Operating Systems (A)
(Honor Track)

Lecture 13: Scheduling (II)

Yao Guo (郭耀)

Peking University

Fall 2021

Acknowledgements: Prof. Xiangqun Chen & Tao Wang at PKU and Prof. Yuanyuan Zhou at UCSD

Review: Scheduling

Scheduling objectives:

- Performance
- Fair
- Priority
- Encourage good behavior
- Support heavy loads
- Adapt to different environments
  - interactive, real-time, multi-media

Single Processor Scheduling

- Scheduling for batch systems
  - First Come First Serve (FCFS)
  - Shortest Job First (SJF)
- Interactive scheduling
  - Round Robin (RR)
  - Priority Scheduling
    - Priority inversion
    - Priority inheritance
  - Multiple-Level Feedback Queues (MLFQ)

This Lecture

Multiprocessor and Real-Time Scheduling

Multiprocessor scheduling

Real-time scheduling

Buzz Words

- **Load sharing**
- **Gang scheduling**
- **Hard/soft real-time task**
- **Periodic/aperiodic task**
- **Deadline scheduling**
- **Rate monotonic scheduling (RMS)**

This Lecture

Multiprocessor and Real-Time Scheduling

Multiprocessor scheduling

Real-time scheduling

Classifications of Multiprocessor Systems

- Loosely coupled processors
  - Each has their memory & I/O channels
- Functionally specialized processors
  - Such as I/O processor
  - Controlled by a master processor
- Tightly coupled multiprocessor
  - Processors share main memory
  - Controlled by operating system
- We will mainly talk about tightly coupled multiprocessor in this lecture

Assignment of Processes to Processors

- The simplest way: treat processors as a pooled resource and assign processes to processors on demand
- Should the assignment be static or dynamic?
- Dynamic assignment
  - Threads can move from a queue for one processor to a queue for another processor
  - E.g., Linux

Static Assignment

- Permanently assign processes to a processor
  - Dedicated short-term queue for each processor
  - Less overhead
  - Allows the use of ‘group’ or ‘gang’ scheduling (see later)
- But may leave a processor idle, while others have a backlog
- Solution: use a common queue
  - All processes go into one global queue and are scheduled to any available processor

Assignment of Processes to Processors

- Both dynamic and static methods require some way of assigning a process to a processor
- Two methods:
  - Master/Slave
  - Peer
- There are of course a spectrum of approaches between these two extremes

Master / Slave Architecture

- Key kernel functions always run on a particular processor
- Master: responsible for scheduling
- Slave: sends service request to the master
- Disadvantages
  - Failure of master brings down whole system
  - Master can become a performance bottleneck

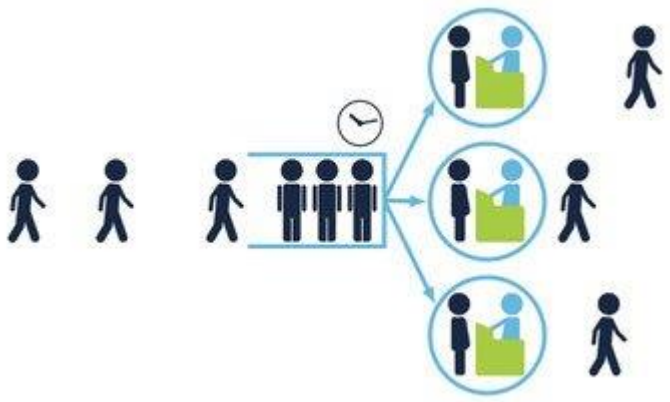
Peer Architecture

- Kernel can execute on any processor
- Each processor does self-scheduling from the pool of available processes.

- Complicates the operating system
  - The operating system must ensure that two processors do not choose the same process and that the processes are not somehow lost from the queue.
  - Techniques must be employed to resolve and synchronize competing claims to resources.

Process Scheduling

- Usually processes are not dedicated to processors
- A single queue is used for all processes
- Or use multiple queues based on priority
  - All queues feed to the common pool of processors
- We can view the system as being a multiserver queuing architecture.



Thread Scheduling

- Threads execute separately from the rest of the process
  - An application can be implemented as a set of threads that cooperate and execute concurrently in the same address space
- Threads can be used to exploit true parallelism in an application
  - Dramatic gains in performance are possible in multi-processor systems, compared to running in uniprocessor systems

Approaches to Thread Scheduling

- Many proposals exist but four general approaches stand out:
  - Load Sharing
  - Gang Scheduling
  - Dedicated processor assignment
  - Dynamic scheduling

Load Sharing

- A global queue of ready threads is maintained
  - In the (shared) main memory
- Each processor, when idle, selects a thread from the queue

- Load is distributed evenly across the processors
- No centralized scheduler required

Disadvantages of Load Sharing

- The central queue needs mutual exclusion
  - It may become a bottleneck if many processors look for work at the same time.
- Preempted threads are unlikely to resume execution on the same processor
  - Cache issues…
- If all threads are in the global queue, all threads of a program will not be likely to gain access to the processors at the same time

Gang Scheduling

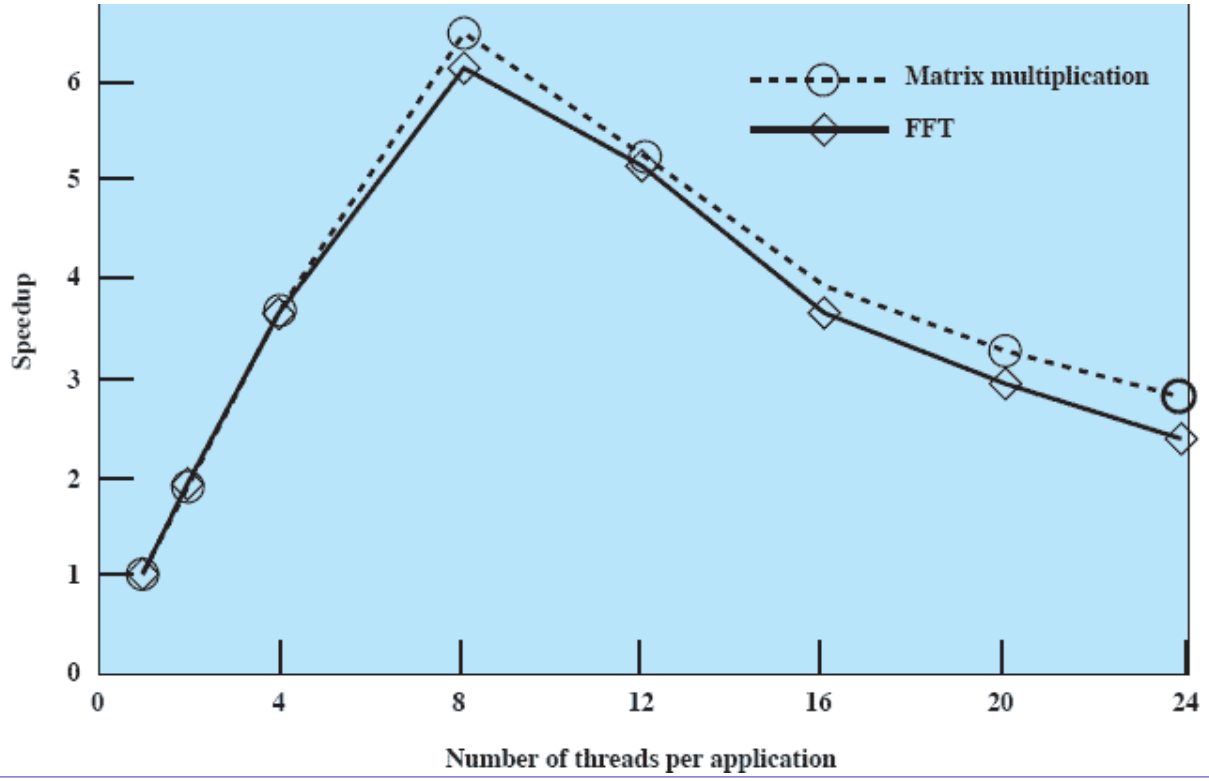
- A set of related threads is scheduled to run on a set of processors at the same time
  - If closely related processes execute in parallel, synchronization blocking may be reduced, less process switching may be necessary, and performance will increase.
  - Scheduling overhead may be reduced because a single decision affects a number of processors and processes at one time.

