[537] Schedulers

Tyler Harter
9/10/14

Overview

Review processes

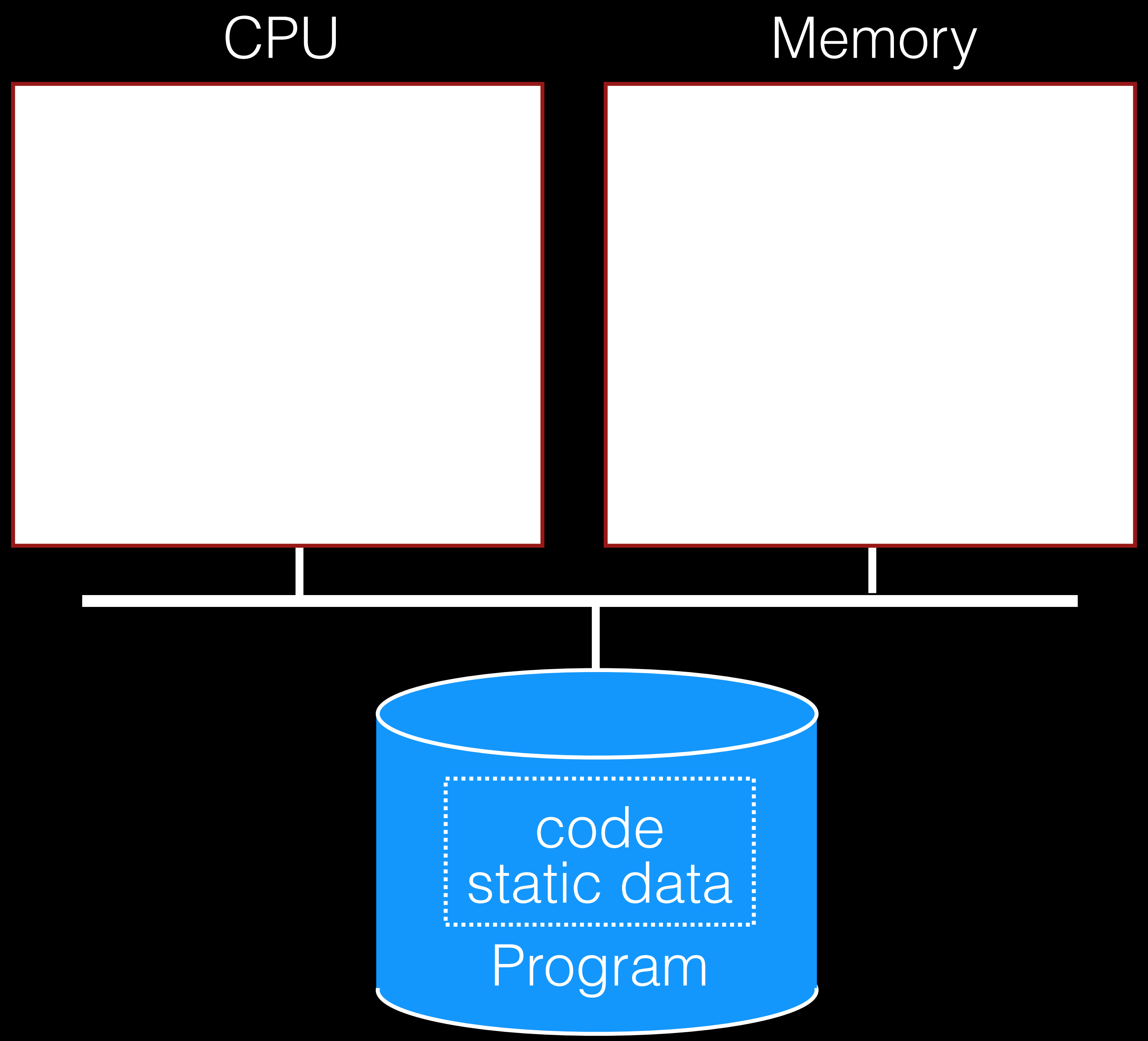
Workloads, schedulers, and metrics (Chapter 7)

A general purpose scheduler, MLFQ (Chapter 8)

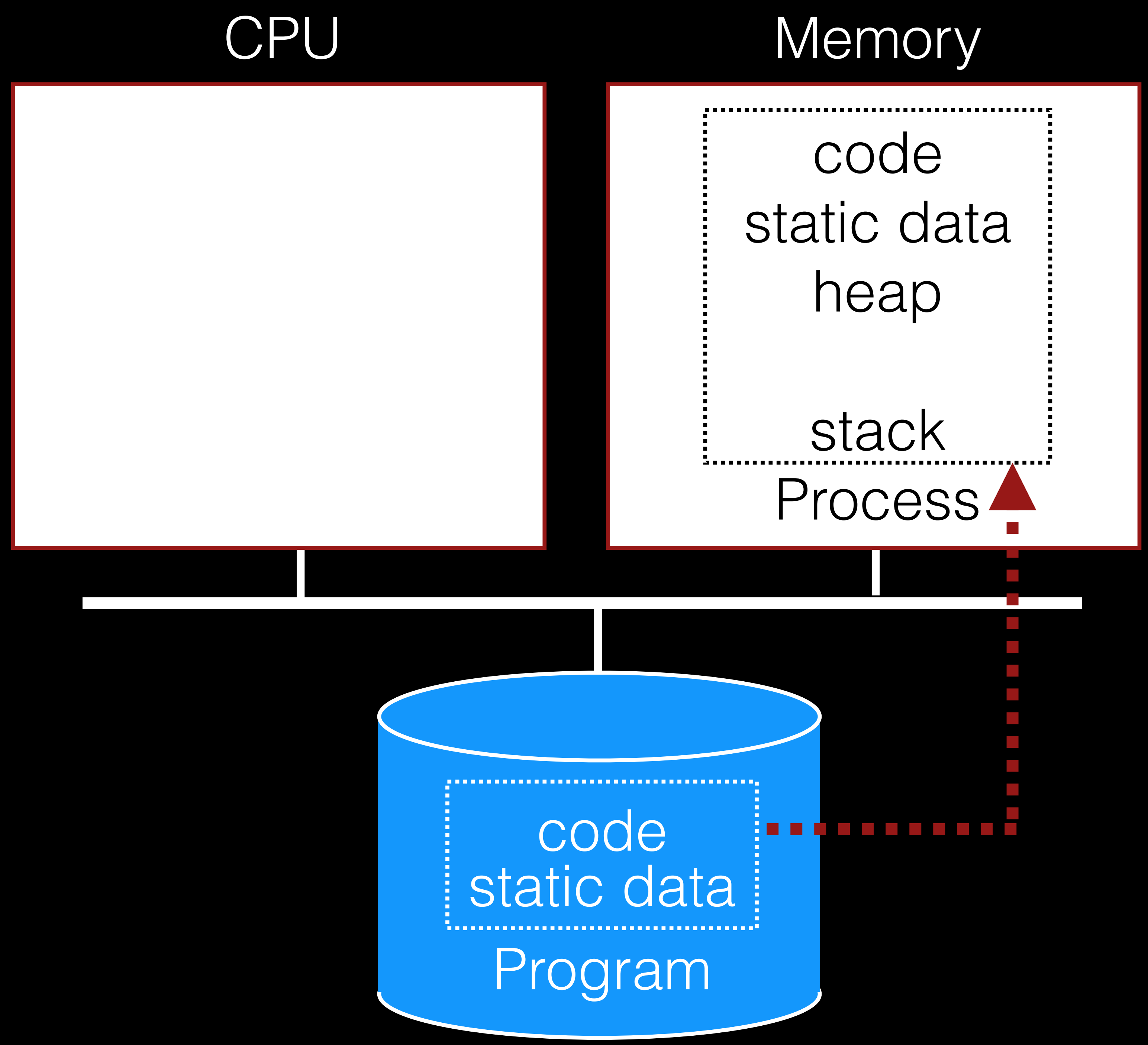
Lottery scheduling (Chapter 9)

