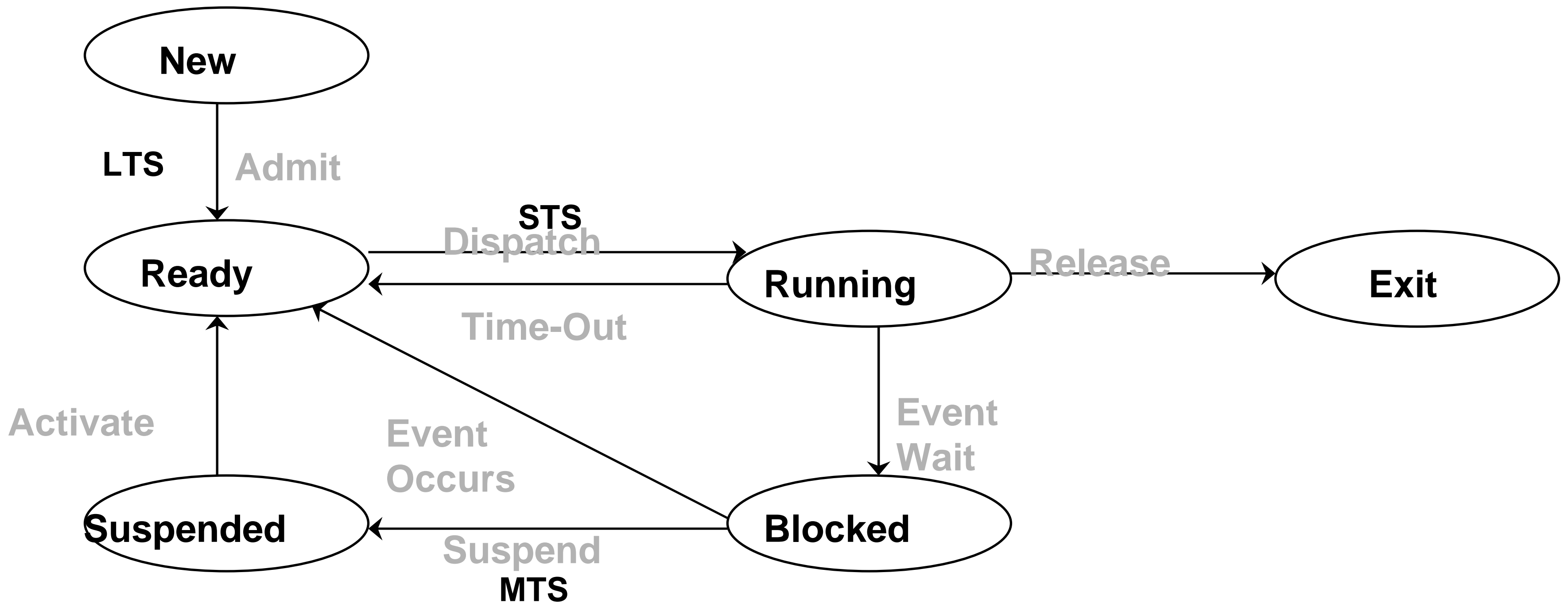
Motivation

- Scheduling defines the strategies used to allocate the processor.
  - Successful scheduling tries to meet particular objectives such as fast response time, high throughput and high process efficiency.

