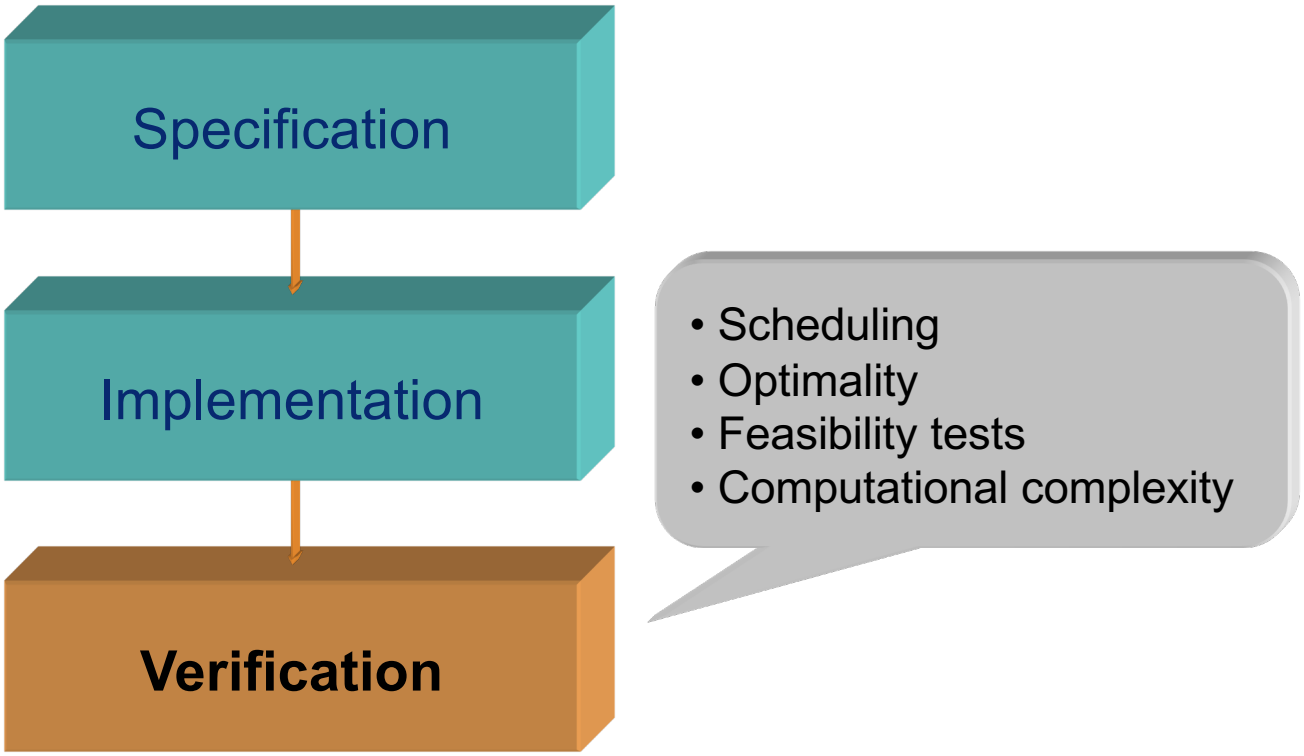
Real-Time Systems

Lecture #9

Professor Jan Jonsson

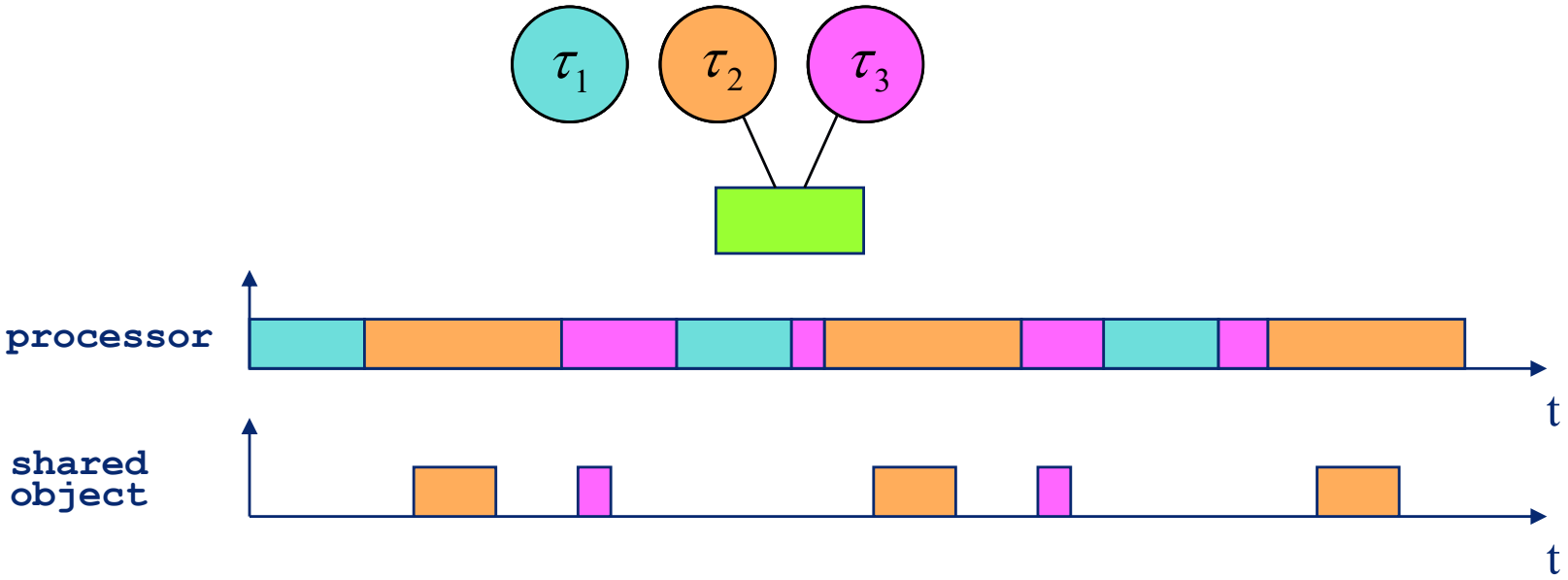
Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



Scheduling

A schedule is a reservation of spatial (e.g., processor, shared objects) and temporal (time) resources for a given set of tasks.



Scheduling

A scheduling algorithm is used for generating a schedule for a given set of tasks for a particular type of run-time system.

- The scheduling algorithm is implemented by a scheduler in the run-time system, that decides in what order the tasks should be executed.
- Note that the scheduler decides which task should be executed next, whereas the dispatcher is responsible for starting the selected task.

Scheduling

How is scheduling implemented?

- Cyclic executive:
  - The schedule is generated ”off-line” before the tasks becomes ready, sometimes even before the system is in mission.
  - The schedule consists of a time table, containing explicit start and completion times for each task instance, that controls the order of execution at run-time.
- Pseudo-parallel execution:
  - The schedule is generated ”on-line” as a side effect of tasks being executed, that is, when the system is in mission.
  - Ready tasks are sorted in a queue and receive access to the processor based on priority and/or time quanta (”round-robin”).

Scheduling

How is scheduling implemented? (cont'd)

- Cyclic executive:
  - The implementation of the scheduler is relatively simple because the next task is chosen with a table look-up.
  - However, the time table must be generated off-line (before the system is in mission) by a more advanced algorithm.
- Pseudo-parallel execution:
  - The implementation of the scheduler is more sophisticated because it consists of a decision algorithm that must be activated regularly (at each system event).
  - If shared resources are used the scheduler must also handle protocols for avoiding priority/deadline inversion.

Scheduling

When are scheduling decisions taken?

- Non-preemptive scheduling:
  - Scheduling decisions are taken when no task executes.
  - Mutual exclusion can be automatically guaranteed.
  - Corresponds to fundamental assumption in WCET analysis.
- Preemptive scheduling:
  - Scheduling decisions may be taken as soon as the system state changes (that is, even during an ongoing task execution).
  - Mutual exclusion may have to be guaranteed with semaphores (or similar primitives).
  - WCET analysis becomes more complicated, because the state in caches and pipelines will change at a task switch.

Scheduling

When are scheduling decisions taken? (cont'd)

- Myopic scheduling:
  - Scheduling algorithm only knows about tasks that are ready.
  - Scheduling decisions are only taken when system state changes.
  - On-line myopic scheduling is state-of-the-art in run-time systems.
- Clairvoyant scheduling:
  - Scheduling algorithm "knows the future"; that is, it knows in advance all the arrival times of all the tasks.
  - Scheduling decisions may be taken at any time, not necessarily when system state changes.
  - On-line clairvoyant scheduling is very difficult (often impossible) to realize in practice.

Scheduling

A schedule is said to be feasible if it fulfills all application constraints for a given set of tasks.

A set of tasks is said to be schedulable if there exists at least one scheduling algorithm that can generate a feasible schedule.

Scheduling

A scheduling algorithm is said to be optimal with respect to schedulability if it can always find a feasible schedule whenever any other scheduling algorithm can do so.

A scheduling algorithm is said to be optimal with respect to a performance metric if it can always find a schedule that maximizes/minimizes that metric value.

Feasibility tests

A feasibility test is used for deciding whether a set of tasks is feasible or not for a given scheduler.

Important characteristics of feasibility tests:

- Exactness
  - What conclusions can be drawn regarding feasibility based on the outcome of the test?
- Computational complexity
  - How long time does it take for the test to produce an outcome with a decision regarding feasibility?

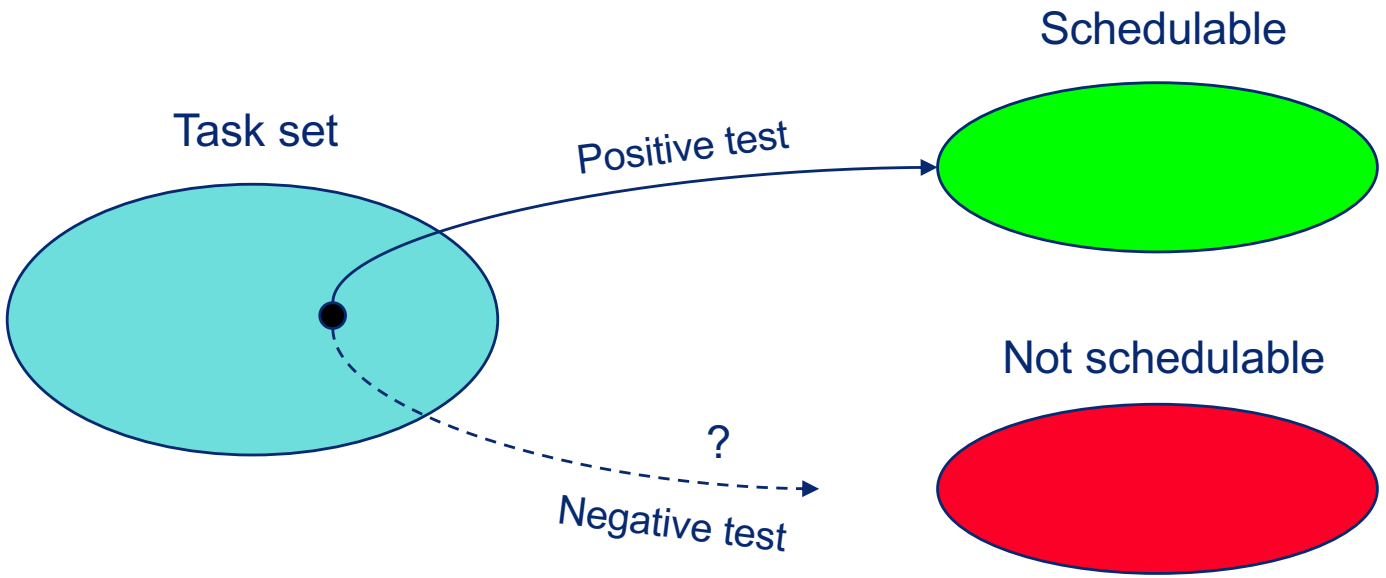
Feasibility tests

Exactness of a feasibility test

- The outcome of a feasibility test is binary:
  - Positive or Negative
  - True or False
  - Yes or No
- The conclusions that can be drawn depends on whether the test is:
  - Sufficient
  - Necessary
  - Exact (= sufficient and necessary)

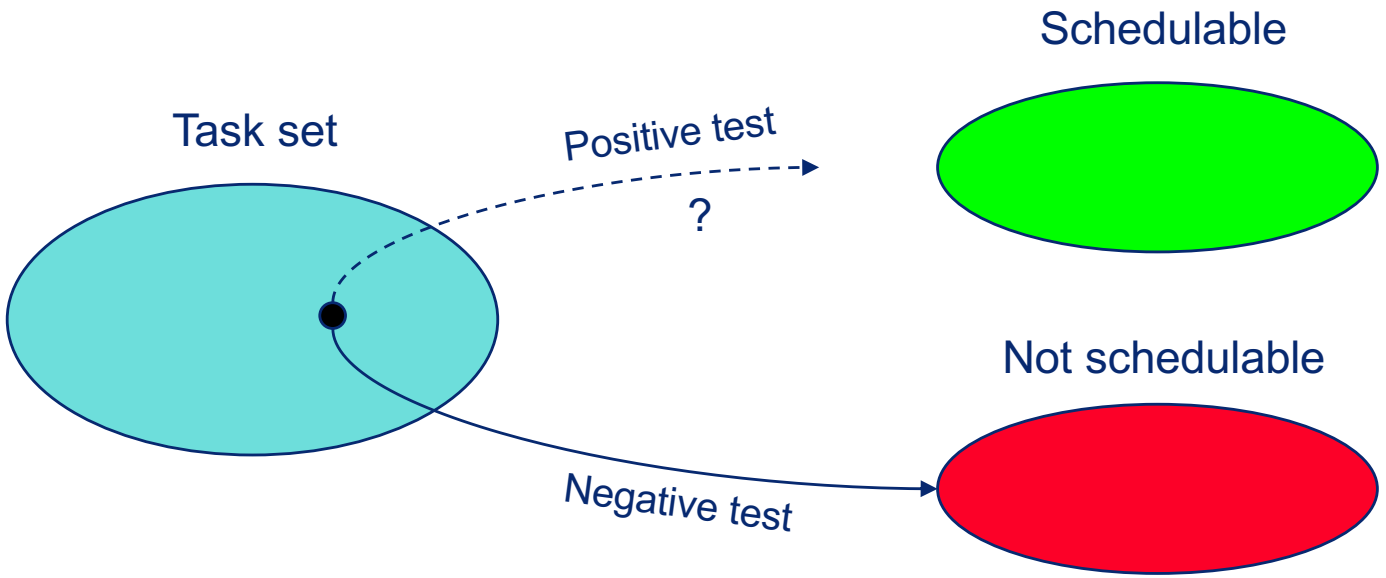
Feasibility tests

- A feasibility test is sufficient if it with a positive outcome shows that a set of tasks is definitely schedulable.
  - A negative outcome says nothing! A set of tasks can still be schedulable despite a negative outcome.



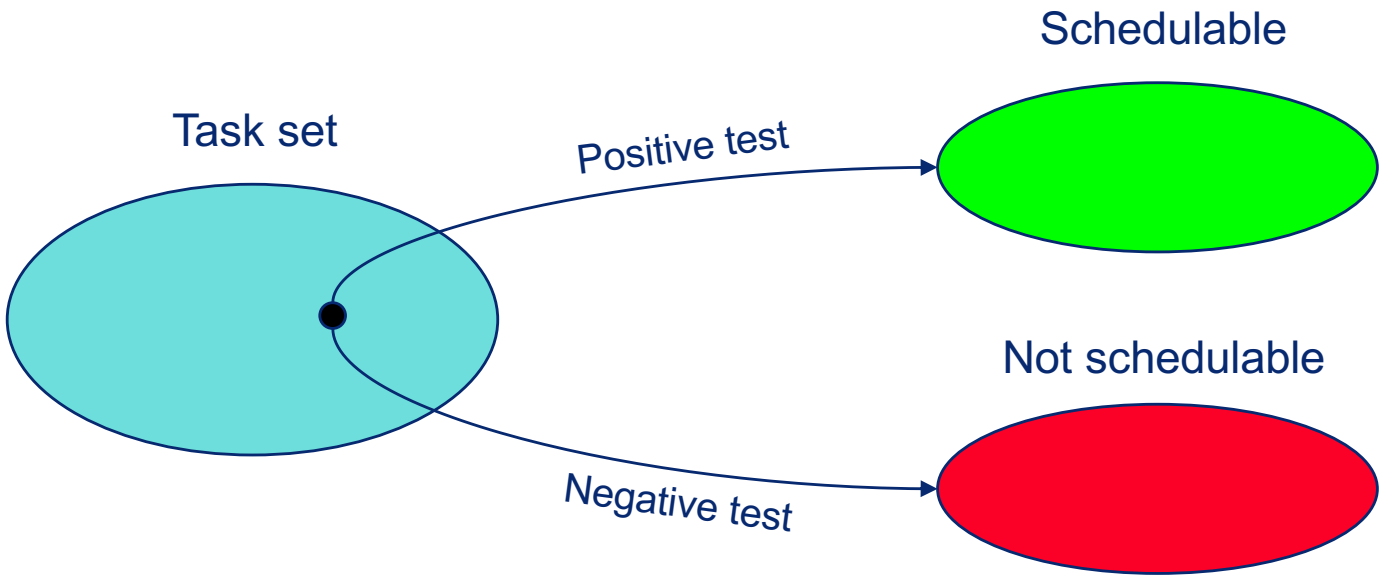
Feasibility tests

- A feasibility test is necessary if it with a negative outcome shows that a set of tasks is definitely not schedulable.
  - A positive outcome says nothing! A set of tasks can still be impossible to schedule despite a positive outcome.



Feasibility tests

- An exact feasibility test is both sufficient and necessary. If the outcome of the test is positive the set of tasks is definitely schedulable, and if the outcome is negative the set of tasks is definitely not schedulable.



Feasibility tests

Computational complexity of a feasibility test:

- For some schedulers the feasibility test can be done with polynomial time complexity.
  - These feasibility tests typically have relaxed assumptions regarding the task model, for example: independent tasks (with no shared resources) or implicit-deadline tasks (with deadline = period).
- For most schedulers the feasibility test cannot be done with polynomial time complexity.
  - These feasibility tests are either NP-complete problems, or have exponential time complexity (because all possible schedules must be considered in the worst case.)

Feasibility tests

What types of feasibility tests exist?

- Hyper period analysis (for any type of scheduler)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (static/dynamic priority scheduling)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (static priority scheduling)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (dynamic priority scheduling)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Feasibility tests

What types of feasibility tests exist?

- Hyper period analysis (exponential time complexity)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (polynomial time complexity)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (NP-complete problem)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (NP-complete problem)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Why NP-completeness matters

Assume that your boss gives you the following problem:

Find a good algorithm (method) for determining whether or not any given set of specifications for your company's new bandersnatch component can be met and, if so, find a good algorithm for constructing a design that meets those specifications.

The bandersnatch example is taken from "A Guide to the Theory of NP-Completeness" by M. R. Garey and D. S. Johnson

Why NP-completeness matters

Initial attempt:

Pull down your reference books and plunge into the task with great enthusiasm.

Some weeks later ...

Your office is filled with crumpled-up scratch paper, and your enthusiasm has lessened considerable because …

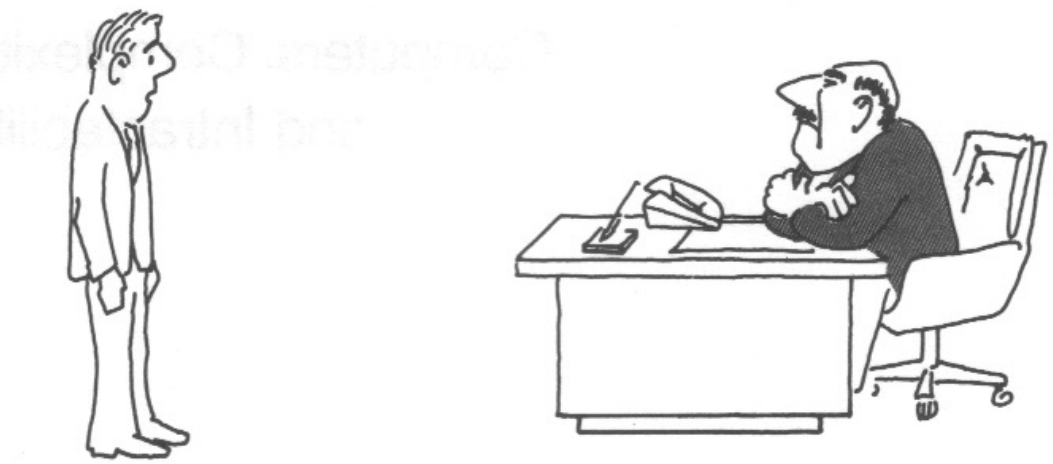
… the solution seems to be to examine all possible designs!

You now have a new problem:

How do you convey the bad information to your boss?

Why NP-completeness matters

Approach #1: Take the loser's way out

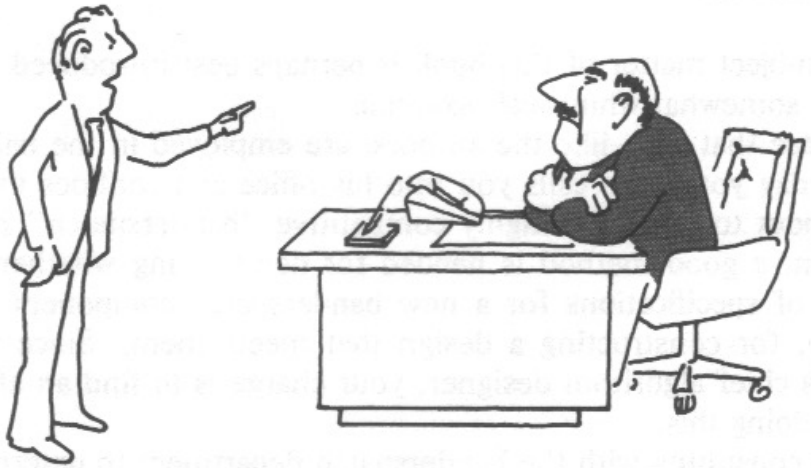


"I can't find an efficient algorithm, I guess I'm just too dumb."

Drawback: Could seriously damage your position within the company

Why NP-completeness matters

Approach #2: Prove that the problem is inherently intractable



"I can't find an efficient algorithm, because no such algorithm is possible!"

Drawback: Proving inherent intractability can be as hard as finding efficient algorithms. Even the best theoreticians have failed!

Why NP-completeness matters

Approach #3: Prove that the problem is NP-complete

"I can't find an efficient algorithm, but neither can all these famous people."

Advantage: This would inform your boss that it is no good to fire you and hire another expert on algorithms.

NP-complete problems

NP-complete problems:

Problems that are “just as hard” as a large number of other problems that are widely recognized as being difficult by algorithmic experts.

NP-complete problems can (most likely) only be solved by an exponential-time algorithm in the general case.

NP-complete problems

Problem:

- A general question to be answered

  Example: The “traveling salesman optimization problem”

Parameters:

- Free problem variables, whose values are left unspecified

  Example: A set of “cities” $C = \{c_1, ..., c_n\}$ and a “distance” $d(c_i, c_j)$ between each pair of cities c_i and c_j

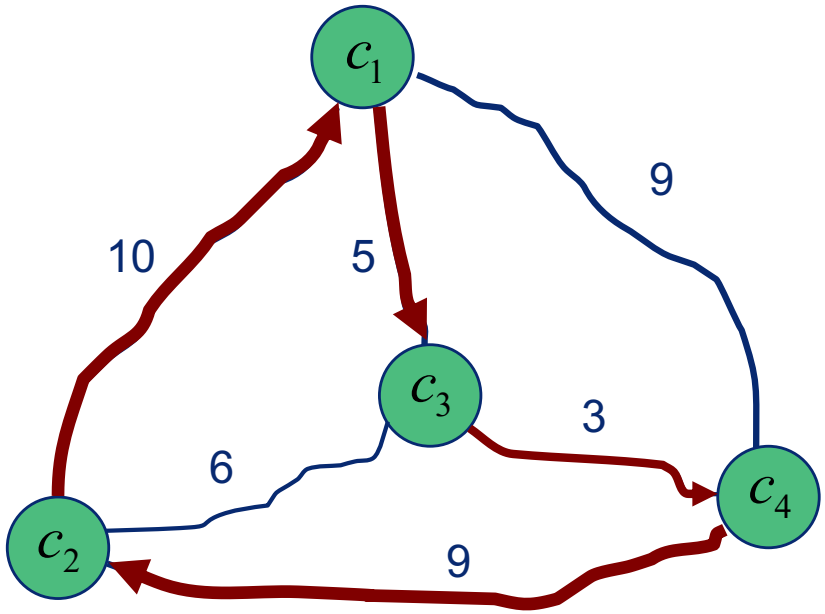
Instance:

- An instance of a problem is obtained by specifying particular values for all the problem parameters

  Example: $C = \{c_1, c_2, c_3, c_4\}, d(c_1, c_2) = 10, d(c_1, c_3) = 5, d(c_1, c_4) = 9,$
  $d(c_2, c_3) = 6, d(c_2, c_4) = 9, d(c_3, c_4) = 3$

NP-complete problems

The Traveling Salesman Optimization Problem:



Minimum “tour” length = 27

Minimize the length of the “tour” that visits each city in sequence, and then returns to the first city.

