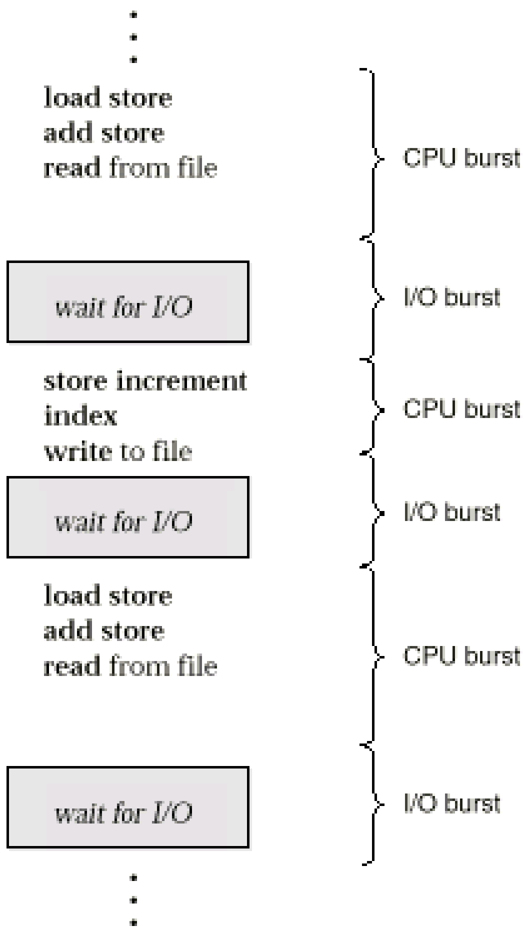
Operating Systems

CPU Scheduling

- Fundamentally, scheduling is a matter of managing queues to minimize queuing delay and to optimize performance in a queuing environment.
- Scheduling needs to meet system objectives, such as:
  - ✓minimize response time
  - ✓maximize throughput
  - ✓maximize processor efficiency
  - ✓support multiprogramming
- Scheduling is central to OS design

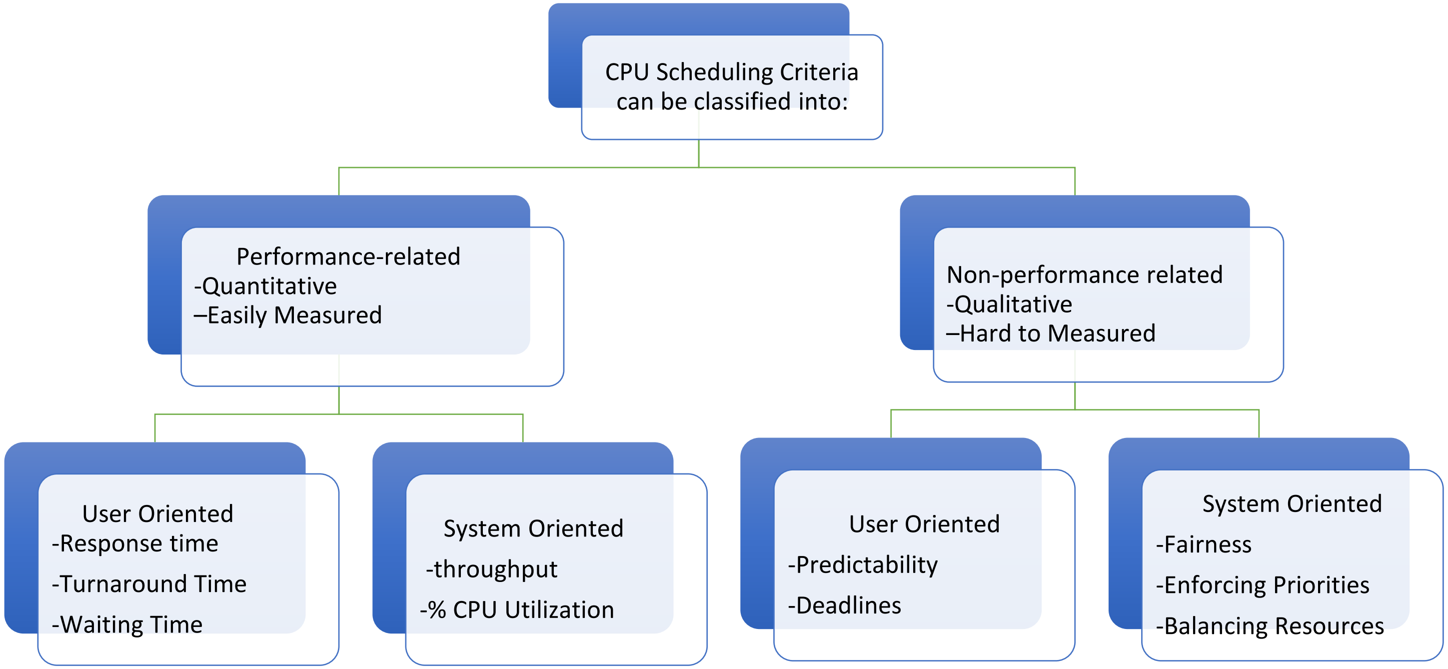


"CPU-bound" processes require more CPU time than I/O time

"I/O-bound" processes spend most of their time waiting for I/O.

