CPU Scheduling

Jo, Heeseung

Today's Topics

General scheduling concepts

Scheduling algorithms

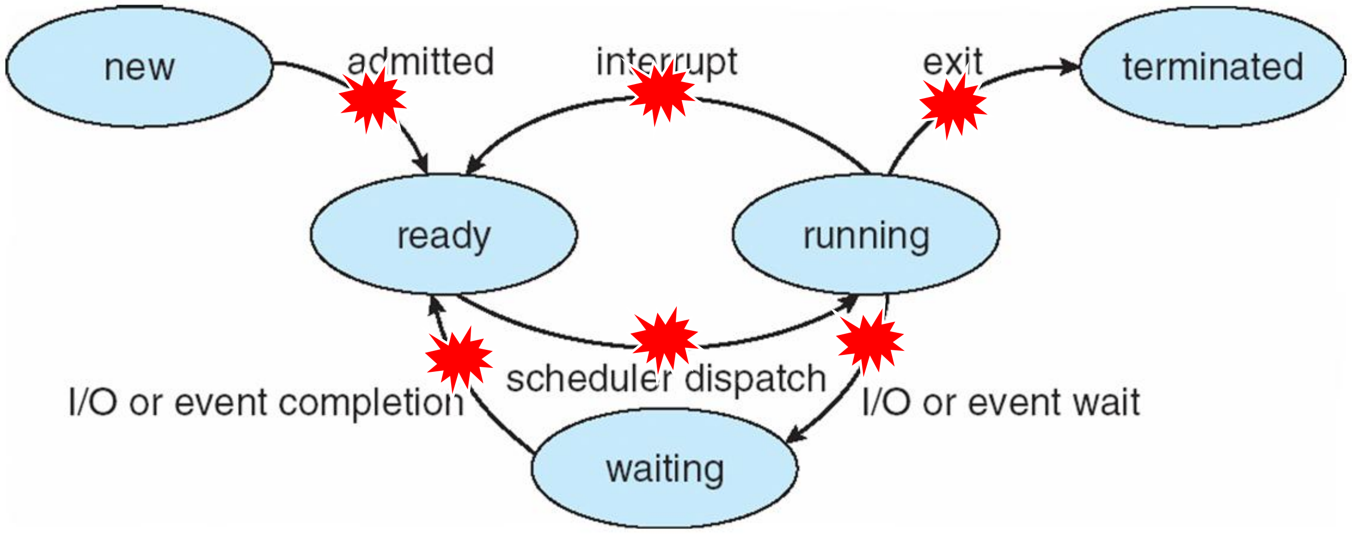
Case studies

CPU Scheduling (1)

CPU scheduling

- Deciding which process to run next, given a set of runnable processes
- Happens frequently, hence should be fast

Scheduling points



CPU Scheduling (2)

Scheduling algorithm goals

- All systems
  - No starvation
  - Fairness: giving each process a fair share of the CPU
  - Balance: keeping all parts of the system busy
- Batch systems
  - Throughput: maximize jobs per hour
  - Turnaround time: minimize time between submission and termination
  - CPU utilization: keep the CPU busy all the time
- Interactive systems
  - Response time: respond to requests quickly
- Real-time systems
  - Meeting deadlines: avoid losing data
  - Predictability: avoid quality degradation in multimedia system

CPU Scheduling (3)

Starvation

- A situation where a process is prevented from making progress because another process has the resource it requires
  - Resource could be the CPU or a lock
- A poor scheduling policy can cause starvation
  - If a high-priority process always prevents a low-priority process from running on the CPU
- Synchronization can also cause starvation
  - One thread always beats another when acquiring a lock

CPU Scheduling (4)

Non-preemptive scheduling – 뺏지 않고 기다림.

- The scheduler waits for the running job to voluntarily yield the CPU
- Jobs should be cooperative

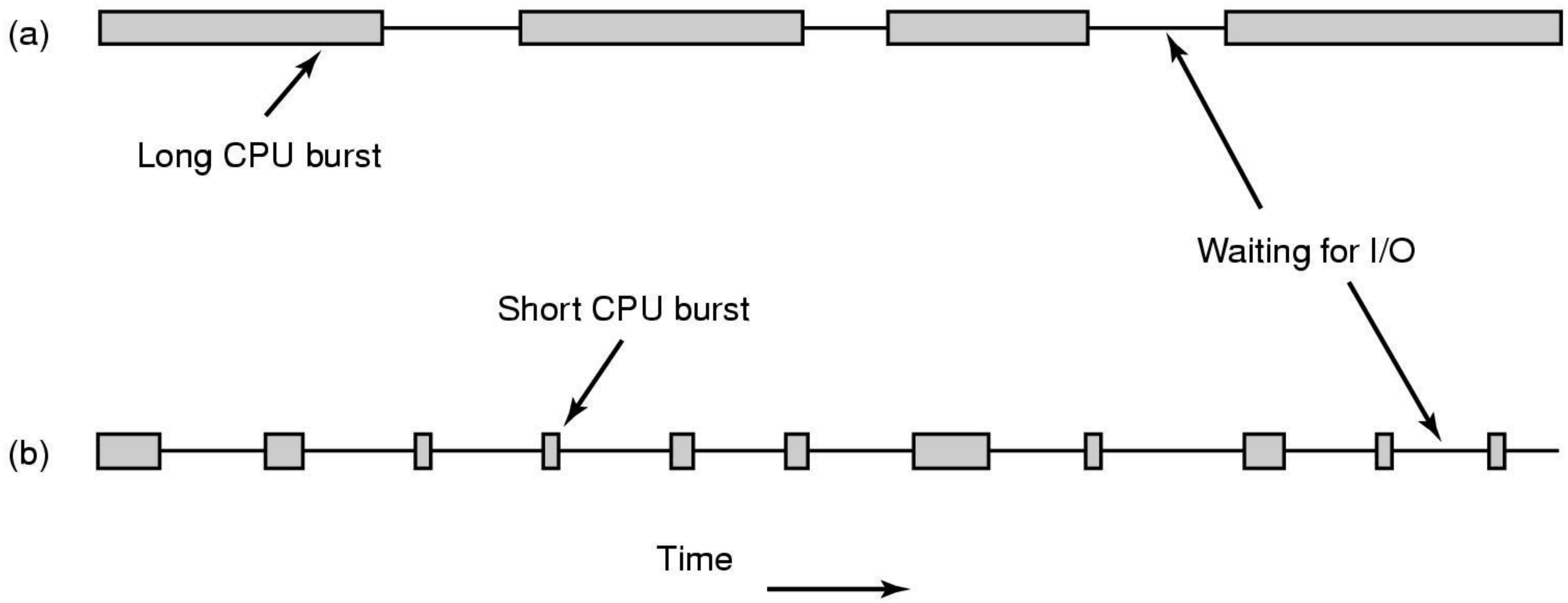
Preemptive scheduling – CPU 강제로 뺏는 것.

