UNIT II

THREADS & CPU SCHEDULNG

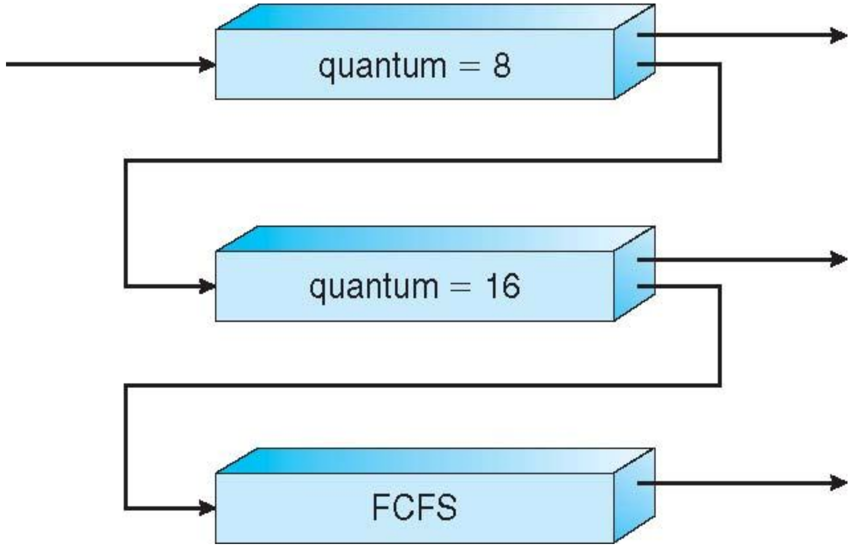
Threads & CPU Schedulng

- **Threads**
  - Overview
  - Multicore Programming
  - Multithreading Models
  - Implicit Threading
  - Threading Issues

- **CPU Scheduling**
  - Basic Concepts
  - Scheduling Criteria
  - Scheduling Algorithms
  - Thread Scheduling
  - Multiple-Processor Scheduling
  - Real-Time CPU Scheduling

Example of Multilevel Feedback Queue

- **Three queues:**
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR 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



Thread Scheduling

- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  - Known as **process-contention scope** (**PCS**) since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope** (**SCS**) – competition among all threads in system

