OPERATING SYSTEMS:

Lesson 4:
Process Scheduling

Jesús Carretero Pérez
David Expósito Singh
José Daniel García Sánchez
Francisco Javier García Blas
Florin Isaila

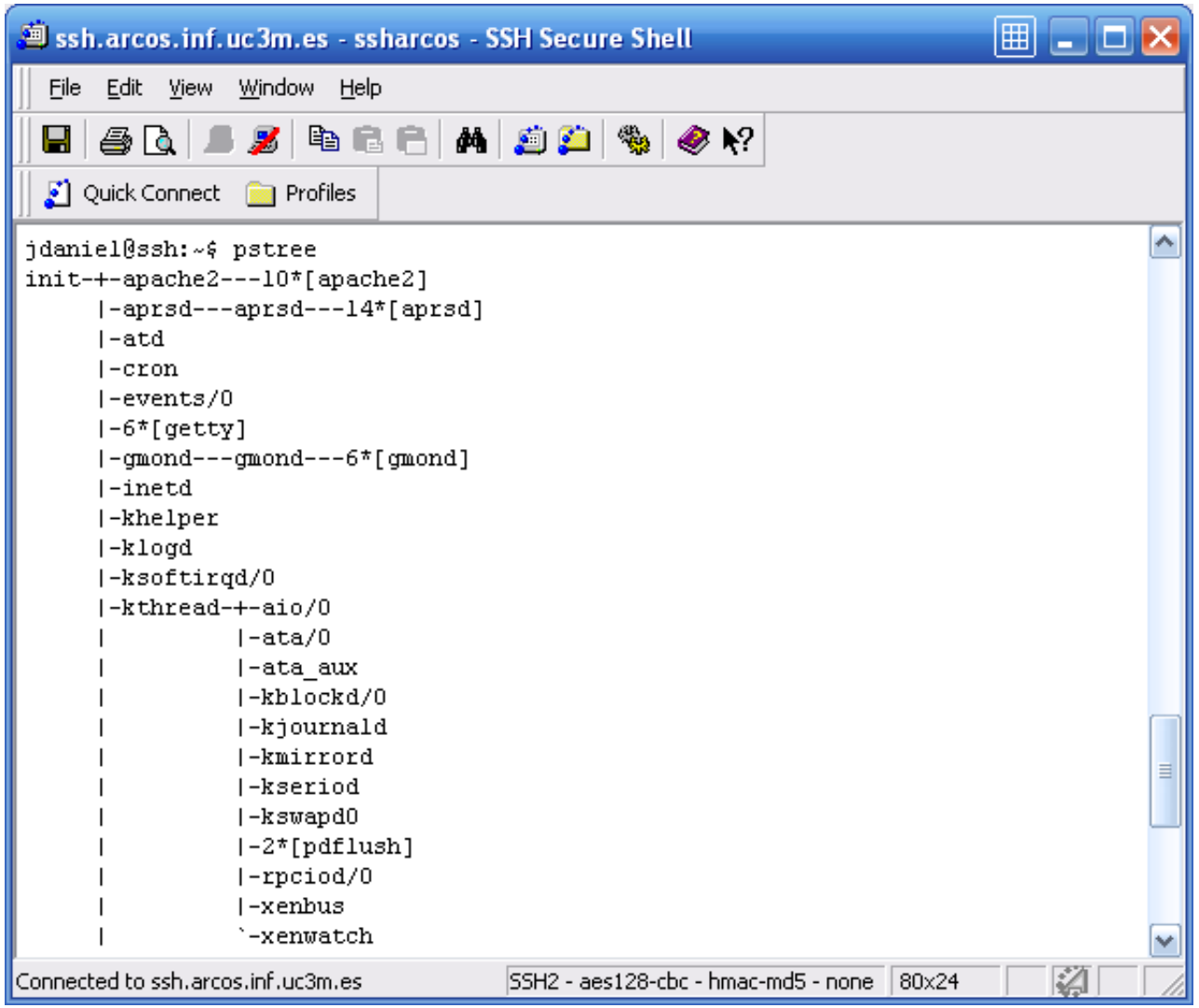
Content

- **Process creation.**
- Process termination.
- Process lifecycle.
- Kinds of scheduling.
- Scheduling algorithms.

Process creation

- OS provides mechanism to allow a process to create other processes → **System call**.
- Process creation can be repeated recursively leading to a *"family structure"* → Process tree.
- Resource allocation for new process:
  - Directly obtained from the OS.
  - Parent must give out its resources to the child process or share part of them with it.
    - To avoid a system hang by indefinitely replicating the process resources.

Process hierarchy (pstree)

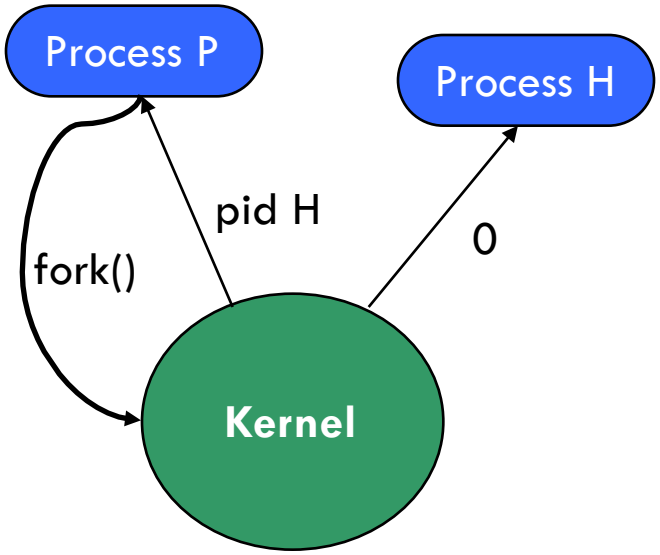


Process creation

- When a process is created:
  - In terms of **execution**:
    - Parent process runs in parallel with children.
    - Parent process waits until some or all of its children have terminated.
  - In terms of **memory space**:
    - Child process is a clone of the parent process.
    - Child process is converted into another program loaded in memory.

