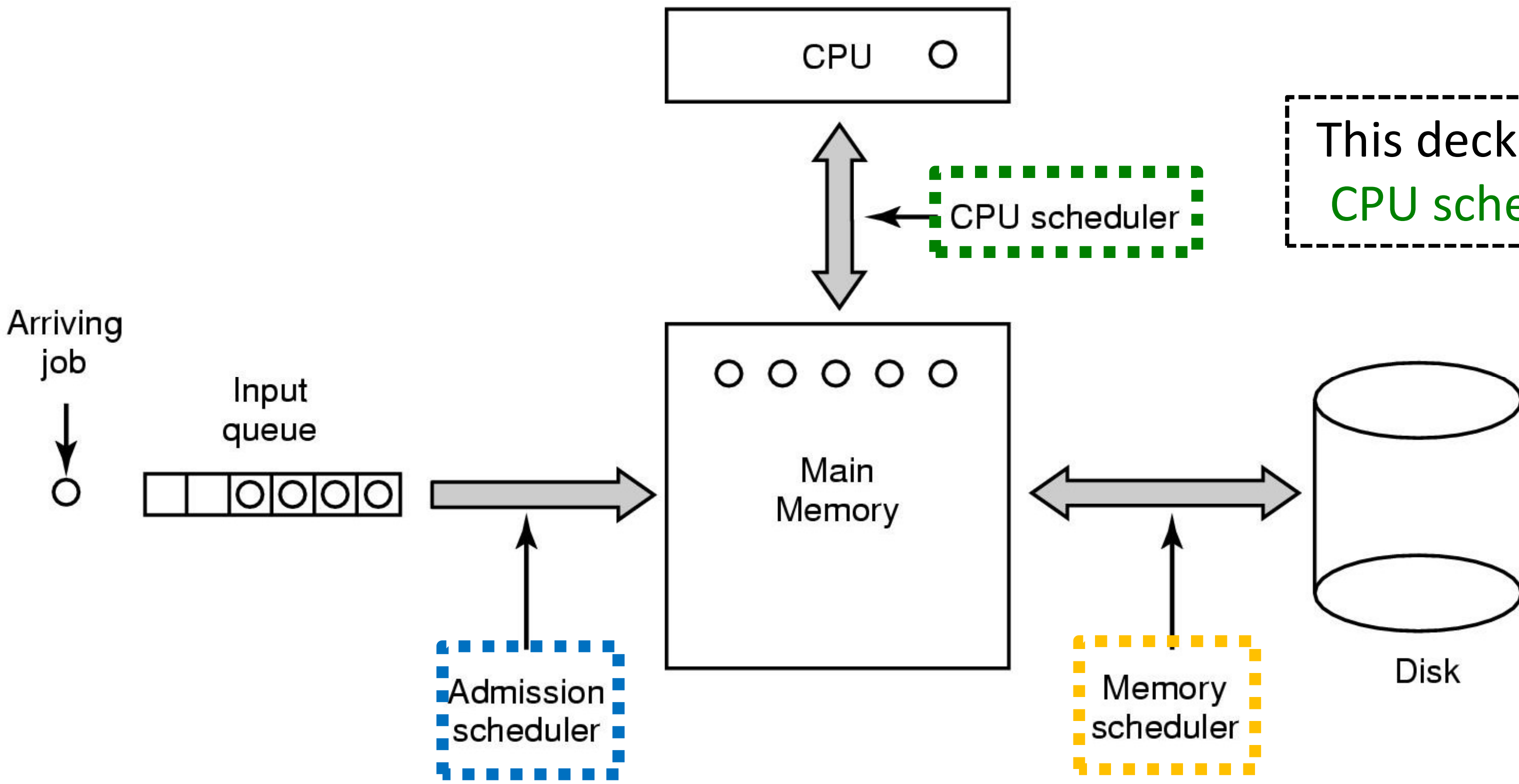
Operating Systems

CPU Scheduling

ENCE 360

Operating System Schedulers

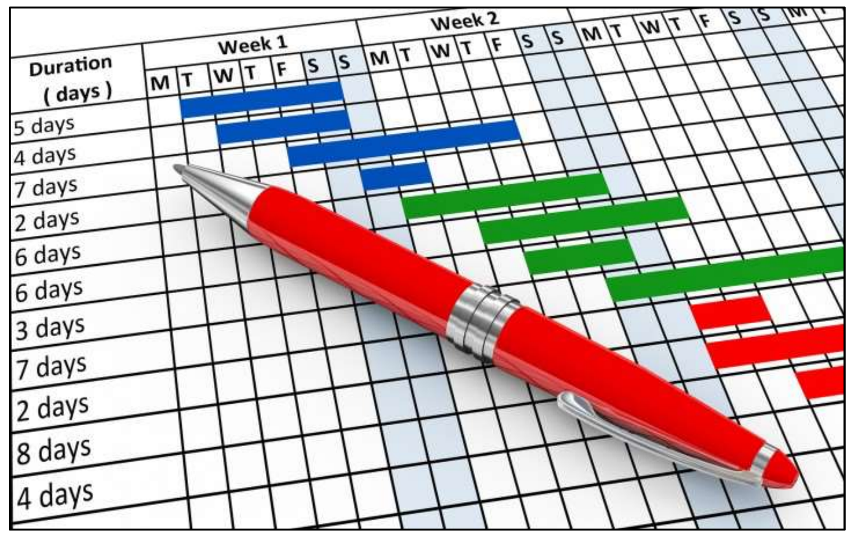
Short-Term	Long-Term (batch)	Medium-Term
"Which Ready process to Running ?"	"Which requested process into Ready Queue?"	"Which Ready process to memory?"
CPU Scheduler	Admission scheduler	Memory scheduler



This deck about
CPU scheduler

Outline

- Introduction (done)
- Scheduling Policies (next)
  - FIFO
  - SJF
  - SCTF
  - RR
  - SOS
  - MLFQ
- Other topics

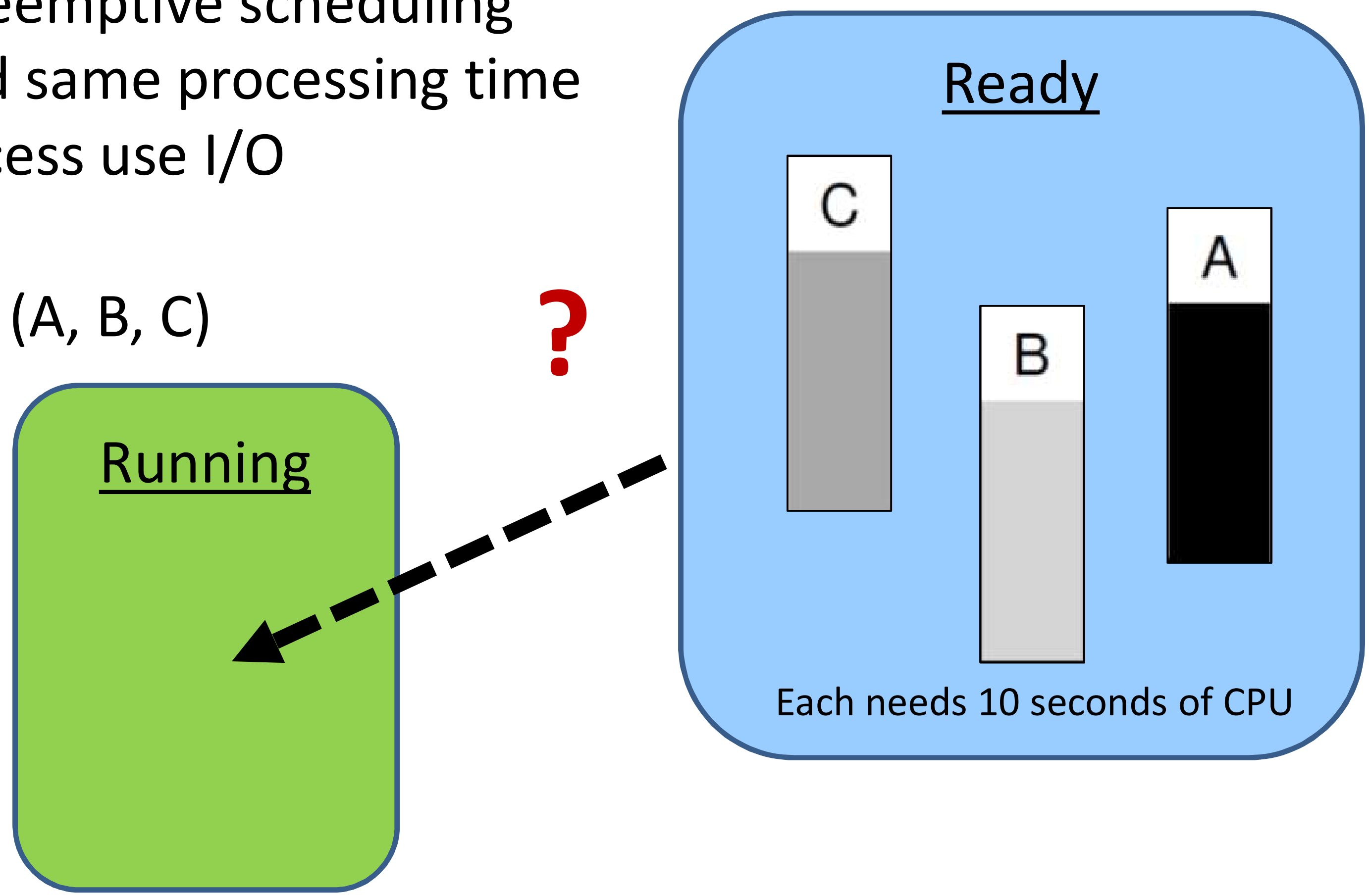


Chapter 2.4
MODERN OPERATING SYSTEMS (MOS)
By Andrew Tanenbaum

Chapters 7 & 8
OPERATING SYSTEMS: THREE EASY PIECES
By Arpaci-Dusseau and Arpaci-Dusseau