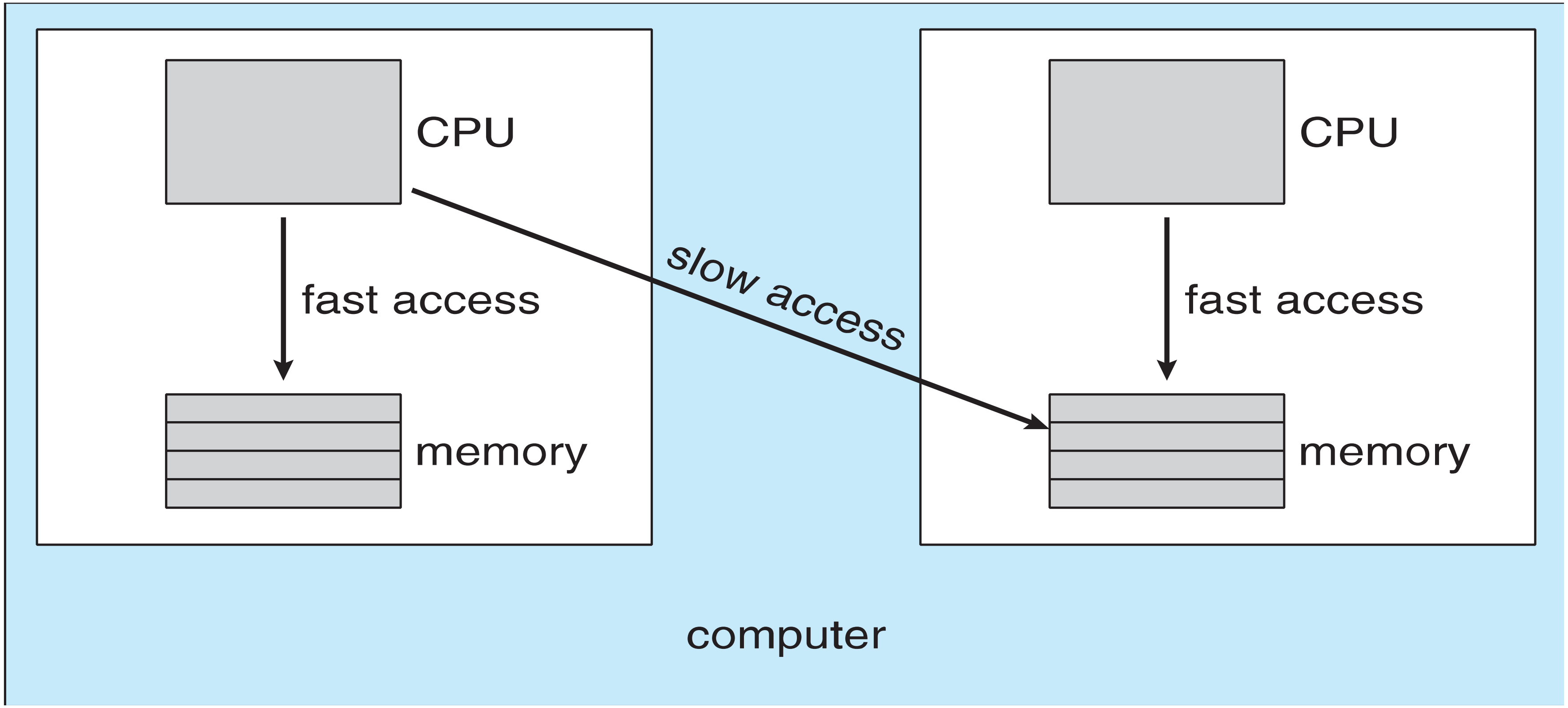
Pthread Scheduling

- API allows specifying either PCS or SCS during thread creation
  - PTHREAD_SCOPE_PROCESS schedules threads using PCS scheduling
  - PTHREAD_SCOPE_SYSTEM schedules threads using SCS scheduling
- Can be limited by OS – Linux and Mac OS X only allow PTHREAD_SCOPE_SYSTEM

Multiple-Processor Scheduling

- 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** – process has affinity for processor on which it is currently running
  - **soft affinity**
  - **hard affinity**
  - Variations including **processor sets**

