Scheduling Criteria

CPU utilization: minimize overhead

throughput: processes completed per unit time

turnaround time: turnaround time is measured from creation time to termination time

response time: response time is measured from creation time to the time of first output

fairness

The Nature of Program Executions

- A running threads can be modeled as alternating series of *CPU bursts* and *I/O bursts*
  - during a CPU burst, a thread is executing instructions
  - during an I/O burst, a thread is waiting for an I/O operation to be performed and is not executing instructions

Preemptive vs. Non-Preemptive

- A *non-preemptive* scheduler runs only when the running thread gives up the processor through its own actions, e.g.,
  - the thread terminates
  - the thread blocks because of an I/O or synchronization operation
  - the thread performs a Yield system call (if one is provided by the operating system)
- A *preemptive* scheduler may, in addition, force a running thread to stop running
  - typically, a premptive scheduler will be invoked periodically by a timer interrupt handler, as well as in the circumstances listed above
  - a running thread that is preempted is moved to the ready state

FCFS and Round-Robin Scheduling

First-Come, First-Served (FCFS):

- non-premptive - each thread runs until it blocks or terminates
- FIFO ready queue

Round-Robin:

- preemptive version of FCFS
- running thread is preempted after a fixed time quantum, if it has not already blocked
- preempted thread goes to the end of the FIFO ready queue

Shortest Job First (SJF) Scheduling

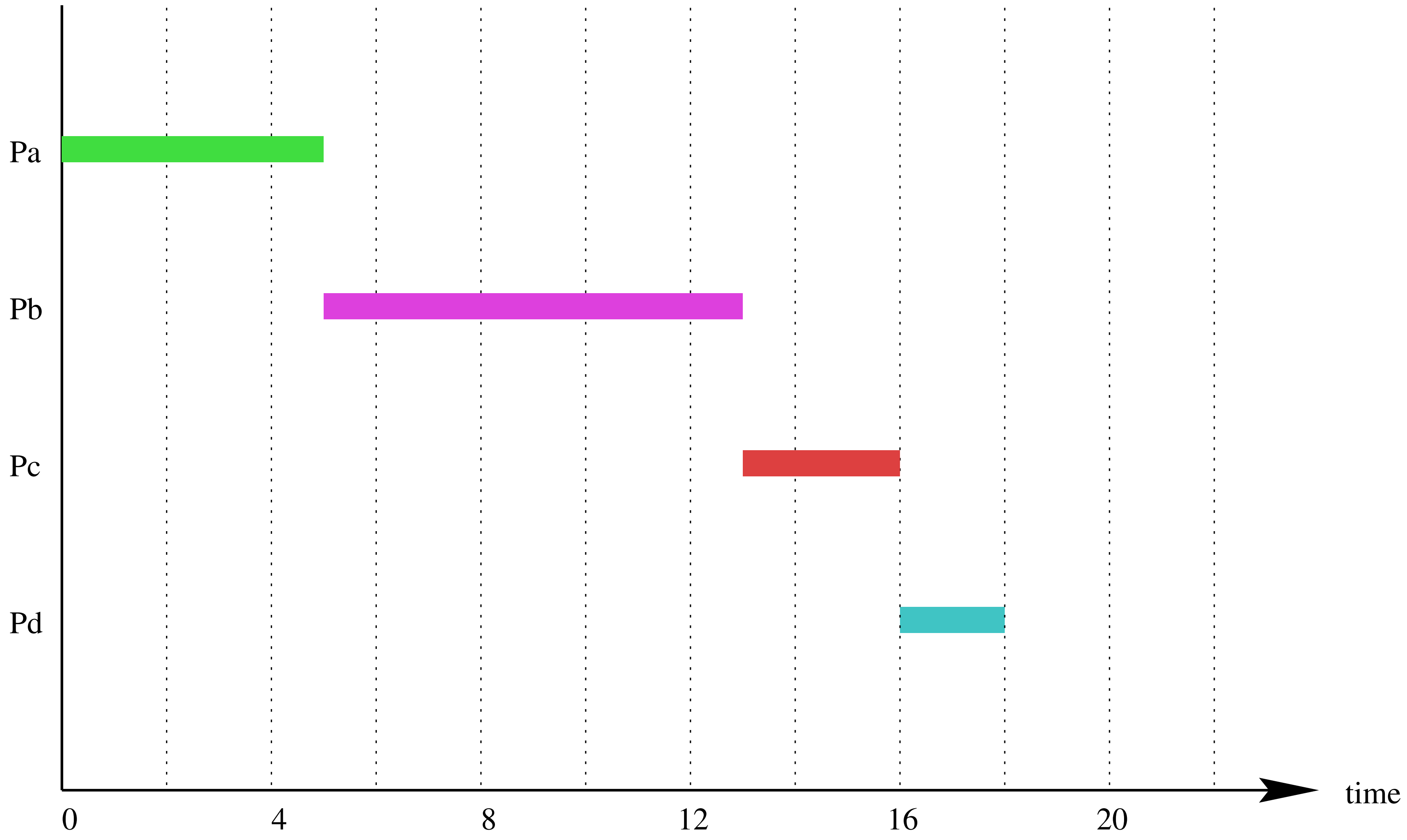
- non-preemptive
- ready threads are scheduled according to the length of their next CPU burst - thread with the shortest burst goes first
- SJF minimizes average waiting time, but can lead to starvation
- SJF requires knowledge of CPU burst lengths
  - Simplest approach is to estimate next burst length of each thread based on previous burst length(s). For example, exponential average considers all previous burst lengths, but weights recent ones most heavily:

$$B_{i+1} = \alpha b_i + (1 - \alpha) B_i$$

    where B_i is the predicted length of the ith CPU burst, and b_i is its actual length, and $0 \leq \alpha \leq 1$.

- Shortest Remaining Time First is a preemptive variant of SJF. Preemption may occur when a new thread enters the ready queue.

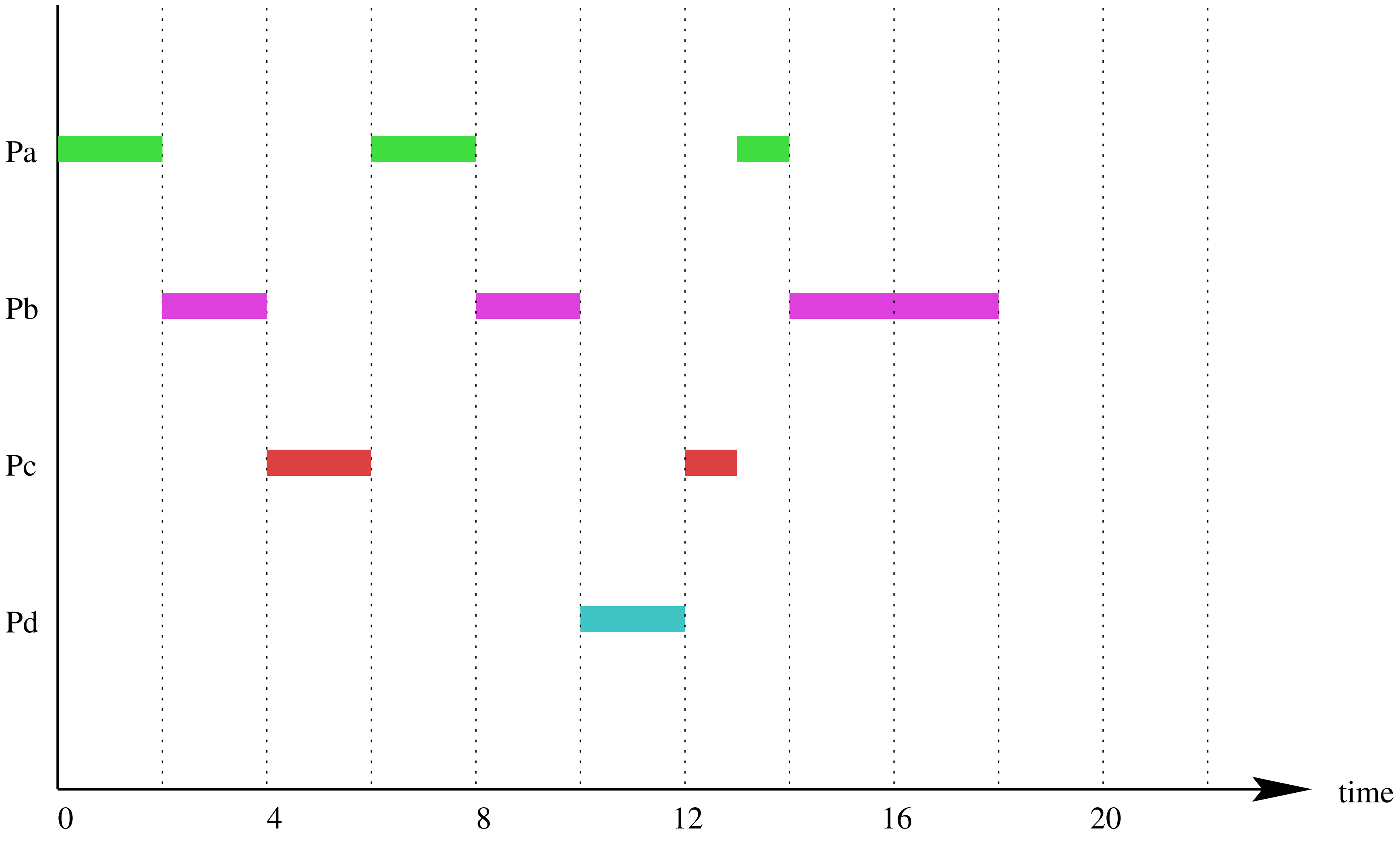
FCFS Gantt Chart Example



Initial ready queue: Pa = 5 Pb = 8 Pc = 3

Thread Pd (=2) "arrives" at time 5

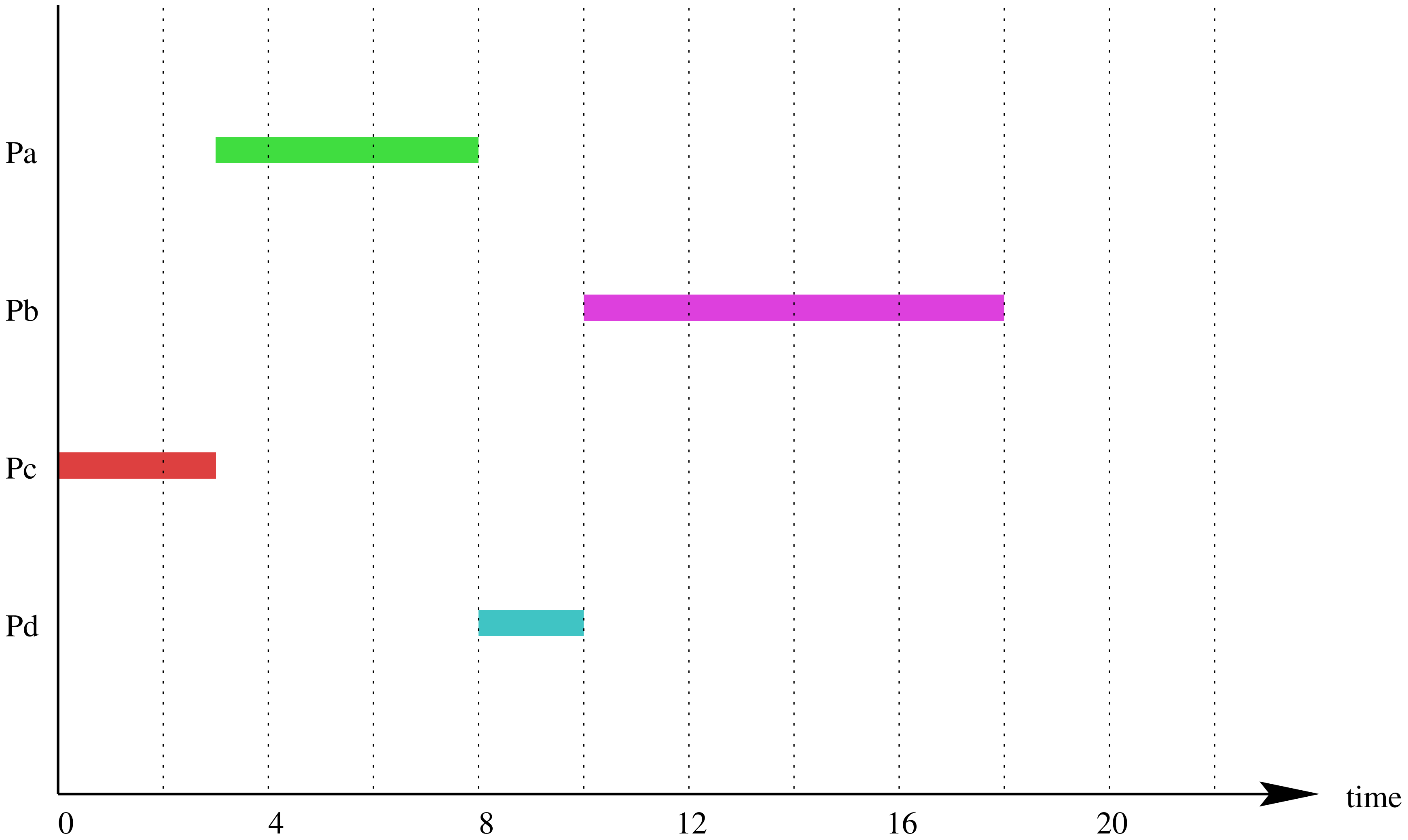
Round Robin Example



Initial ready queue: Pa = 5 Pb = 8 Pc = 3 Quantum = 2

Thread Pd (=2) "arrives" at time 5

