# CPU Scheduling

CS 4410
Operating Systems

[R. Agarwal, L. Alvisi, A. Bracy, M. George, E. Sirer, R. Van Renesse]

# 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

What if instead you are:

- the owner of an (expensive) container ship and have cargo across the world
- the head nurse managing the waiting room of the emergency room
- a student who has to do homework in various classes, hang out with other students, eat, and occasionally sleep

# Schedulers in the OS

- 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 on **CPU Scheduling**

# Kernel Operation (conceptual, simplified)

1. Initialize devices
2. Initialize "first process"
3. `while (TRUE) {`
   - while device interrupts pending
     - handle device interrupts
   - while system calls pending
     - handle system calls
   - **if run queue is non-empty**
     - **select process and switch to it**
   - otherwise
     - wait for device interrupt

`}`

# Performance Terminology

**Task/Job**

- User request: e.g., mouse click, web request, shell command, …

**Response time (latency, delay):** How long?

- User-perceived time to do some task.

**Initial waiting time:** When do I start?

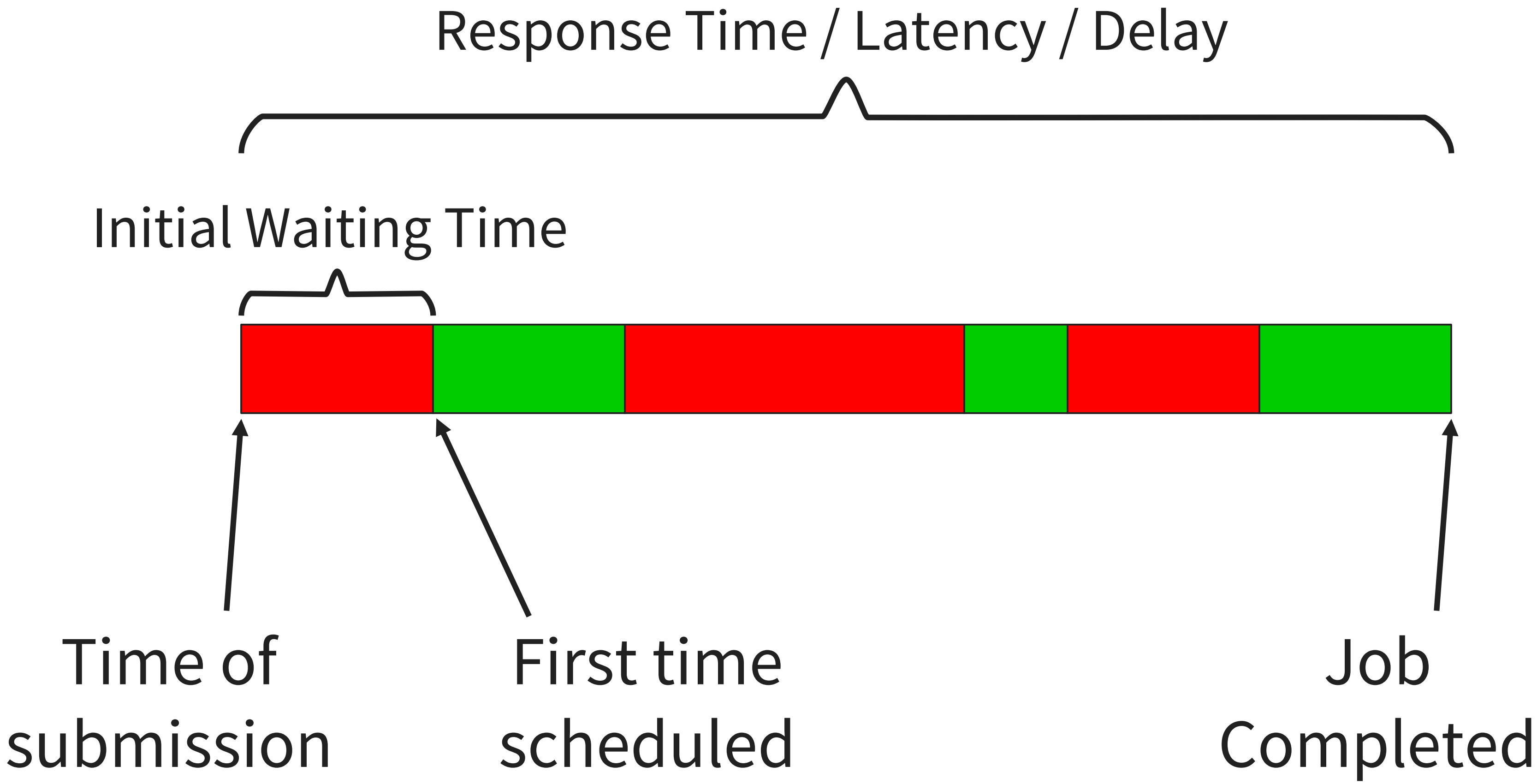
- User-perceived time before task begins.

