Operating Systems

Scheduling

Lecture Overview

- Scheduling

Last Week

Virtualisation

- Each process thinks it has its own computer
- Direct execution is faster
- Limited execution at key points to ensure OS control
- Hardware provides a lot of OS functionality
  - User vs Kernel
  - Clock
  - Register Saving

Scheduling

Who, when?

Scheduler Summary

- First Come, First Serve (FCFS)
- Shortest Job First
- Shortest Time to Completion First (STCF)
- Round Robin (RR)
- Multilevel Feedback Queue (MLFQ)

Scheduling vs Dispatching

Dispatcher

- Determines how to switch between processes

- User mode/kernel mode
- Saving PCBs…

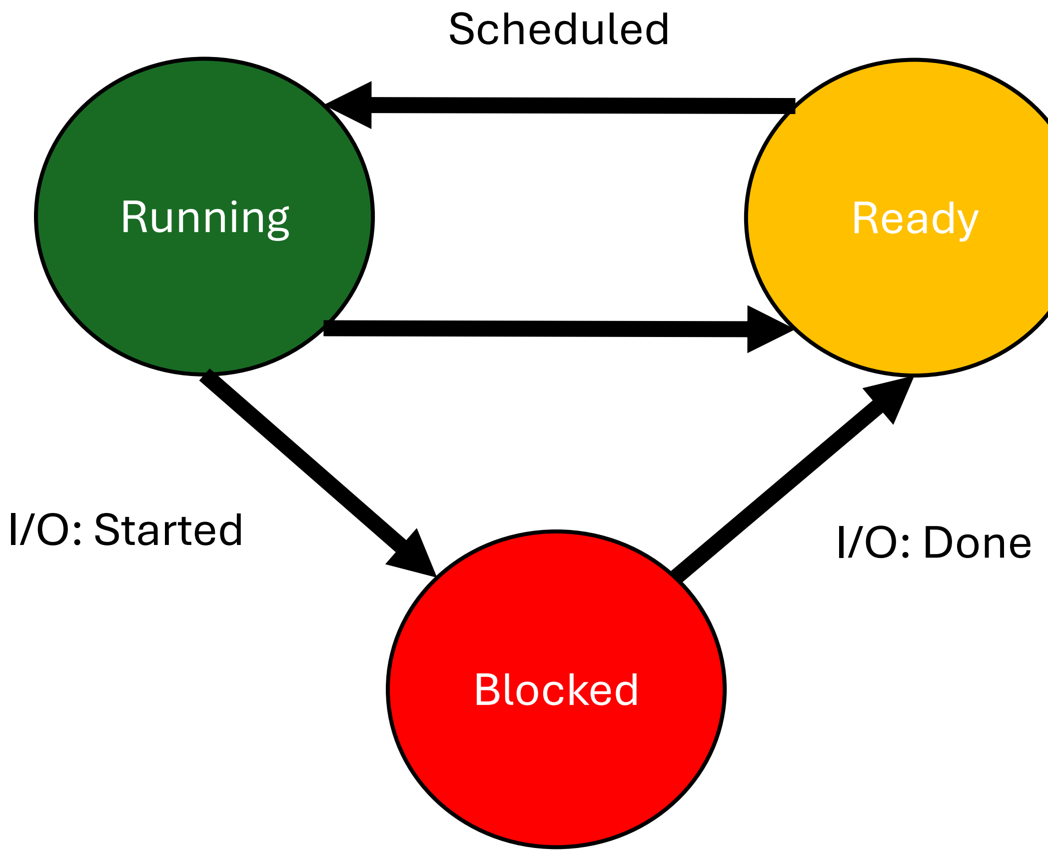
- Mechanical (**How**)

Scheduler

- Determines which process to dispatch

- Conceptual (**When**)

Scheduling and States



Definitions

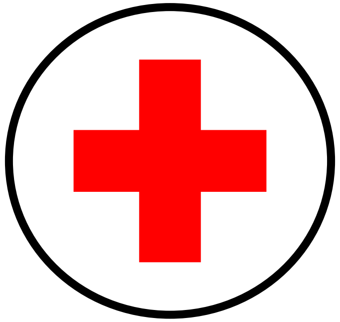
- **Workload:** The ‘**jobs**’ that need to be done
- **Job:** View as current CPU burst of a process
  - Process sometimes alternates between CPU & I/O
  - Process moves between ready and blocked queues
- **Scheduler:** Logic for the job choice
- **Metric:** Something to determine **good** from **bad**

Good Scheduling vs Bad Scheduling

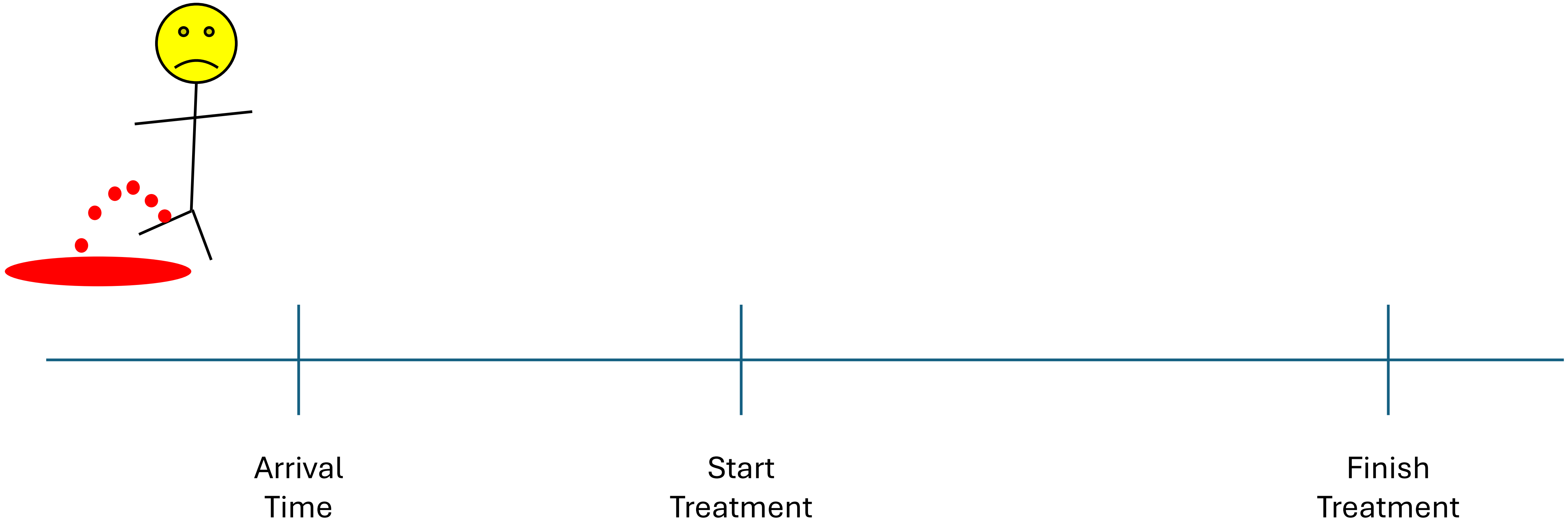
Activity:

- Imagine you are running a hospital and you have a *operating room* (for surgery).
- You have a list of patients who need various operations…

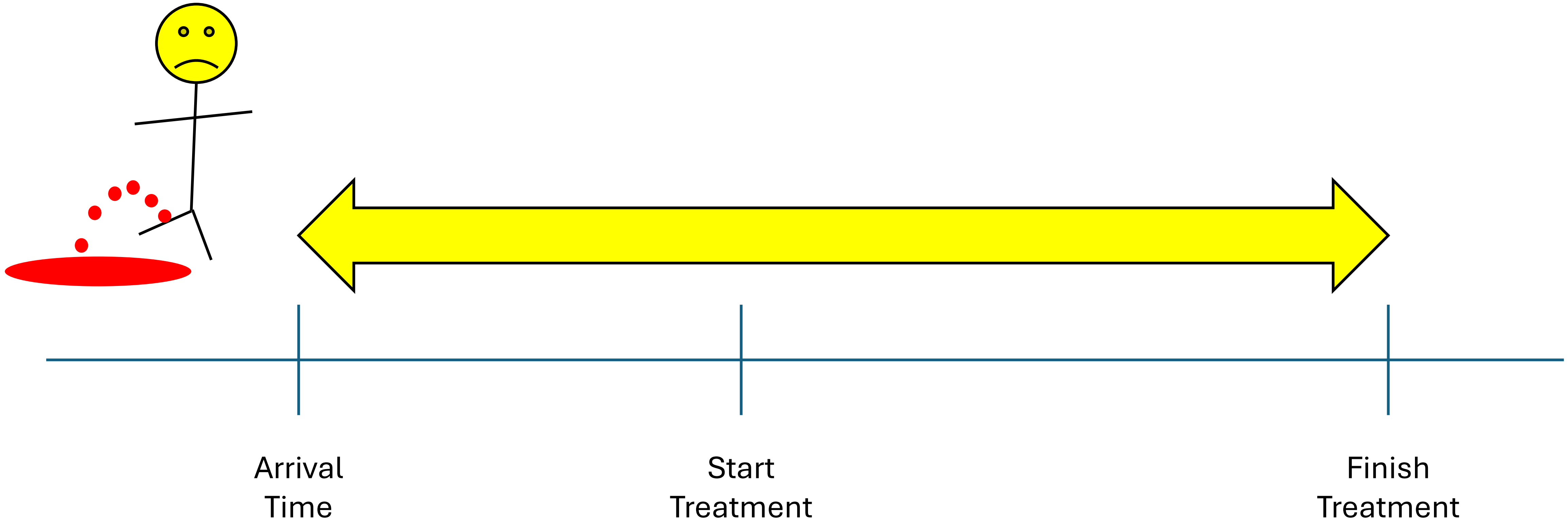
- How do we determine the best way to use the *operating room*?
- What things do we want to **maximise** or **minimise**?



