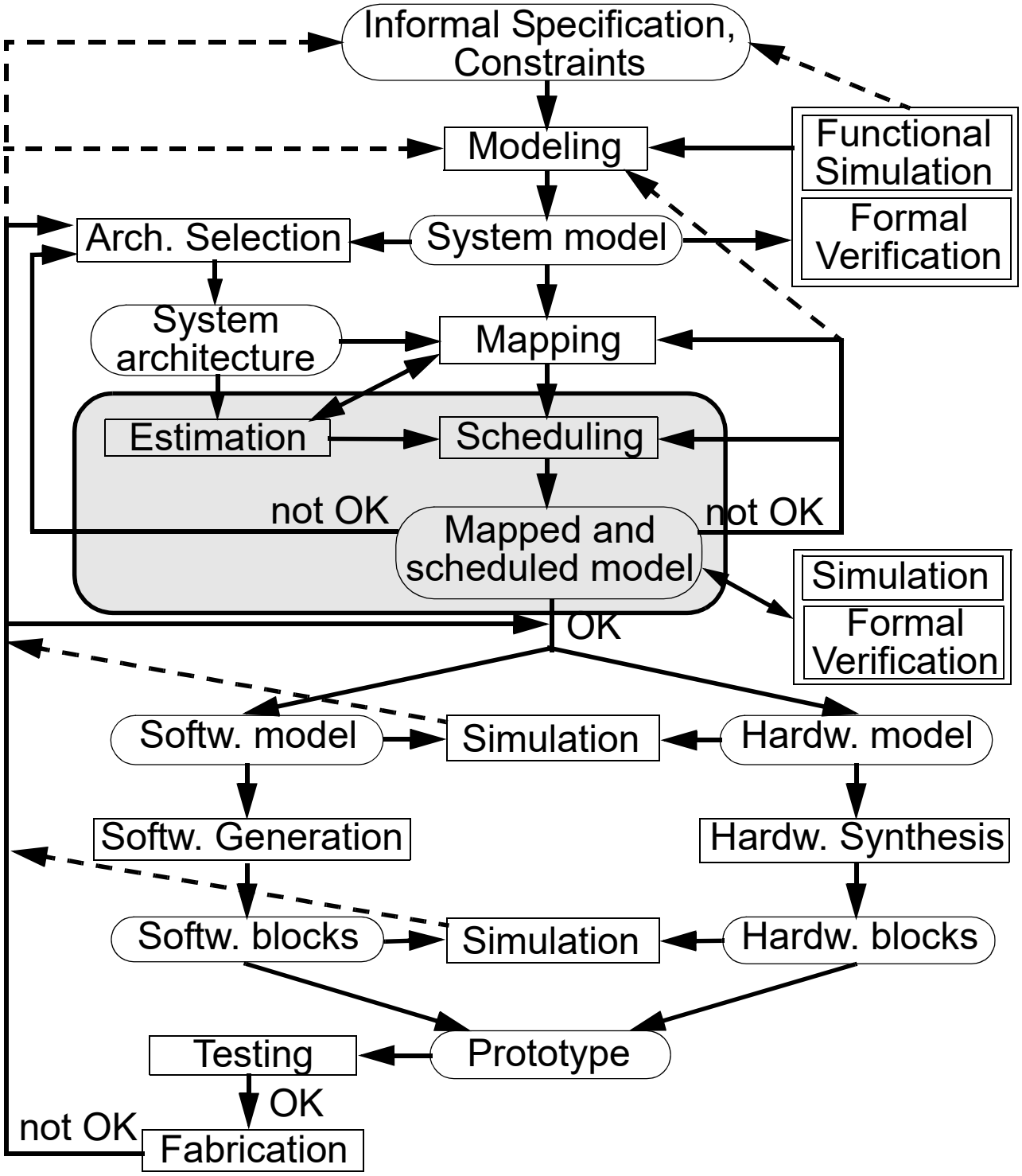
REAL-TIME EMBEDDED SYSTEMS: TASK SCHEDULING

1. Real-Time Systems and Their Typical Features
2. Task Scheduling Policies
3. Static Cyclic Scheduling
4. What is Good and Bad with Static Cyclic Scheduling
5. Priority Based Preemptive Scheduling
6. Schedulability Analysis


Informal Specification, Constraints
Modeling
Functional Simulation
Formal Verification
Arch. Selection
System model
System architecture
Mapping
Estimation
Scheduling
not OK
Mapped and scheduled model
not OK
Simulation
Formal Verification
OK
Softw. model
Simulation
Hardw. model
Softw. Generation
Hardw. Synthesis
Softw. blocks
Simulation
Hardw. blocks
Testing
Prototype
OK
not OK
Fabrication

Real-Time Systems

- Many (most) embedded systems are real-time systems.
  - A real-time system is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations but also on the time when the results are produced.

