PURDUE
UNIVERSITY®

CS 25200: Systems Programming

Lecture 19: Processes and Scheduling

Prof. Turkstra

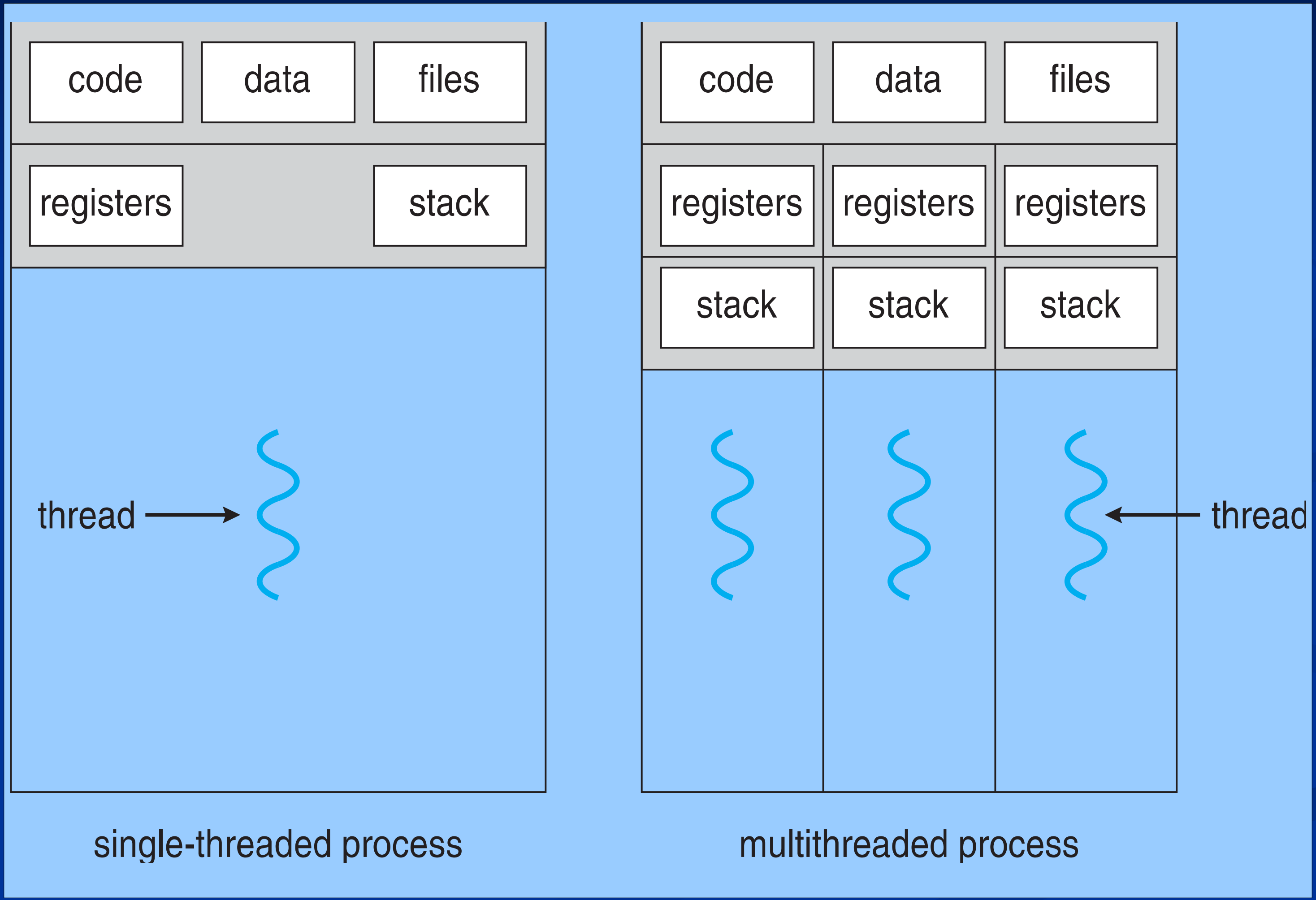
Lecture 19

- Threads and context switches
- Non-preemptive vs preemptive scheduling
- FCFS scheduling
- Round robin scheduling
- Multilevel feedback queue scheduling

Threads

- Process includes…
  - Address space (code, data, etc)
  - Resource container (OS resource, accounting)
  - A “thread” of control – PC, regs, stack
- Threads
  - Share some code and data (address space)
  - Same files, I/O channels, resource containers
  - Do not share thread of control

