Chapter 5: CPU Scheduling

肖 卿 俊
办公室：计算机楼212室
电邮：csqjxiao@seu.edu.cn
主页：http://cse.seu.edu.cn/PersonalPage/csqjxiao
电话：025-52091022

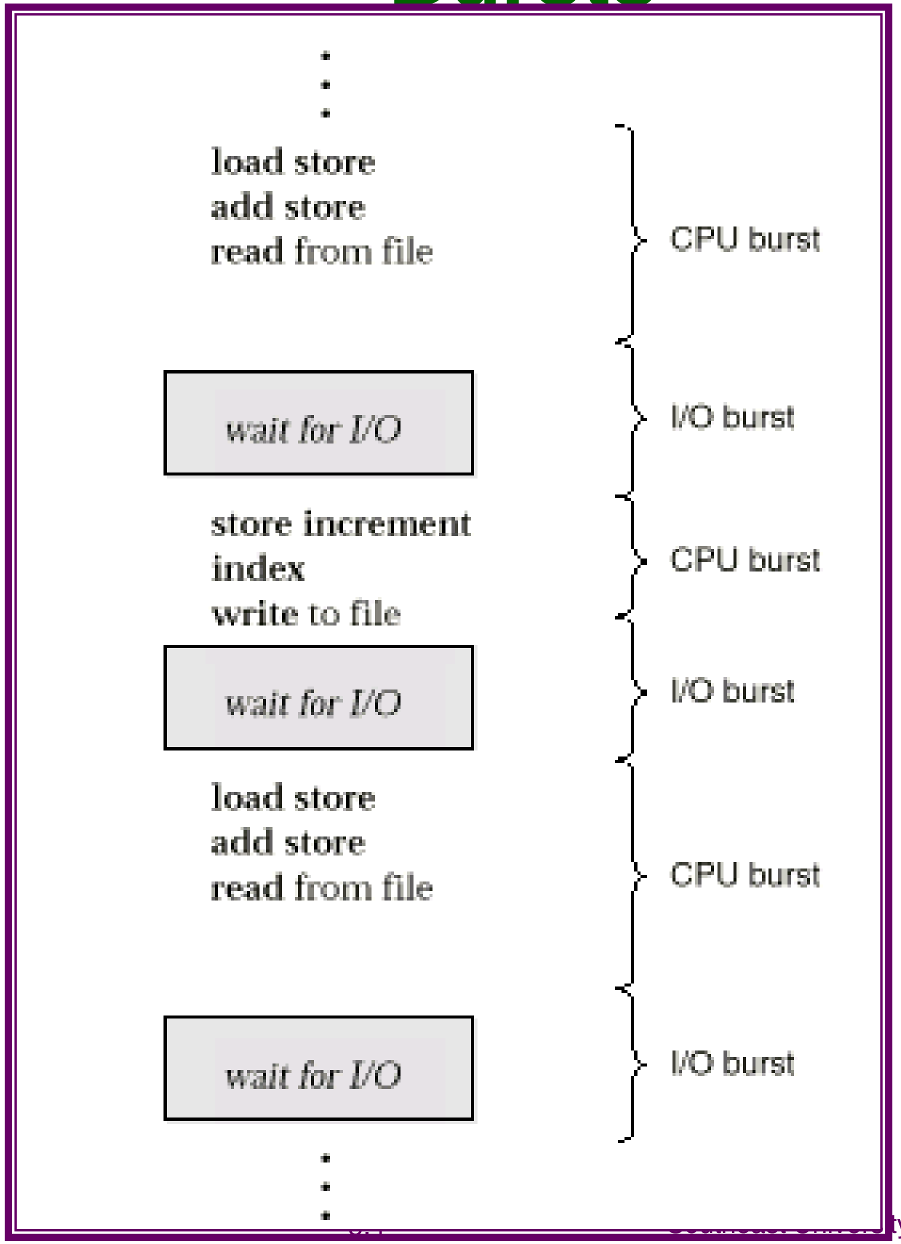
Chapter 5: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Thread Scheduling
- Operating System Examples
- Real-Time Scheduling
- Algorithm Evaluation
- Process Scheduling Models

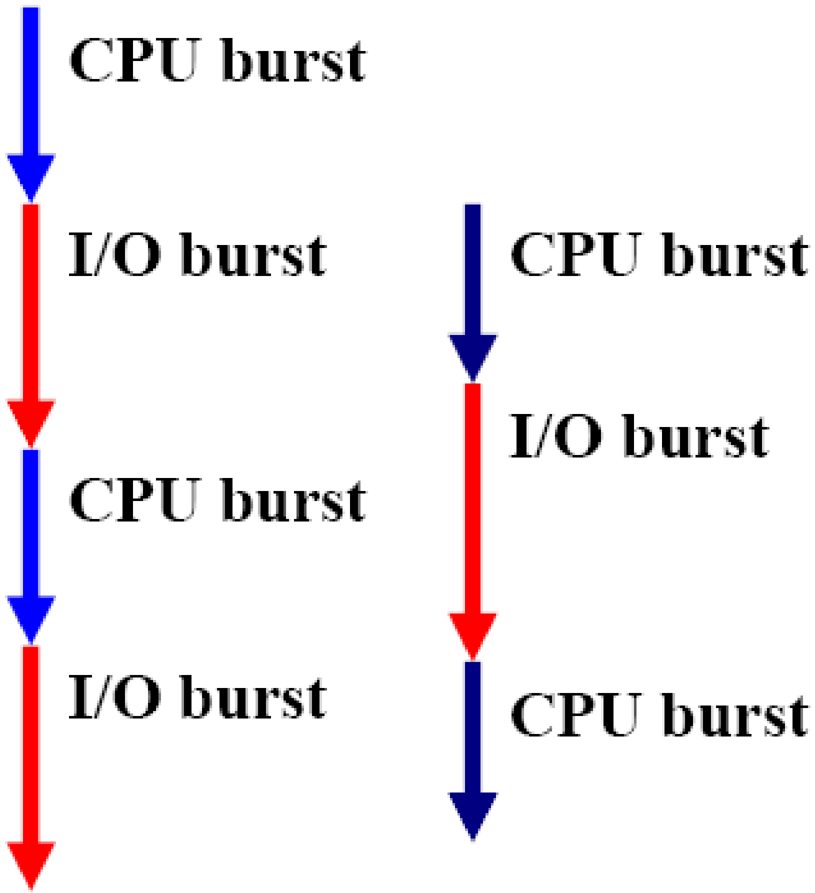
Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait.

Alternating Sequence of CPU And I/O Bursts



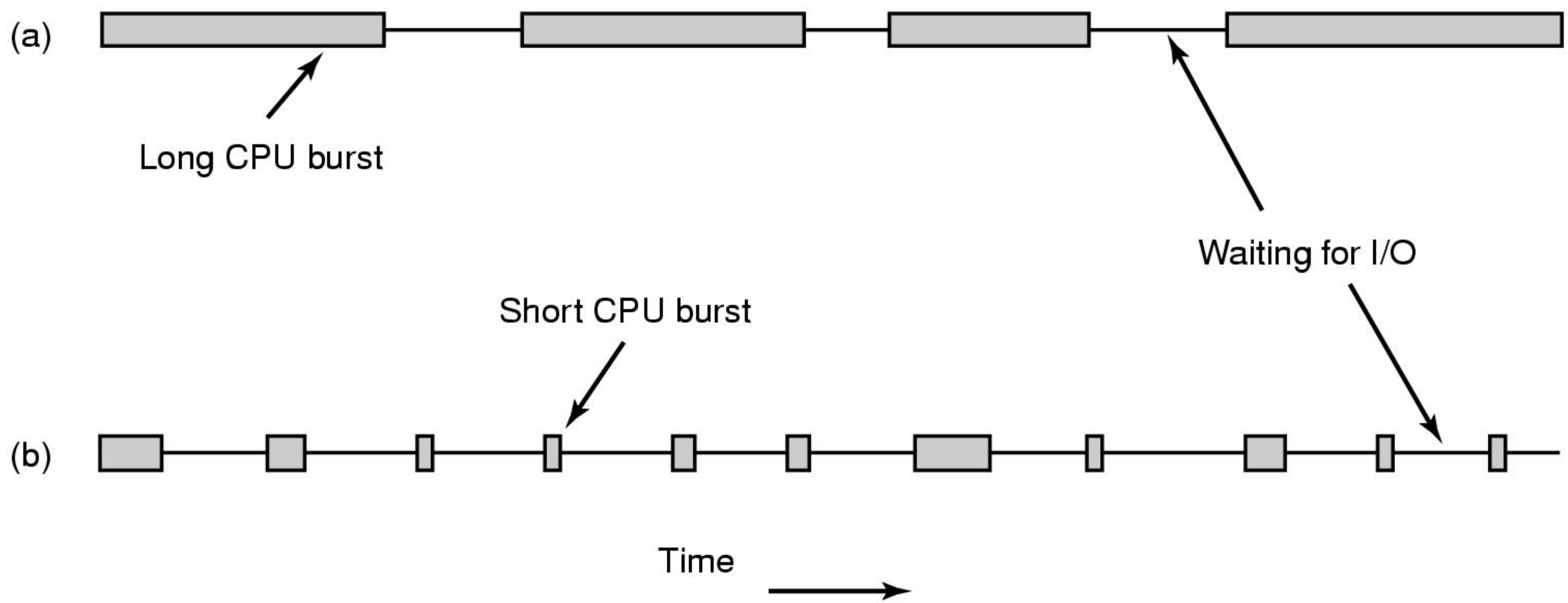
CPU-I/O Burst Cycle



- **Process execution repeats the CPU burst and I/O burst cycle.**
- **When a process begins an I/O burst, another process can use the CPU for a CPU burst.**

