CPU Virtualization:
Scheduling

OSTEP Chapter 7:
http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf

Jan Reineke
Universität des Saarlandes

"The best performance improvement is the transition from the non-working state to the working state.

That's infinite speedup."

John Ousterhout

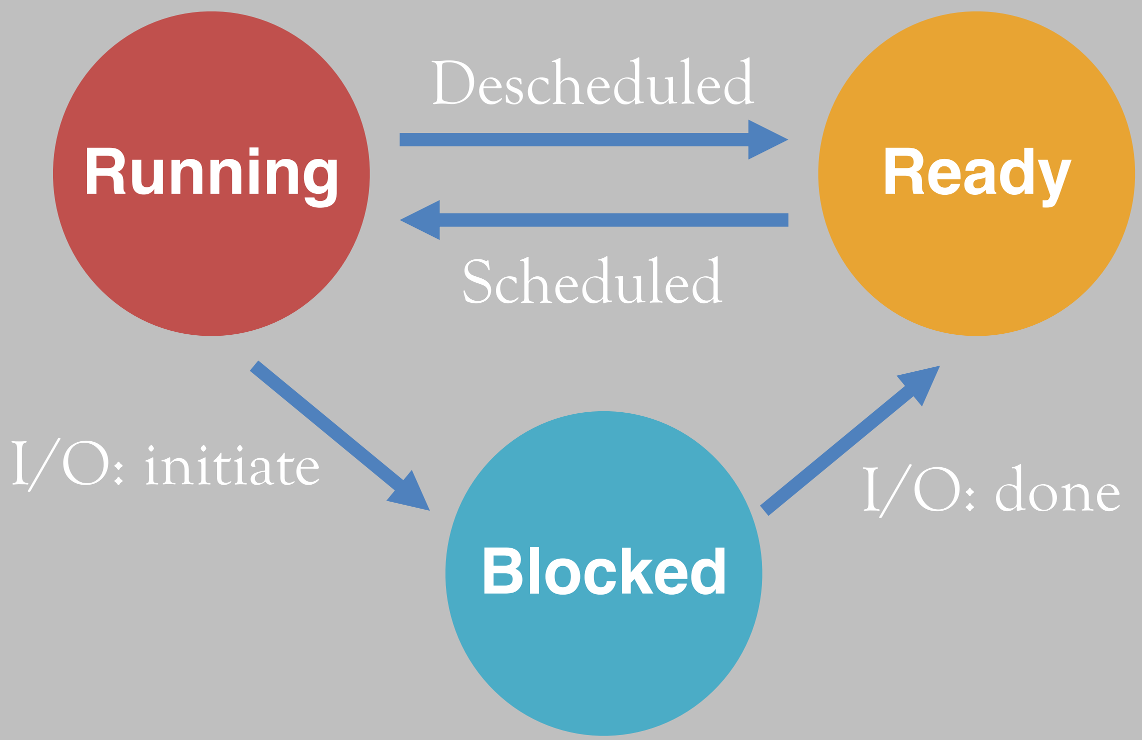
CPU Virtualization: Two Components

Dispatcher (*Previous lecture*)

- Low-level mechanism
- Performs context-switch
  - Switch from user mode to kernel mode
  - Save execution state (registers) of old process in PCB
  - Insert PCB in ready queue
  - Load state of next process from PCB to registers
  - Switch from kernel to user mode
  - Jump to instruction in new user process

- **Scheduler** (*Today*)
  - Policy to determine which process gets CPU when

Review: Mechanism vs Policy



How to transition? → "mechanism"
When to transition? → "policy"

Vocabulary

Workload: set of **job** descriptions (arrival time, run-time)

- Job: View as current CPU burst of a process
- Process alternates between CPU and I/O
  process moves between ready and blocked queues

Scheduler: logic that decides which ready job to run

Metric: measurement of scheduling quality

Performance metrics

Minimize turnaround time

- Do not want to wait long for job to complete
- Completion_time – arrival_time $T_{turnaround} = T_{completion} - T_{arrival}$

Minimize response time

- Schedule interactive jobs promptly so users see output quickly
- Initial_schedule_time – arrival_time $T_{response} = T_{firstrun} - T_{arrival}$

Maximize throughput

- Want many jobs to complete per unit of time

Minimize overhead

- Reduce number of context switches

Maximize fairness

- All jobs get same amount of CPU over some time interval

Meet deadlines: real-time systems

- Some tasks **must** finish at a given point in time

Workload assumptions

Initially, we make the following unrealistic assumptions:

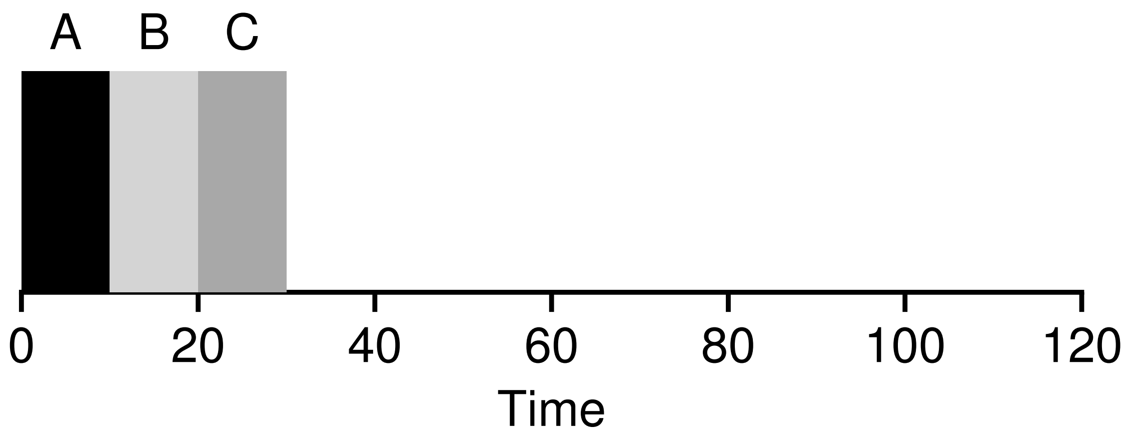
1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

We lift these assumptions one by one later on.