NUMA and CPU Scheduling



Note that memory-placement algorithms can also consider affinity

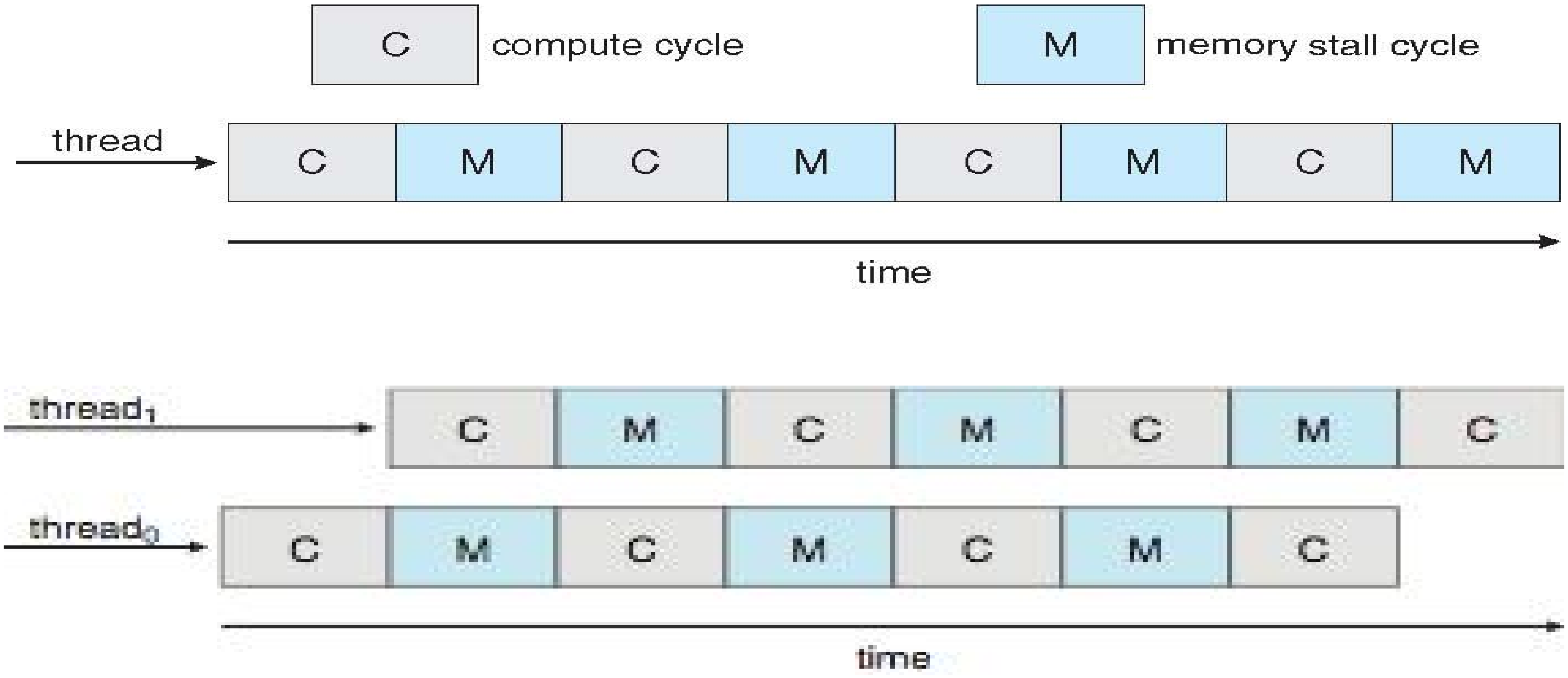
Multiple-Processor Scheduling – Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
- **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- **Pull migration** – idle processors pulls waiting task from busy processor

Multicore Processors

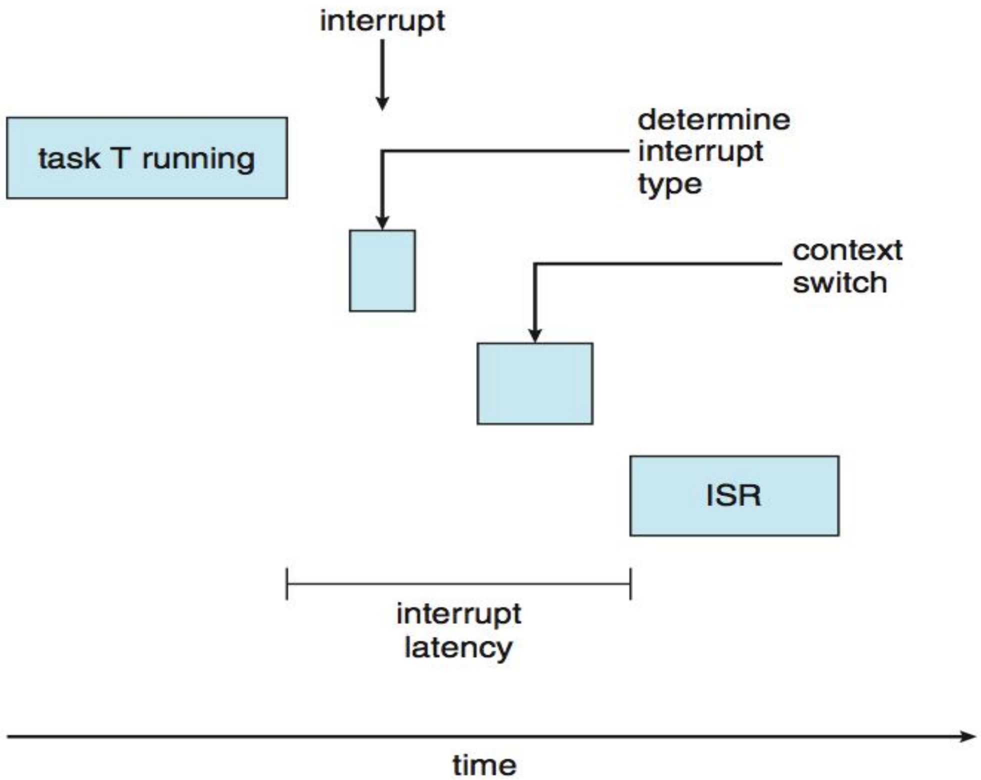
- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

