Operating system

Part IV: CPU Scheduling algorithms

By KONG LingBo (孔令波)

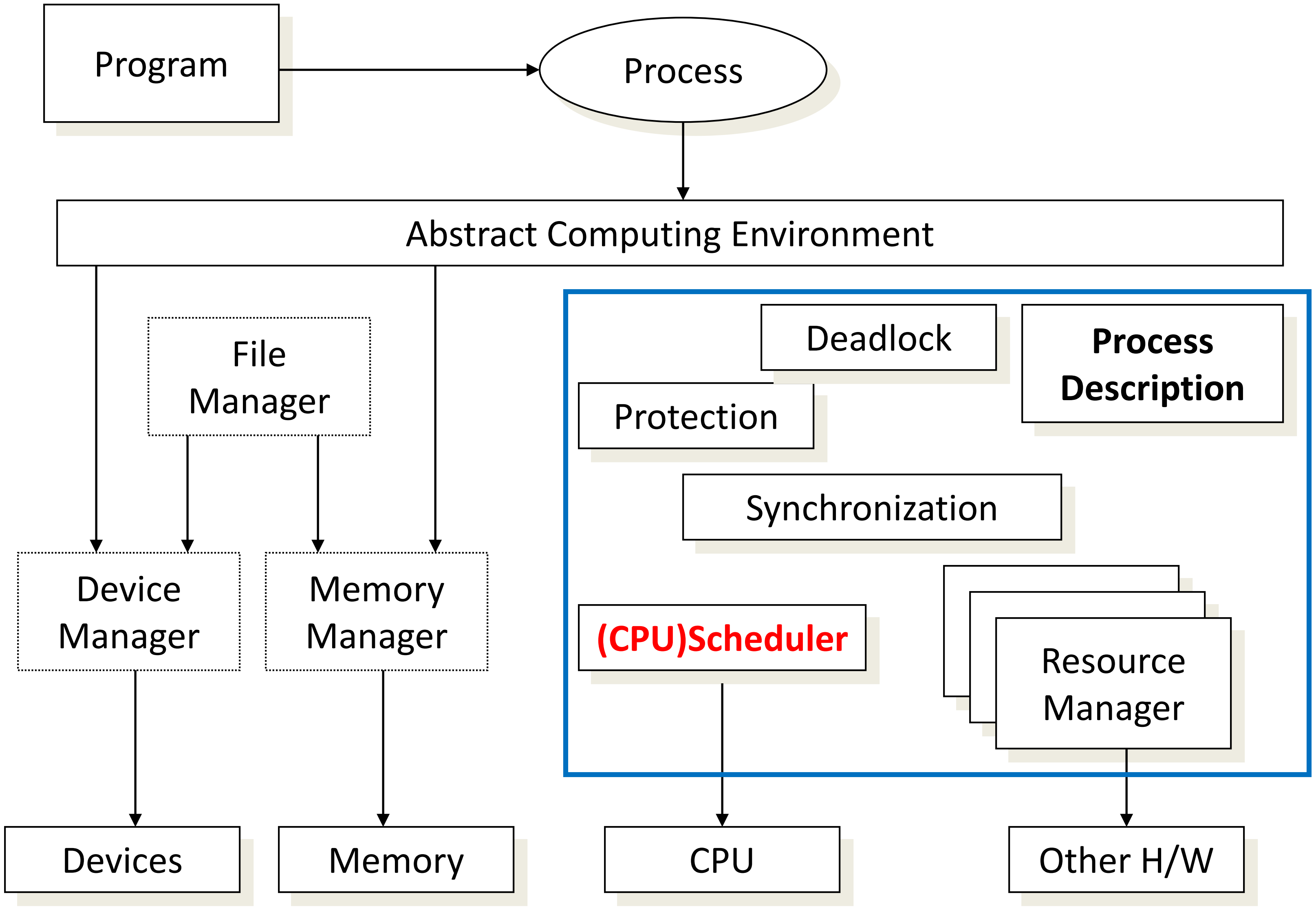
Outline of topics covered by this course

- **Introduction** – why should we learn OS?
- **Overview** – what problems are considered by modern OS in more details?
- **Execution** – CPU management
  - Process and Thread
  - CPU scheduling
- **Execution** – competition (synchronization problem)
  - Synchronization
  - Deadlock

Goals

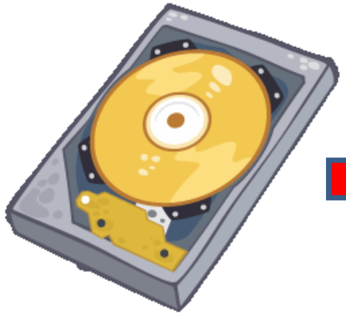
- To know the mechanism of CPU scheduling
  - Basic Concepts, Criteria
- CPU scheduling Algorithms
  - FCFS, SJF (Non-Preemptive), SRJF (Preemptive), Priority, Round Robin, Multilevel Queue, Multilevel Feedback Queue, Lottery etc.
- Algorithms Evaluation
  - How to simulate and evaluate those CPU scheduling algorithms

Overview of OS

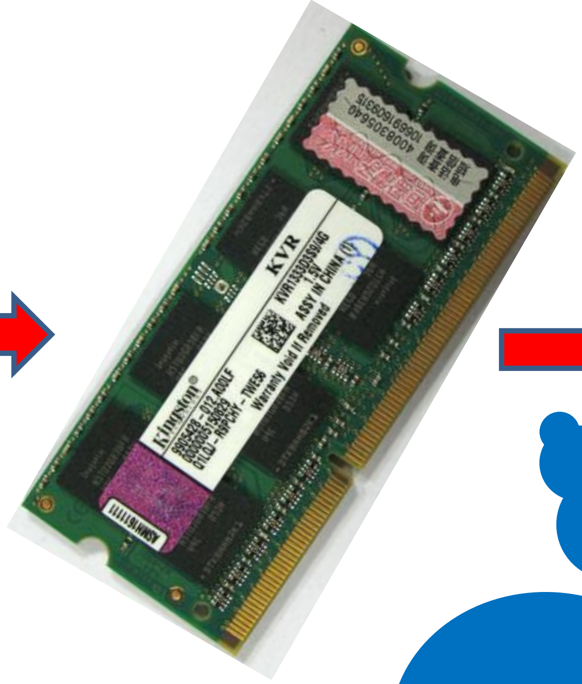


Three kinds of schedulers

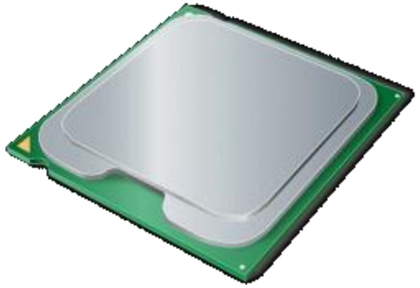
Storage media
(Magnetic disk)



Main Memory



CPU



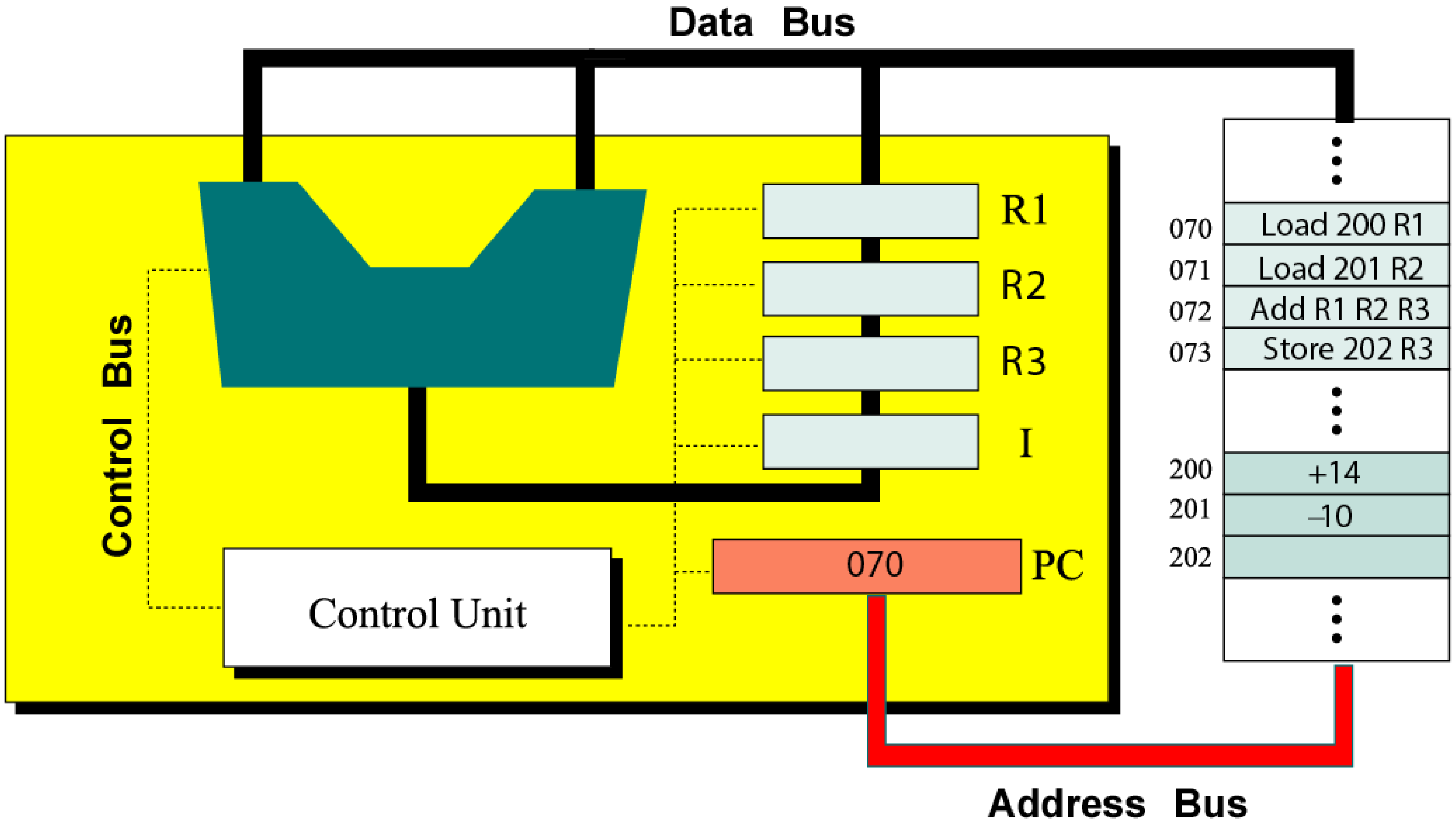
Short-term scheduler determines which processes could get the usage of CPU

