



Scuola Superiore Sant'Anna



Scheduling

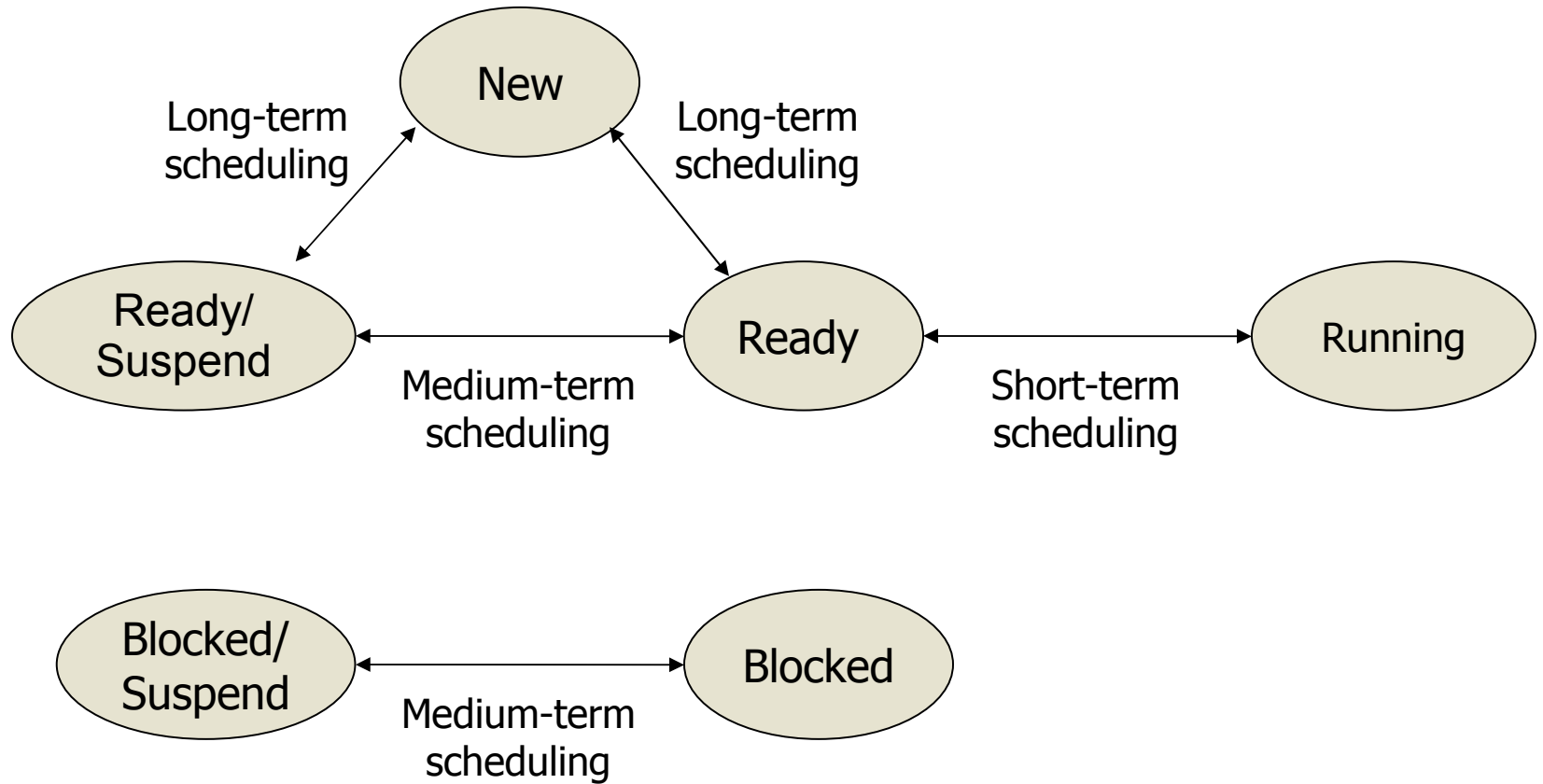
Giuseppe Lipari



Scheduling

- Scheduling is the activity of selecting which process/thread should be executed next
- We can distinguish 3 types of scheduling
 - Long term scheduling
 - Before creating a process, decides if it must be activated
 - Medium term scheduling
 - Decides if a process has to be swapped out/in
 - Short term scheduling
 - Decides which process has to be executed next

