Sistemi Operativi

Corso di Laurea in Informatica

a.a. 2019-2020

Gabriele Tolomei

Dipartimento di Informatica

Sapienza Università di Roma

tolomei@di.uniroma1.it

SAPIENZA UNIVERSITÀ DI ROMA

Recap from Last Lecture

- Process is the **unit of execution** (running on a single CPU)
- OS keeps track of process-related information using an ad hoc data structure called **Process Control Block** (**PCB**)
- Process can be in one of **5 possible states**: **new**, **ready**, **waiting**, **running**, or **terminated**
- **Context switch** to intertwine the execution of multiple processes
- Process communication either via **message passing** or **shared memory**

Today: CPU Scheduling

Policy to establish which process to execute on the CPU

- Basic scheduling concepts
- Scheduling **criteria/metrics**
- Scheduling **algorithms**
- Advanced scheduling concepts

Basic Concepts

- Almost every program has some alternating cycles of CPU computations and I/O waiting
- Even a simple fetch from main memory takes a long time relatively to CPU speed
- **Our assumptions:** Multi-programmed, uni-processor system

Basic Concepts

- In a system running a single process, the time spent waiting for I/O is wasted, and those CPU cycles are lost forever

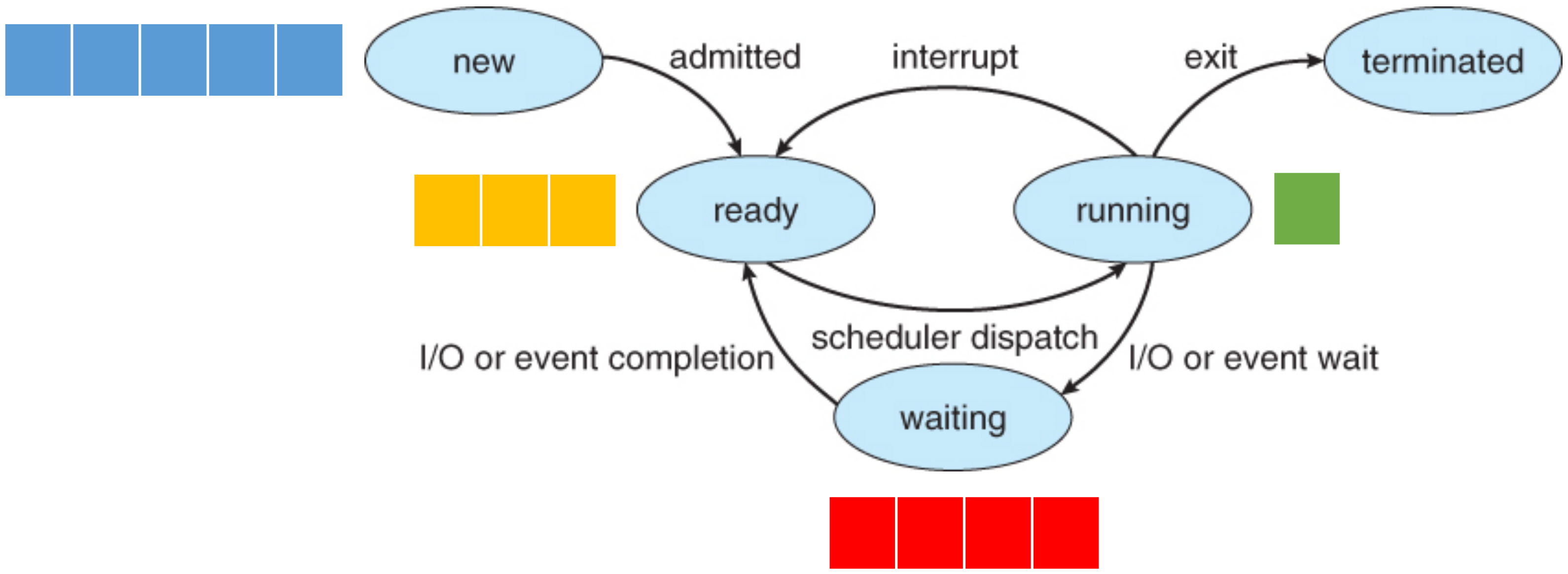
Basic Concepts

- In a system running a single process, the time spent waiting for I/O is wasted, and those CPU cycles are lost forever
- A scheduling system allows one process to use the CPU while another is waiting for I/O, thereby maximizing system utilization

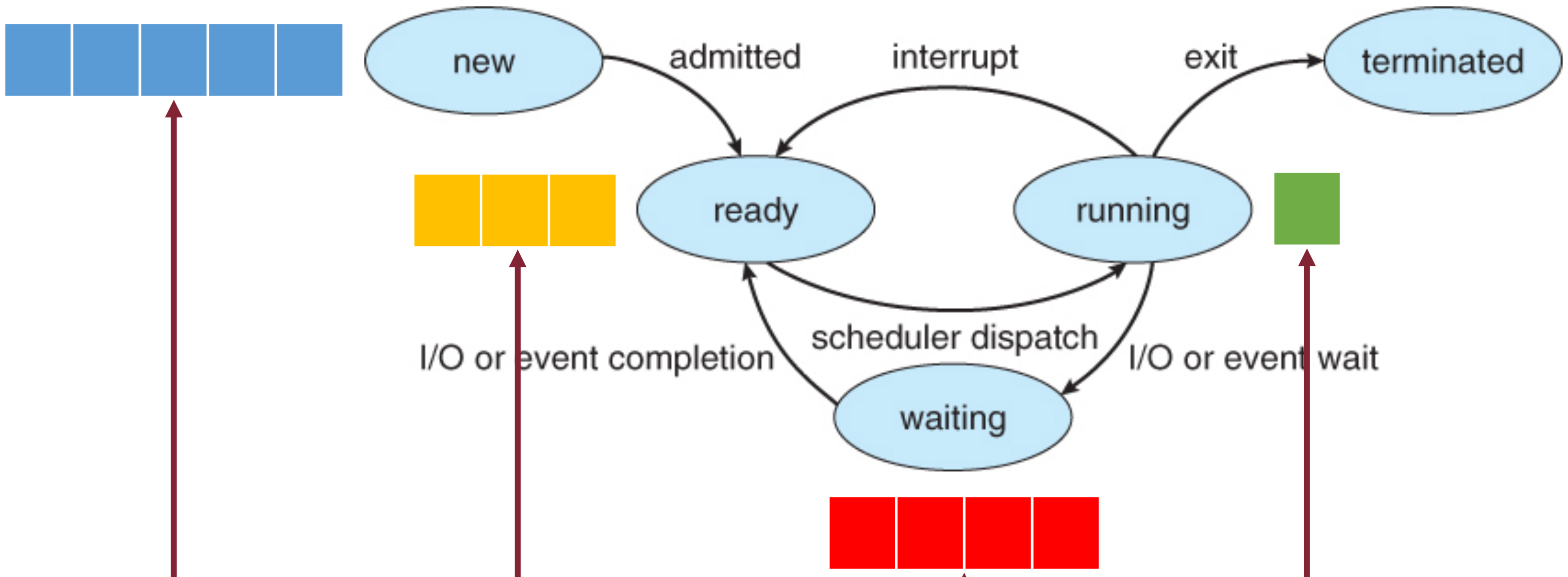
Basic Concepts

- In a system running a single process, the time spent waiting for I/O is wasted, and those CPU cycles are lost forever
- A scheduling system allows one process to use the CPU while another is waiting for I/O, thereby maximizing system utilization
- **Challenge:** Make the system as "efficient" and "fair" as possible, subject to varying and often dynamic conditions

Process Execution State Diagram

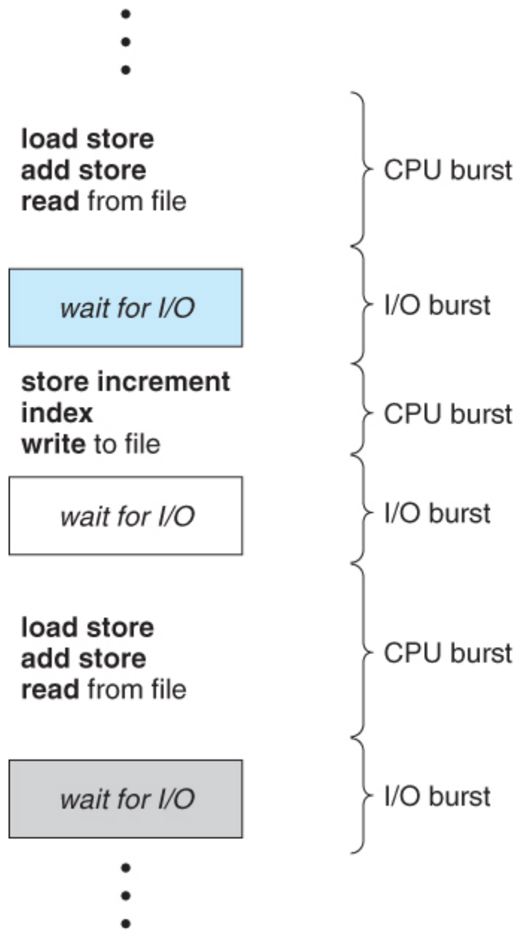


Process Execution State Diagram



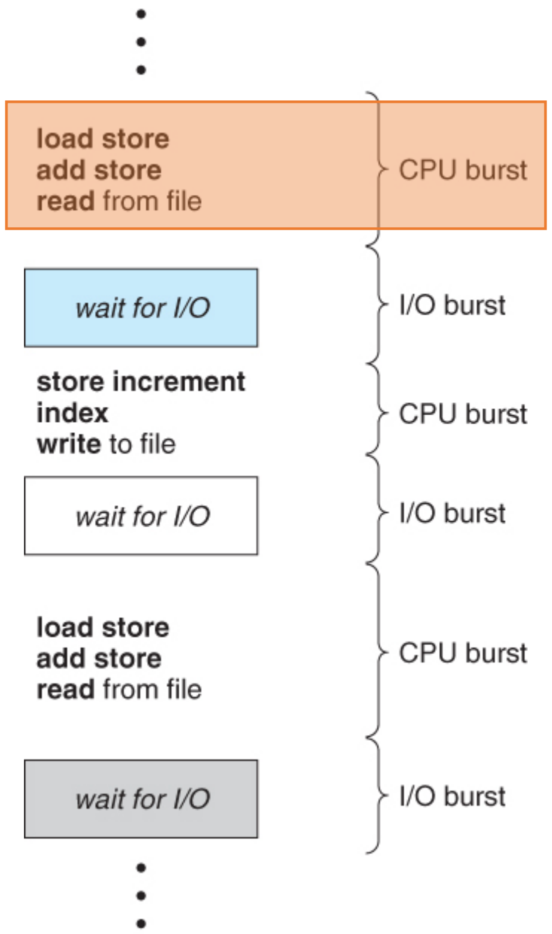
Processes managed by the OS reside in exactly one of the state queues

