INF113: Scheduling

Kirill Simonov

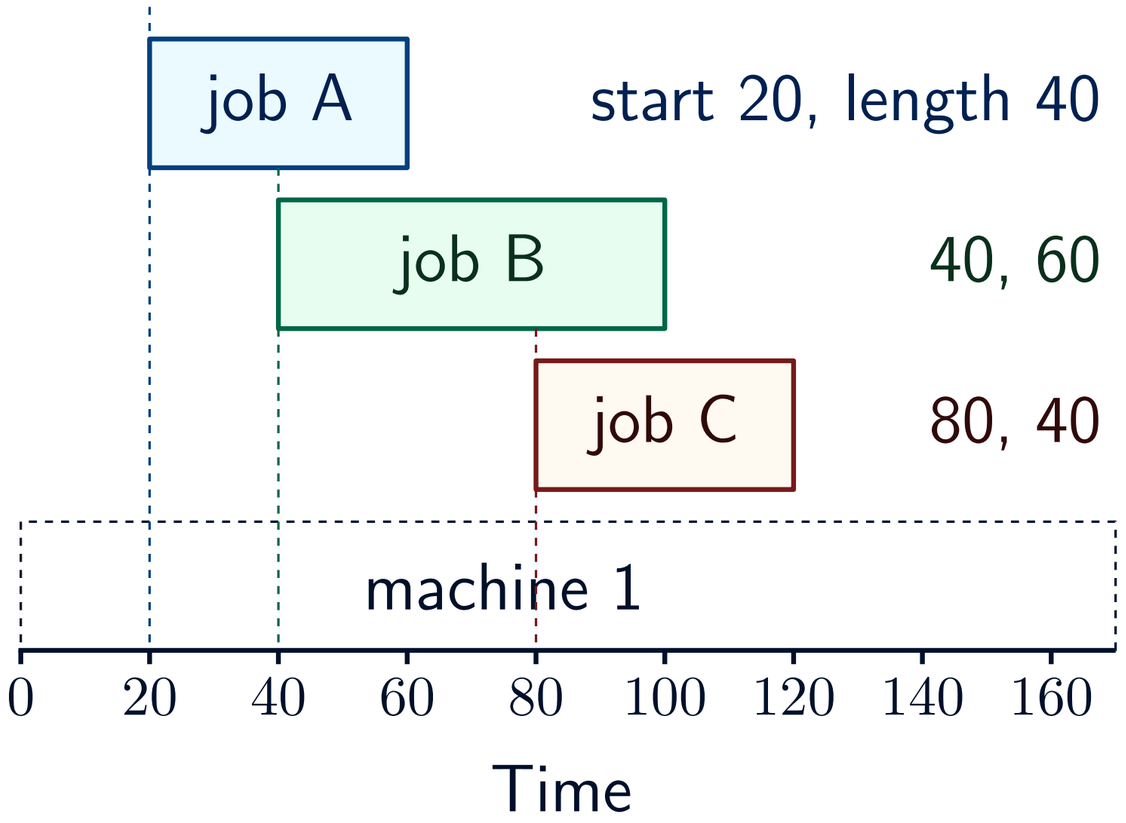
10.09.2025

Questions for today

- How to model scheduling?
- How to measure that a schedule is “good”?
- How to schedule fast and well without prior knowledge?

Scheduling in general

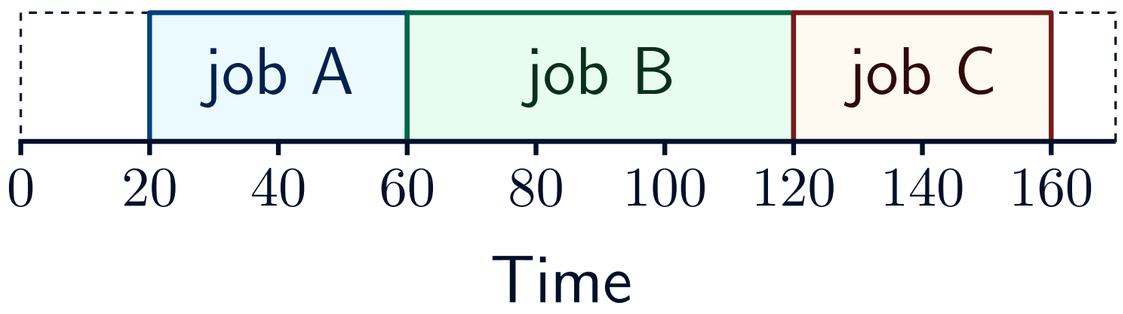
- A broad class of optimization problems can be viewed as scheduling
- **Input**: a collection of jobs, and a number of machines
- Each job is defined by
  - length
  - arrival time
- Each machine serves jobs, one at a time
- **Task:** Come up with an optimal schedule



