COS 318: Operating Systems

CPU Scheduling

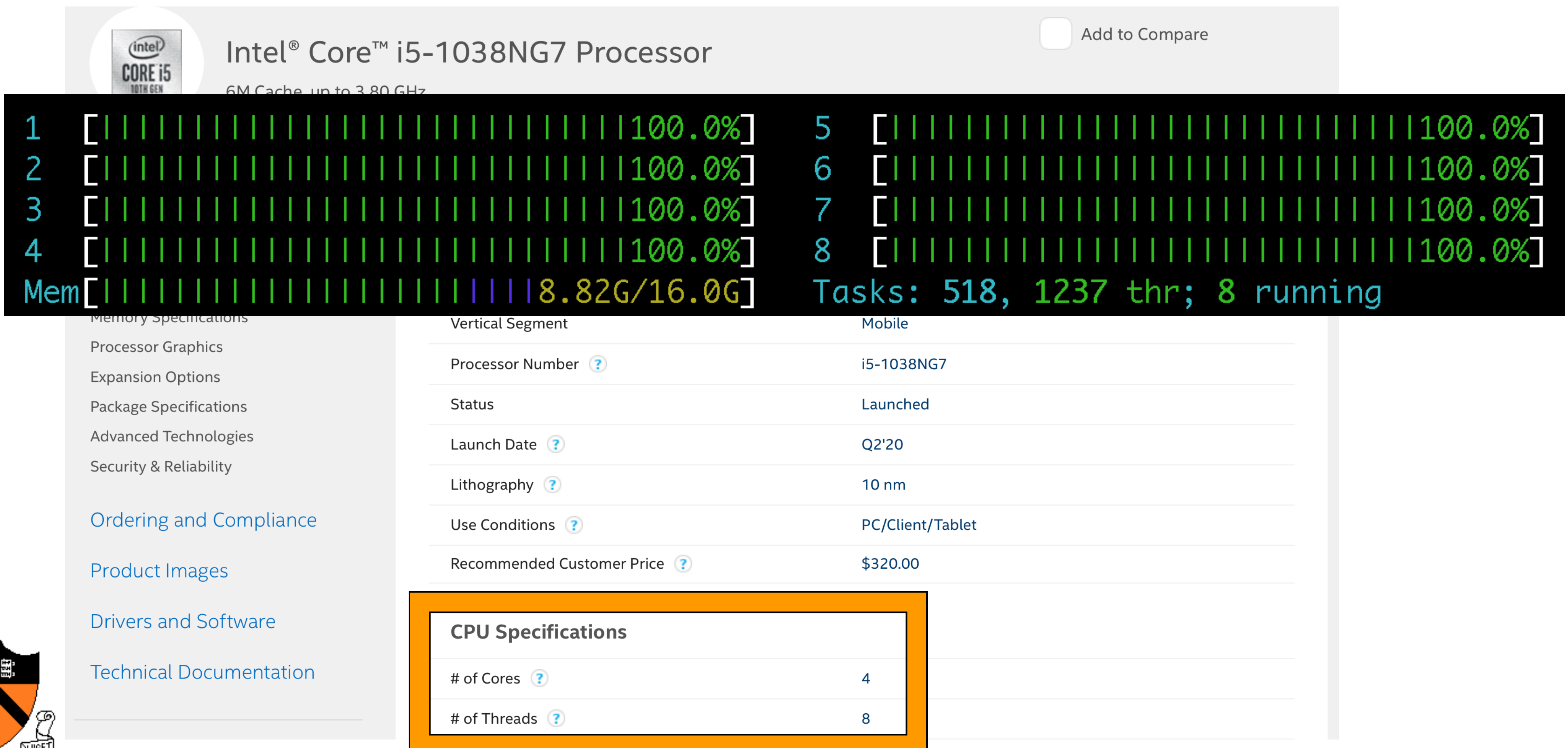
Today's Topics

- CPU scheduling basics
- CPU scheduling algorithms

Why schedule CPU?

