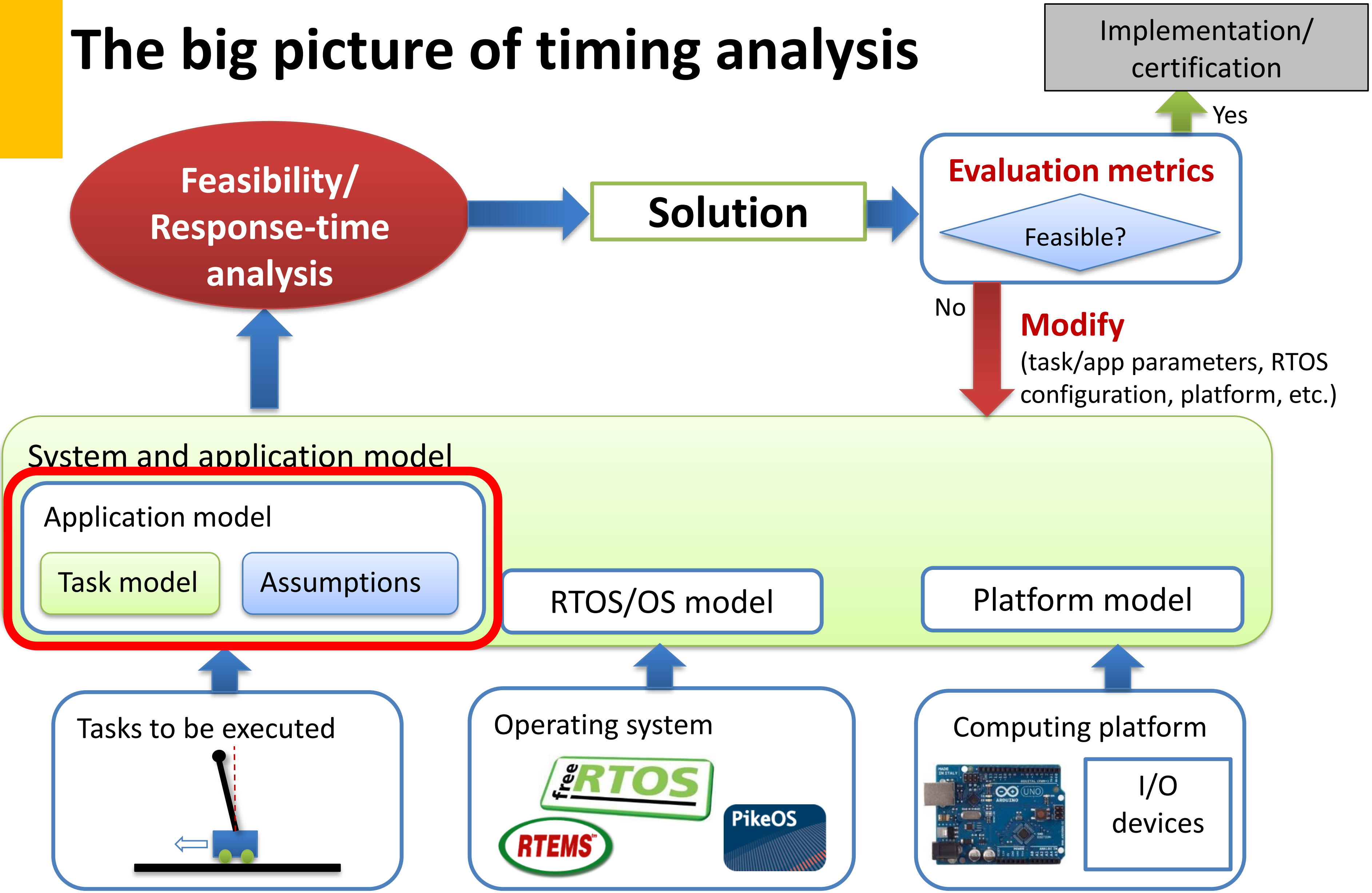
2IMN20 - Real-Time Systems

Modeling real-time systems

Geoffrey Nelissen

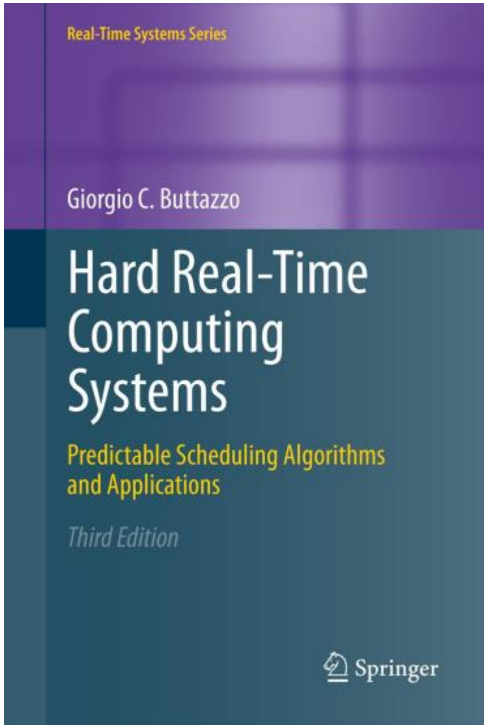
2023-2024

The big picture of timing analysis



Reading material

Buttazzo’s book, chapter 1 and 2



Disclaimer:

Most slides were provided by Dr. Mitra Nasri

Some slides have been taken from Giorgio Buttazzo’s website

Recap: Task

- **Task**
  - A **sequence of actions** that must be carried out to, for example, implement a functionality or respond to an event
  - A task may use multiple inputs and produce multiple outputs
  - It may be activated by
    - **events** (e.g., user command or an interrupt) or
    - **time** (e.g., by timer interrupts)

```
While(true)
{
    mq_receive (mq, (char*)&temp, sizeof(temp));
    if (temp > 42)
        send(-1);
    else
    {
        int * array = read10Data( );
        int max = -1;
        for (int i=0; i < 10; i++)
            if (max < 0 || array[i] > max)
                max = array[i];
        send(max);
    }
    sleep (100, ms);
}
```

Question: how is this task activated (on **event** or **time**)?

