

(Introduction to) Embedded Systems

Peter Marwedel
TU Dortmund,
Informatik 12



Motivation for Course (1)

According to forecasts, future of IT characterized by terms such as

- Disappearing computer,
- Ubiquitous computing,
- Pervasive computing,
- Ambient intelligence,
- Post-PC era,
- Cyber-physical systems.

Basic technologies:

- *Embedded Systems*
- Communication technologies



Motivation for Course (2)

“Information technology (IT) is on the verge of another revolution.

networked systems of embedded computers ... have the potential to change radically the way people interact with their environment by linking together a range of devices and sensors that will allow information to be collected, shared, and processed in unprecedented ways. ...

The use ... throughout society **could well dwarf previous milestones in the information revolution.**”

*National Research Council Report (US)
Embedded Everywhere, 2001*

Motivation for Course (3)



➡ **The future is embedded,
embedded is the future**

Embedded Systems & Cyber-Physical Systems

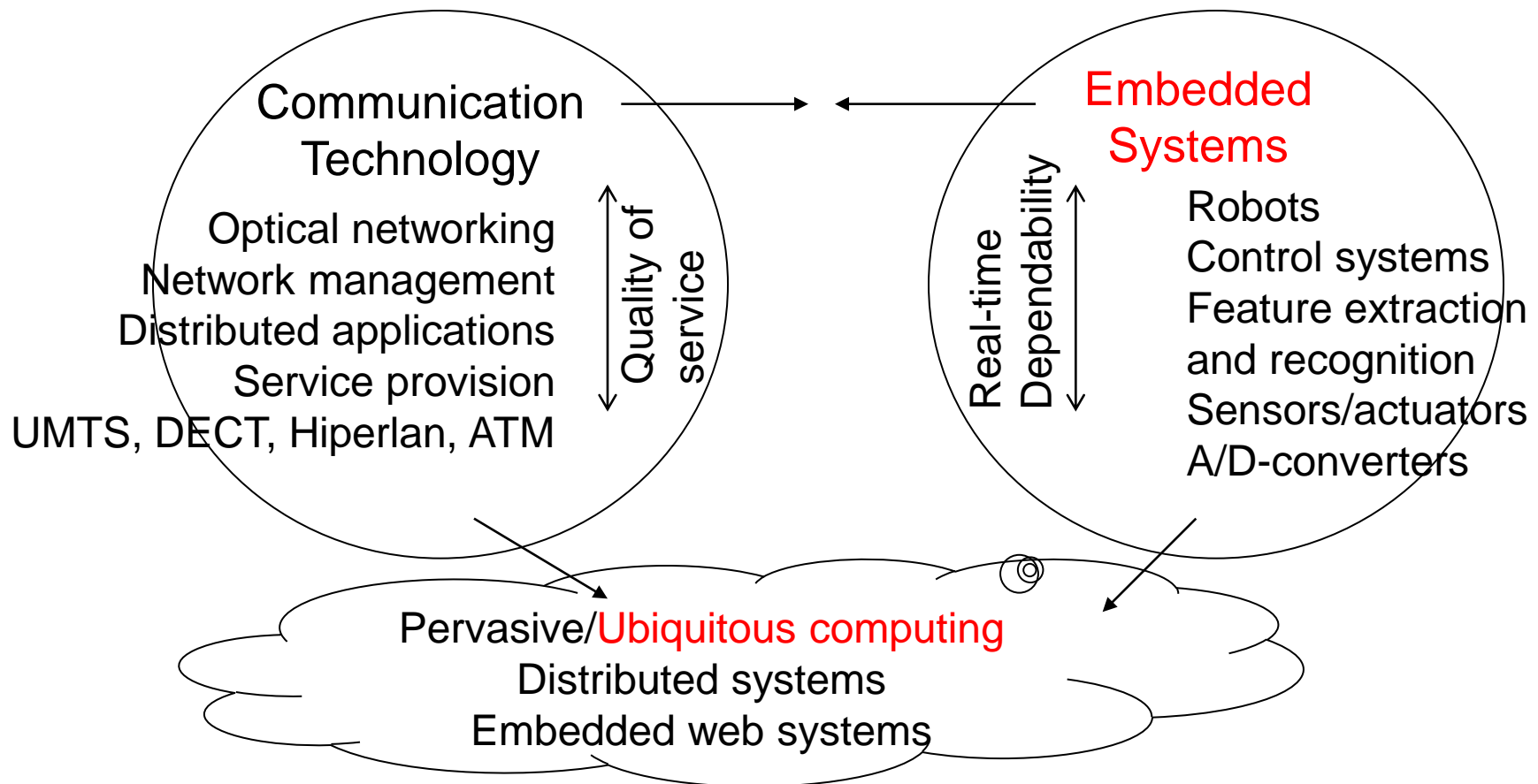
“Dortmund” Definition: [Peter Marwedel]

Embedded systems are information processing systems embedded into a larger product

☞ **Definition: Cyber-Physical (cy-phy) Systems** (CPS) are integrations of computation with physical processes [Edward Lee, 2006].

Extending the motivation: Embedded systems and ubiquitous computing

Ubiquitous computing: Information anytime, anywhere.
Embedded systems provide fundamental technology.



Growing importance of embedded systems (1)



- *the global mobile entertainment industry is now worth some \$32 bln...predicting average revenue growth of 28% for 2010* [www.itfacts.biz, July 8th, 2009]
- *..., the market for **remote home health monitoring** is expected to generate **\$225 mln** revenue in 2011, up from less than **\$70 mln** in 2006, according to Parks Associates. .* [www.itfacts.biz, Sep. 4th, 2007]
- *According to IDC the **identity and access management** (IAM) market in Australia and New Zealand (ANZ) ... is expected to increase at a compound annual growth rate (CAGR) of **13.1%** to reach \$189.3 mln by 2012* [www.itfacts.biz, July 26th, 2008].
- *Accessing the Internet via a mobile device up by **82%** in the US, by **49%** in Europe, from May 2007 to May 2008* [www.itfacts.biz, July 29th, 2008]

Growing importance of embedded systems (2)

- *.. but embedded chips form the backbone of the electronics driven world in which we live ... they are part of almost everything that runs on electricity*
[Ryan, EEDesign, 1995]

Application areas and examples

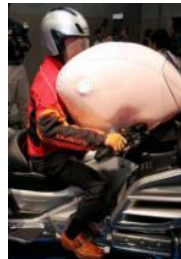


1.1 Application areas and examples

Automotive electronics

Functions by embedded processing:

- ABS: Anti-lock braking systems
- ESP: Electronic stability control
- Airbags
- Efficient automatic gearboxes
- Theft prevention with smart keys
- Blind-angle alert systems
- ... etc ...



Multiple networks

- Body, engine, telematics, media, safety, ...

Multiple networked processors

- Up to 100



© Jakob Engblom

Avionics

- Flight control systems,
- anti-collision systems,
- pilot information systems,
- power supply system,
- flap control system,
- entertainment system,
- ...

Dependability is of outmost importance.



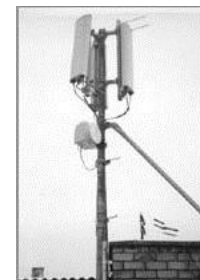
Railways

- Safety features contribute significantly to the total value of trains, and dependability is extremely important



Telecommunication

- Mobile phones have been one of the fastest growing markets in the recent years,
- Geo-positioning systems,
- Fast Internet connections,
- Closed systems for police, ambulances, rescue staff.



Medical systems

- For example:
 - Artificial eye: several approaches, e.g.:
 - Camera attached to glasses; computer worn at belt; output directly connected to the brain, “pioneering work by William Dobelle”. Previously at [www.dobelle.com]
- Translation into sound; claiming much better resolution.
[<http://www.seeingwithsound.com/etumble.htm>]



