Scheduling

OPERATING SYSTEMS COURSE

THE HEBREW UNIVERSITY

SPRING 2023

Outline

- **CPU Scheduling**
  - First come first serve (FCFS)
  - Shortest time first (SJF)
  - Shortest remaining time first (SRTF)
  - Priority scheduling (PS)
  - Round robin (RR)
  - Multi-level queue
  - Multi-level feedback queue
- **Parallel System Scheduler**

Scheduling Criteria

- CPU utilization (max)
  - keep the CPU as busy as possible
- Throughput (max)
  - number of processes that complete their execution per time unit
- Waiting time (min)
  - amount of time a process has been waiting in the ready queue
- Turnaround time (min)
  - amount of time to execute a particular process

Scheduling Criteria

First-Come First-Served Scheduling

First-Come First-Served Scheduling

- The process that requests the CPU first is allocated the CPU first.
  - The simplest CPU-scheduling algorithm
  - Implementation with a FIFO queue
    - Add to the tail of the queue
    - Remove from the head to the queue
  - The code is simple to write and understand

First-Come First-Served Scheduling

Process	Burst Time
P_1	10
P_2	1
P_3	1

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

- Waiting time for P_1 = 0; P_2 = 10 + cs ; P_3 = 11 + 2cs
- Average waiting time: (0 + 10+ 11 +3cs)/3 = 7 +cs

First-Come First-Served Scheduling

Process	Burst Time
P_1	10
P_2	1
P_3	1

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

- Waiting time for $P_1 = 0$; $P_2 = 10 + cs$; $P_3 = 11 + 2cs$
- Average waiting time: $(0 + 10 + 11 + 3cs)/3 = 7 + cs$

First-Come First-Served Scheduling

- Processes:

Process	Burst Time
A	8
B	4
C	9
D	5

- Gantt Chart :

- Performance:

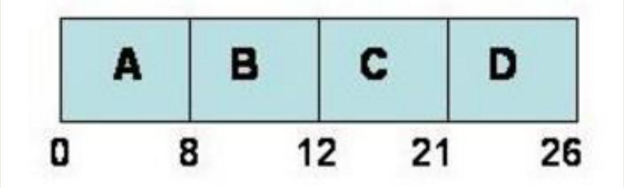
Metric (Avg)	FCFS
CPU Utilization	26/(26+3**cs**)
Turn around time	((8)+(12+cs)+(21+2cs)+(26+3cs))/4 = 16.75 +1.5cs
Waiting	((0)+(8+cs)+(12+2cs)+(21+3**cs)**)/4 = 10.25 + 1.5cs
Throughput	4/(26 + 3**cs**)

First-Come First-Served Scheduling

- Processes:

Process	Burst Time
A	8
B	4
C	9
D	5

- Gantt Chart :



- Performance:

Metric (Avg)	FCFS
CPU Utilization	26/(26+3**cs**)
Turn around time	((8)+(12+cs)+(21+2cs)+(26+3cs))/4 = 16.75 +1.5cs
Waiting	((0)+(8+cs)+(12+2cs)+(21+3**cs)**)/4 = 10.25 + 1.5cs
Throughput	4/(26 + 3**cs**)

First-Come First-Served Scheduling

- Fair scheduler
- Average waiting time is often quite long.
  - The problem: short processes wait after long processes
  - Provide good performance when the variance between the jobs is small
- FCFS is non-preemptive
  - Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU.

Shortest Job First (SJF) Scheduling

Shortest-Job-First Scheduling

- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
  - Another term – Shortest Job Next (SJN)
  - FCFS scheduling is used to break the tie
- Non-preemptive scheduler
- Provably optimal
  - Minimum average waiting time for a given set of processes [if all the jobs arrive at the same time, “offline” scheduling]
- Problems:
  - Jobs execution time is often unknown
  - Fairness (including starvation)

Shortest-Job-First Scheduling

Process	Burst Time
P_1	8
P_2	4
P_3	9
P_4	5

The Gantt Chart for the schedule is:

- Waiting time for $P_2 = 0; P_4 = 4; P_1 = 9; P_3 = 17;$
- Average waiting time: (0 + 4 + 9 + 17) / 4 = 7. 5
- Using FCFS scheme : (0 + 8 + 12 + 21) / 4 = 10.25

Ignoring context switch cost

Shortest-Job-First Scheduling

Process	Burst Time
P_1	8
P_2	4
P_3	9
P_4	5

The Gantt Chart for the schedule is:

P_2	P_4	P_1	P_3
0–4	4–9	9–17	17–26

0 4 9 17 26

- Waiting time for $P_2 = 0; P_4 = 4; P_1 = 9; P_3 = 17;$
- Average waiting time: $(0 + 4 + 9 + 17) / 4 = 7.5$
- Using FCFS scheme : $(0 + 8 + 12 + 21) / 4 = 10.25$

Ignoring context switch cost

Shortest-Job-First Scheduling

- Processes:

Process	Burst Time
A	8
B	4
C	9
D	5

- Gantt Chart:

- Performance:

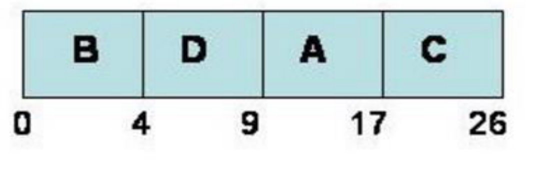
Metric (Avg)	SJF
Utilization	26/(26+3CS)
Turn around time	(4+9+CS+17+2CS+26+3CS)/4 = 14 + 1.5cs
Waiting	(0+4+CS+9+2CS+17+3CS)/4 = 7.5 + 1.5cs
Throughput	4/(26 + 3CS)

Shortest-Job-First Scheduling

- Processes:

Process	Burst Time
A	8
B	4
C	9
D	5

- Gantt Chart:



- Performance:

Metric (Avg)	SJF
Utilization	26/(26+3CS)
Turn around time	(4+9+CS+17+2CS+26+3CS)/4 = 14 + 1.5cs
Waiting	(0+4+CS+9+2CS+17+3CS)/4 = 7.5 + 1.5cs
Throughput	4/(26 + 3CS)

Shortest Remaining Time First (SRTF)

Shortest Remaining Time First (SRTF)

- Often called Shortest Remaining Time.
- A preemptive variant of SJF
  - If a new process arrives with CPU burst length less than remaining time of current executing process, preempt.
  - Short processes are handled very quickly
- Same problems as SJF.
  - Jobs execution time is often unknown
  - Fairness (including starvation)

Example of SRTF

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- Gantt Chart:

Ignoring context switch cost

- Average waiting time = ((10-1) + (1-1) + (17-2) + (5-3)) / 4 = 6.5

Example of SRTF

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- Gantt Chart:

P_1	P_2	P_4	P_1	P_3	
0	1	5	10	17	26

Ignoring context switch cost

- Average waiting time = ((10-1) + (1-1) + (17-2) + (5-3)) / 4 = 6.5

Priority Scheduling

Priority Scheduling

- The CPU is allocated to the process with the highest priority.
  - A priority is associated with each process
    - Fixed range of number, such as 0 to 7
    - We use low numbers to represent high priority
  - Equal-priority processes are scheduled in FCFS order
  - SJF is a special case of the general priority-scheduling algorithm
    - The priority is the predicted next CPU burst

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Gantt Chart:

- Average waiting time = (6 + 0 + 16 + 18 + 1) / 5 = 8.2

Ignoring context switch cost

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Gantt Chart:

P_2	P_5	P_1	P_3	P_4
0–1	1–6	6–16	16–18	18–19

0 1 6 16 18 19

Ignoring context switch cost

- Average waiting time = (6 + 0 + 16 + 18 + 1) / 5 = 8.2

Priority Scheduling

- Priorities can be defined
  - Internally
    - Use some measurable quantity
      - Time limits, memory requirements…
  - Externally
    - Set by criteria external to OS
      - importance, political factors

- Priority scheduling can be
  - Preemptive
  - Non-preemptive

- Major problem
  - Indefinite blocking or starvation
  - Solution: aging
    - Gradually increase the priority of processes that wait for a long time

Round-Robin Scheduling

Round-Robin Scheduling

- A small unit of time, called a time quantum is defined.
  - Generally from 10 to 100 milliseconds
- The ready queue is treated as a circular, FIFO queue.
- The CPU scheduler goes around the ready queue
  - Allocate the CPU to each process for a time interval of up to 1 time quantum.
- Two cases
  - CPU burst less than 1 time quantum
    - The process released the CPU voluntarily
  - CPU burst longer than 1 time quantum
    - Context switch will be executed

Example of Round-Robin Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Round-robin scheduling
  - A time-quantum of 4 milliseconds
- Average waiting time = (6 + 4 + 7) / 3 = 5.66
- RR scheduling is preemptive.

Example of Round-Robin Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Round-robin scheduling
  - A time-quantum of 4 milliseconds

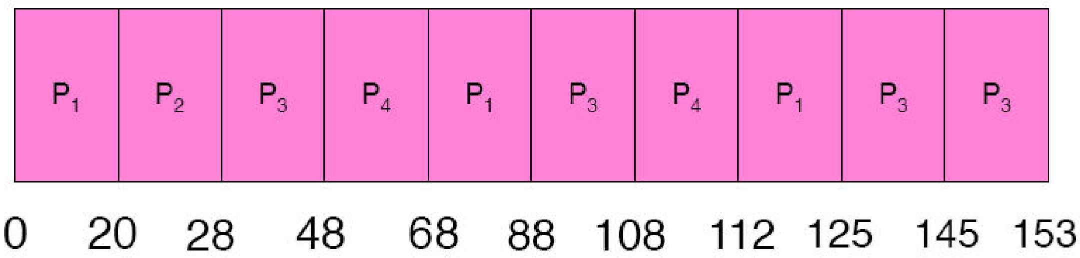
P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

- Average waiting time = (6 + 4 + 7) / 3 = 5.66
- RR scheduling is preemptive.

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24



- Waiting Time:
  - P1: 0+(68-20)+(112-88) = 72
  - P2: (20-0) = 20
  - P3: (28-0)+(88-48)+(125-108) = 85
  - P4: (48-0)+(108-68) = 88
- Completion Time (Turn Around Time):
  - P1: 125
  - P2: 28
  - P3: 153
  - P4: 112
- Average Waiting Time: (72+20+85+88)/4 = 66.25
- Average Completion Time: (125+28+153+112)/4 = 104.5

Example of Round-Robin Scheduling

- Processes:
- Quantum=3

Process	Burst Time
A	8
B	4
C	9
D	5

- Gantt Chart :

- Performance:

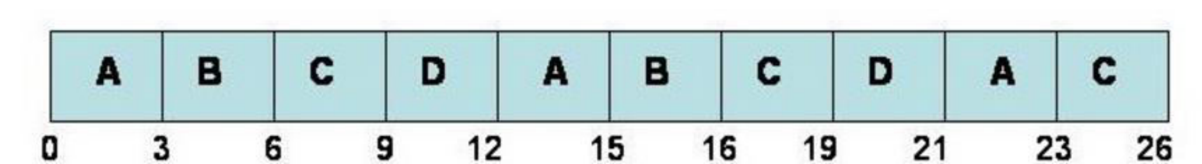
Metric	RR
Utilization	26/(26+9CS)
Avg Turn around time	(23+16+26+21+29cs)/4 = 21.5 ignoring CS
Avg Waiting	(15+12+17+16+29cs)/4 = 15 ignoring CS
Throughput	4/(26 + 9CS)

Example of Round-Robin Scheduling

- Processes:
- Quantum=3

Process	Burst Time
A	8
B	4
C	9
D	5

- Gantt Chart :



- Performance:

Metric	RR
Utilization	26/(26+9CS)
Avg Turn around time	(23+16+26+21+29cs)/4 = 21.5 ignoring CS
Avg Waiting	(15+12+17+16+29cs)/4 = 15 ignoring CS
Throughput	4/(26 + 9CS)

Example of Round-Robin Scheduling

- Processes:
- Quantum=6

Process	Burst Time
A	8
B	4
C	9
D	5

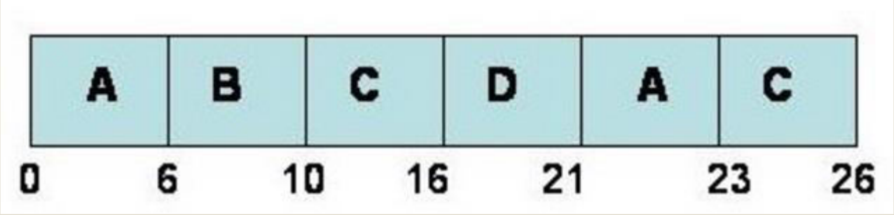
- Gantt Chart :

Example of Round-Robin Scheduling

- Processes:
- Quantum=6

Process	Burst Time
A	8
B	4
C	9
D	5

- Gantt Chart :



Comparing FCFS and RR

- Assuming zero-cost context switching time, is RR always better than FCFS?
- Assume 10 jobs, all start at the same time, and each require 100 seconds of CPU time
- RR scheduler quantum of 1 second
- Completion Times (CT)
  - Both FCFS and RR finish at the same time
  - But average response time is much worse under RR!