- The scheduler can interrupt a job and force a context switch
- What happens
  - If a process is preempted in the midst of updating the shared data?
  - If a process in a system call is preempted?

Execution Characteristics (1)

CPU burst vs. I/O burst

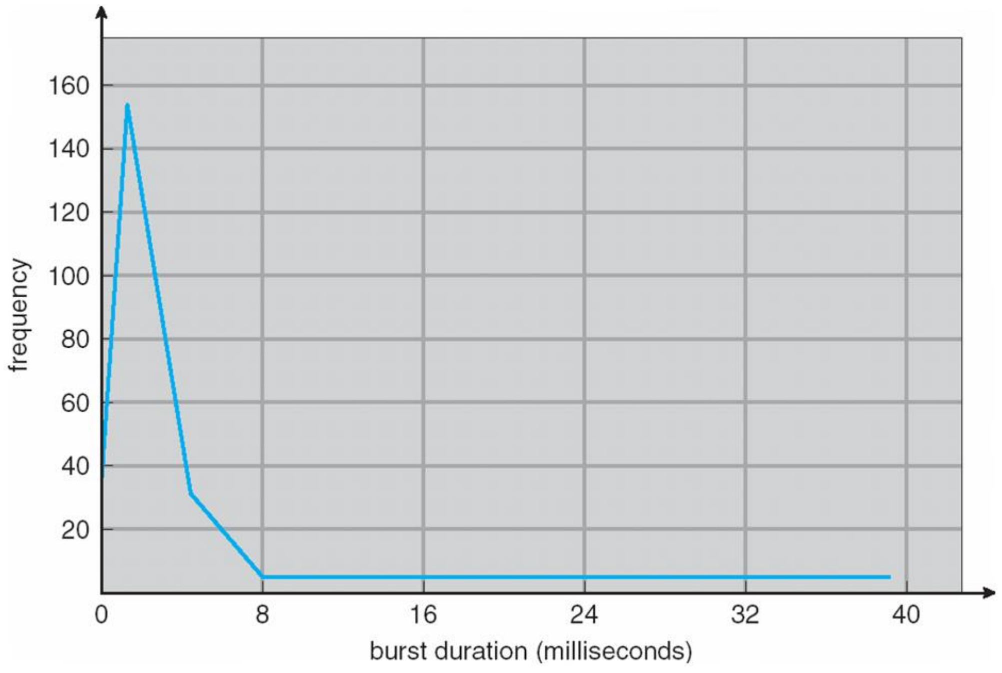
- A CPU-bound process
- An I/O-bound process



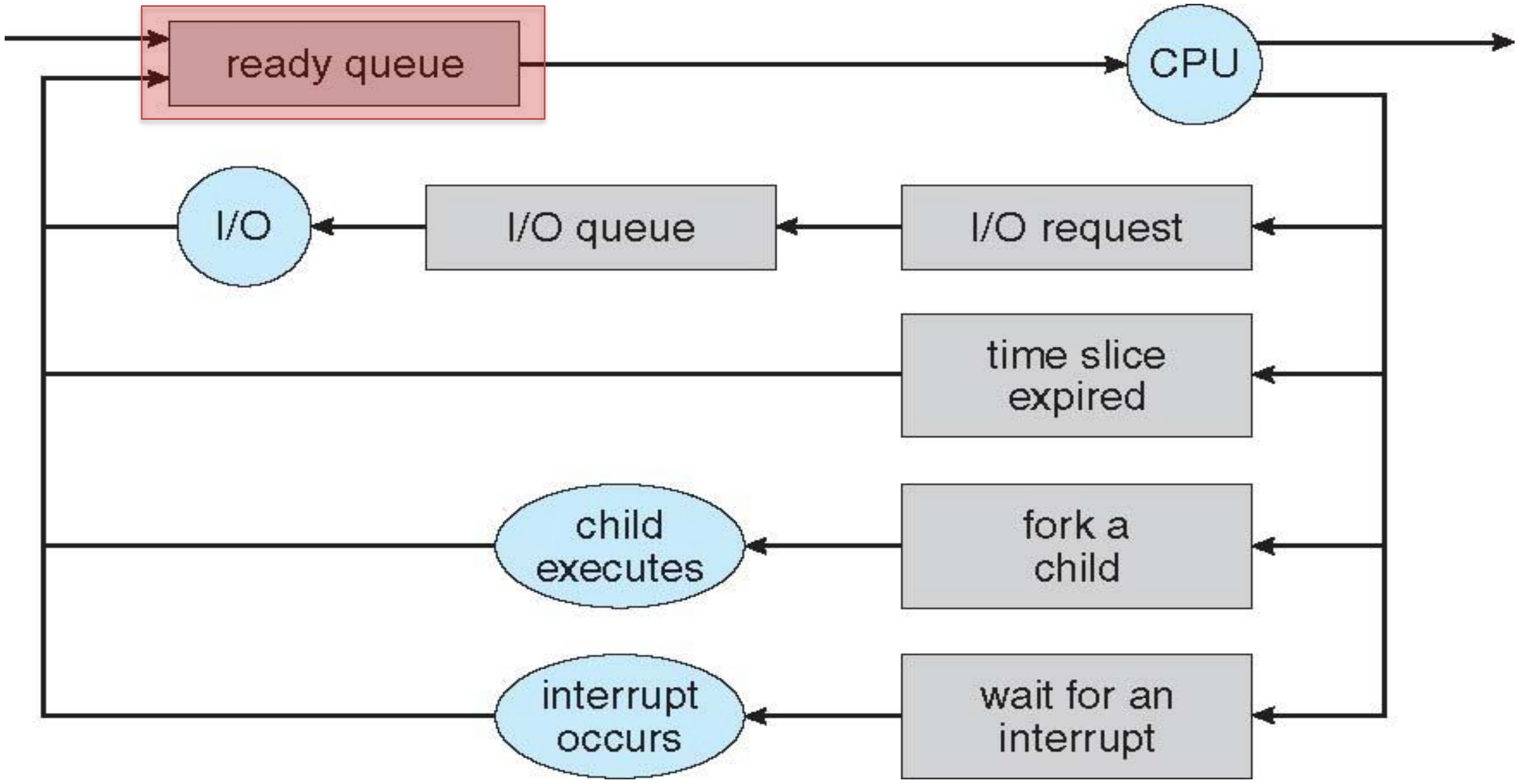
Execution Characteristics (2)

