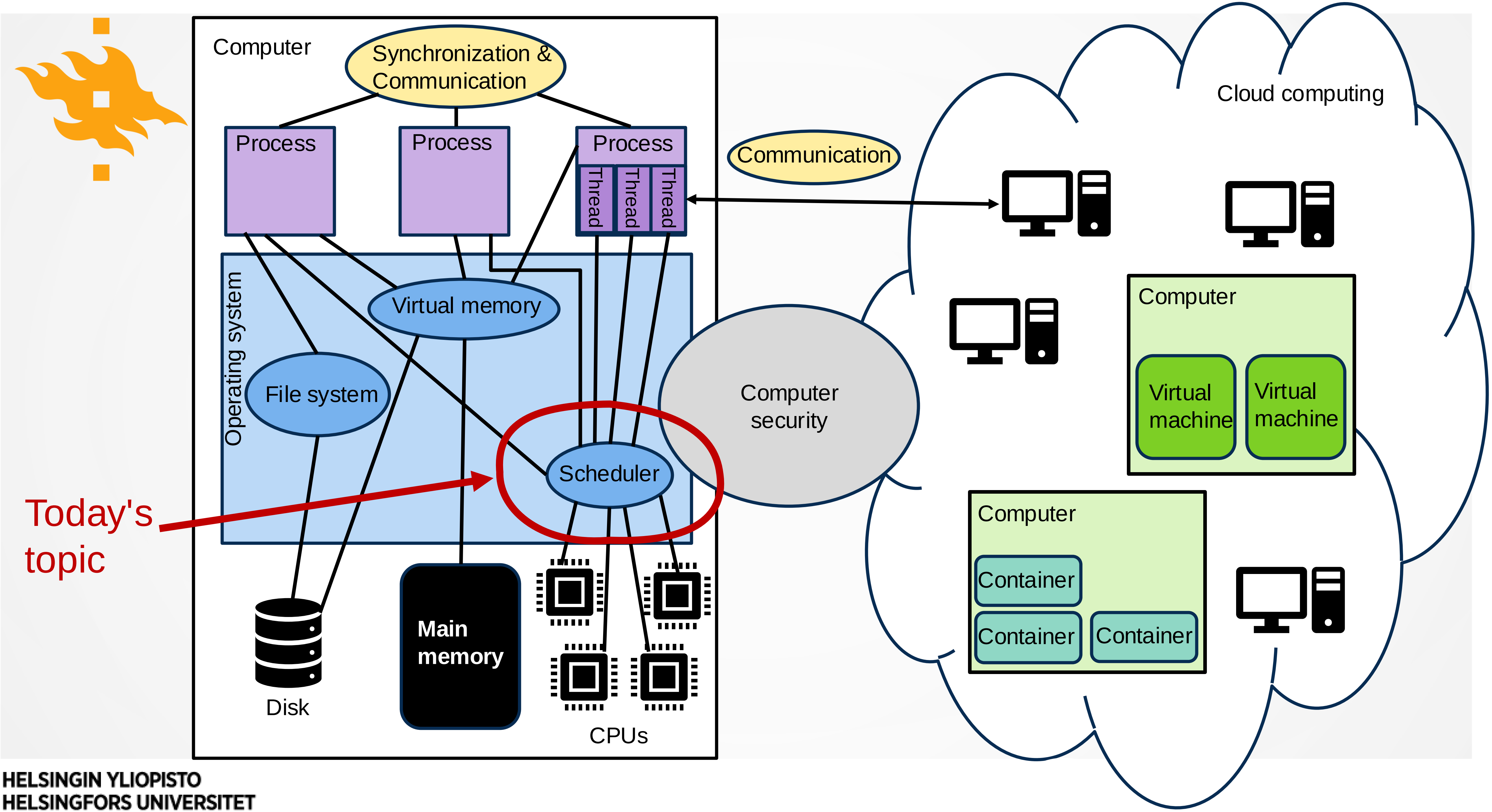
COMPUTING PLATFORMS

Scheduling:
Simple algorithms on uniprocessor systems


Computer
Synchronization & Communication
Process
Process
Process
Thread
Thread
Thread
Communication
Cloud computing
Operating system
Virtual memory
File system
Scheduler
Computer security
Computer
Virtual machine
Virtual machine
Computer
Container
Container
Container
Today's topic
Main memory
Disk
CPUs

LEARNING OUTCOMES

- After today's lecture, you
  - Understand why and when scheduling is needed
  - Know of different objectives for process scheduling
  - Are able to describe and apply simple algorithms for uniprocessor scheduling

WHY SCHEDULING?

