



Real-Time Scheduling for Mono-Processors Systems

Laurent Pautet
strec.wp.mines-telecom.fr
Laurent.Pautet@enst.fr

Version 3.1

Real-Time Embedded Systems

Examples

Airplane



Cellphone



Automated Subway



Nuclear Plant



?

What do they have in common ?



Embedded Systems

Definition and Requirements

An embedded system is a special-purpose system which software, hardware, mechanical, ... components are encapsulated in the device it controls

As opposed to general-purpose systems, they have specific properties such as low consumption, small size and weight, limited resources ...

A cruise control, a washing machine, factory robot, ...



Real-Time Systems

Definition and Requirements

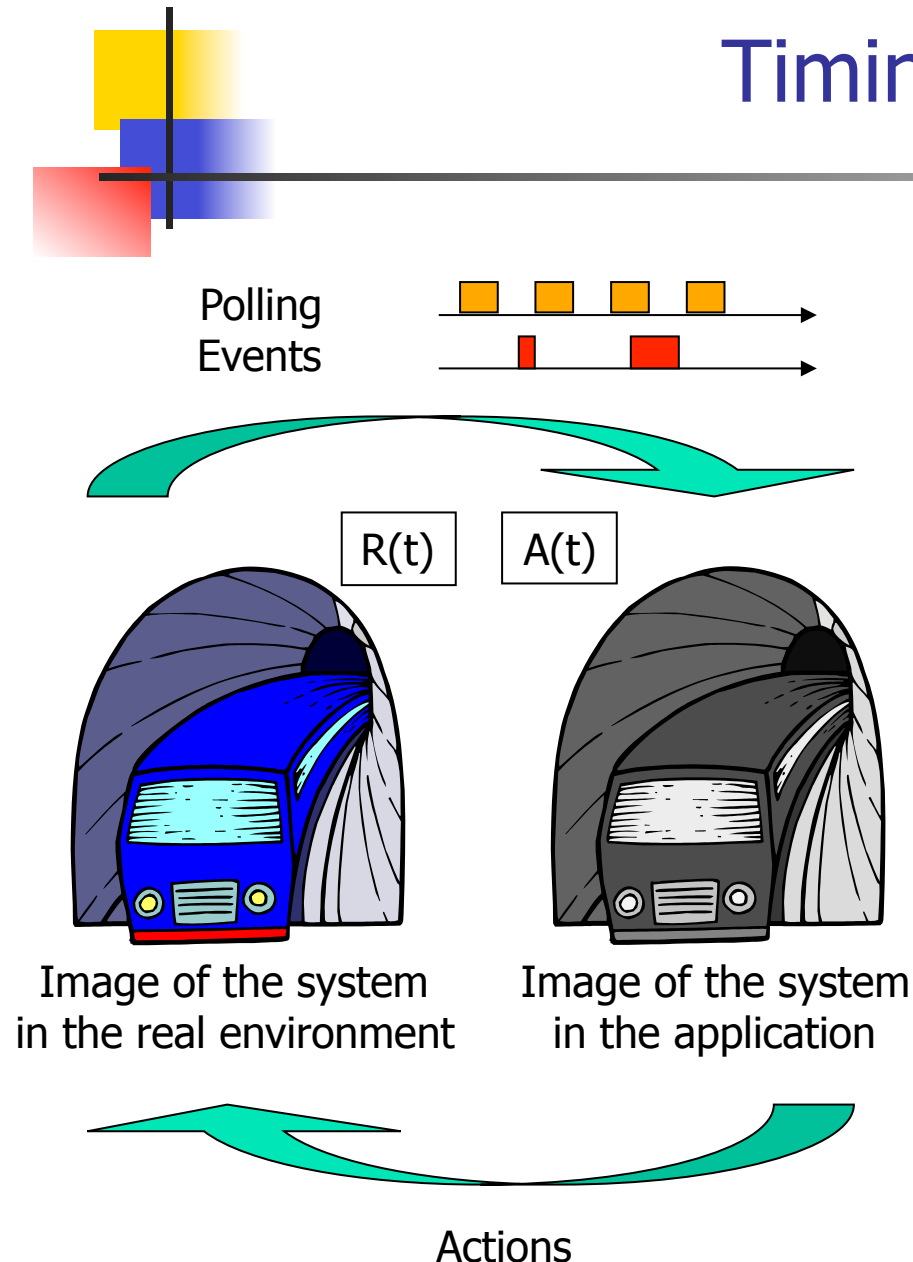
A real-time system consists in one or more sub-systems that have to react under specified time requirements to stimuli produced by the environment

A response after a deadline is invalid
Even if the response is logically correct

A cruise control, a washing machine, factory robot,
a nuclear plant, an air traffic control, trading center, ...

Most real-time systems are embedded systems

Timing constraints



- The application must have a precise and consistent image of the system in its environment at anytime
- The goal of real-time systems is to minimize the difference between the images of the system in reality and in its application ($|R(t) - A(t)| < \epsilon$)
- To update the image in the application, it reads in particular sensors periodically. The period being a temporal granularity during which the measures evolve significantly)



Non Fonctional Properties

These systems have to be predictable

- **Reactivity and temporal consistency**
 - Define temporal interval during which data is valid
 - Define time granularity (ship sec, rail msec, airplane usec)
 - Guarantee response time boundaries (known in advance)

- **Reliability and Availability**
 - Guarantee the correctness of the computed data values
 - Enforce system availability in presence of hostile conditions (fault tolerance, malicious behavior ...)

Non-Fonctional Properties -> temporal & structural



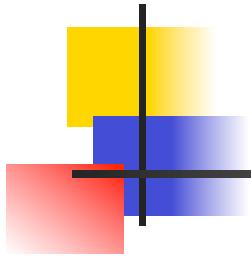
From requirements to technical solutions

- Requirements
 - Reactivity & temporal consistency
 - Reliability & Availability
- Solutions
 - Architectures and frameworks to help the design (Kernels, RT buses and networks, ...)
 - Models and methods to enforce predictability (RT scheduling, Fault tolerance, ...)
 - Suitable programming languages (C-Misra, Java-RT, Standard POSIX 1003.1c, Ada, ...)
 - Tools to integrate modeling, analysis and synthesis (AADL, Marte, verification, simulation, generation, testing)

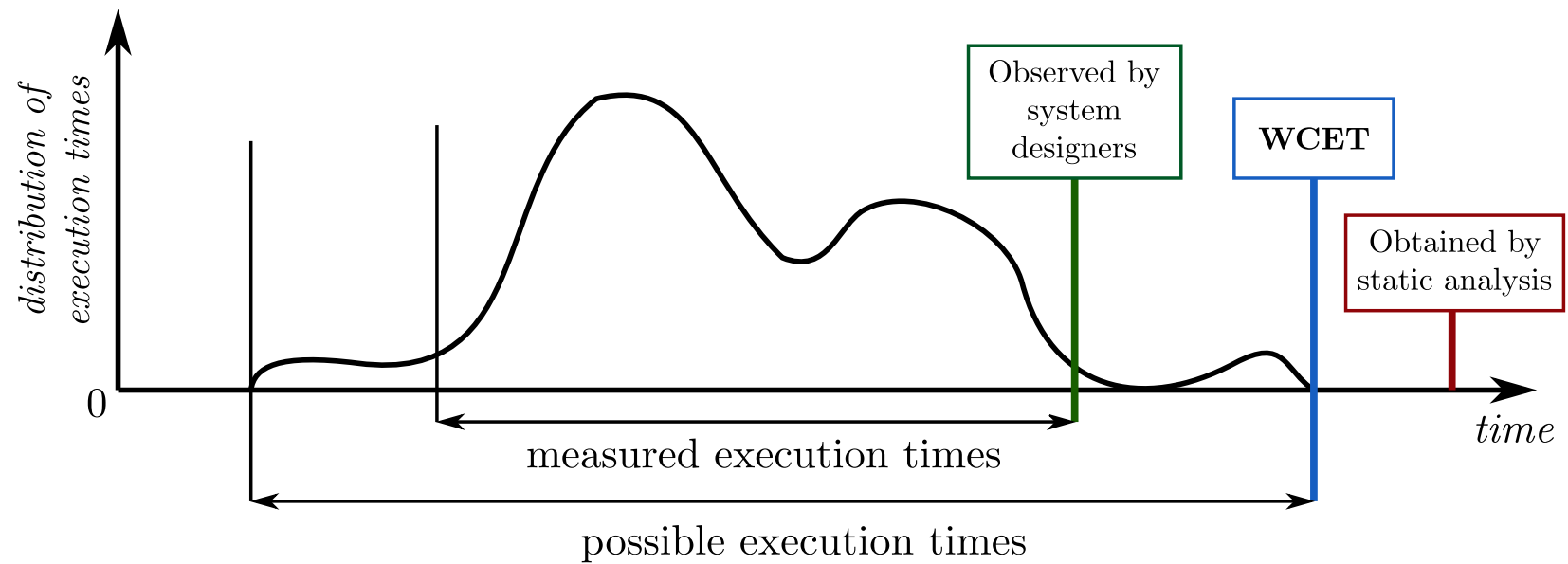


Notations

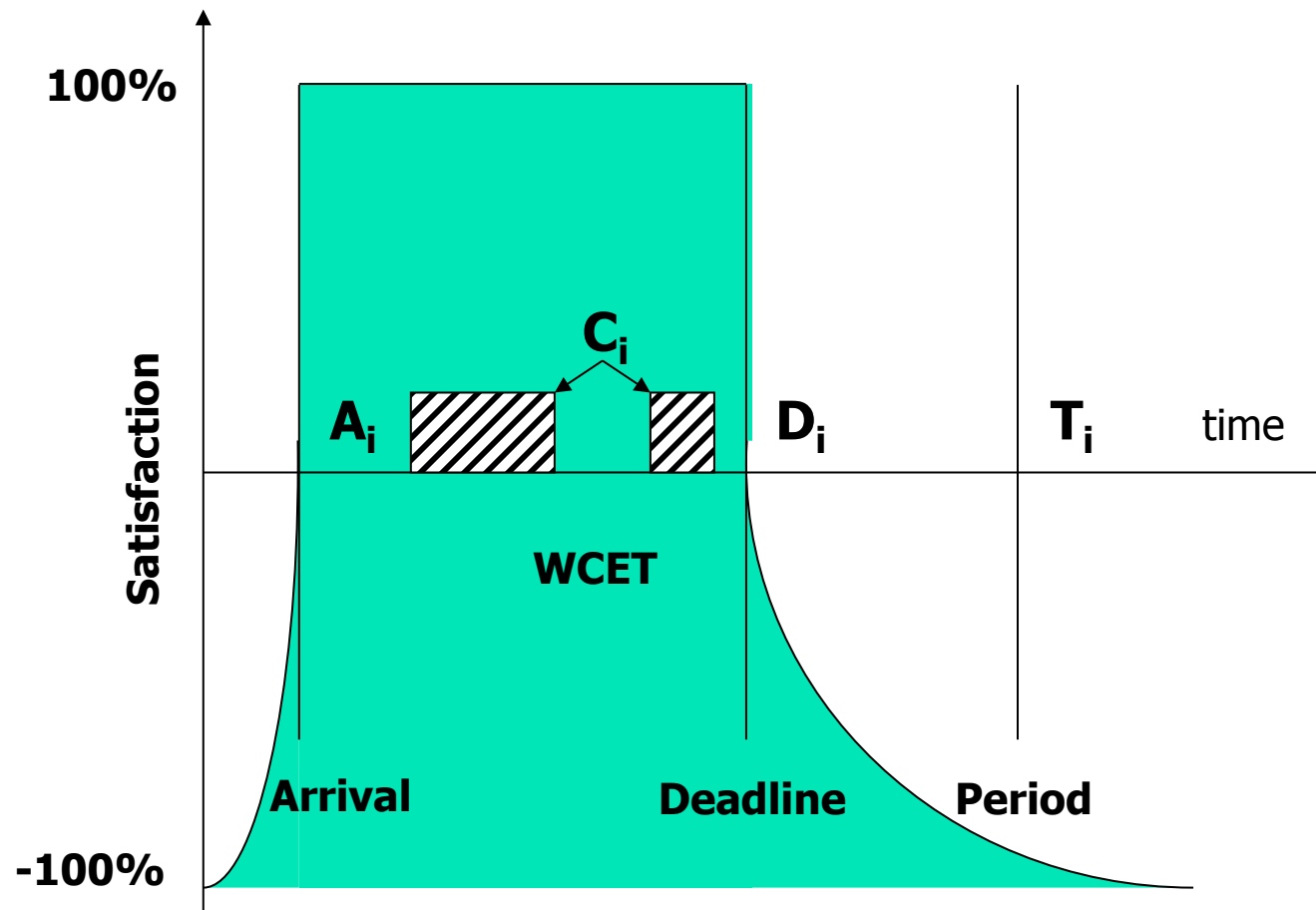
- Parameters of task t_i
 - C_i : Worst Case Execution Time (WCET) of task t_i
 - A_i : Arrival time of task t_i
 - Task must not arrive before A_i
 - A_i may be different from 0 (dependency)
 - T_i : Period of task t_i
 - D_i : Deadline of task t_i
 - Task must not complete after D_i
 - $A_i + C_i < D_i$ however ...
 - $D_i \leq T_i$ is not mandatory (constrained deadline)
 - $U_i = C_i / T_i$ = processor utilisation of task t_i
- Operators
 - Ceiling $\lceil x \rceil$ (least integer greater than or equal to x)
 - Floor $\lfloor x \rfloor$ (greatest integer less than or equal to x)