Review: Processes

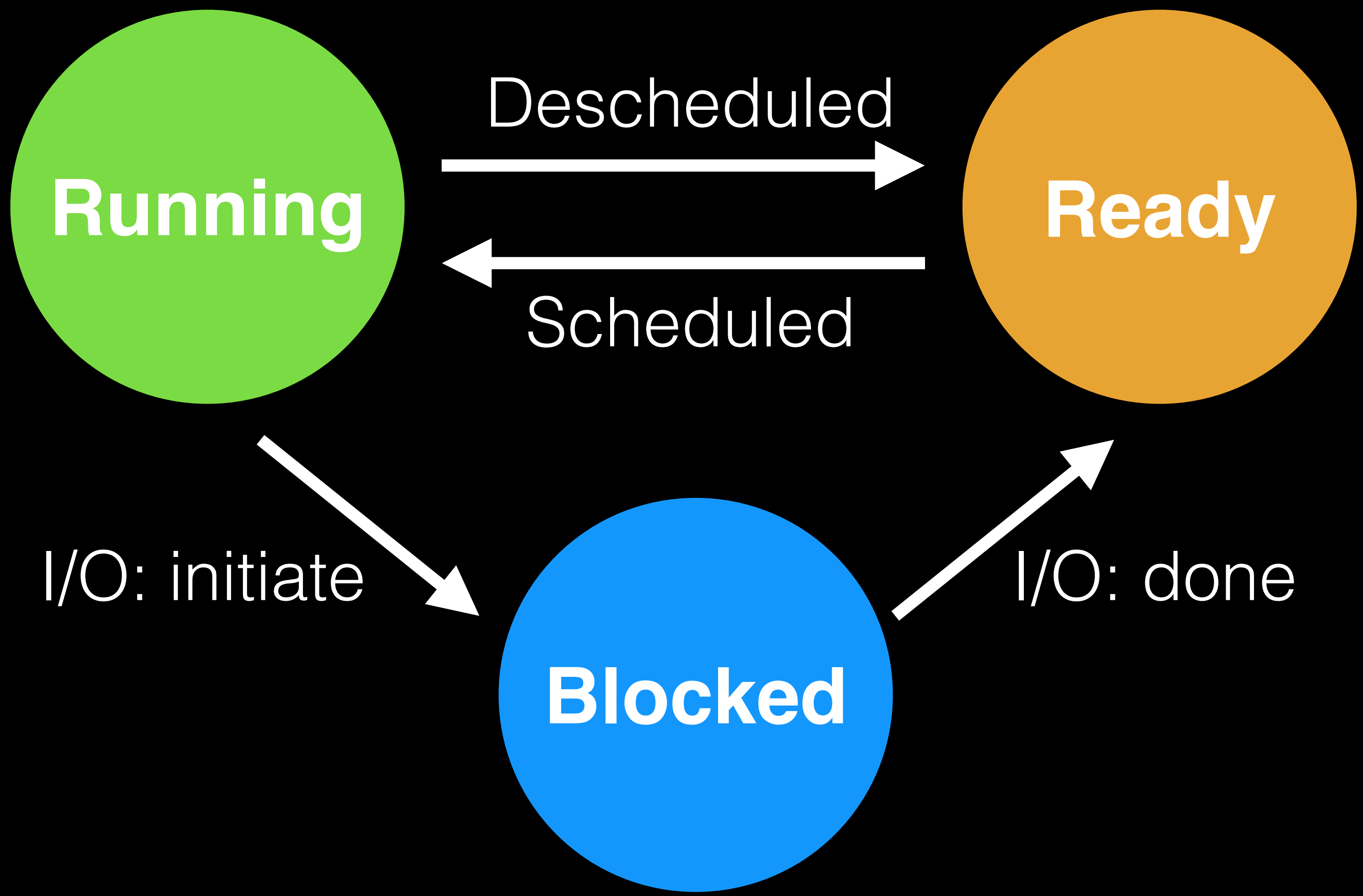
Process Creation



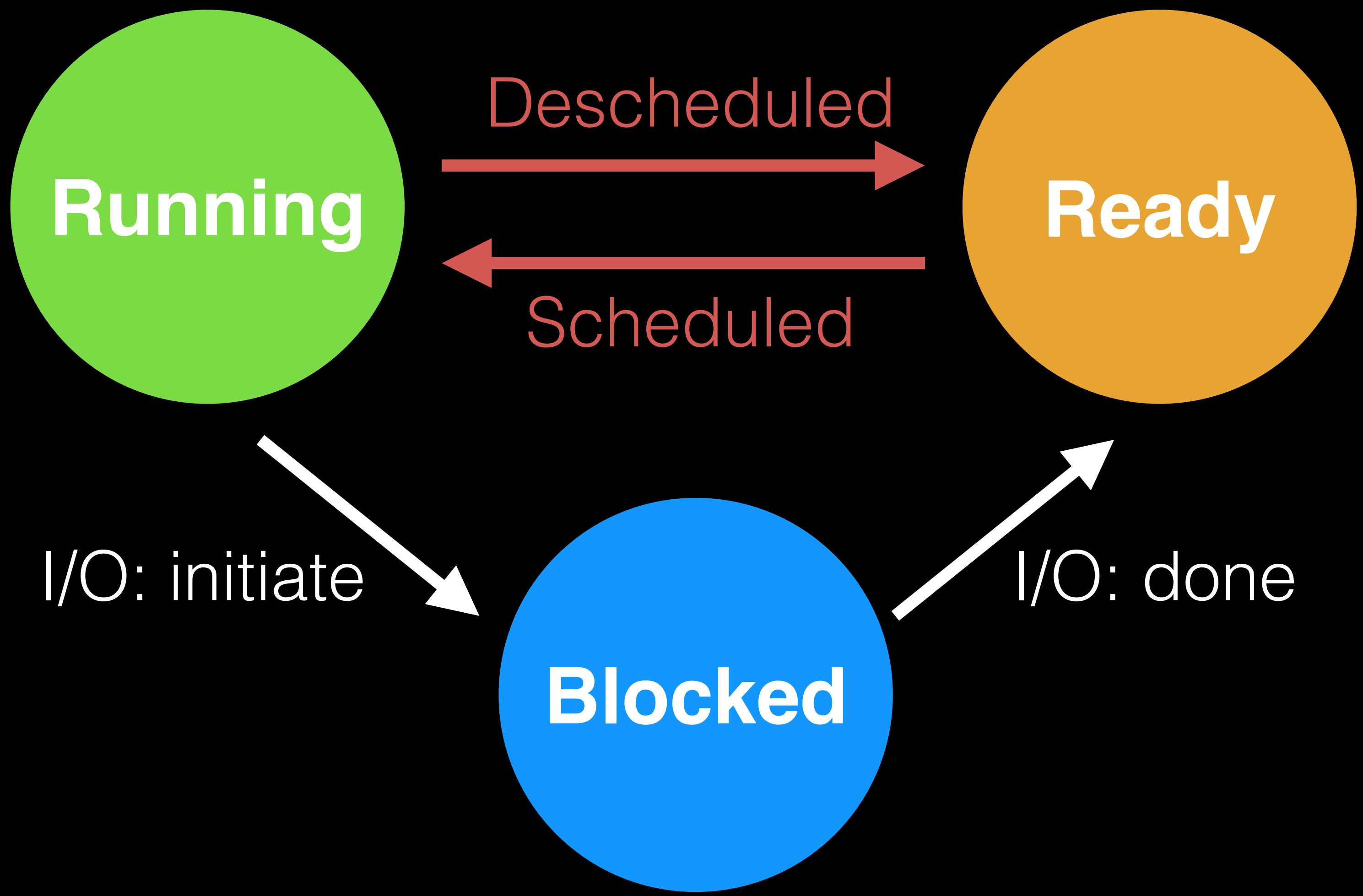
Process Creation



State Transitions



How to transition? ("mechanism")
When to transition? ("policy")



```c
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kern stack for this proc
  enum procstate state;        // Process state
  volatile int pid;            // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

```c
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kern stack for this proc
  enum procstate state;        // Process state
  volatile int pid;            // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

Operating System	Hardware	Program
		Process A …

Operating System	Hardware	Program
		Process A …
	timer interrupt save regs(A) to k-stack(A) move to kernel mode jump to trap handler	

Operating System	Hardware	Program
		Process A …
	timer interrupt save regs(A) to k-stack(A) move to kernel mode jump to trap handler	
Handle the trap Call **switch()** routine save regs(A) to proc-struct(A) restore regs(B) from proc-struct(B) switch to k-stack return-from-trap (into B)		

Operating System	Hardware	Program
		Process A …
	timer interrupt save regs(A) to k-stack(A) move to kernel mode jump to trap handler	
Handle the trap Call **switch()** routine save regs(A) to proc-struct(A) restore regs(B) from proc-struct(B) switch to k-stack return-from-trap (into B)		
	restore regs(B) from k-stack(B) move to user mode jump to B’s IP	

Operating System	Hardware	Program
		Process A …
	timer interrupt save regs(A) to k-stack(A) move to kernel mode jump to trap handler	
Handle the trap Call **switch()** routine save regs(A) to proc-struct(A) restore regs(B) from proc-struct(B) switch to k-stack return-from-trap (into B)		
	restore regs(B) from k-stack(B) move to user mode jump to B's IP	
		Process B …

Basic Schedulers

Vocabulary

Workload: set of job descriptions

Scheduler: logic that decides when jobs run

Metric: measurement of scheduling quality

Vocabulary

Workload: set of job descriptions

Scheduler: logic that decides when jobs run

Metric: measurement of scheduling quality

Scheduler “algebra”, given 2 variables, find the 3rd:

$$f(\mathbf{W}, \mathbf{S}) = \mathbf{M}$$

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. The run-time of each job is known

Scheduling Basics

Workloads:
arrival_time
run_time

Schedulers:
FIFO
SJF
STCF
RR

Metrics:
turnaround_time
response_time

Example: workload, scheduler, metric

JOB	arrival_time (s)	run_time (s)
A	0.0001	10
B	0.0002	10
C	0.0003	10

FIFO: First In, First Out (run jobs in *arrival_time* order)

What is our turnaround?: *completion_time - arrival_time*

Example: workload, scheduler, metric

JOB	arrival_time (s)	run_time (s)
A	~0	10
B	~0	10
C	~0	10

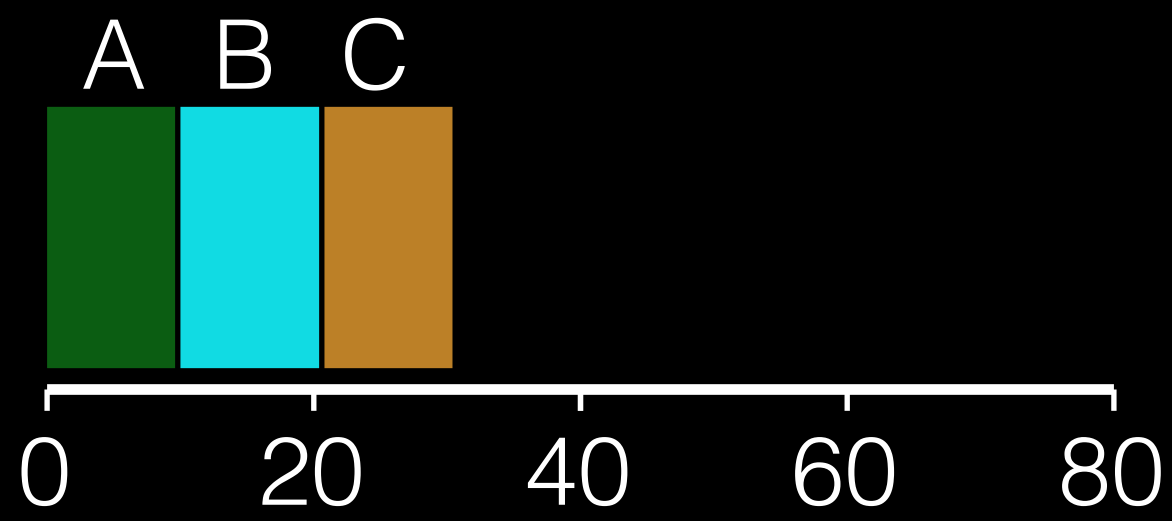
FIFO: First In, First Out (run jobs in *arrival_time* order)

What is our turnaround?: *completion_time* - *arrival_time*

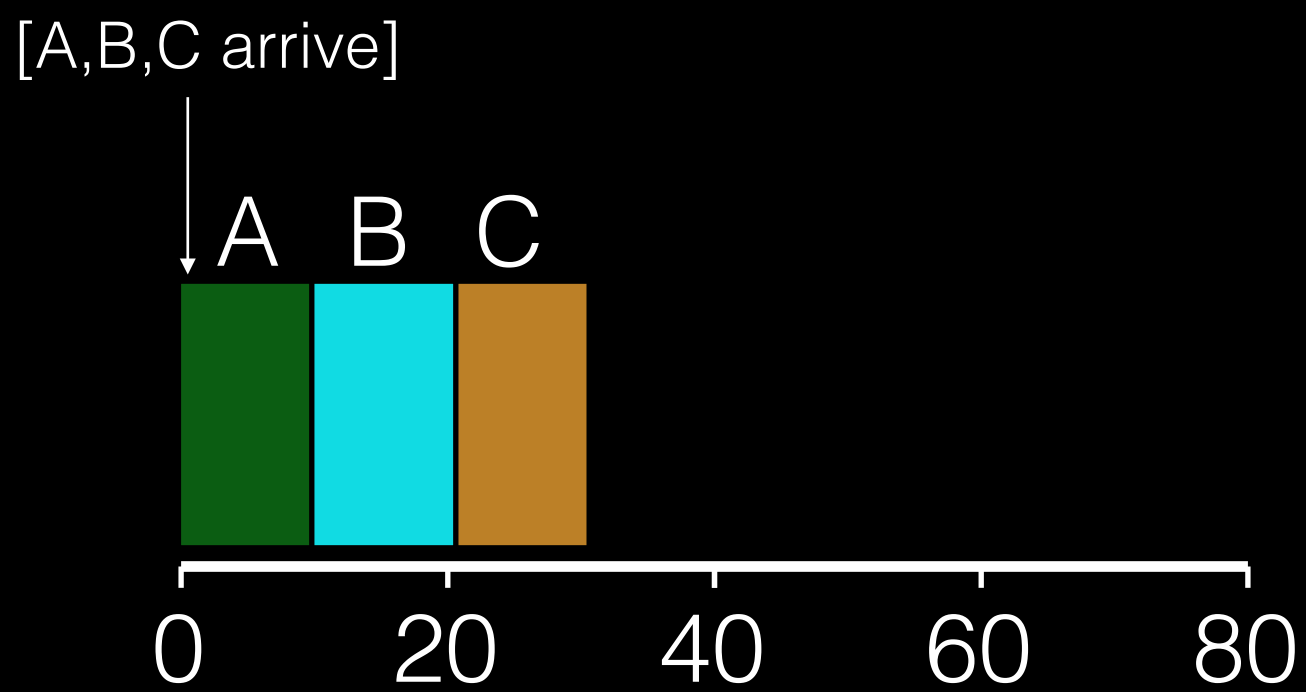
Event Trace

Time	Event
0	A arrives
0	B arrives
0	C arrives
0	run A
10	complete A
10	run B
20	complete B
20	run C
30	complete C

Trace Visualization



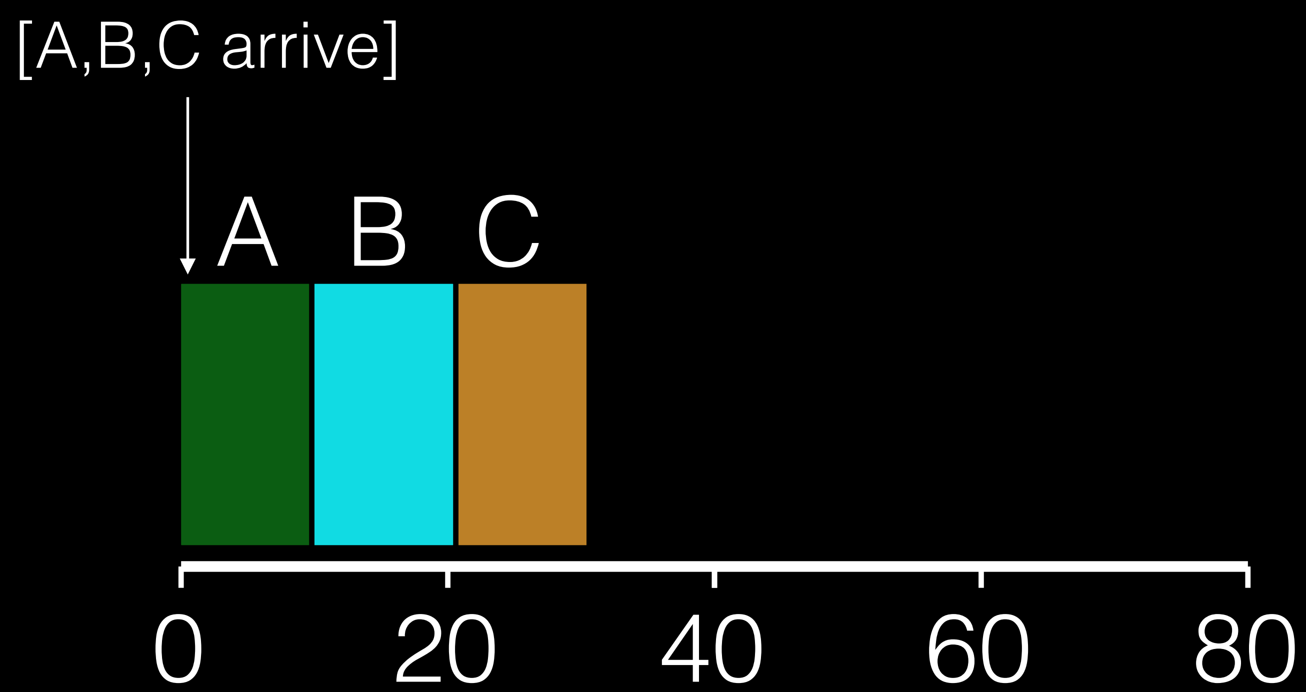
Trace Visualization



What is the average turnaround time? (Q1)