Scheduling in general

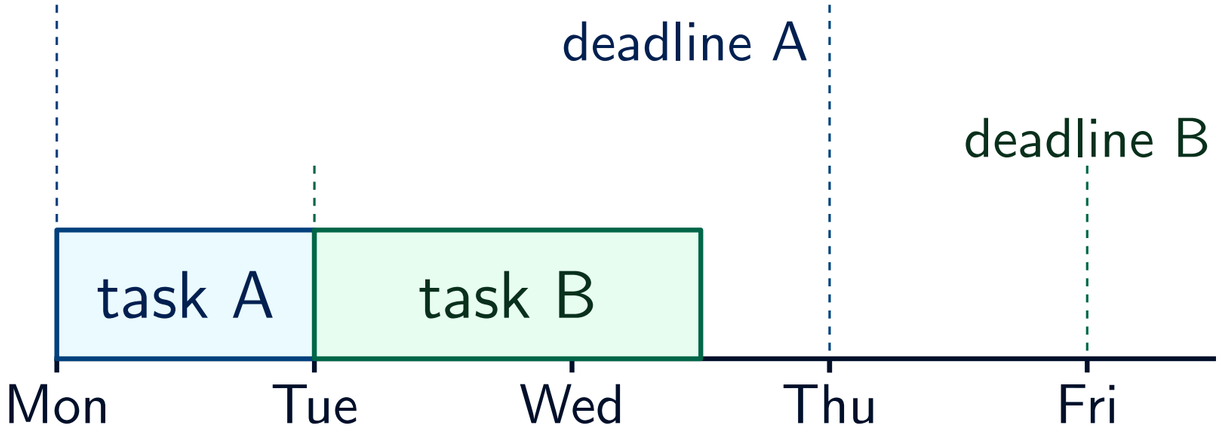
- A broad class of optimization problems can be viewed as scheduling
- **Input**: a collection of jobs, and a number of machines
- Each job is defined by
  - length
  - arrival time
- Each machine serves jobs, one at a time
- **Task:** Come up with an optimal schedule

what is optimal?..