First In, First Out (FIFO), *also*: First Come, First Served (FCFS)

In daily life: checkout at supermarket

Example: Jobs A, B, C, Run-time: 10 time units each, all jobs arrive at time 0.



Average turnaround time?

Average response time?

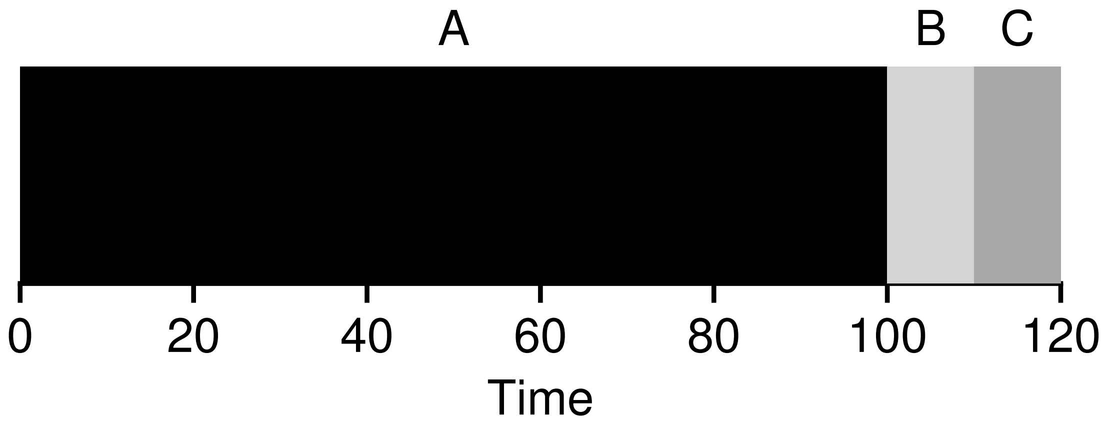
Workload assumptions

1. ~~Each job runs for the same amount of time~~
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

Convoy effect

Convoy effect

Extreme example: first job takes very long:

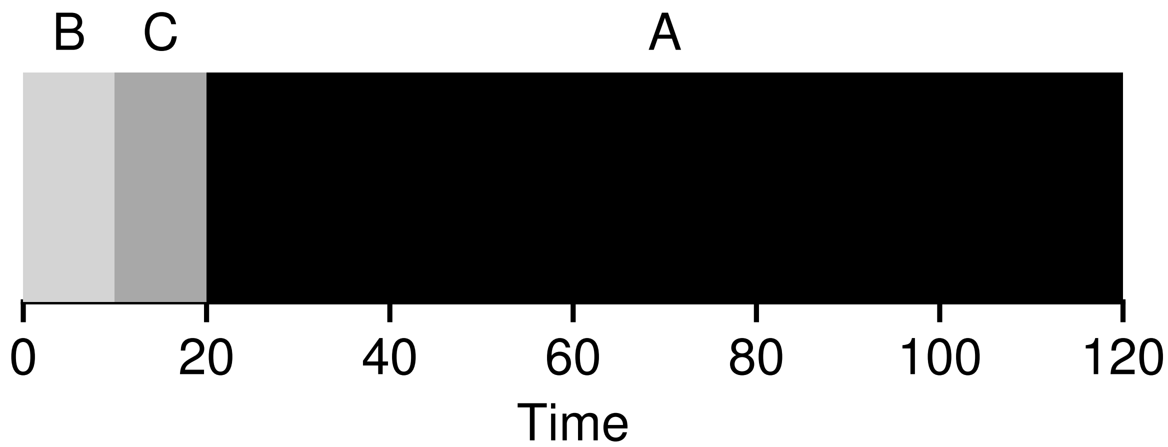


Analogy: Country road

Average turnaround time?

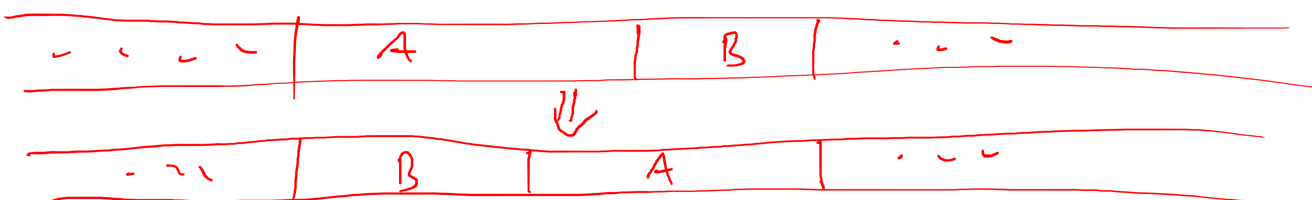
Alternatives?

Shortest Job First



Analogy in daily life: express checkouts

Optimal w.r.t. average turnaround time!



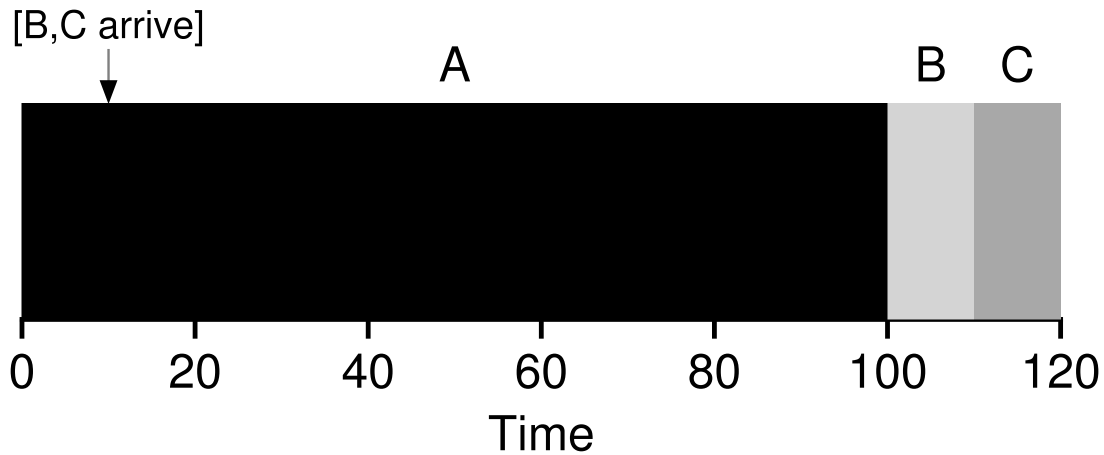
Why? Proof?