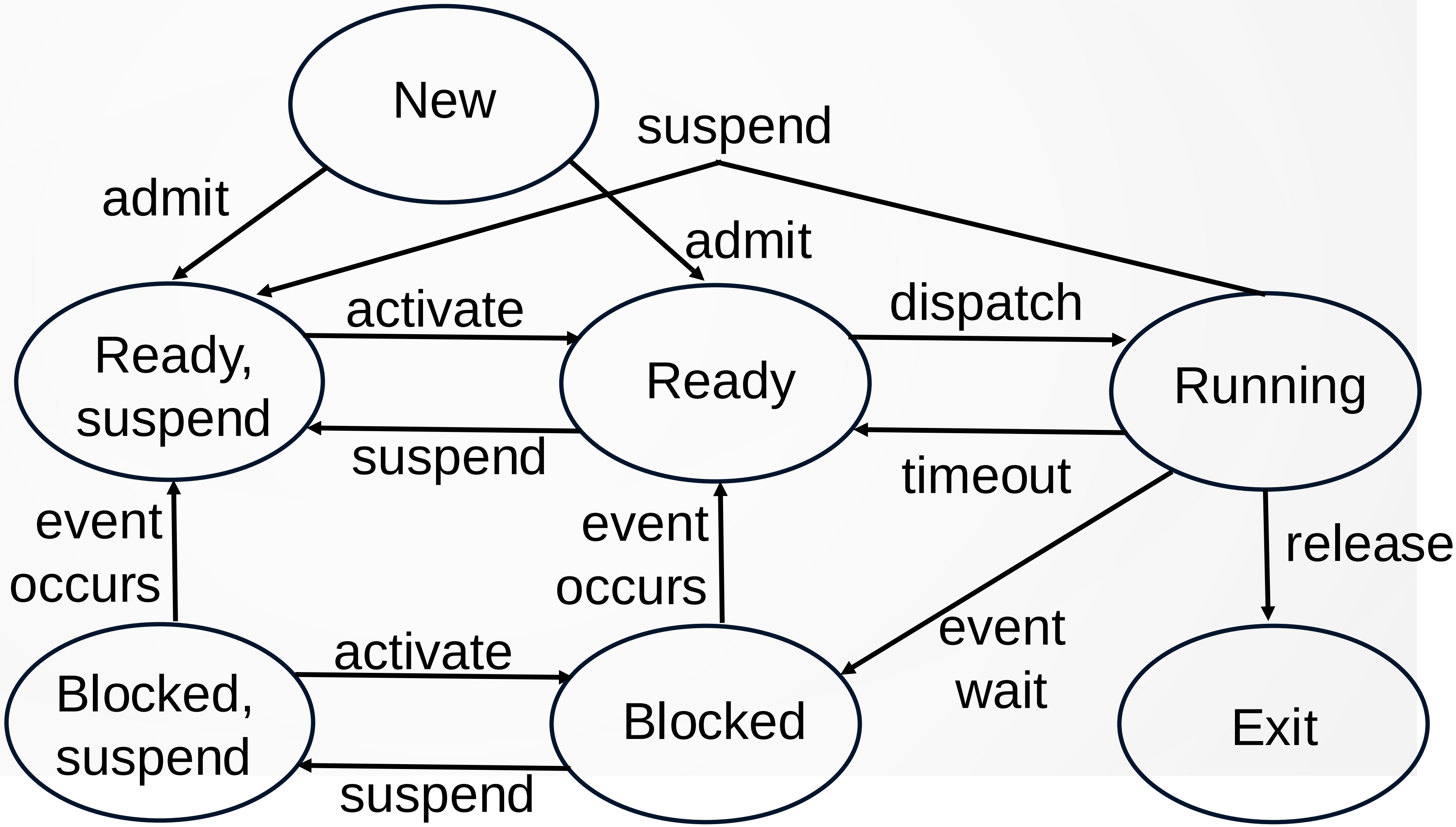
- A single CPU can process one thread at a time
- Many processes, few CPUs
- Example: Some processes running on my computer with an i5 processor having 4 cores with hyperthreading

```
lmsalmel@lx8-500-101: ~
File  Edit  View  Search  Terminal  Help
top - 14:15:50 up  6:10,  1 user,  load average: 2.09, 1.96, 1.87
Tasks: 343 total,   2 running, 341 sleeping,   0 stopped,   0 zombie
%Cpu(s): 18.2 us,  2.1 sy,  0.0 ni, 79.6 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :  15895.4 total,   7554.7 free,   3589.6 used,   4751.0 buff/cache
MiB Swap:  16428.0 total,  16428.0 free,      0.0 used.  10754.6 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   3983 lmsalmel  20   0  501812 155620 123924 S  57.5   1.0  27:51.92 RDD Process
   3750 lmsalmel  20   0 2986176 314828 113976 R  41.9   1.9 201:44.03 Isolated Web Co
   3488 lmsalmel  20   0   13.2g 758808 316864 S  33.2   4.7 132:48.98 firefox-bin
  21888 lmsalmel  20   0 3132372 485048 121560 S  12.3   3.0   1:54.44 Isolated Web Co
  21180 lmsalmel  20   0 2599044 121924  98068 S   6.6   0.7   0:48.12 Isolated Web Co
   2187 lmsalmel  20   0 5617440 318184 150988 S   6.3   2.0  27:50.58 gnome-shell
   4067 lmsalmel  20   0 3120644 393372 115856 S   5.3   2.4   1:55.54 Isolated Web Co
   2581 lmsalmel  20   0  837876  94104  60324 S   3.3   0.6  19:20.48 Xwayland
   4090 lmsalmel  20   0  347960  47672  35460 S   1.7   0.3   0:17.53 Utility Process
  15351 root      20   0       0      0      0 I   1.3   0.0   1:33.59 kworker/6:2-events
   2017 lmsalmel   9 -11 2824488  30732  21960 S   1.0   0.2  20:42.74 pulseaudio
  22670 lmsalmel  20   0   36760   4524   3712 R   1.0   0.0   0:00.08 top
     53 root      20   0       0      0      0 S   0.7   0.0   7:23.98 ksoftirqd/6
     15 root      20   0       0      0      0 I   0.3   0.0   0:20.71 rcu_preempt
```

