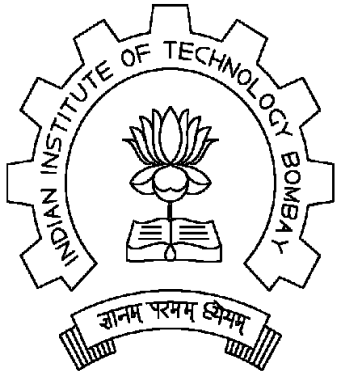
Real-Time Embedded Systems

Krithi Ramamritham

IIT Bombay



Embedded Systems?

Plan

- **Embedded Systems**
  - Introduction
  - Application Examples
- **Real-time support for ESW**

Embedded Systems

- **Single functional e.g. pager, mobile phone**
- **Tightly constrained**
  - cost, size, performance, power, etc.
- **Reactive & real-time**
  - e.g. car's cruise controller
  - delay in computation => failure of system

Why Is Embedded Software Not Just Software On Small Computers?

- **Embedded = Dedicated**
- **Interaction with physical processes**
  - sensors, actuators, processes
- **Critical properties are not all functional**
  - real-time, fault recovery, power, security, robustness
- **Heterogeneity**
  - hardware/software tradeoffs, mixed architectures
- **Concurrency**
  - interaction with multiple processes
- **Reactivity**
  - operating at the speed of the environment

Source:
Edward A. Lee, UC Berkeley
SRC/ETAB Summer Study 2001

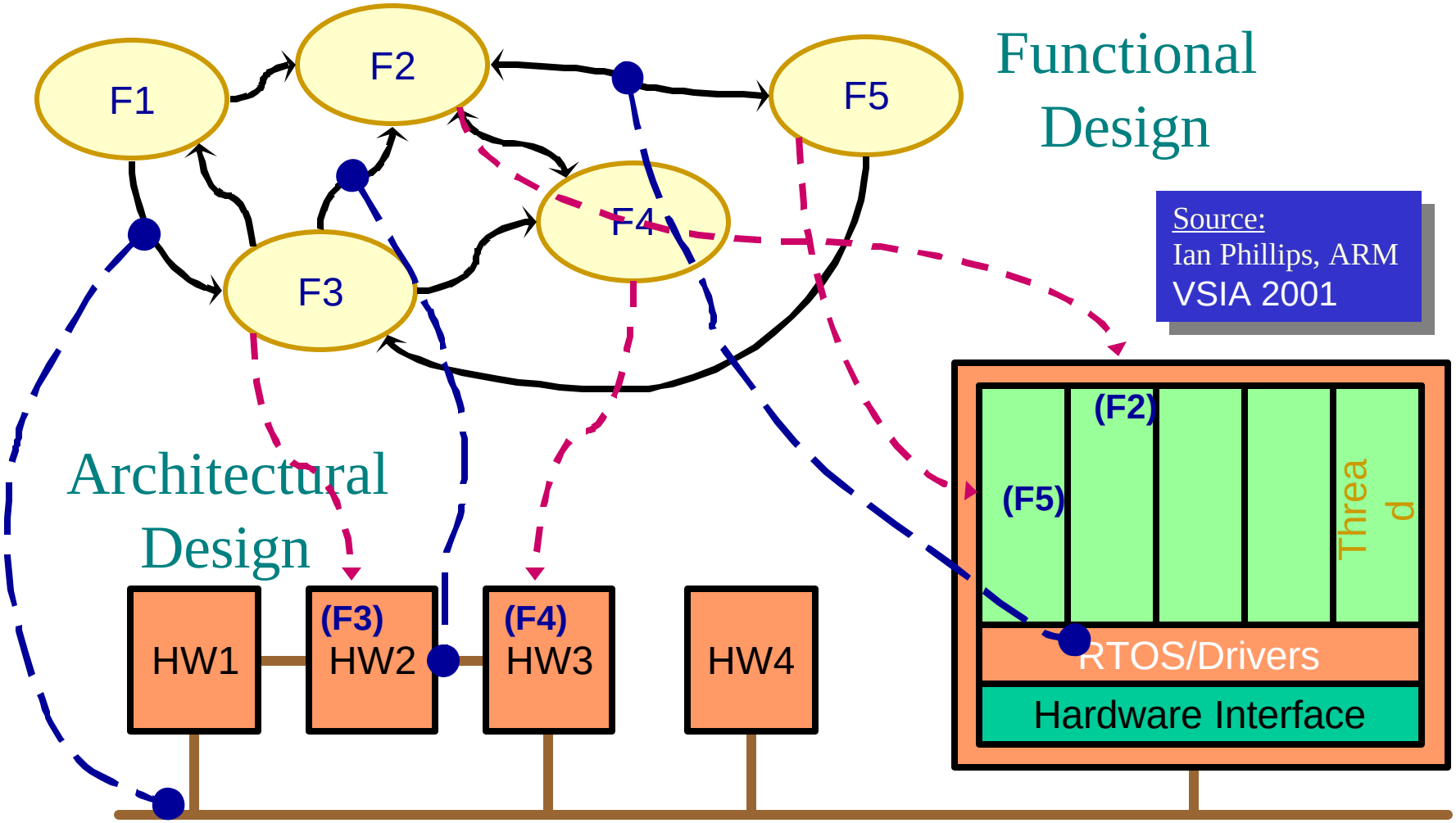
These features look more like hardware!

What is Embedded SW?

One definition:

"Software that is directly in contact with, or significantly affected by, the hardware that it executes on, or can directly influence the behavior of that hardware."

Functional Design & Mapping



Embedded Applications

An embedded system typically has a digital signal processor and a variety of I/O devices connected to sensors and actuators.

Computer (controller) is surrounded by other subsystems, sensors and actuators

Computer -- Controller's function is :

- to monitor parameters of physical processes of its surrounding system
- to control these processes whenever needed.

