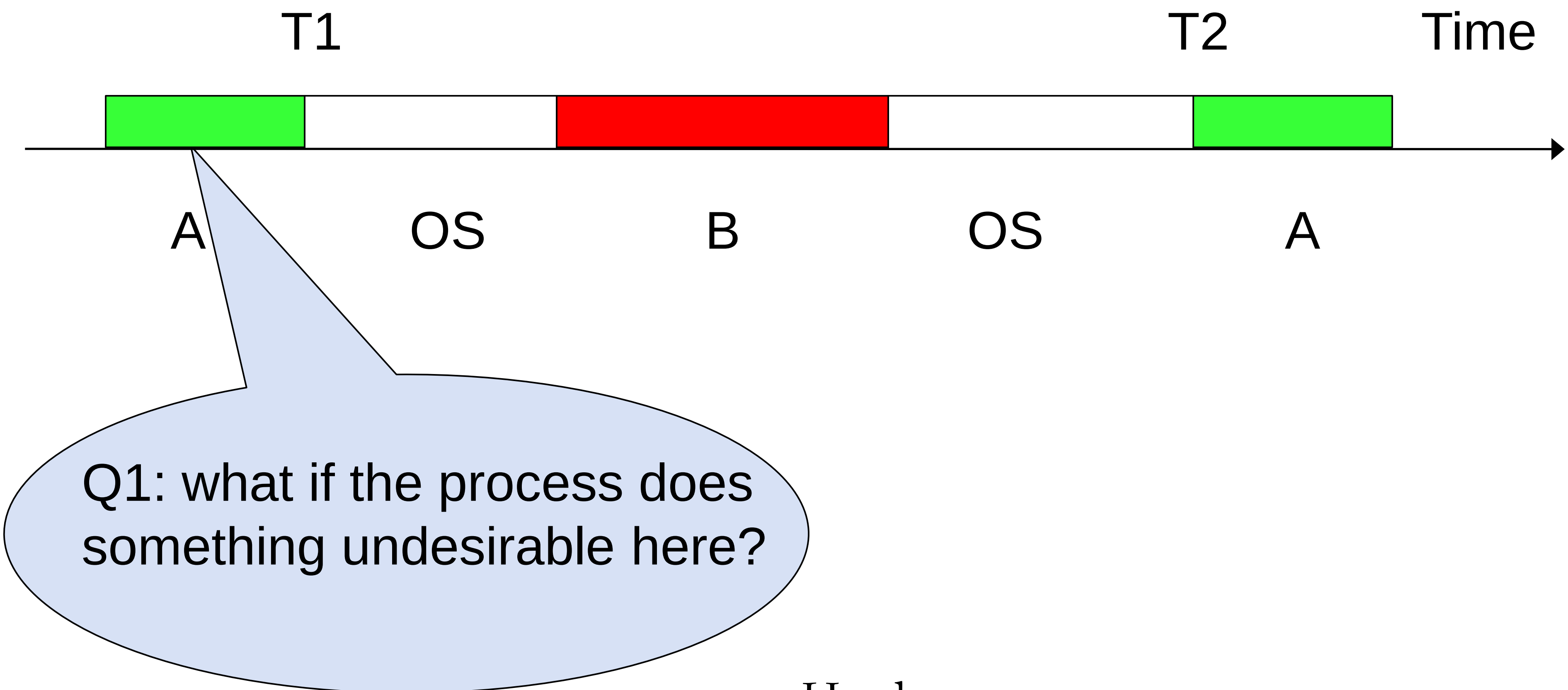
Operating Systems
CMPSC 473

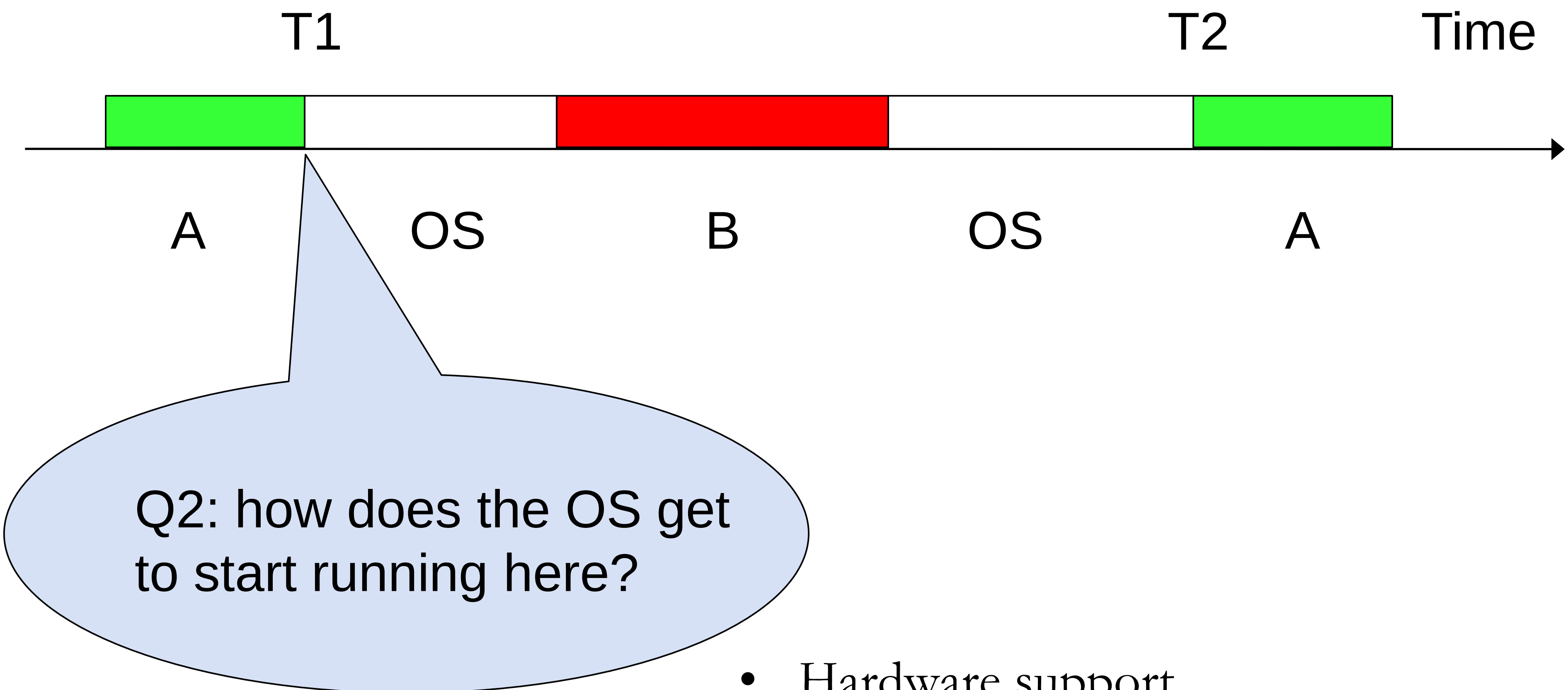
CPU Virtualization - Scheduling
Lecture 14: October 5, 2023