**Total waiting time:** How much thumb-twiddling?

- Time on the run queue but not running.

Terminology Alert!

# Per Job or Task Metrics




Total Waiting Time: sum of “red” periods

# More Performance Terminology

**Throughput:** How many tasks over time?

- The rate at which tasks are completed.

**Predictability:** How consistent?

- Low variance in response time for repeated requests.

**Overhead:** How much extra work?

- Time to switch from one task to another.

**Fairness:** How equal is performance?

- Equality in the number and timeliness of resources given to each task.

**Starvation:** How bad can it get?

- The lack of progress for one task, due to resources given to a higher priority task.

# The Perfect Scheduler

- Minimizes latency
- Maximizes throughput
- Maximizes utilization:
  keeps all devices busy
- Meets deadlines:
  think image processing, car brakes, *etc.*
- Is Fair:
  everyone makes progress, no one starves

No such scheduler exists! ☹

# When does scheduler run?

## Non-preemptive

Process runs until it voluntarily yields 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
- When processes cannot be trusted to yield
- Incurs some overhead

# Process Model

Processes switch between CPU & I/O bursts

CPU-bound jobs: Long CPU bursts


I/O-bound: Short CPU bursts


Problems:

- don’t know job’s type before running
- jobs also change over time

# Basic scheduling algorithms:

- First in first out (FIFO)
- Shortest Job First (SJF)
- Round Robin (RR)

# First In First Out (FIFO)

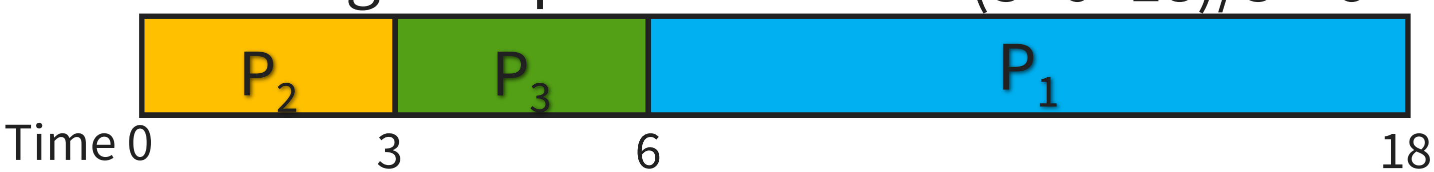
Processes $P_1$, $P_2$, $P_3$ with compute time 12, 3, 3

Scenario 1: arrival order $P_1$, $P_2$, $P_3$

Average Response Time: (12+15+18)/3 = 15


Scenario 2: arrival order $P_2$, $P_3$, $P_1$

Average Response Time: (3+6+18)/3 = 9




Note: this is always non-preemptive

# FIFO Roundup


+ Simple
+ Low-overhead
+ No Starvation
+ Optimal avg. response time if all tasks same size


