Challenges in ES Design

Peter Marwedel
TU Dortmund,
Informatik 12

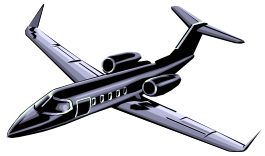
2010/09/20

Quite a number of challenges, e.g. dependability

Dependability?

- Non-real time protocols used for real-time applications (e.g. Berlin fire department)

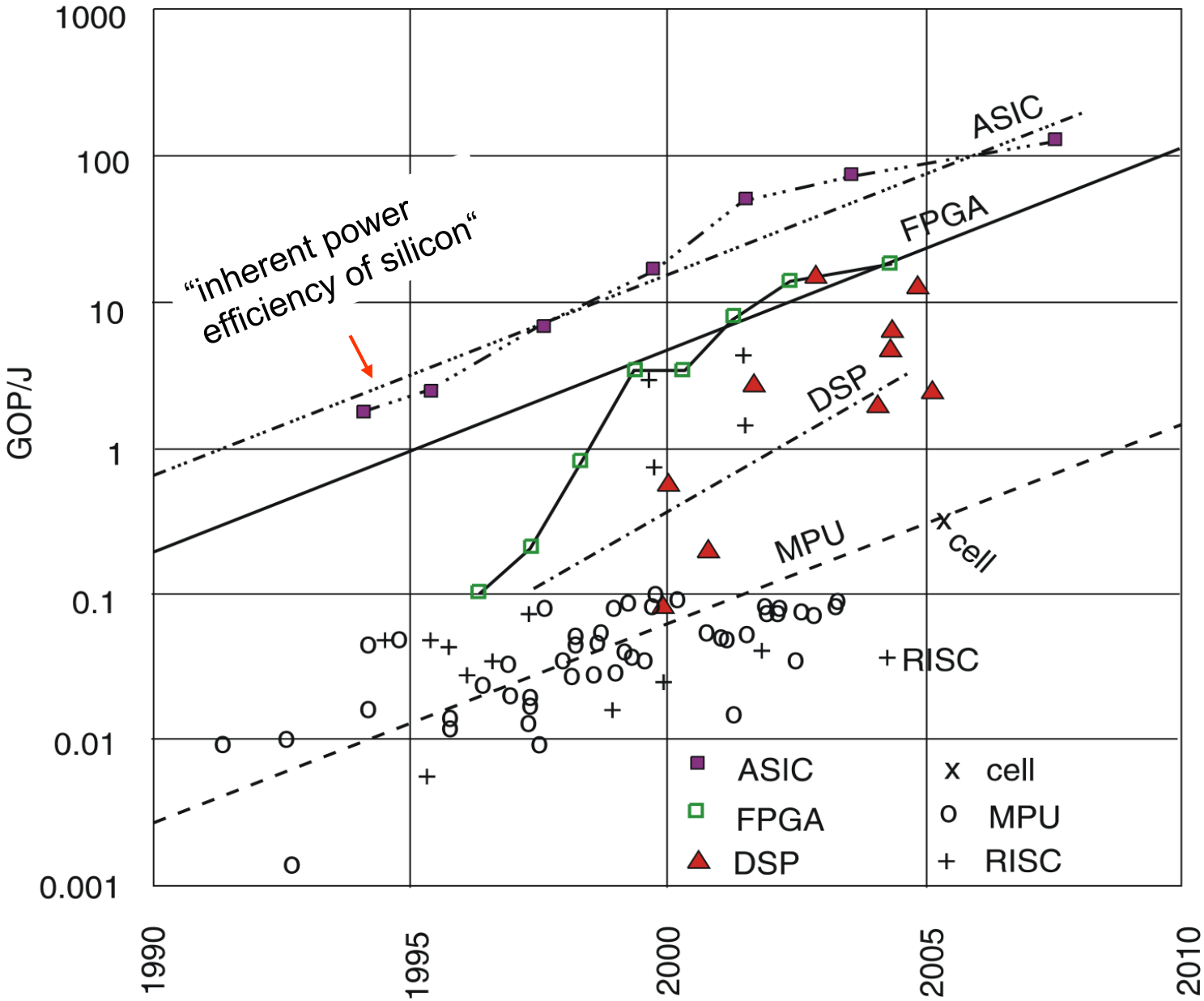
- Over-simplification of models (e.g. aircraft anti-collision system)



- Using unsafe systems for safety-critical missions (e.g. voice control system in Los Angeles; ~ 800 planes without voice connection to tower for > 3 hrs

Importance of Energy Efficiency

Efficient software design needed, otherwise, the price for software flexibility cannot be paid.