Instructor: Ruslan Nikolaev



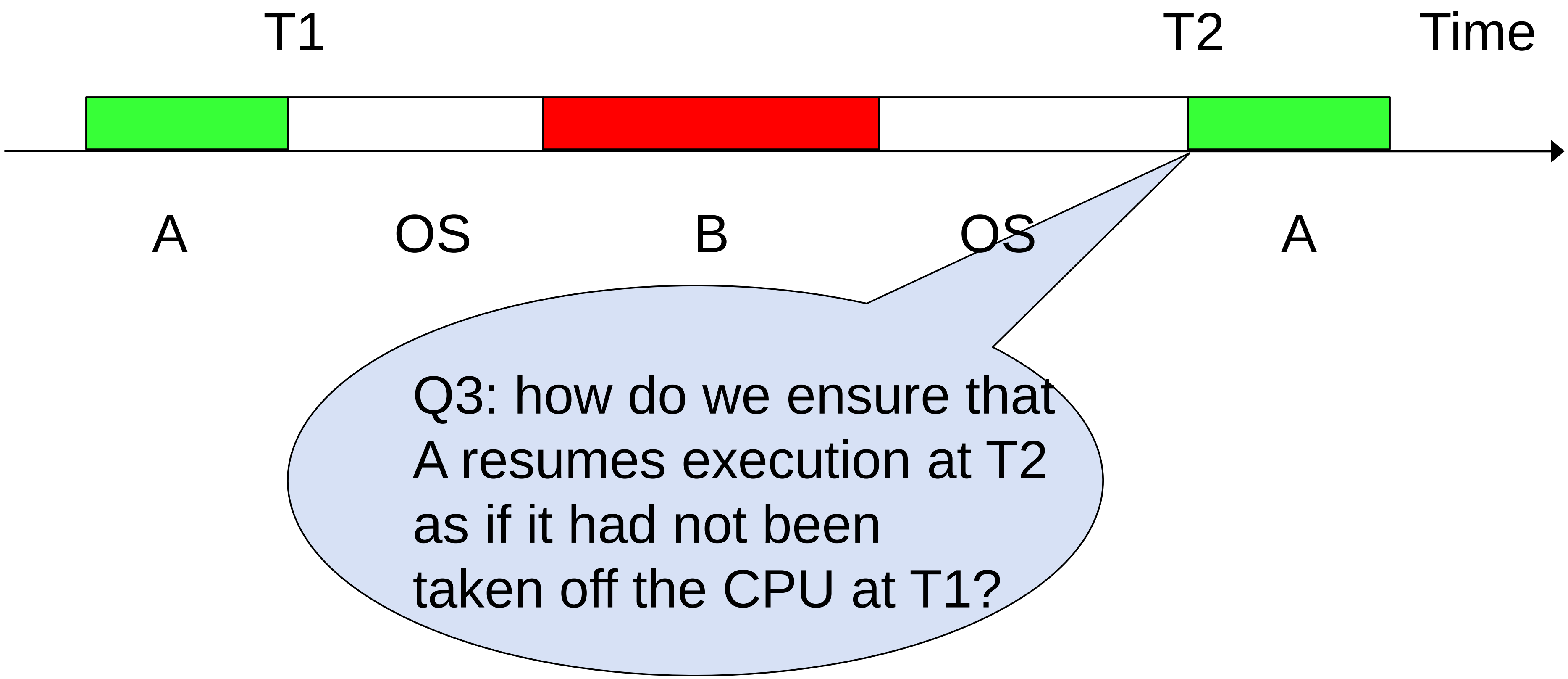
- Hardware support
  - Trap mechanism, CPU modes
  - Switching to the specified (per-CPU) kernel stack, saving a user stack register, and program counter
  - MMU, page tables, TLB

- OS support
  - Trap handlers, signal handling
  - Per-process kernel stack
  - Virtual memory, page tables

