Issues in realizing an end-to-end embedded system tool-chain:

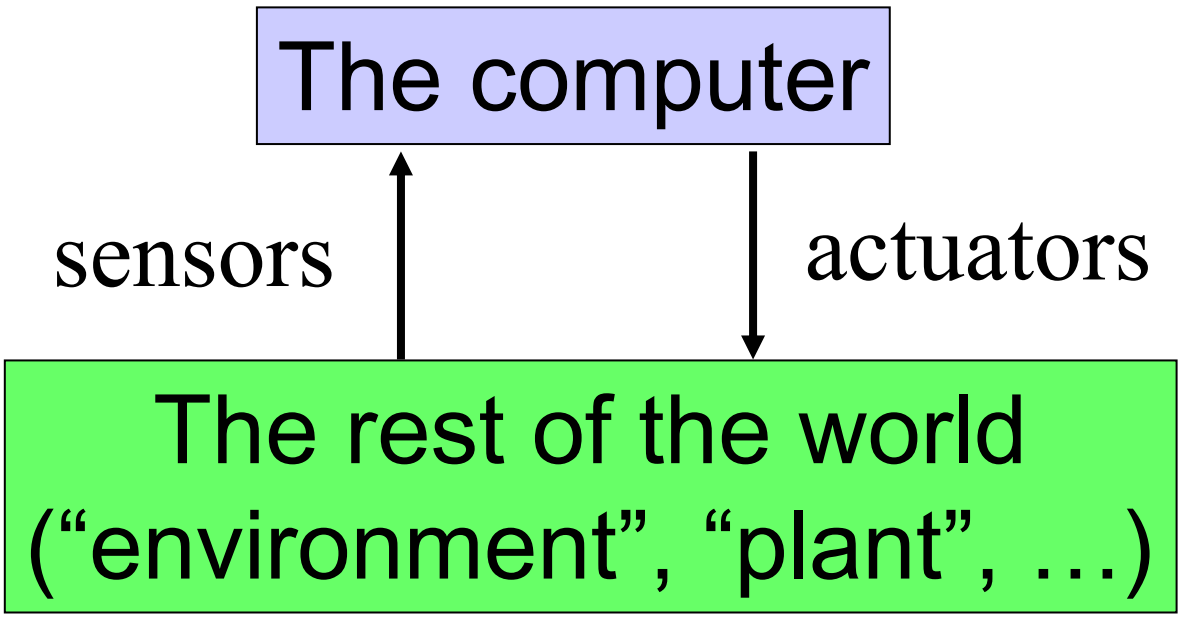
Experiences from the European IST projects "NEXT TTA" and "RISE"

Stavros Tripakis

VERIMAG

Joint work with: Paul Caspi, Adrian Curic, Aude Maignan, Christos Sofronis

Embedded systems



Embedded systems: what's new about them ?

- Is emacs an embedded system ?

- In a sense yes, but often there's more:
  - Cheap computers (low-speed, low-memory, low-battery, …)
  - Real-time environments (can't wait)
  - Safety critical, control applications
  - Distributed, multiprocessor, networked, …

A point of view

- Embedded systems is an application domain.
- Requires harmonious contribution of many scientific domains:
  - control
  - verification
  - scheduling
  - programming/compilation
  - OS and middleware
  - architectures and hardware

A point of view

- Pushes these domains to reconsider their assumptions:
  - control:
    - controllers under architecture/implementation constraints (communication delays, sampling, …)
  - verification:
    - heterogeneous semantics
  - scheduling:
    - variable CPU rate (for power savings), more complex environment models, …

A point of view

- Pushes these domains to reconsider their assumptions:
  - programming/compilation
    - interfaces, RT extensions, “safety” features (e.g., exception handling), ...
  - OS and middleware:
    - micro-kernels, real-time guarantees, resource monitoring and management, ...
  - architectures and hardware:
    - power, fault-tolerance, ...

A point of view

- Introduces interdisciplinary problems:
  - implementation of controllers (control-theory and computer science)
  - WCET (verification and hardware architectures)

A point of view

- Involves lots of politics:
  - fights for standards, ...

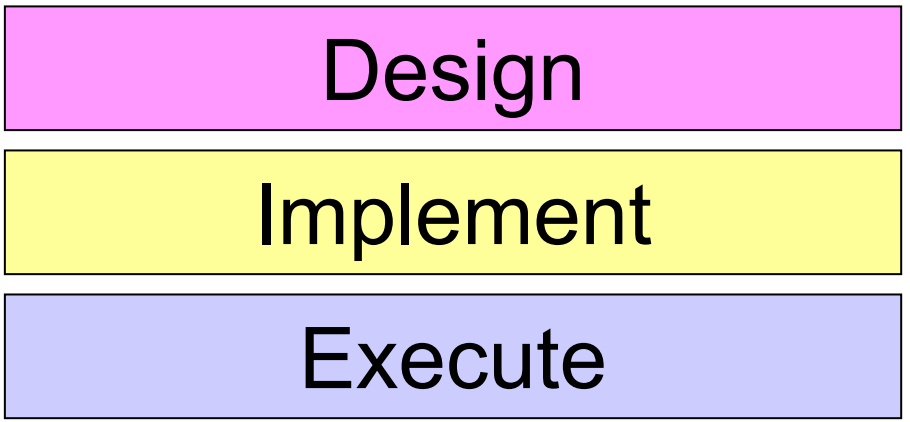
Our work

- In the context of European IST projects:
- “NEXT TTA” (2002-2003) and
- “RISE” (2003-2004).

