# Operating Systems (Client)

## Lecture 4
## Uniprocessor Scheduling

Dr. Kevin Farrell

# Uniprocessor Scheduling

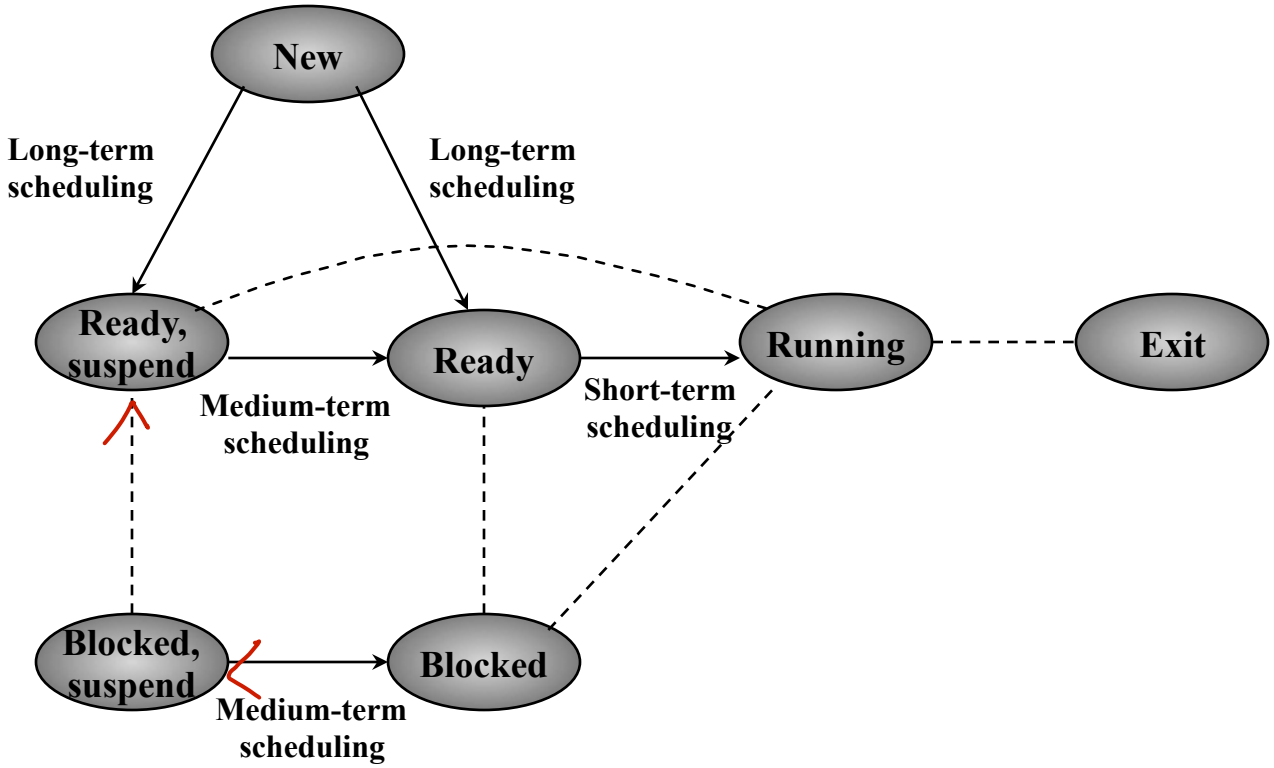
# Goals of Scheduling

- Quick response time
- Fast throughput
- Processor efficiency
- To be fair to all users
- Degrade performance gracefully
- Be consistent and predictable