- **Uniprocessor Scheduling**: scheduling a *single* CPU among all the processes in the system

Short-Term Scheduling Criteria: Performance



- Turnaround time
  - ✓ Time from submission to completion (batch Process)
  - ✓ CPU-bound process equivalent of response time
- Response time
  - ✓ Time to start responding (interactive users)
  - ✓ I/O bound processes
- Throughput
  - ✓ Number of Process processed per unit of time
  - ✓ Productivity
- Processor(CPU) utilization
  - ✓ Percent of time CPU is busy
  - ✓ Efficiency
- Waiting time
  - ✓ Time spent waiting in READY Queue
  - ✓ Each process should get a fair share of the CPU

- Predictability
  - ✓ Same time/cost regardless of load on the system
- Deadlines
  - ✓ Maximize number of deadlines met
- Fairness
  - ✓ No process should suffer starvation
- Enforcing priorities
  - ✓ Favor higher priority processes
- Balancing resources
  - ✓ Keep system resources busy

- All systems
  - ✓ Fairness: give each process a fair share of the CPU
  - ✓ Enforcement: ensure that the stated policy is carried out
  - ✓ Balance: keep all parts of the system busy
- Batch systems
  - ✓ Throughput: maximize jobs per unit time (hour)
  - ✓ Turnaround time: minimize time users wait for jobs
  - ✓ CPU utilization: keep the CPU as busy as possible
- Interactive systems
  - ✓ Response time: respond quickly to users' requests
  - ✓ Proportionality: meet users' expectations
- Real-time systems
  - ✓ Meet deadlines: missing deadlines is a system failure!
  - ✓ Predictability: same type of behavior for each time slice

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
  - ✓ When does scheduler make decisions? When a process :
    1. Switches from running to waiting state
    2. Switches from running to ready state
    3. Switches from waiting to ready
    4. Terminates
- Scheduling under 1 and 4 is **non-preemptive**
  - ✓ Processes keep CPU until it releases either by terminating or I/O wait.
- All other scheduling is **preemptive**
  - ✓ Interrupts

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - ✓switching context
  - ✓switching to user mode
  - ✓Pumping 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

- Important in scheduling and resource allocation algorithms
- Policy
  - ✓What is to be done
- Mechanism
  - ✓How to do it
- Policy: All users equal access
- Mechanism: round robin scheduling
- Policy: Paid Ps get higher priority
- Mechanism: Preemptive scheduling algorithm

- Dispatcher
  - ✓ Low-level: mechanism
  - ✓ Responsibility: context switch
- Scheduler
  - ✓ High-level: policy
  - ✓ Responsibility: deciding which process to run
- Could have an allocator for CPU as well
  - ✓ Parallel and distributed systems

Selection function

- Which process in the ready queue is selected next for execution?

Decision mode

- At what times is the selection function exercised?
  - ✓ **Nonpreemptive**
    - ❑ **A process in the running state runs until it blocks or ends**
  - ✓ **Preemptive**
    - ❑ **Currently running process may be interrupted and moved to the Ready state by the OS**
    - ❑ **Prevents any one process from monopolizing the CPU**

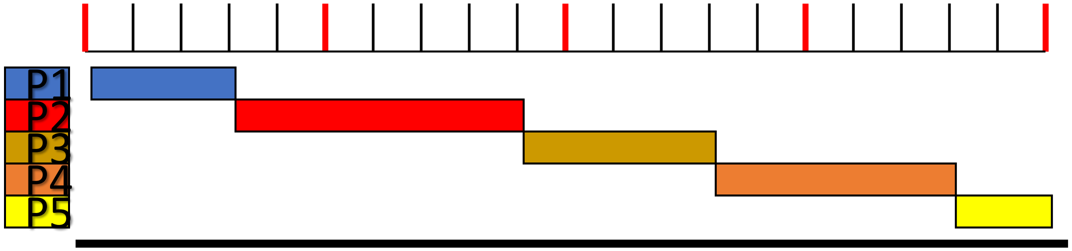
Process	Arrival Time	Burst Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

- **Selection function:** the process that has been waiting the longest in the ready queue (hence, FCFS, FIFO queue)
- **Decision mode**: non-preemptive- a process runs until it blocks itself (I/O or other)
- Pros:
  - ✓ Simple to implement
  - ✓ Fair policy
- Cons:
  - ✓ Favors CPU-bound processes
    - ❑ A process that does not perform any I/O will monopolize the processor!
    - ❑ I/O-bound processes have to wait until CPU-bound process completes
    - ❑ They may have to wait even when their I/Os have completed
      - o poor device utilization
    - ❑ We could reduce the average wait time by giving more priority to I/O bound processes
  - ✓ waiting time depends on arrival order
  - ✓ **Convoy effect** - short process stuck waiting for long process

P #	1	2	3	4	5
Arrival time	0	2	4	6	8
Finish time	3	9	13	18	20
TAT/Response time	**3**	**7**	**9**	**12**	**12**
Service time	3	6	4	5	2
Wait time	**0**	**1**	**5**	**7**	**10**

Average response time = (3+7+9+12+12)/5 = 8.6
Throughput = 5/20 = 1/4
Ave (RespTime/ServTime) = (1+7/6+9/4+12/5+12/2)/5 = 2.564