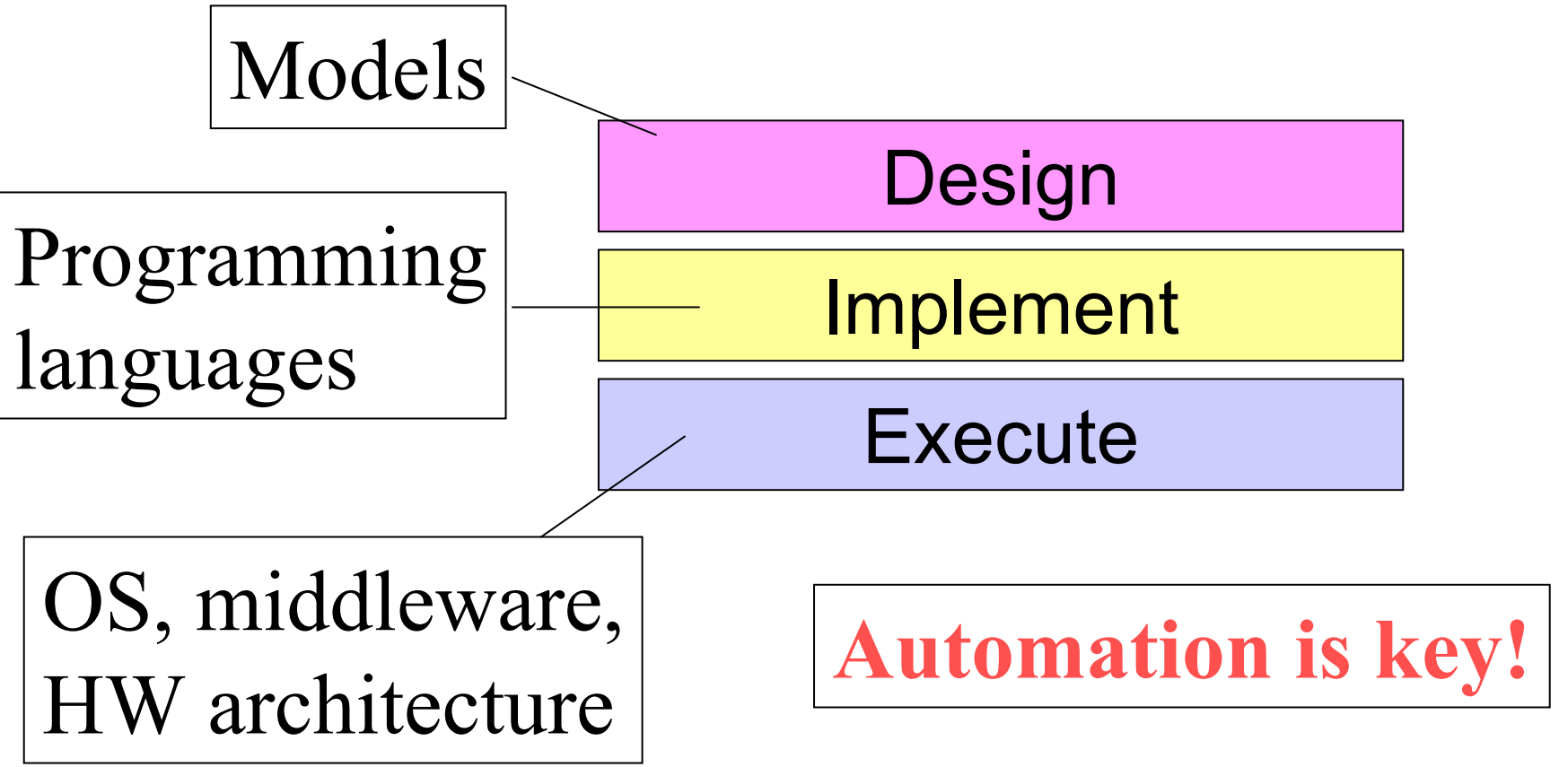
- Automotive applications (partner: Audi).

- Other industrial partners:
  - TTTech, Esterel Technologies, Austria Microsystems, ...

Our view: a development process in three layers

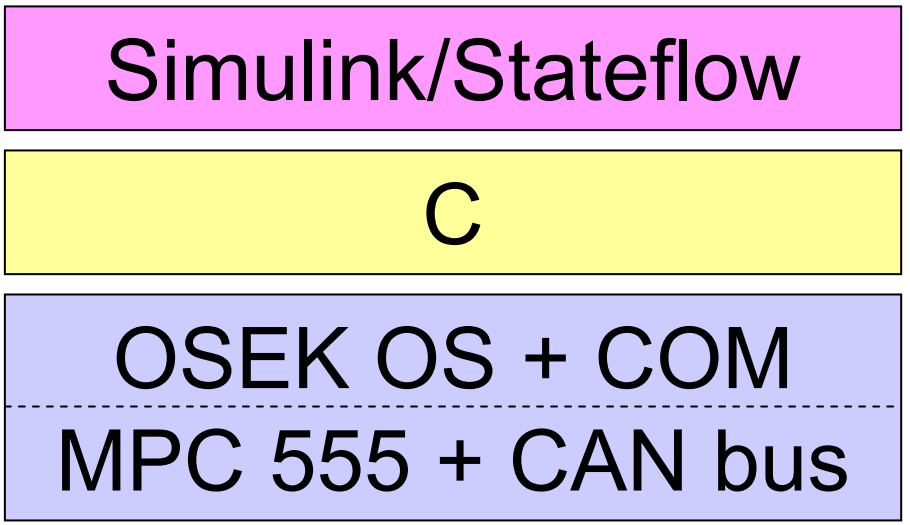


Our view: a development process in three layers, supported by:



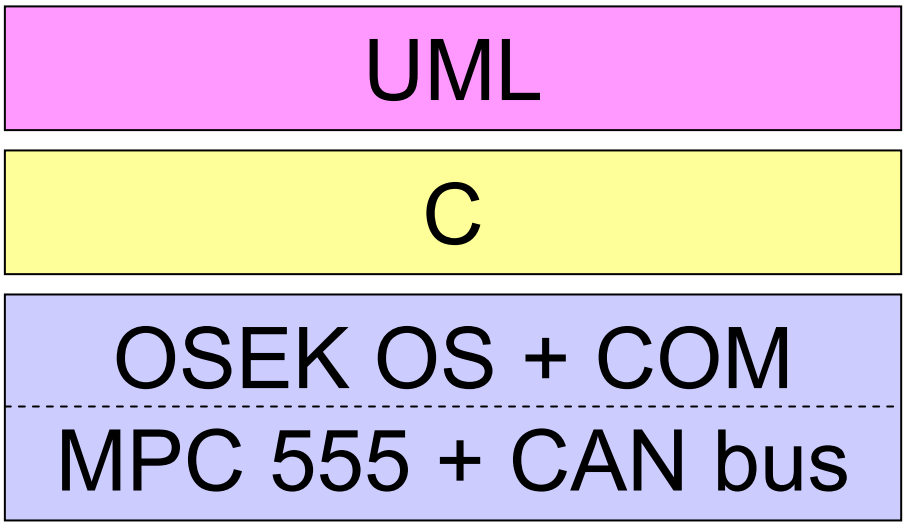
A possible stack

- “control” stack:



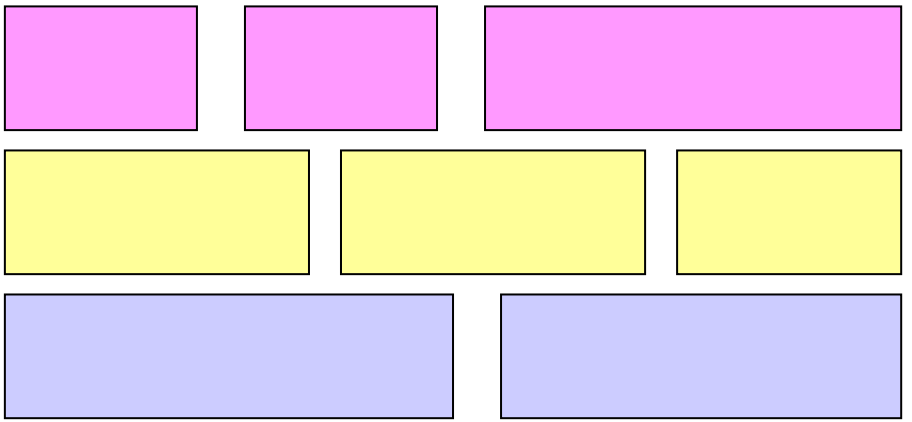
Multiple stacks may be needed

- "software architecture" stack:

