CHALMERS

Real-Time Systems

7.5 credit points

Docent Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

CHALMERS

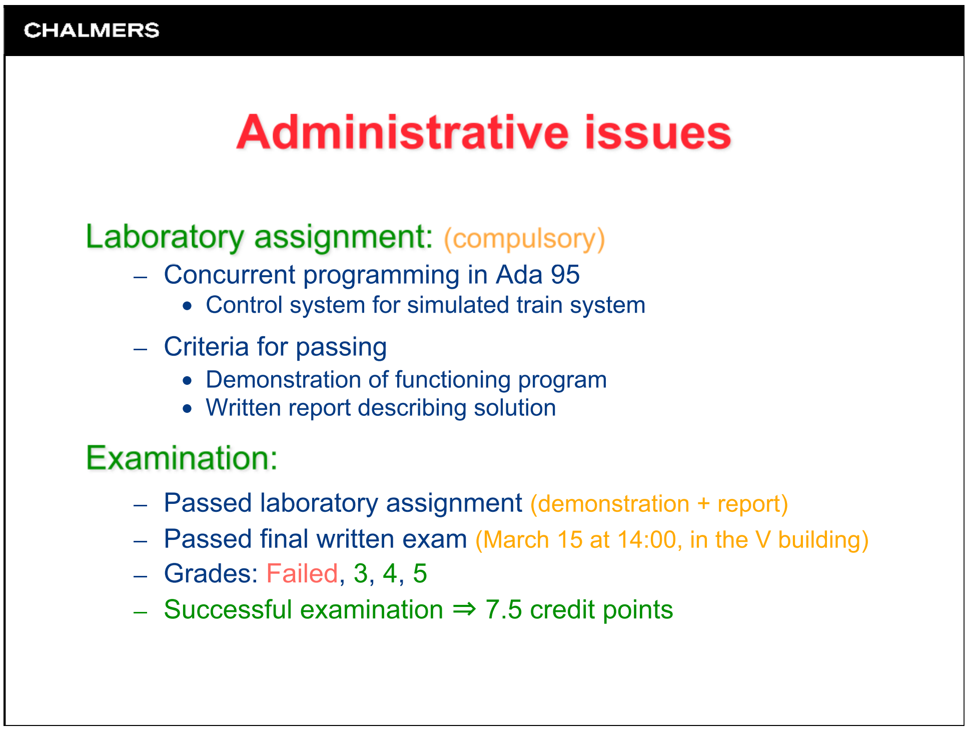
Administrative issues

Lectures: (Jan Jonsson + special guests)

- Fundamental methods and theory
  - Real-time programming, run-time systems and scheduling
- 16 classroom lectures
  - Tuesday at 10:00 – 11:45 in lecture room HA3
  - Wednesday at 08:00 – 09:45 in lecture room HA3 (week 1 & 2 only)
  - Thursday at 13:15 – 15:00 in lecture room HA3

Exercise sessions: (Risat Pathan)

- Complementary lectures in programming and theory
  - Programming language Ada 95 and laboratory assignment
  - Programming in Ada 95 and scheduling theory
- Seven exercise sessions
  - Thursday at 15:15 – 17:00 in lecture room HA3

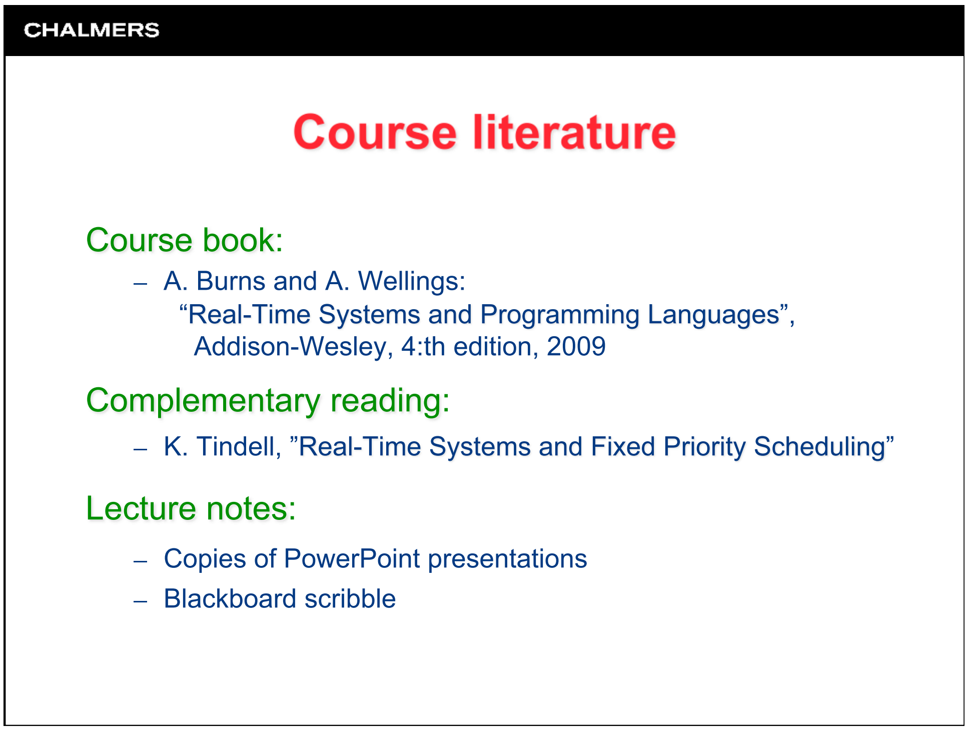


CHALMERS

Administrative issues

Laboratory assignment: (compulsory)
- Concurrent programming in Ada 95
  - Control system for simulated train system
- Criteria for passing
  - Demonstration of functioning program
  - Written report describing solution

Examination:
- Passed laboratory assignment (demonstration + report)
- Passed final written exam (March 15 at 14:00, in the V building)
- Grades: Failed, 3, 4, 5
- Successful examination ⇒ 7.5 credit points



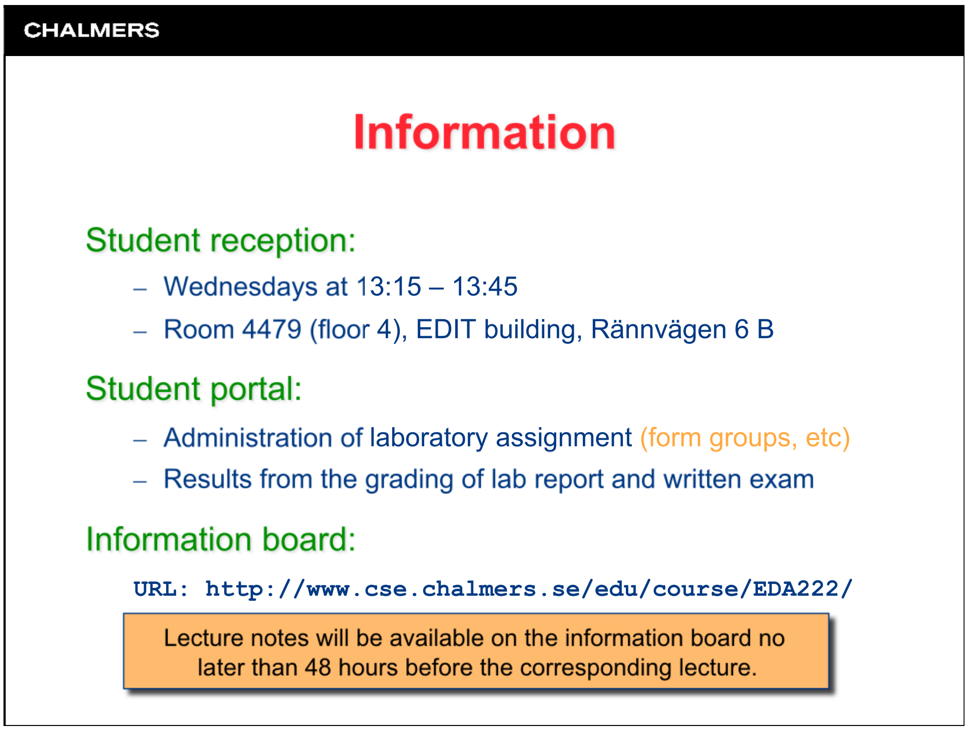
CHALMERS

Course literature

Course book:
- A. Burns and A. Wellings:
  "Real-Time Systems and Programming Languages",
  Addison-Wesley, 4:th edition, 2009

Complementary reading:
- K. Tindell, "Real-Time Systems and Fixed Priority Scheduling"