Threads


code
data
files
registers
stack
thread
code
data
files
registers
registers
registers
stack
stack
stack
thread
single-threaded process
multithreaded process

Threads

- Can have several threads in a single address space
- Threads are units of scheduling
- Processes are containers in which threads execute

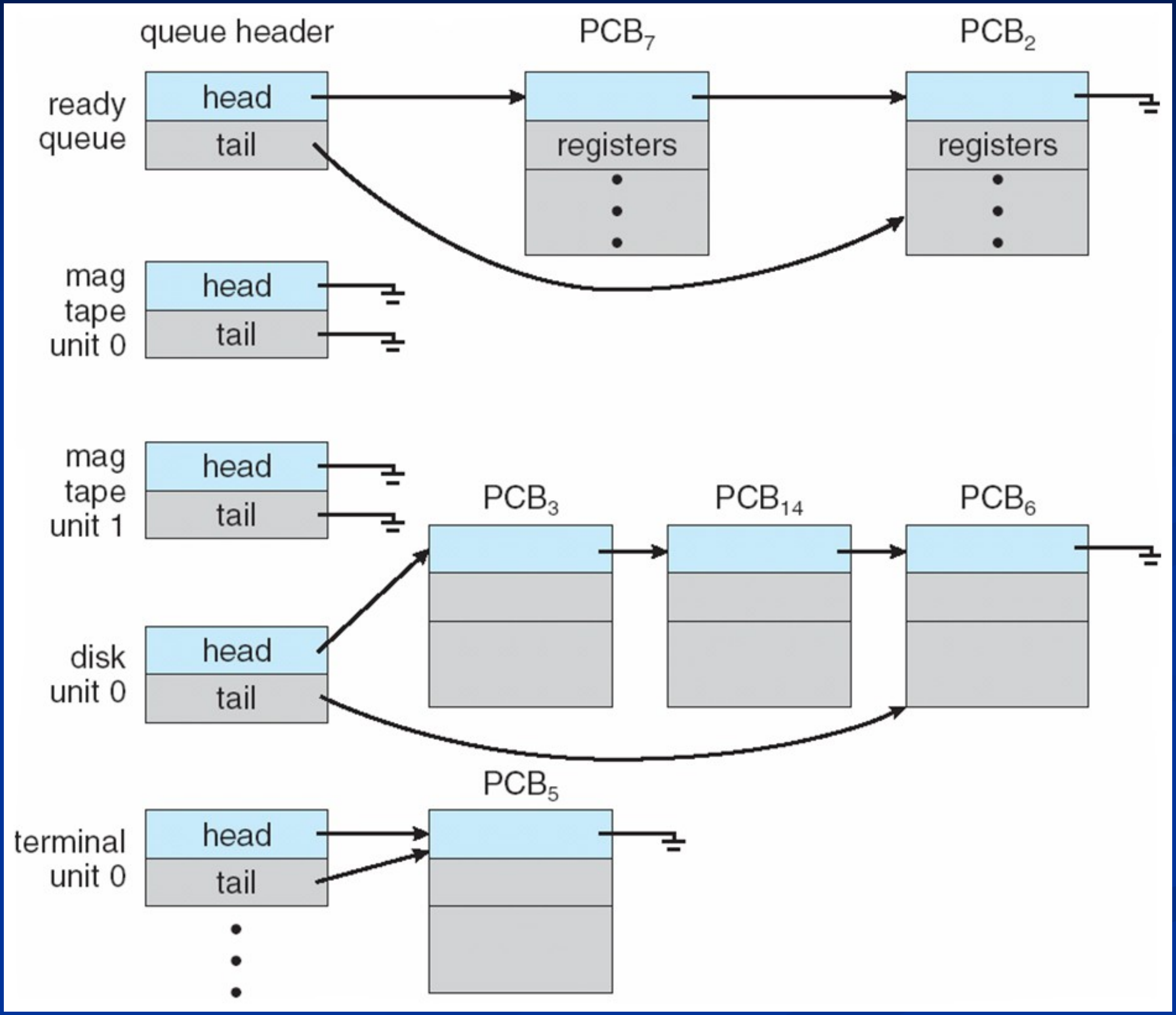
Threading

- Userland threads
  - POSIX Pthreads (IEEE 1003.1c)
  - Mach C-threads
  - Solaris threads
  - Windows threads
  - Java threads
- Kernel threads
  - Solaris LWP
  - Linux tasks (clone())
  - etc

