EECS 507: Introduction to Embedded Systems Research
Scheduling, embedded OSs, and CPS

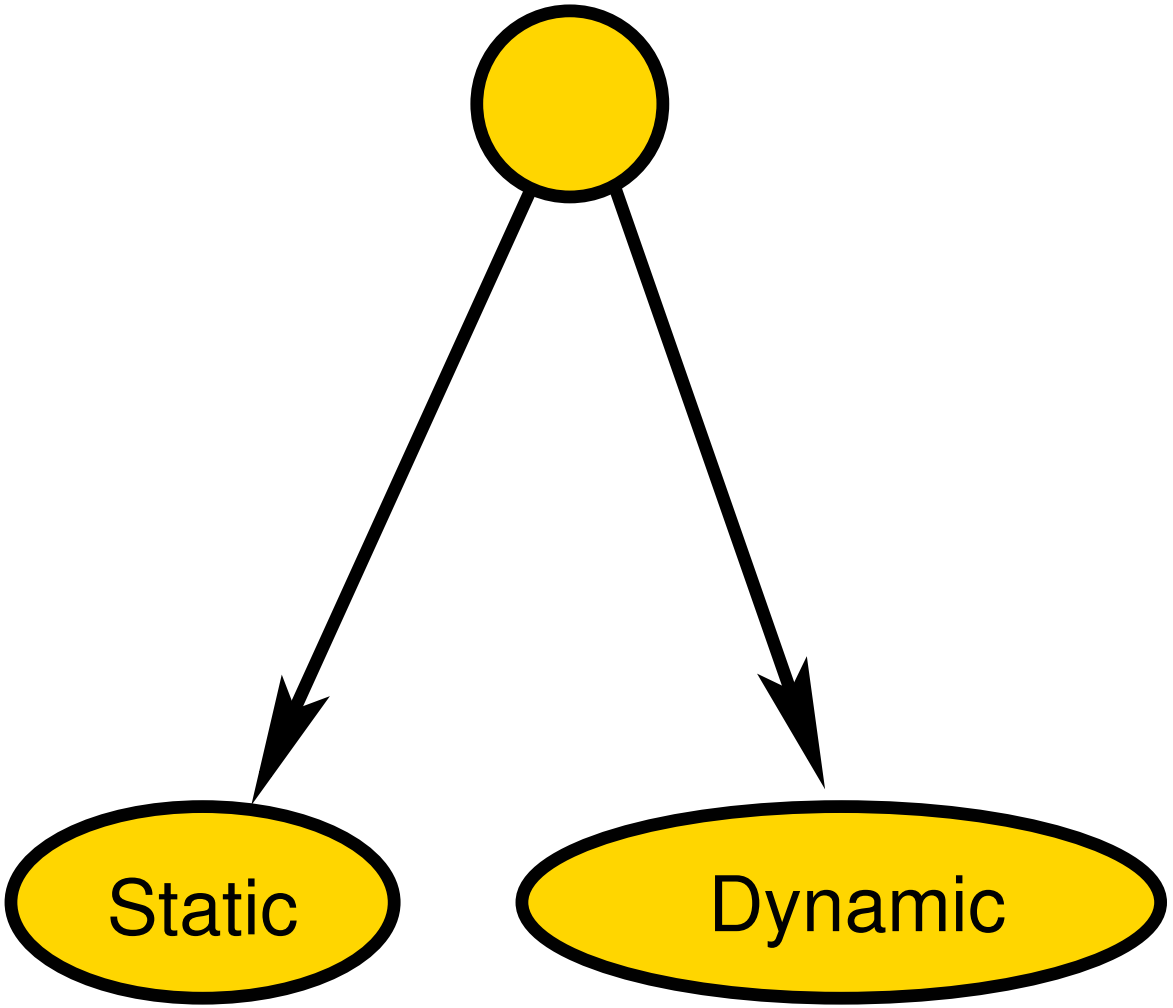
Robert Dick

University of Michigan

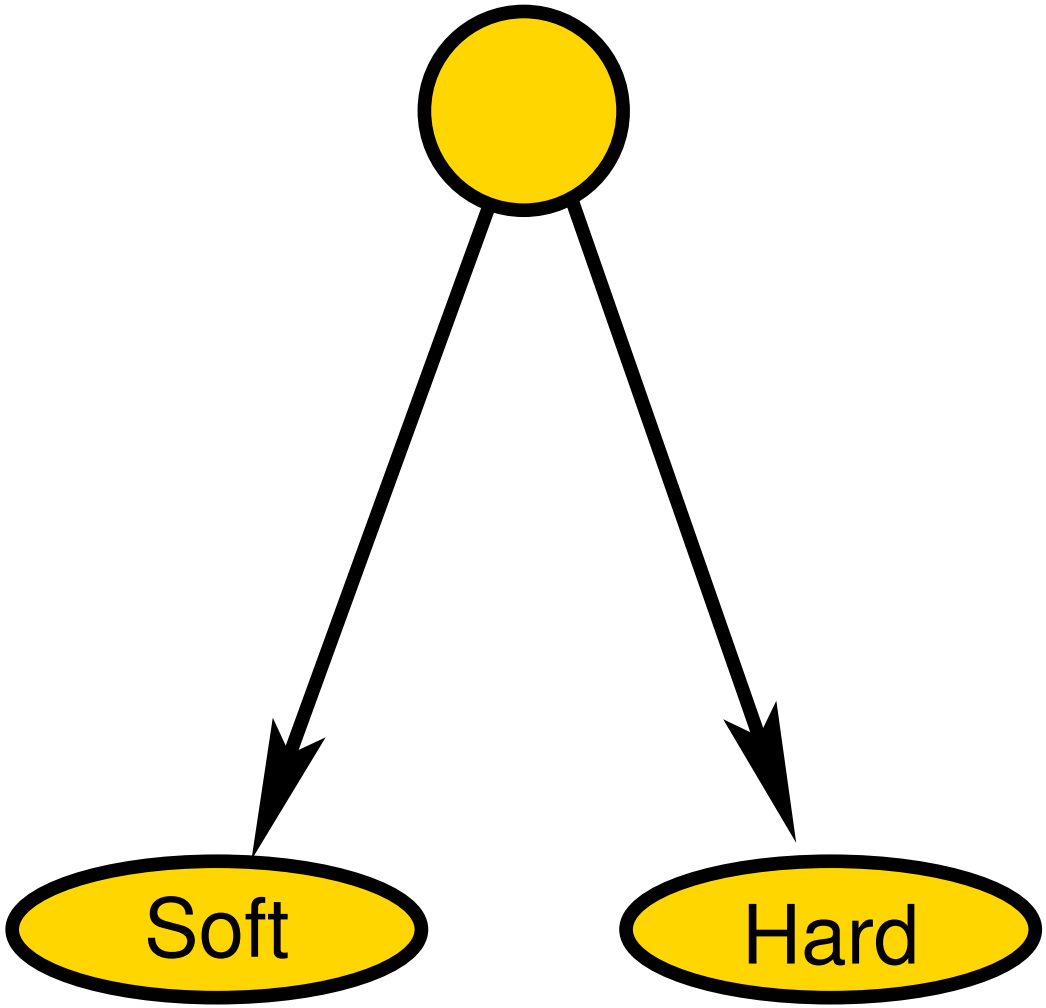
Outline

Section outline

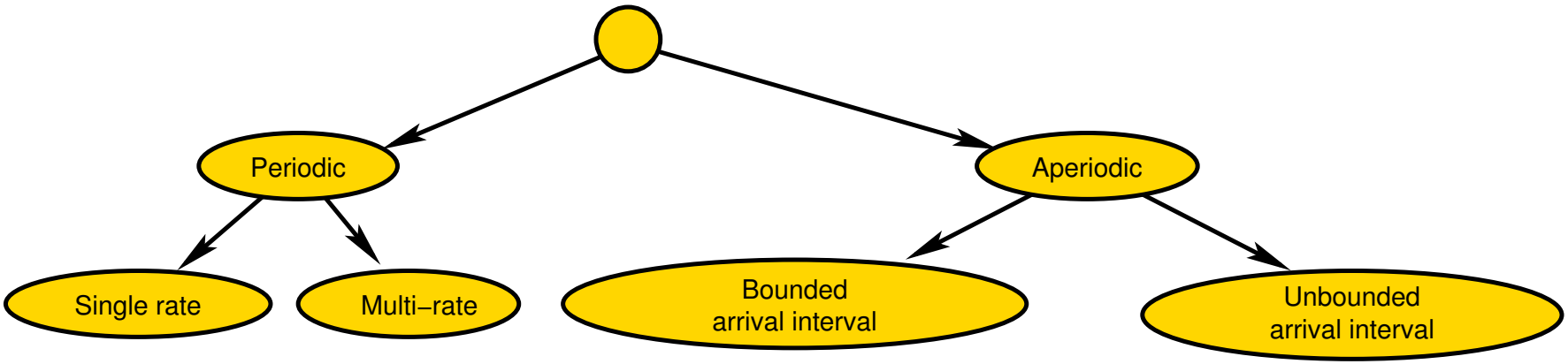
Taxonomy of real-time systems



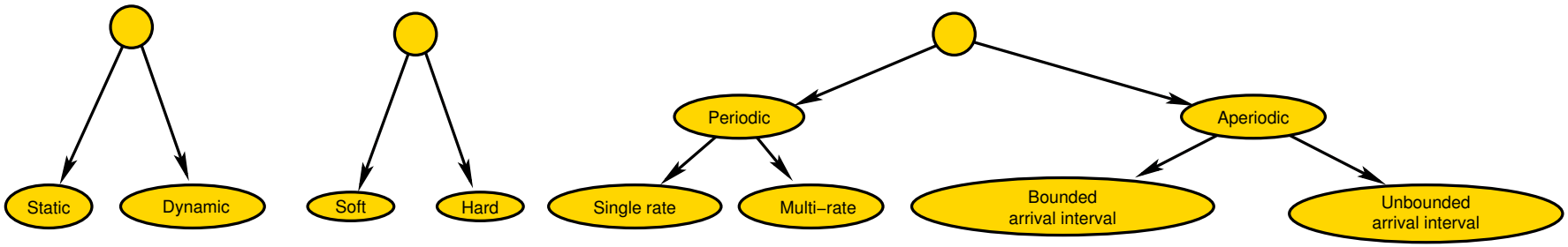
Taxonomy of real-time systems



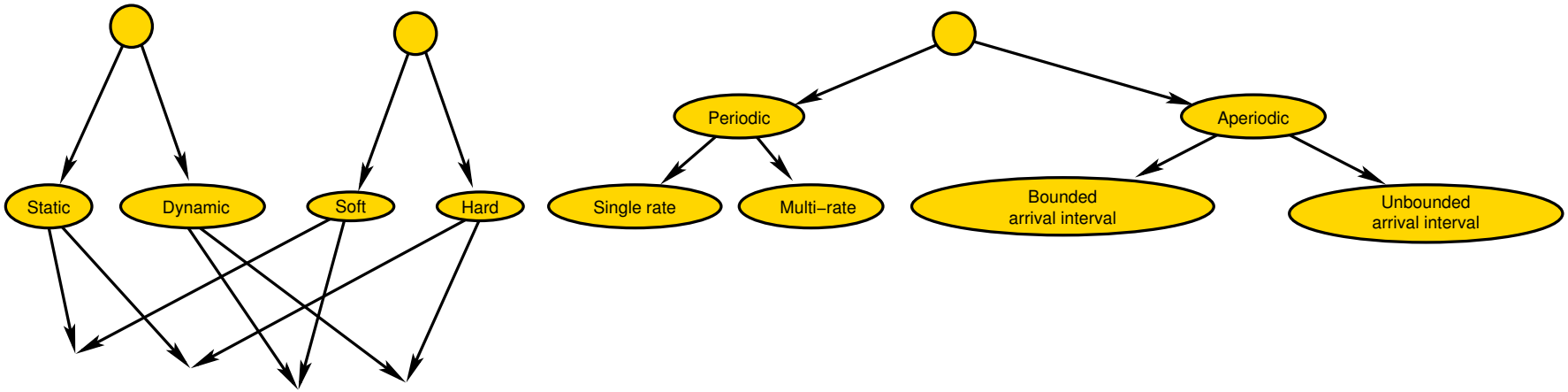
Taxonomy of real-time systems



Taxonomy of real-time systems



Taxonomy of real-time systems



Static

Task arrival times can be predicted.

Static (compile-time) analysis possible.

Allows good resource usage (low processor idle time proportions).

Sometimes designers shoehorn dynamic problems into static formulations allowing a good solution to the wrong problem.

Dynamic

Task arrival times unpredictable.

Static (compile-time) analysis possible only for simple cases.

Even then, the portion of required processor utilization efficiency goes to 0.693.

In many real systems, this is very difficult to apply in reality (more on this later).

Use the right tools but don't over-simplify, e.g.,

We assume, without loss of generality, that all tasks are independent.

Soft real-time

More slack in implementation.

Timing may be suboptimal without being incorrect.

Problem formulation can be much more complicated than hard real-time.

Two common (and one uncommon) methods of dealing with non-trivial soft real-time system requirements.

- Set somewhat loose hard timing constraints.
- Informal design and testing.
- Formulate as optimization problem.

Hard real-time

