Scheduling

Daniel Bosk[1]

Department of Information and Communication Systems (ICS),
Mid Sweden University, Sundsvall.

sched.tex 195 2016-12-17 22:47:16Z jimahl

Overview

1. Process Scheduling
   - CPU Scheduling
   - CPU-I/O Burst Cycle
   - Preemptive and Non-Preemptive Scheduling
2. Algorithms
   - Scheduling Criteria
   - First-Come, First-Served (FCFS)
   - Shortest Job First (SJF)
   - Priority Scheduling
   - Round Robin (RR)
   - Multilevel Queue
   - Multilevel Feedback Queue
3. Multiprocessor Scheduling
   - Asymmetric Multiprocessor Scheduling
   - Symmetric Multiprocessor Scheduling

Literature

This lecture covers process scheduling and its algorithms. It gives an overview of Chapter 5 “Process Scheduling” in [SGG13a].

Overview

CPU Scheduling

- CPU-scheduling is at the heart of multiprogramming.
- It concerns the short-term scheduler.
- It is used to schedule which process is allowed to execute its instructions on the CPU at any given time.
- If the operating system supports kernel threads it's kernel threads, not processes, being scheduled.

CPU Scheduling

- The short-term scheduler selects which process to allocate the CPU to next.
- The dispatcher is the function which performs the context switch, switching to user mode, and restarting the process on the correct instruction.
- It's important that these functions execute quickly, as they might be executed every 10 to 100 milliseconds.

CPU-I/O Burst Cycle

- Process execution can be divided into two classes.
  1. CPU-bursts: The process executes on the CPU, actually executing instructions, e.g. performing computations.
  2. I/O-bursts: The process spends its time reading from or writing to various devices, i.e. the process spends a lot of time waiting for the I/O-devices.

CPU-I/O Burst Cycle

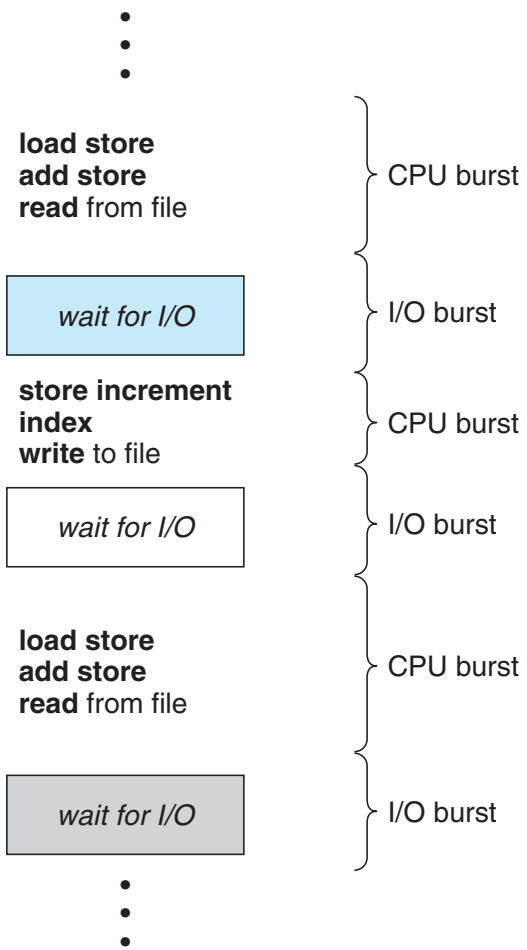

Figure: An illustration of the CPU-I/O burst cycle for a process. Image: [SGG13b].

