Chapter 7
Scheduling

Dong Dai
CIS/UD
dai@udel.edu

Scheduling

- CPU Virtualization
  - low-level **mechanisms** - context switching
  - high-level **policies** – **scheduling policies**
- Appears in many fields – e.g., factory assembly lines for efficiency
- How to develop scheduling policies? How to develop a basic **framework** for studying scheduling policies
  1. assumptions of **workload**
  2. metrics of **performance**
  3. approaches used

1. Workload Assumptions

Workload – {processes} running in the system

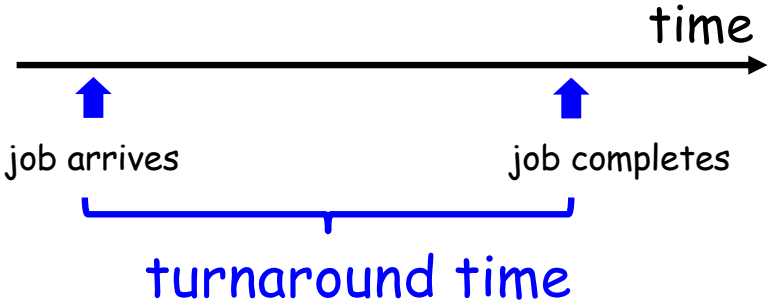
- critical to building policies - the more you know, the more fine-tuned your policy can be
- **unrealistic**, initially, but will be relaxed as we go

1. each job runs for the same amount of time
2. all jobs arrive at the same time
3. once started, each job runs to completion
4. all jobs only use CPU (i.e., they perform **no** I/O)
5. the run-time of each job is known

Two kinds: **batch** (CPU-bound) vs. **interactive** (I/O-bound)

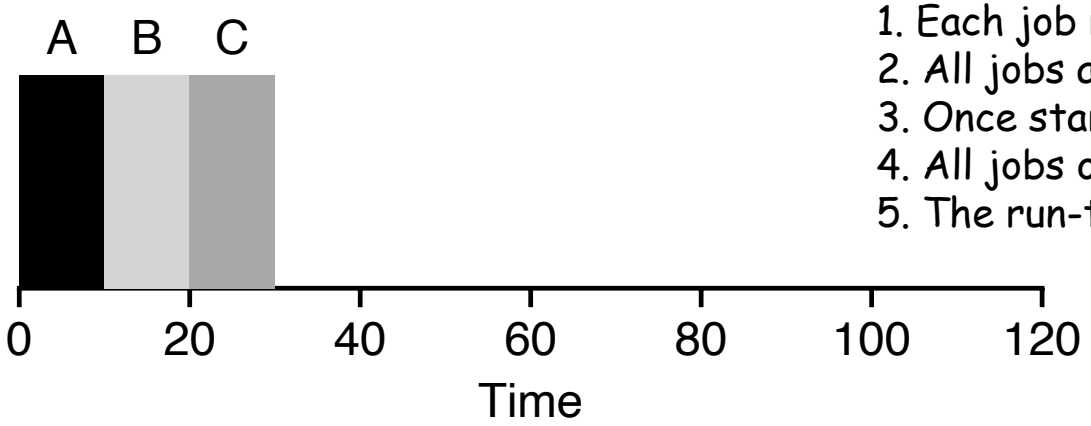
2. Scheduling Metrics

- To compare different scheduling policies, we need **metrics** to measure behavior and hence performance
- Turnaround time - the time at which the job completes **minus** the time at which the job arrived in the system
  - a **performance** metric
- Fairness metric
  - Jain's Fairness Index
- Performance vs. fairness **tradeoff**
  - a scheduler may optimize performance but at the cost of preventing a few jobs from running, thus decreasing fairness



First In First Out (FIFO)

- First come, first served (FCFS)



Workload assumptions:
1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. Once started, each job runs to completion
4. All jobs only use CPU (i.e., they perform no I/O)
5. The run-time of each job is known

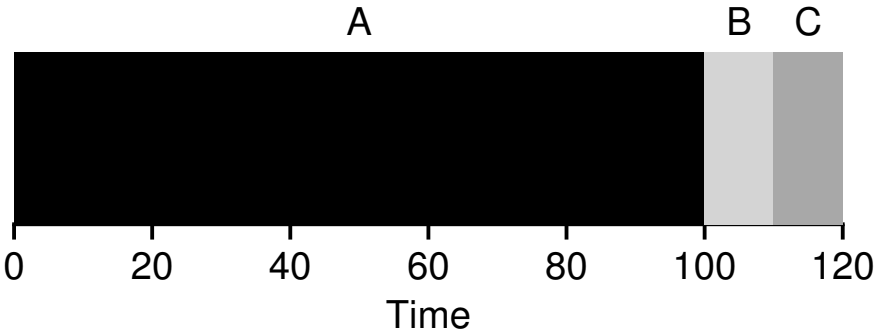
- Each job runs 10 sec.
- Average turnaround time = ?