Examples:

- Process control systems
- Computer-integrated manufacturing systems
- Aerospace and avionics systems
- Automotive electronics
- Medical equipment
- Nuclear power plant control
- Defence systems
- Consumer electronics
- Multimedia
- Telecommunications

Real-Time Systems: Some Typical Features

- They are time-critical.

  The failure to meet time constraints can lead to degradation of the service or to catastrophe.

- They are made up of concurrent tasks.

  The tasks share resources (e.g. processor) and communicate to each other. This makes scheduling of tasks a central problem.

- Reliability and fault tolerance are essential.

  Many applications are safety critical.

Soft and Hard Real-Time Systems

- Time constraints are often expressed as *deadlines* at which tasks have to complete their execution.

A deadline imposed on a task can be:

Soft and Hard Real-Time Systems

- Time constraints are often expressed as *deadlines* at which tasks have to complete their execution.

A deadline imposed on a task can be:

- Hard deadline: has to be met strictly, if not $\Rightarrow$ "catastrophe".
  Has to be guaranteed at design time!

Soft and Hard Real-Time Systems

- Time constraints are often expressed as *deadlines* at which tasks have to complete their execution.

A deadline imposed on a task can be:

- Hard deadline: has to be met strictly, if not ⇒ "catastrophe".

  Has to be guaranteed at design time!

- Soft deadlines: tasks can finish after their deadline, although the value provided by completion may degrade with time.

Soft and Hard Real-Time Systems

- Time constraints are often expressed as *deadlines* at which tasks have to complete their execution.

A deadline imposed on a task can be:

- Hard deadline: has to be met strictly, if not $\Rightarrow$ "catastrophe".

  Has to be guaranteed at design time!

- Soft deadlines: tasks can finish after their deadline, although the value provided by completion may degrade with time.

- Firm deadlines: similar to hard deadlines, but if the deadline is missed there is no catastrophe; only the result produced is of no use any more.

Predictability

- Predictability is a very important property of any real-time system.

- Predictability means that it is possible to guarantee that deadlines are met as imposed by requirements:
  - Hard deadlines are always fulfilled.
  - Soft deadlines are fulfilled to a degree which is sufficient for the required quality of service.

Predictability

Some problems concerning predictability:

- Determine worst case execution times for each task.
- Determine worst case communication delays on the interconnection network.
- Determine time overheads due to operating system (interrupt handling, task management, context switch, etc.).

Predictability

Some problems concerning predictability:

- Determine worst case execution times for each task.
- Determine worst case communication delays on the interconnection network.
- Determine time overheads due to operating system (interrupt handling, task management, context switch, etc.).
  - After the above problems have been solved, comes the "big question":

    Can the given tasks and their related communications be scheduled on the available resources (processors, buses), so that deadlines are satisfied?

Task Scheduling

The scheduling problem:

Which task and communication has to be executed at a certain moment on a given processor or bus respectively, so that time constraints are fulfilled?

- **A set of tasks is *schedulable* if, given a certain scheduling policy, all constraints will be completed (which means, a solution to the scheduling problem can be found).**
- **At least for hard real-time systems, it is needed to check off-line, in advance, if the system is schedulable.**

Task Parameters

What do we assume to know about a task?

- Computation time (worst case computation time), *c*.
  For communication, we assume to know communication time.
- Deadline for task completion, *d*.
- Regularity of task arrival:
  - *periodic* tasks, with period *T* (infinite sequence of identical activities).
  - *aperiodic* tasks: no fixed period of arrival
    - *sporadic* tasks: bound minimum inter-arrival time $\Rightarrow$ deadlines can be guaranteed off-line.
    - If no bounds on inter-arrival time are known, schedulability cannot be guaranteed.

Scheduling Policies

- **Static cyclic scheduling**

  A table is generated off-line containing activation times for each task (communication). The activation sequence captured by the table is repeated cyclically.

Scheduling Policies

- **Static cyclic scheduling**

  A table is generated off-line containing activation times for each task (communication). The activation sequence captured by the table is repeated cyclically.

- **Priority based scheduling**

  Tasks are activated in response to a certain event. In case of conflict (several tasks ready to execute on the same processor), priorities are considered.

