Embedded Real-Time Operating Systems

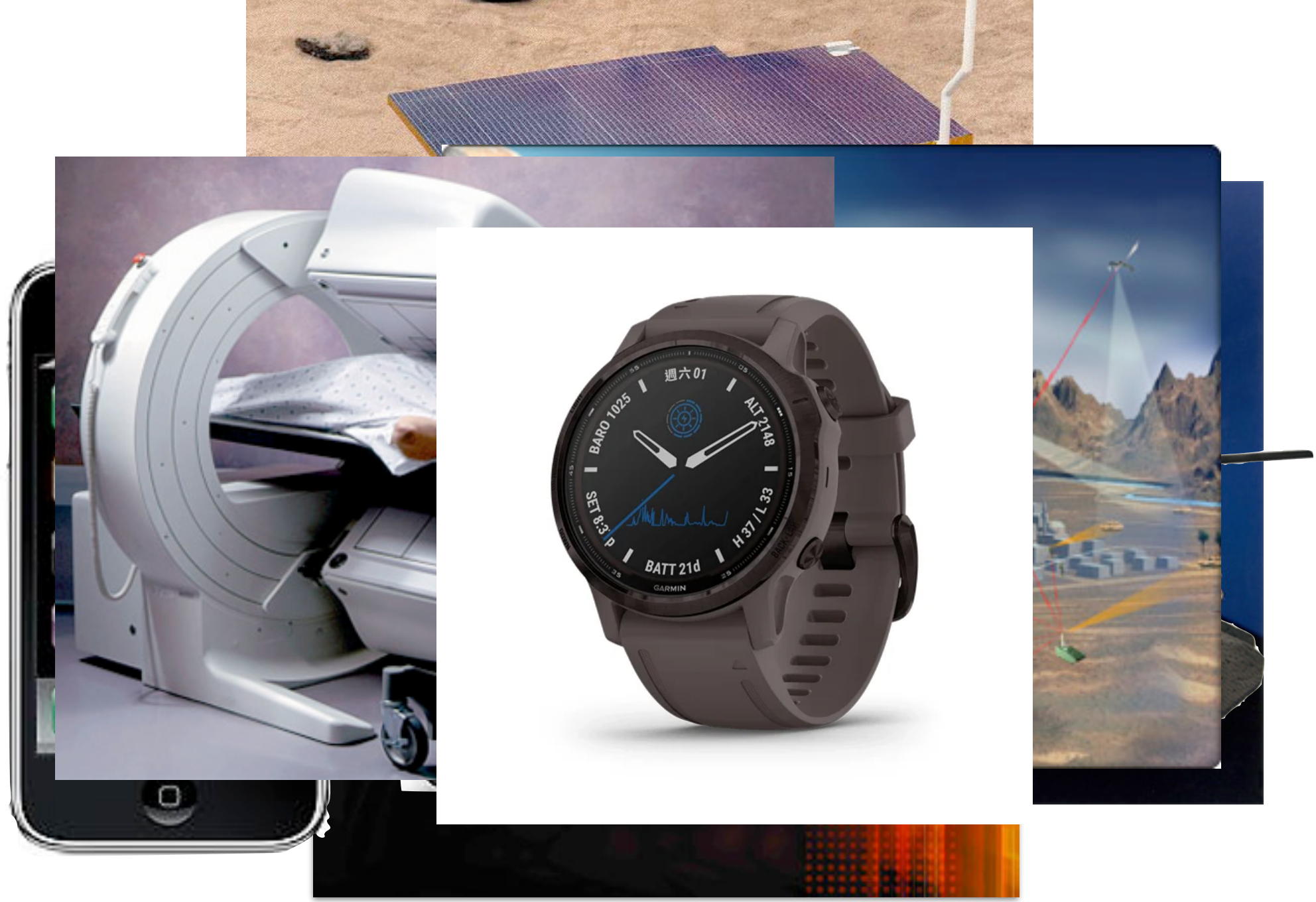
國立臺灣大學

NEWS Lab
嵌入式系統暨無線網路實驗室

Outline

- Introduction to Embedded and Real-Time Systems
- Real-Time Operating Systems
- Real-Time Scheduling
- Schedulability Analysis and unbounded priority inversion

Introduction to Embedded (and) Real-Time Systems



Embedded Systems

What are Embedded Systems?

- Typical textbook definition:

  A computer that is a component in a larger system, and is not visible as a computer to a user of that system.

- But - An embedded system may:
  - Look and function like a traditional computer,
  - Have a typical computer User Interface, or
  - Not contain a traditional CPU at all!

What are Embedded Systems?

- Our (better) definition:

  A programmable component or subsystem providing some intelligence functions to the system of which it is a part.

- This can include:
  - Any device, or collection of devices, that contain one or more dedicated computers, microprocessors, or micro-controllers.
  - Microprocessor chips and Programmable logic elements (FPGA, ASIC etc.)
  - Device(s) may be local - Printer, automobile, etc.
  - Devices may be distributed - Aircraft, ship, internet appliance.
- Key point:
  - Embedded computing devices have rigidly defined operational bounds.
  - Not general purpose computers (PC, Unix workstation).

Characteristics of Embedded Systems

Properties

- No architectural link to standard platforms.
- Embedded systems may or may not have Operating System (OS) services available.
- Tolerance for bugs is much lower in embedded systems than in desktop computers.
- Embedded systems are cost sensitive.

Constraints

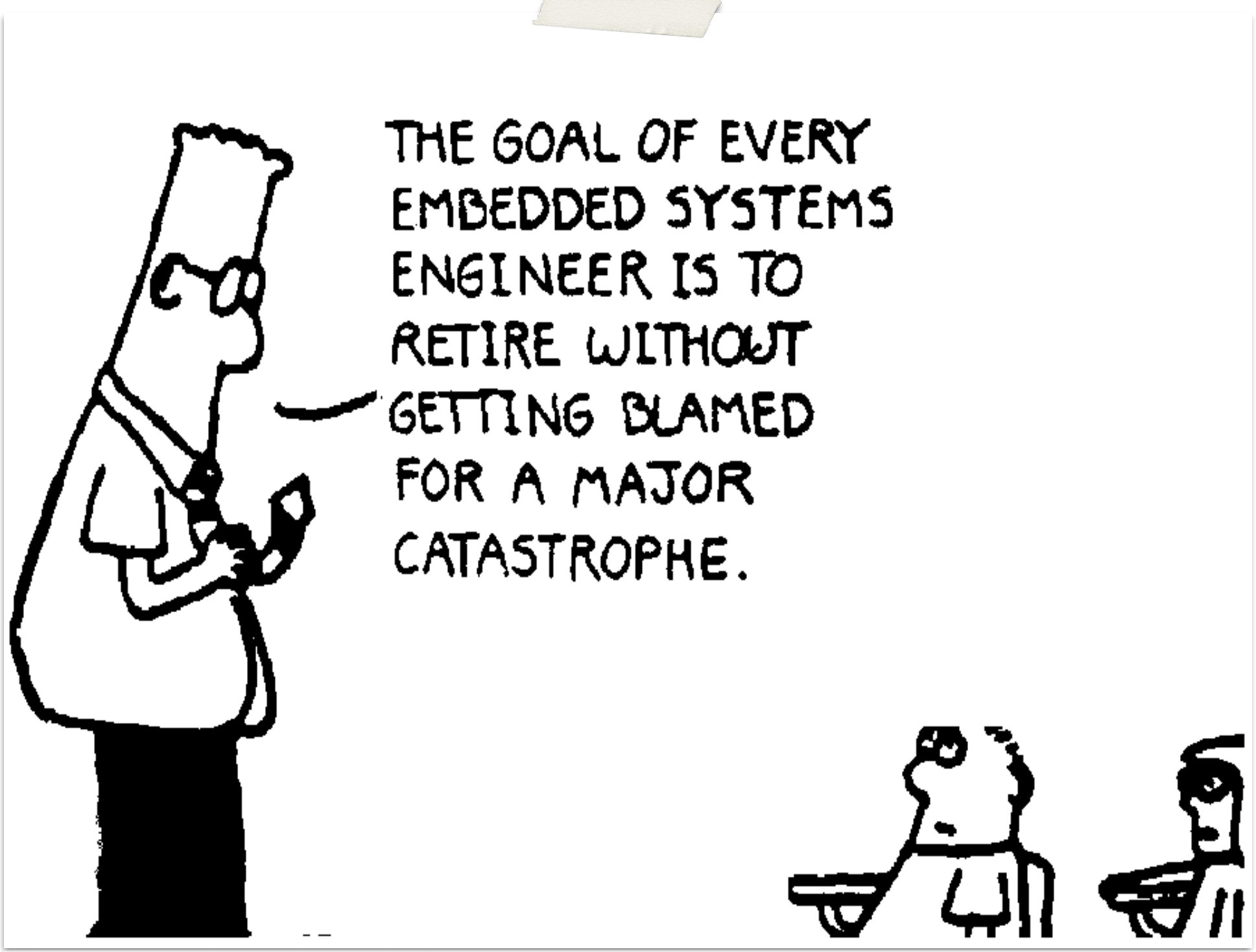
- Power/Energy constraints.
- Reliability
- Robust
- Moderate to severe real-time constraints.

Emergency airworthiness directive by FAA

- ISSUE DATE: **August 29, 2005**
- AD: 2005-18-51; FAA-2005-22252; Directorate Identifier 2005-NM-182-AD
- Report: a recent report of a significant **nose-up pitch event** on a Boeing Model **777-200** series airplane while climbing through 36,000 feet altitude. The flight crew disconnected the autopilot and stabilized the airplane, during which time the airplane climbed above 41,000 feet, decelerated to a minimum speed of 158 knots, and activated the stick shaker. **Operational Program Software (OPS) using data from faulted sensors**, if not corrected, could result in anomalies of the fly-by-wire primary flight control, autopilot, auto-throttle, pilot display, and auto-brake systems, which could result in high pilot workload, deviation from the intended flight path, and **possible loss of control of the airplane**.

Emergency airworthiness directive by FAA

- ISSUE DATE: **February 5, 2009**
- FAA-2007-0254; Directorate Identifier 2007-NM-209-AD; Amendment 39-15795; AD 2009-02-05
- Report: We are adopting a new airworthiness directive (AD) for certain **Boeing Model 777 airplanes**. This AD requires installing software upgrades to the airplane information management system (AIMS) located in the flight compartment. This AD results from an investigation that revealed that detrimental effects could occur on certain AIMS software during flight. We are issuing this AD to prevent **an unannunciated loss of cabin pressure**. If an undetected loss of pressure event were to cause **an unsafe pressure in the cabin**, the flight crew could become **incapacitated**.


THE GOAL OF EVERY EMBEDDED SYSTEMS ENGINEER IS TO RETIRE WITHOUT GETTING BLAMED FOR A MAJOR CATASTROPHE.

Misconceptions for Embedded Systems

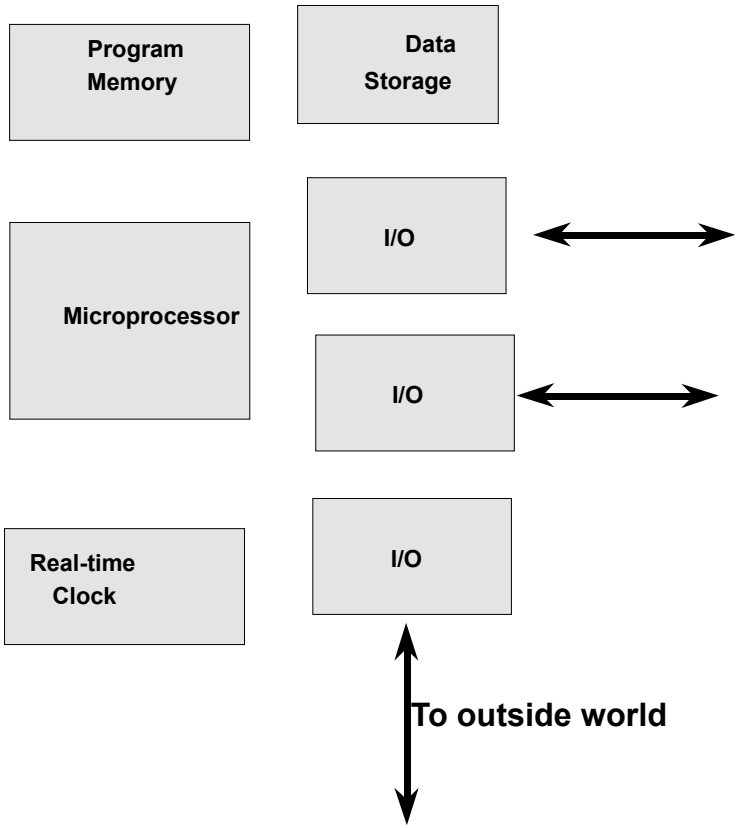
- Embedded system design is just an engineering practice, not a science.
- Endless device driver implementation.
- Porting operating systems from ARM platform, to Amtel platform, to Hitachi platform, etc.
- Embedded systems are tiny systems.
- Embedded systems are consumer electronics.