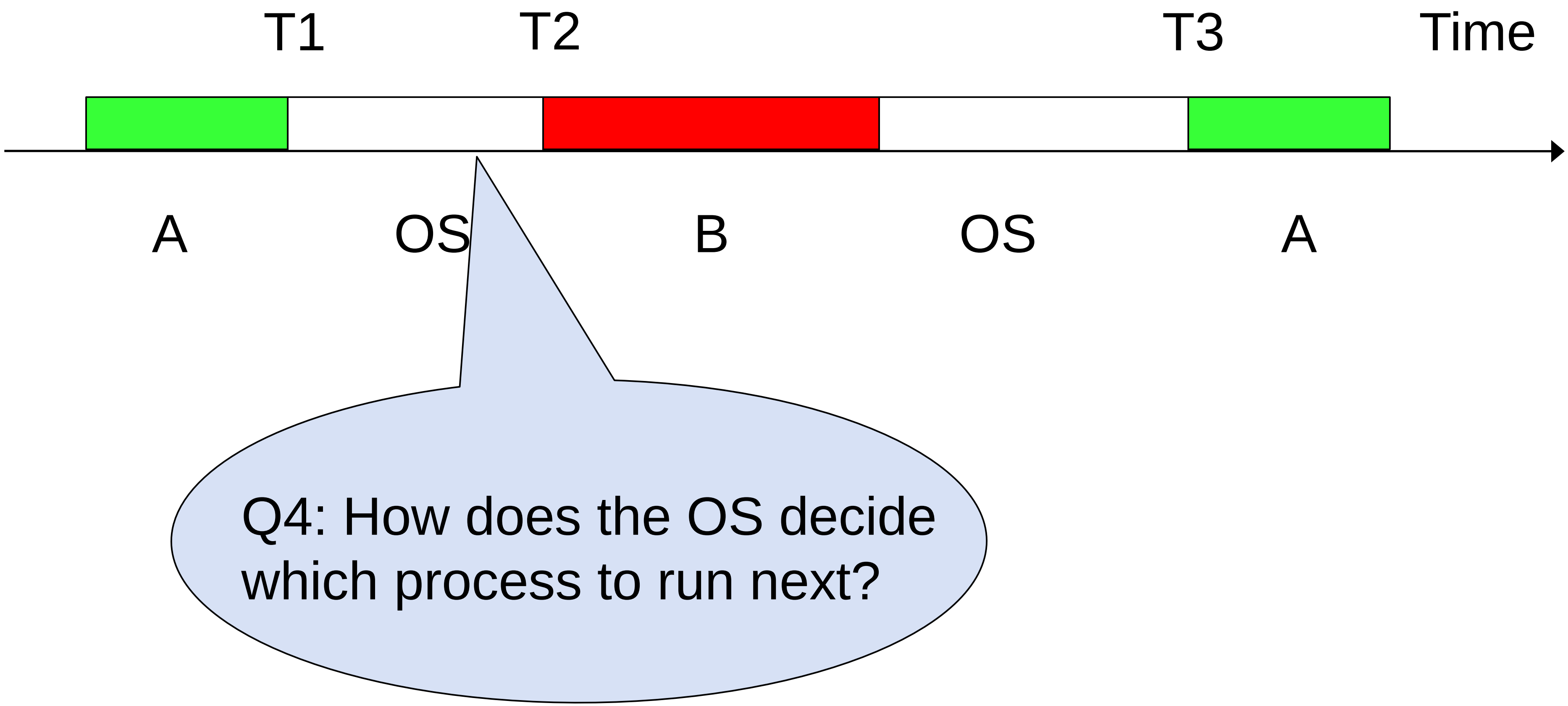


- Hardware support
  - Interrupt mechanism, e.g., a timer interrupt
  - Switching to the specified (per-CPU) kernel stack, saving a user stack register, and program counter

