CSCE 4114

(Real Time) Operating Systems

David Andrews

dandrews@uark.edu

Operating Systems

- Originally developed to ease sharing of resources between users and foster portability
  - Early programming involved developing program specifically for a machine
  - Programs had to be re-written for each new machine.
- An OS is a "Virtual Machine"
  - Machine capabilities accessed through API's
  - User's code to API not machine specific registers, protocols, addresses, etc.
  - Specific implementations of API's provided through libraries
    - Libraries linked into source code

Operating System Objectives

- **Multi-user** - A multi-user operating system enables multiple users to use the same computer at the same time and different times.

Operating System Objectives

- **Multi-user** - A multi-user operating system enables multiple users to use the same computer at the same time and different times.
- **Multiprocessing** - An operating system capable of supporting and utilizing more than one computer processor.

Operating System Objectives

- **Multi-user** - A multi-user operating system enables multiple users to use the same computer at the same time and different times.
- **Multiprocessing** - An operating system capable of supporting and utilizing more than one computer processor.
- **Multitasking** - An operating system that is capable of allowing multiple software processes to run at the same time.

Operating System Objectives

- **Multi-user** - A multi-user operating system enables multiple users to use the same computer at the same time and different times.
- **Multiprocessing** - An operating system capable of supporting and utilizing more than one computer processor.
- **Multitasking** - An operating system that is capable of allowing multiple software processes to run at the same time.
- **Multithreading** - Operating systems that allow different parts of a software program to run concurrently.

Operating System Objectives

- **Multi-user** - A multi-user operating system enables multiple users to use the same computer at the same time and different times.
- **Multiprocessing** - An operating system capable of supporting and utilizing more than one computer processor.
- **Multitasking** - An operating system that is capable of allowing multiple software processes to run at the same time.
- **Multithreading** - Operating systems that allow different parts of a software program to run concurrently.

↑ Our Interest in this class

Real Time Operating Systems

- Real-time OS: a multitasking[/multithreading] operating system for executing real-time applications.
  - Use specialized scheduling algorithms to deliver deterministic behavior.
  - Latency Considerations instead of throughput drives design. Sometimes miss-interpreted as “fast”.
  - Typically modeled as event-driven: responds to change’s in external environment such as input sensors
  - Event-driven system switches between tasks based on their priorities or external events while time-sharing operating systems switch tasks based on clock interrupts.

From: http://en.wikipedia.org/wiki/Operating_system

Embedded Operating Systems

- Embedded OS: Designed to operate on small machines like PDAs with less autonomy. They are able to operate with a limited number of resources. They are very compact and extremely efficient by design.
  - Small footprints
  - Scaled back capabilities
    - Virtual Memory Support

Operating System Services

- Program Management
  - Scheduling
- Timer Services
  - Date/Time
  - Watchdog Timers
- Synchronization/Communications
- File Services
- Networking
- Security

Scheduling

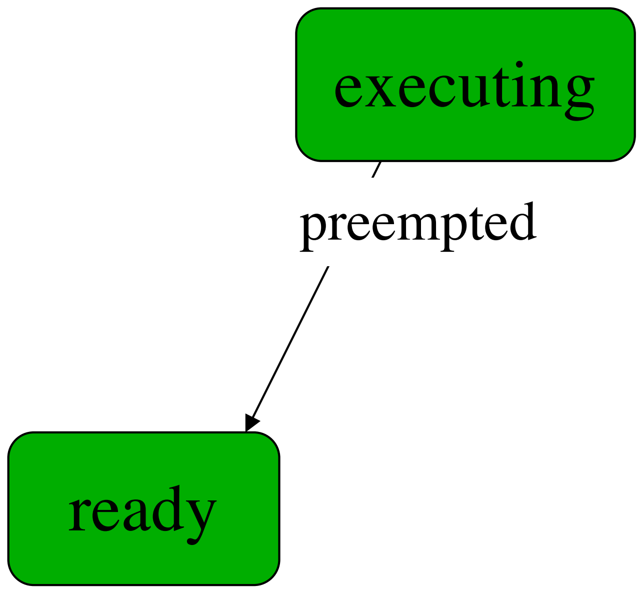
Task State

- 1 – has CPU

Scheduling

Task State

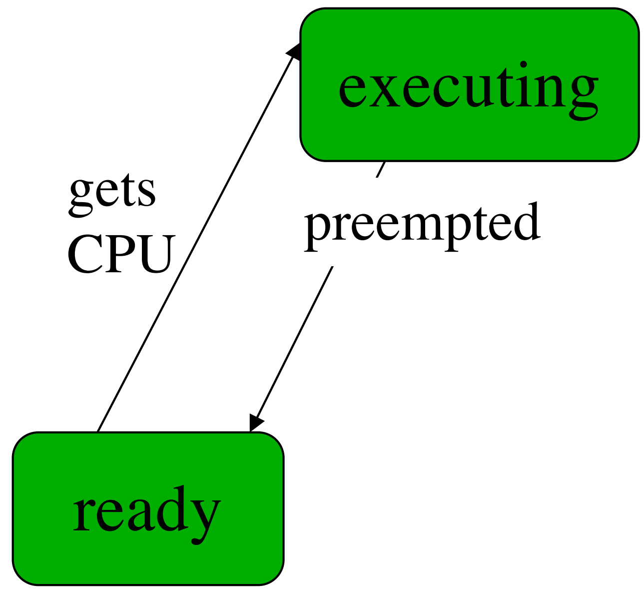
- 1 – has CPU
- No Longer Has CPU: Why ?
- Gets Preempted



Scheduling

Task State

- 1 – has CPU
- No Longer Has CPU: Why ?
- Gets Preempted
- Still can run if possible



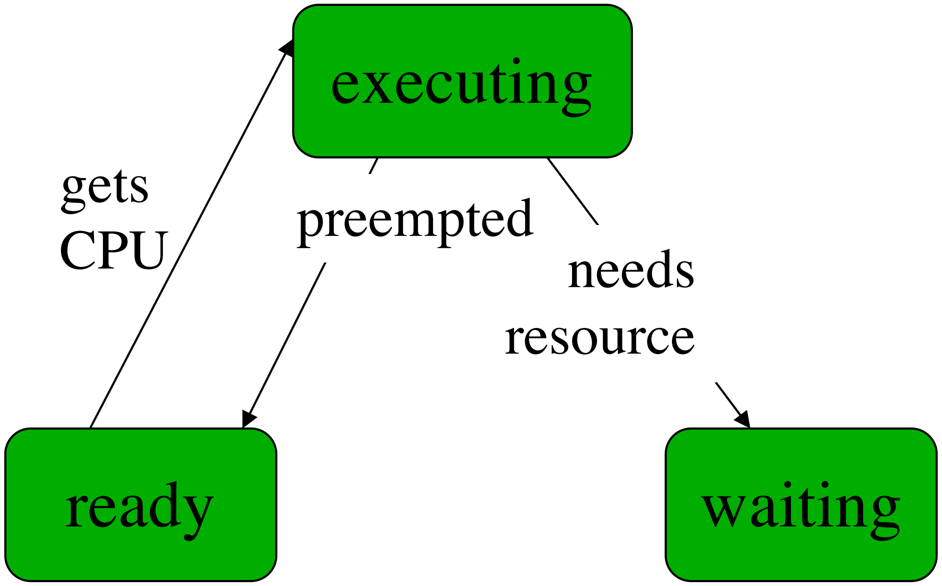
Scheduling

Task State

- 1 – has CPU
- No Longer Has CPU: Why ?
- Gets Preempted
- Needs to Wait on some resource