CPU vs. I/O Burst Cycle



All processes alternate between two states in a continuing **cycle**: **CPU burst** and **I/O burst**

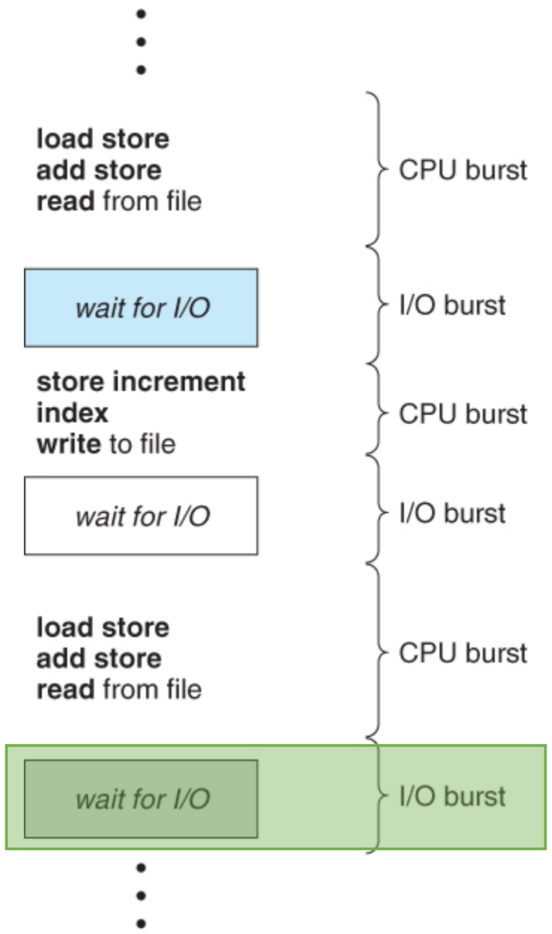
CPU vs. I/O Burst Cycle



All processes alternate between two states in a continuing **cycle**: **CPU burst** and **I/O burst**

CPU burst → performing calculations

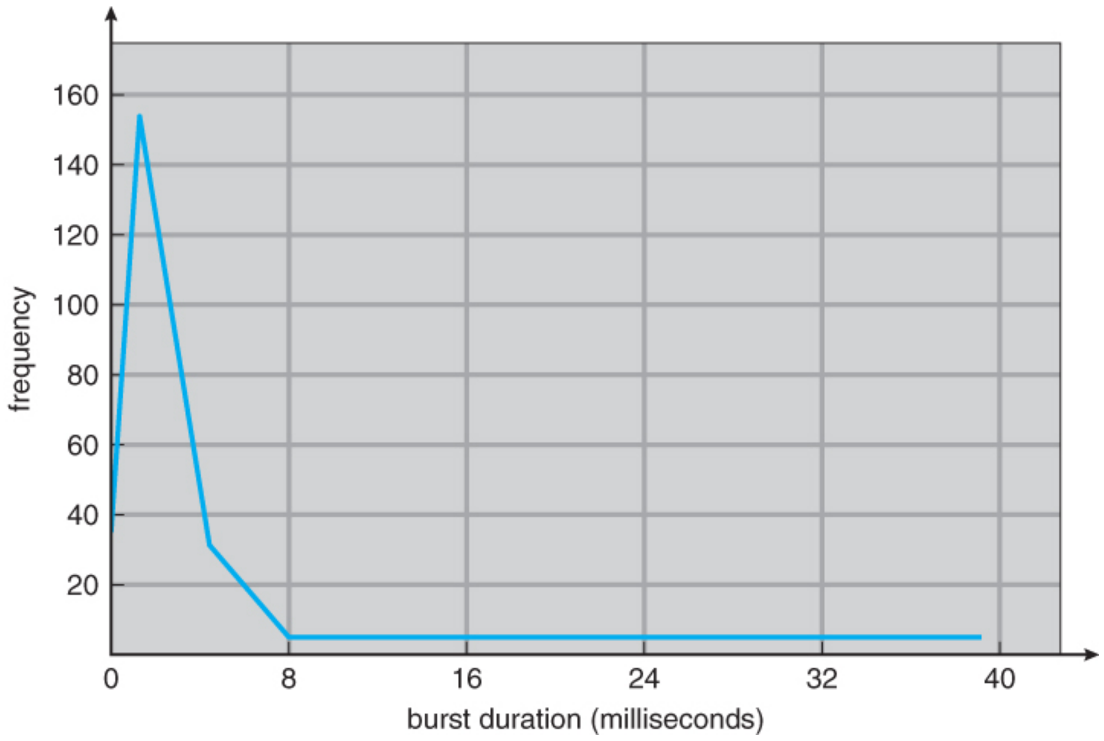
CPU vs. I/O Burst Cycle



All processes alternate between two states in a continuing **cycle**: **CPU burst** and **I/O burst**

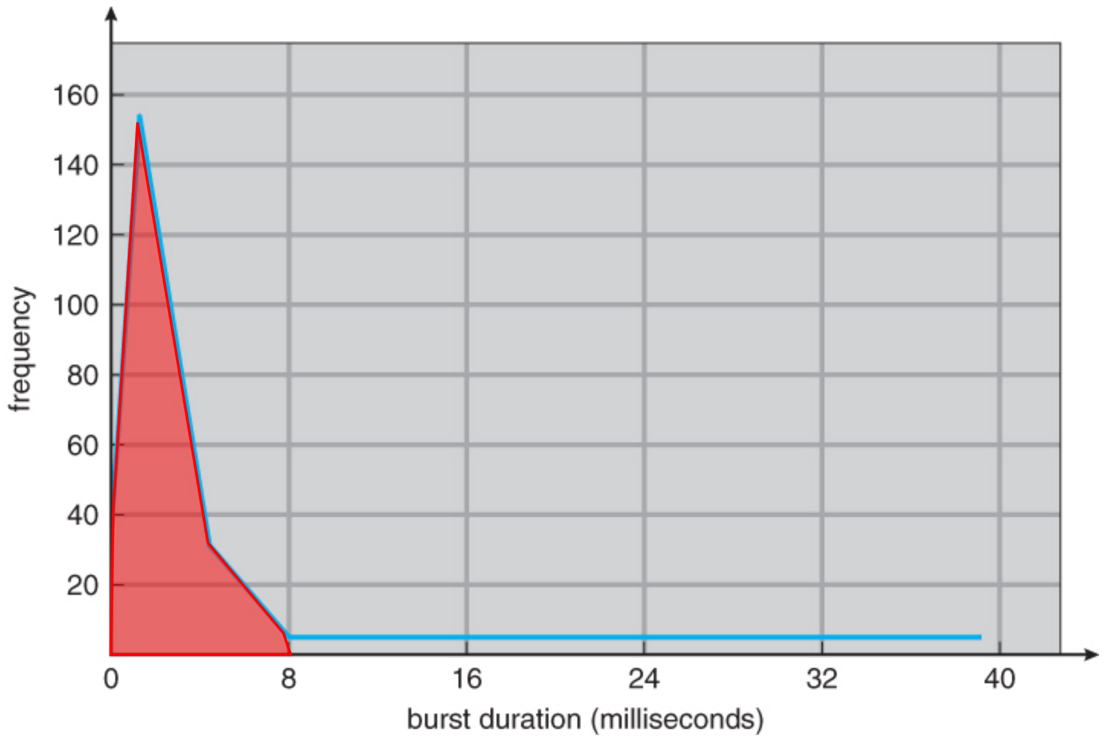
I/O burst → waiting for data transfer in or out of the system

CPU Burst Cycle: Frequency Pattern



Highly skewed distribution

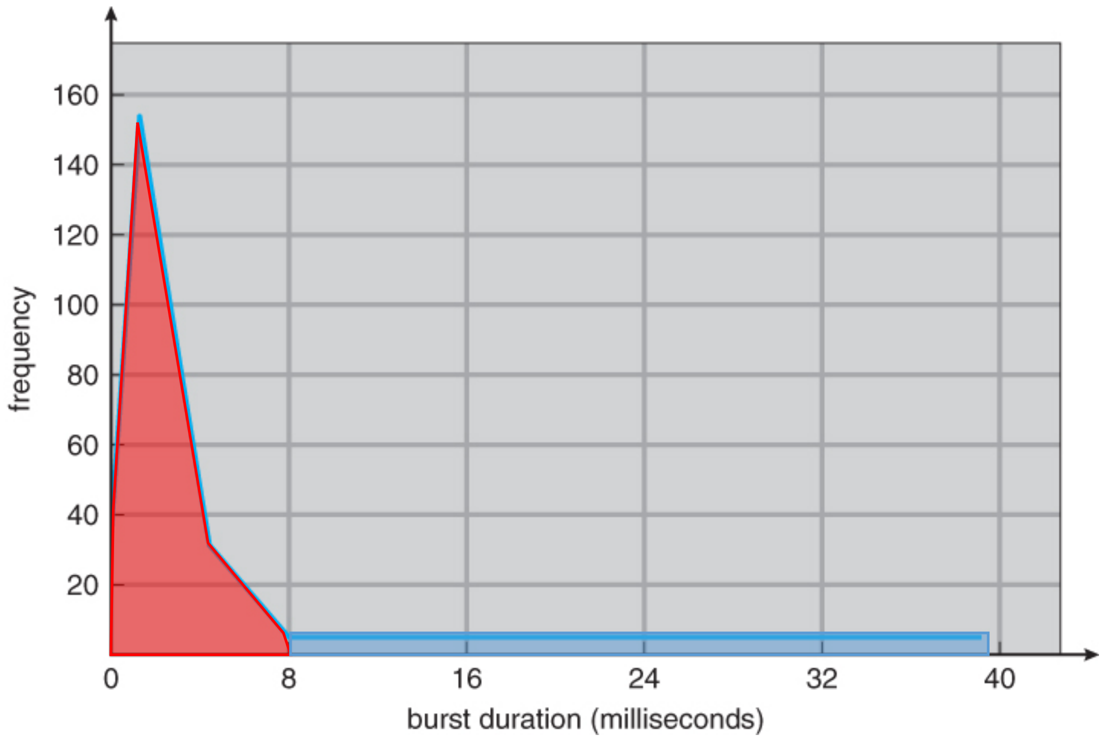
CPU Burst Cycle: Frequency Pattern



Highly skewed distribution

The vast majority of processes have **short** CPU bursts

CPU Burst Cycle: Frequency Pattern



Highly skewed distribution

The vast majority of processes have **short** CPU bursts

Few processes exhibit **very long** CPU bursts

Long- vs. Short-term Scheduling

Long-term scheduling

How does the OS determine the level of multiprogramming (i.e., the number of processes to be loaded in main memory)

Long- vs. Short-term Scheduling

Long-term scheduling

How does the OS determine the level of multiprogramming (i.e., the number of processes to be loaded in main memory)

Short-term scheduling

How does the OS select a process to execute from the ready queue

Long- vs. Short-term Scheduling

Long-term scheduling

How does the OS determine the level of multiprogramming (i.e., the number of processes to be loaded in main memory)

Short-term scheduling

How does the OS select a process to execute from the ready queue

CPU Scheduler

CPU Scheduler

- Whenever the CPU becomes idle, it picks another process from the **ready** queue to execute next
- The data structure for the ready queue and the algorithm used to select the next process are not necessarily based on a FIFO queue

CPU Scheduler

- Whenever the CPU becomes idle, it picks another process from the **ready** queue to execute next
- The data structure for the ready queue and the algorithm used to select the next process are not necessarily based on a FIFO queue

Policy goals vs. Mechanism implementations

CPU Scheduling: When?