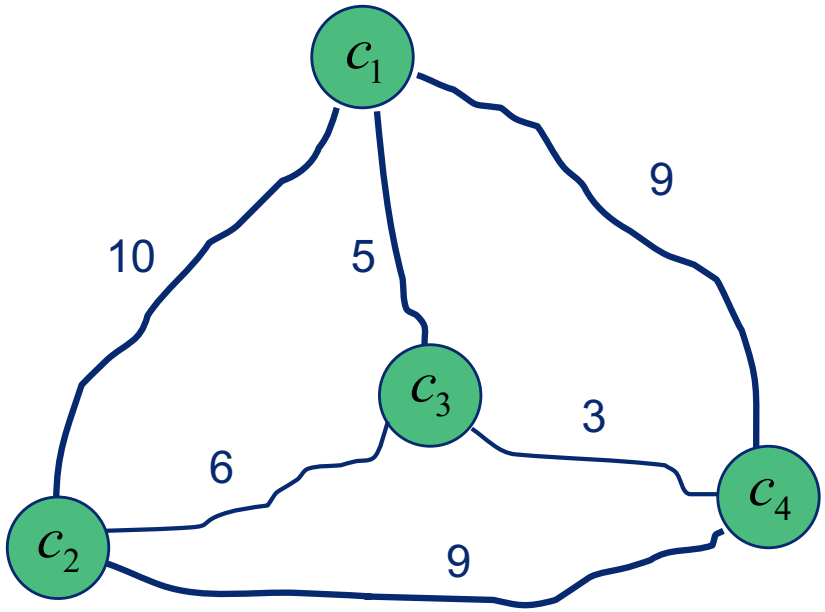
NP-complete problems

The theory of NP-completeness applies only to decision problems, where the solution is either a “Yes” or a “No”.

If an optimization problem asks for a solution that has minimum “cost”, we can associate with that problem a decision problem that includes a numerical bound B as an additional parameter and that asks whether there exists a solution having cost no more than B.

NP-complete problems

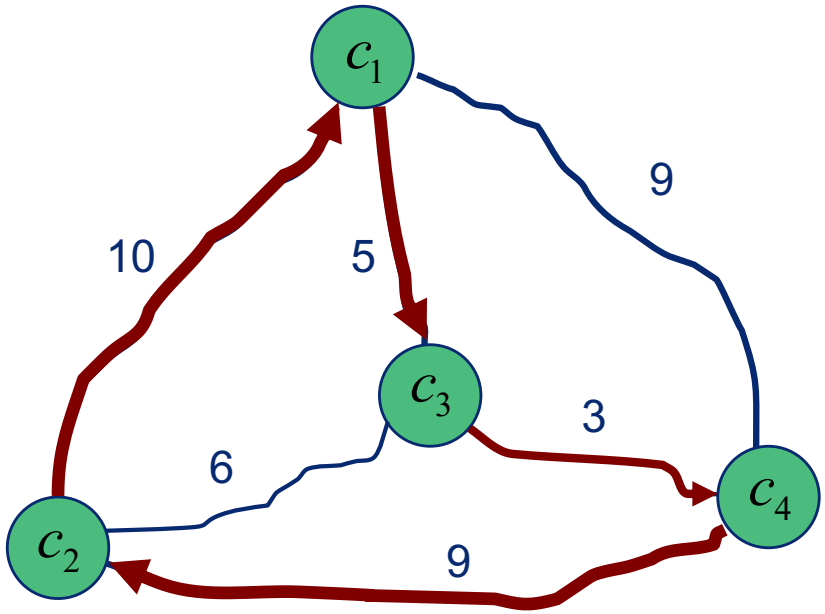
The Traveling Salesman Decision Problem:



Is there a "tour" of all the cities in C having a total length of no more than B?

NP-complete problems

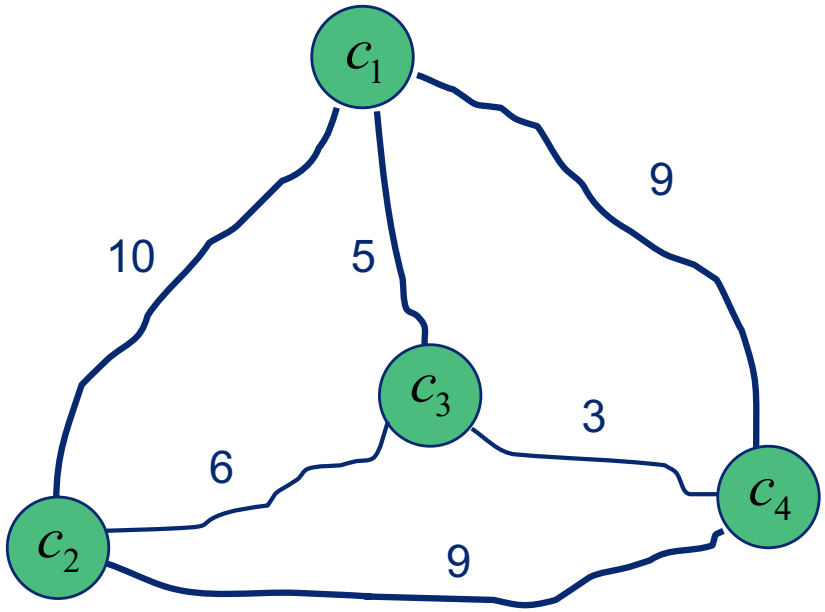
The Traveling Salesman Decision Problem:



Is there a "tour" of all the cities in C having a total length of no more than **30**? Yes! At least one (of length 27).

NP-complete problems

The Traveling Salesman Decision Problem:

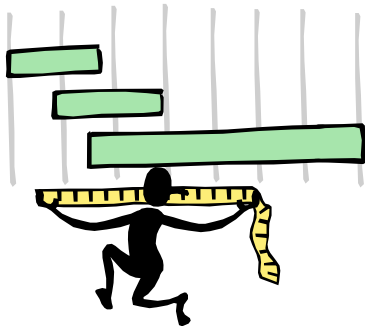


Is there a "tour" of all the cities in C having a total length of no more than **20**? No! The shortest is of length 27.

Intractability

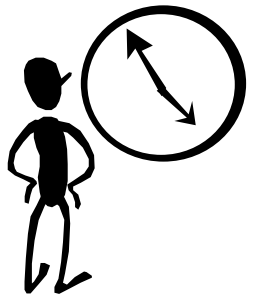
Input length:

- The number of information symbols (e.g. bits) needed for representing a problem instance of a given size.



Time-complexity function:

- Expresses an algorithm's worst-case run-time requirements giving, for each possible input length, the largest amount of time needed by the algorithm to solve a problem instance of that size.



Intractability

Polynomial-time algorithm:

- An algorithm whose time-complexity function is proportional to $p(n)$ for some polynomial function p, where n is the input length.

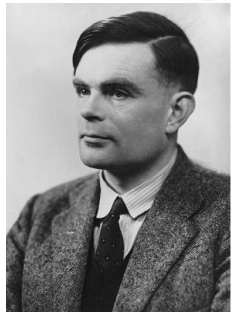
Exponential-time algorithm:

- Any algorithm whose time-complexity function cannot be bounded as above.

A problem is said to be intractable if it is so hard that no polynomial-time algorithm can possibly solve it.

Class P

Alan Turing



Deterministic algorithm: (Deterministic Turing Machine)

- Finite-state control:
  - The algorithm can pursue only one computation at a time
  - Given a problem instance I, some solution S is derived by the algorithm
  - The correctness of S is inherent in the algorithm

The class P is the class of all decision problems Π that can be solved by polynomial-time deterministic algorithms.

Class NP

Non-deterministic algorithm: (Non-Deterministic Turing Machine)

1. Guessing stage:
   - Given a problem instance I, some solution S is "guessed".
   - The algorithm can pursue an unbounded number of independent computational sequences in parallel.
2. Checking stage:
   - The correctness of S is verified in a normal deterministic manner

The class NP is the class of all decision problems Π that can be solved by polynomial-time non-deterministic algorithms.

NP-complete problems

Reducibility:

- A problem Π' is reducible to problem Π if, for any instance of Π', an instance of Π can be constructed in polynomial time such that solving the instance of Π will solve the instance of Π' as well.

A decision problem Π is said to be NP-complete if $\Pi \in$ NP and, for all other decision problems $\Pi' \in$ NP, Π' reduces to Π in polynomial time .

Reference only, not examined

Some original NP-complete problems

NP-completeness in practice

Pseudo-polynomial time complexity:

- Number problems
  - This is a special type of NP-complete problems for which the largest number (parameter value) in a problem instance is not bounded by the input length (size) of the problem.
- Number problems are often quite tractable
  - If the time complexity of a number problem can be shown to be a polynomial-time function of both the input length and the largest number, that number problem is said to have pseudo-polynomial time complexity.

(More details regarding tractability aspects of NP-complete problems will be given in Lecture #15)

Feasibility tests

What types of feasibility tests exist? (revisited)

- Hyper period analysis (exponential time complexity)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (polynomial time complexity)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (pseudo-polynomial complexity)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (pseudo-polynomial complexity)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

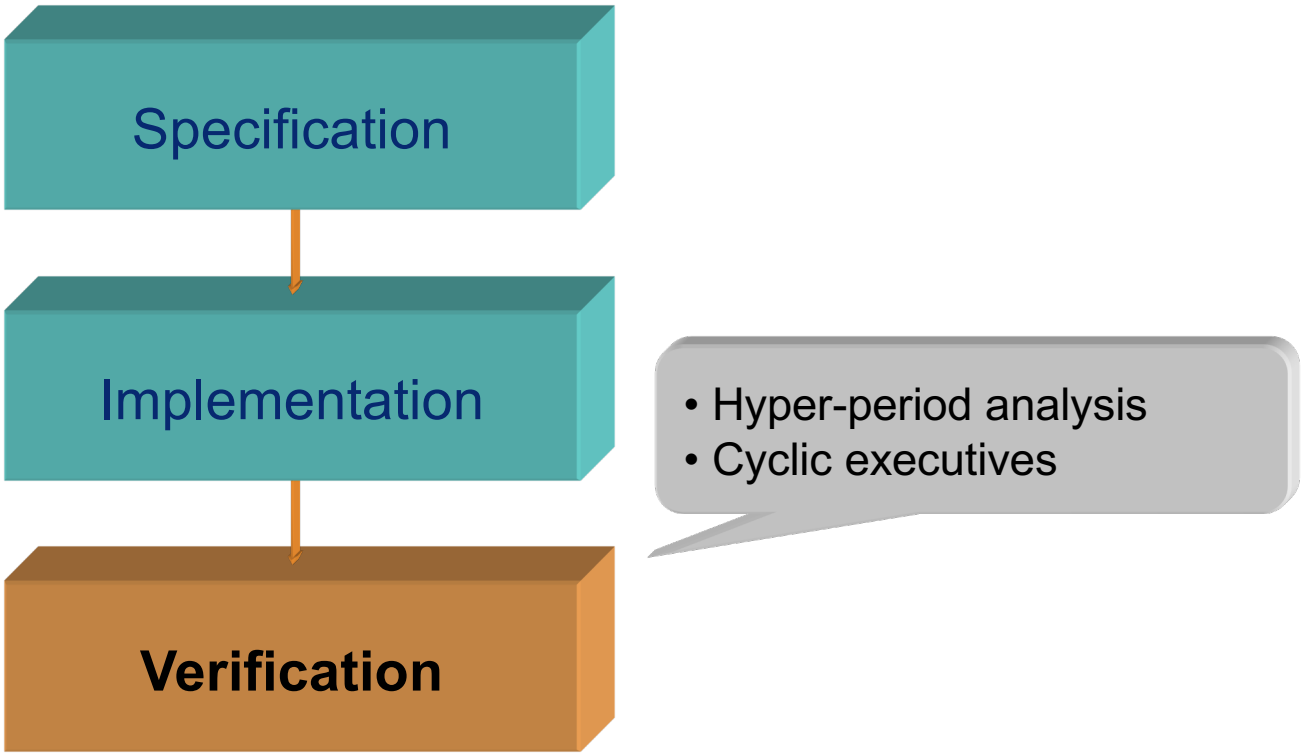
Real-Time Systems

Lecture #10

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



Feasibility tests

What types of feasibility tests exist?

- Hyper period analysis (for any type of scheduler)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (static/dynamic priority scheduling)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (static priority scheduling)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (dynamic priority scheduling)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Hyper period analysis

Motivation:

- When it is not obvious which feasibility analysis should be used for a given task set and a given scheduler it is always possible to generate a schedule by simulating the execution of the tasks, and then check feasibility for individual tasks.
- The schedule interval that is sufficient to investigate is related to the hyper period of the task set, that is, the least common multiple (LCM) of the task periods.

  NOTE: Unless the periods of all tasks are harmonically related (multiples of each other) hyper-period analysis will in general have an exponential time complexity.

Hyper period analysis

Schedule interval to investigate:

- For synchronous task sets: $\forall i, j : O_i = O_j$

  It is sufficient to investigate the interval $[0, P]$ where P is the hyper period of the task set.

- For asynchronous task sets: $\exists i, j : i \neq j, O_i \neq O_j$

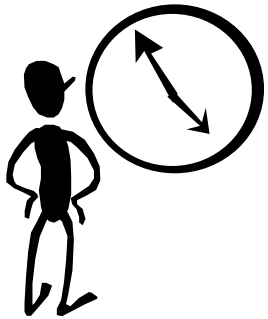
  It is sufficient to investigate the interval $[0, P]$ if no task instance that arrives within the interval executes beyond time P.

  In all other cases it is necessary to investigate more than one hyper period.

Cyclic executives

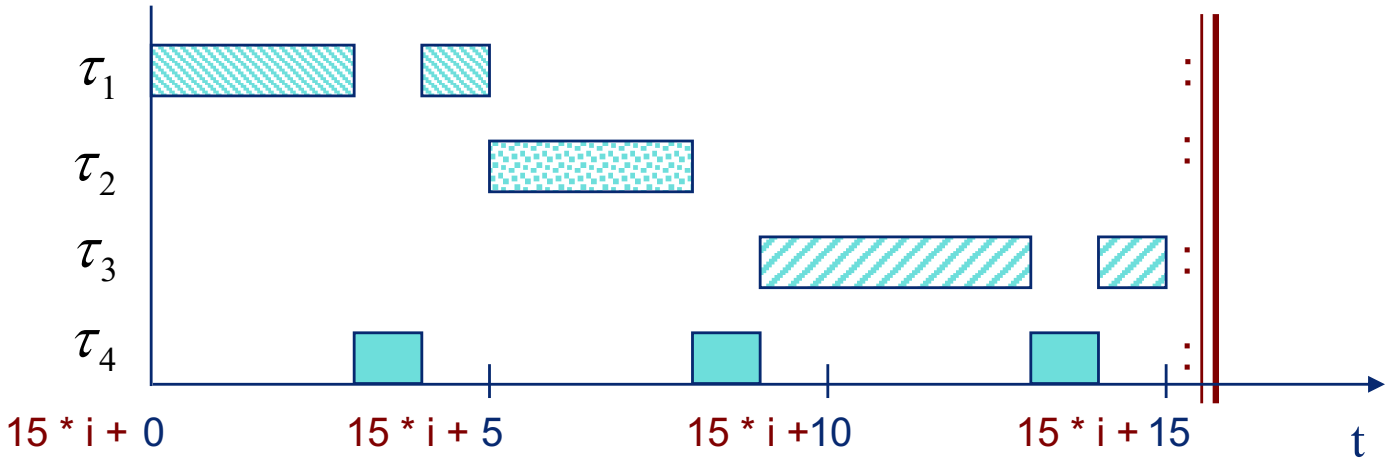
Because of its deterministic properties the cyclic executive is often the choice of scheduler in safety-critical real-time systems, such as automotive and aircraft applications.

Cyclic executives



General properties:

- Table-based schedule
- Feasibility test performed when generating table
- Schedule repeats itself (= “cyclic executive”)



Cyclic executives

General properties:

- Off-line schedule generation
  - Explicit start and finish times for each task are derived off-line, and chosen so that at most one task at a time requests access to the processor during run time.
- Mutual exclusion is handled explicitly
  - The schedule must be generated in such a way that a task switch is not made within a critical region (= no need for mutual exclusion support at run-time, e.g. mutex objects)
- Precedence constraints are handled explicitly
  - The schedule must be generated in such a way that specified task execution orderings are respected (= no need for task synchronization at run-time, e.g. semaphores)

Cyclic executives

Advantages:

- Communication between tasks is facilitated
  - The time instant when data becomes available is known
  - Task execution can easily be adapted to any existing time-slot network protocol (e.g., TTCAN, FlexRay).
- Low overhead for scheduling decisions
  - Everything is pre-planned: time table guides the run-time system
  - Feasibility test is done off-line during time table generation
- Task execution becomes very deterministic
  - Simplifies feasibility tests (compare finish time against deadline)
  - Simplifies software debugging (increased observability)
  - Simplifies fault tolerance (natural points in time for self control)

Cyclic executives

Disadvantages:

- Low flexibility (a.k.a. the ”Skalman” factor)
  - The run-time system cannot adapt its schedule to changes in the task set or in the system environment
- External events are not handled efficiently
  - Data from I/O-based events (interrupts) may not be consumed directly by a periodic task due to the pre-planned schedule, which could lead to long response times.
  - An external event with a short deadline must be handled by a task with short period, which may lead to resource waste
- Not so efficient for tasks with ”bad” periods
  - Tasks with mutually inappropriate periods give rise to large time tables, which may require more program code and/or data

Cyclic executives

How is the schedule generated?

- Simulation of pseudo-parallel execution:
  - Simulate a run-time system with a (myopic) priority-based scheduler and then ”execute” the tasks on that simulator.
  - Example: find a schedule by simulating a run-time system with the (dynamic priority) earliest-deadline-first scheduler.
- Exhaustive search:
  - Use an algorithm that searches for a feasible schedule by considering all possible execution orders for the tasks.
  - Example: use the well-known A* search algorithm to find a feasible (optimal or non-optimal) schedule.

If the simulated scheduler or search algorithm is optimal for the given system model a feasible schedule will be found whenever one exists.

Cyclic executives

How is the size of the time table restricted?

- Only cyclic schedules are considered:
  - Schedule is repeated with a cycle time ("hyper period") that is equal to the LCM ("least common multiple") of the task periods.
  - Tasks that are not periodic, or that have very long periods, can be handled by reserving time slots in the schedule for a "server" that can handle such special tasks when they arrive.
- Suitable task periods are chosen:
  - To obtain reasonably large time tables, the task periods should (if application allows) be adjusted to be multiples of each other.
  - Example:
    7, 13, 23 ms ⇒ cycle time 2093 ms (551 task instances), but
    5, 10, 20 ms ⇒ cycle time 20 ms (only 7 task instances)

Cyclic executives

How is the scheduler implemented?

- Use a circular queue that corresponds to the time table
  - Each element in the queue contains start and finish times for a certain task (or task segment in case of preemptive scheduling)
  - The elements in the queue are sorted by the start time
- Use clock interrupts
  - When a task starts executing, a real-time clock is programmed to generate an interrupt at the start time of the next (the one whose start time is closest in time) element in the queue.
  - When the interrupt occurs, the next element in the circular queue is fetched and the procedure is repeated.

Cyclic executives

How is a generated schedule visualized?

- Timing diagram
  - A timing diagram illustrates how each task executes at run-time, by explicitly denoting on a time line the start and finish times for each execution of a task (or a task segment).
- Execution semantics
  - In a timing diagram all task executions must run to completion, regardless of whether the task misses its deadline or not. This also applies when simulating the pseudo-parallel execution of tasks in a priority-based run-time system.

    A run-time system normally does not terminate the execution of a task that has missed a deadline, because it is assumed that the system designer has made sure that a task will never miss a deadline at run-time by means of schedulability analysis.

Cyclic executives

Programming with the TinyTimber kernel (p. 17):

```
int main() {
   INSTALL( &sonar, echo, IRQ_ECHO_DETECT );
   return TINYTIMBER( &sonar, tick, 0 );
}
```

TinyTimber uses both deadlines and baselines as input to its scheduling algorithm, although it is actually only the deadlines that pose any real challenge to the scheduler. However, missed deadlines are not trapped at run-time; if such behavior is desired it must be programmed by means of a separate watchdog task.

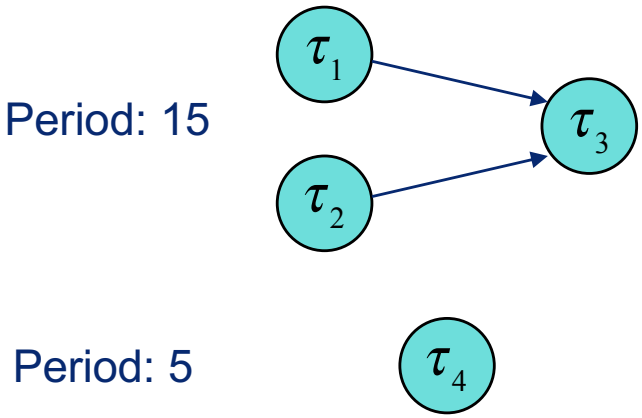
Judging whether a TinyTimber program will meet all its deadlines at run-time is an interesting problem, that can only be solved using a separate schedulability analysis and known worst-case execution times for all methods. That topic, however, is beyond the scope of the present text.

7 Summary of the TinyTimber interface

- `#include "TinyTimber.h"`

Example: simulating EDF

Problem: Assume a system with tasks and precedence constraints according to the figure below. Timing constraints for the tasks are given in the table. Generate a cyclic schedule for these tasks by simulating preemptive earliest-deadline-first (EDF) scheduling.



Task	C_i	O_i	D_i	T_i
τ_1	4	0	7	15
τ_2	3	0	12	15
τ_3	5	0	15	15
τ_4	1	3	1	5

Lecture #11 – blackboard scribble

Policy	Priority
RM	$\sim 1/T_i$
DM	$\sim 1/D_i$
WM	$\sim C_i/T_i$
SM	$\sim 1/(T_i - C_i)$

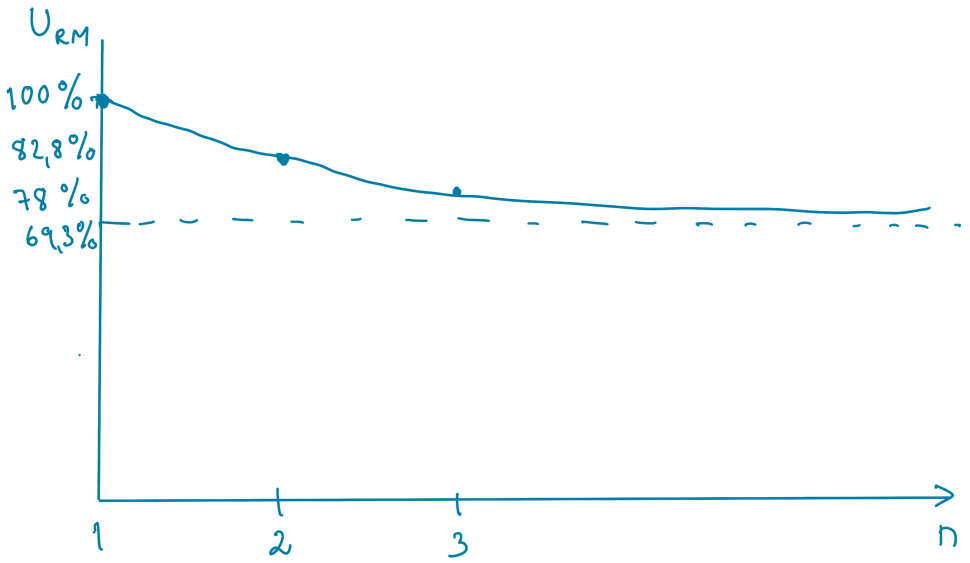
Policy	Priority
EDF	$\sim \frac{1}{\text{abs. deadline}}$
LLF	$\sim \frac{1}{\text{abs. deadline} - \text{remain. exec. time}}$

When $n \to \infty$

$$U_{RM} = n\left(\underbrace{2^{1/n} - 1}_{\to 0}\right)$$

$n \to \infty$, $2^{1/n} - 1 \to 1$

$$\lim_{n \to \infty} n\left(2^{1/n} - 1\right) = \ln 2$$



Lecture #11 – blackboard scribble

Task	C_i	O_i	T_i
τ_1	1	0	3
τ_2	1	0	4
τ_3	1	0	5

a) The utilization U of the system is

$$U = \sum_{i=1}^{3} \frac{C_i}{T_i} = \frac{1}{3} + \frac{1}{4} + \frac{1}{5} \approx 0{,}783$$

b) The utilization bounds U_{RM} and U_{EDF} are:

$U_{RM} = n\left(2^{1/n} - 1\right) = 3\left(2^{1/3} - 1\right) \approx 0{,}780$ $\quad U > U_{RM}$ The test fails!

$U_{EDF} = 1$ $\quad U < U_{EDF}$ The test succeeds!

c) For EDF: Since $U < U_{EDF}$ the task set is schedulable

For RM: Since $U > U_{RM}$ and the test is only sufficient, we cannot yet determine if the task set is schedulable or not.

(So, how do we do that?)

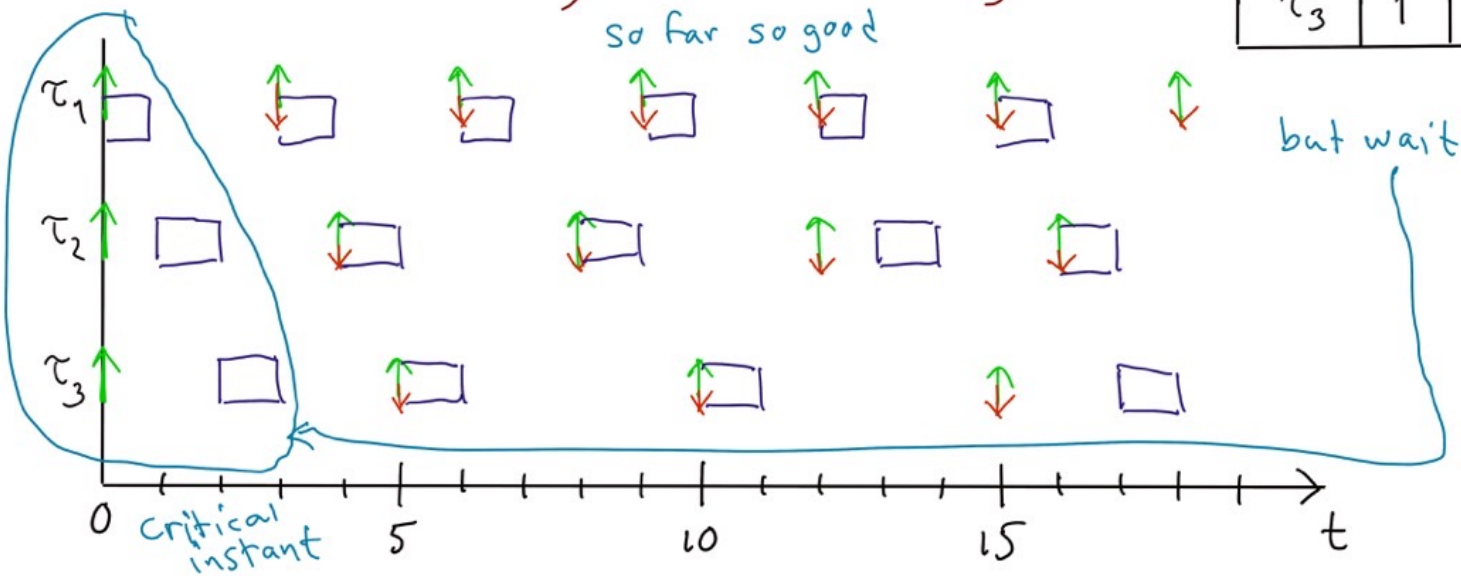
Lecture #11 – blackboard scribble

we learned that hyper period analysis can always be used.

