Week 2

Introduction to Process Management

Oana Balmau

January 12, 2023

Schedule Highlights

Lab recording on grading infrastructure will be posted on MyCourses by the end of the week. Tutorial coming soon. Have a look and try the HelloWorld if you have not done so already.

Topic	Monday	Tuesday	Wednesday	Thursday	Friday
Week 1 **Introduction**	jan 2	jan 3	jan 4 – first day of class 😊	jan 5 **Course logistics and Intro to OS**	jan 6
Week 2 **Process Management**	jan 9 **Workflow: working with mimi and GitLab, Git basics**	jan 10 **Intro to Process Management (1/2)** Optional reading: OSTEP Chapters 3 – 7	jan 11	jan 12 **Intro to Process Management (2/2)**	jan 13
Week 3 **Process Management**	jan 16 C Review: C Basics	jan 17 **Synchronization Primitives (1/2)** Optional reading: OSTEP Chapters 25 – 32 add/drop deadline	jan 18 OS Shell Assignment Released	jan 19 **Synchronization Primitives (2/2)** OS Shell Assignment Overview – with Jiaxuan	jan 20
Week 4 **Process Management**	jan 23 C Tools: GDB basics	jan 24 **Multi-process Structuring (1/2)** **Team registration deadline**	jan 25	jan 26 **Multi-process Structuring (2/2)**	jan 27
Week 5 **Process Management**	jan 30 C Review: Pointers & Memory Allocation I	jan 31 **Multithreading (1/2)** Practice Exercises Sheet: Process Management	feb 1	feb 2 **Multithreading (2/2)**	feb 3

Autograder will start sending you email

Teams and GitLab account need to be set up by this day

Use Ed Megathread if you have issues

Team registration info

You register teams on MyCourses, by uploading a text document that contains:

- a link to your fork of the starter code
  - You need to fork https://gitlab.cs.mcgill.ca/balmau/comp310-winter23
  - If you want to code from scratch, you can delete the starter code, but it needs to be a fork of the repository for the autograder to work.
  - The testcases are given with the starter code.
  - You **must use the same fork** for the whole semester, for the 3 assignments.

- Full names and McGill IDs of the teammates
  - Need to register even if you have a “team of 1”.
  - There is one repository per team.

IMPORTANT: If you have already created a fork, please pull the code again on Jan 18, to get the correct version of the starter code and the updated testcases.

Assignments

New Assignment | Edit Categories | More Actions

Bulk Edit

	Assignment	New Submissions	Completed	Evaluated	Feedback Published
	No Category				
	Project Teams	7	5/396	0/396	0/396

What does a process do? (as far as the OS is concerned)

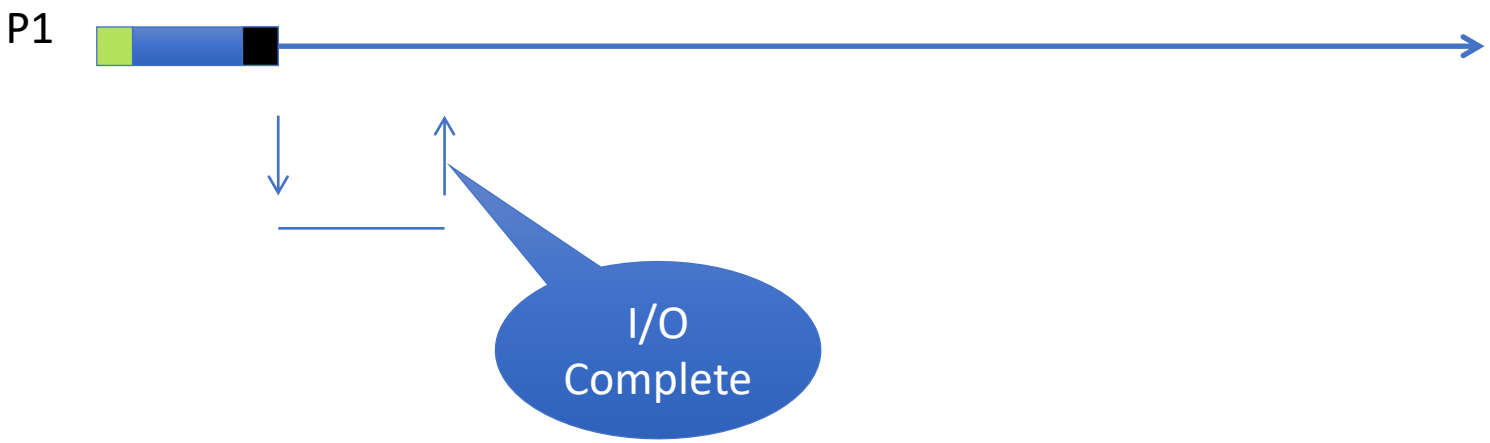
- Either it computes (uses the CPU)
- Or it does I/O (uses a device)

Single Process System

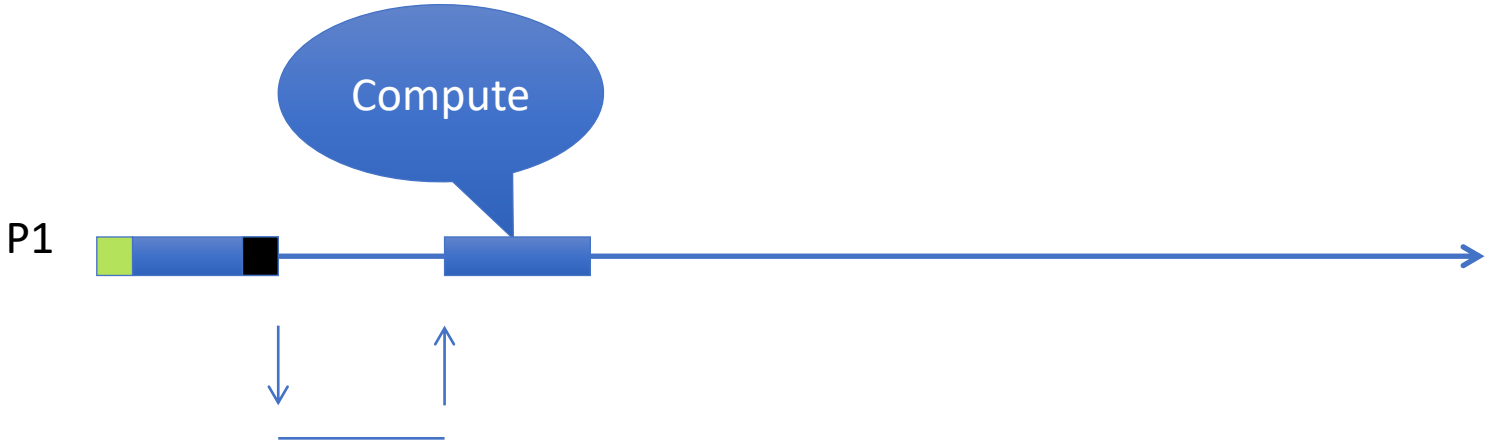
Single Process System

Single Process System

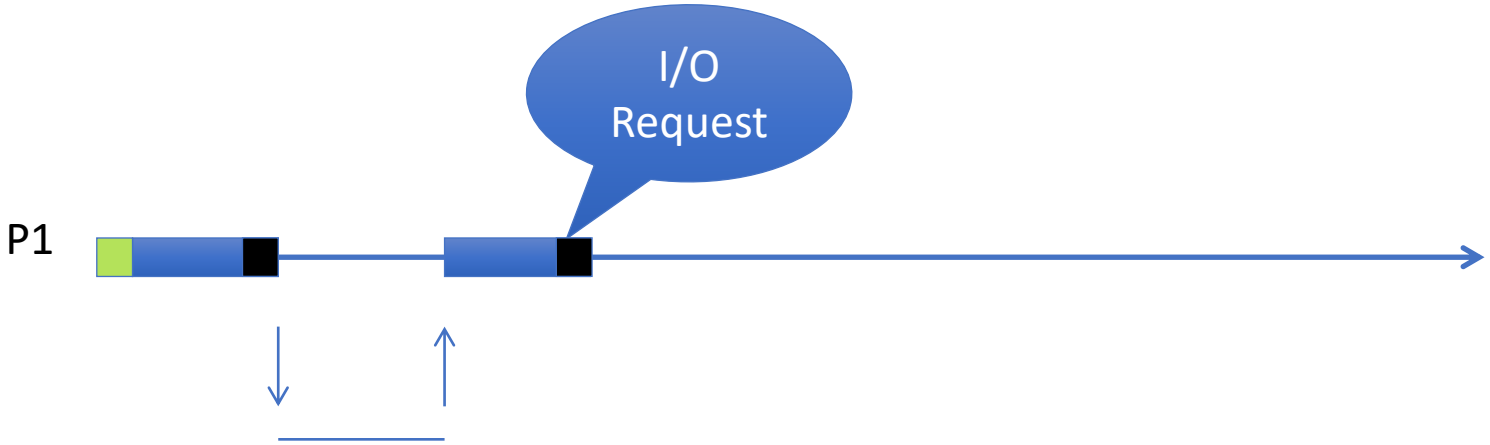
Single Process System



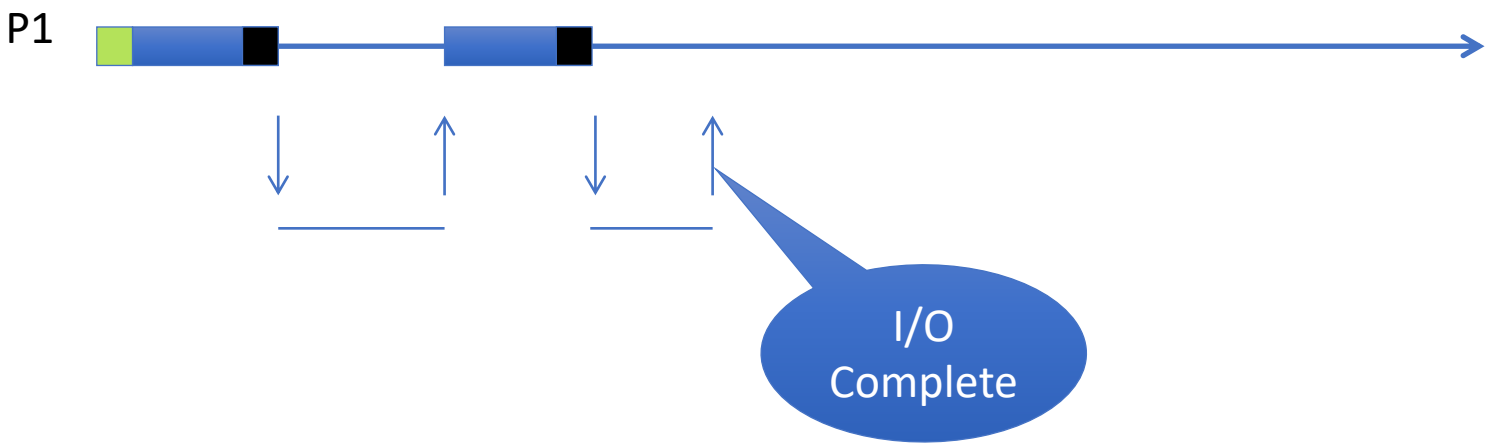
Single Process System



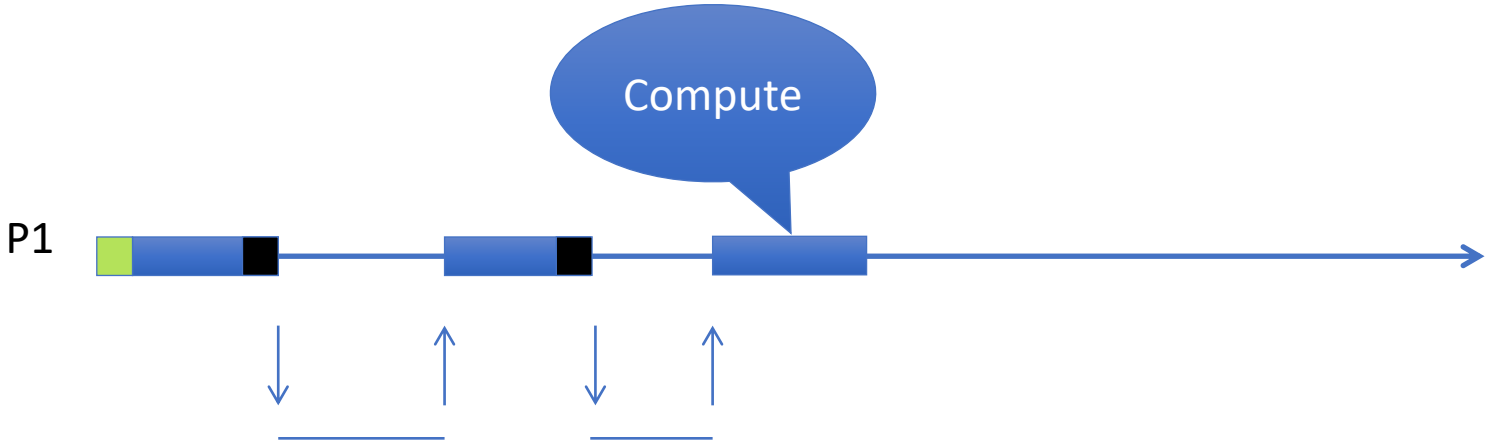
Single Process System



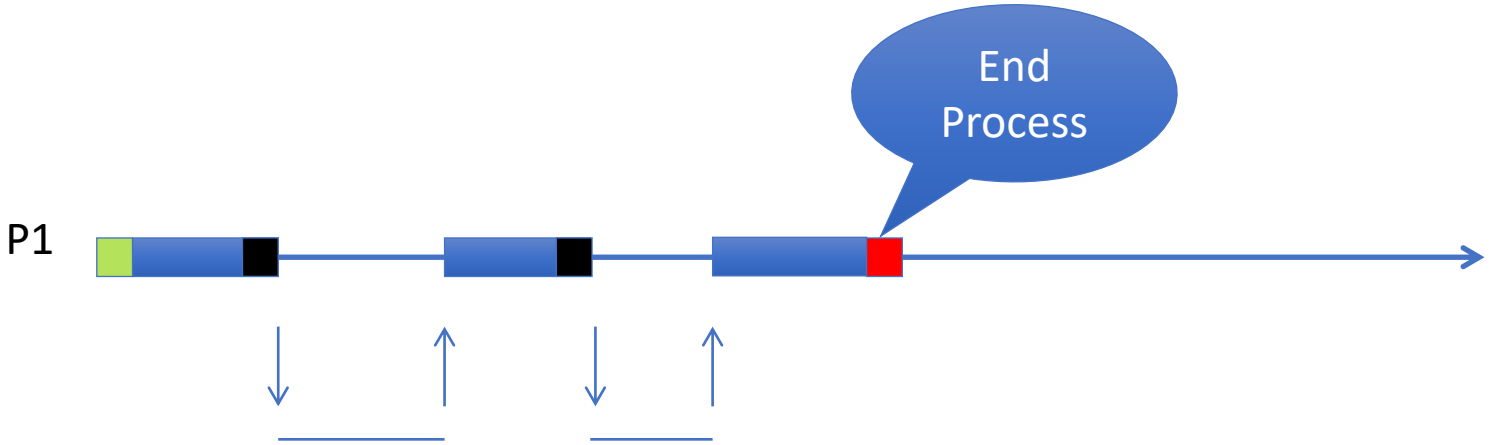
Single Process System



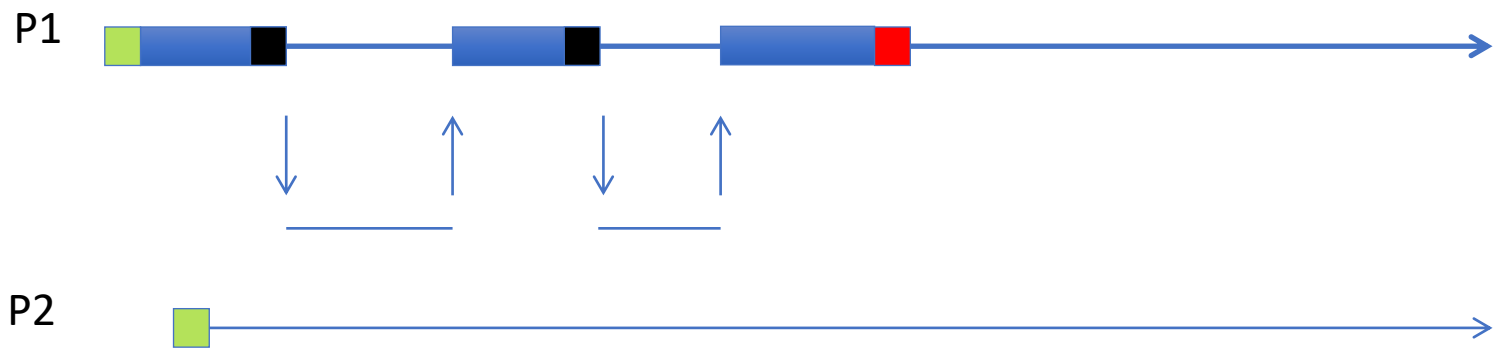
Single Process System



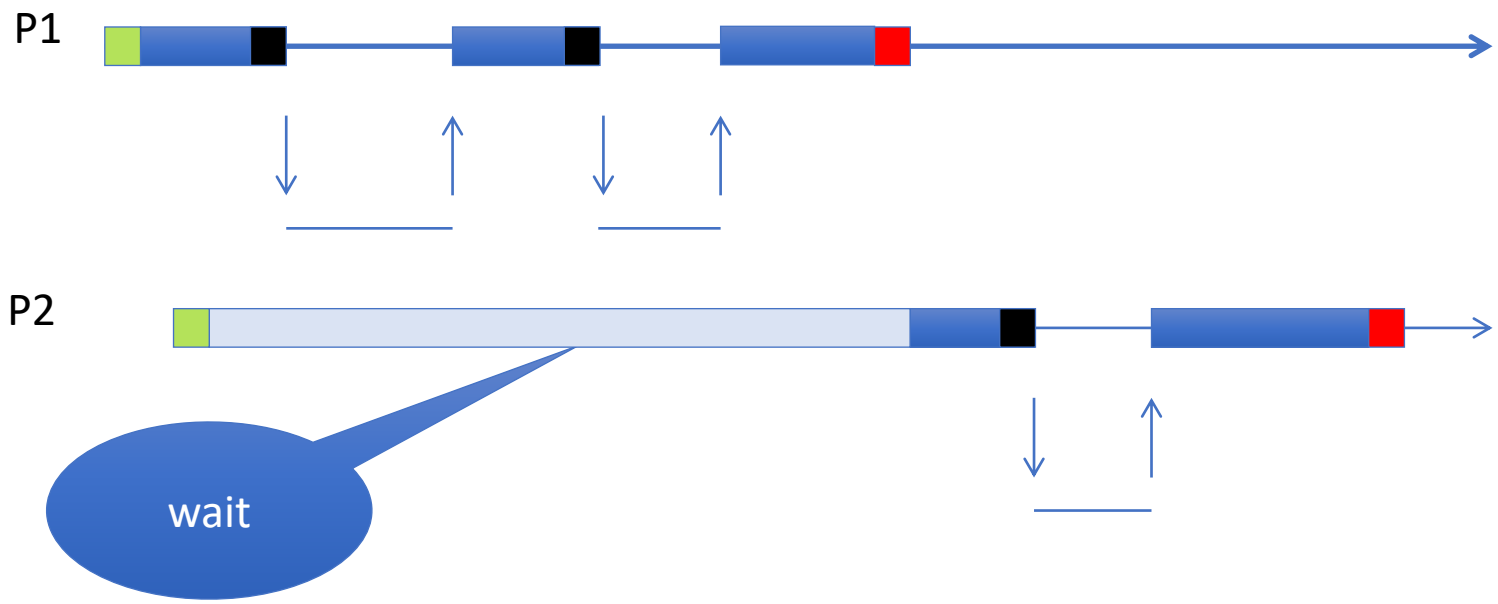
Single Process System



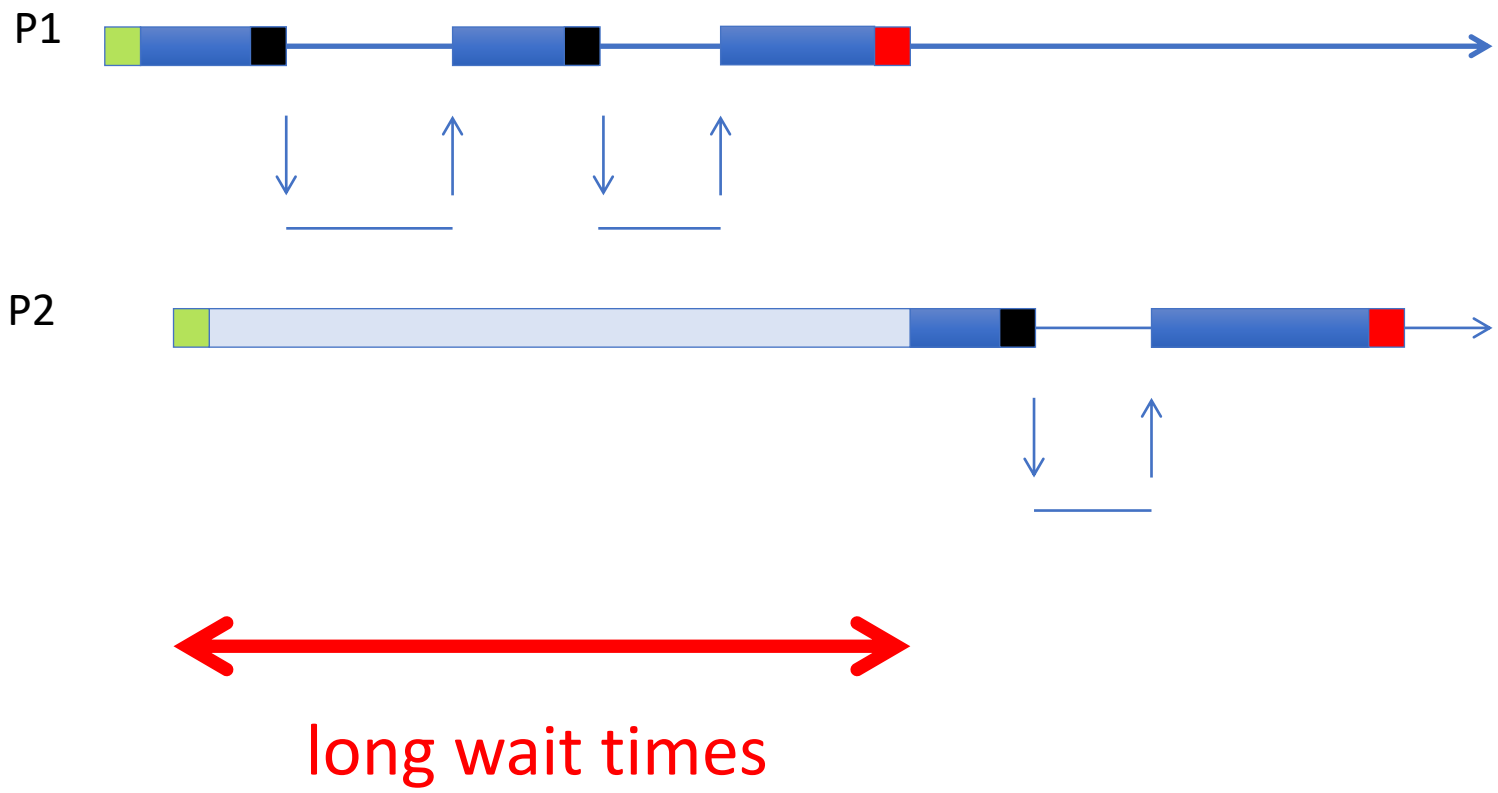
A Second Process



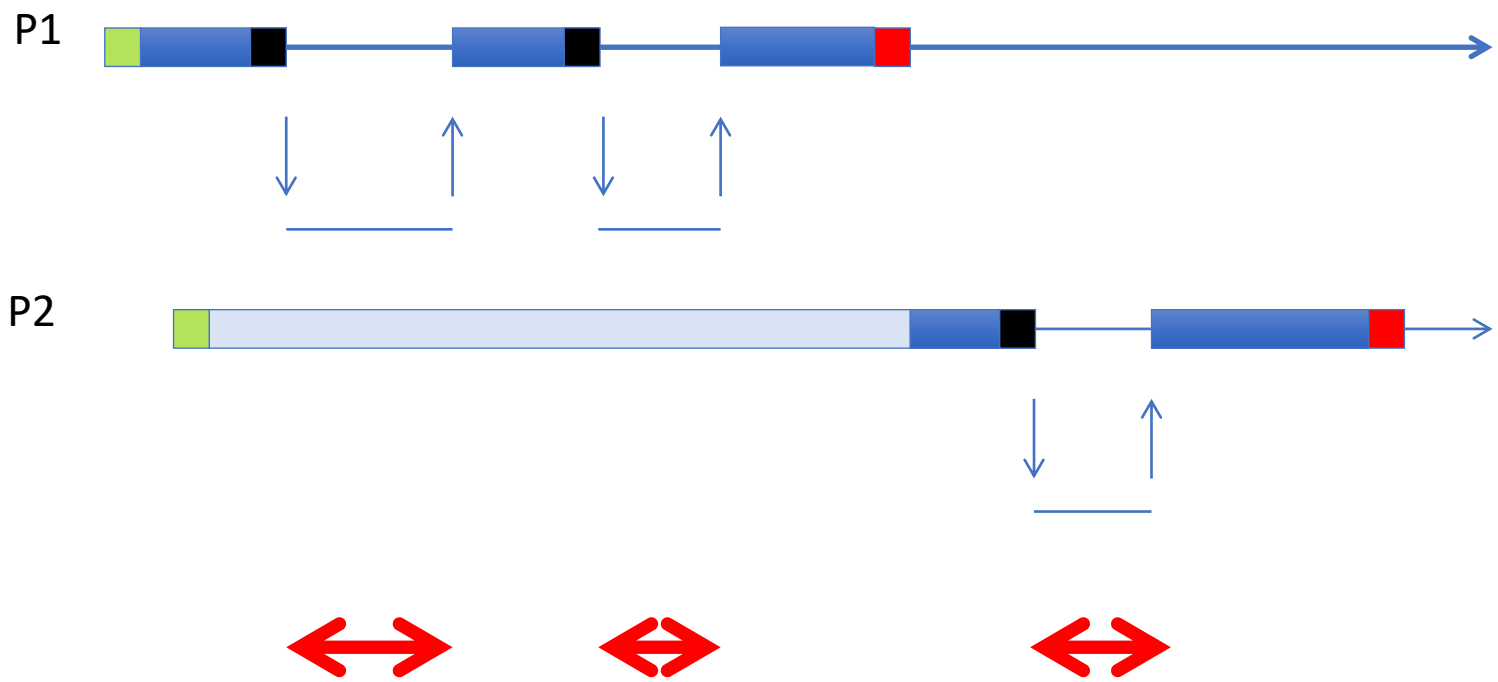
A Second Process



Two Issues



Two Issues



Single Process System

- Is very inefficient
  - Very poor CPU utilization
- Is very annoying
  - You can’t do anything else

Multiprocess System

- Many processes in the system
- One uses the CPU
- When it does an I/O
  - It waits for the I/O to complete
  - It leaves the CPU idle
- *Another process gets the CPU*