Question: What is the activation model of this task (periodic, sporadic or aperiodic)?

Workload characterization

- A task workload is characterized by three main aspects:
  - **Execution time**
  - **Activation/Arrival model**
  - **Computation model**

Workload characterization

- A task workload is characterized by three main aspects:
  - ~~**Execution time**~~
  - **Activation/Arrival model**
  - **Computation model**

Arrival model of a task

- A task releases a **sequence of jobs** to be executed. Job may be released
- **Periodically**

  with or without **release jitter**

  with or without **release offset** (first job released later than 0)
- **Sporadically**
- **Aperiodically**

Time-triggered activation

Event-triggered activation

- More complex arrival patterns may be modelled with **arrival curves**

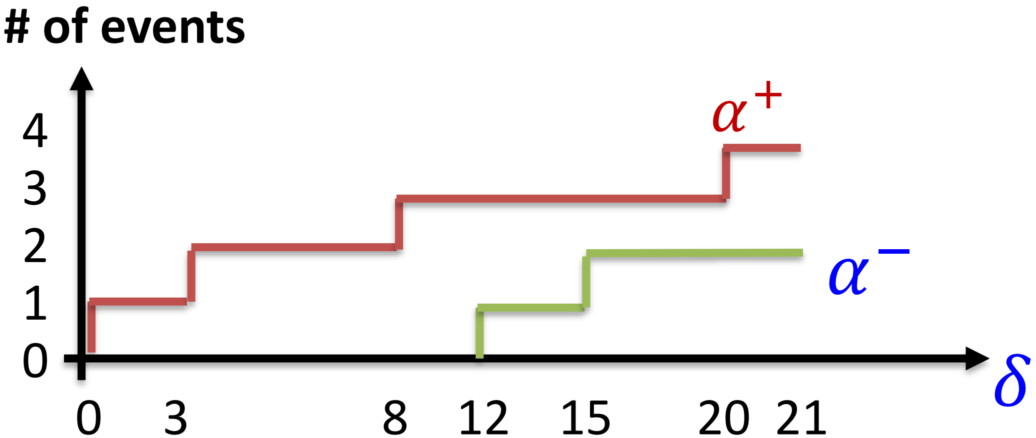
Note: many other arrival models exist. For example, **elastic tasks, variable-rate tasks, digraph tasks, …**

Modelling complex arrival patterns with arrival curves

- An arrival curve represents the **lower bound** and **upper bound** on the **number of events in any time interval**.

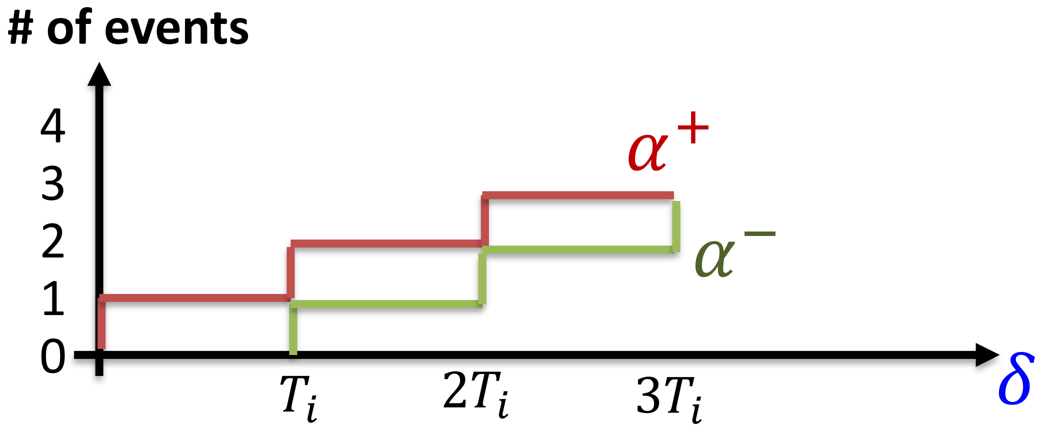
α^+ = maximum number of events in any interval of duration δ

α^- = minimum number of events in any interval of duration δ



Arrival curves

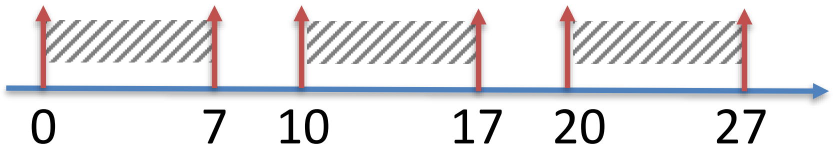
Question: How does the arrival curve of a periodic task with period T_i look like?