Types of Scheduling

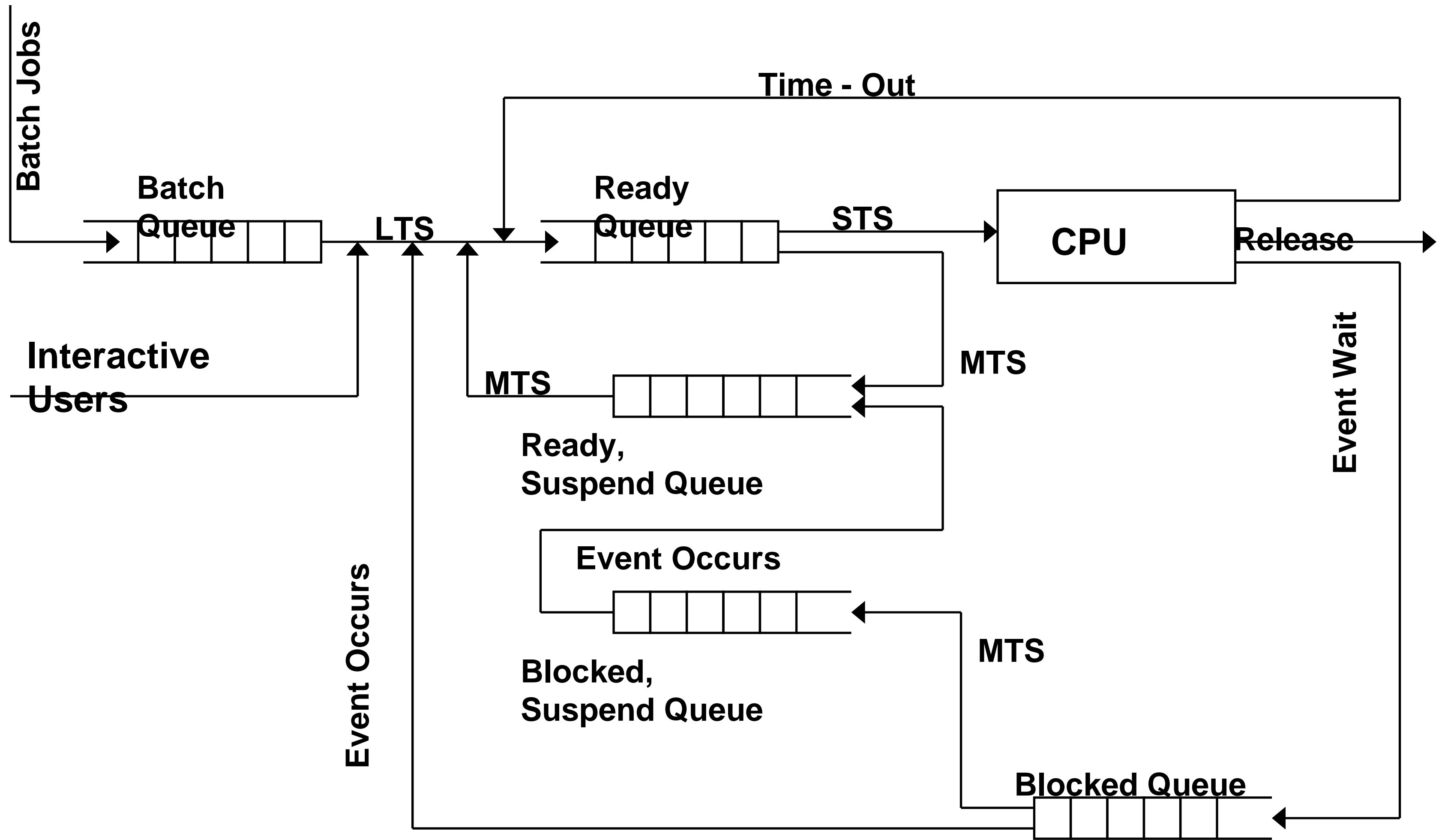
- Long-term scheduling
  - The long-term scheduler controls the degree of multiprogramming in the system.
  - It determines when a process is allowed to enter the system.
- Medium-term scheduling
  - The medium-term scheduler is used for process swapping.
- Short-term scheduling
  - The short-term scheduler of CPU scheduler selects a process from the ready queue and dispatches it.

Process State Transitions and Scheduling



LTS: Long Term Scheduling
MTS: Medium Term Scheduling
STS: Short Term Scheduling

Queuing Diagram for Scheduling



Scheduling Criteria

- User Oriented, Performance-Related Criteria:
  - Response time
    - In an interactive system, this is the time interval between submission of the request (hitting the enter key) and the reception of some response.
  - Turnaround time
    - For batch jobs, this is the time interval between submission of a process and its completion.
  - Deadlines
    - When process deadlines are specified, execution of processes should be prioritized to increase the likelihood that deadlines will be met.
      - Especially important for some critical real-time systems.

- User Oriented, Other Criteria:
  - Predictability
    - A job should run in about the same amount of time and at about the same cost independent of the load on the system.

