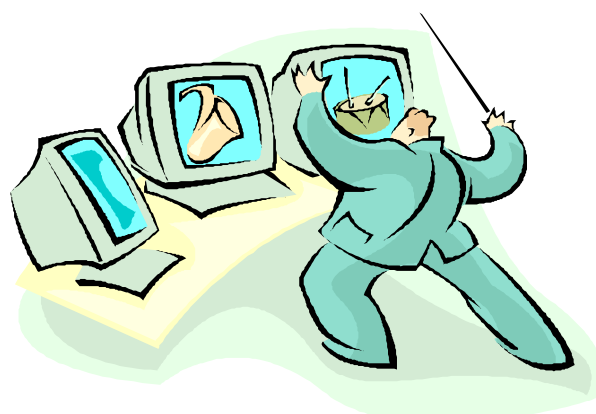
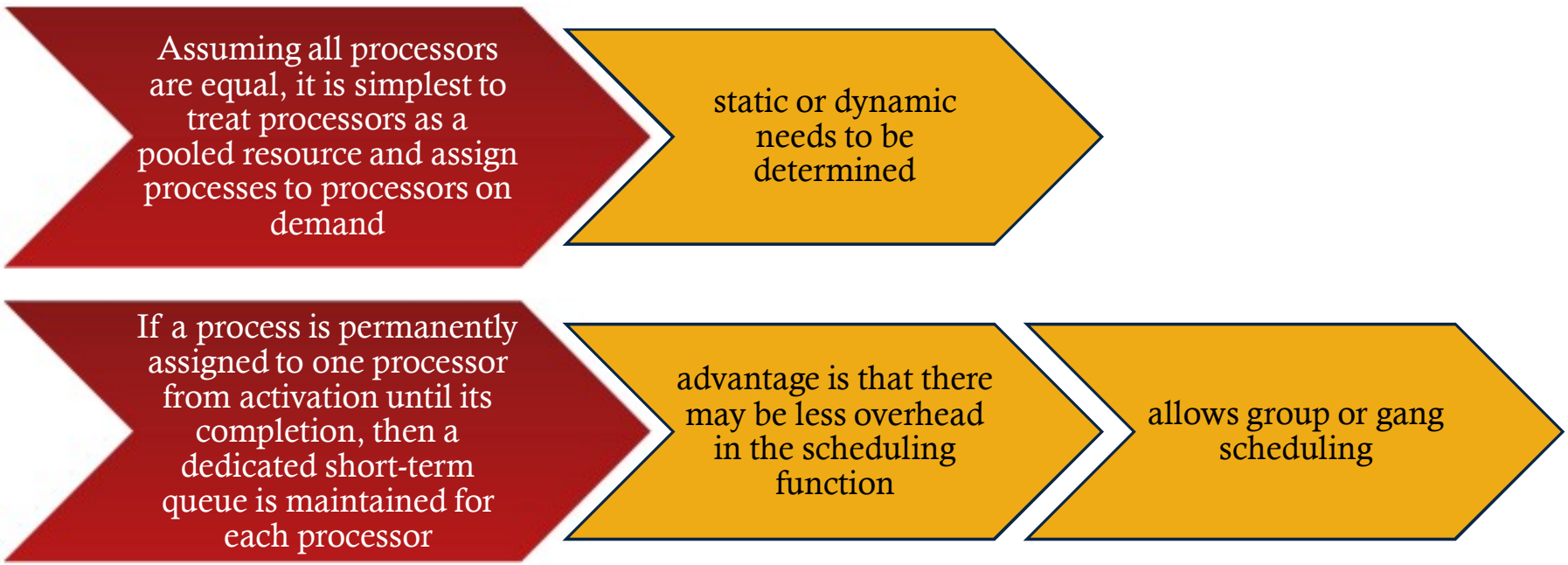
Multiprocessor and Multicore Scheduling

Coarse and Very Coarse Grained Parallelism

- Synchronization among processes, but at a very gross level
- Good for concurrent processes running on a multiprogrammed uniprocessor
  - can be supported on a multiprocessor with little or no change to user software



