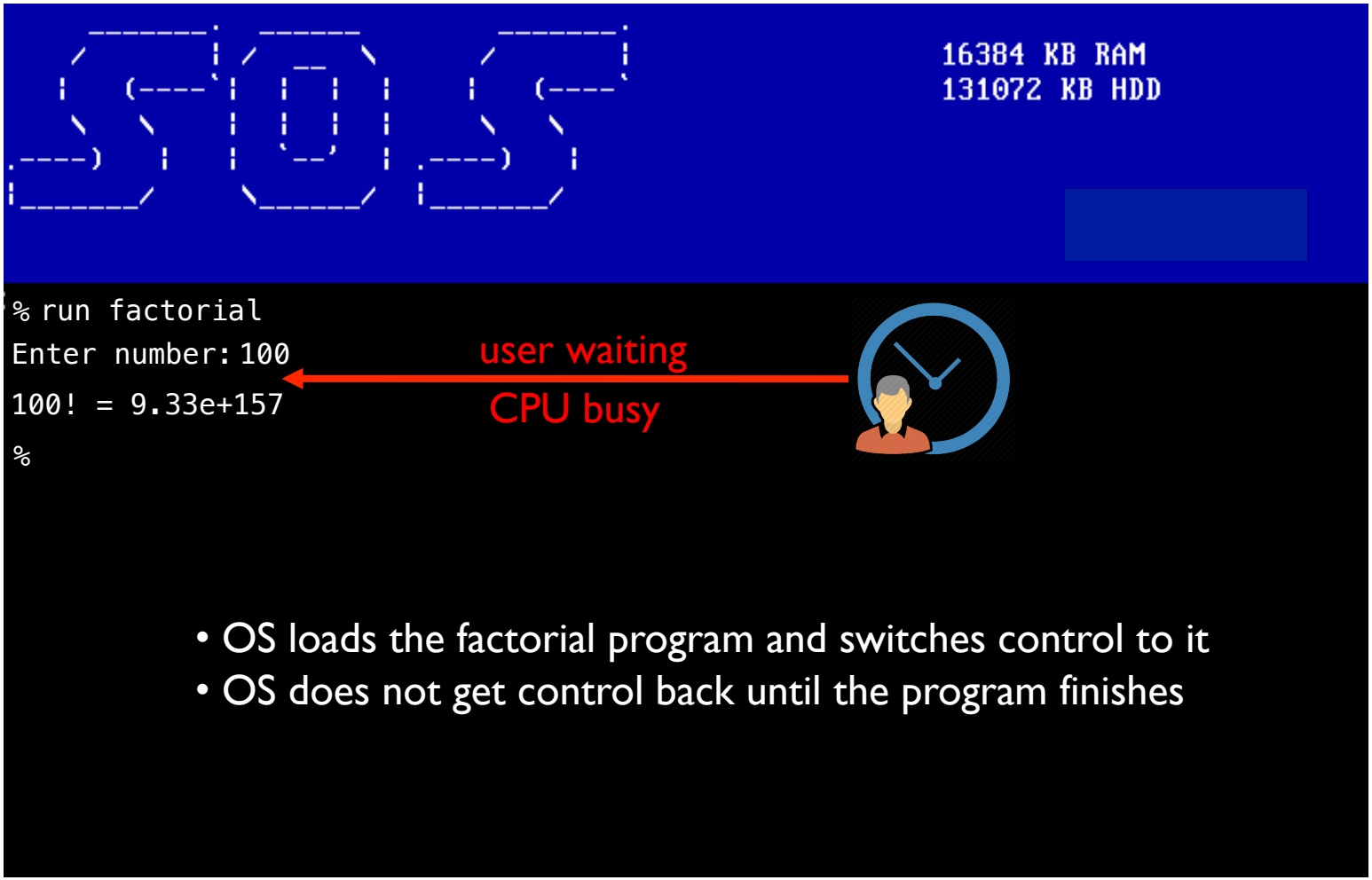
CPU Scheduling

COMP 3361: Operating Systems I

Winter 2015

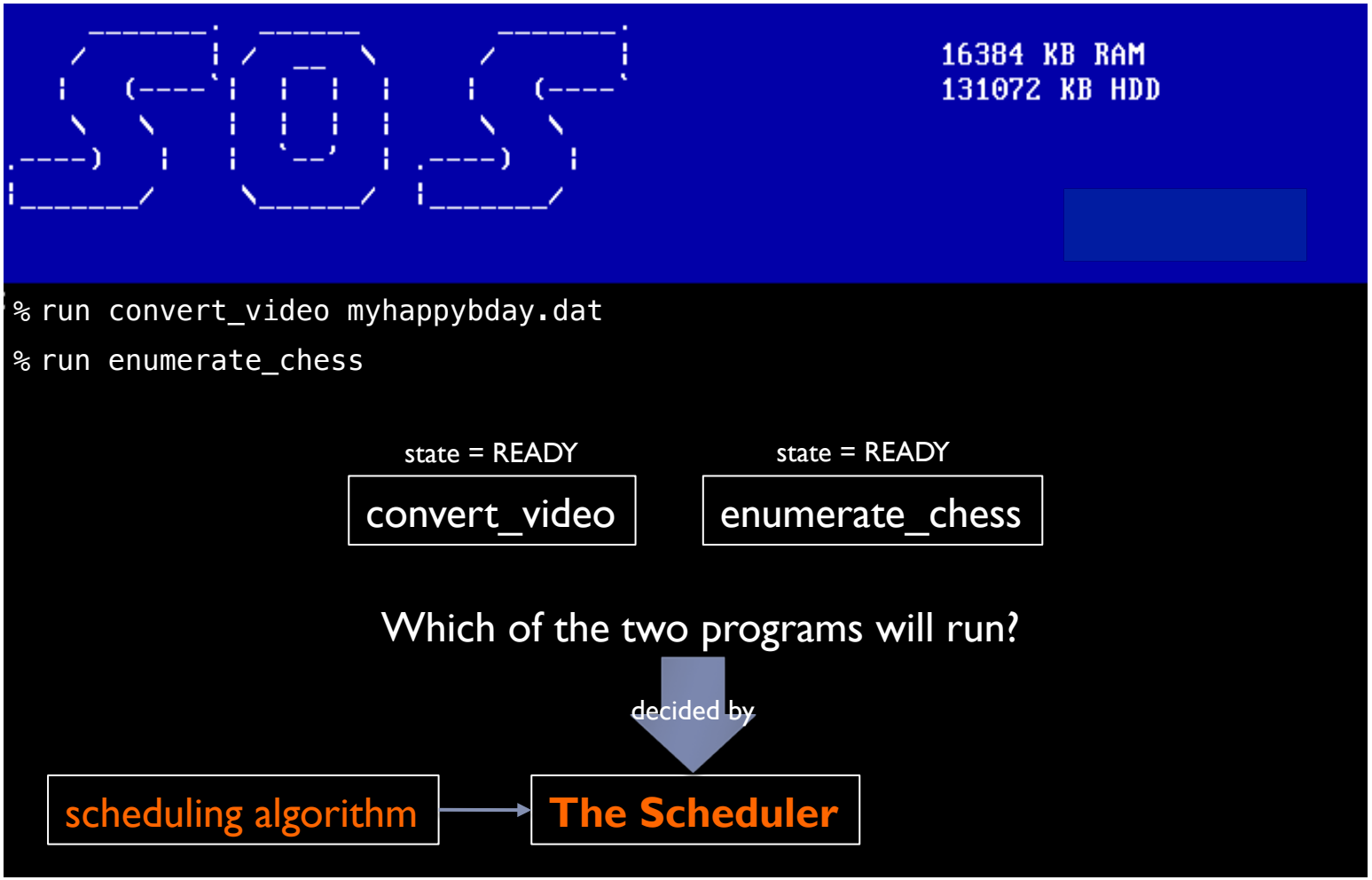
http://www.cs.du.edu/3361

1 Consider This Scenario

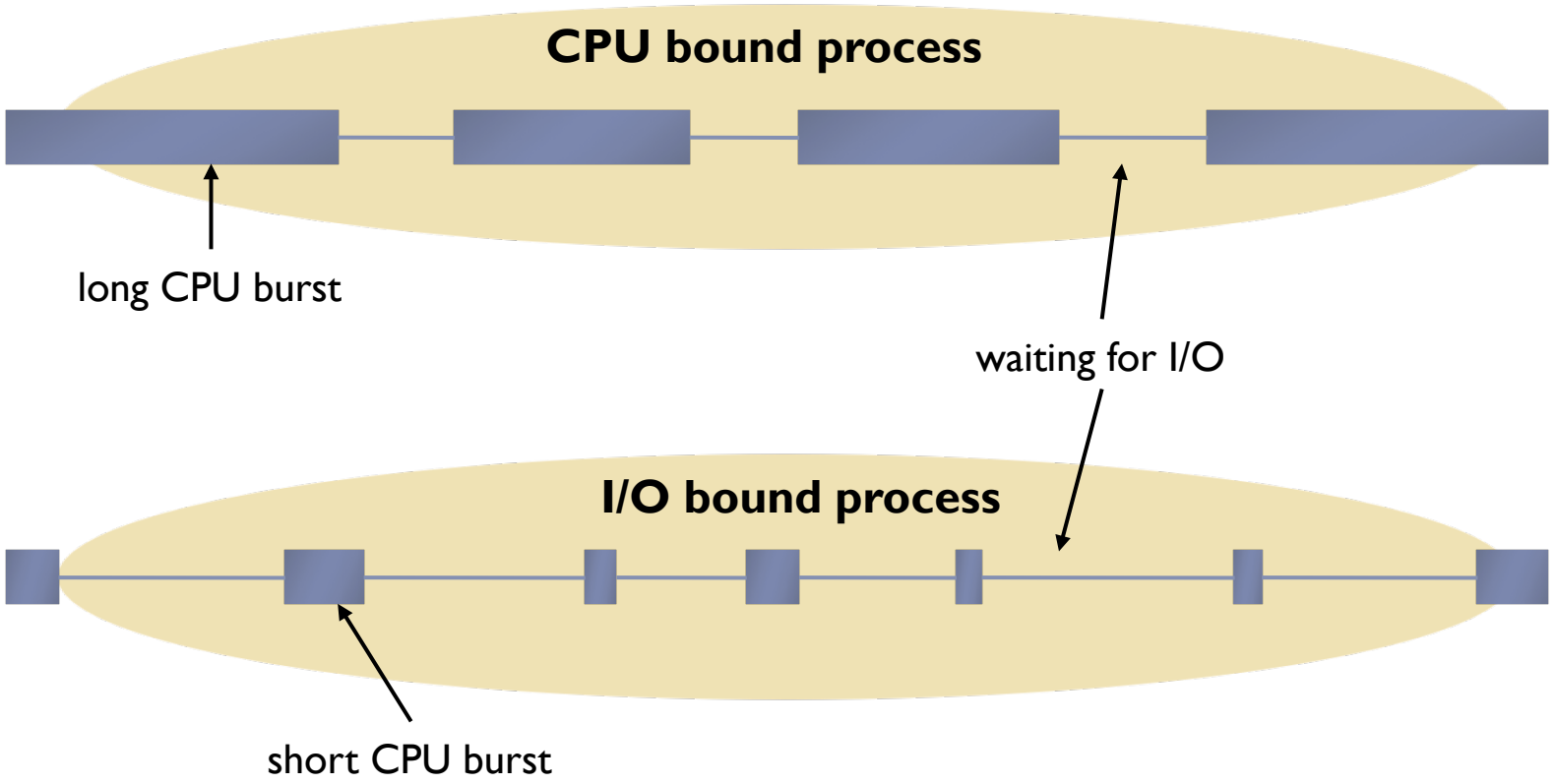


- OS loads the factorial program and switches control to it
- OS does not get control back until the program finishes

2 Now This Scenario



Process Behavior



4 Scheduler

- Selects from among the processes in the ready state and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process
  - is created
  - terminates
  - switches from running to waiting state
  - an interrupt occurs (I/O complete, timer, etc.)
- Operations performed to do process switching is pure **overhead**

Scheduling Types

- **Non-preemptive:** Once a process starts running, it is allowed to run until it blocks
  - after an interrupt is handled, CPU goes back to the process running prior to the interrupt
- **Preemptive:** A process may be forcibly removed from the CPU
  - after interrupt is handled, CPU **may not** go back to the process running prior to the interrupt