Dedicated Processor Assignment

- When an application is scheduled, each of its threads is assigned to a processor
- Some processors may be idle
  - No multiprogramming of processors (time-multiplexing)
- ***But***
  - In *highly* parallel systems, processor utilization is less important than effectiveness
  - Avoiding process switching speeds up programs

Dynamic Scheduling

- The number of threads in a process are altered dynamically by the application
  - This allows OS to adjust the load to improve utilization



This Lecture

Multiprocessor and Real-Time Scheduling

Multiprocessor scheduling

Real-time scheduling

Real-Time Scheduling

- Real-time computing:
  - 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

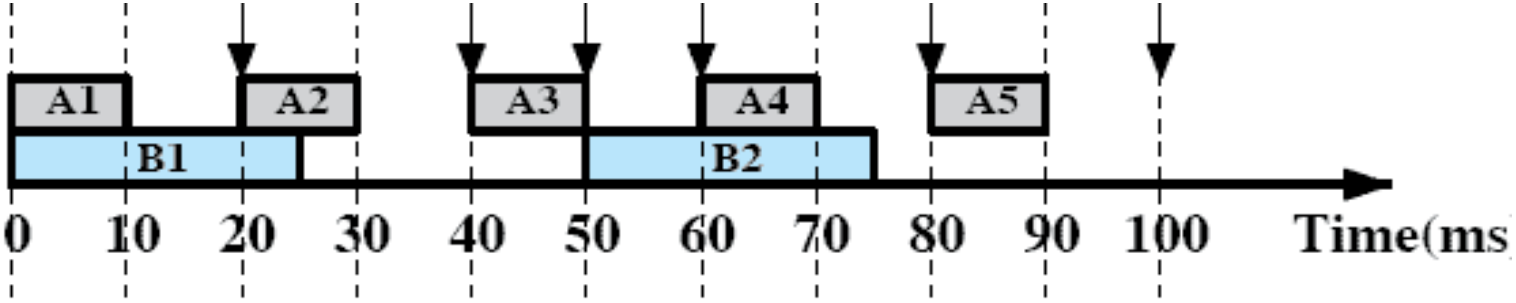
Hard vs Soft Real-time

- A hard real-time task is one that must meet its deadline
  - Otherwise it will cause unacceptable damage or a fatal error to the system.

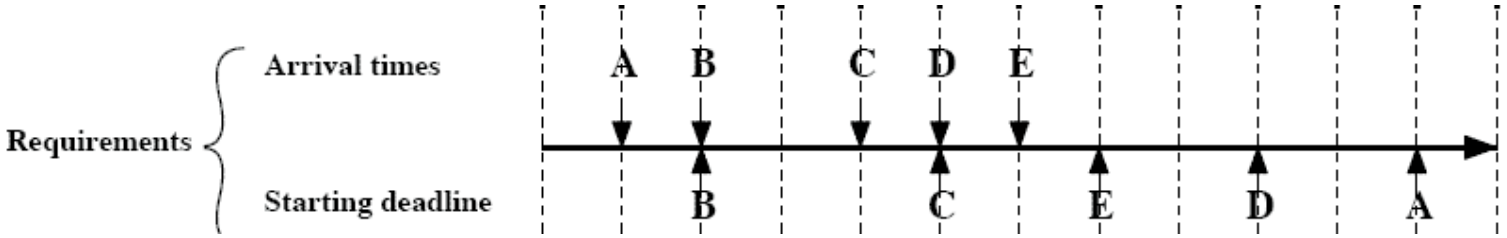
- A soft real-time task has an associated deadline that is desirable but not mandatory
  - it still makes sense to schedule and complete the task even if it has passed its deadline.

Periodic vs Aperiodic

- Periodic tasks
  - The requirement may be stated as "once per period T" or "exactly T units apart."



- Aperiodic tasks
  - Have time constraints either for deadlines or start



Real-Time Systems

- Control of laboratory experiments
- Process control in industrial plants
- Air traffic control
- Telecommunications
- Military command and control systems
- Robotics

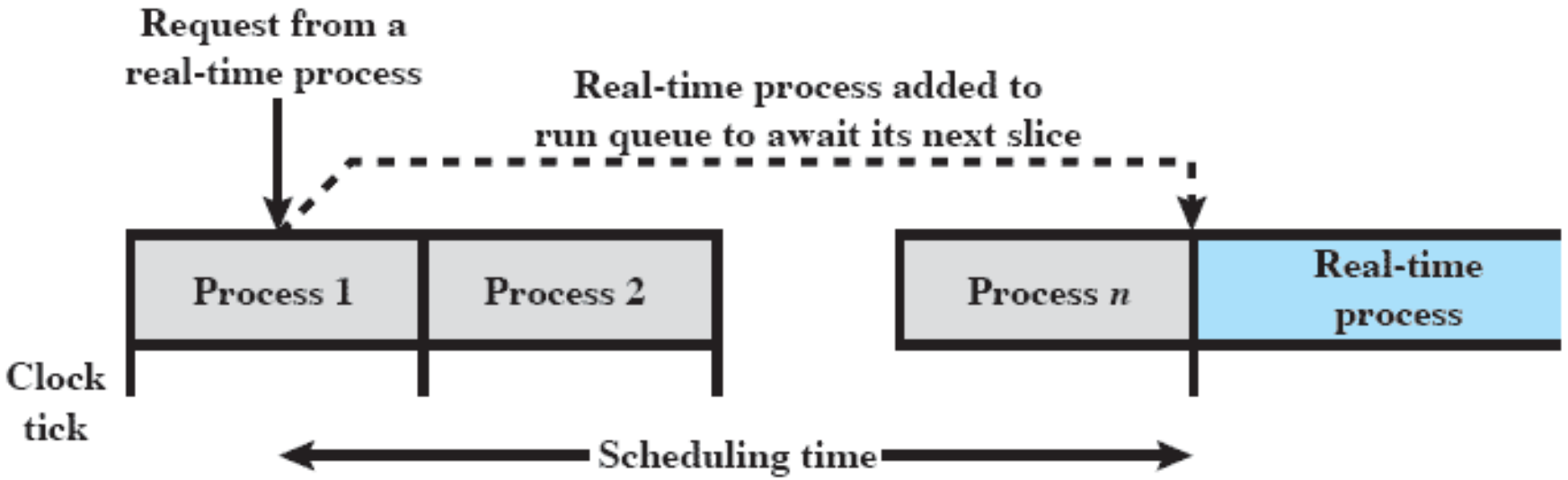
Features of Real-Time OS

- Fast process or thread switch
- Small size
- Ability to respond to external interrupts quickly
- Multitasking with inter-process communication tools such as semaphores, signals, and events

Features of Real-Time OS (cont…)