Scheduling Policies

- Preemptive scheduling
  - A running task can be interrupted in order to execute another task.
- Non-preemptive scheduling
  - A task, ones started, may not be stopped.

Static Cyclic Scheduling

- Generate activation times for each task instance:
  - These times determine the task activations over a (hyper)period T^h.
  - This sequence of activations is repeated in a cyclic manner.
    - If all tasks have the same period $T \Rightarrow T^h = T$.
    - If the tasks have different periods $T_1, T_2, ..., T_n \Rightarrow T^h = \text{LCM}(T_1, T_2, ..., T_n)$.

Static Cyclic Scheduling: Example

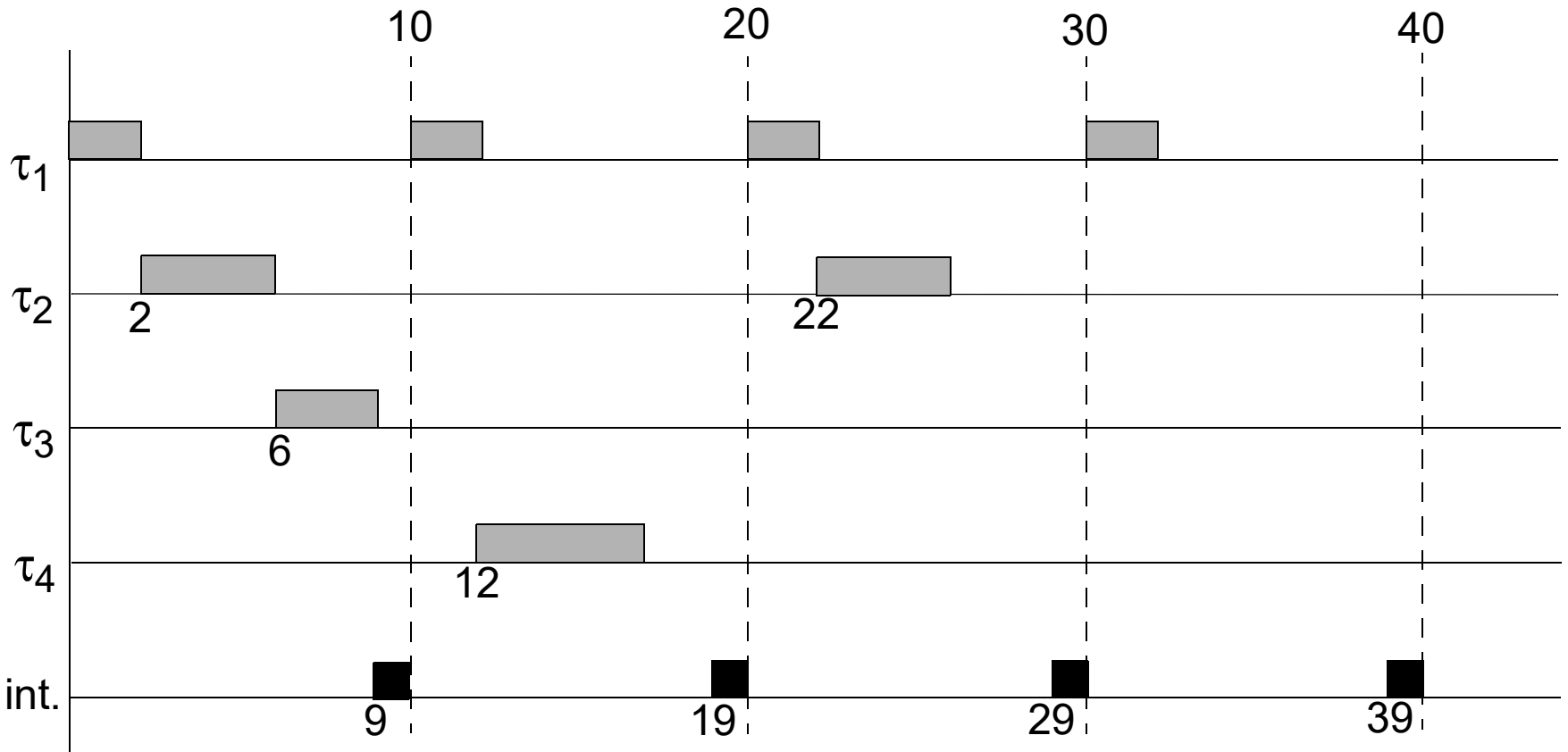
	Period=deadline	Worst case comp. time
τ_1	10	2
τ_2	20	4
τ_3	40	3
τ_4	40	5
System management	10	1