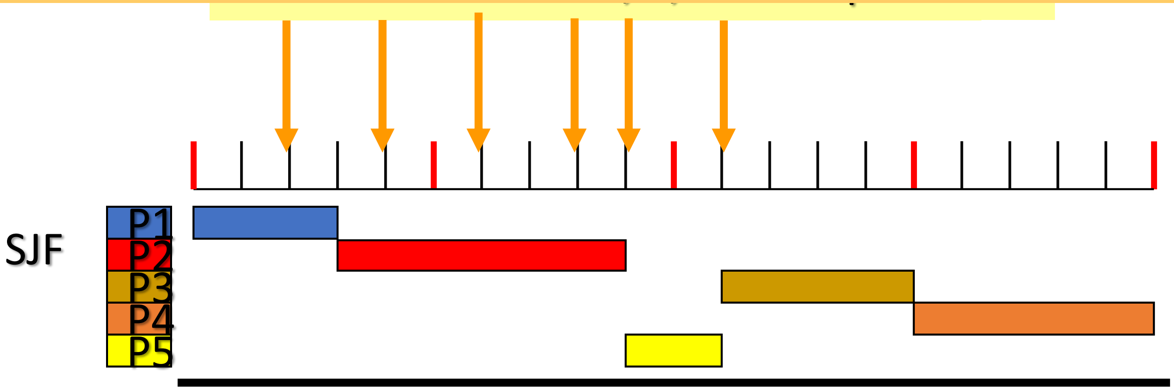
FIFO



- **Selection function:** the process with the shortest expected CPU burst time
- **Decision mode:** non-preemptive
- SJF implicitly incorporates priorities: shortest Jobs are given preference.
  - ✓ Typically these are I/O bound Ps
- Lack of preemption not suitable in a time sharing environment
  - ✓ CPU bound process gets lower priority
  - ✓ But a process doing no I/O at all could monopolize the CPU if it is the first one in the system
- Advantages:
  - ✓ Minimizes average wait time. Provably optimal
- Disadvantages
  - ✓ Not practical: difficult to predict burst time
    - ❑ Possible: past predicts future
  - ✓ May starve long jobs

P #	1	2	3	4	5
Arrival time	0	2	4	6	8
Finish time	3	9	15	20	11
TAT/Response time	**3**	**7**	**11**	**14**	**3**
Service time	3	6	4	5	2
Wait time	**0**	**1**	**7**	**9**	**1**

Average response time = (3+7+11+14+3)/5 = 7.6
Throughput = 5/20 = 1/4
Ave (RespTime/ServTime) = (1+7/6+11/4+14/5+3/2)/5 = 1.844



- Can average all past history equally
- But recent history of a process is more likely to reflect future behavior
- A common technique for that is to use *exponential averaging*

1. $t_n =$ actual length of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

$$\tau_{n=1} = \alpha\, t_n + (1-\alpha)\tau_n .$$

✓Puts more weight on recent instances whenever $\alpha > 1/n$

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

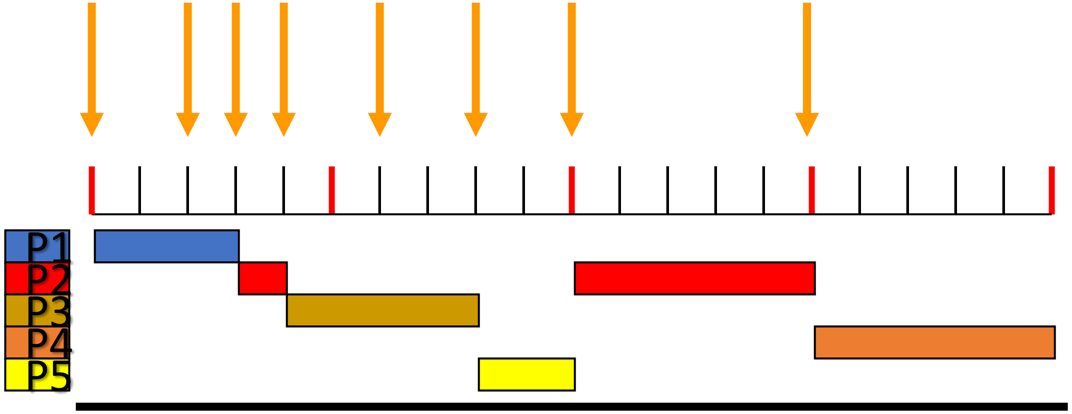
$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_n - 1 + \ldots$$
$$+ (1 - \alpha)^P \alpha\, t_{n-P} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

- Shortest Remaining Time (SRT) = Preemptive SJF
- If a process arrives in the Ready queue with estimated CPU burst less than remaining time of the currently running process, preempt.
- Prevents long Process's from dominating.
  - ✓But must keep track of remaining burst times
- Better turnaround time than SJF
  - ✓Short Process's get immediate preference

P #	1	2	3	4	5
Arrival time	0	2	4	6	8
Finish time	3	15	8	20	10
TAT/Response time	**3**	**13**	**4**	**14**	**2**
Service time	3	6	4	5	2
Wait time	**0**	**7**	**0**	**9**	**0**

Average response time = (3+13+4+14+2)/5 = 7.2
Throughput = 5/20 = 1/4
Ave (RespTime/ServTime) = (1+13/6+1+14/5+1)/5 = 1.6