WCET evaluation

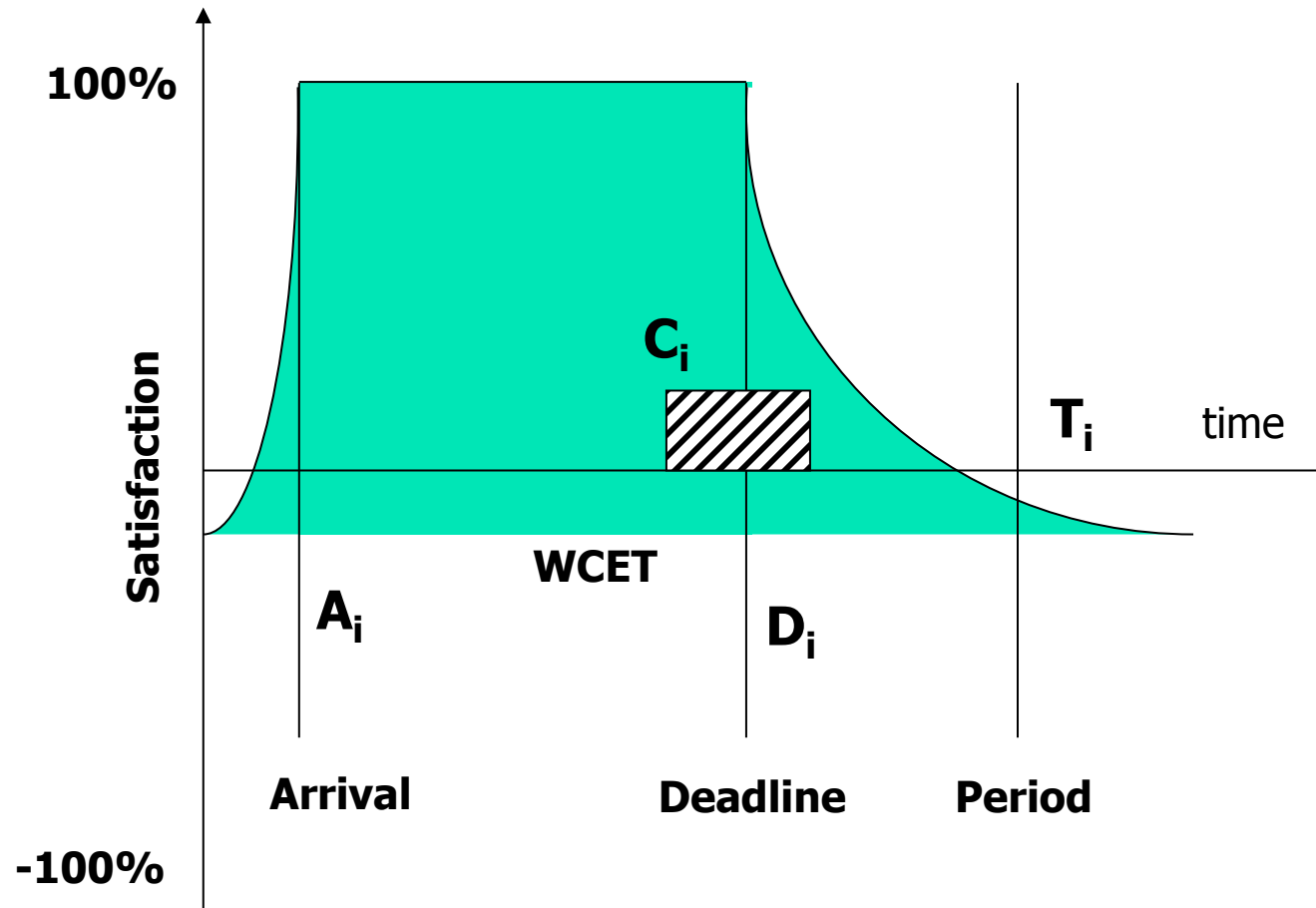


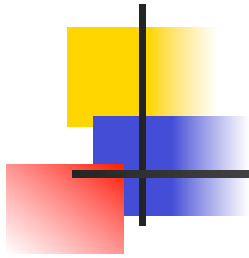
Hard Real-Time Task



Soft Real-Time Task

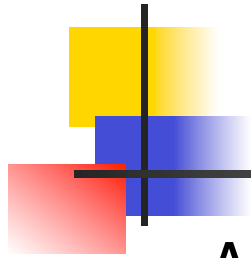
(Different from Best Effort Task that has No Deadline)





Missing deadlines

- For a hard real-time task, deadlines must be fulfilled
 - Enforce a maximal determinism
 - WCET : *Worst Case Execution Time*
 - Reduce non-deterministic behavior
 - Pre-allocated resources
 - System over-dimensioning
- For a soft real-time task, missing deadlines can be tolerated under some circumstances
 - For a given percentage of times
 - For a given number of times
 - For a given frequency
 - And result in a degraded execution mode



Sub-systems of real-time systems

A real-time system is composed of several sub-systems with different real-time properties

Some of these sub-systems may be non real-time, soft real-time or hard real-time sub-systems

- Hard real-time tasks must fulfill their deadlines.
- Soft real-time tasks may fail to fulfill their deadlines. If so, they may execute in a degraded mode.
- Other tasks execute in best-effort mode.



Real-Time Schedulers

- Allocate (temporal) resources to guarantee safety properties
- In normal mode, respect the time constraints of all tasks
- Otherwise, limit the effects of time overflows and ensure compliance with the constraints of the most critical tasks

In the following, it will be ensured that the time required for the implementation of the scheduling algorithm and that of context change are negligible which implies a low complexity and effective implementation



Definitions

Execution Model

- Dependent or independent tasks
 - Independent tasks sharing only the processor
 - Dependent tasks with shared resources or linked by precedence constraints
- Synchronous task means task of zero activation time
- Periodic task with implicit deadline means periodic task with deadline equals to period
- Task job : instantiation of a task during period
- Worst Case Execution Time : worst computation time
- Response time : time to complete a job while other jobs are also running on the same processor



Definitions

- Preemptive and non-preemptive scheduling
 - A preemptive scheduler can interrupt a task for a higher priority task when a non-preemptive scheduler executes the task until it completes
- Offline or online scheduling
 - A scheduler decides offline or online when and which task to execute
- Optimal scheduling
 - Algorithm that produces a schedule for any set of schedulable tasks (if an algorithm does, it does too)
- Scheduling test
 - A necessary and / or sufficient condition for an algorithm to satisfy the temporal constraints of a set of tasks



Overview of algorithms

- Scheduling periodic tasks
 - Non-preemptive table-based scheduling
 - Preemptive scheduling with static priorities
 - Rate and Deadline Monotonic Scheduling
 - Preemptive scheduling with dynamic priorities
 - Earliest Deadline First and Least Laxity First
- Scheduling aperiodic tasks
 - Background, polling, deferred & sporadic servers
- Sharing resources
 - Priority Inheritance, Priority Ceiling & Highest Locker



Proving schedulability using a scheduling algorithm

- A set of synchronous periodic tasks repeats itself after an hyper-period, least common multiple of all task periods
 - Feasibility interval in a more general case: independent a/synchronous periodic tasks, $\forall i: D_i \leq T_i$ with a fixed priority scheduling $[0, 2 * \text{LCM}(\forall i: T_i) + \max(\forall i: A_i)]$
- To prove schedulability of a task set
 - Execute the algorithm over an hyper-period
 - Compute a (necessary - sufficient) scheduling test
 - Compute response time & check against deadlines



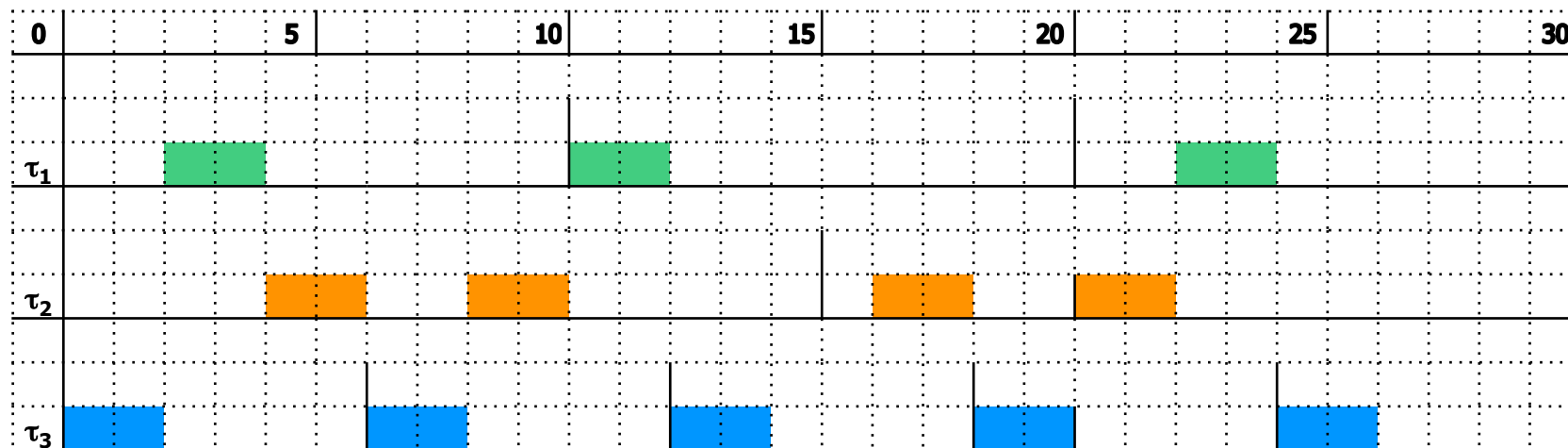
Table Driven Scheduling

Principles

- Hypotheses
 - Periodic tasks
- Principles
 - Major cycle = LCM of the task periods
 - Minor cycle = non-preemptible block
 - The minor cycle divides the major cycle
 - A cyclic scheduler loops on the major cycle by executing the sequence of minor cycles
 - The minor cycle provides a control point to check the respect of the timing constraints

Table Driven Scheduling Example

	Period	Deadline	WCET	Usage
τ_1	10	10	2	0,200
τ_2	15	15	4	0.267
τ_3	6	6	2	0.333



Minor Cycle

Major Cycle



Table Driven Scheduling

Advantages and Disadvantages

- Advantages

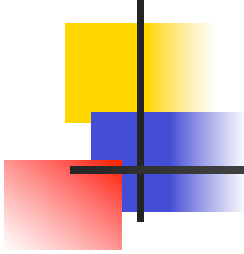
- Effective implementation
- No need for mutual exclusion between tasks

- Disadvantages

- Not work conserving :
 - the processor may be idle while jobs are not completed
- Impact of an additional task
- Execution of aperiodic tasks
- Difficult construction of the table
 - Allocating slots is a complex problem since it has to take into account time constraints, shared resources & aperiodic tasks.