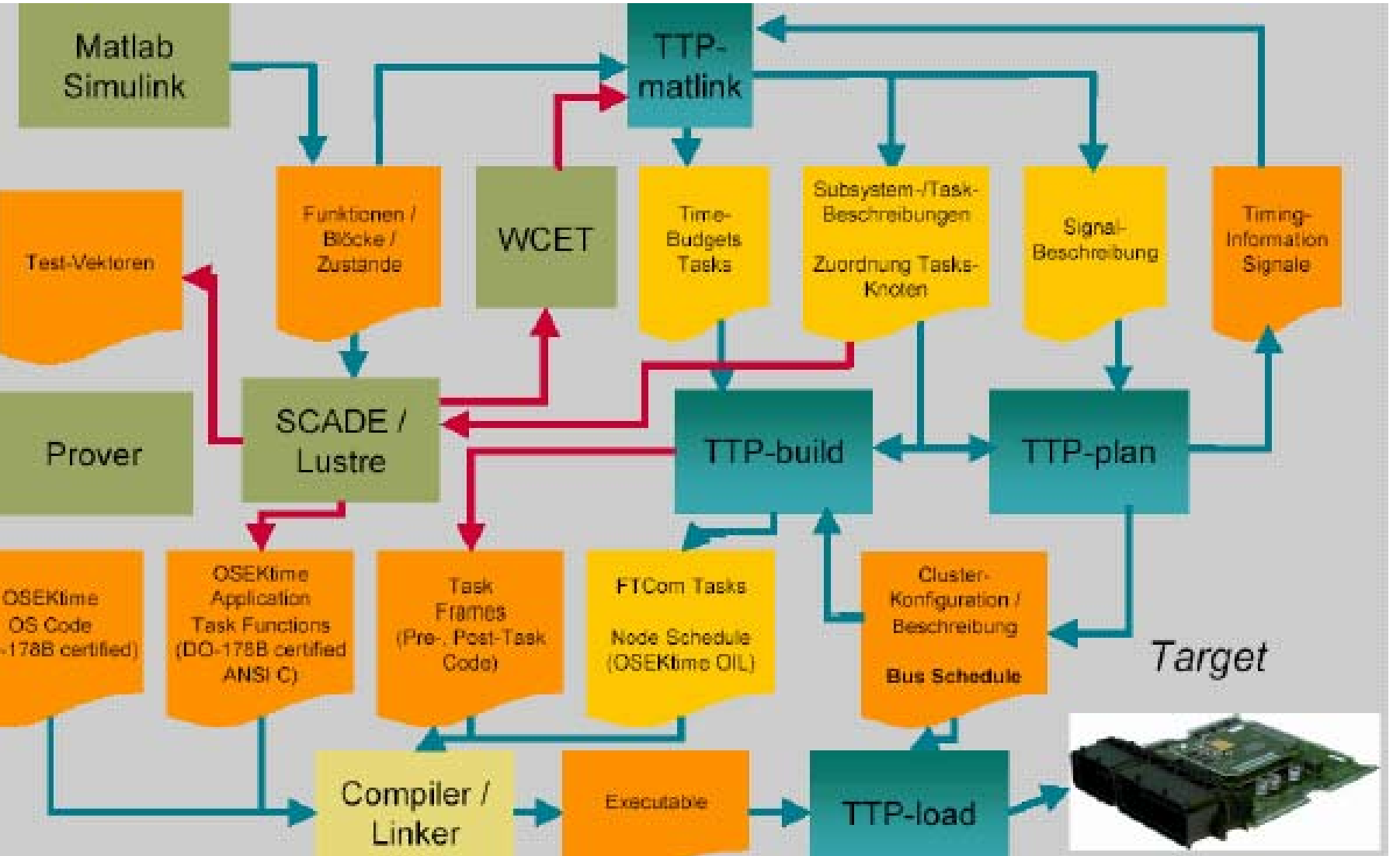


Multiple stacks may be needed

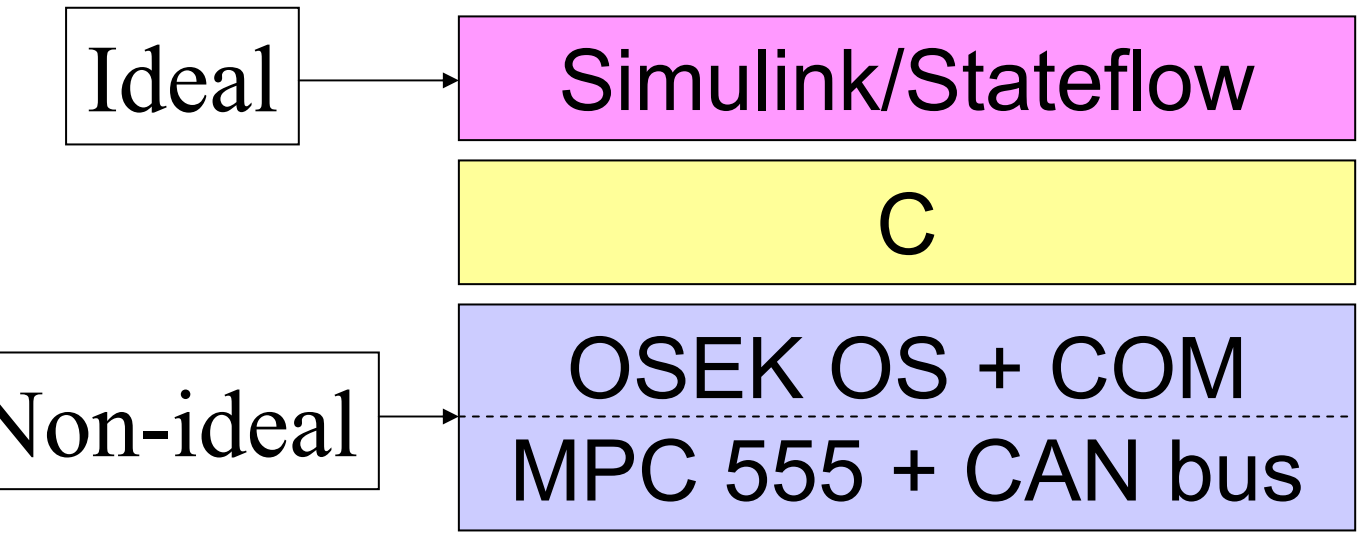
- The “integration problem”:



The integrated tool-chain

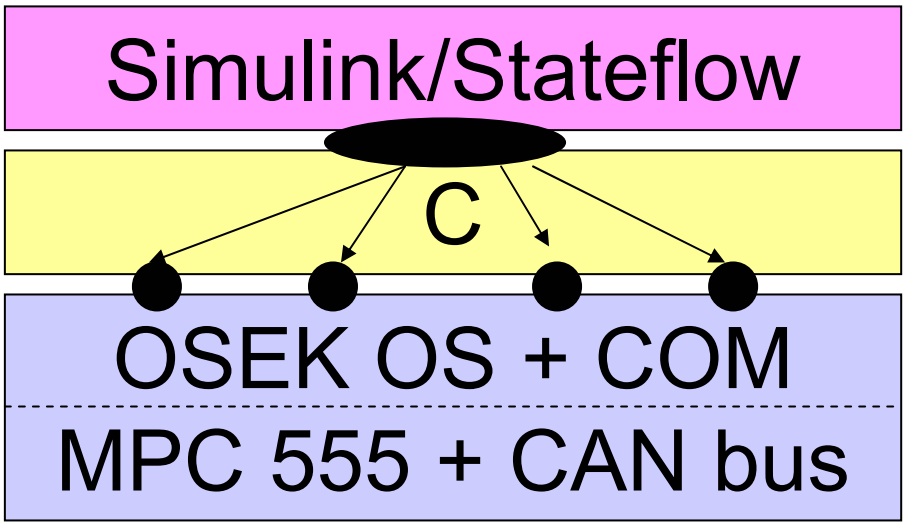


The resource allocation problem



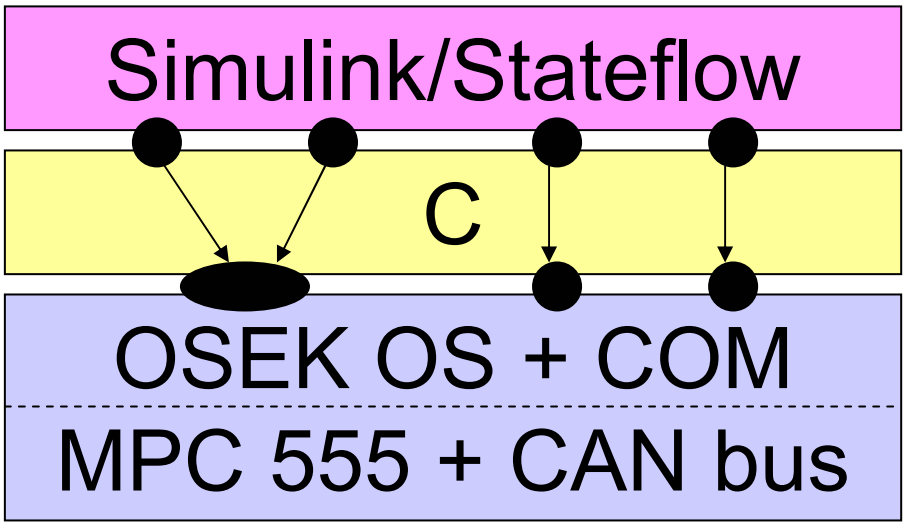
The resource allocation problem

- Can be solved at various levels:

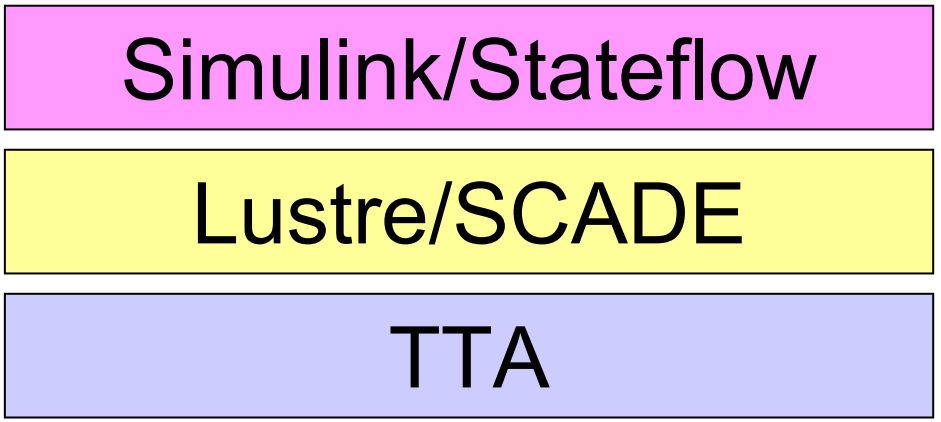


The resource allocation problem

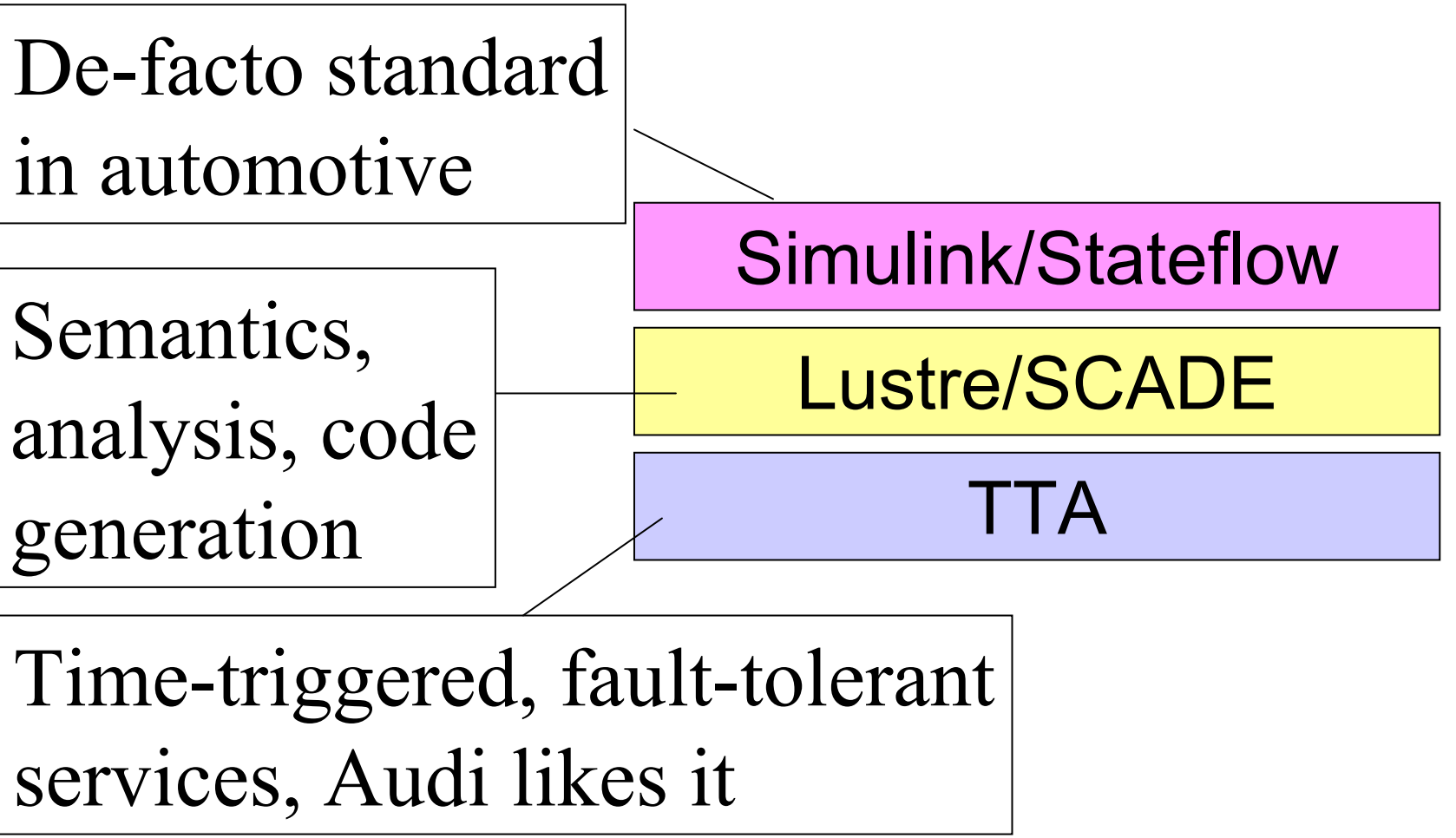
- Can be solved at various levels:



Our work

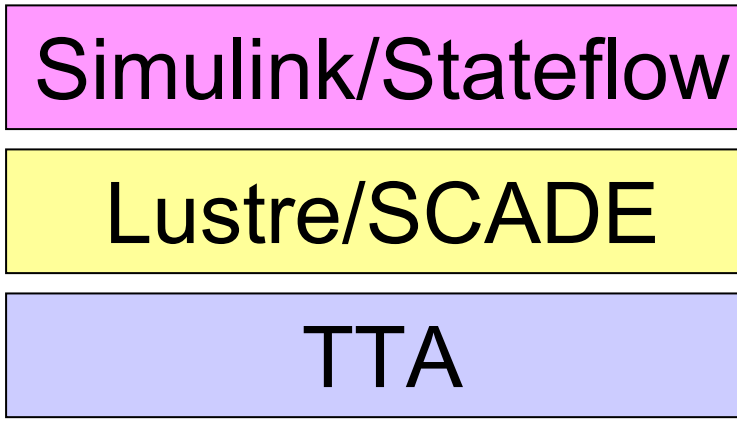


Our work


De-facto standard in automotive
Simulink/Stateflow
Semantics, analysis, code generation
Lustre/SCADE
TTA
Time-triggered, fault-tolerant services, Audi likes it

The development process

1. Design your controller in Simulink/Stateflow.
2. Translate it to Lustre.
3. Verify the Lustre program (transparent).
4. Distribute the Lustre program on TTA.
5. Generate code, compile and run.



Goal: Automatic as much as possible!

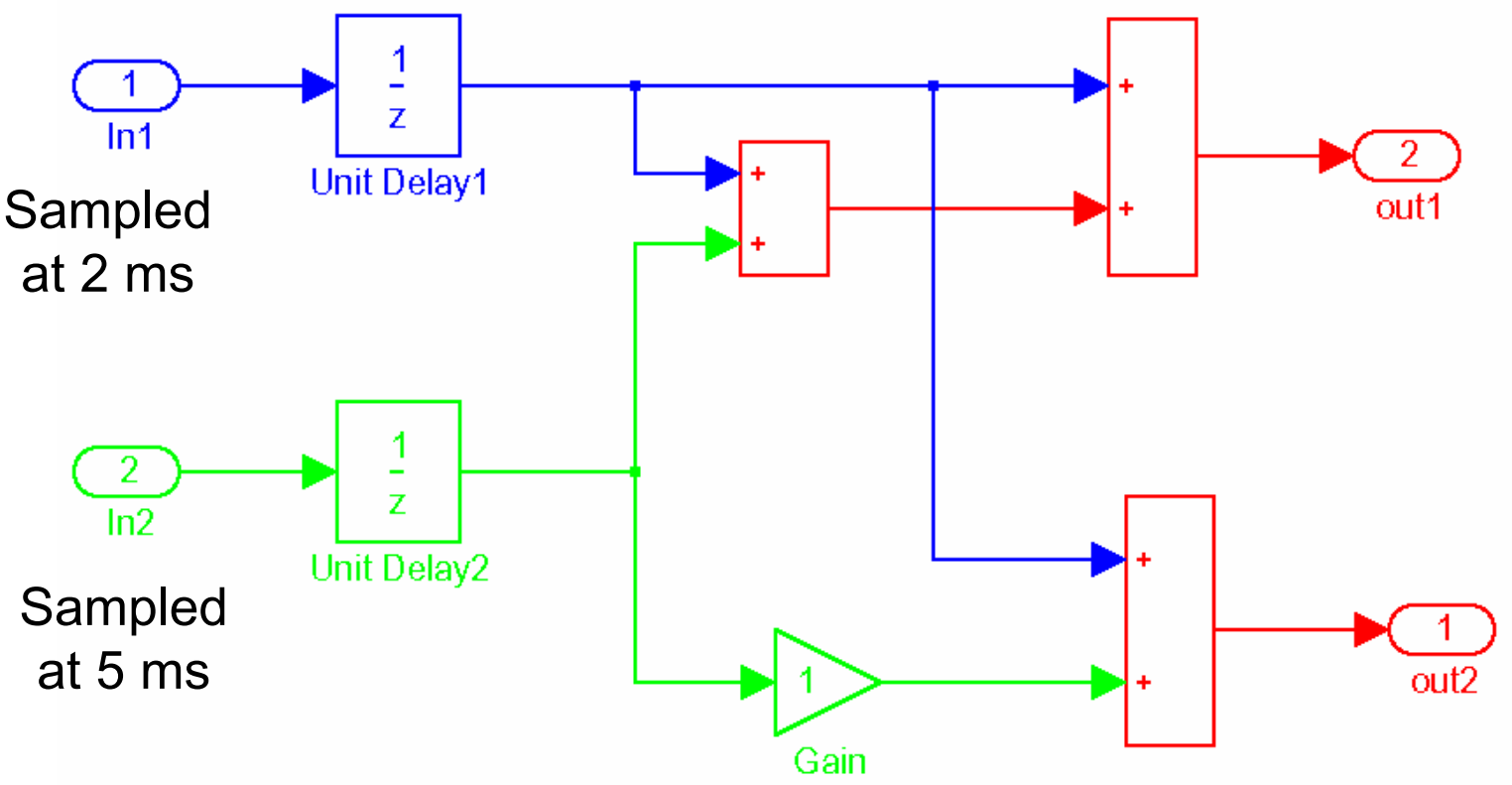
Two papers

- "From Simulink to Lustre to TTA":
  - In "Languages, Compilers and Tools for Embedded Systems" (LCTES'03).
- "Translating Discrete-time Simulink to Lustre":
  - In EMSOFT'03.
  - Talk tomorrow morning.

Translating Simulink into Lustre

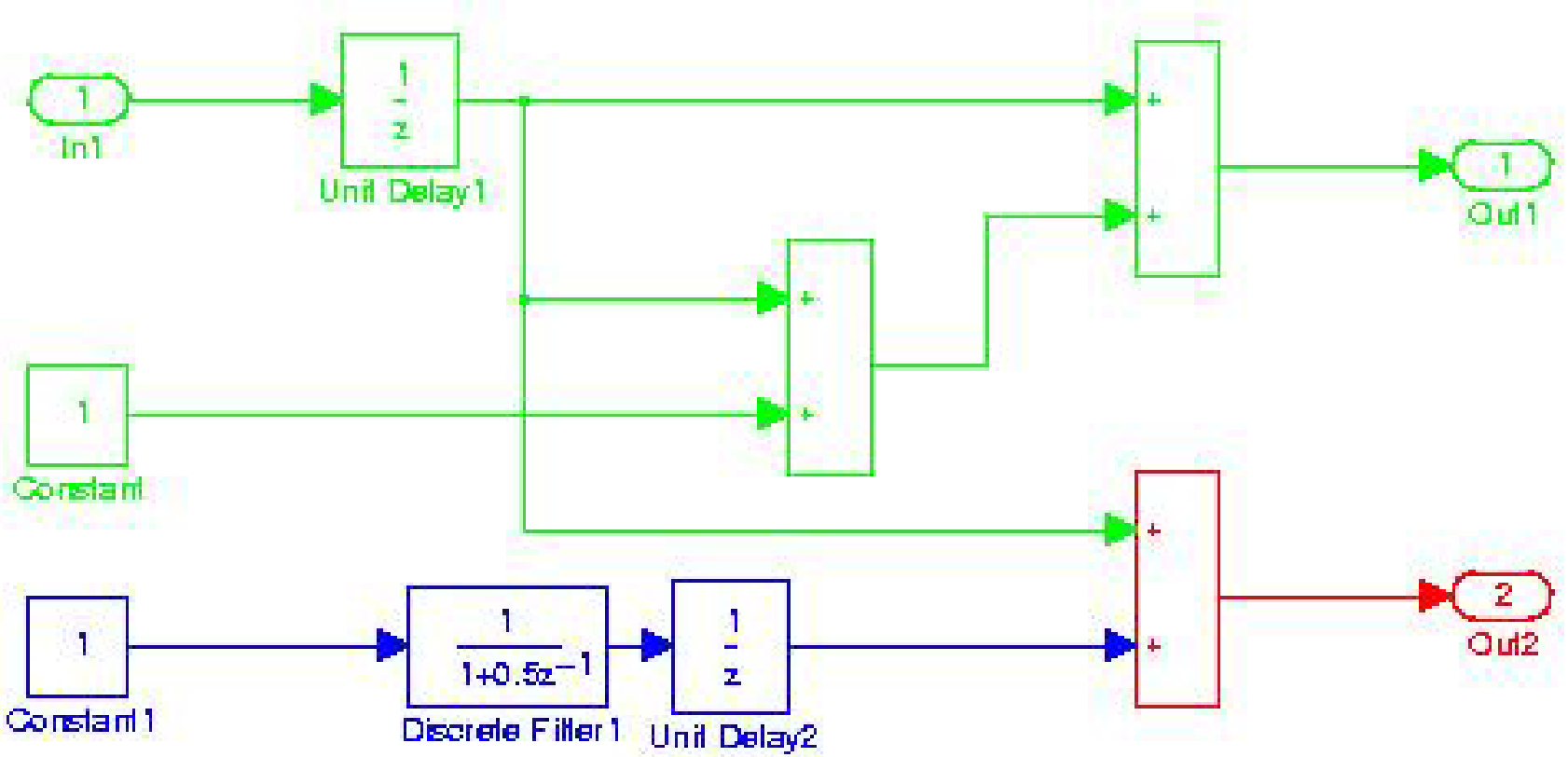
- Goal: implement the controller(s).
- Assumption: the controller is designed using the Discrete-time Simulink library.
- Translation: easy, *once you figure out the semantics of Simulink*.

Simulink features I



Simulink rejects the model with the Gain.
It accepts the model if the Gain is removed !

Simulink features II



Simulink rejects the model with two constant blocks.
It accepts it if they are replaced by one block !

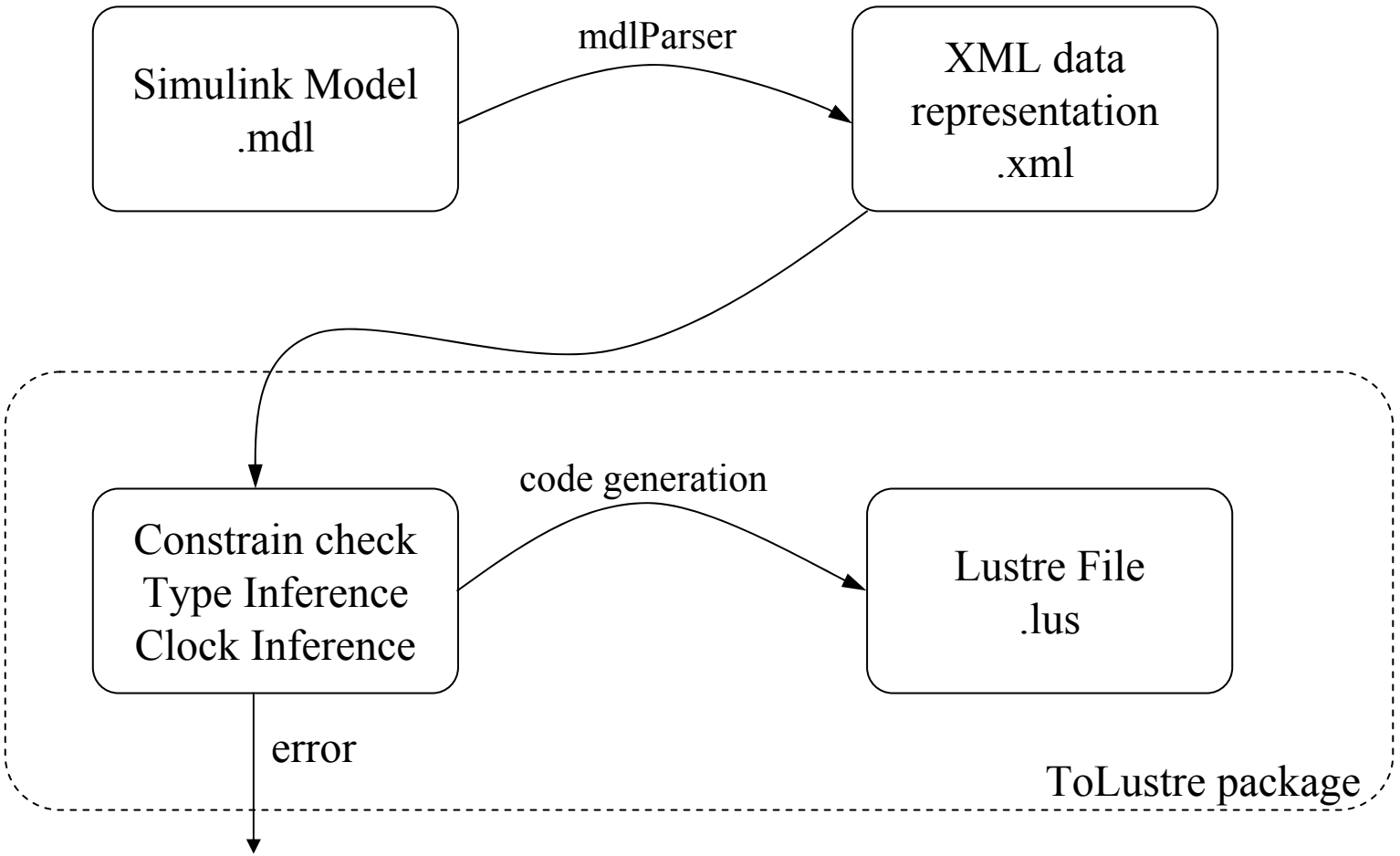
Translating Stateflow

- Things get worse…
  - Entry, during, exit, …, actions.
  - Semantics depends on graphical layout
    - "top-to-bottom, left-to-right" rule for states,
    - "clock-wise" rule for transitions.
  - Backtracking in the middle of a transition.
  - Non-terminating transitions…

Lesson learned

- Engineers (of today) tolerate this.
- Why?
  - The design tool is like any other tool: you have to learn to use it.
  - Power more important than sanity/simplicity.
- BUT: will gladly use anything that can help !
  - Type checking, static analysis, verification, …

Prototype translator: Sim2Lus



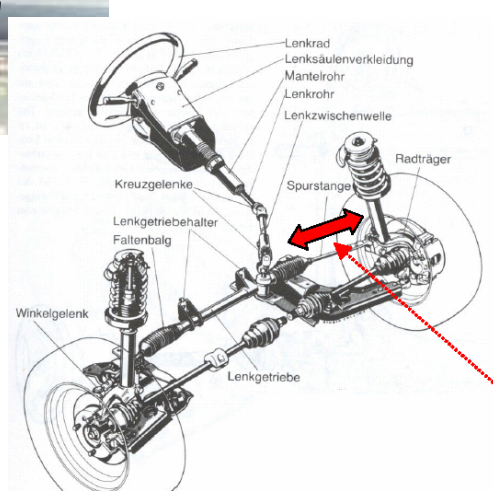
Case studies

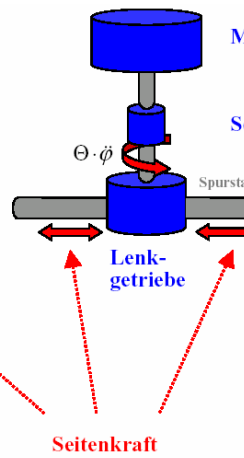
- Translated two case studies from Audi.
  - A warning-filtering system:
    - 6 levels, 20 subsystems, 113 total blocks.
    - 800 lines of generated Lustre code.
  - A steer-by-wire application:
    - 6 levels, 18 subsystems, 157 total blocks.
    - 387 lines of generated Lustre code.
    - Upcoming demo for closing NEXT TTA review: Jan 2004.

A steer-by-wire application

Consists of:

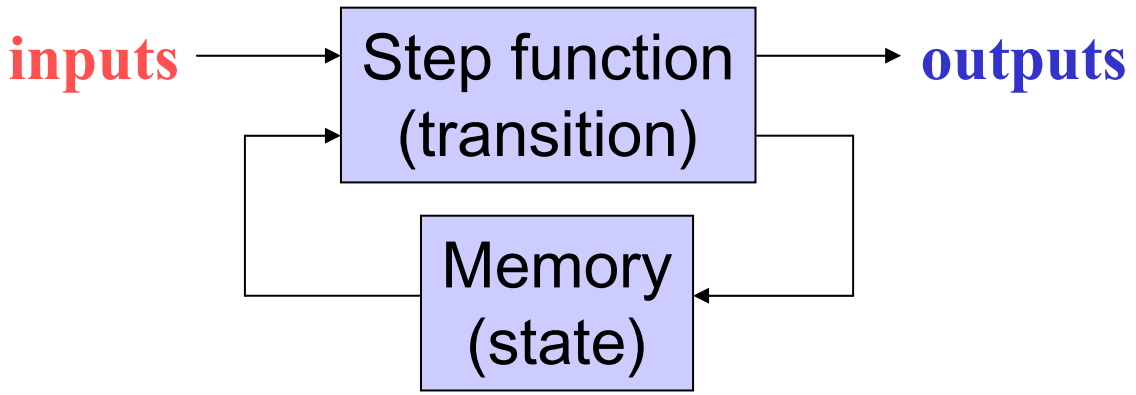
- camera
- steering actuator



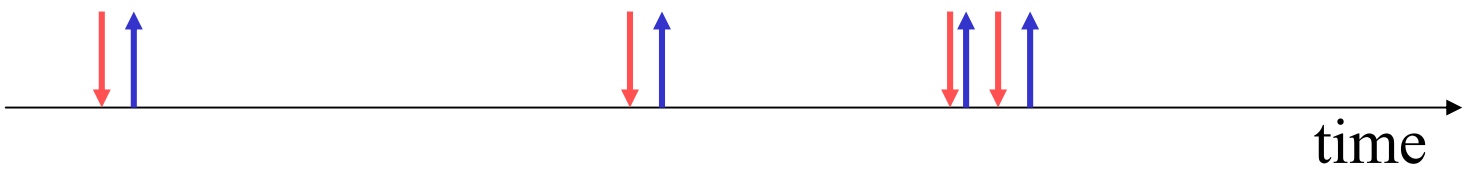


A glance into Lustre

- A Lustre program models an I/O automaton:



- Semantics (synchrony hypothesis):



Why Lustre ?

- The translation **does not** solve the resource allocation problem, but:
- It offers formal semantics:
  - verification, testing, thm proving possible
- It brings us closer to implementation:
  - strong typing, type modularity, …
  - DO178B-level A certified C code generator (by Esterel Technologies).

Distributing Lustre on TTA

- A resource allocation problem:
  - computation is not free
  - parallelism (communication) is not free

- First, a description problem:
  - how to *express* available/required resources
  - few languages support this today

Lustre extensions

- Code distribution
  - available processors
  - assignment of modules to processors
- Required computation resources
  - WCETs of modules on certain processors
- Required communication resources
  - Size of messages or network speed/overhead
- Other constraints
  - Deadlines, “freshness” constraints, ...

Back to resource allocation...

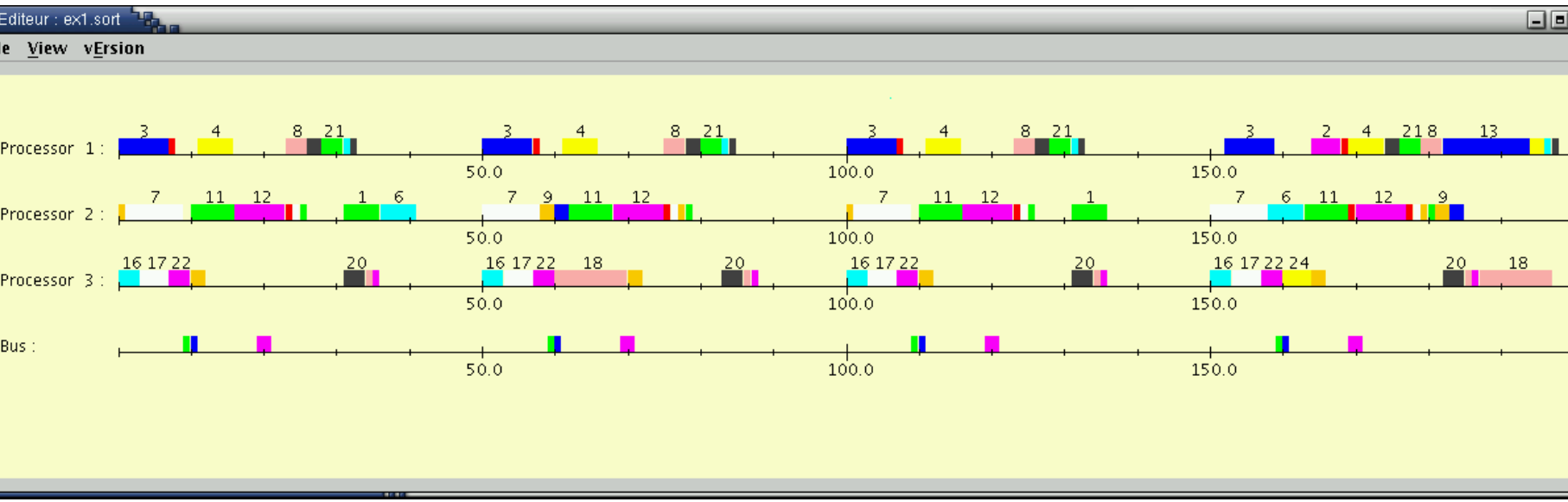
- Two possibilities:
- "I know what I want"
  - distribution, scheduling policy (e.g., priorities) are fixed by the designer, but do they work?
  - verification problem
  - classic theory often inadequate (c.f. TAXYS)
- "Help me"
  - partly known, designer wants to try alternatives
  - a blend of verification, and synthesis

Bus/Processor Scheduling

- Schedule tasks on processors and messages on the TTA bus.
  - Assignment of tasks to processors assumed known.
  - WCETs assumed known.
  - TTA is a synchronous bus (global clock).
- NP-hard.
- Developed branch-and-bound algorithm.

A result of the scheduling tool

- 21 tasks on 3 processors:



Scheduling can become harder...

- Distribution of tasks unknown:
  - where to assign tasks ?
  - to compute or to communicate ?
- Environment not (multi-)periodic.
- Network not synchronous:
  - e.g., find priorities for CAN messages, jitter, ...
- Adaptive to changes on-line:
  - a task finishing earlier may violate freshness

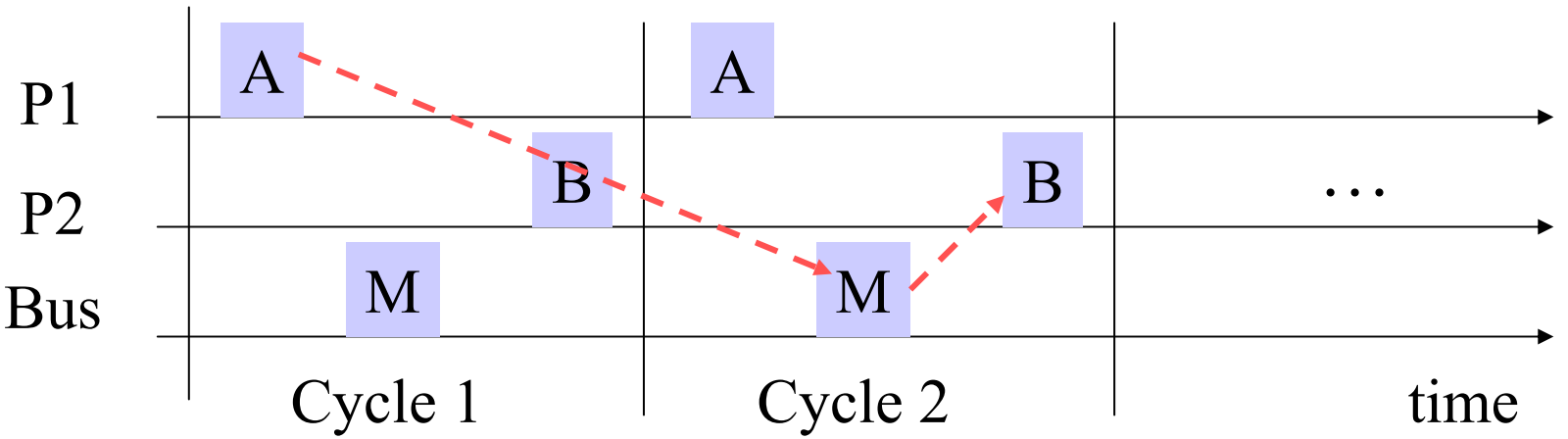
Did we forget something ?

- Preservation:
  - What happened to the semantics of the original Simulink model / Lustre program ?
  - Most current (commercial) tools forget this.
  - Back to simulation, testing, …
- Glue code is often necessary.

Example

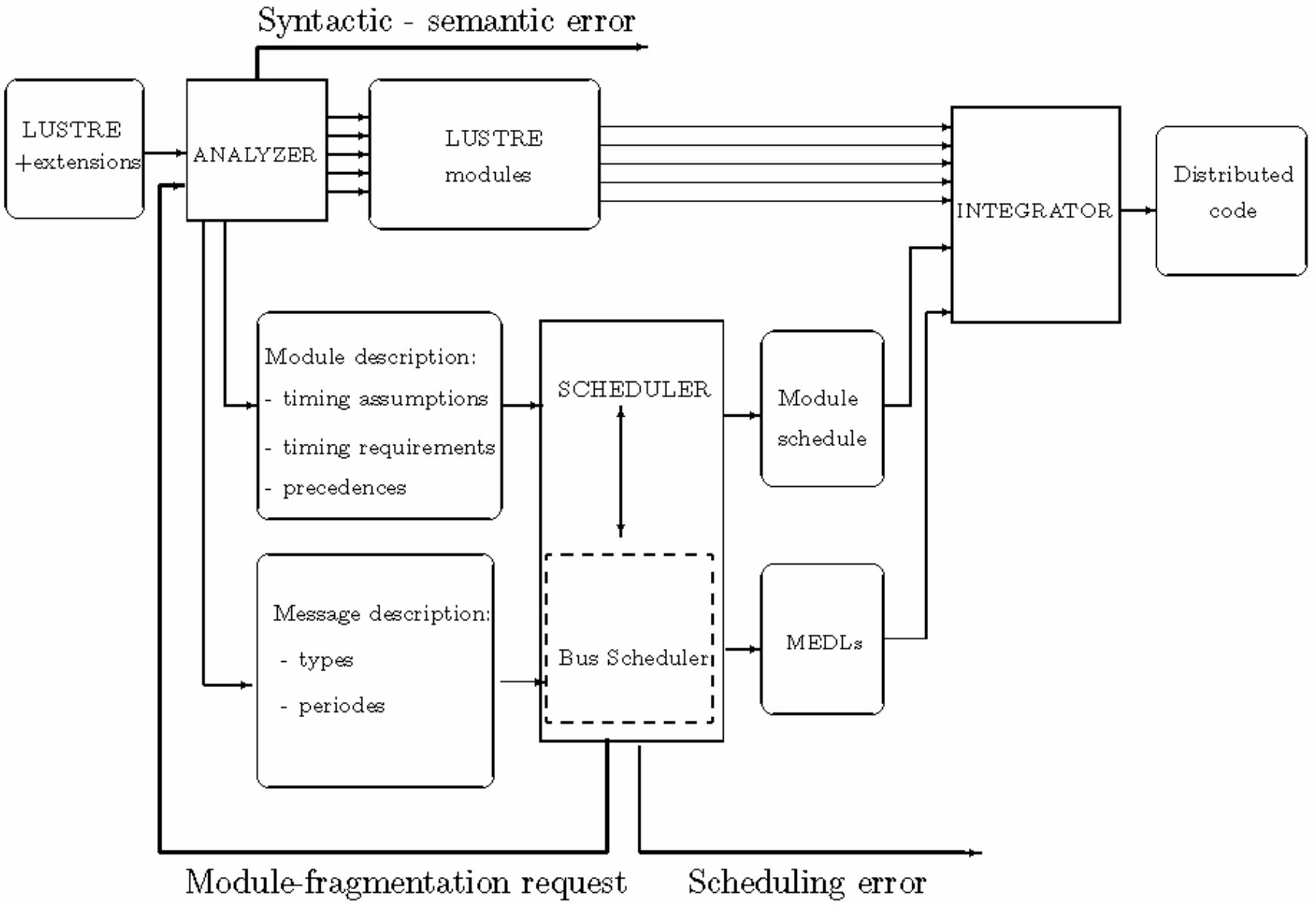
- Original Lustre program:

- A possible schedule:



- Glue code must remember previous value produced by A.

Tool chain


Syntactic - semantic error
LUSTRE +extensions
ANALYZER
LUSTRE modules
INTEGRATOR
Distributed code
Module description:
- timing assumptions
- timing requirements
- precedences
SCHEDULER
Module schedule
Message description:
- types
- periodes
Bus Scheduler
MEDLs
Module-fragmentation request
Scheduling error

Conclusions

- End-to-end tool-chain:
  - from design to implementation to execution.
- Need better models:
  - simple, intuitive semantics
- Need analysis tools
- should not have to repeat analysis twice 
- Need powerful resource allocation techniques
  - that do not forget semantic preservation