Chapter 7: Scheduling Introduction

Adam Disney

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

Scheduling Policies

- We now understand the **mechanisms** of running processes...
- Now we need to go into the **policies** for using the **mechanisms**.

The Crux of Scheduling

THE CRUX: HOW TO DEVELOP SCHEDULING POLICY

How should we develop a basic framework for thinking about scheduling policies? What are the key assumptions? What metrics are important? What basic approaches have been used in the earliest of computer systems?

Workload Assumptions

- Workload – The processes running on the system
- Understanding the workload is critical to building policies.
- Let's make some simplifying (and unrealistic) assumptions
  - Each job runs for the same amount of time
  - All jobs arrive at the same time
  - Once started, each job runs to completion
  - All jobs only use the CPU
  - The run-time of each job is known

Scheduling Metrics

- We need a way to compare scheduling policies
- For now, we'll keep it simple with a single metric **Turnaround time**
- This is a performance metric and our primary focus for now (there are other metrics we can use)
- For now, we assume all jobs arrive at 0 thus Turnaround Time = Completion Time

$$T_{turnaround} = T_{completion} - T_{arrival} \quad (7.1)$$

First In, First Out (FIFO) or First Come, First Served (FCFS)

- Most basic scheduling algorithm.
- Assume jobs A, B, C each take 10 seconds and arrive nearly simultaneously in the order given.

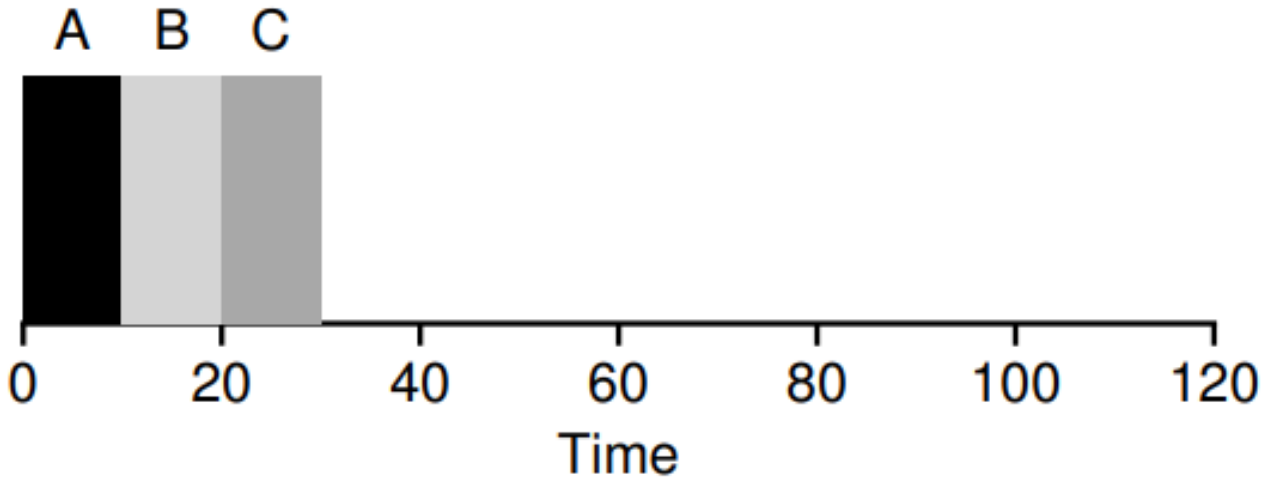

Figure 7.1: **FIFO Simple Example**

?

First In, First Out (FIFO)

- Let us drop the assumption that jobs have the same running time
- What kind of workload would make this perform poorly?

First In, First Out (FIFO)

- Let us drop the assumption that jobs have the same running time
- What kind of workload would make this perform poorly?

