Modern Operating Systems

Chapter 2 – Scheduling

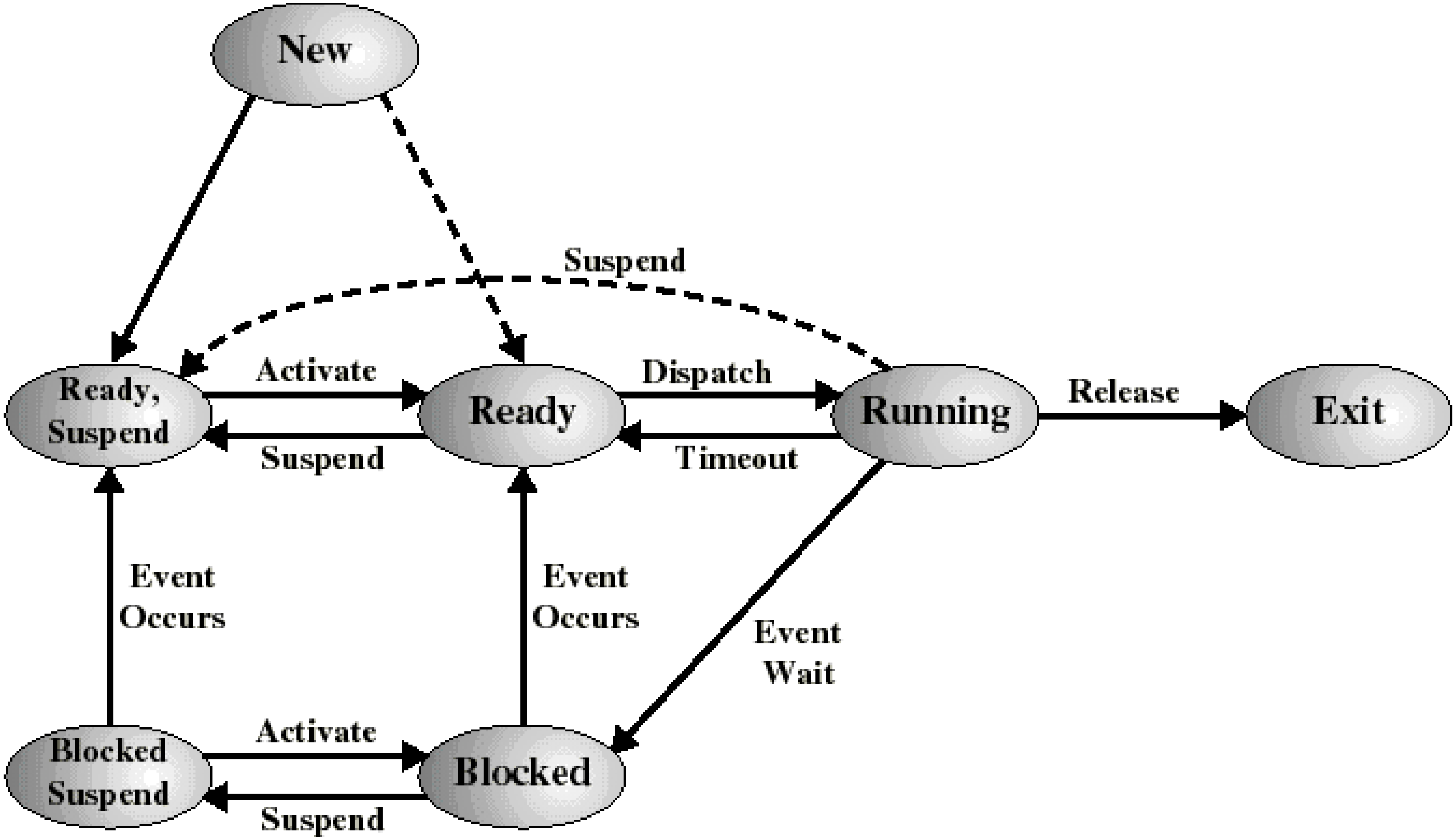
Zhang Yang

Spring 2022

Content of this lecture

- 2.4 Scheduling
    - A Seven-State Process Model
    - Classification of Scheduling Activity
    - What is Scheduling?
    - Process Behavior
    - When to Schedule?
    - Scheduling Algorithms
    - Summary

A Seven-State Process Model (1)



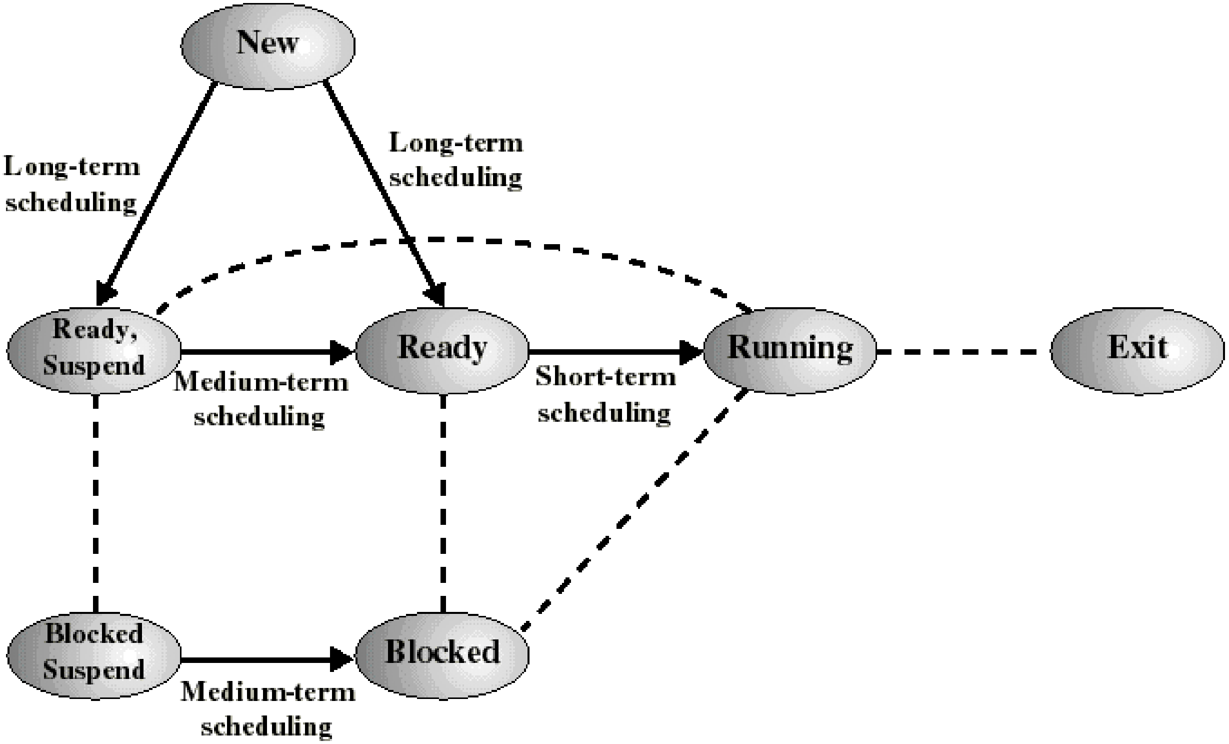
A Seven-State Process Model (2)

- New
  - A process that has been created, but not yet accepted in the pool of executable processes by OS.
  - Typically, a new process has not yet been loaded into main memory.
- Ready
  - The process is in main memory and available for execution.
- Blocked
  - The process is in main memory and awaiting an event.

A Seven-State Process Model (3)

- Blocked, Suspend
  - The process is in secondary memory and awaiting an event.
- Ready, Suspend
  - The process is in secondary memory but is available for execution as soon as it is loaded into main memory.
- Exit
  - A process that has been released from the pool of executable processes by the OS.
  - Completed or due to some errors.

Classification of Scheduling Activity



- Long-term: which process to admit.
- Medium-term: which process to swap in or out.
- Short-term: which ready process to execute next.

Long-Term Scheduling

- Determines which programs are admitted to the system for processing.
- Controls the degree of multiprogramming.
- If more processes are admitted
  - Less likely that all processes will be blocked.
  - Better CPU usage.
  - Each process has less fraction of the CPU.
- The long term scheduler will attempt to keep a mix of processor-bound and I/O-bound processes.

Medium-Term Scheduling

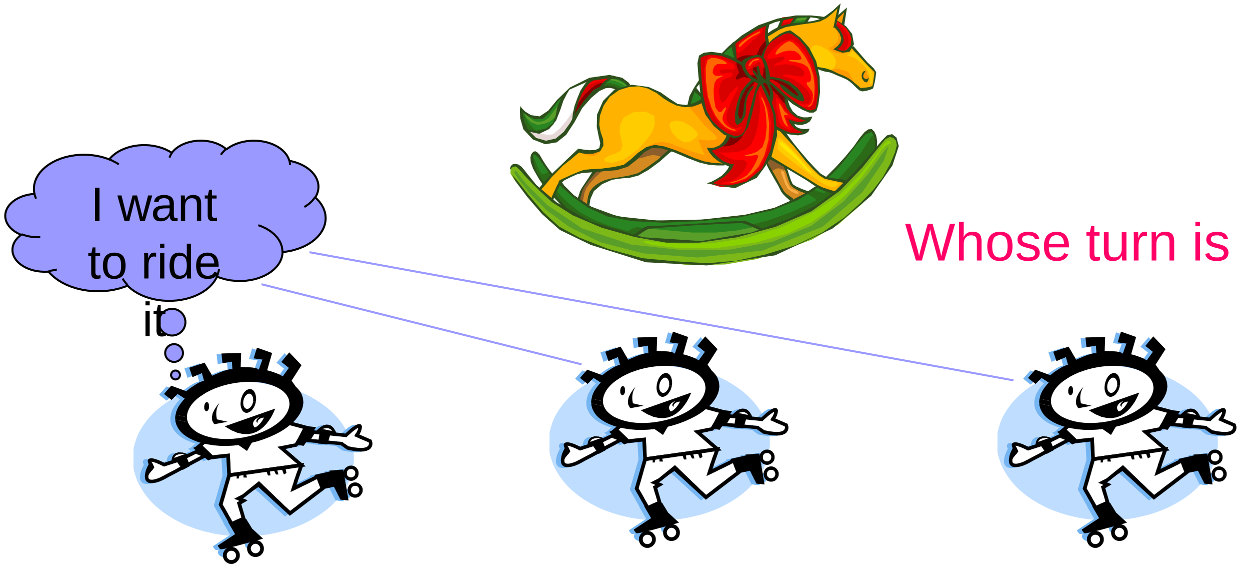
- Swapping decisions based on the need to manage multiprogramming.
- Done by memory management software.
  - See resident set allocation and load control. (in Chapter 3)

Short-Term Scheduling

- Determines which process will execute next (also called CPU scheduling).
- The short term scheduler is known as the dispatcher.
- Is invoked on a event that may lead to choose another process for execution:
  - Clock interrupts
  - I/O interrupts
  - Operating system calls and traps
  - Signals

What is Scheduling?

- Deciding which process/thread should occupy the resource (CPU, disk, etc)



(CPU (horsepower))

I want to ride it

Whose turn is it?

Process 1

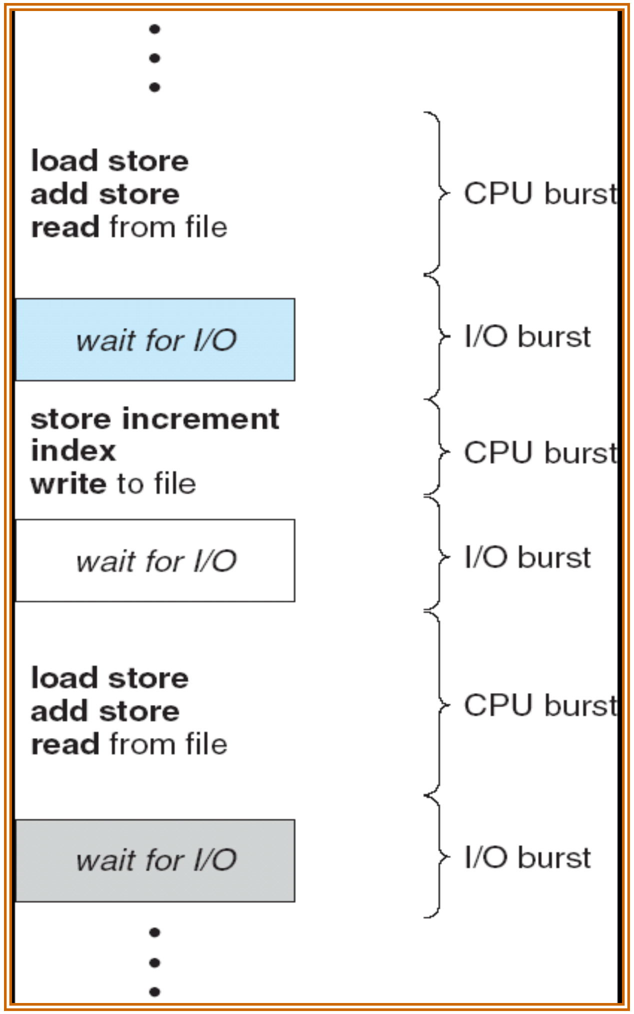
Process 2

Process 3

Process Behavior (1)

- Processes typically consist of
  - CPU Bursts
    - A period of time when a process needs the CPU is called a CPU burst.
  - I/O Bursts
    - A period of time when a process needs I/O is called an I/O burst.