Lecture notes:
- Copies of PowerPoint presentations
- Blackboard scribble



CHALMERS

Information

Student reception:

- Wednesdays at 13:15 – 13:45
- Room 4479 (floor 4), EDIT building, Rännvägen 6 B

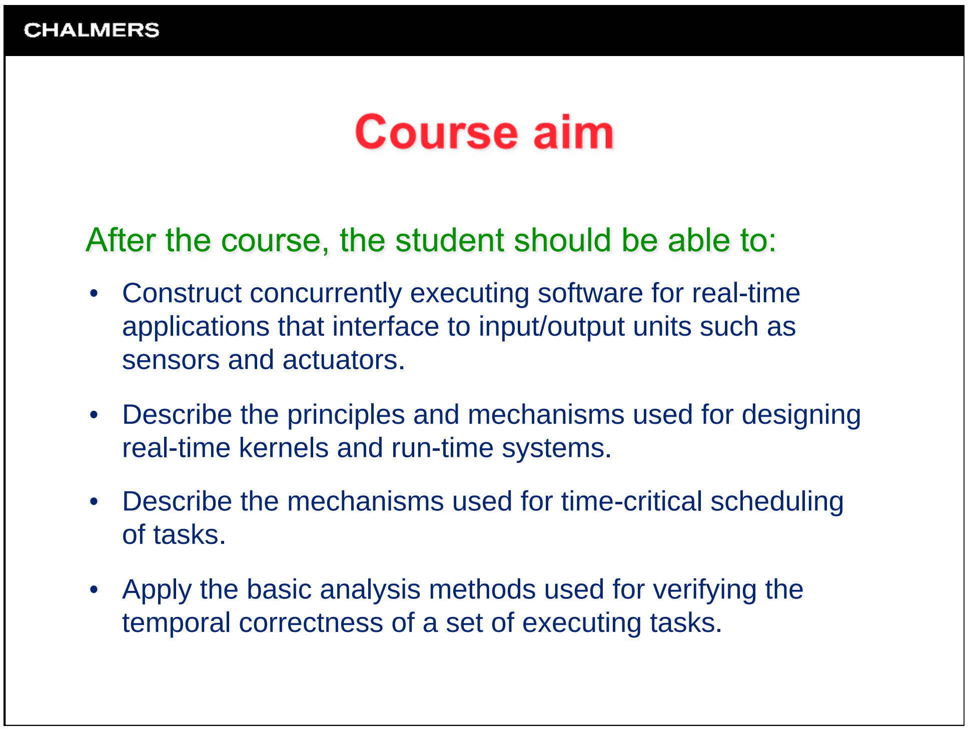
Student portal:

- Administration of laboratory assignment (form groups, etc)
- Results from the grading of lab report and written exam

Information board:

`URL: http://www.cse.chalmers.se/edu/course/EDA222/`

Lecture notes will be available on the information board no later than 48 hours before the corresponding lecture.

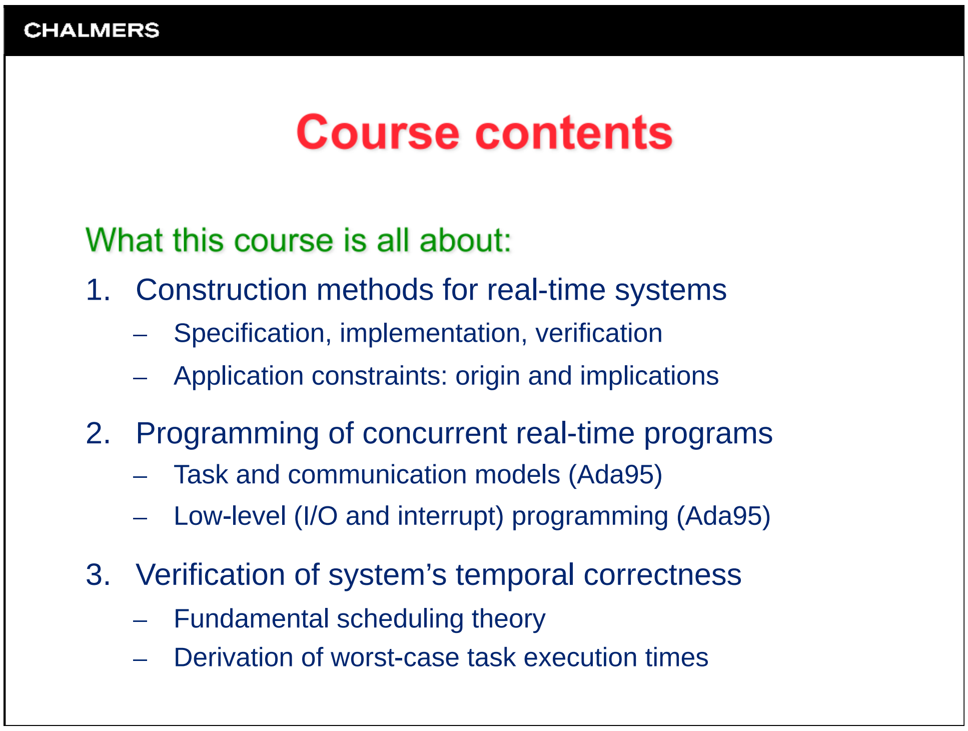


CHALMERS

Course aim

After the course, the student should be able to:

- Construct concurrently executing software for real-time applications that interface to input/output units such as sensors and actuators.
- Describe the principles and mechanisms used for designing real-time kernels and run-time systems.
- Describe the mechanisms used for time-critical scheduling of tasks.
- Apply the basic analysis methods used for verifying the temporal correctness of a set of executing tasks.

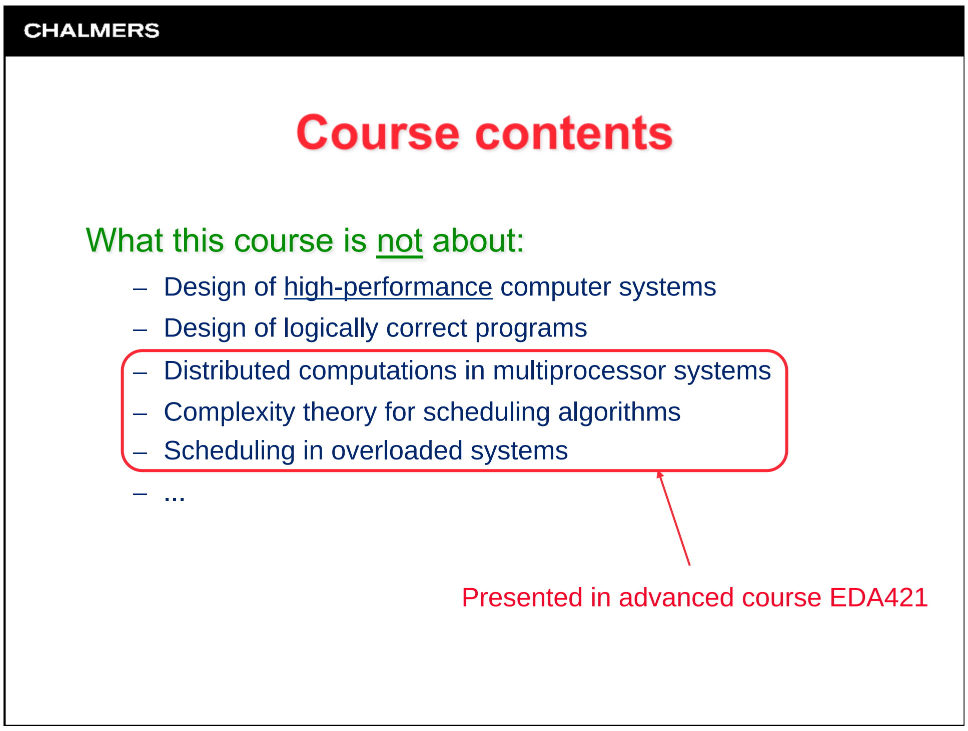


CHALMERS

Course contents

What this course is all about:

1. Construction methods for real-time systems
   - Specification, implementation, verification
   - Application constraints: origin and implications
2. Programming of concurrent real-time programs
   - Task and communication models (Ada95)
   - Low-level (I/O and interrupt) programming (Ada95)
3. Verification of system's temporal correctness
   - Fundamental scheduling theory
   - Derivation of worst-case task execution times