Def: *turnaround_time = completion_time - arrival_time*

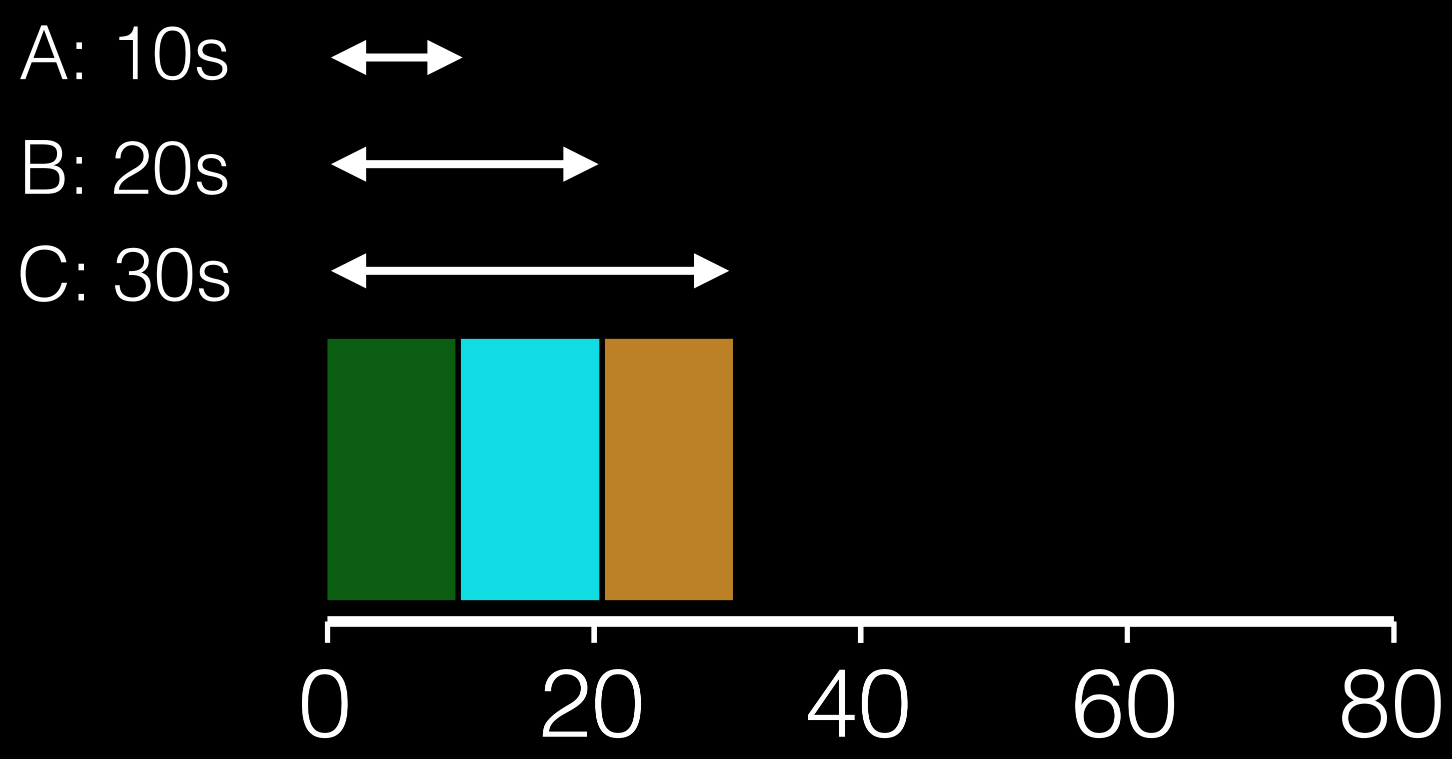
Trace Visualization



What is the average turnaround time? (Q1)

Def: *turnaround_time = completion_time - arrival_time*

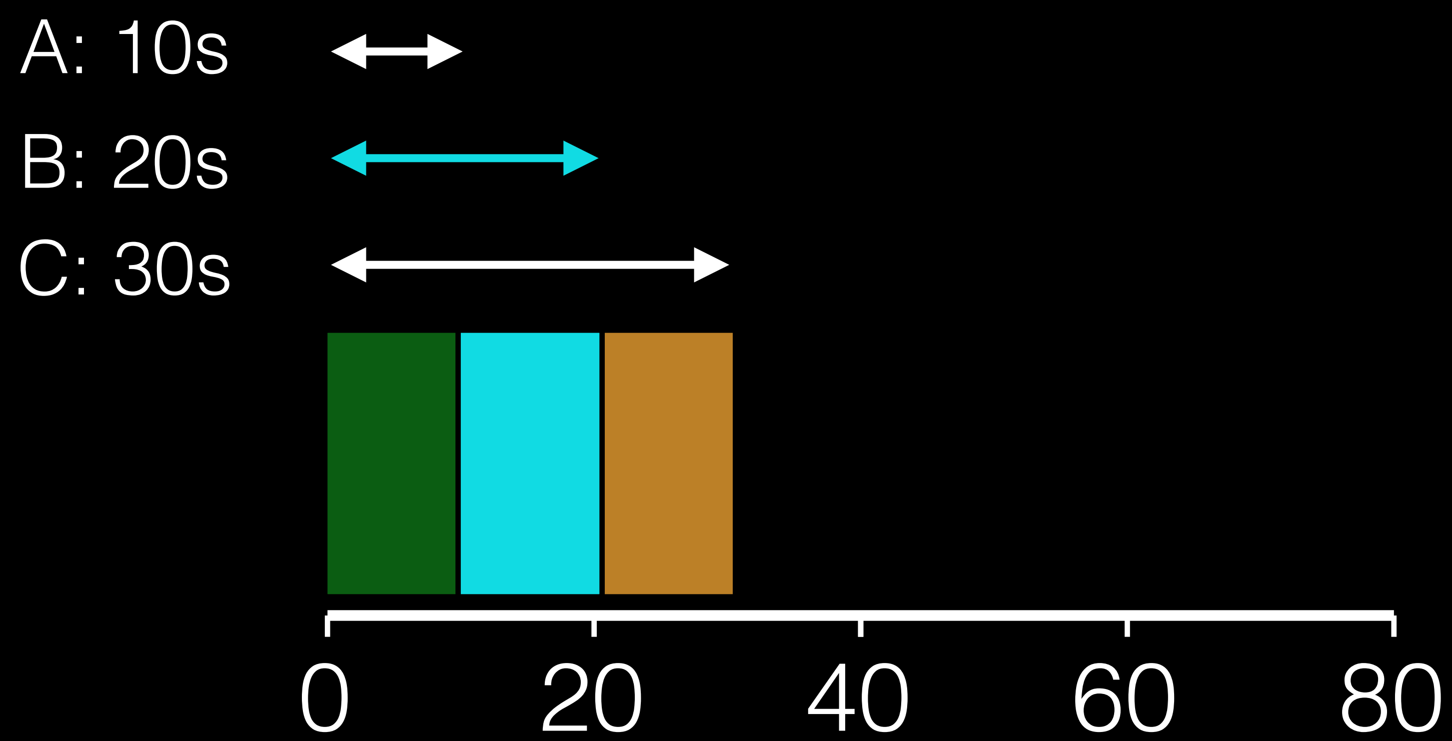
Trace Visualization



What is the average turnaround time? (Q1)

Def: *turnaround_time = completion_time - arrival_time*

Trace Visualization



What is the average turnaround time? (Q1)

(10 + 20 + 30) / 3 = **20s**

Scheduling Basics

Workloads:
- arrival_time
- run_time

Schedulers:
- FIFO
- SJF
- STCF
- RR

Metrics:
- turnaround_time
- 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. The 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. The run-time of each job is known

“Solve” for W

f(**W**, **S**) = **M**

Workload: ?

Scheduler: FIFO

Metric: turnaround is high

Example: Big First Job

JOB	arrival_time (s)	run_time (s)
A	~0	**60**
B	~0	10
C	~0	10

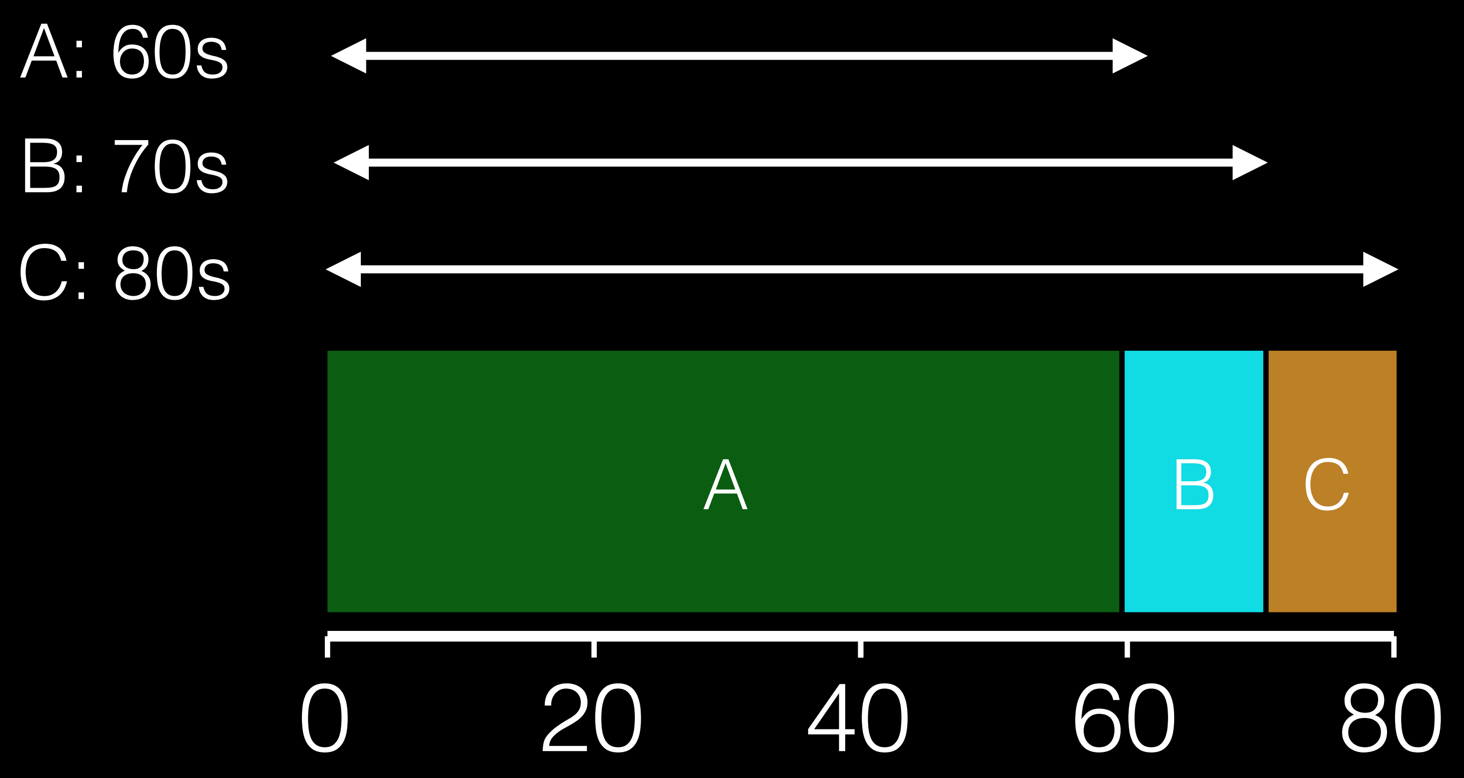
What is the average turnaround time? (Q2)

Example: Big First Job

JOB	arrival_time (s)	run_time (s)
A	~0	**60**
B	~0	10
C	~0	10

What is the average turnaround time? (Q2)

Example: Big First Job



Average turnaround time: **70s**

Convoy Effect

Passing the Tractor

New scheduler: SJF (Shortest Job First)