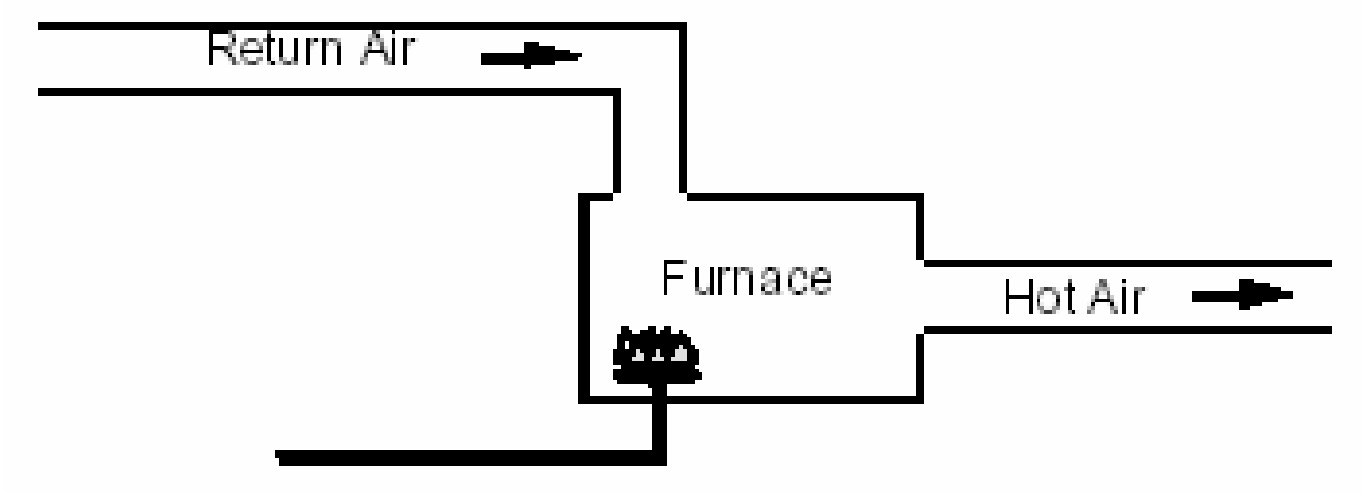
Simple Examples

A simple thermostat controller

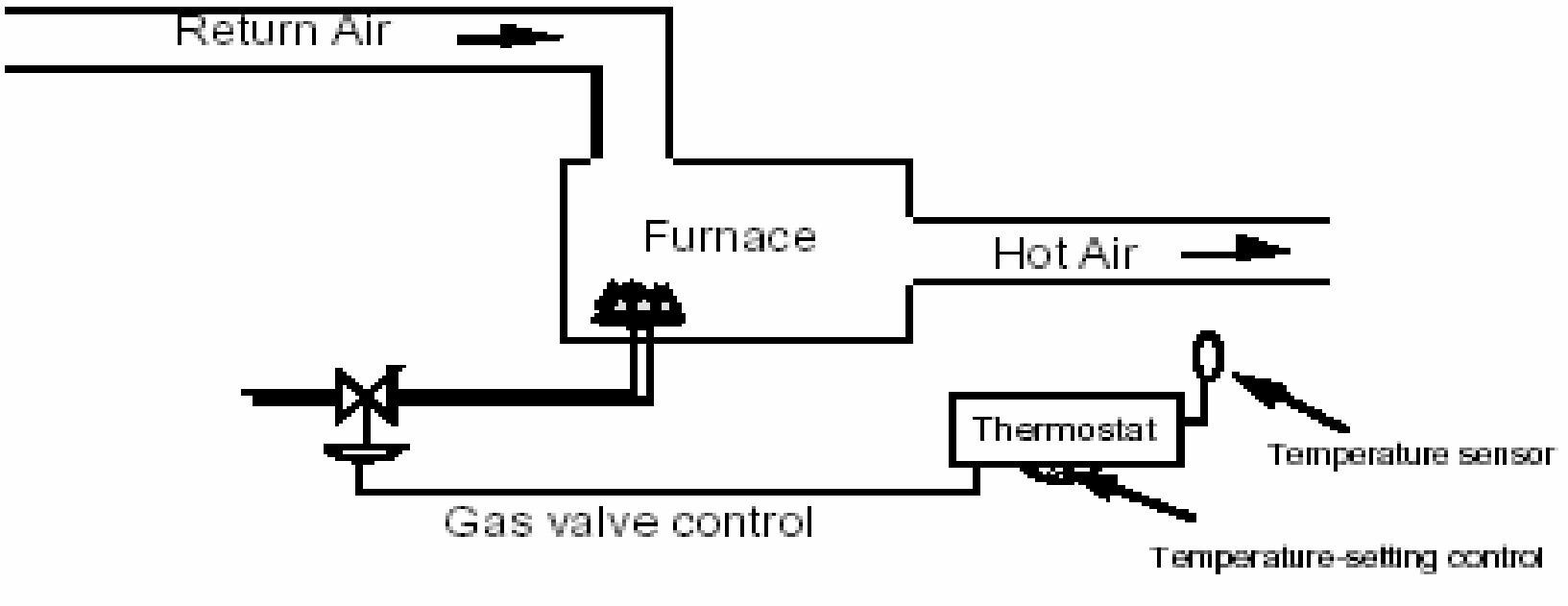
- **periodically reads the temperature of the chamber**
- **switches on or off the cooling system.**

a pacemaker

- **constantly monitors the heart**
- **paces the heart when heart beats are missed**

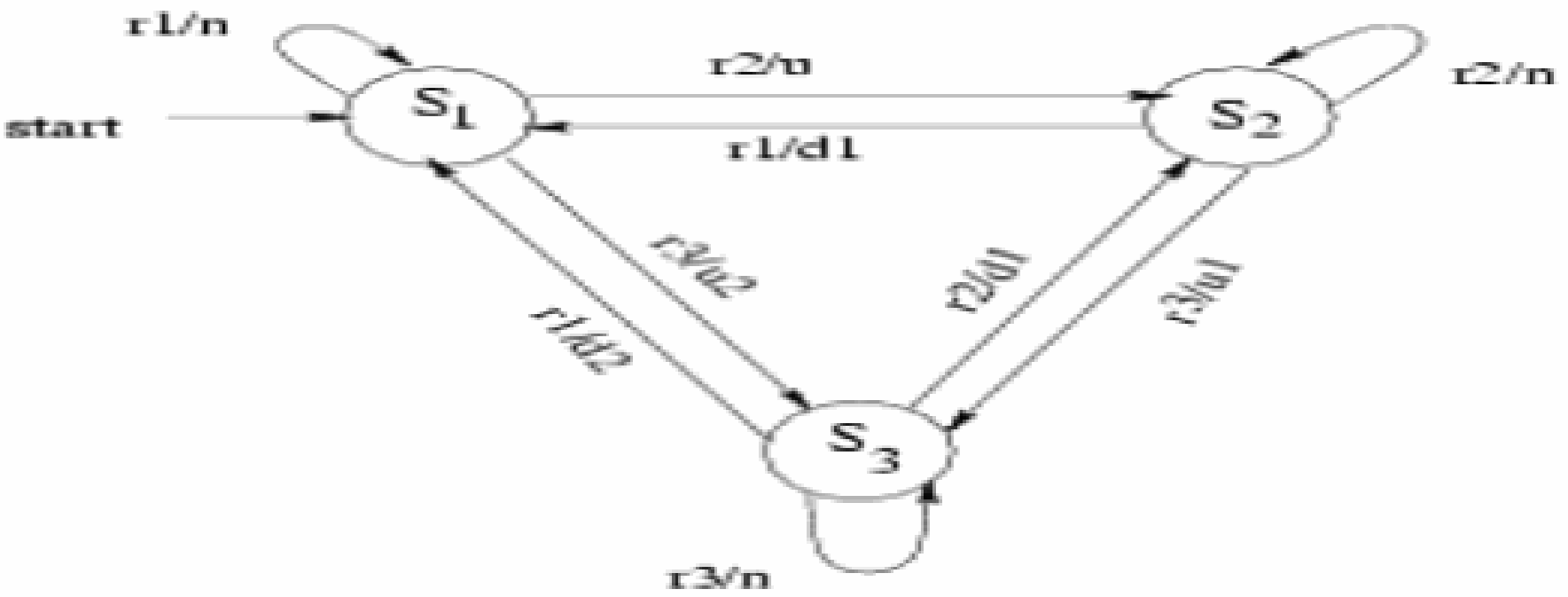


Open loop temperature control



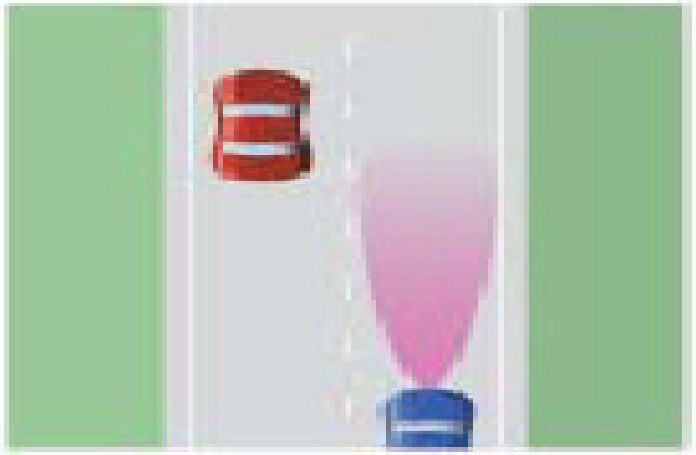
Closed loop temperature control

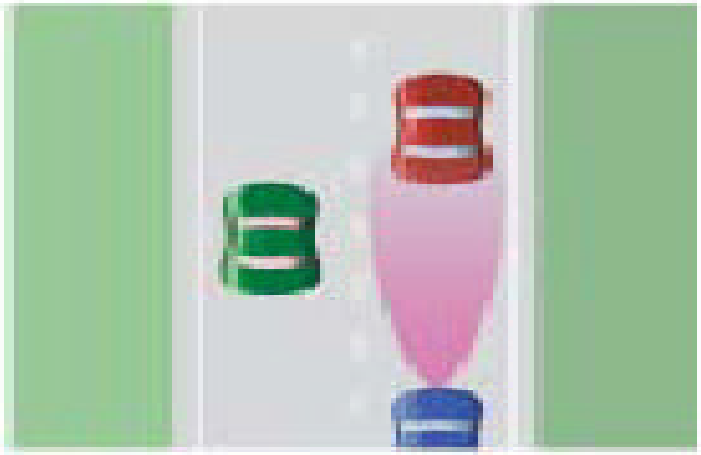
Example: Elevator Controller



- 3-floor elevator, S_i - in floor i
- ri - request from floor i
- uj,dj - up/down by j floors
- This machine is nondeterministic
- In S_2 when requests r1 and r3 arrive.
- Non-determinism due to abstraction

Adaptive Cruise Control with Driver Alert





- Helps to reduce the need for drivers to manually adjust speed or disengage cruise control when encountering Slower traffic.
- Automatically manages vehicle speed to maintain a distance set by the driver.
- Alerts drivers when slower traffic is detected in the path.
- Audible and visual alerts warn the driver when braking is necessary to avoid slower moving vehicles ahead.
- Drivers can adjust system sensitivity to their preferred driving style.