CPU-bound and I/O-bound

- A process is CPU-bound if it generates I/O requests infrequently, using more of its time doing computation.
- A process is I/O-bound if it spends more of its time to do I/O than it spends doing computation.
- A CPU-bound process might have a few very long CPU bursts.
- An I/O-bound process typically has many short CPU bursts


(a)
Long CPU burst
Waiting for I/O
Short CPU burst
(b)
Time

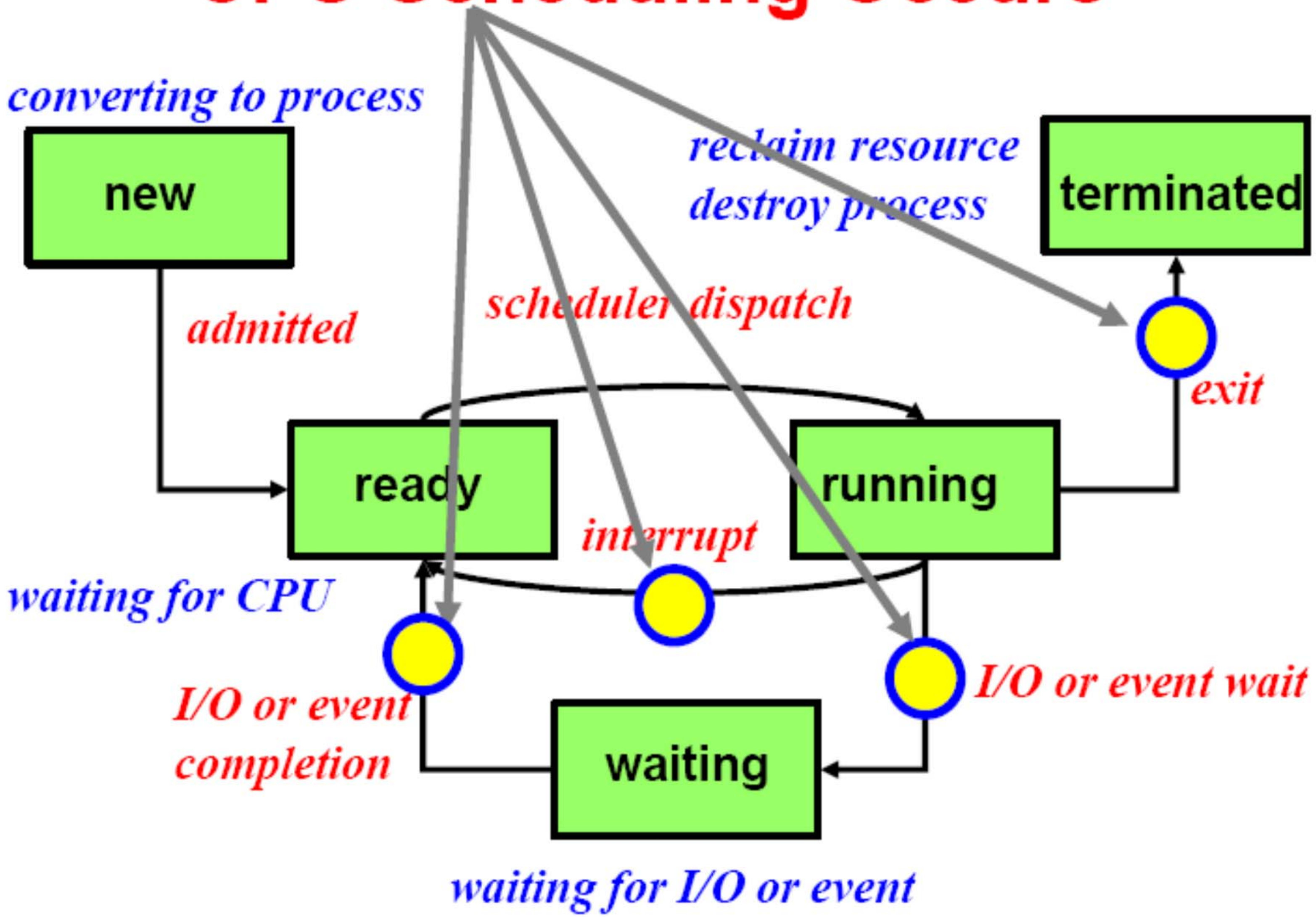
CPU Scheduler

- **When the CPU is idle, the OS must select another process to run.**
- **This selection process is carried out by the *short-term scheduler* (or *CPU scheduler*).**
- **The CPU scheduler selects a process from the ready queue, and allocates the CPU to it.**
- **The ready queue does not have to be a FIFO one. There are many ways to organize the ready queue.**

Circumstances that scheduling may take place

1. A process switches from the **running** state to the **wait** state (*e.g.*, doing for I/O)
2. A process switches from the **running** state to the ready state (*e.g.*, an **interrupt** occurs)
3. A process switches from the **wait** state to the **ready** state (*e.g.*, I/O completion)
4. A process **terminates**

CPU Scheduling Occurs



Preemptive vs. Non-preemptive

- *Non-preemptive scheduling*: scheduling occurs when a process voluntarily enters the wait state (case 1) or terminates (case 4).
  - Simple, but very inefficient
- *Preemptive scheduling*: scheduling occurs in all possible cases.
  - What if the kernel is in its critical section modifying some important data? Mutual exclusion may be violated.
  - The kernel must pay special attention to this situation and, hence, is more complex

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- *Dispatch latency* – time it takes for the dispatcher to stop one process and start another running.

Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Algorithm Evaluation

Scheduling Criteria

- **There are many criteria for comparing different scheduling algorithms. Here are five common ones:**
  - CPU utilization
  - Throughput
  - Turnaround time
  - Waiting time
  - Response time

CPU Utilization

- We want to keep the CPU as busy as possible.
- CPU utilization ranges from 0 to 100 percent. Normally 40% is lightly loaded and 90% or higher is heavily loaded.
- You can bring up a CPU usage meter to see CPU utilization on your system.

Throughput

- The number of processes completed per time unit is called *throughput*.
- Higher throughput means more jobs get done.
- However, for long processes, this rate may be one job per hour, and, for short jobs, this rate may be 10 per minute.

Turnaround Time

- The time period between job submission to completion is the turnaround time.
- From a user's point of view, turnaround time is more important than CPU utilization and throughput.
- Turnaround time is the sum of
  - waiting time before entering the system
  - waiting time in the ready queue
  - waiting time in all other events (e.g., I/O)
  - time the process actually running on the CPU

Waiting Time

- Waiting time is the sum of the periods that a process spends waiting in the ready queue.
- Why only ready queue?
  - CPU scheduling algorithms do not affect the amount of time during which a process is waiting for I/O and other events.
  - However, CPU scheduling algorithms do affect the time that a process stays in the ready queue

Response Time

- The time from the submission of a request (in an interactive system) to the first response is called response time. It does not include the time that it takes to output the response.
- For example, in front of your workstation, you perhaps care more about the time between hitting the Return key and getting your first output than the time from hitting the Return key to the completion of your program (e.g., turnaround time).

Optimization Criteria

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

Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Algorithm Evaluation

Scheduling Algorithms

- We will discuss a number of scheduling algorithms:
  - First-Come, First-Served (FCFS)
  - Shortest-Job-First (SJF)
  - Priority
  - Round-Robin
  - Multilevel Queue
  - Multilevel Feedback Queue

First-Come, First-Served (FCFS) Scheduling

- The process that requests the CPU first is allocated the CPU first.
- This can easily be implemented using a queue.
- FCFS is not preemptive. Once a process has the CPU, it will occupy the CPU until the process completes or voluntarily enters the wait state.

