Real-Time Scheduling analysis

Frank Singhoff

Office C-202

University of Brest, France

singhoff@univ-brest.fr

Summary

Real-time systems

- "**Real-time systems** are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but **also on the time** at which the results are produced" [STA 88].

- **Properties we look for**:
  - **Functions must be predictable**: the same data input will produce the same data output.
  - **Timing behavior must be predictable**: must meet temporal constraints (e.g. deadline, response time).

$\Longrightarrow$ Predictable means ... we can **compute** the system temporal behavior **before** execution time.

What is real-time scheduling theory (1)

- **Many real-time systems are built with operating systems providing multitasking facilities, in order to:**
  - Ease the design of complex systems (one function = one task).
  - Increase efficiency (I/O operations, multi-processor architecture).
  - Increase re-usability.

What is real-time scheduling theory (1)

- **Many real-time systems are built with operating systems providing multitasking facilities, in order to:**
  - Ease the design of complex systems (one function = one task).
  - Increase efficiency (I/O operations, multi-processor architecture).
  - Increase re-usability.

But, multitasking makes the predictability analysis difficult to do : we must take the task scheduling into account in order to check temporal constraints $\Longrightarrow$ schedulability analysis.

What is real-time scheduling theory (2)

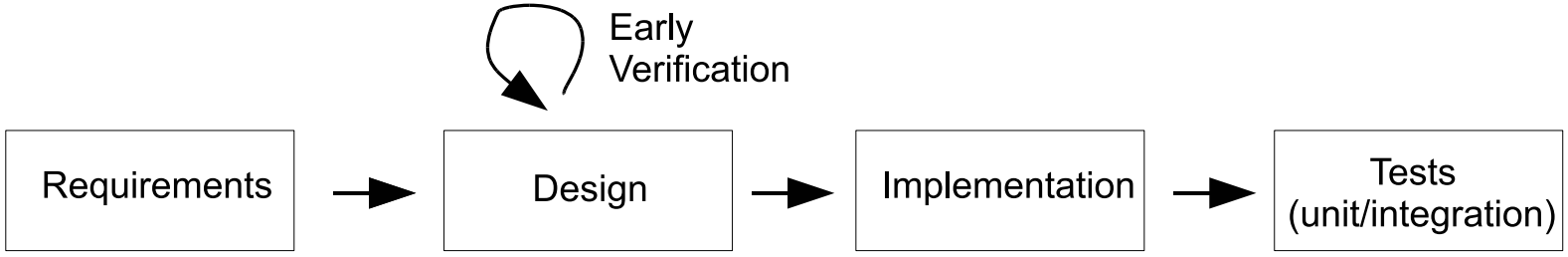
- **Example of a software embedded into a car:**

1. Tasks are released several times and have a job to do for each release.
2. Each task completes its current job before it has been released for the next one.
3. A task displays every 100 milliseconds the current speed of the car.
4. A task reads a speed sensor every 250 milliseconds.
5. A task performs an engine monitoring algorithm every 500 milliseconds.

$\Longrightarrow$ How can we check that tasks will be run at the proper rate? Do they meet their timing requirements? Do we have enough processor resource?

$\Longrightarrow$ If the system is complex (e.g. large number of tasks), the designer must be helped to perform such an analysis.

What is real-time scheduling theory (3)



- The real-time scheduling theory is a framework which provides:

1. **Algorithms to share a processor** (or any resources) by a set of tasks (or any resource users) according to some timing requirements $\Longrightarrow$ take urgency of the tasks into account.
2. **Analytical methods,** called **feasibility tests** or **schedulability tests**, which allow a system designer to early analyze/"compute" the system behavior before implementation/execution time.

Scheduling algorithms (1)

- **Different kinds of real-time schedulers:**
  - **On-line/off-line scheduler:** the scheduling is computed before or at execution time?
  - **Static/dynamic priority scheduler:** priorities may change at execution time?
  - **Preemptive or non preemptive scheduler:** can we stop a task during its execution ?
    1. Preemptive schedulers are more efficient than non preemptive schedulers (e.g. missed deadlines).
    2. Non preemptive schedulers ease the sharing of resources.
    3. Overhead due to context switches.

Scheduling algorithms (2)

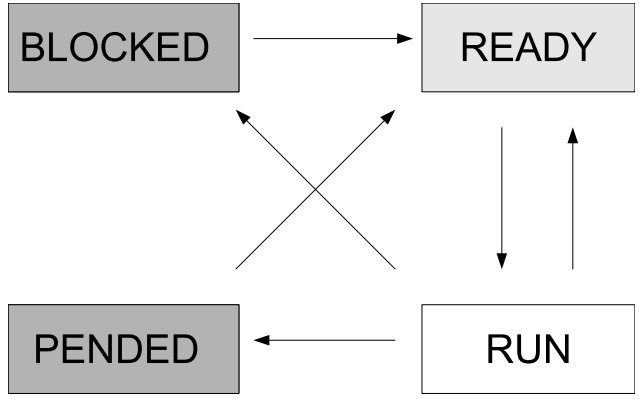
- **Different kinds of real-time schedulers:**
  - **Feasibility tests (schedulability tests) exist or not:** can we prove that tasks will meet deadlines before execution time?
  - **Complexity:** can we apply feasibility tests on large systems (e.g. large number of tasks)?
  - **Suitability:** can we implement the chosen scheduler in a real-time operating system?

Scheduling algorithms (3)

- What we look for when we choose a scheduler: we can compare them according to their ability to meet task deadlines.

  - A valid schedule is a schedule in which all task deadlines are met.
  - A feasible task set is a task set for which a valid schedule can be find.
  - **Optimality:** a scheduler a is optimal if it is able to find a valid schedule, each time a valid schedule exists for a task set.
  - **Dominant:** a is dominant comparing to b if all task sets that are feasible by scheduler b, are also feasible by a and if it exists some task sets that are feasible by a but not by b.
  - **Equivalent** : a and b are equivalent if all task sets that are feasible by a are also feasible by b and respectively.
  - **Incomparable** : a and b are incomparable is a can find a valid schedule for some task sets that b is not able to find and respectively.

Models of task (1)

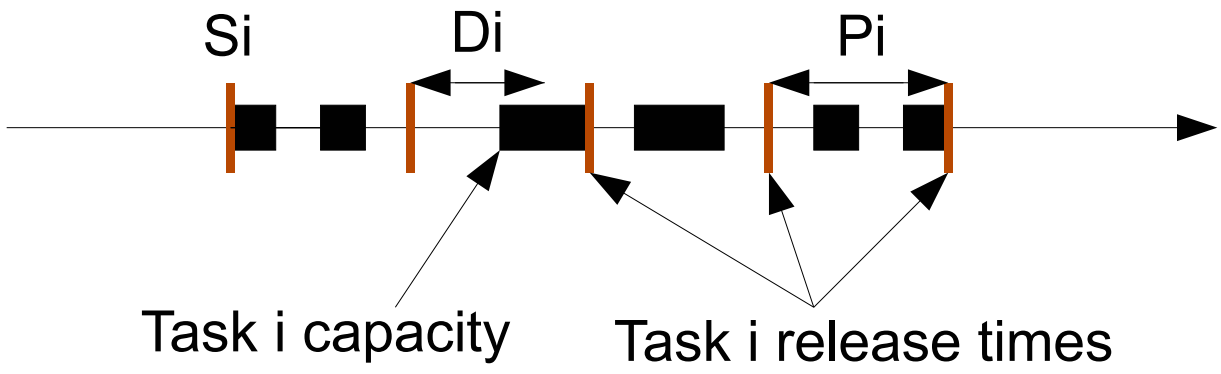


- **Task**: sequence of statements + data + execution context (processor and MMU). Usual states of a task.

- **Usual task types**:

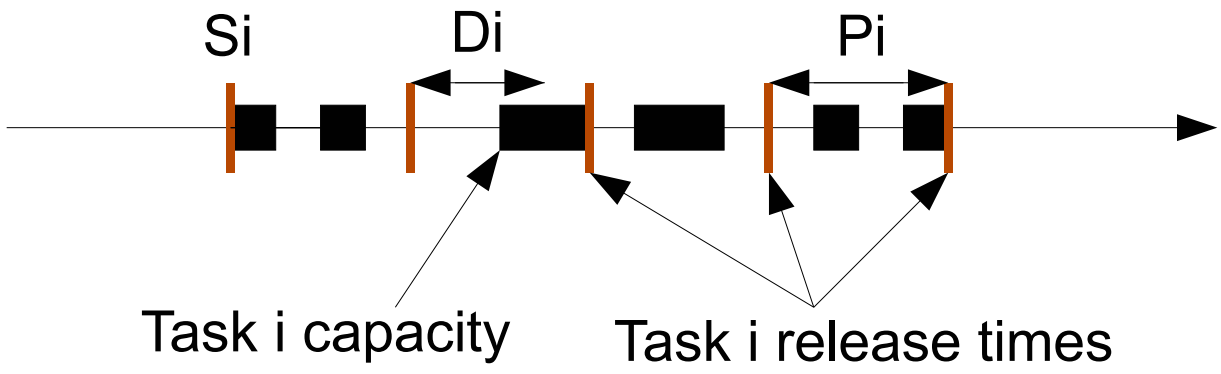
  - Urgent or/and critical tasks.
  - Independent tasks or dependent tasks.
  - Periodic and sporadic tasks (critical tasks). Aperiodic tasks (non critical tasks).

Models of task (2)



- **Usual parameters of a periodic task i:**
  - Period: P_i (duration between two periodic release times). A task starts a job for each release time.
  - Static deadline to meet: D_i, timing constraint relative to the period/job.
  - First task release time (first job): S_i.
  - Worst case execution time of each job: C_i (or capacity).
  - Priority: allows the scheduler to choose the task to run.

Models of task (3)



- **Assumptions for this lecture (synchronous periodic task with deadlines on requests) [LIU 73]:**

1. All tasks are periodic.
2. All tasks are independent.
3. $\forall i : P_i = D_i$: a task must end its current job before its next release time.
4. $\forall i : S_i = 0 \implies$ called critical instant (worst case on processor demand).

Verification of a real-time system

- How to perform verification of timing constraints of a real-time system (example):

1. Define/model hardware architecture and execution environment: capacity of the memory, processor, operating system features (and then the scheduler).
2. Implement functions (source code).
3. Design software architecture and software deployment on the hardware. Lead to a design of the software as a set of tasks, with their parameters and constraints.
4. Verify schedulability.

5. If task set is not feasible, go back to 1, 2 or 3.

Summary

1. Introduction.
2. Classical real-time scheduler
3. Shared resources and precedence constraints.
4. Schedulability tools and architecture languages.
5. Real-time scheduling for multiprocessor and distributed systems.
6. Conclusion.
7. References.

Usual real-time schedulers

1. **Fixed priority scheduler:** Rate Monotonic priority assignment (sometimes called Rate Monotonic Scheduling or Rate Monotonic Analysis, RM, RMS, RMA).
2. **Dynamic priority scheduler:** Earliest Deadline First (or EDF).

Fixed priority scheduling (1)

- **Assumptions/properties of fixed priority scheduling :**
  - Scheduling based on fixed priority $\Longrightarrow$ static and critical applications.
  - Priorities are assigned at design time (off-line).
  - Efficient and simple feasibility tests.
  - Scheduler easy to implement into real-time operating systems.
- **Assumptions/properties of Rate Monotonic assignment:**
  - Optimal assignment in the case of fixed priority scheduling.
  - Periodic tasks only.

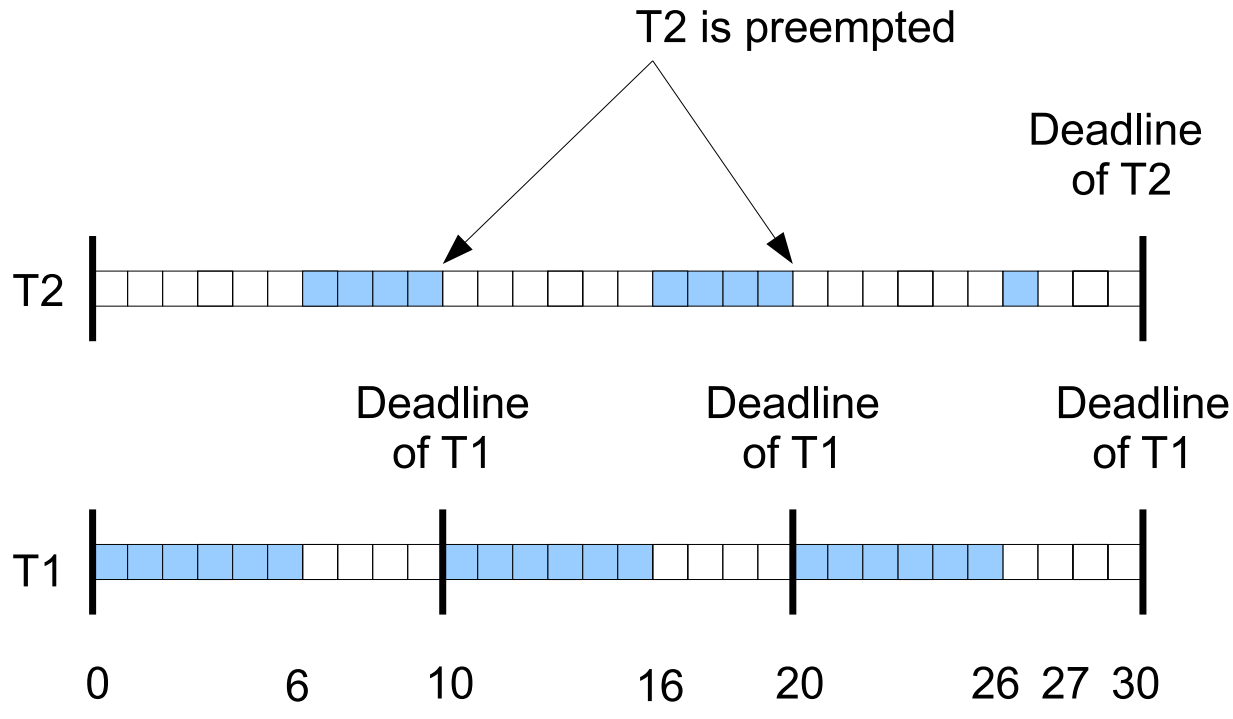
Fixed priority scheduling (2)

- **How does it work:**

1. **"Rate monotonic" task priority assignment:** the highest priority tasks have the smallest periods. Priorities are assigned off-line (e.g. at design time, before execution).
2. **Fixed priority scheduling:** at any time, run the ready task which has the highest priority level.

