CPU Scheduling

Prof. Bracy and Van Renesse
CS 4410
Cornell University

based on slides designed by Prof. Sirer and Dr. De Bruijn

The Problem

- You're the cook at State Street Diner
  - customers continuously enter and place orders 24 hours a day
  - dishes take varying amounts to prepare
- What is your *goal*?
  - minimize average latency
  - minimize maximum latency
  - maximize throughput
  - …
- Which *strategy* achieves your goal?

Goals depend on Context

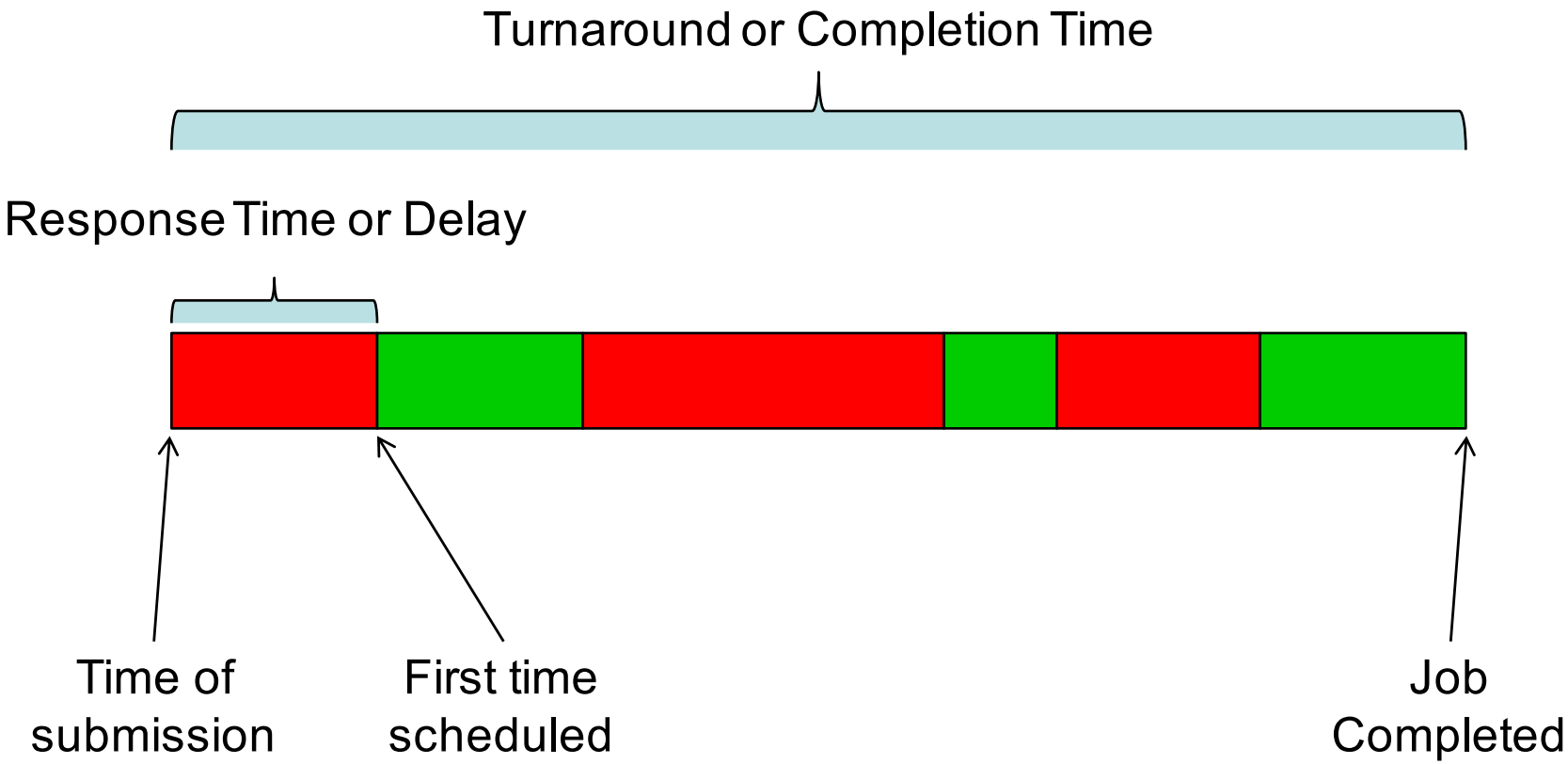
Suppose you:

- own an (expensive) container ship and have cargo across the world
- are a nurse who has to combine care and administration
- are a student who has to do homework in various classes, hang out with other students, and sleep occasionally

Schedulers

- Process/thread migrates among several queues
  - Device wait queue, run queue, …
- Various schedulers in O.S.:
  - CPU Scheduler selects a process to run from the run queue
  - Disk Scheduler selects next read/write operation
  - Network Scheduler selects next packet to send or process
  - Page Replacement Scheduler selects page to evict
  - …
- We'll focus here on *CPU scheduling*
- Which process to run?
  - no process to run: run idle loop or halt CPU until interrupt
  - one process to run: that's easy!
  - >one process ready: what to do?