Process Behavior (2)



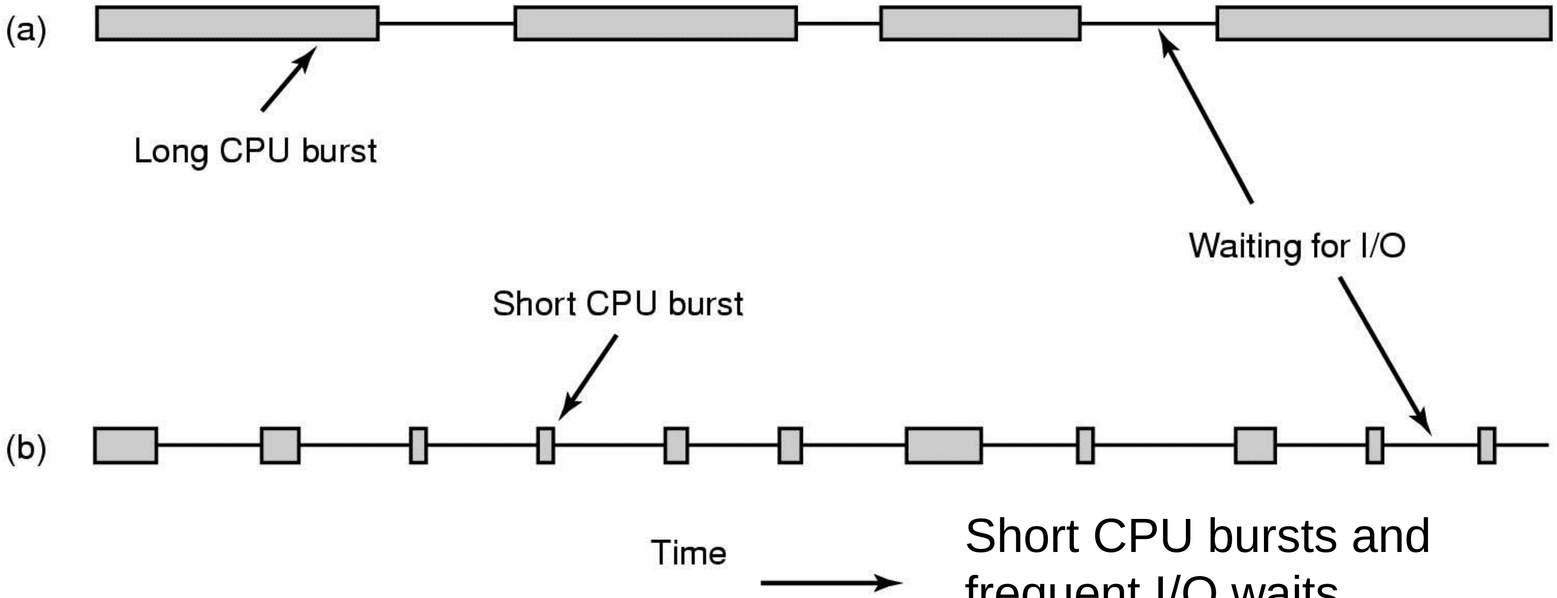
Alternating Sequence of CPU And I/O Bursts

Process Behavior (3)

- Duration and frequency of bursts vary greatly from process to process.
  - CPU-bound Process
    - Long CPU bursts, infrequent I/O waits.
    - E.g. Number crunching tasks, image processing
  - I/O-bound Process
    - Short CPU bursts, frequent I/O waits.
    - E.g. A task that processes data from disk, for example, counting the number of lines in a file is likely to be I/O bound.
- As CPU get faster, processes tend to get more I/O bound.

Process Behavior (4)

Long CPU bursts and infrequent I/O waits



Short CPU bursts and frequent I/O waits

Figure 2-38 Bursts of CPU usage alternate with periods of I/O wait

- (a) a CPU-bound process
- (b) an I/O bound process

CPU Scheduling

- Problem to Solve
  - □When (scheduling opportunity)

    When to allocate CPU to process ?
  - □What (scheduling algorithm)

    What is the principle of allocation ?
  - □How (context - switch)

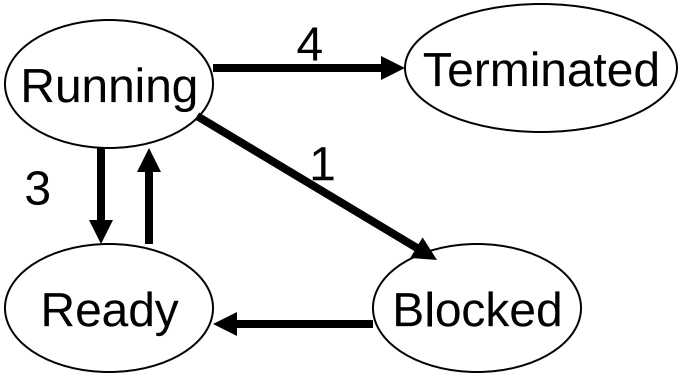
    How to allocate CPU?

When to Schedule?

- A new process is created.
- The running process exits.
- The running process is blocked.
- I/O interrupt (some processes will be ready) or Clock interrupt (e.g. every 10 milliseconds).

Preemptive vs. Non-preemptive

- Preemptive Scheduling
  - The running process can be interrupted and must release the CPU (can be forced to give up CPU)
- Non-preemptive Scheduling
  - The running process keeps the CPU until it voluntarily gives up the CPU
    - Process exits.
    - Switches to blocked state
    - 1and 4 only (no 3)



Categories of Scheduling Algorithms

- Batch System
  - Non-preemptive algorithms
  - Preemptive algorithms with long time periods for each process.
- Interactive System
  - Preemption is essential.
- Real-Time System
  - Preemption is sometimes not needed.

Scheduling Algorithm Goals (1)

- All Systems
  - Fairness
    - No process should suffer starvation.
  - Policy Enforcement
    - Seeing that stated policy is carried out.
  - Efficiency
    - Keep resources as busy as possible.

Scheduling Algorithm Goals (2)

- Batch Systems
  - Throughput
    - Number of jobs processed per unit of time.
  - Turnaround Time
    - Time from submission to completion (batch jobs).
  - Processor Utilization
    - Percent of time CPU is busy.

Scheduling Algorithm Goals(3)

- Interactive Systems
  - Response Time
    - Amount of time from when a request was first submitted until first response is produced.
  - Proportionality
    - Meet users' expectation.
- Real-Time Systems
  - Deadline
    - Meet all deadlines.
  - Predictability
    - Same time/cost regardless of load on the system.

Single Processor Scheduling Algorithms

- Batch Systems
  - First-Come First-Served (FCFS)
  - Short Job First
- Interactive Systems
  - Round Robin
  - Priority Scheduling
  - Multi-level Queue & Multi-level Feedback Queue
  - Shortest Process Next
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair Sharing Scheduling

First-Come, First-Served (1)

- Process that requests the CPU FIRST is allocated the CPU FIRST.
- Also called FIFO
- Non-preemptive
- Used in Batch Systems.
- Real life analogy: Fast food restaurant

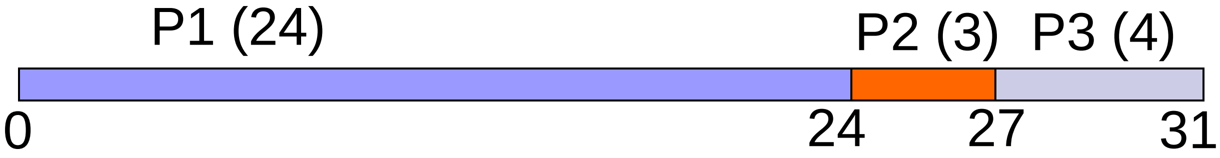
First-Come, First-Served (2)

- Implementation: FIFO queues
  - A new process enters the tail of the queue.
  - An unblocked process re-enters at the tail of the queue
  - The scheduler selects from the head of the queue.
- Performance Metric: Average Waiting Time
- Waiting Time: The total amount of time that a process is in the ready queue.

FCFS Example (1)

Process	Duration	Order	Arrival Time
P1	24	1	0
P2	3	2	0
P3	4	3	0

The final schedule:



P1 waiting time: 0

P2 waiting time: 24

P3 waiting time: 27

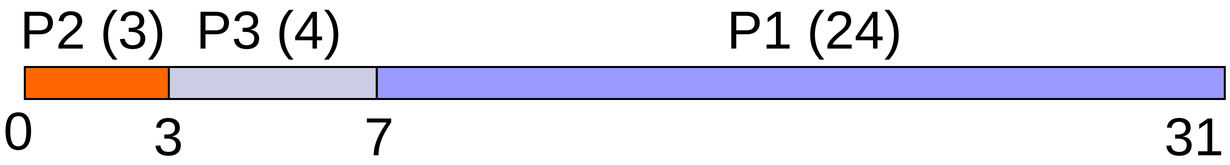
The average waiting time:
(0+24+27)/3 = 17

FCFS Example (2)

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:



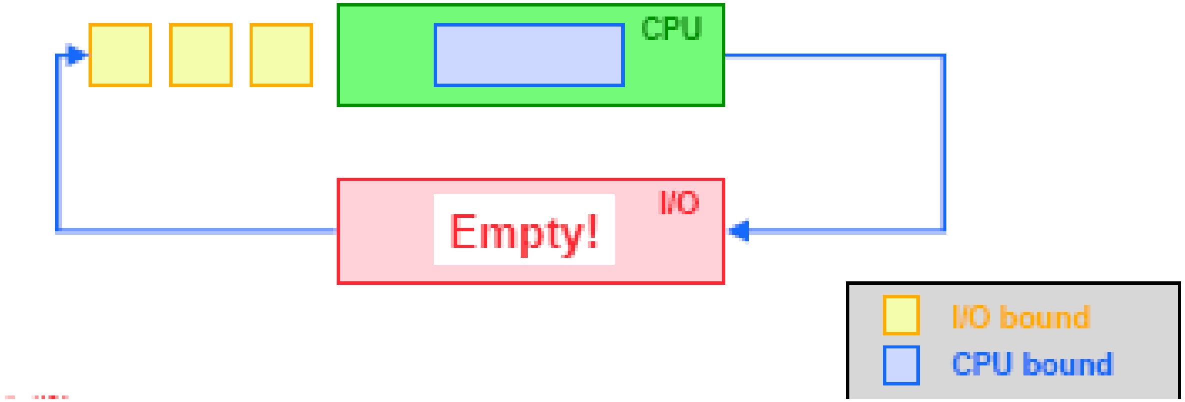
- Waiting time for $P_1 = 7; P_2 = 0_; P_3 = 3$
- Average waiting time: (7 + 0 + 3)/3 = 3.3
- Much better than previous case.

FCFS Discussion

- Advantage
  - Very simple implementation.
- Disadvantages
  - Waiting time depends on arrival order (non-preemptive).
  - Potentially long wait for jobs that arrive later (not optimal AWT).
  - Convoy effect: Short jobs stuck waiting for long jobs.
    - Hurts waiting time of short jobs.
    - Reduces utilization of I/O devices.

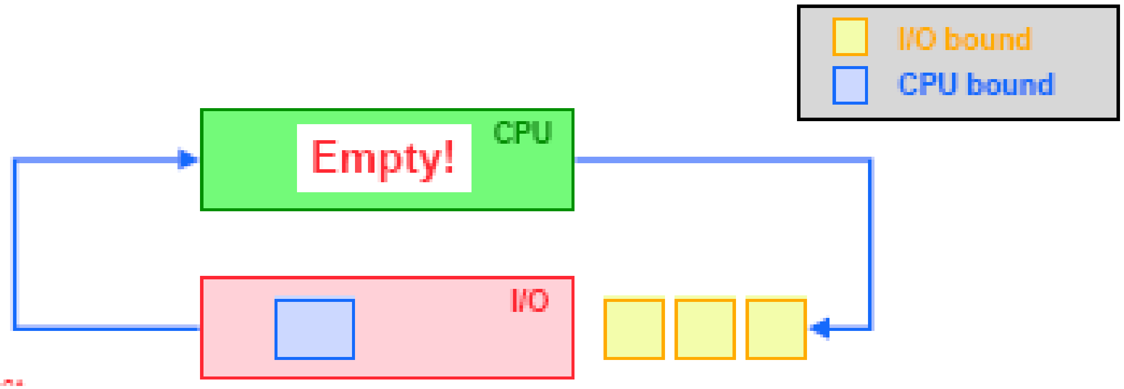
Convoy Effects (1)

- All I/O devices *idle* even when the system contains lots of I/O jobs.
  - CPU bound jobs get CPU and holds it.
  - I/O bound jobs move onto ready queue and waits.