CPU scheduling decisions take place under one of **4 conditions**:

CPU Scheduling: When?

CPU scheduling decisions take place under one of **4 conditions**:

1. When a process switches from the running state
   to the waiting state

CPU Scheduling: When?

CPU scheduling decisions take place under one of **4 conditions**:

1. When a process switches from the running state to the waiting state

for an I/O request or invocation of the **`wait`** system call

CPU Scheduling: When?

CPU scheduling decisions take place under one of **4 conditions**:

1. When a process switches from the running state to the waiting state
2. When a process switches from the running state to the ready state

CPU Scheduling: When?

CPU scheduling decisions take place under one of **4 conditions**:

1. When a process switches from the running state to the waiting state
2. When a process switches from the running state to the ready state

in response to an interrupt

CPU Scheduling: When?

CPU scheduling decisions take place under one of **4 conditions**:

1. When a process switches from the running state to the waiting state
2. When a process switches from the running state to the ready state
3. When a process switches from the waiting state to the ready state

CPU Scheduling: When?

CPU scheduling decisions take place under one of **4 conditions**:

1. When a process switches from the running state to the waiting state
2. When a process switches from the running state to the ready state
3. When a process switches from the waiting state to the ready state

at completion of I/O or a return from **`wait`**

CPU Scheduling: When?

CPU scheduling decisions take place under one of **4 conditions**:

1. When a process switches from the running state to the waiting state
2. When a process switches from the running state to the ready state
3. When a process switches from the waiting state to the ready state
4. When a process is created or terminates

CPU Scheduling: When?

CPU scheduling decisions take place under one of **4 conditions**:

1. When a process switches from the running state to the waiting state
2. When a process switches from the running state to the ready state
3. When a process switches from the waiting state to the ready state
4. When a process is created or terminates

No choice!
A new process must be selected

CPU Scheduling: When?

CPU scheduling decisions take place under one of **4 conditions**:

1. When a process switches from the running state to the waiting state
2. When a process switches from the running state to the ready state
3. When a process switches from the waiting state to the ready state
4. When a process is created or terminates

Either continue with the current process or select a new one

Non-preemptive vs. Preemptive

Non-preemptive scheduling

If it takes place only when there is no choice (i.e., conditions 1 and 4)

Once a process starts it keeps running until it either voluntarily blocks or it finishes

Non-preemptive vs. Preemptive

Non-preemptive scheduling

If it takes place only when there is no choice (i.e., conditions 1 and 4)

Once a process starts it keeps running until it either voluntarily blocks or it finishes

Preemptive scheduling

Whenever scheduling takes *also* place under conditions 2 and 3

Non-preemptive vs. Preemptive: Examples

	Windows	Mac	UNIX-like
Non-preemptive	up to Win 3.x	up to Mac OS 9.x	-
Preemptive	since Win 95	since Mac OS X	since forever

Preemption: Issues

- Preemption might cause troubles if it occurs while:
  - two processes share data, one may get interrupted in the middle of updating shared data structures
  - the kernel is busy implementing a system call (e.g., updating critical kernel data structures)

Preemption: Issues

- Possible countermeasures:
  - Make the process wait until the system call has either completed or blocked before allowing the preemption

Preemption: Issues

- Possible countermeasures:
  - Make the process wait until the system call has either completed or blocked before allowing the preemption

problematic for real-time systems, as real-time response can no longer be guaranteed

Preemption: Issues

- Possible countermeasures:
  - Make the process wait until the system call has either completed or blocked before allowing the preemption

problematic for real-time systems, as real-time response can no longer be guaranteed

  - Disable interrupts before entering critical code section and re-enabling immediately afterwards

Preemption: Issues

- Possible countermeasures:
  - Make the process wait until the system call has either completed or blocked before allowing the preemption

problematic for real-time systems, as real-time response can no longer be guaranteed

  - Disable interrupts before entering critical code section and re-enabling immediately afterwards

should only be done in rare situations, and only on very short pieces of code that will finish quickly

The Dispatcher

- The module that gives control of the CPU to the process selected by the scheduler

The Dispatcher

- The module that gives control of the CPU to the process selected by the scheduler
- Its functions include:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the newly loaded program

The Dispatcher

- The module that gives control of the CPU to the process selected by the scheduler
- Its functions include:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the newly loaded program
- The dispatcher is run on every context switch therefore the time it consumes (**dispatch latency**) must be as shortest as possible

Useful Definitions

- **Arrival Time:** time at which the process arrives in the ready queue

Useful Definitions

- **Arrival Time:** time at which the process arrives in the ready queue
- **Completion Time:** time at which the process completes its execution

Useful Definitions

- **Arrival Time:** time at which the process arrives in the ready queue
- **Completion Time:** time at which the process completes its execution
- **Burst Time:** time required by a process for CPU execution

Useful Definitions

- **Arrival Time:** time at which the process arrives in the ready queue
- **Completion Time:** time at which the process completes its execution
- **Burst Time:** time required by a process for CPU execution
- **Turnaround Time:** time difference between completion and arrival time

Useful Definitions

- **Arrival Time:** time at which the process arrives in the ready queue
- **Completion Time:** time at which the process completes its execution
- **Burst Time:** time required by a process for CPU execution
- **Turnaround Time:** time difference between completion and arrival time
- **Waiting Time:** time difference between turnaround time and burst time

Useful Definitions

$$T^{arrival} = \text{arrival time}$$

$$T^{completion} = \text{completion time}$$

$$T^{burst} = \text{burst time}$$

$$T^{turnaround} = \text{tournaround time} = T^{completion} - T^{arrival}$$

$$T^{waiting} = \text{waiting time} = T^{turnaround} - T^{burst}$$

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - Throughput
  - Tournaround time
  - Waiting time
  - Response time

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - **CPU utilization**
  - Throughput
  - Tournaround time
  - Waiting time
  - Response time

Intuitively, the percentage of time the CPU is busy

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - **CPU utilization**
  - Throughput
  - Tournaround time
  - Waiting time
  - Response time

Intuitively, the percentage of time the CPU is busy

Ideally the CPU would be busy 100% of the time, so as to waste 0 CPU cycles

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - **CPU utilization**
  - Throughput
  - Tournaround time
  - Waiting time
  - Response time

Intuitively, the percentage of time the CPU is busy

Ideally the CPU would be busy 100% of the time, so as to waste 0 CPU cycles

On a real system CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded)

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - **CPU utilization**
  - Throughput
  - Tournaround time
  - Waiting time
  - Response time

Intuitively, the percentage of time the CPU is busy

Ideally the CPU would be busy 100% of the time, so as to waste 0 CPU cycles

On a real system CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded)

maximize

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - **Throughput**
  - Tournaround time
  - Waiting time
  - Response time

The number of processes completed in a unit of time

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - **Throughput**
  - Tournaround time
  - Waiting time
  - Response time

The number of processes completed in a unit of time

May range from 10 per second to 1 per hour

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - **Throughput**
  - Tournaround time
  - Waiting time
  - Response time

The number of processes completed in a unit of time

May range from 10 per second to 1 per hour

maximize

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - Throughput
  - **Tournaround time**
  - Waiting time
  - Response time

Time required for a particular process to complete, from submission time to completion

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - Throughput
  - **Tournaround time**
  - Waiting time
  - Response time

Time required for a particular process to complete, from submission time to completion

Includes all the waiting time

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - Throughput
  - **Tournaround time**
  - Waiting time
  - Response time

Time required for a particular process to complete, from submission time to completion

Includes all the waiting time

minimize

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - Throughput
  - Tournaround time
  - **Waiting time**
  - Response time

How much time processes spend in the ready queue waiting their turn to get on the CPU

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - Throughput
  - Tournaround time
  - **Waiting time**
  - Response time

How much time processes spend in the ready queue waiting their turn to get on the CPU

Not to be confused with the waiting state!

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - Throughput
  - Tournaround time
  - **Waiting time**
  - Response time

How much time processes spend in the ready queue waiting their turn to get on the CPU

Not to be confused with the waiting state!

Processes in the waiting state are not under control of the CPU scheduler (they are just not ready to run!)

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - Throughput
  - Tournaround time
  - **Waiting time**
  - Response time

How much time processes spend in the ready queue waiting their turn to get on the CPU

Not to be confused with the waiting state!