Per Job or Task Metrics



Waiting Time: sum of “red” periods (time on run queue but not running)

Scheduling Evaluation Metrics

- Many quantitative criteria for scheduler algorithm:
  - CPU utilization: percentage of time the CPU is not idle
  - Throughput: completed processes per time unit
  - Turnaround time: submission to completion
  - Waiting time: time spent on the run queue
  - Response time: response latency
  - Predictability: variance in any of these measures

- The right metric depends on the context

An underlying assumption:

  - "response time" most important for interactive jobs (I/O bound)

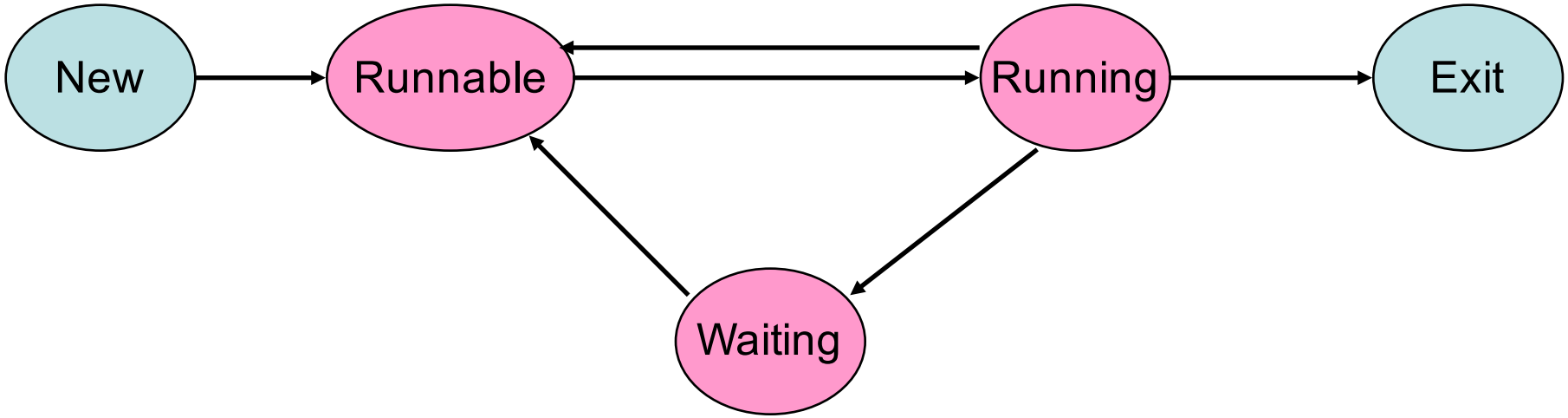
The Perfect Scheduler

- Minimizes latency: response or job completion time
- Maximizes throughput: maximize #jobs / time
- Maximizes utilization: keep all devices busy
- Meets deadlines: think car brakes, etc.
- Fairness: everyone makes progress, no one starves

No such scheduler exists!

Process Scheduling

- "process" and "thread" used interchangeably
- Many processes in may be in "runnable" state
  - aka "ready" state
  - These jobs are on the run queue (aka ready queue)



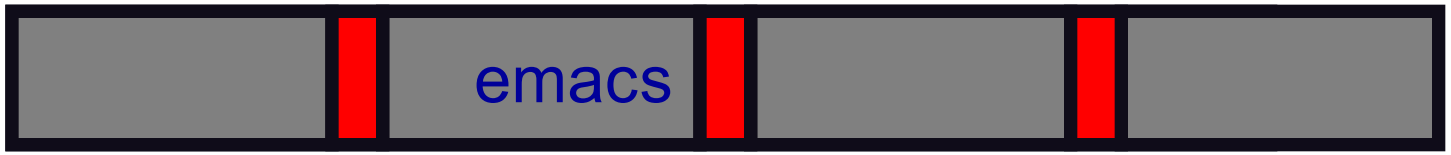
When does scheduler run?

- Non-preemptive
  - Process runs until voluntarily relinquish CPU
    - process blocks on an event (e.g., I/O or synchronization)
    - process yields
    - process terminates
- Preemptive
  - All of the above, plus:
    - Timer and other interrupts
    - Required when processes cannot be trusted to yield
    - Incurs some overhead

Process Model

- Process alternates between CPU and I/O bursts
  - CPU-bound jobs: Long CPU bursts

  - I/O-bound: Short CPU bursts



I/O burst cycle = processes cycle between using the CPU and waiting for I/O completion

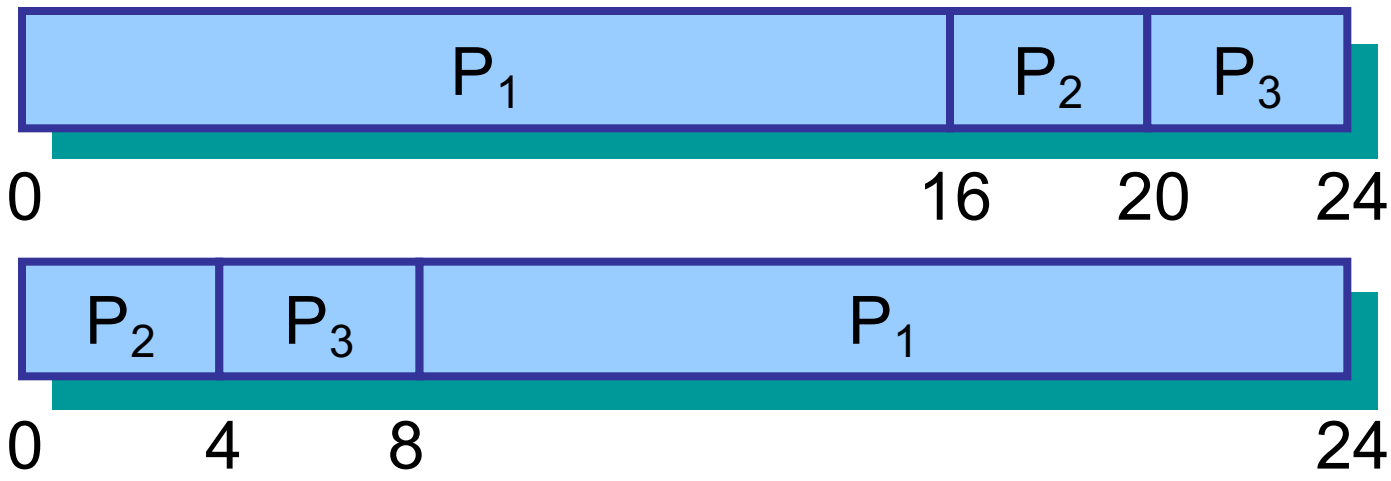
- Problem: don't know job's type before running
  - and jobs also change over time