Plan

- **Embedded Systems**
- **Real-Time Support**
  - Special Characteristics of Real-Time Systems
  - Real-Time Constraints
  - Canonical Real-Time Applications
  - Scheduling in Real-time systems
  - Operating System Approaches

What is "real" about real-time?

computer world
e.g., PC

average response for user, interactive

occasionally longer

***reaction:* user annoyed**

computer controls speed of user

"computer time"

real world
industrial system, airplane

events occur in environment at own speed

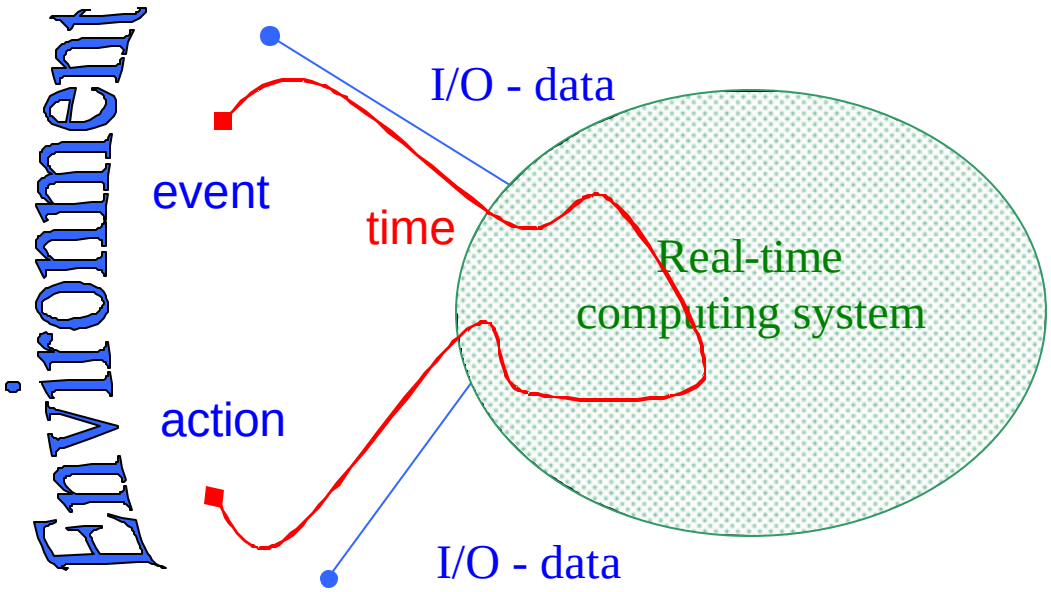
reaction too slow: deadline miss

***reaction:* damage, pot. loss of human life**

computer must follow speed of environment

"real-time"

Real-Time Systems



A real-time system is a system that reacts to events in the environment by performing predefined actions

<u>within specified time intervals.</u>

Real-Time Systems: Properties of Interest

- ***Safety*: Nothing bad will happen.**
- ***Liveness*: Something good will happen.**
- ***Timeliness*: Things will happen on time -- by their deadlines, periodically, ...**

In a Real-Time System....

Correctness of results depends on value
and its time of delivery

correct value* delivered *too late* is *incorrect

e.g., traffic light: light must be green *when crossing*, not enough before

***Real*-time:**

(Timely) reactions to events *as they occur*, at their pace:

(real-time) system (internal) time same time scale as environment (external) time

Performance Metrics in Real-Time Systems

- Beyond minimizing response times and increasing the throughput:

  – *achieve timeliness*.

- More precisely, how well can we predict that deadlines will be met?

Types of RT Systems

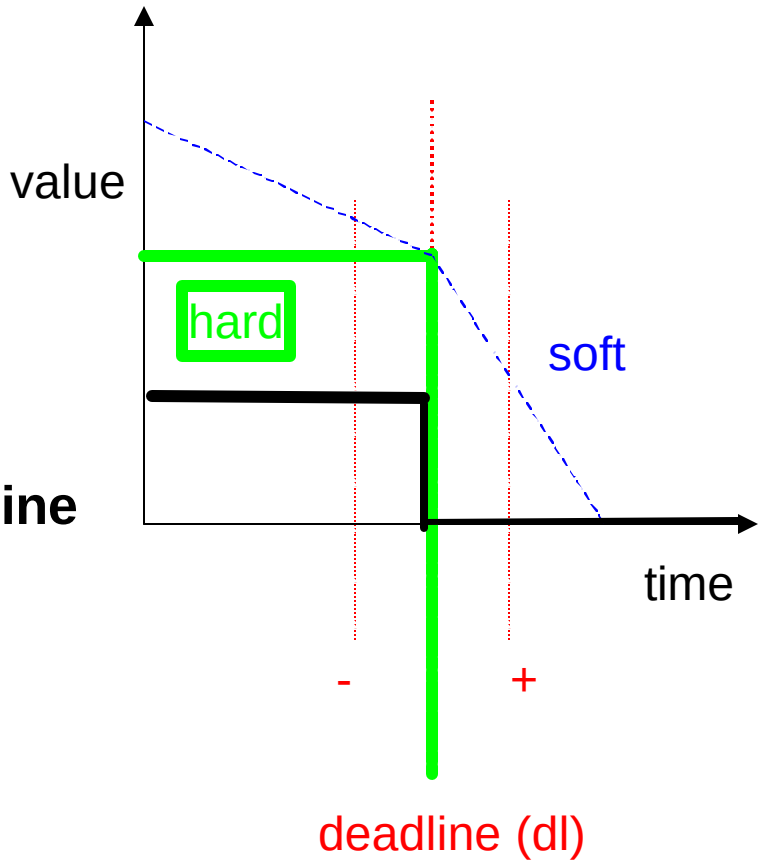
Dimensions along which real-time activities can be categorized:

- *how tight are the deadlines?* --deadlines are tight when the laxity (deadline -- computation time) is small.
- *how strict are the deadlines?* what is the value of executing an activity after its deadline?
- *what are the characteristics of the environment?* how static or dynamic must the system be?

Designers want their real-time system to be *fast, predictable, reliable, flexible*.