CPU Scheduling

- Basic Concepts
  - CPU and some IO devices could work in parallel
  - Process could be categorized into IO-burst and CPU burst types
- Scheduling Criteria & Metrics
- Different Scheduling Algorithms
- Algorithm Evaluation

We have learned

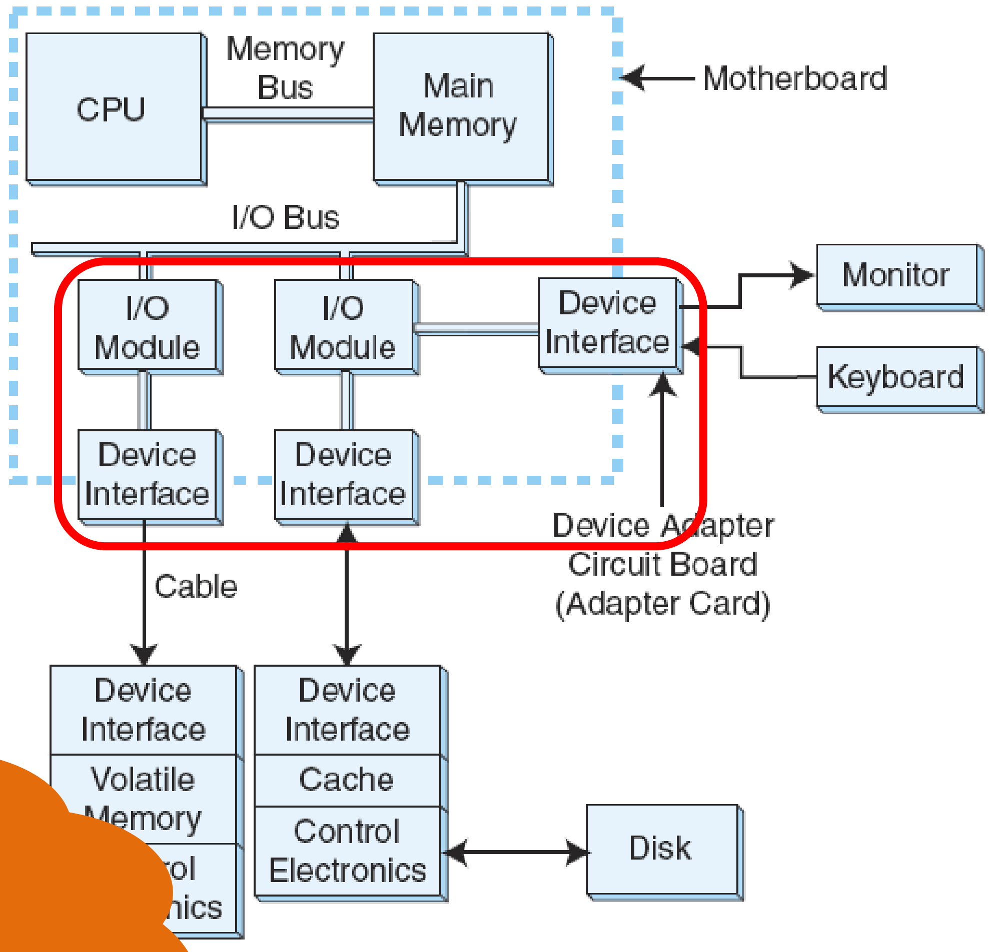
- CPU is the most important resource
  – Its job is to execute the instructions following Machine Cycle



Now

- CPU and IO can work in parallel
  - CPU now focuses on the **computation**
  - IO chipsets take over the IO tas

We'll learn the <u>**framework**</u> to manage IO devices in later IO chapter



.1 **A Model I/O Configuration**

[2003.Essentials Of Computer Organization And Architecture+++++.pdf]

Now

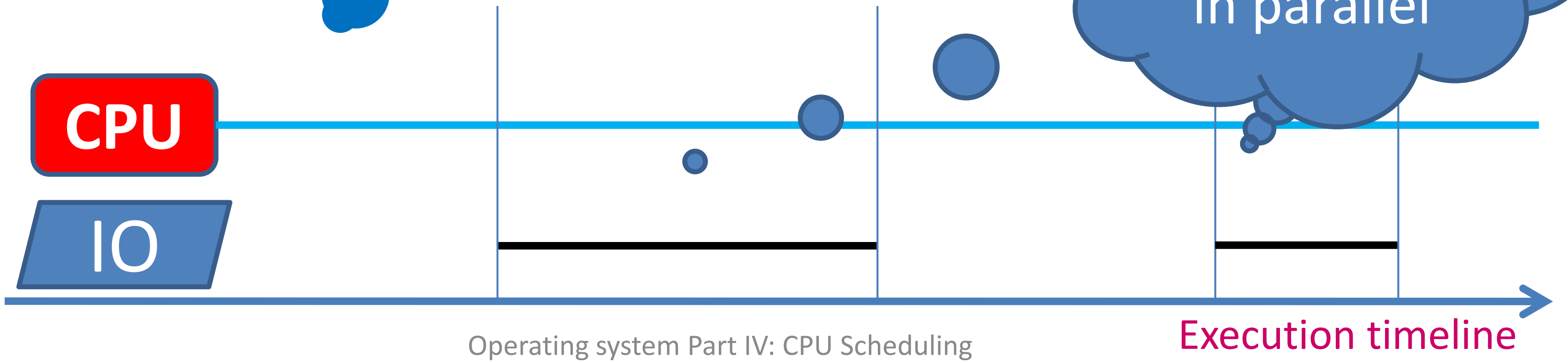
- Traditionally
  - CPU has to manage ... O operat...

CPU is waiting

This can make the execution of processes/Threads **efficiently**!

- Now
  - CPU and IO wor... parallel with the supp...

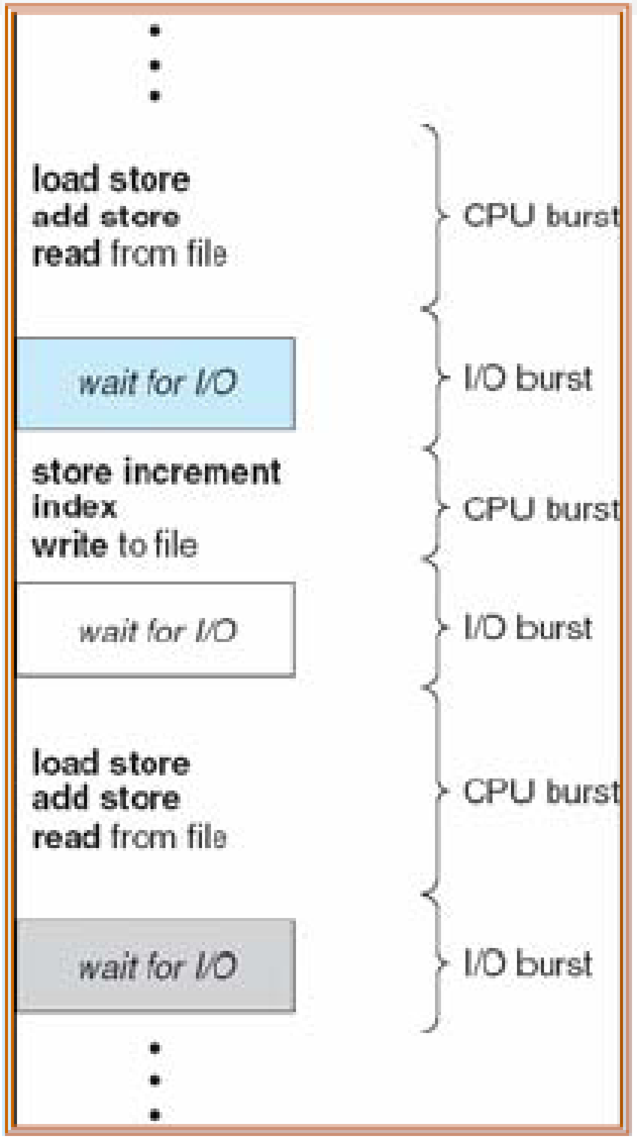
CPU and IO ideally work in parallel



Process's property: CPU-I/O Burst [突发, 爆发] Cycle

- Process execution consists of a cycle of **CPU execution** and **I/O request**

- Process execution begins with a **CPU burst** and followed by an **I/O burst**



PPTs from others\www.cs.gsu.edu_~cscbecx\csc4320 Chapter 5-1.ppt

CPU-I/O Burst

- Burst/Service time = total processor time needed in one CPU-I/O burst cycle.
- Jobs/Process with long CPU burst time are **CPU-bound jobs/processes** and are also referred to as "long jobs/processes".
- Jobs with short CPU burst time are **IO-bound jobs/processes** and are also referred to as "short jobs/processes".
- CPU-bound processes have longer CPU bursts than I/O-bound processes.

