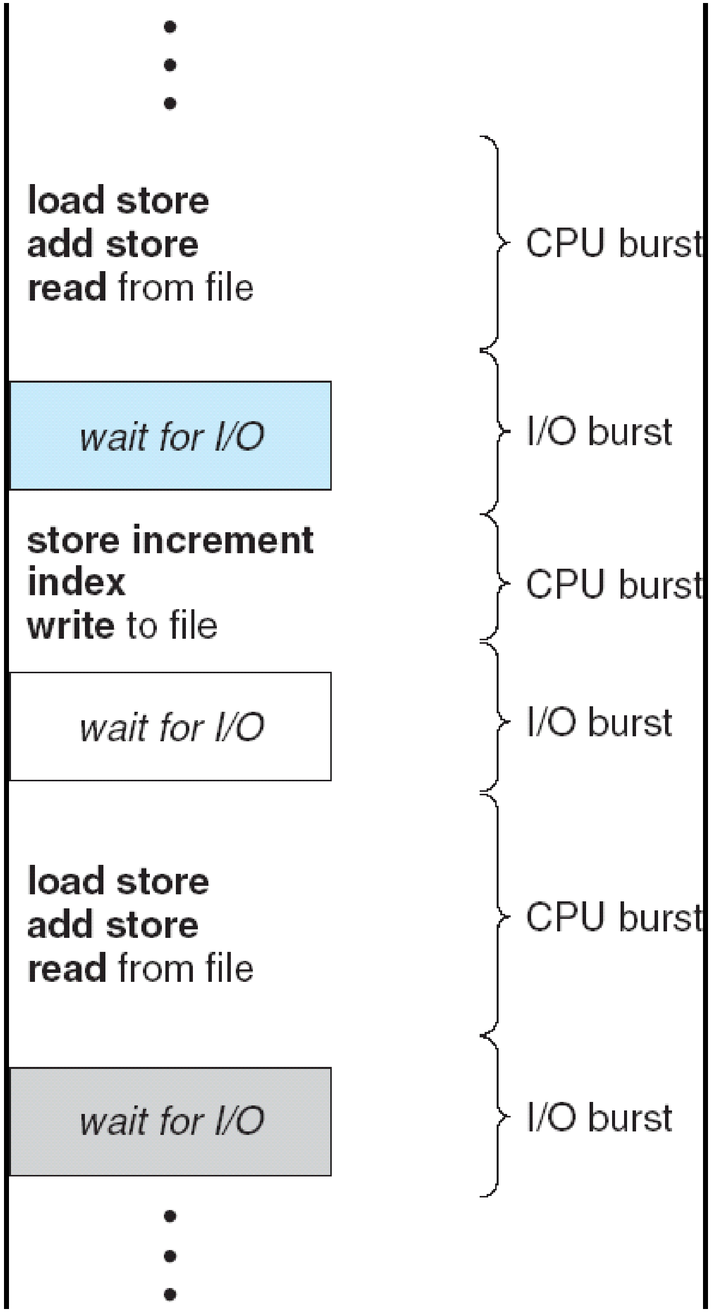
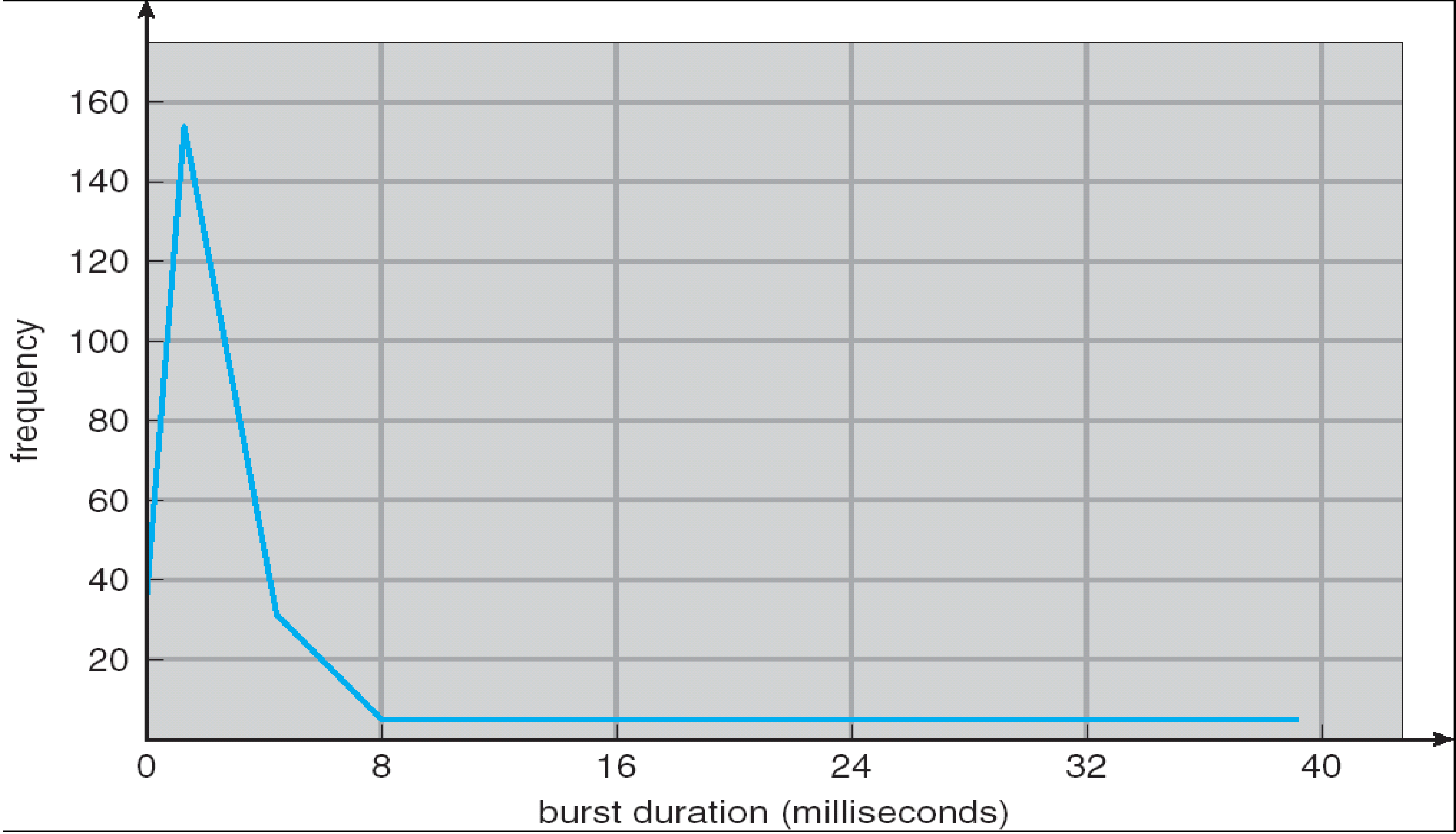
Chapter 5
Scheduling