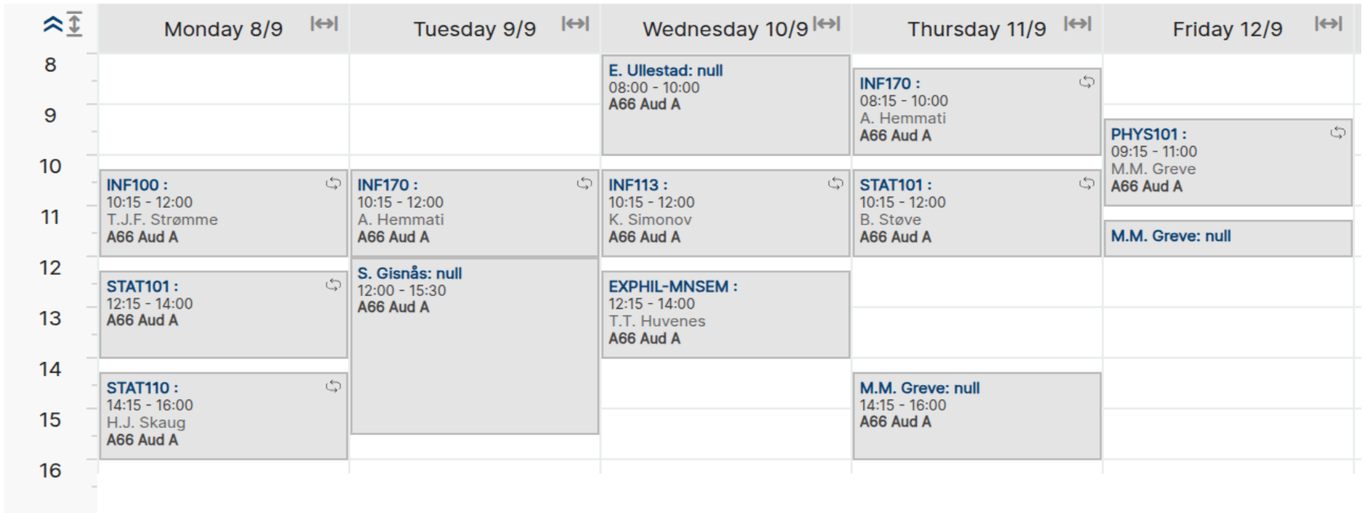


Examples of scheduling

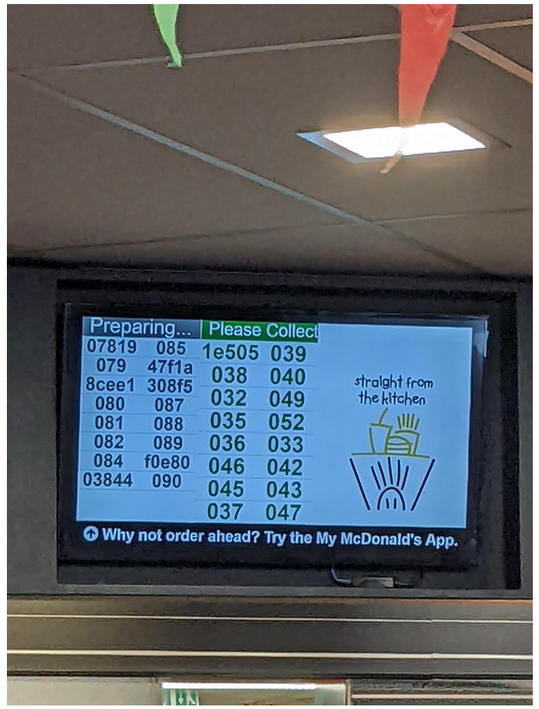
- Completing homework



- Arranging classes



- Food/drink orders

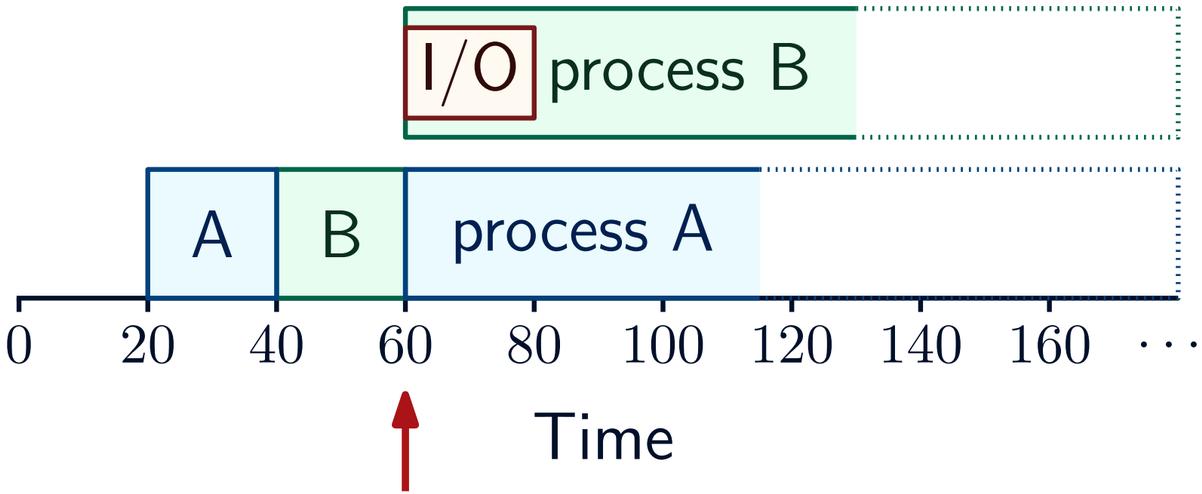


- Ambulance dispatch, air traffic, TikTok queue...

CPU Scheduling

- One machine—the CPU
- Jobs—processes—are only revealed at arrival time
- The length of a job is unknown
- Jobs may be interrupted
- Jobs may need to wait on I/O

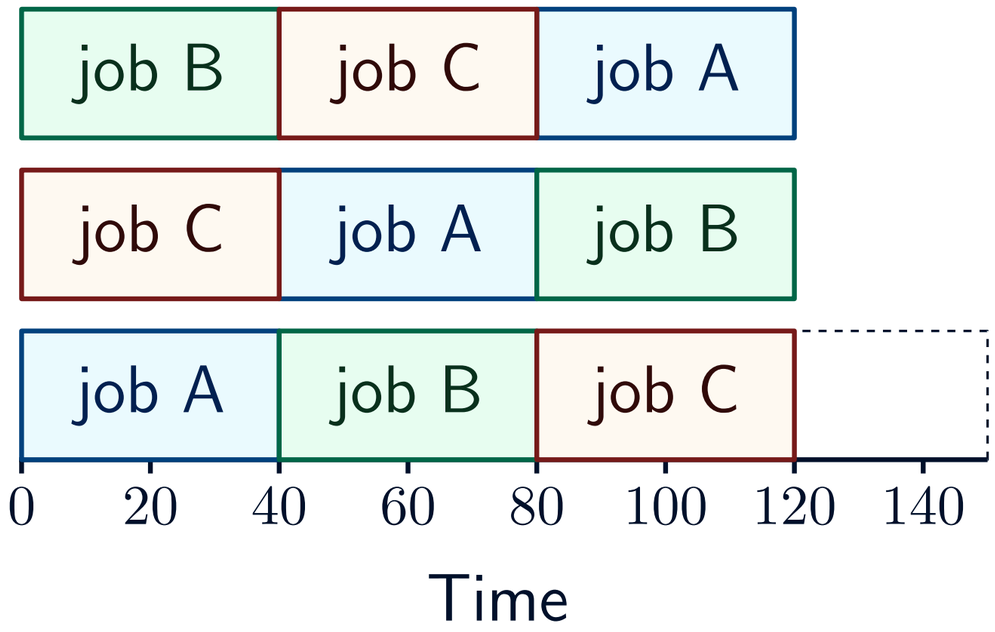
Scheduler should be **fast** and **robust**



