Topics: Process (CPU) Scheduling
(SGG[9ed] 6.1-6.3)
(SGG[older ed] 5.1-5.3)

CS 3733 Operating Systems

Instructor: Dr. Turgay Korkmaz
Department Computer Science
The University of Texas at San Antonio

Office:	**NPB 3.330**
Phone:	**(210) 458-7346**
Fax:	**(210) 458-4437**
e-mail:	**korkmaz@cs.utsa.edu**
web:	**www.cs.utsa.edu/~korkmaz**

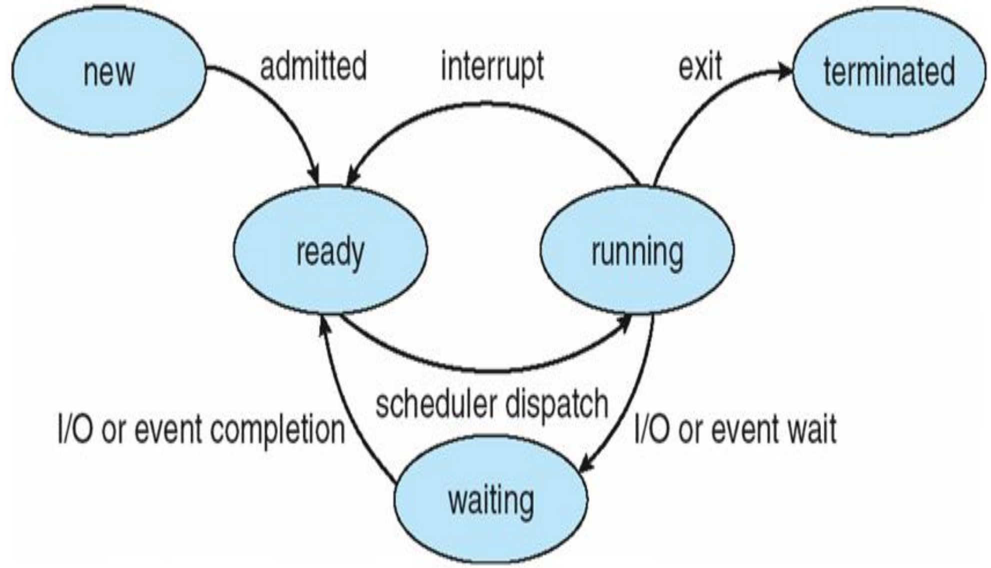
These slides are prepared based on the materials provided by Drs. Robbins, Zhu, and Liu. Thanks to all.

Outline

- Reviews on process
- Process queues and scheduling events
- Different levels of schedulers
- Preemptive vs. non-preemptive
- Context switches and dispatcher
- Performance criteria
  - Fairness, efficiency, waiting time, response time, throughput, and turnaround time;
- Classical schedulers: FIFO, SFJ, RR, and PR
- CPU Gantt chart vs. process Gantt charts

Reviews

- Process
  - Execution of program
- States of Process
  - **New** and **terminated**
  - **Running (R)**→ using CPU
  - **ready (r)**→ in memory, ready for the CPU
  - **waiting (w)**→ waiting for I/O device or interrupt

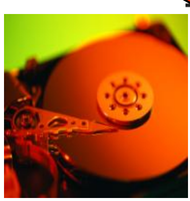


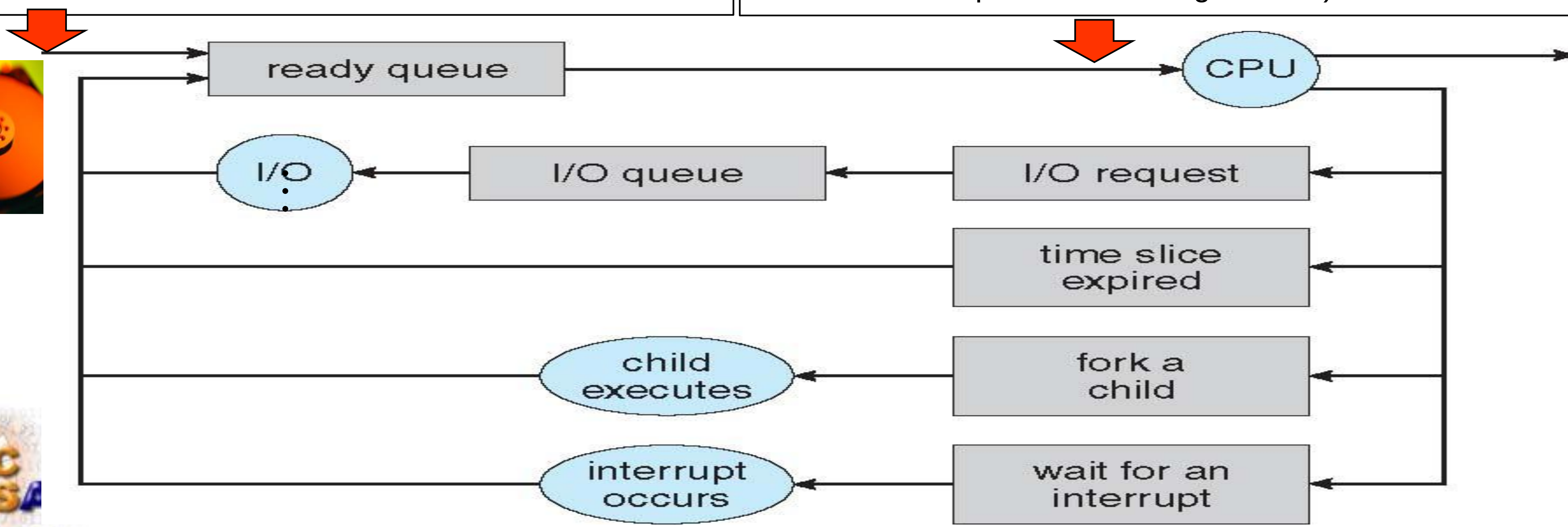
What may cause a process to move out of the CPU?

Different Levels of Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
  - Less frequent
  - Controls degree of multiprogramming

- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - More frequent (e.g., every 100 ms)
  - Must be fast (if it takes 10ms, then we have ~10% performance degradation)

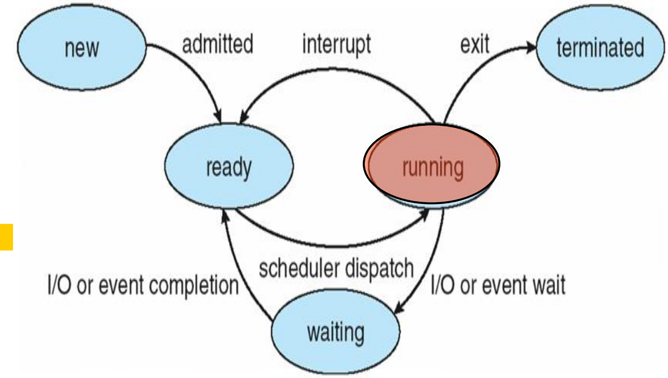




Different Levels of Schedulers

- **Short-term (CPU) scheduler**
  - **Selects which process from ready queue(s);**
  - **Must operate frequently and fast, several times a second**
- Medium-term scheduler (for time-sharing systems)
  - Swapping --- moving processes in and out of memory
  - Too many → lots of paging with decreased performance
- Long-term (job) scheduler
  - Decide which processes are admitted to the system
  - Determines the **degree of multiprogramming**
  - In stable conditions: invoke only when a process terminates
  - May take long time; for batch systems; may not be good for time-sharing systems;

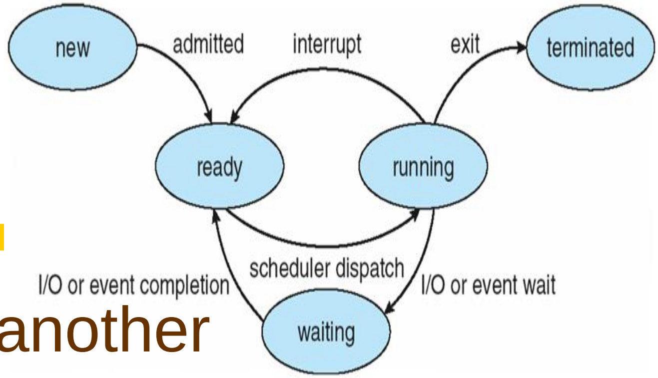
When to move out of CPU?