Hard, soft, firm

- **Hard**
  result useless or dangerous if deadline exceeded
- **Soft**
  result of some - lower - value if deadline exceeded
- **Firm**
  If value drops to zero at deadline



Deadline intervals:
result required not later *and not before*

Examples

- **Hard real time systems**
  - Aircraft
  - Airport landing services
  - Nuclear Power Stations
  - Chemical Plants
  - Life support systems
- **Soft real time systems**
  - Mutlimedia
  - Interactive video games

Real-Time: Items and Terms

Task

- program, perform service, functionality
- requires resources, e.g., execution time

Deadline

- specified time for completion of, e.g., task
- time interval or absolute point in time
- value of result may depend on completion time

Plan

- Special Characteristics of Real-Time Systems
- **Real-Time Constraints**
- **Canonical Real-Time Applications**
- **Scheduling in Real-time systems**
- **Operating System Approaches**

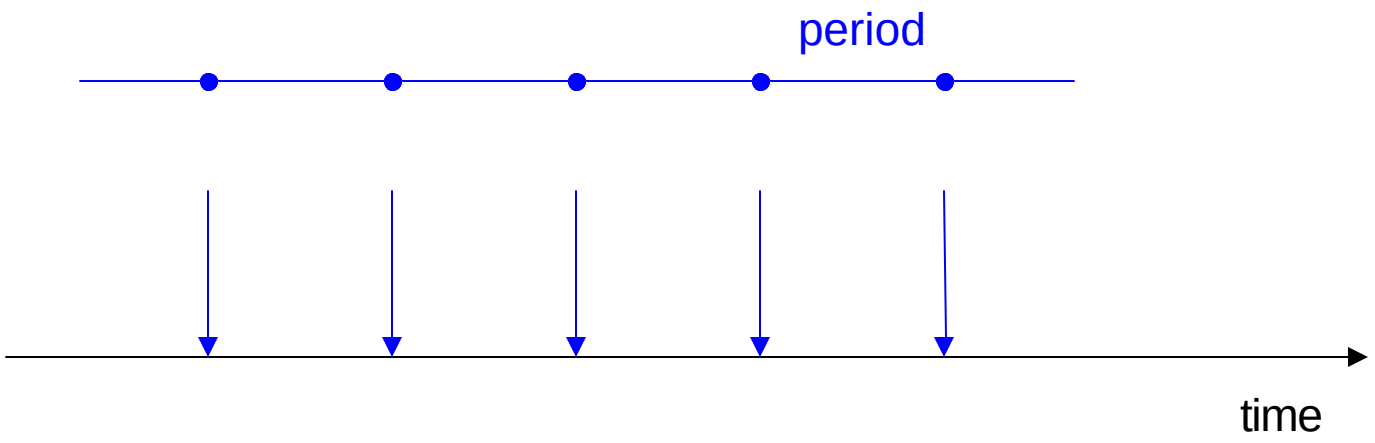
Timing Constraints

Real-time means to be in time ---

how do we know something is "in time"?
how do we express that?

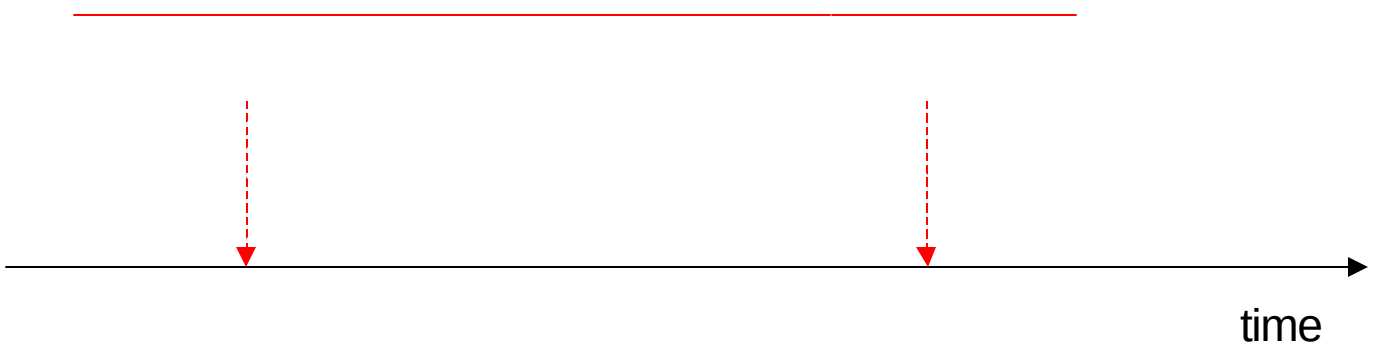
- *Timing constraints* **are used to specify temporal correctness**
  e.g., "finish assignment by 2pm", "be at station before train departs".
- **A system is said to be *(temporally) feasible*, if it meets all specified timing constraints.**
- **Timing constraints do not come out of thin air: *design* process identifies *events*, derives, models, and finally *specifies* timing constraints**

- **Periodic**
  - activity occurs repeatedly
  - e.g., to monitor environment values, temperature, etc.



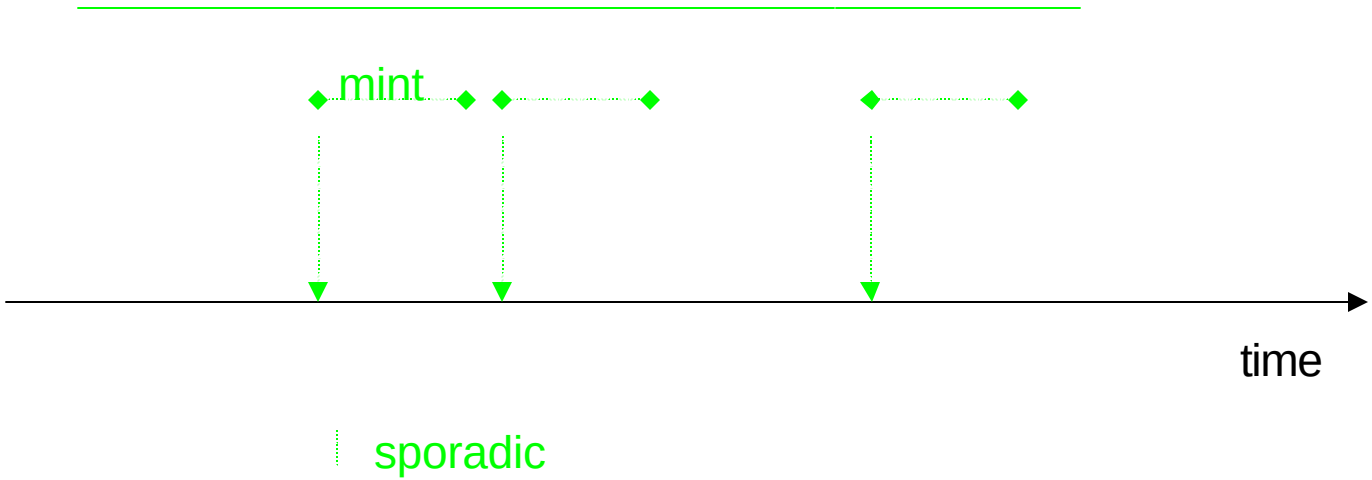
periodic

- **Aperiodic**
  - can occur any time
  - no arrival pattern given



aperiodic

- **Sporadic**
  - can occur any time, but
  - minimum time between arrivals



Who initiates (triggers) actions?

Example: Chemical process

- controlled so that temperature stays below *danger* level
- warning is triggered before danger point

