# Synchronous Languages—Lecture 01

Prof. Dr. Reinhard von Hanxleden

Christian-Albrechts Universität Kiel
Department of Computer Science
Real-Time Systems and Embedded Systems Group

24 Oct 2016

*Last compiled: October 24, 2016, 9:29 hrs*

## *Introduction*

# Overview

# Aim of this Lecture

After this lecture, you should have an idea on . . .

- . . . what this class and related classes are about
- . . . whether this class is for you
- . . . what is expected of you should you decide to participate

# What this class will be about

- Synchronous Languages and the Synchrony Hypothesis: Separate design of control from timing constraints
- Esterel: a textual, synchronous language
  - Formal semantics
  - Code and hardware synthesis for Esterel programs
  - Analysis, constructiveness
  - Reactive processing (Kiel Esterel Processor)
- Other synchronous languages:
  - Lustre
  - Scade
  - SC: SyncCharts in C
  - SCL: Sequentially Constructive Language
  - Statecharts, expecially SyncCharts (the graphical counterpart to Esterel) and SCCharts (Sequentially Constructive Charts)
- Optionally: further concurrent models of computation

# Related classes

- Embedded RT Systems (WS 15/16, WS 17/18)
  - Modeling dynamic behaviors
  - Design of Embedded Systems
  - Analysis and verification
  - Lego Mindstorms
- Graph Drawing (SS 16, SS 18)
  - Explains algorithms behind, *e. g.*, SCCharts browser
  - Force-directed approaches
  - Layer-based / Sugiyama
  - Tree drawing

# What you should learn in this course

1. You should know what synchronous languages are
2. You should know about their theoretical foundation of synchronous languages
3. You should have a detailed knowledge about Esterel and SCCharts, including their semantics
4. You should be aware of possibilities and problems in code/hardware generation from synchronous languages

# People

Lectures:
Reinhard von Hanxleden
`rvh@...` Tel.: 880-7281

Recitations (*Übungen*):
Alexander Schulz-Rosengarten
`als@...` Tel.: 880-7526

Corrections:
Lars Peiler
`lpe@...`

- Office hours: by appointment—or just contact us after class

# The Class Homepage

- `https://ilearn.ps.informatik.uni-kiel.de/public/courses/112`
- Contents:
  - Link to register for this class
  - Link to register for the mailing list
  - Lecture slides (with/without notes, with/without animation)
  - Homework assignments
  - Current information
  - Further links

# Notation

The markups (the "secondary notation") used on these slides follows (mostly) the following scheme:

- Definitions, first use of a term
- Text structuring
- Examples
- Normal text
- `Code, keywords, identifiers`
- `paths, executables`
- `URLs`
- *MathematicalSymbols*
- *General emphasis*
- **Reeeally important stuff**

# Homeworks

- Homeworks
  - generally given at Tuesday,
  - due by following Tuesday (23:59 hrs),
  - should be submitted via iLearn (see class homepage)
  - discussed following Friday recitation
  - First recitation: Nov. 4.
- Homeworks shall be submitted by groups
  - Ideal group size: 2 students
  - **Each group member should be able to present submissions**
- Questions
  - may be asked at any time, on anything . . .
  - . . . however, questions on the homework are better asked **before** the deadline and before submitting the homework!

# Grading (*Scheinkriterien*)

- Can get bonus points for outstanding solutions
- Can also get point deductions for late submissions, multiple submissions, etc.
- Will receive regular feedback on accumulated score
- For all participants, there will be one final exam

# Final Exam

- Tentative date: **Thu, Feb. 9** (Must be within Feb. 7 – 20)
- Need at least 50% to pass
- In borderline cases, also consider participation in class
- Results in exercises can improve grade, if 85% exam + 15% exercises are better than exam score

Admitted to final exam if:

- Received at least 50% of homework assignment points
- Missed at most two recitation classes

# Priorities

Your grade depends on

- Final exam
- Homework submissions
- Participation in class (in borderline cases)