Images from Silberschatz

CPU usage/IO bursts

- Life time of a single process
- What would an IO bound process look like?
- What would a CPU bound process look like?





- Single process
- What would an IO bound process look like?
- What would a CPU bound process look like?

CPU Scheduler

- Short term scheduler
- Takes process from ready queue and runs it
  - Various algorithms used here.....
  - Data structure? Why?
  - puts it on the CPU
- Takes a process off the CPU and puts it on the ready queue
  - Why?
- Swapping processes around causes a ......

Scheduling events

- Processes moved from the CPU when:
  - Switches from running to waiting state
  - Switches from running to ready state
  - Switches from waiting to ready
  - Terminates

  - What if only first and last are implemented?
    - Why would we ever do this?

Problems

- What happens if a process is preempted while in a system call?
  - Possible bad outcomes?
  - How to fix this?

Dispatcher

- Module/code that puts the process on the CPU
  - Switch context
  - Switch to user mode
  - Restart at correct program counter (PC)

- Dispatch Latency:

Goals

- CPU Utilization
- Throughput
- Turnaround time
- Waiting time
- Response time
- Usually optimize average
  - Sometimes optimize the minimum or maximum
  - Sometimes minimize the *variance*
  - Why? Which values?

Scheduling Algorithms

- First-Come, First-Served (FCFS)
  - Non-preemptive (cooperative!)
  - Data structure?

Process	Burst Time
P_1	24
P_2	3
P_3	3

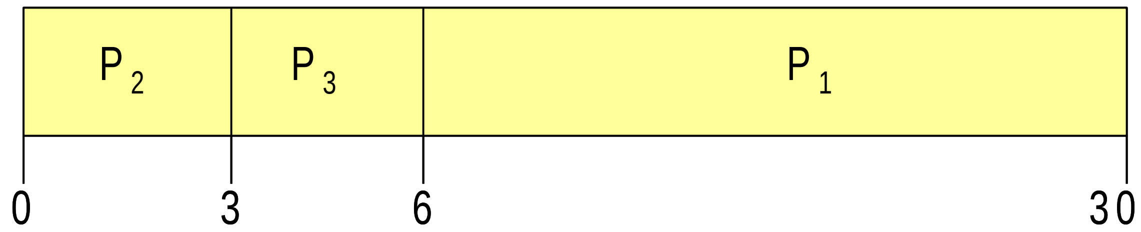
- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:
- Waiting time
- Average waiting time:

FCFS, cont

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:



- Waiting time
- Average waiting time:
- *Convoy effect*

- Advantages?