Bad when all jobs are same length

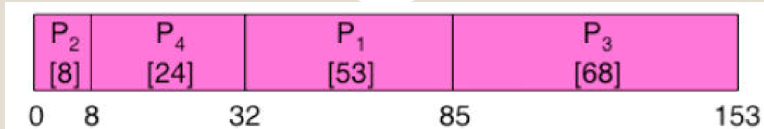
Job #	FCFS CT	RR CT
1	100	991
2	200	992
…	…	…
9	900	999
10	1000	1000

RR performance

- Run time distribution:
  - RR is poor if the jobs variance is small
  - RR is good for "real life"

- Context switch hurts the performance!
  - Context switch's time
  - Cache state must be shared between all jobs with RR
    - Total time for RR longer even for zero-cost context switch!

Comparing FCFS and RR



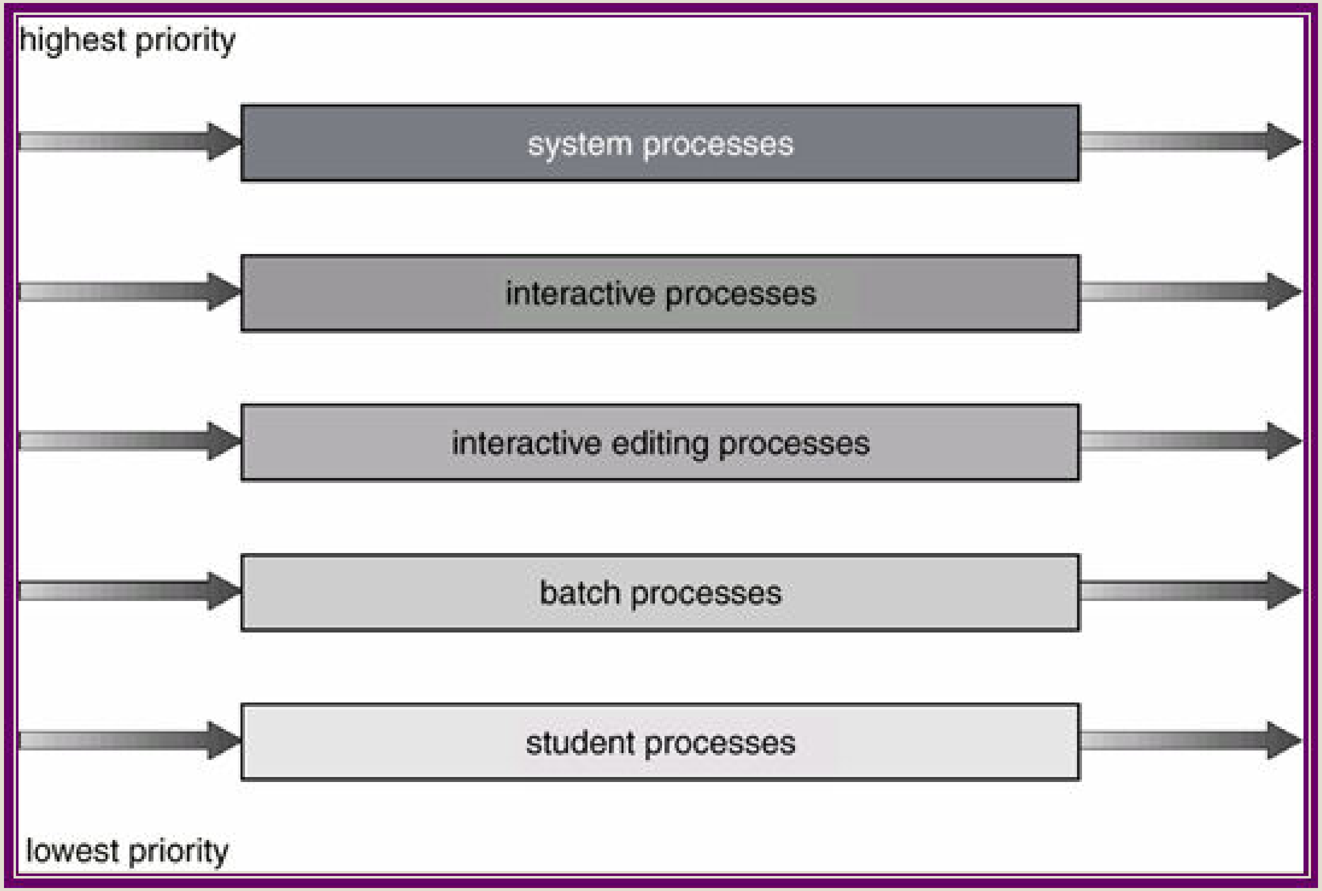
	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Best FCFS	32	0	85	8	$31\frac{1}{4}$
	Q = 1	84	22	85	57	62
	Q = 5	82	20	85	58	$61\frac{1}{4}$
	Q = 8	80	8	85	56	$57\frac{1}{4}$
	Q = 10	82	10	85	68	$61\frac{1}{4}$
	Q = 20	72	20	85	88	$66\frac{1}{4}$
	Worst FCFS	68	145	0	121	$83\frac{1}{2}$
Completion Time	Best FCFS	85	8	153	32	$69\frac{1}{2}$
	Q = 1	137	30	153	81	$100\frac{1}{2}$
	Q = 5	135	28	153	82	$99\frac{1}{2}$
	Q = 8	133	16	153	80	$95\frac{1}{2}$
	Q = 10	135	18	153	92	$99\frac{1}{2}$
	Q = 20	125	28	153	112	$104\frac{1}{2}$
	Worst FCFS	121	153	68	145	$121\frac{3}{4}$

Multilevel Queue Scheduling

Multilevel Queue Scheduling

- Multilevel queue-scheduling algorithm
  - Partition the ready queue into several separate groups
  - Each group has its own scheduling algorithm
  - Scheduling among the queues
    - Fixed-priority preemptive scheduling
      - Possibility of starvation
    - Time-slice between the queues
      - A certain portion of the CPU time
- Processes are classified into different groups
  - Foreground (or interactive) processes
  - Background (or batch) processes

Multilevel Queue Scheduling



Multilevel Feedback Queue

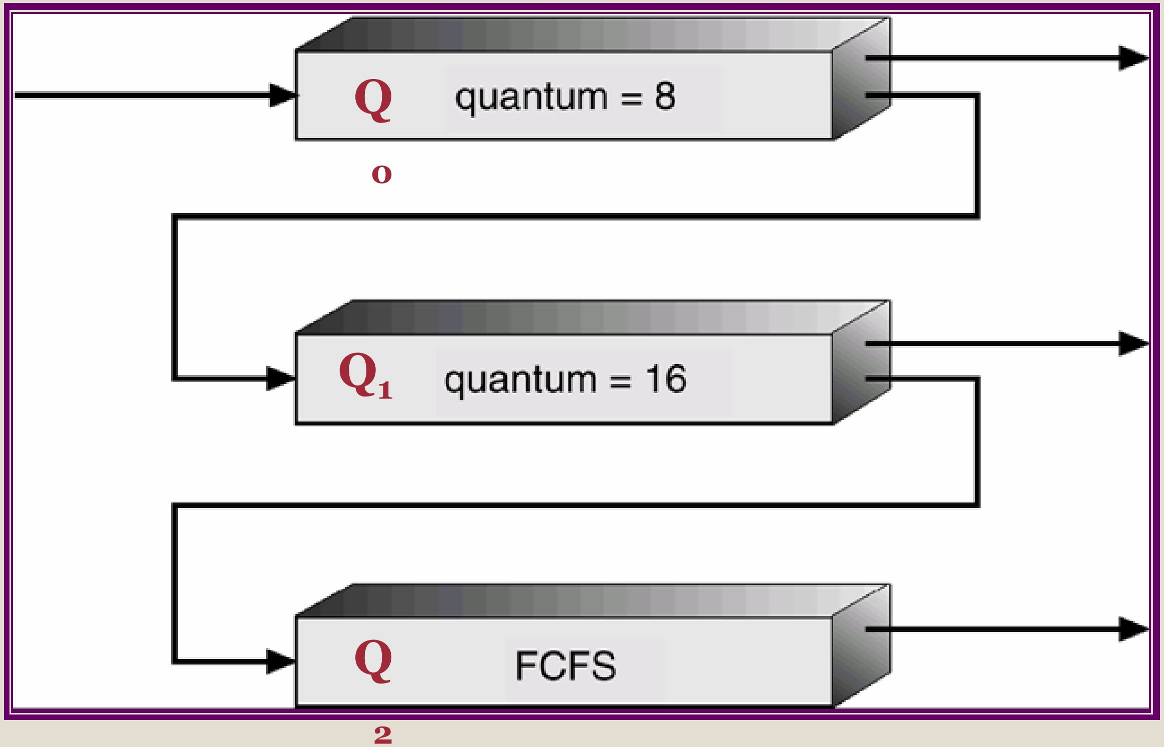
- A process can move between the various queues
- For example:
  - Use too much CPU time: move to a lower-priority queue
  - Wait too long: move to a higher-priority queue
    - This form of aging prevents starvation
- Multilevel-feedback-queue scheduler defined by:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine which queue a process will enter when that process needs service
  - Method used to determine when to upgrade a process
  - Method used to determine when to demote a process

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – time quantum 8 milliseconds
  - Q_1 – time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

Multilevel feedback queues

- The most general scheme



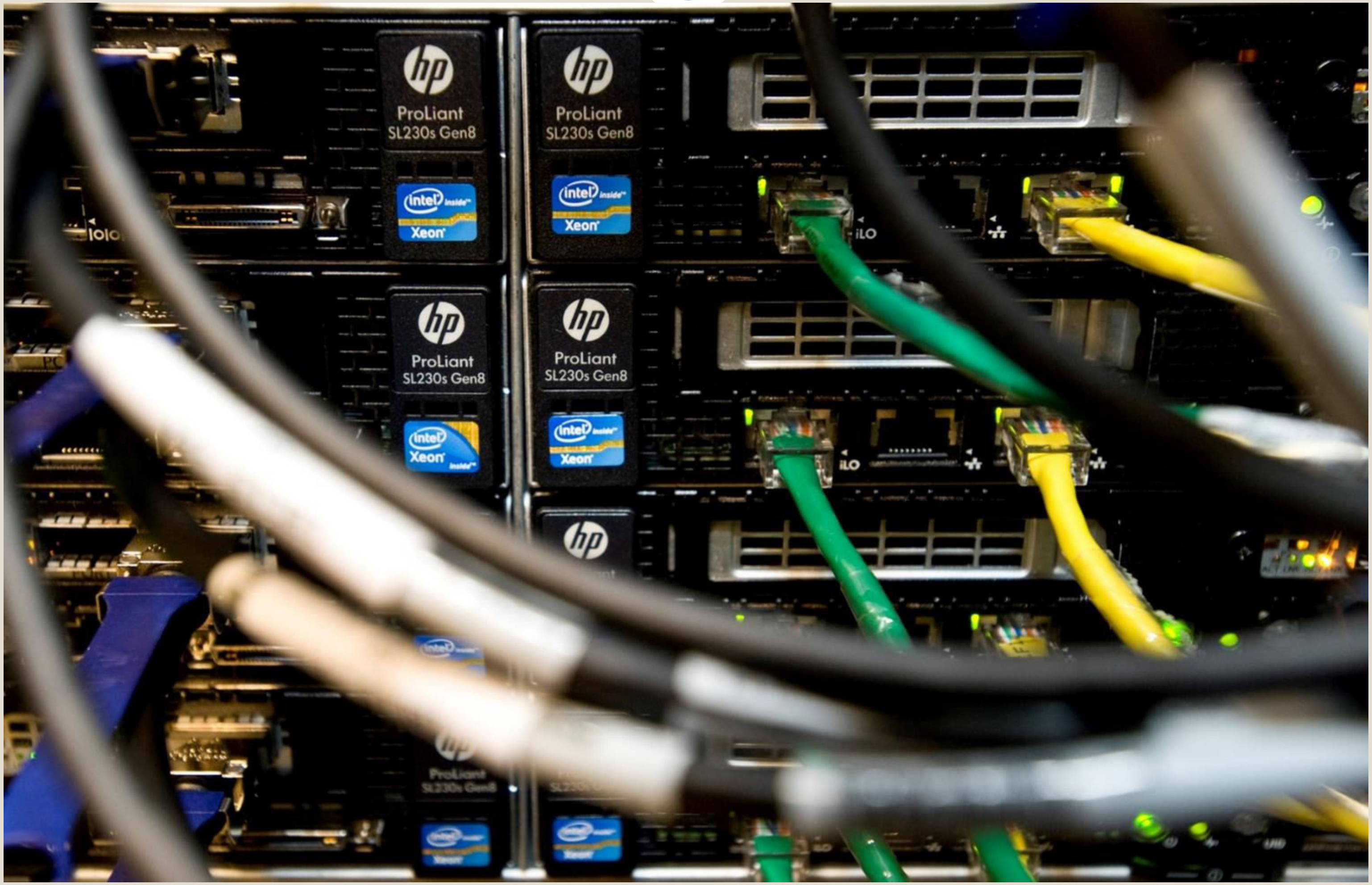
Outline

- CPU Scheduling
- **Parallel Systems Scheduling**
  - Schedulers
  - Performance Evaluation