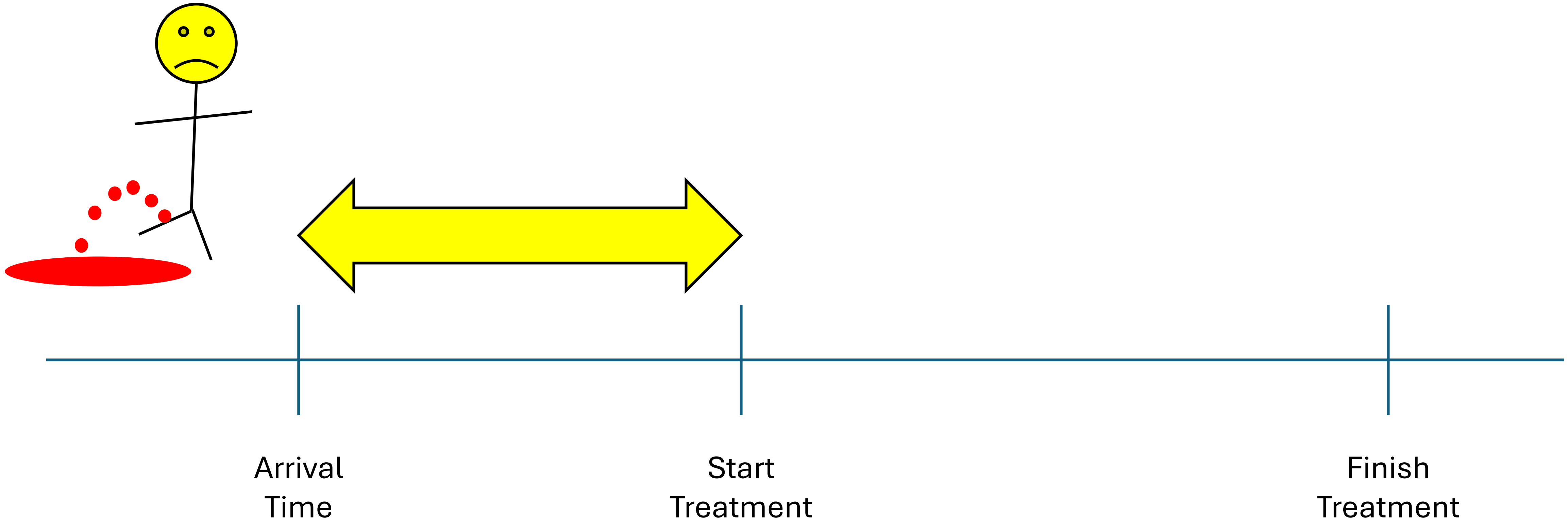
Some Answers



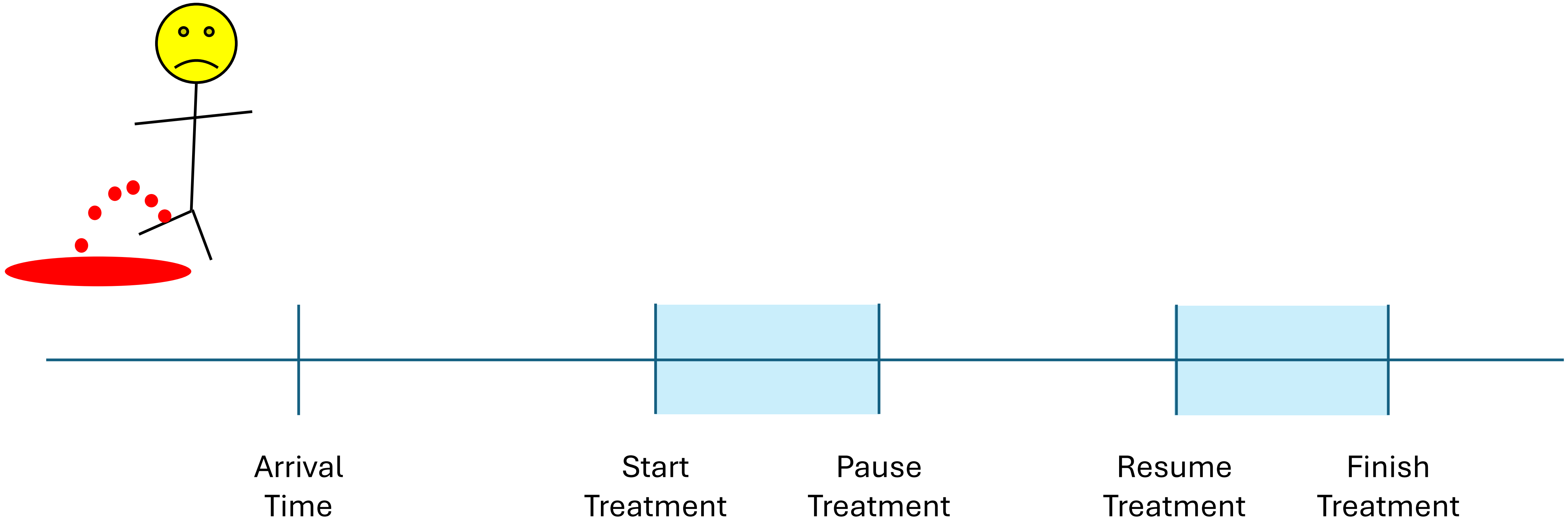
Some Answers: **Turnaround Time**



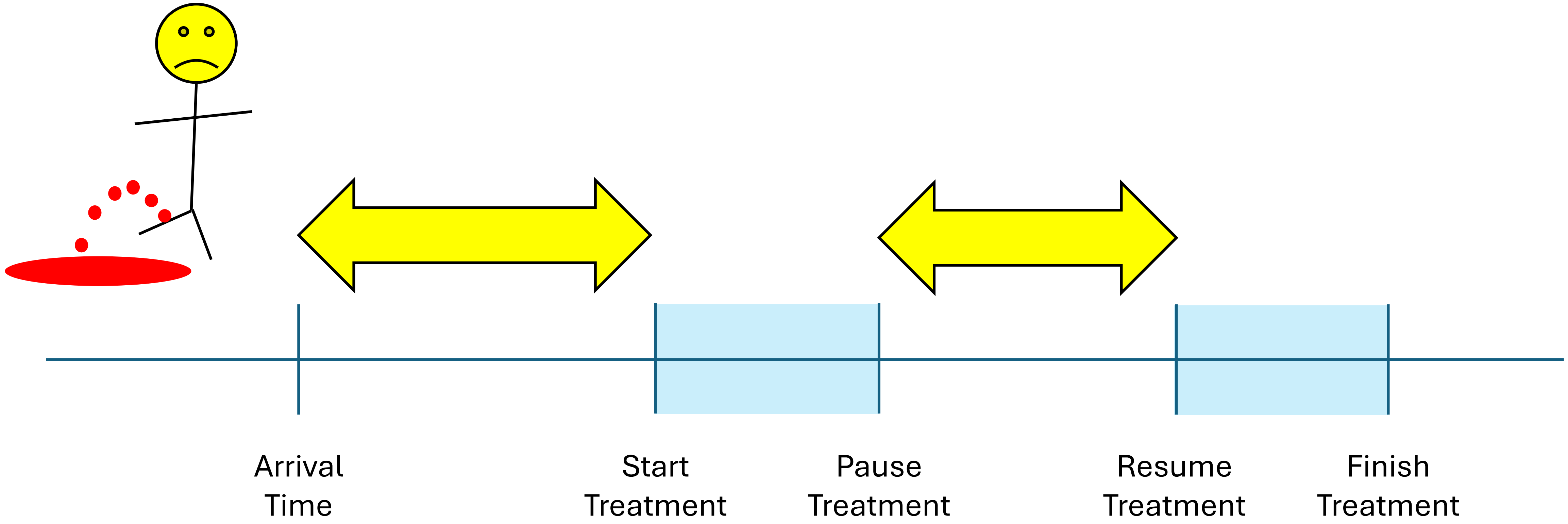
Some Answers: **Response Time**



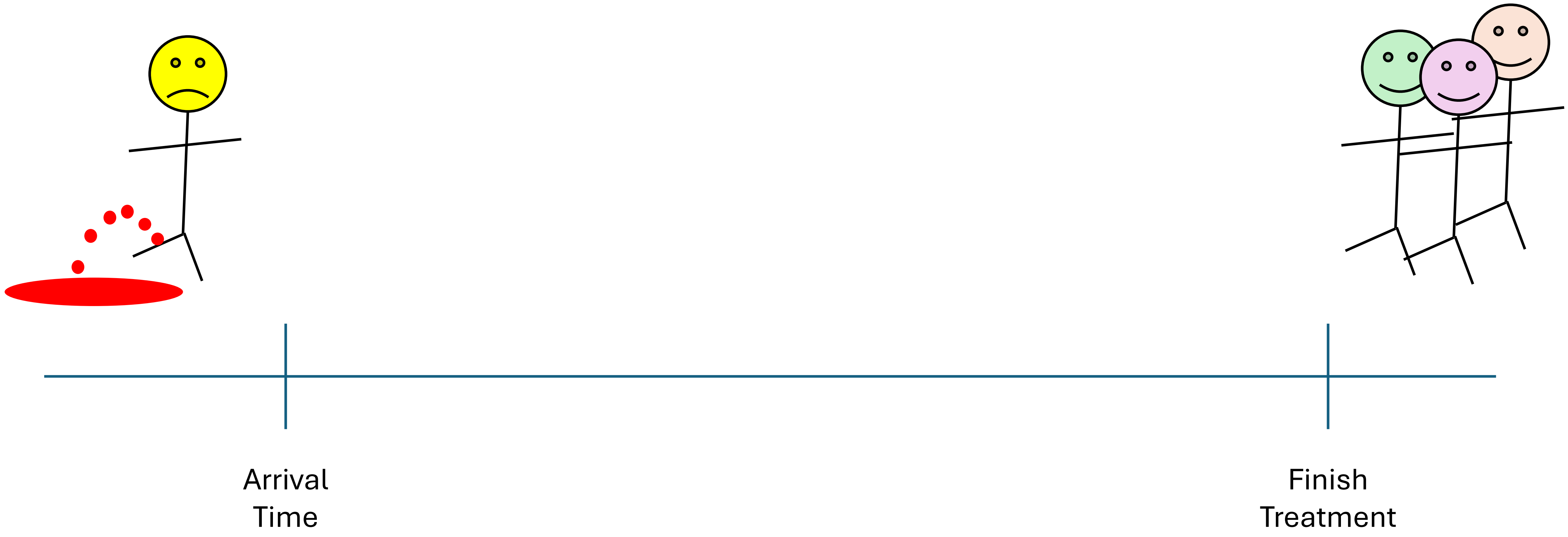
Some Answers



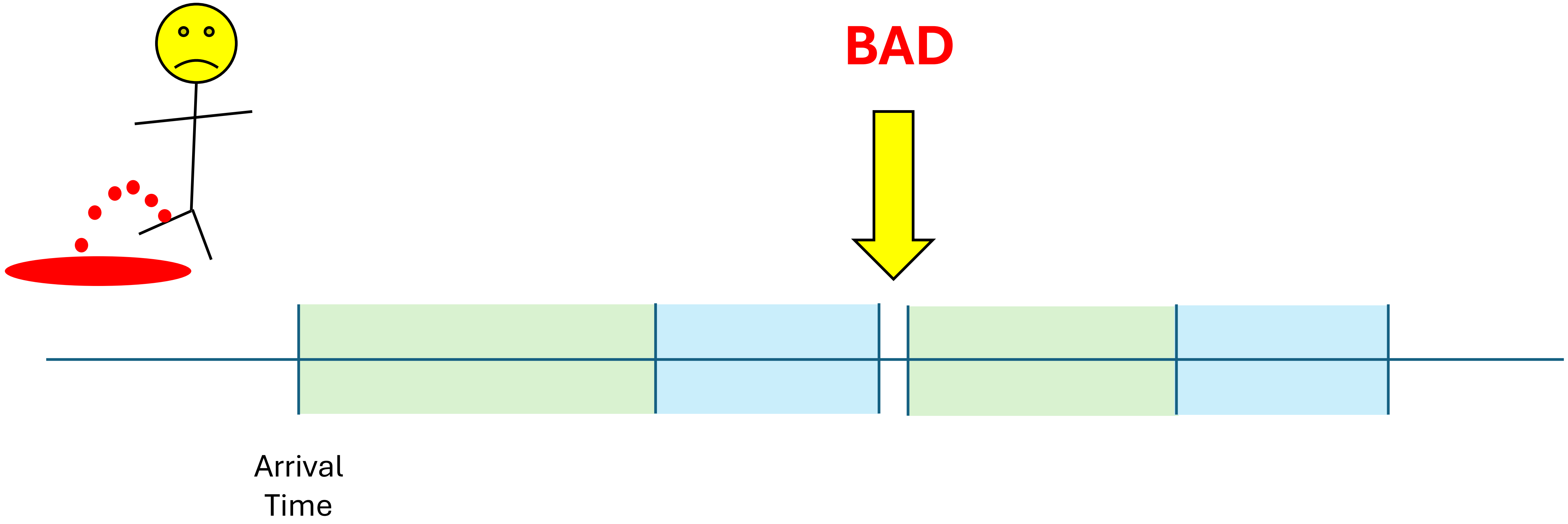
Some Answers: **Wait Time**



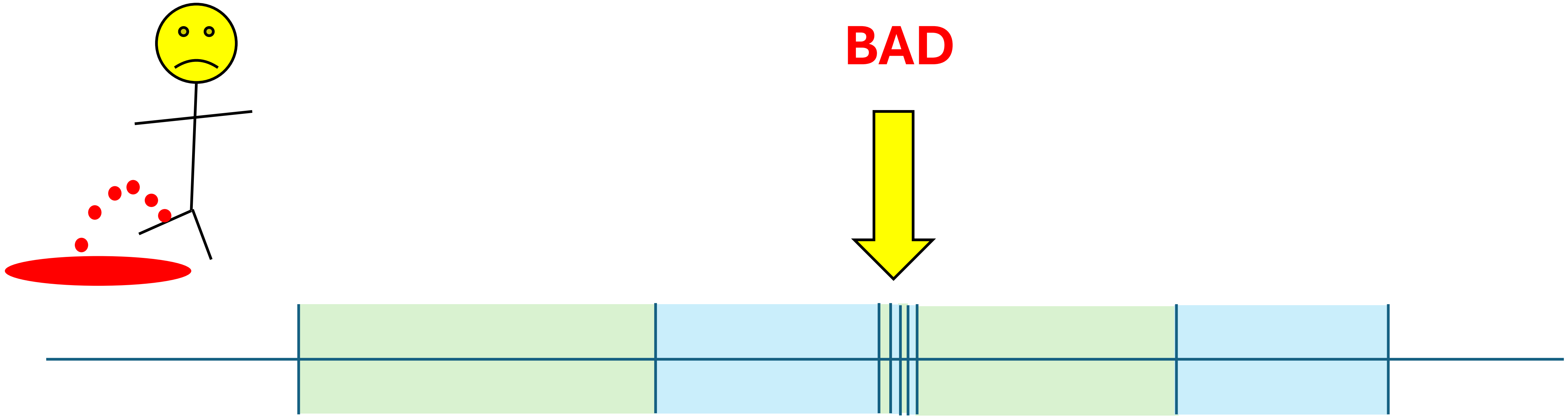
Some Answers: **Throughput**



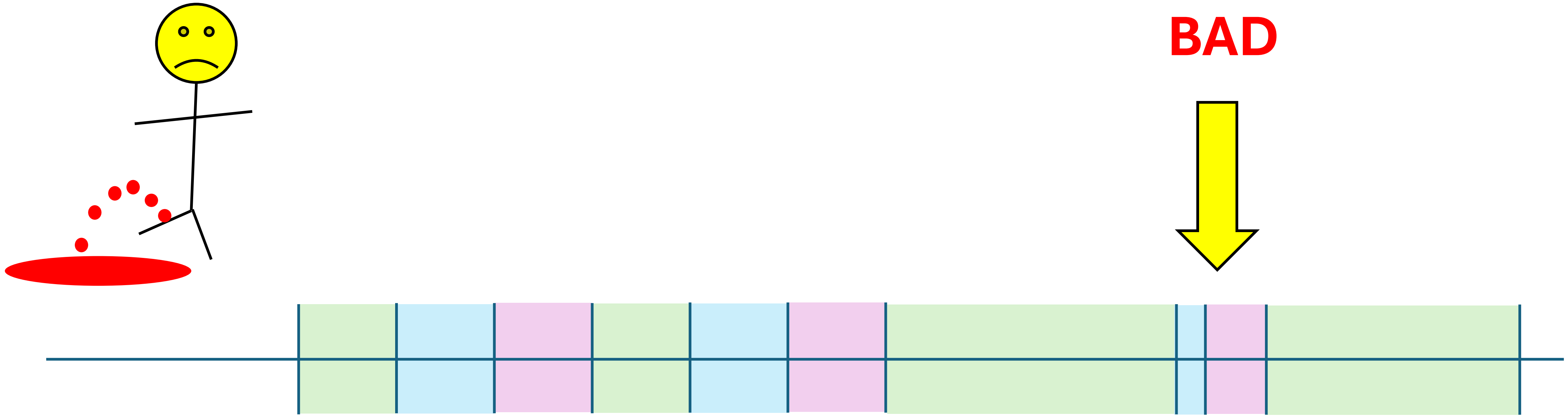
Some Answers: **Resource Utilisation**



Some Answers: **Overhead (Switches)**



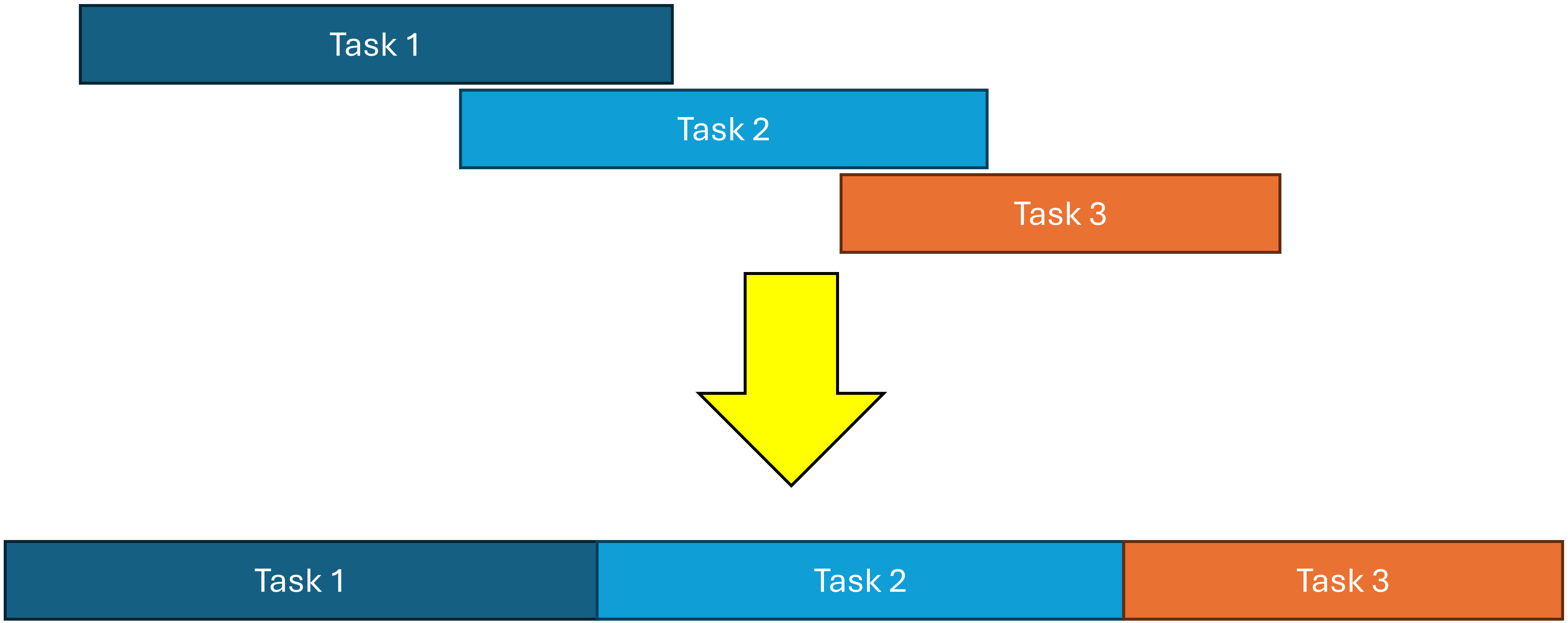
Some Answers: **Fairness**



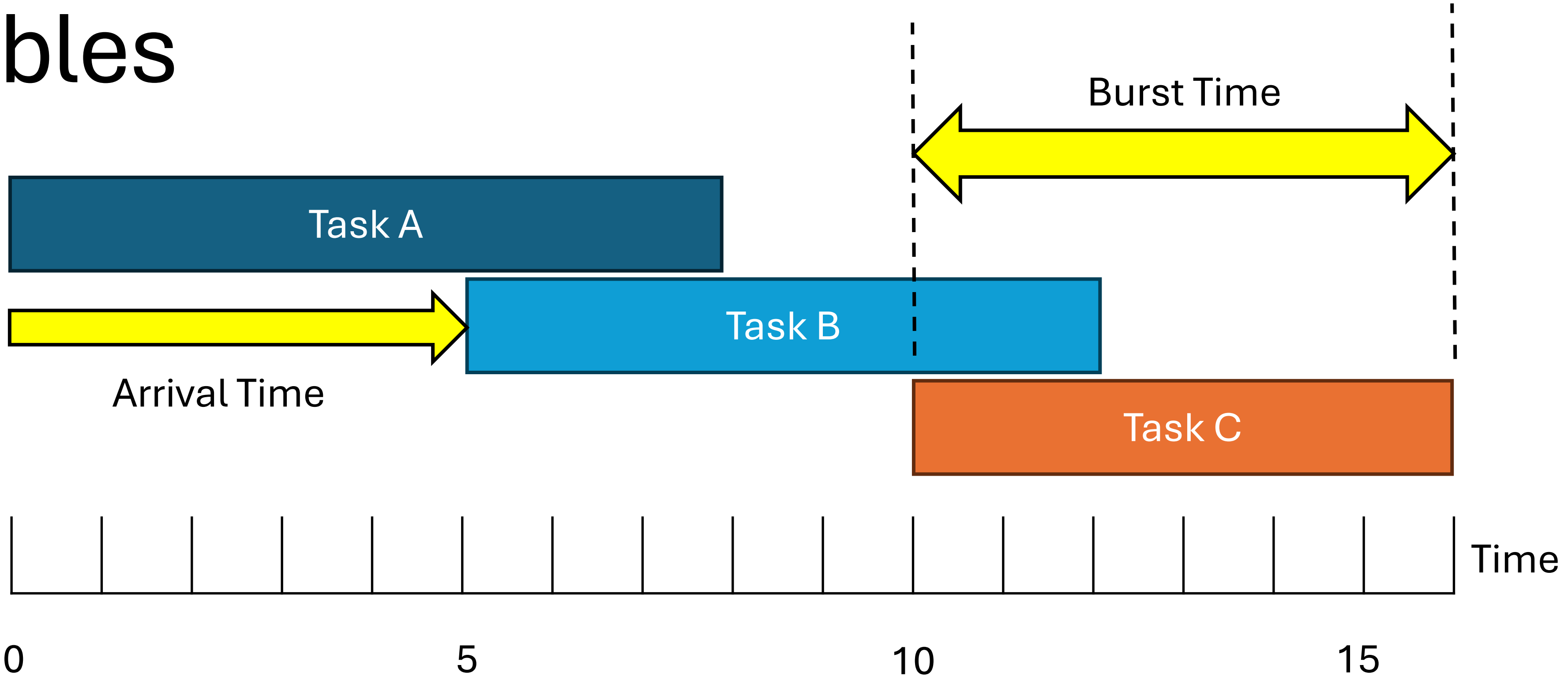
Scheduling Algorithms