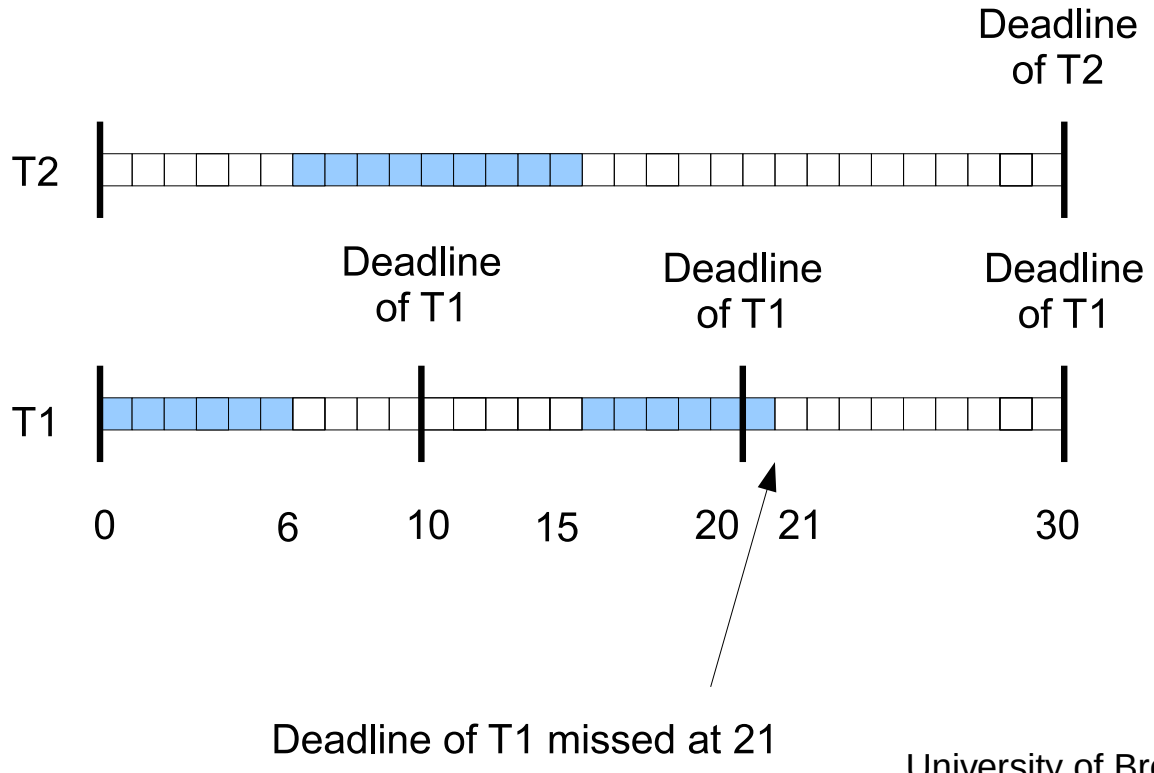
Fixed priority scheduling (3)

- Rate Monotonic assignment and preemptive scheduling:
  - Assuming VxWorks priority levels (high=0 ; low=255)
  - T1 : C1=6, P1=10, Prio1=0
  - T2 : C2=9, P2=30, Prio2=1



Fixed priority scheduling (4)

- Rate Monotonic assignment and non preemptive scheduling:
  - Assuming VxWorks priority levels (high=0 ; low=255)
  - T1 : C1=6, P1=10, Prio1=0
  - T2 : C2=9, P2=30, Prio2=1



Fixed priority scheduling (5)

- **Feasibility/Schedulability tests:**

1. **Run simulations on hyperperiod** = $[0, LCM(P_i)]$. Sufficient and necessary (exact result). Any priority assignment and preemptive/non preemptive scheduling.

2. **Processor utilization factor test**:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1)$$

   Rate Monotonic assignment and preemptive scheduling. Sufficient but not necessary. Does not compute an exact result.

3. **Task worst case response time, noted** $r_i \implies$ delay between task release time and task end time. Can compute an exact result. Any priority assignment and preemptive scheduling.

Fixed priority scheduling (6)

- **Compute r_i, the task worst case response time:**
  - Assumptions: preemptive scheduling, synchronous periodic tasks.
  - task i response time $=$ task i capacity + delay the task i has to wait due to higher priority task j. Or:

$$r_i = C_i + \sum_{j \in hp(i)} WaitingTime_j$$

or:

$$r_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil C_j$$

  - $hp(i)$ is the set of tasks which have a higher priority than task i. $\lceil x \rceil$ returns the smallest integer not smaller than x.

Fixed priority scheduling (7)

- **To compute task response time**: compute w_i^k with:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{P_j} \right\rceil C_j$$

- Start with $w_i^0 = C_i$.
- Compute w_i^1, w_i^2, w_i^3, ... w_i^k upto:
  - If $w_i^k > P_i$. No task response time can be computed for task i. Deadlines will be missed !
  - If $w_i^k = w_i^{k-1}$. w_i^k is the task i response time. Deadlines will be met.

Fixed priority scheduling (8)

- **Example:** T1 (P1=7, C1=3), T2 (P2=12, C2=2), T3 (P3=20, C3=5)

  - $w_1^0 = C1 = 3 \Longrightarrow r_1 = 3$
  - $w_2^0 = C2 = 2$
  - $w_2^1 = C2 + \left\lceil \frac{w_2^0}{P1} \right\rceil C1 = 2 + \lceil \frac{2}{7} \rceil 3 = 5$
  - $w_2^2 = C2 + \left\lceil \frac{w_2^1}{P1} \right\rceil C1 = 2 + \lceil \frac{5}{7} \rceil 3 = 5 \Longrightarrow r_2 = 5$
  - $w_3^0 = C3 = 5$
  - $w_3^1 = C3 + \left\lceil \frac{w_3^0}{P1} \right\rceil C1 + \left\lceil \frac{w_3^0}{P2} \right\rceil C2 = 5 + \lceil \frac{5}{7} \rceil 3 + \lceil \frac{5}{12} \rceil 2 = 10$
  - $w_3^2 = 5 + \lceil \frac{10}{7} \rceil 3 + \lceil \frac{10}{12} \rceil 2 = 13$
  - $w_3^3 = 5 + \lceil \frac{13}{7} \rceil 3 + \lceil \frac{13}{12} \rceil 2 = 15$
  - $w_3^4 = 5 + \lceil \frac{15}{7} \rceil 3 + \lceil \frac{15}{12} \rceil 2 = 18$
  - $w_3^5 = 5 + \lceil \frac{18}{7} \rceil 3 + \lceil \frac{18}{12} \rceil 2 = 18 \Longrightarrow r_3 = 18$

Fixed priority scheduling (9)

- How to schedule both periodic and aperiodic tasks, both critical and non critical functions:

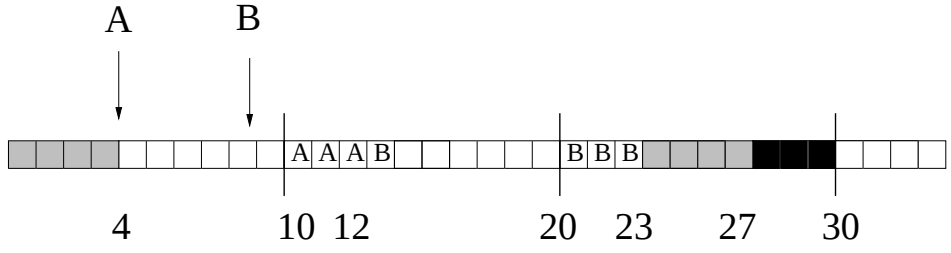
1. If aperiodic tasks are not urgent: give them a low priority level.
2. If aperiodic tasks are urgent: use aperiodic tasks servers.

Fixed priority scheduling (10)

- **Aperiodic tasks server**: periodic task devoted to the scheduling of aperiodic tasks.

T1 : C1=4 ; P1=20 (grey)
T2 : C2=12 ; P2=30 (white)
S : CS=4 ; PS=10;
A is released at t=4 ; CA=3
B is released at t=9 ; CB=4

Black=idle



- **Polling server:** run aperiodic task arrived before server activation. Does not use processor if no aperiodic task is present. Capacity is lost if no aperiodic task is present. Server stops aperiodic execution when its capacity is exhausted.
- **Many other servers:** sporadic server, priority exchange server, ...

Fixed priority scheduling (11)

- **Analysis of the car with embedded software example:**

1. $Tdisplay$: task which displays speed. P=100, C=20.
2. $Tspeed$: task which reads speed sensor. P=250, C=50.
3. $Tengine$: task which runs an engine monitoring program. P=500, C=150.

- **Processor utilization test:**

$U = \sum_{i=1}^{n} \frac{C_i}{P_i} = 20/100 + 150/500 + 50/250 = 0.7$

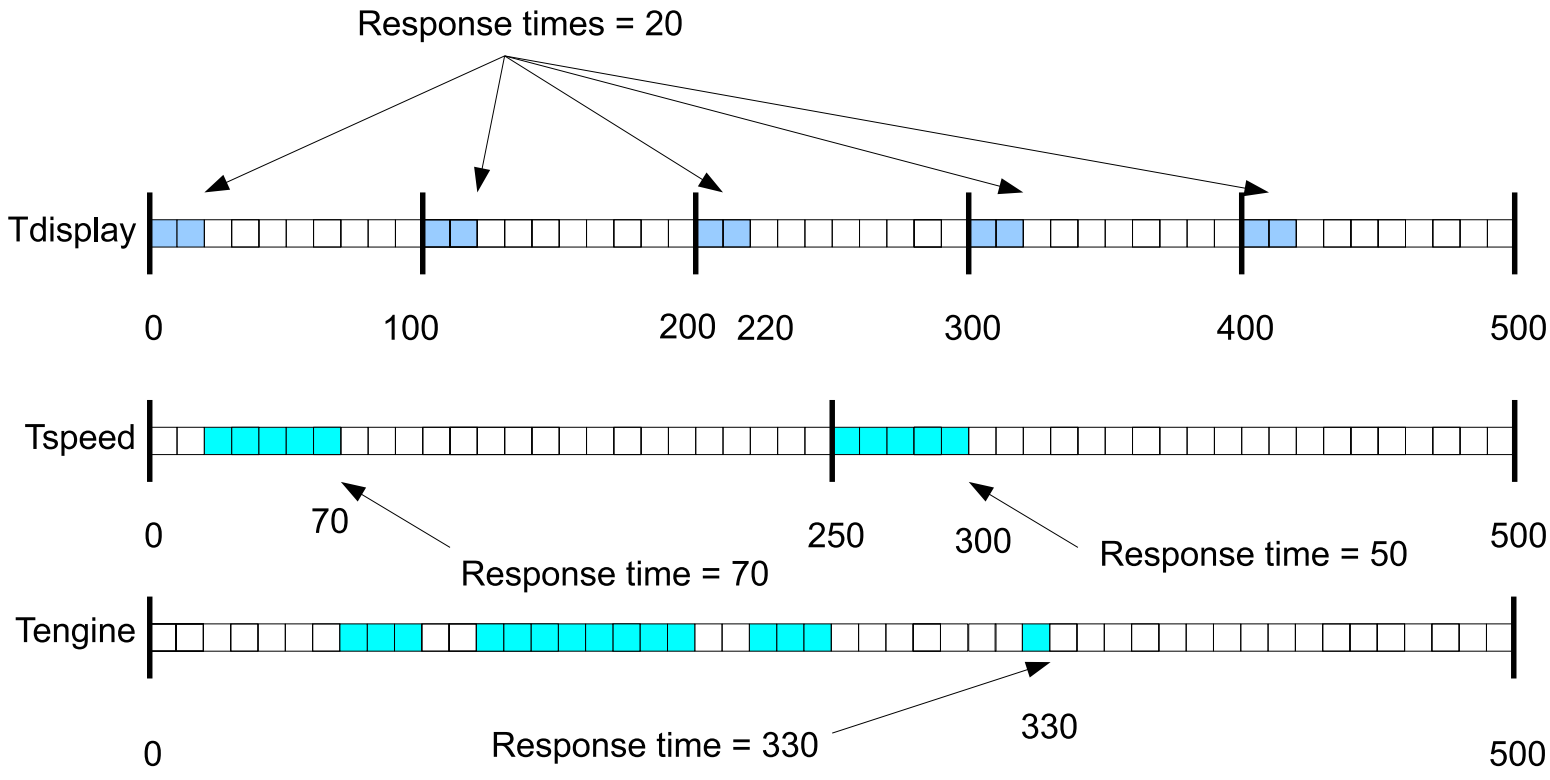
$Bound = n(2^{\frac{1}{n}} - 1) = 3(2^{\frac{1}{3}} - 1) = 0.779$

$U \leq Bound \Longrightarrow$ deadlines will be met.