Difficult problem. Some timing constraints inflexible.

Simplifies problem formulation.

Periodic

Each task (or group of tasks) executes repeatedly with a particular period.

Allows some nice static analysis techniques to be used.

Matches characteristics of many real problems...

...and has little or no relationship with many others that designers try to pretend are periodic.

Periodic → single-rate

One period in the system.

Simple.

Inflexible.

This is how a *lot* of wireless sensor networks are implemented.

Periodic → multirate

Multiple periods.

Can use notion of circular time to simplify static (compile-time) schedule analysis E. L. Lawler and C. U. Martel, "Scheduling periodically occurring tasks on multiple processors," *Information Processing Ltrs.*, vol. 7, pp. 9–12, Feb. 1981.

Co-prime periods leads to analysis problems.

Periodic → other

It is possible to have tasks with deadlines less than, equal to, or greater than their periods.

Results in multi-phase, circular-time schedules with multiple concurrent task instances.

If you ever need to deal with one of these, see me (take my code). This class of scheduler is nasty to code.

Aperiodic

Also called sporadic, asynchronous, or reactive.

Implies dynamic.

Bounded arrival time interval permits resource reservation.

Unbounded arrival time interval impossible to deal with for any resource-constrained system.

Section outline

Definitions

Task.

Processor.

Graph representations.

Deadline violation.

Cost functions.

Task

Some operation that needs to be carried out.

Atomic completion: A task is all done or it isn't.