Histogram of CPU-burst Times

- Most are short CPU burst
- Rarely long CPU burst
- Reference for CPU scheduling algorithm design



Process State Queues



FCFS/FIFO

First-Come, First-Served / First-In, First-Out

- Jobs are scheduled in order that they arrive
- "Real-world" scheduling of people in lines
  - e.g., supermarket, bank tellers, McDonalds, etc.
- Typically, non-preemptive
- Jobs are treated equally: no starvation

Problems

- Average waiting time can be large if small jobs wait behind long ones
  - Basket vs. cart
- May lead to poor overlap of I/O and CPU

FCFS/FIFO

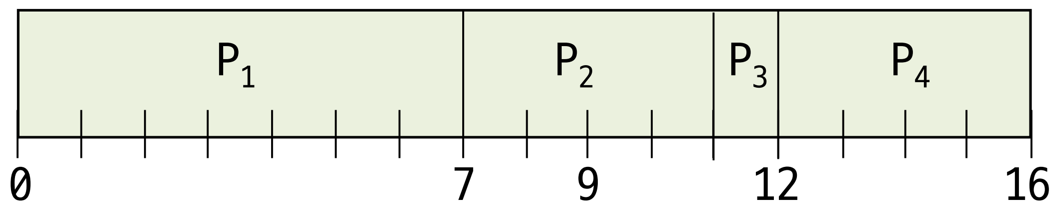
First-Come, First-Served / First-In, First-Out

큐에 도착한 시간.

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

Ready Queue

FIFO



Non-preemptive

SJF

Shortest Job First

CPU를 가장 짧게 쓸 애한테 할당해주자.

- Choose the job with the smallest expected CPU burst
- Can prove that SJF has optimal min. average waiting time
  - Only when all jobs are available simultaneously
- Non-preemptive

Problems

- Impossible to know the size of future CPU burst
- Can you make a reasonable guess?
- Can potentially starve

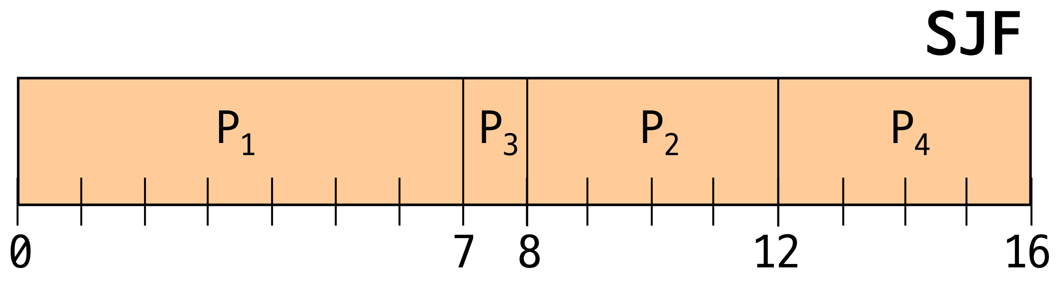
Starvation 발생 가능.

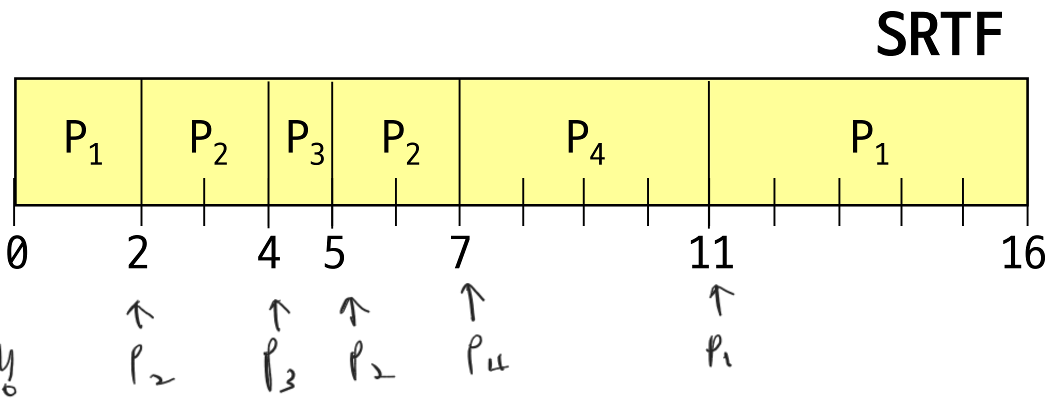
SRTF

Shortest Remaining Time First

- Preemptive version of SJF → 의도적으로 뺏을 수 있음.
- If a new process arrives, rethink preemption
  - With CPU burst length less than remaining time of current executing process, preempt

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





누굴 줄지 계속 생각하면

→ Context Switch (overhead) 다수 발생

→ 시스템 느려짐.