Multithreaded Multicore System



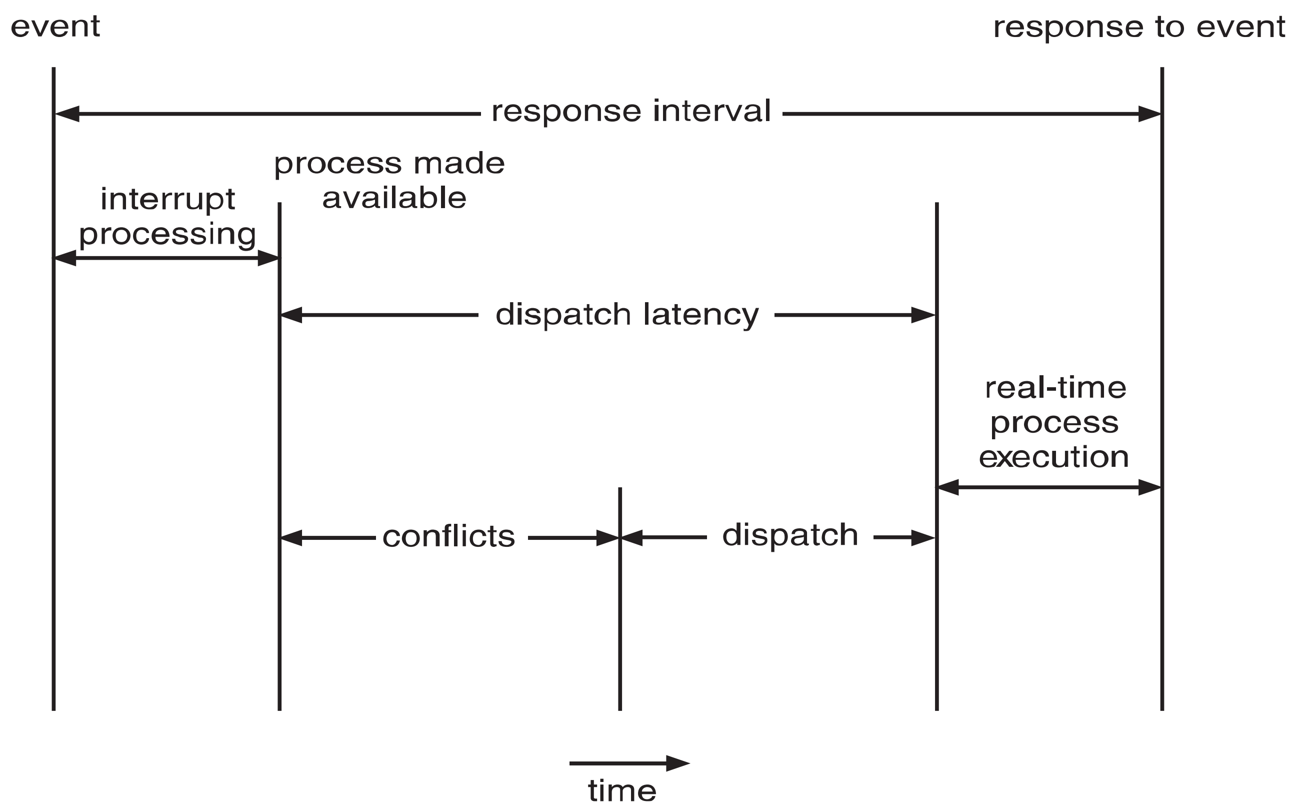
Real-Time CPU Scheduling

- Can present obvious challenges
- **Soft real-time systems** – no guarantee as to when critical real-time process will be scheduled
- **Hard real-time systems** – task must be serviced by its deadline
- Two types of latencies affect performance
  1. **Interrupt latency** – time from arrival of interrupt to start of routine that services interrupt
  2. **Dispatch latency** – time for schedule to take current process off CPU and switch to another