How long is the hyper period? LCM{3,4,5} = 60

Simulate RM scheduling to check feasibility:

Task	C_i	O_i	T_i	
τ_1	1	0	3	H
τ_2	1	0	4	M
τ_3	1	0	5	L



Lin and Layland said that if the task set is schedulable at the critical instant (i.e. at t=0), then the task set is also schedulable in all other cases. Hence, it is enough to show that the first instance of each task meets its deadline!

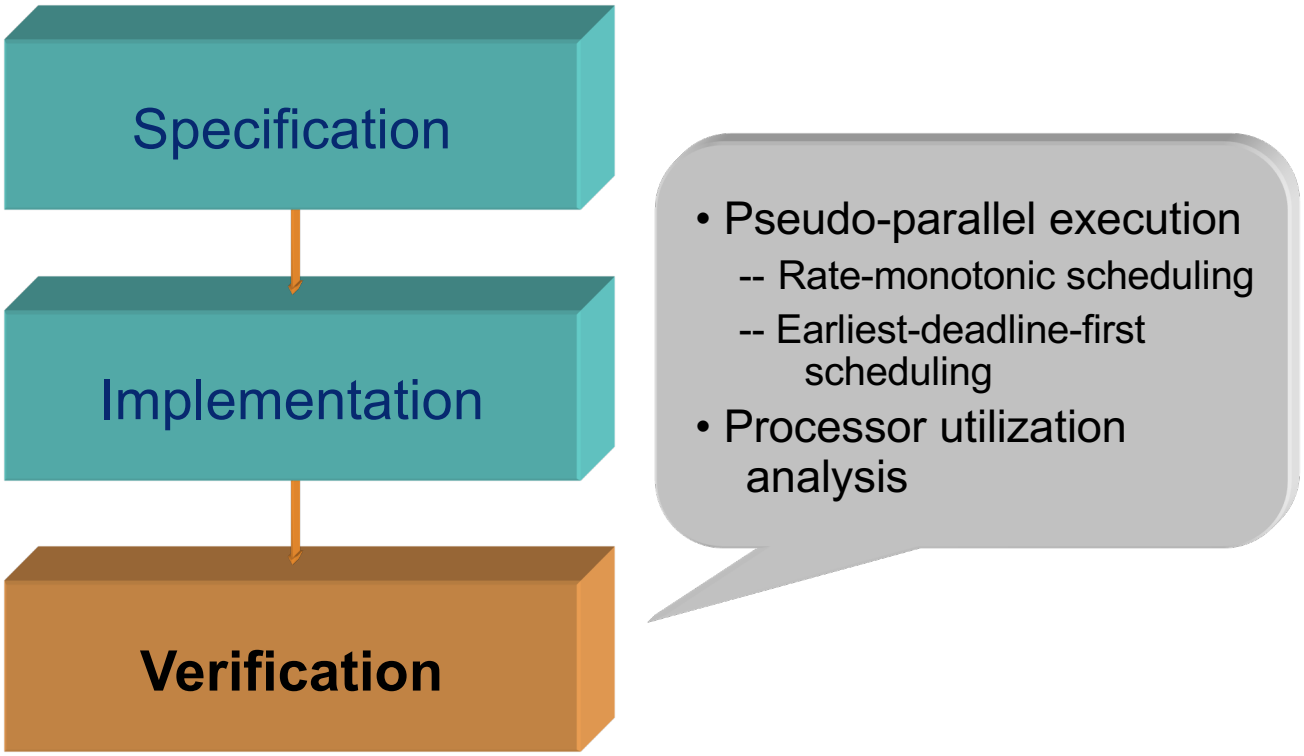
Real-Time Systems

Lecture #11

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



Pseudo-parallel execution

General properties:

- On-line schedule generation
  - Schedule determined by run-time behavior controlled by priorities or time quanta to resolve access to the processor
  - Feasibility must be tested off-line by predicting run-time behavior
- Mutual exclusion must be handled on-line
  - Support for mutual exclusion needed in run-time system (e.g., mutex objects, disabling of interrupts)
- Precedence constraints must be handled on-line
  - Dependent tasks must synchronize using semaphores, or by adding suitable time offsets

Pseudo-parallel execution

Advantages:

- High flexibility
  - Schedule can easily adapt to changes in the task set of the system environment, e.g. new tasks can be added dynamically
- External events are handled efficiently
  - I/O-based events handled via interrupt which activates a task immediately through call-back functionality
- Efficient for different types of tasks
  - Sporadic tasks can be easily supported, via suitable priority assignment
  - Scheduling algorithms are often optimal

Pseudo-parallel execution

Disadvantages:

- Complicates communication between tasks
  - Exact time of data availability is not known in advance, which requires extra synchronization between tasks
  - Task execution is difficult to adapt to existing time-slot-based network protocols (but works well with many priority-based network protocols, e.g. CAN and Token Ring)
- Task execution becomes non-deterministic
  - Temporary deviations (”jitter”) in task periodicity may occur
  - Exact feasibility tests often have high time complexity
  - Low observability (difficult to debug)

Pseudo-parallel execution

How is task scheduling done?

- Using static or dynamic priorities:
  - Ready tasks are stored in a queue, sorted by priority
  - At scheduling decisions, the task with highest priority is selected
- Using time quanta: (”round-robin”)
  - Ready tasks are stored in a circular FIFO queue
  - Each task gets access to the processor for a certain time interval (quantum); real-time clock is used for interrupting the execution
  - New scheduling decisions can be taken sooner if the executing task terminates or gets blocked

In this course, we only study pseudo-parallel execution using task priorities.

Pseudo-parallel execution

How are task priorities assigned?

- Static assignment:
  - Rate-monotonic (RM) scheduling
  - Deadline-monotonic (DM) scheduling
  - Weight-monotonic (WM) scheduling
  - Slack-monotonic (SM) scheduling
- Dynamic assignment:
  - Earliest-deadline-first (EDF) scheduling
  - Least-laxity-first (LLF) scheduling

In this course, we only study rate-monotonic, deadline-monotonic and earliest-deadline-first scheduling.

Pseudo-parallel execution

How is the scheduler implemented?

- Use a queue for the ready tasks
  - The elements in the queue (i.e., the tasks) are sorted according to task priorities; if multiple tasks have equal priority, the sorting is arbitrary (e.g., FIFO)
- The queue is updated at external or internal events
  - An external event: for example, an I/O unit generates an interrupt because data has become available at a sensor
  - An internal event: for example, a system timer generates an interrupt because a certain point in time has been reached
- Run tasks to completion
  - The scheduler normally does not terminate task executions that miss their deadlines; it is assumed that schedulability analysis has been used to verify timing correctness.

Pseudo-parallel execution

Programming with the TinyTimber kernel (p. 17):

```
int main() {
   INSTALL( &sonar, echo, IRQ_ECHO_DETECT );
   return TINYTIMBER( &sonar, tick, 0 );
}
```

TinyTimber uses both deadlines and baselines as input to its scheduling algorithm, although it is actually only the deadlines that pose any real challenge to the scheduler. However, missed deadlines are not trapped at run-time; if such behavior is desired it must be programmed by means of a separate watchdog task.

Judging whether a TinyTimber program will meet all its deadlines at run-time is an interesting problem, that can only be solved using a separate schedulability analysis and known worst-case execution times for all methods. That topic, however, is beyond the scope of the present text.

7 Summary of the TinyTimber interface

- `#include "TinyTimber.h"`

Rate-monotonic scheduling

Properties:

- Uses static priorities
  - Priority is determined by task frequency (rate): the task with the highest rate (= shortest period) receives highest priority
- Theoretically well-established
  - Sufficient feasibility test can be performed in linear time (under certain simplifying assumptions)
  - Exact feasibility test is an NP-complete problem (pseudo-polynomial time with response-time analysis)
  - RM is optimal among all scheduling algorithms that use static task priorities for implicit-deadline tasks (with $D_i = T_i$) (shown by C. L. Liu and J. W. Layland in 1973)

Earliest-deadline-first scheduling

Properties:

- Uses dynamic priorities
  - Priority is determined by how critical the task is at a given point in time: the task whose absolute deadline is earliest in time receives highest priority
- Theoretically well-established
  - Exact feasibility test can often be performed in linear time (under certain simplifying assumptions)
  - Exact feasibility test is in general an NP-complete problem (pseudo-polynomial time with processor-demand analysis)
  - EDF is optimal among all scheduling algorithms that use dynamic task priorities (shown by C. L. Liu and J. W. Layland in 1973)

Feasibility tests

What types of feasibility tests exist?

- Hyper period analysis (for any type of scheduler)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (static/dynamic priority scheduling)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (static priority scheduling)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (dynamic priority scheduling)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Processor utilization analysis

The utilization U for a set of periodic tasks is the fraction of the processor's capacity that is used for executing the tasks.

Since C_i / T_i is the fraction of processor time that is used for executing task τ_i the utilization for n tasks is

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

Simple feasibility test for RM

(Sufficient condition)

A sufficient condition for RM scheduling of synchronous task sets, based on the utilization U is

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

where n is the number of tasks.

This is a classic feasibility test presented by C. L. Liu and J. W. Layland in 1973.

Simple feasibility test for RM

(Sufficient condition)

Observe that it is possible to derive a conservative lower bound on utilization by letting $n \to \infty$.

$$\lim_{n \to \infty} n\left(2^{1/n} - 1\right) = \ln 2 \approx 0.693$$

This means that a set of tasks (regardless of number of tasks) whose total utilization does not exceed 0.693 is always schedulable with RM!

Simple feasibility test for RM

(Sufficient condition)

The test is valid under the following assumptions:

1. All tasks are independent
   - There must not exist dependencies due to precedence or mutual exclusion
2. All tasks are periodic or sporadic
3. All tasks have identical offsets (= synchronous task set)
4. Task deadline equals the period (= implicit-deadline tasks)
5. Task preemptions are allowed

Simple feasibility test for RM

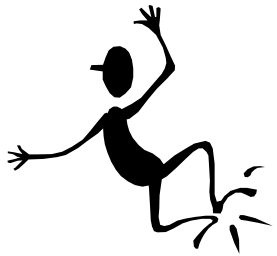
(Sufficient condition)

The proof of the test includes the following observation:

> The response time for a task is maximized at a special task-arrival pattern, referred to as the critical instant.

The feasibility test is derived using an analysis of this special case. It is shown that if the task set is schedulable for the critical instant case, it is also schedulable for any other case.

NOTE: For single-processor systems (assumed in this proof) the critical instant occurs when the analyzed task arrives at the same time as all tasks with higher priority.



Simple feasibility test for EDF

(Sufficient and necessary condition)

A sufficient and necessary condition for EDF scheduling of synchronous task sets, based on the utilization U is

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

where n is the number of tasks.

This is another classic feasibility test presented by C. L. Liu and J. W. Layland in 1973. The test is exact!

Simple feasibility test for EDF

(Sufficient and necessary condition)

The test is valid under the following assumptions:

1. All tasks are independent
   - There must not exist dependencies due to precedence or mutual exclusion
2. All tasks are periodic or sporadic
3. All tasks have identical offsets (= synchronous task set)
4. Task deadline equals the period (= implicit-deadline tasks)
5. Task preemptions are allowed

Example: RM/EDF scheduling

Problem: Assume a system with tasks according to the figure below. The timing properties of the tasks are given in the table. Consider scheduling the tasks using rate-monotonic (RM) and earliest-deadline-first (EDF) scheduling, respectively.

a) What is the utilization of the task set?

b) What do Liu & Layland's feasibility tests for RM and EDF say?

c) Are the tasks are schedulable using RM and EDF, respectively?

Task	C_i	O_i	T_i
τ_1	1	0	3
τ_2	1	0	4
τ_3	1	0	5

Lecture #12 – blackboard scribble

Calculation of response times:

	Task	C_i	D_i	T_i
H	τ_1	4	6	8
L	τ_2	3	14	16
M	τ_3	2	10	32

[τ_1 has highest priority (DM)]

$R_1 = C_1 = 4 \leq D_1 = 6 \Rightarrow$ ok!

[τ_3 has medium priority (DM)]

Note: ceiling function

$$R_3 = C_3 + \left\lceil \frac{R_3}{T_1} \right\rceil \cdot C_1 \qquad [\text{Assume } R_3^0 = C_3 = 2]$$

$$R_3^1 = 2 + \underbrace{\left\lceil \frac{2}{8} \right\rceil}_{1} \cdot 4 = 2 + 1 \cdot 4 = 6$$

$$R_3^2 = 2 + \underbrace{\left\lceil \frac{6}{8} \right\rceil}_{1} \cdot 4 = 2 + 1 \cdot 4 = 6$$

Convergence because $R_3^2 = R_3^1$

$\leq D_3 = 10 \Rightarrow$ ok!

Lecture #12 – blackboard scribble

[τ_2 has lowest priority (DM)]

$$R_2 = C_2 + \left\lceil \frac{R_2}{T_1} \right\rceil \cdot C_1 + \left\lceil \frac{R_2}{T_3} \right\rceil \cdot C_3 \quad [\text{Assume } R_2^0 = C_2 = 3]$$

$$R_2^1 = 3 + \left\lceil \frac{3}{8} \right\rceil \cdot 4 + \left\lceil \frac{3}{32} \right\rceil \cdot 2 = 3 + 1 \cdot 4 + 1 \cdot 2 = 9$$

$$R_2^2 = 3 + \left\lceil \frac{9}{8} \right\rceil \cdot 4 + \left\lceil \frac{9}{32} \right\rceil \cdot 2 = 3 + 2 \cdot 4 + 1 \cdot 2 = 13$$

$$R_2^3 = 3 + \left\lceil \frac{13}{8} \right\rceil \cdot 4 + \left\lceil \frac{13}{32} \right\rceil \cdot 2 = 3 + 2 \cdot 4 + 1 \cdot 2 = 13$$

Convergence

$\leq D_2 = 14 \Rightarrow$ ok!

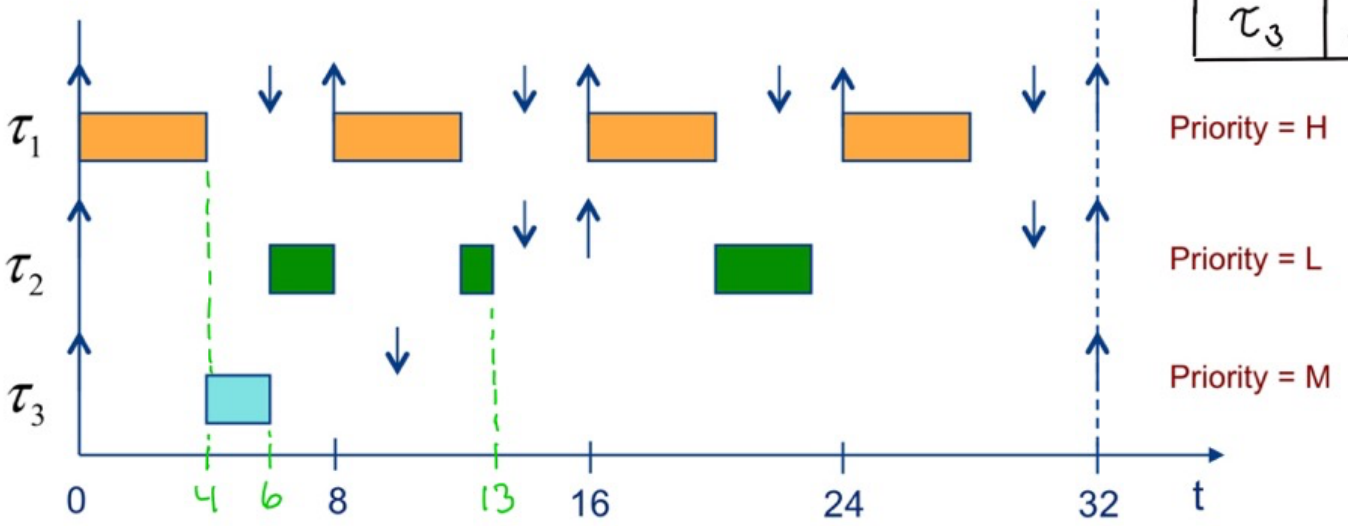
	Task	C_i	D_i	T_i
~~H~~	τ_1	4	6	8
L	τ_2	3	14	16
M	τ_3	2	10	32

All deadlines are met!

Lecture #12 – blackboard scribble

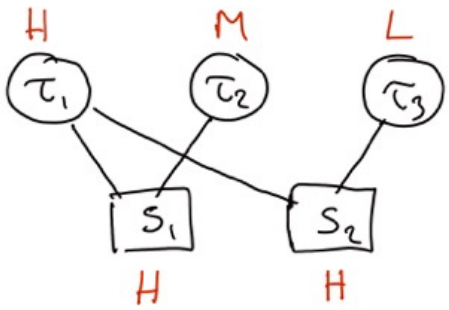
As we saw in the beginning of the lecture the resulting schedule looks like this:

Task	C_i	D_i	T_i
τ_1	4	6	8
τ_2	3	14	16
τ_3	2	10	32



Consequently, the analysis calculates worst-case response times that correspond exactly to the response times of the first instance of each task.

Lecture #12 – blackboard scribble

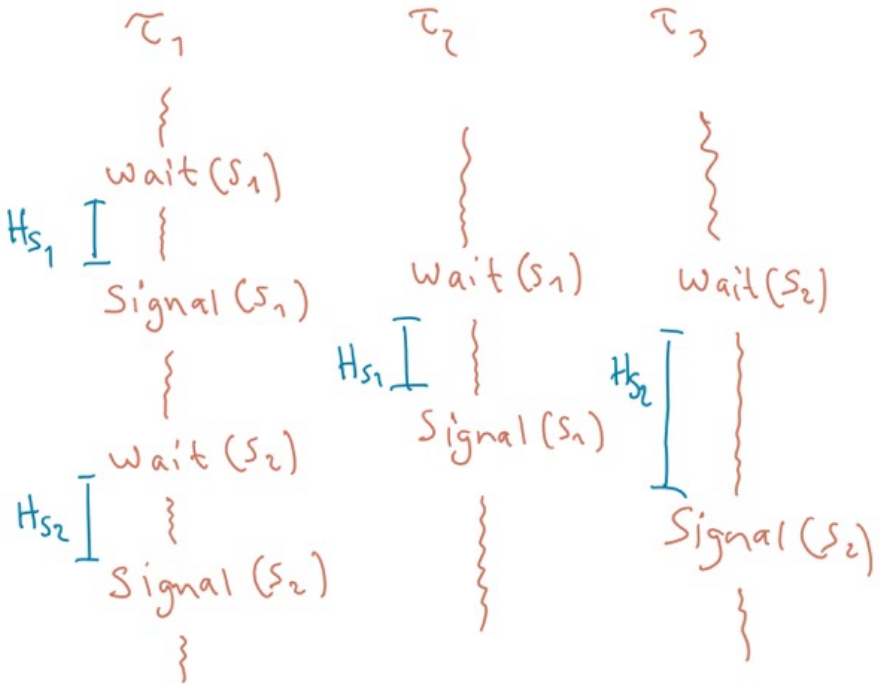


	Task	C_i	D_i	T_i	H_{S1}	H_{S2}
H	τ_1	2	4	5	1	1
M	τ_2	3	12	12	1	–
L	τ_3	8	24	25	–	2

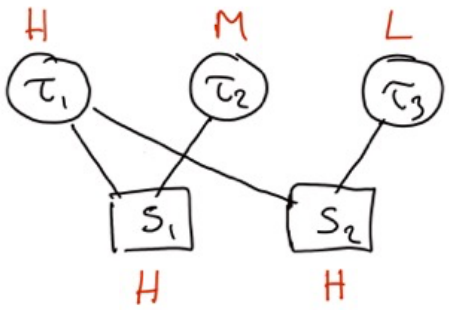
a) Ceiling priorities

S_1: $\max\{H, M\} = H$

S_2: $\max\{H, L\} = H$



Lecture #12 – blackboard scribble



Task	C_i	D_i	T_i	H_{S1}	H_{S2}
τ_1	2	4	5	1	1
τ_2	3	12	12	1	–
τ_3	8	24	25	–	2

b) Blocking factors:

Since both semaphores have highest priority ceiling (H) task τ_1 and τ_2 may be blocked by a task with lower priority regardless of which semaphore that lower-priority task uses.

$B_1 = \max\{1, 2\} = 2$ { τ_2 may use semaphore S_1 or τ_3 may use semaphore S_2

$B_2 = 2$ { τ_3 may use semaphore S_2; NOTE: τ_2 may be blocked although it does not use S_2

$B_3 = 0$ ← NOTE: lowest-priority task can never be blocked

Lecture #12 – blackboard scribble

c) Calculate response times!

Task	C_i	D_i	T_i	H_{s1}	H_{s2}
τ_1	2	4	5	1	1
τ_2	3	12	12	1	–
τ_3	8	24	25	–	2

[τ_1 has highest DM priority]

$R_1 = C_1 + B_1 = 2 + 2 = 4 \leq D_1 = 4 \Rightarrow$ ok! (but barely)

[τ_2 has medium DM priority]

$R_2 = C_2 + B_2 + \left\lceil \frac{R_2}{T_1} \right\rceil \cdot C_1$ [Assume $R_2^0 = C_2 = 3$]

$R_2^1 = 3 + 2 + \left\lceil \frac{3}{5} \right\rceil \cdot 2 = 3 + 2 + 1 \cdot 2 = 7$

$R_2^2 = 3 + 2 + \left\lceil \frac{7}{5} \right\rceil \cdot 2 = 3 + 2 + 2 \cdot 2 = 9$

$R_2^3 = 3 + 2 + \left\lceil \frac{9}{5} \right\rceil \cdot 2 = 3 + 2 + 2 \cdot 2 = 9$

} Convergence $\leq D_2 = 12 \Rightarrow$ ok!

Lecture #12 – blackboard scribble

[τ_3 has lowest priority]

$$R_3 = C_3 + \left\lceil \frac{R_3}{T_2} \right\rceil C_2 + \left\lceil \frac{R_3}{T_1} \right\rceil C_1$$

[Assume $R_3^0 = C_3 = 8$]

Task	C_i	D_i	T_i	H_{S1}	H_{S2}
τ_1	2	4	5	1	1
τ_2	3	12	12	1	–
τ_3	8	24	25	–	2

$$R_3^1 = 8 + \left\lceil \frac{8}{12} \right\rceil \cdot 3 + \left\lceil \frac{8}{5} \right\rceil \cdot 2 = 8 + 1 \cdot 3 + 2 \cdot 2 = 15$$

$$R_3^2 = 8 + \left\lceil \frac{15}{12} \right\rceil \cdot 3 + \left\lceil \frac{15}{5} \right\rceil \cdot 2 = 8 + 2 \cdot 3 + 3 \cdot 2 = 20$$

$$R_3^3 = 8 + \left\lceil \frac{20}{12} \right\rceil \cdot 3 + \left\lceil \frac{20}{5} \right\rceil \cdot 2 = 8 + 2 \cdot 3 + 4 \cdot 2 = 22$$

$$R_3^4 = 8 + \left\lceil \frac{22}{12} \right\rceil \cdot 3 + \left\lceil \frac{22}{5} \right\rceil \cdot 2 = 8 + 2 \cdot 3 + 5 \cdot 2 = 24$$

$$R_3^5 = 8 + \left\lceil \frac{24}{12} \right\rceil \cdot 3 + \left\lceil \frac{24}{5} \right\rceil \cdot 2 = 8 + 2 \cdot 3 + 5 \cdot 2 = 24$$

Convergence

$\leq D_3 = 24 \Rightarrow$ ok!

(but barely)

All deadlines are met!