T^h = LCM(*10*, *20*, *40*) = 40

Static Cyclic Scheduling: Example

	Period=deadline	Worst case comp. time
τ_1	10	2
τ_2	20	4
τ_3	40	3
τ_4	40	5
System management	10	1

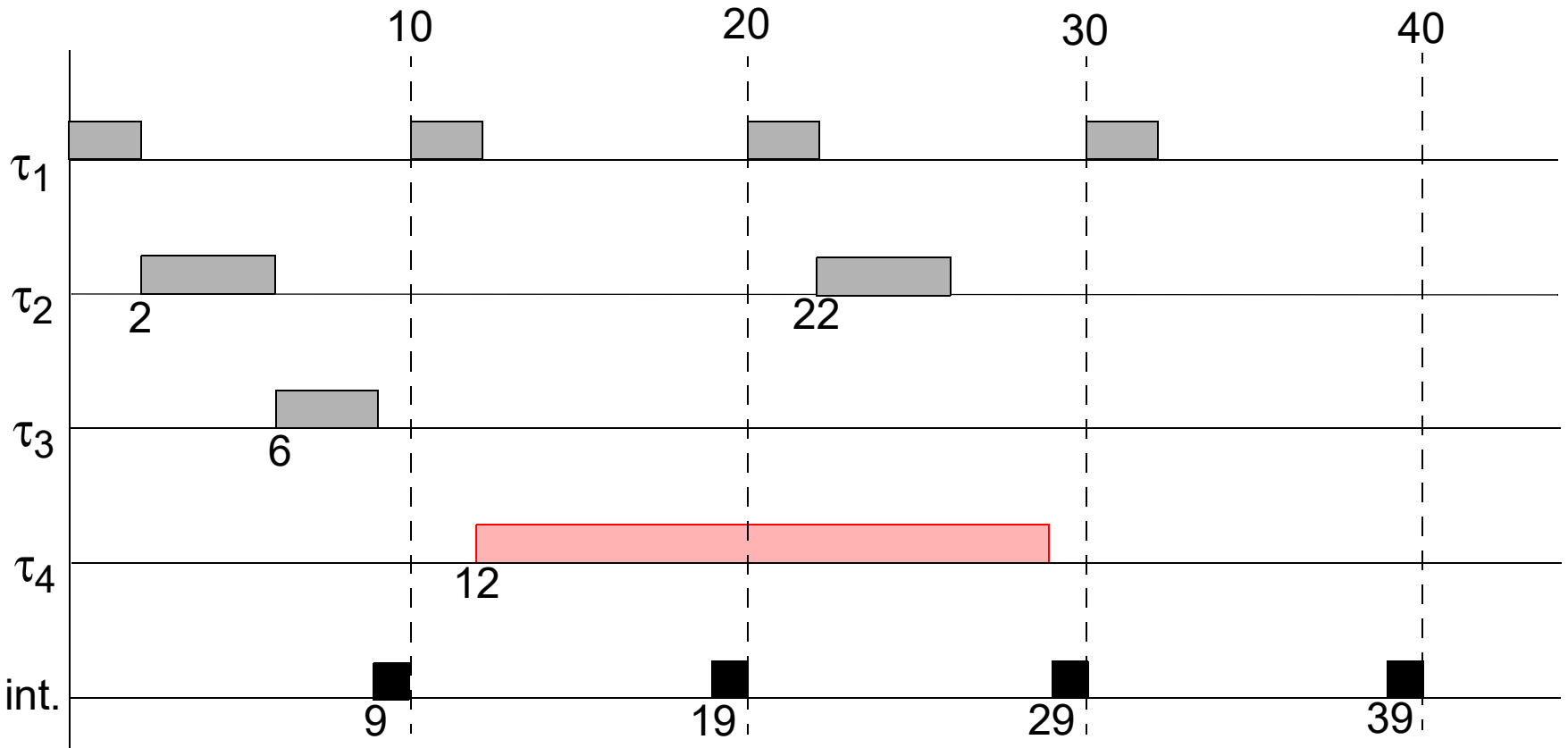
T^h = LCM(*10*, *20*, *40*) = 40



Static Cyclic Scheduling: Example

	Period=deadline	Worst case comp. time
τ_1	10	2
τ_2	20	4
τ_3	40	3
τ_4	40	17
System management	10	1