Starting simple

- For now, assume that
  1. All jobs arrive at the same time
  2. All jobs have the same length
  3. The jobs cannot be interrupted
  4. The jobs do not wait on I/O
  5. The length of a job is known in advance

At this point, maybe just run in any order?



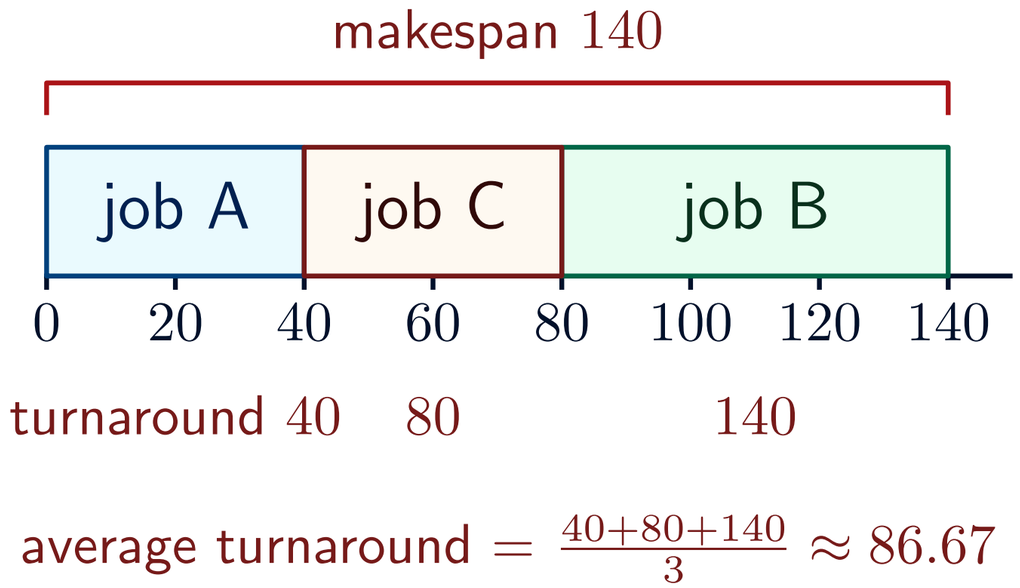
Measuring performance

- Convenient to have a numerical value for the efficiency
- **Attempt:** Let **makespan** be the completion time of the last job
  - Effectively, aiming for less gaps
  - Almost useless for a single machine
- **Better:** Let **turnaround** be the completion time minus arrival time of the job

$$T_{\text{completion}} - T_{\text{arrival}}$$

- We will measure the **average turnaround**

$$\frac{\sum T_{\text{completion}} - T_{\text{arrival}}}{|\text{jobs}|}$$



First In, First Out

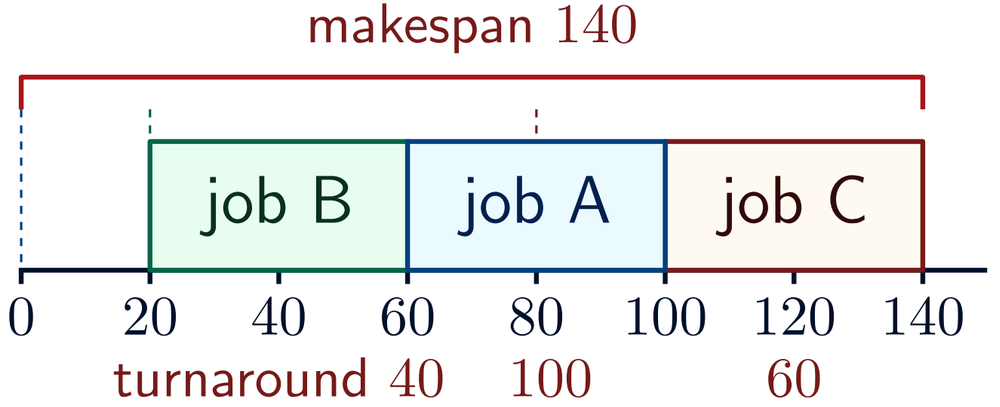
- **Algorithm:** Run the job that arrived the earliest
- **Theorem:** FIFO is optimal for average turnaround when lengths are the same
  no proof in this class