Policy: when deciding what job to run next,
choose the one with smallest *run_time*

Example: Shortest Job First

JOB	arrival_time (s)	run_time (s)
A	~0	**60**
B	~0	10
C	~0	10

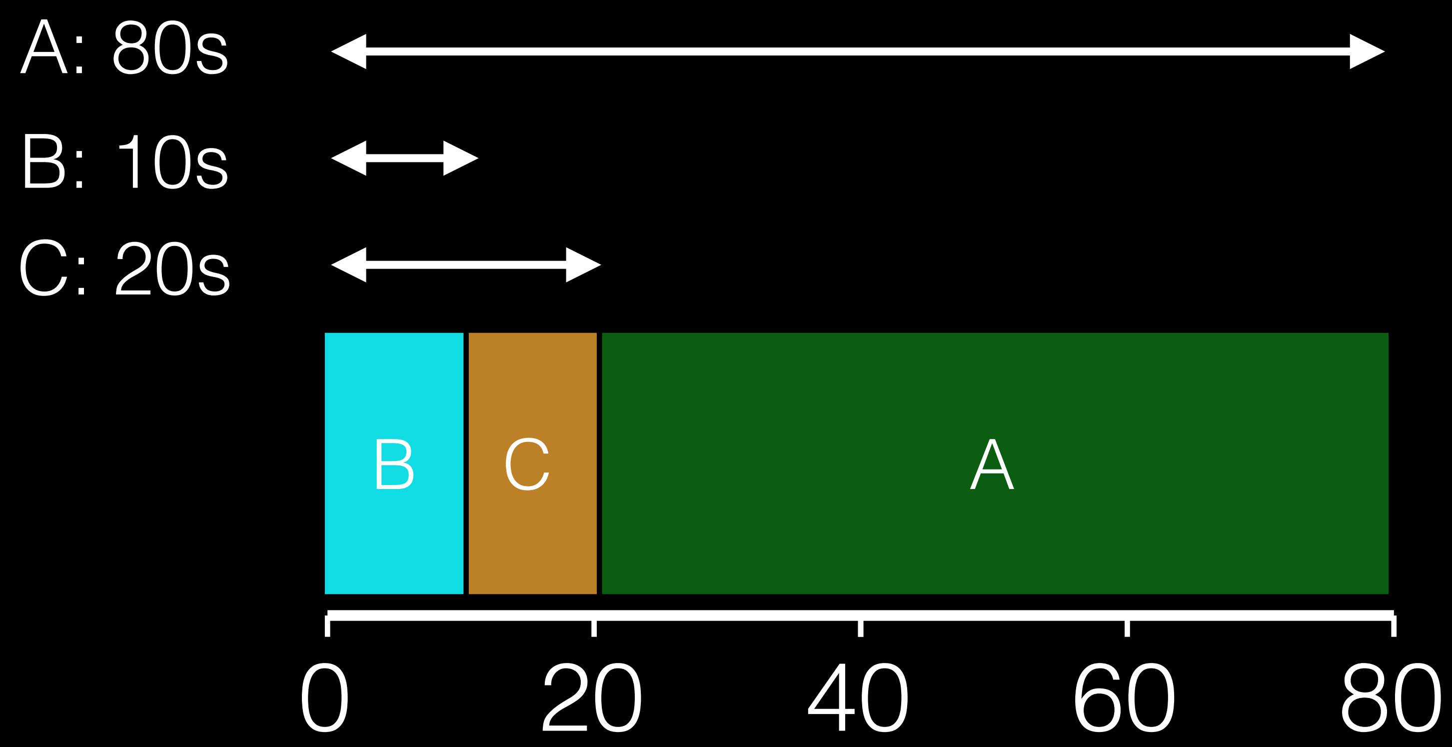
What is the average turnaround time with SJF? (Q3)

Example: Shortest Job First

JOB	arrival_time (s)	run_time (s)
A	~0	**60**
B	~0	10
C	~0	10

What is the average turnaround time with SJF? (Q3)

Q3 Answer



What is the average turnaround time with SJF? (Q3)

(80 + 10 + 20) / 3 = **~36.7s**

Scheduling Basics

Workloads:
- arrival_time
- run_time

Schedulers:
- FIFO
- SJF
- STCF
- RR

Metrics:
- turnaround_time
- 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. The 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. The run-time of each job is known

Shortest Job First (Arrival Time)

JOB	arrival_time (s)	run_time (s)
A	~0	60
B	**~10**	10
C	**~10**	10

What is the average turnaround time with SJF?

Stuck Behind a Tractor Again

What is the average turnaround time?

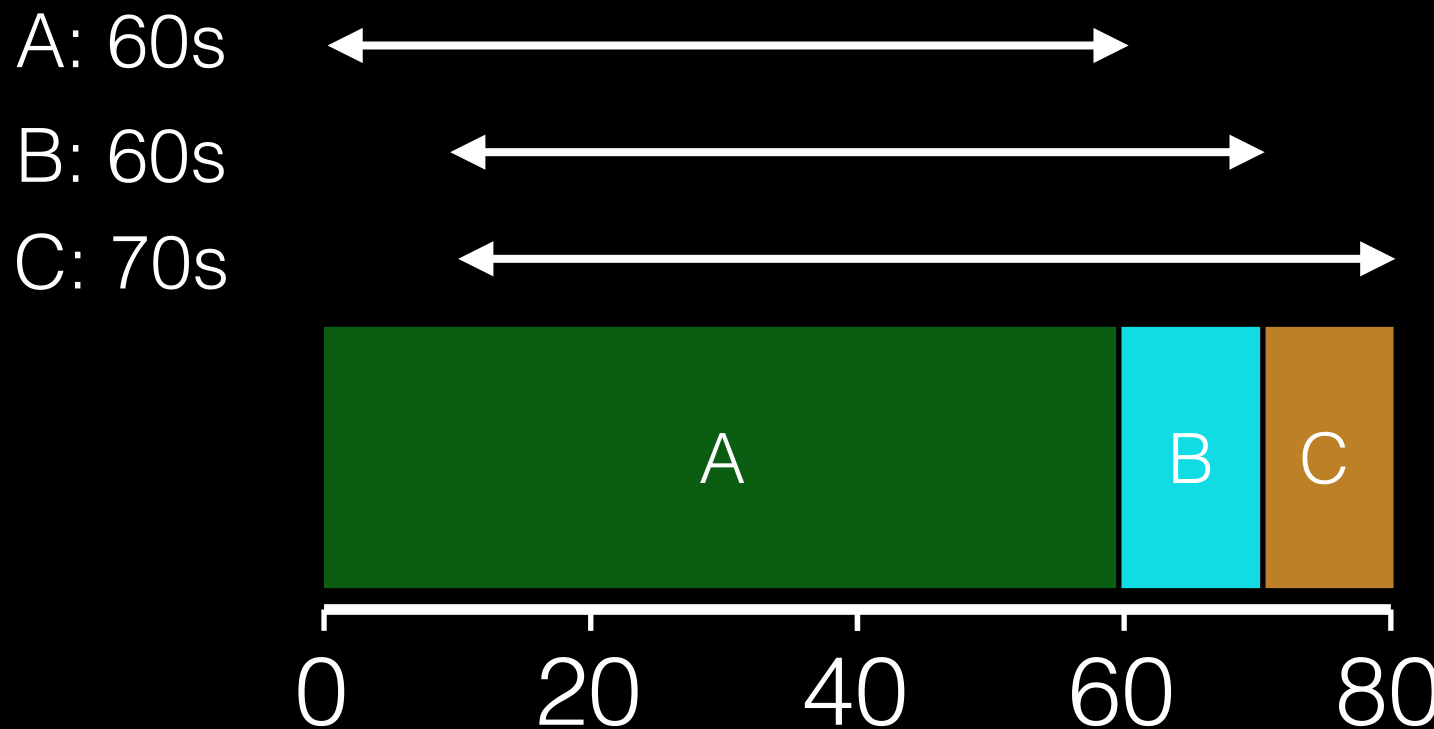
(Q4)

Stuck Behind a Tractor Again

What is the average turnaround time?

(Q4)

Stuck Behind a Tractor Again



What is the average turnaround time?

(60 + 60 + 70) / 3 = **63.3s**

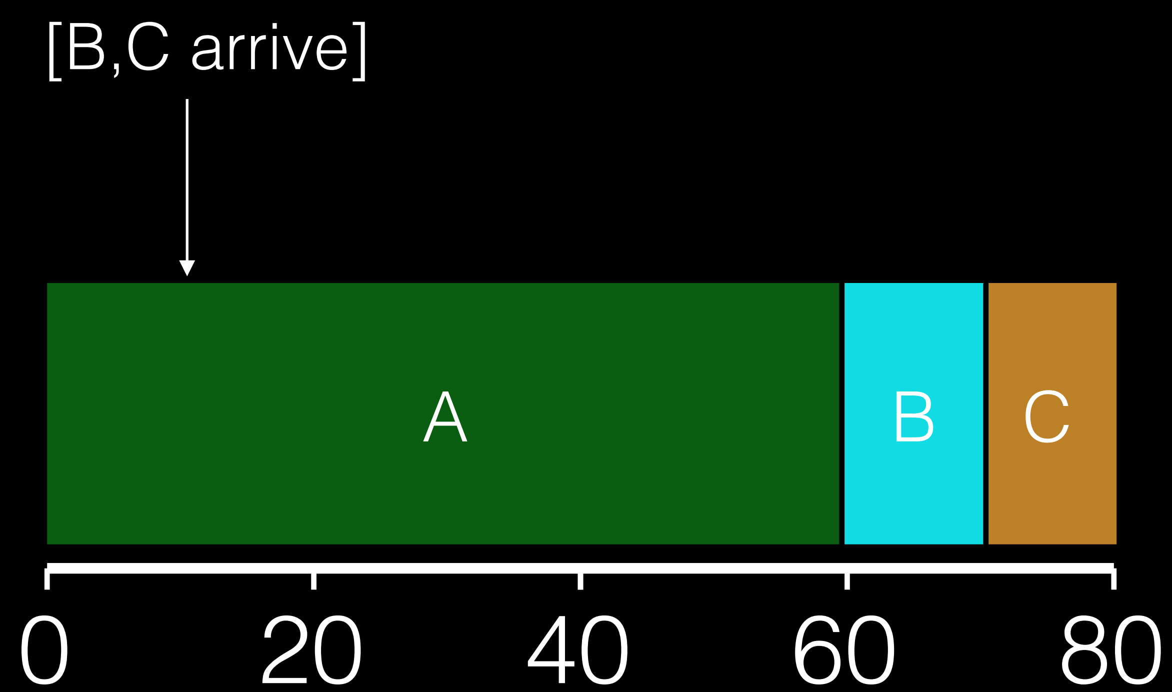
A Preemptive Scheduler

Prev schedulers: FIFO and SJF are non-preemptive

New scheduler: STCF (Shortest Time-to-Completion First)

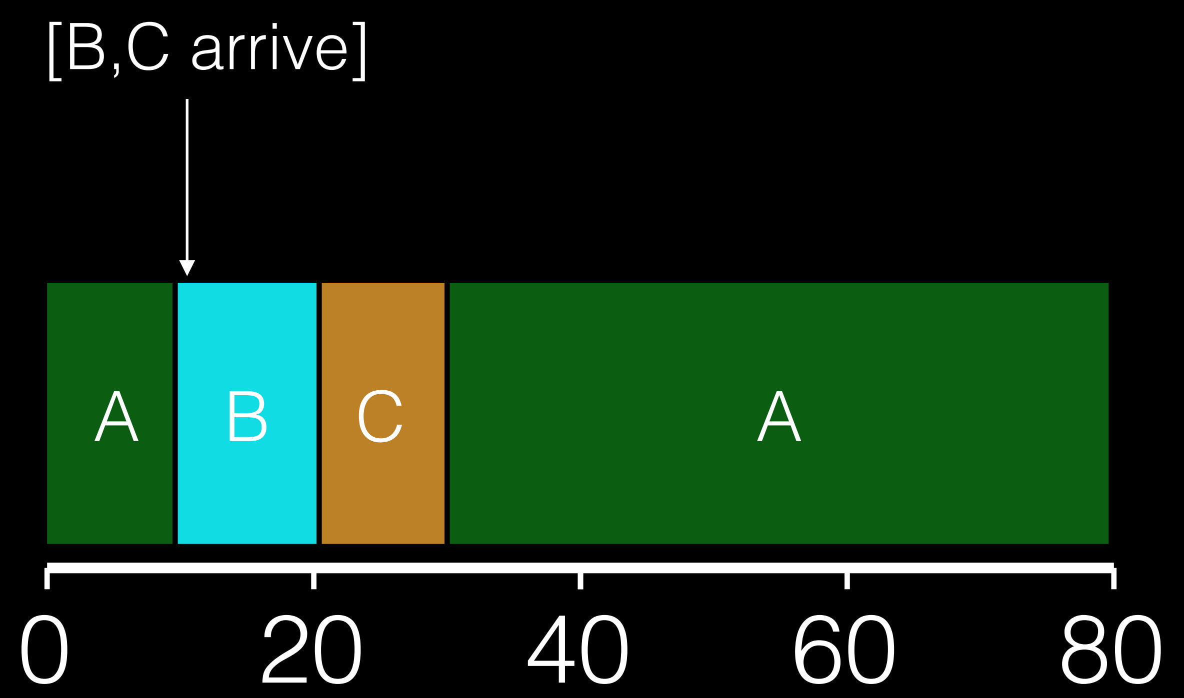
Policy: switch jobs so we always run the one that
will complete the quickest