  -semaphore

  -I/O



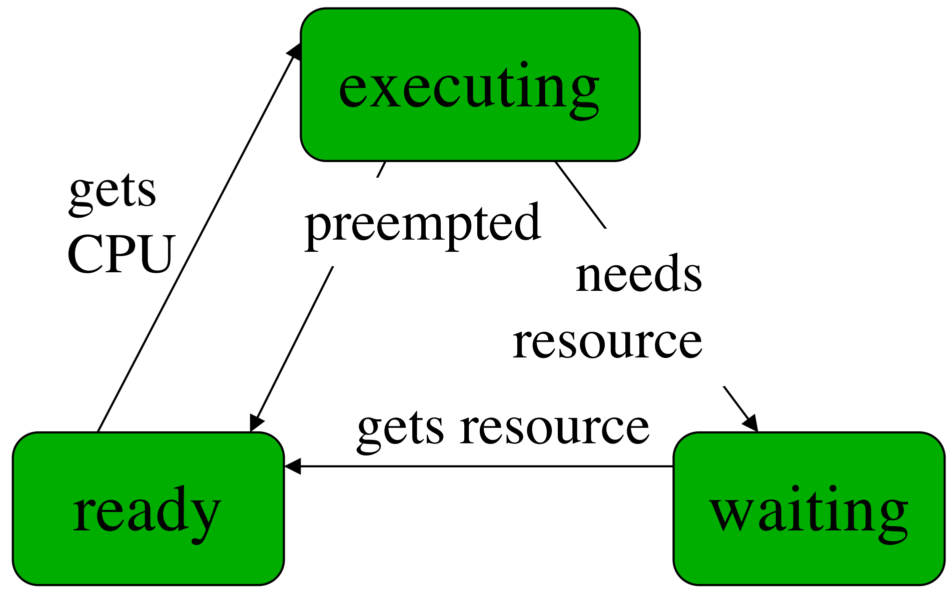
Scheduling

Task State

- 1 – has CPU
- No Longer Has CPU: Why ?
- Gets Preempted
- Needs to Wait on some resource

  -semaphore

  -I/O
- Gets Data and can now run



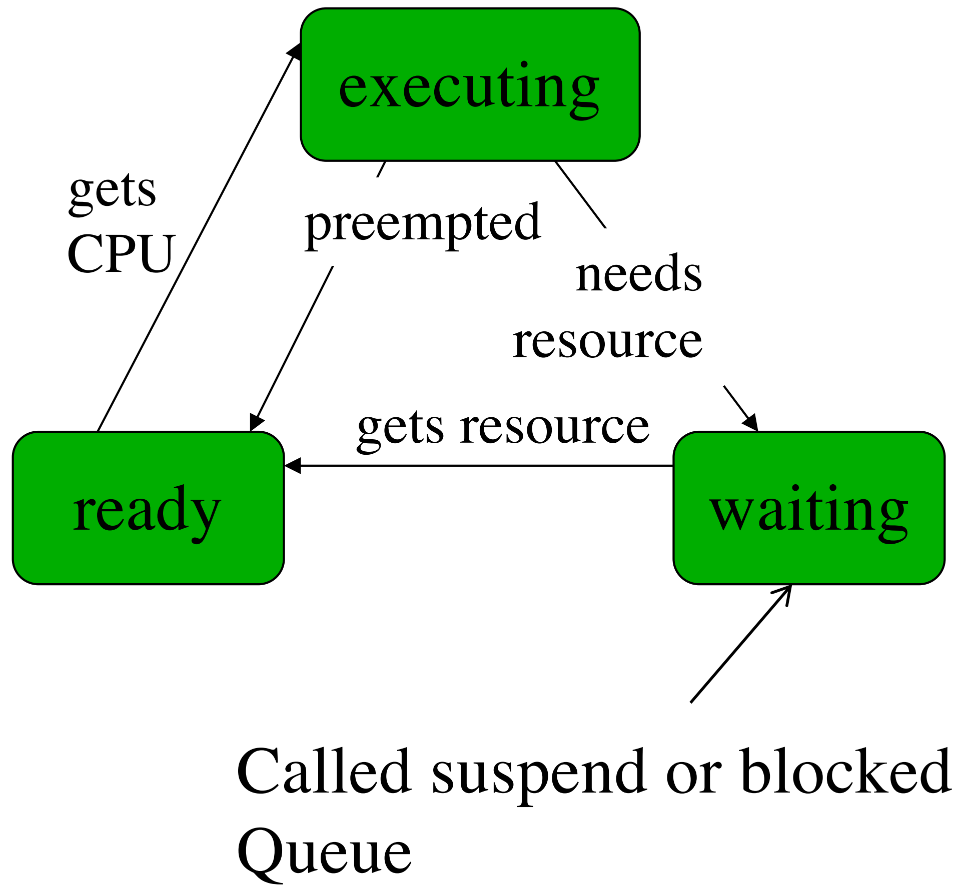
Scheduling

Task State

- 1 – has CPU
- No Longer Has CPU: Why ?
- Gets Preempted
- Needs to Wait on some resource

