Operating System Concepts

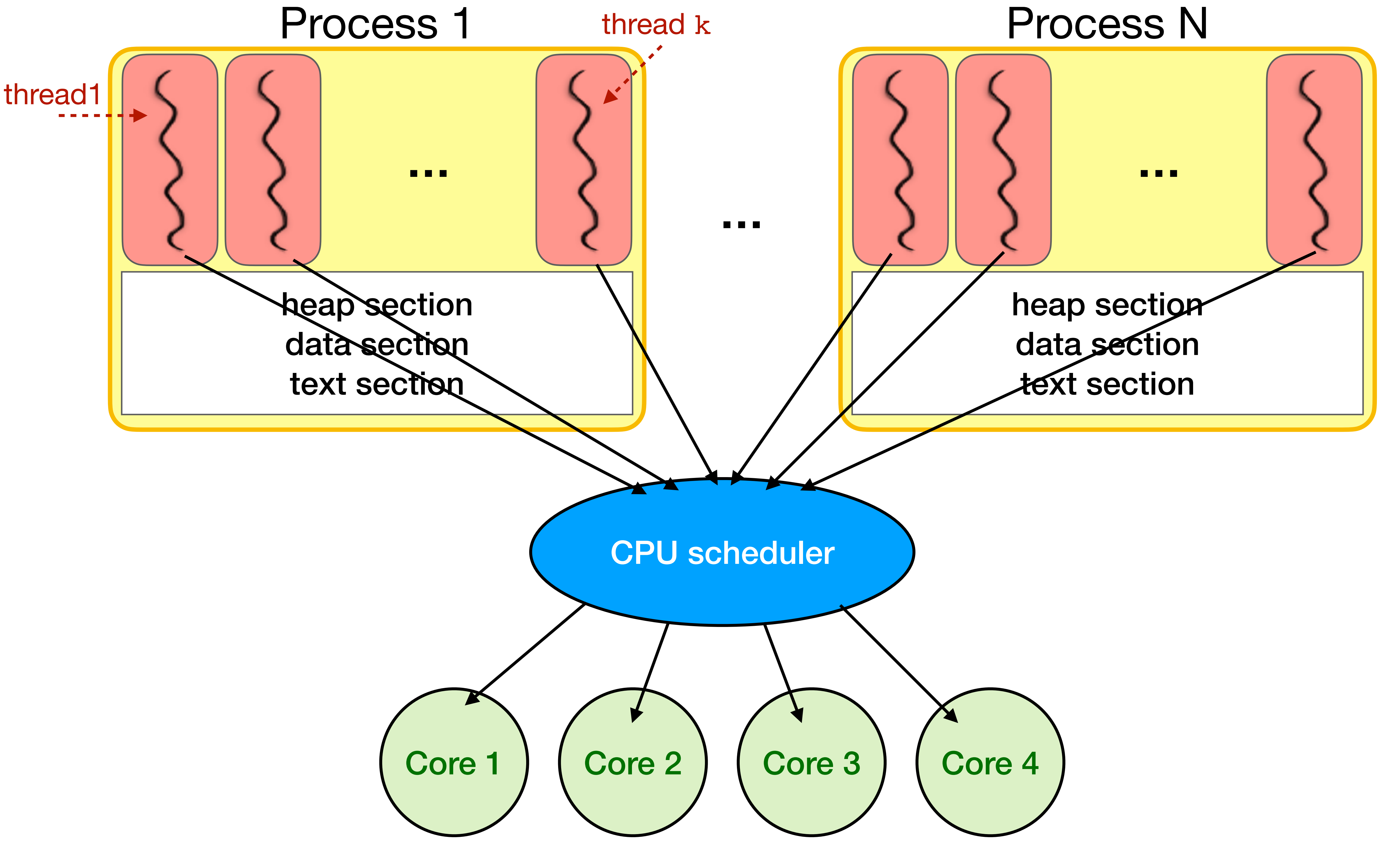
Lecture 12: CPU Scheduling - Part 2

Omid Ardakanian
oardakan@ualberta.ca
University of Alberta

Recap

- Turnaround time: time elapsed from task submission until task completion
- Response time: time elapsed from task submission until the first response is produced
- Waiting time: total time spent by a task waiting in the ready queue
- Throughput: rate of task completion
- CPU utilization: percentage of time CPU is busy (i.e. the ready queue is not empty)

Process versus thread scheduling

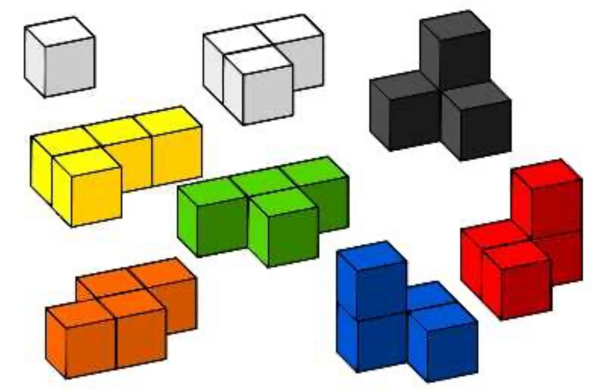


Assumptions

- We make a couple of simplifying assumptions today
  - one process per user
  - one thread per process
  - independent processes
  - just one processing core
- Scheduling algorithms were developed in the 70's when these assumptions were realistic

Today's class

- Scheduling algorithms
  - FCFS: First-Come, First-Served
  - RR: Round Robin
  - SJF: Shortest Job First