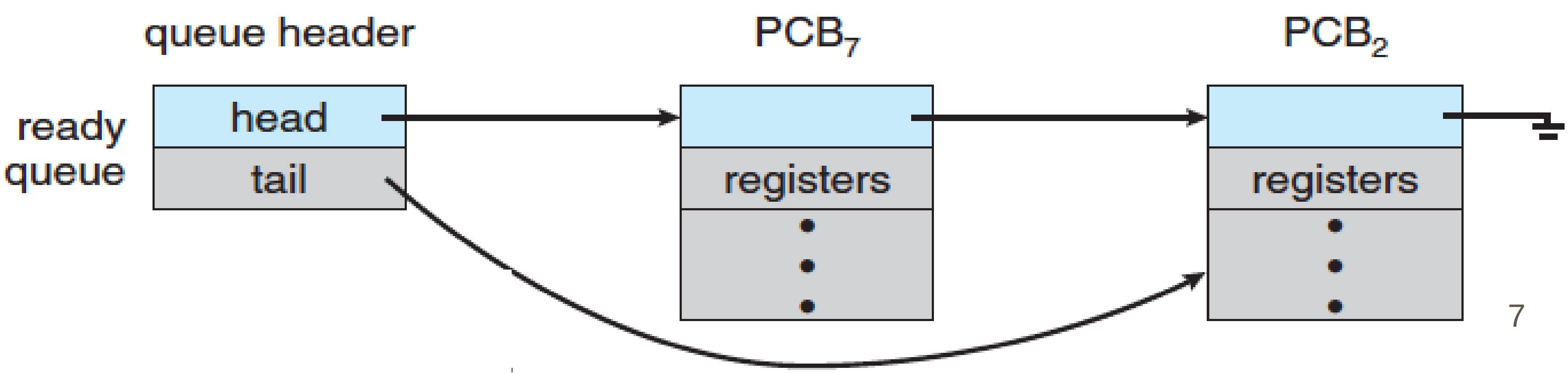
- I/O request
  - Need to read/write data from/to a file (on disk)
- Invoke **`fork`** and wait for child process
  - When create a new process with **`fork`**
  - **`fork`** → return to both parent/child processes with different return values;
    - ✓ Parent gets child process ID while Child gets 0;
- An interrupt or signal
  - Timer interrupt: when time quantum is used up
  - Signal
  - Other synchronization like wait/join

Not every system call results in context-switch

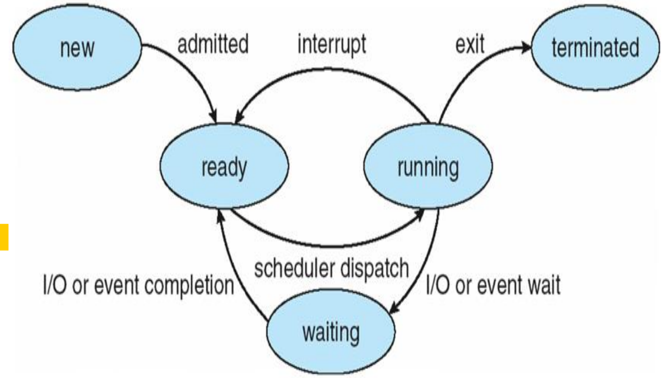
Process Queues



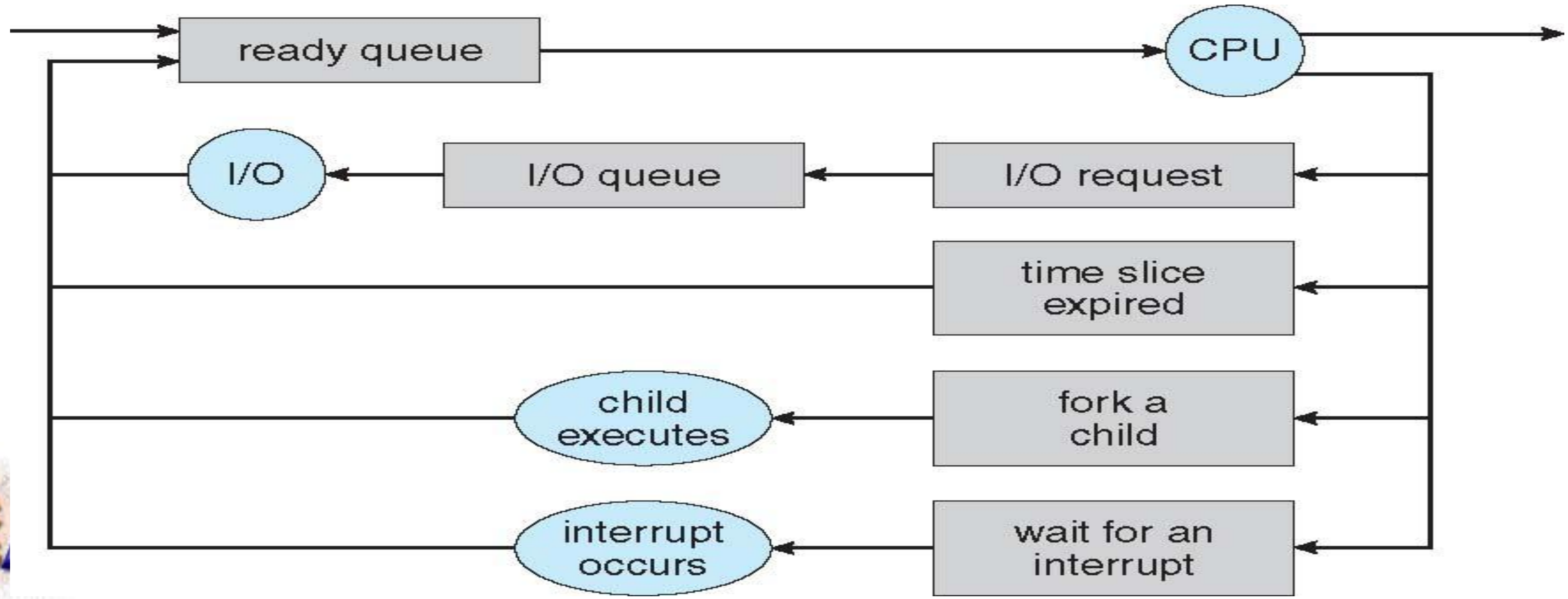
- Process state transition from one queue to another
- **Job queue** (before process gets into main memory)
  - All processes in mass storage (disk) waiting for allocation of memory (non-demand-paging system)
- **Ready queue**
  - In main memory waiting for the CPU
  - Usually a **linked list** is used to manage PCBs of all processes in the ready queue



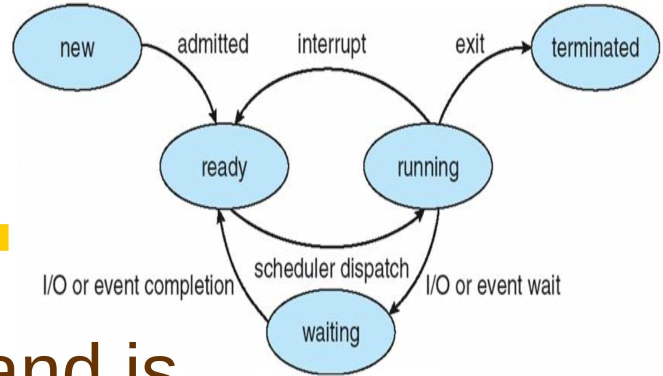
Process Queues (cont.)