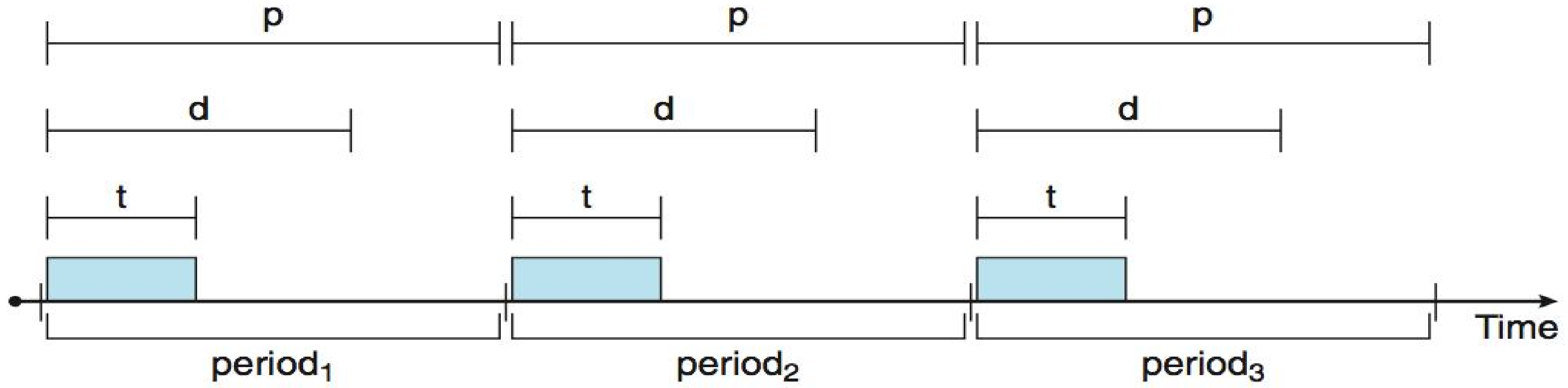
Real-Time CPU Scheduling (Cont.)

- Conflict phase of dispatch latency:
    1. Preemption of any process running in kernel mode
    2. Release by low-priority process of resources needed by high-priority processes



Priority-based Scheduling

- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
- For hard real-time must also provide ability to meet deadlines
- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing time *t*, deadline *d*, period *p*
  - $0 \leq t \leq d \leq p$
  - **Rate** of periodic task is 1/*p*