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

turnaround time = completion time – arrival time

First In First Out (FIFO)

- What kind of **workload** could you construct to make FIFO perform poorly?
- **Relax assumption #1**: jobs could have different (still known) run times



- Average turnaround time = ?
  = (100+110+120)/3 = 110

Workload assumptions
~~1. Each job runs for the same amount of time~~
2. All jobs arrive at the same time
3. Once started, each job runs to completion
4. All jobs only use CPU (i.e., they perform no I/O)
5. The run-time of each job is known

- Convoy effect (supermarket checkout line)
- How can you improve it? (new scheduling policy?)

Shortest Job First (SJF)

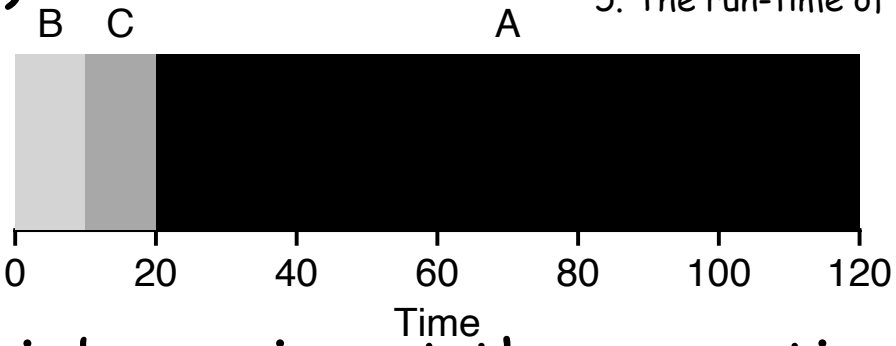
Workload assumptions

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

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

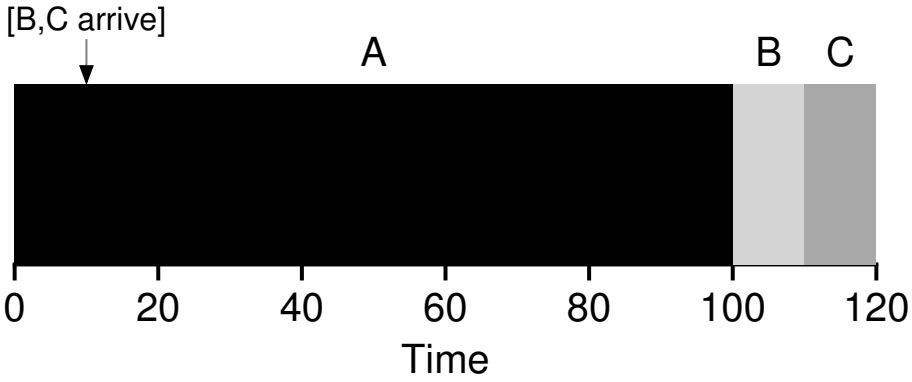
3. Once started, each job runs to completion
4. All jobs only use CPU (i.e., they perform no I/O)
5. The run-time of each job is known

- Average turnaround time = ?
  = (10+20+120)/3 = 50



- Assuming all jobs arrive at the **same** time, SJF is **optimal**
- **Relax assumption #2: arrival time**

  average turnaround time
  = (100+(110-10)+(120-10))/3
  = 103.33



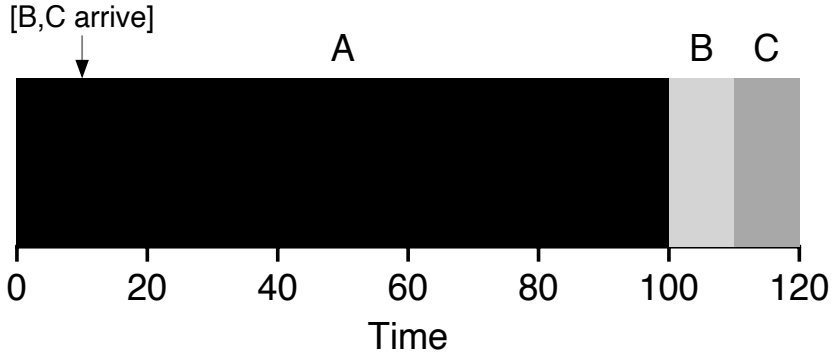
- How can we do better?