Let's Define Some Terms

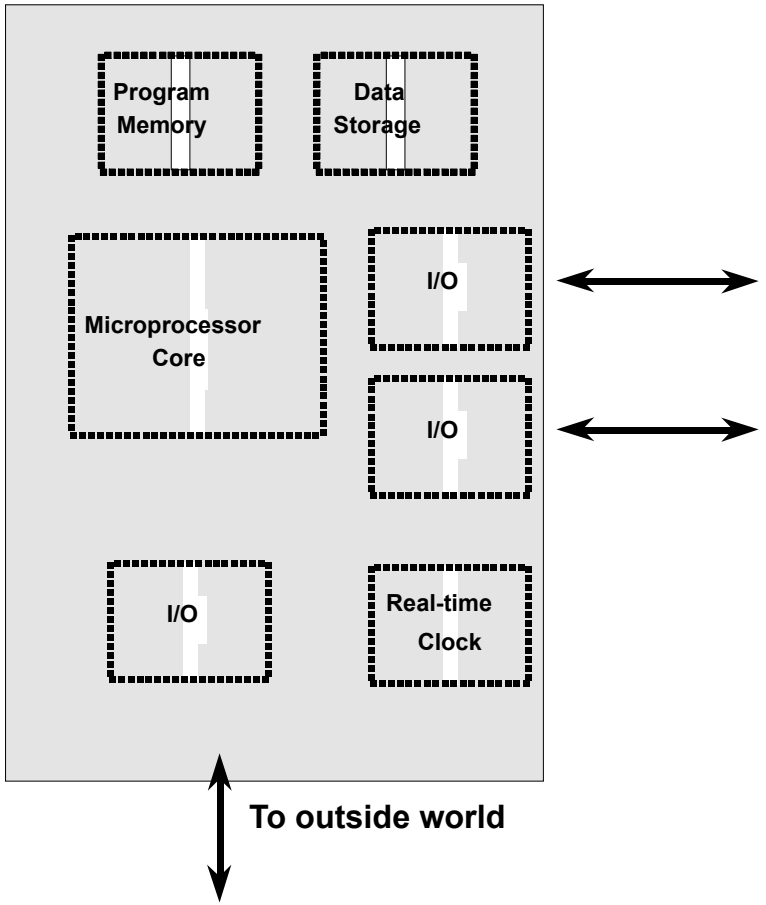
- Microprocessor
  - An integrated circuit which forms the central processing unit for a computer or embedded controller, but requires additional support circuitry to function
  - MC68000, 80486, Pentium, K6, MicroChip PIC, etc.
- Microcontroller
  - A microprocessor plus additional peripheral support devices integrated into a single package
  - Peripheral support devices may include:
    - Serial ports (COM), Parallel (Ports), Ethernet ports, A/D & D/A
    - Interval timers, watchdog timers, event counter/timers, real time clock (RTC)
    - Other local processors (DSP, numeric coprocessor, peripheral controller)
  - BrainStem on PPRK is a microcontroller

Microprocessor and Microcontroller

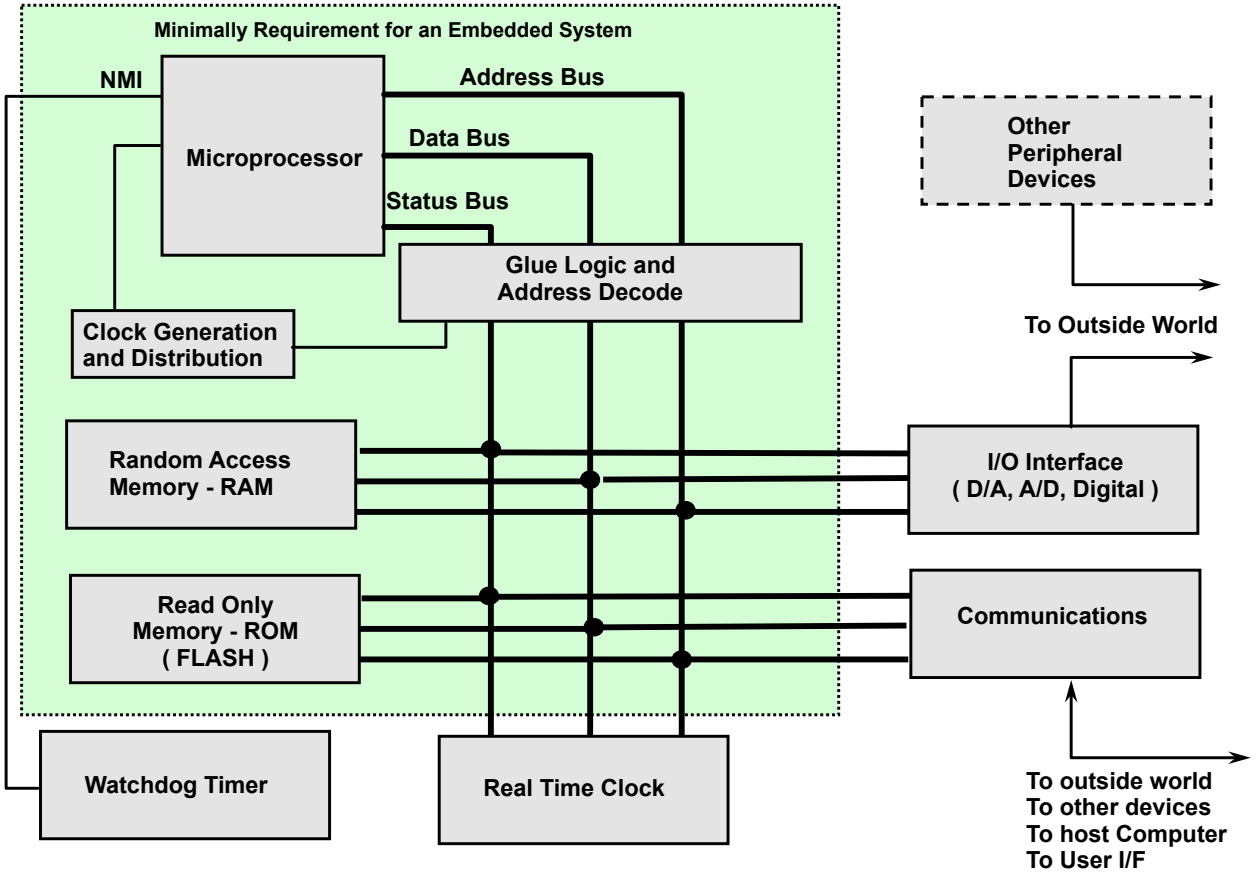
A Microprocessor-Based Embedded System



A Microcontroller-Based Embedded System

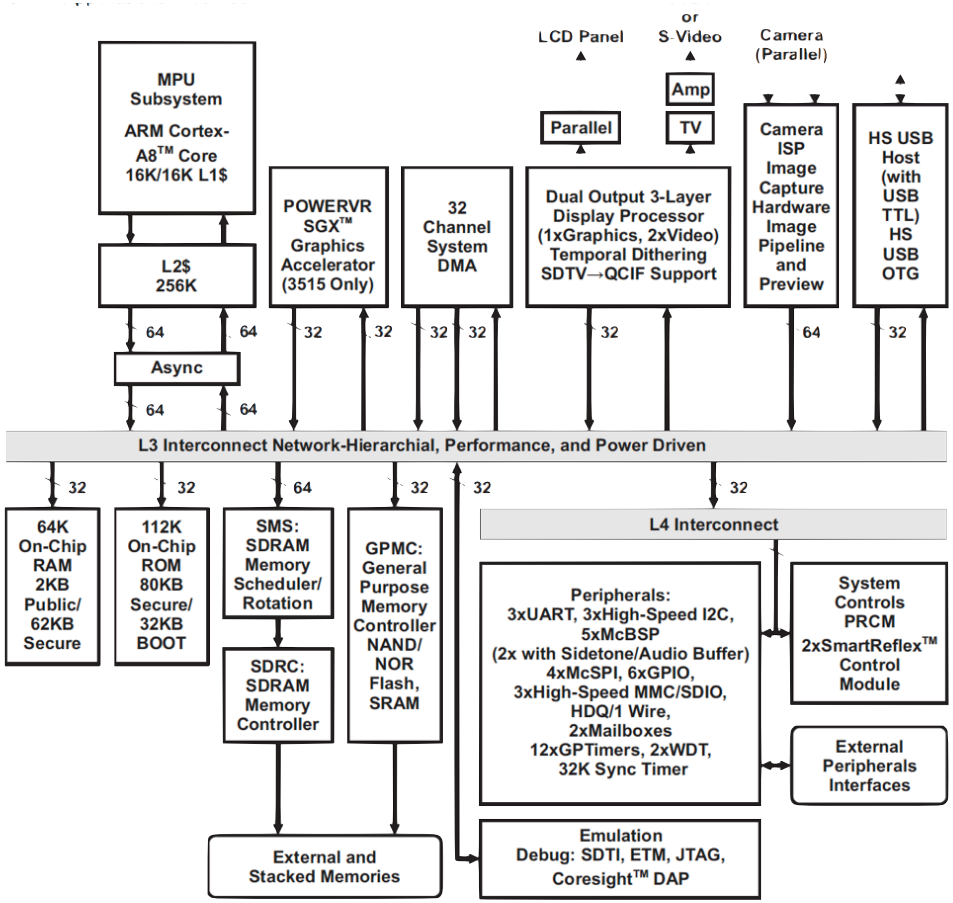


A “Typical” Embedded System



Recent developments

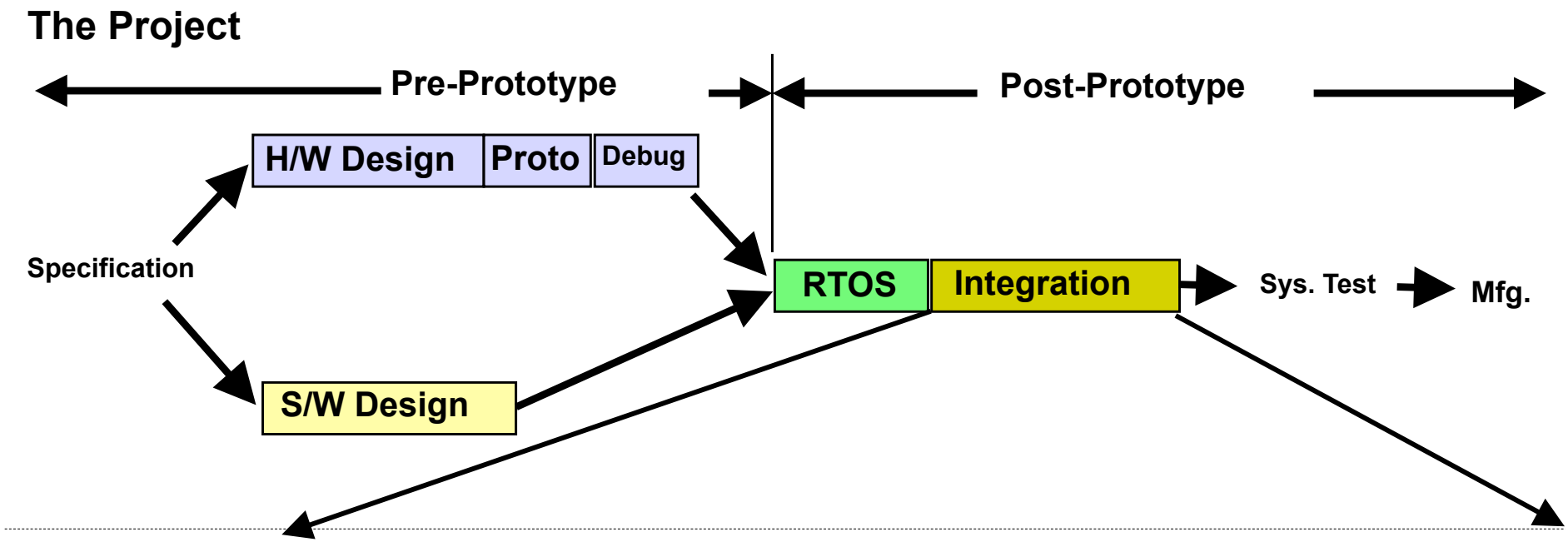
- Moore's Law: the complexity of integrated circuits will double every 18 months
- Process technology able to put more and more functionality on the same chip as the cpu
- Buzz Word: System on a Chip (SOC), or System on Silicon

