Operating Systems – Scheduling

ECE 344 – Week 9

Scheduling

- **Task**: Determine which process is allowed to run
- *What are the objectives?*
  - Maximize
    - CPU utilization
    - Throughput (tasks per unit of time)
  - Minimize
    - Turnaround time (submission-to-completion)
    - Waiting time (sum of times spend in ready queue)
    - Response time (production of first response)
  - Fairness
    - Every task should be handled eventually (no starvation)
    - Tasks with similar characteristics should be treated equally
- *Who are the stake holders?* (owner, user, system)

Systems

- Batch systems
- Interactive systems
- Real-time systems

- Desktops
- Servers

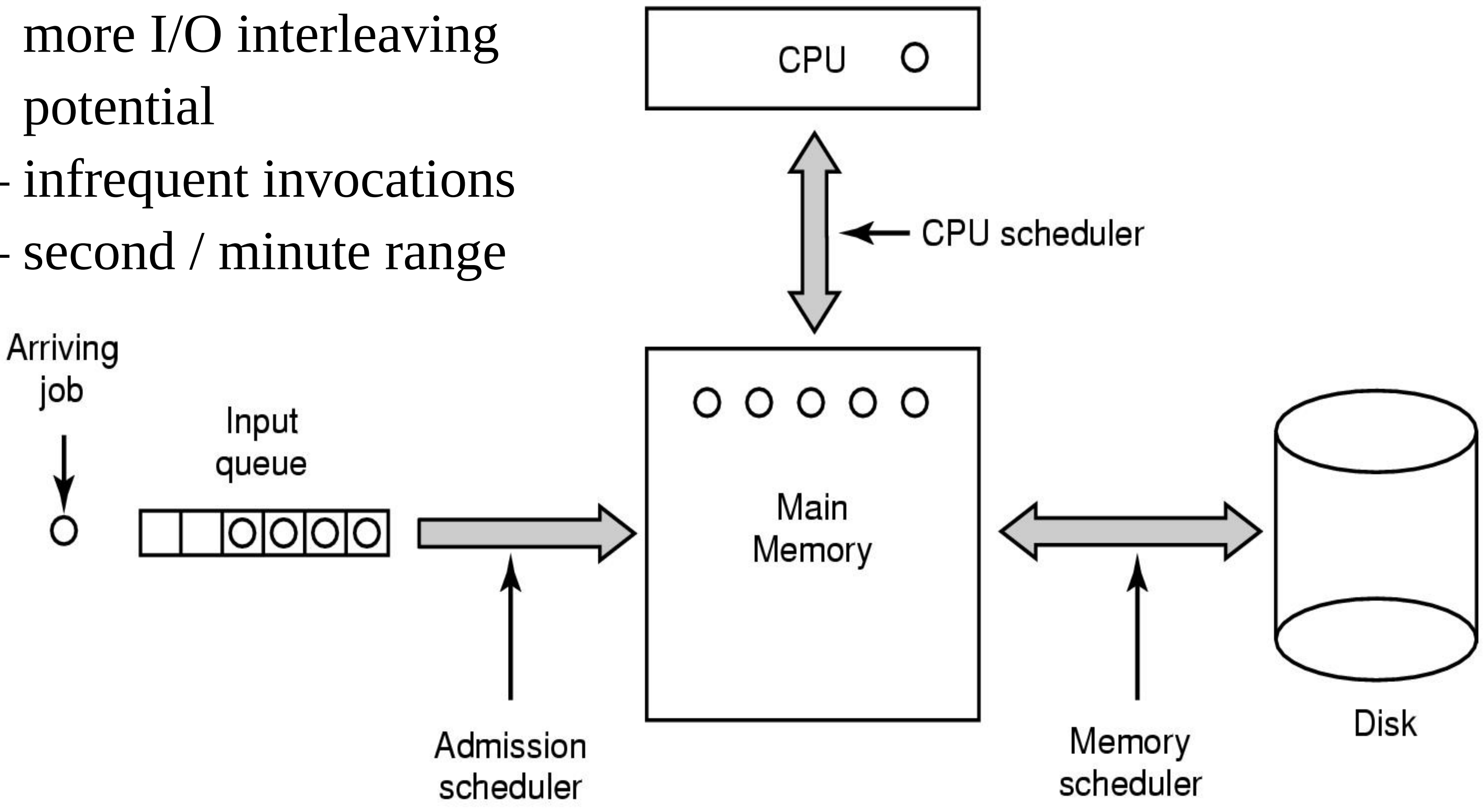
Types of Scheduling

- **Long-term** (admission scheduler, job scheduler)
  - Decision to admit a process to system (into the ready queue)
  - Controls degree of multiprogramming
  - Batch systems
- **Medium-term** (memory scheduler)
  - Decision to put process image on disk vs. keep in memory
  - Part of swapping mechanism
  - Need to manage and control the degree of multiprogramming
- **Short-term** (CPU scheduler)
  - Decision which of the ready processes to execute next
  - Executes most frequently, executes when an event occurs

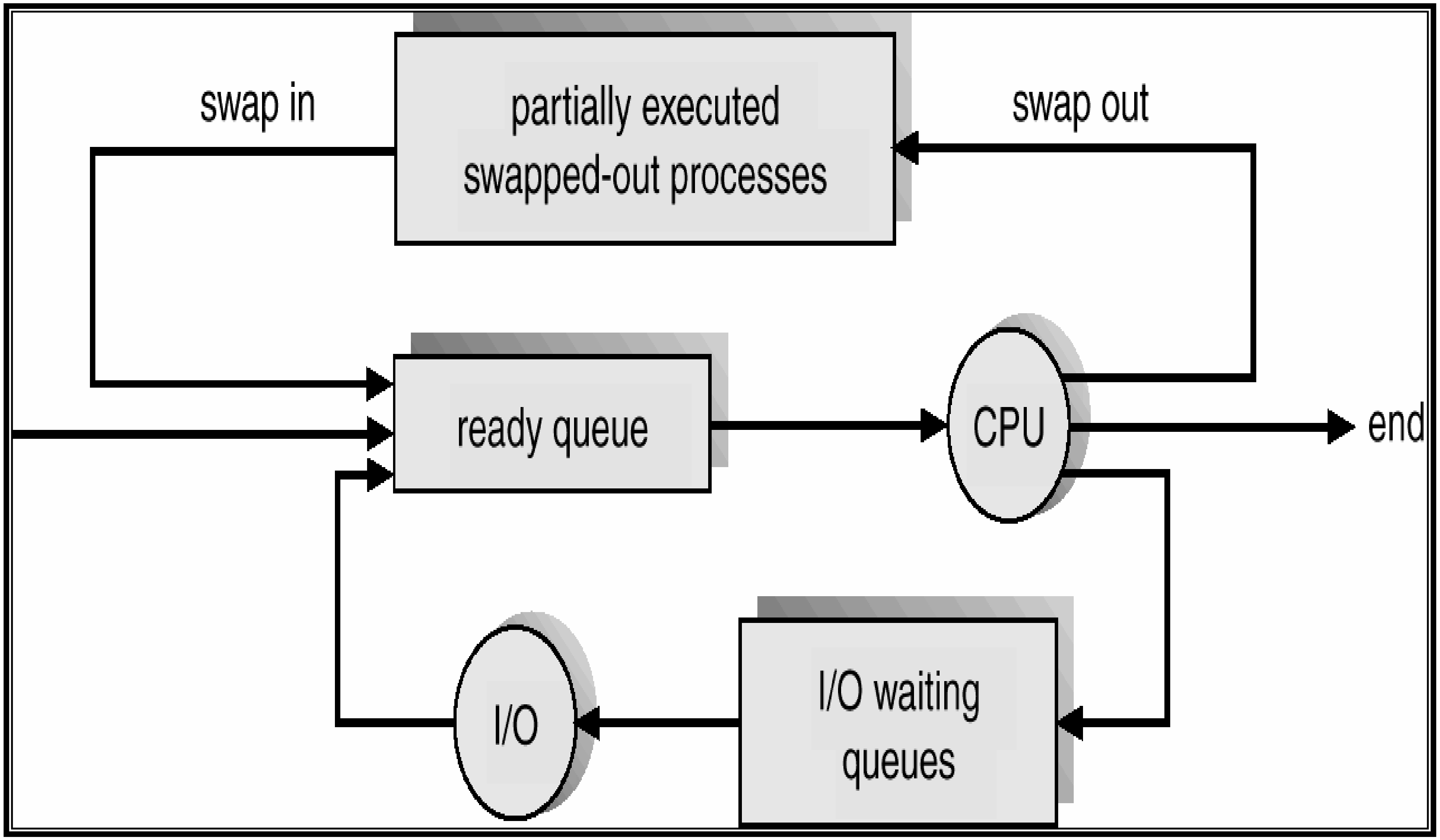
Scheduling

– more processes,
less CPU time,
more I/O interleaving
potential
– infrequent invocations
– second / minute range