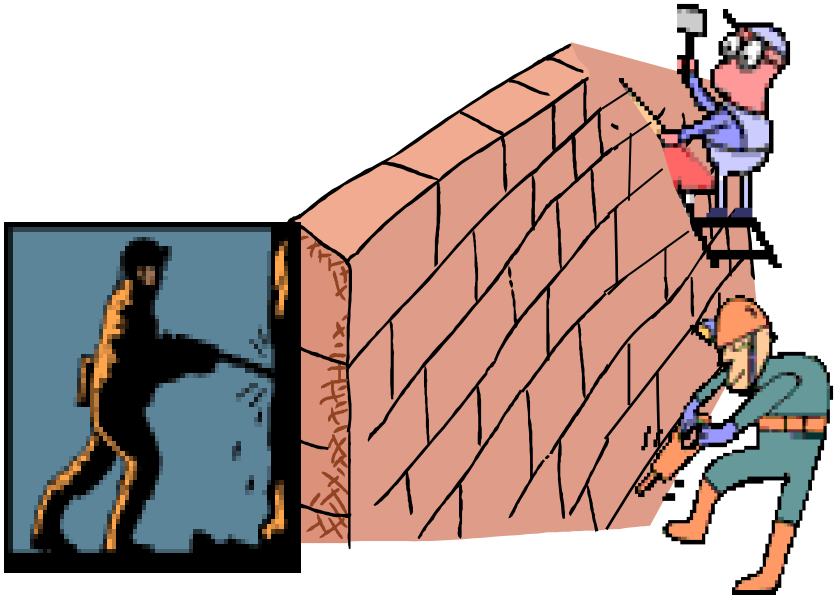


It is not sufficient to consider ES just as a special case of software engineering

EE knowledge must be available,
Walls between EE and CS must be torn down

CS

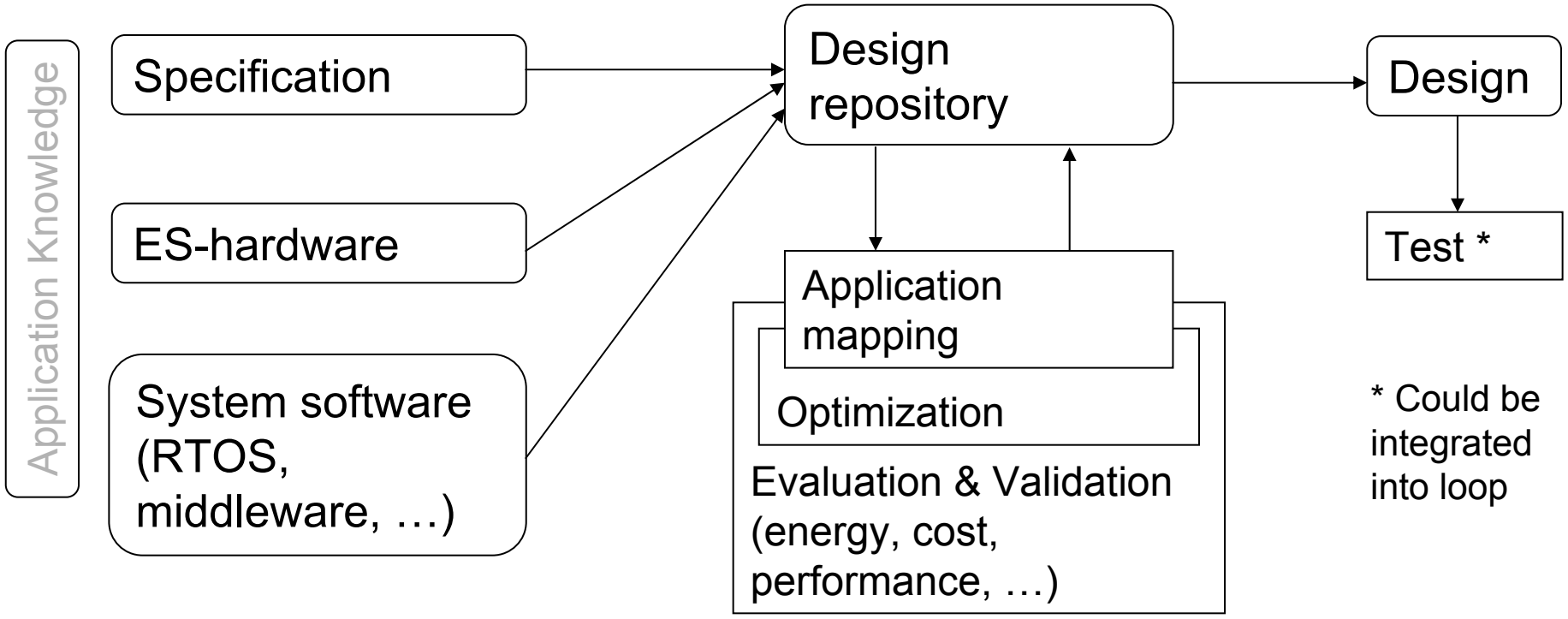
EE



The same for walls to other disciplines and more challenges ….