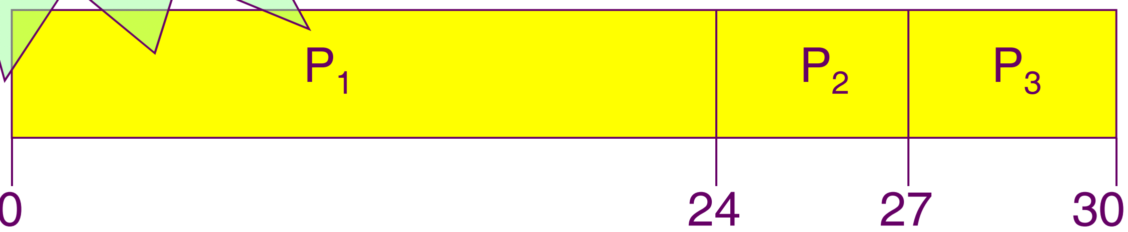
FCFS Scheduling (Cont.)

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

Average waiting time?

Waiting time?

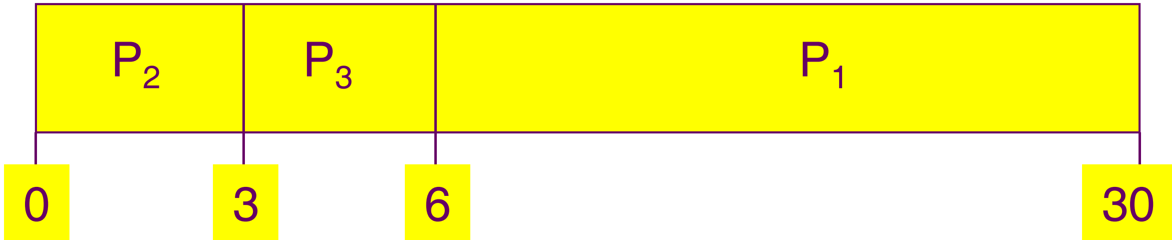


FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1 .$$

- The Gantt chart for the schedule is:



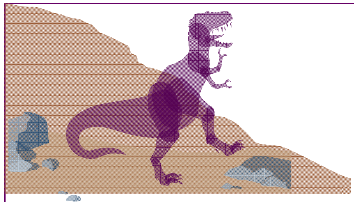
- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case.
- *Convoy effect* short process behind long process

FCFS Problems

- It is easy to have the *convoy effect*: all the processes wait for the one big process to get off the CPU. CPU utilization may be low.
- Consider a CPU-bound process running with many I/O-bound process.
- It is in favor of long processes and may not be fair to those short ones. What if your 1-minute job is behind a 10-hour job?
- It is troublesome for time-sharing systems, where each user needs to get a share of the CPU at regular intervals.

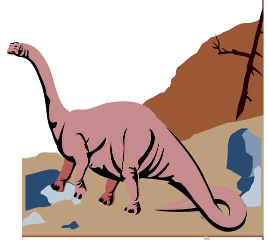
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- When a process must be selected from the ready queue, the process with the smallest next CPU burst is selected.
- Thus, the processes in the ready queue are sorted in CPU burst length.

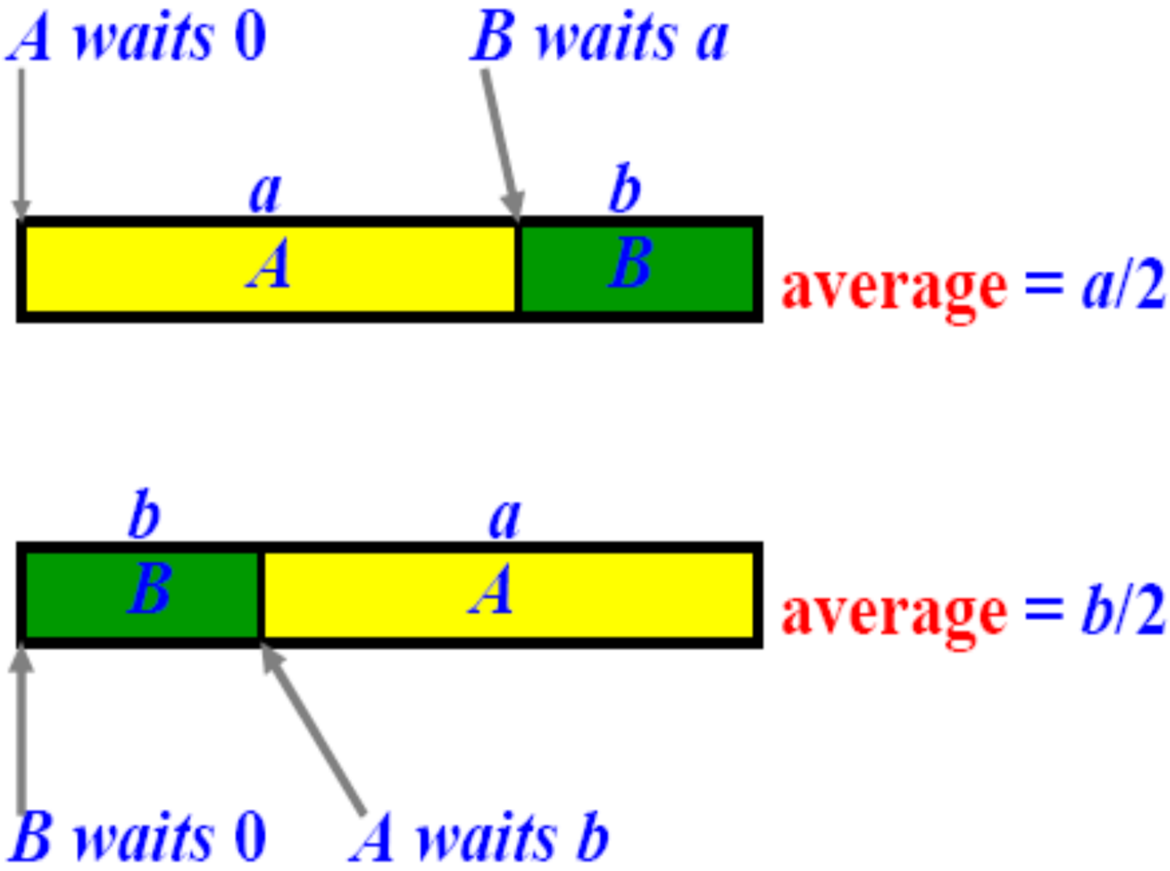


Shortest-Job-First (SJF) Scheduling (Cont.)

- SJF can be non-preemptive or preemptive.
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.



SJF is provably optimal

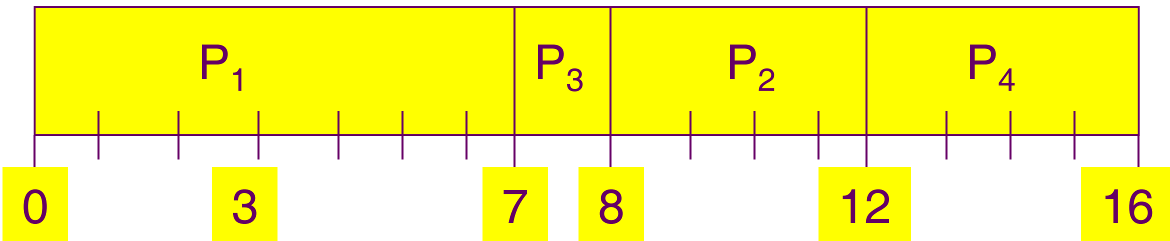


- Every time we make a short job before a long job, we reduce average waiting time.
- We may switch out of order jobs until all jobs are in order.
- If the jobs are sorted, job switching is impossible.

Example of Non-Preemptive 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)

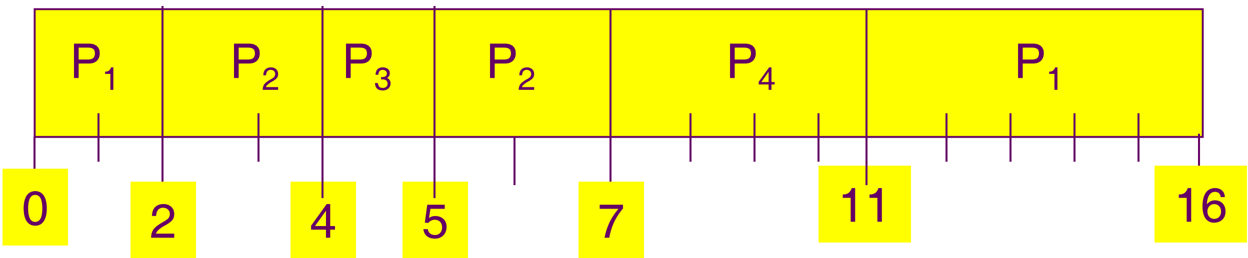


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive 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 (preemptive)



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

How do we know the Next CPU Burst?

- Without a good answer to this question, SJF cannot be used for CPU scheduling.
- We try to predict the next CPU burst!
- Can be done by using the length of previous CPU bursts, using exponential averaging.