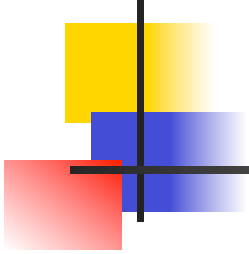
Static Priority Scheduling

Highest Priority First

- 
- Each task is assigned a priority (integer number) before runtime
 - The scheduler always executes the task of the ready tasks list with the highest priority
 - The scheduler can preempt the current task to execute a new task that has just been activated
 - There are many algorithms to assign priorities to tasks (mostly based on their temporal parameters)
 - The objective is to find a mapping that makes the task set schedulable

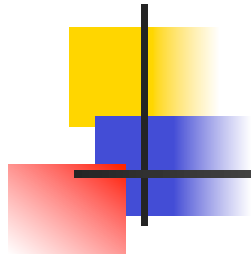
Static Priority Scheduling

Response Time

- 
- The *critical instant* for a set of synchronous periodic tasks is when all jobs start at the same time
 - For each task, compute time t at which its first activation completes by integrating the execution of highest priority tasks activated in the mean time
 - Start with a first response time $R_0^i = C_i$
 - Compute $R_{n+1}^i = \sum_{j \leq i} C_j * \lceil R_n^i / T_j \rceil$ to integrate the execution of the tasks of highest priority
 - Reiterate until a fixed point is reached
 - The task is schedulable if the response time is a fixed value less than or equal to the deadline
 - Valid for any static priority scheduling

Static Priority Scheduling

Response Time (RMS)



	T	C	P
τ_1	3	1	3
τ_2	5	2	2
τ_3	15	4	1

1. Check for τ_1
 1. $R_0 = C_1 = 1$
 2. $R_1 = \text{Response}(R_0) = 1 * 1 = 1$



2. Check for τ_2 and take into account τ_1

1. $R_0 = C_2 = 2$



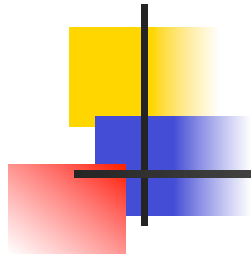
2. $R_1 = \text{Response}(R_0) = 1 * 1 + 1 * 2 = 3$

3. $R_2 = \text{Response}(R_1) = 1 * 1 + 1 * 2 = 3$



Static Priority Scheduling

Response Time (RMS)



	T	C	P
τ_1	3	1	3
τ_2	5	2	2
τ_3	15	4	1

1. Check for τ_3 and take into account τ_1 and τ_2

1. $R_0 = C_3 = 4$



2. $R_1 = \text{Response}(R_0) = 2*1 + 1*2 + 1*4 = 8$



3. $R_2 = \text{Response}(R_1) = 3*1 + 2*2 + 1*4 = 11$



4. $R_3 = \text{Response}(R_2) = 4*1 + 3*2 + 1*4 = 14$



5. $R_4 = \text{Response}(R_3) = 5*1 + 3*2 + 1*4 = 15$

6. $R_5 = \text{Response}(R_4) = 5*1 + 3*2 + 1*4 = 15$





Static Priority Scheduling

OPA – Optimal Priority Assignment

- Let have N fixed priority tasks
- Among these tasks, find a task that can have the lowest priority ...
 - Its response time should be less than its deadline when all the others have a higher priority
 - If there is such a task, give it the lowest priority
 - Otherwise, the system is not schedulable
- Repeat with the $N-1$ remaining tasks



Static Priority Scheduling

Rate Monotonic Scheduling

■ Hypotheses

- Synchronous, deadline implicit & independent tasks
- Synchronous ($A_i = 0$)
- Deadline implicit ($D_i = T_i$)

■ Principle

- Task activation or completion wake up the scheduler
- Select the ready task with the shortest period

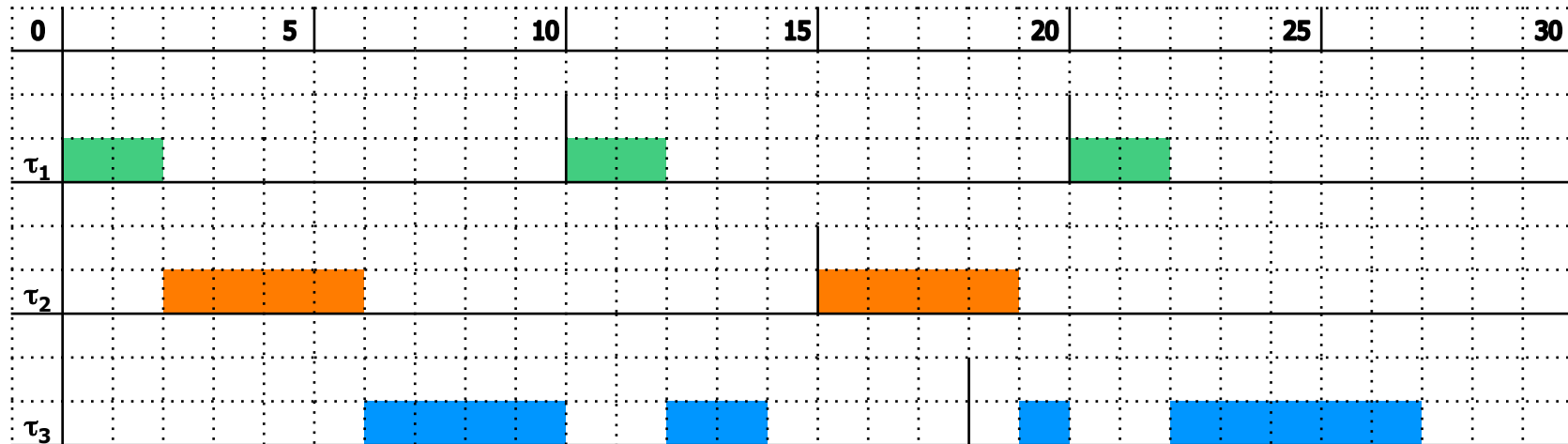
■ Scheduling test

- Necessary condition: $U \leq 1$
- Sufficient condition: $U \leq n(2^{1/n} - 1)$
 $\lim_{n \rightarrow +\infty} n(2^{1/n} - 1) = \log(2) = 69\%$

Static Priority Scheduling

Rate Monotonic Scheduling

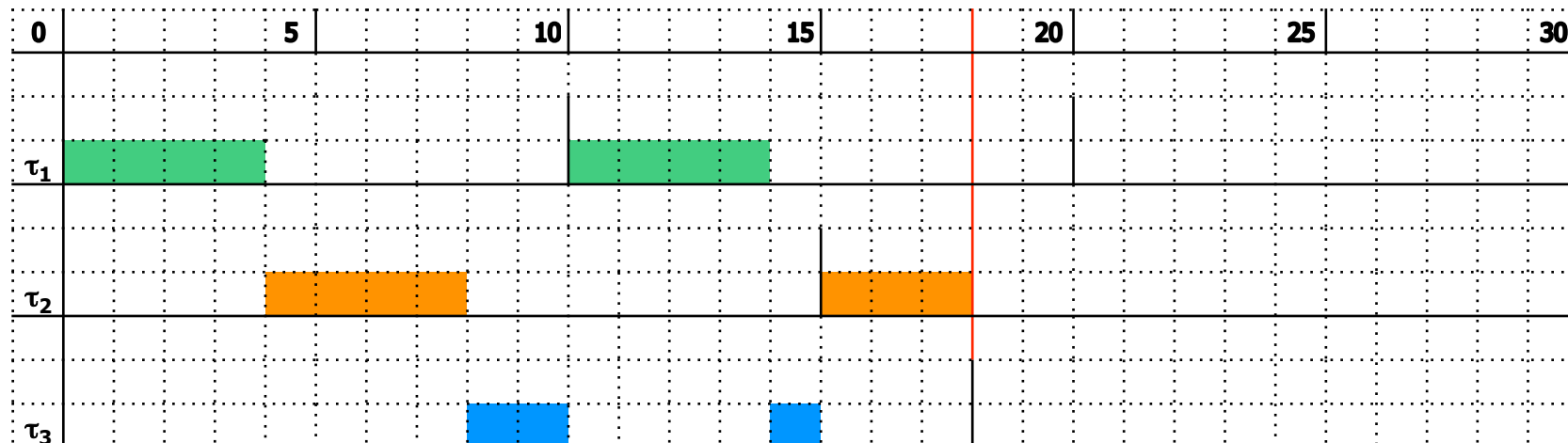
	Period	WCET	Usage
τ_1	10	2	0.200
τ_2	15	4	0.267
τ_3	18	6	0.333



Static Priority Scheduling

Rate Monotonic Scheduling

$3 \times (2^{1/3} - 1) \approx 0.78$	Period	WCET	Usage
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	18	6	0.333