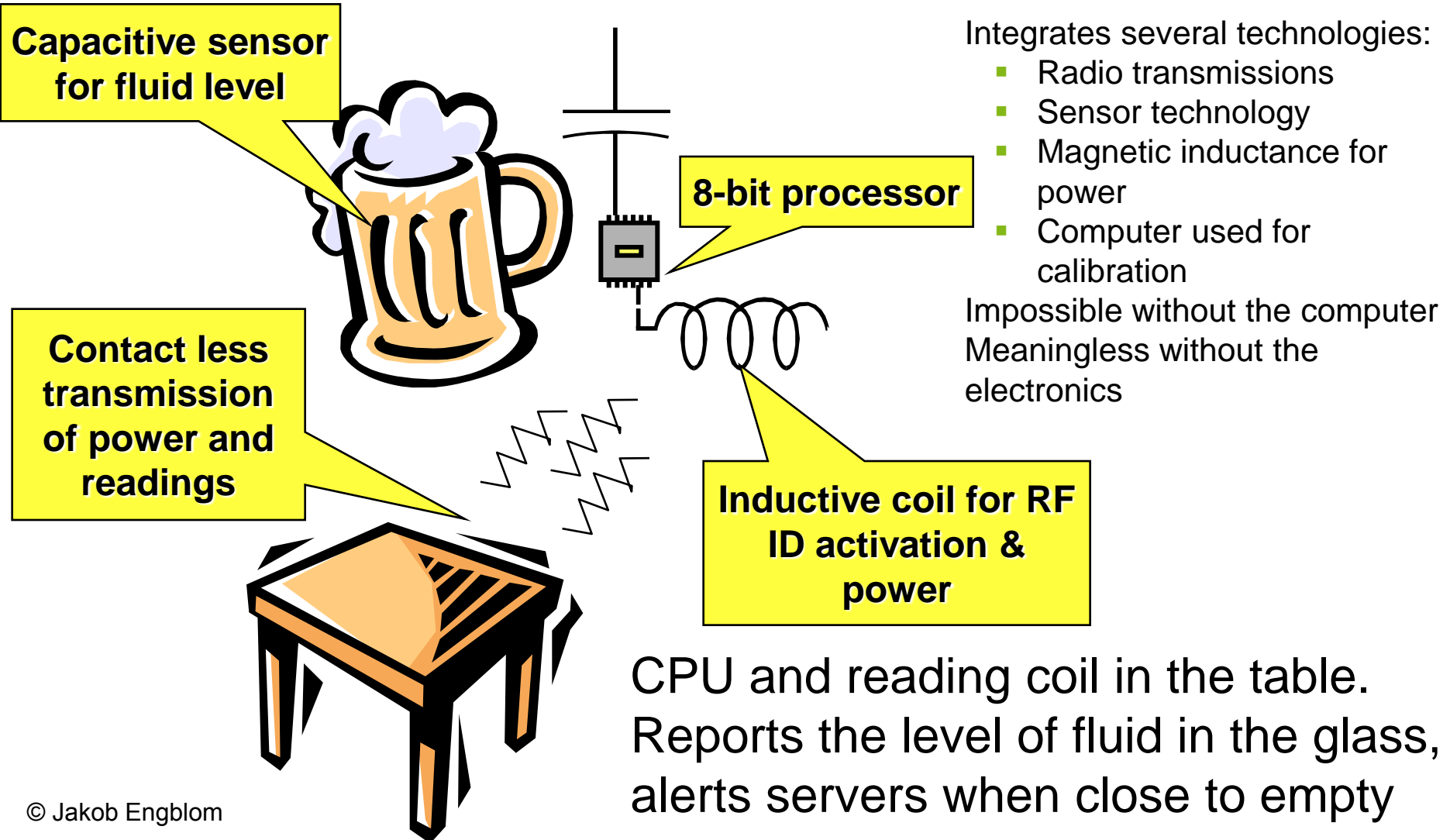
Authentication systems

- Finger print sensors
- Access control
- Airport security systems
- Smartpen®
- Smart cards
-



[tomsguide.com]

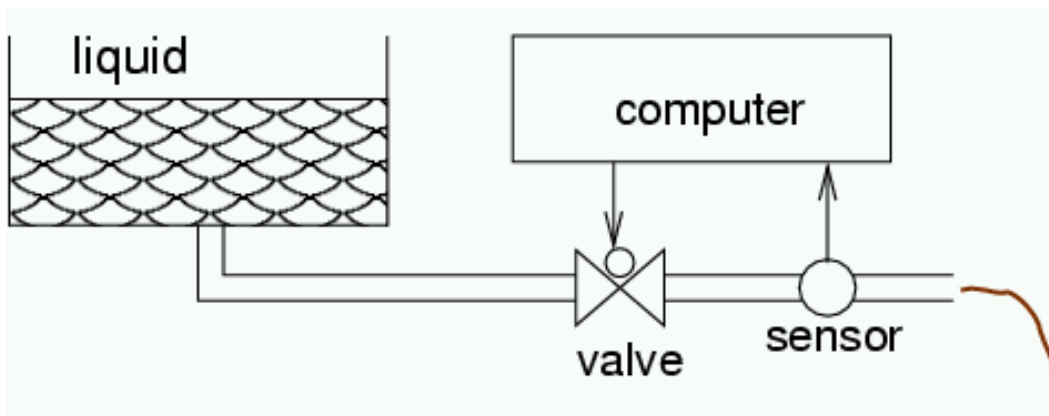
Smart Beer Glass



© Jakob Engblom

Industrial automation

Examples



Forestry Machines

Networked computer system



- Controlling arms & tools
- Navigating the forest
- Recording the trees harvested
- Crucial to efficient work

“Tough enough to be out in the woods”

Smart buildings

Examples

- Integrated cooling, lightning, room reservation, emergency handling, communication
- Goal: “Zero-energy building”
- Expected contribution to fight against global warming



Logistics

Applications of embedded/cyber-physical system

technology to logistics:

- Radio frequency identification (RFID) technology provides easy identification of each and every object, worldwide.
- Mobile communication allows unprecedented interaction.
- The need of meeting real-time constraints and scheduling are linking embedded systems and logistics.
- The same is true of energy minimization issues

Educational concept



From the preface of the book

Broad scope avoids problems with narrow perspectives reported in ARTIST curriculum guidelines

“The lack of maturity of the domain results in a large variety of industrial practices, often due to cultural habits”

“curricula ... concentrate on one technique and do not present a sufficiently wide perspective.”

“As a result, industry has difficulty finding adequately trained engineers, fully aware of design choices.”

Source: ARTIST network of excellence:

Guidelines for a Graduate Curriculum on Embedded Software and Systems,
<http://www.artist-embedded.org/Education/Education.pdf>, 2003

Scope consistent with ARTIST guidelines

"The development of ES cannot ignore the underlying HW characteristics.

Timing, memory usage, power consumption, and physical failures are important."

"It seems that fundamental bases are really difficult to acquire during continuous training if they haven't been initially learned, and we must focus on them."



Textbook(s)

- Peter Marwedel, “Embedded System Design”, Kluwer Academic Publishers, 2003.
- Peter Marwedel, “Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems”, Springer, 2011.
- Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, Gunar Schirner, “Embedded System Design: Modeling, Synthesis and Verification”, Springer 2009.

Slides, References, ...



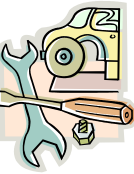


- Slides are available at:
<http://www2.itu.edu.tr/~orssi/>

Common characteristics



1.2 Common characteristics

Dependability

- ES Must be **dependable**, 
 - **Reliability** $R(t)$ = probability of system working correctly provided that it was working at $t=0$ 
 - **Maintainability** $M(d)$ = probability of system working correctly d time units after error occurred. 
 - **Availability** $A(t)$: probability of system working at time t
 - **Safety**: no harm to be caused 
 - **Security**: confidential and authentic communication 

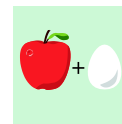
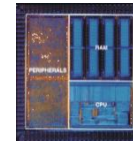
Even perfectly designed systems can fail if the assumptions about the workload and possible errors turn out to be wrong.

Making the system dependable must not be an after-thought, it must be considered from the very beginning

Efficiency

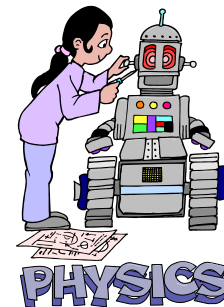
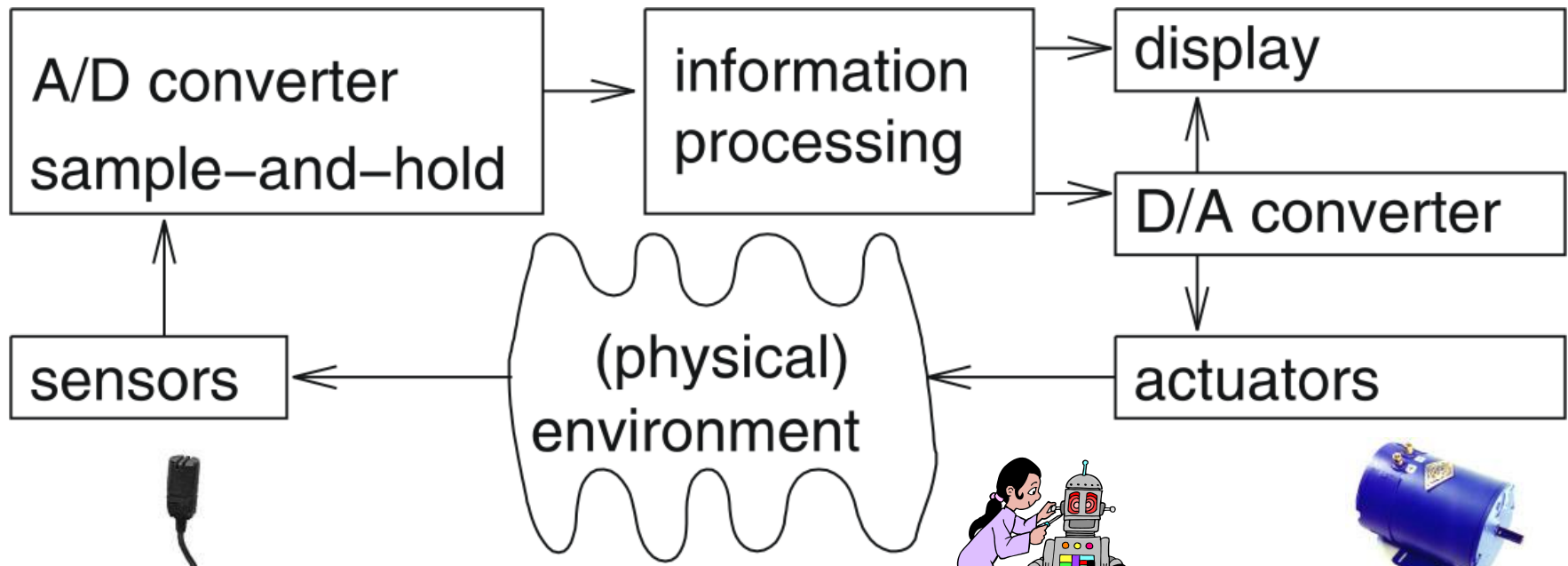
- ES must be **efficient**

- Code-size efficient
(especially for systems on a chip)
- Run-time efficient
- Weight efficient
- Cost efficient
- Energy efficient



Embedded System Hardware

Embedded system hardware is frequently used in a loop (*“hardware in a loop”*):