- Device queues
  - One for each device containing all processes waiting for it
  - Disk drives, tape drives, and terminals
  - Shareable devices (disk drives): may have processes
  - Dedicated devices (tape drives): have at most one process



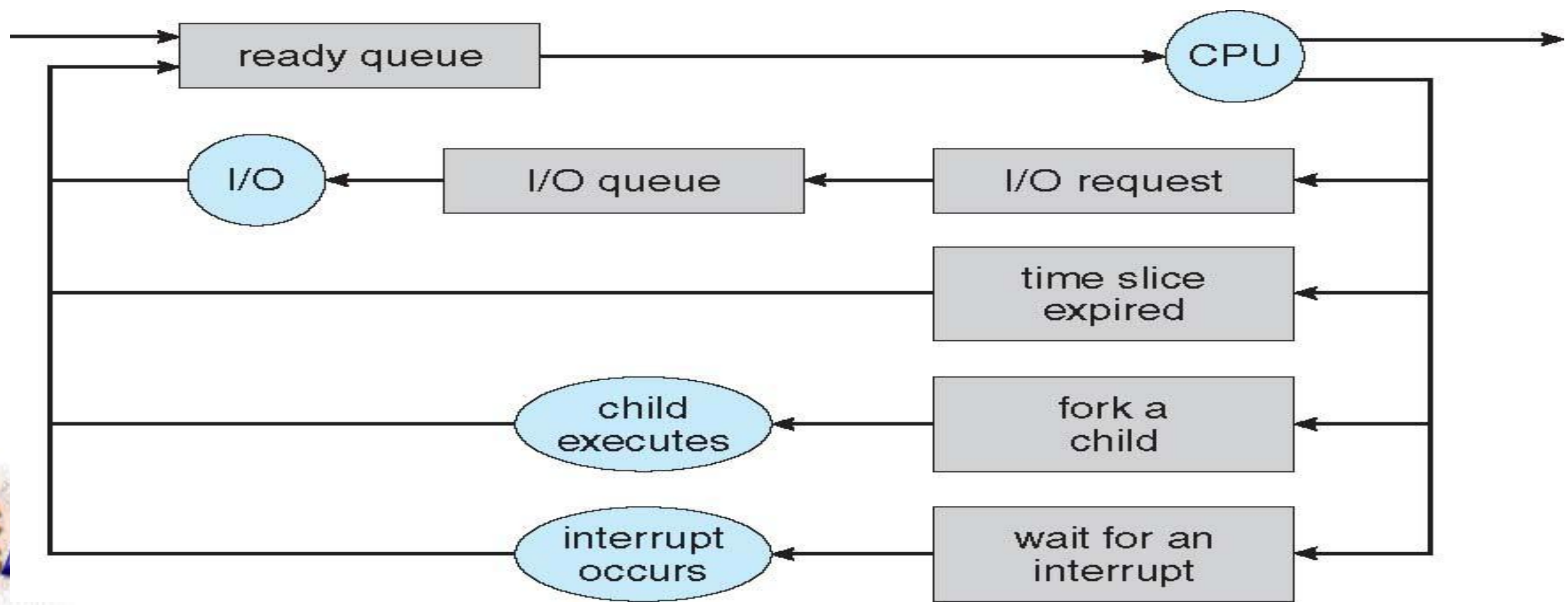
Process Queues (cont.)



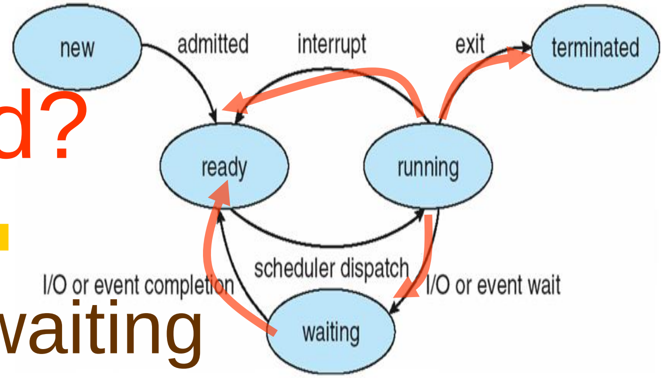
- Once the process is allocated the CPU and is executing, one of several events could occur:
  - The process could issue an I/O request and then be placed in an I/O queue.
  - The process could create a new child process and wait for the child's termination.
  - The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.
- A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.

Scheduler

- A process is migrated among various queues
- Operating system must select (schedule) processes from these queues in some fashion.
  - The selection process is called as "Scheduler".



When CPU Scheduler is Invoked?



- 1. Process switches from running to waiting
  - Requests for I/O operations
- 2. Process switches from waiting to ready
  - I/O completed and interrupt
- 3. Process switches from running to ready
  - Due to timer interrupt
- 4. Process terminates
  - When a process complete its work

Non-Preemptive vs. Preemptive

- Non-preemptive scheduling: **voluntarily give up CPU**
  - The process having the CPU uses it until finishing or needing I/O
  - Not suitable for time-sharing
  - Only IO (case 1) and process termination (case 4) can cause scheduler action
- Preemptive scheduling: **system forcibly gets CPU back**
  - Process may be taken off CPU **non-voluntarily**
  - Both occasions 2 and 3 may cause scheduler action
  - Time-sharing systems have to be **preemptive**!

Dispatcher and Context Switch

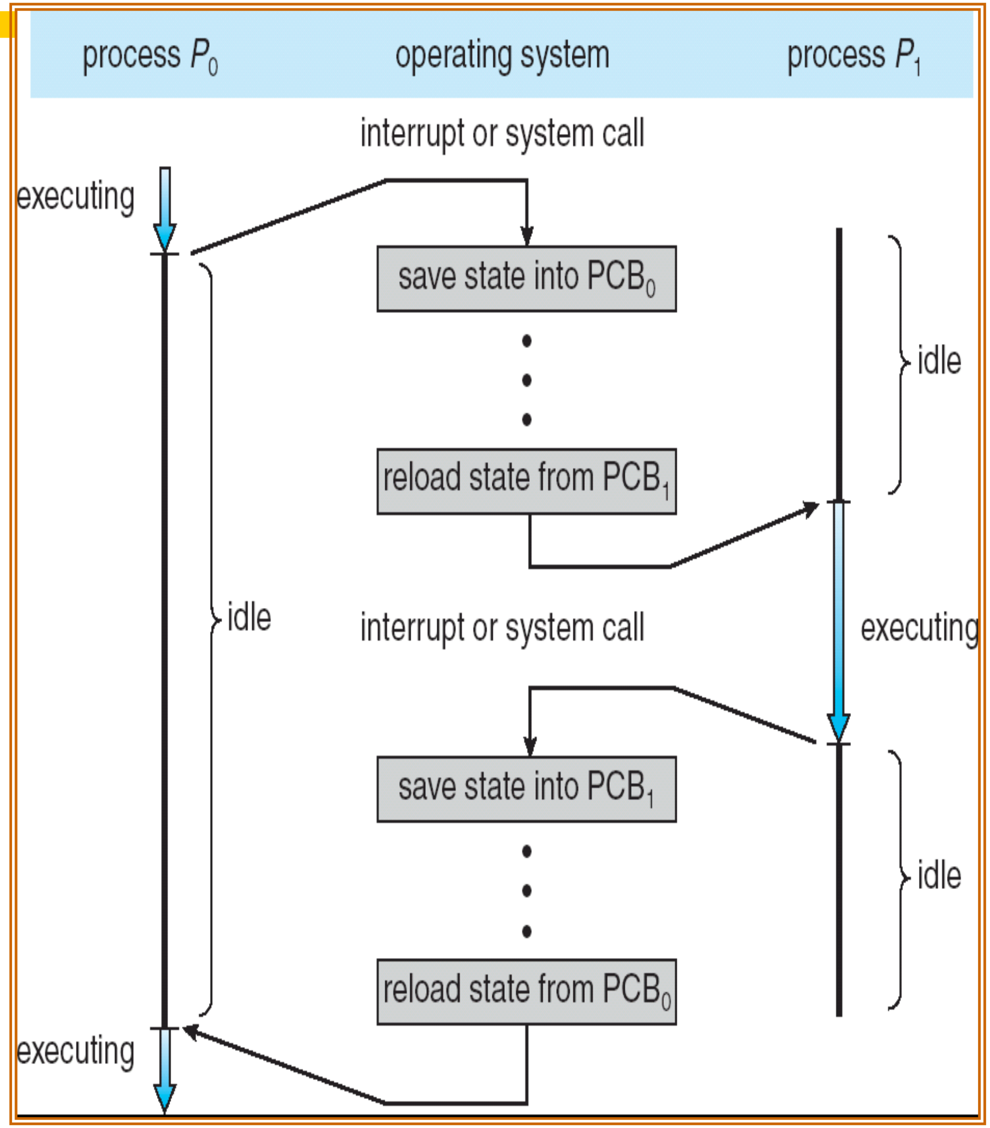
- **Dispatcher**: gives control of CPU to selected process
  - **Context switch**
  - Setting the program counter (PC)