– must operate fast
– millisecond range



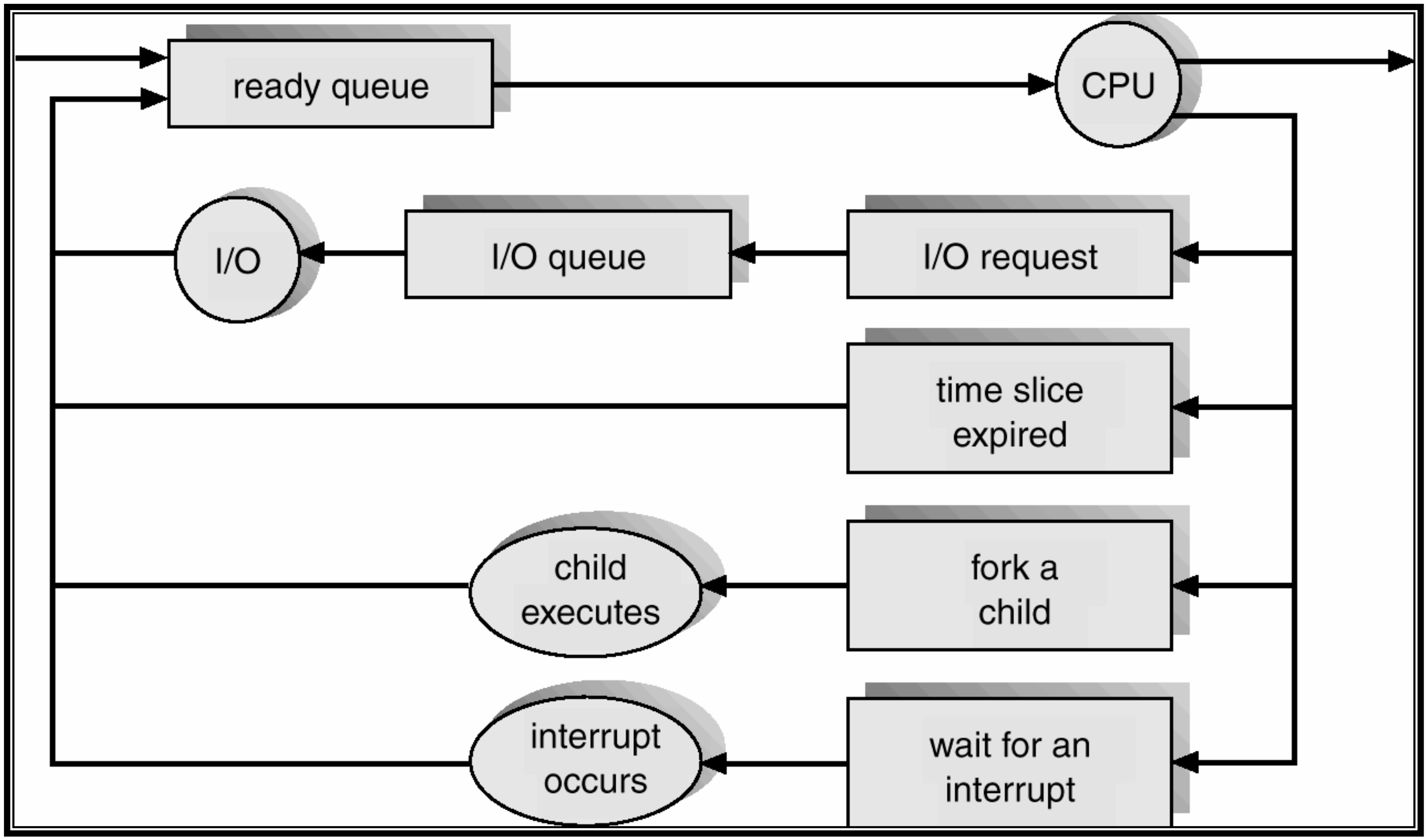
Addition of Medium Term Scheduling



When are scheduling decision made?

- Switch from running to waiting
- Switch from running to ready
- Switch from waiting to ready
- On process termination

When are scheduling decisions made?



Preemptive vs. non-preemptive

- **Non-preemptive scheduling**
  - Once in running state, process will continue
  - Potential to monopolize the CPU
  - May voluntarily yield the CPU
- **Preemptive scheduling**
  - Currently running process may be interrupted by OS and put into ready state
  - Timer interrupts required (for IRP)
  - Incurs context switches
- Should kernel code be preemptive or non-preemptive?

Scheduling Criteria 1

- User-oriented
  - Response time
    - Elapsed time between submission of a request and until there is an output
  - Waiting time
    - Total time process is spending in ready queue
  - Turnaround time
    - Amount of time to execute a process, from creation to exit

Scheduling Criteria 2

- System-oriented
  - Effective and efficient utilization of CPU(s)
  - Throughput
    - Number of jobs executed per unit of time
- Often, conflicting goals

Scheduling Criteria 3

- Performance related
  - Quantitative
  - Measurable, such as response time & throughput
- Non-performance related
  - Qualitative
  - Predictability
  - Proportionality

Criteria for each type of system

Different “priorities” for different types of systems

- All Systems
  - Fairness, give each process fair share of CPU
  - Balance, keep all system components busy
  - Enforce system-wide policies
- Batch
  - non-preemptive policies, or preemptive with long time quanta
  - Throughput, turnaround, CPU utilization

…

- Interactive
  - Preemptive is essential,
  - Response time, proportionality (meet user expectation)
- Real-time (hard & soft)
  - Preemptive often not necessary for hard real-time systems
  - Meeting deadlines (avoid loosing data), predictability (avoid quality degradation, e.g., in multimedia systems)

Optimization Criteria

- Max. CPU utilization
- Max. throughput
- Min. turnaround time
- Min. waiting time
- Min. response time