- **Task response time:** $r_{Tengine} = 330$, $r_{Tdisplay} = 20$, $r_{Tspeed} = 70$.

Fixed priority scheduling (12)

- **... and check on the computed scheduling**



- Run simulations on **scheduling period** = $[0, LCM(P_i)]$ = $[0, 500]$.

Earliest Deadline First (1)

- **Assumptions and properties:**
  - Dynamic priority scheduler $\Longrightarrow$ suitable for dynamic real-time systems.
  - Is able to schedule both aperiodic and periodic tasks.
  - Optimal scheduler: can reach 100 % of the cpu usage.
  - But difficult to implement into a real-time operating system.
  - Becomes unpredictable if the processor is over-loaded $\Longrightarrow$ not suitable for hard-critical real-time systems.

Earliest Deadline First (2)

- **How does it work:**

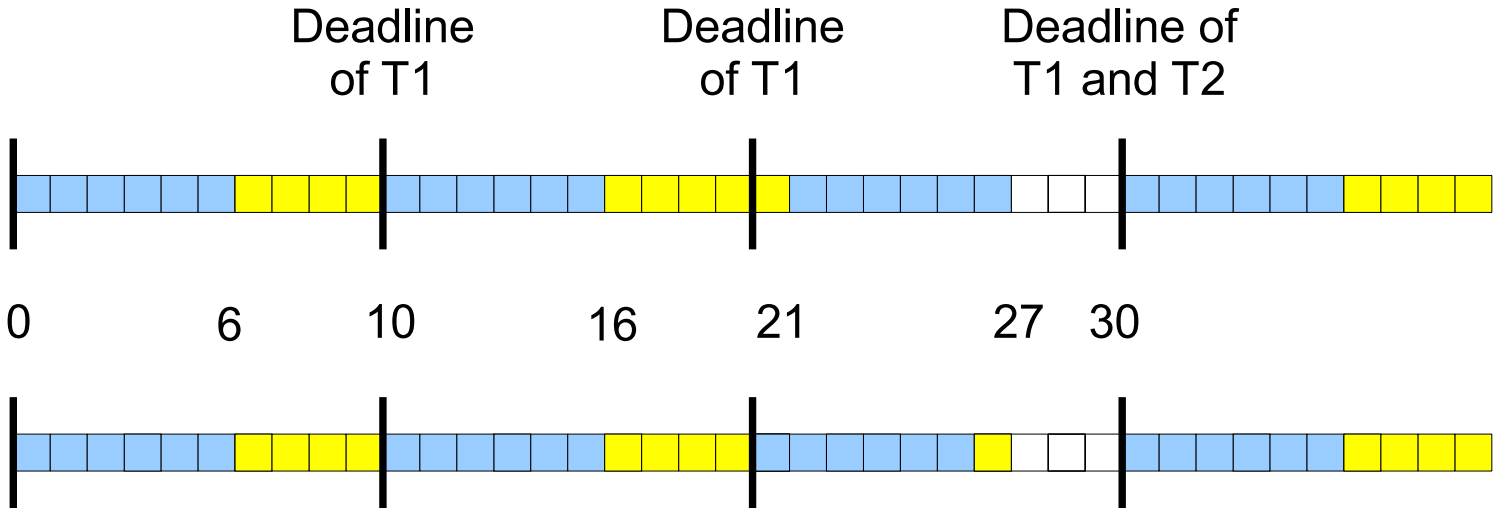
1. **Compute task priorities (called "dynamic deadlines")** $\Longrightarrow D_i(t)$ is the priority/dynamic deadline of task i at time t:
   - Aperiodic task : $D_i(t)$ = D_i + S_i.
   - Periodic task: $D_i(t)$ = k + D_i, where k is the task release time before t.

2. **Select the task:** at any time, run the ready task which has the shortest dynamic deadline.

Earliest Deadline First (3)

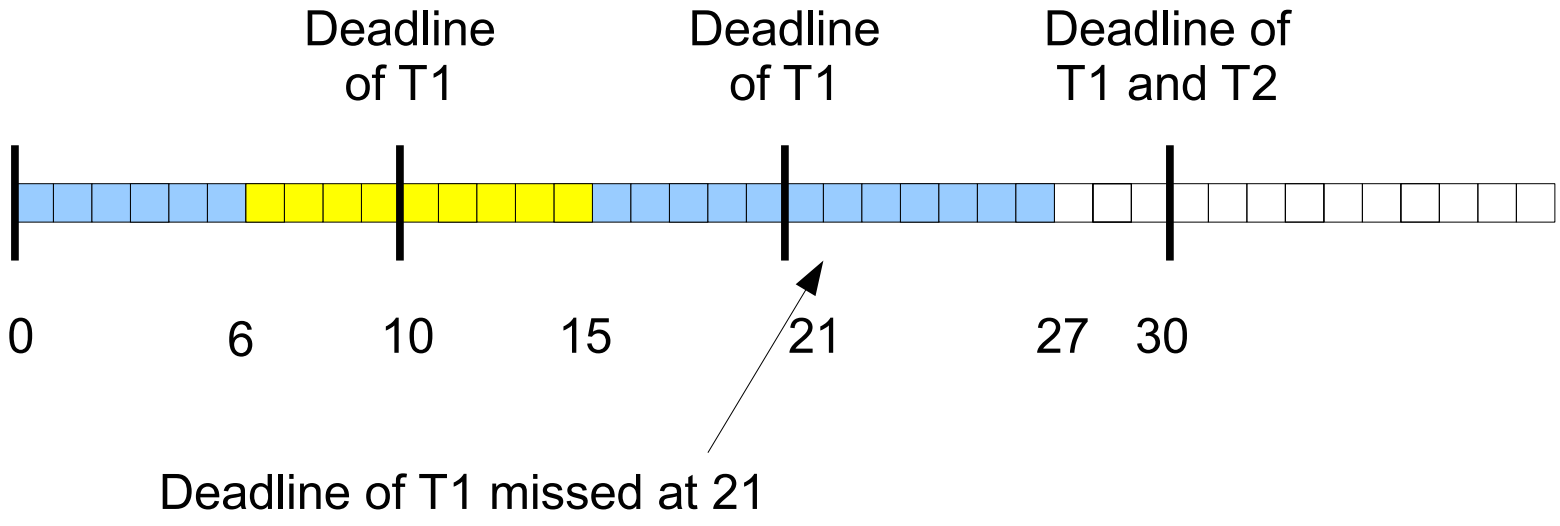
- Preemptive case: (T1/blue, T2/yellow, C1=6; P1=10; C2=9; P2=30)

t	$D_1(t)$	$D_2(t)$
0..9	$k + D_1 = 0 + 10 = 10$	$k + D_2 = 0 + 30 = 30$
10..19	$k + D_1 = 10 + 10 = 20$	30
20..29	$k + D_1 = 20 + 10 = 30$	30



Earliest Deadline First (4)

- Non preemptive case:



Earliest Deadline First (5)

- **Feasibility tests/schedulability tests:**

1. Run simulations on **hyperperiod** = $[0, LCM(P_i)]$. Sufficient and necessary (exact result).

2. **Processor utilization factor test** (e.g. preemptive case):

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

   Sufficient and necessary. Difficult to use in real life applications. Compute an exact result.

3. **Task response time:** a bit more complex to compute (dynamic scheduler) !

EDF vs RM: summary [BUT 03]

- Aperiodic task = non critical task.
- Periodic task = critical task.

	FP/RM	EDF
Applications	critical, static	dynamic, less critical
RTOS implementation	easy	more difficult
Tasks	Periodic only	Aperiodic and periodic
Efficiency	upto 69 %	upto 100 %
Predictability	high	less than FP/RM if $U > 1$

- Warning: the real picture is more complicated: aperiodic tasks scheduling with FP/RM, U=1 with FP/RM, ...

Summary

1. Introduction.
2. Classical real-time schedulers.
3. Shared resources and precedence constraints.
4. Schedulability tools and architecture languages.
5. Real-time scheduling for multiprocessor and distributed systems.
6. Conclusion.
7. References.

Task dependencies

- **Usual dependencies:**

1. Shared resources.
2. Precedences between tasks.

Shared resource support (1)

- **In real-time scheduling theory, a shared resource is modeled by a semaphore:**

Task i

```
P(sem)
access_shared_resource(k)
V(sem)
```

Task j

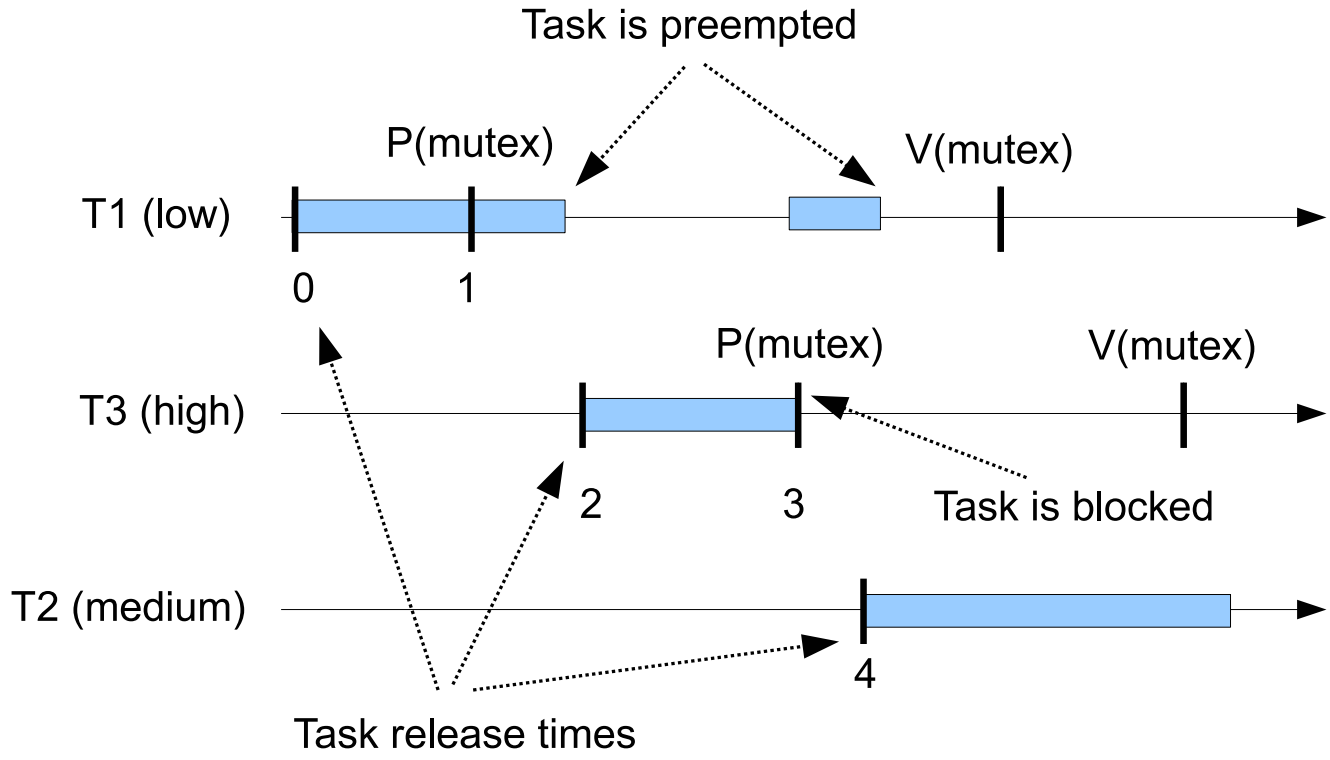
```
P(sem)
access_shared_resource(k)
V(sem)
```

```
P(s)
    {
    s.cpt=s.cpt−1;
    if (s.cpt<0)
        {
        add_task_to_the_queue(s.file);
        task_is_blocked();
        }
    }
```

```
V(s)
    {
    s.cpt=s.cpt+1;
    if (s.cpt<=0)
        {
        remove_task_from_the_queue(s.file)
        task_is_released();
        }
    }
```

- Semaphore = counter + a FIFO queue.
- A semaphore may be used to model a critical section.
- We use special semaphores in real-time scheduling systems.

Shared resource support (2)



- **What is Priority inversion:** a low priority task blocks a high priority task ... allowing a medium priority task to held the processor $\Longrightarrow$ a non critical task runs before a critical task !
- B_i = bound on the shared resource waiting time.