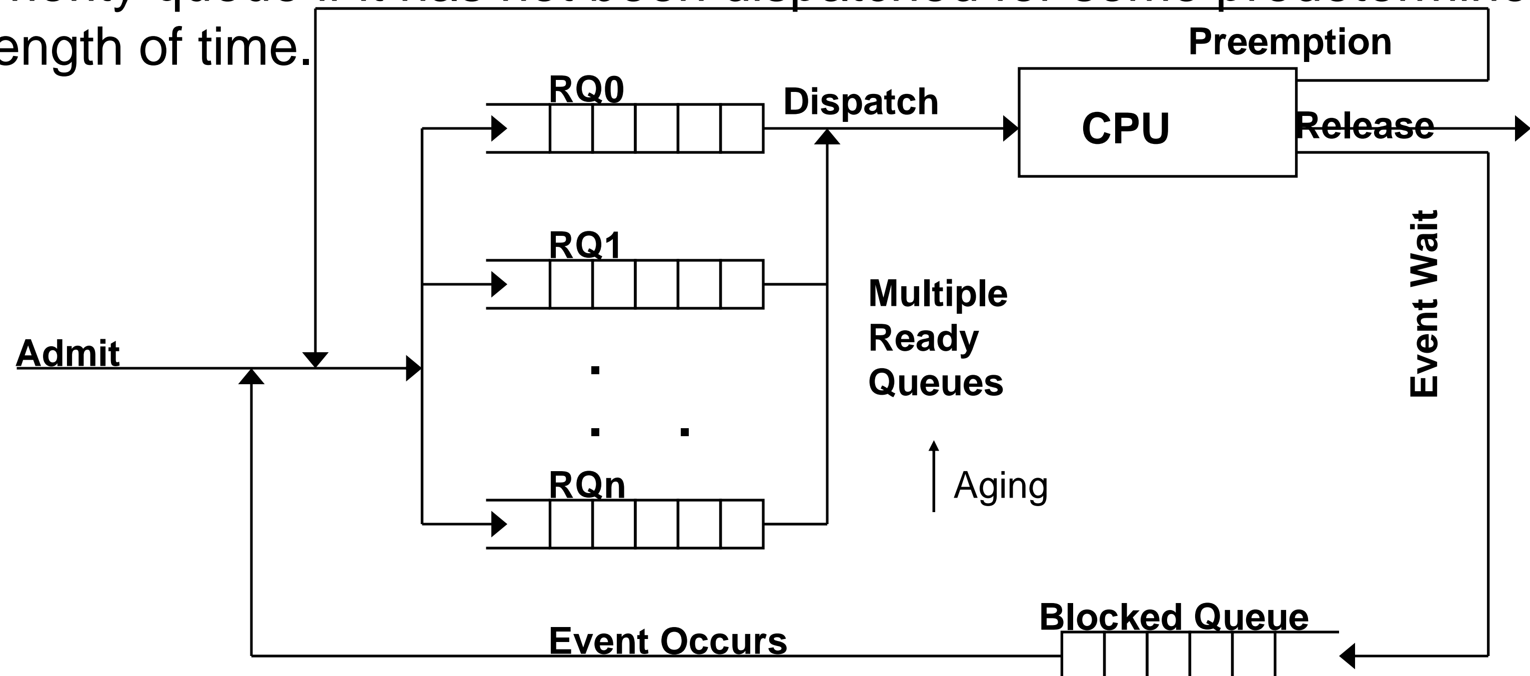
Scheduling Criteria (cont.)

- **System Oriented, Performance-Related Criteria:**
  - Throughput
    - Throughput is the number of processes completed per unit of time.
    - Since short response times may involve considerable context switching, high throughput may be somewhat compromised by short response times.
  - Processor utilization
    - This is the percentage of time that the processor is busy.

- **System Oriented, Other Criteria:**
  - Fairness
    - Unless there is a priority mechanism in place, the system should give processes equal opportunities to secure resources including the processor itself.
    - In particular, no process should suffer starvation.
  - Enforcing priorities
    - The scheduling policy should favour processes with a higher priority.
  - Balancing resources
    - The scheduling policy should keep the resources of the system busy.

Process Priorities

- Process priorities can be facilitated by using multiple ready queues.
  - The dispatcher will select processes from queue RQ_j only if queue RQ_i is empty (for all i<j).
  - To prevent starvation of processes in the lower priority queues, we can use an aging policy that allows a process to move up to a higher priority queue if it has not been dispatched for some predetermined length of time.



Priority Policies

- An OS may have a policy that decides on the priority (i.e. which ready queue) is to be used for both newly admitted processes and processes returning to the ready queue.

- This may depend on:
  - inherent priority of the process
  - predicted execution time
  - recent request for I/O
  - an aging policy.

Preemptive vs. Nonpreemptive

- Nonpreemptive
  - If the process is in a running state, it stays in that state until it terminates or blocks itself to wait for an I/O completion or some OS service.
    - (NO time slicing!)

- Preemptive
  - The currently running process may be interrupted and moved to the ready state by the OS.
    - Examples:
      - time slicing
      - SRT (to be described later)

  - We now consider various scheduling algorithms.