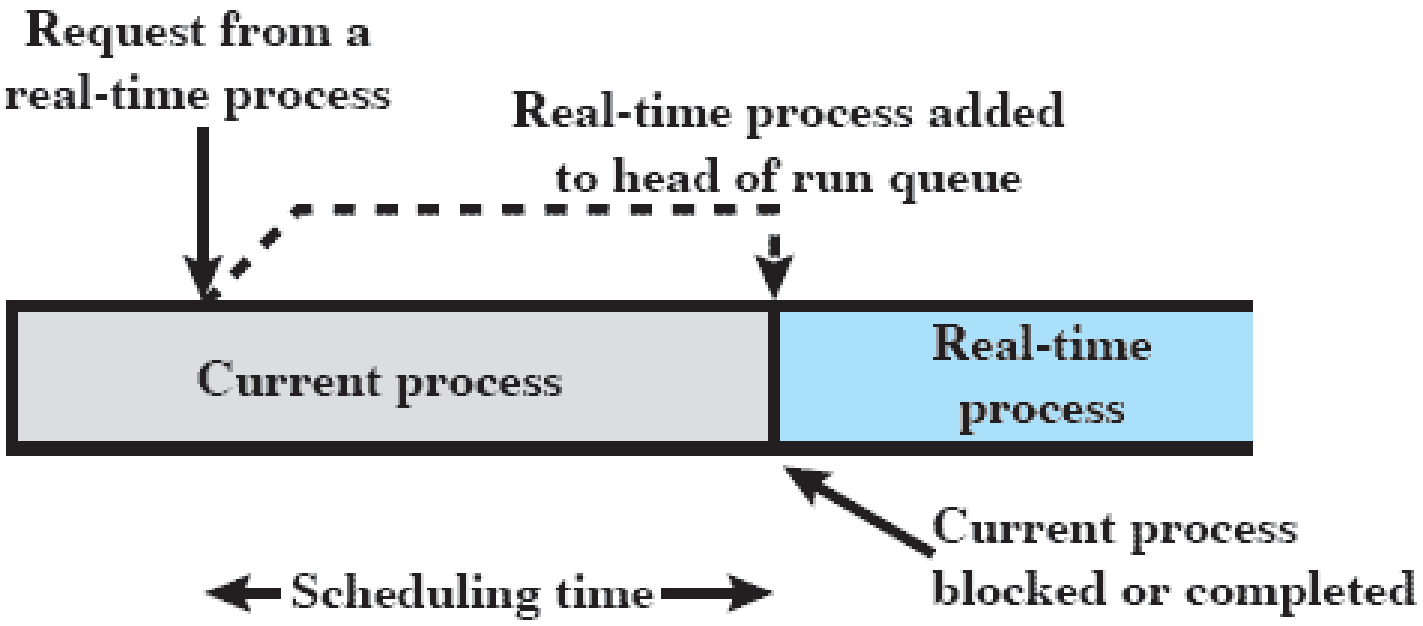
- Use of special sequential files that can accumulate data at a fast rate
- Preemptive scheduling based on priority
- Minimization of intervals during which interrupts are disabled
- Delay tasks for fixed amount of time
- Special alarms and timeouts

Round-robin Preemptive Scheduling: Unacceptable



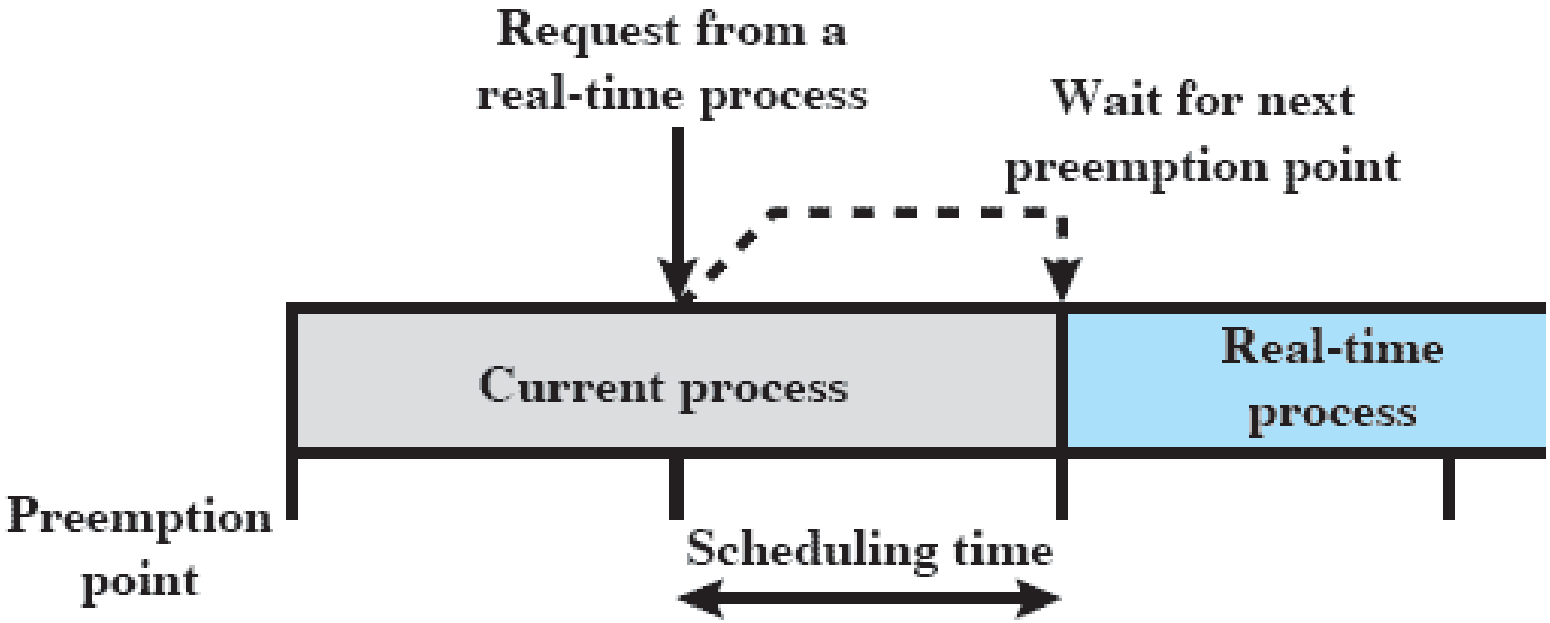
(a) Round-robin Preemptive Scheduler

Priority-driven Nonpreemptive Scheduling: Unacceptable



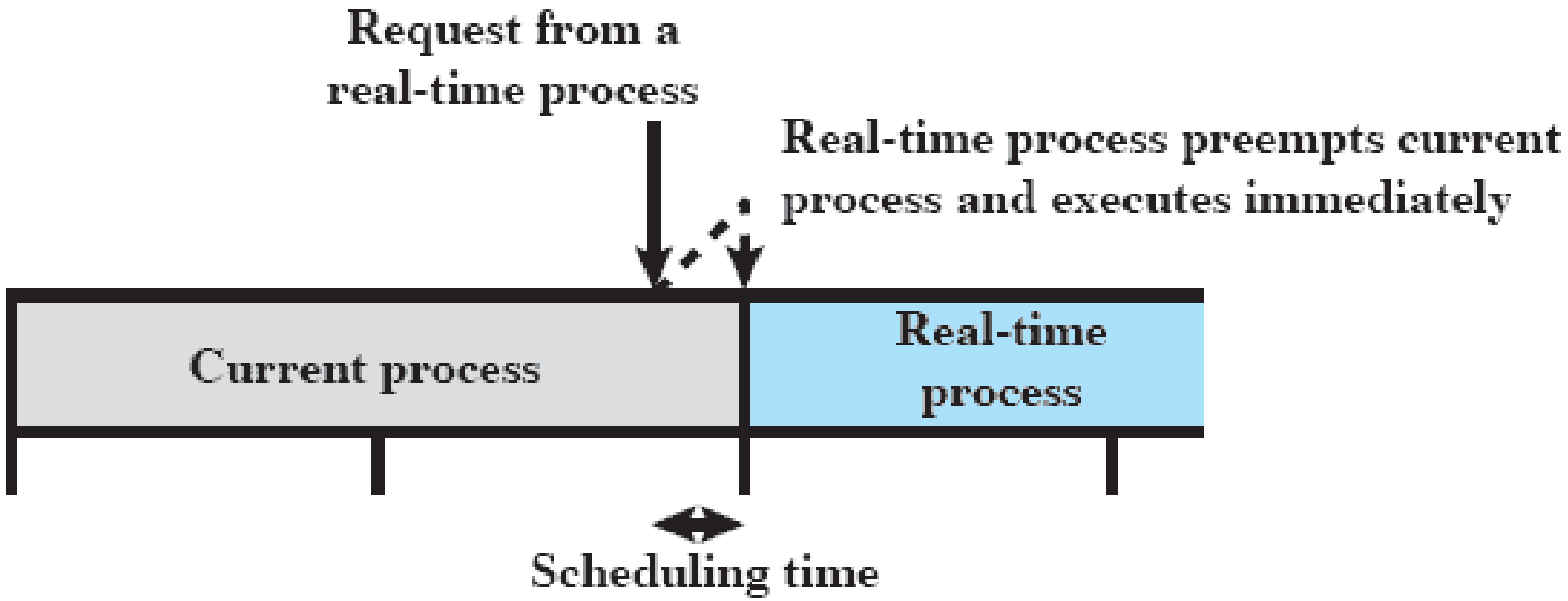
(b) Priority-Driven Nonpreemptive Scheduler