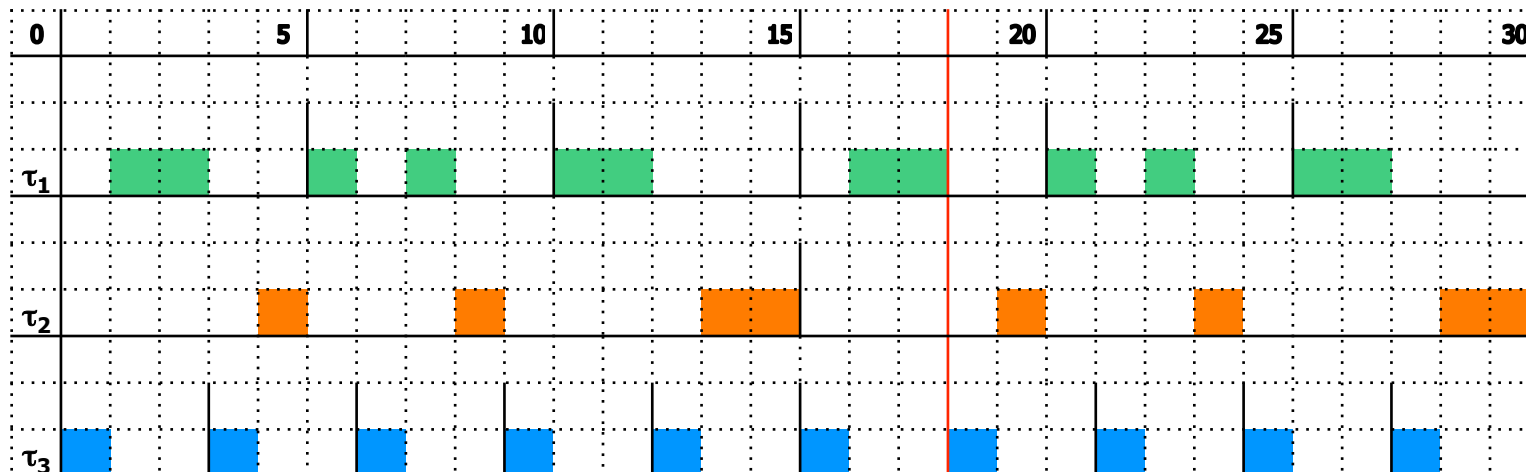


Static Priority Scheduling

Rate Monotonic Scheduling

$3 \times (2^{1/3} - 1) = 0.78$	Period	WCET	Usage
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	18	6	0.333

$3 \times (2^{1/3} - 1) = 0.78$	Period	WCET	Usage
$\tau'_1 = \tau_1 / 2$	5	2	0.400
τ'_2	15	4	0.267
$\tau'_3 = \tau_3 / 6$	3	1	0.333





Static Priority Scheduling

Rate Monotonic Scheduling

- Advantages

- Easy to implement
- Optimal for static priority scheduling
- Frequent in the classic executives
- Good behavior in case of overload

- Disadvantages

- Possible oversizing of the system

- RMS is always a possible result of OPA

- Both RMS and OPA are optimal



OPA vs RMS

$3 \times (2^{1/3} - 1) = 0.78$	Period	WCET	Usage
τ_1	5	2	0.400
τ_2	15	4	0.267
τ_3	3	1	0.333

- τ_1 lowest priority: $R_0 = 2$; $R_1 = 7$; or $R_1 > T_1$
- τ_2 lowest priority: $R_0 = 4$; $R_1 = 8$; $R_2 = 14$; $R_3 = 15$;
 - τ_1 : $R_0 = 2$; $R_1 = 3$; $\tau_2 < \tau_1 < \tau_3$: same as RMS
 - τ_3 : $R_0 = 1$; $R_1 = 3$; $\tau_2 < \tau_3 < \tau_1$: different from RMS
- τ_3 lowest priority: $R_0 = 1$; $R_1 = 7$; or $R_1 > T_3$
- OPA always finds an assignment if it exists (optimal), in particular the assignment of RMS (also optimal)



Static Priority Scheduling

Deadline Monotonic Scheduling

- Hypotheses
 - Synchronous and independant tasks
 - The deadline is less than the period ($D_i \leq T_i$)
- Principle
 - Select the ready task with the shortest deadline
 - When for all tasks $T_i = D_i$, DMS becomes RMS
- Scheduling test
 - The necessary and sufficient condition exists
 - Sufficient condition:
$$\sum \frac{C_i}{D_i} \leq n (2^{1/n} - 1)$$



Static Priority Scheduling

Deadline Monotonic Scheduling

- Advantages
 - See RMS
 - RMS penalizes long period but short deadline tasks
 - DMS is better in this case.
- Disadvantages
 - See RMS
 - Do not to be confused with EDF



Dynamic Priority Scheduling

Earliest Deadline First

- Hypotheses

- Periodic, independent tasks
- Deadline implicit ($D_i = T_i$) or not ($D_i \leq T_i$)

- Principle

- Task activation or completion wake up the scheduler
- Select the ready task with the earliest deadline

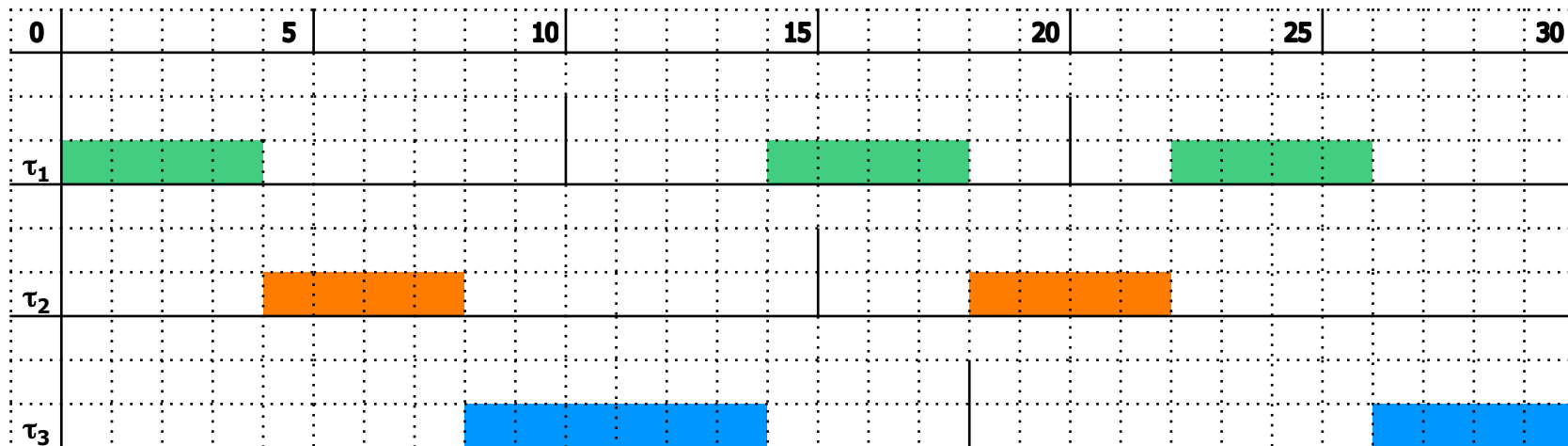
- Scheduling test

- Necessary and sufficient condition
- Sufficient when not implicit ($D_i \leq T_i$)
$$\sum C_i / T_i \leq 1$$
$$\sum C_i / D_i \leq 1$$

Dynamic Priority Scheduling

Earliest Deadline First

	Period	WCET	Usage
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	18	6	0.333





Dynamic Priority Scheduling

Earliest Deadline First

- Advantages

- Possible use of 100% of the processor
- Optimal for dynamic priority scheduling if the deadlines are lower than the periods

- Disadvantages

- Slight complexity of implementation
- Less common in executives than RMS
- Bad behavior in case of overload

- Remarks

- If D_i is arbitrary compared to T_i , the necessary and sufficient condition is no longer sufficient $\sum C_i / T_i \leq 1$



Dynamic Priority Scheduling

Least Laxity First

- Hypotheses

- Similar to those of EDF

- Principle

- Task activation or completion wake the scheduler
- Select the ready task with the lowest margin
- margin = deadline – remaining comp. time – current time

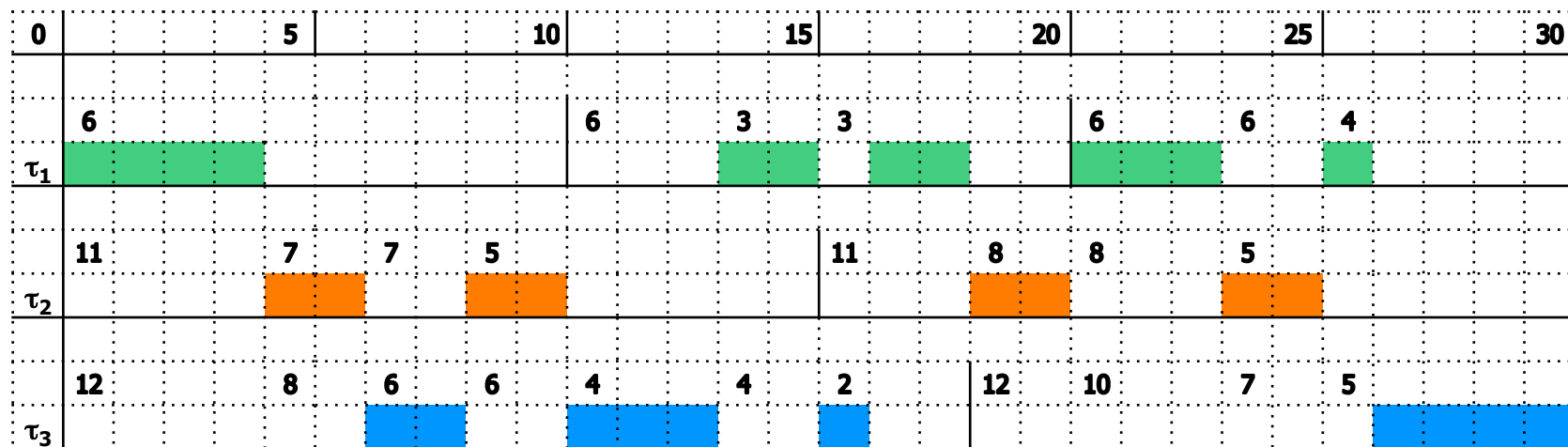
- Scheduling test

- Necessary and sufficient condition: $\sum C_i / T_i \leq 1$

Dynamic Priority Scheduling

Least Laxity First

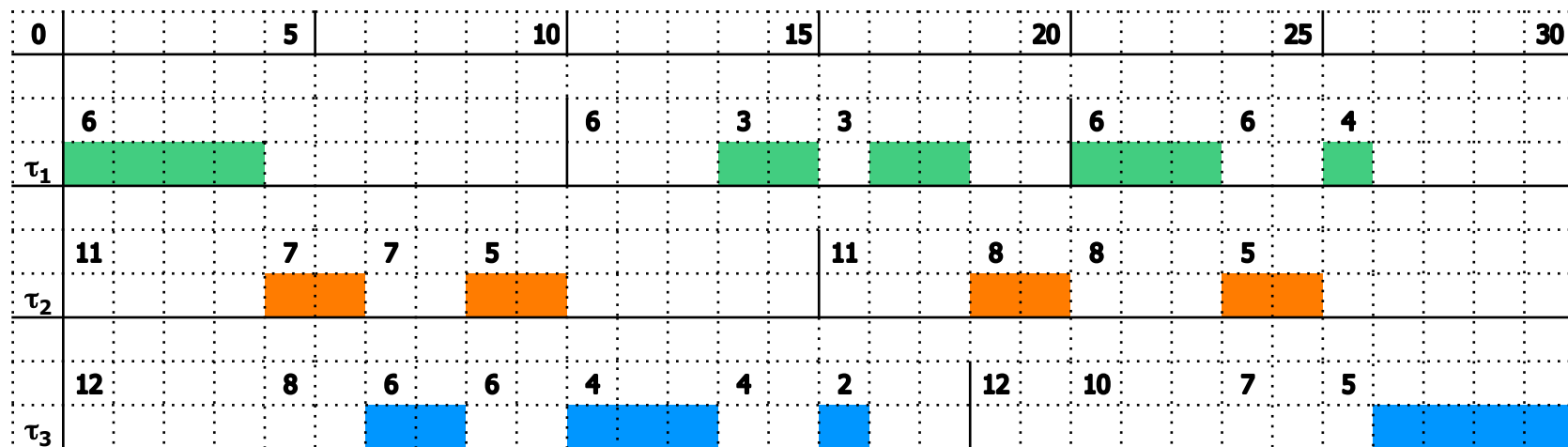
	Period	WCET	Usage
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	18	6	0.333