Convoy Effects (2)

- CPU ***idle*** when even if system contains CPU bound jobs.
  - I/O jobs get CPU and finish quickly and goes back to I/O.
  - Now CPU may be idle!
  - Later I/O bound jobs again wait for CPU.

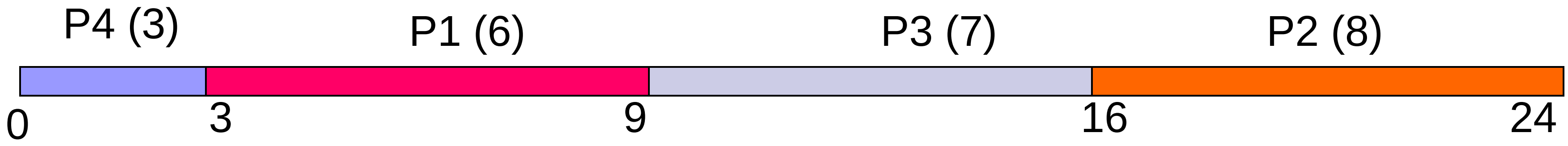


Shortest Job First (SJF)

- Schedule the job with the shortest elapse time first.
- Use FCFS if jobs are of same length.
- Scheduling in Batch Systems.
- Two Versions of SJF
  - Non-preemptive
  - Preemptive
- Requirement
  - The elapse time needs to know in advance.
- Optimal if all the jobs are available simultaneously (provable).
  - Gives the best possible AWT (average waiting time).

Non-preemptive SJF: Example

Process	Duration	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0



P4 waiting time: 0

P1 waiting time: 3

P3 waiting time: 9

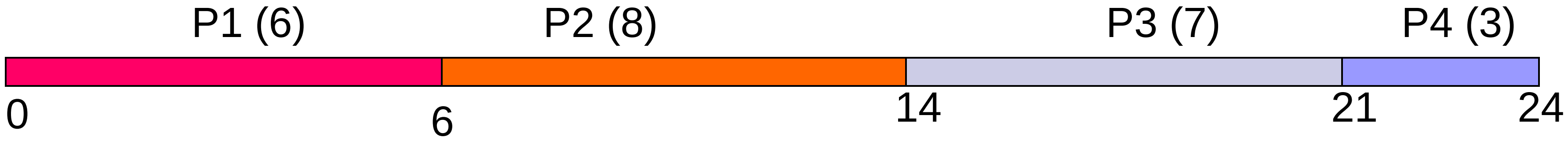
P2 waiting time: 16

The total time is: 24

The average waiting time (AWT):
(0+3+9+16)/4 = 7

Comparing to FCFS

Process	Duration	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0



P1 waiting time: 0
P2 waiting time: 6
P3 waiting time: 14
P4 waiting time: 21

The total time is the same.

The average waiting time (AWT):

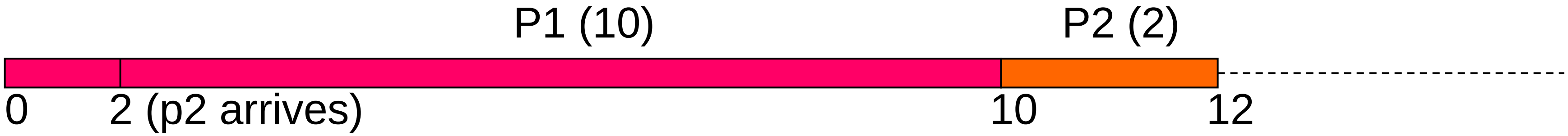
(0+6+14+21)/4 = 10.25

(comparing to 7)

SJF is not always optimal (1)

- Is SJF optimal if all the jobs are not available simultaneously?
- Example

Process	Duration	Order	Arrival Time
P1	10	1	0
P2	2	2	2



P1 (10)

P2 (2)

0

2 (p2 arrives)

10

12

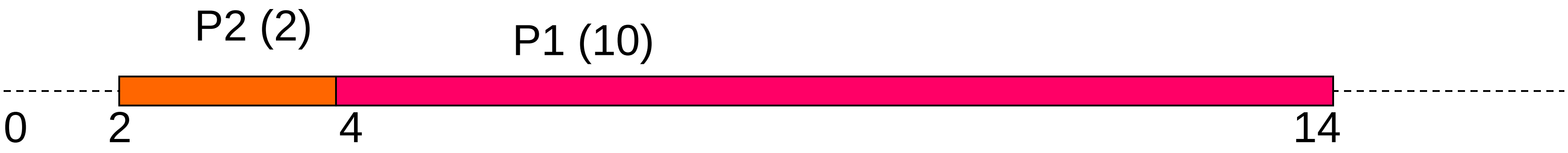
P1 waiting time: 0

P2 waiting time: 8

The average waiting time (AWT):
(0+8)/2 = 4

SJF is not always optimal (2)

- Example (ctd.)
  - What if the scheduler waits for 2 time units? (Do it yourself)



P2 waiting time: 0

P1 waiting time: 4

The average waiting time (AWT):
(0+4)/2 = 2

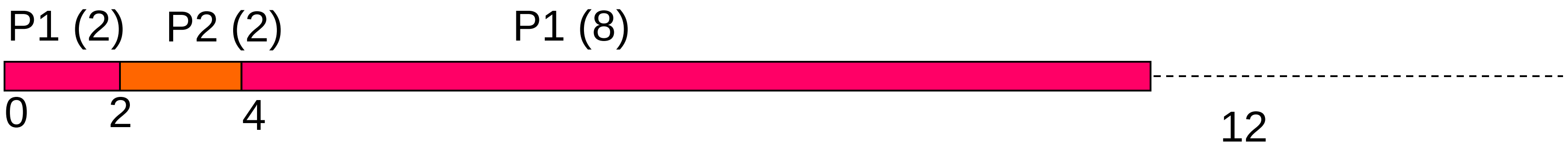
However: waste 2 time units of CPU

Preemptive SJF

- Also called *Shortest Remaining Time Next (SRTN)*
  - Schedule the job with the shortest remaining time required to complete
- Requirement: the elapse time needs to be known in advance

Preemptive SJF: Same Example

Process	Duration	Order	Arrival Time
P1	10	1	0
P2	2	2	2



P1 waiting time: 4-2 =2
P2 waiting time: 0

The average waiting time (AWT):
(0+2)/2 = 1

No CPU waste!!!

SJF Discussion

- Advantages
  - Provably optimal for minimizing average wait time (with no preemption).
    - Moving shorter job before longer job improves waiting time of short job more than it harms waiting time of long job.
  - Helps keep I/O devices busy.
- Disadvantages
  - Not practical: Cannot predict future CPU burst time.
    - OS solution: Use past behavior to predict future behavior.
  - Starvation: Long jobs may never be scheduled
- SJF implicitly incorporates priorities.
  - Shortest jobs are given preferences.

Starvation

- In some condition, a job is waiting for ever
  - Example: SJF
    - Process A with elapse time of 1 hour arrives at time 0.
    - But ever 1 minute, a short process B with elapse time of 2 minutes arrive.
    - Then C, D…
    - Result of SJF: A never gets to run.

Interactive Scheduling Algorithms

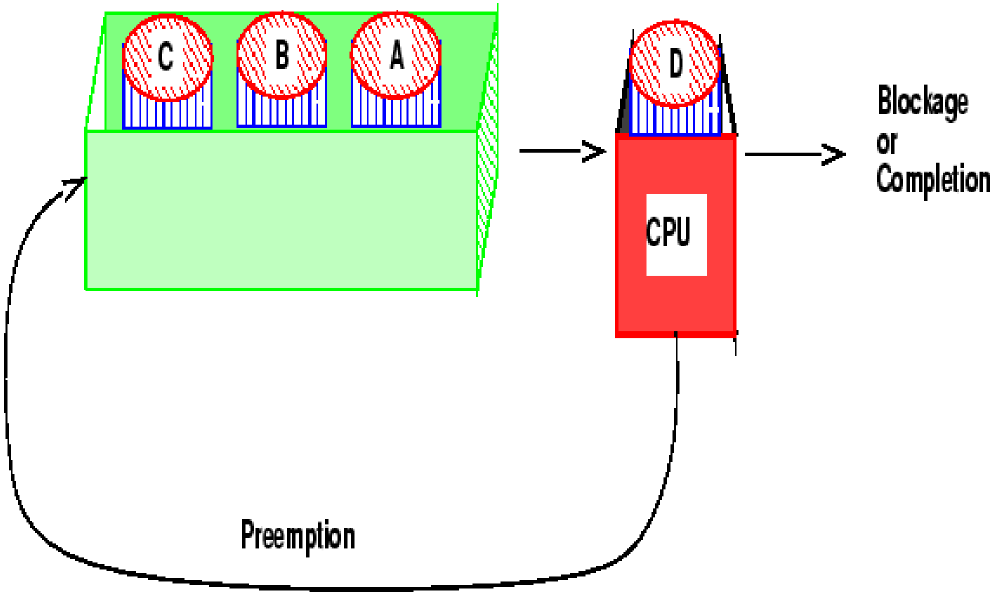
- Usually preemptive
  - Time is sliced into quantum (time intervals).
  - Scheduling decision is also made at the beginning of each quantum.
- Performance Criteria
  - Min Response time
  - Best proportionality
- Representative Algorithms
  - Round-robin Scheduling
  - Priority Scheduling
  - Multi-level Queue & Multi-level Feedback Queue
  - Shortest Process Next
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair Sharing Scheduling

Round-robin Scheduling(1)

- One of the oldest, simplest, most commonly used scheduling algorithm.
  - Each process gets a small unit of CPU time (*time quantum* or *time slice*), usually 10-100 milliseconds.
  - A process is allowed to run until the time slice period has expired, then a clock interrupt occurs and the running process is put on the ready queue.
  - If the process has blocked or finished before the quantum has elapsed, the CPU switching is done.

Round-robin Scheduling (2)

- Assumes all processes have same priority.
  - Guaranteed to be starvation-free.
- Preemptive FCFS.
- Ready queue treated as circular queue.
- Processes that exceed their time slice return to the end of the ready queue.



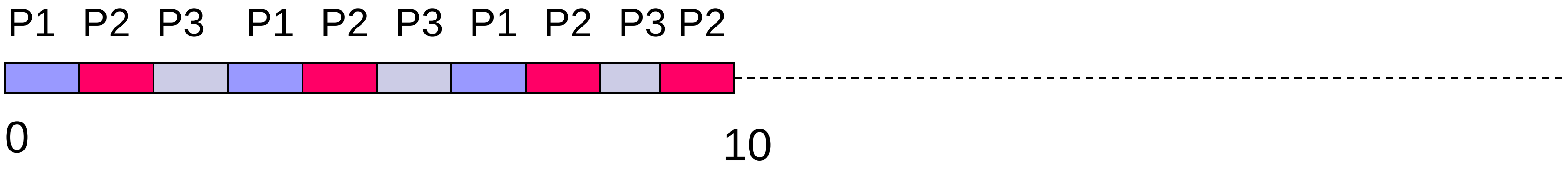
Round-robin Scheduling (3)

- n processes in the ready queue and time quantum q
  - Each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once.
  - No process waits more than (*n*-1)*q* time units.
    - Good response time.

