CS 423
Operating System Design:
Midterm Review

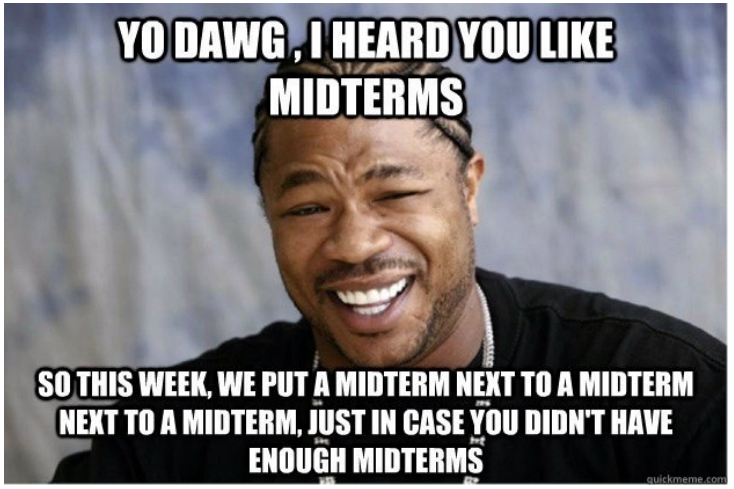
Professor Adam Bates
Spring 2018

Goals for Today

- Learning Objective:
  - Review material, and also my strategies for writing midterm questions
- Announcements, etc:
  - Midterm exam on Wednesday at 11

Reminder: Please put away devices at the start of class

Midterm Details



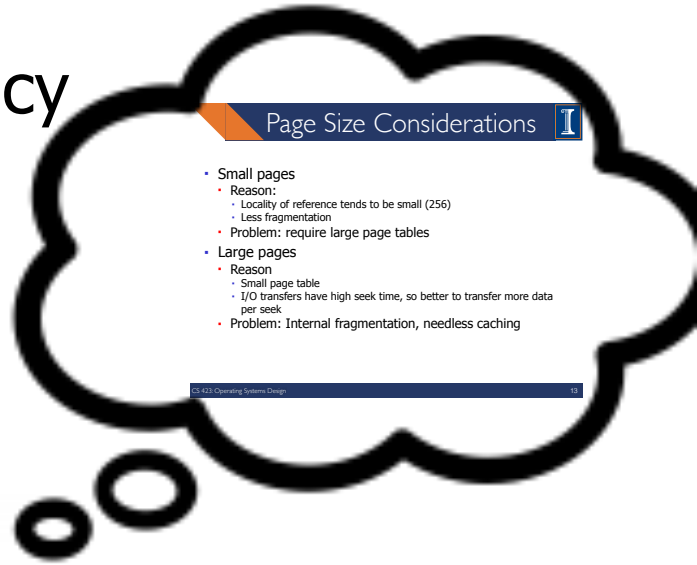
- In-Class on March 6th.
  - i.e., *50 minutes*
- Scantron Multiple choice
  - bring pencils!
- 20-30 Questions
- **Openbook**: Textbooks, paper notes, printed sheets allowed. *No electronic devices permitted (or necessary)!*
- **Content**: All lecture and text material covered prior to March 6th (i.e., up to and including memory)

Sample Midterm Q

- Which of the following is <u>not</u> a good reason for increasing the size of a system's page frames?
  - Improves memory utilization/efficiency
  - Decreases memory footprint of virtual memory management
  - Improves disk utilization/efficiency

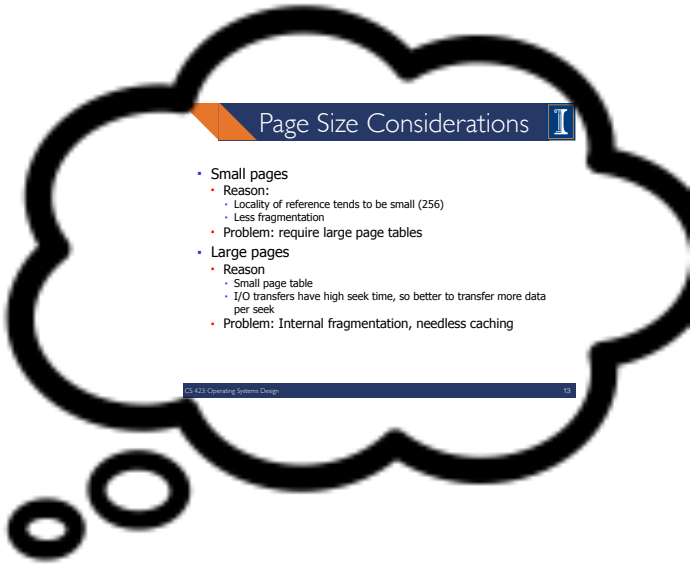
Sample Midterm Q

- Which of the following is <u>not</u> a good reason for increasing the size of a system's page frames?
  - Improves memory utilization/efficiency
  - Decreases memory footprint of virtual memory management
  - Improves disk utilization/efficiency



Sample Midterm Q

- Which of the following is <u>not</u> a good reason for increasing the size of a system's page frames?
  - Less Fragmentation
  - Smaller Page Table
  - Better to transfer more data per disk seek

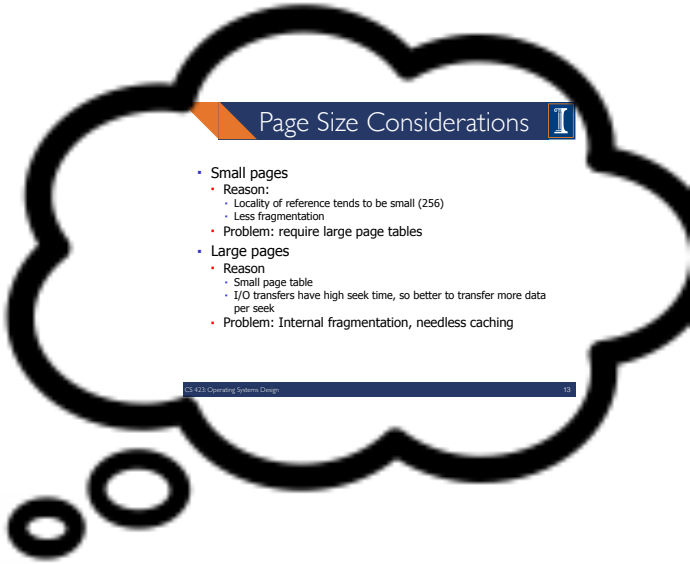


Page Size Considerations

- Small pages
  - Reason:
    - Locality of reference tends to be small (256)
    - Less fragmentation
  - Problem: require large page tables
- Large pages
  - Reason
    - Small page table
    - I/O transfers have high seek time, so better to transfer more data per seek
  - Problem: Internal fragmentation, needless caching

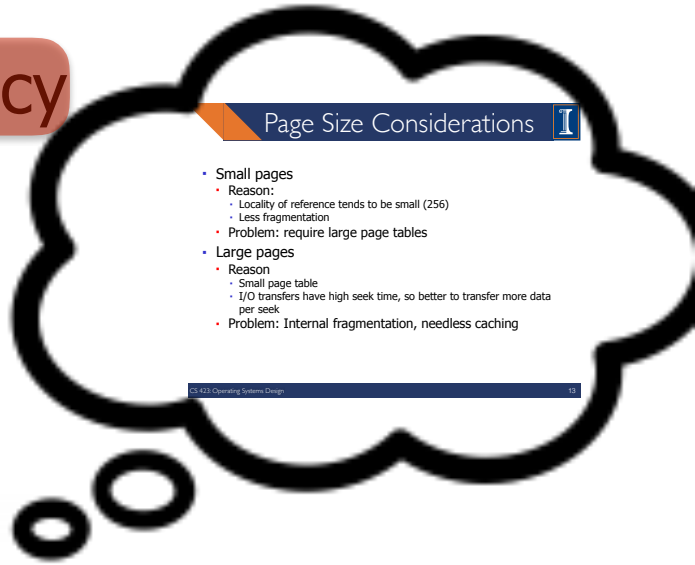
Sample Midterm Q

- Which of the following is <u>not</u> a good reason for increasing the size of a system's page frames?
  - Less Fragmentation
  - Smaller Page Table
  - Better to transfer more data per disk seek



Sample Midterm Q

- Which of the following is not a good reason for increasing the size of a system's page frames?
  - Improves memory utilization/efficiency
  - Decreases memory footprint of virtual memory management
  - Improves disk utilization/efficiency

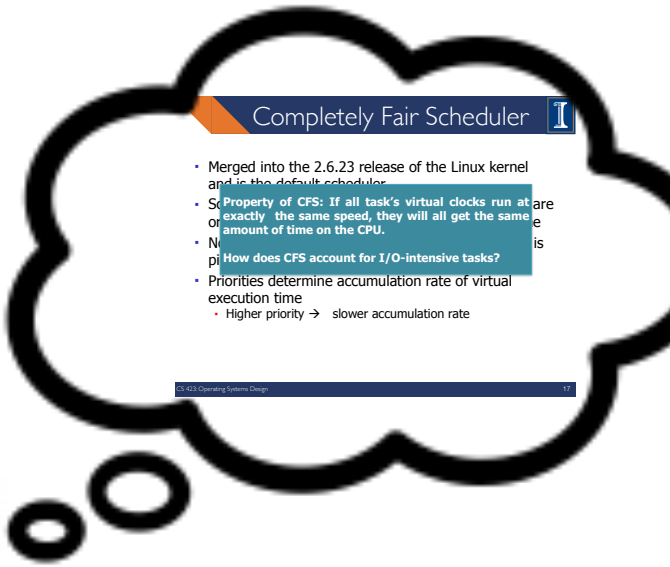


Sample Midterm Q

- With CFS active, tasks X, Y, and Z accumulate virtual execution time at a rate of 1, 2, and 3, respectively. What is the expected share of the CPU that each gets?
  - X=17%, Y=33%, Z=50%
  - X=55%, Y=27%, Z=18%
  - X=50%, Y=33%, Z=17%
  - X=18%, Y=27%, Z=55%

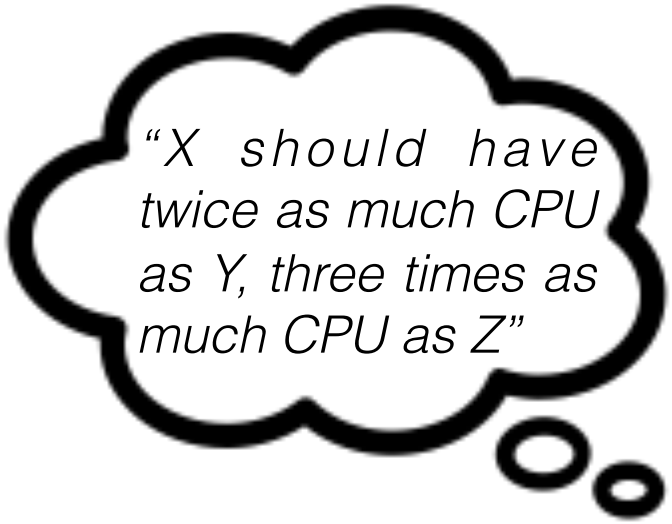
Sample Midterm Q

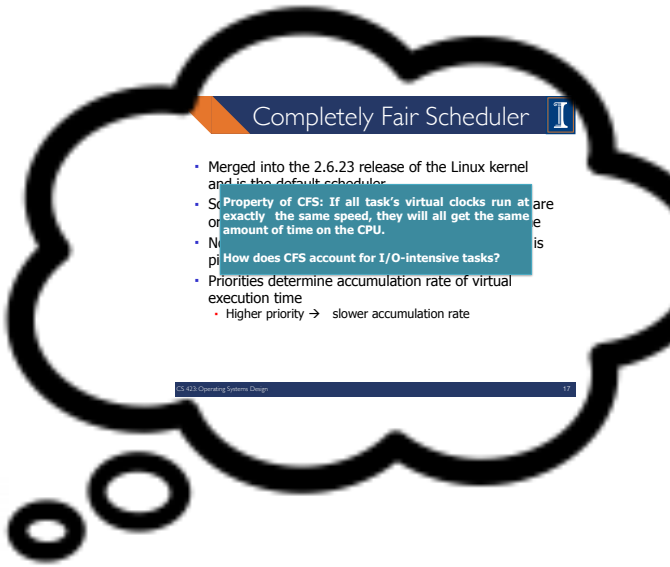
- With CFS active, tasks X, Y, and Z accumulate virtual execution time at a rate of 1, 2, and 3, respectively. What is the expected share of the CPU that each gets?
  - X=17%, Y=33%, Z=50%
  - X=55%, Y=27%, Z=18%
  - X=50%, Y=33%, Z=17%
  - X=18%, Y=27%, Z=55%