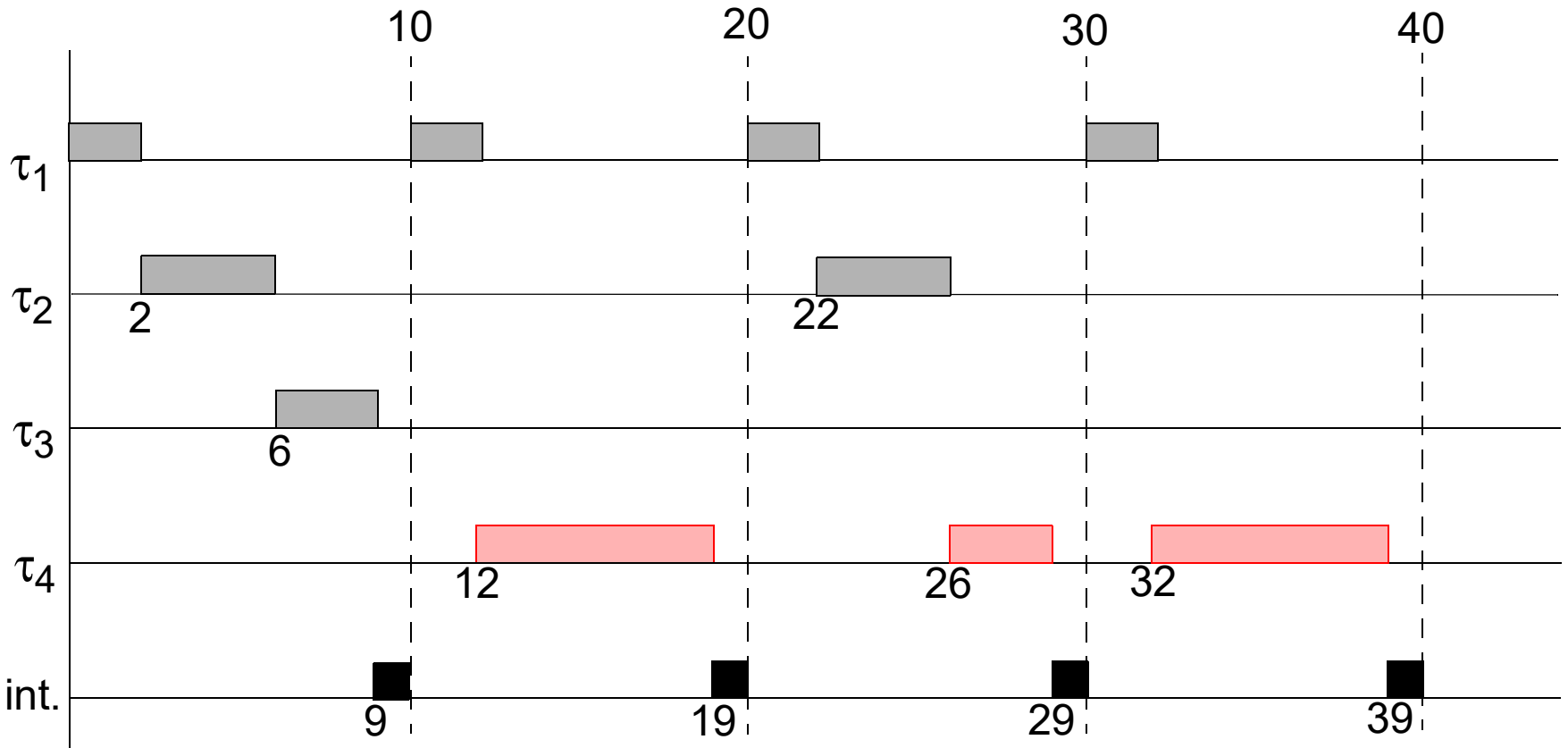
T^h = LCM(*10*, *20*, *40*) = 40



Static Cyclic Scheduling: Example

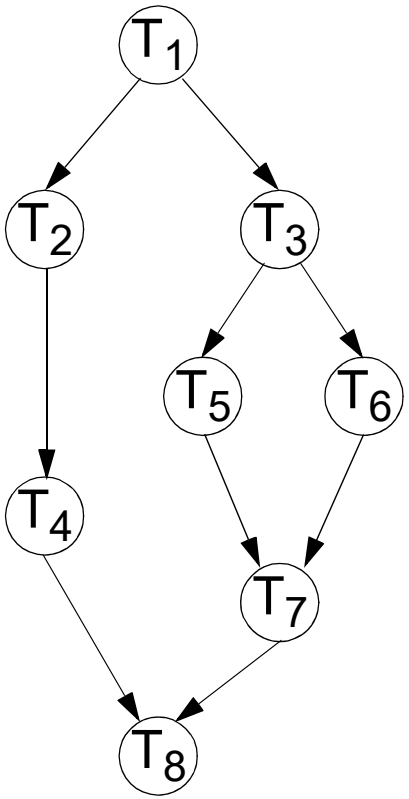
	Period=deadline	Worst case comp. time
τ_1	10	2
τ_2	20	4
τ_3	40	3
τ_4	40	17
System management	10	1

