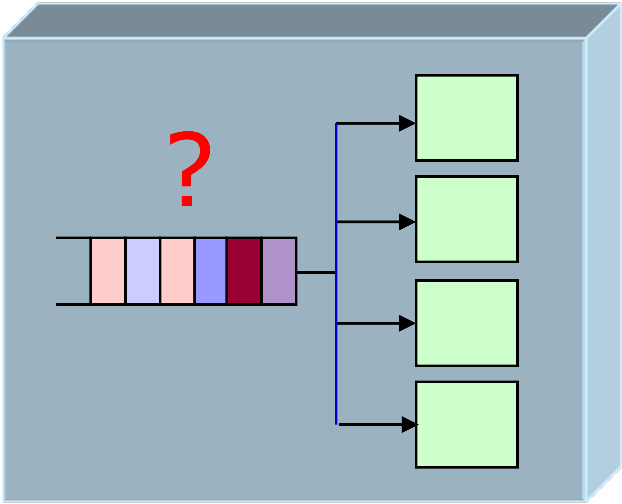
Desirable Properties

1. **Timeliness.** Results have to be correct not only in their value but also in time.
2. **Design for peak load.** Real-time systems must not collapse when they are subject to peak-load conditions.
3. **Predictability.** To guarantee a minimum level of performance, the system must be able to predict the consequences of any scheduling decision.
4. **Fault tolerance.** Single hardware and software failures should not cause the system to crash.
5. **Maintainability.** Possible system modifications are easy to perform.

Task Scheduling Concept

1. Many real-time systems have many tasks to be executed in parallel.
2. An *efficient* scheduling algorithm is used to schedule all tasks that have some jobs ready to run.

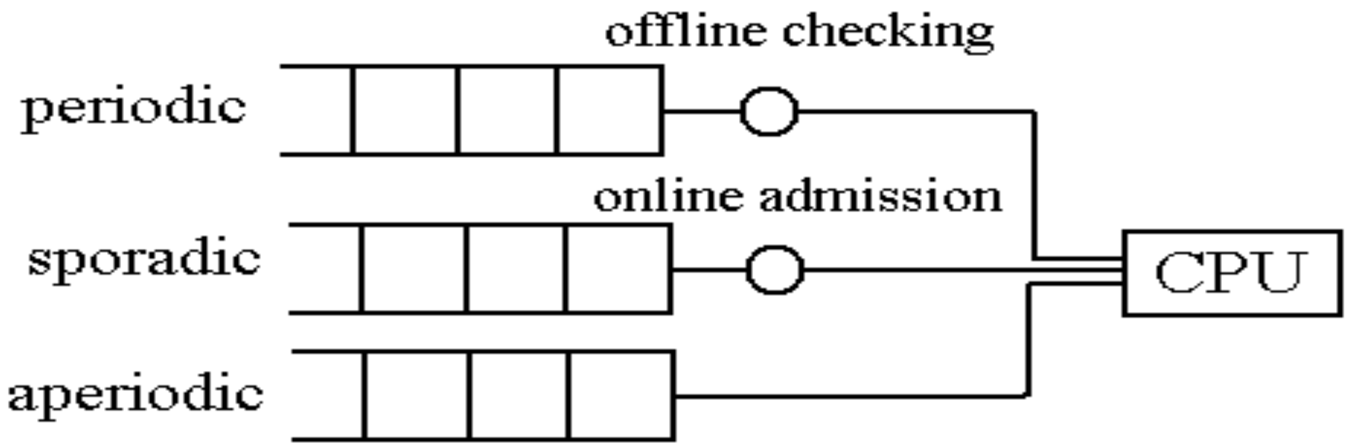
- A single task queue for all jobs that are ready.
- Tasks may *be assigned* to any processor for execution.
- The **BIG** question is how we put jobs in front of the queue.