Processes in the waiting state are not under control of the CPU scheduler (they are just not ready to run!)

minimize

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - Throughput
  - Tournaround time
  - Waiting time
  - **Response time**

The time taken from the issuance of a command to the commence of a response to that command

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - Throughput
  - Tournaround time
  - Waiting time
  - **Response time**

The time taken from the issuance of a command to the commence of a response to that command

Mostly, for interactive processes

Scheduling Criteria/Metrics

- There are several different criteria to consider when trying to select the "best" scheduling algorithm, including:
  - CPU utilization
  - Throughput
  - Tournaround time
  - Waiting time
  - **Response time**

The time taken from the issuance of a command to the commence of a response to that command

Mostly, for interactive processes

minimize

Scheduling: Trade-off

- Ideally, choose a CPU scheduler that optimizes all metrics simultaneously
- Generally, the above is impossible and a trade-off is needed!
- **Idea:** Choose a scheduling algorithm based on its ability to satisfy a given policy

Scheduling Policies

- Minimize **average** response time
  - Provide output to the user as quickly as possible

Scheduling Policies

- Minimize **average** response time
  - Provide output to the user as quickly as possible
- Minimize the **maximum** response time
  - Worst-case bound

Scheduling Policies

- Minimize **average** response time
  - Provide output to the user as quickly as possible
- Minimize the **maximum** response time
  - Worst-case bound
- Minimize **variance** of response time
  - Users are more accepting of a consistent predictable system than an inconsistent one

Scheduling Policies

- Minimize **average** response time
  - Provide output to the user as quickly as possible
- Minimize the **maximum** response time
  - Worst-case bound
- Minimize **variance** of response time
  - Users are more accepting of a consistent predictable system than an inconsistent one

Typical of interactive systems

Scheduling Policies

- Maximize throughput translates into:
  - Minimizing overhead (OS context switching)
  - Efficient usage of system resources (CPU and I/O devices)

Scheduling Policies

- Maximize throughput translates into:
  - Minimizing overhead (OS context switching)
  - Efficient usage of system resources (CPU and I/O devices)
- Minimize waiting time
  - Give every process the same amount of time on the CPU
  - Might increase the average response time

Scheduling Policies

- Maximize throughput translates into:
  - Minimizing overhead (OS context switching)
  - Efficient usage of system resources (CPU and I/O devices)
- Minimize waiting time
  - Give every process the same amount of time on the CPU
  - Might increase the average response time

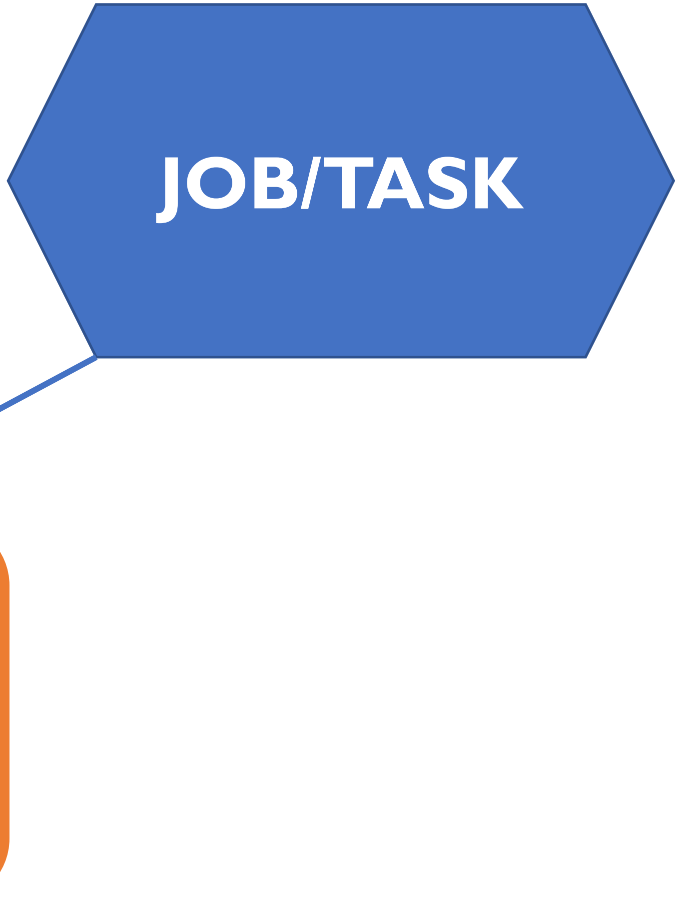
Typical of batch systems

A Quick Note on Terminology

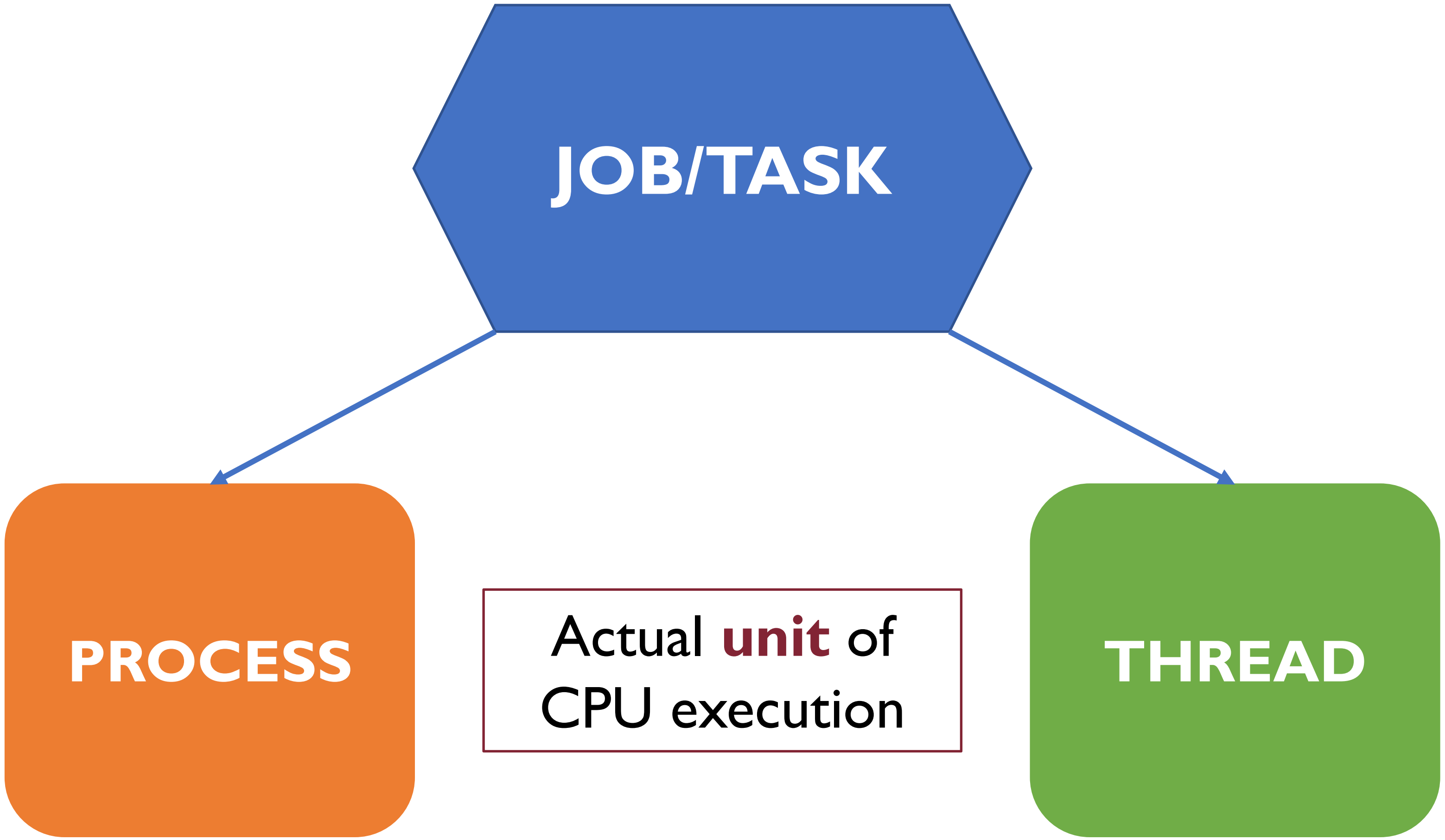
JOB/TASK

General **unit** of CPU execution

A Quick Note on Terminology



A Quick Note on Terminology



Scheduling Policies: Our Assumptions

- One process per user

Scheduling Policies: Our Assumptions

- One process per user
- Processes are independent from each other (i.e., no communication)

Scheduling Policies: Our Assumptions

- One process per user
- Processes are independent from each other (i.e., no communication)
- One single thread (of execution) per process

Scheduling Policies: Our Assumptions

- One process per user
- Processes are independent from each other (i.e., no communication)
- One single thread (of execution) per process

We will talk about threads very soon but for now most of the things we will be discussing remain valid even on a multi-threaded system

Scheduling Algorithms: An Overview

- First-Come-First-Serve (FCFS)
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Priority Scheduling
- Multilevel Queue (MQ)
- Multilevel Feedback-Queue (MFQ)

Scheduling Algorithms: An Overview

- **First-Come-First-Serve (FCFS)**
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Priority Scheduling
- Multilevel Queue (MQ)
- Multilevel Feedback-Queue (MFQ)

First-Come-First-Serve (FCFS)

- Very simple! Just a FIFO queue, like customers waiting in line at the post office
- The scheduler executes jobs to completion in arrival order
- The scheduler takes over only when the currently running job asks for an I/O operation (or finishes its execution)
- A job may keep using the CPU indefinitely (i.e., until it blocks)

First-Come-First-Serve (FCFS)

- Very simple! Just a FIFO queue, like customers waiting in line at the post office
- The scheduler executes jobs to completion in arrival order
- The scheduler takes over only when the currently running job asks for an I/O operation (or finishes its execution)
- A job may keep using the CPU indefinitely (i.e., until it blocks)

Non-preemptive

First-Come-First-Serve (FCFS): Scenario 1

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

First-Come-First-Serve (FCFS): Scenario 1

New A B C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

First-Come-First-Serve (FCFS): Scenario 1

New | A | B | C |

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

Arrival time = 0 for all

No I/O burst

First-Come-First-Serve (FCFS): Scenario I

New A B C

Ready A B C

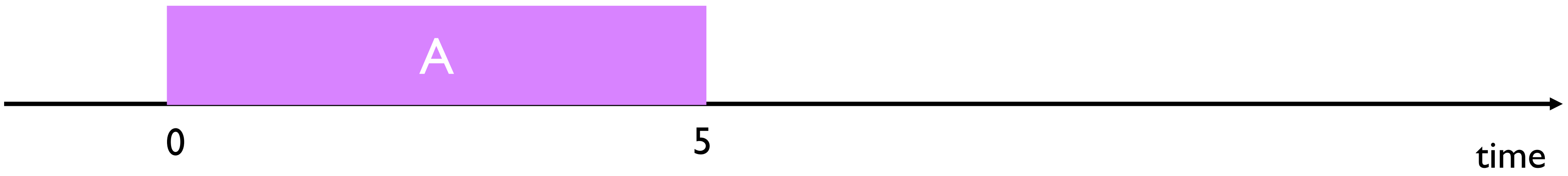
Waiting

Running

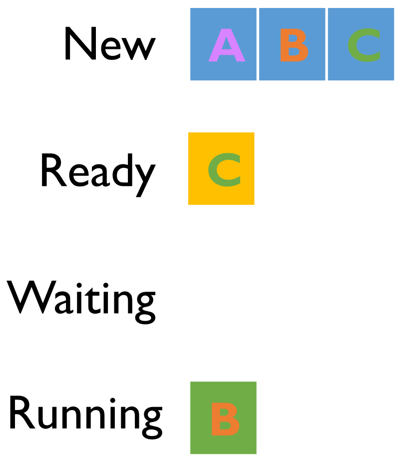
Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