👉 cyber-physical systems

Real-time constraints

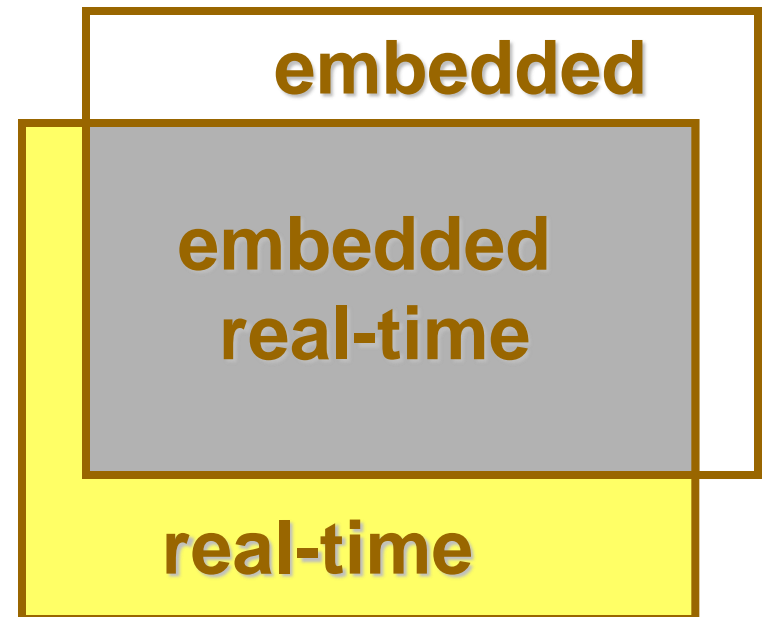
- Many ES must meet **real-time constraints**
 - A real-time system must react to stimuli from the controlled object (or the operator) within the time interval **dictated** by the environment.
 - For real-time systems, right answers arriving too late are wrong.
 - “**A real-time constraint is called hard, if not meeting that constraint could result in a catastrophe**“ [Kopetz, 1997].
 - All other time-constraints are called **soft**.
 - A guaranteed system response has to be explained without statistical arguments



Real-Time Systems

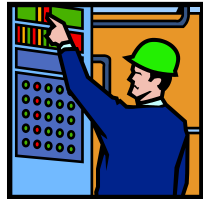
Embedded and Real-Time Synonymous?

- Most embedded systems are real-time
- Most real-time systems are embedded



Reactive & hybrid systems

- Typically, ES are **reactive systems**:
“A reactive system is one which is in continual interaction with its environment and executes at a pace determined by that environment”
[Bergé, 1995]
Behavior depends on input **and current state**.
- **Hybrid systems**
(analog + digital parts).



Dedicated systems

- **Dedicated** towards a certain **application**
Knowledge about behavior at design time can be used to minimize resources and to maximize robustness
- **Dedicated user interface**
(no mouse, keyboard and screen)



Challenges in ES Design

Peter Marwedel
TU Dortmund,
Informatik 12

2010/09/20



These slides use Microsoft clip arts.
Microsoft copyright restrictions apply.

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)

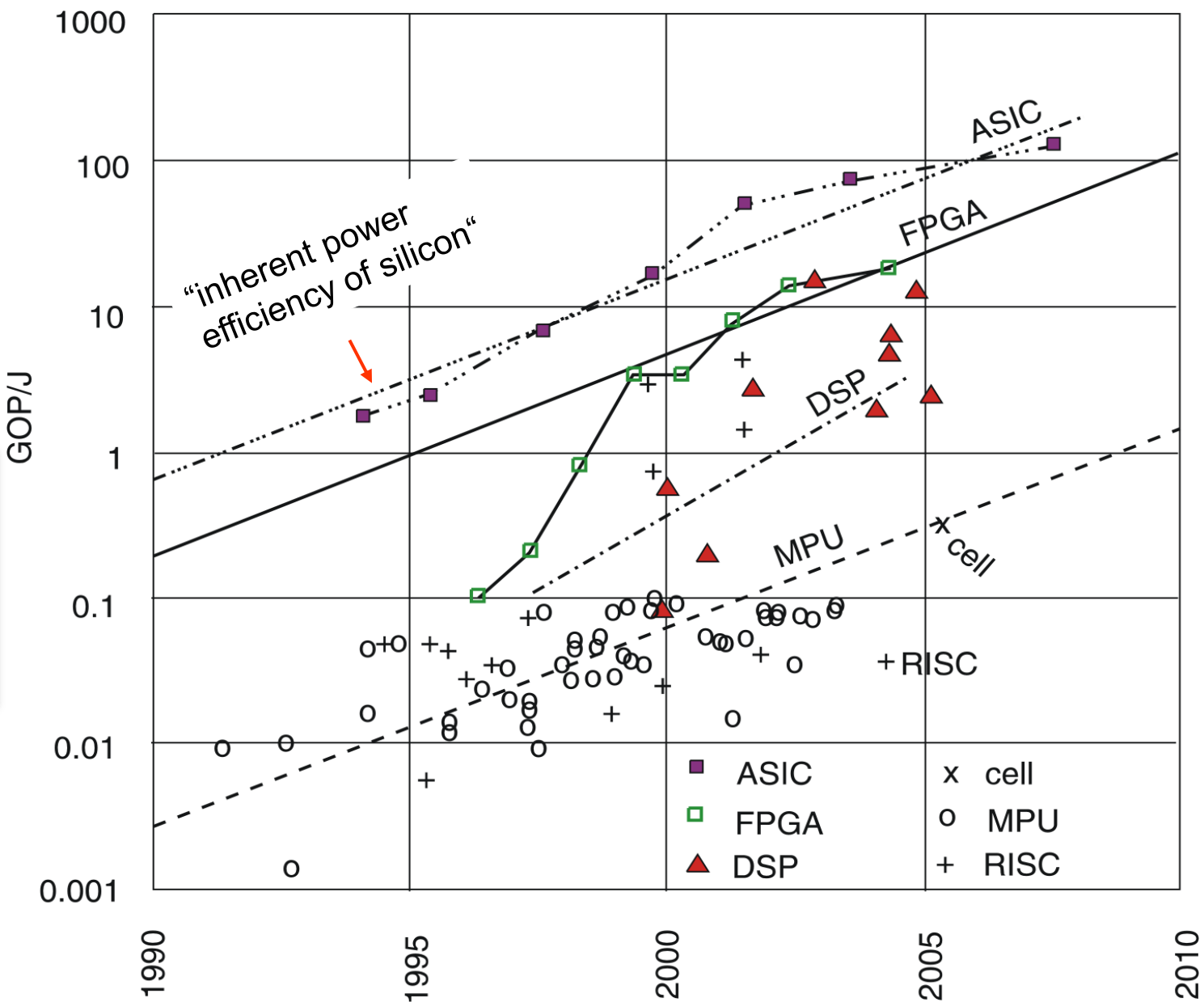


- Using unsafe systems for safety-critical missions (e.g. voice control system in Los Angeles; ~ 800 planes without voice connection to tower for > 3 hrs)