Real-time Task Models

- The term ‘task’ is a design-time concept. Each task represents a unit of concurrency. A task with real-time constraints is called a real-time task.
- A task can be executed multiple times. Each individual run to completion of a task is called a *job* of that task.
- Periodic tasks: a periodic task can be executed repeatedly on a continuing basis. The inter-arrival times between consecutive jobs are almost the same, which is called its period.
- Sporadic tasks: a sporadic task is executed in response to external events which occur at random instants of time. The inter-arrival times between consecutive jobs may vary widely, and can be arbitrarily small (bursty occurrence).
- Aperiodic tasks: an aperiodic task is also executed in response to external events which may occur at random instants of time. The inter-arrival times of aperiodic jobs may follow some probability distribution function. Aperiodic tasks have either soft or no deadlines.

Scheduling Classes



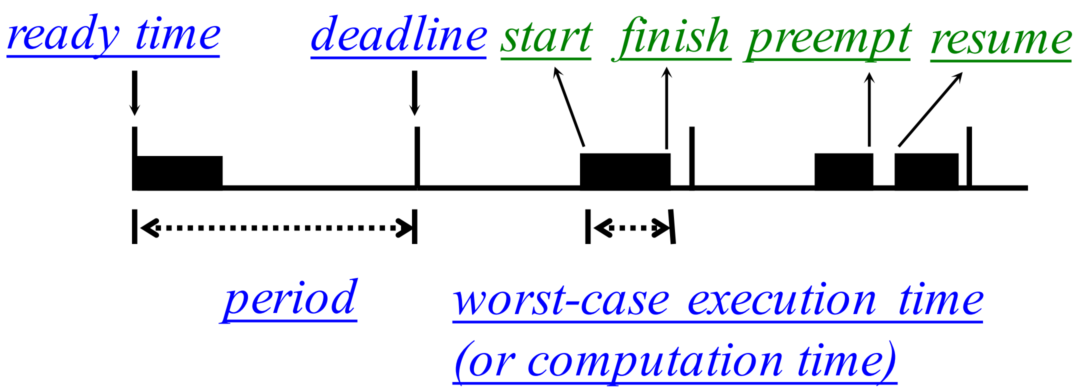
- We could use different schedulers in one real-time system for different classes of tasks

Model: Sampling Rate Decides Period

- Sampling rate
  - An application specification (e.g. 30 frames/sec, 6000 RPM)
- Period
  - How often a function is executed
- Relative deadline
  - The deadline with respect to when the function is ready for execution.