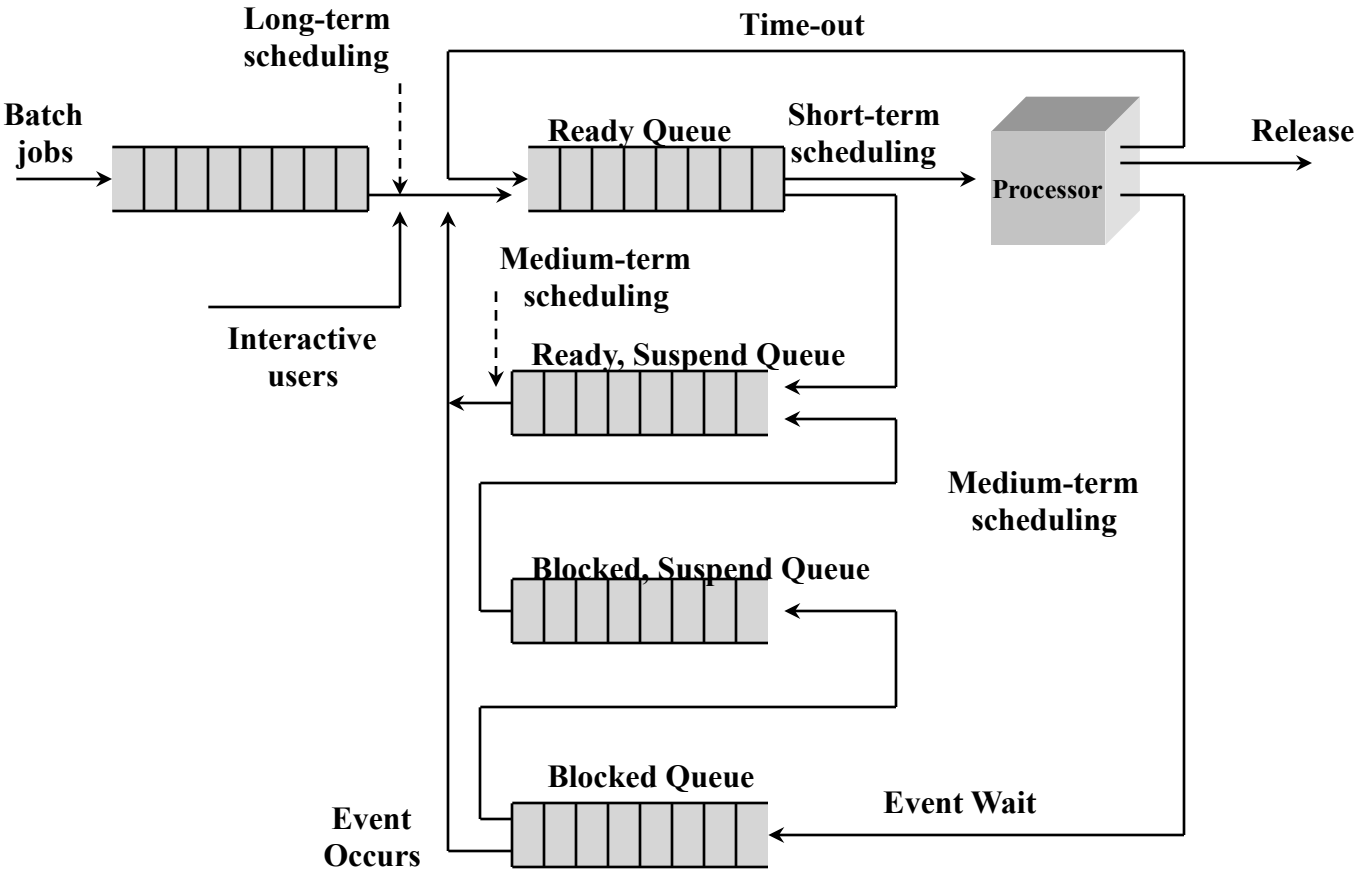
# Type of Scheduling

- Long-term
  - performed when new process is created
- Medium-term
  - swapping
- Short-term
  - which ready process to execute next
- I/O
  - decision as to which process's pending I/O request shall be handled by available I/O device

# Scheduling and Process State Transition



# Queuing Diagram for Scheduling



# Long-Term Scheduling

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
- More processes, smaller percentage of time each process is executed

# Medium-Term Scheduling

- Swapping
- Based on the need to manage multiprogramming

transfer of pages between physical and virtual memory

# Short-Term Scheduling

- Sometimes known as the dispatcher (but the dispatcher is just a portion of the short-term scheduler)
- Invoked when an event occurs
  - clock interrupts
  - I/O interrupts
  - operating system calls
  - signals

# Short-Tem Scheduling Criteria

- User-oriented
  - Response Time
    - Elapsed time between the submission of a request until there is output.
- System-oriented
  - effective and efficient utilization of the processor

# Short-Term Scheduling Criteria

- Performance-related
  - measurable such as response time and throughput
- Not performance related
  - predictability

Need to use a 'metric' to measure performance.

# Priorities

Execute next, or, put to the head of the Queue

- Scheduler will always choose a process of higher priority over one of lower priority
- Have multiple ready queues to represent each level of priority – 'Multi-level Queues'
- Lower-priority may suffer starvation
  - allow a process to change its priority based on its age or execution history