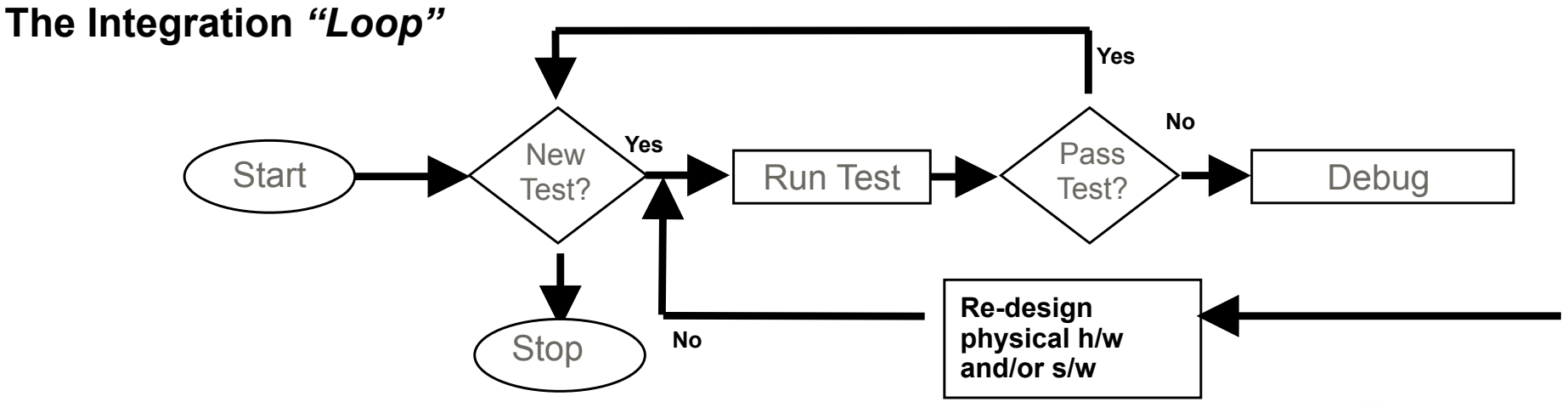


Let's Define Some Terms - 2

- Target system
  - The embedded system under development
- Host computer
  - The standard platform being used to develop the software and link to the target system for debugging
- Cross-development
  - Using host-based tools to create a code image that will execute on a different instruction set architecture
  - Example:
    - Write a C program on your PC
    - Compile it to run on a PowerPC 603 using a Cross-compiler
    - Create a runtime image for execution in the target system

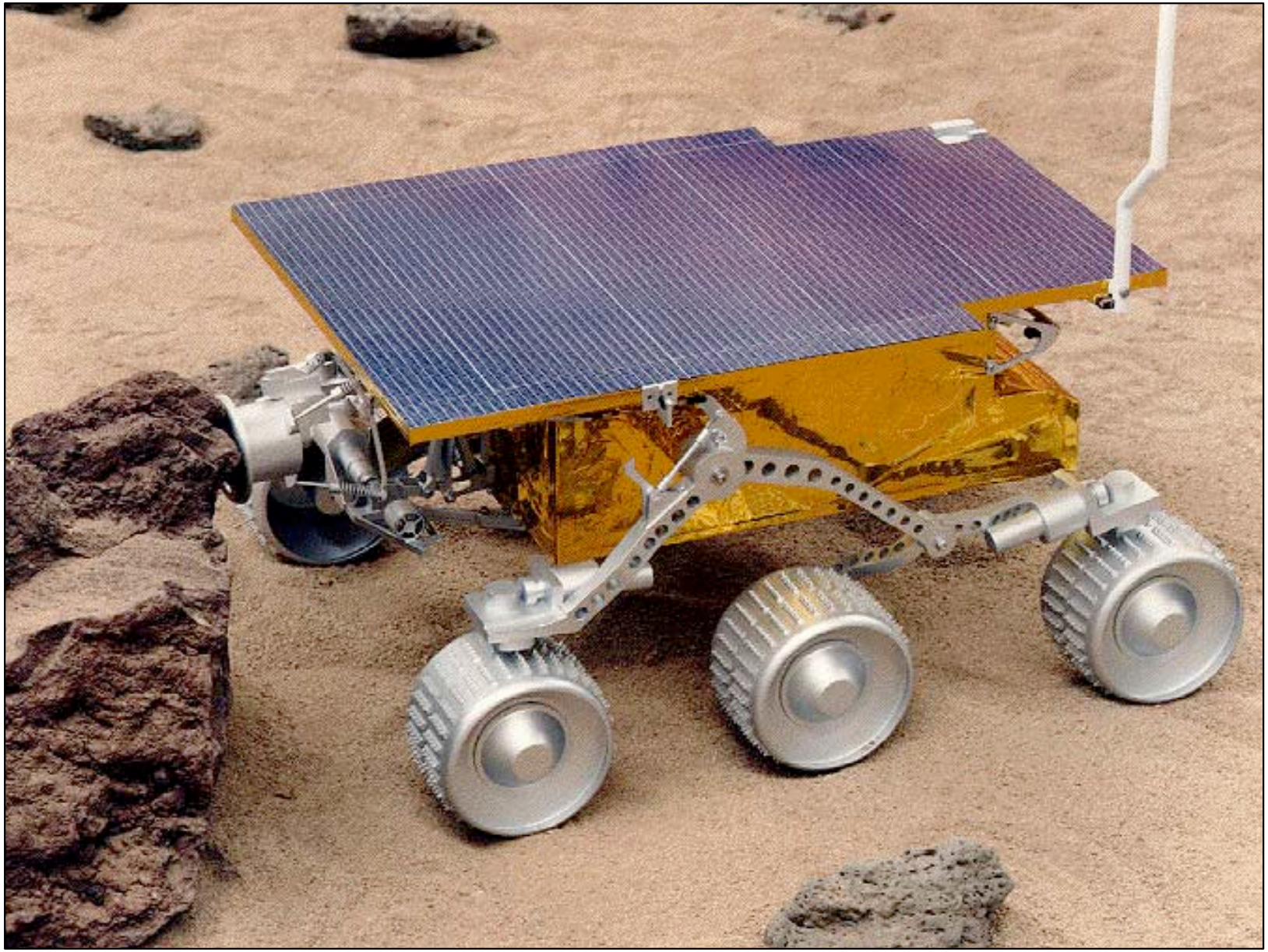
The embedded life-cycle





Let's Define Some Terms - 3

- Time sensitive
  - If a task or operation does not complete in the specified amount of time, the embedded device will perform below design requirements
  - Example: A laser printer prints 8 pagers per minute instead of 10 ppm (HP trumps Lexmark once again!)
  - Device continues to function.
- Time critical
  - If a task or operation does not complete in the specified amount of time, the embedded device will fail.
  - Example: Flight control system on a fly-by-wire aircraft.
  - Device will not operate.



Real-Time Systems

Computation Modes

Batch Mode

Online Mode

Real-Time Mode

Real-time Systems

A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time in which the result is produced.

- Timing constraints can vary between different real-time systems. Therefore, systems can fall into one of three categories:
  - Soft Real-time Systems
  - Hard Real-time Systems
  - Firm Real-time Systems

Soft Real-time Systems

- Timing requirements are defined by using an average response time. A single computation arriving late is not significant to the operation of the system, though many late arrivals might be.
- Example: Airline reservation system - If a single computation is late, the system's response time may lag. However, the only consequence would be a frustrated potential passenger.

Hard Real-Time Systems

Hard Real-time Systems

- Timing requirements are vital!
  - A response that's late is incorrect and system failure results.
  - Activities must be completed by a specified deadline, always.
  - Deadlines can be a specific time, a time interval, or the arrival of an event.
  - If a deadline is missed the task fails. This demands that the system has the ability to predict how long computations will take in advance.
- Example: Pacemaker – If the system takes longer than expected to initiate treatment, patient death could result.

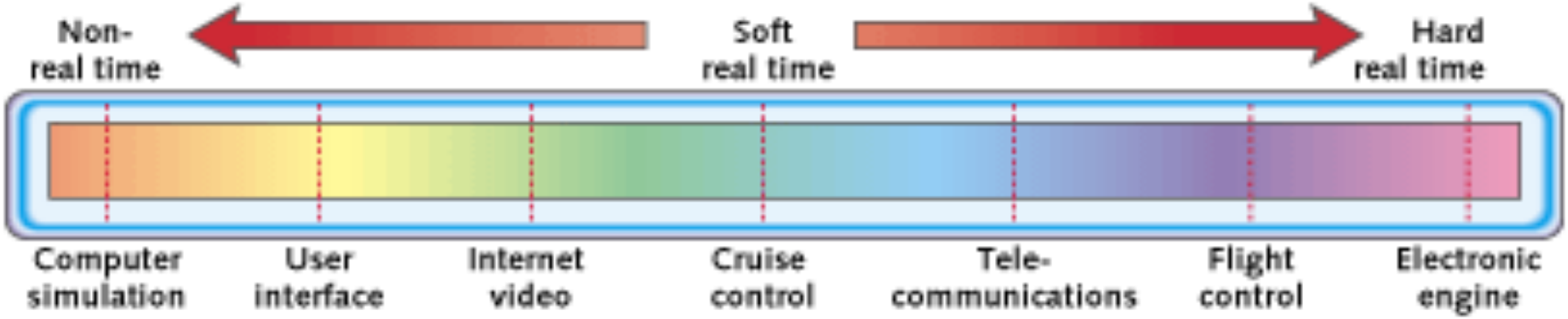
Firm Real-time Systems

- Timing requirements are a combination of both hard and soft ones. Typically the computation will have a shorter soft requirement and a longer hard requirement.
- Example: Ventilator – The system must ventilate a patient so many times within a given time period. But a few second delay in the initiation of the patient's breath is allowed, but not more.

Real-time Systems

- The distinction between systems can obviously become fuzzy.
  - At one end of the software spectrum are non-real-time systems where all deadlines can be missed.
  - At the other end are hard real-time systems where every deadline must be met.