PPTs from others\From Ariel J. Frank\OS381\os9-1_cpu.ppt

CPU Scheduling

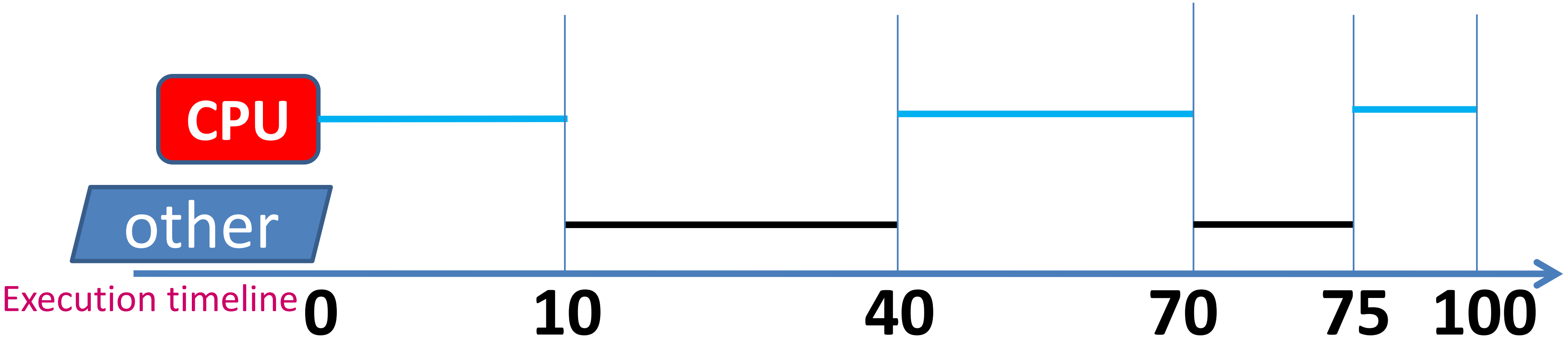
- Basic Concepts
- Scheduling Criteria & Metrics
- Different Scheduling Algorithms
- Algorithm Evaluation

Parameters to evaluate the scheduling

- CPU utilization [**CPU使用率**] (**Efficiency**)
  - keep the CPU as busy as possible (from 0% to 100%)
- **Fairness**: each process gets a "fair share" of the CPU
- Throughput [吞吐量]
  - # of processes that complete their execution per time unit
- Turnaround time [周转时间]
  - amount of time to execute a particular Process
    - i.e. execution time + waiting time
- Waiting time [等待时间]
  - amount of time a process has been waiting in the ready queue
- Response time [响应时间]
  - amount of time it takes from when a request was submitted until the first response is produced , not output (for time-sharing environment)

PPTs from others\www.cs.gsu.edu_~cscbecx\csc4320 Chapter 5-1.ppt

CPU utilization

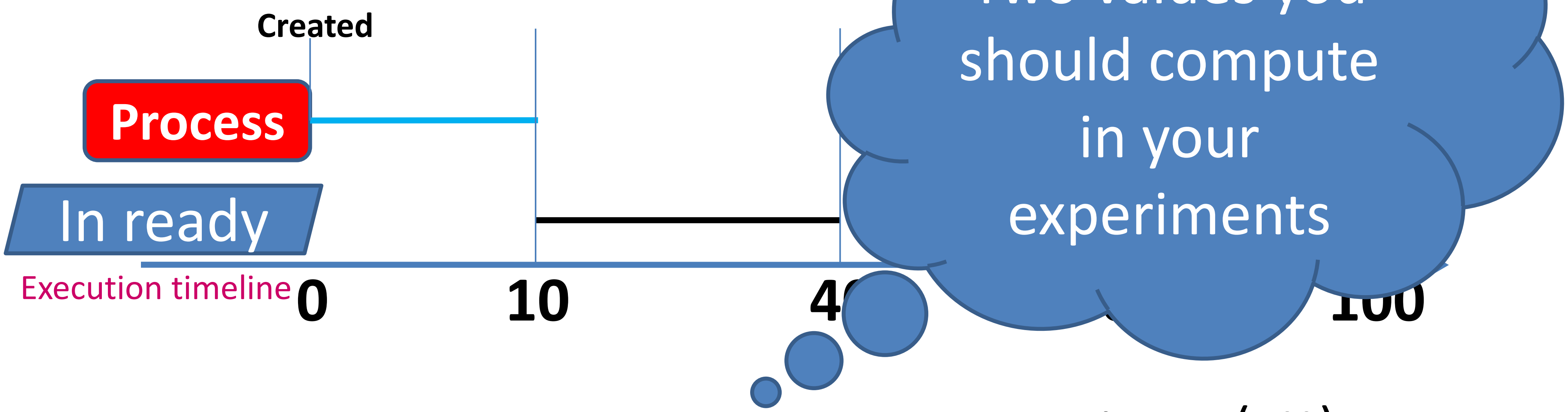


- CPU Utilization

  = (10 + 30 + 25)/100

  = 65%

Turnaround time & Waiting tim



- Turnaround time (tt_i)

  = 100

- Waiting time (wt_i)

  = 30 + 5

  = 35

- Average Turnaround time (att)

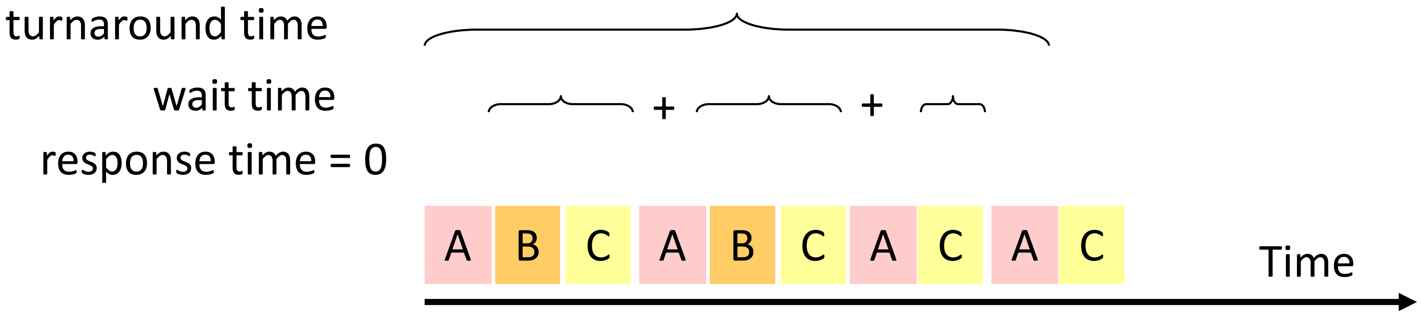
$$att = \frac{\sum tt_i}{n}$$

- Average Waiting time (awt)

$$awt = \frac{\sum wt_i}{n}$$

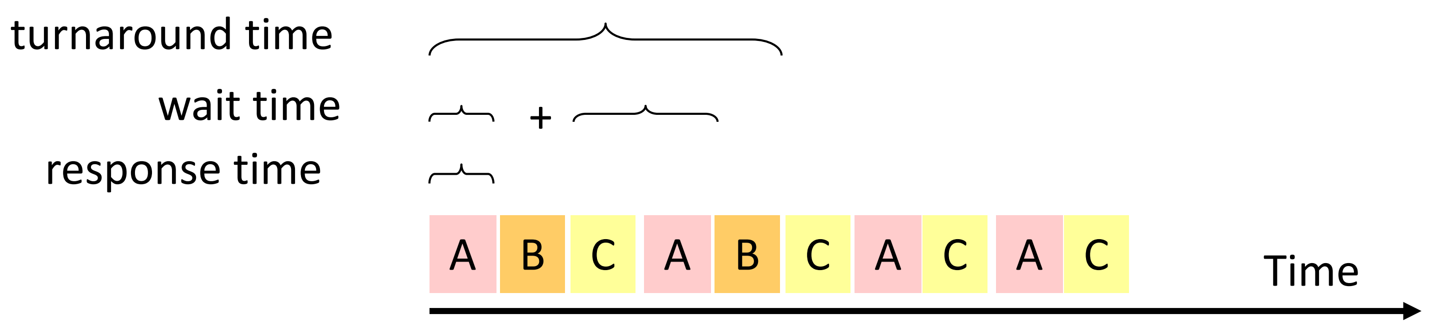
Goals for a Scheduler

- Suppose we have processes A, B, and C, submitted at time 0
- We want to know the response time, waiting time, and turnaround time of process **A**