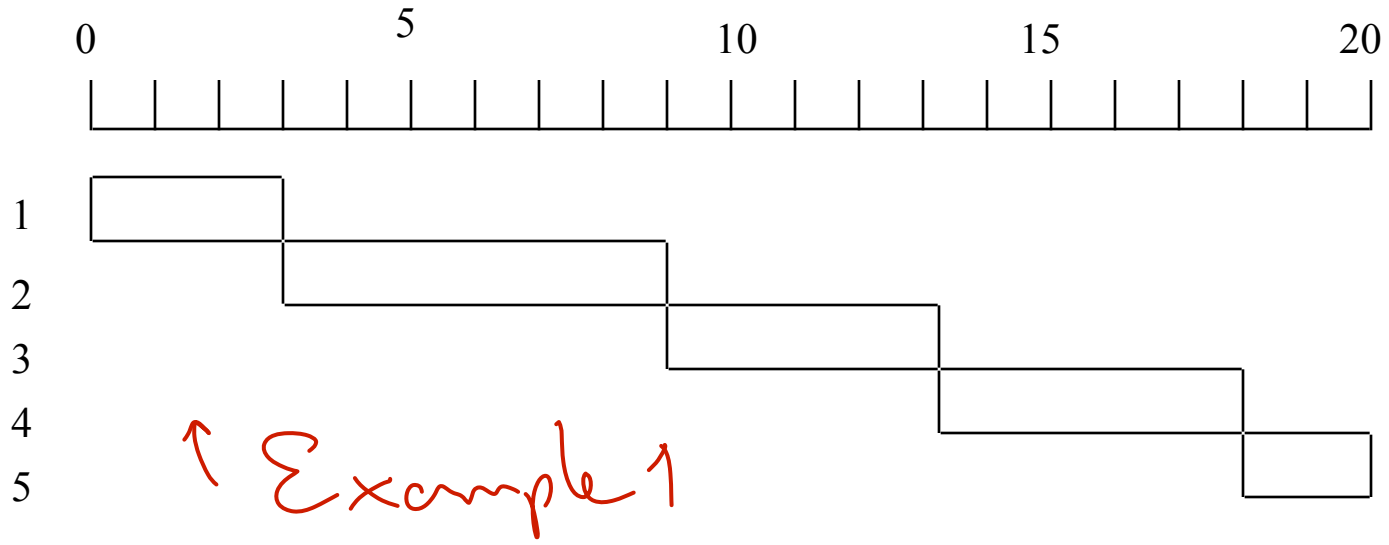
Multi-level feedback Queues

may never be given an opportunity to be executed in a reasonable time

# Decision Mode

- Nonpreemptive
  - Once a process is in the running state, it will continue until it terminates or blocks itself for I/O
- Preemptive
  - Currently running process may be interrupted and moved to the Ready state by the operating system
  - Allows for better service since any one process cannot monopolize the processor for very long

# First-Come-First-Served (FCFS)




- Each process joins the Ready queue
- When the current process ceases to execute, the oldest process in the Ready queue is selected

# First-Come-First-Served (FCFS)

- A short process may have to wait a very long time before it can execute
- Favors CPU-bound processes
  - I/O-bound processes have to wait until CPU-bound process complete

Example 2

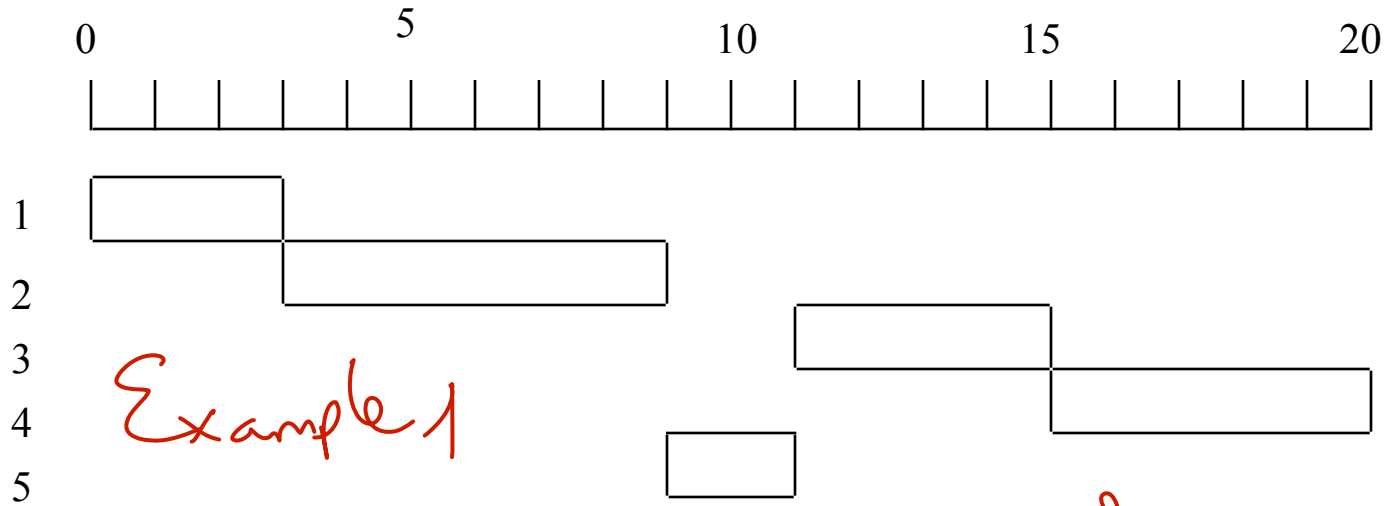
'Metric 1'

**Example of FCFS Policy:** *(p67 Ritchie)*

| | (a) | (b) | (c) |
|---|---|---|---|
| *Job* | *Est. Run Time* | *Waiting* | *Ratio b/a* |
| 1 | 2 | 0 | 0 |
| 2 | 60 | 2 | $\frac{2}{60} = 0.03$ |
| 3 | 1 | 62 | 62 |
| 4 | 3 | 63 | 21 |
| 5 | 50 | 66 | 1.32 |

'Metric 2' = Average Wait time = 193 ÷ 4 = 48 s (approx.)
(Exclude the first process)

# Shortest Job First (SJF)




- Nonpreemptive policy
- Process with shortest expected processing time is selected next
- Short process jumps ahead of longer processes

Incurs overhead

# Shortest Job First

- Predictability of longer processes is reduced
- If estimated time for process not correct, the operating system may abort it
- Possibility of starvation for longer processes