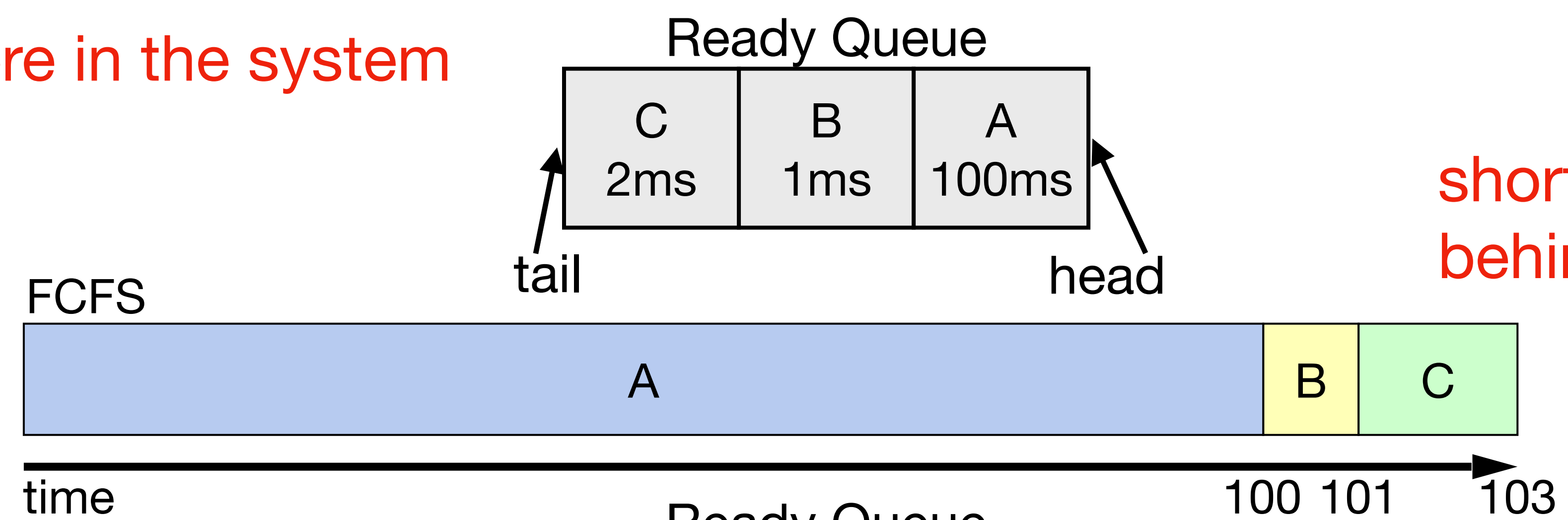
FCFS scheduling

- FCFS: First-Come-First-Served (or FIFO)
  - the scheduler executes jobs **to completion** in the order they arrive
  - in early FCFS schedulers, the job did not relinquish the CPU even when it was doing I/O
  - we will assume a FCFS scheduler that runs when processes are blocked waiting for I/O, but that is **non-preemptive**, i.e., the job keeps the CPU until it blocks (say on an I/O device)