UNIX process creation

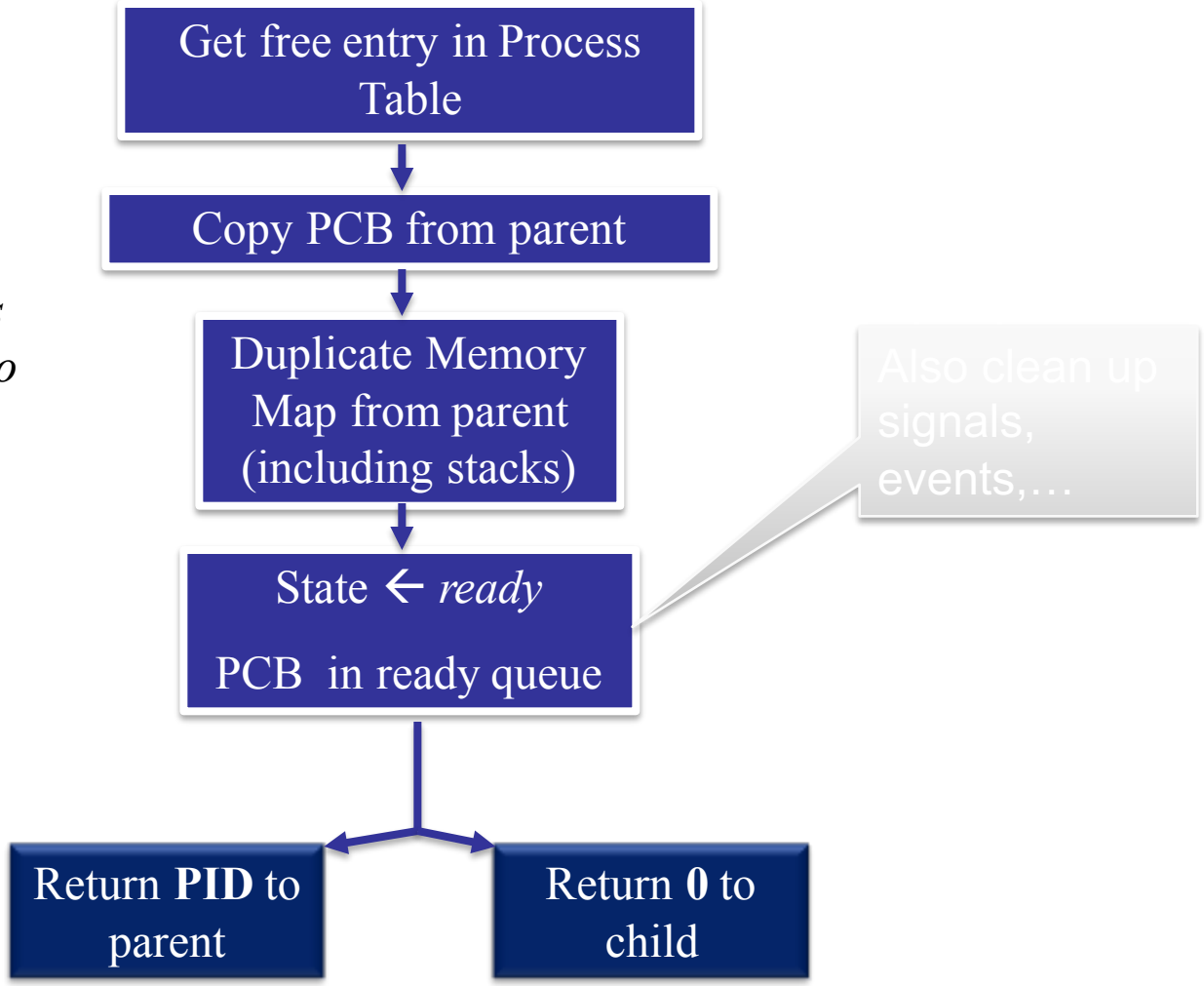
- Unix family makes distinction between process creation and new program execution.
- System call to create a new process is ***fork()***
- This system call creates a copy almost identical of the parent process.
  - Both processes, parent and child, continue execution in parallel.
  - Parent gets as a result from the *fork()* call the child PID and child gets a 0.
  - Some resources are not inherited (e.g.: pending signals).



Linux process creation

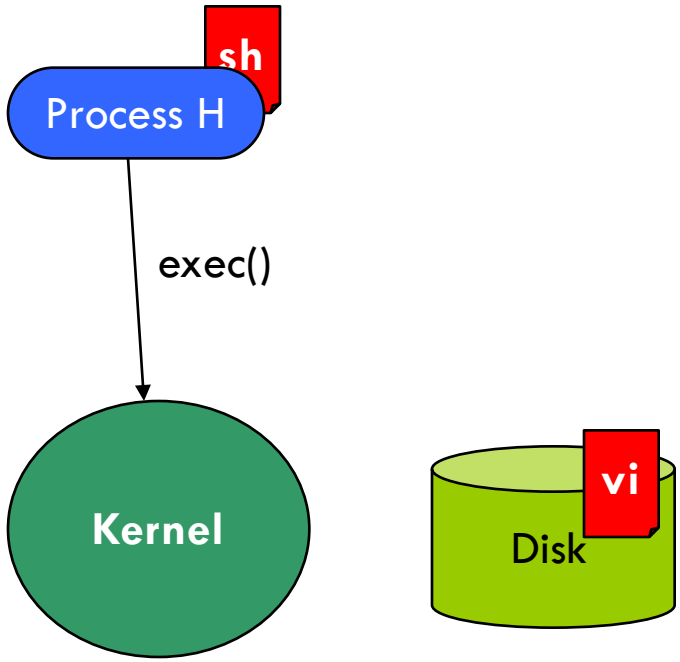
`fork:`

"Copies parent process and gives new identity to child"



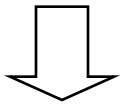
UNIX process creation

- Child process may invoke the **exec*()** system call.
  - Changes its memory image with a different program.
- Parent may continue creating more children, or waiting until the created child finishes.
  - **wait()** takes the process from the *"Ready"* queue until the child has terminated.