Dynamic Priority Scheduling

Modified Least Laxity First

	Period	WCET	Usage
τ_1	10	4	0.400
τ_2	15	4	0.267
τ_3	18	6	0.333





Dynamic Priority Scheduling

Least Laxity First

- Advantages
 - Better than EDF in the case of multi-processor
- Disadvantages
 - High complexity of implementation
 - Complex to compute remaining execution time
 - Bad behavior in case of overload
 - High number of preemptions
 - LLF oscillates in case of tied-laxities tasks



Real-Time Scheduling for Multi-Processors Systems

Frank Singhoff
Laurent Pautet
strec.wp.mines-telecom.fr

Version 1.0

Architecture Issues

No Mono-Processor Architecture Anymore

- Historically ... mono-processors
 - platform = a dedicated processor, a clock and a common memory ...
 - predictable (cache and pipeline inhibited)
 - no longer common technology, limited performance
- Trends ... multi-processors
 - Use COTS (not dedicated) processors (FAA, 2011).
 - Shared resources => +interferences ; -predictable
 - More powerful, but less predictable (cannot inhibit interconnection bus)



Architecture Issues

Interferences with Multi-Processors

- Let's have task T_1 (resp T_2) running on core C_1 (resp C_2) ; C_1 and C_2 share a common cache L_2 or an interconnection bus
- T_1 and T_2 are functionally independent ... but finally dependent because of shared hardware resources inducing interferences
- A task can be delayed due to contention / interference on shared hardware
- This can be an even more important problem in multi-processors than in mono-processor

Multi-Processors Architecture

Processors

- Identical processors: processors all executing the same units of work during the same units of time
- Uniform processors: processor j with speed s_j executes $s_j \cdot t$ units of work for t units of time.
- Heterogeneous processors: processor j executes $s_{i,j} \cdot t$ units of work of job i for t units of time.
- Heterogenous processors : no shared memory, nor migration (a distributed system)



Multi-Processors Architecture Scheduling

- Mono-Processor scheduling : 1 problem
 - Time Allocation – when to execute a task
- Multi-Processors scheduling : 2 problems
 - Processor Allocation – where to execute a task
 - Time Allocation – when to execute a task

As a consequence, most results from
mono-processor real-time scheduling theory
are **no longer true** for
multi-processors real-time scheduling theory

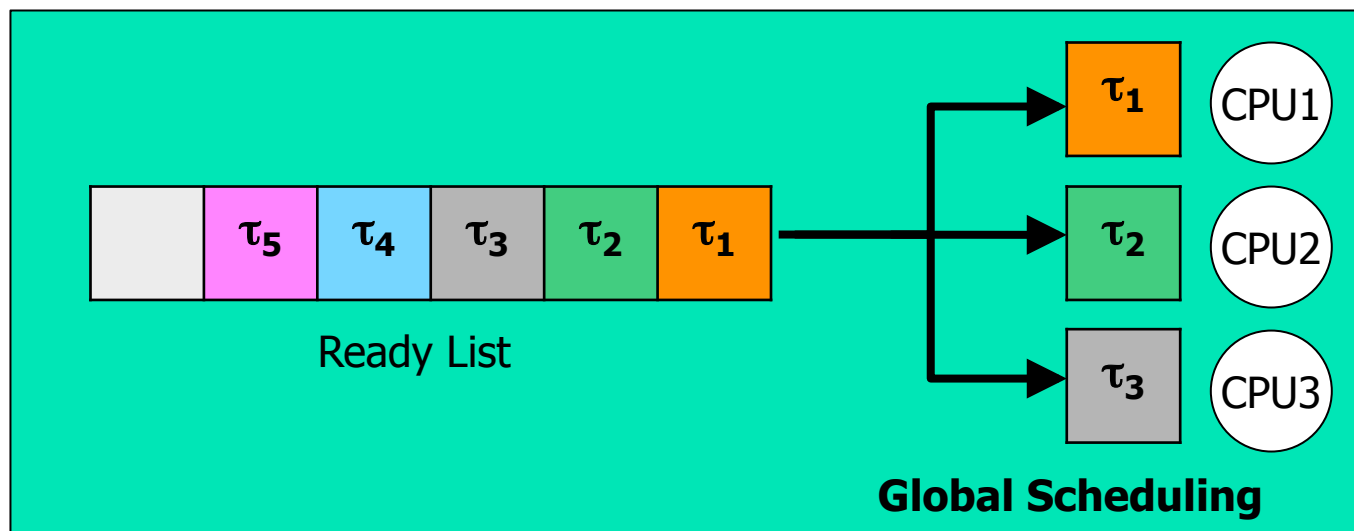
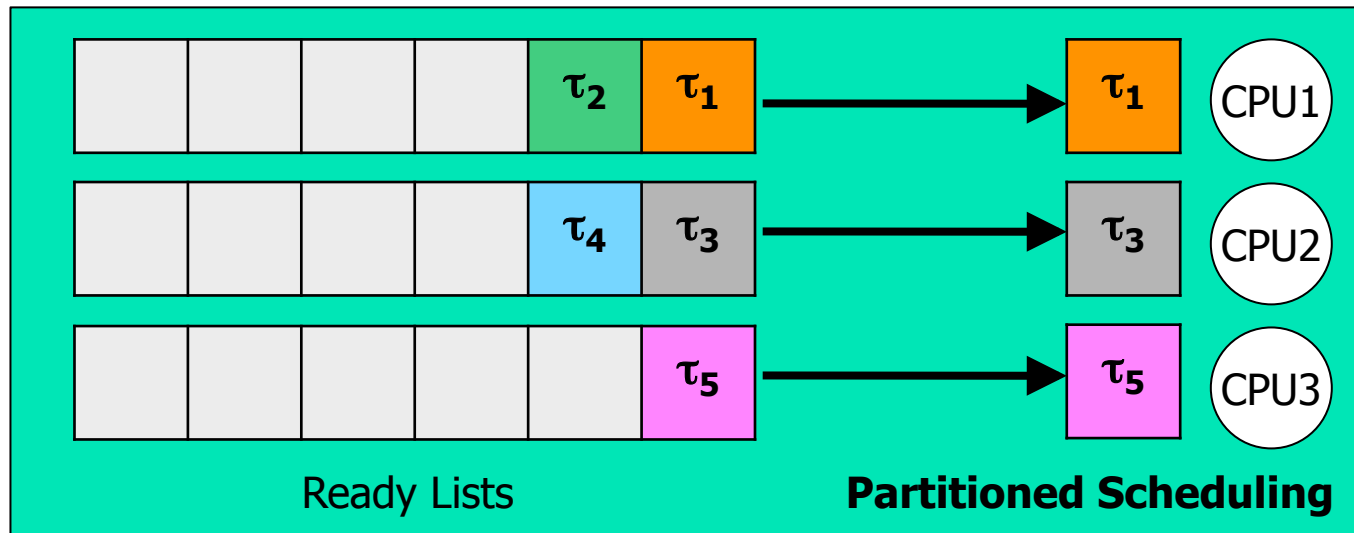


Multi-Processors Scheduling Different Approaches

- Partitioned scheduling (offline processor allocation)
 - Handle separately processor and time allocations
 - Map all tasks on processors
 - Schedule tasks on each processor.
 - Possible end-to-end delay verification
- Global scheduling (online processor allocation)
 - Handle globally processor and time allocations
 - Pick a task from a global ready list
 - Map it on one of the idle processors
- Hybrid scheduling (mixed approach)
 - Offline allocation of tasks to Virtual Processes (servers)
 - Online scheduling of Virtual Processes (and tasks as well)

Scheduling Approaches

Partitioned Scheduling Approach





Partitioned Scheduling Task Assignment

- How to statically assign tasks to processors
- Bin-packing problem: minimize the number of bags to pack bins of different volumes
- NP-hard problem => partitioning heuristics
 - Different parameters:
 - Processors (identical or not), tasks (periods, budgets), etc.
 - Task communications, shared resources, etc.
 - Different objective functions:
 - Minimize processors, communications, latencies, etc.
- Difficult to compare heuristics
 - Especially when the final objective is actually schedulability



Partitioned Scheduling

Assignment and Scheduling Variants

- Sort tasks before packing
 - Ascending/descending order of utilization/period
- Select a mono-processor scheduling
 - RM or DM, EDF or LLF
 - Schedulability test to allocate a task to a processor
- Select a bin-packing heuristic
 - First-Fit, Next-Fit, Worst-Fit or Best-Fit

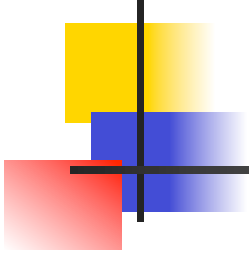


Partitioned Scheduling

Rate-Monotonic Next-Fit

- List tasks in ascending order of their utilization/period.
- Processor $p=0$
- For task $t=0$ to n
 - Assign task t to processor p if the feasibility test is met (eg: $U \leq 0.69$ or response time computation)
 - Stop when no processor found
 - Loop to next processor $p = (p+1) \bmod m$

Partitioned Scheduling Limitations

- 
- Partitioned Scheduling cannot be optimal
 - m processors
 - $(m+1)$ tasks of parameters (C, T) , $C = T/2 + \varepsilon$
 - Exercice : Prove that for periodic tasksets with implicit deadlines, the largest worst-case utilization bound for any partitioning algorithm is $(m+1)/2$.



Partitioned Scheduling

Pros and Cons

■ Pros

- Better suitability for heterogeneous systems
- Inherit from mature mono-processor scheduling
- Time and space isolation (major safety property)
 - Failures / anomalies limited to one processor

■ Cons

- 2 problems both being NP-hard
 - Processor allocation (mapping)
 - Time allocation (scheduling)
- Less optimal use of resources (idle processors)