CPU-I/O Burst Cycle

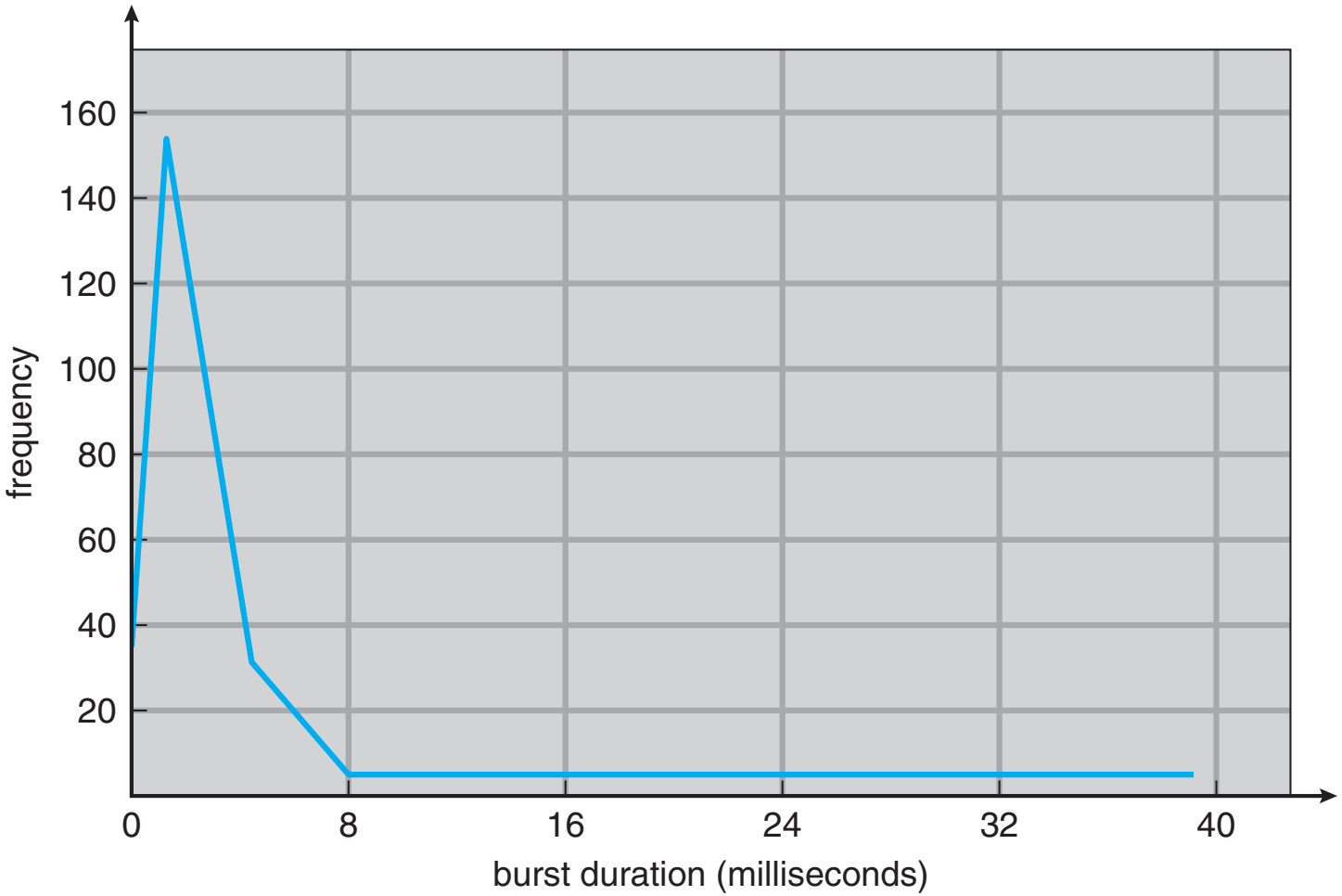

Figure: Histogram of CPU-burst durations. Image: [SGG13b].

Preemptive and Non-Preemptive Scheduling

- Scheduling can be divided into the two classes preemptive and non-preemptive scheduling.
- Non-Preemptive or cooperative scheduling leaves it up to the processes to give up the processor, e.g. through blocking on I/O.
- Preemptive does abort one process in favour of another process, e.g. every process may have 100 milliseconds each in turn.

Preemptive and Non-Preemptive Scheduling

- Preemptive is most prominent.
- Requires special hardware though, e.g. timers.
- Synchronization issuse also arises, e.g. race conditions.