Shortest Time-to-Completion First

- SJF with A arrives at 0, and B, C at 10

Workload assumptions

~~1. Each job runs for the same amount of time~~
~~2. All jobs arrive at the same time~~
~~3. Once started, each job runs to completion~~
4. All jobs only use CPU (i.e., they perform no I/O)
5. The run-time of each job is known

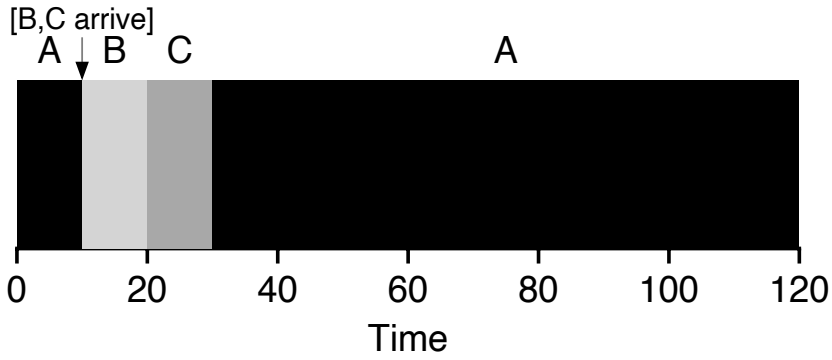


- How would you do better?

  Relax assumption #3

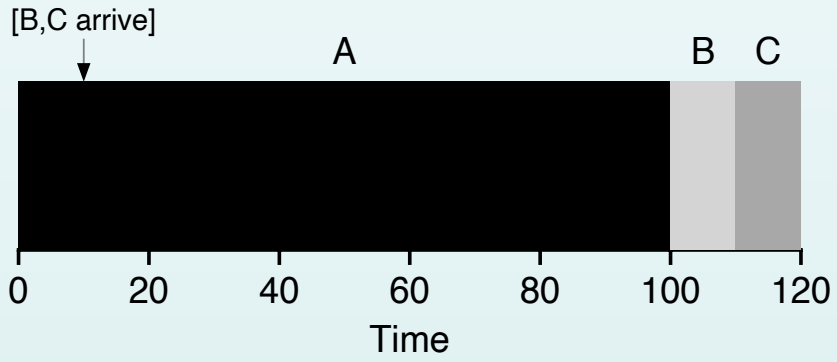
  ➔preemption

  ➔STCF



or **preemptive SJF** (also **optimal** with assumption that jobs arrive at **any** time)

- What if we can not **Relax assumption #3?**



Delay Scheduling*

SchedInspector: A Batch Job Scheduling Inspector Using Reinforcement Learning

Di Zhang
Computer Science Department, University of North Carolina at Charlotte
Charlotte, NC, USA
dzhang16@uncc.edu

Dong Dai
Computer Science Department, University of North Carolina at Charlotte
Charlotte, NC, USA
ddai@uncc.edu

Bing Xie
Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory
Oak Ridge, TN, USA
xieb@ornl.gov

ACM HPDC 2022
The 31st International Symposium on High-Performance Parallel and Distributed Computing
Minneapolis, Minnesota, United States, June 27 - July 1, 2022