Invoking the Scheduler

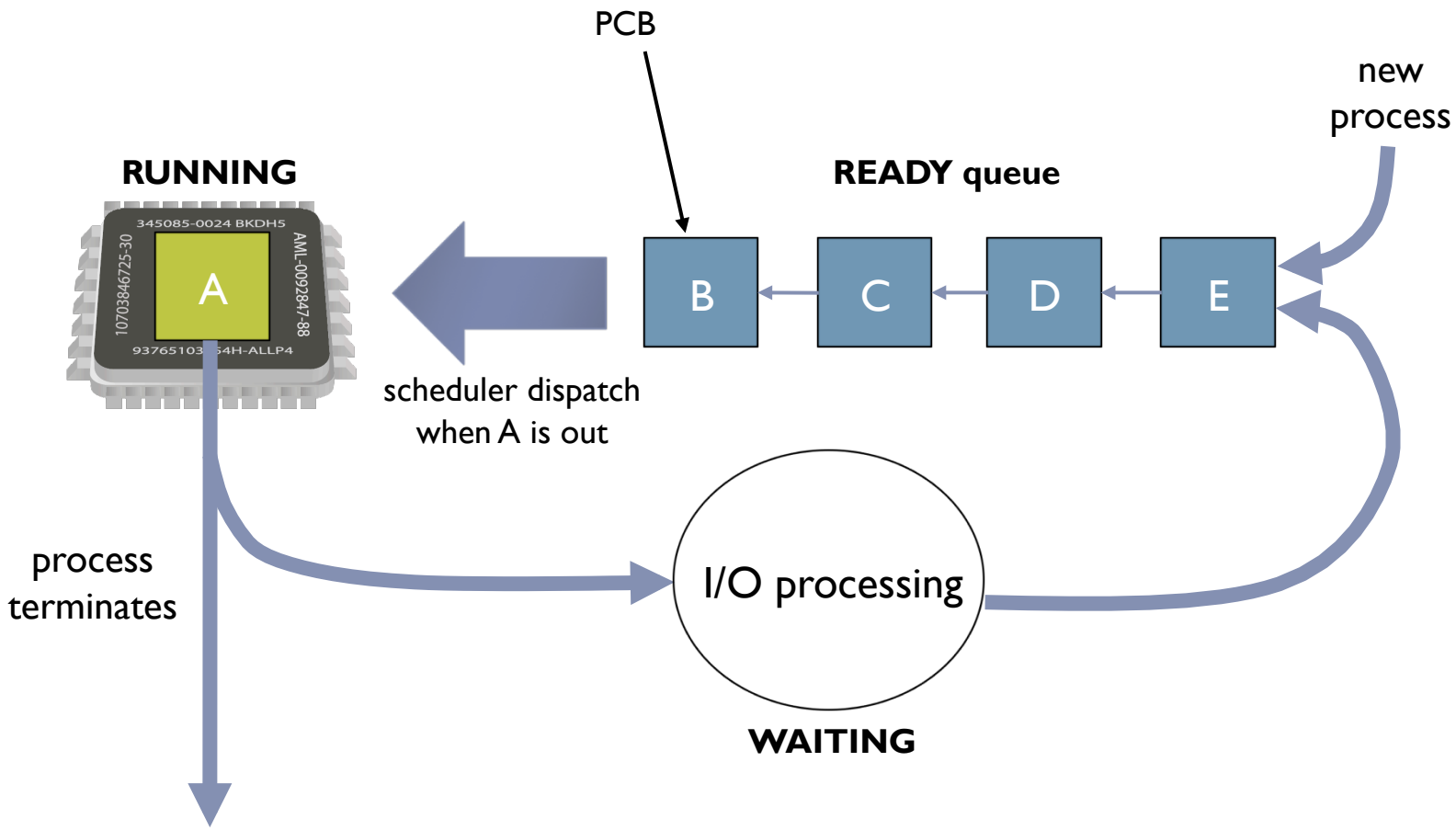
- Scenario:
  - user types command to start a program
  - the program starts running
  - user is unable to type anything on the console until program finishes
  - Why?

- We need to invoke the scheduler periodically
- **Timer** interrupts can be used to hand over control to the OS at regular intervals
  - configure timer hardware to generate an interrupt after fixed interval
  - timer interrupt handler invokes scheduler

7 Scheduling Algorithm Goals

- Any system
  - fairness, policy enforcement, balance
- Batch systems
  - throughput, turnaround time, CPU utilization
- Interactive systems
  - response time, proportionality
- Real-time systems
  - meeting deadlines, predictability

8 First-Come First-Served (FCFS)



First-Come First-Served (FCFS)

Process	Time on CPU
P_1	24
P_2	3
P_3	3

- Ready queue: P_1, P_2, P_3

Gantt Chart

- Waiting time for $P_1 = 0; P_2 = 24; P_3 = 27$
- Average waiting time = (0+24+27)/3 = 17

First-Come First-Served

- If ready queue is P_2, P_3, P_1

- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time = (6+0+3)/3 = 3
  - much better than the previous case
  - **convoy effect** is prevented
    - many I/O bound processes behind a CPU bound process

- Preemptive or non-preemptive?

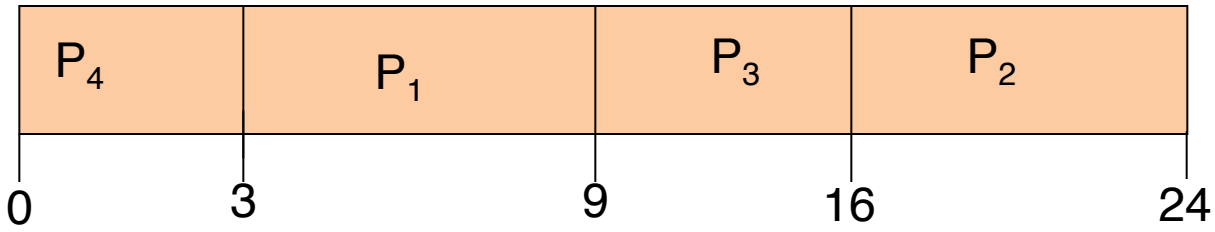
Shortest Job First (SJF)