- There can be a lot more <u>ready threads</u> than available <u>CPU hardware threads</u>.
- Let's check this by running **`htop`** in terminal:

A MacBook with a Quad-Core Intel Core i5-1038NG7 CPU

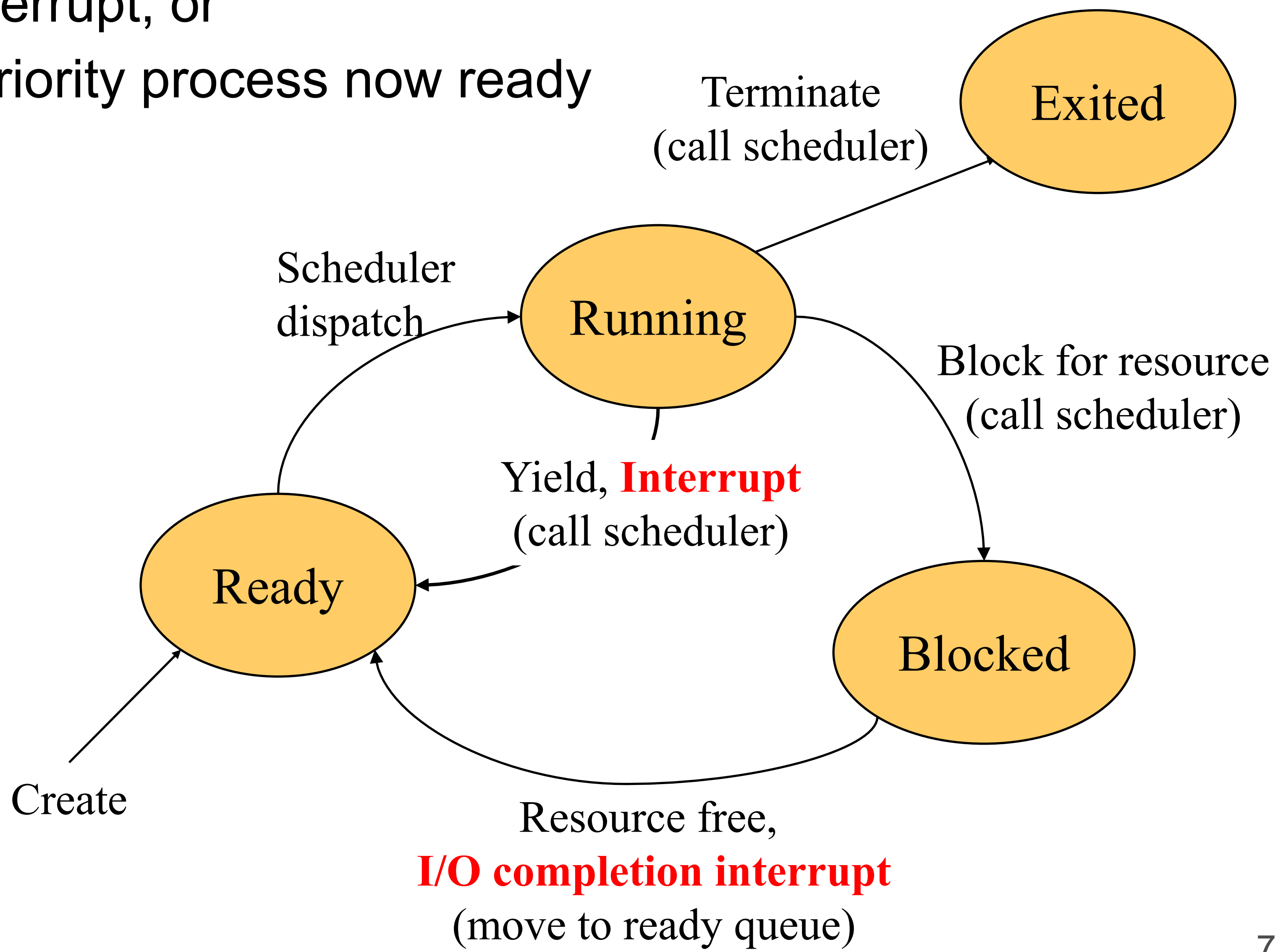


When to schedule?

