A Reference Model for Real-Time Systems

Embedded Real-Time Software
Lecture 2

Lecture Outline

- **Why a reference model?**
- **Jobs and tasks**
- **Processors and resources**
- **Time and timing constraints**
  - Hard real-time
  - Soft real-time
- **Periodic, aperiodic and sporadic tasks**
- **Precedence constraints and dependencies**
- **Scheduling**

Material corresponds to chapters 2 and 3 of Liu's book

A Reference Model of Real-Time Systems

- **Want to develop a model to let us reason about the timing behavior of the systems**
  - Consistent terminology
  - Lets us to focus on the important aspects of a system while ignoring the irrelevant properties and details
- **Our reference model is characterized by:**
  - A workload model that describes the applications supported by the system
  - A resource model that describes the system resources available to the applications
  - Algorithms that define how the application system uses the resources at all times
- **Today: focus on the first two elements of the reference model**
  - The remainder of the module will study the algorithms, using the definitions from this lecture

Jobs and Tasks

- **A *job* is a unit of work that is scheduled and executed by a system**
  - e.g. computation of a control-law, computation of an FFT on sensor data, transmission of a data packet, retrieval of a file
- **A *task* is a set of related jobs which jointly provide some function**
  - e.g. the set of jobs that constitute the "maintain constant altitude" task, keeping an airplane flying at a constant altitude

Processors

- **A job executes – or is executed by the operating system – on a processor and may depend on some resources**
- **A processor, *P*, is an active component on which jobs scheduled**
  - Examples:
    - Threads scheduled on a CPU
    - Data scheduled on a transmission link
    - Read/write requests scheduled to a disk
    - Transactions scheduled on a database server
  - Each processor has a speed attribute which determines the rate of progress a job makes toward completion
    - May represent instructions-per-second for a CPU, bandwidth of a network, etc.
  - Two processors are of the same type if they are functionally identical and can be used interchangeably

Resources

- **A resource, *R*, is a passive entity upon which jobs may depend**
  – E.g. memory, sequence numbers, mutexes, database locks, etc.
  – Resources have different *types* and *sizes,* but do not have a *speed* attribute
  – Resources are usually *reusable*, and are not consumed by use

Use of Resources

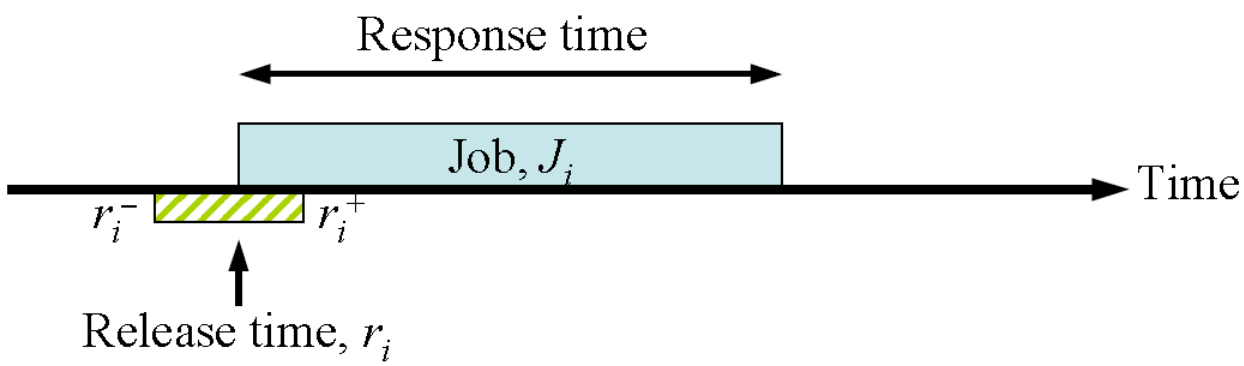
- **If the system contains ρ ("rho") types of resource, this means:**
  - There are ρ different types of *serially reusable* resources
  - There are one or more units of each type of resource, only one job can use each unit at once (mutually exclusive access)
  - A job must obtain a unit of a needed resource, use it, then release it
- **A resource is *plentiful* if no job is ever prevented from executing by the unavailability of units of the resource**
  - Jobs never block when attempting to obtain a unit of a plentiful resource
  - We typically omit such resources from our discussion, since they don't impact performance or correctness