Non-atomic execution: A task may be interrupted and resumed.

Processor

Processors execute tasks.

Distributed systems.

- Contain multiple processors.
- Inter-processor communication has impact on system performance.
- Communication is challenging to analyze.

One processor type: Homogeneous system.

Multiple processor types: Heterogeneous system.

Task/processor relationship

WC exec time (s)

	IBM PowerPC 405GP 266 MHz	IDT79RC32364 100 MHz	Imsys Cjip 40 MHz
Tooth	7.7E−6	···	
Road	330E−9	...	
FIR	4.1E−6	···	
Matrix	310E−3	···	

Relationship between tasks, processors, and costs.

Examples: power consumption or worst-case execution time.

Cost functions

Mapping of real-time system design problem solution instance to cost value.

I.e., allows price, or hard deadline violation, of a particular multi-processor implementation to be determined.

Back to real-time problem taxonomy: jagged edges

Some things can dramatically complicate real-time scheduling.

Data dependencies.

Unpredictability.

Distributed systems.

Heterogeneous processors.

Preemption.

Section outline

Central areas of real-time study

Allocation, assignment and **scheduling**.

Operating systems and **scheduling**.

Distributed systems and **scheduling**.

Scheduling is at the core of real-time systems study.

Operating systems and scheduling

How does one best design operating systems to

Support sufficient detail in workload specification to allow good control, e.g., over scheduling, without increasing design error rate.

Design operating system schedulers to support real-time constraints?