Combine Priorities with Clock-based Interrupts



(c) Priority-Driven Preemptive Scheduler on Preemption Points

Immediate Preemption



(d) Immediate Preemptive Scheduler

Classes of Real-Time Scheduling Algorithms

- Static table-driven
  - Performs a static analysis of feasible schedules of dispatching
  - Determines when a task must begin execution at run time
- Static priority-driven preemptive
  - A static analysis is performed, but no schedule is drawn up
  - Traditional priority-driven preemptive scheduler is used
- Dynamic planning-based
  - Feasibility determined at run time (dynamically) rather than offline
  - An arriving task is accepted for execution only if it is feasible to meet its time constraints
- Dynamic best effort
  - No feasibility analysis is performed
  - The system tries to meet all deadlines and aborts any started process whose deadline is missed

Policy: Deadline Scheduling

- Real-time applications are generally not concerned with sheer speed but with completing (or starting) tasks
  - at the most valuable times, neither too early nor too late
  - despite dynamic resource demands and conflicts, processing overloads, and hardware or software faults

- "Priorities" are a crude tool and may not capture the requirement of completion (or initiation) at the most valuable time

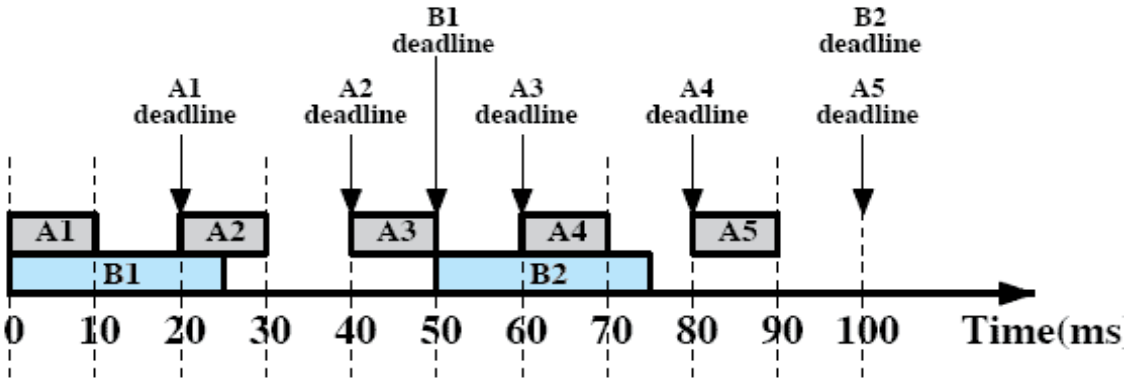
Deadline Scheduling

- Information used
  - Ready time
  - Starting deadline
  - Completion deadline
  - Processing time
  - Resource requirements
  - Priority
  - Subtask scheduler
- For
  - Periodic tasks
  - Aperiodic tasks

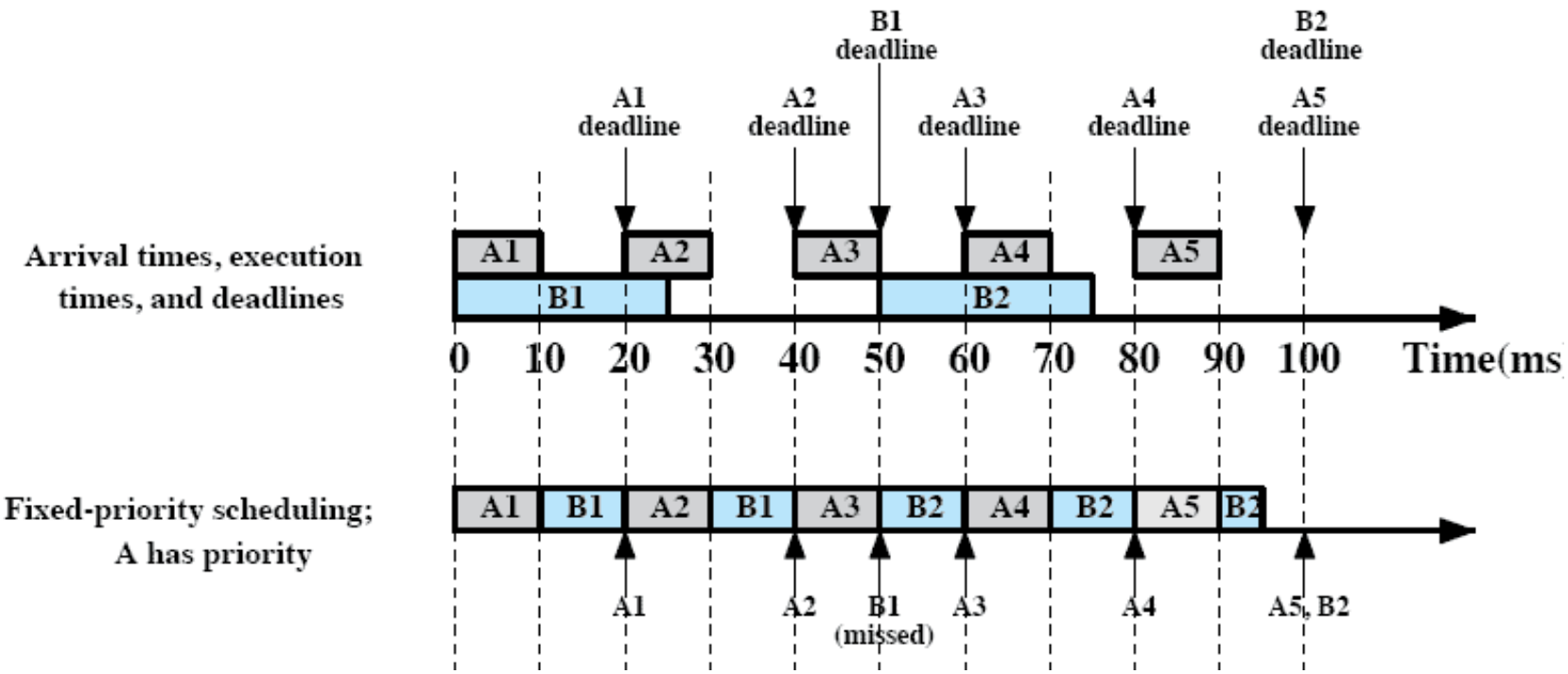
Periodic Scheduling: Two Periodic Tasks

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

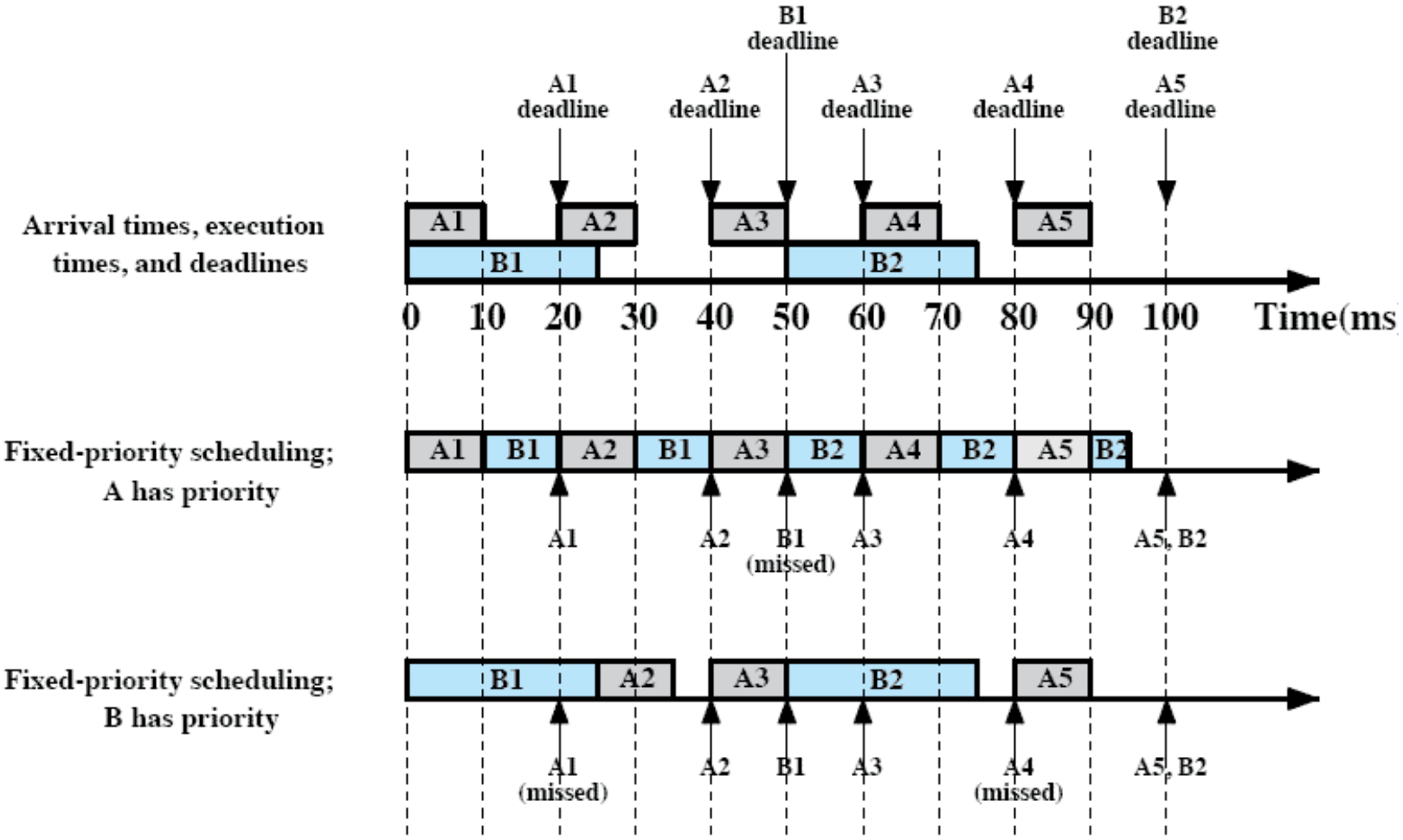
Arrival times, execution times, and deadlines