  -semaphore

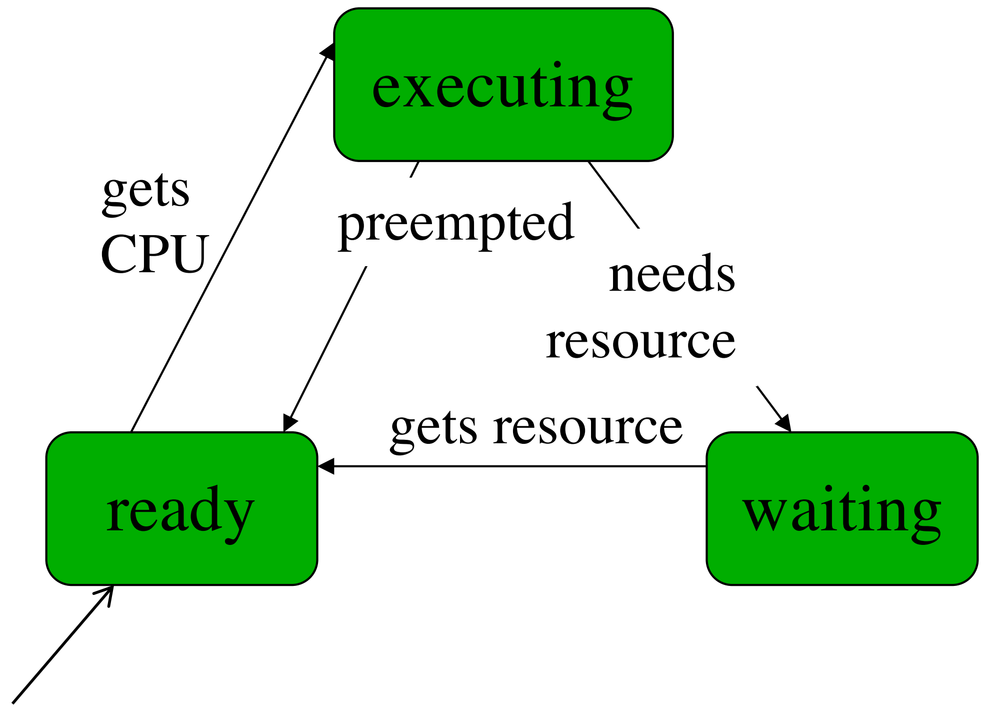
  -I/O
- Gets Data and can now run



Scheduling

Task State

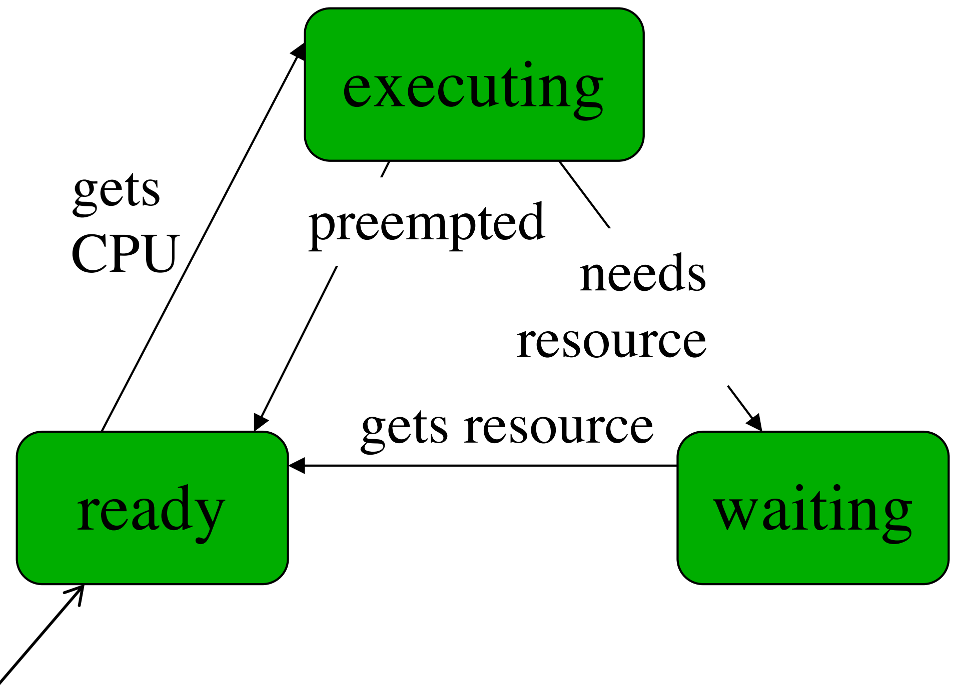
- 1 – has CPU
- No Longer Has CPU: Why ?
- Gets Preempted
- Needs to Wait on some resource
  -semaphore
  -I/O
- Gets Data and can now run



Called Ready to Run (R2R)
Or scheduler Queue

Scheduling

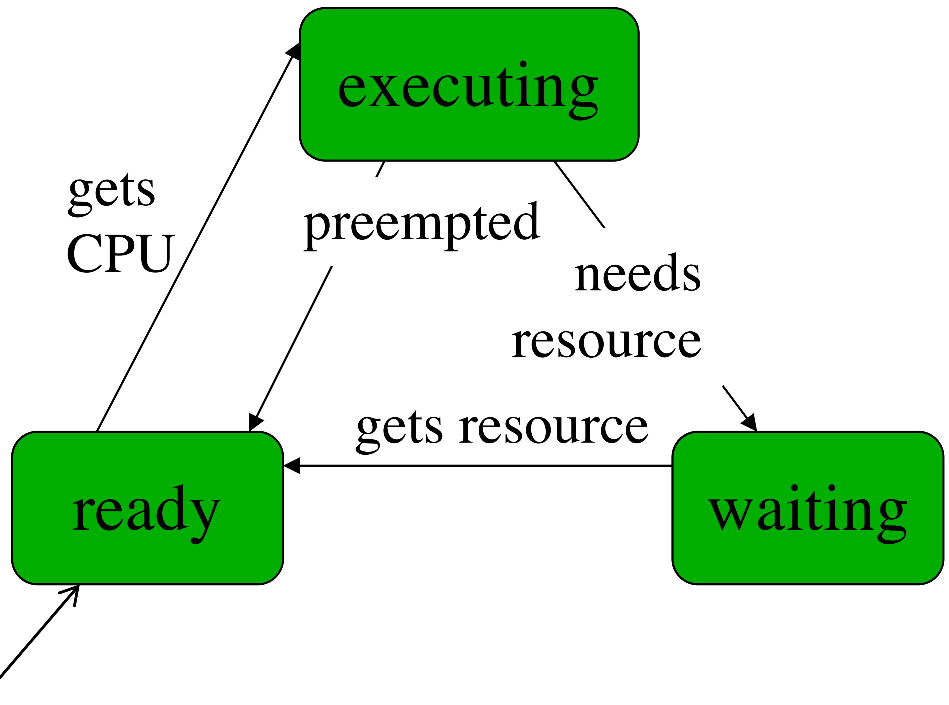
- If multiple threads in R2R Queue:
  - How does OS make scheduling decision?



Called Ready to Run (R2R)
Or scheduler Queue

Scheduling

- If multiple threads in R2R Queue:
  - How does OS make scheduling decision?
  - When does OS make scheduling decision ?