Support predictable costs for task and OS service execution.

Distributed systems and scheduling

How does one best dynamically control

The assignment of tasks to processing nodes...

... and their schedules.

for systems in which computation nodes may be separated by vast distances such that

Task deadline violations are bounded (when possible)...

... and minimized when no bounds are possible.

The value of formality: optimization and costs

The design of a real-time system is fundamentally a cost optimization problem.

Minimize costs under constraints while meeting functionality requirements.

Functionality requirements are actually just constraints.

Why view problem in this manner?

Without having a concrete definition of the problem.

- How is one to know if an answer is correct?
- More subtly, how is one to know if an answer is optimal?

Optimization

Thinking of a design problem in terms of optimization gives design team members objective criterion by which to evaluate the impact of a design change on quality.

Know whether your design changes are taking you in a good direction

Outline

Primary publication

C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

Rate mononotic scheduling (RMS)

- Single processor.
- Independent tasks.
- Differing arrival periods.
- Schedule in order of increasing periods.
- No fixed-priority schedule will do better than RMS.
- Guaranteed valid for loading $\leq \ln 2 = 0.69$.
- For loading $> \ln 2$ and < 1, correctness unknown.
- Usually works up to a loading of 0.88.

Rate monotonic scheduling

1973, Liu and Layland derived optimal scheduling algorithm(s) for this problem.

C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

Schedule the job with the smallest period (period = deadline) first.

Analyzed worst-case behavior on any task set of size n.

Found utilization bound: $U(n) = n \cdot (2^{1/n} - 1)$.

0.828 at $n = 2$.

As $n \to \infty$, $U(n) \to \log 2 = 0.693$.

Result: For any problem instance, if a valid schedule is possible, the processor need never spend more than 31% of its time idle.

Optimality and utilization for limited case

Simply periodic: All task periods are integer multiples of all lesser task periods.

In this case, RMS/DMS optimal with utilization 1.

However, this case rare in practice.

Remains feasible, with decreased utilization bound, for in-phase tasks with arbitrary periods.

Rate monotonic scheduling

Constrained problem definition.

Over-allocation often results.

However, in practice utilization of 85%–90% common.

Lose guarantee.

If phases known, can prove by generating instance.

Critical instants

Definition

A job's critical instant a time at which all possible concurrent higher-priority jobs are also simultaneously released.

Useful because it implies latest finish time

Definitions

Period: T.

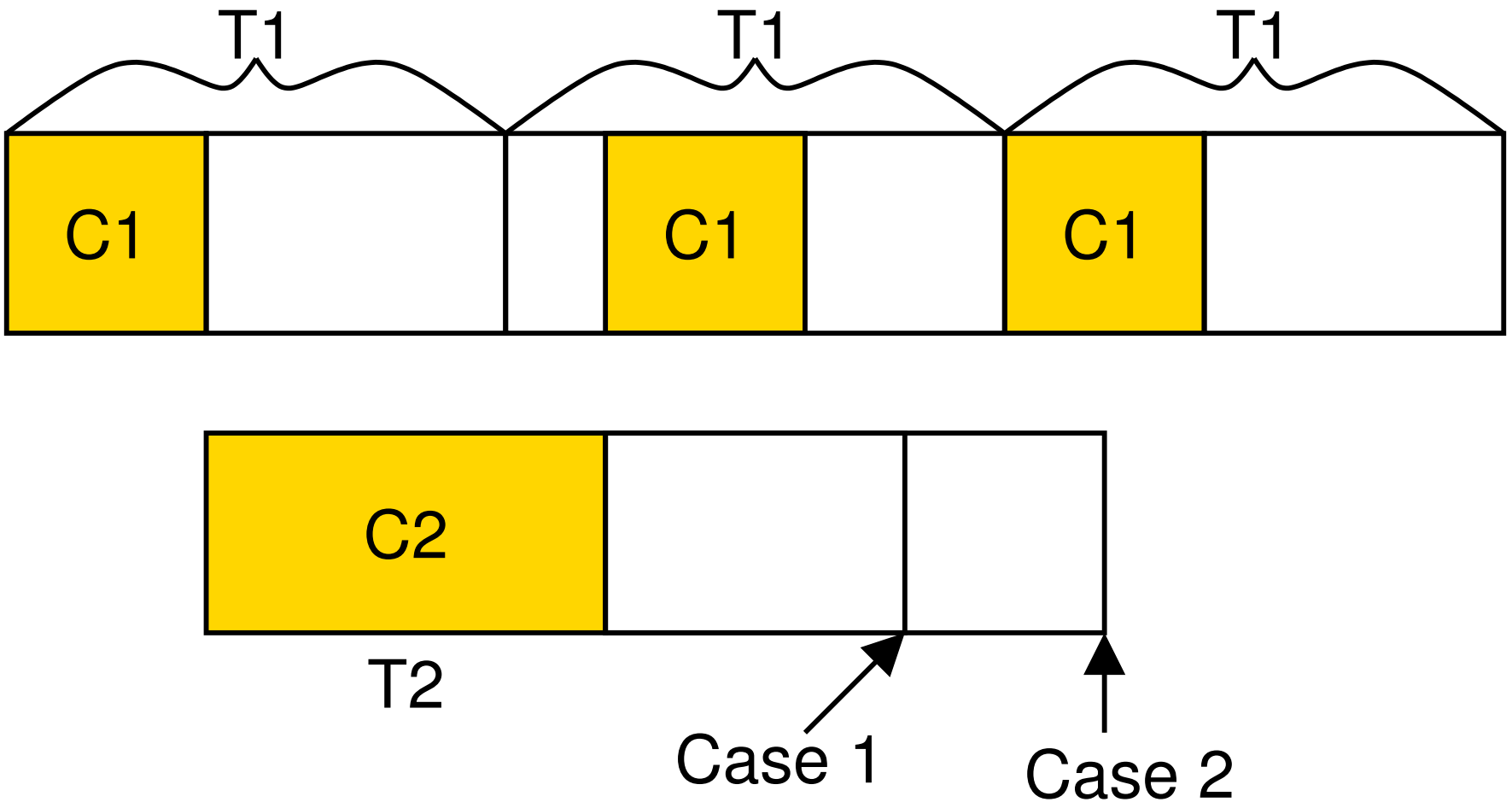
Execution time: C.