Workload assumptions

1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

Stuck behind a tractor again!

B and C arrive shortly after A:



Ideas?

Preemptive scheduling

Previous schedulers:

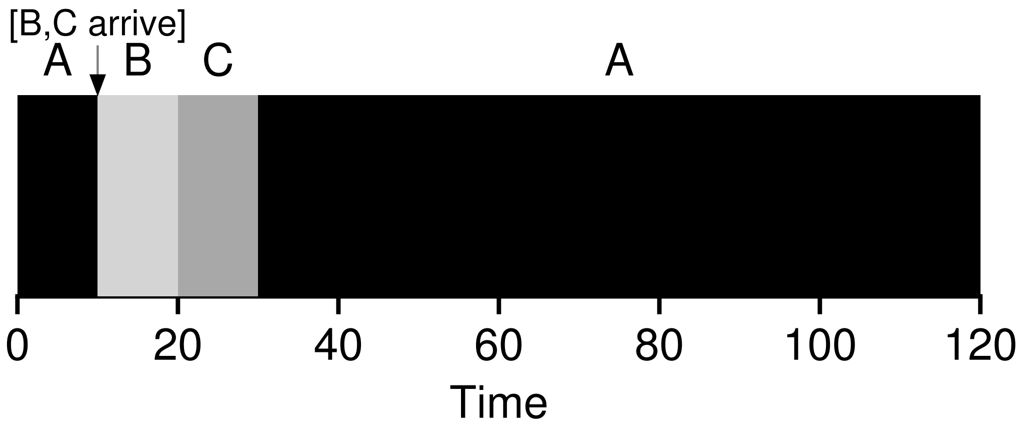
- FIFO and SJF are non-preemptive
  - Only schedule new job when previous job voluntarily relinquishes CPU (performs I/O or exits)

New scheduler:

- Preemptive: Potentially schedule different job at any point by taking CPU away from job
- STCF (Shortest Time-to-Completion First)

Shortest Time-to-Completion First (STCF)

Preemptive variant of Shortest Job First.



Optimal w.r.t. average turnaround time!

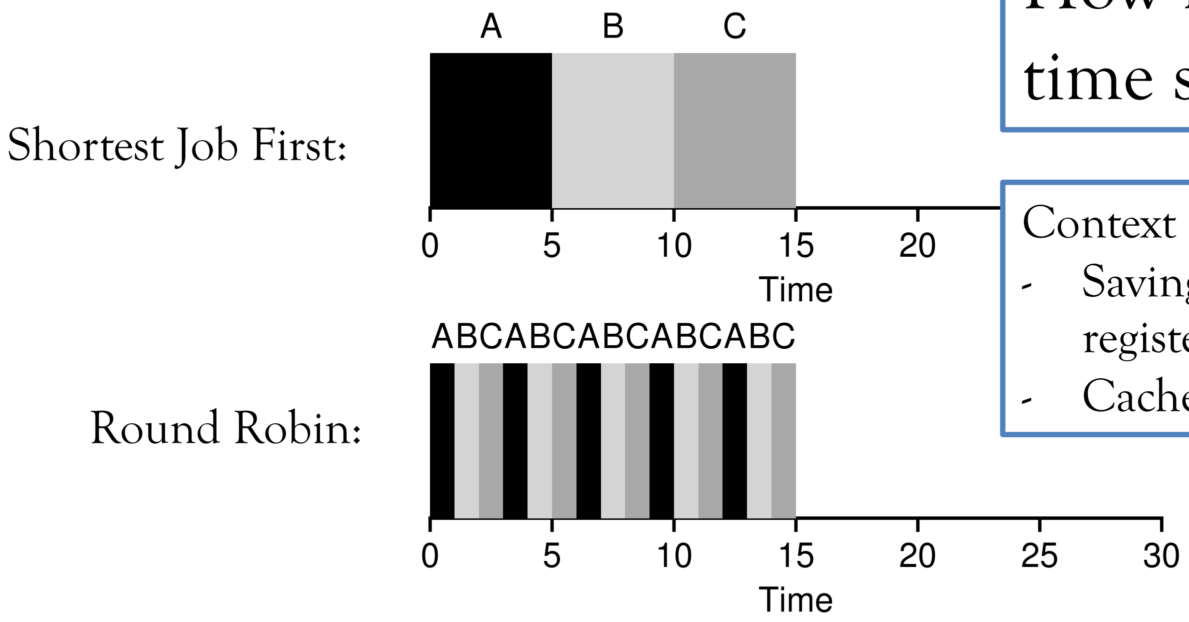
→ used e.g. in webservers: handle short pages first

Disadvantages:

1. Response time for interactive processes?
2. Very long jobs may starve ("starvation")

Round Robin

Alternate between ready processes every fixed-length time time slice.



How long should time slices be?