  …… so that cooling can still occur

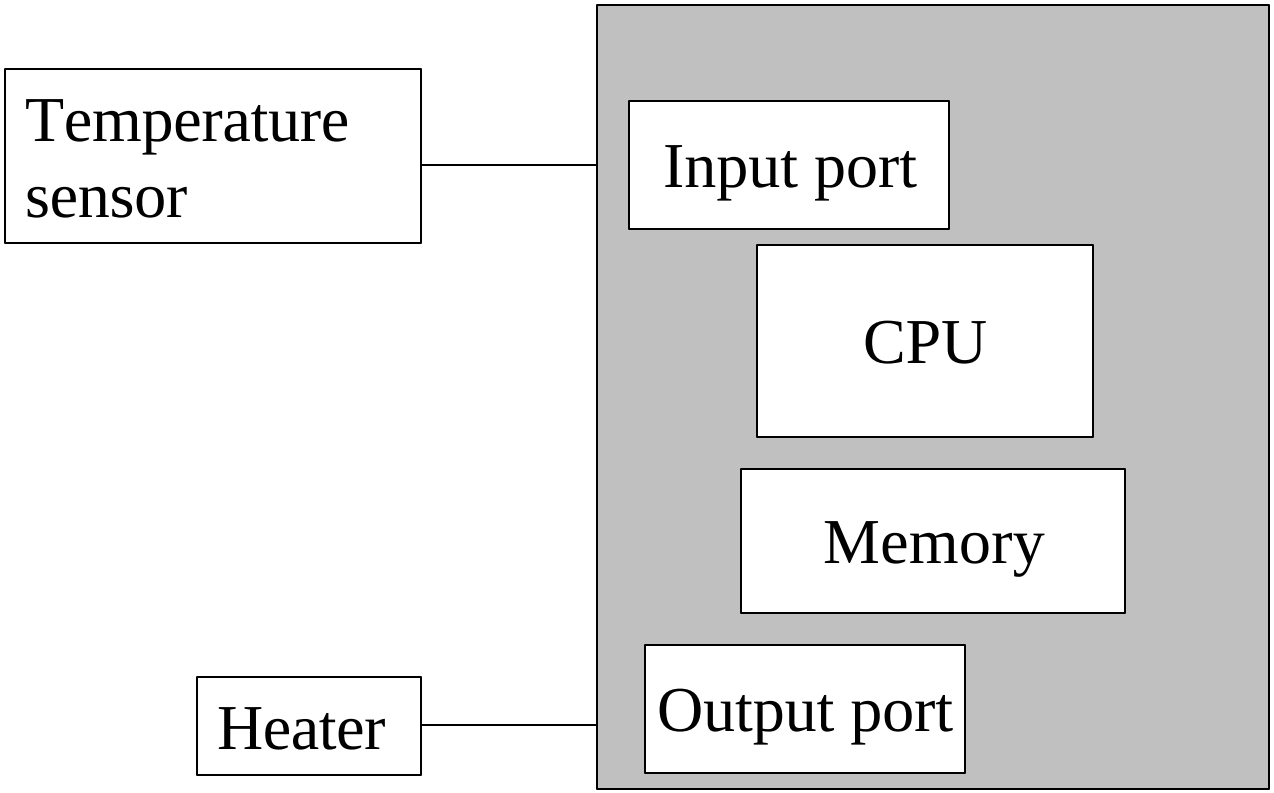
Two possibilities:

- action whenever temp raises above *warn;*
  event triggered
- look every *int* time intervals; action when temp if measures above *warn*
  time triggered

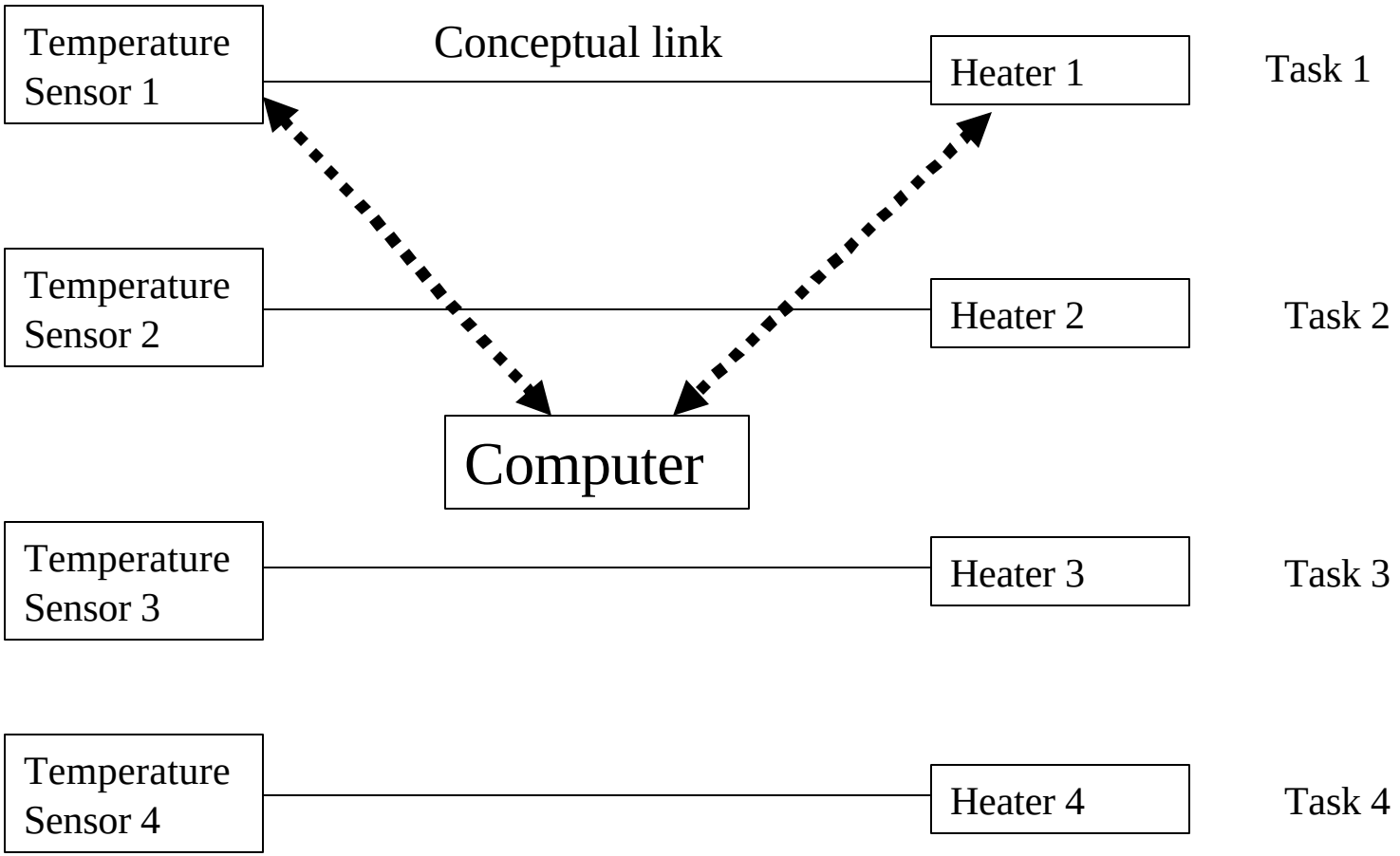
Plan

- Special Characteristics of Real-Time Systems
- Real-Time Constraints
- **Canonical Real-Time Applications**
- **Scheduling in Real-time systems**
- **Operating System Approaches**