CPU-I/O Burst Cycles

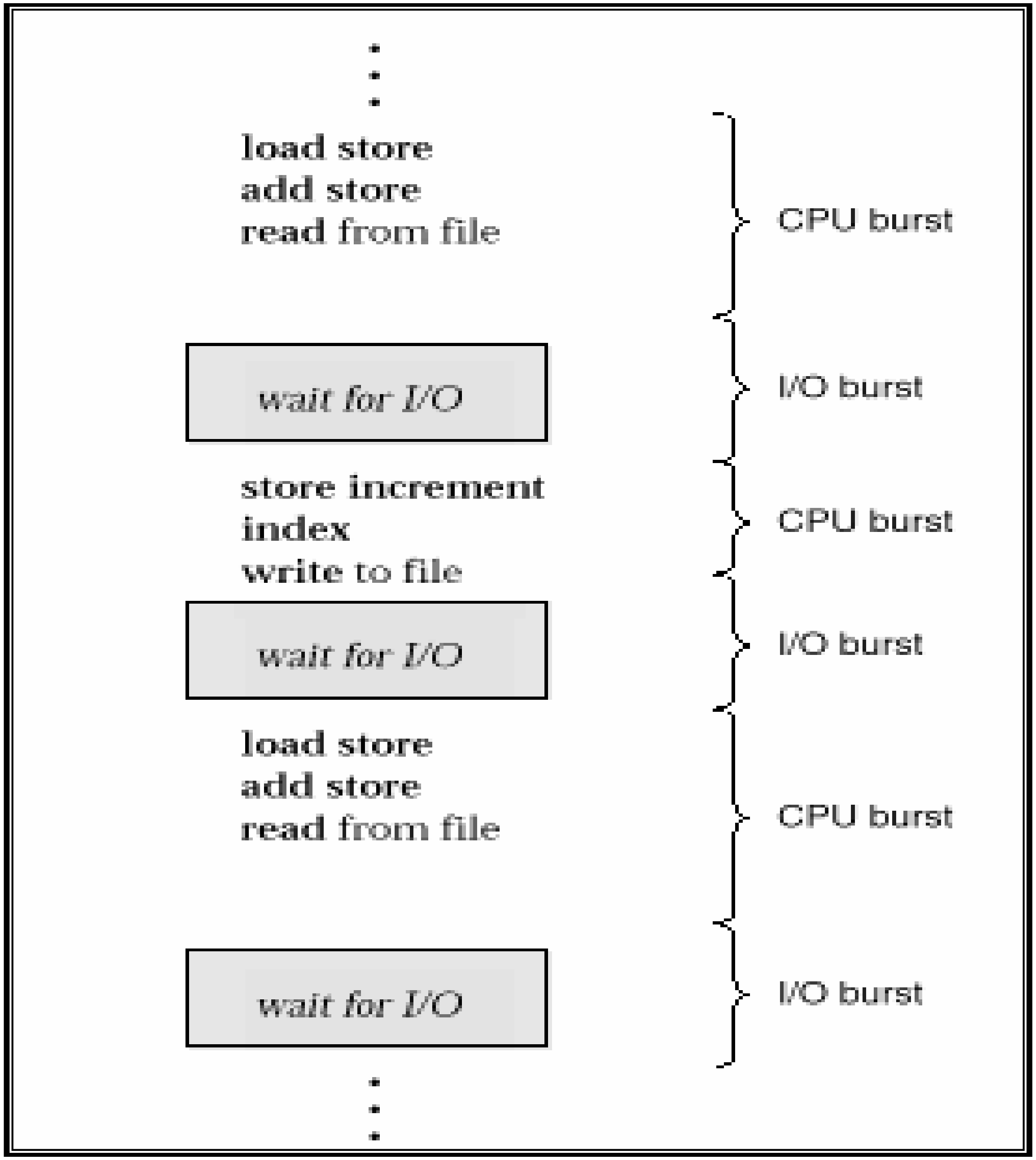
Processes typically consist of

- CPU bursts
- I/O bursts

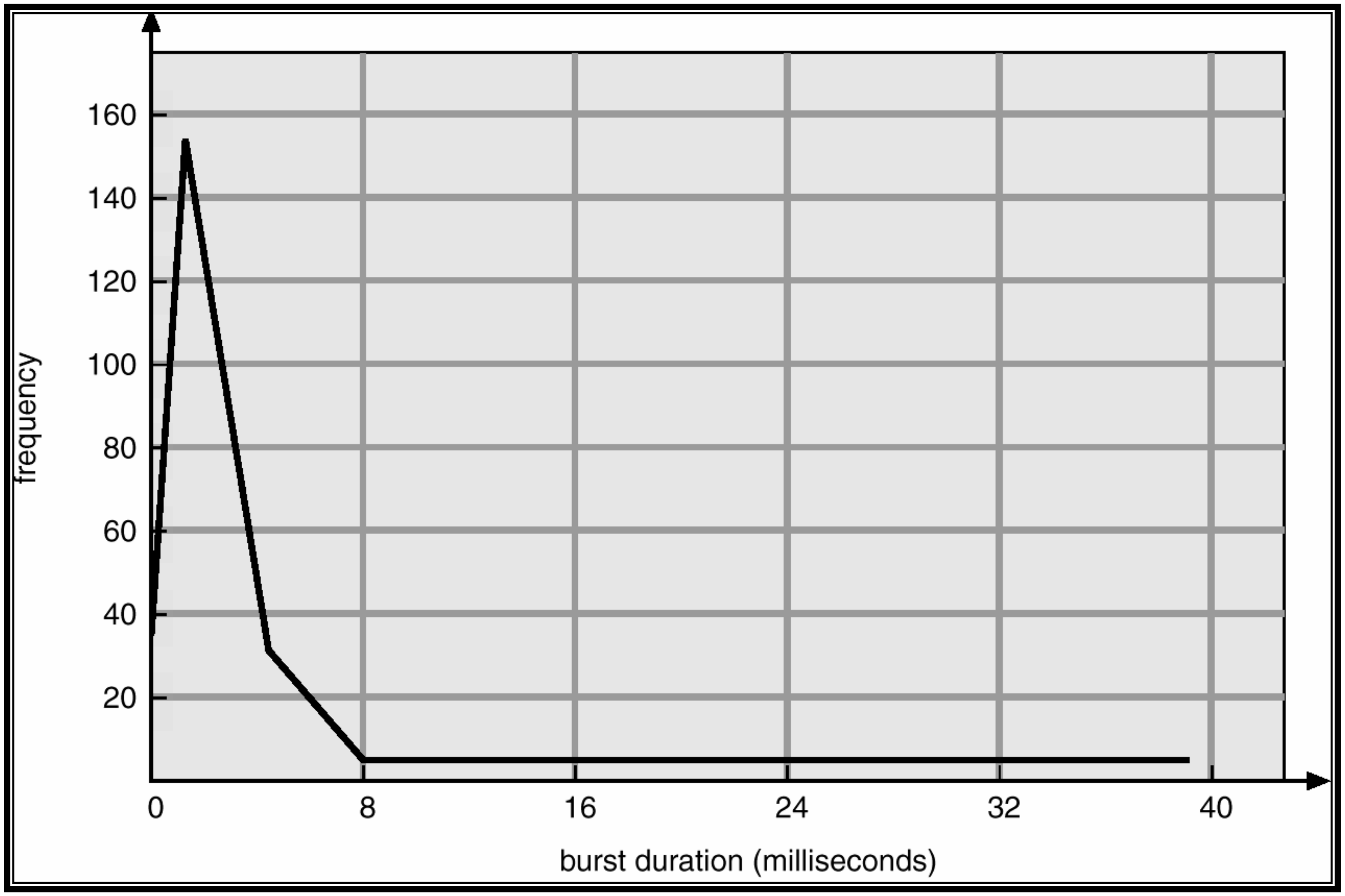
Duration and frequency of bursts vary greatly from process to process

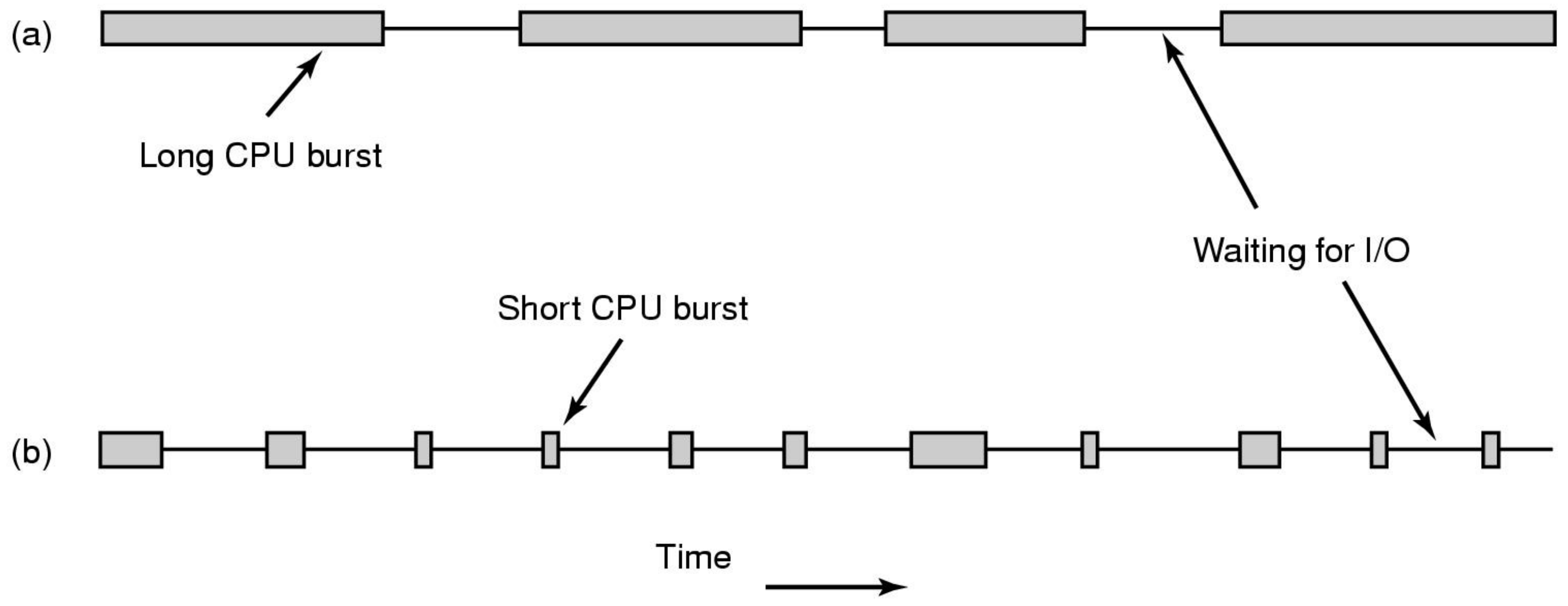
- – **CPU-bound** few very long CPU bursts
  - Number crunching tasks, image processing
- – **I/O-bound** many short CPU bursts

- Maximum CPU utilization obtained with multiprogramming and mixing CPU and I/O bound tasks for maximal parallel resource utilization


load store
add store
read from file
wait for I/O
store increment
index
write to file
wait for I/O
load store
add store
read from file
wait for I/O
CPU burst
I/O burst
CPU burst
I/O burst
CPU burst
I/O burst

Histogram of CPU-burst Times





- Bursts of CPU usage alternate with periods of I/O wait
  - a CPU-bound process
  - an I/O bound process

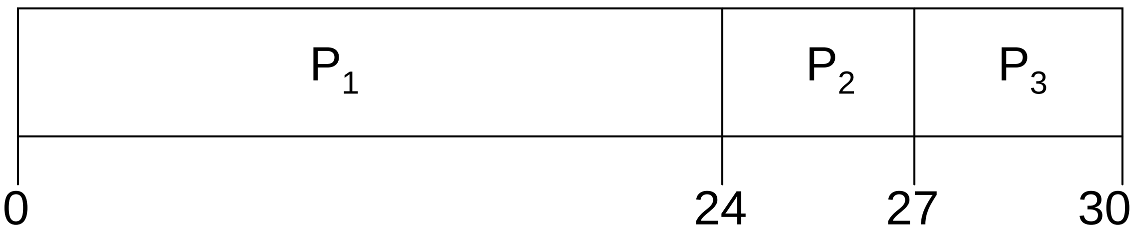
Dispatcher

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

Scheduling Algorithms

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

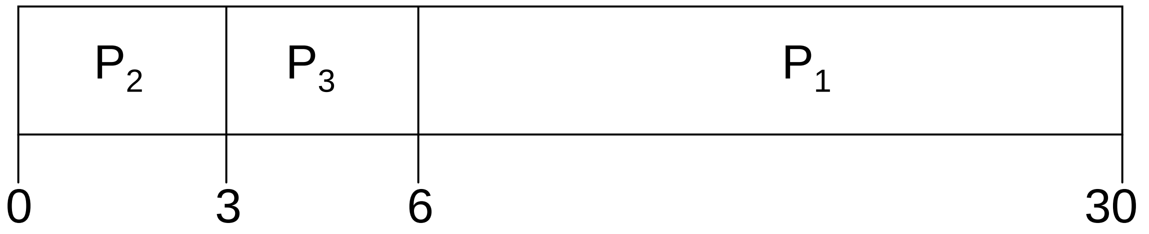


- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1 .



- Waiting time for $P_1 = 6$; $P_2 = 0$, $P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case.

FCFS cont.’d.