Delay Scheduling*

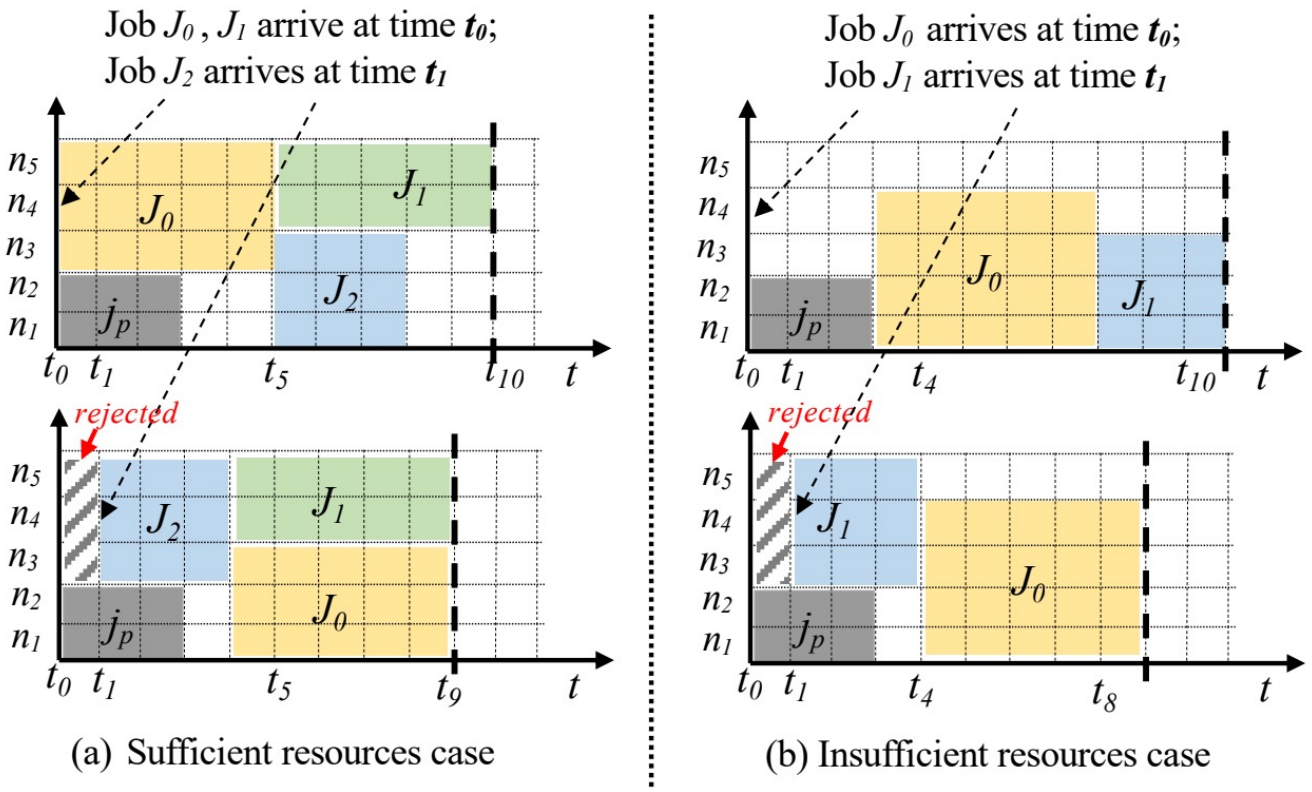


Figure 1: Scheduling jobs with/without SchedInspector. ***x-axis shows the timeline in minutes; y-axis shows the compute nodes ($n_{1\rightarrow5}$); each block represents a job that takes amount of nodes and time; J_p is the preliminary job running before the scheduling starts.***