Context switches take time:
- Saving and restoring register contents
- Cache and TLB contents

→ Fair, **no** starvation, **good** in terms of response times

→ **Horrible** turnaround times with equal job lengths

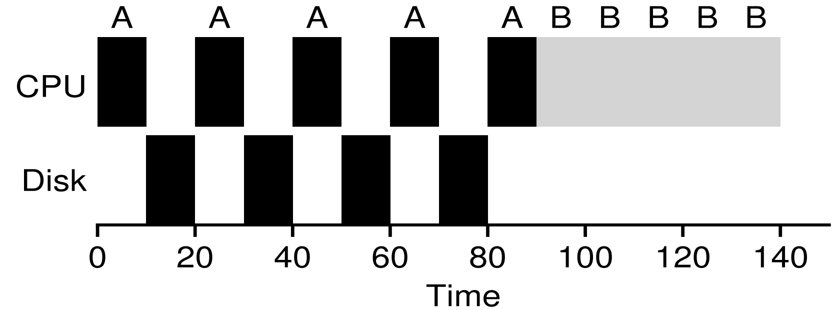
Workload assumptions

1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. ~~All jobs only use the CPU (no I/O)~~
4. Run-time of each job is known

Are there any sensible programs without I/O?

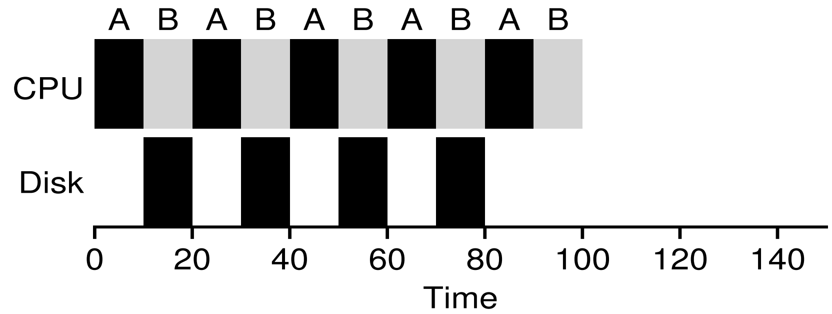
Handling I/O

Poor resource utilization:



E.g. Bittorrent in combination with CPU-intensive jobs

Goal: Overlap CPU and disk utilization:



Open questions

- How to combine the advantages of Shortest-Time-to-Completion First and Round Robin?
  - Short turnaround time (STCF)
  - Short response time (Round Robin)
  - Fairness (Round Robin)
- How to lift the final assumption? ("Run-time of each job is known")

Workload assumptions

1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. ~~All jobs only use the CPU (no I/O)~~
4. ~~Run-time of each job is known~~

Multi-Level Feedback Queue (MLFQ)

Introduced in 1962 as part of the Compatible Time-Sharing Systems.

Turing Award 1990 for Fernando Corbato (USA).

Variants of MLFQ found in Windows, MacOS, and Linux.

Multi-Level Feedback Queue: *Goals*

Make the impossible possible:

- Short response times for interactive processes
- Short average turnaround time

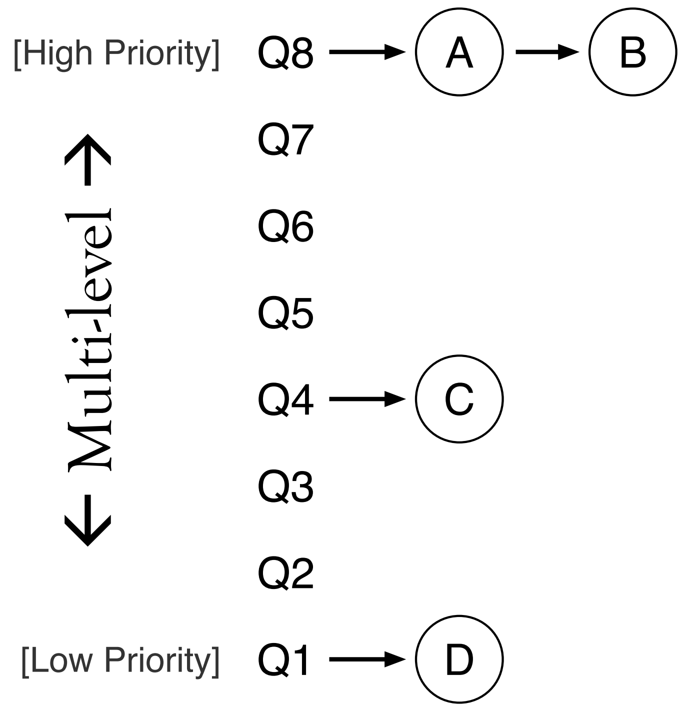
... with *a priori* unknown run-times.

Basic idea:
Learn from history, as e.g. with caches.

Multi-Level Feedback Queue: Basic rules

Rule 1: Priority(A) > Priority(B)
→ A runs

Rule 2: Priority(A) = Priority(B)
→ Round Robin between A and B



But: How are priorities set?

Learning from history

- Processes alternate between I/O and CPU work

- *Assumption:*
  Run-time of the next CPU burst (job) will be similar to run-time of previous CPU burst of the same process