1. New process created
   - `fork()` → child process created
   - Schedule parent or child (or both)
2. Process dies and returns exit status
   - Due to calling `exit()`, or fatal exception/signal
3. Blocked process
   - E.g. on I/O and semaphore
4. I/O interrupt
5. HW clock interrupt
   - E.g., with 250 Hz frequency
   - **Preemptive** scheduler uses this to replace running processes

Preemptive and Non-Preemptive Scheduling

- Preemption happens due to:
  1. Timer interrupt, or
  2. Higher priority process now ready



Scheduling Categories

Different ways to categorize:

- Non-preemptive (uncommon these days)
- Preemptive

Batch systems	→	throughput ↑	turnaround time ↓
Interactive systems	→	response time ↓	proportionality ✓
Real-time systems	→	meet deadlines ✓	predictability ✓

- Uniprocessor
- Multiprocessor

Our assumptions:

- Uniprocessor
- One process per user
- One thread per process
- Processes are independent

Scheduling Algorithms

- Simplified view of scheduling:
  - Save process state (to PCB)
  - Pick a process to run next
  - Dispatch process

First-Come-First-Serve (FCFS) Policy

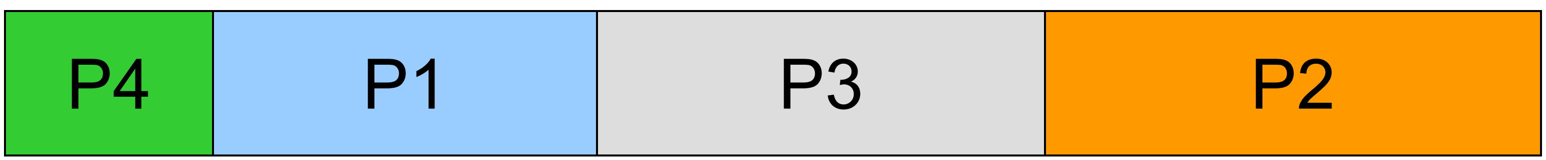
- Non-preemptive
- Schedule tasks in the order they arrive
  - Run them until completion, block, or yield
- Example 1
  - P1 = 24 sec, P2 = 3 sec, and P3 = 3 sec, submitted ‘same’ time in that order
  - Avg. response time: (24+27+30)/3 = 27s. Avg. wait time: (0+24+27)/3 = 17s

- Example 2
  - Same jobs but come in different order: P2, P3 and P1
  - Avg. response time: (3 + 6 + 30) / 3 = 13s. Avg wait time: (0+3+6)/3 = 3s

- FIFO pro: Simple. Con: Short jobs get stuck behind long ones

Shortest Job First (SJF) Policy

- Shortest Remaining Time to Completion First (SRTCF)
- Whenever scheduling decision is to be made, schedule process with shortest remaining time to completion
  - Non-preemptive case: straightforward
  - Preemptive case: if new process arrives with smaller remaining time, preempt running process and schedule new one
- Simple example: all arrive at same time:
  - P1 = 6sec, P2 = 8sec, P3 = 7sec, P4 = 3sec

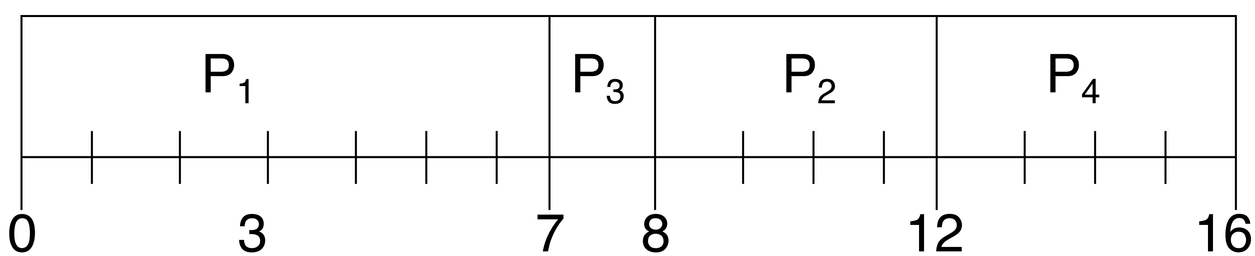


- SJF is the optimal policy to minimize average response time.

