Unit 6.-Synchronization in Real-Time Systems

Concurrency and Distributed Systems

Teaching Unit Objectives

- Understand what Real Time Systems (RTS) are and their particularities about task synchronization.
  - Know the main features of the RTS
  - Understand how task scheduling is performed in RTS and how the analysis of RTS is carried out.

- Analyze the synchronization problems that appear in this type of systems.
  - Identify the priority inversion problem.
  - Learn resolution strategies to solve these synchronization problems.

Content

- Introduction to RTS
  - Definition of a Real-Time System (RTS)
  - Features of RTS
  - Task Scheduling in RTS
  - Analysis of RTS
- Task Synchronization
  - The Priority Inversion Problem
  - The Immediate Priority Ceiling Protocol
  - Response time with blocking factors

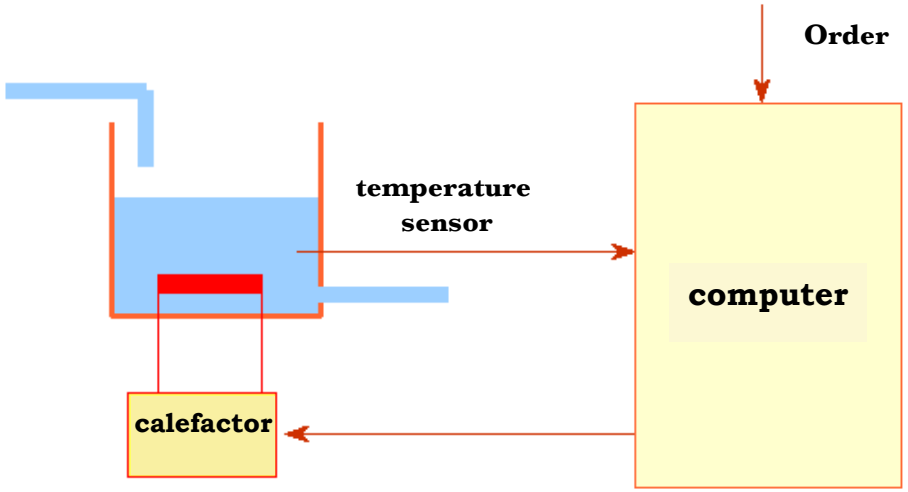
What is a Real Time System?

- A **real time system** (RTS) is a computing system that:
  - Repeatedly interacts with its physical environment
  - Responds to the stimuli received from the enviroment within a period of time
  - For the **correct** behaviour of the system is not enough that the actions are correct, but they must be executed within the specified time interval

The **time** in which actions are executed in the system is **significant**

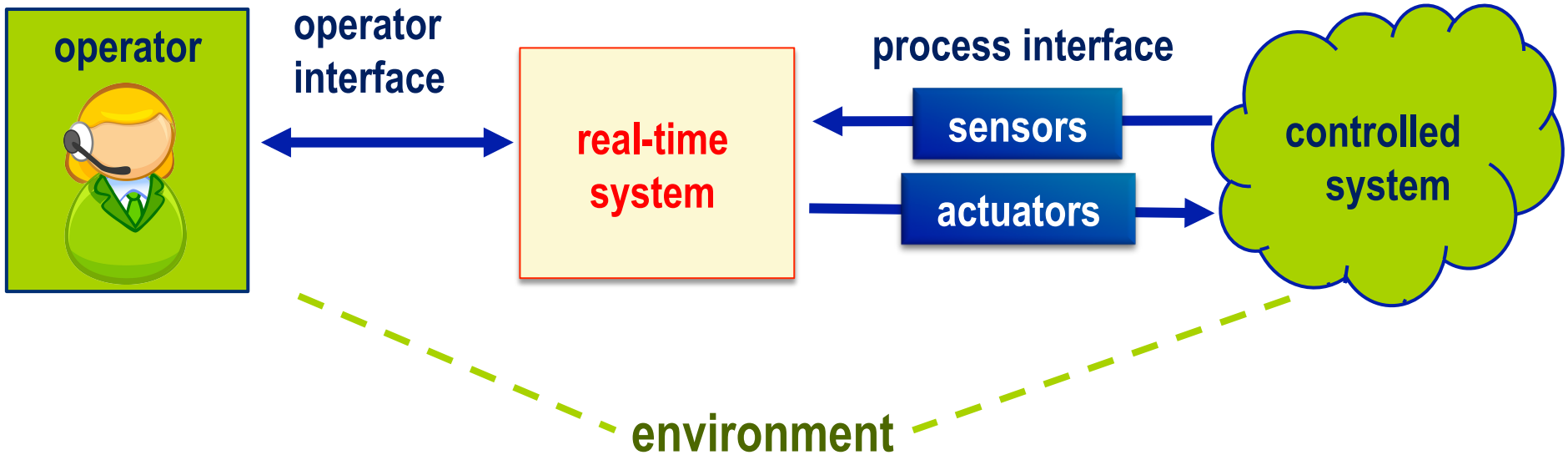
- Examples:
  - control of industrial processes, avionics, air traffic control, train control, control of automobiles, telecommunications, consumer electronics, multimedia systems.

Example: control of processes



- The real time system acts over the controlled system to achieve it has a certain behaviour.
- The behaviour is very regular (periodic sampling)
- Typically used with a feedback scheme
  - the control action is a function of the deviation between the reference values and the measured values of the variables.
  - the design of control algorithms is a major problem → studied in Control Theory

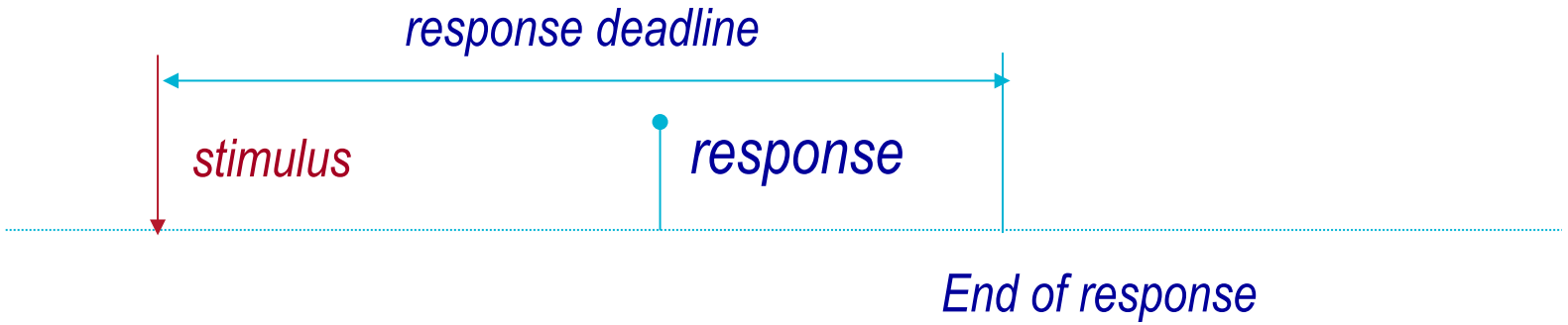
Environment of a real time system



- A real time system must respond to **stimuli** of its environment within a **time interval** determined by the dynamic properties of the environment.

Stimulus and Response

- The response interval is characterized by a **deadline**



  - **Hard**: a response out of deadline is unacceptable
    - Example: ICU (intensive cares unit), nuclear power, braking control, airbag …
  - **Firm**: a response out of deadline is useless
    - Example: in multimedia systems (audio or video processing), when there is a loss of a frame of audio or video
  - **Soft**: a response out of deadline has little use (but can still be used)
    - Example: acquisition of meteorological data, voicemail

Features of RTS

- **Concurrency**
  - Controlled system components operate simultaneously.
  - The RTS must attend them and generate control actions simultaneously

- **Interaction with physical devices**
  - RTS interact with environment through various types of **non-conventional devices** : A/D and D/A convertors, digital input/ outputs, … interfaces with sensors, actuators, special peripherals…
  - Software components that control the behaviour of these devices (i.e. drivers) are, in general, **dependent of the particular system**

Features of RTS

- **Reliability and security**
  - A fail in the controller system might provoke that the controlled system behaves in a dangerous and non-economic way.
  - It is important to ensure that if the controller system fails it must do it in a way that the controlled system remains in a safe state:
    - We must take into account these possible errors or exceptions when designing the system

- **Temporal Determinism**
  - Actions in certain time intervals
  - Essential that the temporal behaviour of the RTS is **deterministic**
    - The system must respond **correctly** in all situations
    - We must predict its behaviour in the **worst case**

Software development for Hard RTS

- The specific features of **hard real-time systems** determine the methods and tools to use for developing their software
- Not all techniques used to build other types of systems can be employed for hard real-time software
  - There are normally problems of reliability and temporal determinism

- **Predictability** is one of the main objectives in real-time systems.
  - We must do a **schedulability analysis** of tasks in a real-time system to predict whether tasks will meet their timing requirements.

