EE5903 RTS
Chapter 2
Task Scheduling Basics

Bharadwaj Veeravalli

elebv@nus.edu.sg

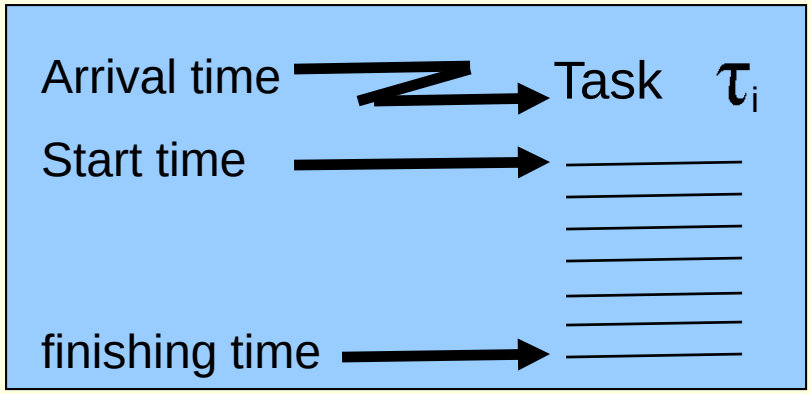
Acknowledgements: Parts of this material from Hard Real-Time Computing Systems, G.C, Buttazzo, Springer, 3rd Edition

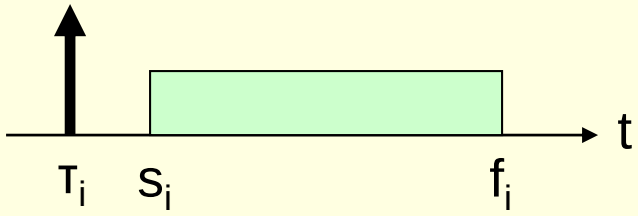
Outline – Task Scheduling

- Task specifications, definitions, examples of RTs
- RT task characteristics & task models,
- Timing & Precedence Constraints
- Resource and Resource Constraints
- Scheduling problem & Taxonomy
- Classification of RT Scheduling algorithms
- Classical scheduling policies – Some Real-time/Non-RT scheduling policies
- Scheduling a Bag of Tasks

Some useful definitions

- **Process (**or **task)**
  - is a sequence of instructions that in the absence of other activities is continuously executed by the processor until completion.

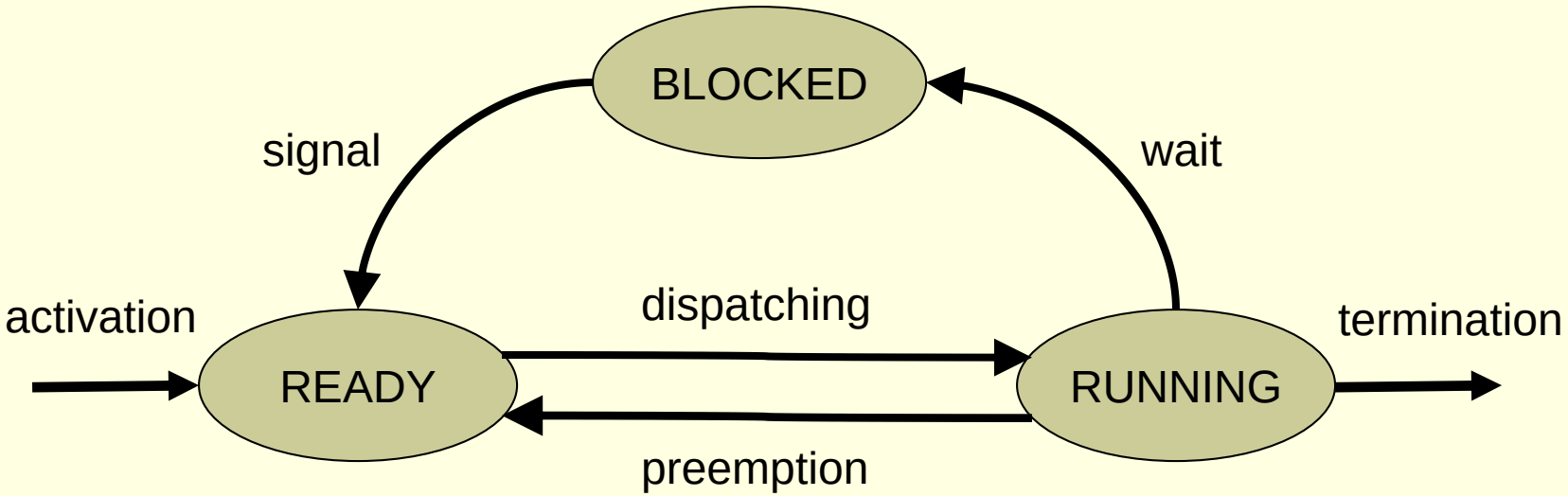




Task states

- A task is said to be:
  - **ACTIVE**: if it can be executed by the CPU;
  - **BLOCKED**: if it is waiting for an event;
- An **active** task can be:
  - **RUNNING**: if it is being executed by the CPU;
  - **READY**: if it is waiting for the CPU.