Delay Scheduling*

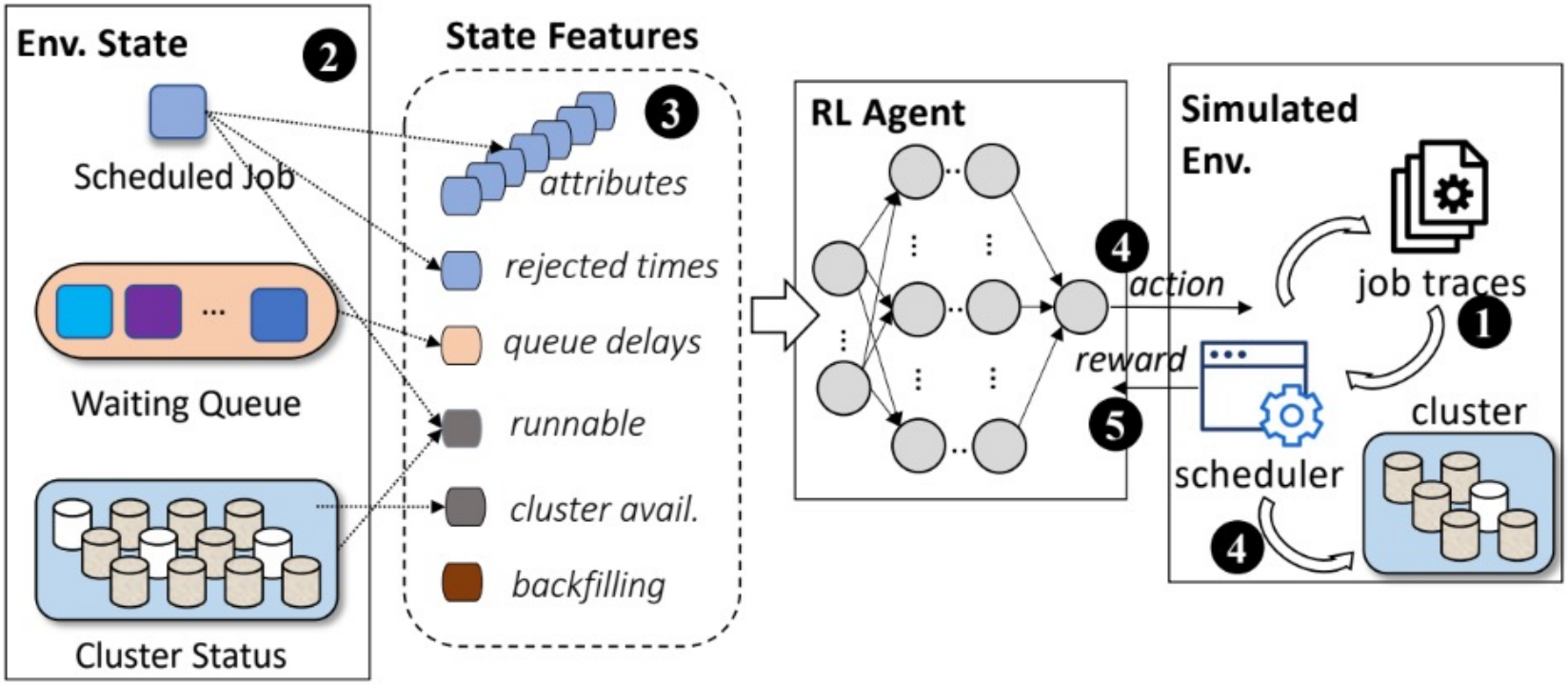
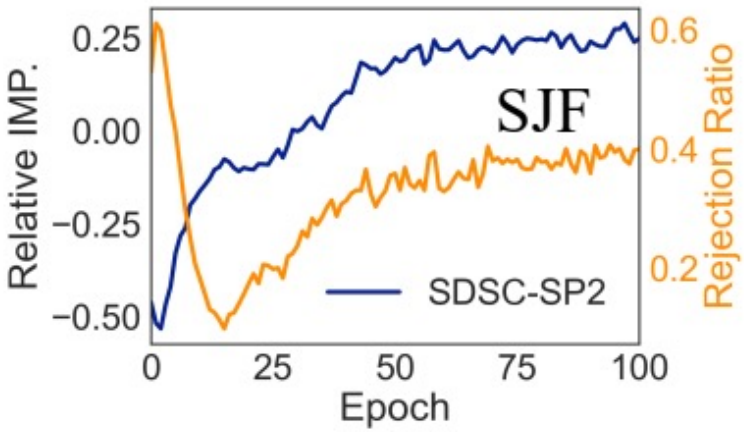
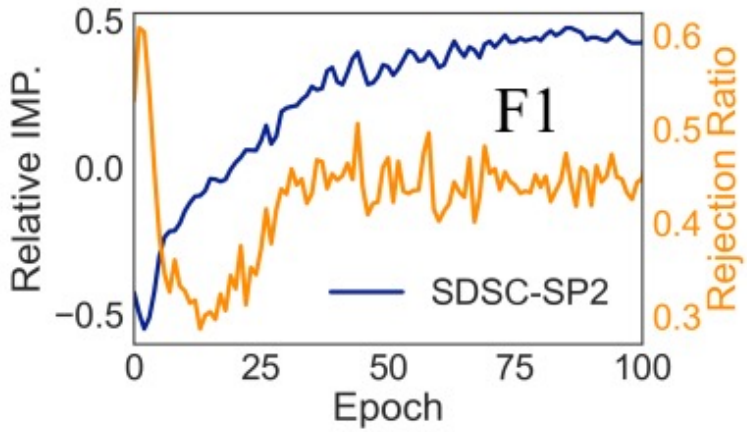


Figure 3: Architecture of SchedInspector.