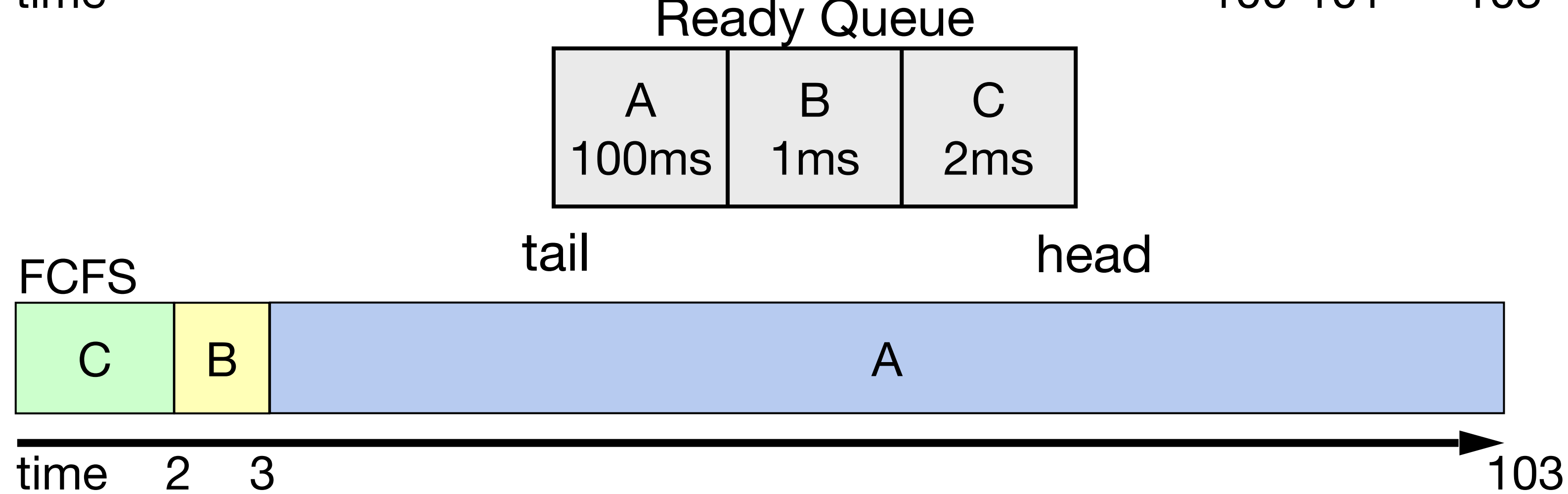
FCFS example

A, B, C were in the system at time 0

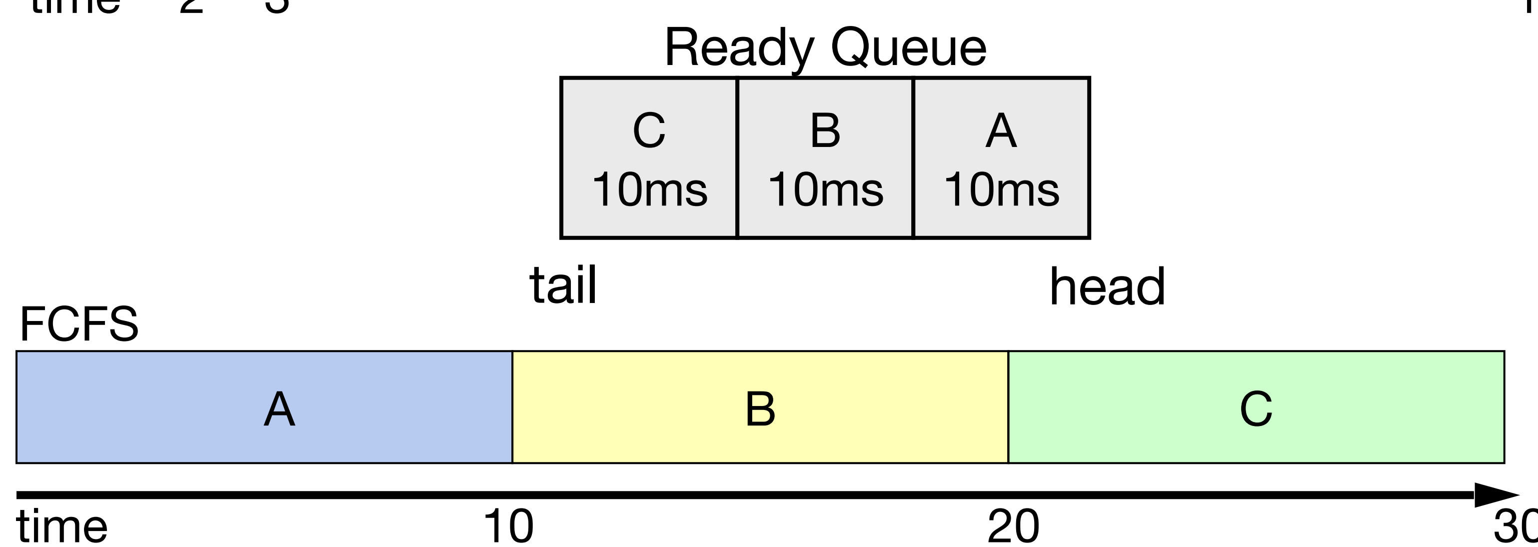
short processes stuck behind long processes