Task State Transitions

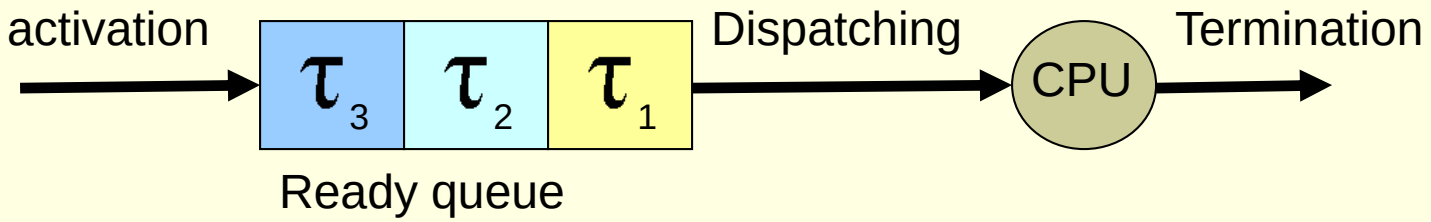


Generic task transition diagram

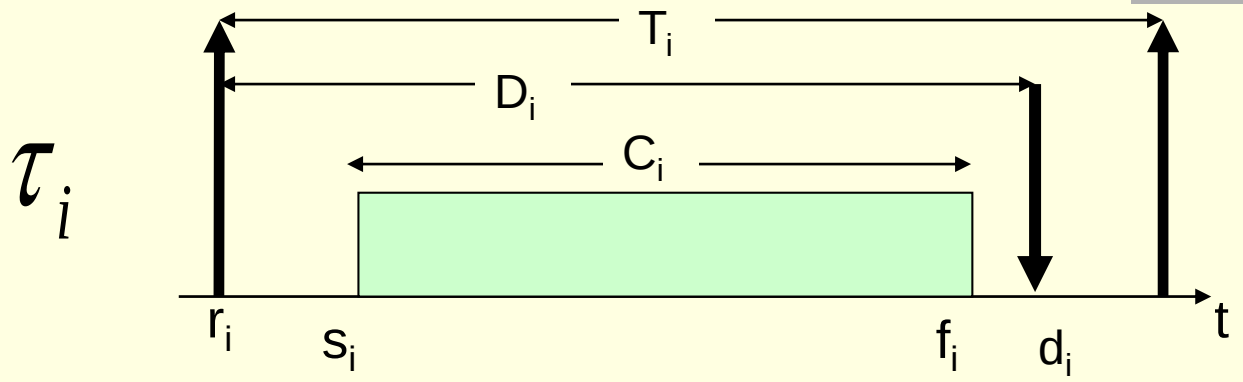
Ready Queue

- The **ready** tasks are kept in a waiting queue, called the **ready queue**;

- A strategy for choosing the ready task to be executed on the CPU is the **scheduling algorithm**.

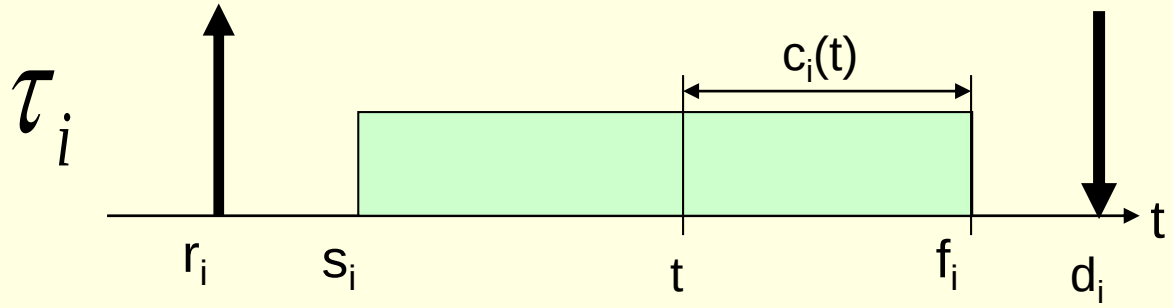


Real-Time tasks – Typical parameters



- T_i inter-arrival time (period) – applicable to periodic tasks
- **r_i** request time (arrival time **a_i**)
- **s_i** start time
- **C_i** worst-case execution time (wcet / avg-cet (acet))
- **d_i** absolute deadline
- **D_i** relative deadline (relative to the arrival time)
- **f_i** finishing time
- Response time = finish time – request time

Other parameters



- **Lateness**: $L_i = f_i - d_i$

Tardiness: $\max(0, L_i)$

(Exceeding time – Time a task stays active after its deadline)

Laxity: Maximum time a task can be delayed on its activation to complete within its deadline; Laxity = $(d_i - r_i - C_i)$

Laxity (or Slack) plays a crucial role in deciding schedulability/acceptance of a generated schedule for tasks, especially while handling aperiodic tasks.

Task Criticality

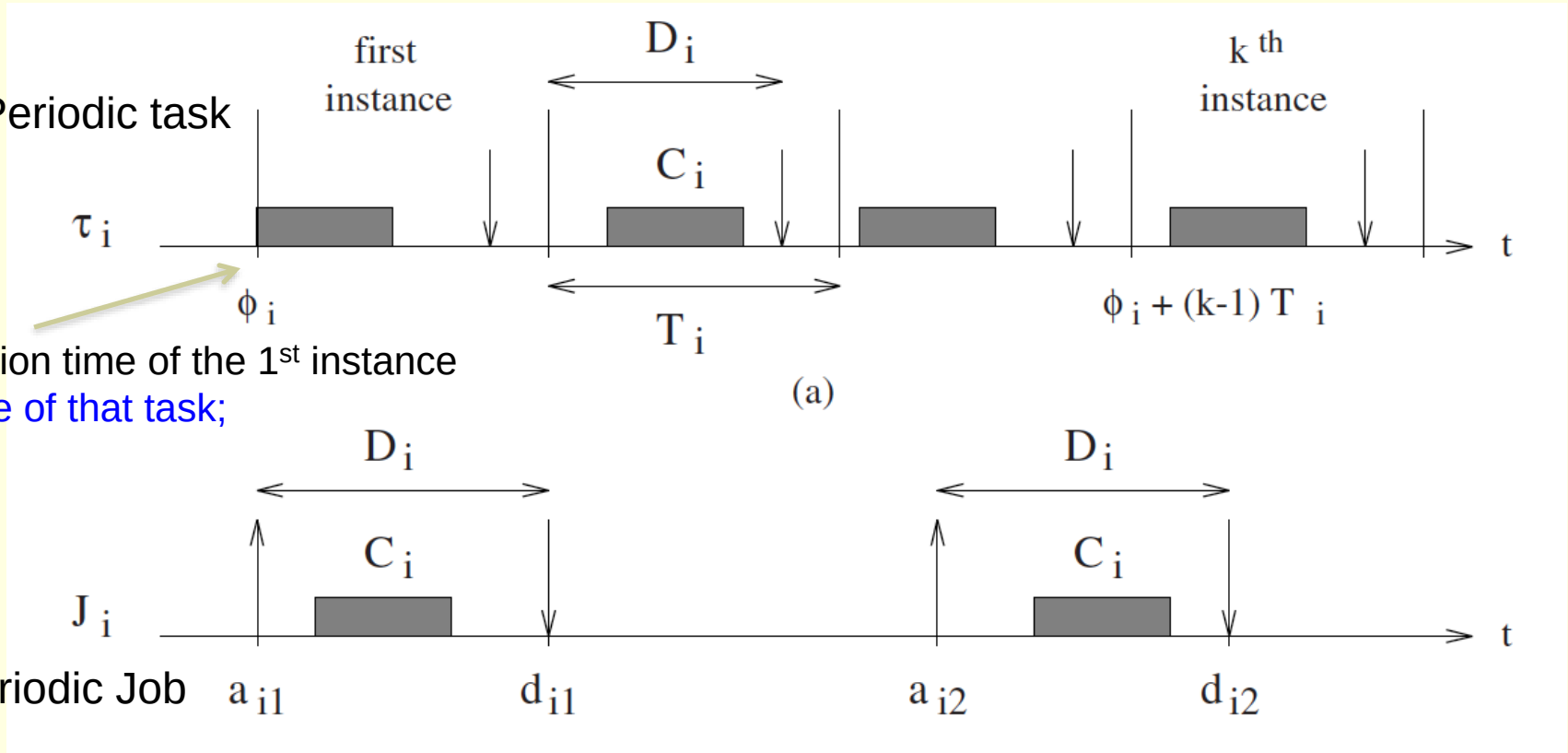
- **HARD real time tasks**
  - Missing a deadline is highly undesirable
  - May cause catastrophical effects on the system
- **FIRM real time tasks**
  - Missing a deadline is highly undesirable
  - Same as Hard RT, except safety is not an issue
- **SOFT tasks**
  - Occasional miss of a deadline is tolerable
  - Causes a performance degradation