Process: i.

Utilization: $U = \sum_{i=1}^{m} \frac{C_i}{T_i}$.

Assume Task 1 is higher priority than Task 2, and thus $T_1 < T_2$.

Critical instants



Case 1 I

All instances of higher-priority tasks released before end of lower-priority task period complete before end of lower-priority task period.

$$C_1 \leq T_2 - T_1 \left\lfloor \frac{T_2}{T_1} \right\rfloor.$$

I.e., the execution time of Task 1 is less than or equal to the period of Task 2 minus the total time spent within the periods of instances of Task 1 finishing within Task 2's period.

Now, let's determine the maximum execution time of Task 2 as a function of all other variables.

Number of T_1 released $= \left\lceil \frac{T_2}{T_1} \right\rceil$ so $C_{2,max} = T_2 - C_1 \left\lceil \frac{T_2}{T_1} \right\rceil$.

Case 1 II

I.e., the maximum execution time of Task 2 is the period of Task 2 minus the total execution time of instances of Task 1 released within Task 2's period.

Case 1 III

In this case,

$$\begin{aligned} U &= U_1 + U_2 \\ &= \frac{C_1}{T_1} + \frac{C_{2,max}}{T_2} \\ &= \frac{C_1}{T_1} + \frac{T_2 - C_1 \left\lceil \frac{T_2}{T_1} \right\rceil}{T_2} \\ &= \frac{C_1}{T_1} + 1 - \frac{C_1 \left\lceil \frac{T_2}{T_1} \right\rceil}{T_2} \\ &= 1 + C_1 \left(\frac{1}{T_1} - \frac{1}{T_2} \left\lceil \frac{T_2}{T_1} \right\rceil \right) \end{aligned}$$

Is $\frac{1}{T_1} - \frac{1}{T_2} \left\lceil \frac{T_2}{T_1} \right\rceil \leq 0$?

Case 1 IV

If $T_2 = T_1 + \epsilon$, this is

$$\begin{aligned} &\frac{1}{T_1} - \frac{1}{T_1 + \epsilon} \left\lceil \frac{T_1 + \epsilon}{T_1} \right\rceil \\ =&\frac{1}{T_1} - \frac{2}{T_1 + \epsilon} \end{aligned}$$

which is less than or equal to zero.

Thus, U is monotonically nonincreasing in C_1.

Case 2 I

Instances of higher-priority tasks released before end of lower-priority task period complete after end of lower-priority task period.

$$C_1 \geq T_2 - T_1 \left\lfloor \frac{T_2}{T_1} \right\rfloor.$$

$$C_{2,max} = T_1 \left\lfloor \frac{T_2}{T_1} \right\rfloor - C_1 \left\lfloor \frac{T_2}{T_1} \right\rfloor.$$

$$U_1 = C_1/T_1.$$

$$U_2 = C_2/T_2 = \frac{T_1}{T_2} \left\lfloor \frac{T_2}{T_1} \right\rfloor - \frac{C_1}{T_2} \left\lfloor \frac{T_2}{T_1} \right\rfloor.$$

$$U = U_1 + U_2 = \frac{T_1}{T_2} \left\lfloor \frac{T_2}{T_1} \right\rfloor + C_1 \left(\frac{1}{T_1} - \frac{1}{T_2} \left\lfloor \frac{T_2}{T_1} \right\rfloor \right).$$

Minimal U I

$$C_1 = T_2 - T_1 \left\lfloor \frac{T_2}{T_1} \right\rfloor.$$

$$U = 1 - \frac{T_1}{T_2}\left(\left\lceil \frac{T_2}{T_1} \right\rceil - \frac{T_2}{T_1}\right)\left(\frac{T_2}{T_1} - \left\lfloor \frac{T_2}{T_1} \right\rfloor\right).$$

Let $I = \left\lfloor \frac{T_2}{T_1} \right\rfloor$ and

$$f = \frac{T_2}{T_1}.$$

Then, $U = 1 - \frac{f(1-f)}{I+f}$.

To maximize U, minimize I, which can be no smaller than 1.

$$U = 1 - \frac{f(1-f)}{1+f}.$$

Minimal U II

Differentiate to find mimima, at $f = \sqrt{2} - 1$.

Thus, $U_{min} = 2\left(\sqrt{2} - 1\right) \approx 0.83$.

Is this the minimal U? Are we done?

Proof sketch for RMS utilization bound

Consider case in which no period exceeds twice the shortest period.

Find a pathological case: in phase

- Utilization of 1 for some duration.
- Any decrease in period/deadline of longest-period task will cause deadline violations.
- Any increase in execution time will cause deadline violations.