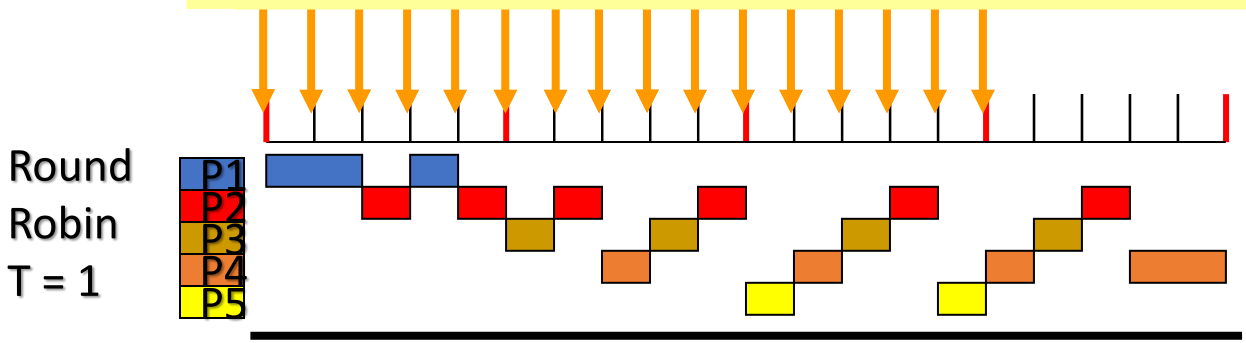
- **Selection function:** same as FCFS
- **Decision mode:** Preemptive
- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds.
  - ✓ After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once.
  - ✓ No process waits more than $(n-1)q$ time units.

P #	1	2	3	4	5
Arrival time	0	2	4	6	8
Finish time	4	18	17	20	15
TAT/Response time	**4**	**16**	**13**	**14**	**7**
Service time	3	6	4	5	2
Wait time	**1**	**10**	**9**	**9**	**5**

Average response time = (4+16+13+14+7)/5 = 10.8

2

P P S S S S S S S S S S S S S Switch to P4

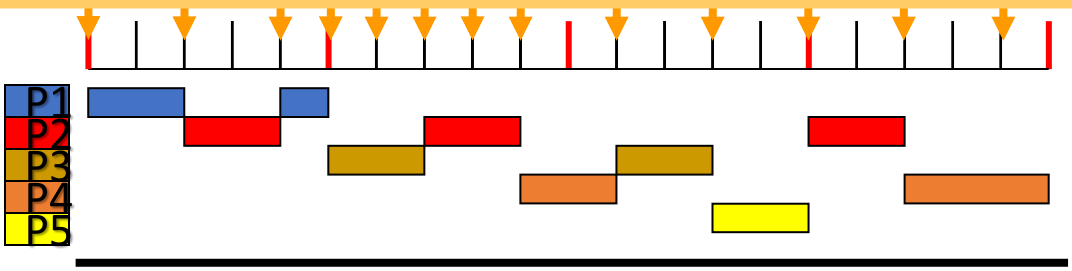


P #	1	2	3	4	5
Arrival time	0	2	4	6	8
Finish time	5	17	13	20	15
TAT/Response time	**5**	**15**	**9**	**14**	**7**
Service time	3	6	4	5	2
Wait time	**2**	**9**	**5**	**9**	**5**

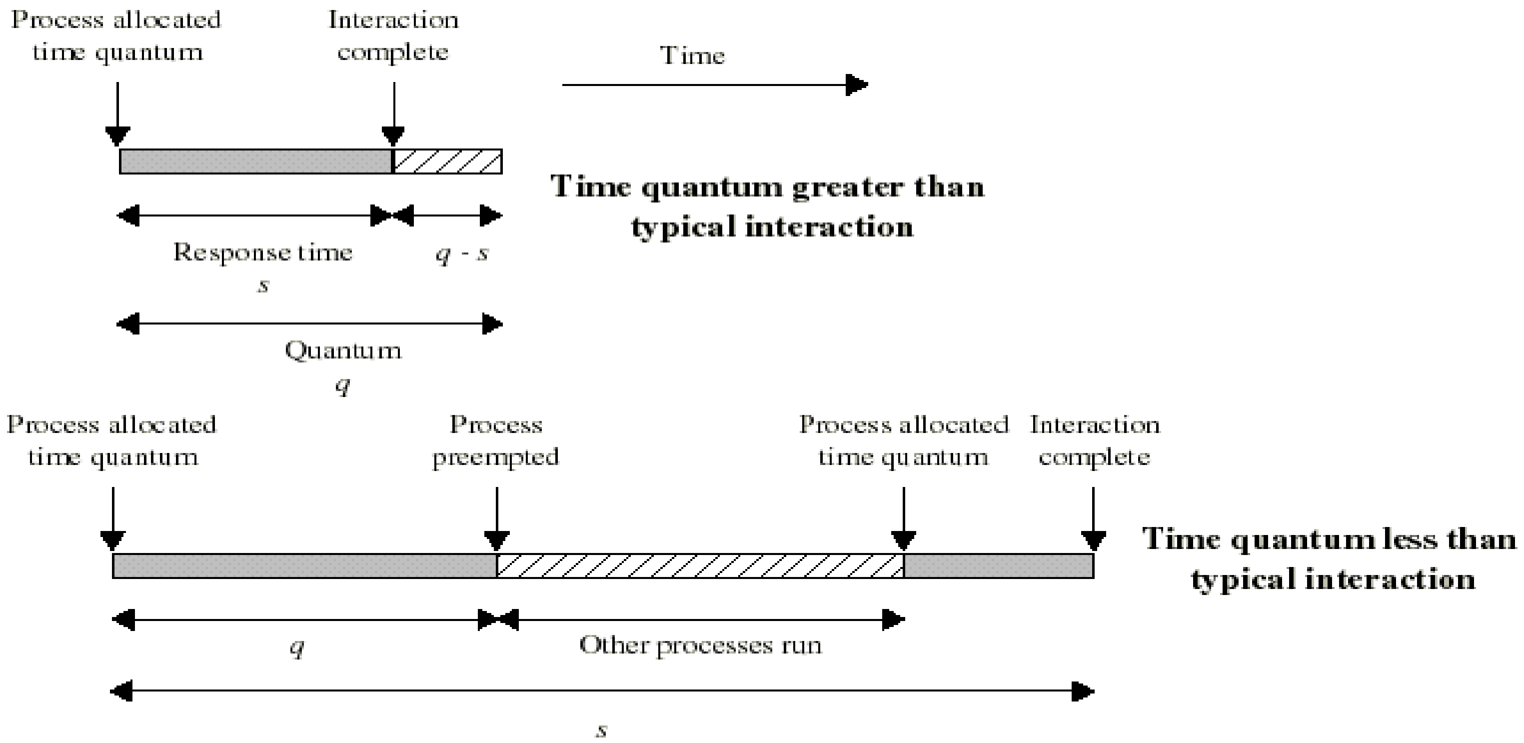
P2 e P3 P P4 P3 P4 P2 t P4 t P3 f P5 f P2 fi P4 time expires
Que Que Q Q Q Q Q Que Que Que Que Que Queue: P4
P1 s Swit Sw Sw P3 Sw P3 Swit Swit Swit Swit Swit P4 Continues