Overview

Scheduling Criteria

CPU utilization	We want to keep the CPU as busy as we possibly can.
Throughput	One measure of performance is how many processes complete per time unit.
Turnaround time	Yet another measure is how long it takes from jobs submission to job completion.
Waiting time	The scheduling algorithm doesn't affect the time it takes for a process to execute instructions or do I/O, it only affects the waiting time.
Response time	In an interactive system the turnaround time or throughput might not be relevant measures. Response time is more suitable as a process might produce some output even before it's done.

First-Come, First-Served (FCFS)

- A very basic scheduling algorithm allocating the CPU to the processes in FIFO order.
- The downside of this algorithm is that average waiting time is usually quite long.

First-Come, First-Served (FCFS)

Proc	Burst-time
P_0	24
P_1	2
P_2	4

Table: An example of processes and their burst-time.

Shortest Job First (SJF)

- This algorithm uses the length of the processes' CPU-bursts to determine the order of execution.
- The problem with this algorithm is that it's nearly impossible to implement, estimating the length of a CPU-burst is very hard to do accurately.
- However, it's probably optimal, giving the minimum average waiting time.
- One can do it by estimation, using the lengths of previous bursts to predict the length of the coming one.

Shortest Job First (SJF)

Proc	Burst-time
P_0	6
P_1	8
P_2	7
P_3	3

Table: An example of processes and their burst-time.

Priority Scheduling

- SJF is a form of priority scheduling where the priority for a process is the inverse predicted length of a CPU-burst.
- Priority scheduling can also be generalized.
- Higher-priority processes are scheduled before lower-priority processes.
- Processes having the same priority is scheduled using FCFS.
- The major problem with all priority-based algorithms is indefinite blocking, or starvation.
- One solution to this is some kind of aging, older processes increase in priority.

Priority Scheduling

Proc	Burst-time	Priority
P_0	10	3
P_1	1	1
P_2	2	4

Table: An example of processes and their burst-time and priorities.

Round Robin (RR)

- Uses time-quantum, each process is allocated this much time on the CPU, then it's preempted.
- Average waiting-time is often long.
- With a time-quantum q and n processes, each process executes q time units and waits at most $(n-1) \times q$ time units.
- The performance depends on the size of the time-quantum.

Round Robin (RR)