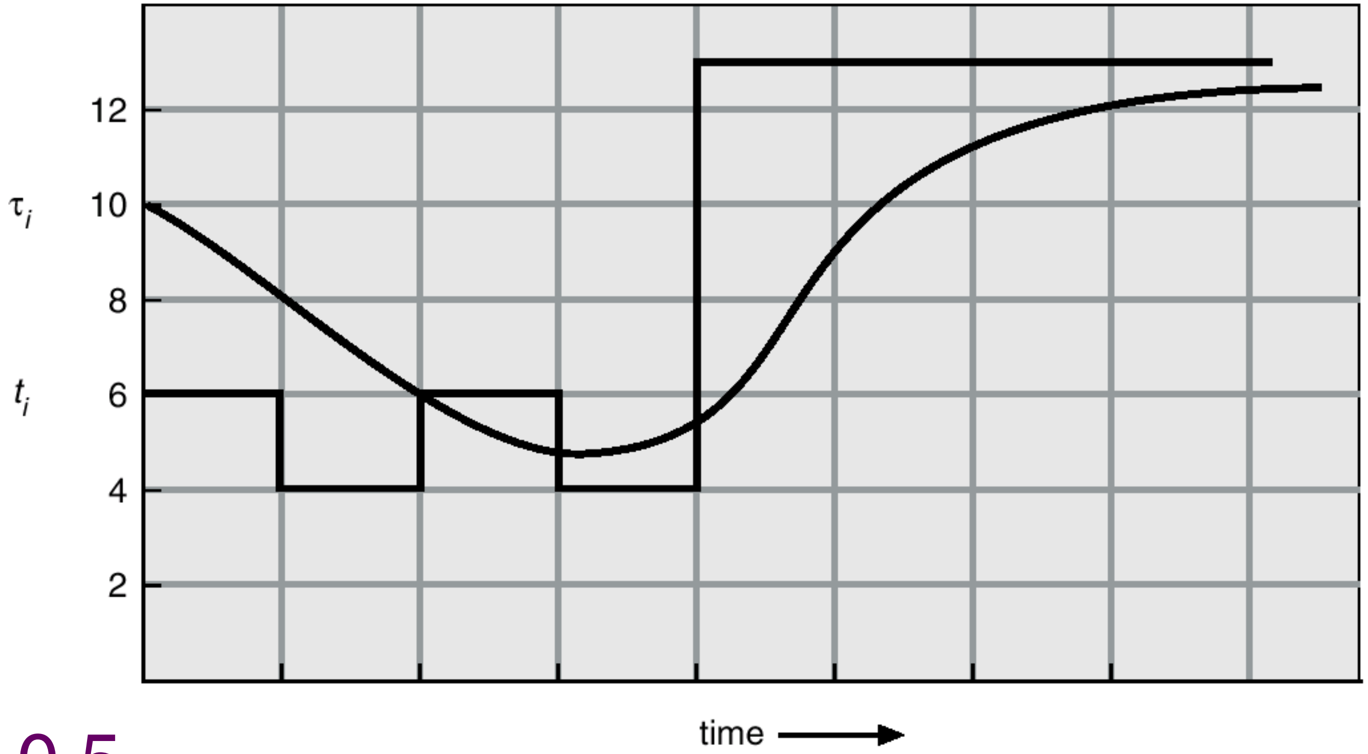
1. $t_n =$ actual lenght of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n .$$

Examples 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.

Prediction of the Length of the Next CPU Burst



$\alpha = 0.5$

CPU burst (t_i)		6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10								. . .

What are predicted values?

Examples of Exponential Averaging (Cont.)

- If we expand the formula, we get:

$$
\begin{aligned}
\tau_{n+1} = \alpha\, t_n &+ (1 - \alpha)\, \alpha\, t_{n-1} + \dots \\
&+ (1 - \alpha)^j\, \alpha\, t_{n-j} + \dots \\
&+ (1 - \alpha)^{n+1}\, \tau_0
\end{aligned}
$$

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

- Many time series prediction tools

SJF Problems

- It is difficult to estimate the next burst time value accurately.
- SJF is in favor of short jobs. As a result, some long jobs may not have a chance to run at all. This is *starvation*.

Priority Scheduling

- Each process has a *priority*.
- Priority may be determined internally or externally:
  - internal priority: determined by time limits,memory requirement, # of files, and so on.
  - external priority: not controlled by the OS (*e.g.*, importance of the process)
- The scheduler always picks the process (in ready queue) with the highest priority to run.
- FCFS and SJF are special cases of priority scheduling. (Why?)

Priority Scheduling (Cont.)

- Priority scheduling can be non-preemptive or preemptive.
- With preemptive priority scheduling, if the newly arrived process has a higher priority than the running one, the latter is preempted.
- Indefinite block (or starvation) may occur: a low priority process may never have a chance to run

Aging

- Aging is a technique to overcome the starvtion problem.
- *Aging*: gradually increases the priority of processes that wait in the system for a long time.
- Example:
  - If 0 is the highest (*resp*., lowest) priority, then we could decrease (*resp*., increase) the priority of a waiting process by 1 every fixed period (*e.g.*, every minute).

Round Robin (RR)

- RR is similar to FCFS, except that each process is assigned a time quantum.
- All processes in the ready queue is a FIFO list.
- When the CPU is free, the scheduler picks the first and lets it run for one time quantum.
- If that process uses CPU for less than one time quantum, it is moved to the tail of the list.
- Otherwise, when one time quantum is up, that process is preempted by the scheduler and moved to the tail of the list.

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

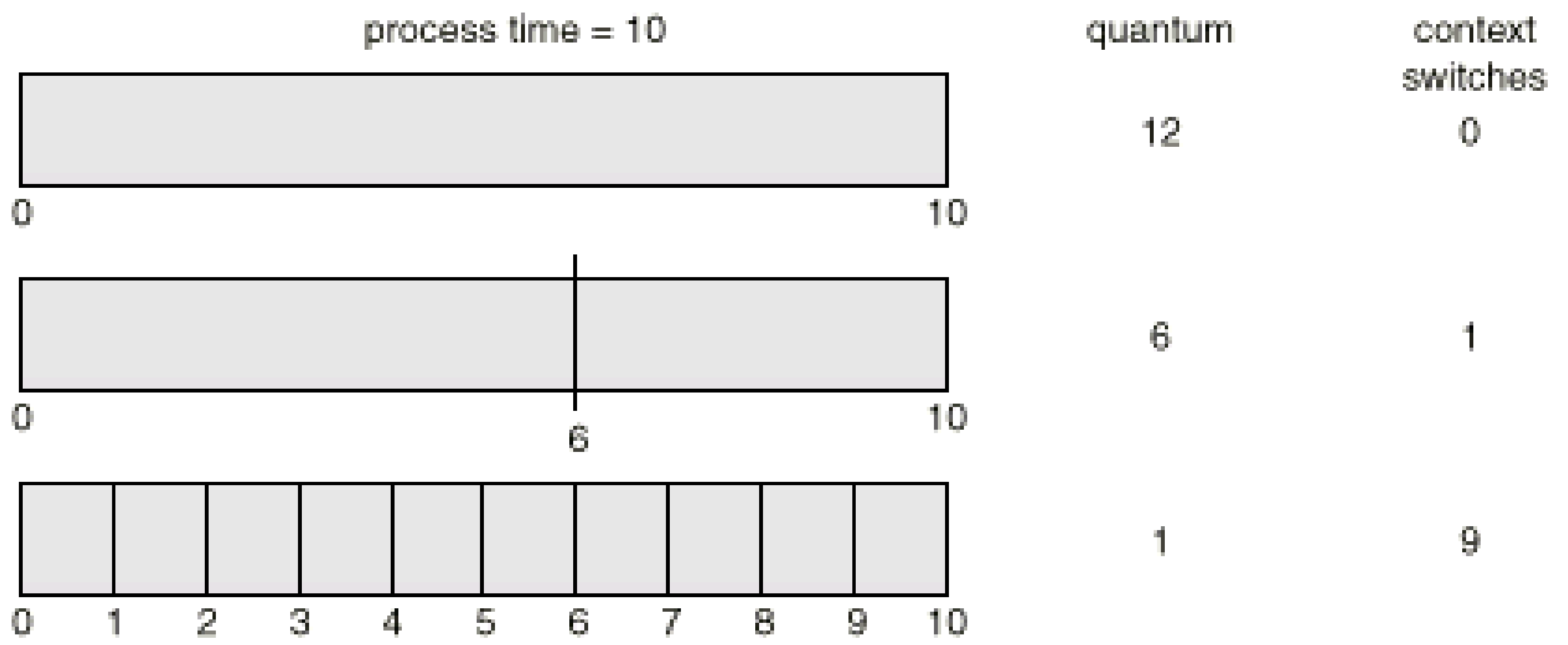
P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 37 57 77 97 117 121 134 154 162