Average response time = (5+15+9+14+7)/5 = 10

Round
Robin
T = 2



- Must be substantially larger than process switch time
- Should be larger than the typical CPU burst
- If too large, degenerates to FCFS
- Too small, excessive context switches (overhead)



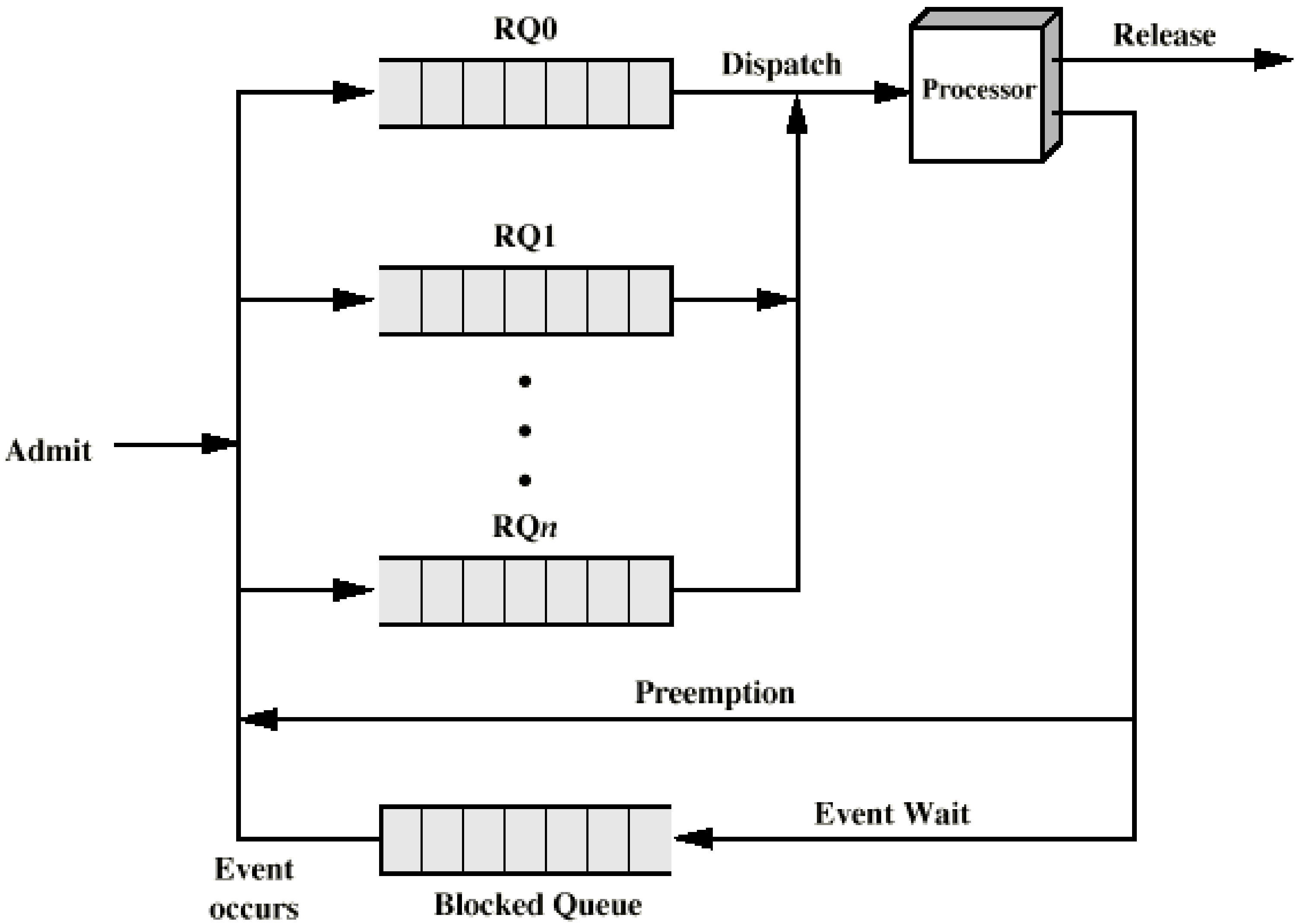
- Each context switch has the OS using the CPU instead of the user process
  - ✓ give up CPU, save all info, reload w/ status of incoming process
  - ✓ Say 20 ms quantum length, 5 ms context switch
  - ✓ Waste of resources
    - ❑ 20% of CPU time (5/20) for context switch
  - ✓ If 500 ms quantum, better use of resources
    - ❑ 1% of CPU time (5/500) for context switch
    - ❑ Bad if lots of users in system – interactive users waiting for CPU
  - ✓ Balance found depends on P mix