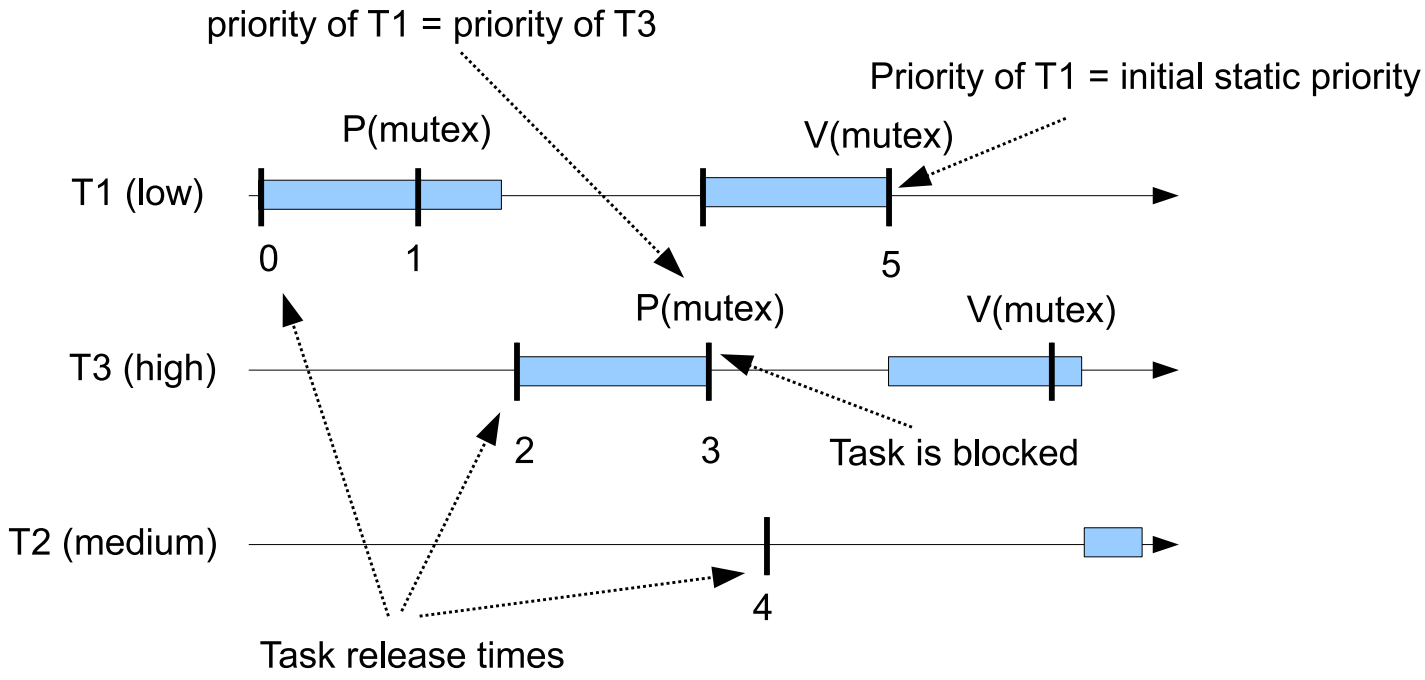
Shared resource support (3)

- How to reduce priority inversion?
- How long a task must wait for the access to a shared resource? How to compute B_i?

$\Longrightarrow$ **To reduce priority inversion, we use priority inheritance.**

- Priority inheritance protocols provide a specific implementation of $P()$ and $V()$ of semaphores.

Shared resource support (4)

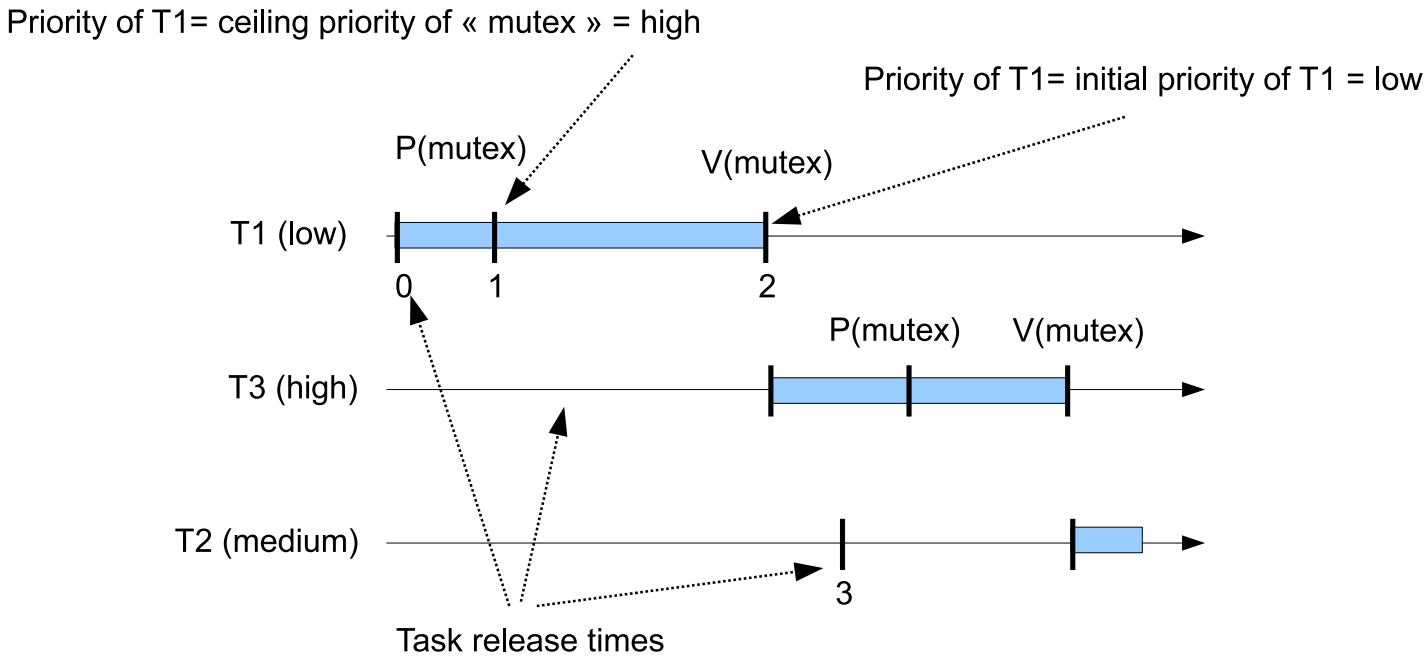


- **Priority Inheritance Protocol or PIP:**
  - A task which blocks a high priority task due to a critical section, sees its priority to be increased to the priority level of the blocked task.
  - $B_i =$ sum of the critical sections of the tasks which have a priority smaller than i.

Shared resource support (5)

- PIP can not be used with more than one shared resource due to deadlock.
- PCP (Priority Ceiling Protocol) [SHA 90] is used instead.
- Implemented in most of real-time operating systems (e.g. VxWorks).
- Several implementations of PCP exists: OPCP, ICPP, ...

Shared resource support (6)



- **ICPP (Immediate Ceiling Priority Protocol):**
  - Ceiling priority of a resource = maximum static priority of the tasks which use it.
  - Dynamic task priority = maximum of its own static priority and the ceiling priorities of any resources it has locked.

Shared resource support (7)

- **How to take into account the blocking time B_i with the processor utilization factor test:**
  - Preemptive RM feasibility test:

$$\forall\, i, 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{C_k}{P_k} + \frac{C_i + B_i}{P_i} \leq i(2^{\frac{1}{i}} - 1)$$

  - Preemptive EDF feasibility test:

$$\forall\, i, 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{C_k}{P_k} + \frac{C_i + B_i}{P_i} \leq 1$$

Shared resource support (8)

- **How to take into account the blocking time B_i with the worst case response time r_i of the task i (example of a preemptive RM scheduler):**

$$r_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil C_j$$

with $hp(i)$ the set of tasks which has a lowest priority than task i.

- Which can be computed by:

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{P_j} \right\rceil C_j$$

Precedence constraints (1)

- **Many different approaches:**

1. Change initial task release times (S_i parameter) $\Longrightarrow$ but require to apply specific feasibility tests since we have no critical instant any more.

2. Express task dependences with usual task parameters:
   (a) Assign priorities according to dependences (Chetto/Blazewicz [BLA 76, CHE 90]).
   (b) Change deadlines according to dependences (Chetto/Blazewicz).

3. Use of the "Jitter" parameter [TIN 94]. Useful to compute worst case response time.

4. Specific heuristic (e.g. Xu et Parnas [XU 90]).

Precedence constraints (2)

- Idea of the Blazewicz [BLA 76] and Chetto et al. [CHE 90] approach: make tasks independent by expressing their dependences with usual task parameters => update task parameters.

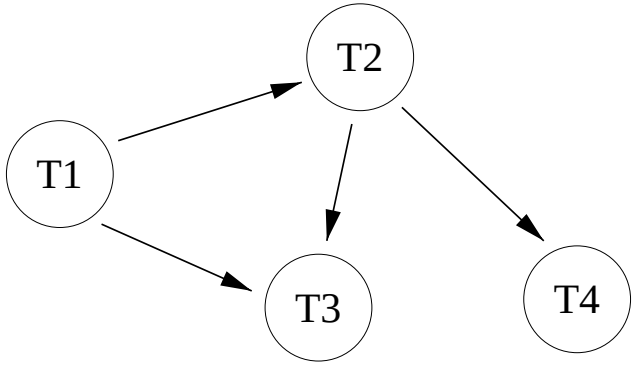
- Assumptions: either aperiodic tasks or tasks with the same period.

- Parameter updates:

1. With RM:
   - $\forall\, i, j \mid i \prec j : priority_i > priority_j$

2. With EDF:
   - $D_i^* = min(D_i, min(\forall\, j \mid i \prec j : D_j^* - C_j))$.

Precedence constraints (3)

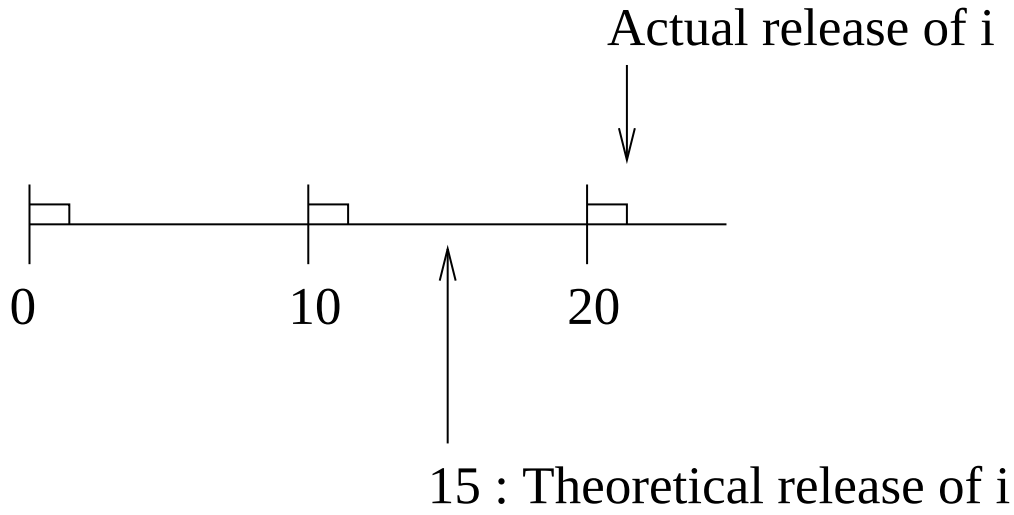


	C_i	D_i	D_i^*
T_4	2	14	14
T_3	1	8	8
T_2	2	10	7
T_1	1	5	5

- Example : EDF + aperiodic tasks.
  - $D_4^* = 14$; $D_3^* = 8$;
  - $D_2^* = min(D_2, D_3^* - C_3, D_4^* - C_4) = min(10, 8-1, 14-2) = 7$;
  - $D_1^* = min(D_1, D_2^* - C_2, D_3^* - C_3) = min(5, 7-2, 8-1) = 5$;

Precedence constraints (4)

- Use of a **Jitter**. Jitter was proposed to model the behavior of a system timer. System timer can be seen as a periodic task $P_{timer} = 10\ ms, C_{timer} = 3\ ms$.
- Assume we would like to release task i at time $t = 15\ ms$.



Actual release time of task i=23 ms. Its jitter is $J_i = 8\ ms$.

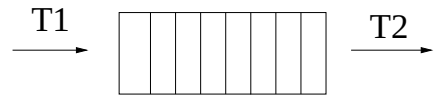
- To compute its worst case response time:

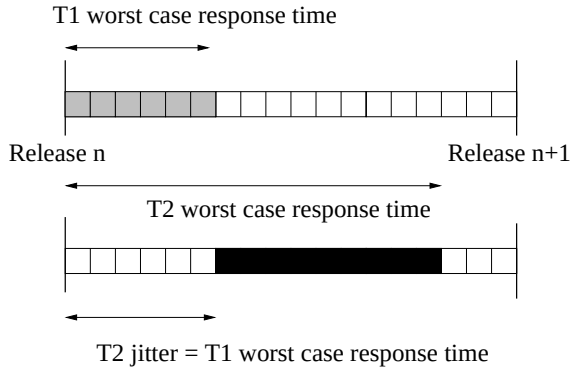
$$r_i = w_i + J_i$$

$$w_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i + J_j}{P_j} \right\rceil C_j$$

Precedence constraints (5)

T1 : C1=6 ; P1=18 (grey)
T2 : C2=9 ; P2=18 (black)





- **Example of a producer/consumer:**
  - T1 and T2 are released every 18 units of time.
  - T1 reads a sensor and sends the data to T2, which displays it to a screen.
  - Then, T2 must be released on T1 completion.
  - What is the worst case response time of T2?

Summary

1. Introduction.
2. Classical real-time schedulers.
3. Shared resources and precedence constraints.
4. Schedulability tools and architecture languages.
5. Real-time scheduling for multiprocessor and distributed systems.
6. Conclusion.
7. References.