Context switches

- TCB – Thread Control Block
  - Shared – parent process, execution time, memory, I/O resources
  - Private – PC, registers, stack, state information, pending/blocked signals
- TCB can be managed almost entirely in userland
  - Lower context switch overhead
- …or may rely on the kernel in some way

Process queues



Process scheduling

- From user standpoint, OS permits many processes executing simultaneously
- In reality, OS switches among processes rapidly to give the illusion of simultaneity

Scheduler

- Operating System subsystem that is responsible for determining which process(es) to run, for how long, and when
- Two types: non-preemptive and preemptive

Non-preemptive

- Context switches happen only when the running process waits or yields
- Also called cooperative multitasking
- Used in Windows 3.1 and initial versions of MacOS

Preemptive

- Context switches can be forced
  - Usually after a fixed period of time, called a quantum
  - E.g., every 1/100sec
- Rely a timer interrupt that invokes the OS scheduler
  - Often the process that has been in the ready state the longest will execute next
- Implemented in *NIX, Windows 95 and above, etc

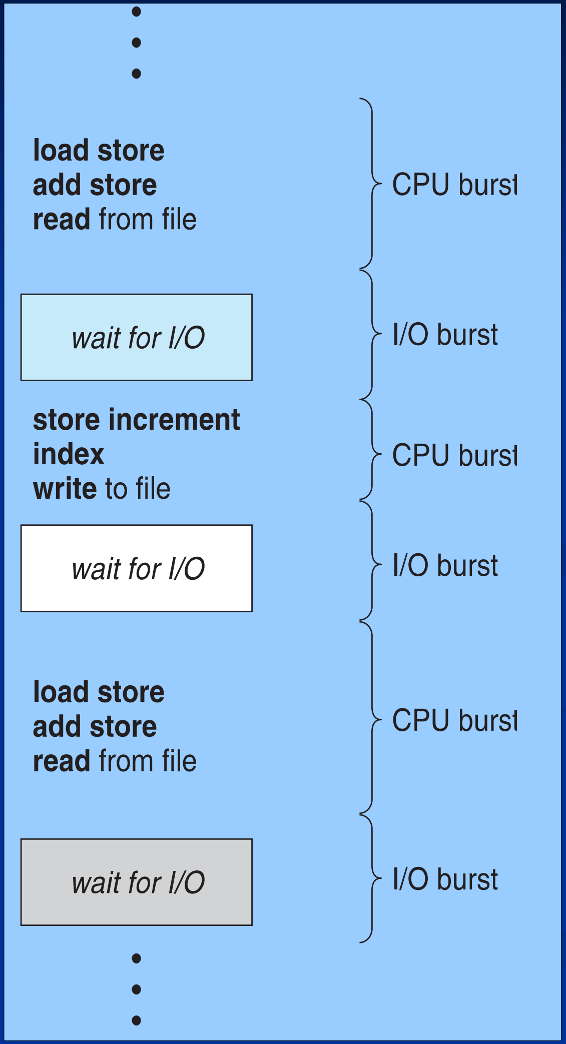
Tradeoffs

- Non-preemptive
  - More (user) control over how the CPU is used
  - Simple
- Preemptive
  - More robust
  - Enforced fairness