Advice: make up your mind on whether you want to participate in this class or not rather soon (within the next two weeks)

- Should participate 0% or 100% :-)

# Literature: Synchronous Languages


- [Halbwachs 1998]
  Nicolas Halbwachs, Synchronous programming of reactive systems, a tutorial and commented bibliography,
  *Tenth International Conference on Computer-Aided Verification*, CAV'98 Vancouver (B.C.),
  LNCS 1427, Springer Verlag, June 1998,
  `http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.8306`
- [Benveniste+ 2003]
  Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert de Simone.
  The Synchronous Languages Twelve Years Later
  IEEE, Special Issue on Embedded Systems, 2003
  `http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.1117`

# Literature: Esterel

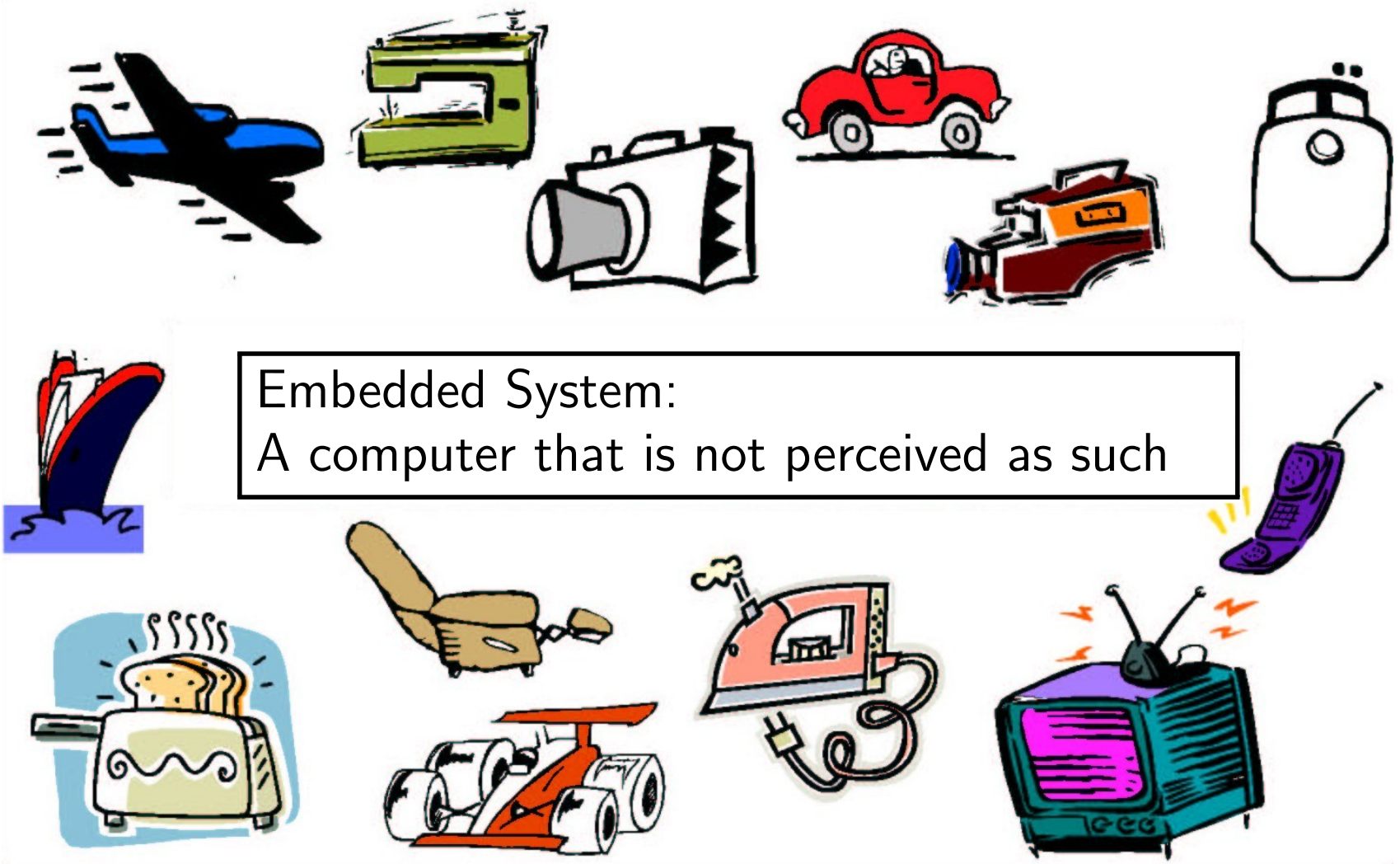

- [Berry 2000]
  Gérard Berry, The Foundations of Esterel,
  Proof, Language and Interaction: Essays in Honour of Robin Milner,
  G. Plotkin, C. Stirling and M. Tofte, editors,
  MIT Press, *Foundations of Computing Series*, 2000,
  `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.6221`
- [Berry 1999]
  Gérard Berry, The Constructive Semantics of Esterel,
  Draft book, current version 3.0, Dec. 2002
  `http://www-sop.inria.fr/members/Gerard.Berry/Papers/EsterelConstructiveBook.zip`
- [Esterel Primer]
  Gérard Berry, The Esterel v5 Language Primer, Version v5_91, 2000
  `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.8212`

# Overview

# Definition of Embedded Systems

- Embedded systems were designed for dedicated applications inside a surrounding system
- Embedded systems normally consist of hard- *and* software
- In addition to standard microprocessors, sometimes special hardware is used *e. g.* for MPEG-decoding
- Often many embedded systems form a *distributed system*
- Often many processes run in parallel on one microprocessor
- Do we need an operating system for process management?

# A Definition:



# Arguments for Embedded Systems

Increase of comfort: simplifies usage
Decrease of physical size: important for mobile devices
Increase of functionality: allows decentralized computations