First-Come-First-Serve (FCFS): Scenario 1

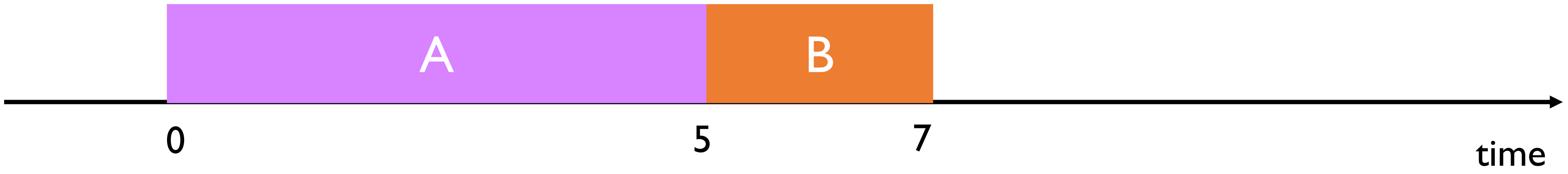
Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



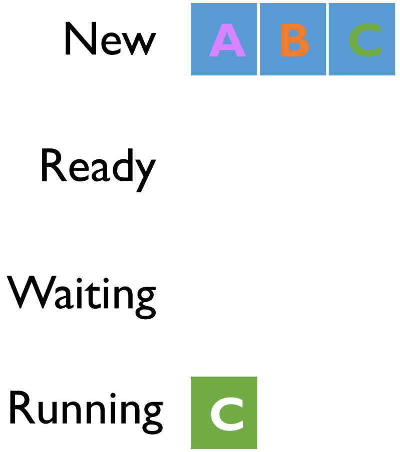
First-Come-First-Serve (FCFS): Scenario 1



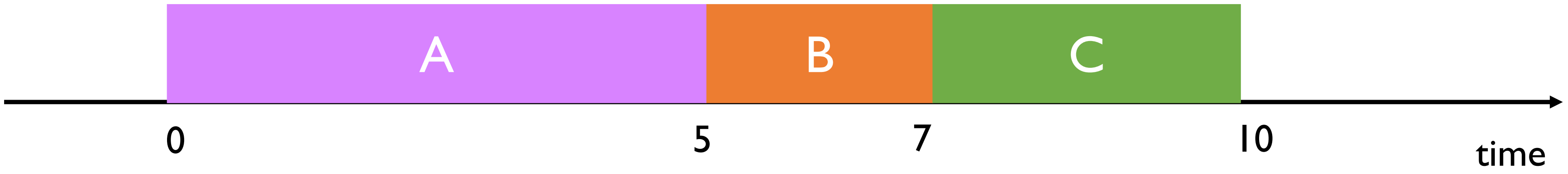
Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



First-Come-First-Serve (FCFS): Scenario 1



Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



Average Waiting Time

$$N = \text{number of jobs}$$

$$T_i^{arrival} = \text{arrival time of job } i$$
$$T_i^{completion} = \text{completion time of job } i$$
$$T_i^{burst} = \text{burst time of job } i$$
$$T_i^{turnaround} = \text{tournaround time of job } i = T_i^{completion} - T_i^{arrival}$$

$$\overline{T}_i^{waiting} = \text{avg. waiting time of job } i = \frac{1}{N}\sum_{i=1}^{N}(T_i^{turnaround} - T_i^{burst})$$

Unless otherwise specified, we will assume all jobs arrive at the same time, i.e.,

$$T_i^{arrival} = 0 \ \forall i \in \{1, \ldots, N\}$$

First-Come-First-Serve (FCFS): Scenario 1

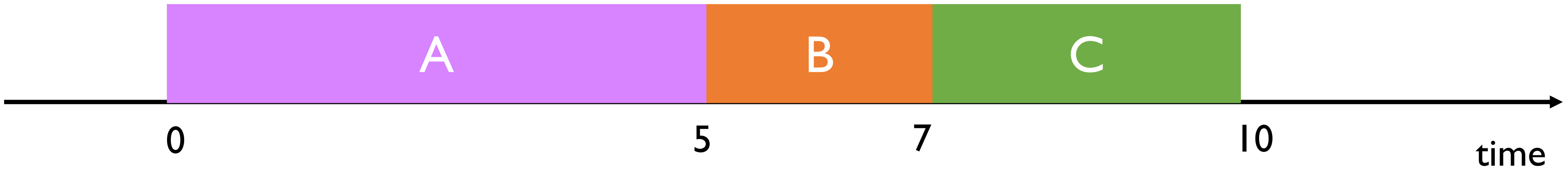
New A B C

Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



avg. waiting time =

First-Come-First-Serve (FCFS): Scenario 1

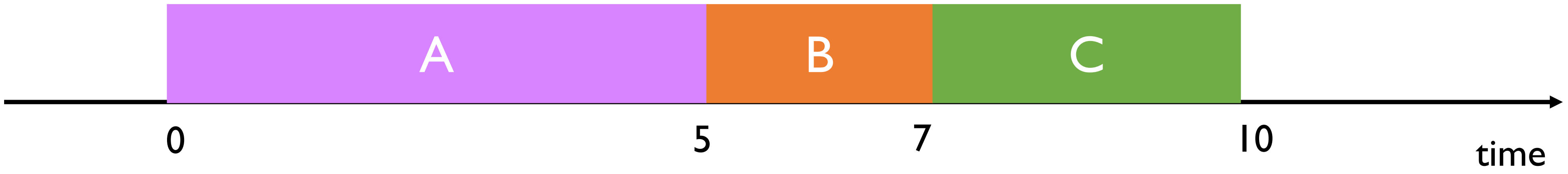
New A B C

Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



avg. waiting time = (0 + 5 + 7)/3 = **4**

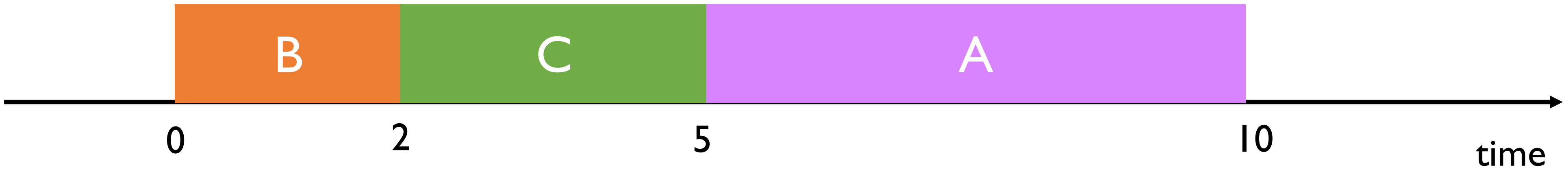
First-Come-First-Serve (FCFS): Scenario II

New B C A

Order	Job	CPU burst (time units)
1	B	2
2	C	3
3	A	5

Arrival time = 0 for all

No I/O burst

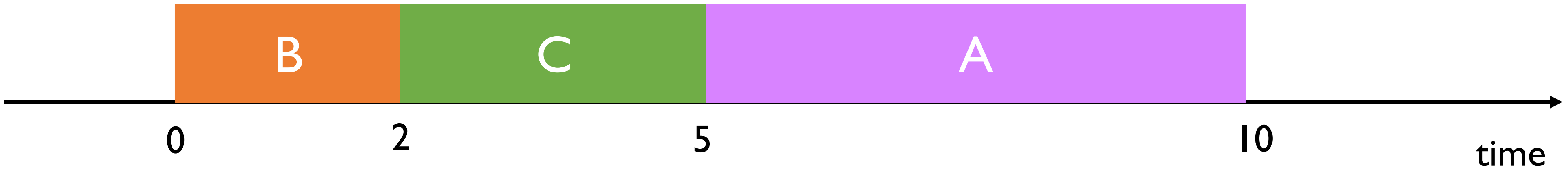


avg. waiting time =

First-Come-First-Serve (FCFS): Scenario II

New 

Order	Job	CPU burst (time units)
1	B	2
2	C	3
3	A	5



avg. waiting time = (5 + 0 + 2)/3 ~ **2.3**

First-Come-First-Serve (FCFS): Scenario III

New A B C

Ready A B C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

First-Come-First-Serve (FCFS): Scenario III

New A B C

Ready A B C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O

First-Come-First-Serve (FCFS): Scenario III

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O

First-Come-First-Serve (FCFS): Scenario III

New: A B C

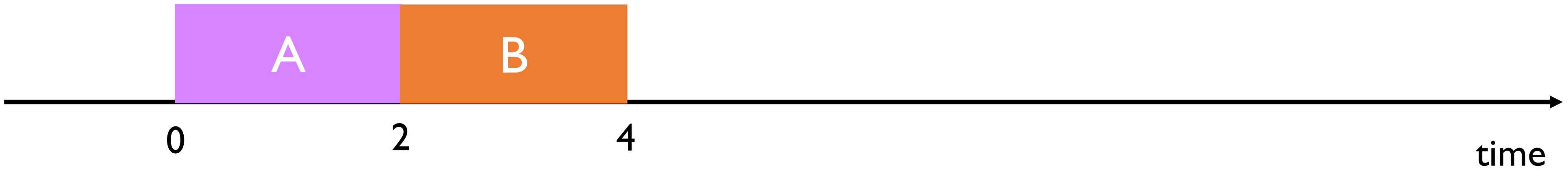
Ready: C

Waiting: A

Running: B

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O



First-Come-First-Serve (FCFS): Scenario III

New A B C

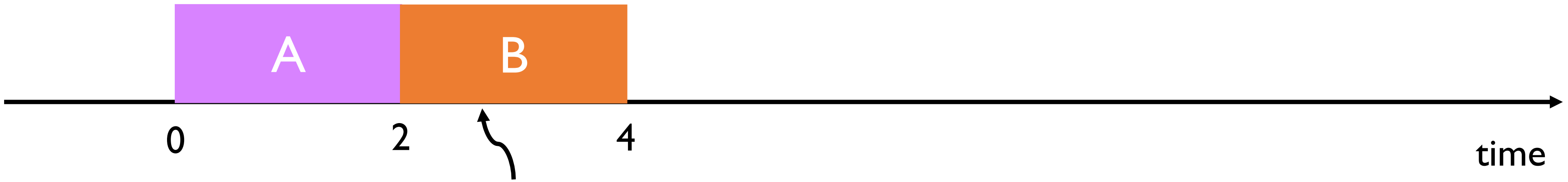
Ready C

Waiting A

Running B

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O



I/O task requested by A might have finished before B finished

Non-preemptive scheduler!