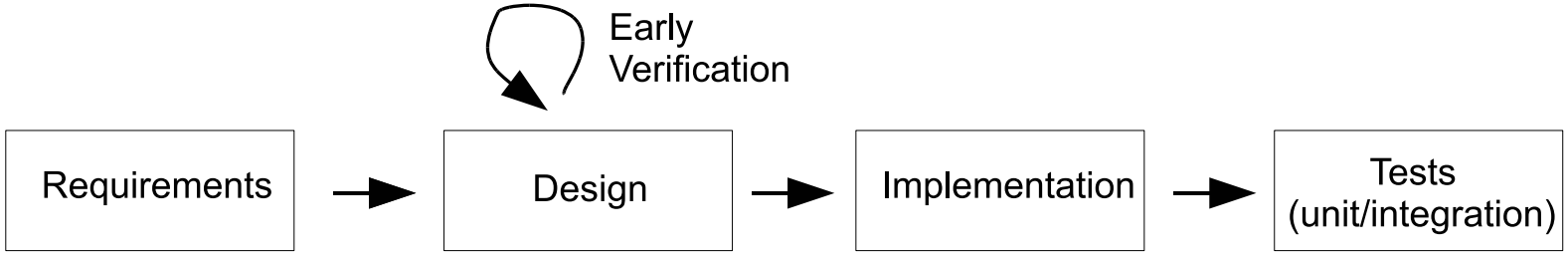
Verification of a real-time system

- How to perform verification of timing constraints of a real-time system (example):

1. Define/model hardware architecture and execution environment: capacity of the memory, processor, operating system features (and then the scheduler).
2. Implement functions (source code).
3. Design software architecture and software deployment on the hardware. Lead to a design of the software as a set of tasks, with their parameters and constraints.
4. Verify schedulability.

5. If task set is not feasible, go back to 1, 2 or 3.

Scheduling analysis tools (1)



- **Scheduling analysis tools must provide:**
  - A way to model the system architecture to be analyzed.
  - Analysis tools based on:
    1. **Algebraic/analytical tools:** run feasibility tests.
    2. **Model-checking:** compute all reachable states of the system and analyze this set of states.
    3. **Scheduling simulations:** compute scheduling time-lines and analyze them (partial checking).

Scheduling analysis tools (2)

	Feasibility tests	Model checking	Scheduling simulation
Provide proofs	yes	yes	no
Suitable for large system	yes	it depends	it depends
Suitable for specific scheduler/task model	no	yes	yes
Easy to use	yes	no	it depends

- **What kind of tool should we use?** Scheduling simulation tools vs feasibility tests tools vs model-checking tools? All of them ... they are complementary.

Scheduling analysis tools (3)

- **Examples of both commercial and open-source tools:**
  - MAST (University of Cantabria, `http://mast.unican.es/`).
  - Rapid-RMA (Tri-Pacific Software Inc, `http://www.tripac.com/`).
  - Times (University of Uppsala, `http://www.timestool.com/`)
  - Cheddar (University of Brest and Ellidiss Technologies, `http://beru.univ-brest.fr/~singhoff/cheddar/`).
  - ...
- **There are some tools, but they are not so easy to use:**
  - When and how to use them?
  - Require deep real-time scheduling analysis theory skills.
  - Relationships with design tools and programming language features.

Scheduling analysis tools (4)

- **Cheddar[SIN 04]** = Ada framework implementing performance analysis tools and the required modeling features.
- **Support of standard architecture design languages is better for tool interoperability.**
- **Cheddar supports the following architecture design languages:**
  - Support for AADL, PPOOA and UML/Marte architecture design languages.
  - Works with model editors such as Stood (Ellidiss Tech.) or TASTE (ESA/Ellidiss Tech.).
  - Also provides its own architecture design language which allows the modeling of real-time schedulers.
- **Analysis with both feasibility tests and scheduling simulation.**

Scheduling analysis tools (5)

- **Architecture and Analysis Design Language (AADL):**
  - International standard from the SAE (Society of Automotive Engineers): standard number AS-5506.
  - Version 1 published in 2004, version 2 in 2008.
  - See http://www.aadl.info for extra information.
  - Architecture design languages for critical real-time systems:
    1. Concurrency.
    2. Modeling both hardware and software components (to define available hardware resources).
    3. Expressing and enforcing timing constraints.

Scheduling analysis tools (6)

- **AADL component:**
  - Model a software or an hardware entity of the system to implement. Reusable entity (type/interface, several implementations, packages).
  - A component may be composed of sub-components.
  - **Component relationships/dependencies:** features (part of the component interface) + connections.
  - **Component properties:** attributes required for the analysis or the implementation of the component.

$\Longrightarrow$ Architecture model with AADL = component hierarchy.

Scheduling analysis tools (7)

- **Declaring a component:**
  - **Component type:** interface (category, identifier/name, properties and features).
  - **Component implementation:** internal design (subcomponents, properties).
  - **Component categories:** model real-time abstractions (semantic, component behavior).
  - **Three families of category:** hardware (models components of the execution environment), software (models components of the software to implement), system (deployment of the software on the hardware).

Scheduling analysis tools (8)

- **Software component category:**
  - **thread:** flow of control running a program (Ada/VxWorks task, POSIX/Java thread, ...).
  - **data:** data implemented in the target programming language (C struct, C++/Java class, Ada record, ...).
  - **process:** models an address space (Unix process). Contains at least one thread.
  - **subprogram:** models an executable program that is ran sequentially (C function, Java method, Ada subprogram, ...). Related to a source code.

Scheduling analysis tools (9)

- **Example of software components:**

```
thread receiver
end receiver;

thread implementation receiver.impl
end receiver.impl;

thread analyser ...
thread implementation analyser.impl ...

process processing
end processing;

process implementation processing.others
subcomponents
   receive : thread receiver.impl;
   analyse : thread analyser.impl;
end processing.others;
```

Scheduling analysis tools (10)

- **Hardware component category:**
  - **processor :** models software and hardware entities required to schedule threads (DSP, core unit, dual-core processor, ...).
  - **memory :** models any device that is able to store data (hard-disk, RAM, EEPROM, Flash memory, ...).
  - **device :** any hardware device for which we do not want to describe its internal design (antenna, engine, aircraft door).
  - **bus :** models communication devices between components (Ethernet network, PCI bus, serial link, USB link,...).

- **Example:**

```
device antenna
end antenna;
```

```
processor leon2;
end leon2;
```

Scheduling analysis tools (11)

- **System component category:**
  - Allow to design the system as a set of independent components => reusability.
  - Specification of the deployment (software => hardware components).

Scheduling analysis tools (12)

- **Example of a system:**

```
thread implementation receiver.impl ...
process implementation processing.others ...
processor leon2  ...

system radar
end radar;

system implementation radar.simple
subcomponents
   main : process processing.others;
   cpu  : processor leon2;
properties
   Actual_Processor_Binding =>
      reference cpu applies to main;
end radar.simple;
```

Scheduling analysis tools (13)

- **Property:**
  - Typed attribute, assigned to components.
  - Property = identifier(name) + data type + related component.
  - Property sets. Property sets defined by the standard: $AADL\_Properties$ and $AADL\_Project$. User-defined property sets may also exist.
- **Example:**

```
property set AADL_Properties is
   Deadline : aadlinteger
      applies to (thread, device, ...);
   Source_Text : inherit list of aadlstring
      applies to (data, port, thread, ...);
   ...
end AADL_Properties;
```

Scheduling analysis tools (14)

- **Property assignment:**
  - Assign a value to the property, for a given component.
  - Value assignments can be made either in the component type or in the component implementation. Assignments may be inherited.
- **Example:**

```
thread receiver
properties
  Compute_Execution_Time => 3 .. 4 ms;
  Period => 150 ms;
end receiver;

thread implementation receiver.impl
properties
   Deadline => 150 ms;
end receiver.impl;
```

Scheduling analysis tools (15)

- **Component connections:** model communications or synchronizations between components.

  - **Features:** each feature models a type of synchronization/communication. Defined by a name, a category, a type of data, ...
  - **Category of feature** = types of communications/synchronizations:
    - event port: asynchronous signal communication.
    - data port/event data port: synchronous/asynchronous message communication.
    - subprogram parameter: parameters required to call a subprogram.
    - data access: shared data (data component).
    - subprogram access: Remote procedure call.
    - ...

Scheduling analysis tools (16)

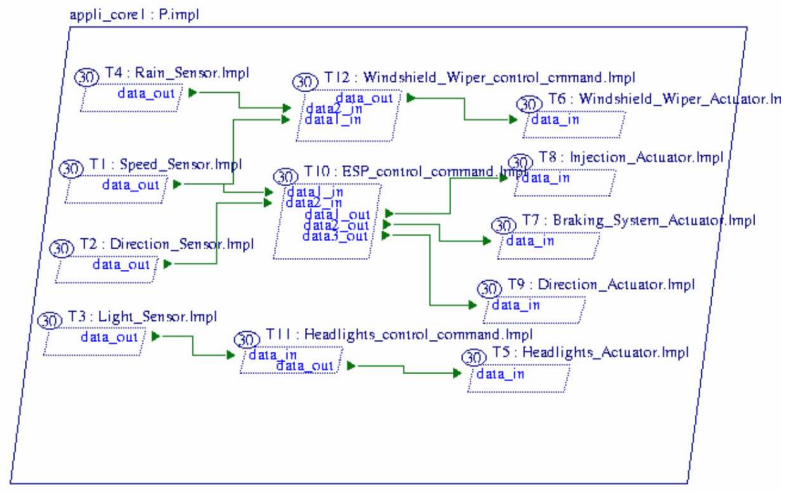
- **Shared data connection:**

```
process implementation processing.others
   subcomponents
      analyse : thread analyser.impl;
      display : thread display_panel.impl;
      a_data  : data shared_var.impl;
   connections
      data a_data -> display.share;
      data a_data -> analyse.share;
end processing.others;

data shared_var;
end shared_var;
data  implementation shared_var.impl
end shared_var.impl;

thread analyser
features
 share : requires data access shared_var.impl;
end analyser;
```

Scheduling analysis tools (17)



- **Let run some demos:**

1. "Car embedded software" architecture example: 1) edit an AADL model with STOOD 2) run scheduling analysis with Cheddar or AADLInspector.
2. Examples of POSIX scheduling: $SCHED\_RR$ and $SCHED\_FIFO$ policies.
3. Examples of user-defined schedulers: EDF, ARINC 653

Summary

1. Introduction.
2. Classical real-time schedulers.
3. Shared resources and precedence constraints.
4. Schedulability tools and architecture languages.
5. Real-time scheduling for multiprocessor and distributed systems.
6. Conclusion.
7. References.

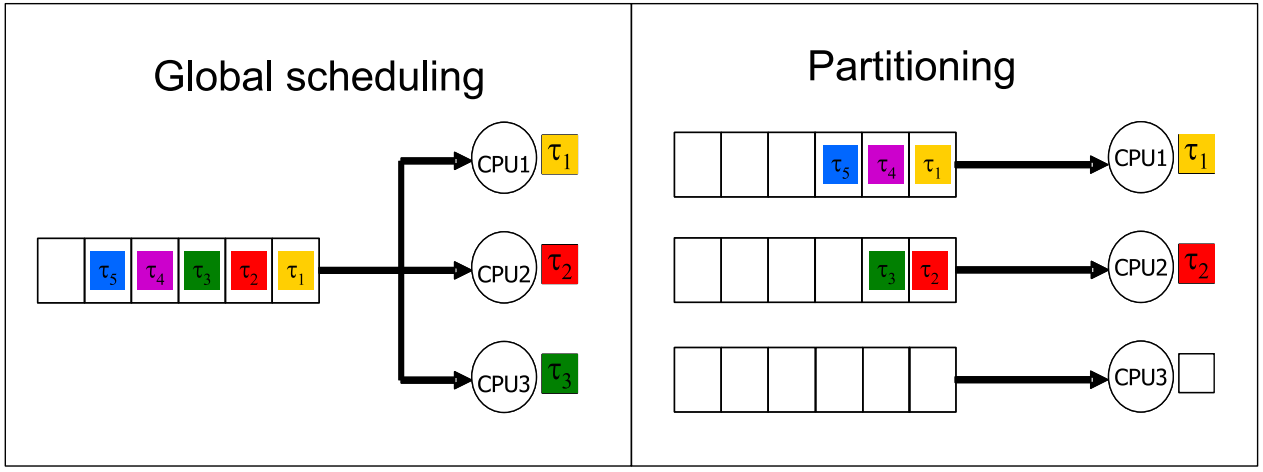
Multiprocessor/distributed (1)

- **Distributed systems:** "*A distributed system is a set of autonomous processors that are connected by a network and which have software to coordinate them self or share resources." Coulouris et al. [COU 94].* No shared memory. Message passing communication. A clock for each processor.
- **Multiprocessors systems:** a multiprocessors system is a set of autonomous processors which have software to coordinate them self and share resources but share the same clock and the same main memory.

- **Two scheduling approaches:**

1. Global scheduling.
2. Partitioning.

Multiprocessor/distributed (2)



1. **Global scheduling:** choose a task, and assign it to one of the idle processors (otherwise, preempt one of the running task). On-line processor assignment. With migration.
2. **Partitioning:** choose a processor for all tasks, and then run local scheduler on each processor. No migration. May apply end-to-end worst case response time analysis. Off-line processor assignment.

Multiprocessor/distributed (3)

- **Global scheduling:**
  - Few theoretical results (feasibility tests) compared to mono-processor real-time scheduling.
  - We can expect optimal processor usage: busy processors, less preemptions ... but task migrations.
  - Difficult to apply to heterogeneous systems: task migration, hardware, operating systems.
  - Well suited for multiprocessors architectures.