Avg. turnaround time: (100+101+103)/3=101.3
Avg. waiting time: (0+100+101)/3=67



Avg. turnaround time: 36
Avg. waiting time: 1.6

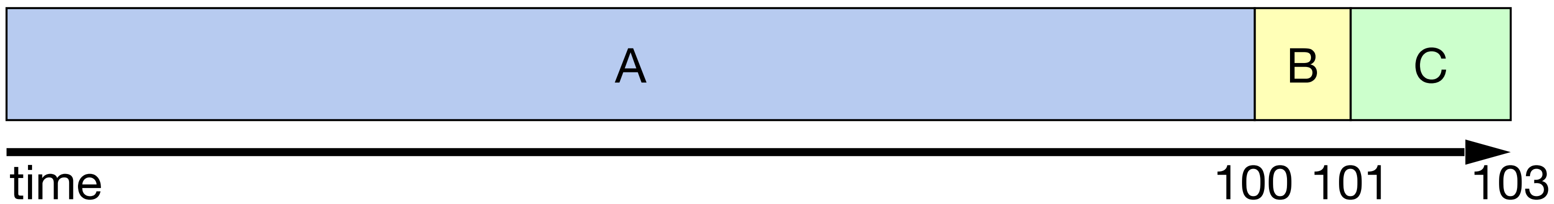


Avg. turnaround time: 20
Avg. waiting time: 10

FCFS example

Process	Burst length	Arrival time
A	100	0
B	1	10
C	2	50

FCFS



Avg. turnaround time:
[100+(101-10)+(103-50)]/3 = 81.3

Avg. waiting time:
[0+(100-10)+(101-50)]/3= 47

Pros and cons of FCFS scheduling

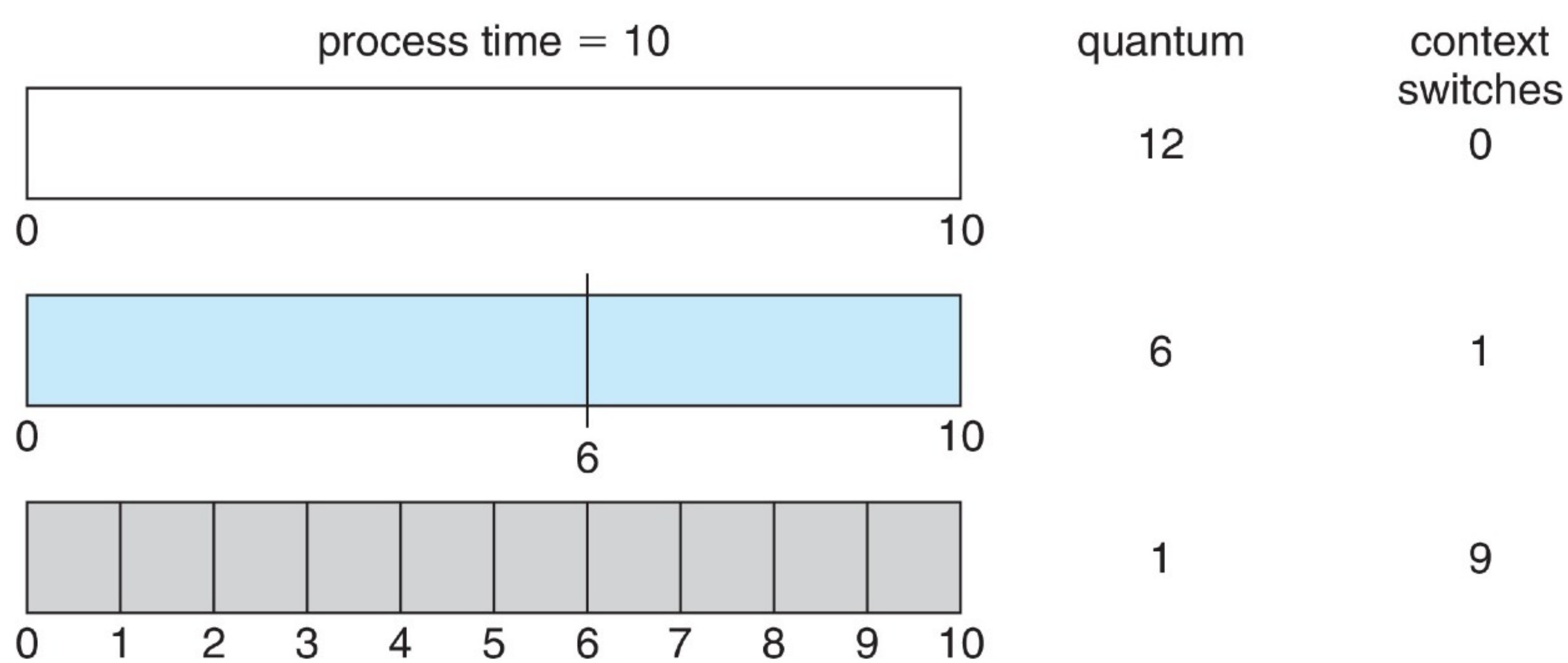
- Advantage: simplicity and low overhead
- Disadvantage:
  - average wait time is highly variable as short jobs may wait behind long jobs
    - if tasks are variable in size, FCFS can cause **poor response time** on average
    - If tasks are equal in size, FCFS is **optimal** in terms of **average response time**
  - not fair
  - may lead to poor overlap of I/O and CPU since CPU bound processes will force I/O bound processes to wait for the CPU, leaving the I/O devices idle