Execution Time

- **A job J_i *will execute for time* e_i**
  - This is the amount of time required to complete the execution of when it executes J_i alone and has all the resources it needs
  - Value of e_i depends upon complexity of the job and speed of the processor on which it is scheduled; may change for a variety of reasons:
    - Conditional branches
    - Cache memories and/or pipelines
    - Compression (e.g. MPEG video frames)
  - Execution times fall into an interval $[e_i^-, e_i^+]$; assume that we know this interval for every hard real-time job, but not necessarily the actual e_i
    - Terminology: $(x, y]$ is an interval starting immediately after x, continuing up to and including y
- **Often, we can validate a system using e_i^+ for each job; we assume and ignore the interval lower bound $[e_i , e_i^+]$**
  - Inefficient, but safe bound on execution time

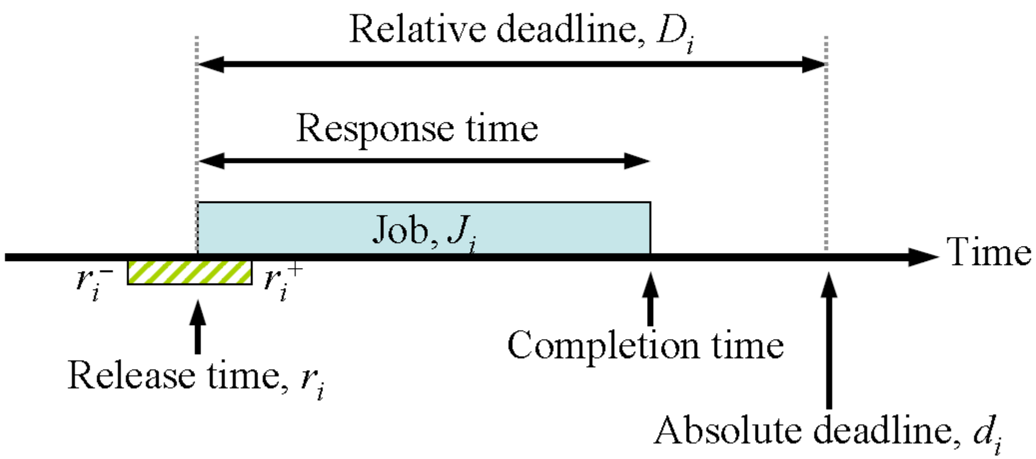
Release and Response Time

- ***Release time*** **– the instant in time when a job becomes available for execution**
  - May not be exact: *Release time jitter* so r_i is in the interval $[r_i^-, r_i^+]$
  - A job can be scheduled and executed at any time at, or after, its release time, provided its resource dependency conditions are met
- ***Response time*** **– the length of time from the release time of the job to the time instant when it completes**
  - Not the same as execution time, since may not execute continually



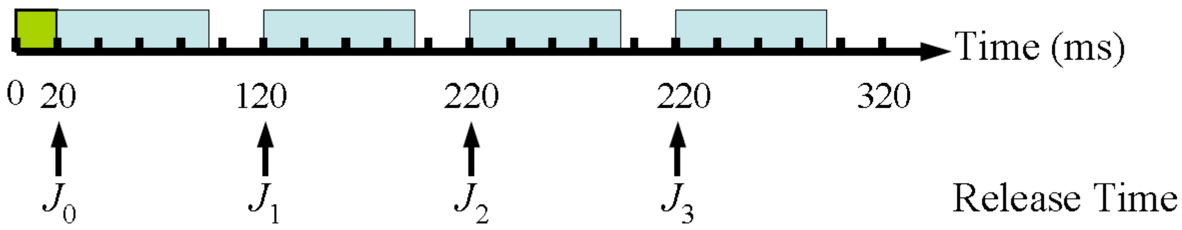
Deadlines and Timing Constraints

- ***Completion time*** **– the instant at which a job completes execution**
- ***Relative deadline*** **– the maximum allowable job response time**
- ***Absolute deadline*** **– the instant of time by which a job is required to be completed (often called simply the *deadline*)**
  - *absolute deadline = release time + relative deadline*
  - *Feasible interval for a job J_i is the interval* (r_i, d_i]
- **Deadlines are examples of *timing constraints***