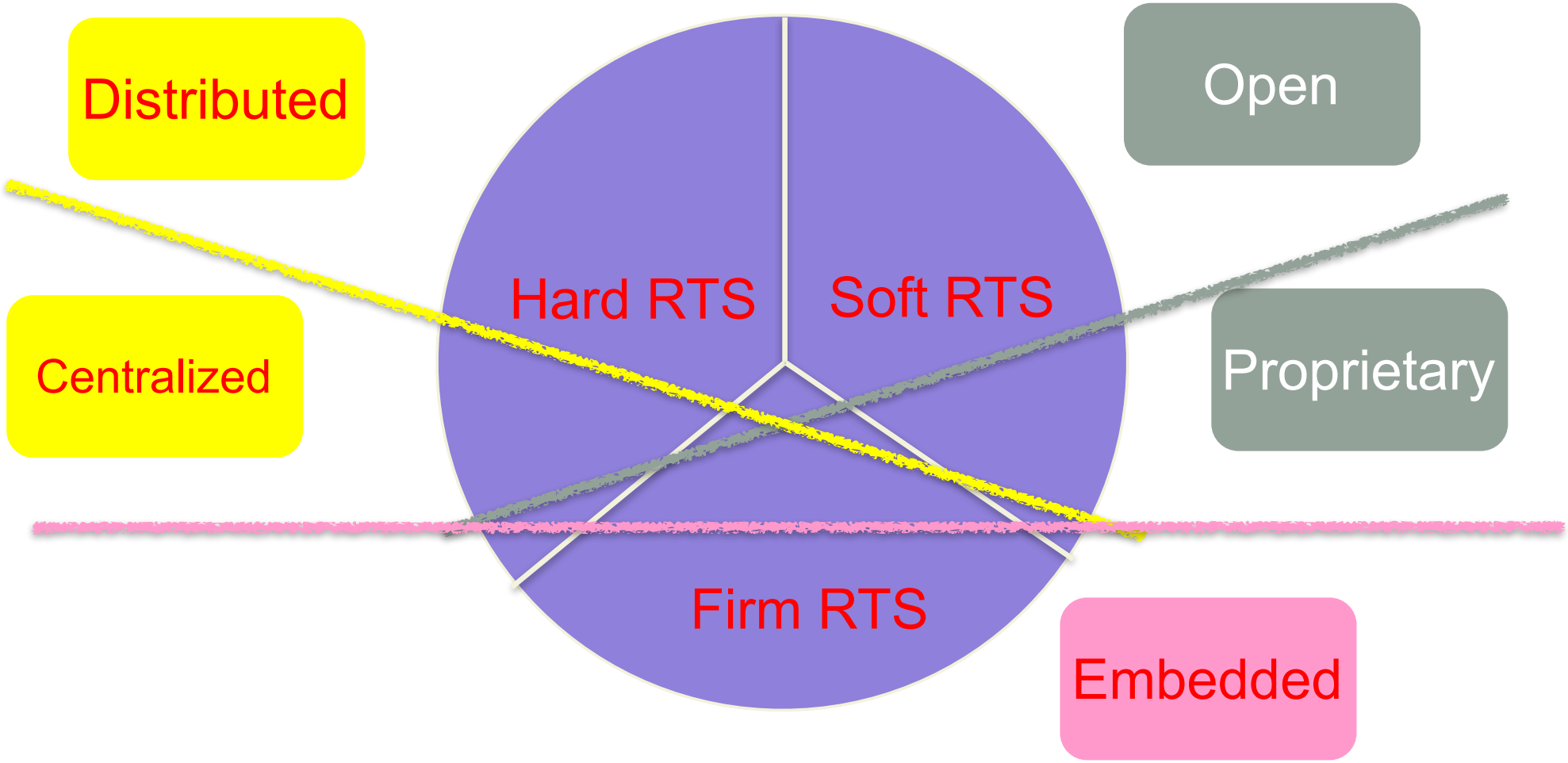
Figure 1: The real-time spectrum



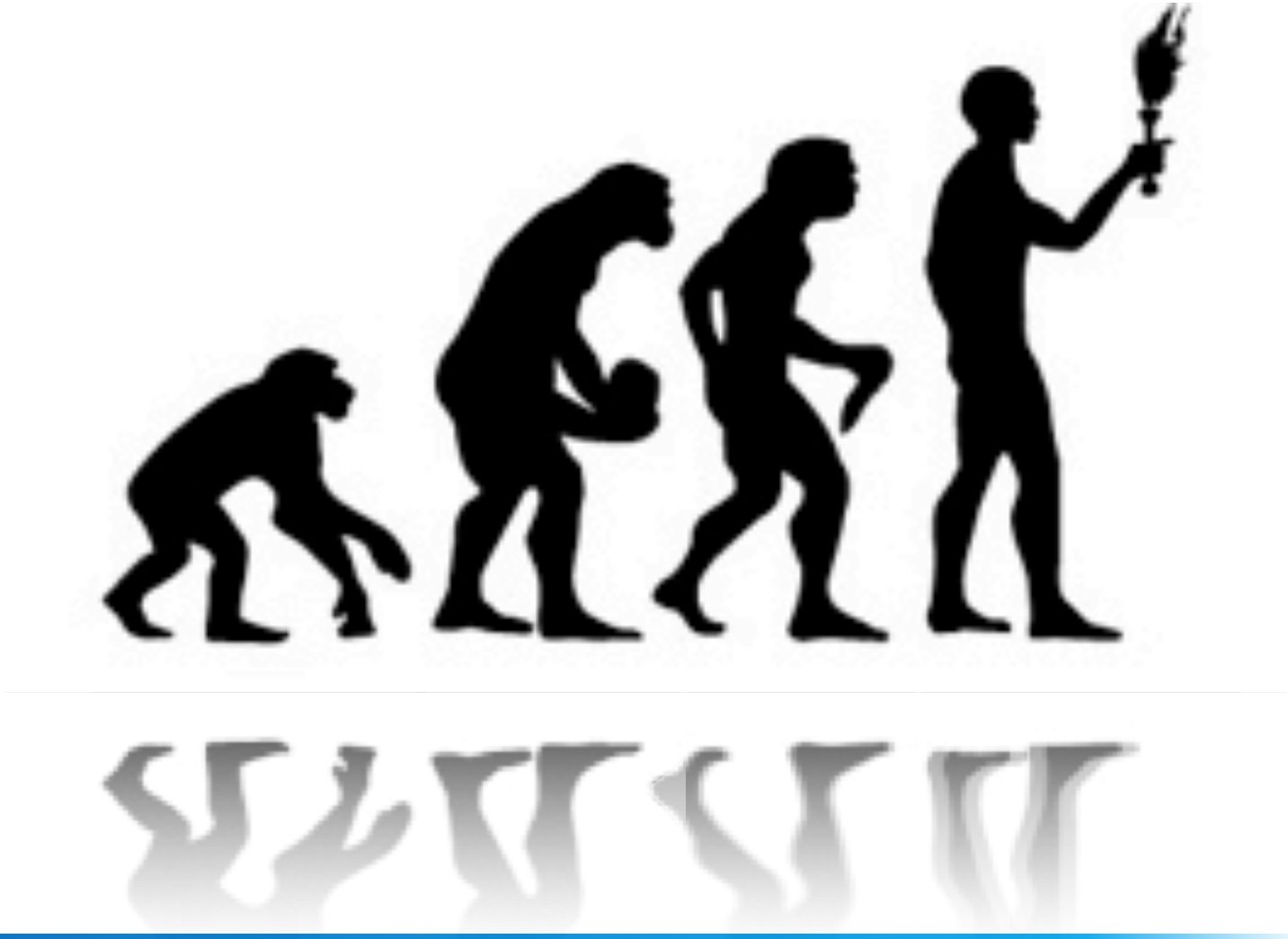
Misconceptions of real-time systems

- Real-time computing is equivalent to fast computing.
- Real-time programming is assembly coding.
- “Real-time” is performance engineering.
- Real-time systems function in a static environment (or closed system).

Real-Time Systems (RTS)

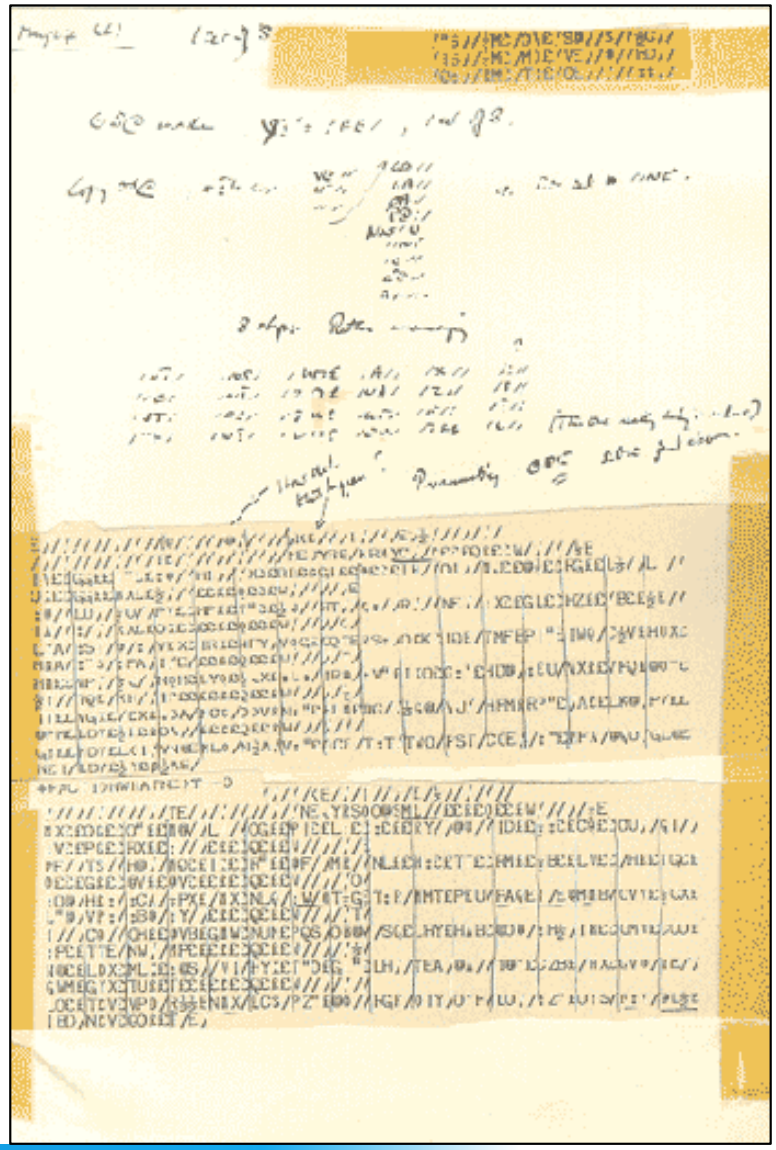


Milestones of Embedded Systems (Reference)

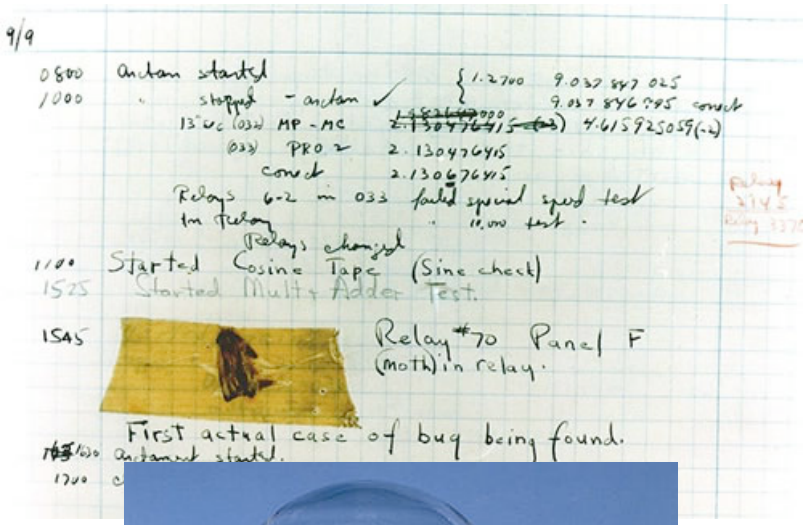


~ 1940

- 1936: Alan Turing publishes article "On Computable Numbers, with an Application to the Entscheidungsproblem" which paves the way not only for computers but for stored-program architectures.
- 1939: John Atanasoff and Clifford Berry create a prototype of the Atanasoff–Berry Computer (ABC), the first digital computer.



1940 ~ 1960



- 1945: First computer bug found by Grace Hopper.
- 1947: The first practical point-contact transistor at Bell Labs by William Shockley, John Bardeen and Walter Brattain
- 1950: Assembly languages were first developed.


```
;
;  enable the timer for tics
;
ticson:
  ld      a,0
  out0    tmdr0l  ; set timer count lo=0
  out0    rldr0l  ; set reload
  ld      a,tmrtic
  out0    tmdr0h  ; set reload and count=1e00=60 tics/sec
  out0    rldr0h
  ld      a,11h
  out0    tcr ; enable timer 0 + interrupts
  ret
```

1960 ~ 1980

- 1963: AGC, the first embedded systems, was installed on Appollo 7 through 17.
  - Autonetics D-17 guidance computer for the Minuteman missile
- 1971: Intel releases first microprocessor, the 4004, a 4-bit central processing unit.
- 1972: Dennis Ritchie developed C, the most used programming language for embedded systems.
- 1973: C. L. Liu and James W. Layland published the most fundamental scheduling theory for embedded real-time systems: “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.”
- 1976: Apple I computer was released.

Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment

C. L. LIU

Project MAC, Massachusetts Institute of Technology

AND

JAMES W. LAYLAND

Jet Propulsion Laboratory, California Institute of Technology

ABSTRACT. The problem of multiprogram scheduling on a single processor is studied from the viewpoint of the characteristics peculiar to the program functions that need guaranteed service. It is shown that an optimum fixed priority scheduler possesses an upper bound to processor utilization which may be as low as 70 percent for large task sets. It is also shown that full processor utilization can be achieved by dynamically assigning priorities on the basis of their current deadlines. A combination of these two scheduling techniques is also discussed.

KEY WORDS AND PHRASES: real-time multiprogramming, scheduling, multiprogram scheduling, dynamic scheduling, priority assignment, processor utilization, deadline driven scheduling

CR CATEGORIES: 3.80, 3.82, 3.83, 4.32

1. *Introduction*

The use of computers for control and monitoring of industrial processes has expanded greatly in recent years, and will probably expand even more dramatically in the near future. Often, the computer used in such an application is shared between a certain number of time-critical control and monitor functions and a non-time-critical batch processing job stream. In other installations, however, no non-time-critical jobs exist, and efficient use of the computer can only be achieved by a careful scheduling of the time-critical control and monitor functions themselves. This latter group might be termed “pure process control” and provides the background for the combinatoric scheduling analyses presented in this paper. Two

This paper presents the results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, under Contract No. NAS-7-100, sponsored by the National Aeronautics and Space Administration.

Authors’ present addresses: C. L. Liu, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801; James W. Layland, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91103.

Journal of the Association for Computing Machinery, Vol. 20, No. 1, January 1973, pp. 46–61.

```
/* Funct                                          ents
 If Fifo
 else fu
 FifoHea
 */
int Fifo
{
  if (Fi
  {
     ret
  }
   FifoB                                         ntry;
  return
}
```

1980 ~ 2000

- 1980: Hunter&Ready (James Ready and Colin Hunter's company) release first commercial operating system for embedded systems, VRTX (Versatile Real-Time Executive).
- 1980s: WindRiver acquires rights to resell VRTX with an extension named VxWorks.
- 1980: Intel introduces 8051 (Harvard architecture, single ch micro-controller).
- 1985: Acorn Computers Ltd. makes samples of ARM1 architecture available. ARM2 ships in 1986.
- 1985: Joint Test Action Group (JTAG) forms
- 1989: First Embedded Systems Conference is held at the Sir Francis Drake hotel in San Francisco.
- 1995: Sun release JAVA.

```
/**
 * Helper class that returns a random number.
 */
private static int getRandom(int mod) {
    Random rand = new Random();
    return Math.abs(rand.nextInt()) % mod + 1;
}
```

- 1996: Microsoft releases first version of Windows Embedded CE.
- 1997: UML version 1.0 proposed to the OMG.

2000 ~ 2010

- 2001: First Tele-surgery machine
- 2004: Sony and IBM begin producing cell computer chips, a supercomputer on a chi was on your palm.
- 2004: First urban challenge and no participant finished the challenge.
- 2005:IBM, Intel and AMD released their first multi-core processors.
- 2007: Third urban challenge

Prof Jacques Marescaux, New York & European Institute of Telesurgery, Strasbourg

Round distance =14,000 km

Round Trip Time = 200 msec; video and hi-speed fibre-optic link

June 2001: Johns Hopkins University, Baltimore & Rome Policlinico Casilino University

http://news.bbc.co.uk/2/hi/science/nature/1552211.stm

Participants for 2007 Challenges

- Technical Abilities:
  - Real-time by-wire actuator control
  - Mission and path planning
  - Perception and multiple sensor fusion
  - Vehicle behavior
  - Situational awareness
  - System Engineering

Thanks to Prof. Rajkumar @CMU for pictures and texts

Trends for Embedded Systems Design

- Machine centric to Human Centric
- Performance centric to Environment friendly
- Computing everywhere to Invisible computing

Cyber-Physical Systems

Summary for Embedded System Design

Embedded system design is a science for understanding

- interaction among hardware and software components,
- resource management,
- performance predicability, and
- systems integration.

so as to design robust and cost effective embedded systems.