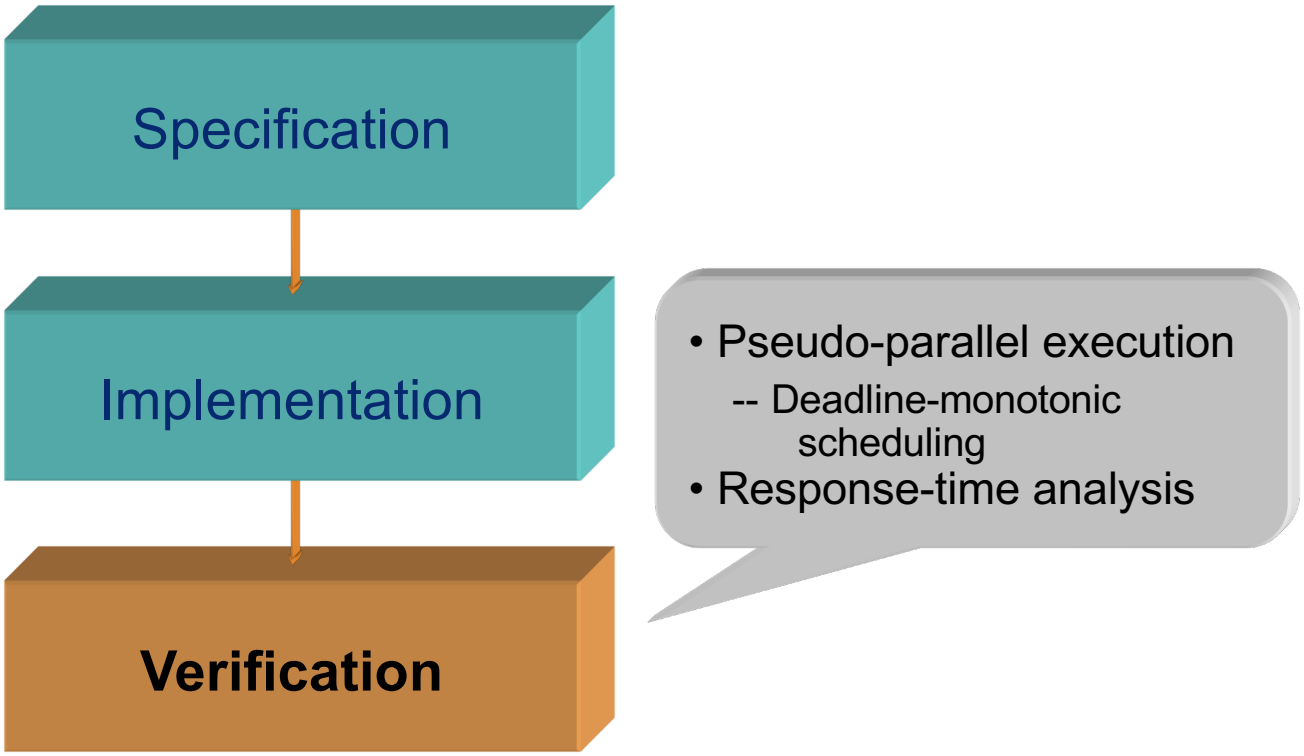
Real-Time Systems

Lecture #12

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



Example: scheduling using RM

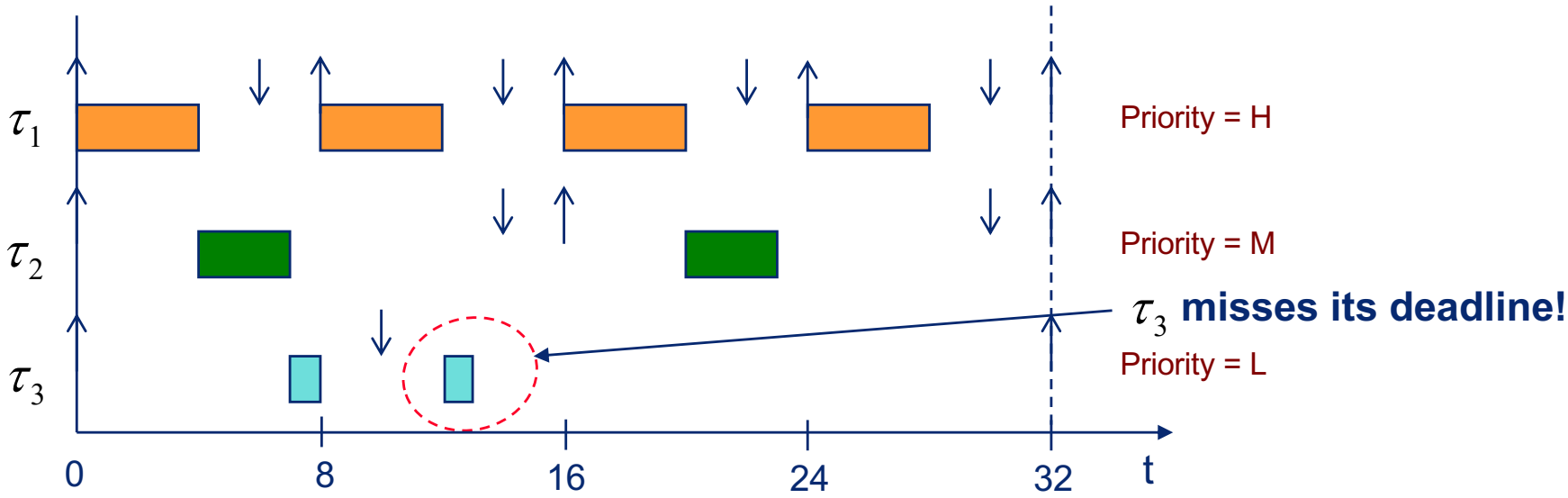
Problem: Assume a system with tasks according to the figure below. The timing properties of the tasks are given in the table. All tasks arrive the first time at time 0.

Investigate the schedulability of the tasks when RM is used. (Note that $D_i < T_i$ for all tasks)

Task	C_i	D_i	T_i
τ_1	4	6	8
τ_2	3	14	16
τ_3	2	10	32

Example: scheduling using RM

Simulate an execution of the tasks using RM:



The tasks are not schedulable even though

$$U = \frac{4}{8} + \frac{3}{16} + \frac{2}{32} = \frac{24}{32} = 0.75 < U_{RM} = 3\left(2^{1/3} - 1\right) \approx 0.780$$

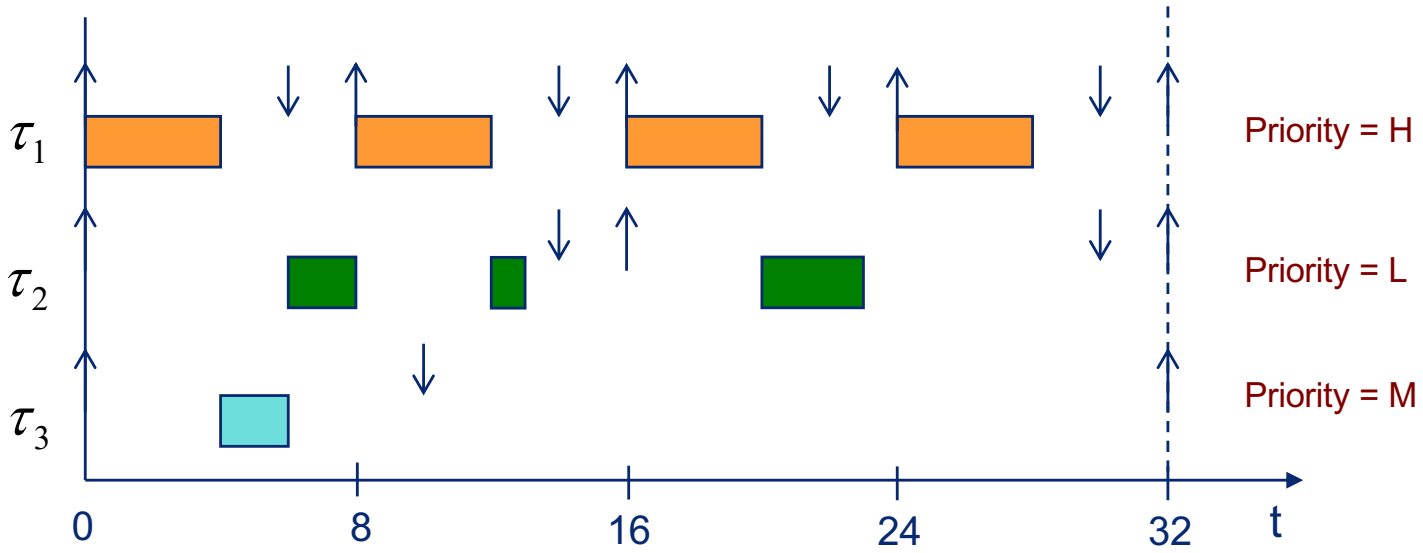
Deadline-monotonic scheduling

Properties:

- Uses static priorities
  - Priority is determined by urgency: the task with the shortest relative deadline receives highest priority
  - Proposed as a generalization of rate-monotonic scheduling (RM is a special case of DM, with $D_i = T_i$)
- Theoretically well-established
  - Exact feasibility test is an NP-complete problem (pseudo-polynomial time with response-time analysis)
  - DM is optimal among all scheduling algorithms that use static task priorities for constrained-deadline tasks (with $D_i \leq T_i$) (shown by J. Leung and J. W. Whitehead in 1982)

Example: scheduling using DM

Simulate an execution of the task set given earlier using DM:



All tasks now meet their deadlines!

Feasibility tests

What types of feasibility tests exist?

- Hyper period analysis (for any type of scheduler)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (static/dynamic priority scheduling)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (static priority scheduling)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (dynamic priority scheduling)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Response-time analysis

The response time R_i for a task τ_i represents the worst-case completion time of the task when execution interference from other tasks are accounted for.

The response time for a task τ_i consists of:

- C_i The task's uninterrupted execution time (WCET)
- I_i Interference from higher-priority tasks

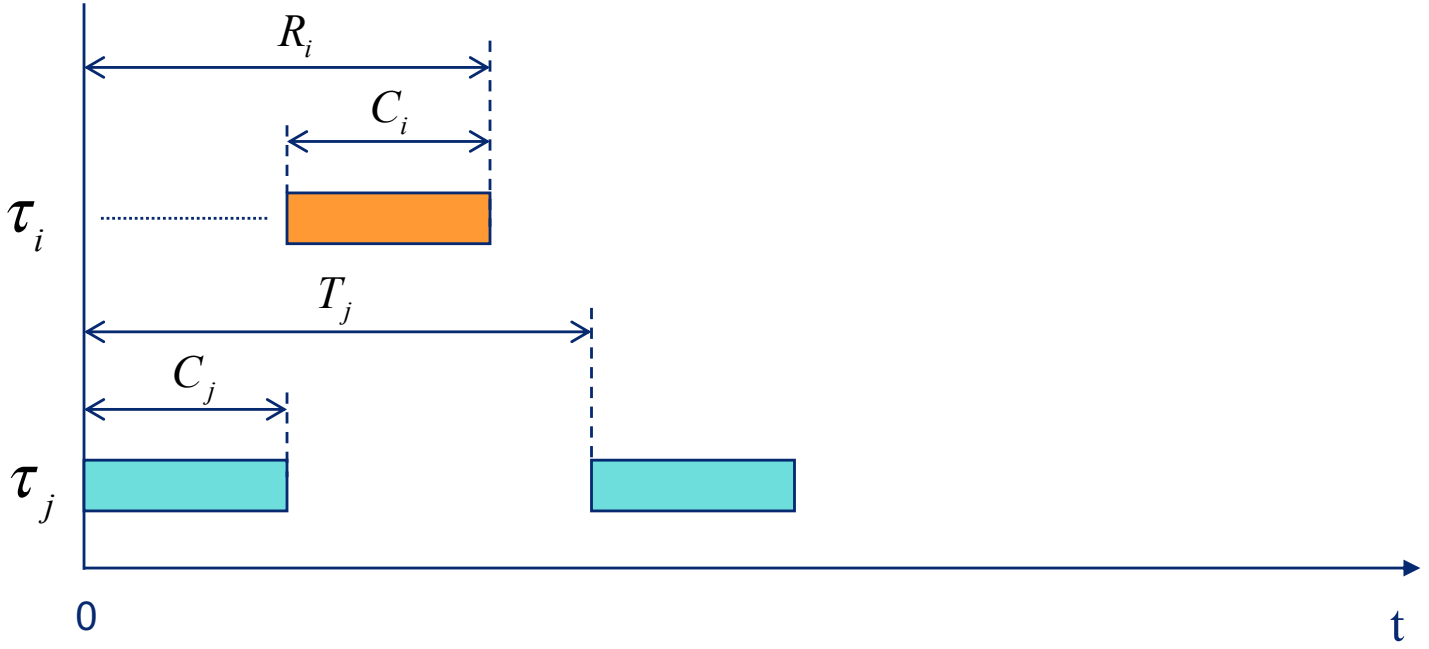
$$R_i = C_i + I_i$$

Response-time analysis

Interference:

Consider two tasks, τ_i and τ_j, where τ_j has higher priority

Case 1: $0 < R_i \leq T_j \Rightarrow R_i = C_i + C_j$

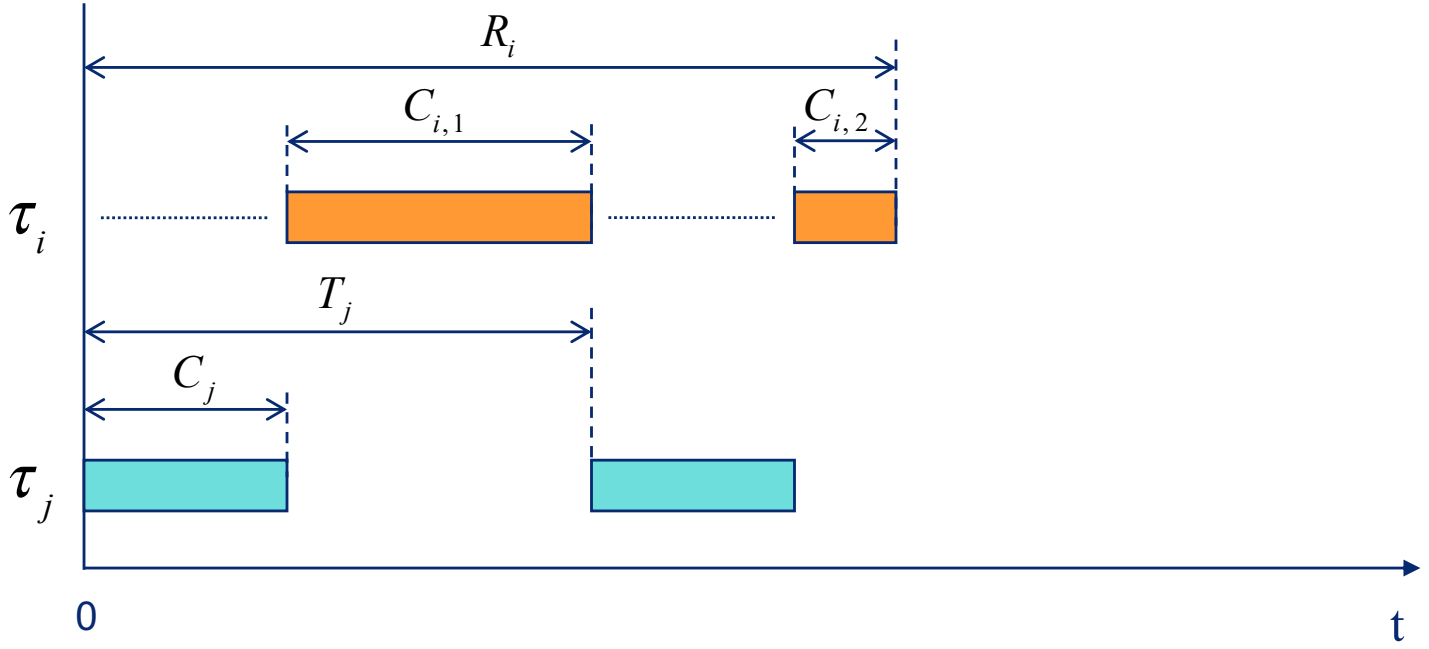


Response-time analysis

Interference:

Consider two tasks, τ_i and τ_j, where τ_j has higher priority

Case 2: $T_j < R_i \leq 2T_j \Rightarrow R_i = C_i + 2C_j$



Response-time analysis

Interference:

When task τ_i is preempted by higher-priority task τ_j:

The response time for τ_i is at most R_i time units.

If $0 < R_i \leq T_j$, task τ_i can be preempted at most one time by τ_j

If $T_j < R_i \leq 2T_j$, task τ_i can be preempted at most two times by τ_j

If $2T_j < R_i \leq 3T_j$, task τ_i can be preempted at most three times by τ_j

...

The number of interferences from τ_j is thus limited by: $\left\lceil \frac{R_i}{T_j} \right\rceil$

The total time for these interferences are: $\left\lceil \frac{R_i}{T_j} \right\rceil C_j$

Response-time analysis

Interference:

- For static-priority scheduling, the interference term is

$$I_i = \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

where $hp(i)$ is the set of tasks with higher priority than τ_i.

- The response time for a task τ_i is thus:

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

Response-time analysis

Interference:

- The equation does not have a simple analytic solution.
- However, an iterative procedure can be used:

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

- The iteration starts with a value that is guaranteed to be less than or equal to the final value of R_i (e.g. $R_i^0 = C_i$)
- The iteration completes at convergence ($R_i^{n+1} = R_i^n$) or if the response time exceeds some threshold (e.g. D_i)

Exact feasibility test for DM

(Sufficient and necessary condition)

A sufficient and necessary condition for DM scheduling of synchronous task sets, for which $D_i \leq T_i$, is

$$\forall i: R_i \leq D_i$$

where R_i is the worst-case response time for task τ_i

In other words: *for the task set to be schedulable with DM there must not exist an instance of a task execution in the schedule where the worst-case response time of the task exceeds its deadline.*

The response-time analysis and associated feasibility test was presented by M. Joseph and P. Pandya in 1986.

Exact feasibility test for DM

(Sufficient and necessary condition)

The test is valid under the following assumptions:

1. All tasks are independent
   - There must not exist dependencies due to precedence or mutual exclusion
2. All tasks are periodic or sporadic
3. All tasks have identical offsets (= synchronous task set)
4. Task deadline does not exceed the period
   (= constrained-deadline tasks)
5. Task preemptions are allowed

Example 1: scheduling using DM

Problem: We once again assume the system with tasks given in the beginning of this lecture.

Show, by using response-time analysis, that the tasks are schedulable using DM.

Task	C_i	D_i	T_i
τ_1	4	6	8
τ_2	3	14	16
τ_3	2	10	32

Extended response-time analysis

The test can be extended to handle:

- Blocking
- Start-time variations (”release jitter”)
- Time offsets (asynchronous task sets)
- Deadlines exceeding the period
- Overhead due to context switches, timers, interrupts, …

In this course, we only show how blocking is handled.

Extended response-time analysis

Blocking can be accounted for in the following cases:

- Blocking caused by critical regions
  - Blocking factor B_i represents the length of critical region(s) that are executed by tasks with lower priority than τ_i
- Blocking caused by non-preemptive scheduling
  - Blocking factor B_i represents largest WCET (not counting τ_i)

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

Recollection from an earlier lecture

Priority Ceiling Protocol:

- Basic idea:

  Each resource is assigned a priority ceiling equal to the priority of the highest-priority task that can lock it.

  Then, a task τ_i is allowed to enter a critical region only if its priority is higher than all priority ceilings of the resources currently locked by tasks other than τ_i.

  When a task τ_i blocks one or more higher-priority tasks, it temporarily inherits the highest priority of the blocked tasks.

Recollection from an earlier lecture

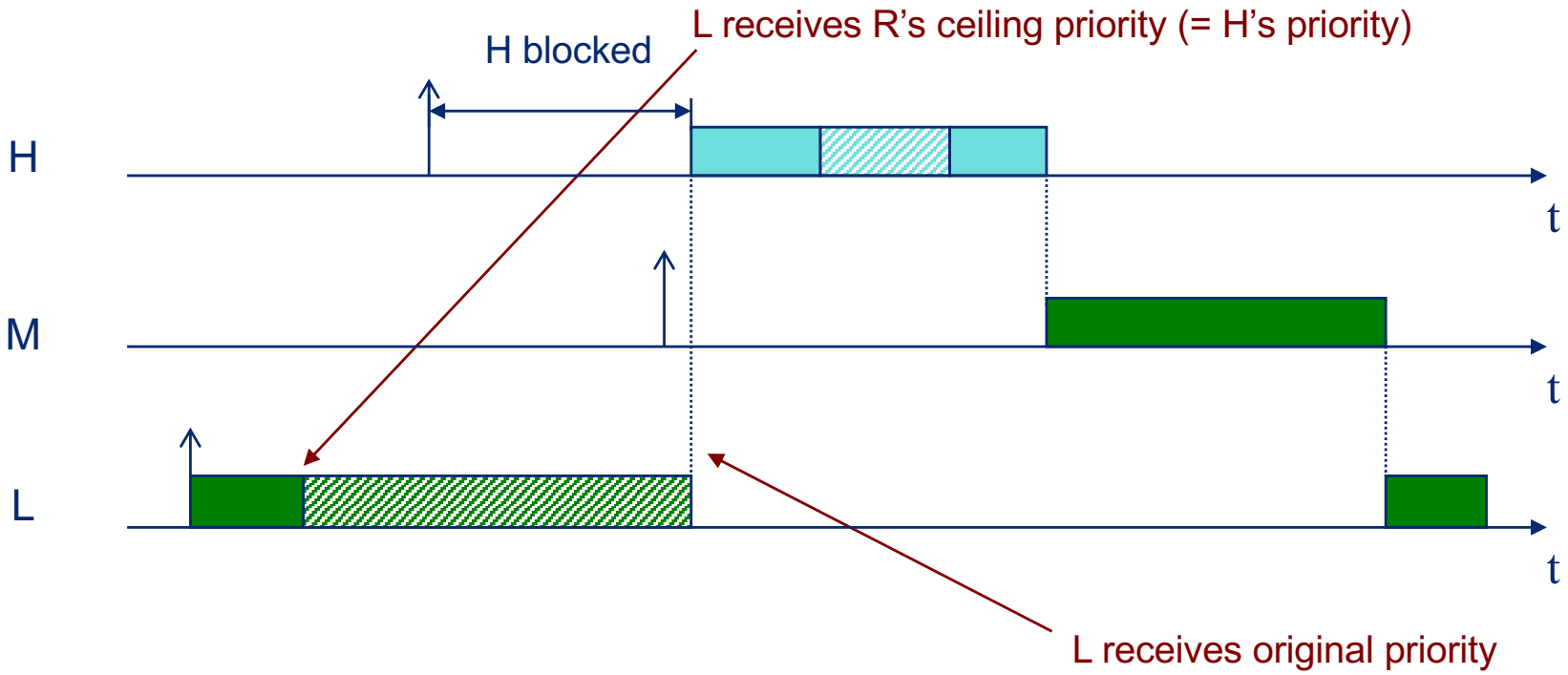
Blocking using ceiling priority protocol ICPP:

priority (H) > priority (M) > priority (L)

H and L share mutex resource R

normal execution

critical region



Extended response-time analysis

Blocking caused by lower-priority tasks:

- When using a priority ceiling protocol (such as ICPP), a task τ_i can only be blocked once by a task with lower priority than τ_i.
- This occurs if the lower-priority task is within a critical region when τ_i arrives, and the critical region's ceiling priority is higher than or equal to the priority of τ_i.
- Blocking now means that the start time of τ_i is delayed (= the blocking factor B_i)
- As soon as τ_i has started its execution, it cannot be blocked by a lower-priority task.

Extended response-time analysis

Determining the blocking factor for task τ_i:

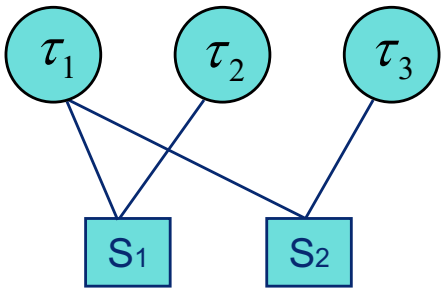
1. Determine the ceiling priorities for all critical regions.
2. Identify the tasks that have a priority lower than τ_i and that calls critical regions with a ceiling priority equal to or higher than the priority of τ_i.
3. Consider the times that these tasks lock the actual critical regions. The longest of those times constitutes the blocking factor B_i.

Example 2: scheduling using DM

Problem: Assume a system with three tasks using two resources, according to the figure below. The timing properties of the tasks are given in the table. Note that $D_i \leq T_i$.

Two semaphores, S_1 and S_2, are used for protecting the resources.

The parameters H_{S1} and H_{S2} represent the longest time a task may lock semaphore S_1 and S_2, respectively.



Task	C_i	D_i	T_i	H_{S1}	H_{S2}
τ_1	2	4	5	1	1
τ_2	3	12	12	1	-
τ_3	8	24	25	-	2

Example 2: scheduling using DM

Problem: (cont'd)

Examine the schedulability of the tasks when ICPP (Immediate Ceiling Priority Protocol) is used.

a) Derive the ceiling priorities of the semaphores.

b) Derive the blocking factors for the tasks.

c) Determine whether the tasks are schedulable or not using DM.