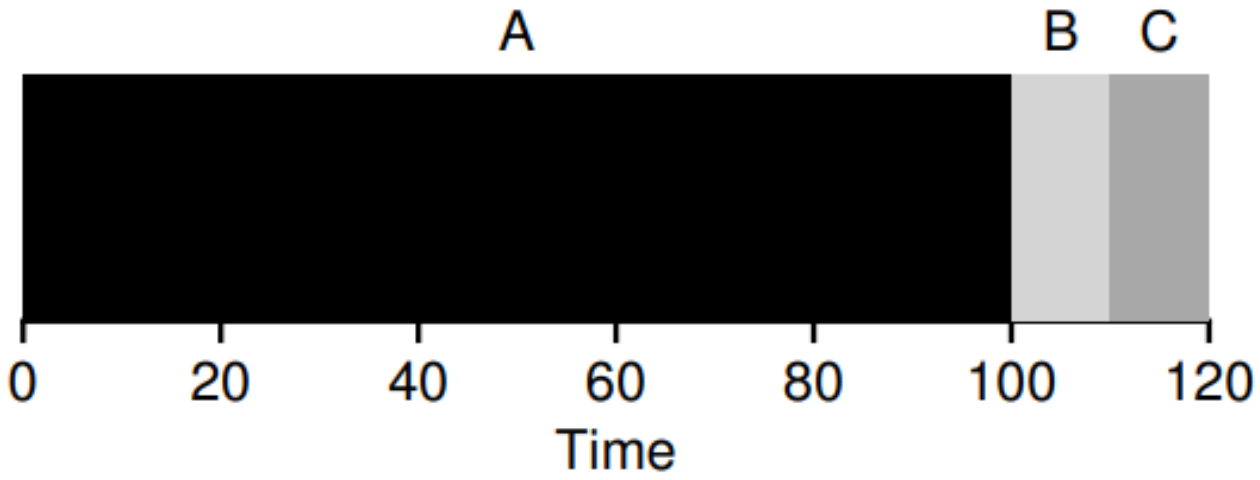

Figure 7.2: **Why FIFO Is Not That Great**

?

First In, First Out (FIFO)

- This problem is generally referred to as the **convoy effect,** where short jobs get queued behind a long job.

Shortest Job First (SJF)

- Well, if we're concerned with turnaround time, how about we schedule the shortest job first!

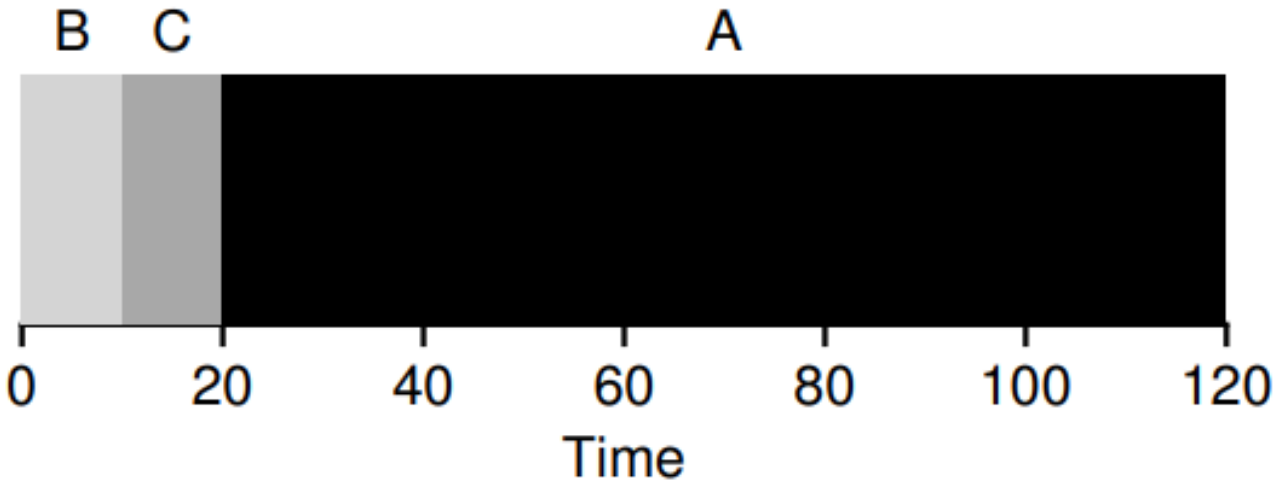

Figure 7.3: **SJF Simple Example**

?

Shortest Job First (SJF)

TIP: THE PRINCIPLE OF SJF

Shortest Job First represents a general scheduling principle that can be applied to any system where the perceived turnaround time per customer (or, in our case, a job) matters. Think of any line you have waited in: if the establishment in question cares about customer satisfaction, it is likely they have taken SJF into account. For example, grocery stores commonly have a "ten-items-or-less" line to ensure that shoppers with only a few things to purchase don't get stuck behind the family preparing for some upcoming nuclear winter.