Multiprocess System

Multiprocess System

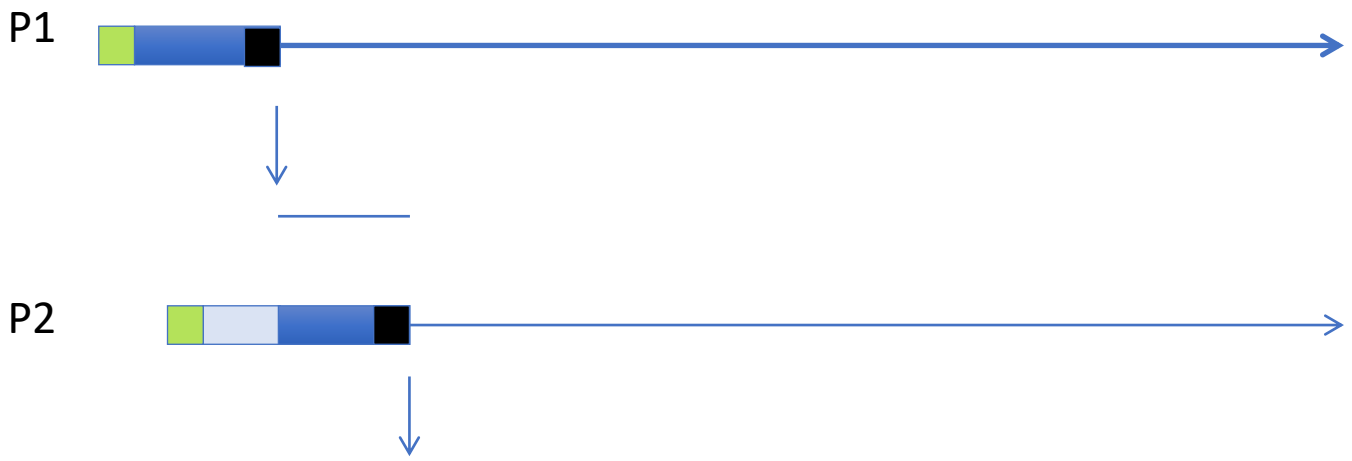
Multiprocess System

Multiprocess System

Multiprocess System

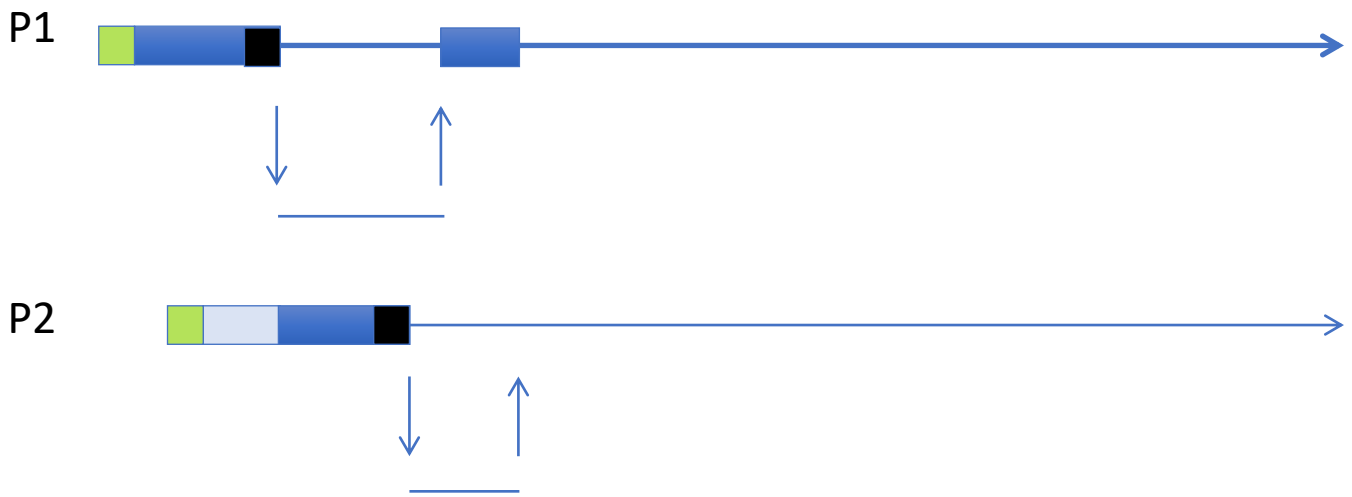
Multiprocess System

Multiprocess System

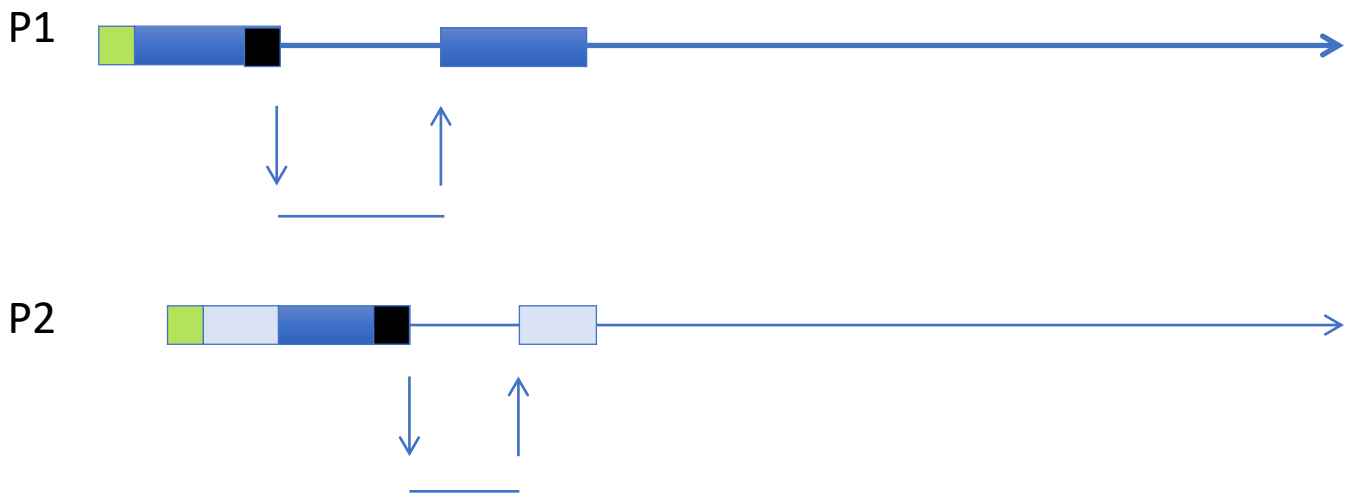


Multiprocess System

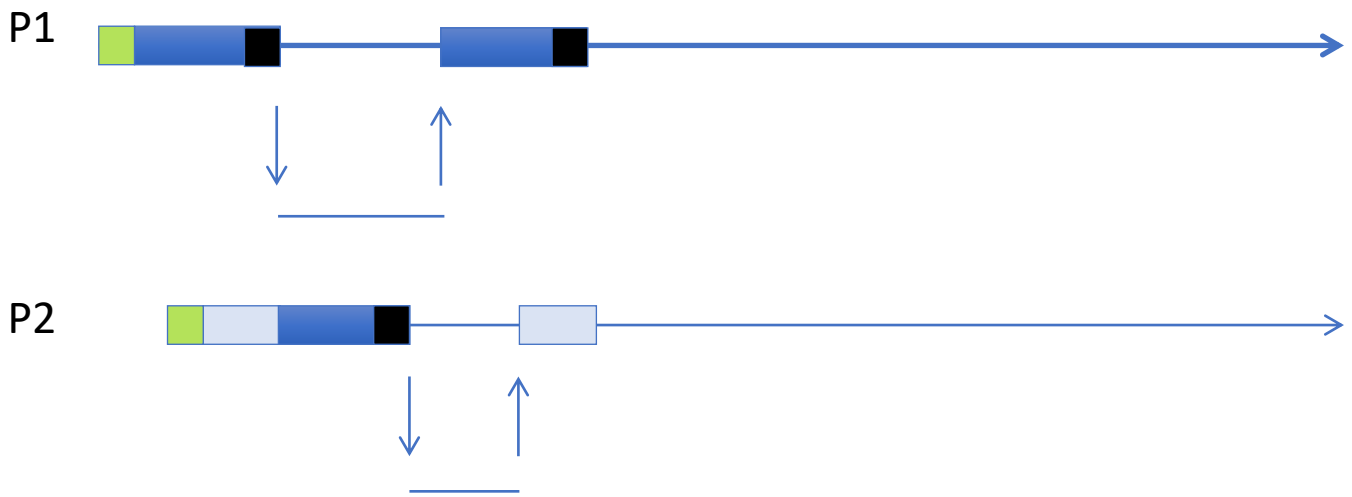
Multiprocess System



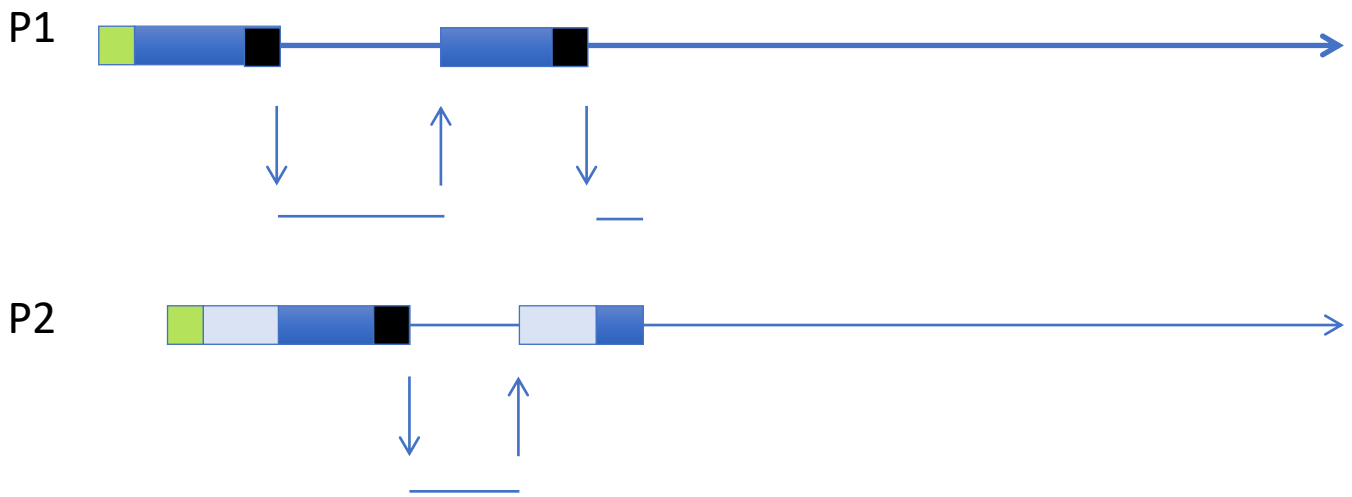
Multiprocess System



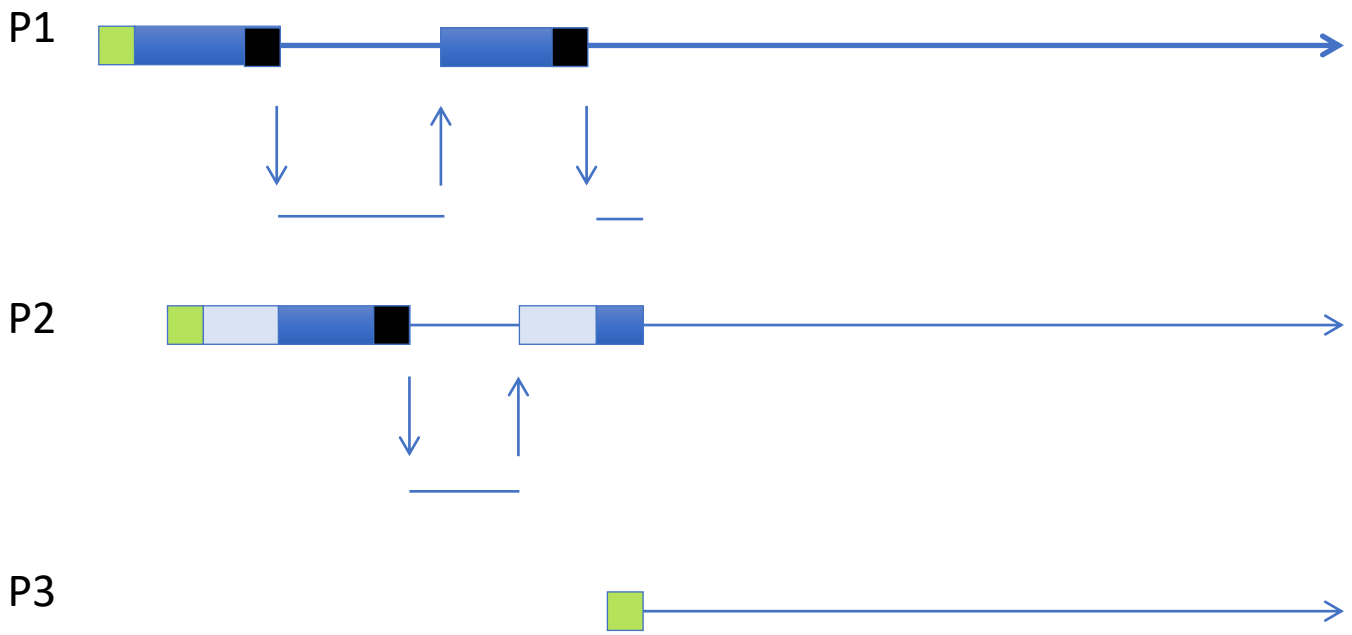
Multiprocess System



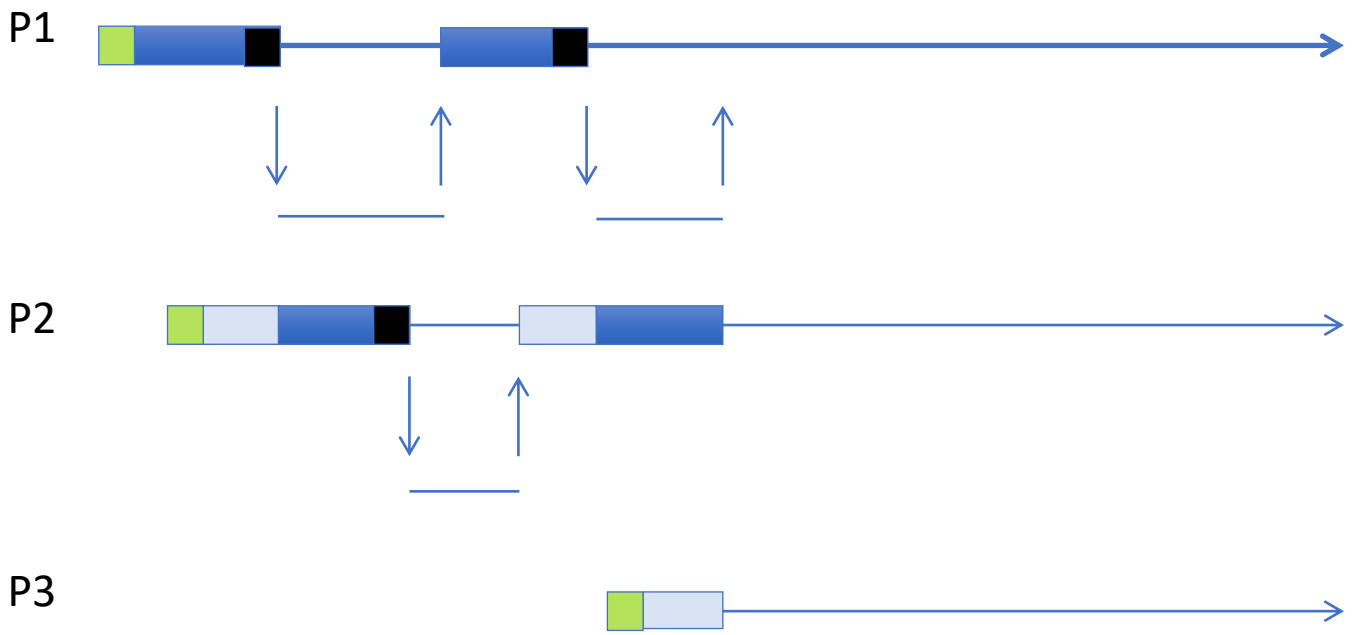
Multiprocess System



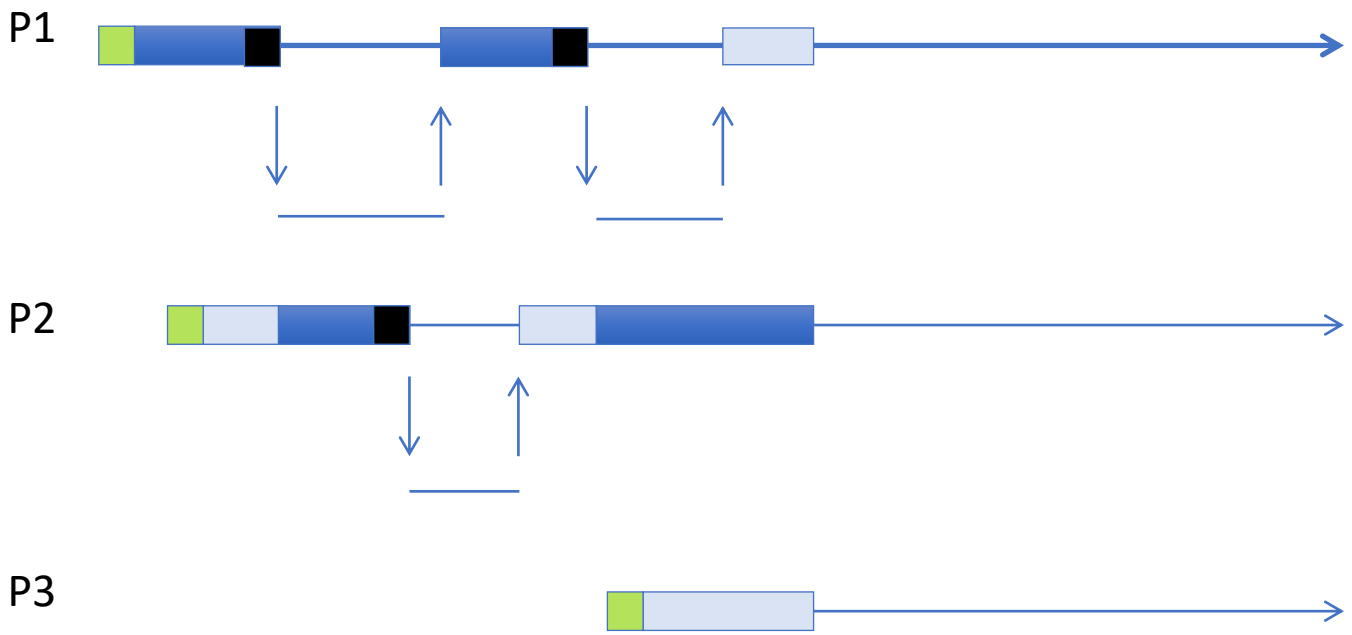
Multiprocess System



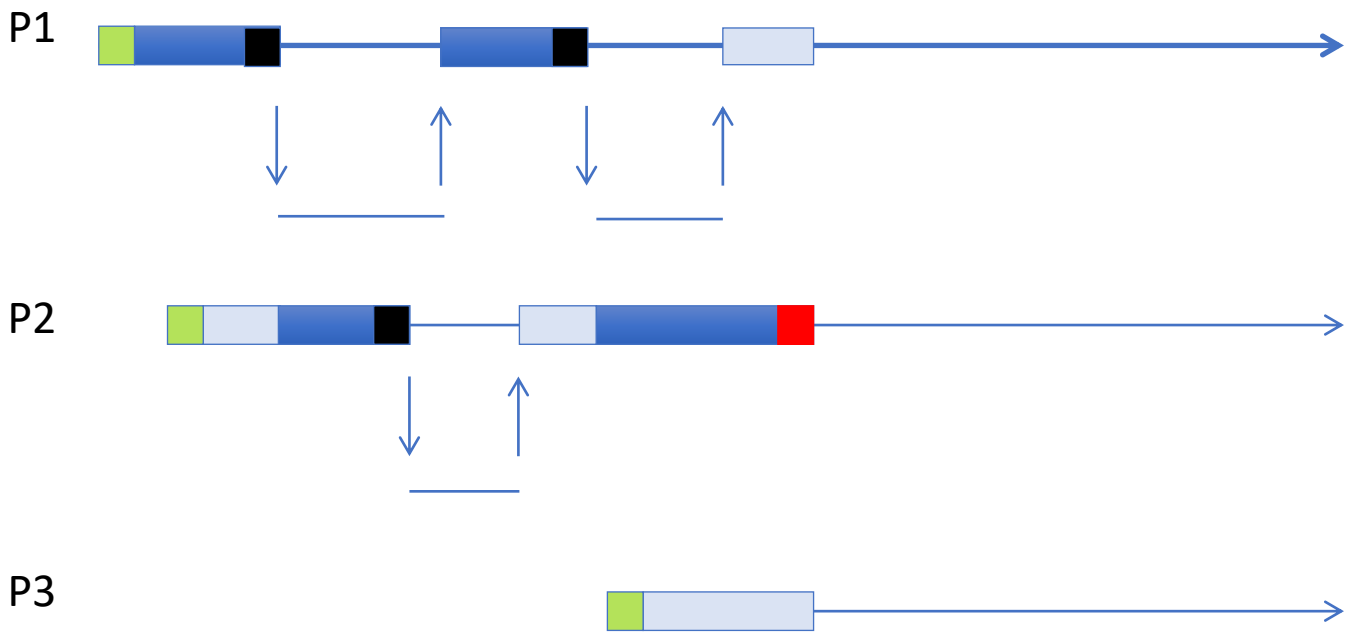
Multiprocess System



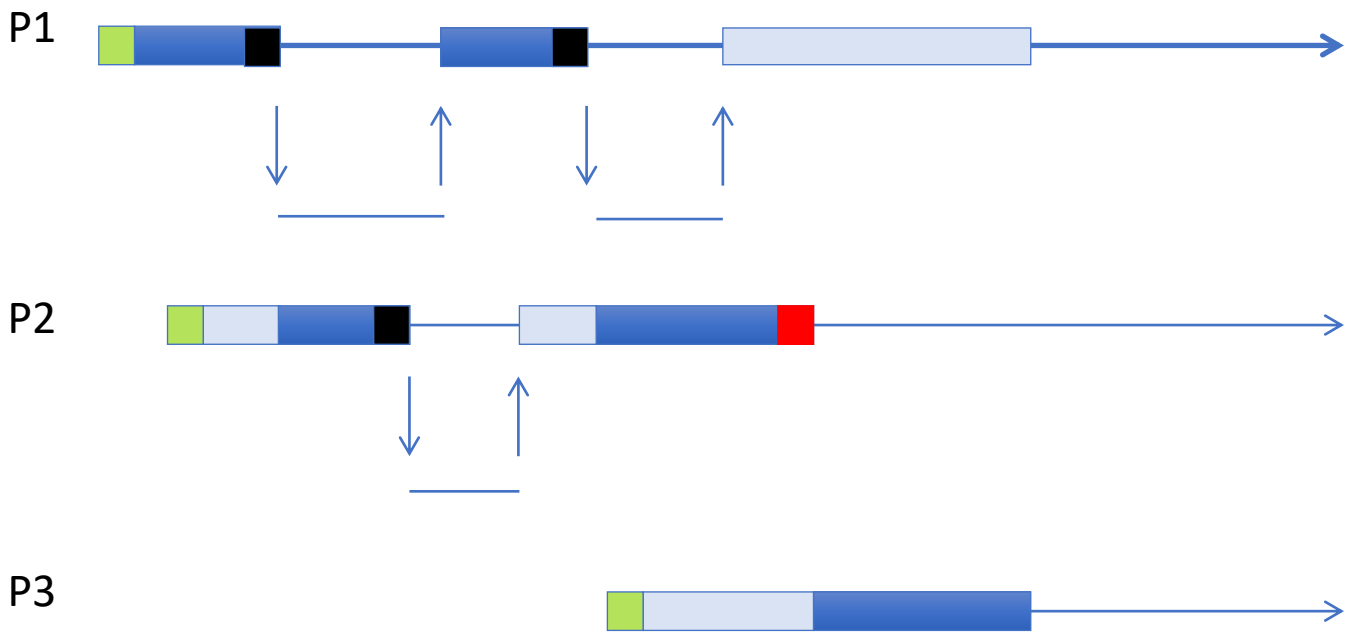
Multiprocess System



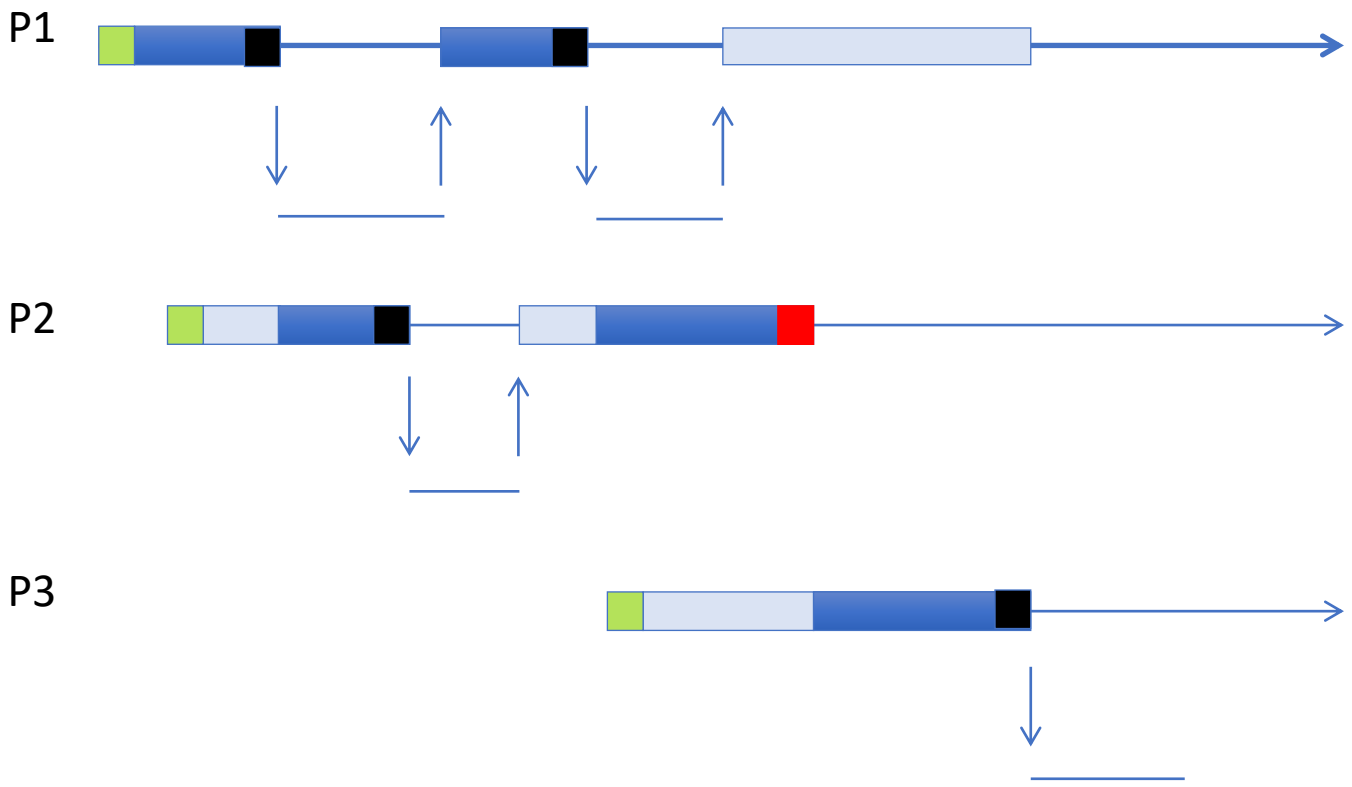
Multiprocess System



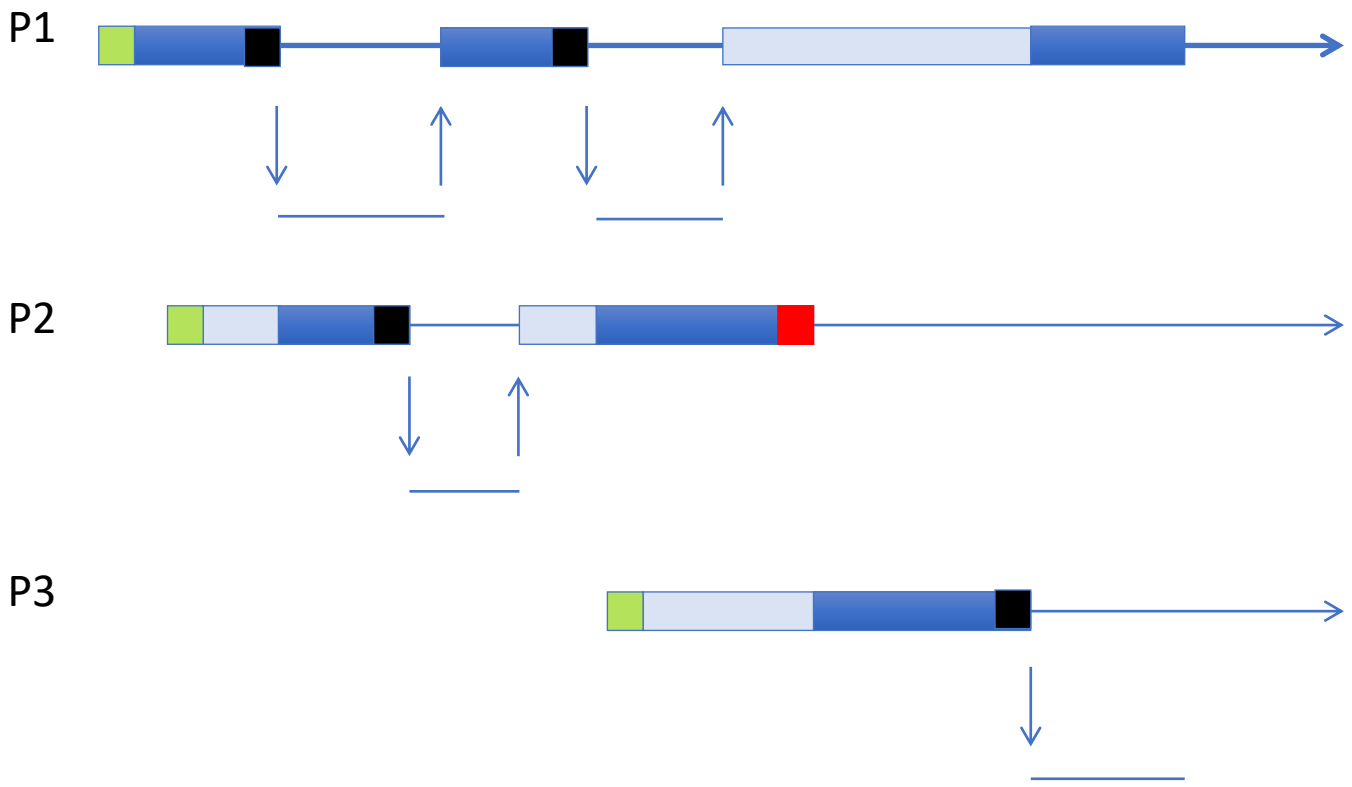
Multiprocess System



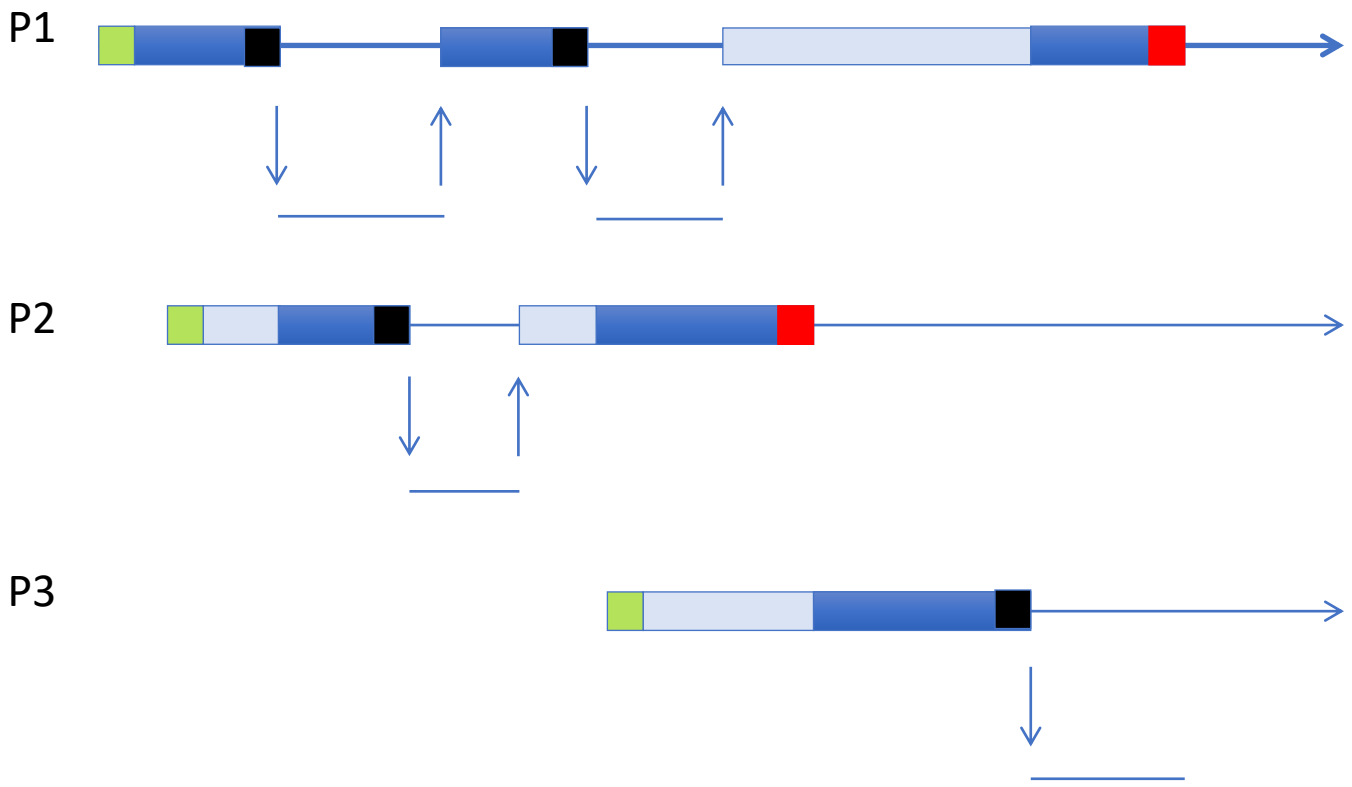
Multiprocess System



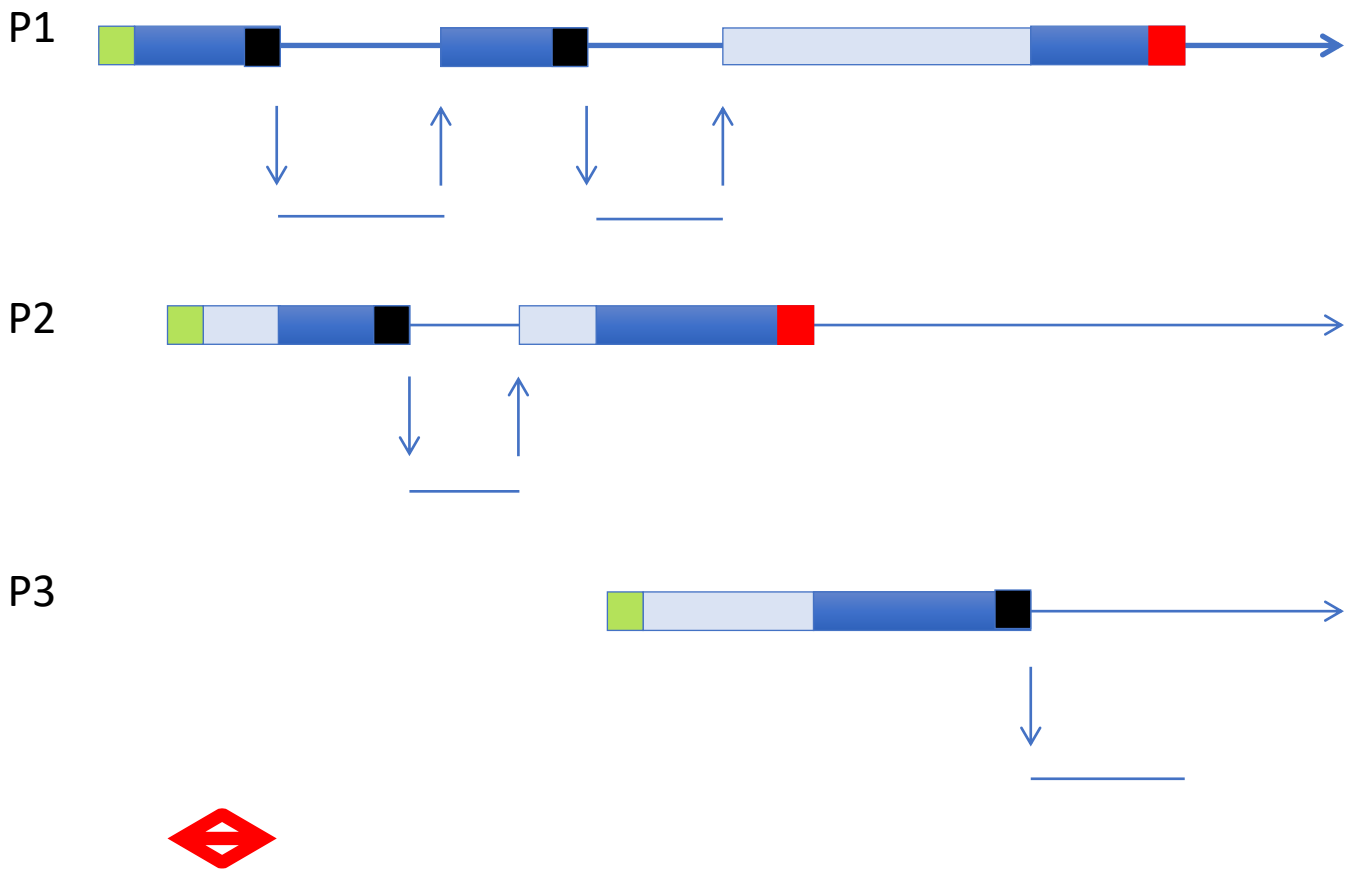
Multiprocess System



Multiprocess System

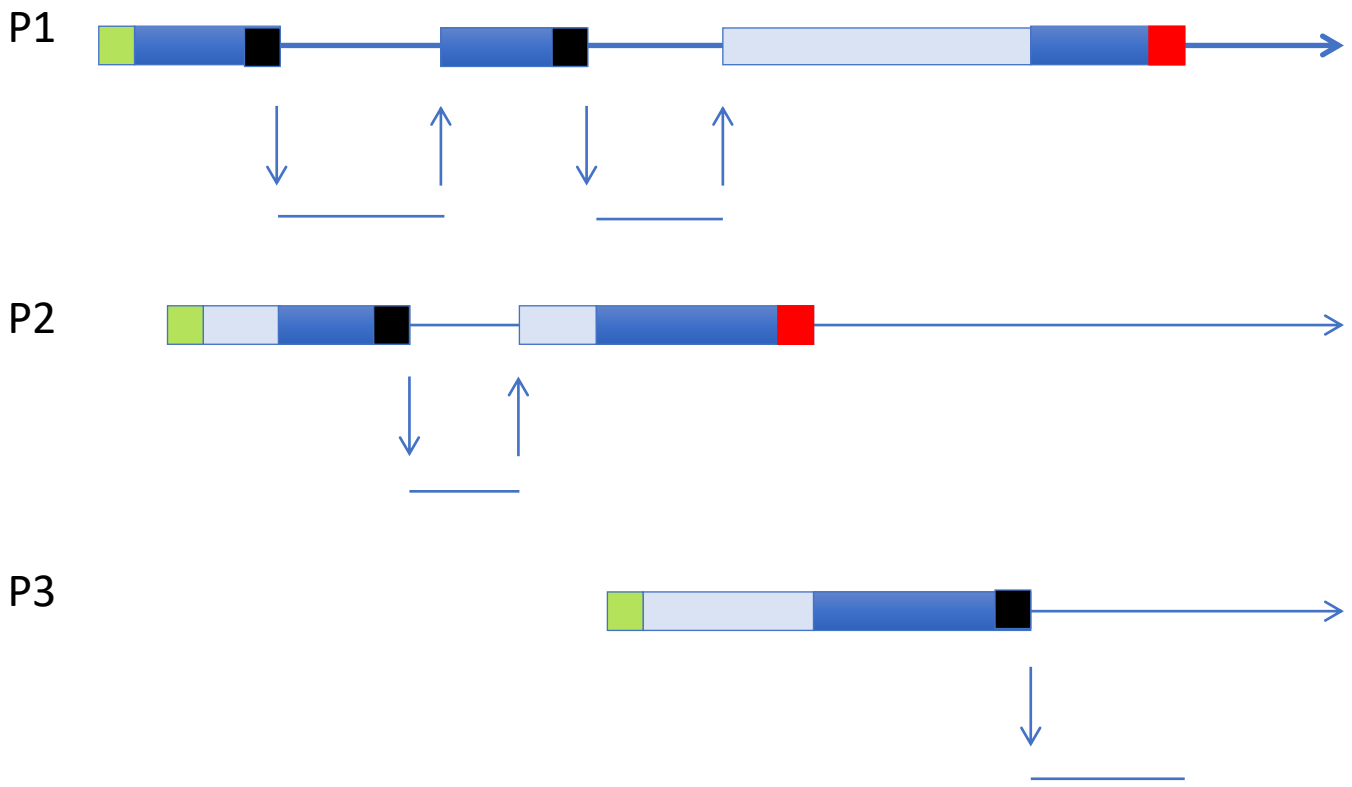


Multiprocess System



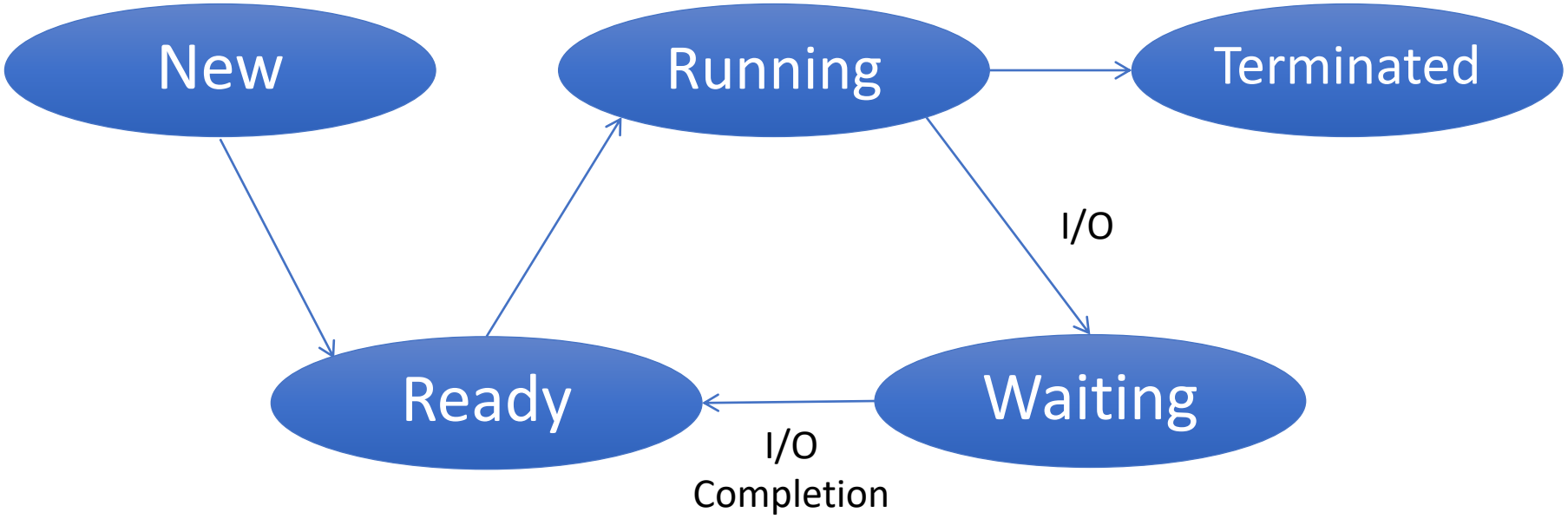
short wait time

Multiprocess System



high utilization (short CPU idle times)

Process State Diagram for Multiprocessing System



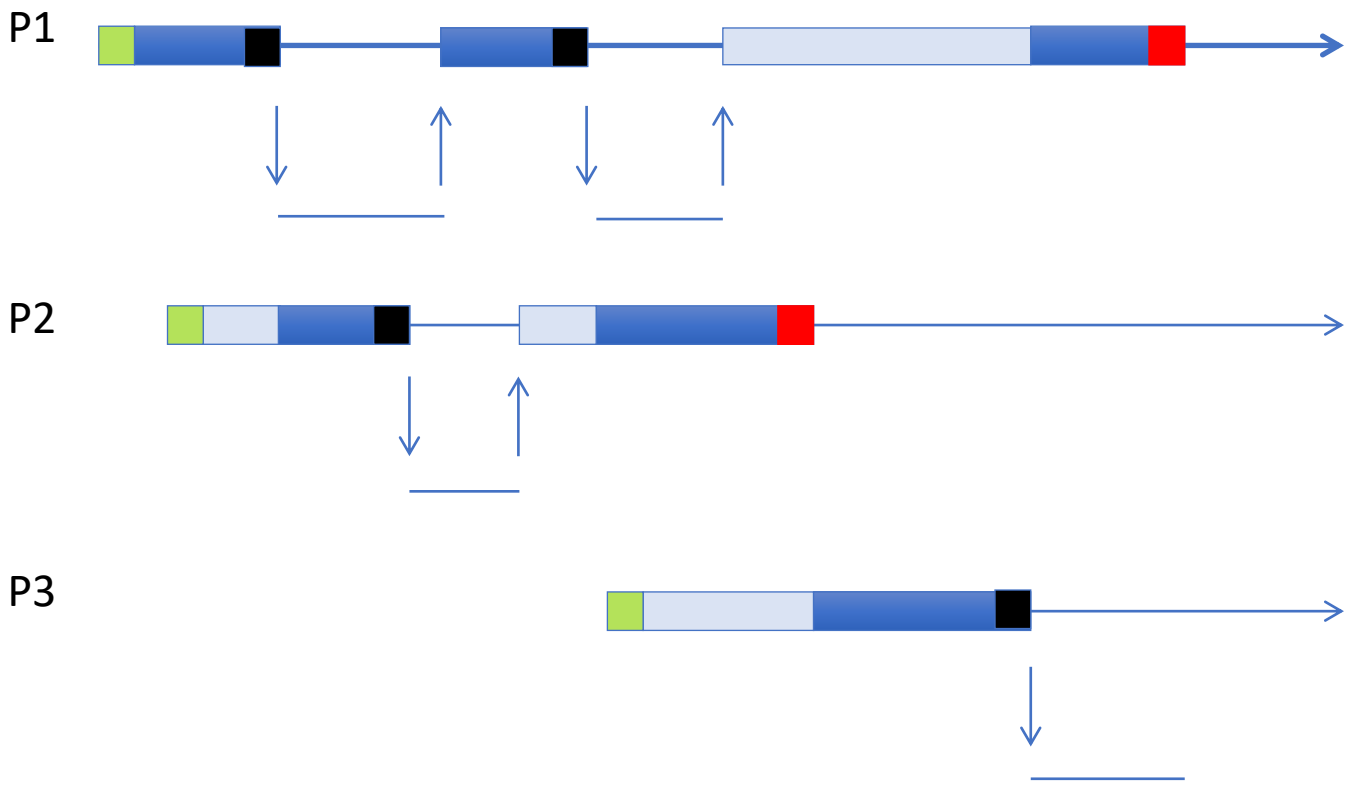
Two Important Concepts

- Process switch
- Process scheduling

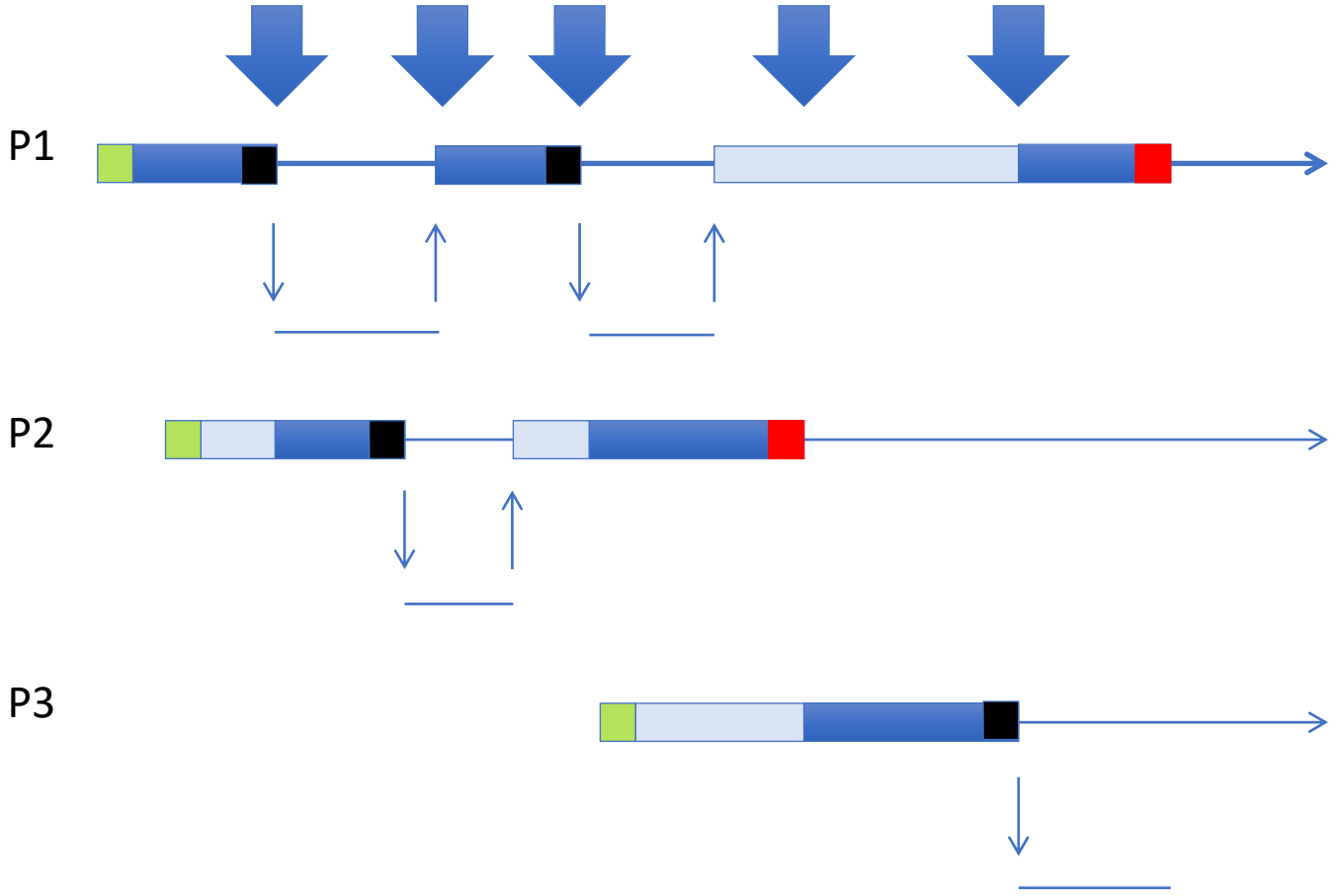
Process Switch

- Switch from one process running on the CPU to another process
- Such that you can later switch back to the process currently holding the CPU