- Applied in batch systems (non-preemptive)
- Each task runs, once started, runs to completion
- Scheduler selects oldest process in ready queue
- ***Convoy effect*** short process behind long process (I/O bound tasks may be waiting behind CPU bound tasks
- Not suitable for time sharing

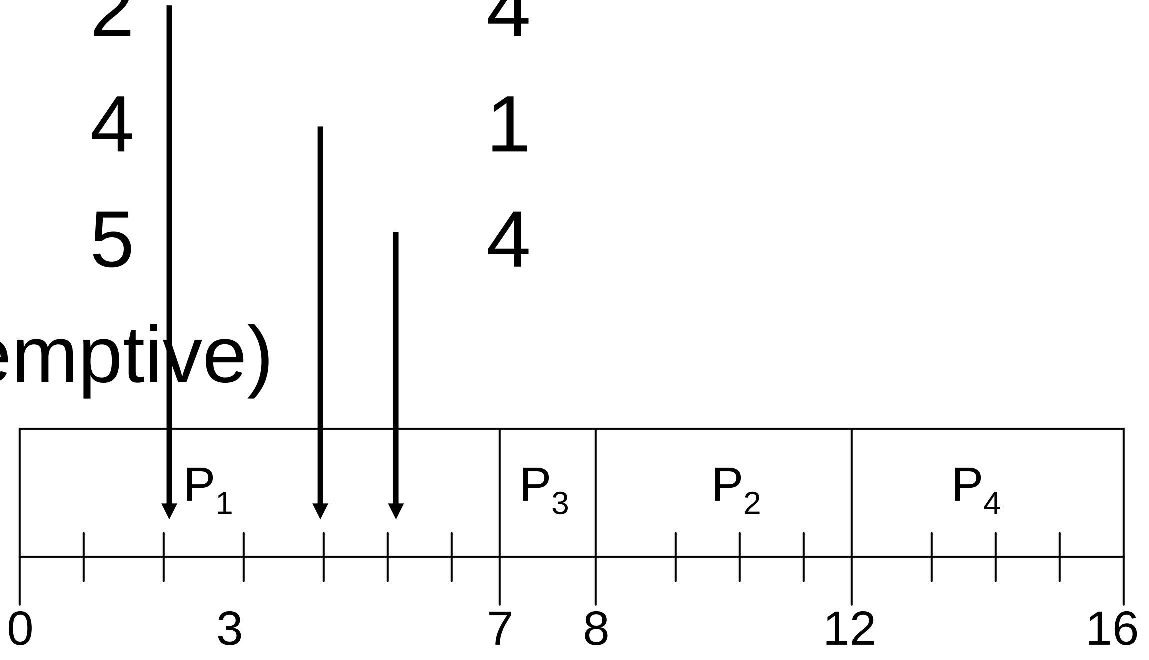
Shortest-Job-First (SJR) Scheduling

- Associate with each task the **length of its next CPU burst**.
- Use these lengths to schedule the task with the shortest time.
- An **expected** next burst length
- Two schemes:
  - **non-preemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - **preemptive** – Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (non-preemptive)



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

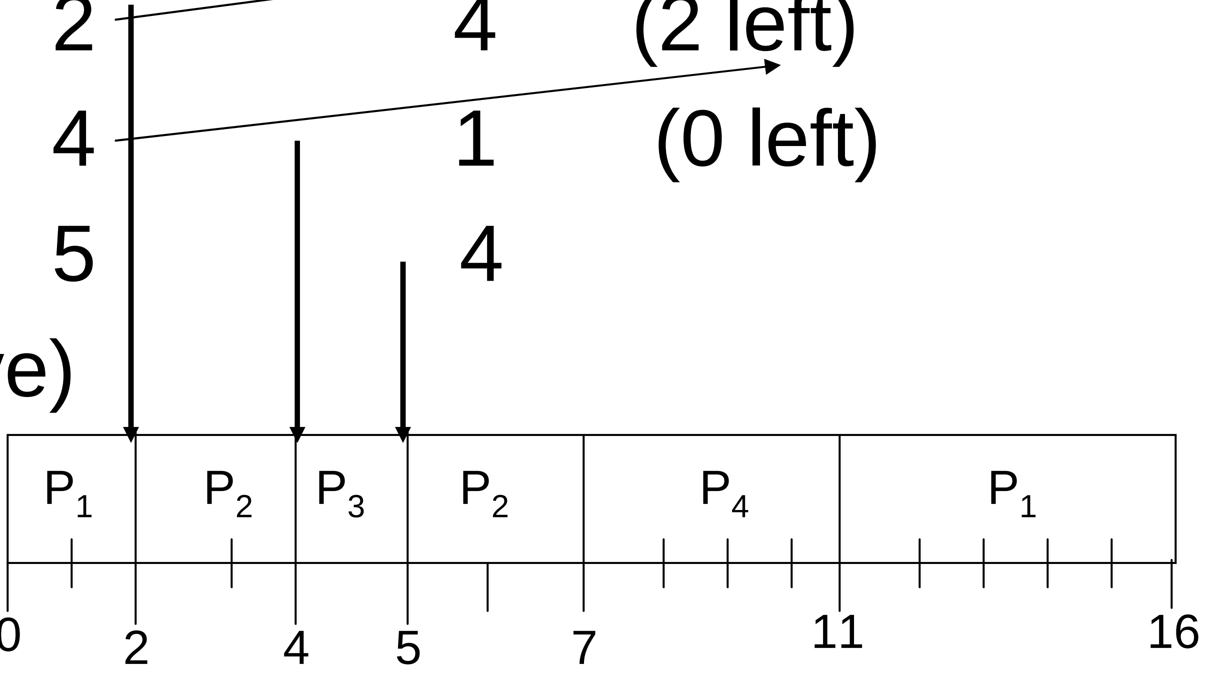
SJF

- Short tasks jump ahead of longer ones
- May need to abort tasks exceeding their burst length expectations
- Long running tasks may be starved
- Burst length could be user/application provided
- Could be based on past execution pattern of task

Example of Preemptive SJF

Process	Arrival Time	Burst Time	
P_1	0	7	(5 left)
P_2	2	4	(2 left)
P_3	4	1	(0 left)
P_4	5	4	

- SJF (preemptive)



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

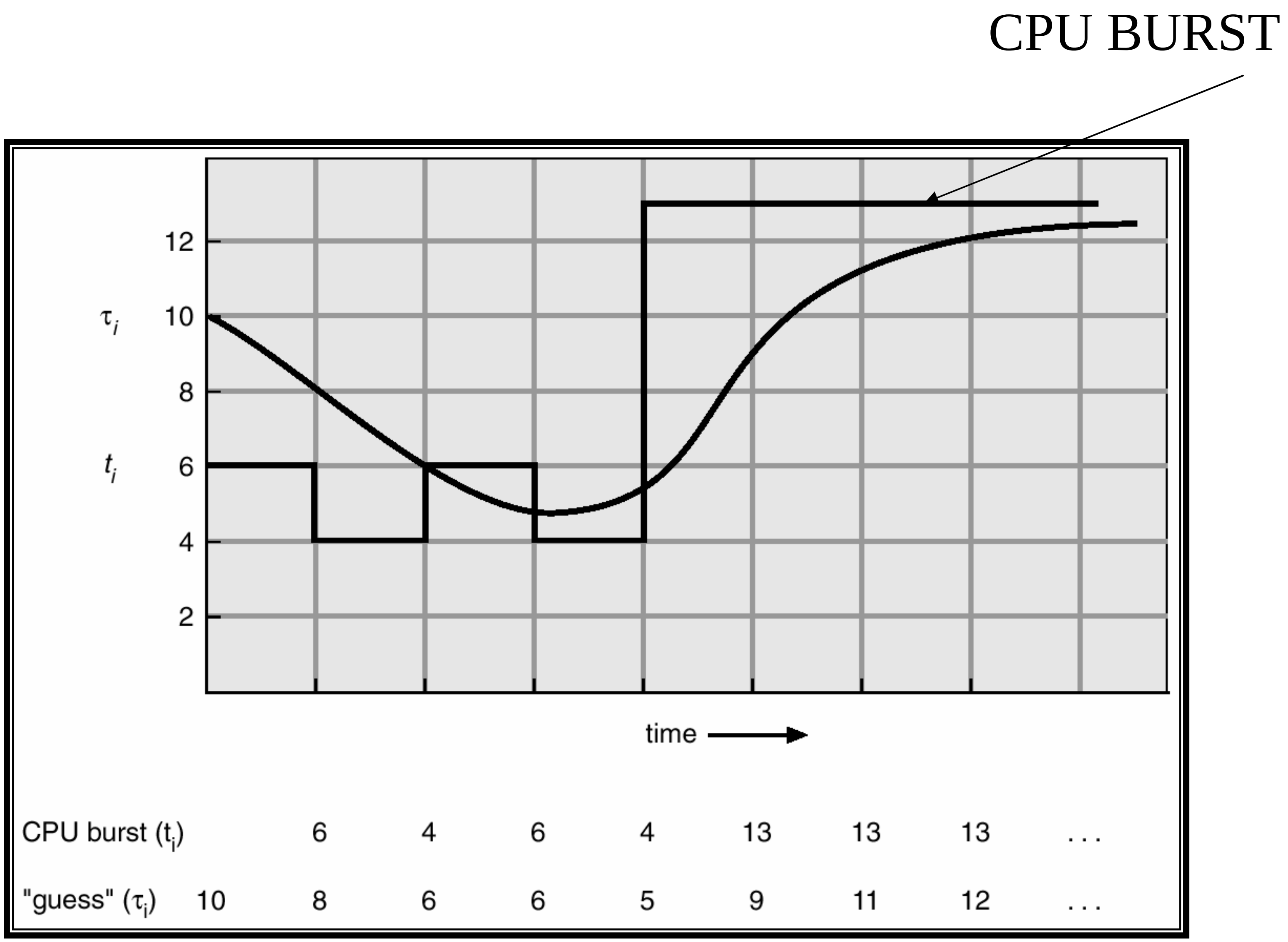
Determining Length of Next CPU Burst

- Can only estimate the length.
- Can be done by using the length of previous CPU bursts, using exponential averaging.

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

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

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)		6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10	8	6	6	5	9	11	12	. . .

Examples of Exponential Averaging

- α =0
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count.
- α =1
  - $\tau_{n+1} = t_n$
  - Only the actual last CPU burst counts.

Examples of Exponential Averaging

- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\ t_n + (1 - \alpha)\ \alpha\ t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j\ \alpha\ t_{n-i} + \ldots$$
$$+ (1 - \alpha)^n\ t_0\ \tau_0$$

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

Priority Scheduling

- 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 vs. nonpreemptive
- SJF is 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/decrease the priority of tasks

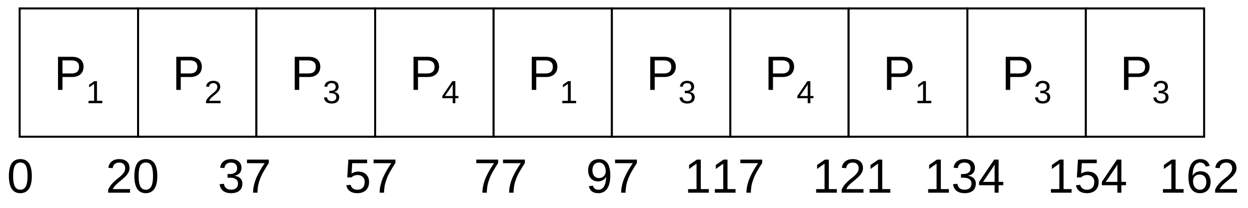
Round Robin (RR)

- Each process gets a small unit of CPU time
  - Called a ***time quantum***
  - usually 10-100 milliseconds
  - preempted and added to the end of the ready queue.
- *n* processes in the ready queue and time quantum *q*
  - 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.
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high.