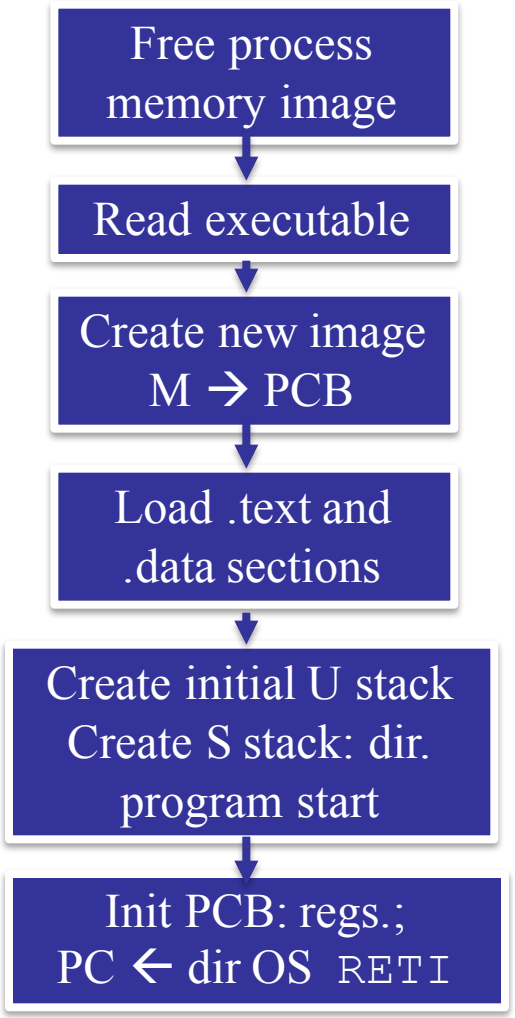


Process creation in Linux

`exec:`

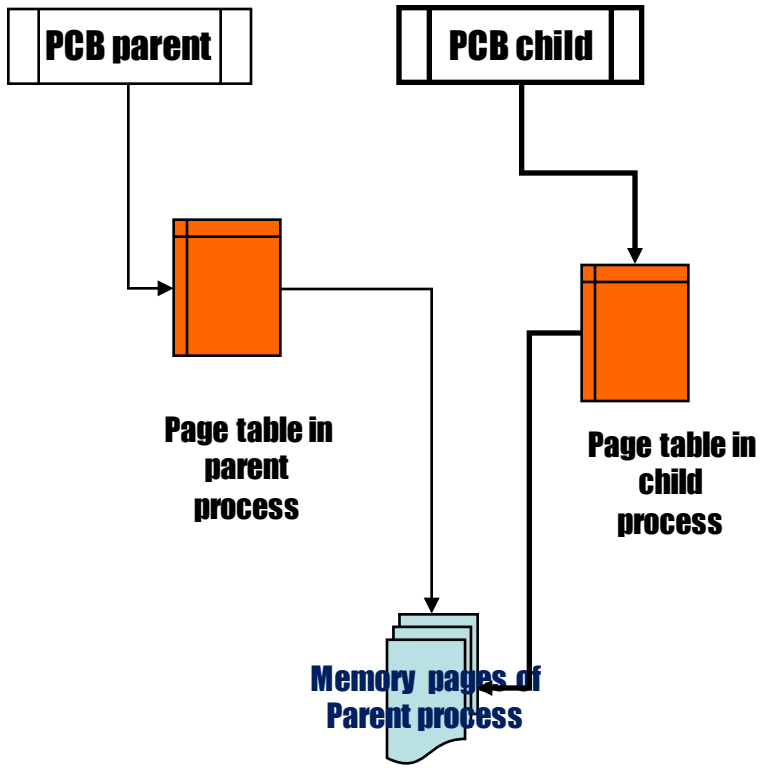


"Change memory image from a process using as a new container a new one"



Copy on Write (COW)

- Inefficiencies of the **fork()** model:
  - A lot of data is copied, but data could be shared instead.
  - If finally another memory image is loaded, is even worse as all copies are discarded
- Many UNIX systems use COW:
  - *Copy-on-Write* is a technique to delay or avoid copying data when performing a fork.
  - Data pages are shared and marked as COW so that if a modification is applied, then a copy is done the process (parent and child).
  - Now fork() only copies the page table from parent (but not the pages) and creates a new PCB for child.



Sharing go avoid duplication

Content

- Process creation.
- **Process termination.**
- Process lifecycle.
- Kinds of scheduling.
- Scheduling algorithms.

Process termination

- When a process frees all its allocated resources.
  - Memory, open files, entries in tables, …
- and the kernel notifies that to parent process.

- A process may terminate in 2 ways:
  - Voluntarily: **exit()** system call.
  - Involuntarily:
    - Exceptions: divide by zero, segment violation, …
    - Aborted by user (ctrl-c) or other process (kill)
      - » i.e.: signals that cannot be handled or ignored.

Process termination

- When a process terminates two outcomes are possible:
  - Its children are not affected.
  - All children also terminate → **cascade termination** (e.g. VMS operating system)

- In Unix,
  - Terminated child processes are now children of the **init** process.
  - Terminated process changes to ***zombie*** state until parent process gets its termination code.

When is PCB eliminated?

