Concurrent Systems

3D4 ⟷ CS2016

Operating Systems

Andrew Butterfield
ORI.G39, Andrew.Butterfield@scss.tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

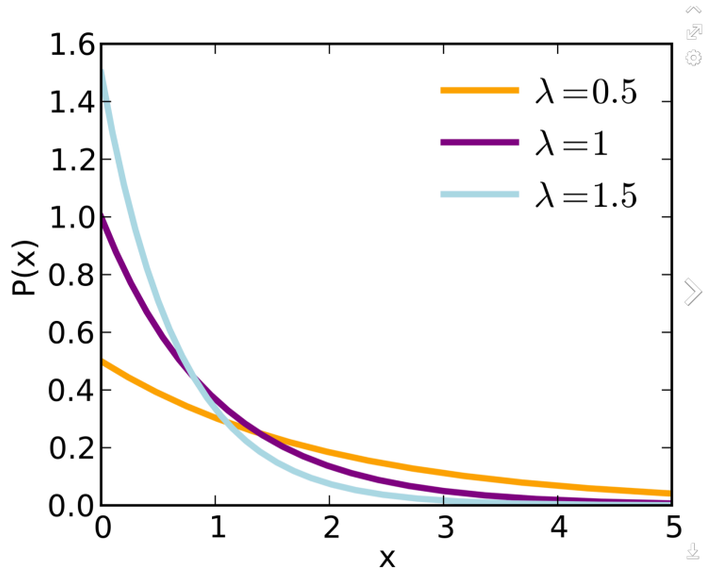
with thanks to Mike Brady

Scheduling

- Only one process can execute at a time on a single processor system
  - A process is typically executed until it is required to wait for some event
    - e.g. I/O, timer, resource
      - When a process' execution is stalled, we want to maximise CPU utilisation by executing another process on the CPU.
- The component of an OS that gives time to processes is the *Scheduler*
- Schedulers implement various *scheduling policies*

Typical Process Behaviour

- CPU I/O Burst Cycle
  - It's been observed that processes often alternate between bursts of CPU and I/O activity
  - CPU burst times seem to have an exponential frequency curve with many short bursts and very few long bursts
    - i.e., they occur independently at a constant average rate
    - they are "memoryless" - past occurrences have no effect on likelihood of future ones.



https://en.wikipedia.org/wiki/Exponential_distribution#/media/File:Exponential_pdf.svg