- OS support
  - Interrupt handlers
  - Per-process kernel stack

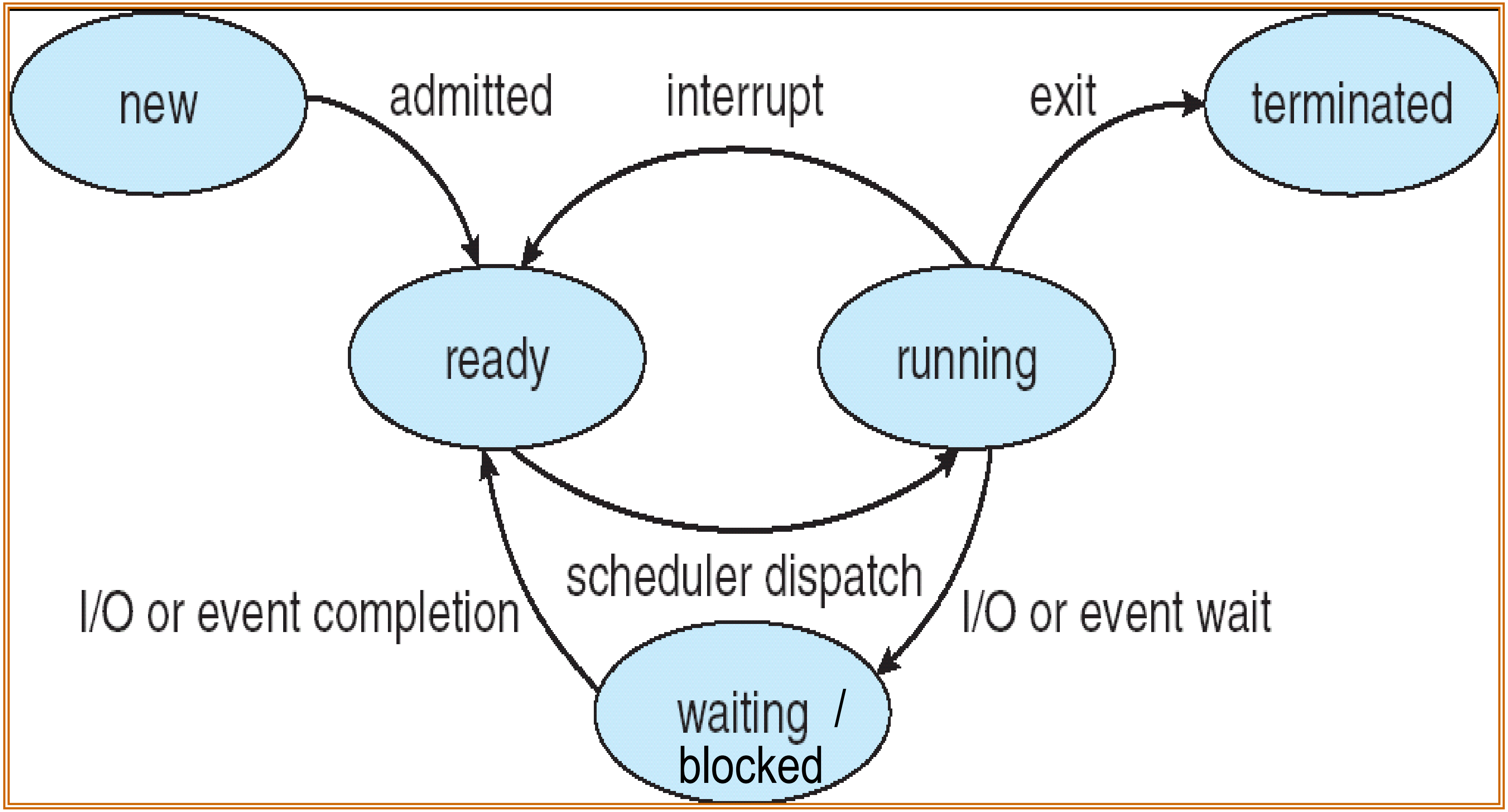


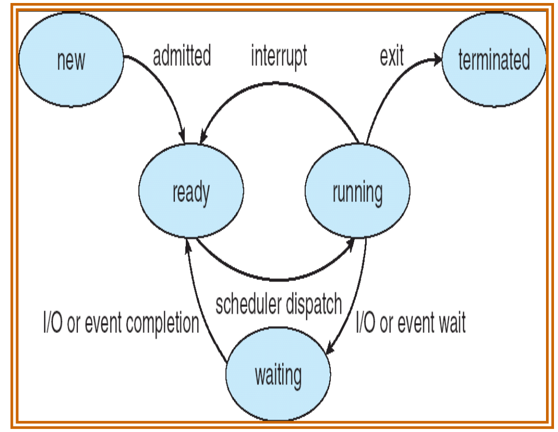
- OS support
  - Saving/restoring registers in/from PCB (process control block)
- Hardware support
  - Not necessary unless using "hardware context switching"