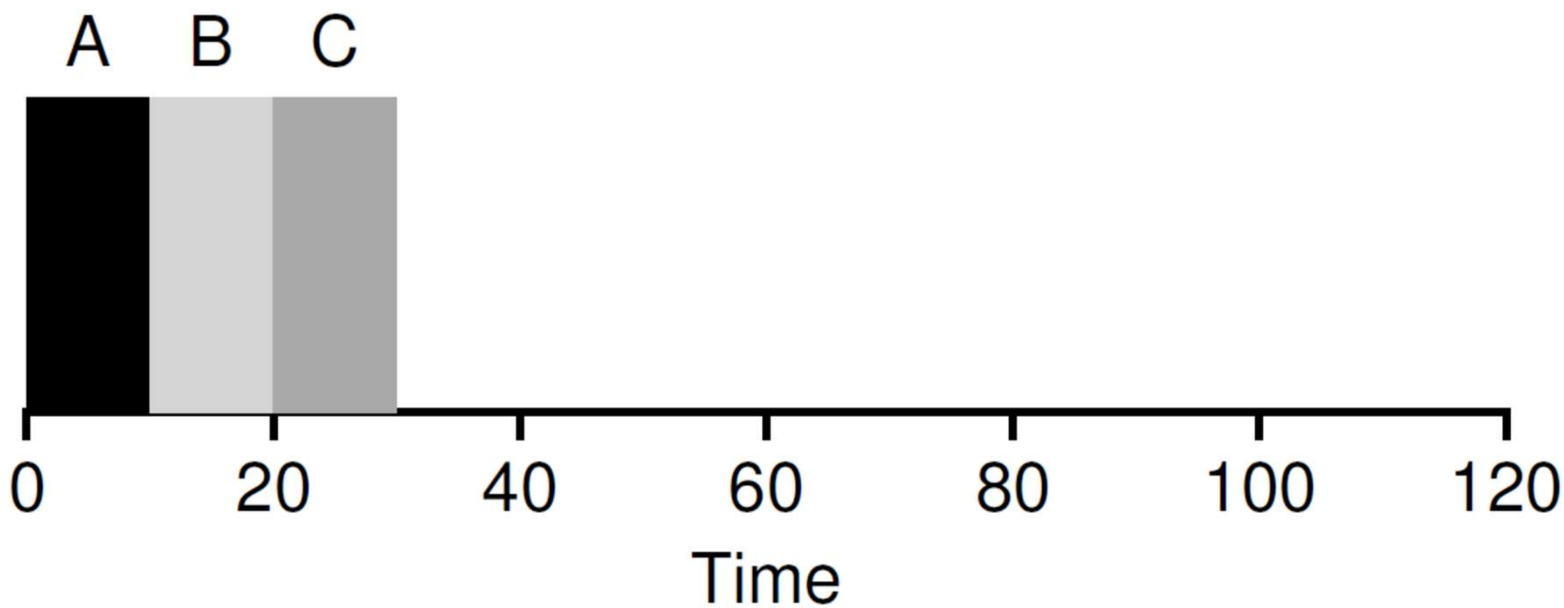
A CPU Scheduling Scenario

- Assume:
  1. Fixed number of processes
  2. All "ready" at same time
  3. Non-preemptive scheduling
  4. All need same processing time
  5. No process use I/O
- Have:
  – 3 process (A, B, C)



What is simplest policy?

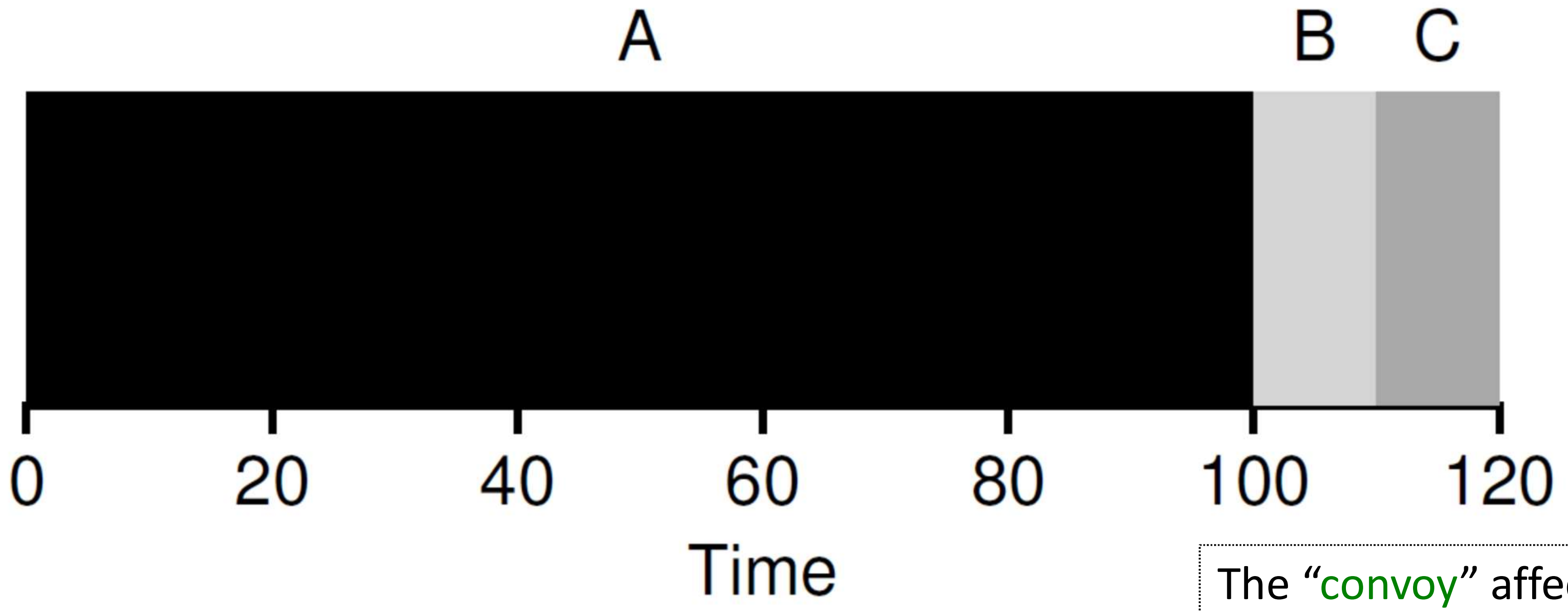
First In, First Out – Easy, Peasy!



Average turn around time = (10 + 20 + 30) / 3 = **10**

Relax assumption #4 (equal time).
When might this perform poorly?

First In, First Out – Uh, oh!