Periodic Job Scheduling

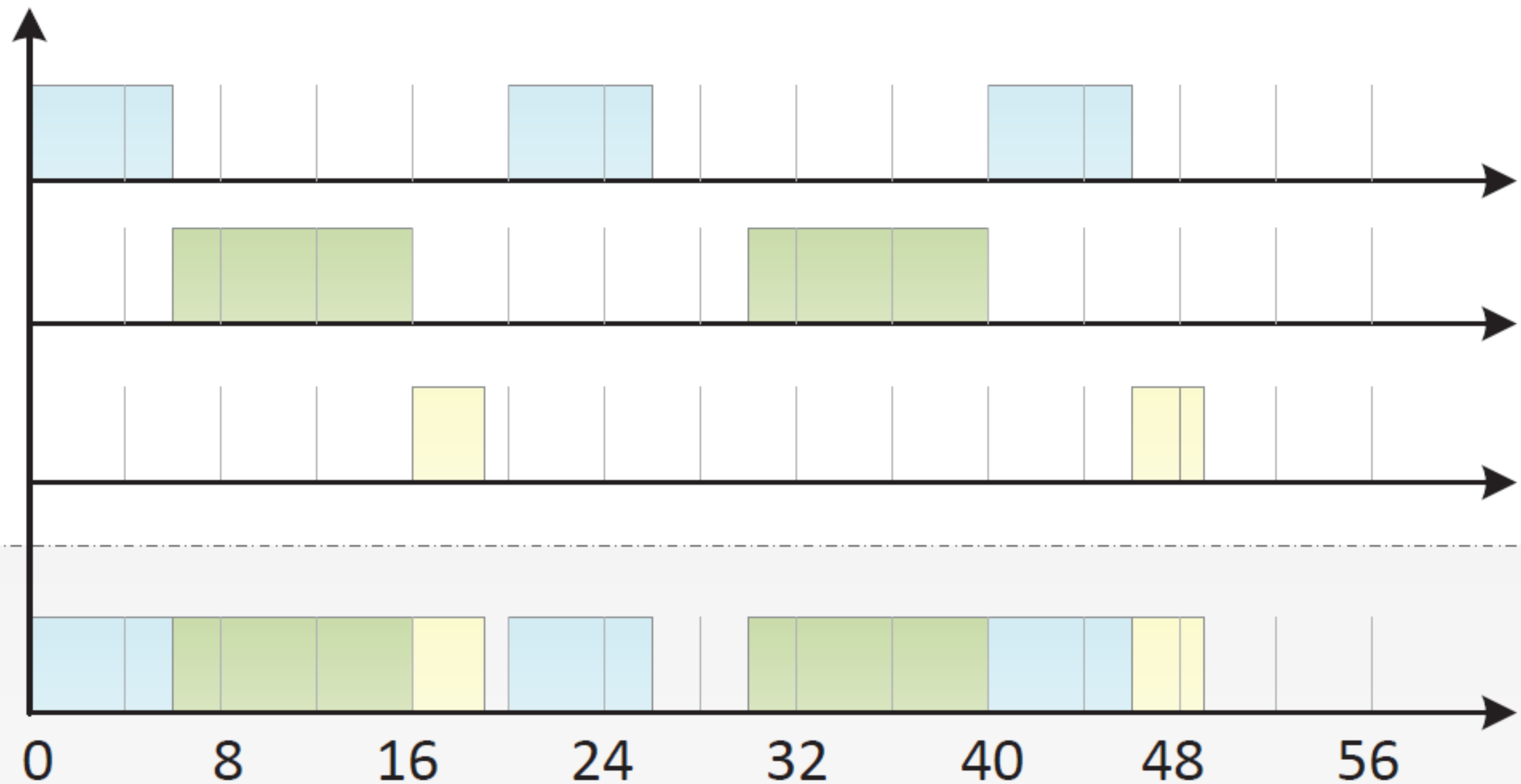
- Periodic Task Model
  - parameters are known *a priori*



- Predictability vs. Schedulability
  - high predictability and flexibility
  - easy-to-check schedulability condition

A timing diagram showing the design of task schedule

Task A: period=20ms, execution time=6ms, deadline=20ms
Task B: period=30ms, execution time=10ms,deadline=30ms
Task C: period=40ms, execution time=3ms, deadline=40ms



Task schedule

Periodic Task vs. Job

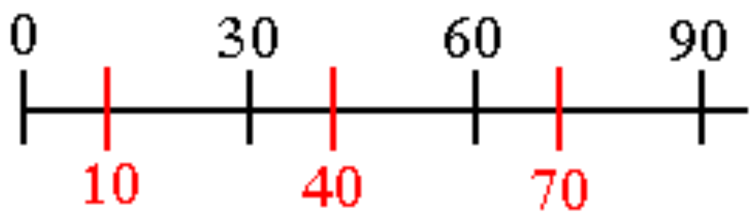
- A periodic task is a stream of jobs (execution time, period)
- $T_1 = (3,10) = \{J_{11} \quad J_{12} \quad J_{13} \ldots\}$
- The time each job is eligible for execution is:
  - (0,10] (10,20] (20,30]