Task Scheduling in RTS

- Objective: Schedule the use of system resources (specially the processor) to guarantee the timing requirements of tasks*.
- A scheduling paradigm consists of:
  - A scheduling algorithm, which determines the access order of tasks to system resources
  - An analytical method for calculating the temporal behaviour of the system
    - To verify that the requirements are guaranteed in all cases
    - We will always study the **worst case**
    - We need to know the duration of tasks in the worst case

*Task= Activity = Thread

Algorithm: Fixed-Priority Pre-emptive Scheduling

- In RTS the **Fixed-Priority Pre-emptive Scheduling** is the most popular Scheduling policy:
  - The temporal behaviour is easier to understand and predict
  - Treatment with overloads is easy to predict
  - There are complete analytical techniques
  - It is required by standards of operating systems and concurrent languages

RTS Analysis: Task model

- Let us consider initially a simple task model:
  - Static set of tasks, $\tau = \{\tau_1, \tau_2..., \tau_n\}$
  - All tasks are periodic , with **period** T_i
  - Tasks do not cooperate between them (they are independent)
  - We know the **maximum execution time** of each task C_i
  - Each task has a **deadline** $D_i \leq T_i$
- Regarding the execution of tasks, we assume that:
  - The scheduling policy is Priority Pre-emptive
  - The machine is dedicated to the application
  - Context switches have zero cost
  - Tasks cannot voluntarily suspend. Once a task is ready, it cannot delay its execution.

Activity 1: Schedule of Periodic Tasks

- Draw the execution schedule of the next set of periodic tasks:

Task	T_i	C_i	D_i	Pri *
τ_1	20	5	10	1
τ_2	40	10	15	2
τ_3	80	40	80	3

** pri(1) >pri(2)>pri(3)*

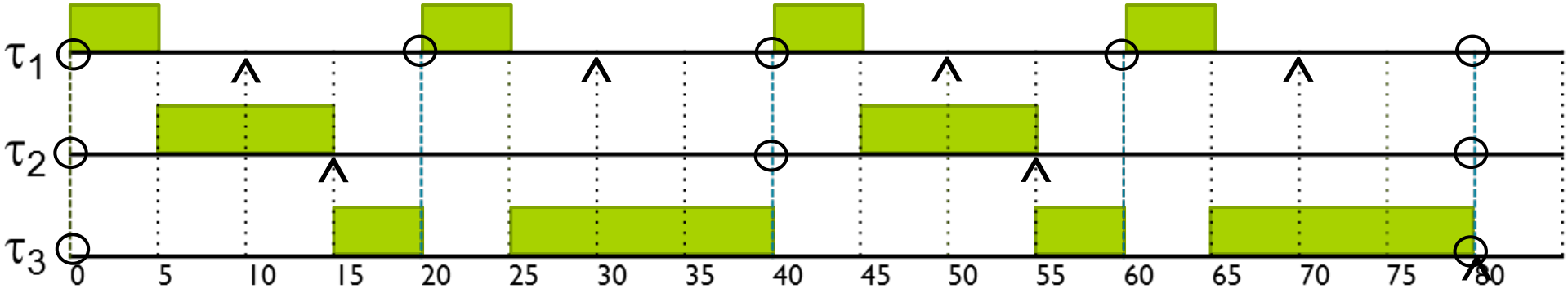
- Observe that the execution pattern repeats every certain time:
  - **hyper period**= least multiple common of the periods of the tasks

Activity 1: Schedule of Periodic Tasks

Task	T_i	C_i	D_i	Pri *
τ_1	20	5	10	1
τ_2	40	10	15	2
τ_3	80	40	80	3

○ activation instant

^ deadline



pri(1) >pri(2)>pri(3)

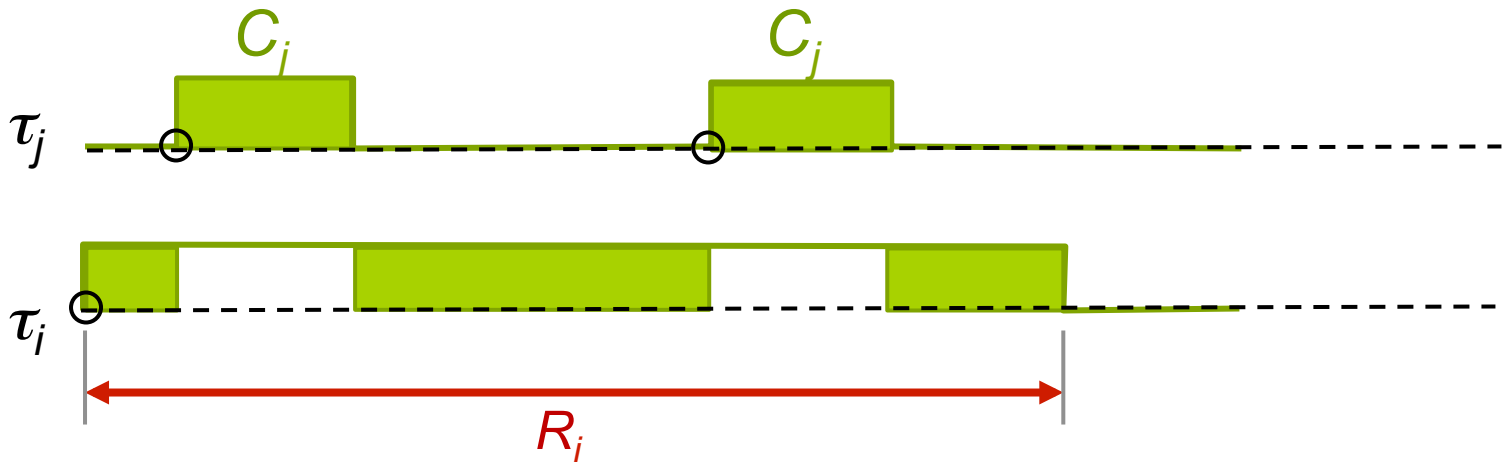
RTS Analysis: Basic Principles

- Two concepts help to build the worst case situation for independent periodic tasks:
  - Critical instant: The response time in the worst case of all tasks is obtained if we activate them all at once.
  - Just check the first deadline: After a critical instant, if a task meets its first deadline it will also meet all its subsequent deadlines.

- Based on these concepts we obtain the **schedulability test**:
  - Exact scheduling test (or of response times)

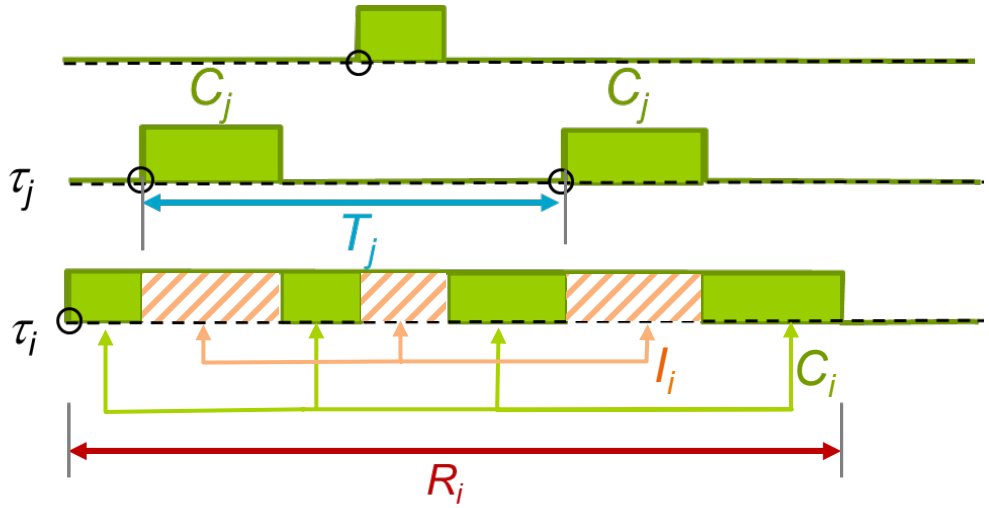
Response Time Equation

- The response time of a task is the sum of its computation time plus the interference that it suffers by the execution of more priority tasks



$$R_i = C_i + I_i$$

Calculating the maximum interference I_i



$\left\lceil \frac{R_i}{T_j} \right\rceil$: number of activations of task j in R_i