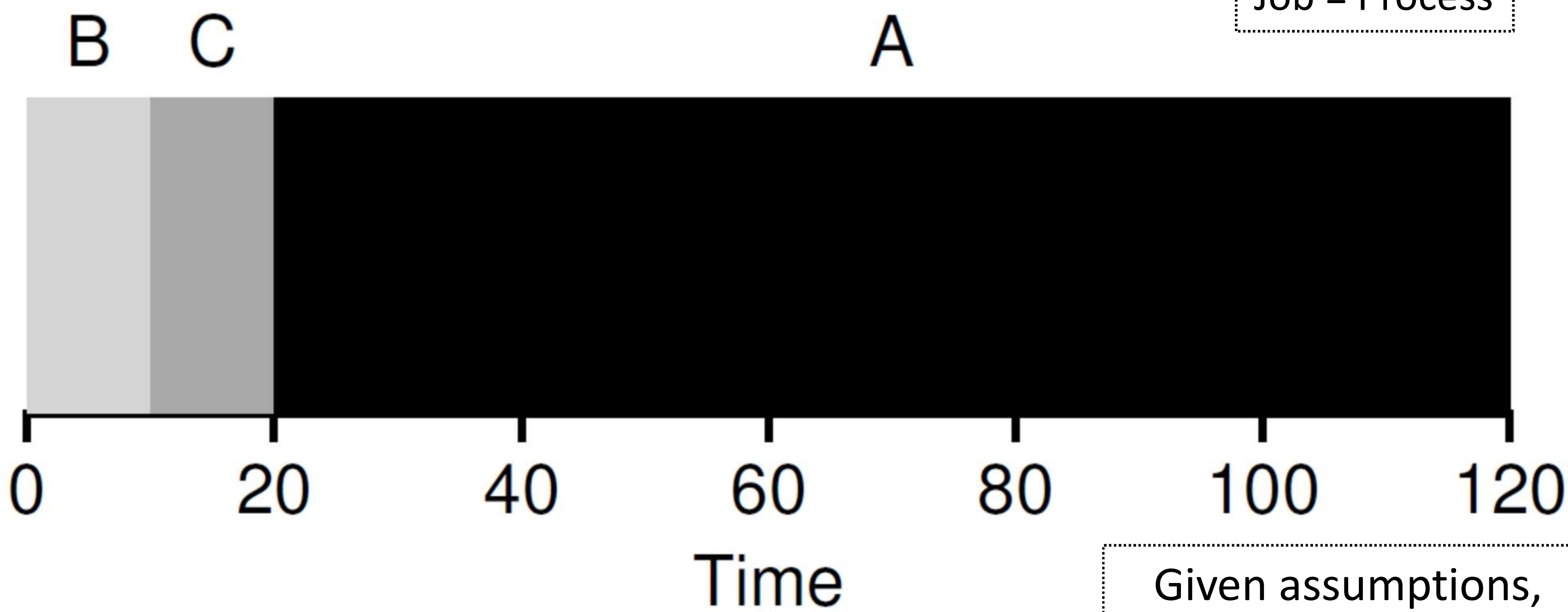
The "convoy" affect

Average turn around time = (100 + 110 + 120) / 3 = **110**

How to do better?
(Hint: think about grocery stores)

Shortest Job First (SJF)

Job = Process

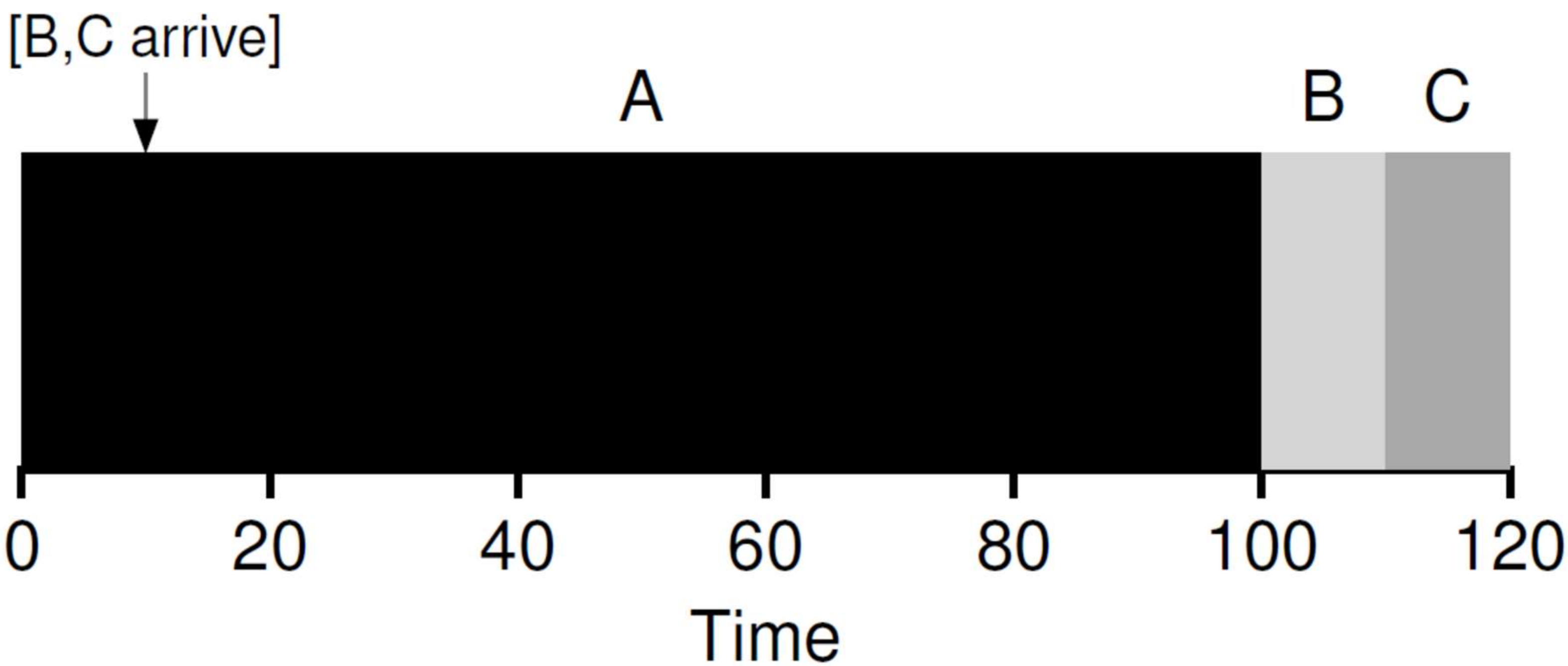


Given assumptions, SJF is provably optimal

Average turn around time = (120 + 10 + 20) / 3 = **50**

Relax assumption #2 (same starting time). When might this perform poorly?

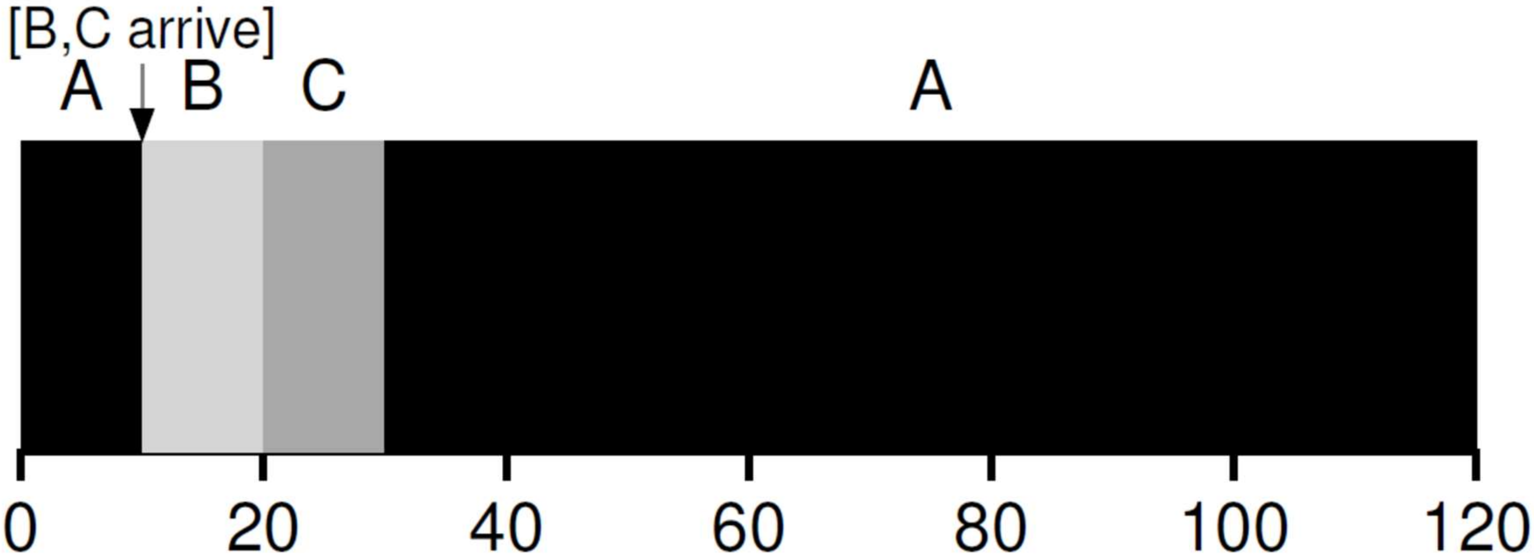
Shortest Job First – Uh, oh!



Average turn around time = (100 + 110-10 + 120-10) / 3 = **103**

Relax assumption #3 (pre-emption).
How can we make this better?

