Scheduling

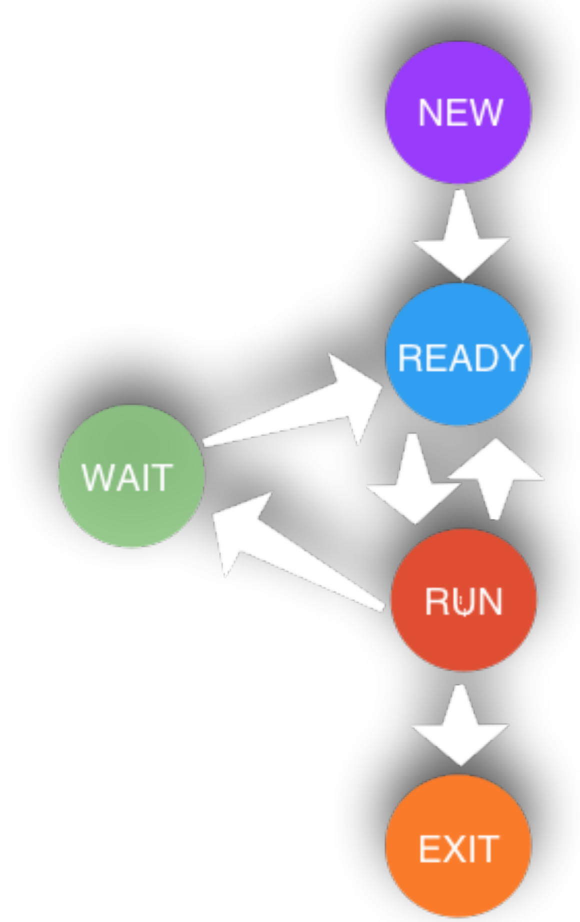
Chapter 2: Section 2.4

Process State Diagram

Transitioning to Run

How do processes transition from Ready to Run?

What decides when it should run in relation to other processes that are ready?



Scheduling

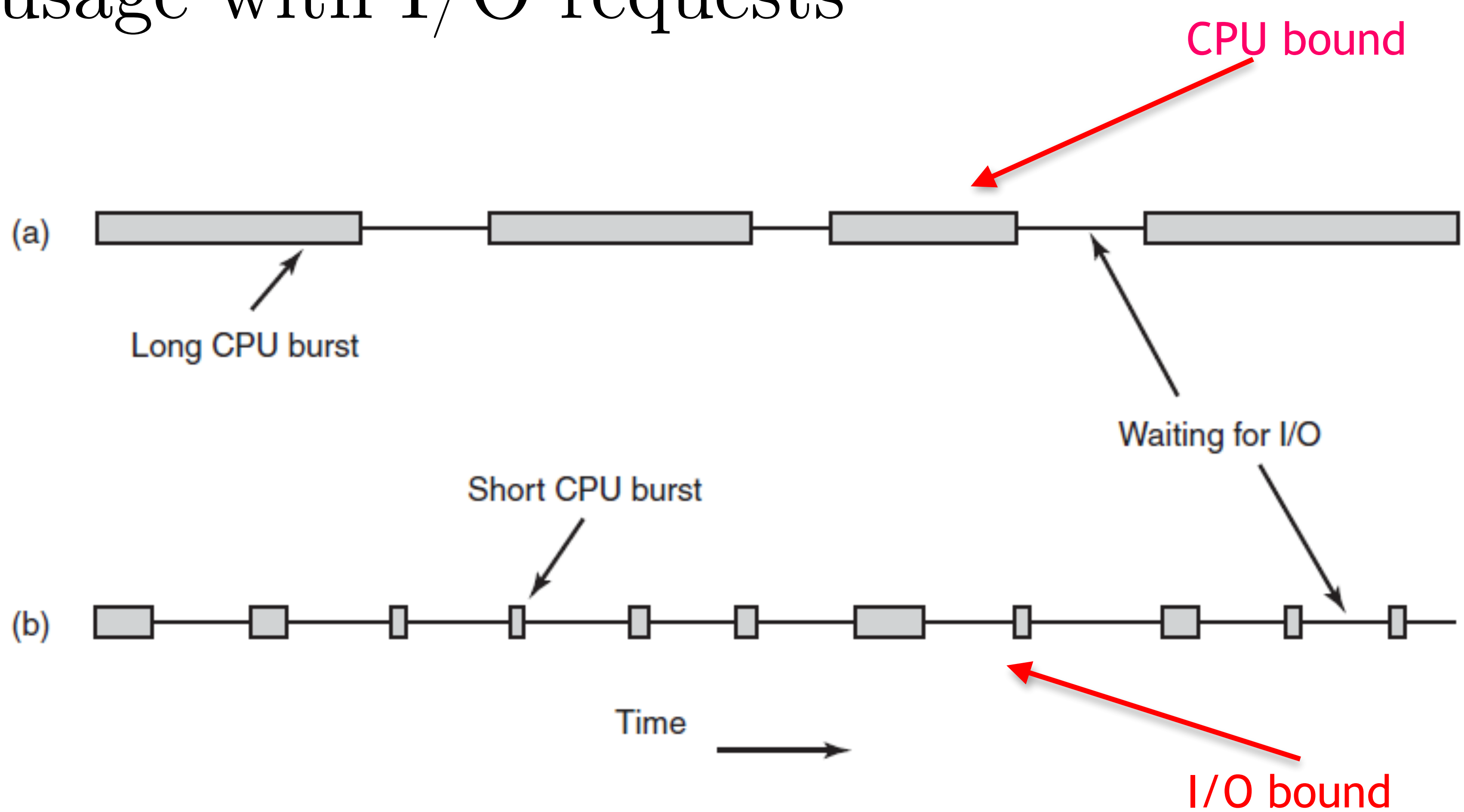
- Cooperative Multitasking
- Preemptive Multitasking