Round Robin scheduling

- Each task gets resources for a fixed period of time (**time quantum**)
  - if it does not finish its execution, it goes back in line (inserted at the end of the ready queue)
  - how to implement? add a timer and use a **preemptive** policy
- With quantum length Q *ms*, process waits at most (N-1)*Q *ms* to run again if there is a total of N processes

Round Robin scheduling

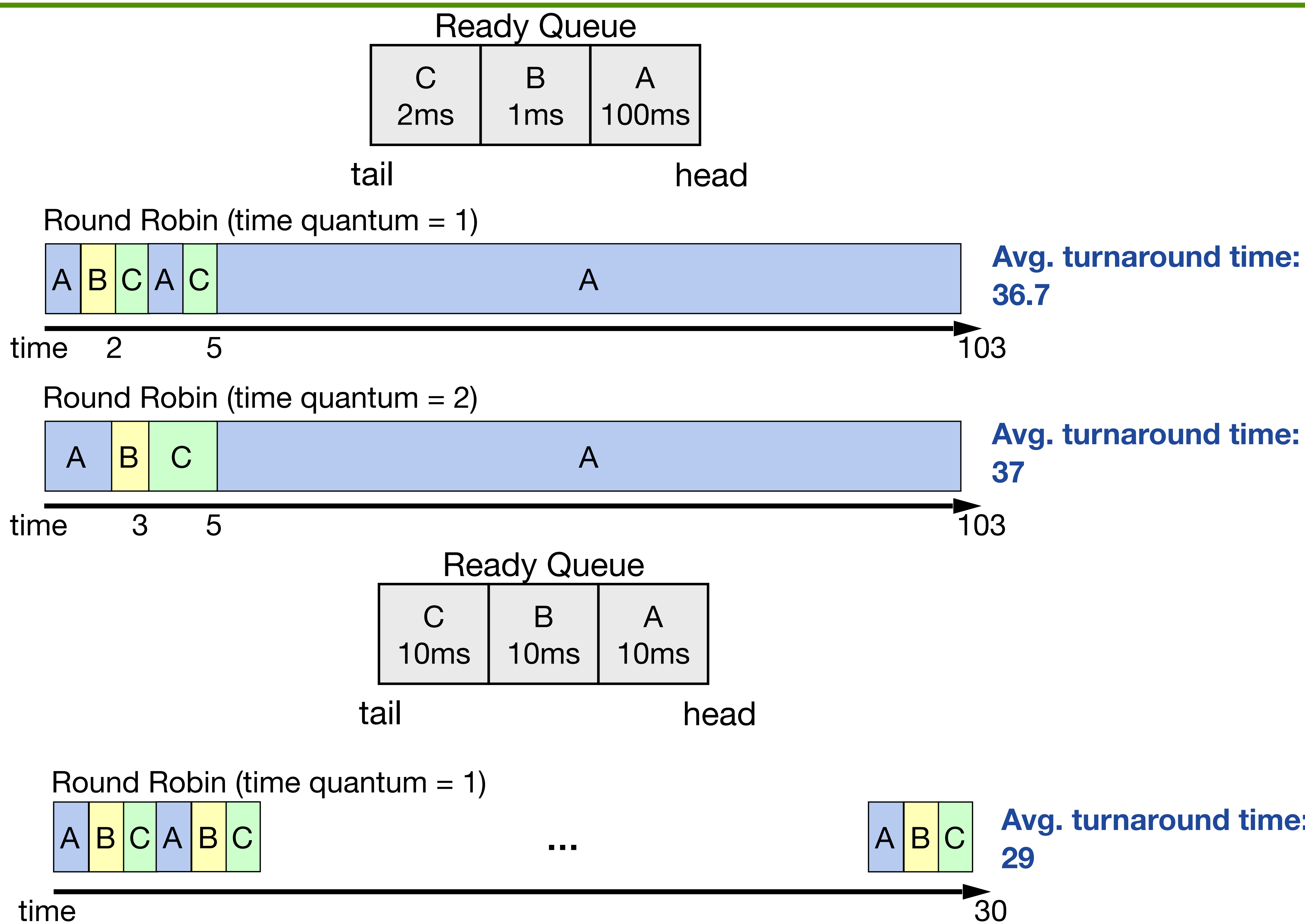
- Choosing the time quantum (Q) is key
  - if too long - waiting time suffers, degenerates to FCFS if processes are never preempted
  - if too short - throughput suffers because too much time is spent context switching (high overhead)
- It is possible to strike a balance between waiting time and throughput by selecting a time slice where context switching is roughly 1% of the time slice
  - today: typical time slice is 10-100ms, context switch time is 0.1-1ms