FIFO: makespan 120
av. turnaround ≈ 46.67

Dummy: makespan 140
av. turnaround ≈ 66.67

Assume that

1. ~~All jobs arrive at the same time~~
2. All jobs have the same length
3. The jobs cannot be interrupted
4. The jobs do not wait on I/O
5. The length of a job is known in advance



average turnaround $= \frac{40+100+60}{3} \approx 66.67$

Shortest Job First

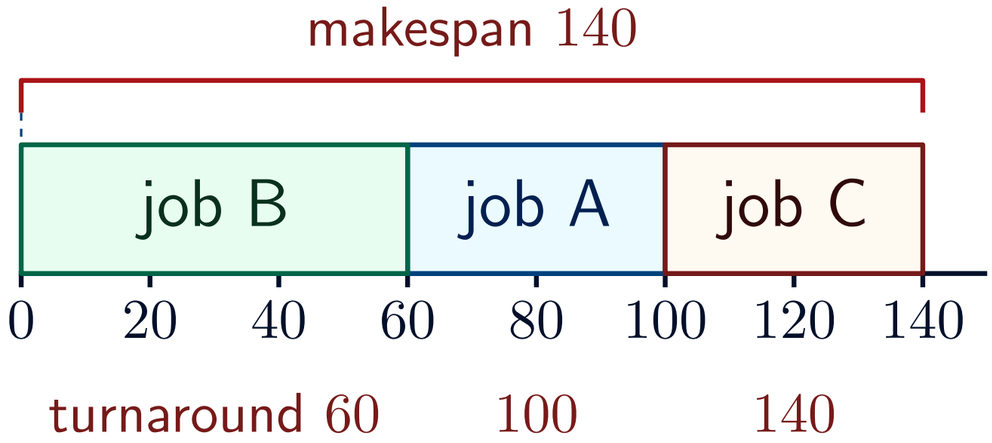
- **Algorithm:** Run the shortest available job
- **Theorem:** SJF is optimal for average turnaround when starting time is the same
  no proof in this class

SJF: makespan 140
av. turnaround ≈ 86.67

FIFO: makespan 140
av. turnaround $= 100$

Assume that

1. All jobs arrive at the same time
2. ~~All jobs have the same length~~
3. The jobs cannot be interrupted
4. The jobs do not wait on I/O
5. The length of a job is known in advance



average turnaround $= \frac{60+100+140}{3} = 100$

SJF needs interrupts

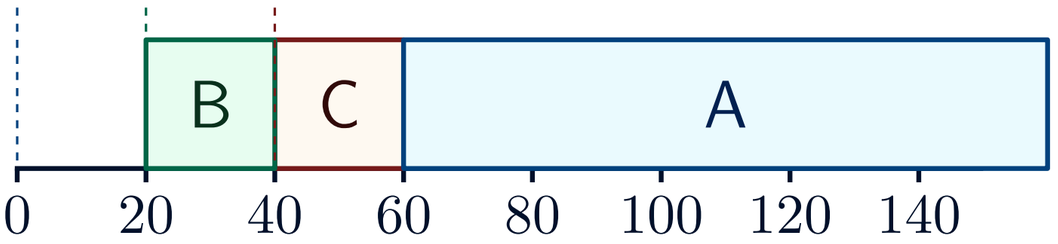
- **Problem:** SJF does not use arrival times

SJF:
av. turnaround $= 100$

Optimal:
av. turnaround ≈ 66.67

Assume that

1. ~~All jobs arrive at the same time~~
2. ~~All jobs have the same length~~
3. The jobs cannot be interrupted
4. The jobs do not wait on I/O
5. The length of a job is known in advance



turnaround 20 20 160

average turnaround $= \frac{20+20+160}{3} \approx 66.67$

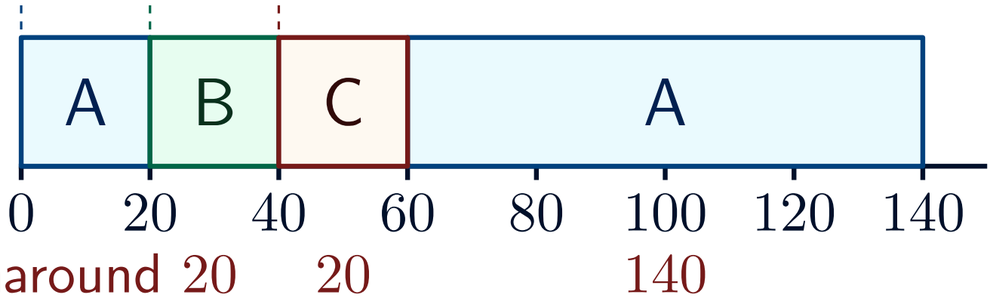
Shortest to Completion First

- **Algorithm:** Run the job with the least time **left**
- No need to wait for completion thanks to interrupts and context switches