Scheduling

- Multiple processes or threads competing for the CPU
  - Choice needs to be made of which to run next
    - Scheduler - part of the OS that makes that decision
    - Scheduling algorithm

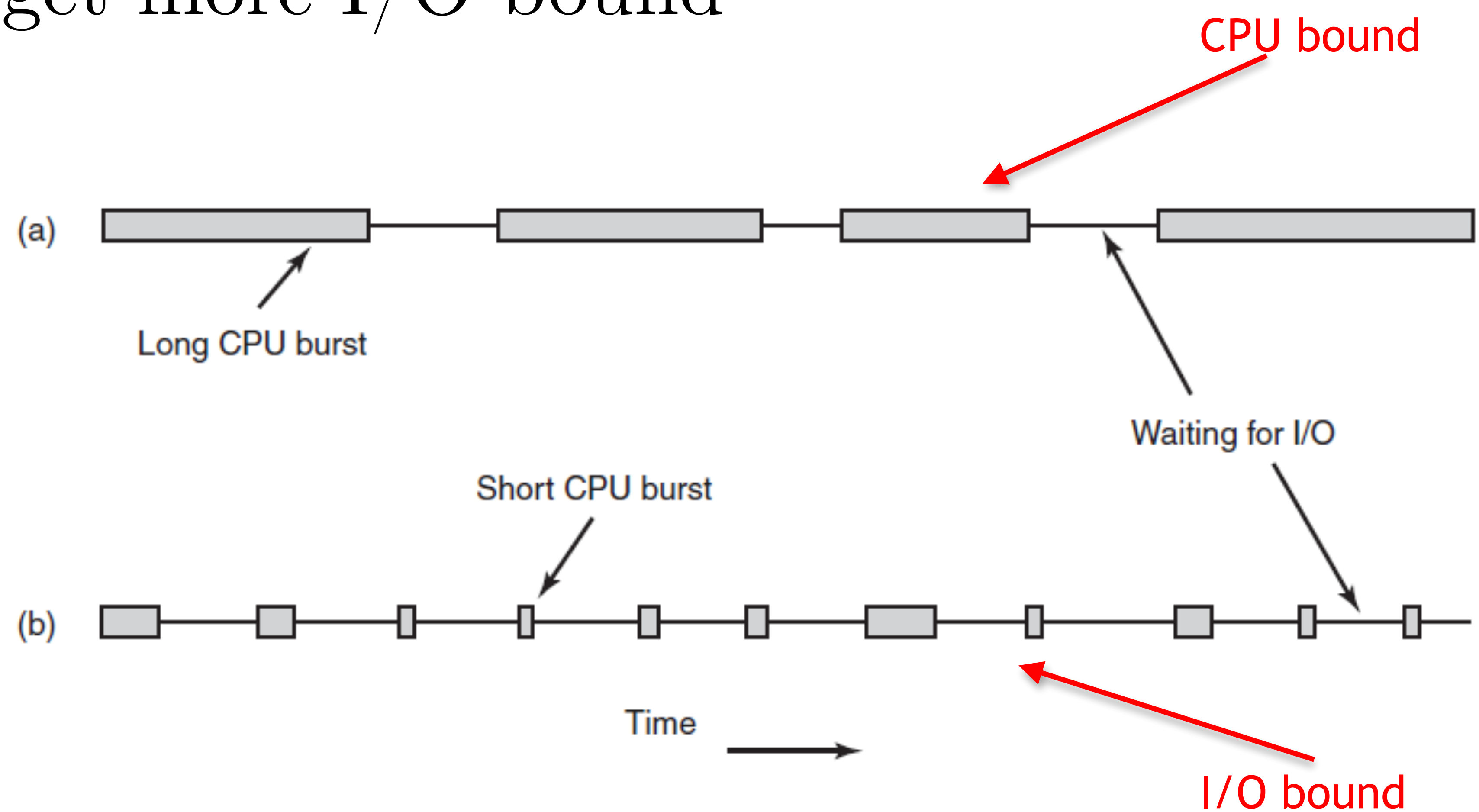
Scheduling

- Nearly all processes alternate burst of CPU usage with I/O requests



Scheduling

- As processors get faster, processes tend to get more I/O bound



Scheduling

- Key issue: when to make the scheduling decision

3. When a process blocks on I/O, semaphore, mutex, etc.
   - Scheduler doesn't know process dependencies
4. When an I/O interrupt occurs

Scheduling

- I/O interrupt
  - Hardware clock provides 50 or 60 hz interrupts
- Scheduling algorithms can be divided into 2 categories based on how they deal with clock interrupts
  - Non-preemptive (cooperative)
  - Preemptive

Categories of Scheduling

1. Batch
   - Still in widespread use
   - Nonpreemptive or preemptive with long time quantum ok
2. Interactive
   - Preemptive to provide responsiveness
   - General purpose
3. Real Time
   - Preemptive not always needed

Scheduling Algorithm Goals

- All Systems
  - Fairness - give each process a fair share
  - Policy enforcement - see the stated policy is carried out
  - Balance - keep all parts of the system busy

Scheduling Algorithm Goals

- Batch Systems
  - Throughput - maximize jobs per hour
  - Turnaround time - minimize time between submission and termination
  - CPU utilization - you paid for that screaming CPU, use it

Scheduling Algorithm Goals

- Interactive systems
  - Response time
  - Proportionality - meets users expected performance

Scheduling Algorithm Goals

- Real-time systems
  - Meet deadlines
  - Predictability - avoid quality degradation in multimedia systems