Parallel System

Parallel System

Parallel System



Supercomputers

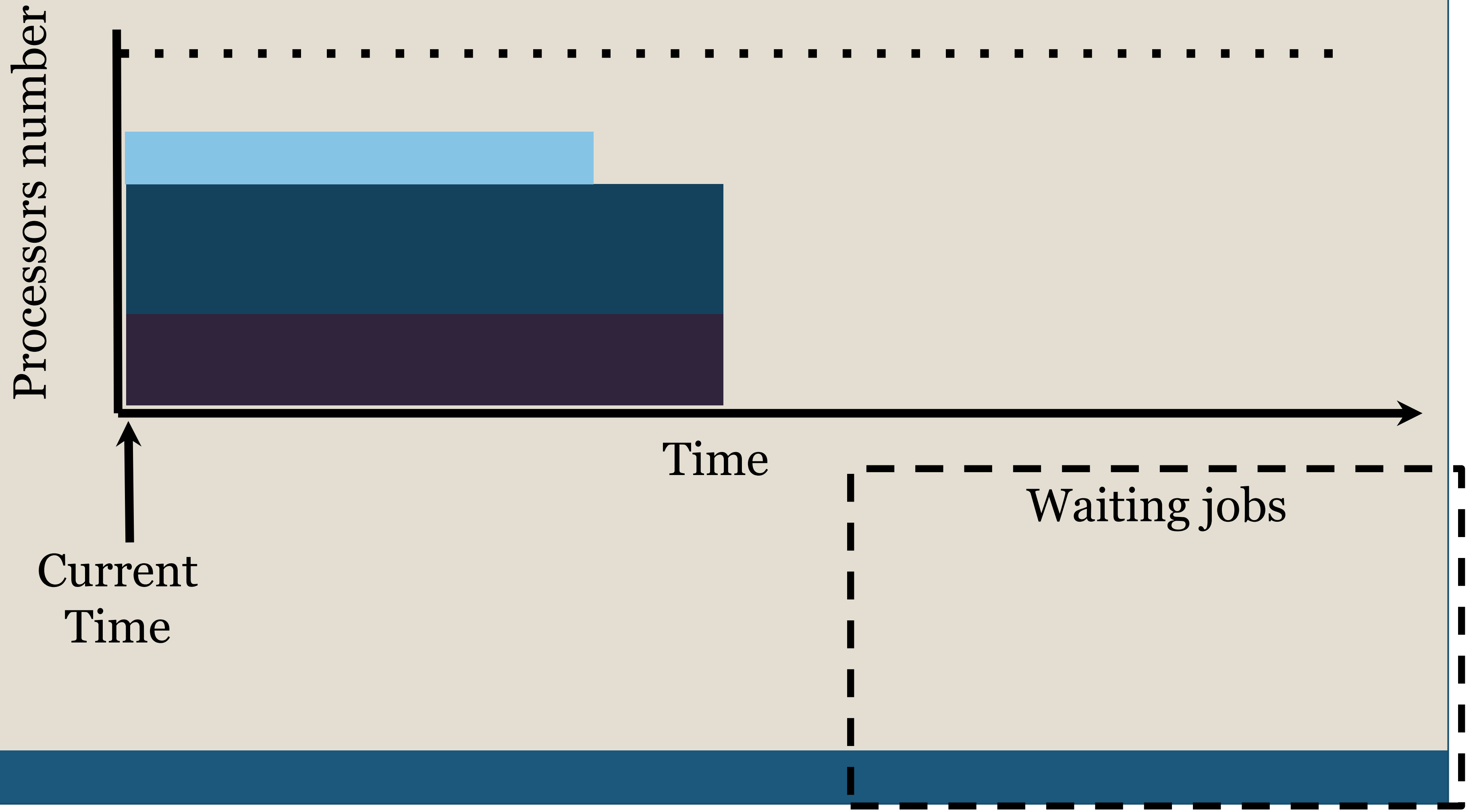
- Provide extremely high speed of calculation (HPC)
- Each computer has a basic OS.
- The computers are connected with high speed network
- Regularly run parallel jobs
- Each job contain the required number of processors
- No preemption.

Supercomputers' Scheduler

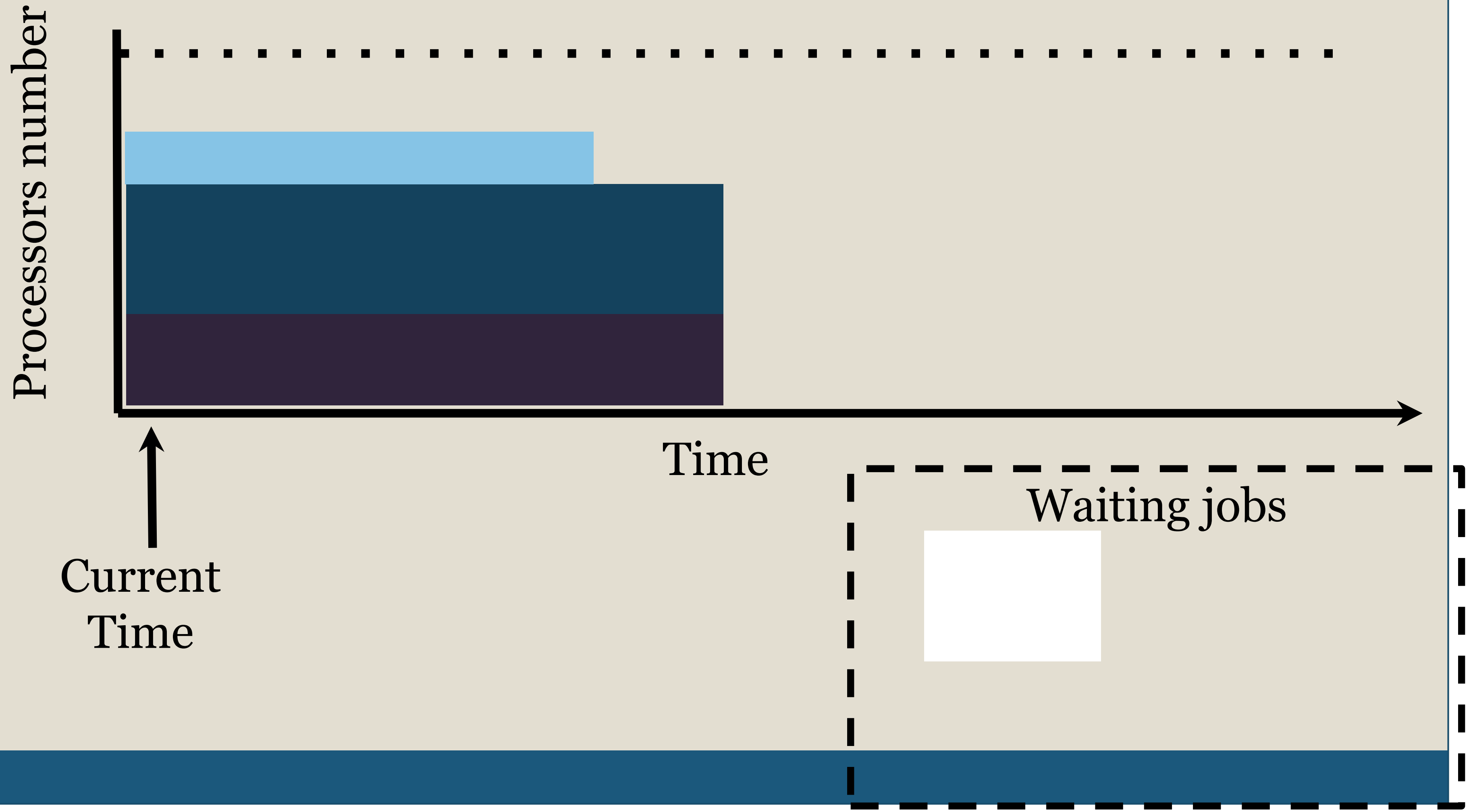
- The scheduler's purpose is to determine which processors will be allocated to each job.

- Example of schedulers: FCFS, SJF, Backfilling schedulers.