$\lceil x \rceil$: $ceiling(x)$, lower integer number ≥ x

- For a task *j* of higher priority than *i*: $I_i^j = \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$
- For all tasks of higher priority that *i*:

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$$

$hp(i)$: $high\,priority(i)$, set of tasks with higher priority than i

Schedulability test

- *In a real-time system with n independent tasks, scheduled with a Fixed-Priority pre-emptive policy, and for a specific assignment of priorities to tasks,* ***all deadlines will be met if, and only if,***

$$\forall i, \quad 1 \le i \le n, \quad R_i \le D_i$$

where:

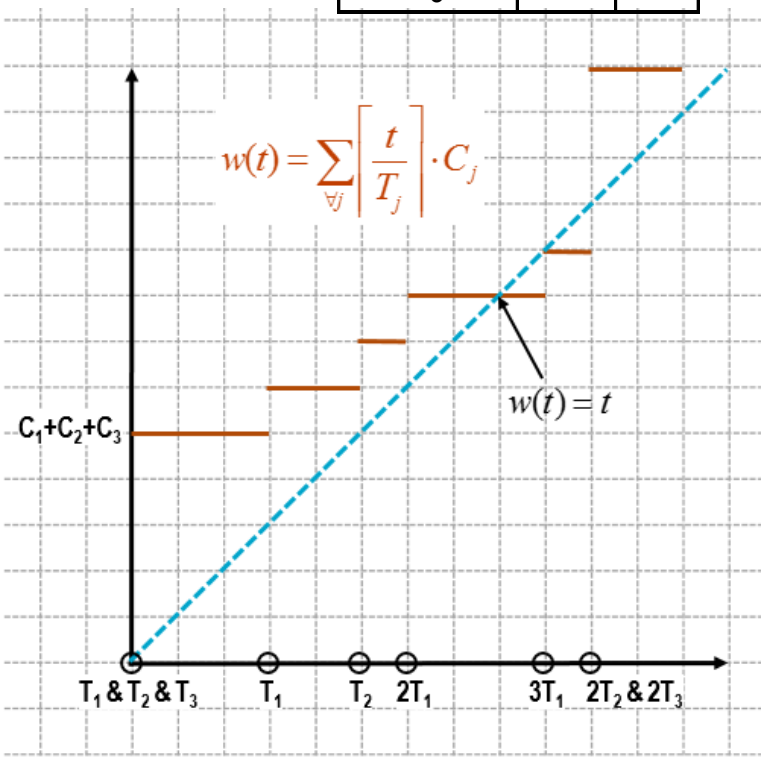
$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$$

Calculating the Response Time (I)

- R_i is the minimum solution to the equation:

$$w = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w}{T_j} \right\rceil \cdot C_j$$

Task	T_i	C_i
τ_1	3	1
τ_2	5	1
τ_3	10	3



- This equation is not continuous nor lineal
- It cannot be solved analytically

Calculating the Response Time (II)

- But it can be solved by the following recurrence relation:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil \cdot C_j$$

- An acceptable initial value is:

$$w_i^0 = C_i + \sum_{j \in hp(i)} C_j$$

- This relation ends when:
  - $w^{n+1} = w^n$ or
  - $w^{n+1} > D_i$ (the deadline is not met)

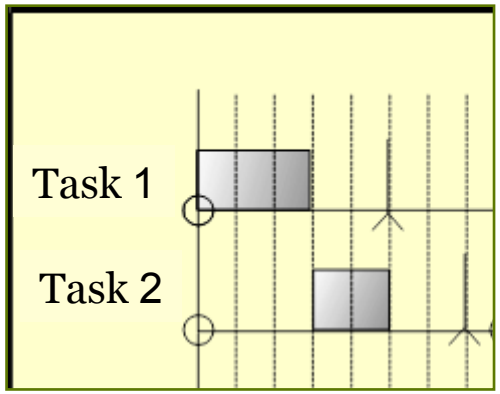
Example of calculation of Response Times (I)

- **Activity-** Calculate R_i for the next set of tasks:

Task	T_i	C_i	D_i	Prio	R_i
τ_1	12	3	5	1	
τ_2	8	2	7	2	
τ_3	20	3	16	3	
τ_4	25	4	22	4	

Example of calculation of Response Times (II)

Task	T_i	C_i	D_i	Prio	R_i
τ_1	12	3	5	1	3
τ_2	8	2	7	2	5
τ_3	20	3	16	3	
τ_4	25	4	22	4	



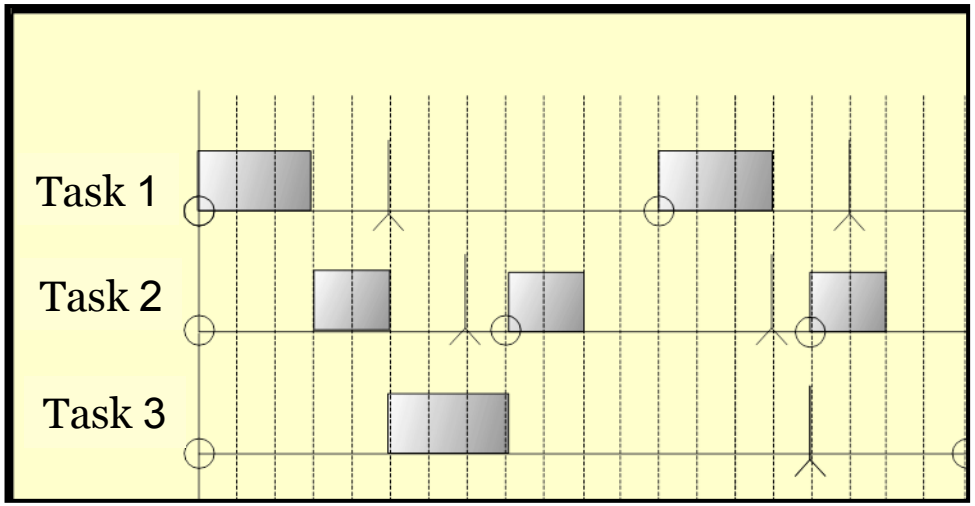
$$\tau_1 : w_1^0 = 3$$

$$\tau_2 : w_2^0 = 2 + 3 = 5$$

$$w_2^1 = 2 + \left\lceil \frac{5}{12} \right\rceil \cdot 3 = 5$$

Example of calculation of Response Times (III)

Task	T_i	C_i	D_i	Prio	R_i
τ_1	12	3	5	1	3
τ_2	8	2	7	2	5
τ_3	20	3	16	3	8
τ_4	25	4	22	4	

