Lecture 07: Classical Scheduling

CS343 – Operating Systems

Branden Ghena – Fall 2020

Some slides borrowed from:
Stephen Tarzia (Northwestern), Shivaram Venkataraman (Wisconsin), and UC Berkeley CS162

Today's Goals

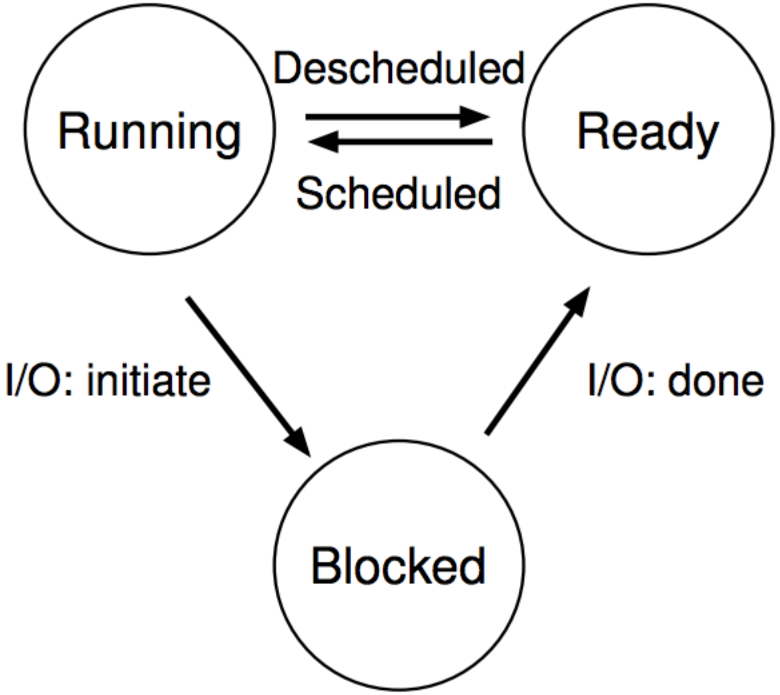
- Introduce the concept and challenges of scheduling
- Explore scheduling for batch and interactive systems
- Identify important metrics for measuring scheduler performance
- Examine several scheduling policies that target these metrics

Outline

- **Scheduling Overview**
- Batch Systems
    1. First In First Out scheduling
    2. Shortest Job First scheduling
    3. Shortest Remaining Processing Time scheduling
- Interactive Systems
    1. Round Robin scheduling
    2. Multi-Level Feedback Queue scheduling

Recall: processes don’t run all the time

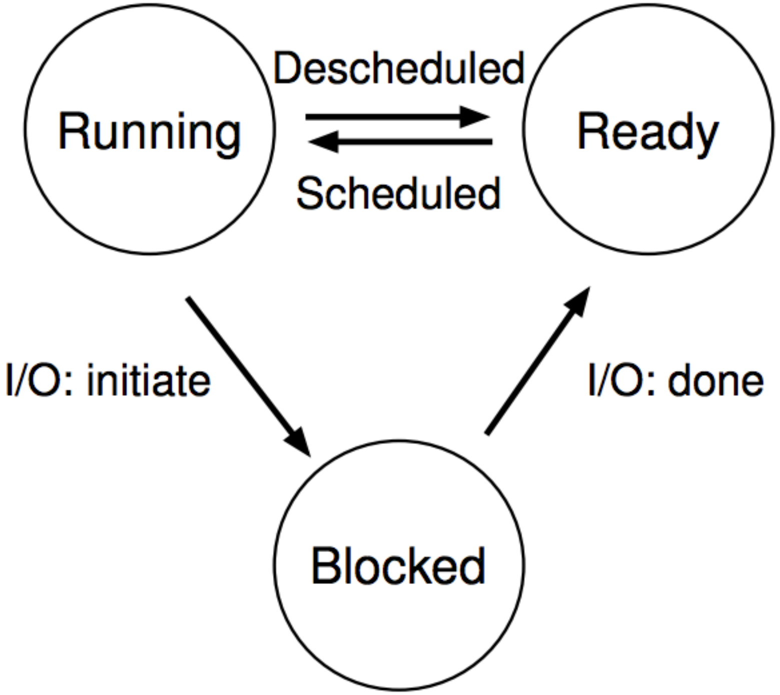
The three basic process states:



- OS ***schedules*** processes
  - Decides which of many competing processes to run.
- A ***blocked*** process is not ready to run.
- I/O means input/output – anything other than computing.
  - For example, reading/writing disk, sending network packet, waiting for keystroke, condvar/semaphore!
  - While waiting for results, the OS **blocks** the process, waiting to do more computation until the result is ready

Recall: multiprogramming processes

The three basic process states:



- When one process is Blocked, OS can schedule a different process that is Ready
- Even with a single processor, the OS can provide the illusion of many processes running simultaneously
  - And also use this opportunity to get more useful work done

Scheduling

- We know that multiple processes will be sharing the CPU
  - Possibly multiple threads in each process
  - Possibly multiple cores in the CPU

- Scheduling is creating a *policy* for sharing the CPU
  - Which process/thread is chosen to run, and when?
  - When (if ever) does the OS change which process is running?

When can the OS make scheduling decisions?

- Whenever the OS is actually running
  - i.e. after a context switch

- Possible triggers
  - System calls
    - Process/Thread creation/termination
    - I/O requests
    - Synchronization primitives (mutex/condvar/semaphore)
  - Interrupts
    - I/O complete
    - Timer triggers

Scheduling terminology

- Job - an execution unit handled by the scheduler
  - Thread or process
  - Moves between Ready and Blocked queues

- Workload – set of jobs
  - Arrival time of each job
  - Run time of each job

Scheduler Metrics

- Metric – standard for measuring something
  - Mathematical optimization: objective function
  - Economics: utility function

- For different computing scenarios, different metrics will be most important
  - Computing systems have different goals and uses
  - Performance metrics are often in conflict with each other

- Operating Systems are full of *tradeoffs*

Global scheduling metric

- Fairness
  - Each job should get a “fair” share of the processor

- Fair means different things of course
  - Could be “each job gets equal time”
  - Could be “each job starts in order it arrives”
  - Could be “each job is handled based on its priority”

- Scheduler should be fair with regards to the system it runs on

Different systems have different fairness requirements

- Example: network server
  - Request for home page
  - Request for contact page

- Example: personal computer
  - gedit that the user is actively interacting with
  - Compilation running in the background

- Example: autonomous vehicle
  - Image recognition algorithms
  - Radio

Different systems have different fairness requirements

- Example: network server – **Batch System**
  - Request for home page
  - Request for contact page
- Example: personal computer – **Interactive System**
  - gedit that the user is actively interacting with
  - Compilation running in the background
- Example: autonomous vehicle – **Real-time System**
  - Image recognition algorithms
  - Radio

Scheduling assumptions

1. Jobs all arrive at the same time
2. Each job runs for the same amount of time
3. Jobs cannot be stopped while executing
4. No new jobs are created while running existing jobs
5. Job runtime is known in advance
6. All jobs only use CPU (no I/O)
7. All jobs have equal priority
8. There is only one core (we’ll discuss this one next lecture)

Example: Random Scheduling

- Pick a job that is Ready at random
- Run job until it is complete (or blocked)
- Pick another job that is Ready at random

- Fairness
  - Each job has an equal chance of running
  - All jobs will *eventually* run

- If don't know anything about the jobs at all, this works at least
  - Also if we cannot distinguish among the jobs

Outline

- Scheduling Overview
- **Batch Systems**
  1. First In First Out scheduling
  2. Shortest Job First scheduling
  3. Shortest Remaining Processing Time scheduling
- Interactive Systems
  1. Round Robin scheduling
  2. Multi-Level Feedback Queue scheduling

What are batch systems?