Importance of Energy Efficiency

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



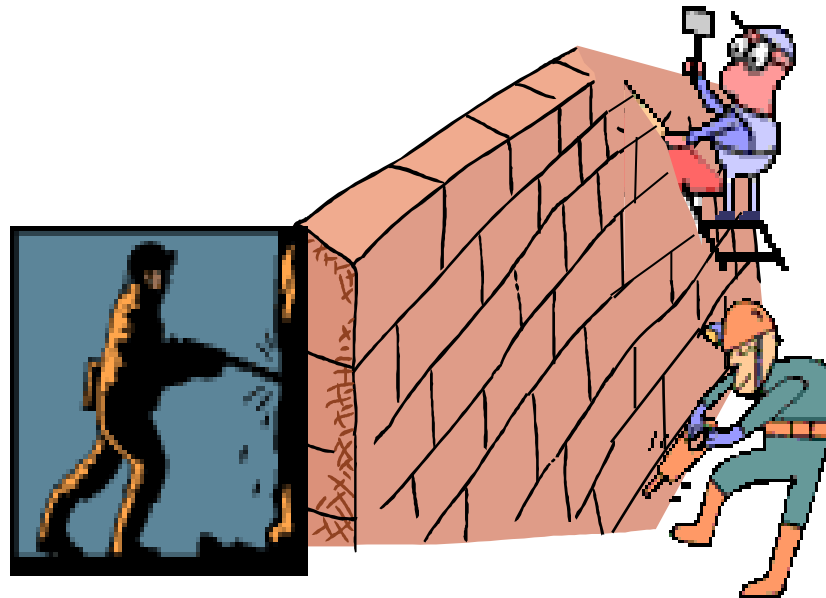
© Hugo De Man, IMEC, Philips, 2007

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



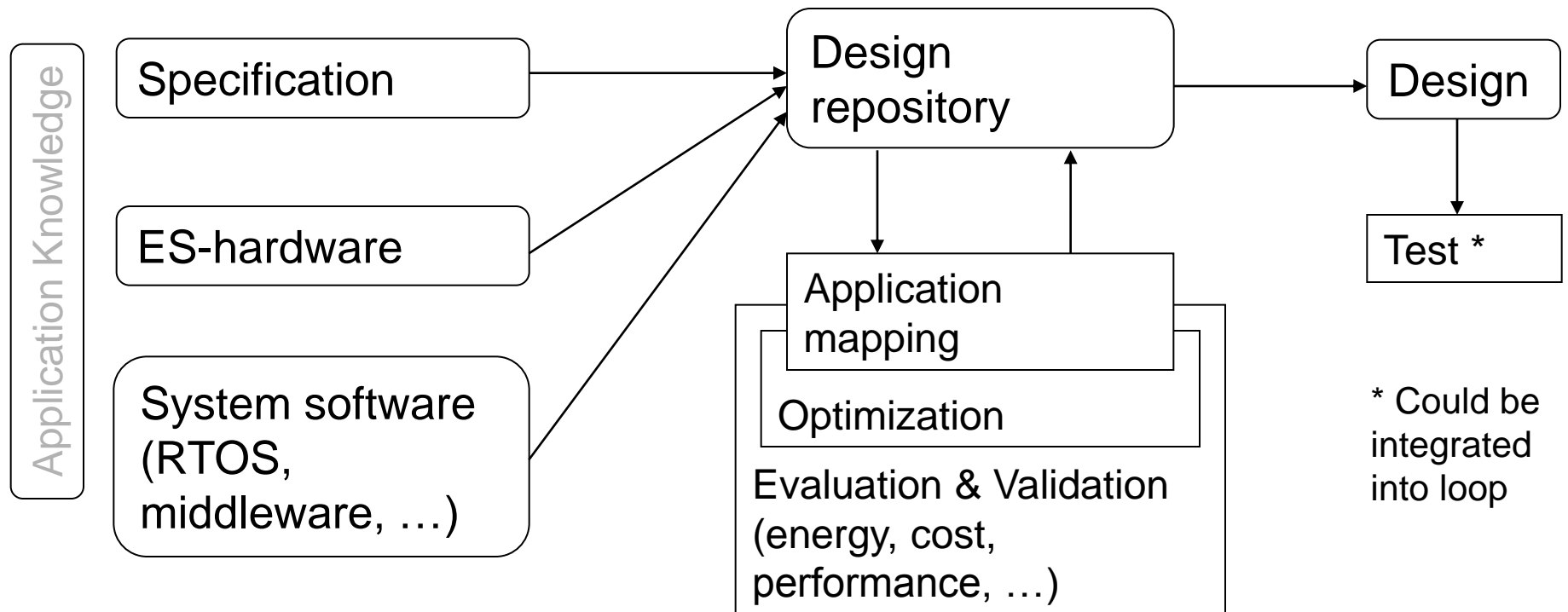
The same for walls to other disciplines and more challenges

Design flows



1.3 Design flows

Hypothetical design flow



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

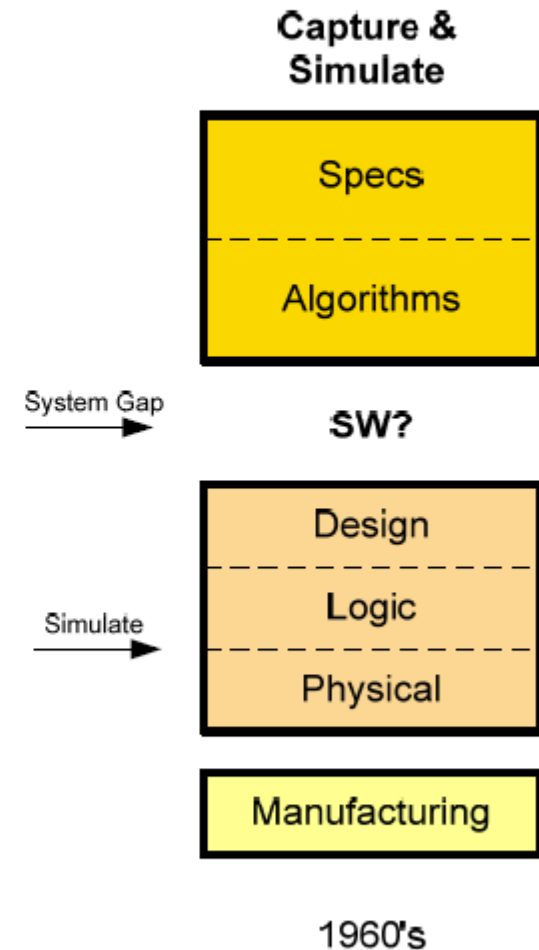
Evolution of Design Flow

- **Three evolutionary design flows**
 - Capture and Simulate
 - Designers complete design and validate through simulation
 - Describe and Synthesize
 - Introduction of design synthesis
 - For given functionality, design structure generated by tools
 - Specify-Explore-Refine
 - System design flow extended to four levels
 - Large number of levels and metrics for validation
 - Design flow performed in several steps
 - Each step follows describe-and-synthesize concept



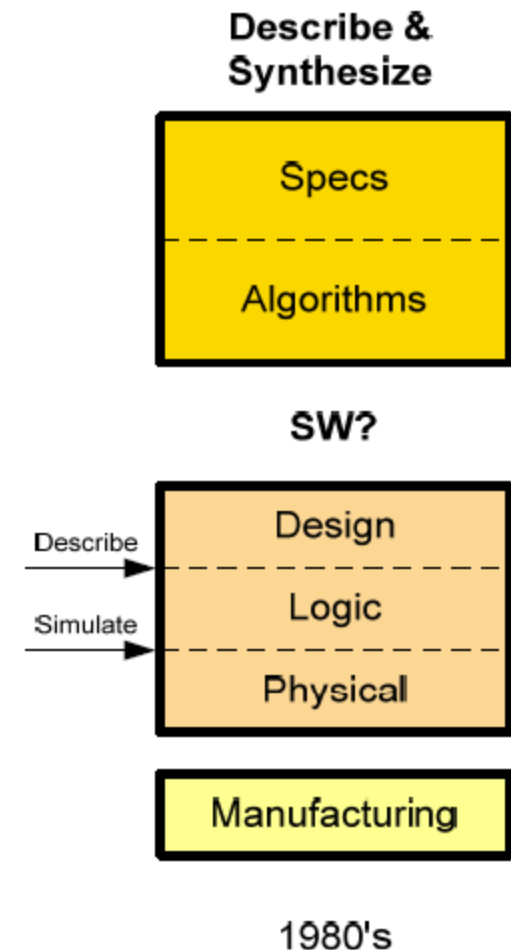
Capture-and-Simulate Design Flow

- **System designers generate**
 - Preliminary specification
 - Basic algorithms
 - No software design
- **HW designers generate**
 - Architecture, RTL, Logic
 - Logic-level netlist for simulation
 - Simulate and optimize
- **System gap between SW and HW**
- **Simulation at the end of design**
- **Manual design, no automation**



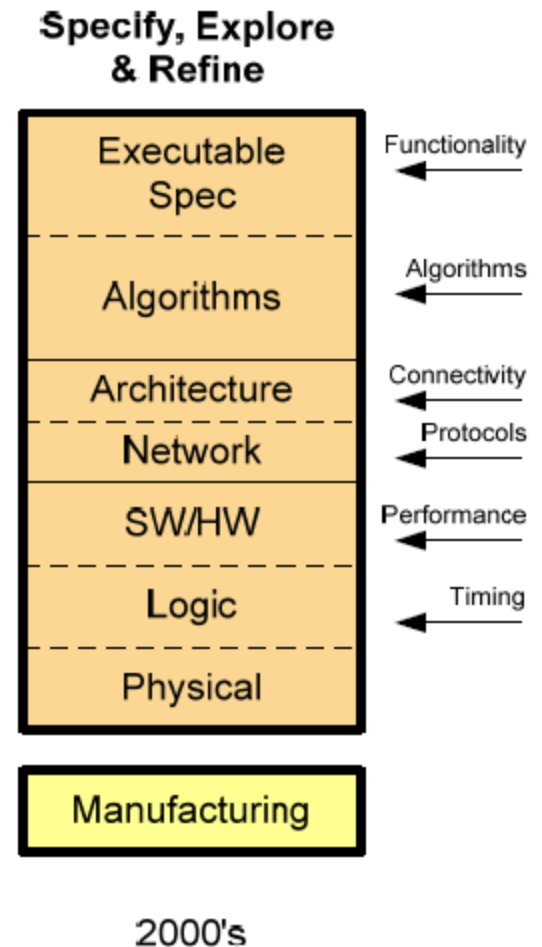
Describe-and-Synthesize Design Flow

- **System designers generate**
 - Preliminary specification
 - Basic algorithms
 - SW design after HW design
- **HW designers generate**
 - Architecture, RTL, Logic behavior
- **Logic synthesis tools generate logic**
 - Designers simulate and optimize
- **Simulation before and after synthesis**
- **Designers describe just functionality, tools synthesize structure**
- **System gap still persists**