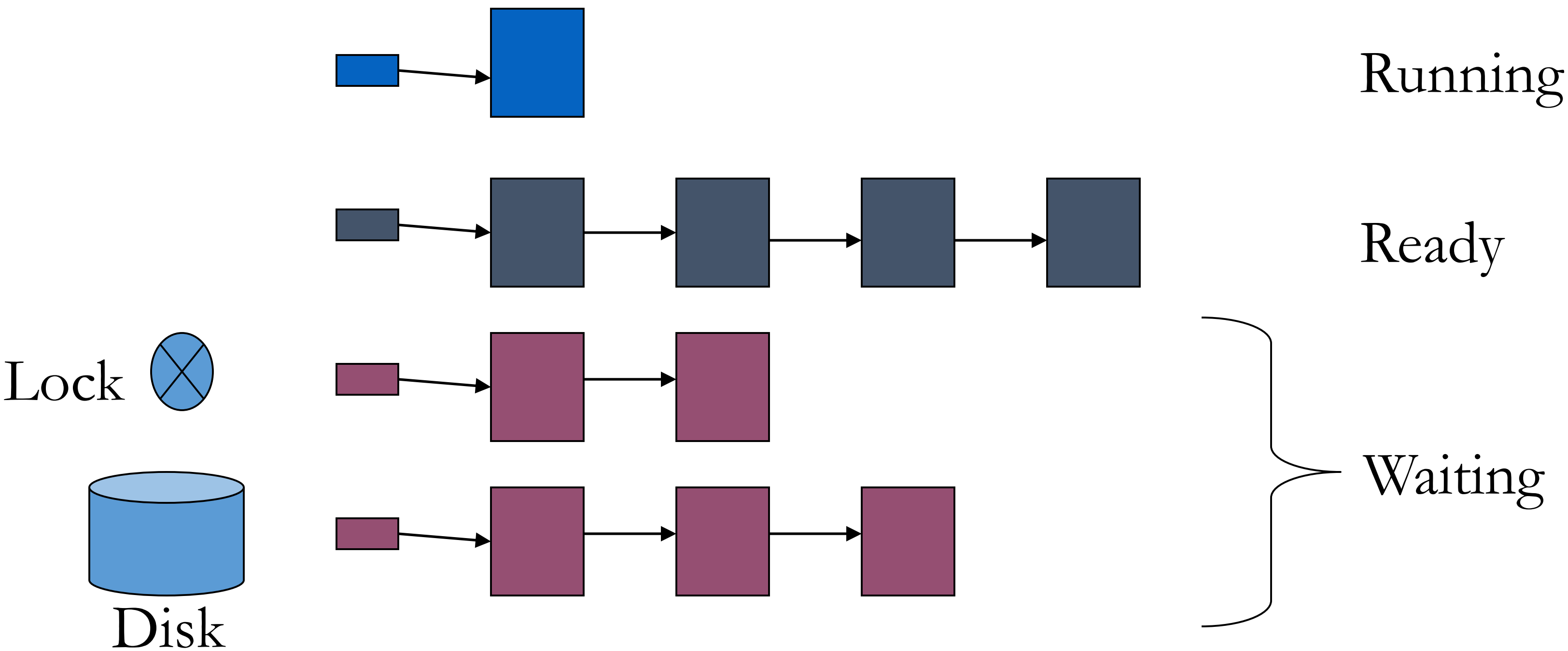


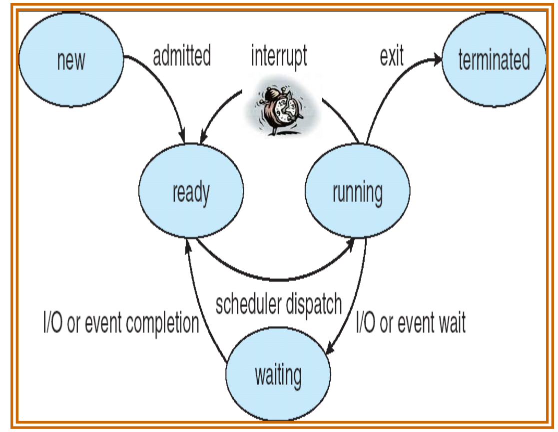
- The CPU scheduler

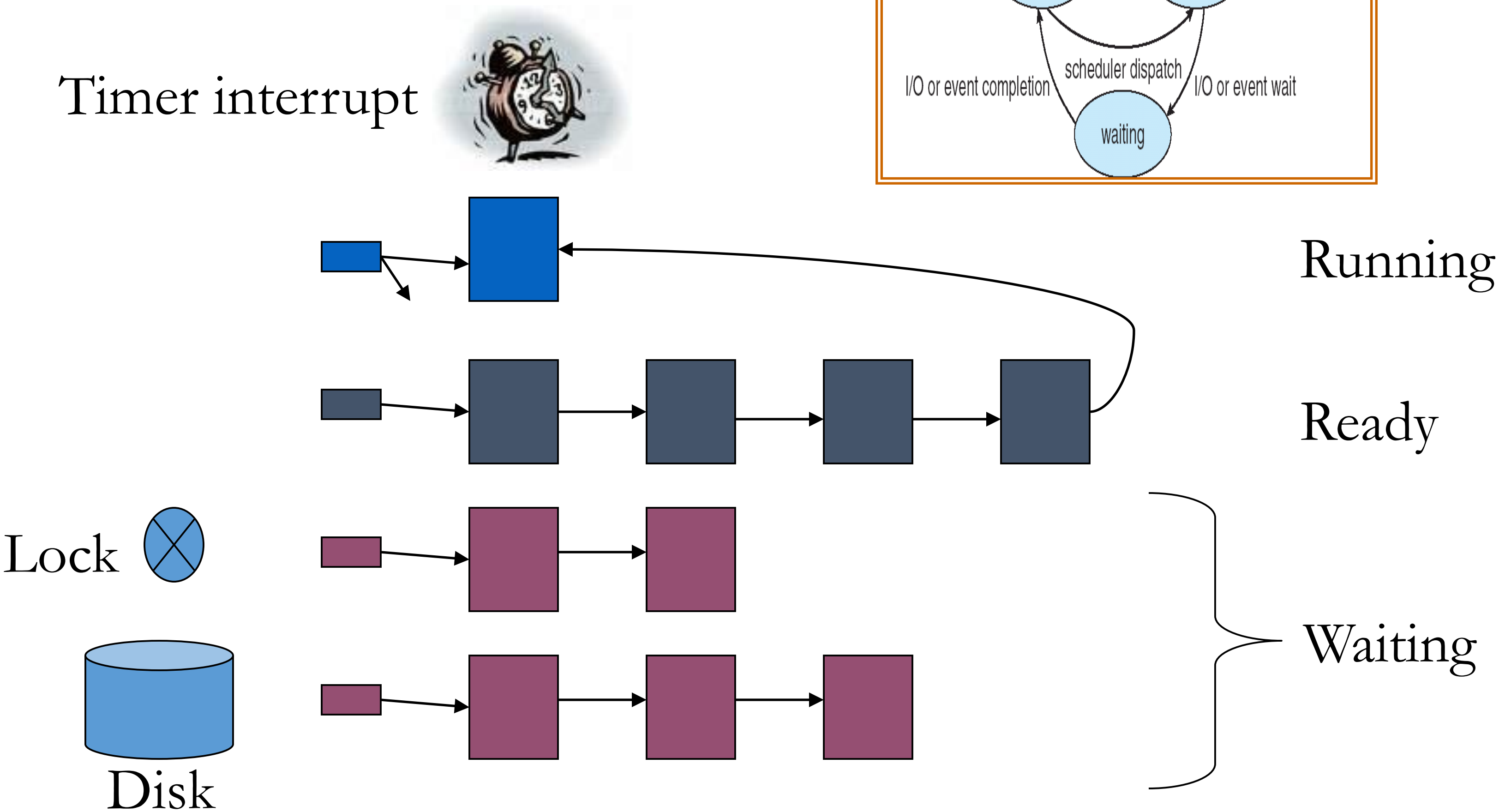
Scheduling states of a process/thread