Sample Midterm Q

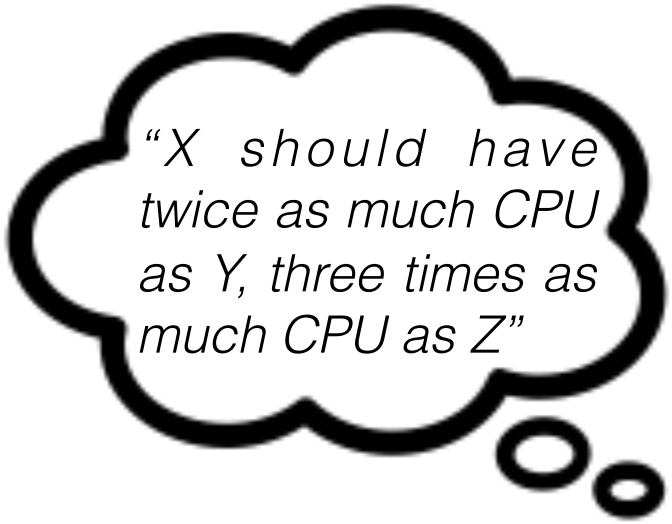
- With CFS active, tasks X, Y, and Z accumulate virtual execution time at a rate of 1, 2, and 3, respectively. What is the expected share of the CPU that each gets?
  - X=17%, Y=33%, Z=50%
  - X=55%, Y=27%, Z=18%
  - X=50%, Y=33%, Z=17%
  - X=18%, Y=27%, Z=55%

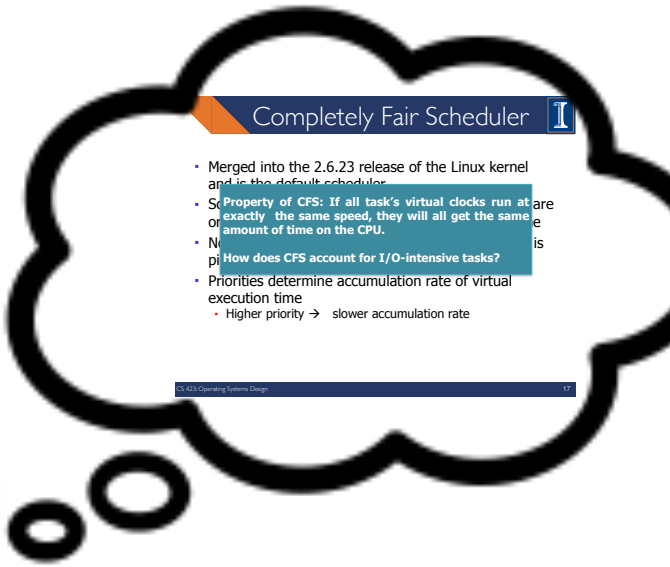




Sample Midterm Q

- With CFS active, tasks X, Y, and Z accumulate virtual execution time at a rate of 1, 2, and 3, respectively. What is the expected share of the CPU that each gets?
  - X=17%, Y=33%, Z=50%
  - X=55%, Y=27%, Z=18%
  - X=50%, Y=33%, Z=17%
  - X=18%, Y=27%, Z=55%





Sample Midterm Q

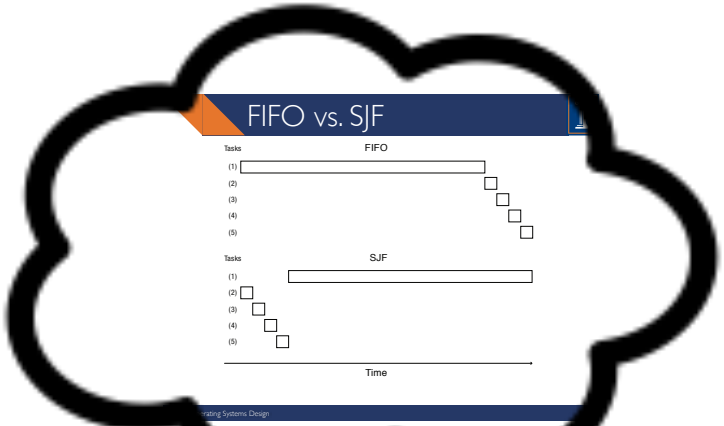
- Below are chronologically-ordered series of tasks with their completion time shown. Which sequence offers a pessimal (i.e., worst-case) average response time for FIFO scheduling?
  - 1, 2, 3, 4
  - 2, 2, 2, 2
  - 3, 1, 3, 1
  - 4, 3, 2, 1

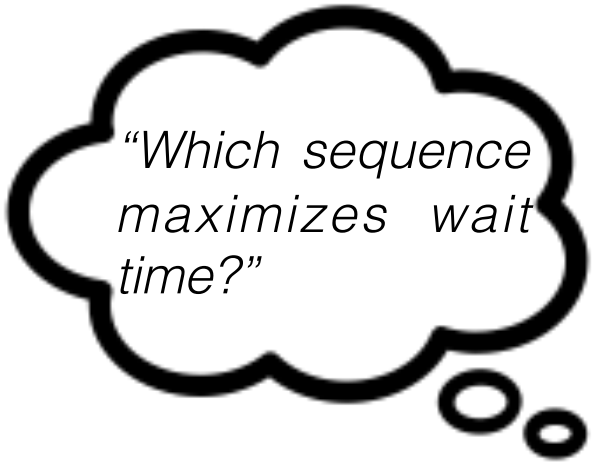
Sample Midterm Q

- Below are chronologically-ordered series of tasks with their completion time shown. Which sequence offers a pessimal (i.e., worst-case) average response time for FIFO scheduling?
  - 1, 2, 3, 4
  - 2, 2, 2, 2
  - 3, 1, 3, 1
  - 4, 3, 2, 1

Sample Midterm Q

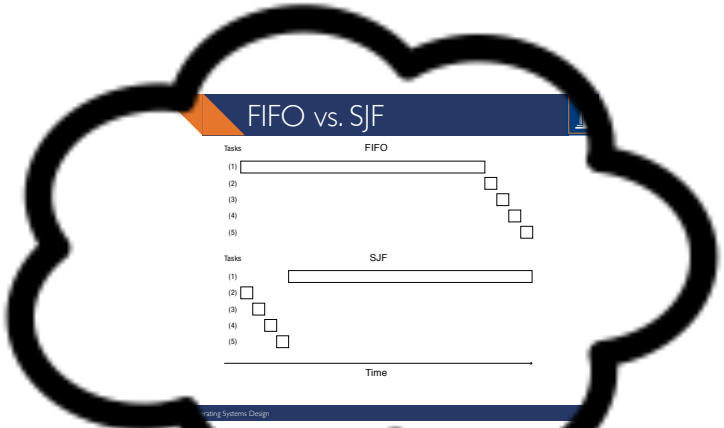
- Below are chronologically-ordered series of tasks with their completion time shown. Which sequence offers a pessimal (i.e., worst-case) average response time for FIFO scheduling?
  - 1, 2, 3, 4
  - 2, 2, 2, 2
  - 3, 1, 3, 1
  - 4, 3, 2, 1

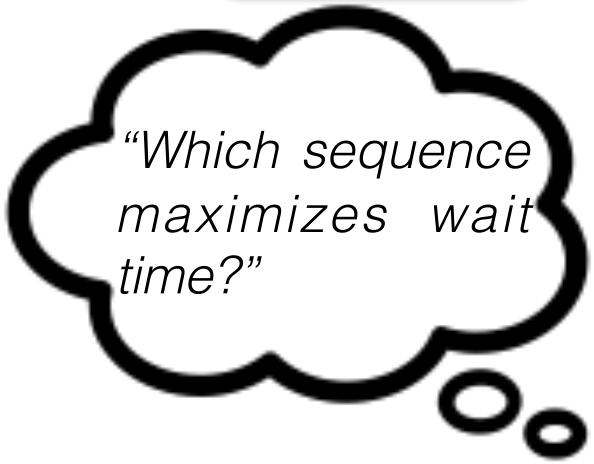




Sample Midterm Q

- Below are chronologically-ordered series of tasks with their completion time shown. Which sequence offers a pessimal (i.e., worst-case) average response time for FIFO scheduling?
  - 1, 2, 3, 4
  - 2, 2, 2, 2
  - 3, 1, 3, 1
  - 4, 3, 2, 1





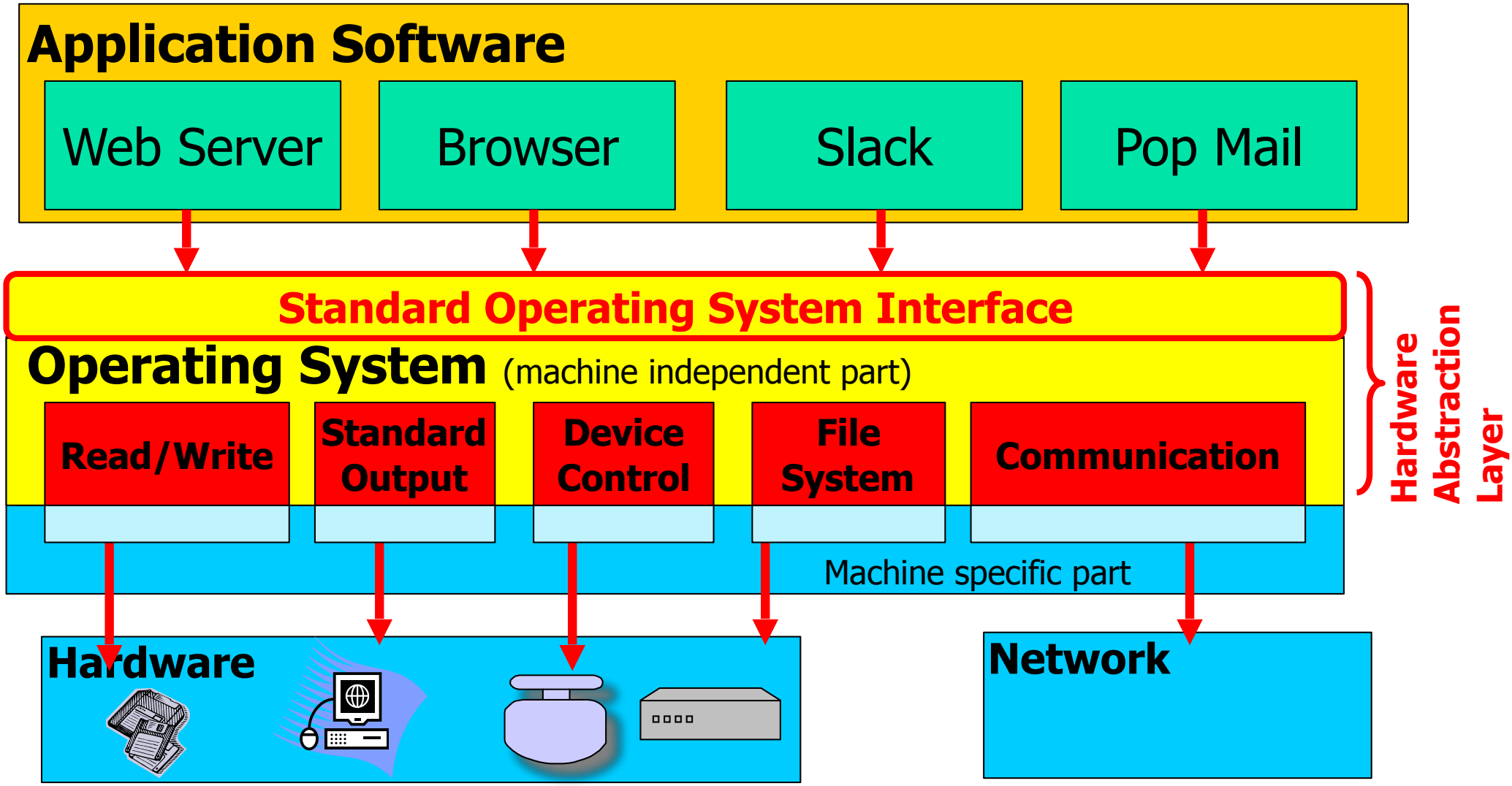
More Q&A

Remainder of these slides

- This is not a study guide
- I prepared these by walking the lecture slides from start to finish and sampling important concepts
- Slides intended to prompt discussion and questions
- Test is written at this point, but this deck leaks minimal information; don't try to read into which slides I did/ didn't copy over to here.
- There are no memory slides since we just covered it, but obviously there will be questions about memory on the exam.

