Embedded Systems

Real-Time Scheduling

Dr. Anas A. Youssef
anas.youssef@ci.menofia.edu.eg

Operating Systems: Internals and Design Principles

Chapter 10 Multiprocessor, Multicore and Real-Time Scheduling

Eighth Edition
By William Stallings

Real-Time Systems

- The operating system, and in particular the scheduler, is perhaps the most important component

Examples:

- control of laboratory experiments
- process control in industrial plants
- robotics
- air traffic control
- telecommunications
- military command and control systems

- Correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced
- Tasks or processes attempt to control or react to events that take place in the outside world
- These events occur in "real time" and tasks must be able to keep up with them

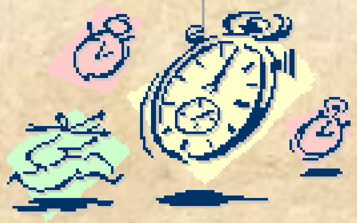
Hard and Soft Real-Time Tasks

Hard real-time task

- one that must meet its deadline
- otherwise it will cause unacceptable damage or a fatal error to the system

Soft real-time task

- has an associated deadline that is desirable but not mandatory
- it still makes sense to schedule and complete the task even if it has passed its deadline



Periodic and Aperiodic Tasks

- **Periodic tasks**
  - requirement may be stated as:
    - once per period T
    - exactly T units apart
- **Aperiodic tasks**
  - has a deadline by which it must finish or start
  - may have a constraint on both start and finish time

Characteristics of Real Time Systems

Real-time operating systems have requirements in five general areas:

- Determinism
- Responsiveness
- User control
- Reliability
- Fail-soft operation

Determinism

- Concerned with how long an operating system delays before acknowledging an interrupt
- Operations are performed at fixed, predetermined times or within predetermined time intervals
    - when multiple processes are competing for resources and processor time, no system will be fully deterministic

The extent to which an operating system can deterministically satisfy requests depends on:

the speed with which it can respond to interrupts

whether the system has sufficient capacity to handle all requests within the required time

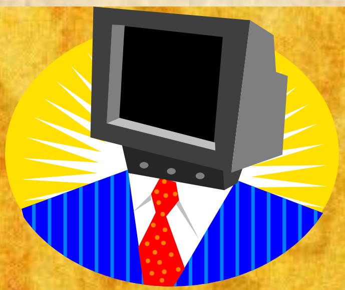
Responsiveness

- Together with determinism make up the response time to external events
  - critical for real-time systems that must meet timing requirements imposed by individuals, devices, and data flows external to the system
- Concerned with how long, after acknowledgment, it takes an operating system to service the interrupt

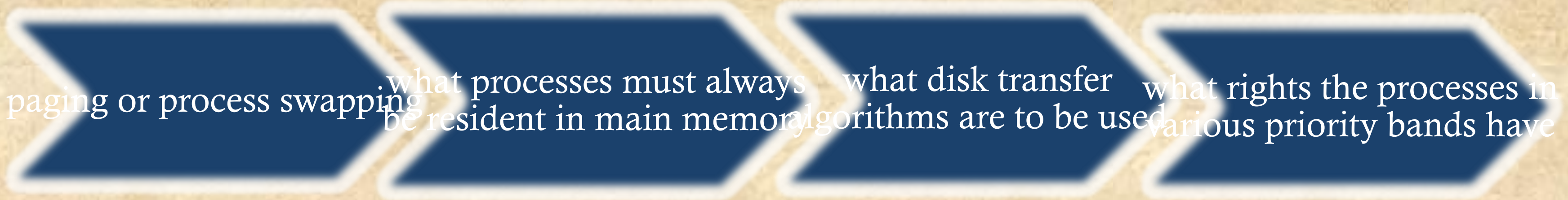
Responsiveness includes:

- amount of time required to initially handle the interrupt and begin execution of the interrupt service routine (ISR)
- amount of time required to perform the ISR
- effect of interrupt nesting

User Control

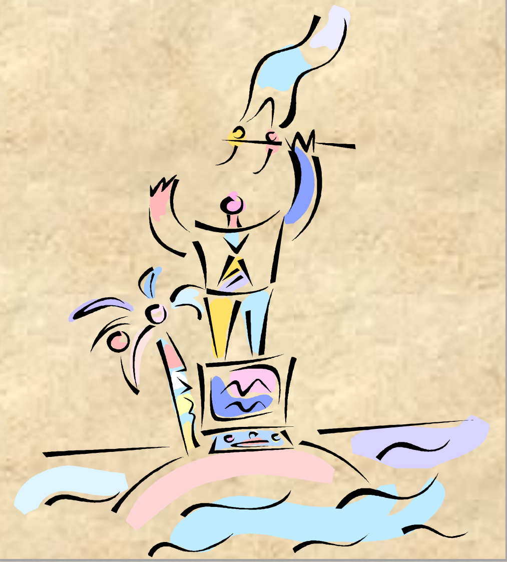


- Generally much broader in a real-time operating system than in ordinary operating systems
- It is essential to allow the user fine-grained control over task priority
- User should be able to distinguish between hard and soft tasks and to specify relative priorities within each class
- May allow user to specify such characteristics as:


paging or process swapping
what processes must always be resident in main memory
what disk transfer algorithms are to be used
what rights the processes in various priority bands have

Reliability

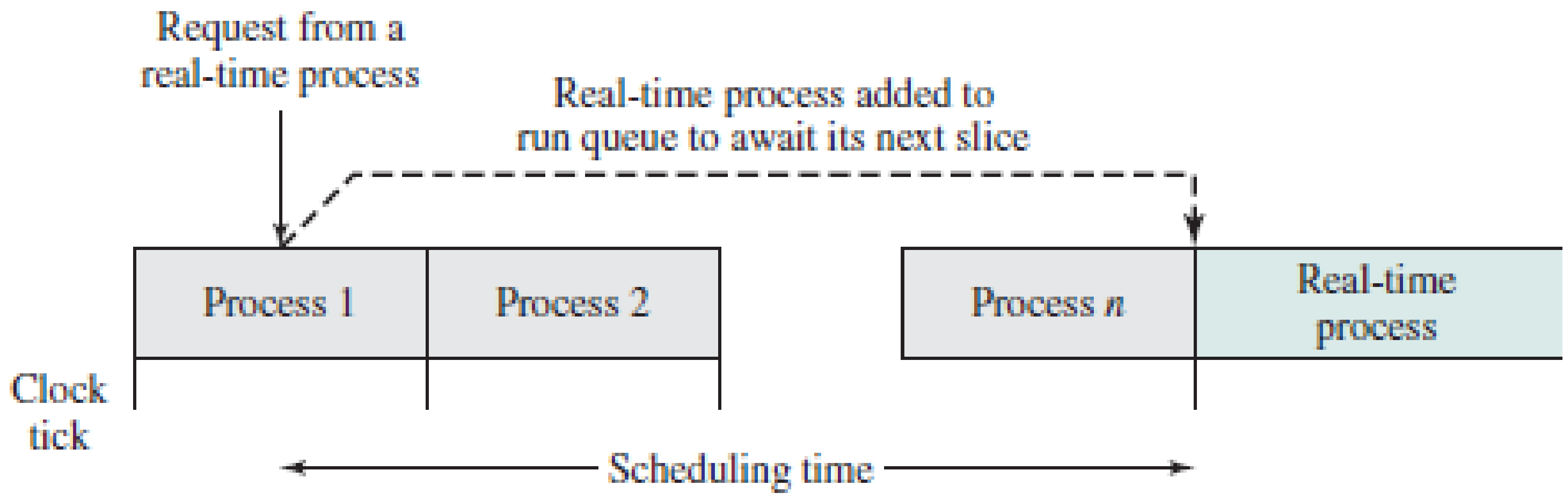
- More important for real-time systems than non-real time systems
- Real-time systems respond to and control events in real time so loss or degradation of performance may have catastrophic consequences such as:
  - financial loss
  - major equipment damage
  - loss of life