First-Come-First-Serve (FCFS): Scenario III

New: A B C

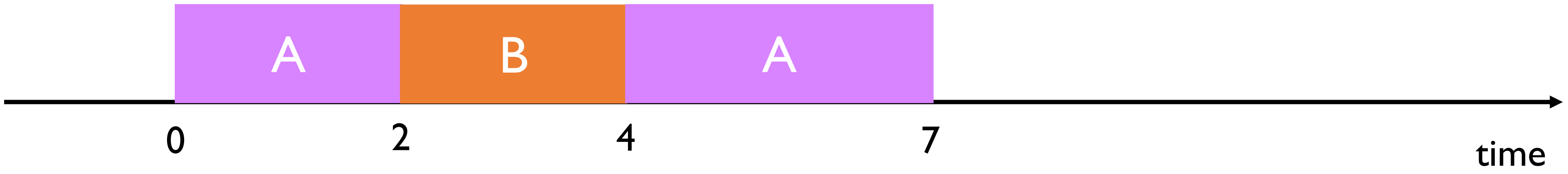
Ready: C

Waiting

Running: A

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O



First-Come-First-Serve (FCFS): Scenario III

New A B C

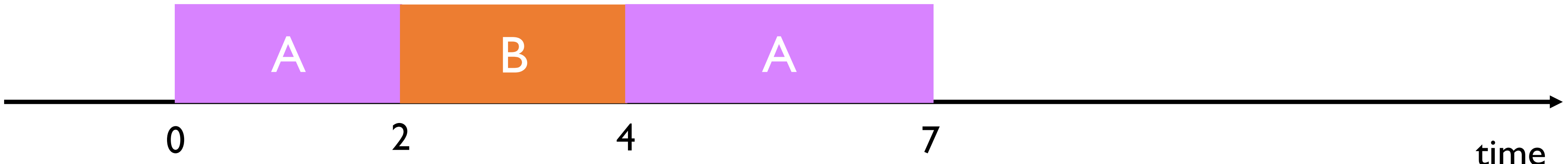
Ready C

Waiting

Running A

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O



Why A and not C?

First-Come-First-Serve (FCFS): Scenario III

New A B C

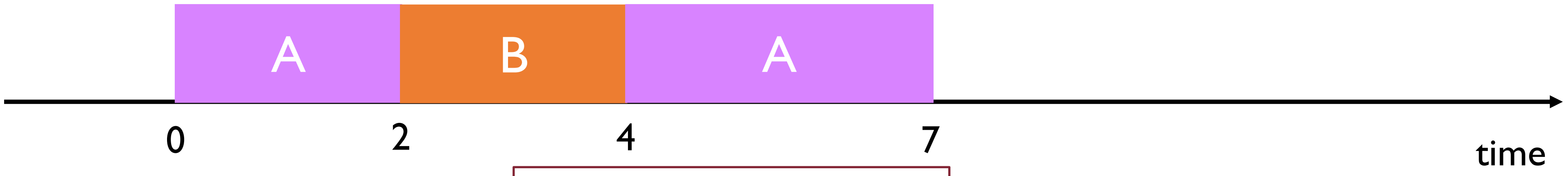
Ready C

Waiting

Running A

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

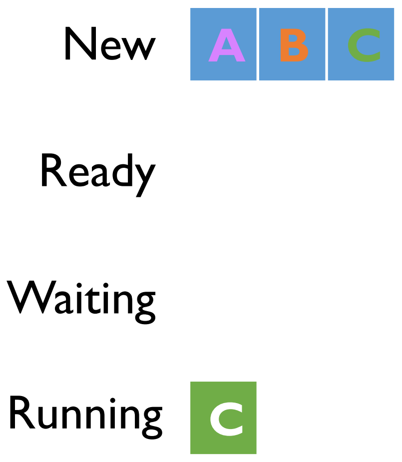
A does also I/O



Why A and not C?

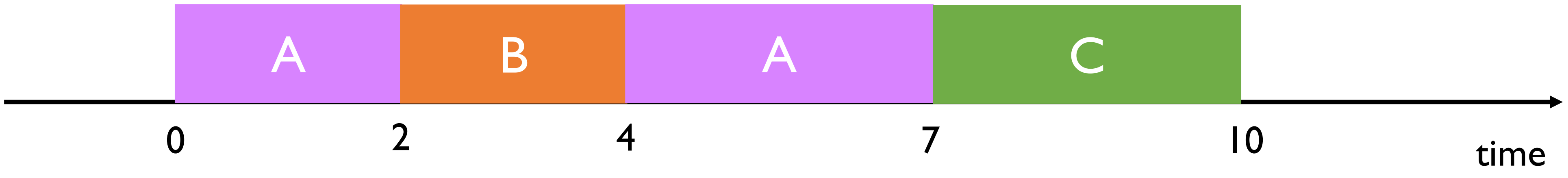
Because the FCFS scheduler cares only about the arrival time on the **new** queue

First-Come-First-Serve (FCFS): Scenario III



Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O



First-Come-First-Serve (FCFS): Scenario III

New A B C

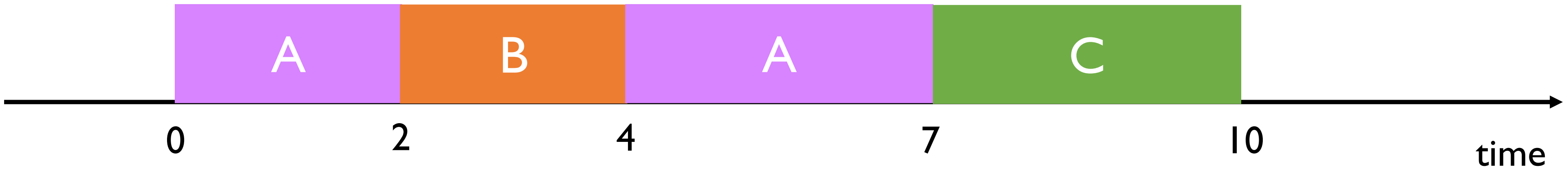
Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O



avg. waiting time =

First-Come-First-Serve (FCFS): Scenario III

New | A | B | C

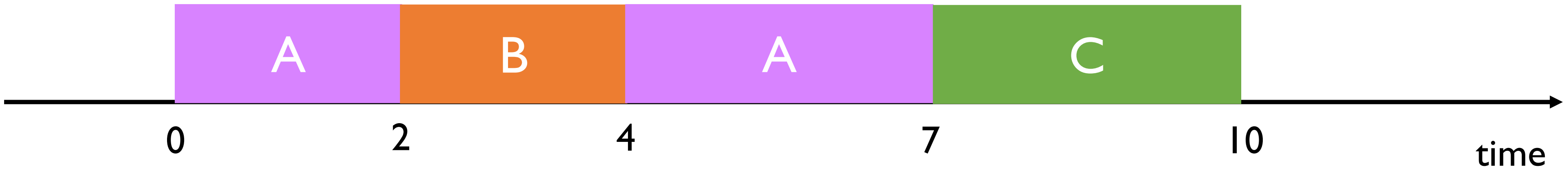
Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O



avg. waiting time = (2 + 2 + 7)/3 ~ **3.7**

FCFS: PROs and CONs

- **PRO:**
  - very simple!

FCFS: PROs and CONs

- **PRO:**
  - very simple!
- **CONs:**
  - (average) waiting time is highly variable as short CPU-burst jobs may sit behind very long ones

FCFS: PROs and CONs

- **PRO:**
  - very simple!
- **CONs:**
  - (average) waiting time is highly variable as short CPU-burst jobs may sit behind very long ones
  - **convoy effect** → poor overlap between CPU and I/O since CPU-bound jobs will force I/O bound jobs to wait

Scheduling Algorithms: An Overview

- First-Come-First-Serve (FCFS)
- **Round Robin (RR)**
- Shortest-Job-First (SJF)
- Priority Scheduling
- Multilevel Queue (MQ)
- Multilevel Feedback-Queue (MFQ)

Round Robin (RR)

- Similar to FCFS, except that CPU bursts are assigned with limits called **time quantum** or (**time slice**)
- When a job is given the CPU, a timer is set for a certain value:
  - If the job finishes before the time quantum expires, then it is swapped out of the CPU just like the normal FCFS algorithm
  - If the timer goes off first, then the job is swapped out of the CPU and moved to the back end of the ready queue
- Used in many time-sharing systems in combination with timer interrupts

Round Robin (RR)

- Similar to FCFS, except that CPU bursts are assigned with limits called **time quantum** or (**time slice**)
- When a job is given the CPU, a timer is set for a certain value:
  - If the job finishes before the time quantum expires, then it is swapped out of the CPU just like the normal FCFS algorithm
  - If the timer goes off first, then the job is swapped out of the CPU and moved to the back end of the ready queue
- Used in many time-sharing systems in combination with timer interrupts

Preemptive

Round Robin (RR)

- The **ready** queue is maintained as a **circular queue**
- When all jobs have had a turn, the scheduler gives the first job another turn, and so on…
- RR is fair as it shares the CPU equally among all the jobs
- The average waiting time can be longer than with other scheduling algorithms

Round Robin (RR): The Time Quantum

- The performance of RR is sensitive to the time quantum selected

Round Robin (RR): The Time Quantum

- The performance of RR is sensitive to the time quantum selected
- Too large time quantum degenerates to FCFS as jobs are never preempted from the CPU (high average waiting time)

Round Robin (RR): The Time Quantum

- The performance of RR is sensitive to the time quantum selected
- Too large time quantum degenerates to FCFS as jobs are never preempted from the CPU (high average waiting time)
- Too small time quantum implies more context switches, which eventually dominate over the actual CPU utilization (low throughput)