An operating system able to handle hard RT tasks is called a **hard real-time** system.

Activation modes

- **Time driven**: **periodic** tasks
  - Task is automatically activated by the kernel at regular intervals.
- **Event driven**: **aperiodic/ sporadic** tasks
  - Task is activated upon the arrival of an event or through an explicit invocation of the activation primitive.
  - Sporadic – Consecutive tasks have *known minimum inter-arrival times*; often done for worst case calculations for aperiodic tasks;

Periodic & Aperiodic Tasks



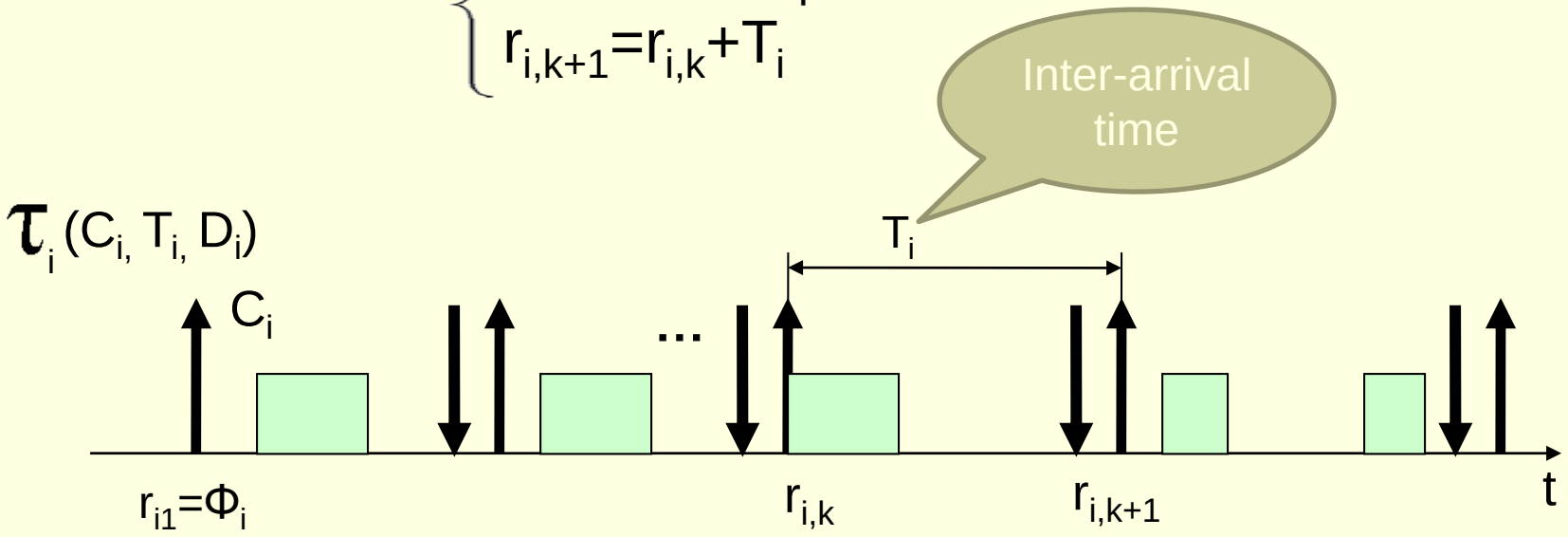
Activation time of the 1st instance - phase of that task;

Periodic tasks consist of an infinite sequence of identical activities, called instances or jobs , that are regularly activated at a constant rate.

Periodic task model

$$\begin{cases} r_{i1} = \Phi_i \\ r_{i,k+1} = r_{i,k} + T_i \end{cases}$$

Phase – Activation time of the first instance of a periodic task i



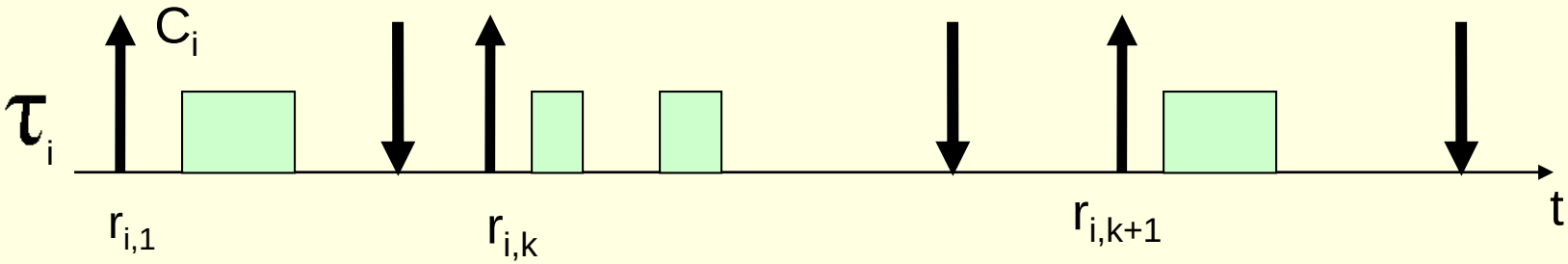
Activation time of the k-th instance of the periodic task: r_{ik}

$$r_{i,k} = \Phi_i + (k-1)T_i$$
$$d_{i,k} = r_{i,k} + D_i$$

[often $D_i = T_i$ but need not be true always]

Aperiodic task model

- **Aperiodic:** $r_{i,k+1} > r_{i,k}$
- **Sporadic:** $r_{i,k+1} \geq r_{i,k} + T_i$



Aperiodic tasks also consist of an infinite sequence of identical jobs (or instances); however, their activations are not regularly interleaved.