- **Context Switch**: switch CPU to another process
  - Save **running state** of the old process: **where to save** ?
  - Load the **saved state** of the new process: **from where** ?
  - A few microsecond to 100's of microseconds depending on **hardware support**

PBC

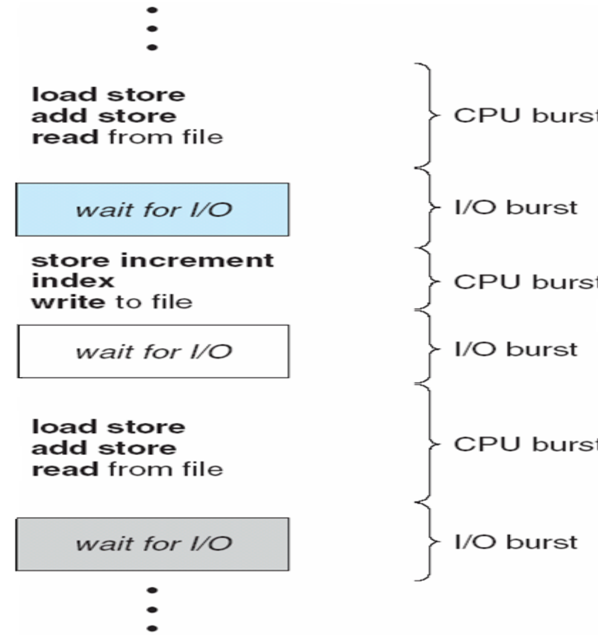
Context Switch

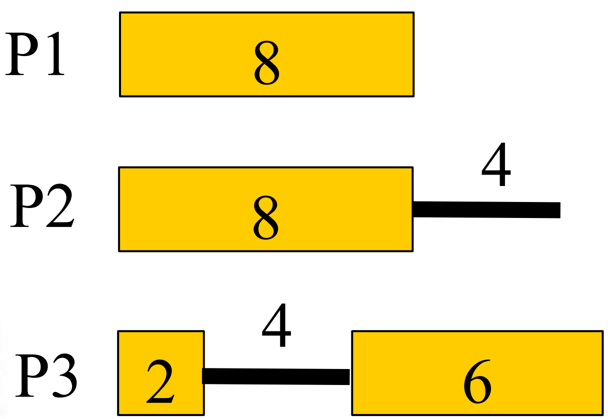
- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
- Hardware support
  - Multiple set of registers then just change pointers
- Other performance issues/problems
  - Cache content: locality is lost
  - TLB content: may need to flush



Representation of Process

- Model of Process
  - Cycle of (interleaving) CPU and I/O operations
- **CPU bursts**: time to use CPU
- **I/O bursts**: time to use I/O devices





(one CPU burst)

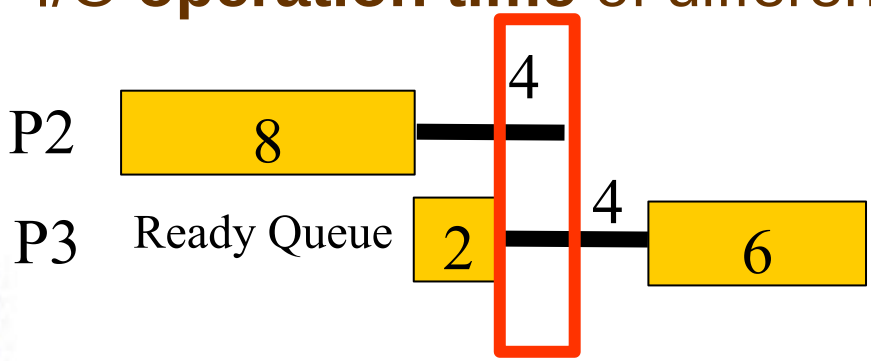
(CPU + I/O bursts)

(CPU, I/O, CPU)

Models/Assumptions for CPU Scheduling

- CPU model
  - By default, assume **only a single CPU core**
  - **Exclusive use of CPU**: only one process can use CPU

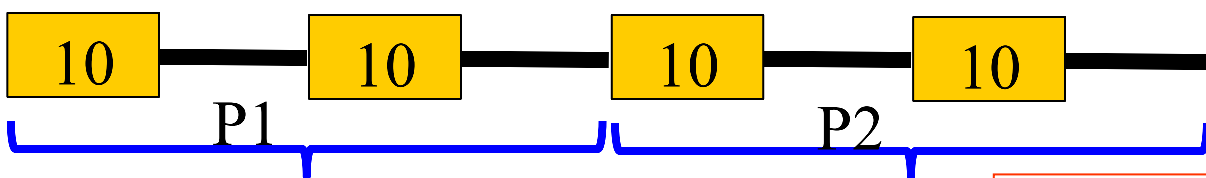
- I/O model
  - Multiple I/O devices (so **no** waiting in IO queues!)
  - Processes can access/request different I/O devices
  - I/O **operation time** of different processes can overlap



An Example: No Multiprogramming

- Suppose 2 processes, where each process
  - Requires 20 seconds of CPU time
  - Waits 10 second for I/O for every 10 seconds execution

- How will they run without multiprogramming?
  - Run one after another

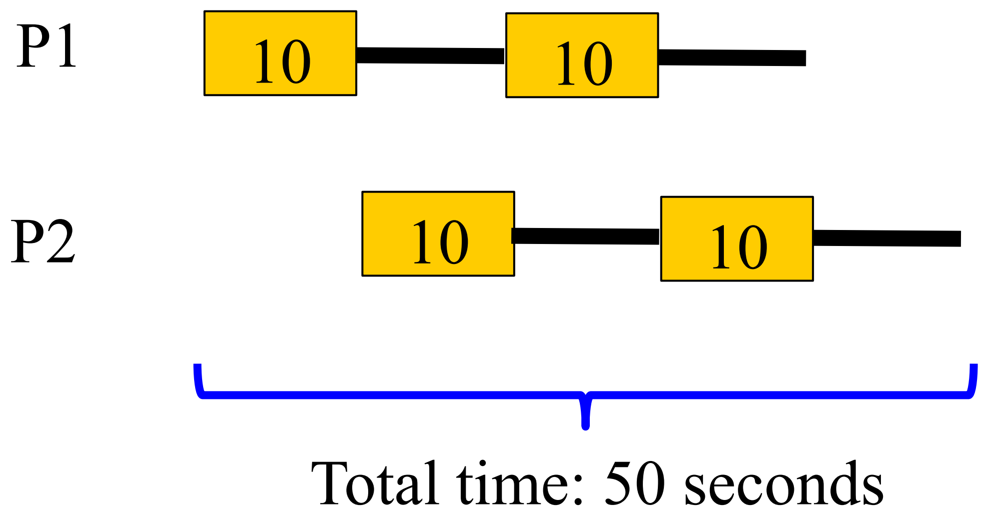


- How well do we utilize our CPU?
  - Out of 80 sec, CPU was busy for 40 sec
  - **CPU utilization is about 40/80*100 = 50%**