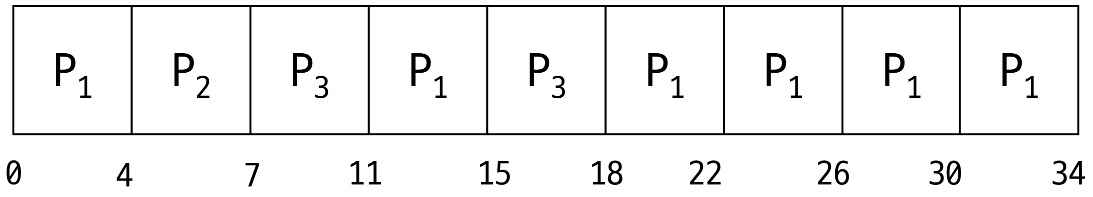
RR

Round Robin

- Ready Q is treated as a circular FIFO Q
- Each job is given a time slice (or time quantum)
  - Usually 10-100 ms
- Great for timesharing
  - No starvation
  - Typically, higher average turnaround time than SJF, but better response time
- Preemptive
- What do you set the quantum to be?
  - A rule of thumb: 80% of the CPU bursts should be shorter than the time quantum
  - Longer quantum : Higher throughput
  - Shorter quantum : Shorter response
- Treats all jobs equally

Example of RR with Time Quantum = 4

Process	Arrival Time	Burst
P_1	0.0	24
P_2	1.0	3
P_3	2.0	7

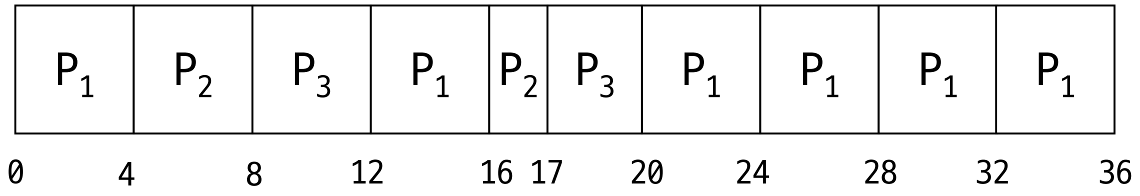


$\downarrow$

$P_1 = 20$

Example of RR with Time Quantum = 4

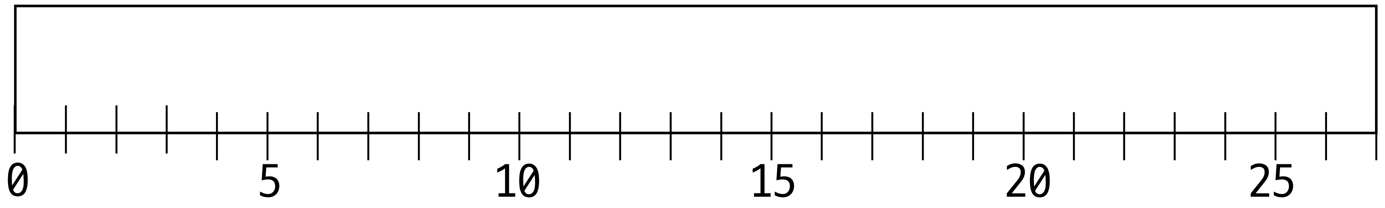
Process	Arrival Time	Burst
P_1	0.0	24
P_2	1.0	5
P_3	2.0	7



Exercise

FCFS

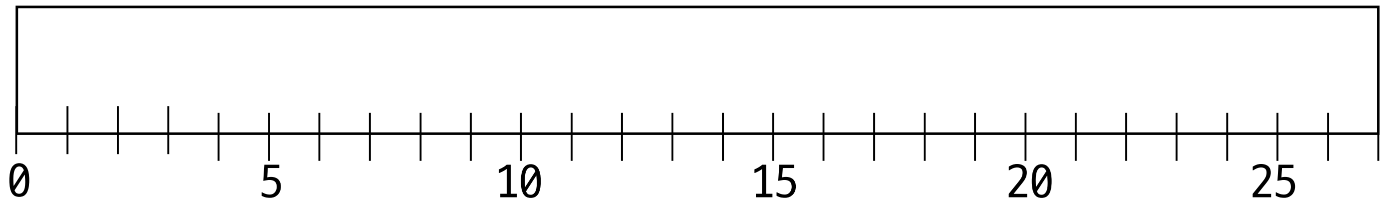
Process	Arrival Time	Burst
P_1	0.0	3
P_2	1.0	5
P_3	2.0	7
P_4	5.0	6
P_5	6.0	3



Exercise

SJF

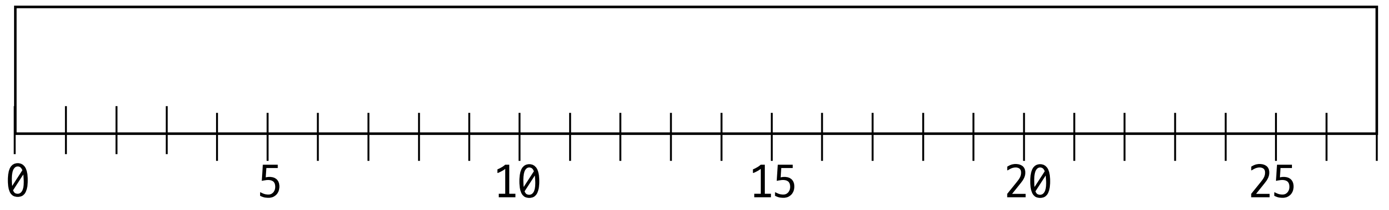
Process	Arrival Time	Burst
P_1	0.0	3
P_2	1.0	5
P_3	2.0	7
P_4	5.0	6
P_5	6.0	3



