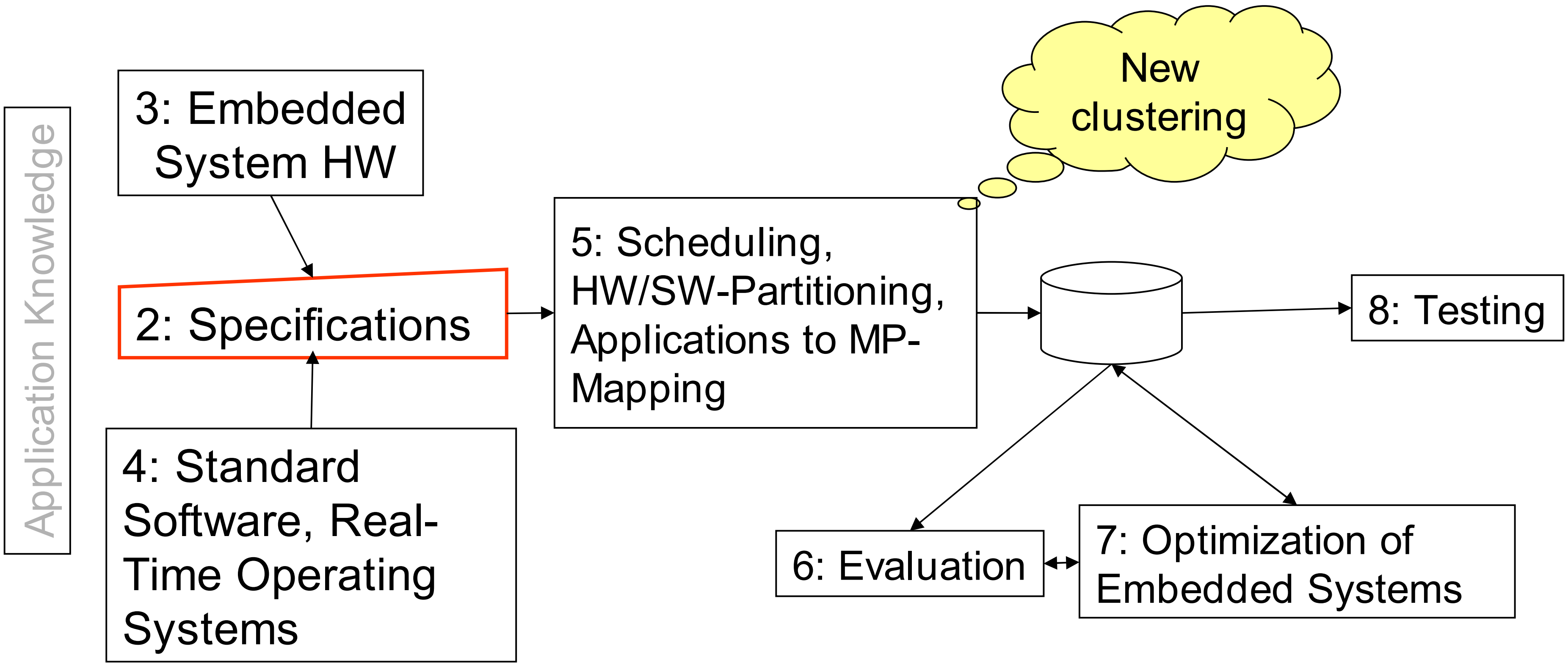
Specifications

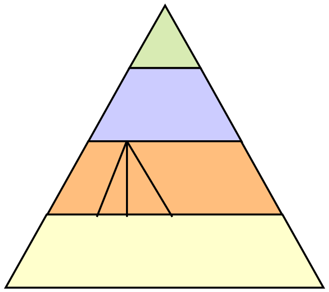
Peter Marwedel
Informatik 12
TU Dortmund
Germany

Structure of this course



Specification of embedded systems: Requirements for specification techniques (1)

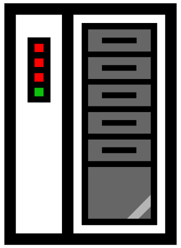
- **Hierarchy**
  Humans not capable to understand systems containing more than ~5 objects.
  Most actual systems require more objects
  ☞ Hierarchy



  - Behavioral hierarchy
    Examples: states, processes, procedures.