Pros and cons of Round Robin scheduling

- Variants of the round robin scheduling are used in most time-sharing systems
- Advantage: round robin is **fair**; each job gets an equal shot at the CPU
- Disadvantage: average waiting time can be bad if tasks are equal in size

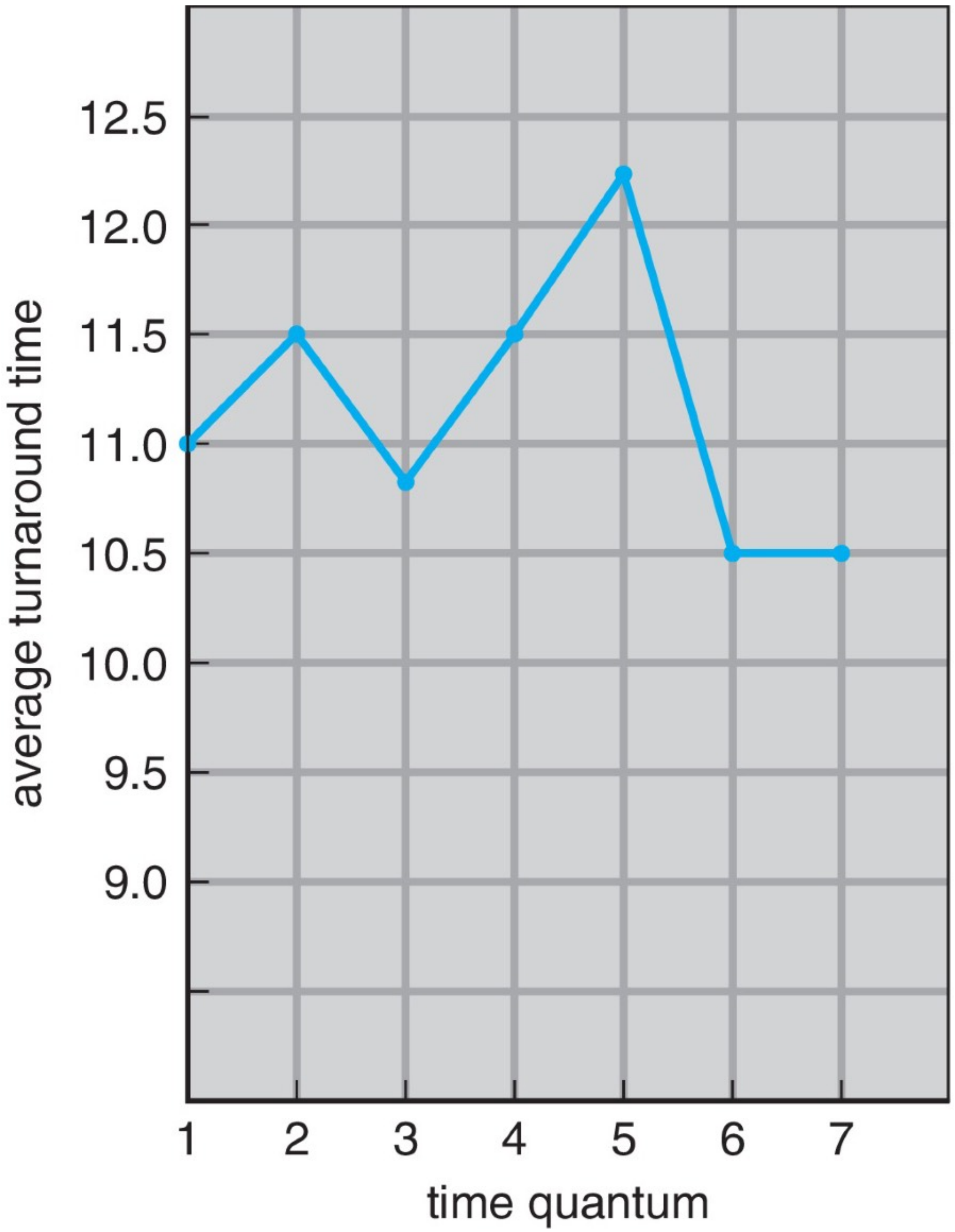
RR example



assume all processes arrive at t=0

Setting the time quantum

suppose there is no context switching overhead and all process arrive at t=0



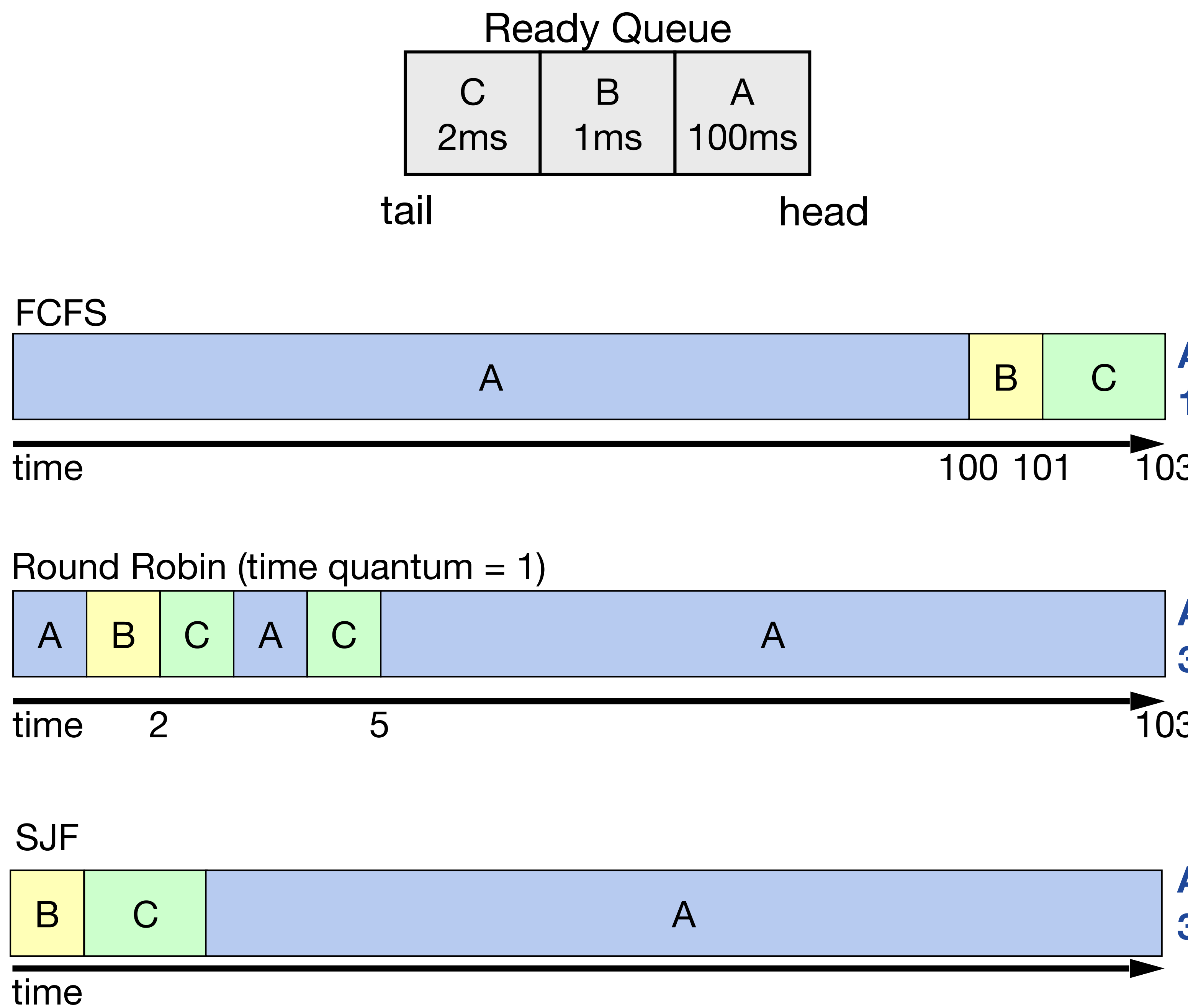
process	time
P_1	6
P_2	3
P_3	1
P_4	7

increasing the time quantum can have different effects on the average turnaround time

SJF scheduling

- Schedule the job that has the least (expected) amount of work (CPU time) to do until its next I/O request or termination
  - much less sensitive to the arrival order!
- If tasks are variable in size, Round Robin approximates SJF
- Advantages
  - provably optimal with respect to **minimizing the average waiting time**
  - works for preemptive and non-preemptive systems
    - preemptive SJF is called Shortest Remaining Time First (SRTF)
- Disadvantages
  - it is not possible to accurately predict the amount of CPU time that a job needs
  - with SRTF, long running CPU bound jobs can **starve** (if new short jobs keep arriving)