Design flows

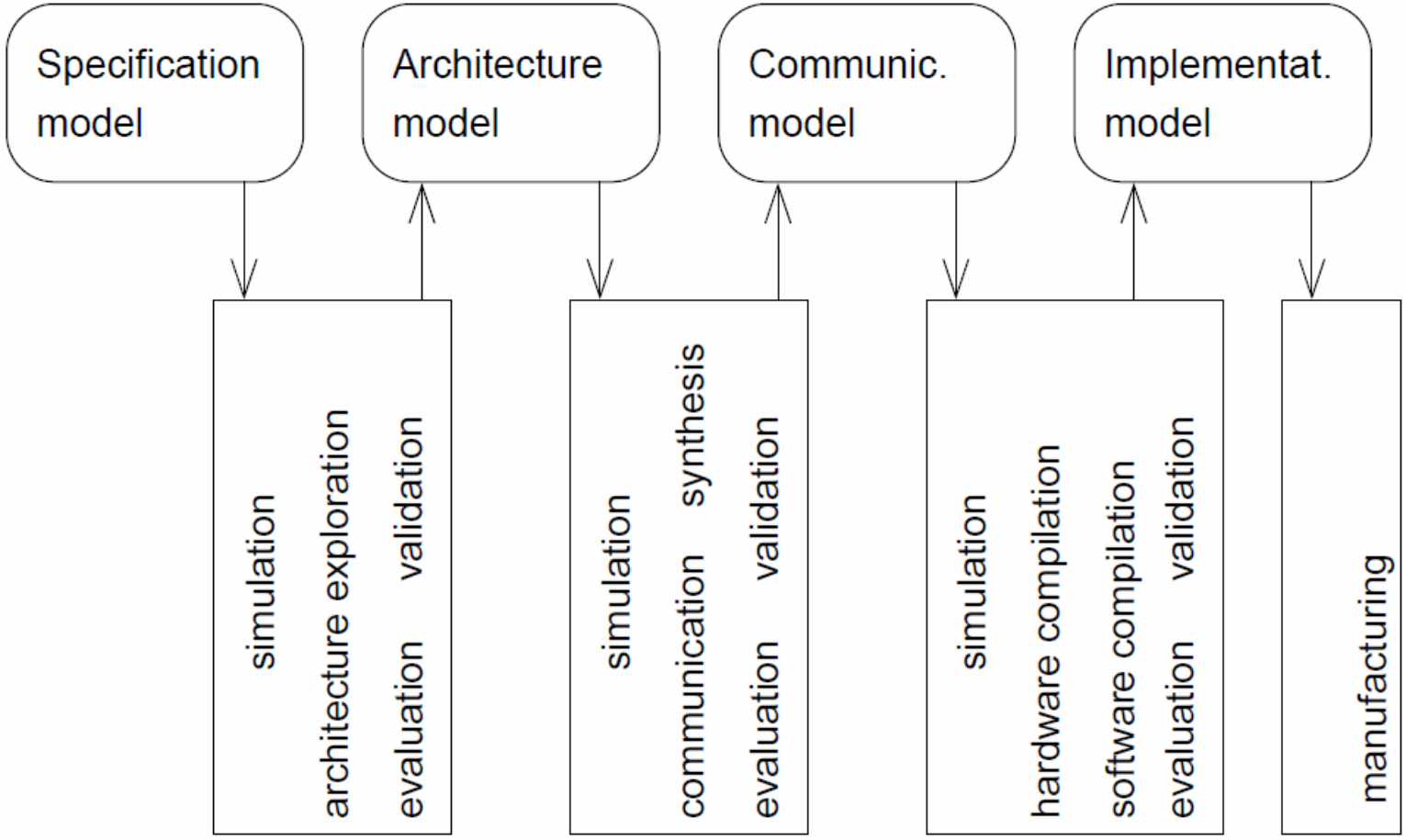
Hypothetical design flow



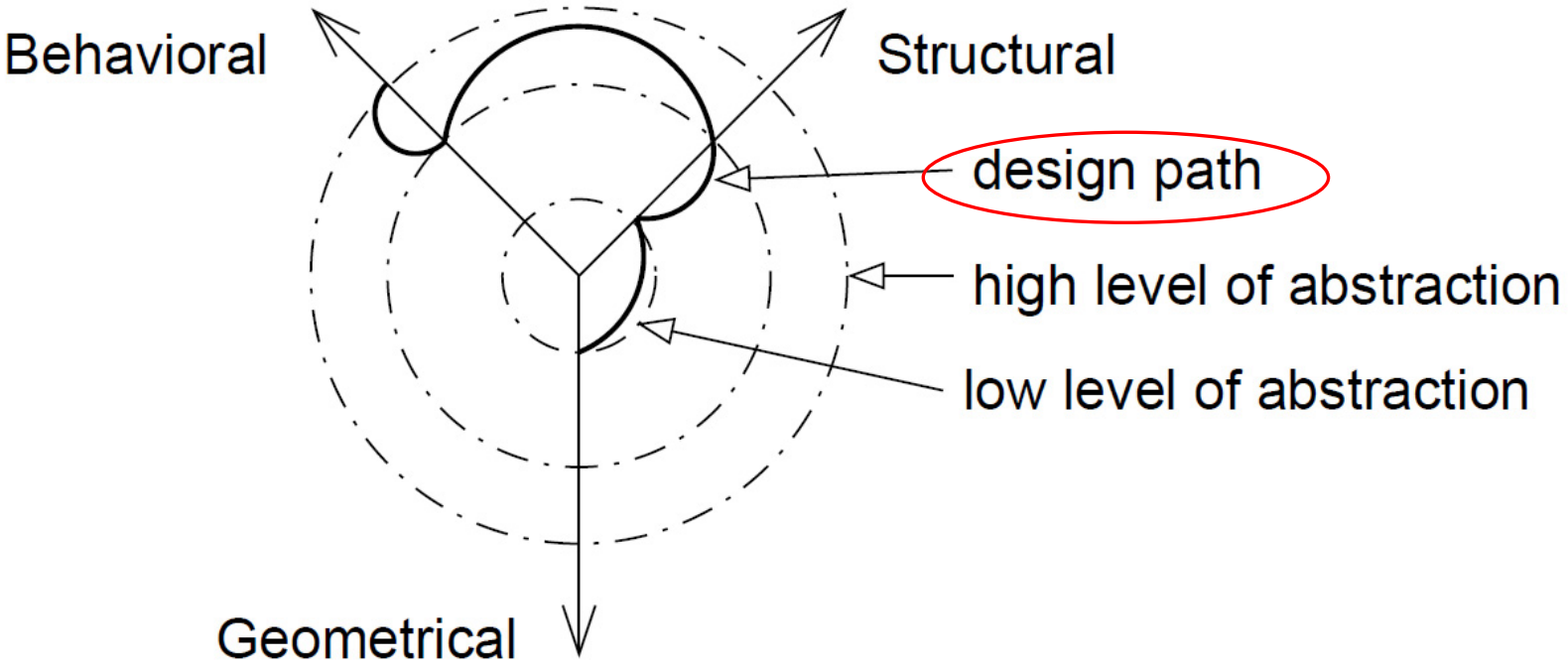
Generic loop: tool chains differ in the number and type of iterations