Process Switch – where do we switch?



Answer: Process Switch



Process Switch Implementation

- Process consists of:
  - Code (including libraries)
  - Stack
  - Heap
  - Registers (including PC)
  - MMU info (ignore for now)

Process Switch Implementation

- Process:
  - Code
  - Stack
  - Heap
  - Registers
  - MMU info

Resides in process-private locations

Resides in shared locations
(ignore for now)

Process Switch P1 ➔ P2

- Save registers(P1) to somewhere
- Restore registers(P2) from somewhere

- Where to save to and restore from?

Process Control Block

- Kernel must remember processes
- Each process has a process control block (PCB)
- Process control block contains
  - Process identifier (unique id)
  - Process state
  - *Space to support process switch (save area)*
- Process Control Block Array
  - Indexed by hash(pid)

Process Switch P1 ➔ P2

- Save registers ➔ PCB[P1].SaveArea
- Restore PCB[P2].SaveArea ➔ registers

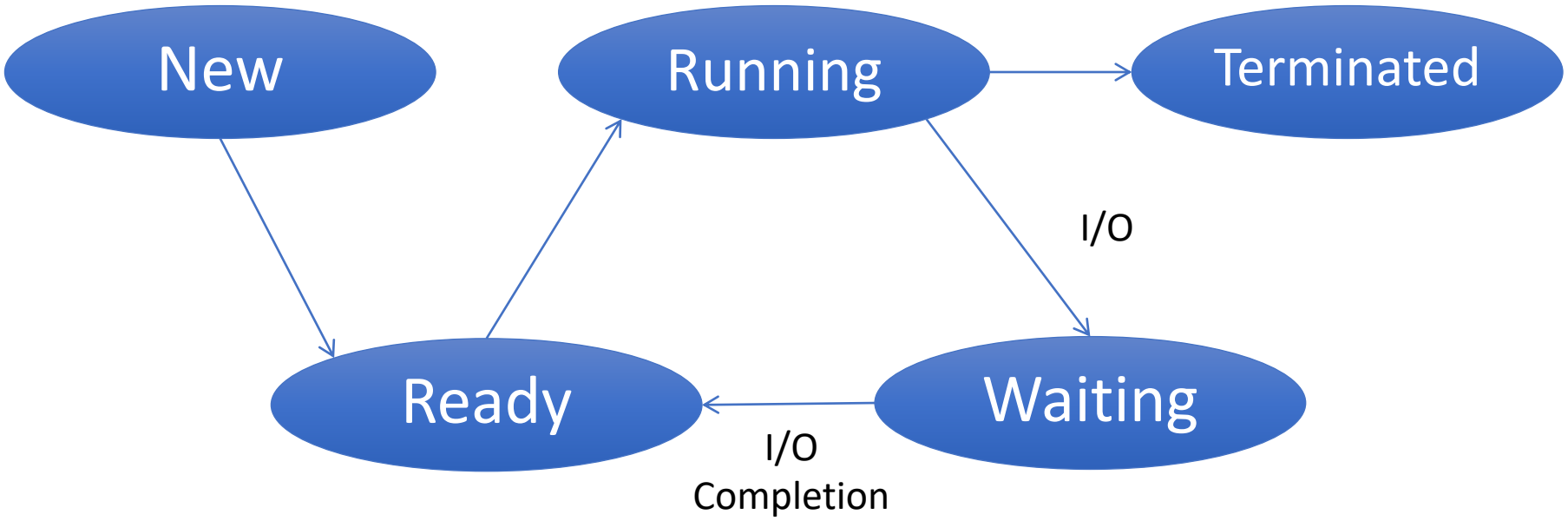
Process Switch - Caveat

- A process switch is an expensive operation!
- Requires saving and restoring lots of stuff
  - Not just registers
  - Also MMU information
- Has to be implemented very efficiently
- Has to be used with care

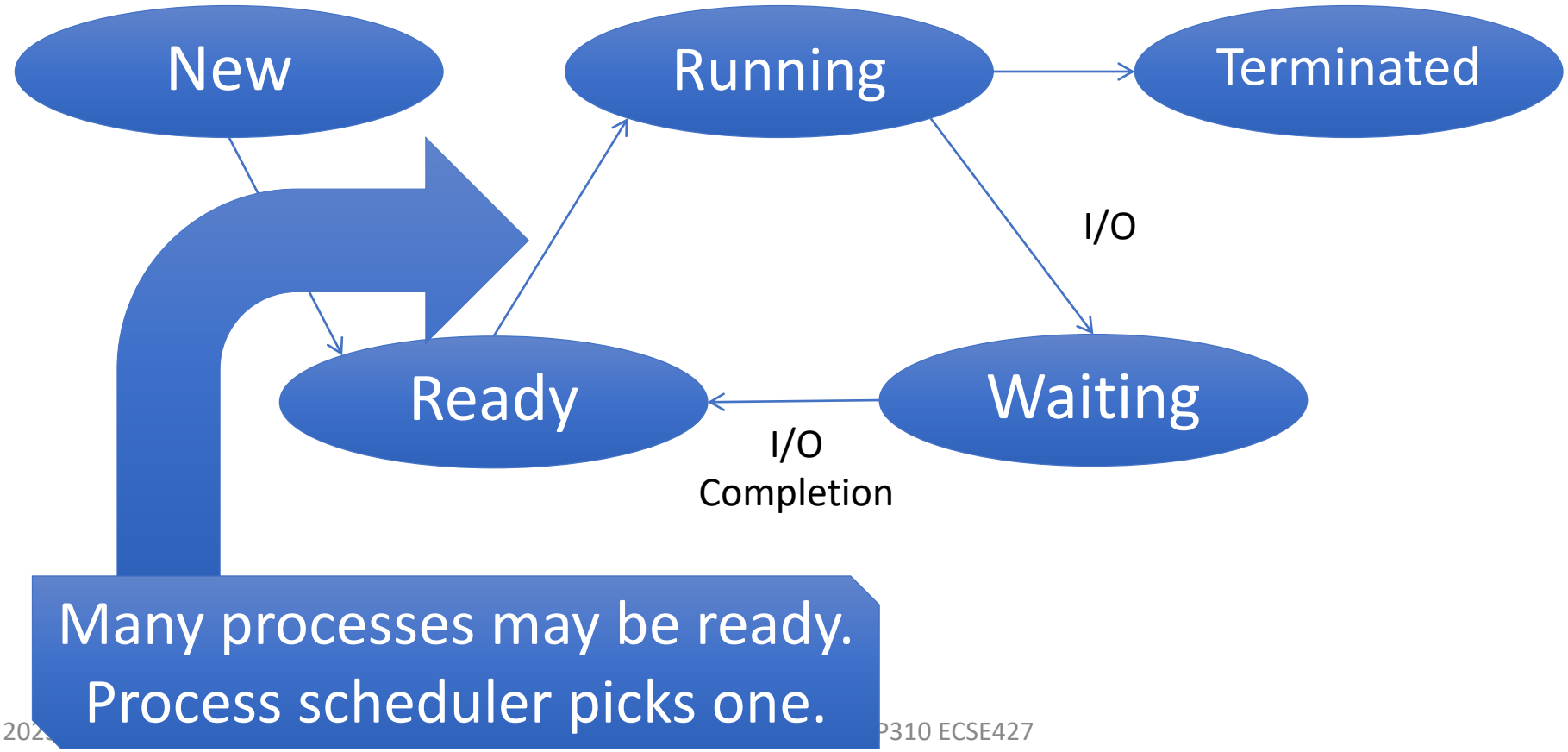
Two Important Concepts

- Process switch
- ***Process scheduling***

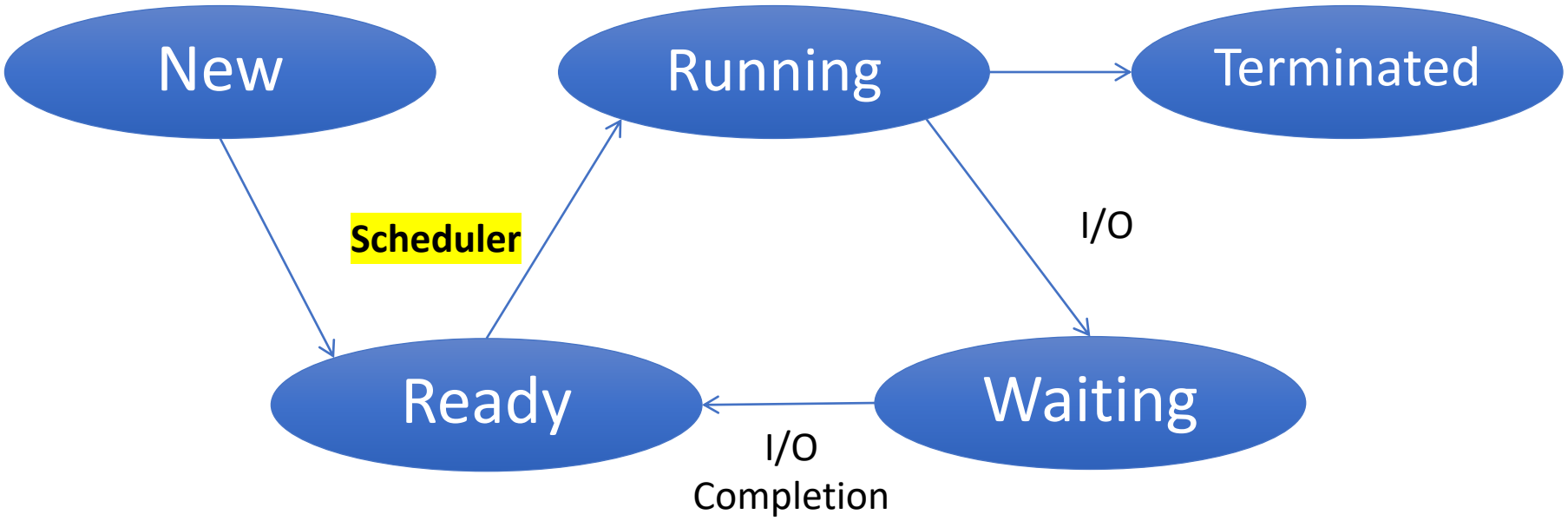
Process Scheduling



Process Scheduling



Process Scheduling

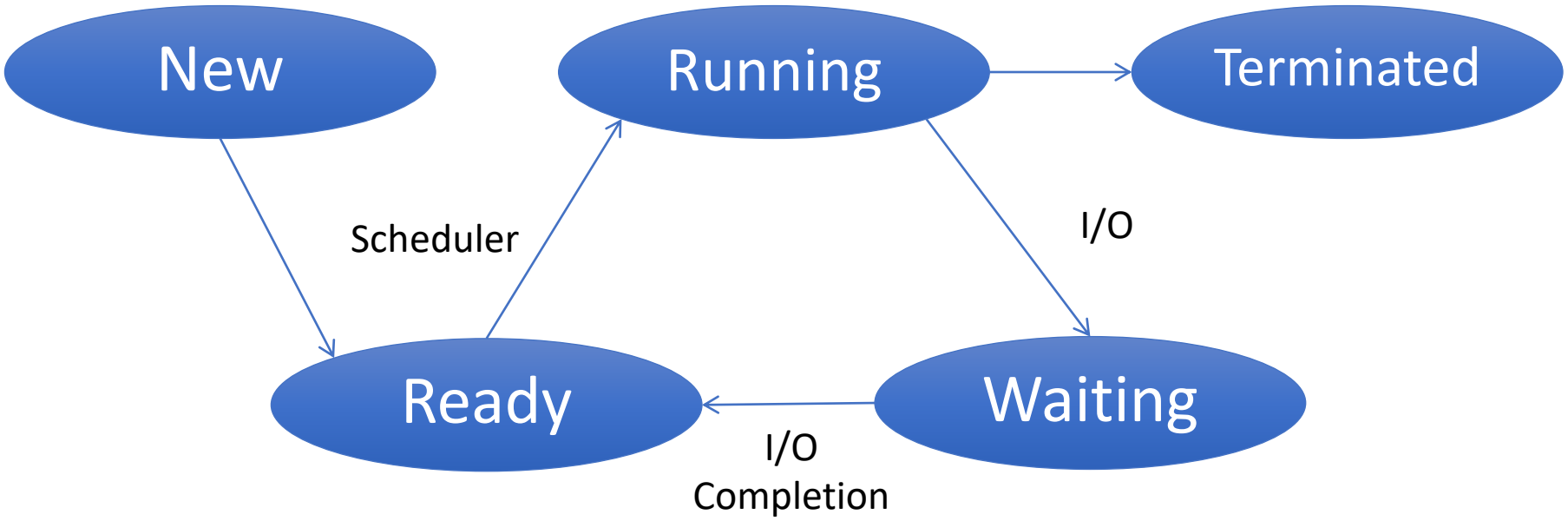


What is the role of the scheduler in an OS?