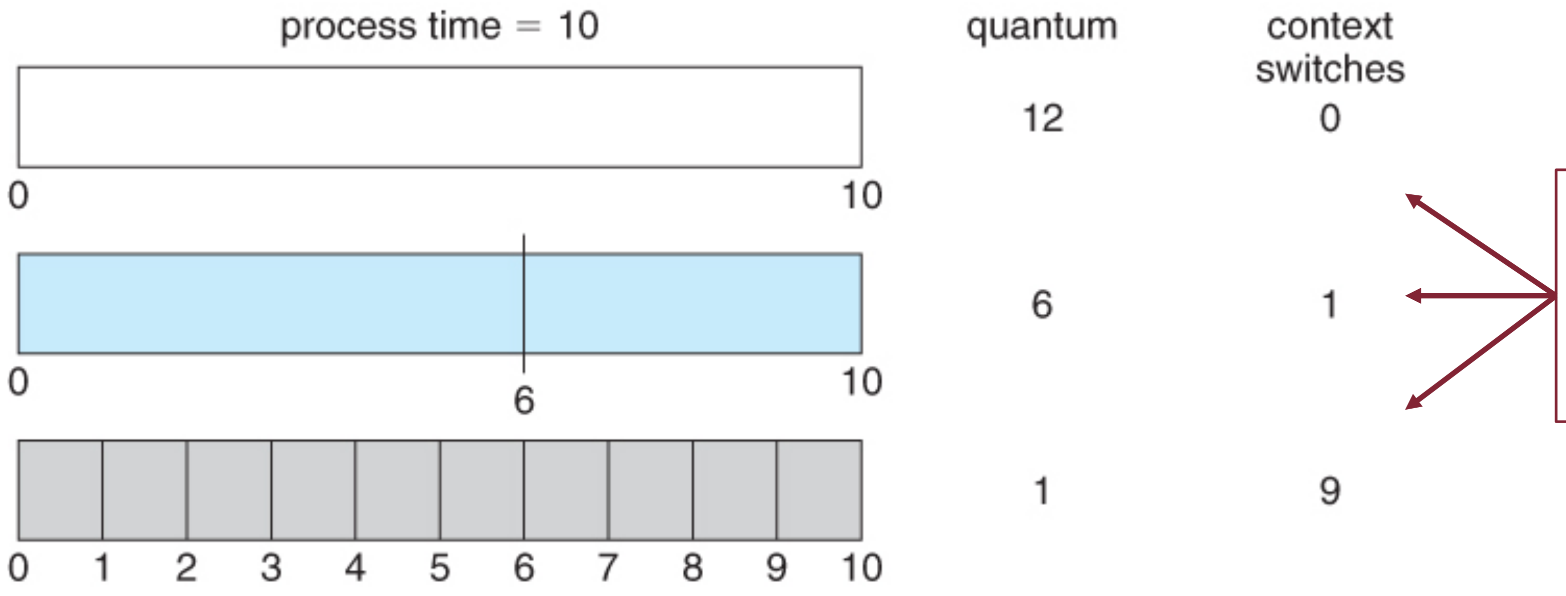
Round Robin (RR): The Time Quantum

- The performance of RR is sensitive to the time quantum selected
- Too large time quantum degenerates to FCFS as jobs are never preempted from the CPU (high average waiting time)
- Too small time quantum implies more context switches, which eventually dominate over the actual CPU utilization (low throughput)

Trade-off:
Overhead for context switching should be <u>relatively small</u> compared to time slice

Example: time slice = 10÷100 msec. and context switch = 0.01÷0.1 msec.

Round Robin (RR): The Time Quantum



By decreasing the time quantum the number of context switches increase

Round Robin (RR): The Time Quantum

$$N = \text{number of jobs}$$

$$\delta = \text{time slice}$$

$$\sup\{T_i^{start}\} = \delta * (i-1),\ \forall i \in \{1, \ldots, N\}$$

upper-bound on the
time a job is scheduled for the first time

worst-case scenario:
all job in front of the queue will use the whole time slice

Round Robin (RR): Example

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

Round Robin (RR): Example

New | A | B | C |

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

Round Robin (RR): Example

New | A | B | C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

No I/O burst

Time quantum = 2

Context switch = 0

Round Robin (RR): Example

New A B C

Ready A B C

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

Round Robin (RR): Example

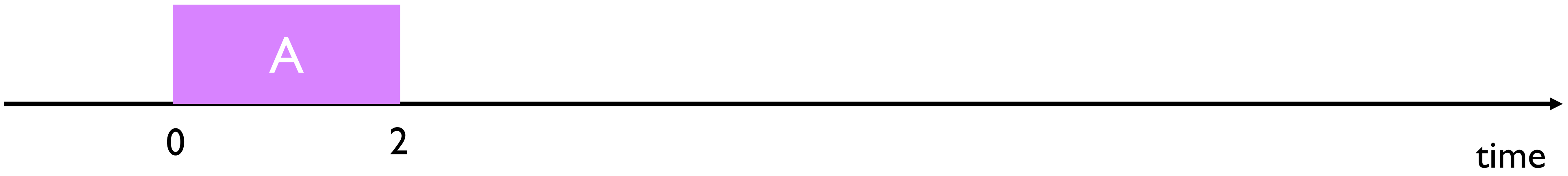
New: A B C

Ready: B C

Waiting

Running: A

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



Round Robin (RR): Example

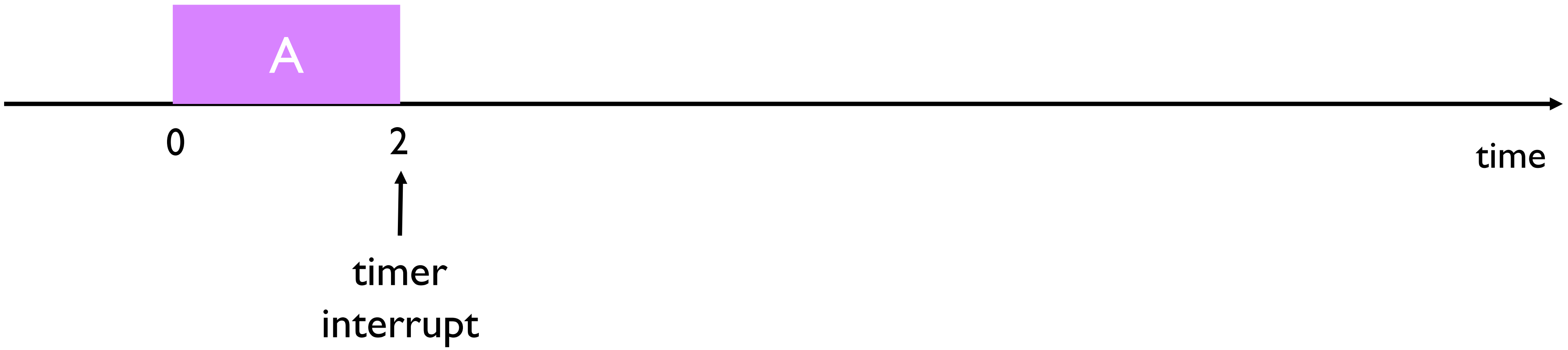
New A B C

Ready B C

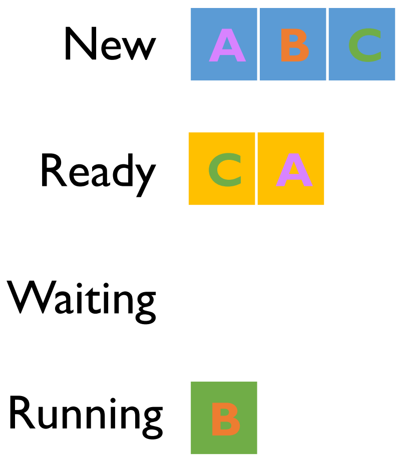
Waiting

Running A

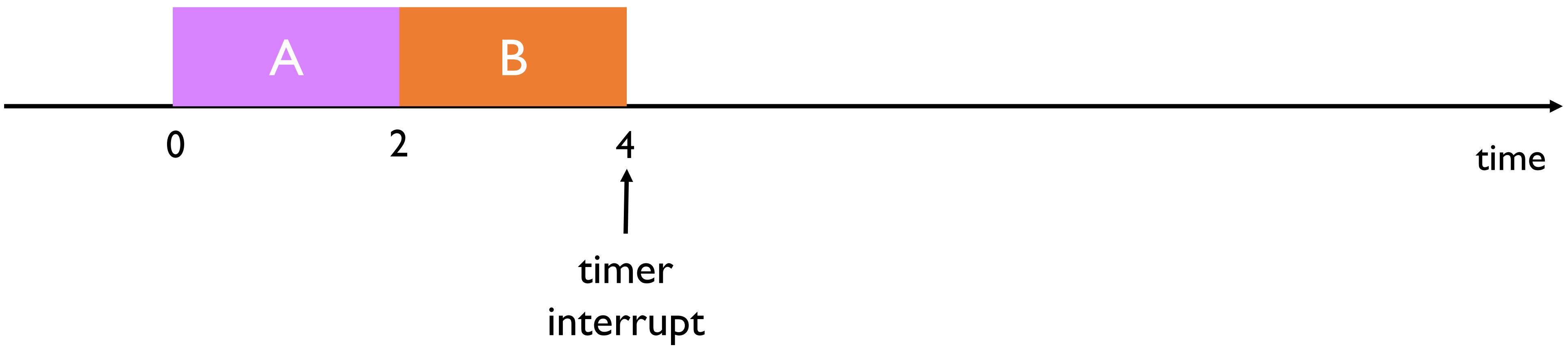
Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



Round Robin (RR): Example

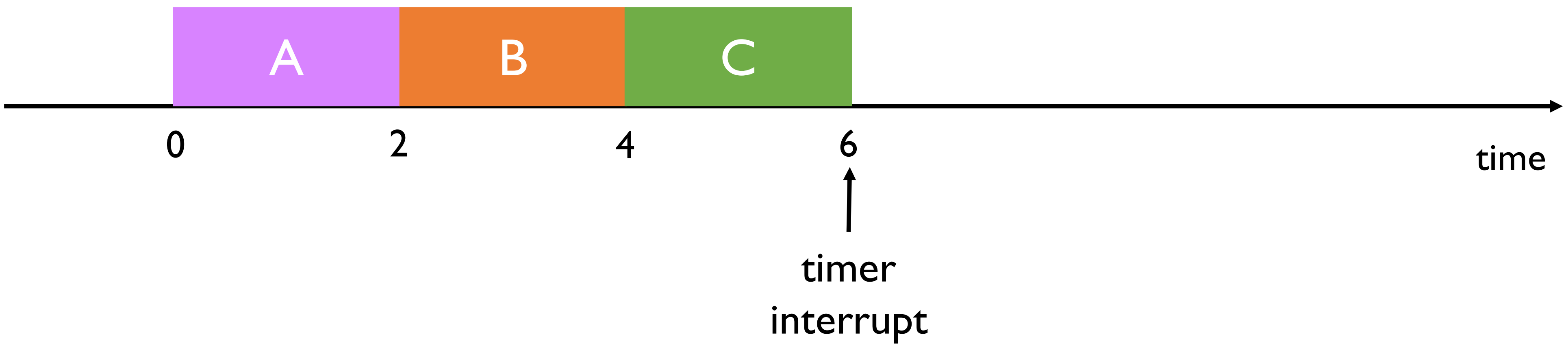


Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

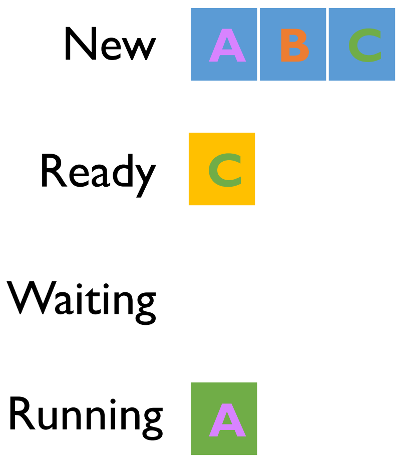


Round Robin (RR): Example

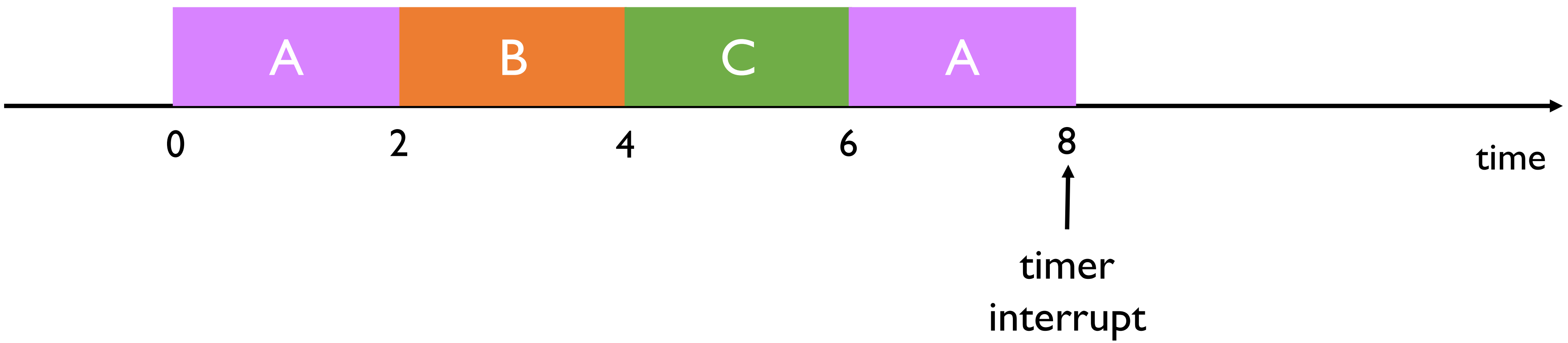
Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