- Systems designed to run a set of provided tasks
  - No direct interaction with users
  - Predominantly run-to-completion jobs

- Example: banking systems or payroll management

- Modern example: network servers
  - Tasks are serving requests
  - Multiple types of requests, each with known runtimes

Metrics for batch systems

- Throughput
  - Jobs completed per unit time
  - Throughput = jobs_completed / total_duration
  - Higher is better

- Turnaround time
  - Duration from job arrival until job completion
  - $T_{turnaround} = T_{completion} - T_{arrival}$
  - Lower is better
  - Average turnaround time is computed across all jobs

Check your understanding – throughput and turnaround

- Process A arrives at t=10, finishes at t=40
- Process B arrives at t=10, finishes at t=60

Throughput = jobs_completed / total_duration

$T_{turnaround} = T_{completion} - T_{arrival}$

Throughput

Turnaround for A **Turnaround for B** **Average Turnaround**

Check your understanding – throughput and turnaround

- Process A arrives at t=10, finishes at t=40 (duration 30)
- Process B arrives at t=10, finishes at t=60 (duration 20)

Throughput = jobs_completed / total_duration

$T_{turnaround} = T_{completion} – T_{arrival}$

Throughput
2 jobs / 50 time = 0.04

Turnaround for A
40-10 = 30

Turnaround for B
60-10 = 50

Average Turnaround
(30+50)/2 = 40

Check your insight – batch scheduler metric

- Which metric is most relevant to a batch system scheduler?
  - Throughput or Turnaround

Check your insight – batch scheduler metric

- Which metric is most relevant to a batch system scheduler?
  - Throughput or Turnaround

- Throughput only cares about sum of durations of jobs
  - Throughput is the same no matter whether A or B goes first

- Turnaround accounts for delays in scheduling a job
  - Swapping A and B would result in better average turnaround

Turnaround for A	**Turnaround for B**	**Average Turnaround**
60-10 = 50	30-10 = 20	(50+20)/2 = 35

Schedulers for batch systems

1. First In First Out
2. Shortest Job First
3. Preemptive Shortest Remaining Processing Time

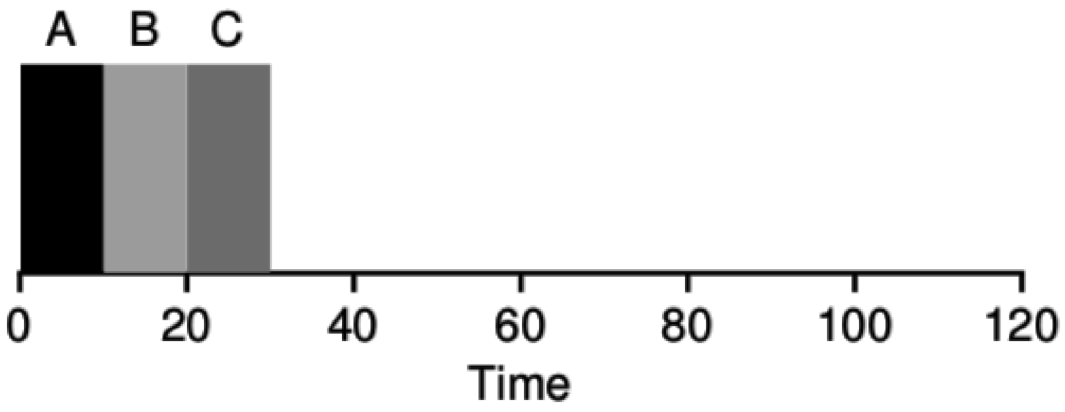
Revisiting scheduling assumptions

1. ~~**Jobs all arrive at the same time**~~
   Jobs have arrival times
2. Each job runs for the same amount of time
3. Jobs cannot be stopped while executing
4. No new jobs are created while running existing jobs
5. Job runtime is known in advance
6. All jobs only use CPU (no I/O)
7. All jobs have equal priority

1. FIFO Scheduling

- First In, First Out (FIFO)
  - also known as First Come First Served (FCFS)
  - assumption: scheduler doesn't start jobs until after they all arrive

- Schedule first job to arrive first
- Let a job continue until it is complete
- Then schedule next remaining job with earliest arrival



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

Revisiting scheduling assumptions

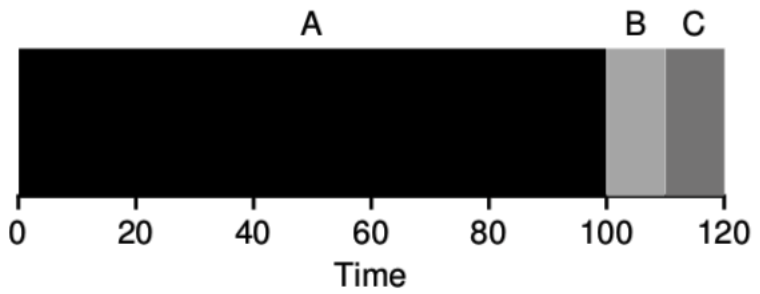
1. ~~Jobs all arrive at the same time~~
   Jobs have arrival times
2. **~~Each job runs for the same amount of time~~**
   Jobs can have different run durations
3. Jobs cannot be stopped while executing
4. No new jobs are created while running existing jobs
5. Job runtime is known in advance
6. All jobs only use CPU (no I/O)
7. All jobs have equal priority

Check your understanding – FIFOs with different durations

- What is a problematic scenario for FIFO scheduling?

Check your understanding – FIFOs with different durations

- What is a problematic scenario for FIFO scheduling?
- One big job can cause lots of jobs behind it to wait
    - Convoy effect – lots of small jobs stuck behind one big job

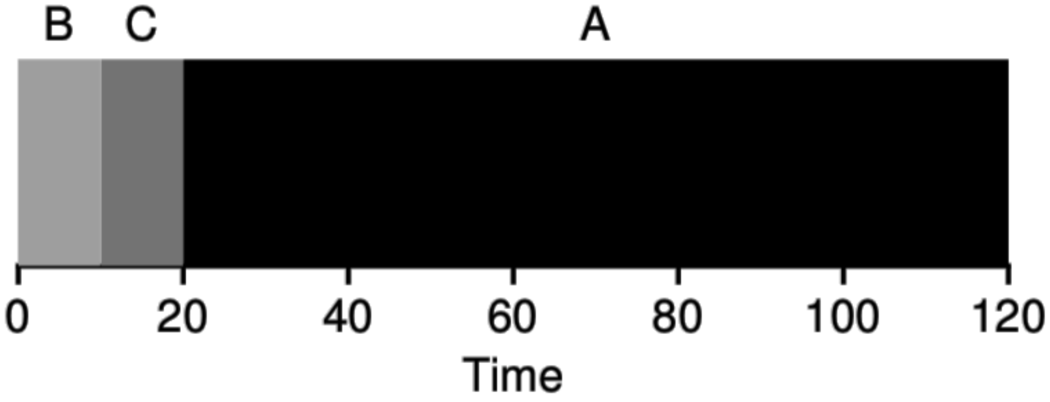


- Average turnaround time = (100+110+120)/3 = 110
    - Minimum average turnaround time = (10+20+120)/3 = 50

2. Shortest Job First

- Schedule job with smallest duration first
- Let a job continue until it is complete
- Then schedule next remaining job with smallest duration

- Essentially: complete a job as soon as possible
  - Minimizes the number of waiting jobs, minimizing average turnaround



Average Turnaround
(10+20+120)/3 = 50

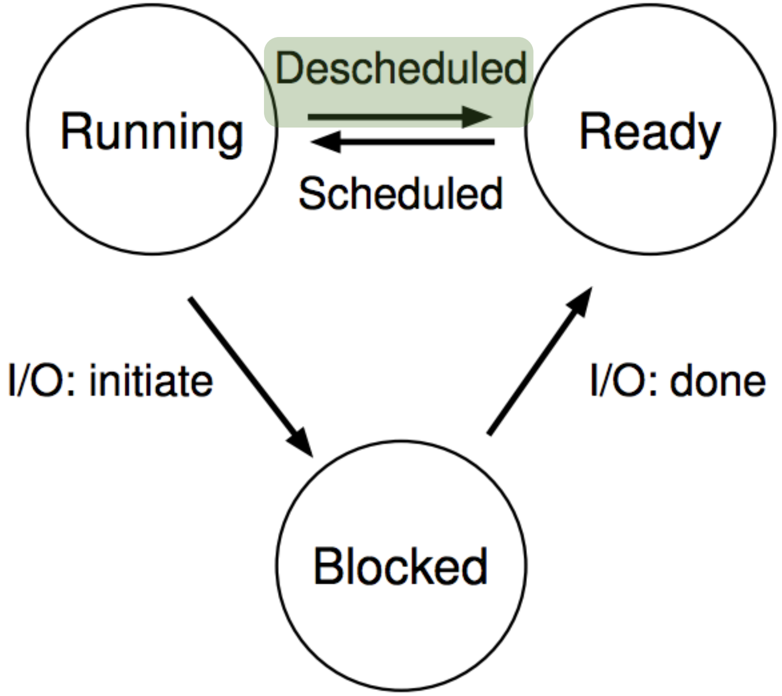
Revisiting scheduling assumptions

1. ~~Jobs all arrive at the same time~~
   Jobs have arrival times
2. ~~Each job runs for the same amount of time~~
   Jobs can have different run durations
3. **~~Jobs cannot be stopped while executing~~**
   Jobs can be *preempted* during execution
4. No new jobs are created while running existing jobs
5. Job runtime is known in advance
6. All jobs only use CPU (no I/O)
7. All jobs have equal priority