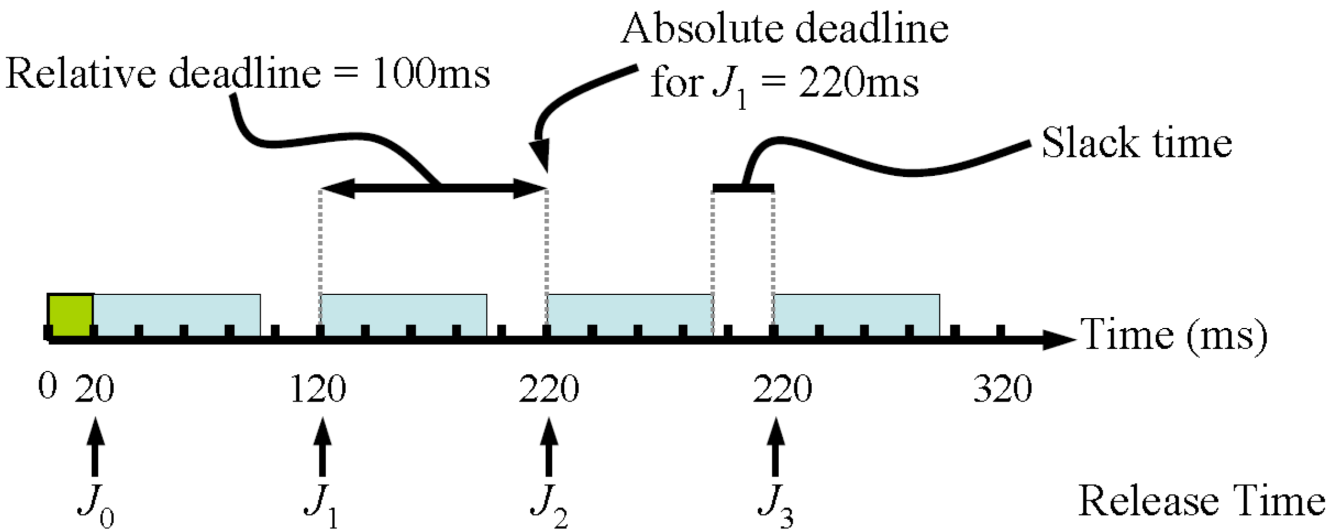
Example

- **A system to monitor and control a heating furnace**
- **The system takes 20ms to initialize when turned on**
- **After initialization, every 100 ms, the system:**
  - Samples and reads the temperature sensor
  - Computes the control-law for the furnace to process temperature readings, determine the correct flow rates of fuel, air and coolant
  - Adjusts flow rates to match computed values
- **The periodic computations can be stated in terms of release times of the jobs computing the control-law: J_0, J_1, …, J_k, …**
  - The release time of J_k is 20 + (k × 100) ms



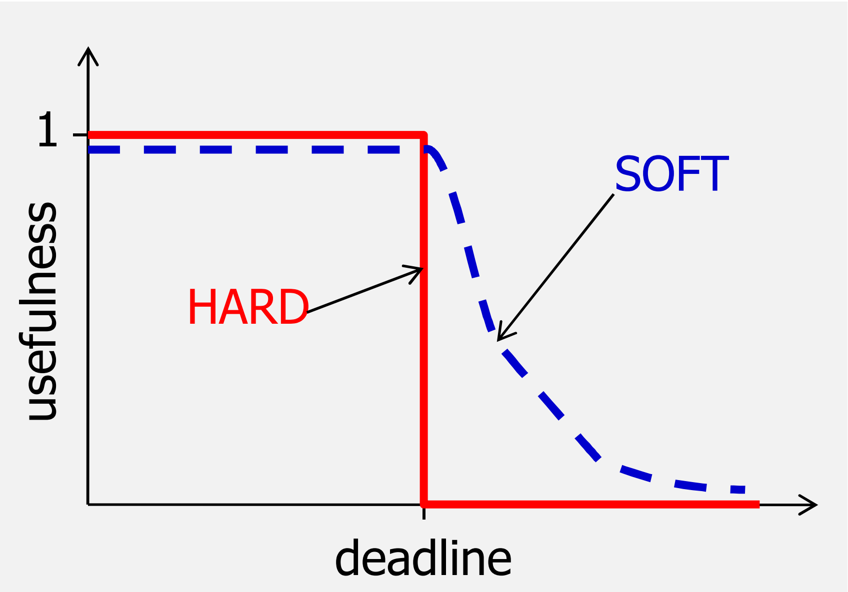
Example

- **Suppose each job must complete before the release of the next job:**
  - J_k' s relative deadline is 100 ms
  - J_k' s absolute deadline is $20 + ((k + 1) \times 100)$ ms
- **Alternatively, each control-law computation may be required to finish sooner – i.e. the relative deadline is smaller than the time between jobs, allowing some *slack time* for other jobs**
  - Slack time : the difference between the completion time and the earliest possible completion time