CPU Scheduler

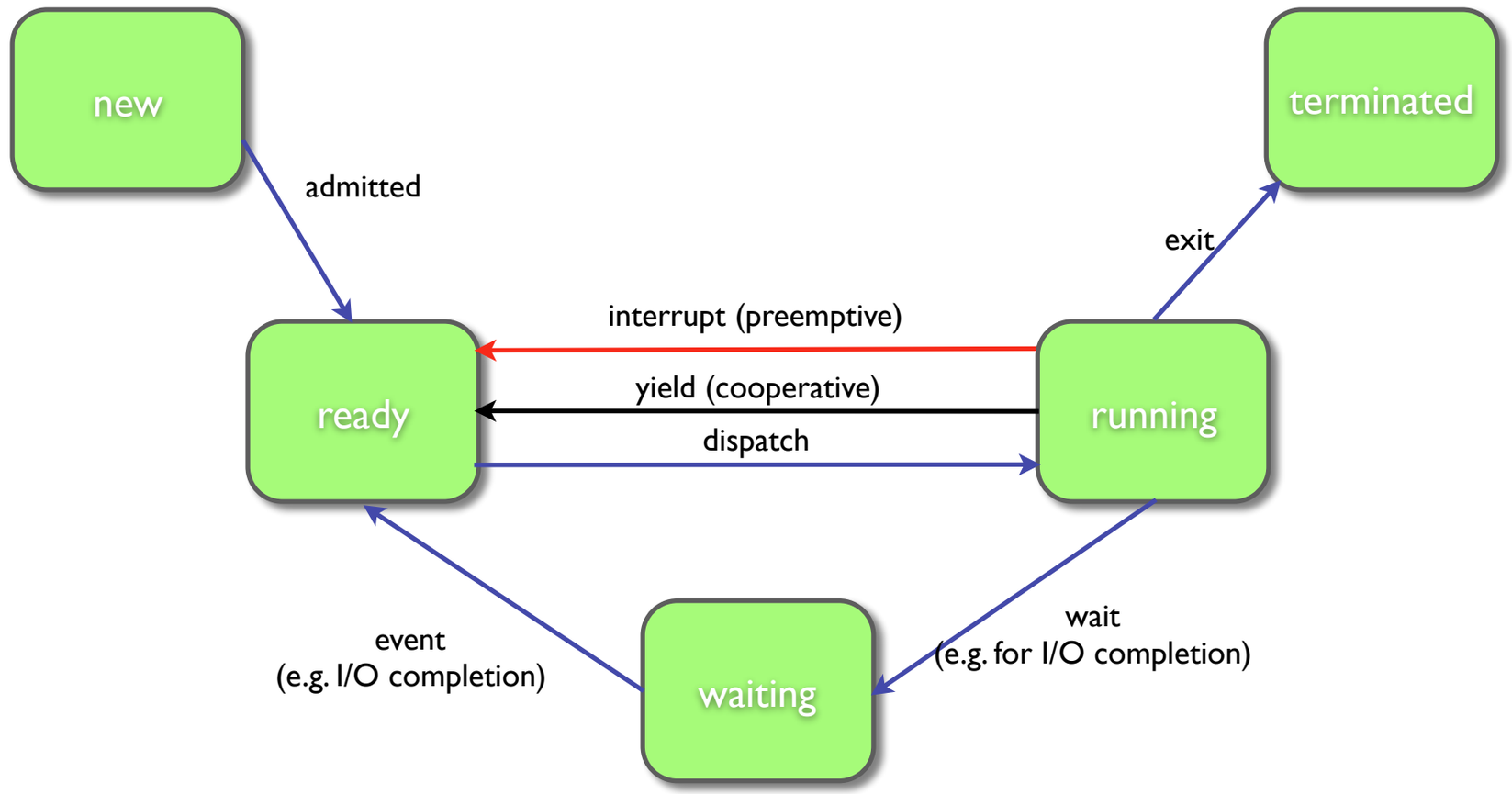
- When the CPU becomes idle, the operating system must select another process to execute
- This is the role of the *short-term scheduler* or CPU scheduler.
- Candidate processes for scheduling are those processes that are in memory and are ready to execute
- Candidate processes are stored on the ready pool
- The ready pool is not necessarily a FIFO queue, but depends on the implementation of the scheduler, e.g.
  - FIFO queue, priority queue, unordered list, …

The long view…

- The number of processes or programs eligible for execution in the first place must also be managed
  - a more long-term approach is needed.

Process state transitions

- Process state transitions
  - New process is admitted when code and initial data are loaded into memory
  - Only one process can be in the running state on a single processor at any time
  - Multiple processes may be in the waiting or ready states