SJF



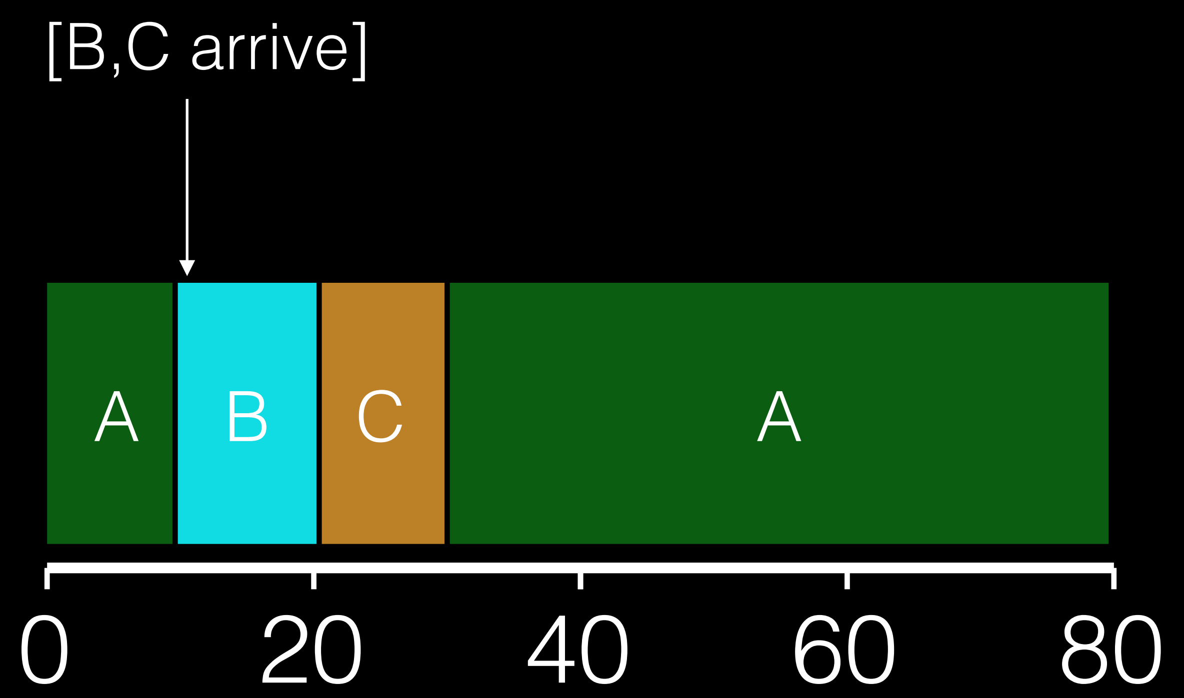
Average turnaround time: **70s**

STCF



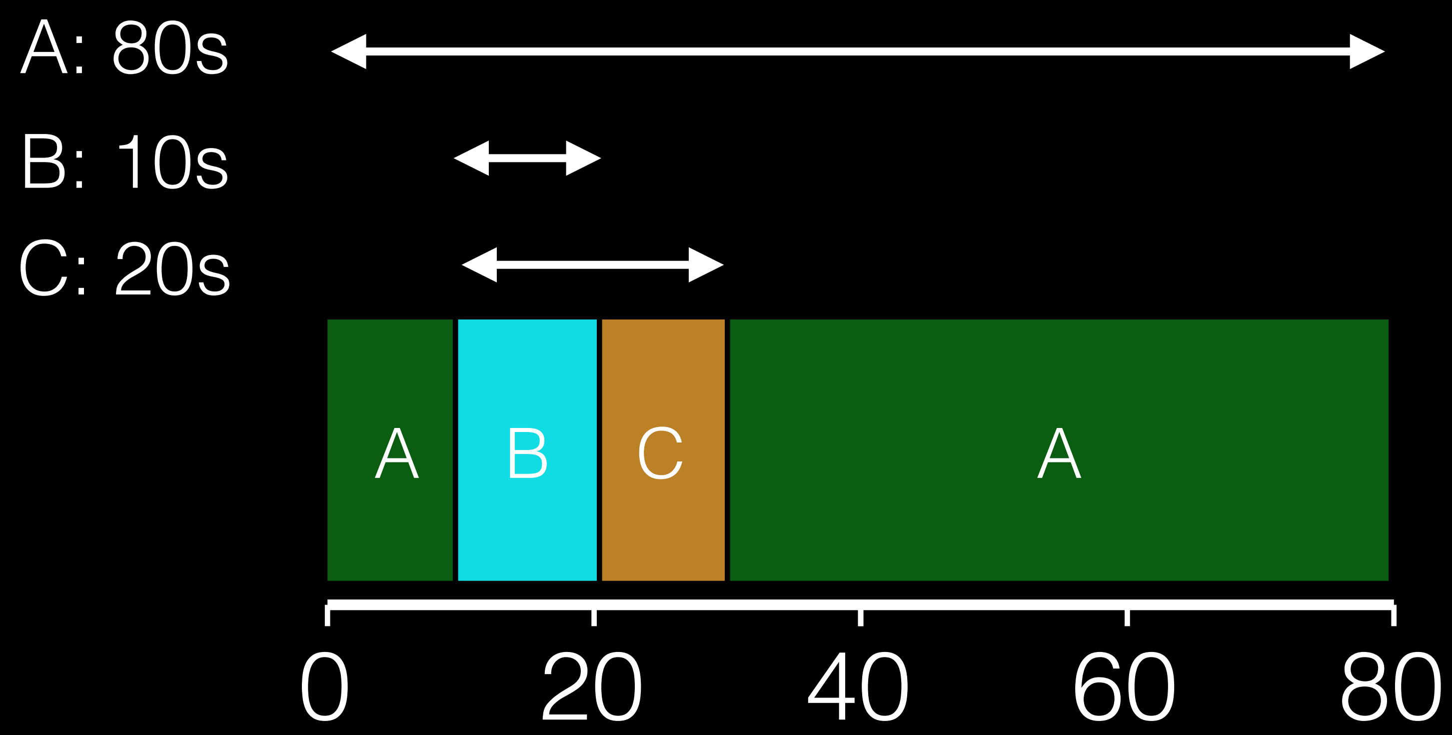
Average turnaround time: **(Q4)**

STCF



Average turnaround time: (Q4)

STCF



Average turnaround time: **36.6**

Scheduling Basics

Workloads:
- arrival_time
- run_time

Schedulers:
- FIFO
- SJF
- STCF
- RR

Metrics:
- turnaround_time
- response_time

Break!

Administrative Stuff

P1 due on 9/16 (6 days left!)

Office hours: today in CS 7373, 2-3pm

Reading: chapters 3-6 (last time) and 7-11 (today)

Discussion tomorrow: fork/exec, C review

Scheduling Basics

Workloads:
- arrival_time
- run_time

Schedulers:
- FIFO
- SJF
- STCF
- RR

Metrics:
- turnaround_time
- response_time

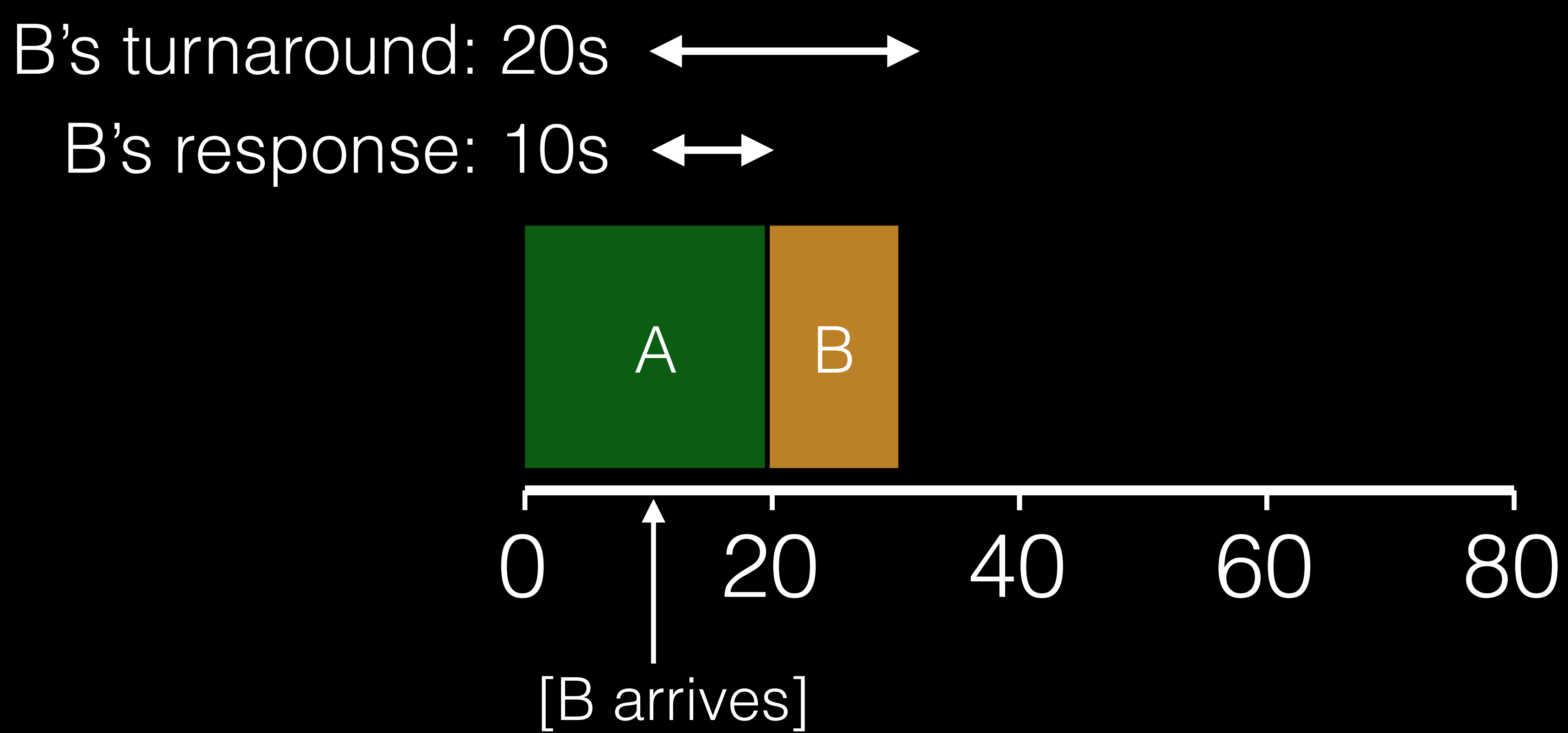
Response Time

Sometimes we care about when a job starts instead of when it finishes.