Overview: OS Stack

OS Runs on Multiple Platforms while presenting the same Interface:



Overview: OS Roles

Role #1: Referee

- Manage resource allocation between users and applications
- Isolate different users and applications from one another
- Facilitate and mediate communication between different users and applications

Role #2: Illusionist

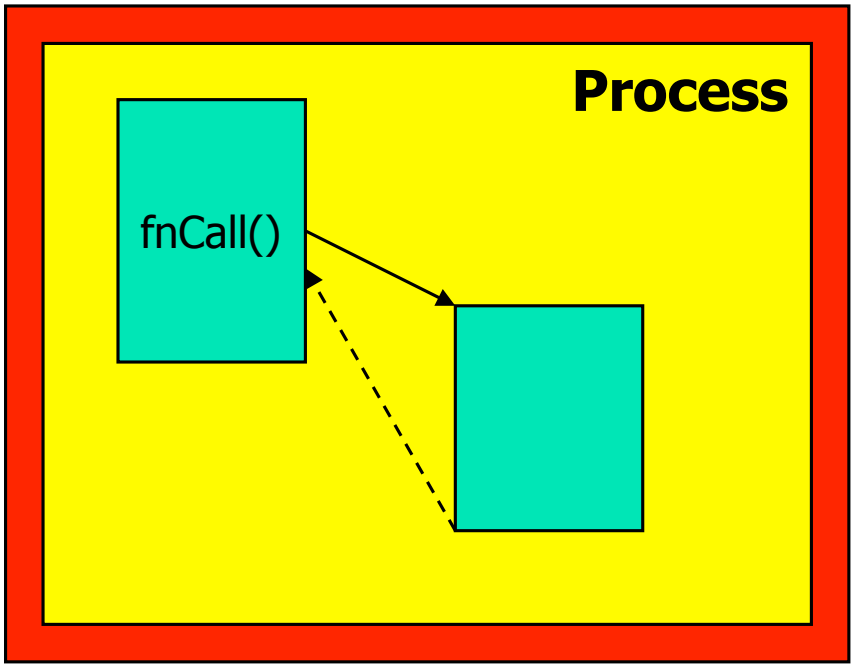
- Allow each application to believe it has the entire machine to itself
- Create the appearance of an Infinite number of processors, (near) infinite memory
- Abstract away complexity of reliability, storage, network communication…

Role #3: Glue

- Manage hardware so applications can be machine-agnostic
- Provide a set of common services that facilitate sharing among applications
- **Examples of "Glue" OS Services?**

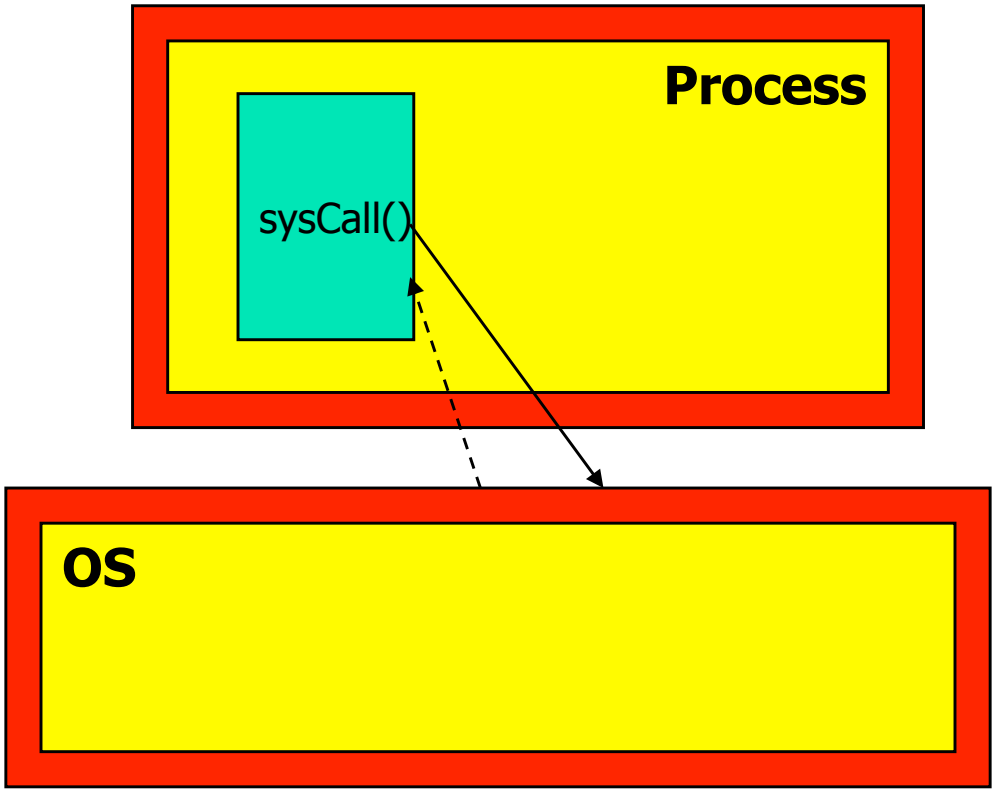
Review: System Calls

Function Calls



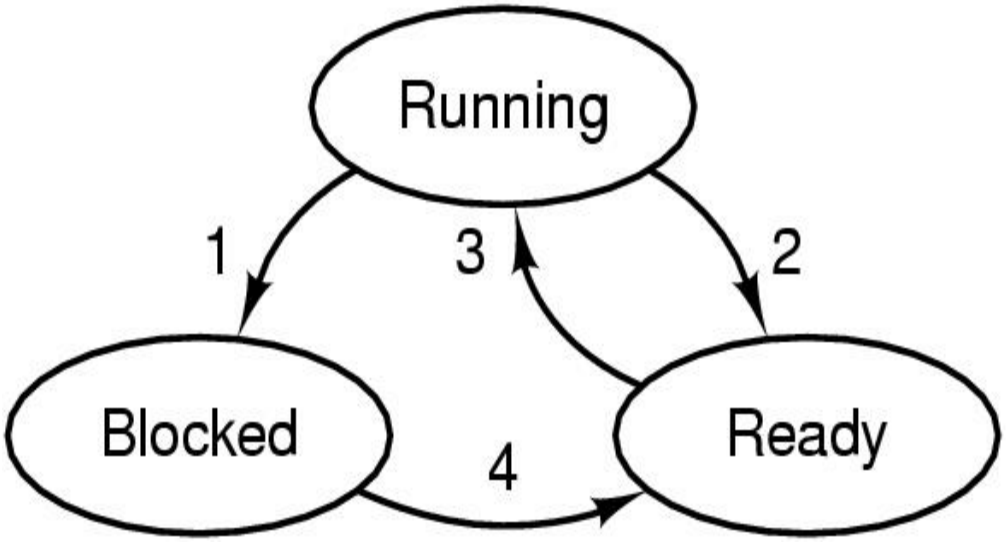
Caller and callee are in the same Process
- Same user
- Same "domain of trust"

System Calls



- OS is trusted; user is not.
- OS has super-privileges; user does not
- Must take measures to prevent abuse

Review: Process Abstraction

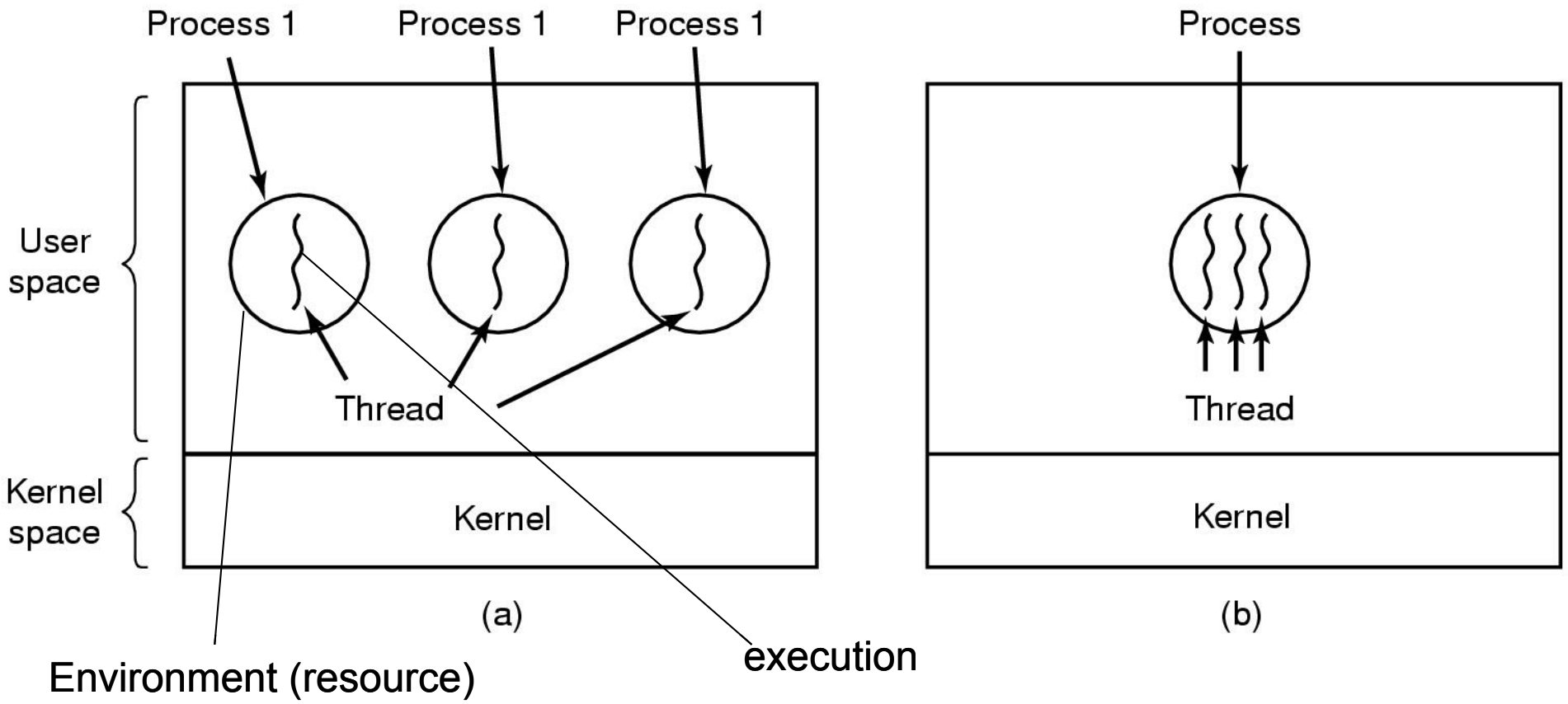


1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

- Possible process states
  - Running (occupy CPU)
  - Blocked
  - Ready (does not occupy CPU)
  - Other states: suspended, terminated