Round-robin: Example

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is: 1 unit, P1, P2 & P3 never block



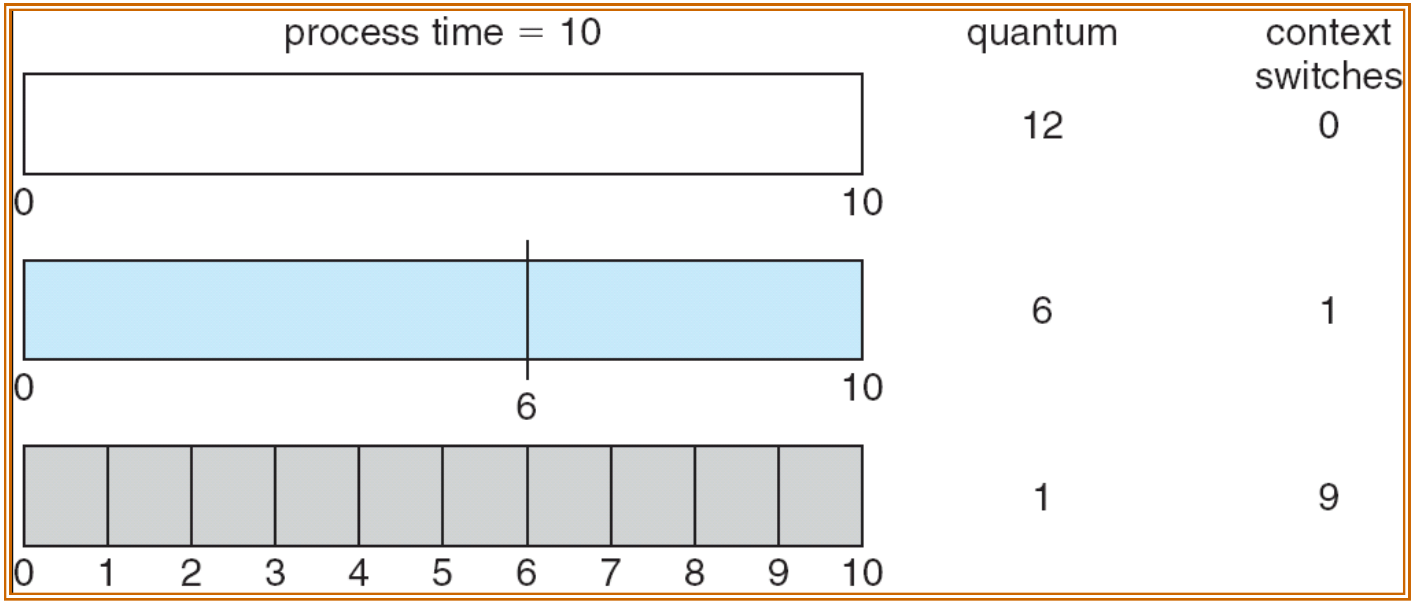
P1 waiting time: 4
P2 waiting time: 6
P3 waiting time: 6

The average waiting time (AWT): (4+6+6)/3 = 5.33

Time Quantum (1)

- Time quantum too large
  - If quantum size is large enough to finish most processes within a single quantum →Low context switch overhead.
  - If quantum size is too large → FCFS behavior, poor response time.
- Time quantum too small
  - Too many context switches (overheads)
    - Context switch time= 0.1-1ms
  - Inefficient CPU utilization.

Time Quantum (2)



quantum	context switches
12	0
6	1
1	9

Quantum Size vs. Context Switch Tradeoff

Time Quantum (3)

- Actual Choices of Time Quantum
  - Initially, UNIX time slice is 1 second.
    - Worked OK when UNIX was used by one or two people.
    - What if three compilations going on? 3 seconds to echo each keystroke!
  - In practice, need to balance short-job performance and long-job throughput:
    - Typical time slice today is between 10ms – 100ms.
    - Typical context-switching overhead is 0.1ms – 1ms.
    - Roughly 1% overhead due to context-switching.

Round-robin Discussion (1)

- Advantages
  - Fair allocation of CPU across jobs, no starvation.
  - Low average waiting time when job lengths vary.
  - Shortest jobs finish relatively quickly.
  - Turnaround time typically larger than SRTF but better response time.

Round-robin Discussion (2)

- Disadvantages
  - Poor average waiting time with similar job lengths.
    - Example: 3 jobs each requiring 3 time slices
    - RR: AWT=(4+5+6)/3=5
    - FCFS performs better! AWT=(0+3+6)/3=3
  - 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.
  - Do not consider priority.
  - High overhead due to frequent context switches.

Priority Scheduling (1)

- Each job is assigned a priority.
- FCFS within each priority level.
- Select highest priority job over lower ones.
- Rational: Higher priority jobs are more mission-critical.
  - Example: DVD movie player vs. send email
- SJF is priority scheduling where priority is inversely proportional to length of next CPU burst.

Priority Scheduling (2)

- Priority Setting
  - Two Approaches
    - Static (for system with well known and regular application behaviors)
      - A process with a static priority keeps that priority for the entire life of the process.
    - Dynamic (otherwise)
      - A process with a dynamic priority will have that priority changed by the scheduler during its course of execution.
    - Static and dynamic priorities can coexist.

Priority Scheduling (3)

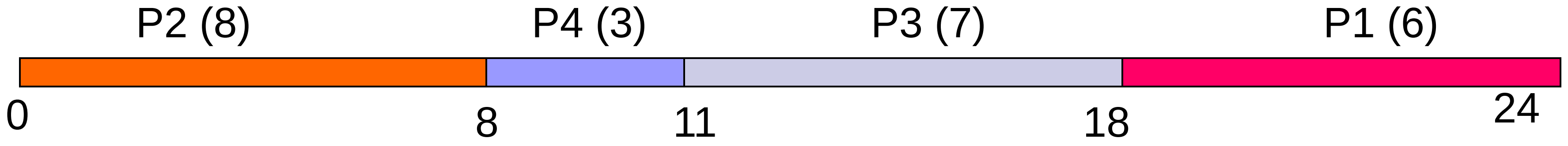
- Priority Setting (ctd.)
  - Priority may be based on
    - Cost to user
    - Importance of user
    - Process type
    - Requirement to resource
    - Aging
    - Percentage of CPU time used in last X hours.

Priority Scheduling (4)

- Dynamic Priority Example
  - Assign priority depending on the fraction of quantum each process has used.
  - P = 1/ f, f is the fraction of the last quantum that a process used.
  - Time slice: 50ms
  - process A uses 1ms, f=1/50, priority = 50
  - process B uses 50 ms, f=50/50, priority = 1

Priority Scheduling: Example

Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0

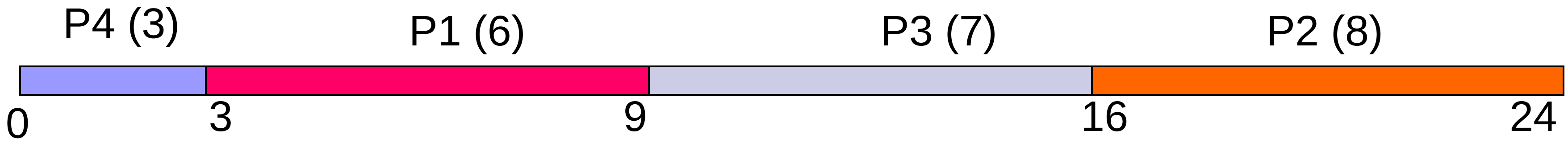


P2 waiting time: 0
P4 waiting time: 8
P3 waiting time: 11
P1 waiting time: 18

The average waiting time (AWT):
(0+8+11+18)/4 = 9.25
(worse than SJF 7)

Non-preemptive SJF: Example

Process	Duration	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0



P4 waiting time: 0
P1 waiting time: 3
P3 waiting time: 9
P2 waiting time: 16

The total time is: 24

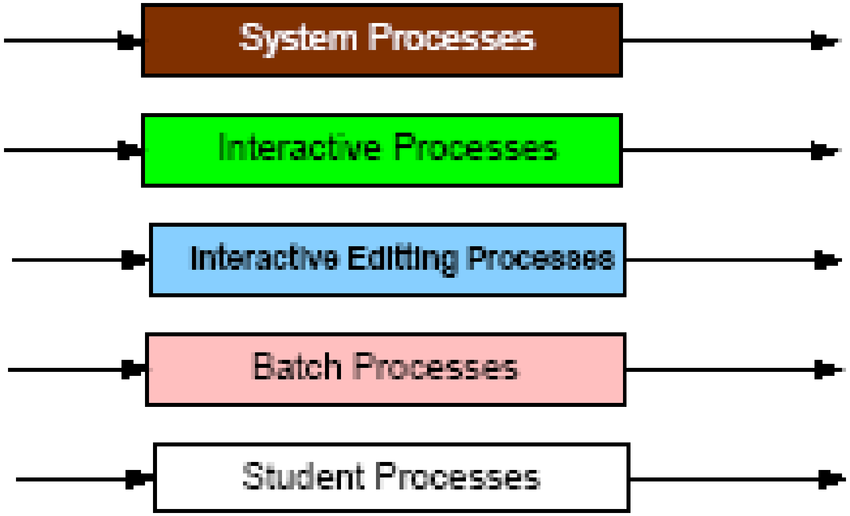
The average waiting time (AWT):
(0+3+9+16)/4 = 7

Priority Scheduling Discussion

- Advantage
  - Provide a good mechanism where the relative importance of each process may be precisely defined.
- Disadvantages
  - Low priority jobs can starve.
    - Can avoid by aging processes: increase priority as they spend time in the system.
  - May not give the best AWT.

Multi-Level Queue (1)

- Extension of priority scheduling, which also combines other strategies.
- Used in scenarios where the processes can be classified into groups based on property like process type, CPU time, IO access, memory size, etc.
- Split the Ready Queue in several queues, processes assigned to one queue *permanently*.

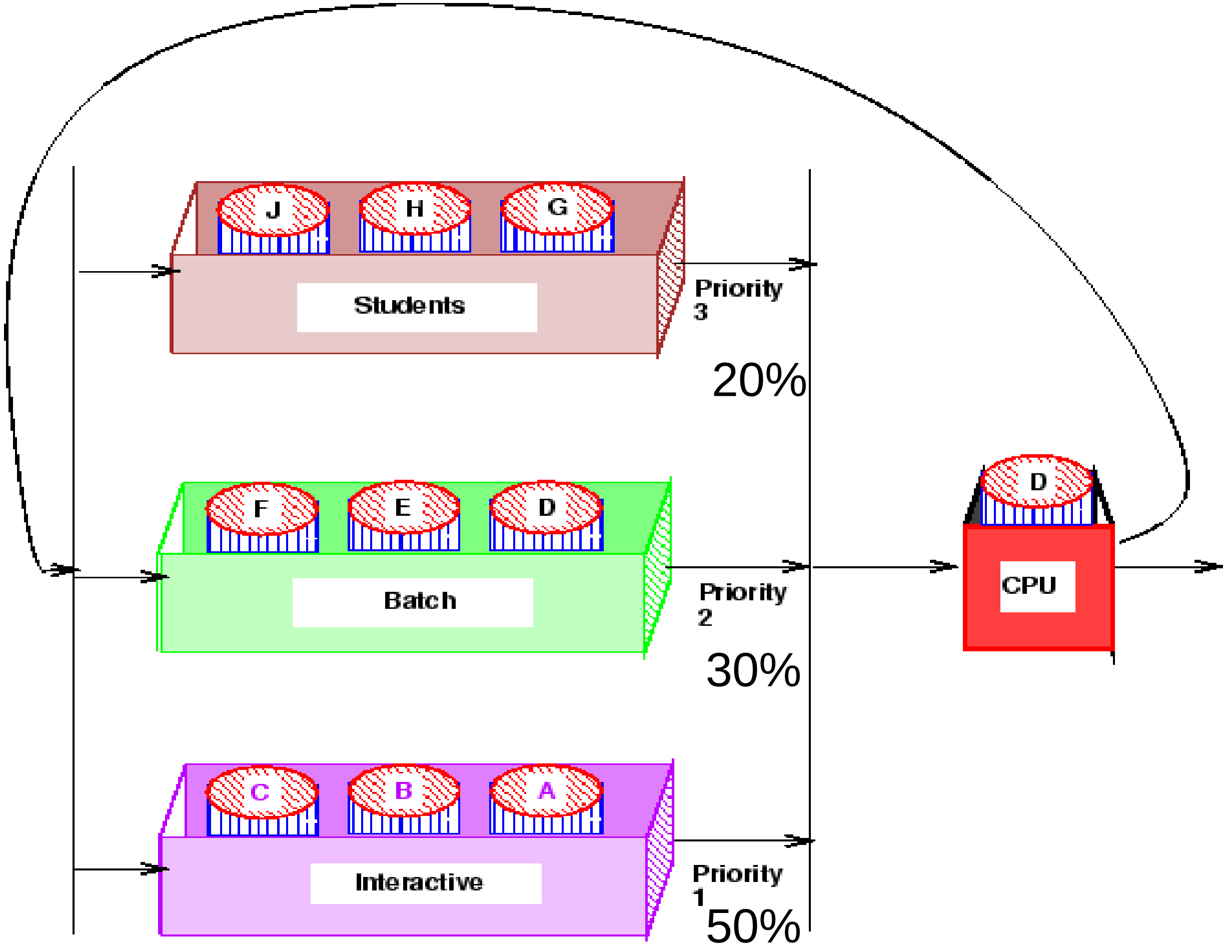


Multi-Level Queue (2)

- Scheduling between queues
  - □ Fixed Priorities
    - E.g. 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 (% CPU spent on queue).
    - E.g. 80% to foreground process，20% to background process
  - □ Each queue with its own scheduling algorithm.
    - E.g. interactive processes: RR, background processes: FCFS/SRTF

Multi-Level Queue (3)

- Scheduling between queues (ctd.)
  - Example



Multi-Level Queue (4)

- Real Life Analogy
  - Tasks (to-do list) for poor *Bob*
    - Class 1 priority (highest) : tasks given by his boss
      - Finish the project (50%)
    - Class 2 priority: tasks for his wife
      - Buy a valentine present (30%)
    - Class 3 priority (lowest): Bob’s tasks
      - Watch TV (20%)

Multi-Level Queue (5)

- Problem
  - The process needs to be assigned to the most suitable priority queue a priori.
    - E.g. If a CPU-bound process is assigned to a short-quantum, high-priority queue, that’s not optimal for either the process nor for overall throughput.