Shortest Job First (SJR)

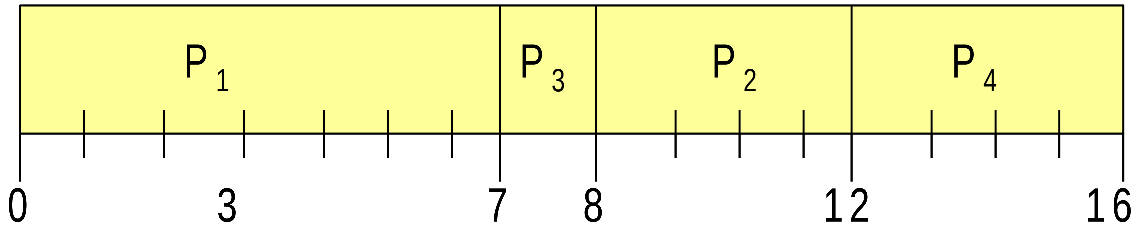
- Choose process who's next CPU *burst* is the shortest
  - Not really shortest JOB first
- May be preemptive (or not)
  - Preemptive (Shortest-Remaining-Time-First (SRTF))

- Gives minimum average waiting time
  - Provably optimal
  - Preemptive
  - With perfect knowledge

Example (cooperative!)

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)

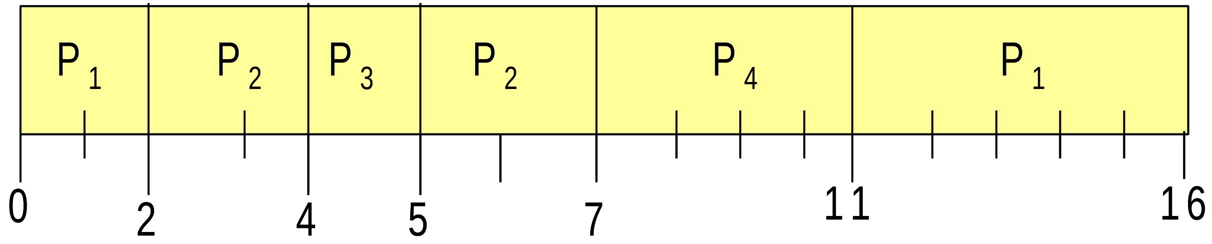


- Average waiting time

Preemptive

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

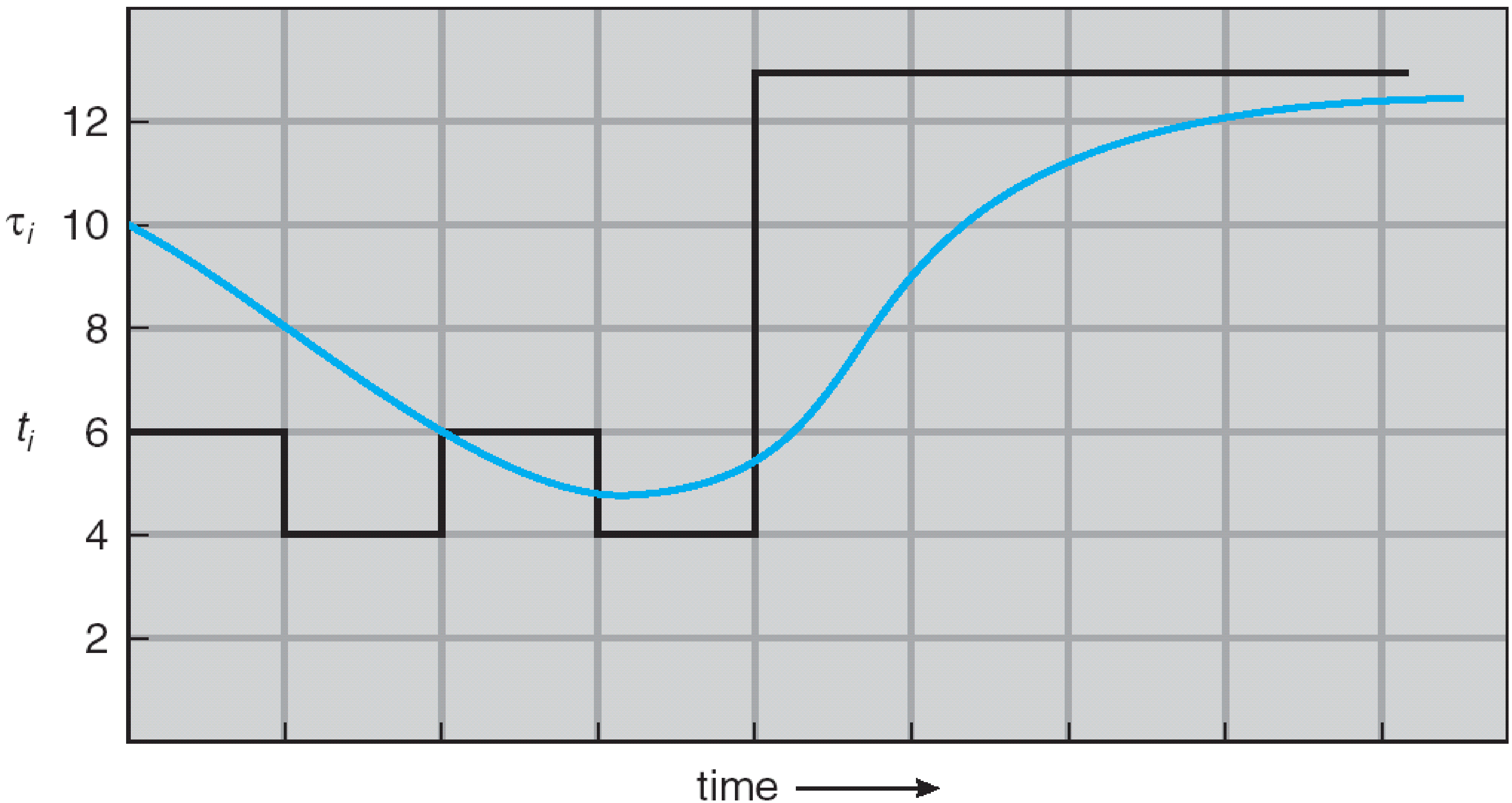
Why is this hard?