Iterative design (1)
- After unrolling loop -

Example: SpecC tools

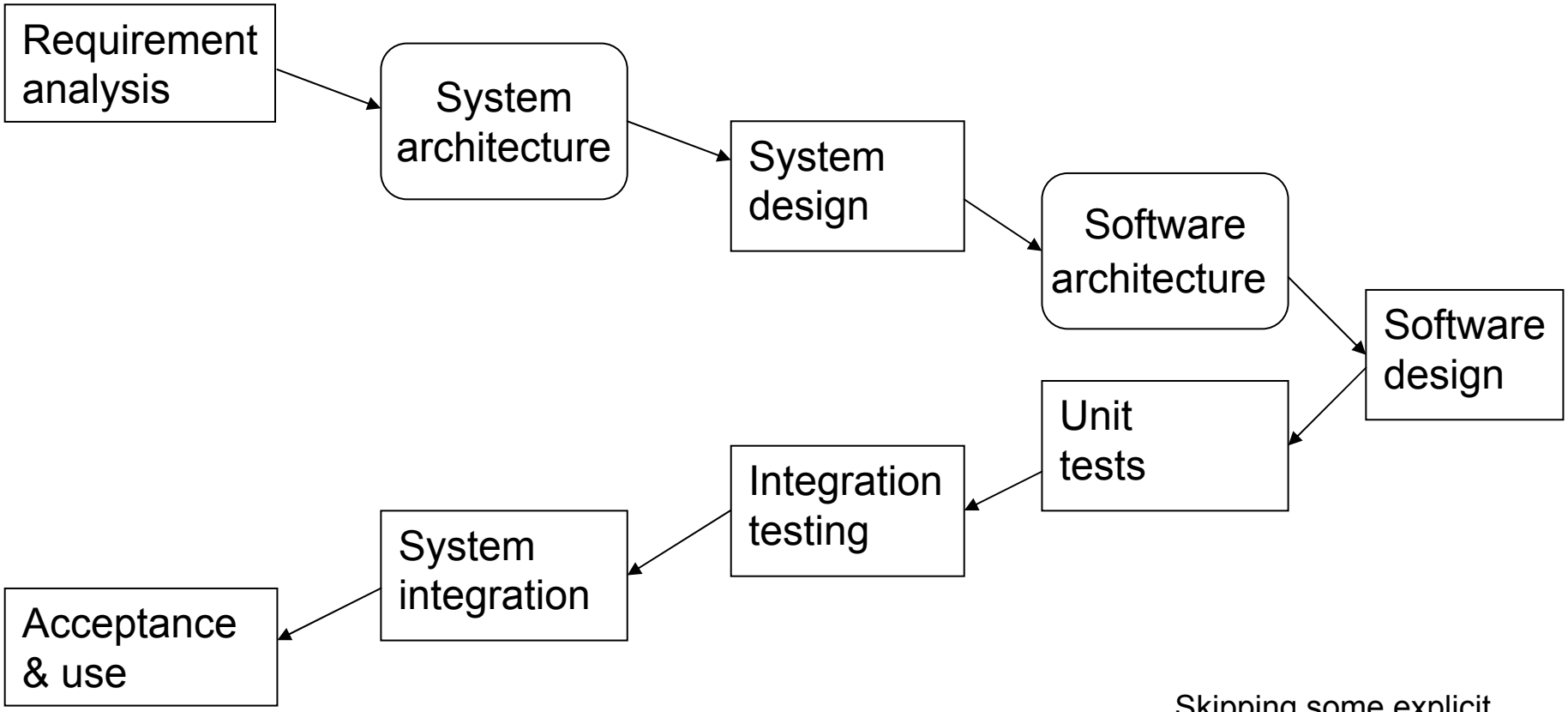


Iterative design (2)
- Gajski‘s Y-chart -



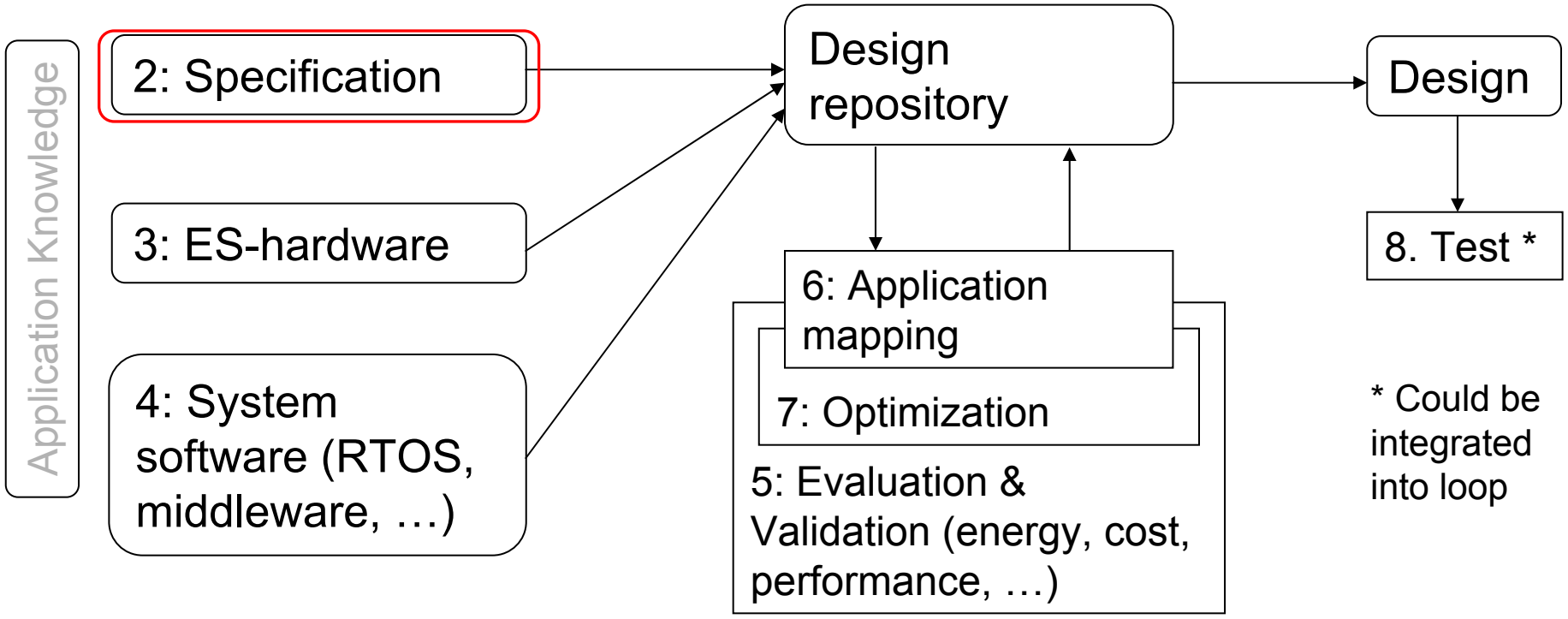
Iterative design (3)
- After unrolling loop -

Example: V-model



Skipping some explicit repository updates ..

Hypothetical design flow



Numbers denote sequence of chapters

Motivation for considering specs

- Why considering specs?
- If something is wrong with the specs, then it will be difficult to get the design right, potentially wasting a lot of time.
- Typically, we work with **models** of the **system under design** (SUD)

☞ What is a *model* anyway?