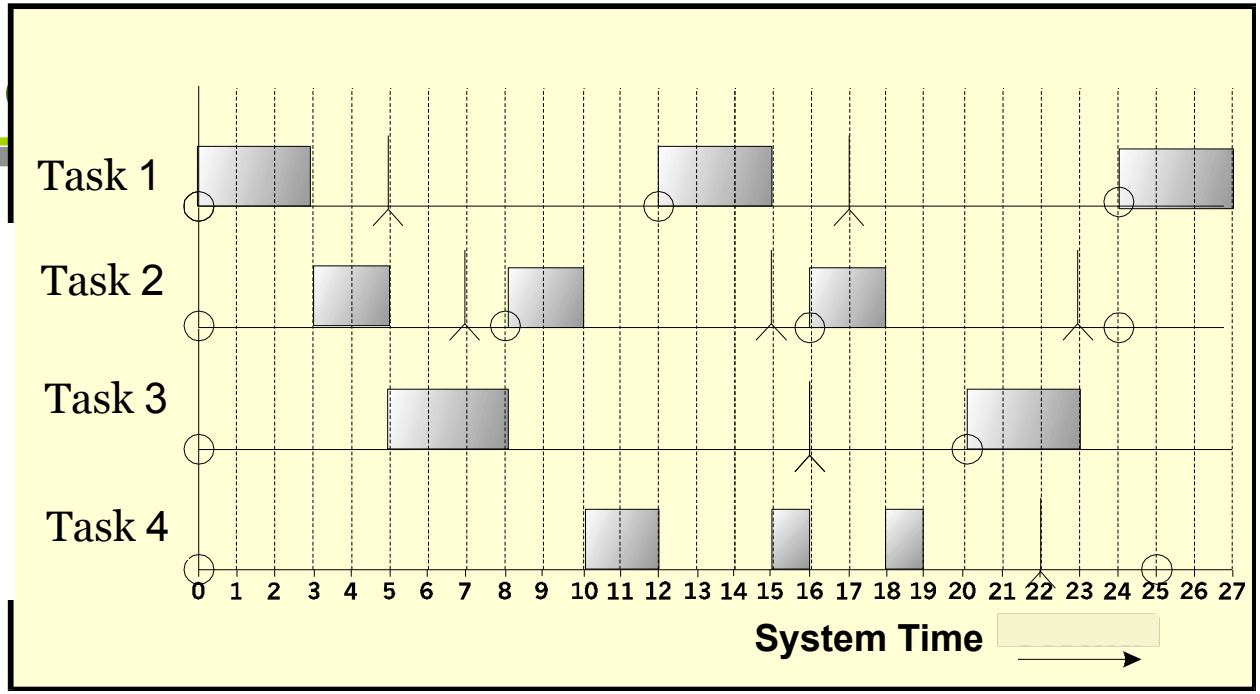


$$\tau_3 : w_3^0 = 2 + 3 + 3 = 8$$

$$w_3^1 = 3 + \left\lceil \frac{8}{12} \right\rceil \cdot 3 + \left\lceil \frac{8}{8} \right\rceil \cdot 2 = 8$$

Example of cal

Task	T_i	C_i	D_i	Prio	R_i
τ_1	12	3	5	1	3
τ_2	8	2	7	2	5
τ_3	20	3	16	3	8
τ_4	25	4	22	4	19



$$\tau_4 : w_4^0 = 2 + 3 + 3 + 4 = 12$$

$$w_4^1 = 4 + \left\lceil \frac{12}{12} \right\rceil \cdot 3 + \left\lceil \frac{12}{8} \right\rceil \cdot 2 + \left\lceil \frac{12}{20} \right\rceil \cdot 3 = 14$$

$$w_4^2 = 4 + \left\lceil \frac{14}{12} \right\rceil \cdot 3 + \left\lceil \frac{14}{8} \right\rceil \cdot 2 + \left\lceil \frac{14}{20} \right\rceil \cdot 3 = 17$$

$$w_4^3 = 4 + \left\lceil \frac{17}{12} \right\rceil \cdot 3 + \left\lceil \frac{17}{8} \right\rceil \cdot 2 + \left\lceil \frac{17}{20} \right\rceil \cdot 3 = 19$$

$$w_4^4 = 4 + \left\lceil \frac{19}{12} \right\rceil \cdot 3 + \left\lceil \frac{19}{8} \right\rceil \cdot 2 + \left\lceil \frac{19}{20} \right\rceil \cdot 3 = 19$$

Activity 2: Schedulability Test

- Determine the schedulability of the next set of tasks, using the Response Time test, in the following cases:
  a) The priority assignment is: $\tau_1 > \tau_2 > \tau_3$
  b) The priority assignment is: $\tau_1 < \tau_2 < \tau_3$

Task	T_i	C_i	D_i
τ_1	4	1	4
τ_2	5	2	5
τ_3	20	3	10

Solution case a)

Task	T_i	C_i	D_i
τ_1	4	1	4
τ_2	5	2	5
τ_3	20	3	10

$$\tau_1 : w_1^0 = 1$$

$$\tau_2 : w_2^0 = 2 + 1 = 3$$

$$w_2^1 = 2 + \left\lceil \frac{3}{4} \right\rceil \cdot 1 = 3$$

$$\tau_3 : w_3^0 = 3 + 1 + 2 = 6$$

$$w_3^1 = 3 + \left\lceil \frac{6}{4} \right\rceil \cdot 1 + \left\lceil \frac{6}{5} \right\rceil \cdot 2 = 9$$

$$w_3^2 = 3 + \left\lceil \frac{9}{4} \right\rceil \cdot 1 + \left\lceil \frac{9}{5} \right\rceil \cdot 2 = 10$$

$$w_3^3 = 3 + \left\lceil \frac{10}{4} \right\rceil \cdot 1 + \left\lceil \frac{10}{5} \right\rceil \cdot 2 = 10$$

$R_1 = 1 \quad < D_1$
$R_2 = 3 \quad < D_2$
$R_3 = 10 \quad \leq D_3$

All deadlines are guaranteed

Content

- Introduction to RTS
  - Definition of a Real-Time System (RTS)
  - Features of RTS
  - Task Scheduling in RTS
  - Analysis of RTS
- Task Synchronization
  - The Priority Inversion Problem
  - The Immediate Priority Ceiling Protocol
  - Response time with blocking factors

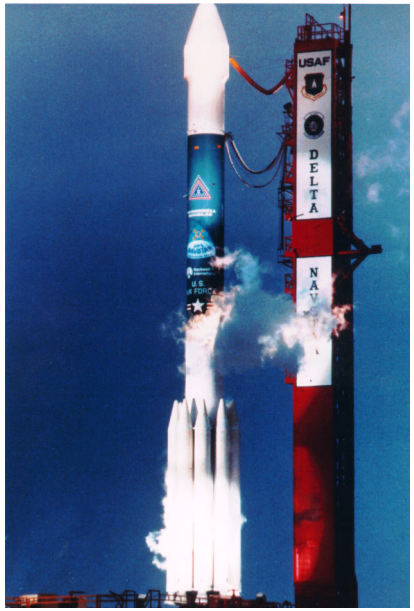
Task Synchronization

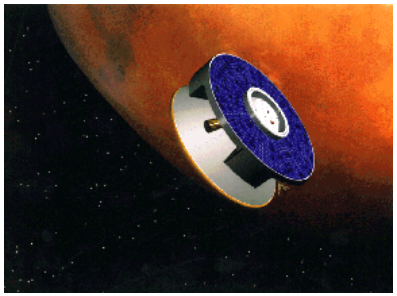
- One of the most important restrictions of the previous scheduling test is to assume that all tasks are *independent*.
- In most practical systems, tasks need to communicate between them.
  - When communication is performed by sharing common data, then *synchronization* is necessary to avoid race conditions.
  - We will assume that tasks only need *mutual exclusion* in the access to their critical sections, and that they protect them using a synchronization primitive equivalent to locks:
    - P(S) is equivalent to **close_lock**(S);
    - V(S) is equivalent to **open_lock**(S);

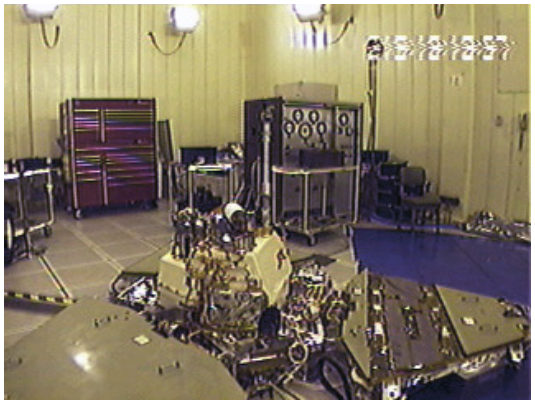
Task Synchronization

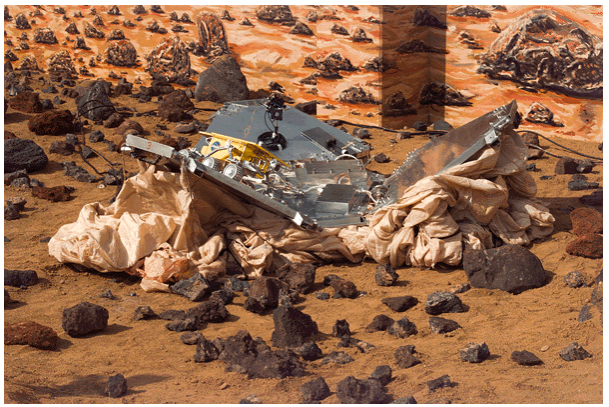
- However, in real-time systems this solution is not valid because it can cause a situation called priority inversion, which invalidates the schedulability test.