- Length of next CPU burst is?

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

- Why? What does this mean? What does this look like?

Prediction of 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	. . .

Priority Scheduling

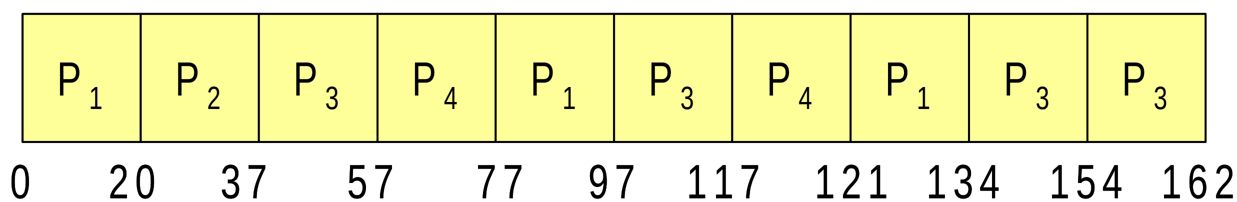
- Give each process a priority (an integer)
- Schedule process with highest priority
  - May be the lowest integer (to make things more confusing)
- Preemptive or not
- SJF is a special case of this
  - What is the priority?
- Where might a problem arise?

Round Robin

- Each process gets some amount of time (10-100 milliseconds)
  - Time quantum/slice
  - Put at the end of the queue when done

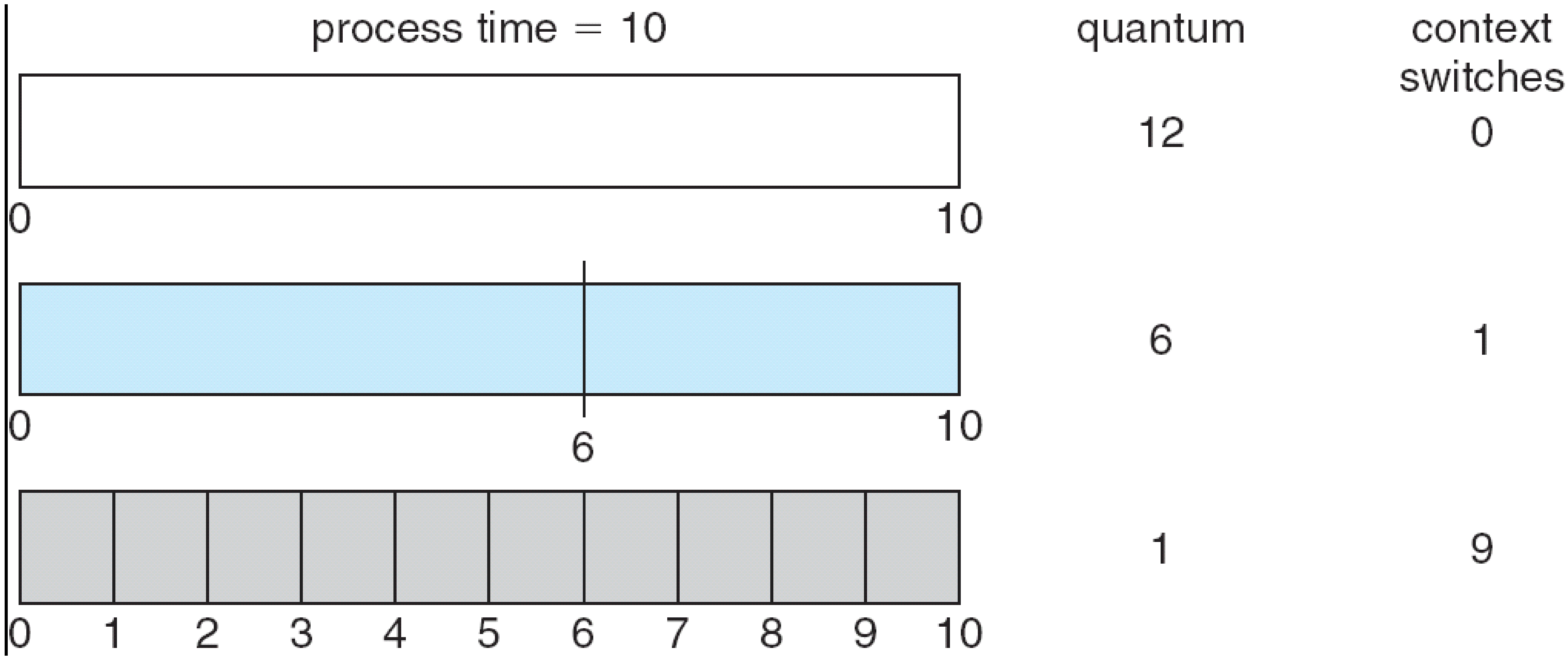
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