Shortest Job First (SJF)

- That's much better and in fact can be proven to be an optimal solution given our assumptions!
- What if we relax the assumption that jobs arrive at the same time?
- What can go poorly here?

Shortest Job First (SJF)

- That's much better and in fact can be proven to be an optimal solution given our assumptions!
- What if we relax the assumption that jobs arrive at the same time?
- What can go poorly here?

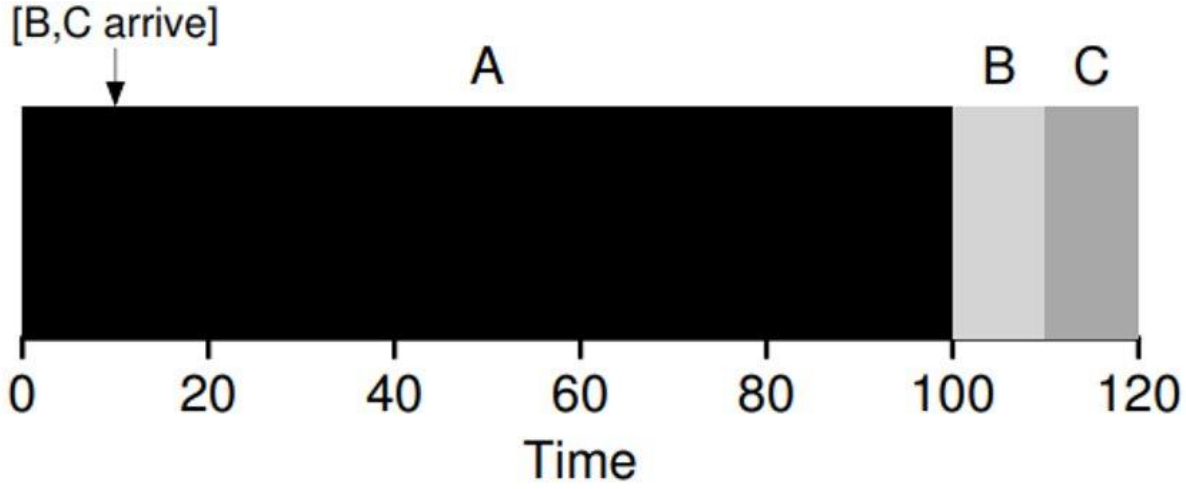

Figure 7.4: **SJF With Late Arrivals From B and C**

Shortest Time-to-Completion First (STCF) or Preemptive Shortest Job First (PSJF)

- To address this issue, let's relax the assumption that jobs must be run to completion.
- This essentially adds preemption to SJF. (Thus, PSJF)

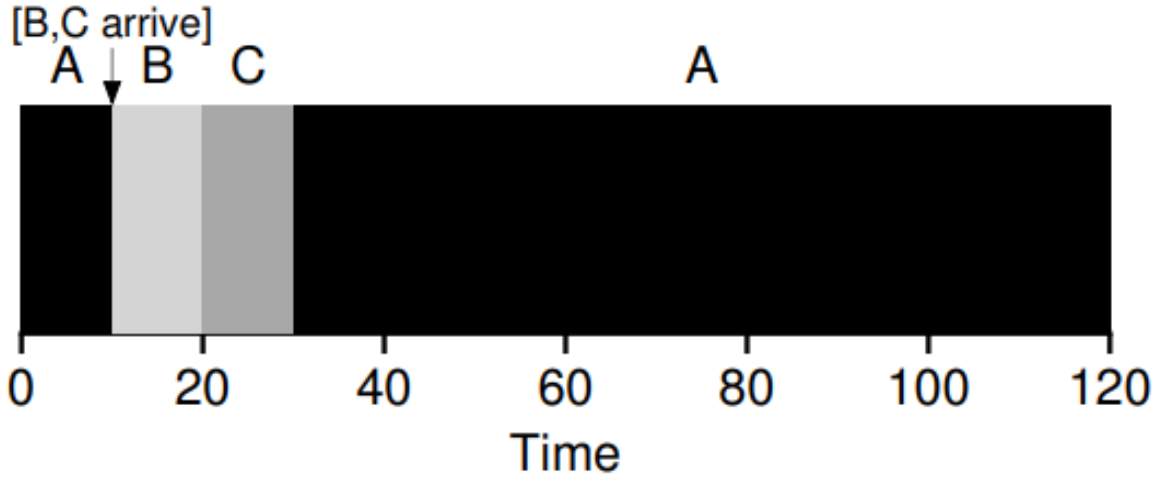

Figure 7.5: **STCF Simple Example**

?