- Think of scheduler as managing a queue
- Process ready: insert it into queue
  - According to scheduling policy
- Scheduling decision: run head of queue

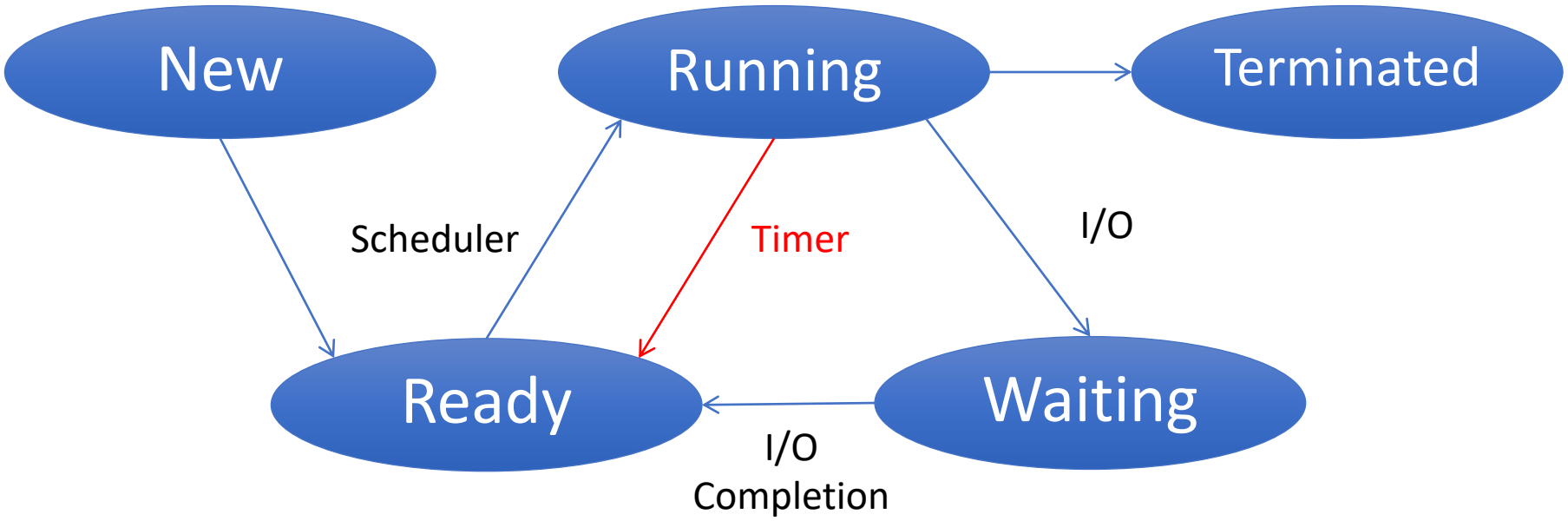
- Not always implemented this way!!

Problem



A process could run forever, locking all other processes out

Solution


New
Running
Terminated
Scheduler
Timer
I/O
Ready
Waiting
I/O
Completion

Preemptive vs Non-preemptive Scheduler

- Non-preemptive:
  - Process only voluntarily relinquishes CPU
- Preemptive
  - Process may be forced off CPU

Advantages - Disadvantages

- Non-preemptive
- Preemptive
- Intermediate solutions are possible

Advantages - Disadvantages

- Non-preemptive
  - Process can monopolize CPU
  - Only useful in special circumstances
- Preemptive
  - Process can be thrown out at any time
  - Usually not a problem, but sometimes it is
- Intermediate solutions are possible

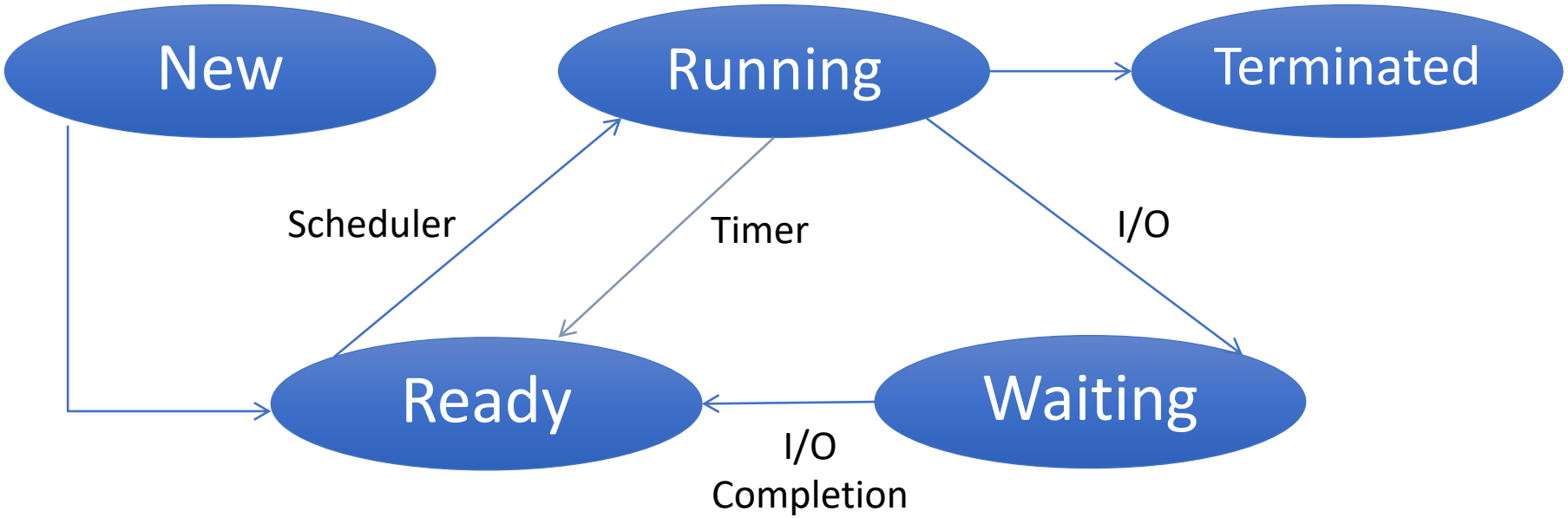
Process Scheduling Implementation

- Remember running process
- Maintain sets of queues
  - (CPU) ready queue
  - I/O device queue (one per device)
- PCBs sit in queues

How does the Scheduler run?

- Scheduler is part of the kernel
- How does kernel run?

How does Scheduler run?



The scheduler runs when

1) process starts or terminates (system call)
2) running process performs an I/O (system call)
3) I/O completes (I/O interrupt)
4) timer expires (timer interrupt)

How does the Scheduler Run?

- At end of handlers for
  - System calls
  - Interrupts
  - Traps
- Scheduler runs: decides on process to run
- Switches to a new process
- Sets another timer

Scheduling Algorithm

- Decides which ready process gets to run

What makes a good scheduling algorithm?

What makes a good scheduling algorithm?

- It depends …

Scheduling Performance Metrics

- Minimize turnaround time
  - Do not want to wait long for job to complete
  - Completion_time – arrival_time
- Minimize response time
  - Schedule interactive jobs promptly so users see output quickly
  - Initial_schedule_time – arrival_time
- Minimize waiting time
  - Do not want to spend much time in Ready queue
- Maximize throughput
  - Want many jobs to complete per unit of time
- Maximize resource utilization
  - Keep expensive devices busy
- Minimize overhead
  - Reduce number of context switches
- Maximize fairness
  - All jobs get same amount of CPU over some time interval

Answer: Scheduling Performance Metrics

- Minimize turnaround time
  - Do not want to wait long for job to complete
  - Completion_time – arrival_time
- Minimize response time
  - Schedule interactive jobs promptly so users see output quickly
  - Initial_schedule_time – arrival_time
- Minimize waiting time
  - Do not want to spend much time in Ready queue
- Maximize throughput
  - Want many jobs to complete per unit of time
- Maximize resource utilization
  - Keep expensive devices busy
- Minimize overhead
  - Reduce number of context switches
- Maximize fairness
  - All jobs get same amount of CPU over some time interval

Conflicting goals

→ A good scheduling algorithm depends on what we want to run

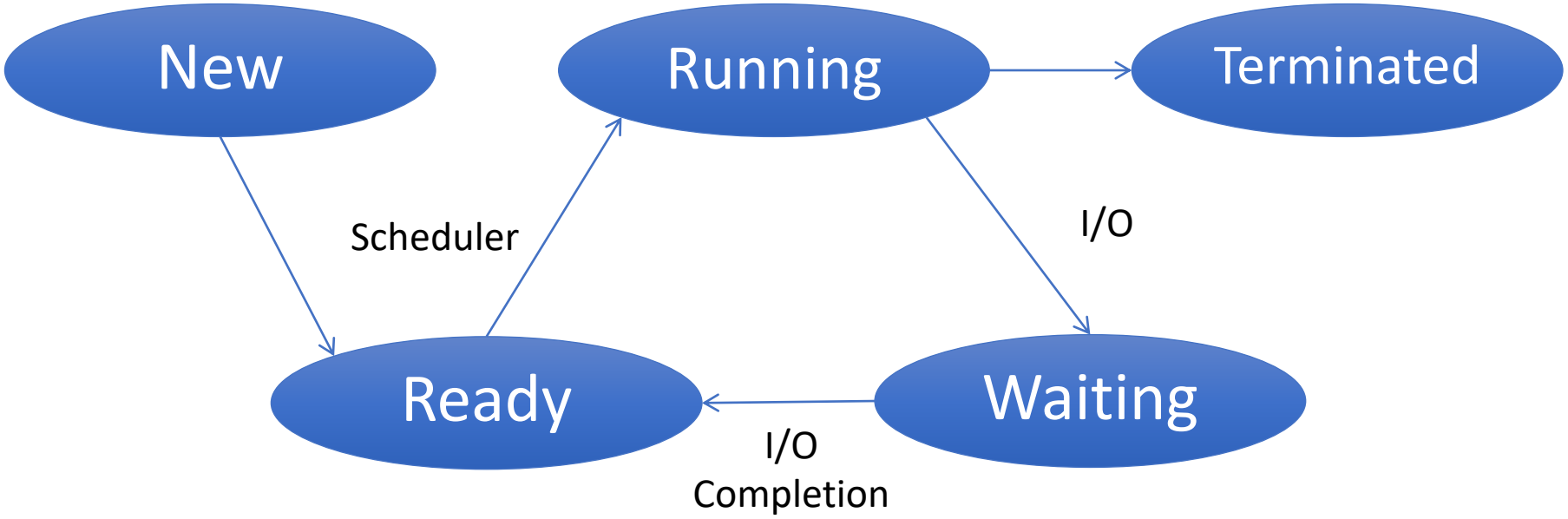
→ Most of the time, scheduler does not know in advance the job type and its needs

Question: Interactive vs. Batch Jobs

What makes a good scheduler for interactive/for batch?

- Interactive = you are waiting for the result
  - E.g., browser, editor, …
  - Tend to be short
- Batch = you will look at result later
  - E.g., supercomputing center, offline analysis, …
  - Tend to be long

Question: Interactive vs. Batch Jobs



Answer: What makes a good scheduler for interactive?

- Short response time
- Response time = wait from ready → running
  - Initial_schedule_time – arrival_time

Answer: What makes a good scheduler for batch?

- High throughput
- Throughput = number of jobs completed
  - Minimize scheduling overhead
    - Reduce number of ready → running switches

Issue: Response Time vs. Throughput

- Conflicting goals
- From throughput perspective
  - Scheduler is overhead
  - Run scheduler as little as possible
- From response time perspective
  - Want to go quickly from ready to running
  - Run scheduler often

Scheduling policies

- First come, first served (FCFS)
- Shortest job first (SJF)
- Round robin (RR)

First come first served (FCFS)

- Process ready: insert at tail of queue
- Head of queue: “oldest” ready process
- By definition, non-preemptive

Question

- Advantages and disadvantages of FCFS?

Answer: FCFS advantages and disadvantages

- Low overhead – few scheduling events
- Good throughput
- Uneven response time – stuck behind long job
- Extreme case – process monopolizes CPU

Shortest job first (SJF)

- Process ready
  - Insert in queue according to length
- Head of queue: “shortest” process
- Can be preemptive or non-preemptive
- From now on, only consider preemptive

Question

- Advantages and disadvantages of SJF?

Answer: SJF advantages and disadvantages

- Good response time for short jobs
- Can lead to starvation of long jobs
- Difficult to predict job length

Round Robin (RR)

- Define time quantum Δ
- Process ready: put at tail of queue
- Head of queue: run for Δ time
- After Δ
  - Put running process at the tail of the queue
  - Re-schedule

Question

- Advantages and disadvantages of RR?

Answer: RR advantages and disadvantages

- Good compromise for long and short jobs
- Short jobs finish quickly (a few rounds)
- Long jobs are not postponed forever

- No need to know job length
  - Discover length by how many Δ's it needs

RR Issue – How to pick Δ

- Too small
  - Many scheduling events
  - Good response time
  - Low throughput
- Too large
  - Few scheduling events
  - Good throughput
  - Poor response time
- Typical value: ~ 10 milliseconds

Scheduling Exercise

A. Describe on a timeline the order of execution of the following five processes, with arrival and execution times as shown in the table, using the following scheduling algorithms:

1. FCFS,
2. SJF – preemptive,
3. RR with a time quantum of 1.

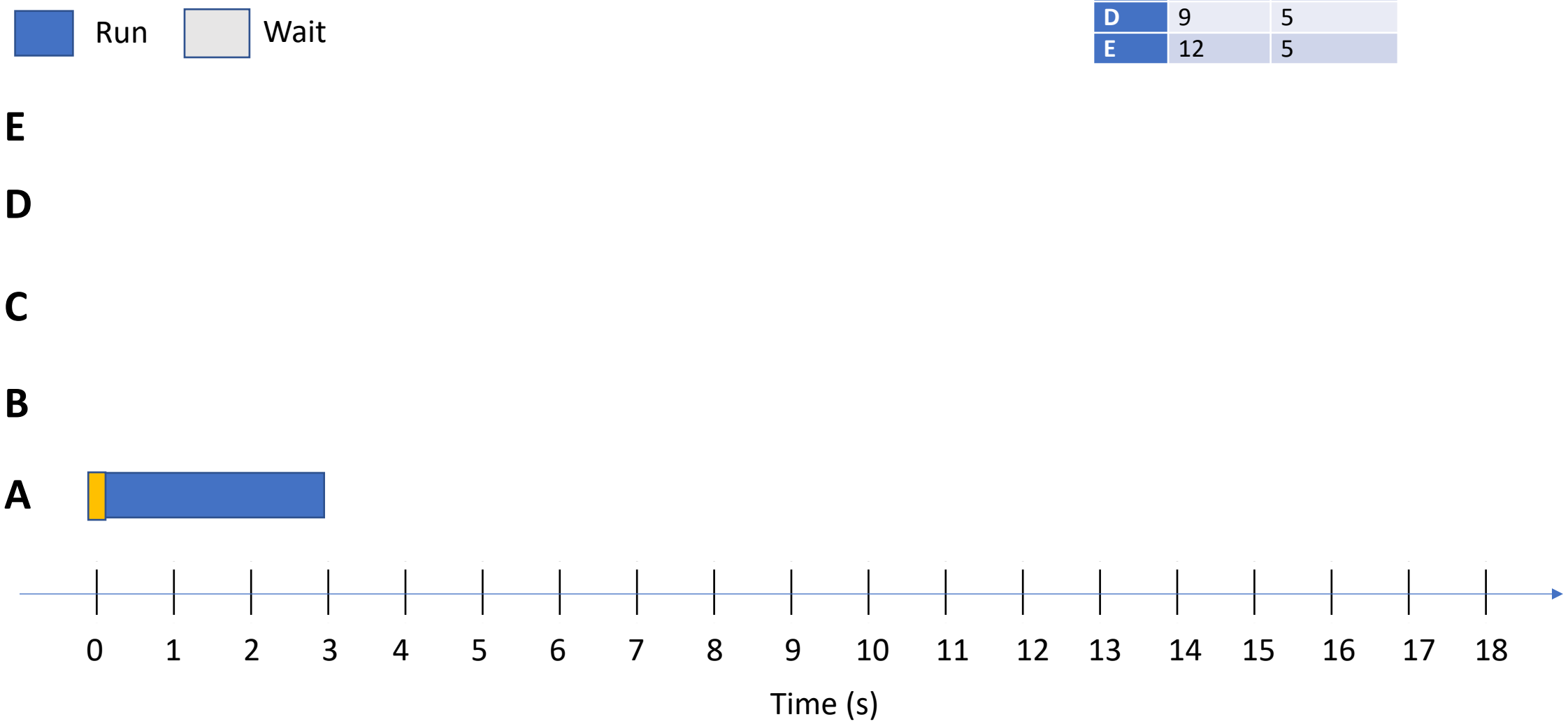
B. What is the turn-around time & response time for each process in each scenario?