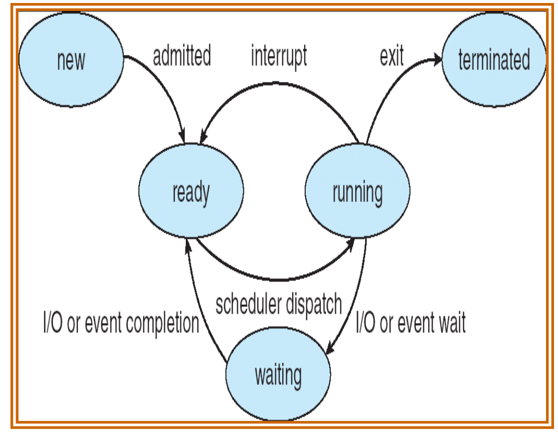


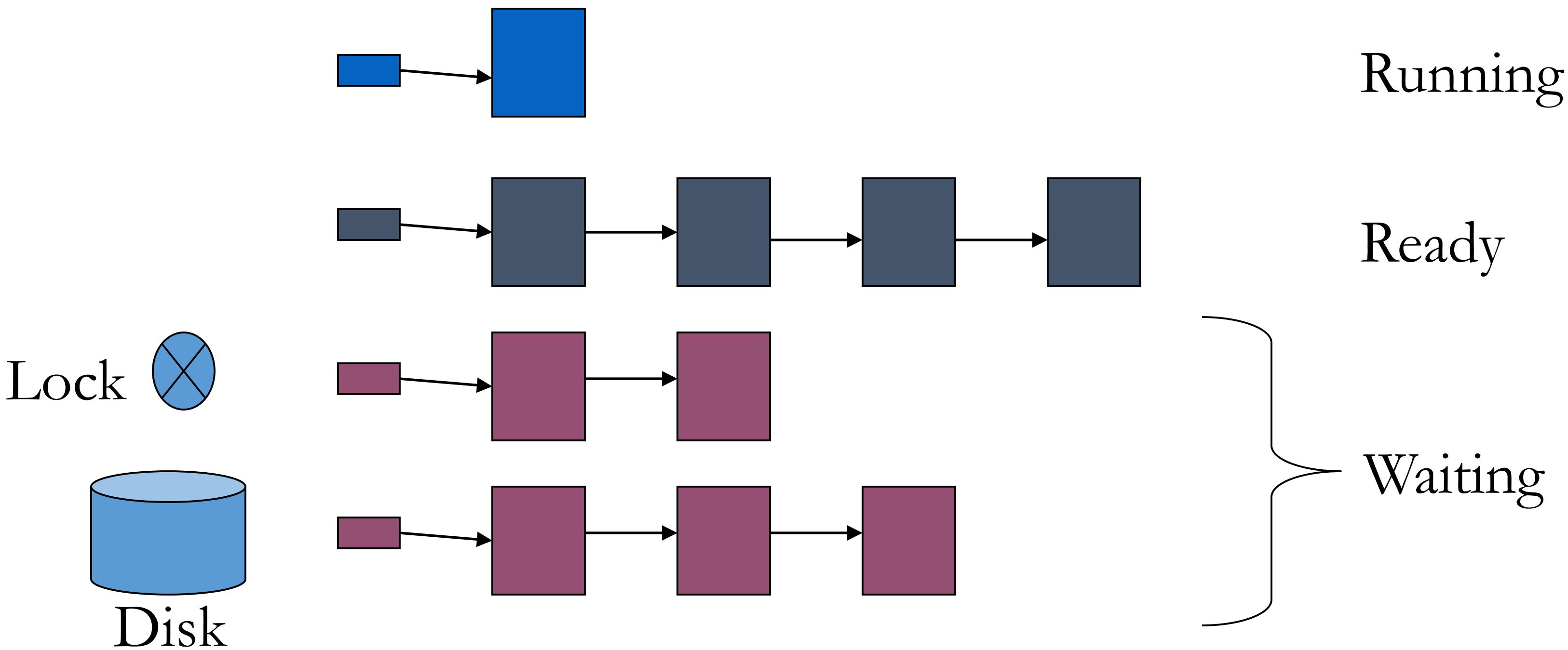

new
admitted
interrupt
exit
terminated
ready
running
scheduler dispatch
I/O or event completion
I/O or event wait
waiting