Exercise

SRTF

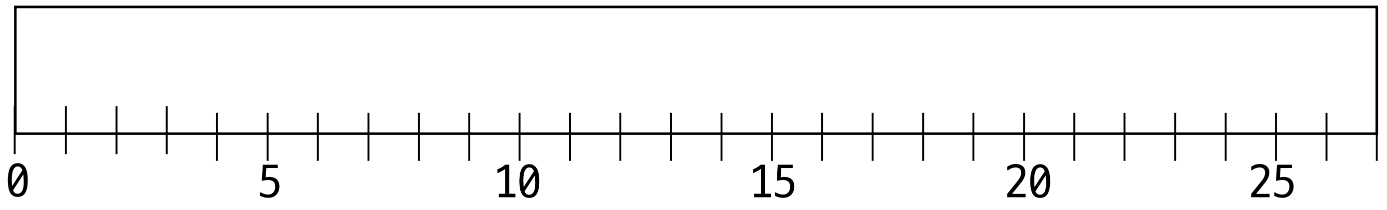
Process	Arrival Time	Burst
P_1	0.0	3
P_2	1.0	5
P_3	2.0	7
P_4	5.0	6
P_5	6.0	3



Exercise

RR (Q = 4)

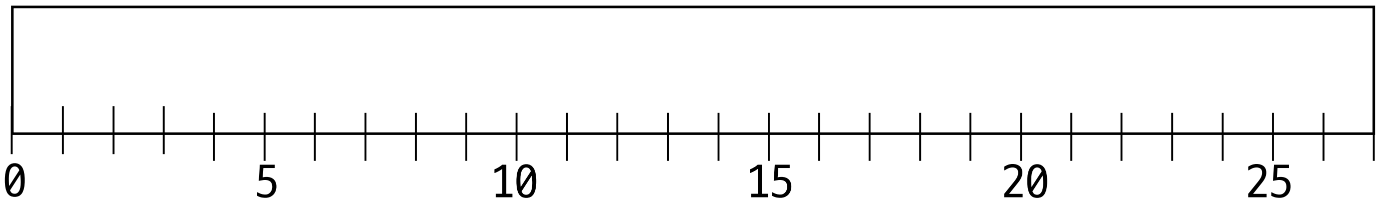
Process	Arrival Time	Burst
P_1	0.0	3
P_2	1.0	5
P_3	2.0	7
P_4	5.0	6
P_5	6.0	3



Exercise

RR (Q = 5)

Process	Arrival Time	Burst
P_1	0.0	3
P_2	1.0	5
P_3	2.0	7
P_4	5.0	6
P_5	6.0	3



Priority Scheduling (1)

Priority scheduling

- Choose job with highest priority to run next
- SJF = Priority scheduling, where
  priority = expected length of CPU burst
- Round-robin or FIFO within the same priority
- Can be either preemptive or non-preemptive
- Priority is dynamically adjusted
- Modeled as a Multi-level Feedback Queue (MLFQ)

Priority Scheduling (2)

Starvation problem 할당을 못받음 → 굶주림, 기아

- If there is an endless supply of high priority jobs, no low priority job will ever run

Solution: Aging

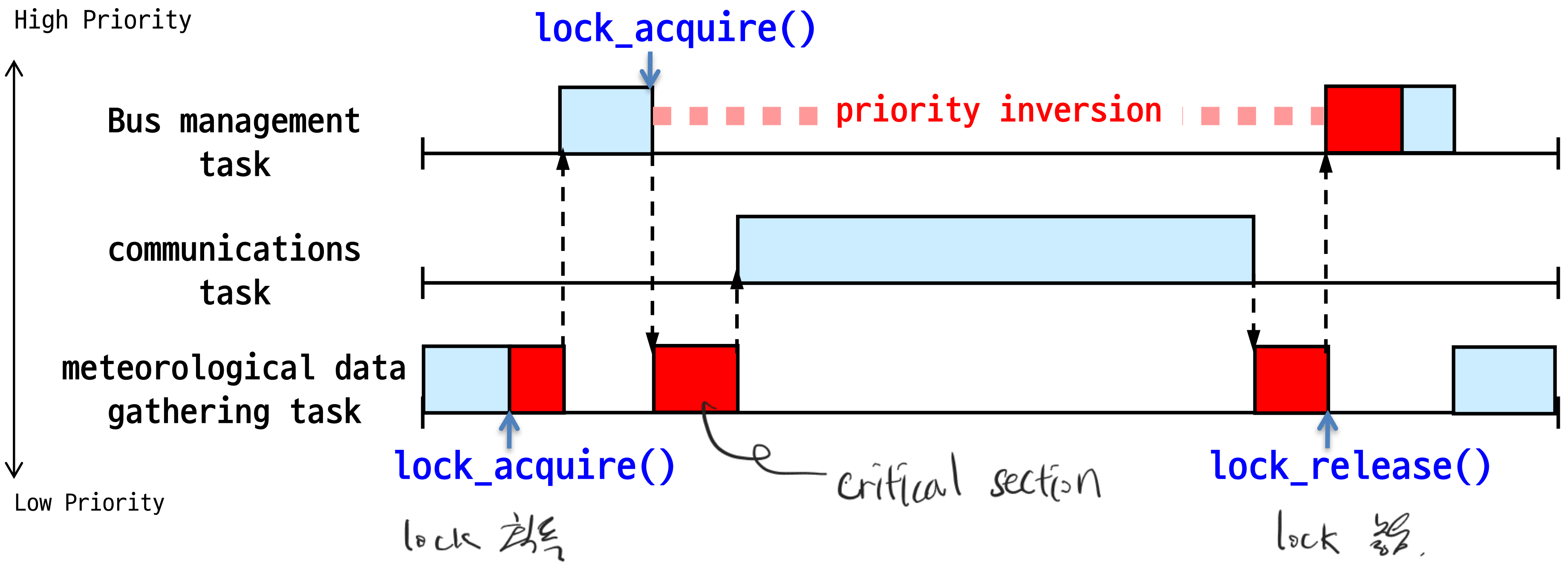
- Increase priority as a function of wait time
- Decrease priority as a function of CPU time
- Many ugly heuristics have been explored in this area

Priority Scheduling (3)

Pathfinder, 1997

Priority inversion problem

- A situation where a higher-priority job is unable to run because a lower-priority job is holding a resource it needs, such as a lock



- What really happened on Mars? - google search

우선순위가 낮은 Task가 리소스를 점유하고 있어
우선순위가 높은 Task가 리소스를 사용하지 못하는 문제

Priority Scheduling (4)

Priority inheritance protocol (PIP)

- The higher-priority job can donate its priority to the lower-priority job holding the resource it requires

우선순위 기부

Priority ceiling protocol (PCP)

- The priority of the low-priority thread is raised immediately when it gets the resource
- The priority ceiling value must be predetermined

lock: 공유 리소스를 한 스레드가 점유(사용) 하고 있을 때 다른 스레드가 점유하지 못하도록 잠그는 것.

lock_acquire(), lock_release().