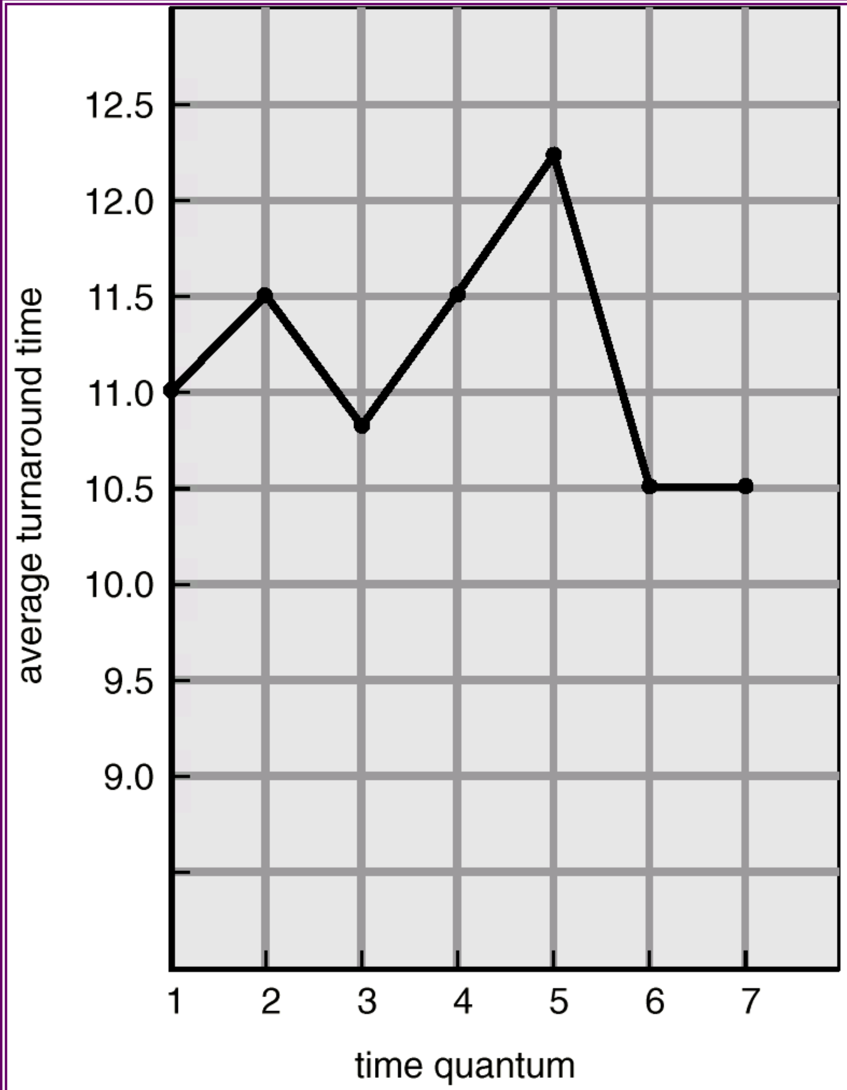
RR Scheduling: Some Issues

- If time quantum is too large, RR reduces to FCFS
- If time quantum is too small, RR becomes processor sharing
- Context switching may affect RR's performance
  - Shorter time quantum means more context switches
- Turnaround time also depends on the size of time quantum.
- In general, 80% of the CPU bursts should be shorter than the time quantum

Time Quantum and Context Switch Time



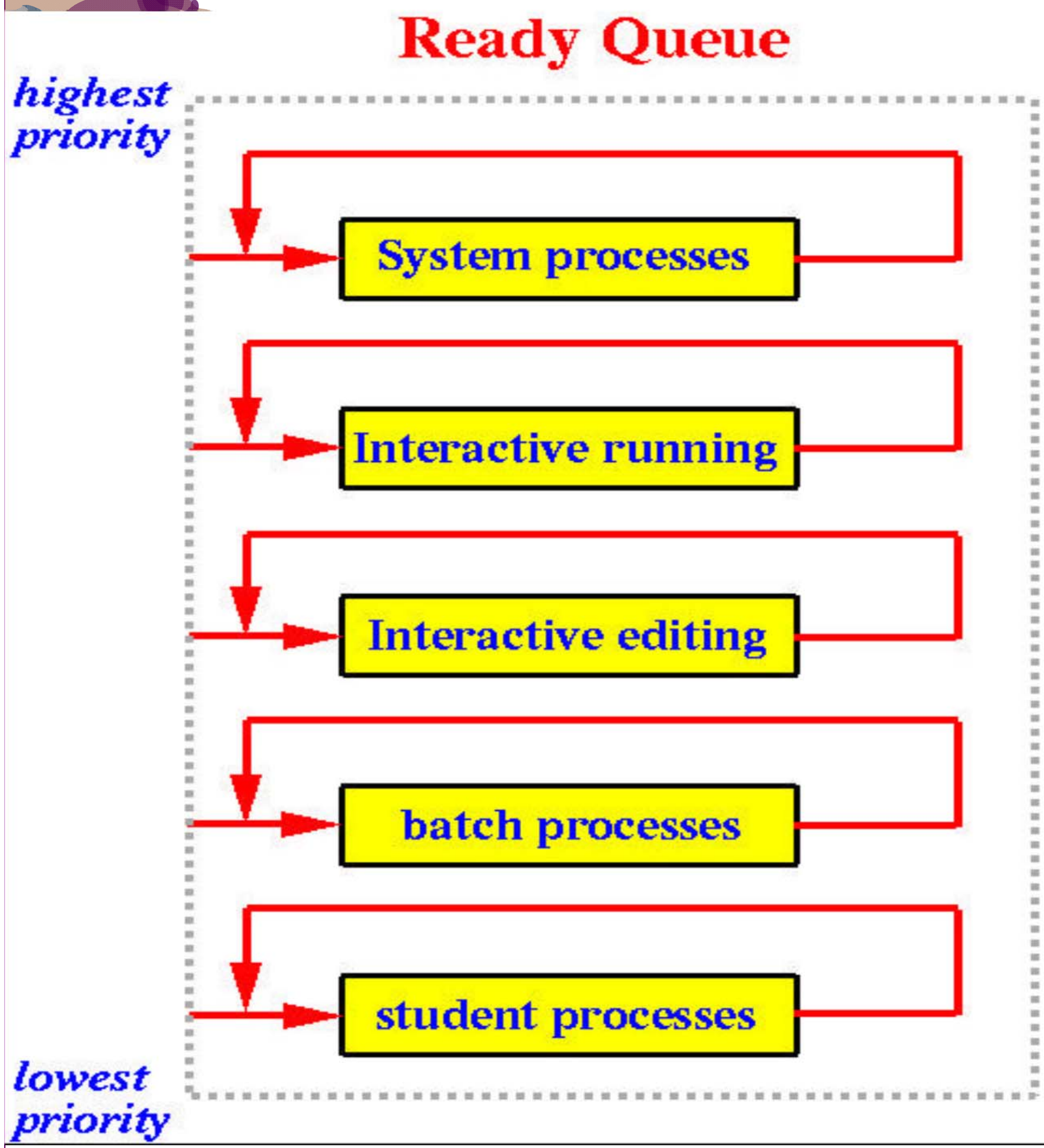
Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Multilevel Queue

- Ready queue is partitioned into separate queues:
  foreground (interactive)
  background (batch)
- Each process is assigned permanently to one queue based on some properties of the process (*e.g.*, memory usage, priority, process type)
- Each queue has its own scheduling algorithm,
  foreground – RR
  background – FCFS



- A process P can run only if all queues above the queue that contains P are empty.
- When a process is running and a process in a higher priority queue comes in, the running process is preempted.

Multilevel Queue (Cont.)