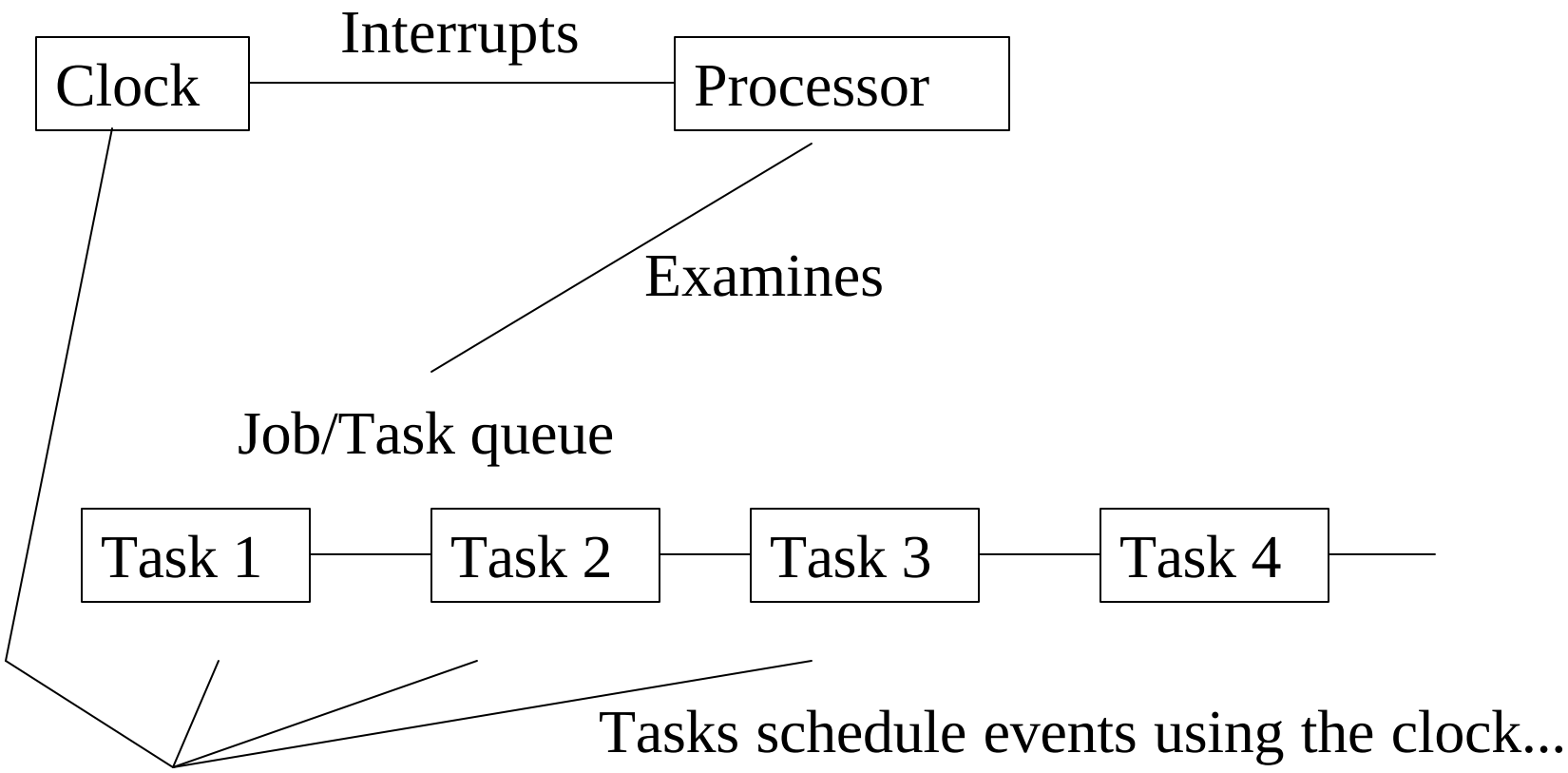
A Typical Real time system



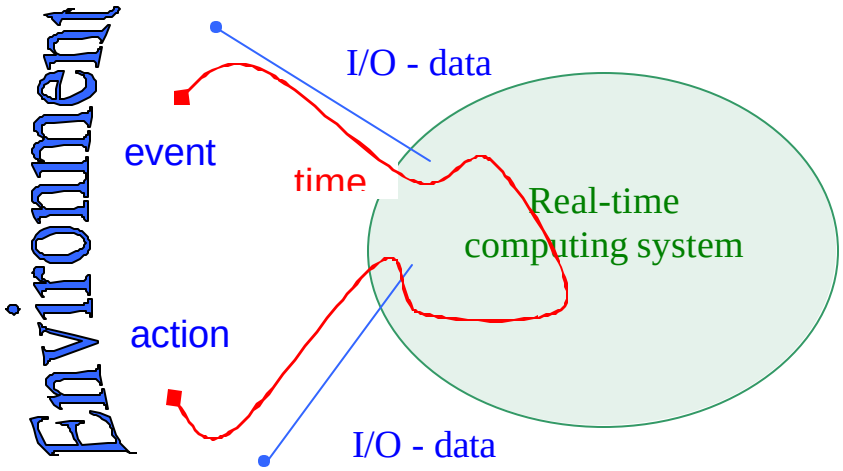
Extended polling example



Clock, interrupts, tasks



Why is scheduling important?



Definition:

A real-time system is a system that reacts to events in the environment by performing predefined actions within specified time intervals.

Schedulability analysis

a.k.a. *feasibility checking:*

check whether tasks will meet their

***timing constraints*.**

Scheduling Paradigms

Four scheduling paradigms emerge, depending on

- **whether a system performs schedulability analysis**
- **if it does,**
  - whether it is done statically or dynamically
  - whether the result of the analysis itself produces
    a schedule or plan according to which
    tasks are dispatched at run-time.

1. Static Table-Driven Approaches

- **Perform static schedulability analysis by checking if a schedule is derivable.**
- **The resulting schedule (table) identifies the start times of each task.**
- **Applicable to tasks that are periodic (or have been transformed into periodic tasks by well known techniques).**
- **This is highly predictable but, highly inflexible.**
- **Any change to the tasks and their characteristics may require a complete overhaul of the table.**

2. Static Priority Driven Preemptive Approaches

- **Tasks have -- systematically assigned -- static priorities.**
- **Priorities take timing constraints into account:**
  - e.g. RMA: Rate-Monotonic ---- the lower the period, the higher the priority.
  - e.g. EDF: Earliest-deadline-first --- the earlier the deadline, the higher the priority.

- **Perform static schedulability analysis but no explicit schedule is constructed**
  - RMA - Sum of task Utilizations <= ln 2.
  - EDF - Sum of task Utilizations <= 1

Task utilization = computation-time / Period

- **At run-time, tasks are executed highest-priority-first, with preemptive-resume policy.**
- **When resources are used, need to compute worst-case blocking times.**

Static Priorities: Rate Monotonic Analysis

presented by Liu and Layland in 1973

Assumptions

- **Tasks are periodic with deadline equal to period. Release time of tasks is the period start time.**
- **Tasks do not suspend themselves**
- **Tasks have bounded execution time**
- **Tasks are independent**
- **Scheduling overhead negligible**

RMA: Design Time vs. Run Time

At Design Time:

Tasks priorities are assigned according to their periods; shorter period means higher priority

Schedulability test

Taskset is schedulable if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

Very simple test, easy to implement.

Run-time

The ready task with the highest priority is executed.

RMA: Example

taskset: t1, t2, t3, t4

t1 = (3, 1)
t2 = (6, 1)
t3 = (5, 1)
t4 = (10, 2)

The schedulability test:

$$1/3 + 1/6 + 1/5 + 2/10 = 4\,(2^{(1/4)} - 1)\ ?$$

$$0.9 < 0.75\ ?$$

…. not schedulable

RMA...

A schedulability test is

- Sufficient:
  there *may* exist tasksets that *fail* the test, but are *schedulable*
- Necessary:
  tasksets that *fail* are (definitely) *not schedulable*

The RMA schedulability test is *sufficient*, but not *necessary.*

e.g., when periods are harmonic,
i.e., multiples of each other, utilization can be 1.

Exact RMA

by Joseph and Pandya, based on critical instance analysis

(longest response time of task, when it is released at same time as all higher priority tasks)

What is happening at the critical instance?

- Let T_1 be the highest priority task. Its response time

  $R_1 = C_1$ since it cannot be preempted

- What about T_2 ?

  $R_2 = C_2$ + delays due to interruptions by T_1.

  Since T_1 has higher priority, it has shorter period. That means it will interrupt T_2 at least once, probably more often. Assume T_1 has half the period of T_2,

  $R_2 = C_2 + 2 \times C_1$

3. Dynamic Planning based Approaches

- Feasibility is checked at run-time -- a dynamically arriving task is accepted only if it is feasible to meet its deadline.
  - Such a task is said to be guaranteed to meet its time constraints
- One of the results of the feasibility analysis can be a schedule or plan that determines start times

- Has the flexibility of dynamic approaches with some of the predictability of static approaches

- If feasibility check is done sufficiently ahead of the deadline, time is available to take alternative actions.

4. Dynamic Best-effort Approaches

- The system tries to do its best to meet deadlines.
- But since no guarantees are provided, a task may be aborted during its execution.
- Until the deadline arrives, or until the task finishes, whichever comes first, one does not know whether a timing constraint will be met.

- Permits any reasonable scheduling approach, EDF, Highest-priority,…

Cyclic scheduling

- **Ubiquitous in large-scale dynamic real-time systems**
- **Combination of both table-driven scheduling and priority scheduling.**
- **Tasks are assigned one of a set of harmonic periods.**
- **Within each period, tasks are dispatched according to a table that just lists the order in which the tasks execute.**
- **Slightly more flexible than the table-driven approach**
- **no start times are specified**
- **In many actual applications, rather than making worse-case assumptions, confidence in a cyclic schedule is obtained by very elaborate and extensive simulations of typical scenarios.**

Plan

- **Special Characteristics of Real-Time Systems**
- **Real-Time Constraints**
- **Canonical Real-Time Applications**
- **Scheduling in Real-time systems**
- **Operating System Approaches**

Real-Time Operating Systems

Support process management and synchronization, memory management, interprocess communication, and I/O.