Start with the basics…

GANTT Charts



Tables

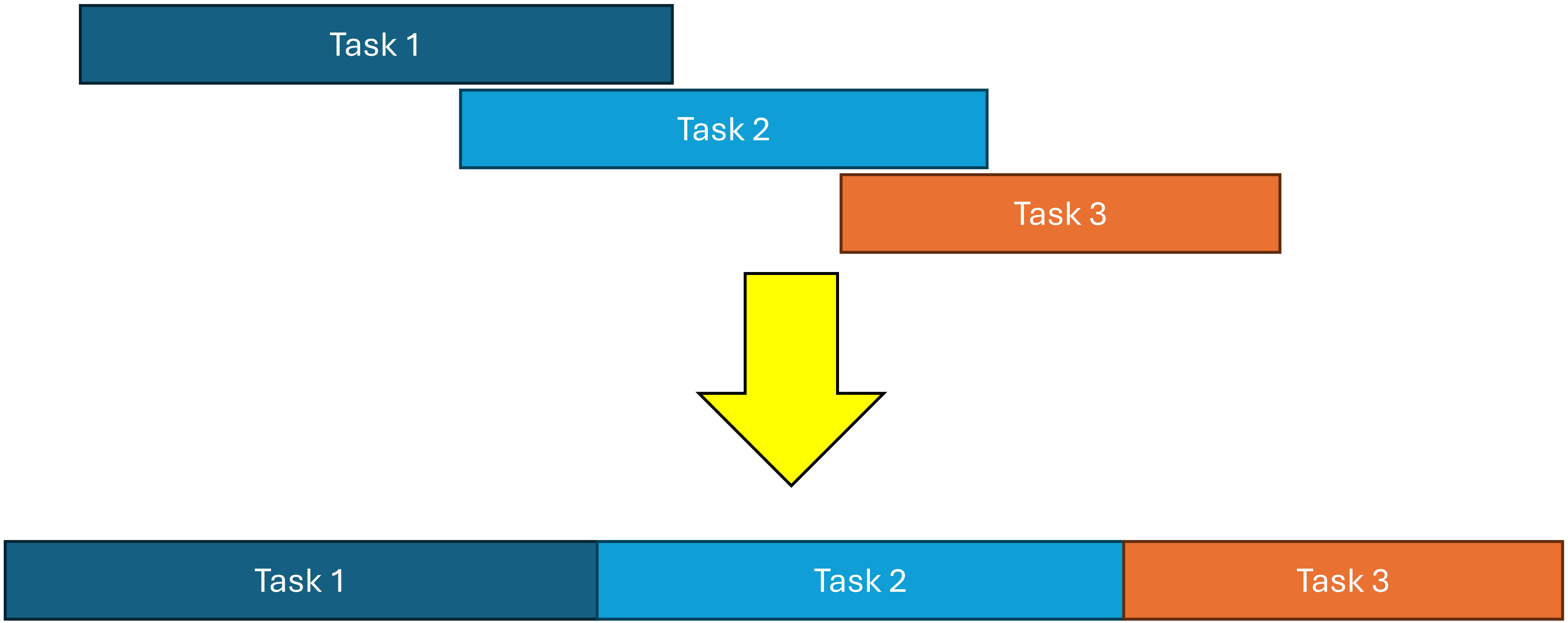


Job	Arrival Time	Burst Time
A	0	8
B	5	7
C	10	6

Scheduling Assumptions

1. Jobs run for a fixed time
2. All jobs arrive simultaneously
3. All jobs are CPU-only
4. Run-time of each job is known

First Come, First Serve (FCFS)

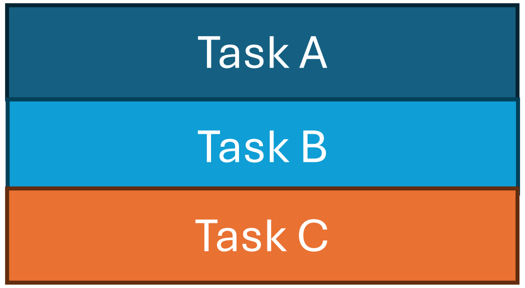


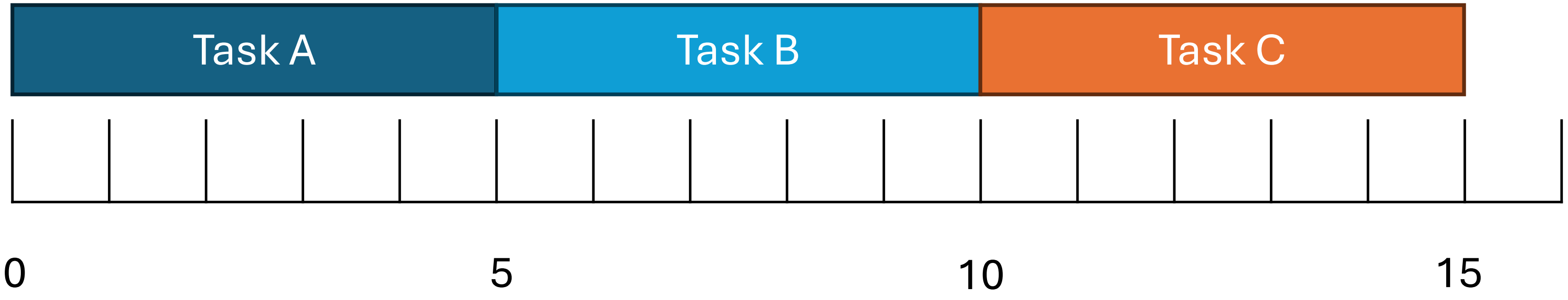
First Come, First Serve (FCFS)

Metrics

- Turnaround Time completion_time – arrival_time
- Response Time

Job	Arrival Time	Burst Time
A	0	5
B	0	5
C	0	5



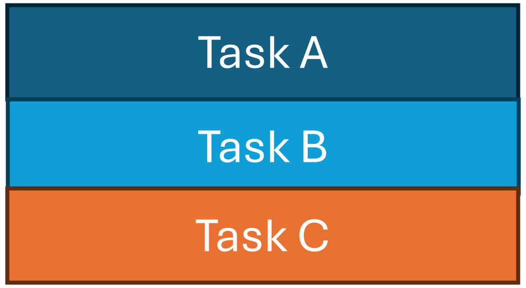


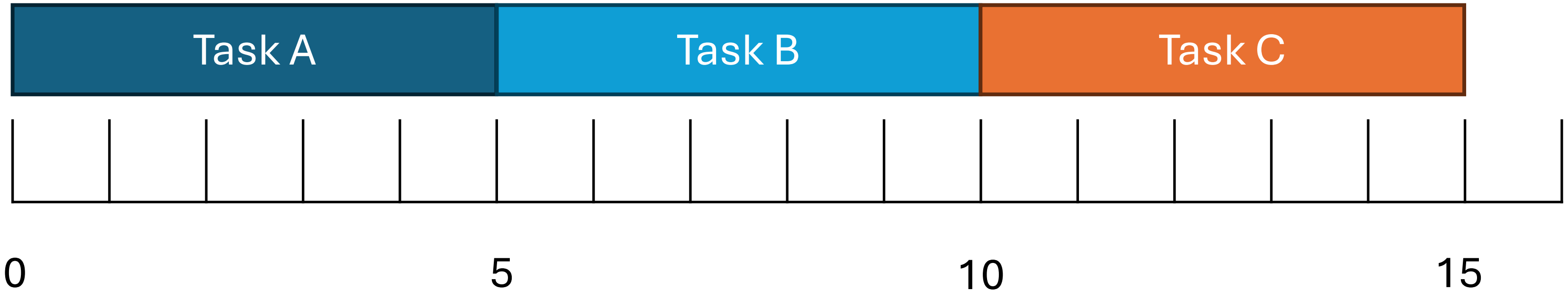
First Come, First Serve (FCFS)