Hard vs. Soft Real-Time Systems

- **The firmness of timing constraints affects how we reason about, and engineer, the system**
- **If a job must never miss its deadline, then the system is described as *hard real-time***
  - A timing constraint is hard if the failure to meet it is considered a fatal error; this definition is based upon the functional criticality of a job
  - A timing constraint is hard if the usefulness of the results falls off abruptly (or may even go negative) at the deadline
  - A timing constraint is hard if the user requires *validation* (formal proof or exhaustive simulation) that the system always meets its timing constraint



- **If some deadlines can be missed occasionally, with acceptably low probability, then the system is described as *soft real-time***
  - This is a *statistical constraint*

Hard vs. Soft Real-Time Systems

- **Note: there may be no advantage in completing a job early**
  - It is often better to keep *jitter* (variation in timing) in the response times of a stream of jobs small
- **Timing constraints can be expressed in many ways:**
  - Deterministic
    - e.g. the relative deadline of every control-law computation is 50 ms; the response time of at most 1 out of 5 consecutive control-law computations exceeds 50ms
  - Probabilistic
    - e.g. the probability of the response time exceeding 50 ms is less than 0.2
  - In terms of some usefulness function
    - e.g. the usefulness of every control-law computation is at least 0.8

  [In practice, usually *deterministic* constraints, since easy to validate]

Examples: Hard & Soft Real-Time Systems

- **Hard real-time:**
  - Flight control
  - Railway signaling
  - Anti-lock brakes
  - Etc.
- **Soft real-time:**
  - Stock trading system
  - DVD player
  - Mobile phone
  - Etc.

Can you think of more examples?

Is the distinction always clear cut?

Types of Task

- **There are various types of task**
  - Periodic
  - Aperiodic
  - Sporadic

- **Different execution time patterns for the jobs in the task**
- **Must be modeled differently**
  - Differing scheduling algorithms
  - Differing impact on system performance
  - Differing constraints on scheduling

Periodic Tasks

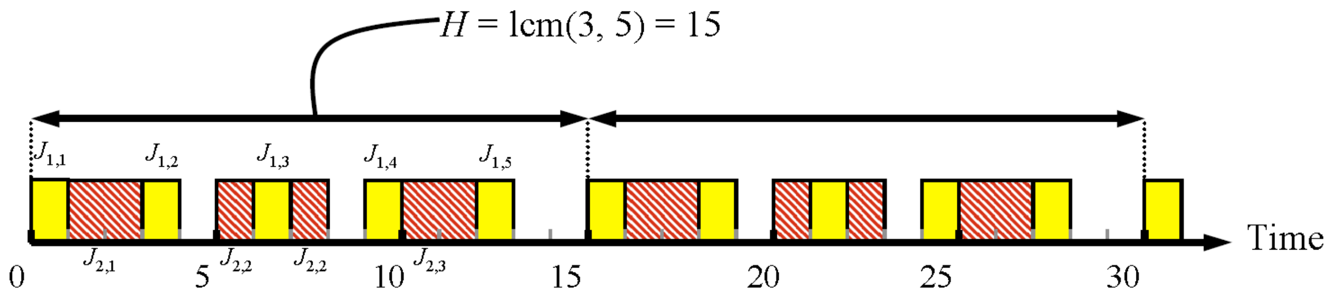
- **Periodic Task - a set of jobs that are executed repeatedly at regular time intervals**
- **Each periodic task T_i *is a sequence of jobs* $J_{i,1}, J_{i,2}, \ldots, J_{i,n}$**
  - The *phase* of a task T_i is the release time $r_{i,1}$ of the first job $J_{i,1}$ in the task. It is denoted by ϕ_i ("phi")
  - The *period* p_i of a task T_i is the length of time intervals between release times of two consecutive jobs
  - The execution time e_i of a task T_i is the maximum execution time of all jobs in the periodic task
  - The period and execution time of every periodic task in the system are known with reasonable accuracy at all times

Hyper-Periodic Tasks

- **The hyper-period of a set of periodic tasks is the least common multiple of their periods:**
  - $H = \text{LCM}(p_i)$ for $i = 1, 2, \ldots, n$
  - Time after which the pattern of job release/execution times starts to repeat, limiting analysis needed
- **Example:**

$T_1 : p_1 = 3, e_1 = 1$

$T_2 : p_2 = 5, e_2 = 2$



Utilization

- **The ratio $u_i = e_i/p_i$ is the *utilization* of task T_i**
  - The fraction of time a periodic task with period *pi and execution time* e_i keeps a processor busy
- **The *total utilization* of a system is the sum of the utilizations of all tasks in a system: $U = \Sigma u_i$**
- **We will usually assume the relative deadline for the jobs in a task is equal to the period of the task**
  - It can sometimes be shorter than the period, to allow slack time