Preemption

- OS can "deschedule" jobs that are running

- This means it can make scheduling decisions more frequently
  - System calls
  - Interrupts
  - Timers



Context switching overhead

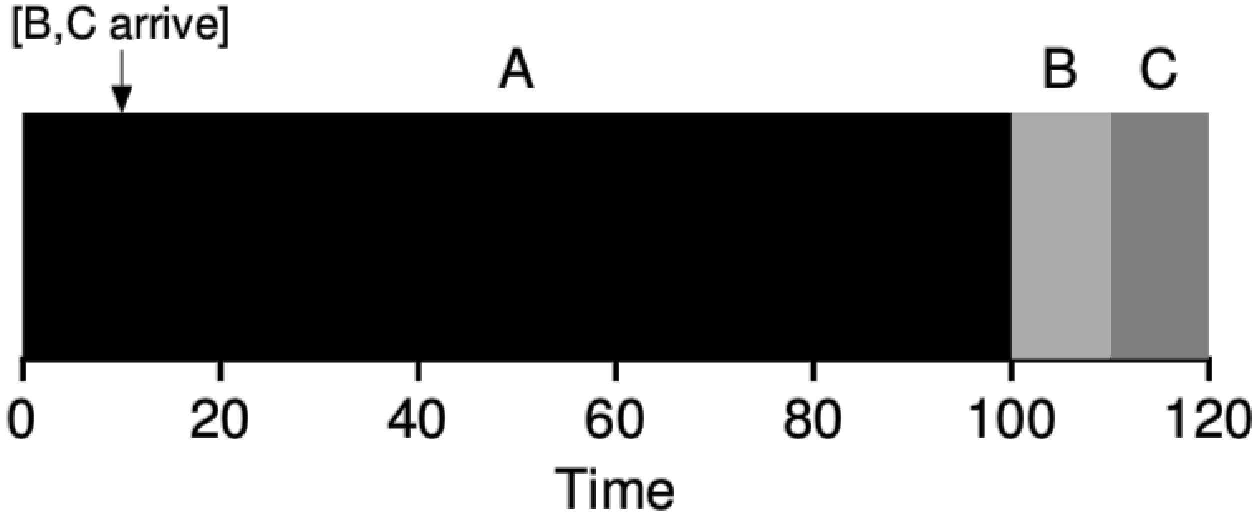
- Switching processes is expensive
  - Context switch to OS is on the order of 1 μs
  - Switching registers and CPU mode
- Memory is often the larger expense though
  - New process has different physical memory pages
  - Which means that caches have to be cleared
  - Caches will “warm up” as the process runs
  - Less of a penalty to threads (only stack changes)
- Alternative option: cooperative scheduling through yield()

Revisiting scheduling assumptions

1. ~~Jobs all arrive at the same time~~
   Jobs have arrival times
2. ~~Each job runs for the same amount of time~~
   Jobs can have different run durations
3. **~~Jobs cannot be stopped while executing~~**
   Jobs can be *preempted* during execution
4. **~~No new jobs are created while running existing jobs~~**
   Jobs can be created at any time
5. Job runtime is known in advance
6. All jobs only use CPU (no I/O)
7. All jobs have equal priority

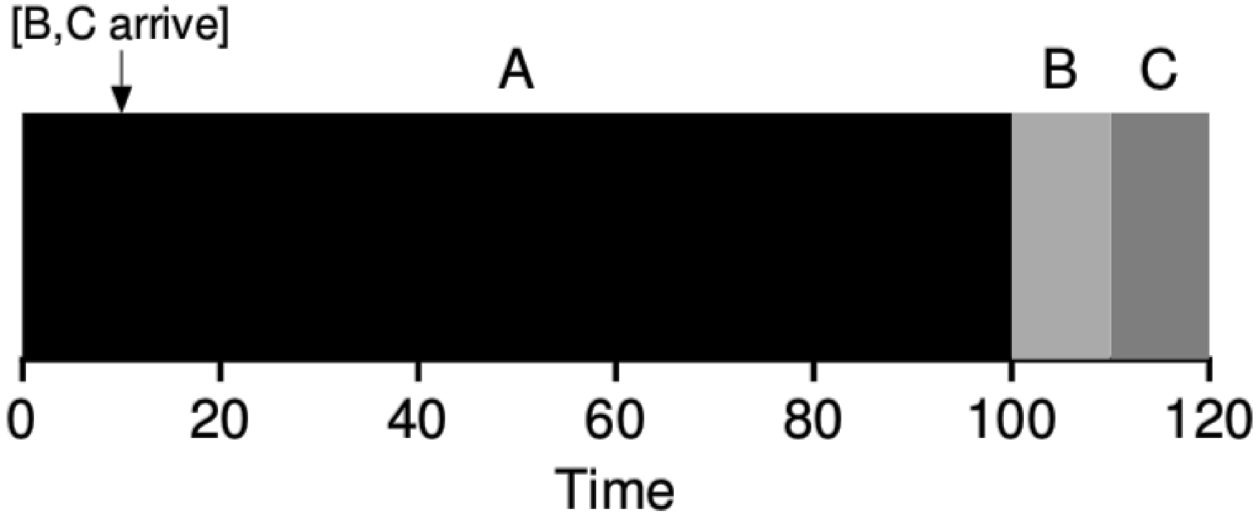
Shortest Job First can fail with late arrivals

- Scheduler's previously optimal decision could be invalidated by new job arrivals
  - If B and C arrive late, they will have to wait because A is already running



Check your understanding

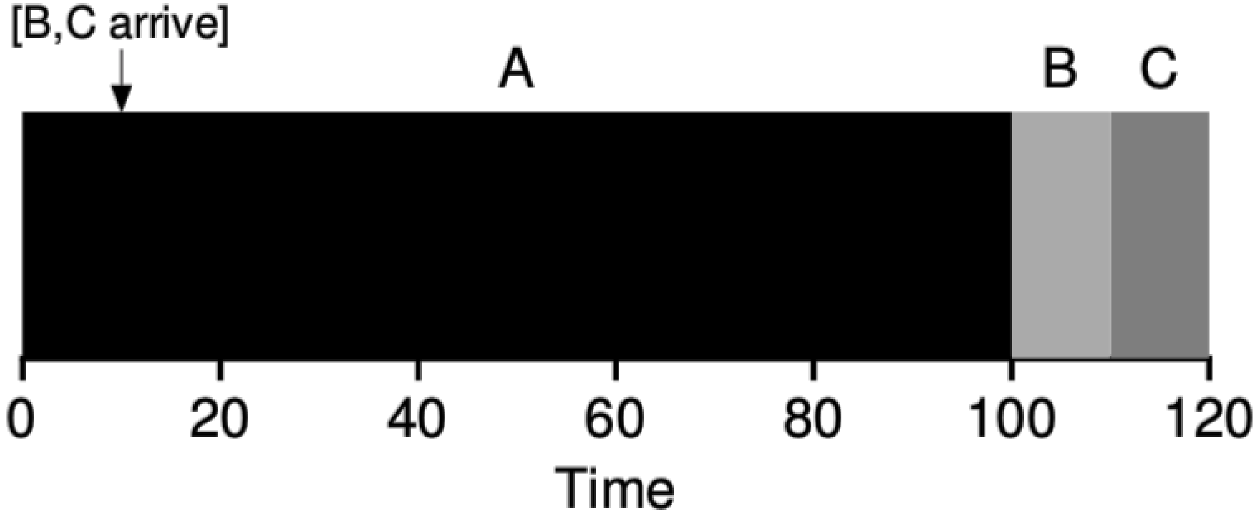
- What is the average turnaround time for this example?
  - B and C arrive at time 10



Check your understanding

- What is the average turnaround time for this example?
  - B and C arrive at time 10