Multi-Level **Feedback** Queue: Determination of priorities

- *Rule 3*: Processes start at top priority

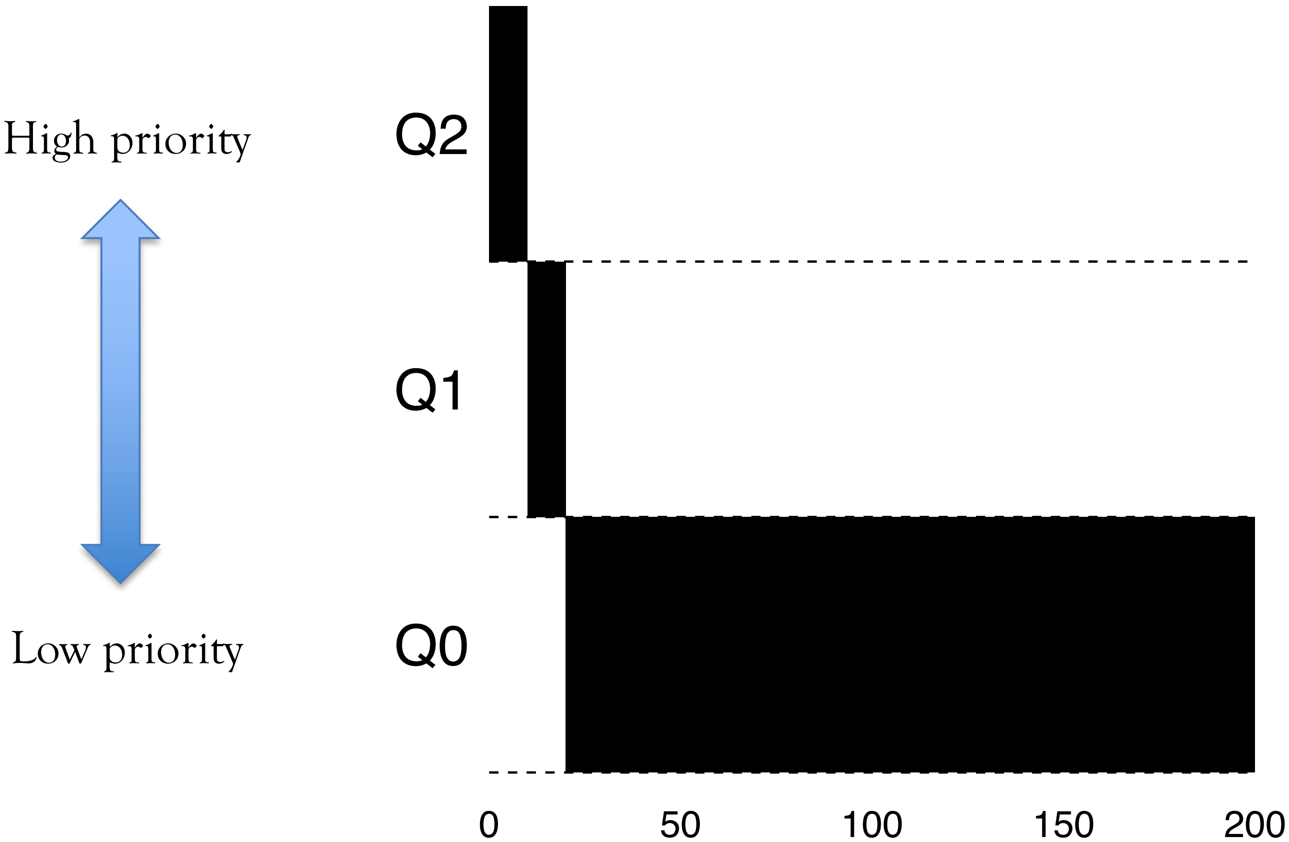
- *Rule 4a*: Priority of processes that use their entire time slice is reduced

  → Probably high run-time

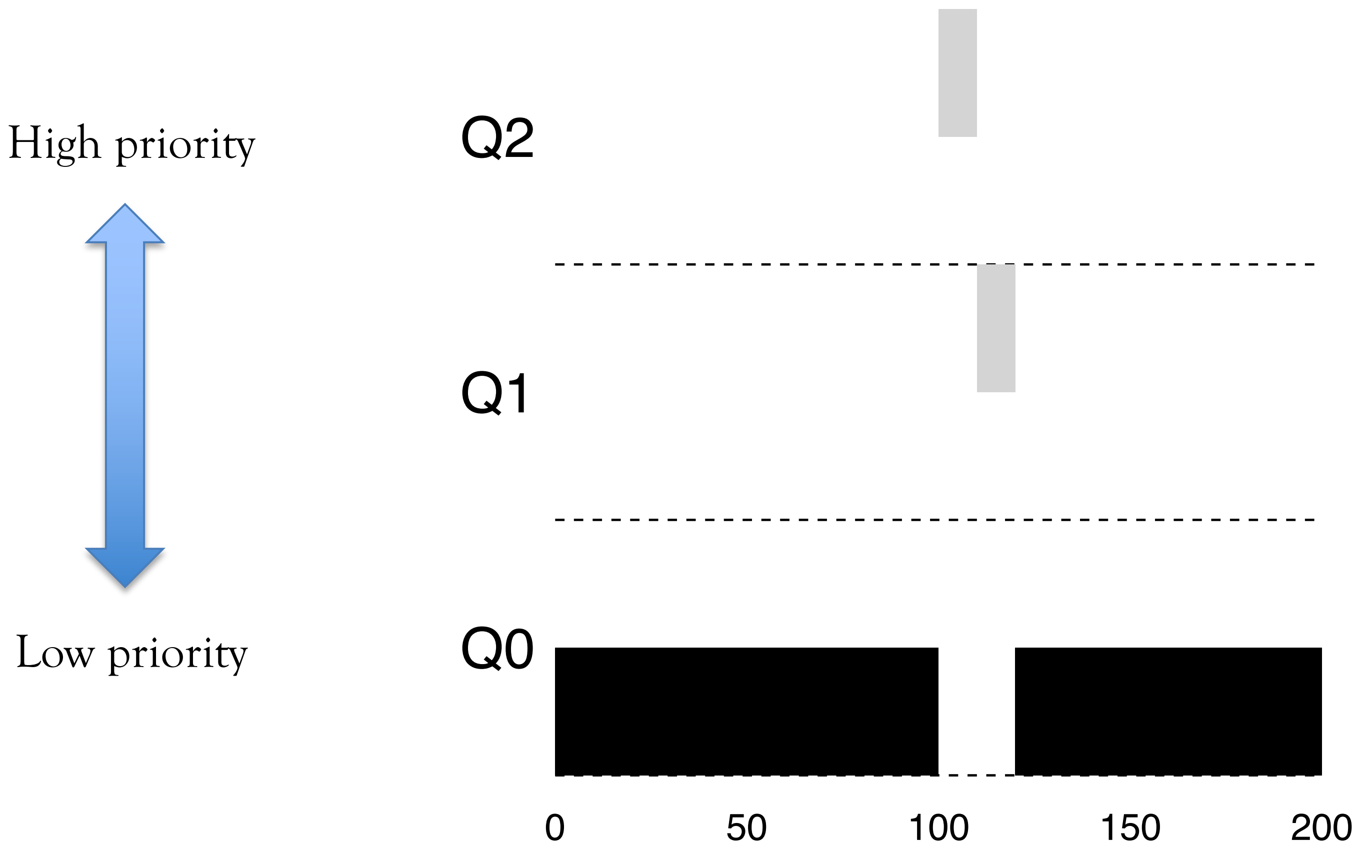
- *Rule 4b*: Priority of processes that **do not** use their entire time slice is not reduced