Specify-Explore-Refine Design Flow

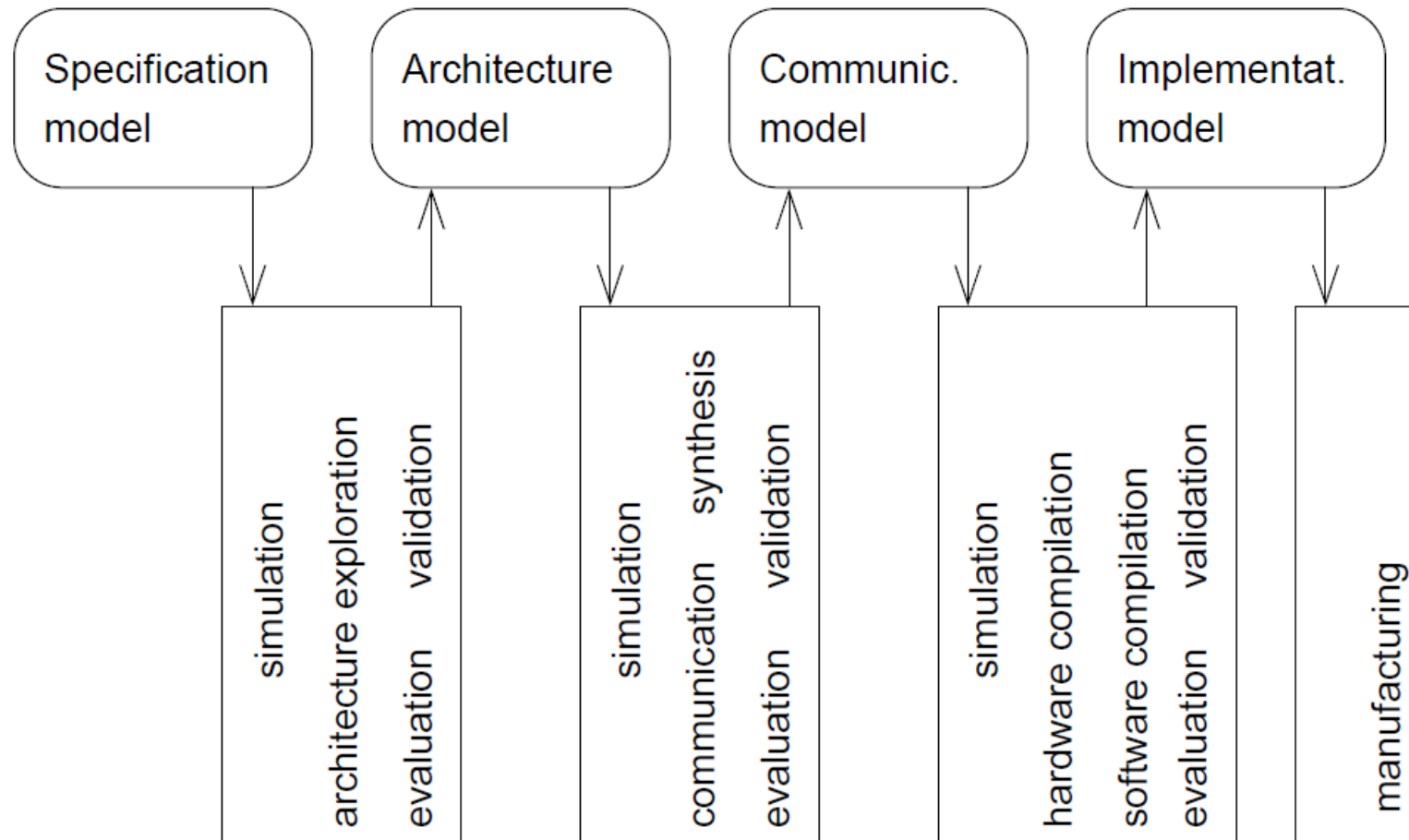
- **Application designers generate**
 - Executable specification
 - Application algorithms
 - Platform model for application/platform optimization
- **System designers generate**
 - Detailed architecture and network
 - SW and HW components
 - Logic and layout
- **Many design levels and models**
- **Many metrics for validation**
- **Solution: Step-by-step strategy**
 - For each model explore design decisions
 - Refine model after exploration
 - Refined model = specification for the next level



Iterative design (1)

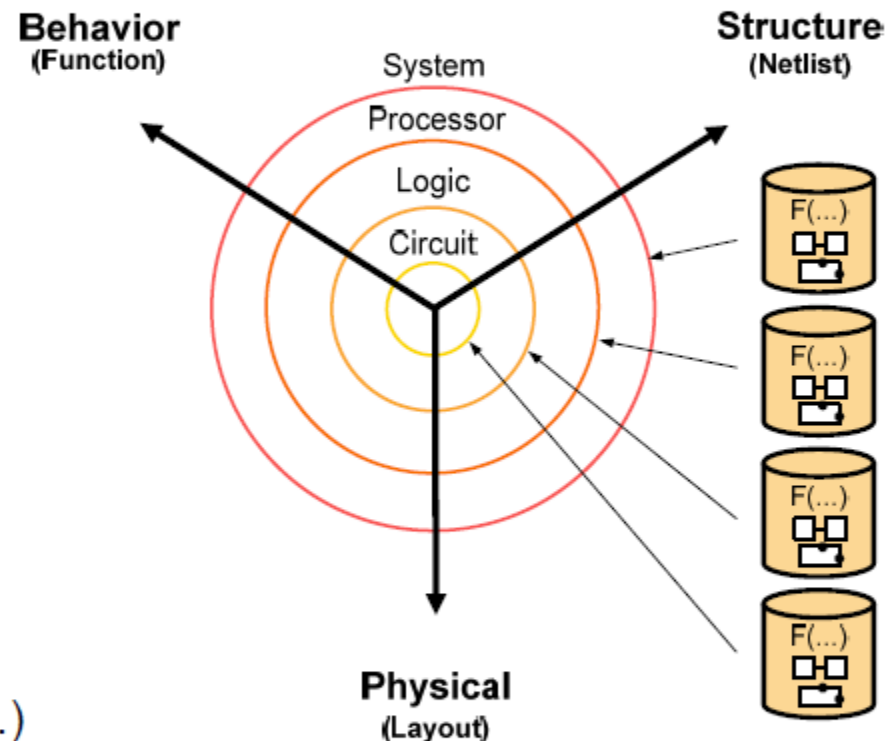
- After unrolling loop -

Example:
SpecC
tools



Y Chart

- **3 design views**
 - Behavior (functionality)
 - Structure (netlist)
 - Physical (layout)
- **4 abstraction levels**
 - Circuit
 - Logic
 - Processor
 - System
- **5 component libraries**
 - Transistor
 - Logic (standard cells)
 - RTL (ALUs, RF, Memories,...)
 - Processor (standard, custom)
 - System (multi-cores with NoCs)



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?



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

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?

Models and Architectures

- The system = collection of simpler subsystems or pieces
- There are different methods of decomposing functionality
- A particular method = a model

Model Taxonomy

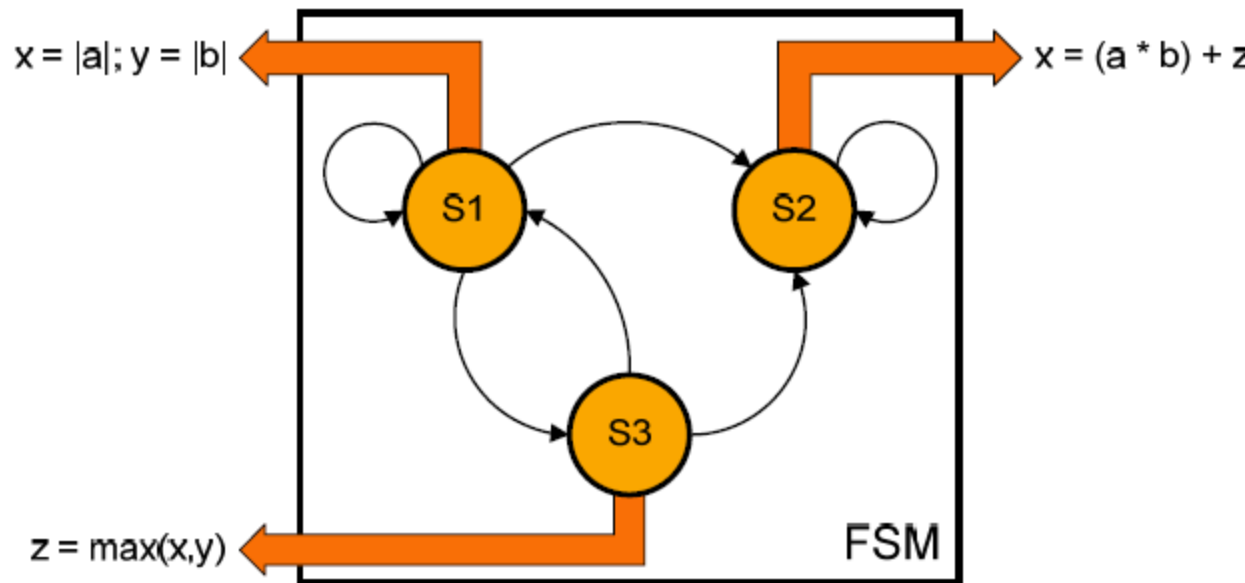
1. **State Oriented** (state diagram)
for control systems \Rightarrow temporal behavior
2. **Activity Oriented** (dataflow graph)
for transformational systems like digital signal processing
3. **Structure Oriented** (block diagram)
4. **Data Oriented** (entity relationship diagram)
for information systems like databases
3. **Heterogeneous**

Processor Abstraction Level

- **Processor level generates system components**
 - Computation components
 - Standard processors
 - Custom processor
 - Custom functions
 - Controllers
 - Memories
 - Communication components
 - Arbiters
 - Bridges and transducers
 - Controllers (interrupt, memory, DMA,)
 - Interfaces
- **Processor-level synthesis**
 - Behavior specification (FSMD, CDFG, IS)
 - Component structure
 - Synthesis on processor level



FSMD Model



- **FSM**

- Set of states and transitions
- Transition executed under conditional statements of input variables