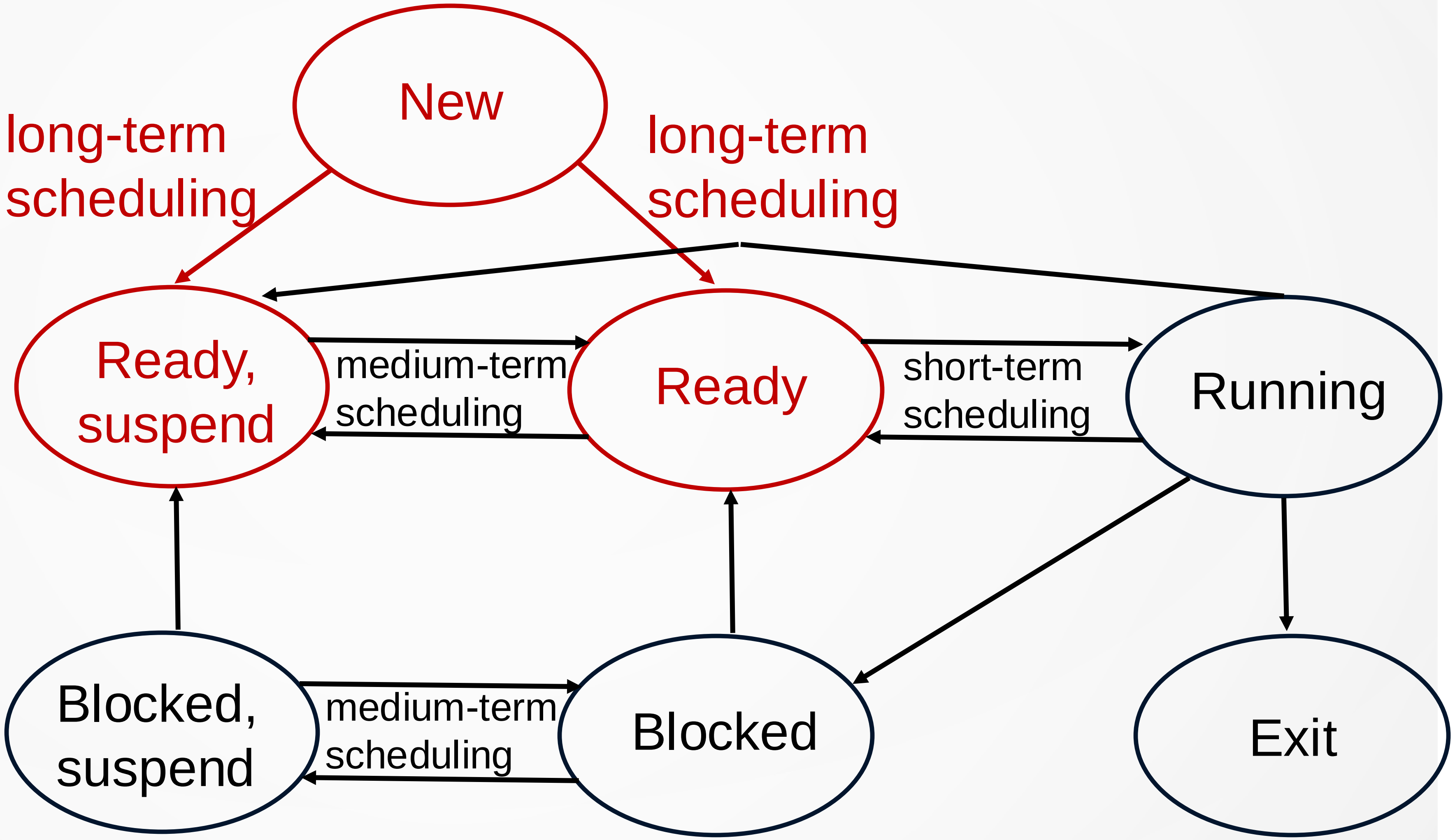
WHEN TO SCHEDULE?

- Recall **process states** and **state transitions**
- Several transitions need **scheduling decisions**: new process created, process exits or is blocked, I/O interrupt