Example of non-preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

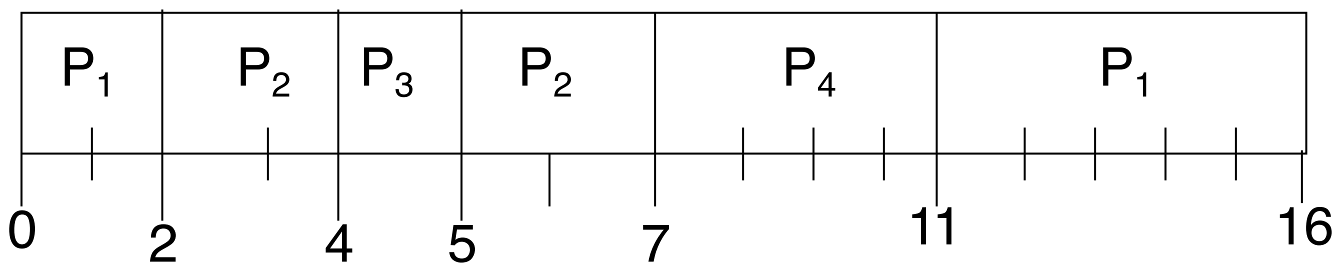


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)

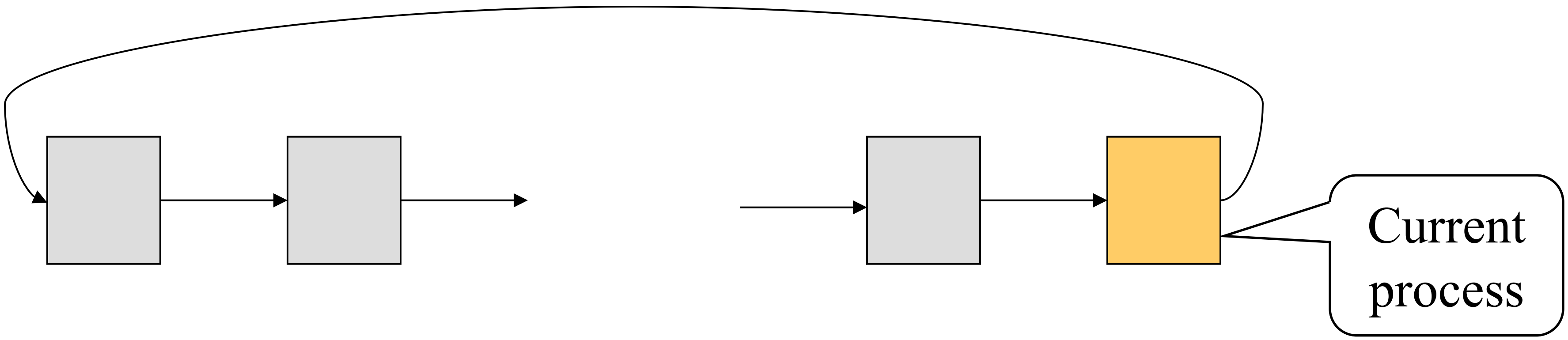


- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Evaluating SJF

- Q1: What might go wrong if you use a SJF policy to do your assignments?
  - The longer assignments might never be completed and deadlines would be missed.
- Q2: What practical limitation prevents using SJF?
  - It is not always feasible to know completion times in advance.

Round Robin Scheduling Policy



- Like FCFS, but with a time slice (quantum) for timer interrupt
  - Time-interrupted process is moved to end of queue
- Mitigates the starvation issue of SJF
- Real systems also have I/O interrupts in the mix
- How do you choose the time slice?