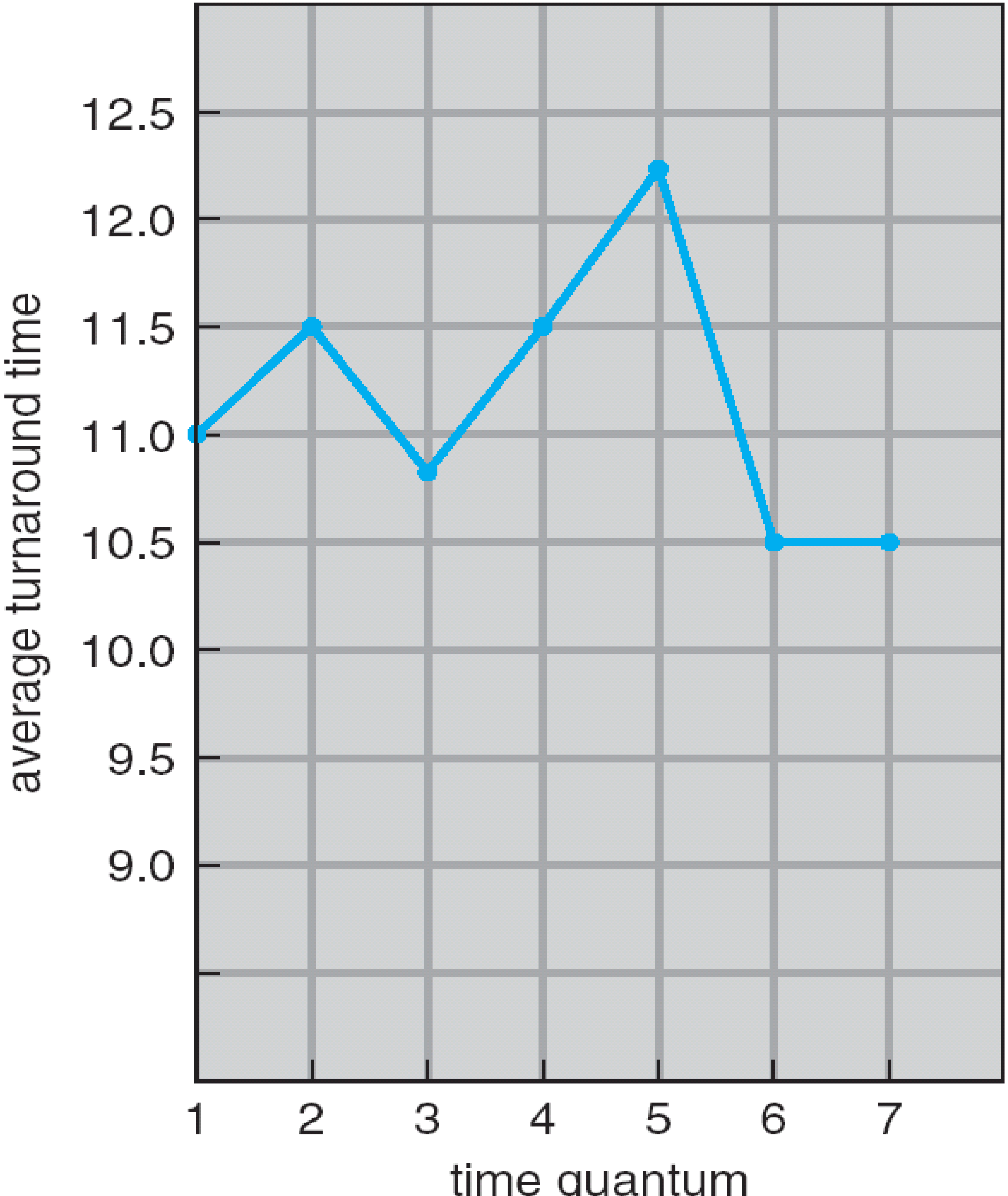


- Typically, higher average turnaround than SJF, but better *response*

Time quanta & context switches



Turnaround time



process	time
P_1	6
P_2	3
P_3	1
P_4	7

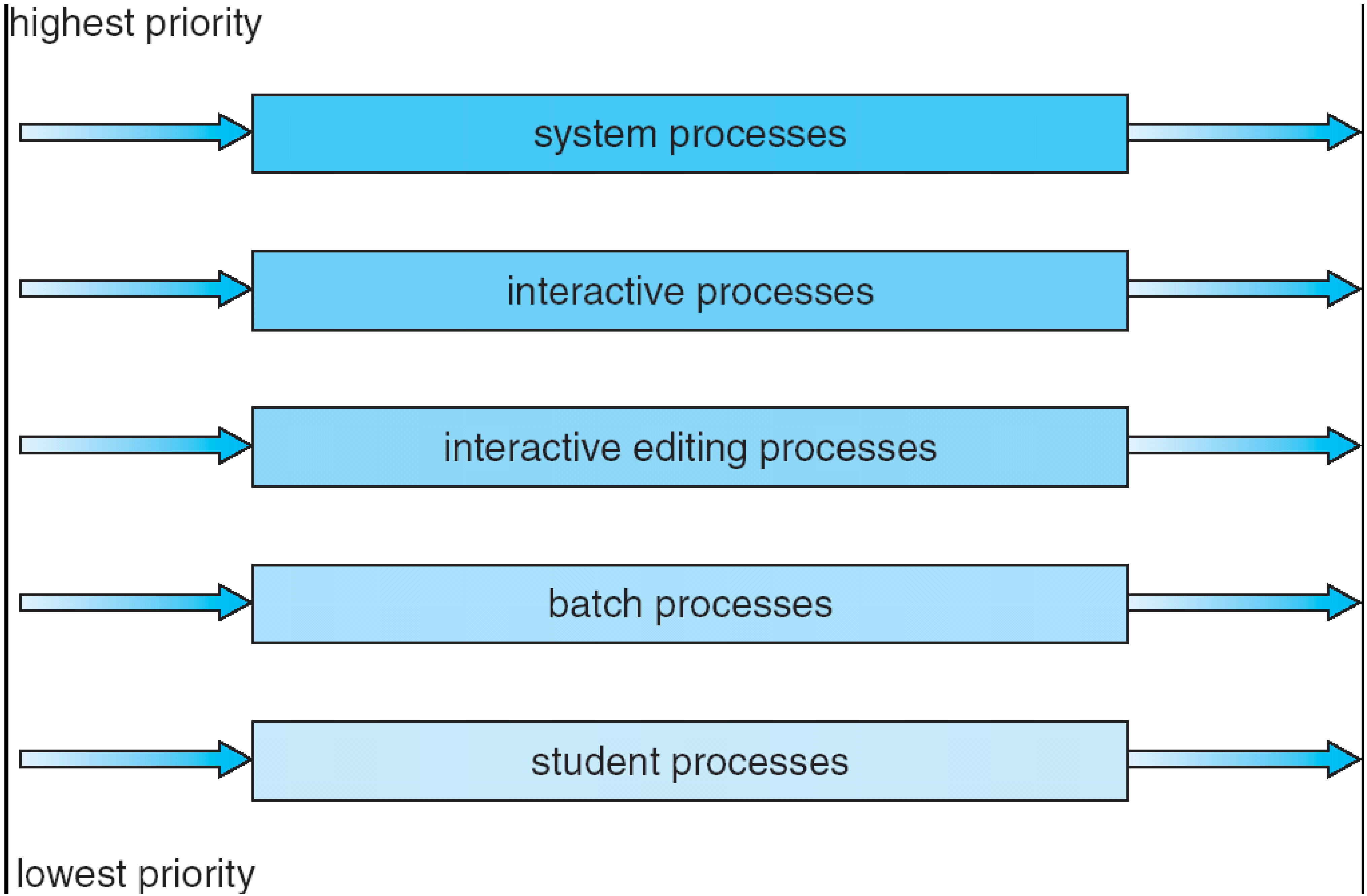
Multilevel Queue Scheduling

- Different Queues, different algorithms