– Poor avg. response time if tasks have variable size

– Average response time very sensitive to arrival time

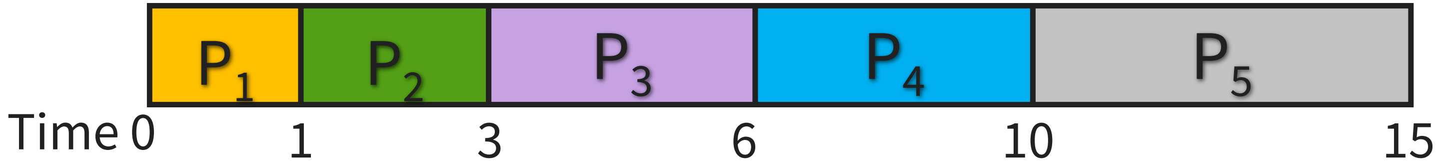

– Not responsive to interactive tasks

# Shortest Job First (SJF)

Schedule in order of estimated completion† time

Scenario : each job takes as long as its number

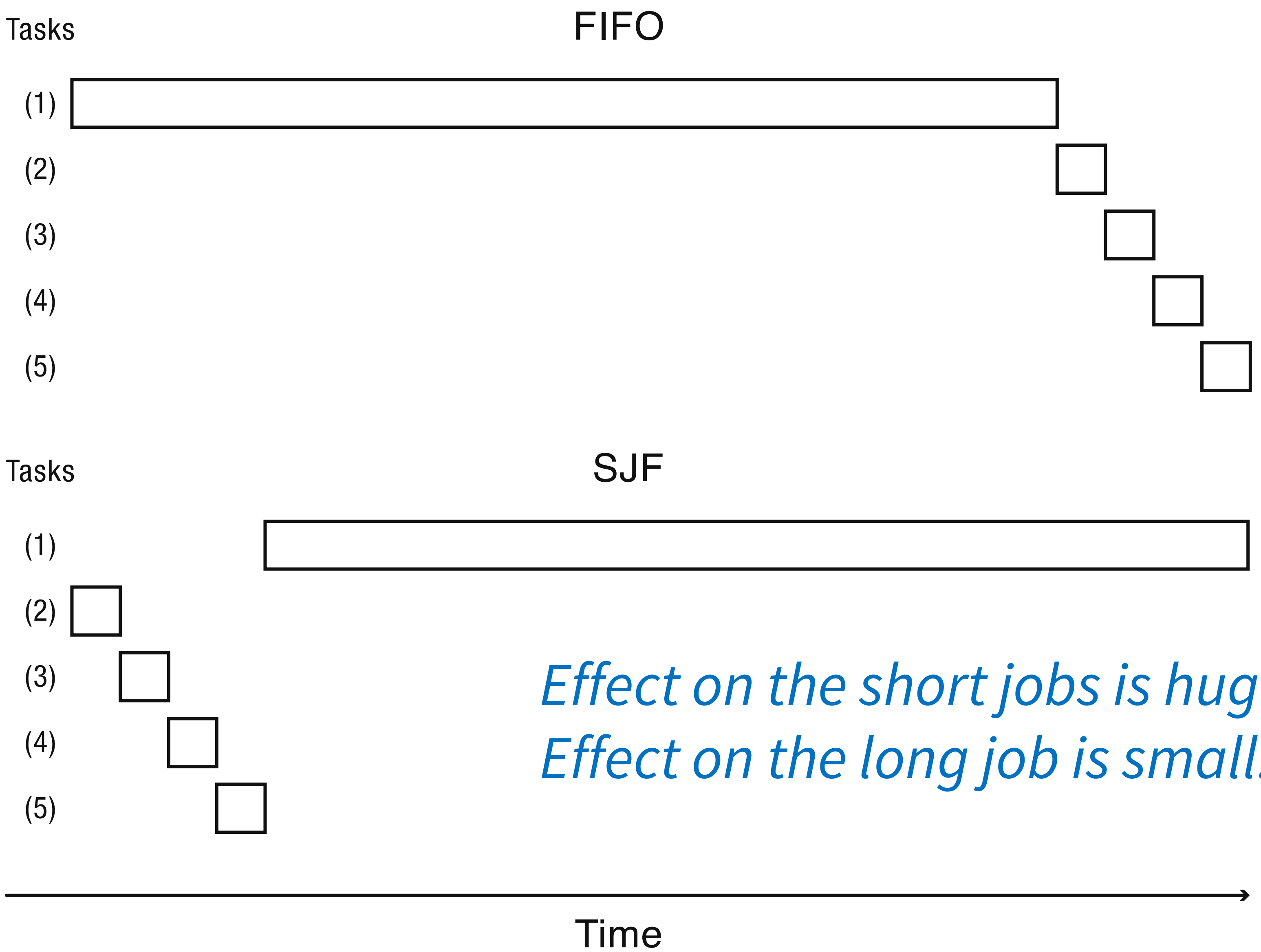
Average Response Time: (1+3+6+10+15)/5 = 7




*Would another schedule improve avg response time?*

†with preemption, remaining time

# FIFO vs. SJF




*Effect on the short jobs is huge.*
*Effect on the long job is small.*

# Shortest Job First Prediction

SJF is optimal **if** we know how long each process will run.
How to approximate duration of next CPU-burst?