higher context switch overhead

can increase starvation

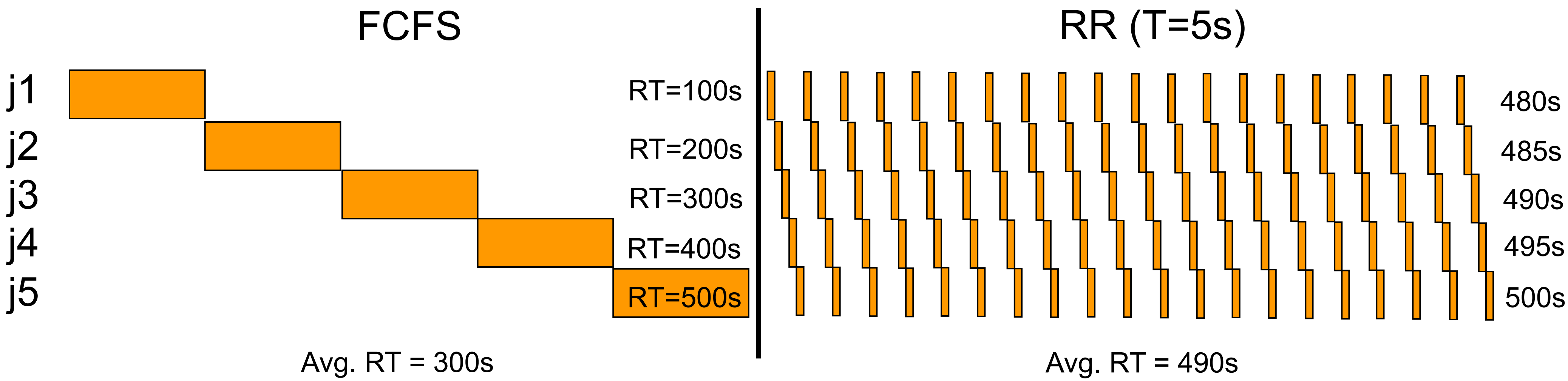
short quantum

Overhead kept at ≤ 1% typically

long quantum

FCFS vs. Round Robin

- 5 jobs, each taking 100 seconds, all coming at t=0s



- Comparisons
  - FCFS has less average response time and no task is worse off
  - 1) SJF result same as FCFS, 2) SJF is optimal
    → FCFS optimal here
  - But, e.g. for video streaming, RR is good, since everyone makes progress and gets a share “all the time”

Resource Utilization Example

- A, B, and C run forever (in this order)
  - A and B each uses 100% CPU forever [both **CPU-bound**]
  - C: CPU + I/O job (1ms CPU + 10ms disk I/O) [**I/O-bound**]
- RR with time slice 100ms:
  - A (100ms CPU), B (100ms CPU), C (1ms CPU + 10ms I/O)

CPU Util: 201/(201+3δ) ≈ 100%	**I/O Util: 10/(201+3δ) ≈ 5%**

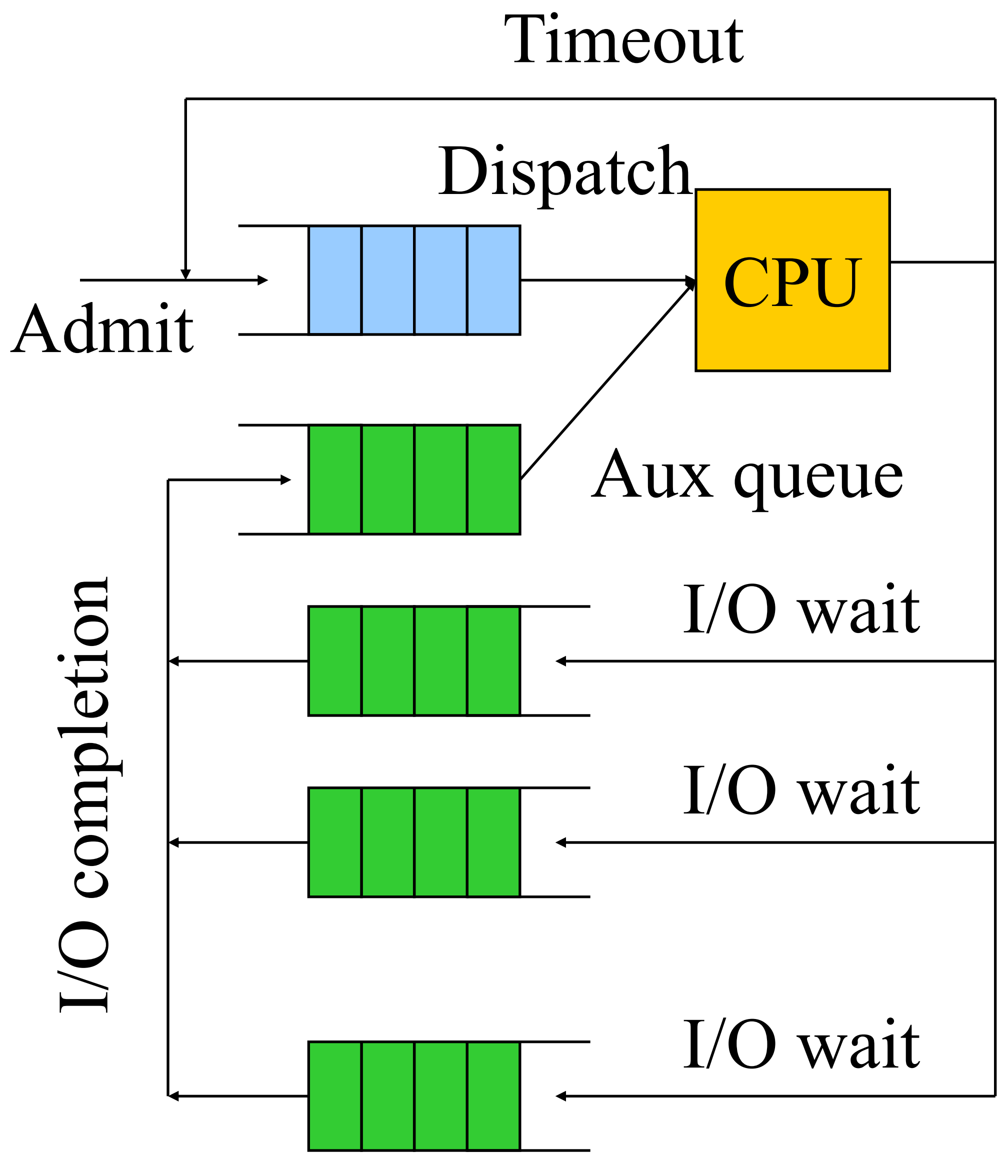
- RR with time slice 1ms:
  - A (1ms CPU), B (1ms CPU), C (1ms CPU), A (1ms CPU), B (1ms CPU), C(10ms I/O) || A, B, …, A, B