Real-Time Operating Systems

What is a Real-Time Operating System?

- What is a real-time operating system?
  - An operating system enforcing timing constraints, Lynx, pSOS, VxWorks, eCOS, uCLinux, LynxOS, RTLinux, KURT,RTAI, uC/OS-II, …

RTOS at a Glance

- **AMX, KwikNet, KwikPeg** *(from KADAK Products Ltd.)*
- **C EXECUTIVE** *(from JMI Software Systems, Inc.)*
- **CMX-RTX** *(from CMX Systems, Inc.)*
- **DeltaOS** *(from CoreTek Systems, Inc.)*
- **eCos** *(from Red Hat, Inc.)*
- **embOS** *(from SEGGER Microcontroller Systeme GmbH)*
- **eRTOS** *(from JK microsystems, Inc.)*
- **ETS** *(from VenturCom)*
- **EYRX** *(from Eyring Corporation)*
- **INTEGRITY** *(from Green Hills Software, Inc.)*
- **INtime® real time extension to Windows®** *(from TenAsys Corporation)*
- **IRIX** *(from SGI)*
- **iRMX** *(from TenAsys Corporation)*
- **Jbed** *(from esmertec, inc.)*
- **LynxOS** *(from LynuxWorks)*
- **MQX** *(from Precise Software Technologies Inc)*
- **Nucleus PLUS** *(AcceleratedTechnology, ESD Mentor Graphics)*
- **On Time RTOS-32** *(from On Time Informatik GmbH)*
- **OS-9** *(from Microware Systems Corporation)*
- **OSE** *(from OSE Systems)*
- **PDOS** *(from Eyring Corporation)*
- **PSX** *(from JMI Software Systems, Inc.)*
- **QNX Neutrino** *(from QNX Software Systems Ltd.)*
- **QNX4** *(from QNX Software Systems Ltd.)*
- **REDICE-Linux** *(from REDSonic, Inc.)*
- **RTLinux** *(from Finite State Machine Labs, Inc.)*
- **RTX 5.0** *(from VenturCom)*
- **Portos** *(from Rabih Chrabieh)*
- **smx** *(Micro Digital, Inc.)*
- **SuperTask!** *(from US Software)*
- **ThreadX** *(from Express Logic, Inc.)*
- **Treck MicroC/OS-II** *(from Elmic Systems USA, Inc.)*
- **TronTask!** *(from US Software)*
- **TTPos:** *(from TTTech Computertechnik AG)*
- **VxWorks 5.4** *(from Wind River)*
- **SCORE, DACS and TADS** *(from DDC-I)*
- **Nimble - the SoC RTOS** *(from Eddy Solutions)*
- **Nucleus** *(from Accelerated Technology)*
- **Fusion RTOS** *(from DSP OS, Inc.)*
- **FreeRTOS** *(from Richard Barry)*

Systems Issues

- In RTS, the OS and AP are very tightly coupled, than time-sharing systems.
  - shared memory, special buses (instruction, data, event, memory, control, invalidate, …).
- A RTOS must response to internal and external events **deterministically**.
- Low-priority tasks may wait for high-priority task or events indefinitely.
- System architecture needs to provide high computational speed, high-speed interrupt handling, and high I/O throughput, + fault-tolerance.

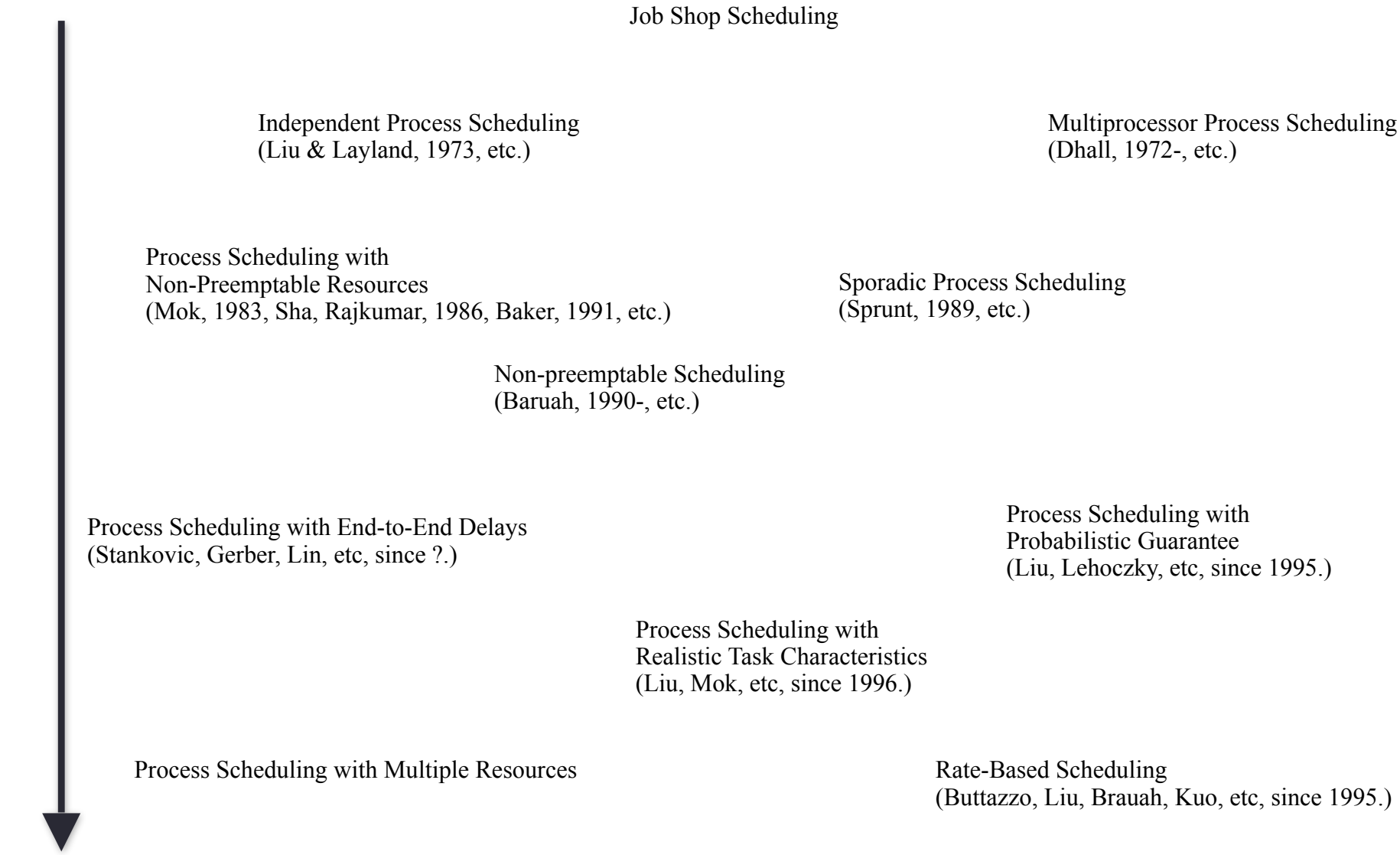
Real-Time Scheduling

Real-Time scheduling algorithms

- We use real-time scheduling algorithms every day.
  - Dynamic priority scheduling algorithms
    - How do you plan your days to do the homework and exams?
    - Earliest Deadline First algorithm [Liu and Layland 1973]
  - Static priority scheduling algorithms
    - Do you prioritize your works?
    - Rate-Monotonic algorithm [Liu and Layland 1973]
- The algorithms should schedule the tasks to meeting the requirements.

Introduction to Real-Time Process Scheduling

- Q: Many theories and algorithms in real-time process scheduling seem to have simplified assumptions without direct solutions to engineers' problems. Why should we know them?
- A:
  - Provide insight in choosing a good system design and scheduling algorithm.
  - Avoid poor or erroneous choices.


Time
Job Shop Scheduling
Independent Process Scheduling
(Liu & Layland, 1973, etc.)
Multiprocessor Process Scheduling
(Dhall, 1972-, etc.)
Process Scheduling with
Non-Preemptable Resources
(Mok, 1983, Sha, Rajkumar, 1986, Baker, 1991, etc.)
Sporadic Process Scheduling
(Sprunt, 1989, etc.)
Non-preemptable Scheduling
(Baruah, 1990-, etc.)
Process Scheduling with End-to-End Delays
(Stankovic, Gerber, Lin, etc, since ?.)
Process Scheduling with
Probabilistic Guarantee
(Liu, Lehoczky, etc, since 1995.)
Process Scheduling with
Realistic Task Characteristics
(Liu, Mok, etc, since 1996.)
Process Scheduling with Multiple Resources
Rate-Based Scheduling
(Buttazzo, Liu, Brauah, Kuo, etc, since 1995.)

Uni-processor Scheduling

- Fixed-Priority vs. Dynamic-Priority Scheduling
- Rate-Monotonic Scheduling Algorithm
- Earliest Deadline First Scheduling

Process Model

- Periodic process
  - each periodic process arrives at a regular frequency – a special case of demand.
  - r: ready time, d: relative deadline, p: period, c: worst case computation time.
  - For example, maintaining a display
- Sporadic process
  - An aperiodic process with bounded inter-arrival time p.
  - For example, turning on a light
- Other requirements and issues:
  - process synchronization including precedence and critical sections, process value, etc.

Performance Metrics

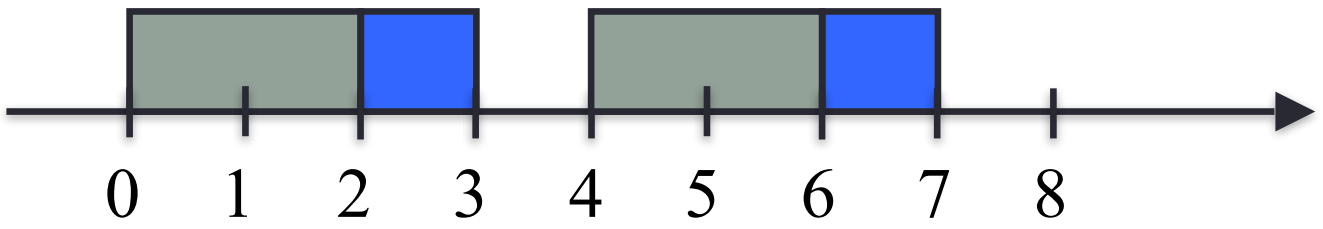
- Metrics for hard real-time processes:
  - Schedulability, etc.
- Metrics for soft real-time processes:
  - Miss ratio
  - Accumulated value
  - Response time, etc.
- Other metrics:
  - Optimality, overload handling, mode-change handling, stability, jitter, etc.
  - Combinations of metrics.