- Based on the durations of the past bursts
- Past can be a good predictor of the future
- **No need to remember entire past history!**

Use exponential average:

$t_n$ actual duration of $n^{th}$ CPU burst
$\tau_n$ predicted duration of $n^{th}$ CPU burst
$\tau_{n+1}$ predicted duration of $(n+1)^{th}$ CPU burst

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

$0 \leq \alpha \leq 1$, $\alpha$ determines weight placed on past behavior

# SJF Roundup


\+ Optimal average response time (when jobs available simultaneously)


– Pessimal variance in response time


– Needs estimate of execution time
– Can starve long jobs
– Frequent context switches

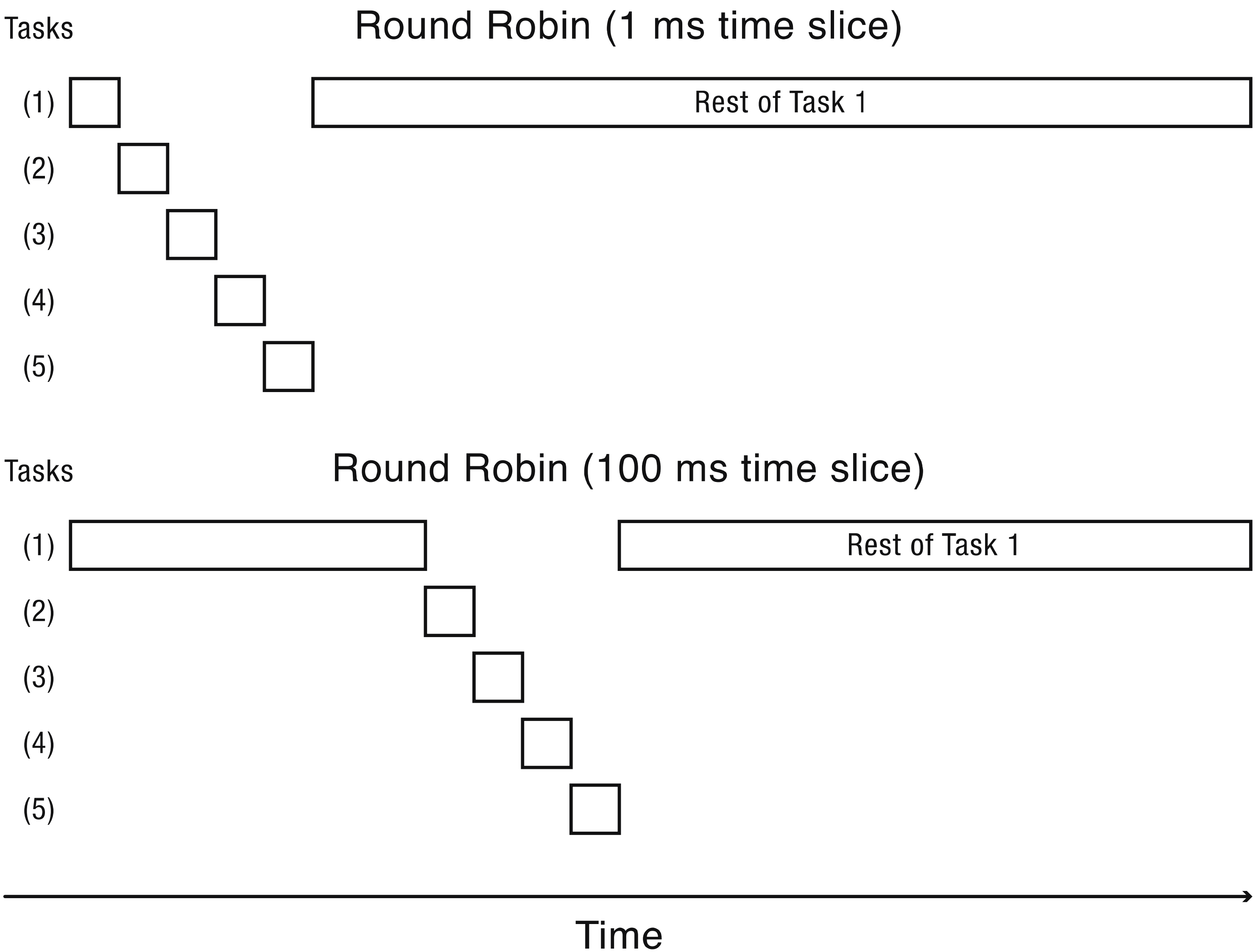
# Round Robin (RR)