Definition of “load”

- The load of a machine is the number of processes that are either runnable or running.
- Typically averaged over a period using exponential smoothing
  - Coming up later

Scheduling Algorithms FCFS

- **First-come First-served (FCFS) (FIFO)**
  - Jobs are scheduled in order of arrival
  - Non-preemptive
- **Problem:**
  - Average waiting time depends on arrival order



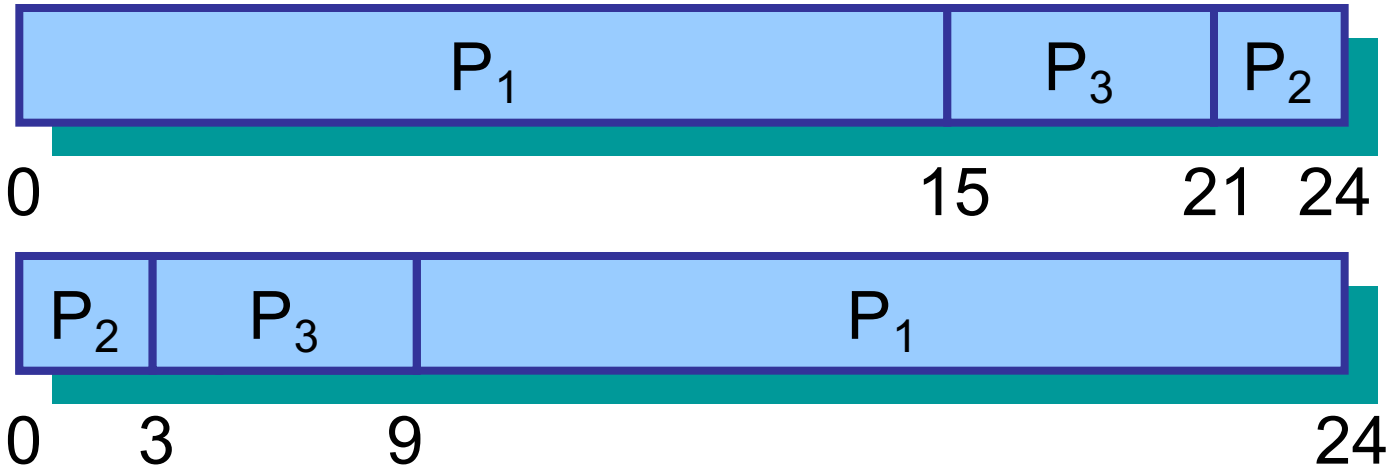
- **Advantage:** really simple!

Scheduling Algorithms LIFO

- **Last-In First-out (LIFO)**
  - Newly arrived jobs are placed at head of ready queue
  - Improves response time for newly created threads
- **Problem:**
  - May lead to starvation – early processes may never get CPU

Scheduling Algorithms: SJF

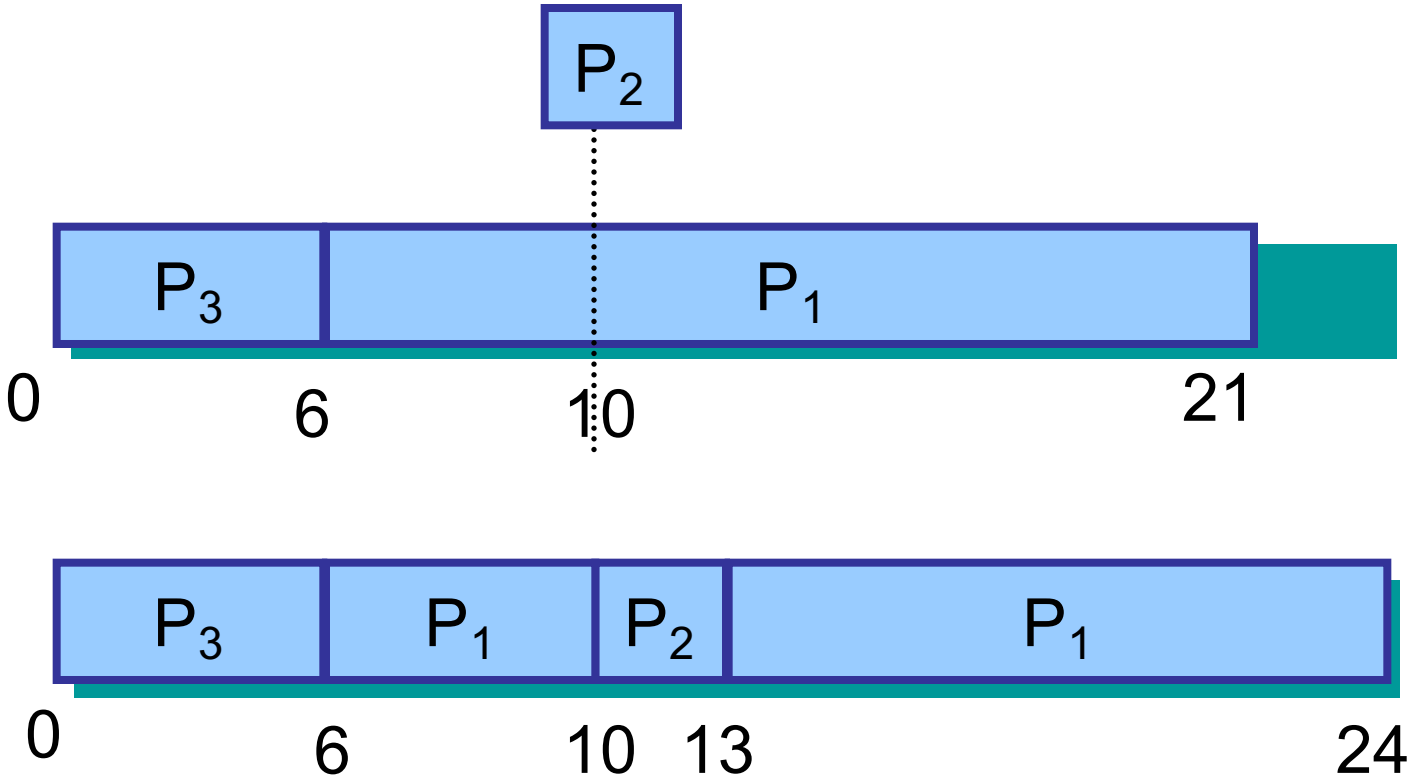
- **Shortest Job First (SJF)**
  - Choose the job with the shortest next CPU burst
  - Provably optimal for minimizing average waiting time