Metrics

- Turnaround Time completion_time – arrival_time
- Response Time

Job	Arrival Time	Burst Time	Turnaround Time
A	0	5	5
B	0	5	10
C	0	5	15

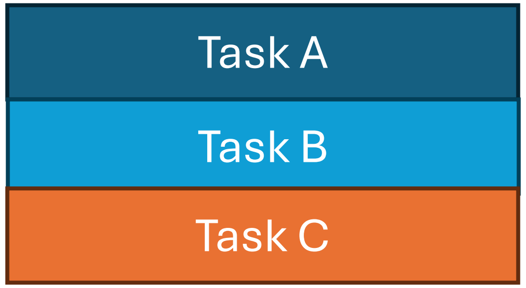




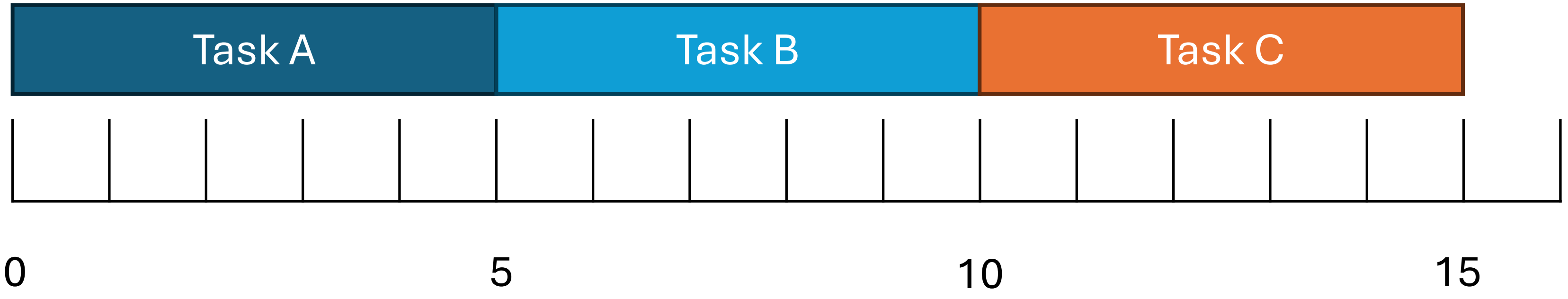
First Come, First Serve (FCFS)

Metrics

- Turnaround Time   completion_time – arrival_time
- Response Time



Job	Arrival Time	Burst Time	Turnaround Time
A	0	5	5
B	0	5	10
C	0	5	15



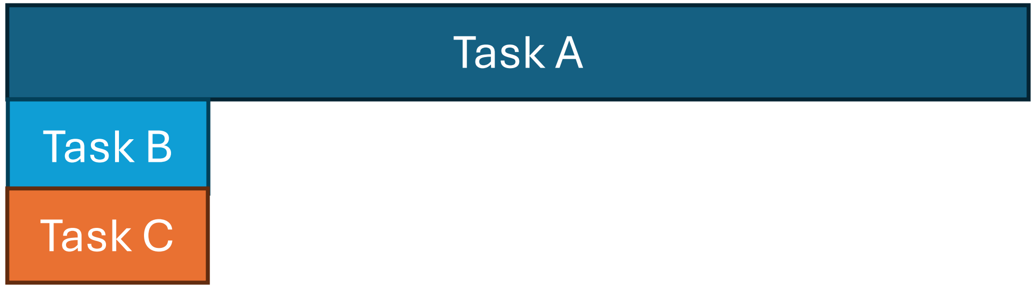
$$\frac{5 + 10 + 15}{3} = 10$$

Average

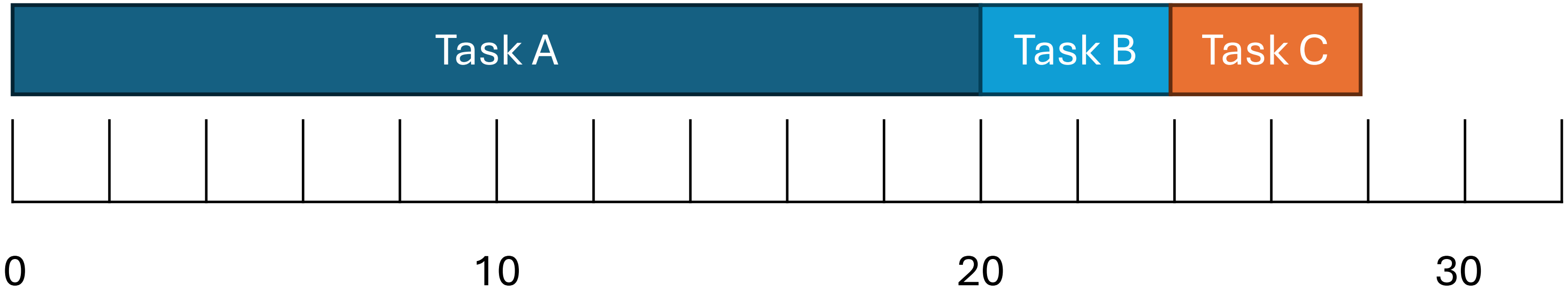
First Come, First Serve (FCFS)

Metrics

- Turnaround Time   completion_time – arrival_time
- Response Time



Job	Arrival Time	Burst Time	Turnaround Time
A	0	20	20
B	0	4	24
C	0	4	28



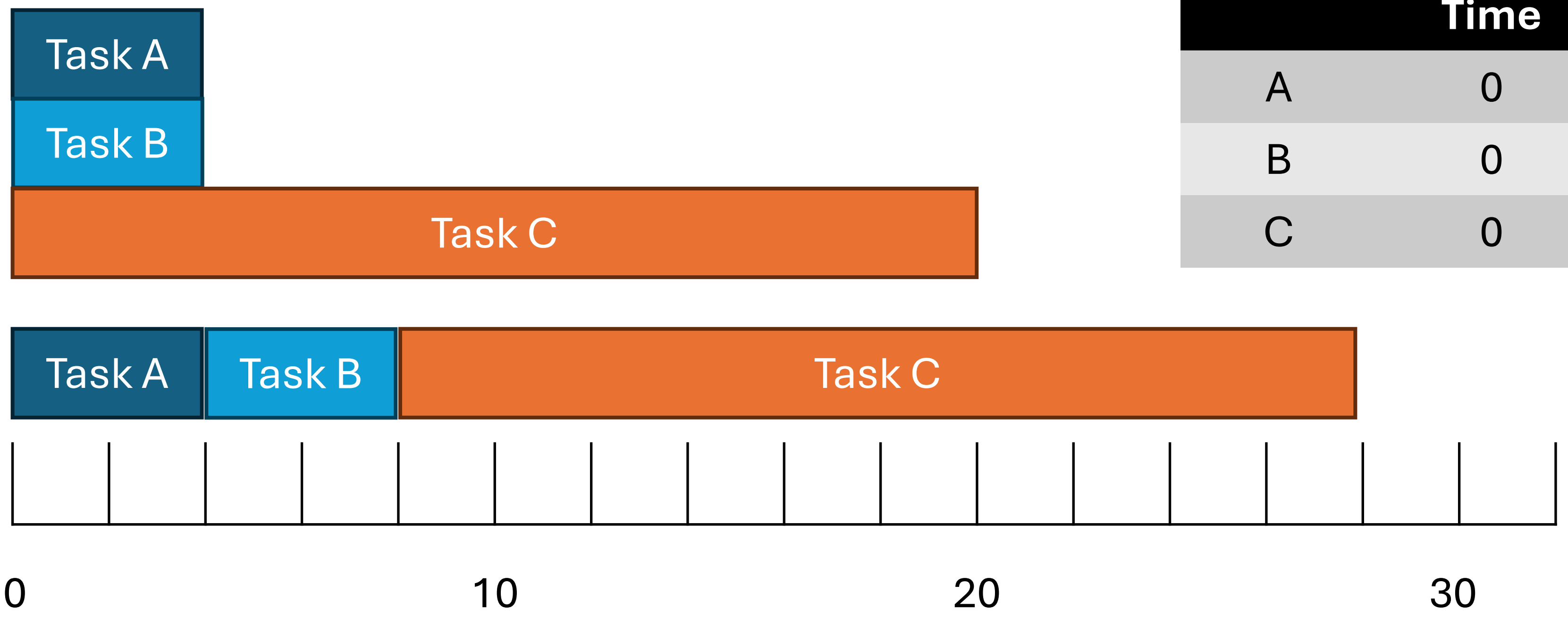
$$\frac{20 + 24 + 28}{3} = 24$$

Average

First Come, First Serve (FCFS)

Metrics

- Turnaround Time completion_time – arrival_time
- Response Time



Job	Arrival Time	Burst Time	Turnaround Time
A	0	4	4
B	0	4	8
C	0	20	28

$$\frac{4 + 8 + 28}{3} = 16.6$$

Average

First Come, First Serve (FCFS)

Advantages

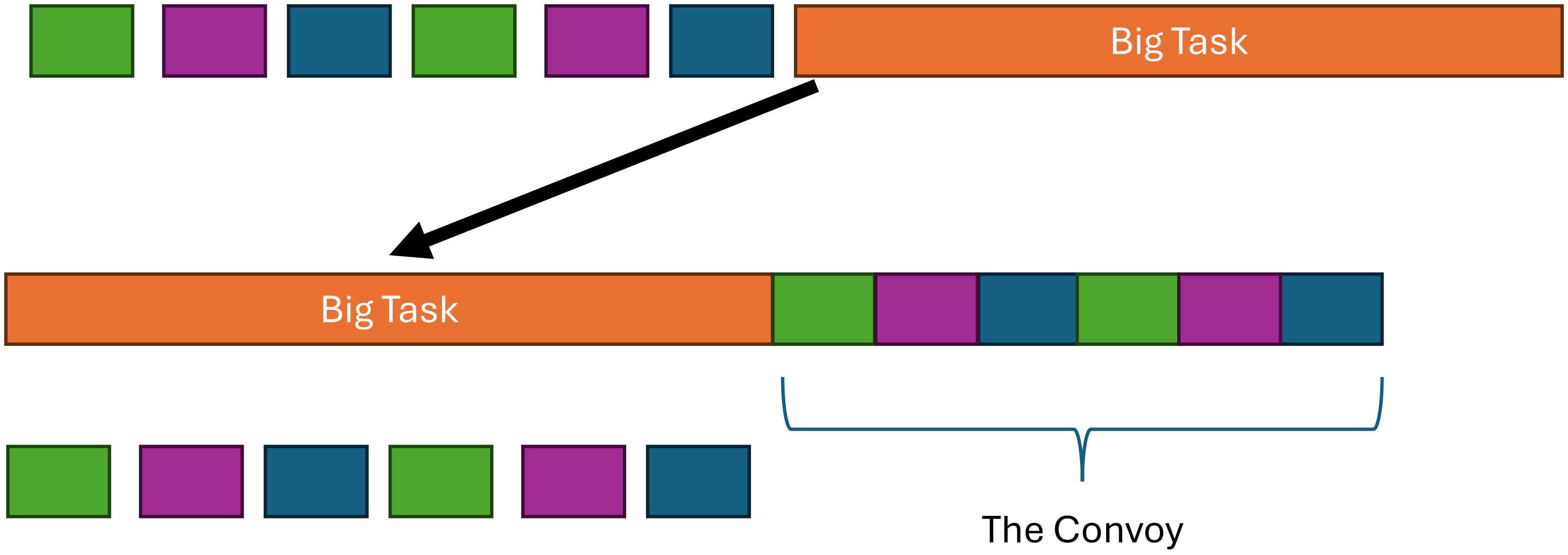
- Very simple to implement
- Has a ‘veneer’ of fairness

Problems

- Dependent on job order
- Big tasks can ‘starve’ small tasks

The Convoy Effect

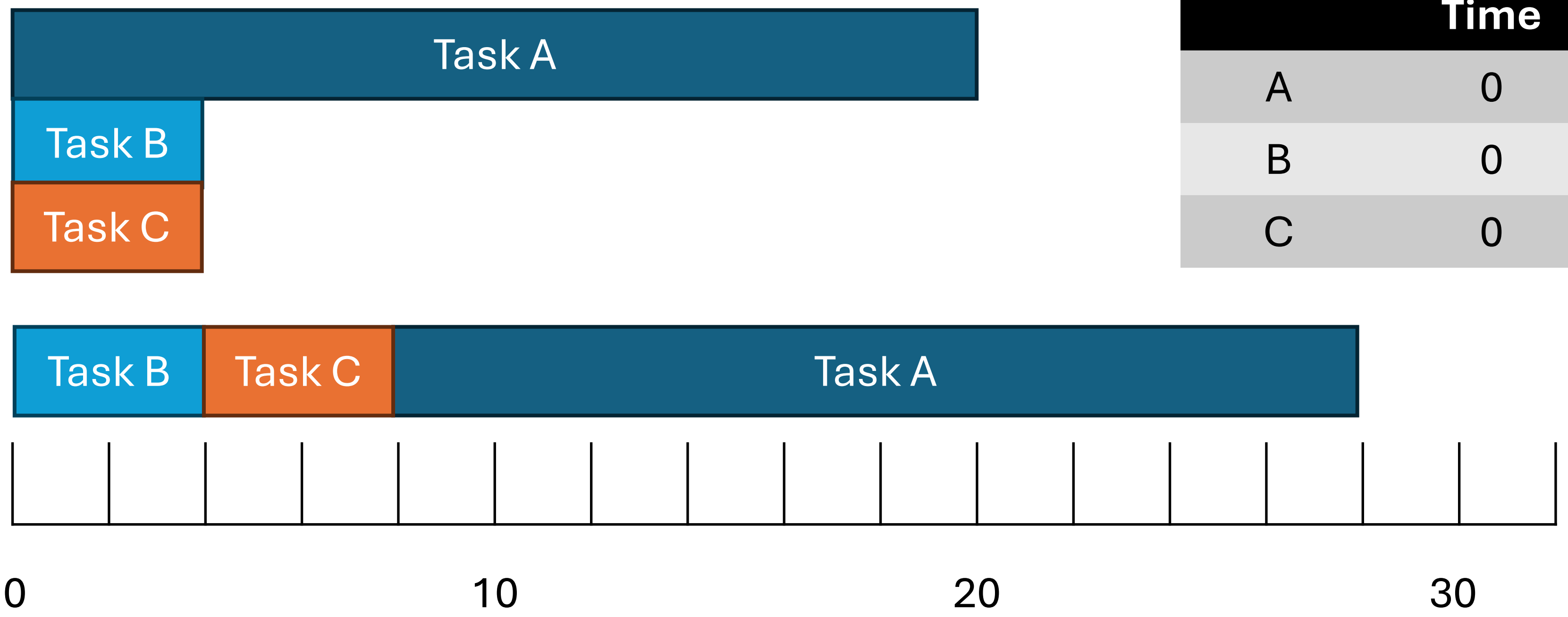
The Convoy Effect



Shortest Job First (SJF)