Shortest Time-to-Completion First (STCF)

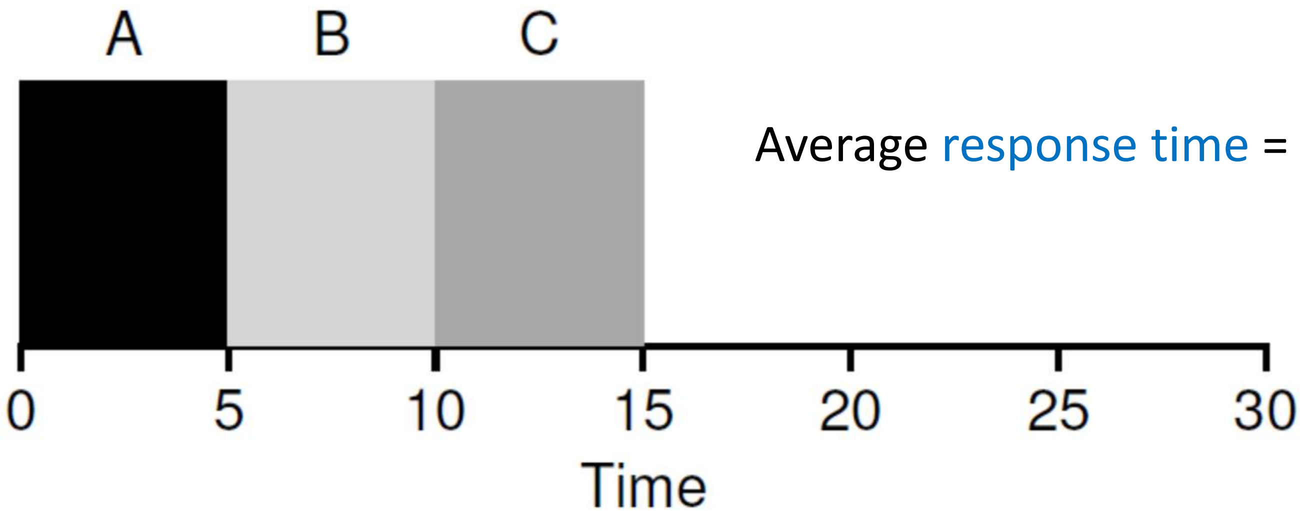


Given assumptions, provably optimal

Average turn around time = (120-0 + 20-10 + 30-20) / 3 = **50**

What if we consider users in *interactive* system?
In other words, instead of turnaround time, what might they want?

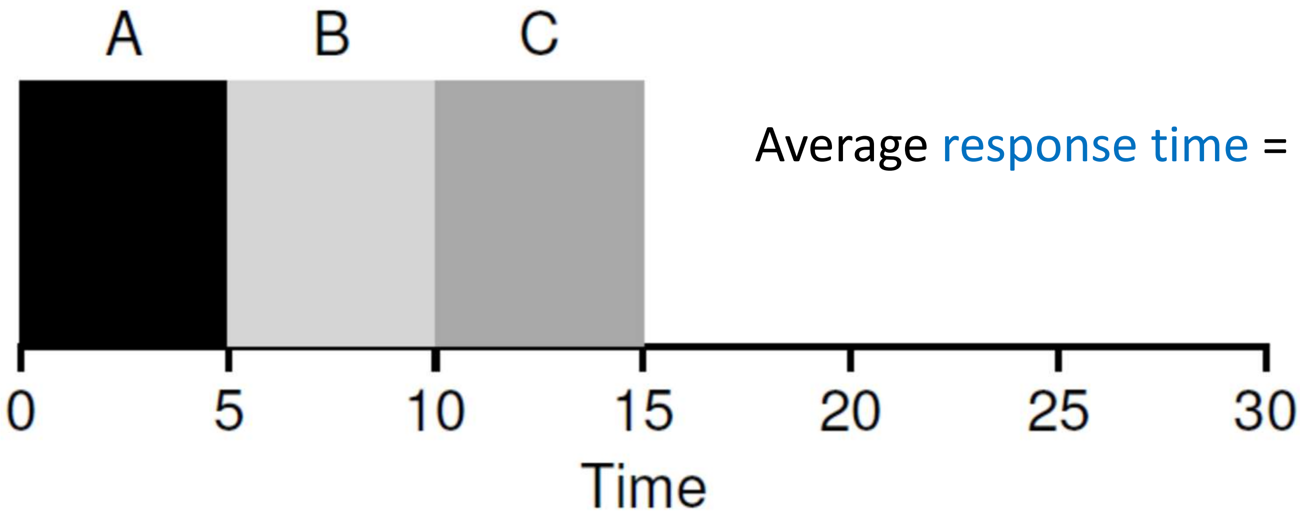
Response Time Woes



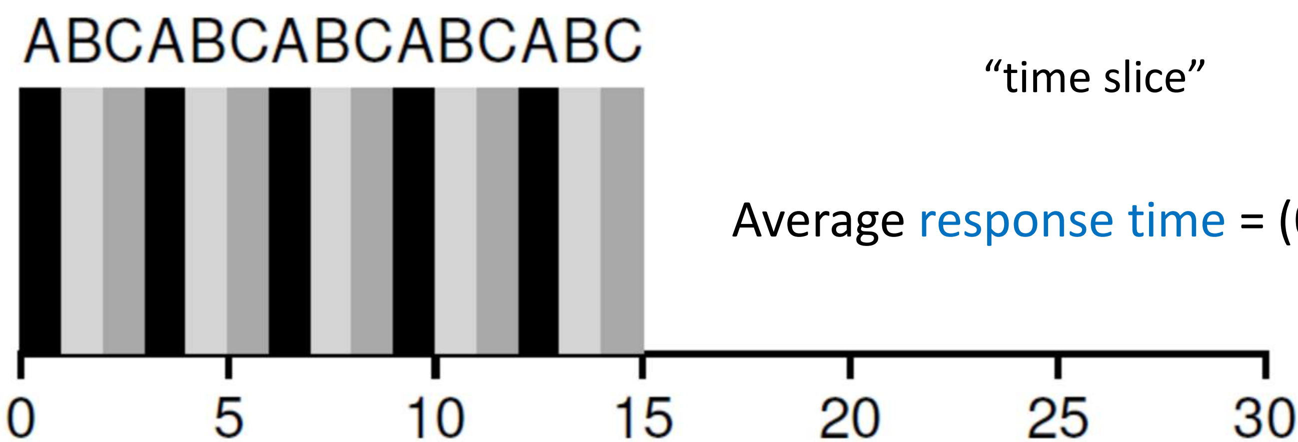
Average response time = (0 + 5 + 10) / 3 = **5**

How can we make response time better?

Round Robin (RR) to the Rescue!



Average response time = (0 + 5 + 10) / 3 = **5**

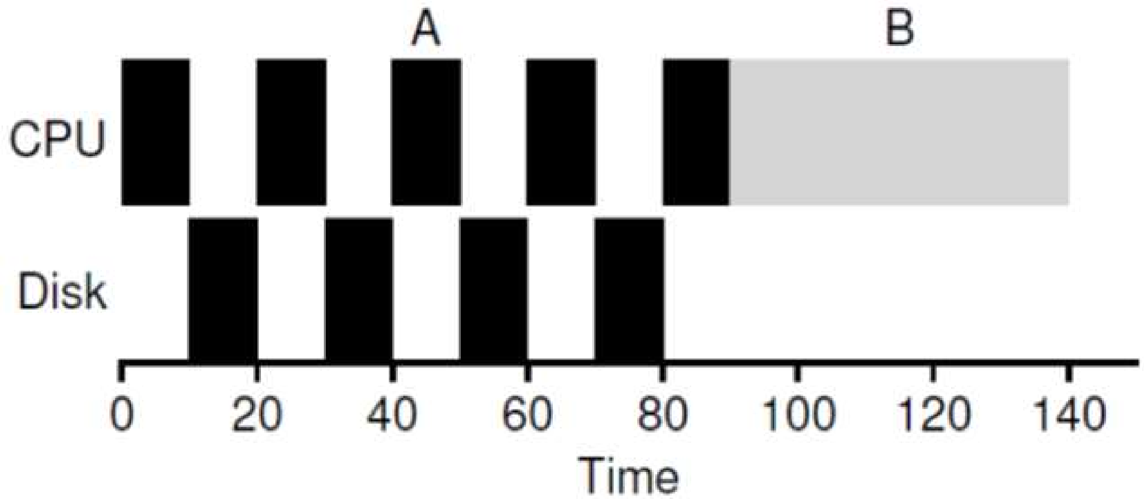


“time slice”

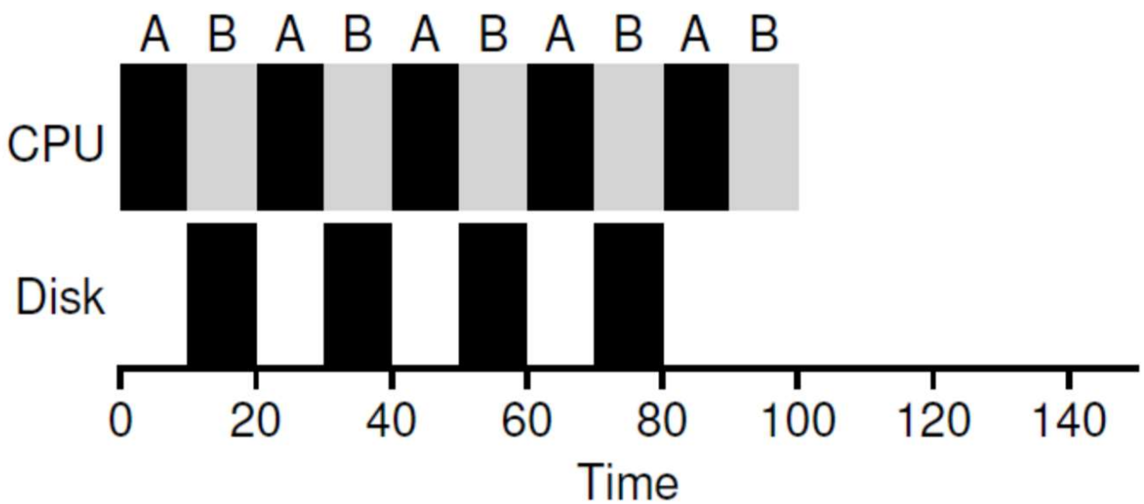
Average response time = (0 + 1 + 2) / 3 = **1**

Let’s relax assumption #5 – of course processes do I/O!