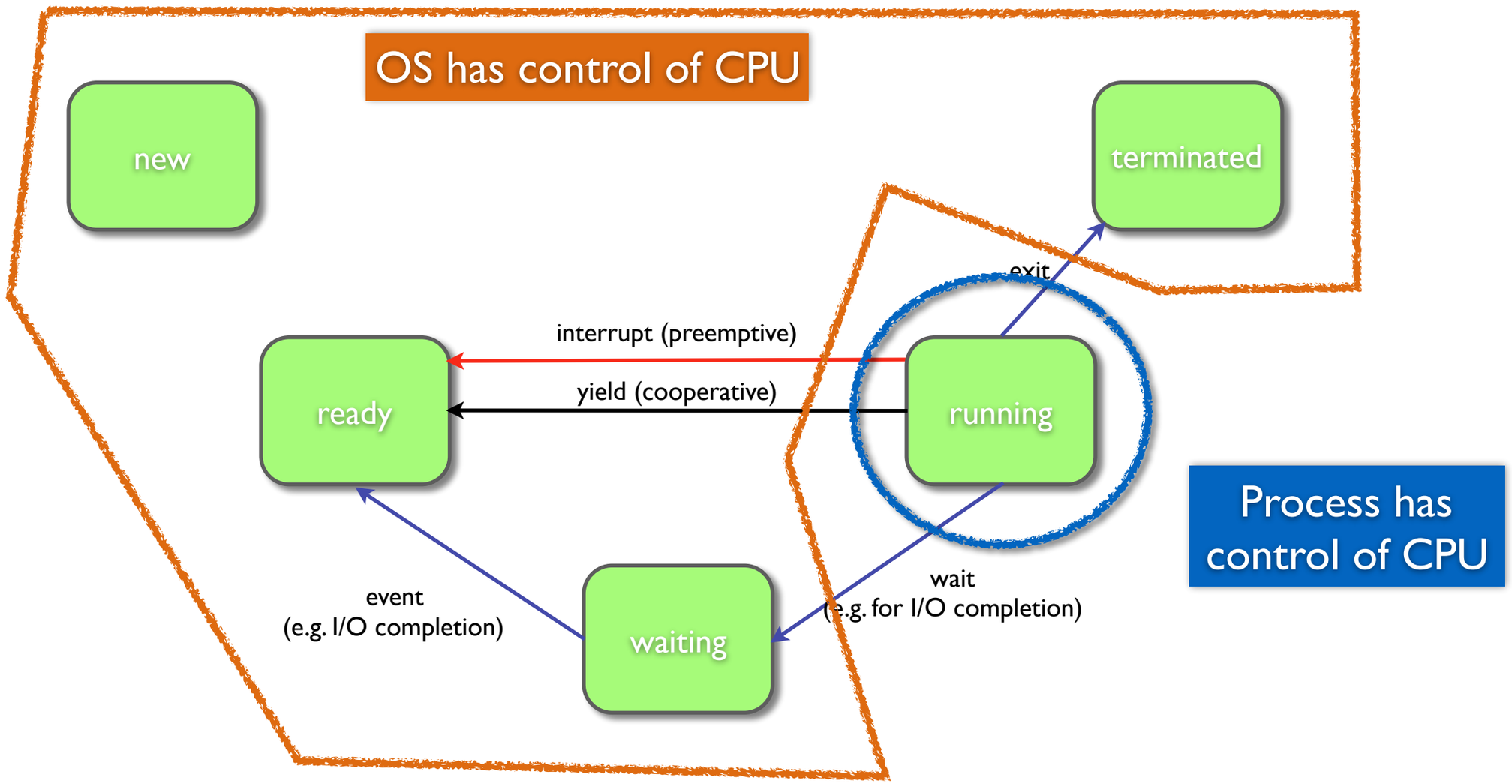


Scheduling Opportunities

- Scheduling may be performed in any of these situations

1. When a process leaves the running state and enters the waiting state, waiting for an event (e.g. completion of an I/O operation)
2. [PREEMPTIVE] When a process leaves the running state and enters the ready state (e.g. when an interrupt occurs)
3. [PREEMPTIVE] When a process leaves the waiting state and enters the ready state
4. When a process terminates

Scheduling opportunities



Preemptive and Cooperative scheduling

- With a cooperative (i.e. non-preemptive) scheme
  - Once a process has entered the running state on a CPU, it will remain running until it relinquishes the CPU to another process
- Preemptive scheduling incurs a cost
  - Access to shared data?
  - Synchronisation of access to kernel data structures?

Non-Preemptive Scheduling

- Non-Preemptive Scheduling is also called *Cooperative Scheduling* and relies on the processes to be "well-behaved" and to voluntarily relinquish the processor.
  - Thus, cooperative scheduling (and thus cooperative multitasking) is always fragile!
- The advantage of Cooperative Scheduling is that, because the processes are designed to relinquish the processor, they can be designed to minimise the overhead of doing so.
  - This is often a very big advantage in specialised situations, e.g. inside the kernel of an OS, where efficiency is paramount and processes can be trusted.

Scheduling Goals

- Fast response time
  - Look at Waiting Time (i.e. latency) and Turnaround Time (i.e. time from arrival to end of burst).
- High throughput
  - Avoid wasting time – avoid high level of context switches
- Fairness
  - If we implement priorities, be careful to ensure starvation is avoided.

Some Scheduling Algorithms

- These are just some standard scheduling algorithms.
- In practice, real scheduling uses variants and mixtures.