Scheduling Metrics

- Throughput - number jobs per unit of time
- Turnaround time - average of time when jobs are submitted to when they complete
- Response time - average of time when jobs are submitted to jobs start running
- Wait time - time jobs spend in the wait queue

Selecting the best algorithm

- Criteria:
  1. throughput - jobs run per hour or per minute
  2. average turnaround time - time from start to end of job
  3. average response time - time from submission to start of output
  4. CPU utilization - percent of time the CPU is running real jobs (not switching between processes or doing other overhead; of more interest in large systems)
  5. average wait time - the time that processes spend in the ready queue

Selecting the best algorithm

- Criteria:
  1. throughput - jobs run per hour or per minute
  2. average turnaround time - time from start to end of job
  3. average response time - time from submission to start of output

Depends on job mix, so difficult to compare fairly and accurately

Selecting the best algorithm

- Criteria:
    4. CPU utilization - percent of time the CPU is running real jobs (not switching between processes or doing other overhead; of more interest in large systems)

Interesting and easy to measure, but in personal computers we really don’t care about it.

Selecting the best algorithm

- Criteria:
  5. average wait time - the time that processes spend in the ready queue

Makes the most sense. We want to make sure the that most computing is getting done in the least amount of wasted time

FCFS Scheduling

- First come, first served algorithm (FCFS).
- Easy to implement
- Well understood by anyone
- The fairest algorithm. No process is favored over another.