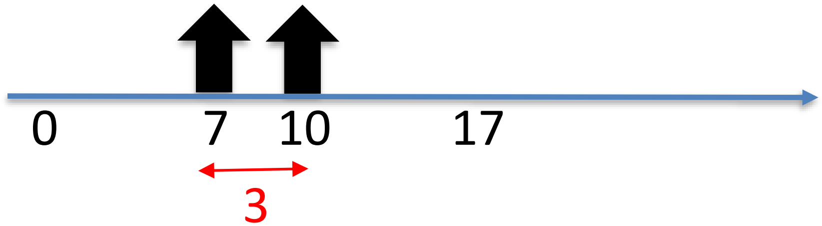
Question: Why $\alpha^- \neq \alpha^+$ even for a periodic task with no offset or release jitter?

Arrival curves

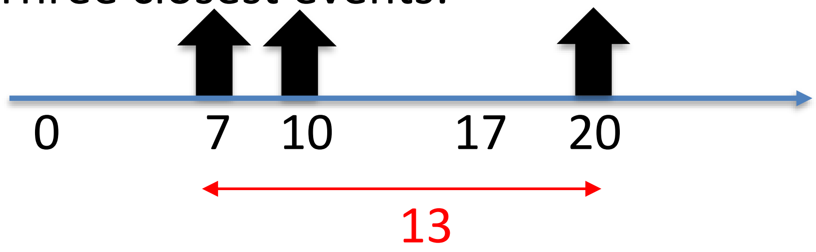
Question: How does the α^+ arrival curve of a periodic task with release jitter look like?
Period = 10 and release jitter σ = 7

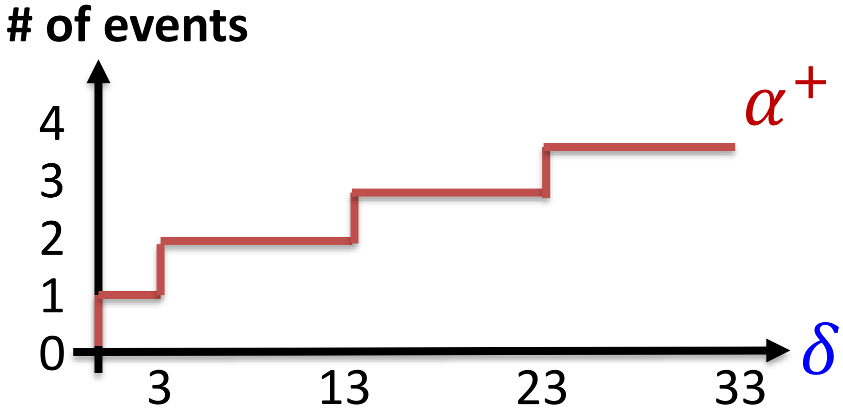


Two closest events:



Three closest events:

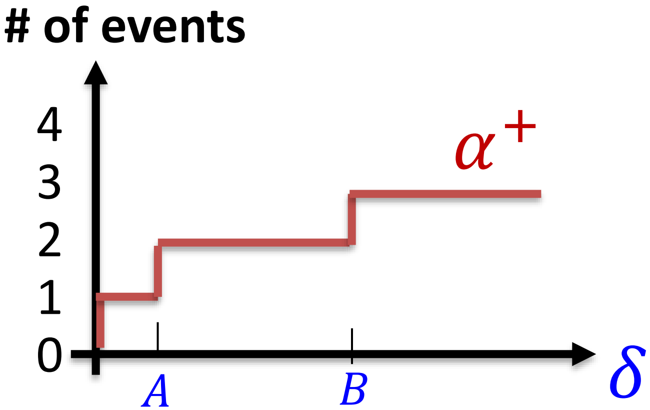




At home: Derive the α^- arrival curve

Arrival curves

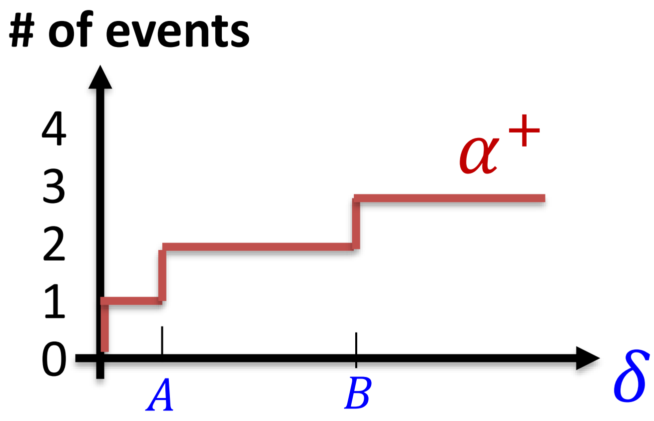
Question: How does the α^+ arrival curve of a periodic task with period T and release jitter σ look like?



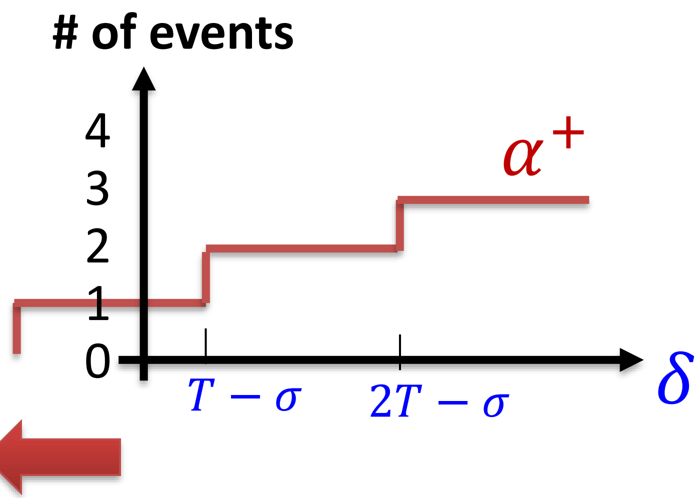
Determine A and B

Arrival curves

Question: How does the α^+ arrival curve of a periodic task with period T and release jitter σ look like?