Scheduling





Long-term scheduling

- When a program is created, it goes through an admission test
 - If the program is admitted, the process is created and put in the ready queue
 - If the program is not admitted, it is put in the suspend state, until the multiprogramming level allows the process to start
 - The key decision is the multiprogramming level
 - If the multiprogramming level is too high, each process executes too slowly, and we risk the trashing
 - So we can “hold back” the new processes until the load is decreased



Medium-term scheduling

- It is the part of the OS that decides with processes to swap-in and out
 - It has to do with the multiprogramming level!
 - It has been already discussed in the previous part



Short-term scheduling

- We distinguish
 - **Selection function**: Decides which process is selected from the ready queue, according to some rule
 - **Decision mode**: when the decision is taken
 - Non-preemptive: once the process goes into running state, no scheduling decision can be taken until it finishes
 - Preemptive:
 - Periodically (like in round-robin)
 - When a process is unblocked
 - When an interrupt arrives



Scheduling criteria

- How to evaluate a scheduler?
 - User-oriented criteria
 - Response time of processes
 - System-oriented criteria
 - Throughput: how much work the system can do in any interval of time
- Performance is important
 - But not only!
 - In some case we may want some other criteria,
 - Predictability (see real-time systems)
 - Fairness



CPU-Bound and I/O bound

- CPU-bound processes perform very little I/O operations
 - Most of the time, they execute on the processor
 - For example,
 - the gcc compiler, a scientific computation program
- I/O bound processes perform many I/O operations
 - Most of the time, they are blocked waiting for an I/O device
 - For example,
 - The find and grep commands, the disk de-fragmenter
- Most processes are a mixture of the two