Essential Real-Time Attributes

1. Deadline
2. Consequence (criticality)
   - Hard vs. Soft
3. Usefulness
4. Execution time

Deadline

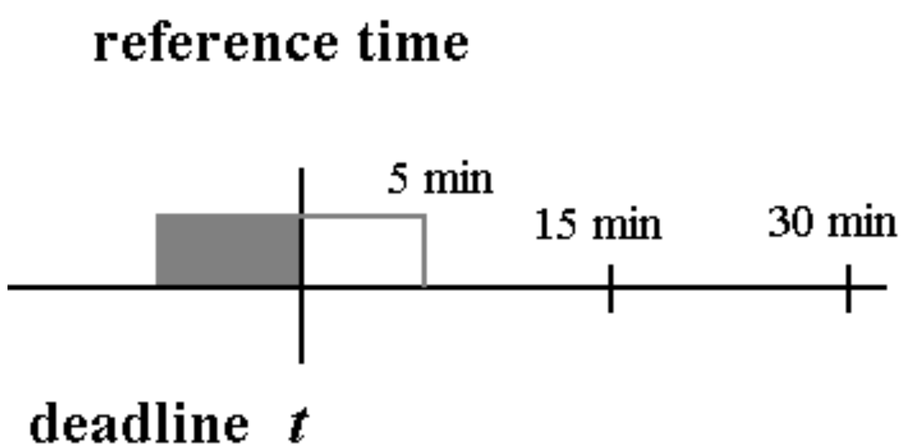
- If a task’s period is 30 ms, and the *relative* deadline is 10 ms, then the *absolute* deadlines are 10 ms, 40 ms, 70 ms…



- The relative deadline cannot be less than the computation, or execution time

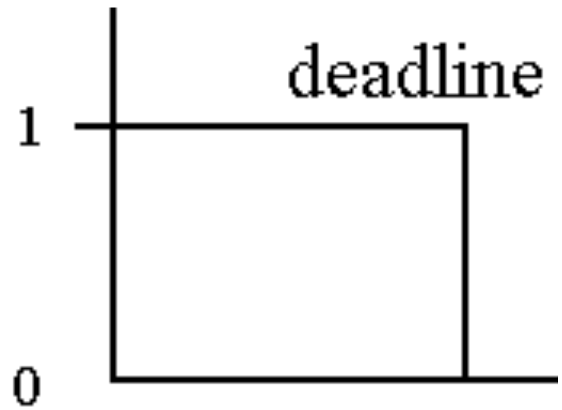
Criticality

- A *hard* deadline must be completed, or a serious consequence will follow
  - Airplane instruments computing altitude
  - Air-bags in cars
- A *soft* deadline may be missed
  - How long are you willing to wait for your taxi reservation?



Value

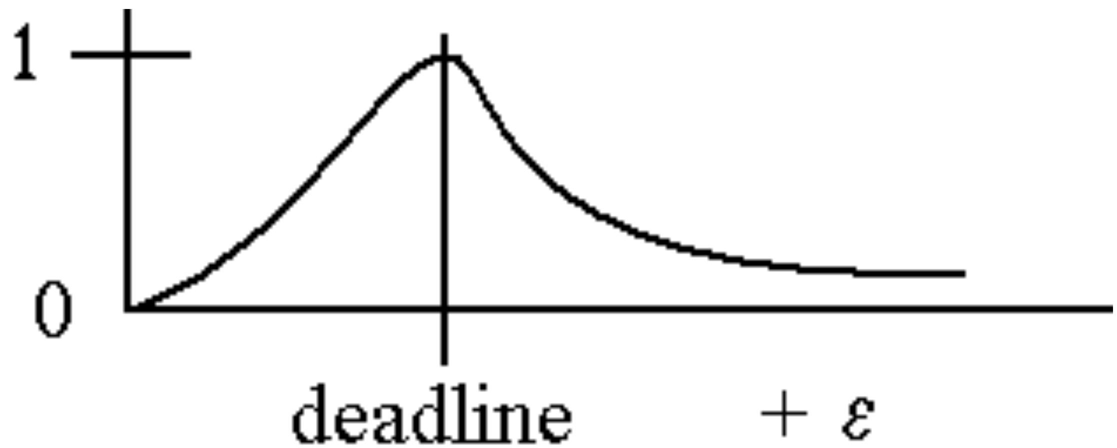
- Most systems are a step function of value, with 1 for meeting the deadline and 0 for failure.



- Value of the success for a hard deadline is 0 or 1

Value (cont.)

- A soft deadline can have a range of values.
  - Completing before the deadline may not be advantageous.