- **Problem:**
  - Impossible to know the length of the next CPU burst
  - Starvation…

Scheduling Algorithms SRTF

- SJF can be either preemptive or non-preemptive
  - New, short job arrives; current process has long time to execute
- Preemptive SJF is called *shortest remaining time first*



Shortest Job First Prediction

- Approximate next CPU-burst duration
  - from the durations of the previous bursts
    - The past can be a good predictor of the future
- No need to remember entire past history
- Use exponential average:

  t_n duration of the n^{th} CPU burst

  τ_{n+1} predicted duration of the $(n+1)^{st}$ CPU burst

  $$\tau_{n+1} = \alpha\tau_n + (1- \alpha)\, t_n$$

  where $0 \leq \alpha \leq 1$

  α determines the weight placed on past behavior

Priority Scheduling

- **Priority Scheduling**
  - Choose next job based on priority
  - Can be either preemptive or non-preemptive
- Priority schedulers can emulate other algorithms:
  - **P == arrival time → FIFO**
  - **P == now – arrival time → LIFO**
  - **P == job length → SJF**
  - **P == remaining job length → SRTF**
- **Solution to starvation**
  - Age processes: increase priority as a function of waiting time

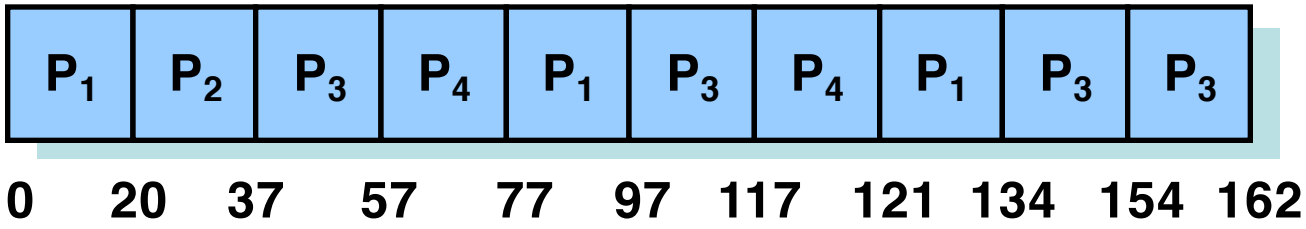
Round Robin

- **Round Robin (RR)**
  - First-Come-First-Served + Preemption
  - Often used for timesharing
  - Run queue is treated as a circular queue (FIFO)
  - Each process is given a time slice called a *quantum*
  - It is run for the quantum or until it blocks
  - RR allocates the CPU uniformly (fairly) across participants.
  - If average queue length is n, each participant gets 1/n

RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The timing diagram is:



- Higher average turnaround than SJF,
- But better response time

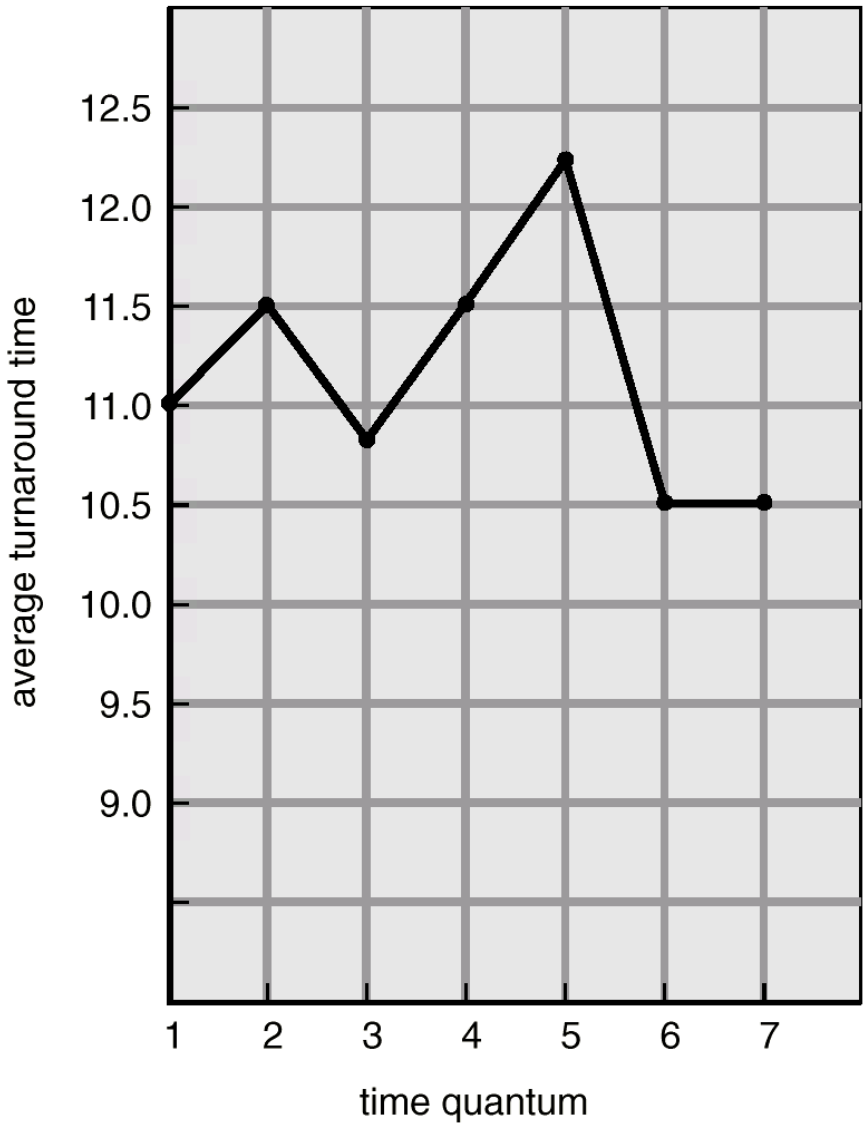
Choice of Time Quantum

Performance depends on length of the quantum

- Context switching isn't a free operation.
- If quantum is set too high
  - starts to approximate FCFS
  - bad interactive response
- If it's set too low
  - you're spending all of your time context switching between threads.
- Quantum frequently set to ~10-100 milliseconds
- Context switches typically cost << 1 millisecond

Moral:
Context switch is usually negligible (< 1% per quantum)
unless you context switch too frequently and lose all productivity

Turnaround Time w/ Time Quanta



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Problem Revisited

- Cook at State Street Diner
  - which algorithm would you use to minimize the average time that customers wait for their food?
  - note: most restaurants use FCFS