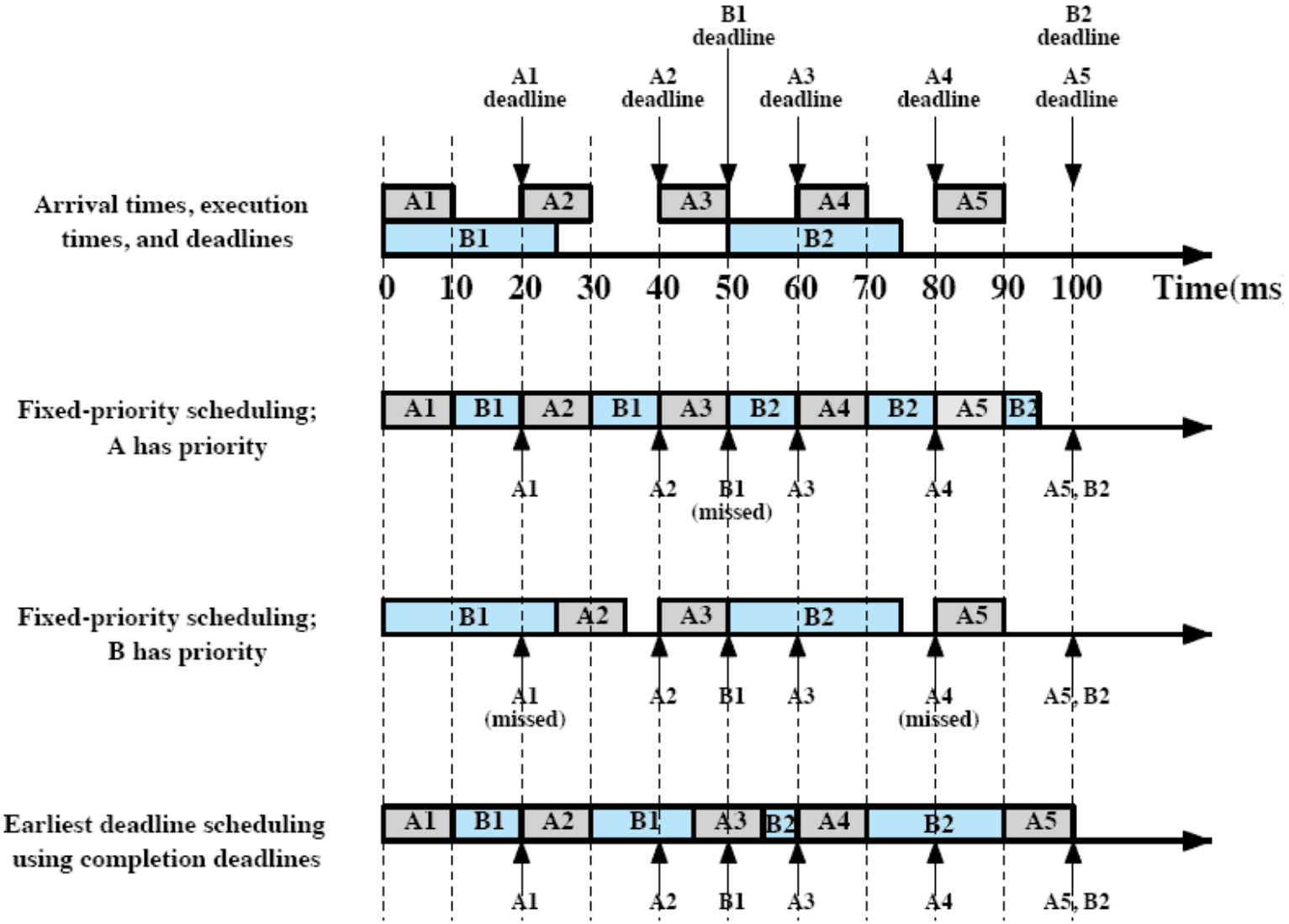
Periodic Scheduling



Periodic Scheduling



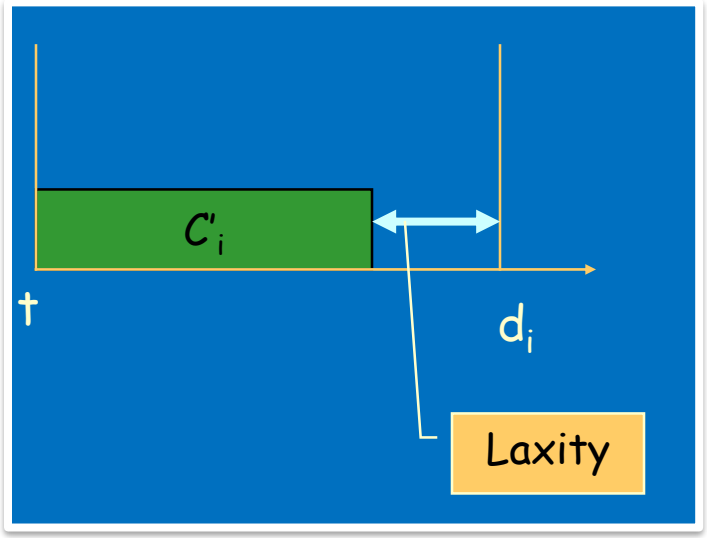
Periodic Scheduling



Periodic Scheduling: Priority-driven Preemptive Scheduling

Assumptions & Definitions

- Tasks are periodic
- No aperiodic or sporadic tasks
- Job (instance) deadline = end of period
- No resource constraints
- Tasks are preemptable



- Laxity of a Task
- $T_i = d_i - (t + c_i')$

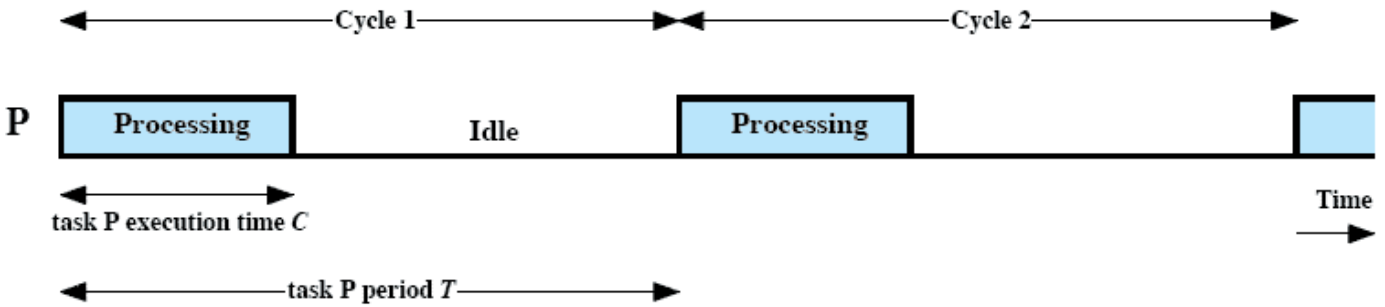
  where di: deadline;

  t : current time;