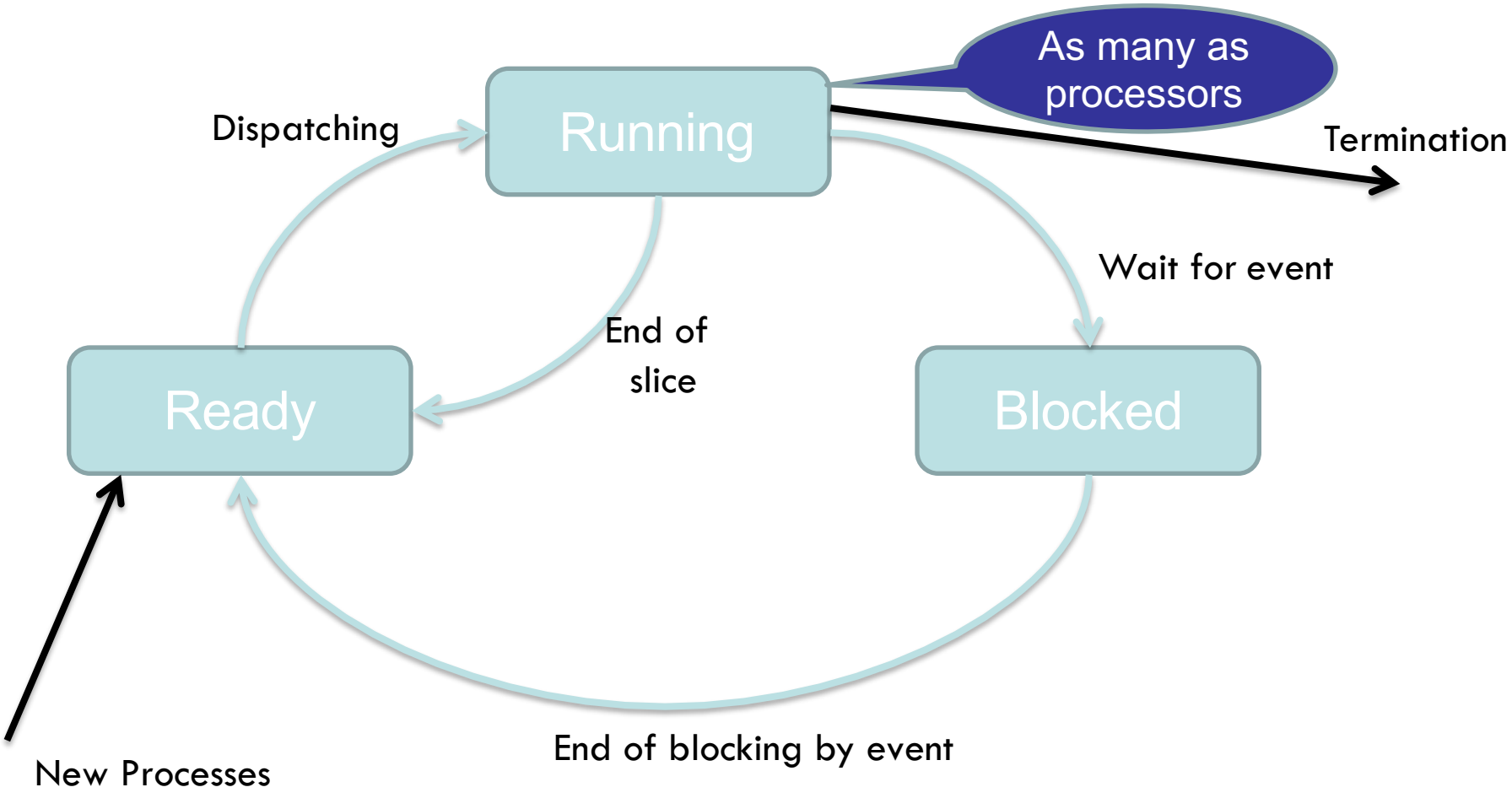
- Process termination and PCB elimination are two different tasks:
  - When parent gets information from child, data structures can be removed.
  - **wait()** system call:
    - Blocks process until a child terminates.
    - Returns **PID** of the terminated child.

Code1.c

Content

- Process creation.
- Process termination.
- **Process lifecycle.**
- Kinds of scheduling.
- Scheduling algorithms.