- Associate with each process the length of its next CPU burst
- Use these lengths to first schedule the process with the shortest time
- SJF schedule gives the minimum average waiting time for a given set of processes
  - the difficulty is knowing the length of the next CPU burst

SJF Example

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF Gantt chart



- Average waiting time: (3+16+9+0)/4 = 7

Length of Next CPU Burst

- Can only estimate the length

- Can be done by using the length of previous CPU bursts, using exponential averaging

  - t_n = actual length of n^{th} CPU burst
  - τ_{n+1} = predicted value for the next CPU burst
  - $\alpha, 0 \le \alpha \le 1$
  - Define: $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$

Example of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$; recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = t_n$; only the actual last CPU burst counts
- If we expand the formula, we get:

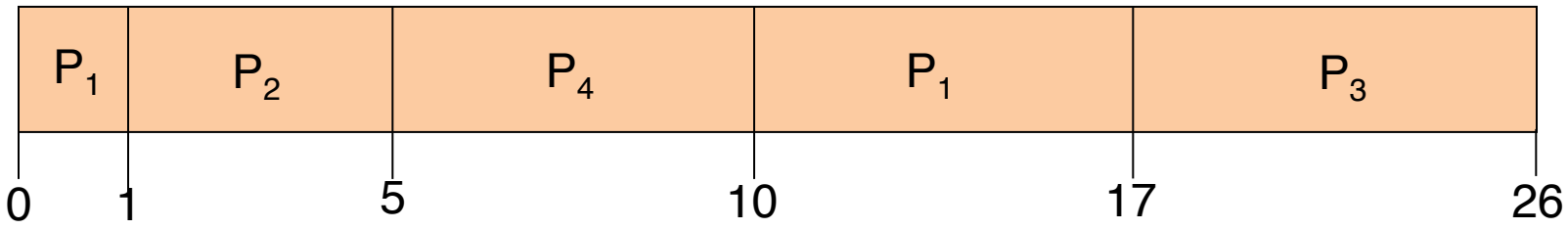
$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots$$
$$+(1 - \alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+(1 - \alpha)^{n+1} \tau_0$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

15 Shortest Remaining Time First

- Preemptive shortest job first
  - always run the job that can finish the earliest

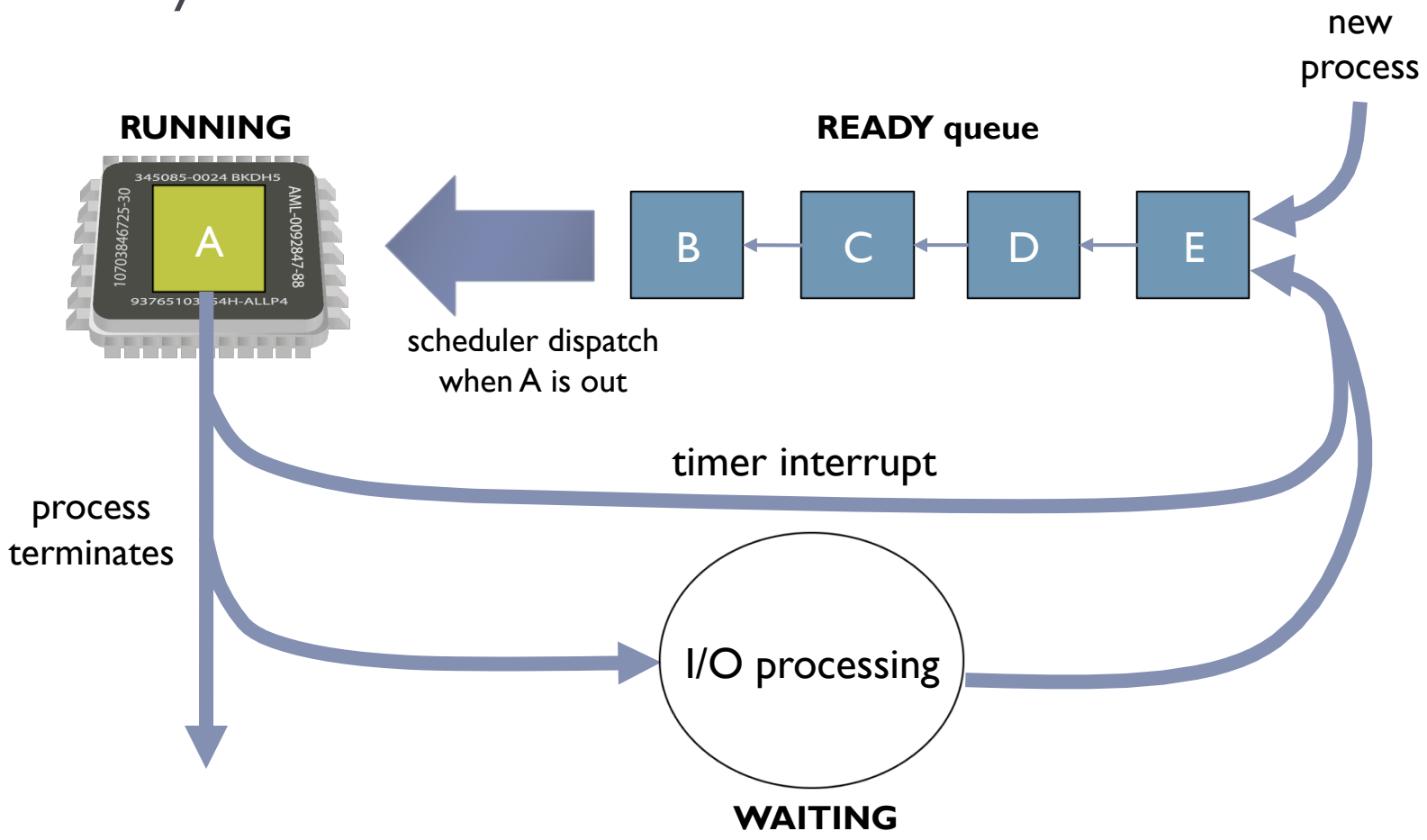
Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