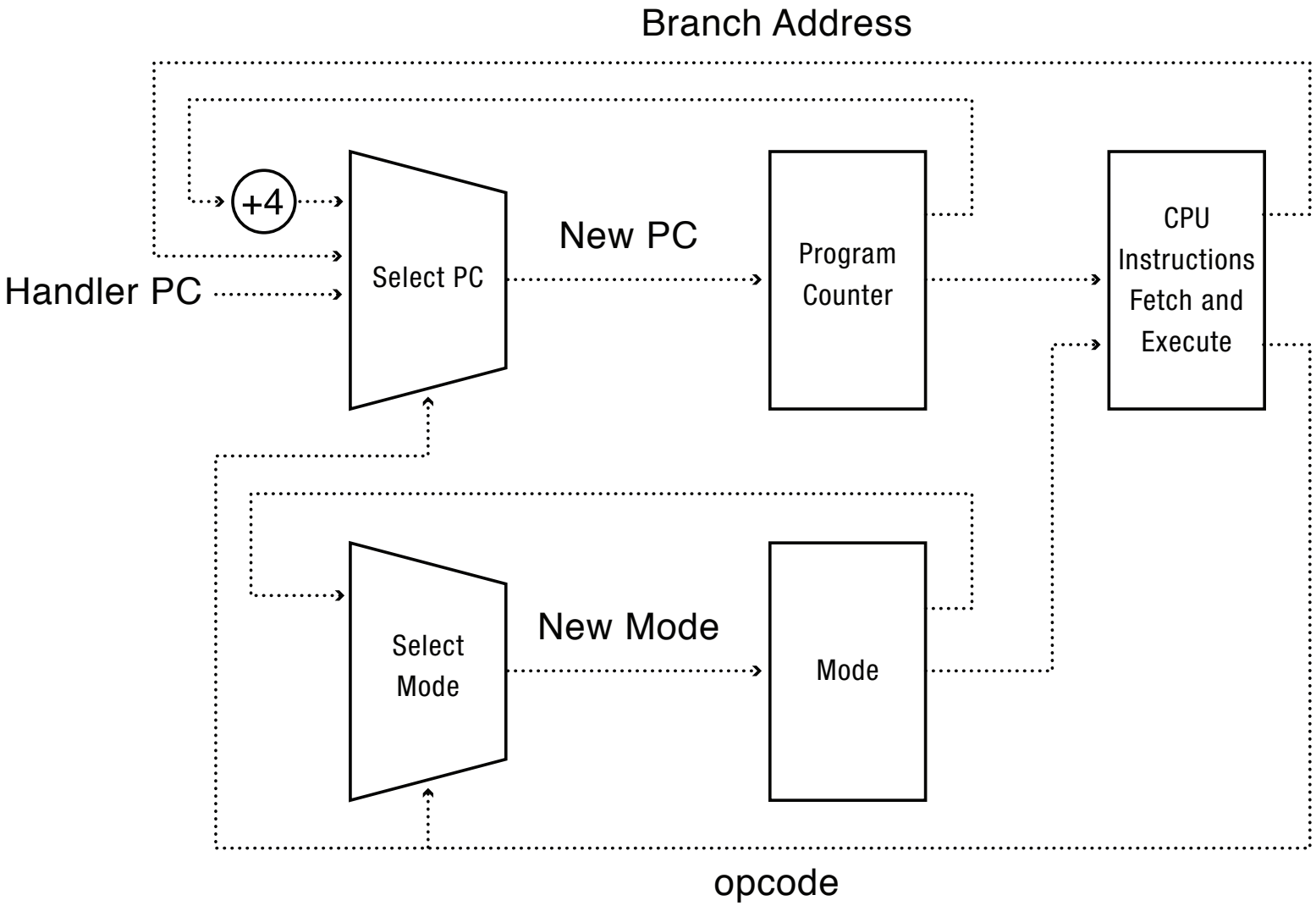
Question: in a single processor machine, how many process can be in running state?

Review: Threads

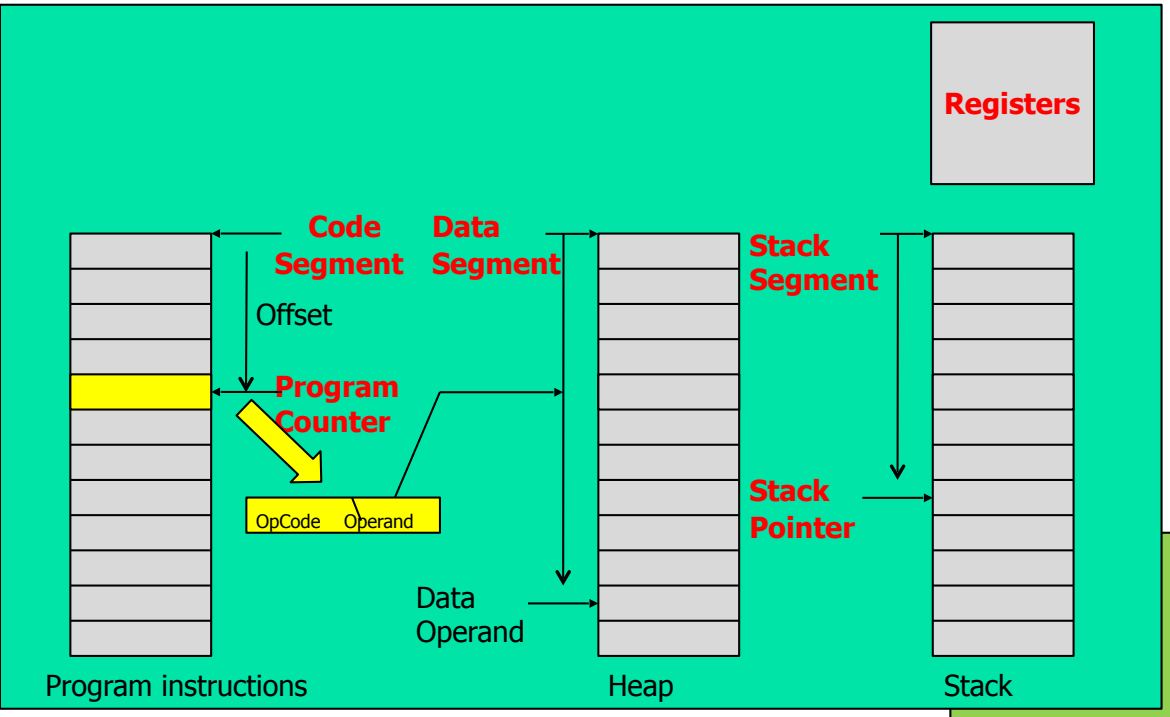


- (a) Three processes each with one thread
- (b) One process with three threads

Kernel Abstraction: HW Support

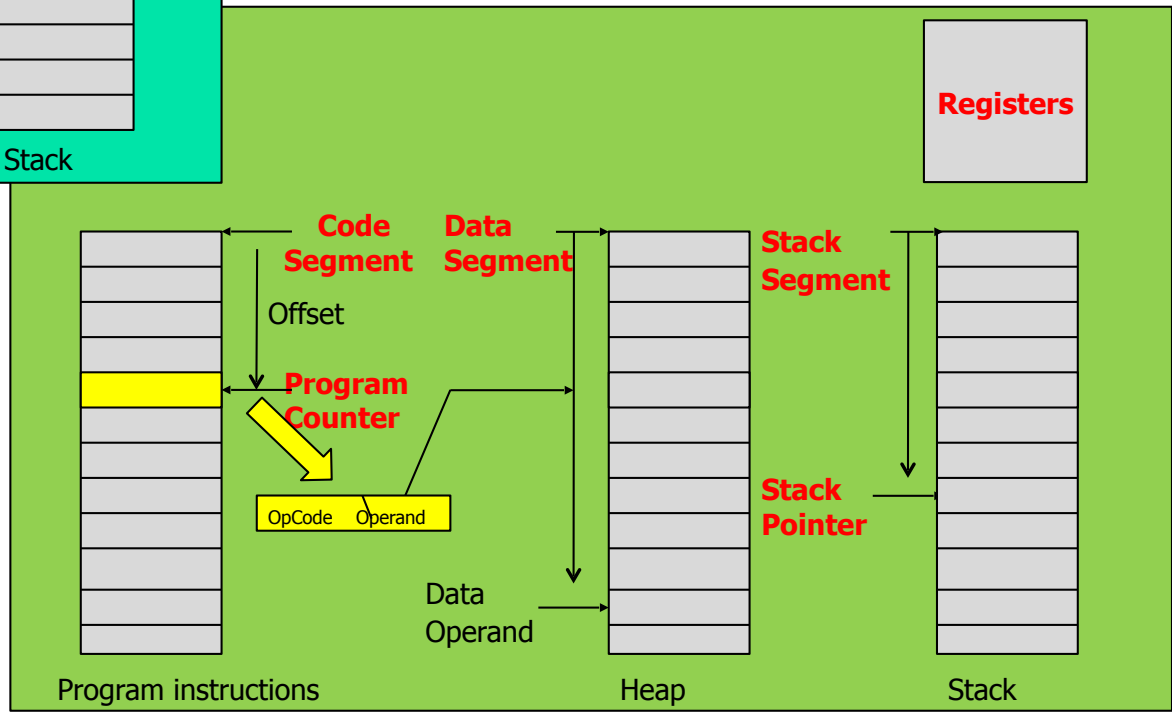


Kernel Abstraction: CTX Switch



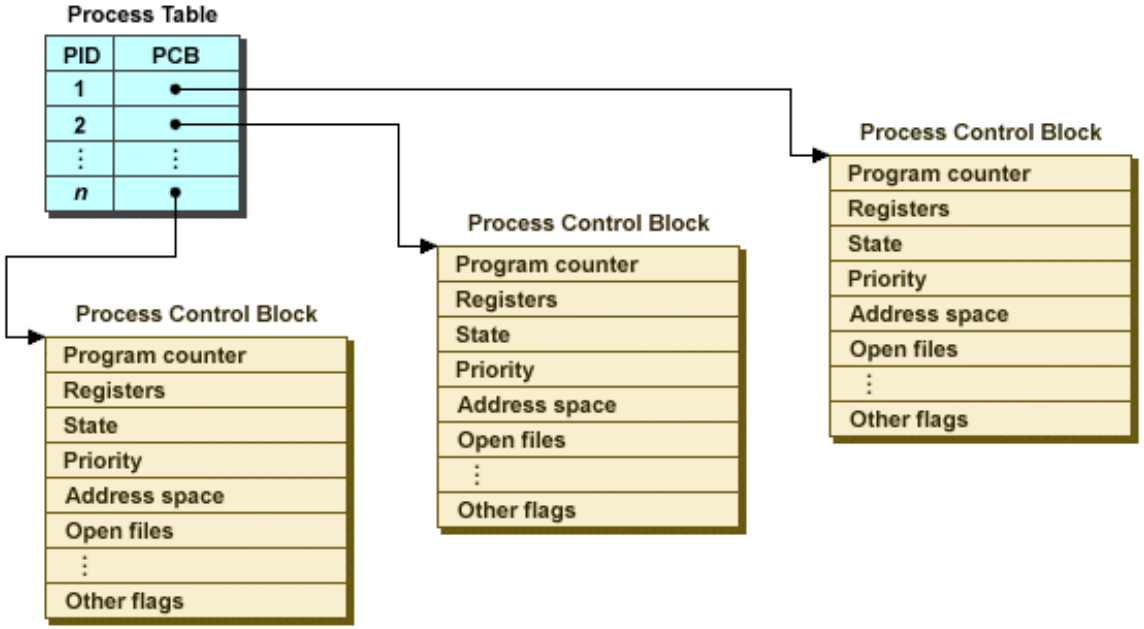
Load State (Context)

Save State (Context)