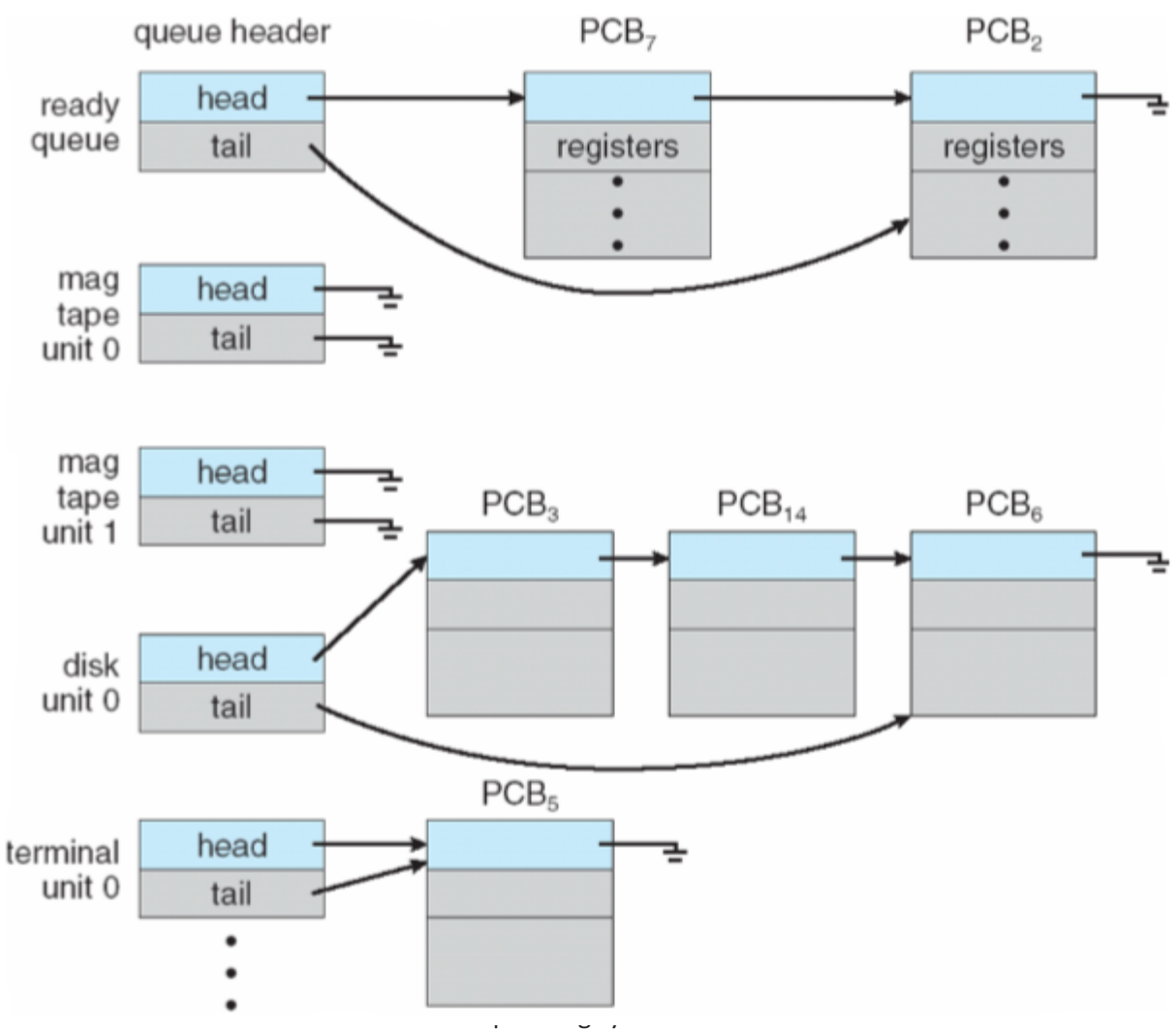
Process basic lifecycle



Process Control Block (PCB)

- Information associated with each process
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information
- PCBs are stored in a global **process table**

Ready Queue And Various I/O Device Queues

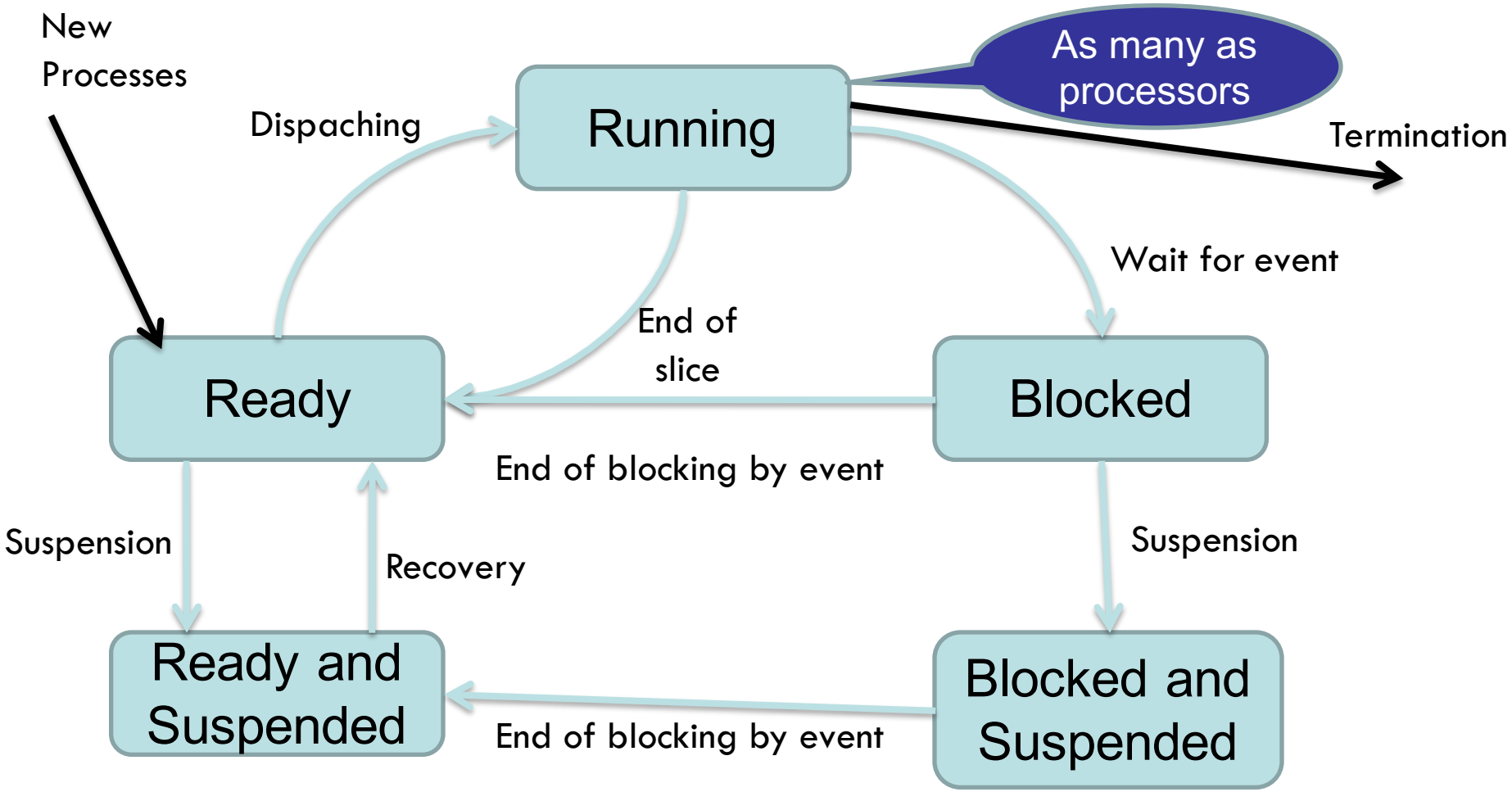


Spawning to disk (swap)

- When there are many process in execution, performance may degrade due to excessive memory paging.
  - Solution: Operating System may need to send a process to the swap area in disk.

- New process states.
  - Blocked and suspended.
  - Ready and suspended.