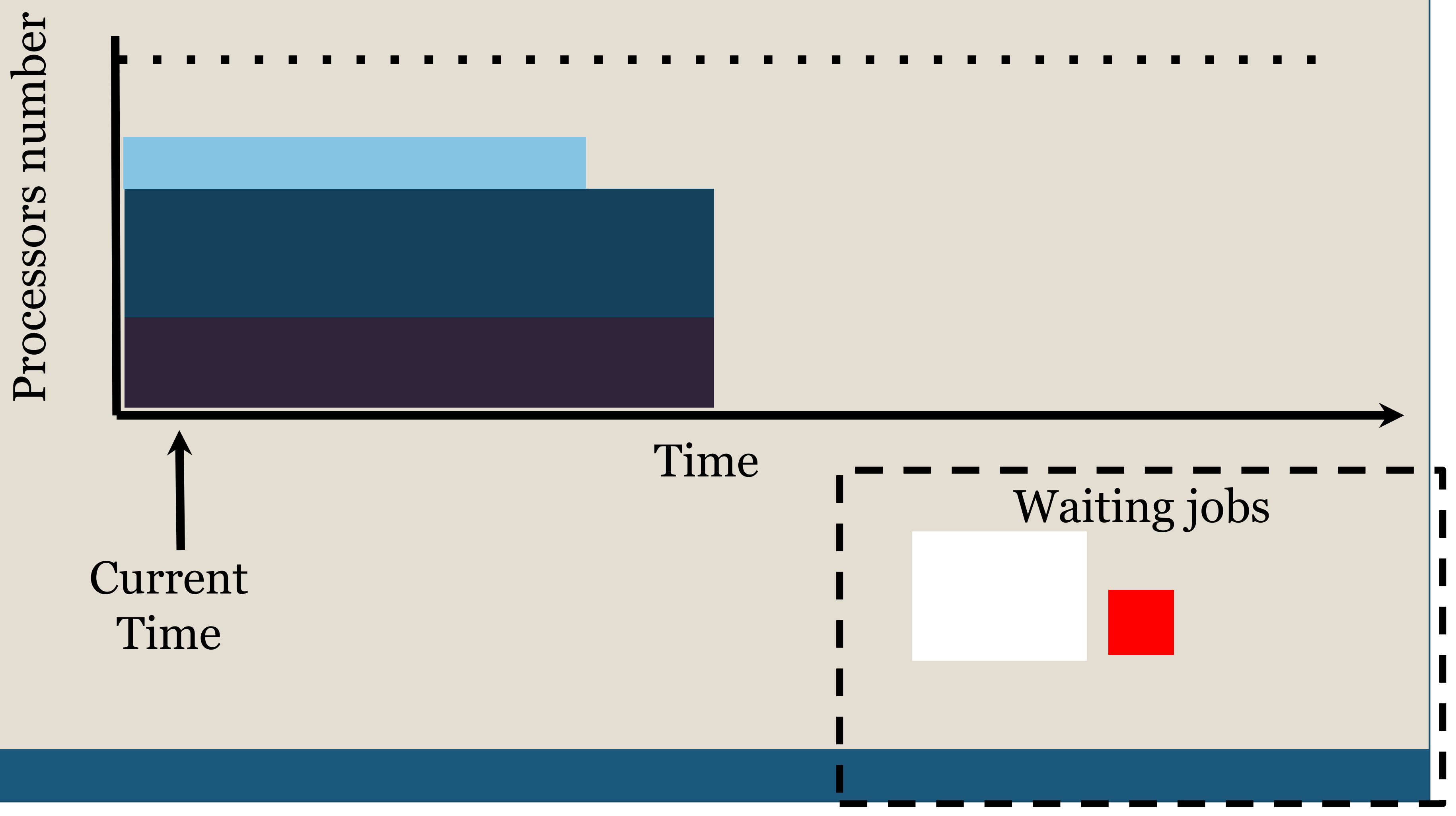
FCFS Scheduler



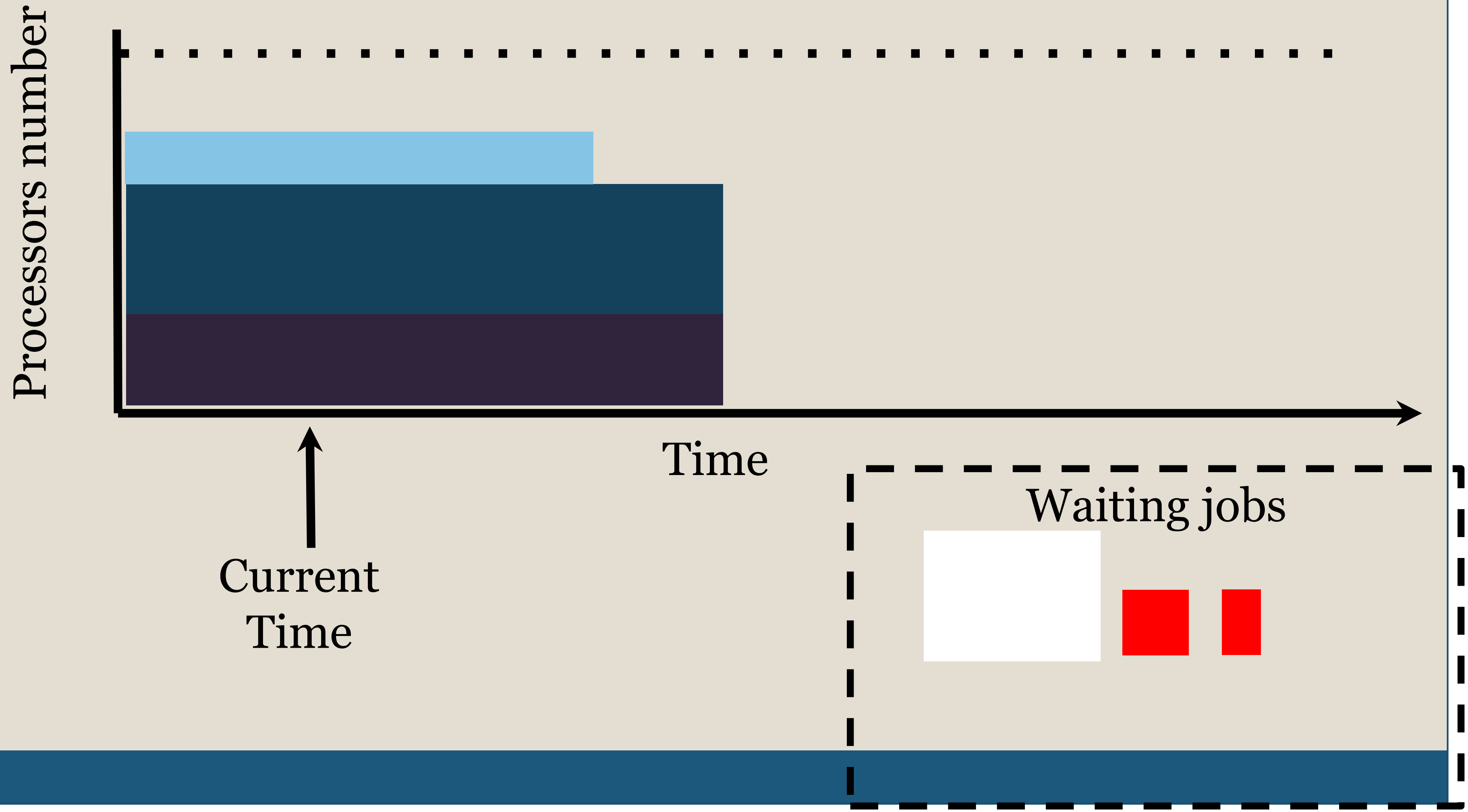
FCFS Scheduler



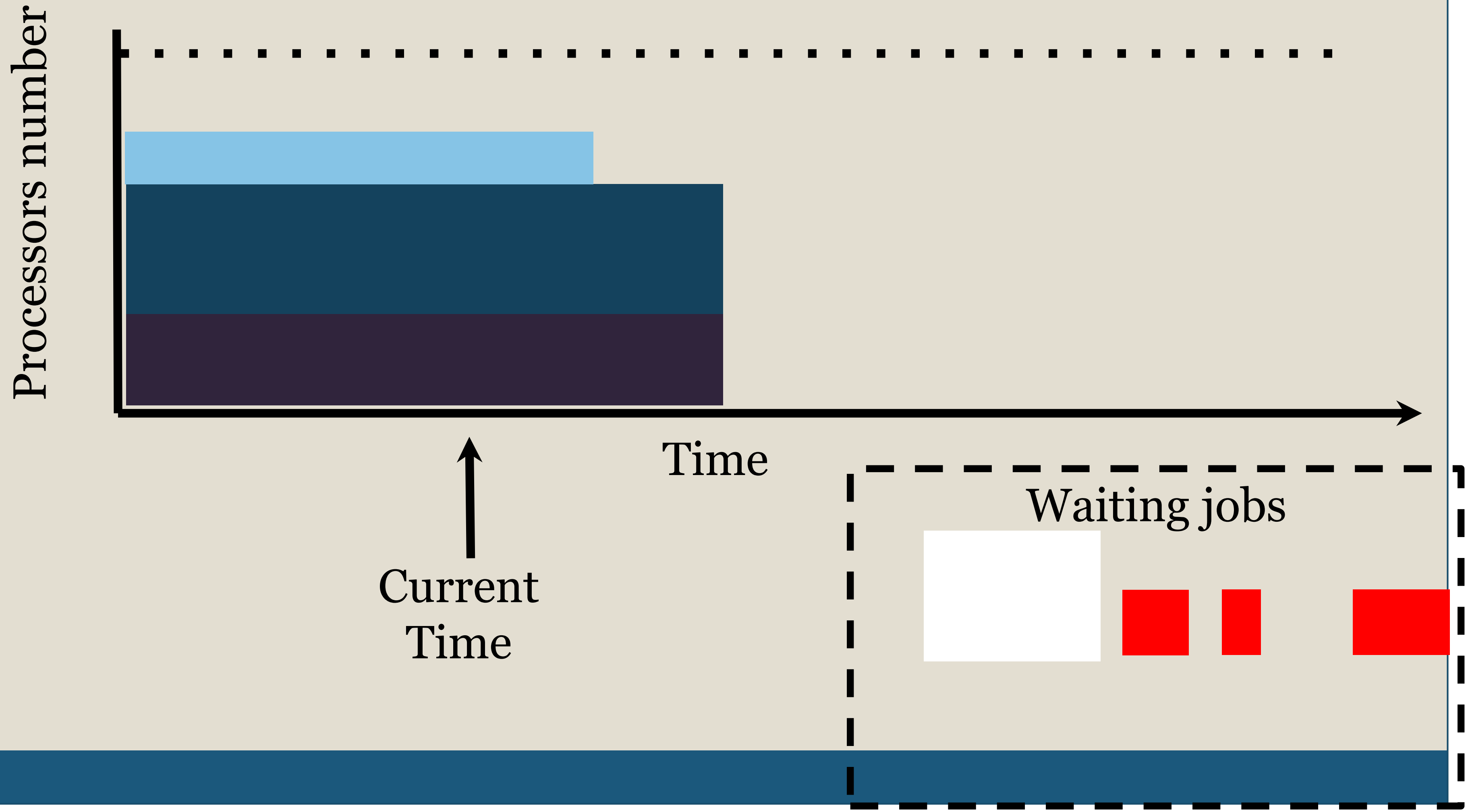
FCFS Scheduler



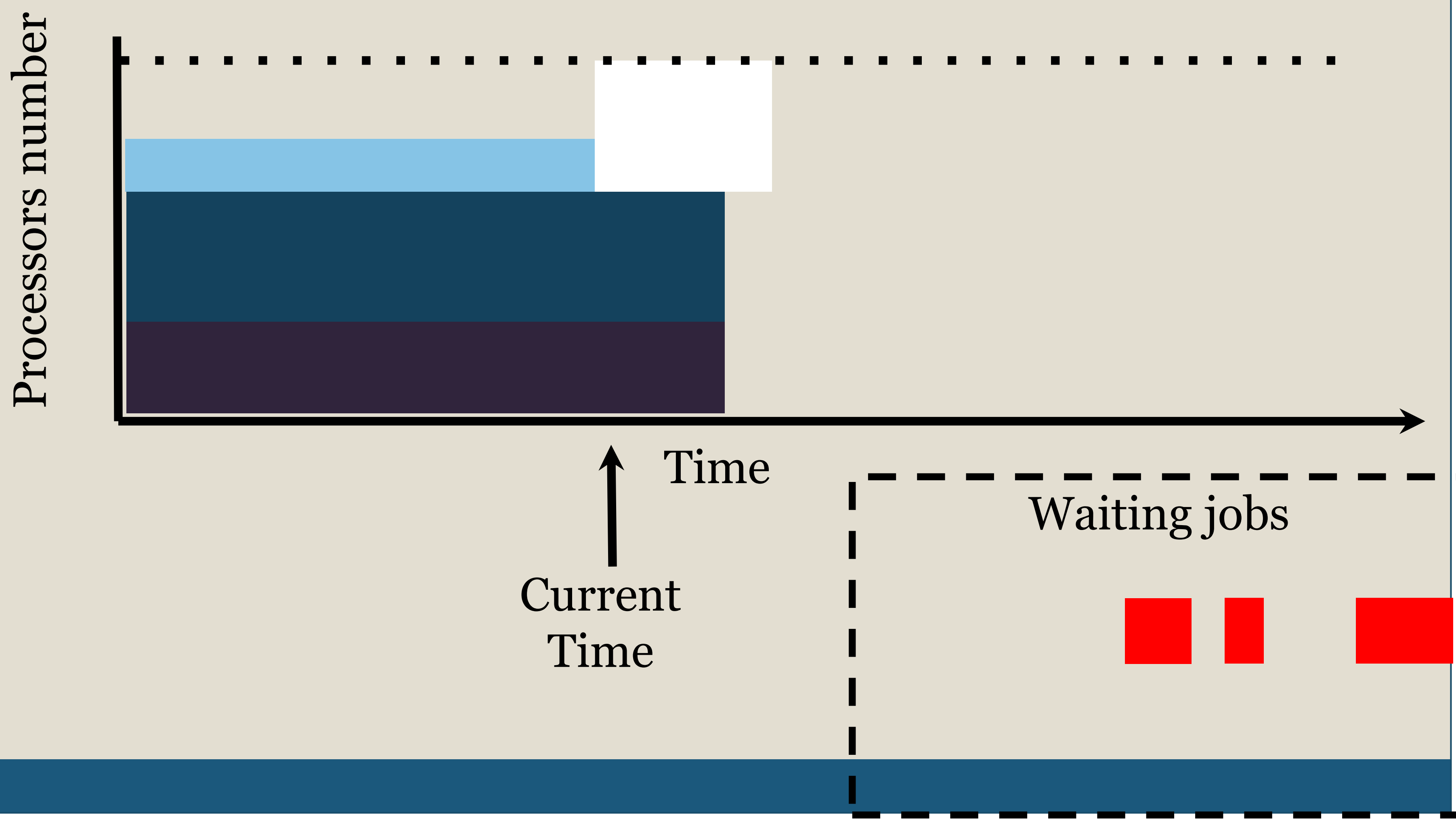
FCFS Scheduler



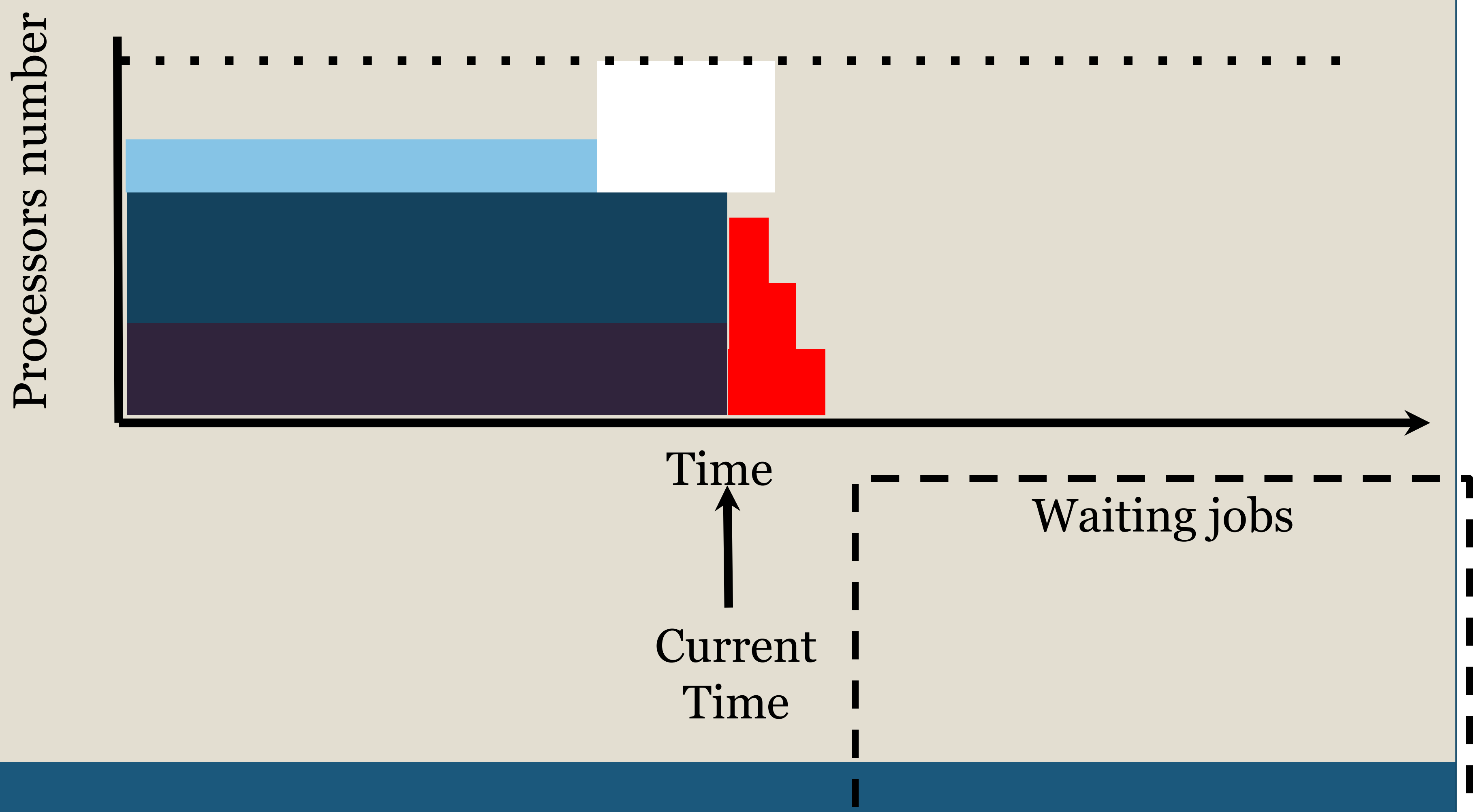
FCFS Scheduler



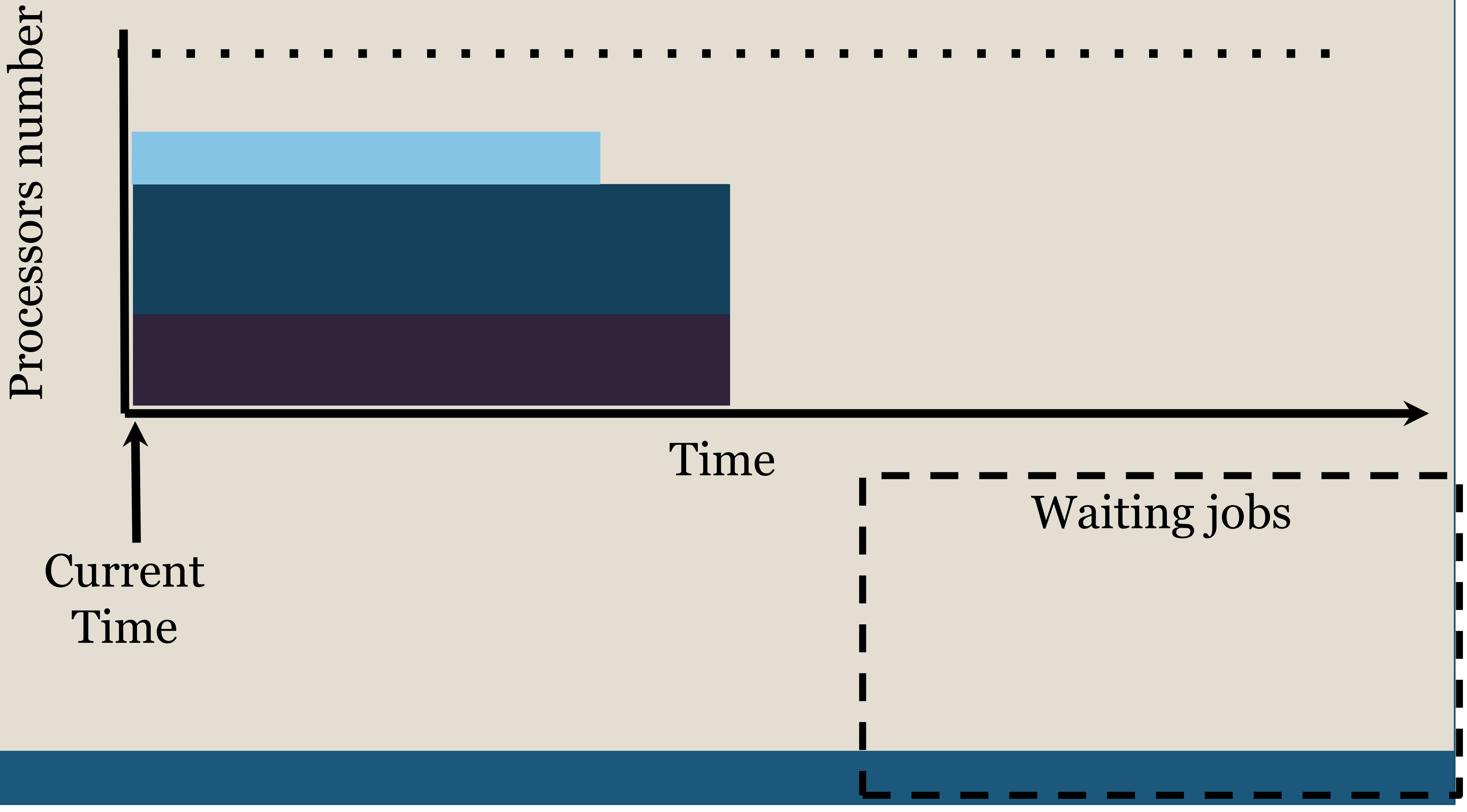
FCFS Scheduler



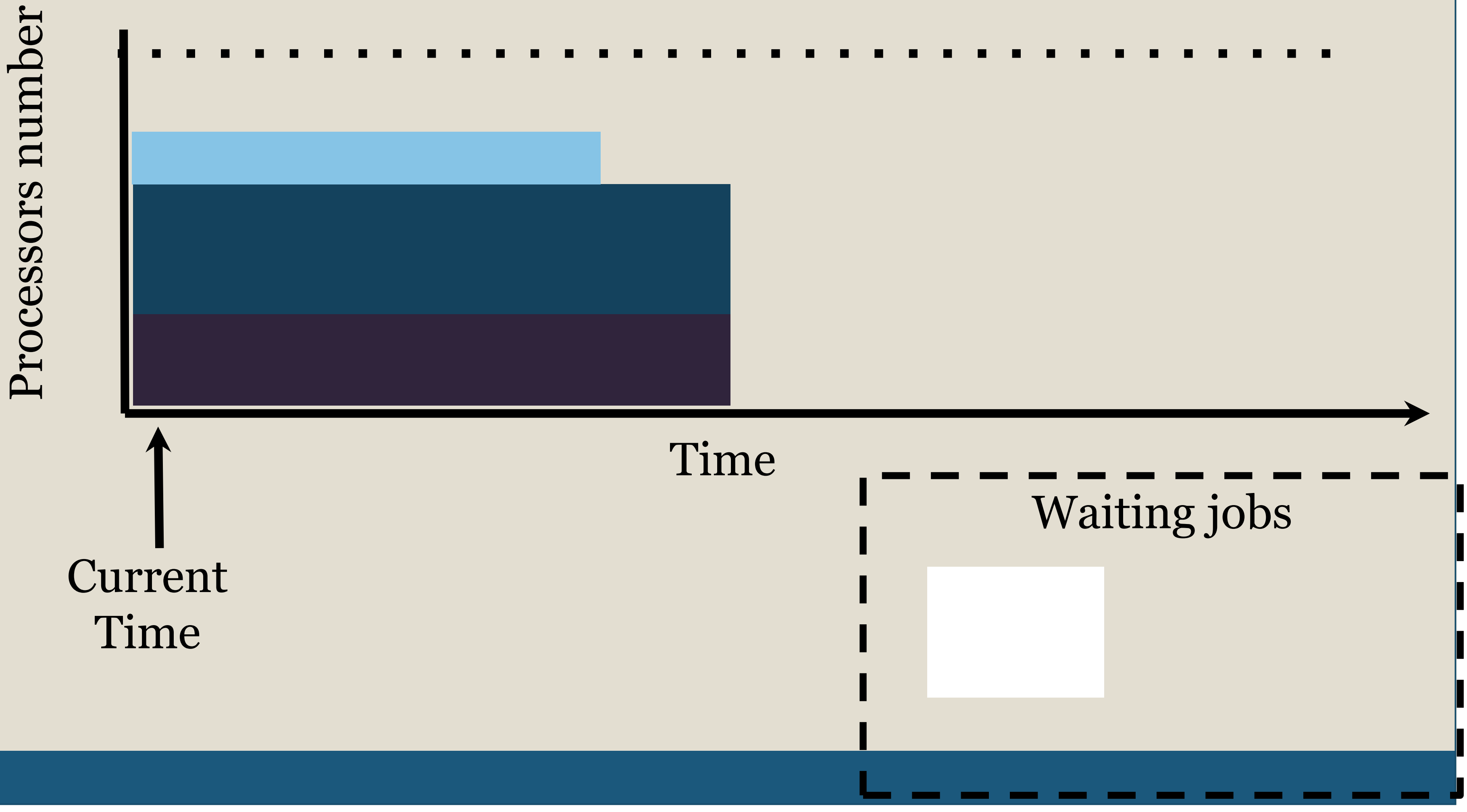
FCFS Scheduler



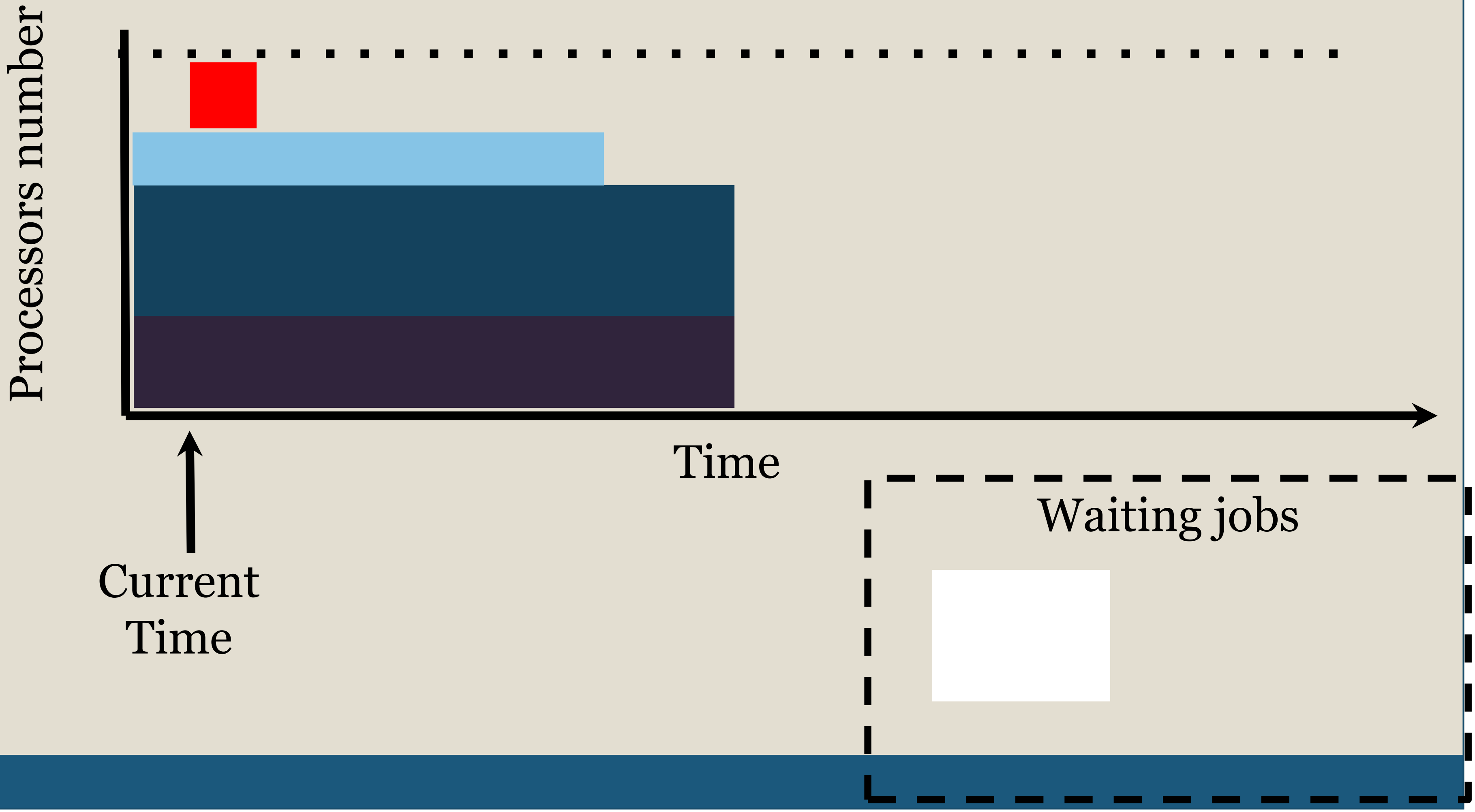