- Advantages
  - ✓ Low response time, good interactivity
  - ✓ Fair allocation of CPU across processes
  - ✓ Low average waiting time when job lengths vary widely
- Disadvantages
  - ✓ Poor average waiting time when jobs have similar lengths
    - ❑ Average waiting time is even worse than FCFS!
  - ✓ Performance depends on length of time slice
  - ✓ Still favours CPU-bound processes
    - ❑ An I/O bound process uses the CPU for a time less than the time quantum and then is blocked waiting for I/O
    - ❑ A CPU-bound process runs for its whole time slice and goes back into the ready queue (in front of the blocked processes)

- One solution: virtual round robin (VRR)
  - ✓ When a I/O has completed, the blocked process is moved to an auxiliary queue which gets *preference* over the main ready queue
  - ✓ A process dispatched from the auxiliary queue gets a shorter time quantum (what is “left over” from its quantum when it was last selected from the ready queue)

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - ✓Preemptive
  - ✓nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

Scheduler chooses from low priority queue only if higher ones



- Non-preemptive, tries to get best average normalized turnaround time
- Choose next process with the highest ratio

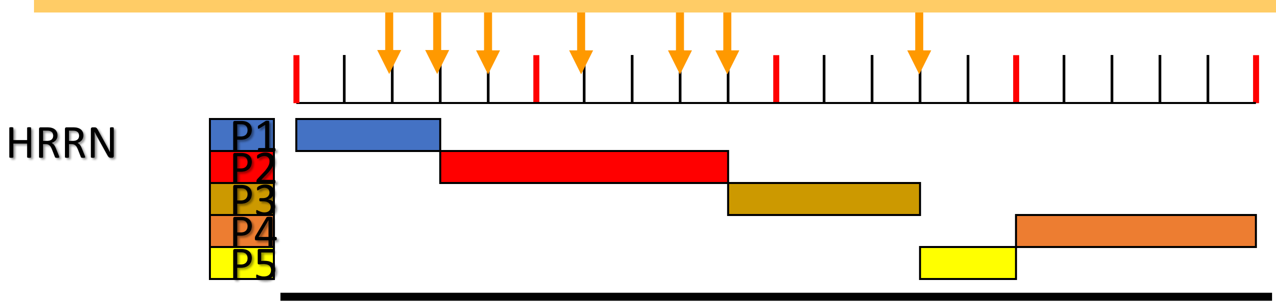
$$Ratio = \frac{time\ spent\ waiting + expected\ service\ time}{expected\ service\ time}$$

- Attractive approach to scheduling because it accounts for the age of a process.
- While shorter processes are favored (a smaller denominator yields a larger ratio), aging without service increases the ratio so that a longer process will eventually get past competing shorter processes .