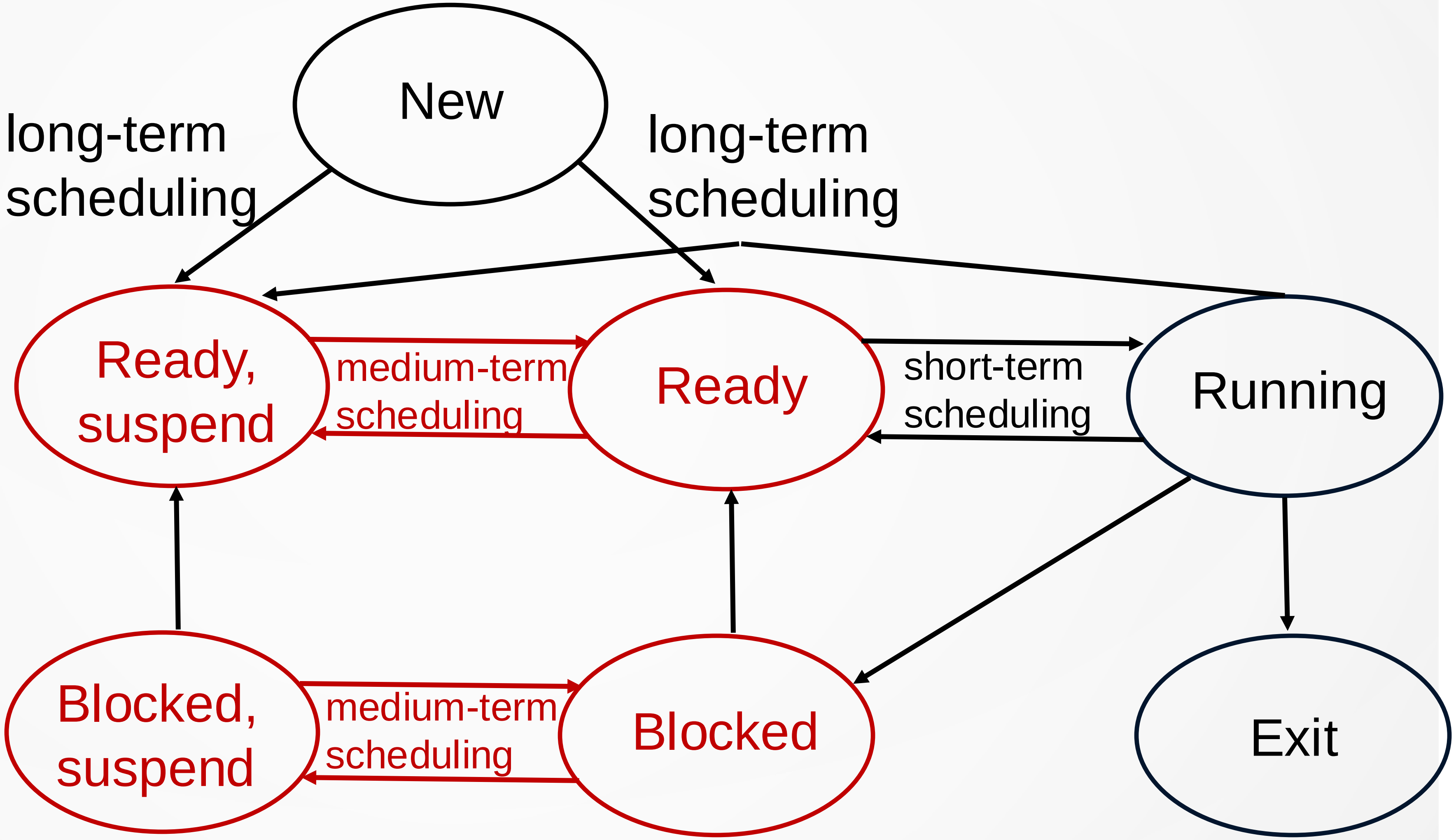
LONG-TERM SCHEDULING

- Which process to admit to the system?
- Controls level of **multiprogramming**, important in **batch systems**
- Ideally a good mix of CPU- and IO-bound processes



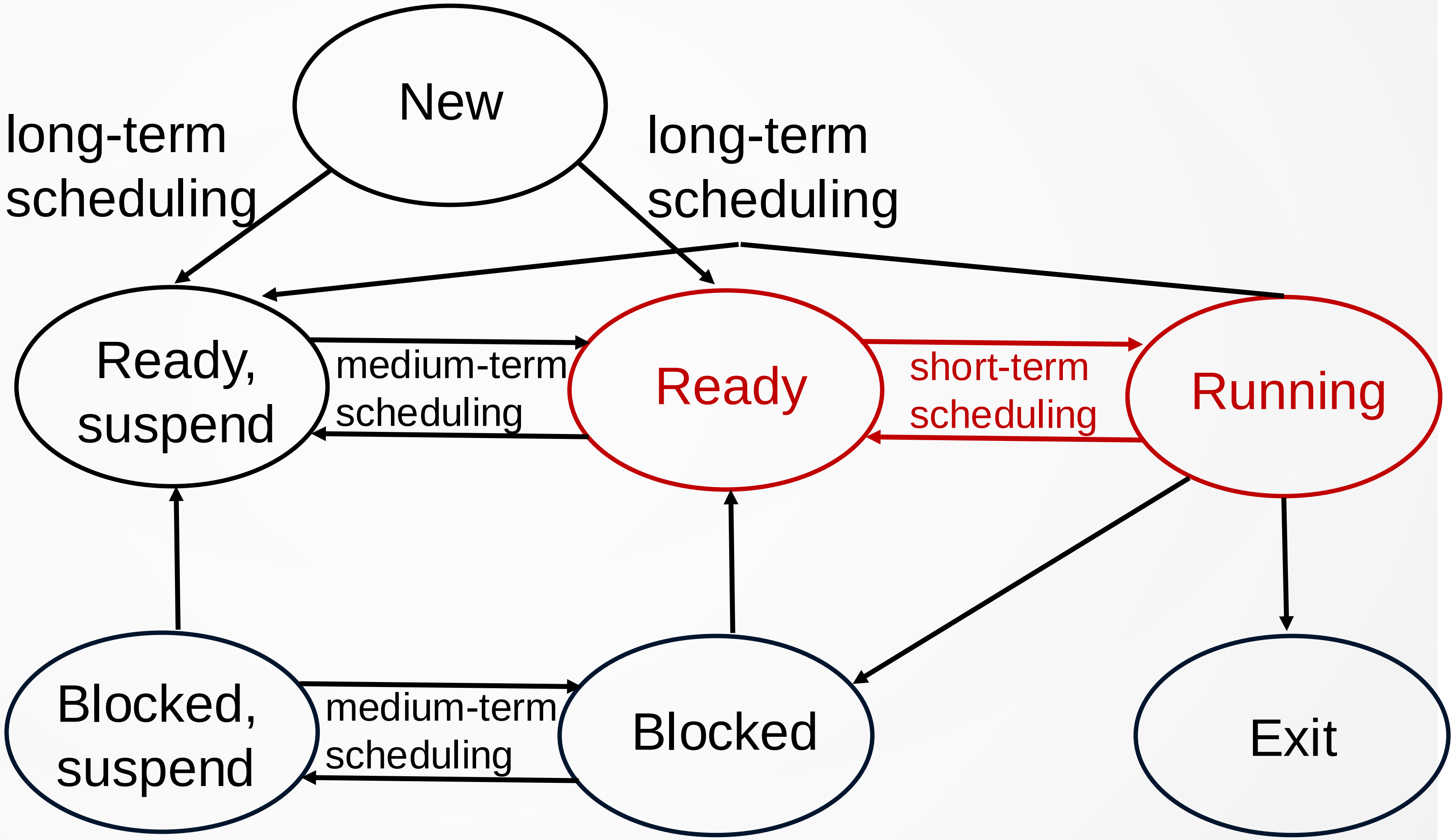
MEDIUM-TERM SCHEDULING

- Decide which processes to bring to memory
- Part of **swapping** function (discussed later with virtual memory)