Scheduler with Backfilling



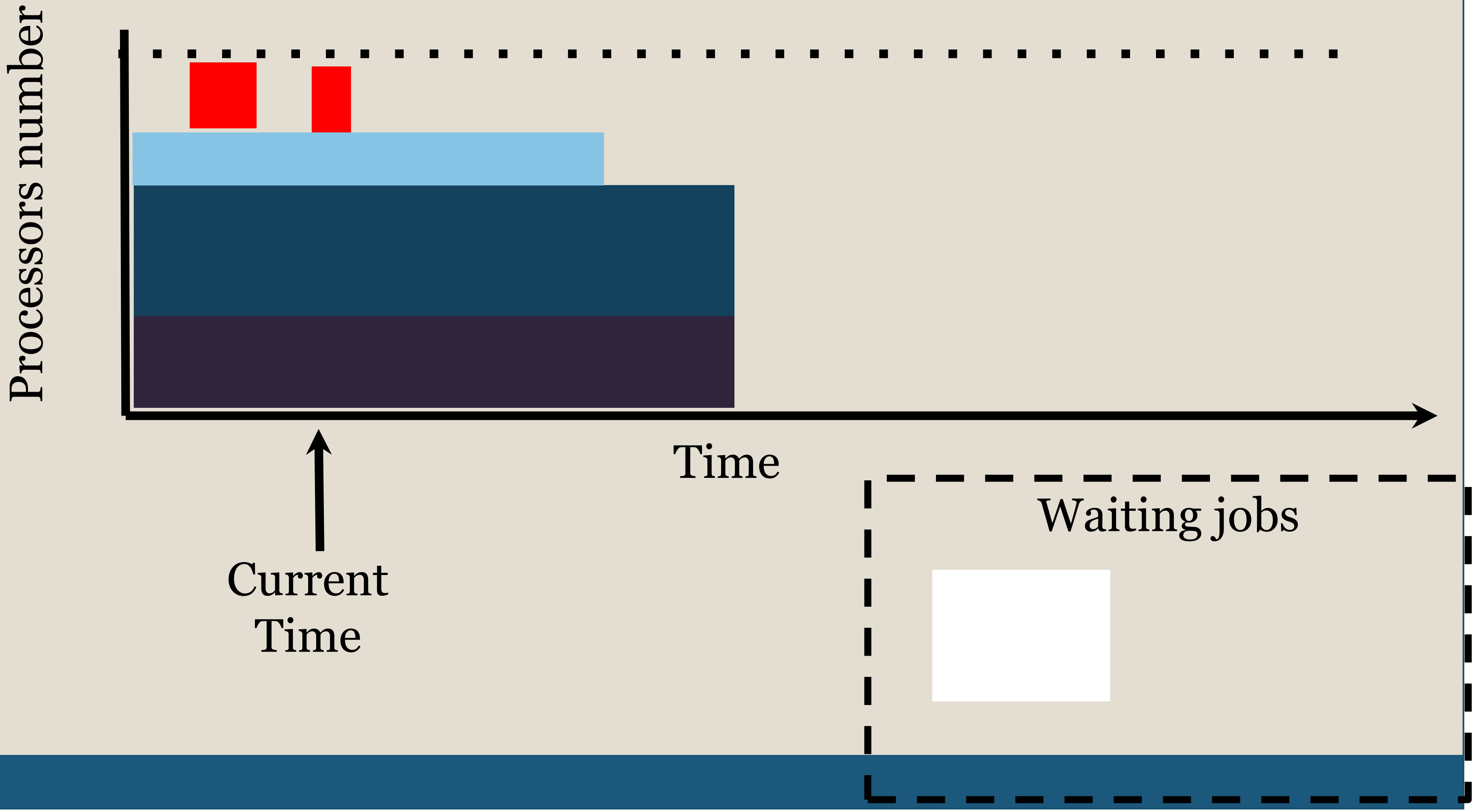
Scheduler with Backfilling



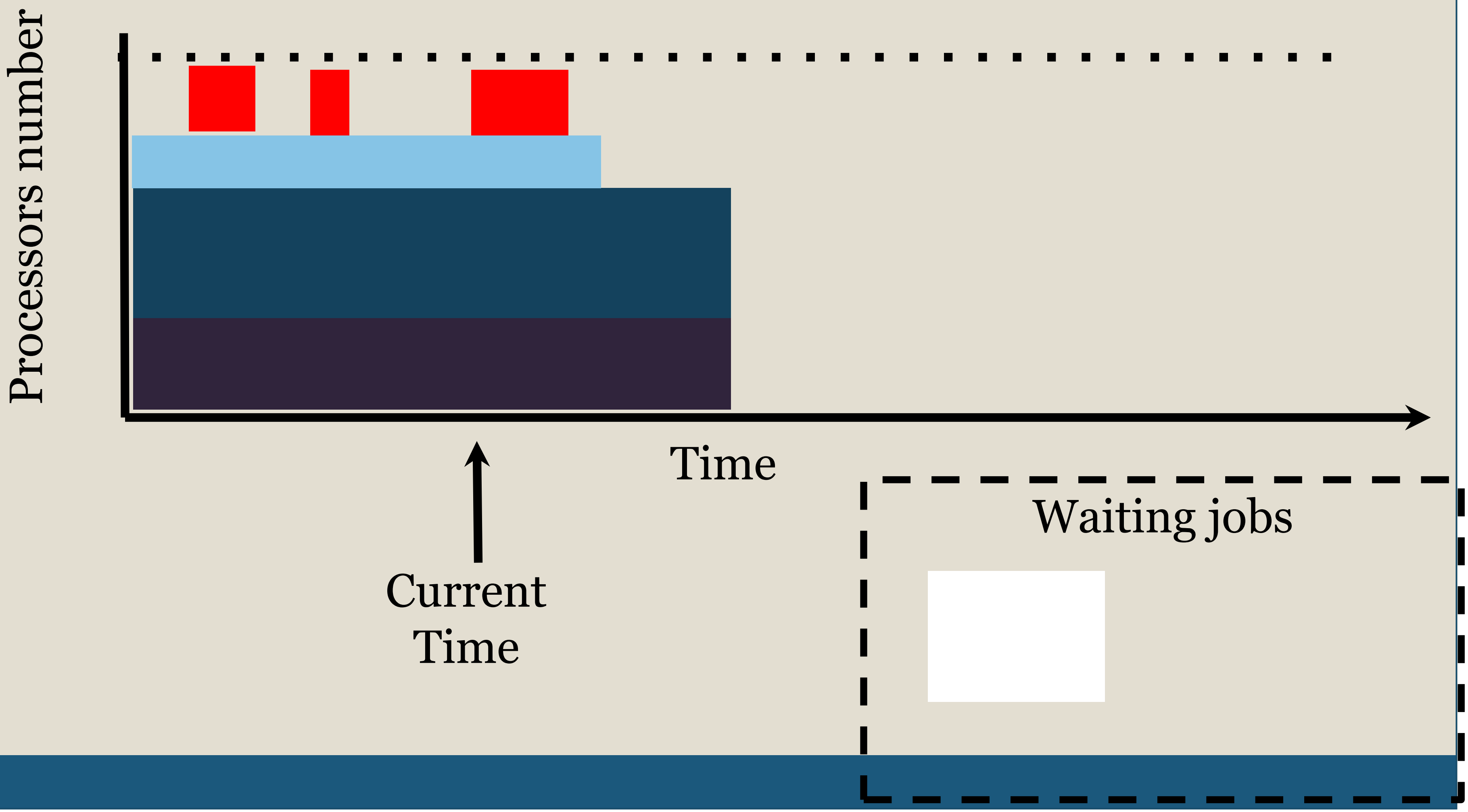
Scheduler with Backfilling



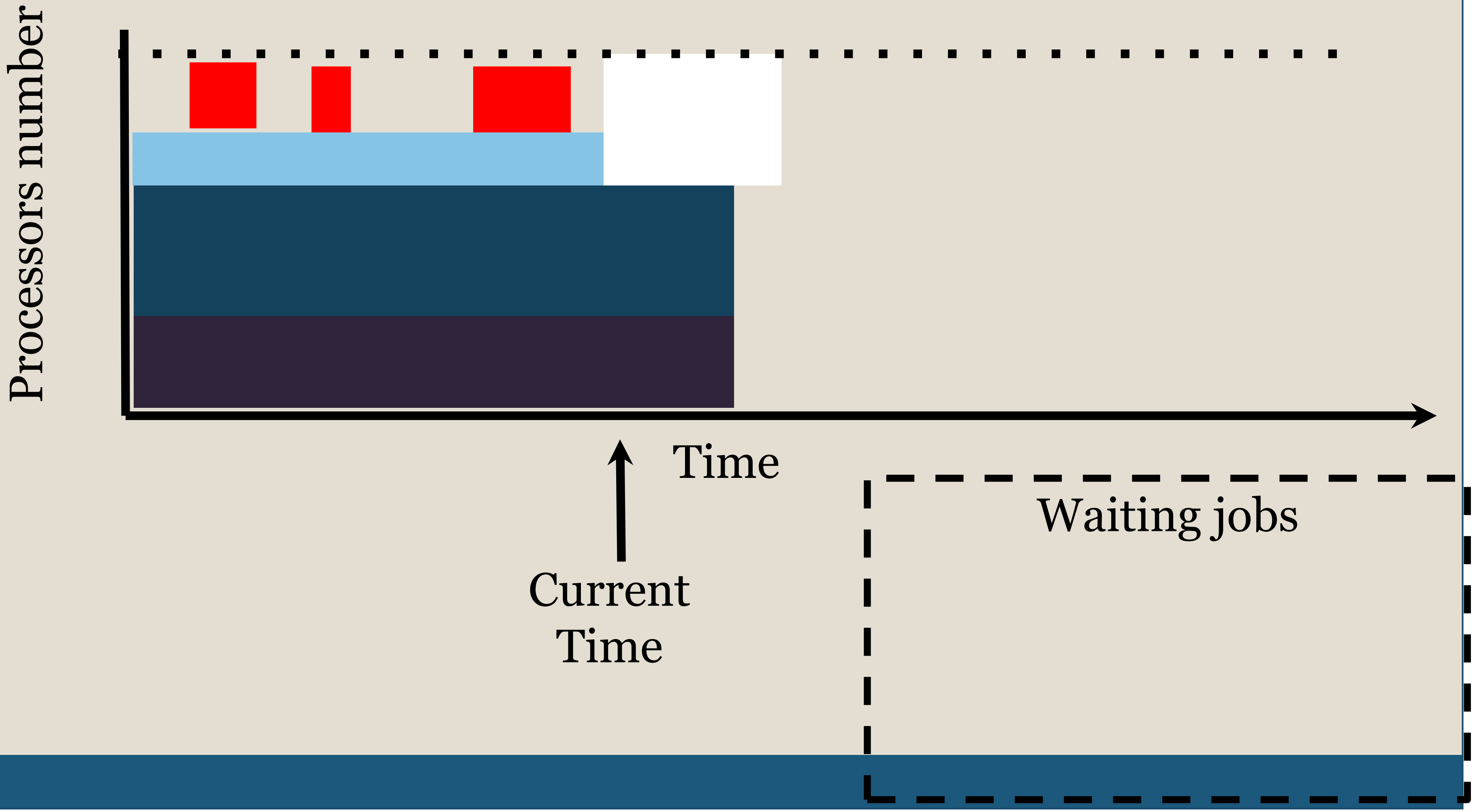
Scheduler with Backfilling



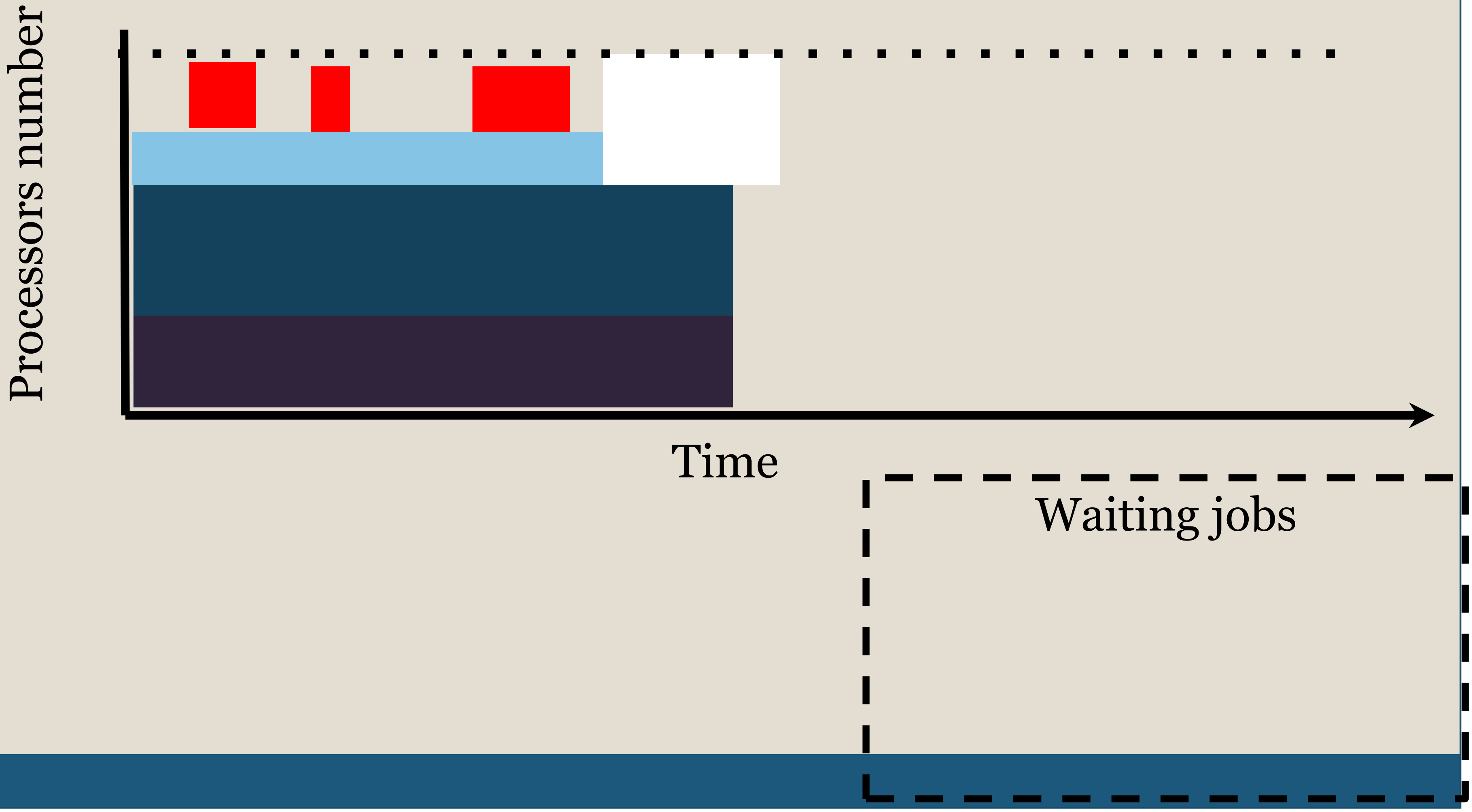
Scheduler with Backfilling



Scheduler with Backfilling



Scheduler with Backfilling

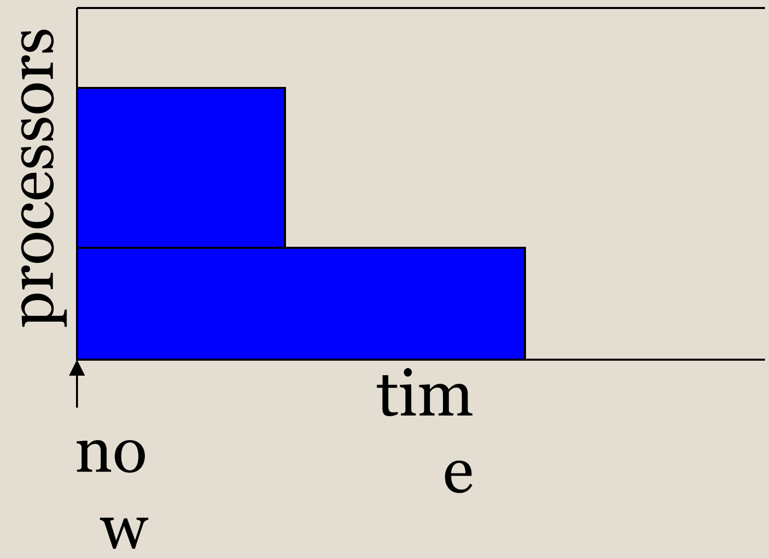


The EASY Scheduler

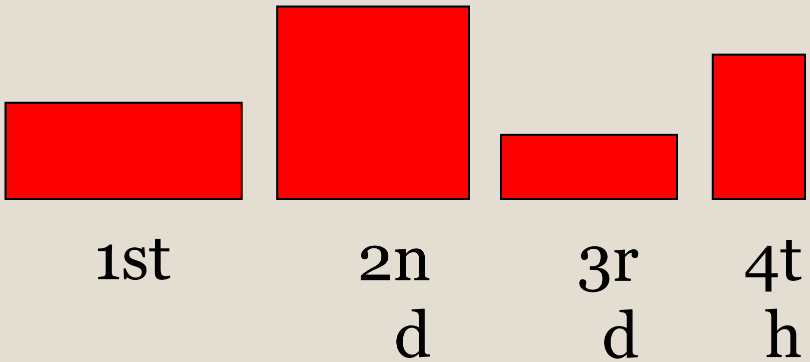