average waiting time = ?

Round Robin (RR) Scheduling

- Each process gets a small unit of CPU time (**time quantum**)
  - usually 10-100 milliseconds



Round Robin (RR) Scheduling

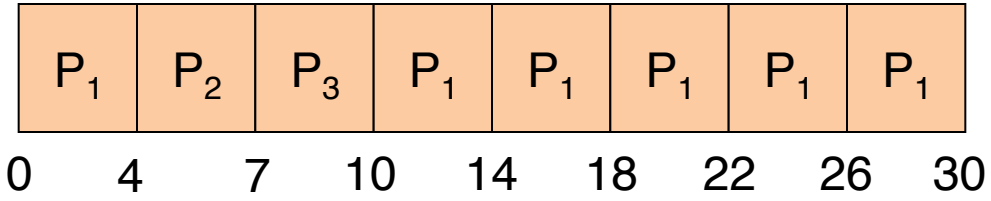
- Scheduler switches to next ready process in the queue
  - when the running process leaves the CPU voluntarily
  - when the timer hardware generates an interrupt

- Performance
  - *long quantum* $\Rightarrow$ FIFO, not good for interactive environments
  - *short quantum* $\Rightarrow$ quantum must be large with respect to context switch time, otherwise overhead is too high

RR Example

time quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3



- Higher average turnaround than SJF, but better *response*

RR with CPU Shares

- Process P_1, P_2 and P_3
- Assume that P_1 is guaranteed a 50% CPU share; the remaining is divided between the rest
- Round Robin schedules

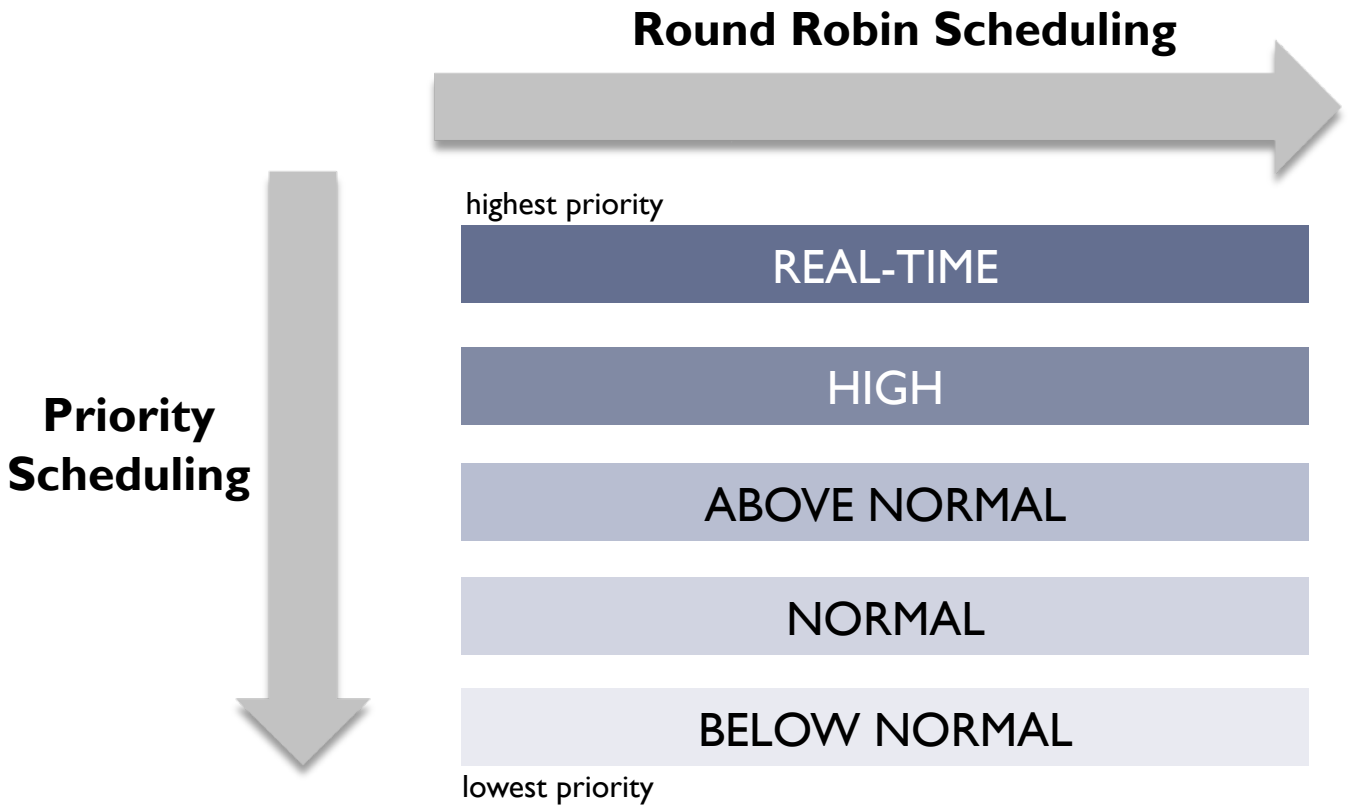
P_1	P_1	P_2	P_3	P_1	P_1	P_2	P_3	P_1	...