Lecture #13 – blackboard scribble

Determine L_{max}, the largest interval to examine:

$$U = U_1 + U_2 + U_3 = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} = 0,5 + 0,25 + 0,125 = 0,875$$

Since $U < 1$: $L_{max} = \min(L_{BRH}, L_{LCM})$

$$L_{BRH} = \max\left\{D_1, D_2, D_3, \frac{\sum_{i=1}^{3}(T_i - D_i)U_i}{1-U}\right\}$$

$$(T_1 - D_1)\cdot U_1 = (2-1)\cdot 0,5 = 0,5$$

$$(T_2 - D_2)\cdot U_2 = (4-2)\cdot 0,25 = 0,5$$

$$(T_3 - D_3)\cdot U_3 = (8-3)\cdot 0,125 = 0,625$$

$$L^* = \frac{\sum(T_i - D_i)U_i}{1-U} = \frac{0,5 + 0,5 + 0,625}{1 - 0,875} = \frac{1,625}{0,125} = 13$$

$$L_{BRH} = \max\{D_1, D_2, D_3, L^*\} = \max\{1, 2, 3, 13\} = 13$$

$$L_{LCM} = LCM\{T_1, T_2, T_3\} = LCM\{2, 4, 8\} = 8$$

$$L_{max} = \min(L_{BRH}, L_{LCM}) = \min(13, 8) = 8$$

Task	C_i	D_i	T_i
τ_1	1	1	2
τ_2	1	2	4
τ_3	1	3	8

Lecture #13 – blackboard scribble

Determine the control points:

$$K = \{ D_i^k \mid D_i^k = kT_i + D_i,\ D_i^k \le L_{max},\ 1 \le i \le n,\ k \ge 0 \}$$

$L_{max}/T_1 = 4$

$$K_1 = \{ D_1^k \mid D_1^k = kT_1 + D_1,\ D_1^k \le 8,\ k = 0,1,2,3 \} = \{1,3,5,7\}$$

$$K_2 = \{ D_2^k \mid D_2^k = kT_2 + D_2,\ D_2^k \le 8,\ k = 0,1 \} = \{2,6\}$$

$$K_3 = \{ D_3^k \mid D_3^k = kT_3 + D_3,\ D_3^k \le 8,\ k = 0 \} = \{3\}$$

Processor demand must be checked at the following control points:

$$K = \{1,2,3,5,6,7\}$$

we define a table and examine each control point

Task	C_i	D_i	T_i
τ_1	1	1	2
τ_2	1	2	4
τ_3	1	3	8

Lecture #13 – blackboard scribble

L	$N_1^L \cdot C_1$	$N_2^L \cdot C_2$	$N_3^L \cdot C_3$	$C_p(0,L)$	$C_p(0,L) \leq L$?
1	$\left(\left\lfloor \frac{1-1}{2} \right\rfloor + 1\right) \cdot 1 = 1$	$\left(\left\lfloor \frac{1-2}{4} \right\rfloor + 1\right) \cdot 1 = 0$	$\left(\left\lfloor \frac{1-3}{8} \right\rfloor + 1\right) \cdot 1 = 0$	$1+0+0=1$	ok!
2	$\left(\left\lfloor \frac{2-1}{2} \right\rfloor + 1\right) \cdot 1 = 1$	$\left(\left\lfloor \frac{2-2}{4} \right\rfloor + 1\right) \cdot 1 = 1$	$\left(\left\lfloor \frac{2-3}{8} \right\rfloor + 1\right) \cdot 1 = 0$	$1+1+0=2$	OK!
3	$\left(\left\lfloor \frac{3-1}{2} \right\rfloor + 1\right) \cdot 1 = 2$	$\left(\left\lfloor \frac{3-2}{4} \right\rfloor + 1\right) \cdot 1 = 1$	$\left(\left\lfloor \frac{3-3}{8} \right\rfloor + 1\right) \cdot 1 = 1$	$2+1+1=4$	Not ok!

$$N_i^L = \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1$$

Task	C_i	D_i	T_i
τ_1	1	1	2
τ_2	1	2	4
τ_3	1	3	8

Lecture #13 – blackboard scribble

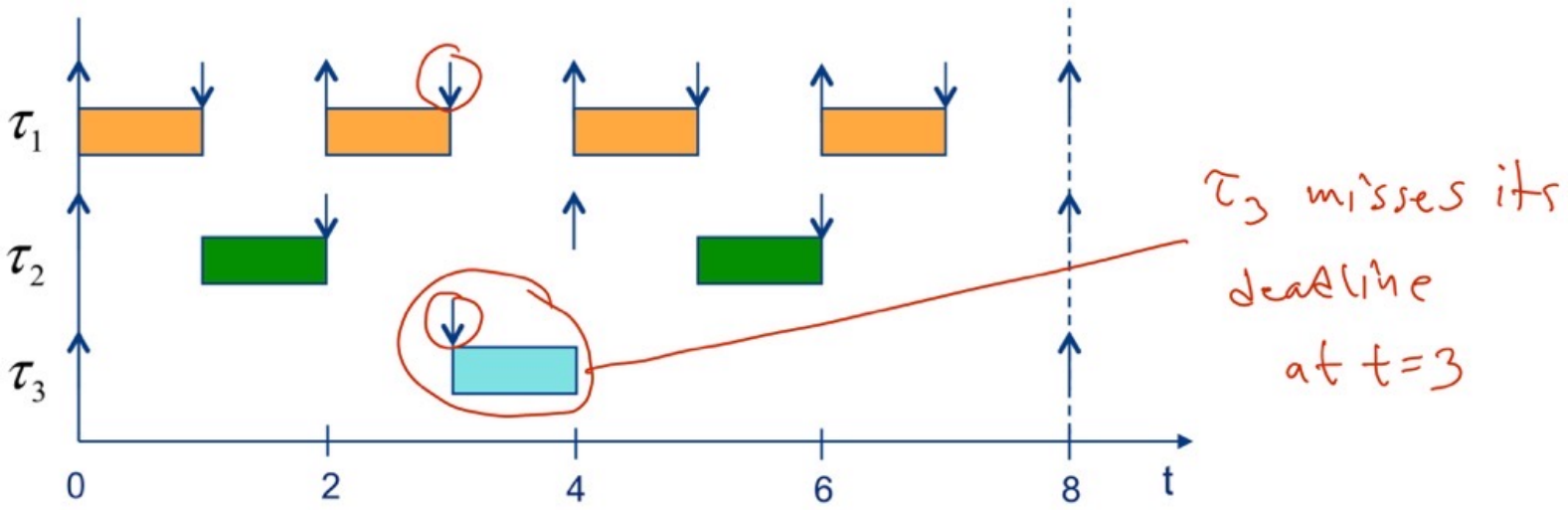
L	$N_1^L \cdot C_1$	$N_2^L \cdot C_2$	$N_3^L \cdot C_3$	$C_p(0,L)$	$C_p(0,L) \leq L$?
1	$\left(\left\lfloor \frac{1-1}{2} \right\rfloor + 1\right) \cdot 1 = 1$ (0)	$\left(\left\lfloor \frac{1-2}{4} \right\rfloor + 1\right) \cdot 1 = 0$ (−1)	$\left(\left\lfloor \frac{1-3}{8} \right\rfloor + 1\right) \cdot 1 = 0$ (−1)	$1+0+0=1$	ok!
2	$\left(\left\lfloor \frac{2-1}{2} \right\rfloor + 1\right) \cdot 1 = 1$	$\left(\left\lfloor \frac{2-2}{4} \right\rfloor + 1\right) \cdot 1 = 1$	$\left(\left\lfloor \frac{2-3}{8} \right\rfloor + 1\right) \cdot 1 = 0$	$1+1+0=2$	OK!
3	$\left(\left\lfloor \frac{3-1}{2} \right\rfloor + 1\right) \cdot 1 = 2$	$\left(\left\lfloor \frac{3-2}{4} \right\rfloor + 1\right) \cdot 1 = 1$	$\left(\left\lfloor \frac{3-3}{8} \right\rfloor + 1\right) \cdot 1 = 1$	$2+1+1=4$	Not OK!
5	$\left(\left\lfloor \frac{5-1}{2} \right\rfloor + 1\right) \cdot 1 = 3$	$\left(\left\lfloor \frac{5-2}{4} \right\rfloor + 1\right) \cdot 1 = 1$	$\left(\left\lfloor \frac{5-3}{8} \right\rfloor + 1\right) \cdot 1 = 1$	$3+1+1=5$	ok!
6	$\left(\left\lfloor \frac{6-1}{2} \right\rfloor + 1\right) \cdot 1 = 3$	$\left(\left\lfloor \frac{6-2}{4} \right\rfloor + 1\right) \cdot 1 = 2$	$\left(\left\lfloor \frac{6-3}{8} \right\rfloor + 1\right) \cdot 1 = 1$	$3+2+1=6$	ok!
7	$\left(\left\lfloor \frac{7-1}{2} \right\rfloor + 1\right) \cdot 1 = 4$	$\left(\left\lfloor \frac{7-2}{4} \right\rfloor + 1\right) \cdot 1 = 2$	$\left(\left\lfloor \frac{7-3}{8} \right\rfloor + 1\right) \cdot 1 = 1$	$4+2+1=7$	ok!

NOTE: floor function

Some task missed deadline

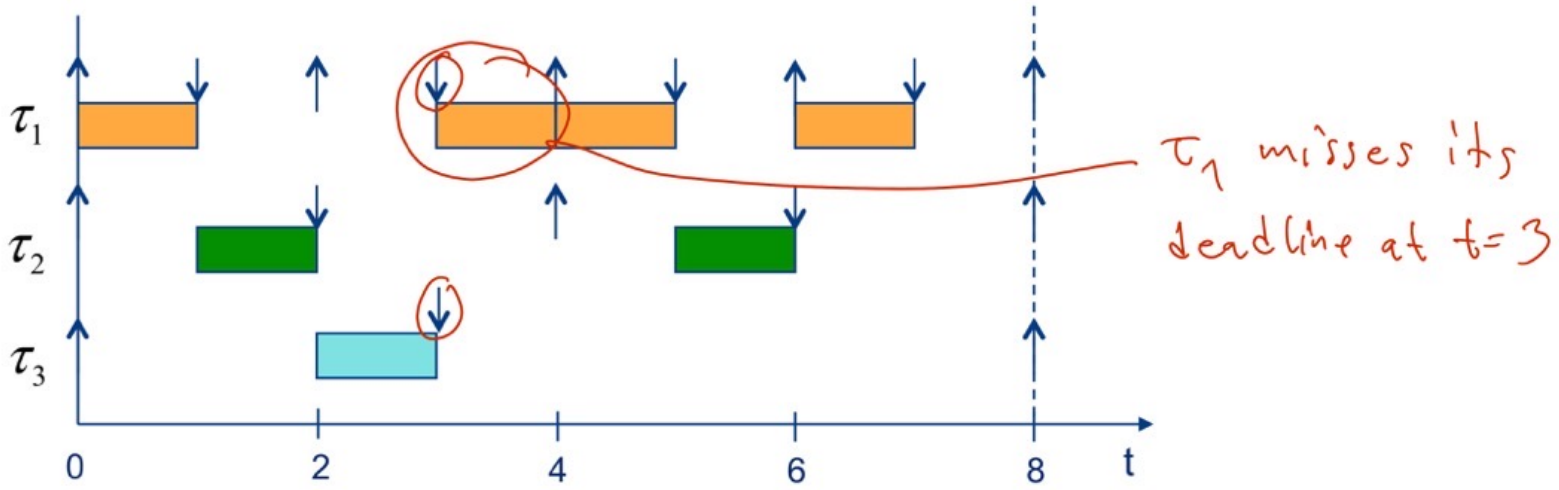
Lecture #13 – blackboard scribble

As we saw in the beginning of the lecture the resulting schedule could look like this:



Lecture #13 – blackboard scribble

But, it could also look like this, as tasks τ_1 and τ_3 both have a deadline at $t=3$ (= they have the same EDF priority)



Consequently, the analysis does not say which task will miss its deadline at $t=3$ (only that some task does)

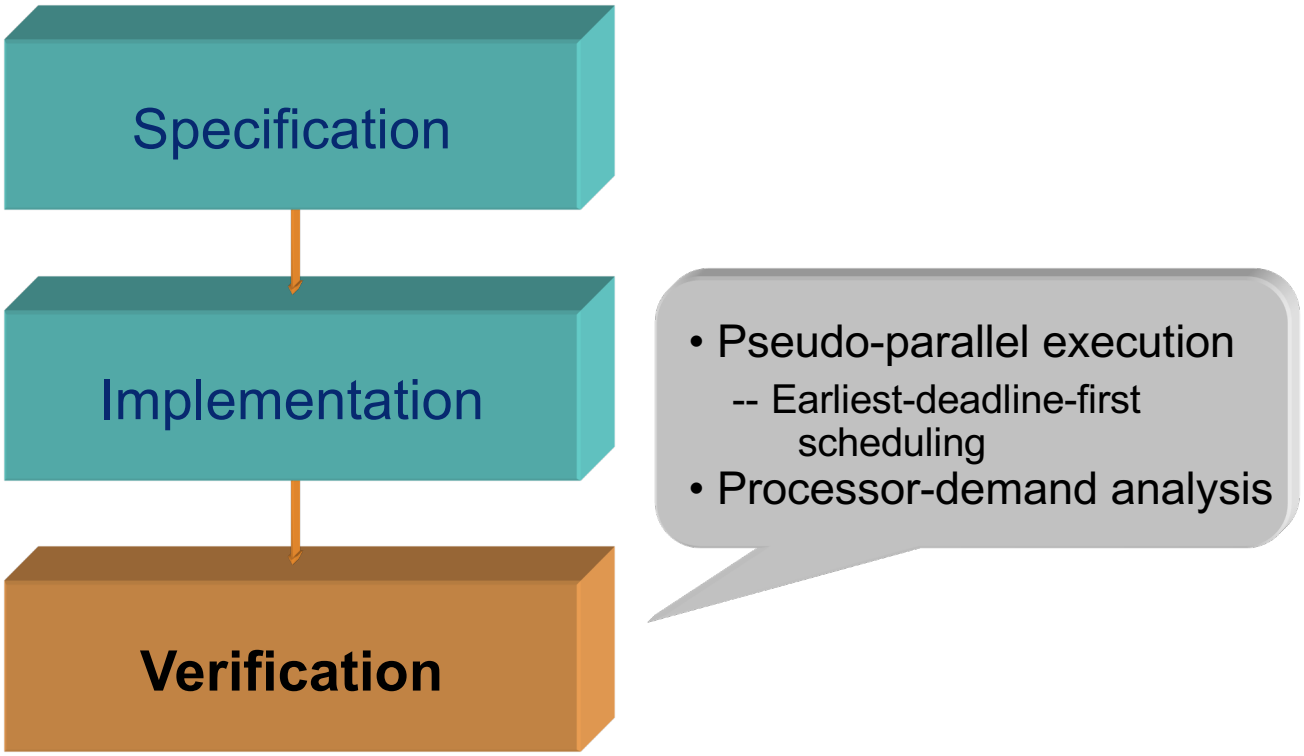
Real-Time Systems

Lecture #13

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



Example: scheduling using EDF

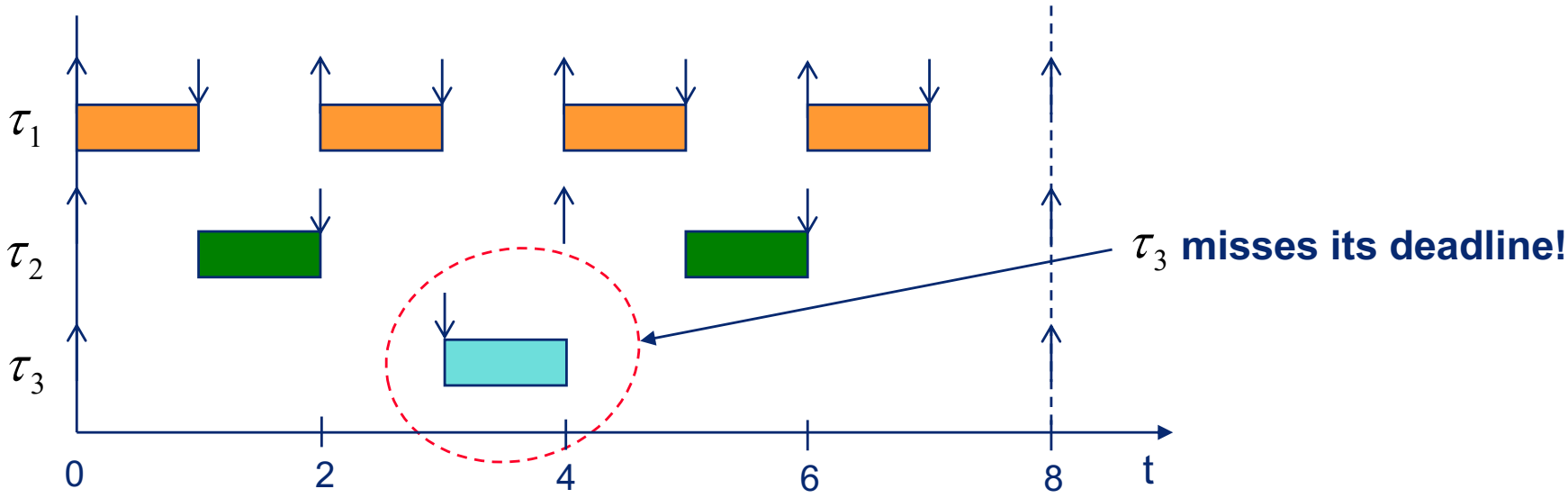
Problem: Assume a system with tasks according to the figure below. The timing properties of the tasks are given in the table. All tasks arrive the first time at time 0.

Investigate the schedulability of the tasks when EDF is used. (Note that $D_i < T_i$ for all tasks)

Task	C_i	D_i	T_i
τ_1	1	1	2
τ_2	1	2	4
τ_3	1	3	8

Example: scheduling using EDF

Simulate an execution of the tasks:



The tasks are not schedulable even though

$$U = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} = \frac{7}{8} = 0.875 < 1$$

Feasibility analysis for EDF

What analysis methods are suitable for general EDF:

- Utilization-based analysis?
  Not suitable! Not general enough or exact enough
  - Does not work well for the case of $D_i < T_i$
- Response-time analysis?
  Not suitable! Analysis much more complex than for DM/RM
  - The critical instant of a task does not necessarily occur when the task arrives at the same time as some other tasks.
  - Instead, the response time of the task may be maximized at some other (asynchronous) arrival pattern.
  - Consequently, the critical instant for each task can in general only be identified by observing the actual schedule.

Feasibility tests

What types of feasibility tests exist?

- Hyper period analysis (for any type of scheduler)
  - In an existing schedule no task execution may miss its deadline
- Processor utilization analysis (static/dynamic priority scheduling)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (static priority scheduling)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (dynamic priority scheduling)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Processor-demand analysis

Processor demand:

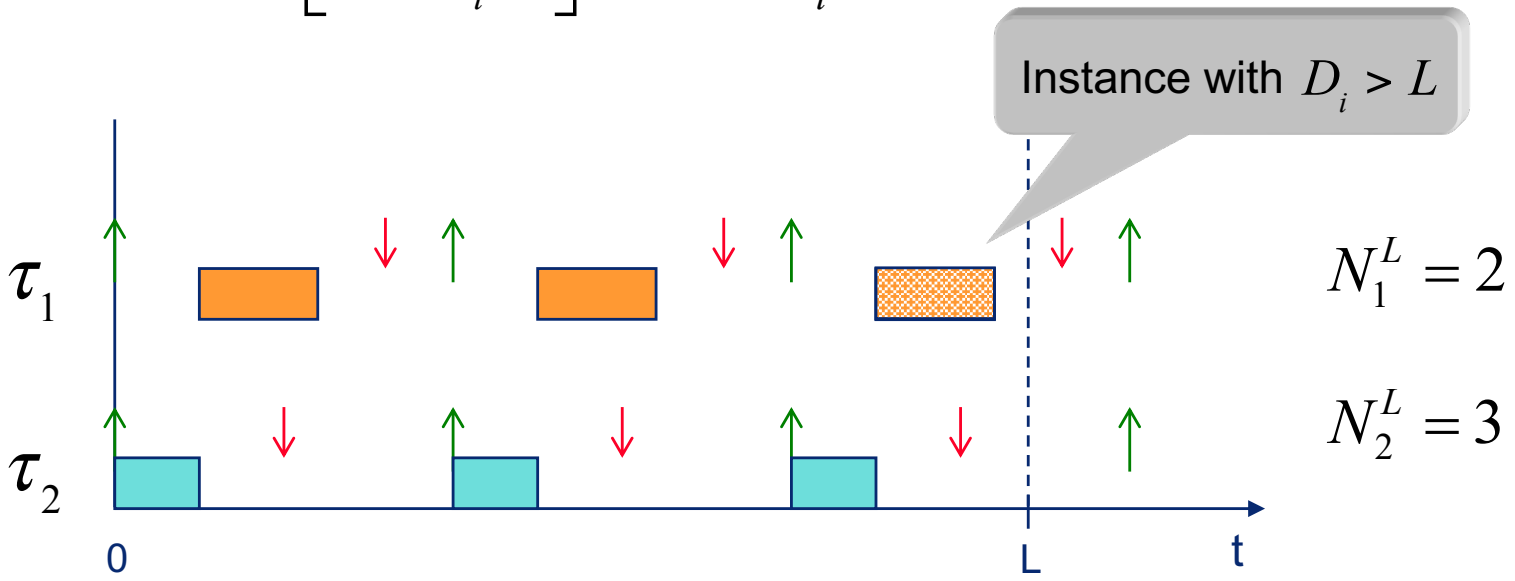
- The processor demand for a task τ_i in a given time interval $[0, L]$ is the amount of processor time that the task needs in the interval in order to meet the deadlines that fall within the interval.
- Let N_i^L represent the number of instances of τ_i that must complete execution before L.
- The total processor demand up to L is

$$C_P(0, L) = \sum_{i=1}^{n} N_i^L C_i$$

Processor-demand analysis

Processor demand:

- We can calculate N_i^L by counting how many times task τ_i has arrived during the interval $\left[0, L - D_i\right]$.
- We can ignore instances of the task that arrived during the interval $\left[L - D_i, L\right]$ since $D_i > L$ for these instances.



Processor-demand analysis

Processor-demand analysis:

- We can express N_i^L as

$$N_i^L = \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1$$

- The total processor demand is thus

$$C_P(0, L) = \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i$$

Exact feasibility test for EDF

(Sufficient and necessary condition)

A sufficient and necessary condition for EDF scheduling of synchronous task sets, for which $D_i \leq T_i$, is

$$\forall L : C_P(0, L) \leq L$$

where $C_P(0, L)$ is the total processor demand in $[0, L]$.

In other words: *for the task set to be schedulable with EDF there must not exist an interval of length L in the schedule where the processor demand in that interval exceeds the length L.*

The processor-demand analysis and associated feasibility test was presented by S. Baruah, L. Rosier and R. Howell in 1990.

Exact feasibility test for EDF

(Sufficient and necessary condition)

The test is valid under the following assumptions:

1. All tasks are independent
   - There must not exist dependencies due to precedence or mutual exclusion
2. All tasks are periodic or sporadic
3. All tasks have identical offsets (= synchronous task set)
4. Task deadline does not exceed the period
   (= constrained-deadline tasks)
5. Task preemptions are allowed

Processor-demand analysis

(Performance aspects)

How many intervals must be examined?

- Only intervals coinciding with the absolute deadlines of tasks need to be examined

For synchronous task sets the feasibility test can consequently be rewritten as follows:

$$\forall L \in K : C_P(0, L) \leq L$$

$$K = \left\{ D_i^k \mid D_i^k = kT_i + D_i, D_i^k \leq L_{\max}, 1 \leq i \leq n, k \geq 0 \right\}$$

Processor-demand analysis

(Performance aspects)

What is the largest interval that must be examined?

- For synchronous task sets in general the largest required interval will always be bounded by the hyper period.

  ⇒ the analysis may have exponential time complexity

$$L_{\text{LCM}} = \text{LCM}\{T_1, \ldots, T_n\}$$

- For most synchronous task sets the largest required interval can be shorter than the hyper period.

  ⇒ the analysis can have pseudo-polynomial time complexity

  (by using a special upper bound) [see Appendix]

Processor-demand analysis

(Performance aspects)

What is the largest interval that must be examined?

- For synchronous task sets with a utilization U < 1 we can use the bound by Baruah, Rosier and Howell:

$$L_{\text{BRH}} = \max\left\{ D_1, \ldots, D_n, \frac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1 - U} \right\}$$

- Properties:
  - ✓ For most task sets $L_{\text{BRH}} < L_{\text{LCM}}$
  - ✓ However, there exist task sets for which $L_{\text{BRH}} > L_{\text{LCM}}$ (typically: task sets with harmonic periods)

Processor-demand analysis

(Performance aspects)

Recommendations for largest required interval:

- For synchronous task sets in general:

$$L_{\max} = L_{\text{LCM}}$$

- For synchronous task sets with utilization U < 1 the following least upper bound can be used:

$$L_{\max} = \min(L_{\text{BRH}}, L_{\text{LCM}})$$

Processor-demand analysis

(Performance aspects)

Examples:

Task	C_i	D_i	T_i
τ_1	1	1	2
τ_2	1	2	4
τ_3	1	3	8

$L_{BRH} = 13 \quad L_{LCM} = 8$

Note: $L_{BRH} > L_{LCM}$

Task	C_i	D_i	T_i
τ_1	3	10	20
τ_2	10	27	30
τ_3	25	54	60

$L_{BRH} = 54 \quad L_{LCM} = 60$

Note: $L_{BRH} < L_{LCM}$

Task	C_i	D_i	T_i
τ_1	1	4	4
τ_2	3	10	15
τ_3	8	14	17

$L_{BRH} = 30 \quad L_{LCM} = 1020$

Note: $L_{BRH} \ll L_{LCM}$

Feasibility tests

Summary for single-processor scheduling

	Implicit-deadline tasks $D_i = T_i$	Constrained-deadline tasks $D_i \le T_i$
Static priority (RM/DM)	$U \le n(2^{1/n} - 1)$	$\forall i : R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \le D_i$
Dynamic priority (EDF)	$U \le 1$	$\forall L : \sum_{i=1}^{n} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i \le L$

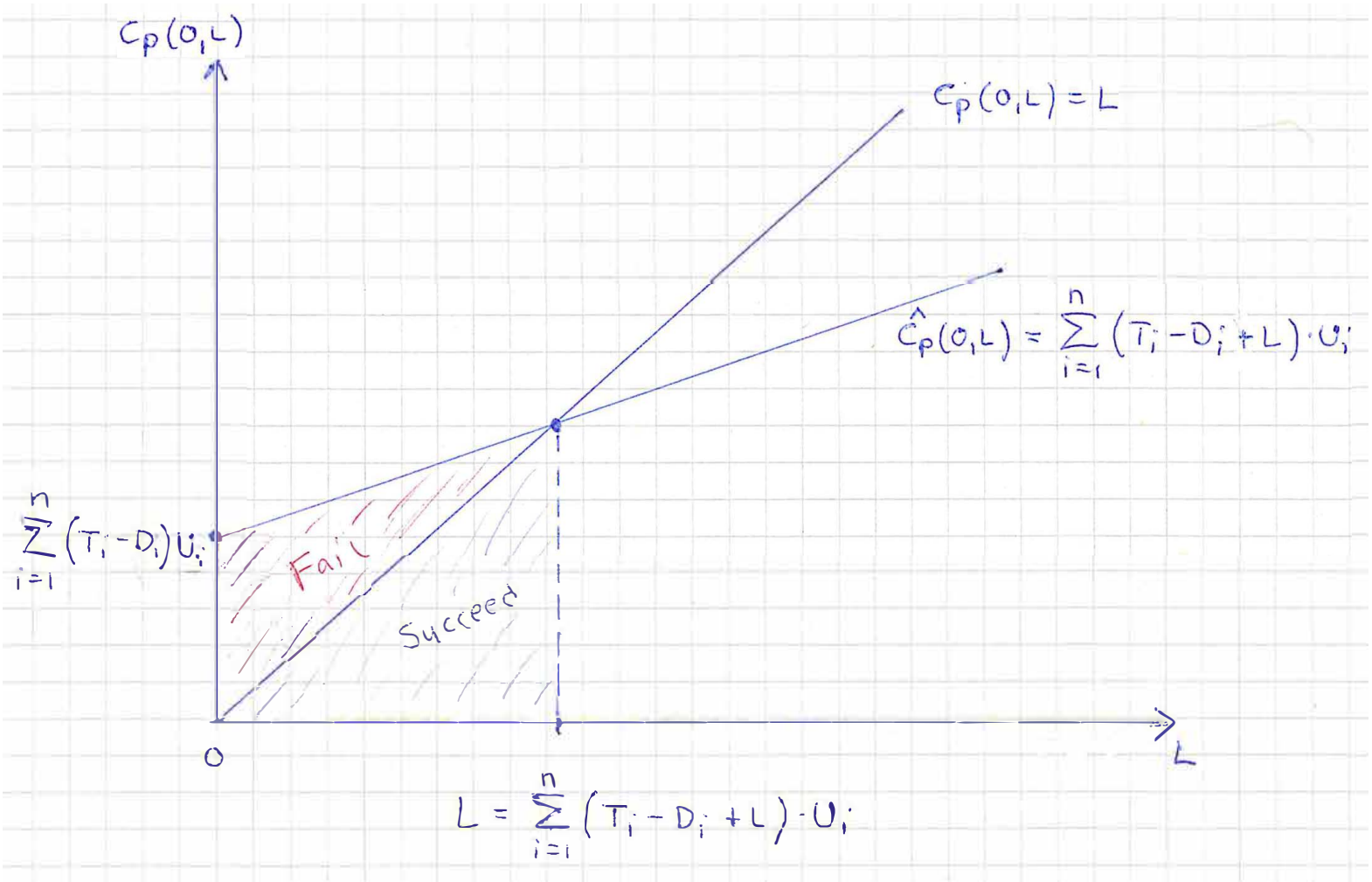
Example: scheduling using EDF

Problem: We once again assume the system with tasks given in the beginning of this lecture.