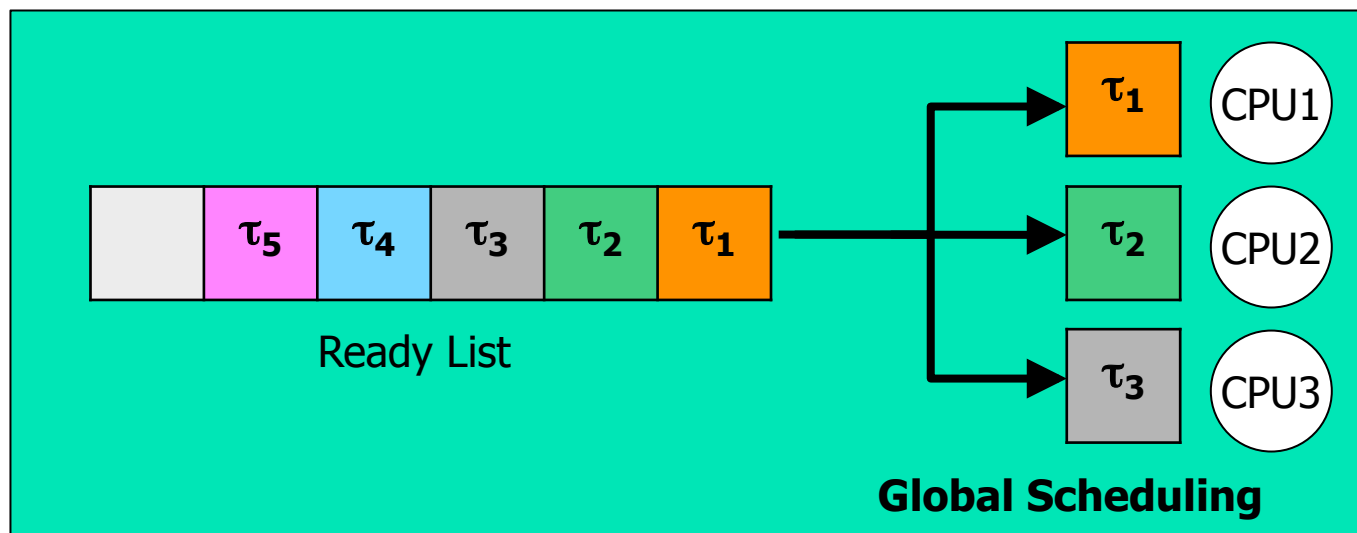
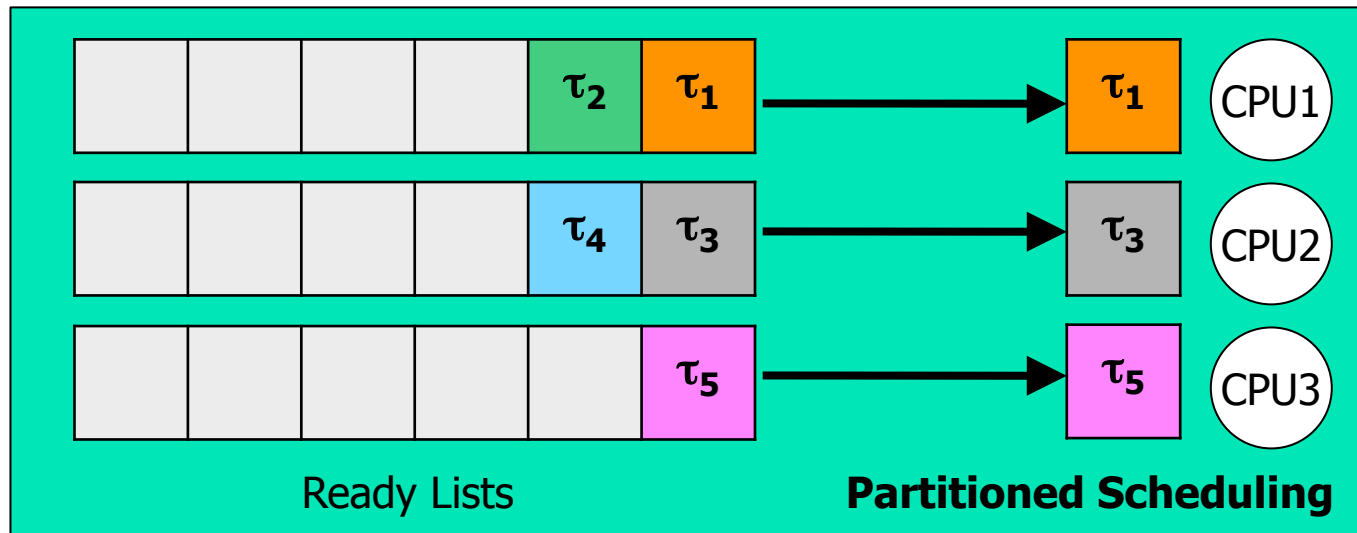
Partitioned Scheduling

Other resources (memory, bus, ...)

- Similar benefits/limitations for other resources
 - resource partitioning and
 - resource sharing
- Resource partitioning : great predictability ... but resources less efficiently used
- Global resource sharing: poor predictability ... but resources more efficiently used
- Example: partitioned cache vs shared cache
 - Partition too small: time to reload data
 - Partition too large: waste of resource

Scheduling Approaches

Global Scheduling





Global Scheduling Pros and Cons

■ Pros

- Optimal scheduling exist
- Better suited for homogeneous multi-core architectures
- Better resource optimization : busy cores, less preemptions ... but migrations

■ Cons

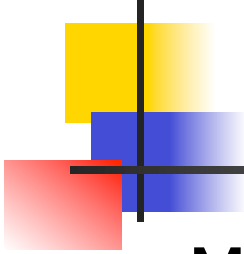
- Not well suited at all to heterogeneous systems
- More recent and less numerous results of scheduling theory
- ... for simple architectures and task models



Global scheduling

Sharing resources

- A global scheduler deals with two problems:
 - When and how to assign task / job priorities.
 - Choose a processor on which to run the task.
- Sharing time
 - Preemption (same as mono-processor)
 - A job starts its execution in a time interval and ends in another time interval
- Sharing processors
 - Migration
 - A job starts its execution on a processor and ends on another processor



Global Scheduling

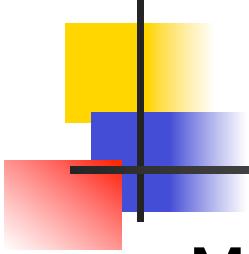
Migrations and Priorities

Migration strategies

- No task migration: All its jobs are assigned to a given processor => partitioning
- Task migration: Jobs can start executing on different processors but complete on their selected processor
- Job migration: A job can migrate during its execution.

Priority assignments

- Fixed priority associated to a task (eg: RM).
- Fixed priority associated to a job (eg: EDF).
- Dynamic priority associated to a job (ex: LLF).



Global Scheduling

Two general approaches

Mono-processor based global scheduling:

- Global RM, Global DM, Global EDF, Global LLF, ...
 - Variants depending on migration level (task or job)
- Globally apply a mono-processor scheduling strategy on all processors. Assign the m highest priority tasks or jobs to the m processors at any time.
- Task or job preemption when all processors are busy

New algorithms: PFair, RUN, ...

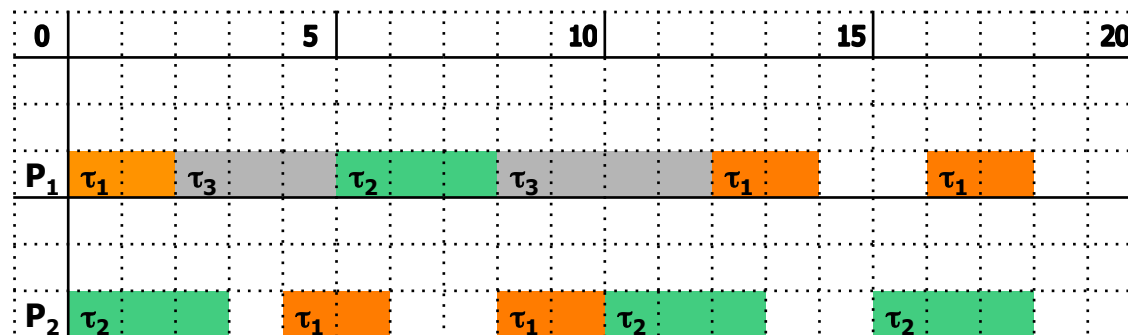
Different and fewer results and properties
compared to mono-processor scheduling

Mono-Processor based Global Scheduling

Different response times

- Use of Global Deadline Monotonic scheduling
- Priority assignment: $\tau_1 > \tau_2 > \tau_3$
- Tasks can migrate, jobs cannot

	C	T	D
τ_1	2	4	4
τ_2	3	5	5
τ_3	7	20	20

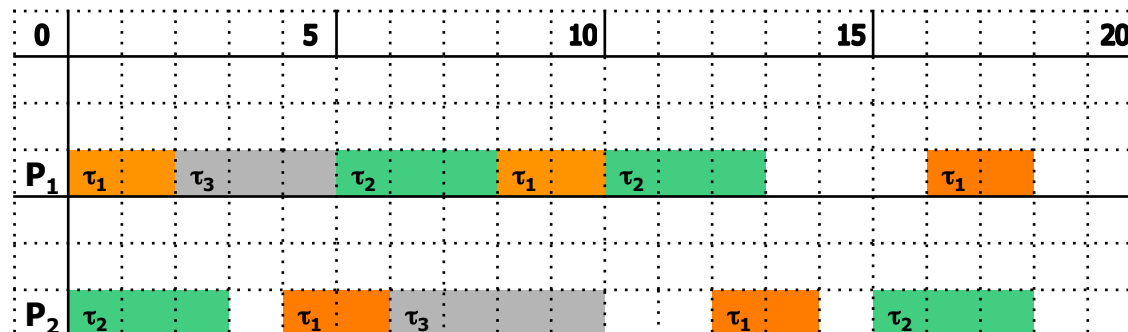


Mono-Processor based Global Scheduling

Different response times

- Use of Global Deadline Monotonic scheduling
- Priority assignment: $\tau_1 > \tau_2 > \tau_3$
- Jobs can migrate
- Not the same response time for τ_3

	C	T	D
τ_1	2	4	4
τ_2	3	5	5
τ_3	7	20	20

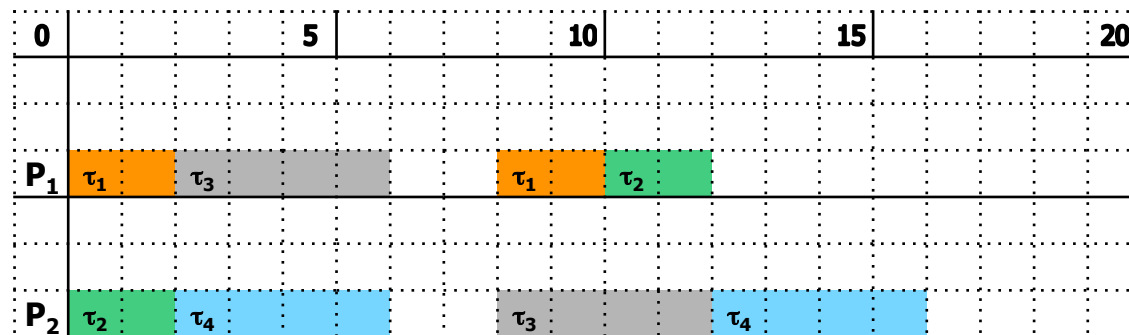


Mono-Processor based Global Scheduling

No Critical Instant

- In a mono-processor, the critical instant is the worst case scenario for periodic tasks
- All tasks are released at the same instant
- Used to compute the worst response time
- But not the worst scenario in multi-processors.
- Here, $R_4=8$ but with critical instant $R_4=6$

	C	D	T
τ_1	2	2	8
τ_2	2	4	10
τ_3	4	6	8
τ_4	4	8	8





Mono-Processor based Global Scheduling

Different feasibility interval

- In mono-processor, the feasibility interval is used to check schedulability of independent asynchronous / synchronous periodic tasks, $\forall i: D_i \leq P_i$ with a fixed priority scheduling $[0, 2 * \text{LCM}(\forall i: P_i) + \max(\forall i: S_i)]$
- In multi-processors, a similar result: $[0, \text{LCM}(\forall i: P_i)]$ but for a set of independent synchronous periodic tasks only