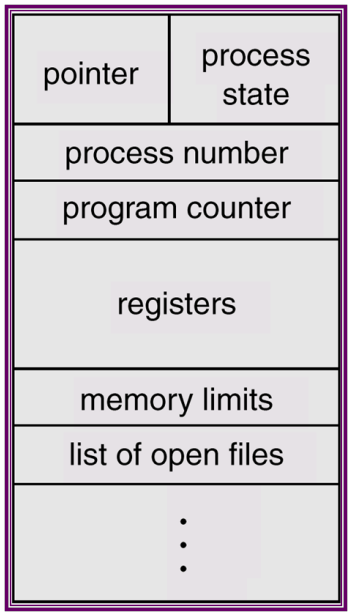


Kernel Abstraction: PCBs

The state for processes that are not running on the CPU are maintained in the Process Control Block (PCB) data structure

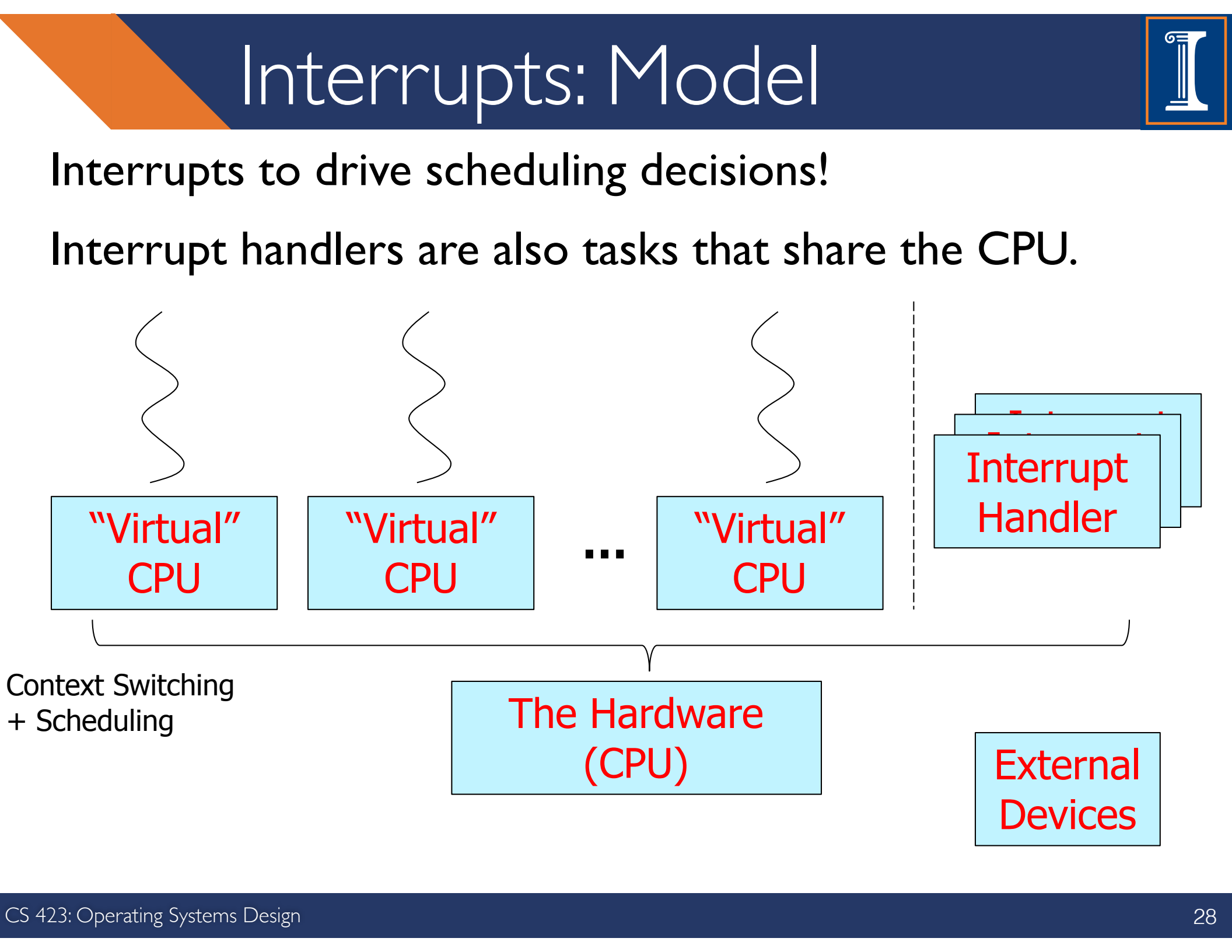


Updated during context switch



An alternate PCB diagram

Interrupts: Model

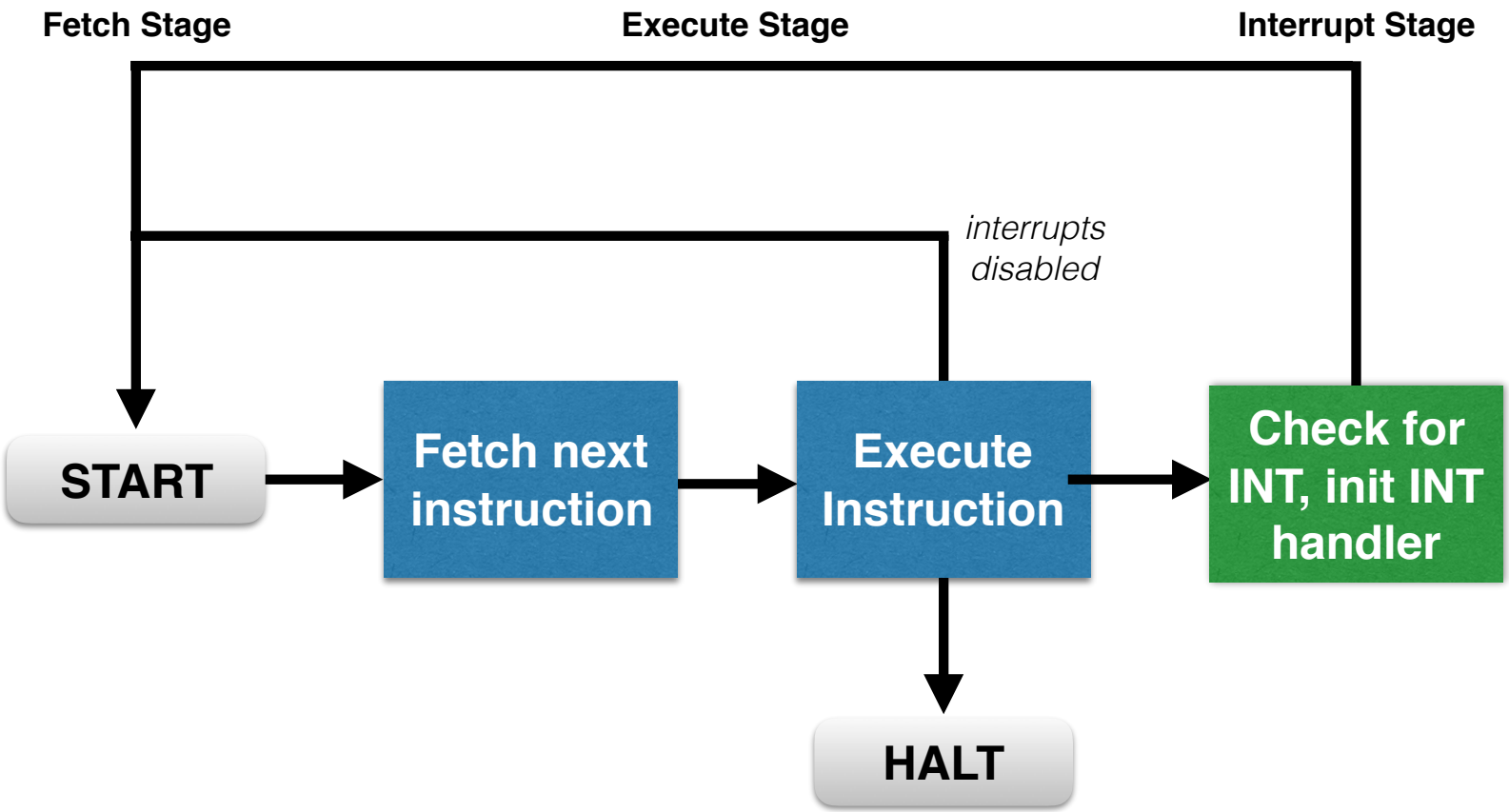


Interrupts to drive scheduling decisions!

Interrupt handlers are also tasks that share the CPU.

Interrupts: Handling

How does interrupt handling change the instruction cycle?