Multiprocessor/distributed (4)

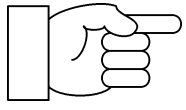
- **Partitioning:**
  - Theoretical foundations of mono-processor scheduling compliant with current system implementation.
  - Non optimal resource sharing: a processor may stay idle, even if a task waits for a processor elsewhere in the system.
  - Better reliability, deterministic behavior in case of failure: failures of a task may only imply failure on its processor.
  - End-to-end worst case response time verification is difficult to do (too pessimistic most of the time).
  - Partitioning is a NP-hard problem: Bin-packing.
  - Well suited for current synchronous distributed systems (e.g. aircraft, ARINC 653).

Multiprocessor/distributed (5)

Global scheduling

versus

partitioning:



The two approaches are incomparable!

Global scheduling (1)

- **Types de processors:**
  - **Identical:** processors have the same computing capability and run task at the same rate.
  - **Uniform:** each processor is characterised by its own computing capacity: for a processor of computing capacity s and for a work of duration t, the processor allocates $s \cdot t$ units of time to run the work.
  - **Specialised:** we define an execution rate for every uplet $(r_{i,j}, i, j)$: the work i requires $r_{i,j}$ units of time on the processor j.

  - Identical $\subset$ Uniform $\subset$ Specialized.
  - Specialized and uniform = heterogeneous processors.
  - Identical = homogeneous processors.

Global scheduling (2)

- **A global scheduler has to solve two problems[DAV 09, BER 07]:**

  - When and how to choose the processor to run tasks or jobs.
  - When and how to assign priority to tasks or jobs.

Global scheduling (3)

- **When migrations occur:**
  - **No migration:** a task must always run on a given processor=>partitioning.
  - **Job level migration:** each job of a task can be run on any processor. But a job started on a given processor can not be moved on another.
  - **Task level migration:** a task can run at any time on any processor.
- **Priority assignment:**
  - Fixed priority assigned to tasks (e.g. RM).
  - Fixed priority assigned to jobs (e.g. EDF).
  - Fully dynamic priority, may change at any time (e.g. LLF).

Global scheduling (4)

- **Two types of scheduling algorithms:**

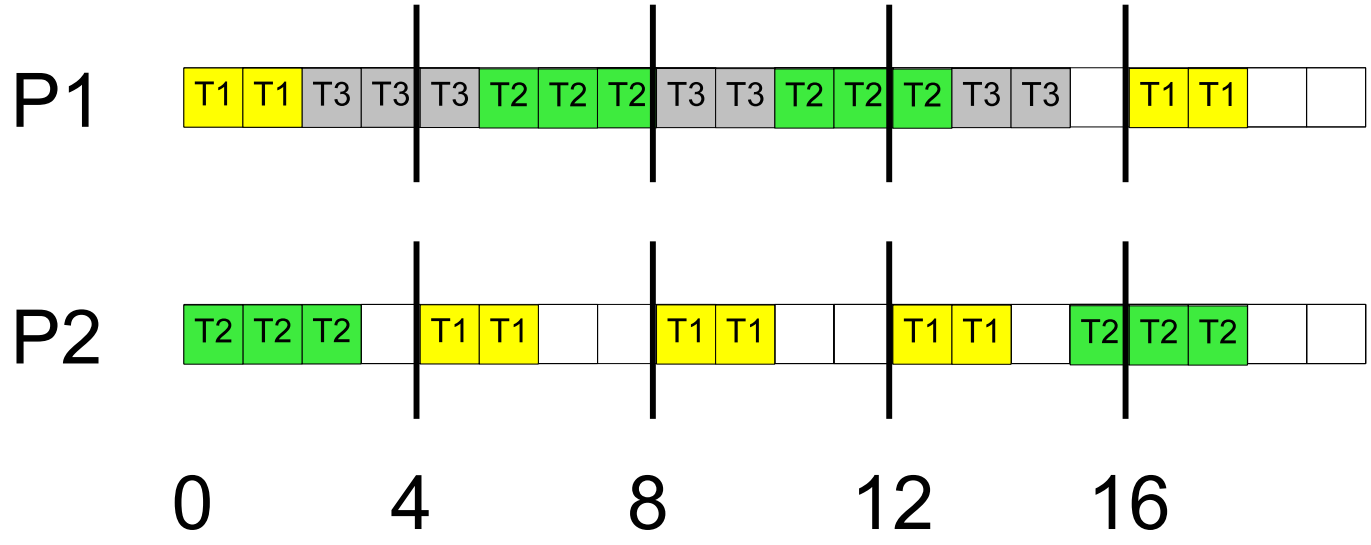
  1. Adapt mono-processor scheduling algorithms:
     - Global RM, global EDF, global DM, global LLF, ...
     - Choose the level of migration.
     - Apply the scheduling algorithm on all the set of the processors. At each time, assign m highest priority tasks to the m processors.
     - Preemptions occur when a job/task has to be run since it priority is high and when all processors are busy.
  2. New algorithms: PFair, LLREF, EDF(k), SA, EDZL, ...

- Be careful: multiprocessor scheduling is not a simple extension of monoprocessor scheduling $\Longrightarrow$ theoretical results are different and there are very few results.
- In the sequel, we assume identical processors.

Global scheduling (5)

- **Example:** global Deadline Monotonic

	Ci	Pi	Di
T1	2	4	4
T2	3	5	5
T3	7	20	20

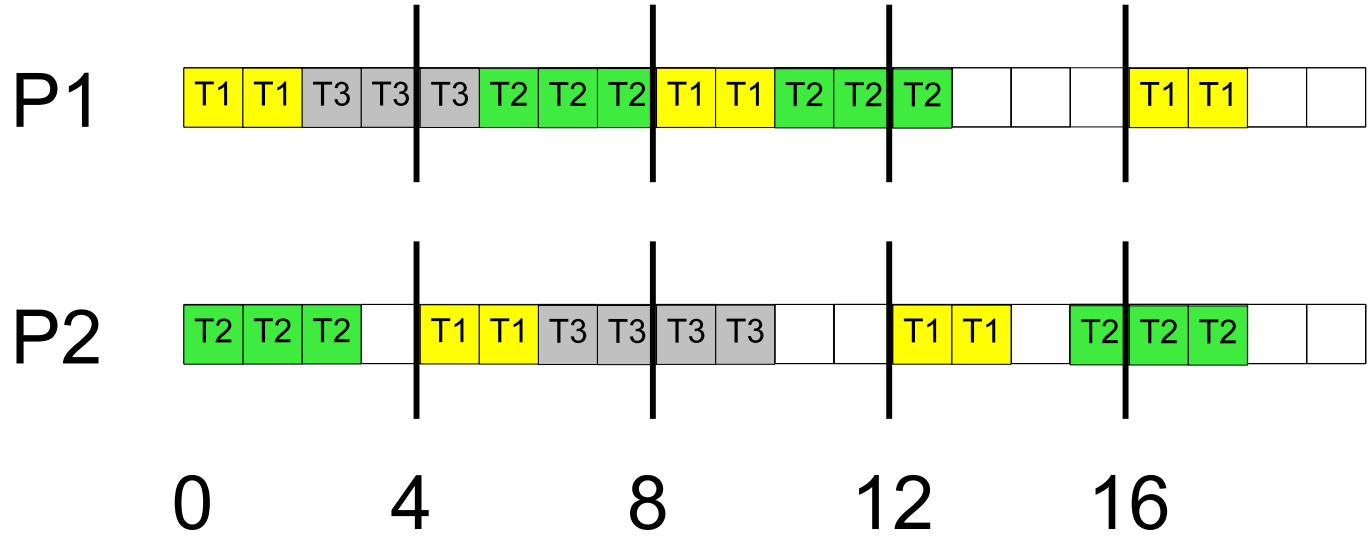


- Priority assignment: $T_1 > T_2 > T_3$.
- Job level migration.

Global scheduling (6)

- **Example:** global Deadline Monotonic

	Ci	Pi	Di
T1	2	4	4
T2	3	5	5
T3	7	20	20



- Priority assignment: $T_1 > T_2 > T_3$.
- Task level migration.

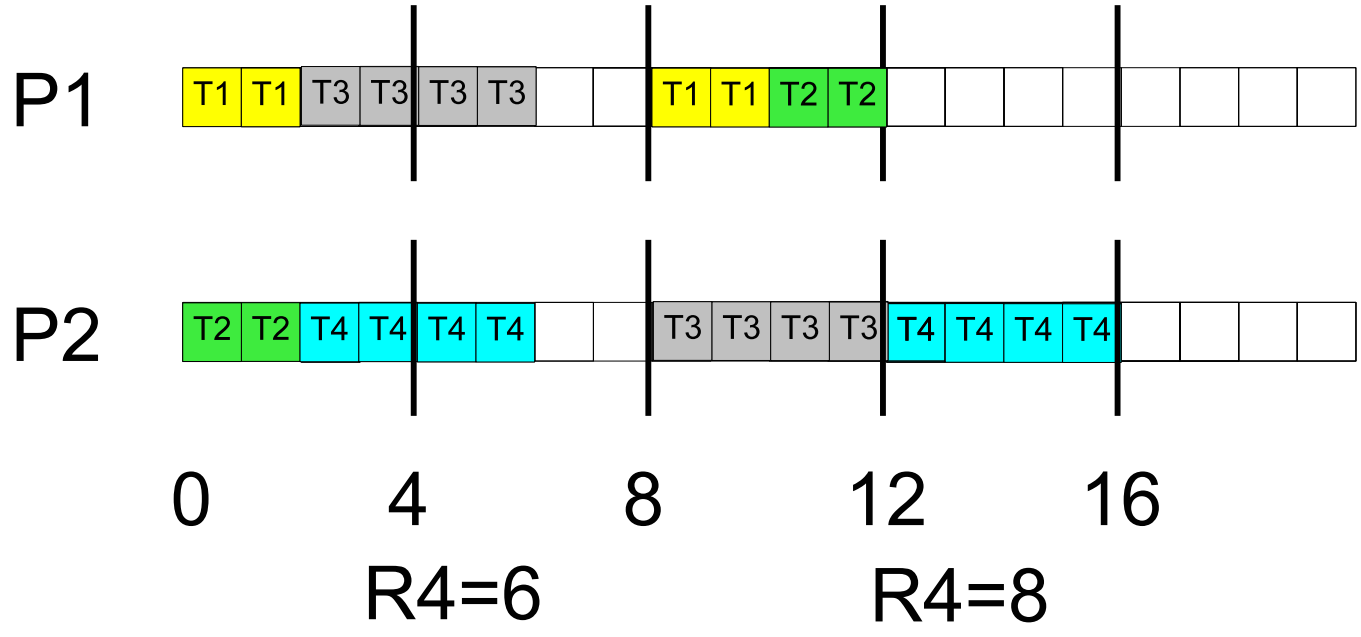
Global scheduling (7)

Task level migration	(1,3)	(2,3)	(3,3)
Job level migration	(1,2)	(2,2)	(3,2)
No migration	(1,1)	(2,1)	(3,1)
	1: task fixed priority	2: job fixed priority	3: dynamic priority at each unit of time

- **Comparing algorithms:**
  - Dynamic priority scheduling (3,3) are dominant. Optimality hierarchy is different from mono-processor hierarchy: global LLF is dominant comparing to global EDF.
  - (1,*) are incomparable each others.
  - (*,1) are incomparable with (*,2) and (*,3).
  - Pfair scheduler: identical processors + synchronous periodic tasks with deadline on request = optimal scheduler.

Global scheduling (8)

	Ci	Di	Pi
T1	2	2	8
T2	2	4	10
T3	4	6	8
T4	4	7	8



- **Critical instant:**
  - In the monoprocessor case with periodic tasks, critical instant is the worst case on processor demand and occurs in the beginning of the scheduling (busy period).
  - Assumption to compute worst case response times.
  - Is not true any more in the multiprocessor case.

Global scheduling (9)

- **Hyperperiod:**
  - In the monoprocessor case, the hyperperiod allows the verification:
    1. Of a periodic task set with any S_i.
    2. With any deterministic scheduler.

$$[0, lcm(\forall i : P_i) + 2 \cdot max(\forall i : S_i)]$$

  - In the multiprocessor case, only one known result:

$$[0, lcm(\forall i : P_i)]$$

    only for synchronous periodic task with deadline on request and preemptif fixed priority scheduling :-(

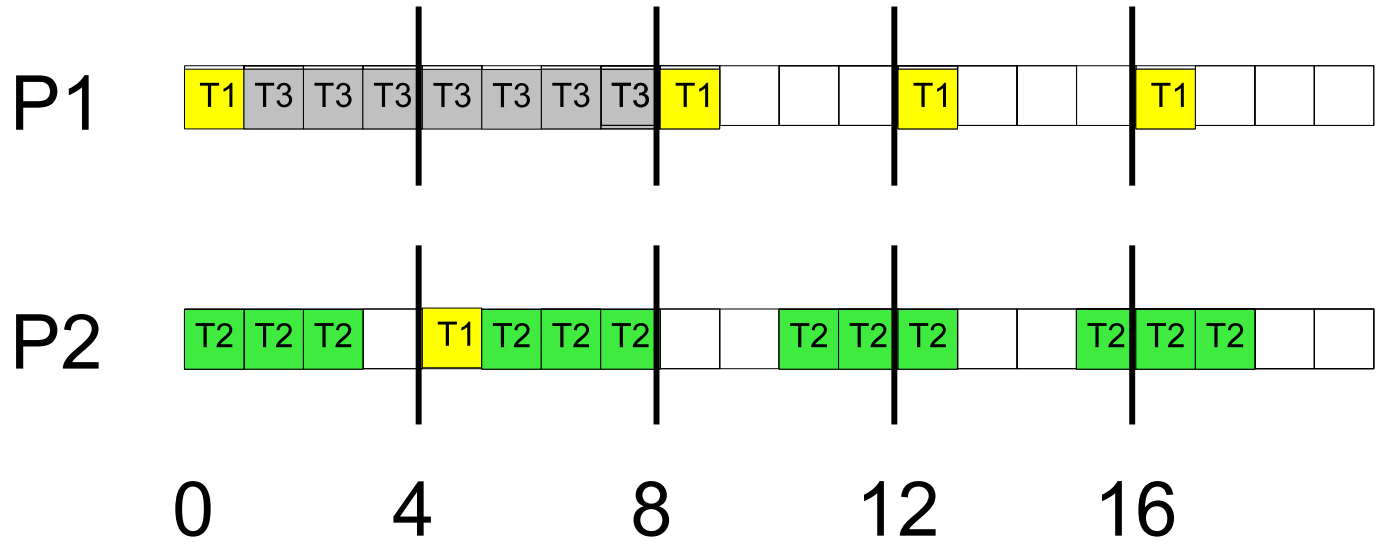
Global scheduling (10)

- **Scheduling errors:**
  - Errors: positive changes on a feasible task set lead to an unfeasible task set.
  - Scheduling verification is frequently made on worst cases. Example : a sporadic task set can be verified as a periodic task set.
  - Parameters that could be related to scheduling errors: C_i, $P_i \Longrightarrow$ decrease processor utilization factor.