First-Come, First-Served Scheduling

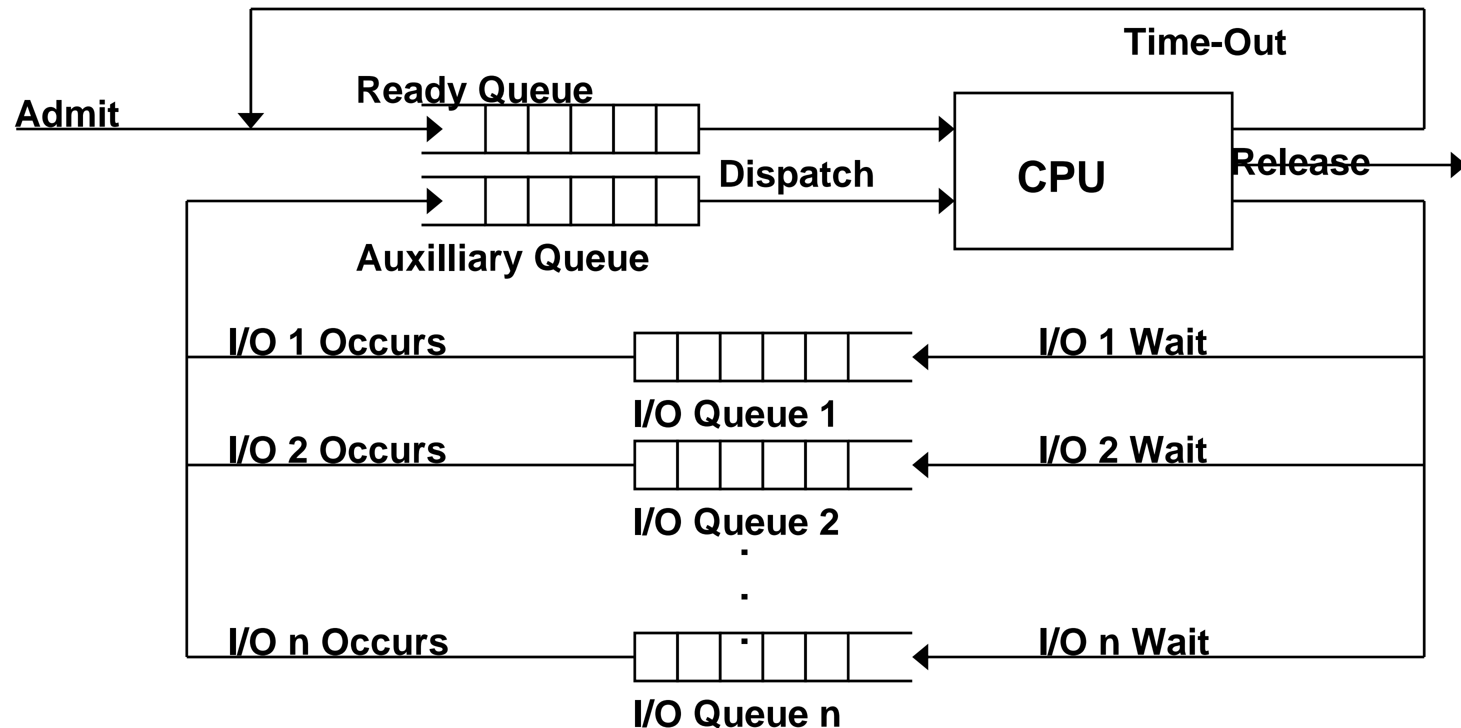
- Strategy:
  - As each process becomes ready, it joins the ready queue.
  - When the currently running process stops, the oldest process is next selected from the ready queue.
    - FCFS is nonpreemptive.
- Pro:
  - FCFS is simple and has the least overhead.
  - Process starvation cannot occur.
- Con:
  - FCFS tends to penalize short processes and I/O bound processes.
    - Since processes execute to completion CPU bound processes are favored over I/O bound processes.
  - Response times may be too long.
    - (bad for a multiprocessor environment)
  - Note: Despite the shortcomings, it is worthwhile to study FCFS as a starting point for other more sophisticated strategies.