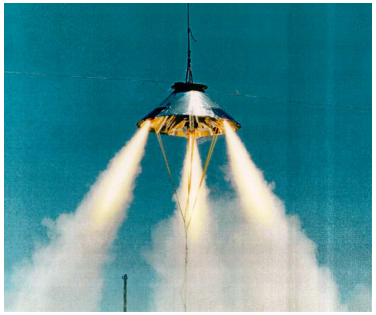
Mars Pathfinder mission, July 1997

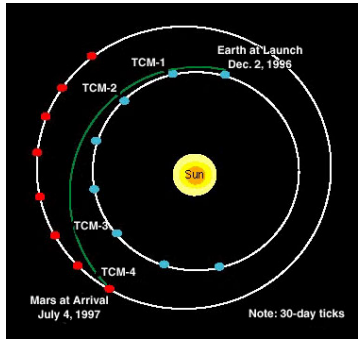


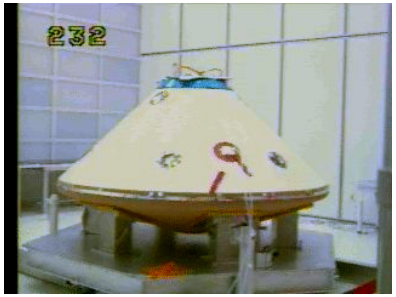






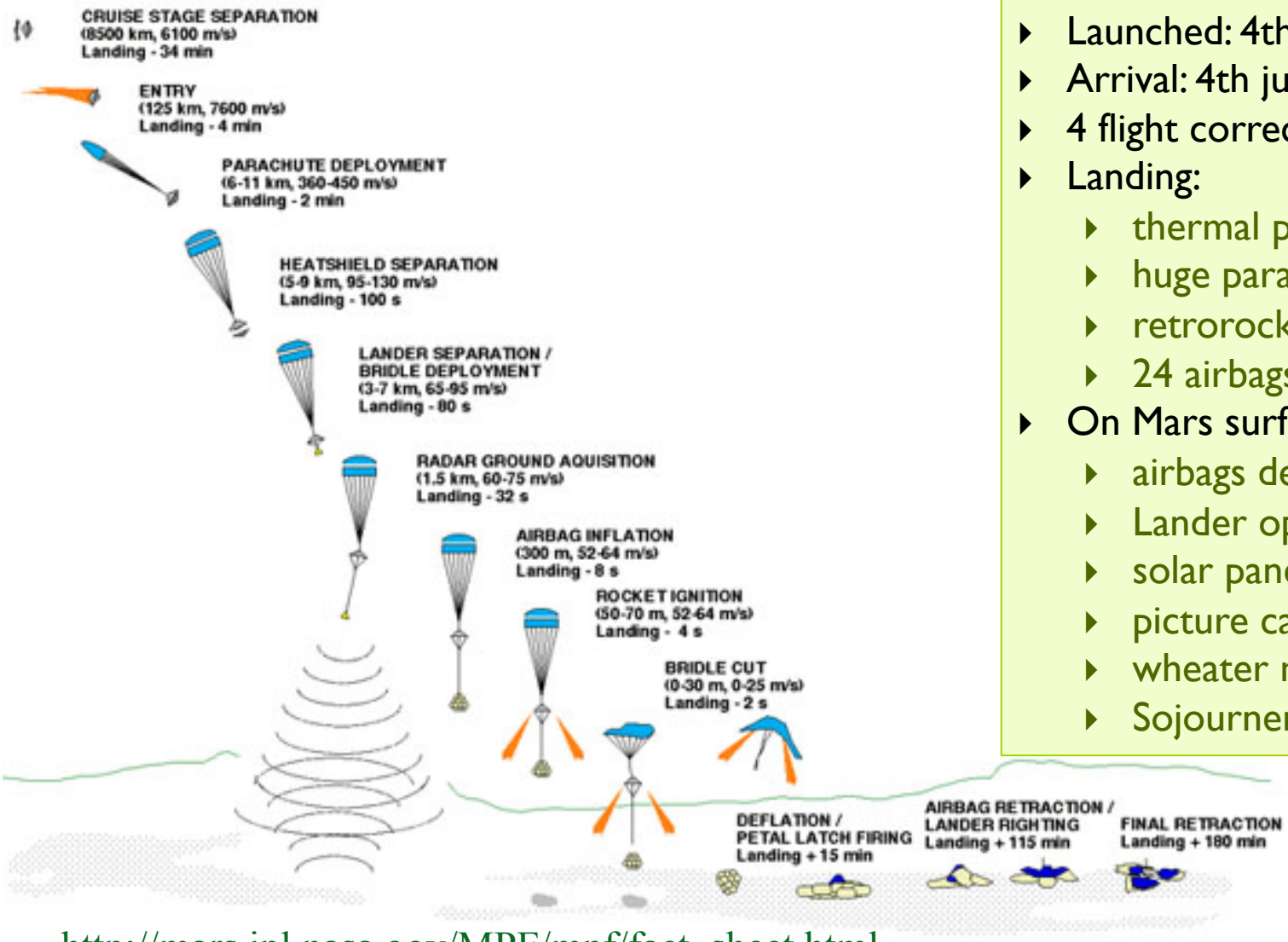






http://mars.jpl.nasa.gov/MPF/index0.html

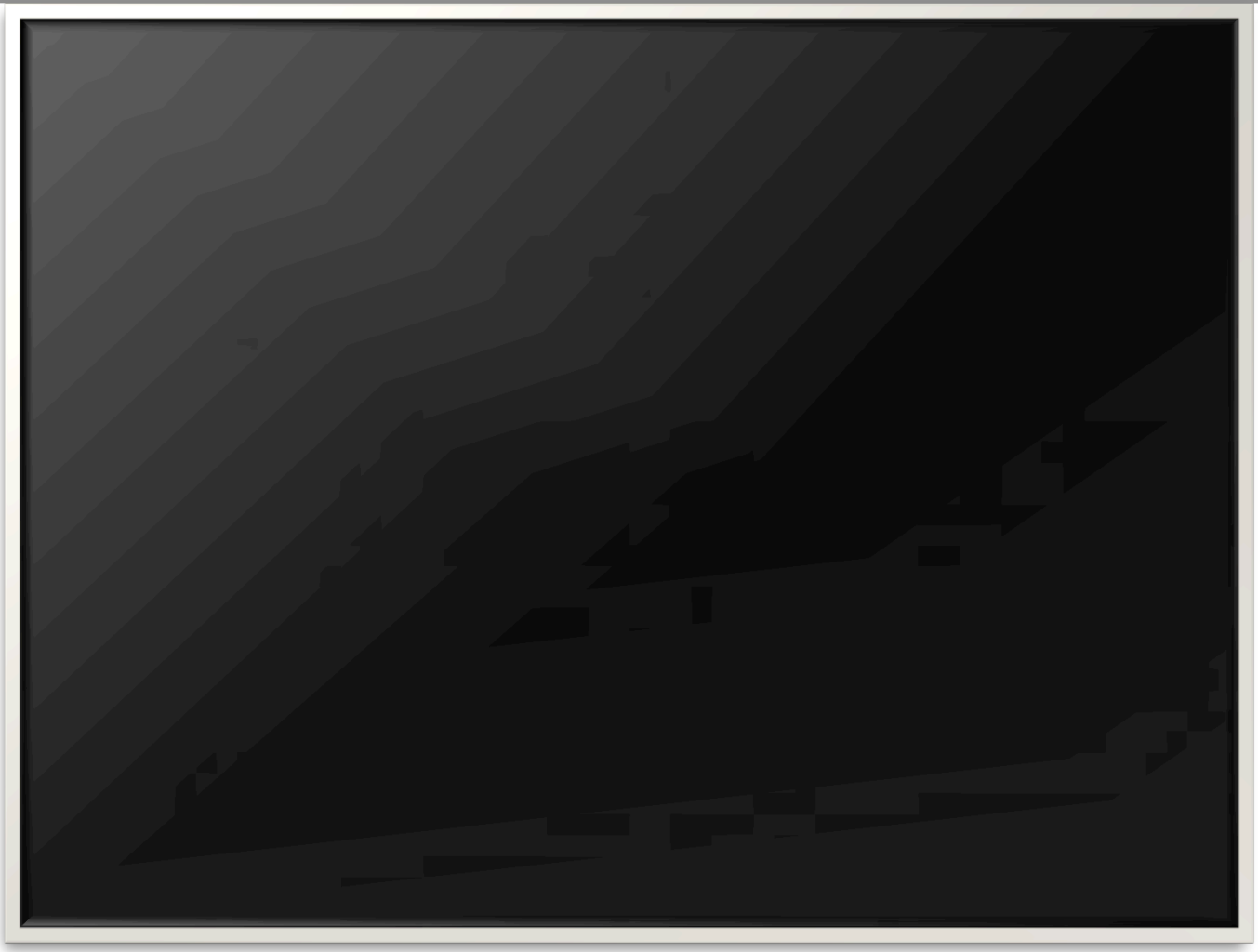
Mars Pathfinder



- Launched: 4th december 1996
- Arrival: 4th july 1997
- 4 flight corrections
- Landing:
  - thermal protection shield
  - huge parachute
  - retrorockets
  - 24 airbags
- On Mars surface:
  - airbags deflating
  - Lander opening
  - solar panels unfolding
  - picture capturing
  - wheater measurements
  - Sojourner descending ramp