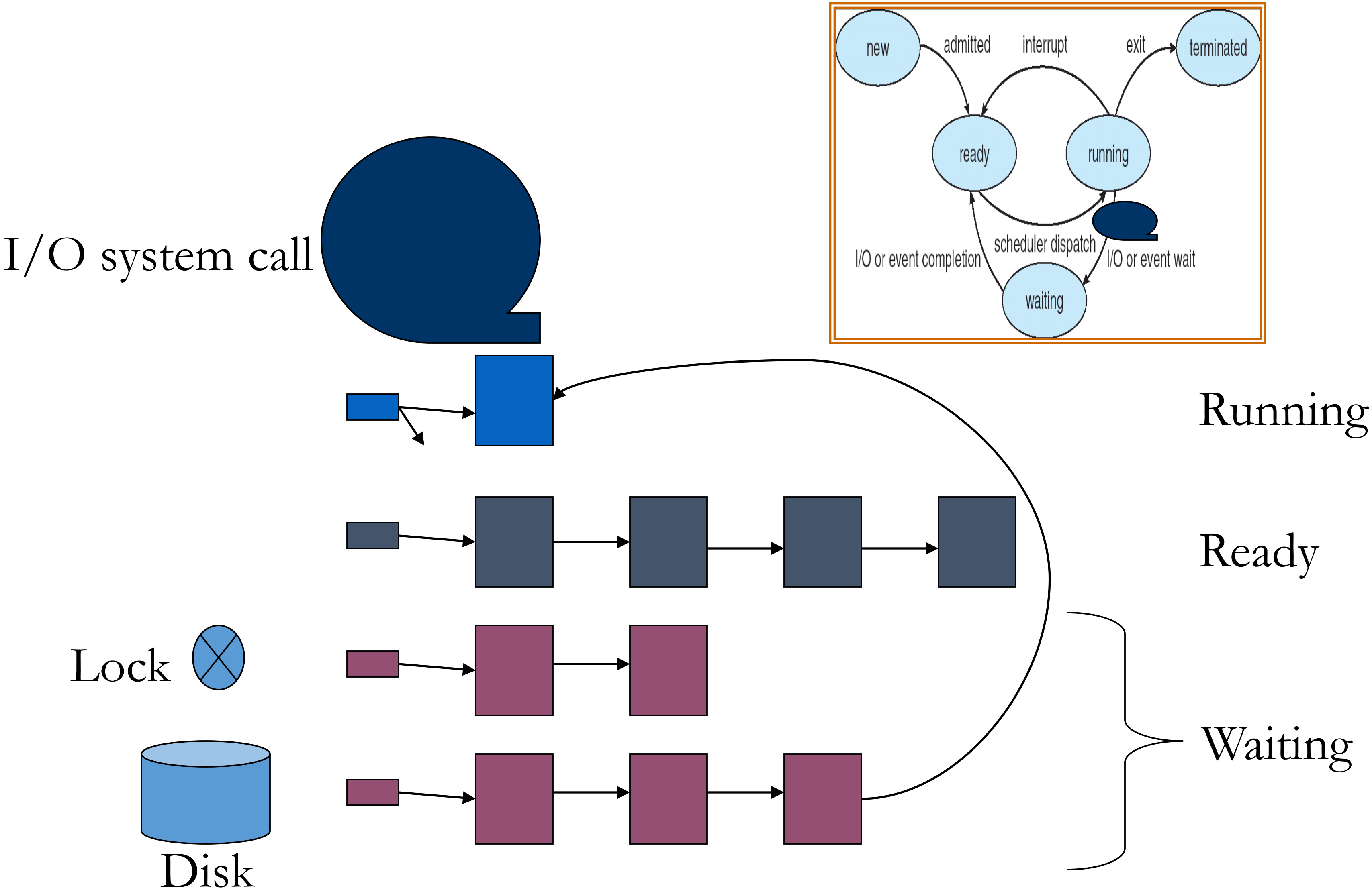

Running
Ready
Lock
Waiting
Disk


new
admitted
interrupt
exit
terminated
ready
running
scheduler dispatch
I/O or event completion
I/O or event wait
waiting

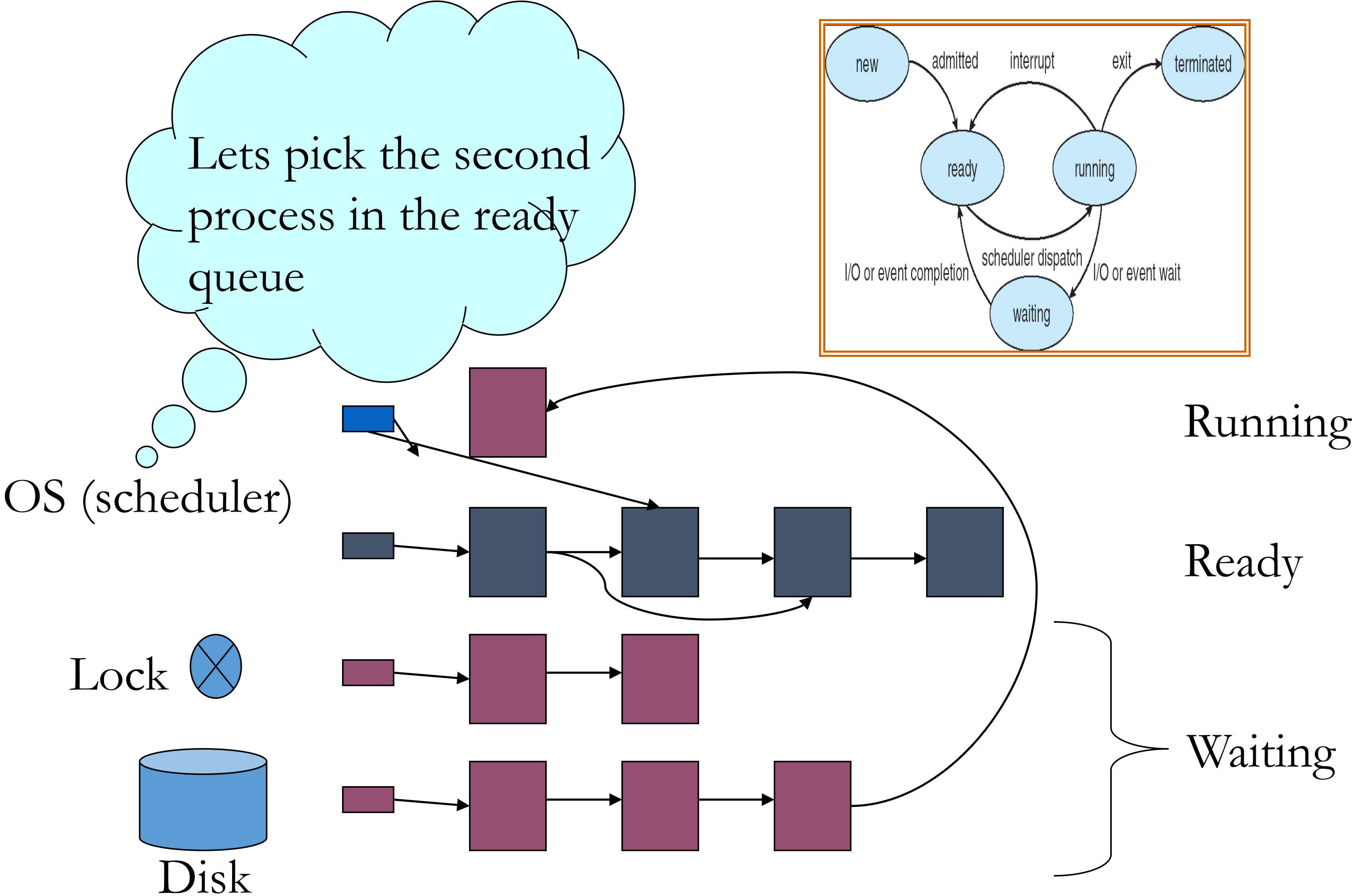

Timer interrupt
Running
Ready
Lock
Disk
Waiting


new
admitted
interrupt
exit
terminated
ready
running
scheduler dispatch
I/O or event completion
I/O or event wait
waiting