Round-Robin Scheduling

- A Preemptive Strategy:
  - A clock interrupt is generated at regular intervals to limit execution times
    - The interrupt defines a *time slice* for a process.
  - When the interrupt occurs, the currently running process is preempted (placed on the ready queue) and the next process to be dispatched is taken from the ready queue on a FCFS basis.
    - So: RR is essentially FCFS + time slicing.
    - RR involves more overhead (throughput is decreased) but the CPU is shared in a more equitable fashion.
    - The time quantum should be slightly greater than the time required for a “typical” transaction.
    - Note: fraction of time that process runs is $q/(q+v)$ where v represents overhead time.
- Pro:
  - RR is very effective in multi-user time-sharing environments and provides good response times for short processes.
  - Process starvation is not possible.
- Con:
  - There is tendency to favor CPU bound processes over I/O bound processes
    - the latter miss out on full time slices since they frequently block to do I/O.

Virtual RR Scheduling

- Strategy:
  - This is the same as RR except that an auxiliary queue is used to hold processes that have completed and I/O wait.
    - The auxilliary queue has a *higher* dispatch priority than the ready queue.
    - I/O bound processes now do better.



Shortest Process Next Scheduling

- The SPN Strategy:
  - The process with the shortest *expected* process time is selected next.
    - SPN is nonpreemptive.
  - SPN approximates the idealized (optimal) strategy of dispatching the process with the smallest *next* CPU burst.
    - If this could be done it would give us the optimal strategy in providing the minimum average wait time.
  - SPN attempts to predict the length of the next CPU burst by working with previous behaviour.
    - A common approach is to use an *exponential average*:
      $S_{n+1} = \alpha T_n + (1-\alpha)S_n$ where:
      S_n = predicted CPU burst for the nth dispatch
      T_n = actual CPU burst for the nth dispatch.
    - Usually α may be chosen as 0.5 (perhaps a bit more).
    - A high value of α will quickly reflect a rapid change in CPU bursts.
- Pro:
  - SPN provides high throughput and good response times for short processes.
- Con:
  - Starvation is possible and there is no preemption.

Effect of SPN

- Consider a ready queue containing three processes with next CPU bursts of:

  12, 3, and 9.

- Scenario A:
  - running the processes in a non-SPN order:

	12	9	3
Response times:	12	21	24

Average Response Time = (12+21+24) / 3 = 19

- Scenario B:
  - running the processes in the SPN order:

	3	9	12
Response times:	3	12	24

Average Response Time = (3+12+24) / 3 = 13

Shortest Remaining Time Scheduling (SRT)

- Strategy:
  - The dispatcher chooses the process that has the shortest expected remaining CPU burst.
    - The preemptive version of SPN.
    - If a new process in the ready queue has a predicted CPU burst that is shorter than the currently running process the dispatcher will let it run by preempting the running process.
    - Overhead is somewhat higher than SPN but the benefits are worth the extra expense.
- Pro:
  - SRT provides high throughput and good response times.
- Con:
  - Process starvation is possible.