Increase of safety: autopilot in aircrafts, brake-by-wire in cars
Decrease of production costs: electronic systems often cheaper
Increase of maintainability: by diagnosis devices
Optimization of control: *e. g.* dynamic control of fuel injection
Personalization: systems can be adapted for different users
Decrease of power consumption: important for mobile devices
Protection of intellectual property: difficult to copy by competitors

*Thanks to Klaus Schneider (Kaiserslautern) for providing part of this material*

# Design of Embedded Systems

Embedded systems (ES) are built for years.
What are the new challenges in their design?

- *More ESs* are included in one system
- ESs are more and more *responsible for economic success*
- ESs are more and more *responsible for design costs*
- *Product differentiation* more and more by embedded systems
- *Supervision of safety-critical systems*

Example application: cars

- Supervise and correct driving actions of driver
- Detect other cars and object in the environment
- Predict unavoidable collisions, and initiate driving actions to decrease damage
- Post-crash behavior: notify hospital and send GPS coordinates

# Problems with Embedded Systems

Are there any disadvantages? Of course:

**Many systems like cars are used for 20 years, while computer systems have much shorter lifetimes**

- Problem: supply with parts for many years
- Problem: lifetime of ESs must be long enough

**Safety-critical applications are controlled by ESs**

- Problem: computer systems do also have errors
- Problem: complex systems have many errors
- Problem: unfriendly environment (*e. g.* high/low temperature)
- Is there really a gain in safety?

# Design Problems: Design Exploration

- Due to manual design, there is no time to evaluate different design variants
- In particular, the HW/SW partitioning phase cannot be repeated

$\rightsquigarrow$ Trend towards 'overdesign', *i. e.*, the systems are more expensive and more powerful than necessary

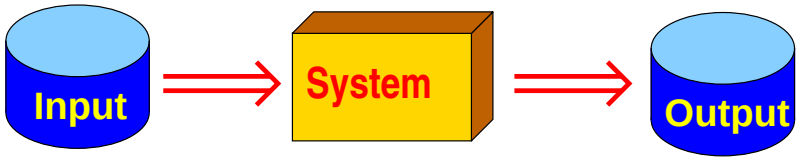
$\rightsquigarrow$ Realization independent design necessary, *i. e.*, early design phases should not fix on HW or SW solutions

- Problem: which languages to use for these descriptions?