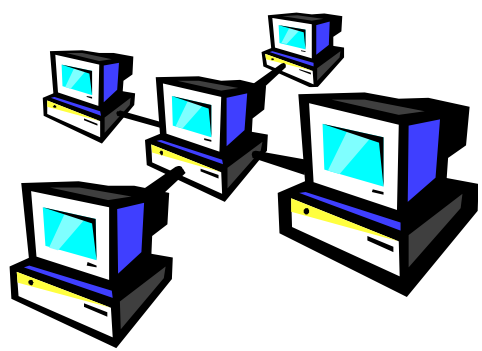
Assignment of Processes to Processors



- A disadvantage of static assignment is that one processor can be idle, with an empty queue, while another processor has a backlog
  - to prevent this situation, a common queue can be used
  - another option is dynamic load balancing

Process Scheduling

- Usually processes are not dedicated to processors
- A single queue is used for all processors
  - if some sort of priority scheme is used, there are multiple queues based on priority
- System is viewed as being a multi-server queuing architecture



Load Sharing

- Simplest approach and carries over most directly from a uniprocessor environment

Advantages:
• load is distributed evenly across the processors • no centralized scheduler required • the global queue can be organized and accessed using any of the schemes discussed in Chapter 9

- Versions of load sharing:
  - first-come-first-served
  - smallest number of threads first
  - preemptive smallest number of threads first

Dedicated Processor Assignment

- When an application is scheduled, each of its threads is assigned to a processor that remains dedicated to that thread until the application runs to completion
- If a thread of an application is blocked waiting for I/O or for synchronization with another thread, then that thread's processor remains idle
  - there is no multiprogramming of processors
- Defense of this strategy:
  - in a highly parallel system, with tens or hundreds of processors, processor utilization is no longer so important as a metric for effectiveness or performance
  - the total avoidance of process switching during the lifetime of a program should result in a substantial speedup of that program

Cache Sharing

Cooperative resource sharing

- Multiple threads access the same set of main memory locations
- Examples:
  - applications that are multithreaded
  - producer-consumer thread interaction

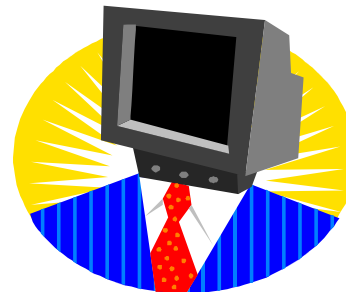
Resource contention

- Threads, if operating on adjacent cores, compete for cache memory locations
- If more of the cache is dynamically allocated to one thread, the competing thread necessarily has less cache space available and thus suffers performance degradation
- Objective of contention-aware scheduling is to allocate threads to cores to maximize the effectiveness of the shared cache memory and minimize the need for off-chip memory accesses

Periodic and Aperiodic Tasks

- **Periodic tasks**
  - requirement may be stated as:
    - once per period T
    - exactly T units apart
- **Aperiodic tasks**
  - has a deadline by which it must finish or start
  - may have a constraint on both start and finish time

User Control

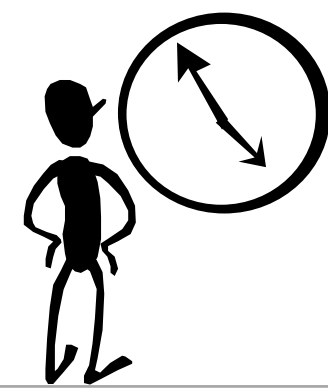


- Generally much broader in a real-time operating system than in ordinary operating systems
- It is essential to allow the user fine-grained control over task priority
- User should be able to distinguish between hard and soft tasks and to specify relative priorities within each class
- May allow user to specify such characteristics as:

Real-Time Scheduling

Scheduling approaches depend on:

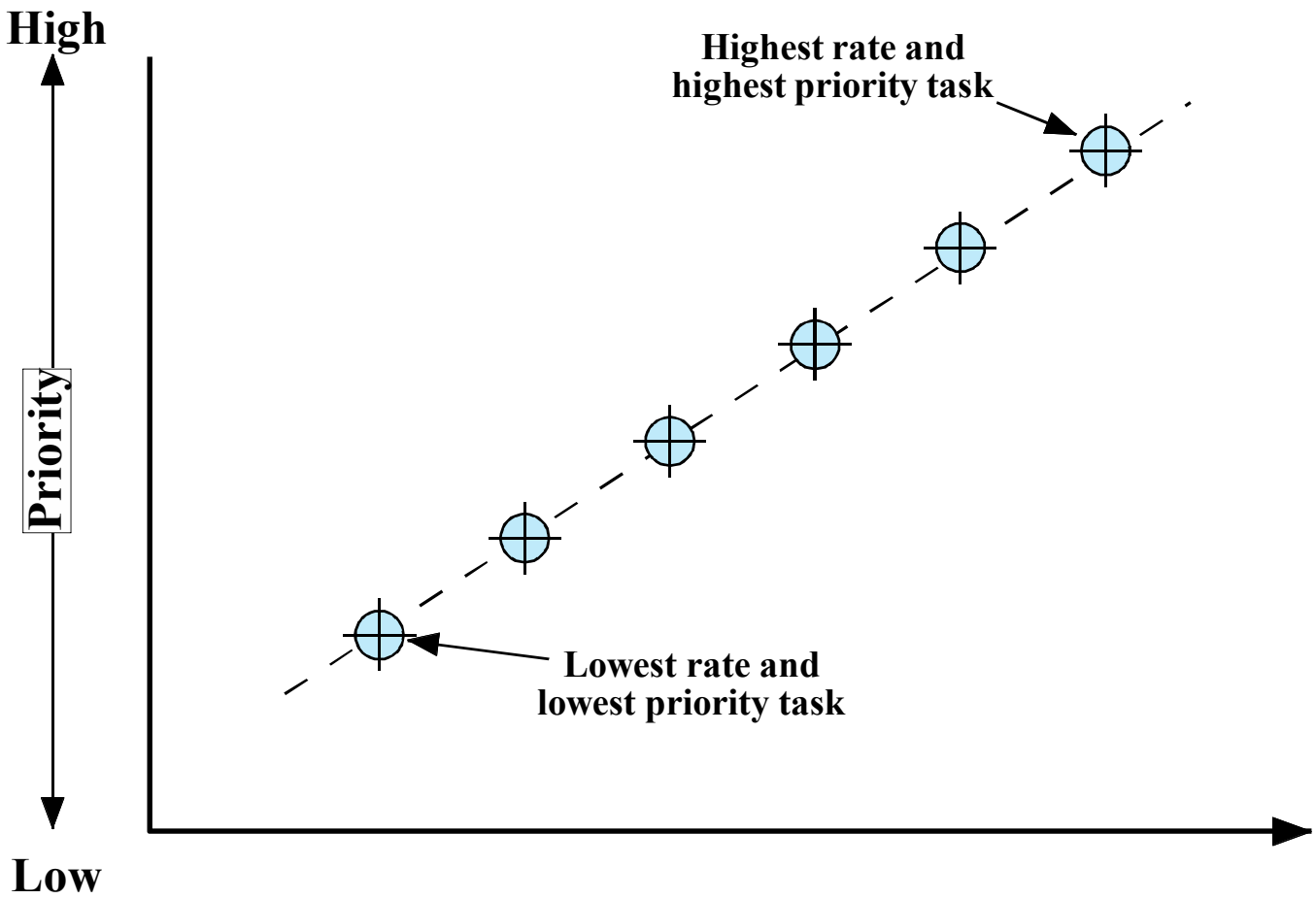
- whether a system performs schedulability analysis
  - if it does, whether it is done statically or dynamically
- whether the result of the analysis itself produces a scheduler plan according to which tasks are dispatched at run time