http://mars.jpl.nasa.gov/MPF/mpf/fact_sheet.html

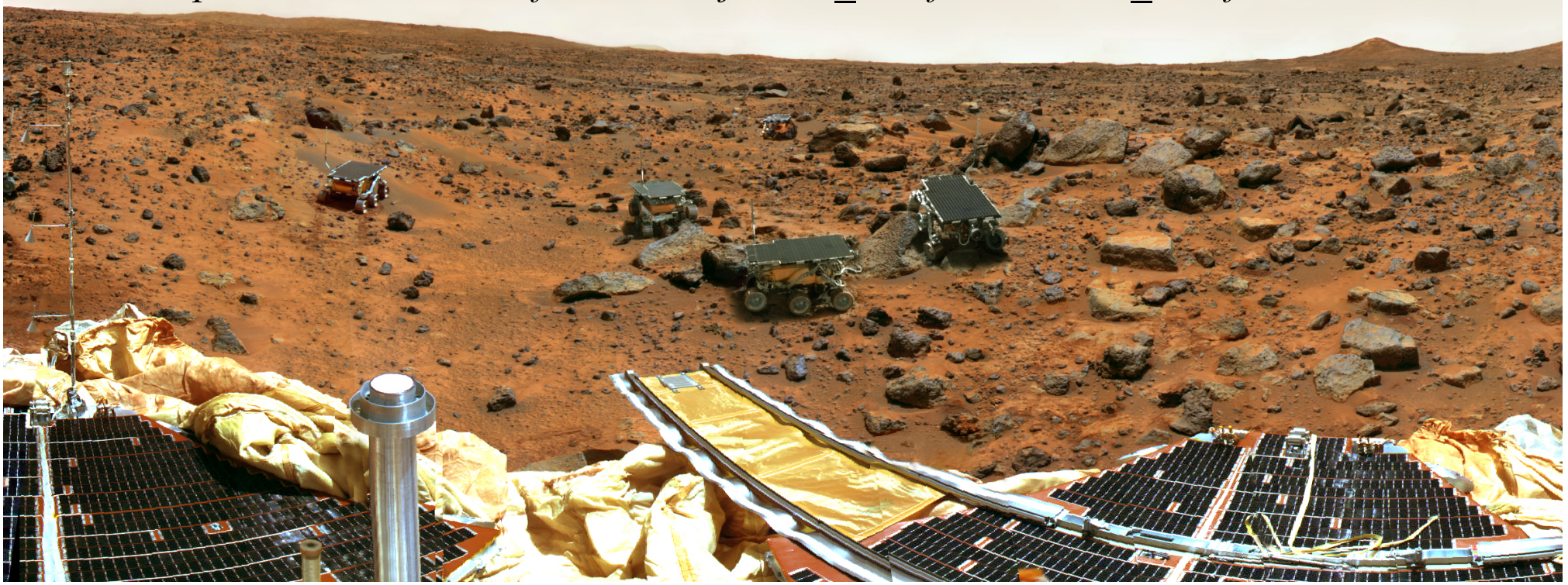
Motivación: Tribute to the Mars Pathfinder Team



https://www.youtube.com/watch?v=6jspO2CsFNA

Incident in the "Mars Pathfinder"

http://research.microsoft.com/~mbj/Mars_Pathfinder/Mars_Pathfinder.html



The Mars Pathfinder mission was widely proclaimed as "flawless" in the early days after its July 4th, 1997 landing on the Martian surface.

But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total **system resets**, each resulting in losses of data. The press reported these failures in terms such as "**software glitches**" and "the computer was trying to do too many things at once".

Incident in the "Mars Pathfinder"

http://research.microsoft.com/~mbj/Mars_Pathfinder/Mars_Pathfinder.html

The New York Times

www.nytimes.com

July 15, 1997

Mars Craft Again Halts Transmission

PASADENA, Calif., July 14— The computer aboard the Mars Pathfinder overloaded and reset itself early today for the second time in just over three days, interrupting the transmission of a full-color panoramic scene.

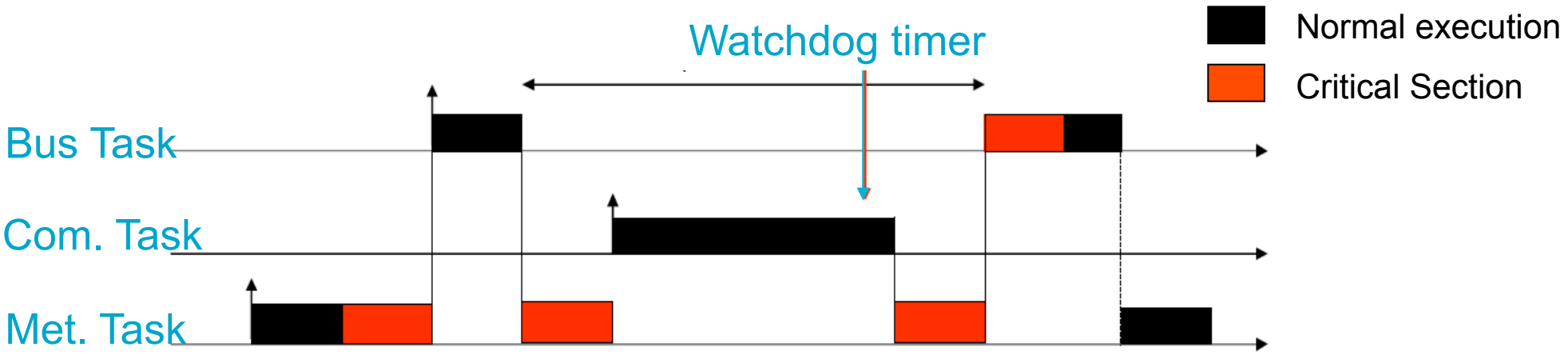
The mishap delayed chemical analysis of a tubby rock named Yogi, but no information was lost, and controllers will be able to resume transmission where it was left off, officials said.

Mary Beth Murrill, a spokeswoman for NASA's Jet Propulsion Laboratory, said transmission of the panoramic shot took "a lot of processing power." She likened the data overload to what happens with a personal computer "when we ask it to do too many things at once."

The project manager, Brian Muirhead, said that to prevent a recurrence, controllers would schedule activities one after another, instead of at the same time. It was the second time the Pathfinder's computer had reset itself while trying to carry out several activities at once.

data, the spacecraft began experiencing total **system resets**, each resulting in losses of data. The press reported these failures in terms such as "**software glitches**" and "the computer was trying to do too many things at once".

Incident in the "Mars Pathfinder"



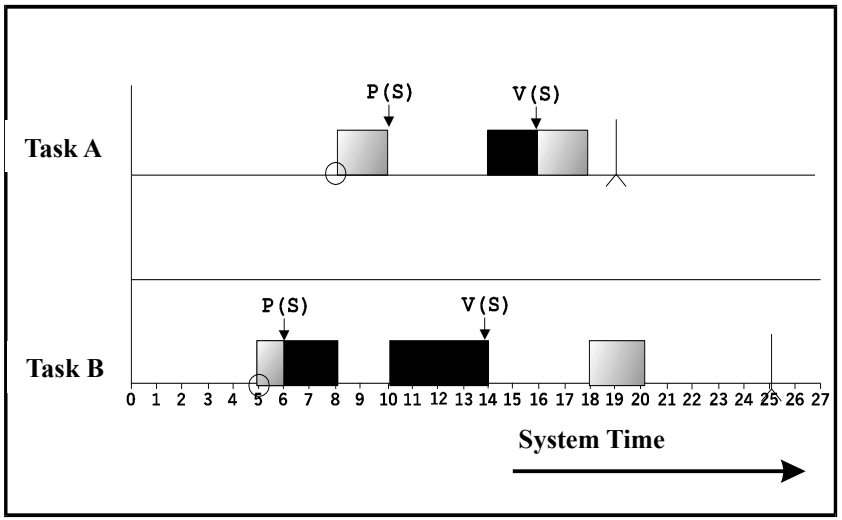
" …. Most of the time this combination worked fine. However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running.