- Each process allowed to run for a quantum
- Context is switched (at the latest) at the end of the quantum

What is a good quantum size?

- Too long, and it morphs into FIFO
- Too short, and much time lost context switching
- Typical quantum: about 100X cost of context switch (~100ms vs. << 1 ms)

# Effect of Quantum Choice in RR



# Round Robin vs FIFO

Assuming no overhead to time slice, is Round Robin always better than FIFO?

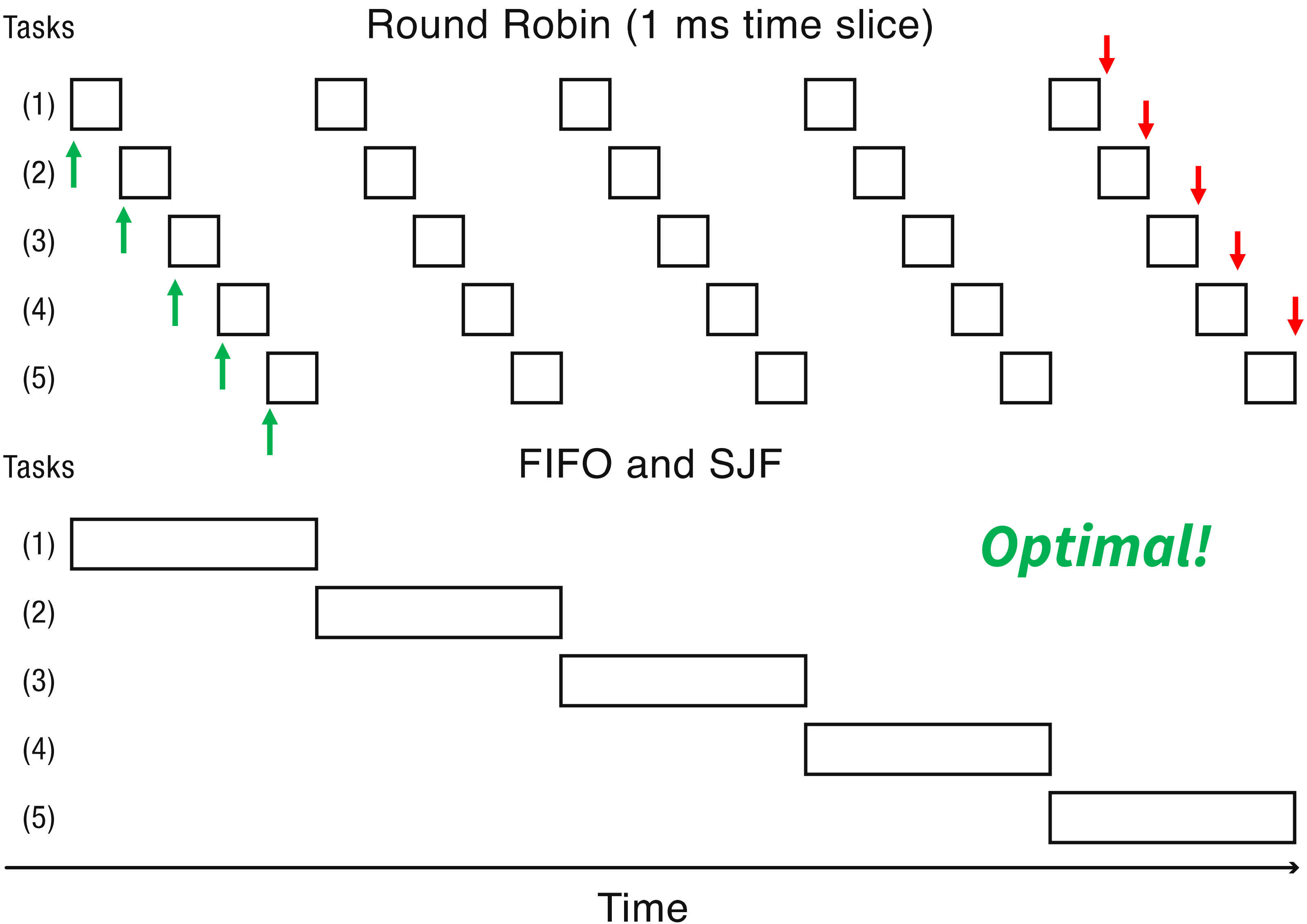
What’s the worst case scenario for Round Robin?