  → Probably interactive processes

Example 1: One long job

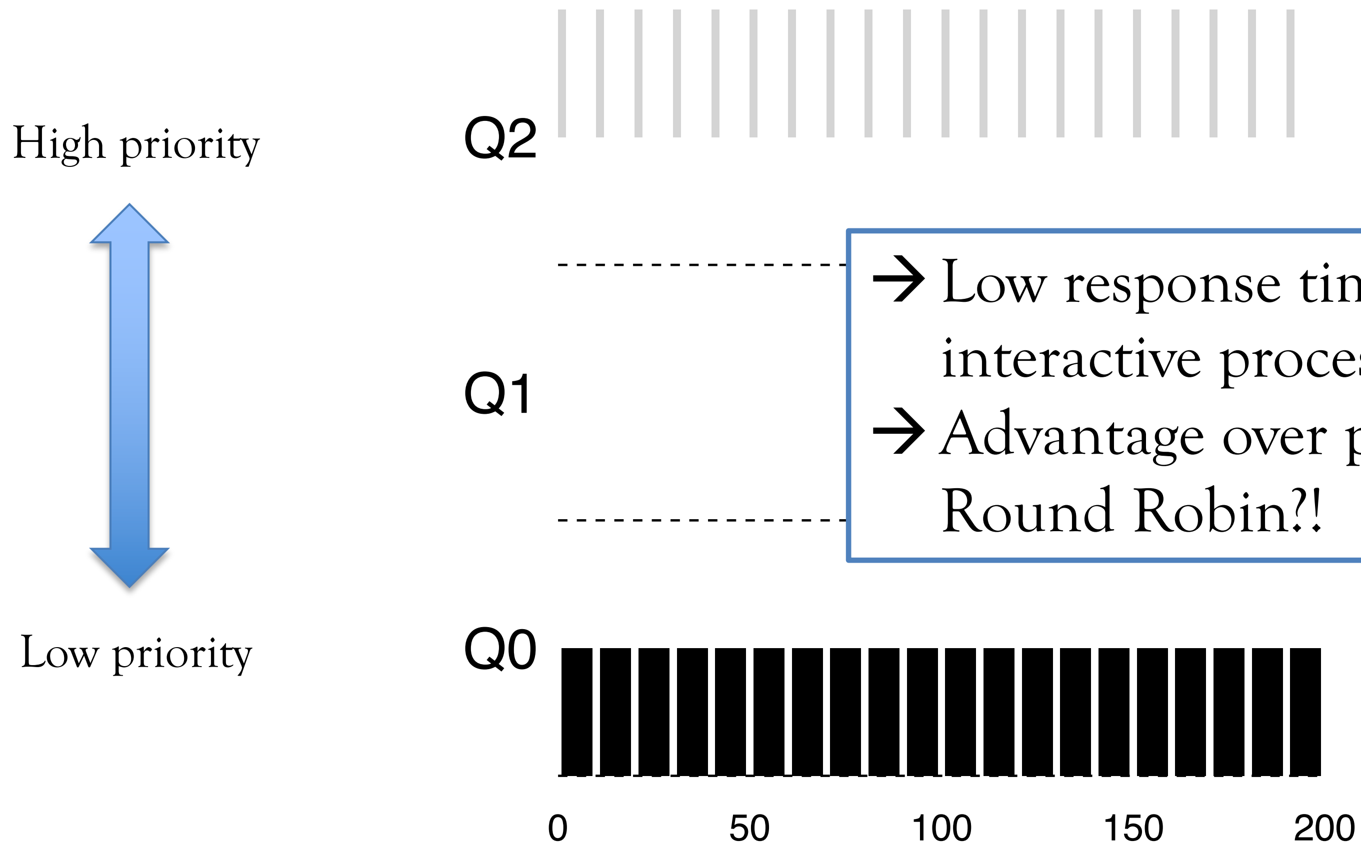


Example 2: A short job joins



→ Approximates Shortest Time-to-Completion First

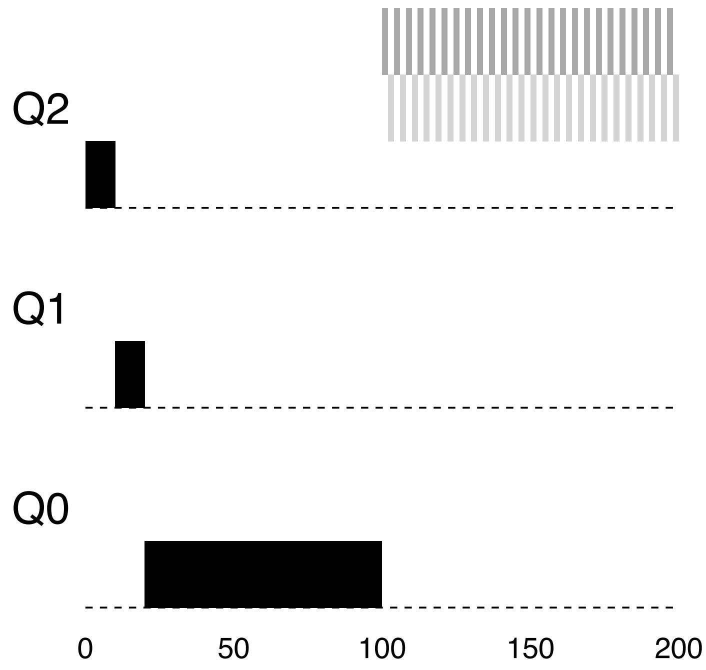
Example 3: I/O-intensive + CPU-intensive processes



→ Low response time for interactive process
→ Advantage over pure Round Robin?!