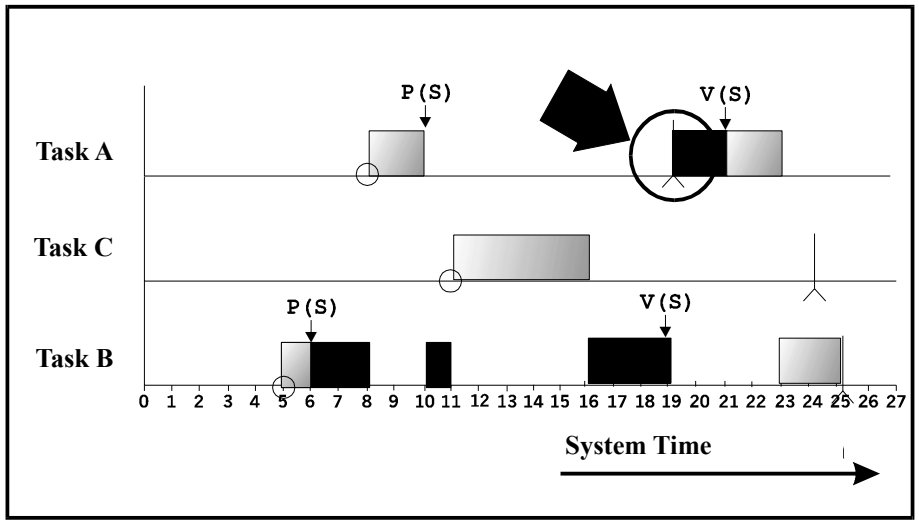
After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset.

This scenario is a classic case of **priority inversion**. "

Priority Inversion

- Priority inversion occurs when a lower priority task is running before a higher priority task.
- The total time of blocking due to a lower priority task is not bounded, since intermediate priority tasks can also be executed meanwhile.
- Example :





Synchronization Protocols

- The priority inversion cannot be avoided. Blockings (or deadlocks) compromise the schedulability of the system.
  - We need to achieve that the blocking is bounded, is produced the least number of times as possible and is measurable.
- The protocols for real-time synchronization avoid unbounded priority inversion.
- The most important ones are:
  - Priority Inheritance
  - Immediate Priority Ceiling (or protection by priority)
- Both cause that the priority inversion, or the delay that a task suffers due to lower priority tasks:
  - is based on the duration of one or more critical sections
  - and not on the duration of complete tasks

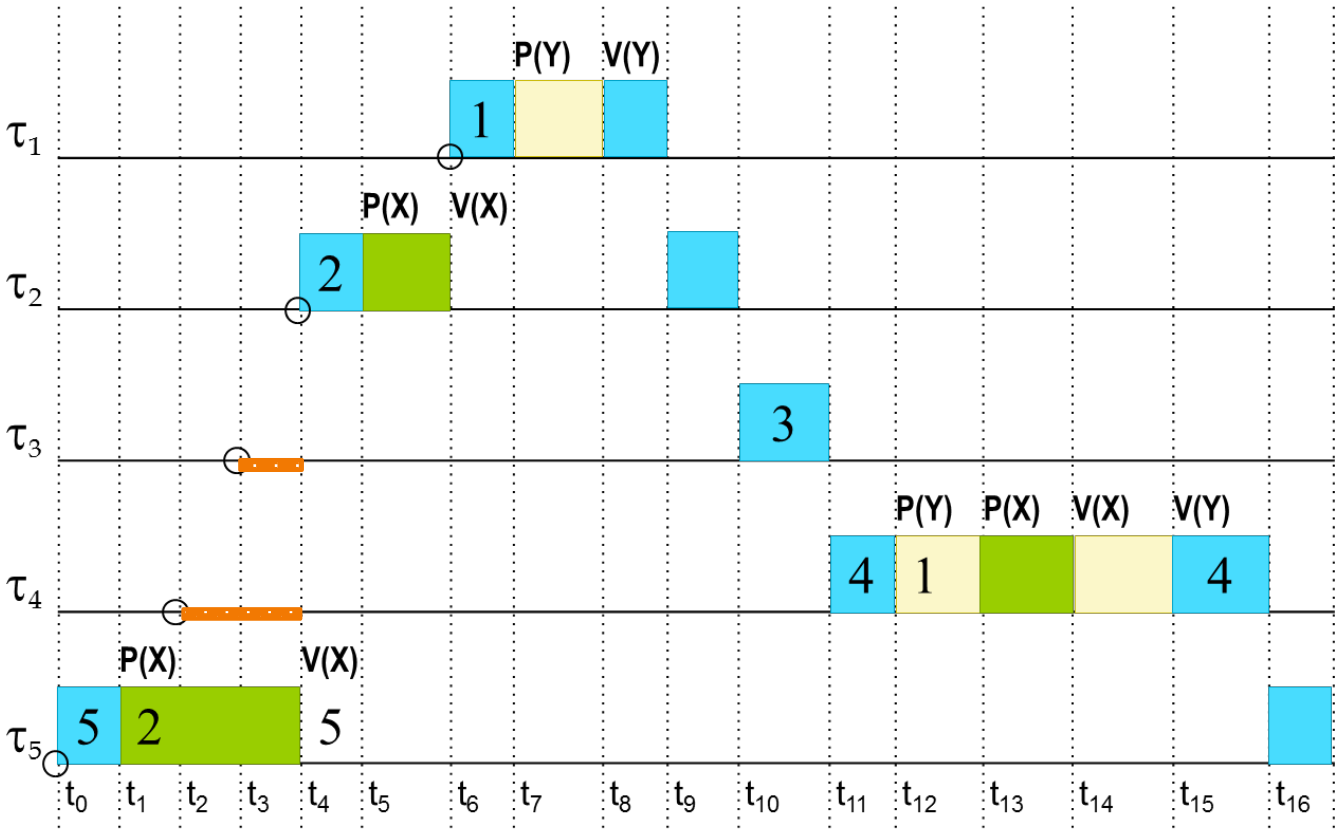
Content

- Introduction to RTS
  - Definition of a Real-Time System (RTS)
  - Features of RTS
  - Task Scheduling in RTS
  - Analysis of RTS
- Task Synchronization
  - The Priority Inversion Problem
  - The Immediate Priority Ceiling Protocol
  - Response time with blocking factors

Immediate Priority Ceiling Protocol

- Priority ceiling concept: each semaphore will be assigned a ceiling, equivalent to the highest priority of tasks that can use this semaphore.
- With this protocol, a task that accesses a semaphore immediately inherits the priority ceiling of the semaphore
  - The critical section is executed with the priority of the ceiling of the semaphore that protects it.
- Properties:
  - Each task can be blocked at most once in each cycle. Moreover, if a task is blocked, it does it at the beginning of the cycle.
  - There cannot be deadlocks.
- This protocol is also known by other names:
  - priority ceiling emulation (RT Java)
  - priority protect protocol (POSIX)
  - ceiling locking protocol (Ada)

Immediate Priority Ceiling Protocol



ceiling(X) = 2
ceiling(Y) = 1

i Execution in non-critical section, at priority *i*

Execution in a critical section, protected by semaphore X

Execution in a critical section, protected by semaphore Y

Indirect blocking

Calculation of maximum blocking factors

- The blocking factor B_i is the maximum time of delay that a task i suffers due to all other tasks of lower priority
- For the immediate priority ceiling protocol:
  - B_i is the maximum duration of all critical sections whose ceiling is bigger or equal (>=) than the priority of task i.
  - For the task with lowest priority: $B_n = 0$
  - For the rest of tasks:

$$B_i = \max_{\{k,s \mid k \in lp(i) \wedge s \in uses(k) \wedge ceiling(s) \geq prio(i)\}} C_{k,s}$$

    - We have to consider tasks *k* of priority lower than *i* (*k in lp(i)*) , and then we look at semaphores *s* that they can close (*s in uses(k)*), we select those whose ceiling has priority equal or bigger than i (*ceiling(s)>=prio(i)*), and finally we look at the computing times of the sections protected by those semaphores $C_{k,s}$ and we select the maximum duration.

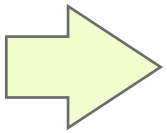
Incorporating the blocking factor to the Schedulability Test

- The **initial test** based on fixed priorities employed a recurrence relation as follows:

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

And we need to check if for each task holds $R_i \leq D_i$.