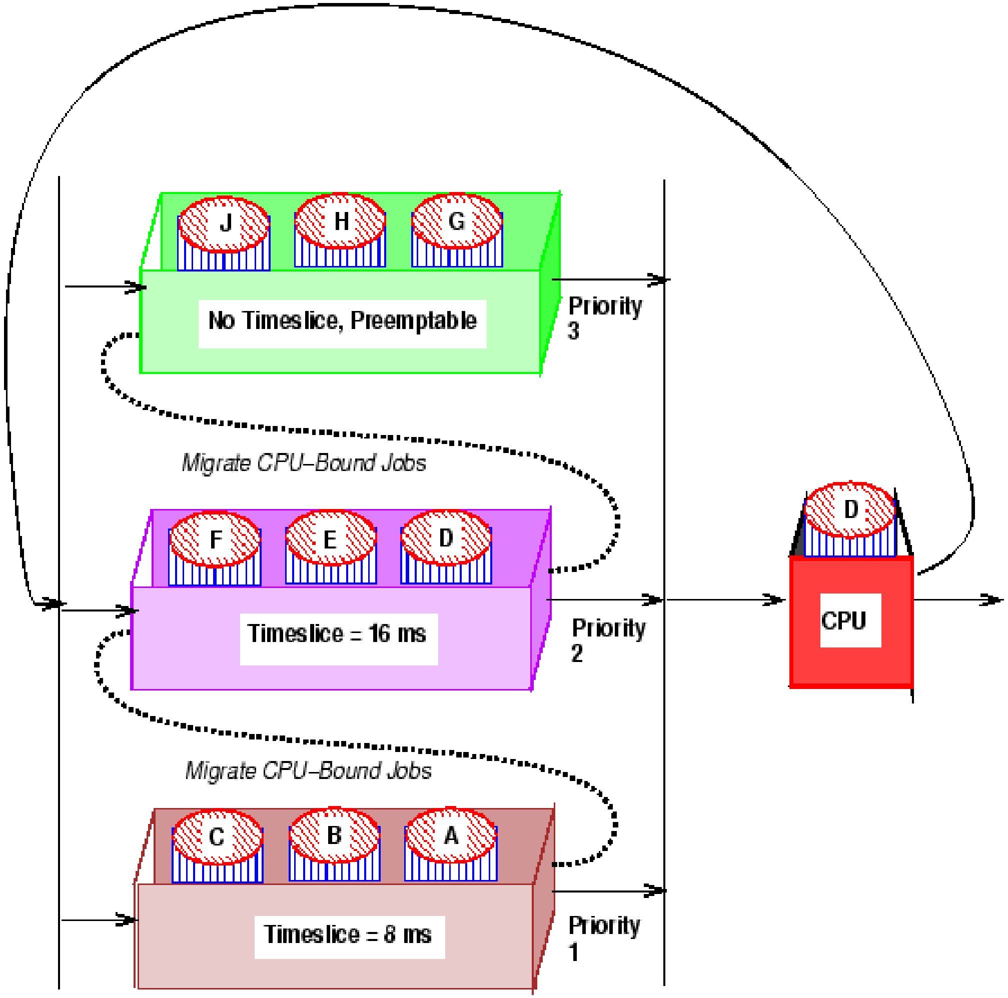
Multi-Level Feedback Queue (1)

- First described in 1962 in CTSS.
- Processes move between queues.
  - Start each process in a high-priority queue; as it finishes each CPU burst, move it to a lower-priority queue.
  - Feedback = use the past to predict the future — favor jobs that haven’t used the CPU much in the past — close to SRTN!
- Each queue represents jobs with similar CPU usage.
- Jobs in a given queue are executed with a given time slice.

Multi-Level Feedback Queue (2)

- Example
  - Three queues:
    - Q_0 – time quantum 8 milliseconds
    - Q_1 – 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.

Multi-Level Feedback Queue (3)

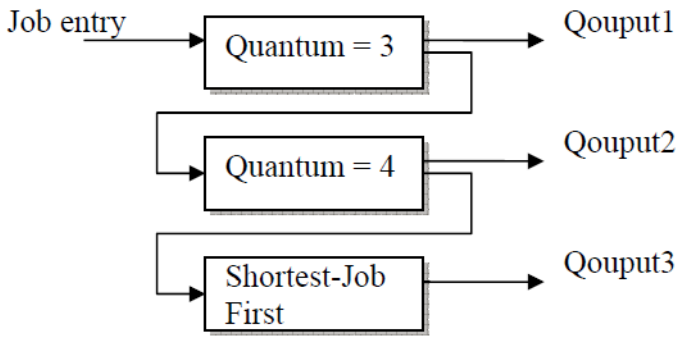


Multi-Level Feedback Queue (4)

- Advantages
  - Good for separating processes based on CPU burst needs.
  - Let I/O bound processes run often.
  - Give CPU-bound processes longer chunks of CPU.
- Disadvantages
  - Priorities get controlled by the system. A process is considered important because it uses a lot of I/O.
  - Processes whose behavior changes may be poorly scheduled.

Multi-Level Feedback Queue: Exercise (1)

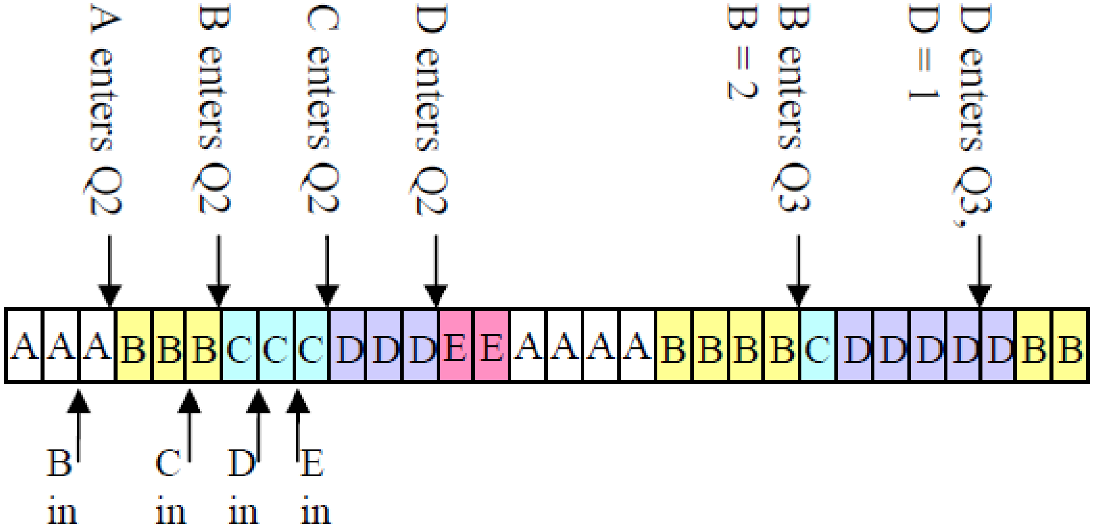
- Problem: Show your schedule with timeline and Calculate the **average turnaround time** when use the **multi-level feedback queue** as below. (Please take arrival time into account.) Note that the priority of the top 2 queues is based on arrival times.



Process ID	Arrival Time	Burst Time
A	0	7
B	2	9
C	5	4
D	7	8
E	8	2

Multi-Level Feedback Queue: Exercise (2)

- Solution:



Turnaround Time

A = 18

B = 28

C = 18

D = 21

E = 6

Average = 91/5 = 18.2

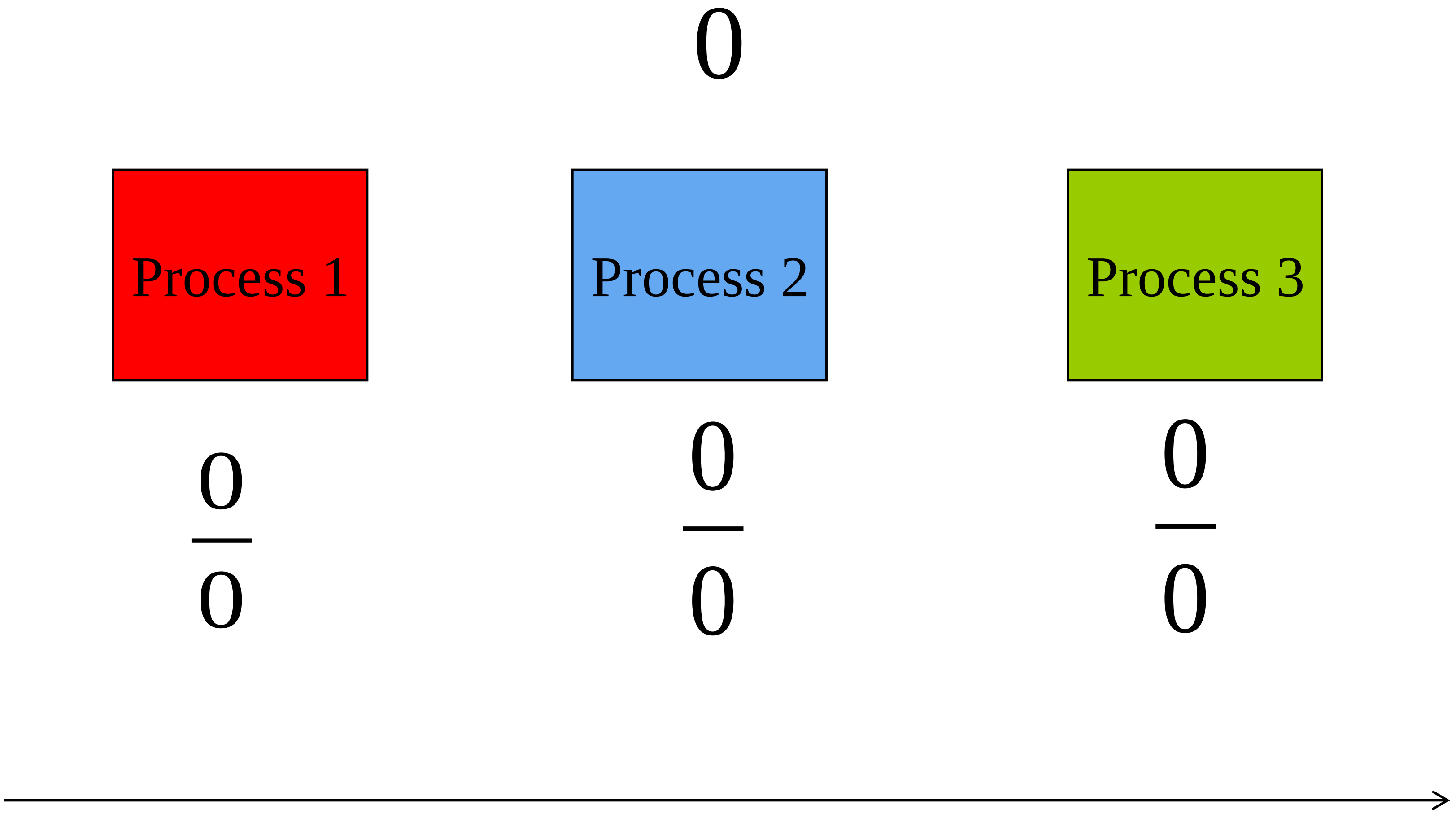
Shortest Process Next

- Scheduler selects process with smallest time to finish.
  - Lower average wait time than FCFS.
    - Reduces the number of waiting processes.
  - Nonpreemptive
    - Results in slow response times to arriving interactive requests.
  - Relies on estimates of time-to-completion.
    - Can be inaccurate or falsified.
  - Unsuitable for use in modern interactive systems.