Definitions

- Preemptive scheduling: allows process preemptions. (vs. nonpreemptive scheduling)
- Online scheduling: allocates resources for processes depending on the current workload. (vs. offline scheduling)
- Static scheduling: operates on a fixed set of processes and produces a single schedule that is fixed at all time. (vs. dynamic scheduling)
- Firm real-time process: will be killed after it misses its deadline. (vs. hard and soft real-time)
- Fixed-priority scheduling: in which the priority of each process is fixed for any instantiation. (vs. dynamic-priority scheduling)

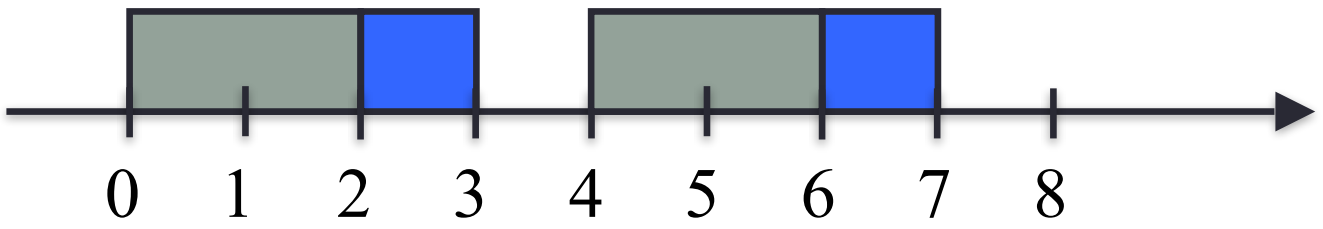
Rate Monotonic Scheduling Algorithm

- Assumptions:
  - all periodic fixed-priority processes
  - relative deadline = period
  - independent process - no non-preemptable resources
- Rate Monotonic (RM) Scheduling Algorithm
  - RM priority assignment: priority ~ 1/period.
  - preemptive priority-driven scheduling.
- Example: T_1 (p_1=4, c_1=2) and T_2 (p_2=5, c_2=1)



Rate Monotonic Scheduling Algorithm

- Critical Instant
  - An instant at which a request of the process have the largest completion/response time.
  - An instance at which the process is requested simultaneously with requests of all higher priority processes.
- Usages
  - Worst-case analysis
  - Fully utilization of the processor power
  - Example: T_1 (p_1=4, c_1=2) and T_2 (p_2=5, c_2=1)



Rate-Monotonic Analysis

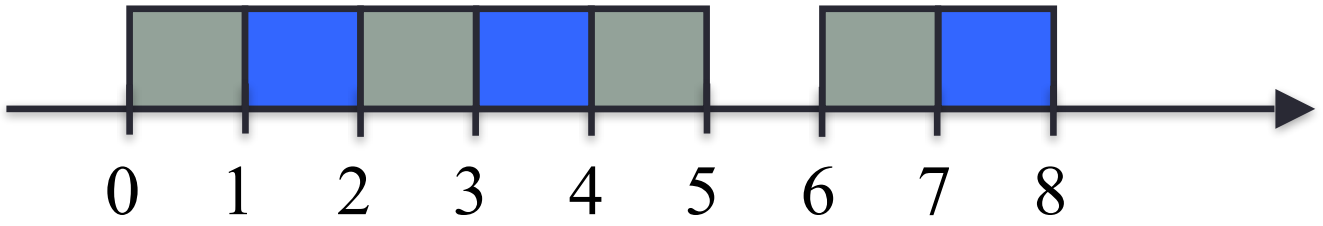
- Schedulability Test:
  - A sufficient but not necessary condition.
  - Achievable utilization factor αof a scheduling policy P: any process set with total utilization factor $\sum \frac{c_i}{P_i}$ no more than α is schedulable.
  - Given n processes, $\alpha = n(2^{1/n} - 1)$
- Stability:
  - Let processes be sorted in RM order. The ith process is schedulable if

$$\sum_{j=1}^{i} \frac{c_j}{p_j} \le i(2^{1/i} - 1)$$

  - An optimal fixed priority scheduling algorithm

Earliest Deadline First Scheduling Algorithm

- Assumptions (similar to RM):
  - all periodic dynamic-priority processes
  - relative deadline = period
  - independent process - no non-preemptable resources
- Earliest Deadline First (EDF) Scheduling Algorithm:
  - EDF priority assignment: priority ~ absolute deadline. i.e., arrival time **t** + relative deadline **d**.
  - preemptive priority-driven scheduling
- Example: $T_1(c_1=1, p_1=2)$, $T_2(c_2=2, p_2=7)$



Earliest Deadline First (EDF) Scheduling Algorithm

- Schedulability Test:
  - A sufficient and necessary condition
  - Any process set is schedulable by EDF iff

$$\sum \frac{c_i}{P_i} \leq 1$$

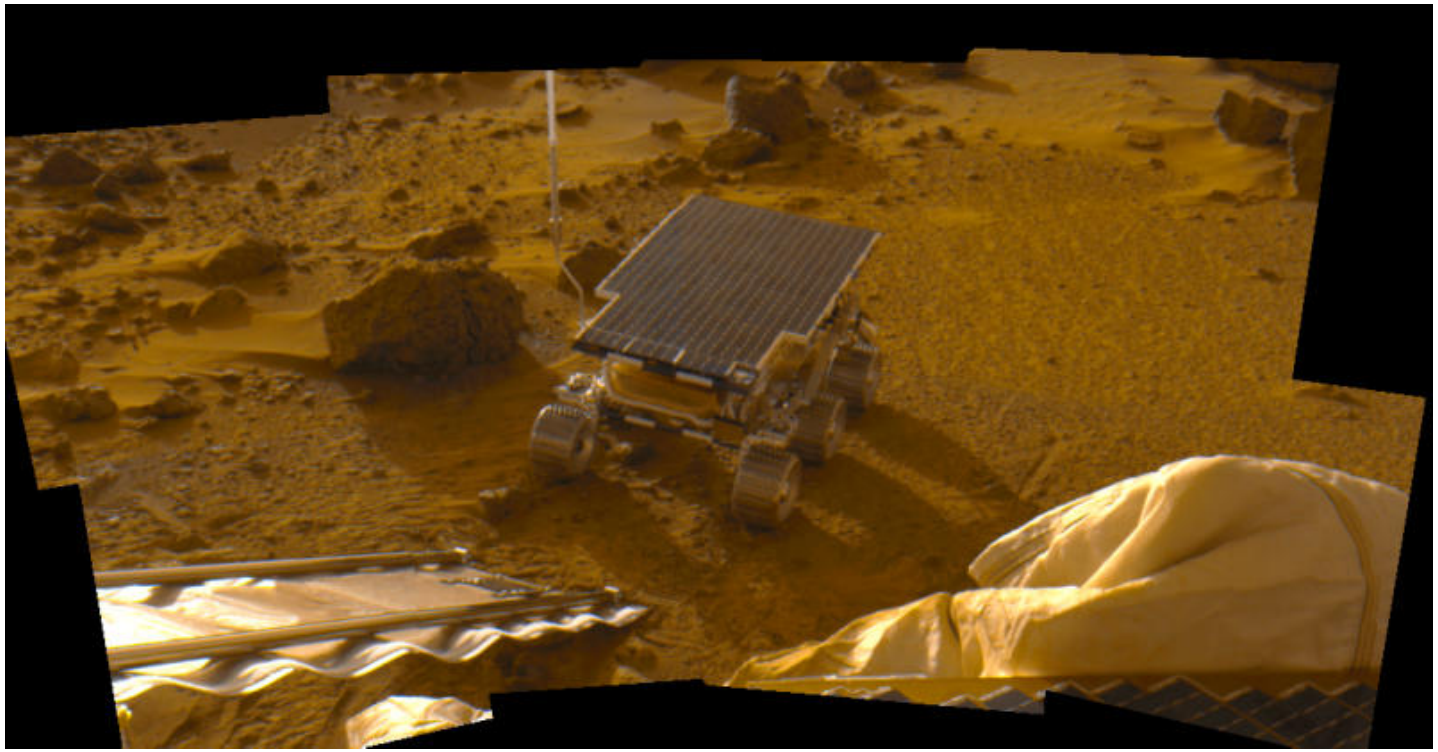
- EDF is optimal for any independent process scheduling algorithms.
- However, its implementation has considerable overheads on OS's with a fixed-priority scheduler and is bad for (transiently) overloaded systems.

Priority Inversion and Schedulability Analysis

(Optional)

What Happened on Mars?

- Pathfinder was launched on Dec. 4 1996 and landed on Mars on July 4, 1997.



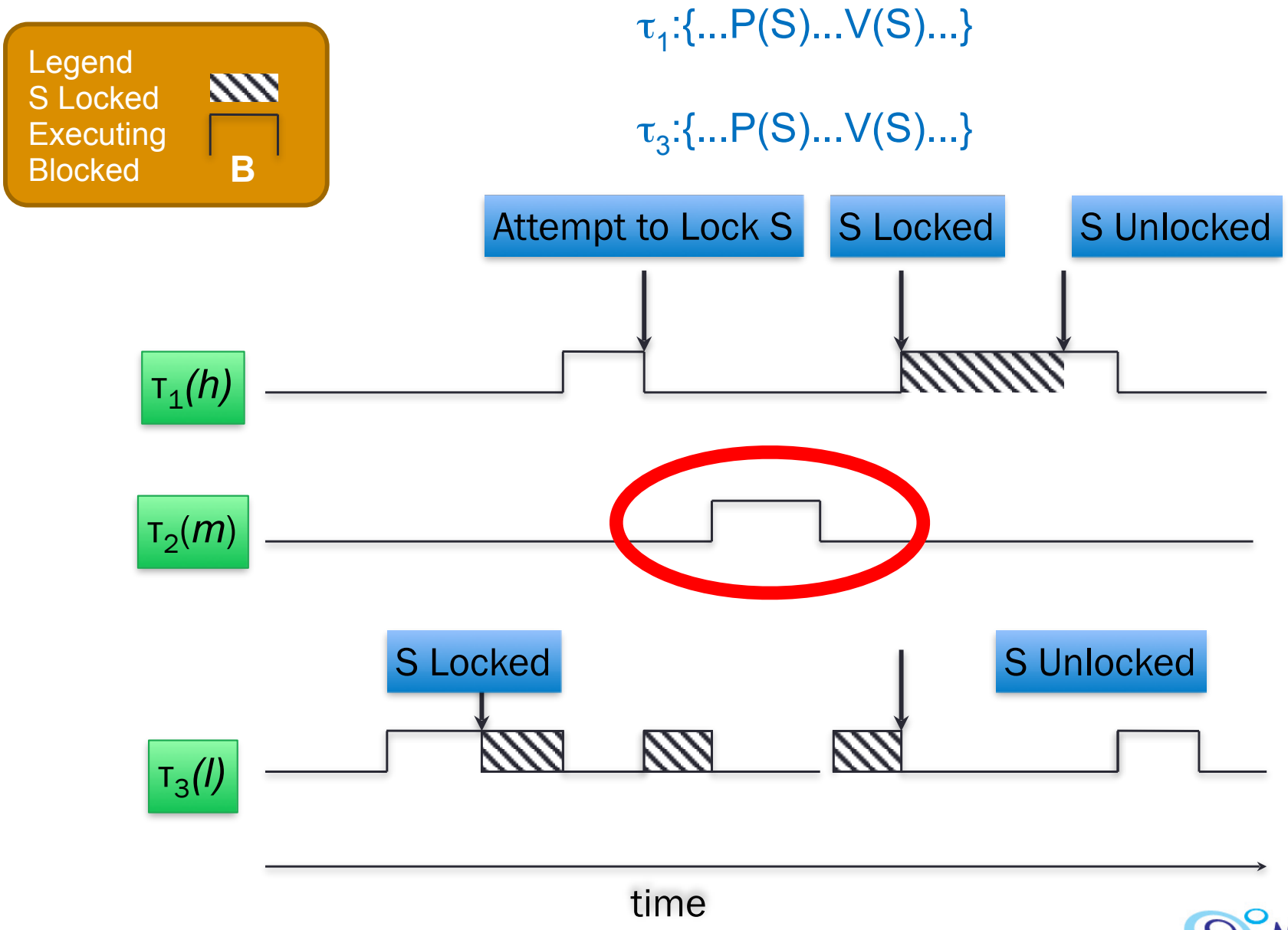
- All of sudden, the pathfinder kept reset itself and doing nothing.

What happened on Mars?

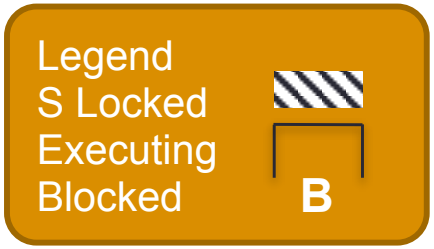
- Pathfinder contained an "information bus", which you can think of as a shared memory area used for passing information between different components of the spacecraft.
- A high priority task was blocked for unexpected long time interval.
- After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset.

- This scenario is a classic case of priority inversion.