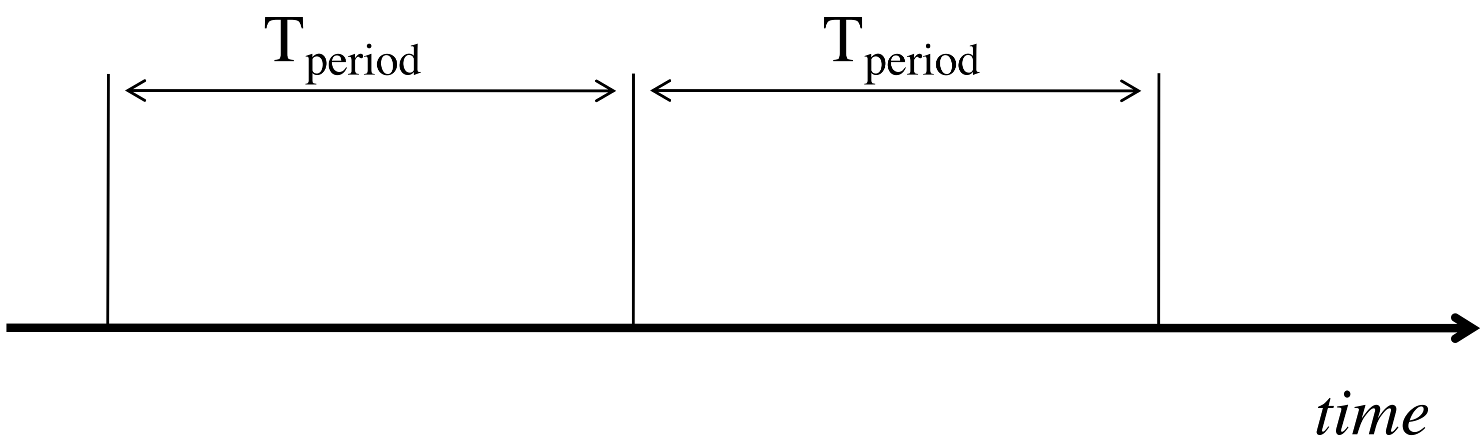


Called Ready to Run (R2R)
Or scheduler Queue

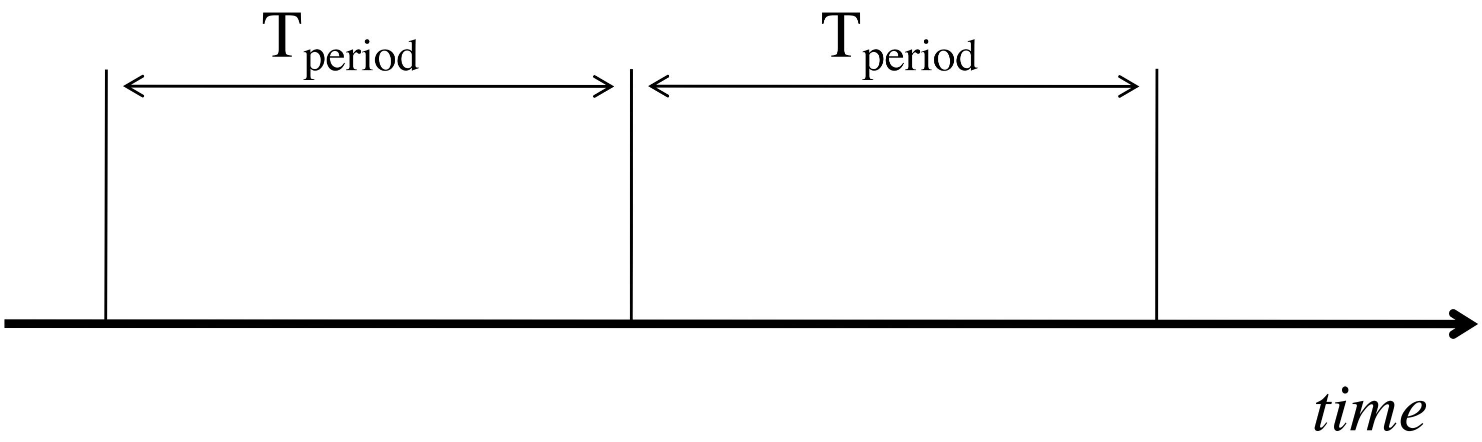
RT- Scheduling Algorithms

- Schedule multiple threads/tasks on shared resource(s) such that they all meet their deadlines…….
- Need to Know…
  - Execution time of each task
    - Study of Worst Case Execution Time
  - Deadline of each task
    - When all must be completed
  - When can task begin to execute
    - Periodic is simplest (aperiodic much more difficult)

A little theory….

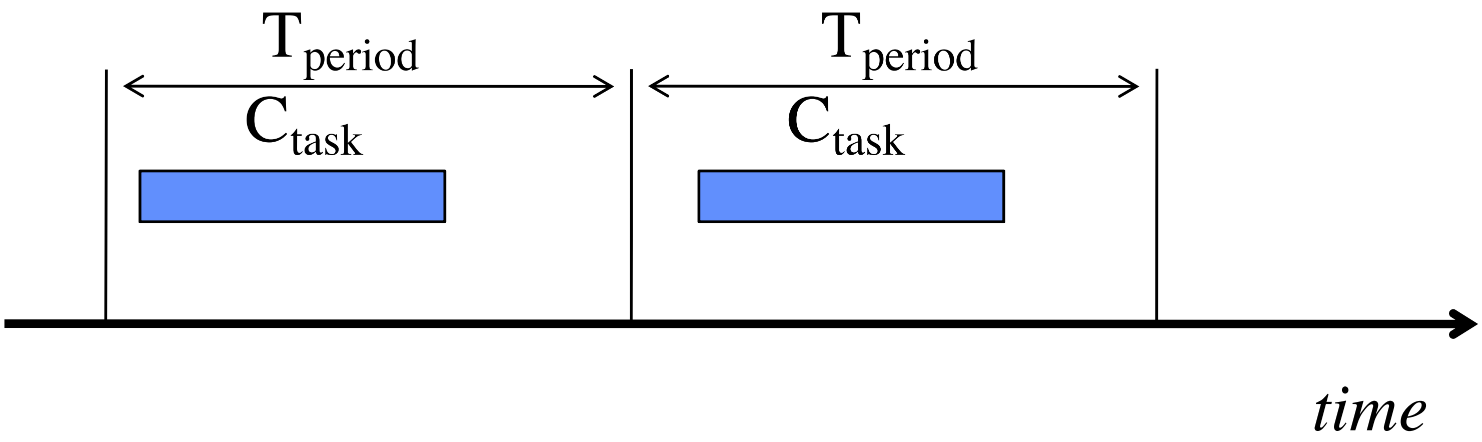


A little theory….

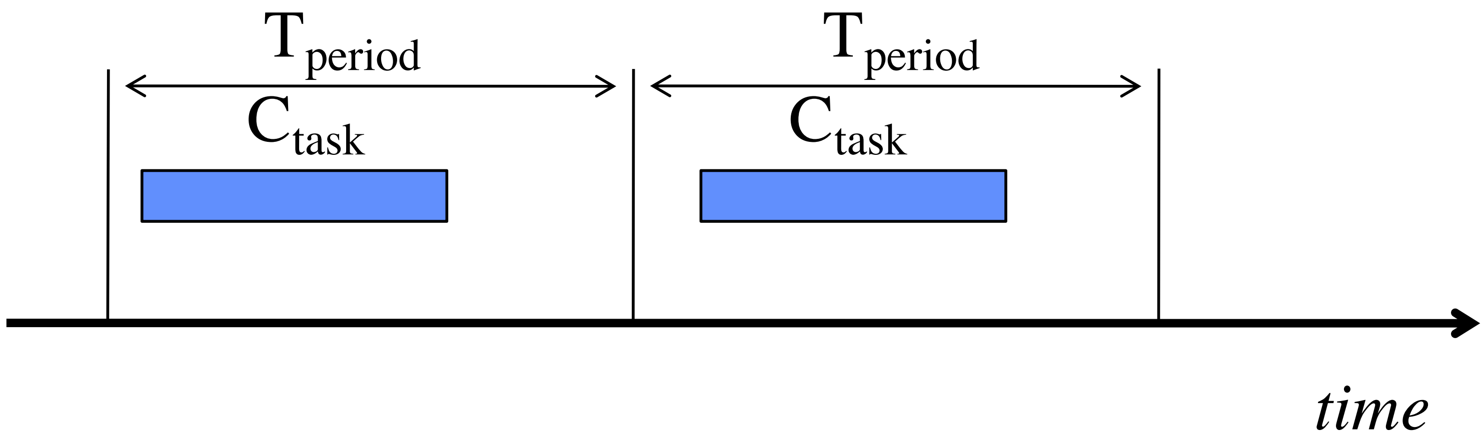


Task will "start" at the beginning of it's period

A little theory....