Running
Ready
Lock
Waiting
Disk



new
admitted
interrupt
exit
terminated
ready
running
scheduler dispatch
I/O or event completion
I/O or event wait
waiting
I/O system call
Running
Ready
Lock
Disk
Waiting


Lets pick the second process in the ready queue
OS (scheduler)
Running
Ready
Waiting
Lock
Disk
new
admitted
interrupt
exit
terminated
ready
running
scheduler dispatch
I/O or event completion
I/O or event wait
waiting

CPU scheduler: important concerns

- Optimization metric/criterion
  - Latency metrics:
    - Turnaround/completion time: time between when a "job" is submitted and when it finishes
    - Response time: time between when a process desires the CPU ("ready") and when it gets to run on the CPU ("running")
  - Throughput metrics: amount of "work" per unit time
    - e.g., # of requests or processes finishing per unit time
  - "Fairness"
    - Proportional-share
      - A job has a weight and gets resources proportional to its weight
    - Max-min
    - Many others
  - Combinations of these
- Overheads of the algorithm itself
  - Runtime and space complexity

FIFO/FCFS

- Overheads
  - Picking the next process to run
  - Process arrival, process departure
- How does it do for these metrics?
  - Response time
  - Throughput
  - Fairness
- Pros:
  - Simple to implement! Low overheads
  - Does well for “batch” jobs
- Cons:
  - Short jobs will suffer, a large job will monopolize the CPU

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- SJF is optimal for avg. waiting time – gives minimum average waiting time for a given set of processe

Why Preemption is Necessary

- To improve CPU utilization
  - Most processes are not *ready* at all times during their lifetimes
  - E.g., think of a text editor waiting for input from the keyboard
  - Also improves I/O utilization
- To improve responsiveness
  - Many processes would prefer to receive CPU quickly when they need it
- Modern CPU schedulers are preemptive

SJF: Variations on the theme

- **Non-preemptive**: once CPU given to the process it cannot be preempted until completes its CPU burst - the SJF we already saw
- **Preemptive**: if a new process arrives with CPU length less than remaining time of current executing process, preempt
  - This scheme is known as *Shortest-Remaining-Time-First (SRTF)*
  - Also called *Shortest Remaining Processing Time (SRPT)*
- Overheads?
  - Compare using unordered linked list vs. ordered list vs. heap
- Why SJF/SRTF may not be practical
  - CPU requirement of a process rarely known in advance

Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 1-10 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units
- Performance
  - q large => FCFS
  - q small => q must be large with respect to the context switch cost, otherwise overhead is too high

Proportional-Share Schedulers

- A generalization of round robin
- Process P_i given a CPU weight $w_i > 0$

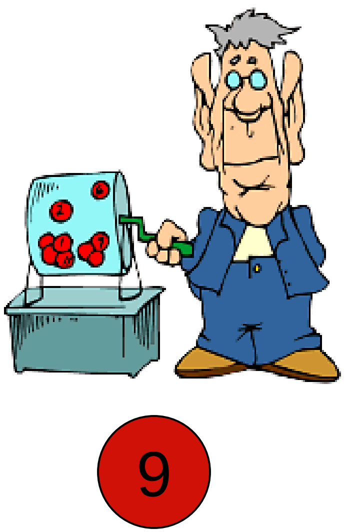
Lottery Scheduling

- Perhaps the simplest proportional-share scheduler
- Create lottery tickets equal to the sum of the weights of all processes
- Draw a lottery ticket and schedule the process that owns that ticket

Lottery Scheduling Example

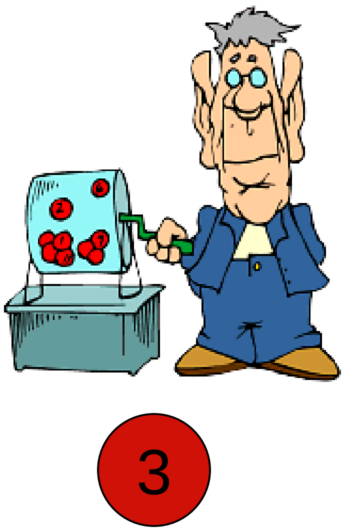
P1=6 *P2=9*

1	4		7	10	13
2	5		8	11	14
3	6		9	12	15



Schedule P2

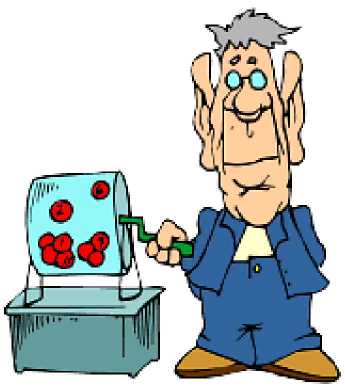
Lottery Scheduling Example



P1=6		*P2=9*		
1	4	7	10	13
2	5	8	11	14
3	6	9	12	15

Schedule P1

Lottery Scheduling Example



P1=6

1	4
2	5
3	6

P2=9

7	10	13
8	11	14
9	12	15

11

- As t $\longrightarrow \infty$, processes will get their share (unless they were blocked a lot)
- Problem with Lottery scheduling: Only probabilistic guarantee
- What does the scheduler have to do
  - When a new process arrives?
  - When a process terminates?

Schedule P2

Multi-Level Feedback Queue (MLFQ)

- See Chapter 8 of OSTEP
- Optional reading, not on syllabus
- Very old idea, a great example of heuristics that
  - Try to combine conflicting goals of good response times for interactive jobs and fairness
  - Learn dynamically the nature of a job (whether it is interactive)