RR Plays Nicely with I/O, Too!!



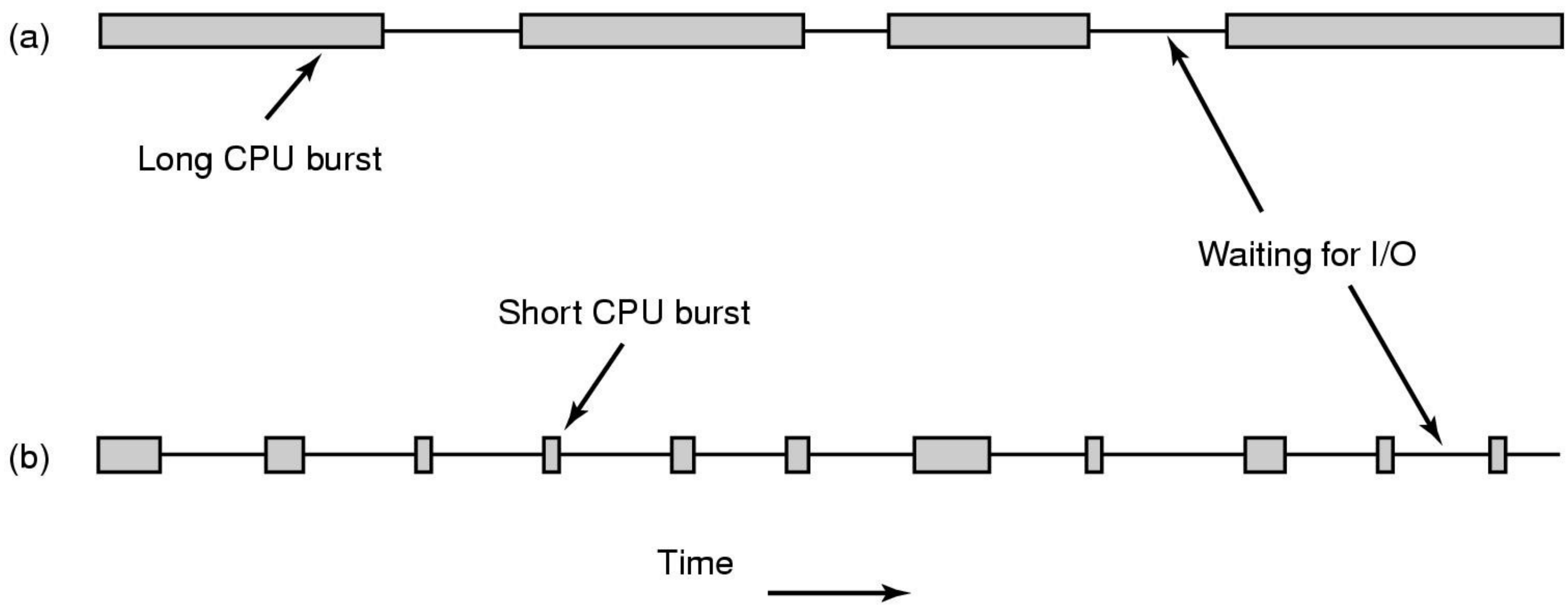
No Round Robin



Round Robin (with overlap)

How big should time slice be? What are tradeoffs?

Scheduling – Process Behavior



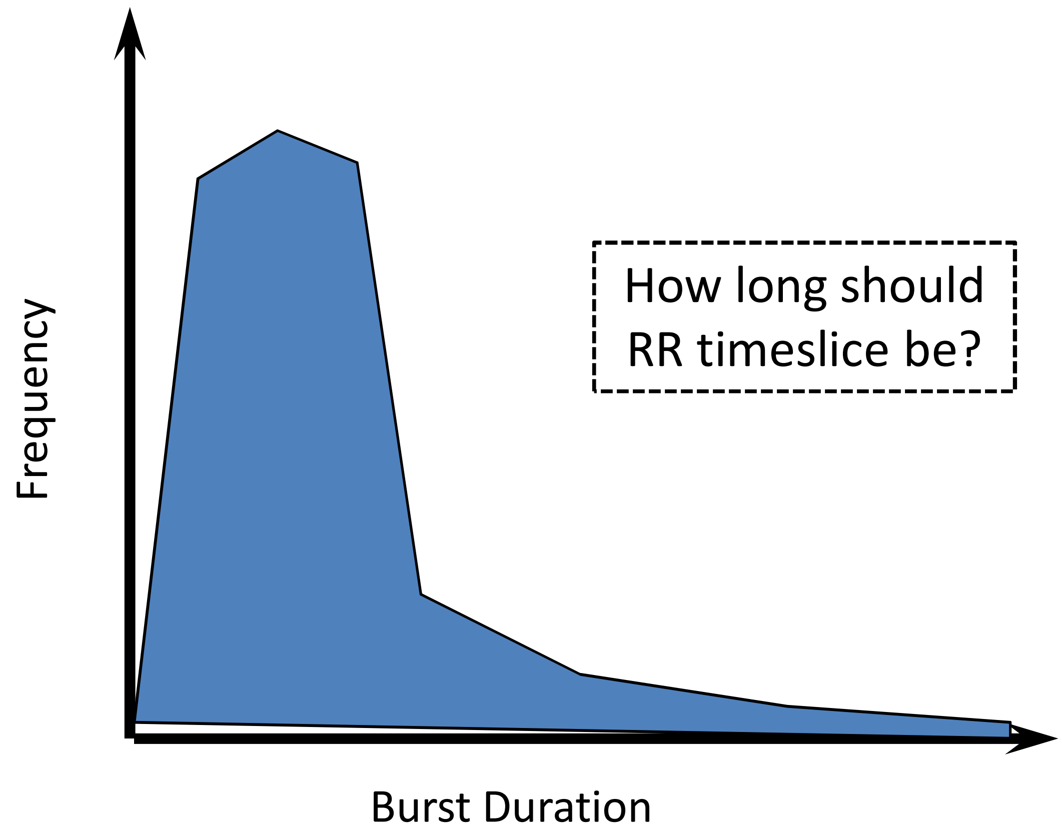
- Broadly, two kinds of processes
  a. CPU-bound
  b. I/O-bound

Which kind are there more of?

Scheduling – Process Behavior

I/O Bound Processes

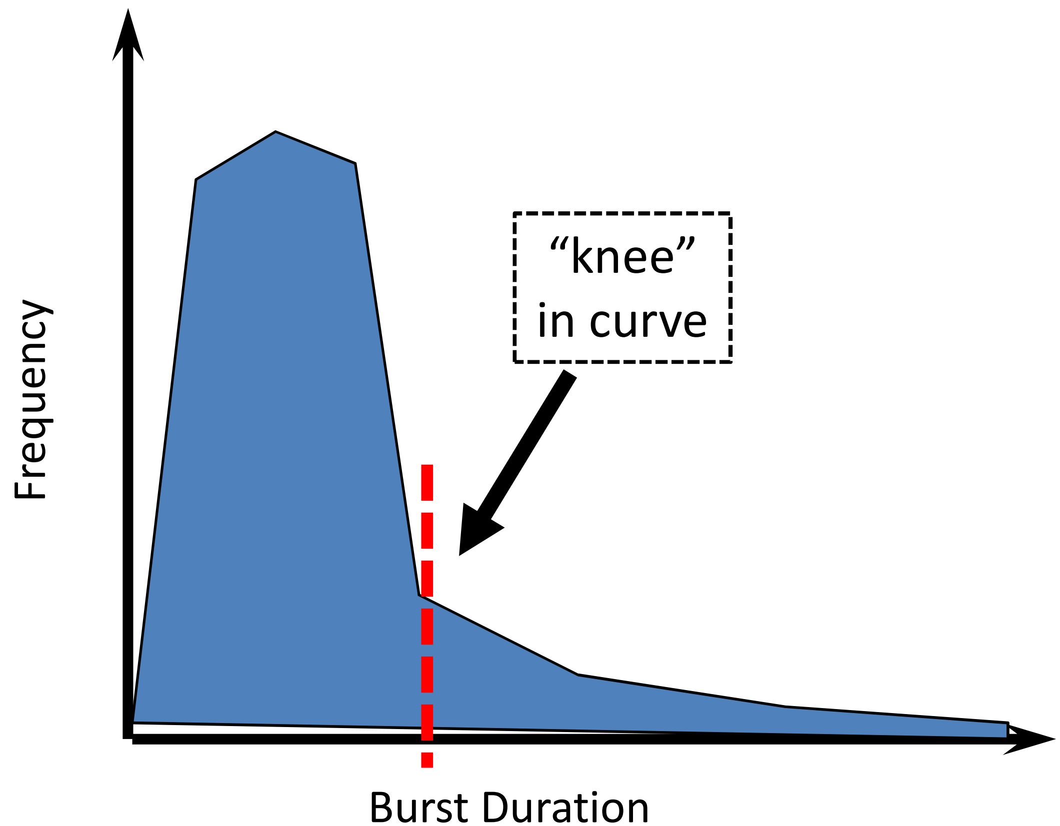
```
add
read
(I/O Wait)
store
increment
write
(I/O Wait)
```



Scheduling – Process Behavior

I/O Bound Processes

```
add
read
(I/O Wait)
store
increment
write
(I/O Wait)
```



Set timeslice so most I/O bound processes finish in once slice
Still protects against CPU bound!