CPU Util: 15/(15+16δ) ≈ 100%	**I/O Util: 10/(15+16δ) ≈ 67%**

- What do we learn from this example?

Virtual Round Robin Policy

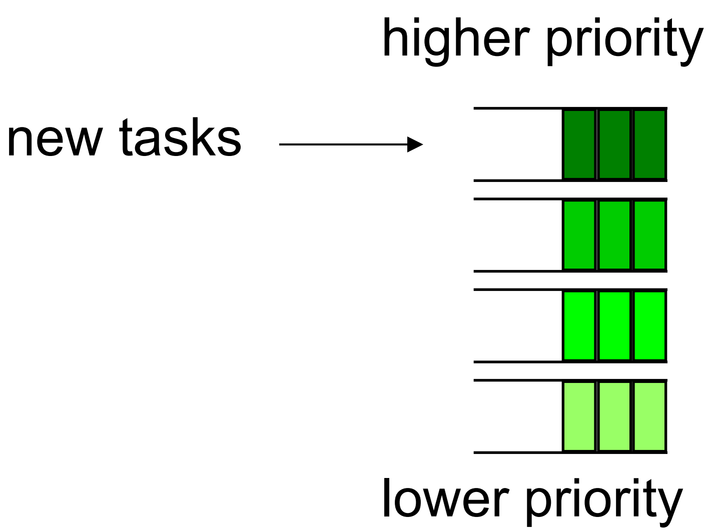
- I/O-bound processes go to auxiliary queue (instead of ready queue) to get scheduled
- Aux queue is FIFO
- Aux queue has preference over ready queue



Priority Scheduling Policy

- Not all processes are equal, so rank them
- The method
  - Assign each process a priority
  - Run the process with highest priority in the ready queue first
  - Adjust priority dynamically
    - **I/O wait raises the priority, reduce priority as process runs**
- Why adjusting priorities dynamically
  - T1 at priority 4, T2 at priority 1 and T2 holds lock L
  - Scenario
    - T1 tries to acquire L, fails, blocks.
    - T3 enters system at priority 3.
    - T2 never gets to run, and T1 is never unblocked

Multi-level Feedback Queues (MFQ)