Multilevel Queue Scheduling

Ready queue is partitioned into separate queues, eg:

- foreground (interactive)
- background (batch)

Process permanently in a given queue

Each queue has its own scheduling algorithm:

- foreground – RR
- background – FCFS

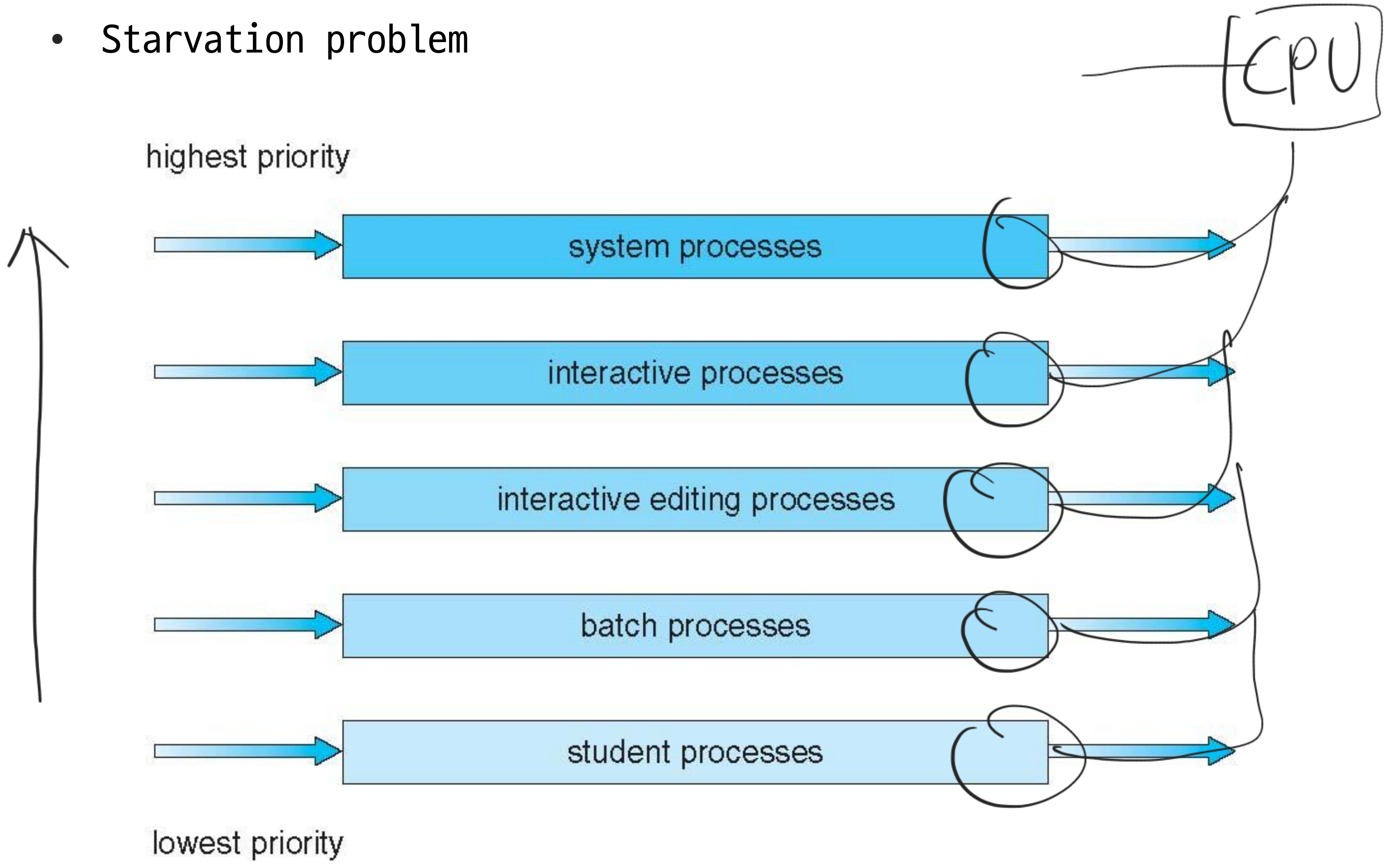
Scheduling must be done between the queues:

- Fixed priority scheduling
- i.e., serve all from foreground then from background
- Possibility of starvation

Multilevel Queue Scheduling

Process permanently in a given queue

- Starvation problem



Multilevel Feedback Queue

Multilevel Feedback Queue

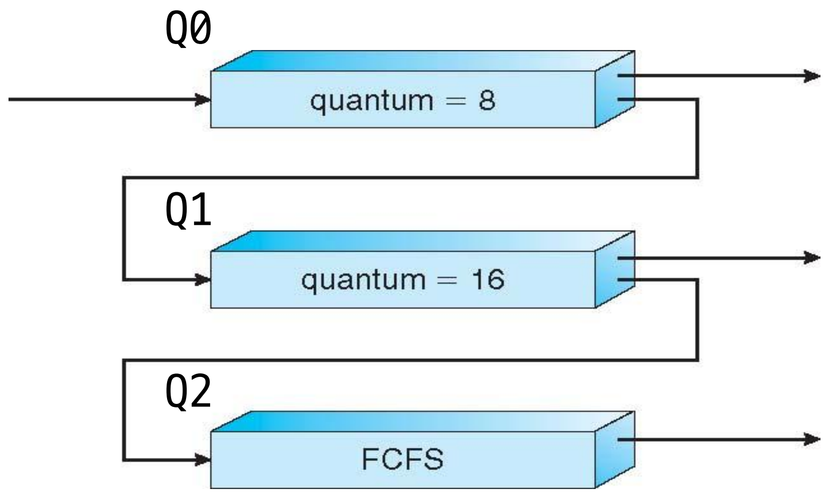
- Multilevel feedback queue scheduling, which allows a job to move between the various queues
- Queues have priorities
- When a process uses too much CPU time, move to a lower-priority queue
  - Aging
  - Leaves I/O-bound and interactive processes in the higher-priority queues
- When a process waits too long in a lower priority queue, move to a higher-priority queue
  - Prevents starvation