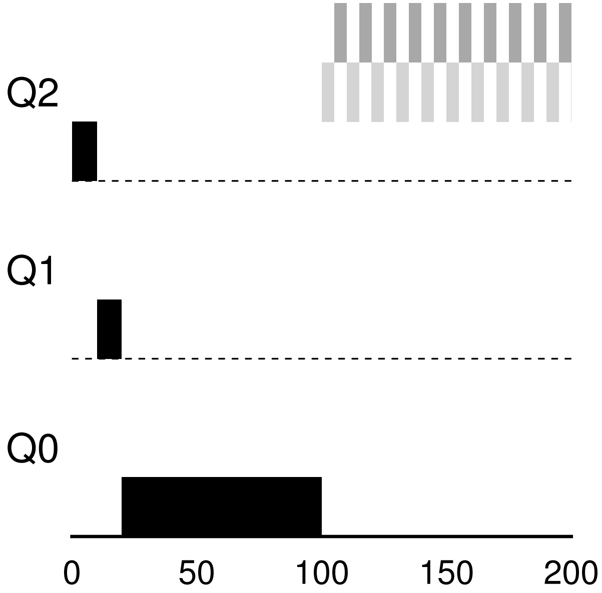
So all is good?

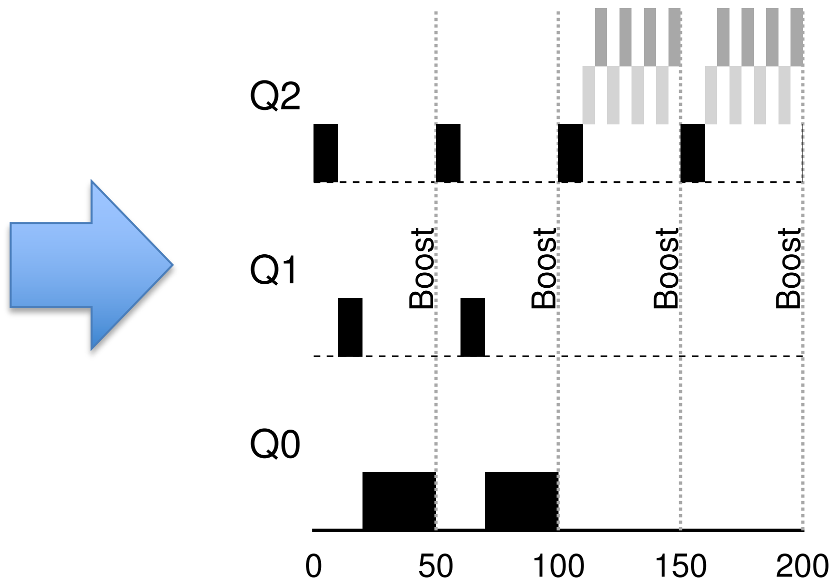
Problem 1: "starvation": long-running jobs may never get to run



Priority boost

Rule 5: Every S time units boost the priority of all jobs



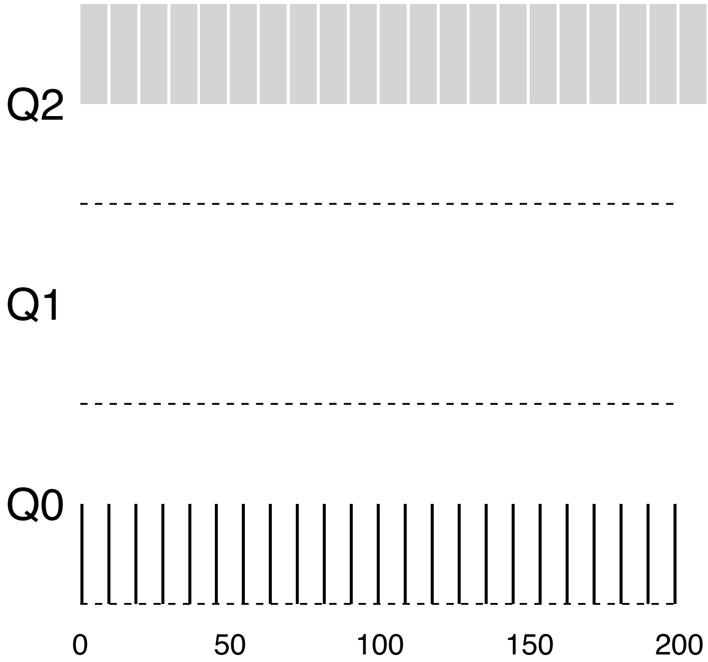


→ How should S be chosen?

→ "Voodoo" constant

So all is good?