- What’s the least efficient way you could get work done this semester using RR?

# Round Robin vs. FIFO

Tasks of same length that start ~same time

***At least it's fair?***



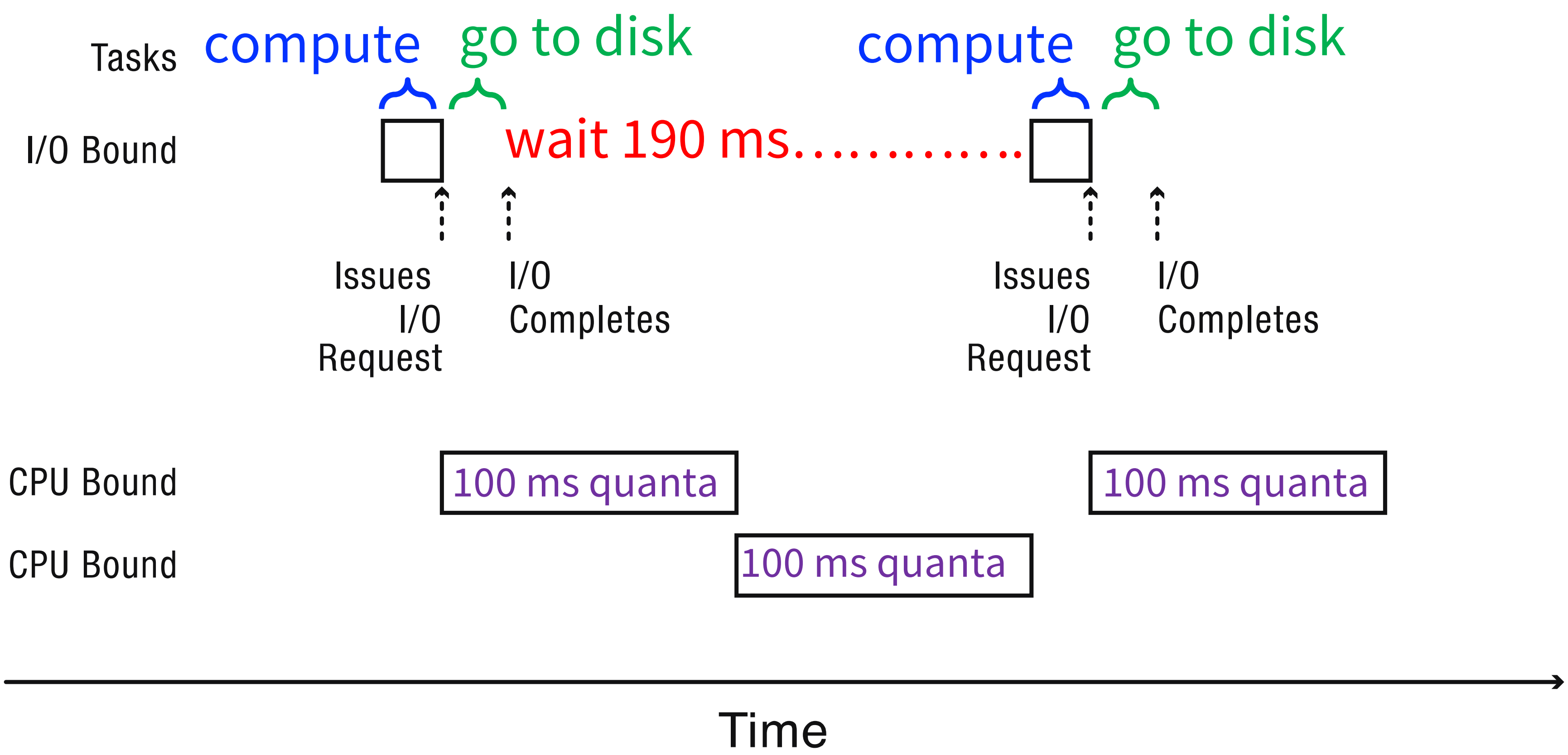

***Optimal!***

# More 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….*



# RR Roundup


+ No starvation
+ Can reduce response time
+ Low Initial waiting time


– Overhead of context switching
– Mix of I/O and CPU bound


– Particularly bad for simultaneous, equal length jobs