Three categories of real-time operating systems:

small, proprietary kernels.

e.g. VRTX32, pSOS, VxWorks

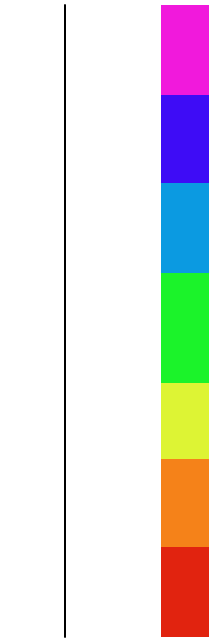
real-time extensions to commercial timesharing operatin systems.

e.g. RT-Linux, RT-NT

research kernels

e.g. MARS, ARTS, Spring, Polis

Real-Time Applications Spectrum



Real-Time Operating System

VxWorks, Lynx, QNX, ...

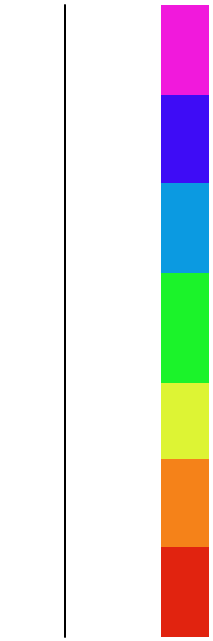
Intime, HyperKernel, RTX

Windows CE

General-Purpose Operating System

Windows NT

Real-Time Applications Spectrum



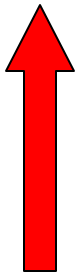
Real-Time Operating System

VxWorks, Lynx, QNX, ...
Intime, HyperKernel, RTX

Windows CE

Windows NT

General-Purpose Operating System



Embedded (Commercial) Kernels

Stripped down and optimized versions of timesharing operating systems.

- **Intended to be fast**
  - a fast context switch,
  - external interrupts recognized quickly
  - the ability to lock code and data in memory
  - special sequential files that can accumulate data at a fast rate
- **To deal with timing requirements**
  - a real-time clock with special alarms and timeouts
  - bounded execution time for most primitives
  - real-time queuing disciplines such as earliest deadline first,
  - primitives to delay/suspend/resume execution
  - priority-driven best-effort scheduling mechanism or a table-driven mechanism.
- **Communication and synchronization via mailboxes, events, signals, and semaphores.**

Real-Time Extensions to General Purpose Operating Systems

E.g., extending LINUX to RT-LINUX, NT to RT-NT

- Advantage:
  - based on a set of familiar interfaces (standards) that speed development and facilitate portability.
- Disadvantages
  - Too many basic and inappropriate underlying assumptions still exist.

Using General Purpose Operating Systems

- **GPOS offer some capabilities useful for real-time system builders**
- **RT applications can obtain leverage from existing development tools and applications**
- **Some GPOSs accepted as de-facto standards for industrial applications**

Real Time Linux approaches

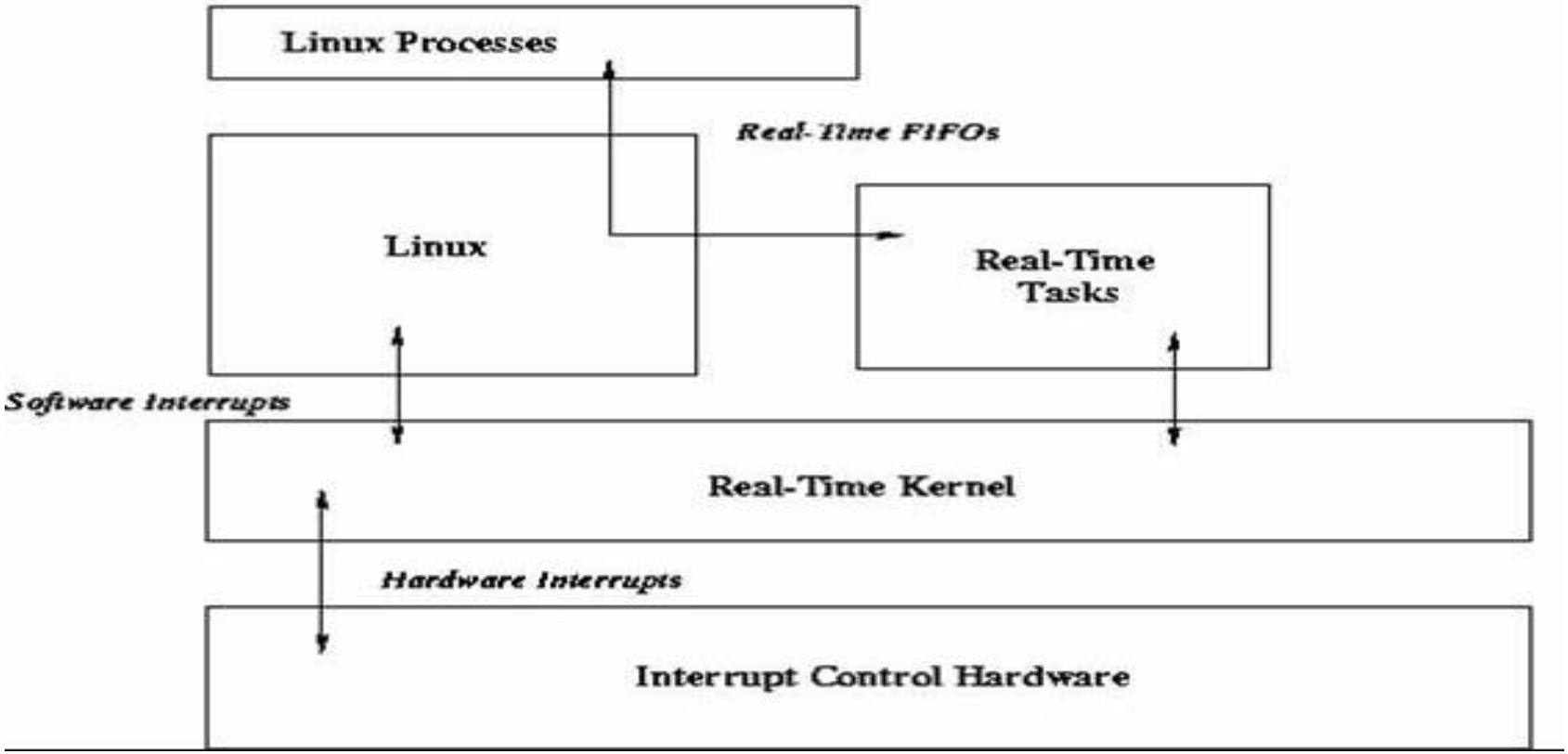
1. **Modify the current Linux kernel to handle RT constraints**
   - Used by KURT
2. **Make the standard Linux kernel run as a task of the real-time kernel**
   - Used by RT-Linux, RTAI

Modifying Linux kernel

- **Advantages**
  - Most problems, such as interrupt handling, already solved
  - Less initial labor
- **Disadvantages**
  - No guaranteed performance
  - RT tasks don't always have precedence over non-RT tasks.