  - Process stays in one queue forever

- Foreground

- Background

- Other categories



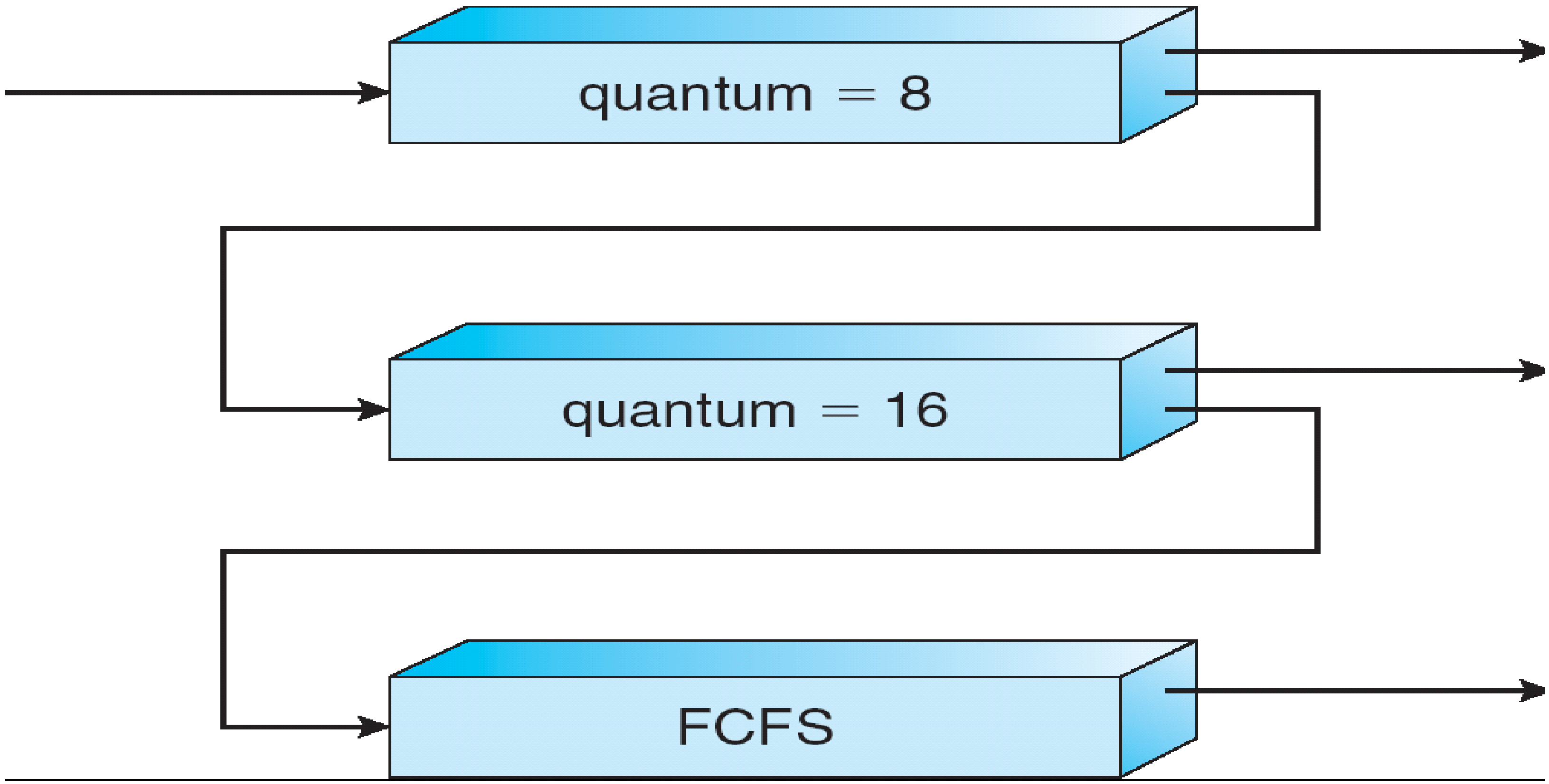
- Absolute vs Time slicing

Multilevel Feedback-Queue Scheduling

- Processes move between queues
  - Use CPU burst information to move processes
  - Aging may play a role
- Defining characteristics
  - 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

Example (p168)

- Three queues
  - Q0 RR with time quantum 8 milliseconds
  - Q1 RR with time quantum 16 milliseconds
  - Q2 FCFS



Multiple-Processor Scheduling

- Asymmetric Multiprocessor

- Symmetric Multiprocessor

- Processor Affinity
  - Soft vs hard

Cont.