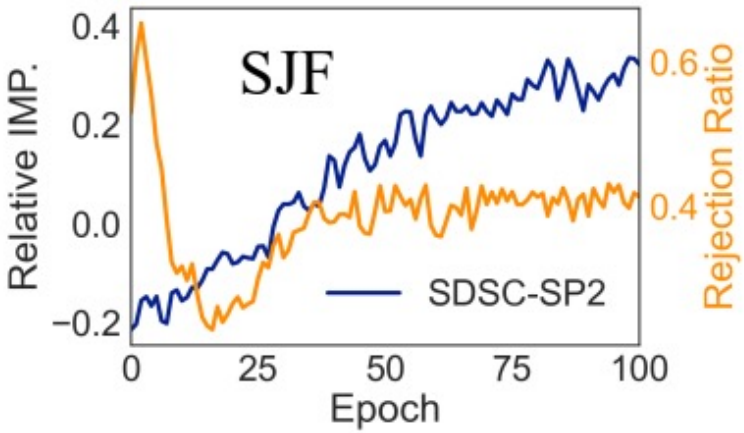
Delay Scheduling*

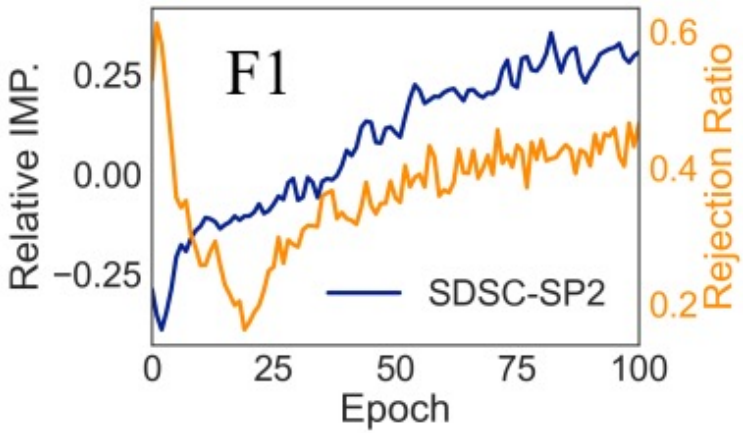
(a) wait





(b) mbsld





New Metric

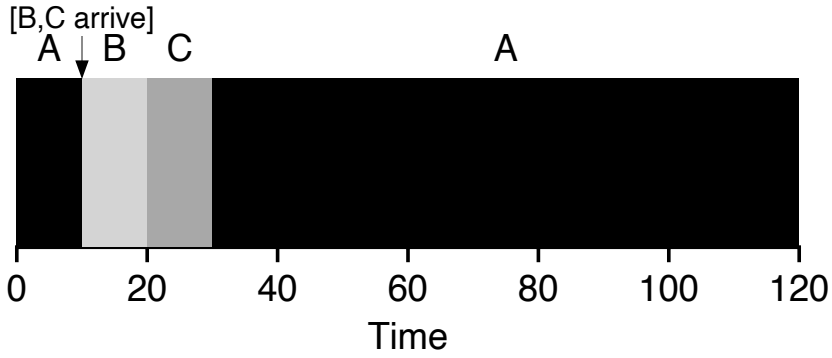
Workload assumptions

~~1. Each job runs for the same amount of time~~
~~2. All jobs arrive at the same time~~
~~3. Once started, each job runs to completion~~
4. All jobs only use CPU (i.e., they perform no I/O)
5. The run-time of each job is known