Fail-Soft Operation

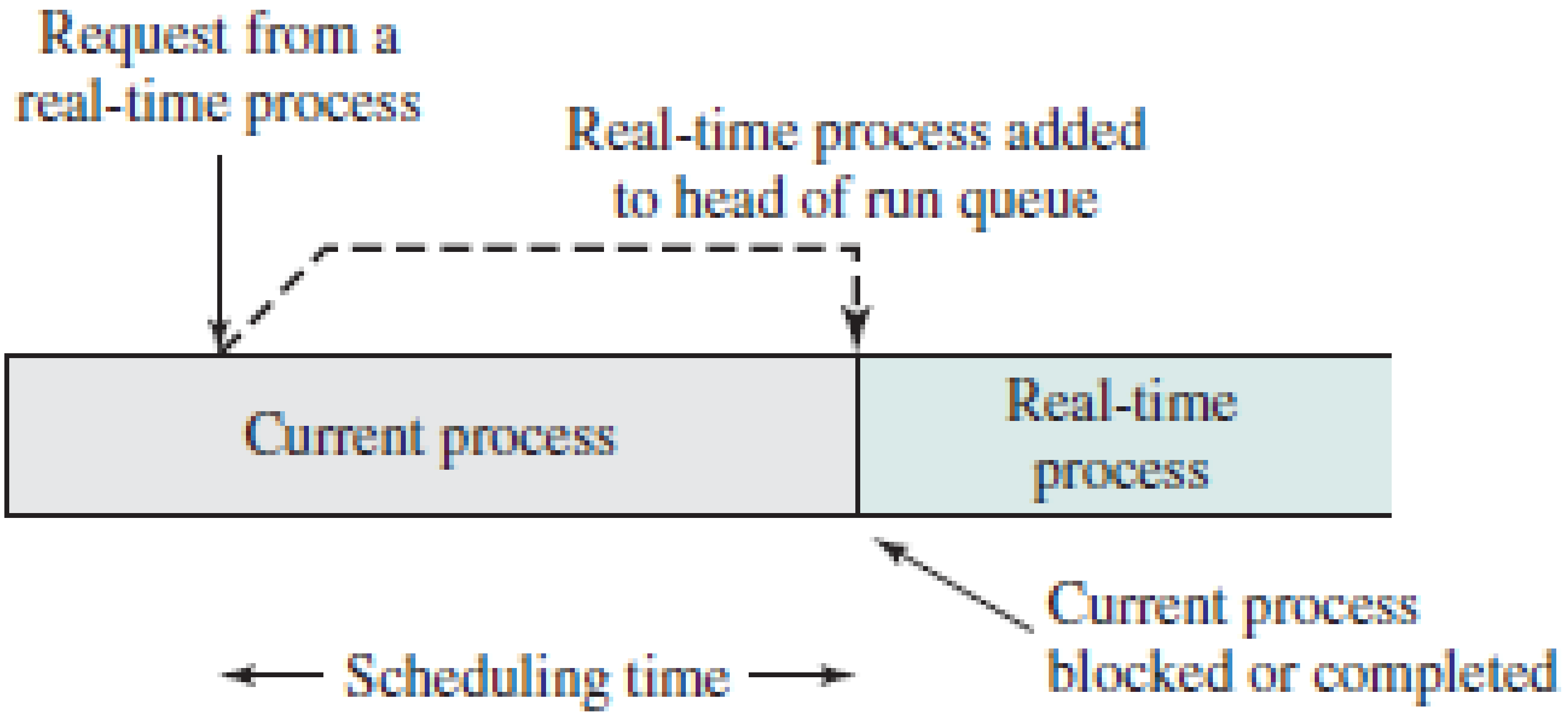
- A characteristic that refers to the ability of a system to fail in such a way as to preserve as much capability and data as possible
- Important aspect is stability
  - a real-time system is stable if the system will meet the deadlines of its most critical, highest-priority tasks even if some less critical task deadlines are not always met

Scheduling of Real-Time Process


Request from a real-time process
Real-time process added to run queue to await its next slice
Process 1
Process 2
Process *n*
Real-time process
Clock tick
Scheduling time

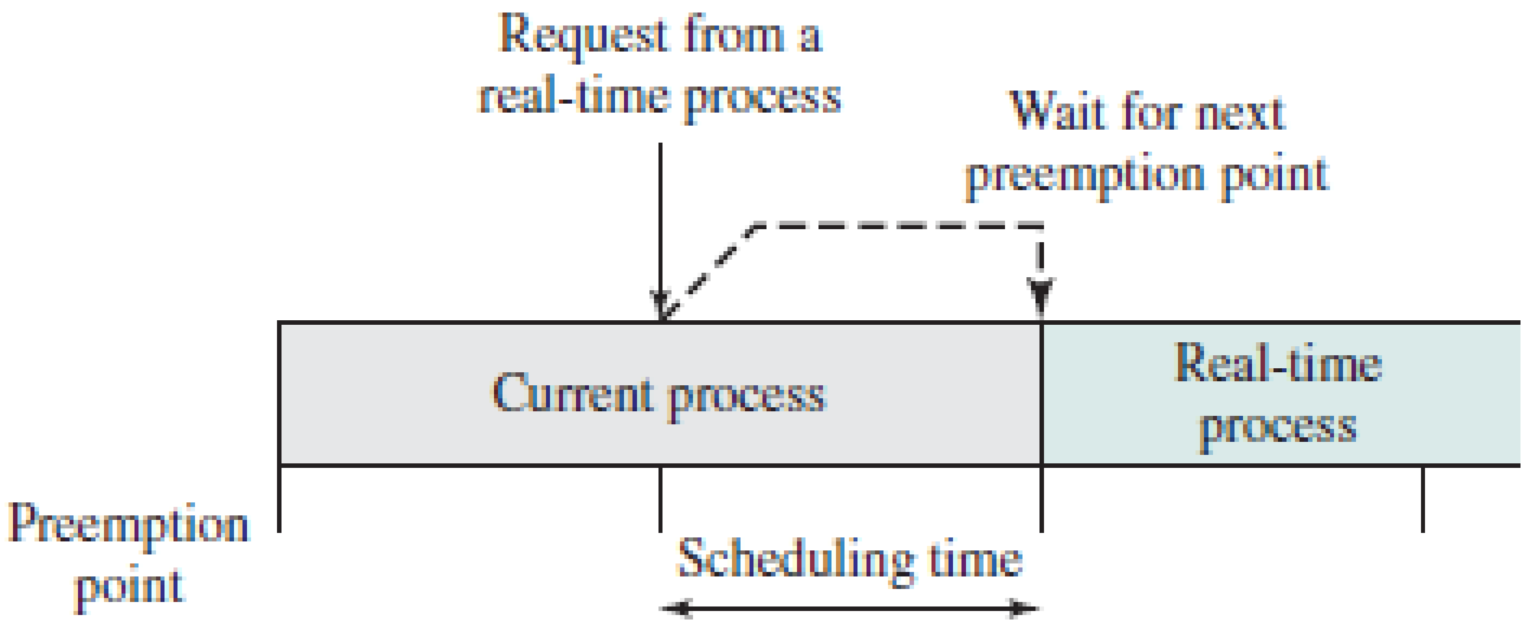
(a) Round-robin preemptive scheduler

Scheduling of Real-Time Process


Request from a real-time process
Real-time process added to head of run queue
Current process
Real-time process
Current process blocked or completed
Scheduling time