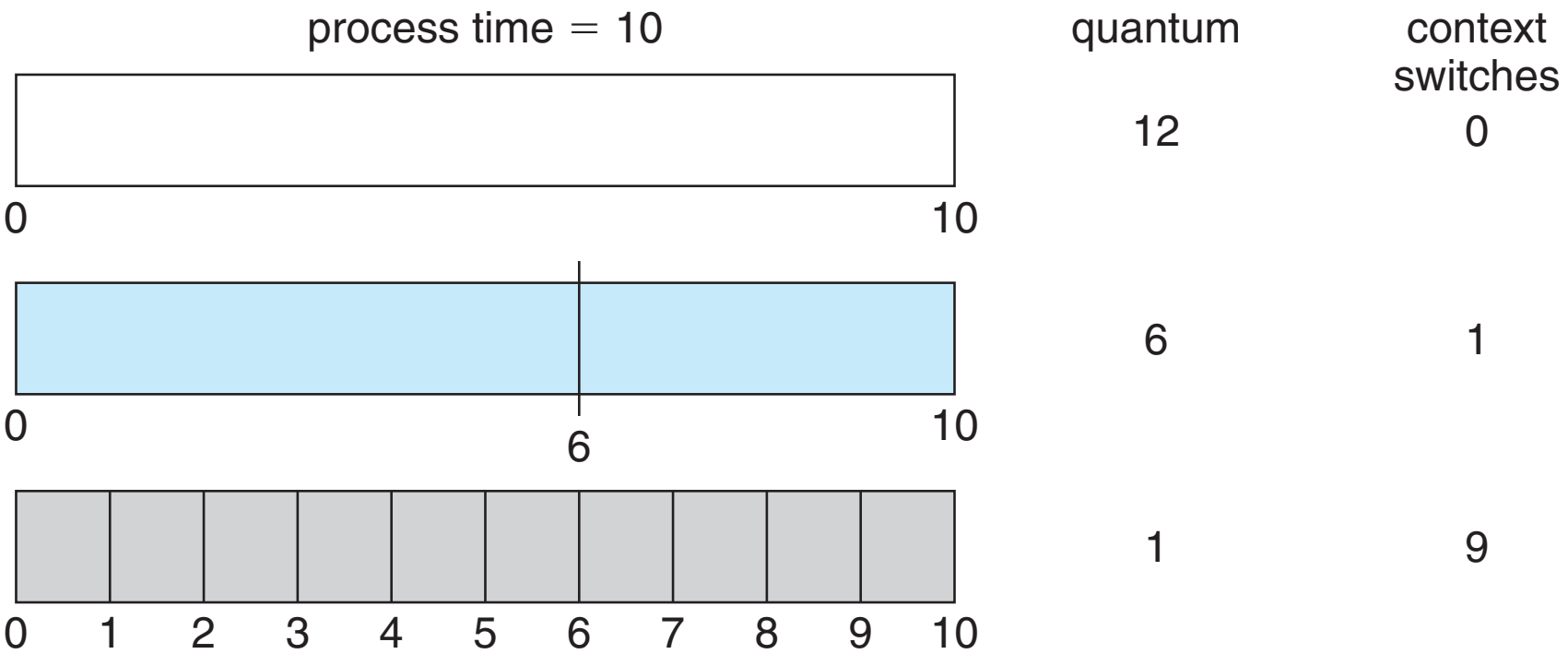

Figure: An illustration showing that smaller time-quantum increases the number of context switches. Image: [SGG13b].

Round Robin (RR)

Proc	Burst-time
P_0	24
P_1	2
P_2	4

Table: An example of processes and their burst-time.

Multilevel Queue

- We have several ready queues.
- Each queue has its own scheduling algorithms.
- On top of that we have scheduling among queues.
- Processes in lower-priority queues are preempted when a process enters in a higher-priority queue.