Show, by using processor-demand analysis, that the tasks are not schedulable using EDF.

Task	C_i	D_i	T_i
τ_1	1	1	2
τ_2	1	2	4
τ_3	1	3	8

Appendix: deriving the L_{BRH} bound



Appendix: deriving the L_{BRH} bound

Upper bound of total processor demand in $[0, L]$:

$$C_p(0,L) \le \hat{C}_p(0,L) = \sum_{i=1}^{n} \left(\frac{L - D_i}{T_i} + 1 \right) C_i = \sum_{i=1}^{n} \left(\frac{L - D_i}{T_i} + \frac{T_i}{T_i} \right) C_i =$$

$$= \sum_{i=1}^{n} \left(T_i - D_i + L \right) \cdot \frac{C_i}{T_i} = \sum_{i=1}^{n} \left(T_i - D_i + L \right) \cdot U_i$$

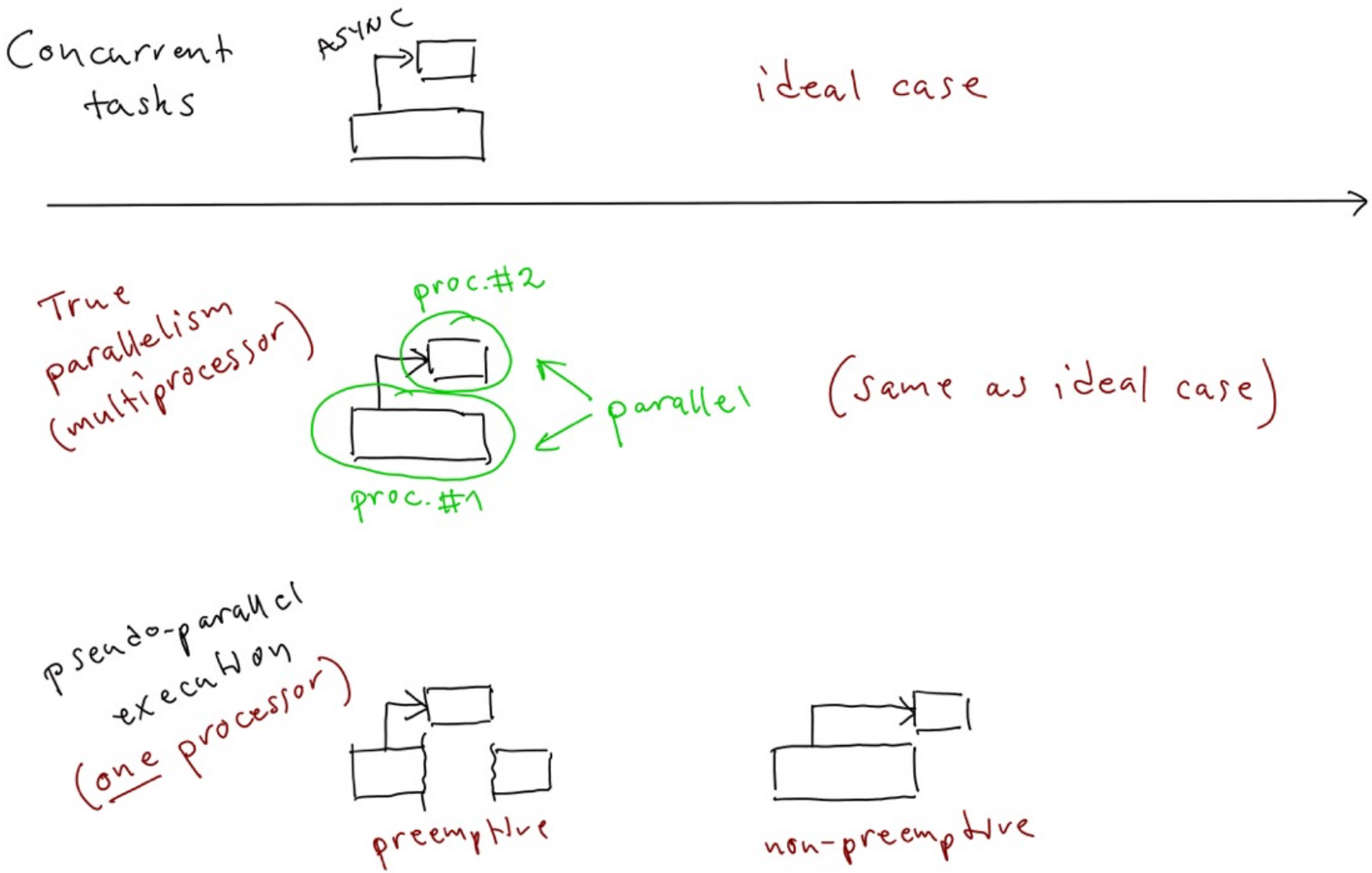
Intersection between the two lines in diagram:

$$L = \sum_{i=1}^{n} \left(T_i - D_i + L \right) U_i = \sum_{i=1}^{n} \left(T_i - D_i \right) U_i + \sum_{i=1}^{n} L \cdot U_i =$$

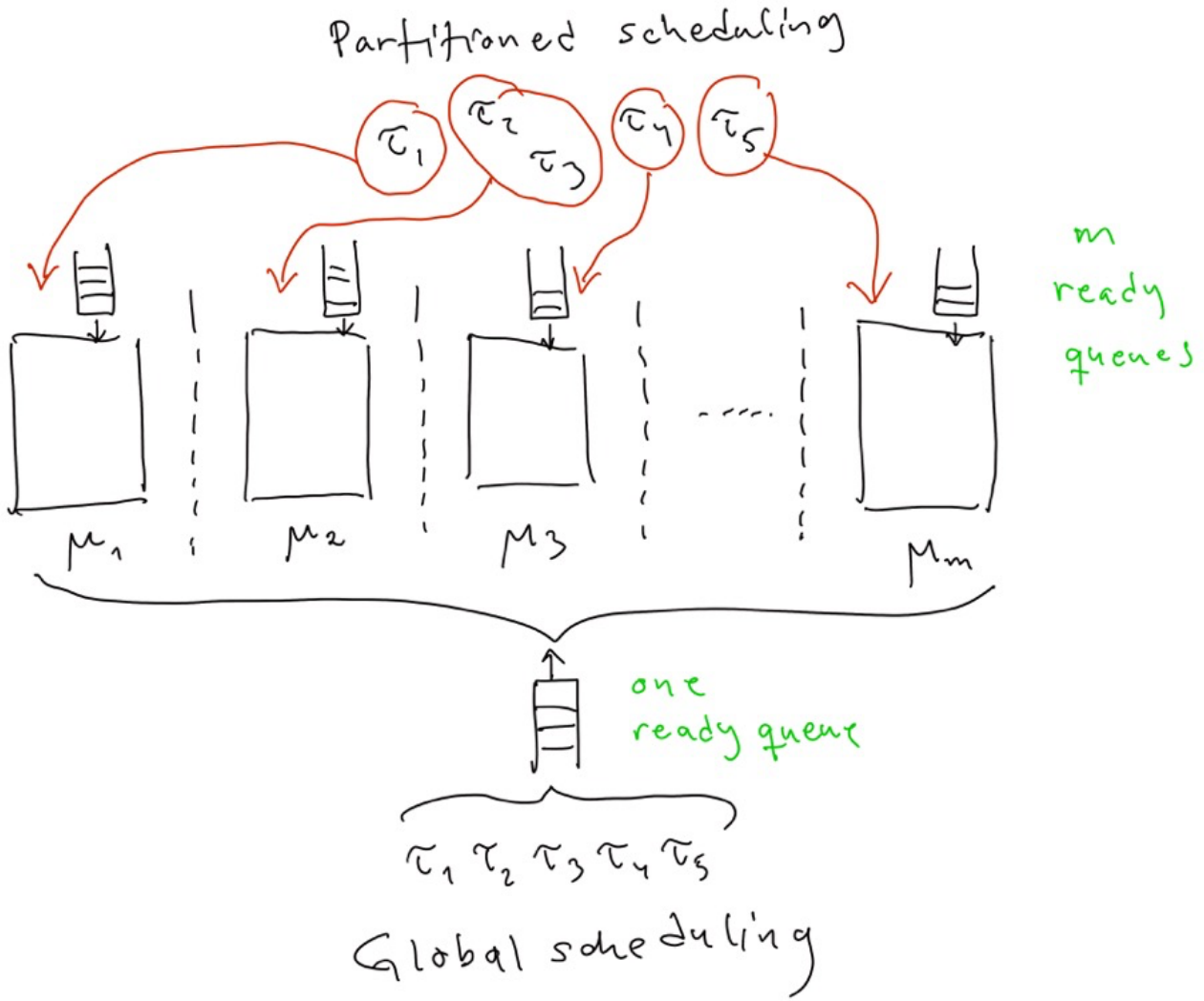
$$= \sum_{i=1}^{n} \left(T_i - D_i \right) \cdot U_i + L \cdot U \quad \Rightarrow$$

$$L(1 - U) = \sum_{i=1}^{n} \left(T_i - D_i \right) \cdot U_i \quad \Rightarrow \quad L = \frac{\sum_{i=1}^{n} \left(T_i - D_i \right) \cdot U_i}{1 - U}$$

Lecture #14 – blackboard scribble



Lecture #14 – blackboard scribble



Lecture #14 – blackboard scribble

RM-US[m/(3m-2)]

m = 3 processors

Calculate utilization-separation (US) bound:

$m/(3m-2) = 3/(3 \cdot 3 - 2) = 3/7 \approx 0,43$

	Task	C_i	T_i	U_i
M_H	τ_1	1	7	0,143
M_L	τ_2	2	10	0,2
H	τ_3	9	20	0,45
H	τ_4	11	22	0,5
L	τ_5	2	25	0,08

Derive priorities:

Based on the US bound (0,43) tasks τ_3 and τ_4 are considered "heavy" tasks, and are assigned highest priority.

The remaining tasks are assigned RM priorities.

This gives two possible priority assignments (high → low)

$\tau_3, \tau_4, \tau_1, \tau_2, \tau_5$

or

$\tau_4, \tau_3, \tau_1, \tau_2, \tau_5$

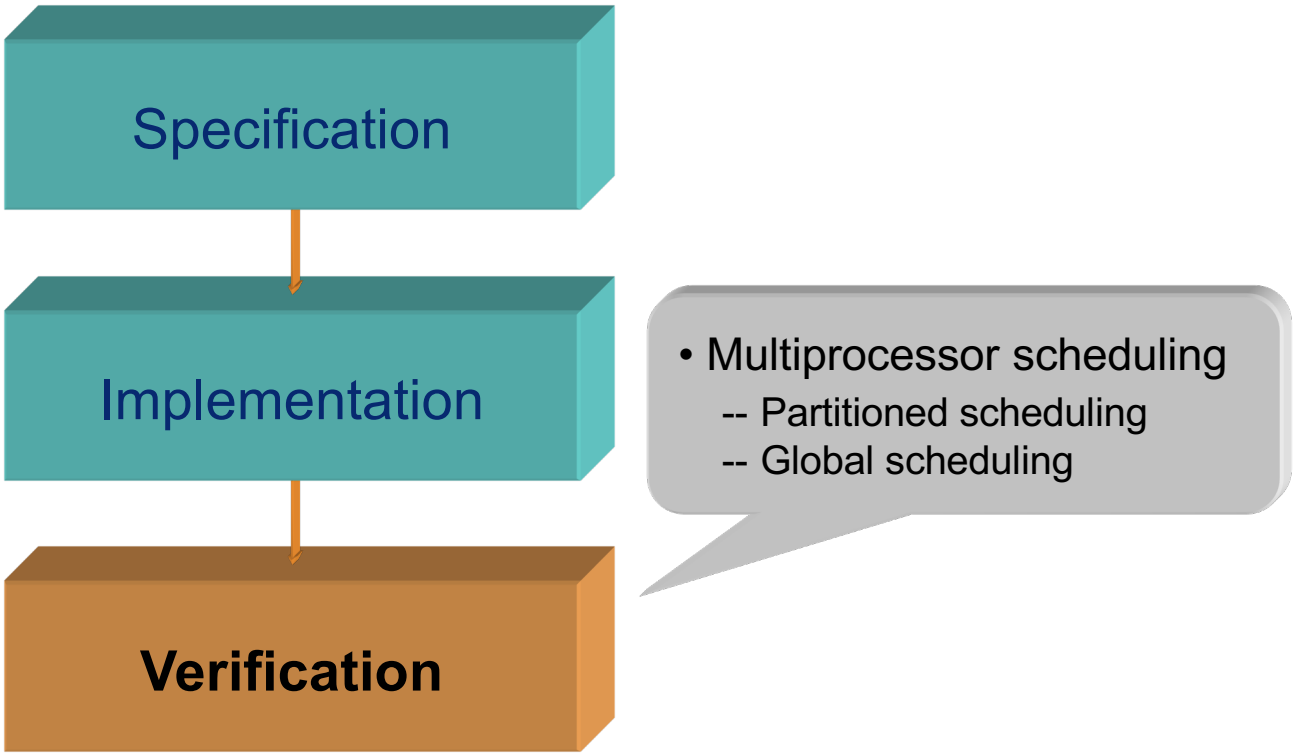
Real-Time Systems

Lecture #14

Docent Risat Pathan

Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



Multiprocessor scheduling

How are tasks assigned to processors?

- Static assignment
  - The processor(s) used for executing a task are determined before system is put in mission (“off-line”)
  - Approach: Partitioned scheduling
- Dynamic assignment
  - The processor(s) used for executing a task are determined during system operation “on-line”
  - Approach: Global scheduling

Multiprocessor scheduling

How are tasks allowed to migrate?

- Partitioned scheduling
  - No migration!
  - Each instance of a task must execute on the same processor
  - Equivalent to multiple single-processor systems!
- Global scheduling
  - Full migration!
  - A task is allowed to execute on an arbitrary processor
  - Migration can occur even during execution of an instance of a task (for example, after being preempted)

Multiprocessor scheduling

A fundamental limit: (Andersson, Baruah & Jonsson, 2001)

> The utilization guarantee bound for multiprocessor scheduling (partitioned or global), using task priorities only, cannot be higher than 50% of the capacity of the processors.

- Hence, we should not expect to utilize more than half the processing capacity if hard real-time constraints exist.
- A way to circumvent this limit is to use p-fair (priorities + time quanta) scheduling and dynamic task priorities.

Partitioned scheduling

General characteristics:

- Each processor has its own queue for ready tasks
- Tasks are organized in groups, and each task group is assigned to a specific processor
  - For example, using a bin-packing algorithm
- When selected for execution, a task can only be dispatched to its assigned processor

Partitioned scheduling

Advantages:

- Mature scheduling framework
  - Most single-processor scheduling theory also applicable here
  - Single-processor resource-management protocols can be used
- Supported by automotive industry
  - AUTOSAR prescribes partitioned scheduling

Disadvantages:

- Cannot exploit all unused execution time
  - Surplus capacity cannot be shared among processors
  - Will suffer from overly-pessimistic WCET derivation

Partitioned scheduling

Bin-packing algorithms:

- Basic idea:
  - The problem concerns packing objects of varying sizes in boxes (”bins”) with the objective of minimizing number of used boxes.
- Application to multiprocessor systems:
  - Bins are represented by processors and objects by tasks.
  - The decision whether a processor is ”full” or not is derived from a utilization-based feasibility test.
- Assumptions:
  - Independent, periodic tasks
  - Preemptive, single-processor scheduling (RM)

Partitioned scheduling

Bin-packing algorithms:

Rate-Monotonic-First-Fit (RMFF): (Dhall and Liu, 1978)

- Let the processors be indexed as $\mu_1, \mu_2, \ldots, \mu_m$
- Assign tasks in order of increasing periods (i.e., RM order).
- For each task τ_i, choose the lowest previously-used j such that τ_i, together with all tasks that have already been assigned to processor μ_j, can be feasibly scheduled according to the utilization-based RM-feasibility test.

If all tasks are successfully assigned using RMFF, then the tasks are schedulable on *m* processors.

Partitioned scheduling

Processor utilization analysis for RMFF:

- A sufficient condition for partitioned RMFF scheduling of synchronous task sets with n tasks on m processors is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq m\left(2^{1/2} - 1\right)$$

(Oh & Baker, 1998)

Note: $U_{RMFF} = m\left(2^{1/2} - 1\right) \approx 0.41m$

Thus: task sets whose utilization do not exceed ≈ 41% of the total processor capacity is always RMFF-schedulable.

Partitioned scheduling

Processor utilization analysis for RMFF:

1. All tasks are independent
2. All tasks are periodic or sporadic
3. All tasks have identical offsets (= synchronous task set)
4. Task deadline equals the period (= implicit-deadline tasks)
5. Task preemptions are allowed
6. All processors are identical
7. Task migrations are not allowed

Global scheduling

General characteristics:

- All ready tasks are kept in a common (global) queue that is shared among the processors
- Whenever a processor becomes idle, a task from the global queue is selected for execution on that processor.
- After being preempted, a task may be dispatched to a processor other than the one that started executing the task.

Global scheduling

Advantages:

- Supported by most multiprocessor operating systems
  - Windows, macOS, Linux, ...
- Effective utilization of processing resources
  - Unused processor time can easily be reclaimed, for example when a task does not execute its full WCET.

Disadvantages:

- Weak theoretical framework
  - Few results from the single-processor analysis can be used

Weak theoretical framework

The ”root of all evil” in global scheduling: (Liu, 1969)

> Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem. The simple fact that *a task can use only one processor even when several processors are free at the same time* adds a surprising amount of difficulty to the scheduling of multiple processors.

Weak theoretical framework

Underlying causes:

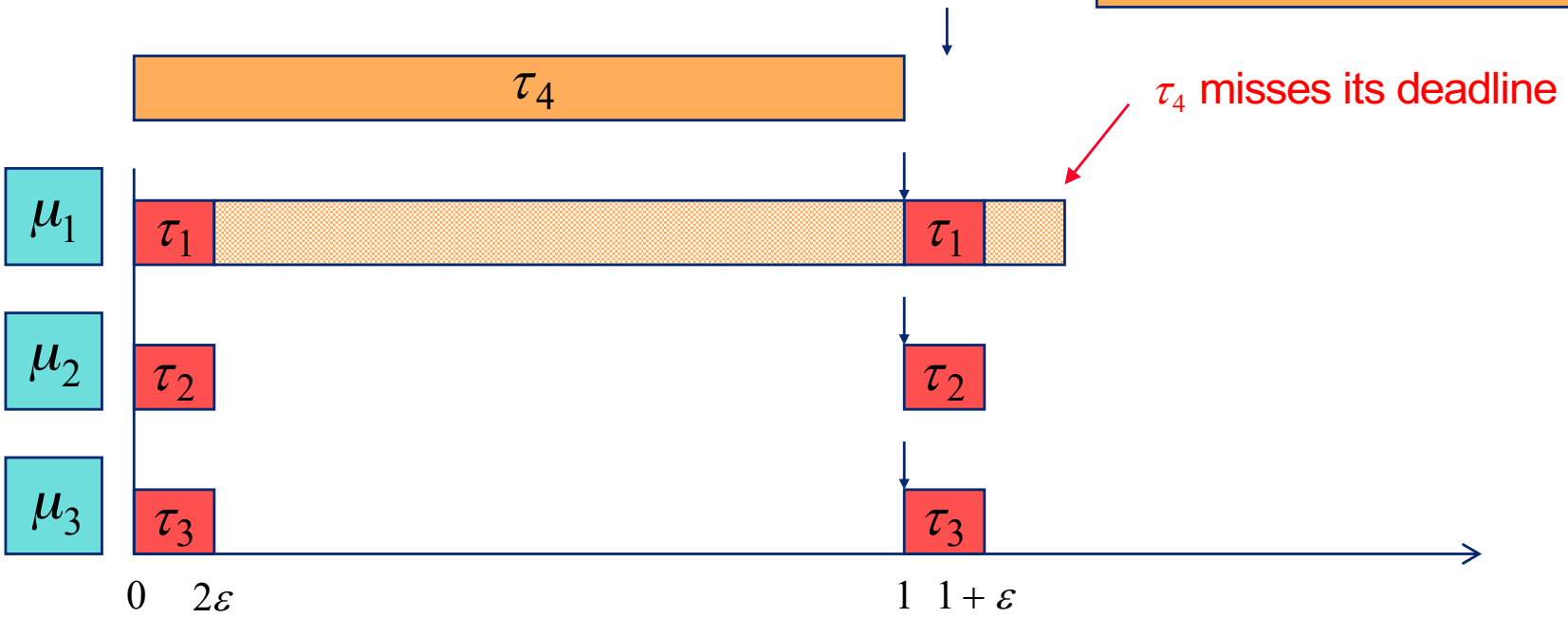
- Dhall's effect:
  - With RM, DM and EDF, some low-utilization task sets can be non-schedulable regardless of how many processors are used. Thus, any utilization guarantee bound would become so low that it would be useless in practice.
  - This is in contrast to the single-processor case, where we have utilization guarantee bounds of 69.3% (RM) and 100% (EDF).
- Hard-to-find critical instant:
  - A critical instant does not always occur when a task arrives at the same time as all its higher-priority tasks.
  - This is in contrast to the single-processor case with RM and DM (and any other static-priority policy).

Weak theoretical framework

Dhall's effect: (Dhall & Liu, 1978)

$$\tau_1 : \{ C_1 = 2\varepsilon, T_1 = 1 \}$$
$$\tau_2 : \{ C_2 = 2\varepsilon, T_2 = 1 \}$$
$$\tau_3 : \{ C_3 = 2\varepsilon, T_3 = 1 \}$$
$$\tau_4 : \{ C_4 = 1, T_4 = 1 + \varepsilon \}$$

(RM scheduling)



Weak theoretical framework

Dhall's effect:

- Applies for RM, DM and EDF scheduling
- The utilization of a non-schedulable task set can be as low as to 1 (= 100%) no matter how many processors are used.

$$U_{global} = m\frac{2\varepsilon}{1} + \frac{1}{1+\varepsilon} \rightarrow 1$$

when $\varepsilon \rightarrow 0$

Note: Total available processor capacity is m (= $m \cdot 100\%$)

Consequence:

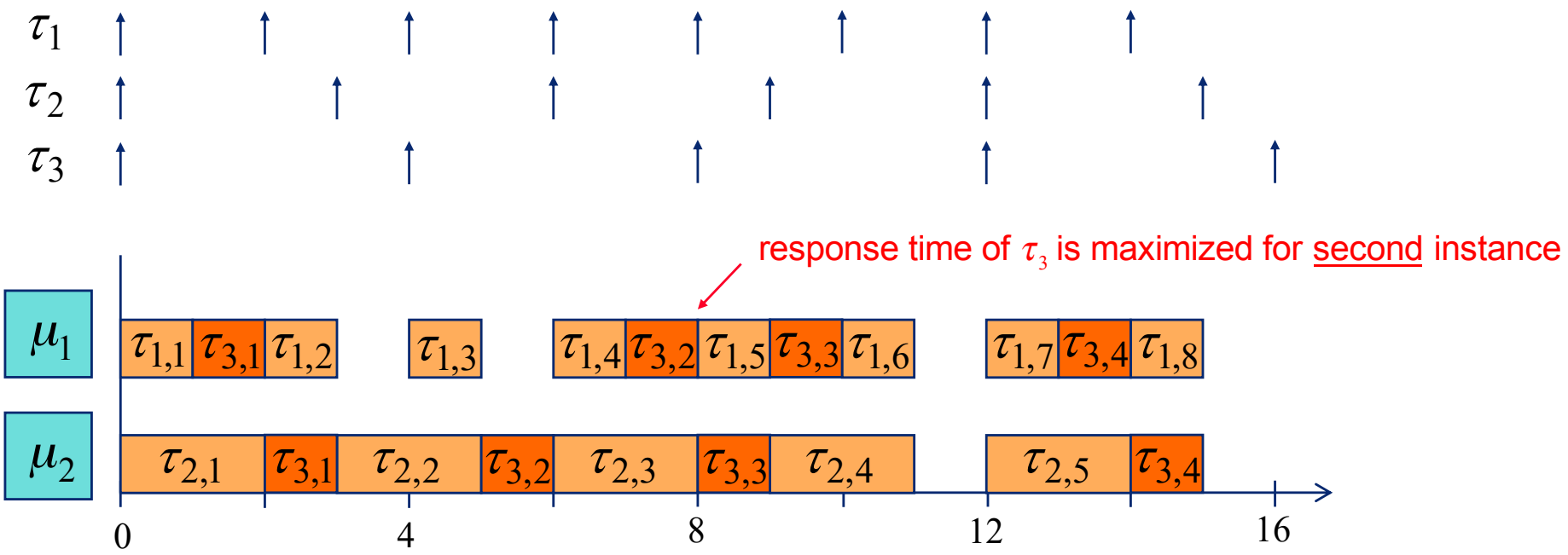
New multiprocessor priority-assignment schemes are needed!