Shortest Time-to-Completion First (STCF)

- Turns out this also provably optimal given our current assumptions!
  - We know job lengths ahead of time
  - Jobs only use the CPU
  - Only metric is turnaround time
- Early batch computing systems used this type of scheduling.
- Introduction of time-shared machines with users waiting at terminals changed all that.

New Metric: Response Time

- Now users are typing at a terminal. When they input something, they want a quick response. (e.g. "ls", "cd")
- Thus, this metric measures the time it takes for the program to be scheduled for the first time.

$$T_{response} = T_{firstrun} - T_{arrival} \tag{7.2}$$

New Metric: Response Time

- Assume that jobs A, B, and C arrive at time 0.

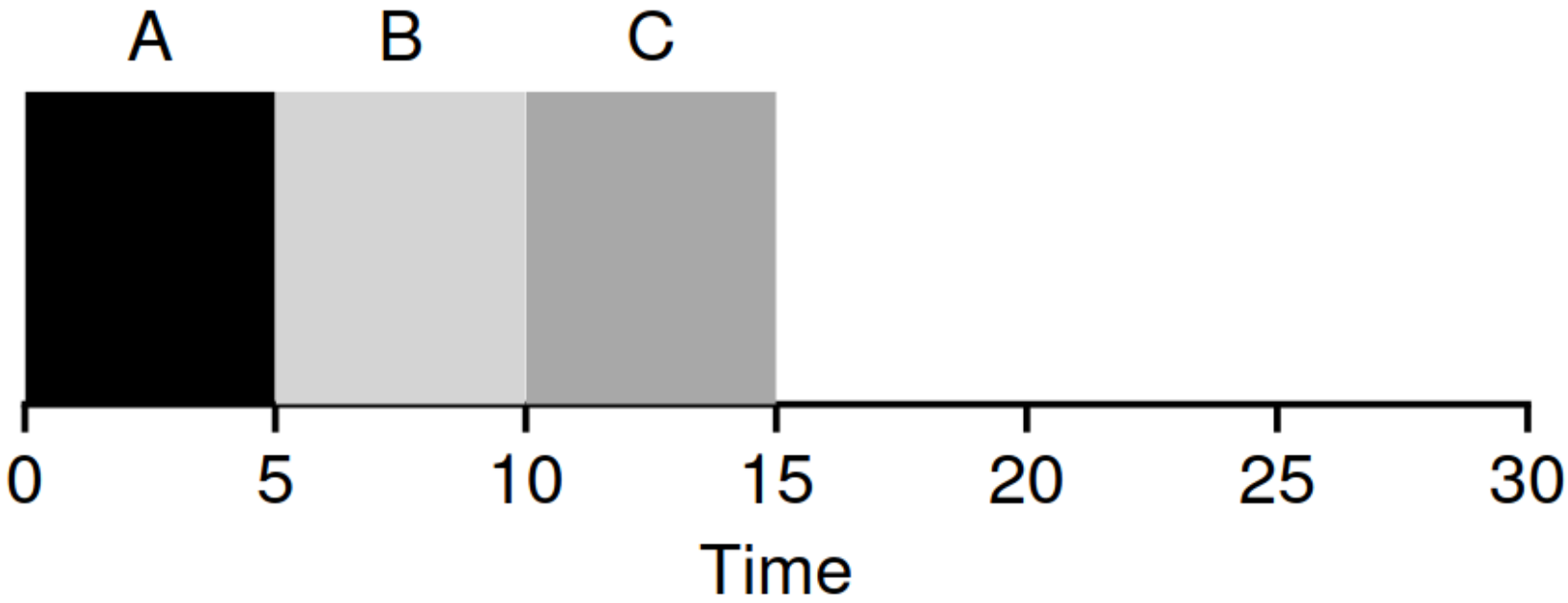

Figure 7.6: **SJF Again (Bad for Response Time)**

?