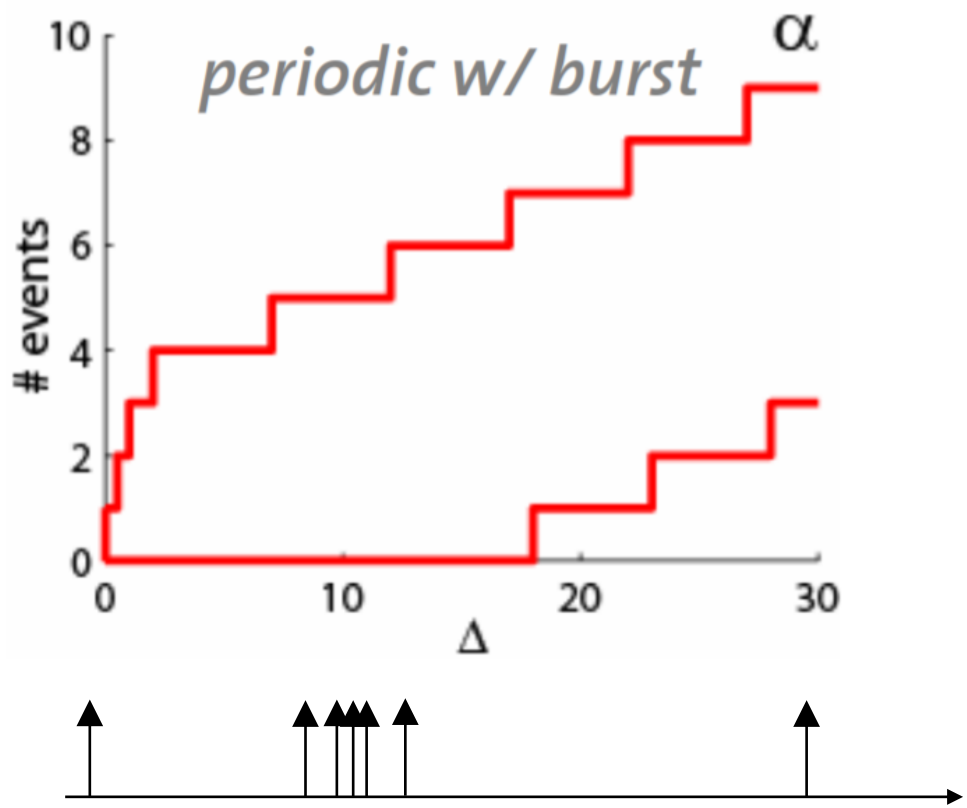


Determine A and B



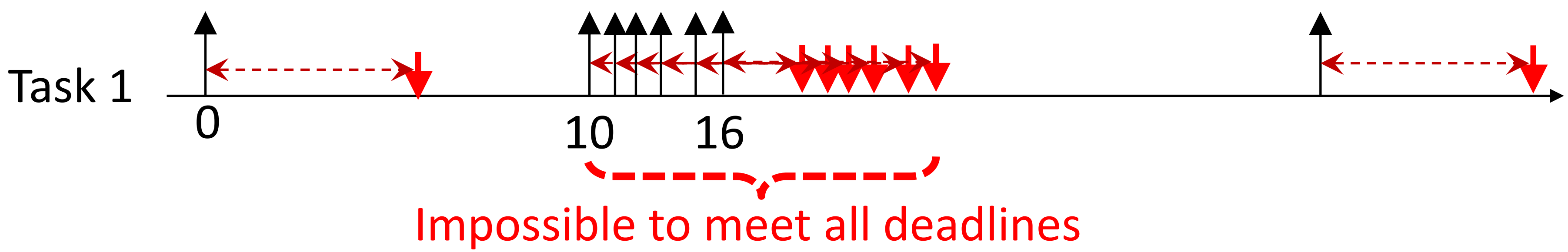
Shift α^+ of a purely periodic task by σ to the left to get α^+ for a task with release jitter σ

Other arrival curves



Analyzing aperiodic tasks is challenging

- Since it is hard to predict the future workload, it is hard to analyze the system to find the worst-case response time of the tasks
- With unbounded inter-arrival times, it is impossible to guarantee the timeliness of a resource-constrained system



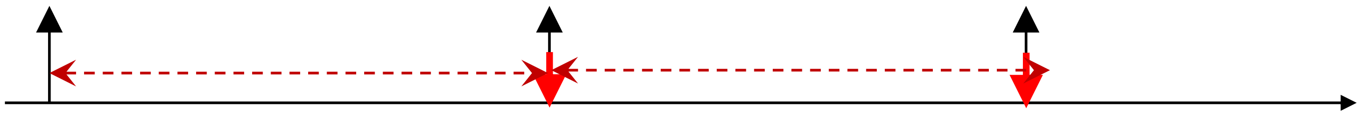
Characterization of the deadline

- Type of deadlines
  - **Hard**
  - **Soft**
  - **Firm**

- Relation between deadline and period
  - **Implicit deadline**
    - Deadline is **equal** to the period/minimum inter-arrival time
  - **Constrained deadline**
    - Deadline is **smaller than** or equal to the period/minimum inter-arrival time
  - **Arbitrary deadline**
    - Deadline may be **smaller than, equal to, or larger than** the period/minimum inter-arrival time