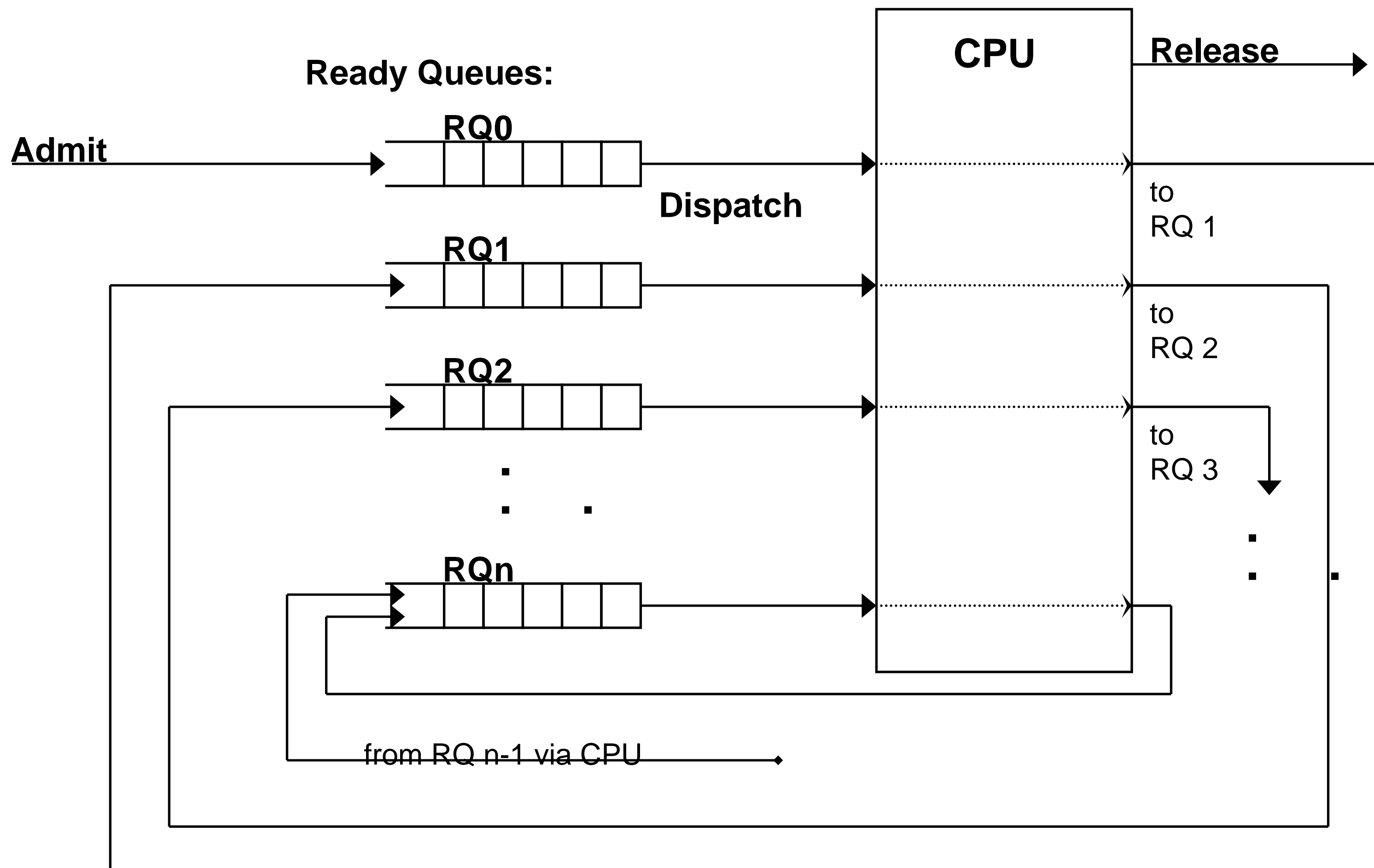
Highest Response Ratio Next Scheduling

- A Non-preemptive Strategy:
  - In an attempt to minimize the normalized turnaround time (ratio of turnaround time to service time) we choose the ready process with the greatest response ratio RR = (w + s)/s.
    - Here w is the time spent waiting for the CPU and s is the expected service time (CPU burst).
    - By picking the largest RR we help to reduce w so that the average normalized turnaround time is also reduced.
    - As for SRT and SPN the service time s is evaluated by prediction using exponential averaging.
  - Note that “aged” processes (no CPU activity for a long time) will automatically get preference.
    - HRRN is a good blend of FCFS and SPN.
- Pro:
  - Throughput is high and response time is good.
  - There is a good balance in the treatment of long and short processes
  - Process starvation is not possible.
- Con:
  - Overhead can be high.

Multilevel Feedback Scheduling

- Strategy:
  - Preemptive scheduling is done with a dynamic priority mechanism using multiple priority queues.
    - The idea is to automatically separate processes with different CPU-burst lengths.
    - A process starts at a particular queue level.
    - Each time a process is preempted by the end of a time slice it goes to a queue with a priority one level lower than the one just used.
    - Within each queue FCFS is used except for the lowest level queue which uses round-robin.
    - The effect is to penalize jobs that have longer CPU-bursts since short jobs complete before going into lower level queues.
    - Short burst processes (I/O bound and interactive) tend to stay in the higher queues.
    - The quantum duration may be longer for successively lower queues.
    - Does not depend on CPU burst prediction as does SPN, SRT, and HRRN.

Multilevel Feedback Scheduling (cont.)



Not Shown: paths leaving the CPU going to the wait event queues. After an event occurs, the process goes back to the *same* queue.

Multilevel Feedback Scheduling (cont.)

- Pro:
  - Feedback provides a simple yet effective strategy.
  - It can be modified for various needs (see the NT approach later).
- Con:
  - I/O-bound processes tend to be favoured and starvation is possible.
    - To avoid starvation we can promote older processes to a higher-priority queue.
- Design aspects:
  - The multilevel feedback scheduling strategy covers a variety of different possibilities specified by the following parameters:
    - the number of queues
    - the scheduling algorithm used for each queue
    - the policy used to upgrade a process in order to avoid starvation
    - the policy used to determine the starting queue for a process having its first CPU burst.
    - the policy used for demotion in the queues.