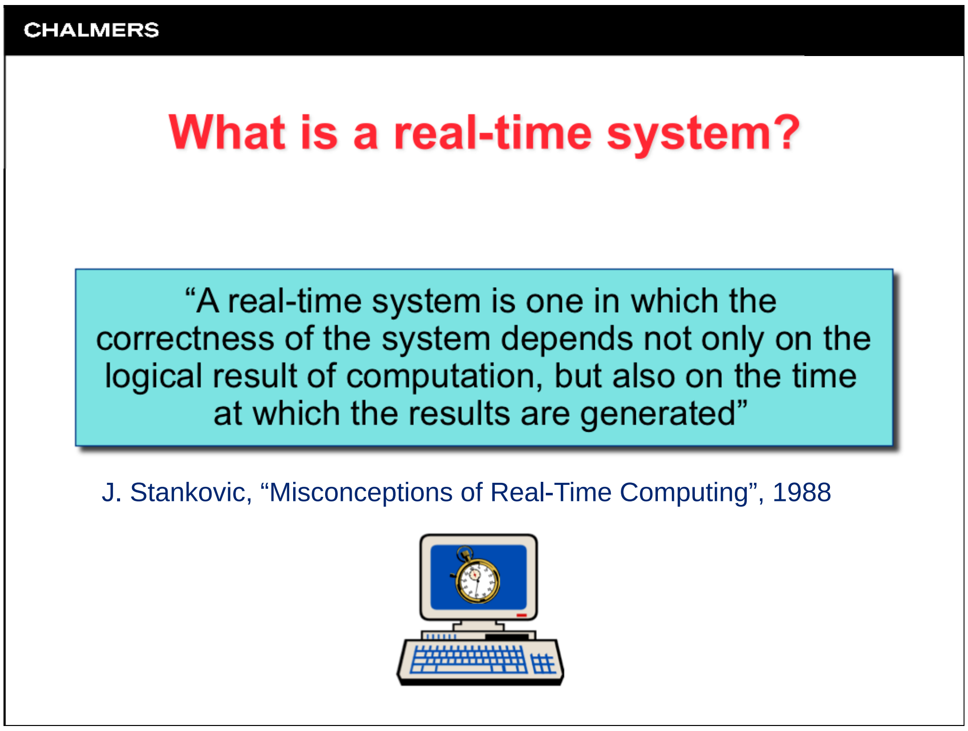
CHALMERS

Course contents

What this course is not about:

- Design of high-performance computer systems
- Design of logically correct programs
- Distributed computations in multiprocessor systems
- Complexity theory for scheduling algorithms
- Scheduling in overloaded systems
- ...

Presented in advanced course EDA421

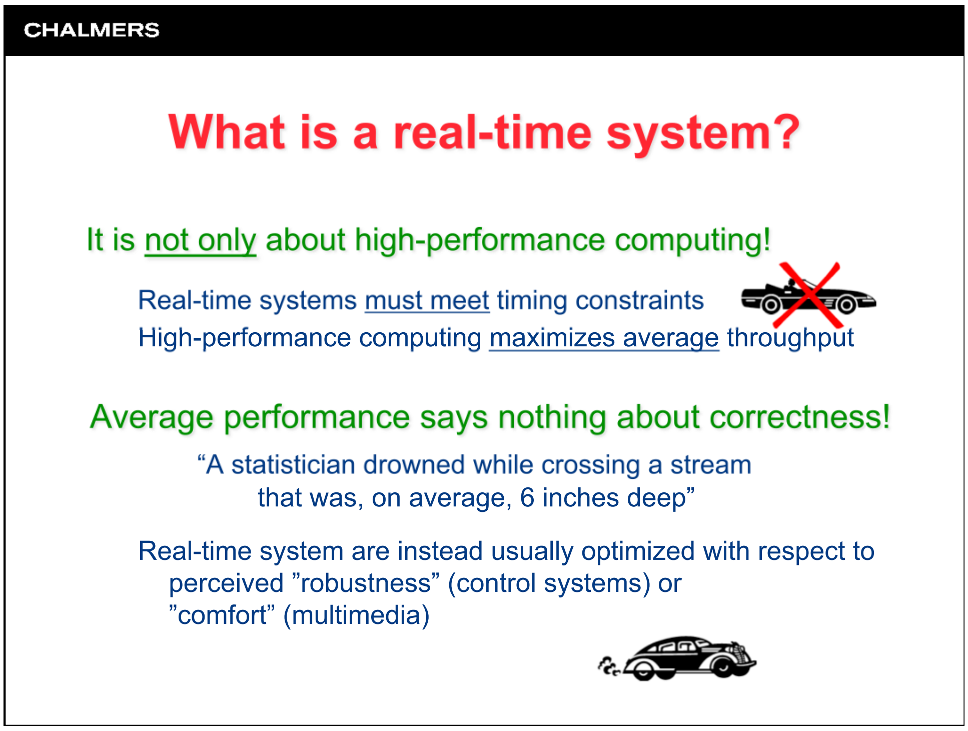


CHALMERS

What is a real-time system?

"A real-time system is one in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are generated"

J. Stankovic, "Misconceptions of Real-Time Computing", 1988



CHALMERS

What is a real-time system?

It is not only about high-performance computing!

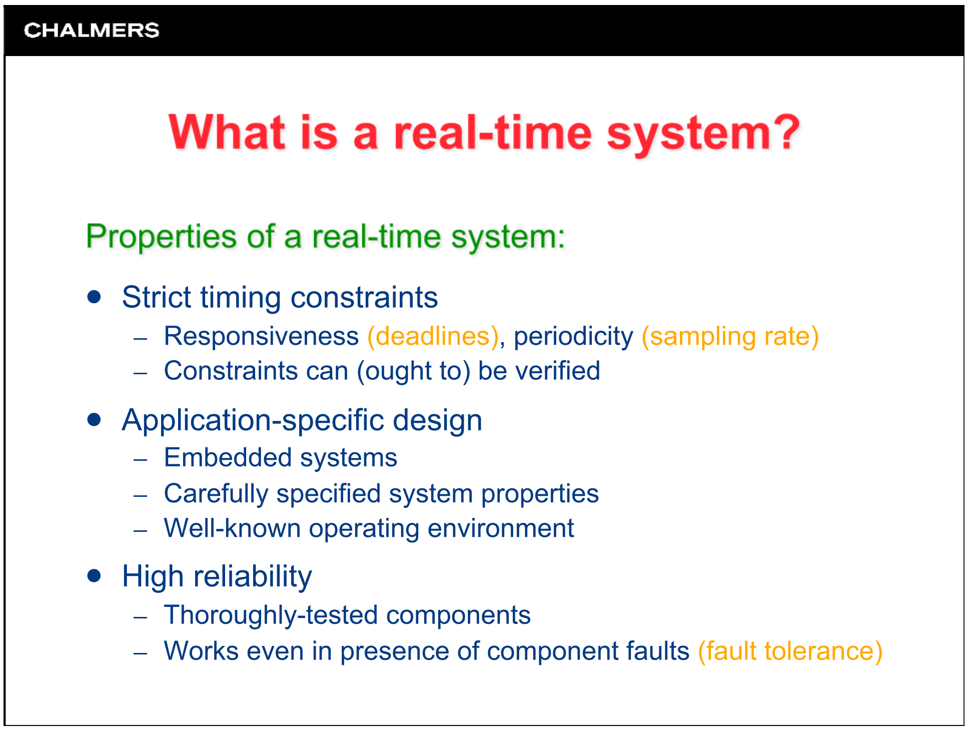
Real-time systems must meet timing constraints

High-performance computing maximizes average throughput

Average performance says nothing about correctness!