## Priority-based scheduling algorithms:

- Priority Scheduling
- Multi-level Queue Scheduling
- Multi-level Feedback Queue Scheduling

# Priority Scheduling

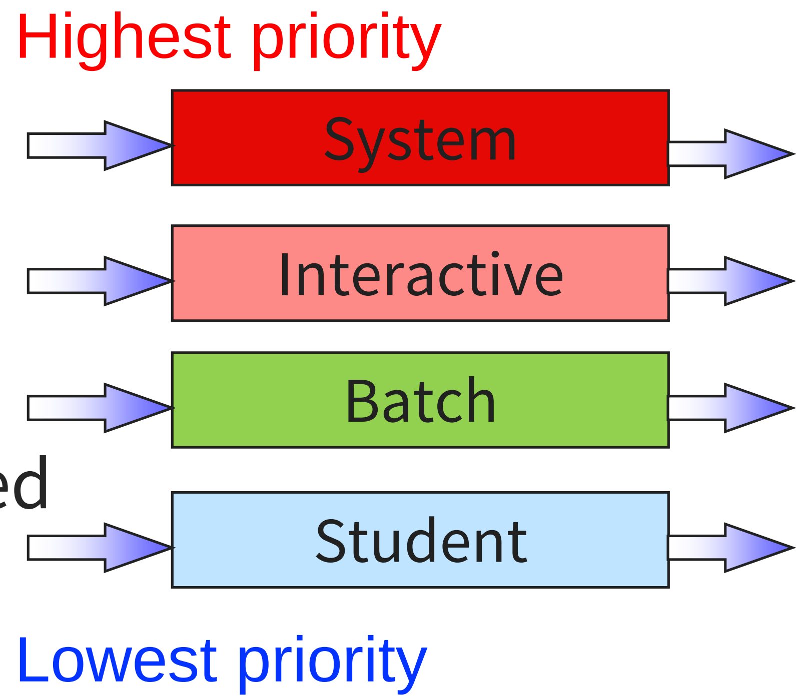
- Assign a number to each job and schedule jobs in (increasing) order
- Reduces to SJF if $\tau_n$ is used as priority
- To avoid starvation, change job's priority with time (aging)

# Multi-Level Queue Scheduling

Multiple ready queues based on job "type"

- interactive processes
- CPU-bound processes
- batch jobs
- system processes
- student programs