Optimal w/o interrupts:
av. turnaround ≈ 66.67

STCF:
av. turnaround $= 60$

1. ~~All jobs arrive at the same time~~
2. ~~All jobs have the same length~~
3. ~~The jobs cannot be interrupted~~
4. The jobs do not wait on I/O
5. The length of a job is known in advance



turnaround $20 \quad 20 \quad 140$

average turnaround $= \frac{20+20+140}{3} = 60$

Response time

batch processing

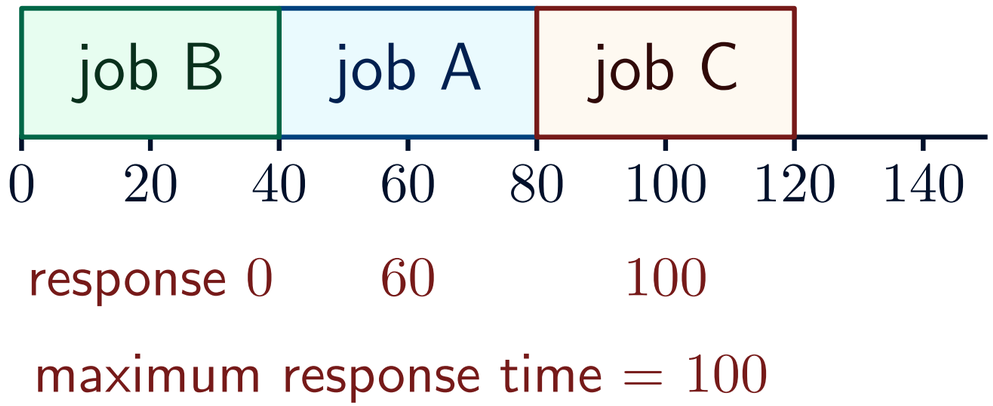
- STCF would be perfect for the early-day OS
- Nowadays, we do not only care about performance, but also interactivity
- **Response time** is the time the job has to wait:

  $$T_{\text{firstrun}} - T_{\text{arrival}}$$

- We would like any job to have low response time

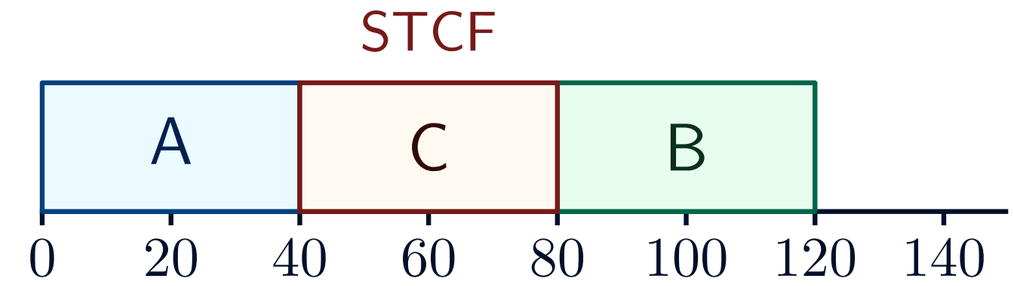
  $$T_{\text{firstrun}} - T_{\text{arrival}}$$

- STCF is not great for response time:



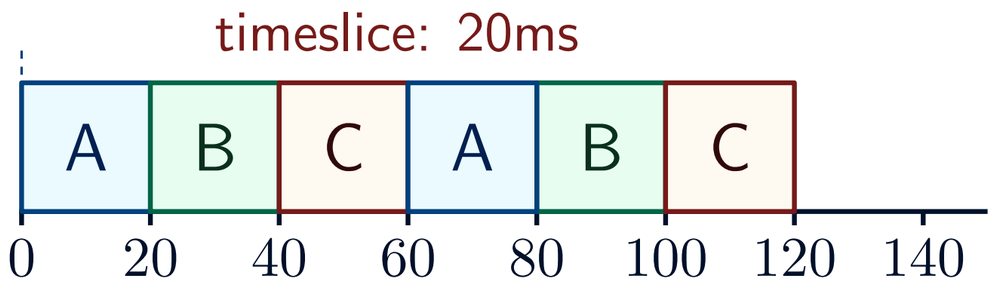
Round Robin

- **Algorithm:** Run any job for a short timeslice, then put it in the back of the queue
- Need to balance the length of the slice
  - Too long—bad response time
  - Too short—context switching dominates
- Round Robin is not great for turnaround time