- Very common and simple algorithm
- Relatively fair
- Uses backfilling based on arrival time.

EASY Data Structures

- List of running jobs
  - Number of processors they use
  - Expected termination

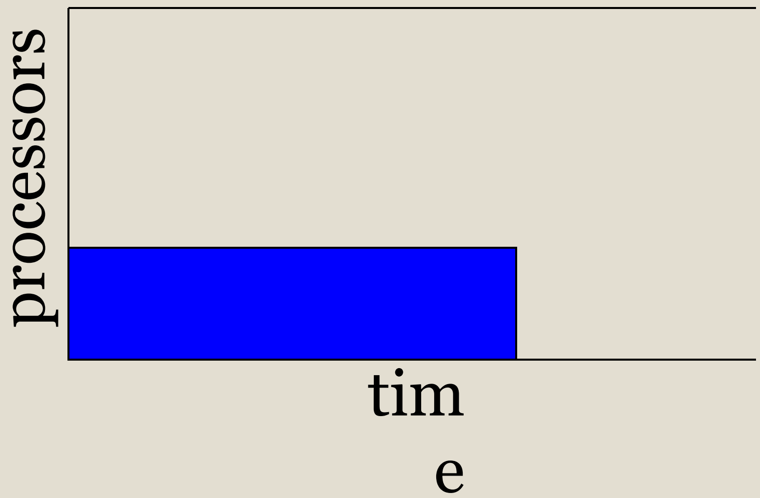


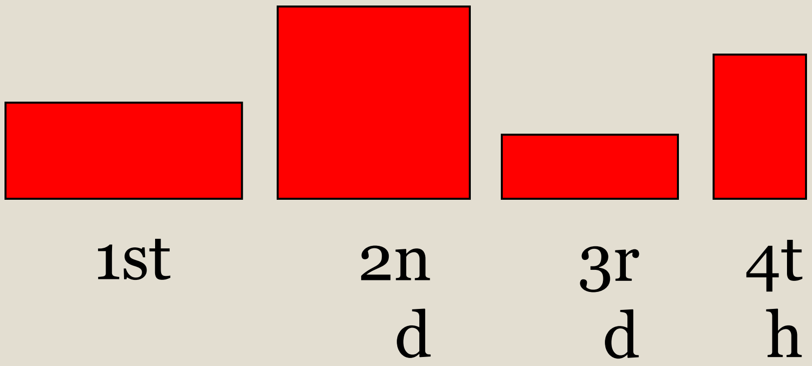
- List of queued jobs
  - How many processors they need
  - How long they are expected to run
  - Sorted in order of arrival



EASY Operation

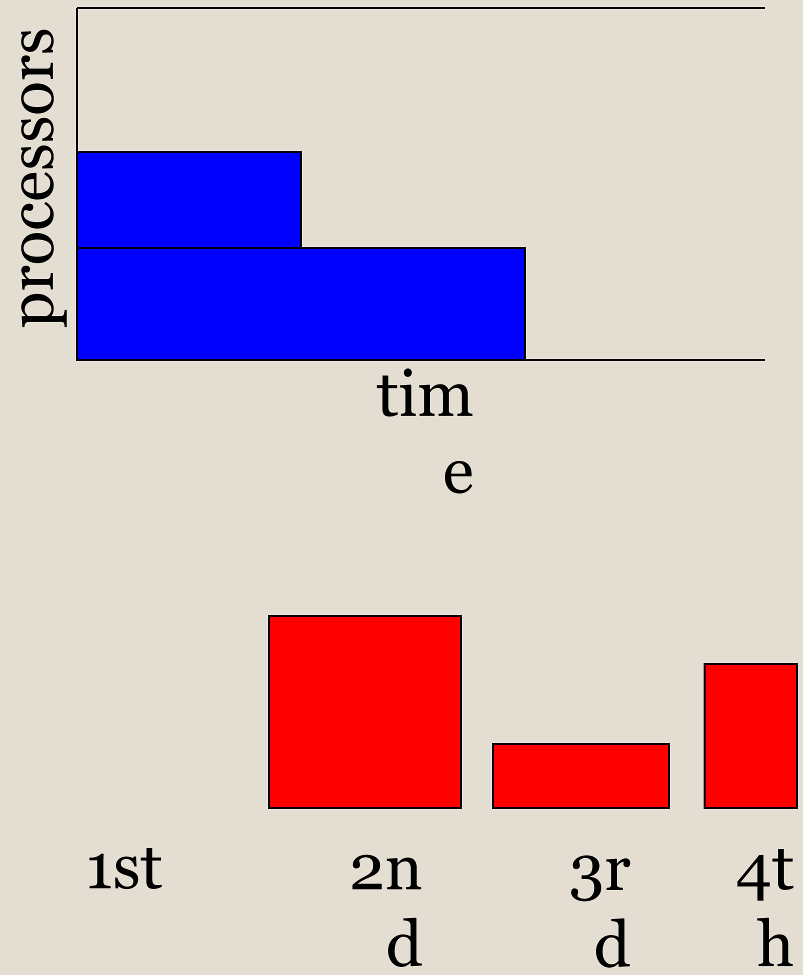