Metrics

- Turnaround Time completion_time – arrival_time
- Response Time



Job	Arrival Time	Burst Time	Turnaround Time
A	0	20	20
B	0	4	24
C	0	4	28

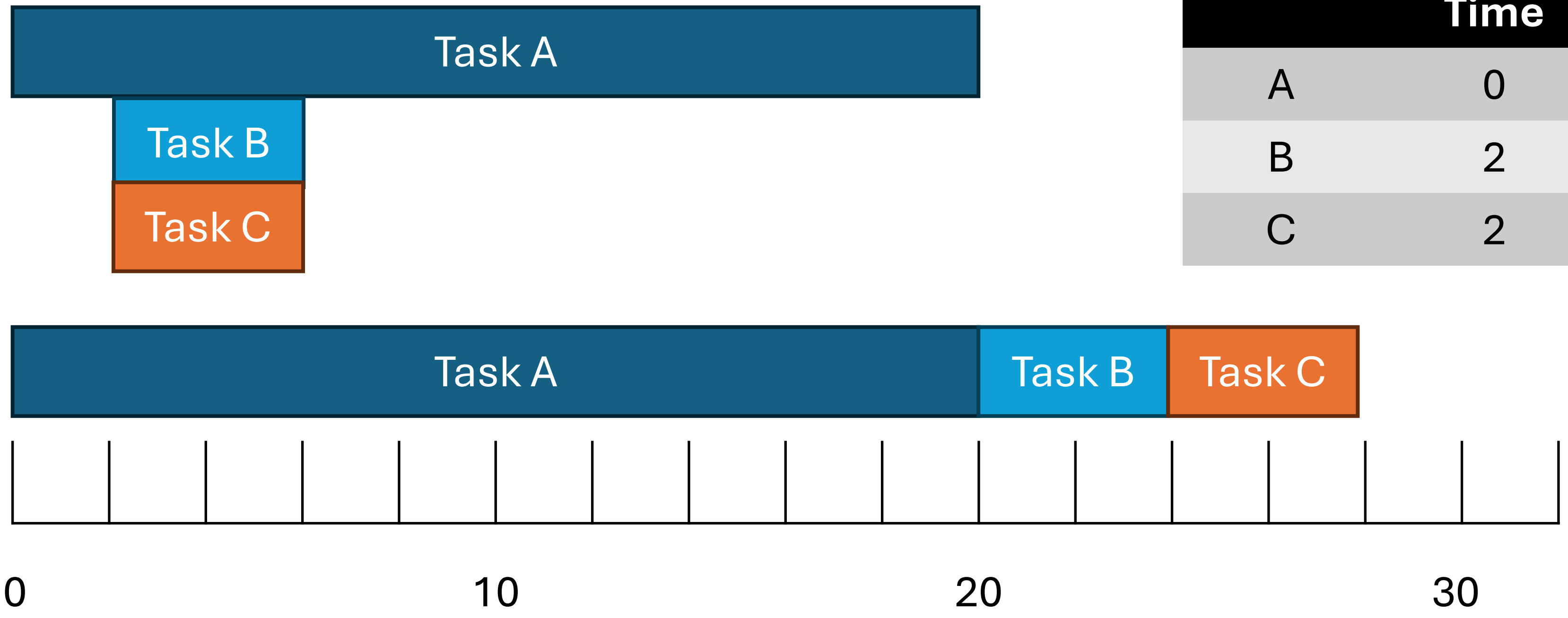
$$\frac{4 + 8 + 28}{3} = 16.6$$

Average

Shortest Job First (SJF)

Metrics

- Turnaround Time completion_time – arrival_time
- Response Time



Job	Arrival Time	Burst Time	Turnaround Time
A	0	20	20
B	2	4	22
C	2	4	26

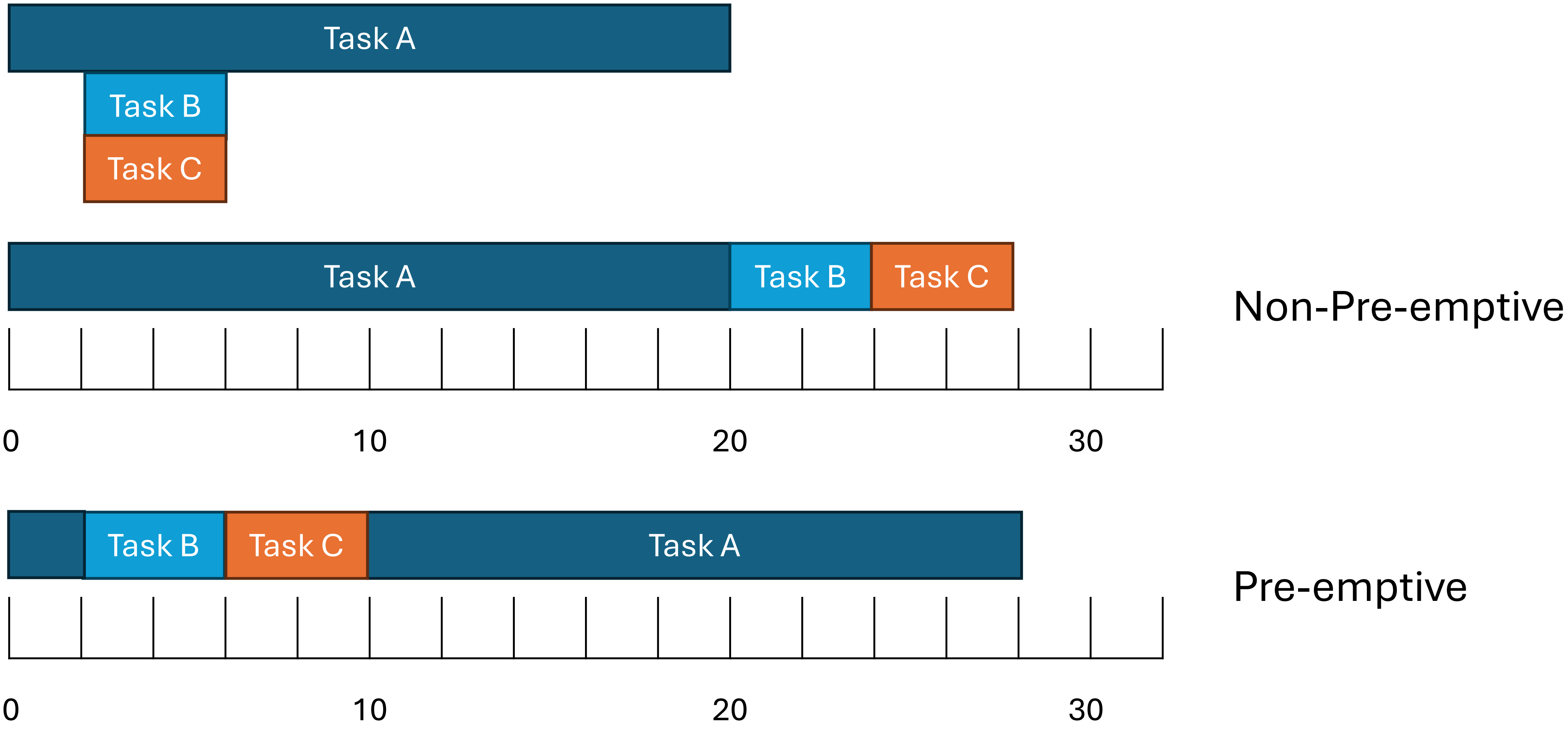
$$\frac{20 + 22 + 26}{3} = 22.6$$

Average

Scheduling Assumptions

1. ~~Jobs run for a fixed time~~
2. ~~All jobs arrive simultaneously~~
3. All jobs are CPU-only
4. Run-time of each job is known

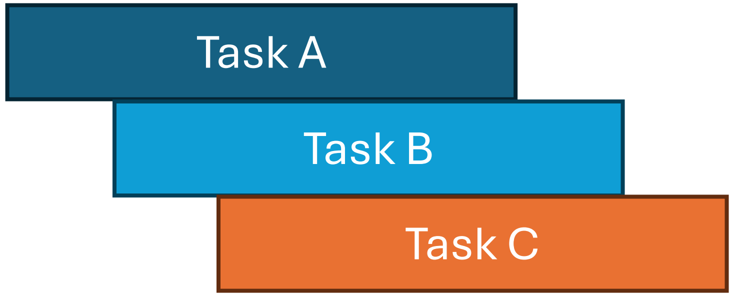
Pre-emptive Scheduling



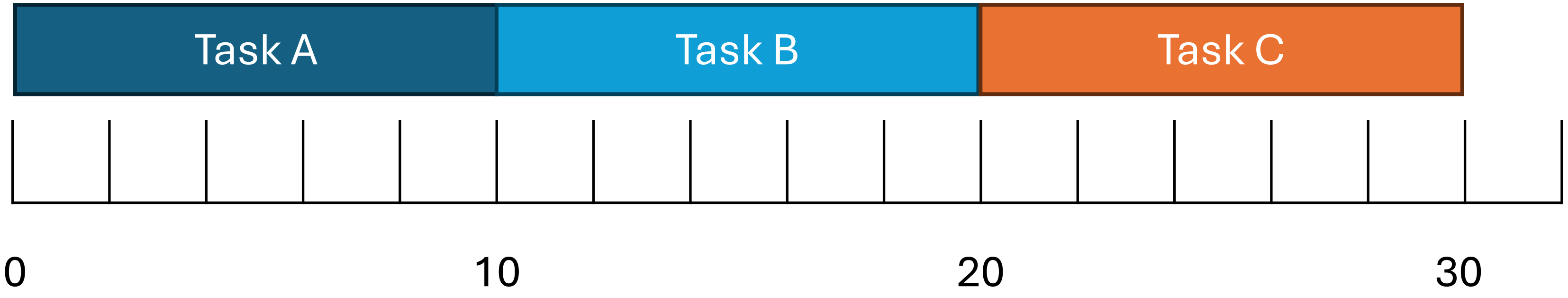
Shortest Time to Completion First (STCF)

Metrics

- Turnaround Time completion_time – arrival_time
- **Response Time** start_time – arrival_time



Job	Arrival Time	Burst Time	Turnaround Time	Response Time
A	0	10	10	0
B	2	10	18	8
C	4	10	26	16



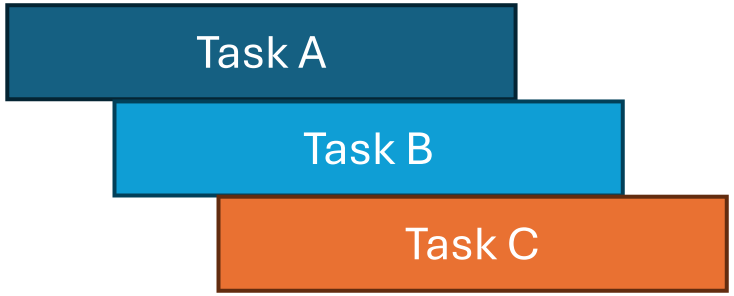
$$\frac{0 + 8 + 16}{3} = 8$$

Average

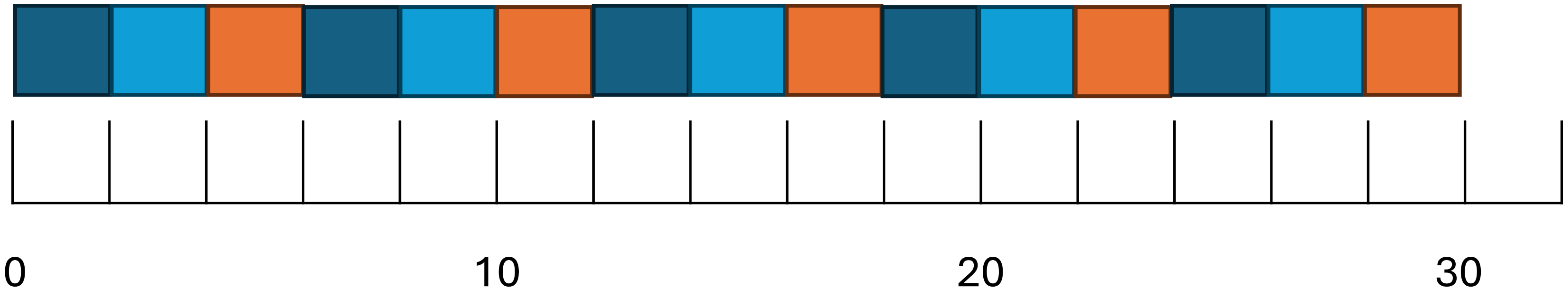
Round Robin (RR)

Metrics

- Turnaround Time completion_time – arrival_time
- Response Time start_time – arrival_time



Job	Arrival Time	Burst Time	Turnaround Time	Response Time
A	0	10	26	0
B	2	10	26	0
C	4	10	26	0



