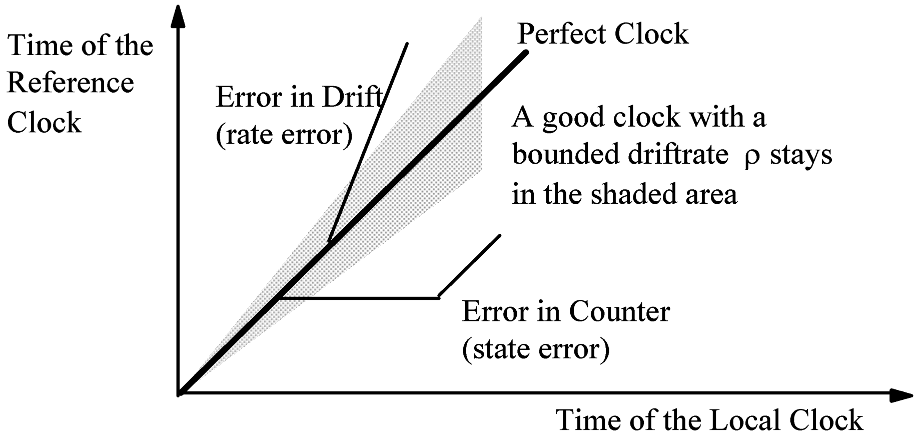
Drift Rate:

$$\rho_i^k = \left| \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k} - 1 \right|$$

Perfect clock has drift rate of 0
Real clocks have drift rates from 10^{-2} to 10^{-8}

n^k nominal number of ticks of the reference clock within a granule of clock *k*.

Failure Modes of a Clock



Time of the Reference Clock

Perfect Clock

Error in Drift (rate error)

A good clock with a bounded driftrate ρ stays in the shaded area

Error in Counter (state error)

Time of the Local Clock

Precision

Offset between two clocks j,k at tick i:

$$offset_i^{jk} = \mid z(microtick_i^j) - z(microtick_i^k) \mid$$

Given an ensemble of clocks {1, 2, . . . , n}, the maximum offset between any two clocks of the ensemble is called the precision of the ensemble at microtick i:

$$\Pi_i = \underset{\forall 1 \le j,k \le n}{Max} \{offset_i^{jk}\}$$

The process of mutual resynchronization of an ensemble of clocks in order to maintain a bounded precision is called *internal synchronization.*

Accuracy

The offset of clock k with respect to the reference clock z at tick i is called the *Accuracy*. The maximum offset over all ticks i that are of interest is called the accuracy of clock k. The accuracy denotes the maximum offset of a given clock from the external time reference during the time interval of interest.

This process of resynchronization of a clock with the reference clock is called *external synchronization.*

If all clocks of an ensemble are externally synchronized with an accuracy A, then the ensemble is also internally synchronized with a precision of at most 2A. The opposite is not true.

Time Standards

International Atomic Time (TAI): TAI is a physical time standard that defines the second as the duration of 9 192 631 770 periods of the radiation of a specified transition of the cesium atom 133. TAI is a chronoscopic timescale, i.e., a timescale without any discontinuities. It defines the epoch, the origin of time measurement, as January 1, 1958 at 00:00:00 hours, and continuously increases the counter as time progresses.

Universal Time Coordinated (UTC): UTC is an astronomical time standard that is the basis for the time on the "wall clock". In 1972 it was internationally agreed that the duration of the second should conform to the TAI standard, but that the number of seconds in an hour will have to be occasionally modified by inserting a leap second into UTC to maintain synchrony between the wall-clock time and the astronomical phenomena, like day and night.

A Problem with the Leap Second

Software Engineering Notes of March 1996 (p.16) reports on a problem that occurred when a leap second was added at midnight on New Year's Eve 1995. The leap second was added, but the date inadvertently advanced to Jan. 2. The synchronization of AP radio broadcast network depends on the official time signal, and this glitch affected their operation for several hours until the problem was corrected.

Making corrections at midnight is obviously risky:

(1) The day increments to January 1, 1996, 00:00:00.

(2) You reset the clock to 23:59:59, back one second.

(3) The clock continues running.

(4) The day changes again, and it's suddenly, January 2, 1996, 00:00:00.

No wonder they had problems.

Global Time

If there is a single reference clock available, all time measurements can be performed by this single clock that acts as a common “global” time.

In a distributed system clocks in order to generate a common notion of time, a “global time” in the distributed system.

However, such a global time is an *abstract notion* that can only be approximated by the clocks in the nodes.

It is possible to select a subset of the microticks of each local clock k for the generation of the local implementation of a global notion of time. We call such a selected local microtick a *macrotick* (or a tick) of the global time.

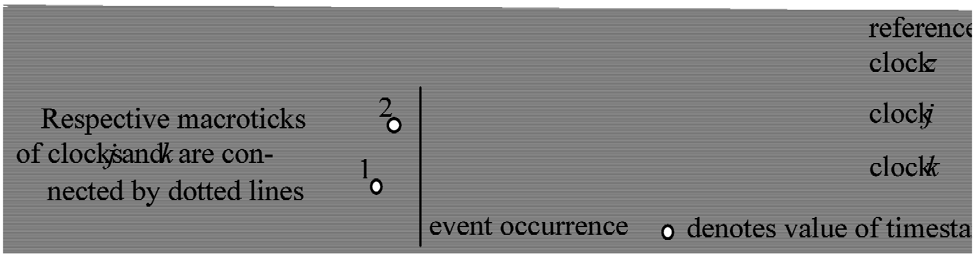
If no global time-base is available, then

- there are n independent local time references and the timestamps can only be related if they originate from the same clock.
- Interval measurements between events observed at different nodes are limited by the end-to-end delay jitter.
- the delay jitter of (ET) communication system determines the jitter in the control loops--this may be unacceptable for many real-time control applications.
- State estimation is very difficult, since the precise point in time of measurement of a process variable is not known.

Requirements of a Global Time Base

- Chronoscopic behavior, i.e. no discontinuities, even at the points of resynchronization
- Known precision
- High dependability
- Metric of the physical second

Reasonableness Condition



reference clock z

clock j

clock k

Respective macroticks of clocks j and k are connected by dotted lines

event occurrence

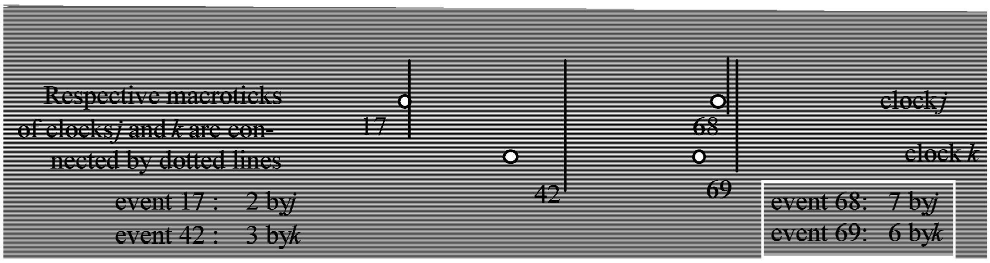
○ denotes value of timesta

The global time t is called reasonable, if all local implementations of the global time satisfy the following reasonableness condition for the global granularity g of a macrotick:

$$g > \Pi$$

This reasonableness condition ensures that the synchronization error is bounded to less than one macrogranule, i.e., the duration between two macroticks.

One Tick Difference: What Does it Mean?



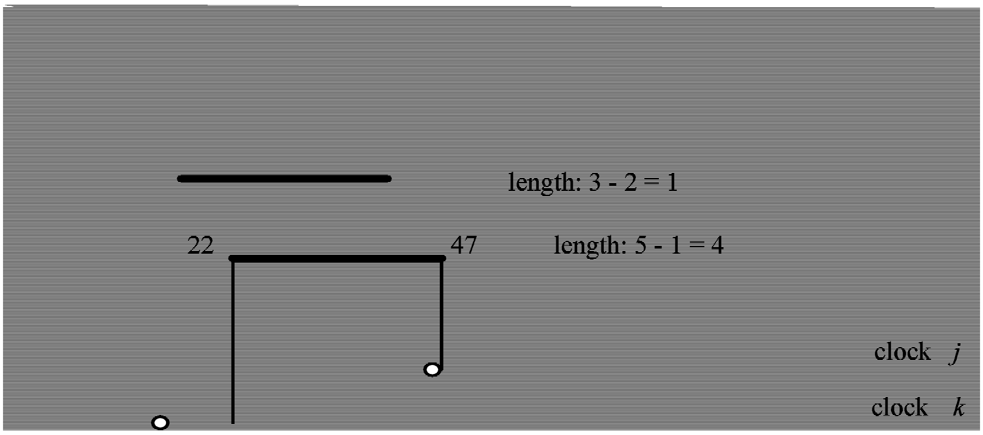
Respective macroticks of clocks *j* and *k* are connected by dotted lines

event 17 : 2 by *j*
event 42 : 3 by *k*

event 68: 7 by *j*
event 69: 6 by *k*

Because of the accumulation of the synchronization error and the digitalization error, it is not possible to reconstruct the temporal order of two events from the knowledge that the global timestamps differ by one.

Interval Measurement

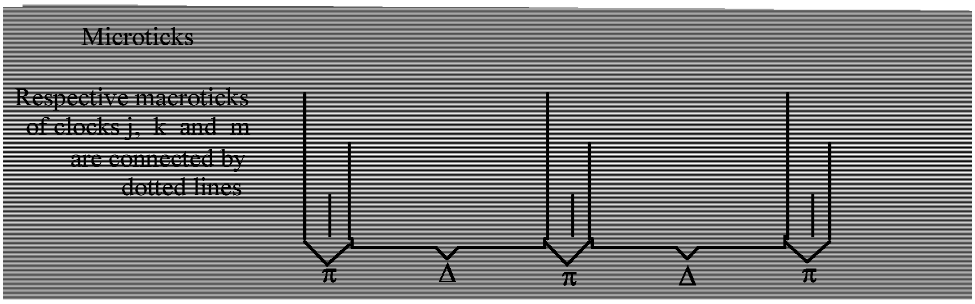


It follows: $(d_{obs} - 2g) < d_{true} < (d_{obs} + 2g)$

π/Δ Precedence

Given a set of events {E} and two durations π and Δ where π<<Δ, such that for any two elements e_i and e_j of this set the following condition holds:

$$[\,|z(e_i) - z(e_j)| \le \pi\,] \vee [\,|z(e_i) - z(e_j)| > \Delta\,]$$


Microticks
Respective macroticks of clocks j, k and m are connected by dotted lines
π
Δ
π
Δ
π

Fundamental Limits to Time Measurement

Given a distributed system with a reasonable global timebase with granularity g. Then the following fundamental limits to time measurement must be observed:

- ♦ If a single event is observed by two nodes, there is always the possibility that the timestamps will differ by one tick
- ♦ Let us assume that d_{obs} is the observed duration of an interval. Then the true duration d_{true} is

  $$(d_{obs} - 2g) < d_{true} <(d_{obs} + 2g)$$
- ♦ The temporal order of events can only be recovered, if the observed time difference $d_{obs} \geq 2g$
- ♦ The temporal order of events can always be recovered, if the event set is 0/3g precedent.

Dense Time

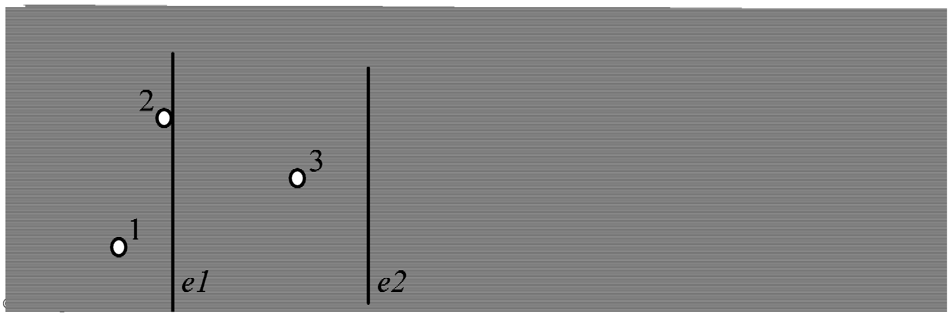
A timebase is dense if events can occur at any point of the timeline.

Consequences of these fundamental limits of time measurement in distributed systems:

- ♦ If a single event occurring on a dense timebase is observed by two nodes of the distributed system (e.g., to achieve redundancy in the observations), then an explicit agreement protocol is needed to establish a consistent view of the temporal point of event occurrence.
- ♦ If two events occur on a dense timebase, then it is impossible to always recover the temporal order of the events if they occur within an interval of 3g.

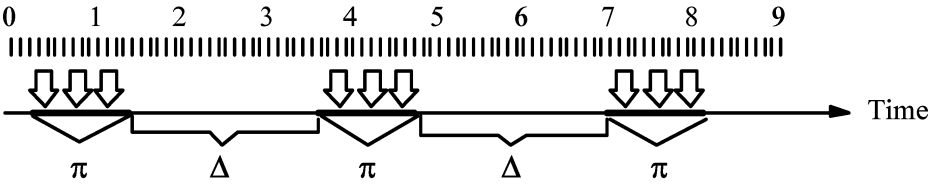
Inconsistent Order on Dense Time Base

Event e1 is observed by node j at time 2 and by node m at time 1, while e2 is only observed by node k that reports its observation "e2 occurred at 3" to node j and node m. Node j calculates a timestamp difference of one and concludes that the events occurred at about the same time and cannot be ordered. Node m calculates a timestamp difference of 2 and concludes that e1 has definitely occurred before e2.



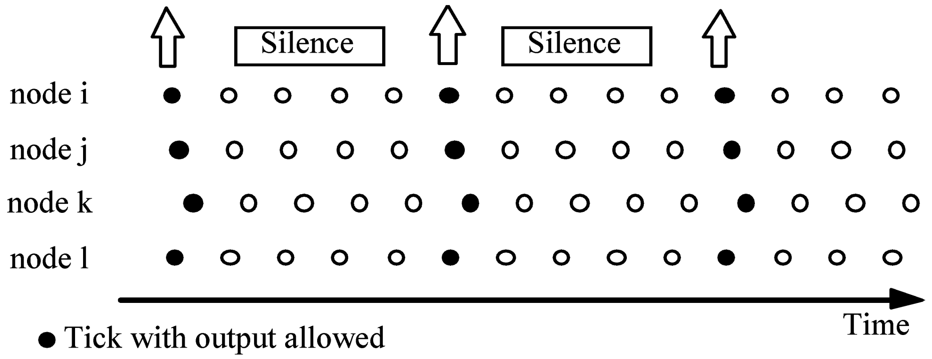
Sparse Time Base

If the occurrence of events is restricted to some active intervals with duration π with an interval of silence of duration Δ between any two active intervals, then we call the timebase π/Δ-sparse, or sparse for short.



Events ⇩ are only allowed to occur at subintervals of the timeline

Space/Time Lattice



● Tick with output allowed

○ Tick with output not allowed

Time and State

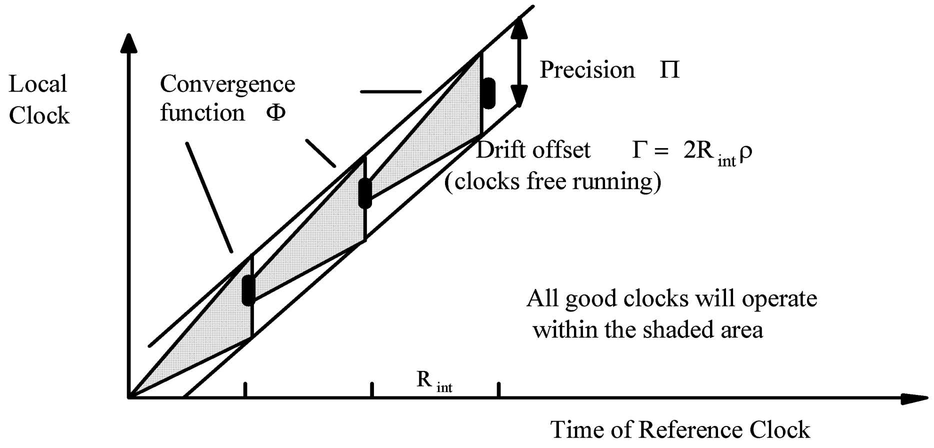
In *abstract system theory (Mesarovic, p.45)*, the notion of *state* is introduced in order to separate the *past* from the *future*:

"The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other word, the state enables a "decoupling" of the past from the present and future. The state embodies all past history of a system. Knowing the state "supplants" knowledge of the past. Apparently, for this role to be meaningful, the notion of past and future must be relevant for the system considered."

A precise concept of time is a prerequisite for a precise concept of state.

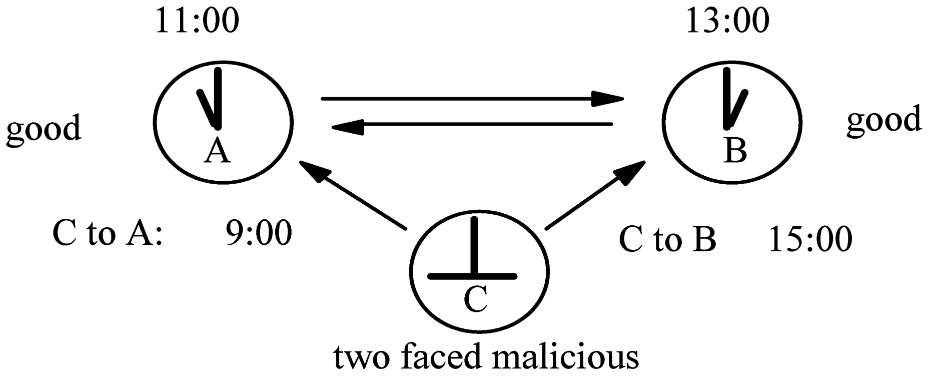
The Synchronization Condition

$\Phi + \Gamma \le \Pi$



All good clocks will operate within the shaded area

Malicious (Byzantine) Clocks



Total Number N of clocks must be $N \ge (3k + 1)$, where k is the number of malicious (Byzantine) faults.

Central Master Algorithm

A unique node, the central master, periodically sends its time counter in synchronization messages to all other nodes, the slave nodes. As soon as a slave receives a new time value from the master, the slave records the state of its local time counter as the time of message arrival. The difference between the master's time contained in the synchronization message and the recorded slave's time of message arrival, corrected by the latency of the message transport, is a measure of deviation of the two clocks. The slave then corrects its clock by this deviation to bring it into agreement with the master's clock.