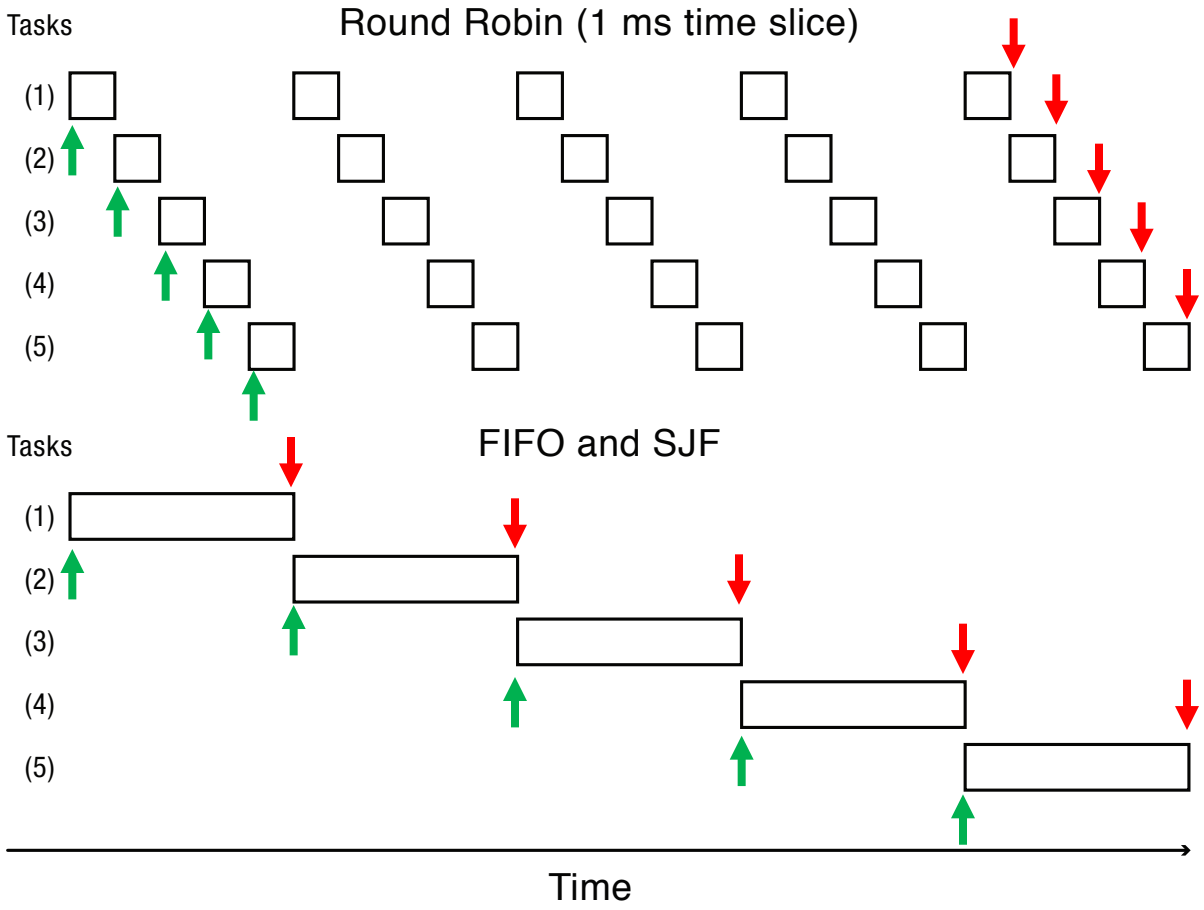
Problems with Round Robin

Tasks of same length that start ~same time

+ response time

– completion time

→ *RR seems like a lot of overhead for not much benefit…*

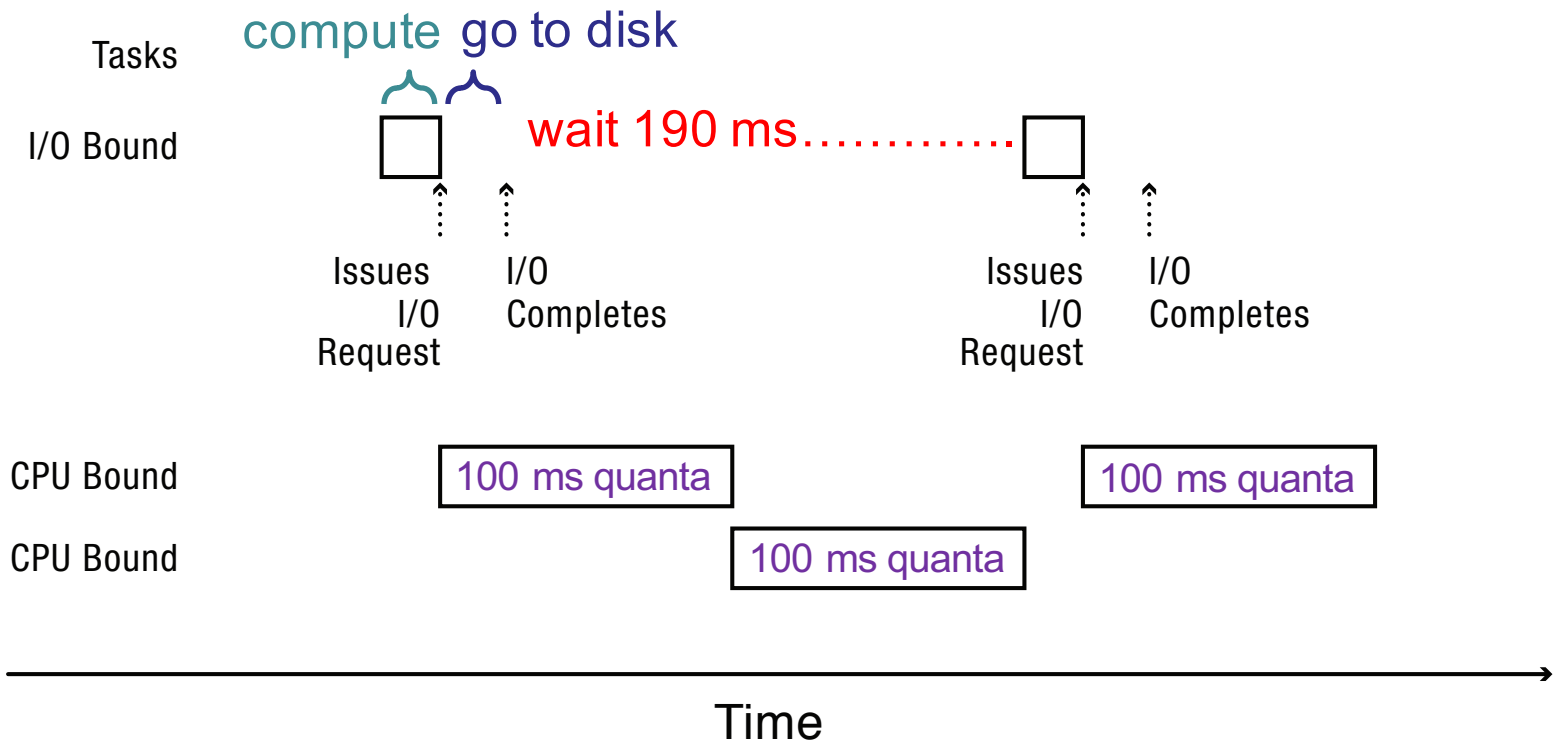


Problems with Round Robin

Mixture of one I/O Bound tasks + two CPU Bound Tasks

I/O bound: compute, go to disk, repeat

→ *RR doesn't seem so fair after all….*