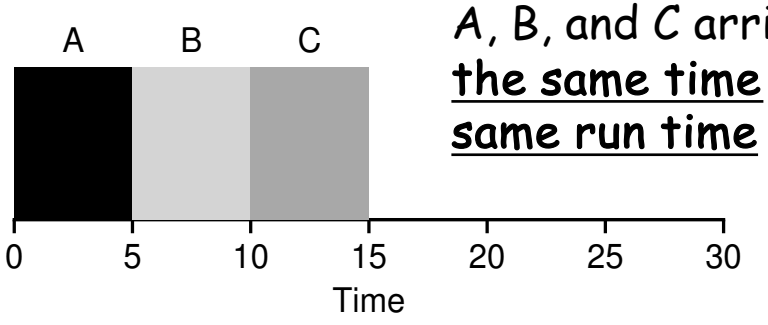
- With known job lengths, **CPU-only jobs**, and metric of turnaround time, STCF is great for **batch** systems (processes that run **without** user interaction)
- **Time sharing** system with **interactive** applications - need new metric
- **Response time** = time from when the job arrives in a system to **the first time it is scheduled**

Response Time

- Response time = (0 + 0 + 10)/3 = 3.33



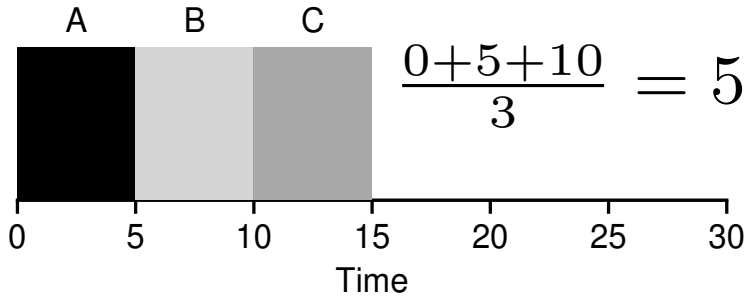
- Is STCF (or SJF) good for **interactive** applications?
  - not at all (with just preemption)!
- What can we do?

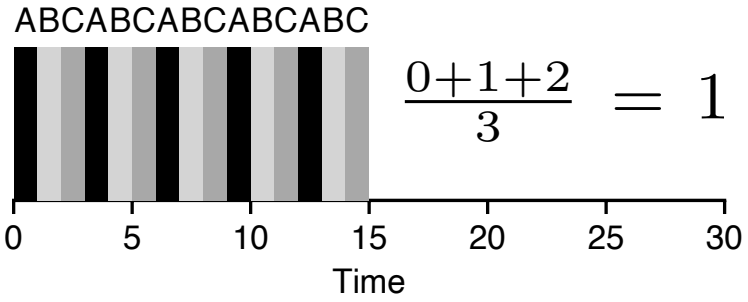


A, B, and C arrive **at the same time** with **same run time** of 5

Round Robin (RR)

- Instead of running jobs to completion, RR runs a job for a (small) **time slice** (scheduling quantum) and switches to the next job in the ready queue; repeatedly does so until jobs are finished
- **Time slicing** - length of time slice = multiple of timer-interrupt period
  - SJF avg. response time = 5
  - RR avg. response time = 1
- **Length of time slice is critical**
  - under response time, the shorter the better
  - too short, overhead of context switching dominates





Round Robin

- **Amortization** (in general)
  - used in systems when there is a **fixed cost** to some operation (e.g., context switching)
  - by incurring that cost **less often**, the total cost to the system is reduced
- For example, if time slice == 10 ms, and the context-switch cost is 1 ms, roughly 10% of time is spent context switching and is thus wasted
- To **amortize** this cost, we can increase the time slice, to, e.g., 100 ms. In this case, less than 1% of time is spent context switching, and thus the **cost** of time-slicing has been amortized

Round Robin

- Tradeoff?
- Length of time slice on response time ???
  - the shorter the time slice, the shorter (better) the response
  - too short, cost of context switching (**overhead**) dominates
  - length of **time slice** is a **tradeoff** - long enough to amortize the cost of context switching without making it so long that the system is no longer responsive
- In addition, any policy (such as RR) that is **fair** (evenly divides CPU among active processes on a small time scale) will perform **poorly** on metrics such as **turnaround time**