1. Schedule jobs on available processors in FCFS order
2. Make reservation for first job that cannot run
3. Schedule additional jobs provided they do not conflict with this reservation





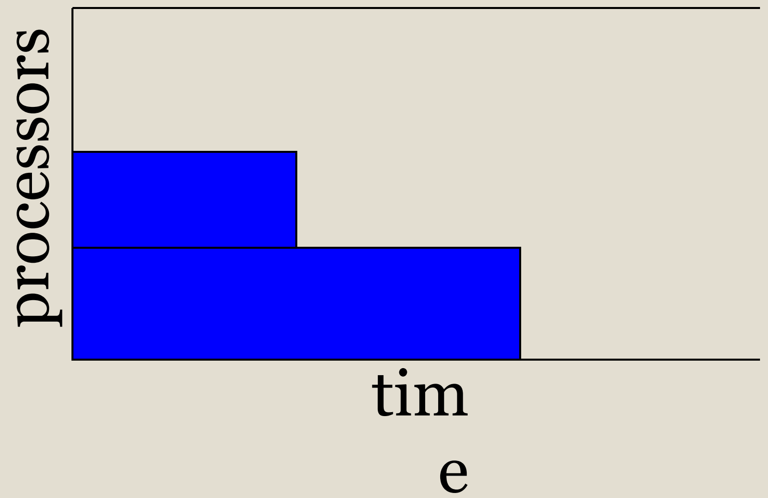
EASY Operation

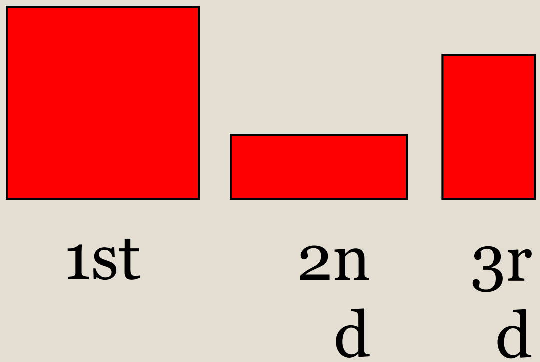
1. Schedule jobs on available processors in FCFS order
2. Make reservation for first job that cannot run
3. Schedule additional jobs provided they do not conflict with this reservation