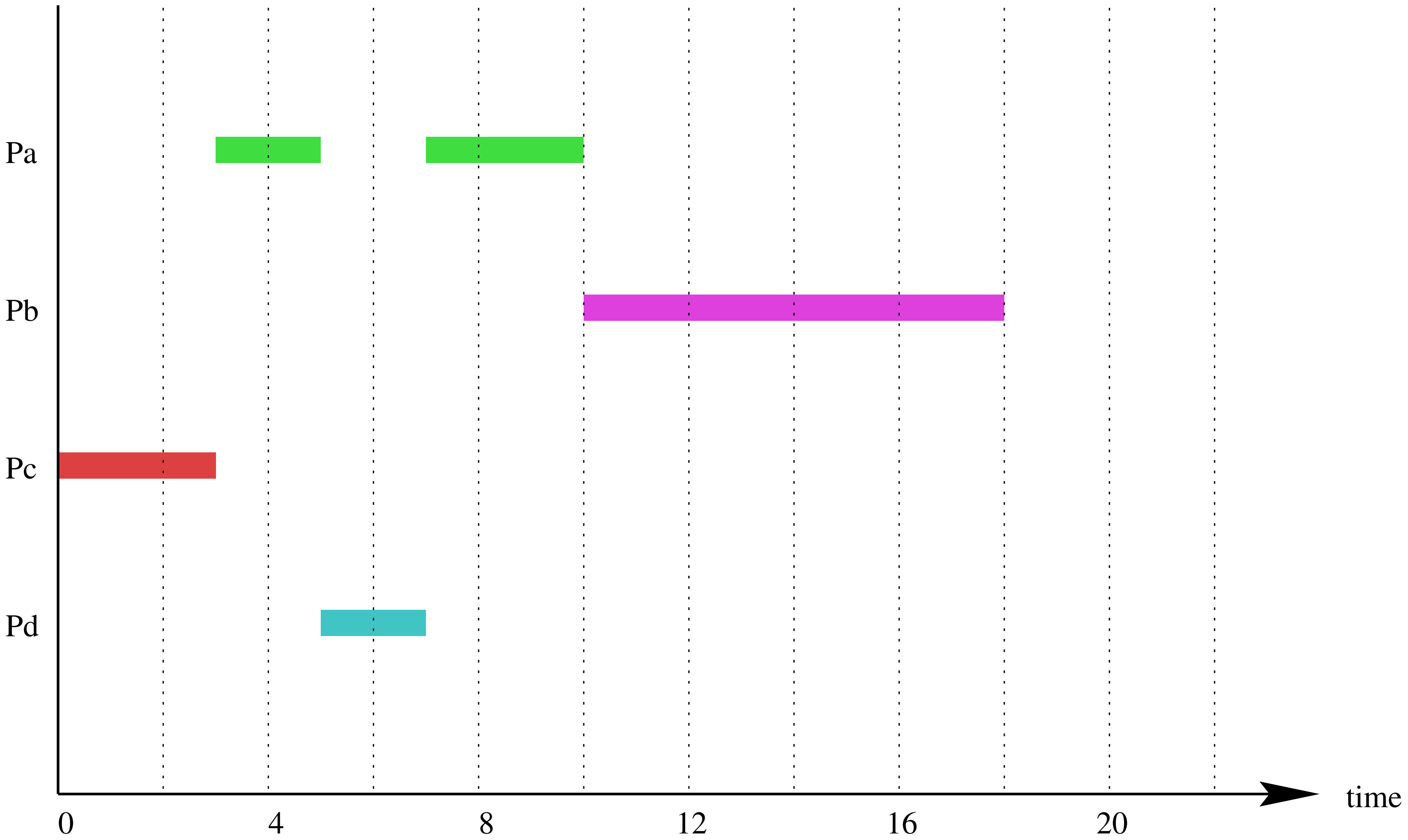
SJF Example



Initial ready queue: Pa = 5 Pb = 8 Pc = 3

Thread Pd (=2) "arrives" at time 5

SRTF Example



Initial ready queue: Pa = 5 Pb = 8 Pc = 3

Thread Pd (=2) "arrives" at time 5

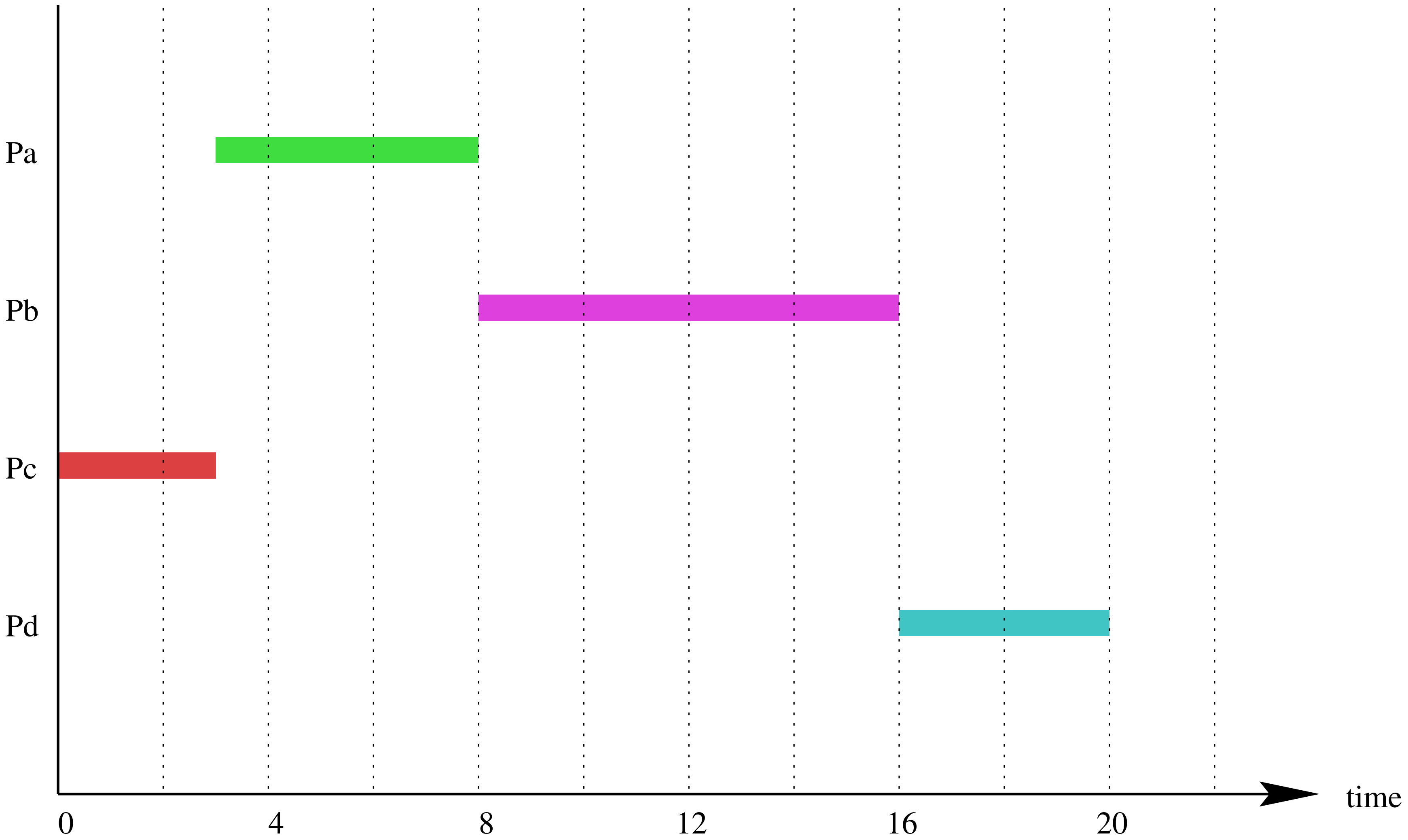
Highest Response Ratio Next

- non-preemptive
- response ratio is defined for each ready thread as:

$$\frac{w + b}{b}$$

  where b is the CPU burst time and w is waiting time
- scheduler chooses the thread with the highest response ratio (choose smallest b in case of a tie)
- HRRN is an example of a heuristic that blends SJF and FCFS