An aperiodic task where consecutive jobs are separated by a minimum inter-arrival time is called a sporadic task .

Types of Real-time constraints

- **Timing constraints**
  - Activation, completion, jitter.
- **Precedence constraints**
  - They impose an ordering in the execution
- **Resource constraints**
  - They enforce a *synchronization* in the access of mutually exclusive resources.

Timing constraints

- Can be explicit or implicit.
  - **Explicit constraints**
    - Usually included in the specification of the system activities, in the context of the problem.
  - Examples
    - CPU signals to open the valve **in** 10 seconds
    - Send the position information **within** 40 ms
    - Read the altimeter buffer **every** 200 ms

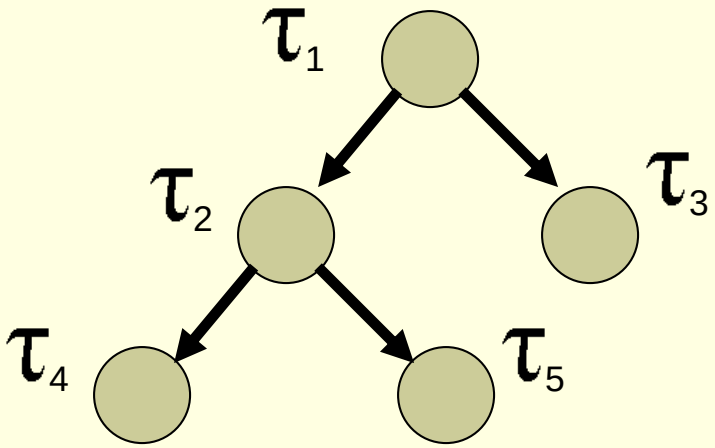
Timing constraints

- **Implicit constraints**
  - Do not appear in the system specification, but must be respected to meet the requirements

- Example
  - Avoid obstacles while driving at speed **v**; In this case, the speed has to be carefully calculated such that all obstacles are avoided

Precedence constraints

- Sometimes tasks must be executed with specific precedence relations, specified by a **Directed Acyclic Graph**:



predecessor

$\tau_1 < \tau_4$

Immediate predecessor

$\tau_1 \rightarrow \tau_2$

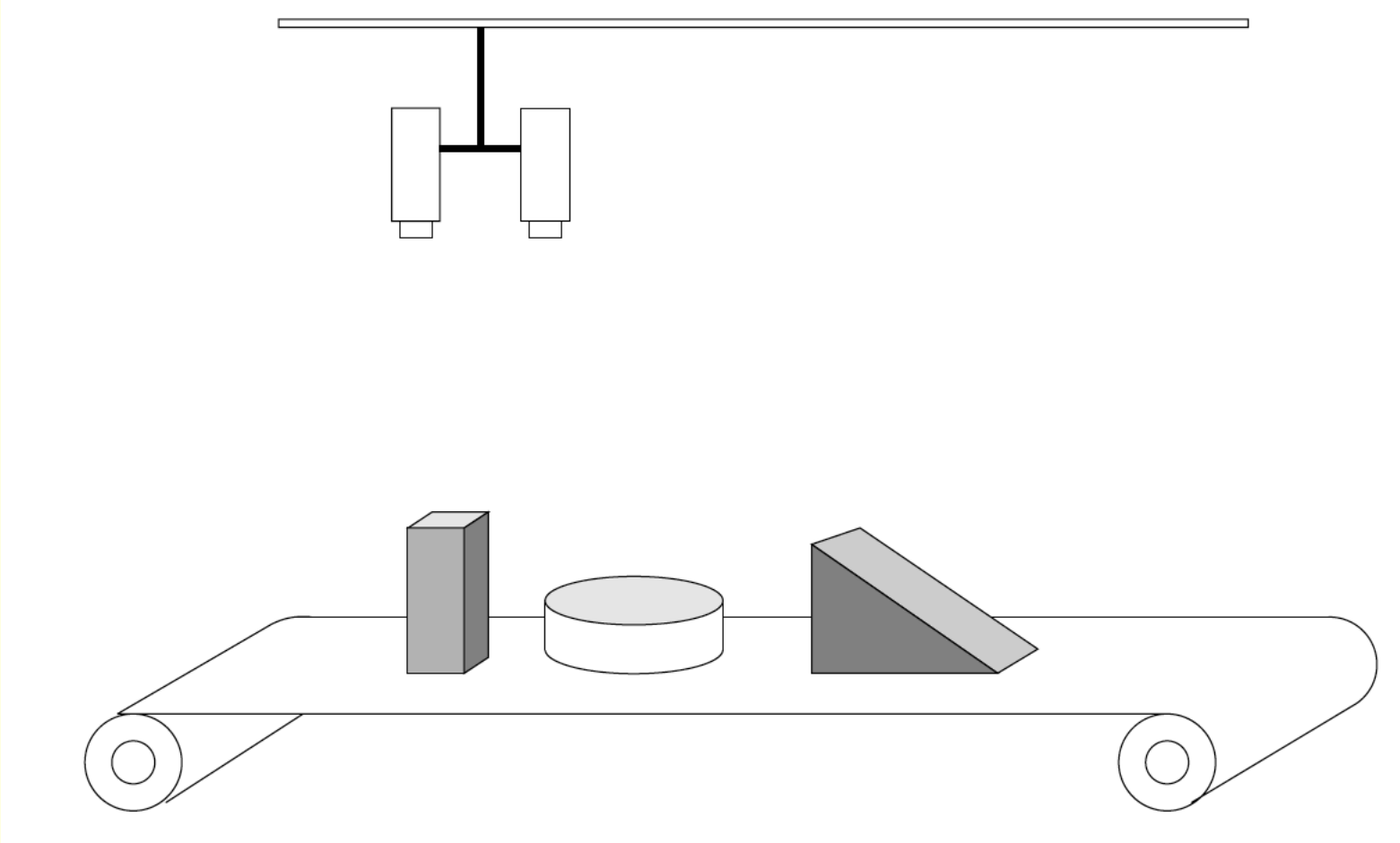
How does an application specific DAG gets generated?