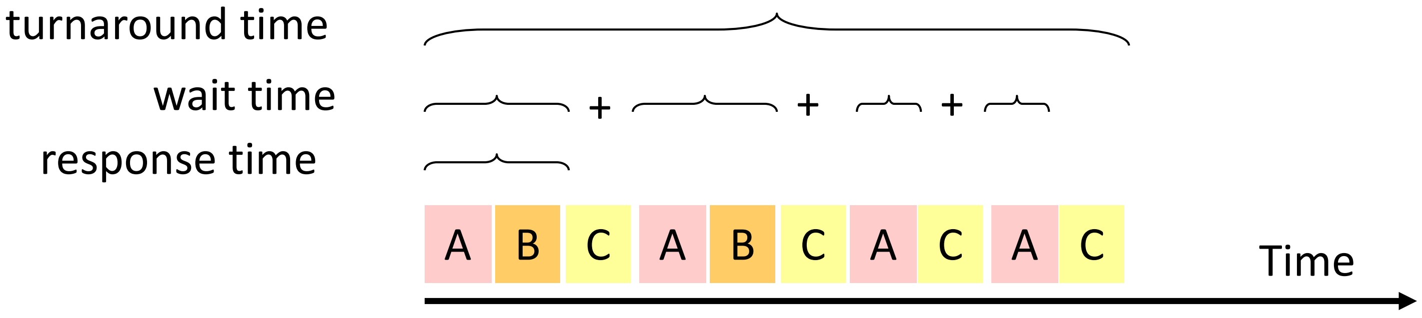
Goals for a Scheduler

- Suppose we have processes A, B, and C, submitted at time 0
- We want to know the response time, waiting time, and turnaround time of process **B**



Goals for a Scheduler

- Suppose we have processes A, B, and C, submitted at time 0
- We want to know the response time, waiting time, and turnaround time of process **C**



You can derive the Optimization Criteria

- To maximize or minimize some average measures:
  - Max **CPU utilization**
    - Fraction of the time the CPU isn't idle
  - Max **throughput**
    - Amount of "useful work" done per time unit
  - Min **turnaround time**
    - Time from process creation to process completion
  - Min **waiting time**
    - Amount of time a process spends in the WAITING state
  - Min **Response time**

PPTs from others\OS PPT in English\ch06.ppt

PPTs from others\www.cs.gsu.edu_~cscbecx\csc4320 Chapter 5-1.ppt

Preemptive vs. non-preemptive scheduling

—抢占式 v.s 非抢占式 调度

- **Non-preemptive** (not forcible removable):
  - Once a process is in the running state, it will continue until it terminates or blocks itself for I/O.
- **Preemptive** (forcible removable) :
  - Currently running process may be interrupted and moved to the Ready state by the OS.
    - processes can be suspended by scheduler
  - incurs a cost associated with access to share data.

Scheduling Algorithms

- **First Come First Serve Scheduling** [先来先服务] (Non-preemptive)
- **Shortest Job First Scheduling** [最短任务先服务]
  - SRTF (Shortest Remaining Time First Scheduling)/SRJF
- **Priority Scheduling** [优先权]
- **Round-Robin Scheduling** [时间片轮转]
- Multilevel Queue Scheduling [多层次队列]
  - Multilevel Feedback-Queue Scheduling [多层次反馈队列]
- Lottery Scheduling [抽彩]

CPU Scheduling

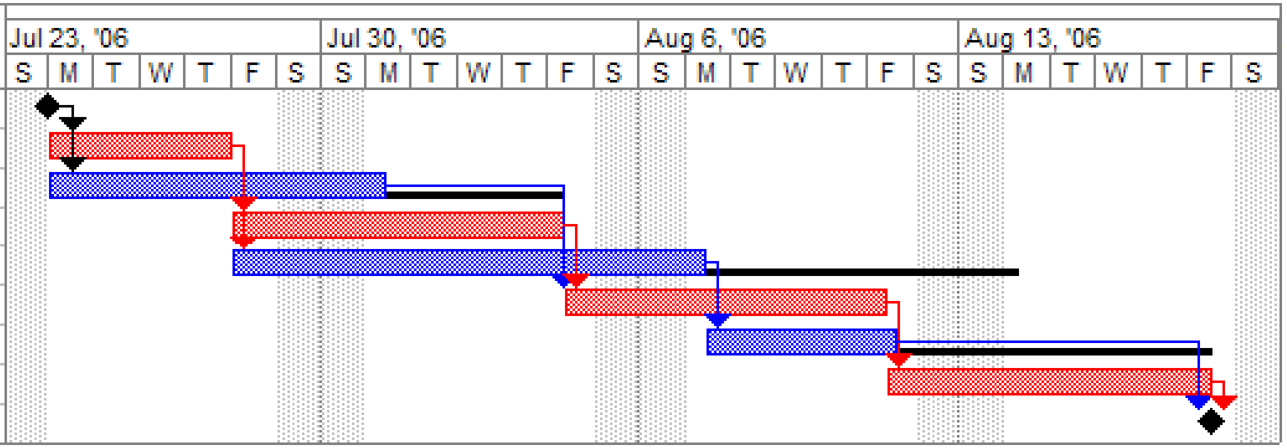
- Basic Concepts
- Scheduling Criteria & Metrics
- Different Scheduling Algorithms
- Algorithm Evaluation

Gantt Chart [甘特表]

http://en.wikipedia.org/wiki/Gantt_chart

- A Gantt chart is a type of bar chart, developed by Henry Gantt, that illustrates a project schedule.
  - Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project.
  - Terminal elements and summary elements comprise the work breakdown structure of the project

ID	Task Name	Predecessors	Duration
1	Start		0 days
2	a	1	4 days
3	b	1	5.33 days
4	c	2	5.17 days
5	d	2	6.33 days
6	e	3,4	5.17 days
7	f	5	4.5 days
8	g	6	5.17 days
9	Finish	7,8	0 days



FCFS (First Come First Serve) Scheduling

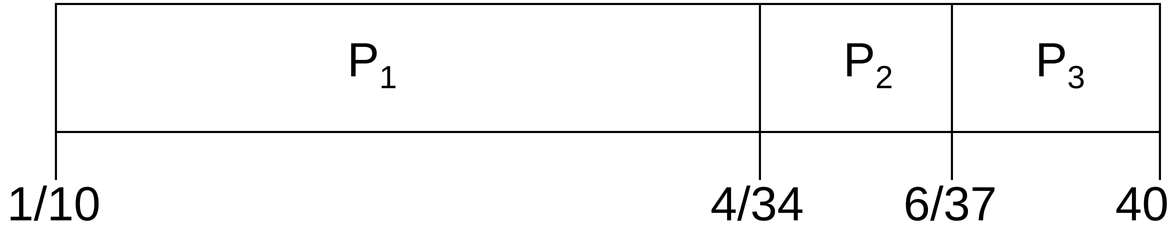
- FIFO scheduling
  - **Simplest** scheme
  - Processes dispatched according to arrival time
  - Non-preemptible/preemptive
  - Rarely used as primary scheduling algorithm

- Or First-In-First-Out (FIFO)

A Simple FCFS Example

Process	Burst/CPU Time	Arrival time
P_1	24	1
P_2	3	4
P_3	3	6

- **Suppose the scheduling time now is 10**
  The Gantt Chart for the schedule is:



- Waiting time for P_1 = 9; P_2 = 30; P_3 = 31
- Average waiting time: (9 + 30 + 31)/3 = 70/3
- **Convoy effect [护航效果]**: short process behind long process; or short process has to wait the long process to finish.

FCFS Drawbacks

- A process that does not perform any I/O will monopolize the processor (**Convoy Effect**).
- Favors CPU-bound processes:
  - I/O-bound processes have to wait until CPU-bound process completes.
  - They may have to wait even when their I/O are completed (poor device utilization).
  - We could have kept the I/O devices busy by giving a bit more priority to I/O bound processes.

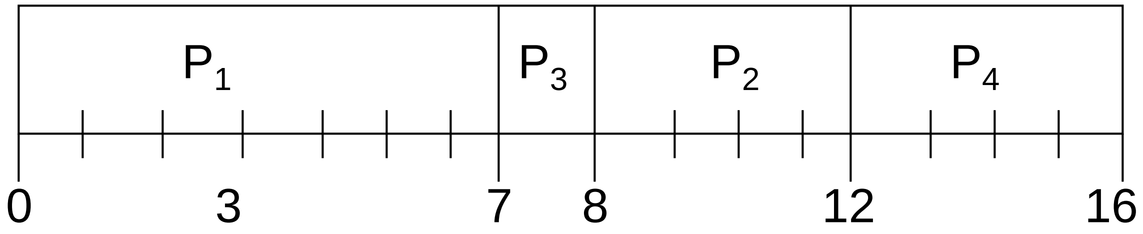
Shortest Job First (SJF) [Optimal]

- Selection function: the process with the shortest expected CPU burst time.
- Decision mode: Non-preemptive.
  - There is a variant which supports "Preemptive", called **SRJF** (Shortest-Remaining Job First)
- also called Shortest Time First (**STF**) and Shortest Process Next (**SPN**).
- I/O bound processes will be picked first.
- SJF is **optimal** – gives minimum average waiting time for a given set of processes.

Example of SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Dynamics of Shortest-Job-First (SJF)

- Associate with each process the length of its next CPU burst.
  - Use these lengths to schedule the process with the shortest time.
- **Non-preemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst.
- We need to somehow estimate the required processing time (CPU burst time) for each process.