Process	Arrival time	Execution time
A	0	3
B	1	5
C	3	2
D	9	5
E	12	5

FCFS

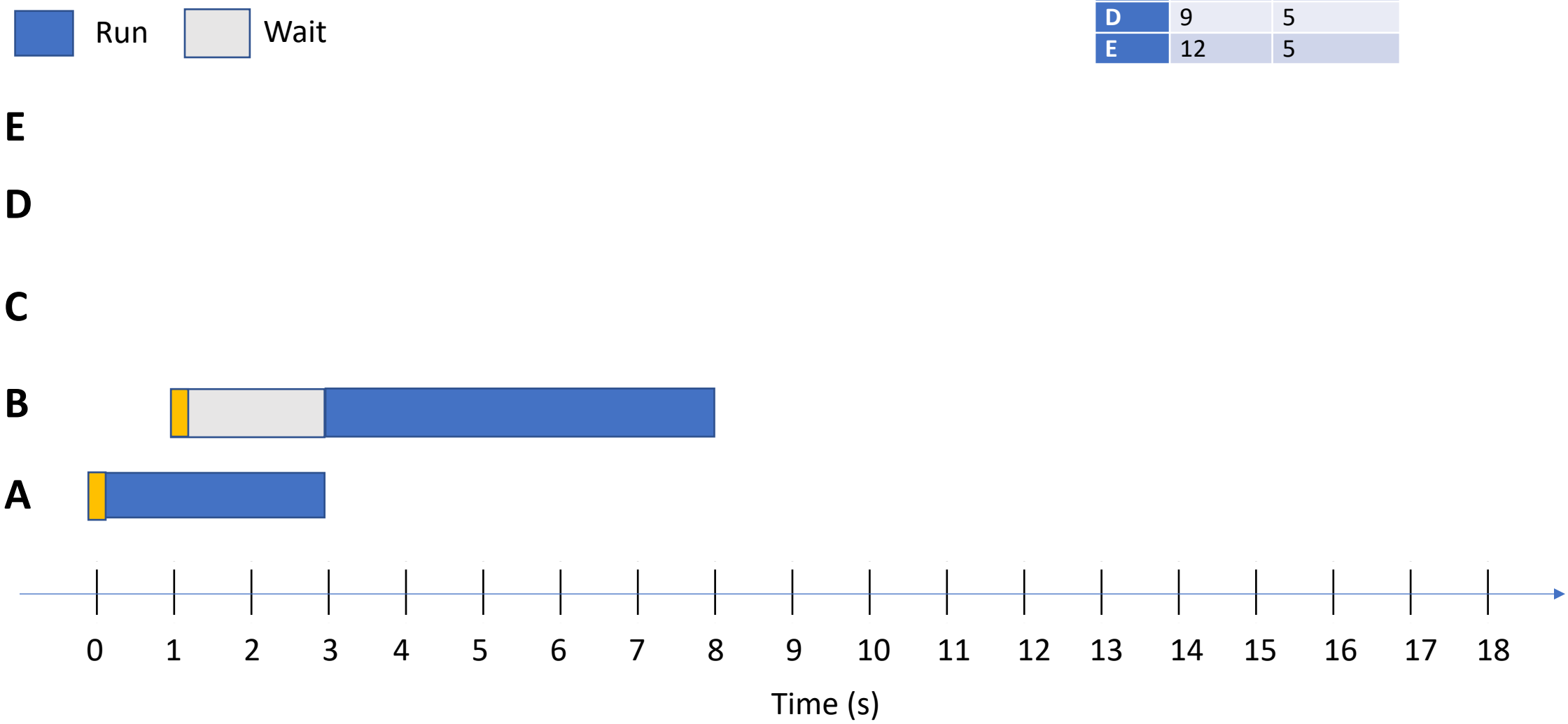
FCFS

Proc	Arrival time	Execution time
A	0	3
B	1	5
C	3	2
D	9	5
E	12	5



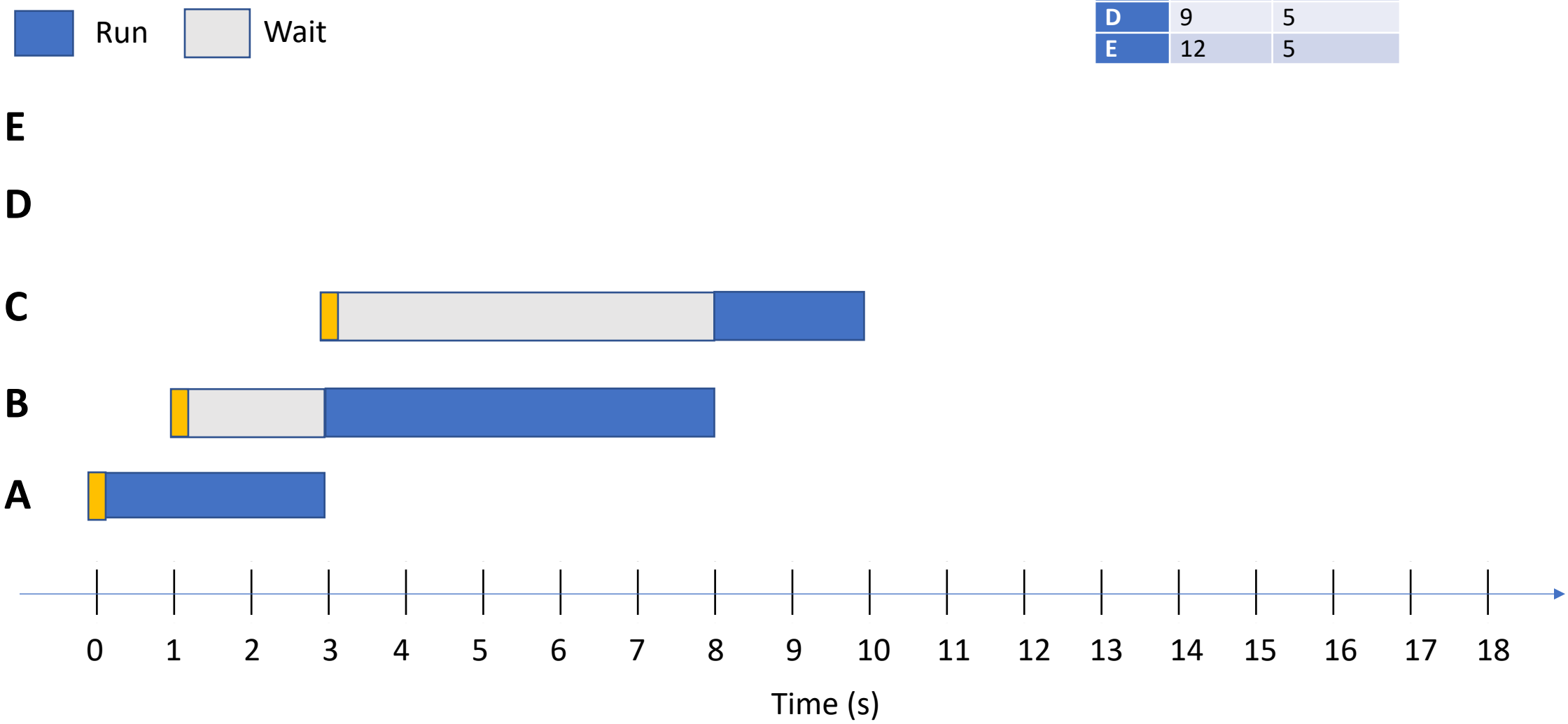
FCFS

Proc	Arrival time	Execution time
A	0	3
B	1	5
C	3	2
D	9	5
E	12	5



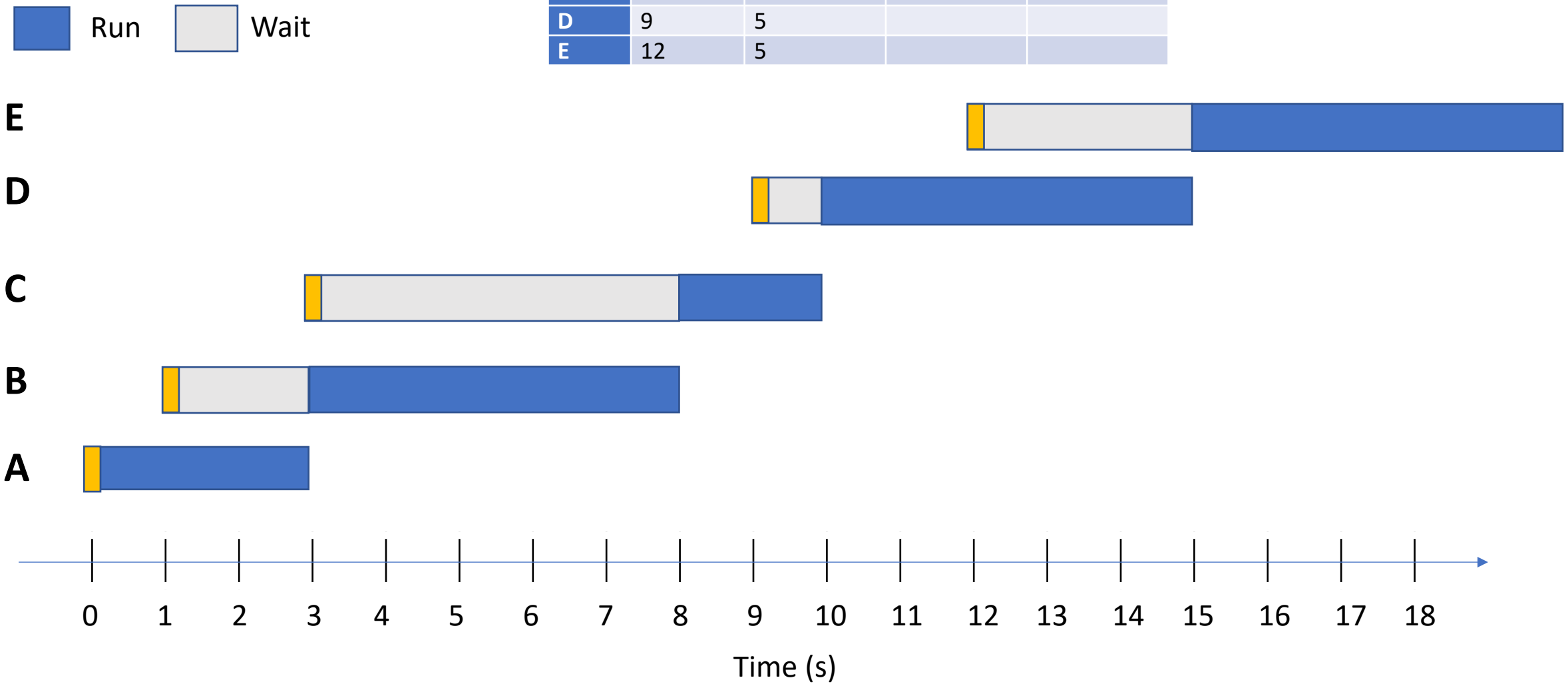
FCFS

Proc	Arrival time	Execution time
A	0	3
B	1	5
C	3	2
D	9	5
E	12	5



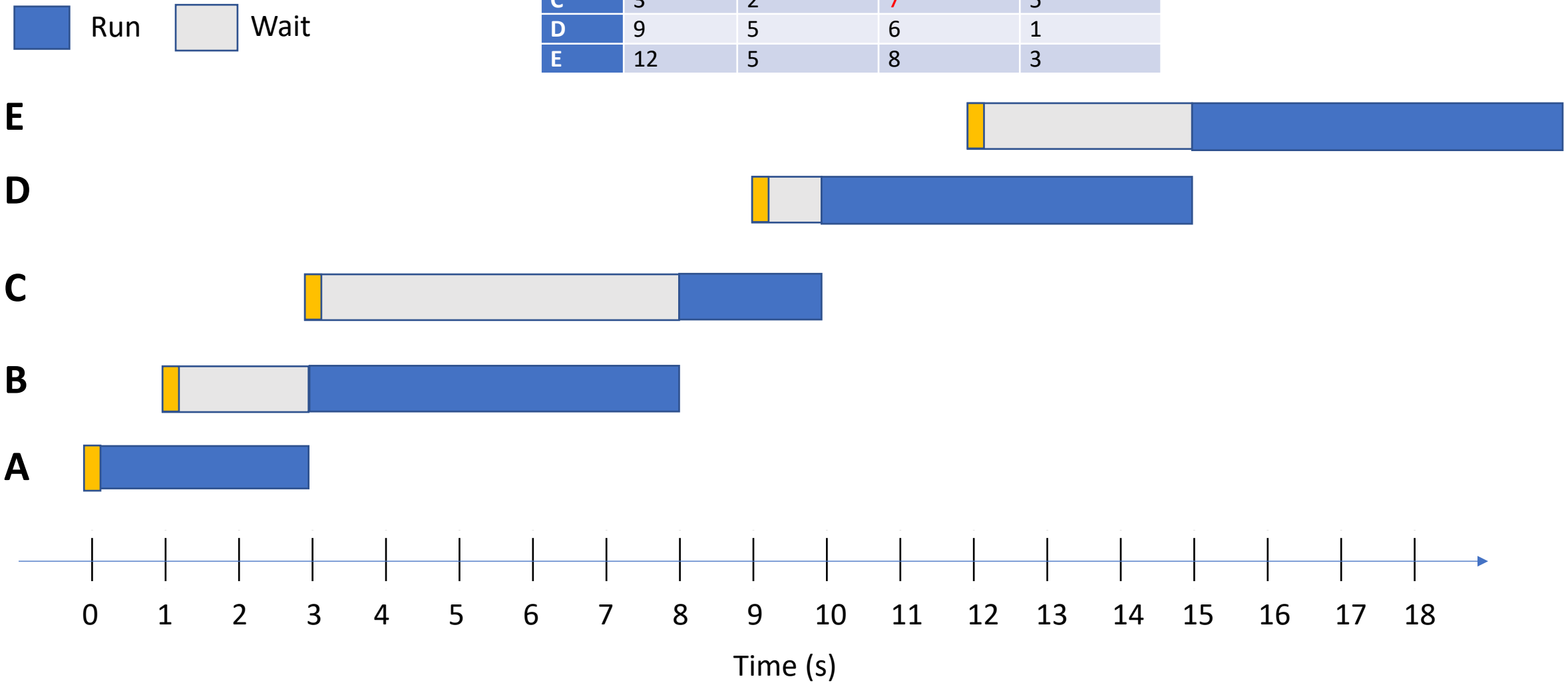
FCFS

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



FCFS

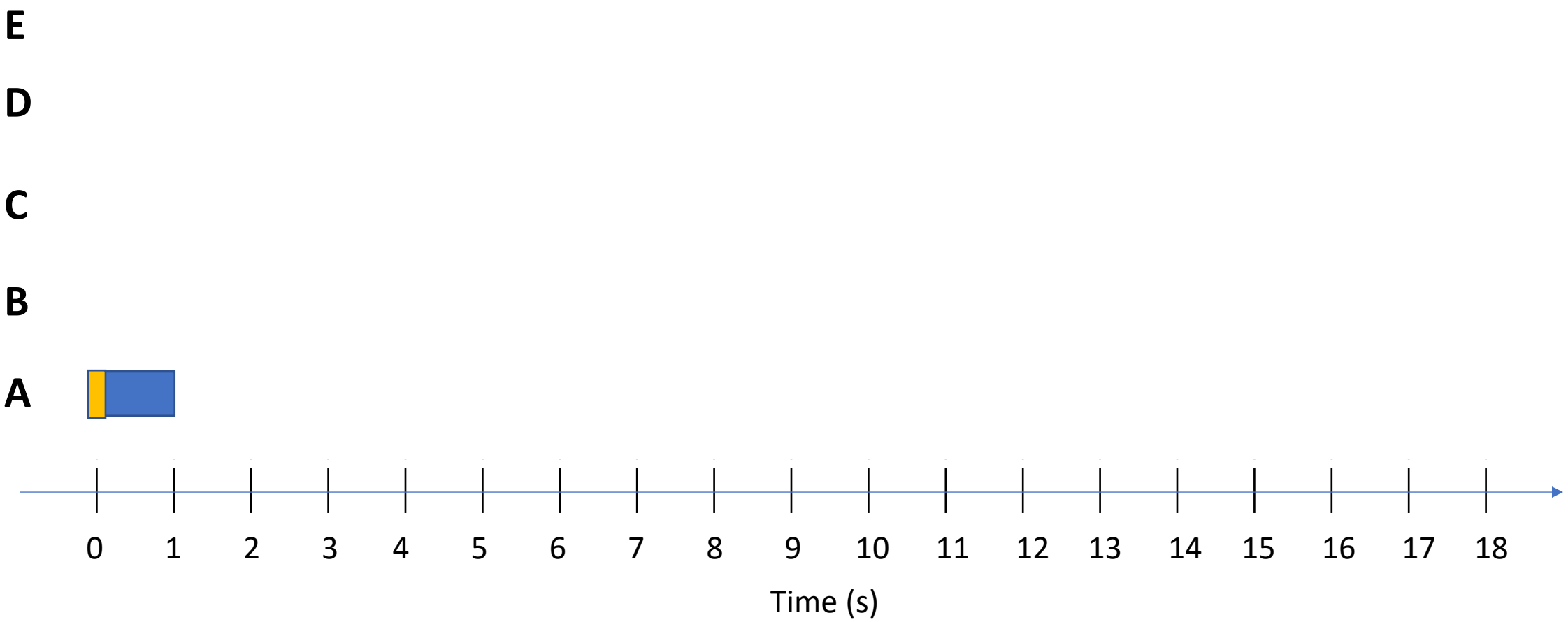
Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3	3	0
B	1	5	7	2
C	3	2	7	5
D	9	5	6	1
E	12	5	8	3



SJF – preemptive

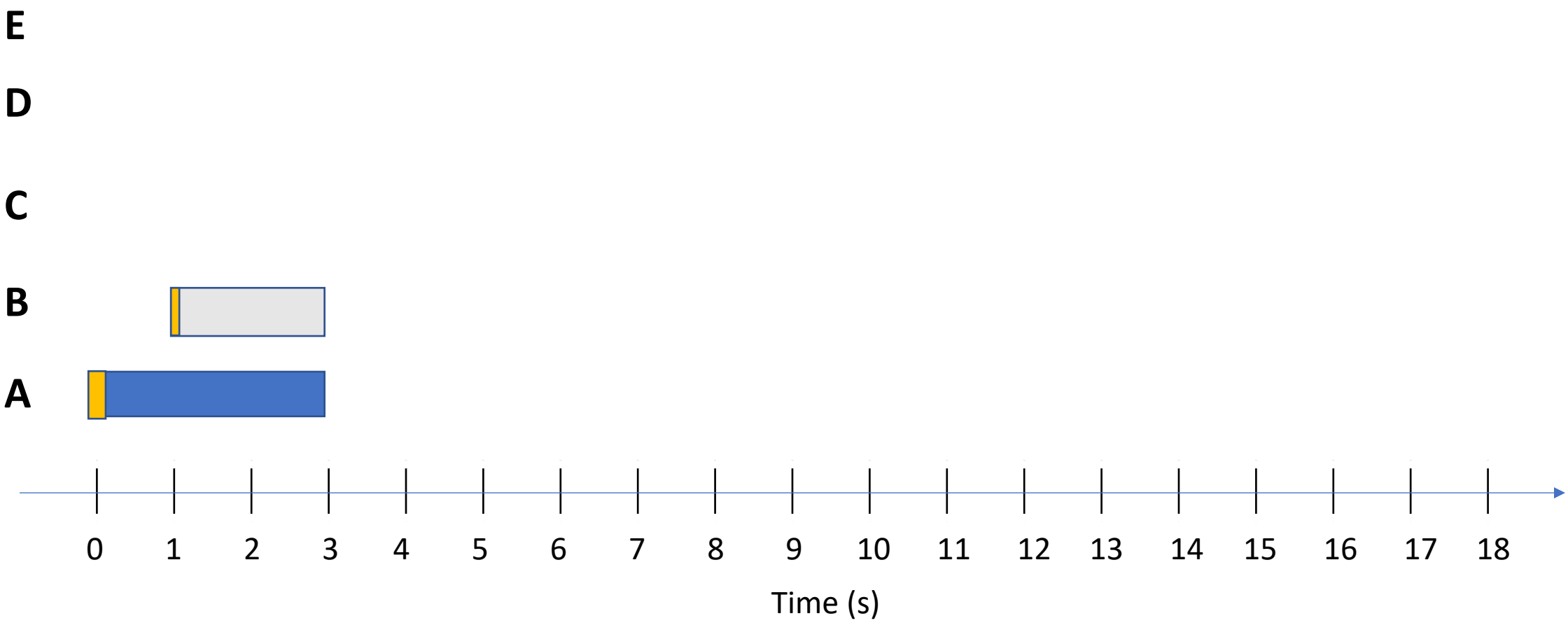
SJF - preemptive

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



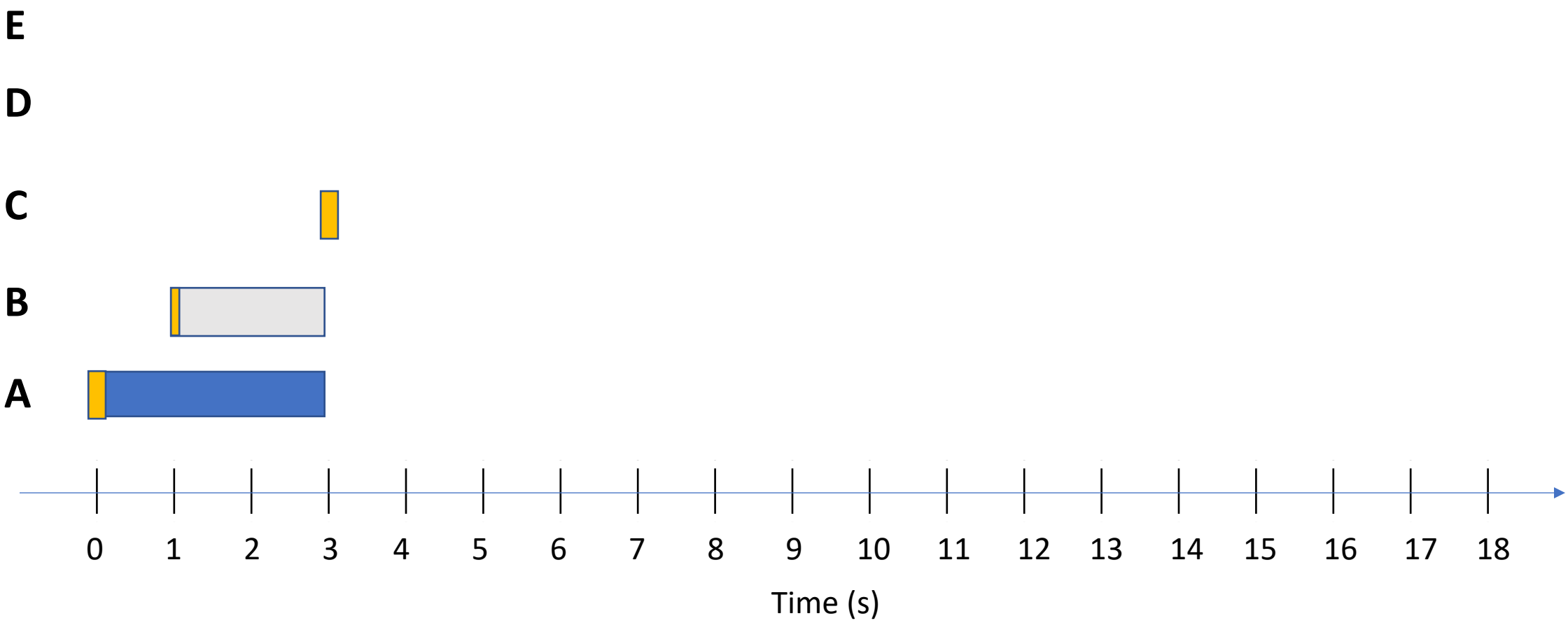
SJF - preemptive

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



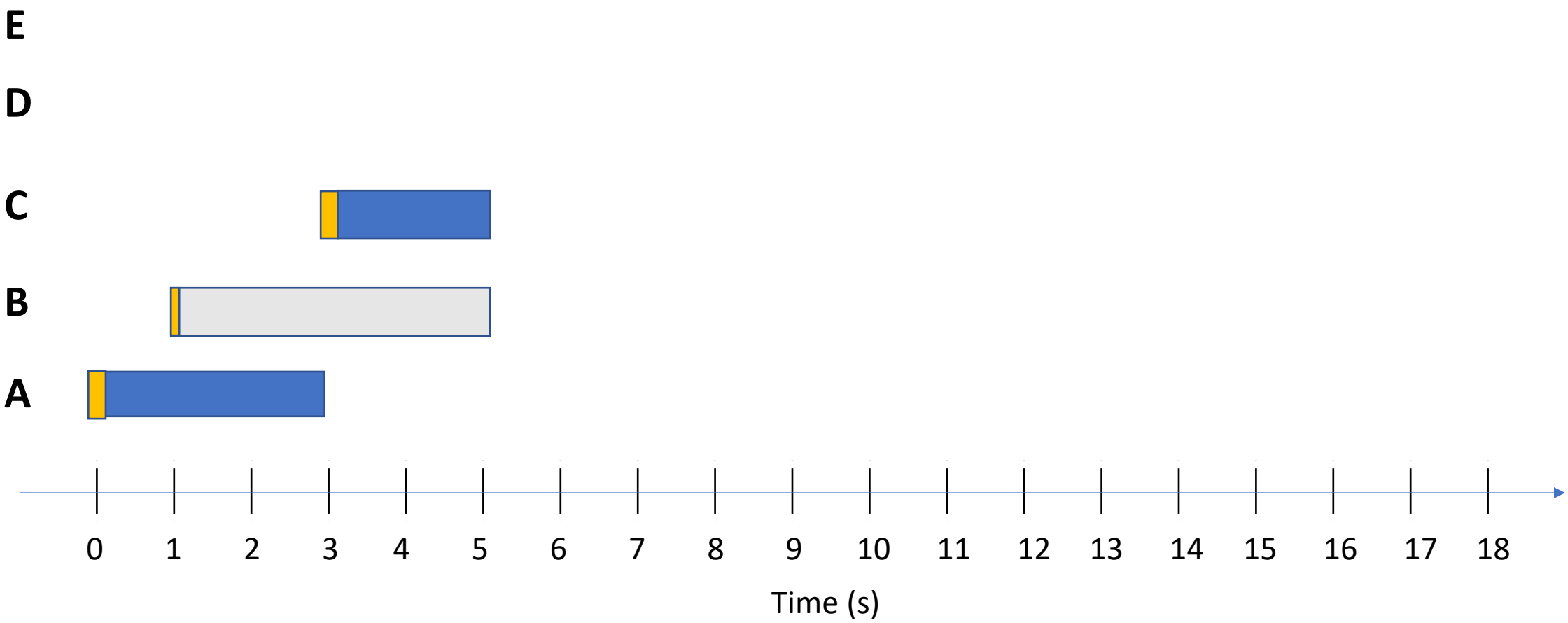
SJF - preemptive

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



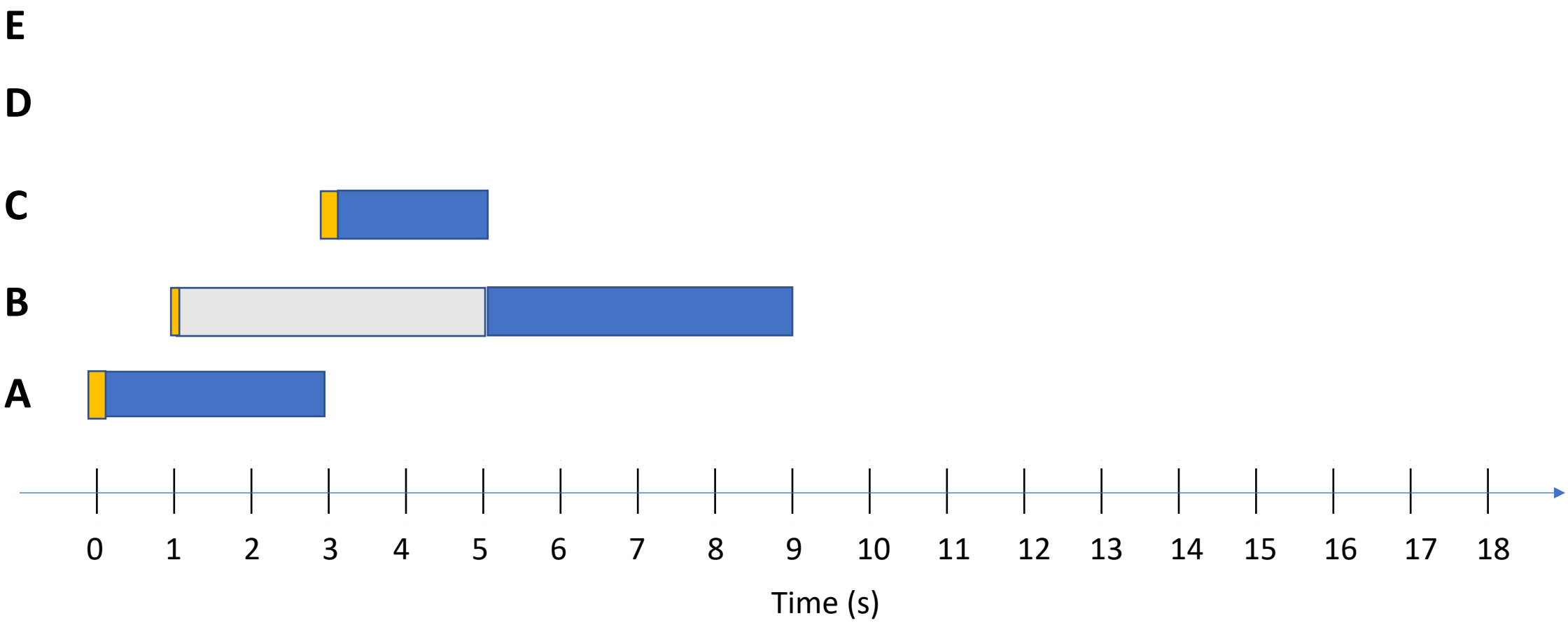
SJF - preemptive

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



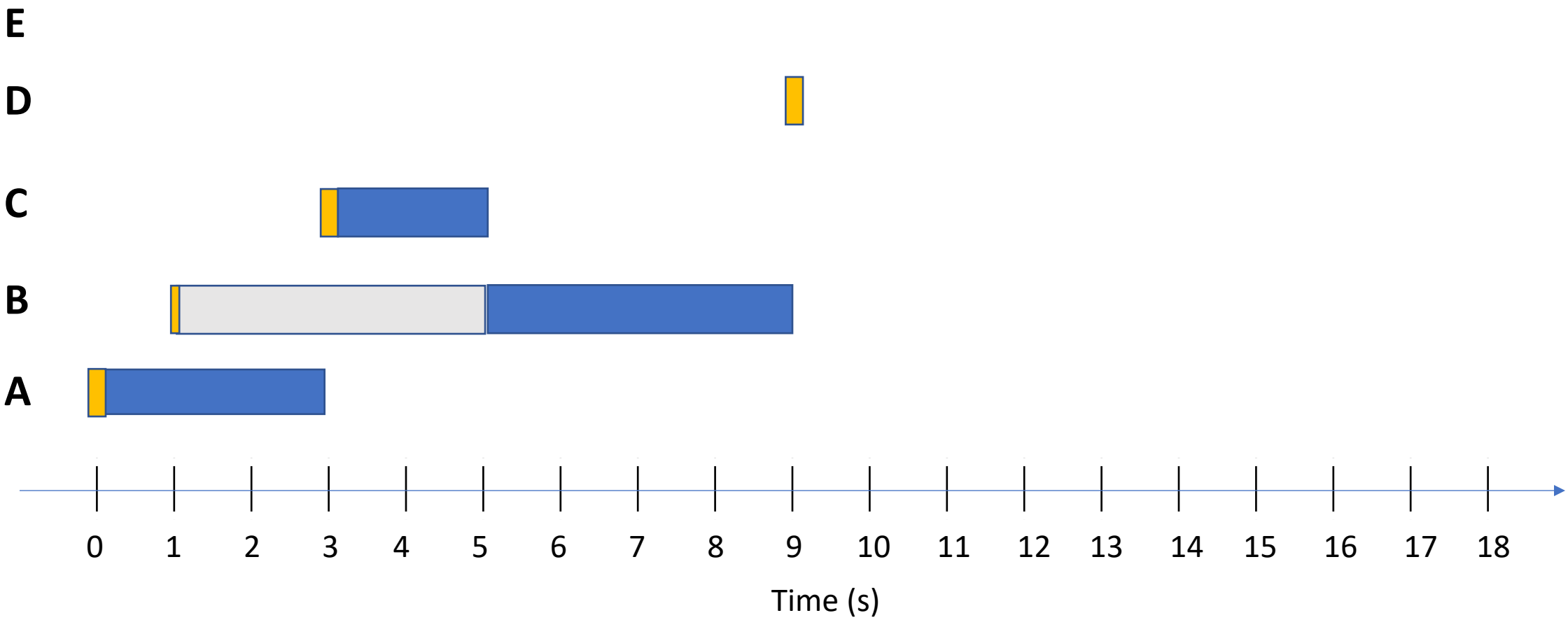
SJF - preemptive

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



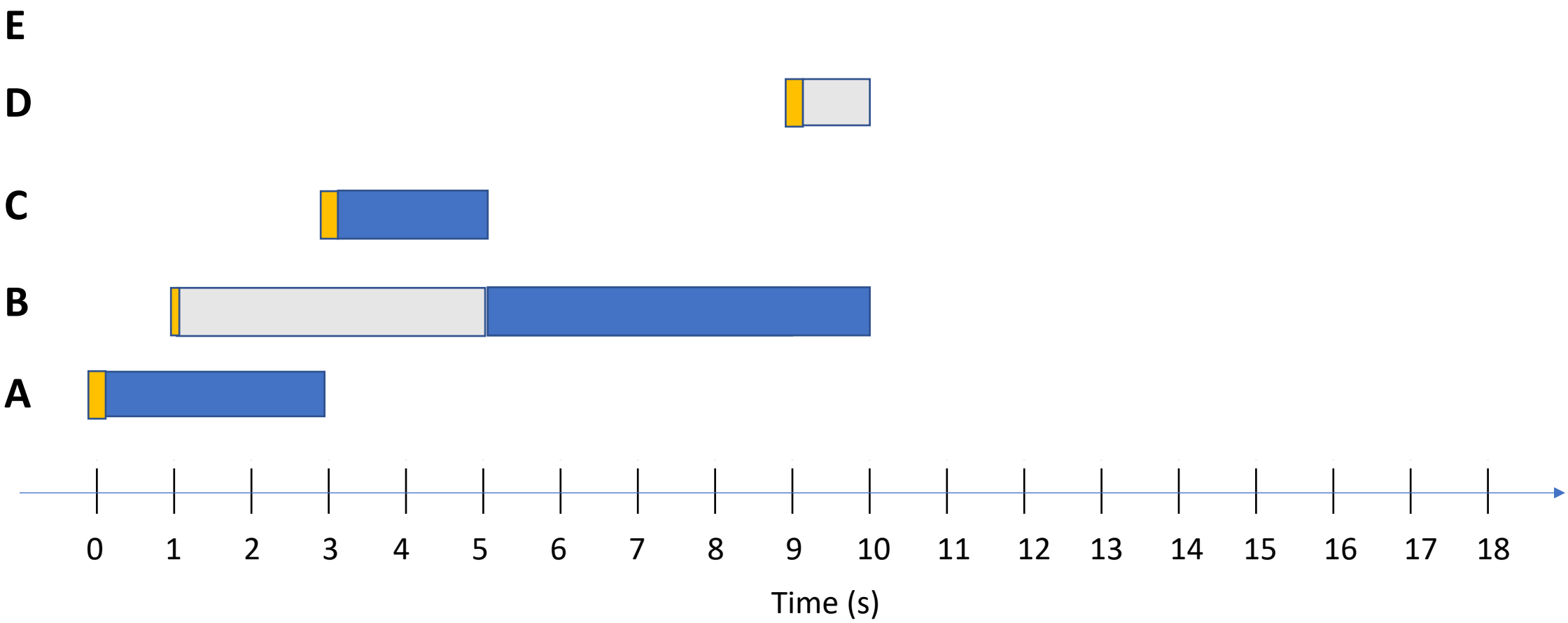
SJF - preemptive

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



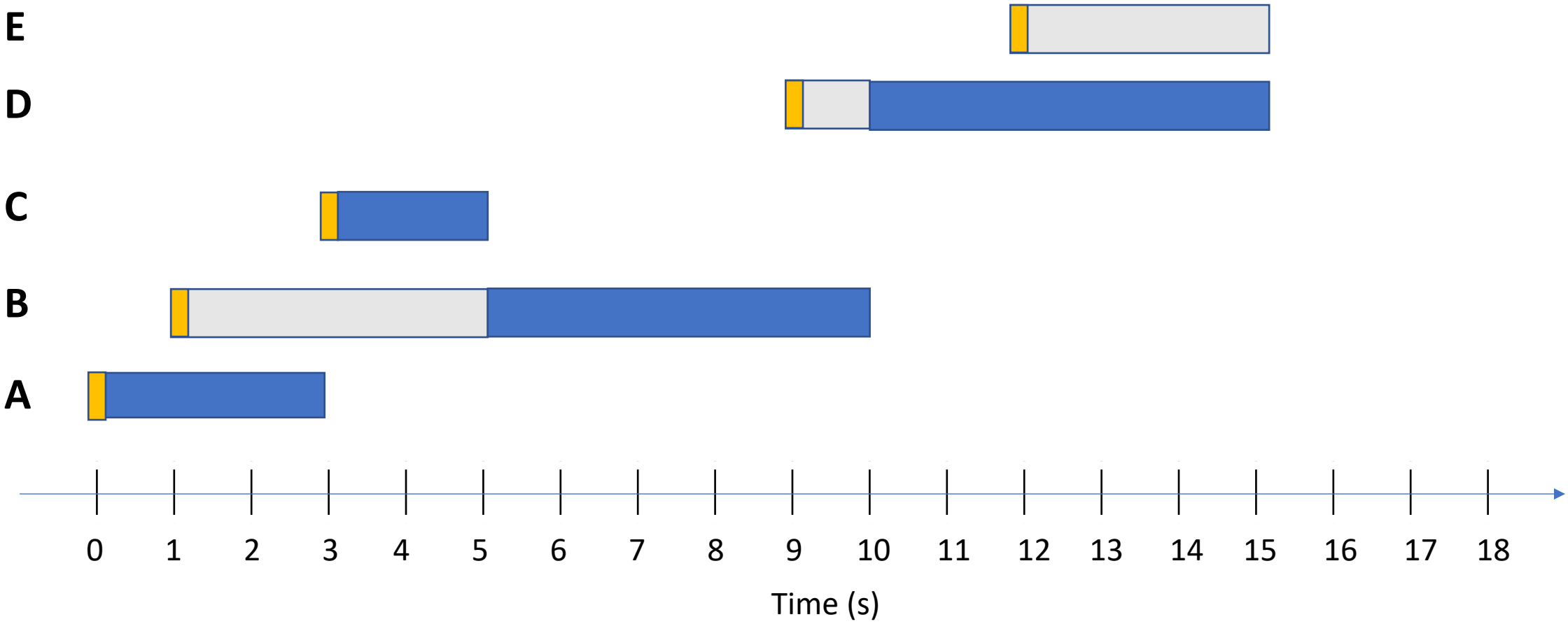
SJF - preemptive

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



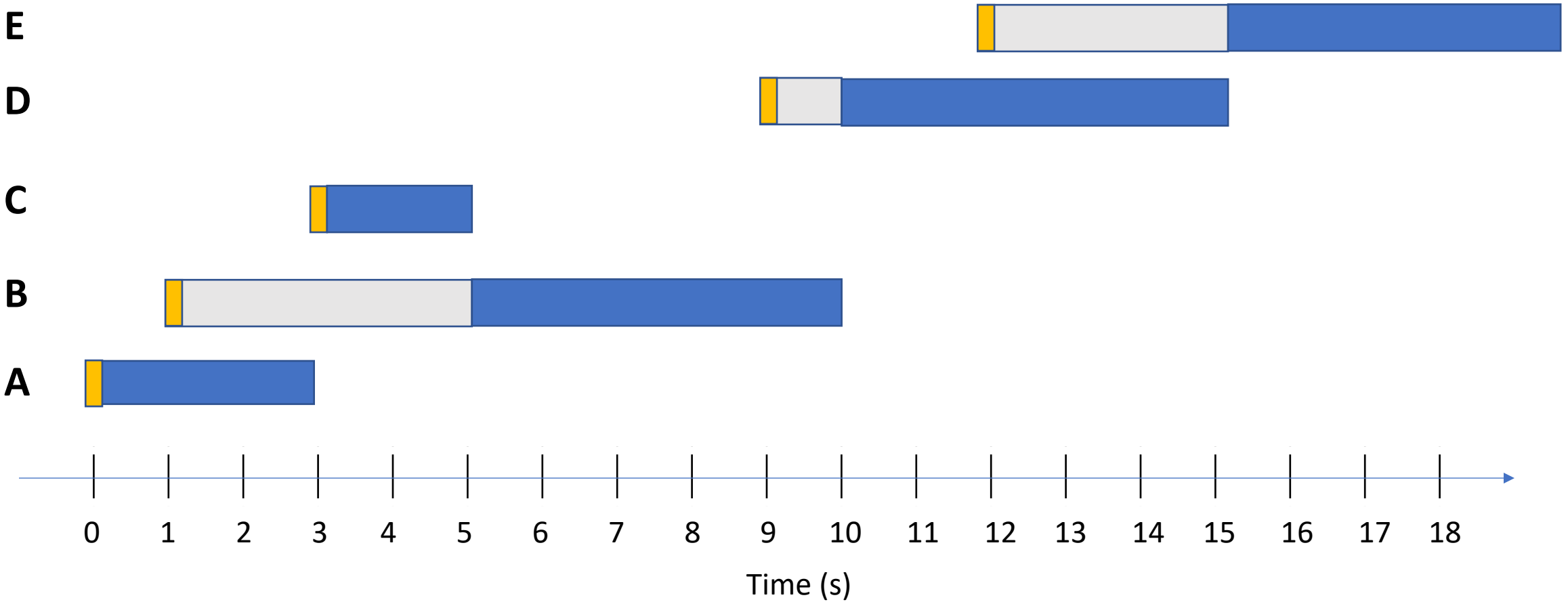
SJF - preemptive

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



SJF - preemptive

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3	3	0
B	1	5	9	4
C	3	2	2	0
D	9	5	6	1
E	12	5	8	3