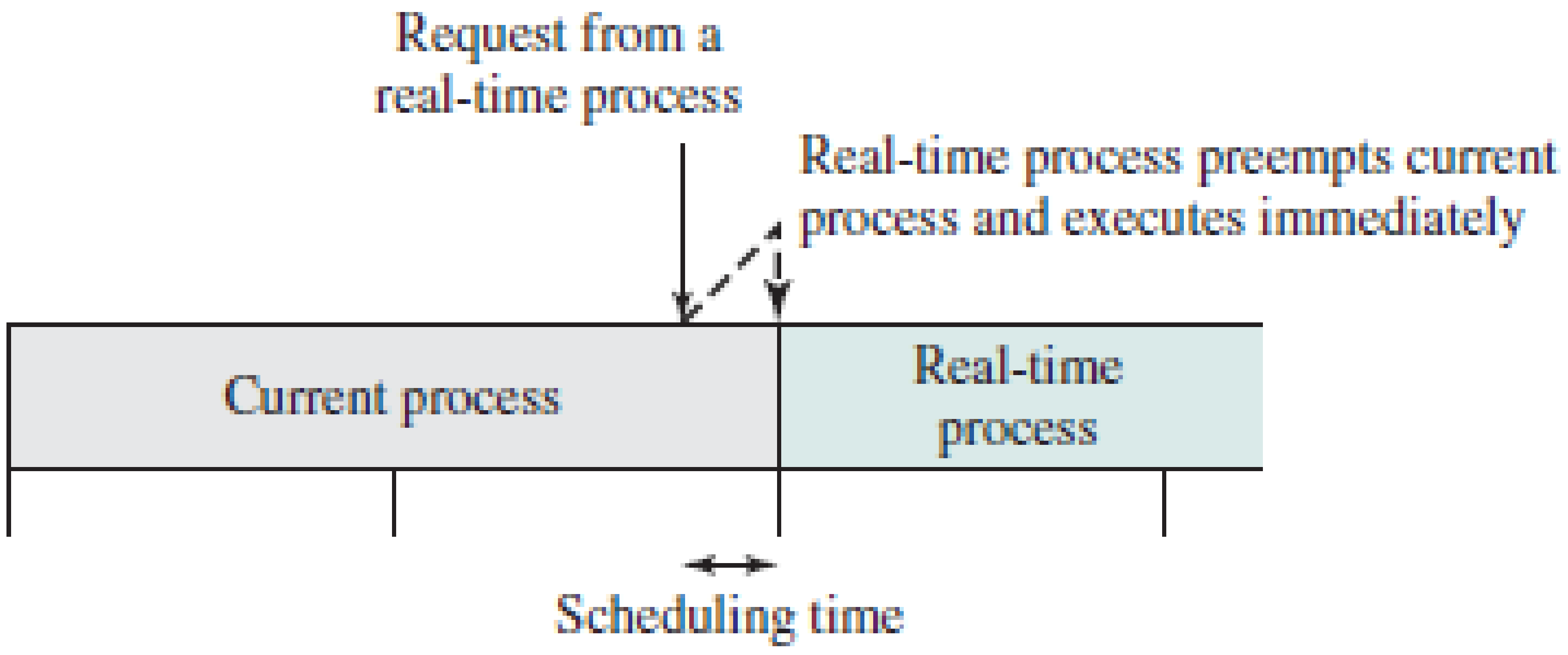
(b) Priority-driven nonpreemptive scheduler

Scheduling of Real-Time Process



(c) Priority-driven preemptive scheduler on preemption points

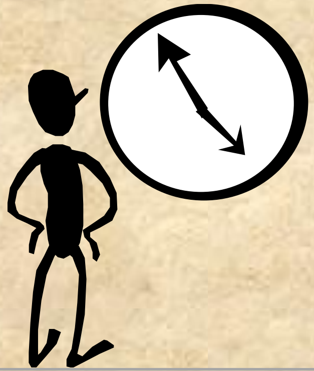
Scheduling of Real-Time Process


Request from a real-time process
Real-time process preempts current process and executes immediately
Current process
Real-time process
Scheduling time

(d) Immediate preemptive scheduler

Real-Time Scheduling

- Scheduling approaches depend 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 scheduler plan according to which tasks are dispatched at run time



Classes of Real-Time Scheduling Algorithms

- Static table-driven
- Static priority-driven preemptive
- Dynamic planning-based
- Dynamic best effort

Static table-driven

- performs a static analysis of feasible schedules of dispatching which result in a schedule that determines, at run time, when a task must begin execution
- applicable to tasks that are periodic
- Input to the analysis consists of the periodic arrival time, execution time, periodic ending deadline, and relative priority of each task.
- The scheduler attempts to develop a schedule that enables it to meet the requirements of all periodic tasks.
- This is a predictable approach but one that is inflexible, because any change to any task requirements requires that the schedule be redone.
- Example: **Earliest-deadline-first**

Static priority-driven preemptive

- a static analysis is performed but no schedule is drawn up
- analysis is used to assign priorities to tasks so that a traditional priority-driven preemptive scheduler can be used
- One example of this approach is the **rate monotonic scheduling** algorithm , which assigns static priorities to tasks based on the length of their periods.

Dynamic planning-based

- feasibility is determined at run time (dynamically) rather than offline prior to the start of execution (statically).
- an arriving task is accepted for execution only if it is feasible to meet its time constraints
- one result of the analysis is a schedule or plan that is used to decide when to dispatch this task
- an attempt is made to create a schedule that contains the previously scheduled tasks as well as the new arrival.
- If the new arrival can be scheduled in such a way that its deadlines are satisfied and that no currently scheduled task misses a deadline, then the schedule is revised to accommodate the new task.

Dynamic best effort

- no feasibility analysis is performed
- when a task arrives, the system assigns a priority based on the characteristics of the task
- system tries to meet all deadlines and aborts any started process whose deadline is missed
- Typically, the tasks are aperiodic and so no static scheduling analysis is possible.
- With this type of scheduling, until a deadline arrives or until the task completes, we do not know whether a timing constraint will be met. This is the major disadvantage of this form of scheduling.
- Its advantage is that it is easy to implement.