"A statistician drowned while crossing a stream that was, on average, 6 inches deep"

Real-time system are instead usually optimized with respect to perceived "robustness" (control systems) or "comfort" (multimedia)

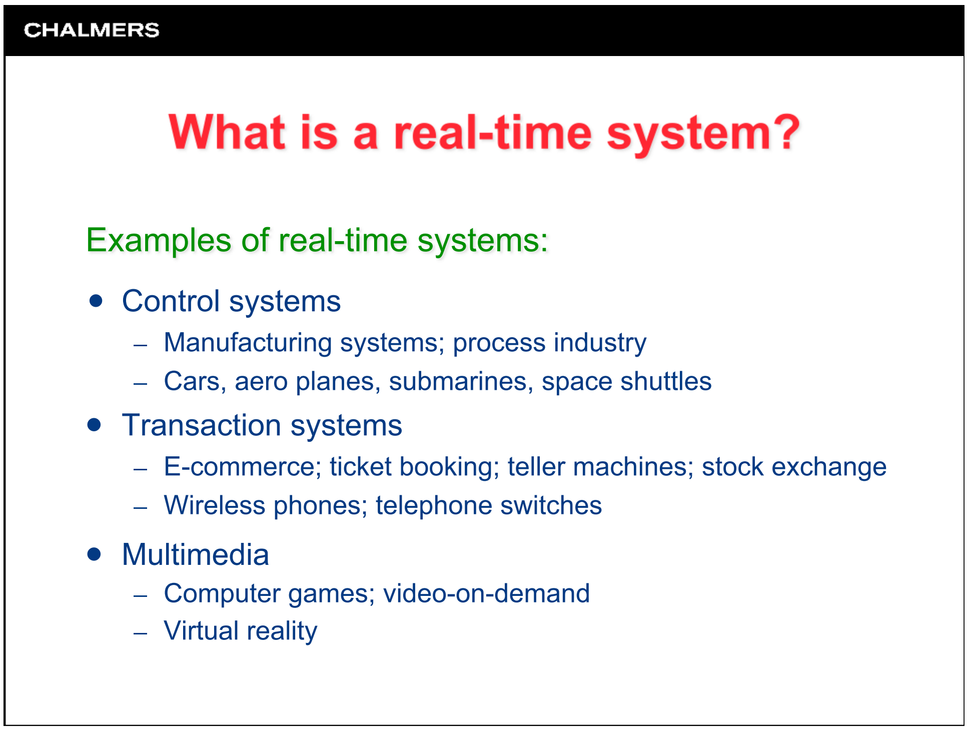


CHALMERS

What is a real-time system?

Properties of a real-time system:

- Strict timing constraints
  - Responsiveness (deadlines), periodicity (sampling rate)
  - Constraints can (ought to) be verified
- Application-specific design
  - Embedded systems
  - Carefully specified system properties
  - Well-known operating environment
- High reliability
  - Thoroughly-tested components
  - Works even in presence of component faults (fault tolerance)

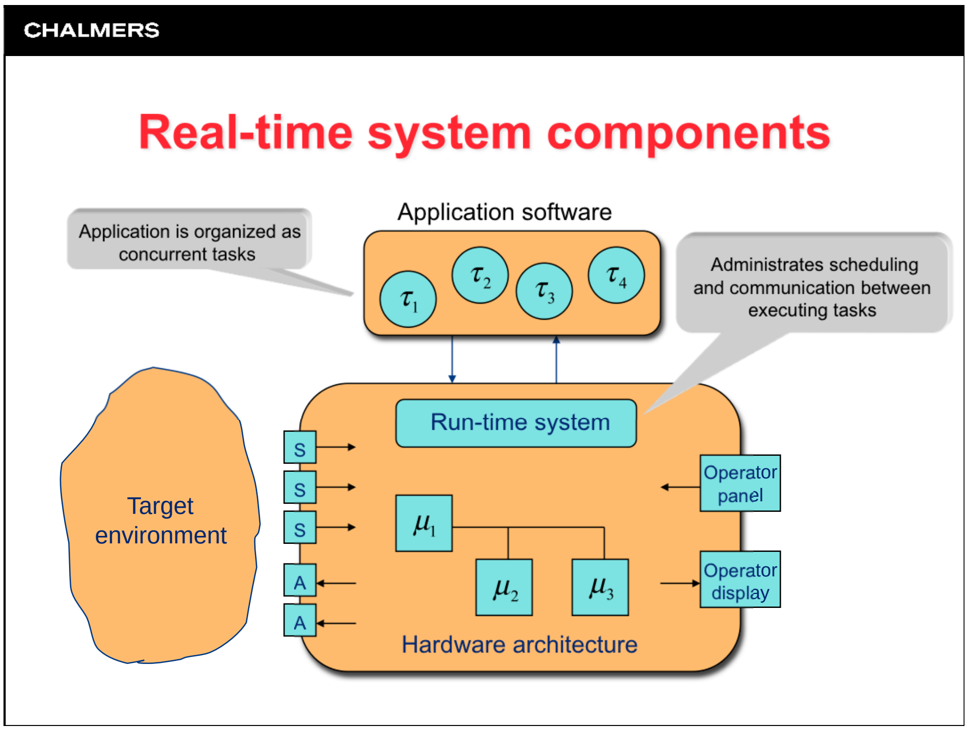


CHALMERS

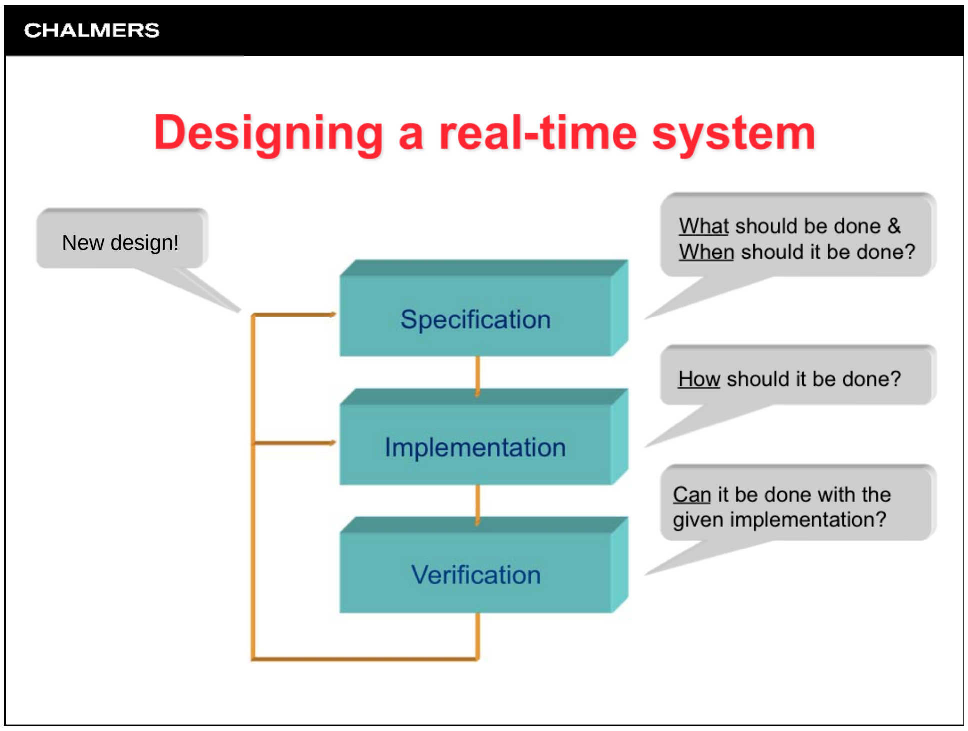
What is a real-time system?

Examples of real-time systems:

- Control systems
  - Manufacturing systems; process industry
  - Cars, aero planes, submarines, space shuttles
- Transaction systems
  - E-commerce; ticket booking; teller machines; stock exchange
  - Wireless phones; telephone switches
- Multimedia
  - Computer games; video-on-demand
  - Virtual reality

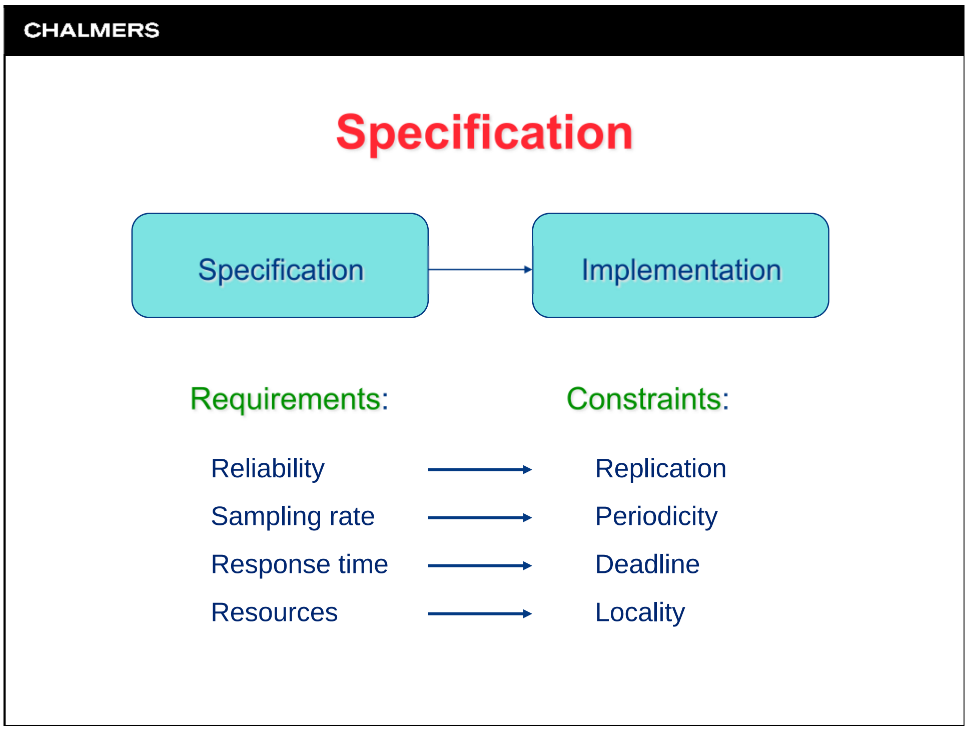

CHALMERS

Real-time system components


CHALMERS

Designing a real-time system

New design!
Specification
Implementation
Verification
What should be done & When should it be done?
How should it be done?
Can it be done with the given implementation?