- What is a DAG in a given context and how it is formed? – Organization of computational activities pertaining to an application

- Camera based automatic device inspection in an industrial environment – a RT challenge!

Number of objects moving on a conveyor belt must be recognized and classified using a stereo vision system, consisting of two cameras mounted in a suitable location.

How does an application specific DAG gets generated?



Stereo vision system – DAG generation

- Objective : To recognize the objects moving on the conveyor belt.

The recognition process is carried out by integrating the 2D features of the top view of the objects;

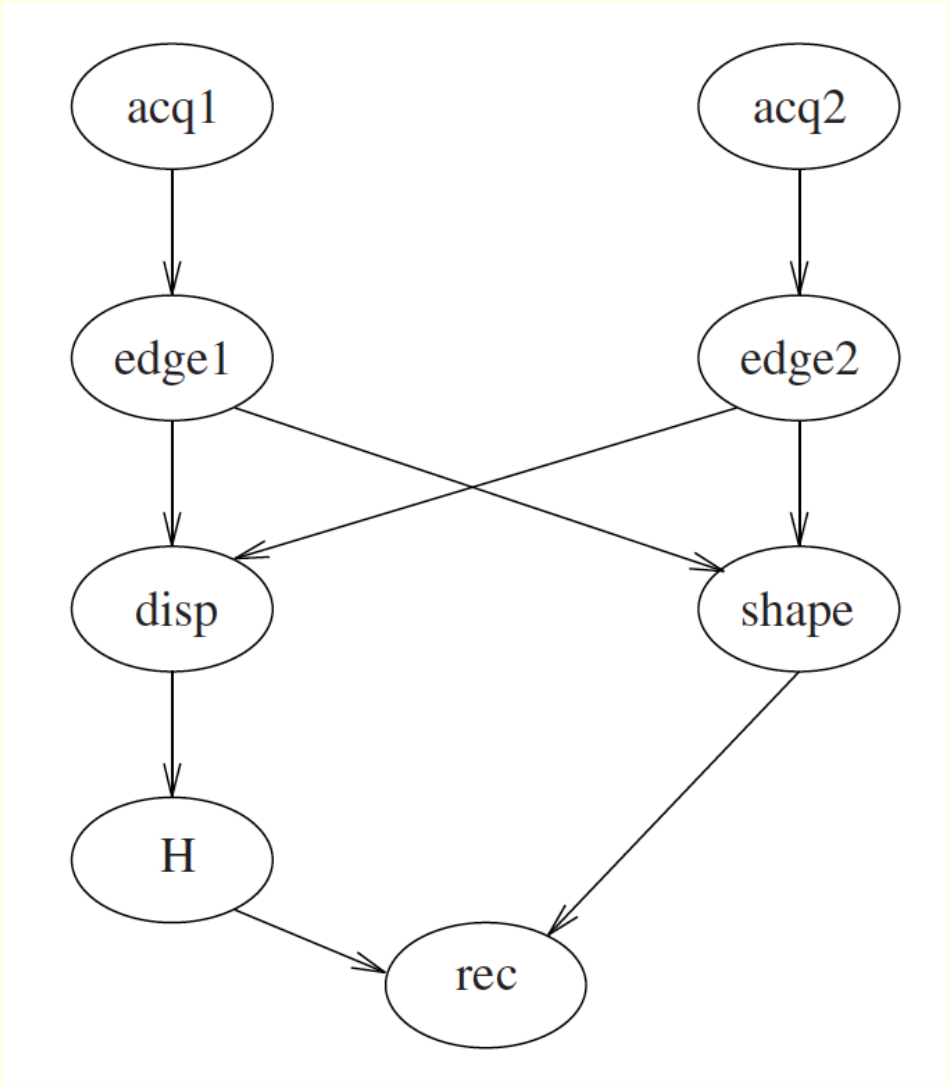
Height information of the objects - Extracted by the computing the pixel disparity on the two images.

As a consequence, the computational activities of the application can be organized by defining the following tasks:

How does an application specific DAG gets generated?

- Two tasks (one for each camera) dedicated to image acquisition, whose objective is to transfer the image from the camera to the processor memory (they are identified by *acq1* and *acq2*);
- Two tasks (one for each camera) dedicated to low-level image processing (typical operations performed at this level include digital filtering for noise reduction and edge detection; we identify these tasks as *edge1* and *edge2*);
- A task for extracting two-dimensional features from the object contours (it is referred as *shape*);
- A task for computing the pixel disparities from the two images (it is referred as *disp*);
- A task for determining the object height from the results achieved by the *disp* task (it is referred as *H*);
- A task performing the final recognition (this task integrates the geometrical features of the object contour with the height information and tries to match these data with those stored in the data base; it is referred as *rec*).

How does an application specific DAG gets generated?



Data acquisition

Edge detection

Disp & Shape detection

Height detection

Recognition

Resource & Resource Constraints

- **Resource** – S/w or sometimes H/w structure used by a process to *advance* its execution

- Examples: A resource can be a data structure, a set of variables, a main memory area, a file, a piece of program, or a set of registers of a peripheral device.

- Private or Shared

Resource & Resource constraints

- To maintain data consistency, many shared resources do not allow simultaneous accesses by competing tasks, but require their *mutual exclusion*. This means that a task cannot access a resource R if another task is inside R manipulating its data structures.

- In this case, R is called a mutually exclusive resource. A piece of code executed under mutual exclusion constraints is called a *critical section*.

More on these in rigor in Chapter 6. Stay tuned!