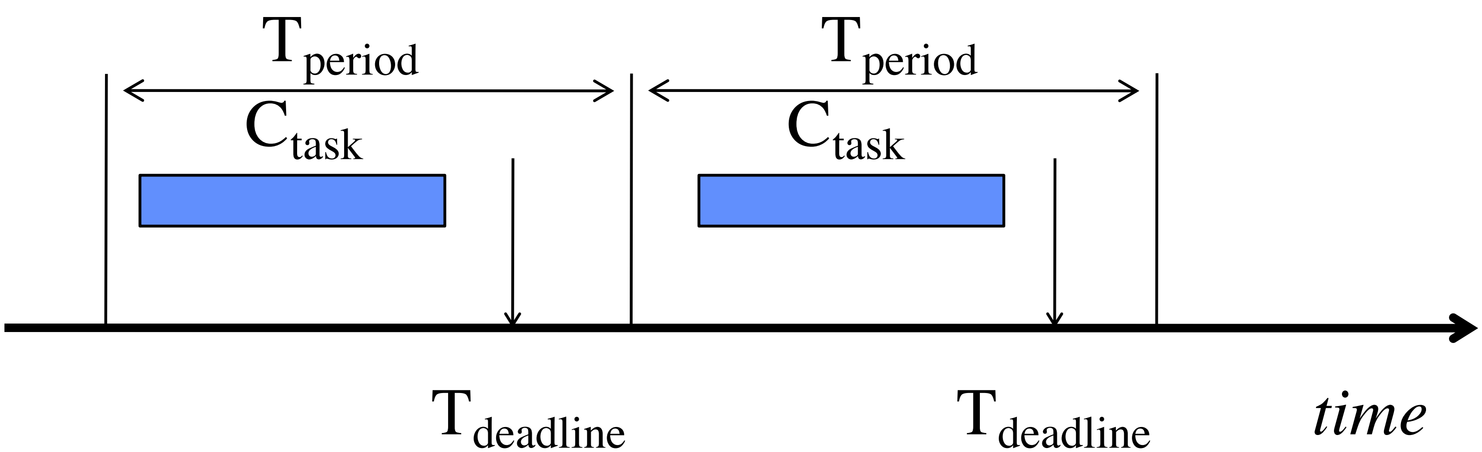


A little theory….

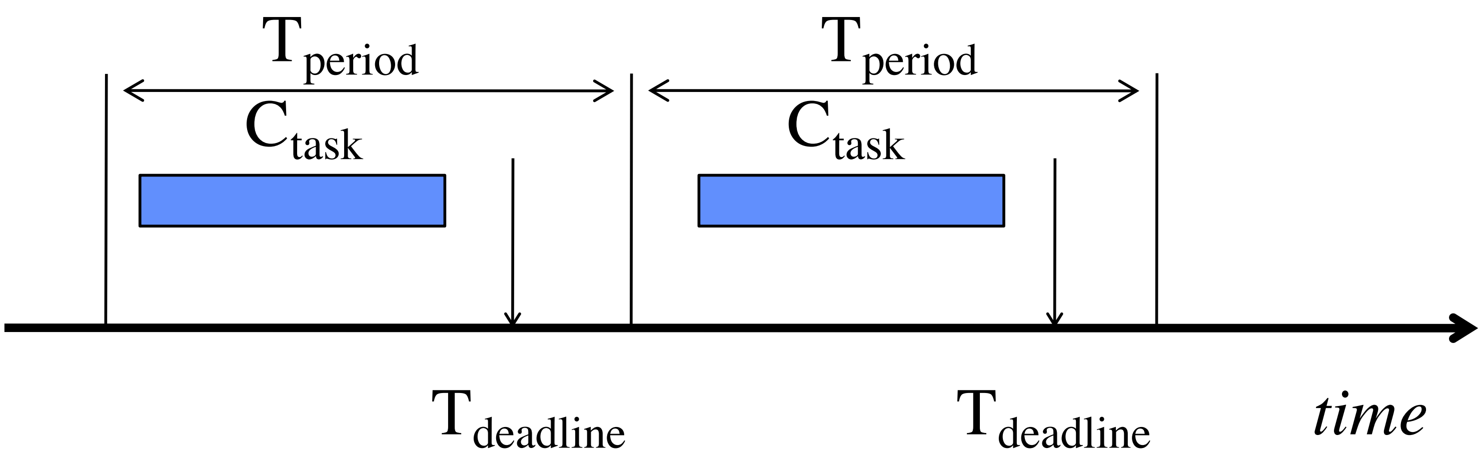


C_{task} :A Task's worst case execution time

A little theory….

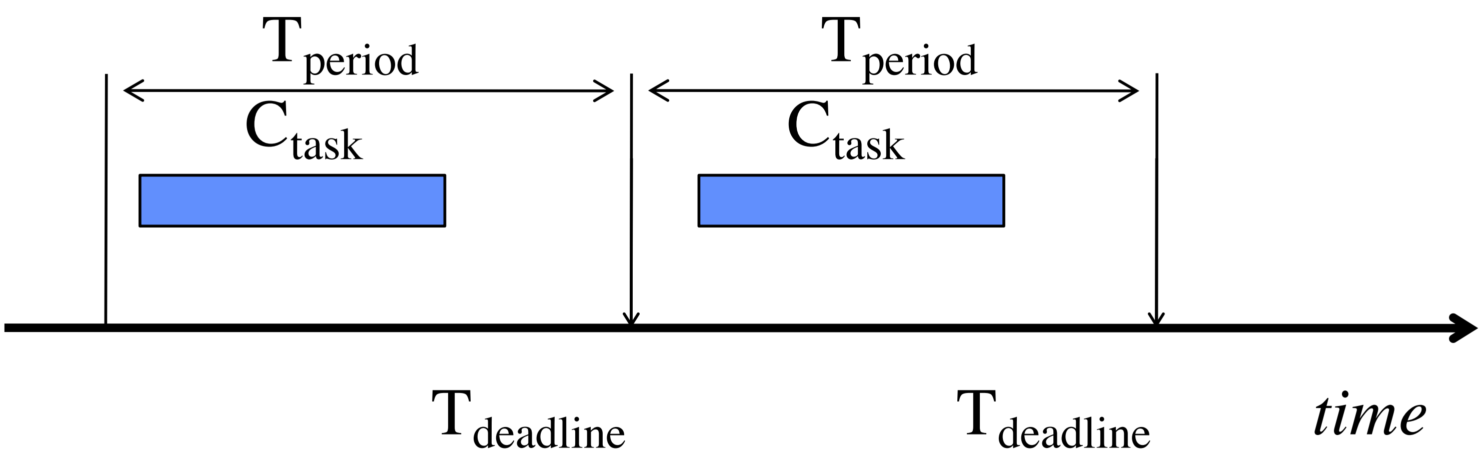


A little theory….