Weak theoretical framework

Hard-to-find critical instant:

$$\tau_1 : \{ C_1 = 1, T_1 = 2 \}$$
$$\tau_2 : \{ C_2 = 2, T_2 = 3 \}$$
$$\tau_3 : \{ C_3 = 2, T_3 = 4 \}$$

(RM scheduling)



Weak theoretical framework

Hard-to-find critical instant:

- A critical instant does not always occur when a task arrives at the same time as all its higher-priority tasks.
- Finding the critical instant is, in general, a problem with exponential time complexity
- Note: recall that knowledge about an easy-to-find critical instant is a fundamental assumption in the single-processor feasibility tests for static-priority scheduling.

Consequence:

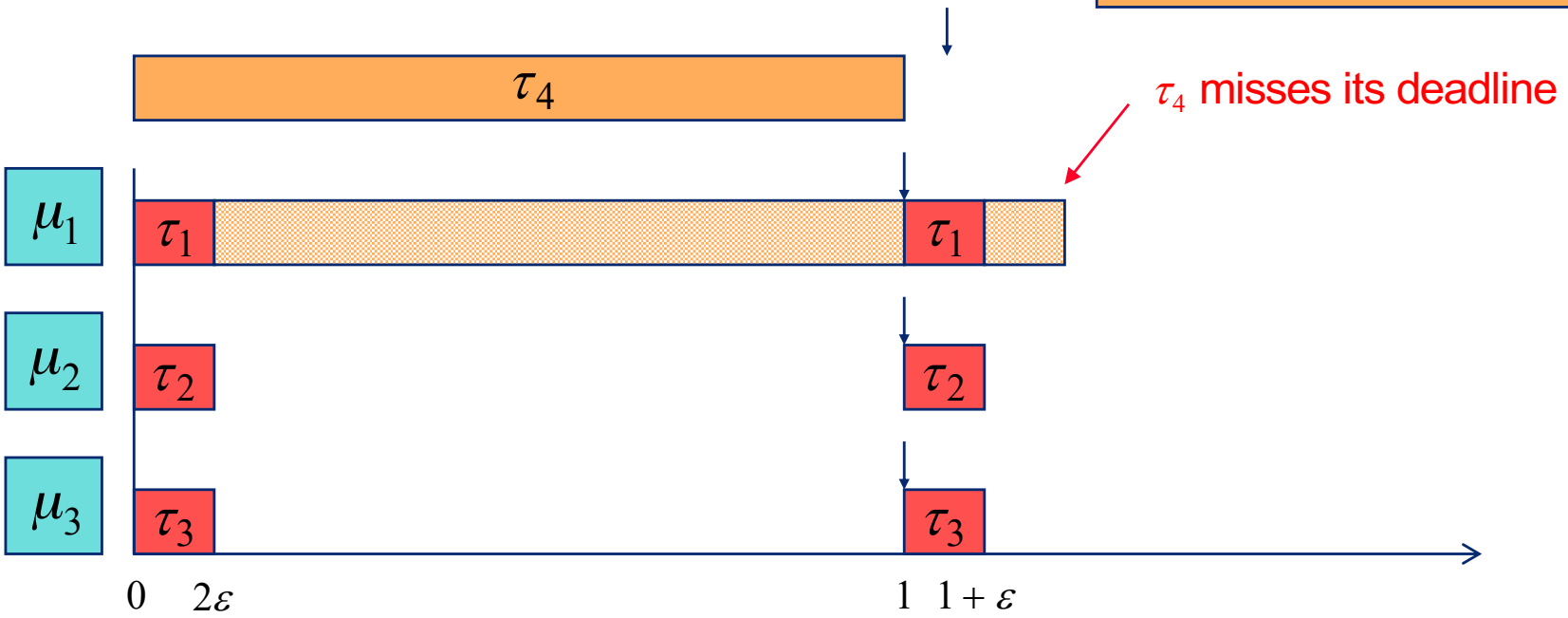
New methods for constructing effective multiprocessor feasibility tests are needed!

Weak theoretical framework

Dhall's effect: (Dhall & Liu, 1978)

$$\tau_1 : \{ C_1 = 2\varepsilon, T_1 = 1 \}$$
$$\tau_2 : \{ C_2 = 2\varepsilon, T_2 = 1 \}$$
$$\tau_3 : \{ C_3 = 2\varepsilon, T_3 = 1 \}$$
$$\tau_4 : \{ C_4 = 1, T_4 = 1 + \varepsilon \}$$

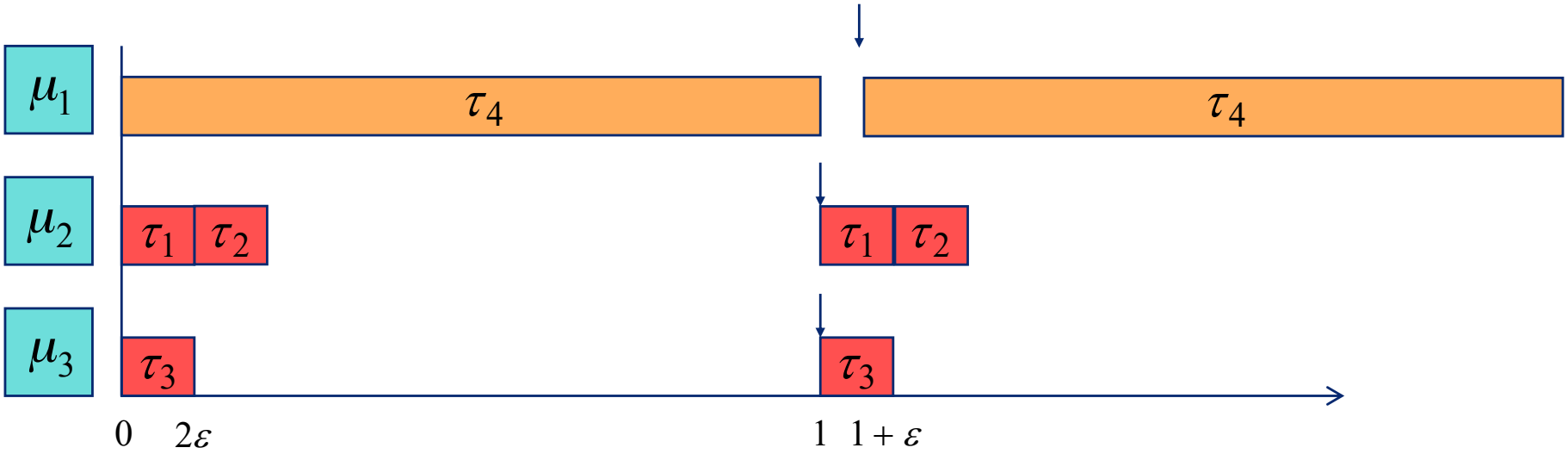
(RM scheduling)



New priority-assignment scheme

How to avoid Dhall's effect:

- Problem: RM, DM & EDF only account for task deadlines! Actual computation demands are not accounted for.
- Solution: Dhall's effect can easily be avoided by letting tasks with high utilization receive higher priority:



New priority-assignment scheme

RM-US[m/(3m-2)]: (Andersson, Baruah & Jonsson, 2001)

- RM-US[m/(3m-2)] assigns (static) priorities to tasks according to the following rule:

  If $U_i > m/(3m-2)$ then τ_i has the highest priority (ties broken arbitrarily)

  If $U_i \leq m/(3m-2)$ then τ_i has RM priority

- Clearly, tasks with higher utilization $U_i = C_i / T_i$ get higher priority.

Example: RM-US[m/(3m-2)]

RM-US[m/(3m-2)] example:

Assign priorities according to RM-US[m/(3m-2)], assuming the following task set to be scheduled on 3 processors:

$$\tau_1 : \{ C_1 = 1, T_1 = 7 \} \qquad \tau_2 : \{ C_2 = 2, T_2 = 10 \}$$

$$\tau_3 : \{ C_3 = 9, T_3 = 20 \} \qquad \tau_4 : \{ C_4 = 11, T_4 = 22 \}$$

$$\tau_5 : \{ C_5 = 2, T_5 = 25 \}$$

New feasibility tests

Processor utilization analysis for RM-US[m/(3m-2)]:

- A sufficient condition for RM-US[m/(3m-2)] scheduling of synchronous task sets with n tasks on m processors is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \frac{m^2}{3m-2}$$

(Andersson, Baruah & Jonsson, 2001)

Question: does RM-US[m/(3m-2)] avoid Dhall's effect?

New feasibility tests

Processor utilization analysis for RM-US[m/(3m-2)]:

- We observe that, regardless of the number of processors, the task set will always meet its deadlines as long as no more than one third of the processing capacity is used:

$$U_{RM-US[m/(3m-2)]} = \lim_{m\to\infty} \frac{m^2}{3m-2} = \frac{m}{3}$$

- RM-US[m/(3m-2)] thus avoids Dhall's effect since we can always add more processors if deadlines were missed.
- Note that this remedy was not possible with traditional RM.

New feasibility tests

Processor utilization analysis for RM-US[m/(3m-2)]:

1. All tasks are independent
2. All tasks are periodic (i.e. not applicable for sporadic tasks)
3. All tasks have identical offsets (= synchronous task set)
4. Task deadline equals the period (= implicit-deadline tasks)
5. Task preemptions are allowed
6. All processors are identical
7. Task migrations are allowed

New feasibility tests

Early breakthrough results in global scheduling:

- Static priorities:
  - 2001: RM-US[m/(3m-2)] is proven to circumvent Dhall's effect and to have a non-zero guarantee bound of m/(3m-2) ≥ 33.3%.
  - 2008: Andersson proposed the SM-US{2/(3+√5)} scheme with a guarantee bound of 2/(3+√5) = 38.2%.
- Dynamic priorities:
  - 2002: Srinivasan & Baruah proposed the EDF-US[m/(2m-1)] scheme with a guarantee bound of m/(2m-1) ≥ 50%.
- Optimal multiprocessor scheduling:
  - 1996: Baruah *et al.* proposed p-fair (priorities + time quanta) scheduling and dynamic task priorities as an approach to achieve a guarantee bound of 100% on a multiprocessor.

New feasibility tests

Response-time analysis for multiprocessors:

- Uses the same principle as the single-processor case, where the response time for a task τ_i consists of:

  C_i The task's uninterrupted execution time (WCET)

  I_i Interference from higher-priority tasks

$$R_i = C_i + I_i$$

- The difference is that the calculation of interference now has to account for the fact that higher-priority tasks can execute in parallel on the processors.

New feasibility tests

Response-time analysis for multiprocessors:

- For the multiprocessor case, with n tasks and m processors, we observe two things:
  1. Interference can only occur when $n > m$.
  2. Interference can only affect the $n - m$ tasks with lowest priority since the m highest-priority tasks will always execute in parallel without contention on the m processors.
- Consequently, interference of a task is a function of the execution overlap of its higher-priority tasks.

New feasibility tests

Response-time analysis for multiprocessors:

- The following two observations give us the secret to analyzing the interference of a task:

  With respect to the execution overlap it can be shown that the interference is maximized when the higher-priority tasks completely overlap their execution.

  Compared to the single-processor case, one extra instance of each higher-priority task must be accounted for in the interference analysis.

  (due to the uncertainty regarding the critical instant).

New feasibility tests

Response-time analysis for multiprocessors:

- The worst-case interference term is

$$I_i = \frac{1}{m} \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j + C_j \right)$$

where $hp(i)$ is the set of tasks with higher priority than τ_i.

- The worst-case response time for a task τ_i is thus:

$$R_i = C_i + \frac{1}{m} \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j + C_j \right)$$

New feasibility tests

Response-time analysis for multiprocessors:

- As before, an iterative approach can be used for finding the worst-case response time:

$$R_i^{n+1} = C_i + \frac{1}{m} \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^n}{T_j} \right\rceil \cdot C_j + C_j \right)$$

- We now have a sufficient condition for static-priority scheduling of periodic tasks on identical processors:

$$\forall i : R_i \leq D_i$$

New feasibility tests

Response-time analysis for multiprocessors:

1. All tasks are independent
2. All tasks are periodic (i.e. not applicable for sporadic tasks)
3. All tasks have identical offsets (= synchronous task set)
4. Task deadline does not exceed the period
   (= constrained-deadline tasks)
5. Task preemptions are allowed
6. All processors are identical
7. Task migrations are allowed

Lecture #15 – blackboard scribble

RTA $R_i^{k+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \cdot C_j \leq D_i$

Consider the decision problem
That is, we terminate
the iterations if a
deadline is missed

Assume that $D_i \leq T_i$ (constrained-deadline tasks)
and that all $O_i = 0$ (synchronous tasks)

1) Show that RTA is a number problem

$O_i < C_i \leq D_i \leq T_i \leq \max_{\forall i} T_i$ Largest number not restricted by n (problem size)

2) Show that RTA has tractable time complexity

Number of iterations per task $\leq D_i \leq T_i \leq \max_{\forall i} T_i$ (largest number)

Total number of iterations (∀ tasks) $\leq n \cdot \max_{\forall i} T_i$

pseudo-polynomial time function of problem size n and
largest number $\max_{\forall i} T_i$

Lecture #15 – blackboard scribble

PDA

Assume that $U < 1$, $D_i \leq T_i$ (constrained-deadline tasks)
and that all $O_i = 0$ (synchronous tasks)

1) Show that PDA is a number problem

$O_i < C_i \leq D_i \leq T_i \leq \max_{\forall i} T_i$ — Largest number, not restricted by n (problem size)

2) Show that PDA has tractable time complexity

Number of control points $\leq L^{\max}_{BRH} = \max\left(1, \frac{U}{1-U}\right) \cdot \max_{\forall i} T_i$

($\max(1, \frac{U}{1-U})$: a constant; $\max T_i$: largest number; $L^{\max}_{BRH}$: see next page)

Total number of iterations ($\forall$ tasks) $\leq n \cdot \max\left(1, \frac{U}{1-U}\right) \cdot \max_{\forall i} T_i$

pseudo-polynomial time function of problem size n and largest number $\max_{\forall i} T_i$

Lecture #15 – blackboard scribble

PDA $\quad L_{BRH} = \max(D_1, D_2, \dots, D_n, L^*)$, where $L^* = \dfrac{\sum_{\forall i} (T_i - D_i) \cdot U_i}{1 - U}$

Note that:

$$\frac{\sum_{\forall i} (T_i - D_i) U_i}{1-U} \leq \frac{\max_{\forall i}(T_i - D_i) \cdot \sum_{\forall i} U_i}{1-U} \leq \frac{\max_{\forall i}(T_i) \cdot \sum_{\forall i} U_i}{1-U} = \frac{U \cdot \max_{\forall i}(T_i)}{1-U}$$

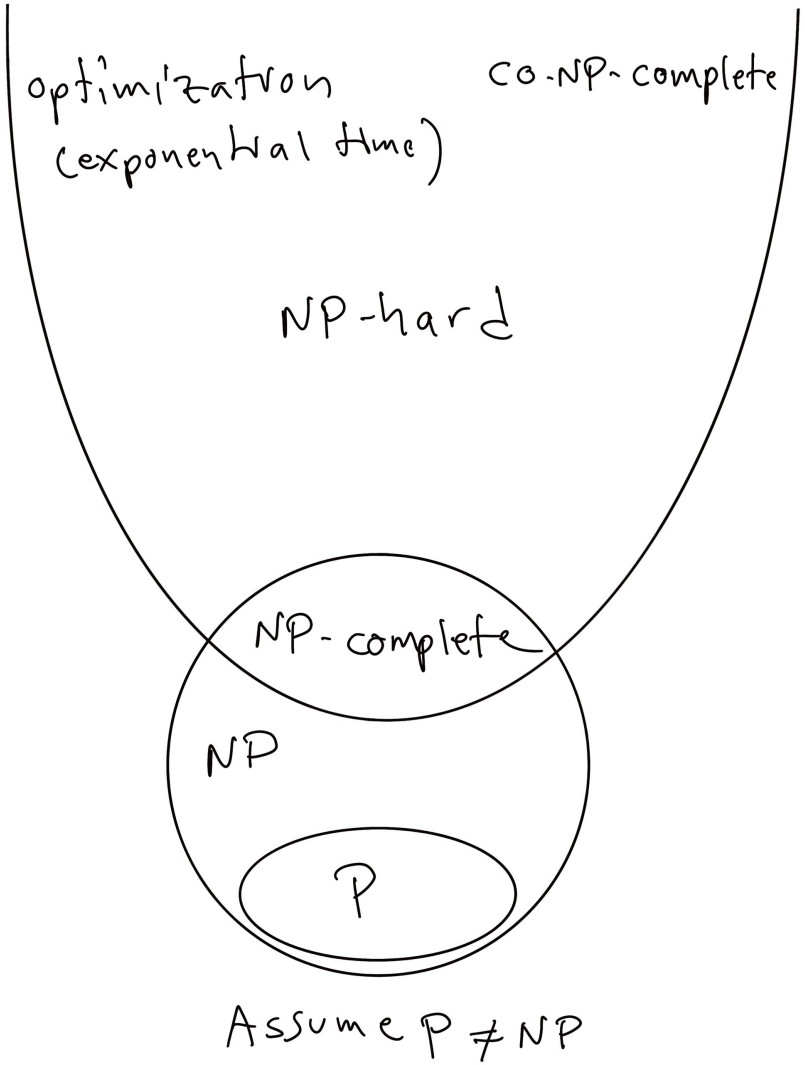
Hence:

$$\max(D_1, D_2, \dots, D_n, L^*) \leq \max\left(\max_{\forall i}(D_i), \frac{U \cdot \max_{\forall i}(T_i)}{1-U}\right) \leq \max\left(\max_{\forall i}(T_i), \frac{U \cdot \max_{\forall i}(T_i)}{1-U}\right)$$

We can now define L^{max}_{BRH}, a useful upper bound of L_{BRH}:

$$L^{max}_{BRH} = \max\left(\max_{\forall i}(T_i), \frac{U \cdot \max_{\forall i}(T_i)}{1-U}\right) = \max\left(1, \frac{U}{1-U}\right) \cdot \max_{\forall i} T_i$$

Lecture #15 – blackboard scribble



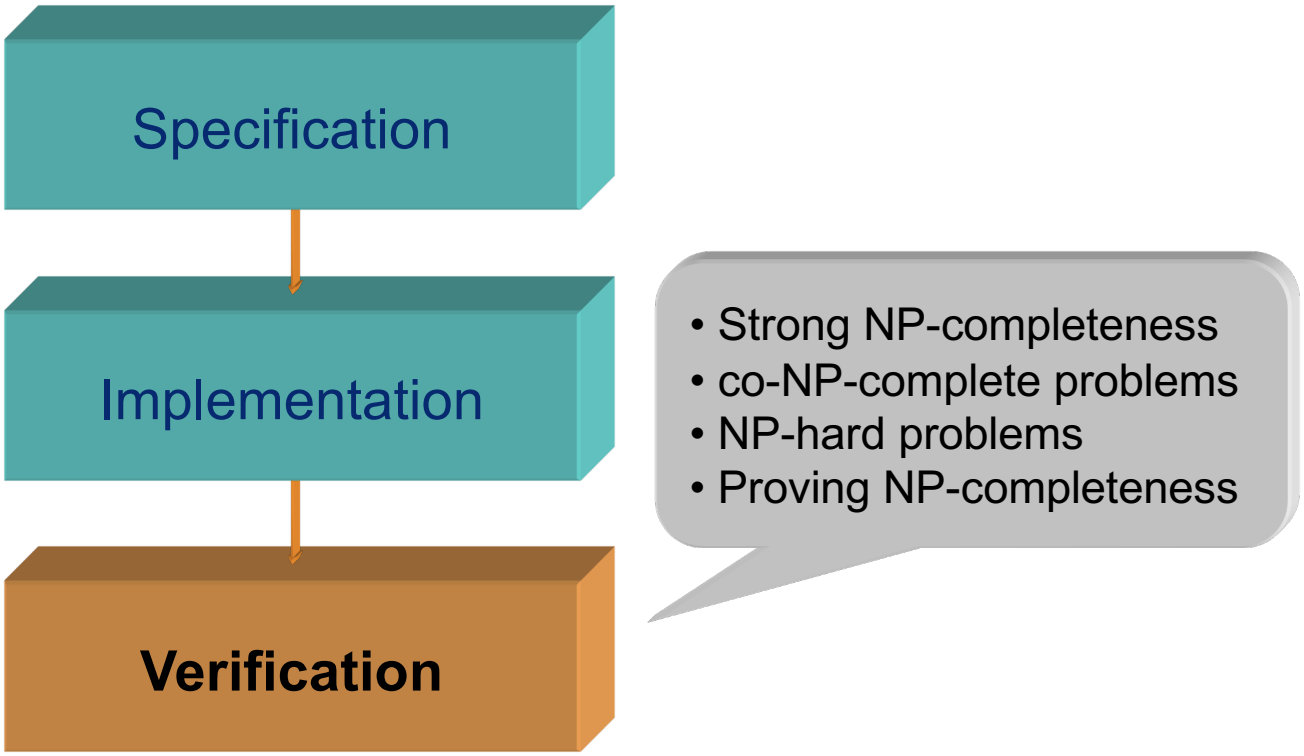
Real-Time Systems

Lecture #15

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



Intractability and NP-completeness

Relationship between P and NP:

1. $P \subseteq NP$
   - Proof: use a polynomial-time deterministic algorithm as the checking stage and ignore the guess ....
2. $P \neq NP$
   - This is a wide-spread belief, but …
   - … no proof of this conjecture exists!

The question of whether or not the NP-complete problems are intractable is now considered to be one of the foremost open questions of contemporary mathematics and computer science!

Strong NP-completeness

Pseudo-polynomial time complexity:

- Number problems
  - This is a special type of NP-complete problems for which the largest number (parameter value) in a problem instance is not bounded by the input length (size) of the problem.
- Number problems are often quite tractable
  - If the time complexity of a number problem can be shown to be a polynomial-time function of both the input length and the largest number, that number problem is said to have pseudo-polynomial time complexity.

    That is, the time-complexity function is proportional to $p(max,n)$ for some polynomial function p, where max is the largest number and n is the input length.

Strong NP-completeness

If a decision problem Π is NP-complete and is not a number problem, then it cannot be solved by a pseudo-polynomial-time algorithm unless P = NP.

↓

Assuming P $\neq$ NP, the only NP-complete problems that are potential candidates for being solved by pseudo-polynomial-time algorithms are those that are number problems.

↓

A decision problem Π which cannot be solved by a pseudo-polynomial-time algorithm, unless P = NP, is said to be NP-complete in the strong sense.

Strong NP-completeness

NP-complete problems that are number problems ...

- ... but are NP-complete in the strong sense regardless
  - Multiprocessor preemptive scheduling (partitioned and global)
  - Single-processor non-preemptive scheduling
  - 3-Partition, Simultaneous Congruences, Traveling Salesman

NP-complete problems that are number problems …

- ... and that do have pseudo-polynomial time complexity
  - Single-processor scheduling of synchronous constrained-deadline tasks with static priorities (using response-time analysis)
  - Single-processor scheduling of synchronous constrained-deadline tasks with dynamic priorities (using processor-demand analysis)

Strong NP-completeness

Proving pseudo-polynomial time complexity:

- It is quite easy to prove that response-time analysis (RTA) has pseudo-polynomial time complexity.
  - First show that RTA is a number problem
  - Then show that RTA has tractable time complexity
- It takes a little more effort to prove that processor-demand analysis (PDA) has pseudo-polynomial time complexity.
  - In the general case PDA may have exponential time complexity (due to length of the hyper period)
  - However, for the restricted case where task utilization $U < 1$ PDA can be shown to have tractable time complexity.

We now study the proofs for the RTA and PDA cases.

(see blackboard scribble)

Co-NP-complete problems

Class co-NP:

- Complement problem:
  - The complement of a decision problem Π is the problem Π^C having the same solution domain as Π, but with the outcome from solving the problem logically reversed.
  - That is, given the same problem instance, a "yes" outcome from solving problem Π would imply a "no" outcome from solving problem Π^C (and vice versa)

A decision problem $\Pi \in$ co-NP if and only if its complement problem $\Pi^C \in$ NP.

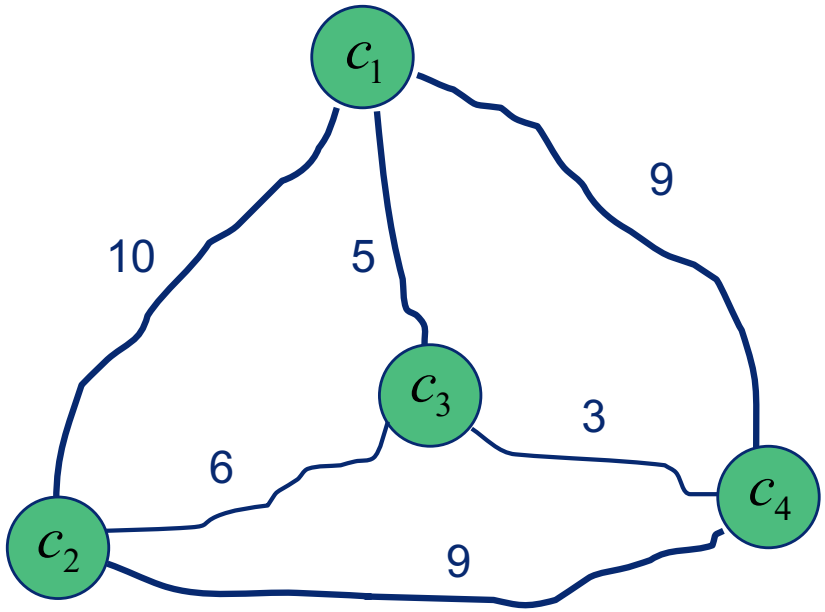
Co-NP-complete problems

NP vs co-NP:

- Problems in NP
  - The class of problems for which there exists a polynomial-time algorithm that can verify a solution that makes the binary problem statement true (“yes” outcome).
- Problems in co-NP
  - The class of problems for which there exists a polynomial-time algorithm that can verify a counterexample solution that makes the binary problem statement false (“no” outcome).
- Co-NP-complete problems
  - Decision problems for which it applies that their complement problem is an NP-complete problem.