$$CPU\_util = \frac{CPU\_busy\_time}{Total\_time}$$

An Example: with Multiprogramming

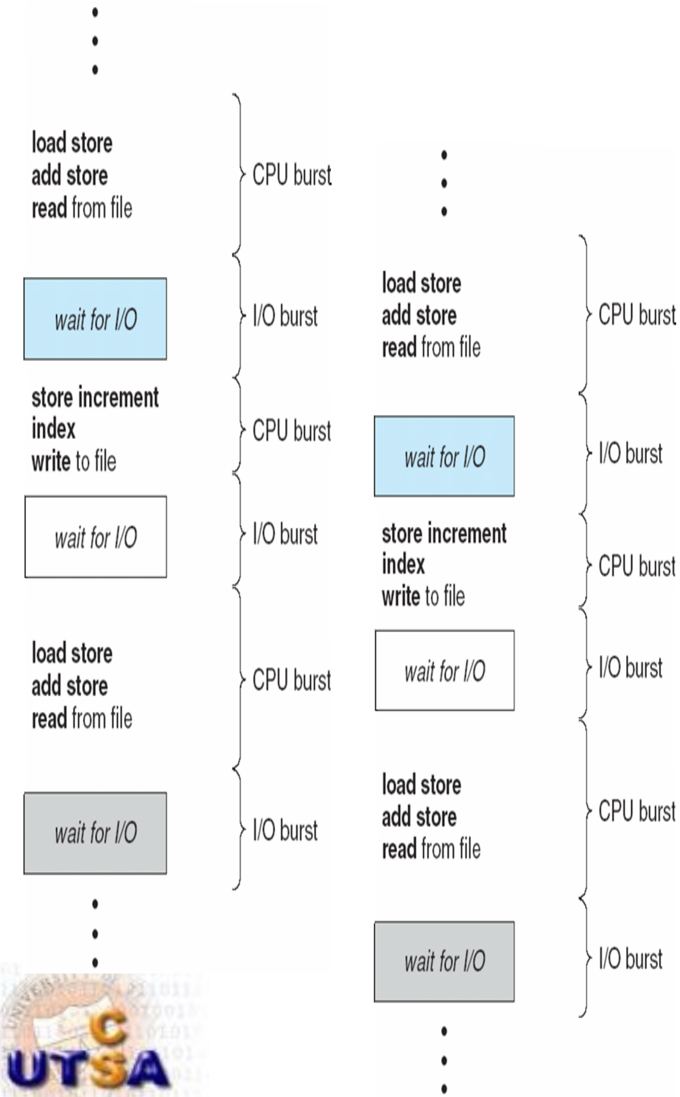
- Multiprogramming: both processes run **together**
  - The first process finishes in 40 seconds
  - The second process uses CPU (I/O) alternatively with first one and finishes 10 second later → **50 seconds**



CPU utilization = ?

40/50*100 = 80%

Multiprogramming



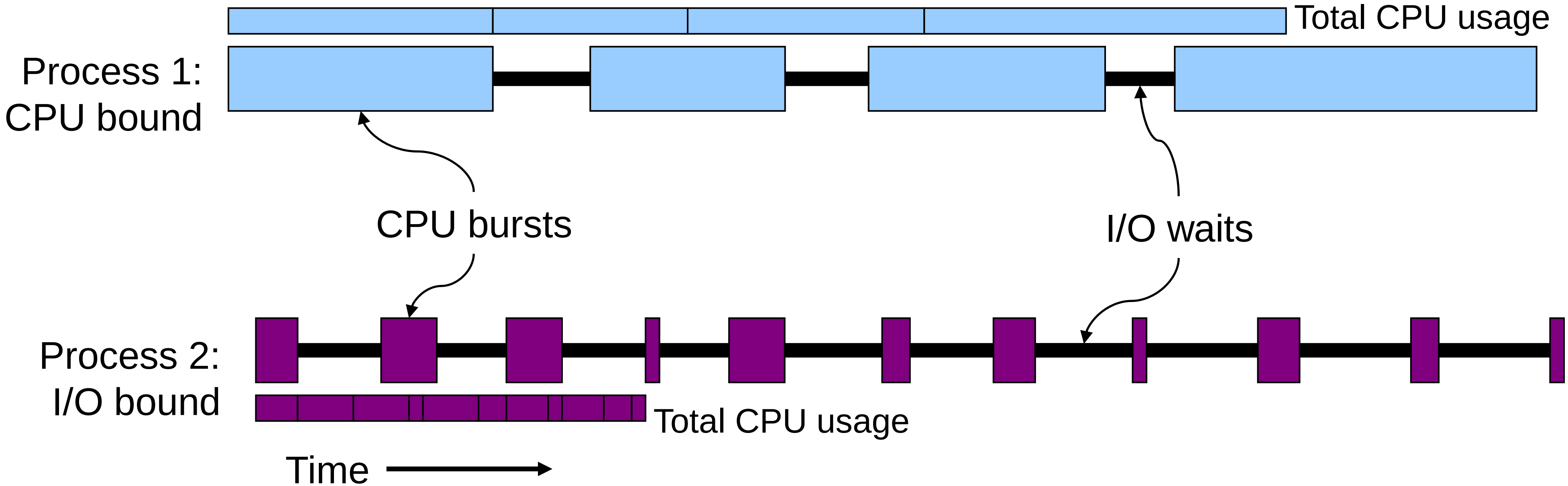
- Multiprogramming is a form of parallel processing in which several programs are run at the same time on a uniprocessor.

Why? Objective?

Maximize CPU utilization.
When a process waits for IO, all waiting time is wasted and no useful work is accomplished.

CPU-bound vs. IO-Bound

***CPU-bound*: spend more time on CPU, high CPU utilization**



***I/O-bound*: spend more time on IO, low CPU utilization**

Outline

- Reviews on process
- Process queues and scheduling events
- Different levels of schedulers
- Preemptive vs. non-preemptive
- Context switches and dispatcher
- Performance criteria
  - Fairness, efficiency (CPU util), waiting time in ready queue, response time, throughput, and turnaround time;
- Classical schedulers: FIFO, SFJ, PSFJ, and RR
- CPU Gantt chart vs. process Gantt charts

Performance Criteria

- Methods to measure performance of CPU schedulers
- We know/measure the followings about each process
  - Arrival time: t_a
  - First response time: t_r
  - Finish time: t_f
  - Total CPU burst time: t_{cpu}
  - Total I/O time: t_{io}
  - Total time process waits in Ready queue t_{wait_ready}
  - Total time process waits in IO queue t_{wait_io} (0, we have multiple I/O !!!)
- We can also measure
  - Total time to finish all processes t_{total}
  - Total time CPU was idle t_{idle}
  - Total time spend for context-switch $t_{dispatch}$

Performance Criteria

- Fairness
  - Each process gets a fair share of CPU in Multiprogramming
- Efficiency: **CPU Utilization**
  - Percentage of time CPU is busy;

$$util = \frac{\sum t_{cpu}}{t_{total}}$$

`util= (t_total - t_idle - t_dispatch) / t_total`

$$util = (t_{total} - t_{idle} - t_{dispatch}) / t_{total}$$

- Throughput
  - Number of processes completed per unit time, e.g., 10 jobs per second;

$$Throughput = \frac{\#of\_processes}{t_{total}}$$

- Turnaround Time
  - Time from submission to termination

$$t_{turn\_arround} = t_f - t_a$$

Performance Criteria (cont.)