Determining length of next CPU Burst

- Can only estimate the length.
- Can be done by using the length of previous CPU bursts, using exponential averaging:

$$1.\ t_n = \text{actual length of } n^{th} \text{CPU burst}$$

$$2.\ \tau_{n+1} = \text{predicted value for the next CPU burst}$$

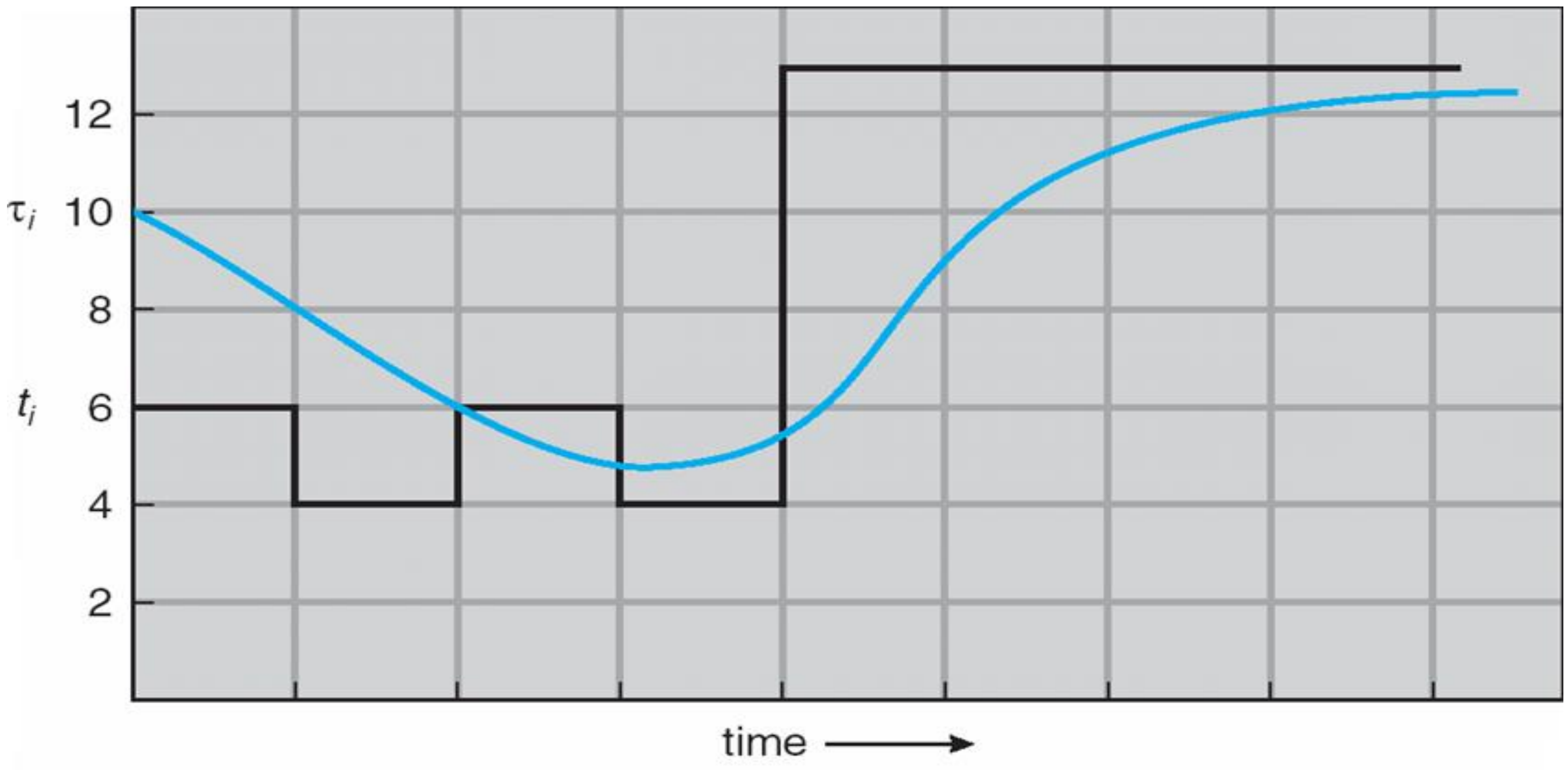
$$3.\ \alpha,\ 0 \le \alpha \le 1$$

$$4.\ \text{Define:} \quad \tau_{n+1} = \alpha\, t_n + \left(1 - \alpha\right)\tau_n .$$

Examples of Exponential Averaging

- How to set α in $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$?
- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$,
  - Recent history does not count.
- $\alpha = 1$
  - $\tau_{n+1} = t_n$,
  - Only the actual last CPU burst counts.
- Let's be balanced: $\alpha = 0.5$
  - See example in next slide.

Prediction of the length of the next CPU Burst



CPU burst (t_i)		6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

Shortest Job First Drawbacks

- Possibility of **starvation** for longer processes as long as there is a steady supply of shorter processes.
- Lack of preemption is not suited in a time sharing environment:
  - CPU bound process gets lower priority (as it should) but a process doing no I/O could still monopolize the CPU if he is the first one to enter the system.
- SJF implicitly incorporates priorities: shortest jobs are given preferences.

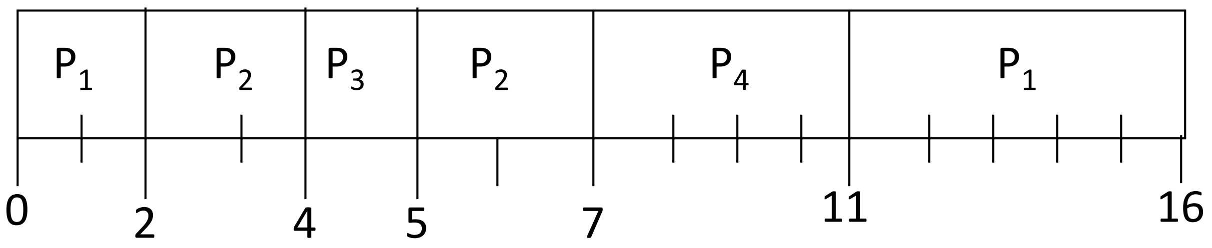
Shortest-Remaining-Job-First (SRJF)

- Idea
  - Associate with each process the length of its next/remaining CPU burst.
  - Use these lengths to schedule the process with the shortest time.
- **Preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt.
- Called Shortest-Remaining-Job-First (SRJF).

Example of Preemptive SJF → SRTF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

PPTs from others\Operating System Concepts 6ed\Slides\ch6.ppt

Shortest Remaining Time Scheduling

- **Preemptive version of SJF/SPN**
- Arriving processes with shorter burst preempt a running process
- Very large variance of response times
  - long processes wait even longer than under SPF/SJF/SPN
- Not always optimal
  - Short incoming process can preempt a running process that is near completion
  - Context-switching overhead can become significant

PPTs from others\www.cs.nmsu.edu_~jinghe_CS474\lect12.ppt

Priority

- A priority number (integer) is assigned for each process
- The CPU is allocated to the process with the highest priority (generally, **smallest integer ≡ highest priority**).
  - Preemptive
  - Non-preemptive

➔ SJF is a priority scheduling where priority is the predicted next CPU burst time.

- Problem：
  - **Starvation** – low priority processes may never execute.
- Solution：
  - **Aging** – as time progresses increase the priority of the process.

PPTs from others\Operating System Concepts 6ed\Slides\ch6.ppt

PPTs from others\www.cs.gsu.edu_~cscbecx\csc4320 Chapter 5-1.ppt

Example: Priority scheduling (non-preemptive)

Process	Arrive Time	Burst Time	Priority
P1	0.0	10	3
P2	4.0	1	1
P3	8.0	2	4
P4	9.0	1	5
P5	11.0	5	2

Priority scheduling (non-preemptive)

Arrival time	**P1**	**P2**	**P3**	**P4**	**P5**
	0	**4**	**8**	**9**	**11**

P1	P2	P5	P3	P4

0 10 11 16 18 19

Waiting time: (0+6+8+9+0)/5 =4.6(ms).

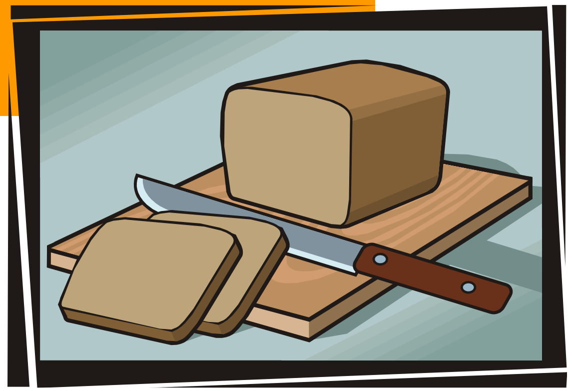
Drawback of priority

- The main drawback of priority scheduling: starvation!
- To avoid:
  - Prevent high priority processes from running indefinitely by changing priorities dynamically:
    - Each process has a base priority
    - increase priorities of waiting processes at each clock tick
  - Preserving the SJF priority
    - increase priorities of i/o bound processes
    - priority = 1/f (f is the fraction of time-quantum used)

- In fact, FCFS and SJF co
  instances of priority
  - Priority in FCFS is the a
  - Priority in SJF therefor
    corresponding proce
- The aging strategy can also be used in real life to improve the customer's satisfaction
  - Such as in **bank waiting**
    - We can upgrade the priority of the customers who have been waiting too long time

This strategy is used in the later MLFQ (Multilevel Feedback Queue) model.