SOS: Dispatcher

See: "`dispatcher.c`"

- What scheduling policy does it follow?
- There is no "`return`" from `Dispatcher()` … Why not?
  - Hint: think of the OS system stack
- There is a `while(1);` → This is an infinite loop! … Why is this ok?
  - Hint: consider other options

Outline

- Introduction (done)
- Scheduling Policies
  - FIFO (done)
  - SJF (done)
  - SCTF (done)
  - RR (done)
  - SOS (done)
  - MLFQ (next)
- Other topics

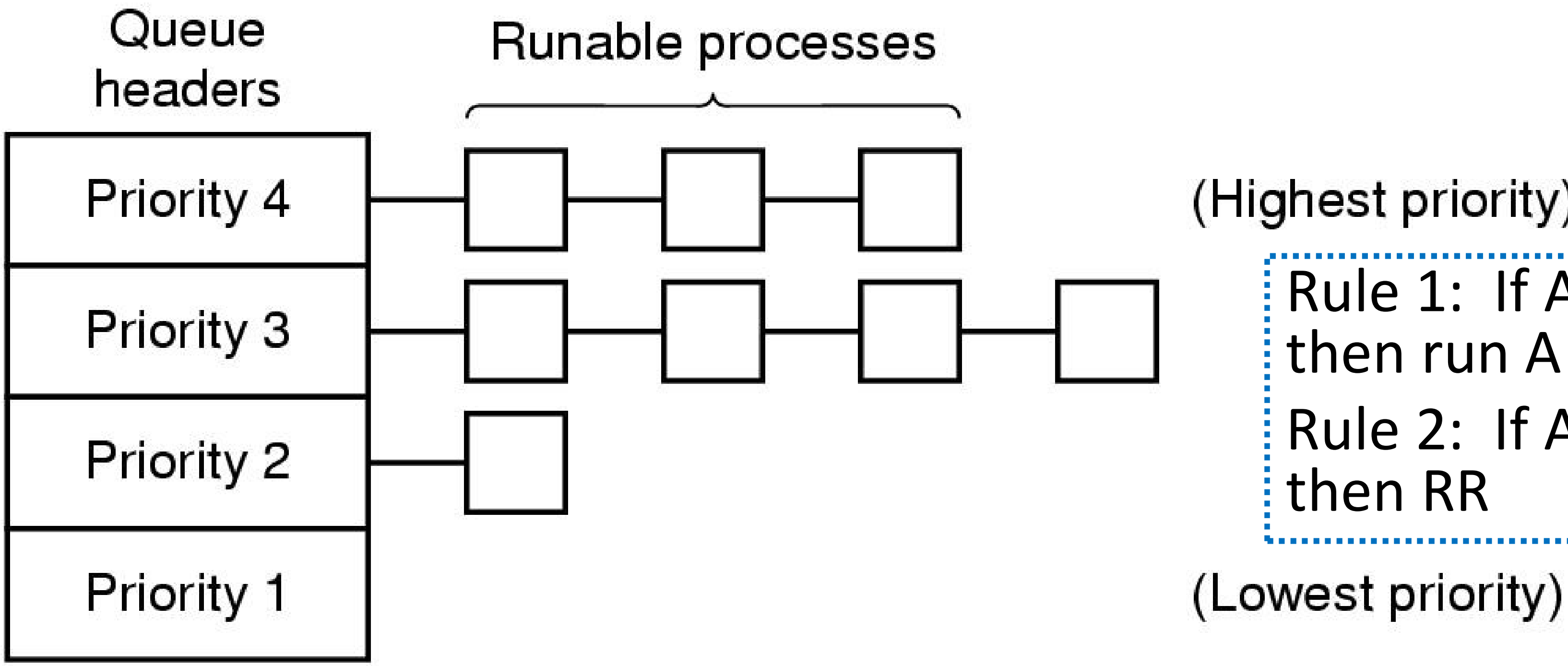
Priority Scheduling

- Want system that is responsive
  - User enters commands, gets feedback
- Want system that is efficient
  - Run processes to completion as quickly as possible

THE CRUX OF THE PROBLEM:
HOW TO SCHEDULE WITHOUT PERFECT KNOWLEDGE?

Minimize response time for interactive processes AND minimize turnaround time for higher throughput, without *a priori* knowledge about burst length?

Priorities via Multi-Level Queue



Rule 1: If A > B, then run A

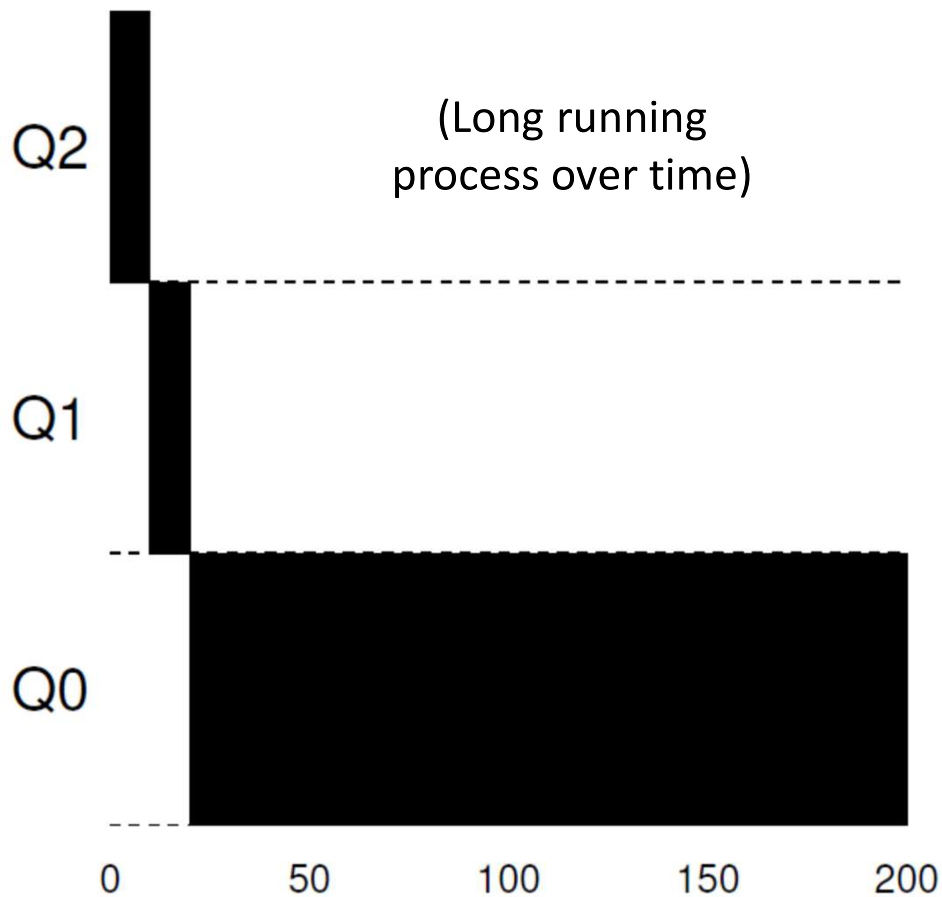
Rule 2: If A = B, then RR

- Put interactive processes in high priority
- Put long-running, CPU-bound processes in low priority
- But … how do we know this? What if process changes?