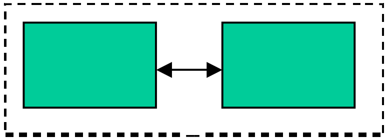
proc
  proc
    proc

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

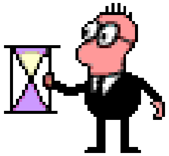


Specification of embedded systems: Requirements for specification techniques (2)

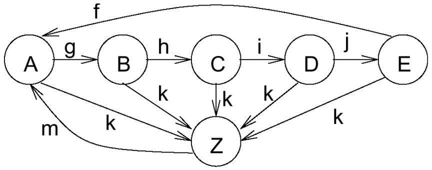
- **Compositional behavior**
  Must be "easy" to derive behavior from behavior of subsystems



- **Timing behavior.**

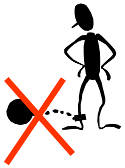


- **State-oriented behavior**
  Required for reactive systems; classical automata insufficient.



- **Event-handling**
  (external or internal events)

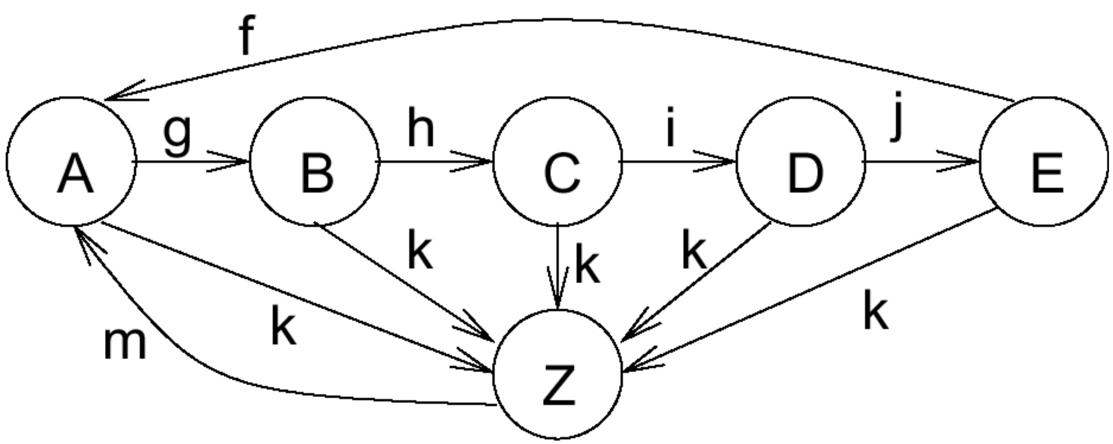
- **No obstacles for efficient implementation**



Requirements for specification techniques (3)

- **Support for the design of dependable systems**
  Unambiguous semantics, ...

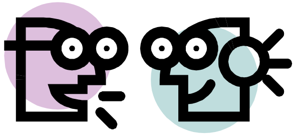
- **Exception-oriented behavior**
  Not acceptable to describe exceptions for every state.



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

Requirements for specification techniques (4)

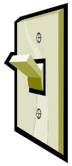
- **Concurrency**
  Real-life systems are concurrent
- **Synchronization and communication**
  Components have to communicate!
- **Presence of programming elements**
  For example, arithmetic operations, loops, and function calls should be available
- **Executability** (no algebraic specification)
- **Support for the design of large systems** (☞ OO)
- **Domain-specific support**



Requirements for specification techniques (5)

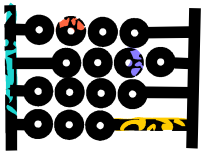

- **Readability**
- **Portability and flexibility**
- **Termination**
  It should be clear, at which time all computations are completed

- **Support for non-standard I/O devices**
  Direct access to switches, displays, ...



- **Non-functional properties**
  fault-tolerance, disposability, EMC-properties, weight, size, user friendliness, extendibility, expected life time, power consumption...

- **Adequate model of computation**



Models of computation

Peter Marwedel
Informatik 12
Univ. Dortmund
Germany

Models of computation
- Definition -

Models of computation define:

- Components and an execution model for computations for each component
- Communication model for exchange of information between components. Asynchronous message passing? *Rendez-vous*?

Communication

- **Shared memory**

Variables accessible to several tasks.

Model is useful only for 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.