- This test is based on the worst case of interference that higher priority tasks have on the task i.



We must also include the **blocking effect** of the lower priority tasks

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

Activity 3 – Schedulability Test with blocking factors

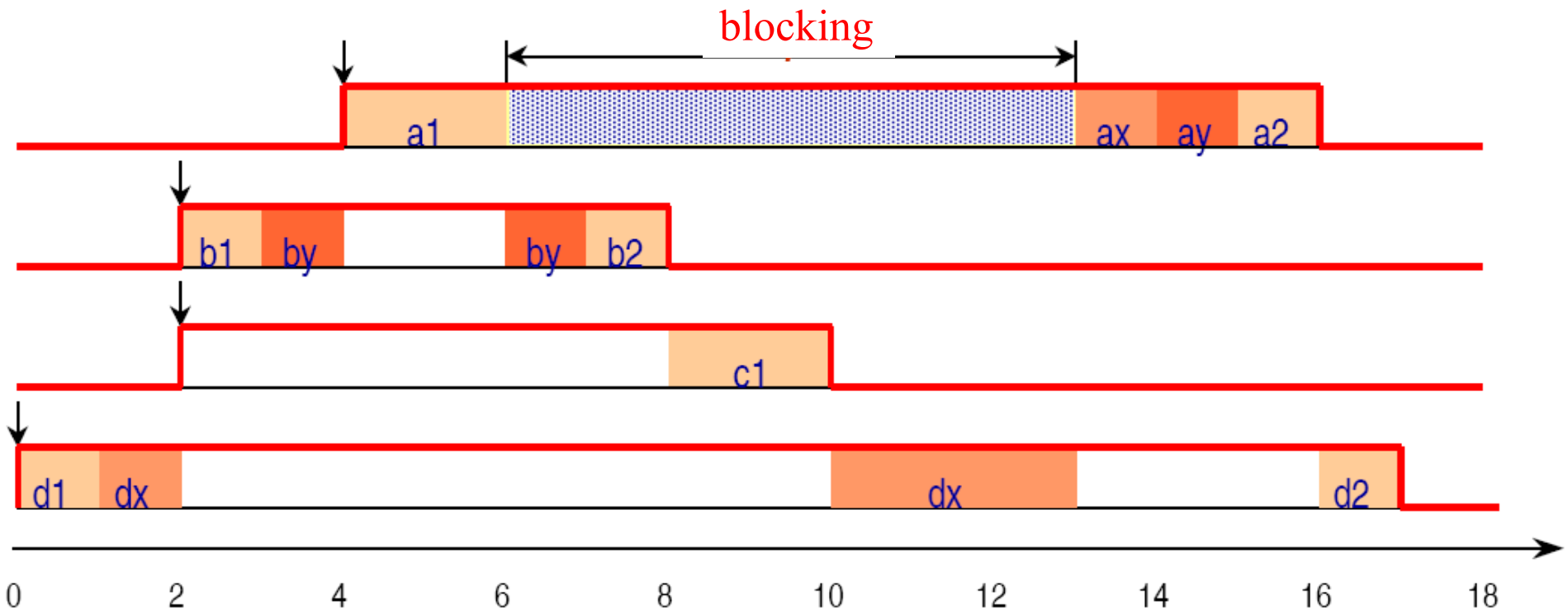
- Consider a system with 4 tasks and 2 semaphores that protect several critical sections as explained in the next tables:

Task	Actions	Activation Instant
τ_1	a1; ax; ay; a2	4
τ_2	b1; by; b2	2
τ_3	c1	2
τ_4	d1; dx; d2	0

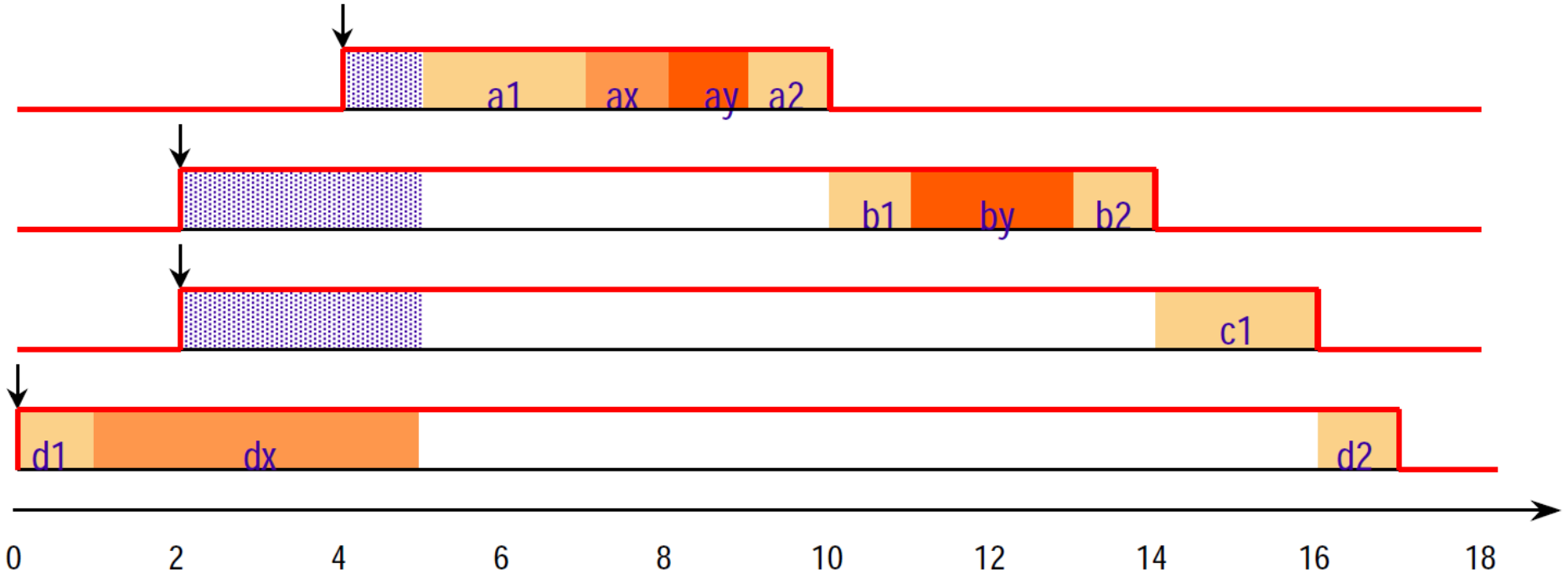
Action	C	uses semaphore
a1	2	
ax	1	X
ay	1	Y
a2	1	
b1	1	
by	2	Y
b2	1	
c1	2	
d1	1	
dx	4	X
d2	1	

- Draw the schedule for the following cases:
  a) Fixed-Priority Pre-emptive Policy with prio(1) >prio(2)>prio(3)>prio(4), and normal semaphores
  b) Semaphores that employ the immediate priority ceiling protocol
- Calculate in case b), for each task its maximum blocking factor

Solution: a) Priority Inversion



Solution: b) Immediate Priority Ceiling



Solution: calculation of blocking factors

- For the immediate priority ceiling protocol:
  - for the lowest priority task: $B_n = 0$
  - for the rest of tasks:

$$B_i = \max_{\{k,s \mid k \in lp(i) \wedge s \in uses(k) \wedge ceiling(s) \geq prio(i)\}} C_{k,s}$$

  - We have to consider tasks *k* of priority lower than *i* (*k in lp(i)*) , and then we look at semaphores *s* that they can close (*s in uses(k)*), we select those whose ceiling has priority equal of bigger than i (*ceiling(s)>=prio(i)*), and finally we look at the computing times of the sections protected by those semaphores $C_{k,s}$ and we select the maximum duration.

Action	C	uses semaphore
a1	2	
ax	1	X
ay	1	Y
a2	1	
b1	1	
by	2	Y
b2	1	
c1	2	
d1	1	
dx	4	X
d2	1	

Task	B_i with Immediate Priority Ceiling
τ_1	4
τ_2	4
τ_3	4
τ_4	0

Learning results of this Teaching Unit

- At the end of this unit, the student should be able to:
  - Adequately characterize real-time systems.
  - Identify the difficulties of scheduling tasks in real-time systems.
  - Apply the schedulability test on different assumptions.
  - Identify the priority inversion problem and the consequences of this problem.
  - Implement the Immediate Priority Ceiling Protocol on systems that require it.