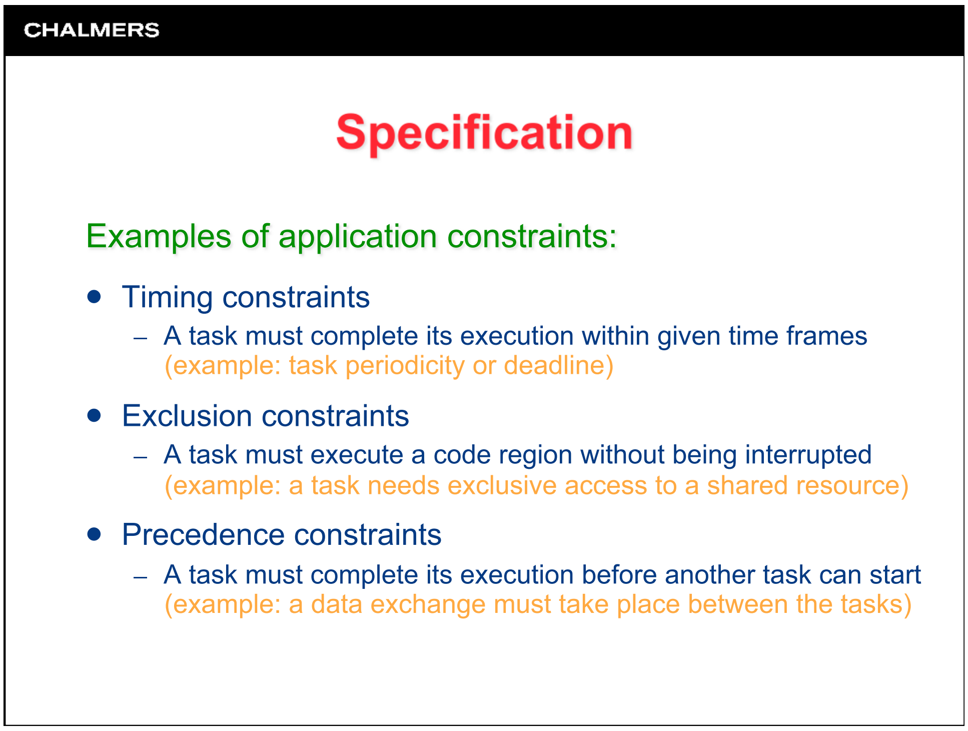


CHALMERS

Specification

$$\begin{tikzcd}\boxed{\text{Specification}} \arrow[r] & \boxed{\text{Implementation}}\end{tikzcd}$$

Requirements:		Constraints:
Reliability	→	Replication
Sampling rate	→	Periodicity
Response time	→	Deadline
Resources	→	Locality

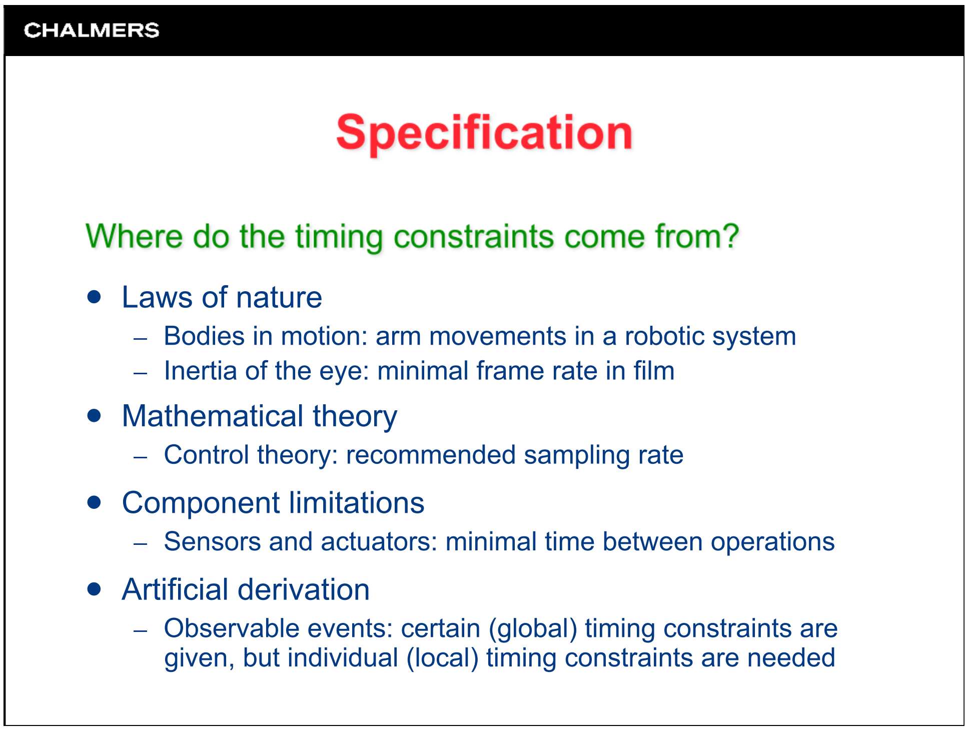


CHALMERS

Specification

Examples of application constraints:

- Timing constraints
  - A task must complete its execution within given time frames (example: task periodicity or deadline)
- Exclusion constraints
  - A task must execute a code region without being interrupted (example: a task needs exclusive access to a shared resource)
- Precedence constraints
  - A task must complete its execution before another task can start (example: a data exchange must take place between the tasks)

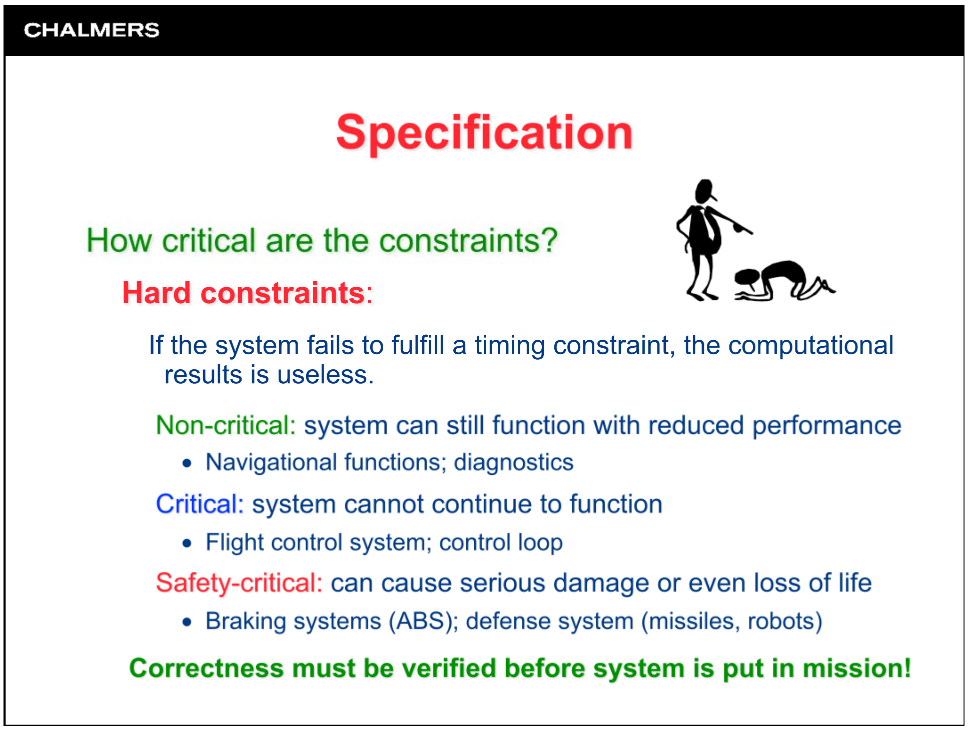


CHALMERS

Specification

Where do the timing constraints come from?