- Waiting time in ready queue
  - Time for a process waiting for CPU in a ready queue

$$\boxed{t_{wait\_ready} = t_{turn\_arround} - t_{cpu} - t_{io} - (t_{wait\_io}=0)}$$

$$\boxed{t_{wait\_ready} \mathrel{+}= (CLK_{taken\_from\_queue} - CLK_{put\_into\_queue})}$$

- Response time
  - Time between submission and the first response

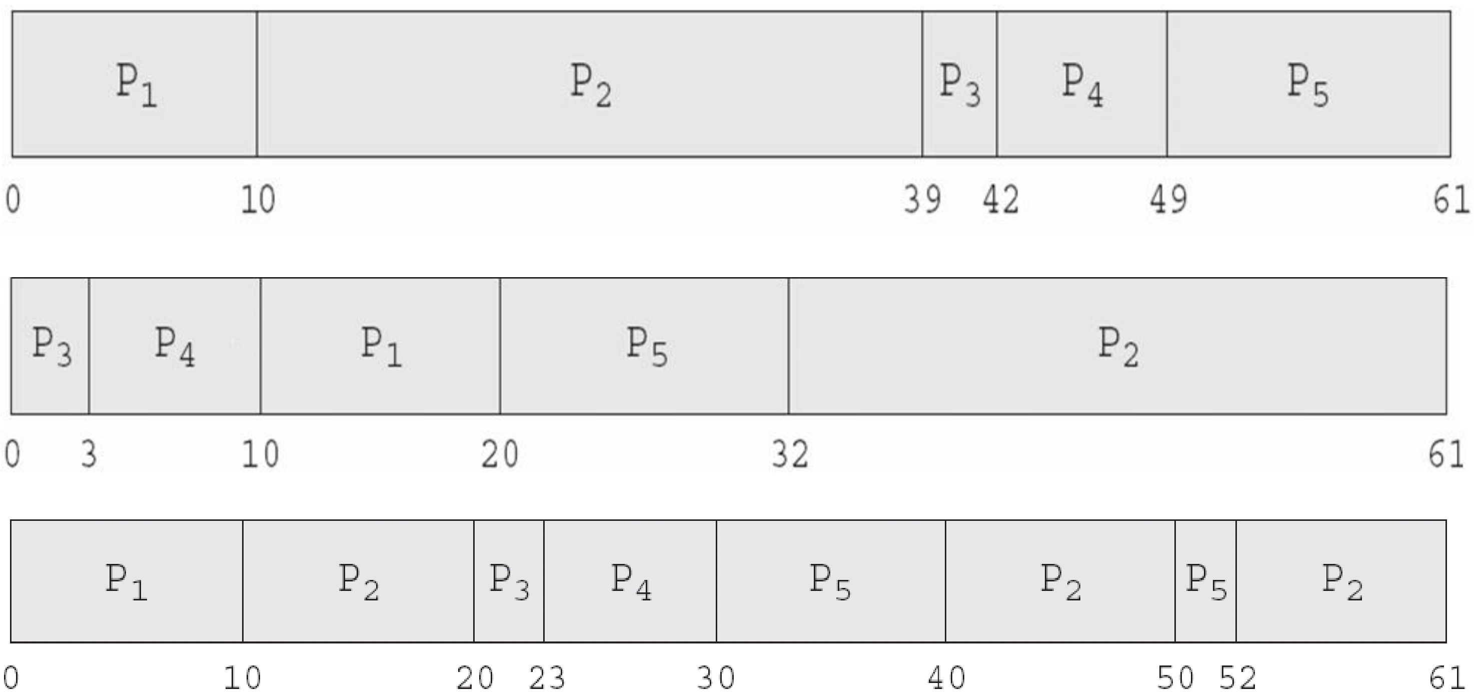
$$\boxed{t_{response} = t_r - t_a}$$

  - Good metric for interactive systems
  - Response time variance
    - ✓ For interactive systems, response time should **NOT** vary too much

Exercise: Compute performance metrics

Suppose all processes arrived at the same time!

Take average of turnaround, waiting, response times!



- **CPU utilization** : What percent of the time the CPU is used
- **Throughput** : Number of processes that complete their execution per time unit
- **Turnaround time :** Amount of time to execute a particular process
- **Waiting time:** Amount of time a process has been waiting in the **ready queue**
- **Response time :** Amount of time it takes from when a request was submitted until the first response is produced

Scheduling Goals

- All systems
  - **Fairness**: give each process a fair share of the CPU
  - **Balance**: keep all parts of the system busy; CPU vs. I/O
  - Enforcement: ensure that the stated policy is carried out
- Batch systems
  - **Throughput**: maximize jobs per unit time (hour)
  - Turnaround time: minimize time users wait for jobs
  - CPU utilization: CPU time is precious → keep the CPU as busy as possible
- Interactive systems
  - **Response/wait time**: respond quickly to users' requests
  - Proportionality: meet users' expectations
- Real-time systems: correctness and in-time processing
  - Meet **deadlines:** deadline miss → system failure!
  - Hard real-time vs. soft real-time: aviation control system vs. DVD player
  - Predictability: timing behaviors is predictable

Max CPU utilization
Max throughput
Min turnaround time
Min waiting time
Min response time

Usually NOT possible to optimize for *all* metrics with the same scheduling algorithm.
So, focus on different goals under different systems

Deciding which of the processes in the ready queue is to be selected.

FIFO	(First In First Out)	: non-preemptive, based on arrival time
SJF	(Shortest Job First)	: preemptive & non-preemptive
PR	(PRiority-based)	: preemptive & non-preemptive
RR	(Round-Robin)	: preemptive

SCHEDULING ALGORITHMS

Scheduling Policy vs. Mechanism

- Separate ***what*** should be done from ***how*** it is done
  - **Policy** sets **what** priorities are assigned to processes
  - Mechanism allows
    - Priorities to be assigned to processes
    - CPU to select processes with high priorities
- Scheduling algorithm parameterized
  - Mechanism in the kernel
  - Priorities assigned in the kernel or by users
- Parameters may be set by user processes
  - Don’t allow a user process to take over the system!
  - Allow a user process to voluntarily lower its own priority
  - Allow a user process to assign priority to its threads

Classical Scheduling Algorithms

- **FIFO** or **FCFS** *: non-preemptive, based on arrival time*
  - Long jobs delay everyone else
- **SJF** *: preemptive & non-preemptive*
  - Optimal in term of waiting time
- **PR** *: preemptive & non-preemptive*
  - Real-time systems: earliest deadline first (EDF)
- **RR** *: preemptive*
  - Processes take turns with fixed time quantum e.g., 10ms
- Multi-level queue (priority classes)
  - System processes > faculty processes > student processes
- Multi-level feedback queues: change queues
  - short → long quantum

First-Come First Served (FCFS)

- Managed by a strict FIFO queue
- **CPU Gantt chart**
  - **show which process uses CPU at any time**
- An Example of 3 processes arrive in order
  - P1: **24** (CPU burst time), P2: **3**, P3: **3**
  - CPU Gantt chart for the example

```
 ------------------------- --- ---
|           P1            |P2 |P3 |
 ------------------------- --- ---
0                        24   27  30
```

  - Average waiting time (in ready queue) =

    (0 + 24 + 27)/3 = 17

- **CPU utilization** : What percent of the time the CPU is used
- **Throughput** : Number of processes that complete their execution per time unit
- **Turnaround time :** Amount of time to execute a particular process
- **Waiting time:** Amount of time a process has been waiting in the ready queue
- **Response time :** Amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

First-Come First Served (FCFS): cont.

- What if the 3 processes arrive in a **different order**
  - P2:**3**, P3:**3**, and P1: **24** (CPU burst time)
  - CPU Gantt chart for the example

```
 --- --- -------------------------
|P2 |P3 |           P1            |
 --- --- -------------------------
0   3   6                        30
```

  - Avg waiting time (AWT) = (0 + 3+ 6)/3 = 3 !!!
  - **Big improvement of AWT over the previous case**!