Interrupts: Handling

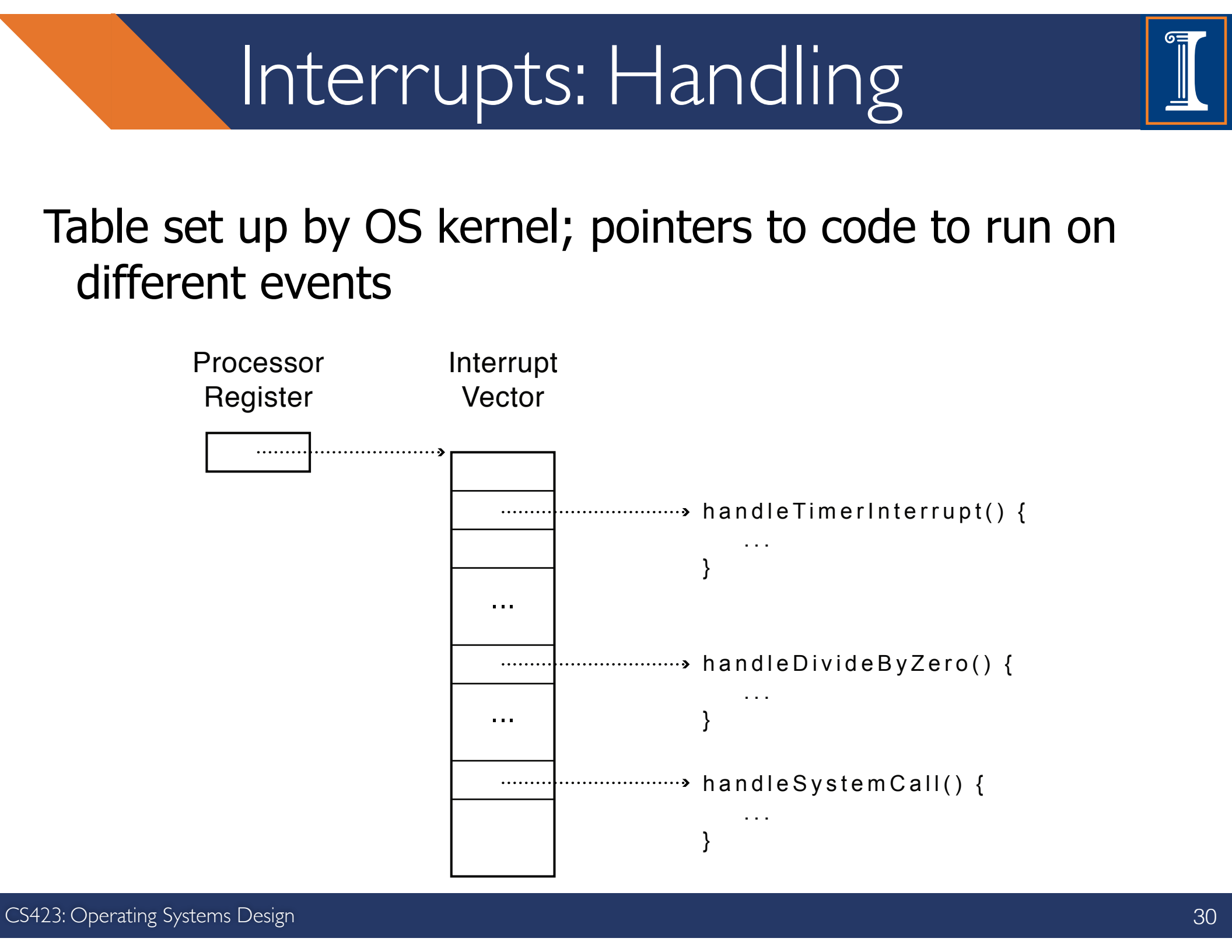
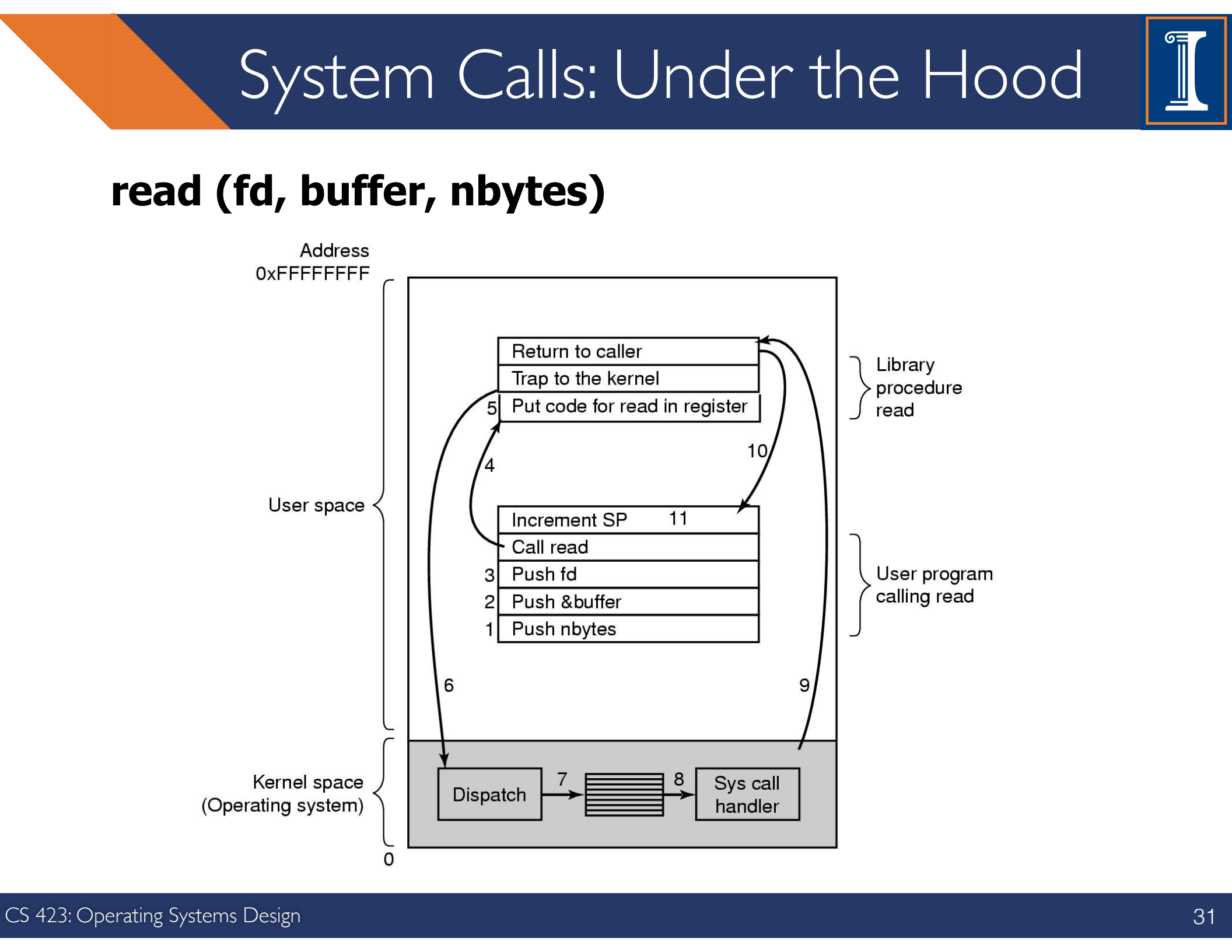


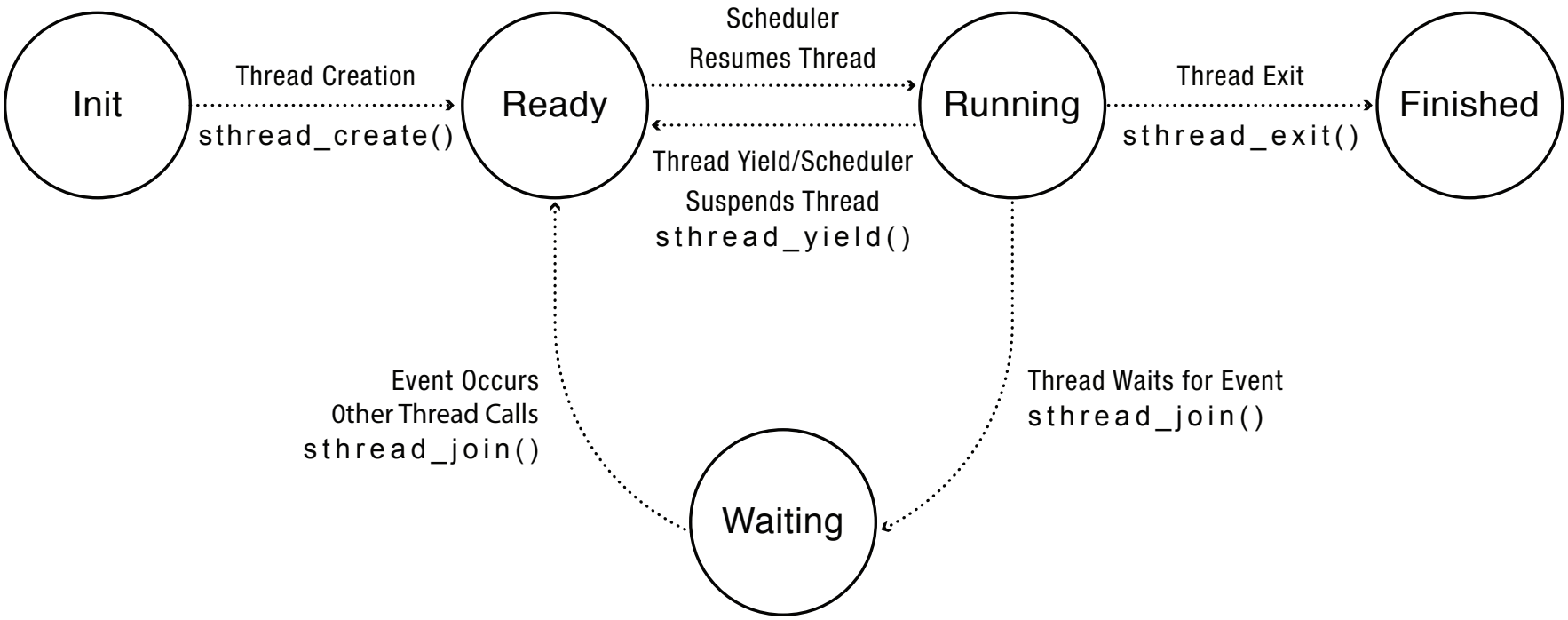
Table set up by OS kernel; pointers to code to run on different events

System Calls: Under the Hood

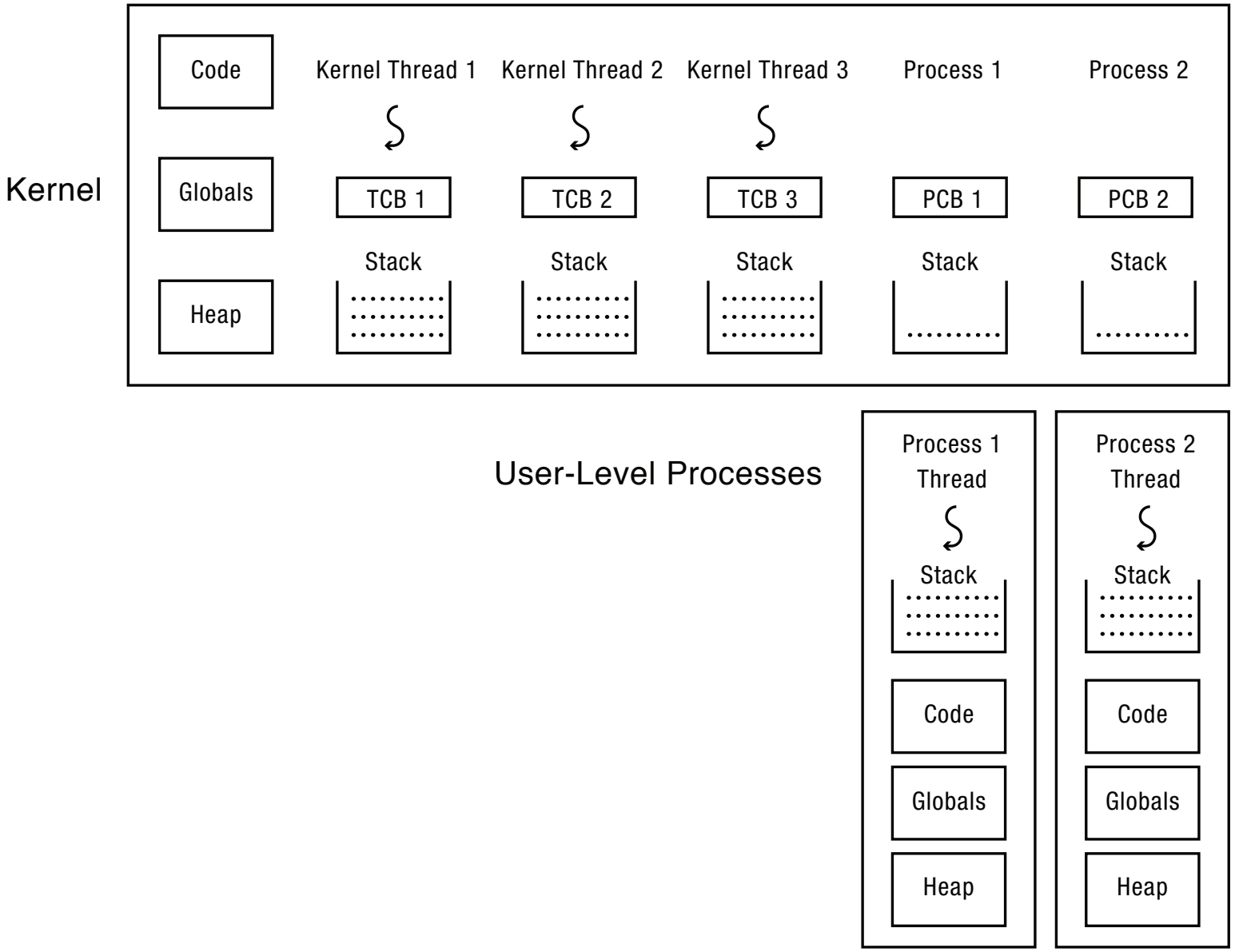


read (fd, buffer, nbytes)

Concurrency: Thread Lifecycle



Concurrency: Thread State



Synchronization: Principals

Concurrent Applications

Shared Objects

Bounded Buffer Barrier

Synchronization Variables

Semaphores Locks Condition Variables

Atomic Instructions

Interrupt Disable Test-and-Set

Hardware

Multiple Processors Hardware Interrupts

Synchronization: Locks

- Lock::acquire
  - wait until lock is free, then take it
- Lock::release
  - release lock, waking up anyone waiting for it

1. At most one lock holder at a time (safety)
2. If no one holding, acquire gets lock (progress)
3. If all lock holders finish and no higher priority waiters, waiter eventually gets lock (progress)

Synchronization: Condition Variables

- Waiting inside a critical section
  - Called only when holding a lock
- CV::Wait — atomically release lock and relinquish processor
  - Reacquire the lock when wakened
- CV::Signal — wake up a waiter, if any
- CV::Broadcast — wake up all waiters, if any