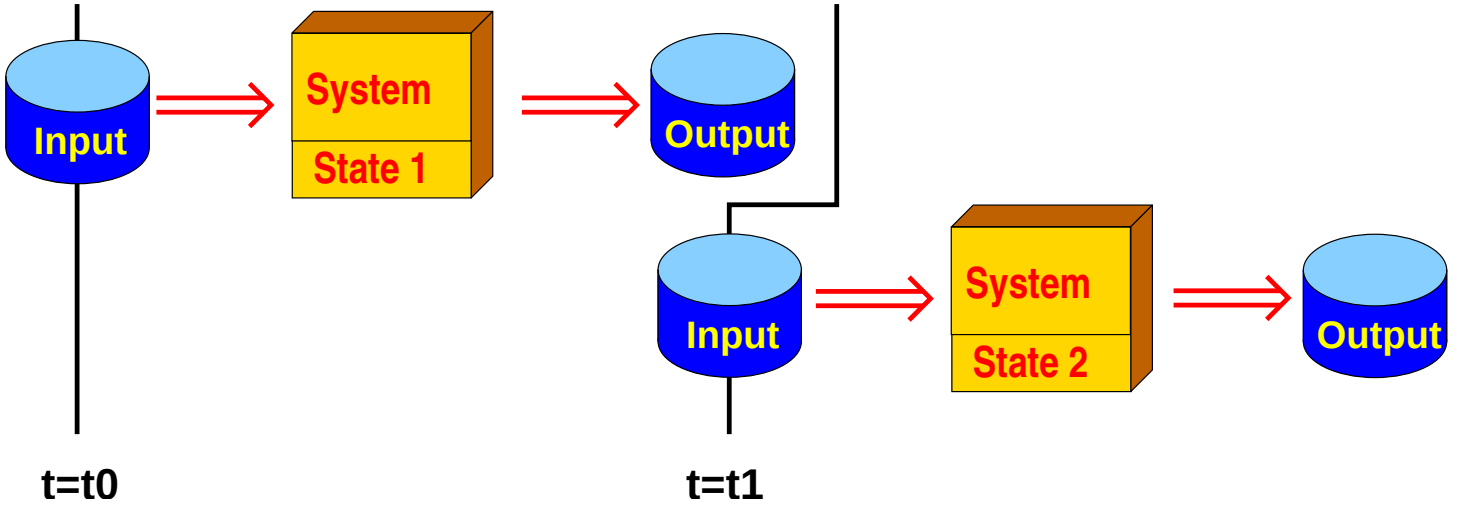
$\rightsquigarrow$ One of the motivations for synchronous languages

# Different Kinds of Systems

## Transformational Systems




## Interactive/Reactive Systems



# Interactive vs. Reactive Systems

Transformational systems:

- read inputs, compute outputs and terminate
- Example: compiler

Interactive systems:

- nonterminating
- continuous interaction
- pace is controlled by system
- Example: on-line reservation system

Reactive systems:

- nonterminating
- continuous interaction
- pace is controlled by environment
- Example: engine controller