EASY Operation

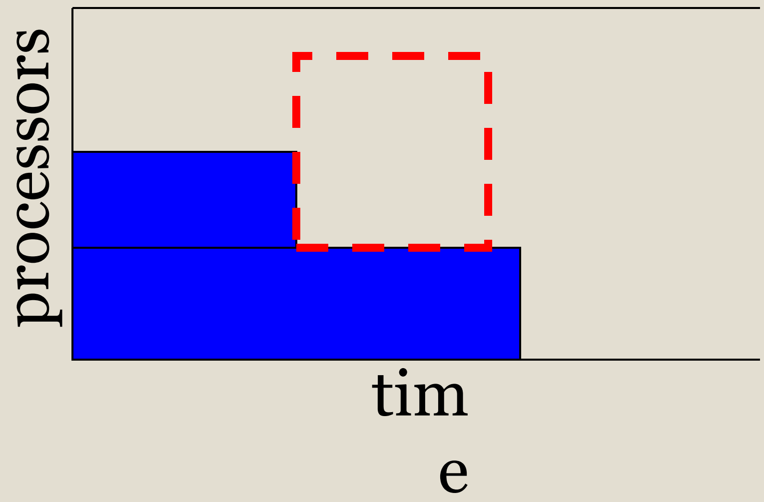
1. Schedule jobs on available processors in FCFS order
2. Make reservation for first job that cannot run
3. Schedule additional jobs provided they do not conflict with this reservation

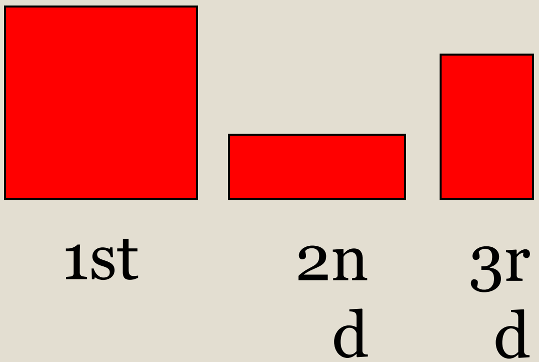




EASY Operation

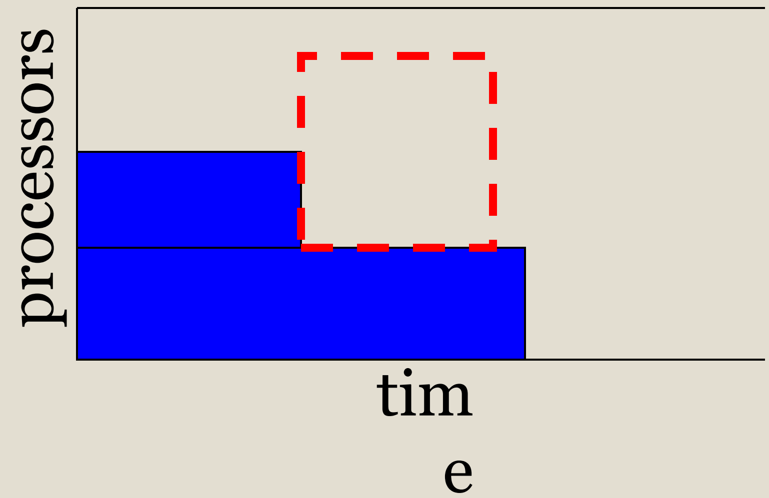
1. Schedule jobs on available processors in FCFS order
2. Make reservation for first job that cannot run
3. Schedule additional jobs provided they do not conflict with this reservation

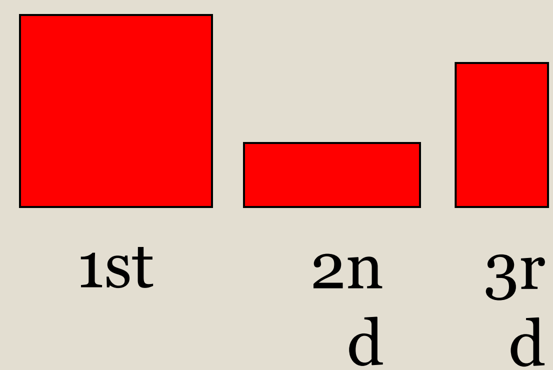




EASY Operation

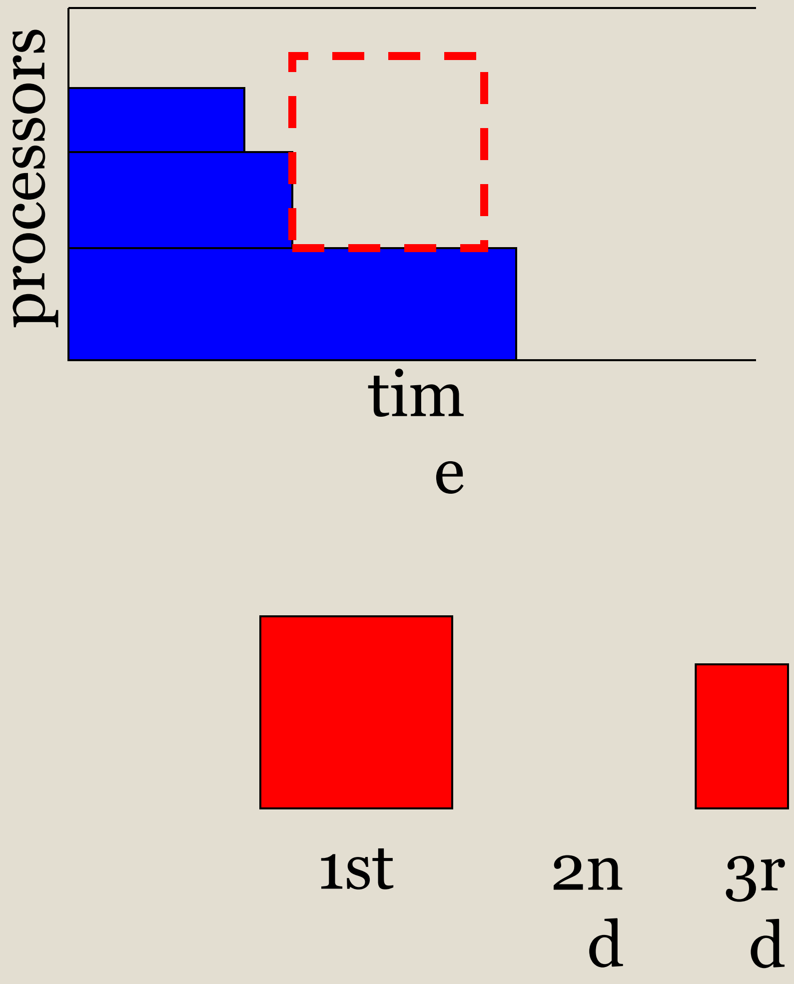
1. Schedule jobs on available processors in FCFS order
2. Make reservation for first job that cannot run
3. Schedule additional jobs provided they do not conflict with this reservation





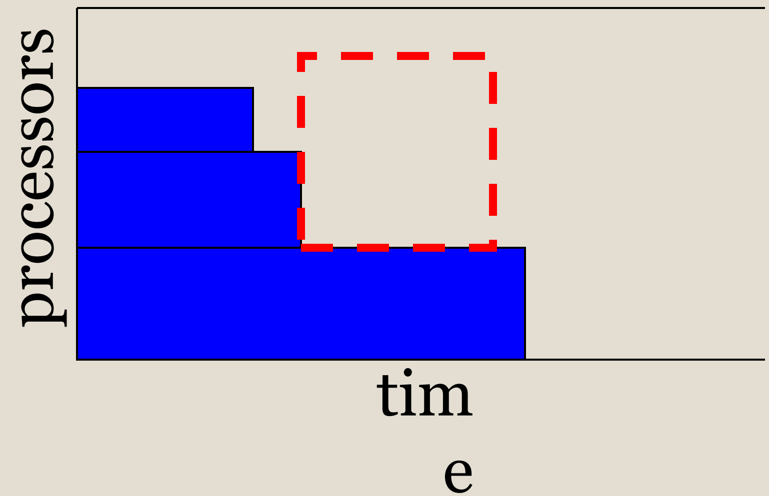
EASY Operation

1. Schedule jobs on available processors in FCFS order
2. Make reservation for first job that cannot run
3. Schedule additional jobs provided they do not conflict with this reservation

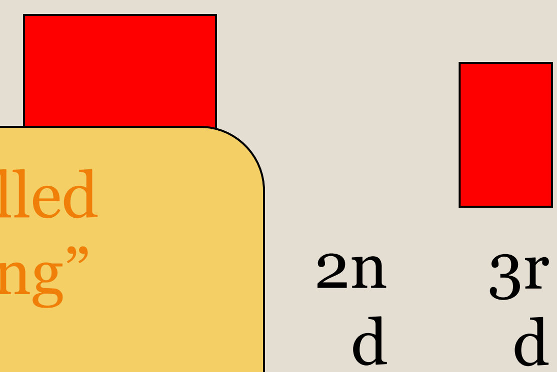


EASY Operation

1. Schedule jobs on available processors in FCFS order
2. Make reservation for first job that cannot run
3. Schedule additional jobs provided they do not conflict with this reservation

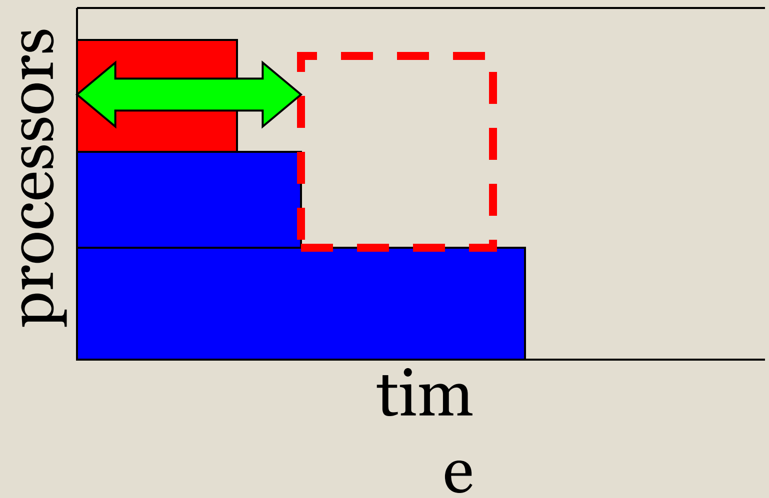


This is called "backfilling"



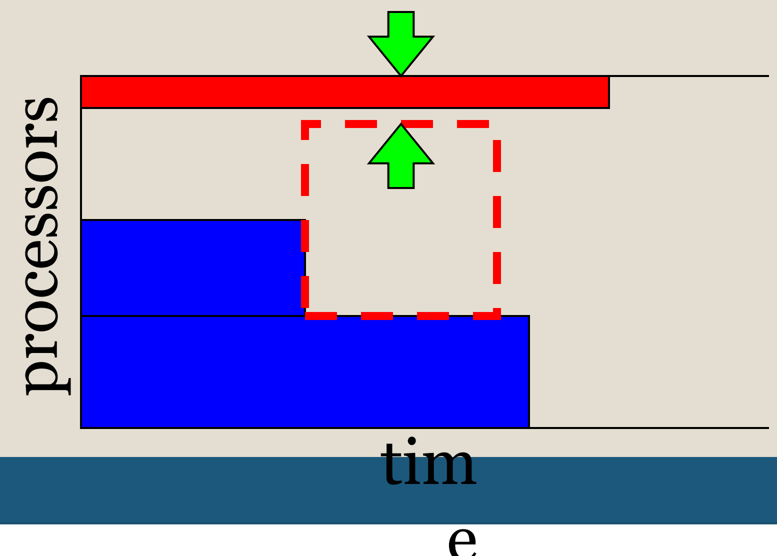
Backfilling Conditions

1. Backfill job will terminate before reservation time



OR

2. Backfill job uses only "extra" processors



Runtime Estimates

- When users submit jobs, they provide
  1. The number of processors to use
  2. An estimate of the job runtime

- Estimates are used to predict when processors will become free for reservation

- Also used to verify that backfill job will terminate before reservation

- If it does not, it will be killed

Questions?