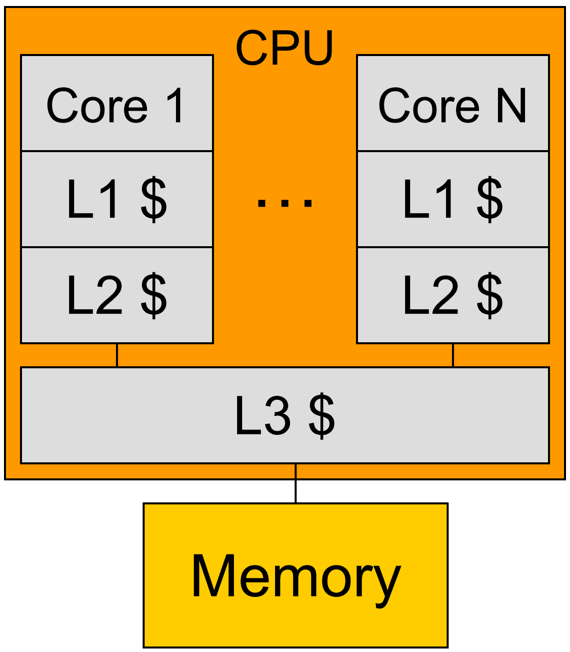
Priority	Time slices
4	1
3	2
2	4
1	8

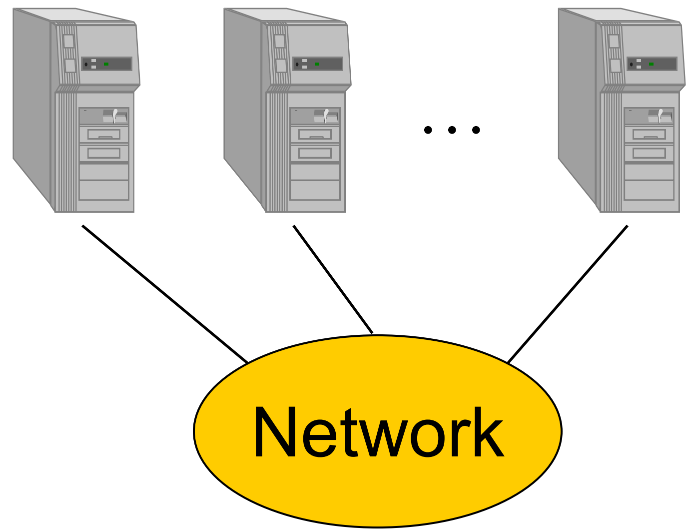
- Round-robin queues, each with different priority
- Higher priority queues have shorter time slices
- Jobs start at highest priority queue
- If timeout expires (needs more CPU), drop one level
- If timeout doesn’t expire (e.g., blocked), stay or pushup one level
- What does this method do?

Lottery Scheduling

- Motivations
  - SJF does well with average response time, but is unfair (long jobs can be starved)
  - Need a way to give everybody *some* chance of running
- Lottery method
  - Give each job a number of tickets
  - Randomly pick a winning ticket
  - To approximate SJF, give short jobs more tickets
  - To avoid starvation, give each job at least one ticket
  - Cooperative processes can exchange tickets

Multiprocessor and Cluster





Multiprocessor architecture

- Single OS
- Cache coherence

Cluster or multicomputer

- An OS in each box
- Distributed memory

Multiprocessor/Cluster Scheduling

- Design issue
  - Process/thread to processor assignment
- Gang scheduling (co-scheduling)
  - Threads of the same process will run together
  - Processes of the same application run together
- Dedicated processor assignment
  - Threads will be running on specific processors to completion
  - On a multiprocessor it is called **affinity** (or CPU pinning)
  - When is this a good idea?

Real-Time Scheduling

- Two types of real-time
  - Hard deadline
    - Must meet, otherwise can cause fatal error
  - Soft Deadline
    - Meet most of the time, but not mandatory
- Admission control
  - Take a real-time process only if the system can guarantee the "real-time" behavior of all processes.
  - Assume periodic processes. The jobs are schedulable, if the following holds:

$$\sum \frac{C_i}{T_i} \leq 1$$

where C_i = computation time, and T_i = period.