큐와 큐 사이를 왔다갔다 함

Example of Multilevel Feedback Queue

Three queues:

- Q0 – RR with time quantum 8 milliseconds
- Q1 – RR time quantum 16 milliseconds
- Q2 – FCFS



Scheduling

- A new job enters queue Q0 which is served FCFS
  - When it gains CPU, job receives 8 milliseconds
  - If it does not finish in 8 milliseconds, job is moved to queue Q1
- At Q1 job is again served FCFS and receives 16 additional milliseconds
  - If it still does not complete, it is preempted and moved to queue Q2

UNIX Scheduler (1)

Characteristics

- Preemptive – OS가 뺏을 일이 생기면 뺏고 생각.
- Priority-based
  - The process with the highest priority always runs
  - 170 priority levels (Solaris 2)
  - 0 – 39 priority levels (Linux) – nice 명령어 존재.
- Time-shared (based on RR)
  - Based on timeslice (or quantum)
- MLFQ (Multi-Level Feedback Queue)
  - Priority scheduling across queues, RR within a queue
  - Processes dynamically change priority

UNIX Scheduler (2)

General principles

CPU를 조금 씀 → 어차피 CPU를 내놓을 거다.

- Favor I/O-bound processes over CPU-bound processes
  - I/O-bound processes typically run using short CPU bursts
  - Provide good interactive response
    - Don't want editor to wait until CPU hog finishes quantum
  - CPU-bound processes should not be severely affected
- No starvation
  - Use aging

어차피 background 작업이다.

Multiple-Processor Scheduling

동일한 프로세서

CPU scheduling more complex when multiple CPUs are available

Homogeneous processors within a multiprocessor

Asymmetric multiprocessing

- Only one processor accesses the system data structures, alleviating the need for data sharing

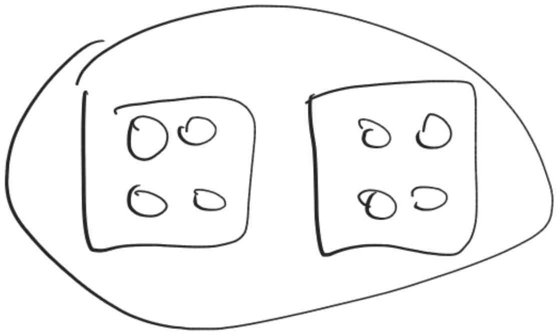
Symmetric multiprocessing (SMP)

- Each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
- Currently, most common

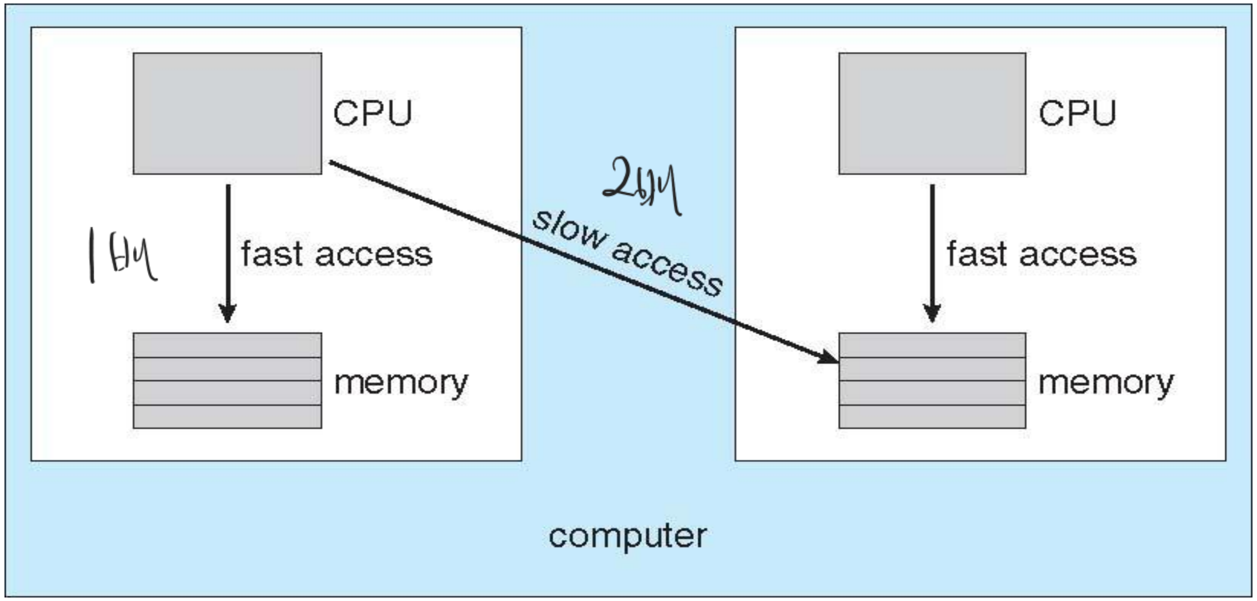
Processor affinity = 친화도, 유사도 등.