Task must complete by it's deadline $T_{deadline}$

A little theory....



Task must complete by it's deadline $T_{deadline}$

Simplify to make $T_{deadline}$ = end of period

RT- Scheduling Algorithms

- Priority based scheduling
  - static priority;
    - Priority set during design time
    - Does not change during system operation
  - Dynamic priorities
    - Change as system runs
- Preemption/Non-Preemption
  - Preemption: Task on CPU can get booted by higher Priority Task ready to run
  - Non-preemption: Task on CPUs keeping executing even if higher priority task ready to run

Priority-driven scheduling example

- Rules:
  - each process has a fixed priority (1 highest);
  - highest-priority ready process gets CPU;
  - process continues until done or wait state.
- Processes
  - P1: priority 1, execution time 10
  - P2: priority 2, execution time 30
  - P3: priority 3, execution time 20

Priority-driven scheduling example

- Rules:
  - each process has a fixed priority (1 highest);
  - highest-priority ready process gets CPU;
  - process continues until done or wait state.
- Processes
  - P1: priority 1, execution time 10
  - P2: priority 2, execution time 30 Ready at T_0
  - P3: priority 3, execution time 20

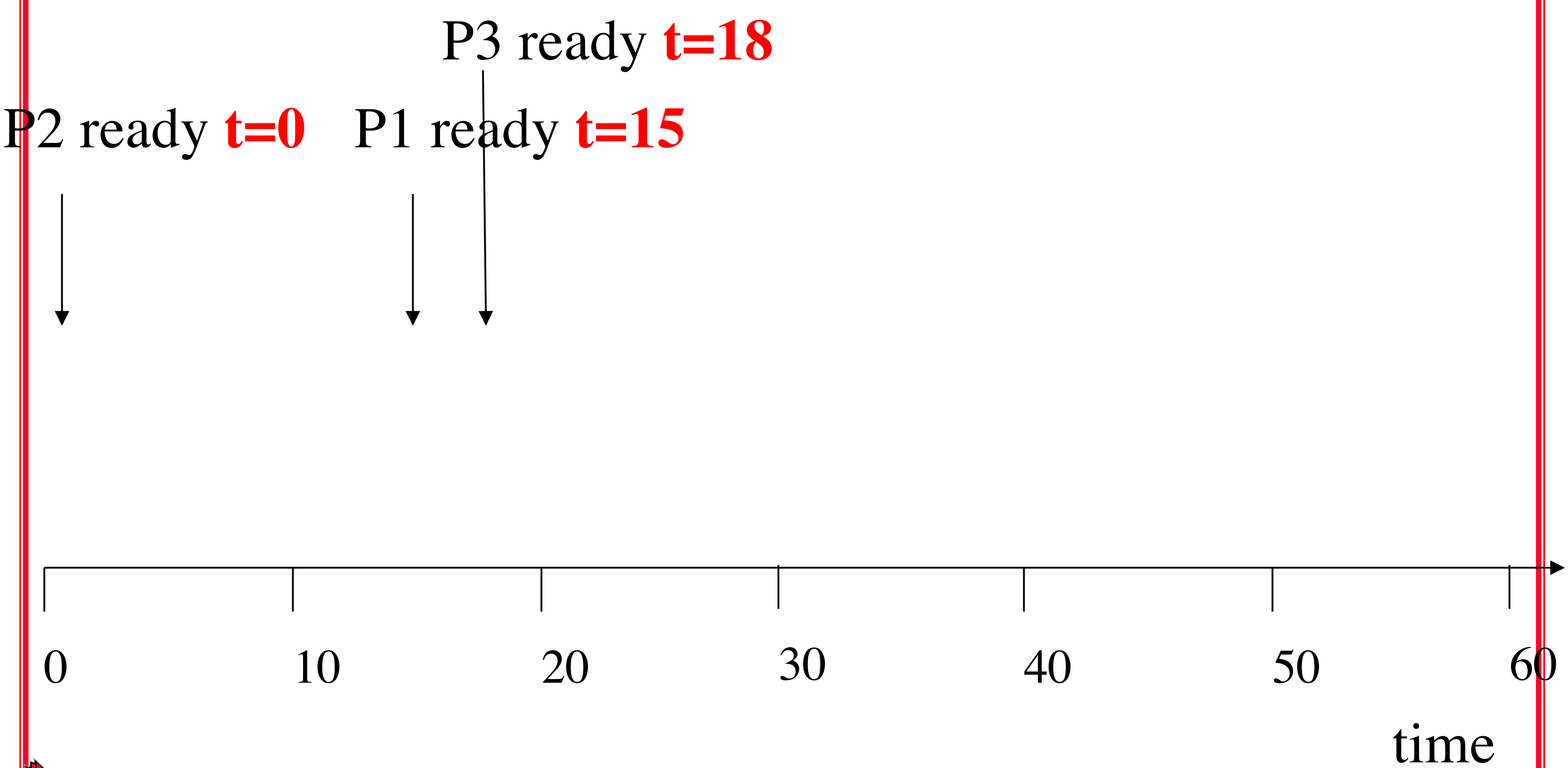
Priority-driven scheduling example

- Rules:
  - each process has a fixed priority (1 highest);
  - highest-priority ready process gets CPU;
  - process continues until done or wait state.
- Processes
  - P1: priority 1, execution time 10 Ready at T_{15}
  - P2: priority 2, execution time 30 Ready at T_0
  - P3: priority 3, execution time 20