Proof sketch for RMS utilization bound

See if there is a way to increase utilization while meeting all deadlines.

Increase execution time of high-priority task.

$e_i' = p_{i+1} - p_i + \epsilon = e_i + \epsilon.$

Must compensate by decreasing another execution time.

This always results in decreased utilization.

- $e_k' = e_k - \epsilon.$
- $U' - U = \frac{e_i'}{p_i} + \frac{e_k'}{p_k} - \frac{e_i}{p_i} - \frac{e_k}{p_k} = \frac{\epsilon}{p_i} - \frac{\epsilon}{p_k}.$
- Note that $p_i < p_k \rightarrow U' > U.$

Proof sketch for RMS utilization bound

Same true if execution time of high-priority task reduced.

$e_i'' = p_{i+1} - p_i - \epsilon$.

In this case, must increase other e or leave idle for $2 \cdot \epsilon$.

$e_k'' = e_k + 2\epsilon$.

$U'' - U = \frac{2\epsilon}{p_k} - \frac{\epsilon}{p_i}$.

Again, $p_k < 2 \rightarrow U'' > U$.

Sum over execution time/period ratios.

Proof sketch for RMS utilization bound

Get utilization as a function of adjacent task ratios.

Substitute execution times into $\sum_{k=1}^{n} \frac{e_k}{p_k}$.

Find minimum.

Extend to cases in which $p_n > 2 \cdot p_k$.

Notes on RMS

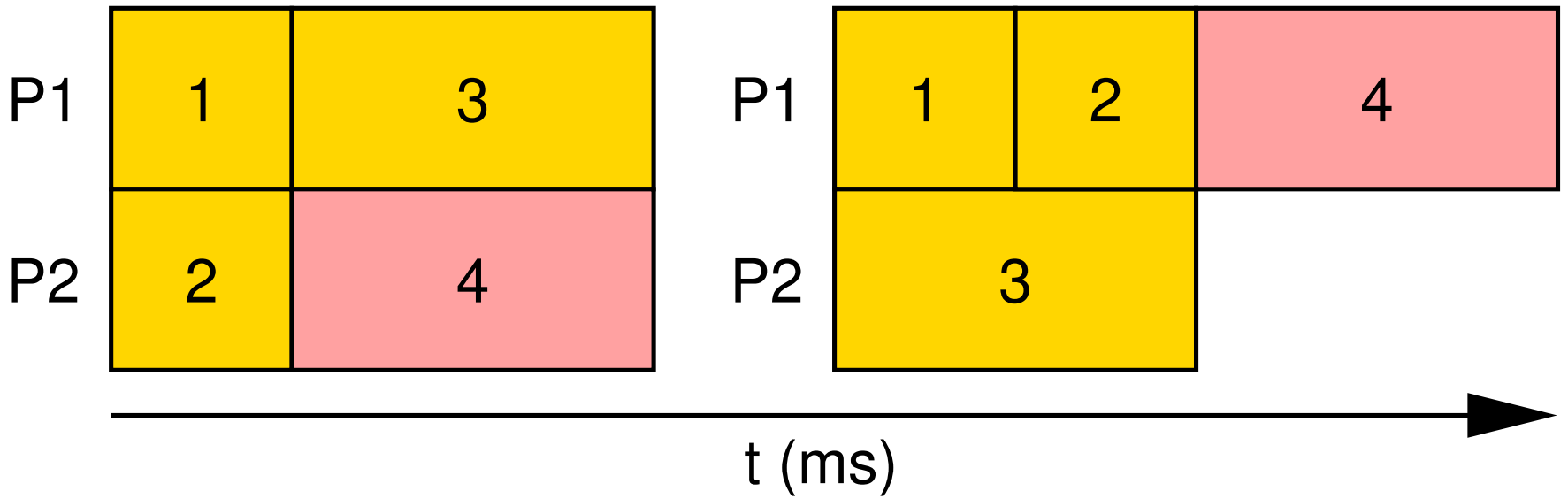
DMS better than or equal RMS when deadline $\neq$ period.

Why not use slack-based?

Why no generate a static schedule and use a table?

What happens if resources are under-allocated and a deadline is missed?

Multiprocessor breaks critical instant



Why cover RMS?

It introduces several concepts that are useful when reasoning about scheduling.

Example of idea that can lead to general conclusions that simplify reliable embedded system design.

It's still useful in some systems, e.g., when overhead of dynamic priority scheduling is substantial.

Outline

Many options

- eCos.
- LynxOS.
- MontaVista Linux.
- QNX.
- RTAI.
- RTLinux.
- Symbian OS.
- VxWorks.
- FreeRTOS.
- Etc.

Threads

Threads vs. processes: Shared vs. unshared resources.

OS impact: Windows vs. Linux.

Hardware impact: MMU.

Threads vs. processes

Threads: Low context switch overhead.

Threads: Sometimes the only real option, depending on hardware.

Processes: Safer, when hardware provides support.

Processes: Can have better performance when IPC limited.

Software implementation of schedulers

- TinyOS.
- Light-weight threading executive.
- μC/OS-II.
- Linux.
- Static list scheduler.

TinyOS

Most behavior event-driven.

High rate $\rightarrow$ livelock.

Research schedulers exist.

BD threads

- Brian Dean: Microcontroller hacker.
- Simple priority-based thread scheduling executive.
- Tiny footprint (fine for AVR).
- Low overhead.
- No MMU requirements.