Global scheduling (11)

- **Scheduling errors:**

	Ci	Pi	Di
T1	1	4	2
T2	3	5	3
T3	7	20	8

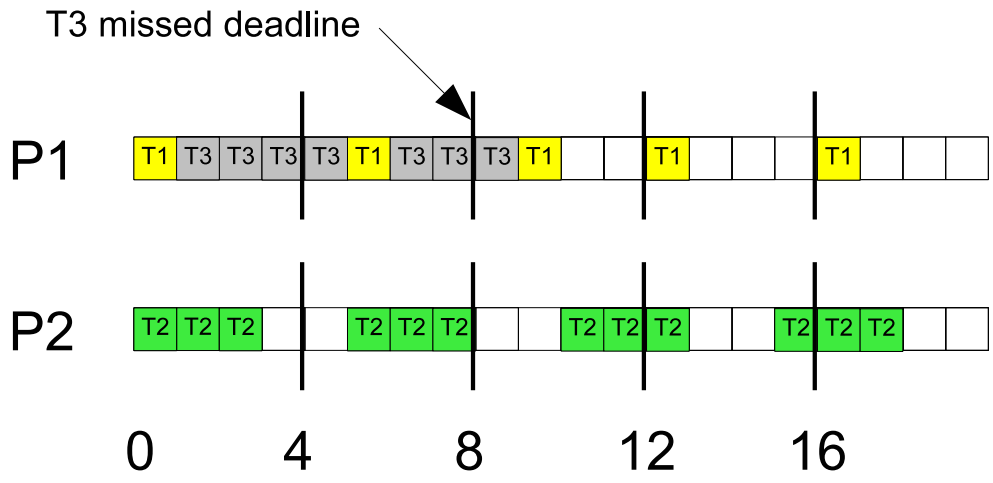


- Job migration level.
- Feasible task set.

Global scheduling (12)

- **Scheduling errors:**

	Ci	Pi	Di
T1	1	5	2
T2	3	5	3
T3	7	20	8



- Job level migration.
- Increasing P_1 ... leads to a task set which is not feasible any more.

Global scheduling (13)

- **Main ideas of the Pfair scheduling:**
  - We expect to find a schedule that leads to an equal processor utilization factor between all processors (called "Proportionate Fair" [AND 05]).
  - Optimal scheduler with identical processors and synchronous periodic tasks with deadline on request.
  - Deadline oriented.
  - Task level migrations.
  - Lead to many context switches.

Global scheduling (14)

- **Main ideas of the Pfair scheduling:**

1. We expect to run each task at a constant rate WT:

$$\forall i : WT(T_i, t) = t \cdot \frac{C_i}{P_i}$$

2. Which can be approximated by:

$$lateness(T_i, t) = WT(T_i, t) - \sum_{k=0}^{t-1} Sched(T_i, k)$$

where $Sched(T_i, t) = 1$ when T_i is scheduled in the time interval $[t, t+1[$ or $Sched(T_i, t) = 0$ otherwise.

3. A scheduling is said Pfair if and only if:

$$\forall i, T_i, t : -1 \leq lateness(T_i, t) \leq 1$$

4. Property: any Pfair scheduling is also feasible.

Global scheduling (15)

- **Main ideas of the Pfair scheduling:**
  - To reach proportionate scheduling, split task capacity in sub-tasks with a capacity of one unit of time : subtask are called quantum.
  - Assign a dynamic priority $d_{(T_i,j)}$ and a release time $r_{(T_i,j)}$ to each subtask:

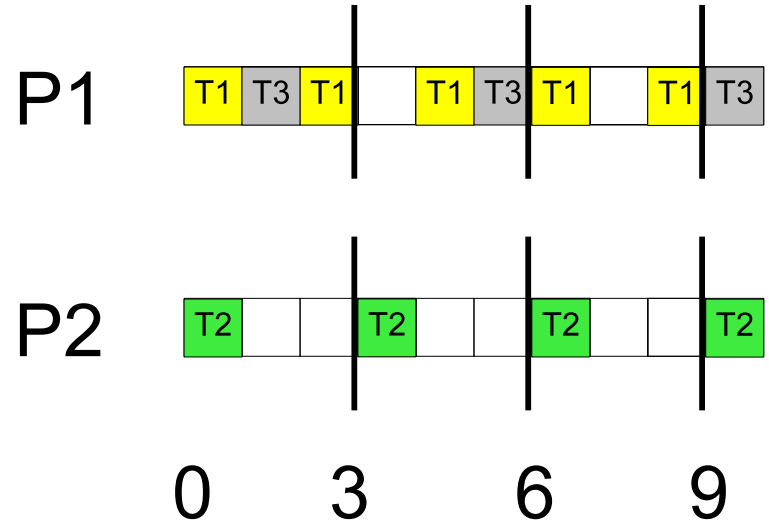
$$d_{(T_i,j)} = \left\lceil \frac{j}{C_i/P_i} \right\rceil$$

$$r_{(T_i,j)} = \left\lceil \frac{j-1}{C_i/P_i} \right\rceil$$

  - Schedule according to $d_{(T_i,j)}$: apply shortest deadline first.
  - Other Pfair schedulers: PF, PD, PD^2. Define how to manage subtasks with equal deadlines.

Global scheduling (16)

	Ci	Pi
T1	1	2
T2	1	3
T3	2	9



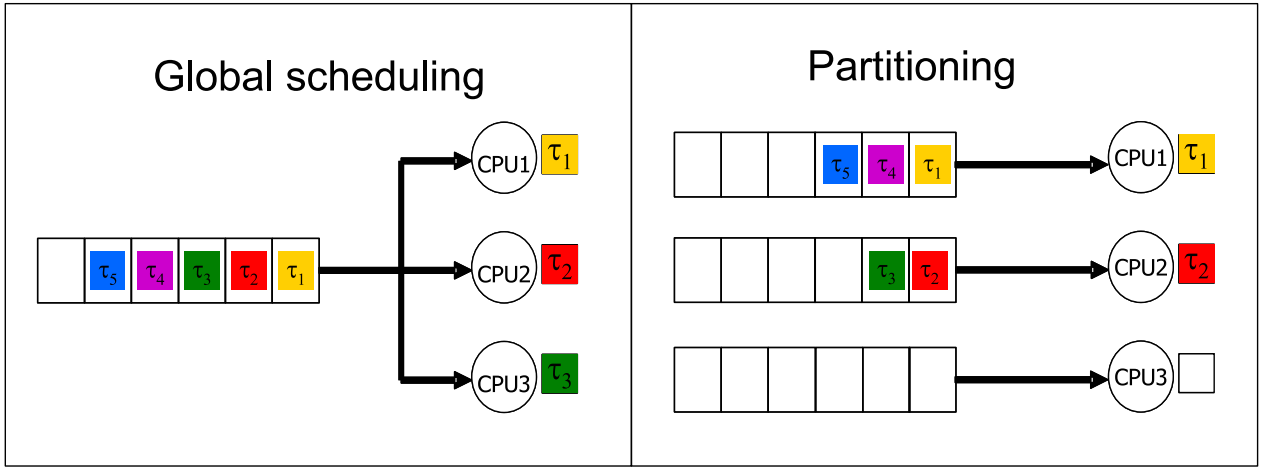
- $d_{(T_1,1)} = \left\lceil \frac{1}{1/2} \right\rceil = 2$
- $d_{(T_2,1)} = \left\lceil \frac{1}{1/3} \right\rceil = 3$
- $d_{(T_3,1)} = \left\lceil \frac{1}{2/9} \right\rceil = \lceil 4,5 \rceil = 5$
- $d_{(T_3,2)} = \left\lceil \frac{2}{2/9} \right\rceil = 9$
- $r_{(T_1,1)} = \left\lceil \frac{1-1}{1/2} \right\rceil = 0$
- $r_{(T_2,1)} = \left\lceil \frac{1-1}{1/3} \right\rceil = 0$
- $r_{(T_3,1)} = \left\lceil \frac{1-1}{2/9} \right\rceil = 0$
- $r_{(T_3,2)} = \left\lceil \frac{2-1}{2/9} \right\rceil = \lceil 4,5 \rceil = 5$

Global scheduling (17)

- **To conclude about global scheduling: global scheduling is different of monoprocessor scheduling:**
  - Different properties of schedulers: dominance/optimality, critical instant, hyperperiod.
  - New parameters: task migration/processor assignment, type of processors, ...

$\Longrightarrow$ Feasibility tests are similar ... but different and usually, their applicability are more restricted.
$\Longrightarrow$ And we do not take care of task dependencies. $\Longrightarrow$ And we only focus on identical processors.

Multiprocessor/distributed



1. **Global scheduling:** choose a task, and assign it to one of the idle processors (otherwise, preempt one of the running task). On-line processor assignment. With migration.
2. **Partitioning:** choose a processor for all tasks, and then run local scheduler on each processor. No migration. May apply end-to-end worst case response time analysis. Off-line processor assignment.

Partitioned systems (1)

- **How to statically assign a set of tasks to a set of processors?**
  - "bin-packing" problem: find how to put a set of object in identical boxes. We expect to minimize the required number of boxes.
  - NP-hard problem $\Longrightarrow$ heuristic, usually based on feasibility tests.
- **Many parameters:**
  - Processors (identical or not), tasks (priority, period, capacity).
  - Take into account task communications, shared resources,...
- **Examples of objective functions:**
  - Minimize the number of processors, the number of communications, latencies, ...

Partitioned systems (2)

- **Skeleton of a partitioning heuristic:**

1. Order tasks according to a given parameter (e.g. period, priority, processor utilization).
2. Do task assignment according to their order.
3. For each task, look for an available processor, according to a policy (e.g.: Best Fit, First Fit, Next Fit, ...).
4. An available processor is a processor that is able to run a task with no missed deadline: processors are selected according to a feasibility test.
5. Stop when all tasks are assigned.

Partitioned systems (3)

- **Examples of partitioning heuristics:**
  - Mostly based on RM [OH 93]: feasibility tests are easy to apply and RM is compliant with a static task assignment approach.
  - Basic RM oriented heuristics: Rate-Monotonic First-Fit, Next-Fit or Best-Fit [DHA 78, OH 93].
  - Other examples: RM-FFDU[OH 95], RM-ST, RM-GT and RM-GT/M [BUR 94], RM-Matching[DAV 09], EDF-FFD (First Fit Decreasing) [BAR 03], EDF-FF/EDF-BF, FFDUF[DAV 09].

Partitioned systems (4)

- **Example of the Rate Monotonic Next Fit** :

1. Tasks are increasingly ordered by their periods.
2. We start with task $i = 0$ and processor $j = 0$.
3. Assign task i on processor j if the feasibility tests is true (e.g. $U \leq 0.69$).
4. Otherwise, put task i on processor $j + 1$.
5. Assign the next task $i + 1$.
6. Stop when all tasks have been assigned. j = the number of required processors.

Partitioned systems (5)

- Trouble with Next Fit = sometimes leads to a low processor utilization factor. Others heuristics to increase it:

  - First Fit : for each task i, start with processor $j = 0$ and assign task i on the first processor on which the feasibility test is true.
  - Best Fit : for each task i, start with processor $j = 0$ and assign task i on the processor on which the feasibility test is true and on which the processor utilization factor has the highest value.
  - ...

Partitioned systems (6)

- Let go back to the overall partitioning approach:

1. Choose processor location for each task (partitioning).
2. Model precedences constraints. e.g. jitter approach.
3. Compute message scheduling. Compute worst case communication time for periodic message. Depend on MAC protocols => See T.U. on real-time network.
4. Check end to end timing constraint.