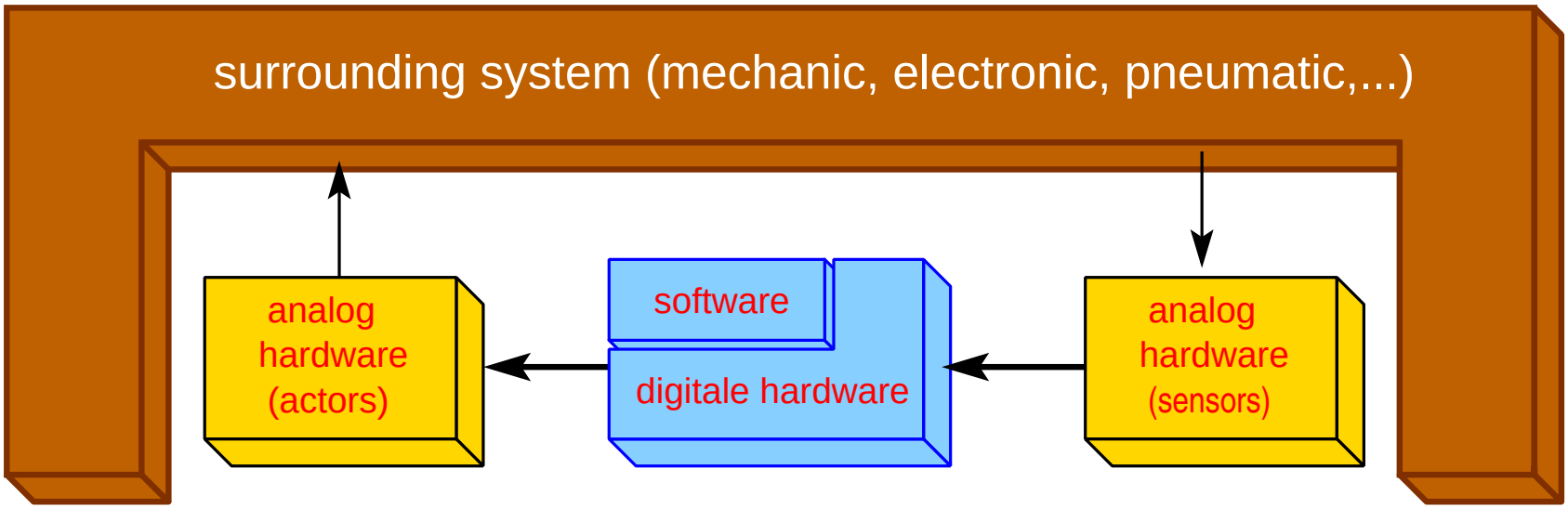
$\Rightarrow$ **Reactive systems are real-time systems!**

# Interactive vs. Reactive Systems

- Interactions with user/environment are basic computation steps of reactive systems
- Logical time: counts only number of interactions
- Interactions consist of micro steps (smaller computations)
- Interactions are often called macro steps
- Remark: inputs are read only once per macro step, hence, they are assumed to be constant for a macro step
- Question: when are outputs produced?
- Answer: perfect synchrony has the view that outputs are generated in zero time for a macro step

# Embedded Systems as Reactive Systems

General Schema:




Embedded systems interact directly with surrounding system and are thus often *reactive systems*

# Reactive Control Flow

Control flow on traditional (non-embedded) computing systems:

- Jumps, conditional branches, loops
- Procedure/method calls

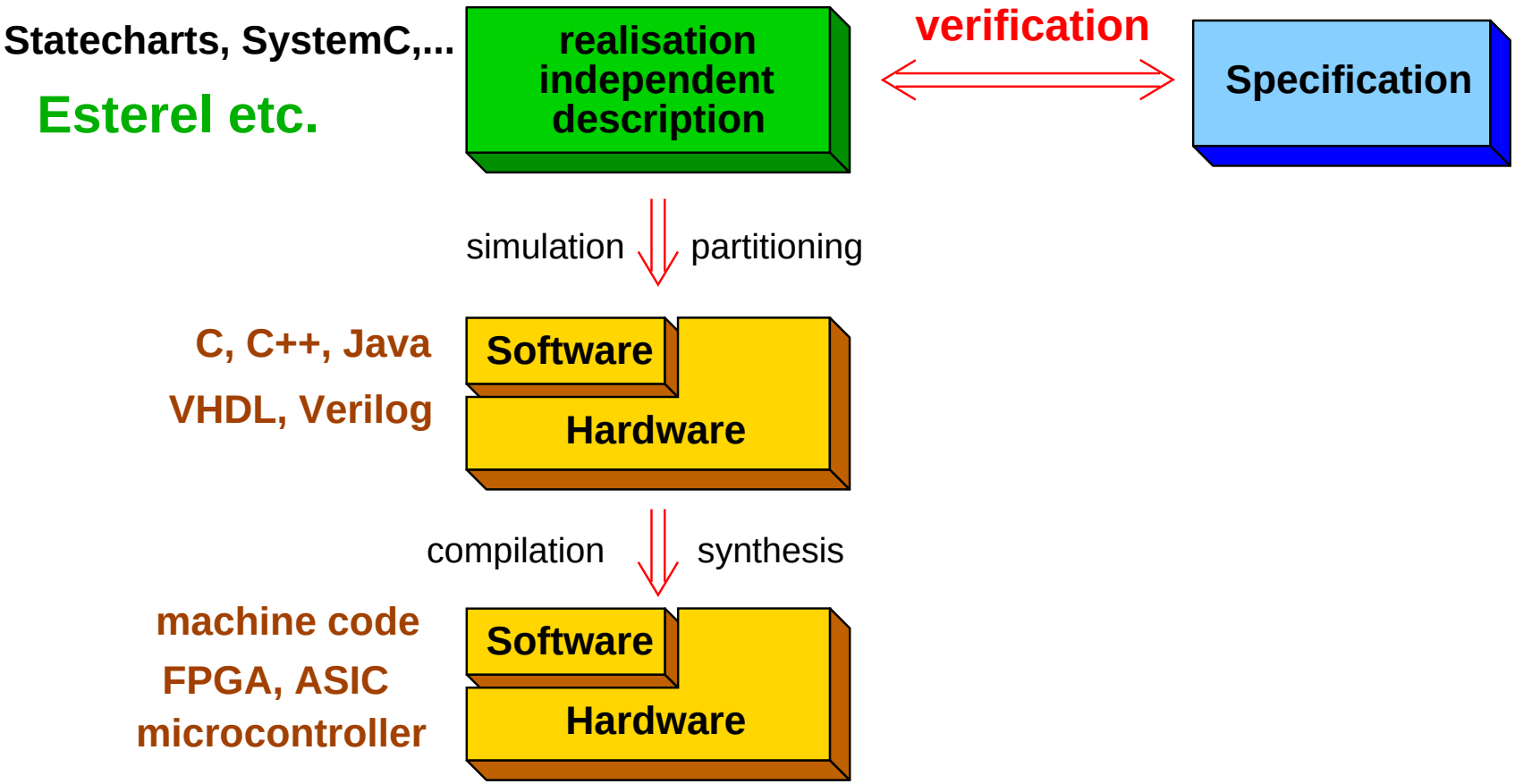
Control flow on embedded, reactive systems: all of the above, plus

- Concurrency
- Preemption

The problem: mismatch between traditional languages and reactive control flow patterns

- Non-determinism, *e. g.* due to scheduler and interrupt handler
- Processing overhead, *e. g.* due to OS involvement or need to save thread states at application level
- Timing unpredictability

# Advanced Design Flows



# Advanced Design Flows

- Early cost estimation
- Simulation of design variants
- Formal verification in early design phases
- Guarantee of real-time constraints
- Support for distributed systems (also multi-processor systems)
- Modeling of the environment, also of analog and mechanical parts

# Summary

- Embedded systems are ubiquitous today
- Distinguish transformational, interactive, reactive systems
- Synchronous languages
  - are domain independent (can describe HW and SW) and allow to work at high abstraction level
  - support reactive control flow (including concurrency and preemption)
  - have deterministic, formally founded semantics
  - support modular design due to perfect synchrony
- This class will explore the family of synchronous languages in depth