Execution Profile of Two Periodic Tasks

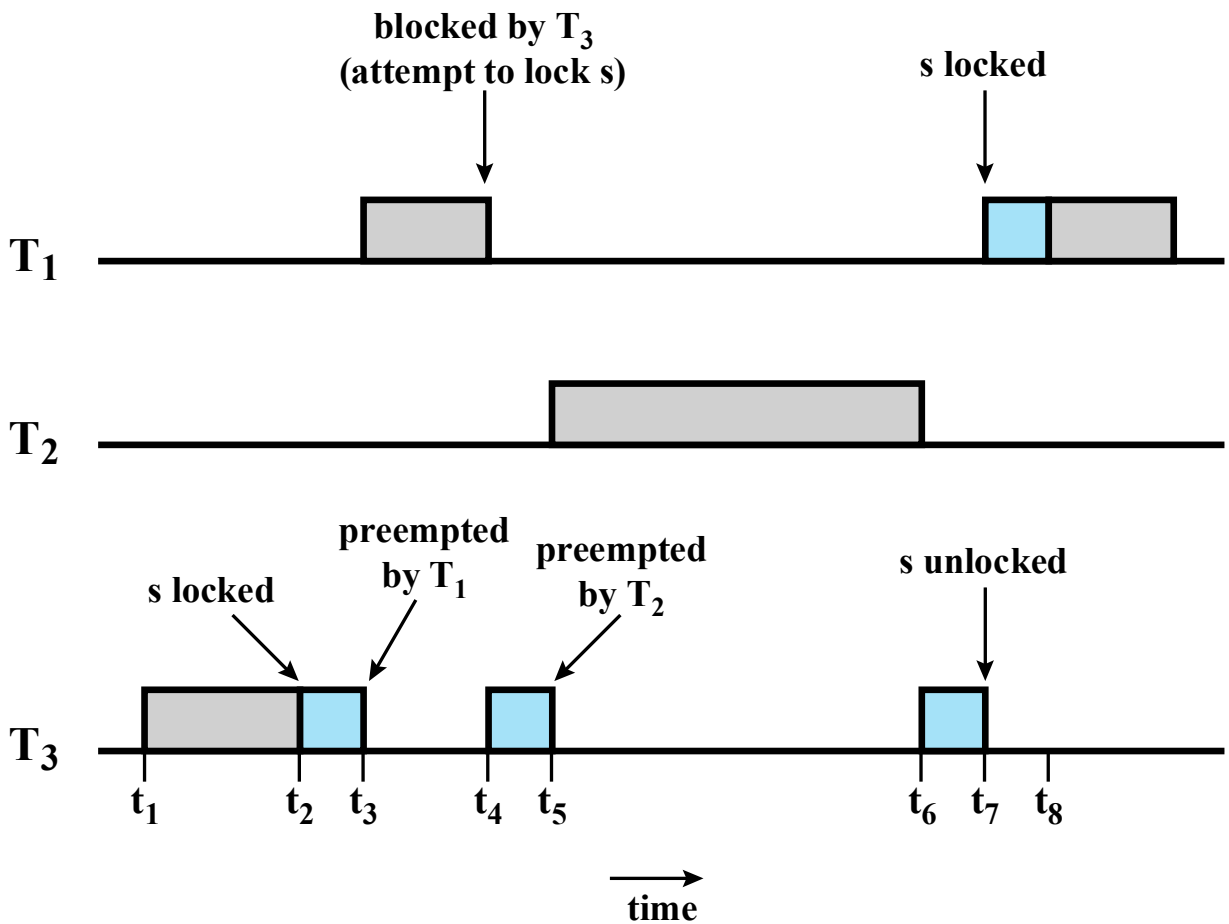
Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
• • •	• • •	• • •	• • •
B(1)	0	25	50
B(2)	50	25	100
• • •	• • •	• • •	• • •

Processor Utilization: are both periodic tasks schedulable in the long run?



A Task Set with RMS

Unbounded Priority Inversion



(a) Unbounded priority inversion