- Average turnaround = (100 + (110-10) + (120-10))/3 = 103.3



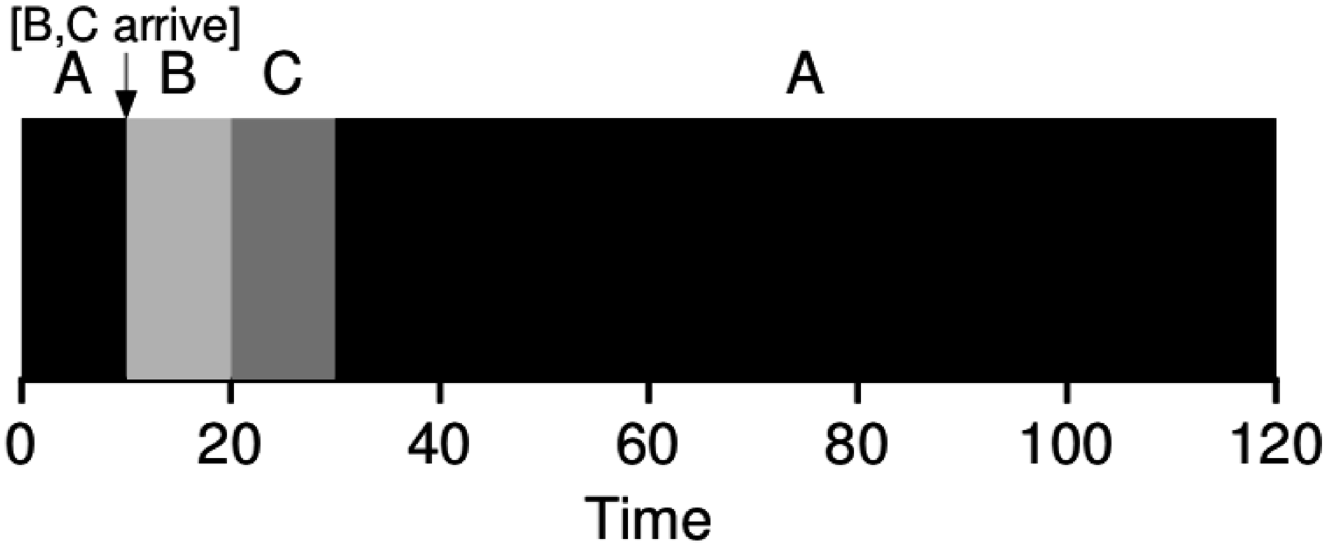
3. Preemptive Shortest Remaining Processing Time

- Also known as Shortest Time-to-Completion First
- Schedule job with smallest duration first
- Preempt a running job when new jobs arrive
- Then schedule job with smallest remaining duration

- Essentially, reevaluate schedule when new information is gained

Shortest Remaining Processing Time example

- A is preempted when B and C arrive at time 10
- Scheduler choose B as new shortest remaining time
  - B=10, C=10, A=90



Average Turnaround
(120+10+20)/3 = 50

Starvation and scheduling

- Starvation can occur in schedulers as well
  - When one job will never actually get a chance to run

- We've discussed:
  - FIFO, Shortest Job First, and Shortest Remaining Processing Time
  - Which of these can exhibit starvation?

Starvation and scheduling

- Starvation can occur in schedulers as well
    - When one job will never actually get a chance to run

- We've discussed:
    - FIFO, Shortest Job First, and Shortest Remaining Processing Time
    - Which of these can exhibit starvation?
        - Shortest Remaining Processing Time
        - Shortest Job First too if we allow new job arrivals (without preemption)

    - Arriving short tasks could lead a long task to never be scheduled

Outline

- Scheduling Overview
- Batch Systems
  1. First In First Out scheduling
  2. Shortest Job First scheduling
  3. Shortest Remaining Processing Time scheduling
- **Interactive Systems**
  1. Round Robin scheduling
  2. Multi-Level Feedback Queue scheduling

What are interactive systems?

- Every computer you directly interact with
  - Desktops, laptops, smartphones
- Differences from batch systems
  - Humans are “in-the-loop”
    - Computer needs to feel responsive for programs they are using
  - Many jobs have no predefined duration
    - How long does Chrome run for?
- Still have some batch jobs though (background services)

Metric for interactive systems

- Response time
  - Time from arrival until the job begins execution
  - Doesn't matter how long the job takes to run
  - $T_{response} = T_{start} – T_{arrival}$

- Particularly good for interactive processes
  - Need to quickly show that they are reacting to user inputs
  - Exact total run duration isn't so important though

Schedulers for interactive systems

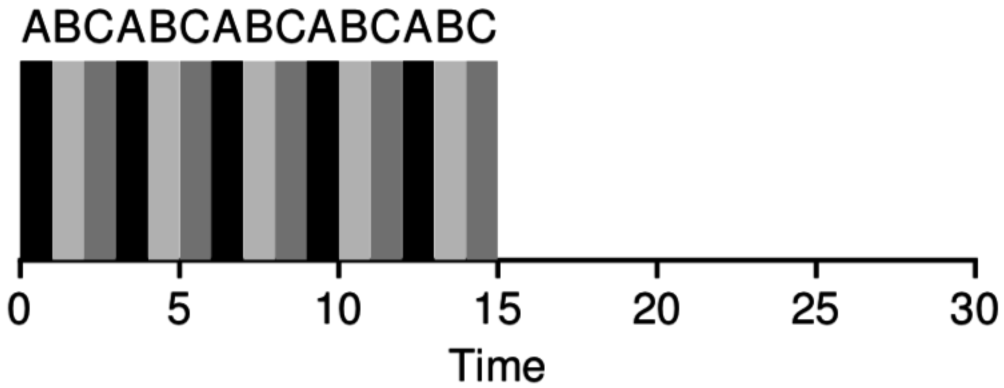
1. Round Robin
2. Multi-Level Feedback Queue

Revisiting scheduling assumptions

1. ~~Jobs all arrive at the same time~~
   Jobs have arrival times
2. ~~Each job runs for the same amount of time~~
   Jobs can have different run durations
3. ~~Jobs cannot be stopped while executing~~
   Jobs can be *preempted* during execution
4. ~~No new jobs are created while running existing jobs~~
   Jobs can be created at any time
5. **~~Job runtime is known in advance~~**
   Job runtime is unknown
6. All jobs only use CPU (no I/O)
7. All jobs have equal priority

1. Round Robin

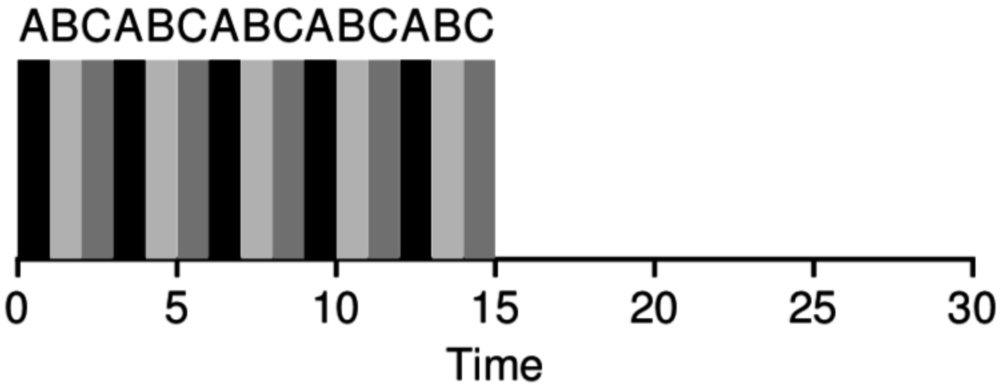
- Round Robin scheduling runs a job for a small *timeslice* (quanta), then schedules the next job