- Laws of nature
  - Bodies in motion: arm movements in a robotic system
  - Inertia of the eye: minimal frame rate in film
- Mathematical theory
  - Control theory: recommended sampling rate
- Component limitations
  - Sensors and actuators: minimal time between operations
- Artificial derivation
  - Observable events: certain (global) timing constraints are given, but individual (local) timing constraints are needed



CHALMERS

Specification

How critical are the constraints?

Hard constraints:

If the system fails to fulfill a timing constraint, the computational results is useless.

Non-critical: system can still function with reduced performance

- Navigational functions; diagnostics

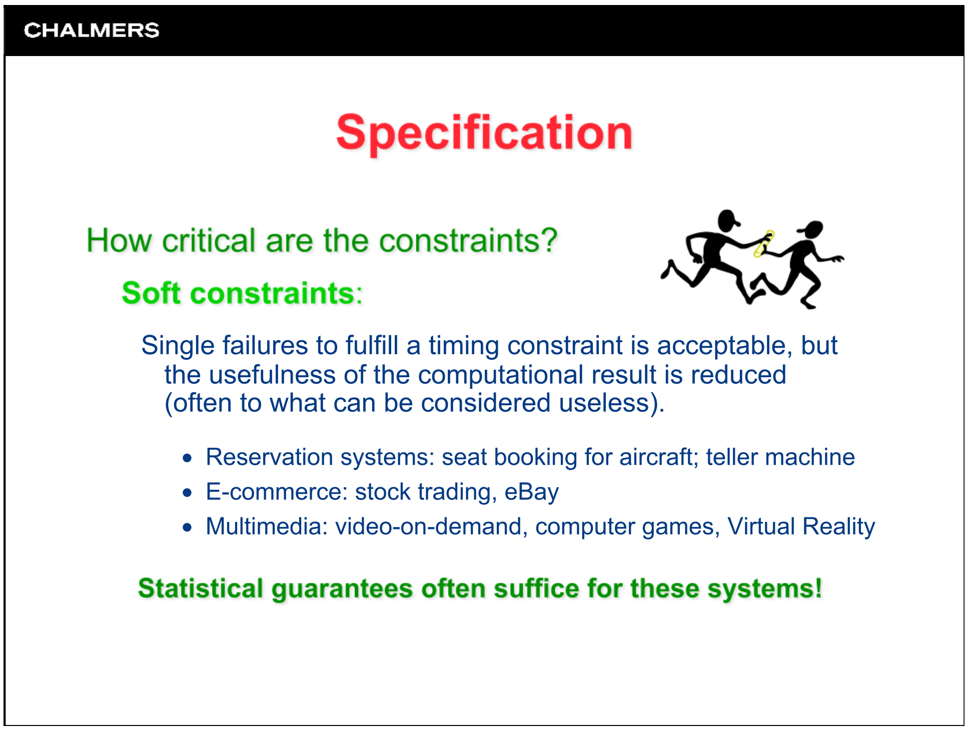
Critical: system cannot continue to function

- Flight control system; control loop

Safety-critical: can cause serious damage or even loss of life

- Braking systems (ABS); defense system (missiles, robots)

Correctness must be verified before system is put in mission!



CHALMERS

Specification

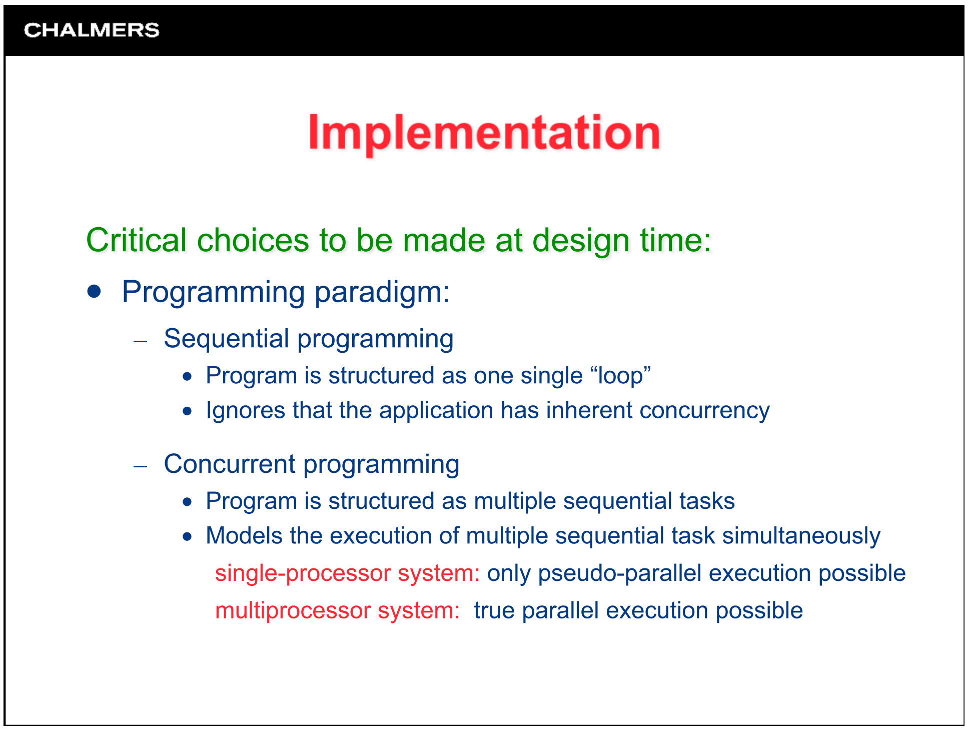
How critical are the constraints?

Soft constraints:

Single failures to fulfill a timing constraint is acceptable, but the usefulness of the computational result is reduced (often to what can be considered useless).

- Reservation systems: seat booking for aircraft; teller machine
- E-commerce: stock trading, eBay
- Multimedia: video-on-demand, computer games, Virtual Reality

Statistical guarantees often suffice for these systems!



CHALMERS

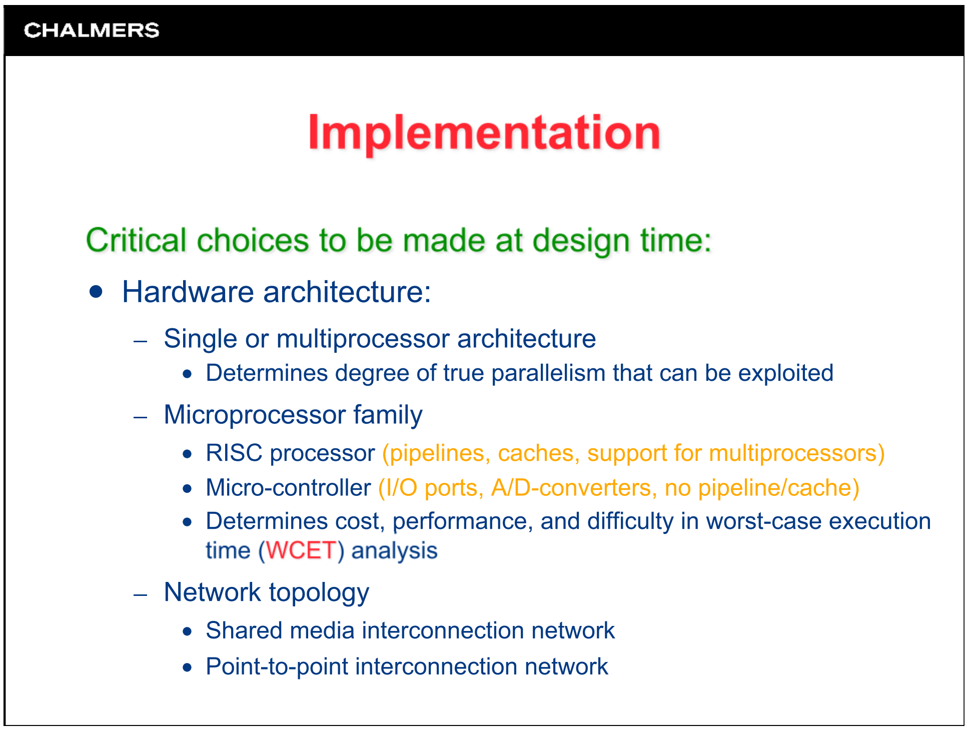
Implementation

Critical choices to be made at design time:

- Programming paradigm:
  - Sequential programming
    - Program is structured as one single “loop”
    - Ignores that the application has inherent concurrency
  - Concurrent programming
    - Program is structured as multiple sequential tasks
    - Models the execution of multiple sequential task simultaneously

      single-processor system: only pseudo-parallel execution possible

      multiprocessor system: true parallel execution possible

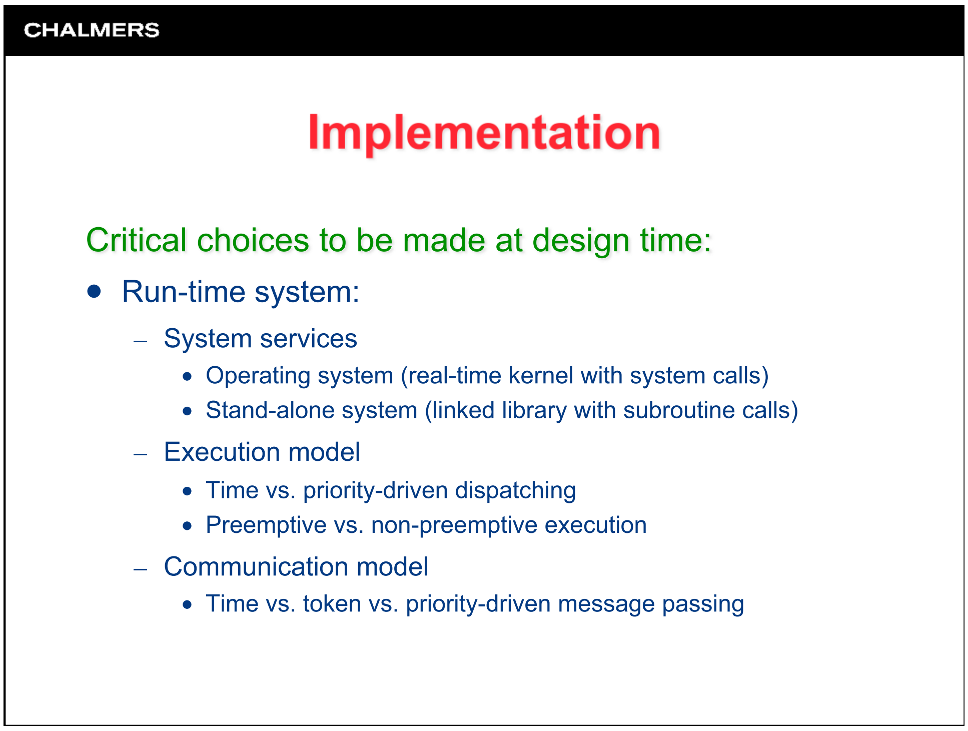


CHALMERS

Implementation

Critical choices to be made at design time:

- Hardware architecture:
  - Single or multiprocessor architecture
    - Determines degree of true parallelism that can be exploited
  - Microprocessor family
    - RISC processor (pipelines, caches, support for multiprocessors)
    - Micro-controller (I/O ports, A/D-converters, no pipeline/cache)
    - Determines cost, performance, and difficulty in worst-case execution time (WCET) analysis
  - Network topology
    - Shared media interconnection network
    - Point-to-point interconnection network

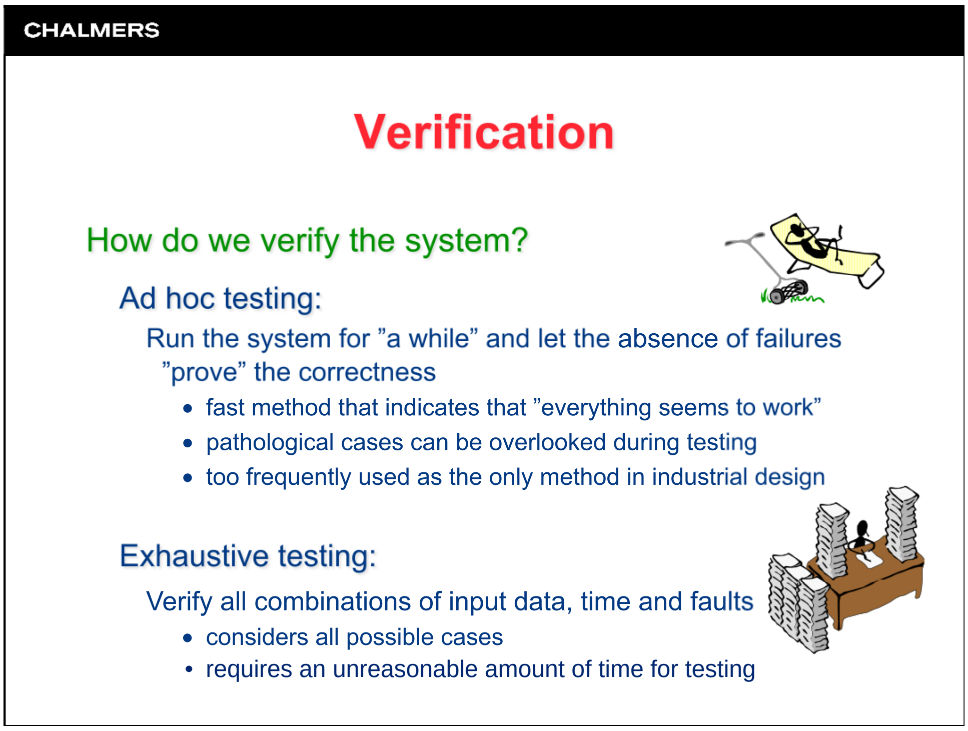


CHALMERS

Implementation

Critical choices to be made at design time:

- Run-time system:
  - System services
    - Operating system (real-time kernel with system calls)
    - Stand-alone system (linked library with subroutine calls)
  - Execution model
    - Time vs. priority-driven dispatching
    - Preemptive vs. non-preemptive execution
  - Communication model
    - Time vs. token vs. priority-driven message passing



CHALMERS

Verification

How do we verify the system?

Ad hoc testing:

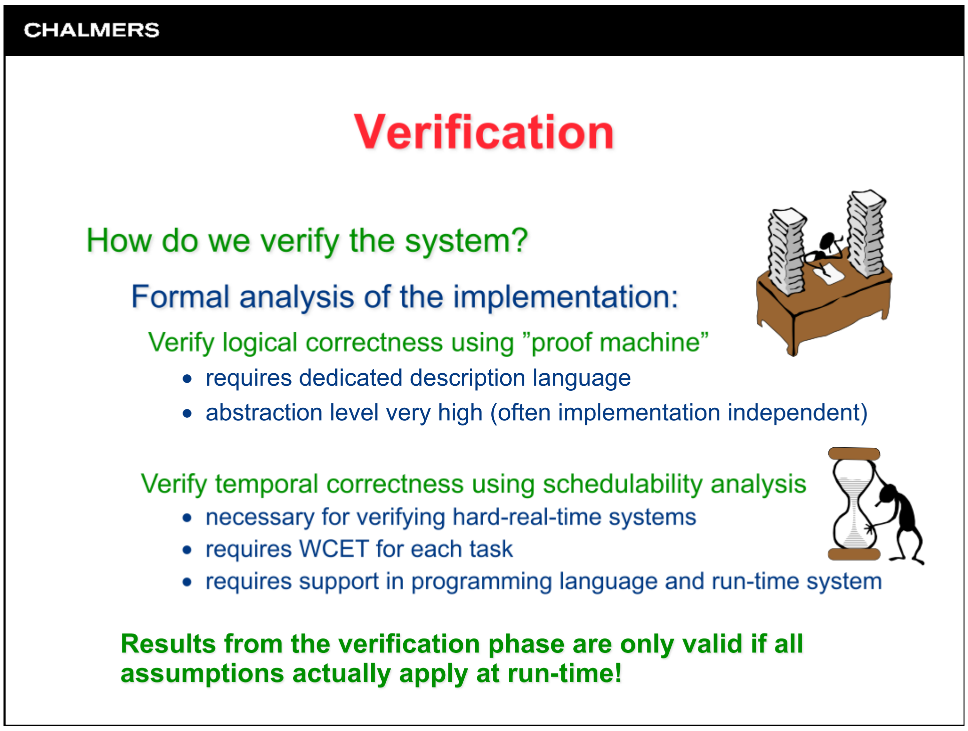
Run the system for ”a while” and let the absence of failures ”prove” the correctness

- fast method that indicates that ”everything seems to work”
- pathological cases can be overlooked during testing
- too frequently used as the only method in industrial design

Exhaustive testing:

Verify all combinations of input data, time and faults

- considers all possible cases
- requires an unreasonable amount of time for testing



CHALMERS

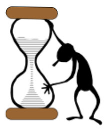
Verification

How do we verify the system?