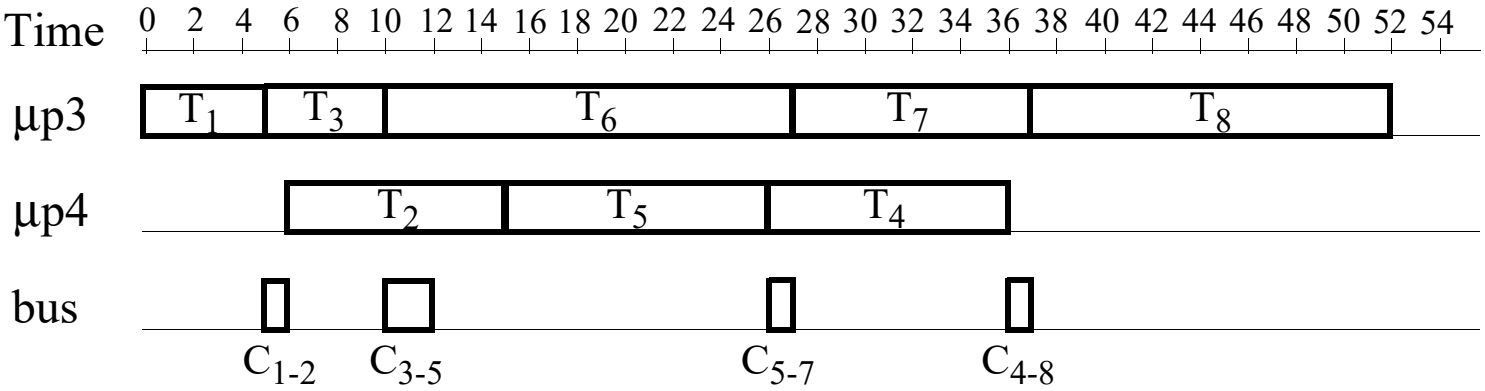
T^h = LCM(*10*, *20*, *40*) = 40



Static Cyclic Scheduling

- Often we have to schedule data dependent tasks; the platform may consist of several processor nodes. Example earlier lecture:





What is Good with Static Cyclic Scheduling?

- High predictability
- Easy to debug
- Low execution time overhead (not much to do for the real-time kernel during execution time)

What is Bad with Static Cyclic Scheduling?

- Not flexible:
  - quality degrades rapidly if periods and execution times deviate from those predicted;
  - if new tasks are added, the whole schedule has to be regenerated.
- Urgent events (interrupts) are handled purely:
  - time slots are statically allocated for polling and handling such events.
- Very long hyper-periods have to be avoided:
  - the periods of individual tasks have to be adjusted;
    this can lead to artificially reduced periods $\Rightarrow$ artificially increased load
    $\Rightarrow$ waste of processor time.
- Tasks have to be “manually” split, in order to fit into available slots.

Priority Based Preemptive Scheduling

- No schedule (predetermined activation times) is generated off-line. Tasks are activated as response to events (e.g. arrival of a signal, message, etc.).

- At any given time the highest priority ready task is running.

  If several tasks are ready to be activated on a processor, the highest priority task will be executed.

- Tasks can be preempted at any moment.

  If a task becomes ready to be executed (the respective event has occurred), and it has a higher *priority* than the running task, the running task will be preempted and the new one will execute.

Priority Based Preemptive Scheduling

- No schedule (predetermined activation times) is generated off-line. Tasks are activated as response to events (e.g. arrival of a signal, message, etc.).

- At any given time the highest priority ready task is running.

  If several tasks are ready to be activated on a processor, the highest priority task will be executed.

- Tasks can be preempted at any moment.

  If a task becomes ready to be executed (the respective event has occurred), and it has a higher *priority* than the running task, the running task will be preempted and the new one will execute.

- Will the tasks meet their deadlines?

  Schedulability analysis tries to answer this question.

Schedulability Analysis

- As result of research in real-time systems, a mathematical apparatus has been developed or schedulability analysis.