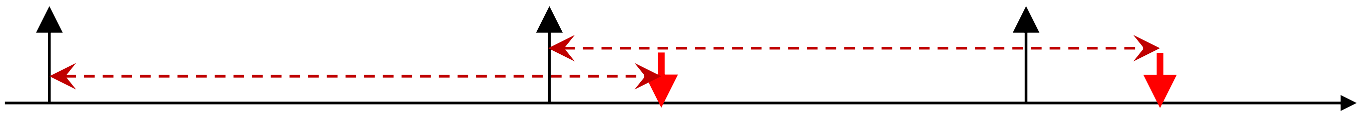
Relation between deadline and period

- **Implicit deadline:** deadline is equal to period



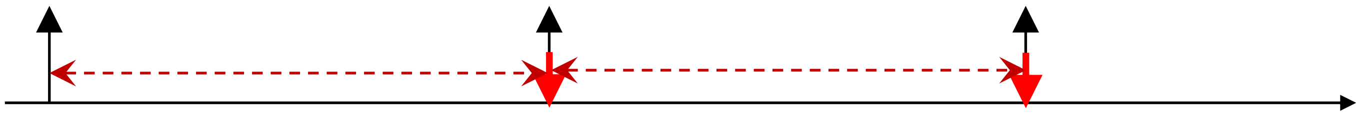
- **Constrained deadline**: deadline is smaller than or equal to period

- **Arbitrary deadline**: deadline can be smaller, equal, or larger than period



Relation between deadline and period

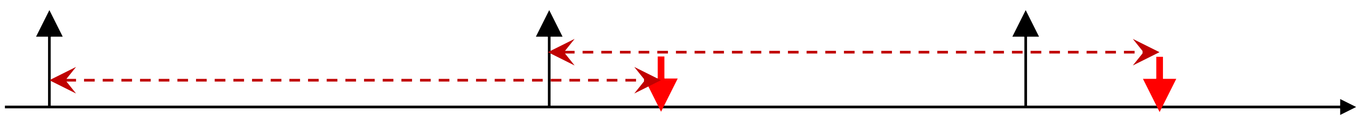
- **Implicit deadline:** deadline is equal to period



Harder to analyze

because at any time, **more than one instance of the task might be ready** for execution

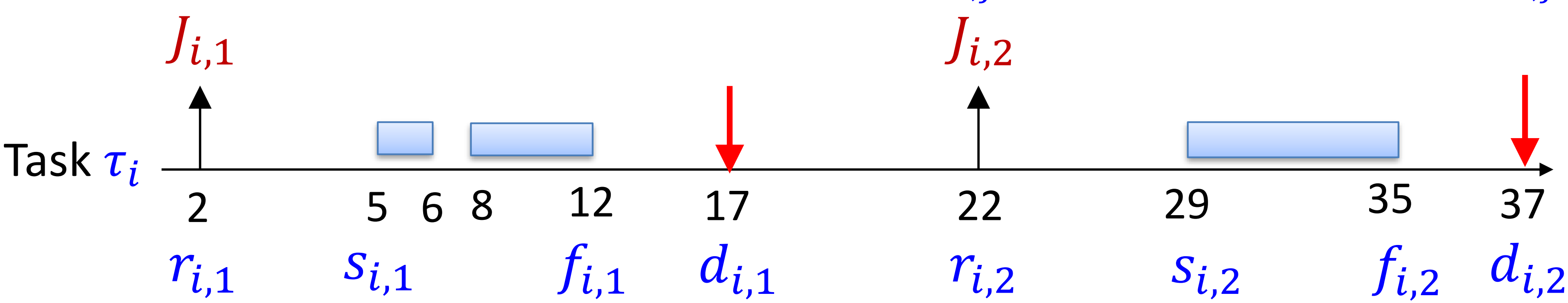
- **Arbitrary deadline**: deadline can be smaller, equal, or larger than period



Notations

τ_i : the i^{th} task in the task set
C_i : the WCET of task τ_i
T_i : the period of τ_i
σ_i : the release jitter of τ_i
Φ_i : the release offset (also called phase) of τ_i
D_i : the relative deadline of τ_i
R_i : the worst-case response time of τ_i

$J_{i,j}$: the j^{th} job of task τ_i
$a_{i,j}$: arrival time of the job $J_{i,j}$
$r_{i,j}$: release time of the job $J_{i,j}$
$c_{i,j}$: execution time of the job $J_{i,j}$
$d_{i,j}$: absolute deadline of the job $J_{i,j}$
$s_{i,j}$: start time of the job $J_{i,j}$
$f_{i,j}$: finish time of the job $J_{i,j}$
$R_{i,j}$: response-time of the job $J_{i,j}$



Question: What is the largest *observed* response time in this schedule?

13

Question: what is the equation to calculate $r_{i,j}$ for a given job $J_{i,j}$ of a periodic task without release jitter?

$$r_{i,j} = \Phi_i + (j-1) \cdot T_i$$

Question: What is the *observed* C_i in this schedule?