Guaranteed Scheduling (QoS)

- Make real promises to the users about performance and then live up to them.
- Example
  - With n processes running, the scheduler makes sure that each one gets 1/n of the CPU cycles.
  - If there are n users, the scheduler makes sure that each will get 1/n of the CPU time.
- Scheduling
  - Compute the ratio of actual CPU time consumed to CPU time entitled (the time since creation divided by n)
  - Select the one with the lowest ratio
  - Switch to another process when the ratio of the running process has passed its *“goal ratio”.*

Guaranteed Scheduling: Example (1)



0

Process 1

Process 2

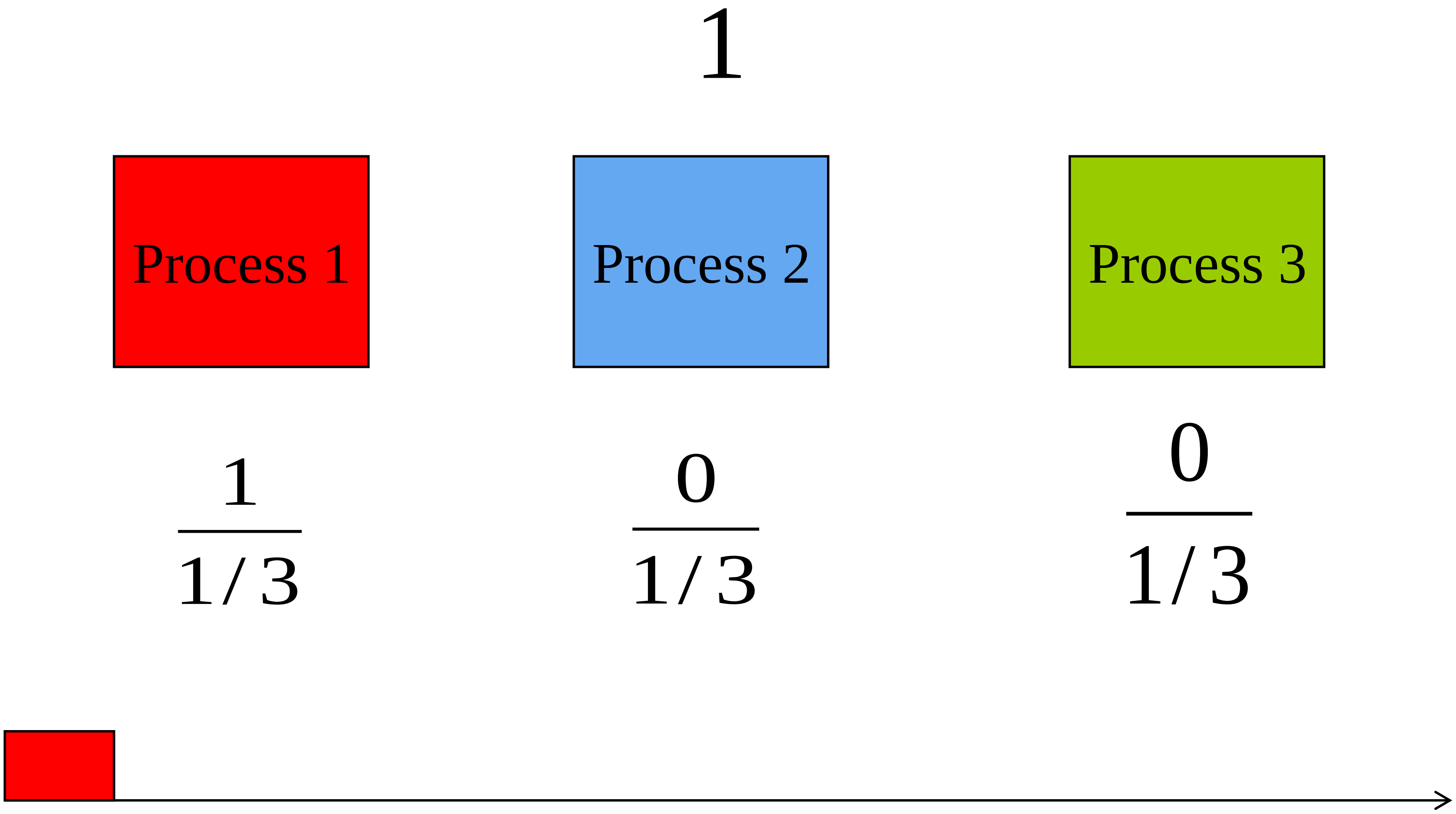
Process 3

$\frac{0}{0}$

$\frac{0}{0}$

$\frac{0}{0}$

Guaranteed Scheduling: Example (2)



1

Process 1

Process 2

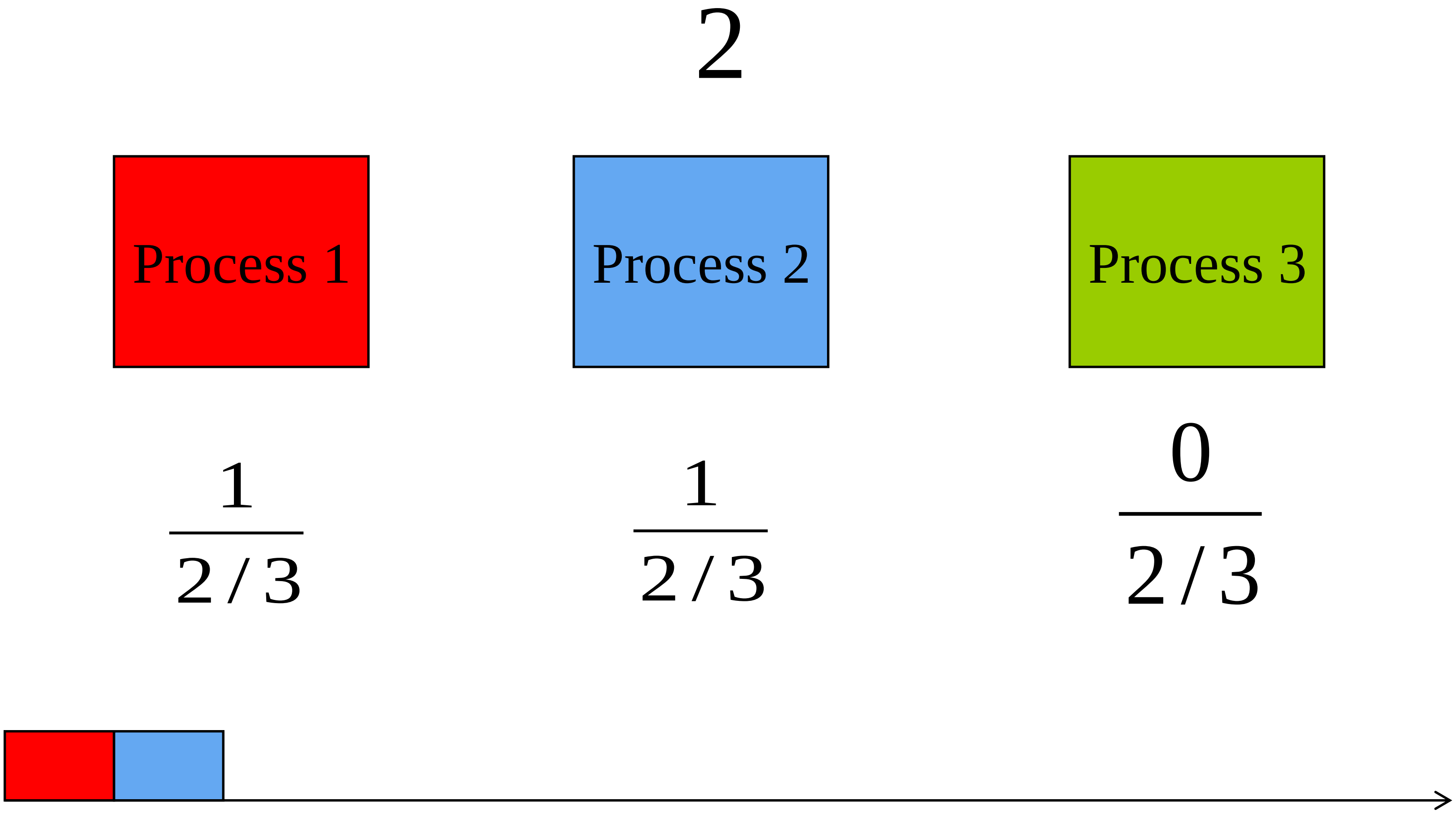
Process 3

$$\frac{1}{1/3}$$

$$\frac{0}{1/3}$$

$$\frac{0}{1/3}$$

Guaranteed Scheduling: Example (3)



2

Process 1

Process 2

Process 3

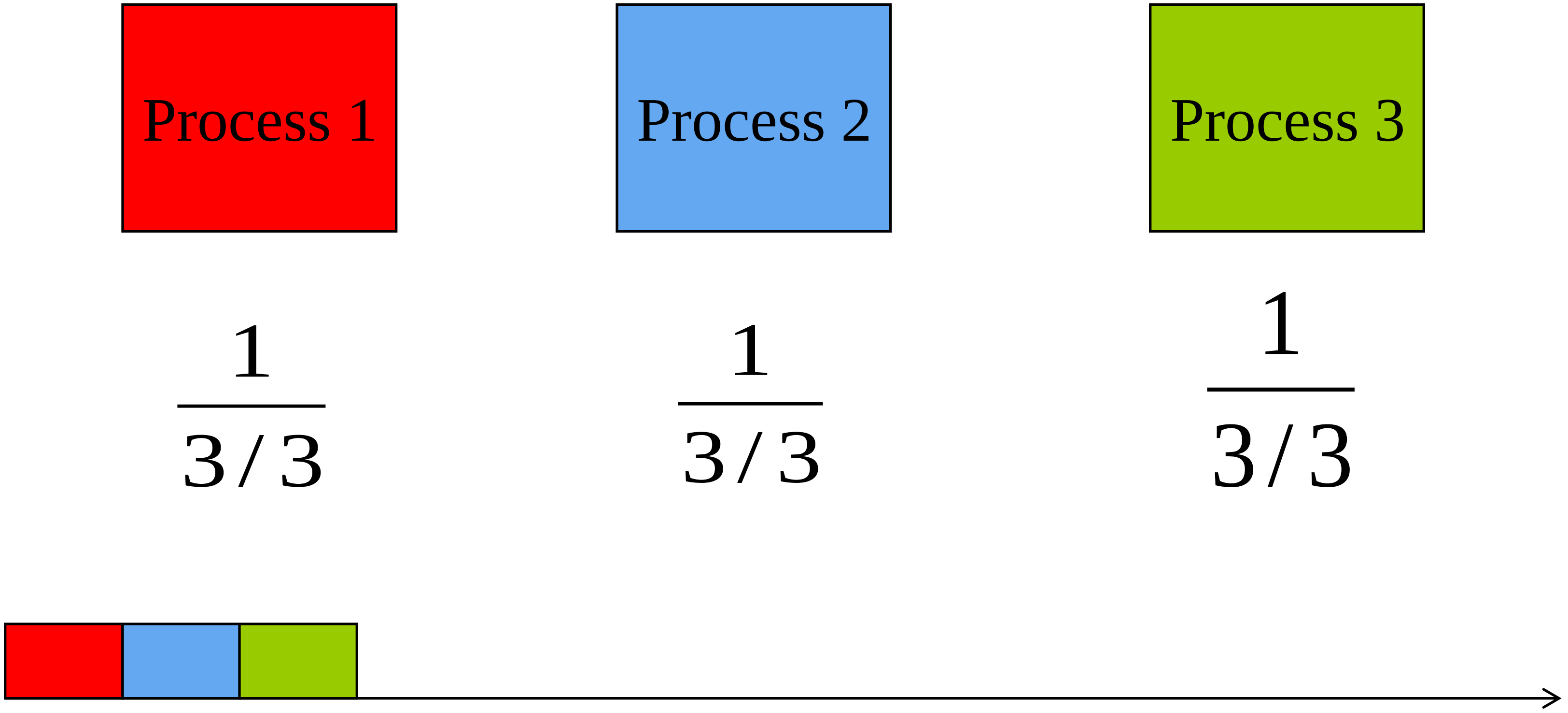
$$\frac{1}{2/3}$$

$$\frac{1}{2/3}$$

$$\frac{0}{2/3}$$

Guaranteed Scheduling: Example (4)

3



Process 1

Process 2

Process 3

$$\frac{1}{3/3} \qquad \frac{1}{3/3} \qquad \frac{1}{3/3}$$

Guaranteed Scheduling: Example (5)