Process lifecycle



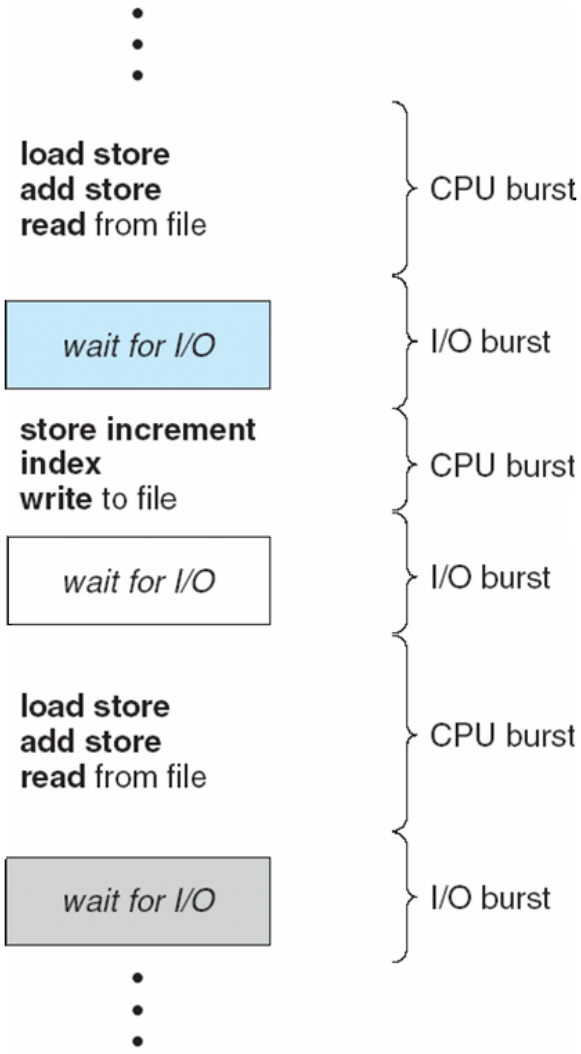
Content

- Process creation.
- Process termination.
- Process lifecycle.
- **Kinds of scheduling.**
- Scheduling algorithms.

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
  - Having when possible at any time the CPU used by a process
- CPU–I/O Burst Cycle
  - Process execution consists of a *cycle* of CPU execution and I/O wait
- CPU burst distribution

Alternating Sequence of CPU And I/O Bursts



Scheduling levels

- Short term scheduling:
  - Selects next process to execute.
- Medium term scheduling:
  - Select process to be added or retired (suspended to swap) from main memory.
- Long term scheduling:
  - Perform admission control of processes.
  - Very used in batch systems.

Kinds of scheduling

- Non-preemptive.
  - Process in execution keeps running on CPU while it is able to use the CPU.
- Preemptive:
  - Operating System may acquire the CPU, evict the process and change the execution to another process.

Scheduling decision points

- Points in time when OS may perform process scheduling:
  - When a process blocks waiting for an event.
    - Perform a system call.
    - I/O request
    - Wait() invocation
  - When an interrupt happens.
    - Clock interrupt.
    - I/O interrupt.
  - When a process switches from waiting state to ready state
    - I/O completion
  - When a process ends.
- Nonpreemptive scheduling: 1 and 4.
  - Windows95, MacOS before 8.
- Preemptive scheduling: 1, 2, 3, and 4.

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- Dispatch latency – time it takes for the dispatcher to stop one process and start another running

Process queues

- Ready processes are kept in a queue.

- Alternatives:
  - Single queue.
  - Queues by types of processes.
  - Priority queues.

Content

- Process creation.
- Process termination.
- Process lifecycle.
- Kinds of scheduling.
- **Scheduling algorithms.**

Scheduling: metrics

- CPU utilization:
  - Percentage of time CPU is used.
  - Goal: Maximize.
- Throughput:
  - Number of jobs finished by unit of time.
  - Goal: Maximize.
- Turnaround time (T_q)
  - Overall time a process is in system.
  - $T_q = T_f - T_i$
    - T_f: Finalization time.
    - T_i: Initiation time.
  - Goal: Minimize.

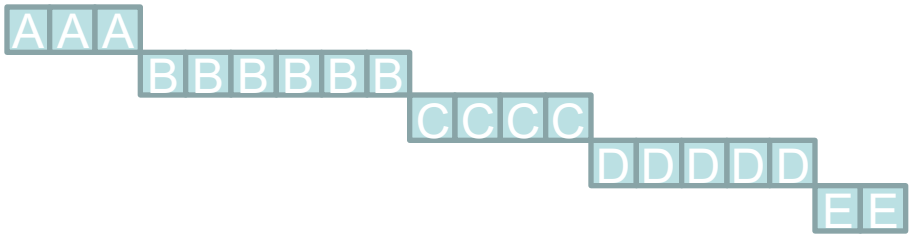
Scheduling: Metrics

- Service time(T_s):
  - Time devoted to productive tasks(cpu, I/O).
  - $T_s = T_{cpu} + T_{I/O}$
- Waiting time (T_w):
  - Time a process spends in waiting queues.
  - $T_w = T_q - T_s$
- Normalized Turnaround time (T_n):
  - Ratio between Turnaround time and service time.
  - $T_n = T_q / T_s$
  - Indication of experienced delay.