HRRN Example



Initial ready queue: Pa = 5 Pb = 8 Pc = 3

Thread Pd (=4) "arrives" at time 5

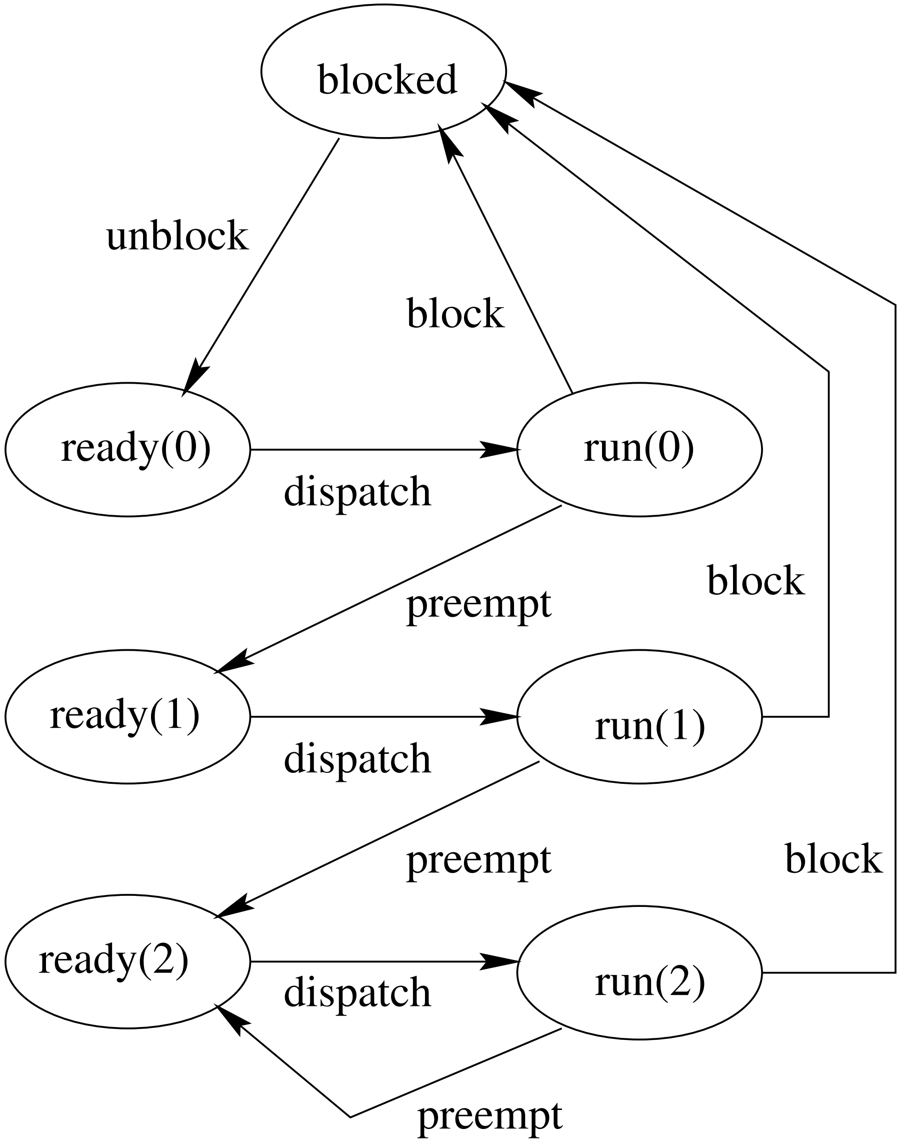
Prioritization

- a scheduler may be asked to take process or thread priorities into account
- for example, priorities could be based on
  - user classification
  - application classification
  - application specification (e.g., NachOS `SetPriority`)
- scheduler can:
  - always choose higher priority threads over lower priority thread
  - use any scheduling heuristic to schedule threads of equal priority
- low priority threads risk starvation. If this is not desired, scheduler must have a mechanism for elevating the priority of low priority threads that have waited a long time

Multilevel Feedback Queues

- gives priority to interactive threads (those with short CPU bursts)
- scheduler maintains several ready queues
- scheduler never chooses a thread in queue i if there are threads in any queue $j < i$.
- threads in queue i use quantum q_i, and $q_i < q_j$ if $i < j$
- newly ready threads go in to queue 0
- a level i thread that is preempted goes into the level $i + 1$ ready queue

3 Level Feedback Queue State Diagram



Lottery Scheduling

- randomized proportional share resource allocation
- resource rights represented by lottery tickets, allocation determined by lottery
  - resource granted to holder of winning ticket
- probabilistically fair with $p = t/T$
  - p = probability of allocation, t = tickets held, T = total tickets