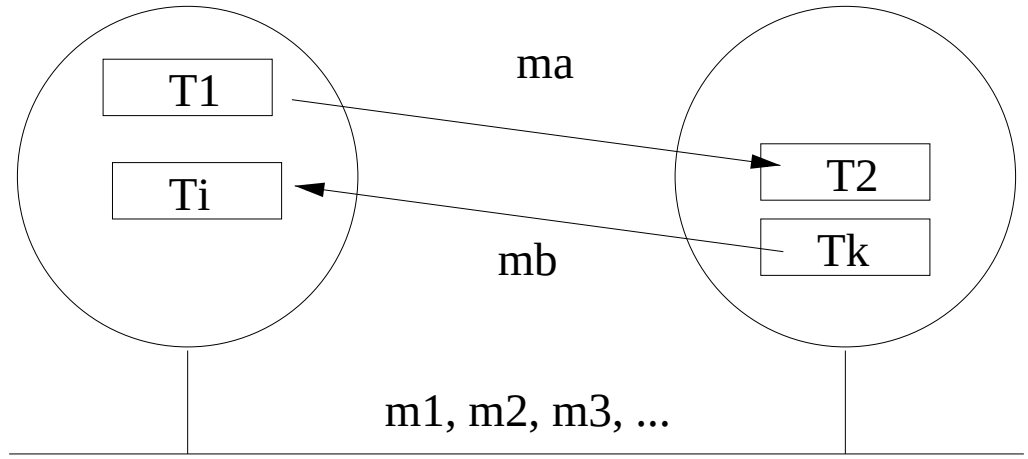
End to end timing constraint (1)

- **ESP : Electronic stability program:**
  - Sensors : Steering wheel angle sensor, Yaw rate sensor, Wheel speed sensor.
  - Actuators : brakes of the wheels, engine.
  - How to check that a given deadline is missed (or not) between sensors and actuators?

End to end timing constraint (2)

- **Several approaches:**
  - Approach based on Chetto and Blazewicz.
  - Approach based on worst case response time with offsets [BAT 97, PAL 03].
  - Approach based on worst case response time with jitters: also called holistic approach
    [TIN 94, LEB 95, RIC 01, CHA 95].

End to end timing constraint (3)



- **Constraint to verify :** $r_{(T1+ma+T2)} \leq D$
- **Holistic approach:** assign jitter of a task with the worst case response time of its predecessors[TIN 94]:

  - Assuming r_{T1}, the worst case response time of T1.
  - Assuming r_{ma} computed according to the real-time network MAC protocol and with $J_{ma} = r_{T1}$.
  - r_{T2} is computed with $J_2 = r_{ma}$.
  - **Compute until convergence.**

End to end timing constraint (4)

```
10    ∀i : J_i := 0, r_i := 0, r'_i := 0;
20    ∀i : Compute_worst_case_response_time (r_i);
30    While (∃i : r_i ≠ r'_i) {
40        ∀i : J_i := max(J_i, ∀j with j ≺ i : r_j);
50        ∀i : r'_i := r_i;
60        ∀i : Compute_worst_case_response_time(r_i);
70    }
```

If i is a periodic task:

$$r_i = J_i + w_i \quad with \quad w_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i + J_j}{P_j} \right\rceil C_j$$

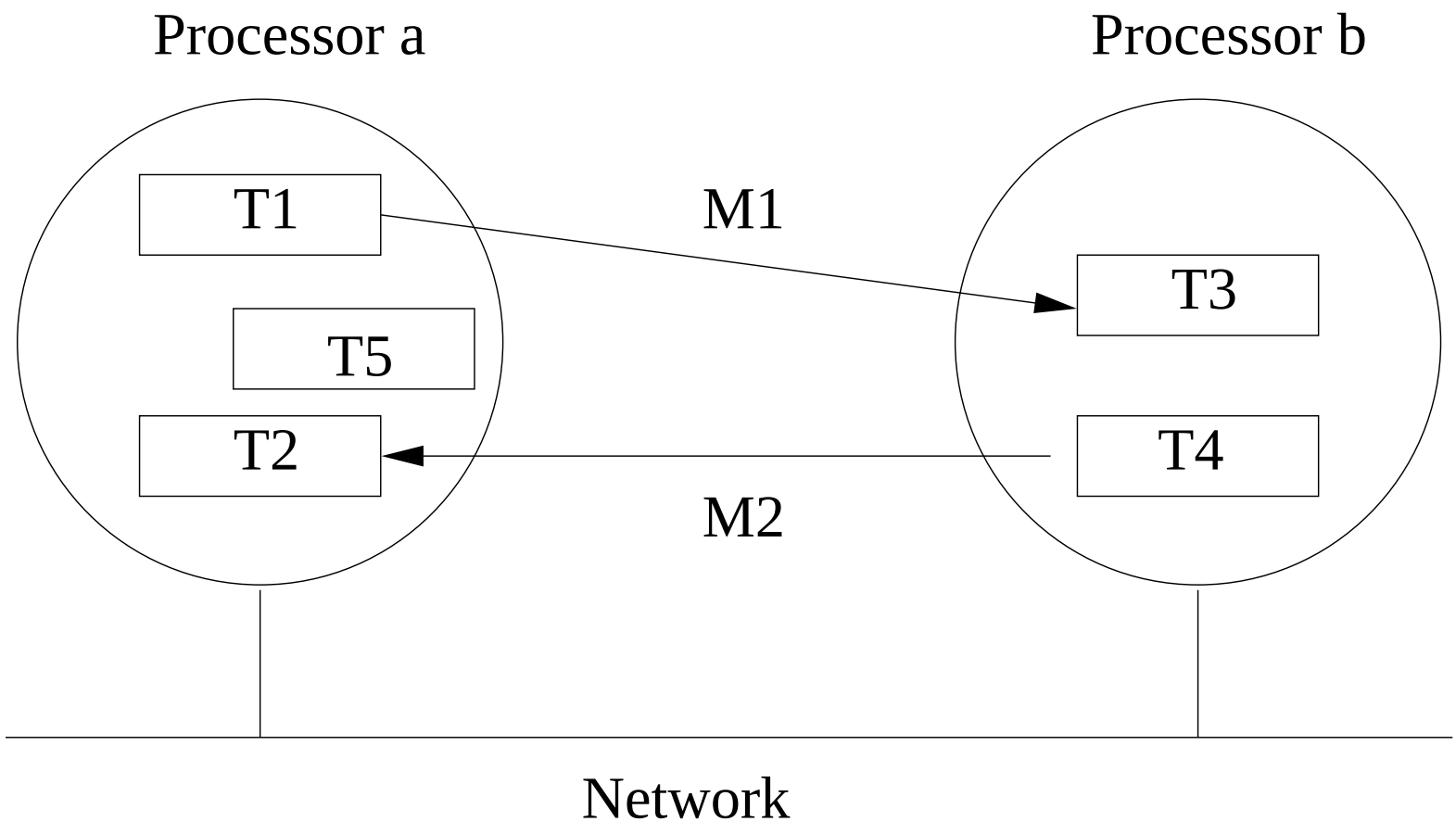
If i is a periodic message:

$$r_i = J_i + M_i$$

with M_i, the communication delay of the message.

End to end timing constraint (5)

- **Example (task partitioning and exchanged messages):**



End to end timing constraint (6)

- **Example (task and message parameters):**

Task	Period	Capacity	Priority	Processor
T_1	100	4	1	a
T_2	60	5	2	a
T_3	100	3	2	b
T_4	60	2	1	b
T_5	90	3	3	a

Message	Period	Communication delay
M_1	100	6
M_2	60	1

End to end timing constraint (7)

- **Lines 10-20 :** $\forall i : J_i = 0$

Message/Task	M_1	M_2	T_1	T_2	T_3	T_4	T_5
J_i	0	0	0	0	0	0	0
r_i	6	1	4	9	5	2	12

- **First iteration, lines 40-60 :** update jitter + compute worst case response time. $J_{M_1} = r_{T_1}$, $J_{M_2} = r_{T_4}$, $J_{T_3} = r_{M_1}$ et $J_{T_2} = r_{M_2}$.

Message/Task	M_1	M_2	T_1	T_2	T_3	T_4	T_5
J_i	**4**	**2**	0	**1**	**6**	0	0
r_i	**6+4 =10**	**1+2 =3**	4	**9+1 =10**	**5+6 =11**	2	12

End to end timing constraint (8)

- **Second iteration, lines 40-60 :** update jitter + compute worst case response time. $J_{M_1} = r_{T_1}$, $J_{M_2} = r_{T_4}$, $J_{T_3} = r_{M_1}$ et $J_{T_2} = r_{M_2}$.

Message/Task	M_1	M_2	T_1	T_2	T_3	T_4	T_5
J_i	**4**	**2**	0	**3**	**10**	0	0
r_i	**6+4 =10**	**1+2 =3**	4	**9+3 =12**	**5+10 =15**	2	12

- **Third iteration, lines 40-60 :** update jitter $J_{M_1} = r_{T_1}$, $J_{M_2} = r_{T_4}$, $J_{T_3} = r_{M_1}$ et $J_{T_2} = r_{M_2}$. Jitters are equal => same worst case response times => convergence => stop calculation.

Message/Task	M_1	M_2	T_1	T_2	T_3	T_4	T_5
J_i	**4**	**2**	0	**3**	**10**	0	0

Summary

1. Introduction.
2. Classical real-time schedulers.
3. Shared resources and precedence constraints.
4. Schedulability tools and architecture languages.
5. Real-time scheduling for multiprocessor and distributed systems.
6. Conclusion.
7. References.

Conclusion

1. Classical schedulers (Rate Monotonic, Earliest Deadline First) and task models (periodic task).
2. Feasibility tests and early verification: at design step, on architecture models. Can be made automatically (tools + architecture design languages).
3. Most of Real-Time operating systems provide fixed priority scheduling.
4. Shared resources and inheritance protocols. Precedence constraints.
5. Mono-processor real-time scheduling is well mastered, but not multiprocessor and distributed scheduling.

Summary

1. Introduction.
2. Classical real-time schedulers.
3. Shared resources and precedence constraints.
4. Schedulability tools and architecture languages.
5. Real-time scheduling for multiprocessor and distributed systems.
6. Conclusion.
7. References.

References (1)

[AND 05] James Anderson, Philip Holman, and Anand Srinivasan. « Fair Scheduling of Real-time Tasks on Multiprocessors ». In *Handbook of Scheduling*, 2005.

[BAR 03] S. K. Baruah and S. Funk. « Task assignment in heterogeneous multiprocessor platforms ». Technical Report, University of North Carolina, 2003.

[BAT 97] I. Bate and A. Burns. « Schedulability Analysis of Fixed priority Real-Time Systems with Offsets », 1997.

[BER 07] M. Bertogna. « Real-Time scheduling analysis for multiprocessor platforms ». In *Phd Thesis, Scuola Seprropre Sant Anna, Pisa*, 2007.

[BLA 76] J. Blazewicz. « Scheduling Dependant Tasks with Different Arrival Times to Meet Deadlines ». In. Gelende. H. Beilner (eds), Modeling and Performance Evaluation of Computer Systems, Amsterdam, Noth-Holland, 1976.

[BUR 94] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son. « Assigning real-time tasks to homogeneous multiprocessor systems ». January 1994.

References (2)

[BUT 03] G. Buttazzo. « Rate monotonic vs. EDF: Judgment day ». *n Proc. 3rd ACM International Conference on Embedded Software, Philadephia, USA*, October 2003.

[CHA 95] S. Chatterjee and J. Strosnider. « Distributed pipeline scheduling : end-to-end Analysis of heterogeneous, multi-resource real-time systems ». 1995.

[CHE 90] H. Chetto, M. Silly, and T. Bouchentouf. « Dynamic Scheduling of Real-time Tasks Under Precedence Constraints ». *Real Time Systems, The International Journal of Time-Critical Computing Systems*, 2(3):181–194, September 1990.

[COU 94] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems—Concepts and Design, 2nd Ed.* Addison-Wesley Publishers Ltd., 1994.

[DAV 09] R.I. Davis and A. Burns. « A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems ». In *University of York Technical Report, YCS-2009-443*, 2009.

[DHA 78] S. Dhall and C. Liu. « On a real time scheduling problem ». *Operations Research*, 26:127–140, 1978.

References (3)

[LEB 95] L. Leboucher and J. B. Stefani. « Admission Control end-to-end Distributed Bindings ». pages 192–208. COST 231, Lectures Notes in Computer Science, Vol 1052, November 1995.

[LIU 73] C. L. Liu and J. W. Layland. « Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment ». *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[OH 93] Y. Oh and S. H. Son. « Tight Performance Bounds of Heuristics for a Real-Time Scheduling P roblem ». Technical Report, May 1993.

[OH 95] Y. Oh and S.H. Son. « Fixed-priority scheduling of periodic tasks on multiprocessor systems ». 1995.

[PAL 03] C. Palencia and M. Gonzalez Harbour. « Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF ». In *15th Euromicro Conference on Real-Time Systems (ECRTS'03)*, 2003.

[RIC 01] P. Richard, F. Cottet, and M. Richard. « On line Scheduling of Real Time Distributed Computers With Complex Communication Constraints ». 7th Int. Conf. on Engineering of Complex Computer Systems, Skovde (Sweden), June 2001.

References (4)

[SHA 90] L. Sha, R. Rajkumar, and J.P. Lehoczky. « Priority Inheritance Protocols : An Approach to real-time Synchronization ». *IEEE Transactions on computers*, 39(9):1175–1185, 1990.

[SIN 04] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. « Cheddar : a Flexible Real Time Scheduling Framework ». Proceedings of the International ACM SIGAda Conference, Atlanta, USA, November 2004.

[STA 88] John Stankovic. « Misconceptions about real-time computing ». *IEEE Computer*, October 1988.

[TIN 94] K. W. Tindell and J. Clark. « Holistic schedulability analysis for distributed hard real-time systems ». *Microprocessing and Microprogramming*, 40(2-3):117–134, April 1994.

[XU 90] J. Xu and D. Parnas. « Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusion Relations ». *IEEE Transactions on Software Engineering*, 16(3):360–369, March 1990.

Acknowledgements

- Some slides on multiprocessors scheduling are taken from lectures of C. Pagetti and J. Goossens. Thanks to these authors!