FCFS

- ***First to Come First to Serve*.**
  - Non-preemptive.
  - Penalizes short processes: *Convoy effect* short process behind long process

Process	Arrival	Service
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



FCFS: Normalized Turnaround Time

Process	Arrival	Service	Init	End	Turnaround	Wait	Normalized Turnaround
A	0	3	0	3	3	0	3/3=1
B	2	6	3	9	7	1	7/6=1.16
C	4	4	9	13	9	5	9/4=1.25
D	6	5	13	18	12	7	12/5=2.4
E	8	2	18	20	12	10	12/2=6

- Average Turnaround time: **4.6**
- Average normalized Turnaround time : **2.5**

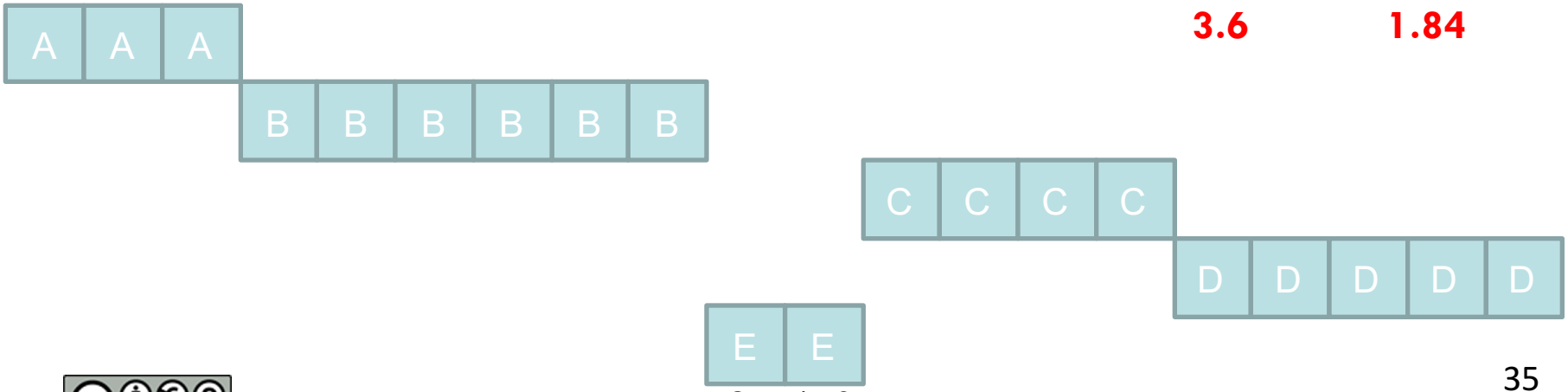
SJF

- ***Shortest Job First*.**
- Non-preemptive algorithm.
- Selects shortest job.
- It can only be applied if duration of each job is known beforehand.
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
- Starvation possibility:
  - If short jobs are continuously arriving, longer jobs never are in position to be executed.

SJF

Process	Arrival	Service	Init	End	Turnaround	Wait	Normalized Turnaround
A	0	3	0	3	3	0	3/3=1
B	2	6	3	9	7	1	7/6=1.16
C	4	4	11	15	11	7	11/4=2.75
D	6	5	15	20	14	9	14/5=2.8
E	8	2	9	11	3	1	3/2=1.5

3.6 **1.84**

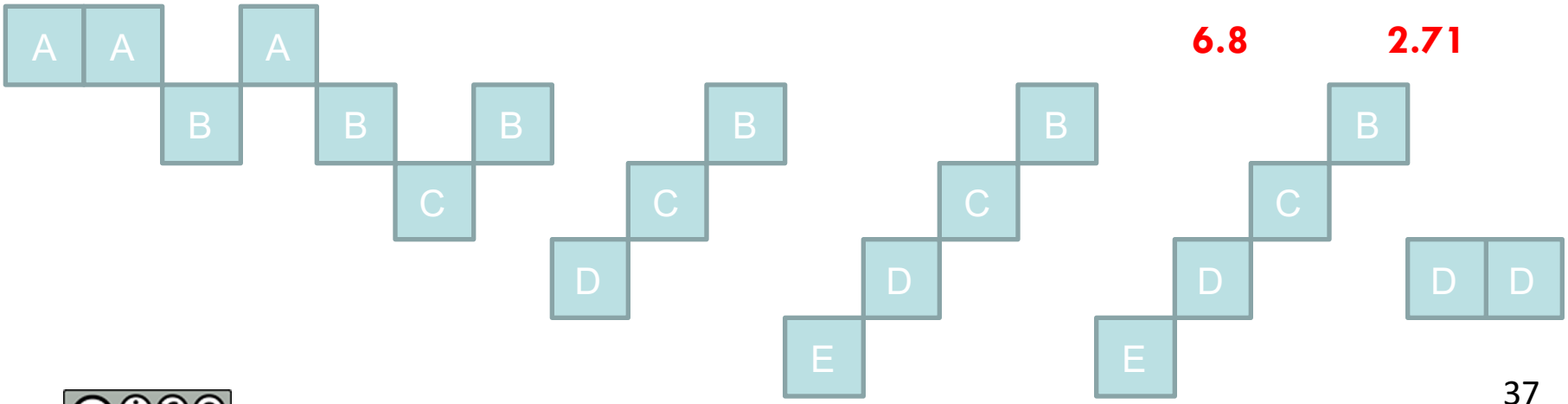


Cyclic or Round-Robin

- Keeps a FIFO queue with processes **ready** to run.
- A process is allocated into a processor for a **time slice**.
  - 10-100 milliseconds
- A process goes back to the **ready** queue when:
  - Its time slice expires.
  - An event that took it to the blocked queue happens.
- A process goes to the **blocked** queue when:
  - Starts waiting for an event.
- It is a preemptive algorithm.
- It is important to remind that every context switch leads to a delay:
  - Time slice >> Context switch time
- Performance
  - q large ⇒ FIFO
  - q small ⇒ q must be large with respect to context switch, otherwise overhead is too high

Round-Robin (q=1)