Execution Time Issues

- Execution time is not constant
  - Load on the system
  - Interrupts and exception handling
  - Processor speed
  - OS/middleware overheads
  - Architecture: Cache, Pipeline, etc
- Execution time has a distribution
- We assume there is a maximum execution time

Worst-Case Task Execution Time

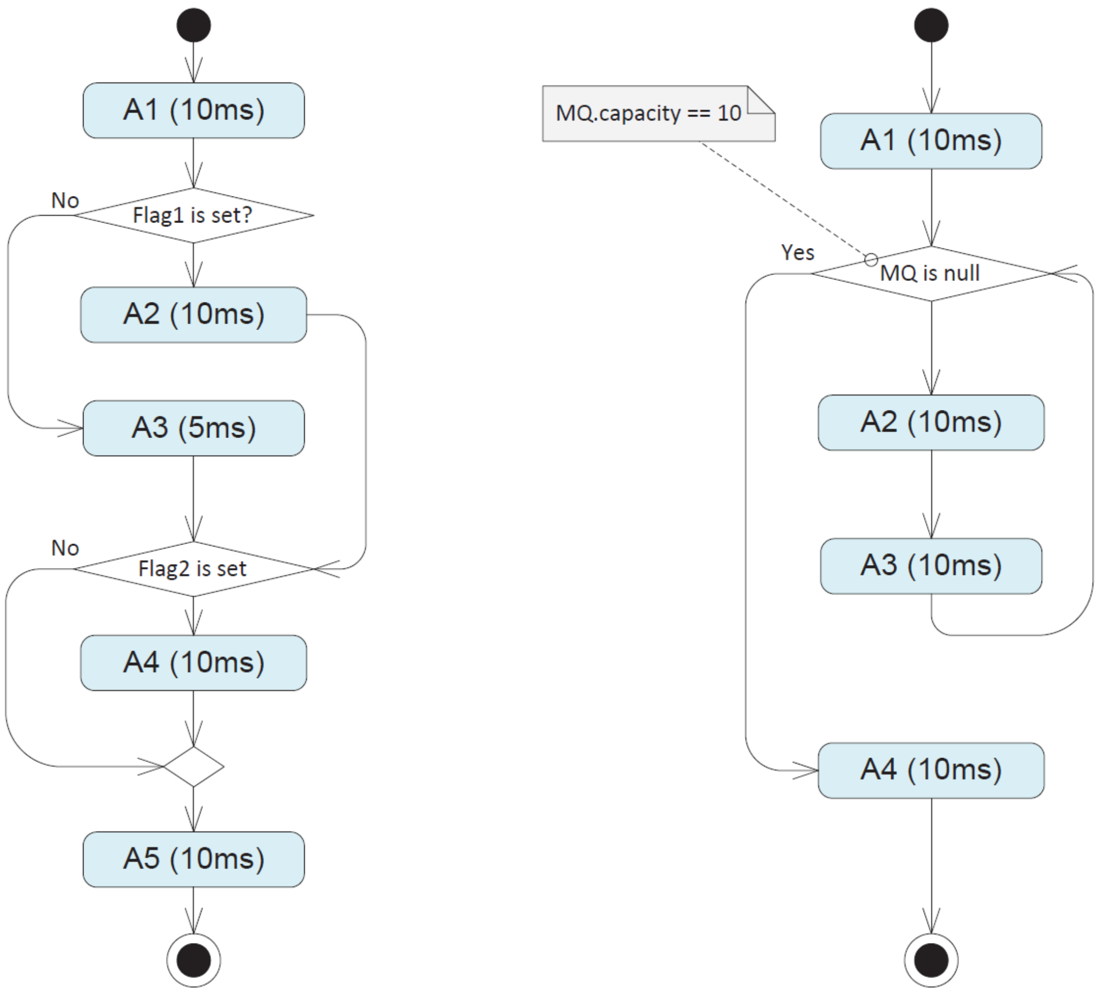


Fig. 12.1 The task shown on the left has a worst-case execution time of 40 ms, the task shown on the right has a worst-case execution time of 220 ms.