Example of RR

- with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24



- Typically, higher average turnaround than SJF, but better *response*.

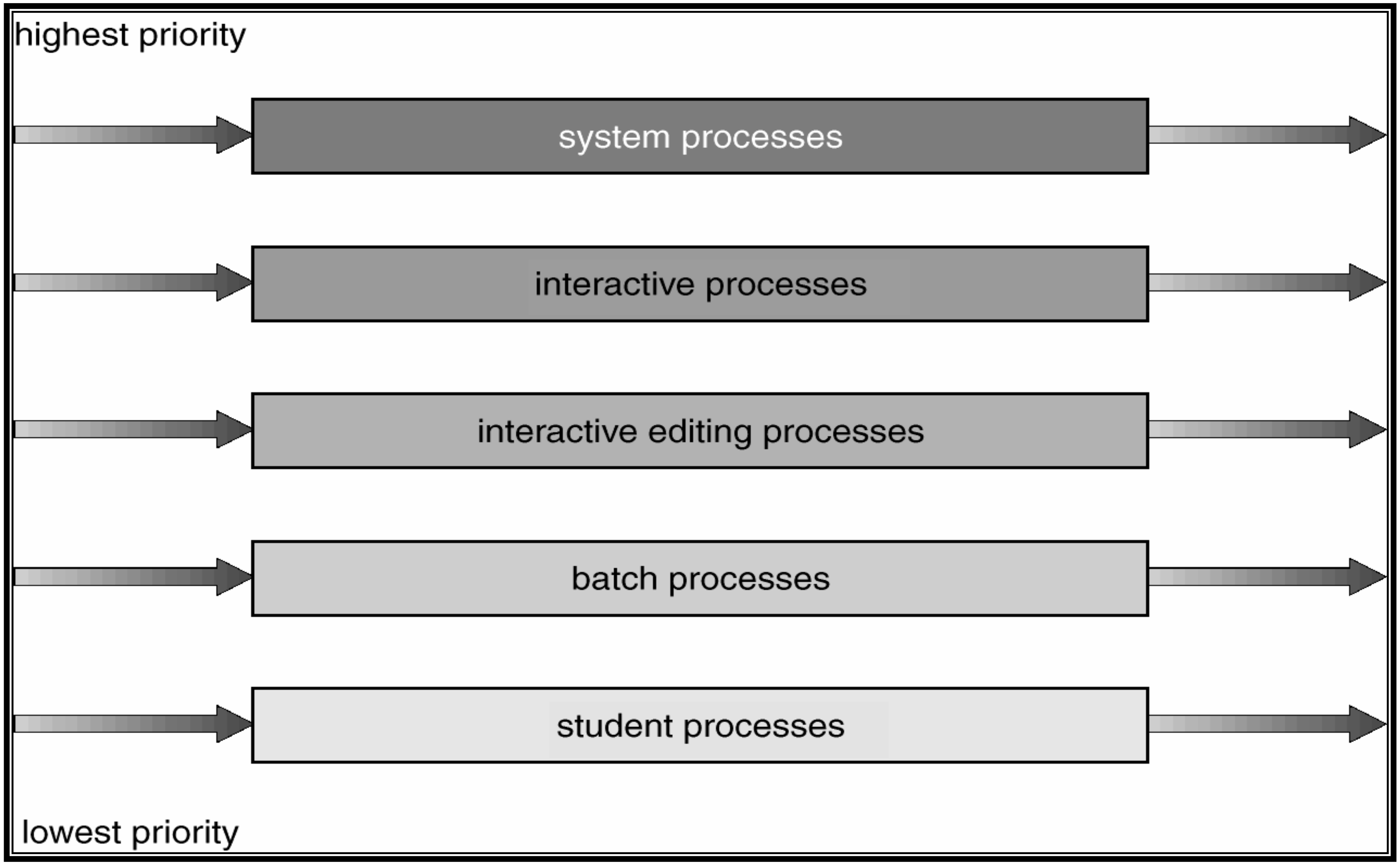
RR

- No starvation, everybody gets to run
- Choice of length of time slice/quantum is crucial

Multilevel Queue

- 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 and 20% to background in FCFS

Multilevel Queue Scheduling



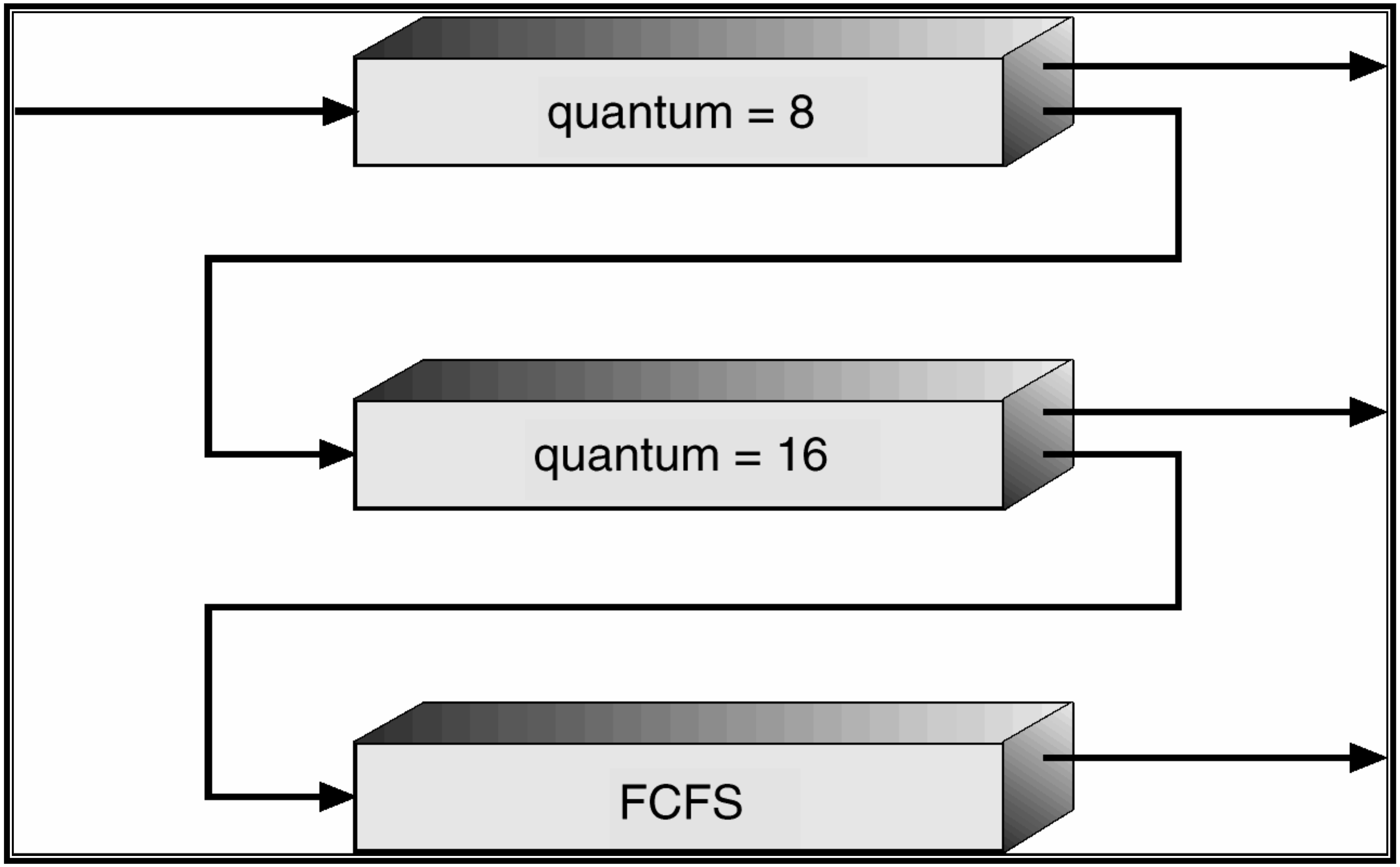
Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – time quantum 8 milliseconds
  - Q_1 – time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 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 Q_1.
  - At Q_1 job 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 Queues