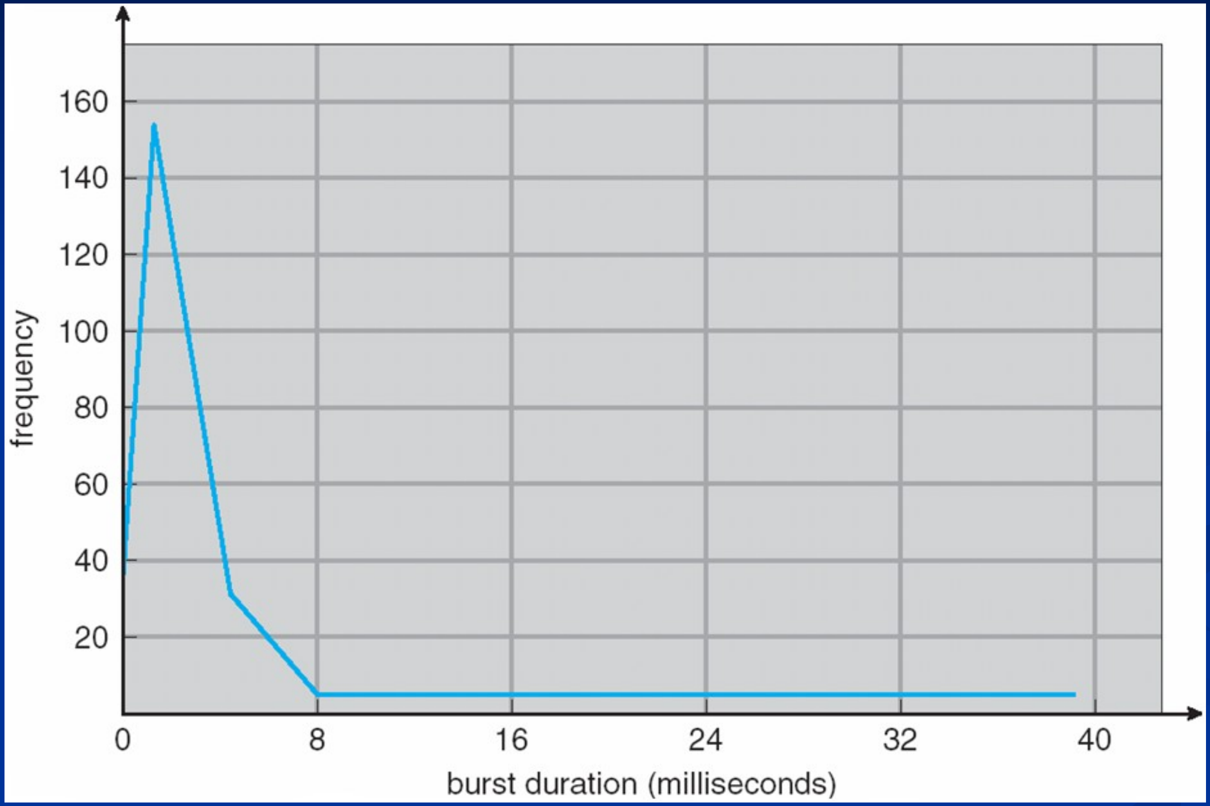
Preemptive scheduling

- Many different algorithms
  - FIFO, fixed priority, round-robin, multilevel queue, shortest remaining time first, worst case execution time, virtual round robin, etc

Execution cycle



CPU burst times



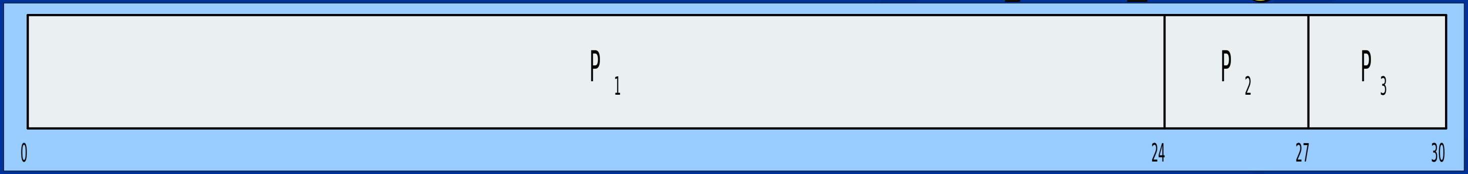
- 90% of CPU bursts are less than 10ms

First come first served (FCFS)

- Also known as FIFO

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Processes arrive in order (P_1, P_2, P_3)



- Waiting times: $P_1 = 0$, $P_2 = 24$, $P_3 = 27$
- Average wait time: $(0 + 24 + 27) / 3 = 17$

FCFS

- Suppose processes arrive:

  P_2, P_3, P_1

- Waiting times: $P_1 = 6$, $P_2 = 0$, $P_3 = 3$
- Average waiting time:

  $(6 + 0 + 3) / 3 = 3$

- Convoy effect – short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

Purdue Trivia

- Purdue’s “Big Bass Drum” (or BBD) was commissioned in 1921 by Paul Spotts Emrick
  - Leedy Manufacturing Company
- Ford Model T back axle and wheelbase

Round robin

- Each process gets a small unit of CPU time (a quantum)
  - Usually 10-100 milliseconds
- Quantum expires → preempt and move to end of the ready queue
  - Switch to the next process