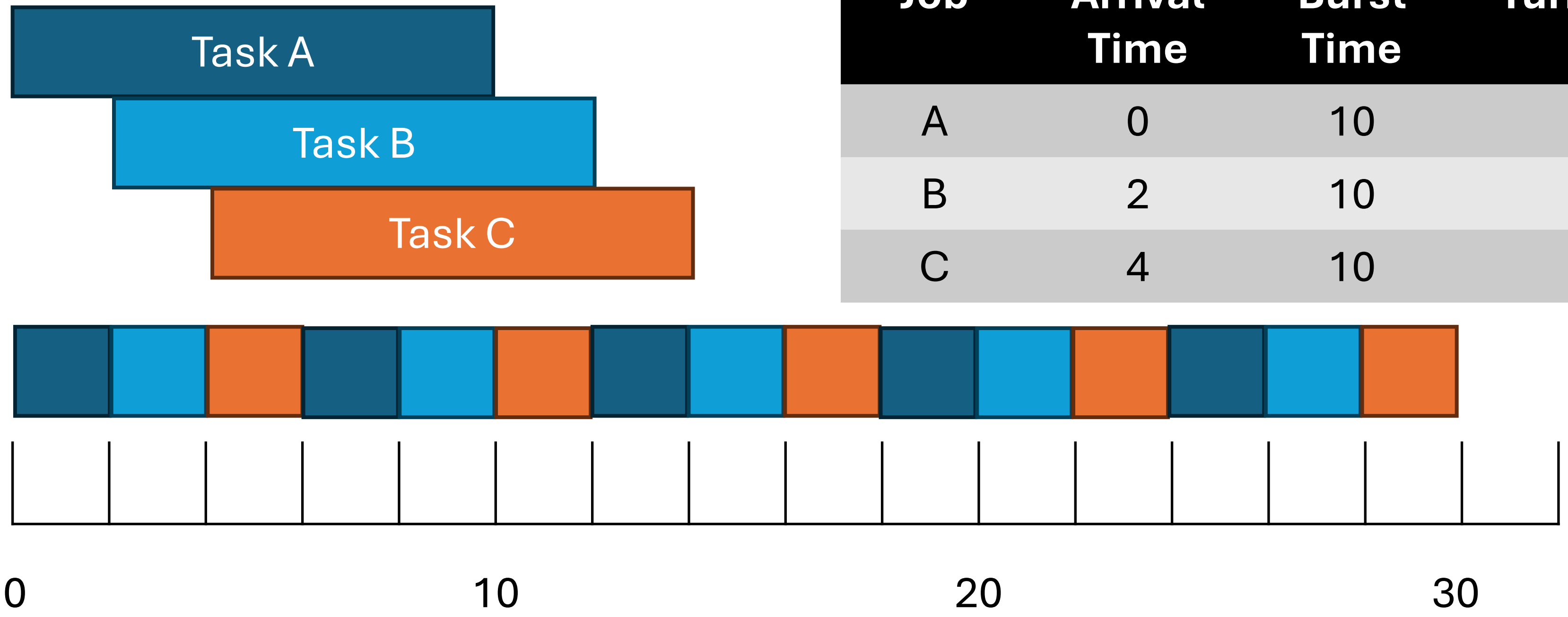
$$\frac{0 + 0 + 0}{3} = 0$$

Average

Round Robin (RR)

Metrics

- Turnaround Time   completion_time – arrival_time
- Response Time   start_time – arrival_time



Job	Arrival Time	Burst Time	Turnaround Time	Response Time
A	0	10	26	0
B	2	10	26	0
C	4	10	26	0

$$\frac{0 + 0 + 0}{3} = 0$$

Average

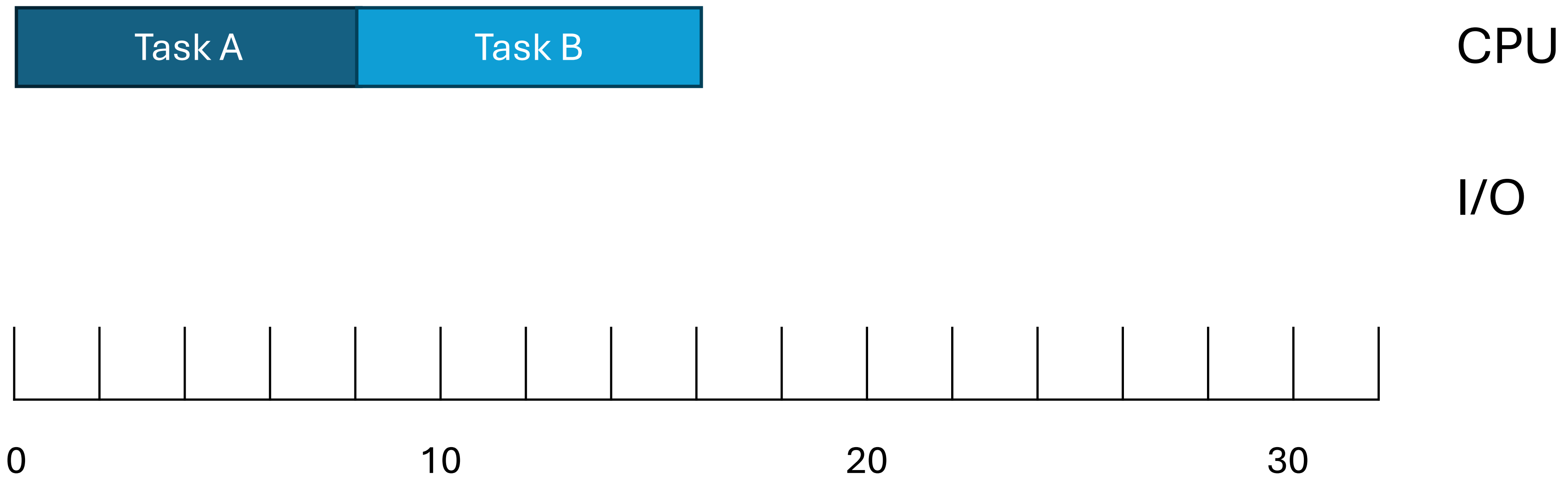
Input/Output Blocking

What are you waiting for?

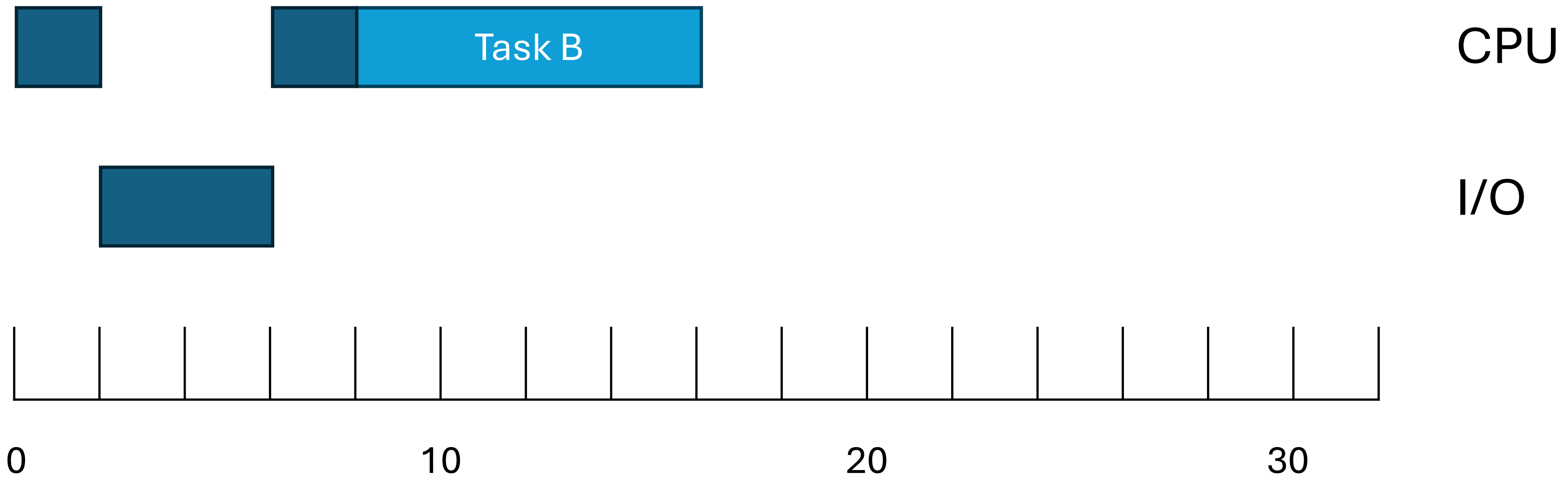
Scheduling Assumptions

1. ~~Jobs run for a fixed time~~
2. ~~All jobs arrive simultaneously~~
3. ~~All jobs are CPU-only~~
4. Run-time of each job is known

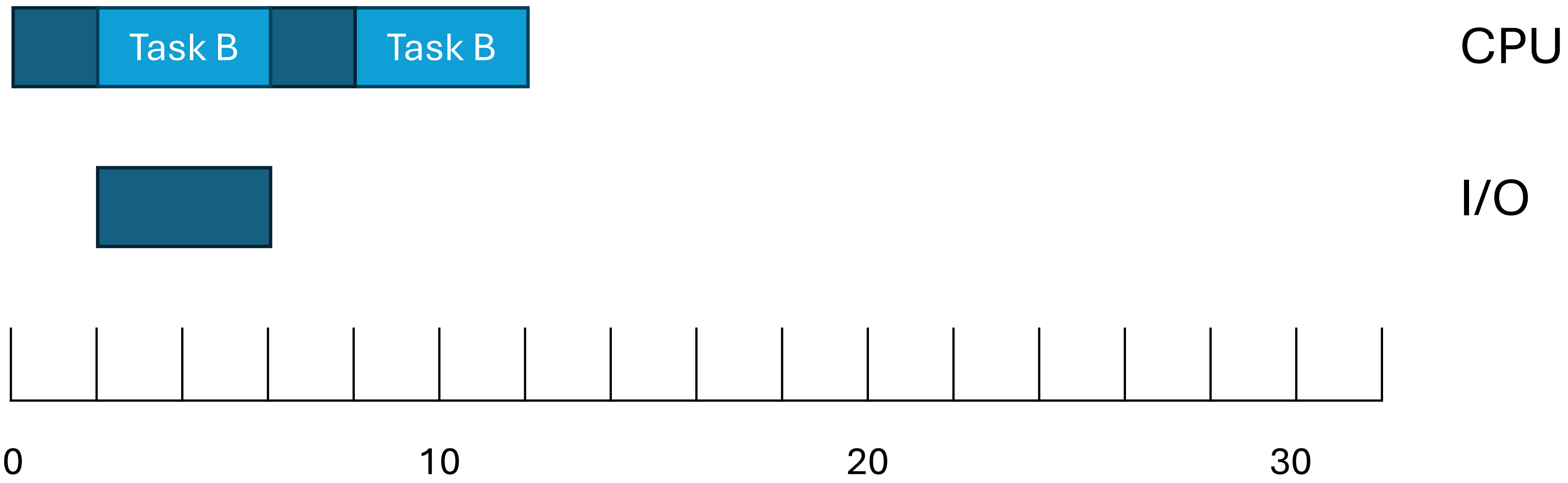
I/O Blocking



I/O Blocking



I/O Blocking



Don’t waste CPU

Scheduling Assumptions

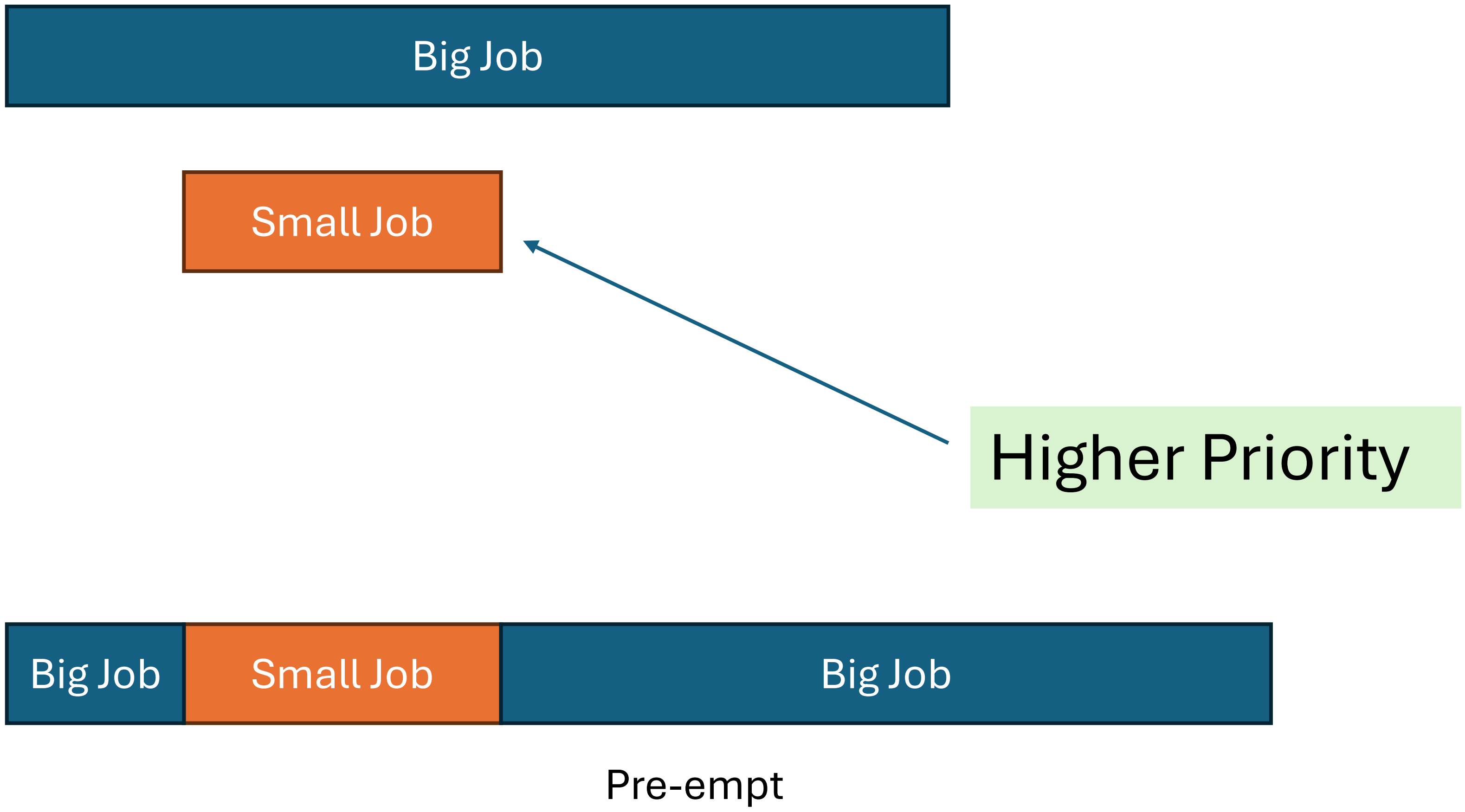
1. ~~Jobs run for a fixed time~~
2. ~~All jobs arrive simultaneously~~
3. ~~All jobs are CPU-only~~
4. ~~Run-time of each job is known~~