- **FSMD**

- FSM + Set of variable statements executed in each state

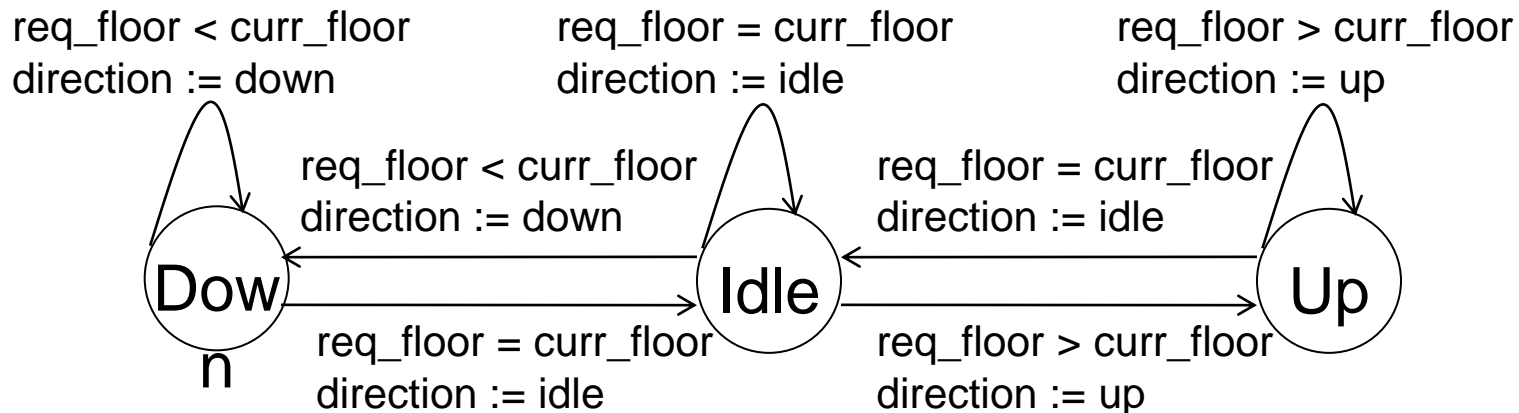
- **No timing, could be assumed**



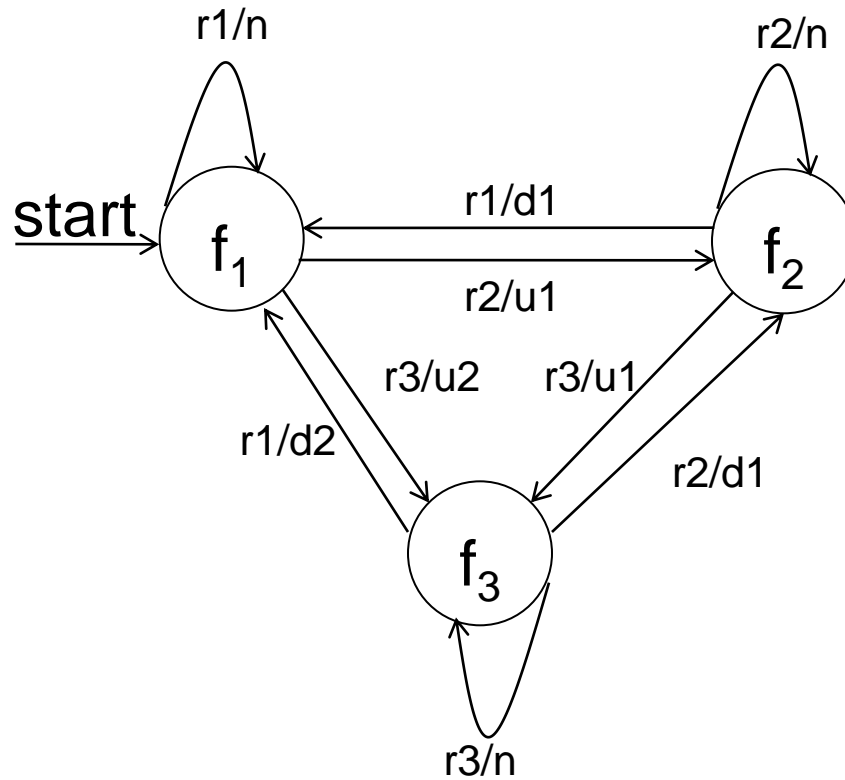
Views of an Elevator Controller

- If the elevator is stationary and the floor requested is equal to the current floor, then the elevator remains idle.
- If the elevator is stationary and the floor requested is less than the current floor, then lower the elevator to the requested floor.
- If the elevator is stationary and the floor requested is greater than the current floor, then raise the elevator to the requested floor.

```
loop
  if (req_floor = curr_floor) then
    direction := idle;
  elsif (req_floor < curr_floor) then
    direction := down;
  elsif (req_floor > curr_floor) then
    direction := up;
  end if;
end loop;
```



Finite State Machine



Finite State Machine

States

A set of transitions

A set of actions

$$\langle S, I, O, f: S \times I \rightarrow S, h: S \times I \rightarrow O \rangle$$

Set of states : $S = \{s_1, s_2, \dots, s_n\}$

Set of inputs : $I = \{i_1, i_2, \dots, i_m\}$

Set of outputs : $O = \{o_1, o_2, \dots, o_k\}$

Next state function : f

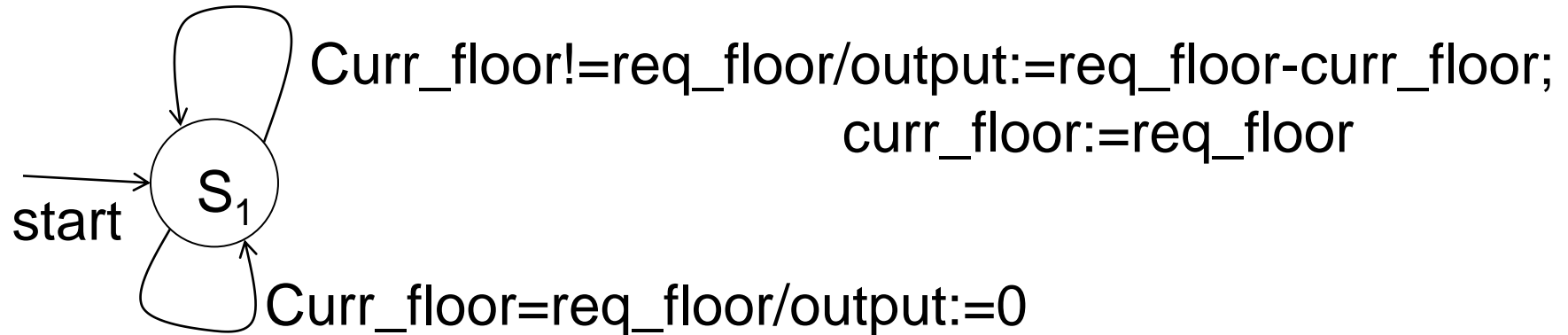
Output function : h

Mealy and Moore Machines

Transition based (Mealy) : $h : S \times I \rightarrow O$

State based (Moore) : $h : S \rightarrow O$

FSM with Data Path



- $\langle S, I \cup STAT, O \cup A, f, h \rangle$
- A set of storage variables : VAR
- A set of expressions : $EXP = \{ f(x, y, z, \dots) \mid x, y, z, \dots \in VAR \}$
- A set of storage assignments : $A = \{ X \leftarrow e \mid X \in VAR, e \in EXP \}$
- A set of status expressions : $STAT = \{ Rel(a, b) \mid a, b \in EXP \}$
- $f : S \times (I \cup STAT) \rightarrow S$
- $h : S \times (I \cup STAT) \rightarrow (O \cup A)$

Problems with FSM and FSMD models

- Neither the FSM nor the FSMD model is suitable for complex systems
- Neither one supports concurrency and hierarchy

Requirements for specification techniques: Hierarchy

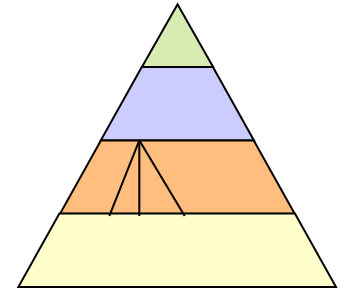
Hierarchy

Humans not capable to understand systems containing more than ~5 objects.

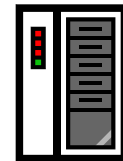
Most actual systems require more objects

☞ Hierarchy

- Behavioral hierarchy
Examples: states, processes, procedures.
- Structural hierarchy
Examples: processors, racks, printed circuit boards



proc
proc
proc



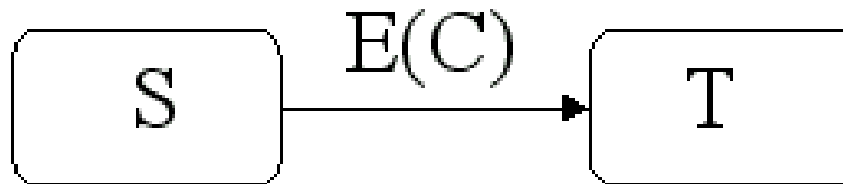
Hierarchical Concurrent Finite State Machine (HCFSM)

- Each set can be decomposed into a set of substates : hierarchy
- Each set can be decomposed into a set of concurrent substates : execute in parallel
- Language for HCFSM : Statecharts

Statechart Notation

Transitions

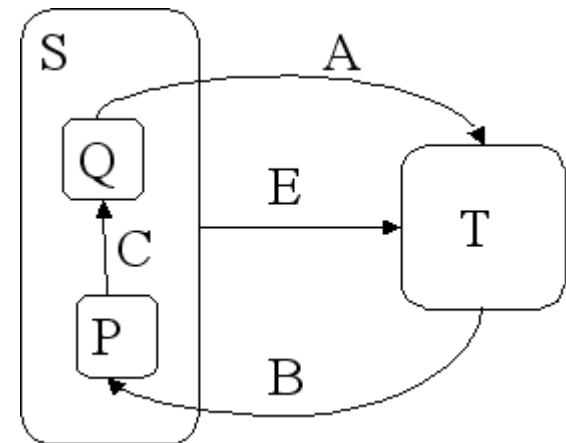
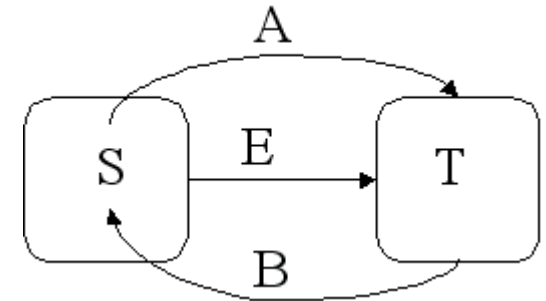
- If event E occurs in state S and condition C holds then make the transition to state T .



- Actions to be carried out when event E occurs in state S are normally put into an event-action table rather than part of the diagram. The actions could be lengthy and detailed.