Why?

response_time = first_run_time - arrival_time

Response vs. Turnaround



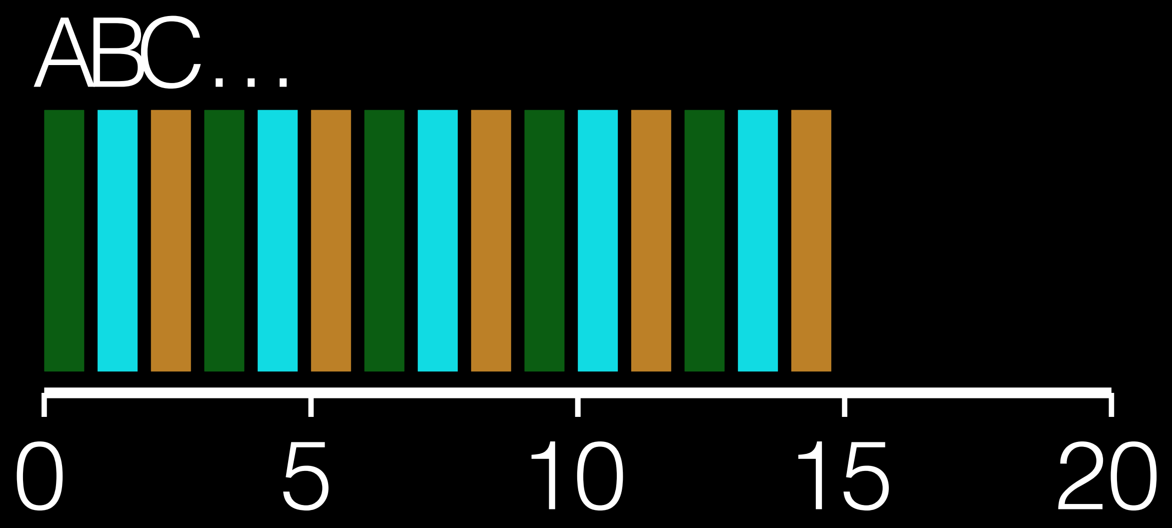
Round-Robin Scheduler

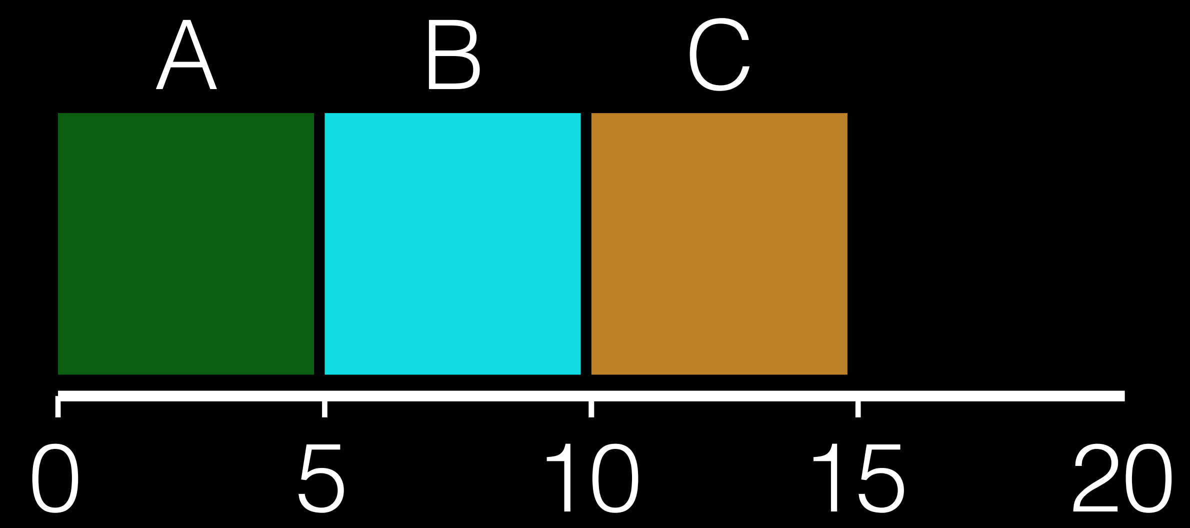
Prev schedulers: FIFO, SJF, and STCF have poor response time

New scheduler: RR (Round Robin)

Policy: alternate between ready processes every fixed-length slice

FIFO vs. RR (Q5) — which is each?

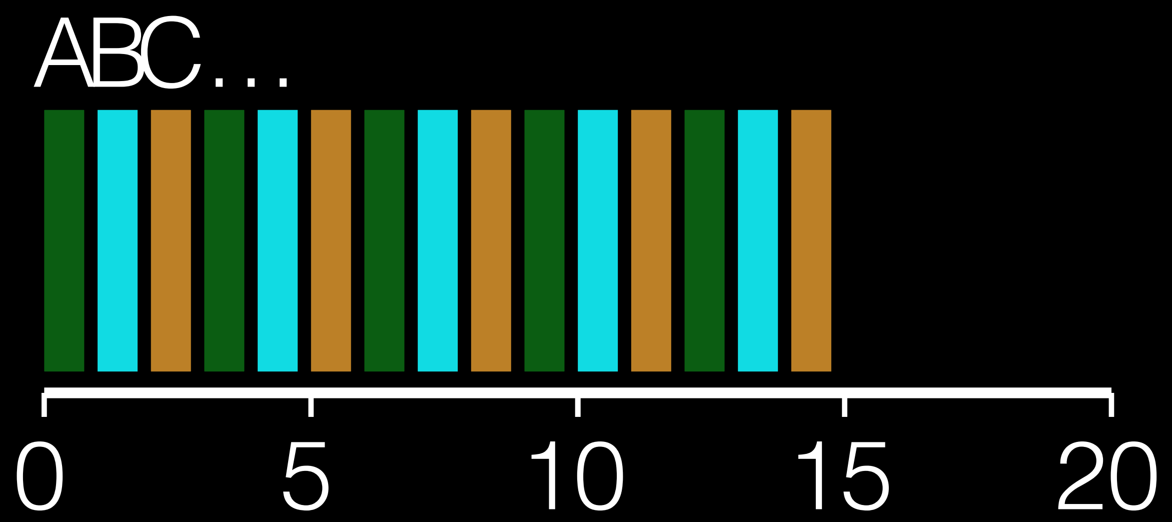


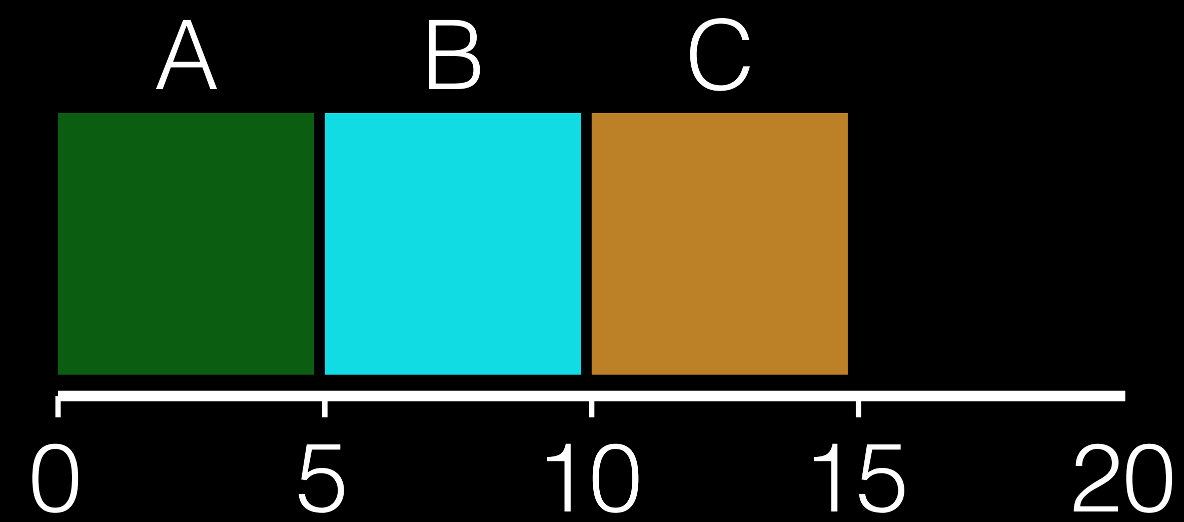


Avg Response Time?
Q5

Avg Response Time?
Q5

FIFO vs. RR (Q5) — which is each?

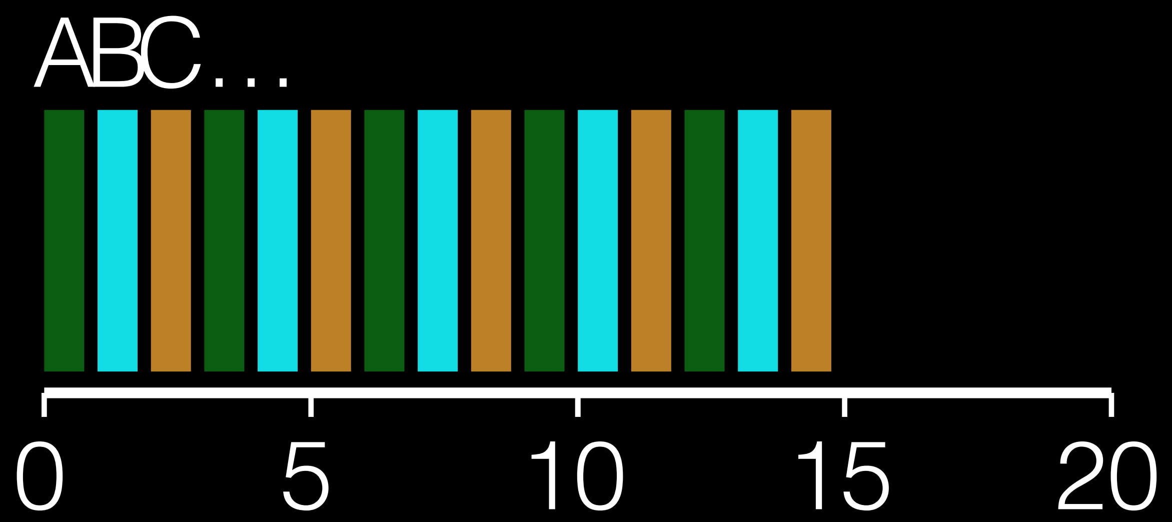


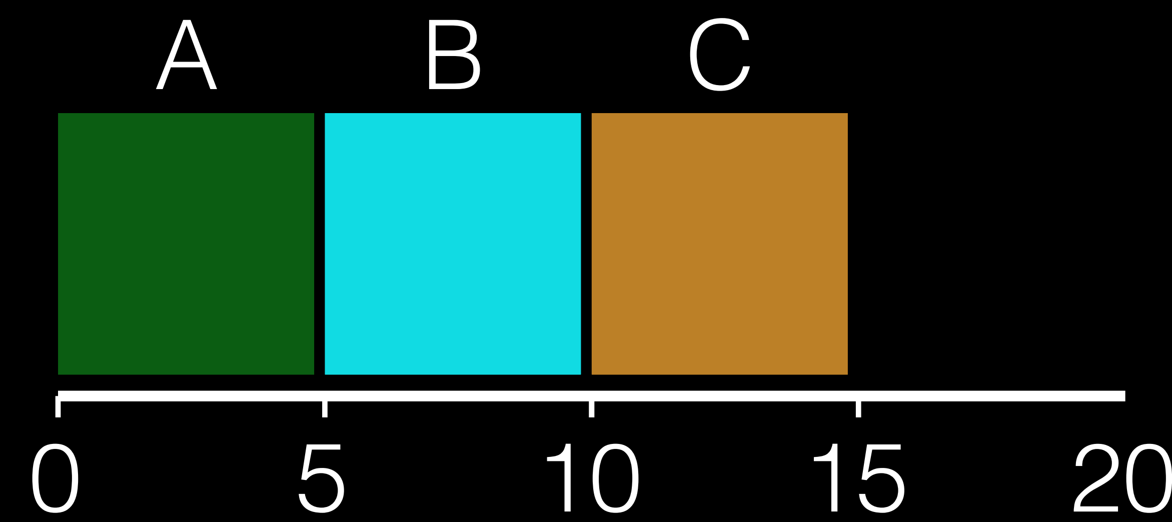


Avg Response Time?
Q5

Avg Response Time?
Q5

FIFO vs. RR (Q5) — which is each?