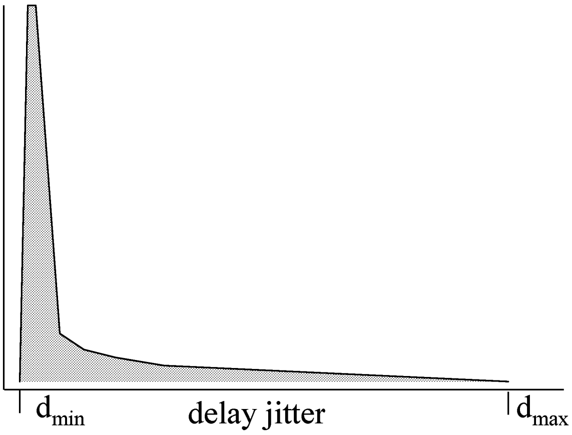
Precision of central Master Algorithm:

$$\Pi_{central} = \varepsilon + \Gamma$$

Delay Jitter

The difference d_{max} - d_{min} is called the *delay jitter* ε.

In standard OSI Protocols (with time redundant transmissions) the typical protocol execution time distribution is as depicted:



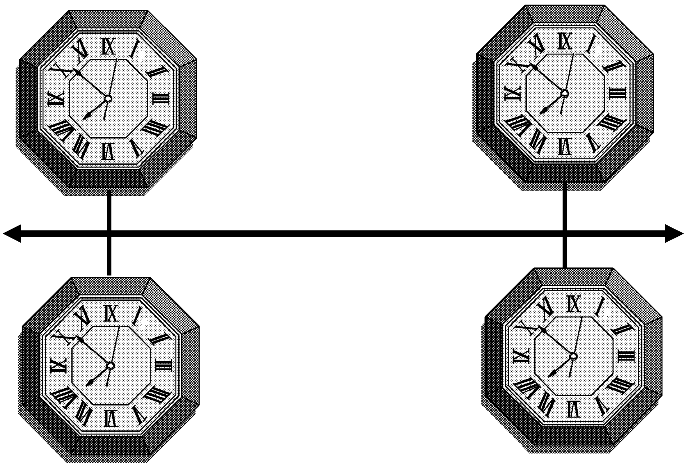
Distributed Clock Synchronization

Typically, distributed fault-tolerant clock resynchronization proceeds in three distinct phases.

- ♦ Every node acquires knowledge about the state of the global time counters in all other nodes by message exchanges among the nodes.
- ♦ Every node analyzes the collected information to detect errors and executes the convergence function to calculate a correction value for the node's local global time counter.
- ♦ The local time counter of the node is adjusted by the calculated correction value.

The algorithms differ in the way in which they collect the time values from the other nodes, in the type of convergence function used, and in the way in which the correction value is applied to the time counter.

How well can we synchronize clocks?



Convergence Function

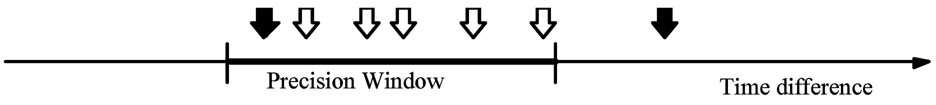
Examples:

- Average Algorithm
- Fault-Tolerant Average (FTA)
- Fault-Tolerant Midpoint
- Interactive Consistency Algorithms

Fault-Tolerant Algorithm

Every node measures the time differences between its own clock and all other clocks and rejects the k extreme differences, where k is the number of Byzantine faults that are to be tolerated.

If k=1, then

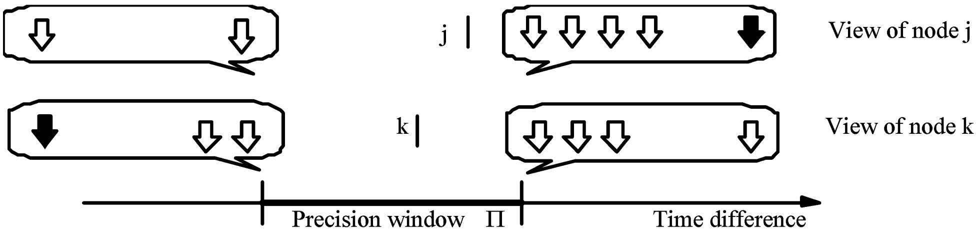


⇩ accepted time value

⬇ rejected time values

Fault Tolerant Average Algorithm

The worst scenario happens, if the Byzantine clock sets its (faulty) time values at different nodes at a different corner of the Precision window:



⇩ good time value

⬇ malicously faulty time value

j   Average value calculated by node j

k   Average value calculated by node k

Precision of the FTA

Convergence Function

$$\Phi(N,k,\varepsilon) = k\Pi / (N-2k) + \varepsilon$$

Precision

$$\Pi(N,k,\varepsilon,\Gamma) = (\varepsilon+\Gamma)\frac{N-2k}{N-3k} = (\varepsilon+\Gamma)\mu(N,k)$$

where *μ (N,k)* is called the *Byzantine error factor* and is tabulated in the following table:

			Number of nodes in the ensemble					
Faults	4	5	6	7	10	15	20	30
1	2	1.5	1.33	1.25	1.14	1.08	1.06	1.03
2				3	1.5	1.22	1.14	1.08
3					4	1.5	1.27	1.22

Interactive Consistency Algorithms

To overcome the problem of the Byzantine error factor, every node sends its view of the ensemble to all other nodes in order that every node has the global view of the situation, i.e., it can find out which node has been cheating.

Every node takes this consistent global view, i.e., the matrix of time vectors, as the basis of the correction factor calculation.

Pro: $\mu(N,k) = 1$

Con: Extra round of communication

Limit to Internal Clock Synchronization

Lundelius and Lynch have shown that in a system with N clocks and a delay jitter ε it is impossible to synchronize clocks better

$$\Delta_{int} = \varepsilon \ (1-1/N)$$

The proof assumes that all clocks have perfect oscillators.

Critical Parameters

What are the critical parameters that determine the quality of the global time base?

- ♦ Drift offset $\Gamma = 2*R_{sync}*\rho$
- ♦ Delay jitter $\varepsilon = d_{max} - d_{min}$
- ♦ The occurrence of Byzantine

The delay jitter is smallest, if the clock synchronization is performed very close to the physical level--by the hardware.

Compared to the delay jitter, the algorithmic effects are small.

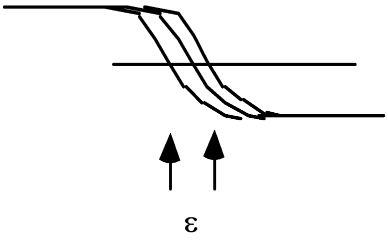
Reading the State of the Remote Clocks

synchronization message assembled and interpreted	approximate range of jitter
at the application software level	500µsec to 5 msec
in the kernel of the operating system	10µsec to 100µsec
in the hardware of the communication controller	1 µsec to 10µsec, or even better

Probabilistic Clock Synchronization:

Measure the time of a request/respond transaction that contains the time value of the partner clock and correct the clock value by half of the transaction duration.

Delay Jitter ε of the Signal Edge



The reading error ε is a fraction (less than one third) of a bitcell

Rate Correction

Precision:

$$\Pi(N,k,\varepsilon,\Gamma) = (\varepsilon + \Gamma)\frac{N-2k}{N-3k} = (\varepsilon + \Gamma)\mu(N,k)$$

Drift offset $\Gamma = 2*R_{sync}*\rho$

We can select a *Master-Rate Clock* and synchronize the rates of all other clocks to the *Master-Rate Clock:*

- Distributed Fault-Tolerant Algorithm for clock state synchronization
- Central Multimaster Algorithm for clock rate synchronization
- Error detection w.r.t. rate interval and speed of change of rate

External Clock Synchronization

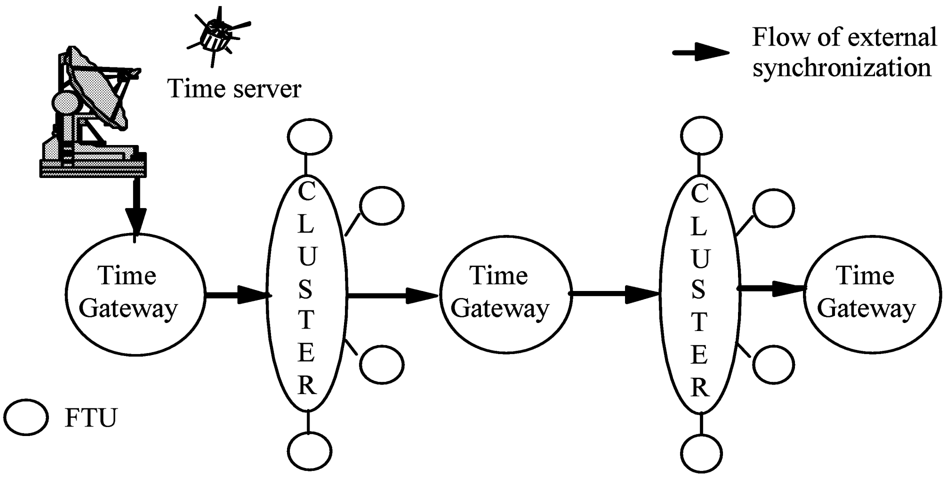
External Clock synchronization is only possible, if the system has access to an external time reference.

In the future, GPS will be an important time source that gives synchronization accuracy in the submicrosecond interval.

External and internal clock synchronization are complementary:

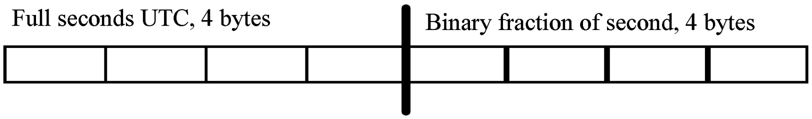
- ♦ Fault tolerant internal synchronization provides high availability and good short term stability.
- ♦ External clock synchronization provides long-term stability, but the availability may be lower.

External Clock Synchronization (2)_



Time Formats

Network Time Protocol:



Full seconds UTC, 4 bytes

Binary fraction of second, 4 bytes

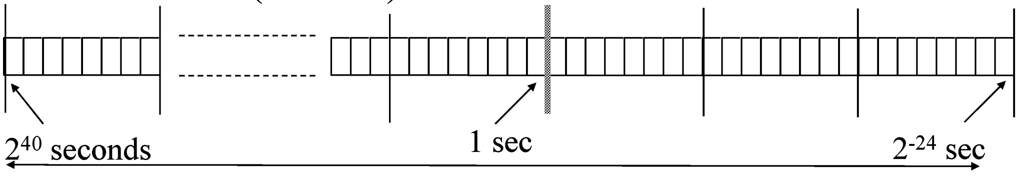
Range up to year 2036, i.e. 136 years wrap around cycle

Uniform Time Format--OMG Standard

Time horizon

Elapsed seconds since January 6, 1980 at 00:00(GPS base).

Time granularity determined by precision of GPS



2^{40} seconds

1 sec

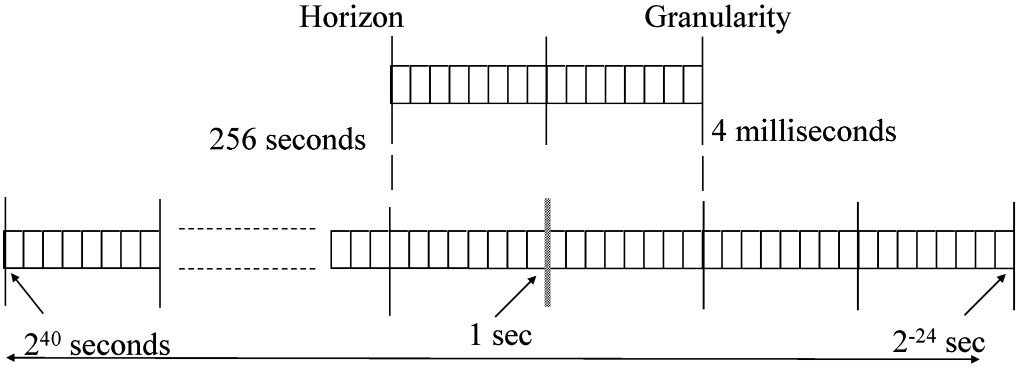
2^{-24} sec

external time format (8 bytes)

Start of epoch: January 6, 1980 at 0:00:00 UTC
Granularity about 60 nanosecond
Horizon 34841 years

Internal Time Format--limited Horizon and Precision

Example of an Internal format (2 byte)



Horizon

Granularity

External time format (8 bytes)

1

Time-Triggered Architecture Overview

H. Kopetz
TU Vienna

2

The Time-Triggered Architecture

Take *Time* from the Problem Domain

And move it into the Solution Domain

What is an Architecture?

An *Architecture* establishes a blueprint and a framework for the design of a class of computing systems that share a common set of characteristics:

- Provides generic mechanisms to the application: *e.g, clock synchronization, membership, diagnostics, fault masking*
- Avoids property mismatches: All **interfaces** adhere to the same architectural style: no glue code needed. *Example: frame formats, byte ordering, protocol structure*
- Supports composability: The **precise specification of stable interfaces** in the domains of time and value makes it possible that subsystems can be developed and tested independently and integrated with manageable effort.
- Supports reuse of subsystems based on **interface specifications**.

Architecture Design is Interface Design.

Time Triggered (TT) vs. Event Triggered (ET)

A Real-Time system is *Event Triggered* (ET) if the control signals are derived from the occurrence of events, e.g.,

- termination of a task
- reception of a message
- an external interrupt

A Real-Time system is *Time Triggered* (TT) if the control signals, such as

- sending and receiving of messages
- recognition of an external state change

are derived from the progression of a (global) time.

In the TTA, all nodes must have access to a fault-tolerant global time base of known precision.

Goals of the TTA

- ♦ To provide a computing infrastructure for safety-critical distributed real-time applications in different application domains.
- ♦ Meet the 10^{-9} failures/hour challenge.
- ♦ Driven by concerns for safety, fault-isolation and certification.
- ♦ Provide generic COTS (commercial of the shelf) hardware and system components for the implementation of safety-critical applications at reasonable costs by taking advantage of mass-produced highly integrated SOCs (System-on-a-chip).
- ♦ Critical algorithms (e.g., clock synchronization, membership) should be validated by all all available means (formal, fault-injection) and should be solidified in silicon.

Time-Triggered Architecture Priorities--in Order

- ♦ **Safety without compromises**
  - •No single point of failure
  - •Formal analysis of critical functions
- ♦ **Composability:**
  - •Fully specified ***operational interfaces*** in the temporal domain and value domain
  - •Building systems out of prevalidated components--Component reuse with established certification argument.
  - •Two-level design methodology
- ♦ **Flexibility**
  - •Flexible reuse of existing components and support for the integration of legacy system

.

Short History of the TTA

1979: The first work on the TTA started at the TU Berlin.

1983: First prototype version of TTA completed with PDP 11s

1987: First clock-synchronization chip available

1989: Second protoype implemention that was subject to extensive fault injection experiments

1993: First publication of TTP/C protocol at FTCS 23

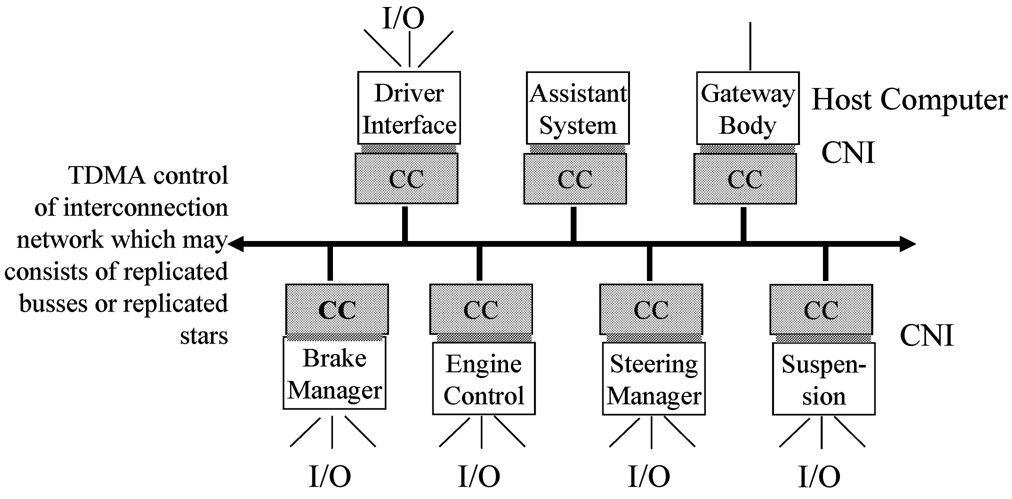
1997: In Cooperation with Daimler-Chrysler the first Brake-by-Wire car was implemented using TTP

1998: The first TTP/C controller chip is produced in the TTA project

1998: TTTech, the spinoff company of TU Vienna, was founded

In total more than 50 Mio $ have been spent on the development of the TT-Technology.

Example of a TTA Cluster



I/O

Driver Interface

Assistant System

Gateway Body

Host Computer

CNI

CC

CC

CC

TDMA control of interconnection network which may consists of replicated busses or replicated stars

CC

CC

CC

CC

CNI

Brake Manager

Engine Control

Steering Manager

Suspen-sion

I/O

I/O

I/O

I/O

CC: Communication Controller CNI: Communication Network Interface

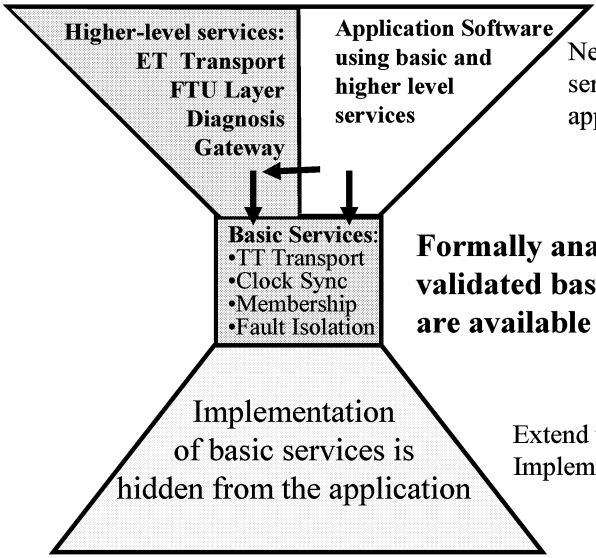
Use of *A Priori* Knowledge in the TTA

The *a priori* knowledge about the behavior is used to improve the Error Detection: It is known a priori when a node has to send a message (*Life sign for membership*).

- ♦ Message Identification: The point in time of message transmission identifies a message (*Reduction of message size*)
- ♦ Flow control: It is known a priori how many messages will arrive in a peak-load scenario (*Resource planning*).

For event-triggered **asynchronous architectures**, there exists an impossibility result: ***'It is impossible to distinguish a slow node from a failed node!'*** This makes the solution to the membership problem and the diagnosis problem difficult.