Resource & Resource constraints – data consistency & use of ME

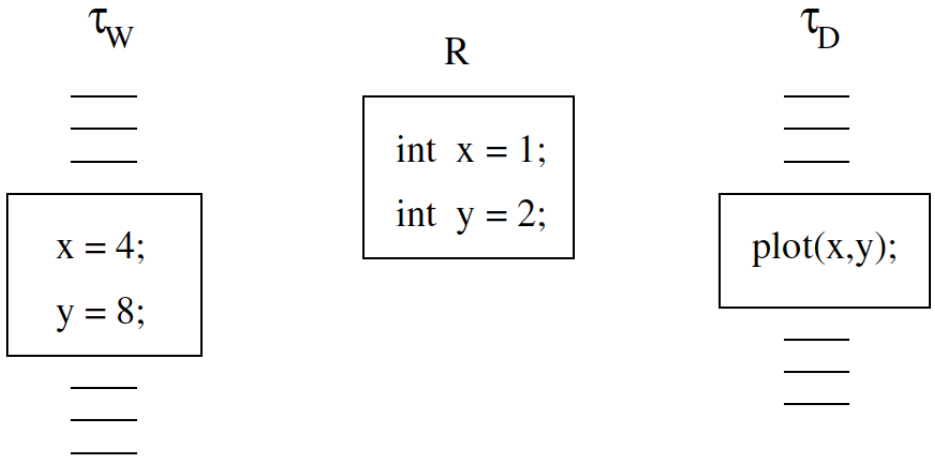
- Example: Object tracking system

Two tasks cooperate to track a moving object

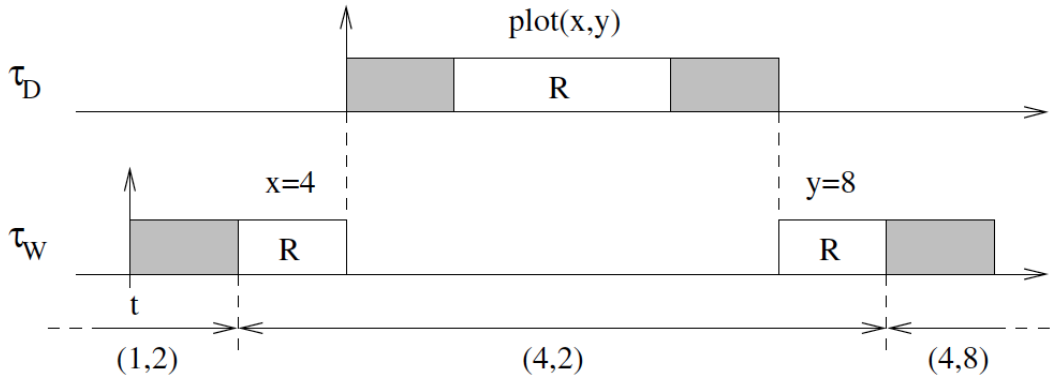
Task τ_W gets the object coordinates from a sensor and writes them into a *shared buffer R* , containing two variables (x, y);

Task τ_D reads the variables from the buffer and plots a point on the screen to display the object trajectory.

Resource constraints – Use of ME Principle - Consistency



Two tasks sharing a buffer with two variables.



Example of schedule creating data inconsistency.

R – Buffer –
Shared resource;

W is a low-priority task than D and can be Preempted by D;

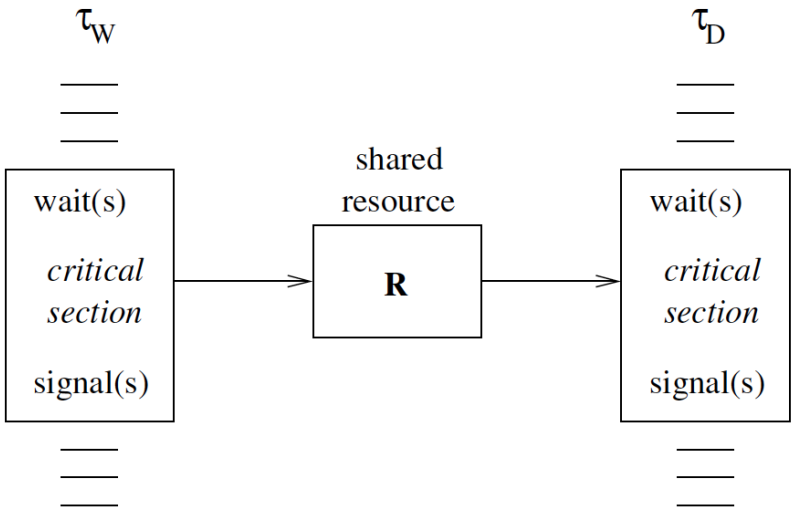
(1,2) – Current (x,y)

(4,8) – Correct (x,y)