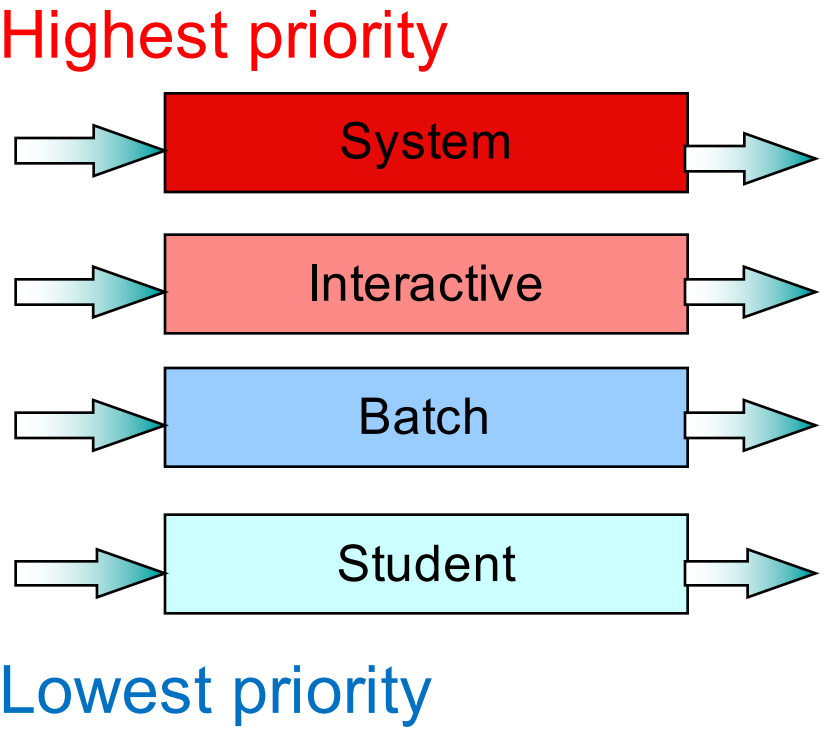
Multi-Level Queue Scheduling

- Multiple ready queues based on job "type"
  - interactive processes
  - CPU-bound processes
  - batch jobs
  - system processes
  - student programs
- Different queues may be scheduled using different algorithms

– Queue classification difficult

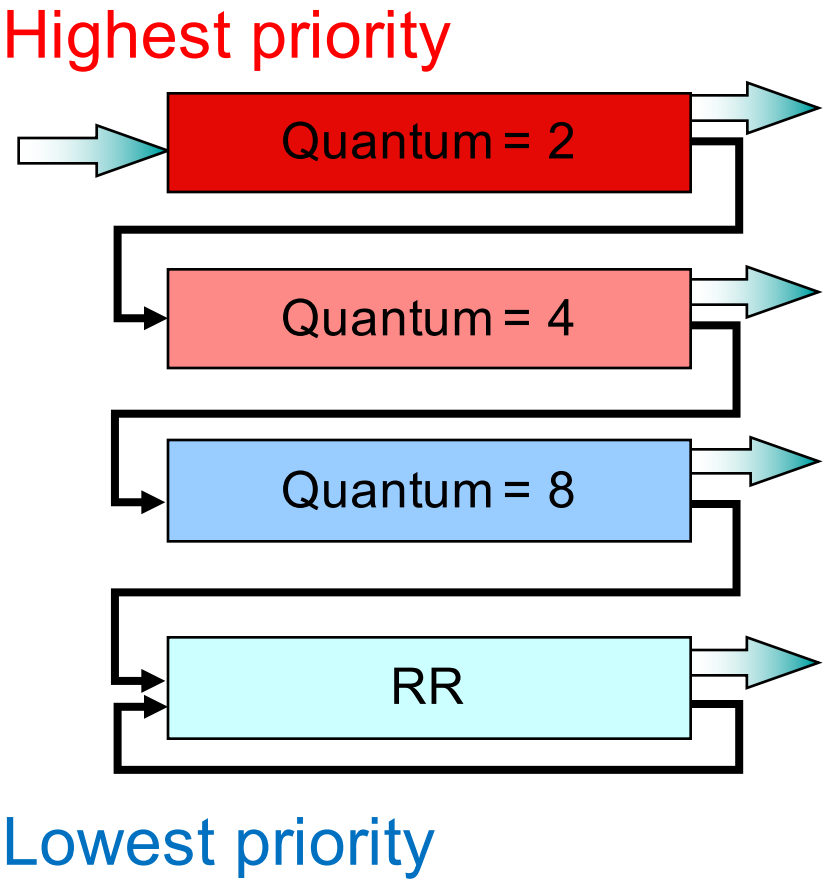
  Process may have CPU-bound and interactive phases