Avg Response Time?
(0+1+2)/3 = **1**

Avg Response Time?
(0+5+10)/3 = **5**

Scheduling Basics

Workloads:
- arrival_time
- run_time

Schedulers:
- FIFO
- SJF
- STCF
- RR

Metrics:
- turnaround_time
- 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. The run-time of each job is known

Workload Assumptions

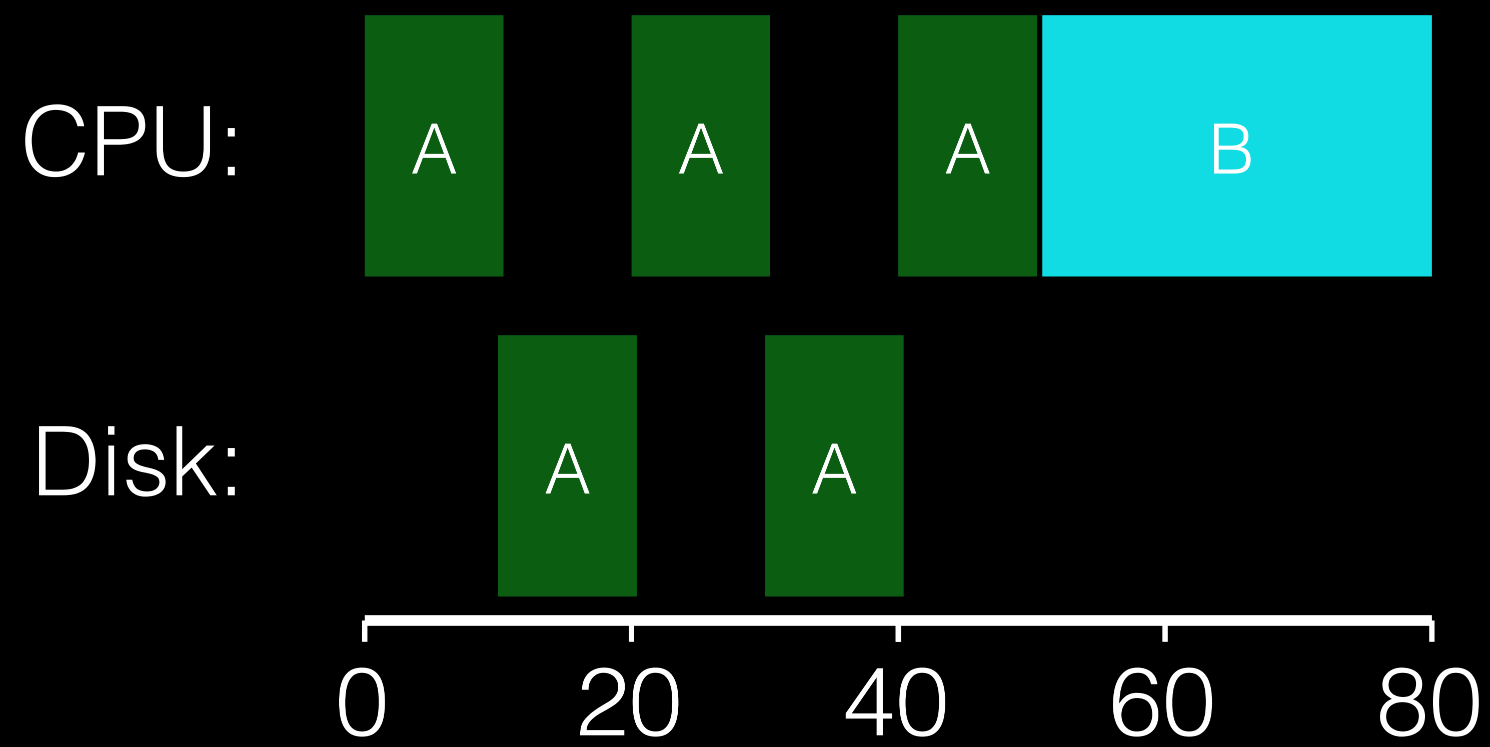
1. ~~Each job runs for the same amount of time~~

2. ~~All jobs arrive at the same time~~

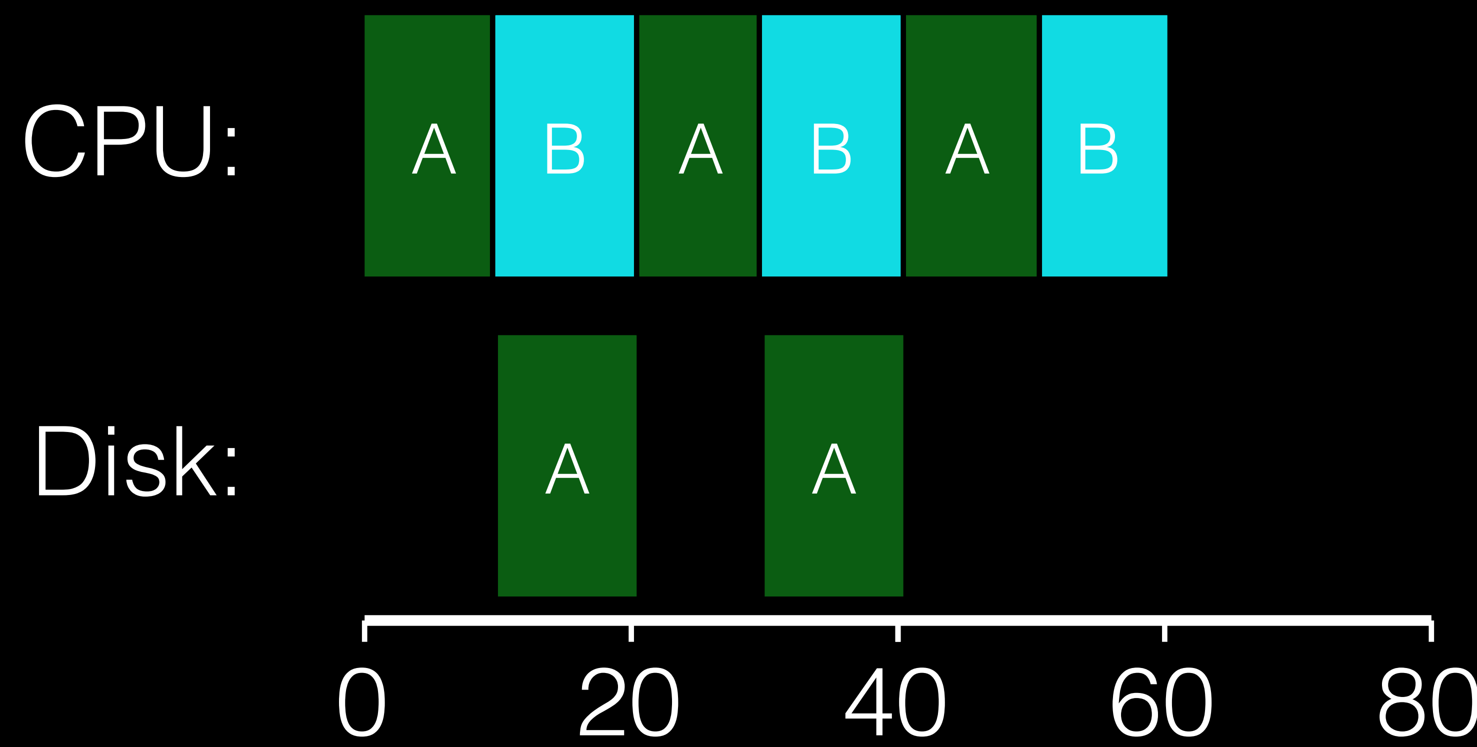
3. ~~All jobs only use the CPU (no I/O)~~

4. The run-time of each job is known

Not I/O Aware



I/O Aware (Overlap)



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. The 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. The run-time of each job is known~~
(need smarter, fancier scheduler)

MLFQ

MLFQ (Multi-Level Feedback Queue)

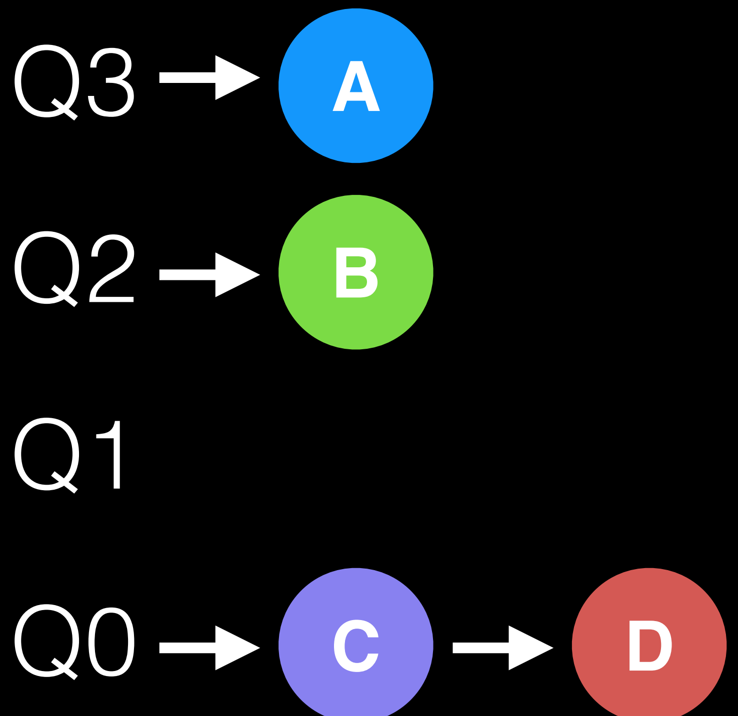
Goal: general-purpose scheduling

Must support two job types with distinct goals
- "interactive" programs care about response time
- "batch" programs care about turnaround time

Approach: multiple levels of round-robin

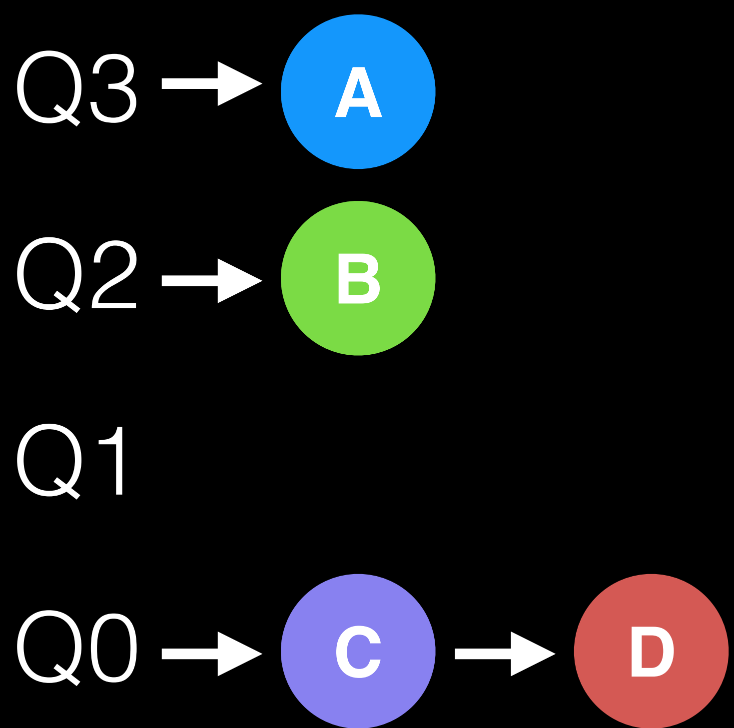
Priorities

Rule 1: If priority(A) > Priority(B), A runs
Rule 2: If priority(A) == Priority(B), A & B run in RR



Priorities

Rule 1: If priority(A) > Priority(B), A runs
Rule 2: If priority(A) == Priority(B), A & B run in RR

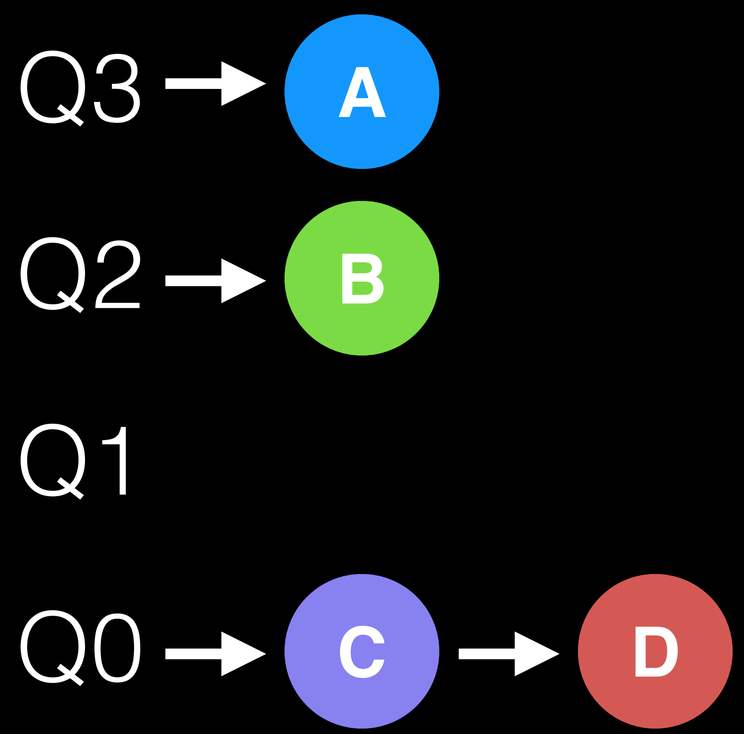


How to know process type to set priority?

Approach 1: nice
Approach 2: history

Priorities

Rule 1: If priority(A) > Priority(B), A runs
Rule 2: If priority(A) == Priority(B), A & B run in RR



How to know process type to set priority?