SHORT-TERM SCHEDULING

- Decide which process to execute after an event: clock interrupt, IO interrupt, OS call, other signal
- Most frequently happening scheduling, **our focus today**



OBJECTIVES FOR SCHEDULING

- What is good service? What kind of criteria to set for scheduling?
- Think about different systems:
    - **Interactive systems** (e.g. personal computer)
    - **Batch systems** (e.g. computing server)
    - **Real-time systems** (e.g. self-driving car, process control systems, …)

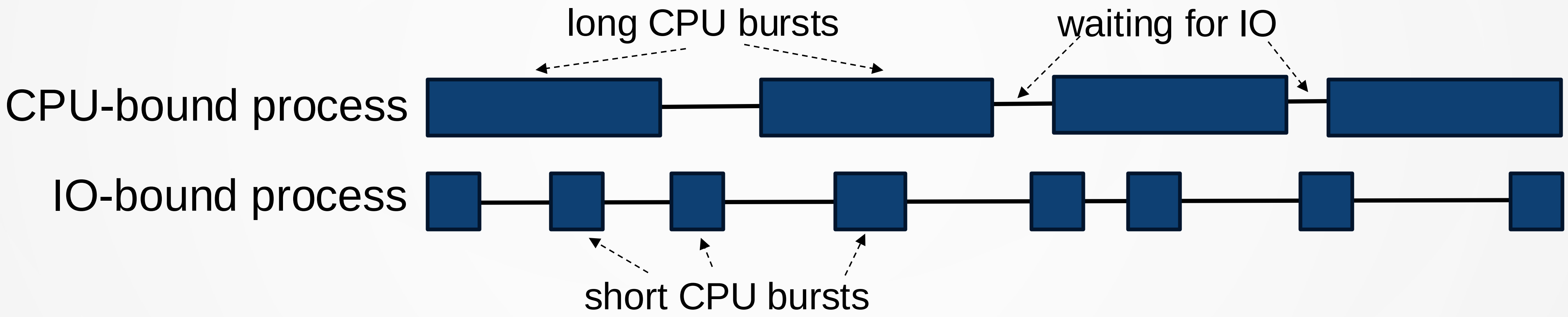
OBJECTIVES FOR SCHEDULING

	System	User
General	• Fairness: no starvation • Enforce priorities • Balance resources	• Predictability
Performance	• Throughput • Processor utilization	• Turnaround time $= T_{\text{complete}} - T_{\text{arrival}}$ • Response time $= T_{\text{first response}} - T_{\text{arrival}}$ • Deadlines

- Further consideration: **Preemptive** or not?
  - Preemption means more process switches but better response

CPU-BOUND AND IO-BOUND PROCESSES

- Processes alternate between executing on CPU and waiting for IO



- For now we assume:
  - A uniprocessor system (generalized to multiprocessor later)
  - In examples the processes consist of a single CPU-burst only (no IO)

SCHEDULING ALGORITHMS: FIRST COME FIRST SERVED (FCFS)

- **Process that first arrived in the ready-queue is scheduled first**

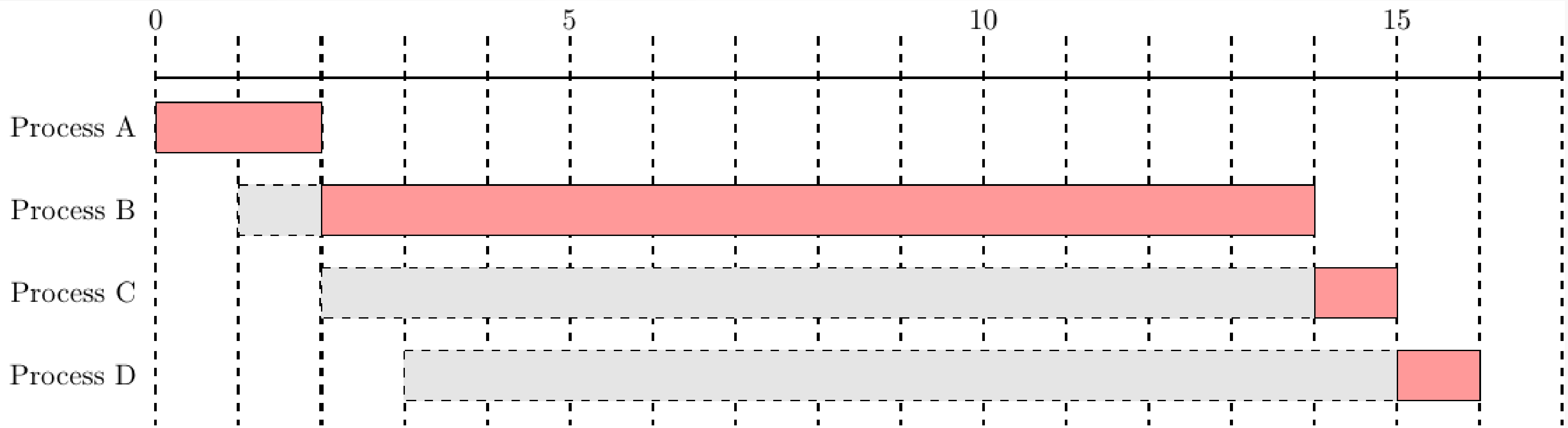
PROS	CONS
• Easy to understand and implement • Non-preemptive • Small overhead • No starvation	• Response times can be high • Penalizes short processes and IO-bound processes