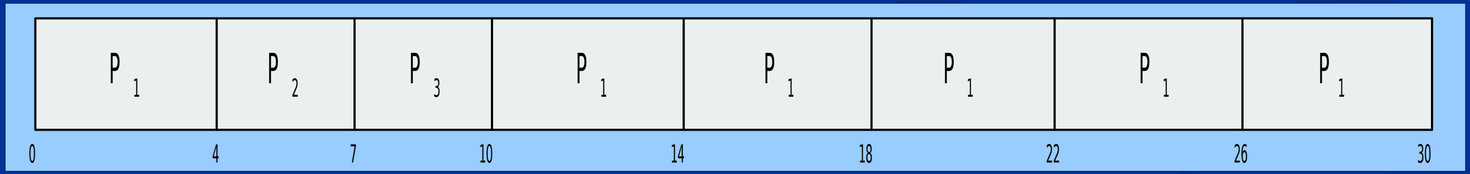
Performance

- n processes in the ready queue
- time quantum q
- Each process will get 1/n of the CPU time in chunks of at most q time units at once
- → Maximum wait time = (n – 1) * q
- What happens when q is large?
- What happens when it is small?

Round robin example

- Quantum = 4 ms

Process	Burst Time
P_1	24
P_2	3
P_3	3



- Average turn around time:

  (30 + 7 + 10) / 3 = 15.7 ms

Round robin example

- Quantum = 1 ms

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Average turn around time:

  (8 + 9 + 29) / 3 = 15.3 ms

Quantum leap

- Shorter time quanta appear to reduce the average completion time
- What is the catch?
- Context switch overhead!