	First-Come-First-Served (FCFS)	**Shortest Job First**	**Priority**	**Round Robin**
Non-Preemptive	✔	Fragile	Fragile	This is FCFS
Pre-Emptive	*What could this be?*	✔	✔	✔

First-Come First-Served (FCFS)

- The process that requests the CPU first is the first to be scheduled
- Implemented with a FIFO ready queue
- When a process enters the ready state, it is placed at the tail of the FIFO ready queue
- When the CPU is free and another process must be dispatched to run, the process at the head of the queue is dispatched and the running process is removed from the queue

First-Come First-Served (FCFS) – 2

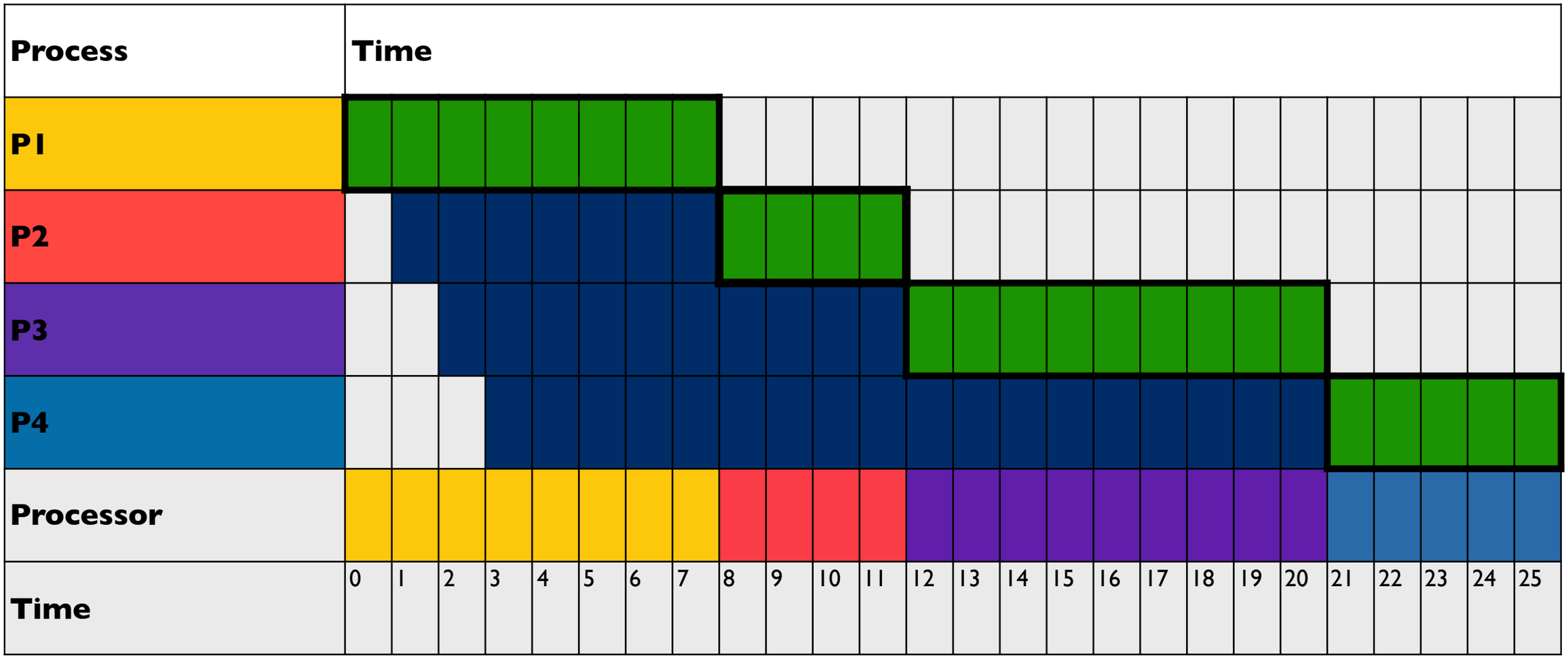
- The average waiting time under FCFS scheduling can be quite long and depends on the CPU burst times of the processes
- Consider four processes: P1, P2, P3 and P4 that appear on the FCFS ready queue in the order P1, P2, P3 and P4 and with burst times of 8, 4, 9 and 3 milliseconds respectively

Example

- Burst Time: the amount of time the process could use in a continuous run, without a break for, e.g. I/O.