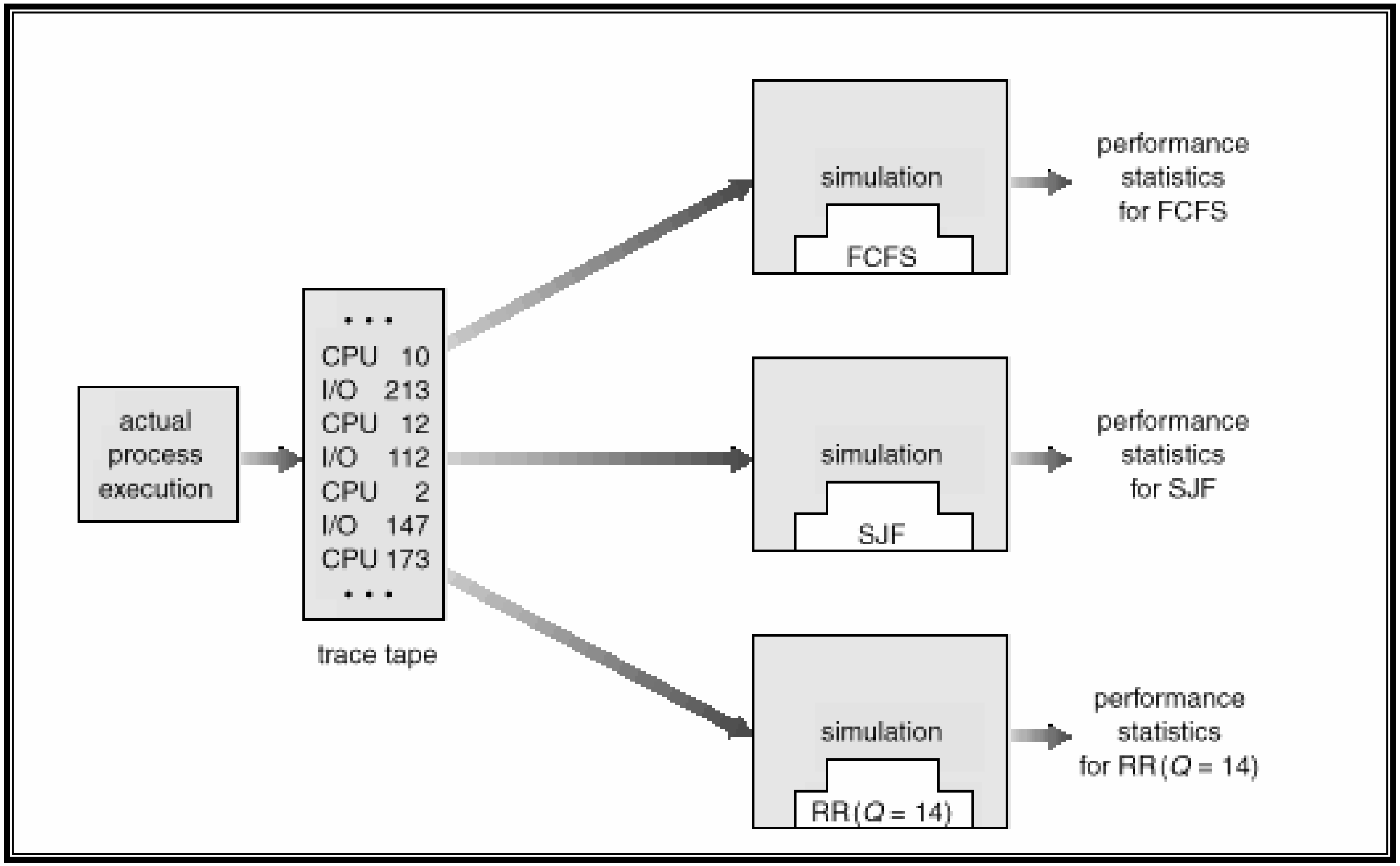
Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous processors* within a multiprocessor.
- *Load sharing*

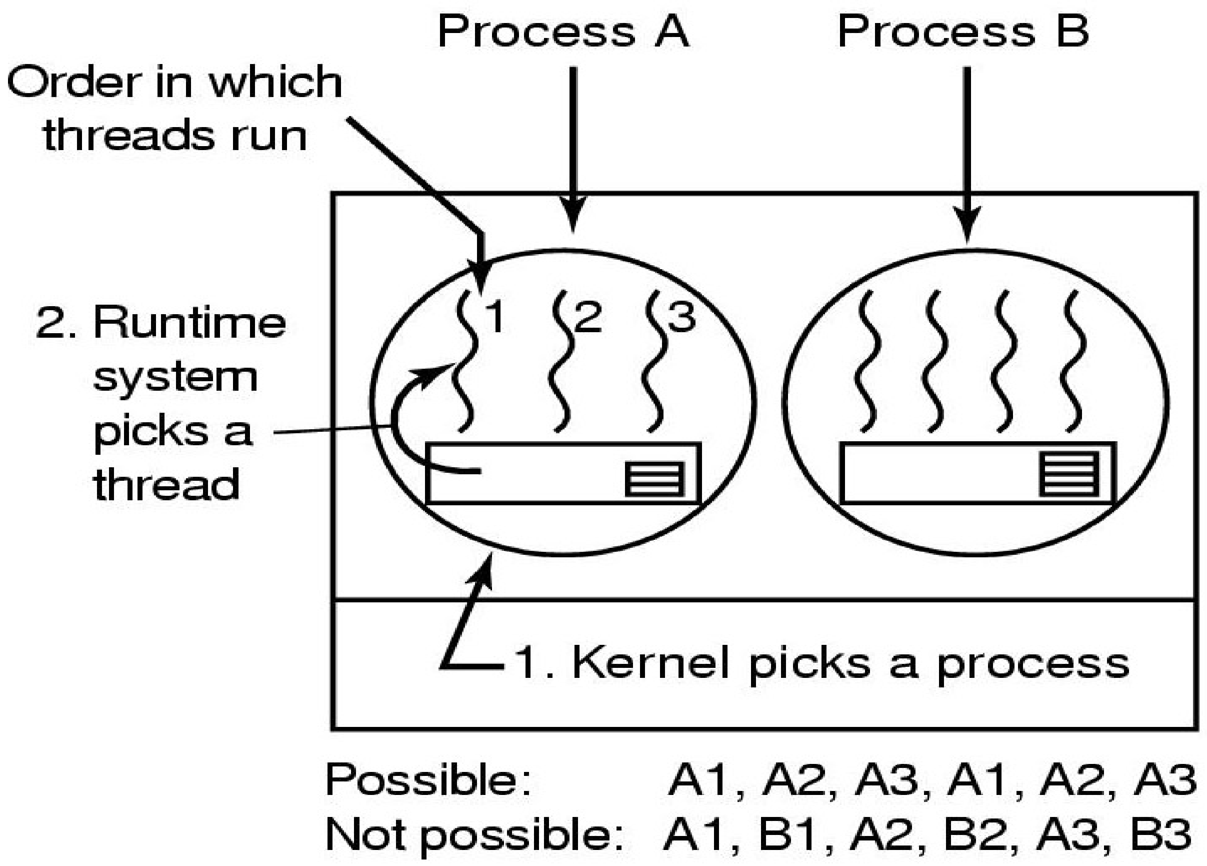
Real-Time Scheduling

- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time.
- *Soft real-time* computing – requires that critical processes receive priority over less fortunate ones.

Evaluation of CPU Schedulers by Simulation



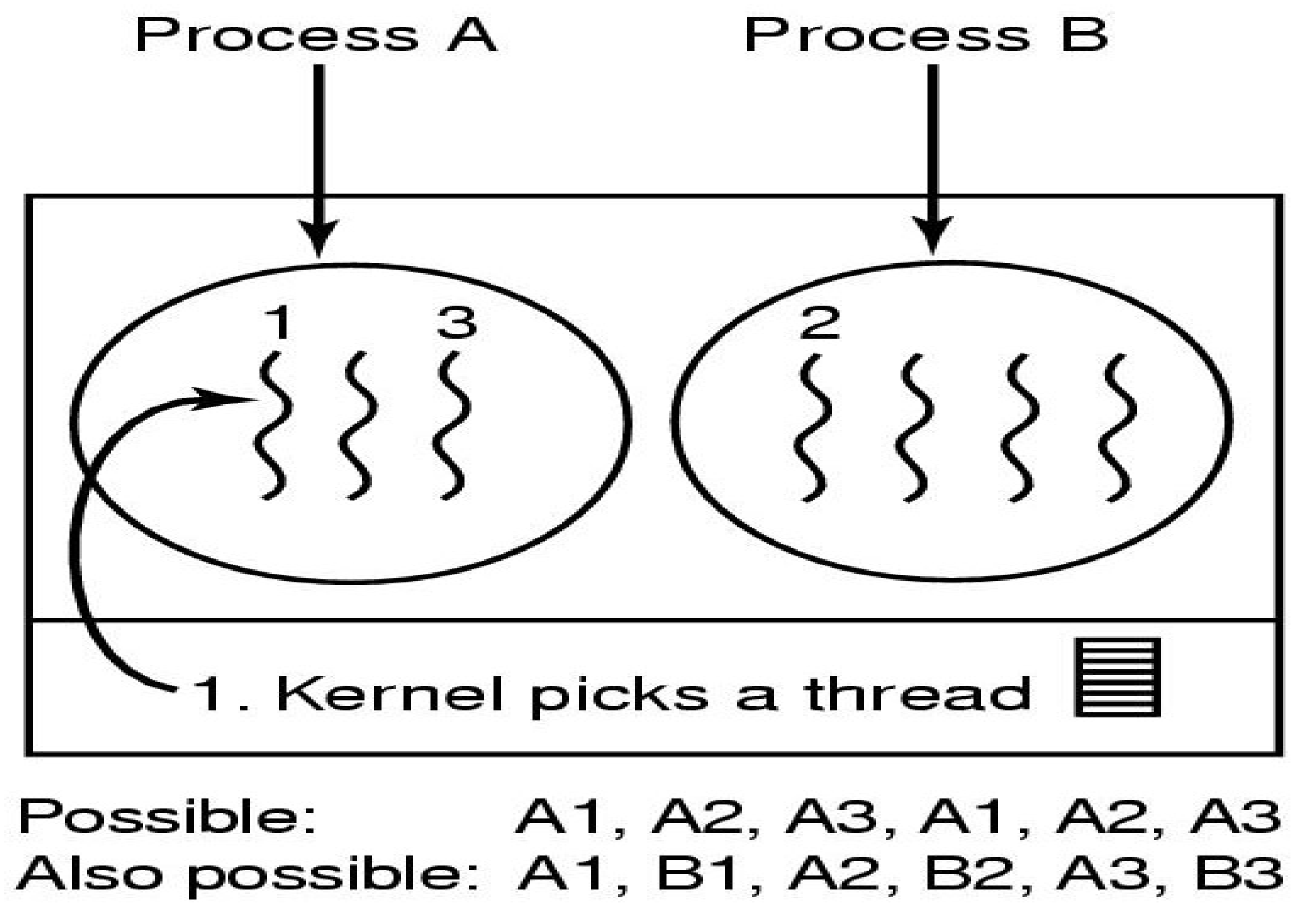
Thread Scheduling


Process A
Process B
Order in which threads run
2. Runtime system picks a thread
1
2
3
1. Kernel picks a process
Possible: A1, A2, A3, A1, A2, A3
Not possible: A1, B1, A2, B2, A3, B3