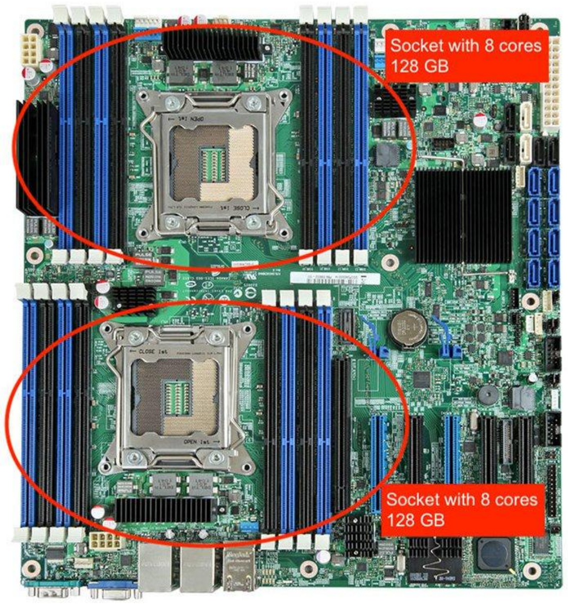
- Process has affinity for processor on which it is currently running
- Soft affinity – 선호.
- Hard affinity – 무조건 이 범위에서 돌아야 됨.

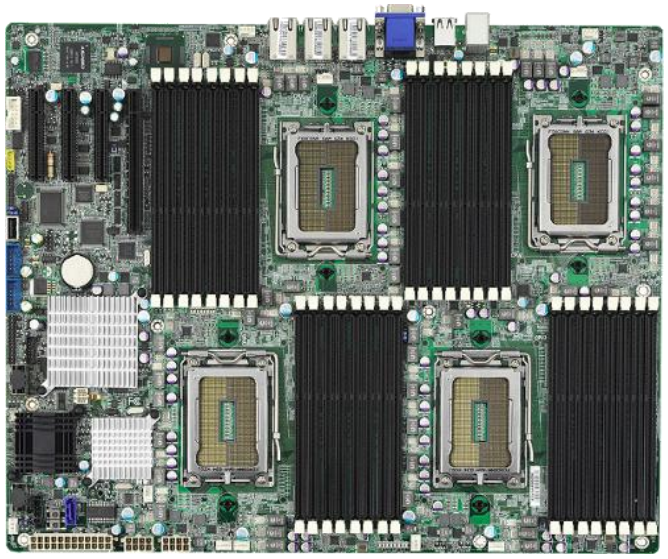
Symmetric multiprocessing)



NUMA and CPU Scheduling







Note that memory-placement algorithms can also consider affinity

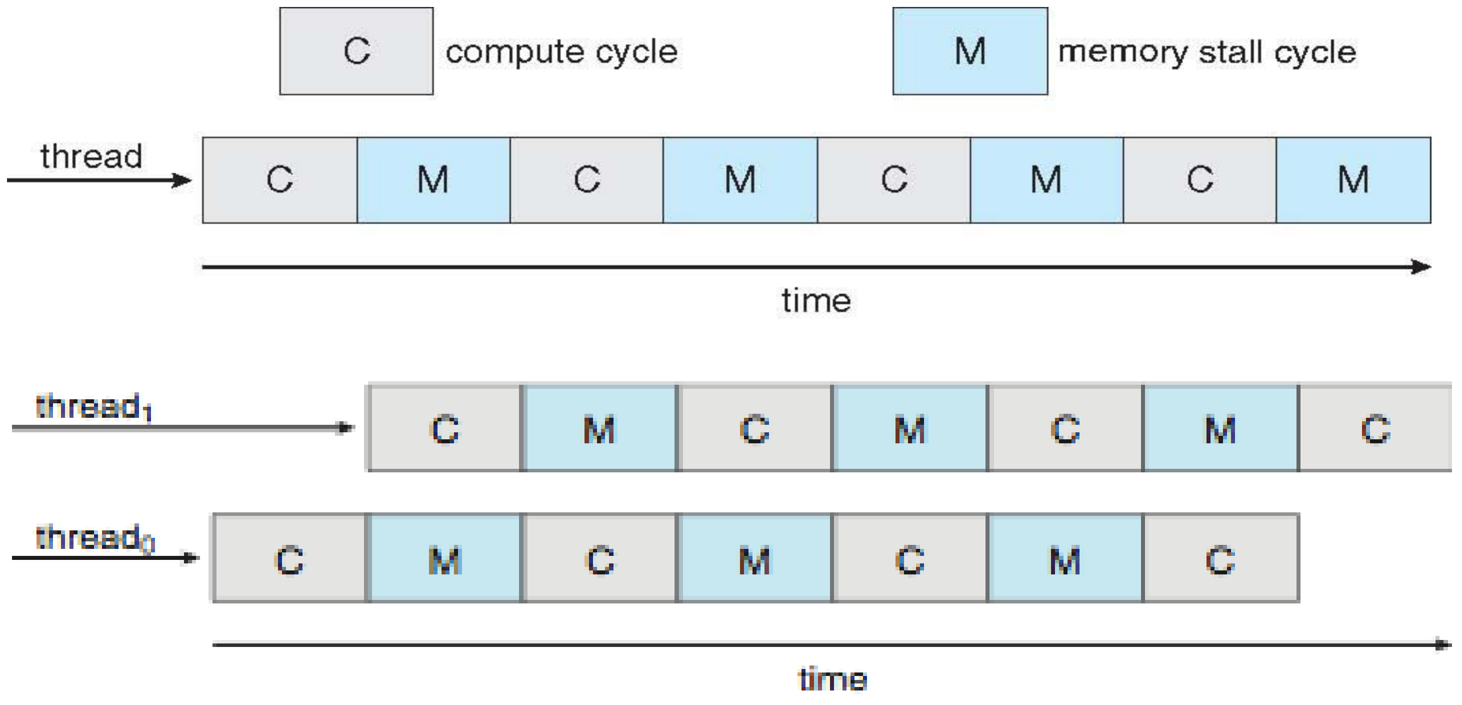
Multithreaded Multicore System (Hyperthreading)

Recent trend to place multiple processor cores on same physical chip

- Faster and consumes less power

Multiple H/W threads per core also growing (hyperthreading)

- Takes advantage of memory stall to make progress on another thread while memory retrieve happens
- Hardware level multithreading


C
compute cycle
M
memory stall cycle
thread
C M C M C M C M
time
thread₁
C M C M C M C
thread₀
C M C M C M C
time