μC/OS-II

Similar to BD threads.

More flexible.

Bigger footprint.

Old Linux scheduler

Single run queue.

$\mathcal{O}(n)$ scheduling operation.

Allows dynamic goodness function.

$\mathcal{O}(1)$ scheduler in Linux 2.6+

- Written by Ingo Molnar.
- Splits run queue into two queues prioritized by goodness.
- Requires static goodness function.
- No reliance on running process.
- Compatible with preemptable kernel.

$\mathcal{O}(\log n)$ scheduler in Linux 2.6.23+

- Written by Ingo Molnar.
- Used red-black tree to maintain accumulated task times.
- Always schedules task with least accumulated time.
- Weights accumulated time based on priority.

Real-time Linux

Run Linux as process under real-time executive.

Complicated programming model.

RTAI (Real-Time Application Interface) attempts to simplify.

Colleagues still have problems at $> 18\,\text{kHz}$ control period.

Real-time operating systems

- Embedded vs. real-time.
- Dynamic memory allocation.
- Schedulers: General-purpose vs. real-time.
- Timers and clocks: Relationship with HW.

Real-time operating systems

Interaction between HW and SW.

- Rapid response to interrupts.
- HW interface abstraction.

Interaction between different tasks.

- Communication.
- Synchronization.

Multitasking

- Ideally fully preemptive.
- Priority-based scheduling.
- Fast context switching.
- Support for real-time clock.

General-purpose OS stress

Good average-case behavior.

Providing many services.

Support for a large number of hardware devices.

RTOSs stress

- Predictable service execution times.
- Predictable scheduling.
- Good worst-case behavior.
- Low memory usage.
- Speed.
- Simplicity.

Predictability

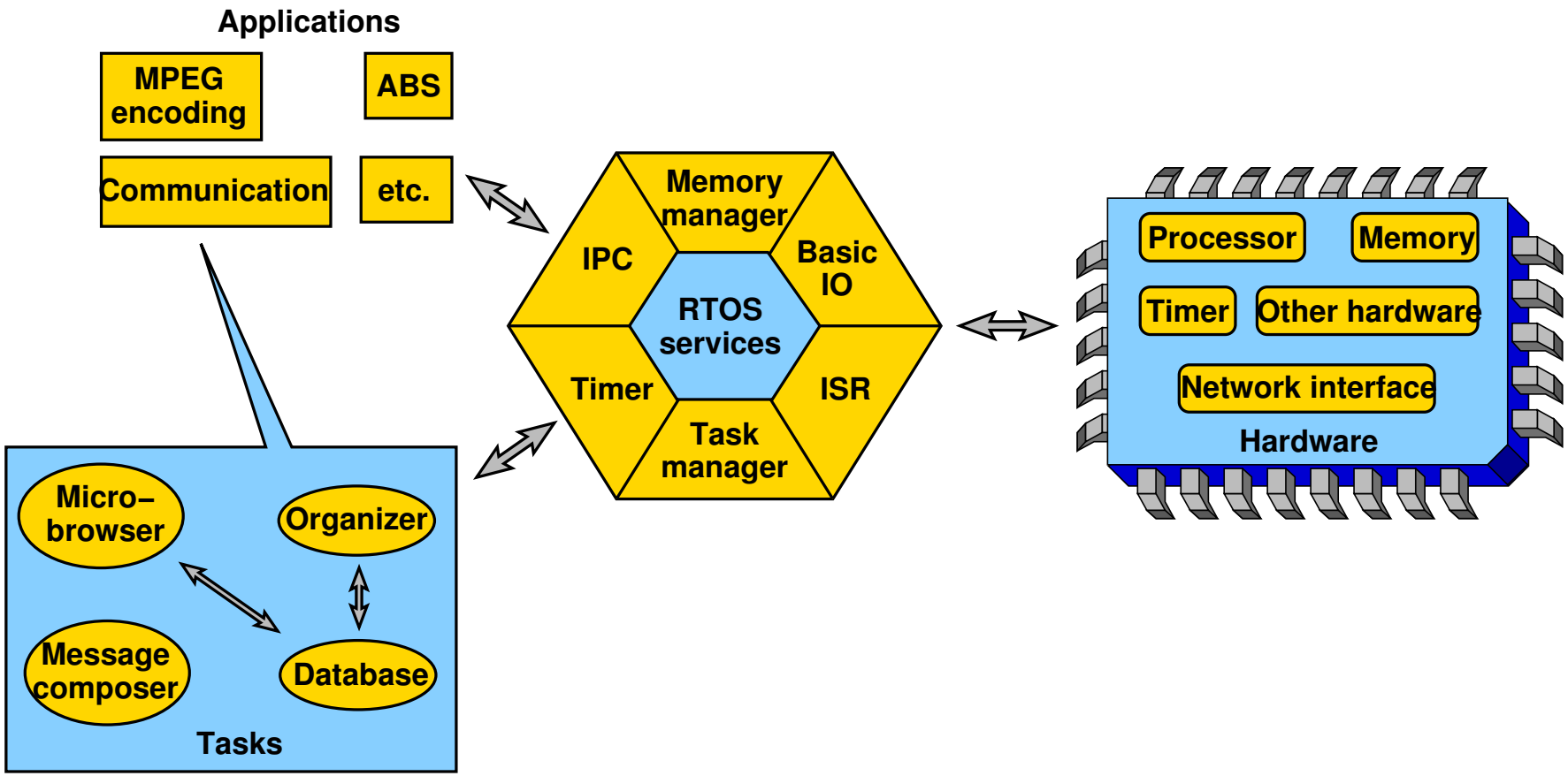
General-purpose computer architecture focuses on average-case.