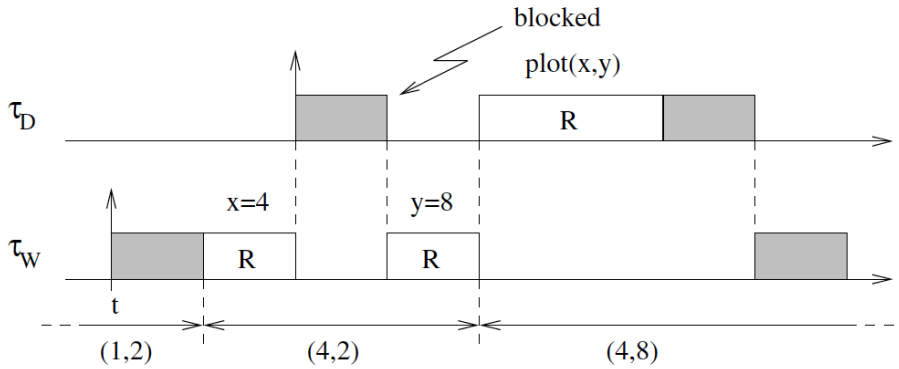
((4,2) – Result of no MEP

Mutual exclusion is desired

Mutual exclusion



s: Semaphore variable; wait(s) and Signal(s) are primitive/atomic functions

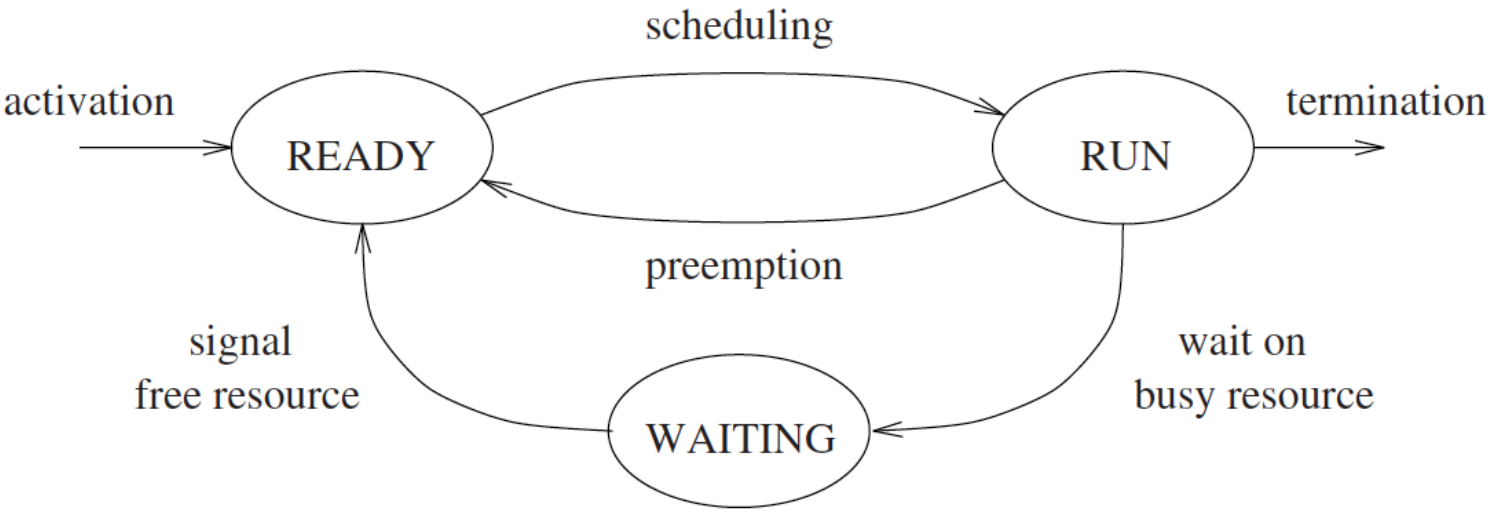


Example of schedule when the resource is protected by a semaphore.

Note that ME introduces Some extra delays!

No choice, but to live with it!

Mutual exclusion…(cont'd)



Waiting state caused by resource constraints

Task waiting for a shared resource implies BLOCKED on that resource;

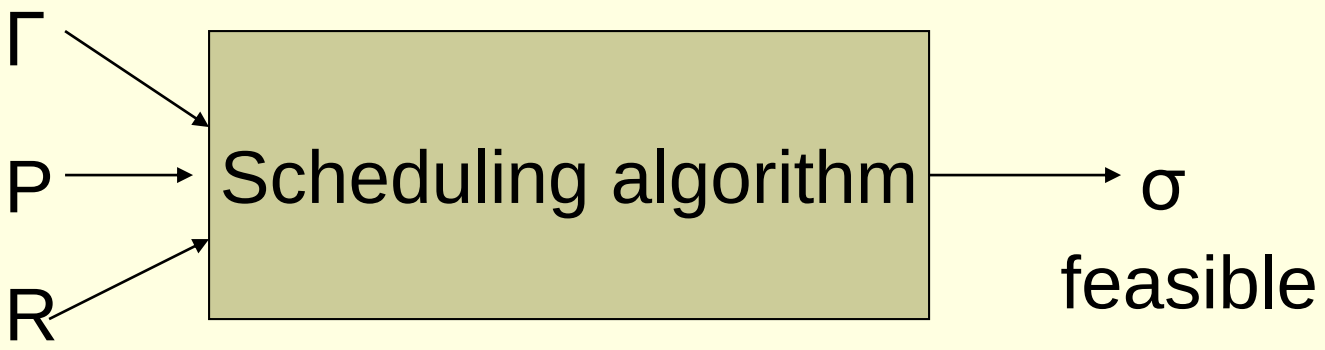
Running task -> Waiting -> Ready (scheduling algorithm decides)

General Definitions - Scheduling

- A **schedule** is a particular assignment of tasks to the processor in time.
- A set of tasks Γ (Gamma) is said to be **schedulable** if there exists a feasible schedule for it.
- A schedule σ is said to be **feasible** if all the tasks are able to complete within a set of constraints.

The general scheduling problem

- Given a set Γ of n tasks, a set **P** of m processors, and a set **R** of r resources, find an assignment of **P** and **R** to Γ which produces a feasible schedule.



Complexity

- In 1975, Garey and Johnson showed that the general scheduling problem is NP hard.
- However, polynomial time algorithms can be found under particular / application specific conditions – Greedy algorithms!
- Greedy algorithms – Exploit certain key characteristics of the problem / application to derive near/sub-optimal solutions;

  -- Important to derive how far you are away from an optimal solution – Quality of the solution generated by your greedy algorithm.

Simplified & *tough* assumptions

- Single CPU/core/thread / Multi-core
- No platform related delays / Delays exist
- Homogeneous task sets / Heterogeneous
- Fully preemptive tasks / Hybrid set
- Simultaneous activations / Aperiodic/sporadic
- No precedence constraints / Hybrid
- No resource constraints / Resource constraints present

- In case of DAGs – WCET / ACET are known (obtained from task profiling exercise;) / + time varying execution times