6

Notations and definitions

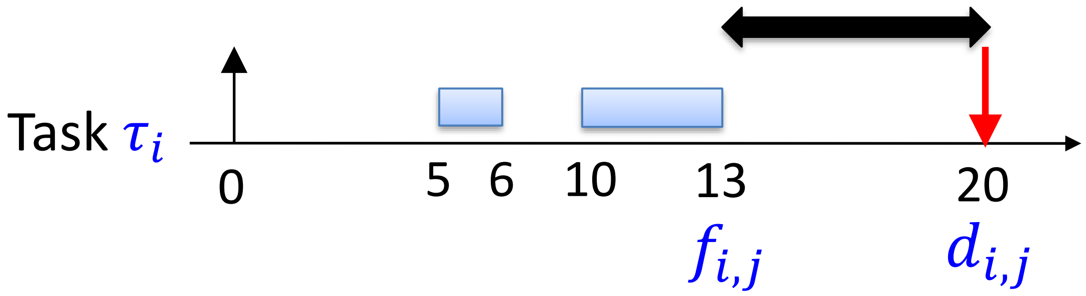
A schedule is said to be **feasible** for a real-time task τ_i if all jobs of this task **complete before their deadline**, that is, if

$$\forall j, \qquad f_{i,j} \leq d_{i,j}$$

or equivalently,

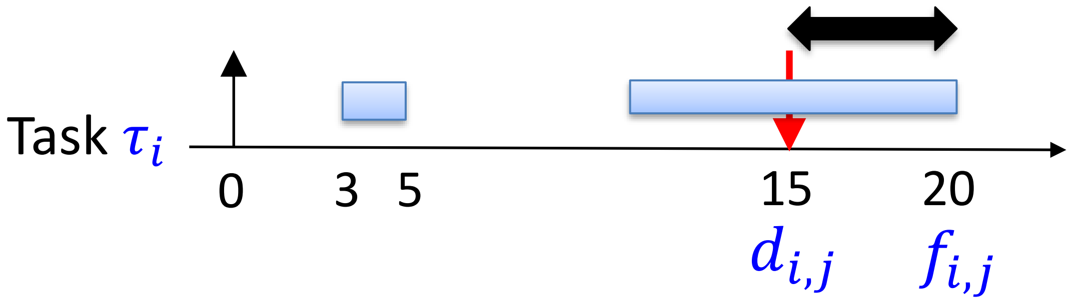
$$\forall j, \qquad R_{i,j} \leq D_i$$

Slack and lateness



Slack:

$$d_{i,j} - f_{i,j}$$



Lateness:

$$L_{i,j} = f_{i,j} - d_{i,j}$$

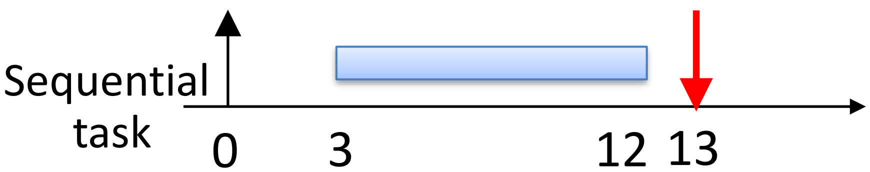
Tardiness: $\max\left(0, f_{i,j} - d_{i,j}\right)$

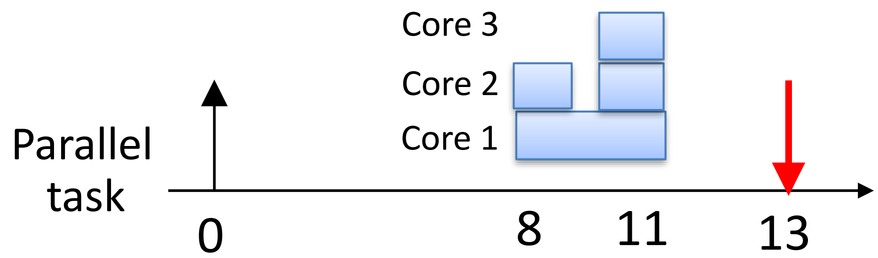
Workload characterization

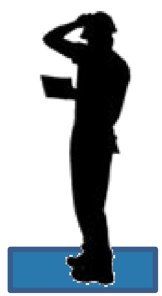
- A task workload is characterized by three main aspects:
  - ~~Execution time~~
  - ~~Activation/Arrival model~~
  - **Computation model**