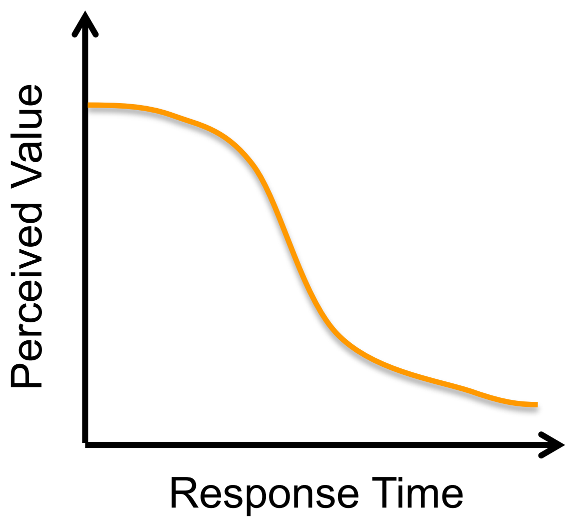
Rate Monotonic Scheduling (Liu & Layland 73)

- Assumptions
  - Each periodic process must complete within its period
  - No process is dependent on any other process
  - A process needs same amount of CPU time on each burst
  - Non-periodic processes have no deadlines
  - Process preemption occurs instantaneously (no overhead)
- Main ideas of RMS
  - Assign each process a fixed priority = frequency of occurrence
  - Run the ready process with highest priority
- Example
  - P1 runs every 30ms gets priority 33 (1s/30ms = 33 times/sec)
  - P2 runs every 50ms gets priority 20 (1s/50ms = 20 times/sec)

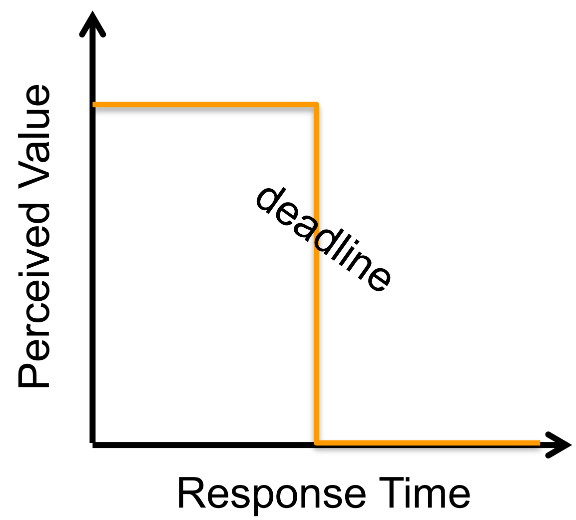
Earliest Deadline Scheduling

- Assumptions
  - When a process needs CPU time, it announces its deadline
  - No need to be periodic process
  - CPU time needed may vary
- Main idea of EDS
  - Sort ready processes by their deadlines
  - Run the first process on the list (earliest deadline first)
  - When a new process is ready, it preempts the current one if its deadline is closer
- Example
  - P1 needs to finish by 30sec, P2 by 40sec and P3 by 50sec
  - P1 goes first
  - More in MOS 7.4.4

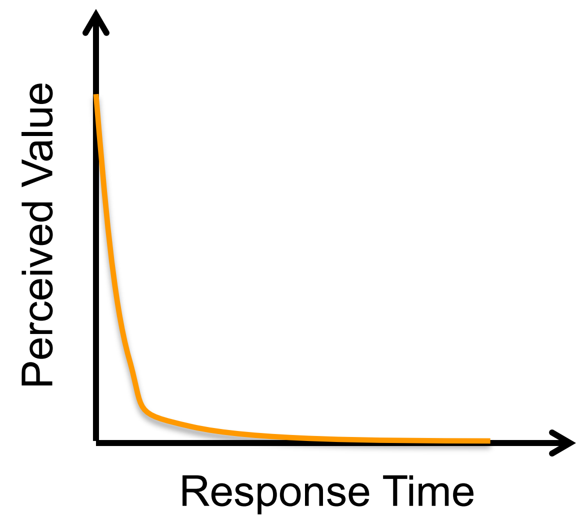
Perceived Value vs. Response Time



General Purpose System



Real-Time System



High frequency Trading

Summary

- Best algorithms may depend on your primary goals
  - FIFO simple, optimal avg response time for tasks of equal size, but can be poor avg response time if tasks vary a lot in size
  - SJF gives the minimal average response time, but can be not great in variance of response times
  - RR has very poor avg response time for equal size tasks, but is close to SJF for variable size tasks
  - Small time slice is important for improving I/O utilization
  - If tasks have mix of processing and I/O, do well under SJF but can do poorly under RR
  - Priority and its variations are used in most systems
  - Lottery scheduling is flexible
  - Multi-queue can achieve a good balance
  - Admission control is important in real-time scheduling