Round Robin (RR): Example



Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



Round Robin (RR): Example

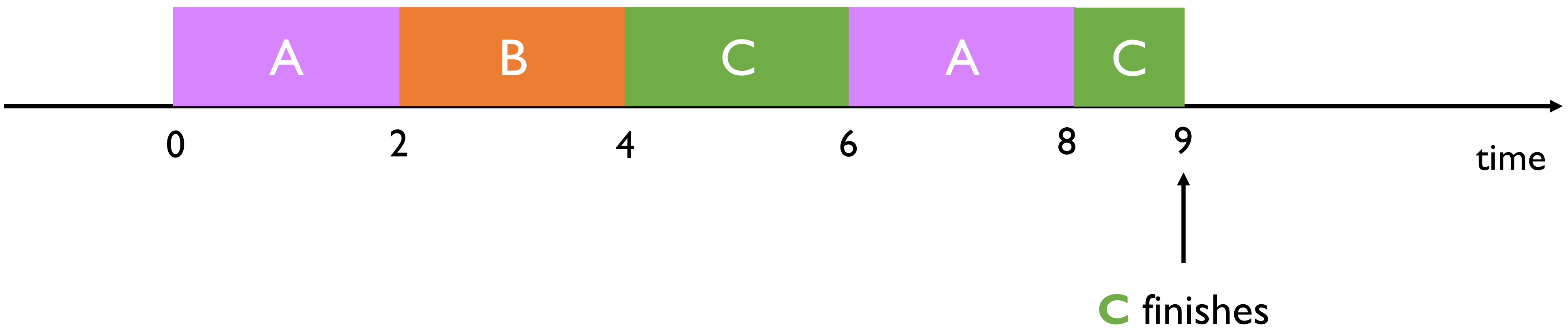
New A B C

Ready A

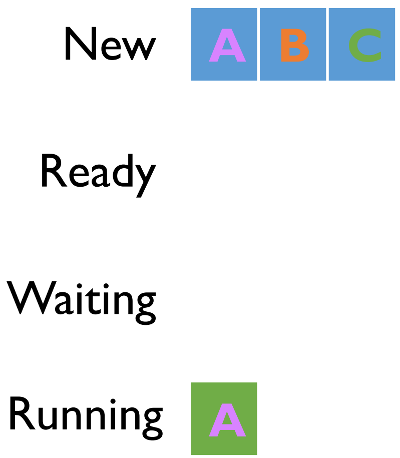
Waiting

Running C

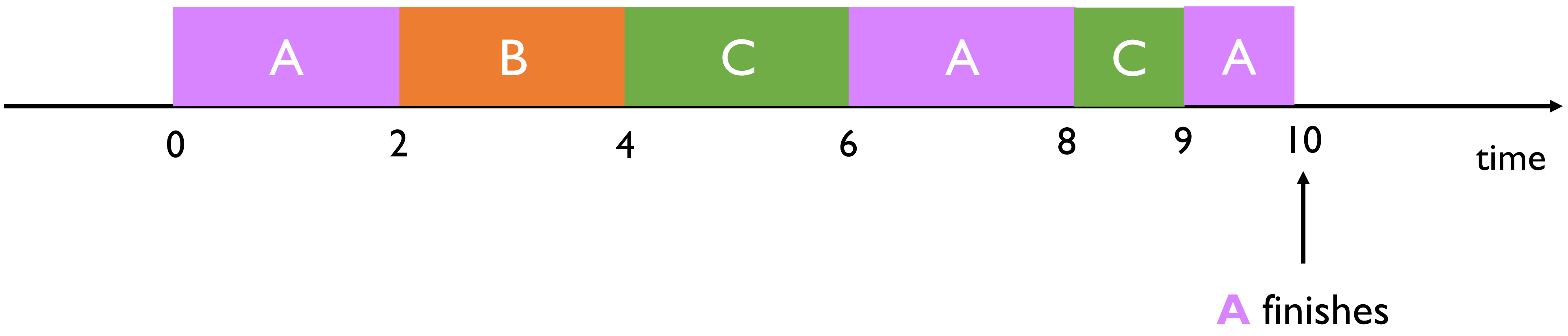
Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



Round Robin (RR): Example



Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



Round Robin (RR): Example

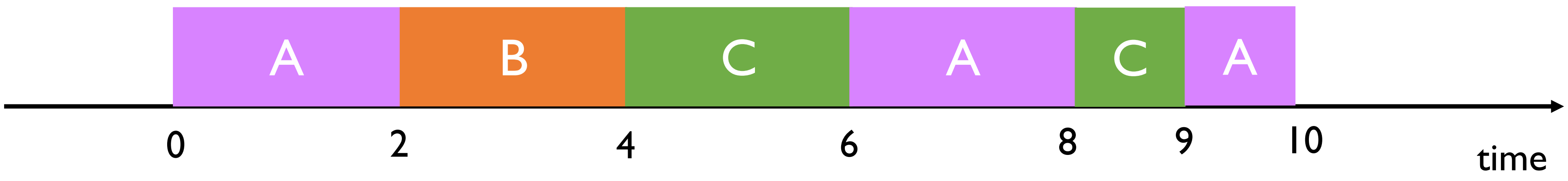
New A B C

Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



avg. waiting time =

Round Robin (RR): Example

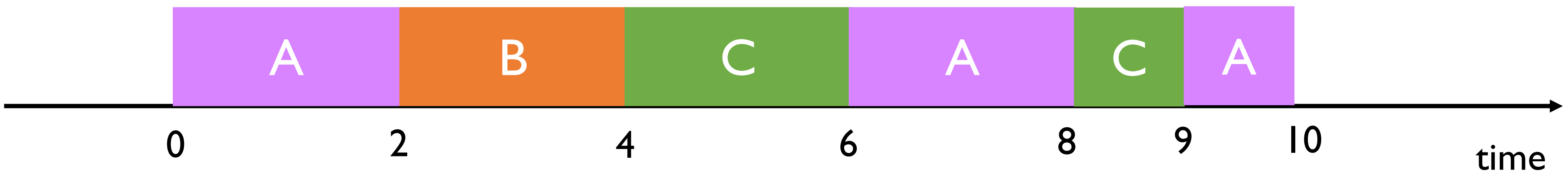
New A B C

Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



avg. waiting time = (5 + 2 + 6)/3 ~ **4.3**

RR vs. FCFS

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	100				
B	100				
C	100				
D	100				
E	100				
Avg.					

RR vs. FCFS

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	100	100			
B	100	200			
C	100	300			
D	100	400			
E	100	500			
Avg.		**300**			

RR vs. FCFS

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	100	100	496		
B	100	200	497		
C	100	300	498		
D	100	400	499		
E	100	500	500		
Avg.		**300**	**498**		

RR vs. FCFS

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	100	100	496	0	
B	100	200	497	100	
C	100	300	498	200	
D	100	400	499	300	
E	100	500	500	400	
Avg.		**300**	**498**	**200**	

RR vs. FCFS

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	100	100	496	0	396
B	100	200	497	100	397
C	100	300	498	200	398
D	100	400	499	300	399
E	100	500	500	400	400
Avg.		**300**	**498**	**200**	**398**

RR vs. FCFS

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	100	100	496	0	396
B	100	200	497	100	397
C	100	300	498	200	398
D	100	400	499	300	399
E	100	500	500	400	400
Avg.		**300**	**498**	**200**	**398**

FCFS seems to outperform RR in both metrics but… is it fair?

Look at the variance rather than the average!

RR vs. FCFS

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	50				
B	40				
C	30				
D	20				
E	10				
Avg.					

RR vs. FCFS

Assumptions:

- 5 jobs, different CPU burst
- Time quantum = 1
- Context switch = 0
- Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	50	50			
B	40	90			
C	30	120			
D	20	140			
E	10	150			
Avg.		**110**			

RR vs. FCFS

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	50	50	150		
B	40	90	140		
C	30	120	120		
D	20	140	90		
E	10	150	50		
Avg.		**110**	**110**		

RR vs. FCFS

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	50	50	150	0	
B	40	90	140	50	
C	30	120	120	90	
D	20	140	90	120	
E	10	150	50	140	
Avg.		**110**	**110**	**80**	

RR vs. FCFS

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	**CPU burst**	**FCFS**	**RR**	**FCFS**	**RR**
A	50	50	150	0	100
B	40	90	140	50	100
C	30	120	120	90	90
D	20	140	90	120	70
E	10	150	50	140	40
Avg.		**110**	**110**	**80**	**80**

Summary (Part 1)

- Scheduling allows one process to use the CPU while another is waiting for I/O, thereby maximizing system utilization

Summary (Part 1)

- Scheduling allows one process to use the CPU while another is waiting for I/O, thereby maximizing system utilization
- **non-preemptive** vs. **preemptive** scheduler

Summary (Part 1)

- Scheduling allows one process to use the CPU while another is waiting for I/O, thereby maximizing system utilization
- **non-preemptive** vs. **preemptive** scheduler
- Different scheduling policies optimize different metrics

Summary (Part 1)

- Scheduling allows one process to use the CPU while another is waiting for I/O, thereby maximizing system utilization
- **non-preemptive** vs. **preemptive** scheduler
- Different scheduling policies optimize different metrics
- 2 out of 6 scheduling algorithms:
  - **First-Come-First-Serve** (**FCFS**)
  - **Round Robin** (**RR**)