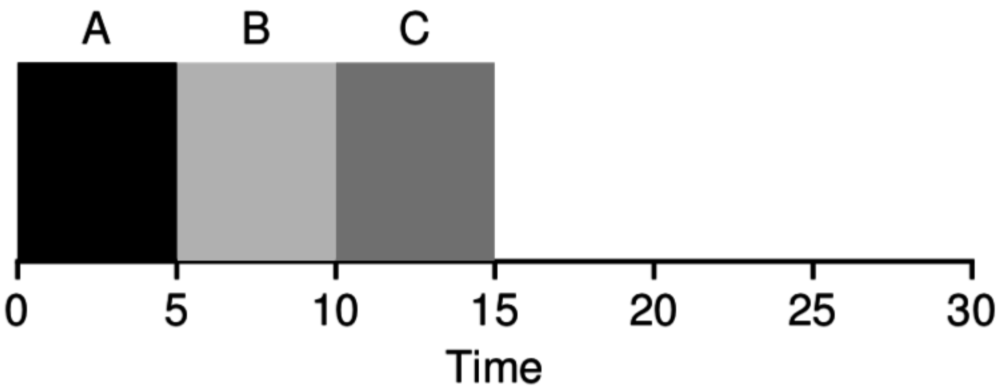


- If all jobs arrive at time 0
  - Average response time = (0 + 1 + 2)/3 = 1

- Smaller timeslice means smaller response time

Check your understanding





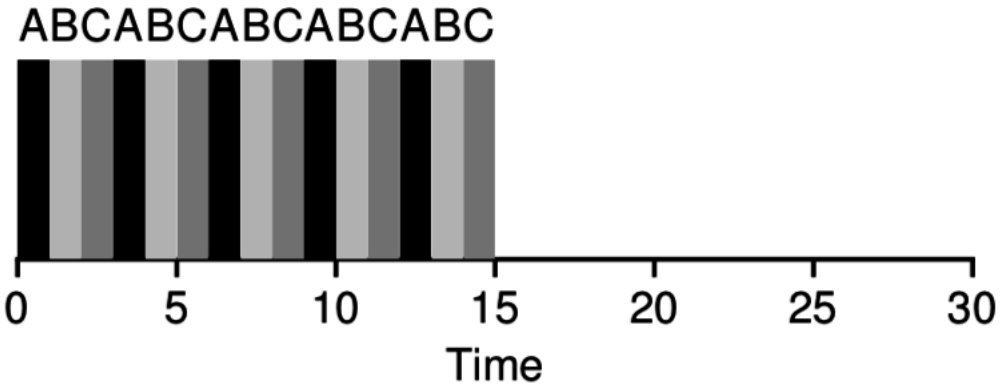
Round Robin scheduling:

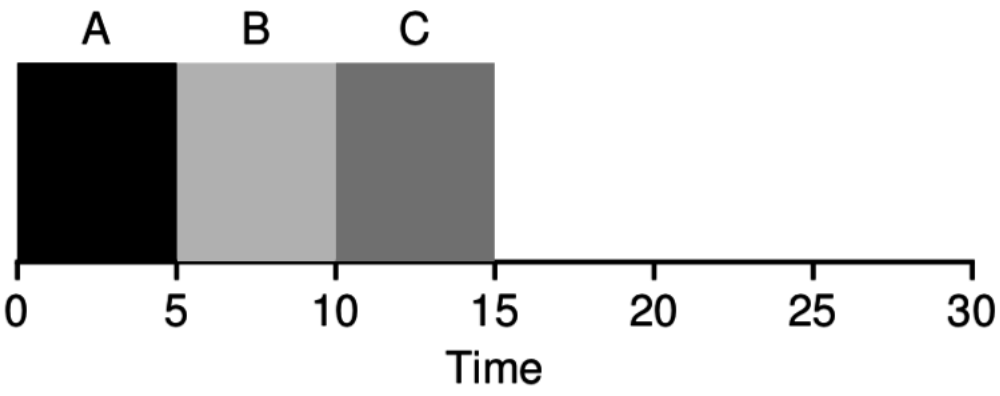
- Avg turnaround time =
- Avg response time =

Shortest Job first or ***SRPT***:

- Avg turnaround time =
- Avg response time =

Different policies favor different metrics





Round Robin scheduling:

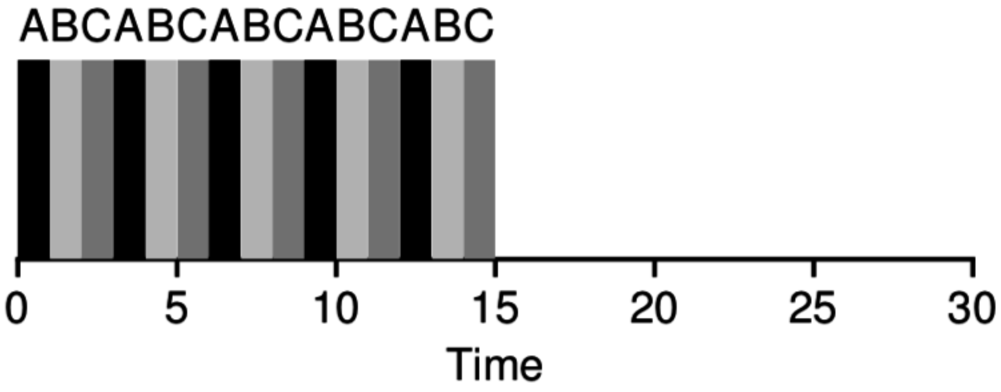
- Avg turnaround time = **14**
- Avg response time = **1**

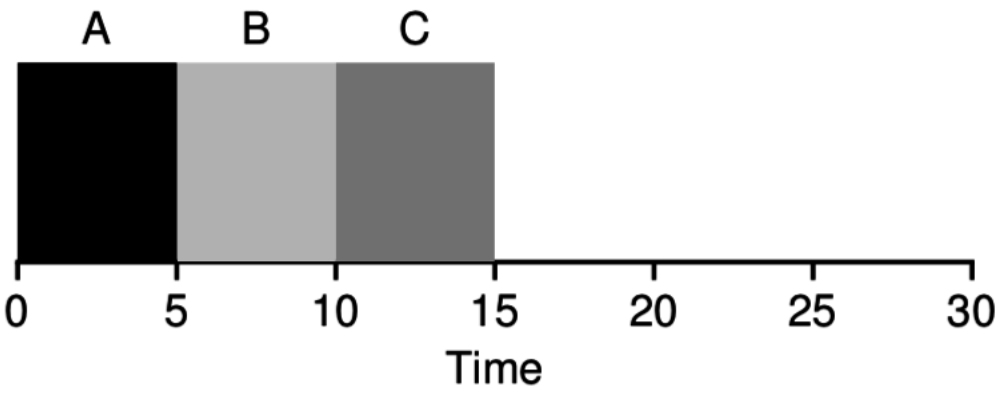
Shortest Job first or ***STCF***:

- Avg turnaround time = **10**
- Avg response time = **5**

Better response time versus Better turnaround time

Remember, context switches are not free





Round Robin scheduling:

- Context switches = **14**

Shortest Job first or ***STCF***:

- Context switches = **2**

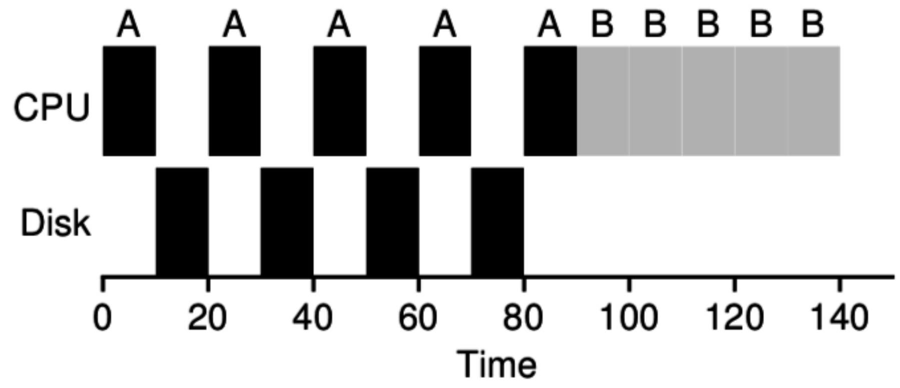
- In a real OS, Round Robin would take an extra ~12 μs
  - Plus more time lost with cold caches…
- Timeslice must be much greater than context switch time ~1 ms

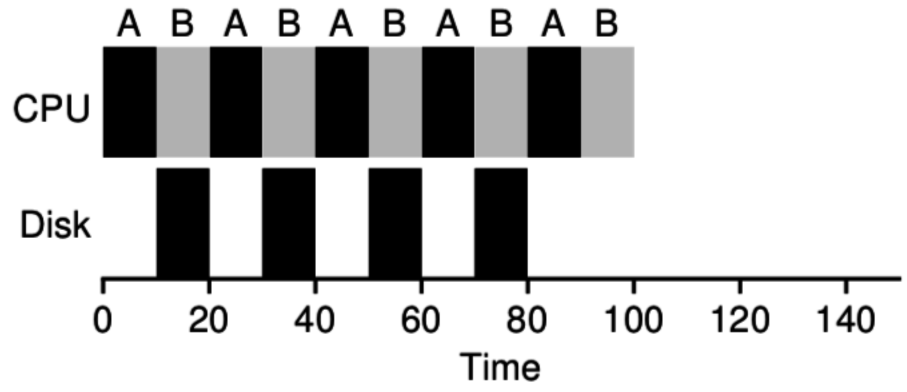
Revisiting scheduling assumptions

1. ~~Jobs all arrive at the same time~~
   Jobs have arrival times
2. ~~Each job runs for the same amount of time~~
   Jobs can have different run durations
3. ~~Jobs cannot be stopped while executing~~
   Jobs can be *preempted* during execution
4. ~~No new jobs are created while running existing jobs~~
   Jobs can be created at any time
5. ~~Job runtime is known in advance~~
   Job runtime is unknown
6. **~~All jobs only use CPU (no I/O)~~**
   Jobs can make I/O requests that block
7. All jobs have equal priority