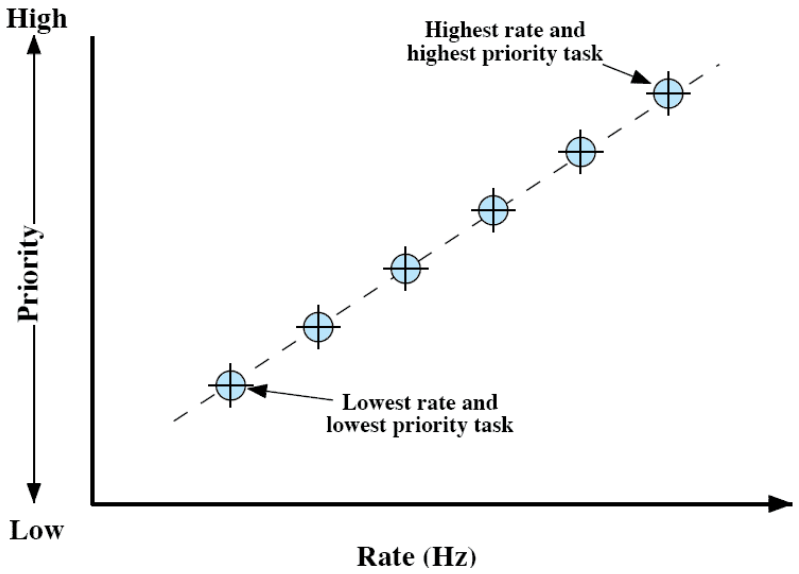
  c_i' : remaining computation time.

Rate Monotonic Scheduling (RMS)

- Assigns priorities to tasks on the basis of their periods



- The highest-priority task is the one with the shortest period



- The rate monotonic algorithm (RMA) is a procedure for assigning fixed priorities to tasks to maximize their "schedulability."
- A task set is considered schedulable if all tasks meet all deadlines all the time.

Rate Monotonic Scheduling (RMS)

- **Schedulability check (off-line)**

  - A set of ***n*** tasks is schedulable on a uniprocessor by the RMS algorithm if the processor utilization (utilization test):

$$\sum_{i=1}^{n} c_i/p_i \leq n(2^{1/n} - 1).$$

  The term $n(2^{1/n} - 1)$ approaches $\ln 2$, (≈ 0.69 as $n \to \infty$).

  - This condition is sufficient, but not necessary.

RMS (cont.)

- Schedule construction (online)

  - Task with the smallest period is assigned the highest priority.

  - At any time, the highest priority task is executed.

- RMS is an optimal preemptive scheduling algorithm with fixed priorities.
- Static/fixed priority algorithm assigns the same priority to all the jobs (instances) in each task.

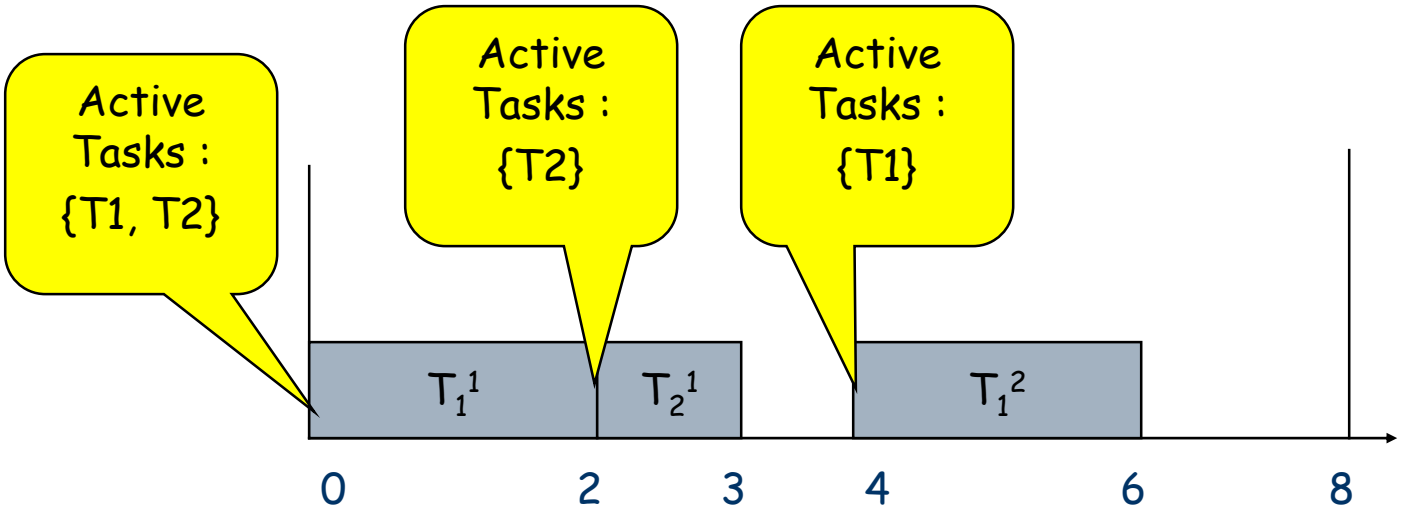
RMS Scheduler -- Example 1

Task set: $T_i = (c_i, p_i)$

T1 = (2,4) and T2 = (1,8)

Schedulability check:

$2/4 + 1/8 = 0.5 + 0.125 = 0.625 \leq 2(\sqrt{2} - 1) = 0.82$



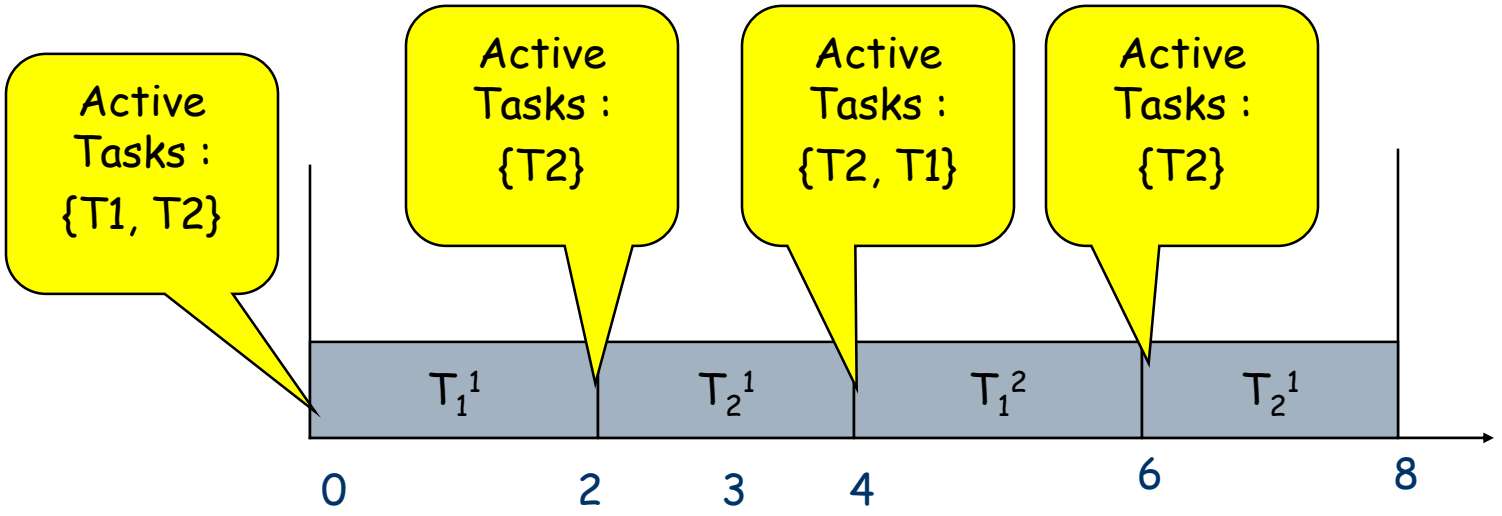
RMS scheduler -- Example-2

Task set: $T_i = (c_i, p_i)$

T1 = (2,4) and T2 = (4,8)

Schedulability check:

$2/4 + 4/8 = 0.5 + 0.5 = 1.0 > 2(\sqrt{2} - 1) = 0.82$



Some task sets that FAIL the utilization-based schedulability test are also schedulable under RMS ➔ We need exact analysis (necessary & sufficient)

Earliest Deadline First (EDF)

- **Schedulability check (off-line)**

  - A set of ***n*** tasks is schedulable on a uniprocessor by the EDF algorithm if the processor utilization

$$\sum_{i=1}^{n} c_i / p_i \leq 1$$

- This condition is both necessary and sufficient.

- **Least Laxity First (LLF)** algorithm has the same schedulability check.

EDF/LLF (cont.)

- **Schedule construction (online)**
  - EDF/LLF: Task with the smallest deadline/laxity is assigned the highest priority.
  - At any time, the highest priority task is executed.

EDF/LLF is an optimal preemptive scheduling algorithm with dynamic priorities.

Dynamic priority algorithm assigns different priorities to the individual jobs (instances) in each task.

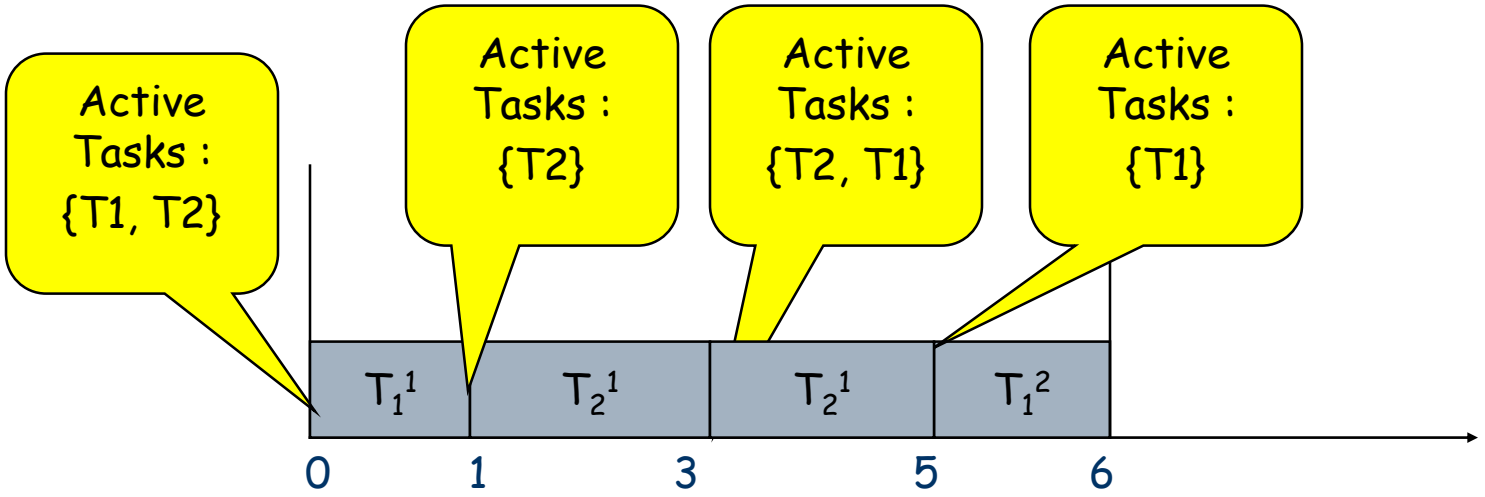
EDF scheduler -- Example

Task set: $T_i = (c_i, p_i, d_i)$

T1 = (1,3,3) and T2 = (4,6,6)

Schedulability check:

1/3 + 4/6 = 0.33 + 0.67 = 1.0



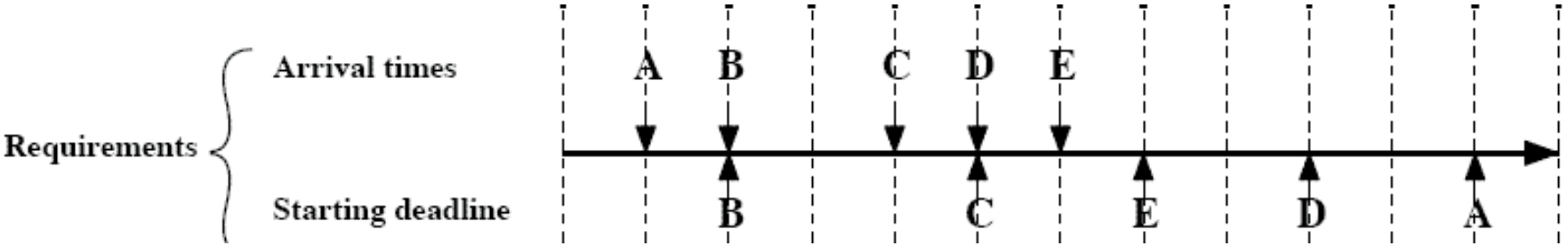
Unlike RMS, Only those task sets which pass the schedulability test are schedulable under EDF

RMS vs. EDF/LLF

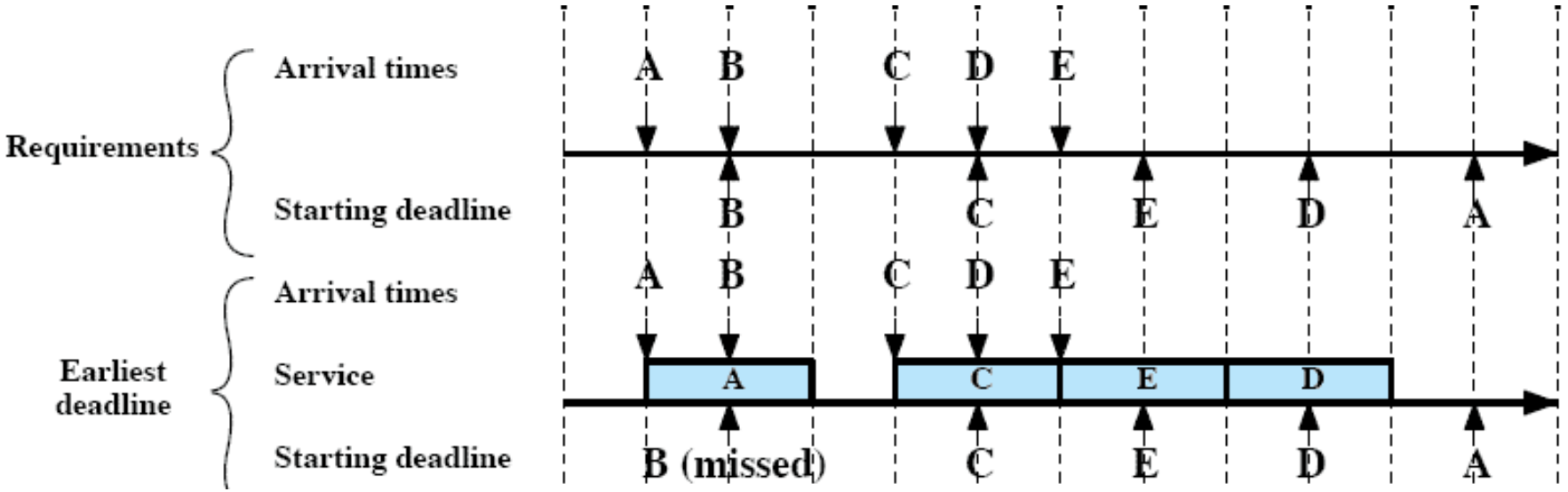
- RMS is an optimal preemptive scheduling algorithm with fixed priorities.
- EDF/LLF is an optimal preemptive scheduling algorithm with dynamic priorities.
- RMS schedulability properties can be analyzed; rich theory exists and it is widely used in practice.
- EDF/LLF offers higher schedulability than RMS, but it is more difficult to implement.

Aperiodic Scheduling: Five Aperiodic Tasks (Non-preemptive)

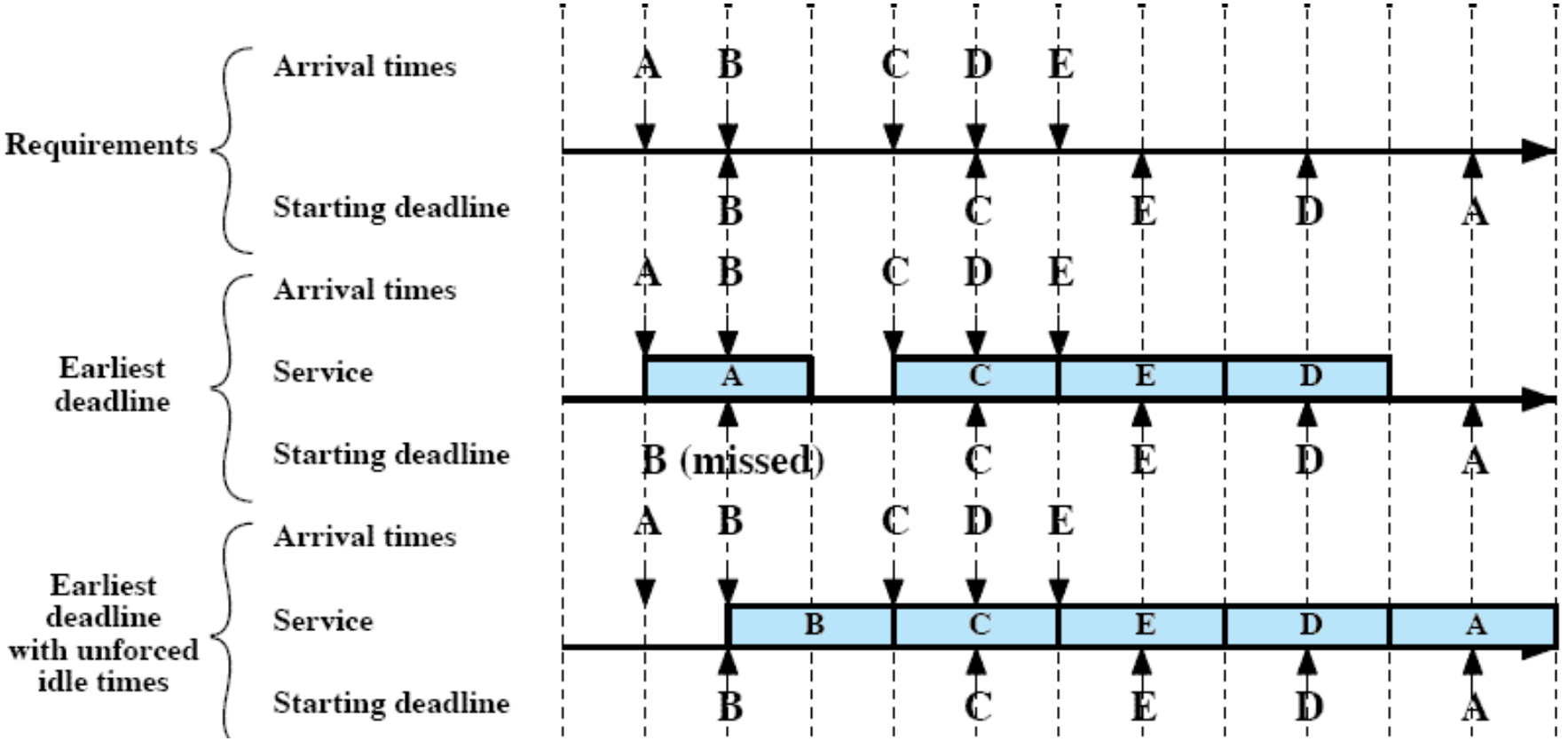
Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70



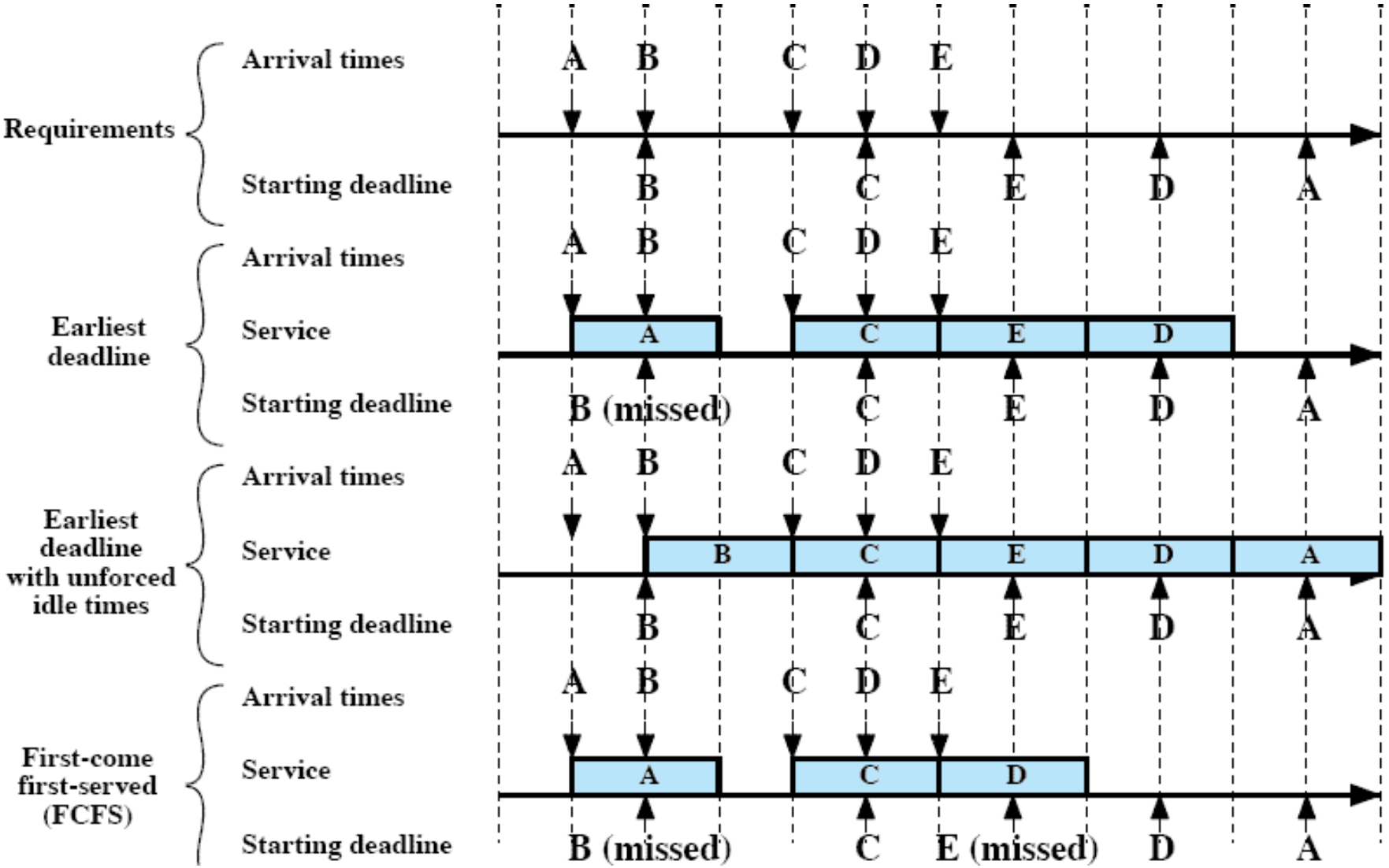
Aperiodic Scheduling



Aperiodic Scheduling



Aperiodic Scheduling



Homework: Scheduling

- Examine the scheduling policies in Linux and Windows and compare these policies (including policies for real-time scheduling).
  - Write a report, describe and compare the key scheduling policies in Linux and Windows.
  - How do they manage processes/threads?
  - What's the complexity (O) of their respective schedulers?
  - How do they handle priority inversion?

- Submission is not required

Summary

- Multiprocessor scheduling
- Real-time scheduling

- Next lecture
  - Synchronization (I) - Basics