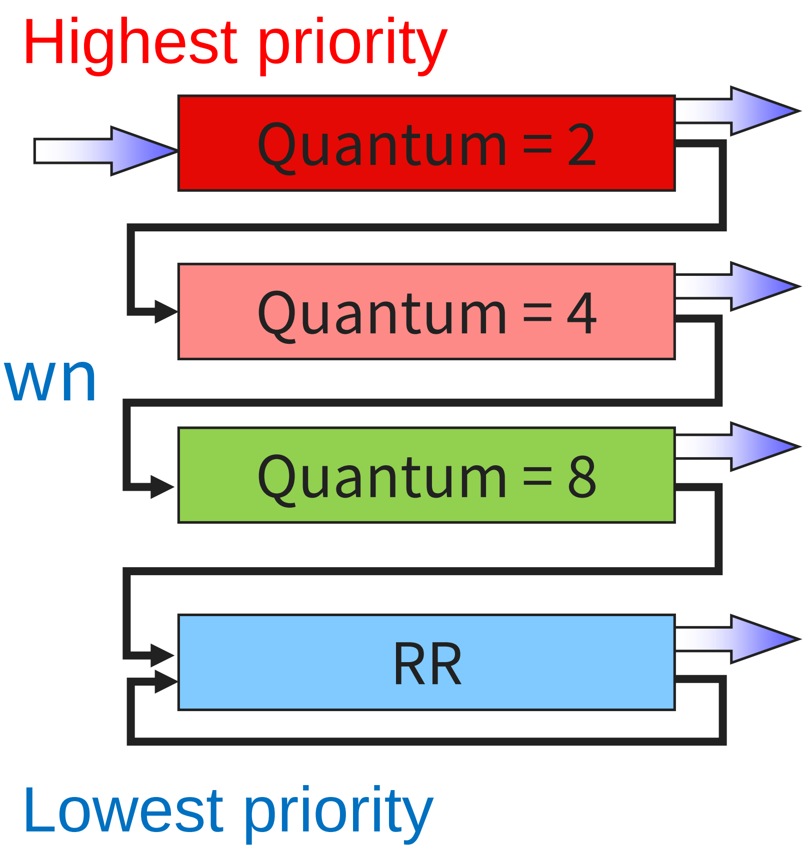
Different queues may be scheduled using different algorithms




– Queue classification difficult
(Process may have CPU-bound and interactive phases)
– No queue re-classification

# Multi-Level Feedback Queues

- Like multilevel queue, but assignments are not static
- Jobs start at the top
  - Use your quantum? move down
  - Don't? Stay where you are




Need parameters for:

- Number of queues
- Scheduling alg. per queue
- When to upgrade/downgrade job

# Problem Revisited

- Cook at State Street Diner: how to minimize the average wait time for food?
  *(most restaurants use FCFS)*

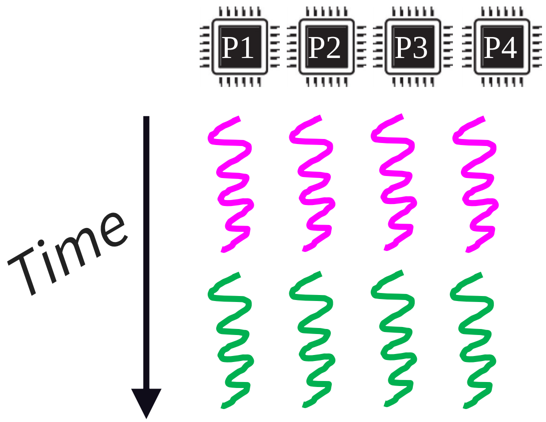
- Nurse in the emergency room

- Student with assignments, friends, and a need for sleep

# Thread Scheduling

Threads share code & data segments

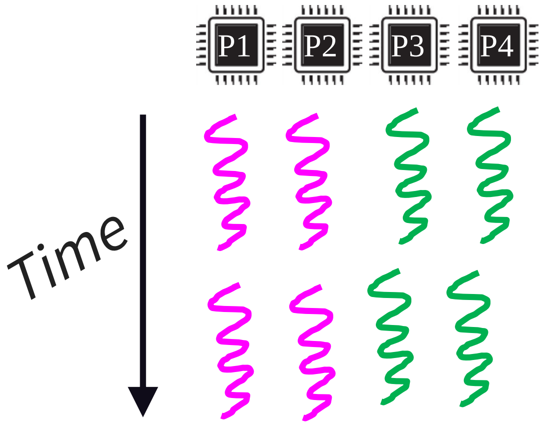
- **Option 1: Ignore this fact**
- **Option 2: Gang scheduling***
  - all threads of a process run together (pink, green)




- **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 Inheritance
  - High priority task (needing lock) donates priority to lower priority task (with lock)