Need to "learn", adapt based on behavior (feedback)

Multi-level Feedback Queue

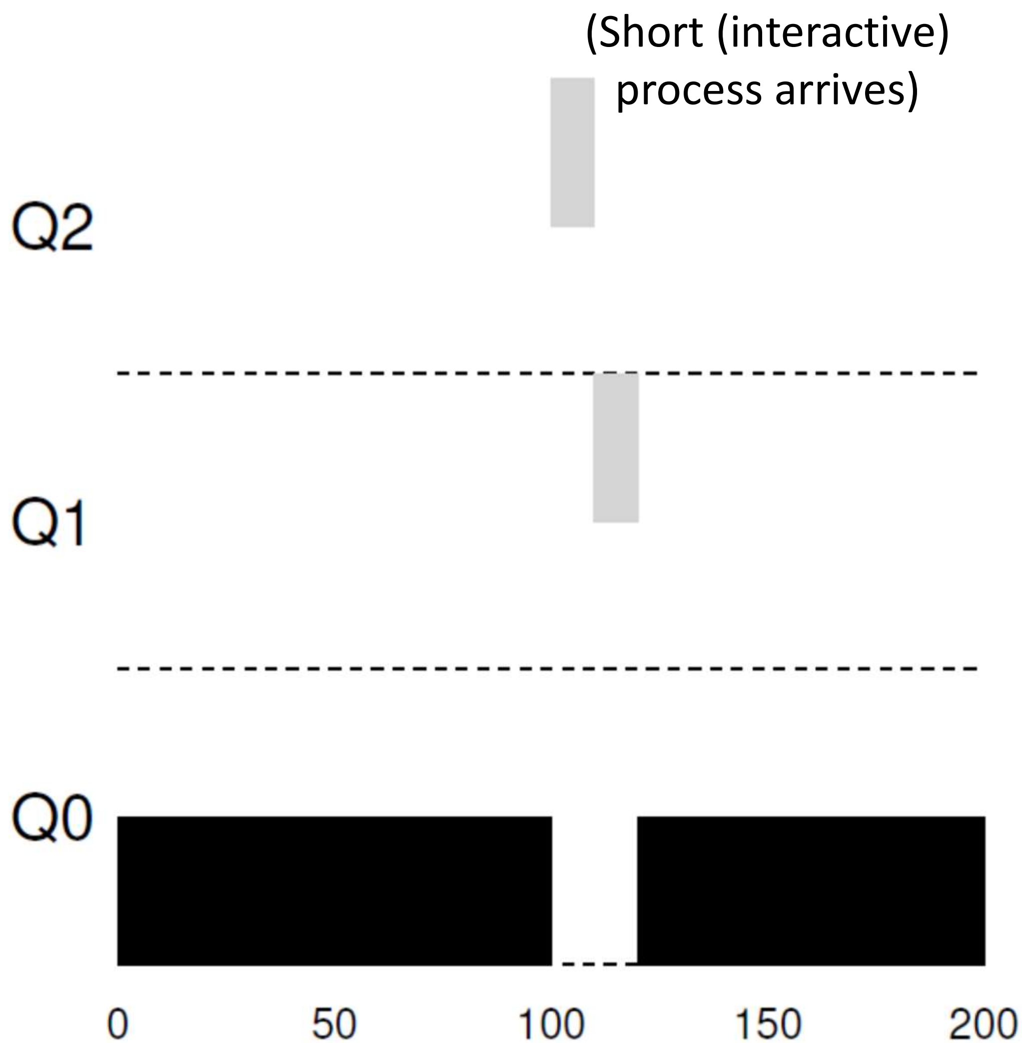
Adapt to Long Running Processes



Rule 3: New process at highest priority

Rule 4: If process uses all of slice, reduce priority

Prioritizes Short Processes

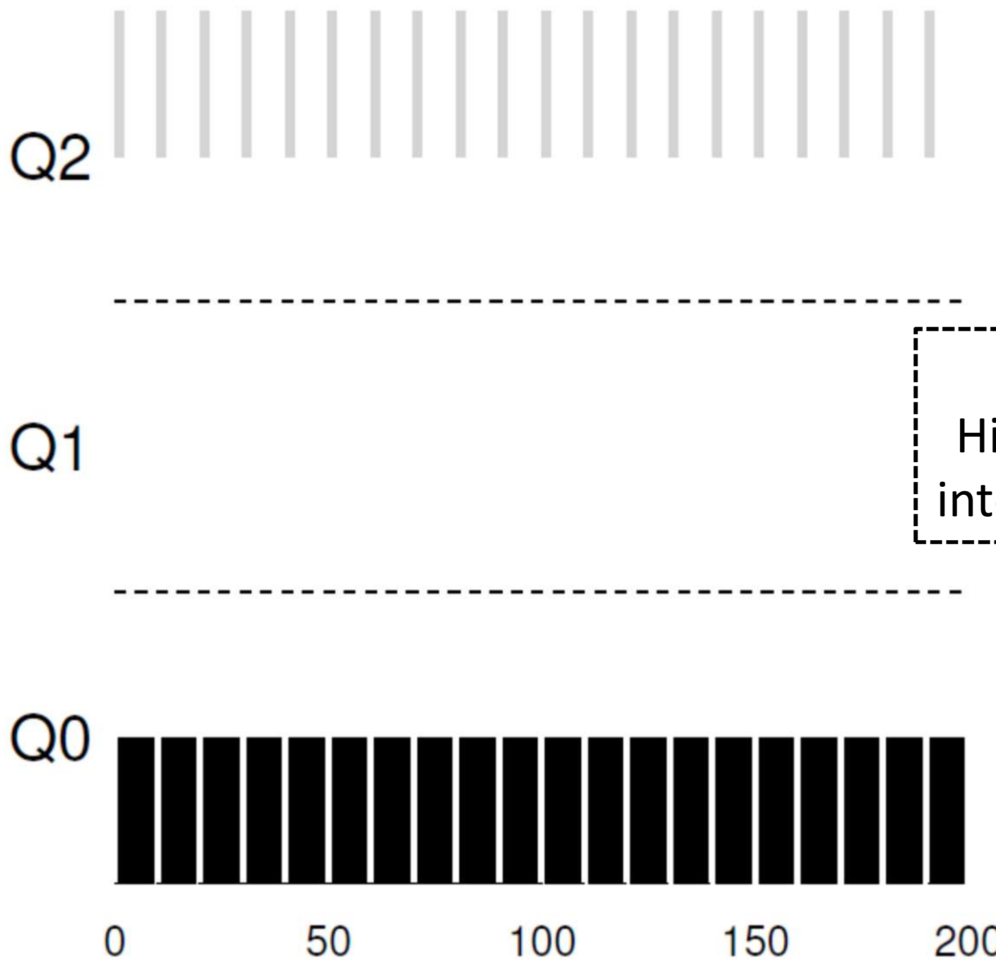


Rule 3: New process at highest priority

Rule 4: If process uses all of slice, reduce priority

Supports I/O-Bound Processes

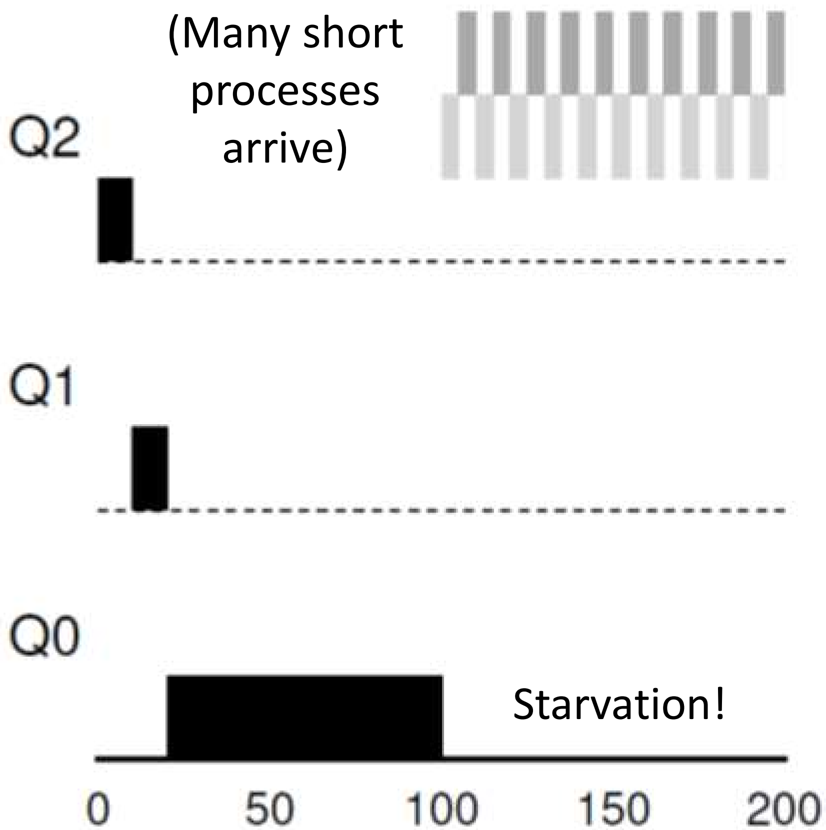
(I/O Bound process makes progress)



Problems?
Hint: think of many interactive processes

(Doesn't interfere with CPU-bound process much)

I’m Starving!