Study of Scheduling Algorithms

- Many process scheduling algorithms can be used by an OS kernel
  - FCFS, RR, EDF, Priority Driven…
- A kernel runs a variety of applications, and the performance of the scheduling algorithm depends on the application
  - Benchmark, Simulation
  - Analysis- Queuing theory gives averages, but can't guarantee a deadline
  - Validation- find the worst-case response time

Optimality

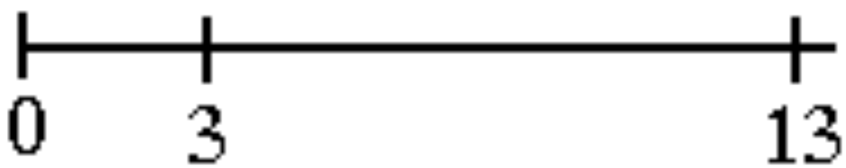
- Validation is a test that returns true or false
- "Optimality"
  - Create a set of tasks, and see which algorithm returns the most number of TRUE's
- If G_1 passes $S_1 = \{S_i\}$ and G_2 passes $S_2 = \{S_i\}$, then G_2 is better that G_1 if $S_2 \supset S_1$.

Algorithm Classification (I)

- **Preemptive.** With preemptive algorithms, the running task can be interrupted at any time to assign the processor to another active task, according to a predefined scheduling policy.
- **Non-preemptive.** With non-preemptive algorithms, a task, once started, is executed by the processor until completion. In this case, all scheduling decisions are taken as a task terminates its execution.

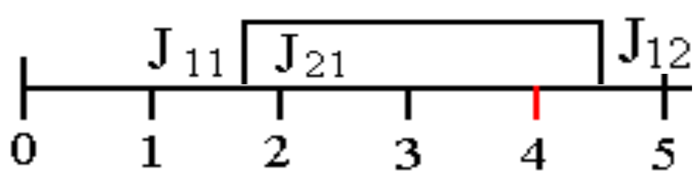
Preemptive Execution

- Preemptive execution: a job execution can be preempted by other jobs
  - The CPU is usually preemptive
  - A resource such as a printer usually is not.
- Phase: the first time a task becomes ready
  - Phase = 3:



Non-Preemptive Scheduling

- Non-preemptive scheduling is risky
  - Like checkout lines in a supermarket, you may wait for a long time if you don't know how long the person in front of you will take
  - J_{12} = (arrival=2, deadline=4, computation time = 1)
  - J_{21} = (arrival=1.5, deadline=7, computation time=3)



Algorithm Classification (II)

- **Static.** Static algorithms are those in which scheduling decisions arc based on fixed parameters, assigned to tasks *before* their activation.

- **Dynamic.** Dynamic algorithms are those in which scheduling decisions are based on dynamic parameters that may change *during* system execution.

Algorithm Classification (III)

- **Online.** We say that a scheduling algorithm is used online if scheduling decisions are taken at *runtime* every time a new task enters the system or when a running task terminates.

- **Offline.** We say that a scheduling algorithm is used offline; if it is executed on the entire task set *before* actual task activation. The schedule generated in this way is stored in a table and later executed by a dispatcher.

Algorithm Classification (IV)

- **Optimal**. An algorithm is said to be optimal if it minimizes some cost function defined over the task set. When no cost function is defined and the only concern is to achieve a feasible schedule, then an algorithm is said to be optimal if it always finds a feasible schedule *whenever one exists*

- **Heuristic**. An algorithm is said to be heuristic if it searches for a feasible schedule using an objective function *(heuristic function)*. Heuristic algorithm *do not guarantee* to find the optimal schedule, even if there exists one

Algorithm Classification (V): 1 vs. Multi-processor

- Migration: if a job is waiting for a resource and goes back to the queue, when it comes time to run the job again, it can go to any processor.
- Parallel execution: can parts of the job be executed simultaneously?
- Typically, allow migration but not parallel execution