Process	Arrival Time	Next Burst Time	Priority (Lower is Better)
P1	0	8	2
P2	1	4	4
P3	2	9	1
P4	3	5	3

First Come, First Serve



First Come First Serve

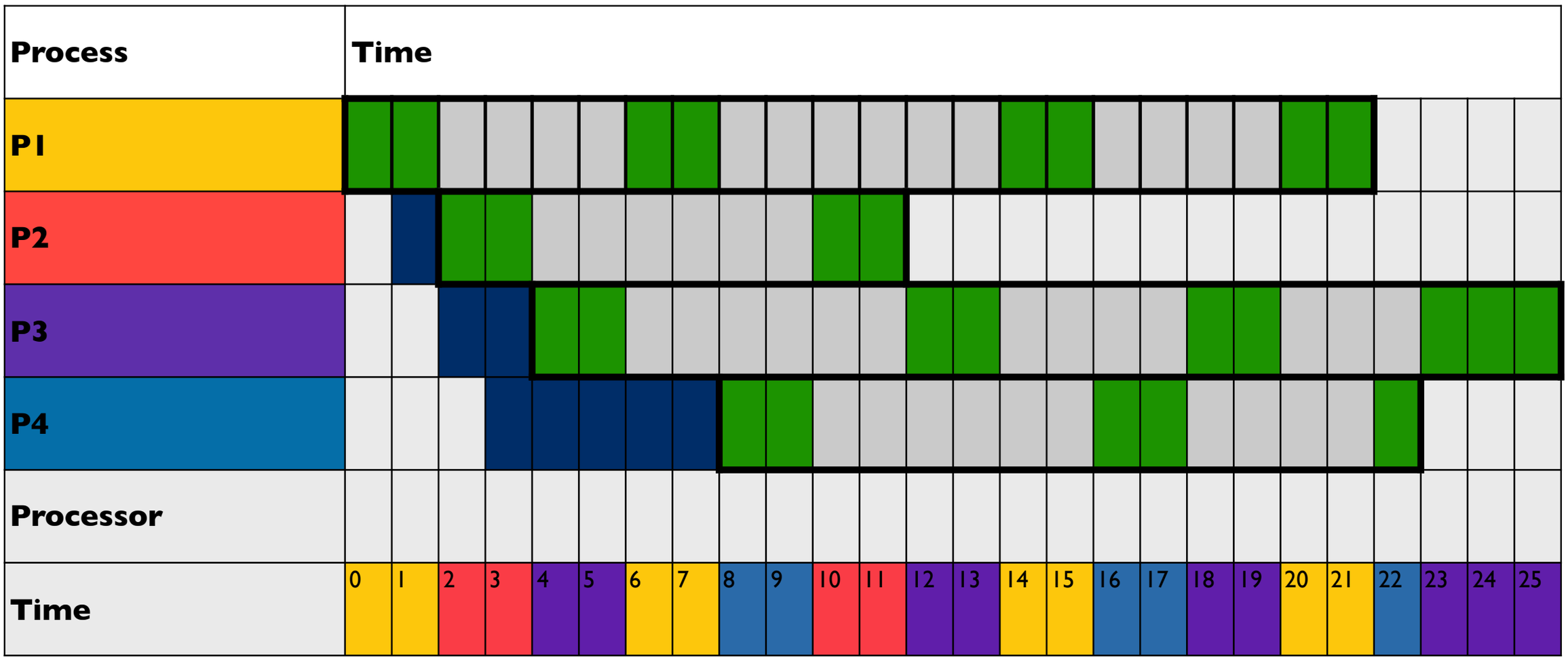
Process	Arrival Time	Next Burst Time	Waiting Time	Turnaround Time
P1	0	8	0	8
P2	1	4	8-1 = 7	12-1 = 11
P3	2	9		
P4	3	5		

- Waiting Time is *Latency*.
  - *Mean time and Variance both important.*
- Turnaround Time = Waiting Time + Burst Time.

Round Robin

- The process that requests the CPU first is the first to be scheduled
- Implemented with a FIFO ready queue
- When a process enters the ready state, it is placed at the tail of the FIFO ready queue
- When the CPU is interrupted, a scheduling event occurs:
  - The next process, at the head of the queue, is dispatched and the interrupt process is placed on the end of the queue