So Far…

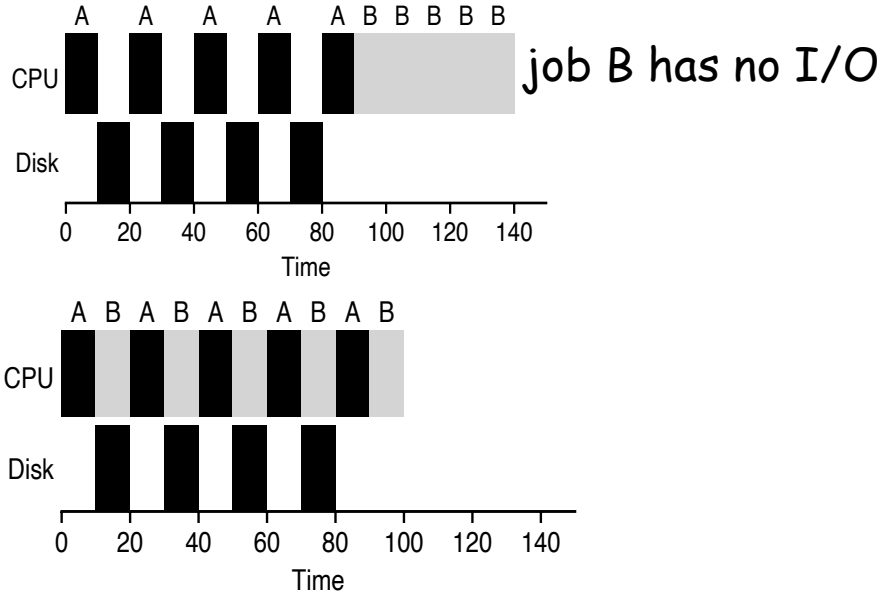
- Two types of schedulers
  - SJF and STCF: optimize **turnaround** time, but is bad for response time
  - RR: optimize **response** time, but is bad for turnaround time

Workload assumptions

~~1. Each job runs for the same amount of time~~
~~2. All jobs arrive at the same time~~
~~3. Once started, each job runs to completion~~
~~4. All jobs only use CPU (i.e., they perform no I/O)~~
5. The run-time of each job is known

Incorporating I/O

- Process is **blocked** waiting for I/O completion
- When I/O completes, an interrupt is raised, and OS runs and moves the process that issued the I/O from **blocked** state back to **ready** (or even running) state
- Poor use of CPU
- Higher CPU utilization?
  - overlap CPU & I/O



- **Interactive jobs** get run frequently; while they are performing I/O, other CPU-intensive jobs run

No More Oracle

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. Once started, each job runs to completion
4. All jobs only use CPU (they perform no I/O)
5. The run-time of each job is known

- **how can we build a scheduler that behaves like SJF/STCF without *a priori* knowledge of runtime?**

Workload assumptions

~~1. Each job runs for the same amount of time~~
~~2. All jobs arrive at the same time~~
~~3. Once started, each job runs to completion~~
~~4. All jobs only use CPU (i.e., they perform no I/O)~~
~~5. The run-time of each job is known~~

next chapter on multi-level feedback queue

Can you answer these?

- True/False
  - A timer tick is identical to a time slice
  - The convoy effect occurs when longer jobs must wait for shorter jobs
  - A RR scheduler may preempt a previously running job
- Essay Questions
  - Assume the OS schedules a workload containing three jobs with the following characteristics:

Job	Arrival Time	CPU Time
A	0	10
B	5	8
C	12	2

  - Which scheduler minimizes the completion time of the entire workload? FIFO, RR, SJF, STCF?
  - Given a **STCF** scheduler, what is the response time for Job B?
  - Given a **STCF** scheduler, what is the response time for Job C?

Can you answer these?

- True/False
  - A timer tick is identical to a time slice **[False]**
  - The convoy effect occurs when longer jobs must wait for shorter jobs **[False]**
  - A RR scheduler may preempt a previously running job **[True]**
- Essay Questions
  - Assume the OS schedules a workload containing three jobs with the following characteristics:

Job	Arrival Time	CPU Time
A	0	10
B	5	8
C	12	2

  - Which scheduler minimizes the completion time of the entire workload? FIFO, RR, SJF, STCF?
    - None of the above. Ass schedulers give the same results. The completion time for the entire workload (i.e., the last job) will be the same independent of the completion time of the individual jobs
  - Given a **STCF** scheduler, what is the response time for Job B?
    - B is scheduled at time 10 (at time 5 when B arrives, it has a burst of 8 > A's remaining burst of 5); 10-5 = 5
  - Given a **STCF** scheduler, what is the response time for Job C?
    - When C arrives at time 12, it has a CPU burst of 2 < B's remaining CPU burst of 8-2 = 6; therefor, C is scheduled immediately when it arrives.