Multi-Level Feedback Queue (MLFQ)

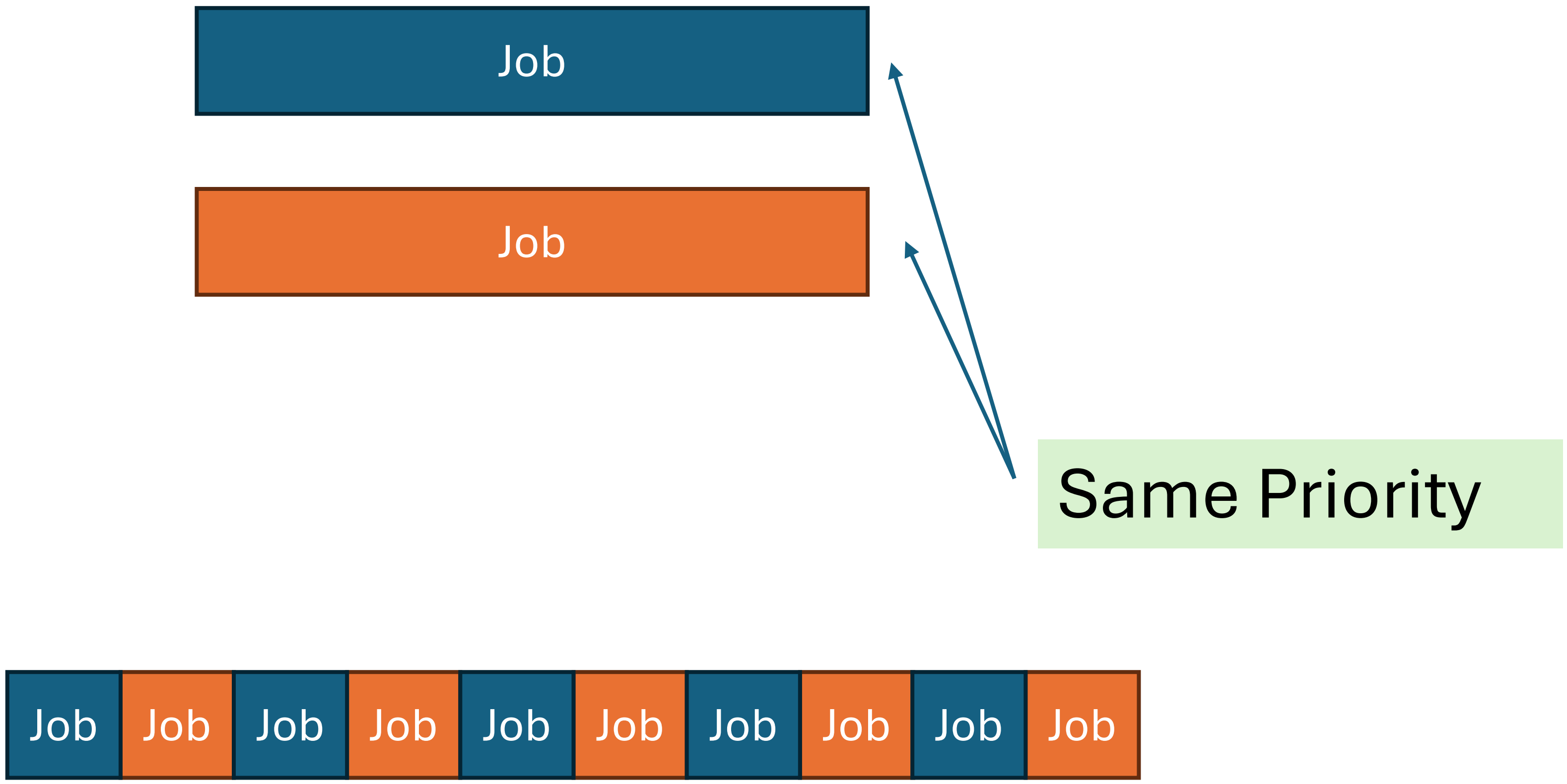
Situation

- Jobs can be of varying length
- Jobs can arrive at any time
- We don’t know when they will finish
- Different jobs have different aims
  - Interactive Jobs: **Response Time**
  - Batch Jobs: **Turnaround Time**

Priority (Fixed)

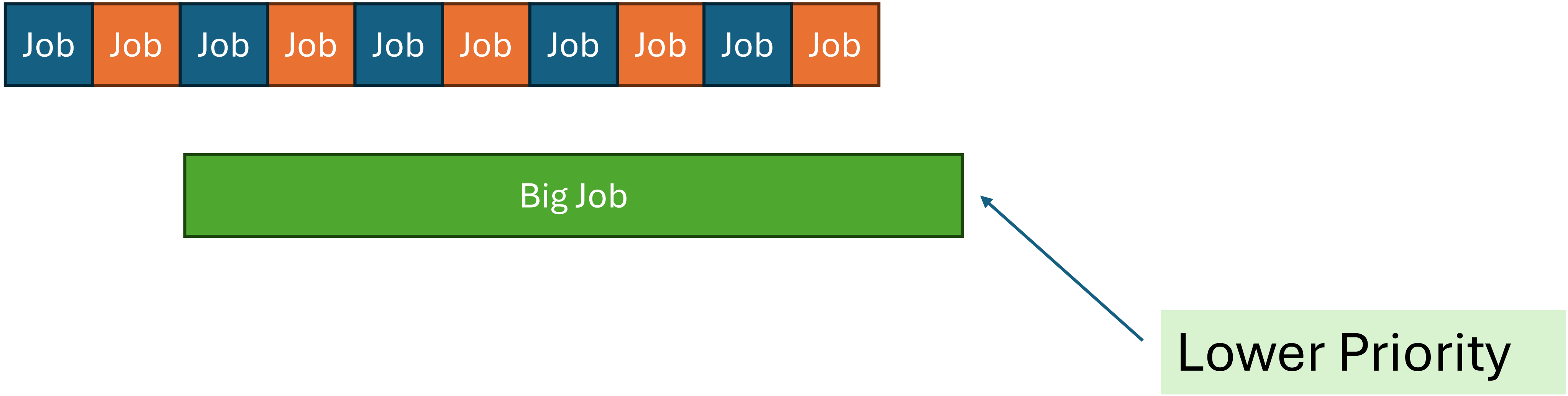


Priority (Fixed)



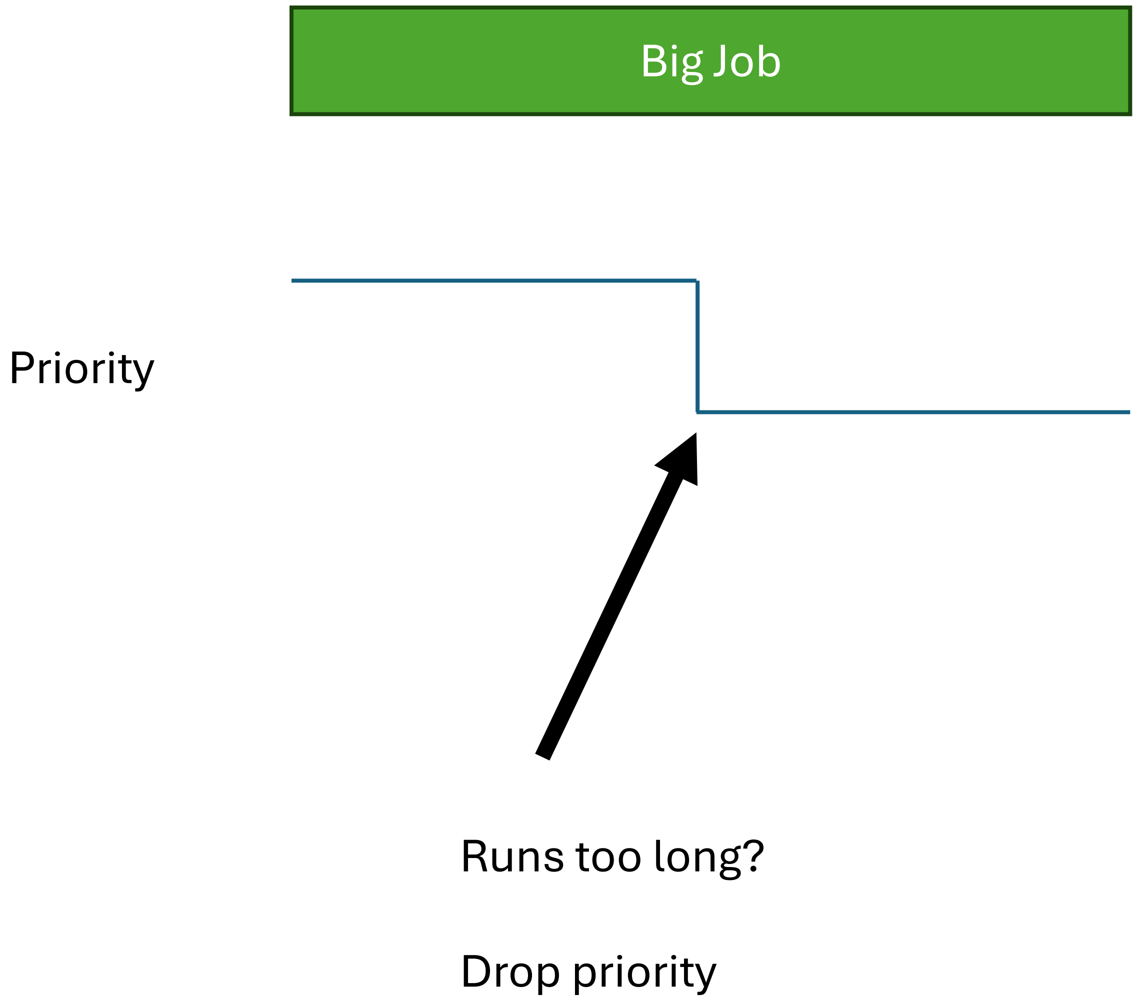
Round Robin

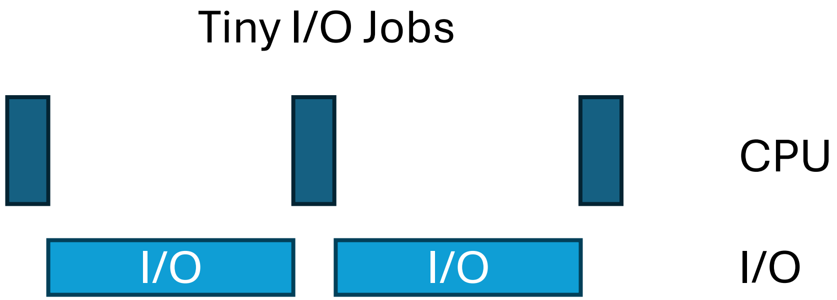
Priority (Fixed)



It will never run...

Priority (Dynamic)





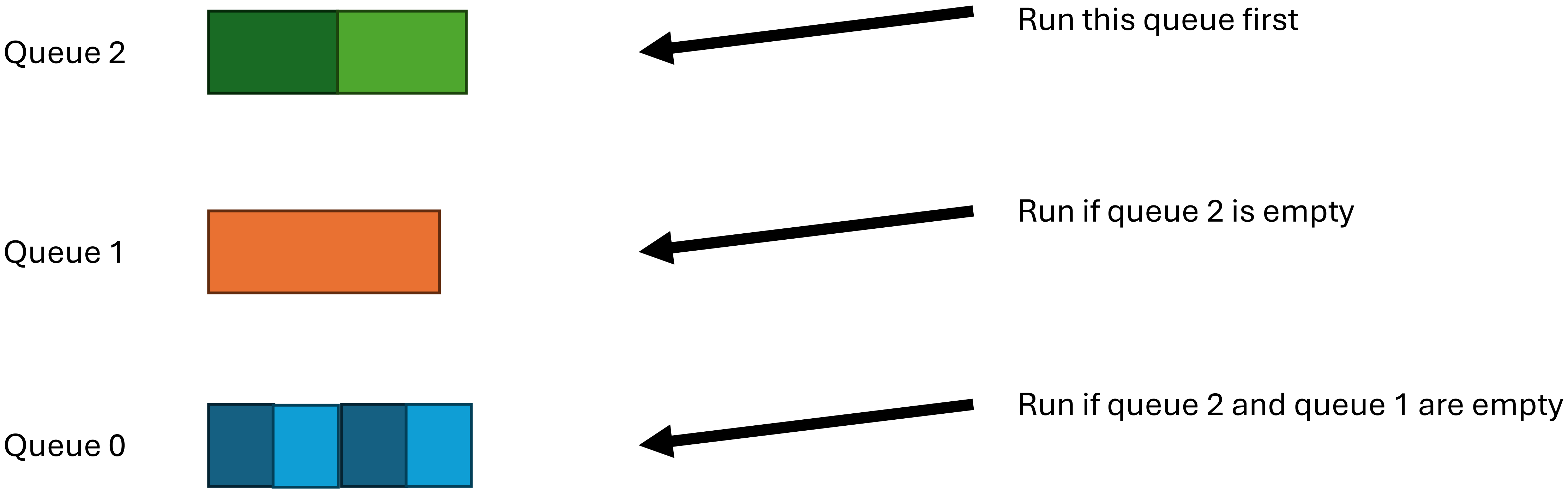
Logic:
If a job takes a lot of CPU (i.e., everything you give it), reduce its priority...

If a job uses only a little CPU, keep its priority

Priority (Dynamic)

If a job can only ‘lose priority’, won’t it eventually get ignored?