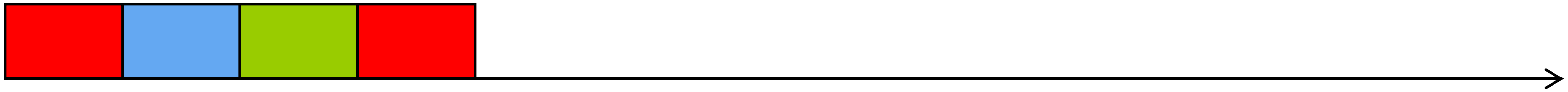
4

Process 1

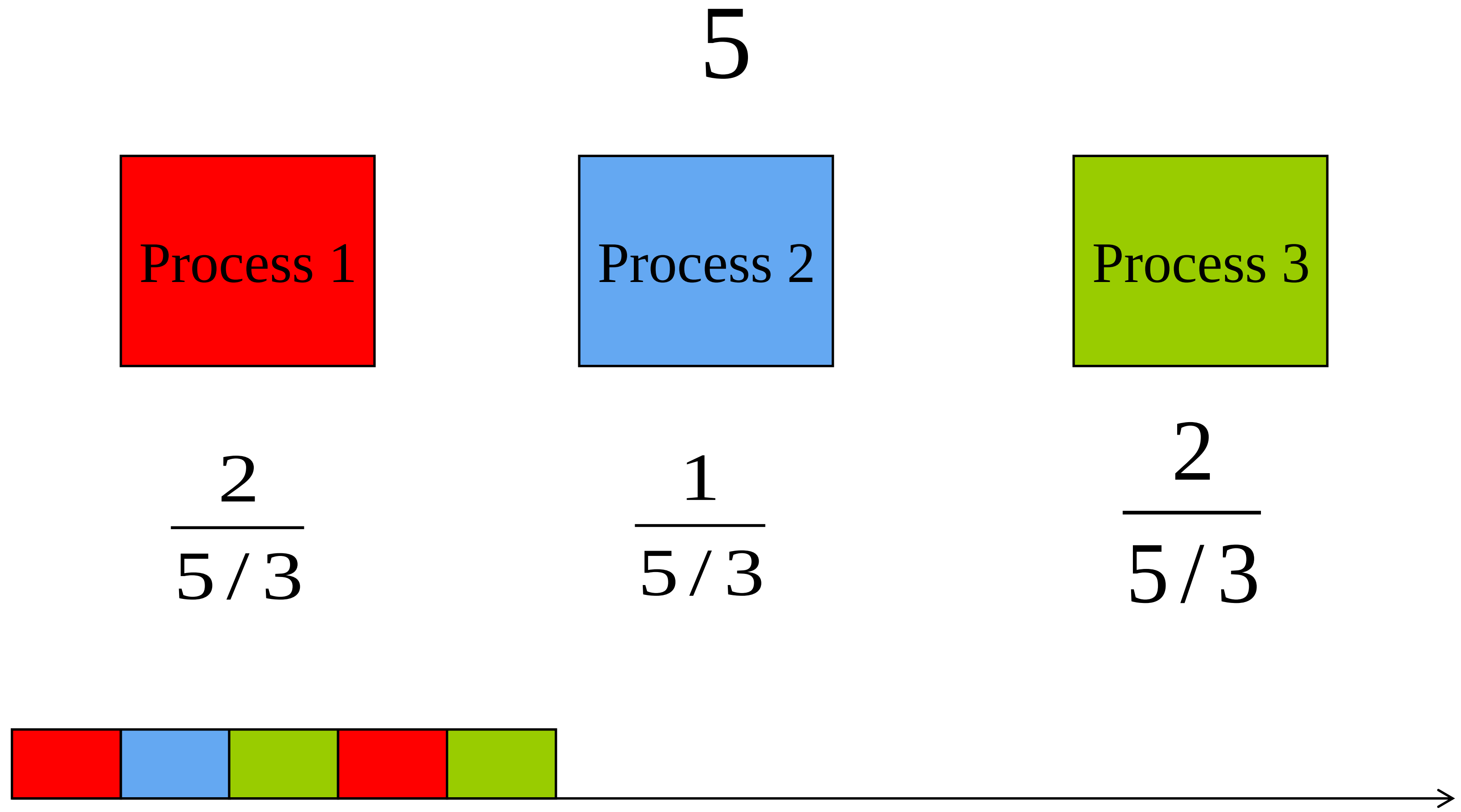
Process 2

Process 3

$\frac{2}{4/3}$ $\frac{1}{4/3}$ $\frac{1}{4/3}$



Guaranteed Scheduling: Example (6)



5

Process 1

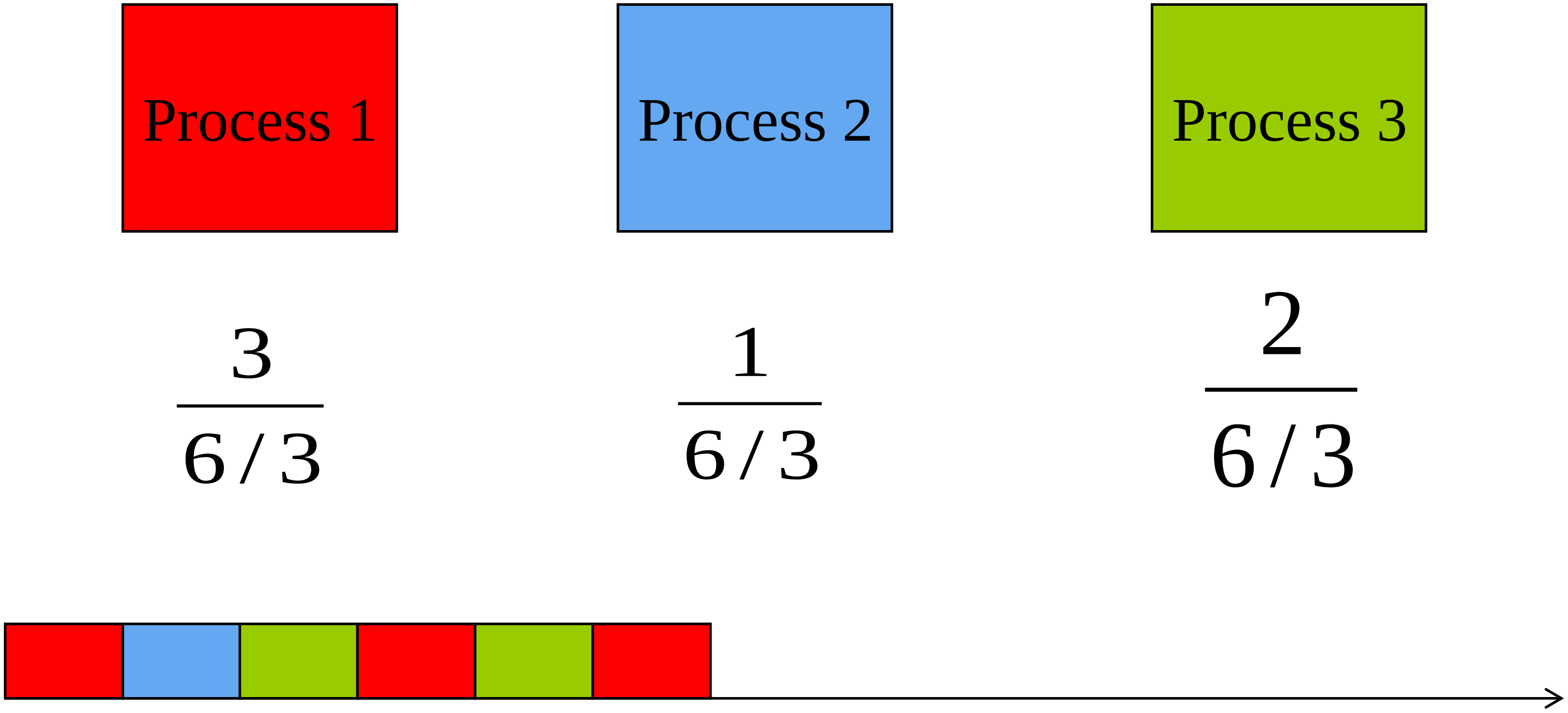
Process 2

Process 3

$$\frac{2}{5/3} \qquad \frac{1}{5/3} \qquad \frac{2}{5/3}$$

Guaranteed Scheduling: Example (7)

6



Process 1

Process 2

Process 3

$$\frac{3}{6/3}$$

$$\frac{1}{6/3}$$

$$\frac{2}{6/3}$$

Guaranteed Scheduling: Example (8)

7

Process 1

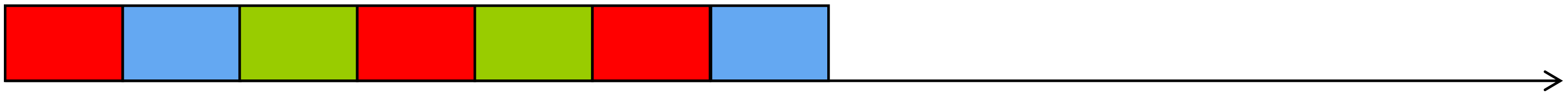
Process 2

Process 3

$$\frac{3}{7/3}$$

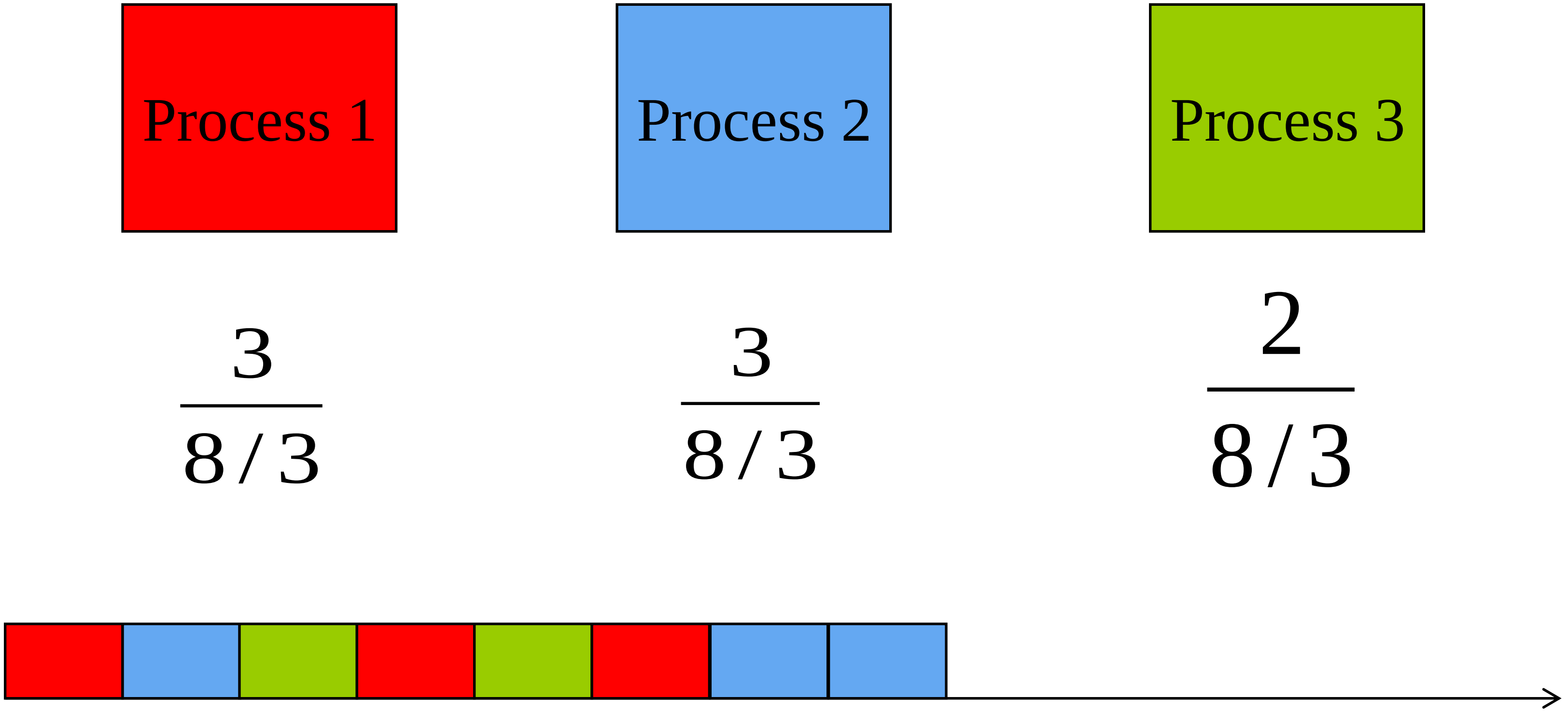
$$\frac{2}{7/3}$$

$$\frac{2}{7/3}$$



Guaranteed Scheduling: Example (9)

8



Process 1

Process 2

Process 3

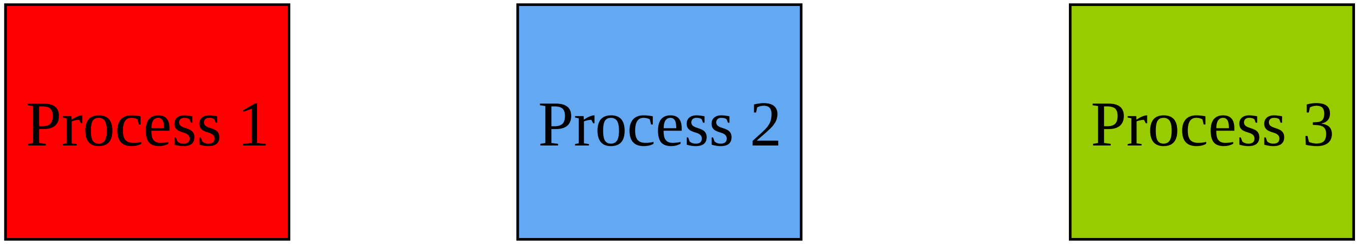
$$\frac{3}{8/3}$$

$$\frac{3}{8/3}$$

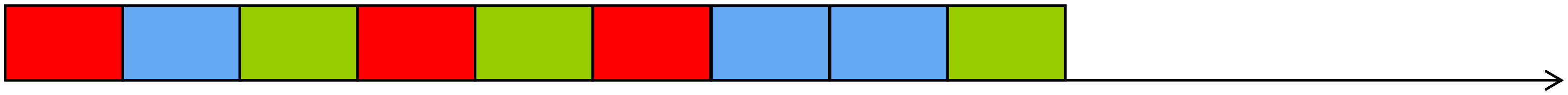
$$\frac{2}{8/3}$$

Guaranteed Scheduling: Example (10)

9



$$\frac{3}{9/3} \qquad \frac{3}{9/3} \qquad \frac{3}{9/3}$$



Lottery Scheduling (1)

- More commonly used
- Probability-based
  - Give processes lottery tickets. At scheduling time, a lottery ticket is chosen at random, and the process holding that ticket gets that resource.
- Give more tickets for higher priority processes
- On average: CPU time proportional to # of tickets

Lottery Scheduling (2)

- Example
  - □ Short jobs: 10 tickets each
  - □ Long jobs: 1 ticket each

# short jobs/# long jobs	% of CPU for each short job	% of CPU for each long job
1/1	91%	9%
0/2	0%	50%
2/0	50%	0%
10/1	10%	1%
1/10	50%	5%

Lottery Scheduling (3)

- How to use?
  - Approximate priority: low-priority, give few tickets, high-priority give many
  - Approximate SJF: give short jobs more tickets, long jobs fewer. Key: If job has at least 1, will not starve
- Advantages
  - Simple
  - Highly responsive
  - Can support cooperation between processes.
  - Easy to support priority and proportion requirement.