Modeling computation

Sequential v.s. **Parallel**







Sequential task

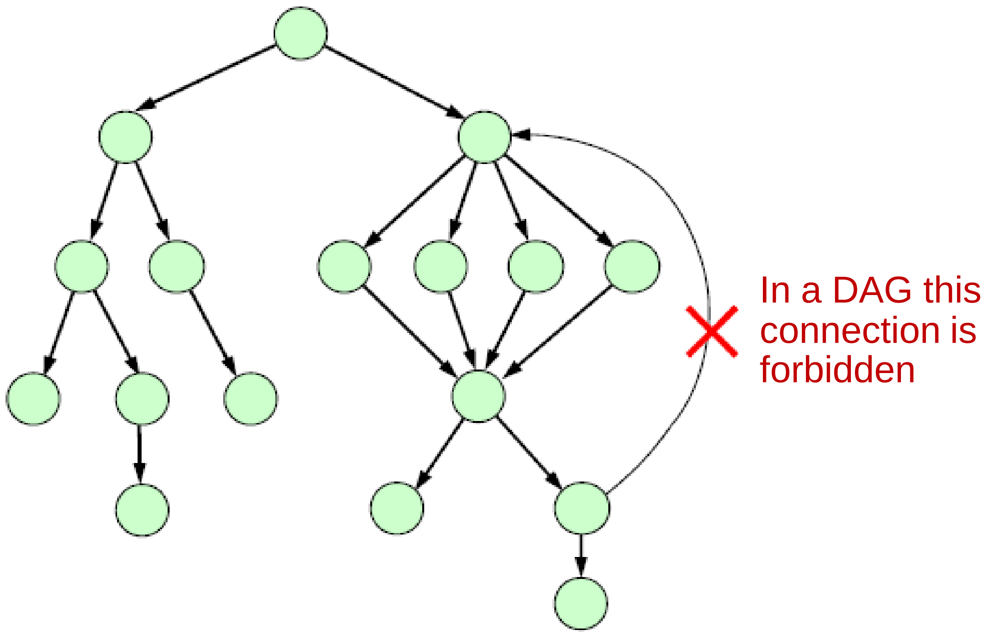
Parallel task

Parallel programming

- Existing parallel programming models/languages
  - OpenMP
  - MPI
  - Java/C++/Python/...
  - IBM's X10
  - Intel TBB
  - Sun's Fortress
  - Cray's Chapel
  - Cilk (Cilk++)
  - Codeplay's Sieve C++
  - Rapidmind Development Platform

Modeling parallel tasks

- Representing a parallel code requires more complex structures like a graph (usually a **directed-acyclic graph**, a.k.a. **DAG**):

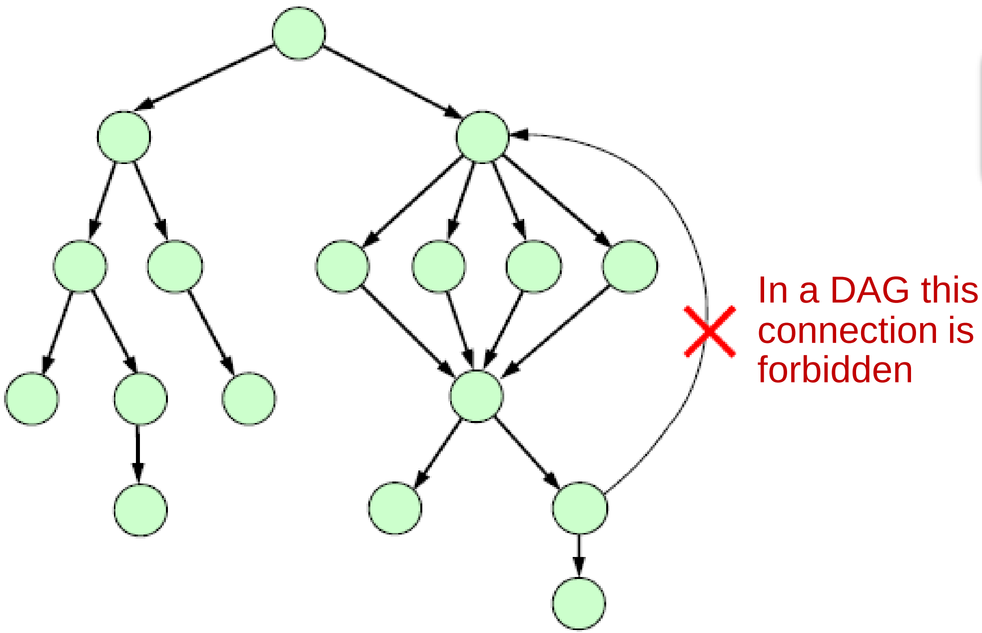


Question: What is the maximum parallelism of that task?

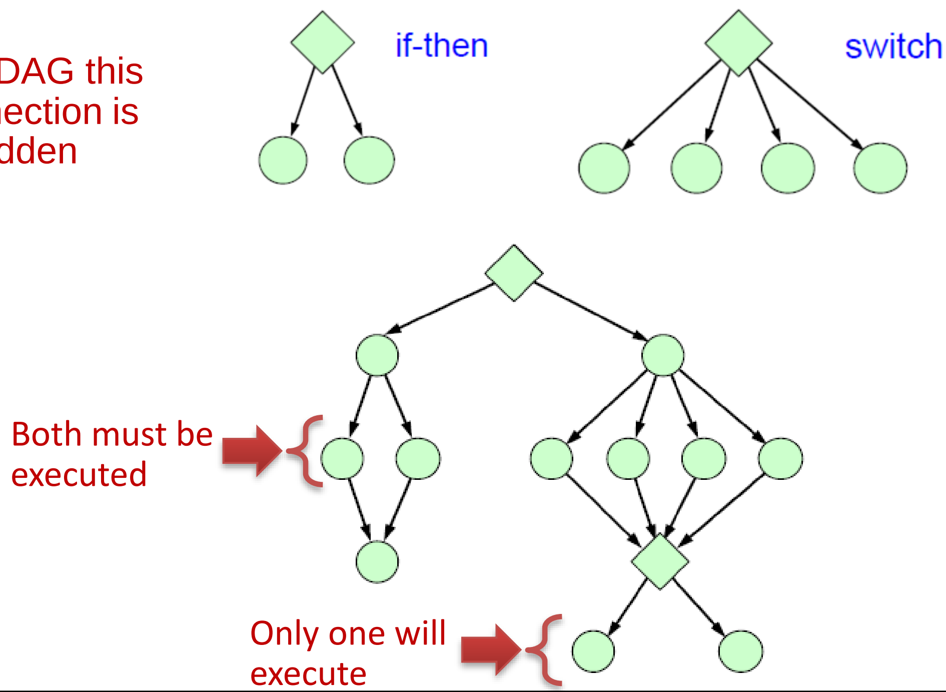
7

Modeling parallel tasks

- Representing a parallel code requires more complex structures like a graph (usually a **directed-acyclic graph**, a.k.a. **DAG**):