turnaround $40 \quad 80 \quad 120$

average turnaround $= \frac{40+80+120}{3} = 80$

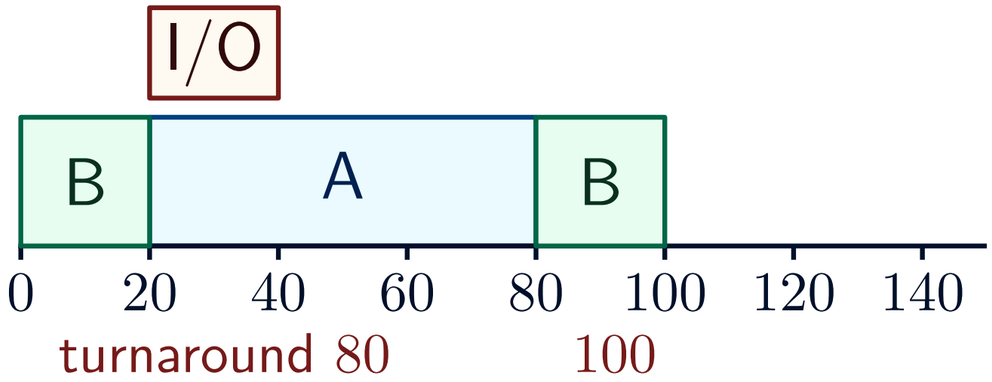


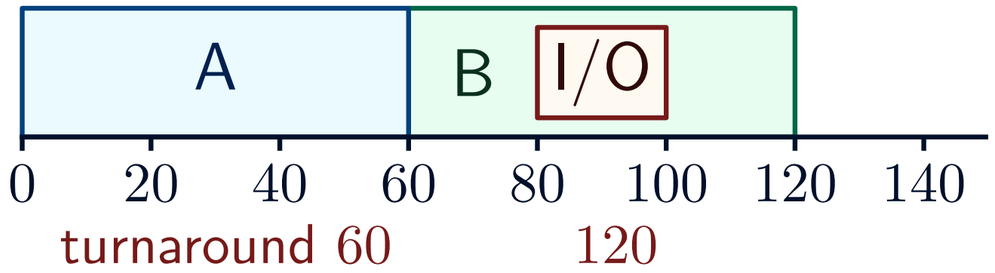
turnaround $80 \quad 100 \quad 120$

average turnaround $= 100$

Incorporating I/O

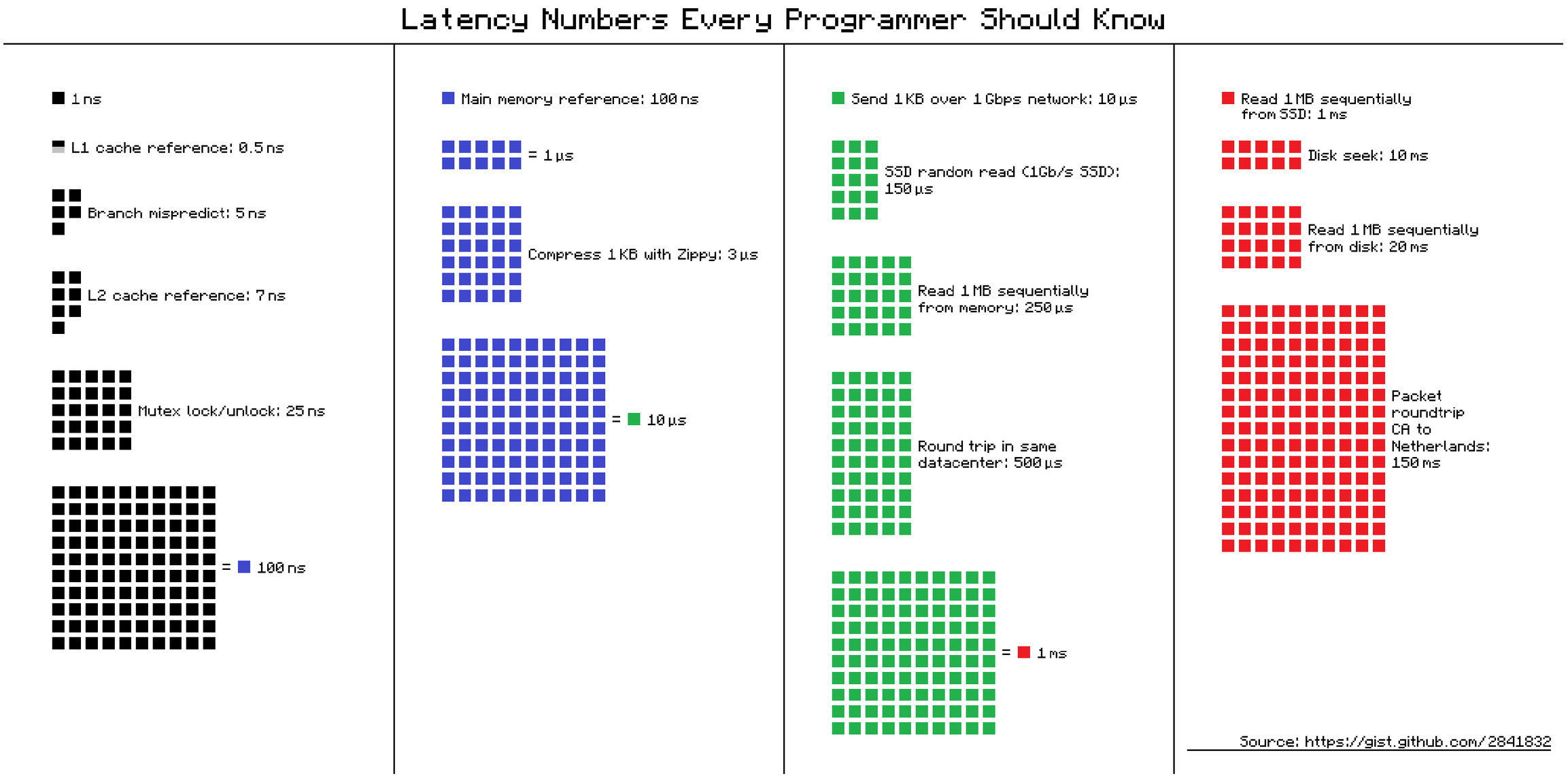
- While a job waits for I/O, run something else
- Putting jobs with I/O first is fine with turnaround time
- Jobs that do I/O a lot tend to use CPU less