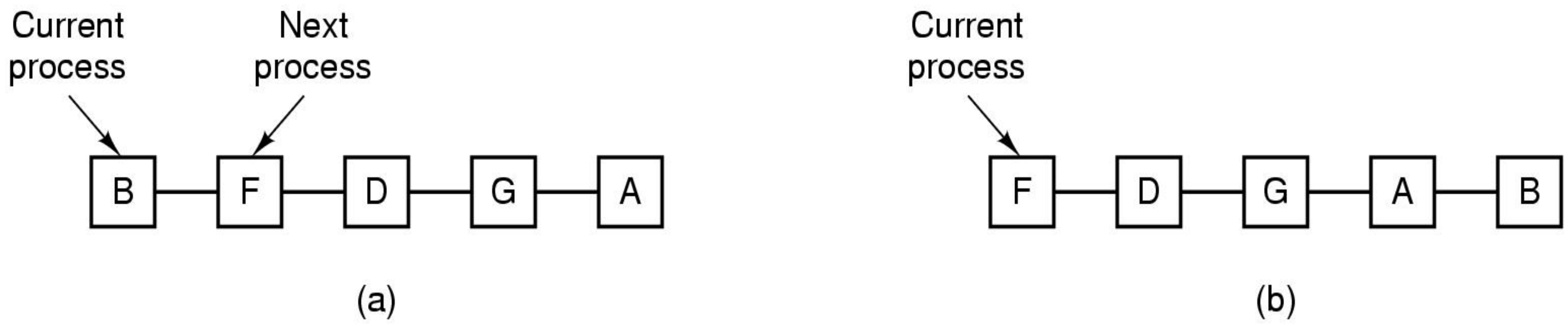
Round-Robin (RR) Scheduling

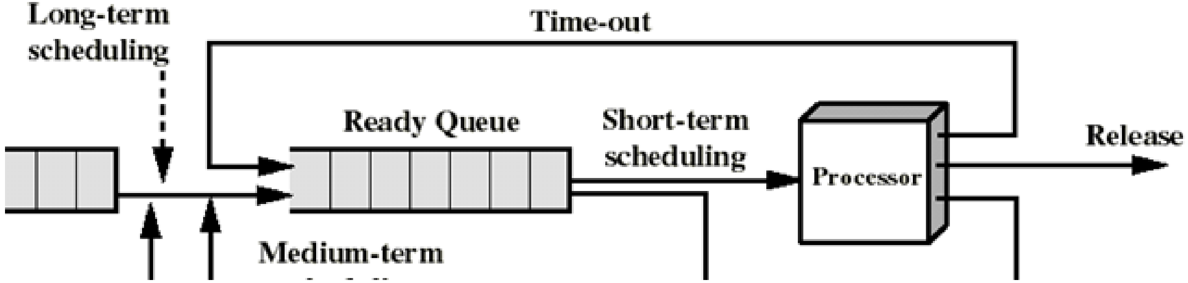


- Round-robin scheduling
  - Based on FIFO
  - Processes run only for a limited amount of time called a **time slice** or **quantum**
  - **Preemptive**
  - Requires the system to maintain several processes in memory to minimize overhead
  - Often used as part of more complex algorithms

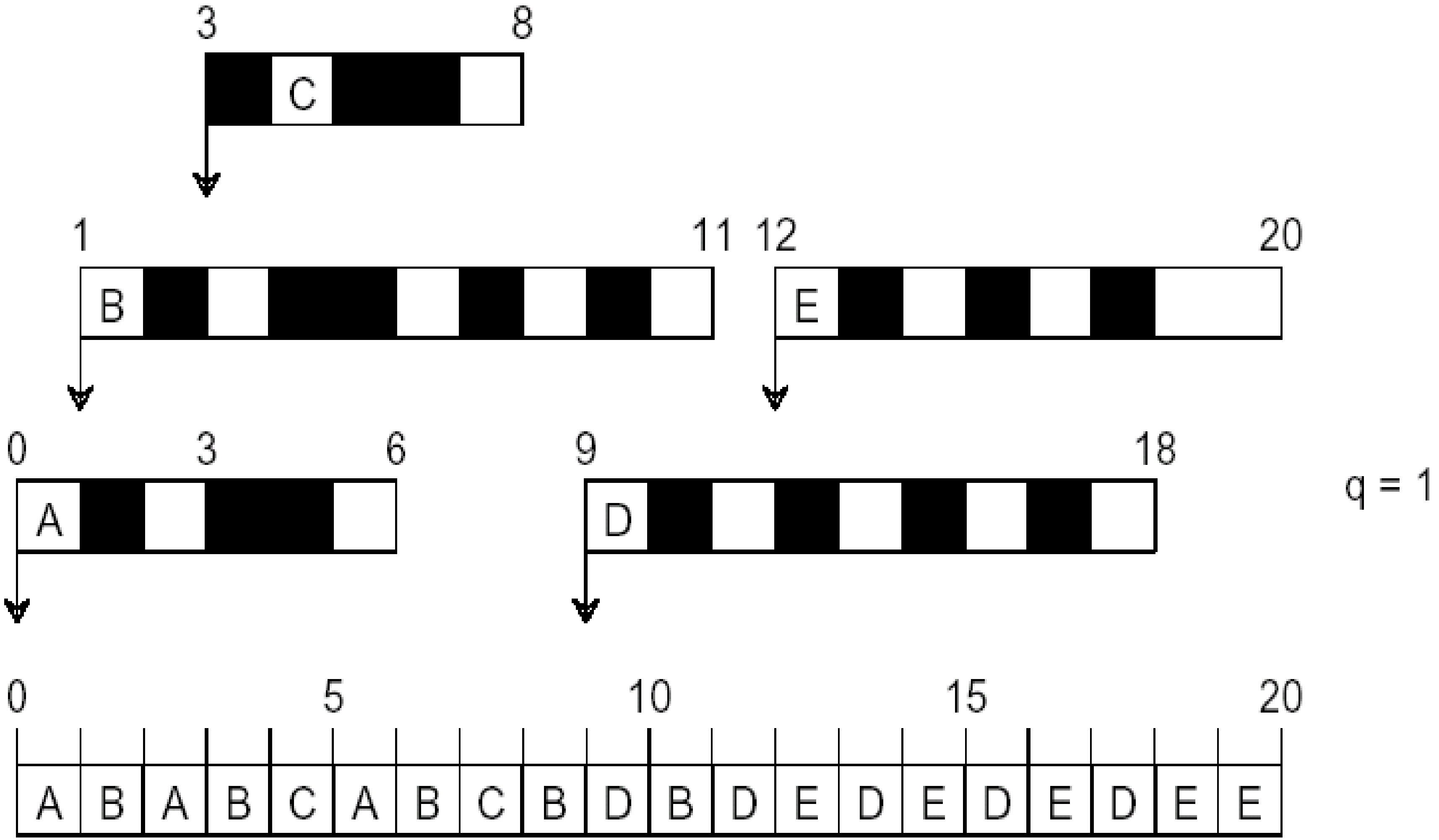
Round-Robin (RR)

- Selection function: (initially) same as FCFS.
- Decision mode: **preemptive** –
  - a process is allowed to run until the time slice period, called time quantum, is reached.
  - then a clock interrupt occurs and the running process is put at the end of the ready queue.





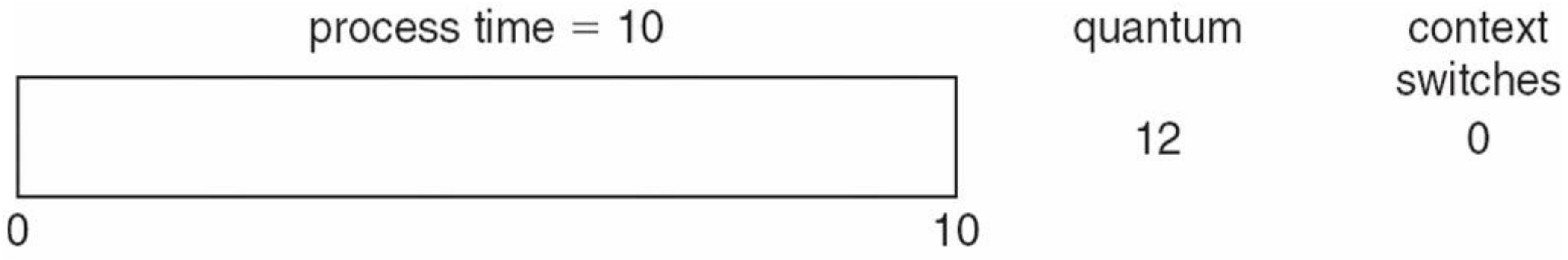
RR Example (q = 1)



Picking the Right Quantum

- Trade-off/Balancing:
  - Short quantum: great response/interactivity but high overhead
    - Hopefully not too high if the dispatcher is fast enough
  - Long quantum: poor response/interactivity, but low overhead
    - With a very long time quantum, RR Scheduling becomes FCFS Scheduling
- If context-switching time is 10% of time quantum, then the CPU spends >10% of its time doing context switches
- In practice, %CPU time spent on switching is very low
  - time quantum: 10ms to 100ms
  - context-switching time: 10μs

Picking the Right Quantum



PPTs from others\navet.ics.hawaii.edu_~casanova_courses_ics412_fall09\ics412_scheduling.ppt

RR Discussion

- Advantages
  - Jobs get fair share of CPU
  - Shortest jobs finish relatively quickly
- Disadvantages
  - Poor average waiting time with similar job lengths
    - Example: 10 jobs each requiring 10 time slices
    - RR: All complete after about 100 time slices
    - FCFS performs better!
  - Performance depends on length of time-slice
    - If time-slice too short, pay overhead of context switch
    - If time-slice too long, degenerate to FCFS