- Caches.
- Prefetching.
- Speculative execution.

Real-time embedded systems need predictability.

- Disabling or locking caches is common.
- Careful evaluation of worst-case is essential.
- Specialized or static memory management common.

RTOS overview

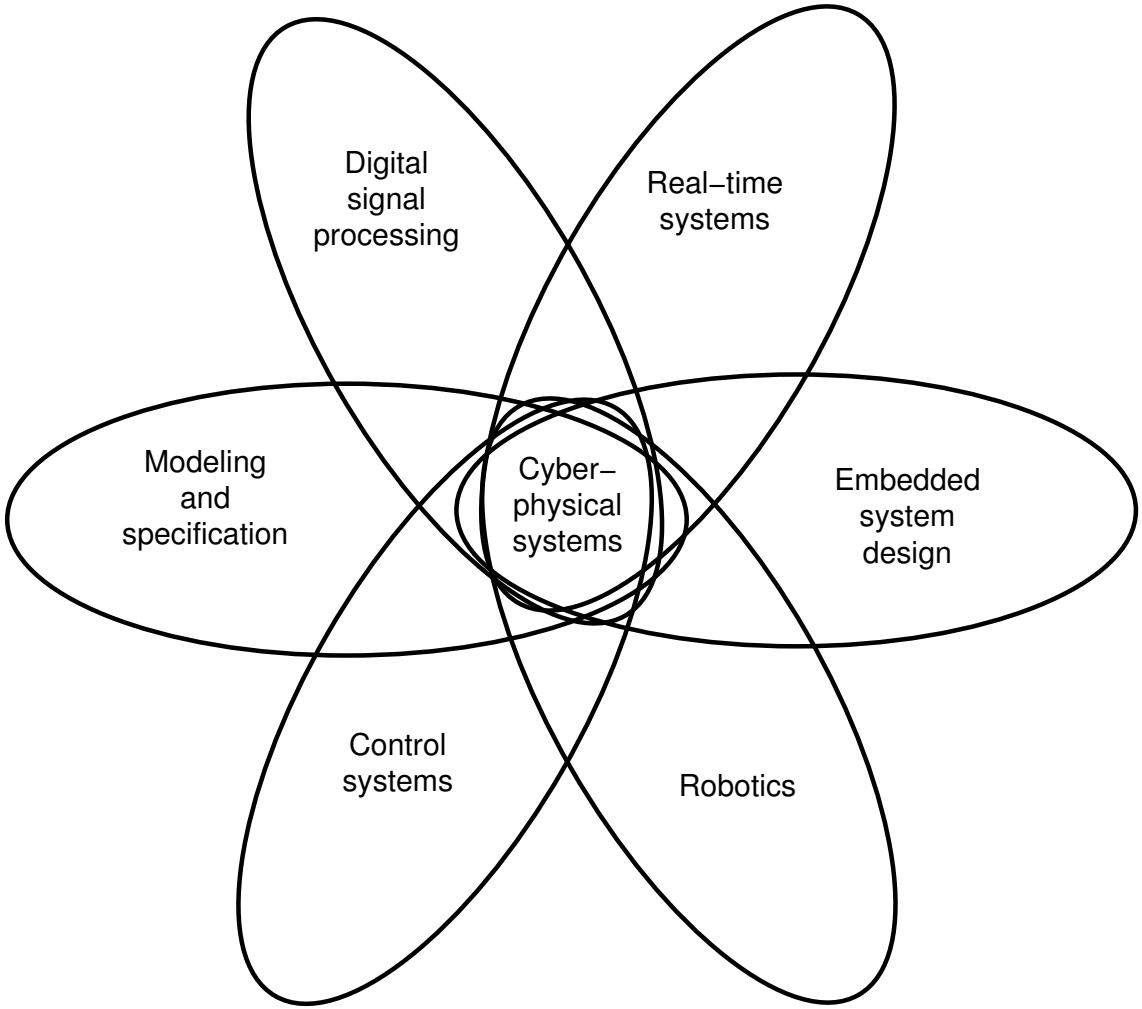


Outline

Cyber-physical systems

- Computer systems involving interaction with the physical world.
- Sensing $\rightarrow$
- Signal processing $\rightarrow$
- Analysis and control $\rightarrow$
- Actuation.
- All tied together via models and hardware/software implementations.

Cyber-physical systems disciplines



Relationship with IoT

Substantial overlap.

IoT systems arguably needn't involve control theory and actuation, although they often do.

CPS need not involve slow, unreliable networks, although they often do.

Outline

Deadlines and announcements I

27 Sep.: Final project proposals.

27 Sep.: L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2010, pp. 105–114.

29 Sep.: J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson, "Analysis of wireless sensor networks for habitat monitoring," in *Wireless Sensor Networks*, C. S. Raghavendra, K. M. Sivalingam, and T. Znati, Eds. Springer US, 2004, ch. 18, pp. 399–423.

4 Oct.: U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low power wide area networks: An overview," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 2, 2017.

Deadlines and announcements II

6 Oct.: "Research challenges for energy-efficient computing in automated vehicles," *IEEE Computer*, 2022, to appear.

11 Oct.: E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn, "IoT goes nuclear: Creating a ZigBee chain reaction," in *Proc. Symp. on Security and Privacy*, May 2017.