- 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

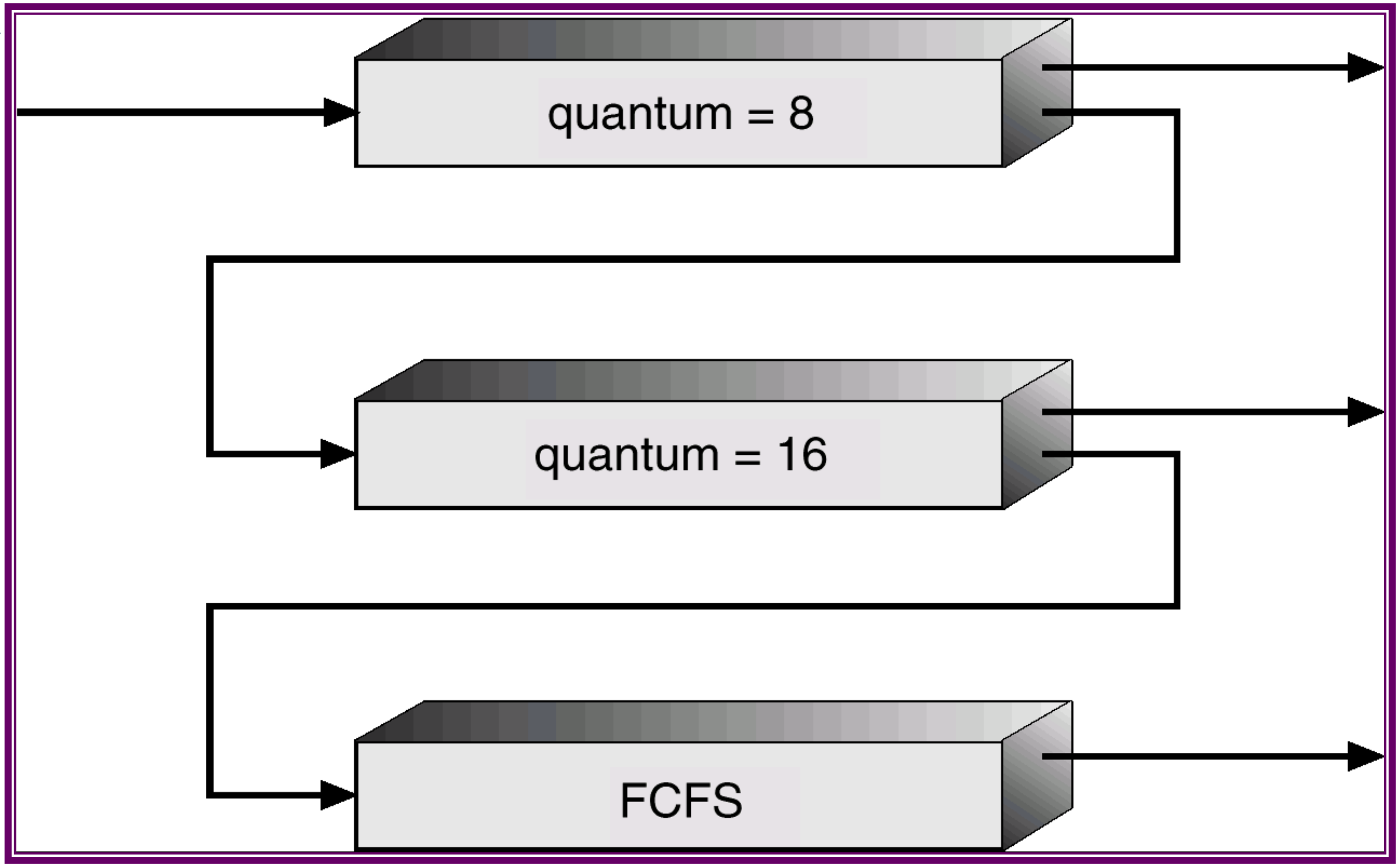
Multilevel Feedback Queue

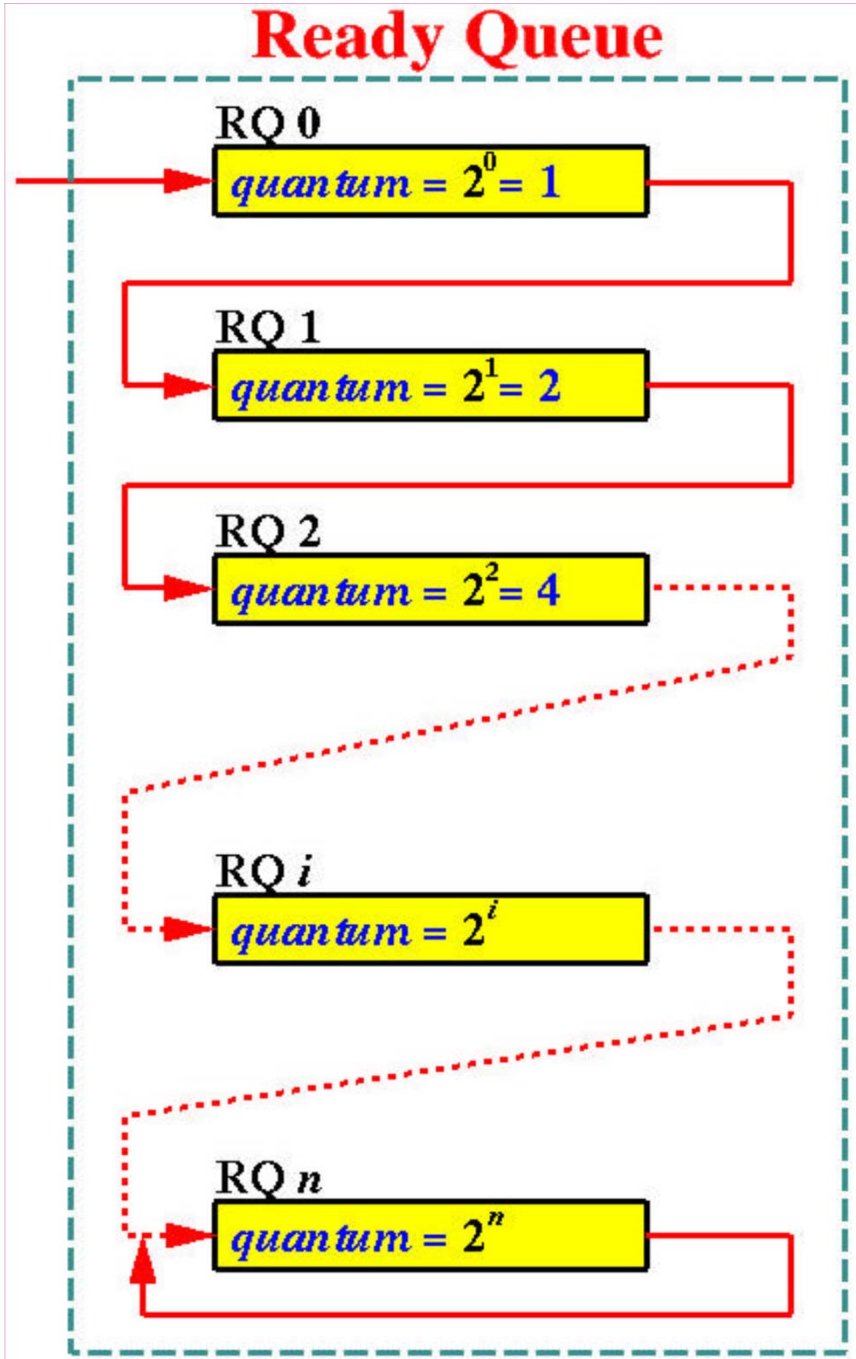
- *Multilevel queue with feedback scheduling* is similar to multilevel queue; however, it allows processes to move between queues.
  - aging can be implemented this way
- If a process uses more (*resp*., less) CPU time, it is moved to a queue of lower (*resp*., higher) priority.
- As a result, I/O-bound (*resp*., CPU-bound) processes will be in higher (*resp*., lower) priority queues.

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – time quantum 8 milliseconds
  - Q_1 – time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters 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 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.

Multilevel Feedback Queues





- Processes in queue *i* have time quantum 2^i
- When a process' behavior changes, it may be placed (*i.e.*, promoted or demoted) into a difference queue.
- Thus, when an I/O-bound process starts to use more CPU, it may be demoted to a lower queue

Multilevel Feedback Queue (Cont.)

- 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

Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Algorithm Evaluation

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous processors* within a multiprocessor.
- *Load sharing*
- *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing.

Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Thread Scheduling
- Algorithm Evaluation

Thread Scheduling

- Local Scheduling – How the threads library decides which thread to put onto an available LWP
- Global Scheduling – How the kernel decides which kernel thread to run next

Pthread Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[])
{    int i;
     pthread_t tid[NUM_THREADS];
     pthread_attr t attr;
     /* get the default attributes */
     pthread_attr_init(&attr);
     /* set the scheduling algorithm to PROCESS or SYSTEM */
     pthread_attr_setscope(&attr, PTHREAD SCOPE SYSTEM);
     /* set the scheduling policy - FIFO, RT, or OTHER */
     pthread_attr_setschedpolicy(&attr, SCHED_OTHER);
     /* create the threads */
     for (i = 0; i < NUM_THREADS; i++)
          pthread_create(&tid[i],&attr,runner,NULL);
```

Pthread Scheduling API