Round Robin (RR)

- Instead of running to completion, we'll run jobs for a **time slice**.

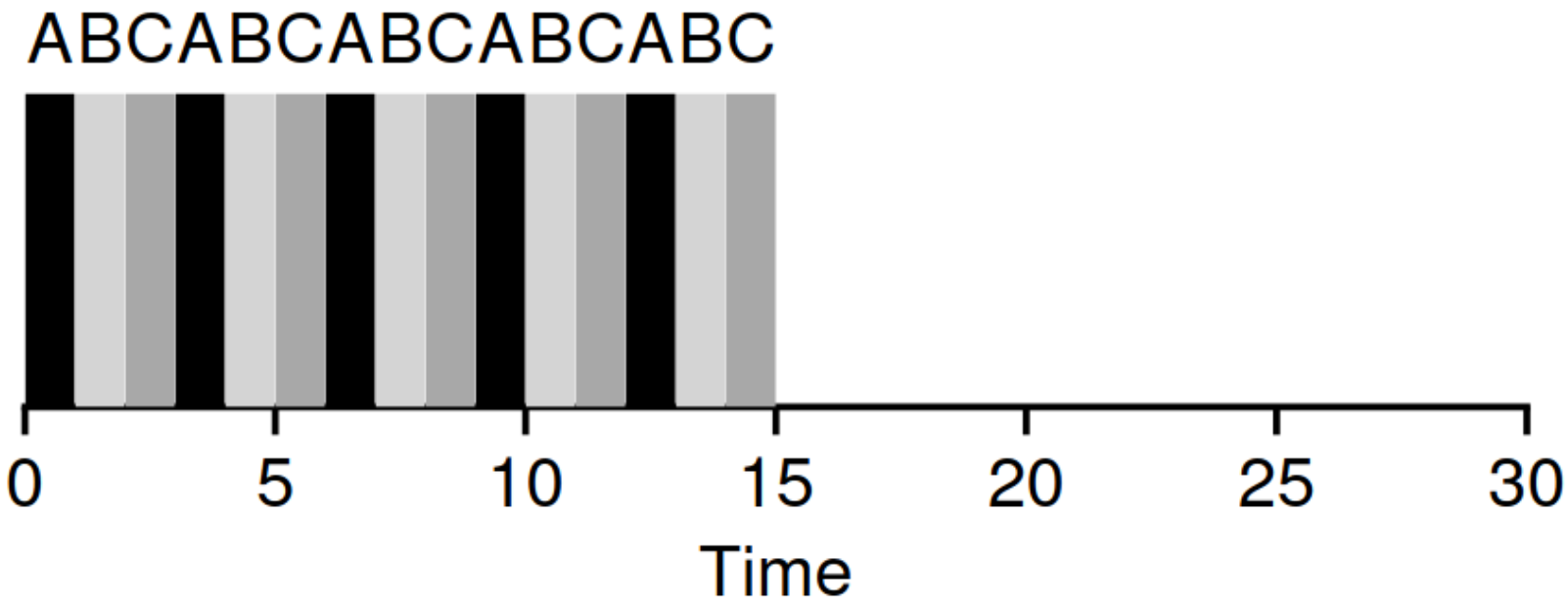

Figure 7.7: **Round Robin (Good For Response Time)**

?

Round Robin (RR)

- Picking the **time slice** length is very important.
- The shorter we make it, the best response time we have...
- But there's a cost to context switching.
- And what about turnaround time?

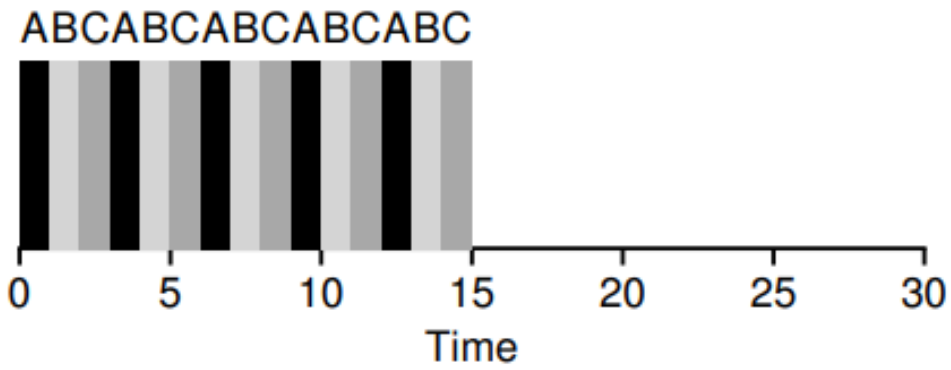

Figure 7.7: **Round Robin (Good For Response Time)**

Round Robin (RR)