Scenario A

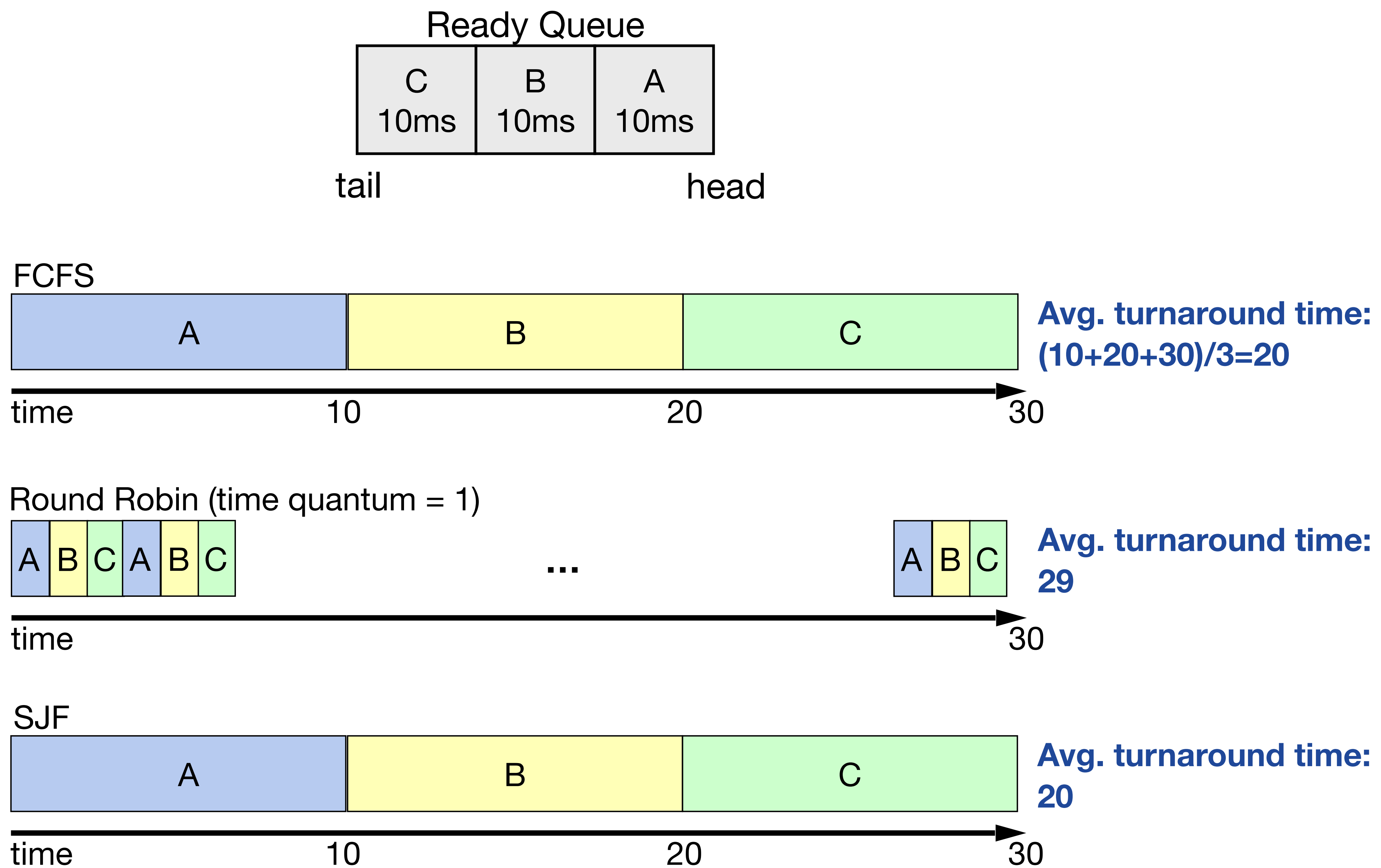


Avg. turnaround time: 101.3

Avg. turnaround time: 36.7

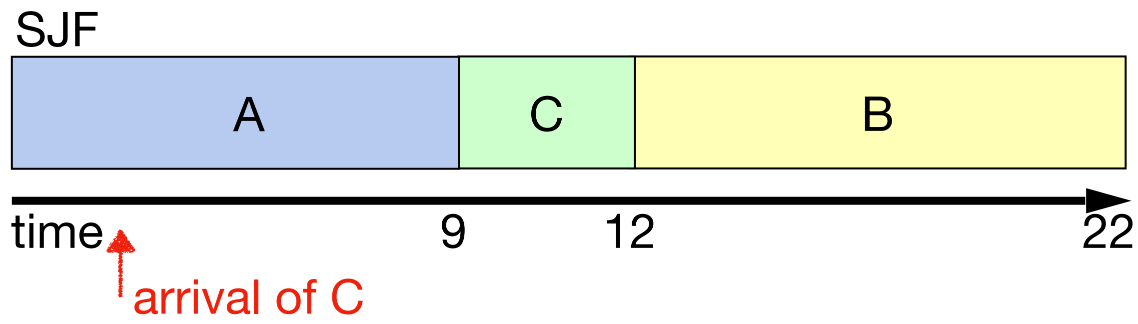
Avg. turnaround time: 35.7

Scenario B

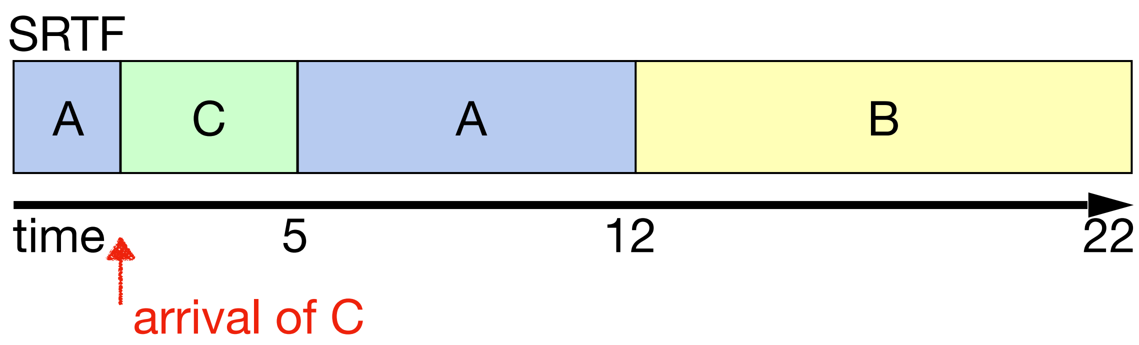

Ready Queue
C
10ms
B
10ms
A
10ms
tail
head
FCFS
A
B
C
time
10
20
30
Avg. turnaround time:
(10+20+30)/3=20
Round Robin (time quantum = 1)
A
B
C
A
B
C
...
A
B
C
time
30
Avg. turnaround time:
29
SJF
A
B
C
time
10
20
30
Avg. turnaround time:
20