Mono-Processor based Global Scheduling

Scheduling anomalies

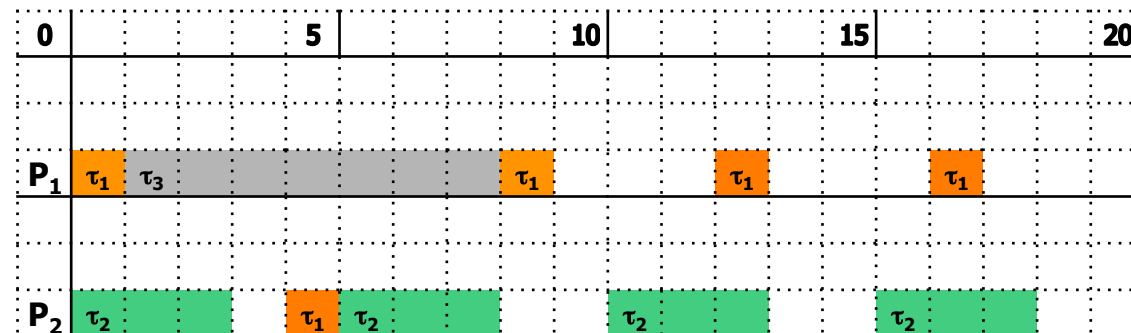
- Anomaly: intuitively positive change in a schedulable set of tasks that leads to a non-schedulable set of tasks
- In mono-processor, when a tasks set is schedulable, it is still schedulable if we lower its utilisation (reduce C_i or increase T_i)
- In multi-processor, this is no longer true

Mono-Processor based Global Scheduling

Scheduling anomalies

- Use of Global Deadline Monotonic Scheduling
- Jobs can migrate
- $U_1 = 1/4$
- Tasks set is schedulable

	C	T	D
τ_1	1	4	2
τ_2	3	5	3
τ_3	7	20	8

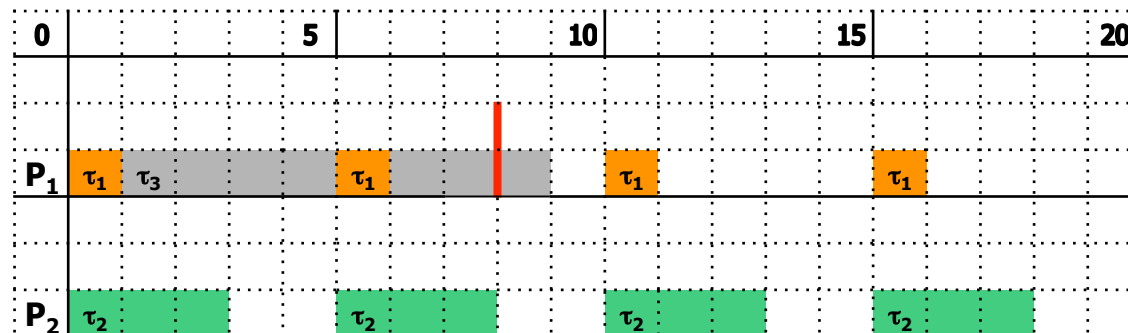


Mono-Processor based Global Scheduling

Scheduling anomalies

- Use of Global Deadline Monotonic Scheduling
- Task τ_1 has a larger period
- Task set with a lower utilisation ($1/4 \rightarrow 1/5$)
- Tasks set is non-schedulable ($R_3=9 > D_3$)

	C	T	D
τ_1	1	5	2
τ_2	3	5	3
τ_3	7	20	8





Mono-Processor based Global Scheduling Limitations

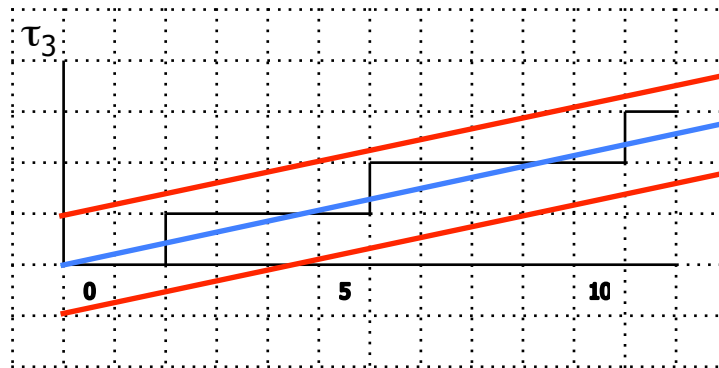
- m processors
- $(m+1)$ tasks of parameters (C, T) , $C = T/2 + \epsilon$
- Exercice : Prove that the maximum utilization bound for any global fixed job priority algorithm is $(m+1)/2$.
- Global LLF (dynamic priority per job) $>$ Global EDF (static priority per job)

Global Scheduling

Pfair Algorithms : Principles

- The proportion of time units allocated at instant t to a task must remain as close as possible to its utilisation
- Optimal algorithm for identical processors and synchronous deadline implicit periodic tasks
- Lots of preemptions and migrations

	C	T
τ_1	1	2
τ_2	1	3
τ_3	2	9





Global Scheduling

Pfair Algorithms : Modelling

- Execute tasks at a constant rate (fluid model) such as $\forall i: \text{workload}(\tau_i, t) = t * C_i / T_i$
- Can be approximated by $\text{sched}(\tau_i, t)$ where $\text{sched}(\tau_i, t) = 1$ when τ_i is scheduled in interval $[t, t + 1[$, $\text{sched}(\tau_i, t) = 0$ otherwise
- A schedule is said to be Pfair if and only if $\text{lag}(\tau_i, t) = \text{workload}(\tau_i, t) - \sum_{k \leq t} \text{sched}(\tau_i, k)$ where $\forall i, \forall t: -1 \leq \text{lag}(\tau_i, t) \leq 1$
- A Pfair scheduling is feasible on m processors as long as $U \leq m$ (full utilization !)



Global Scheduling

Pfair Algorithms : Implementation

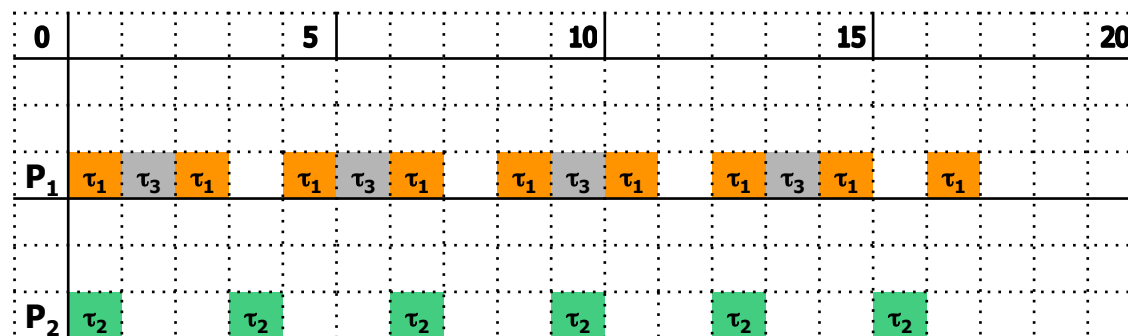
- Split each task i into C_i subtasks (1 time unit)
- Assign a pseudo deadline $d(\tau_i, j)$ and a pseudo release $r(\tau_i, j)$ to subtask j in $[1..C_i]$:
 - $d(\tau_i, j) = \lceil j * T_i / C_i \rceil$
 - $r(\tau_i, j) = \lfloor (j - 1) * T_i / C_i \rfloor$
- Schedule subtask j according to $d(\tau_i, j)$ (EDF)
- Improve Pfair with non-arbitrary tie breaks to reduce context switches and migrations in case of identical pseudo-deadlines

Global Scheduling

Pfair Algorithms : Example

- $r(\tau_i, j) = \lfloor (j-1) * T_i/C_i \rfloor$ and $d(\tau_i, j) = \lceil j * T_i/C_i \rceil$
- $r(\tau_1, 1) = 0$; $d(\tau_1, 1) = 2$; $U_1 = 1/2$
- $r(\tau_2, 1) = 0$; $d(\tau_2, 1) = 3$; $U_2 = 1/3$
- $r(\tau_3, 1) = 0$; $d(\tau_3, 1) = 5$; $U_3 = 2/9$
- $r(\tau_3, 2) = 4$; $d(\tau_3, 2) = 9$; $U_3 = 2/9$

	C	T
τ_1	1	2
τ_2	1	3
τ_3	2	9



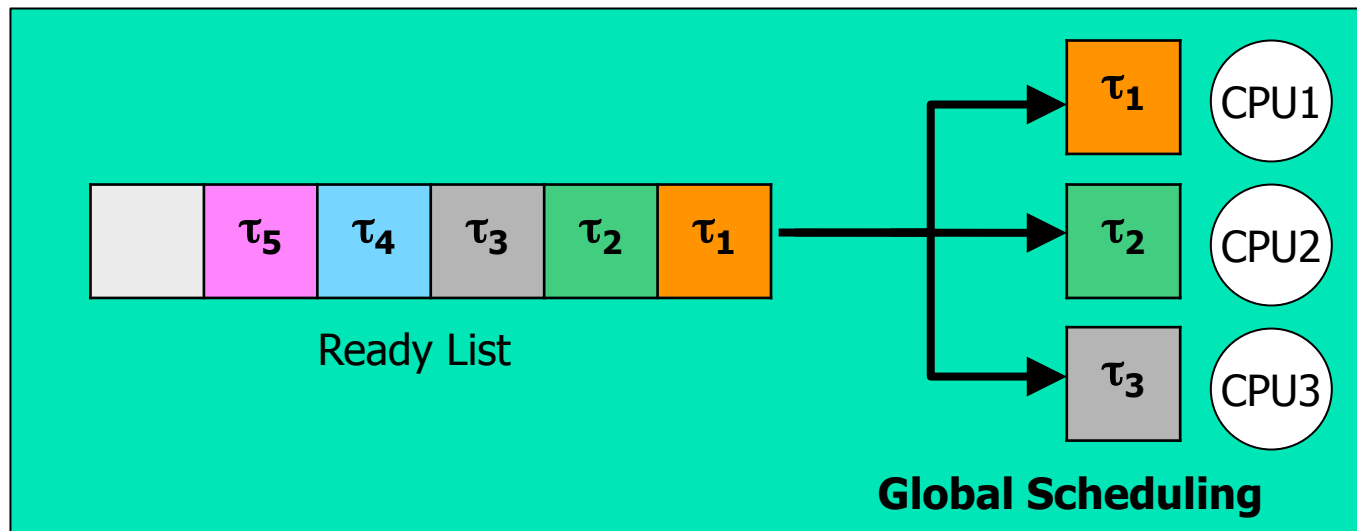
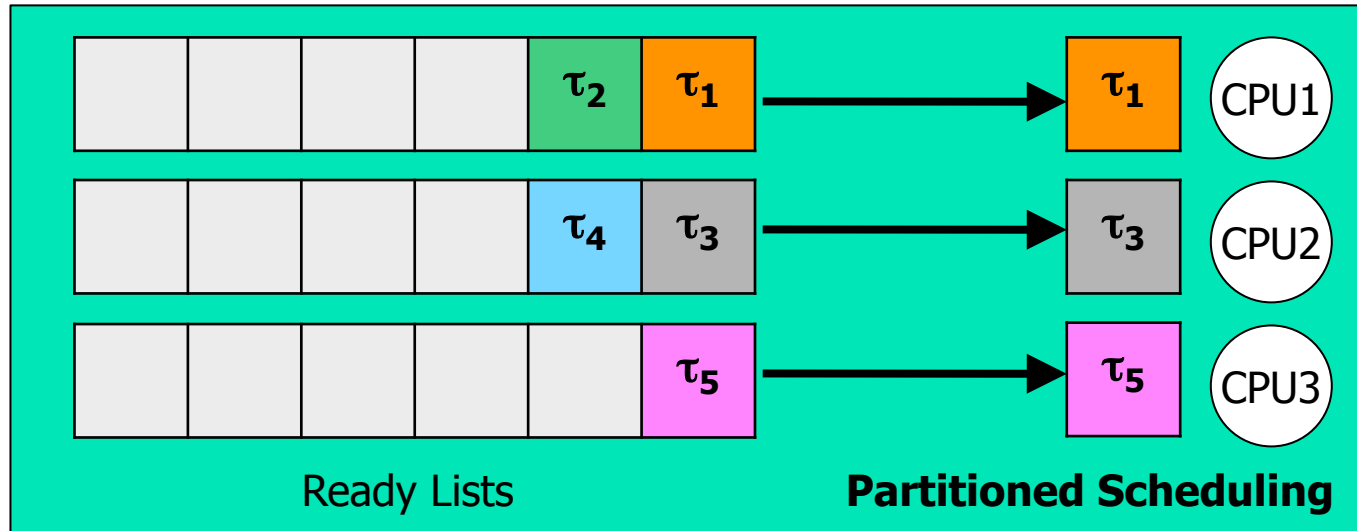


Global Scheduling Conclusions

- Global multi-processor scheduling has different properties compared to mono-processor scheduling (optimality, critical instant, feasibility interval, anomalies, ...).
- Additional parameters : migration, task / processor assignment, ...
- We limited architecture to identical processors, without shared resources
- We have limited task model to a simplified task one
- We have not discussed dependencies between tasks (shared resources, precedence constraints), nor communications.