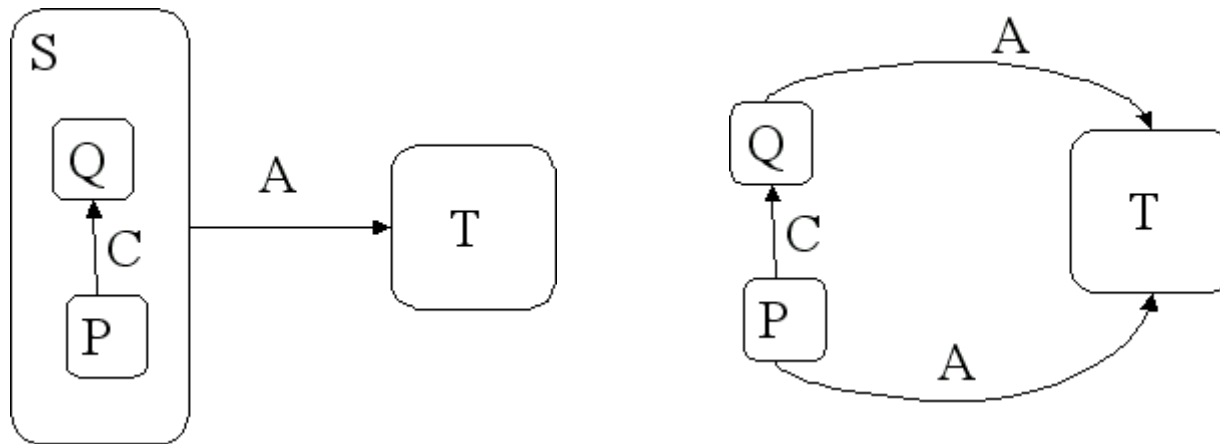
Hierarchical statecharts

- FSMs are flat. They provide no facility to represent hierarchies.
- Details sometimes can to be hidden to portray higher level abstraction.
- Because the arrows go inside node S, events A and B indicate that there is more detail in state S.
 - When in state S it is also in either state P or Q.
 - When in state S, move from P to Q on event C
 - When in state T move to state S and within S to P on event B.
 - Regardless of states P or Q, on event E, move to T
 - Move to T from Q and S on event A



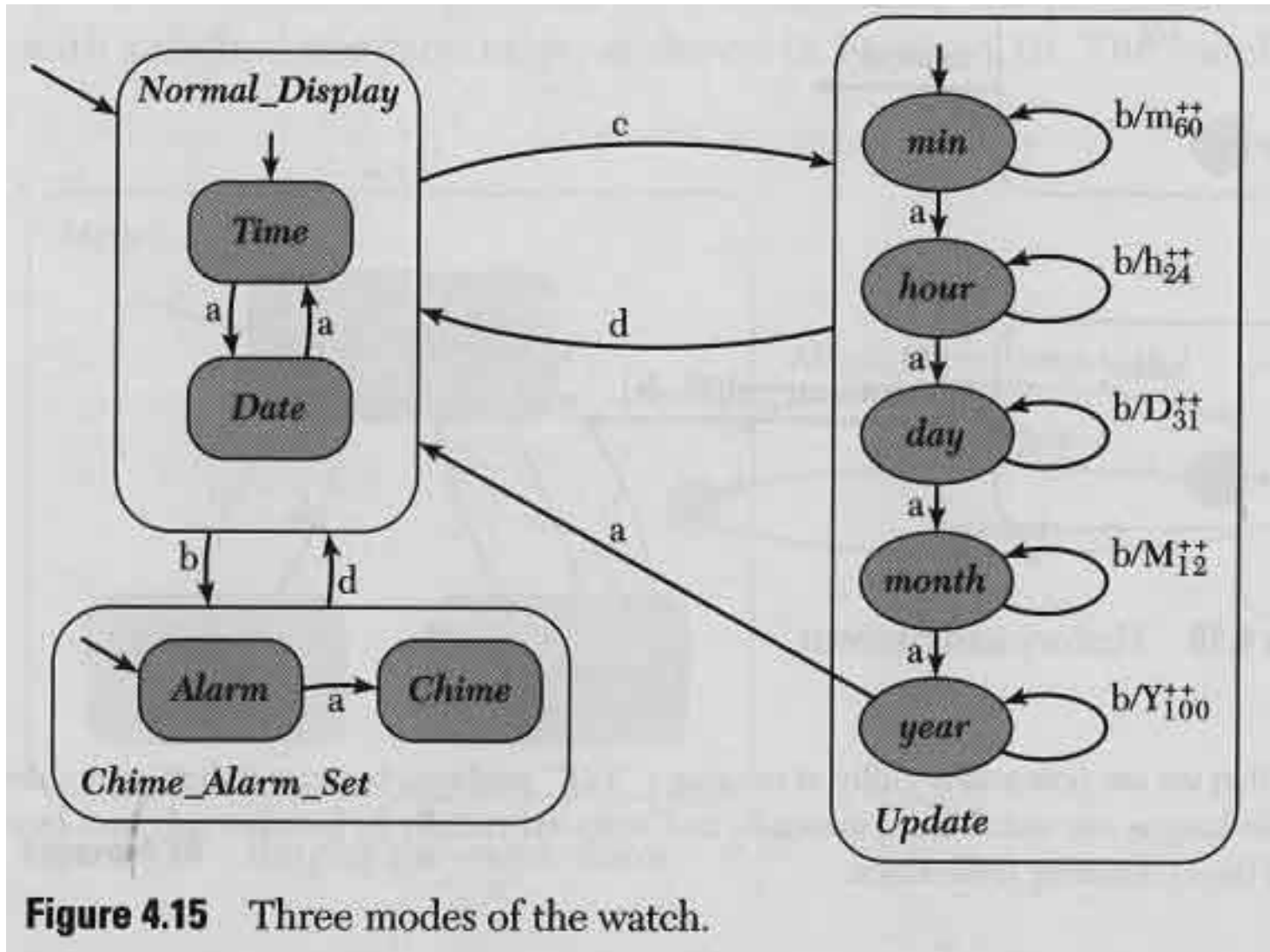
Clustering

- Clustering is another form of hierarchy.



- The advantage is the reduction in the number of arrows.
- Likely states *P* and *Q* are abstractly different from *T*.

Three modes of a digital watch

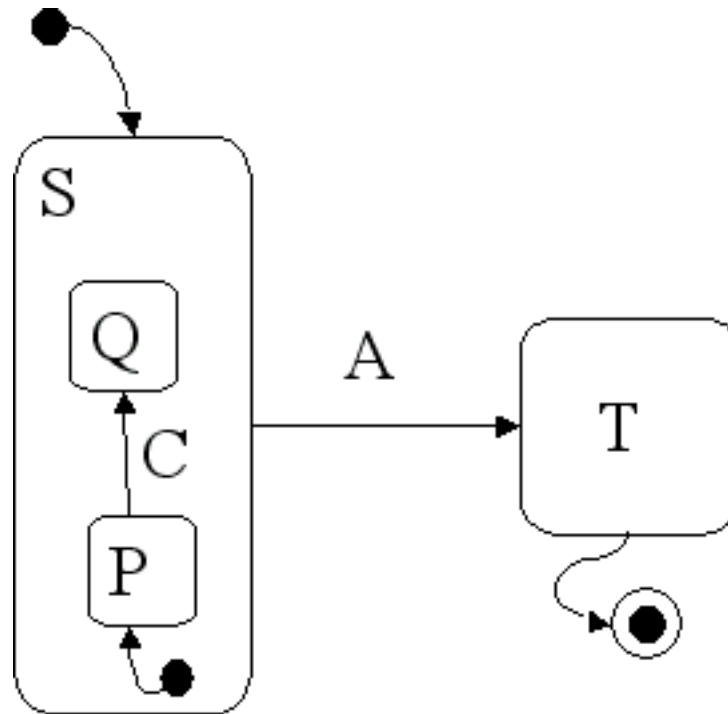


Start and Stop States

Start states are the simple large dots.

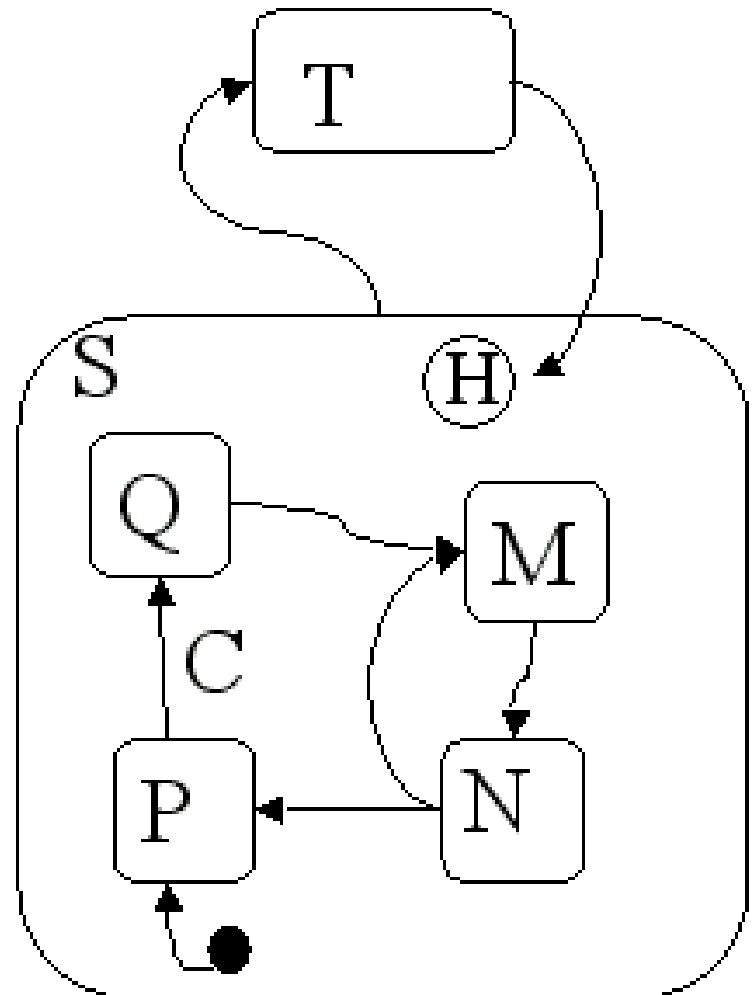
Stop states are the encircled dots.