DAGs can be **conditioned** (e.g., by an if-then-else block)



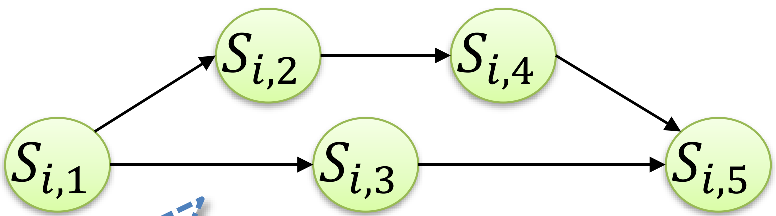
OR nodes represent conditional statements

Notations

Worst-case execution time of segment $S_{i,2}$

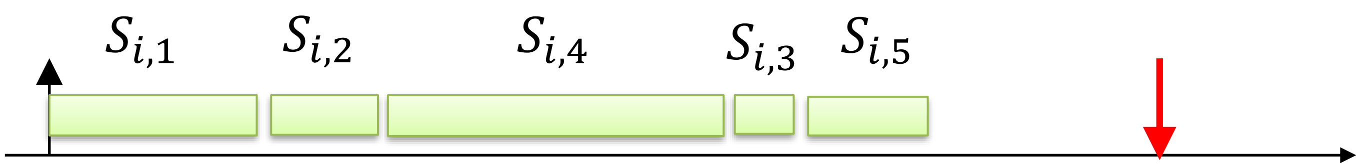
Task parameters:

$$\tau_i = \left(\langle C_{i,1}, C_{i,2}, \ldots, C_{i,m_i}\rangle, D_i, T_i\right)$$



Precedence constraint between task's segments
(a.k.a. **intra-task precedence constraint**)

Single core schedule

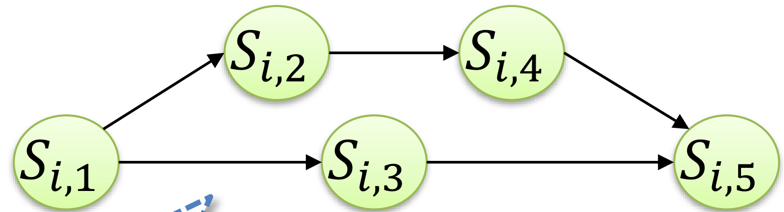


Notations

Worst-case execution time of segment $S_{i,2}$

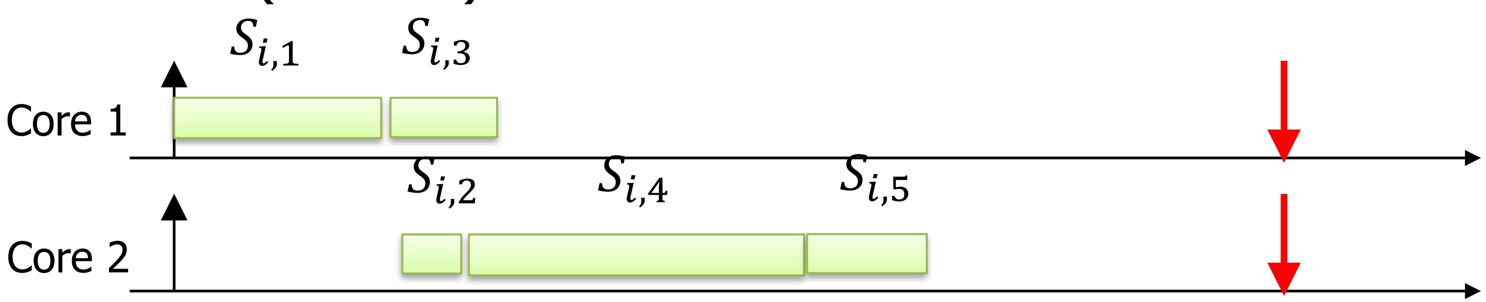
Task parameters:

$$\tau_i = (\langle C_{i,1}, C_{i,2}, \dots, C_{i,m_i} \rangle, D_i, T_i)$$



Precedence constraint between task's segments
(a.k.a. **intra-task precedence constraint**)

Multi-core (2 cores) schedule



Preemptive v.s. non-preemptive execution

- **Preemptive execution**
  - The execution of a task may be forcefully preempted (stopped) by the scheduler
- **Non-preemptive execution**
  - A task does not leave the processor (or resource) until it completes or suspends itself
  - **Non-preemptive execution** is also called "**cooperative execution**" if the task which is running on the platform voluntarily yields the processor so that other tasks can execute

Self-suspension

Self-suspension: At some points in the code, task may suspend itself to access I/O, wait for results from a GPU or other another task, etc.

```
While(true)
{
    int max = -1;
    int * array = read10Data();
    for (int i=0; i < 10; i++)
        if (max < 0 || array[i] > max)
            max = array[i];
    send(max);
    sleep (100, ms);
}
```

Example: A **non-busy-waiting read** results in self-suspension.

In a non-busy-waiting read, the task initiates a request and then waits until it is notified (when data is ready).

Disclaimer

- The course slides came from Koen Langendoen who compiled materials found on the web and obtained from friendly colleagues together.

- Thanks to
  - Koen Langendoen
  - Akos Ledeczi
  - Gerd Gross
  - Giorgio Buttazzo
  - Reinder Bril
  - Zonghua Gu
  - and many others