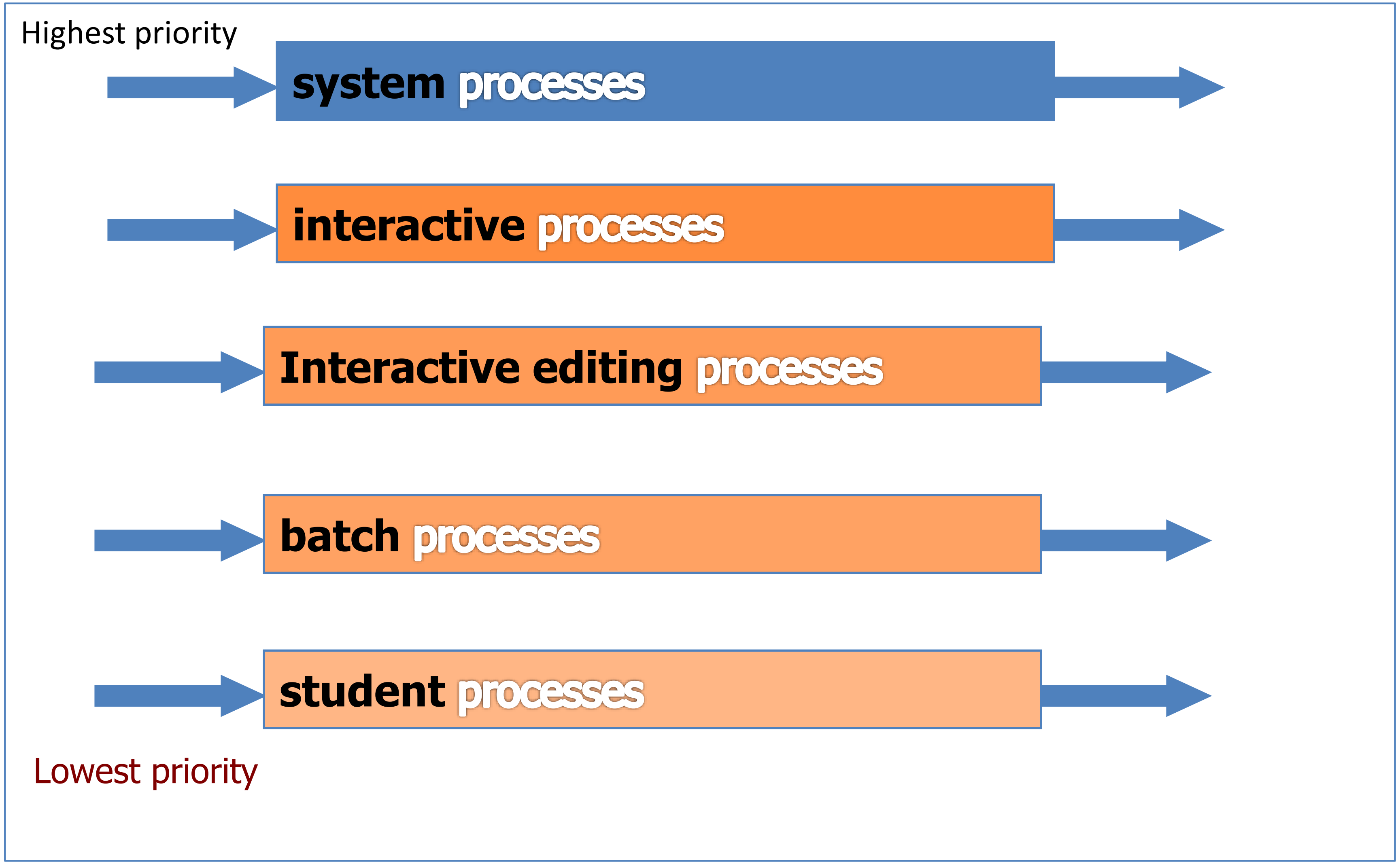
Multilevel Queue

- The RR Scheduling scheme treats all processes equally
- In practice, one often wants to classify processes in groups, e.g., based on externally-defined process priorities
  - Simple idea: use one ready queue per class of processes
    - e.g., if we support 10 priorities, we maintain 10 ready queues
- Scheduling within queues
  - Each queue has its own scheduling policy
    - e.g., High-priority could be RR, Low-priority could be FCFS
- Scheduling between the queues
  - Typically preemptive priority scheduling
    - A process can run only if all higher-priority queues are empty
  - Or time-slicing among queues
    - e.g., 80% to Queue #1 and 20% to Queue #2

Example of Multilevel Queue

- Ready queue is partitioned into separate queues:
  - foreground (**interactive**)
  - background (**batch**)
- Each queue has its own scheduling algorithm,
  - foreground – **RR**
  - background – **FCFS**
- Scheduling must be done between the queues.
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e.,
    - 80% to foreground in RR
    - 20% to background in FCFS

Multi-Level Queue Scheduling



Multilevel **Feedback** Queue

- Processes can move among the queues
  - If queues are defined on internal process characteristics, it makes sense to move a process whose characteristics have changed
    - e.g., based on CPU burst length
  - It’s also a good way to implement priority aging

Example of Multilevel Feedback Queue

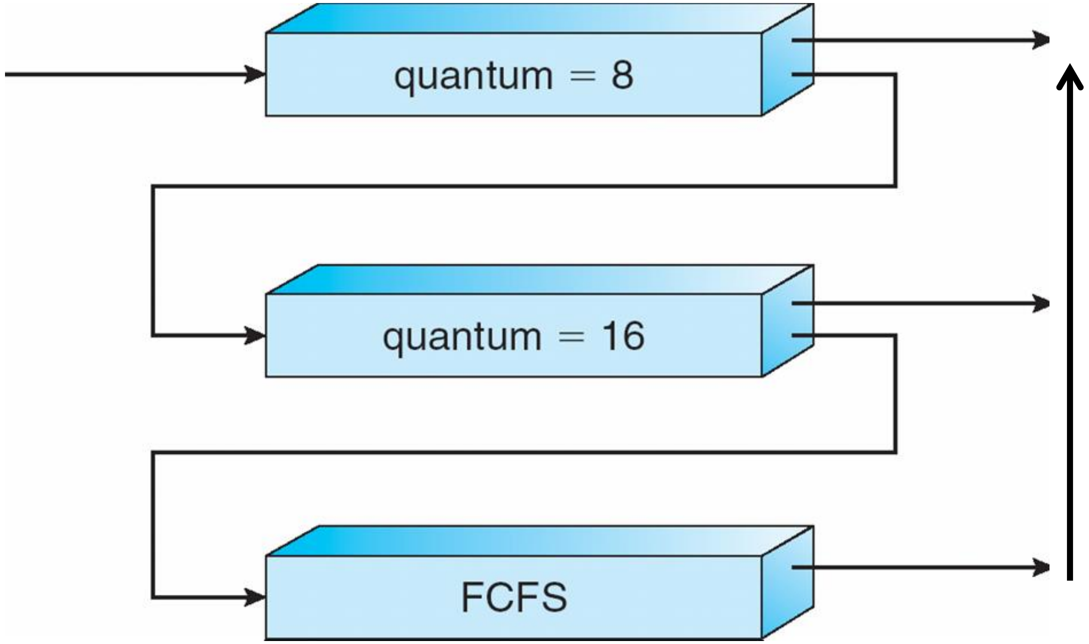
- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
  - 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.

Example of Multilevel Feedback Queue

Q0: RR (q=8)

Q1: RR (q=16)

Q2: FCFS



high

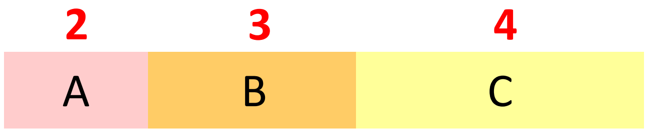
across-queue priority

low

Multilevel Feedback Queues

time = 0

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2):
- Priority 2 (time slice = 4):



Multilevel Feedback Queues

time = 1

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2):
- Priority 2 (time slice = 4):

Multilevel Feedback Queues

time = 2

- Priority 0 (time slice = 1): C
- Priority 1 (time slice = 2): A B
- Priority 2 (time slice = 4):

Multilevel Feedback Queues

time = 3

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2):

- Priority 2 (time slice = 4):

Multilevel Feedback Queues

time = 3

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2): 

- Priority 2 (time slice = 4):

suppose process A is blocked on an I/O

Multilevel Feedback Queues

time = 3

- Priority 0 (time slice = 1): A
- Priority 1 (time slice = 2): B C
- Priority 2 (time slice = 4):

suppose process A is blocked on an I/O

Multilevel Feedback Queues

time = 5

- Priority 0 (time slice = 1): A
- Priority 1 (time slice = 2): C
- Priority 2 (time slice = 4):

suppose process A is returned from an I/O

Multilevel Feedback Queues

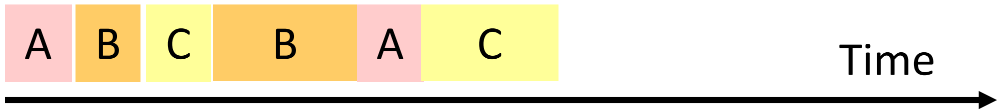
time = 6

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2): C
- Priority 2 (time slice = 4):

Multilevel Feedback Queues

time = 8

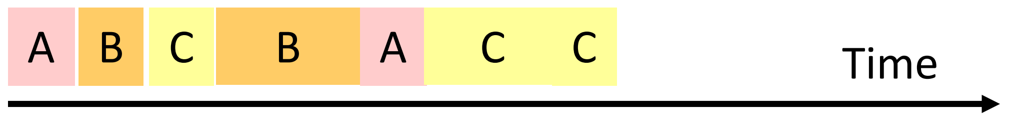
- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2):
- Priority 2 (time slice = 4): C



Multilevel Feedback Queues

time = 9

- Priority 0 (time slice = 1):
- Priority 1 (time slice = 2):
- Priority 2 (time slice = 4):



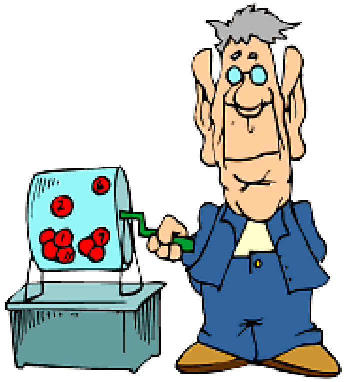
Lottery Scheduling

- Give every job some number of **lottery tickets**.
- On each time slice, randomly pick a winning ticket.
- On average, CPU time is proportional to the number of tickets given to each job.
- Assign tickets by giving the most to short running jobs, and fewer to long running jobs (**approximating SJF**). **To avoid starvation, every job gets at least one ticket**.
- Degrades gracefully as load changes. Adding or deleting a job affects all jobs proportionately, independent of the number of tickets a job has.