P_1	P_2	P_1	P_3	P_1	P_2	P_1	P_3	P_1	...

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority
  - preemptive or non-preemptive?
  - SJF is a priority scheduler where priority is the predicted next CPU burst time
- Assign high priority to I/O bound processes - Why?

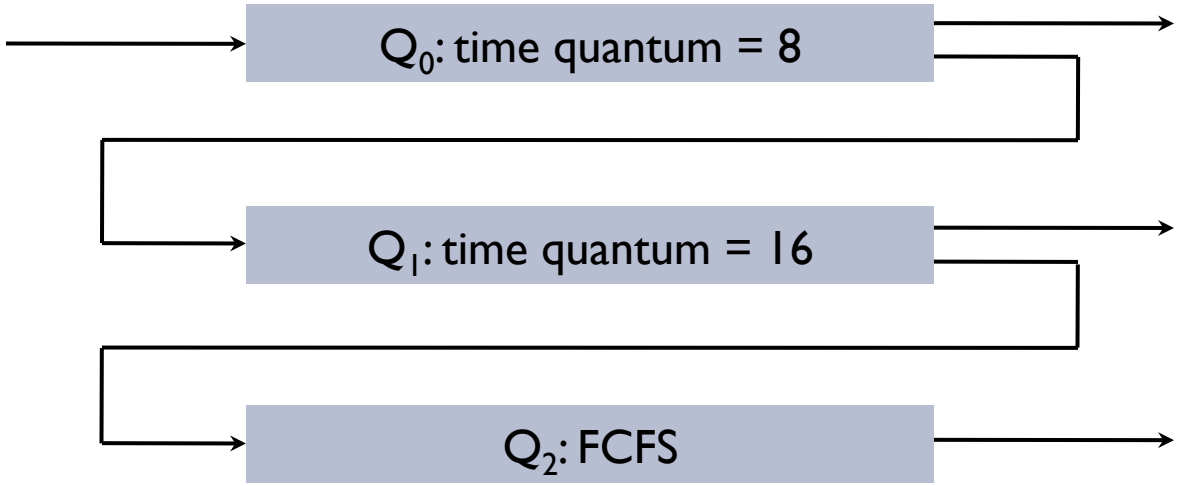
Priority Classes



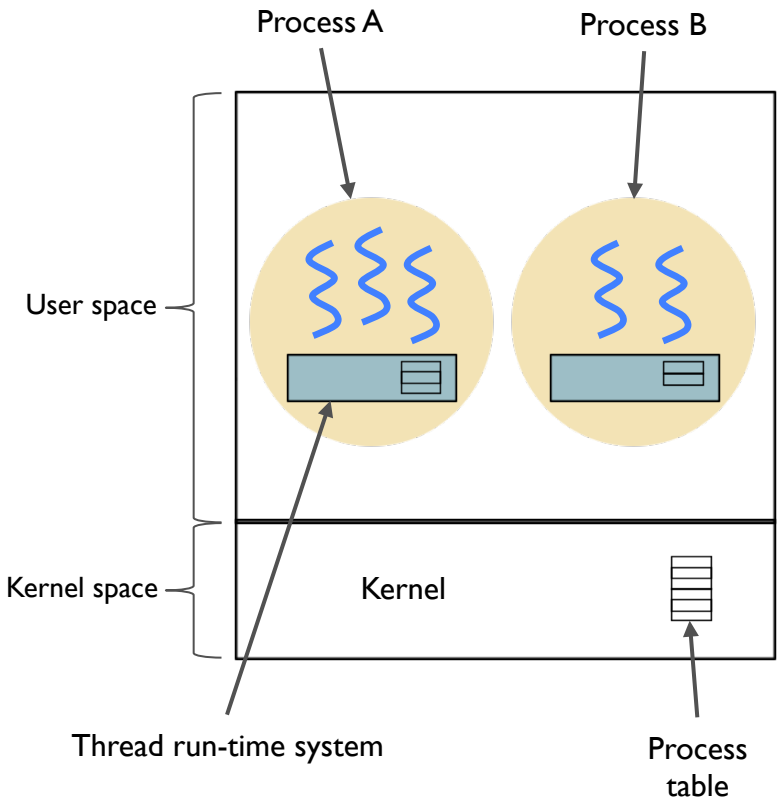
READY queues

Multiple Queues

- Three queues
  - Q_0 : RR with time quantum 8 milliseconds
  - Q_1 : RR with time quantum 16 milliseconds
  - Q_2 : FCFS



23 Scheduling Threads in User Space

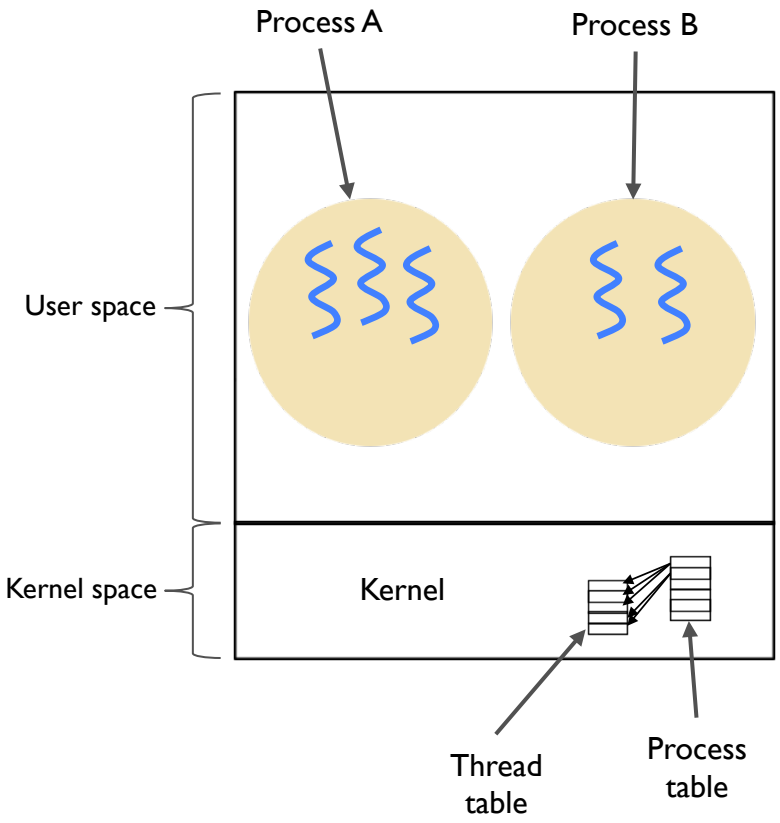


1. Kernel scheduler picks process A to run
2. Scheduler in thread run-time system decides the order in which process A's threads run in the time quantum
3. Kernel scheduler picks process B to run
4. Scheduler in thread run-time system decides the order in which process B's threads run in the time quantum

likely order: $A_1, A_2, A_3, A_1, A_2, A_3, A_1, A_2, A_3$

unlikely order: $A_1, A_2, A_3, B_1, B_2, A_1, A_2, A_3, B_1$

24 Scheduling Threads in Kernel Space



1. Kernel scheduler picks a thread from the thread table

2. When the time quantum ends, kernel scheduler picks another thread from the thread table
   - may belong to same process, or to a different one

likely order: $A_1, A_2, A_3, B_1, B_2, A_1, A_2, A_3, B_1$

also possible: $A_1, B_2, B_1, A_1, B_1, A_3, A_2, B_2, A_1$

References

- Chapter 2.4, Modern Operating Systems, A. Tanenbaum and H. Bos, 4th Edition.