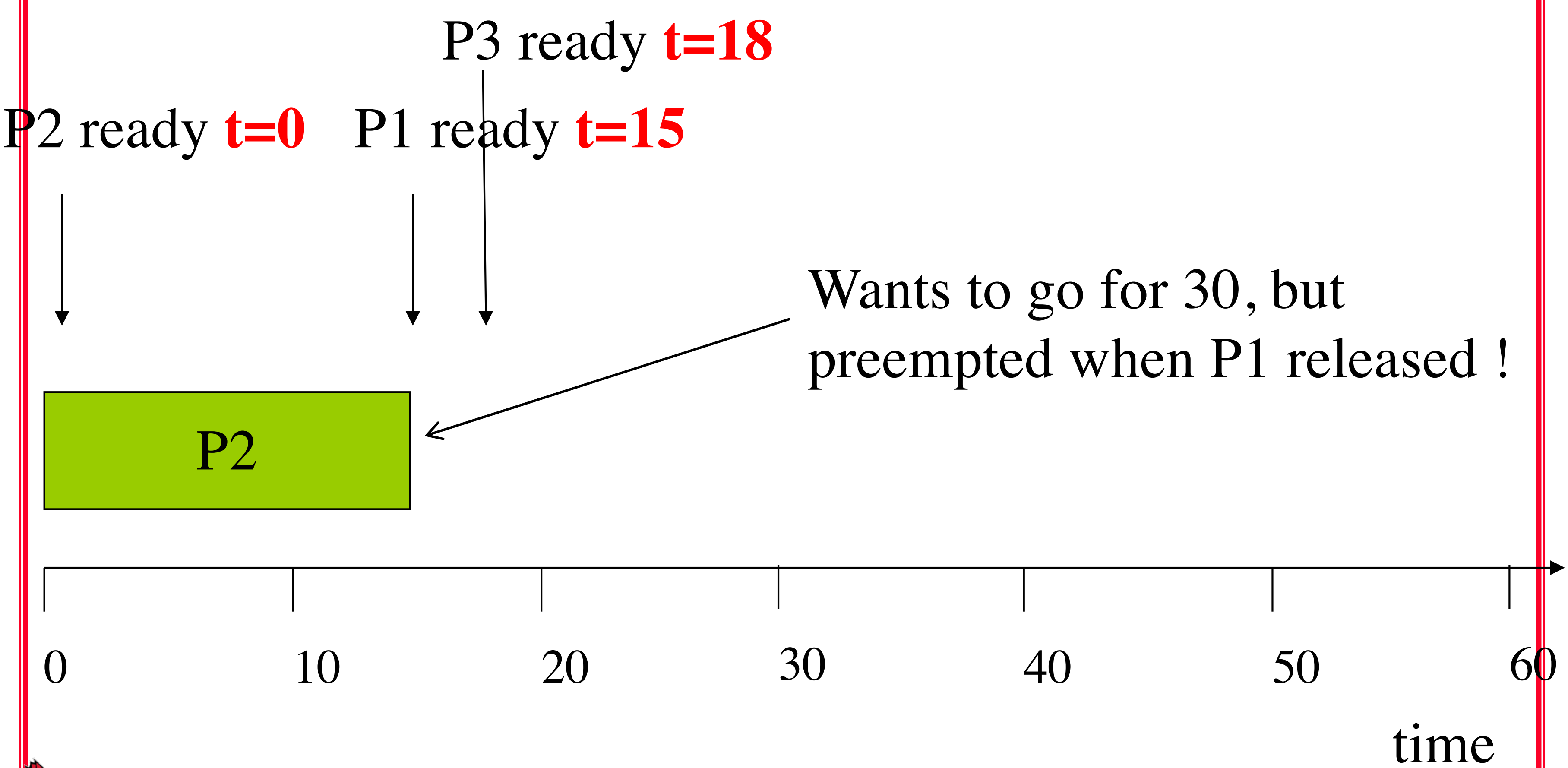
Priority-driven scheduling example

- Rules:
  - each process has a fixed priority (1 highest);
  - highest-priority ready process gets CPU;
  - process continues until done or wait state.
- Processes
  - P1: priority 1, execution time 10 Ready at T_{15}
  - P2: priority 2, execution time 30 Ready at T_0
  - P3: priority 3, execution time 20 Ready at T_{18}