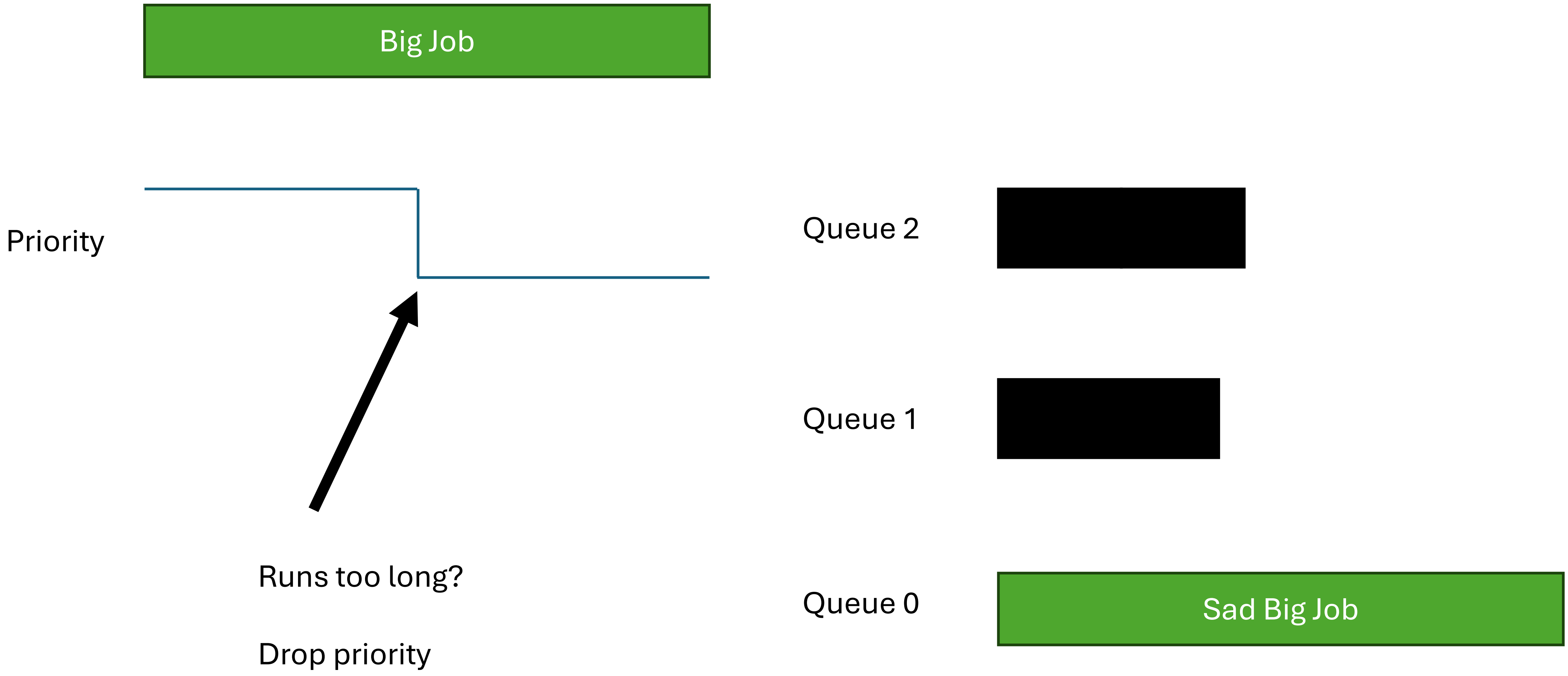
Multi-level Feedback **QUEUE**



Priority (Evil)

Under this system, how can we be evil?

Priority (Dynamic)

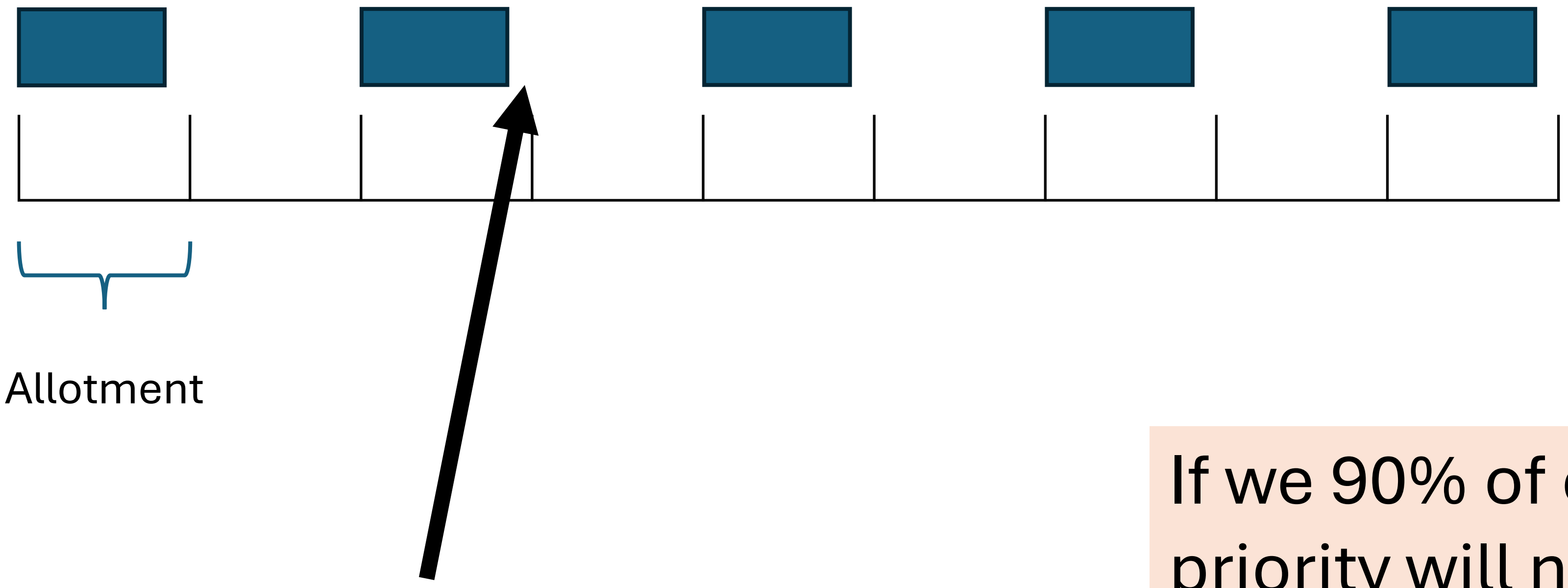


MLFQ

Rules:

- Run high priority jobs first (round robin for ties)
- You start high priority
- If you run for too long, your priority drops
- Every so often, put jobs back into the top priority

Priority (Evil)



Leave a tiny gap 😈

If we 90% of our allotment, our priority will never fall...

MU HA HA HA HA HA HA

MLFQ

Rules:

- Run high priority jobs first (round robin for ties)
- You start high priority
- If you run for **use a full allotment**, your priority drops
- Every so often, put jobs back into the top priority

MLFQ

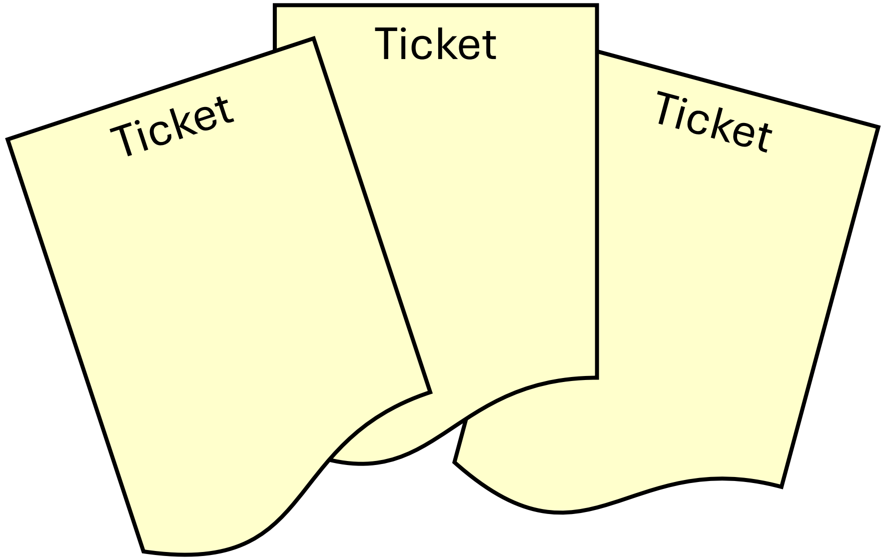
Design Decisions:

- How many queues?
- How long is an allotment?
  - Are allotments the same for each queue level?
- How often to refresh priorities?

Lottery Scheduling

Choose processes ‘at random’

- Processes get tickets, and you choose a random ticket
- Priorities => More Tickets



Some weird thoughts

- Ticket trading?
- Ticket inflation?
- Ticket currencies?

Completely Fair Scheduler (CFS)

Premise:

- All tasks have their **vruntime** measured
  - We can mess with this by changing the weighting of the increments of runtime
- Smallest **vruntime** tasks are run with priority

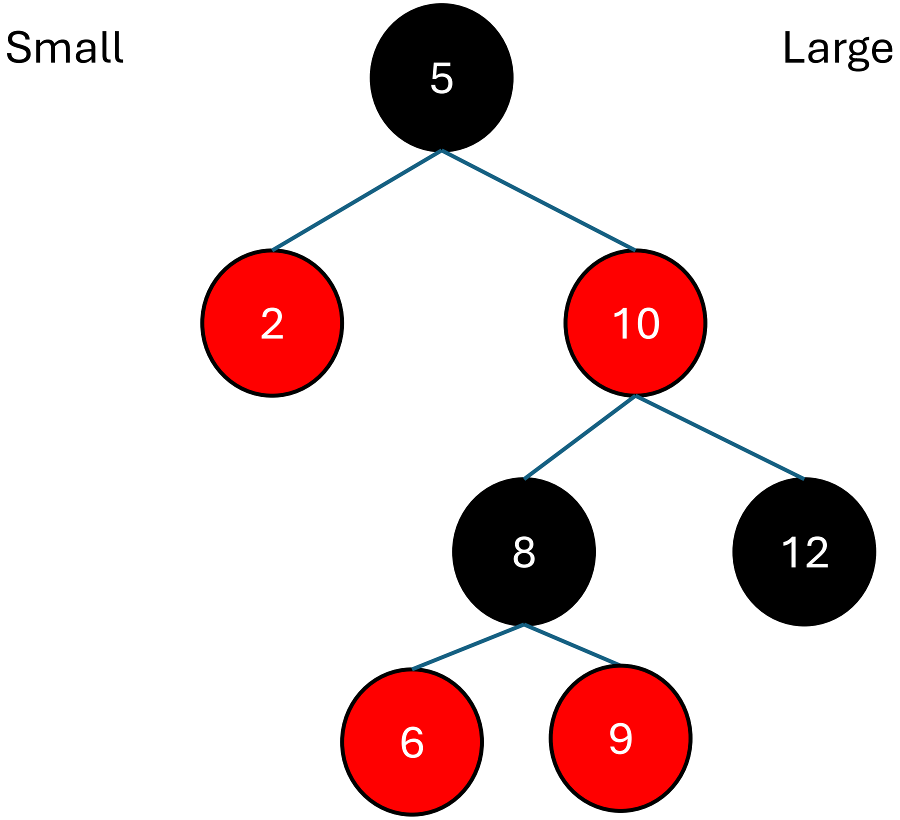
- Each task will run for a bit over a **target latency** period
- Each task gets a share depending on its weight (so no starvation
- Works using a Red/Black tree

Run in Linux since 2007

Completely Fair Scheduler (CFS)

Why?

- Priority is important to know
- Finding the highest priority needs to be low complexity (O(log(n))



Completely Fair Scheduler (CFS)

Example

- Task 1: Video Rendering
  - Long run-time, CPU intensive
  - Non-interactive
- Task 2: Word Processor
  - Intermittent, CPU bursts
  - Interactive

Result

Task 2 accumulates little **vruntime**. When it wakes (user input) it gets run immediately.

Completely Fair Scheduler (CFS)

nice(1)

The ‘**nice**’ can be thought of as a ‘soft’ priority.

- Changing a program’s **nice** value changes the weighting for CFS.

The ‘**nice**’ values correspond to the weightings so you get a proportion of CPU time

(lower nice = more CPU)

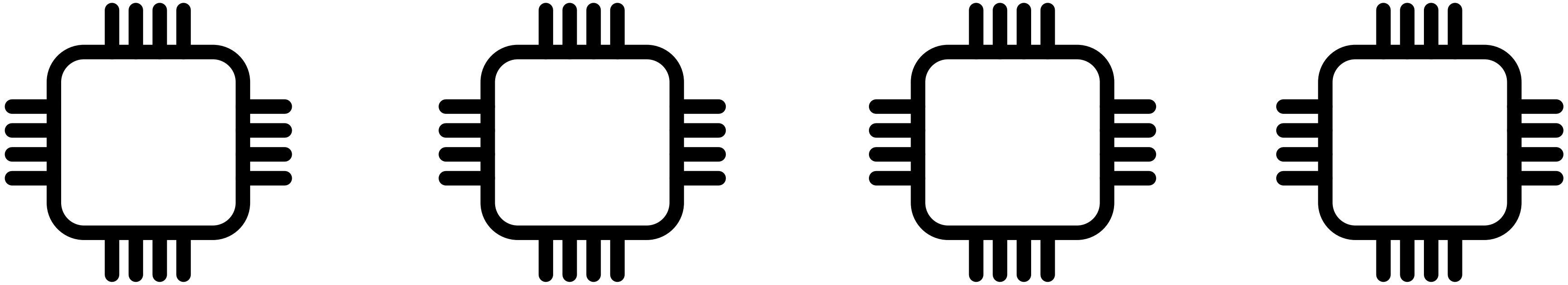
Multi-core CPUs

Linux

- One red-black tree per core

  But…

How to fairly distribute tasks between cores?



Summary

Scheduling:

• First Come, First Serve	FCFS (also FIFO)
• Round-Robin	RR
• Shortest Job First	SJF
• Multi-level Feedback Queues	MLFQ

Questions?