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

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

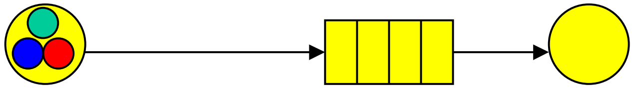
Race-free access to shared memory protected by S possible

This model may be supported by:

- mutual exclusion for critical sections
- cache coherency protocols

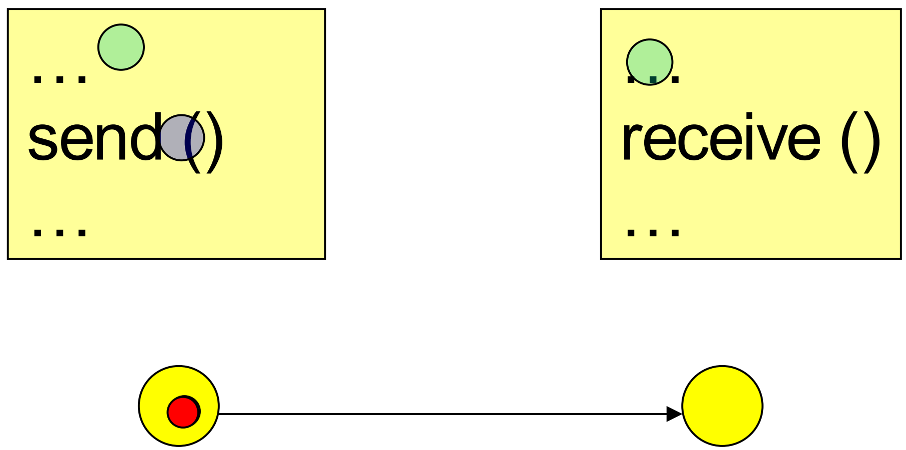
Non-blocking/asynchronous message passing

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



Extended *rendez-vous*

Explicit acknowledge from receiver required. Receiver can do checking before sending acknowledgement.

```
…
send()
…
```

```
…
receive ()
…
ack
…
```

Components (1)

- Von Neumann model

  Sequential execution, program memory etc.

- Discrete event model

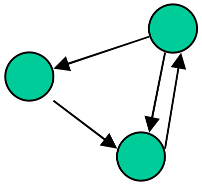
a 6
b 7
c 8

queue

5	10	13	15	19	time
a:=5	b:=7	c:=8	a:=6	a:=9	action

Components (2)

- Finite state machines



- Differential equations

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

Problems with classical CS theory and von Neumann computing

Even the core … notion of "computable" is at odds with the requirements of embedded software. In this notion, useful computation terminates, but termination is undecidable. In embedded software, termination is failure, and yet to get predictable timing, subcomputations must decidably terminate.

Ed Lee: Absolutely Positively on Time, *IEEE Computer*, July, 2005

Problems with thread-based concurrency

*"The lack of timing in the core abstraction is a flaw, from the perspective of embedded software, and **threads as a concurrency model are a poor match for embedded systems**. … they work well only … where best-effort scheduling policies are sufficient.*
What is needed is nearly a reinvention of computer science."

Ed Lee: Absolutely Positively on Time, *IEEE Computer*, July, 2005

☞ Search for non-thread-based, non-von-Neumann MoCs

Ptolemy

Ptolemy (UC Berkeley) is an environment for simulating multiple models of computation.

http://ptolemy.berkeley.edu/

Available examples are restricted to a subset of the supported models of computation.

Newton's craddle

Combined models
- languages presented later in this chapter -

- SDL
  FSM+asynchronous message passing
- StateCharts
  FSM+shared memory
- CSP, ADA
  von Neumann execution+synchronous message passing
- ….

See also

- Work by Ed Lee, UCB
- Axel Jantsch: Modeling Embedded Systems and Soc's: Concurrency and Time in Models of Computation, Morgan-Kaufman, 2004

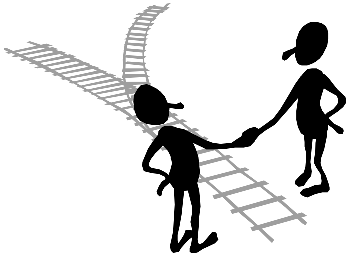
Models considered in this course

<table>
<tr><th>Communication/
local computations</th><th>Shared
memory</th><th colspan="2">Message passing
Synchronous | Asynchronous</th></tr>
<tr><td>Communicating finite state machines</td><td>StateCharts</td><td></td><td>SDL</td></tr>
<tr><td>Data flow model</td><td>Not useful</td><td></td><td>Kahn process networks</td></tr>
<tr><td>Von Neumann model</td><td>C, C++, Java</td><td colspan="2">C, C++, Java with libraries
CSP, ADA |</td></tr>
<tr><td>Discrete event (DE) model</td><td>VHDL, Verilog, SystemC</td><td colspan="2">Only experimental systems, e.g. distributed DE in Ptolemy</td></tr>
</table>

Facing reality

No language that meets all language requirements

☞ using compromises



Summary

Requirements for specification languages

- Hierarchy
- Timing behavior
- State-oriented behavior
- Concurrency
- Synchronization & communication, …

Models of computation

- Models for computation within components
- Models for communication
  - shared memory communication
  - message passing
- Models considered in this course