Scheduling algorithm taxonomy

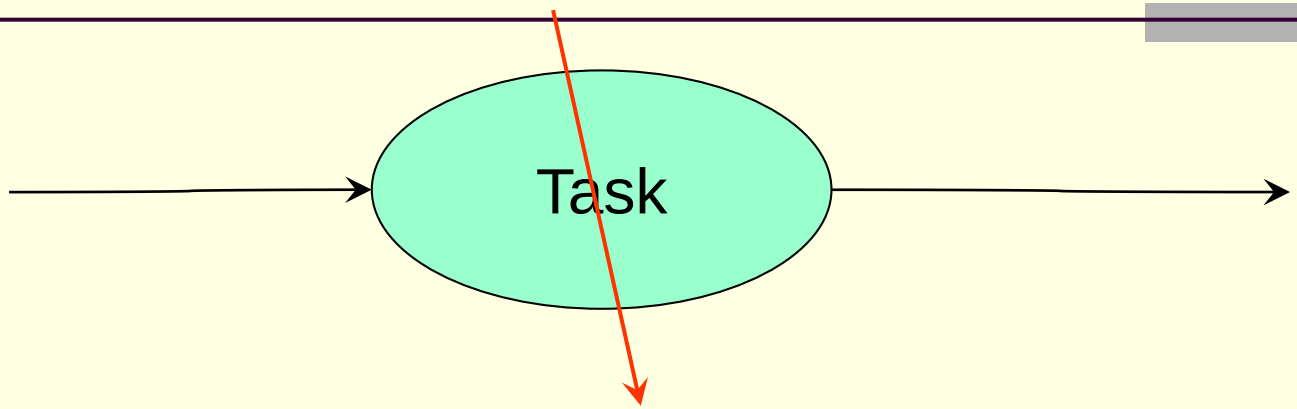
- Preemptive vs. Non Preemptive
- Static vs. dynamic
- On line vs. Off line
- Best Effort vs. Optimal

Preemptive, Non-preemptive & Deferred scheduling

- A scheduling algorithm is said to be:
  - **preemptive**: if the running task can be temporarily suspended in the ready queue to execute a more important task.
  - **non preemptive**: if the running task cannot be suspended until completion.
  - **deferred preemptive**: the running task is allowed to run until a bounded time

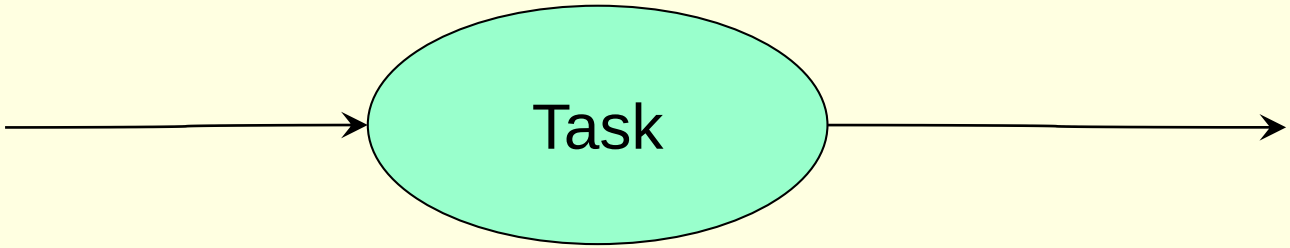
Preemptions – Often guided by priority levels of the tasks;

Preemptive Systems



- Interrupt a task when a higher priority task wants to execute
- For a large class of scheduling problems, it is shown that PSs are easy to manage in providing timing guarantees (*although not proven!*)
- Higher overhead of switching and memory
- Cache pollution; Can be resolved by constraining the amount of cache the prefetched data can occupy or via software techniques;

Non-Preemptive Systems



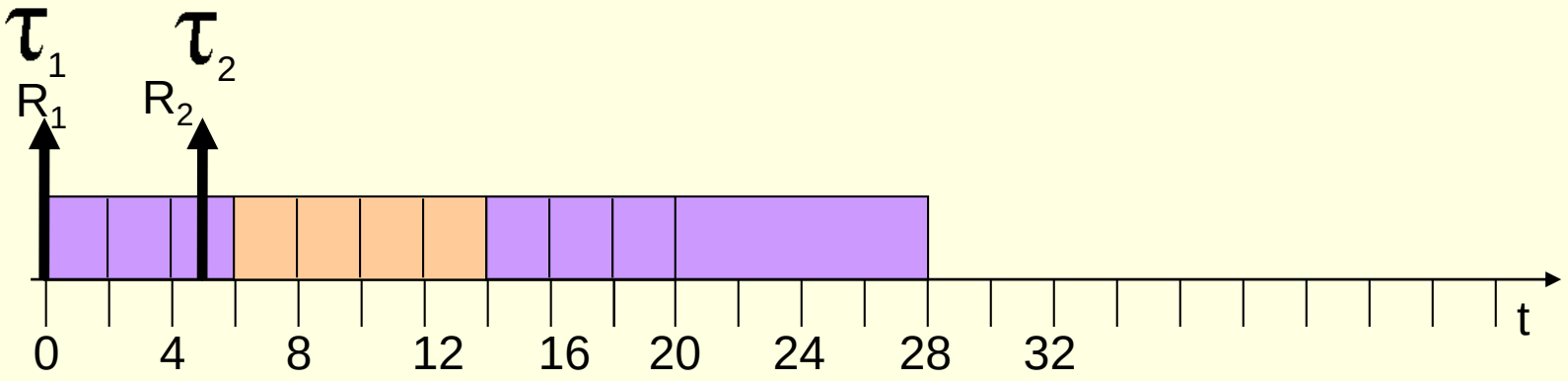
- State-space needed is smaller
- Lower implementation cost
- Less overhead at run-time
- Less cache pollution, smaller memory needed

Deferred Preemptive

- When a high priority task arrives, the low priority task is allowed to continue for a bounded time (not necessary towards completion)

- Essentially each task can be considered to be composed of small non-preemptible tasks – each small task with a bounded time

Deferred Preemptive



Task 1 – current task
Task 2, a high priority task arrives;

Static vs. Dynamic

- **Static**
  - Scheduling decisions are taken based on fixed parameters (e.g. computation time), statically assigned to tasks before activation.
- **Dynamic**
  - Scheduling decisions are taken based on parameters that can change with time (e.g. arrival time, exe time, etc).

Off-line vs. On-line

- **Off-line**
  - All scheduling decisions are taken before task activation: the schedule is stored in a table.
- **On-line**
  - Scheduling decisions are taken at run time on the set of active tasks.

Best-Effort vs. Optimal

- **Best-Effort**
  - Do their best to find a feasible schedule, if there exists one, but they do not guarantee if a best schedule would be generated.

- **Optimal**
  - Always attempts to determine an optimal schedule, if there exists one;

Classification of RT Scheduling Algorithms

- Classification schemes are based on the following three categories:

1) Based on how scheduling points are defined

1) Based on the type of task acceptance tests

1) Based on the target platform used

To be continued...