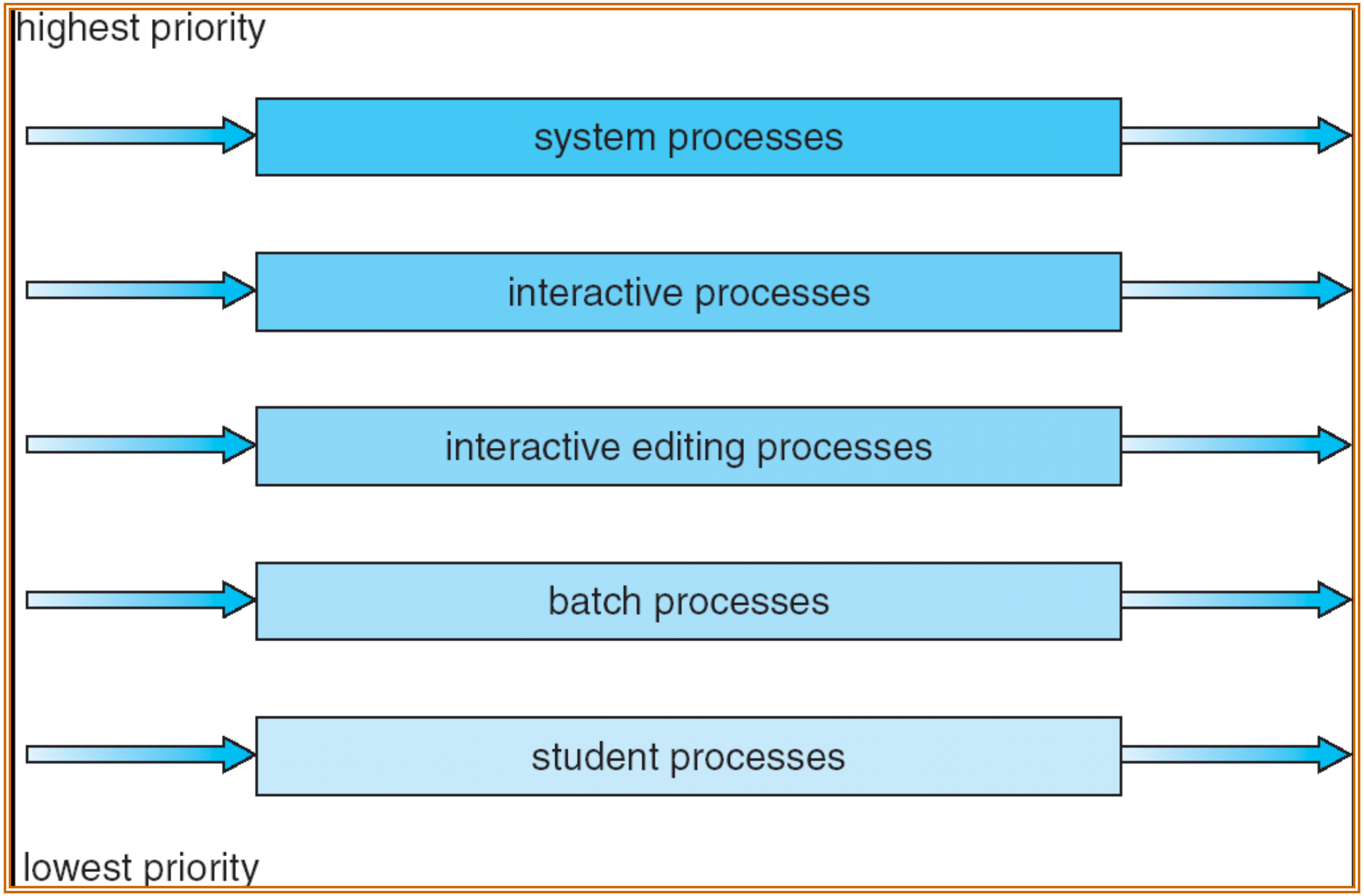
RR = (waiting-time + service-time) / service-time

Process #	1	2	3	4	5
Arrival time	0	2	4	6	8
Finish time	3	9	13	20	15
TAT/Response time	**3**	**7**	**9**	**14**	**7**
Service time	3	6	4	5	2
Wait time	**0**	**0**	**7**	**9**	**0**

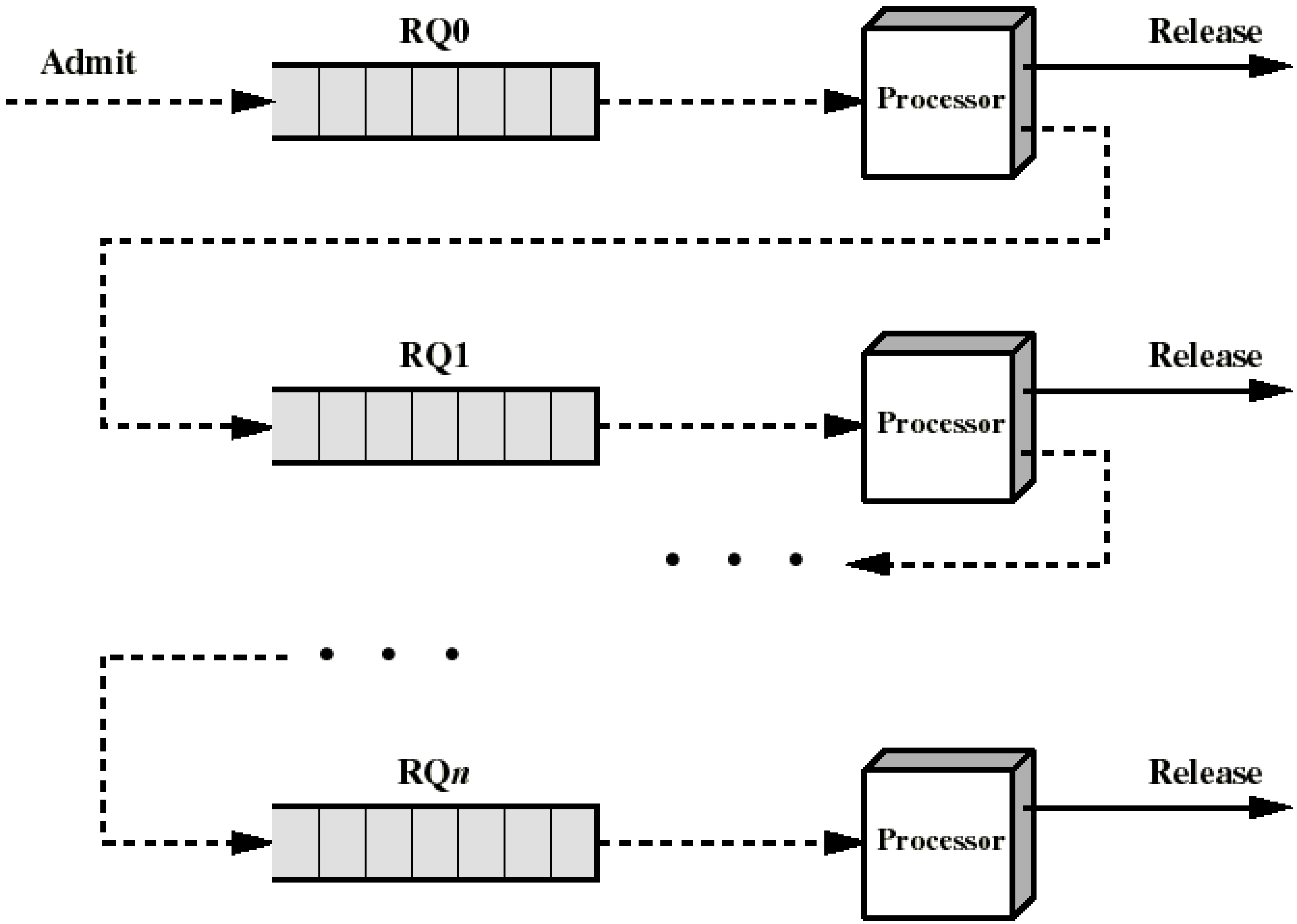
Average response time = (3+7+9+14+7)/5 = 8
Throughput = 5/20 = 1/4
Ave (RespTime/ServTime) = (1+7/6+9/4+14/5+7/2)/5 = 2.144