Unbounded Priority Inversion

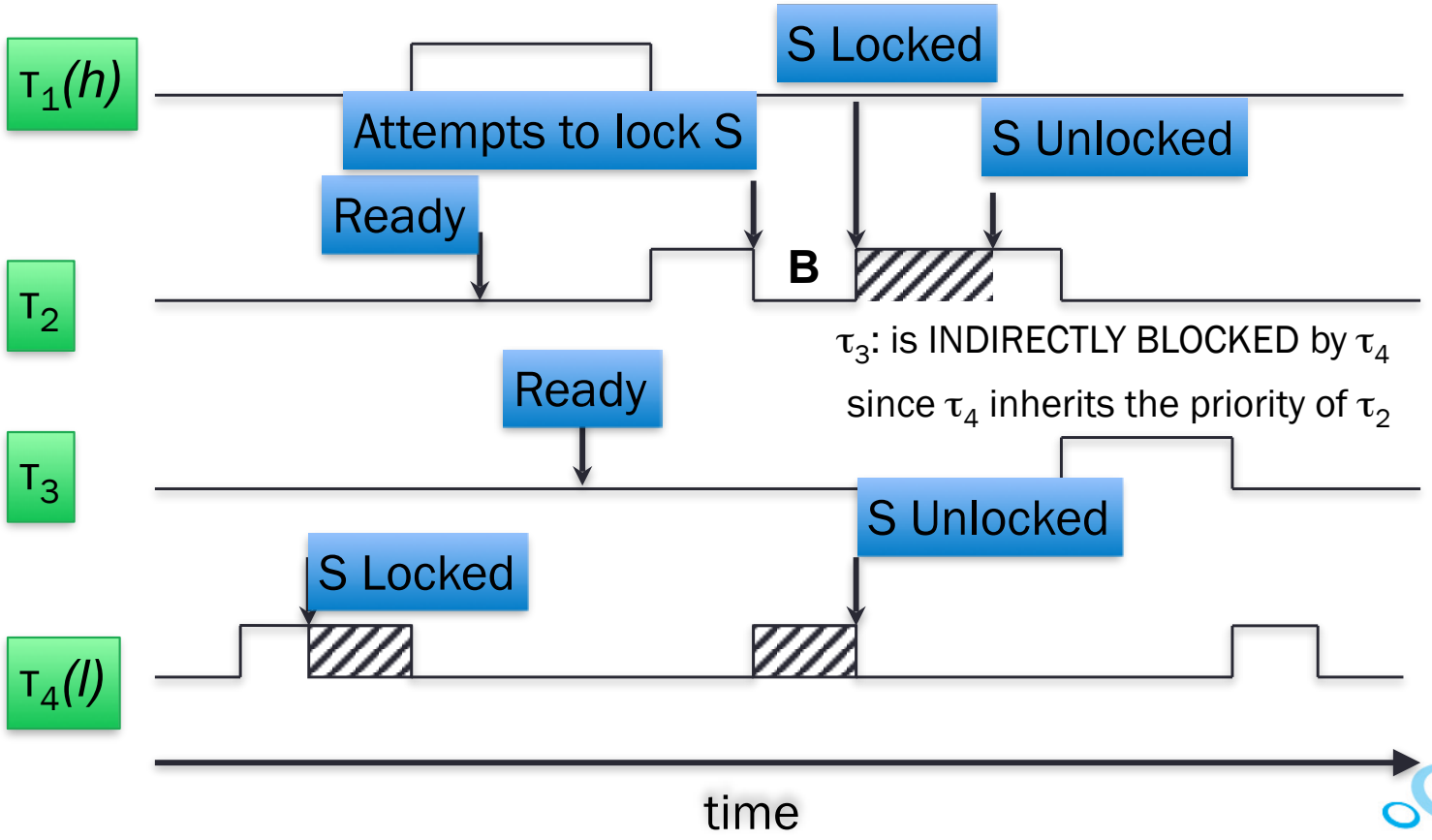


Basic Priority Inheritance Protocol - 1

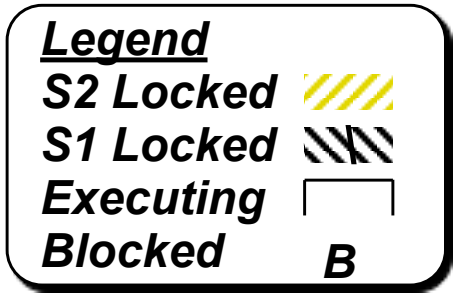


τ_2:{...P(S)...V(S)...}

τ_4:{...P(S)...V(S)...}



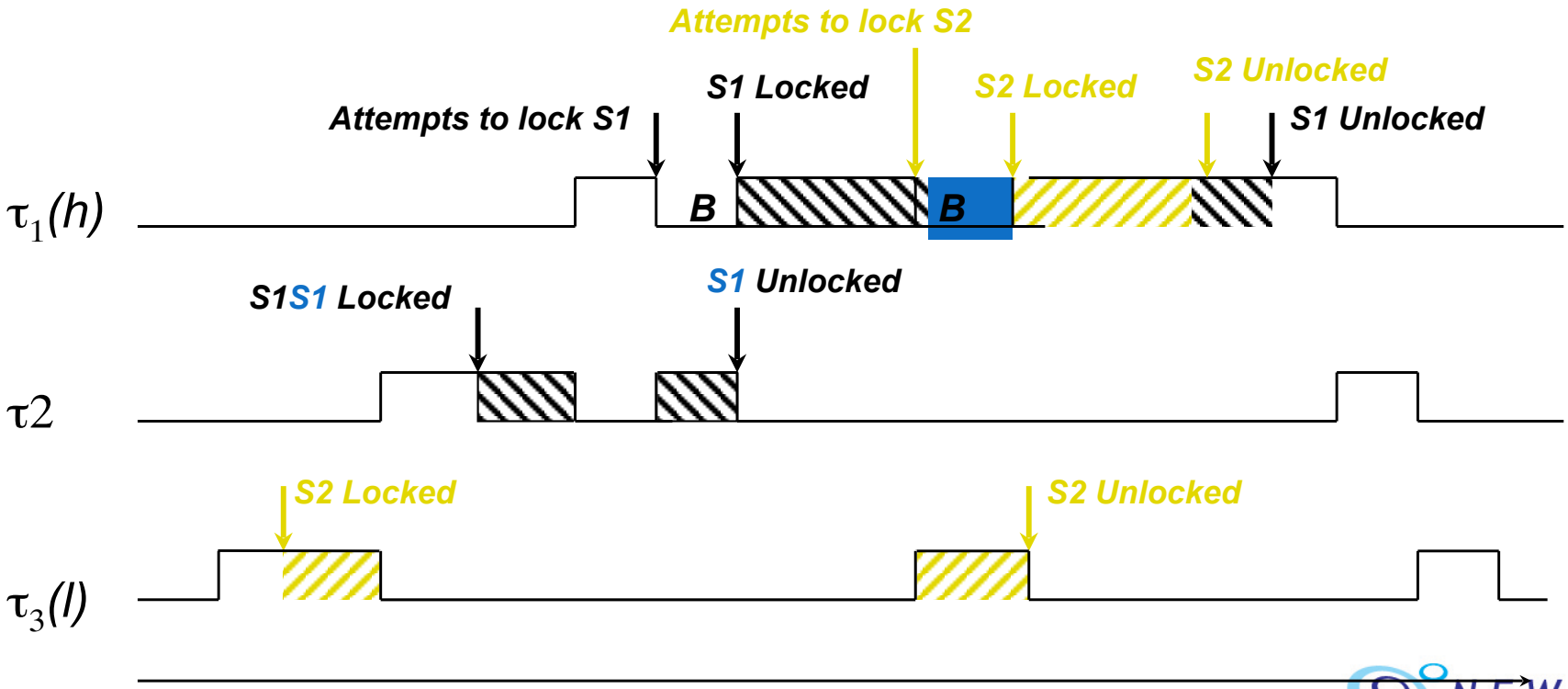
Chained Blocking



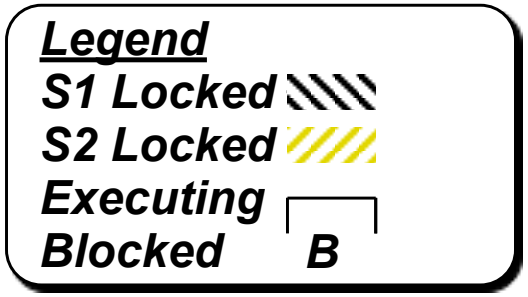
τ_1:{...P(S1)...P(S2)...V(S2)...V(S1)...}

τ_2:{...P(S1)...V(S1)...}

τ_3:{...P(S2)...V(S2)...}

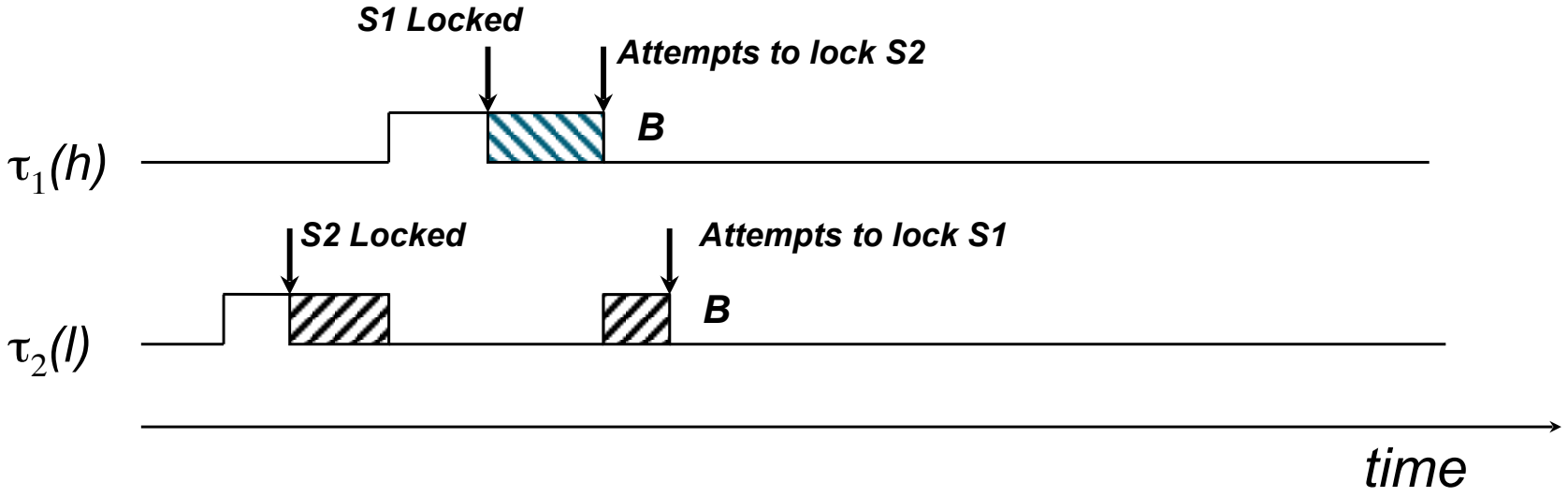


Deadlock Under BPIP



τ_1:{...P(S1)...P(S2)...V(S2)...V(S1)...}

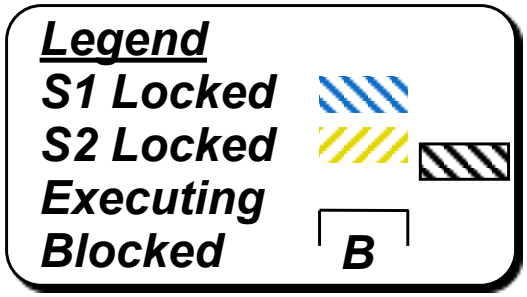
τ_2:{...P(S2)...P(S1)...V(S1)...V(S2)...}



Priority Ceiling Protocol

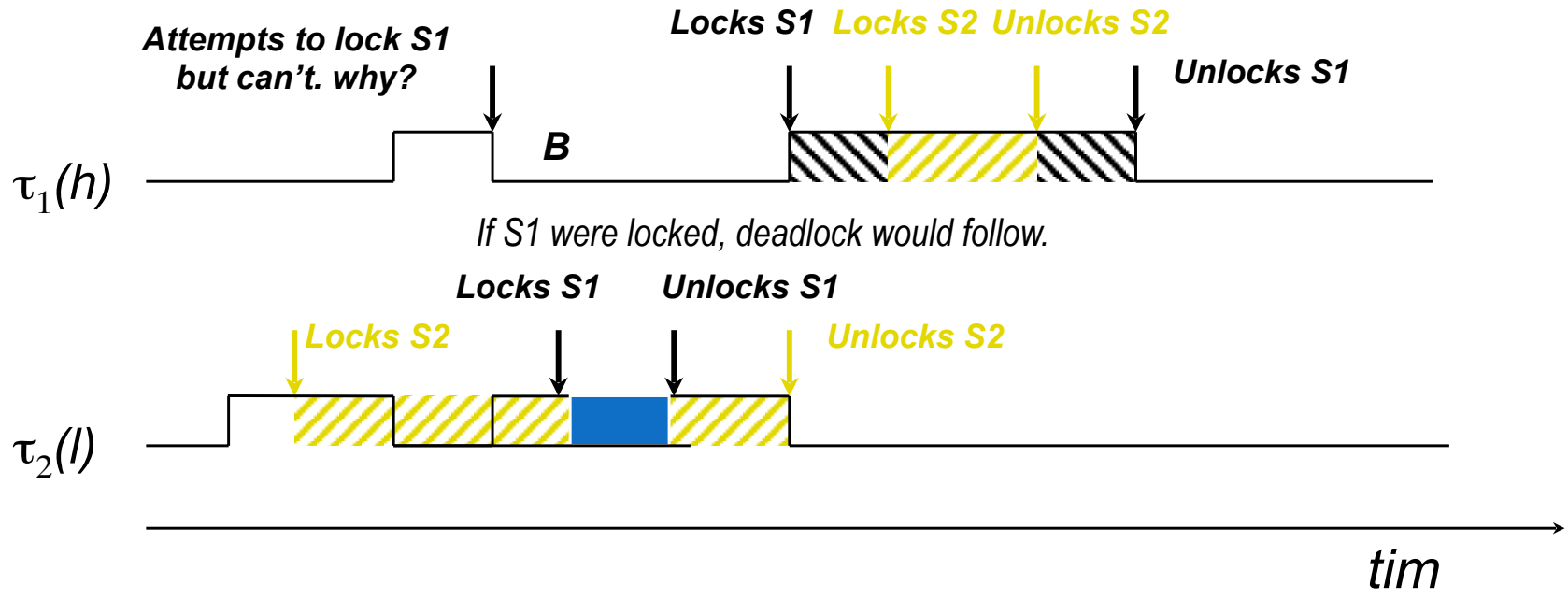
- A priority ceiling is assigned to each semaphore, which is equal to the highest priority task that may use this semaphore.
- A task can lock a semaphore if and only if its priority is higher than the priority ceilings of all semaphores ALREADY LOCKED by ***other*** tasks.
- If a task is blocked by lower priority tasks, the lower priority task inherits its priority.
- Under priority ceiling protocol, a task can be blocked by lower priority tasks at most once no matter how many semaphores they share. In addition, tasks cannot be deadlocked.
- Under priority inheritance protocol, tasks could be deadlocked and chained blocking is a fact of life. But a task is blocked at most by n lower priority tasks sharing resources with it or with higher priority tasks, when there is no deadlock.