Lottery Scheduling Example

P1=6 *P2=9*

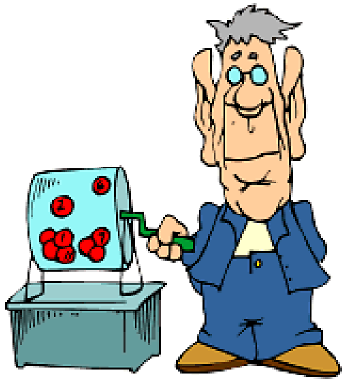
1	4	7	10	13
2	5	8	11	14
3	6	9	12	15



9

Schedule P2

Lottery Scheduling Example



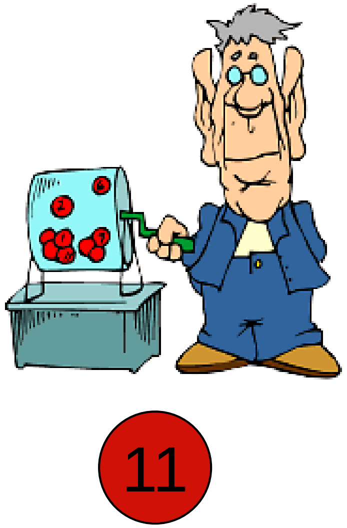
3

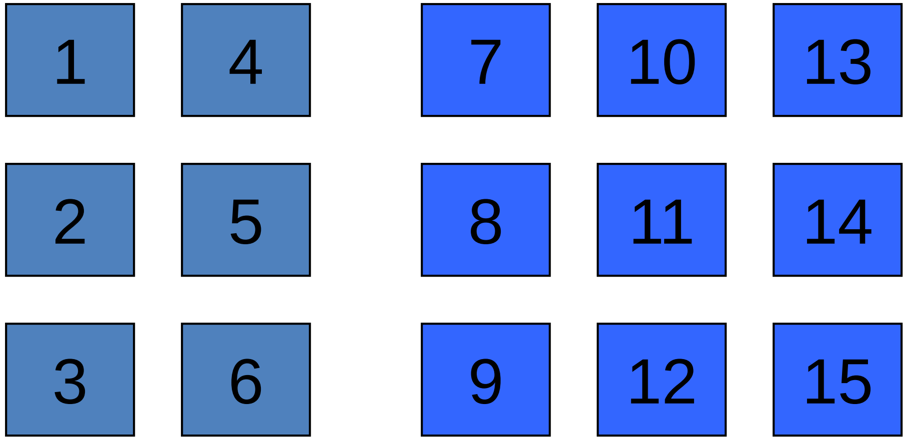
P1=6			*P2=9*	
1	4	7	10	13
2	5	8	11	14
3	6	9	12	15

Schedule P1

Lottery Scheduling Example

P1=6 *P2=9*





- As t → ∞, 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

Highest Response Ratio Next (HRRN)

- aims to minimize **Response ratio (T_q/T_s)** for each process
  - Response ratio (**RR** later) $\mathbf{T_q/T_s}$ = $(T_w + T_s)/T_s$
  - where T_w = waiting time
    T_s = expected service time
- can approximate an *a priori* measure :
  - expected service time must be estimated again
  - waiting time measured as time progresses

PPTs from others\www.dcs.ed.ac.uk_teaching_cs3_osslides\sched.ppt

CPU Scheduling

- Basic Concepts
- Scheduling Criteria & Metrics
- Different Scheduling Algorithms
- Algorithm Evaluation

Algorithms Comparison

- Which one is best?
- The answer depends on:
  - on the system workload (extremely variable).
  - hardware support for the dispatcher.
  - relative weighting of performance criteria (response time, CPU utilization, throughput...).
  - The evaluation method used (each has its limitations...).
- Hence the answer depends on too many factors to give any...

PPTs from others\From Ariel J. Frank\OS381\os9-2_cpu.ppt

Deterministic modeling

- Deterministic modeling [确定模型法] takes a particular predetermined workload and defines the performance of each algorithm for that workload.
  – To describe scheduling algorithms and provide examples,
- Deterministic model is simple and fast. It gives the exact numbers, allowing us to compare the algorithms.
- However, it requires exact numbers for input, and its answers apply only to these cases.

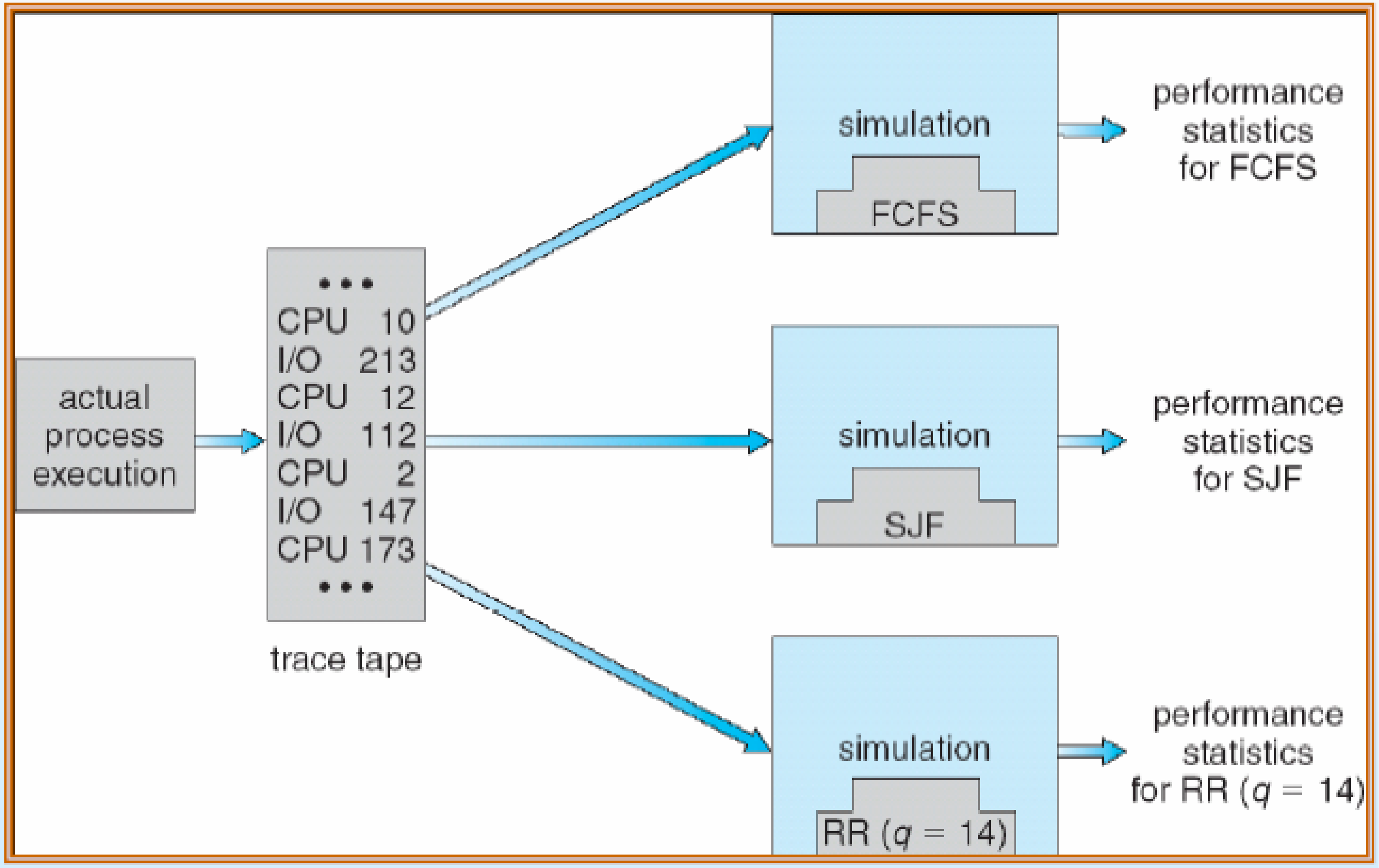
PPTs from others\OS PPT in English\ch06.ppt

Simulations

- Simulations [模拟] involve programming a model of the computer system.
  - Software **data structures** represent the major components of the system.
  - The **simulator** has a variable representing a clock; as this variable's value is increased, the simulator modifies the system to reflect the activities of the device, the processes, and the scheduler.
  - As the simulation executes, **statistics** that indicate algorithm performance are gathered and printed.
- Artificial data or trace tapes.

PPTs from others\OS PPT in English\ch06.ppt

Simulations



PPTs from others\www.cs.gsu.edu_~cscbecx\csc4320 Chapter 5-3.ppt

Simulation - Queuing models

- Queuing models [排队模型]
  - **Queueing-network analysis**
    - The computer system is described as a network of servers. Each server has a queue of waiting process. The CPU is a server with its ready queue, as is the I/O
    - Knowing arrival rates and se utilization, average queu
  - Useful for comparing sch distributions are difficult to work with, and some assumptions required.

Power of mathematics – this is the general way to find solutions for problems

PPTs from others\OS PPT in English\ch06.ppt

- ICQ A [10 pts]
  - Next week
  - 1 hour, close
  - Blank-filling, Computation