FCFS: EXAMPLE

Process	Arrival	Runtime
A	0	2
B	1	12
C	2	1
D	3	1

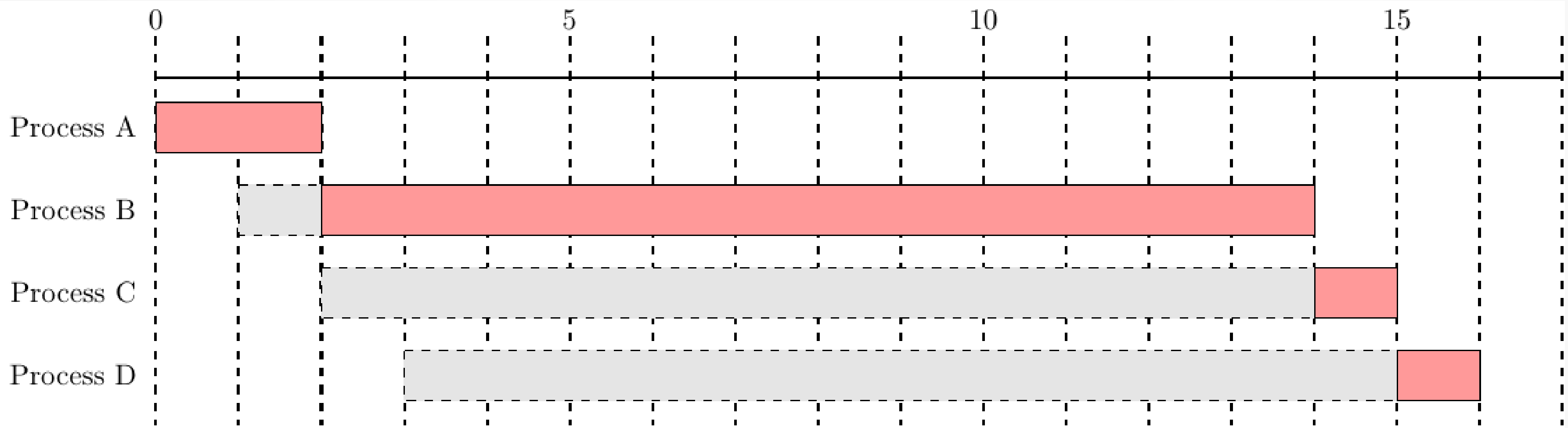
FCFS: EXAMPLE

Process	Arrival	Runtime
A	0	2
B	1	12
C	2	1
D	3	1



FCFS: EXAMPLE

Process	Arrival	Runtime
A	0	2
B	1	12
C	2	1
D	3	1



- **Average turnaround time** of processes A, B, C, D: ((2-0)+(14-1)+(15-2)+(16-3))/4=10.25
- **Average response time** of processes A, B, C, D: ((0-0)+(2-1)+(14-2)+(15-3))/4=6.25

SCHEDULING ALGORITHMS: SHORTEST JOB FIRST (SJF)

- **Process with shortest time to run is scheduled to run next**

PROS	CONS

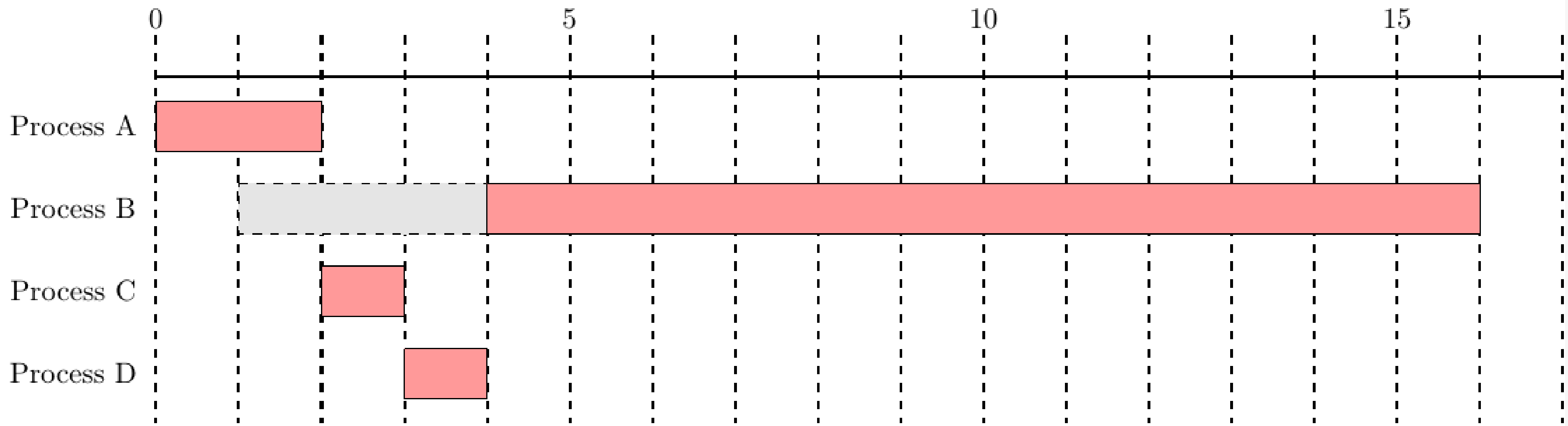
• Optimal turnaround time for processes arriving at the same time
• Typically good response time for short processes
	• Requires knowledge/estimate of runtime
	    • Estimate based on history
	    • User provides
	• Penalizes long processes
	• Starvation possible