⇒Many useful, real-world, systems fit this model; and it is easy to reason about such periodic tasks

Sporadic and Aperiodic

- **Many real-time systems are required to respond to external events**
- **The jobs resulting from such events are *sporadic or aperiodic* jobs**
  - A sporadic job has a hard deadlines
  - An aperiodic job has either a soft deadline or no deadline
- **The release time for sporadic or aperiodic jobs can be modeled as a random variable with some probability distribution, *A*(x)**
  - *A*(x) gives the probability that the release time of the job is not later than x
- **Alternatively, if discussing a stream of similar sporadic/aperiodic jobs, *A*(x) *can be viewed as the probability distribution of their* inter-release times**

[Note: sometimes the terms *arrival time* (or *inter-arrival time*) are used instead of release time, due to their common use in queuing theory]

Modelling Sporadic and Aperiodic Tasks

- **A set of jobs that execute at irregular time intervals comprise a sporadic or aperiodic task**
  - Each sporadic/aperiodic task is a stream of sporadic/aperiodic jobs
- **The inter-arrival times between consecutive jobs in such a task may vary widely according to probability distribution *A*(x) *and* can be arbitrarily small**
- **Similarly, the execution times of jobs are identically distributed random variables with some probability distribution *B*(x)**

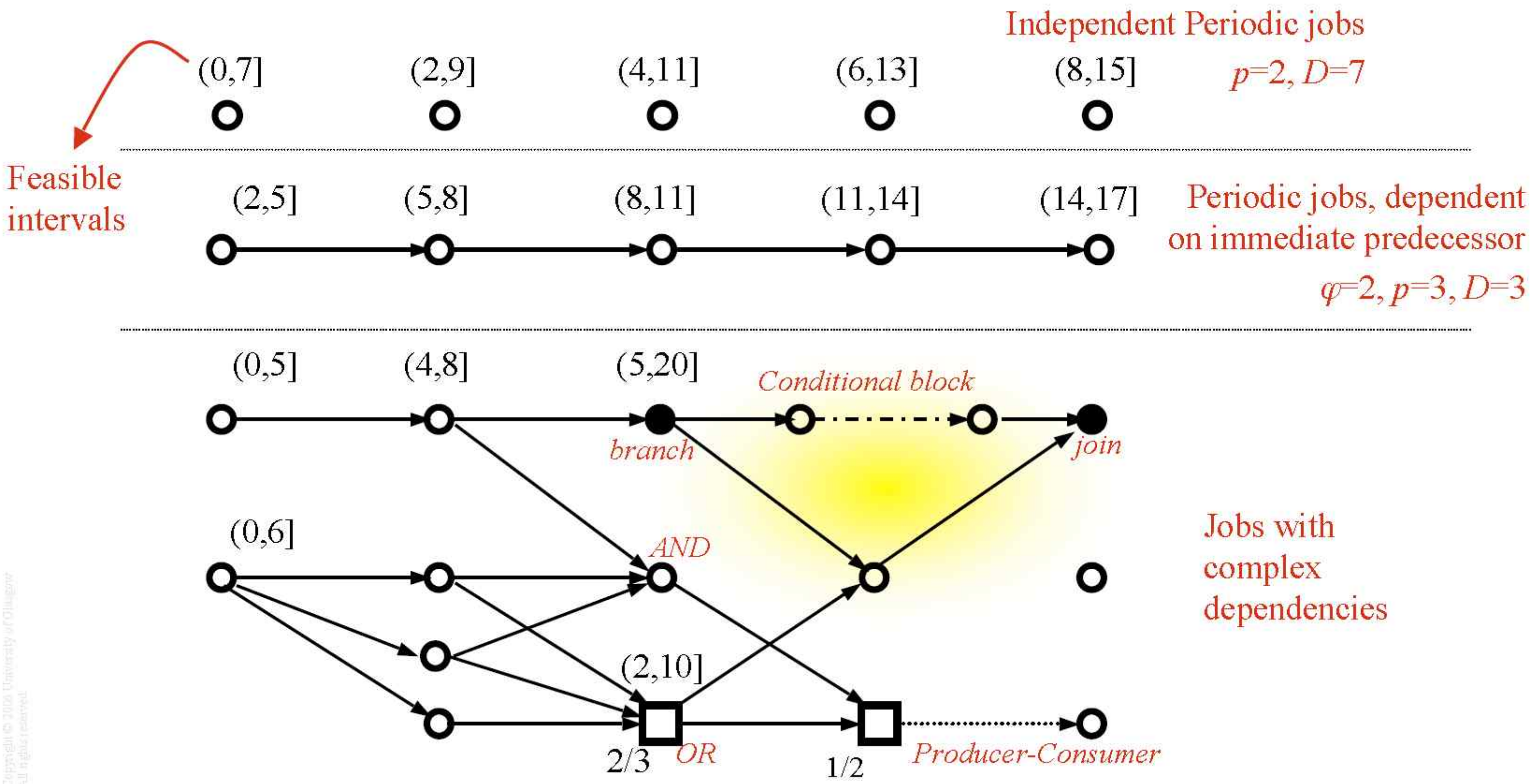
⇒Sporadic and aperiodic tasks occur in some real-time systems, and greatly complicate modelling and reasoning