FCFS

Process ID	Arrival Time	Runtime
1	0	20
2	2	2
3	2	2

Average Waiting Time: (0 + 18 + 20) / 3 = 12.67

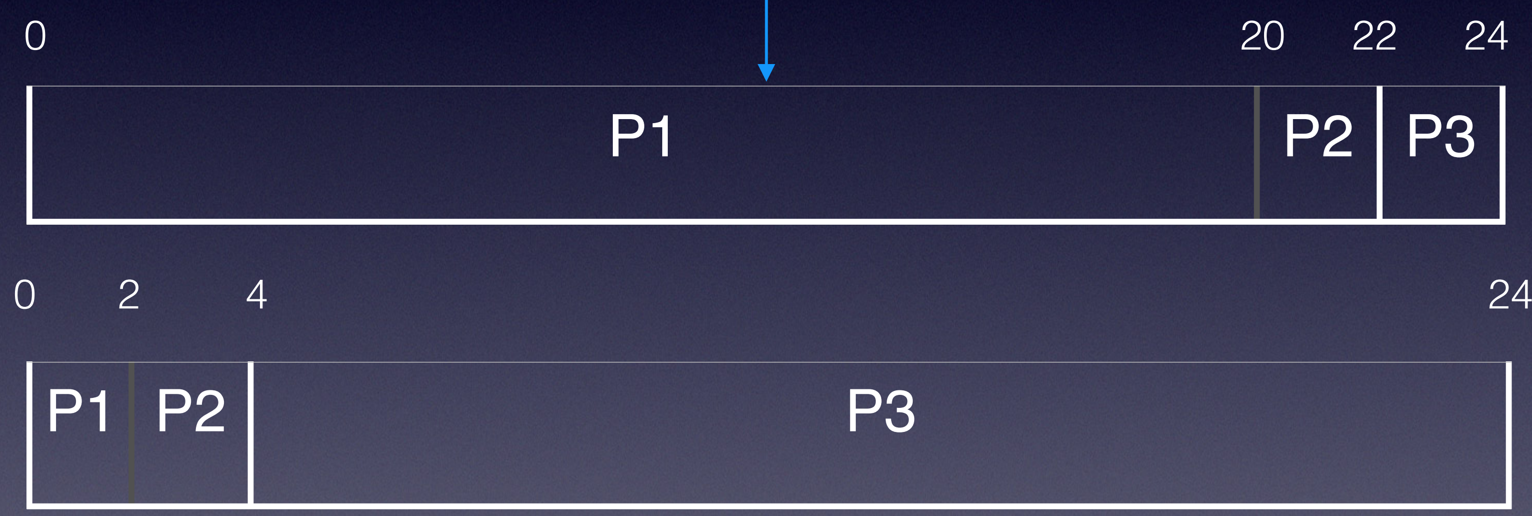
FCFS

Process ID	Arrival Time	Runtime
1	0	2
2	2	2
3	2	20

Average Waiting Time: (0 + 0 + 2) / 3 = 0.67

FCFS

Convoy Effect



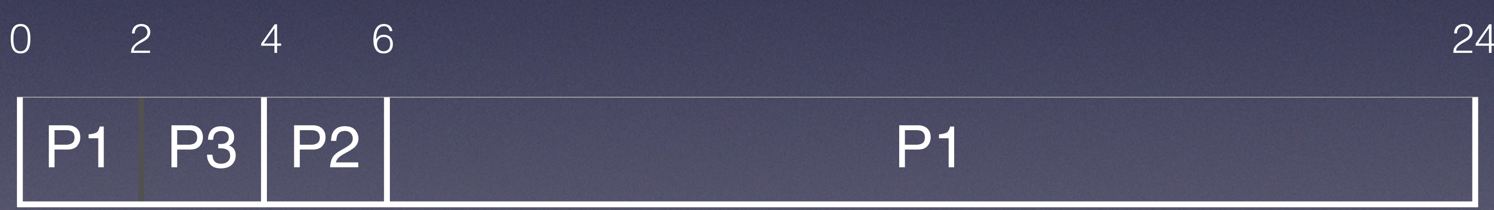
Very Dependent On Job Arrival Time

Preemption

Without Preemption	With Preemption
FCFS	Round Robin
SRTF	Shortest Remaining Time Next
Priority	Priority With Preemption