Multilevel Queue

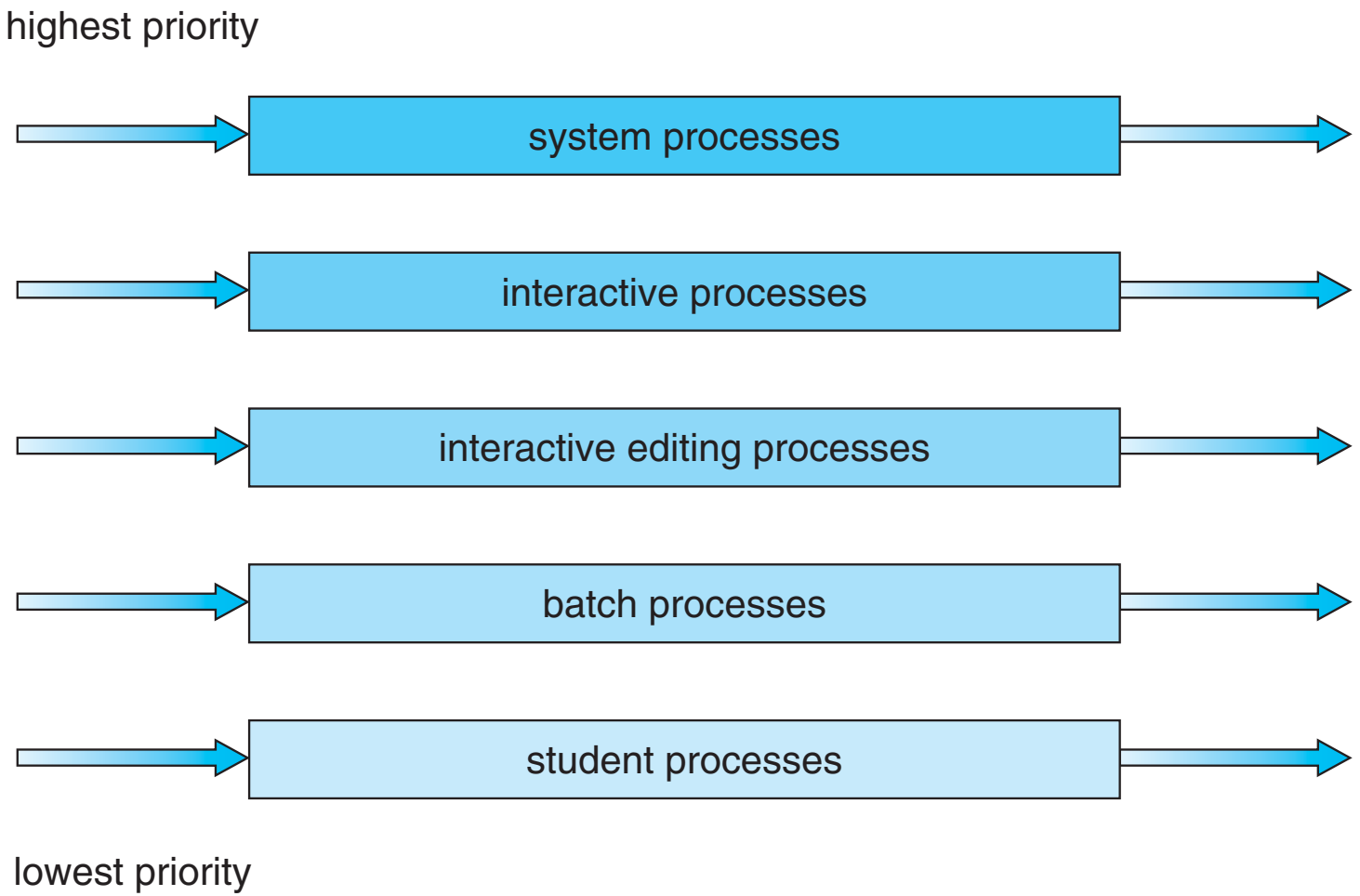

Figure: An example of a multilevel queue structure. Image: [SGG13b].

Multilevel Feedback Queue

- An extension of previous algorithm in that a process may move between queues depending on its CPU-burst characteristic.
- E.g. a process using too much CPU-time will be lowered, a process just doing I/O may be higher.
- When the I/O-bound process becomes a CPU-bound process it will be lowered again.