- Ready queue is partitioned into separate queues:
  foreground (interactive)
  background (batch)
- Each queue has its own scheduling algorithm
  - ✓foreground – RR
  - ✓background – FCFS
- Scheduling must be done between the queues
  - ✓Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - ✓Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - ✓20% to background in FCFS

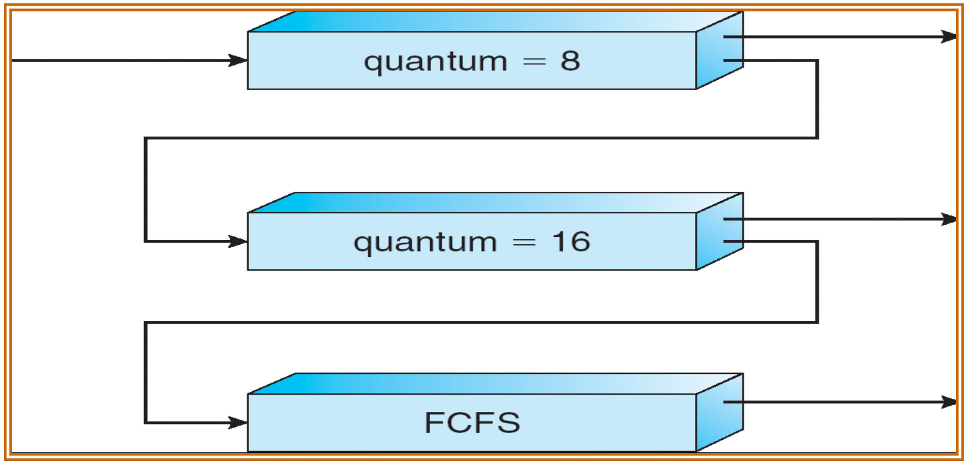

highest priority
system processes
interactive processes
interactive editing processes
batch processes
student processes
lowest priority

- Several READY queues with decreasing priorities:
  - ✓ P(RQ0) > P(RQ1) > ... > P(RQn)
  - ✓ But lower priority queues get longer time slice
- New processes are placed in RQ0
- If they use their time quantum, they are placed in RQ1. If they time out again, they go to RQ2, etc. until they reach lowest priority
- Automatically reduces priority of CPU-bound Ps, leaving I/O-bound ones at top priority
- Dispatcher always chooses a process from highest non-empty queue
- Problem: long Ps can “starve”
  - ✓ Solution: “aging” (promote priority of a process that waits too long in a lower priority queue)



- Note: many variations of policies possible
  - ✓ including different policies at different levels..

- Three queues:
  - ✓ Q_0 – RR with time quantum 8 milliseconds
  - ✓ Q_1 – RR time quantum 16 milliseconds
  - ✓ Q_2 – FCFS
- Scheduling
  - ✓ A new P enters queue Q_0 which is served FCFS. When it gains CPU, P receives 8 milliseconds. If it does not finish in 8 milliseconds, P is moved to queue Q_1.
  - ✓ At Q_1 P is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.



- Multilevel feedback scheduling is the most flexible method
- Can customize:
  - ✓Number of queues
  - ✓Scheduling algorithm per queue
  - ✓Priority “upgrade” criteria
  - ✓Priority “demotion” criteria
  - ✓Policy to determine which queue a process starts in

Bibliography

- Silberschatz, A, Galvin, P.B, and Gagne, G., Operating System Principles, 9e, John Wiley & Sons, 2013.
- Stallings W., Operating Systems-Internals and Design Principles, 7e, Pearson Education, 2014.
- Harvey M. Deital, “Operating System”, Third Edition, Pearson Education, 2013.
- Andrew S. Tanenbaum, “Modern Operating Systems”, Second Edition, Pearson Education, 2004.
- Gary Nutt, “Operating Systems”, Third Edition, Pearson Education, 2004.

Acknowledgements

- I have drawn materials from various sources such as mentioned in bibliography or freely available on Internet to prepare this presentation.
- I sincerely acknowledge all sources, their contributions and extend my courtesy to use their contribution and knowledge for educational purpose.

Thank You!!

?