FCFS w/ Priority (Round Robin)

Process ID	Arrival Time	Runtime	Priority
1	0	20	4
2	2	2	2
3	2	2	1



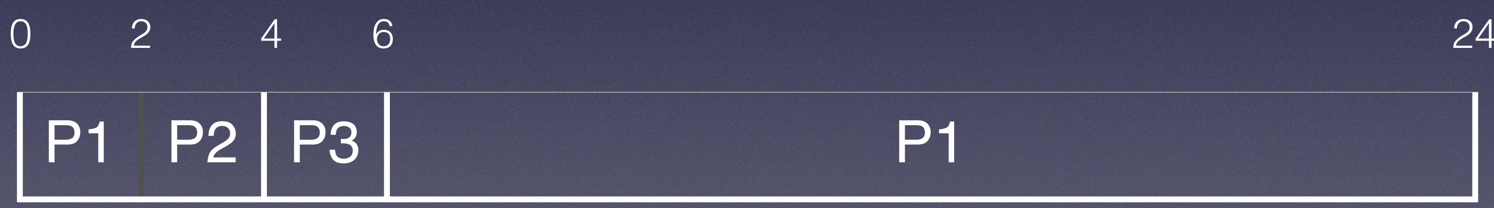
Average Waiting Time: (4 + 2 + 0) / 3 = 2

Time Quantum

- Choosing length is a problem
  - Context switching is expensive
  - Still need responsiveness

SJN with Preemption

Process ID	Arrival Time	Runtime
1	0	20
2	2	2
3	2	2



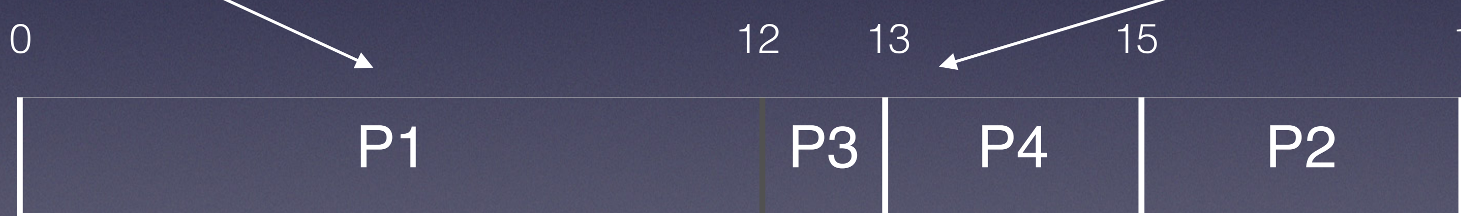
Average Waiting Time: (4 + 0 + 2) / 3 = 2

SJN

Process ID	Arrival Time	Runtime
1	0	12
2	2	4
3	3	1
4	4	2

P1 runs first since it's the only process

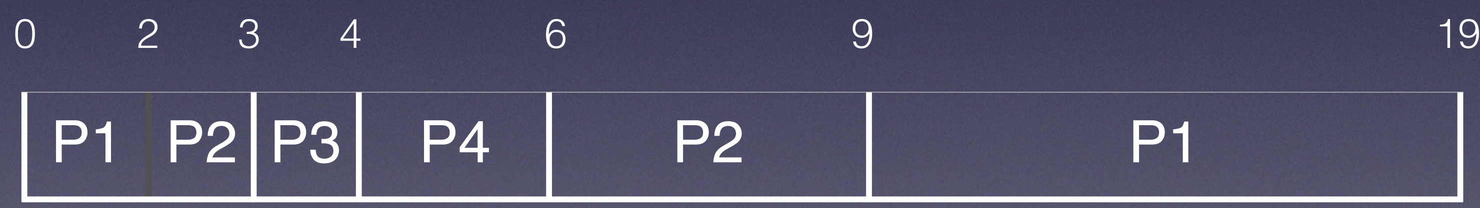
No preemption so P3, P4, and P2 wait. Then sorted by least runtime