SJF: EXAMPLE

Process	Arrival	Runtime
A	0	2
B	1	12
C	2	1
D	3	1

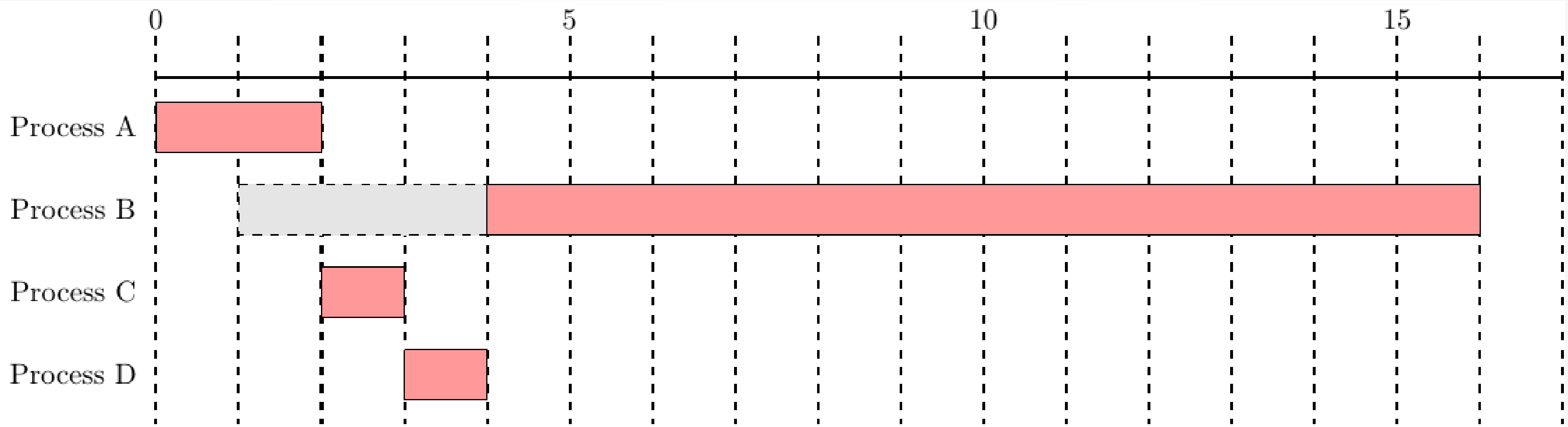
SJF: EXAMPLE

Process	Arrival	Runtime
A	0	2
B	1	12
C	2	1
D	3	1



SJF: EXAMPLE

Process	Arrival	Runtime
A	0	2
B	1	12
C	2	1
D	3	1



- **Average turnaround time** of processes A, B, C, D:
  ((2-0)+(16-1)+(3-2)+(4-3))/4=4.75
- **Average response time** of processes A, B, C, D:
  ((0-0)+(4-1)+(2-2)+(3-3))/4=0.75

SJF: EXAMPLE WITH STARVATION

- When does B get to run if processes with 1ms running time keep arriving every 1ms?

Process	Arrival	Runtime
A	0	2
B	1	12
C	2	1
D	3	1
E	4	1
F	5	1
...	...	...

SCHEDULING ALGORITHMS: ROUND ROBIN (RR)

- **Every process gets to run for a quantum (time slice), then switch to next process (using FCFS)**

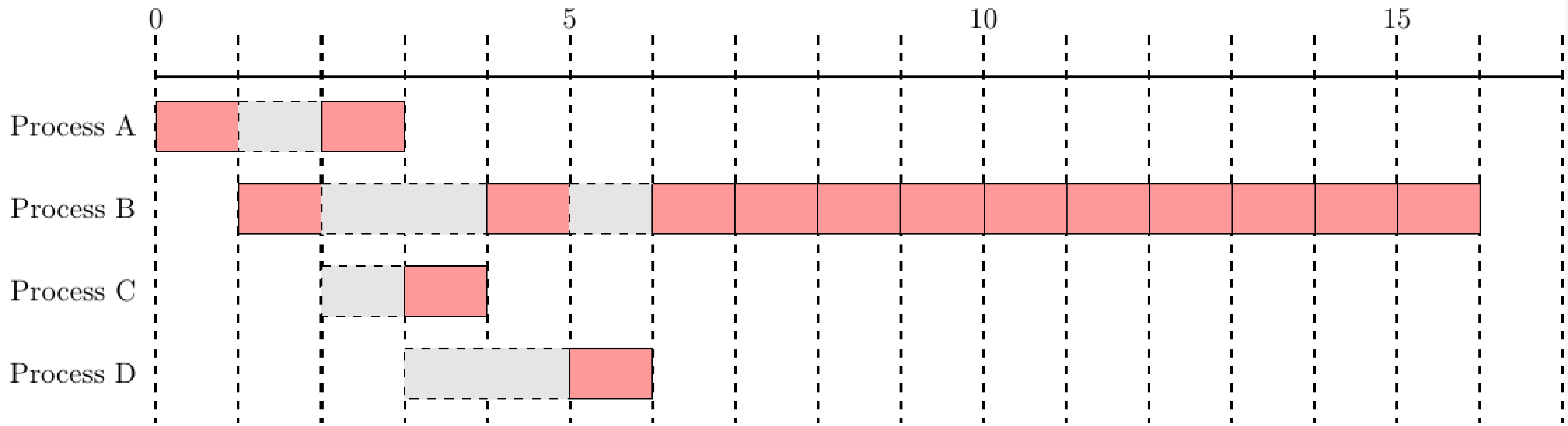
PROS	CONS
• Simple, easy to implement • Equal share of CPU • No starvation • Typically good response time	• Preemption induces overhead • Turnaround time can be long

RR: EXAMPLE (QUANTUM=1MS)