- Load Balancing
  - Push migration
  - Pull migration
- Hyperthreading

Thread Scheduling

- Process-contention-scope

- System-contention-scope

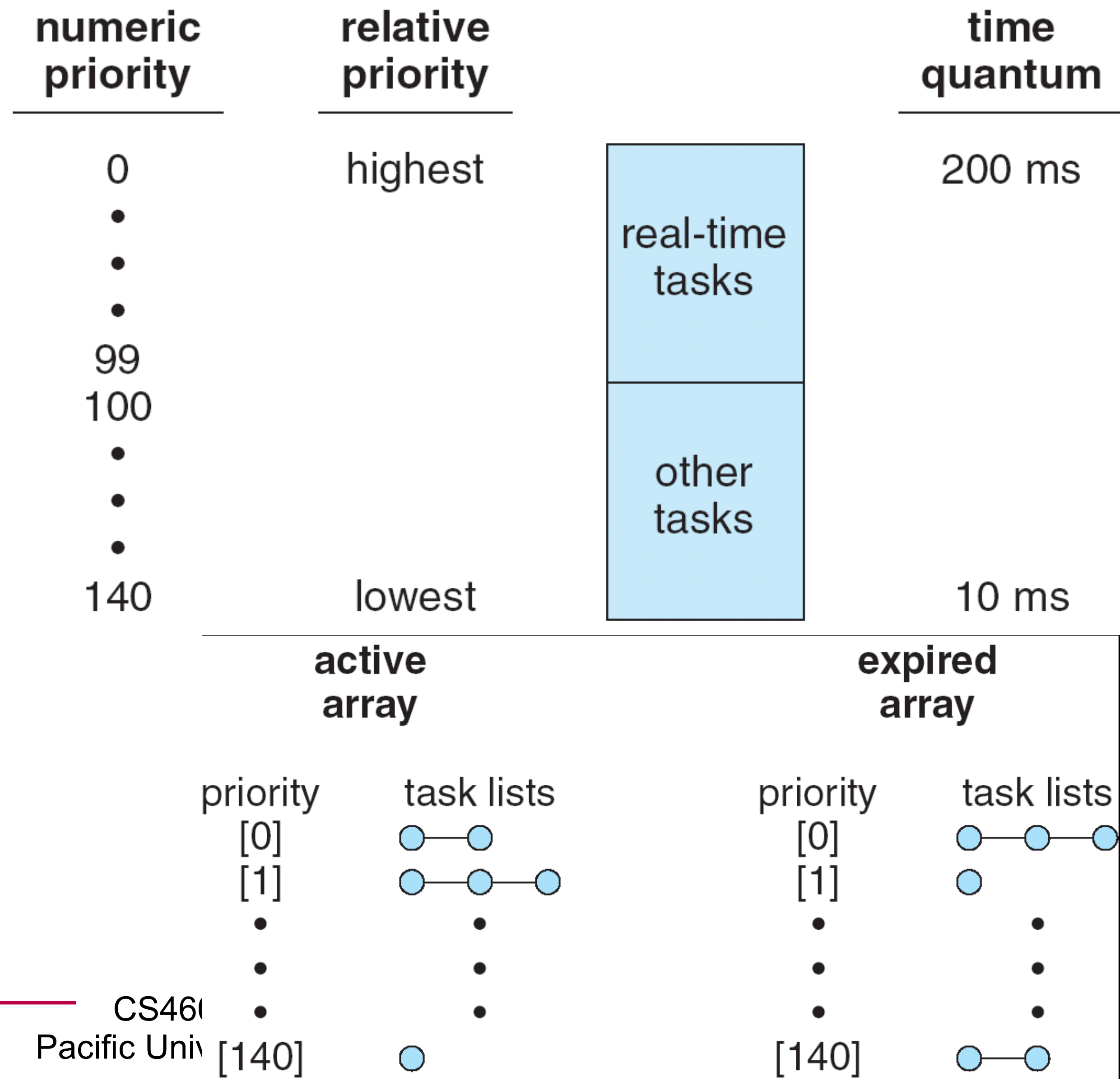
Pthreads

```
#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, RR, 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);
    /* 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);
}
```

Note the coding standards violations!

Linux O(1) Scheduler: no longer used

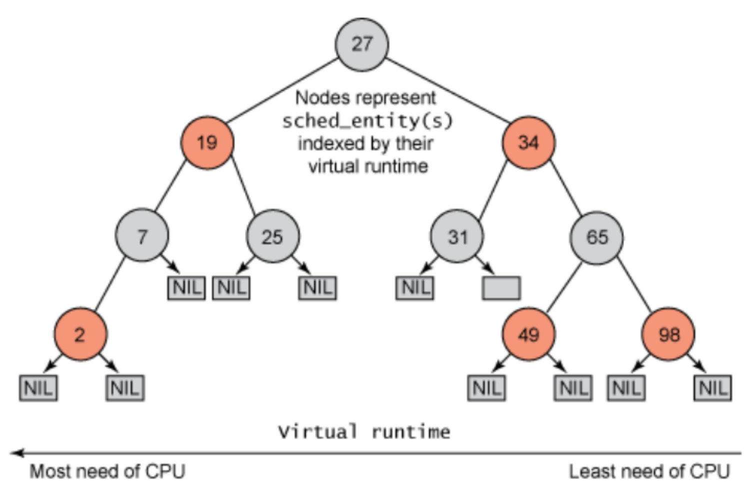
- Preemptive, priority based
- Two priority ranges (lower is better):
  - Real-time: 0-99
  - Nice: 100-140



Linux: Complete Fair Scheduler (CFS)

- Current scheduler kernel/sched/fair.c
  - merged 2.6.23 (October 2007)
  - give each process a fair share of the CPU
  - *virtual runtime* - amount of time provided to a given task (process/thread)
  - priorities cause the virtual runtime to increase more quickly or more slowly
  - each process has its own timeslice metric.
  - **Red-Black tree** is used to hold tasks (indexed by virtual runtime)
  - always choose the left most node to run
  - insert a task into the right and slowly migrate to the left.

Figure 1. Example of a red-black tree



http://www.diit.unict.it/users/llobello/linuxscheduling20122013_P2.pdf
https://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/l-completely-fair-scheduler-pdf.pdf

Algorithm Evaluation

- How to choose a scheduling algorithm?
  - Define goals
    - Minimize wait time? Minimize response time? Maximize CPU utilization?
- Deterministic modeling
- Queuing modeling (queuing network analysis)
  - Little's formula
- Simulations
- Build it