Deadlock Avoidance: Using PCP



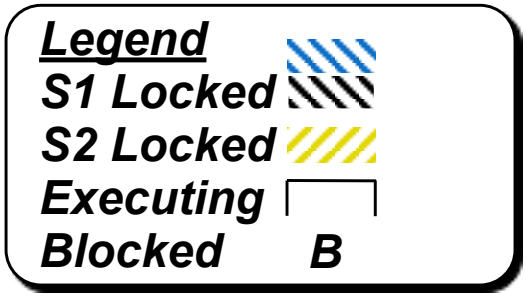
τ_1:{...P(S1)...P(S2)...V(S2)...V(S1)...}

τ_2:{...P(S2)...P(S1)...V(S1)...V(S2)...}



Note: *Task τ_2 can still lock S1 since it owns the lock, S1 is not locked by OTHER tasks*

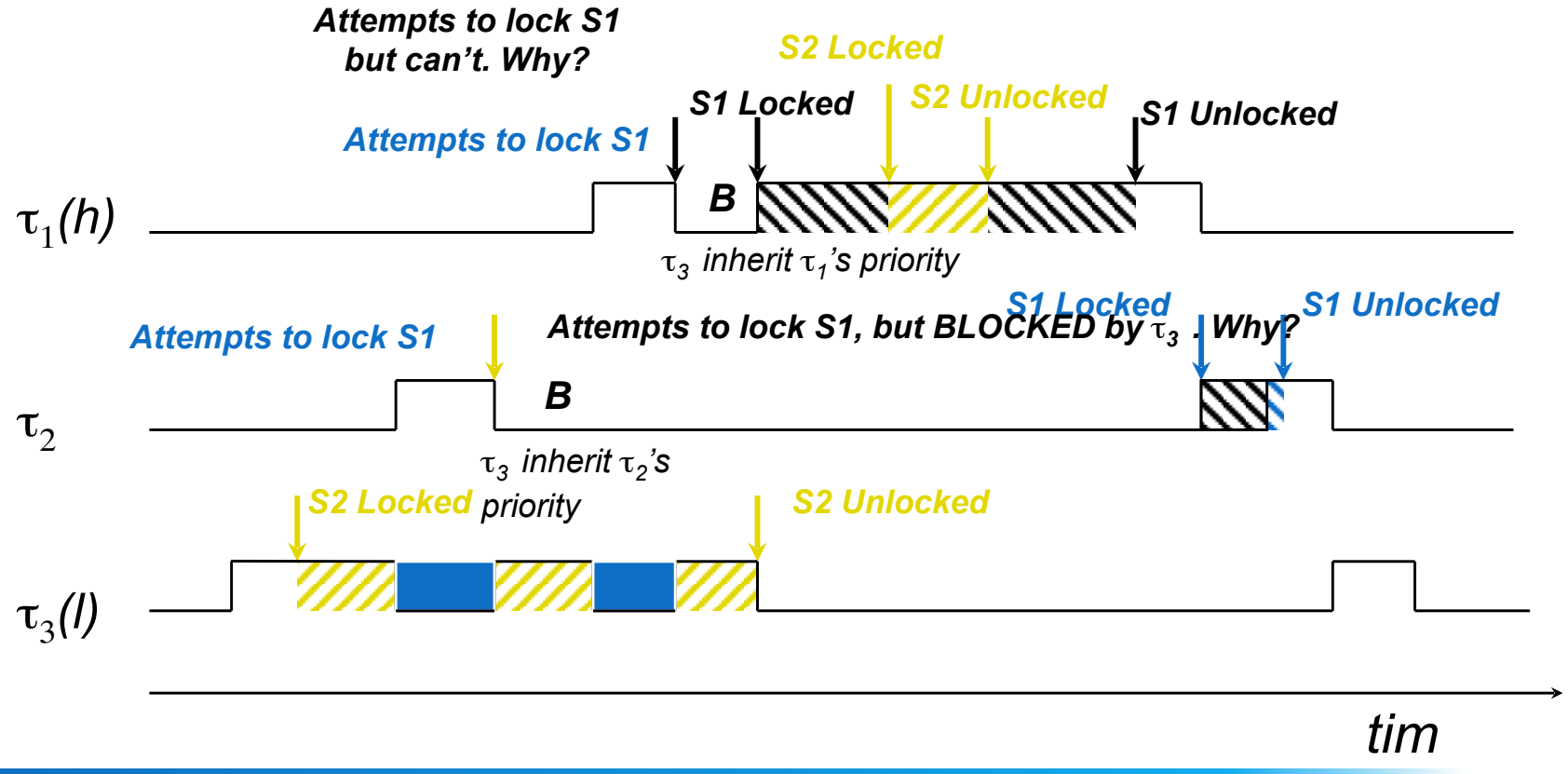
Blocked at Most Once (PCP)



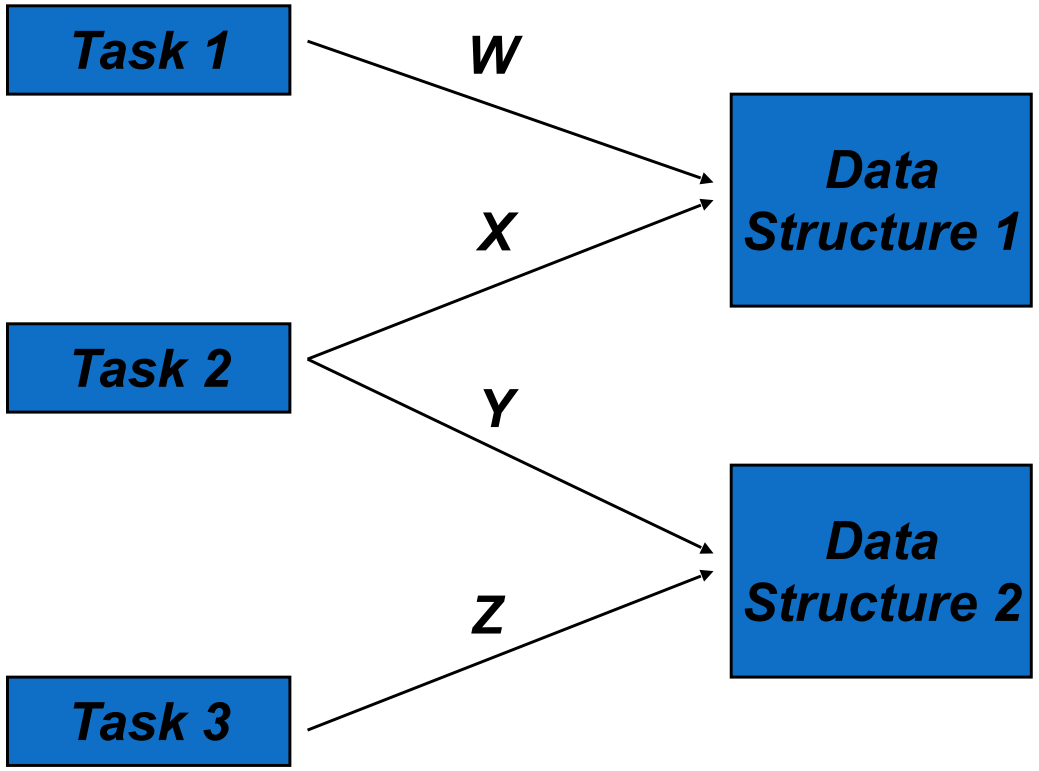
T_1:{...P(S1)...P(S2)...V(S2)...V(S1)...}

T_2:{...P(S1)...V(S1)...}

T_3:{...P(S2)...V(S2)...}



Sample Problem: Worst-Case Blocking Times



Sample Problem: Worst-Case Blocking Times

- Un-nested semaphores
  - First, examine task 3. Recall that *blocking* is when a lower priority task delays the execution of a higher priority task. Since task 3 has the lowest priority, its blocking time will be 0.
  - For task 2, it could only be blocked by task 3. Thus, its B is equal to Z.
  - Similarly, task 1's B is equal to X. Because semaphores cannot be nested, data structure 2 will not affect task 1.
- Nested semaphores
  - Task 3 and 2 are the same as the un-nested case.
  - For task 1, imagine the worst case sequence of events.

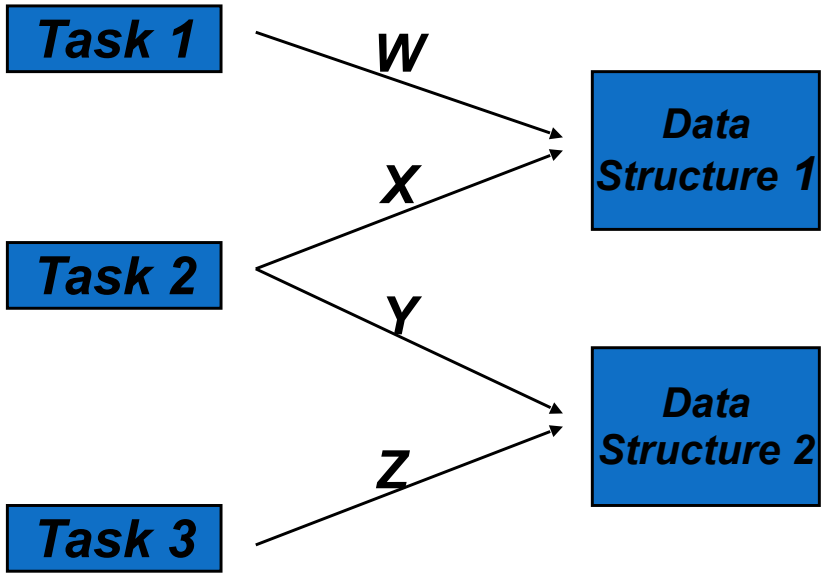
Sample Problem: Worst-Case Blocking Times

Nested semaphores

- The worst sequence for Task 1:
  1. Task 3 acquires data structure 2.
  2. Task 2 acquires data structure 1.
  3. Task 2 asks for data structure 2 (but cannot get it).
     - What is Task 3's priority now?
     - Task 3 has priority of 2.
  4. Task 1 asks for data structure 1.
     - What are Task 2's and 3's priorities now?
     - They both have priority of 1.
  5. Task 3 finishes its critical section (Z units).
  6. Task 2 acquires data structure 2 and finishes its critical section (Y units).
  7. Task 2 finishes its critical section for data structure 1 (X units).
- Total time: X + Y + Z.

Summary

- For a nested semaphore itself, the blocking time it may inflict is the entire nested locks, that is x+y.
- Nested semaphore may link the blocking time from a semaphore that is not directly used by a task. In this case, the blocking time that can be caused by z.



Sample Problem

- Suppose that we have three tasks and there are one data structures shared by tasks τ_1 and task τ_2 and another shared by task τ_1 and τ_3. Task τ_2's critical section is 1 unit long while task τ_3's critical section is 2 units long.
- Fill in the blocking times and determine if these 3 tasks are schedulable under PIP. (Note that the critical section is included in the C's, because it is just the part of code that uses the shared data.)

Basic Priority Inheritance

	C	T	B	D
Task τ1	1	4	?	
Task τ2	2	6	?	1
Task τ3	4	13	?	

Priority Ceiling Protocol

	C	T	B	D
Task τ1	1	4	?	
Task τ2	2	6	?	1
Task τ3	4	13	?	

Blocking Under PIP and PCP

Basic Priority Inheritance

	C	T	B	D
Task τ_1	1	4	3	
Task τ_2	2	6	2	1
Task τ_3	4	13	0	

Priority Ceiling Protocol

	C	T	B	D
Task τ_1	1	4	2	
Task τ_2	2	6	2	1
Task τ_3	4	13	0	

Learn from the Master

Real-Time Operating System Summary

	Foreground/Background	Non-Preemptive Kernel	Preemptive Kernel
Interrupt Latency (Time)	MAX(Longest instruction, User int. disable) + Vector to ISR	MAX(Longest instruction, User int. disable, Kernel int. disable) + Vector to ISR	MAX(Longest instruction, User int. disable, Kernel int. disable) + Vector to ISR
Interrupt response (Time)	Int. latency + Save CPU's context	Int. latency + Save CPU's context	Interrupt latency + Save CPU's context + Kernel ISR entry function
Interrupt recovery (Time)	Restore background's context + Return from int.	Restore task's context + Return from int.	Find highest priority task + Restore highest priority task's context + Return from interrupt
Task response (Time)	Background	Longest task + Find highest priority task + Context switch	Find highest priority task + Context switch
ROM size	Application code	Application code + Kernel code	Application code + Kernel code
RAM size	Application code	Application code + Kernel RAM + SUM(Task stacks + MAX(ISR stack))	Application code + Kernel RAM + SUM(Task stacks + MAX(ISR stack))
Services available?	Application code must provide	Yes	Yes