Deadline Scheduling

- Real-time operating systems are designed with the objective of starting real-time tasks as rapidly as possible and emphasize rapid interrupt handling and task dispatching
- Real-time applications are generally not concerned with sheer speed but rather with completing (or starting) tasks at the most valuable times
- Priorities provide a crude tool and do not capture the requirement of completion (or initiation) at the most valuable time

Information Used for Deadline Scheduling

Ready time	• time task becomes ready for execution
Starting deadline	• time task must begin
Completion deadline	• time task must be completed
Processing time	• time required to execute the task to completion
Resource requirements	• resources required by the task while it is executing
Priority	• measures relative importance of the task
Subtask scheduler	• a task may be decomposed into a mandatory subtask and an optional subtask

Table 10.3
Execution Profile of Two Periodic Tasks

Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
• • •	• • •	• • •	• • •
B(1)	0	25	50
B(2)	50	25	100
• • •	• • •	• • •	• • •

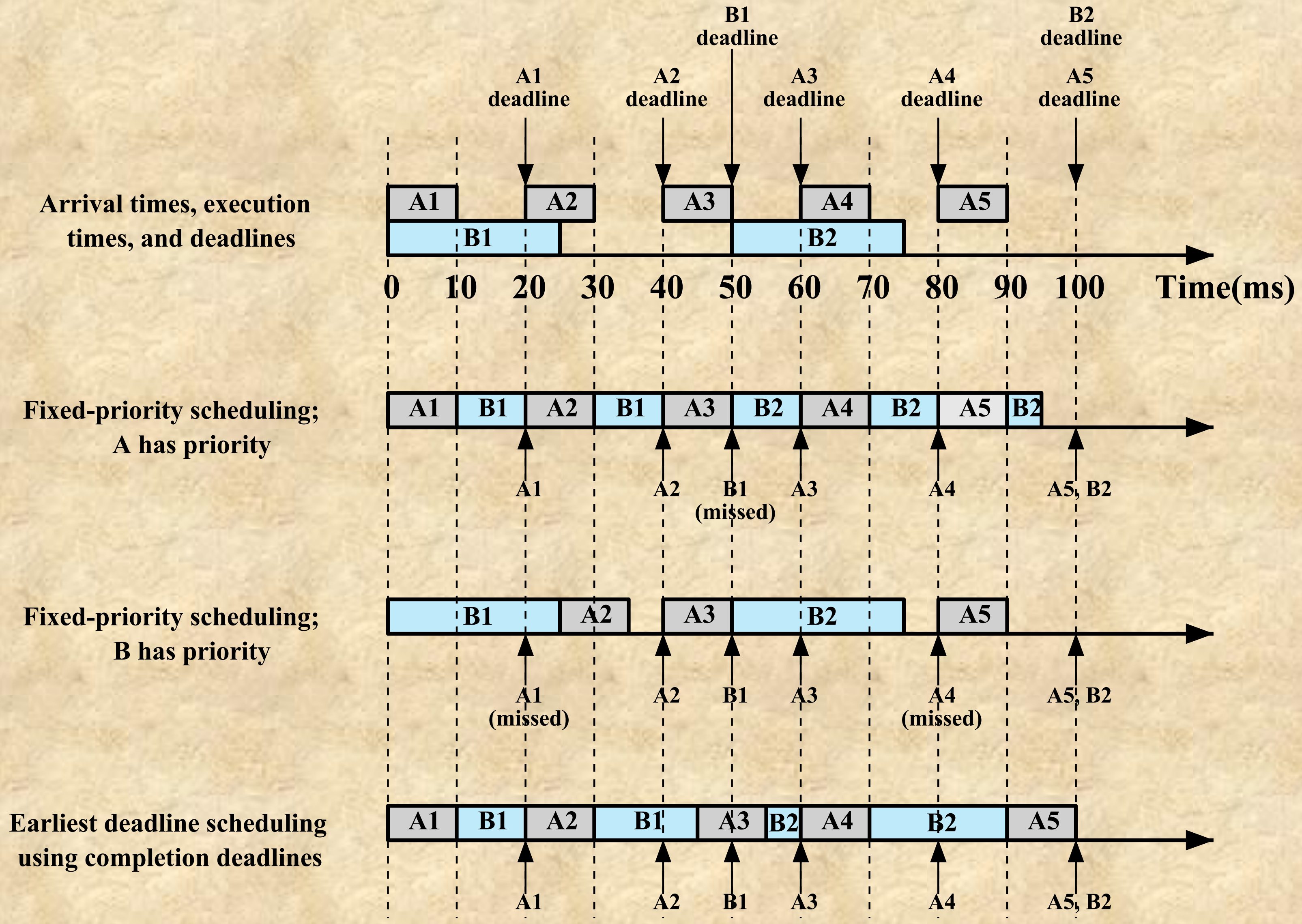

B1 deadline
B2 deadline
A1 deadline
A2 deadline
A3 deadline
A4 deadline
A5 deadline
Arrival times, execution times, and deadlines
A1
A2
A3
A4
A5
B1
B2
0
10
20
30
40
50
60
70
80
90
100
Time(ms)
Fixed-priority scheduling; A has priority
A1 B1 A2 B1 A3 B2 A4 B2 A5 B2
A1
A2
B1 (missed)
A3
A4
A5, B2
Fixed-priority scheduling; B has priority
B1 A2 A3 B2 A5
A1 (missed)
A2
B1
A3
A4 (missed)
A5, B2
Earliest deadline scheduling using completion deadlines
A1 B1 A2 B1 A3 B2 A4 B2 A5
A1
A2
B1
A3
A4
A5, B2

Figure 10.5 Scheduling of Periodic Real-time Tasks with Completion Deadlines (based on Table 1 10.3)