Multilevel Feedback Queue

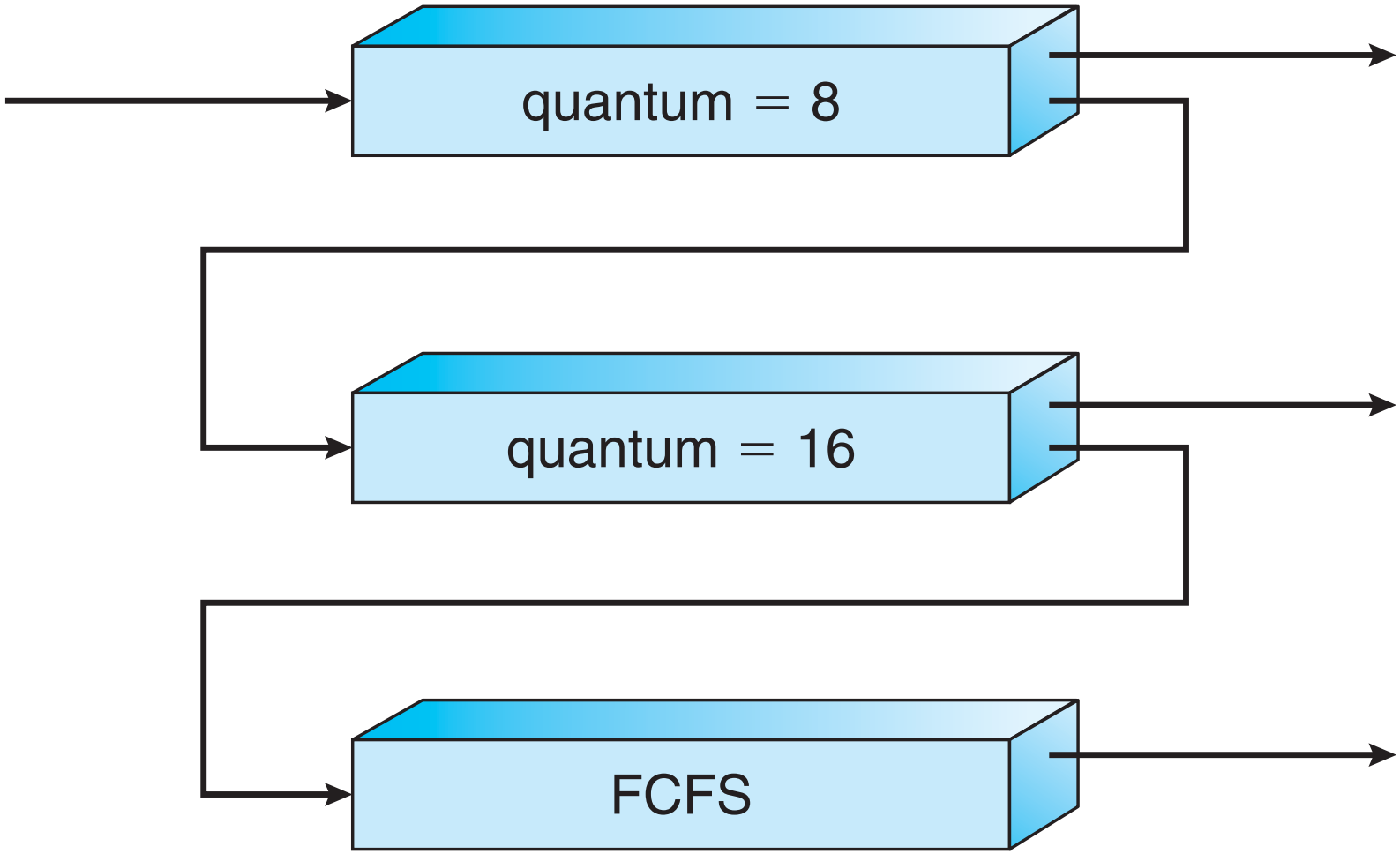

Figure: An example of a multilevel feedback queue system. Image: [SGG13b].

Overview

Asymmetric Multiprocessor Scheduling

- In this kind of setup we have one master server delegating work to the other processors.
- I.e. one processor (the master) does I/O, scheduling, etc. whereas all others execute only user code.

Symmetric Multiprocessor Scheduling

- This is the more common approach.
- All processors are peers, executing everything.
- I.e. all processors does I/O etc.
- Every processor schedules its own execution; has perhaps an own ready queue, perhaps a joint one for all processors.

Symmetric Multiprocessor Scheduling

- One problem which arises with SMP is non-uniform memory access (NUMA), i.e. when different parts of memory is accessed with different speeds.
- Another problem is if a process runs on one processor and is later scheduled on another processor, then the cache is invalid on the previous processor and it must be filled on the new one.
- Hence it's bad to move processes between processors.
- This is called processor affinity.