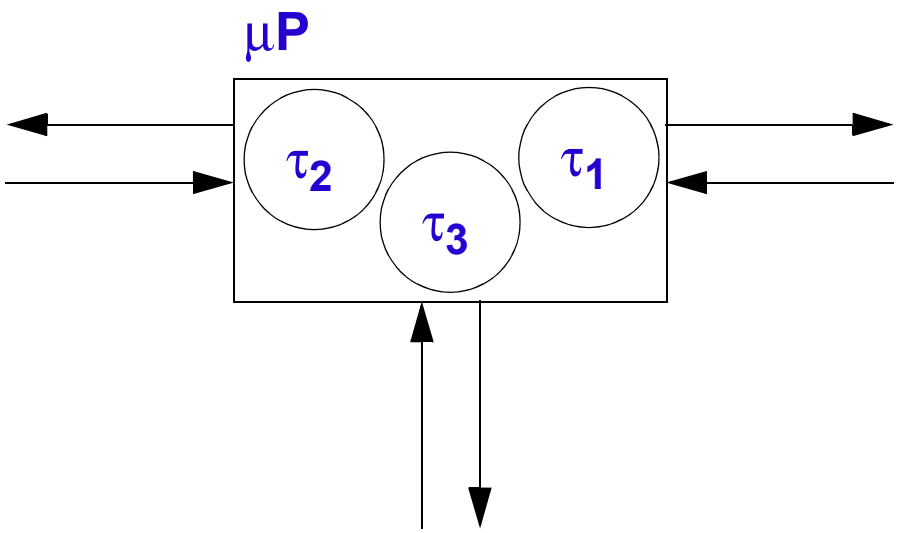
  These results are in form of conditions which can be used in order to check if a certain task set is schedulable (all tasks meet their deadline) or not.

- Schedulability analysis can be based on
  - Sufficient conditions (sometimes too pessimistic).
  - Necessary and sufficient conditions (sometimes difficult to apply).

- We will show some of the simpler formulas, only to give a “feeling”.

Schedulability Analysis

Example from Fö 6:



Task τ_1: Period T_1 = 100 μs; WCET C_1 = 40 μs

Task τ_2: Period T_2 = 30 μs; WCET C_2 = 10 μs

Task τ_3: Period T_3 = 25 μs; WCET C_3 = 10 μs

Does this work?
Can each of the tasks work at the required rate (period)?

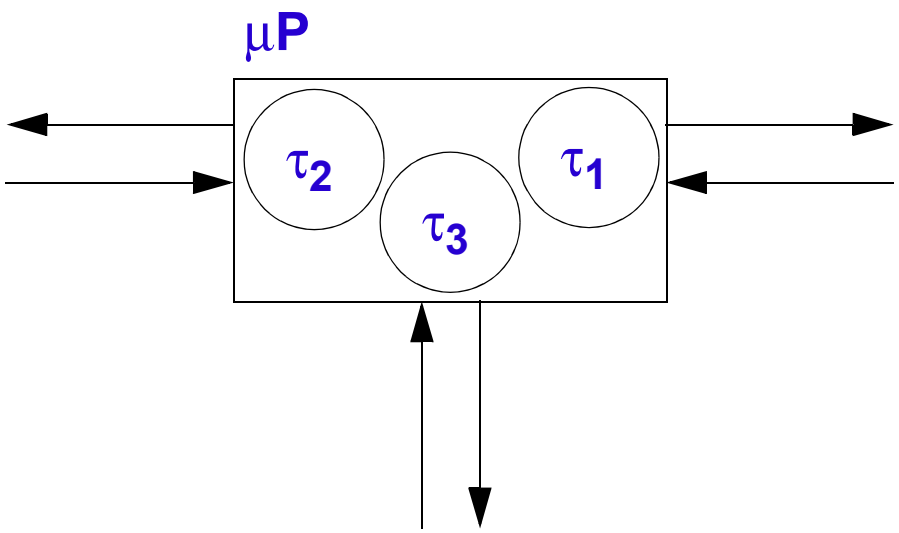
- τ_1 needs to run for 40 μs every 100 μs: 40% of CPU
- τ_2 needs to run for 10 μs every 30 μs: 33% of CPU
- τ_3 needs to run for 10 μs every 25 μs: 40% of CPU

Total: 113%

This will not work!