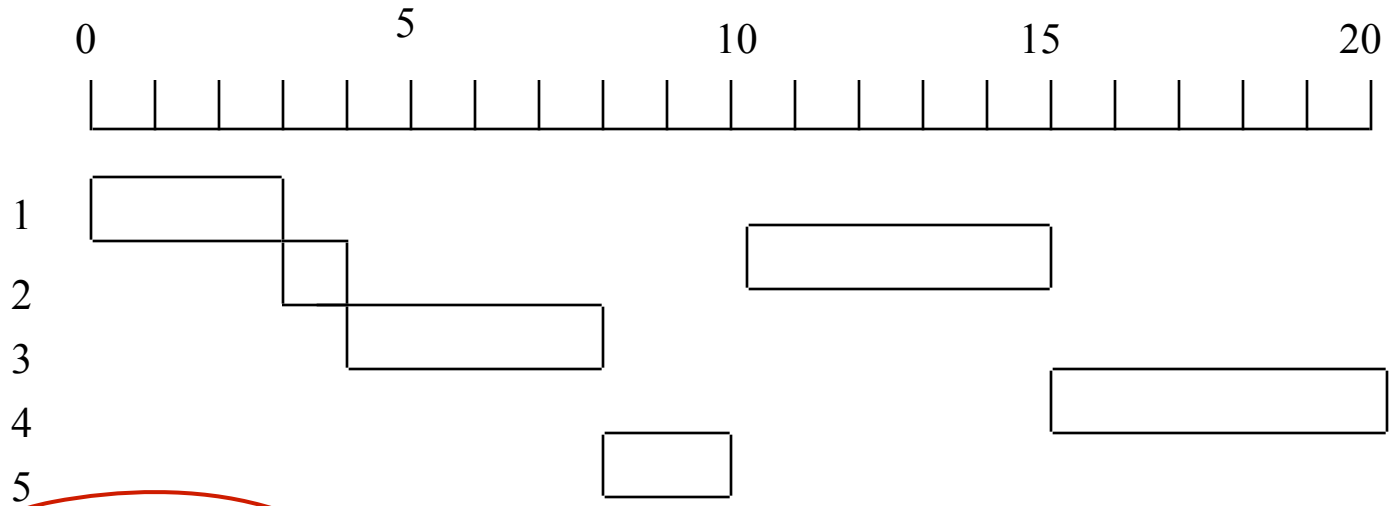
# Example 2

| | (a) | (b) | (c) |
|---|---|---|---|
| *Job* | *Est. Run Time* | *Waiting* | *Ratio b/a* |
| 3 | 1 | 0 | 0.0 |
| 1 | 2 | 1 | 0.5 |
| 4 | 3 | 3 | 1.0 |
| 5 | 50 | 6 | 0.1 |
| 2 | 60 | 56 | 0.9 |

Average Wait time = 66 ÷ 4 = 16.5 (approx)

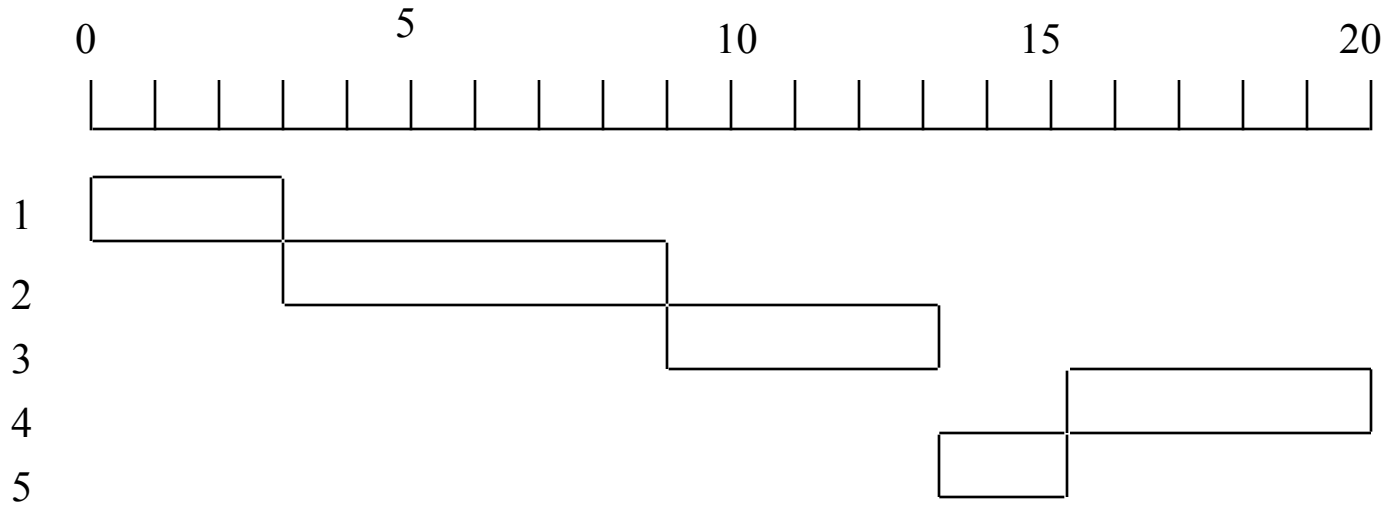
Recall, FCFS Av. wait time = 48 s !!!