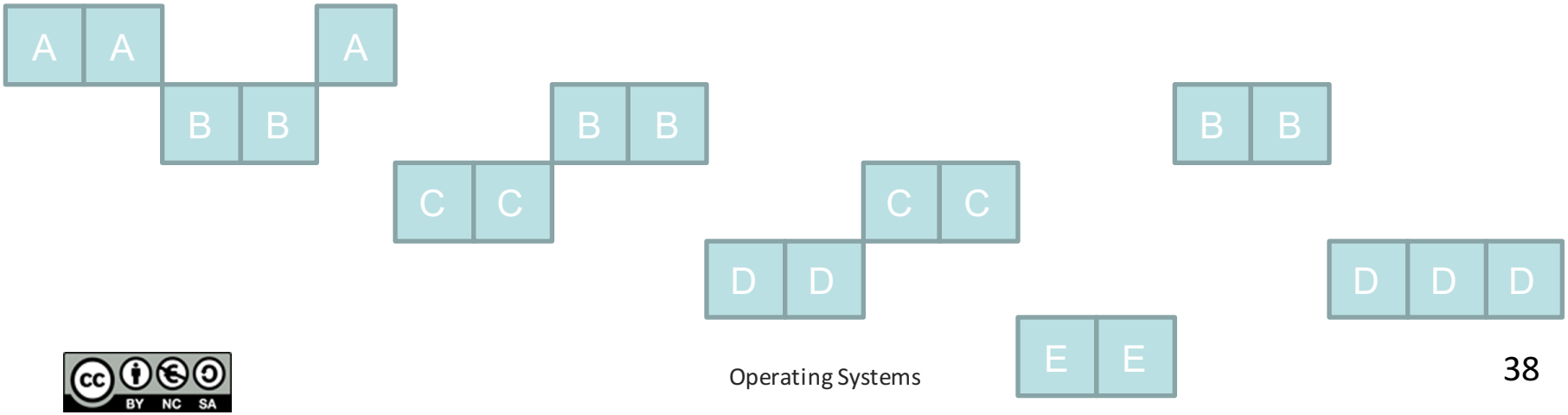
Process	Arrival	Service	Init	End	Turnaround	Wait	Normalized Turnaround
A	0	3	0	4	4	1	4/3=1.33
B	2	6	2	18	16	10	16/6=2.66
C	4	4	5	17	13	9	13/4=3.25
D	6	5	7	20	14	9	14/5=2.8
E	8	2	10	15	7	5	7/2=3.5



Round-Robin (q=2)

Process	Arrival	Service	Init	End	Turnaround	Wait	Normalized Turnaround
A	0	3	0	5	4	1	4/3=1.33
B	2	6	2	17	16	10	16/6=2.66
C	4	4	5	13	13	9	13/4=3.25
D	6	5	9	20	14	9	14/5=2.8
E	8	2	13	15	7	5	7/2=3.5

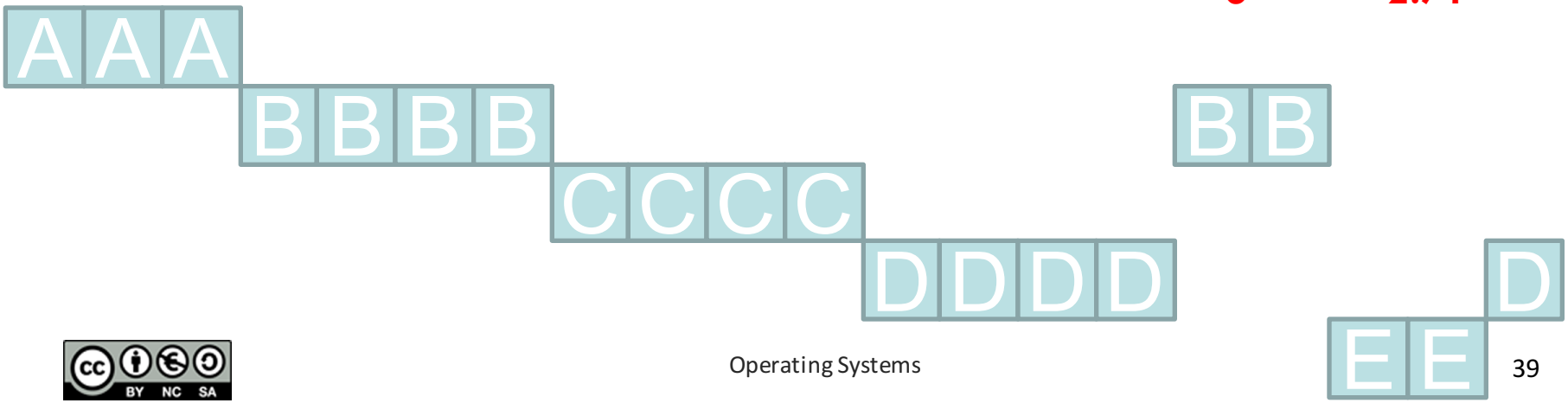
6 **2.54**



Round-Robin (q=4)

Process	Arrival	Service	Init	End	Turnaround	Wait	Normalized Turnaround
A	0	3	0	3	3	0	3/3=1
B	2	6	3	17	15	9	15/6=2.5
C	4	4	7	11	7	3	7/4=1.75
D	6	5	11	20	14	9	14/5=2.8
E	8	2	17	19	11	9	11/2=5.5

6 **2.71**



Priority Scheduling

- Each process has a priority assigned to it.
- Select first processes with higher priority.
  - Preemptive
  - nonpreemptive
- Alternatives:
  - Fixed priorities → starvation problem – low priority processes may never execute.
  - **Solution**: aging mechanisms – as time progresses increase the priority of the process

Scheduling in Windows

- Main characteristics:
  - Priority based
  - Uses time slices.
  - Preemptive scheduling.
  - Scheduling with processor affinity.
- Scheduling at thread level (not process level).
- A thread may loose the processor if another with higher priority becomes ready.
- Scheduling decisions:
  - New threads → Ready.
  - Blocked threads receiving its event → Ready.
  - Thread leaves processor if time slice expires, it terminates or becomes blocked.

Summary

- Process creation implies memory image and PCB creation.
- A process transitions through different states during its execution.
- Operating system is responsible for process scheduling.
- Scheduling may be preemptive or non-preemptive.
- Different process scheduling algorithms may favor a certain type of processes.
- Modern Operating systems use preemptive scheduling.

OPERATING SYSTEMS:

Lesson 4:
Process Scheduling

Jesús Carretero Pérez
David Expósito Singh
José Daniel García Sánchez
Francisco Javier García Blas
Florin Isaila