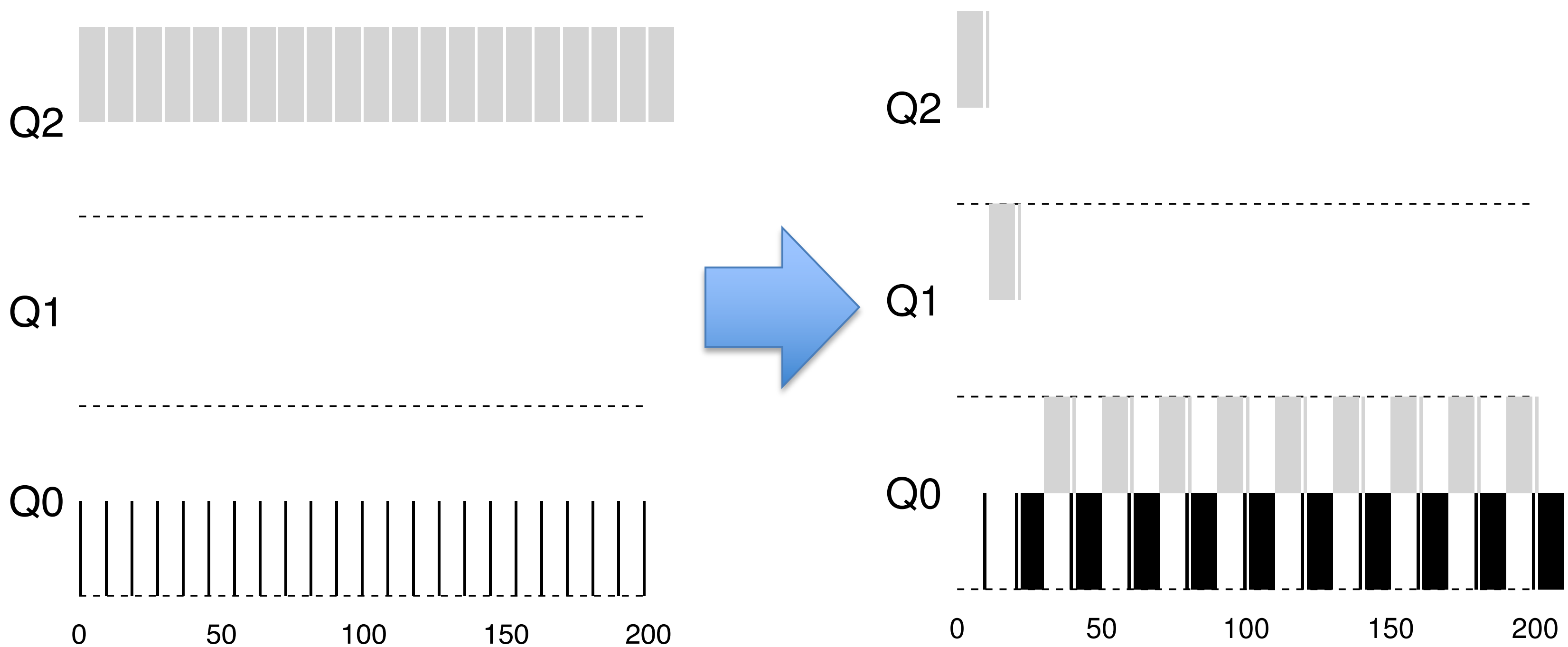
Problem 2: "Gaming" the system
→ Initiate short I/O shortly before end of time slice



Solution?

Better bookkeeping

(New) Rule 4: Reduce priority of a job when it has exhausted its **budget** at a priority level



MLFQ: Summary

Rule 1: Priority(A) > Priority(B)
→ A runs

Rule 2: Priority(A) = Priority(B)
→ Round Robin between A and B

Rule 3: Processes start at top priority

Rule 4: Reduce priority of a job when it has exhausted its budget at a priority level

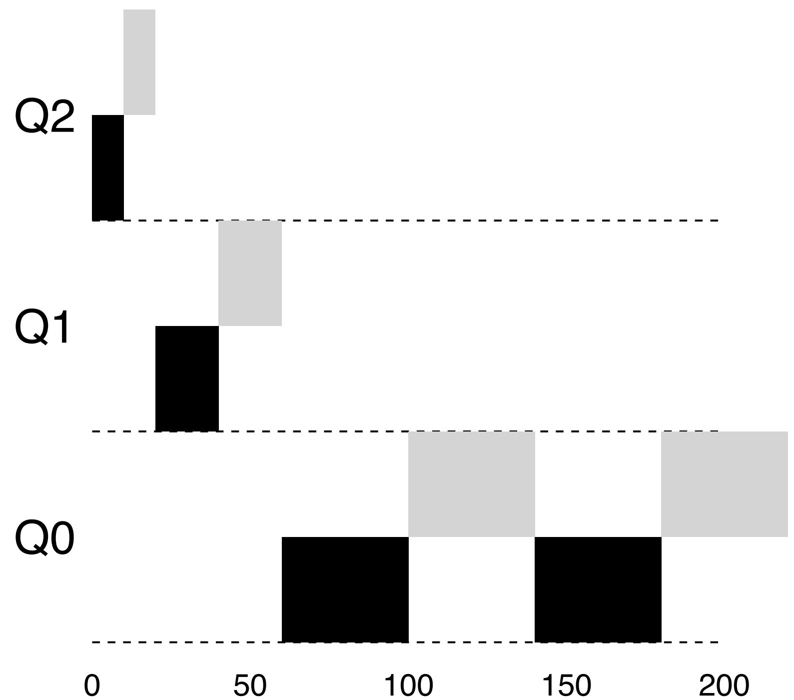
→ Prefers short jobs
→ "Learning"

Rule 5: Every S time units boost the priority of all jobs (that haven't been scheduled)

→ Against starvation

MLFQ: Fine tuning

Lower priority → longer time slices



Minimizes overhead

Quiz: Optimization goals

Optimization goals:

- Maximizing throughput
- Minimizing turnaround time
- Minimizing response time
- Fairness:
  No process should have to wait forever

→ STCF (maximizing throughput, minimizing turnaround time)

→ Round Robin (minimizing response time, fairness)

Which scheduling algorithms are optimal?

Summary

- Schedulers must support different types of processes with different goals:
  - *interactive* vs *non-interactive* processes
- Properties of processes are a priori unknown
  - Can be learned over time

Summary

- Shortest Time-to-Completion First (STCF):
  - Optimal w.r.t. average turnaround time
  - Starvation possible
- Round Robin:
  - No starvation, good in terms of reponse time
  - Poor average turnaround time
- Multi-level Feedback Queue (MLFQ):
  - Gives preference to interactive, short jobs like STCF
  - No starvation
  - Critical: "Voodoo" constants:
    - Length of time slices?
    - Number of priority levels?
    - Time budget at each priority level?