Models

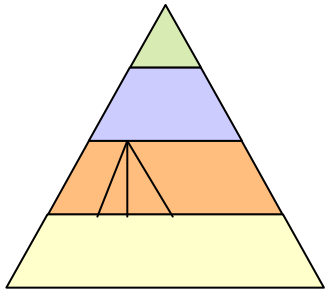
Definition: *A model is a simplification of another entity, which can be a physical thing or another model. The model contains exactly those characteristics and properties of the modeled entity that are relevant for a given task. A model is minimal with respect to a task if it does not contain any other characteristics than those relevant for the task*.

[Jantsch, 2004]:

Which requirements do we have for our models?

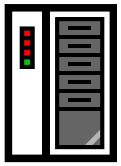
Requirements for specification techniques: Hierarchy

Hierarchy
Humans not capable to understand systems containing more than ~5 objects.
Most actual systems require more objects
☞ Hierarchy



proc
proc
proc

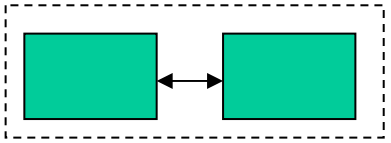
- Behavioral hierarchy
  Examples: states, processes, procedures.



- Structural hierarchy
  Examples: processors, racks, printed circuit boards

Requirements for specification techniques (2): Component-based design

- Systems must be designed from components
- Must be “easy” to derive behavior from behavior of subsystems



☞Work of Sifakis, Thiele, Ernst, …

- Concurrency
- Synchronization and communication

Requirements for specification techniques (3): Timing

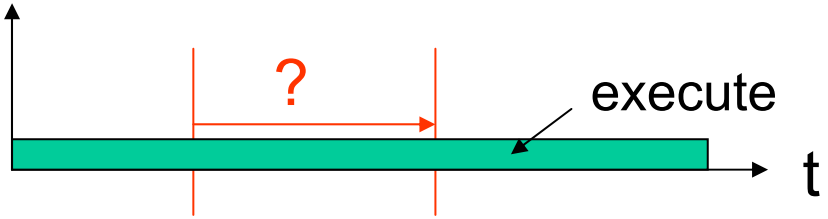
- **Timing behavior**
  Essential for embedded and cy-phy systems!
  - **Additional information (periods, dependences, scenarios, use cases) welcome**
  - **Also, the speed of the underlying platform must be known**
  - **Far-reaching consequences for design processes!**

"The lack of timing in the core abstraction (of computer science) *is a flaw, from the perspective of embedded software"* [Lee, 2005]

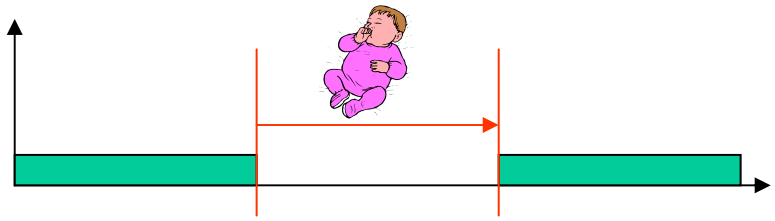
Requirements for specification techniques (3): Timing (2)

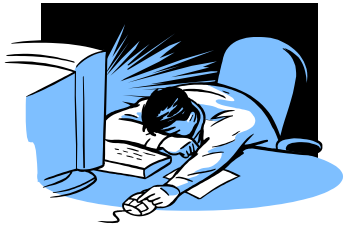
4 types of timing specs required, according to Burns, 1990:

1. Measure elapsed time
   Check, how much time has elapsed since last call



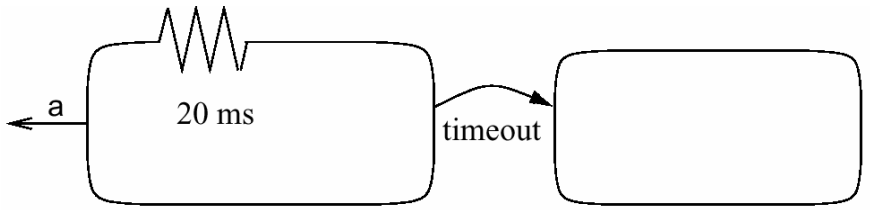
2. Means for delaying processes



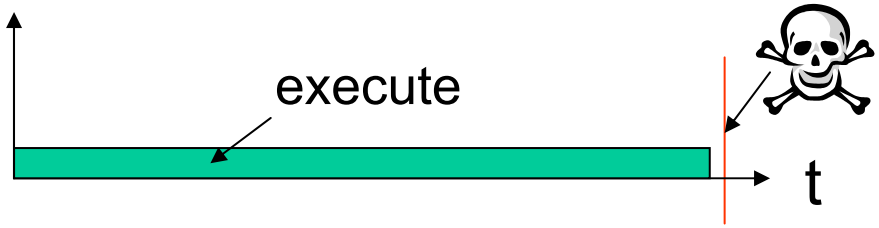


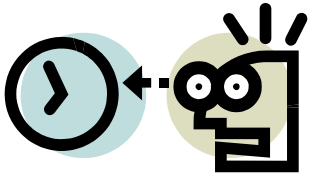
Requirements for specification techniques (3)
Timing (3)