The TTA is Waist-line Architecture



Higher-level services:
ET Transport
FTU Layer
Diagnosis
Gateway

Application Software using basic and higher level services

New high-level services to ease application development

Basic Services:
•TT Transport
•Clock Sync
•Membership
•Fault Isolation

Formally analyzed and validated basic services are available and stable

Implementation of basic services is hidden from the application

Extend the range of Implementation choices

Basic Services versus High-Level Services

The TTA distinguishes between four basis services and an open ended set of high-level services. The basic services are:

(1) Time Triggered Transport of Messages

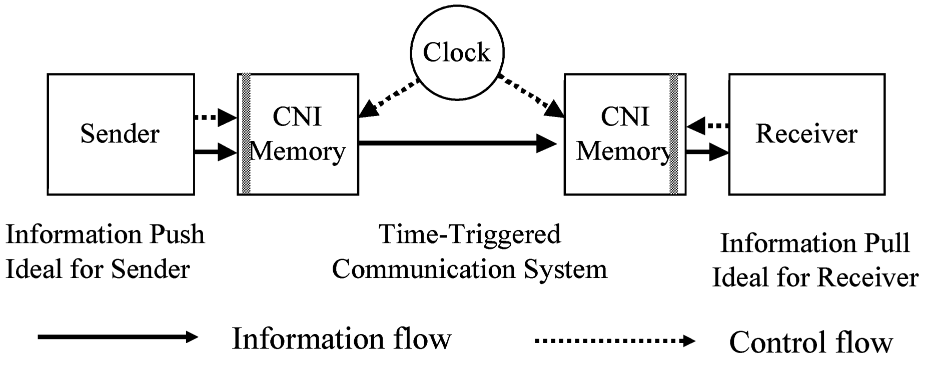
(2) Fault-Tolerant clock synchronization

(3) Membership service

(4) Fault-Isolation Services

The high level services depend on the basic services, while the basic services do not depend on the high-level services!

Basic Service 1: TT Transport--Temporal Firewall



Information Push
Ideal for Sender

Time-Triggered
Communication System

Information Pull
Ideal for Receiver

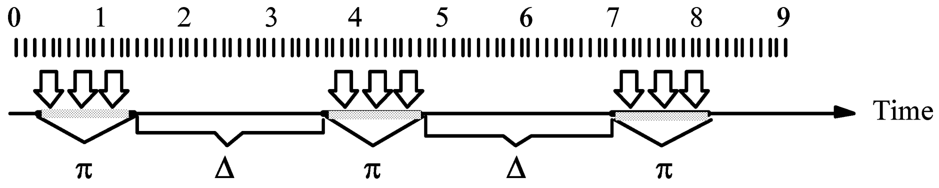
→ Information flow

⋯→ Control flow

Temporal Firewall eliminates control error propagation by design and minimizes coupling between components.

Basic Service 2: Fault-Tolerant Sparse Time Base

If the occurrence of events is restricted to some active intervals with duration π with an interval of silence of duration Δ between any two active intervals, then we call the timebase π/Δ-sparse, or sparse for short.



Events⇩ are only allowed to occur at subintervals of the timeline

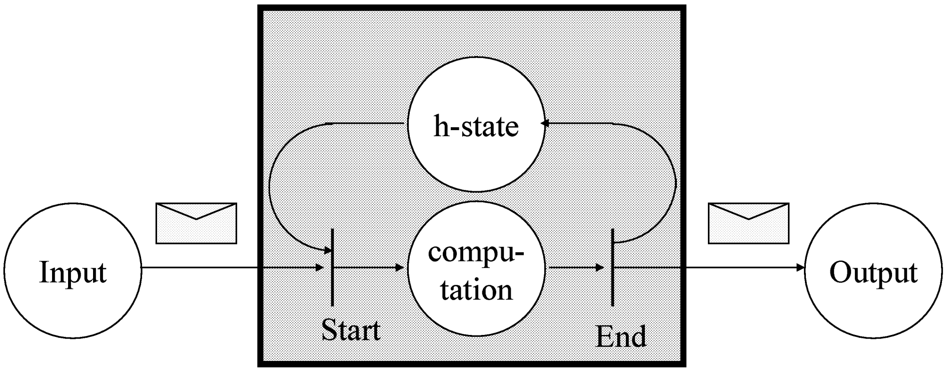
Consistent Notion of State in the TTA

A system-wide consistent notion of a discrete time is a prerequisite for a consistent notion of state, since the notion of *state* is introduced in order to separate the *past* from the *future*:

"The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other word, the state enables a "decoupling" of the past from the present and future. The state embodies all past history of a system. Knowing the state "supplants" knowledge of the past. Apparently, for this role to be meaningful, the notion of past and future must be relevant for the system considered." (Taken from Mesarovic, Abstract System Theory, p.45)

Fault-masking by voting (TMR) requires a consistent notion of state in distributed Fault Containment Regions (FCRs).

Model of a Component–Messages



Message Classification

Attribute	**Explanation**	**Antonym**
valid	A message is *valid* if its checksum and contents are in agreement.	invalid
checked	A message is *checked at source* (or, in short, *checked*) if it passes the output assertion.	not checked
permitted	A message is *permitted* with respect to a receiver if it passes the input assertion of that receiver.	not permitted
timely	A message is *timely* if it is in agreement with the temporal specification	untimely
value-correct	A message is *value-correct* if it is in agreement with the value specification	not value-correct
correct	A message is *correct* if it is both timely and value-correct.	incorrect
insidious	A message is *insidious* if it is permitted but incorrect	not insidious

Basic Service 3: Diagnosis by Membership

The membership service checks continuously, which node is alive and which node has failed. It monitors the correctness of the distributed computing base.

- ♦ The periodic message of each node is interpreted as a life sign of the sender.
- ♦ In order to distinguish between a sender fault and a receiver fault, the view of a third node is considered to be the judge (single fault assumption)
- ♦ Delay of the membership service < 2 TDMA rounds.

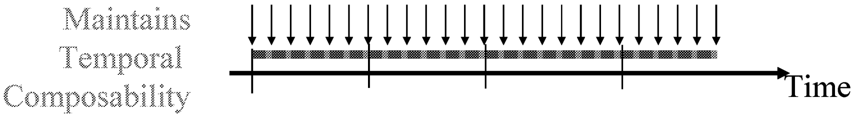
Basic Service 4: Fault Isolation

In the Time-Triggered Architecture FCRs communicate by the exchange of messages:

- ♦ Error Detection in the **Time Domain** is in the responsibility of the architecture. It is performed by independent replicated guardians which are part of the architecture.
- ♦ Error Detection in the **Value Domain** is in the responsibility of the fault-tolerance layer or of the application, supported by **post condition checks** at the guardians.
- ♦ TTP/C contains also a clique avoidance service, based on the membership service.

HL Service: ET Transport

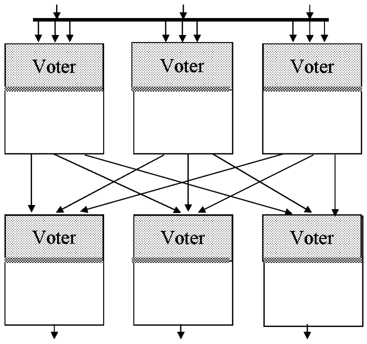
Layered: ET service is implemented on top of a TT protocol
Single time triggered access media access protocol.



The CAN Protocol and the TCP/IP Protocol have been implemented on top of basic TTP/C in order to be able to use legacy software.

HL Service: Fault-Tolerant Unit (FTU) Layer

- ♦ In the TT-OS a fault-tolerant unit (FTU) layer is provided that hides the fault-tolerance mechanism (e.g., voting in TMR systems) from the application.
- ♦ The Communication Network Interface (CNI) - of a fault-tolerant configuration can be configured to be identical for fault-tolerant and non fault tolerant applications.



Precise synchronization of state absolute necessity:
sparse time base, fault isolation

HL Service: Diagnosis

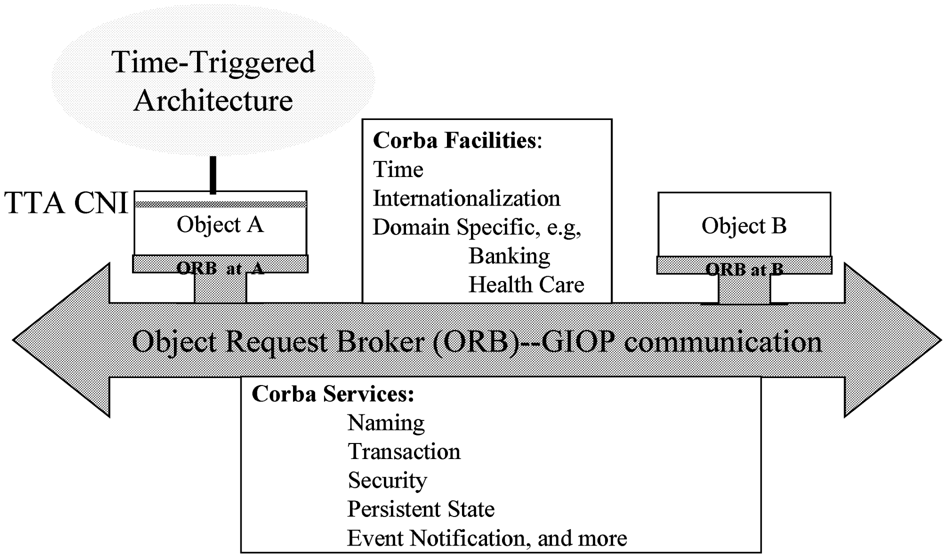
In a safety-critical system every anomaly in the architecture--even if it masked by fault-tolerance mechanisms--must be recorded.

The diagnostic service, which is being implemented as part of the central guardian, continuously monitors

- ♦ The proper operation (sending of messages) of every node
- ♦ The drift rate of all essential clocks
- ♦ Semantic checks on the data (post conditions of messages) if provided.

The diagnostic service is intended to support condition-based maintenance.

HL Service: CORBA Integration



Time-Triggered Architecture

TTA CNI

Object A

ORB at A

Corba Facilities:
Time
Internationalization
Domain Specific, e.g,
Banking
Health Care

Object B

ORB at B

Object Request Broker (ORB)--GIOP communication

Corba Services:
Naming
Transaction
Security
Persistent State
Event Notification, and more

Additional High Level Services

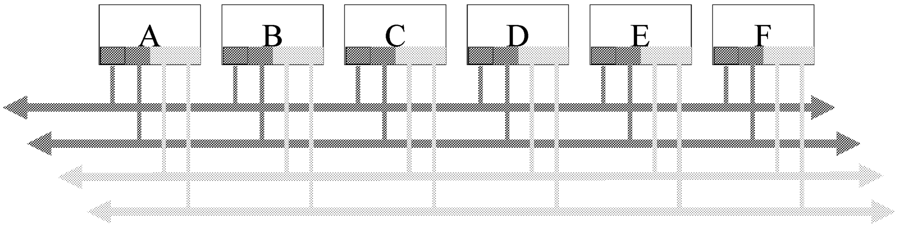
- ♦ Monitoring Service: To observe the contents of messages that are exchanged within a cluster
- ♦ Gateway Service: Support of the construction of multi-cluster Systems
- ♦ LUSTRE integration

TTP/C System Bus - Principle of Operation

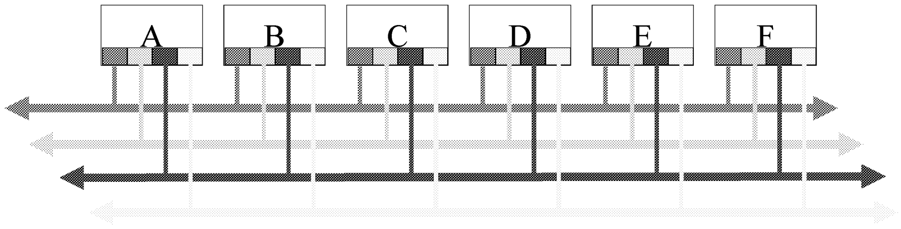
- ♦ TTP generates a fault-tolerant global time-base.
- ♦ Media access is controlled by TDMA, based on this time. ET messages are piggy-packed on the basic TT messages.
- ♦ Information identified by the common knowledge of the send/receive times.
- ♦ Two independent intelligent star couplers provide fault isolation in the temporal domain.
- ♦ Membership service to detect crash/omission (CO) failures. Also used to detect violations of the fault hypothesis.

Four Channel System in TTP/C

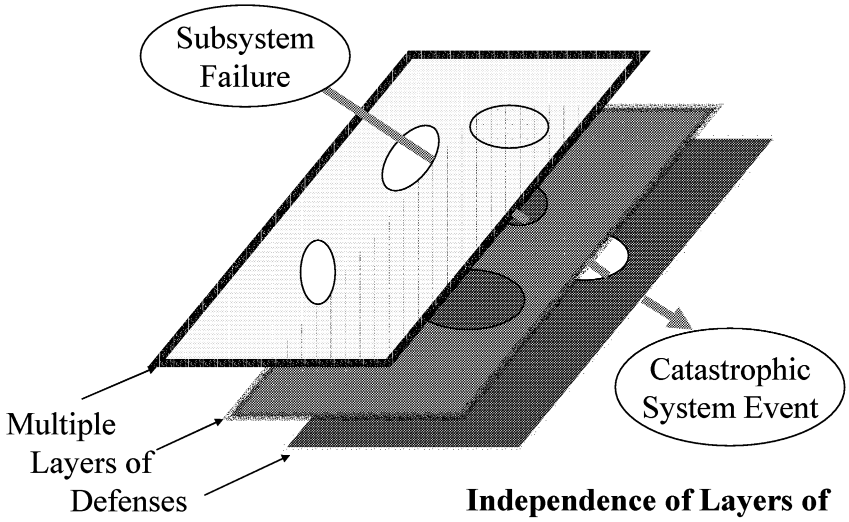
Alternative I: Two Independent Clusters with two channels each



Alternative II: Four Independent Clusters with one channel each



The *Swiss-Cheese* Model of Reason



Subsystem Failure

Catastrophic System Event

Multiple Layers of Defenses

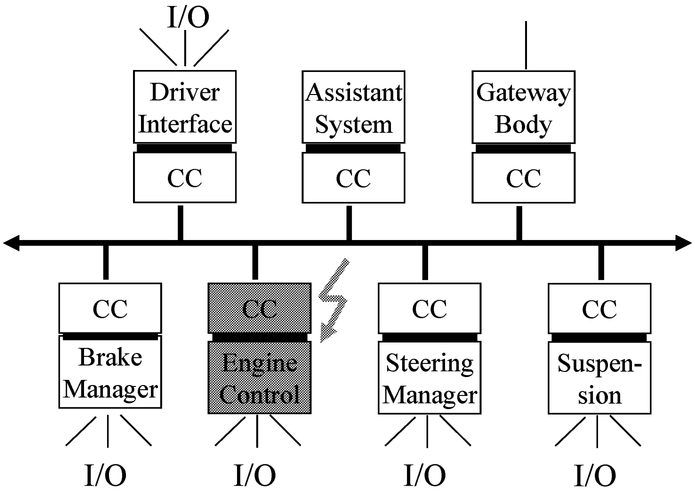
Independence of Layers of Error Detection are important

Fault Containment Region (FCR)

A fault-containment region (FCR) is a set of subsystems that shares one or more common resources that can be affected by a single fault and is assumed to fail **independently** from other FCRs.

- Tolerance w.r.t. spatial proximity faults requires spatial separation of FCRs: **distributed architectures** required.
- The fault hypothesis must specify the failure modes of the FCRs and their associated frequencies.
- Beware of shared resources that compromise the **independence assumption**: common hardware, power supply, oscillator, earthing, single timing source.

Fault Isolation



CC: Communication Controller

Independence of FCRs

There are two basic mechanisms that compromise the independence of FCRs

- ♦ Missing fault isolation
- ♦ Error propagation

The independence of failures of different FCRs is the most critical issue in the design of an ultra-dependable system:

- ♦ Is it justified to assume that a single silicon die contains two independent FCRs?--**NO**
- ♦ Can we assume that the failure modes of a single silicon die are well-behaved (e.g., fail-silent) to the required level of probability?-- **NO**
- ♦ How can we make sure that FCR failures are not correlated, even at a very low level of correlation (e.g., 1 in 1000)?

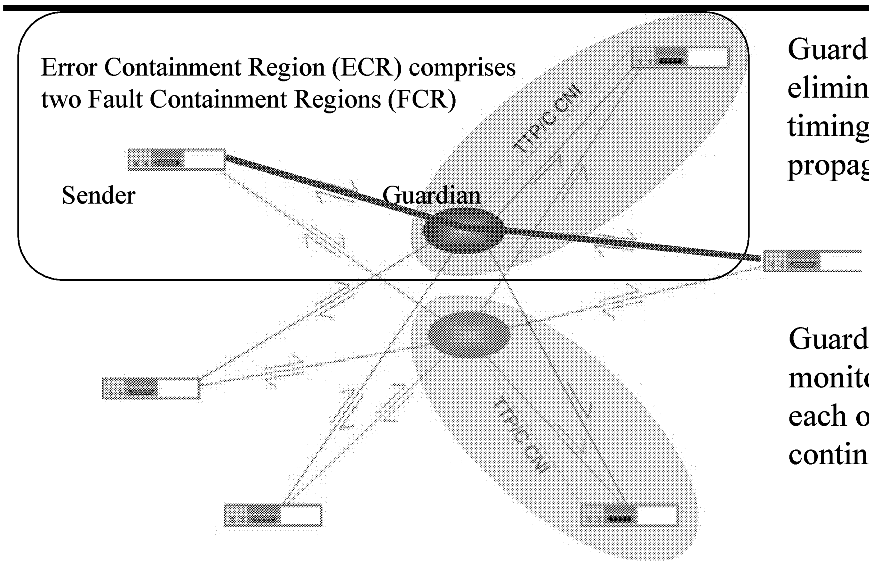
Error Containment Region (ECR)

In a distributed computer system the consequences of a fault, the ensuing error, can propagate outside the originating FCR (Fault Containment Region) by an **erroneous message** of the faulty node to the environment.

- ♦ A propagated error **invalidates** the independence assumption.
- ♦ The error detector must be in a **different FCR** than the faulty unit.
- ♦ Distinguish between architecture-based and application-based error detection
- ♦ Distinguish between error detection in the **time-domain** and error detection in the **value domain**.

Since an Error Containment Region requires at least two independent FCRs, a single die cannot form an Error Containment Region.