Table 10.4
Execution Profile of Five Aperiodic Tasks

Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

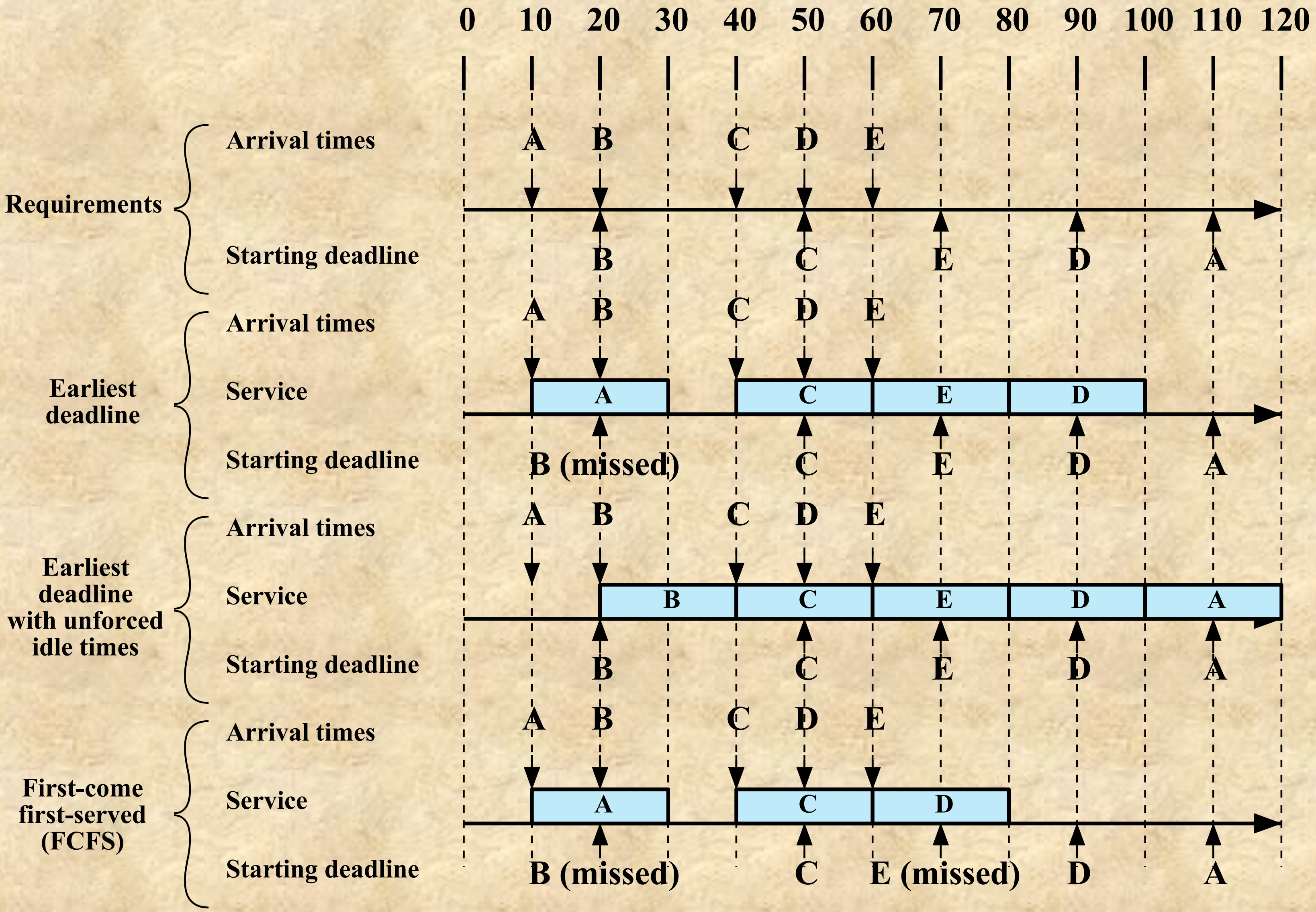
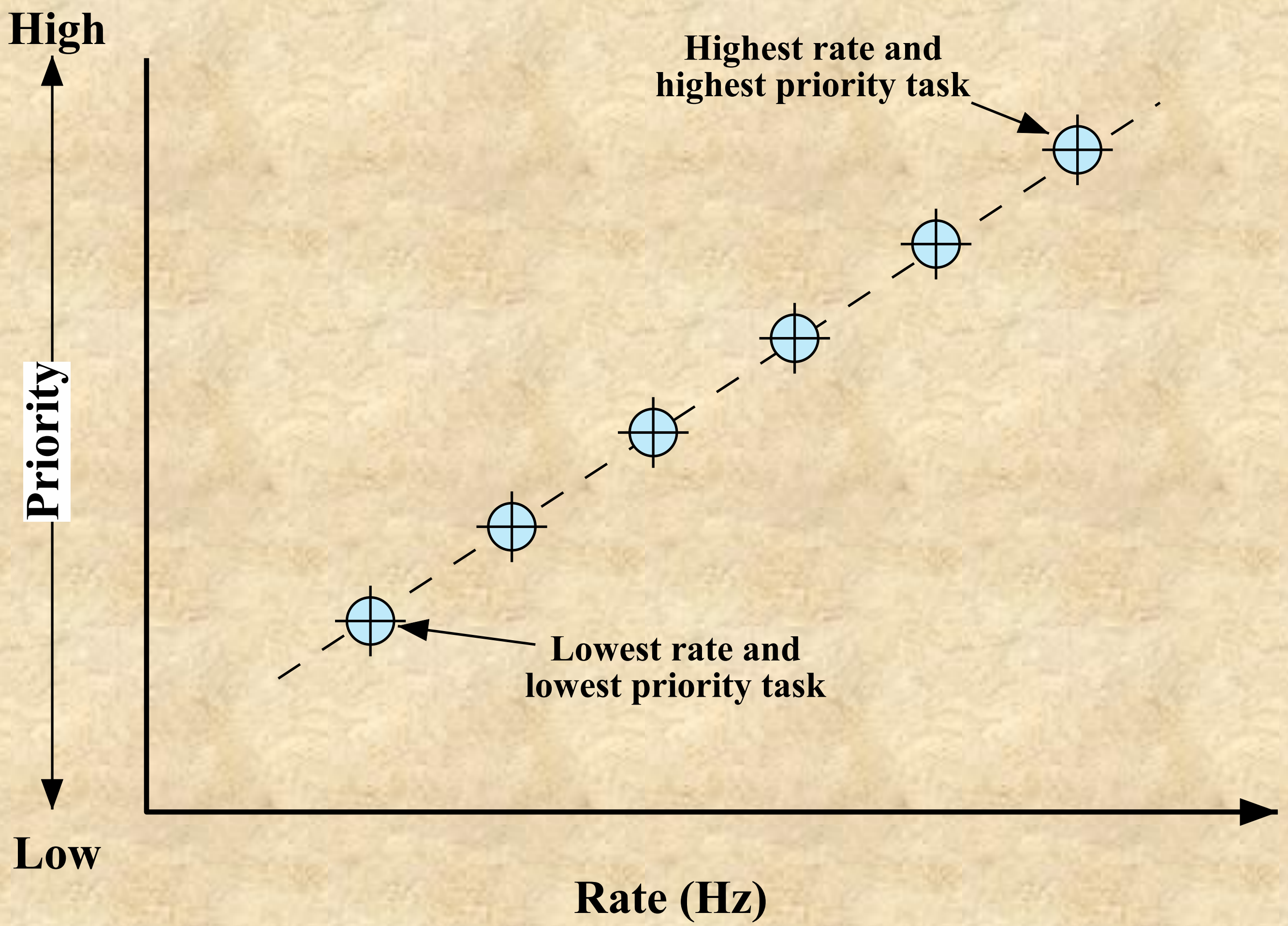
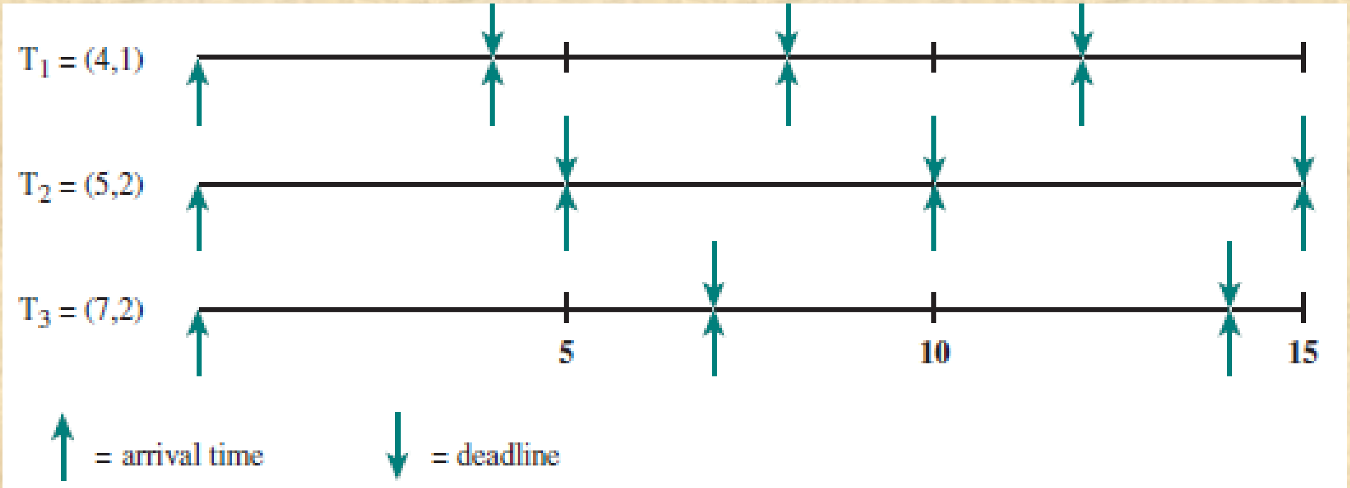