Possible scheduling of user-level threads

- 50-msec process quantum
- threads run 5 msec/CPU burst

Thread Scheduling


Process A
Process B
1
3
2
1. Kernel picks a thread
Possible: A1, A2, A3, A1, A2, A3
Also possible: A1, B1, A2, B2, A3, B3

Possible scheduling of kernel-level threads

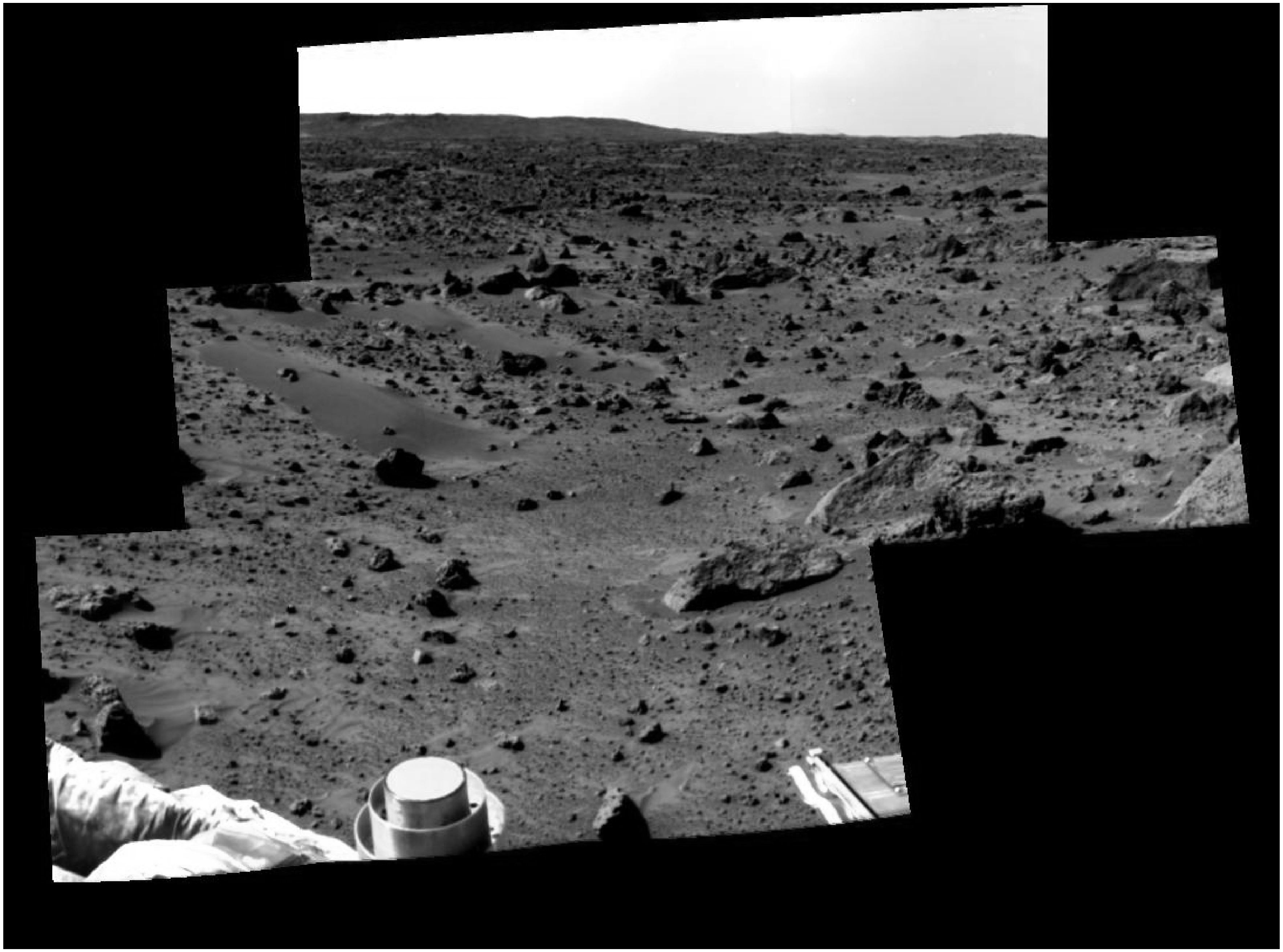
- 50-msec process quantum
- threads run 5 msec/CPU burst

Pathfinder Mission to Mars

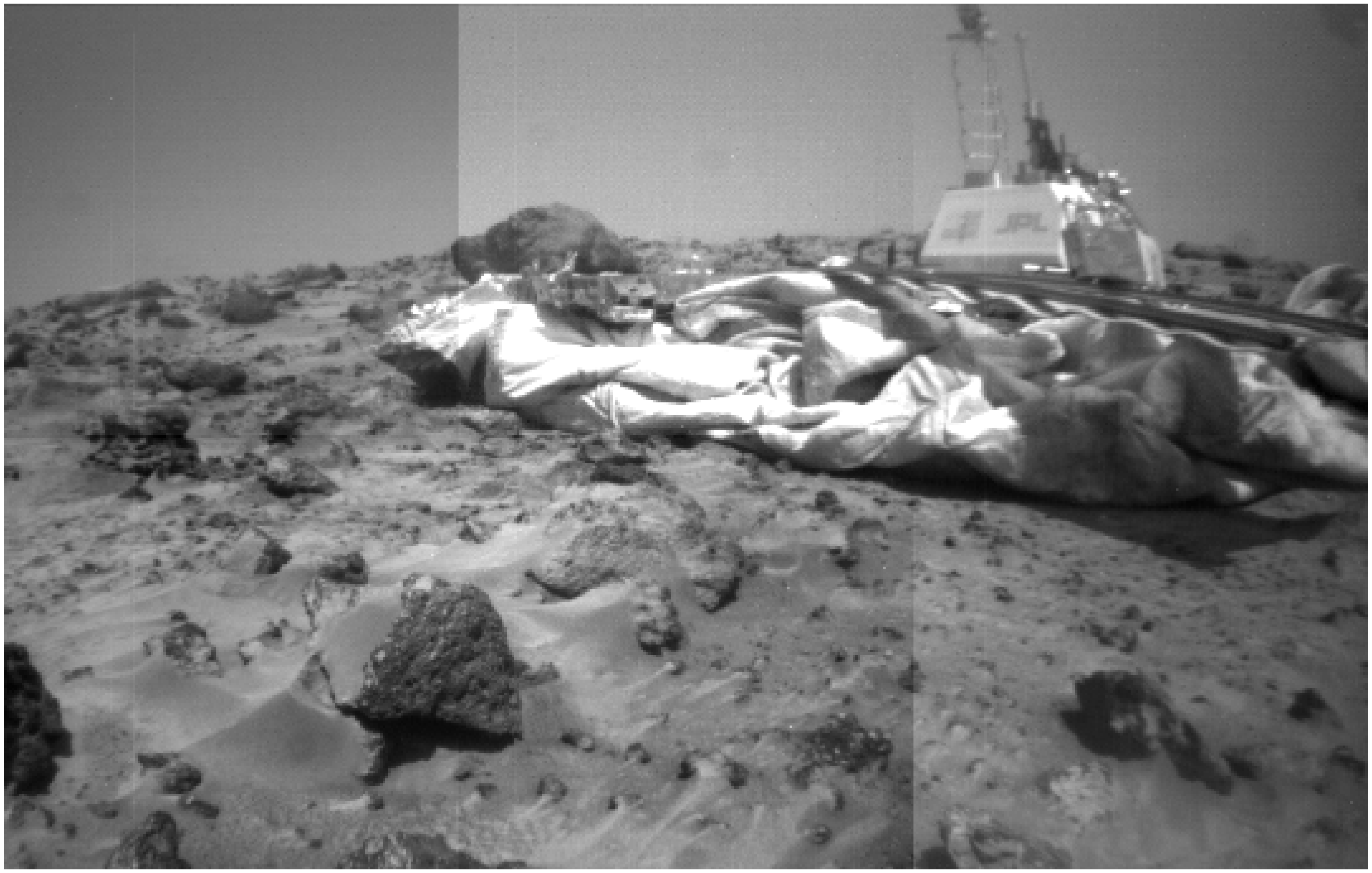
Launch Date:	04 December 1996 UT 06:58
Arrival Date:	04 July 1997 UT 16:57
Launch Vehicle:	Delta II
Mass:	264 kg (lander), 10.5 kg (rover)
Power System:	Solar panels