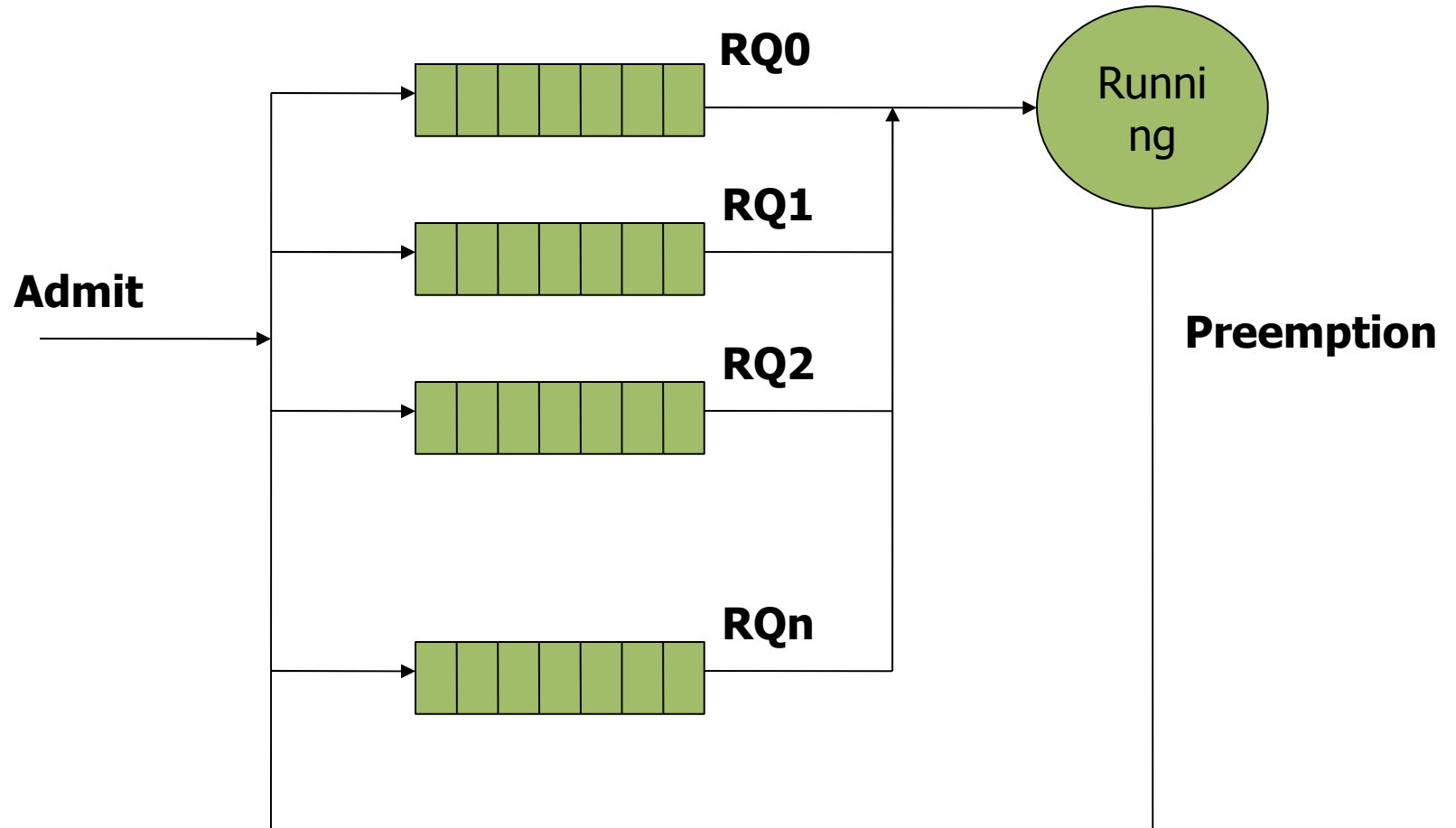


Priority based scheduling

- Each process is assigned a priority
- The highest priority process is selected to execute
 - If we are in a preemptive systems, when a high priority process is unblocked, it can preempt the executing process
 - Otherwise, it has to wait until the previous process finishes



Priority - based





Other scheduling policies

- FCFS
 - Ready queue is ordered by arrival time
- Round Robin (simple)
 - Periodically preempts executing process and put it back in the ready queue
 - The ready queue is ordered FIFO
- SPN (Shortest Process Next)
 - The shortest process executes first
 - Non preemptive
- SRT (Shortest Remaining Time)
 - Like SPN but preemptive
- HRRN (Highest Response Ratio Next)
 - The process with the highest response ratio executes first
- Feedback
 - Like Round Robin but with multiple queues



Measures

- A process is identified by
 - Arrival time
 - Service time s (time to be completed)
 - Waiting time w (time waiting to be serviced)
- Processor-bound process
 - It make little access to I/O
 - Intensive processor use
- I/O-bound processes
 - It makes a big amount of I/O operations
 - Intensive use of I/O devices, it is blocked most of the times
- Turn around time (TAT)
 - It is the total time that the process spends in the system, (service time + waiting time)
- Normalized TAT
 - NTAT = TAT/service time: $(w + s) / s$



FCFS

- This policy is not very good for short processes
- In fact, if a short process arrives just after a long process, it must wait for the long process to complete
 - Very high NTAT
- Process-bound processes have an advantage over I/O bound processes
- Example from the book



Round Robin

- Gives an advantage to short processes
- By “slicing” long processes, the response time of short process is reduced
- However, process switch is not for free!!
- Short quantum size imply high overhead
- Example from the book



Shortest process next

- Policy
 - Select the shortest process from the queue
 - It requires an estimation of the execution time of a process
 - it can be obtained by averaging over past history, in exponential way
 - It can be obtained from past statistics
 - Like RR, it gives advantage to small processes
 - It can lead to starvation!



Multiple-feedback queues

- This scheduler consists of
 - N queues, RQ_0, \dots, RQ_N
 - Each queue is ordered in FIFO order
 - A fixed time quantum Q (like in Round Robin)
- Rules
 - The scheduler chooses the first process in the highest priority queue and sets the timer equal to Q . Let RQ_k be the queue
 - If the process completes or blocks before the timer expires, select the next process (go to 1)
 - If the process has not yet completed by Q , move the process to the next queue $RQ_{(k+1)}$
 - When a process is activated or it is unblocked, it goes in the highest priority queue RQ_0
 - Periodically, processes can be “moved up” if they were not able to execute enough (to avoid starvation)



Multiple-feedback queue

- This scheduling policy tries to overcome the penalty for I/O bound processes
 - A process always starts from RQ_0
 - A short process, will probably complete very soon
 - An I/O-bound process is always executed with very short delay
 - A long CPU-bound eventually goes in the last queue RQ_N
 - If the system is heavily loaded, periodically the CPU-bound processes in the last queue are “promoted” to higher priority queues