Schedulability Analysis

Example from Fö 6:



Task τ_1
- Period T_1 = 100 μs
- WCET C_1 = 40 μs

Task τ_2
- Period T_2 = 30 μs
- WCET C_2 = 10 μs

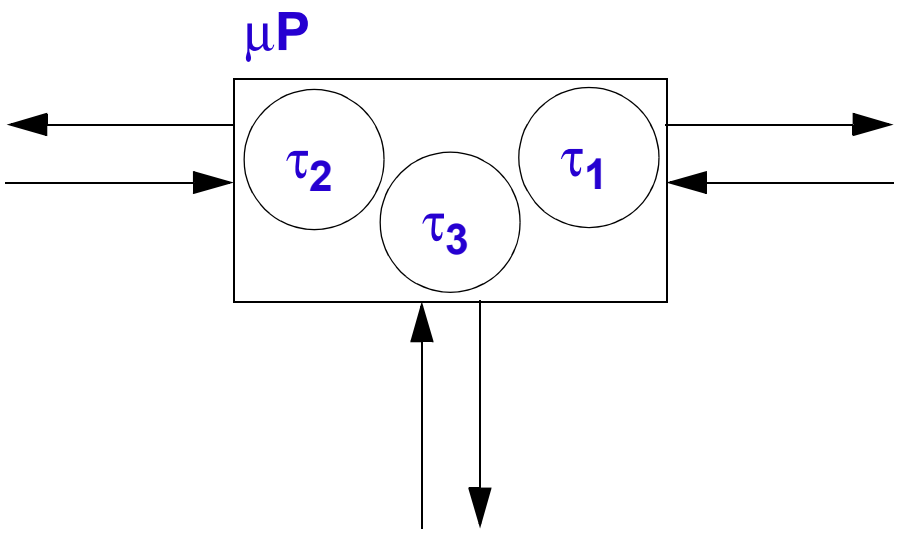
Task τ_3
- Period T_3 = 25 μs
- WCET C_3 = 10 μs

Does this work?
Can each of the tasks work at the required rate (period)?

But what if the Total Utilisation is less/equal 100% ?
Will it always work?

Schedulability Analysis

Example from Fö 6:



Task τ_1
- Period T_1 = 100 μs
- WCET C_1 = 40 μs

Task τ_2
- Period T_2 = 30 μs
- WCET C_2 = 10 μs

Task τ_3
- Period T_3 = 25 μs
- WCET C_3 = 10 μs

Does this work?
Can each of the tasks work at the required rate (period)?

But what if the Total Utilisation is less/equal 100% ?
Will it always work?

DEPENDS!

Schedulability Analysis

- A set of n tasks, with period T_i and worst case execution time c_i.

  Their deadline is equal with their period: $d_i = T_i$.

  At any moment the run-time monitor lets the task with the closest deadline run (EDF: Earliest Deadline First scheduling).

Schedulability Analysis

- A set of *n* tasks, with period T_i and worst case execution time c_i.

  Their deadline is equal with their period: $d_i = T_i$.

  At any moment the run-time monitor lets the task with the closest deadline run (EDF: Earliest Deadline First scheduling).

- With EDF a *sufficient and necessary* condition for the task set to be schedulable is that the total processor load is below/equal 100%:

$$\sum_{i=1}^{n} \frac{c_i}{T_i} \leq 1$$

Schedulability Analysis

- A set of *n* tasks, with period T_i and worst case execution time c_i.

  Their deadline is equal with their period: $d_i = T_i$.

  Task priorities are statically assigned to tasks, by the designer, according to their period: the task with shorter period gets the higher priority.
  (This is different from EDF!)

- A *sufficient (not necessary)* condition for the task set to be schedulable:

$$\sum_{i=1}^{n} \frac{c_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$$

Schedulability Analysis

- A set of *n* tasks, with period T_i and worst case execution time c_i.

  Arbitrary deadline: $d_i \leq T_i$ (This is different from the previous cases!)

  Task priorities are statically assigned, by the designer, *and can be arbitrary*.

Schedulability Analysis

- A set of *n* tasks, with period T_i and worst case execution time c_i.

  Arbitrary deadline: $d_i \leq T_i$ (This is different from the previous cases!)

  Task priorities are statically assigned, by the designer, *and can be arbitrary*.

- The response time for each task can be calculated based on the following recurrence relation:

$$r_i = c_i + \sum_{\forall k \in hp_i} \left\lceil \frac{r_i}{T_k} \right\rceil c_k$$

This is the interference from higher priority tasks.

- A necessary and sufficient condition for schedulability: $r_i \leq d_i$

Schedulability Analysis

- Schedulability conditions exist today that handle more general systems:
  - Deadlines which can be larger then the period
  - Tasks with share critical resources
  - Multiprocessors