same turnaround time, better response time

Why wait on I/O



So far

- Two measures of success: performance and interaction
- Both need to be good for an all-purpose OS
- Knowing running times in advance is a very strong assumption
- Running low-compute I/O-focused jobs first is a good idea

1. ~~All jobs arrive at the same time~~
2. ~~All jobs have the same length~~
3. ~~The jobs cannot be interrupted~~
4. ~~The jobs do not wait on I/O~~
5. The length of a job is known in advance

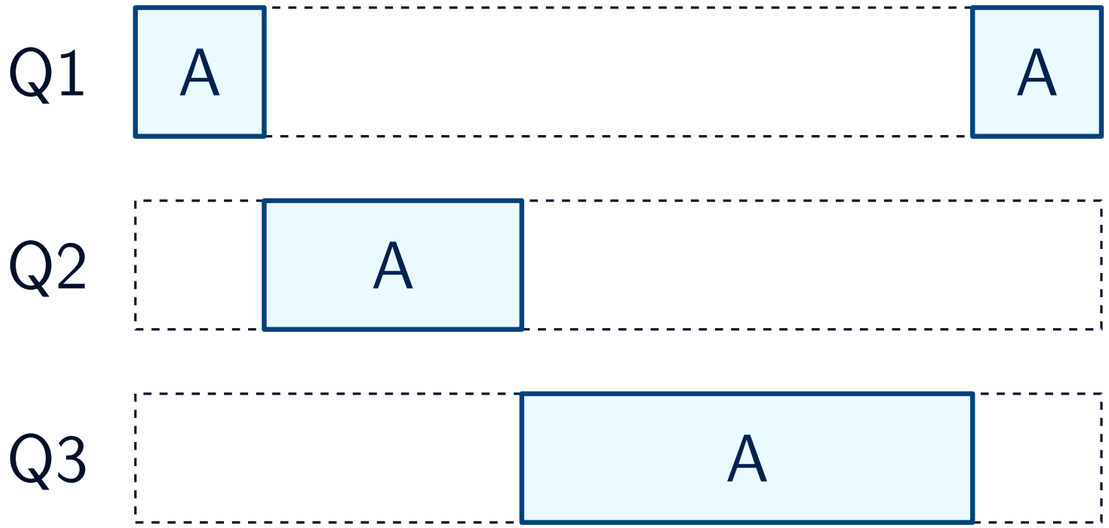
Multi-Level Feedback Queue

- Proposed in 1962 by Corbato et al.
- Good turnaround time while also good response time
- "Learns" from history
- Baseline approach for in Windows, FreeBSD, Solaris

~~1. All jobs arrive at the same time~~
~~2. All jobs have the same length~~
~~3. The jobs cannot be interrupted~~
~~4. The jobs do not wait on I/O~~
~~5. The length of a job is known in advance~~

The Queues in MLFQ

- There are several queues, modelling different priorities
- Run the processes from the highest non-empty queue in RR
- When a process comes, put it in the topmost queue
- If a process uses up its allotment, move it down
- Priority boost: after time S, move all jobs to topmost queue



Before you go

- More details from today: Chapters 7, 8 in the OSTEP book
  - Try out the simulators there

- Mandatory Assignment 1
  - open Friday **September 12**
  - deadline Friday **September 26**
  - description and delivery on Mitt
  - you will do a few experiments in C

- Friday: more on MLFQ

- Friday: starting from fairness in scheduling
  Linux and its Completely Fair Scheduler (CFS)