Problem of FCFS: long jobs delay every job after them. Many processes may wait for a single long job.
Convoy effect: short process behind long process

Process Gantt Chart vs. CPU Gantt chart

- For each process, show its state at any time
- For the last example with the original order
  - CPU Gantt Chart

```
 -------------------------- --- ---
|             P1            |P2 |P3 |
 -------------------------- --- ---
0                          24  27  30
```

  - Process Gantt chart

```
P1: RRRRRRRRRRRRRRRRRRRRRRRR
P2: rrrrrrrrrrrrrrrrrrrrrrrrRRR
P3: rrrrrrrrrrrrrrrrrrrrrrrrrrrRRR
```

R (Running), r (ready), w (waiting)

Another Example: CPU and I/O Bursts

- Two processes

Process	CPU Burst	I/O Burst	CPU Burst
1	8	7	3
2	6	3	2

- Process Gantt chart for FCFS - FIFO

```
P1: RRRRRRRRwwwwwwwRRR
P2: rrrrrrrrRRRRRRwwwrRR
```

- AWT in ready queue = (0+9) / 2 = 4.5
- Notes:
  - **Waiting time in ready queue** for process is the number of **r**'s in the string
  - AWT is total number of r's divided by number of processes
  - **CPU utilization** is the total number of R's divided by the total time (the length of the longer string)

Shortest Job First (SJF)

- SJF: run the job with the shortest CPU burst first!
- Example of 4 processes
  - CPU Gantt chart

```
 ---  -------  --------  ---------
|P4 |   P1    |   P3     |    P2    |
 ---  -------  --------  ---------
0   3         9         16          24
```

Prcocess	Burst Time
1	6
2	8
3	7
4	3

  - AWT in ready queue = (0+3+9+16)/4 = 7
- Which job has the shortest CPU burst in practice?
  - Use past history to predict:

exponential average

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)(\alpha t_{n-1} + (1 - \alpha)\tau_{n-1})$$
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + (1 - \alpha)^2\tau_{n-1}$$
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + (1 - \alpha)^2\alpha t_{n-2} + (1 - \alpha)^3\tau_{n-2}$$
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \ldots + (1 - \alpha)^j\alpha t_{n-j} + \ldots (1 - \alpha)^n\alpha t_0$$

  - Suppose that
    τ is predicted time,
    and t is actual run time
  - τ _(n+1) = a*t_n + (1-a)* τ_n

Shortest Job First (SJF): cont.

- Example 3: SJF
  - Process Gantt chart

Process	CPU Burst	I/O Burst	CPU Burst
1	8	7	3
2	6	3	2

```
P1: rrrrrrRRRRRRRRwwwwwwwRRR
P2: RRRRRRwwwrrrrrRR
```

- Average waiting time:
  - P1: waits for **6 (r)** units first, then CPU, then I/O, and CPU
  - P2: runs for 6, does I/O, waits for **5 (r)** units, and CPU
  - AWT = (6 + 5)/2 = 5.5

SJF is optimal means that it gives minimum average waiting time for a given set of processes

Preemptive Shortest Job First (PSJF)

- So far, we considered **non-preemptive** SJF
- But it can be **Preemptive**, too! How?
  - If a new process enters the ready queue that has a shorter next CPU burst compared to **what is expected to be left (remaining)** of the currently executing process, then
  - Current running job will be replaced by the new one
- *Shortest-**remaining**-time-first* scheduling
- Example 3
  - Process Gantt chart

Process	CPU Burst	I/O Burst	CPU Burst
1	8	7	3
2	6	3	2

```
P1: rrrrrrRRRrrRRRRRwwwwwwwRRR
P2: RRRRRRwwwRR
```

- **AWT = (8+0)/2=4**

Round Robin Scheduling (RR)

- **Non-preemptive**: process keeps CPU until it terminates or requests I/O
- **Preemptive**: allows higher priority process preempt an executing process (if its priority is lower)
- Time sharing systems
  - Need to avoid CPU intensive processes that occupy the CPU too long
- Round Robin scheduler (RR)
  - **Quantum**: a small unit of time (10 to 100 milliseconds)
  - Processes take turns to run/execute for a quantum of time
  - Take turns are done in FIFO or FCFS

Round Robin Scheduling (RR): Examples

- RR with quantum of 4

Prcocess	Burst Time
1	24
2	3
3	3

```
P1:  RRRRrrrrrrRRRRRRRRRRRRRRRRRRRR
P2:  rrrrRRR
P3:  rrrrrrrRRR
```

  - AWT = (6+4+7)/3 = 17/3 = 5.67

- RR with quantum of 3

Process	CPU Burst	I/O Burst	CPU Burst
1	8	7	3
2	6	3	2

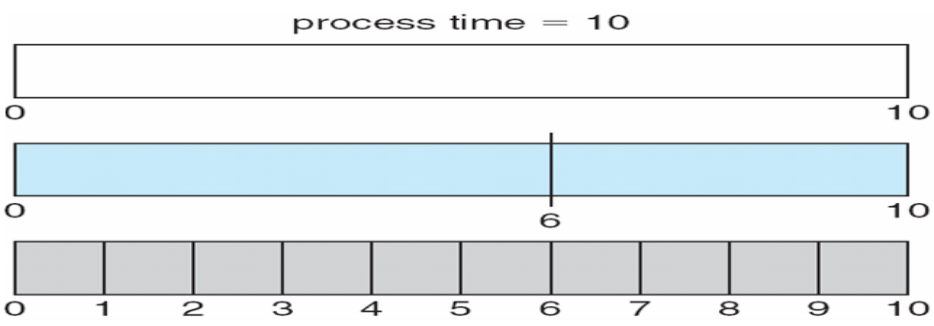
  - AWT = (6+6)/2 = 6.0

```
P1:  RRRrrrRRRrrrRRwwwwwwwRRR
P2:  rrrRRRrrrRRRwwwRR
```

Typically, higher average turnaround than SJF, but better *response*

Round Robin Scheduling (RR): Quantum

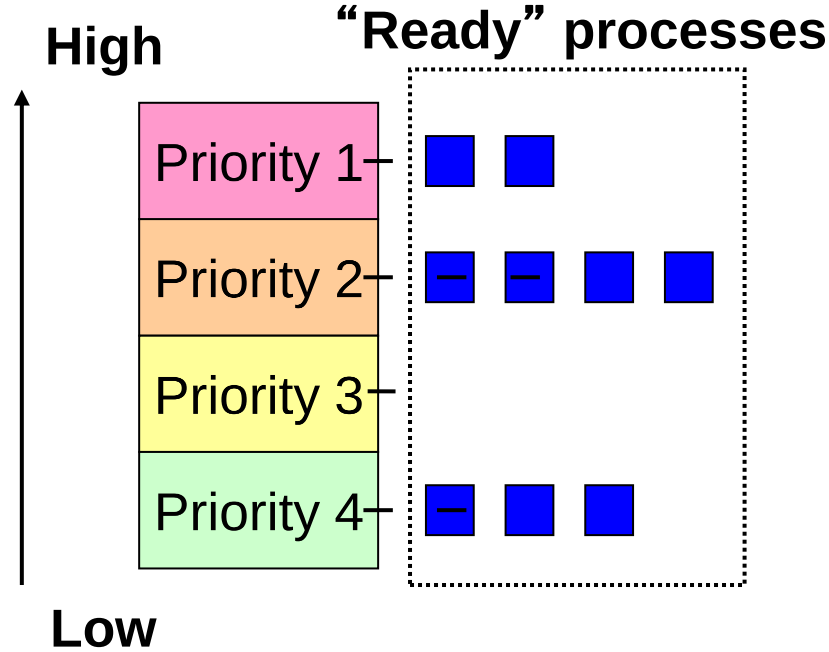
- If quantum is small enough
  - For n processes, each appears to have ***its own CPU*** that is ***1/n of CPU's original speed***
  - *What could be the poblem with too small quantum?*
- Quantum: determines **efficiency** & **response time**
- How to decide quantum size? → **10 to 100 ms**
  - Too small → too many context switches → no useful work
  - Too long → response time suffers
  - Rule of thumb: 80% of CPU bursts <= quantum



quantum	context switches
12	0
6	1
1	9

Priority Based Scheduling

- Assign a priority to each process
  - "Ready" process with highest priority allowed to run
  - Can be preemptive or non-preemptive
  - Same priority: use FIFO
- Priorities may be assigned dynamically
  - Reduced when a process uses CPU time
  - Increased when a process waits for I/O