Approach 1: nice
Approach 2: **history**

History

Processes alternate between I/O and CPU work

Consider each CPU session it's own "job"

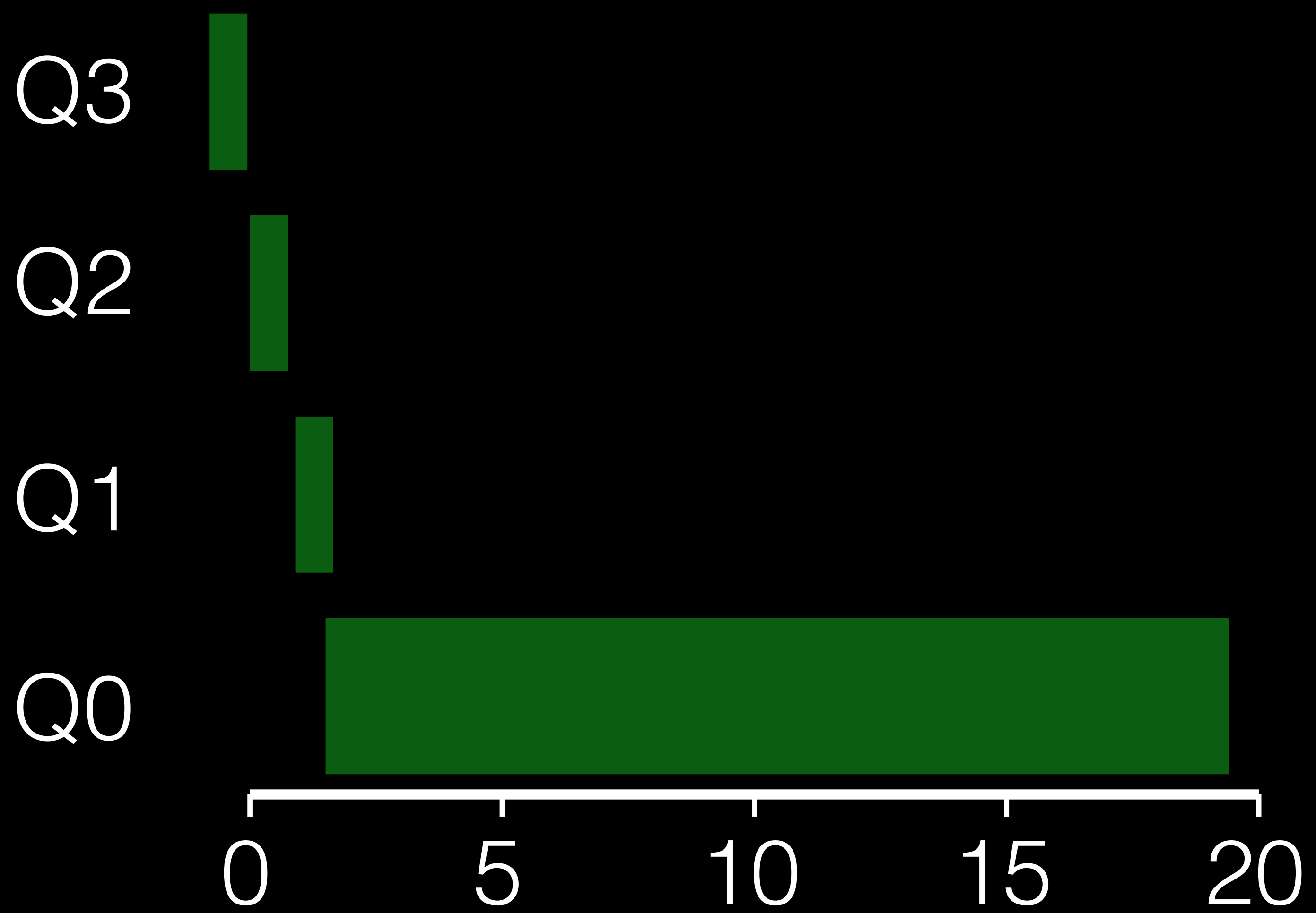
Guess what a job will be like based on past jobs from the same process

More MLFQ Rules

Rule 1: If priority(A) > Priority(B), A runs
Rule 2: If priority(A) == Priority(B), A & B run in RR

More rules:
Rule 3: Processes start at top priority
Rule 4: If job uses whole slice, demote process

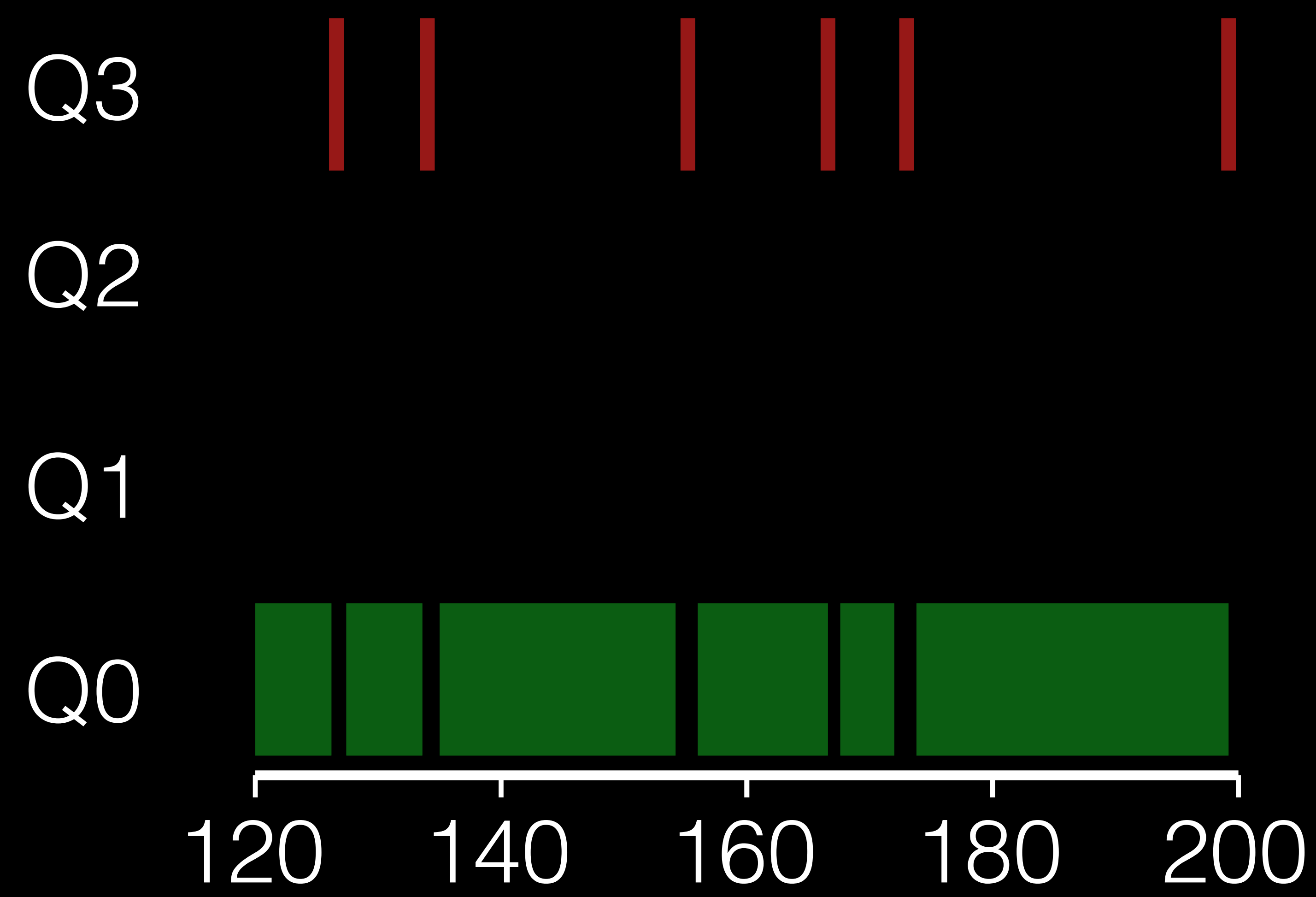
One Long Job (Example)



An Interactive Process Joins

Improvements

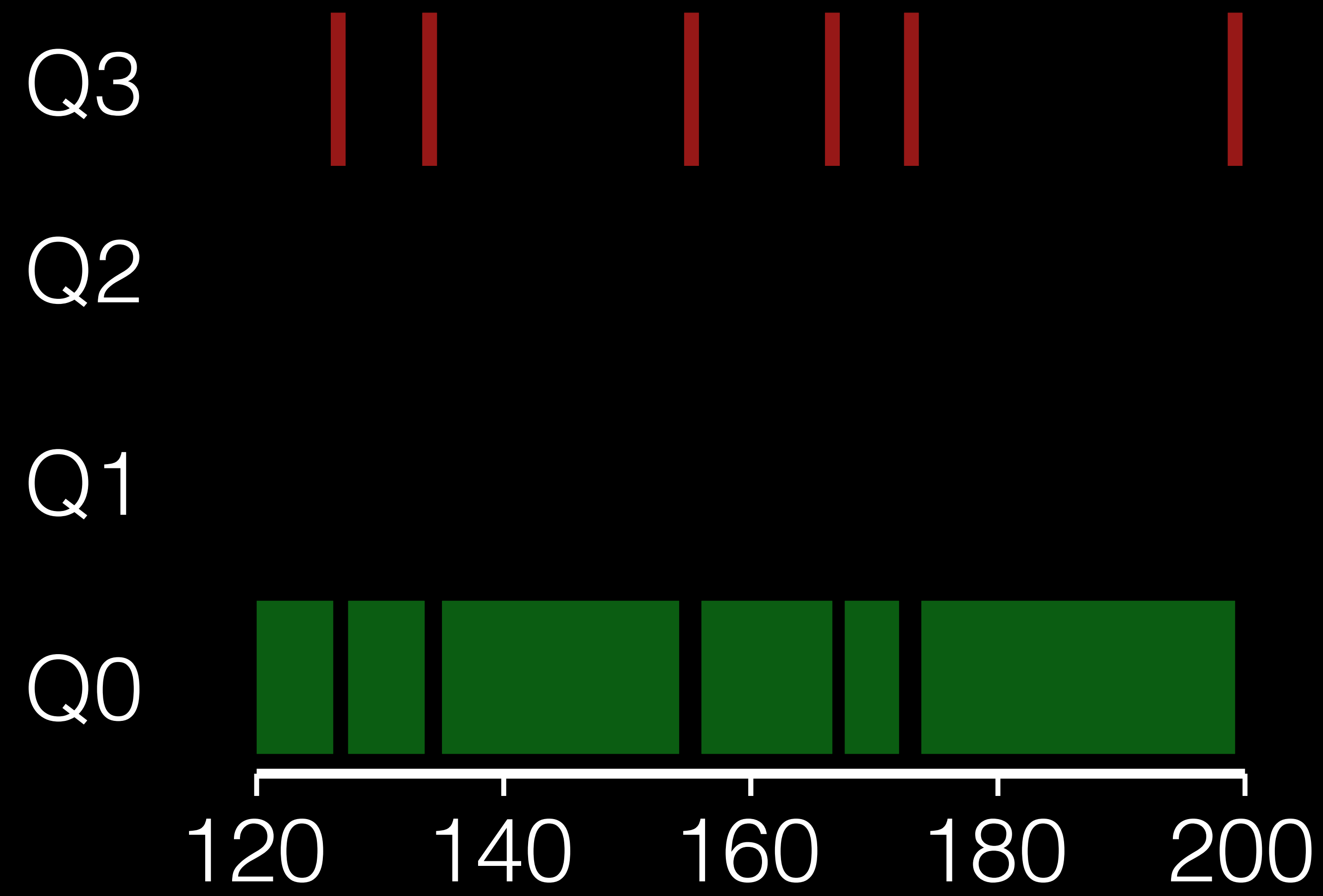
What are problems?



Improvements

What are problems?
- unforgiving
- gaming the system
- hard to tune

(read OSTEP)



Lottery

Lottery Scheduling

Goal: proportional share

Approach:
- give processes lottery tickets
- whoever wins runs
- higher priority => more tickets

Random Algorithms

Discus:

- disadvantages?
- advantages?

Lottery Code

```
int counter = 0;
int winner = getrandom(0, totaltickets);
node_t *current = head;
while(current) {
    counter += current->tickets;
    if (counter > winner)
        break;
    current = current->next;
}
// current is the winner
```

```
int counter = 0;
int winner = getrandom(0, totaltickets);
node_t *current = head;
while(current) {
    counter += current->tickets;
    if (counter > winner)
        break;
    current = current->next;
}
// current gets to run
```

Who runs if winner is:

50	(Q6)
350	(Q7)
0	(Q8)

head → Job A (1) → Job B (1) → Job C (100) → Job D (200) → Job E (100) → null

Other Lottery Ideas

Ticket Transfers

Ticket Currencies

Ticket Inflation

(read more in OSTEP)

Summary

Understand your goals (metrics) and workload, then design your scheduler around that.

General purpose schedulers need to support processes with different types of goals.

Random algorithms are often simple to implement, and avoid corner cases.