Precedence Constraints and Dependencies

- **The jobs in a task, whether periodic, aperiodic or sporadic, may be constrained to execute in a particular order**
  - This is known as a *precedence constraint*
  - A job J_i is a predecessor of another job J_k *(and* J_k *a successor of* J_i*) if* J_k cannot begin execution until the execution of *Ji completes*
    - Denote this by saying $J_i < J_k$
  - J_i *is an immediate predecessor of* J_k *if* $J_i < J_k$ *and there is no other job* J_j such that $J_i < J_j < J_k$
  - J_i *and* J_k *are independent when neither* $J_i < J_k$ *nor* $J_k < J_i$
- **A job with a precedence constraint becomes ready for execution once when its release time has passed and when all predecessors have completed**

Task Graphs

- **Can represent the precedence constraints among jobs in a set *J* using a directed graph *G* = (*J*, <)**
  - Each node represents a job represented; a directed edge goes from J_i to J_k if J_i is an immediate predecessor of J_k



Task Graphs: Dependencies & Constraints

- **Normally a job must wait for the completion of all immediate predecessors; an *AND* constraint**
  - Unfilled circle in the task graph
- **An *OR* constraint indicates that a job may begin after its release time if only some of the immediate predecessors have completed**
  - Unfilled squares in the task graph
- **Represent conditional branches and joins by filled in circles**
- **Represent a pair of producer/consumer jobs with a dotted edge**

- **Use to visualise structure of real time systems**

Functional Parameters

- **Jobs may have priority, and in some cases may be interrupted by a higher priority job**
  - A job is *preemptable* if its execution can be interrupted in this manner
  - A job is *non-preemptable* if it must run to completion once started
- **Many preemptable jobs have periods during which they cannot be preempted; for example when accessing certain resources**
  - The ability to preempt a job (or not) impacts the scheduling algorithm
  - The *context* switch time is the time taken to switch between jobs
    - Forms an overhead that must be accounted for when scheduling jobs
- **Response to missing a deadline can vary**
  - Some jobs have optional parts, that can be omitted to save time (at the expense of a poorer quality result)
  - Usefulness of late results varies; some applications tolerate some delay, others do not

Scheduling

- **Jobs scheduled and allocated resources according to a chosen set of scheduling algorithms and resource access-control protocols**
  - Scheduler implements these algorithms
- **A scheduler specifically assigns jobs to processors**
- **A schedule is an assignment of all jobs in the system on the available processors.**
- **A *valid schedule* satisfies the following conditions:**
  - Every processor is assigned to at most one job at any time
  - Every job is assigned at most one processor at any time
  - No job is scheduled before its release time
  - The total amount of processor time assigned to every job is equal to its maximum or actual execution time
  - All the precedence and resource usage constraints are satisfied

Scheduling

- **A valid schedule is also a *feasible schedule* if every job meets its timing constraints.**
  - *Miss rate* is the percentage of jobs that are executed but completed too late
  - *Loss rate* is the percentage of jobs that are not executed at all

- **A hard real time scheduling algorithm is *optimal* if the algorithm always produces a feasible schedule if the given set of jobs has feasible schedules**

- **Many scheduling algorithms exist: main focus of this module is understanding real-time scheduling**

Summary

- **Outline of terminology and a reference model:**
  - Jobs and tasks
  - Processors and resources
  - Time and timing constraints
    - Hard real-time
    - Soft real-time
  - Periodic, aperiodic and sporadic tasks
  - Precedence constraints and dependencies
  - Scheduling