Symmetric Multiprocessor Scheduling

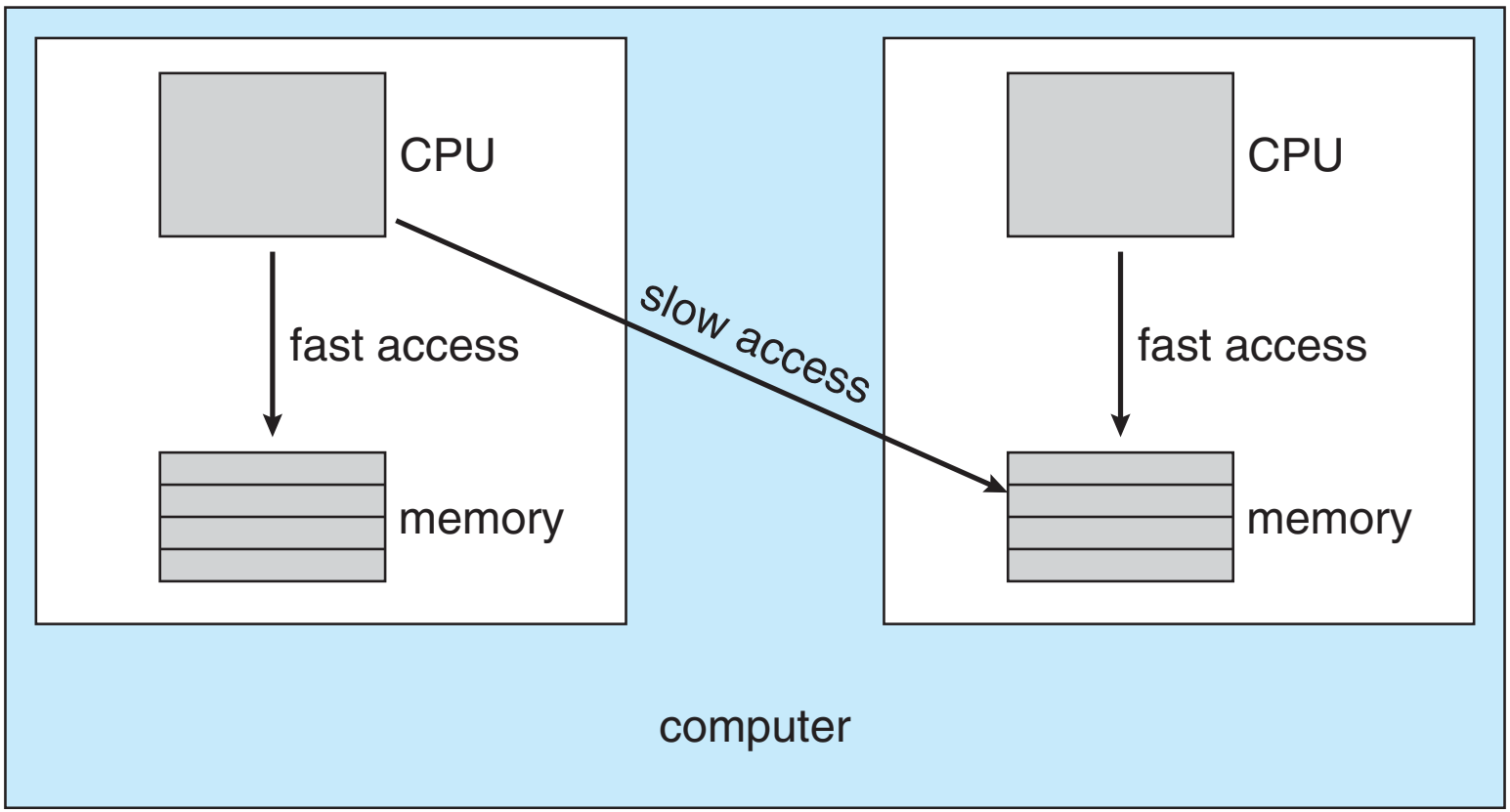

Figure: An illustration of the problem with NUMA. Image: [SGG13b].

Symmetric Multiprocessor Scheduling

- We must have some type of load balancing, otherwise all processes might end up on one processor with all other processors idle.
- We therefore have soft and hard affinity, to allow moves in some cases but not in others.
- Load balancing in SMP can be done with either push or pull migration.
- However, they are not mutually exclusive.
- Carefully considering load balancing and processor affinity is important when designing the operating system, as they counteract each other.

Referenser I

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. 9th ed. International Student Version. Hoboken, N.J.: John Wiley & Sons Inc, 2013.

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. 9th ed. Hoboken, N.J.: John Wiley & Sons Inc, 2013.