Round Robin, Quantum = 2



Round Robin Scheduling

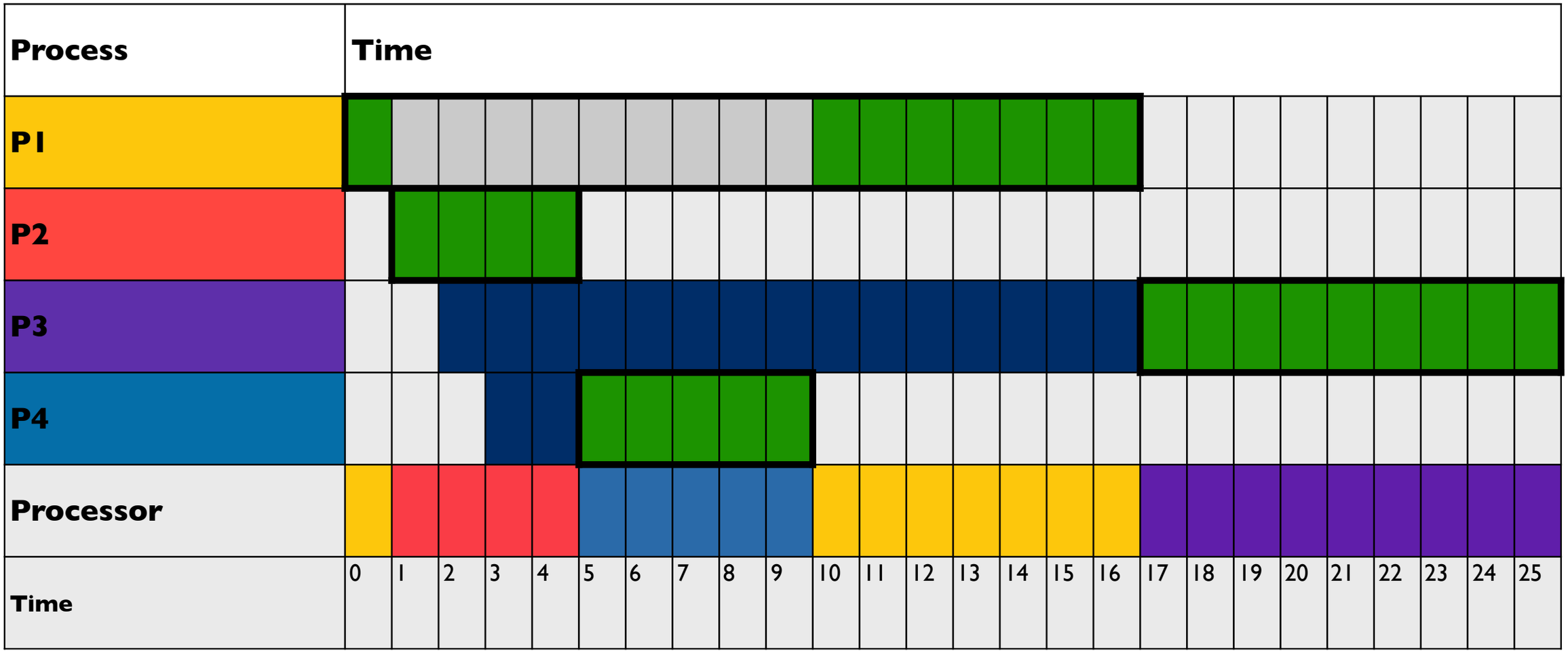
Process	Arrival Time	Next Burst Time	Waiting Time	Turnaround Time
P1	0	8		
P2	1	4		
P3	2	9		
P4	3	5		

- Waiting Time is generally shorter. *(Why?)*
  - *Variance is probably also shorter.*
- Turnaround Time is generally longer. *(Why? How much?)*

Shortest Job First with Pre Emption

- When a process becomes ready:
  - if it is longer than the current running process, then enter the ready queue
  - if shorter, pre-empt the running process, and take its place
- When a process enters the ready state, it is placed in the ready queue based on its length w.r.t. other members of the queue.
- When the CPU is interrupted, a scheduling event occurs:
  - The next process, at the head of the queue, if shorter than the one currently running, is dispatched and the interrupt process is placed on the end of the queue

Shortest Job First with Pre Emption



Shortest Job First (SJF)

Process	Arrival Time	Next Burst Time	Waiting Time	Turnaround Time
P1	0	8		
P2	1	4		
P3	2	9		
P4	3	5		

- Waiting Time is generally shorter. *(Why?)*
  - *Variance is probably also shorter.*
- Turnaround Time is generally longer. *(Why? How much?)*

Ahem! I have a teeny question …

- ?!? How does the scheduler know the job length ?!?