Replicated Guardians in the TTA



Error Containment Region (ECR) comprises two Fault Containment Regions (FCR)

Sender

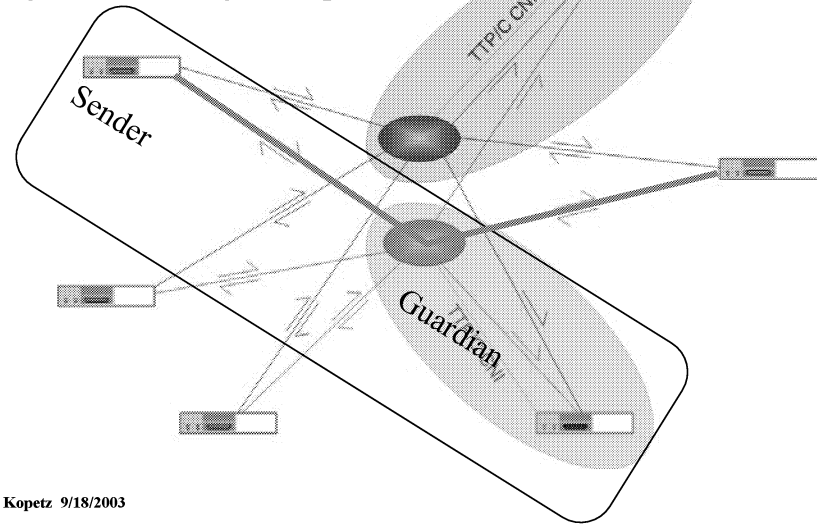
Guardian

Guardian eliminates timing-error propagation

Guardians monitor each other continuously

Replicated Guardians in the TTA

The guardian cannot modify the contents of a message, since it is signature protected

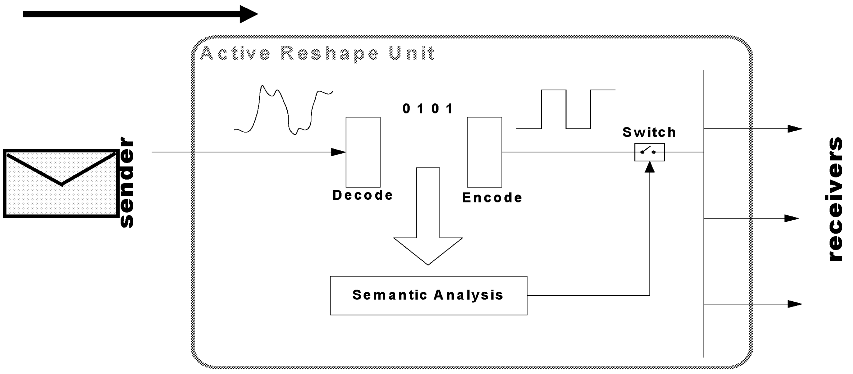


Sender

Guardian

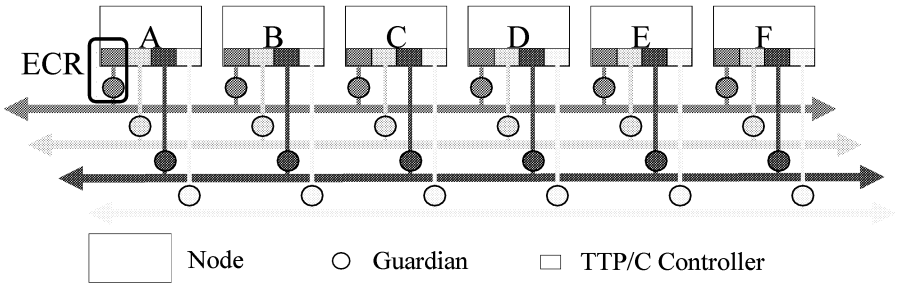
Semantic Analysis for Value Error Detection

The guardian can check the post condition of a message



Four Channel System in TTP/C--Independence

Alternative II: Four Independent Clusters with one channel each



Each physical guardian muliplexes its services to the six nodes.
If a controller starts ***babbling***, neither the node, nore the channel is lost.

Four independent subsystems with 10 FCRs (6 nodes, 4 guardians) and 24 ECRs.
Each subsystem with independent timing and diverse transmission schedule
and possibly diverse bit rates and diverse controllers.

Fault Injection into the TTA

Millions of fault injection experiments have been carried out in the PDCS, TTA and FIT project over a period of more than ten years:

- ♦ Software based (TU Vienna, Austia)
- ♦ Alpha Particle (Chalmers University, Sweden)
- ♦ VLSI-model based (Univ. of Valencia, Spain, Carinthia Tech, Austria)
- ♦ Pin Level (LAAS, Toulouse,France, Univ. of Valencia, Spain)

Main Results:

- ♦ Guardians are needed to avoid error propagation in the temporal domain
- ♦ Guardians must be fully independent: star coupler

Results are documented in the open literature

Formal Analysis of the TTA

In cooperation of the TU Vienna with the University of ULM (Prof. Henke) and SRI (John Rushby) with support from NASA, many of the core algorithms of the TTA are formally analyzed:

- ♦ Clock Synchronization
- ♦ Membership Service
- ♦ Central Guardian Algorithms
- ♦ Start-up Algorithms under fault conditions

The results are published in the open literature.

Fault Scenario 1: Permanent Node Failures

Assumption: Failure Rate: < 100 Fit (> 10 000 000 hours (1000 years)

Source: Field data from automotive ECUs, Bosch, well established data

Relation fail-silent/non fail-silent in temporal domain:

Without guardian: ~ 1: 50

With local guardian: ~ 1: 500

With central bus guardian less than 1: 10000 (none observed)

Source: Fault Injection Experiments in PDCS and FIT project

Fault Scenario 2: Transient Node Failures

Very difficult to get good data!

Assumption: Failure Rate: < 1 000 000 Fit (> 1000 hours)

Source: Papers on neutron induced faults, Xilinx

Relation fail-silent/non fail-silent in temporal domain:

Without guardian: ~ 1:50

With local guardian: ~ 1:500

With central bus guardian less than 1: 10000 (none observed)

Source: Fault Injection Experiments in PDCS and FIT project

Fault Scenario 3: Massive Transient Disturbances

A massive transient disturbance (e.g., caused by EMI) makes it impossible for nodes to communicate for a short interval.

The nodes themselves remain intact.

Detection of massive disturbance: by monitoring membership

Handling of massive disturbance: In the worst case, restart of the whole cluster.

Restart time < 10 TDMA rounds (e.g., 10 msec if TDMA round takes 1 msec)

Strongly dependent on system environment!

Validation of Star Coupler

TTP/C-C1-based hardware prototype with XILINX 600k FPGA (tested by IST project FIT):

Heavy Ion Experiments (at Chalmers):

Bus topology: 37036 faults--78 fail silence violations (0.21 %)

Star topology: 26600 faults-- 0 fail silence violation

Software Implemented Fault Injection (Vienna):

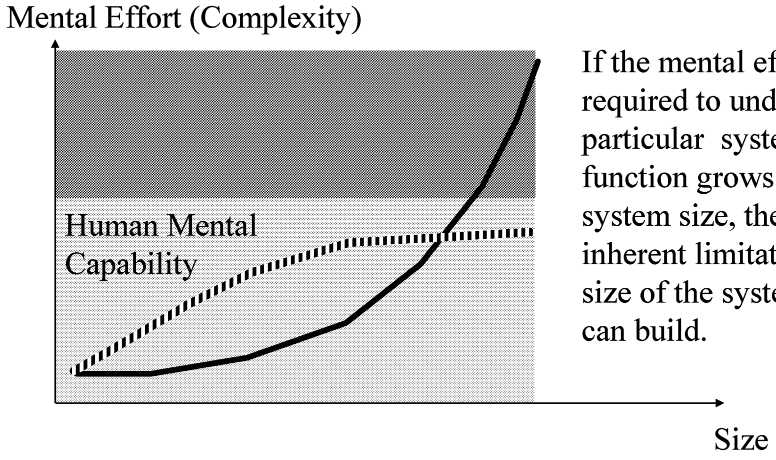
Bus topology: 562122 faults--14 fail silence violations (0.02 %)

Star topology: 541744 faults-- 0 fail silence violation

To be published at DSN, San Francisco, June 2003

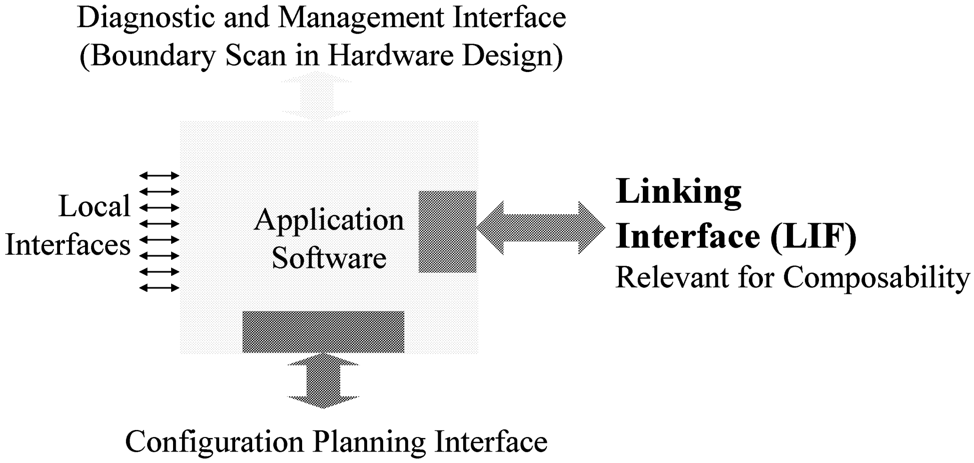
Formal Verification using Model Checking (SAL, UPPAAL2k) and Theorem Proofing (PVS) is ongoing in the NEXT TTA Project.

Size versus Mental Effort to Understand



If the mental effort required to understand a particular system function grows with the system size, there is an inherent limitation to the size of the systems we can build.

Interfaces of a Component


Diagnostic and Management Interface
(Boundary Scan in Hardware Design)

Local
Interfaces

Application
Software

**Linking
Interface (LIF)**
Relevant for Composability

Configuration Planning Interface

Assume that the size of a box is a measure of how long it takes tounderstand the interface

Interfaces of a TTA Component (ii)

Realtime Service (RS) Interface--the linking interface LIF:

- ♦ In control applications periodic
- ♦ Contains RT observations
- ♦ Time sensitive

Diagnostic and Maintenance (DM) Interface:

- ♦ Sporadic access
- ♦ Requires knowledge about internals of a node
- ♦ Not time sensitive

Configuration Planning (CP) Interface:

- ♦ Sporadic access
- ♦ Used to install a node into a new configuration
- ♦ Not time sensitive

Local Interface(s):

- ♦ To other nodes or the environment
- ♦ Not visible to the user of the component

Temporal Firewalls and Composability

A composable architecture must support the

(1) Independent development of components--relates to the architecture

(2) Stability of prior services--relates to the components

(3) Performability of the Communication System--relates to the communication system.

(4) Replica determinism--to support transparent implementation of fault tolerance.

(5) Diagnostics--It mus be possible to identify the sending FCU (Fault Containment Unit) of every message.

The temporal firewall construct supports these five principles of composability.

Linking Interface (LIF) Specification

Operational Specification--Precise in the TTA:

- **Operational Input Interface Specification**
  - Syntactic Specification
  - Temporal Specification
  - Input Assertion
- **Operational Output Interface Specification**
  - Syntactic Specification
  - Temporal Specification
  - Output Assertion
- **Interface State**

Meta-level Specification:

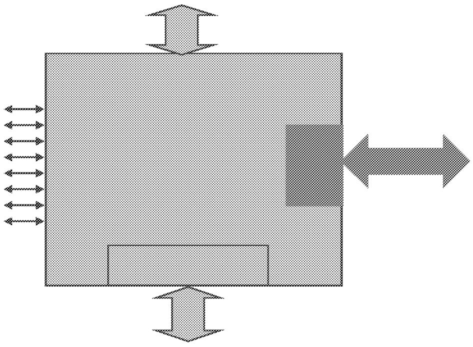
- Meaning of the data elements: Means-and-ends interface model

Complexity Reduction by Partitioning

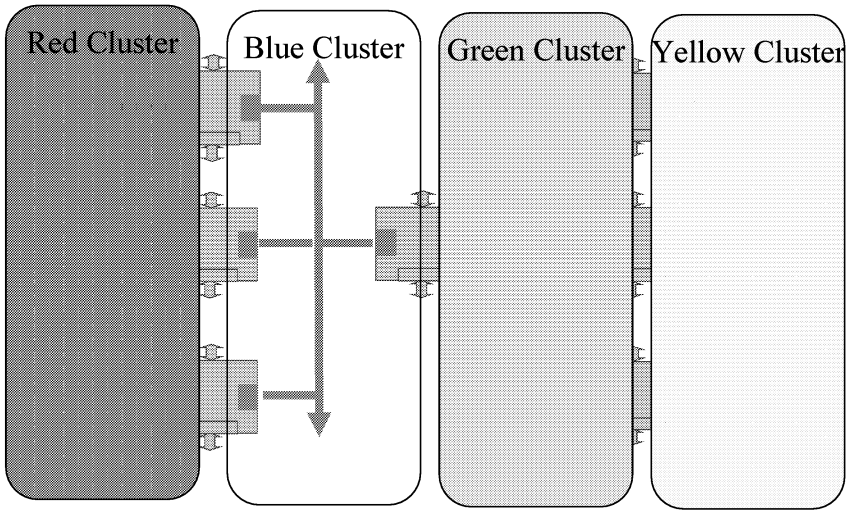
Complexity Reduction:
(LIF Service Interface Complexity)/(Component Complexity)

A *good* decomposition will lead to a significant complexity reduction for the understanding of the emerging functions at the system level.

The easier it is, to understand a LIF interface, the better the decomposition from the point of view of complexity management.



Localized Views of Global System



Red Cluster

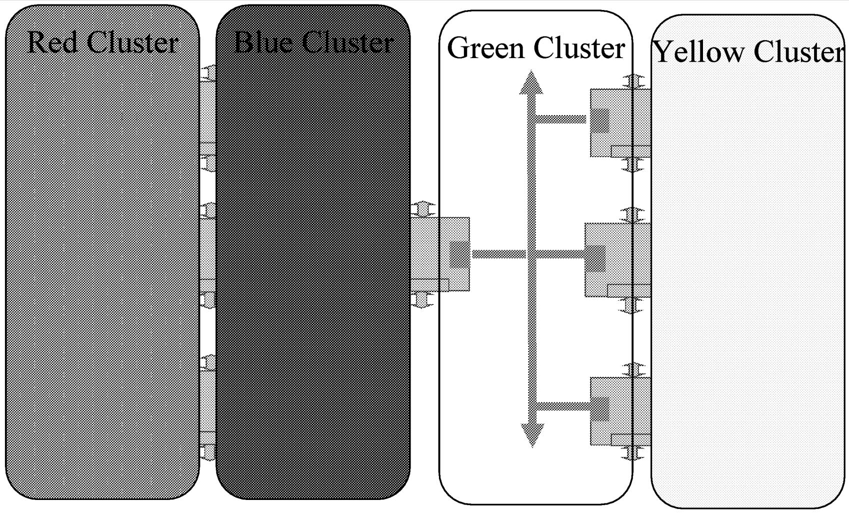
Blue Cluster

Green Cluster

Yellow Cluster

Divide and Conquer

Localized Views of Global System



Red Cluster

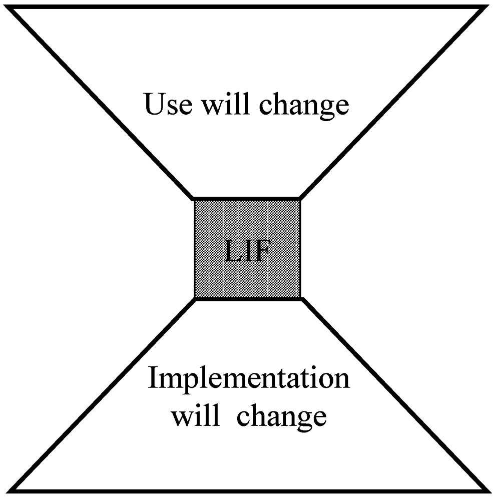
Blue Cluster

Green Cluster

Yellow Cluster

Divide and Conquer

Stable Interface of an Autonomous Subsystem



Use will change

LIF

Implementation will change

Two-level Design Methodology

A two level design methodology is supported by the TTA:

System Level specifies the interactions among components by designing the Temporal Firewalls:

- Data items that are exchanged among the subsystems
- Instants when the TT communication system accesses the data
- Abstract model of the meaning of the data.

Component Level is concerned with the detailed Software Design:

- The host computer provides the intended function, taking the available temporal firewall specifications as constraints.
- Validation of a component with respect to the temporal firewalls can be performed in isolation.

Top-Down Design Process in the TTA

Level 1:

Decompose the design problem into clusters and components

Allocate functions to components

Investigate the data flow among the components

Specify the temporal firewalls in value and time

Estimate the failure rates and specify the fault-hypothesis

Specify the NGU Strategy

Level 2:

Implement the components, taking the temporal firewall specifications as constraints.

Bottom-up Design--Reuse of Components

The bottom up design takes advantage of the existing COTS components and their temporal firewall specification:

- ♦ The input firewall parameters determine what a user is expected to supply
- ♦ The output firewall parameters determine what a user can rely upon

The architecture design must proceed taking these component characteristics as constraints.

Legacy Systems

In many application legacy systems have to be integrated in a new design:

- ♦ Identify the “Linking Interface” of the legacy system.
- ♦ Provide a gateway component that hides the idiosyncracies of the legacy system and provides a standard interface (wrapper technology) to the new architecture.
- ♦ Provide back-pressure flow control in the gateway component.