# Shortest Remaining Time




- Preemptive version of shortest process next policy
- Must estimate processing time => incurs overhead

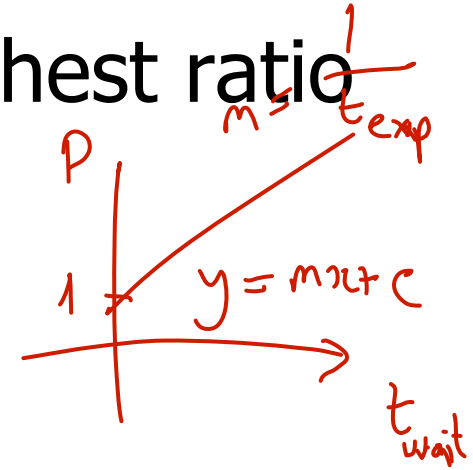
# Highest Response Ratio Next (HRRN)



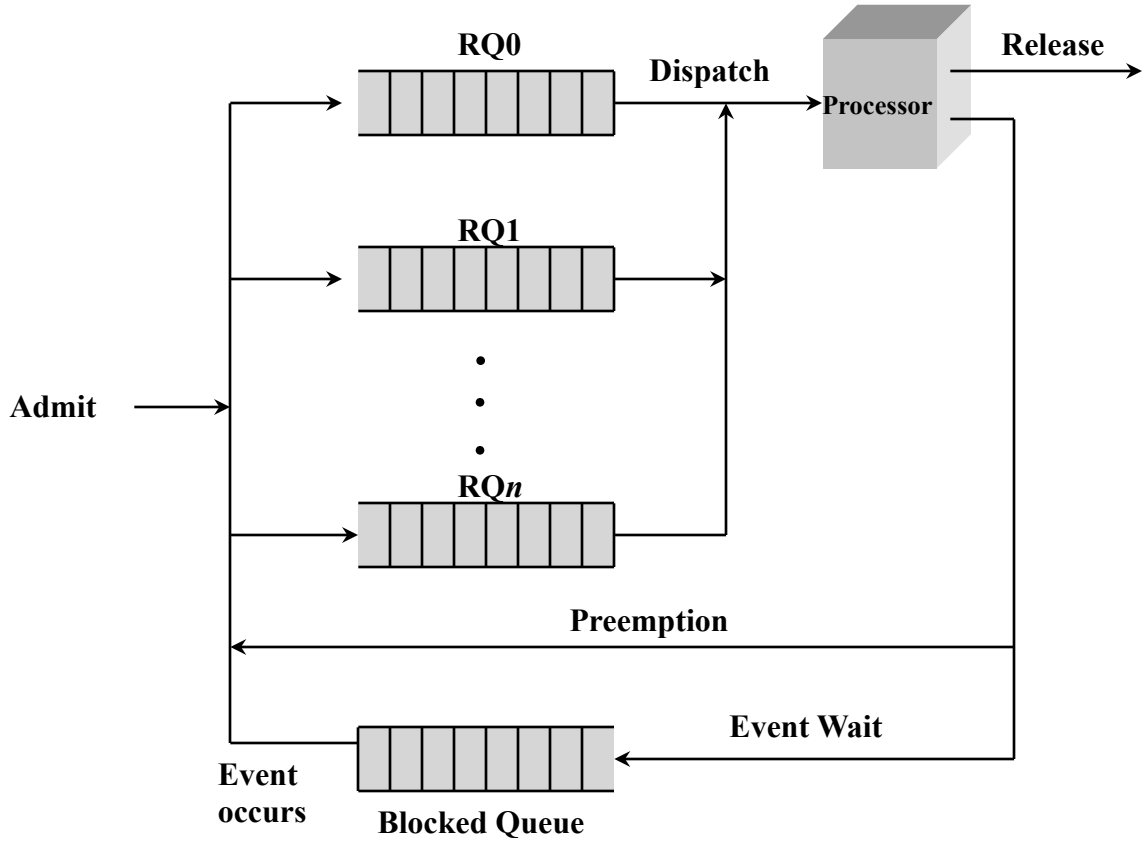

- Choose next process with the highest ratio

$$p = \frac{\text{time waiting + expected run time}}{\text{expected run time}}$$

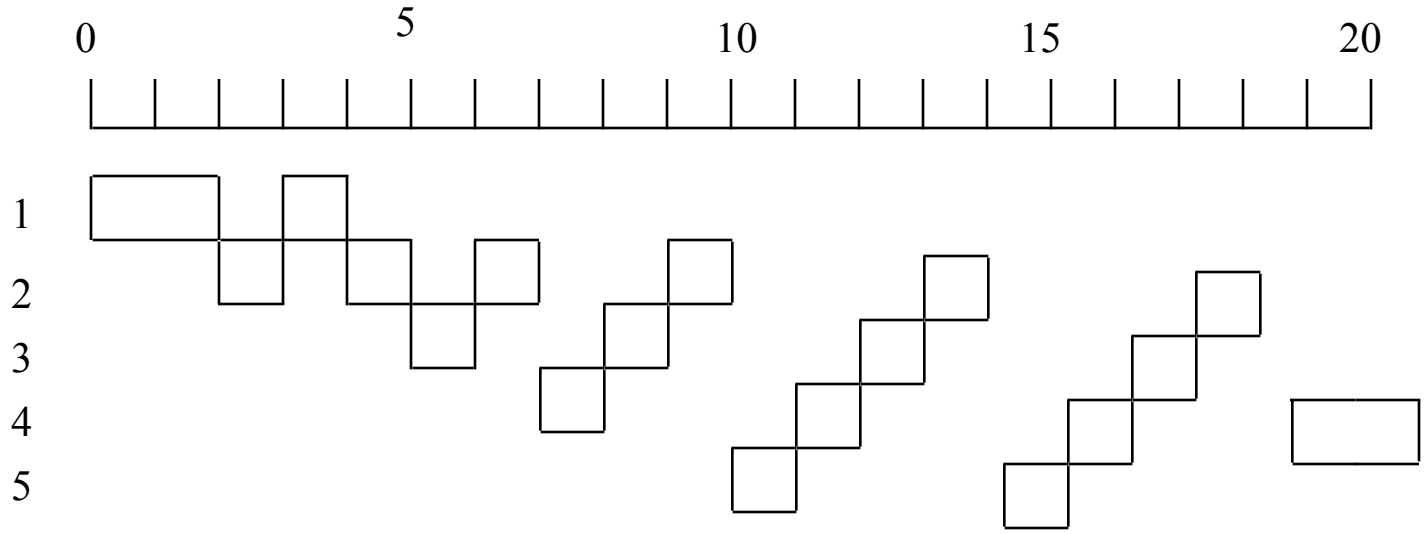
$$p = t_{wait}\left(\frac{1}{t_{exp}}\right) + 1$$