Scheduling Approaches

Hybrid Scheduling





Hybrid Scheduling Principles

- A mixed solution between partitioned (offline) and global scheduling (online)
- Example: RUN (Reduction to Uniprocessor)
 - Optimal, less preemptions compared to PFair
 - Offline: build a reduction tree (PACK & DUAL steps)
 1. PACK tasks on a min nbr of virtual processors/servers
 2. Stop when schedule on a single processor/server
 3. Define idle time of processors/servers as DUAL idle tasks
 4. Loop to step 1
 - Online: schedule reduction tree (schedule tasks / processors in a virtual processor using EDF)

RUN

Offline : PACK + DUAL (first layer)

- Pack tasks on a minimum number of virtual processors (servers) S_1 to S_3 . Use First-Fit.
- So, we cannot merge 2 virtual processors (VP)
- 3 idle time intervals : S_1^* to S_3^*

U=2

	C	T
τ_1	2	7
τ_2	2	7
τ_3	2	7
τ_4	4	14
τ_5	4	14
τ_6	4	14
τ_7	2	7

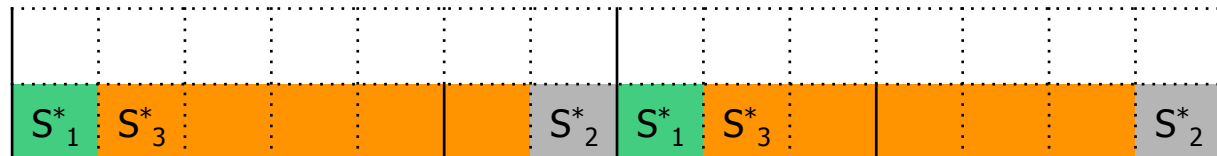
S_1													
τ_1		τ_2		τ_3		S_1^*	τ_1		τ_2		τ_3		S_1^*
S_2													
τ_4				τ_5				τ_6				S_2^*	
S_3													
τ_7		S_3^*					τ_7	S_3^*					

RUN

Offline : DUAL + PACK (second layer)

- Define S_1^* to S_3^* as (dual) tasks
- They model the idle time left on VPs
- Pack and schedule S_1^* to S_3^* on 1 VP
 - This new VP schedules « idle tasks » : we free a processor as the idle time is packed on 1 processor
- While #processors > 1, loop DUAL+PACK steps

	C	T
S_1^*	1	7
S_2^*	2	14
S_3^*	5	7



RUN

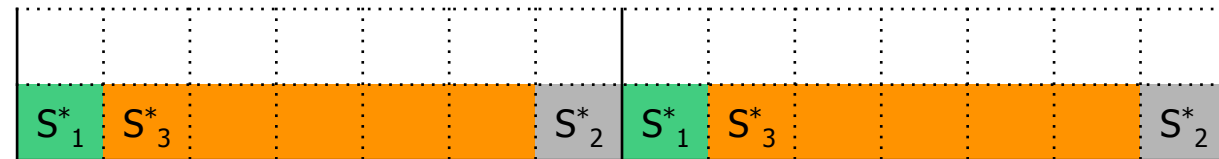
Online: Schedule Reduction Tree

- We have a tree of servers (or a hierarchy of servers) that schedules tasks and servers
 - We start scheduling the root server of the tree
 - When we schedule a dual server, we do not schedule its tasks or servers. We schedule the remaining tasks or servers applying EDF.
 - When we schedule a primary server, we do schedule its tasks or servers applying EDF
- In the example, we start executing S_1^* . Thus, we do not execute S_1 but S_2 ou S_3 . Applying EDF, S_2 will execute τ_1 and S_3 will execute τ_1

RUN

Online: Scheduling previous example

	C	T
S_1^*	1	7
S_2^*	2	14
S_3^*	5	7

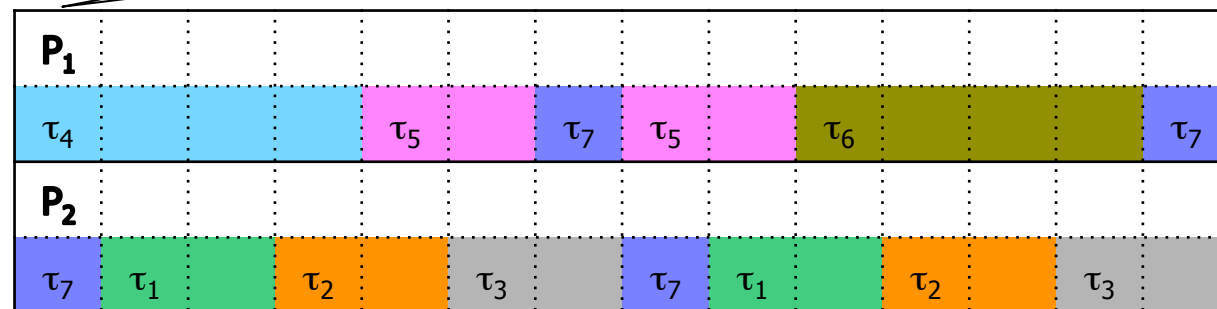


Schedule $S_1^* \Rightarrow$ schedule all but S_1 ,
 \Rightarrow schedule S_2 or S_3

	C	T
τ_1	2	7
τ_2	2	7
τ_3	2	7
τ_4	4	14
τ_5	4	14
τ_6	4	14
τ_7	2	7

U=2

Schedule T_4, T_5 or T_6 on P_1 and
 T_7 on P_2 both using EDF



Laurent Pautet

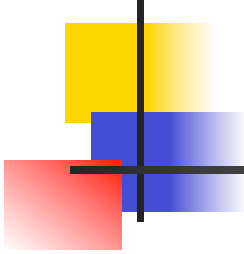


Real-Time Scheduling for Distributed Systems

- Tasks exchange messages
 - 1. Tasks are dependant and assigned to procs
 - The task input is the output of its predecessors
 - 2. Model and schedule messages as tasks

Non-preemptive task	Message
(Mono) Processor	Communication medium
Capacity / Budget	Communication delay (buffer, access, propagation)

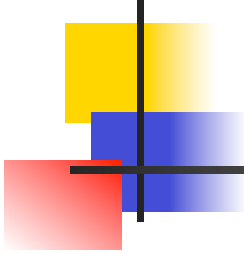
- 3. Schedule messages on bus or network
 - Use non-preemptive tasks scheduling



Distributed Real-Time Scheduling

Step 1: Dependant Tasks on Mono-Processors

- Dependant tasks on a mono-processor
 - Modify task parameters to have independant tasks
 - $A_i^* = \max (A_i, \max_{j \text{ in pred}(i)} A_j^* + C_j)$
 - $D_i^* = \min(D_i, \min_{j \text{ in succ}(i)} D_j^* - C_j)$
- For a static priority scheduling, give higher priority to predecessors than to task (DMS)
 - We can compute response time
- For a dynamic priority scheduling, use new deadlines (EDF)



Distributed Real-Time Scheduling

Step 2 : Dependant tasks on Distributed Systems

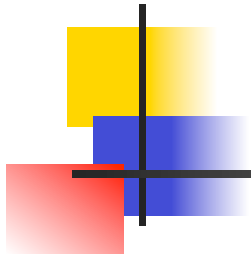
- Holistic Method
 - Compute response time with jitter ...
 - defined as the max response time of predecessors
- Iterative method (as for mono-processors)
- For task with an fixed priority scheduling
 - $R_i^{n+1} = J_i + C_i + \sum_{k \text{ in hp on proc } (i)} C_k * \lceil (J_k + R_i^n) / T_k \rceil$
- For message
 - $R_i = J_i + M_i$



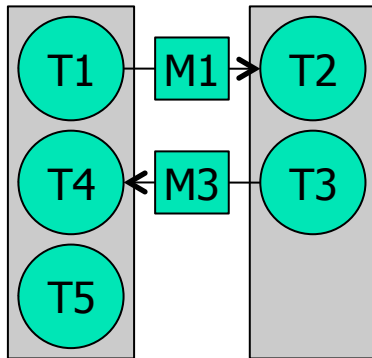
Distributed Real-Time Scheduling

Step 3 : Message Scheduling on (CAN) Bus

- Messages modeled as non-preemptive tasks
- Compute response time for static priority scheduling of non-preemptive tasks
- $$R_{n+1}^i = J_i + C_i + \sum_{k \text{ in } hp(i)} C_k * \lceil (J_k + R_n^i) / T_k \rceil + \max_{l \text{ in } lp(i)} (C_l)$$
 - The last term represents the blocking time induced by a lower priority non-preemptive task



Distributed Real-Time Systems



Step 1 : T_1 (resp T_3) has higher priority than successor T_2 (resp T_4)
 Priorities are computed with DMS and D_1 (resp D_3) $<$ D_2 (resp D_4)

Step 2 : $R_i^{n+1} = J_i + C_i + \sum_{k \in \text{pred}(i)} C_k * [(J_k + R_i^n) / T_k]$

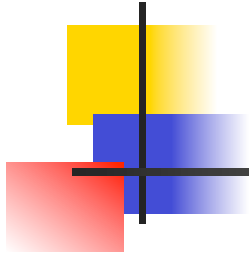
	M1	M3	T1	T2	T3	T4	T5
J	0	0	0	0	0	0	0
R	6	1	4	5	2	9	12

	T	C	Pri
T1	100	4	HI
T2	100	3	ME
T3	60	2	HI
T4	60	5	ME
T5	90	3	LO
M1	100	6	LO
M3	60	1	HI

	M1	M3	T1	T2	T3	T4	T5
J	4	2	0	6	0	1	0
R	10	3	4	11	2	10	12

	M1	M3	T1	T2	T3	T4	T5
J	4	2	0	10	0	3	0
R	10	3	4	15	2	12	12

Step 3 : M_1 and M_3 are schedulable on network (trivial)



Conclusions

- Less mono-processors, more multi-processors or heterogeneous systems on the market
- Very active research domain to design new scheduling approaches
- Less predictive processors on the market ; approximate WCET due to many interferences
- Define modes and change mode when overloaded
- The low criticality mode includes all the tasks
- The high criticality one only high criticality tasks
- Active research domain : mixed criticality Systems