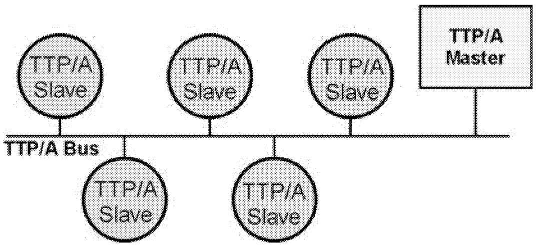
A Smart Transducer Cluster

One active master

Up to 250 slaves

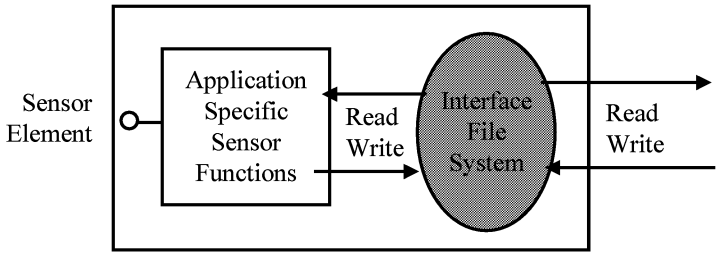
Communication organized in rounds

TDMA bus allocation



TTP/A Sensor Bus

- ♦ Low Cost Time-Triggered Sensor Bus to provide a uniform interface to the different types of smart transducers
- ♦ Has been standardized by the OMG in January 2003
- ♦ Optimized for 8 bit microcontrollers: requires in its minimum version less than 4 kbyte of ROM and 64 bytes of RAM
- ♦ Central to TTP/A is the concept of an interface file system (IFS)



The Three Interfaces of an ST

Real-Time (RS) Service Interface-TT:

- ♦ Contains RT observations
- ♦ Time sensitive
- ♦ In control applications periodic

Diagnostic and Management (DM) Interface-ET

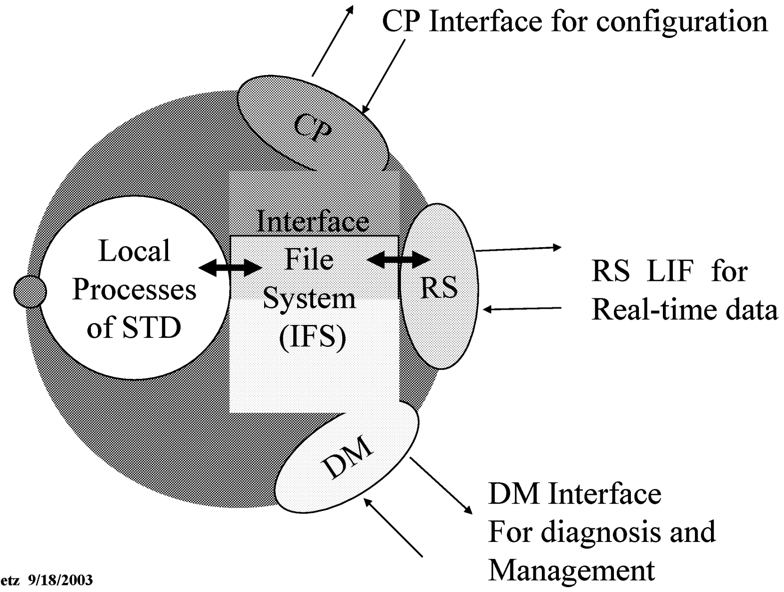
- ♦ Sporadic Access
- ♦ Requires knowledge about internal s of a node
- ♦ Not time sensitive

Configuration Planning (CP) Interface-ET:

- ♦ Used to install a COTS node into a new configuration
- ♦ Not time sensitive

The Protocol supports each one of these interfaces.

The Three Interfaces



CP Interface for configuration

CP

Local Processes of STD

Interface File System (IFS)

RS

RS LIF for Real-time data

DM

DM Interface For diagnosis and Management

Interface File System (IFS)

- ♦ Provides the structured name space for the RT images and other node-relevant data (e.g., documentation).
- ♦ Consists of a set of index-sequential files with constant record length.
- ♦ Records are protected.
- ♦ The following file operation are supported
  - • *read record*
  - • *write record*
  - • *execute record*
- ♦ The configuration information of a round is stored in the file system as a distributed file of a cluster.

Naming

Every record of the IFS can be uniquely identified by the concatenation of

Cluster Name,

Logical Name

File Name, and

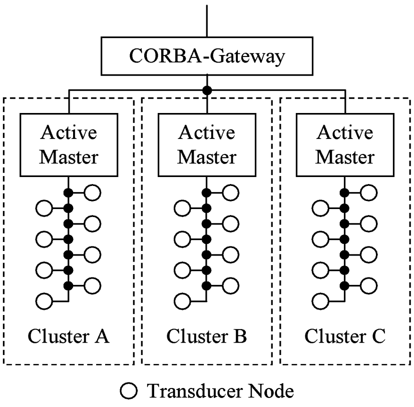
Record Name

Up to 250 Clusters

Up to 250 Nodes

Up to 64 Files

Up to 256 Records



File Operations

The standard supports three operations on an IFS File:

- Read a Record
- Write a Record
- Execute a Record, taking the record contents as a parameter for the execution

The File Operation is encoded in the remaining two bits of the node internal record name.

Meta Information

Every smart transducer node contains a class identifier that points to the meta-information and a unique identifier that identifies every sensor in the universe uniquely.

The OMG manages the identifier names.

The Meta Information about the meaning of the data stored in the IFS is not in the sensor, but on the WEB.

At the moment, research is ongoing to formalize parts the meta-information by using XML.

File Access

Two methods to access a file:

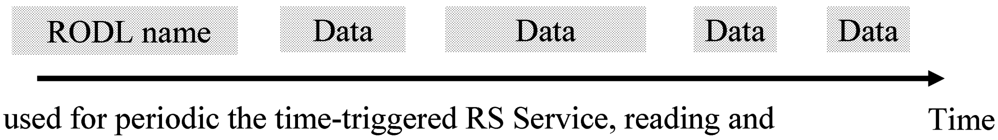
- **Time-Triggered:**
  - Used for RS Interface
  - Periodic execution of a preconfigured round.
  - Round descriptor list RODL can be stored in an assigned IFS file
- **Event-Triggered**:
  - Used for CP and DM Interface
  - Read, write or execute any record within a master slave round:
  - the master (client) must form the full address of the slave record

Principle of Operation

- ♦ Endpoint of the communication is a record in an Interface File System (IFS) located in the transducer node.
- ♦ Communication is organized into Rounds
  - • A round is started by the active master that has knowledge of the global time
  - • The first frame of a round is a fireworks frame, followed by data frames. The structure of a round is described in the round-descriptor list (RODL).
  - • every round is independent of every other round
- ♦ The arrival of the fireworks frame is the global synchronization event starting a new epoch.

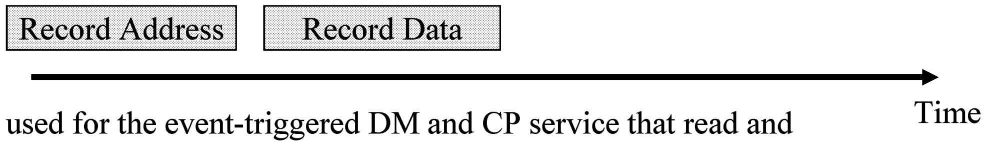
Round Types

Multipartner Round (TT):



used for periodic the time-triggered RS Service, reading and writing data of the IFS records containing RT images.

Master-Slave Round (ET):

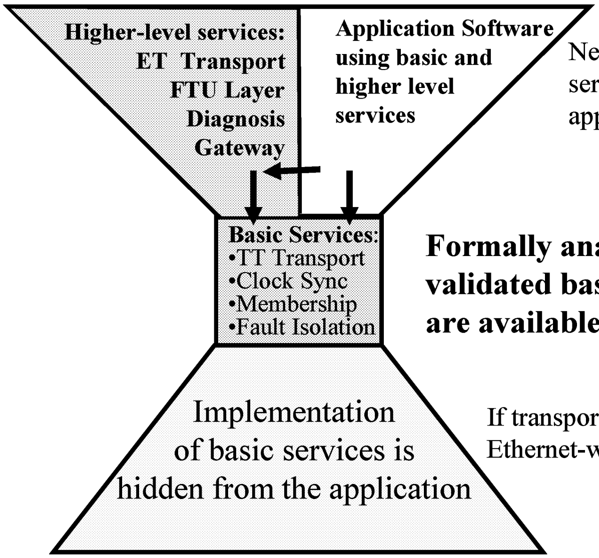


used for the event-triggered DM and CP service that read and write records of the IFS containing calibration, diagnostic and configuration data..

Future Plans

- ♦ **Extend the TTA to higher bandwidths**: an exploratory research project to implement the TTA on Gigabit Ethernet has been completed.
- ♦ **Provide more high-level services as the need arises**.
- ♦ **Investigate the issues related to the implementation of large multicluster systems** (e.g., multi-clusterclock synchronization)
- ♦ Investigate the rigorous specification of interface models in order to support model-based design.
- ♦ Explore issues related to the modular certification of TTA systems.

The TTA is Waist-line Architecture



New high-level services to ease application development

Formally analyzed and validated basic services are available and stable.

If transport is replaced by Ethernet-what are the issues?

TT Gigabit Ethernet Prototype Results

- ♦ TT Ethernet can be implemented without any changes to the Ethernet protocol standard.
- ♦ Current switches have a high jitter and thus give a low precision.
- ♦ New switch, with predictable jitter and integrated guardian functions must be designed.
- ♦ TTA algorithms (clock synchronization, membership, fault isolation) can be transferred to TT new Ethernet switch without any change.
- ♦ Since all basic research issues are resolved, TT Ethernet implementation is an engineering task.

Conclusion

The design of an architecture for safety-critical applications requires

- ♦ A clear set of priorities, where safety is at the top of the list
- ♦ A precise fault-hypothesis, such that the appropriate fault-tolerance mechanisms can be designed.
- ♦ A quest for guaranteeing independence of FCRs and the elimination of all potential paths of error propagation.
- ♦ Multiple defenses and assurances at all levels of the architecture.

Twelve Principles for the Design of Safety-Critical Embedded Systems

H. Kopetz
TU Vienna

January 2003

Outline

- ♦ Introduction
- ♦ Design Challenges
- ♦ The Twelve Design Principles
- ♦ Conclusion

Some Design Challenges

- The 10^{-9} Challenge
- The Process of Abstracting
- Physical Hardware Faults
- Design Faults
- Human Failures

The 10^{-9} Challenge

- **The system as a whole must be more reliable than any one of its components**: *e.g., System Dependability 1 FIT--Component dependability 1000 FIT (1 FIT: 1 failure in 10^9 hours)*
- Architecture **must support fault-tolerance** to mask component failures
- System as a whole is **not testable** to the required level of dependability.
- The safety argument is based on a **combination** of **experimental evidence** about the expected failure modes and failures rates of **fault-containment regions (FCR)** and a **formal dependability model** that depicts the system structure from the point of view of dependability.

The Process of Abstracting

- ♦ The behavior of a safety-critical computer system must be explainable by a hierarchically structured set of behavioral models, each one of them of a cognitive complexity that can be handled by the human mind.
- ♦ Establish a clear relationship between the behavioral model and the dependability model at such a high level of abstraction that the analysis of the dependability model becomes tractable.
- ♦ From the dependability point of view, the future unit of hardware failure is considered to be a complete chip.

Physical Hardware Faults of SoCs:

Assumed Hardware Failure Rates (Orders of Magnitude):

Type of Failure	Failure Rate in Fit	Source
Transient Node Failures (fail silent)	1 000 000 Fit (MTTF = 1000 hours)	Neutron bombardment Aerospace
Transient Node Failure (non-fail silent)	10 000 Fit (MTTF= 100 000)	Fault Injection Experiments
Permanent Hardware Failures	100 Fit (MTTF= 10 000 000)	Automotive Field Data

Tendency: Increase of Transient Failures

Design Faults

No silver bullet has been found yet--and this is no silver bullet either: Interface Centric Design (ICD)!

- ♦ Partition the system along well-specified linking interfaces (LIF) into nearly independent components.
- ♦ Provide a hierarchically structured set of ways-and-means models of the LIFs, each one of a cognitive complexity that is commensurate with the human cognitive capabilities.
- ♦ Design and validate the components in isolation w.r.t. the LIF specification und make sure that the composition is free of side effects (composability of the architecture).

The Twelve Design Principles

1. **Regard the Safety Case as a Design Driver**
2. **Start with a Precise Specification of the Design Hypotheses**
3. **Ensure Fault-Containment and Error Containment**
4. **Establish a Consistent Notion of Time and State**
5. **Partition the System along well-specified LIFs**
6. **Make Certain that Components Fail Independently**
7. **Follow the Self-Confidence Principle**
8. **Hide the Fault-Tolerance Mechanisms**
9. **Design for Diagnosis**
10. **Create an Intuitive and Forgiving Man-Machine Interface**
11. **Record Every Single Anomaly**
12. **Provide a Never Give-Up Strategy**

Regard the Safety Case as a Design Driver

- ♦ A safety case is a set of documented arguments in order to convince experts in the field (e.g., a certification authority) that the provided system as a whole is safe to deploy in a given environment.
- ♦ The safety case, which considers the system as **whole**, determines the criticality of the computer system and analyses the impact of the computer-system failure modes on the safety of the application.
- ♦ The distributed computer system should be structured such that the required experimental evidence can be collected with reasonable effort and that the dependability models that are needed to arrive at the system-level safety are tractable.
- ♦ The safety case should be regarded as a design driver since it establishes the critical failure modes of the computer system.

Start with a Precise Specification of the Design Hypotheses

The design hypotheses is a statement about the assumptions that are made in the design of the system. Of particular importance for safety critical real-time systems is the fault-hypotheses: a statement about the number and types of faults that the system is expected to tolerate:

- ♦ Determine the Fault-Containment Regions (FCR): *A fault-containment region (FCR)* is the set of subsystems that share one or more common resources and that can be affected by a single fault.
- ♦ Specification of the Failure Modes of the FCRs and their Probabilities
- ♦ Be aware of Scenarios that are not covered by the Fault-Hypothesis

Fault Containment

- The immediate consequences of a fault must be isolated to within a well-defined region, the *fault containment region.*
- Fault-Containment Regions must fail independently.
- Consider spatial proximity.
- Design Faults?

Ensure Error Containment

In a distributed computer system the consequences of a fault, the ensuing error, can propagate outside the originating FCR either by an erroneous message or by an erroneous output action of the faulty node to the environment that is under the node's control.

- A propagated error **invalidates** the independence assumption.
- The error detector must be in different FCR than the faulty unit.
- Distinguish between architecture-based and application-based error detection
- Distinguish between error detection in the time-domain and error detection in the value domain.

Establish a Consistent Notion of Time and State

A system-wide consistent notion of a discrete time is a prerequisite for a consistent notion of state, since the notion of *state* is introduced in order to separate the *past* from the *future*:

"The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other word, the state enables a "decoupling" of the past from the present and future. The state embodies all past history of a system. Knowing the state "supplants" knowledge of the past. Apparently, for this role to be meaningful, the notion of past and future must be relevant for the system considered." (Taken from Mesarovic, Abstract System Theory, p.45)

Fault-masking by voting requires a consistent notion of state in distributed Fault Containment Regions (FCRs).

State (ii)

- ♦ Provide a mechanism to heal corrupted state
- ♦ State can be corrupted by the progression of time or the loss of a state update or a bitflip in the state (SER)
- ♦ How to heal corrupted interface state?
- ♦ permanent soft error can be caused by

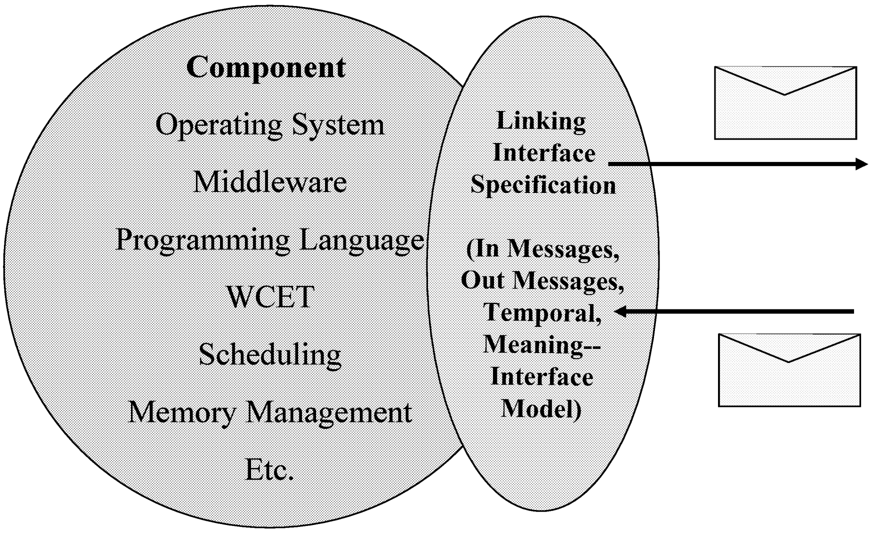
Partition the System along well-specified LIFs

"Divide and Conquer" is a well-proven method to master complexity.

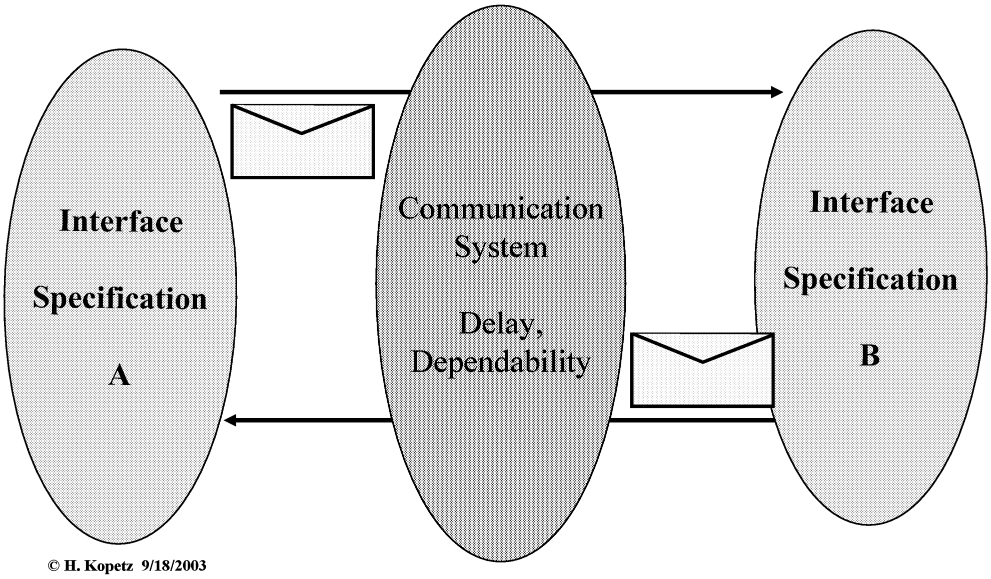
A linking interface (LIF) is an interface of a component that is used in order to integrate the component into a system-of-components.

- ♦ We have identified only two different types LIFs:
  - • time sensitive LIFs and
  - • not time sensitive LIFs
- ♦ Within an architecture, all LIFs of a given type should have the same generic structure
- ♦ Avoid concurrency at the LIF level

The LIF Specification hides the Implementation


Component
Operating System
Middleware
Programming Language
WCET
Scheduling
Memory Management
Etc.
Linking Interface Specification
(In Messages, Out Messages, Temporal, Meaning--Interface Model)

Composability in Distributed Systems



Interface

Specification

A

Communication System

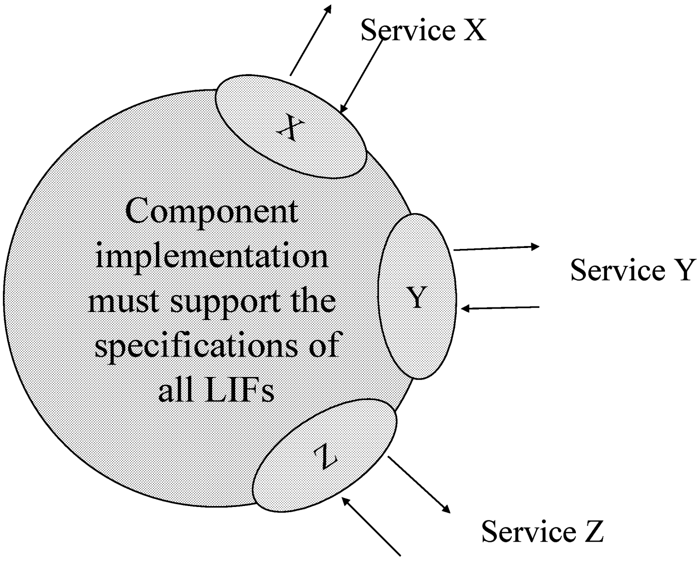
Delay, Dependability

Interface

Specification

B

A Component may support many LIFs



Component implementation must support the specifications of all LIFs

Make Certain that Components Fail Independently

Any dependence of FCR failures must be reflected in the dependability model--a challenging task!