Running Linux as a process of a second RT kernel

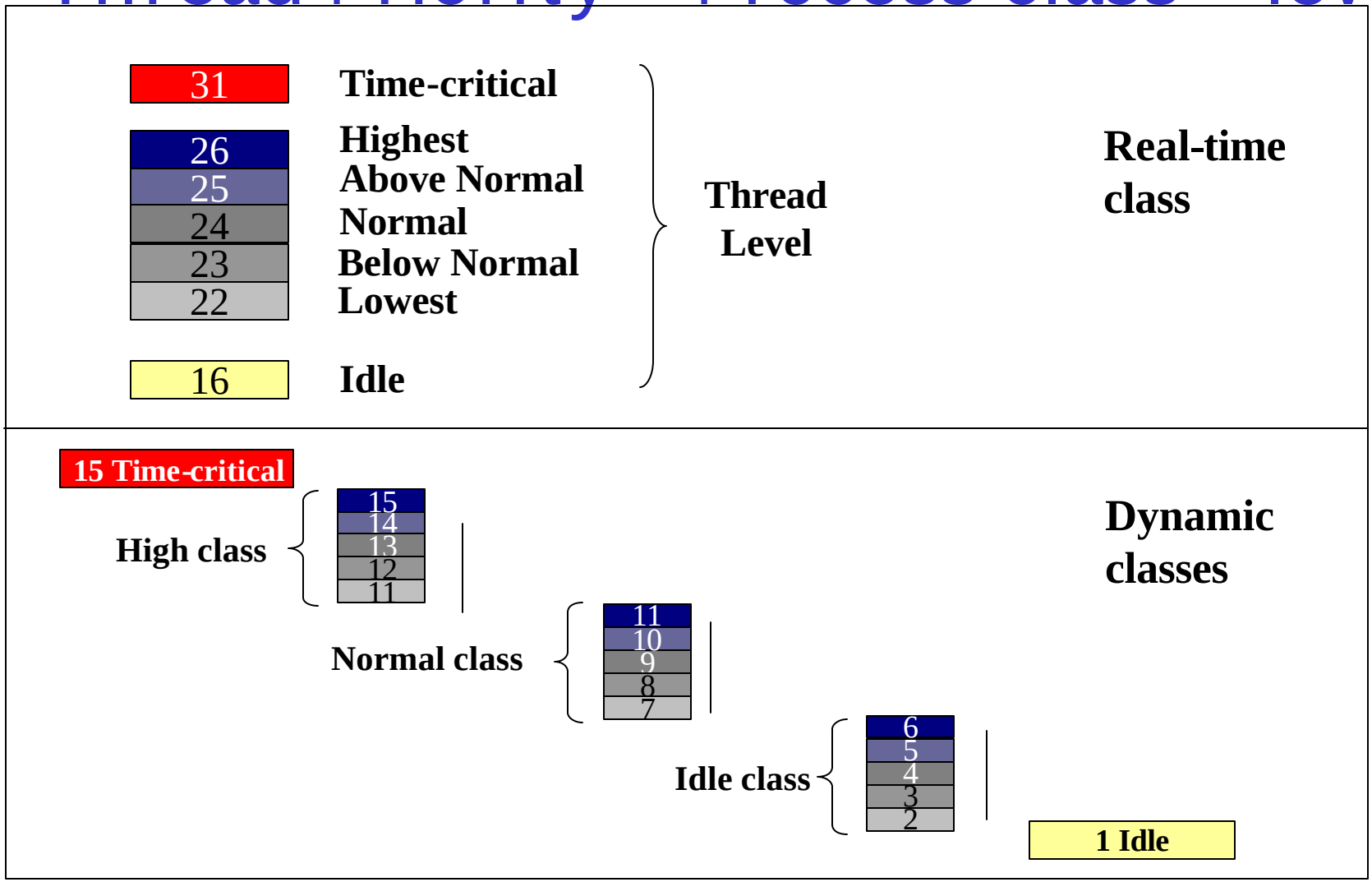
- **Advantages**
  - –Can make hard real time guarantees
  - –Easy to implement a new scheduler
- **Disadvantages**
  - –Initial port difficult, must know a lot about underlying hardware
  - –Running a small real-time executive is not a substitute for a full-fledged RTOS



GPOS -- for RT applications?

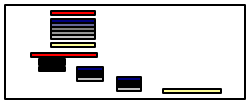
- **Scheduling and priorities**
  - Preemptive, priority-based scheduling
    non-degradable priorities
    priority adjustment
  - No priority inheritance
  - No priority tracking
  - Limited number of priorities
  - No explicit support for guaranteeing timing constraints

Thread Priority = Process class + level



Scheduling Priorities

- Threads scheduled by executive.
- Priority based preemptive scheduling.

Interrupts
Deferred Procedure Calls (DPC)
System and user-level threads 

GPOS -- for RT applications? (contd.)

- Quick recognition of external events
  - **Priority inversion due to Deferred Procedure Calls (DPC)**
- I/O management
- Timers granularity and accuracy
  - **High resolution counter with resolution of 0.8 msec.**
  - **Periodic and one shot timers with resolution of 1 msec.**
- Rich set of synchronization objects and communication mechanisms.
  - **Object queues are FIFO**

Research Operating Systems

- **MARS – static scheduling**
- **ARTS – static priority scheduling**
- **Spring –dynamic guarantees**

MARS -- TU, Vienna (Kopetz)

Offers support for controlling a distributed application based entirely on time events (rather than asynchronous events) from the environment.

- **A priori static analysis to demonstrate that all the timing requirements are met.**
- **Uses flow control} on the maximum number of events that the system handles.**
- **Based on the time driven model -- assume everything is periodic.**
- **Static table-driven scheduling approach**
- **A hardware based clock synchronization algorithm**
- **A TDMA-like protocol to guarantee timely message delivery**

ARTS -- CMU (Tokuda, et al)

- **The ARTS kernel provides a distributed real-time computing environment.**
- **Works in conjunction with the static priority driven preemptive scheduling paradigm.**
- **Kernel is tied to various tools that a priori analyze schedulability.**
- **The kernel supports the notion of real-time objects and real-time threads.**
- **Each real-time object is time encapsulated -- a time fence mechanism:The time fence provides a run time check that ensures that the slack time is greater than the worst case execution time for an object invocation**

SPRING – Umass. (Ramamritham & Stankovic)

- **Real-time support for multiprocessors and distributed sys**
- **Strives for a more flexible combination of off-line and on-line techniques**
  - Safety-critical tasks are dealt with via static table-driven scheduling.
  - Dynamic planning based scheduling of tasks that arrive dynamically.
- **Takes tasks' time and resource constraints into account and avoids the need to a priori compute worst case blocking times**
- **Reflective kernel retains a significant amount of application semantics at run time – provides flexibility and graceful degradation.**

Polis: Synthesizing OSs

- Given a FSM description of a RT application
- Each FSM becomes a task
- Signals, Interrupts, and polling
- Tasks with waiting inputs handled in FIFS order (priority order – TB done)
- Some interrupts can be made to directly execute the corresponding task
- Needed OS execute synthesized based on just what is needed

References

This is a short version of a semester-long course.

Visit http://www.it.iitb.ac.in/~it606

for all the material from that course

- **Jack Ganssle, "The Art of Designing Embedded Systems", Newnes, 1999.**
- **David Simon, "An Embedded Software Primer", Addison Wesley, 2000.**
- **C.M. Krishna and Kang G. Shin, "RTS: Real-Time Systems", McGraw-Hill, 1997, ISBN 0-07-057043.**
- **Frank Vahid, Tony Givargis, "Embedded System Design: A Unified Hardware/ Software Introduction", John Wiley & Sons Inc., 2002.**
- **J. A. Stankovic, and K. Ramamritham, Advances in Hard Real-Time Systems, *IEEE Computer Society Press*, Washington DC, September 1993, 777 pages.**

References…

- K. Ramamritham and J. A. Stankovic, Scheduling Scheduling Algorithms and Operating Systems Support for Real-Time Systems, invited paper, *Proceedings of the IEEE*, Jan 1994, pp. 55-67.
- Sundeep Kapila, K. Ramamritham, Sudhakar, Distributed Real-Time Embedded Applications using Off-the-Shelf Components? Experiences Building a Flight Simulator, *IEEE/IEE Real-Time Embedded Systems Workshop (held in conjunction with the IEEE Real-Time Systems Symposium)*, December 2001.
- Real-Time Linux, http://www.fsmlabs.com/articles/archive/usenix.pdf
- Handel-C material based on "Handel-C Language Reference Manual", Celoxica Ltd.

References…

- David Harel, Hagi Lachover, Ammon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot, Statemate: A working Environment for the Development of Complex Reactive Systems, *IEEE Transactions on Software Engineering, Vol 16 No. 4*, April 1999.
- Ptolemy Project, http://ptolemy.eecs.berkeley.edu/.
- S. Ramesh and P. Bhaduri, Validation of Pipelined processors using Esterel Tools: A Case study, Proc. of Computer Aided Verification, LNCS Vol. 1633, 1999. (pdf version).