I/O creates scheduling ***overlap*** opportunities

- If job A does I/O every ten milliseconds and each I/O takes 10 ms, then the CPU is free during those I/Os:
- A is **blocked** during its I/O.
  - It's just waiting for data from the disk
  - But it does not need the CPU
- We can schedule another job during process A's I/O





Jobs can be I/O-bound or CPU-bound

- CPU-bound process
    - Lots of computation between each I/O request
    - Actually needs to do computation on a processor
    - Example: doing matrix math

- I/O-bound process
    - Very little computation between each I/O request
    - Just needs a processor to figure out its next I/O request
    - Example: walking the disk

Scheduling goal: I/O before Processors

- First maximize I/O
  - Run the I/O-bound jobs as quickly as possible,
  - So they can send next I/O request,
  - And our disks, network cards, etc. are maximally used

- Then fill up the processor(s)
  - Lots of room for multiprogramming between the I/O requests
  - Blocked jobs are still “progressing” as their I/O is fetched

Scheduling goal: I/O before Processors

- First maximize I/O
  - Run the I/O-bound jobs as quickly as possible,
  - So they can send next I/O request,
  - And our disks, network cards, etc. are maximally used

- Then fill up the processor(s)
  - Lots of room for multiprogramming between the I/O requests
  - Blocked jobs are still “progressing” as their I/O is fetched

- But how do you know when a job is going to use I/O?
  - Can’t know the future
  - Can track past behavior of the job

Revisiting scheduling assumptions

1. ~~Jobs all arrive at the same time~~
   Jobs have arrival times
2. ~~Each job runs for the same amount of time~~
   Jobs can have different run durations
3. ~~Jobs cannot be stopped while executing~~
   Jobs can be *preempted* during execution
4. ~~No new jobs are created while running existing jobs~~
   Jobs can be created at any time
5. ~~Job runtime is known in advance~~
   Job runtime is unknown
6. ~~All jobs only use CPU (no I/O)~~
   Jobs can make I/O requests that block
7. **~~All jobs have equal priority~~**
   Jobs have individual priority

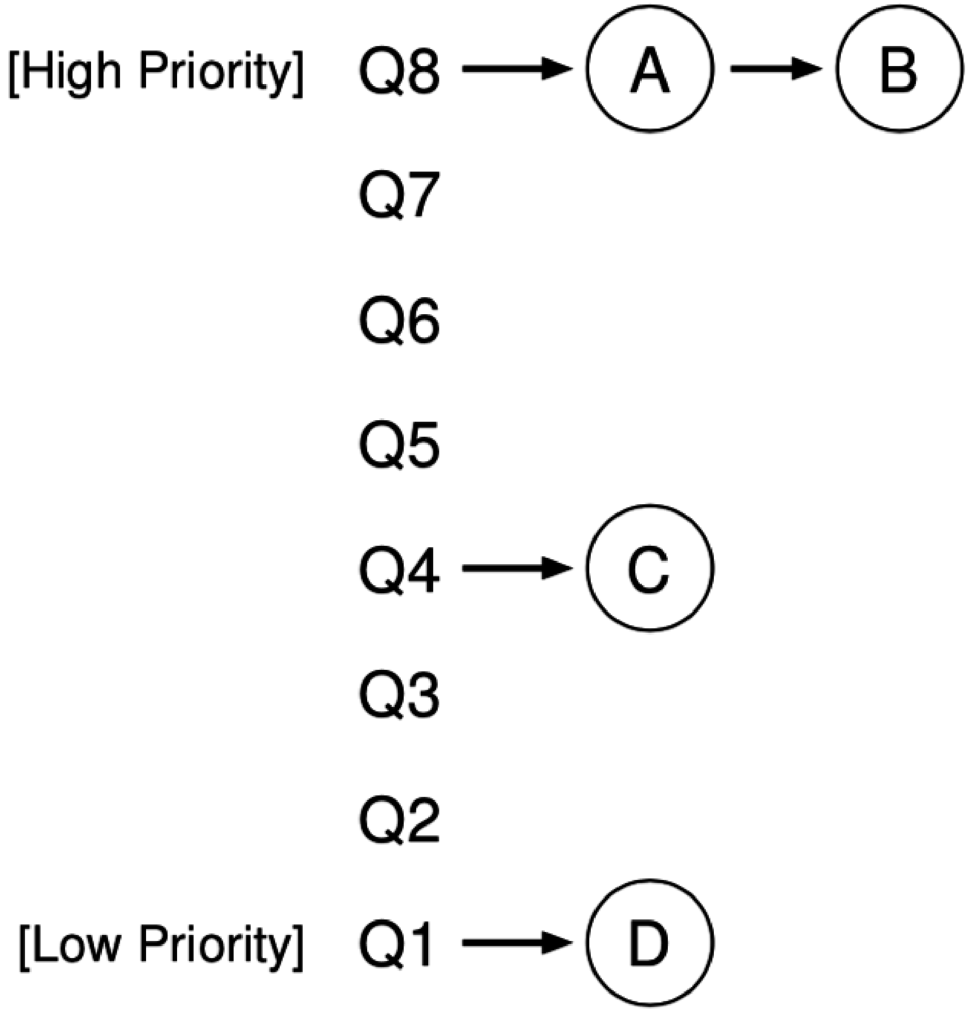
2. Multi-Level Feedback Queue (MLFQ)

- General purpose scheduler to support several goals
  - Good response time for interactive jobs
  - Good turnaround time for batch jobs
  - Prioritize I/O bound jobs over CPU bound jobs

- Approach
  - Automatically attach priority to jobs:
    - Interactive, I/O bound jobs should be highest priority
    - CPU bound, batch jobs should be lowest priority
    - Apply different round robin timeslices to each priority level

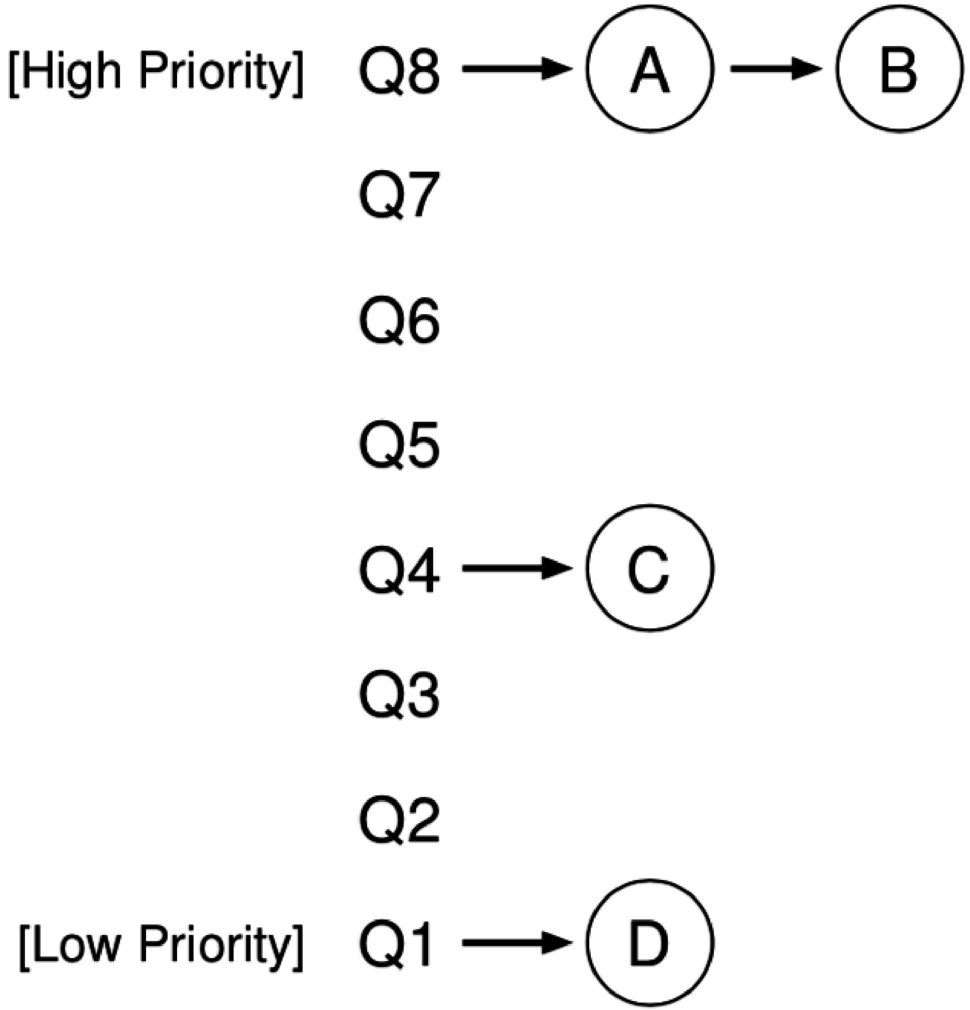
Multi-Level Feedback Queue Details

- Run highest priority level available
  - Round robin among jobs there
- When all jobs at a level are blocked on I/O
  - Move down to next lower level
- Long running jobs lose priority
  - Processor usage quota at a given level
  - When used up, demote job one level