```
    /* now join on each thread */
    for (i = 0; i < NUM_THREADS; i++)
        pthread_join(tid[i], NULL);
}
 /* Each thread will begin control in this
    function */
void *runner(void *param)  {
    printf("I am a thread\n");
    pthread exit(0);
}
```

SCHED_OTHER is the standard Linux time-sharing scheduler that is intended for all processes that do not require the special real-time mechanisms.

http://linux.die.net/man/2/sched_setscheduler

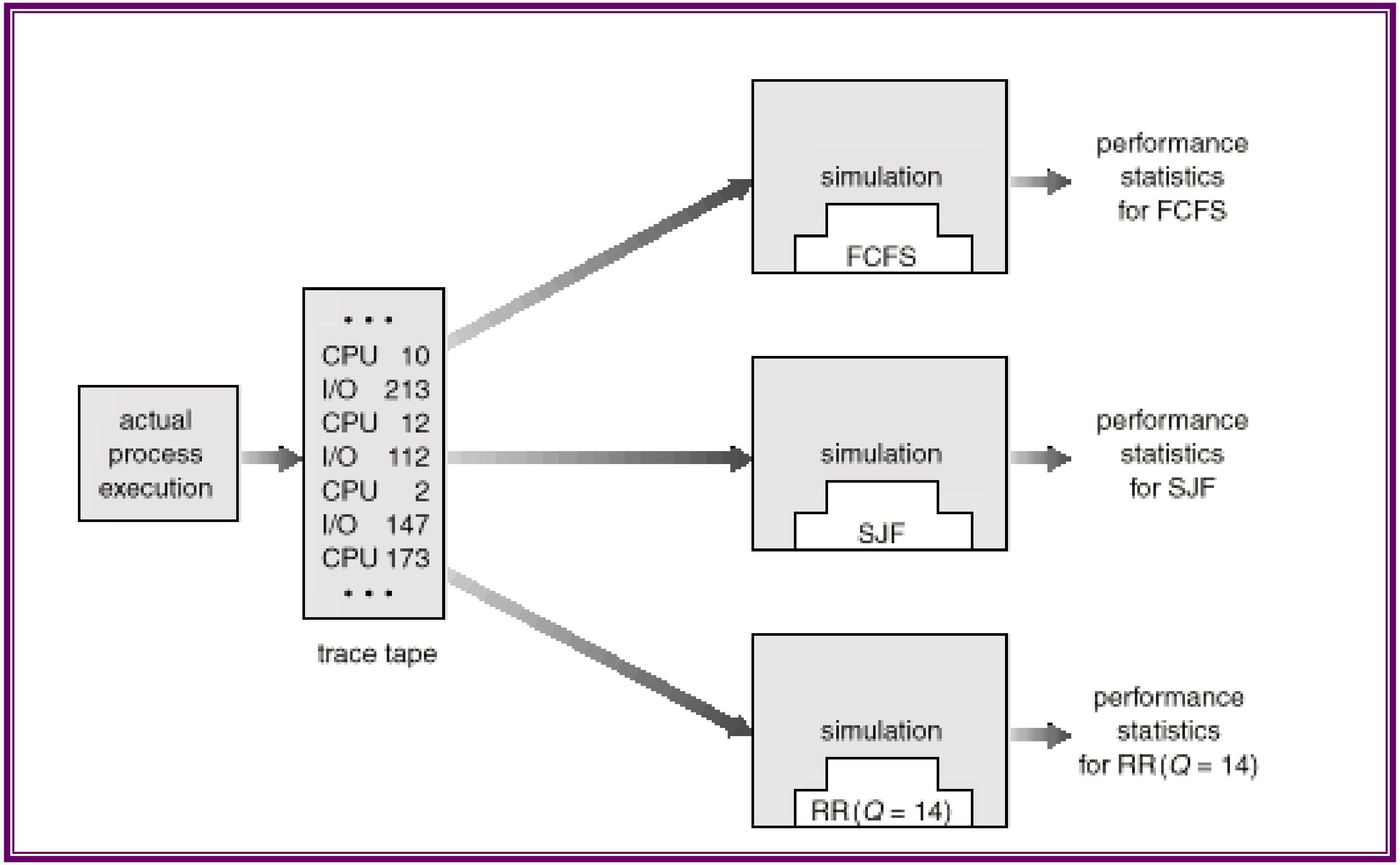
Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Algorithm Evaluation

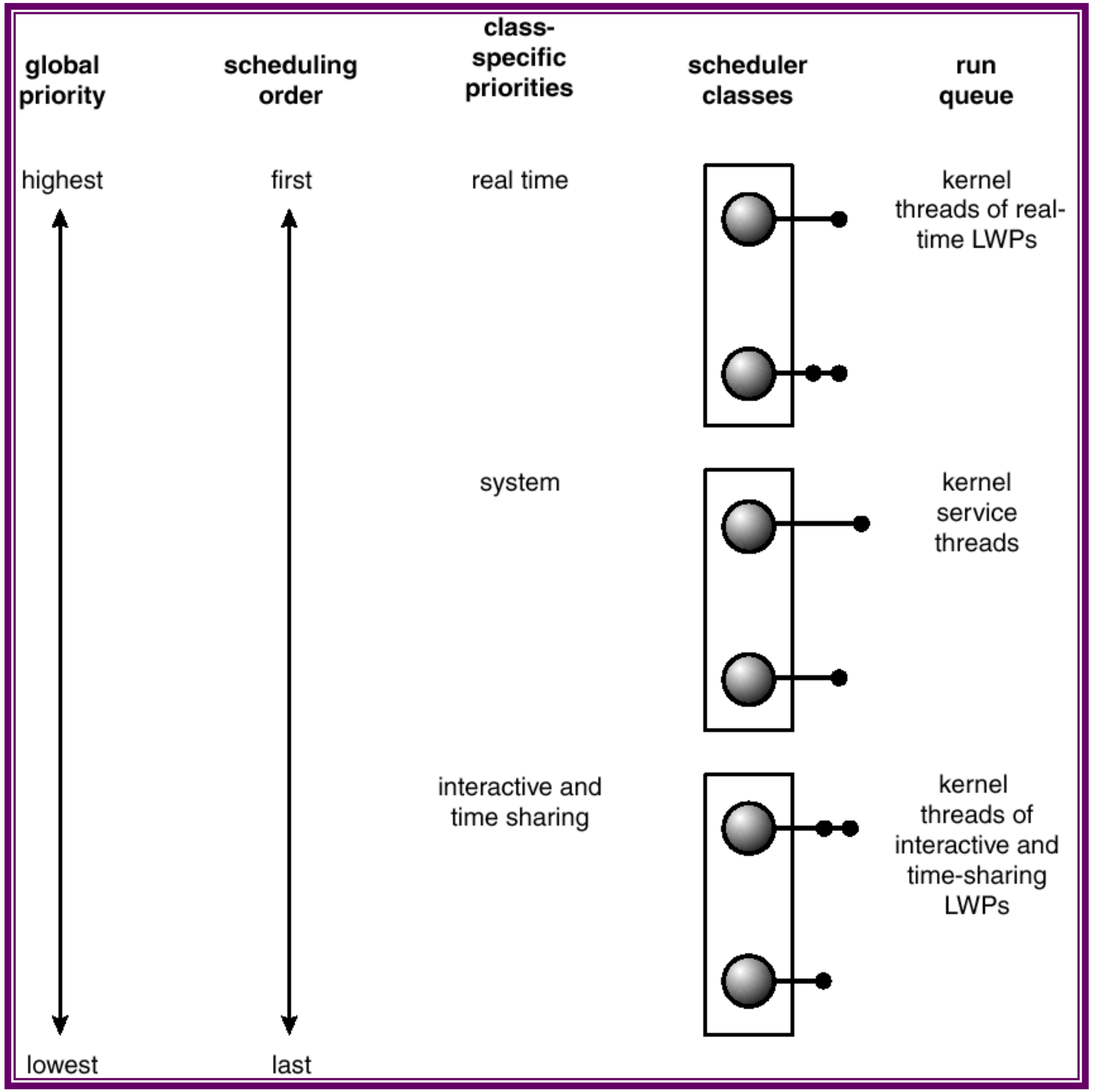
Algorithm Evaluation

- Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload.
- Queuing models
- Simulations
- Implementation

Evaluation of CPU Schedulers by Simulation



Solaris 2 Scheduling



Solaris Dispatch Table

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Windows 2000(XP) Priorities

Thread priority level

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Process priority class

https://msdn.microsoft.com/en-us/library/windows/desktop/ms685100(v=vs.85).aspx

Linux Scheduling

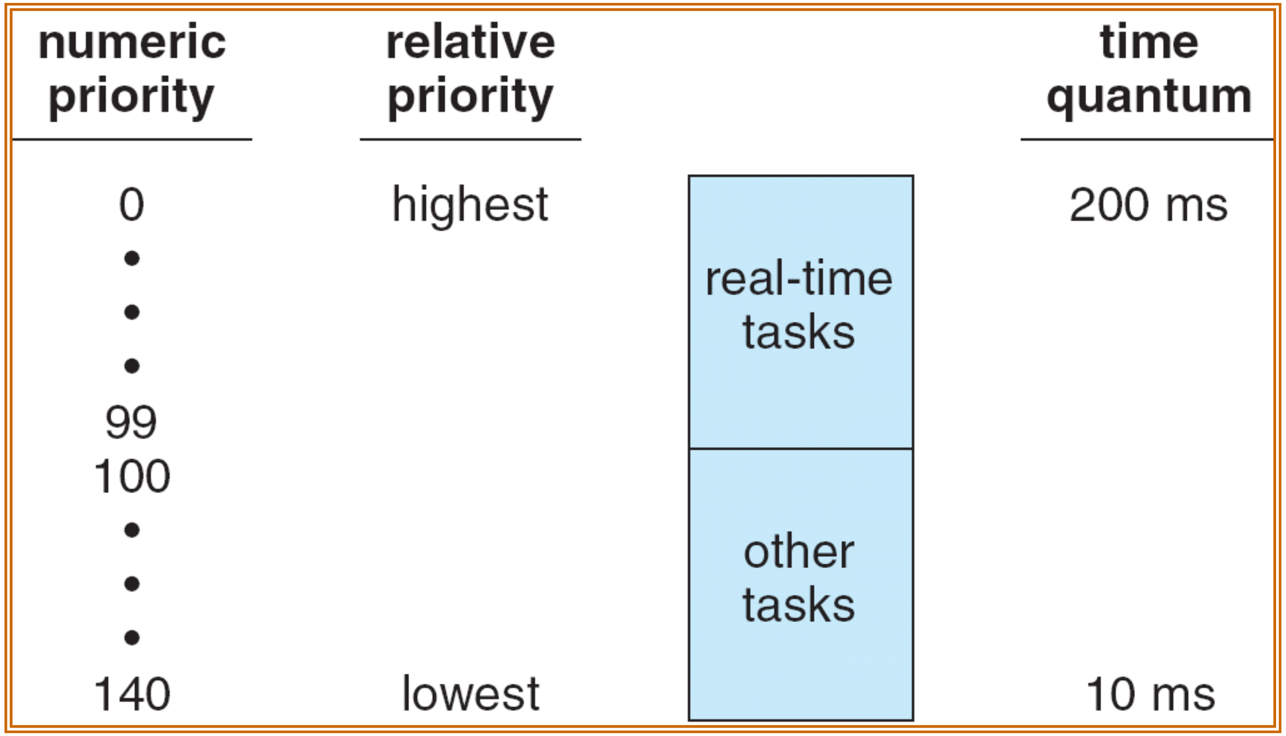
- Two algorithms: time-sharing and real-time

- Time-sharing
  - Prioritized credit-based – process with most credits is scheduled next
  - Credit subtracted when timer interrupt occurs
  - When credit = 0, another process chosen
  - When all runnable processes have credit = 0, recrediting occurs
    - Based on factors including priority and history

Linux Scheduling (Cont.)

- Real-time
  - Posix.1b compliant – two classes
    - ✓FCFS and RR
    - ✓Highest priority process always runs first
  - Soft real-time