– No queue re-classification

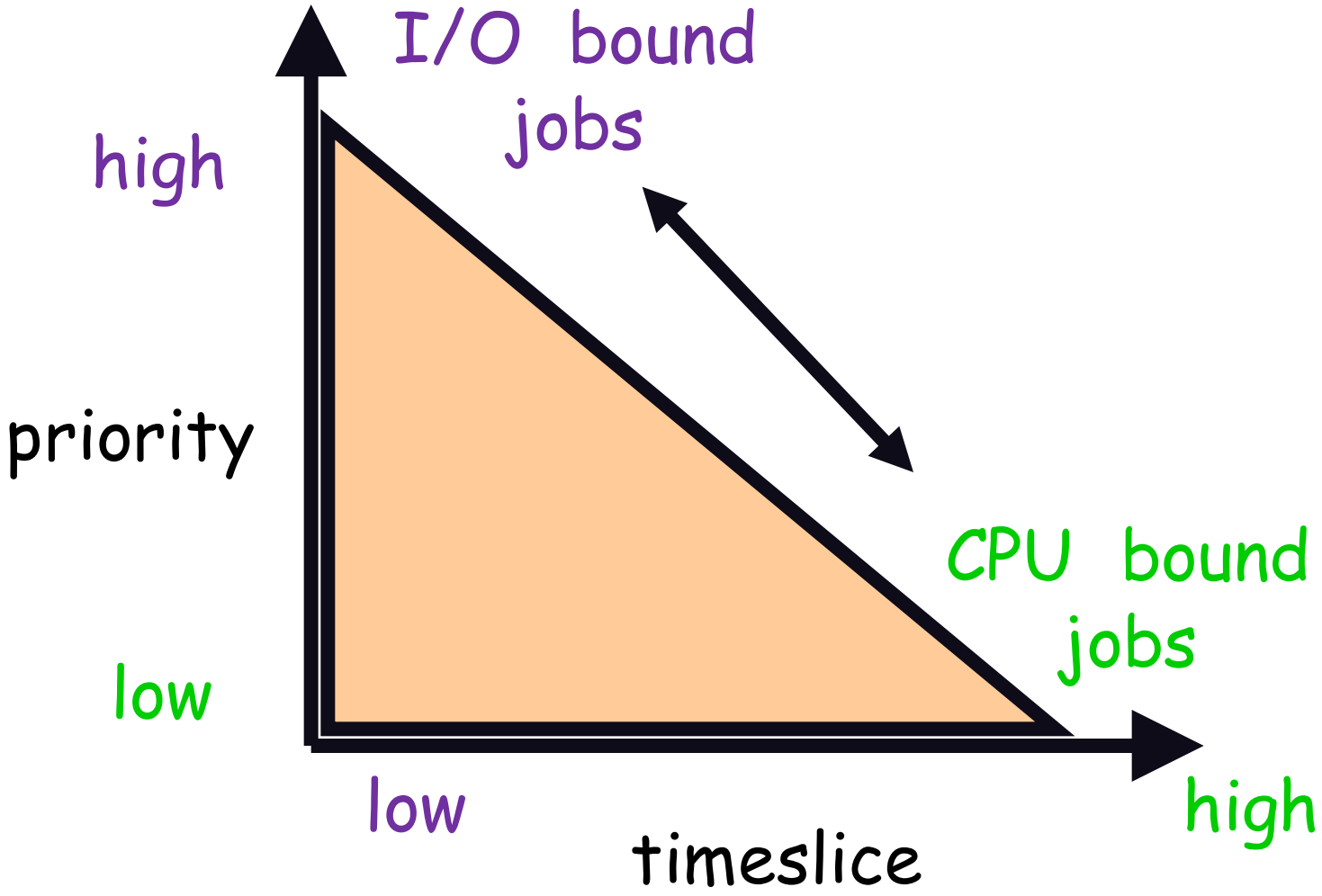


Multi-Level Feedback Queues

- Like multilevel queue scheduling, but assignments are not static
- Jobs move from queue to queue based on feedback
  - does job require the full quantum for computation
  - does job perform frequent I/O
- Need parameters for:
  - Number of queues
  - Scheduling algorithm per queue
  - When to upgrade/downgrade job



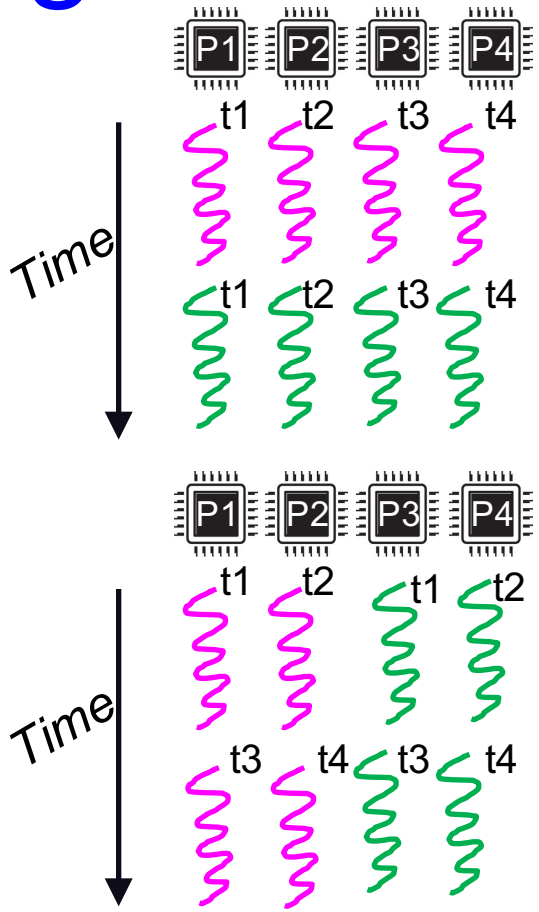
A Multi-level System



Thread Scheduling

Threads share code & data segments

- **Option 1: Ignore this**
- **Option 2: Gang scheduling***
  - all threads of a process run together (pink, green)
  - \+ Need to synchronize? Other thread is available
- **Option 3: Space-based affinity***
  - assign tasks to processors (pink → P1, P2)
  - \+ Improve cache hit ratio
- **Option 4: Two-level scheduling**
  - schedule processes, and within each process, schedule threads
  - \+ Reduce context switching overhead and improve cache hit ratio



***multiprocessor only**

Real-time Scheduling

- Real-time processes have timing constraints
  - Expressed as deadlines or rate requirements
- Common RT scheduling policies
  - Earliest deadline first (EDF) (priority = deadline)
    - Task A: I/O (1ms compute + 10 ms I/O), deadline = 12 ms
    - Task B: compute, deadline = 10 ms
  - Priority Donation
    - High priority task (needing lock) donates priority to lower priority task (with lock)