Independence is a system property. Independence of FCRs can be compromised by

- Shared physical resources (hardware, power supply, time-base, etc.)
- External faults (EMI, heat, shock, spatial proximity)
- Design
- Flow of erroneous messages

Follow the Self-Confidence Principle

The self-confidence principles states that an FCR should consider itself correct, unless **two or more** independent FCRs classify it as incorrect.

If the self-confidence principle is observed then

- a correct FCR will always make the correct decision under the assumption of a single faulty FCR
- Only a faulty FCR will make false decisions.

Hide the Fault-Tolerance Mechanisms

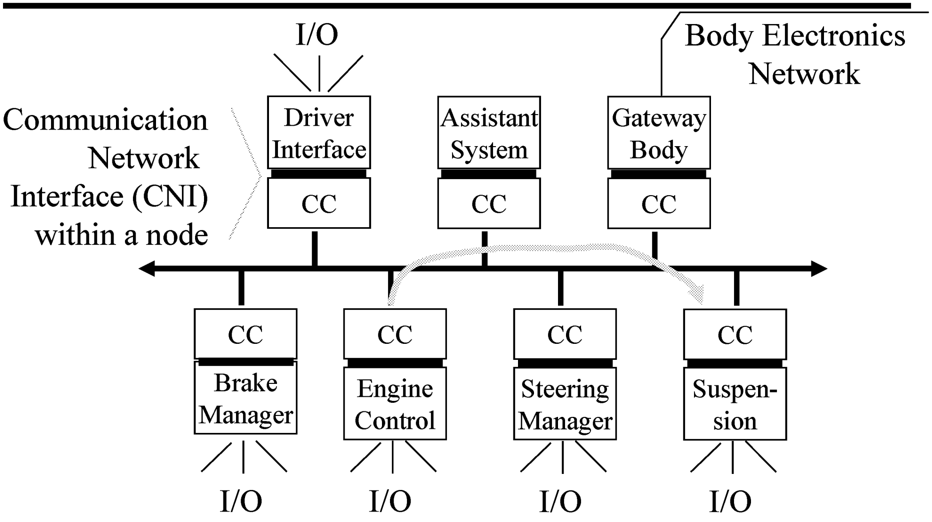
- The complexity of the FT algorithms can increase the probability of design faults and beat its purpose.
- Fault tolerance mechanisms (such as voting, recovery) are generic mechanisms that should be separated from the application in order not to increase the complexity of the application.
- Any fault-tolerant system requires a capability to detect faults that are masked by the fault-tolerance mechanisms--this is a generic diagnostic requirement that should be part of the architecture.

Design for Diagnosis

The architecture and the application of a safety-critical system must support the identification of a field-replaceable unit that violates the specification:

- Diagnosis must be possible on the basis of the LIF specification and the information that is accessible at the LIF
- Transient errors pose the biggest problems--Condition based maintenance
- Determinism of the Architecture helps!
- Avoid Diagnostic Deficiencies
- Scrubbing--Ensure that the FT mechanisms work

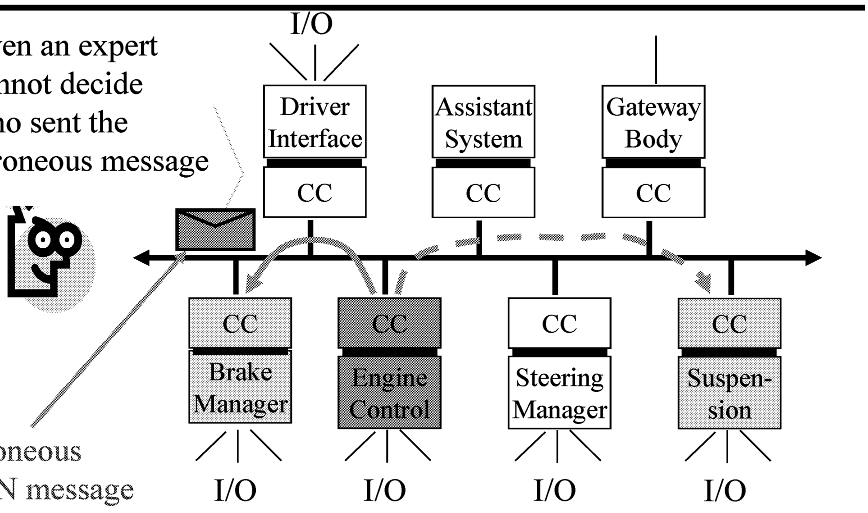
Flexibility vs. Diagnostics in CAN



CC: Communication Controller

Diagnostic Deficiency in CAN

Even an expert cannot decide who sent the erroneous message



Erroneous CAN message with wrong identifier

CC: Communication Controller

Create an Intuitive and Forgiving Man-Machine Interface

- ♦ The system designer must assume that human errors will occur and must provide mechanisms that mitigate the consequences of human errors.
- ♦ Three levels of human errors
  - Mistakes (misconception at the cognitive level)
  - Lapses (wrong rule from memory)
  - Slips (error in the execution of a rule)

Record Every Single Anomaly

- ♦ Every single anomaly that is observed during the operation of a safety critical computer system must be investigated until an explanation can be given.
- ♦ This requires a well-structured design that is analyzable at all levels.
- ♦ Since in a fault-tolerant system many anomalies are masked by the fault-tolerance mechanisms, the observation mechanisms must access the non-fault-tolerant layer.

Out-of-Norm Behavior

- In reality, there is often a gray area between *correct* and *incorrect*.
- Information about *out-of-norm* behavior can be very valuable in the diagnostic process

  Example: borderline between transient and intermittent failures of a chip
- Application specific assertion for the detection of *out-of-norm* behavior

Provide a Never Give-Up Strategy

- There will be situations when the fault-hypothesis is violated and the fault tolerant system will fail.
- Chances are good that the faults are transient and a restart of the whole system will succeed.
- Provide algorithms that detect the violation of the fault hypothesis and that initiate the restart.
- Ensure that the environment is safe (e.g., freezing of actuators) while the system restart is in progress.
- Provide an upper bound on the restart duration.

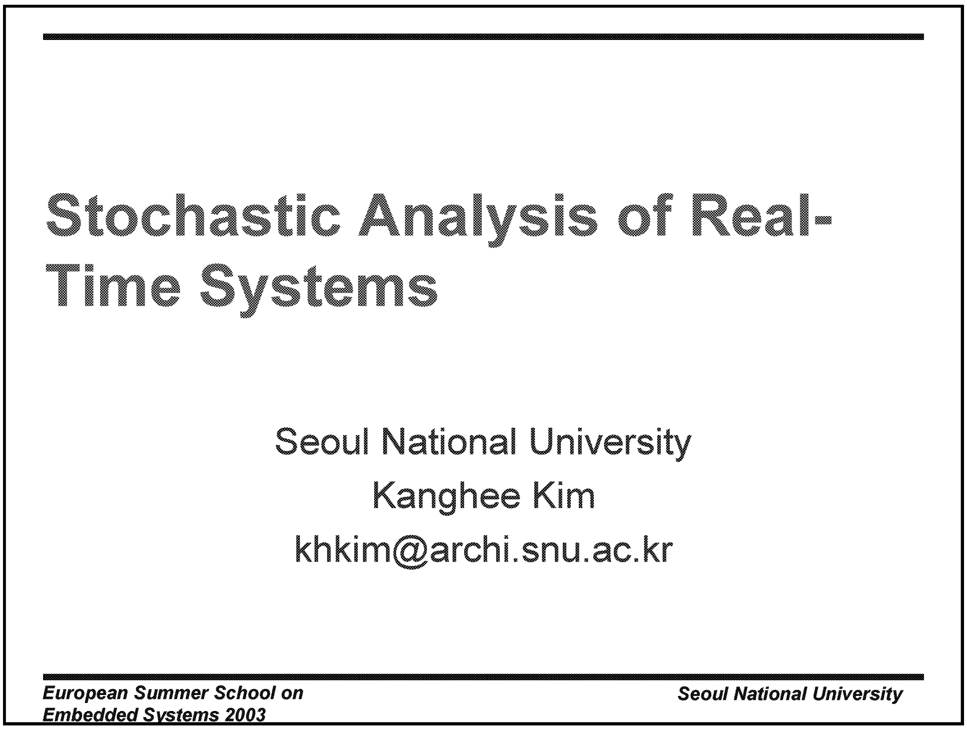
Conclusion

Every one of these twelve design principles can be the topic of a separate talk!

Thank you

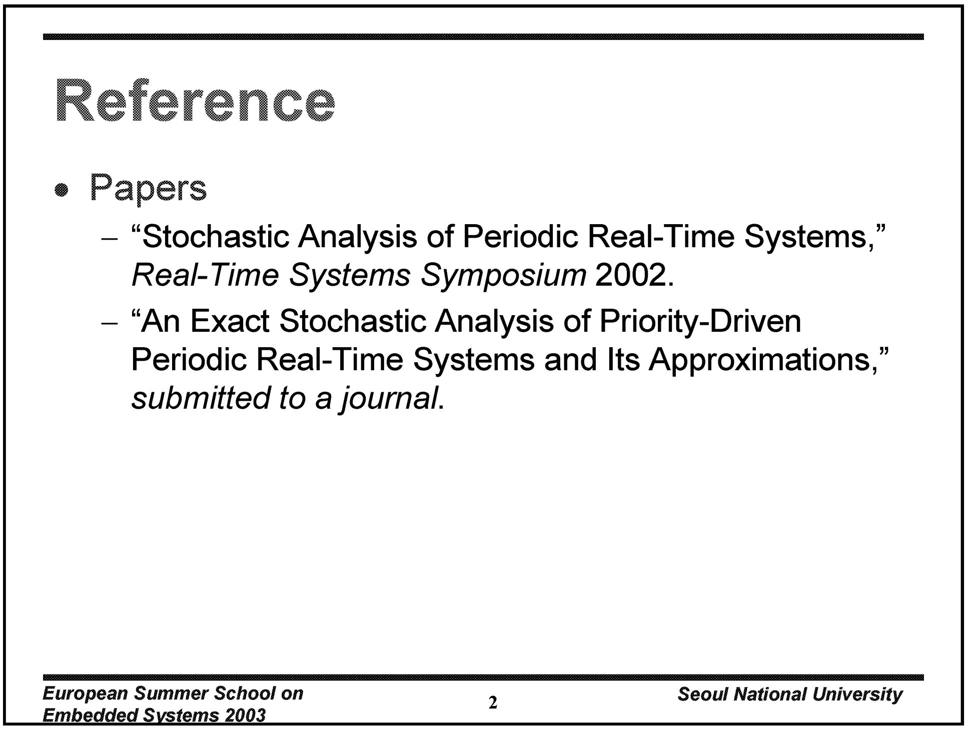
Kanghee Kim

khkim@archi.snu.ac.kr
Seoul National University



Stochastic Analysis of Real-Time Systems

Seoul National University
Kanghee Kim
khkim@archi.snu.ac.kr



Reference

- Papers
  - “Stochastic Analysis of Periodic Real-Time Systems,” *Real-Time Systems Symposium* 2002.
  - “An Exact Stochastic Analysis of Priority-Driven Periodic Real-Time Systems and Its Approximations,” *submitted to a journal*.

Outline

- Introduction
- Related work
- Our stochastic analysis
  - Backlog and interference analysis
  - Steady-state backlog analysis
- Complexity analysis
- Experimental results
- Conclusion

Introduction

- The conventional deterministic schedulability analysis causes an underutilization of the system.
- A stochastic approach to the schedulability analysis can improve the system utilization.
- By the stochastic analysis, each task is guaranteed to meet its deadline with the computed probability ⇒ probabilistic guarantee
- The need for probabilistic guarantees covers both soft real-time and hard real-time systems

Related Work

- Approach 1: "**Analyze the system as it is.**"
  - Gardner & Liu's Stochastic Time Demand Analysis
  - Lehoczky's Real-Time Queueing Theory
- Approach 2: "**Invent a system easy to analyze**."
  - Abeni & Buttazzo's Reservation-based Systems
  - Atlas & Bestavros' Statistical Rate Monotonic Scheduling

Approach 1

- Gardner & Liu's Stochastic Time Demand Analysis
  - a stochastic extension of the Time Demand Analysis
  - tries to compute an upper bound of the deadline miss probability for each task, based on the concept of critical instants.
  - provides no proof for the safeness for systems with $U^{max} > 1$.

Approach 1 (cont'd)

- Lehoczky's Real-Time Queueing Theory
  - an extension of the conventional Queueing Theory to real-time systems
  - an approximated analysis for systems under heavy-traffic conditions $(\overline{U} \cong 1)$
  - assumes that all tasks follow
    - a single interarrival time distribution and
    - a single service time distribution.
  - extends to Real-Time Queueing Networks.

Approach 2

- Abeni & Buttazzo's Reservation-based Systems
  - For each task, one dedicated virtual processor is provided (period based reservation).
  - A job in a task overrunning the allocated budget is allowed to steal that of the next job in the same task.
  - A stochastic analysis is performed for each task assuming the isolated environment.
    - Markov process modeling

Approach 2 (cont'd)

- Atlas & Bestavros' Statistical Rate Monotonic Scheduling
  - For each task, one dedicated virtual processor is provided (super-period based reservation).
    - super-period: a multiple of the period of the task, equal to the period of the next task
  - A job in a task can consume the allocated budget as much as it wants.
  - If the budget is not enough, newly arriving jobs are rejected. (⇒ firm real-time)
  - A stochastic analysis is performed for each task to calculate the rejection probability, which is regarded as the deadline miss probability.

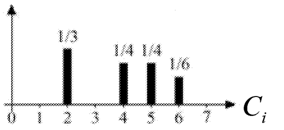
Contributions

- Our stochastic analysis
  - follows "Approach 1: Analyze the system as it is".
  - introduces no pessimistic and restrictive assumptions.
    - tasks with arbitrary execution time distributions (thus, arbitrary utilization factors) and arbitrary relative deadlines.
    - general priority-driven scheduling algorithms
      - includes both fixed-priority and dynamic-priority algorithms (RM, DM, EDF, ...)
  - accurately models all possible interactions between tasks (thus giving the exact deadline miss probability for each task).

System Model

- Task model
  - independent periodic tasks τ_i (i = 1, 2, ..., n)
  - fixed period T, phase ϕ, relative deadline D
  - execution time C: discrete random variable
    - probability mass function (PMF)

$f_{C_i}(c) = P(C_i = c)$

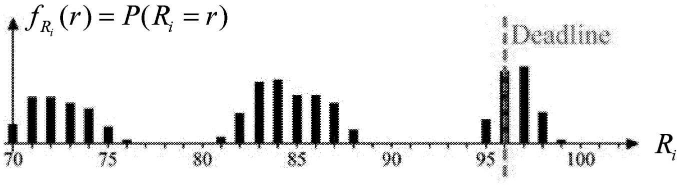


- Scheduling model
  - priority-driven preemptive scheduling (RM, DM, EDF, ...)

The Objective

Compute the exact response time distribution for each task τ_i

$f_{R_i}(r) = P(R_i = r)$



- The response time distribution can be used
  - to compute the deadline miss probability $P(R_i > D_i)$
  - to compute other QoS-related parameters

The Objective (cont'd)

Compute the stationary response time distribution for each task τ_i, that is $\lim_{t\to\infty} f_{R_i}$

(R_i = the response time of τ_i)

- We want the response time distribution observed when the system is in steady state.
- A response time profile obtained for the task in a real system will converge towards the stationary distribution.

The Objective (cont'd)

Compute the stationary response time distribution for each job $J_{i,j}$ in a hyperperiod , that is

$$\lim_{k\to\infty} f_{R_{i,j}^{(k)}} = f_{R_{i,j}^{(\infty)}}$$

($R_{i,j}^{(k)}$ = the response time of $J_{i,j}$ in hyperperiod k)

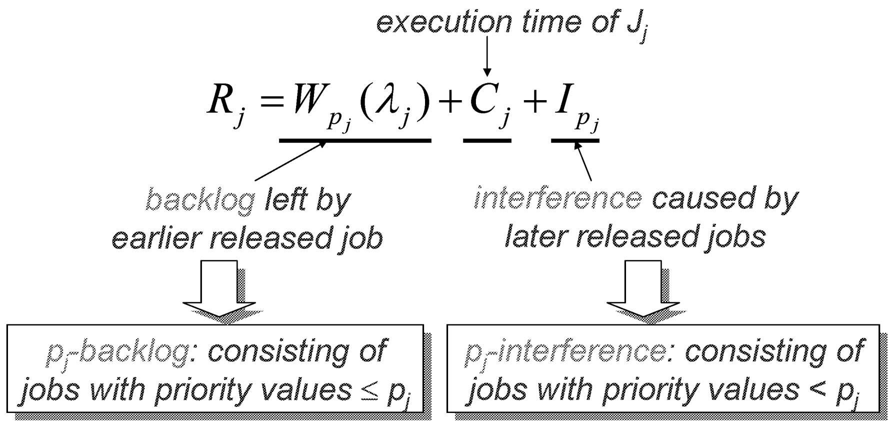
- The response time distribution of task τ_i can be obtained by averaging those of all jobs $J_{i,j}$ belonging to the task.

Notations

- J_j : the *j*-th job in a hyperperiod
- $J_j^{(k)}$: the *j*-th job in hyperperiod *k*
- λ_j : the release time of job J_j
- p_j : the priority value of job J_j
  - A lower priority value means a higher priority.