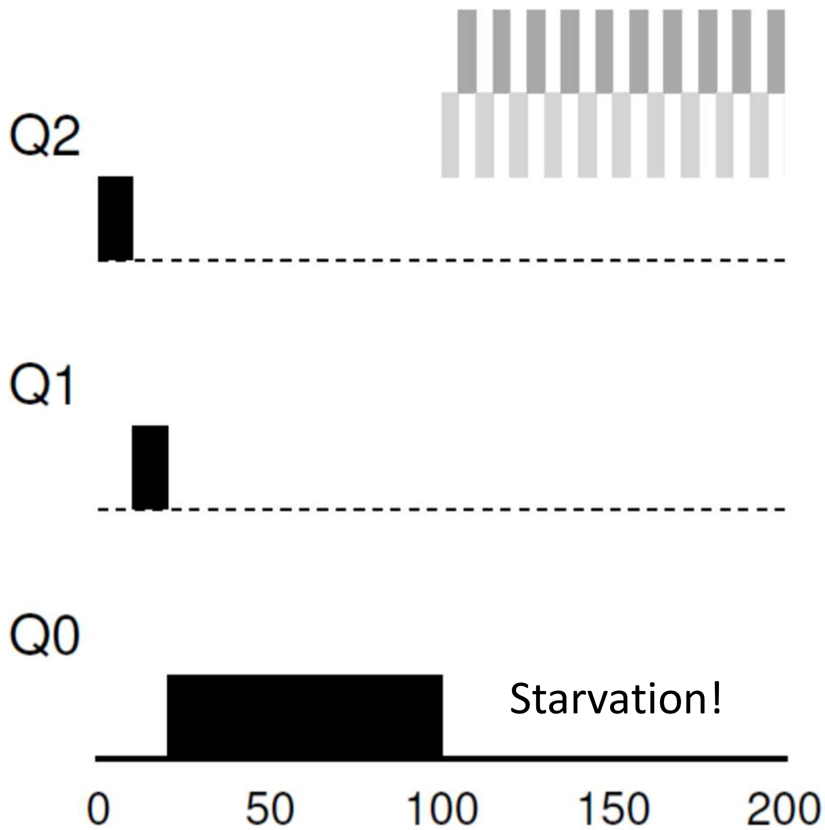


- Process may *never* get CPU (aka “starvation”)
- And may have changed!
  - Was CPU-bound
  - Now I/O-bound

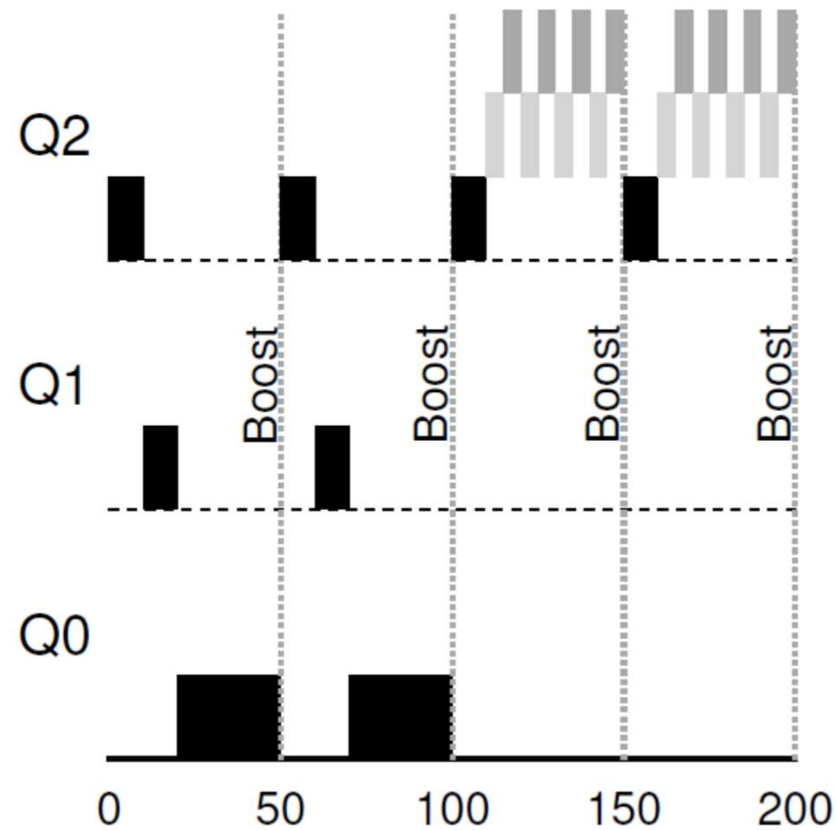
Fixes?
Hint: movement does not have to be one-way

Gimme a Boost!

Rule 5: after some time, all processes move to top

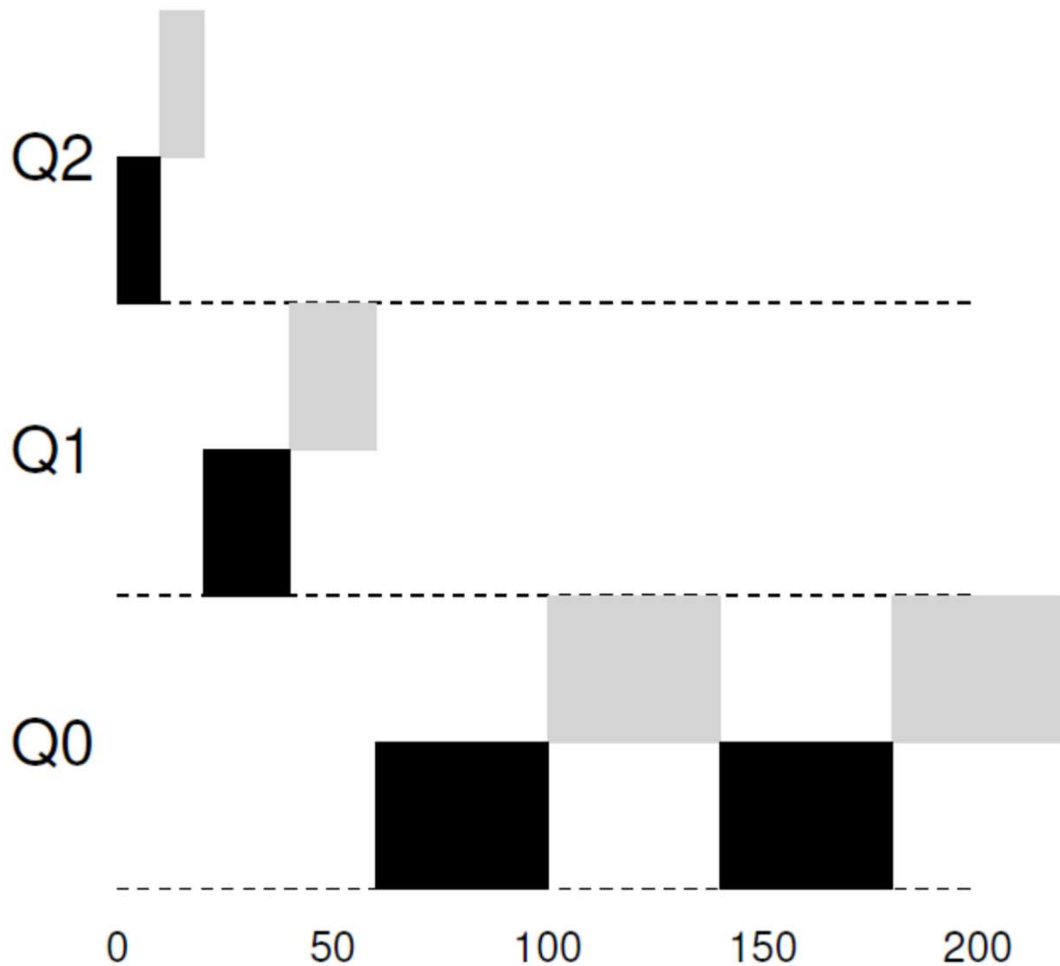


No boost



Boost

Tuning Possible – e.g., Different Quanta Sizes for Improved Throughput



Rule 6: timeslice inversely proportional to priority

- Lots more possibilities!
  - Move up one level
  - Not RR for some queues …

Other Scheduling Topics

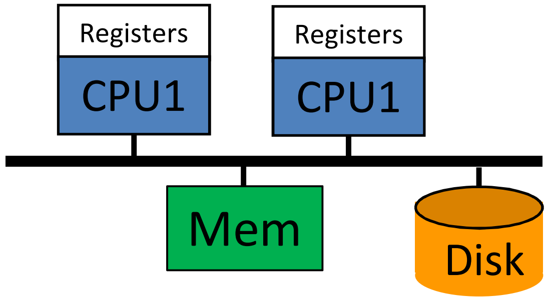
- Linux
  - Good overview
  - Details
    - Completely Fair Scheduler
    - `sched_fair.c`
- Windows
  - Multi-level feedback queue
  - Starvation prevention
  - Details
- Multiprocessors
  -

Chapter10
OPERATING SYSTEMS: THREE EASY PIECES
By Arpaci-Dusseau and Arpaci-Dusseau

http://www.cs.montana.edu/~chandrima.sarkar/AdvancedOS/SchedulingLinux/index.html

https://en.wikipedia.org/wiki/Completely_Fair_Scheduler

https://www.microsoftpressstore.com/articles/article.aspx?p=2233328&seqNum=7



Outline

- Introduction (done)
- Scheduling Policies (done)
  - FIFO (done)
  - SJF (done)
  - SCTF (done)
  - RR (done)
  - SOS (done)
  - MLFQ (done)
- Other Topics (done)