Figure 10.6 Scheduling of Aperiodic Real-time Tasks with Starting Deadlines

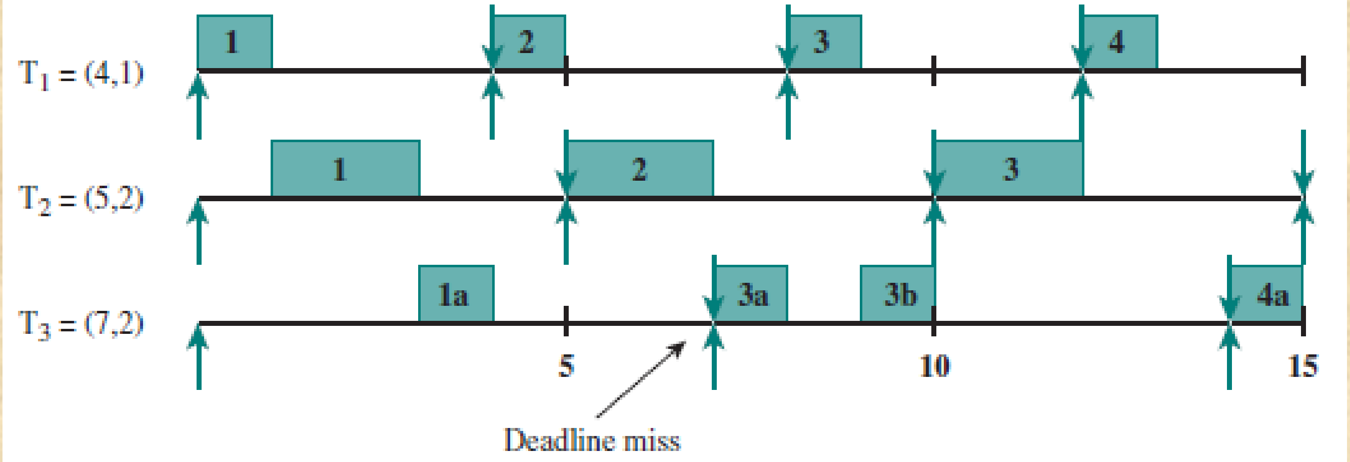


Rate Monotonic Scheduling (RMS)

Figure 10.7 A Task Set with RMS



(a) Arrival times and deadines for task $T_i = (P_i, C_i)$;
P_i = period, C_i = processing time



(b) Scheduling results

Rate Monotonic Scheduling (RMS) Example

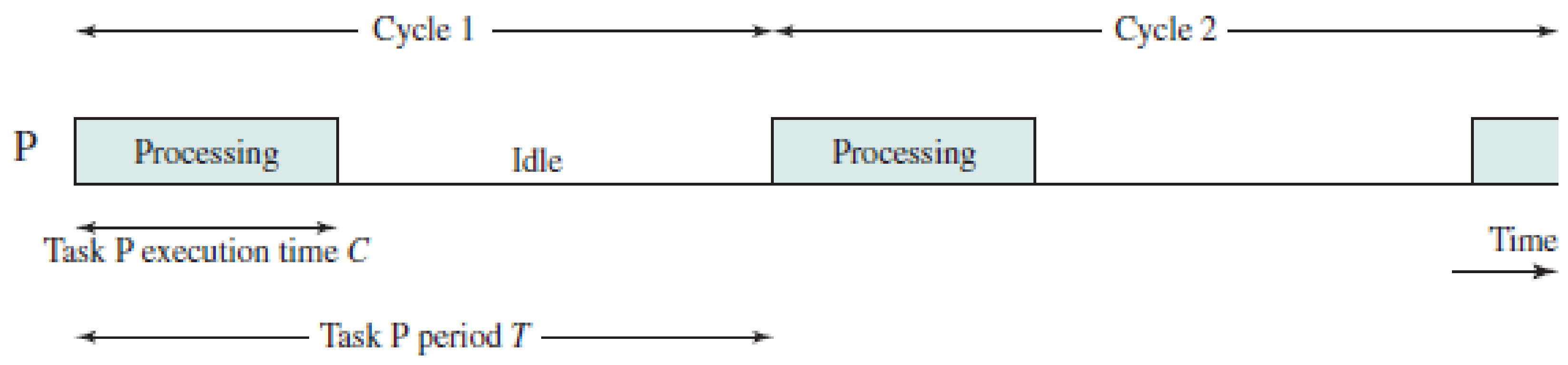


Figure 10.8 **Periodic Task Timing Diagram**

For Perfect Scheduling

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leq 1$$

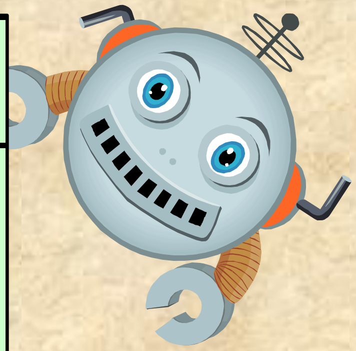
For RMS only

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leq n(2^{1/n} - 1)$$

Table 10.5

Value of the RMS Upper Bound

n	$n(2^{1/n} - 1)$
1	1.0
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
•	•
•	•
•	•
¥	ln 2 » 0.693



As an example, consider the case of three periodic tasks, where $U_i = C_i/T_i$:

- **Task P$_1$:** $C_1 = 20$; $T_1 = 100$; $U_1 = 0.2$
- **Task P$_2$:** $C_2 = 40$; $T_2 = 150$; $U_2 = 0.267$
- **Task P$_3$:** $C_3 = 100$; $T_3 = 350$; $U_3 = 0.286$

The total utilization of these three tasks is $0.2 + 0.267 + 0.286 = 0.753$. The upper bound for the schedulability of these three tasks using RMS is

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} \leq n(2^{1/3} - 1) = 0.779$$