Average Waiting Time: (0+13+9+9)/4 = 7.75

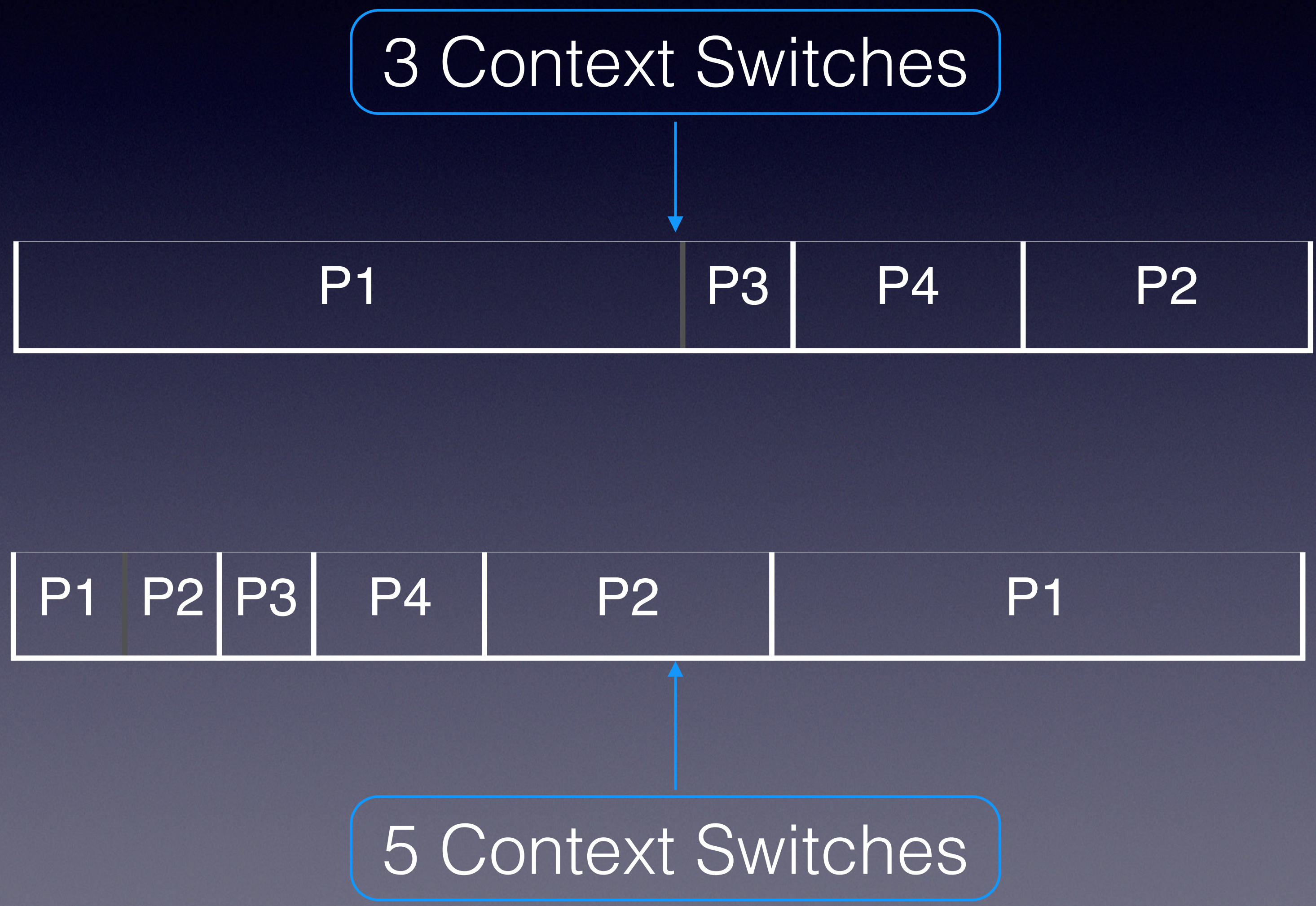
SJN With Preemption

Process ID	Arrival Time	Runtime
1	0	12
2	2	4
3	3	1
4	4	2



Average Waiting Time: (7+3+0+0)/4 = 2.5

SRTF Without and With Preemption



Priority Scheduling

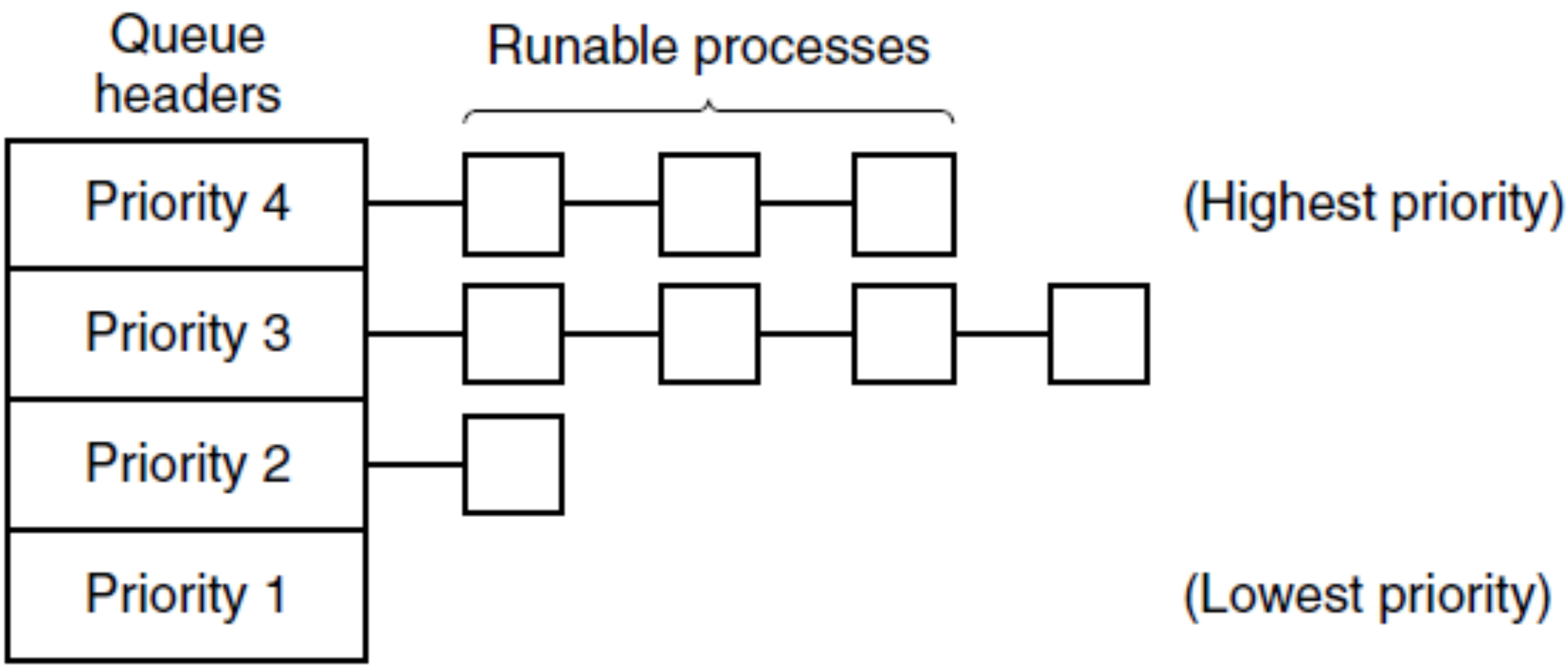


Figure 2-43. A scheduling algorithm with four priority classes.

Guaranteed Scheduling

If n users are logged in, you will receive 1/n of the CPU power.

Lottery Scheduling

- Process receives a lottery ticket
- Lottery ticket chosen at random and the winning process is allowed to run.
- More important processes can be given more lottery tickets.
- Highly responsive.
- Tickets can be exchanged by cooperating processes

Lottery Scheduling

- Lottery scheduling can solve problems difficult to handle by other schedulers.
- Video server with video streams of different frame rates. (10, 20, 25 frames per second)
  - Processes get 10, 20, 25 tickets

Fair Share Scheduling

- So far assumed each process is scheduled on its own.
  - User 1 has 9 processes. User 2 has 1 process. User 1 gets 90% of the CPU.
- Instead, each user gets a fair share of the CPU usage.