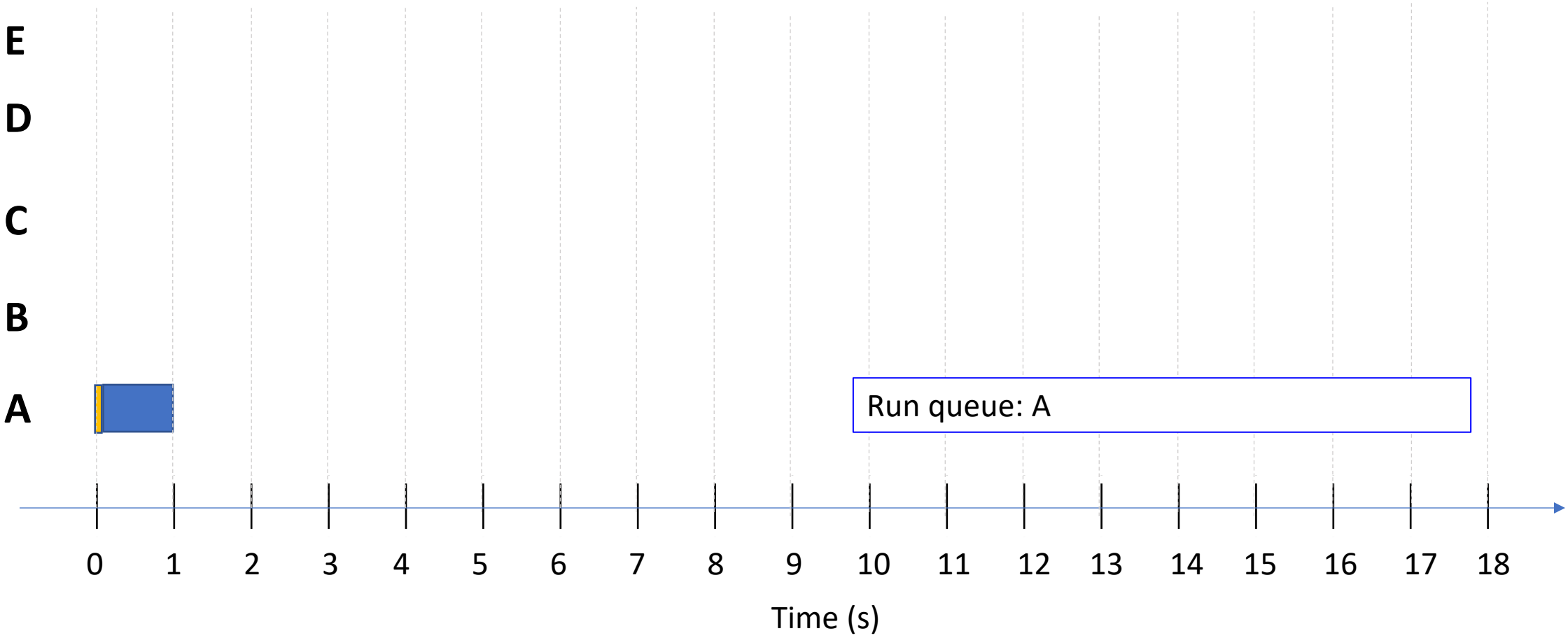


RR

- **time quantum of 1.**

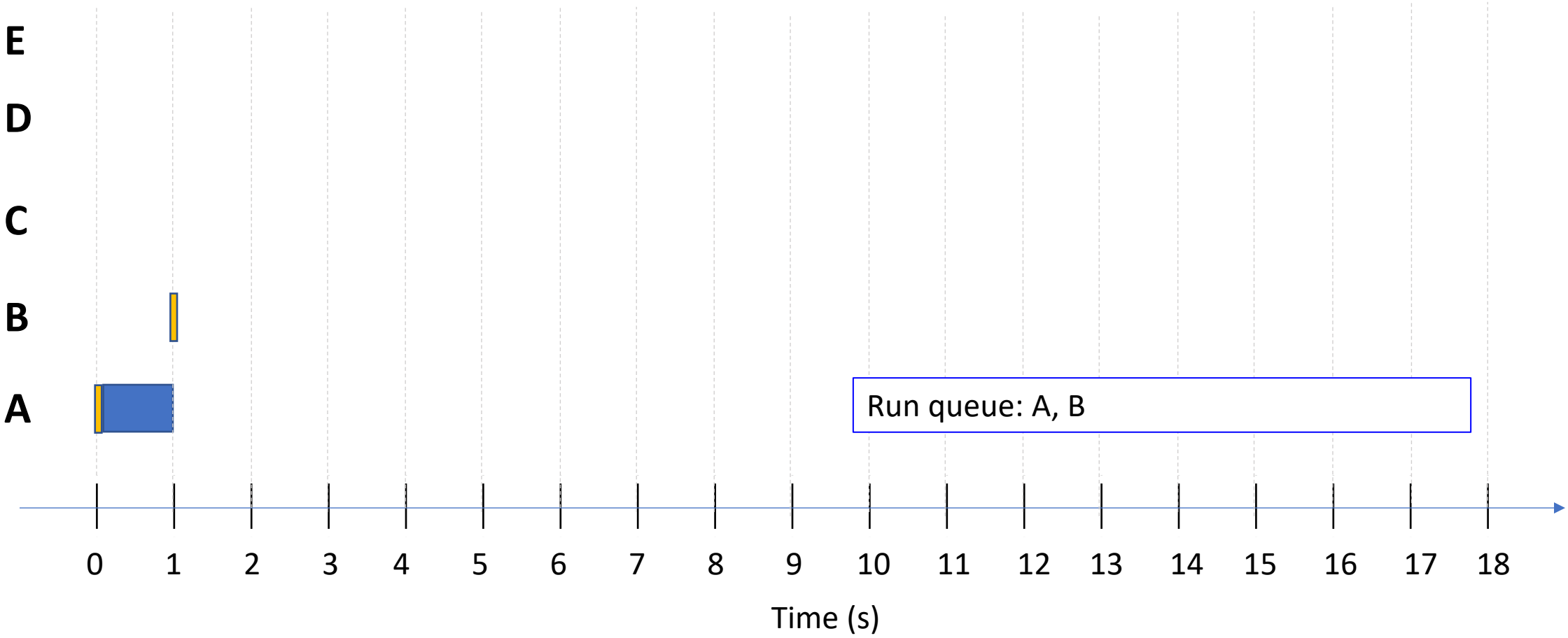
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



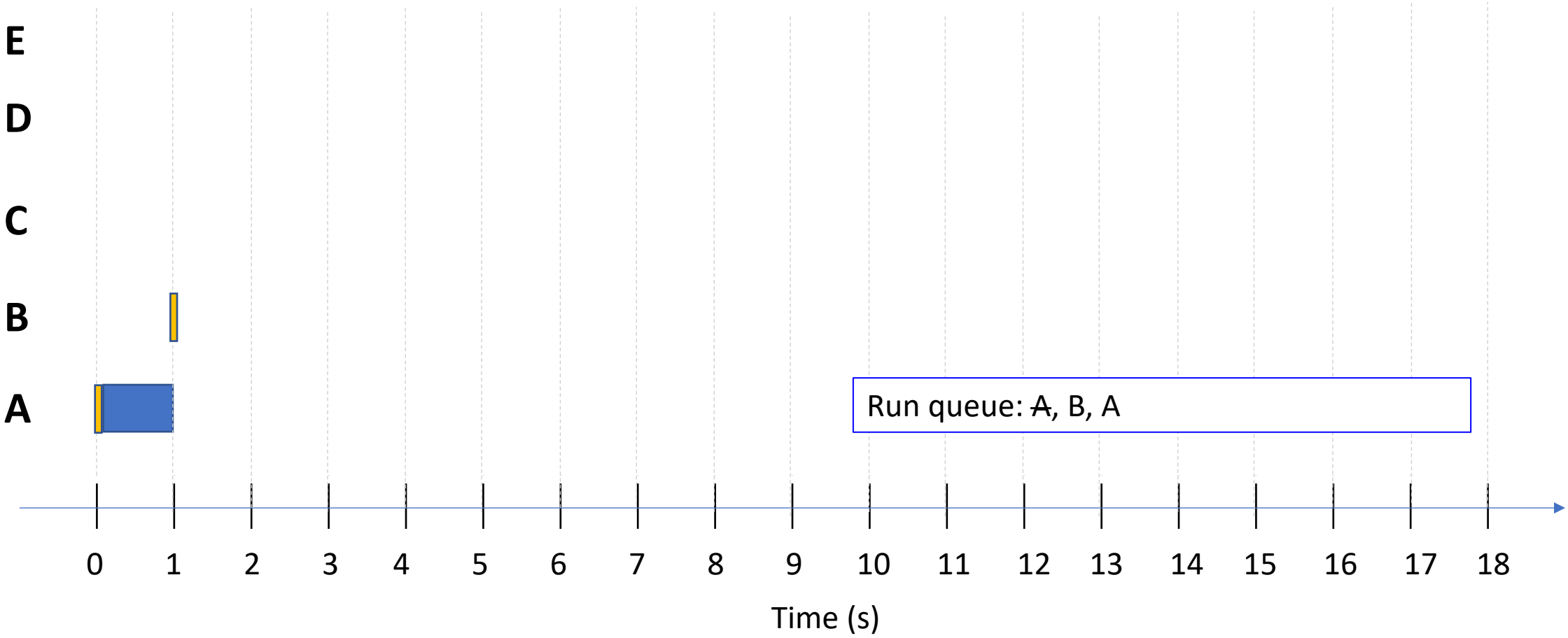
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



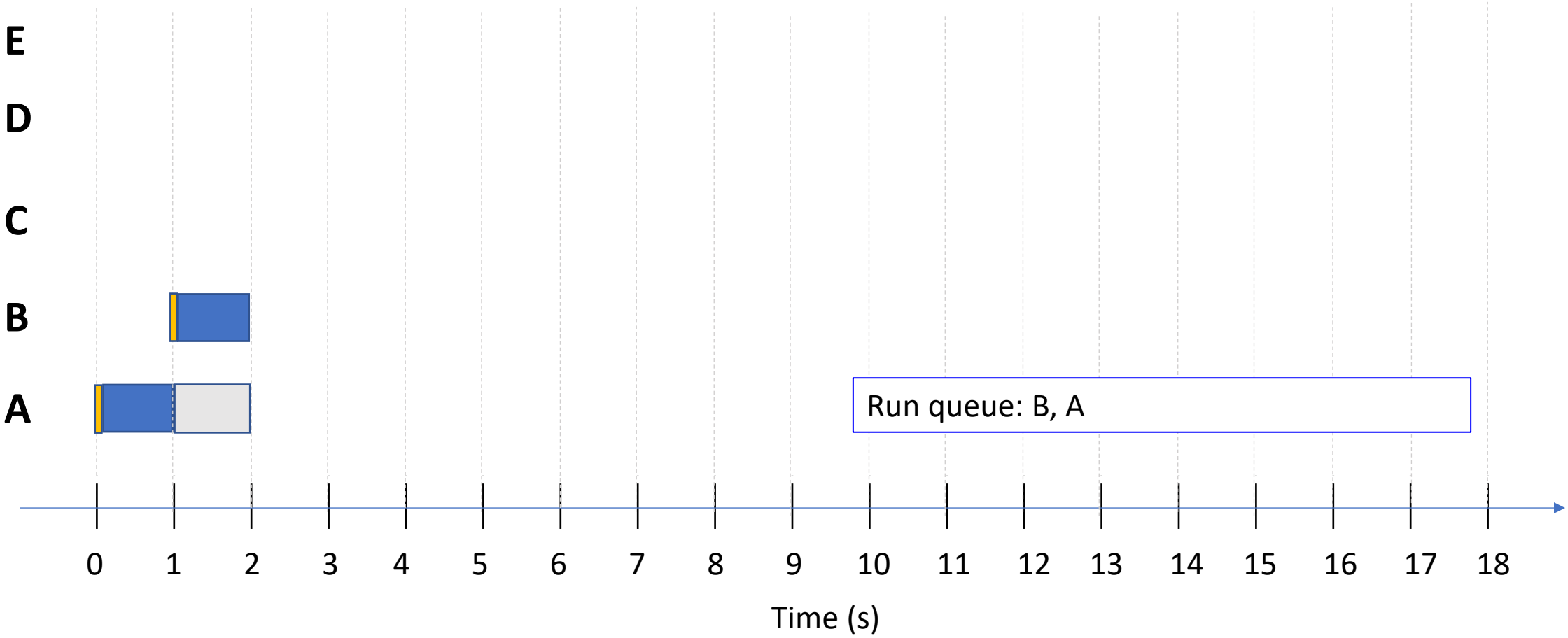
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



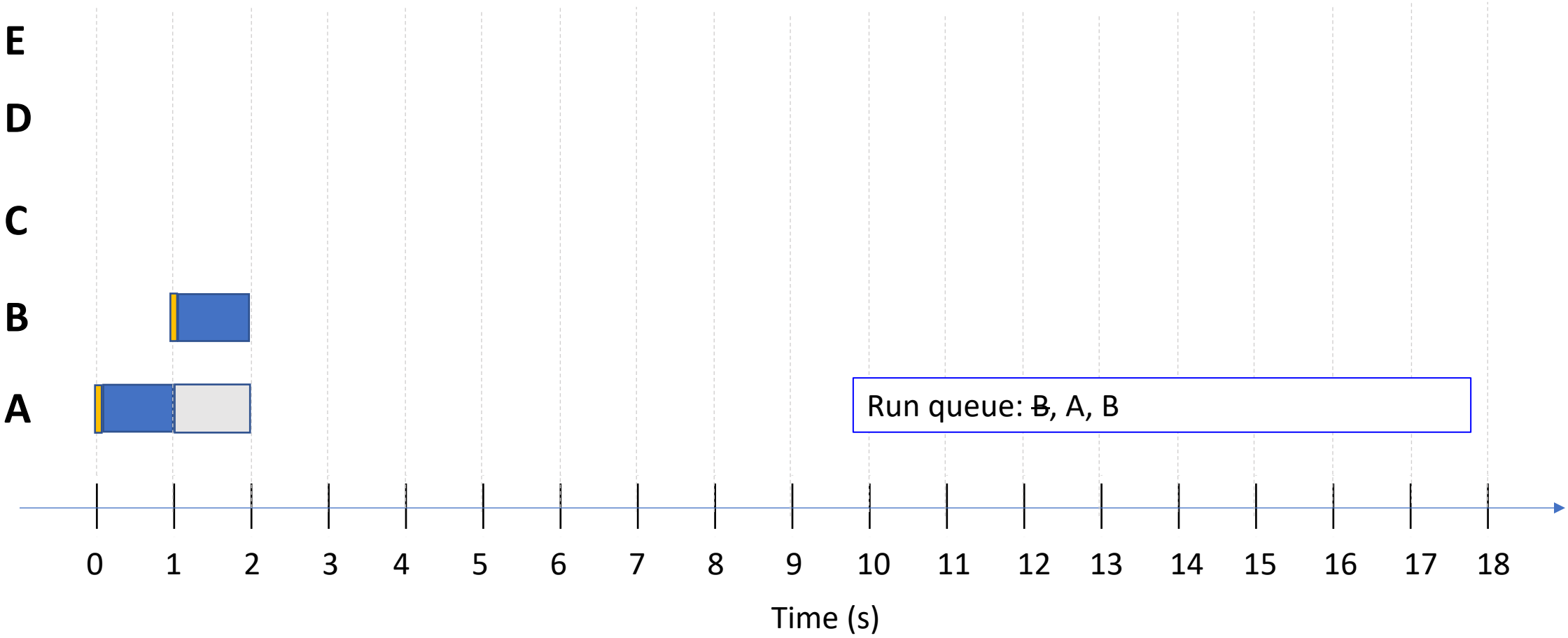
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



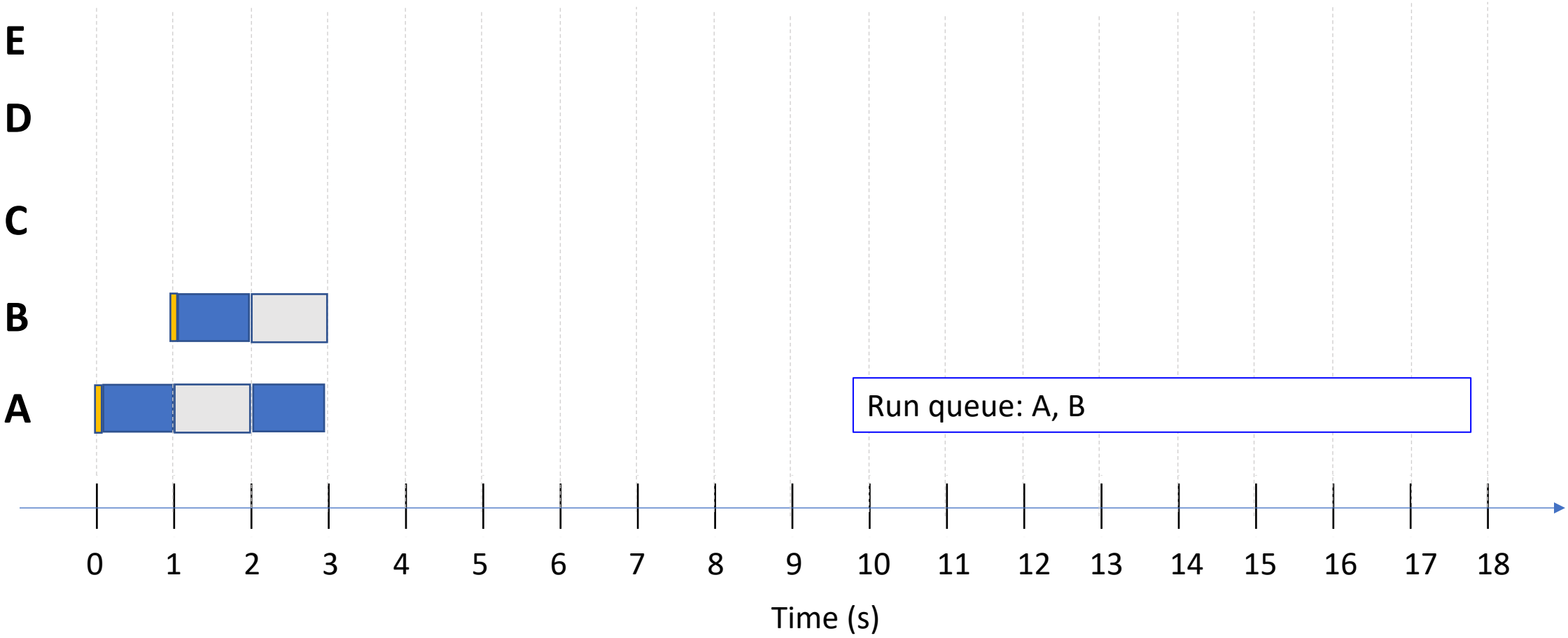
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



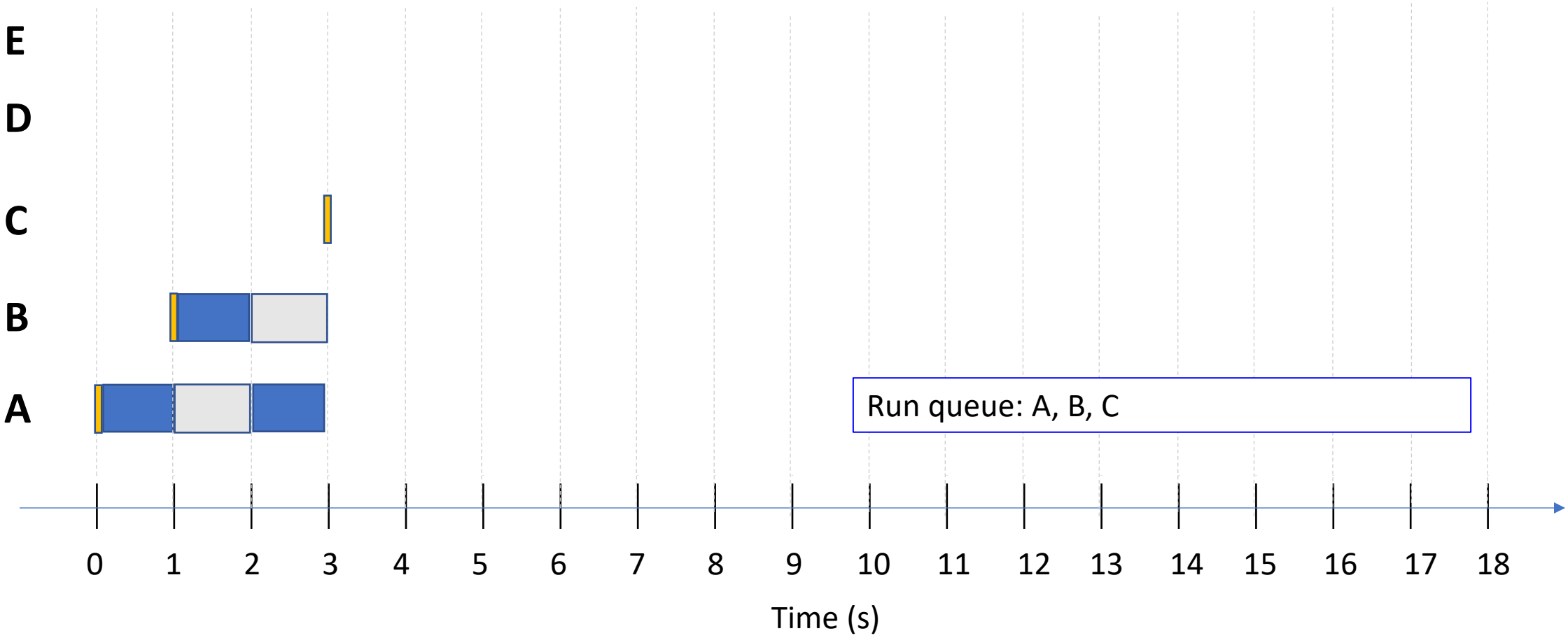
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



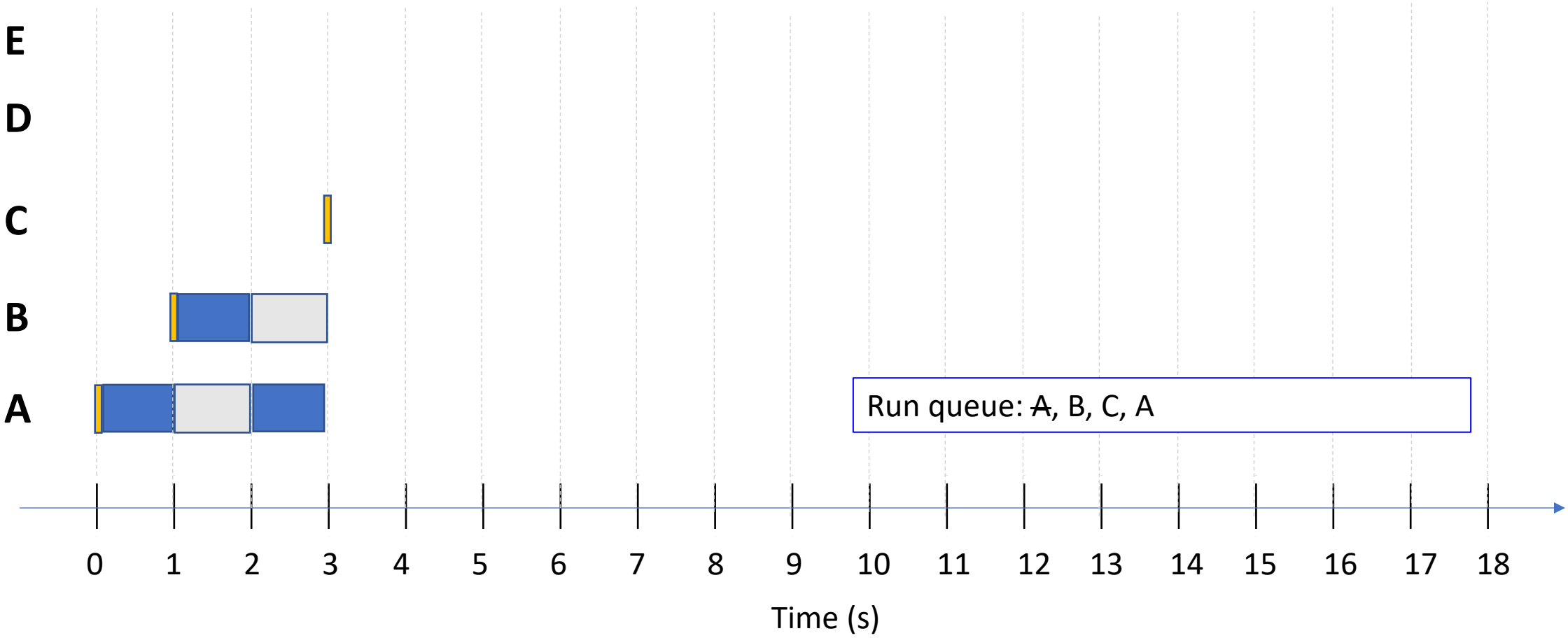
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



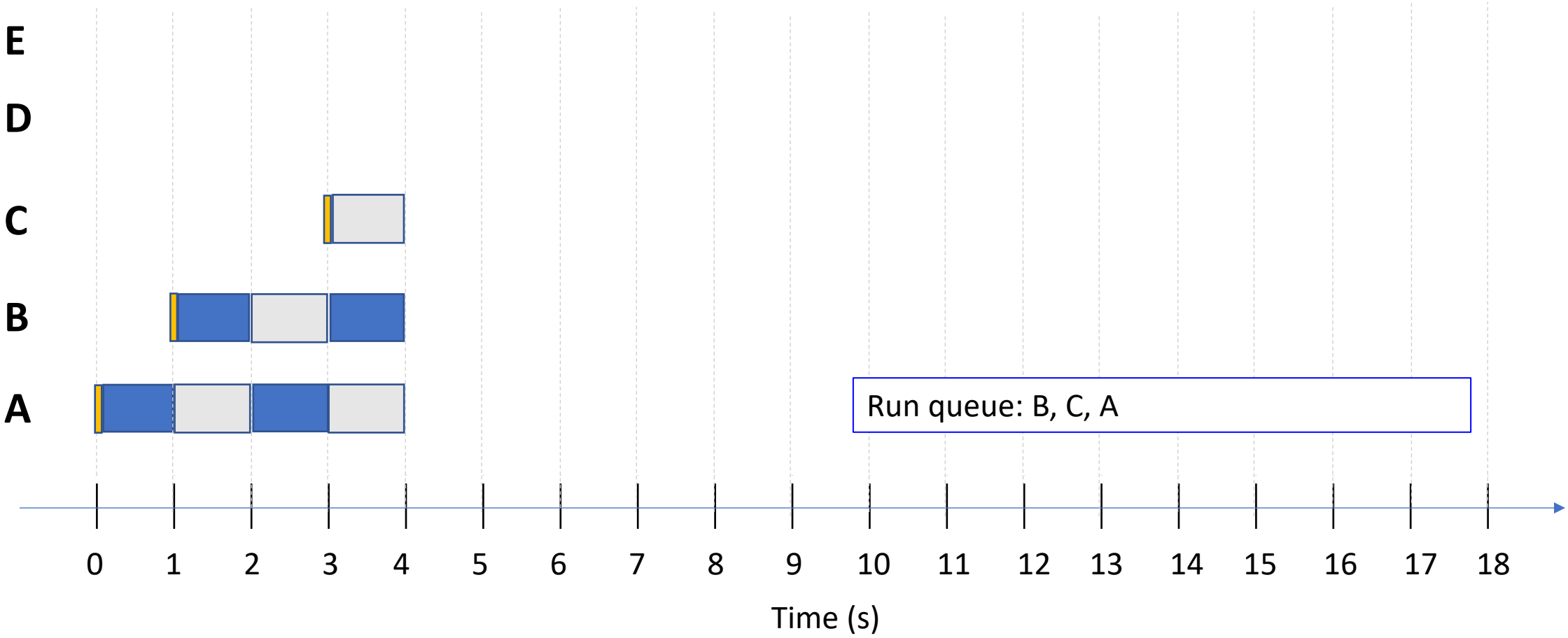
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