  - avoid starvation by ensuring $t > 0$
- uniformly-distributed pseudo-random number generator (10 lines on MIPS)
- can proportionally assign other resources (e.g., memory, bandwidth)
- "Lottery Scheduling: Flexible Proportional-Share Resource Management", Waldspurger and Weihl, Operating System Design and Implementation, 1994.

Processor Scheduling Summary

FCFS:

+ simple, low overhead
+ no starvation
- can give poor response times for interactive processes

RR:

+ no starvation
+ reduced waiting time variance
+ good response times for interactive processes

SJF and SRTF:

+ best response times
- depends on burst length estimates
- starvation is possible

Processor Scheduling Summary

HRRN:

- \+ no starvation
- \+ good response times
- \- depends on burst length estimates

Multilevel Feedback Queues:

- \+ flexible
- \+ good response times for interactive processes
- \- depends on burst length estimates
- \- compute-intensive processes can starve

Other Scheduling Issues

short term scheduling: what has been covered so far

medium term scheduling: suspension/resumption of partially executed processes

- usually because a resource, especially memory, is overloaded
- suspended process releases resources
- operating system may also provide mechanisms for applications or users to request suspension/resumption of processes

long term scheduling: process admission control to limit the degree of multiprogramming

Scheduling States Including Suspend/Resume