- Each CPU has a runqueue made up of 140 priority lists that are serviced in FIFO order. Tasks that are scheduled to execute are added to the end of their respective runqueue's priority list

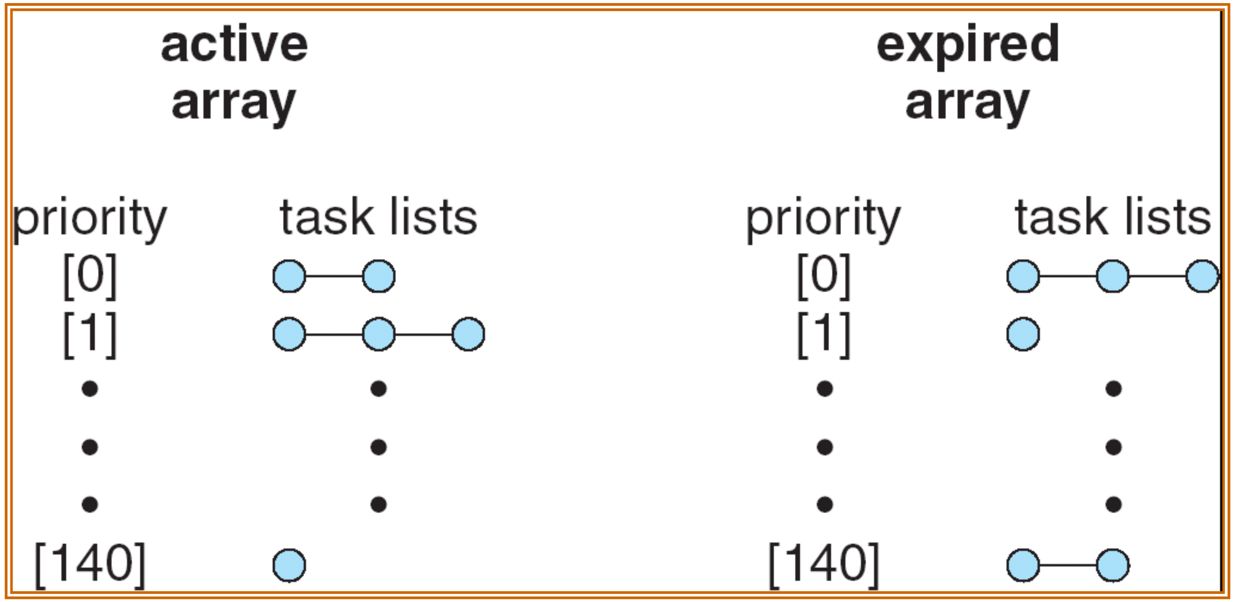
The Relationship Between Priorities and Time-slice length



- The first 100 priority lists of the runqueue are reserved for real-time tasks, and the last 40 are used for user tasks (MAX_RT_PRIO=100 and MAX_PRIO=140)

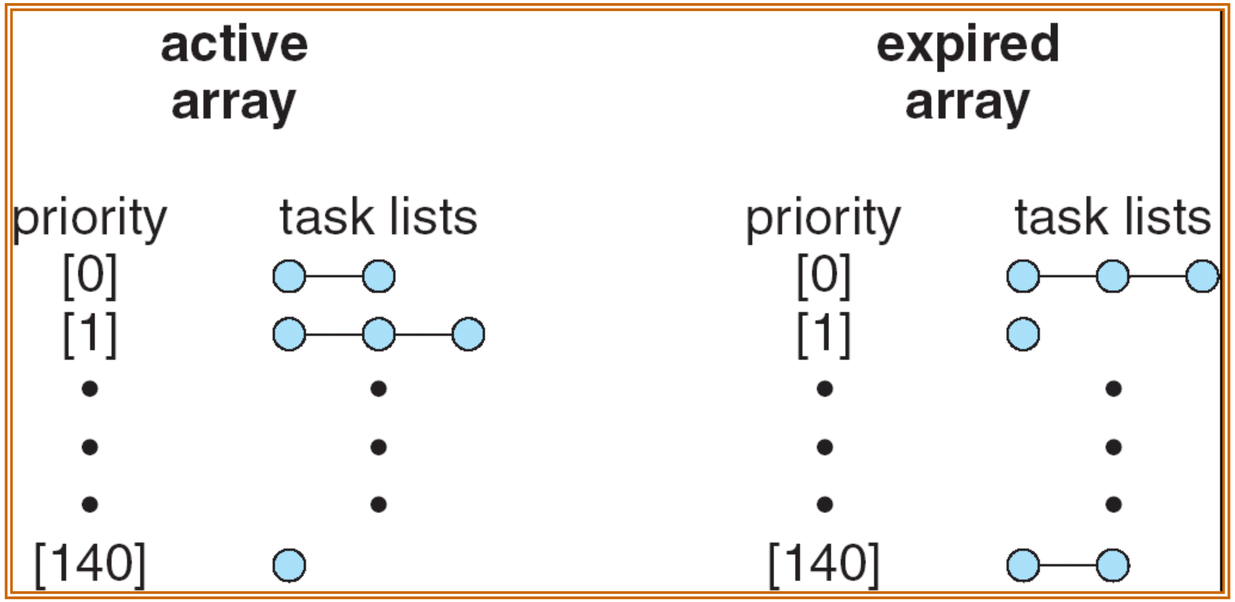
http://www.cs.montana.edu/~chandrima.sarkar/AdvancedOS/CSCI560_Proj_main/index.html

List of Tasks Indexed According to Priorities



- In addition to the CPU's runqueue, which is called the active runqueue, there's also an expired runqueue
- When a task on the active runqueue uses all of its time slice, it's moved to the expired runqueue. During the move, its time slice is recalculated (and so is its priority)

List of Tasks Indexed According to Priorities (cont.)



- If no tasks exist on the active runqueue for a given priority, the pointers for the active and expired runqueues are swapped, thus making the expired priority list the active one

Scheduler Policy

- Each process has an associated scheduling policy and a static scheduling priority
  - SCHED_FIFO - A First-In, First-Out **real-time process**
  - SCHED_RR - A Round Robin **real-time process**
  - SCHED_NORMAL: A conventional, time-shared process (used to be called SCHED_OTHER) for **normal tasks**
  - SCHED_BATCH - for "batch" style execution of processes; for **computing-intensive tasks**
  - SCHED_IDLE - for running very low priority **background job**

http://linux.die.net/man/2/sched_setscheduler