Comparing SRTF and SJF

Process	Burst length	Arrival time
A	9	0
B	10	0
C	3	2



Avg. turnaround time:
[9+(12-2)+22]/3=13.6



Avg. turnaround time:
[(5-2)+12+22]/3=12.3

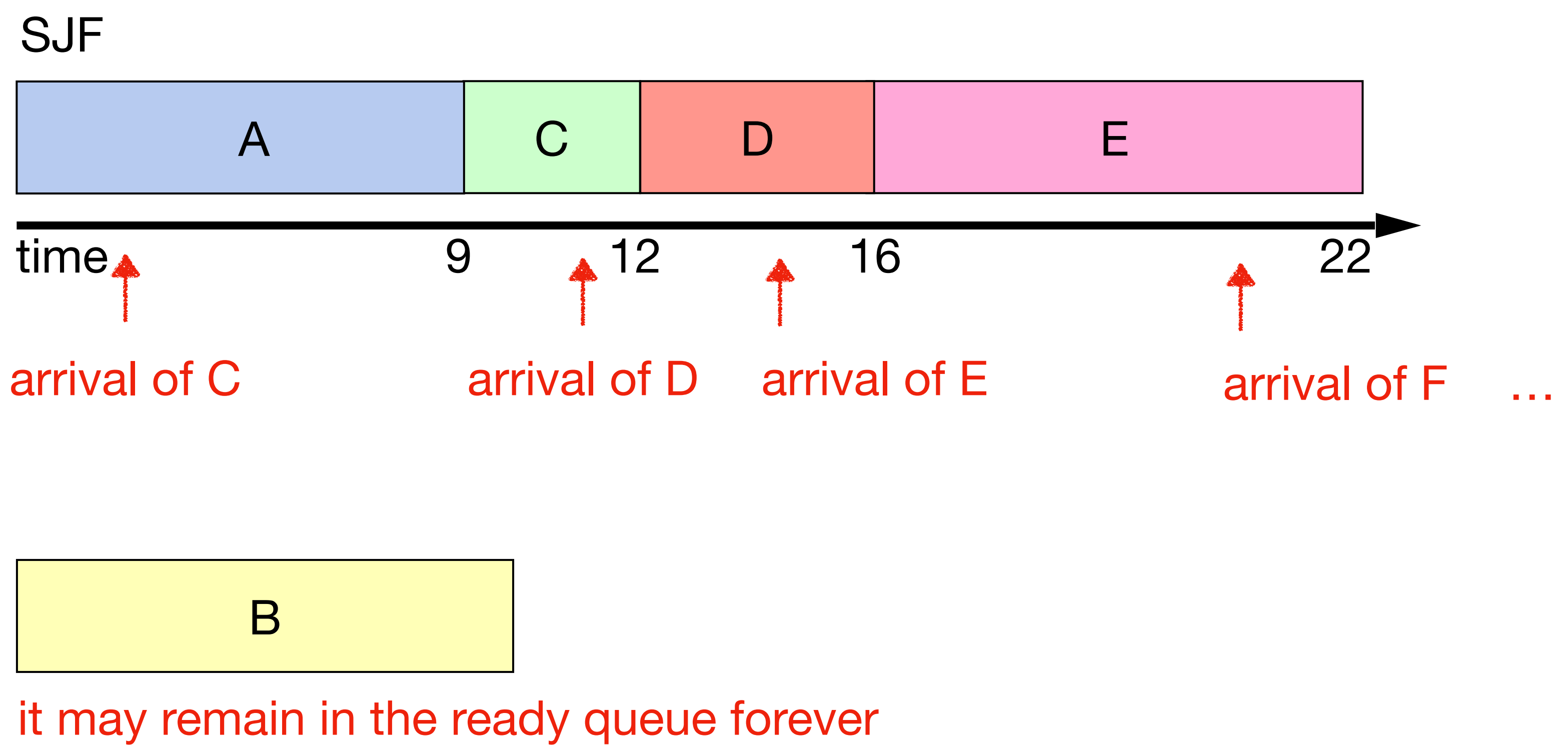
But how do we know the burst length of a process?

- Burst length can be estimated based on previous burst lengths
  - the exponential moving average estimator gives more importance to the recent past
    - $\hat{b}[1] = b[1]$
    - $\hat{b}[t] = \eta b[t] + (1 - \eta)\hat{b}[t-1]$ for $\eta \in (0,1]$
- Users can provide a burst length of their task
  - they can lie (declare a shorter burst length) to game the system
  - how to encourage truthfulness? terminate execution after the specified burst length has passed

SJF is starvation prone

Is SRJT starvation prone too?

Process	Burst length	Arrival time
A	9	0
B	10	0
C	3	2
D	4	11
E	6	14



Homework

- Suppose 1 long process (with burst size of 100 units) and n short processes (with burst size of 1 unit each) arrive in the system in a random order at time 0. Under the SJF policy
  - what is the probability that a short process gets stuck behind the large task?
  - how long is the long process expected to wait?