3. Possibility to specify timeouts
   Stay in a certain state a maximum time.



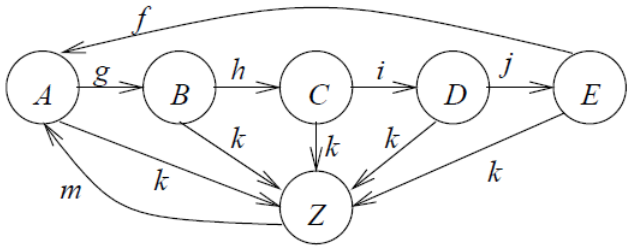
4. Methods for specifying deadlines
   Not available or in separate control file.





Specification of embedded systems (4): Support for designing reactive systems

- **State-oriented behavior**
  Required for reactive systems; classical automata insufficient.
- **Event-handling**
  (external or internal events)
- **Exception-oriented behavior**
  Not acceptable to describe exceptions for every state



We will see, how all the arrows labeled k can be replaced by a single one.

Requirements for specification techniques (5)

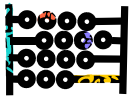
- **Presence of programming elements**
- **Executability** (no algebraic specification)
- **Support for the design of large systems** (☞ OO)
- **Domain-specific support**
- **Readability**
- **Portability and flexibility**
- **Termination**
- **Support for non-standard I/O devices**
- **Non-functional properties**
- **Support for the design of dependable systems**
- **No obstacles for efficient implementation**
- **Adequate model of computation**

  What does it mean “to compute”?







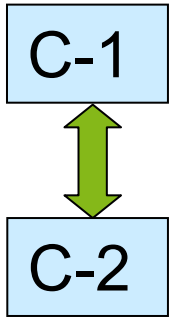


Models of computation

What does it mean, "to compute"?

Models of computation define:

- Components and an execution model for computations for each component
- Communication model for exchange of information between components.



Communication

- **Shared memory**

Variables accessible to several components/tasks.

Model mostly restricted to local systems.

Shared memory

Potential race conditions (☞inconsistent results possible)
☞ Critical sections = sections at which exclusive access to resource *r* (e.g. shared memory) must be guaranteed.

```
task a {
  ..
  P(S)  //obtain lock
  ..    // critical section
  V(S)  //release lock
}
```

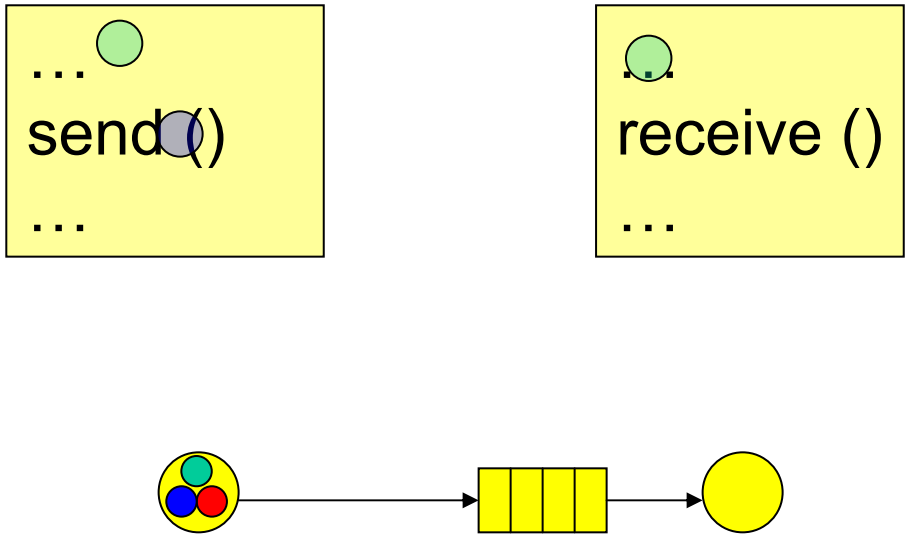
```
task b {
  ..
  P(S)  //obtain lock
  ..    // critical section
  V(S)  //release lock
}
```

Race-free access to shared memory protected by S possible

P(S) and V(S) are **semaphore** operations,
allowing at most n accesses, n =1 in this case (mutex, lock)