Response Time Equation

- Response time of job J_j



execution time of J_j

$$R_j = \underline{W_{p_j}(\lambda_j)} + \underline{C_j} + \underline{I_{p_j}}$$

backlog left by earlier released job

interference caused by later released jobs

p_j-backlog: consisting of jobs with priority values ≤ p_j

p_j-interference: consisting of jobs with priority values < p_j

Overview of the Stochastic Analysis

1st step: backlog analysis

Compute the stationary p_j-backlog distribution for each job J_j in a hyperperiod

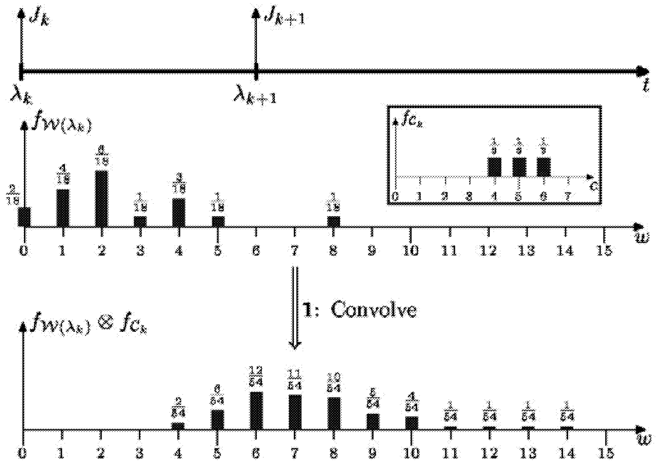
2nd step: interference analysis

Compute the stationary response time distribution by reflecting the p_j-interference effect

Backlog Analysis

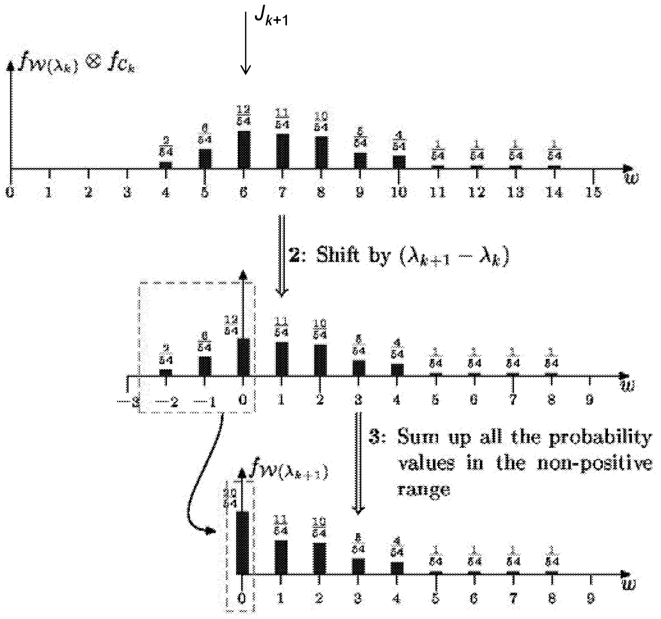
- Algorithm
  - input
    - a sequence of earlier released jobs with priority values $\leq p_j \Rightarrow \{J_1, J_2, ..., J_j\}$
    - the p_j-backlog distribution observed at λ_1
  - output
    - the p_j-backlog distribution observed at λ_j
  - procedure
    - iteratively compute the p_j-backlog distribution observed at λ_k from that observed at $\lambda_{k\text{-}1}$ (k=2,3,...,j)

Example



J_k
J_{k+1}
λ_k
λ_{k+1}
t
$f_{\mathcal{W}(\lambda_k)}$
$\frac{2}{18}$ $\frac{4}{18}$ $\frac{6}{18}$ $\frac{1}{18}$ $\frac{3}{18}$ $\frac{1}{18}$ $\frac{1}{18}$
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
w
f_{C_k}
$\frac{1}{3}$ $\frac{1}{3}$ $\frac{1}{3}$
0 1 2 3 4 5 6 7
c
$f_{\mathcal{W}(\lambda_k)} \otimes f_{C_k}$
$\frac{2}{54}$ $\frac{6}{54}$ $\frac{12}{54}$ $\frac{11}{54}$ $\frac{10}{54}$ $\frac{5}{54}$ $\frac{4}{54}$ $\frac{1}{54}$ $\frac{1}{54}$ $\frac{1}{54}$ $\frac{1}{54}$
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
w

1: Convolve



J_{k+1}
$f_{\mathcal{W}(\lambda_k)} \otimes f_{C_k}$
$\frac{2}{54}$ $\frac{6}{54}$ $\frac{12}{54}$ $\frac{11}{54}$ $\frac{10}{54}$ $\frac{5}{54}$ $\frac{4}{54}$ $\frac{1}{54}$ $\frac{1}{54}$ $\frac{1}{54}$ $\frac{1}{54}$
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
w
$\frac{2}{54}$ $\frac{6}{54}$ $\frac{12}{54}$ $\frac{11}{54}$ $\frac{10}{54}$ $\frac{5}{54}$ $\frac{4}{54}$ $\frac{1}{54}$ $\frac{1}{54}$ $\frac{1}{54}$ $\frac{1}{54}$
−3 −2 −1 0 1 2 3 4 5 6 7 8 9
w
$f_{\mathcal{W}(\lambda_{k+1})}$
$\frac{20}{54}$ $\frac{11}{54}$ $\frac{10}{54}$ $\frac{5}{54}$ $\frac{4}{54}$ $\frac{1}{54}$ $\frac{1}{54}$ $\frac{1}{54}$ $\frac{1}{54}$
0 1 2 3 4 5 6 7 8 9
w

2: Shift by $(\lambda_{k+1} - \lambda_k)$

3: Sum up all the probability values in the non-positive range

Steady-State Backlog Analysis

- Naive approach
  - apply the backlog analysis algorithm for an infinite sequence of jobs required for each job J_j.
  - input
    - an infinite sequence of jobs with priority values $\leq p_j$.

$$\{\underbrace{J_1^{(1)},\ldots,J_j^{(1)},\ldots,J_n^{(1)}}_{\substack{\text{all jobs with } p \leq p_j^{(\infty)} \\ \text{in hyperperiod 1}}},\underbrace{J_1^{(2)},\ldots,J_j^{(2)},\ldots,J_n^{(2)}}_{\substack{\text{all jobs with } p \leq p_j^{(\infty)} \\ \text{in hyperperiod 2}}},\cdots,\underbrace{J_1^{(\infty)},\ldots,J_j^{(\infty)}}_{\substack{\text{all jobs with } p \leq p_j^{(\infty)} \\ \text{in hyperperiod } \infty}}\}$$

    - null backlog at the beginning of hyperperiod 1
  - output
    - the p_j-backlog distributions observed at $\lambda_j^{(1)}$, $\lambda_j^{(2)}$, ..., $\lambda_j^{(\infty)}$

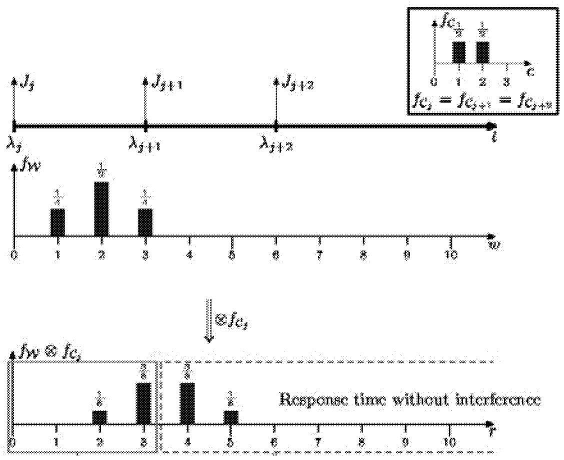
Steady-State Backlog Analysis (cont'd)

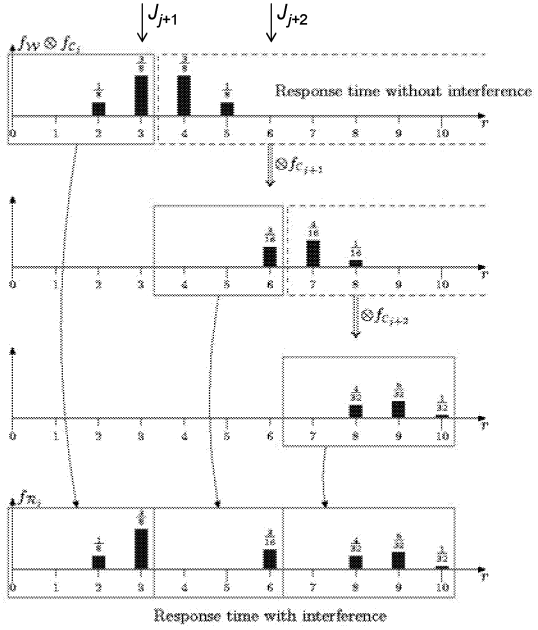
- Problem of the naive approach
  - We have to consider an infinite job sequence for each job J_j in hyperperiod ∞.
  - The computational complexity is too high.
- Solution
  - We show that computing all the stationary backlog distributions (SBDs) of jobs in hyperperiod ∞ reduces to computing the SBD of a single job, which satisfies a certain condition. (⇒ backlog dependency tree)
  - We also show that the SBD of the single job can be analytically computed. (⇒ Markov process modeling)

Interference Analysis

- Algorithm
  - input
    - the p_j-backlog distribution observed at λ_j
    - the execution time distribution of job J_j
    - a sequence of later released jobs with priority values < $p_j \Rightarrow \{J_{j+1}, J_{j+2}, \ldots, J_{j+k}, \ldots\}$
  - output
    - the response time distribution of job J_j
  - procedure
    - iteratively reflect the interference effect of each job J_{j+k} into the response time distribution of J_j being computed (k=1, 2, ...)

Example


J_j
J_{j+1}
J_{j+2}
λ_j
λ_{j+1}
λ_{j+2}
t
fc_j
$\frac{1}{2}$ $\frac{1}{2}$
0 1 2 3
c
$fc_j = fc_{j+1} = fc_{j+2}$
fw
$\frac{1}{4}$ $\frac{1}{2}$ $\frac{1}{4}$
0 1 2 3 4 5 6 7 8 9 10
w
$\otimes fc_j$
$fw \otimes fc_j$
$\frac{1}{8}$ $\frac{3}{8}$ $\frac{3}{8}$ $\frac{1}{8}$
Response time without interference
0 1 2 3 4 5 6 7 8 9 10
r


J_{j+1}
J_{j+2}
$f_W \otimes f_{c_j}$
$\frac{1}{8}$ $\frac{3}{8}$ $\frac{3}{8}$ $\frac{1}{8}$
Response time without interference
0 1 2 3 4 5 6 7 8 9 10
r
$\otimes f_{c_{j+1}}$
$\frac{3}{16}$ $\frac{4}{16}$ $\frac{1}{16}$
$\otimes f_{c_{j+2}}$
$\frac{4}{32}$ $\frac{5}{32}$ $\frac{1}{32}$
f_{R_j}
$\frac{1}{8}$ $\frac{3}{8}$ $\frac{3}{16}$ $\frac{4}{32}$ $\frac{5}{32}$ $\frac{1}{32}$
Response time with interference

Consideration

- What if the set of interfering jobs is infinite?
  - If we are only interested in the deadline miss probability (DMP) of the job J_j, the set of jobs we should consider is limited to those released between [λ_j, λ_j+D_j].

$$DMP_j = P(R_j > D_j) = 1 - P(R_j \leq D_j)$$

  - If a high priority job J_{j+k} is found such that the response time of J_j cannot reach the release time of J_{j+k}, the interference analysis algorithm can be terminated.

Backlog Dependency

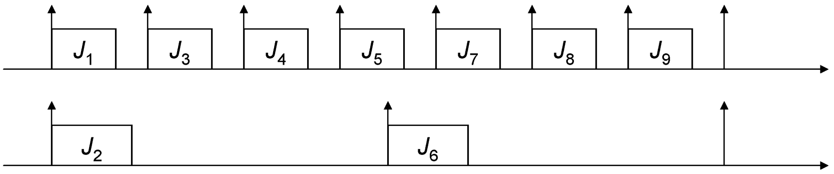
- Motivation
  - We do not want to perform the steady-state backlog analysis for each job in hyperperiod ∞.
  - We observe that there exists a dependency between the p_j-backlog distributions of jobs.
- Advantage
  - We can greatly simplify the steady-state backlog analysis
    - by reducing the problem of computing all the SBDs of jobs in hyperperiod ∞ to computing the SBD of one single job.

Backlog Dependency

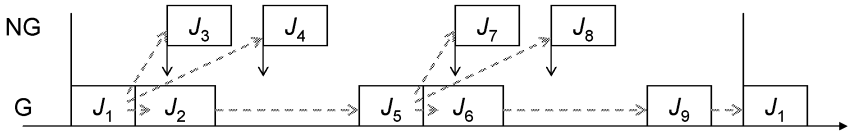
- Procedure for finding the dependencies
  - We classify all jobs in a hyperperiod into ground jobs and non-ground jobs.
    - ground job: p_j-backlog = system backlog
      - A job J_j is a ground one if $p_j \geq p_k$ for all previously released jobs J_k.
    - non-ground job: p_j-backlog ≠ system backlog
  - We find a backlog dependency between ground jobs and non-ground jobs in terms of base jobs.
    - base job: defined for each job J_j as a preceding ground job J_i with $p_i \leq p_j$

Example: EDF

- Task set

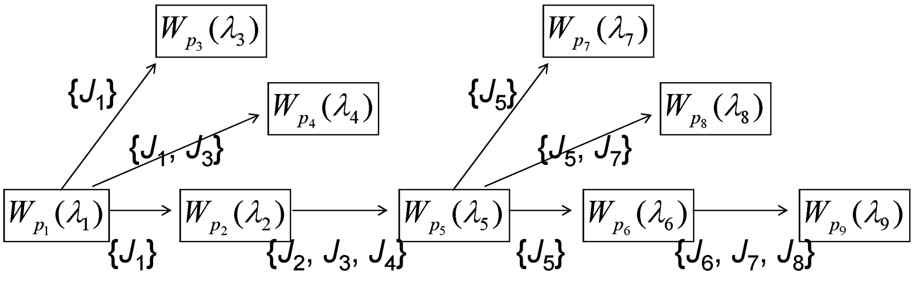


- Ground jobs (G) and non-ground jobs (NG)



Example: EDF (cont'd)

- Backlog dependency tree



Example: EDF (cont'd)

- Derived job sequences assuming $W_{p_1}(\lambda_1)$
  - J_1 : { }
  - J_2 : { J_1 }
  - J_3 : { J_1 }
  - J_4 : { J_1, J_3 }
  - J_5 : { J_1 } U { J_2, J_3, J_4 }
  - J_6 : { J_1 } U { J_2, J_3, J_4 } U { J_5 }
  - J_7 : { J_1 } U { J_2, J_3, J_4 } U { J_5 }
  - J_8 : { J_1 } U { J_2, J_3, J_4 } U { J_5, J_7 }
  - J_9 : { J_1 } U { J_2, J_3, J_4 } U { J_5 } U { J_6, J_7, J_8 }

Properties of EDF

- Existence of ground jobs

Lemma 1: ***There exists at least one ground job in any hyperperiod.***

- Existence of base jobs

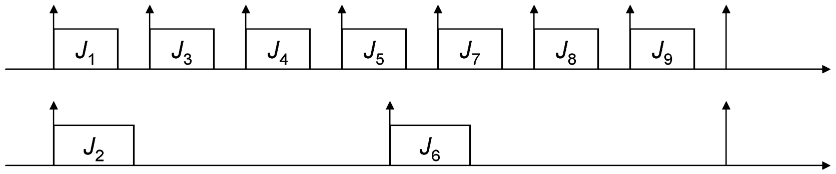
Lemma 2: ***For any non-ground job, there exists the base job.***

- Search limit of base jobs

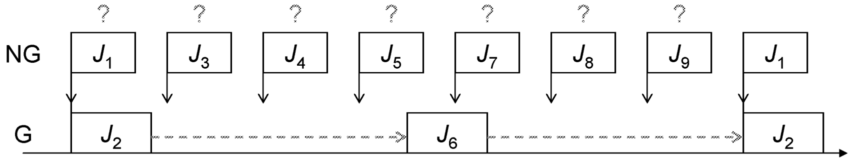
Lemma 3: ***For any non-ground job J_j, the base job can be found in [$\lambda_j - (D^{max} + T_H), \lambda_j$].***

Example: RM

- Task set

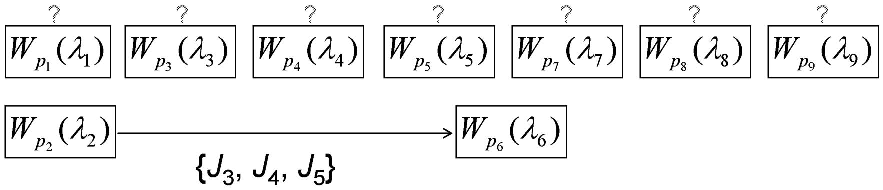


- Ground jobs (G) and non-ground jobs (NG)

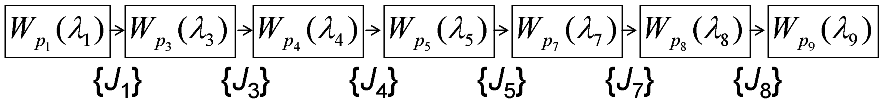


Example: RM (cont'd)

- Backlog dependency lists
  - Priority level 2



  - Priority level 1



Properties of RM

- Existence of ground jobs

Lemma 1: ***There exists at least one ground job in any hyperperiod.***

- Non-existence of base jobs

Lemma 2: ***For a non-ground job, there does not exist the base job.***

Summary on the Backlog Dependency

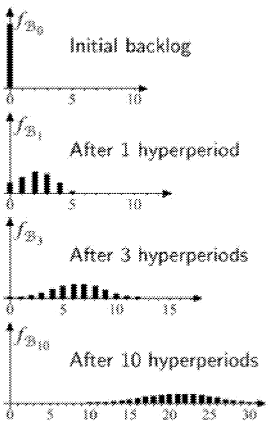
- For EDF, it is possible to build a complete backlog dependency tree.
  - If the SBD of the root node is given, all the other SBDs can be computed.
- For RM and DM, for each priority level, a separate backlog dependency list should be built.
  - If the SBD of the head node is given, all the other SBDs of ground jobs in the same level can be computed.

Markov Process Modeling

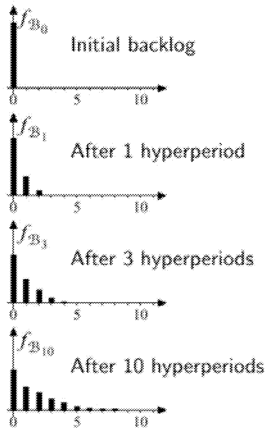
- Motivation
  - We want to know whether the stationary system backlog distribution (SSBD) can be analytically computed.
  - We observe that the backlog process { B_1, B_2, ... } is a Markov process.
    - B_k : the system backlog at the beginning of hyperperiod *k* (i.e., for the root or the head node)
- Advantage
  - We can obtain the exact solution for the SSBD, thus the exact response time distributions for all tasks.