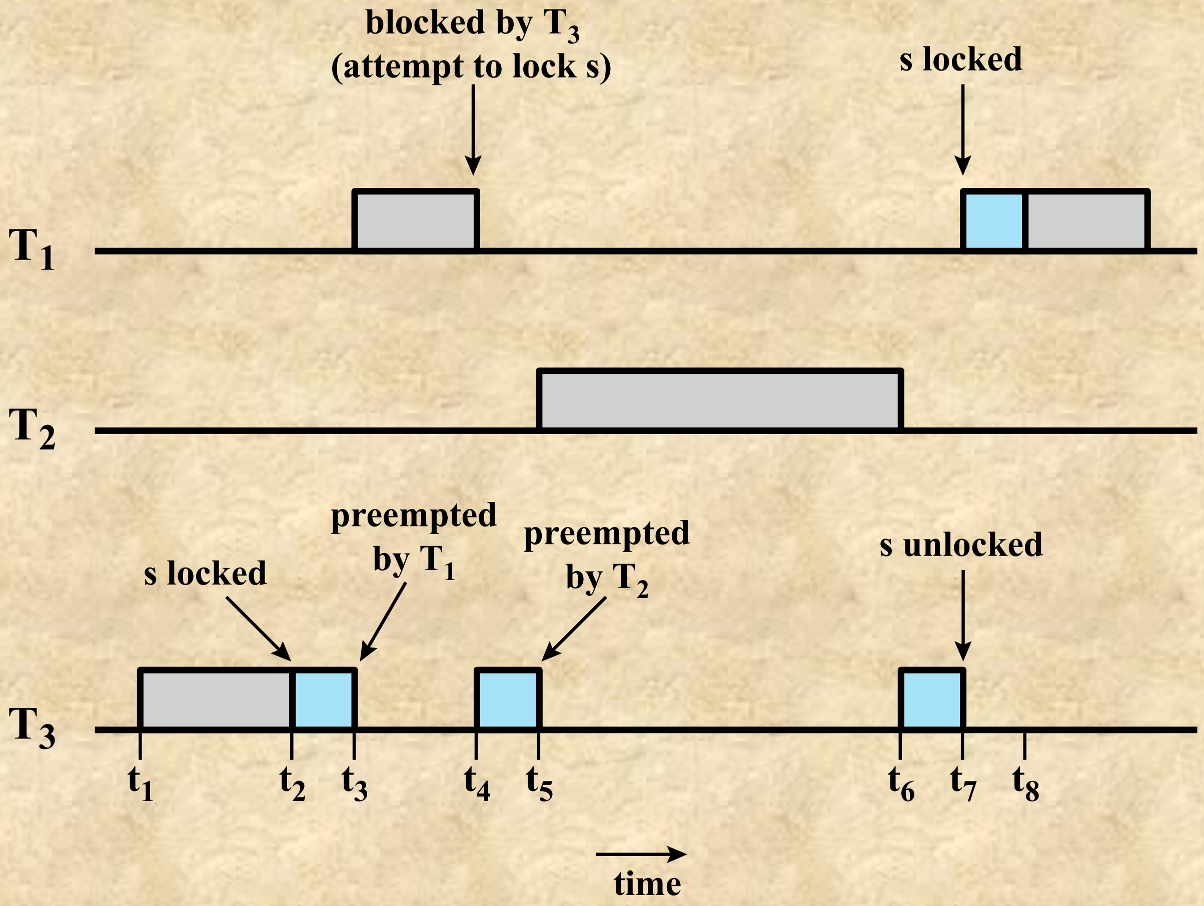
Priority Inversion

- Can occur in any priority-based preemptive scheduling scheme
- Particularly relevant in the context of real-time scheduling
- Best-known instance involved the Mars Pathfinder mission in 1997
- Occurs when circumstances within the system force a higher priority task to wait for a lower priority task
- If a lower-priority task has locked a resource and a higher-priority task attempts to lock that resource, the higher-priority task will be put in a blocked state until the resource is available.
- If the lower-priority task soon finishes with the resource and releases it, the higher-priority task may quickly resume and it is possible that no real-time constraints are violated.

Unbounded Priority Inversion

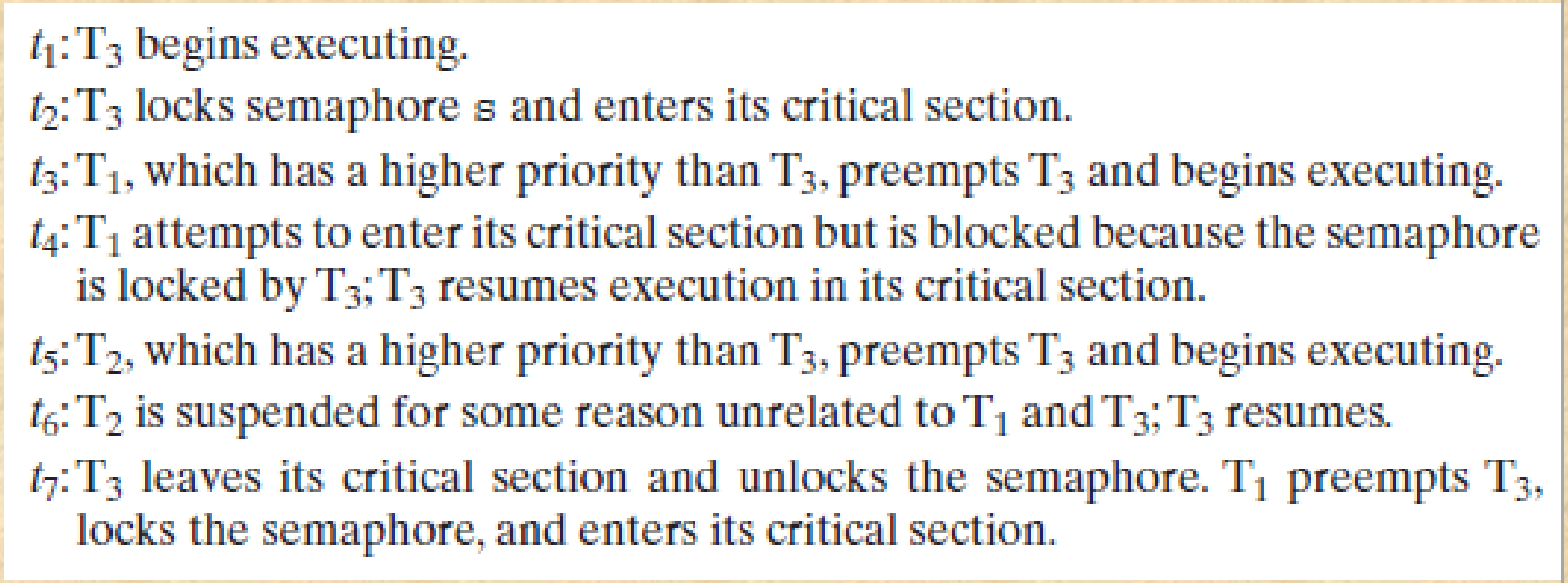
- The duration of a priority inversion depends not only on the time required to handle a shared resource, but also on the unpredictable actions of other unrelated tasks

Unbounded Priority Inversion



(a) Unbounded priority inversion

Unbounded Priority Inversion



t_1: T_3 begins executing.

t_2: T_3 locks semaphore s and enters its critical section.

t_3: T_1, which has a higher priority than T_3, preempts T_3 and begins executing.

t_4: T_1 attempts to enter its critical section but is blocked because the semaphore is locked by T_3; T_3 resumes execution in its critical section.