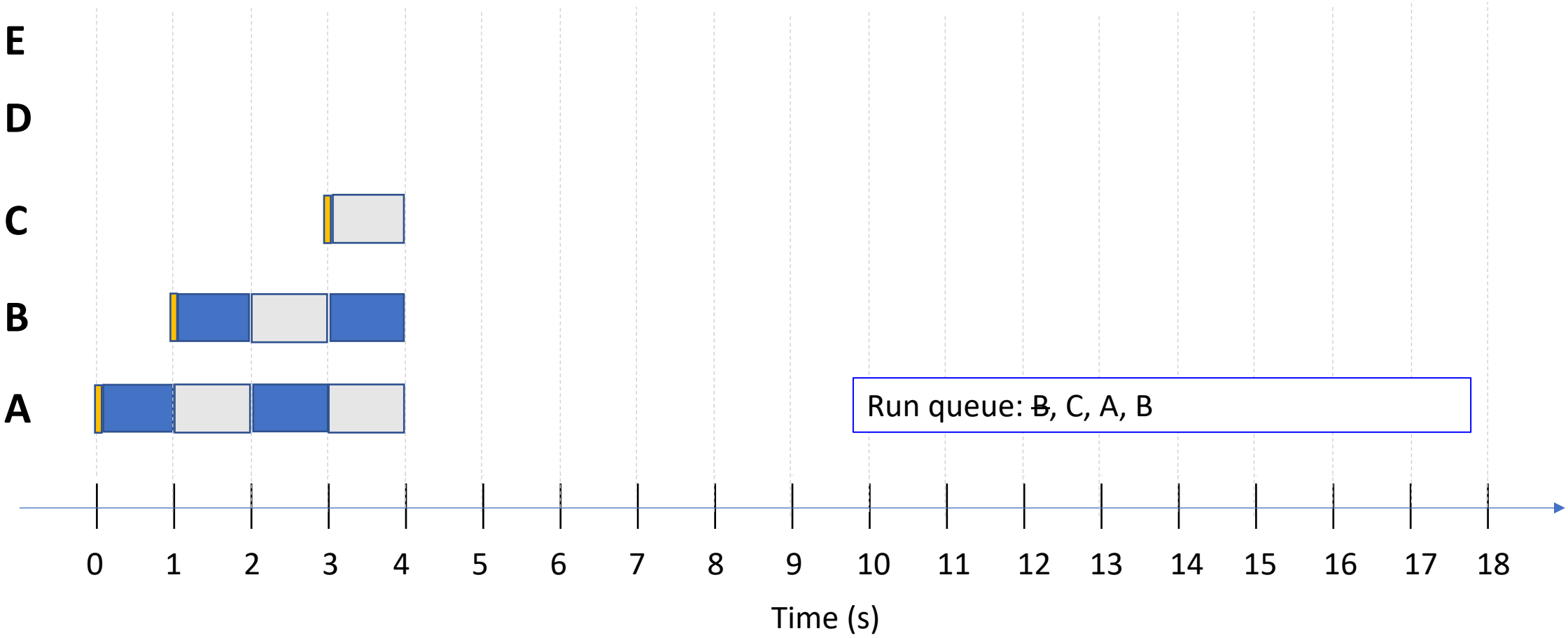
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



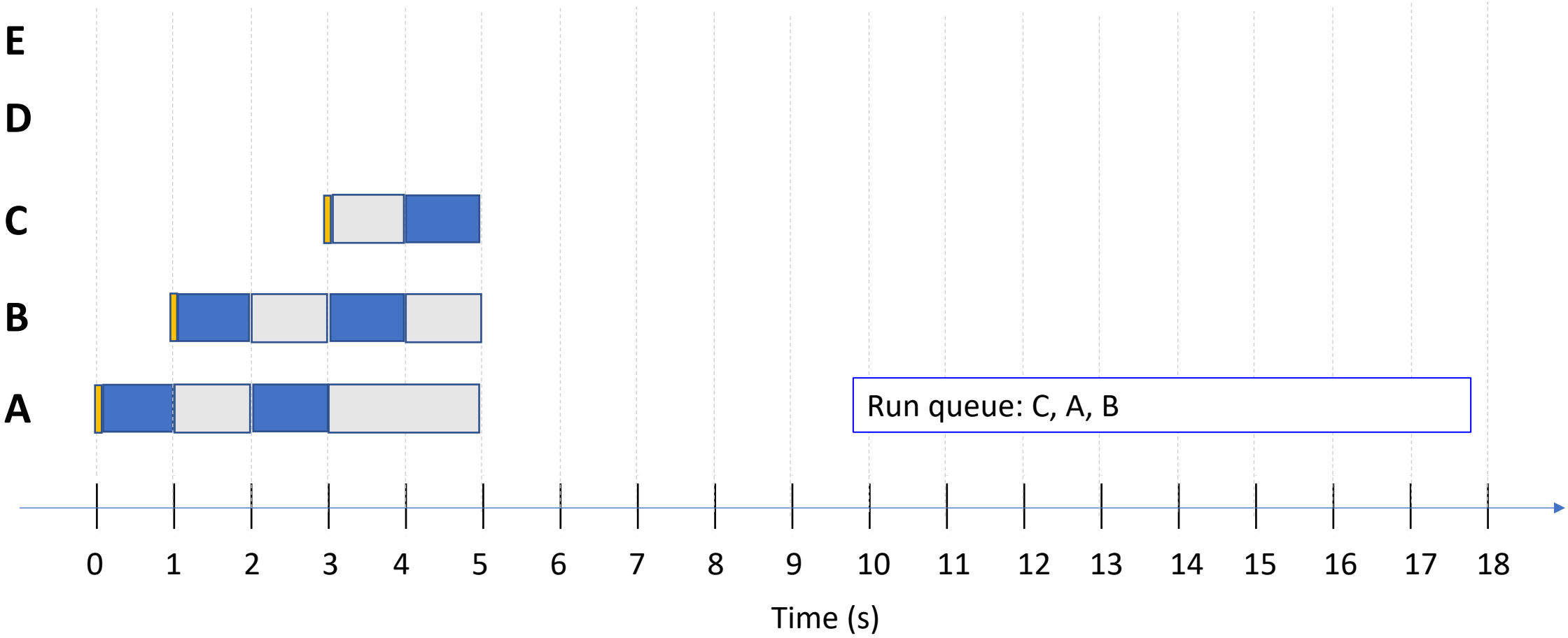
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



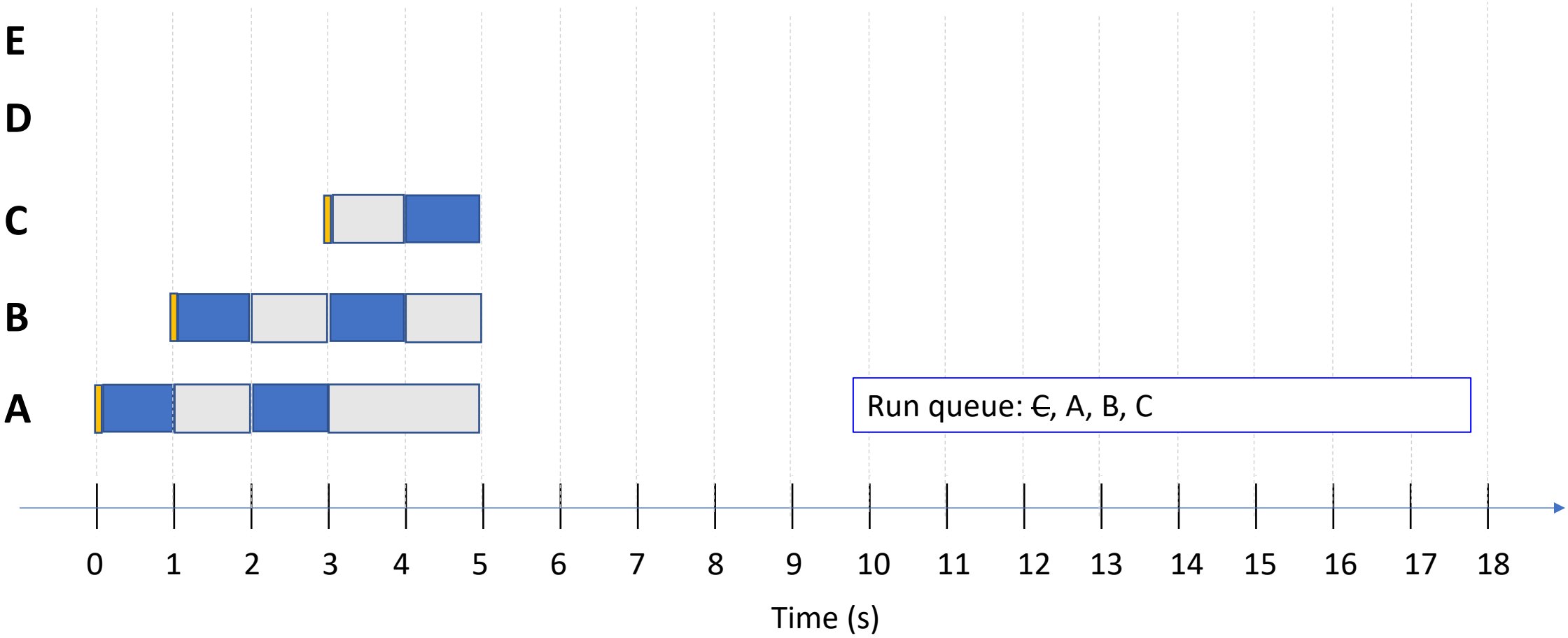
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



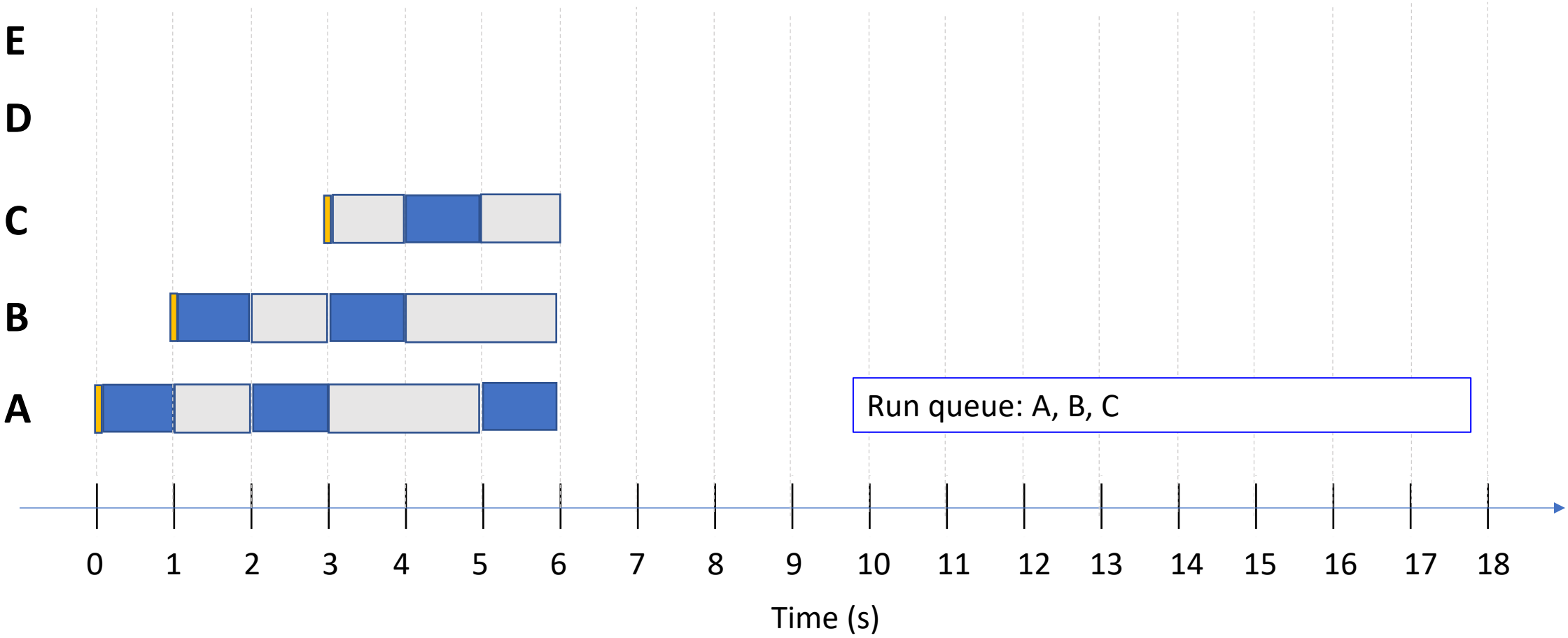
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



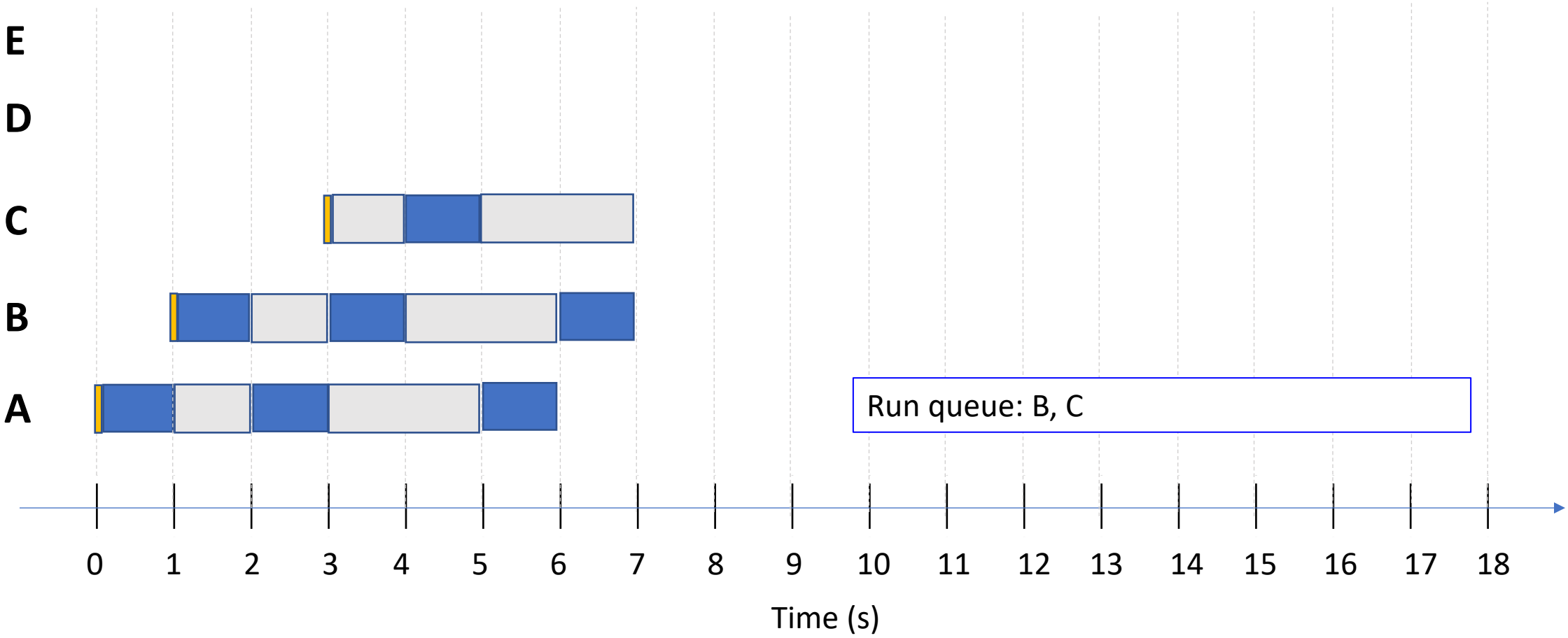
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



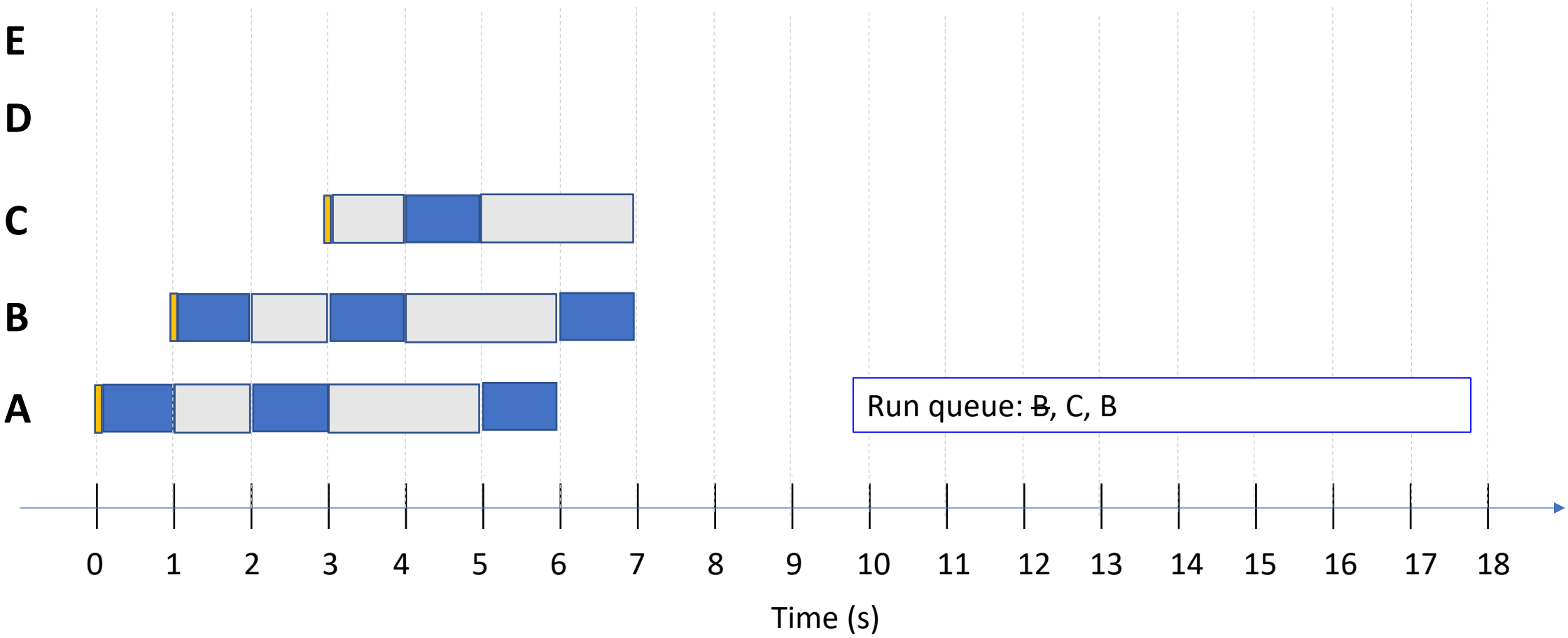
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



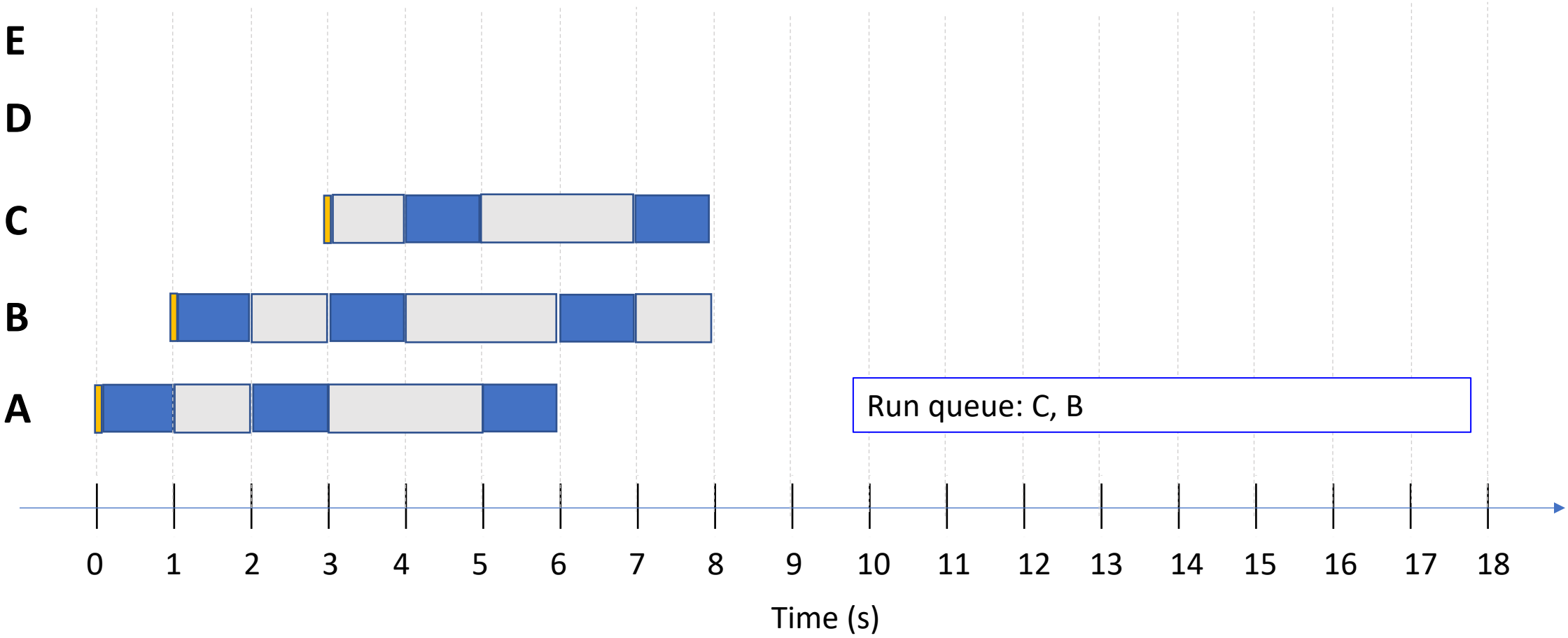
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



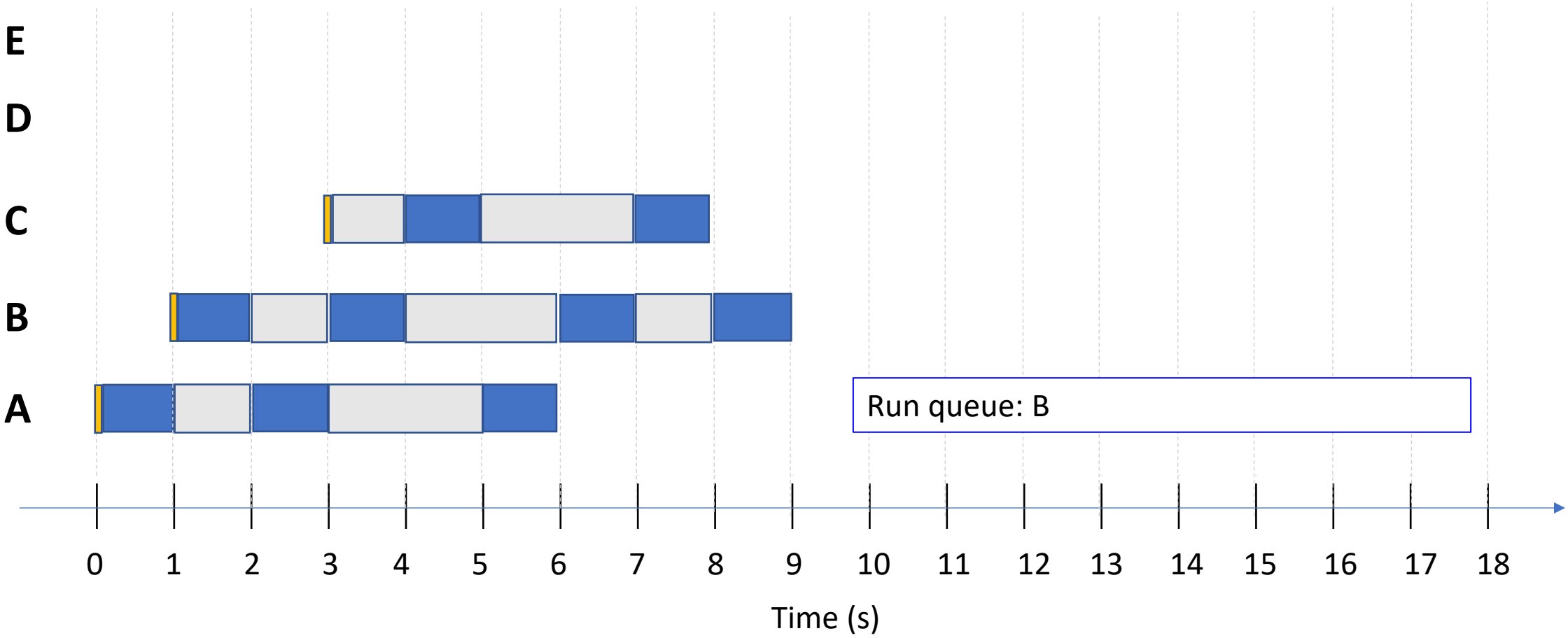
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



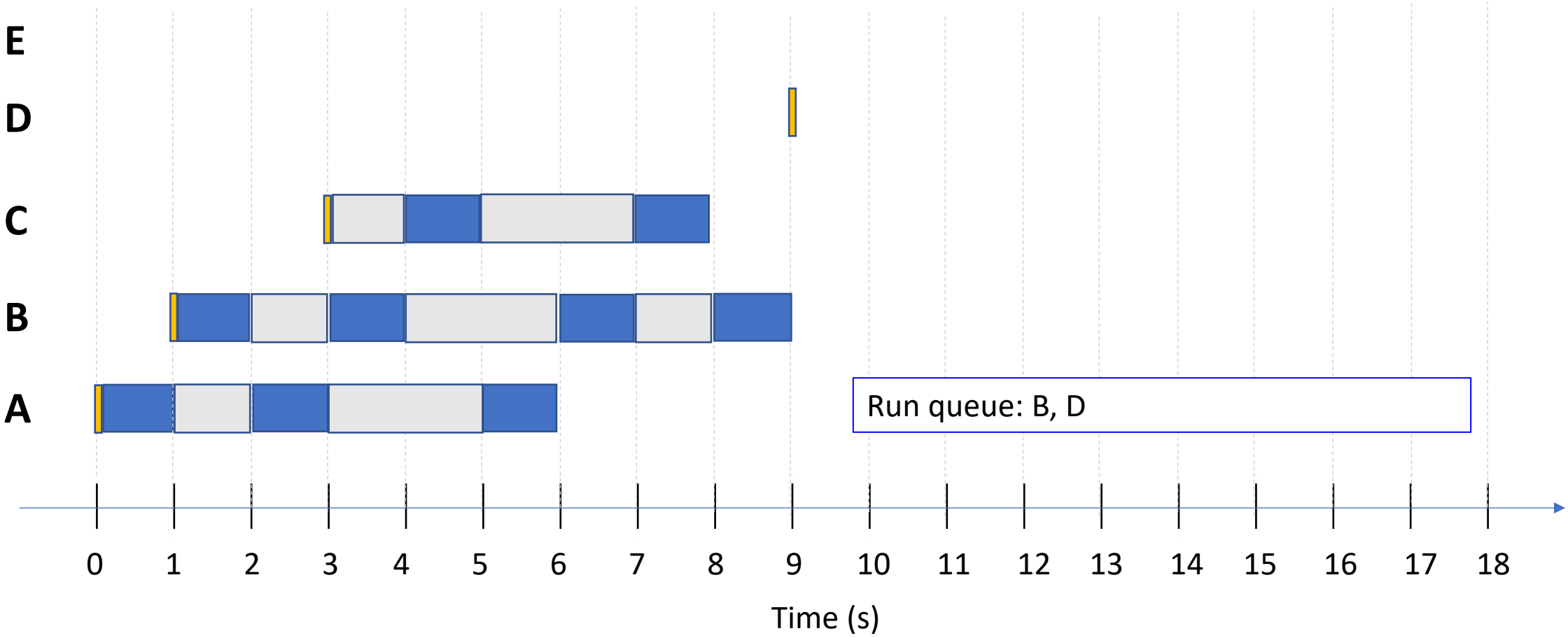
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



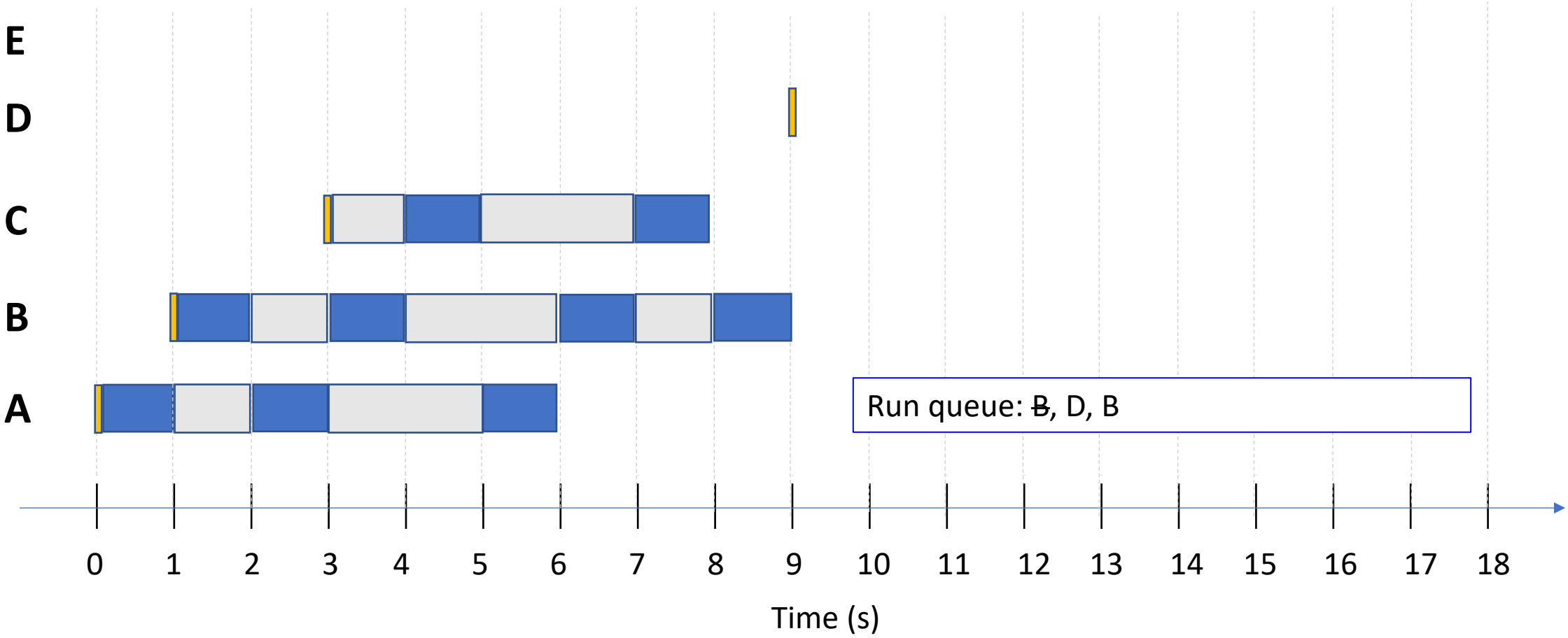
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