Existence of the SSBD

$\overline{U} > 1$ (divergence)



$\overline{U} < 1$ (convergence)



Exact Solution

- We derive a set of equilibrium equations.

$$\mathbf{b}_k = \mathbf{P} \cdot \mathbf{b}_{k-1} \quad (k \to \infty)$$

a column vector representing the system backlog distribution at hyperperiod k

$$[P(B_k = 0), P(B_k = 1), P(B_k = 2), \ldots]^T$$

Markov matrix

$P(B_{k-1} = y \;\rightarrow\; B_k = x)$ *for any x, y*

  - BTW, the number of the derived linear equations is infinite, since the amount of the system backlog can infinitely increase as *k* → ∞.

Exact Solution (cont'd)

- We solve the infinite set of linear equations by deriving a finite set of equations, which is equivalent to the infinite set.
  - The Markov matrix has a regular structure.

$$\mathbf{P} = \begin{pmatrix}
0.8375 & 0.595 & 0.3275 & 0.13125 & 0.035 & 0.005 & - & - & \cdots \\
0.13125 & 0.2425 & 0.2675 & 0.19625 & 0.09625 & 0.03 & 0.005 & - & \ddots \\
0.03125 & 0.13125 & 0.2425 & 0.2675 & 0.19625 & 0.09625 & 0.03 & 0.005 & \ddots \\
- & 0.03125 & 0.13125 & 0.2425 & 0.2675 & 0.19625 & 0.09625 & 0.03 & \ddots \\
- & - & 0.03125 & 0.13125 & 0.2425 & 0.2675 & 0.19625 & 0.09625 & \ddots \\
- & - & - & 0.03125 & 0.13125 & 0.2425 & 0.2675 & 0.19625 & \ddots \\
- & - & - & - & 0.03125 & 0.13125 & 0.2425 & 0.2675 & \ddots \\
- & - & - & - & - & 0.03125 & 0.13125 & 0.2425 & \ddots \\
- & - & - & - & - & - & 0.03125 & 0.13125 & \ddots \\
- & - & - & - & - & - & - & 0.03125 & \ddots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \ddots
\end{pmatrix}$$

Exact Solution (cont'd)

- The closed form solution is given as follows:

$$\boldsymbol{\pi} = \mathbf{b}_\infty = [\underbrace{\pi_0, \pi_1, \pi_2, \ldots, \pi_{m_r}}_{\substack{\text{the solution obtained by} \\ \text{solving the derived finite set of} \\ \text{linear equations} \\ \text{(using the fact that } \pi_x \to 0 \text{ as } x \to \infty\text{)}}}, \underbrace{\pi_{m_r+1}, \pi_{m_r+2}, \ldots}_{\substack{\text{an infinite series generated} \\ \text{from the solution}}}]^T$$

$$\boxed{\pi_x = a_1\lambda_1^{x-m_r-1} + a_2\lambda_2^{x-m_r-1} + \cdots + a_{m_r}\lambda_{m_r}^{x-m_r-1}}$$

  - a_i : a linear combination of $\{\pi_0, \pi_1, \pi_2, \ldots, \pi_{m_r}\}$
  - λ_i : an eigenvalue obtained by diagonalizing a matrix derived from the regularity of the Markov matrix

Exact Solution (cont'd)

- The derived matrix A from the regularity of the Markov matrix P

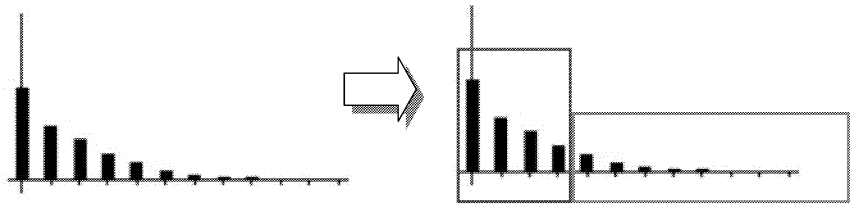
$$\mathbf{Q}_{x+1} = \mathbf{A} \cdot \mathbf{Q}_x \qquad x \geq m_r + 1$$

$$\mathbf{Q}_x = [\pi_{x-d}, \pi_{x-d+1}, \ldots, \pi_{x-1}, \pi_x, \pi_{x+1}, \ldots, \pi_{x-d+m_r-1}]^T \quad (d = m_r - r)$$

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -b_r(m_r)/b_r(0) & -b_r(m_r-1)/b_r(0) & \ldots & -b_r(d+1)/b_r(0) & 1-b_r(d)/b_r(0) & -b_r(d-1)/b_r(0) & \ldots & -b_r(1)/b_r(0) \end{pmatrix}$$

Safe Use of the Exact Solution

- We have to truncate the SSBD (which is infinitely long) in order to use it in a digital computer.



the infinitely long SSBD *preserved part* *dropped part*

  - However, the use of the truncated SSBD is safe.
    - it always gives an upper bound on the deadline miss probability for each task.

Approximated Solutions

- Markov matrix truncation method

$$\boldsymbol{\pi} = \mathbf{P} \cdot \boldsymbol{\pi} \longrightarrow \boldsymbol{\pi}' = \mathbf{P}' \cdot \boldsymbol{\pi}'$$

$$\boldsymbol{\pi}' = [\pi_0', \pi_1', \pi_2', \ldots, \pi_p']^T$$

$$\mathbf{P}'$$

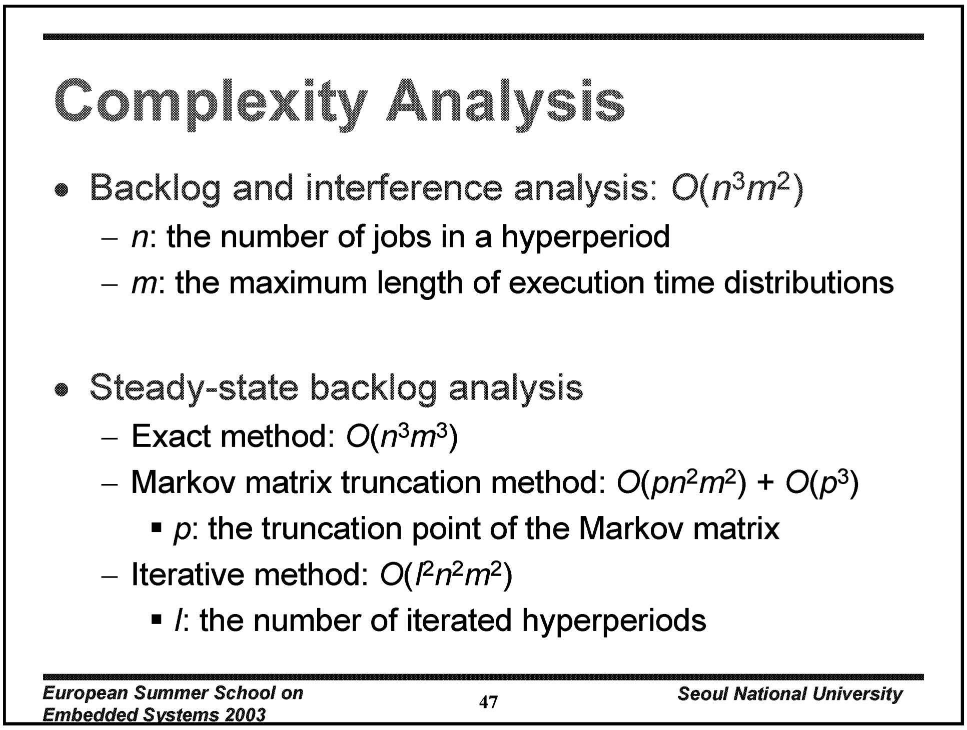
$$\mathbf{P} = \begin{pmatrix} 0,8375 & 0,595 & 0,3275 & 0,13125 & 0,035 & 0,005 & - & - & \cdots \\ 0,13125 & 0.2425 & 0,2675 & 0,19625 & 0,09625 & 0,03 & 0,005 & - & \ddots \\ 0,03125 & 0,13125 & 0,2425 & 0,2675 & 0,19625 & 0,09625 & 0,03 & 0,005 & \ddots \\ - & 0,03125 & 0,13125 & 0,2425 & 0,2675 & 0,19625 & 0,09625 & 0,03 & \ddots \\ - & - & 0,03125 & 0,13125 & 0,2425 & 0,2675 & 0,19625 & 0,09625 & \ddots \\ - & - & - & 0,03125 & 0,13125 & 0,2425 & 0,2675 & 0,19625 & \ddots \\ - & - & - & - & 0,03125 & 0,13125 & 0,2425 & 0,2675 & \ddots \\ - & - & - & - & - & 0,03125 & 0,13125 & 0,2425 & \ddots \\ - & - & - & - & - & - & 0,03125 & 0,13125 & \ddots \\ - & - & - & - & - & - & - & 0,03125 & \ddots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & & \ddots \end{pmatrix}$$

Approximated Solutions (cont'd)

- Iterative method
  - We can compute the system backlog distributions { B_1, B_2, ..., B_k, ...} in turn, until B_k converges.
    - that is, until || $B_k - B_{k-1}$ || falls below a threshold ε.
  - We can guess how close the computed solution is to the exact one, unlike the truncation method.
  - The convergence rate depends on the average system utilization.
    - If $\overline{U}$ is close to 1, the method is not applicable.

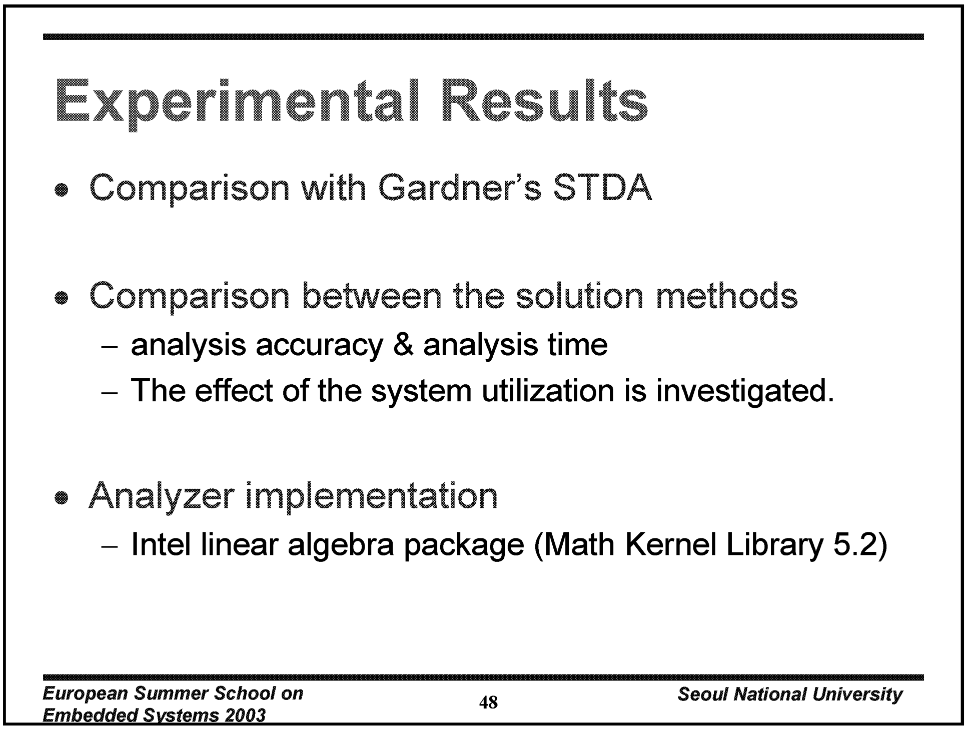
Special Case: No Markov process modeling

- $U^{max} < 1$
  - $B_1 = B_2 = ... = B_k = ... = 0$
  - The SSBD is equal to a null backlog distribution.
    - null backlog distribution: $P(B = 0) = 1$
  - It is possible to apply even a deterministic analysis.



Complexity Analysis

- Backlog and interference analysis: $O(n^3m^2)$
  - n: the number of jobs in a hyperperiod
  - m: the maximum length of execution time distributions
- Steady-state backlog analysis
  - Exact method: $O(n^3m^3)$
  - Markov matrix truncation method: $O(pn^2m^2) + O(p^3)$
    - p: the truncation point of the Markov matrix
  - Iterative method: $O(l^2n^2m^2)$
    - l: the number of iterated hyperperiods



Experimental Results

- Comparison with Gardner's STDA
- Comparison between the solution methods
  - analysis accuracy & analysis time
  - The effect of the system utilization is investigated.
- Analyzer implementation
  - Intel linear algebra package (Math Kernel Library 5.2)

Task sets

task set		T_i	D_i	execution times			utilizations		
				$C_i^{\min}$	$\bar{C}_i$	$C_i^{\max}$	$U^{\min}$	$\bar{U}$	$U^{\max}$
A	τ_1	20	20	4	6	10	.58	.82	1.27
	τ_2	60	60	12	16	22			
	τ_3	90	90	16	23	36			
B	τ_1	20	20	4	6	10	.58	.87	1.27
	τ_2	60	60	12	17	22			
	τ_3	90	90	16	26	36			
C	τ_1	20	20	4	7	10	.58	.92	1.27
	τ_2	60	60	12	17	22			
	τ_3	90	90	16	26	36			
C1	τ_1	20	20	3	7	11	.46	.92	1.38
	τ_2	60	60	10	17	24			
	τ_3	90	90	13	26	39			
C2	τ_1	20	20	2	7	12	.34	.92	1.50
	τ_2	60	60	8	17	26			
	τ_3	90	90	10	26	42			

only $\overline{U}$ is varied

U^{max} and U^{min} are varied

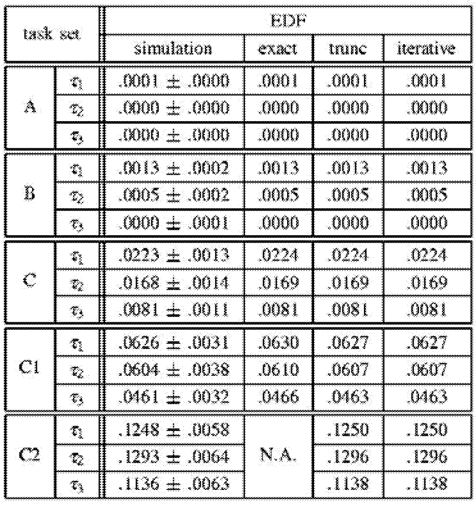
Analysis Accuracy

- deadline miss probability

task set		RM				
		simulation	STDA	exact	trunc	iterative
A	τ_1	.0000 ± .0000	.0000	.0000		
	τ_2	.0000 ± .0000	.0000	.0000		
	τ_3	.0940 ± .0025	.3931	.0940	.0940	.0940
B	τ_1	.0000 ± .0000	.0000	.0000		
	τ_2	.0000 ± .0000	.0000	.0000		
	τ_3	.2173 ± .0033	.6913	.2170	.2170	.2170
C	τ_1	.0000 ± .0000	.0000	.0000		
	τ_2	.0000 ± .0000	.0000	.0000		
	τ_3	.3849 ± .0052	.9075	.3852	.3852	.3852
C1	τ_1	.0000 ± .0000	.0000	.0000		
	τ_2	.0000 ± .0000	.0000	.0000		
	τ_3	.4332 ± .0065	.9209	.4334	.4334	.4334
C2	τ_1	.0000 ± .0000	.0000	.0000		
	τ_2	.0002 ± .0001	.0018	.0002	.0002	.0002
	τ_3	.4859 ± .0081	.9339	N.A.	.4860	.4860

Analysis Accuracy (cont'd)

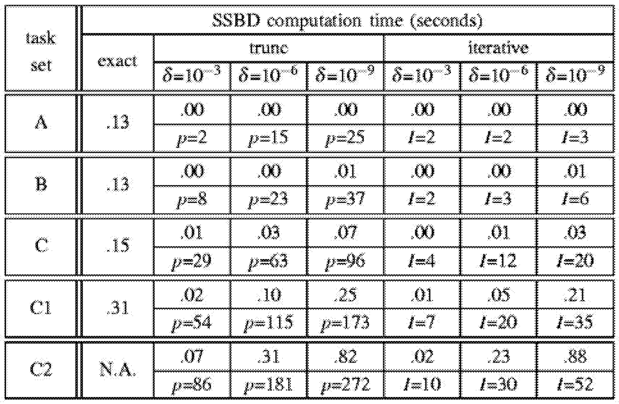
- deadline miss probability



task set		EDF			
		simulation	exact	trunc	iterative
A	τ_1	.0001 ± .0000	.0001	.0001	.0001
	τ_2	.0000 ± .0000	.0000	.0000	.0000
	τ_3	.0000 ± .0000	.0000	.0000	.0000
B	τ_1	.0013 ± .0002	.0013	.0013	.0013
	τ_2	.0005 ± .0002	.0005	.0005	.0005
	τ_3	.0000 ± .0001	.0000	.0000	.0000
C	τ_1	.0223 ± .0013	.0224	.0224	.0224
	τ_2	.0168 ± .0014	.0169	.0169	.0169
	τ_3	.0081 ± .0011	.0081	.0081	.0081
C1	τ_1	.0626 ± .0031	.0630	.0627	.0627
	τ_2	.0604 ± .0038	.0610	.0607	.0607
	τ_3	.0461 ± .0032	.0466	.0463	.0463
C2	τ_1	.1248 ± .0058	N.A.	.1250	.1250
	τ_2	.1293 ± .0064		.1296	.1296
	τ_3	.1136 ± .0063		.1138	.1138

Analysis Time (SSBD computation)

- accuracy level δ = || SSBD$_{exact}$ – SSBD$_{approx}$ ||



task set	SSBD computation time (seconds)						
	exact	trunc			iterative		
		$\delta=10^{-3}$	$\delta=10^{-6}$	$\delta=10^{-9}$	$\delta=10^{-3}$	$\delta=10^{-6}$	$\delta=10^{-9}$
A	.13	.00	.00	.00	.00	.00	.00
		p=2	p=15	p=25	I=2	I=2	I=3
B	.13	.00	.00	.01	.00	.00	.01
		p=8	p=23	p=37	I=2	I=3	I=6
C	.15	.01	.03	.07	.00	.01	.03
		p=29	p=63	p=96	I=4	I=12	I=20
C1	.31	.02	.10	.25	.01	.05	.21
		p=54	p=115	p=173	I=7	I=20	I=35
C2	N.A.	.07	.31	.82	.02	.23	.88
		p=86	p=181	p=272	I=10	I=30	I=52

Conclusion

- It is possible to analyze the exact response time distributions for periodic tasks.
  - Backlog dependency ⇒ covers both fixed-priority and dynamic-priority scheduling algorithms
  - Markov process modeling ⇒ enables the steady-state analysis
- The assumed task model should be extended to include arbitrary tasks
  - tasks with arbitrary interrelease times
- An approximated analysis should be developed to reduce the complexity.

Sponsored by:

ARTES
A network for Real-Time research and graduate Education in Sweden

ARTIST
http://www.artist-embedded.org/

BK21

CISS
CENTER FOR EMBEDDED SOFTWARE SYSTEMS

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE