Scheduling (Part III)

Professor Travis Peters

CSCI 460 Operating Systems

Fall 2019

Some slides & figures adapted from Stallings instructor resources.

Some slides adapted from Adam Bates's F'18 CS423 course @ UIUC
https://courses.engr.illinois.edu/cs423/sp2018/schedule.html

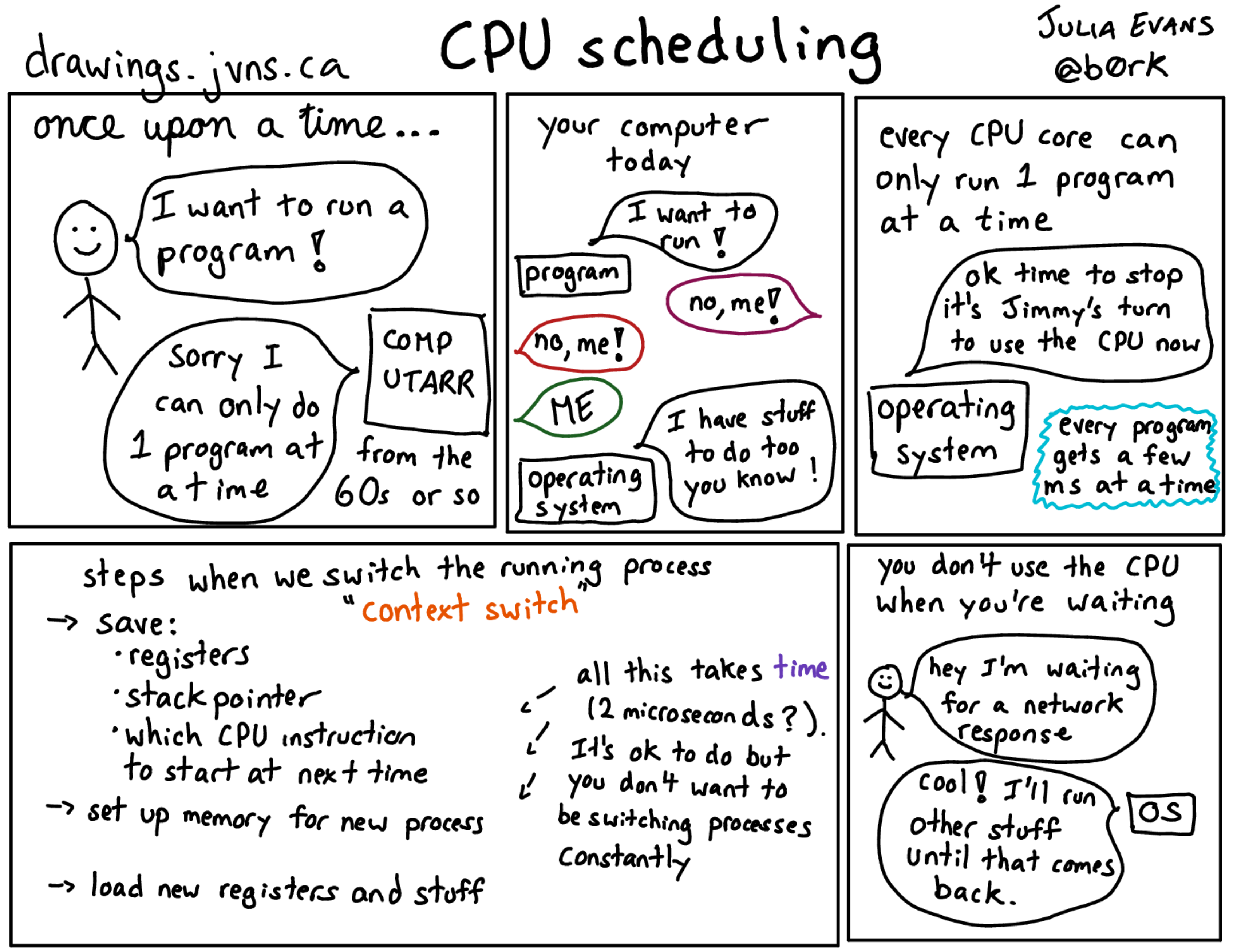
Goals for Today

Learning Objectives

- Discuss the key design issues in multiprocessor scheduling + some of the key approaches to scheduling
- Understand the requirements imposed by real-time scheduling
- Look at some of the scheduling methods used in Linux & UNIX

Announcements

- Programming Assignment was Due Wednesday @10pm!
  →*Please upload as a **zipped folder**… upload issue was fixed ;)*
- Project deadlines posted
- **REMINDER:** You have 1 free late pass… *(see the syllabus)*
- **NOTE:** If using GitHub (**good!**), but
  make sure code isn't public (**BAD!**)



https://drawings.jvns.ca/scheduling/

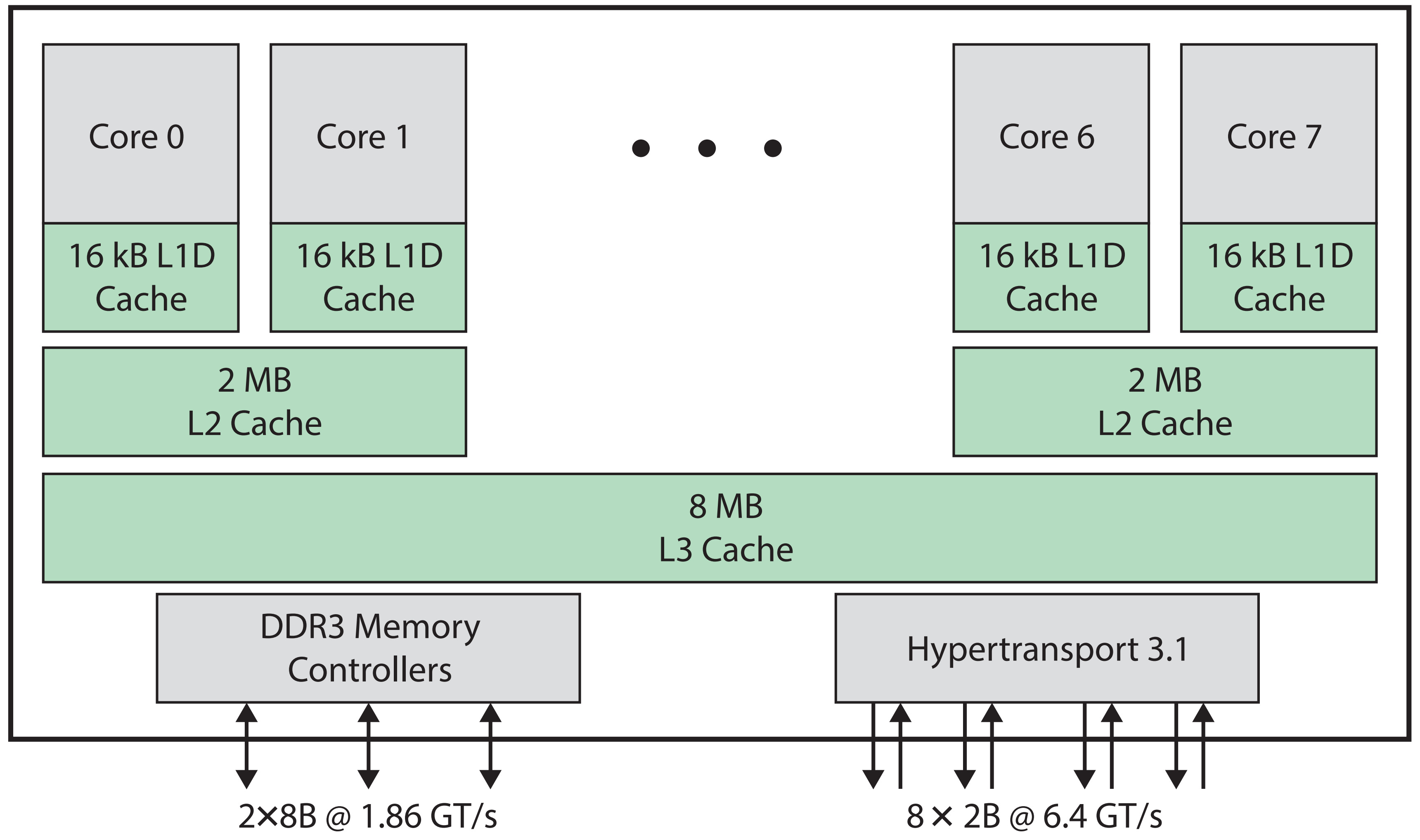
Multiprocessor and Multicore Scheduling (Summary)

- See the text for discussion of various considerations for
  - Synchronization
  - Parallelism
  - etc.
- At the end of the day, its all about assigning process***es*** to process***ors***, and being cognizant of the trade-offs
  - Static Assignment — processes are assigned to a processor-specific queue upon initialization; assignments don't change
  - Dynamic Load Balancing — keep workload distributed (as equally as possible) across all processors
  - Master/Slave *(simple)* vs. Peer *(more complex)* vs. Hybrids

Multiprocessor and Multicore Scheduling (Summary)

Punchline(s):

- more processors → less emphasis on the efficiency of your scheduling algorithm.
- ***Load Sharing*** *(processors pull from global queue of ready-to-run processes)* is probably the most common approach to scheduling on multiprocessor systems.
  - Recall Concurrency—MP scheduling comes w/ all the same advantages and disadvantages… ;)
  - To do this, you ***must enforce*** mutual exclusion
- Try to do things s.t. cost of process switching is avoided as much as possible
- Try to limit the number of threads in an application to no more than the number of processors available on the system
- Most (e.g., Linux, Windows) hold
  multiprocessor scheduling ≈ multicore scheduling



Real-Time Processes, Scheduling, and OSs

- A ***real-time process*** must produce nearly instantaneous output based on new inputs.
  - Each arriving input item is subject to a deadline.
  - ***Ex:*** Streaming of audio or video, control of robots.
  - ***hard real-time tasks*** vs. ***soft real-time tasks***
- A ***period*** is ***a time interval*** (typically in ms or μs) within which each input item must be processed.
  - The end of each period is the ***implicit deadline*** for processing the current item.
- Characteristics of RTOSs
  - Deterministic, Responsiveness, User Control, Reliability, Fail-Soft Operation
  - Most contemporary RTOSs don't deal directly with "deadlines."
    Instead, they try to be as responsive as possible

Real-Time Scheduling

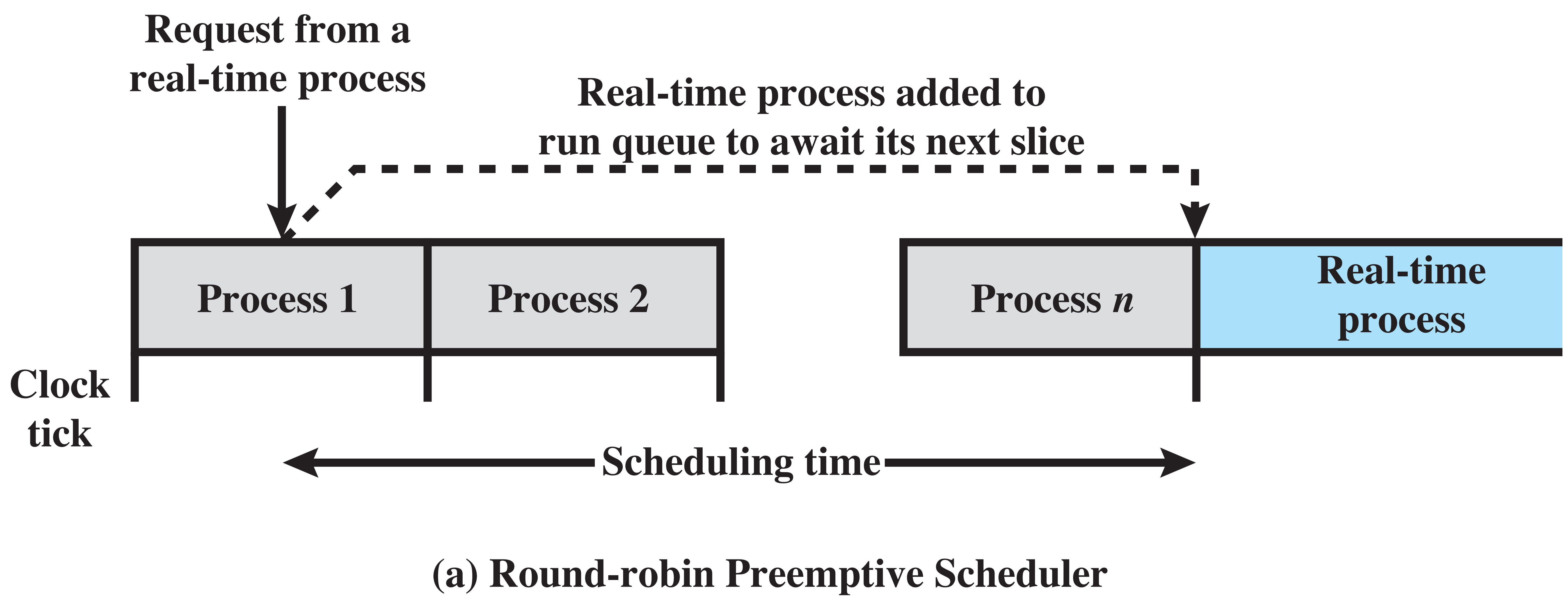


Figure 10.4 Scheduling of Real-Time Process

Real-Time Scheduling

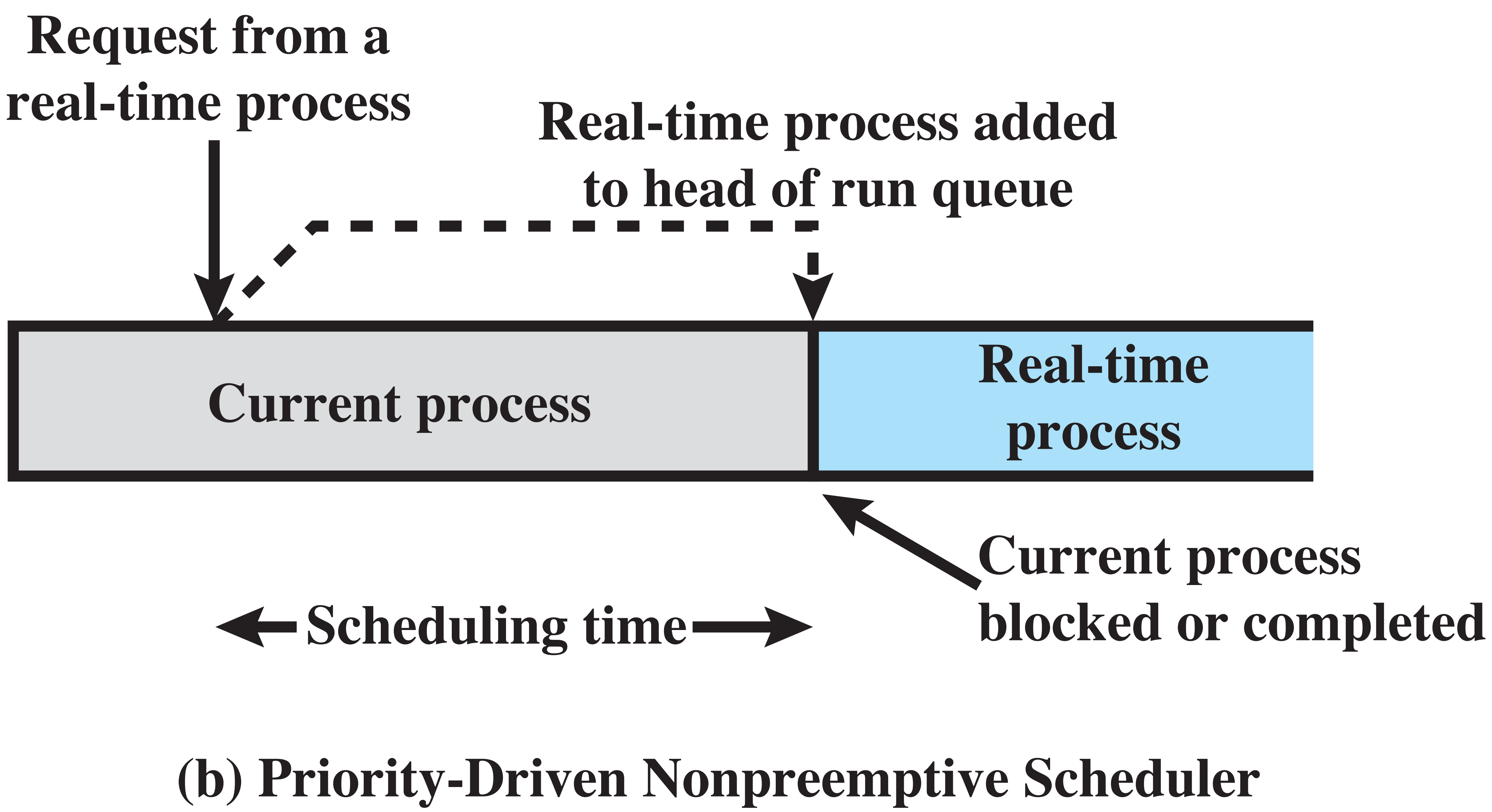
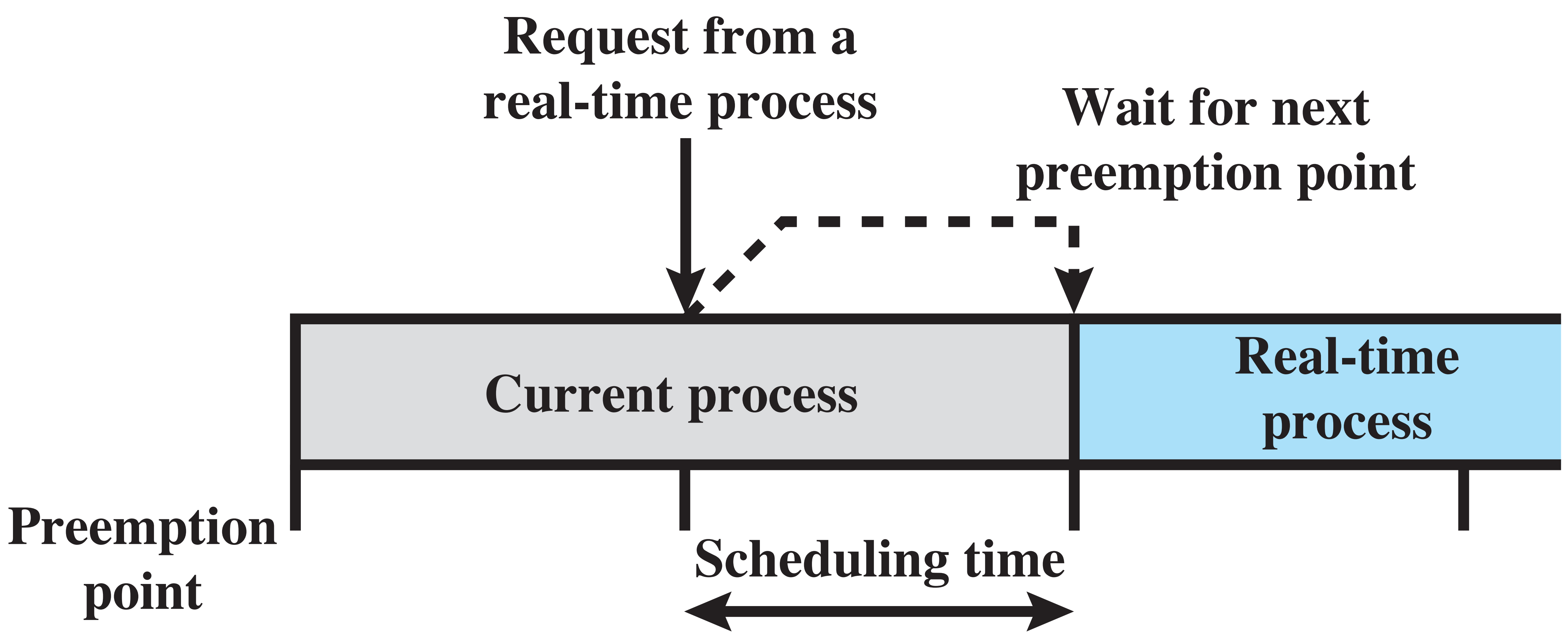


Figure 10.4 Scheduling of Real-Time Process

Real-Time Scheduling



(c) Priority-Driven Preemptive Scheduler on Preemption Points

Figure 10.4 Scheduling of Real-Time Process

Real-Time Scheduling

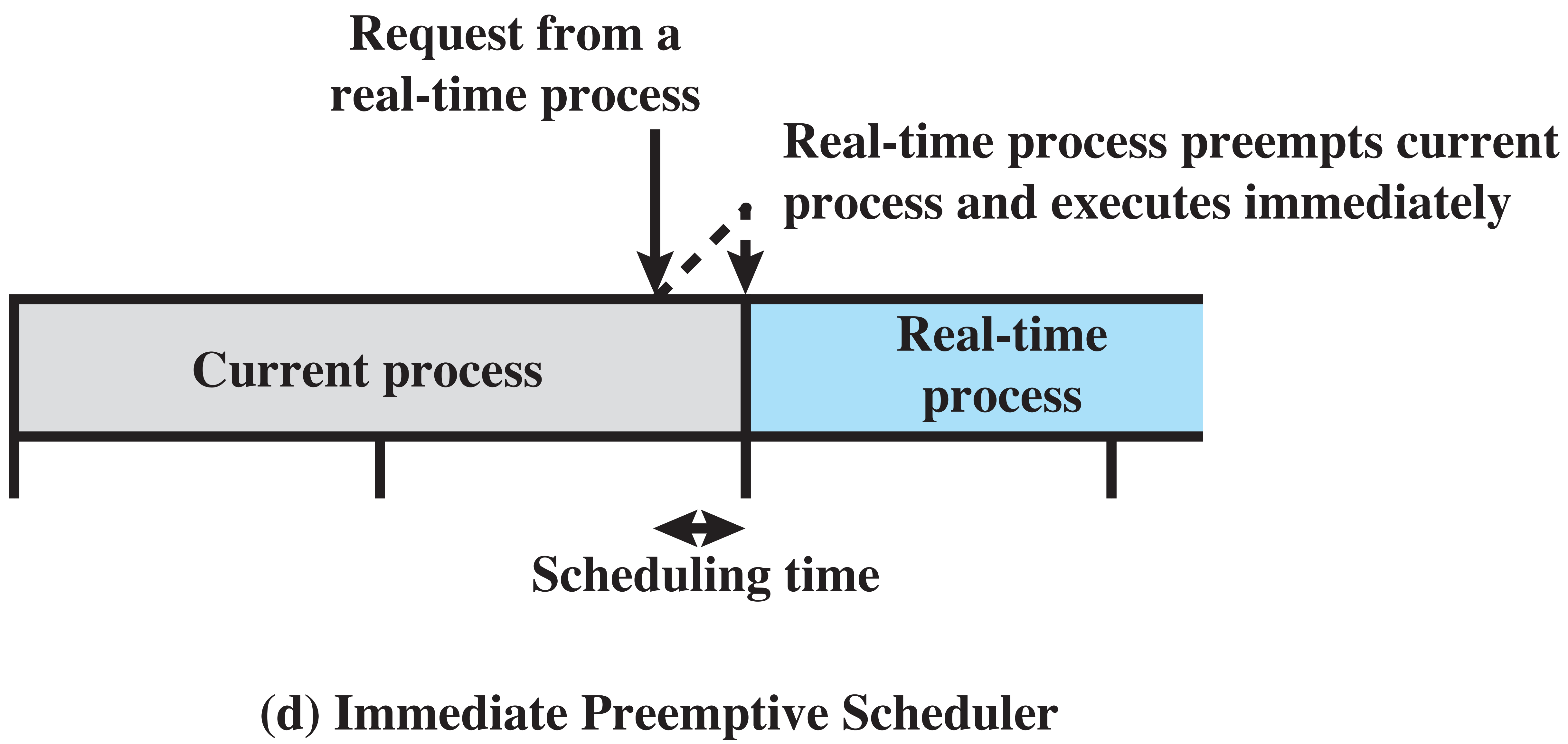
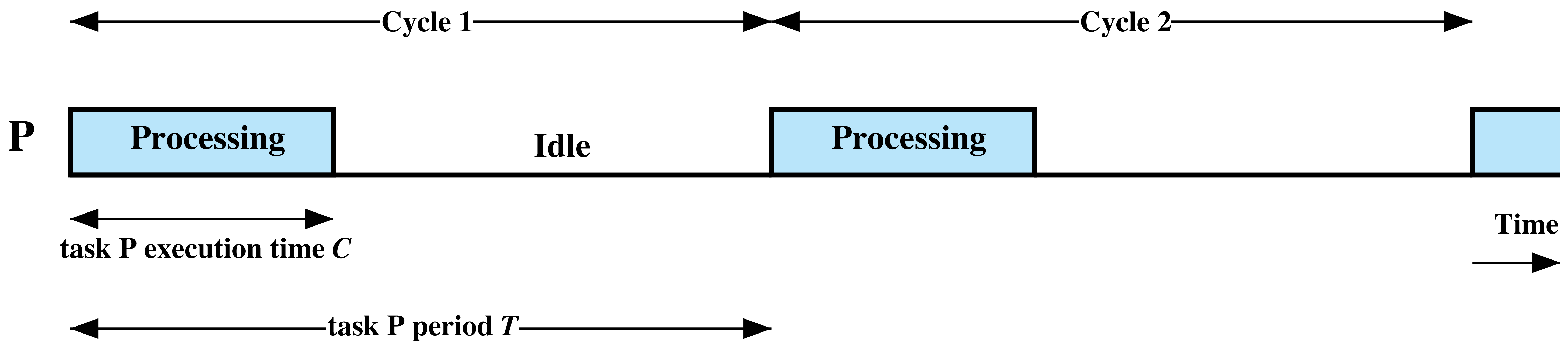


Figure 10.4 Scheduling of Real-Time Process

The ***Rate Monolithic (RM)*** Scheduling Algorithm

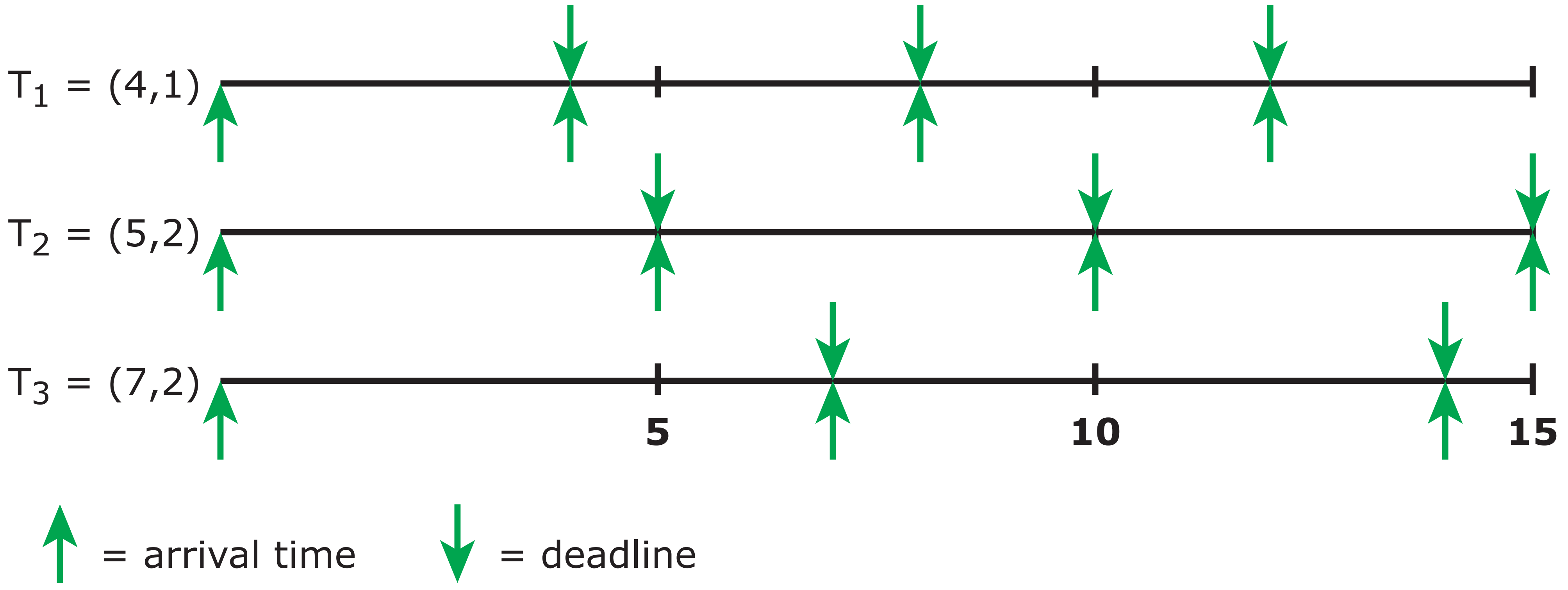
- Schedules processes according to the period.
- The shorter the period, the higher the priority.
- RM is preemptive.

  (But only higher priority processes can preempt lower priority processes)

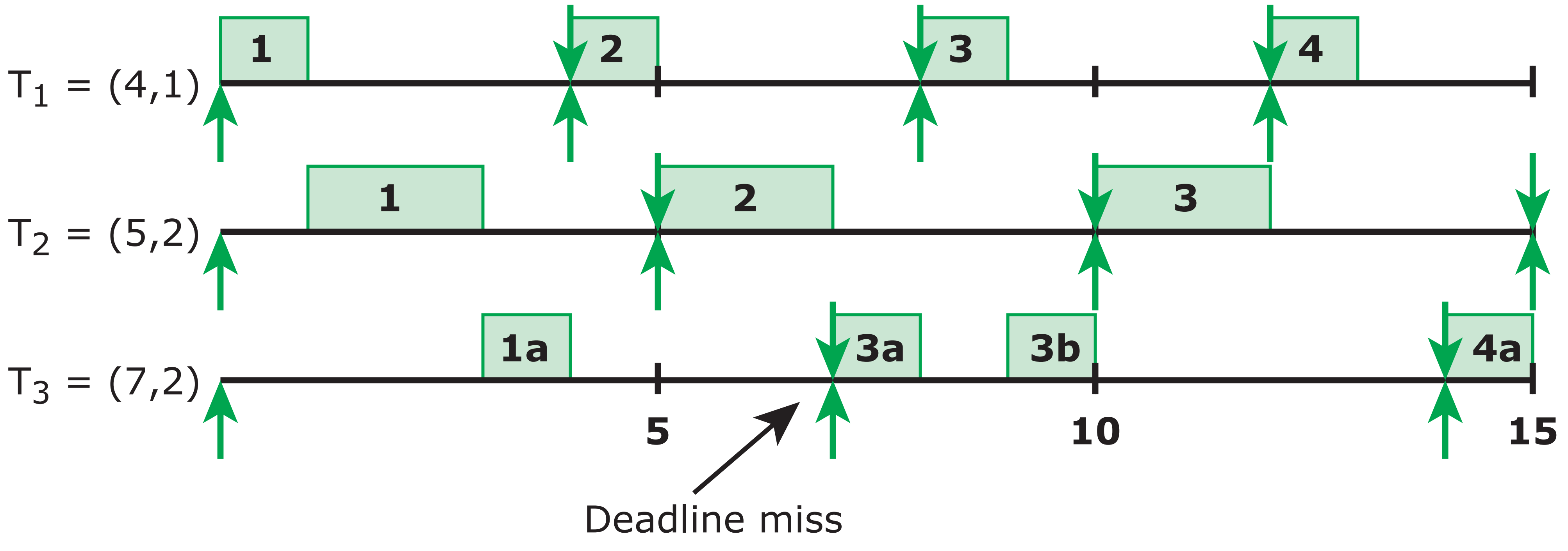


The ***Rate Monolithic (RM)*** Scheduling Algorithm

- Schedules processes according to the period.
- The shorter the period, the higher the priority.
- RM is preemptive.
  (But only higher priority processes can preempt lower priority processes)



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



(b) Scheduling results

Figure 10.8 Rate Monotonic Scheduling Example

Priorities & Scheduling

- **Priority Inversion** — when a high priority task has to wait on a lower priority task
- **Priority Inheritance** — lower priority task inherits the priority of any higher-priority task pending on a shared resource
- **Priority Ceiling** — associate priorities with resources

Path Finder Fail…

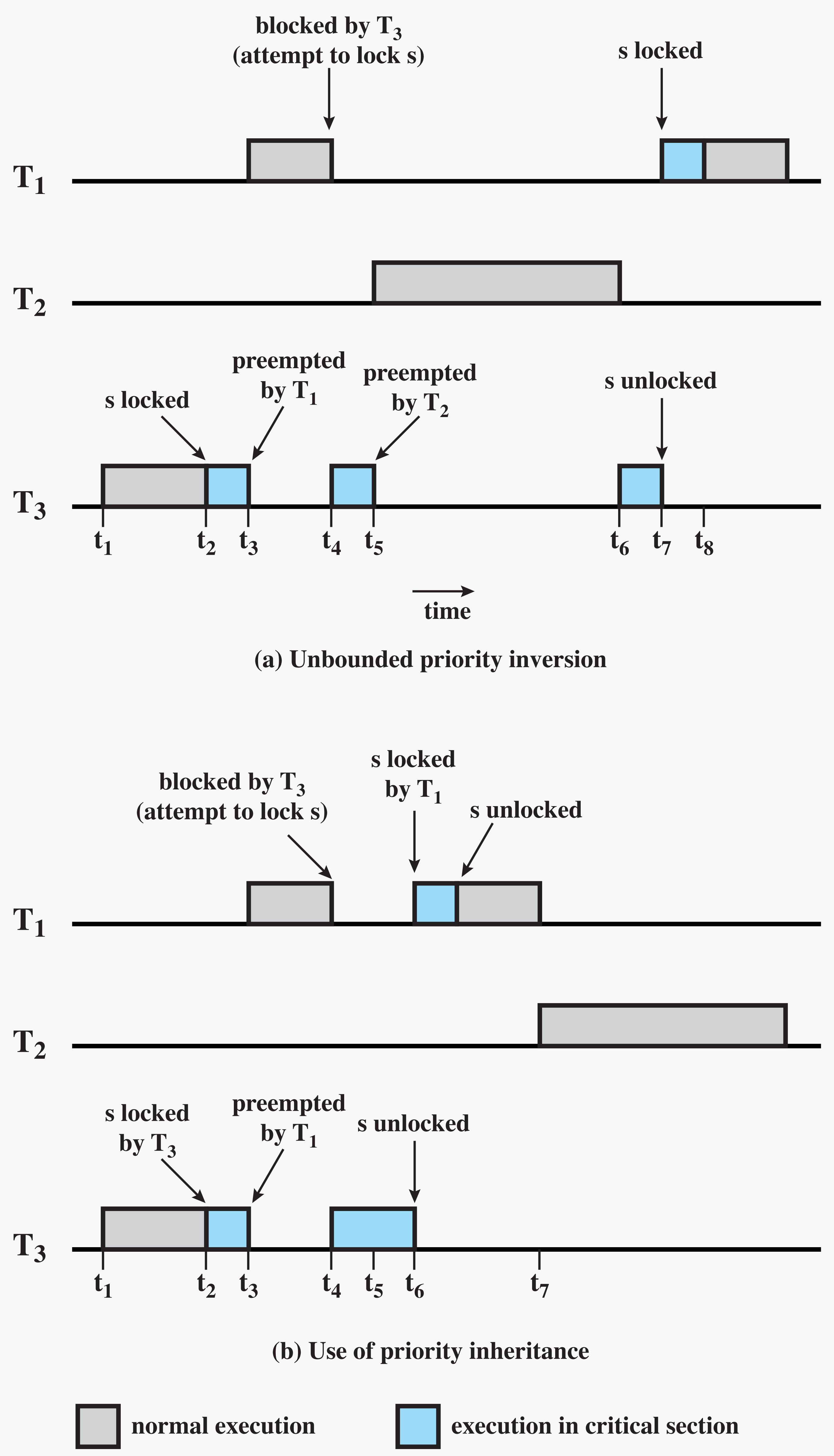


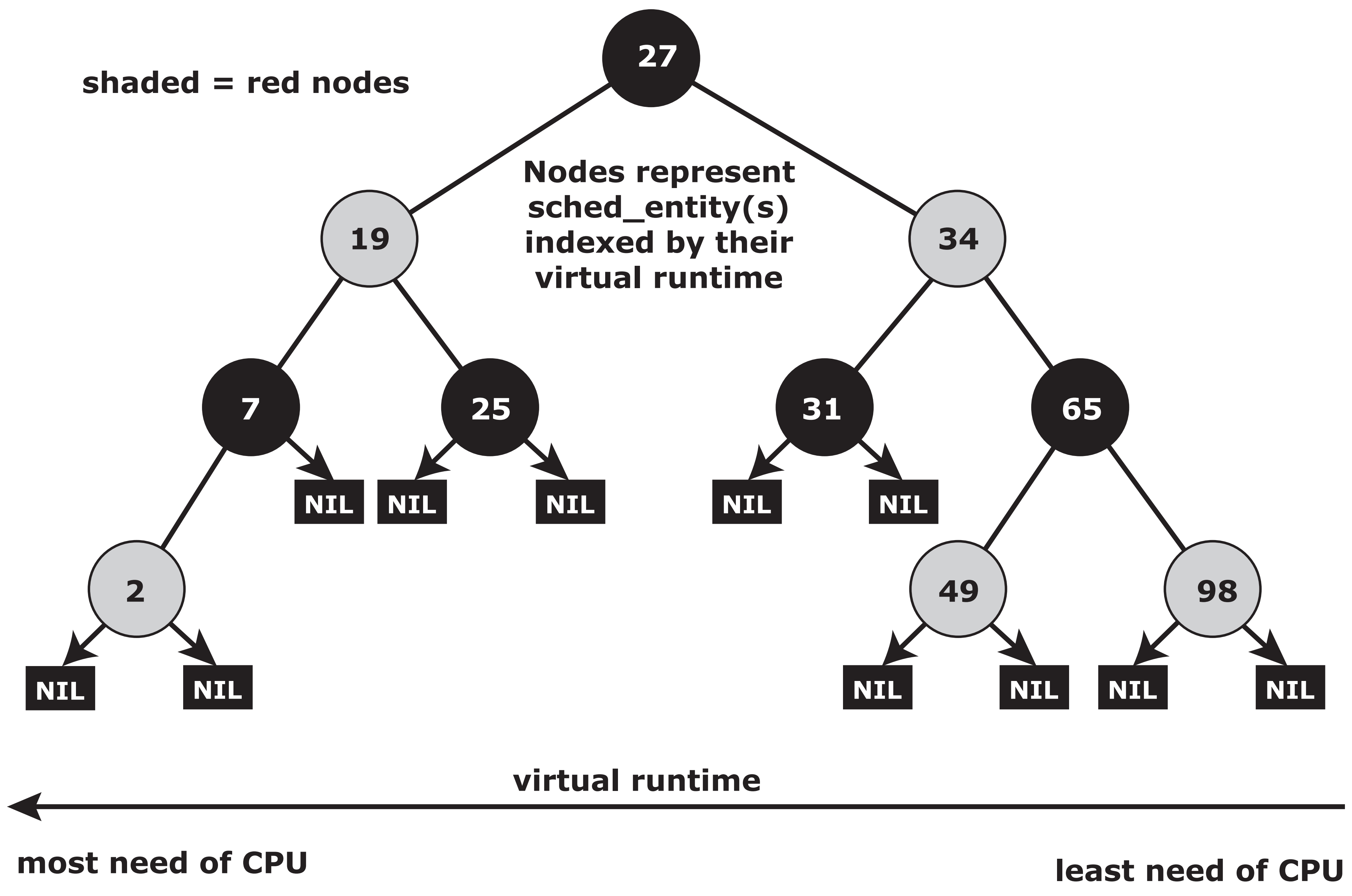
Figure 10.9 Priority Inversion

Linux Scheduling

- Scheduling Classes
  - **SCHED_FIFO** (limited preemption) & **SCHED_RR** (time-sliced) — (0-99)
  - **SCHED_NORMAL** (100-139)

    →lower value == higher priority

    →non-RT thread only executes if no RT threads are ready to execute



- Linux 2.6+ → **O(1) Scheduler**
  - Named so because time to select & run a process takes constant time.
  - Complex and not good to run in the kernel….

- Linux 2.6.23+ → **Completely Fair Scheduler (CFS)**
  - Maintain ***virtual runtime*** value for each task (normalized amt. of time spent executing so far)
  - ***Sleeper Fairness*** (i.e., ensure that processes that wait on I/O get fair access to processor)
  - Use Red-Black Tree to order runnable tasks (efficient inserts/deletes, and searches)