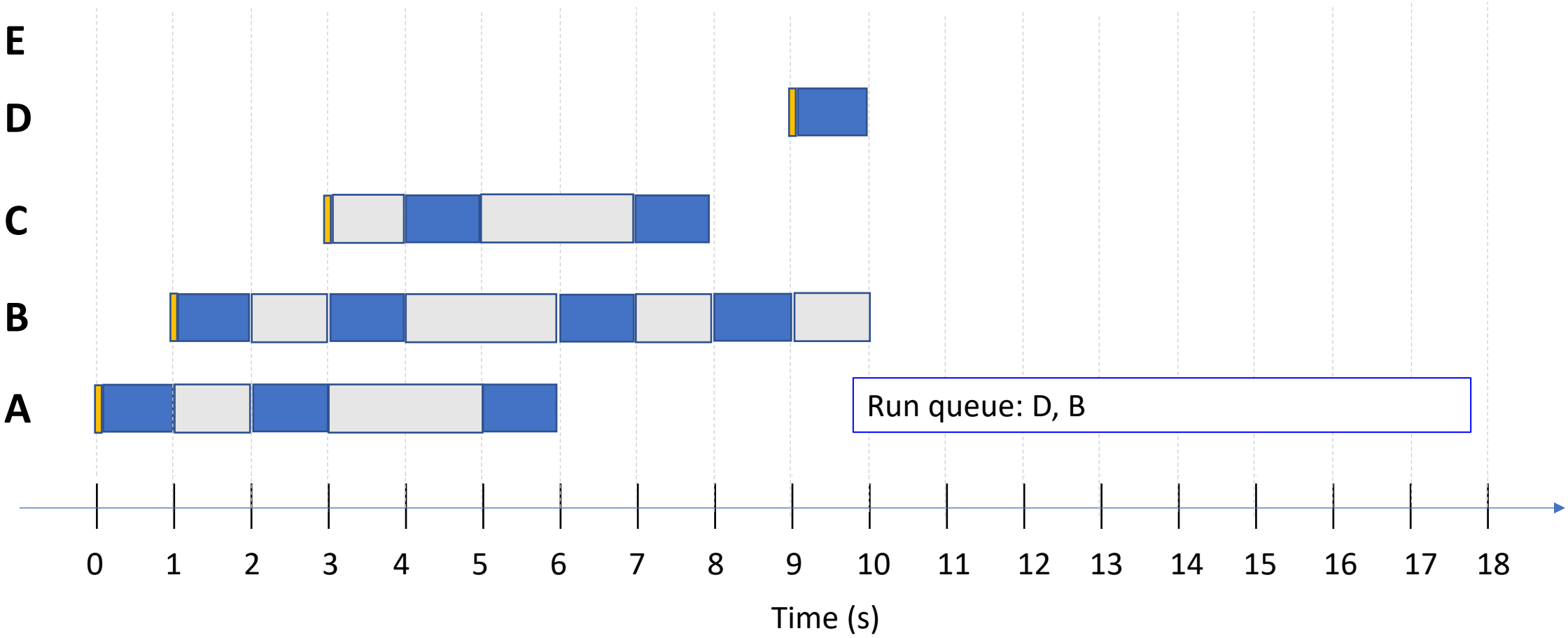
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



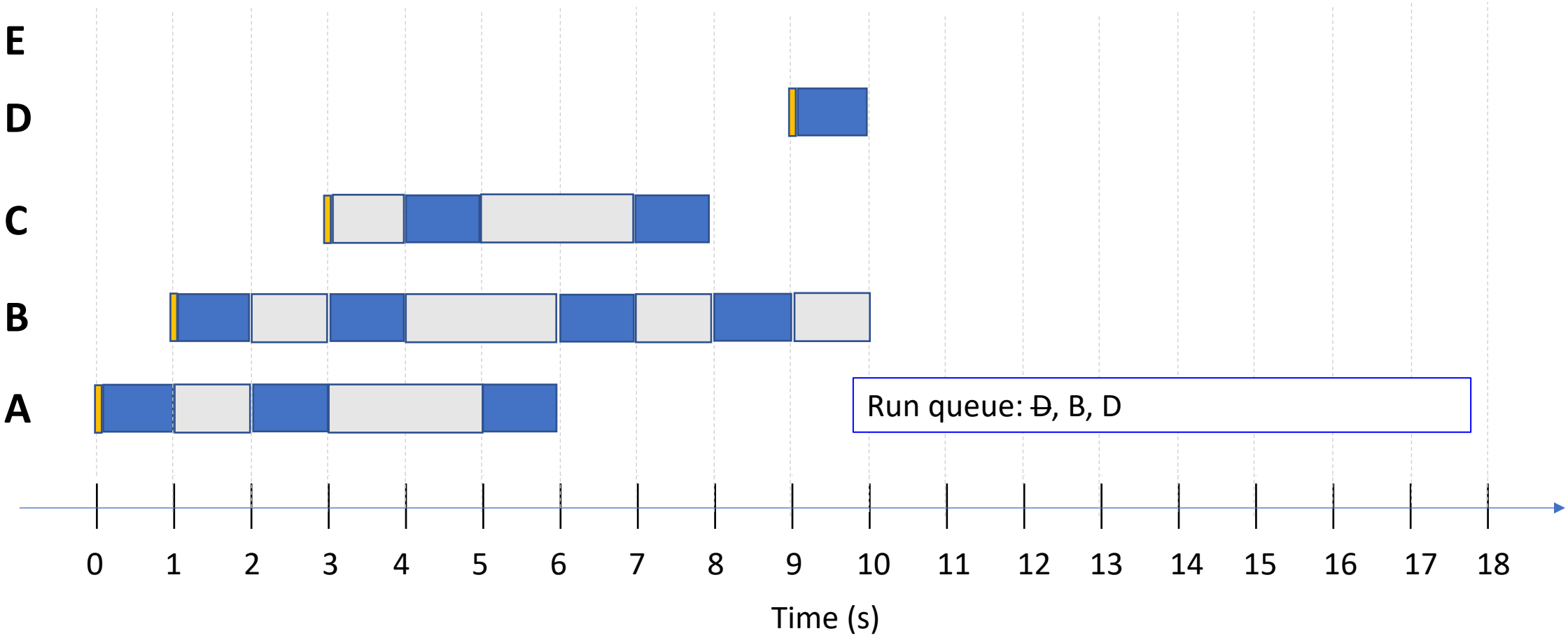
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



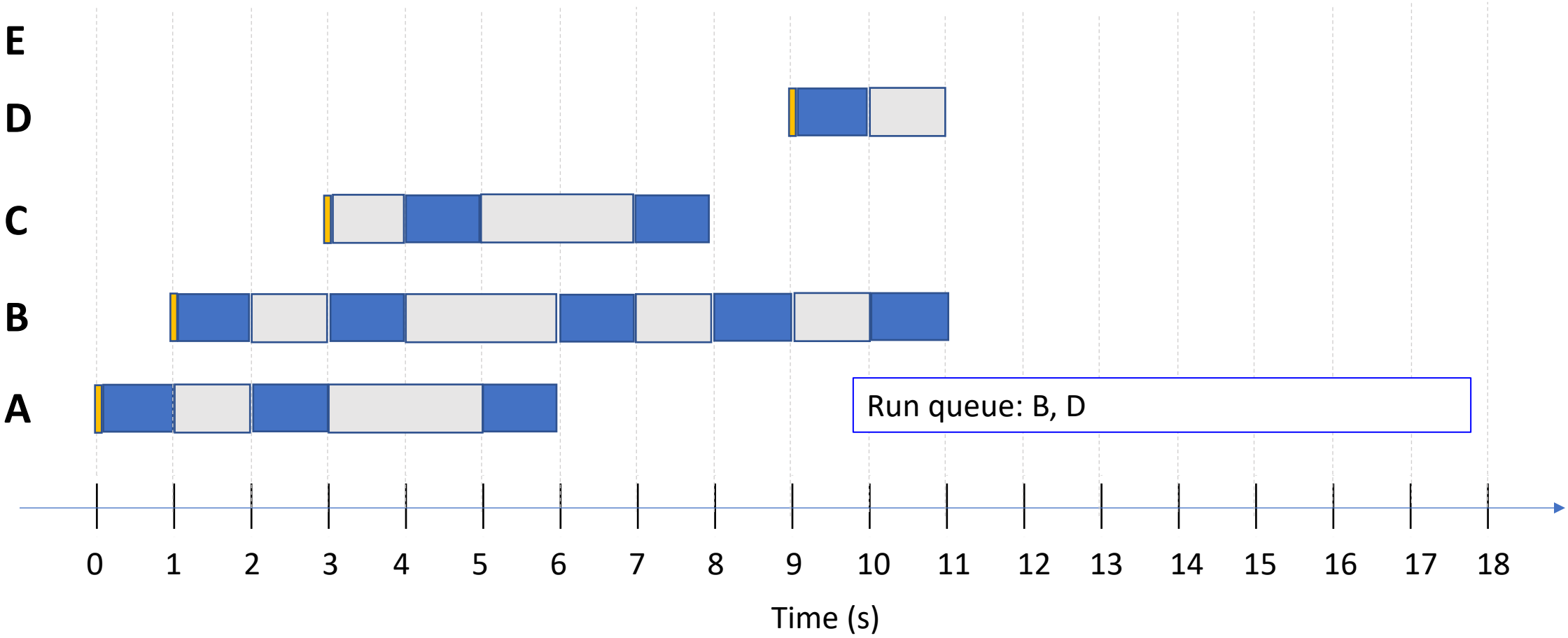
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



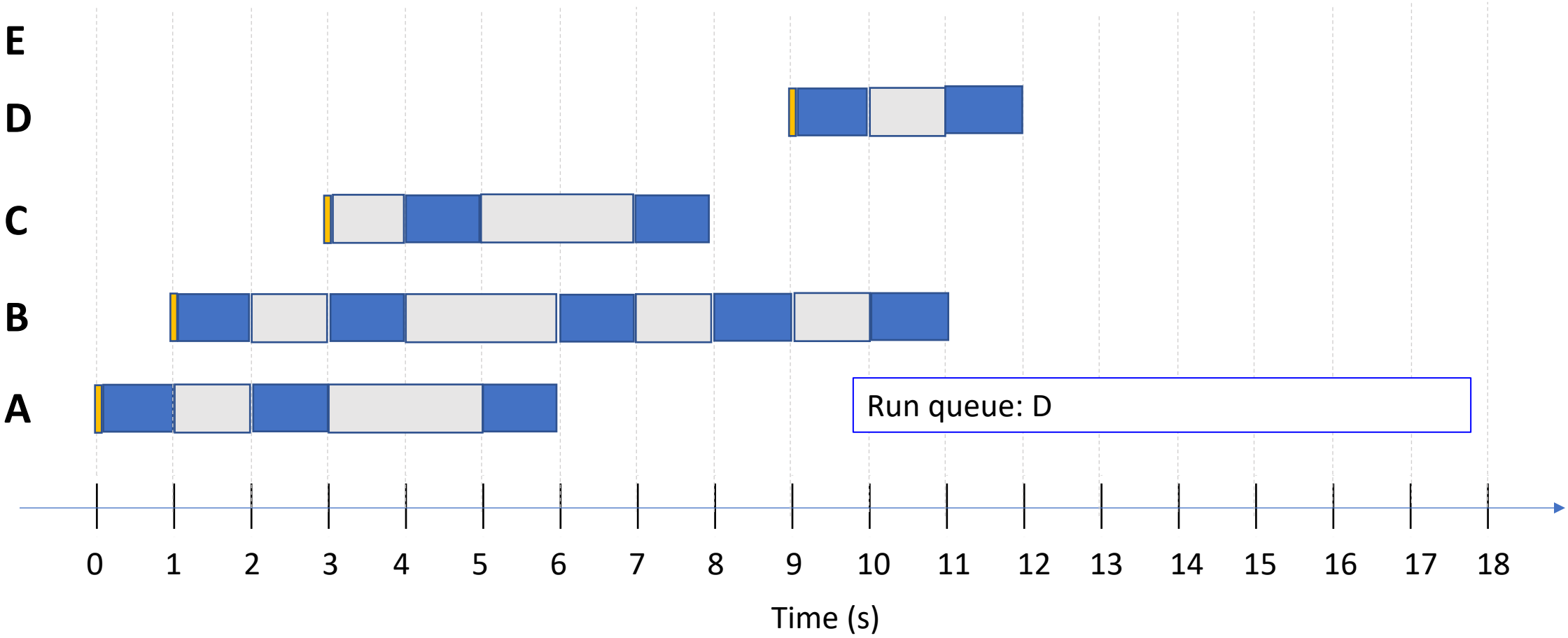
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



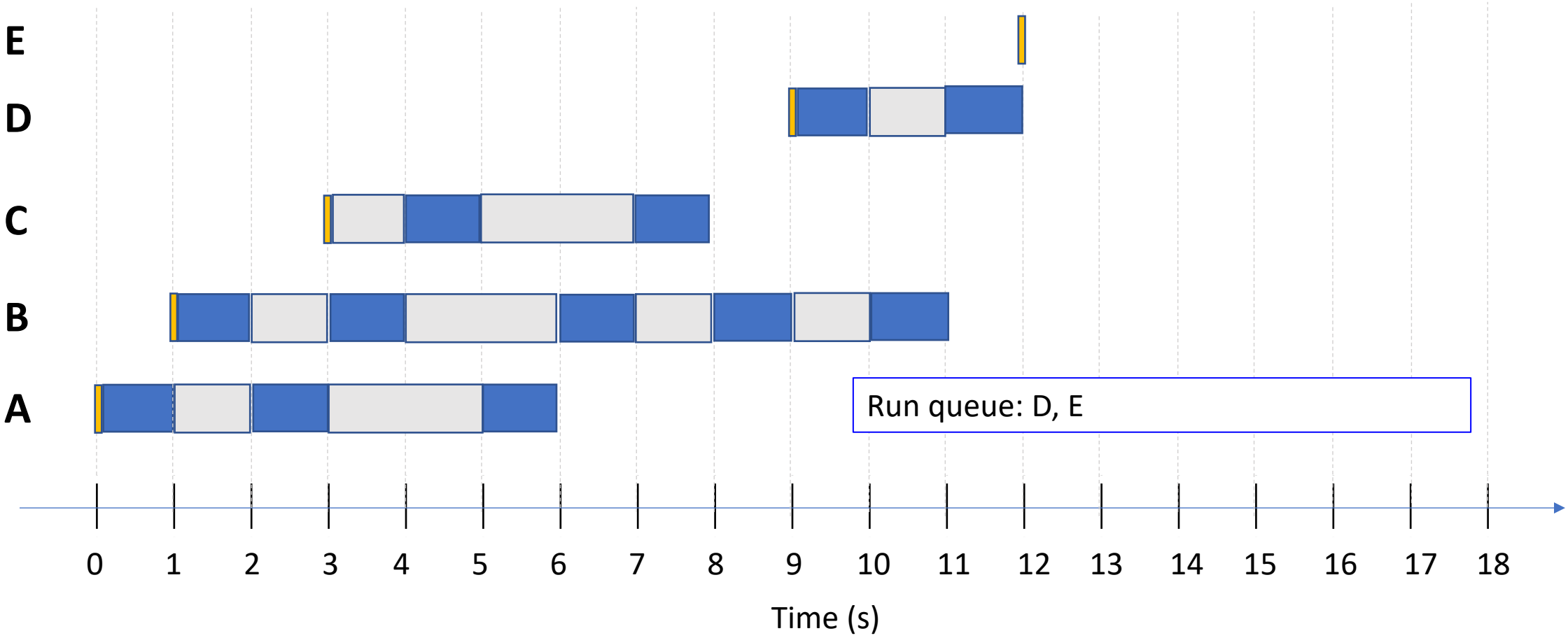
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



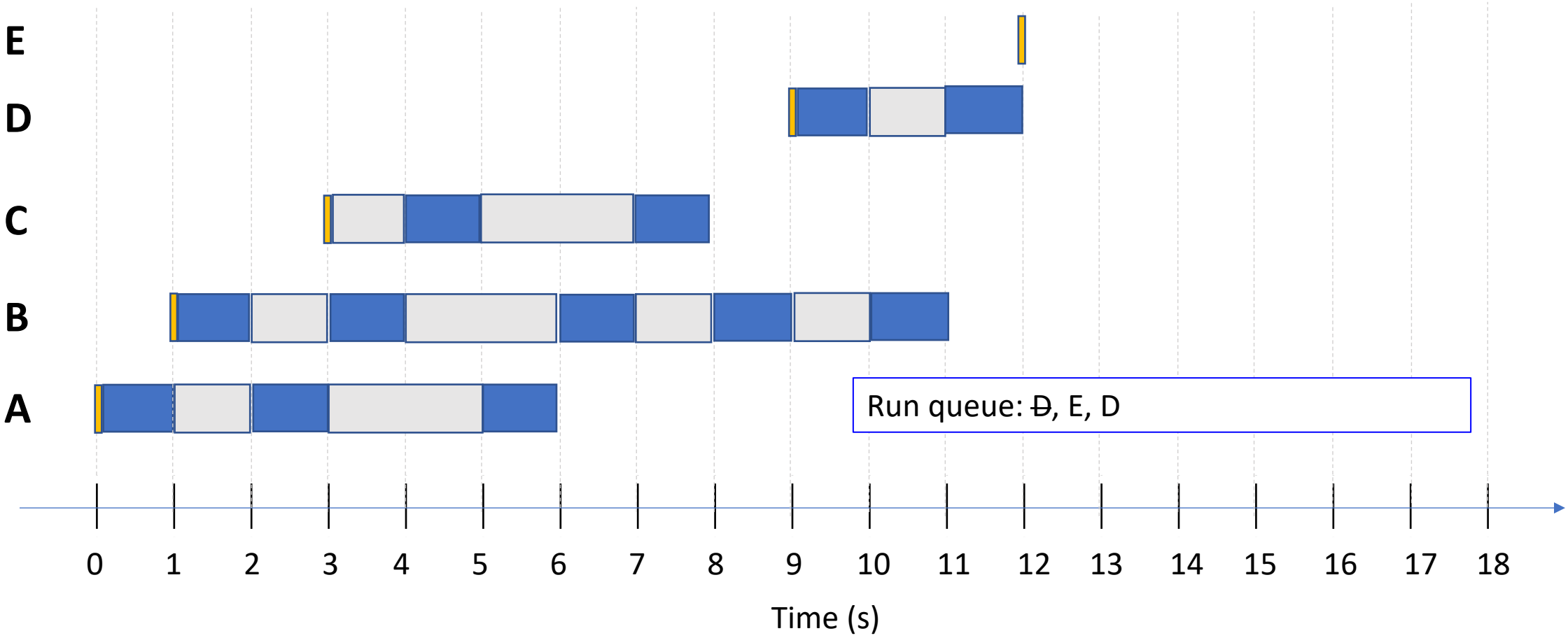
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



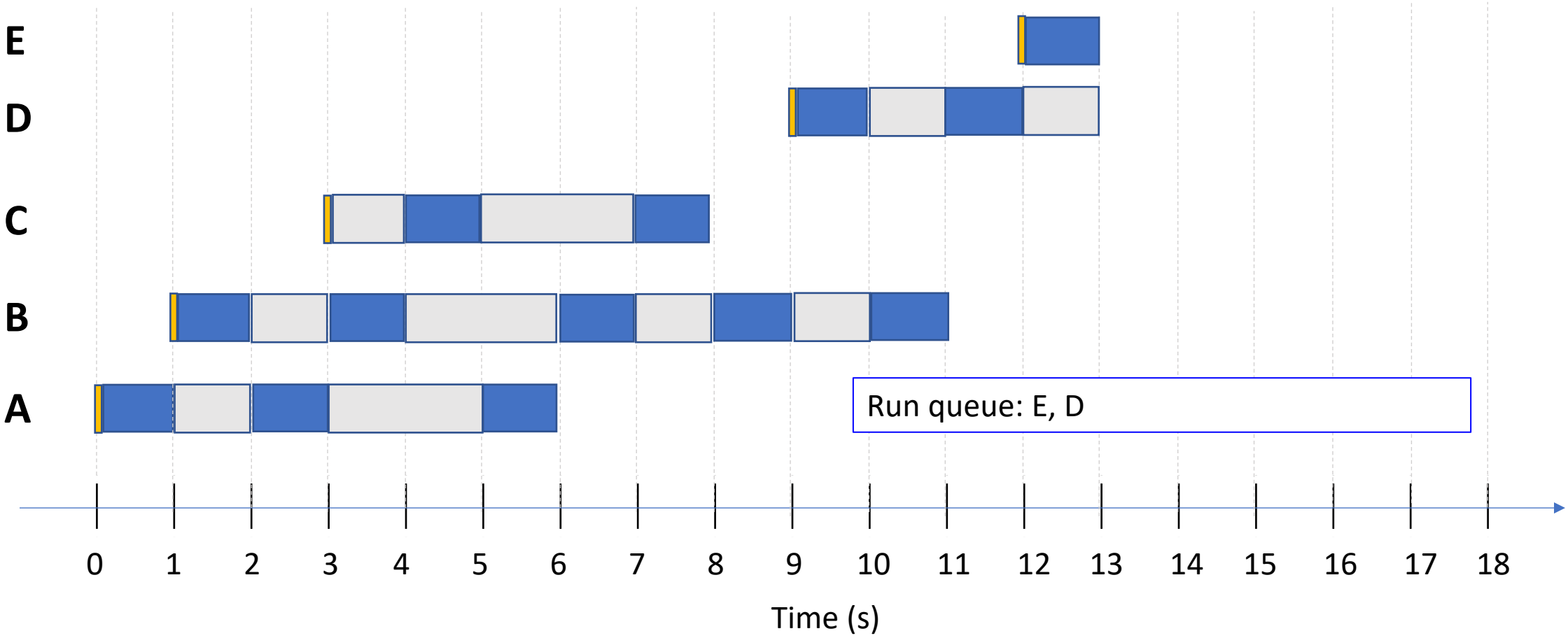
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



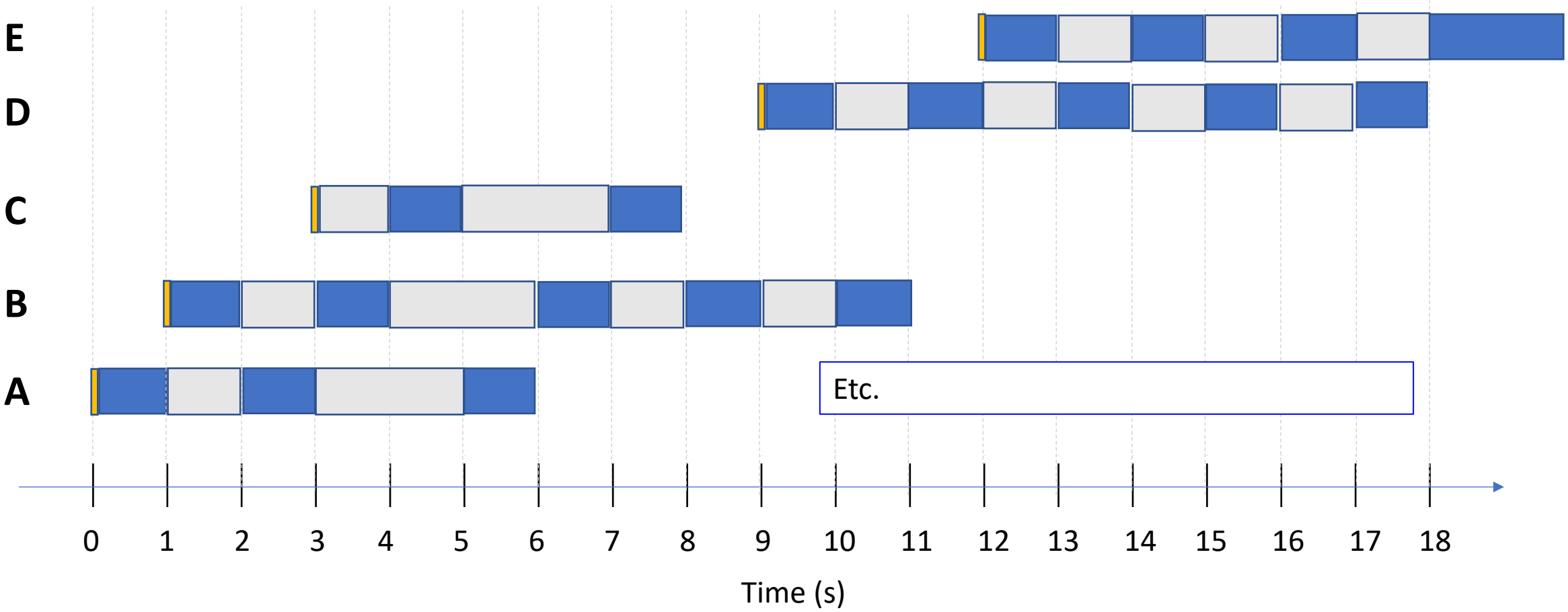
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



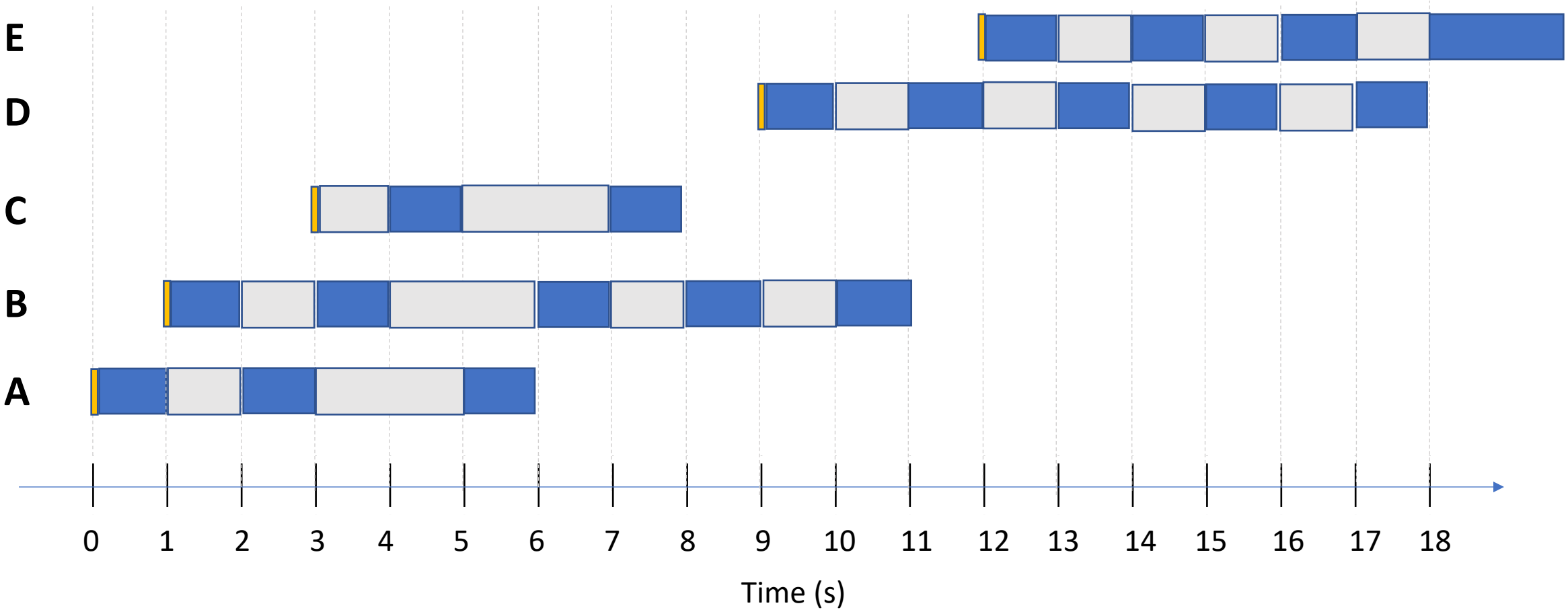
RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3		
B	1	5		
C	3	2		
D	9	5		
E	12	5		



RR

Proc	Arrival time	Execution time	Turnaround time	Response time
A	0	3	6	0
B	1	5	10	0
C	3	2	5	1
D	9	5	9	0
E	12	5	8	0



Summary – Key Concepts

- Process
  - Program in execution
- Linux process tree
  - Created by fork() / exec() / wait() / exit()
- Process switch
  - Change of process using the CPU
  - Save and restore registers and other info
- Process scheduler
  - Decides which process to run next

Further Optional Reading

Operating Systems: Three Easy Pieces by R. & A. Arpaci-Dusseau

Chapters 3 – 7 (inclusive) https://pages.cs.wisc.edu/~remzi/OSTEP/

Credits:

Some slides adapted from the OS courses of Profs. Remzi and Andrea Arpaci-Dusseau (University of Wisconsin-Madison), Prof. Willy Zwaenepoel (University of Sydney), and Prof. Youjip Won (Hanyang University), Prof. Natacha Crooks (UC Berkeley).