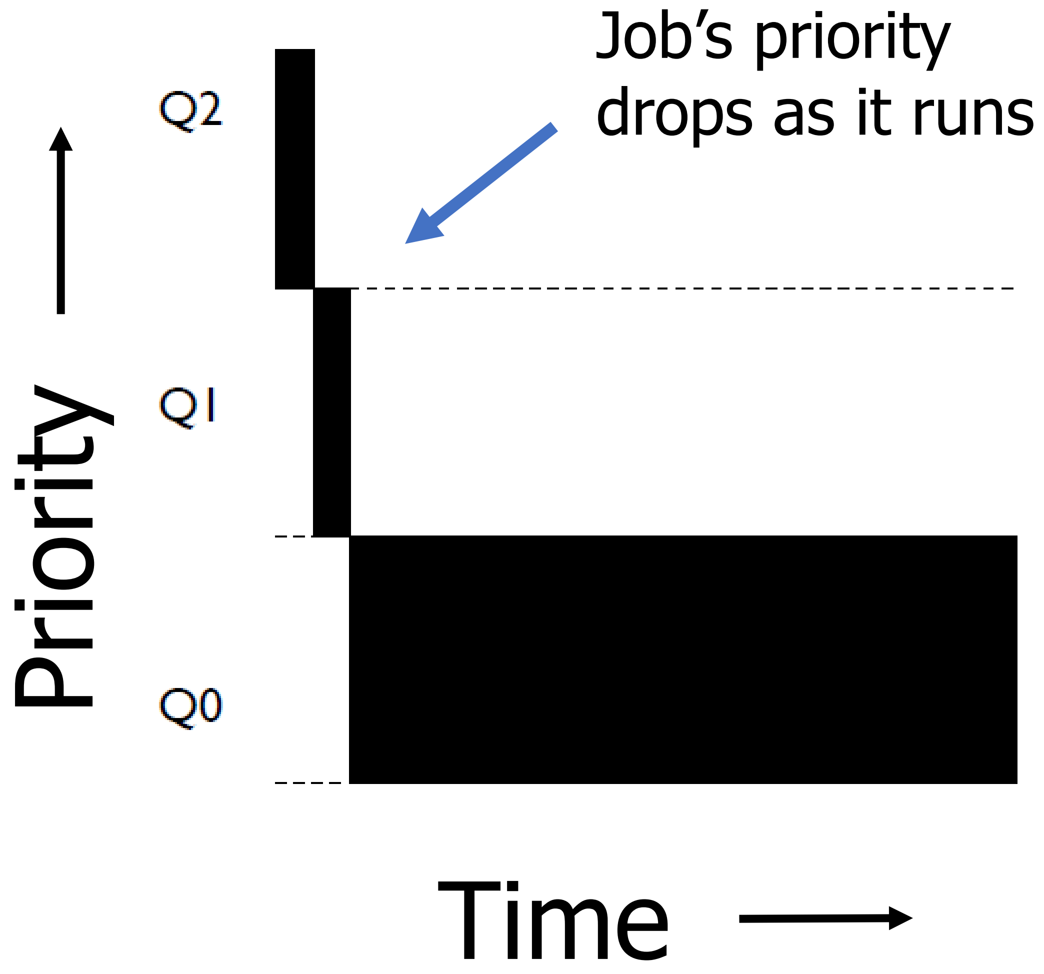


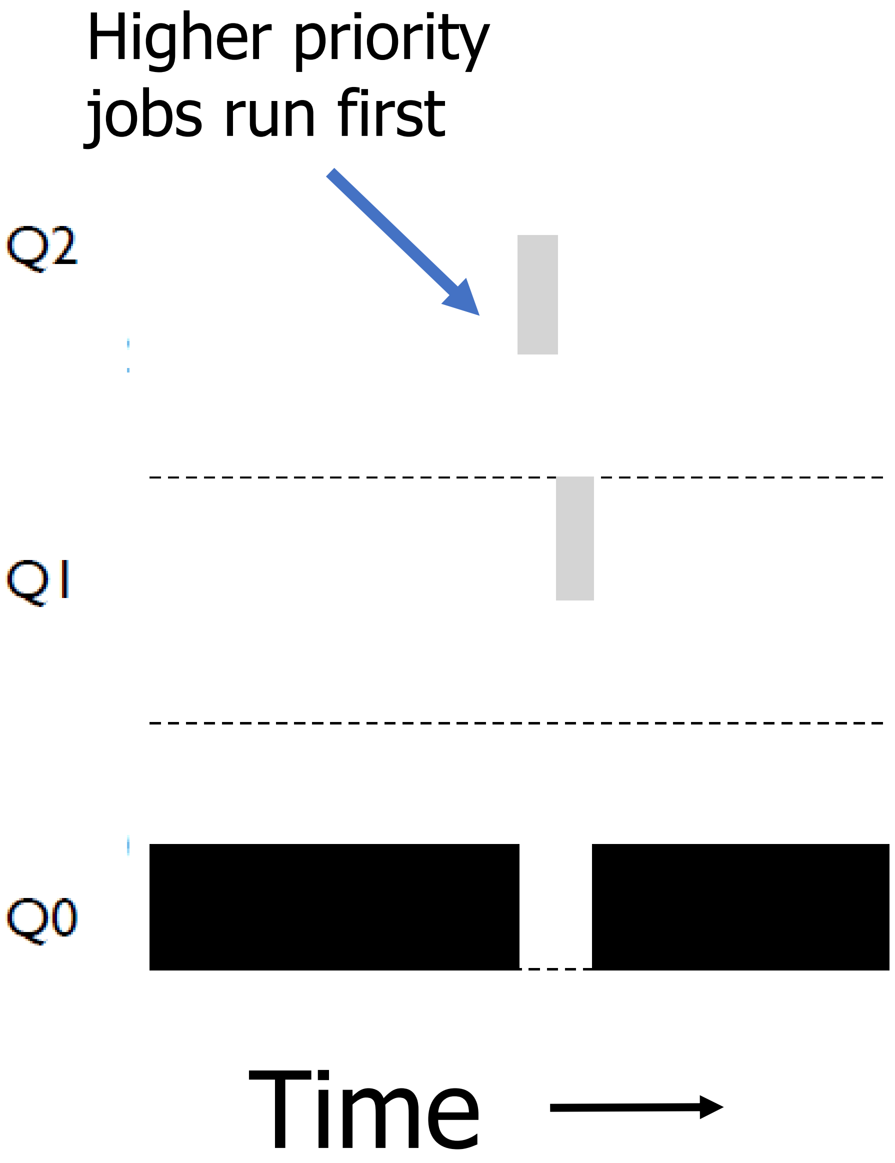
MLFQ Rules

1. If Priority($\mathbf{J_1}$) > Priority($\mathbf{J_2}$), $\mathbf{J_1}$ runs
2. If Priority($\mathbf{J_1}$) = Priority($\mathbf{J_2}$), $\mathbf{J_1}$ and $\mathbf{J_2}$ run in Round Robin
3. Jobs start at top priority
4. When a job uses its time quota for a level, demote it one level
5. Every **S** seconds, reset priority of all jobs to top