Non-blocking/asynchronous message passing

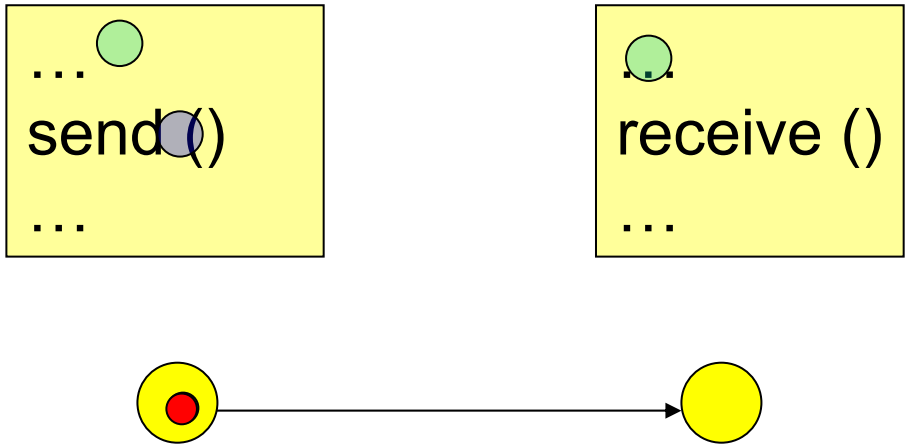
Sender does not have to wait until message has arrived;



Potential problem: buffer overflow

Blocking/synchronous message passing *rendez-vous*

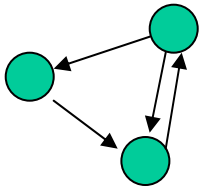
Sender will wait until receiver has received message



No buffer overflow, but reduced performance.

Organization of computations within the components (1)

- Finite state machines

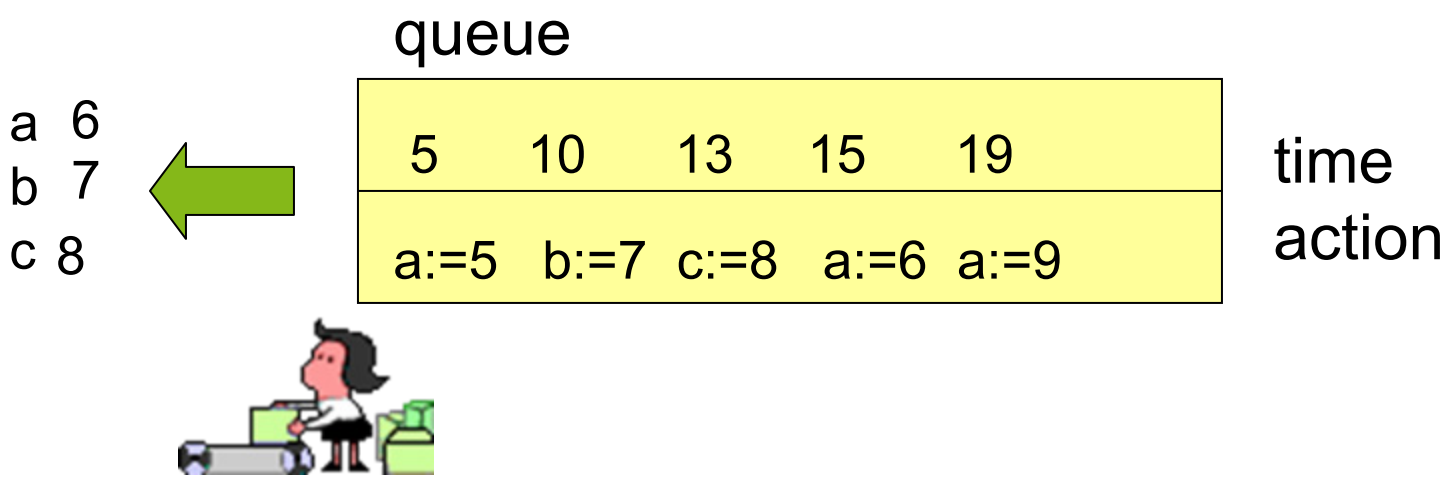


- Data flow
  (models the flow of data in a distributed system)
  ☞ to be discussed next week

- Petri nets
  (models synchronization in a distributed system)
  ☞ to be discussed next week

Organization of computations within the components (2)

- Discrete event model



- Von Neumann model

Sequential execution, program memory etc.

Organization of computations within the components (3)

- Differential equations

$$\frac{\partial^2 x}{\partial t^2} = b$$

Models of computation considered in this course

Communication/ local computations	Shared memory	Message passing Synchronous | Asynchronous	
Communicating finite state machines	StateCharts		SDL
Data flow	(Not useful)		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, …	
Discrete event (DE) model	VHDL, Verilog, SystemC, …	Only experimental systems, e.g. distributed DE in Ptolemy	
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	

Summary

- Challenges in embedded system design
  - Dependability, efficiency, …
- Design flows
- Structure of this course
- Requirements for specification techniques
- Models of computation
  - Local model
  - Communication