TIP: AMORTIZATION CAN REDUCE COSTS

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

Incorporating I/O

- Let's relax the assumption that programs have no I/O
- **Overlap** of CPU and Disk time allows us to do better
- Using STCF and treating each 10ms chunk of A as a separate job, we might do something like Figure 7.9

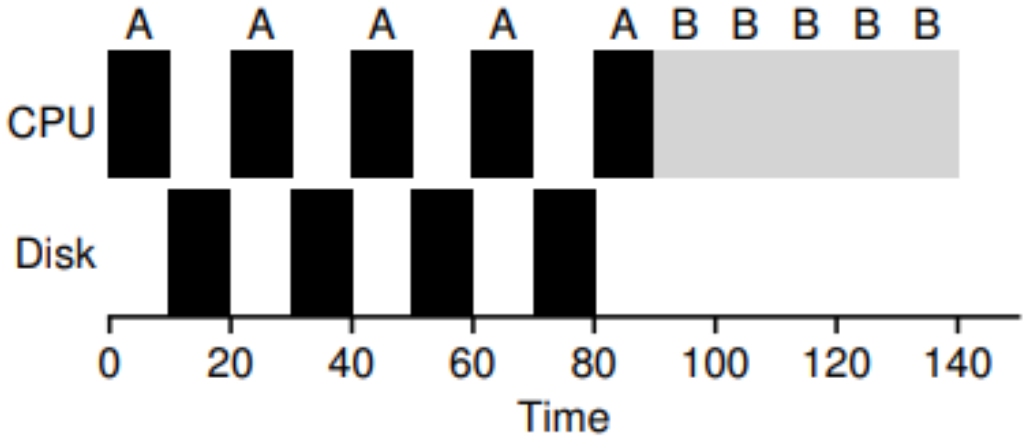

Figure 7.8: **Poor Use Of Resources**

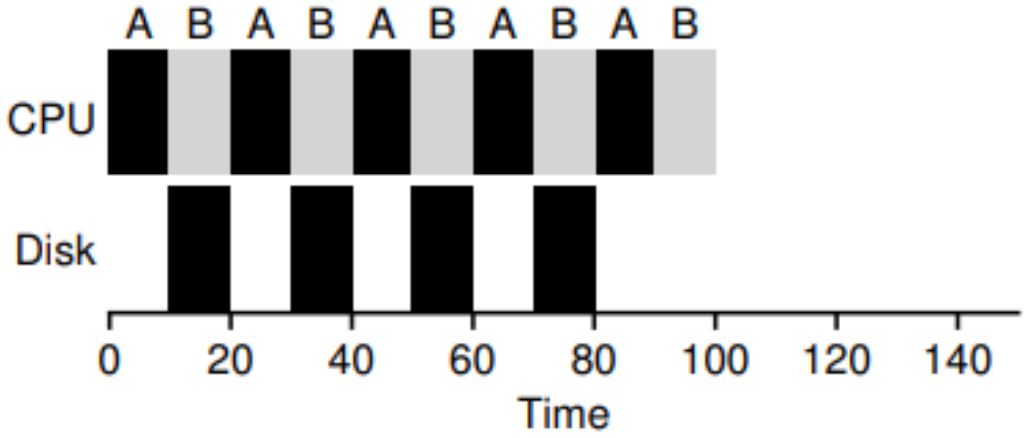

Figure 7.9: **Overlap Allows Better Use Of Resources**

Next Time...

- We've seen the trade off between response time and turnaround time
- We've seen basic idea of I/O handling
- Realistically, our last assumption is also terrible...we never know how long a program will run
- Next time we'll look at **multi-level feedback queue**