Formal analysis of the implementation:

Verify logical correctness using ”proof machine”

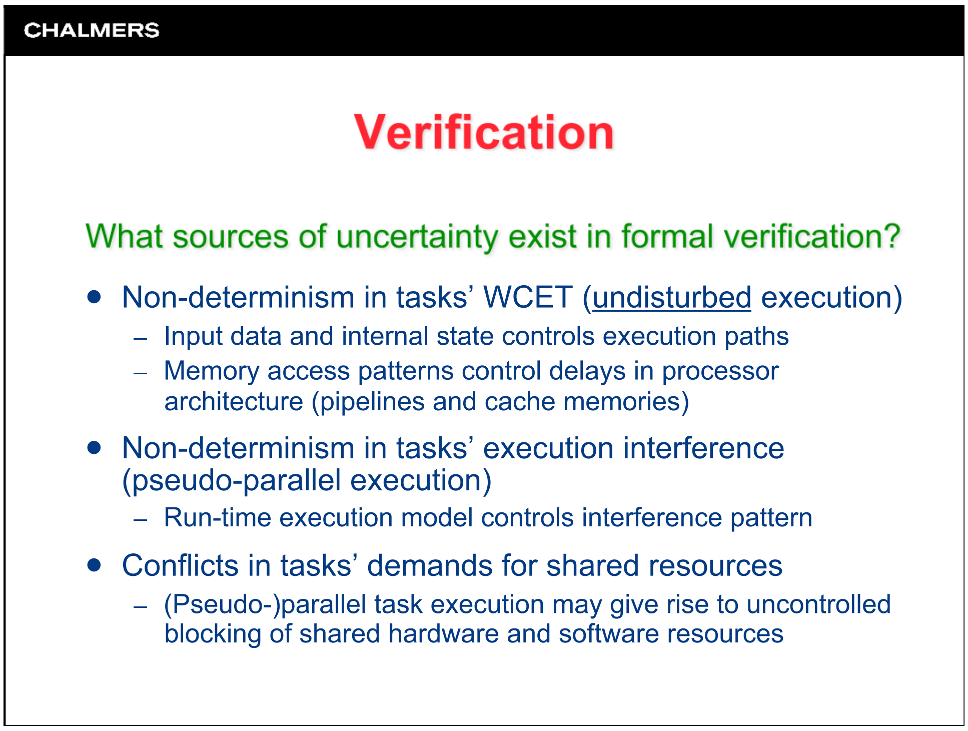
- requires dedicated description language
- abstraction level very high (often implementation independent)



Verify temporal correctness using schedulability analysis

- necessary for verifying hard-real-time systems
- requires WCET for each task
- requires support in programming language and run-time system

Results from the verification phase are only valid if all assumptions actually apply at run-time!

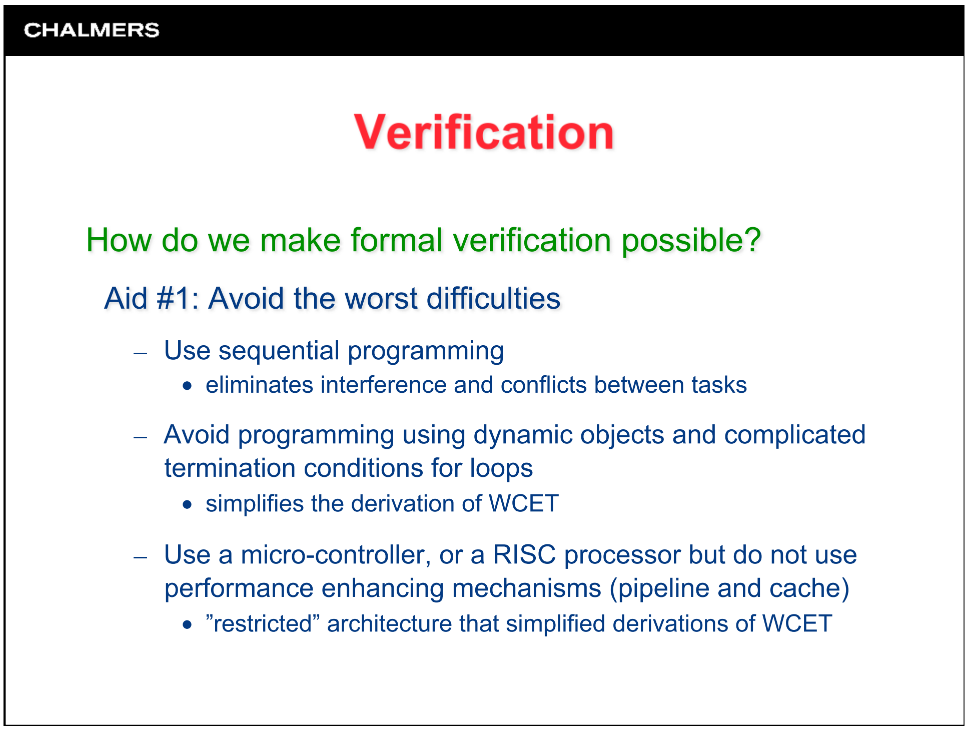


CHALMERS

Verification

What sources of uncertainty exist in formal verification?

- Non-determinism in tasks' WCET (undisturbed execution)
  - Input data and internal state controls execution paths
  - Memory access patterns control delays in processor architecture (pipelines and cache memories)
- Non-determinism in tasks' execution interference (pseudo-parallel execution)
  - Run-time execution model controls interference pattern
- Conflicts in tasks' demands for shared resources
  - (Pseudo-)parallel task execution may give rise to uncontrolled blocking of shared hardware and software resources



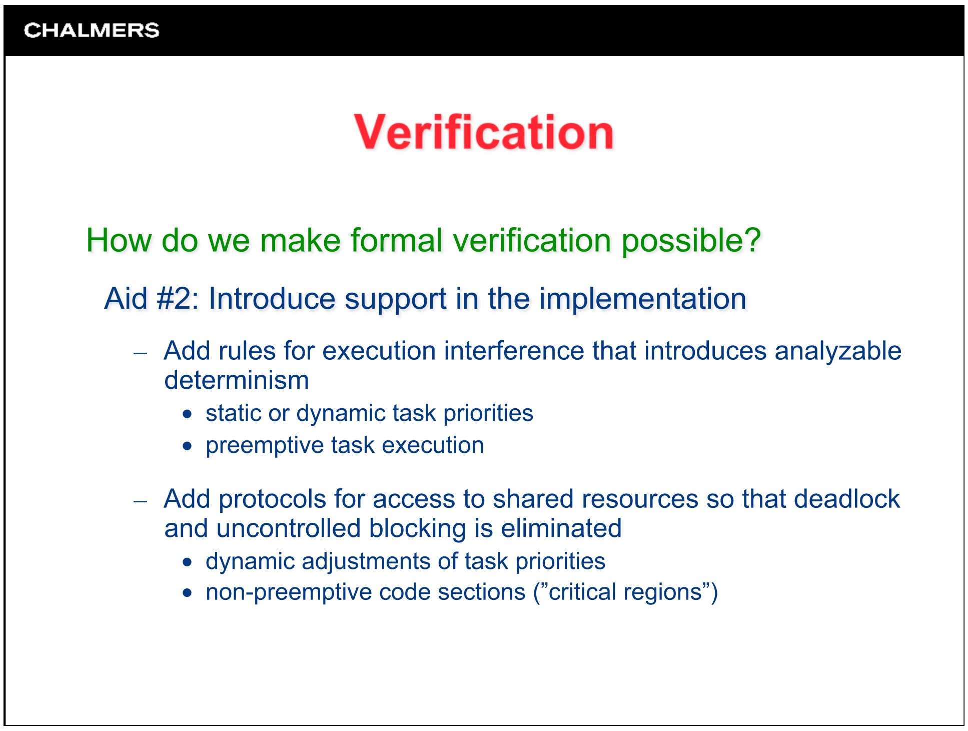
CHALMERS

Verification

How do we make formal verification possible?

Aid #1: Avoid the worst difficulties

- Use sequential programming
  - eliminates interference and conflicts between tasks
- Avoid programming using dynamic objects and complicated termination conditions for loops
  - simplifies the derivation of WCET
- Use a micro-controller, or a RISC processor but do not use performance enhancing mechanisms (pipeline and cache)
  - ”restricted” architecture that simplified derivations of WCET



CHALMERS

Verification

How do we make formal verification possible?

Aid #2: Introduce support in the implementation

- Add rules for execution interference that introduces analyzable determinism
  - static or dynamic task priorities
  - preemptive task execution
- Add protocols for access to shared resources so that deadlock and uncontrolled blocking is eliminated
  - dynamic adjustments of task priorities
  - non-preemptive code sections (”critical regions”)