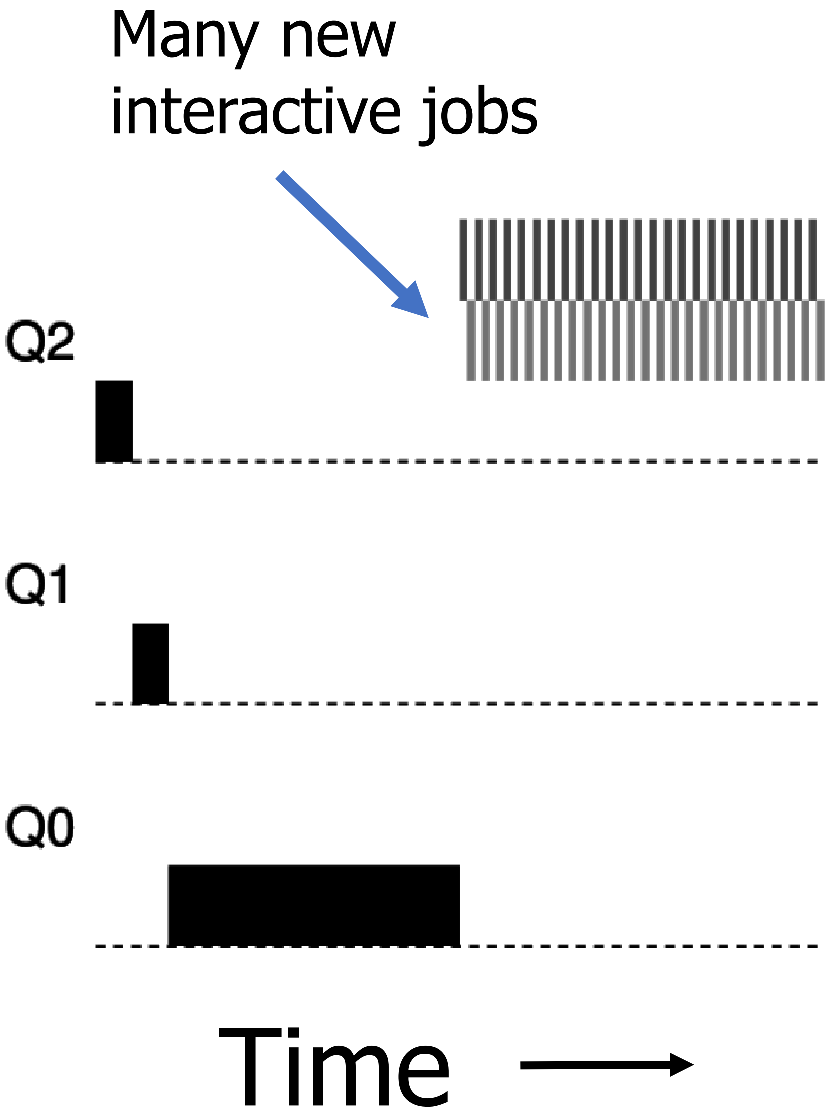
MLFQ Example

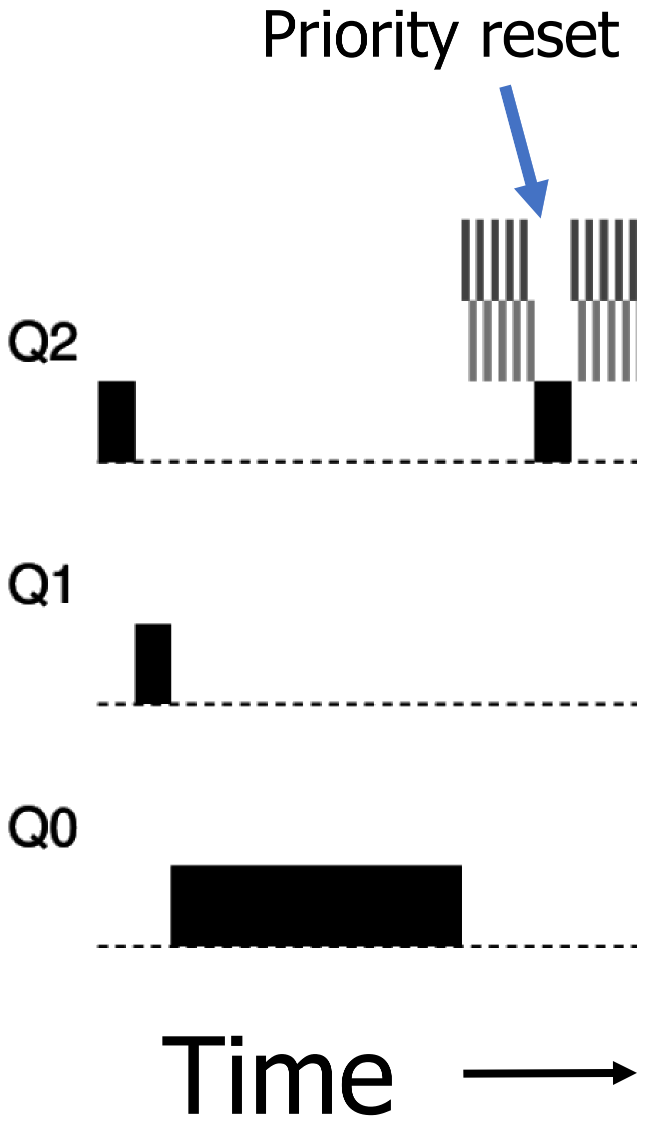




MLFQ avoids starvation with periodic priority reset

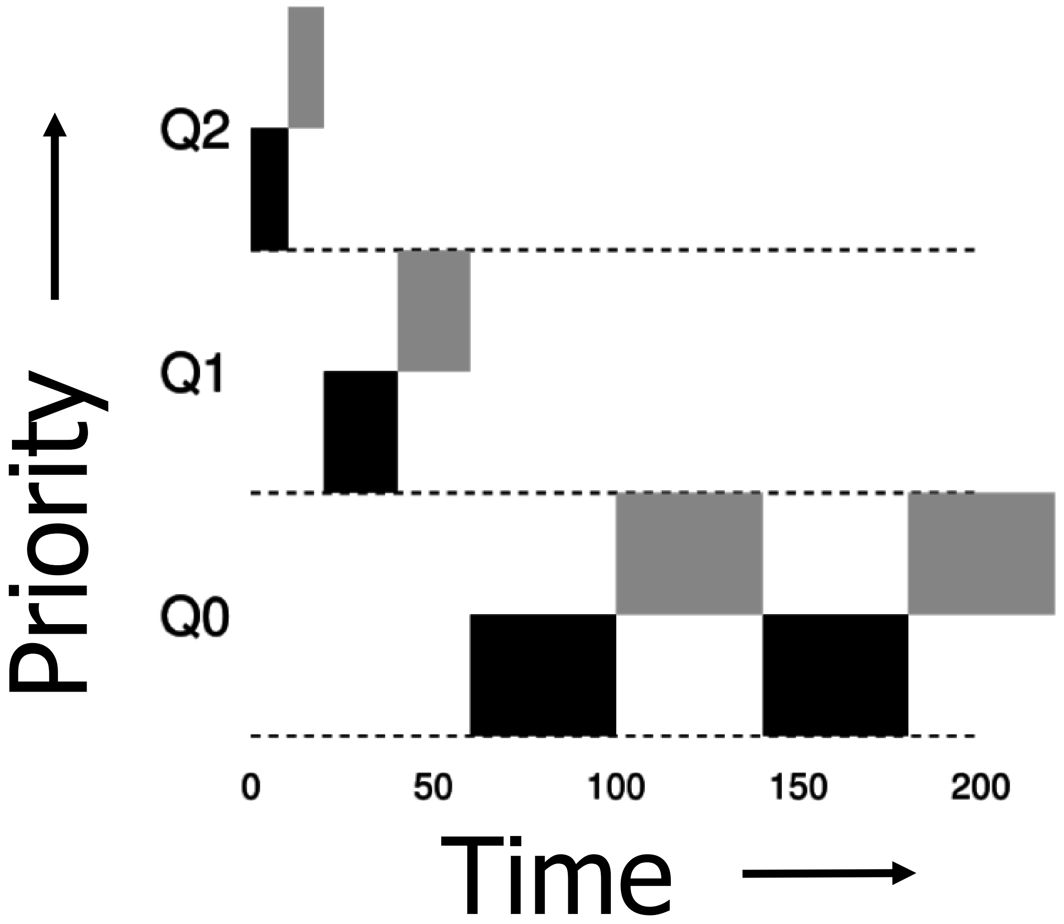
- Low priority jobs could starve if there are enough interactive jobs
- MLFQ avoids starvation by periodically resetting priorities





Change timeslices to optimize response and turnaround

- Lower priority jobs are CPU bound, not interactive
  - So we can use longer timeslices to minimize context switches



Outline

- Scheduling Overview
- Batch Systems (Metric: turnaround time)
    1. First In First Out scheduling
    2. Shortest Job First scheduling
    3. Shortest Remaining Processing Time scheduling
- Interactive Systems (Metric: response time)
    1. Round Robin scheduling
    2. Multi-Level Feedback Queue scheduling