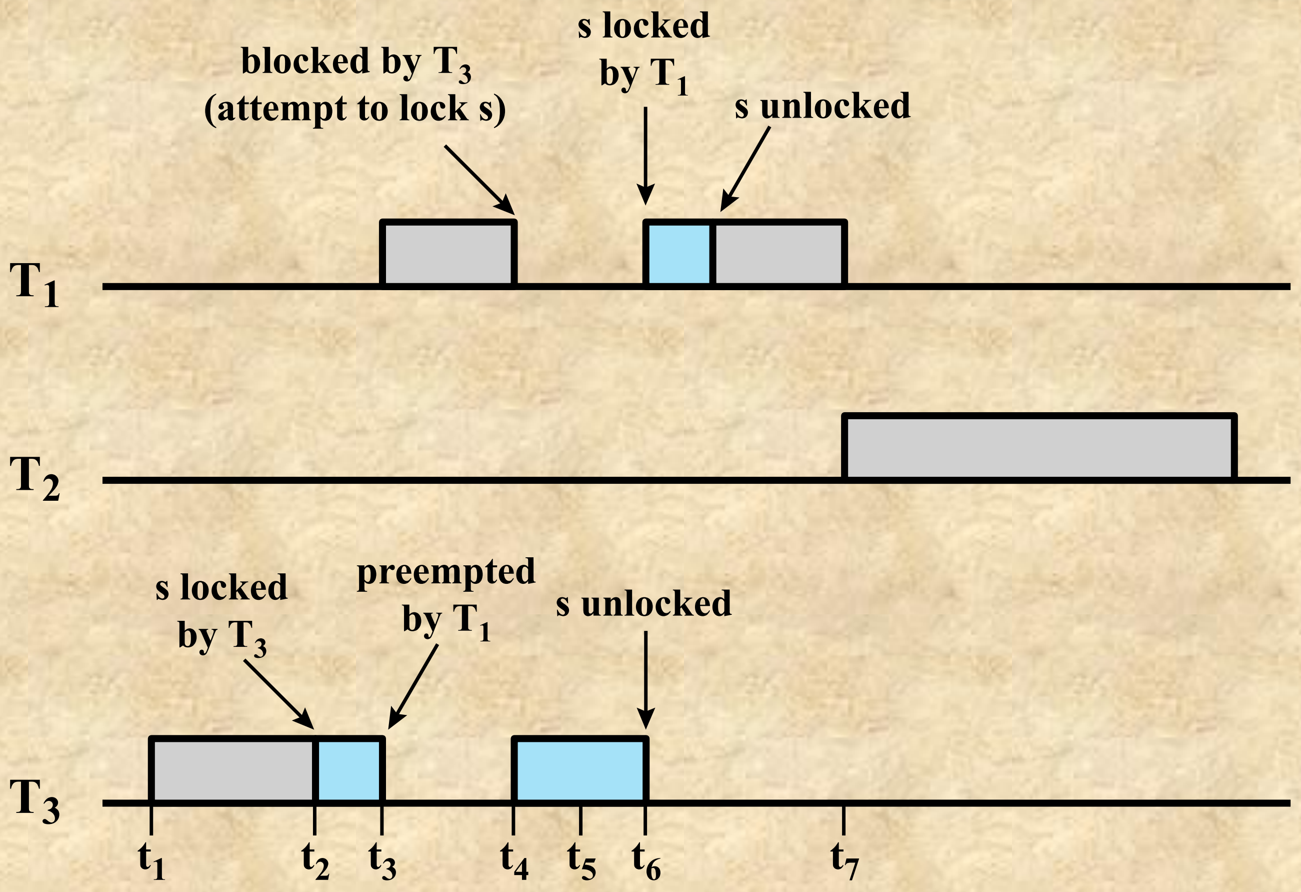
t_5: T_2, which has a higher priority than T_3, preempts T_3 and begins executing.

t_6: T_2 is suspended for some reason unrelated to T_1 and T_3; T_3 resumes.

t_7: T_3 leaves its critical section and unlocks the semaphore. T_1 preempts T_3, locks the semaphore, and enters its critical section.

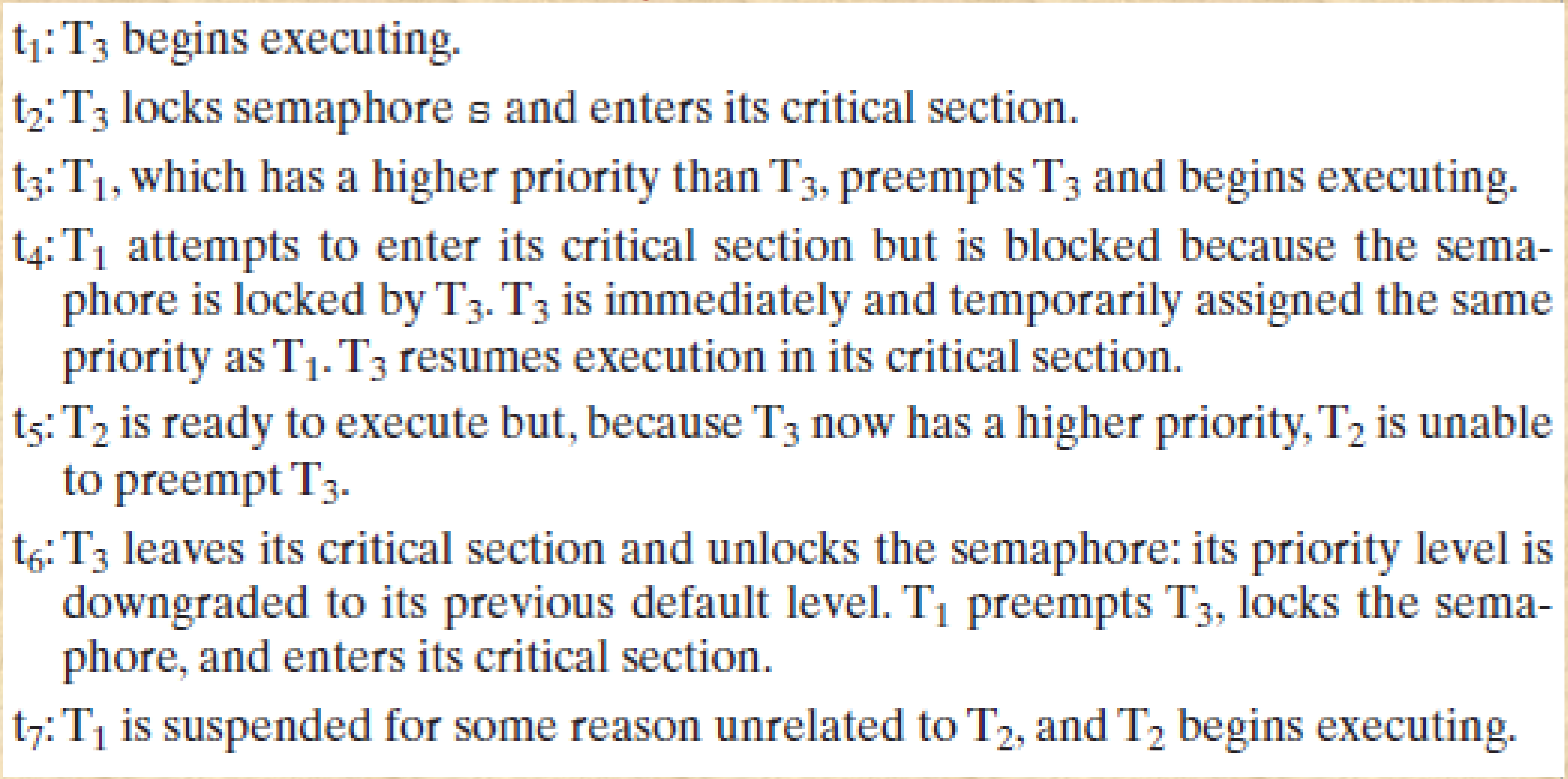
- T_1 must wait for both T_3 and T_2 to complete and fails.

Priority Inheritance



(b) Use of priority inheritance

Priority Inheritance



t_1: T_3 begins executing.

t_2: T_3 locks semaphore s and enters its critical section.

t_3: T_1, which has a higher priority than T_3, preempts T_3 and begins executing.

t_4: T_1 attempts to enter its critical section but is blocked because the semaphore is locked by T_3. T_3 is immediately and temporarily assigned the same priority as T_1. T_3 resumes execution in its critical section.

t_5: T_2 is ready to execute but, because T_3 now has a higher priority, T_2 is unable to preempt T_3.

t_6: T_3 leaves its critical section and unlocks the semaphore: its priority level is downgraded to its previous default level. T_1 preempts T_3, locks the semaphore, and enters its critical section.

t_7: T_1 is suspended for some reason unrelated to T_2, and T_2 begins executing.

Priority Ceiling

- A priority is associated with each resource.
- The priority assigned to a resource is one level higher than the priority of its highest priority user.
- The scheduler then dynamically assigns this priority to any task that accesses the resource.
- Once the task finishes with the resource, its priority returns to normal.

Summary

- Real-time scheduling
  - Background
  - Characteristics of real-time operating systems
  - Real-time scheduling
  - Deadline scheduling
  - Rate monotonic scheduling
  - Priority inversion