Process	Arrival	Runtime
A	0	2
B	1	12
C	2	1
D	3	1

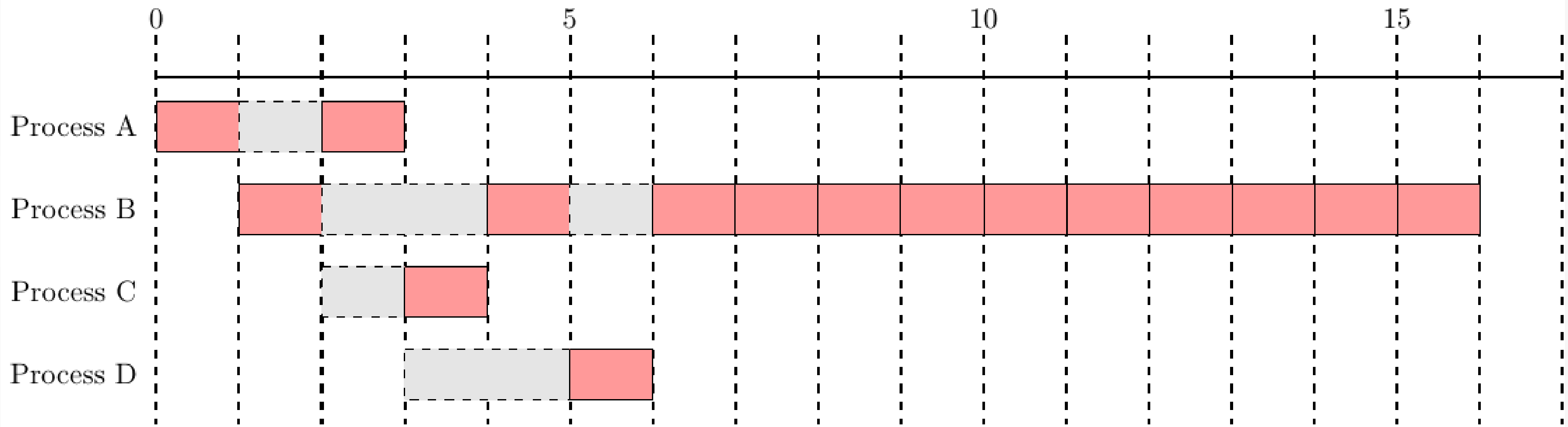
RR: EXAMPLE (QUANTUM=1MS)

Process	Arrival	Runtime
A	0	2
B	1	12
C	2	1
D	3	1



RR: EXAMPLE (QUANTUM=1MS)

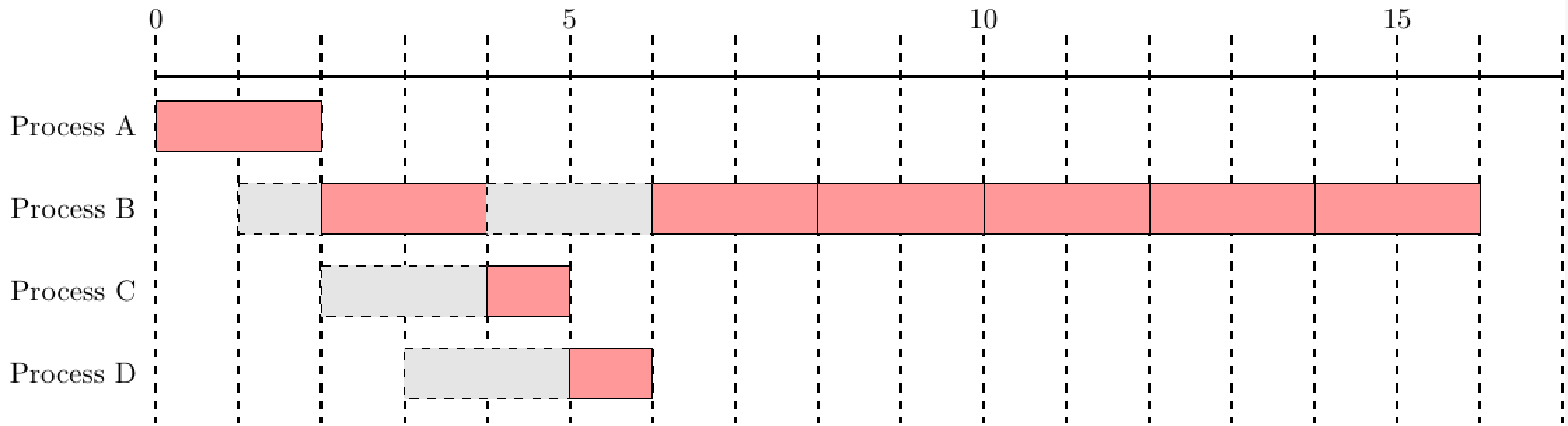
Process	Arrival	Runtime
A	0	2
B	1	12
C	2	1
D	3	1



- **Average turnaround time** of processes A, B, C, D: ((3-0)+(16-1)+(4-2)+(6-3))/4=5.75
- **Average response time** of processes A, B, C, D: ((0-0)+(1-1)+(3-2)+(5-3))/4=0.75
- What happens to turnaround time and response time if quantum=2ms?

RR: QUANTUM=2MS

Process	Arrival	Runtime
A	0	2
B	1	12
C	2	1
D	3	1



- **Average turnaround time** of processes A, B, C, D: ((2-0)+(16-1)+(5-2)+(6-3))/4=5.75
- **Average response time** of processes A, B, C, D: ((0-0)+(2-1)+(4-2)+(5-3))/4=1.25

ROUND ROBIN: QUANTUM LENGTH

- Quantum length is key to performance
  - Multiple of clock-interrupt period
  - Overhead from context switches
- Example: Assume context switch takes 1ms
  - If quantum is 4ms, overhead is 20%
  - If quantum is 100ms, overhead is appr. 1%
- What is the effect of too long quantum?

SUMMARY

- Scheduling of processes is an important part of an OS
- Different objectives for scheduling depending on a system
- Uniprocessor scheduling algorithms: FCFS, SJF, RR