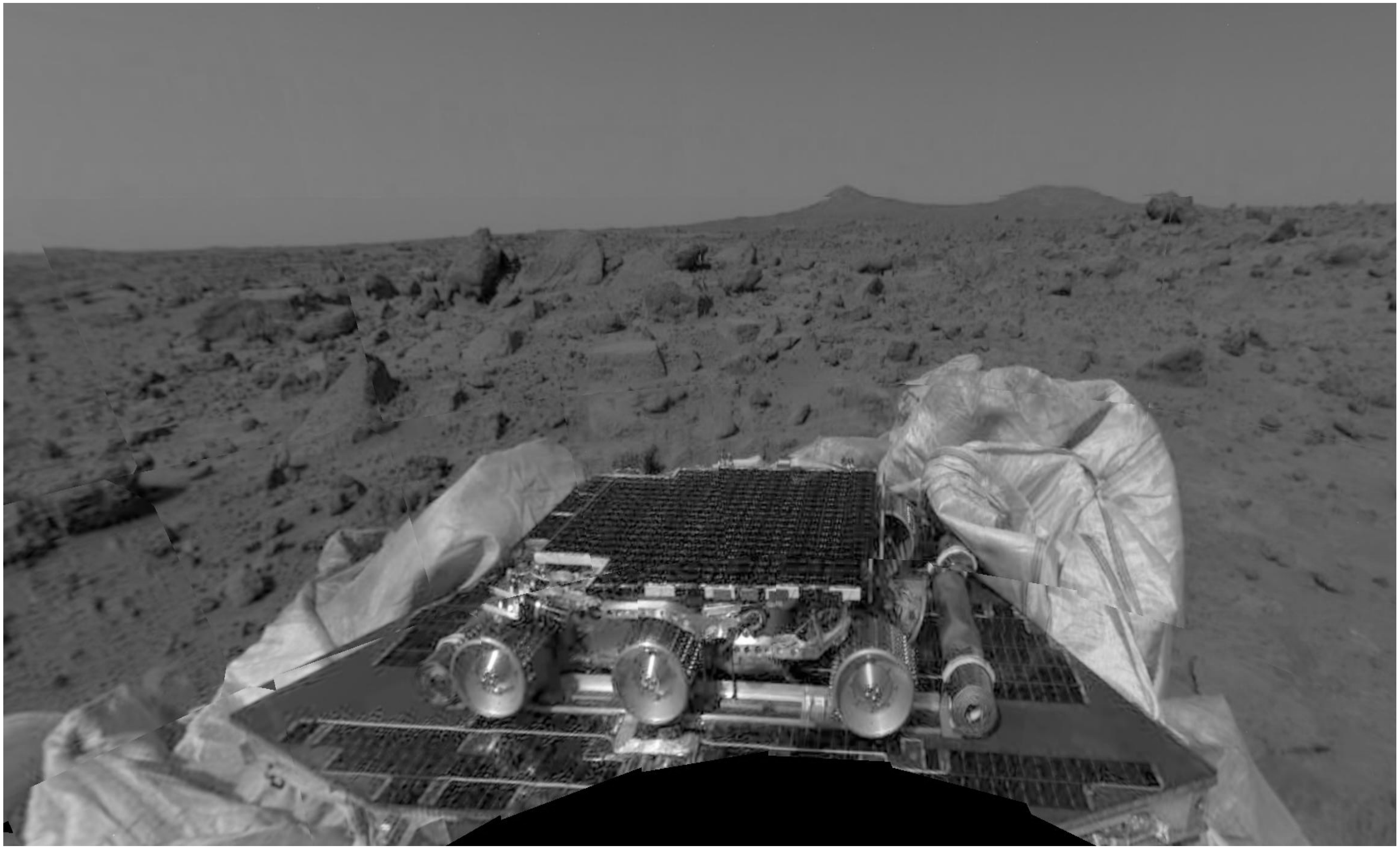
Pathfinder Mission to Mars

- The Mars Pathfinder mission was widely proclaimed as "flawless" in the early days after its July 4th, 1997 landing on the Martian surface.
- Successes included its unconventional "landing" -- bouncing onto the martian surface surrounded by airbags,
- Deploying the Sojourner rover, and **gathering** and **transmitting voluminous data** back to earth, including the panoramic pictures that were *such a **big hit** on the Web*.
- But a few days into the mission, not long after Pathfinder started gathering meteorological data, **the spacecraft began experiencing total system resets**,
- Each resulting in **losses of data**.
- The press reported these failures in terms such as "**software glitches**" and "**the computer was trying to do too many things at once**".



ECE 344 Operating Systems


JPL



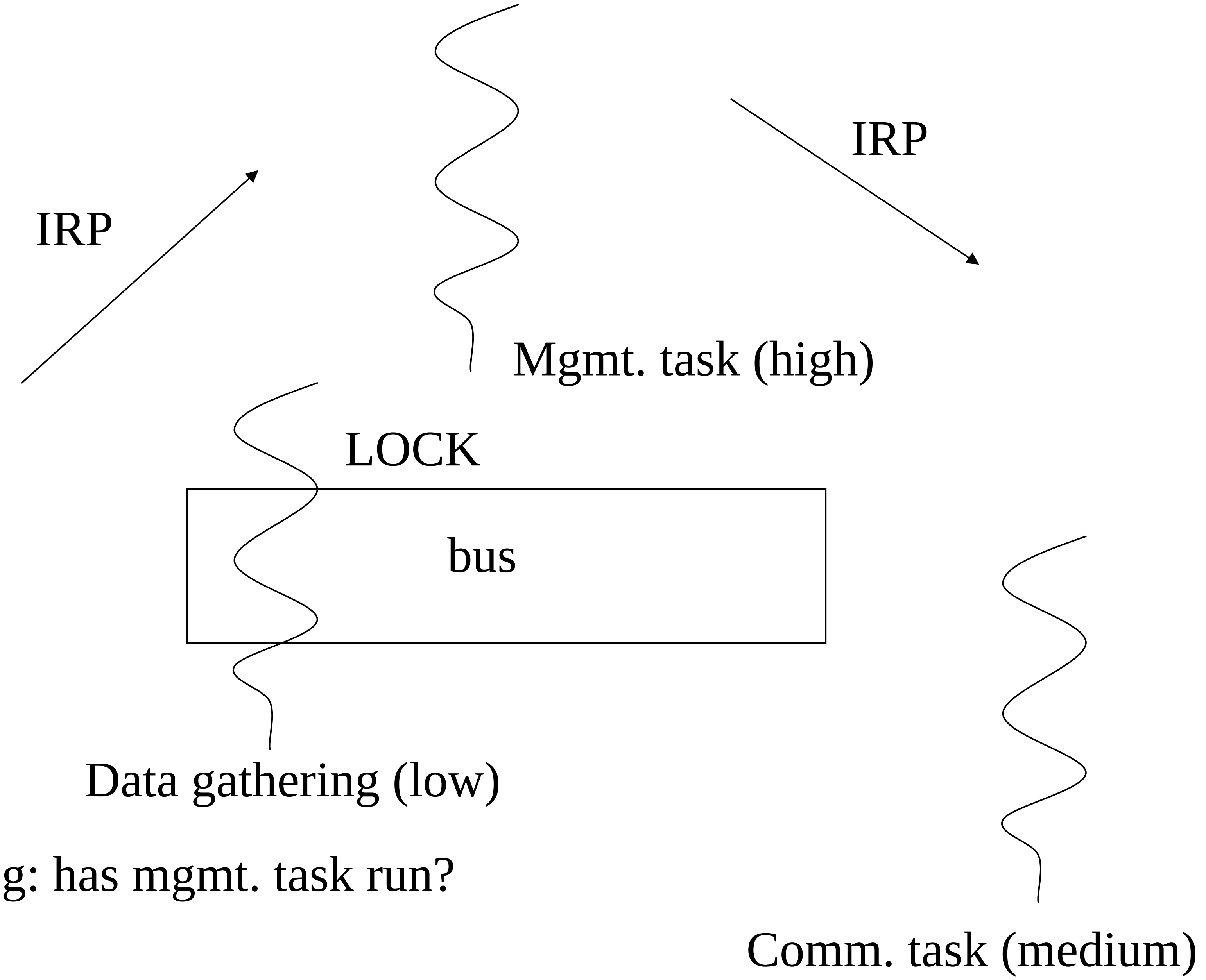
ECE 344 Operating Systems

Software Architecture

- Based on an “information bus”
  - A shared memory area for passing information between components of the spacecraft
- Access to bus is synchronized with a lock
- **High priority bus management task** (ran often) moves data in/out of bus
- **Low priority meteorological data gathering task** (ran infrequently) publishes data to bus
- **Medium priority, long running, communication task**

The Problem

- An interrupt may cause the high priority bus management task to run, while the low priority data gathering task holds the lock on the bus; so the management task must wait
- Another interrupt may cause the medium priority, long running communication task to run
- The communication tasks prevents (due to higher priority) the low priority data gathering task from running, which prevents the lock from being unlocked, which prevents the bus management tasks from running
- After some time a watchdog timer goes off, checks whether the bus management task has run recently
- Since that is not the case, it concludes that there is a problem and initiates a total system reset



Watchdog: has mgmt. task run?

Priority Inversion

- Classical problem of priority inversion
- Higher priority task is waiting for a lower priority task
- **Solution**: lower priority task inherits the priority from the higher priority task waiting on the lock for the time it spends in the critical section
- **Priority inheritance protocol**

Really Remote Debugging

- Pathfinder used VxWorks
- VxWorks can be run in a mode where it records a total trace of all interesting system events, including context switches, uses of synchronization objects, and interrupts. After the failure, JPL engineers spent hours and hours running the system on the exact spacecraft replica in their lab with tracing turned on, attempting to replicate the precise conditions under which they believed that the reset occurred.
- Early in the morning, after all but one engineer had gone home, the engineer finally reproduced a system reset on the replica. Analysis of the trace revealed the priority inversion.

Remote Bug Fixing

- When created, a VxWorks mutex object accepts a boolean parameter that indicates whether priority inheritance should be performed by the mutex. The mutex in question had been initialized with the parameter off; had it been on, the low-priority meteorological thread would have inherited the priority of the high-priority data bus thread blocked on it while it held the mutex, causing it be scheduled with higher priority than the medium-priority communications task, thus preventing the priority inversion. Once diagnosed, it was clear to the JPL engineers that using priority inheritance would prevent the resets they were seeing.
- VxWorks contains a C language interpreter intended to allow developers to type in C expressions and functions to be executed on the fly during system debugging. The JPL engineers fortuitously decided to launch the spacecraft with this feature still enabled. By coding convention, the initialization parameter for the mutex in question (and those for two others which could have caused the same problem) were stored in global variables, whose addresses were in symbol tables also included in the launch software, and available to the C interpreter. A short C program was uploaded to the spacecraft, which when interpreted, changed the values of these variables from FALSE to TRUE. No more system resets occurred.