Lottery Scheduling (4)

- Disadvantage
  - The difficulty is determining ticket distribution, particularly in an environment where processes come and go and get blocked.
  - This isn’t a useful algorithm for general-purpose scheduling but is more useful for environments with long-running processes that may need to be allocated shares of CPUs, such as running multiple virtual machines on a server.

Fair-Share Scheduling

- Is Round-robin fair?
  - Yes, it is (from process point of view)
  - No, it may be not (from user point view)
- User-based fair share scheduling
  - Each user gets fair share
- Example
  - Alice has 4 processes: A1, A2, A3, A4
  - Bob has 1 process: B1
  - Then A1, A2, A3, A4 are entitled only to 50% CPU, while B1 alone is entitled to 50%

Real-Time Systems (1)

- Real-Time Systems
  - Systems whose correctness depends on their temporal aspects as well as their functional aspects.
- Real-Time Demands
  - We don’t always need a LOT of CPU time but we may need it at the right intervals.
    - E.g., Decode 30 frames per second of video
  - We might have tight deadlines
    - E.g., Complete task within the next 500 ms

Real-Time Systems (2)

- Start Time
  - The time at which a task starts execution.
- Deadline
  - The time before which a task should be completed to avoid damage to the system.
- Finishing Time
  - The time at which a task finishes execution.
- Hard Deadline
  - Disastrous or very serious consequences may occur if the deadline is missed.

Real-Time Systems (3)

- Soft Deadline
  - Ideally, the deadline should be met for maximum performance. The performance degrades in case of deadline misses.
- Hard Real-Time Systems
  - Required to complete a critical task within a guaranteed amount of time.
  - Failure to meet deadline can result in disaster.
  - E.g., air bag controller, autopilot system in plane, nuclear plant control system

Real-Time Systems (4)

- Soft Real-Time Systems
  - Missing an occasional deadline is undesirable, but nevertheless tolerable.
  - Failure to meet deadline results in degraded performance.
  - E.g, the sound system in your computer, mobile communication

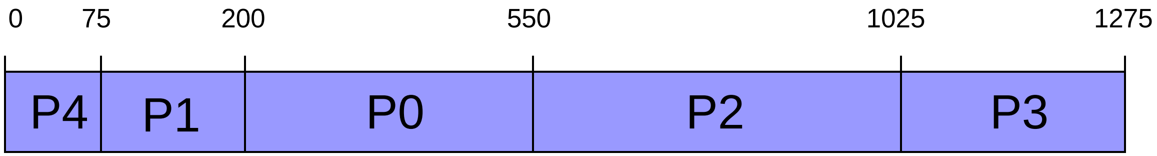
Scheduling in Real-Time Systems (1)

- Earliest Deadline First (EDF)
  - For hard real-time systems, process must finish by a certain time.
  - Turnaround time and wait time are irrelevant.
  - We need to know maximum service time for each process.
  - Deadline must be met for each period in a process’s life.
  - A task with a shorter deadline has a higher priority.
  - Executes a task with the earliest deadline.

Scheduling in Real-Time Systems (2)

- Earliest Deadline First (EDF) (ctd.)
  - Example

i	**t(pi)**	**Deadline**	**i**	**t(pi)**	**Deadline**
0	350	575	1	125	550
2	475	1050	3	250	(none)
4	75	200			



Schedulability in Real-Time Systems (1)

- Events in Real-Time System
  - □ Periodic (occurring at regular intervals)
  - □ Aperiodic (unpredictable)
- Schedulability
  - □ Property indicating whether a real-time system (a set of real-time tasks) can meet their deadlines.
  - □ Consider *m* periodic events, event *i* occurs with period *Pi* and requires *Ci* second of CPU time the system is schedulable if: $\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$

Schedulability in Real-Time Systems (2)

- Example
  - Multimedia System: three processes A, B, C
  - A is periodic, T = 30ms, and uses 10 ms of CPU time
  - B is periodic, f = 25 Hz (T=40ms) and uses 15 ms of CPU time
  - C is periodic, f = 20 Hz, (T=50ms) and uses 5 ms of CPU time
  - Schedulability ? 10/30 + 15/40 + 5/50 = 0.808 < 1

Policy versus Mechanism (1)

- Given a particular task
  - Policy refers to what needs to be done (i.e. activities to perform).
  - Mechanism refers to how to do it (i.e. implementation to enforce policy).
- Policy vs Mechanism Example
  - A website requires users to login to the system. (policy)
  - Users can login using a user name and password pair. (mechanism)
  - Users can also login using their QQ or Wechat accounts. (another mechanism)

Policy versus Mechanism (2)

- Scheduling policy answers the question: Which process/thread, among all those ready to run, should be given the chance to run next?
    - For example, is it priority based ? or just round robin ?

Policy versus Mechanism (3)

- Scheduling mechanisms are the tools for supporting the process/thread abstractions and affect how the scheduling policy can be implemented.
  - How the process or thread is represented to the system - process or thread control blocks.
  - What happens on a context switch.
  - When do we get the chance to make these scheduling decisions (timer interrupts, thread operations that yield or block, user program system calls)

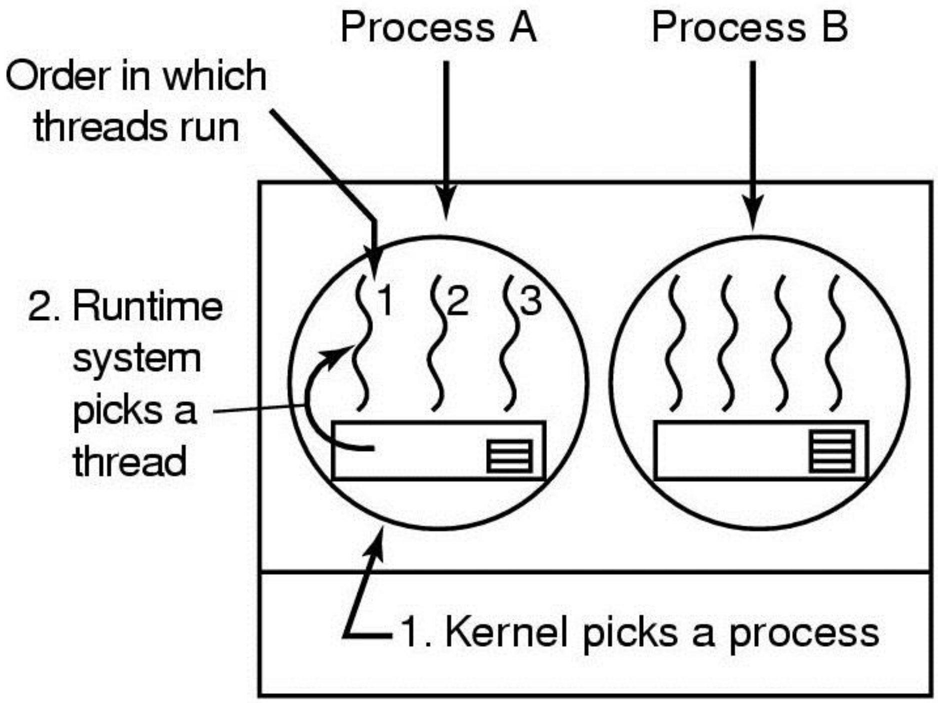
Policy versus Mechanism (4)

- Separation of Policy and Mechanism
  - Separation of policy and mechanism is a design principle to achieve flexibility.
  - The scheduling algorithm is parameterized in some way, but the parameters can be filled in by user processes.
  - The policy is set by a user process, the mechanism is in the kernel.

User-Level Thread Scheduling

Possible Scheduling

- 50-msec process quantum
- run 5 msec/CPU burst

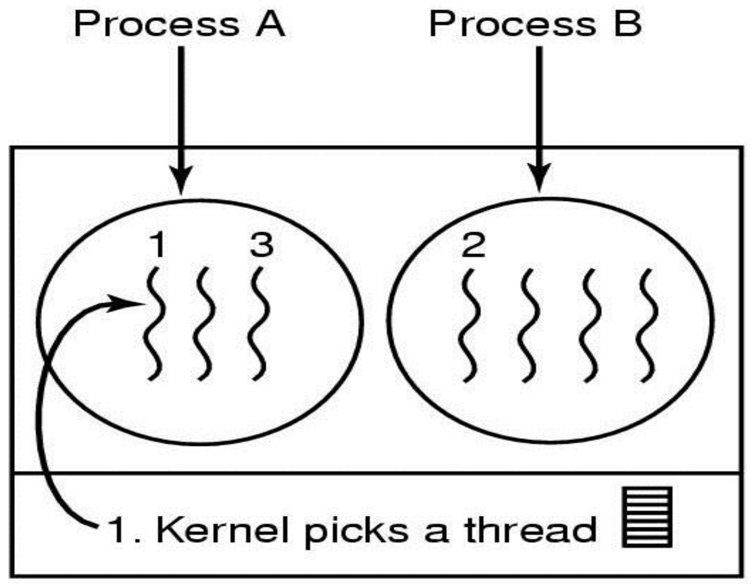

Process A
Process B
Order in which threads run
1
2
3
2. Runtime system picks a thread
1. Kernel picks a process

Possible: A1, A2, A3, A1, A2, A3
Not possible: A1, B1, A2, B2, A3, B3

Kernel-Level Thread Scheduling

Possible scheduling

- 50-msec time quantum
- threads run 5 msec/CPU burst



Possible: A1, A2, A3, A1, A2, A3
Also possible: A1, B1, A2, B2, A3, B3

Summary (1)

- What is scheduling?
- Batch System Scheduling
  - FCFS
  - Shortest Job first (SJF)
  - Shortest Remaining Time Next (SRTN)
- Interactive System Scheduling
  - Round-robin
  - Priority
  - Multi-level Queue & Multi-level Feedback Queue

Summary (2)

- Interactive System (ctd.)
  - Shortest Process Next
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair Sharing Scheduling
- Scheduling for Real-time Systems
  - Schedulability
- Thread Scheduling
  - User-level thread scheduling
  - Kernel-level thread scheduling

Exercise (1)

- 1. Which of the following does not interrupt a running process?
  - A. A device
  - B. Timer
  - C. Scheduler process
  - D. Power failure

Exercise (2)

- 2. An I/O bound program will typically have ( ).
  - □A. a few very short CPU bursts
  - □B. many very short I/O bursts
  - □[✓ C. many very short CPU bursts]
  - □D. a few very short I/O bursts

Exercise (3)

- 3. If all the jobs are available simultaneously, we can prove that (   ) scheduling algorithm is optimal.
  - □ A. First Come First Served (FCFS)
  - □ B. Shortest Remaining Time Next (SRTN)
  - □ [✓ C. Shortest Job First (SJF)]
  - □ D. Lottery

Exercise (4)

- 4. With round robin scheduling algorithm in a time shared system,( ).
  - □ [✓ A. using very large time slices converts it into first come first served scheduling algorithm]
  - □ B. using very small time slices converts it into first come first served scheduling algorithm
  - □ C. using extremely small time slices increases performance
  - □ D. using very small time slices converts it into shortest job first algorithm

Homework (1)

- Reading assignment: Chapter 10.3
- P178　47，50
- 补充题

1. Consider the following set of processes, with the length of CPU burst time given in milliseconds

Process	Duration	Priority	Arrival Time
P1	10	3	0
P2	1	1	0
P3	2	5	0
P4	1	4	0
P5	5	2	0

Homework (2)

- 补充题 （续）

1. The processes are assumed to have arrived in the order of P1,P2,P3,P4,P5 all at time 0.

a. Draw four Gantt chart illustrating the execution of these processes using FCFS, SJF, a non-preemptive priority (a smaller priority number implies a higher priority) and RR scheduling (time quantum=1).

b. What is the turn around time of each process for each scheduling algorithm in part a?

c. What is the waiting time of each process for each scheduling algorithm in part a?

d. Which of the schedules in part a results in the minimal average waiting time?

Homework (3)

- 补充题

2. There are five jobs A, B, C, D, E, whose arrival time and running time are shown in the table below. The quantum is set to 4. Compute the average turnaround time when the round-robin scheduling strategy is adopted.

Job	A	B	C	D	E
Arrival Time	0	1	2	5	6
Running Time	8	4	10	2	3