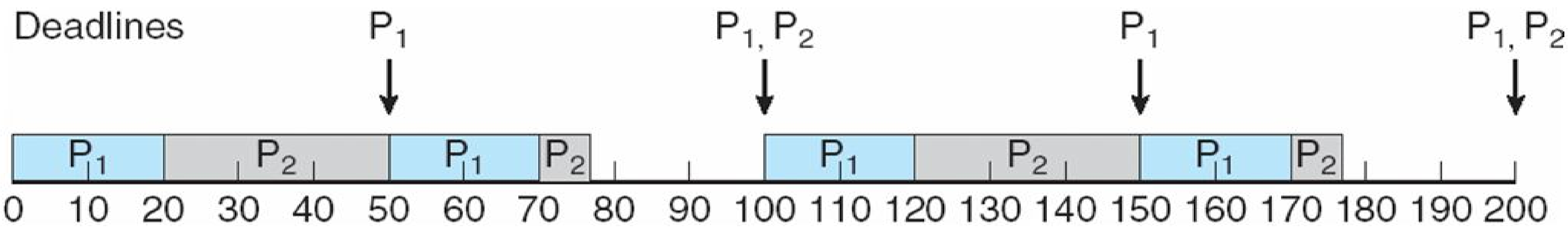


Virtualization and Scheduling

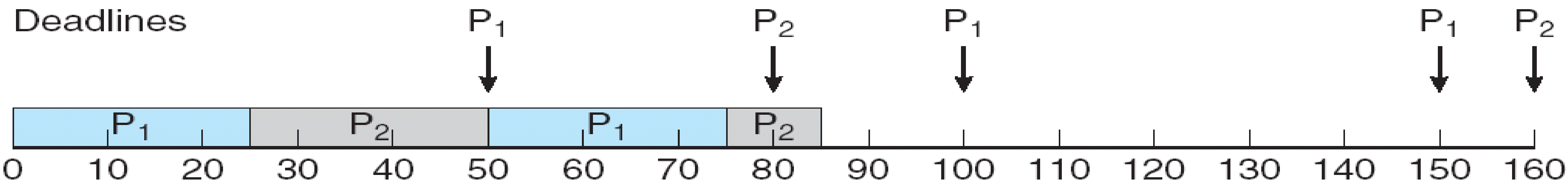
- Virtualization software schedules multiple guests onto CPU(s)
- Each guest doing its own scheduling
  - Not knowing it doesn't own the CPUs
  - Can result in poor response time
  - Can effect time-of-day clocks in guests
- Can undo good scheduling algorithm efforts of guests

Rate Montonic Scheduling

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 is assigned a higher priority than P_2.



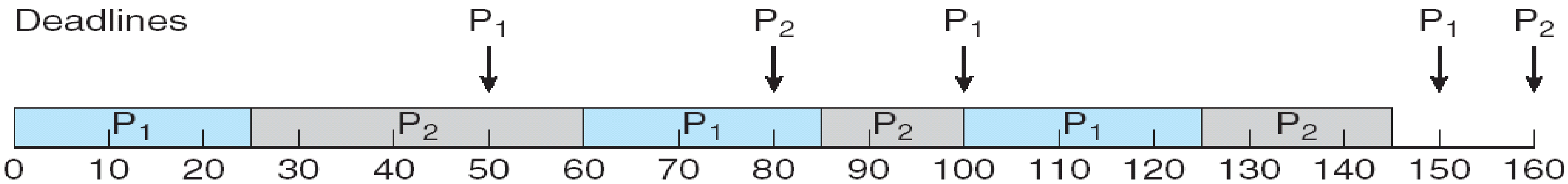
Missed Deadlines with Rate Monotonic Scheduling



Earliest Deadline First Scheduling (EDF)

- Priorities are assigned according to deadlines:

  the earlier the deadline, the higher the priority;
  the later the deadline, the lower the priority



Proportional Share Scheduling

- *T* shares are allocated among all processes in the system
- An application receives *N* shares where $N < T$
- This ensures each application will receive ***N*** / *T* of the total processor time

POSIX Real-Time Scheduling

- The POSIX.1b standard
- API provides functions for managing real-time threads
- Defines two scheduling classes for real-time threads:
- SCHED_FIFO - threads are scheduled using a FCFS strategy with a FIFO queue. There is no time-slicing for threads of equal priority
- SCHED_RR - similar to SCHED_FIFO except time-slicing occurs for threads of equal priority

TEXT BOOK

1. Abraham Silberschatz, Peter B. Galvin, “Operating System Concepts”, 10th Edition, John Wiley & Sons, Inc., 2018.
2. Jane W. and S. Liu. “Real-Time Systems”. Prentice Hall of India 2018.
3. Andrew S Tanenbaum, Herbert Bos, Modern Operating Pearson , 2015.

REFERENCES

1. William Stallings, “Operating Systems: Internals and Design Principles”,9th Edition, Prentice Hall of India., 2018.
2. D.M.Dhamdhere, “Operating Systems: A Concept based Approach”, 3rdEdition, Tata McGraw hill 2016.
3. P.C.Bhatt, “An Introduction to Operating Systems–Concepts and Practice",4th Edition, Prentice Hall of India., 2013.

THANK YOU