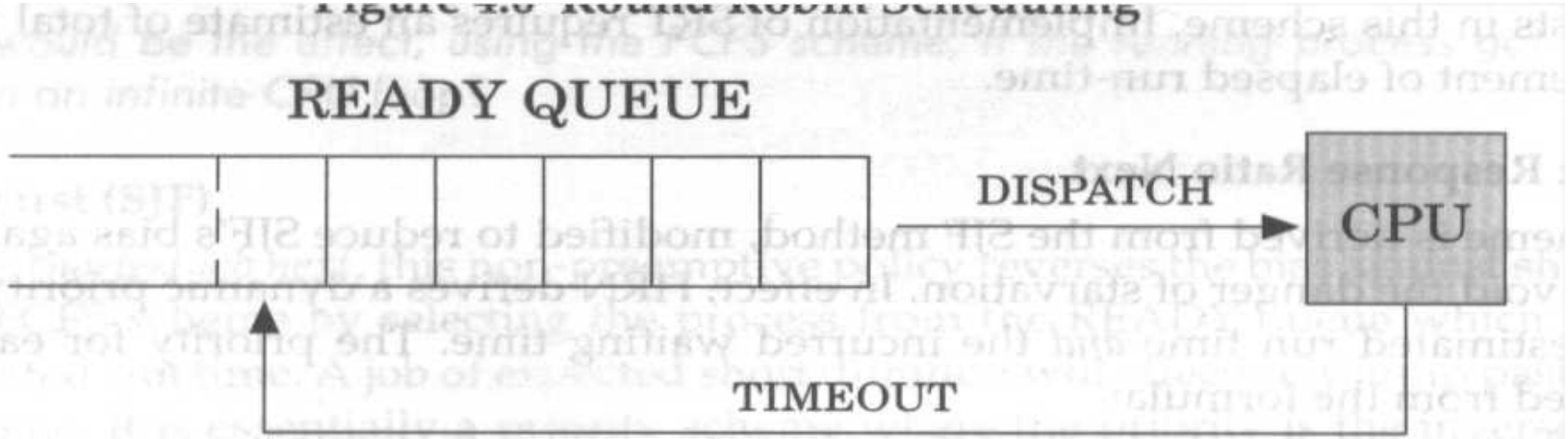
# Priority Queuing

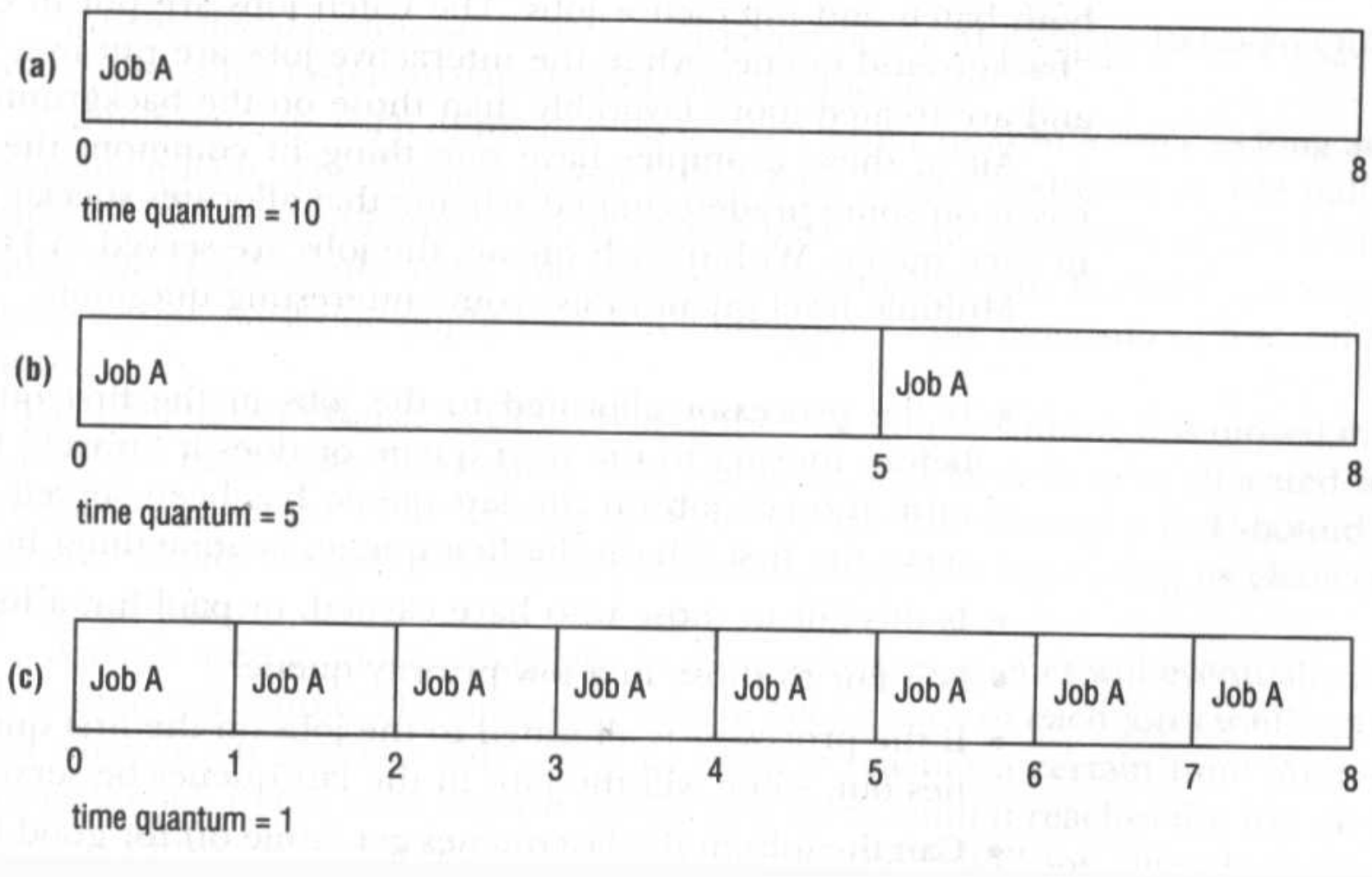


# Round-Robin



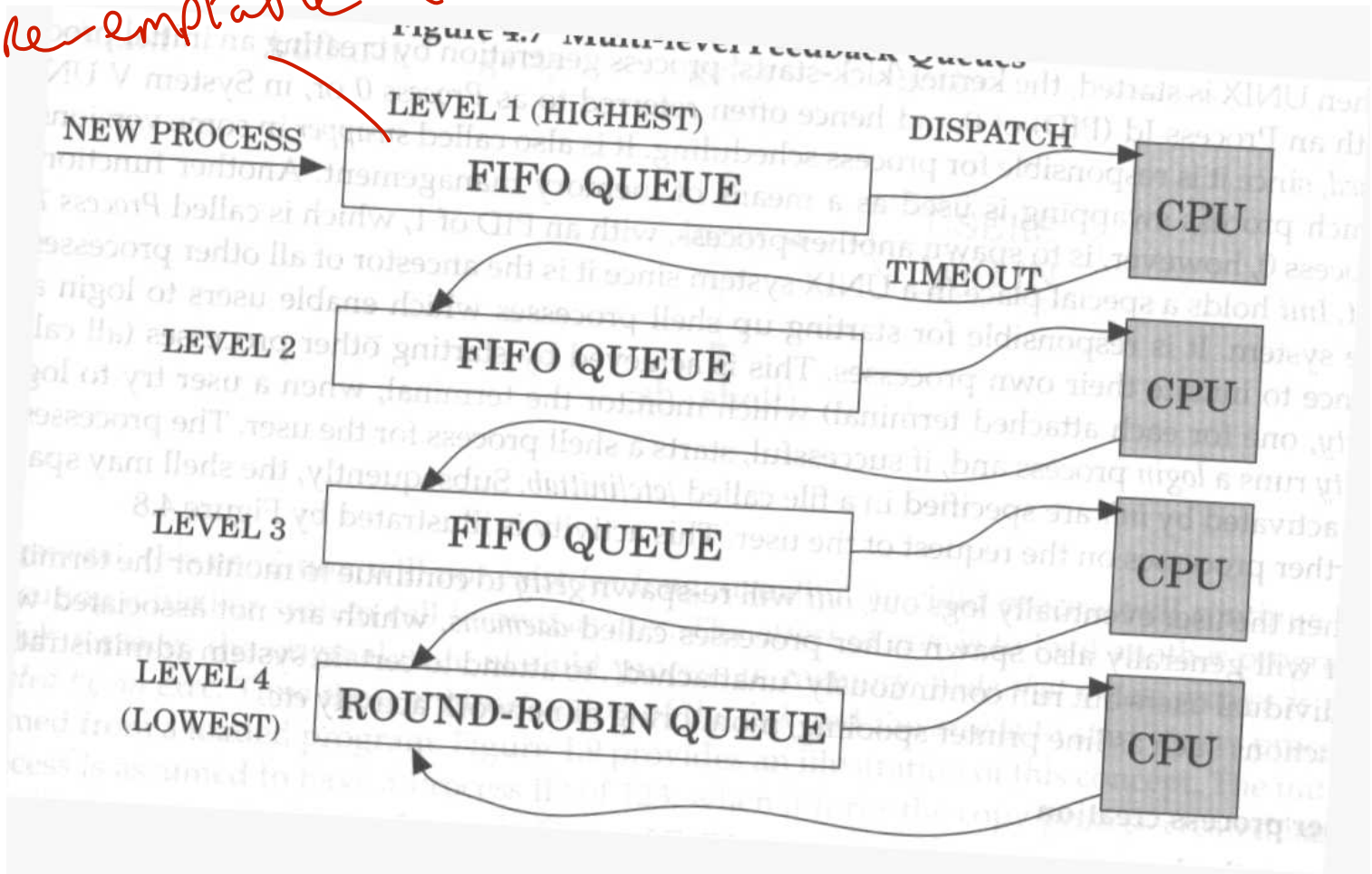

- Uses preemption based on a clock
- An amount of time is determined that allows each process to use the processor for that length of time: **Time Quantum**


READY QUEUE
DISPATCH
CPU
TIMEOUT

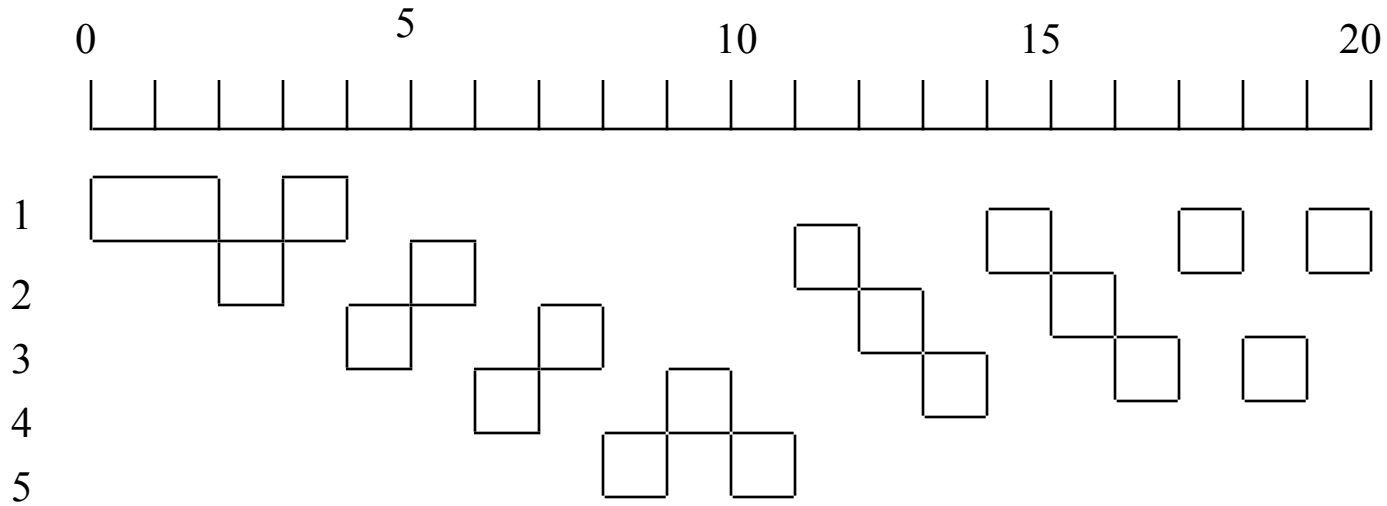

(a)
Job A
0
8
time quantum = 10
(b)
Job A
Job A
0
5
8
time quantum = 5
(c)
Job A
Job A
Job A
Job A
Job A
Job A
Job A
Job A
0
1
2
3
4
5
6
7
8
time quantum = 1

# Multi-Level Feedback Queues

Preemptable Queues



# Feedback




- Penalize jobs that have been running longer
- Don’t know remaining time process needs to execute

# Fair Scheduling

- User's application runs as a collection of processes (threads)
- User is concerned about the performance of the application
- Need to make scheduling decisions based on groups of processes

# UNIX Scheduling

- Priorities are recomputed once per second
- Base priority divides all processes into fixed bands of priority levels
- “Nice” adjustment factor used to keep process in its assigned band (cf. “To re-nice a process”)

# Feedback

- Process is demoted to the next lower-priority queue each time it returns to the ready queue
- Longer processes drift downward
- To avoid starvation, preemption time for lower-priority processes is longer

# References

- “Operating Systems - Internals and Design Principles” - William Stallings (Prentice Hall, 4th edition, 2000)