Co-NP-complete problems

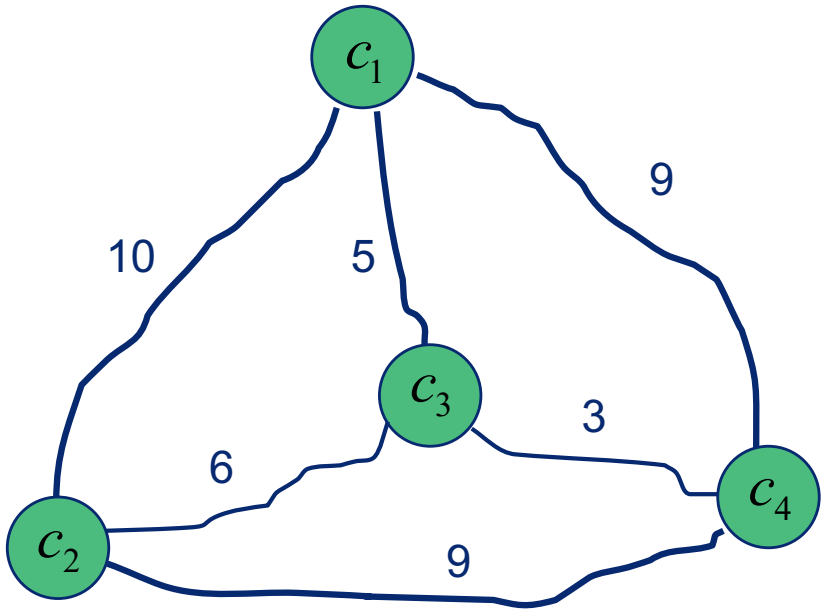
The Original Traveling Salesman Problem:



Does there exist one "tour" of all the cities in C having a total length of no more than B?

Co-NP-complete problems

The Complement Traveling Salesman Problem:



Does every "tour" of all the cities in C have a total length that exceeds B?

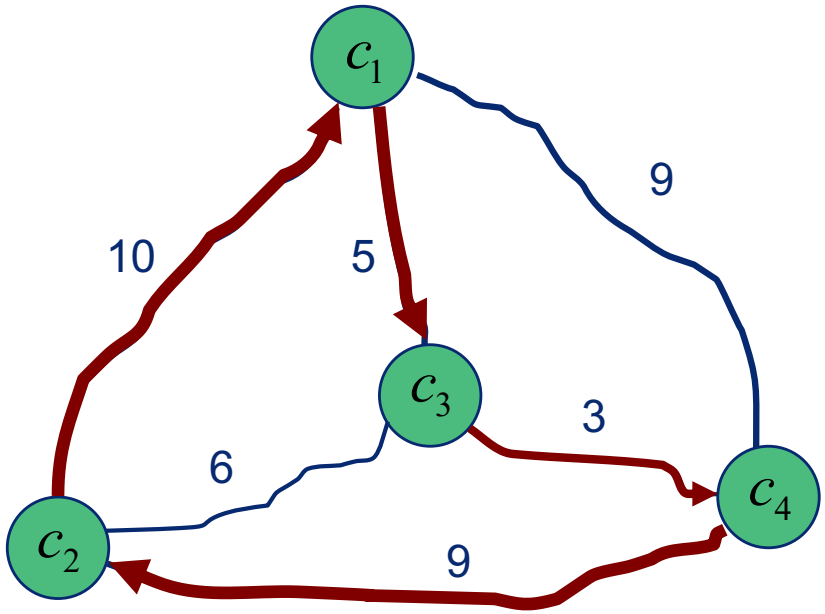
Co-NP-complete problems

The Complement Traveling Salesman Problem:

- Verifying a “yes” outcome
  - Requires checking that all possible solutions (”tours”) to the problem instance fulfills the problem statement. Can in general only be done in exponential time (need to show that every possible “tour” length > B).
- Verifying a “no” outcome
  - Requires checking that one solution (the counterexample “tour”) to the problem instance does not fulfill the problem statement. Can be done in polynomial time (only need to show that the counterexample “tour” length ≤ B).
  - This corresponds exactly to verifying a “yes” outcome in the original Traveling Salesman Problem (which is NP-complete).

Co-NP-complete problems

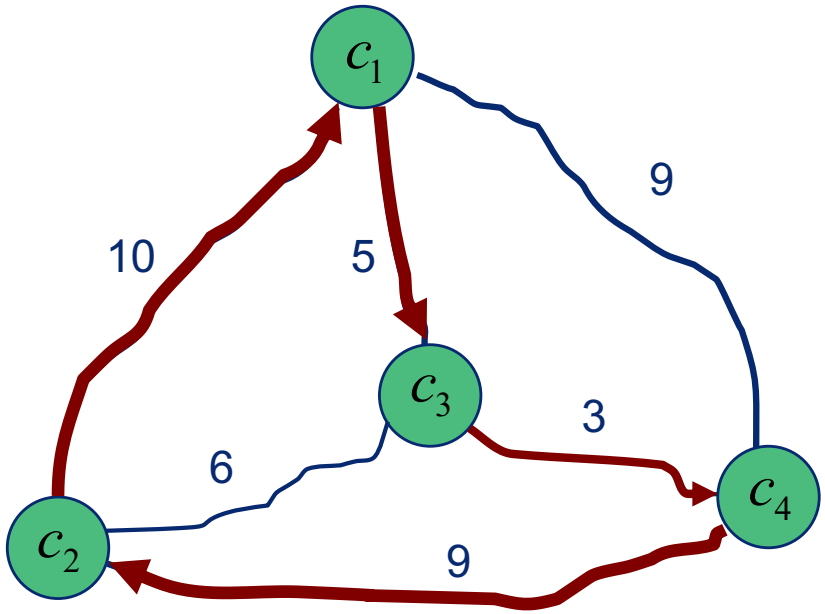
The Complement Traveling Salesman Problem:



Does every "tour" of all the cities in C have a total length that exceeds **30**? No! One has length 27.

Co-NP-complete problems

The Original Traveling Salesman Problem:



Does there exist one "tour" of all the cities in C having a total length of no more than **30**? Yes! One has length 27.

Co-NP-complete problems

Why is co-NP relevant for scheduling problems?

- The original scheduling problem formulation:
  - Does there exist one schedule for the given task set and run-time system that is feasible?
- The complement scheduling problem formulation:
  - Is every schedule for the given task set and run-time system infeasible?
- Reducibility of scheduling problems:
  - For many well-known scheduling problems it turns out that it is only the complement problem that can be reduced to a known NP-complete problem.

Co-NP-complete problems

The Complement Scheduling Problem:

- Verifying a “yes” outcome
  - Requires checking that all possible solutions (“schedules”) will contain at least one missed deadline. Can in general only be done in exponential time (need to show that every possible “schedule” is infeasible).
- Verifying a “no” outcome
  - Only requires checking that one solution (the counterexample “schedule”) will meet all deadlines. Can be done in polynomial time (show that the counterexample “schedule” is feasible).
  - This corresponds exactly to verifying a “yes” outcome in the original Scheduling Problem.

NP-hard problems

NP-hard problems:

Problems that are “at least as hard” as the hardest problems in class NP.

NP-hard problems

Turing reducibility:

- A problem Π' is Turing reducible to problem Π if there exists an algorithm A that solves Π' by using a hypothetical subroutine S for solving Π such that, if S were a polynomial time algorithm for Π, then A would be a polynomial time algorithm for Π' as well.

  When Π' is Turing reducible to Π, we write $\Pi' \propto_T \Pi$

A search problem Π is said to be NP-hard if there exists some NP-complete problem Π' that Turing-reduces to Π.

NP-hard problems

Observations:

- All NP-complete problems are NP-hard
- All co-NP-complete problems are NP-hard

  Turing reduction from Π to Π^C (and vice versa) is trivial.

- Given an NP-complete decision problem, the corresponding optimization problem is NP-hard

  To see this, imagine that the optimization problem (that is, finding the optimal cost) could be solved in polynomial time.

  The corresponding decision problem (i.e., determining whether there exists a solution with a cost no more than B) could then be solved by simply comparing the found optimal cost to the bound B. This comparison is a constant-time operation.

Reference only, not examined

History of NP-completeness

S. Cook: (1971)

"The Complexity of Theorem Proving Procedures"

Every problem in the class NP of decision problems polynomially reduces to the SATISFIABILITY problem.

R. Karp: (1972)

"Reducibility among Combinatorial Problems"

Decision problem versions of many well-known combinatorial optimization problems are "just as hard" as SATISFIABILITY.

Reference only, not examined

History of NP-completeness

D. Knuth: (1974)

"A Terminological Proposal"

Initiated a researcher's poll in search of a better term for "at least as hard as the polynomial complete problems".

The winning suggestion was "NP-complete" problems.

One (rejected, but smart) suggestion was "PET" problems:

- "Probably Exponential Time" (if P = NP remain open question)
- "Provably Exponential Time" (if $P \neq NP$)
- "Previously Exponential Time" (if P = NP)

History of NP-completeness

SATISFIABILITY: The original NP-complete problem

- Variables and literals
  - Let U be a set of Boolean variables.
  - If u is a variable in U then u and u' are literals over U.
- Conjunctive normal form
  - A formula is in conjunctive normal form (CNF) if it is a conjunction of one or more clauses, where a clause is a disjunction of literals.
- SATISFIABILITY question:
  - Given a formula in CNF does there exist a truth assignment for the variables in U that yields a True statement?

Reference only, not examined

History of NP-completeness

Proving NP-completeness

Proving NP-completeness for a decision problem Π:

1. Show that Π is in NP
2. Select a known NP-complete problem Π'
3. Construct a transformation $\propto$ from Π' to Π
4. Prove that $\propto$ is a (polynomial) transformation

The book "Computers and Intractability – A Guide to the Theory of NP-Completeness" (Garey and Johnson, 1979) contains a categorized list of 300+ NP-complete problems, with problem statements and how each problem was proven NP-complete.

Proving NP-completeness

Transformations for real-time scheduling problems:

In published results regarding the time complexity of known real-time scheduling problems, the following NP-complete problems are predominantly used for the transformations:

- 3-PARTITION
  - NP-complete in the strong sense.
  - Used in the proofs for multiprocessor scheduling, and in the proof for non-preemptive single-processor scheduling.
- SIMULTANEOUS CONGRUENCES (SCP)
  - NP-complete in the strong sense.
  - Used in the proofs for preemptive single-processor scheduling, by employing a reverse logic (co-NP) strategy.

Proving NP-completeness

3-PARTITION decision problem:

- Set of elements
  - Let $A = \{a_1, \ldots, a_{3m}\}$ be a set of $3m$ elements.
  - Each element $a_i \in A$ has a positive integer "size" $s(a_i)$.
- Element size constraints using a bound
  - Let B be a positive integer.
  - Each $s(a_i)$ satisfies $B/4 < s(a_i) < B/2$ and $\sum_{a_i \in A} s(a_i) = mB$.
- Question:
  - Can A be partitioned into m disjoint sets $S_1, \ldots, S_m$ such that, for each $1 \leq j \leq m$, it applies that $\sum_{a_i \in S_j} s(a_i) = B$?

Note: constraints dictate that each disjoint set must contain exactly 3 elements!

Proving NP-completeness

SIMULTANEOUS CONGRUENCES decision problem:

- Set of ordered pairs
  - Let $A = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ be a set of n ordered pairs.
  - Each pair $(a_i, b_i) \in A$ consists of positive integers.
- Minimum bound
  - Let B be a positive integer, such that $2 \leq B \leq n$.
- Question:
  - Does there exist a subset $A' \in A$ of at least B pairs and a positive integer x such that, for each $(a_i, b_i) \in A'$, it applies that $x \equiv a_i \pmod{b_i}$?

Note: $x \equiv a_i \pmod{b_i}$ means $x = a_i + k_i \cdot b_i$ for some non-negative integer k_i

Proving NP-completeness

Proving NP-completeness for a decision problem Π:

1. Show that Π is in NP
2. Select a known NP-complete problem Π'
3. Construct a transformation $\propto$ from Π' to Π
4. Prove that $\propto$ is a (polynomial) transformation

Example: the proof by Jeffay, Stanat and Martel (1991)

Transformation from 3-PARTITION is used in proving strong NP-completeness for non-preemptive single-processor EDF scheduling of asynchronous, periodic, implicit-deadline tasks (Theorem 5.2)

Proving NP-completeness

General complexity of real-time scheduling:

- Any type of scheduling of periodic tasks
  - NP-hard (exponential time)
- Any type of non-preemptive scheduling
  - NP-complete in the strong sense (reduction from 3-PARTITION)
- Any type of preemptive multiprocessor scheduling
  - NP-complete in the strong sense (reduction from 3-PARTITION)
  - Note: applies to both partitioned and global approaches
- Preemptive single-processor scheduling of asynchronous tasks
  - Co-NP-complete in the strong sense (reduction from SCP)

Proving NP-completeness

Complexity of preemptive single-processor scheduling:

- Scheduling of synchronous tasks w/ dynamic task priorities
  - Co-NP-complete in the strong sense (reduction from SCP)
  - Co-NP-complete in the weak ("normal") sense for $U < 1$
  - Special cases:
    - Pseudo-polynomial time for constrained-deadline tasks for $U < 1$
    - Polynomial time for implicit-deadline tasks
- Scheduling of synchronous tasks w/ static task priorities
  - NP-hard for arbitrary-deadline tasks (exponential time)
  - NP-complete in the weak sense for constrained-deadline tasks
  - Special cases:
    - Pseudo-polynomial time for constrained-deadline tasks
    - Polynomial time for implicit-deadline tasks for $U \leq \ln 2$

Lecture #16 – blackboard scribble

1a) Processor utilization analysis (RM)

Task	C_i	D_i	T_i
τ_1	3	20	20
τ_2	10	30	30
τ_3	25	60	60

$D_i = T_i$

$$U = \frac{3}{20} + \frac{10}{30} + \frac{25}{60} = 0{,}15 + 0{,}333 + 0{,}417 = 0{,}90$$

$$U_{RM(3)} = n\left(2^{1/n} - 1\right) = \{n=3\} = 3\left(2^{1/3} - 1\right) \approx 0{,}78$$

$$U > U_{RM(3)} \Rightarrow \text{Test fails!}$$

Because the test is only sufficient the schedulability of the task set cannot be determined

Lecture #16 – blackboard scribble

1 b) Response-time analysis (DM)

The final (converged) response times should be calculated for each task, regardless of whether the analysis fails or not!

	Task	C_i	D_i	T_i
H	τ_1	3	5	20
M	τ_2	10	25	30
L	τ_3	25	40	60

$R_1 = C_1 = 3 \leq D_1 = 5 \Rightarrow$ ok!

$R_2 = C_2 + \left\lceil \frac{R_2}{T_1} \right\rceil C_1$ [Assume $R_2^0 = C_2 = 10$]

$R_2^1 = 10 + \left\lceil \frac{10}{20} \right\rceil \cdot 3 = 10 + 1 \cdot 3 = 13$

$R_2^2 = 10 + \left\lceil \frac{13}{20} \right\rceil \cdot 3 = 10 + 1 \cdot 3 = 13$

Convergence $\leq D_2 = 25 \Rightarrow$ ok!

Lecture #16 – blackboard scribble

1b) (cont'd)

Task	C_i	D_i	T_i
τ_1	3	5	20
τ_2	10	25	30
τ_3	25	40	60

$$R_3 = C_3 + \left\lceil \frac{R_3}{T_1} \right\rceil \cdot C_1 + \left\lceil \frac{R_3}{T_2} \right\rceil \cdot C_2 \qquad [\text{Assume } R_3^0 = C_3 = 25]$$

$$R_3^1 = 25 + \left\lceil \frac{25}{20} \right\rceil \cdot 3 + \left\lceil \frac{25}{30} \right\rceil \cdot 10 = 25 + 2 \cdot 3 + 1 \cdot 10 = 41$$

$$R_3^2 = 25 + \left\lceil \frac{41}{20} \right\rceil \cdot 3 + \left\lceil \frac{41}{30} \right\rceil \cdot 10 = 25 + 3 \cdot 3 + 2 \cdot 10 = 54$$

$$R_3^3 = 25 + \left\lceil \frac{54}{20} \right\rceil \cdot 3 + \left\lceil \frac{54}{30} \right\rceil \cdot 10 = 25 + 3 \cdot 3 + 2 \cdot 10 = 54$$

Convergence

$> D_3 = 40$ FAIL!

Test for τ_3 fails as response time exceeds deadline.

Because the test is exact the task set is not schedulable.

Note that, although the analysis failed already at R_3^1, this problem asked for the final (converged) response time values.

Lecture #16 – blackboard scribble

Task	C_i	D_i	T_i
τ_1	3	5	20
τ_2	10	25	30
τ_3	25	40	60

1c) Determine L_{max} (largest interval)

$$U = U_1 + U_2 + U_3 = 0,15 + 0,333 + 0,417 = 0,9$$

Since $U < 1$: $L_{max} = \min(L_{BRH}, L_{LCM})$

$$L_{BRH} = \max\left\{D_1, D_2, D_3, \frac{\sum_{i=1}^{3}(T_i - D_i)\cdot U_i}{1-U}\right\}$$

$(T_1 - D_1)\cdot U_1 = 15 \cdot 0,15 = 2,25$

$(T_2 - D_2)\cdot U_2 = 5 \cdot 0,333 = 1,667$

$(T_3 - D_3)\cdot U_3 = 20 \cdot 0,417 = 8,333$

$$L^* = \frac{\sum (T_i - D_i)U_i}{1-U} = \frac{2,25 + 1,667 + 8,333}{1-0,9} = \frac{12,25}{0,1} = 122,5 \leq 123$$

$$L_{BRH} = \max\{D_1, D_2, D_3, L^*\} = \max\{5, 25, 40, 123\} = 123$$

$$L_{LCM} = LCM\{T_1, T_2, T_3\} = LCM\{20, 30, 60\} = 60$$

$$L_{max} = \min(L_{BRH}, L_{LCM}) = \min(123, 60) = 60$$

Lecture #16 – blackboard scribble

1 d) Control-point calculation for all tasks

$K_1 = \{5, 25, 45\}$ $K_2 = \{25, 55\}$ $K_3 = \{40\}$

$K = \{5, 25, 40, 45, 55\}$

Task	C_i	D_i	T_i
τ_1	3	5	20
τ_2	10	25	30
τ_3	25	40	60

1 e) Processor-demand analysis for the given task set.

Analysis should be performed for every control point, regardless of whether the analysis in another control point fails or not.

Lecture #16 – blackboard scribble

L	$N_1^L \cdot C_1$	$N_2^L \cdot C_2$	$N_3^L \cdot C_3$	$C_p(0,L)$	$C_p(0,L) \leq L$?
5	$\left(\left\lfloor \frac{5-5}{20} \right\rfloor + 1\right) \cdot 3 = 3$	$\left(\left\lfloor \frac{5-25}{30} \right\rfloor + 1\right) \cdot 10 = 0$	$\left(\left\lfloor \frac{5-40}{60} \right\rfloor + 1\right) \cdot 25 = 0$	$3+0+0=3$	OK!
25	$\left(\left\lfloor \frac{25-5}{20} \right\rfloor + 1\right) \cdot 3 = 6$	$\left(\left\lfloor \frac{25-25}{30} \right\rfloor + 1\right) \cdot 10 = 10$	$\left(\left\lfloor \frac{25-40}{60} \right\rfloor + 1\right) \cdot 25 = 0$	$6+10+0=16$	OK!
40	$\left(\left\lfloor \frac{40-5}{20} \right\rfloor + 1\right) \cdot 3 = 6$	$\left(\left\lfloor \frac{40-25}{30} \right\rfloor + 1\right) \cdot 10 = 10$	$\left(\left\lfloor \frac{40-40}{60} \right\rfloor + 1\right) \cdot 25 = 25$	$6+10+25=41$	FAIL!
45	$\left(\left\lfloor \frac{45-5}{20} \right\rfloor + 1\right) \cdot 3 = 9$	$\left(\left\lfloor \frac{45-25}{30} \right\rfloor + 1\right) \cdot 10 = 10$	$\left(\left\lfloor \frac{45-40}{60} \right\rfloor + 1\right) \cdot 25 = 25$	$9+10+25=44$	OK!
55	$\left(\left\lfloor \frac{55-5}{20} \right\rfloor + 1\right) \cdot 3 = 9$	$\left(\left\lfloor \frac{55-25}{30} \right\rfloor + 1\right) \cdot 10 = 20$	$\left(\left\lfloor \frac{55-40}{60} \right\rfloor + 1\right) \cdot 25 = 25$	$9+20+25=54$	OK!

Remember that all control points need to be analyzed for this problem, despite the failure at t=40!

Test for L=40 fails as processor demand exceeds interval length.
Because the test is exact the task set is not schedulable.

Real-Time Systems

Lecture #16

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems

Facing the written exam

Monday, March 17, 2025 @ 08:30−12:30

at Johanneberg campus

Note: in case you need to take a re-exam in August, please note that it takes place at Lindholmen campus

Facing the exam

Permitted to use during the written exam:

- The standard authorised aids
  - Pencils, erasers, rulers and dictionaries
  - Markings or notes are not permitted in the authorised aids.
  - Electronic dictionaries may not be used during the exam
- Chalmers-approved calculator
  - Approved models:

    Casio FX-82, Casio FX-85, Sharp EL-W531, Texas TI-30

Please observe: as of 2022 the compendium "Programming with the TinyTimber kernel" is no longer an authorised aid!

Facing the exam

Reading guidelines:

- Lecture and Exercise notes
  - PowerPoint hand-outs + blackboard scribble
  - All material is very relevant, and may be examined
- Excerpts from research articles and books:
  - Recommended reading, but will not be examined
- Exercise compendium
  - Recommended problem solving …
- Old written exams
  - For inspiration …

Facing the exam

Important knowledge areas:

- Design principles for real-time systems
  - Real-time systems: typical properties, misconceptions
  - Real-time constraints: origin, interpretation (soft/hard)
  - Design phases: specification, implementation, verification
  - Verification: methods, difficulties, pitfalls
  - Network communication: methods (CAN in particular)

Facing the exam

Important knowledge areas (cont'd):

- Principles of concurrent programming
  - Paradigm: reactive (event driven) programming
  - Parallelization: pros & cons
  - Resource management: mutual exclusion, critical region
  - Deadlock: definition, management
  - Starvation: definition, management

Facing the exam

Important knowledge areas (cont'd):

- Language support for concurrent programming
  - Mutual exclusion: protected objects, monitors, semaphores, synchronized methods, mutex'd methods
  - Machine-level mutual exclusion: disable interrupt, test-and-set
- Language support for real-time programming
  - Units of concurrency: task, thread, method
  - Scheduling support: clocks, time, delays, priorities
  - Device drivers: interrupts handlers, call-back functionality
  - TinyTimber: "how it's done"
  - WCET: purpose, required properties, analysis methods

Facing the exam

Important knowledge areas (cont'd):

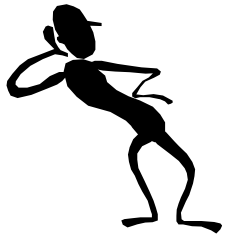
- Scheduling theory
  - Task model: WCET, deadline, period, offset
  - Scheduling: definitions, priorities, preemption
  - Feasibility tests: purpose, exactness (sufficient/necessary)
  - Complexity theory: time complexity, NP-completeness
- Scheduling with cyclic executives
  - Properties: time table, pros & cons
  - Scheduling: generation of time tables, run-time behavior
  - Feasibility test: hyper-period analysis

Facing the exam

Important knowledge areas (cont'd):

- Scheduling with pseudo-parallel execution
  - Properties: priority assignment, optimality, pros & cons
  - Scheduling: run-time behavior, construct timing diagram
  - Feasibility test: theory, assumptions, exactness, complexity
  - RM, DM, EDF: "how it's done"
  - RMFF, RM-US: "how it's done"

Facing the exam



What type of exam problems will there be?

- **Basic** part

  Will probe your skills in performing fundamental calculations relating to single-processor scheduling theory.

  Minimum requirement: to pass the exam (= **grade 3 [G]**) you must obtain ≥ 24 points (out of 30) in the Basic part. Chalmers [GU]

- **Advanced** part

  Will encompass problem solving and scheduling theory insights

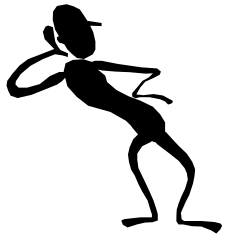
  To get **grade 4** you must obtain ≥ 24 points in the Basic part and ≥ 36 points (out of 60) in both parts combined. Chalmers

  To get **grade 5 [VG]** you must obtain ≥ 24 points in the Basic part and ≥ 48 [44] points (out of 60) in both parts combined. Chalmers [GU]

  Note: Your submitted solutions for the Advanced part will only be graded if you have obtained ≥ 24 points in the Basic part.

Facing the exam

What type of exam problems will there be?



- **Basic** part

  Fundamental calculations relating to single-processor scheduling

  The following types of problems will appear:

  - Perform utilization-based schedulability analysis for a task set
  - Perform worst-case response-time analysis for a task set
  - Perform processor-demand analysis for a task set
  - Draw a timing diagram based on simulation of on-line scheduler

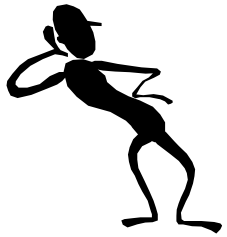
  Will not include unknown parameter values or blocking factors.

  In addition, problems relating to the following topics will appear:

  - WCET analysis, scheduling concepts, real-time programming

Facing the exam

What type of exam problems will there be?



- **Advanced** part

  In-depth problem solving and scheduling theory insights

  The following types of problems will appear:

  - Single-processor scheduling for a task set (may include unknown parameter values and/or blocking factors)
  - Multiprocessor scheduling for a task set (may include unknown parameter values)
  - Scheduling theory problems that will probe your knowledge regarding feasibility testing, optimality and complexity theory

Facing the exam

Deriving the final grade in the course:

The final grade (U, 3, 4, 5) in the course will reflect your laboratory skills as well as your theoretical skills.

The final grade is influenced almost equally* by:

- Your results in course element ‘Laboratory’
  - based on a ‘Pass’ grade (3, 4, 5)
- Your results in course element ‘Examination’
  - based on a ‘Pass’ written exam score (24−60 points)

(* See corresponding look-up table in Canvas for details)

Note: GU students use Chalmers grading scale within Canvas, but get corresponding GU grades in Ladok.