Synchronization: Spinlocks

- A spinlock is a lock where the processor waits in a loop for the lock to become free
  - Assumes lock will be held for a short time
  - Used to protect the CPU scheduler and to implement locks

```
Spinlock::acquire() {
   while (testAndSet(&lockValue) == BUSY)
      ;
}

Spinlock::release() {
   lockValue = FREE;
  memorybarrier();
}
```

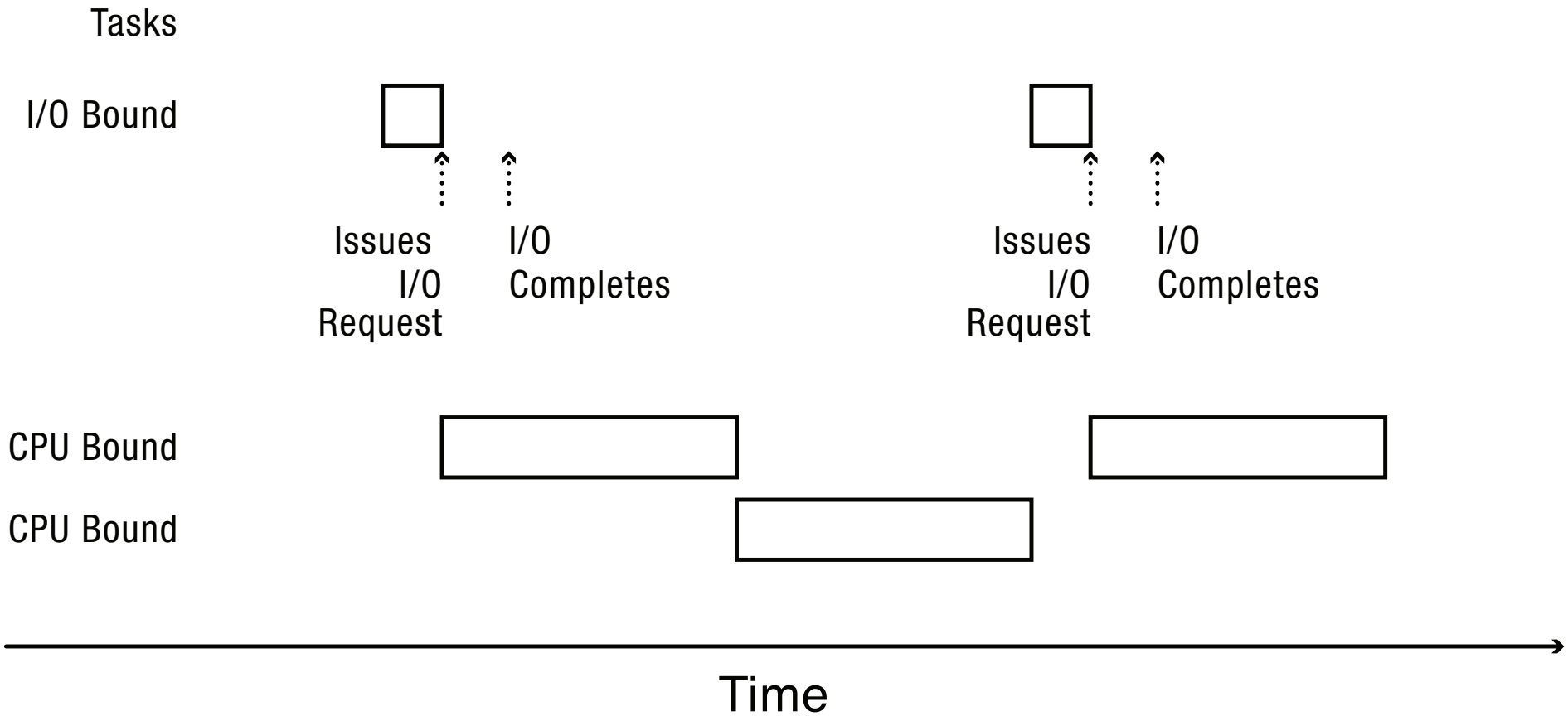
Semaphores

- Semaphore has a non-negative integer value
  - P() atomically waits for value to become > 0, then decrements
  - V() atomically increments value (waking up waiter if needed)
- Semaphores are like integers except:
  - Only operations are P and V
  - Operations are atomic
    - If value is 1, two P's will result in value 0 and one waiter

Scheduling: Principals

- Basic scheduling algorithms
  - FIFO (FCFS)
  - Shortest job first
  - Round Robin
- What is an optimal algorithm in the sense of maximizing the number of jobs finished (i.e., minimizing average response time)?

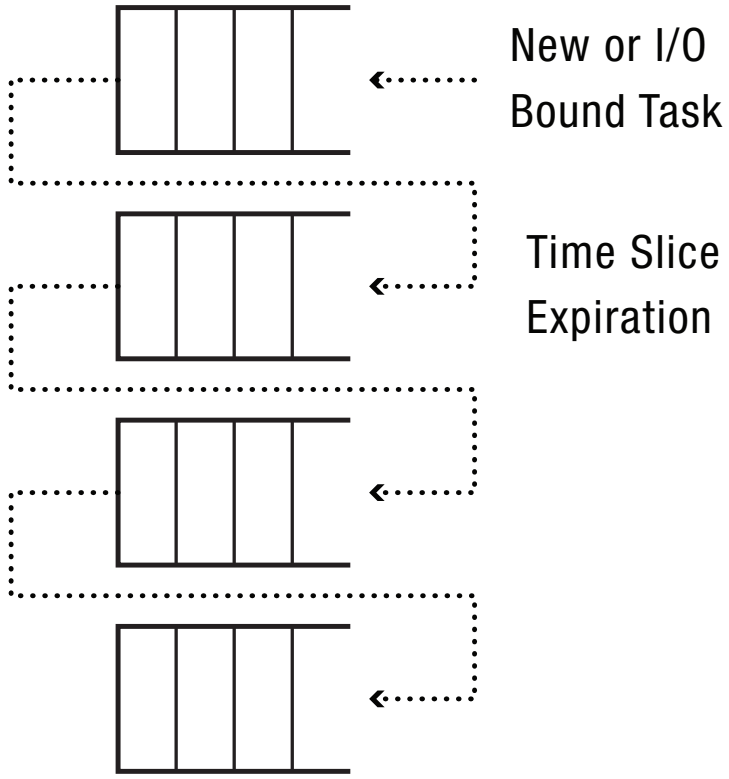
Scheduling: Mixed Workloads??



Scheduling: MFQ

Priority	Time Slice (ms)
1	10
2	20
3	40
4	80

Round Robin Queues



Scheduling: Early Linux

- Linux 1.2: circular queue w/ round-robin policy.
  - Simple and minimal.
  - Did not meet many of the aforementioned goals

- Linux 2.2: introduced scheduling classes (real-time, non-real-time).

```
/* Scheduling Policies
*/
#define SCHED_OTHER  0 // Normal user tasks (default)
#define SCHED_FIFO   1 // RT: Will almost never be preempted
#define SCHED_RR     2 // RT: Prioritized RR queues
```

Scheduling: SCHED_NORMAL

- Used for non real-time processes
- Complex heuristic to balance the needs of I/O and CPU centric applications
- Processes start at 120 by default
  - Static priority
    - A "nice" value: 19 to -20.
    - Inherited from the parent process
    - Altered by user (negative values require special permission)
  - Dynamic priority
    - Based on static priority and applications characteristics (interactive or CPU-bound)
    - Favor interactive applications over CPU-bound ones
  - Timeslice is mapped from priority

Scheduling: SCHED_NORMAL Heuristic

How does a dynamic priority adjust CPU access?

bonus = min (10, (avg. sleep time / 100) ms)

- avg. sleep time is 0 => bonus is 0
- avg. sleep time is 100 ms => bonus is 1
- avg. sleep time is 1000 ms => bonus is 10
- avg. sleep time is 1500 ms => bonus is 10
- Your bonus increases as you sleep more.

Max priority # is still 139

dynamic priority =

max (100, min (static priority – bonus + 5, 139))

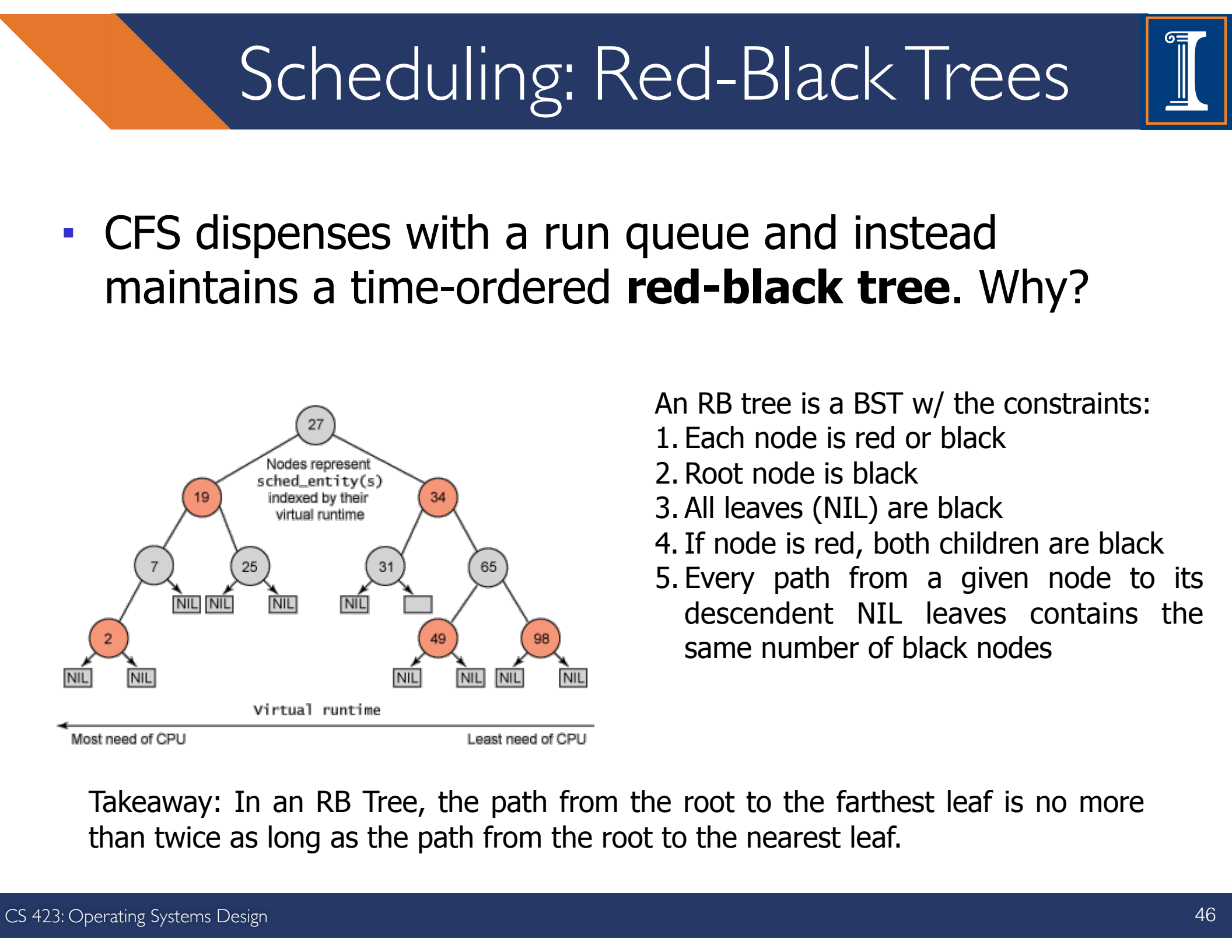
Min priority # is still 100

(Bonus is subtracted to increase priority)

Scheduling: CFS

- Merged into the 2.6.23 release of the Linux kernel and is the default scheduler.
- Scheduler maintains a red-black tree where nodes are ordered according to received virtual execution time
- Node with smallest virtual received execution time is picked next
- Priorities determine accumulation rate of virtual execution time
  - Higher priority → slower accumulation rate

Scheduling: Red-Black Trees



- CFS dispenses with a run queue and instead maintains a time-ordered **red-black tree**. Why?