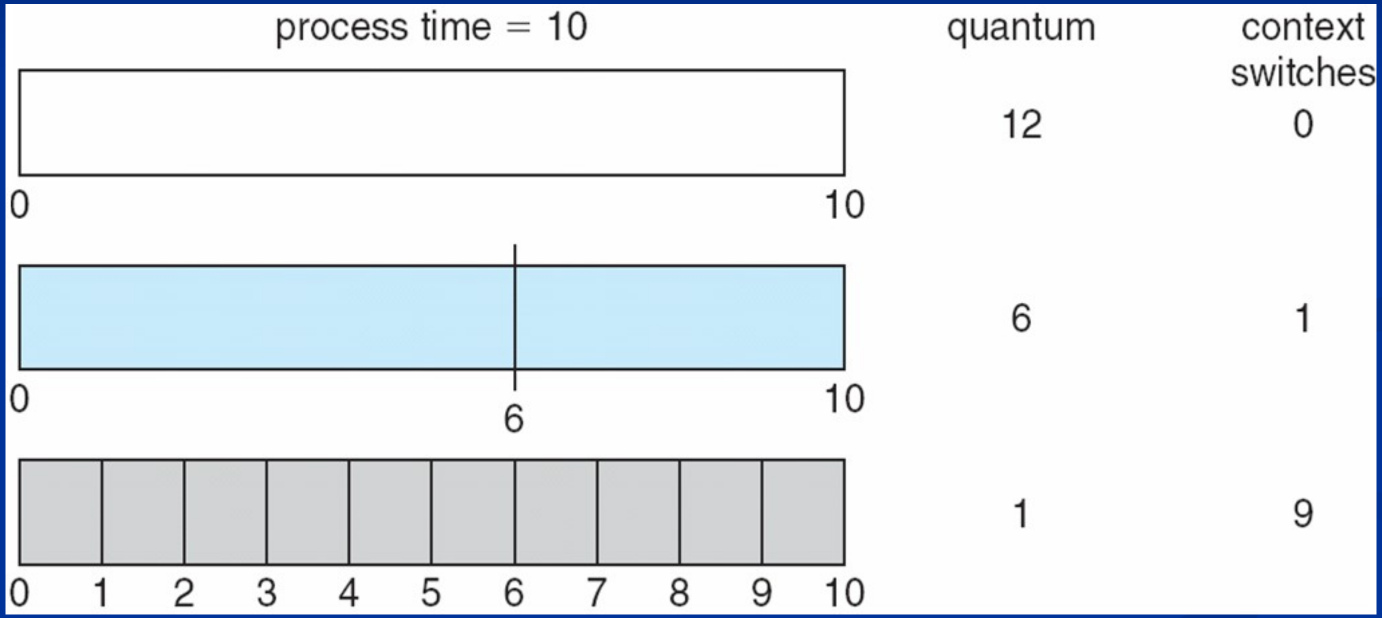
$$\%\text{overhead} = 100 * c / q$$

$$c = \text{context switch overhead}$$

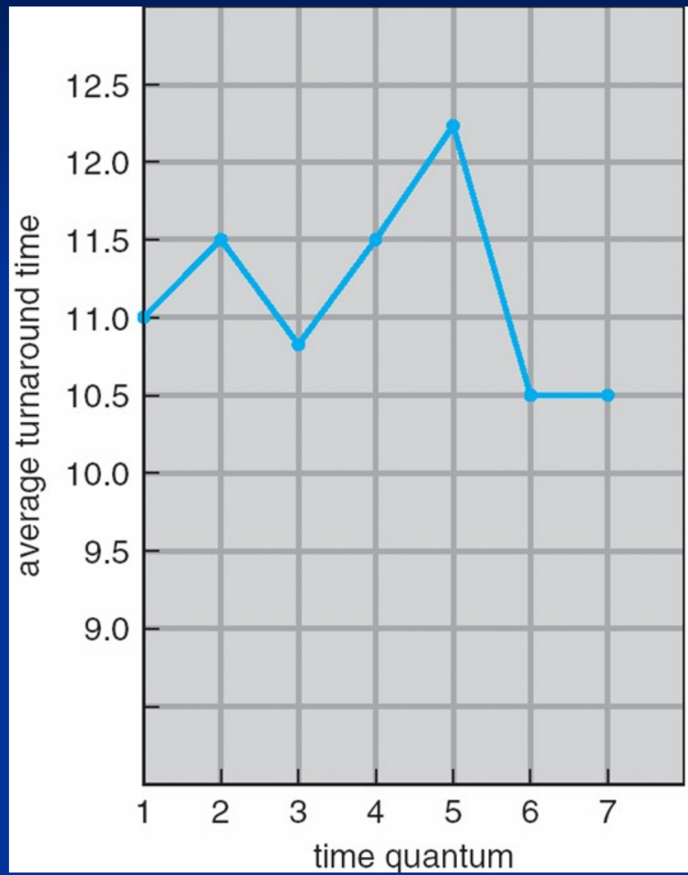
- Suppose c = 0.1ms

$$q = 10 \rightarrow 1\%$$
$$q = 2 \rightarrow 5\%$$
$$q = 0.2 \rightarrow 50\%$$

Quantum and context switches



Turnaround time



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Choosing a quantum

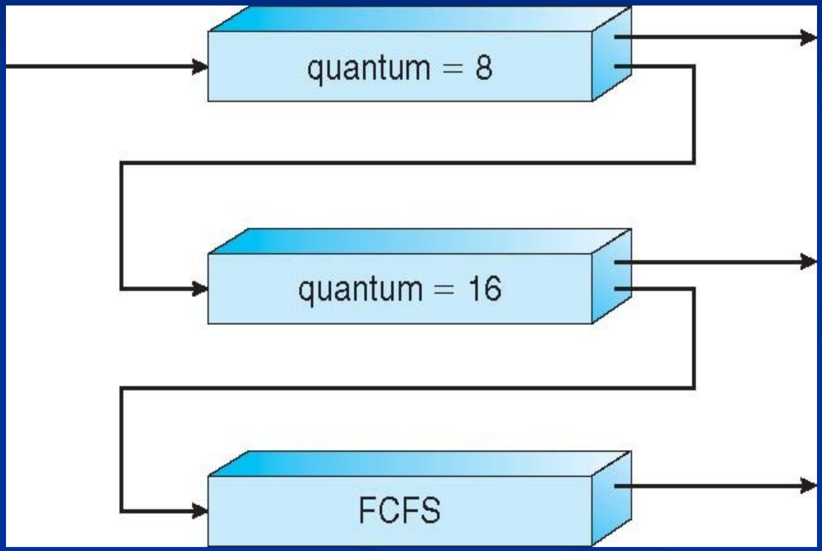
- Small enough to have a reasonable response time
- Large enough to control context switch overhead (should be < 1%)
  - Shoot for at least 80% bursts < q
- 10ms is a good choice
  - 90% of bursts complete without preemption

Multilevel feedback queue

- Multiple queues, processes can move between them
- Aging: processes are moved to higher queue (increased priority) the longer they remain queued
- Reward interactive processes
  - Increase priority when a process yields before quantum expires
- Penalize CPU-bound processes

Example

- Suppose three queues:
  - Q_0: RR, q = 8ms
  - Q_1: RR, q = 16 ms
  - Q_2: FCFS
- Scheduling
  - Process enters Q_0
  - Runs > 8ms, moves to Q_1
  - In Q_1 runs > 16 ms, moves to Q_2



- Reverse can happen
  - Process in Q_2 completes quickly (or ages) elevated to Q_1
  - Completes in < 16ms, elevates to Q_1

Questions?