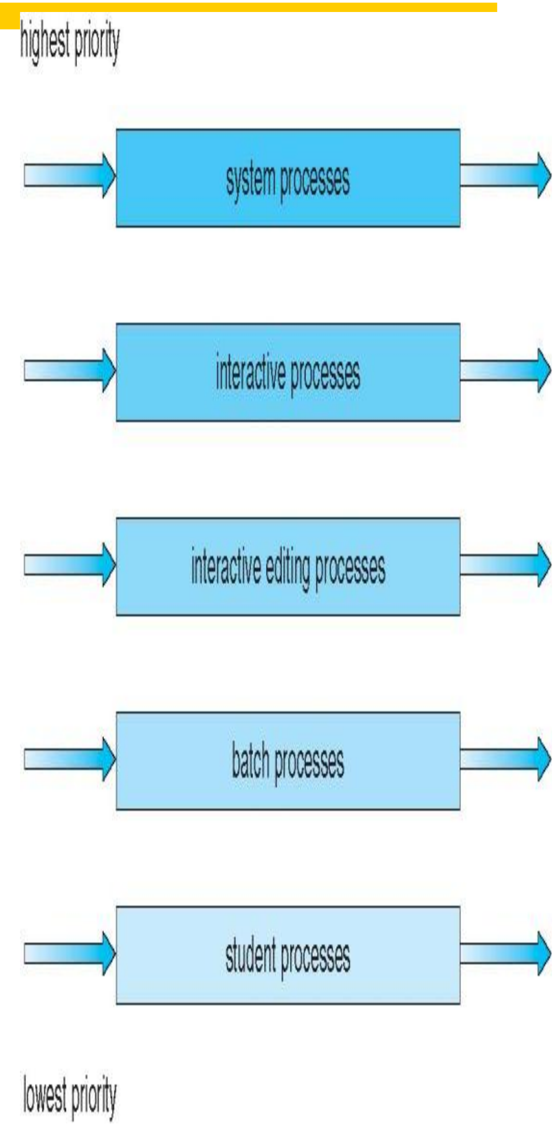


Summary of Schedulers

- FCFS: non-preemptive, based on arrival time
  - Long waiting time, e.g. long process before SSH console?
- SJF(shortest job first): preemptive & non-preemptive
  - Optimal in term of waiting time
- RR (Round-robin): preemptive
  - Processes take turns with fixed time quantum e.g., 10ms
- Priority-based scheduling
  - Real-time systems: earliest deadline first (EDF)
- Multi-level queue (priority classes)
  - System processes >faculty processes >student processes
- Multi-level feedback queues: short → long quantum

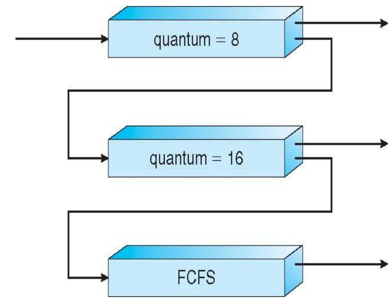
Multilevel Queues (Express lanes)

- Ready queue is partitioned into separate queues: **foreground** (interactive), **background** (batch)
- Each queue has its own scheduling algorithm
  - foreground – RR
  - background – FCFS
- How to do scheduling between the queues?
  - Fixed priority scheduling;
    - ✓ Serve all from foreground then from background
    - ✓ Possibility of starvation.
  - Time slice (Weighted Queuing (WQ))
    - ✓ Each queue gets a certain amount of CPU time which it can schedule among its processes;
      - 80% to foreground in RR
      - 20% to background in FCFS



Multilevel Feedback Queues

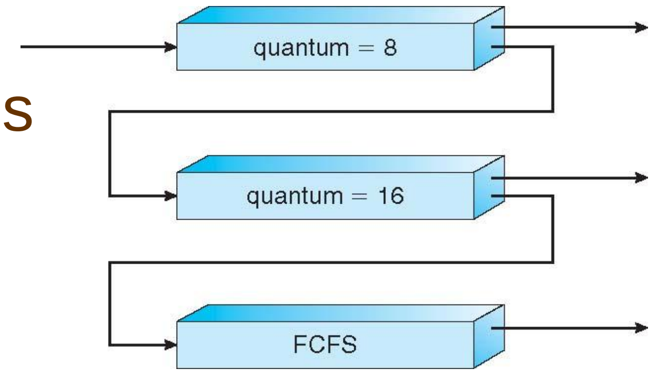
- A process can move between the various queues; aging can be implemented this way
  - CPU bound → move into low priority queue
  - I/O bound → move into high priority queue



- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service
- Most flexible and general, but hard to configure

Example of Multilevel Feedback Queue

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



- Scheduling
  - A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.
  - Processes requiring less than 8 ms will be served quickly…

Exercises: CPU Scheduling q4, q5, and this

a. **[12 points]** Consider two processes as in Assignment 2/Quiz 4, where each process has two CPU bursts with one I/O burst in between on a single core CPU. Suppose P1 and P2 have the following life-cycles:

P1 has x1=**8**, y1=7, z1=**3** units for the first CPU burst, I/O burst, second CPU burst, respectively.

P2 has x2=**6**, y2=**1**, z2=**2** units for the first CPU burst, I/O burst, second CPU burst, respectively.

Both arrives at the same time (in case of ties, pick P1) and there is no other processes in the system.

For each of the scheduling algorithms below, create Gantt charts as you did for the Quiz 4. Fill each box with the state of the corresponding process. Use **R** for **R**unning, **W** for **W**aiting, and **r** for **r**eady. Calculate the waiting times and CPU utilization (as a fraction) for each process and fill in the table below.

Gantt Charts for SJF (Shortest Job First, non-preemptive) [4pt]

a) SJF 5 10 15 20 25 30

P1																																			
P2																																			

Gantt Charts for PSJF (Preemptive SJF) [4pt]

b) PSJF 5 10 15 20 25 30

P1																																			
P2																																			

Waiting time and CPU utilizations [4pt]

Algorithm	Waiting times in ready queue			Finish time		Longest Schedule length	CPU utilization
	Process 1	Process 2	average	Process 1	Process 2		
b) SJF							
c) PSJF							

Exercise: multilevel queue

b. **[8 points]** Suppose we have a system using **multilevel queuing**. Specifically there are two queues and each queue has its own scheduling algorithm: QueueA uses RR with quantum 3 while QueueB uses FCFS. CPU simply gets processes form these two queues in a weighted round robin manner with 2:1 ratio (i.e. it gets **two** processes from QueueA then gets **one** process from QueueB, and then gets **two** processes from QueueA then gets **one** process from QueueB, and so on), But when it gets a process from QueueA, it applies RR scheduling with quantum 3. When it gets a process from QueueB, it applies FCFS scheduling.

Draw the Gantt charts (**5pt**) and compute waiting times (**3pt**) for the following four processes: P1, P2, P3, P4 on a single core CPU. Assume these processes arrived at the same time and in that order. Each process has a single CPU burst time of 5 units. There is no other processes or IO bursts.

	ratio						5					10					15					20	
QueueA	2	P1																					
RR q=3		P2																					
QueueB	1	P3																					
FCFS		P4																					

Compute Waiting times in ready queue				
P1	P2	P3	P4	average

Summary

- Reviews on process
- Process queues and scheduling events
- Different levels of schedulers
- Preemptive vs. non-preemptive
- Context switches and dispatcher
- Preference criteria
  - Fairness, efficiency, waiting time, response time, throughput, and turnaround time;
- Classical schedulers: FIFO, SFJ, PSFJ, and RR
- CPU Gantt chart vs. process Gantt charts
- Multilevel Queues