An RB tree is a BST w/ the constraints:

1. Each node is red or black
2. Root node is black
3. All leaves (NIL) are black
4. If node is red, both children are black
5. Every path from a given node to its descendent NIL leaves contains the same number of black nodes

Takeaway: In an RB Tree, the path from the root to the farthest leaf is no more than twice as long as the path from the root to the nearest leaf.

Scheduling: Multi-Processor

- CPU affinity would seem to necessitate a multi-queue approach to scheduling… but how?
- Asymmetric Multiprocessing (AMP): One processor (e.g., CPU 0) handles all scheduling decisions and I/O processing, other processes execute only user code.
- Symmetric Multiprocessing (SMP): Each processor is self-scheduling. Could work with a single queue, but also works with private queues.
  - Potential problems?

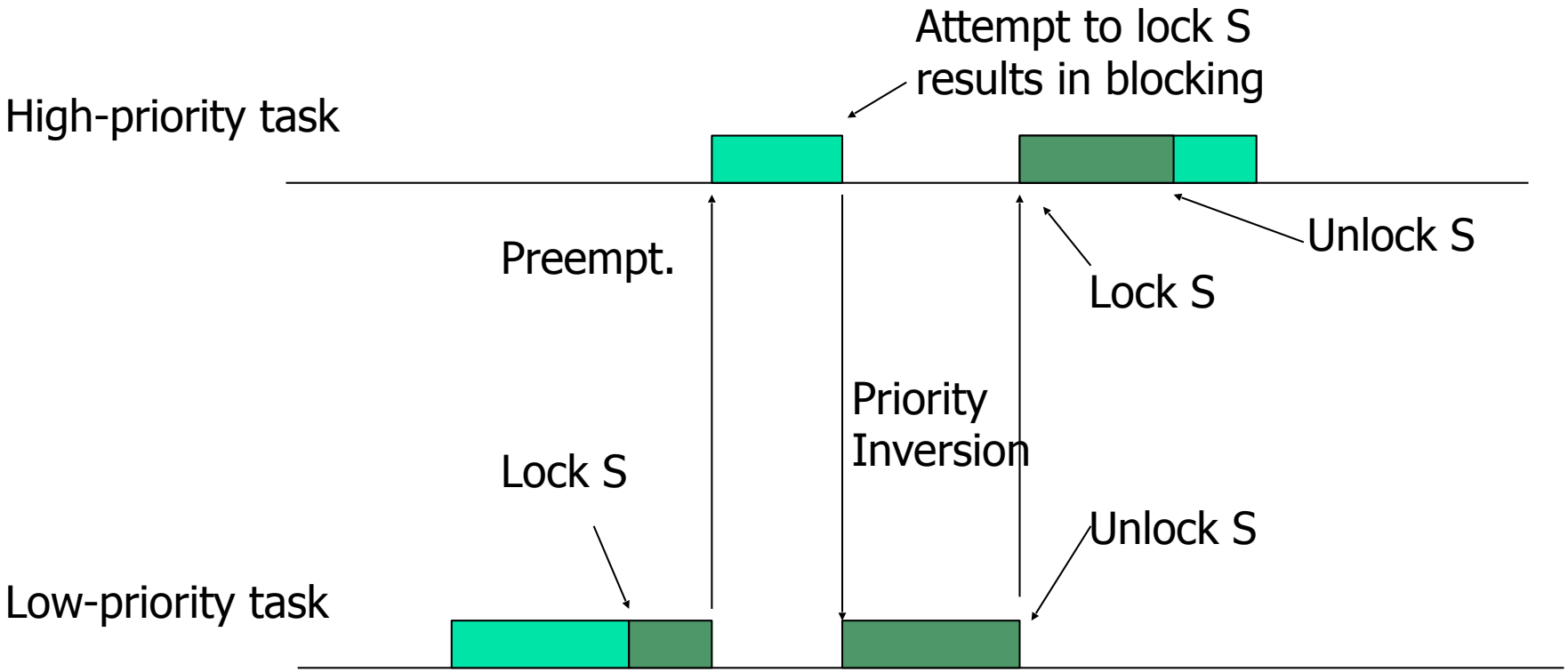
RTS: Scheduling

Re: Real Time Scheduling of Periodic Tasks…

- Result #1: Earliest Deadline First (EDF) is the optimal dynamic priority scheduling policy for independent periodic tasks (meets the most deadlines of all dynamic priority scheduling policies)
- Result #2: Rate Monotonic Scheduling (RM) is the optimal static priority scheduling policy for independent periodic tasks (meets the most deadlines of all static priority scheduling policies)

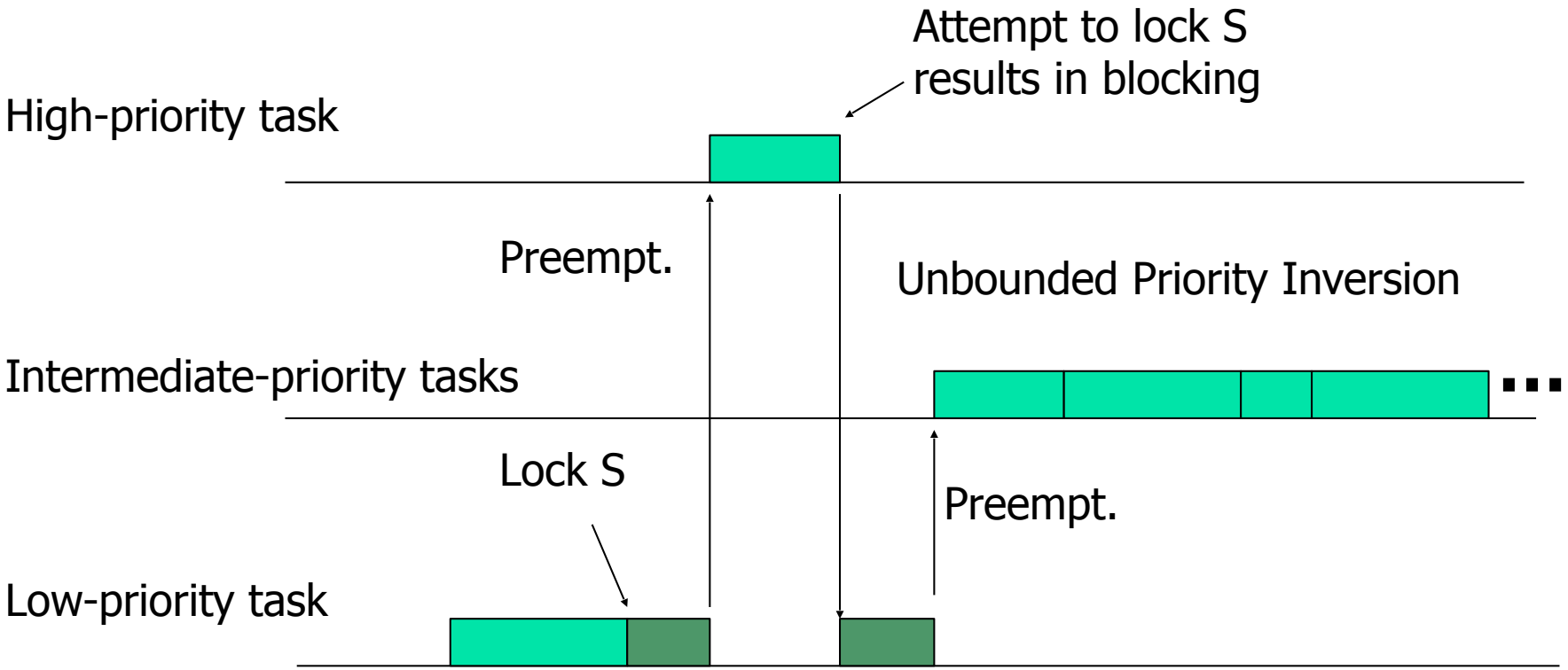
RTS: Priority Inversion

How should we account for priority inversion?



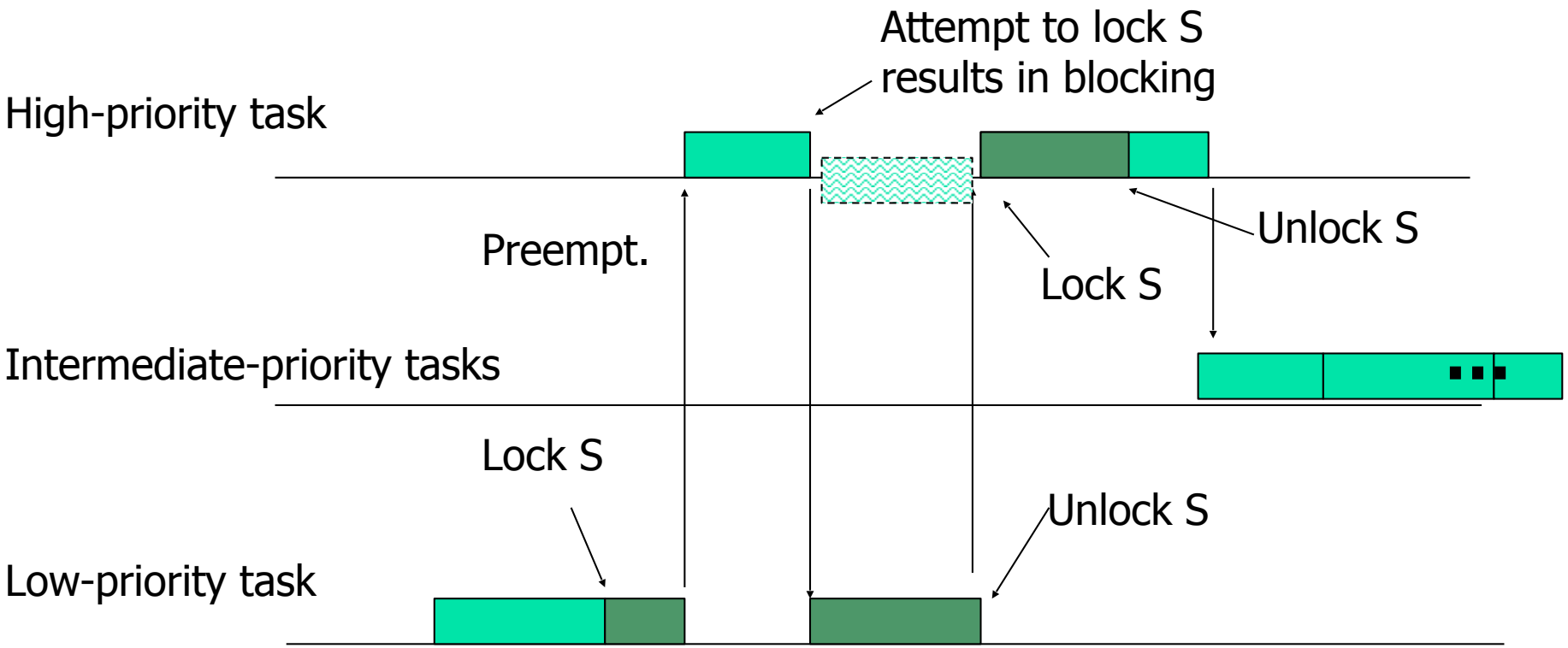
RTS: Unbounded Priority Inversion

Consider the case below: a series of intermediate priority tasks is delaying a higher-priority one



RTS: Priority Inheritance Protocol

Solution: Let a task inherit the priority of any higher-priority task it is blocking



RTS: Priority Ceiling Protocol

- Definition: The priority ceiling of a semaphore is the highest priority of any task that can lock it
- A task that requests a lock R_k is denied if its priority is not higher than the highest priority ceiling of all semaphores currently locked by other tasks (say it belongs to semaphore R_h)
  - The task is said to be blocked by the task holding lock R_h
- A task inherits the priority of the top higher-priority task it is blocking