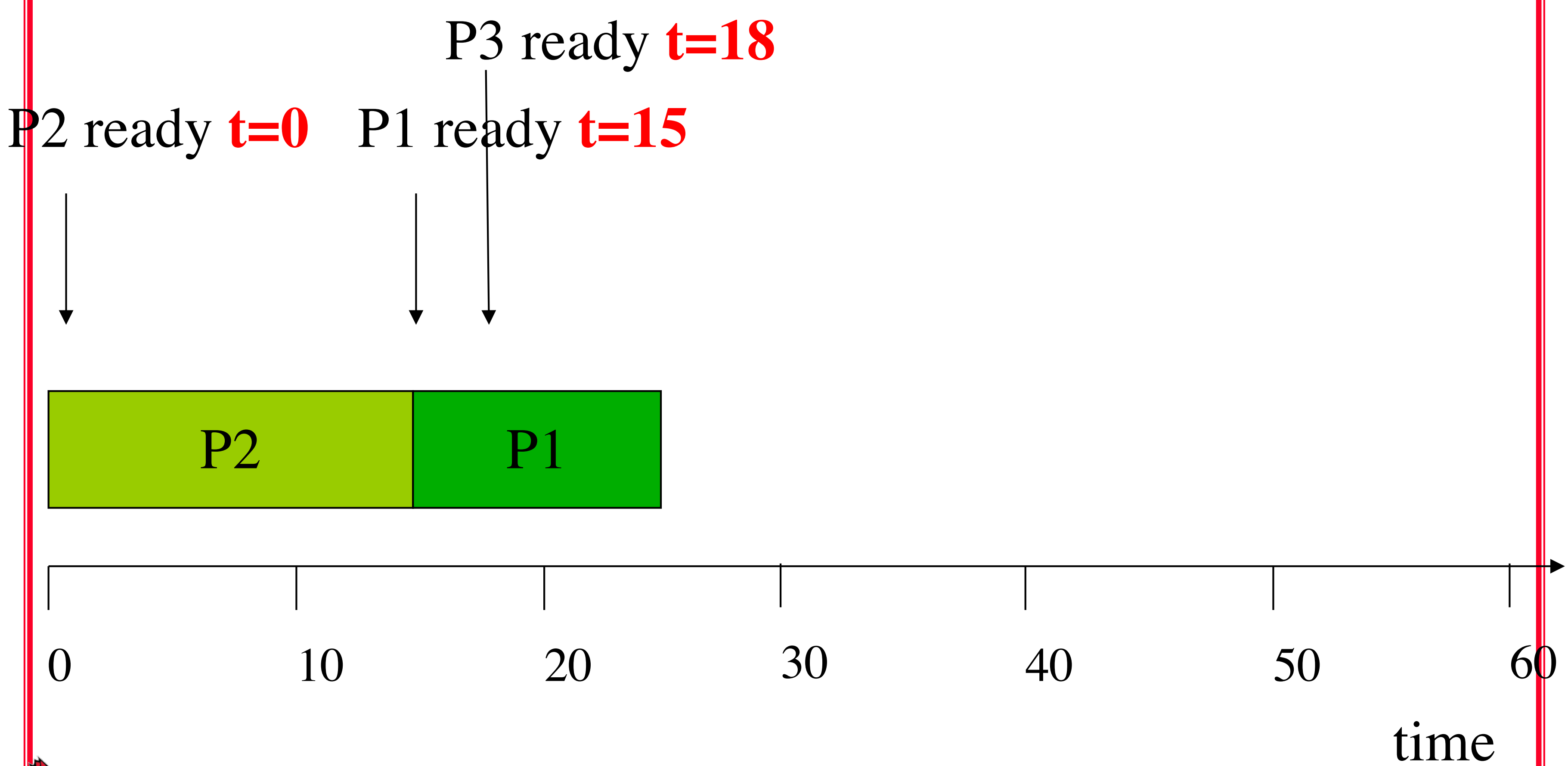
Priority-driven scheduling example



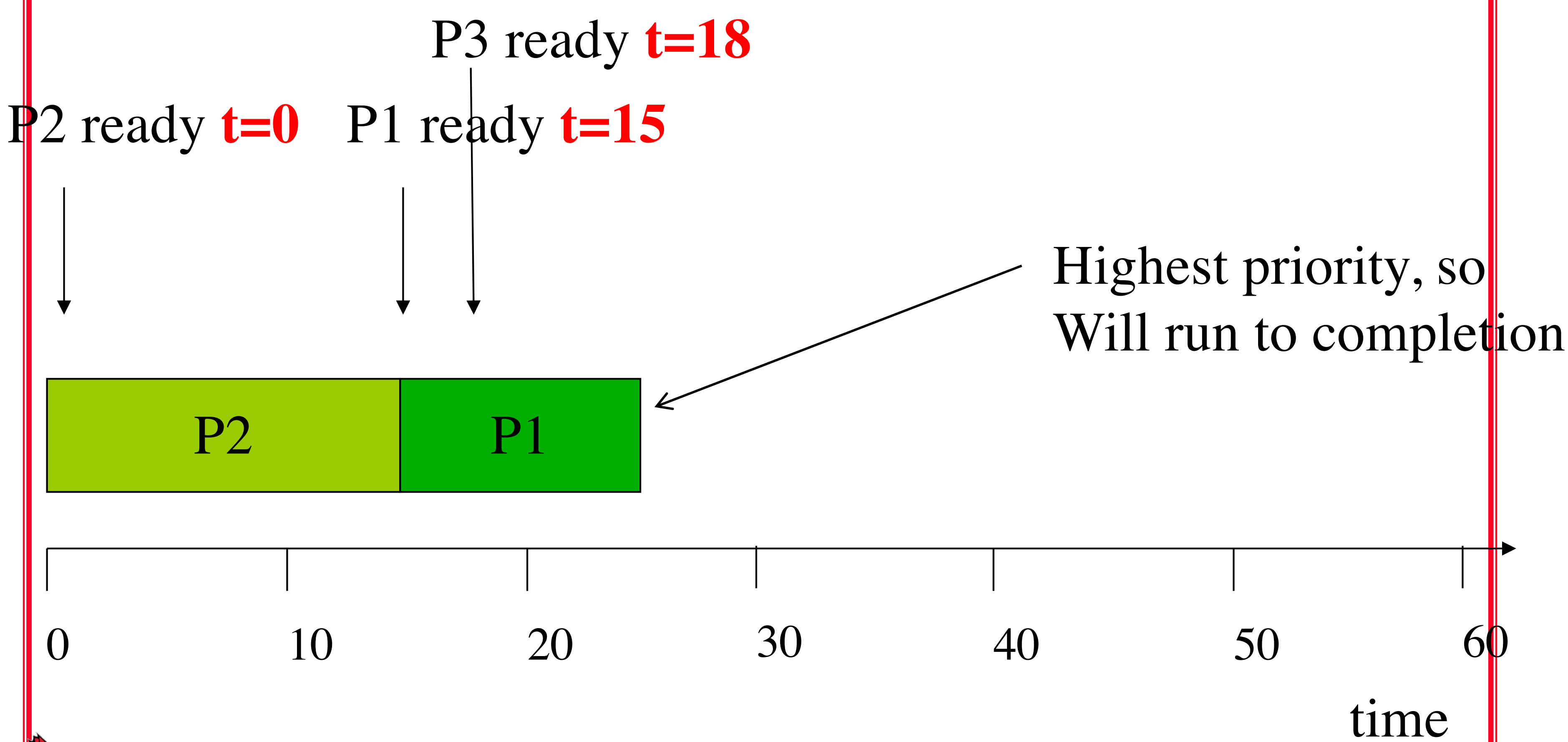
Priority-driven scheduling example



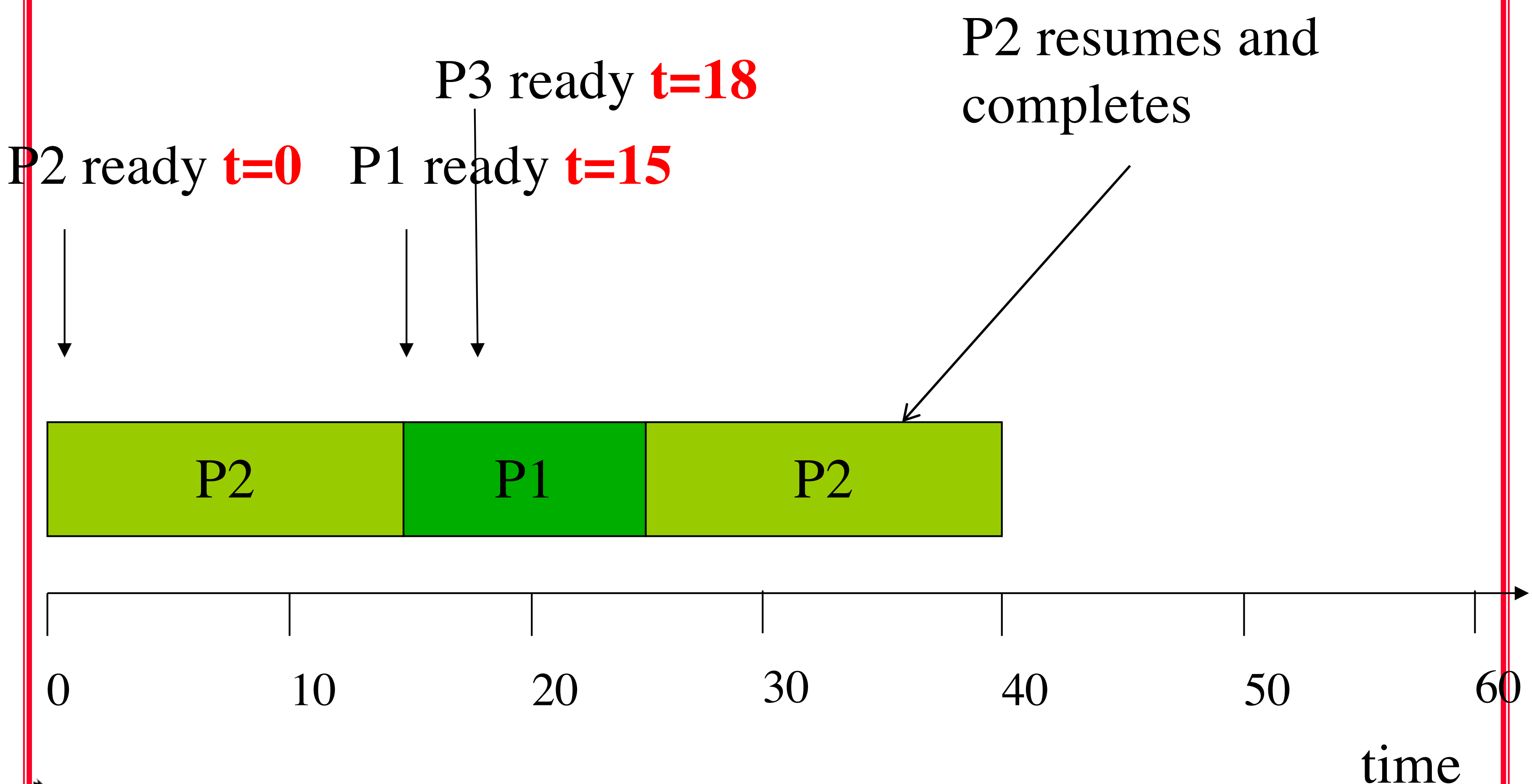
Priority-driven scheduling example



Priority-driven scheduling example



Priority-driven scheduling example



Priority-driven scheduling example