A nested statechart should have a start point indicated and a stop state if it's not obvious.



History mechanism

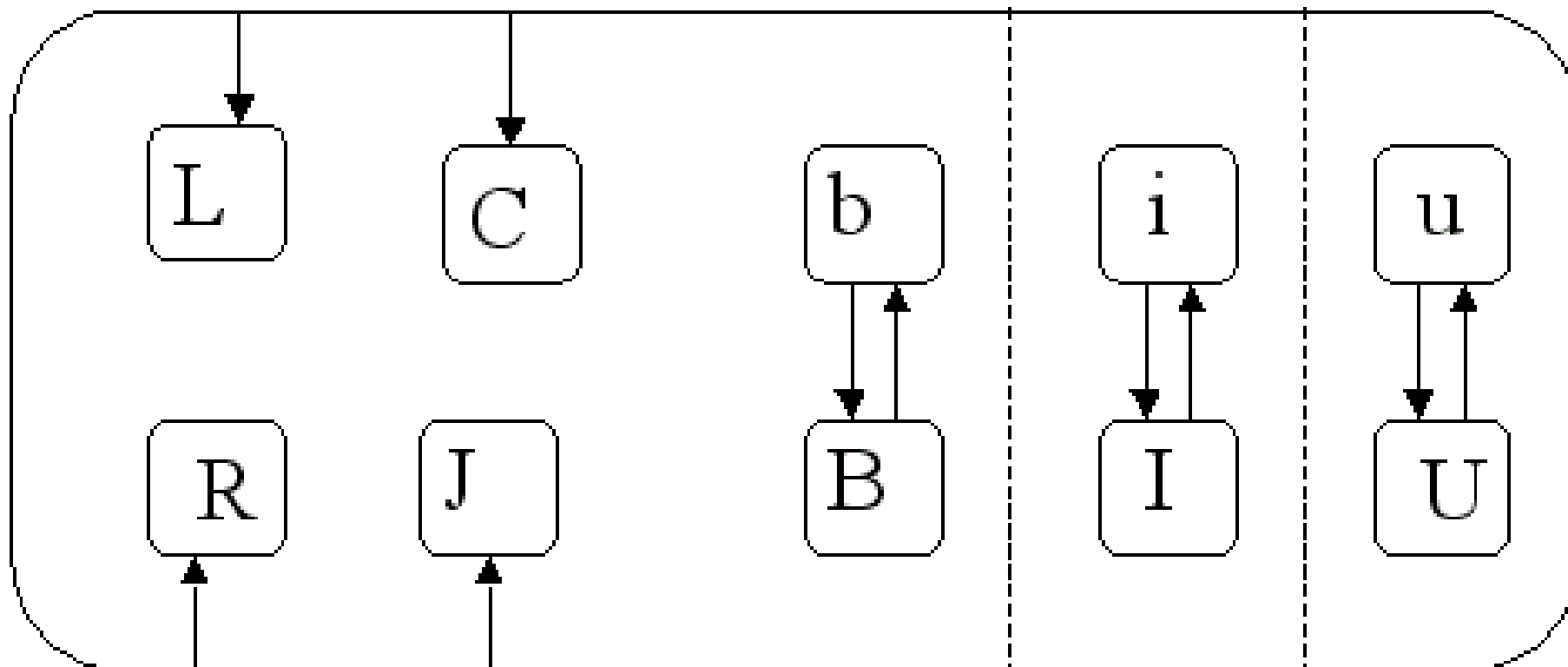
- ◆ When entering a state that has a nested statechart, you may want to resume where you left off when you left the enclosing state. The encircled H as the entry point as below means to start in the state within that was exited.
- ◆ Each level can have its own history mechanism. Each history variable would be initialized to the start state of the level.
- ◆ The encircled H can have an asterisk attached to indicate that the history mechanism is to be used at all levels of the hierarchy.



Concurrency

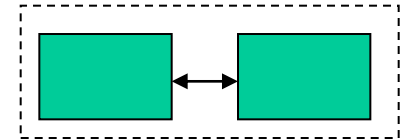
- ◆ Concurrency refers to the ability to manage more than one state simultaneously.
- ◆ If a simple FSM is used, the number of states needed is the product of the number of states that can be represented in each group separately.
- ◆ **Example:** Consider the 3 style types -- bold, italics and underline. These styles can be used in any combination. Bold is on or off, italics is on or off, underline is on or off. The total number of states is $2*2*2 = 8$ and the number of transitions is nearly as bad.
- ◆ Instead, keep each style type as a separate statechart (bold on or off), (italics on or off) and (underline on or off). Each statechart is a simple pair of states.
- ◆ **Example:** Consider the 4 paragraph positions -- left, center, right and justified. These types are mutually exclusive but the number of transition arcs is 10 ($=4*5/2$).

Concurrency



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

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

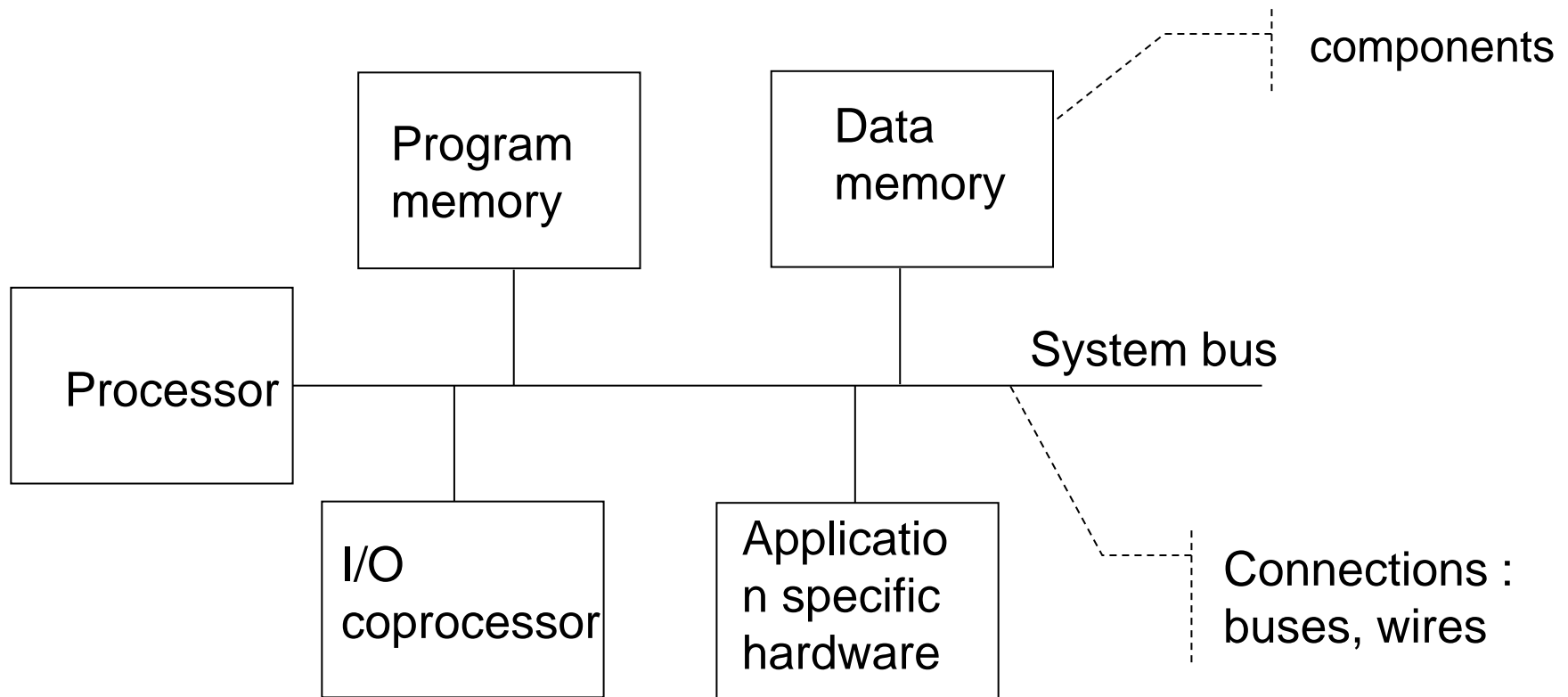


👉 Work of Sifakis, Thiele, Ernst, ...

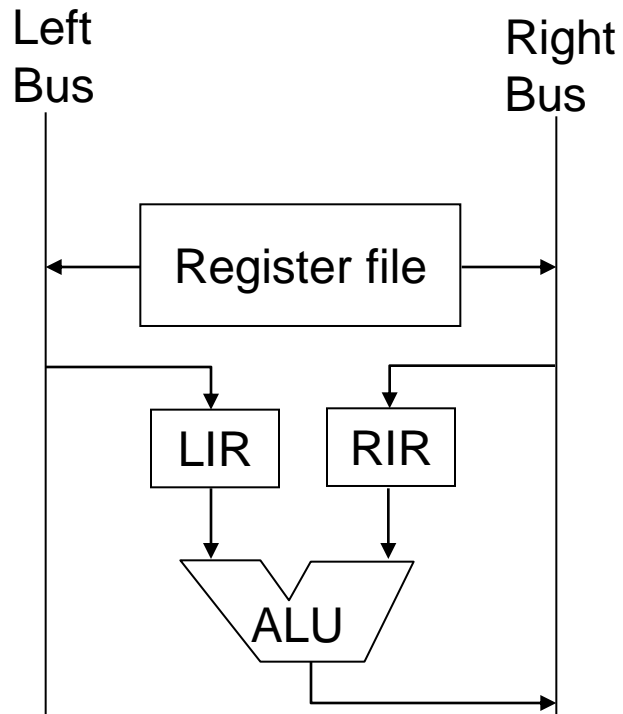
- Concurrency
- Synchronization and communication

Structure oriented models : Component-connectivity diagram

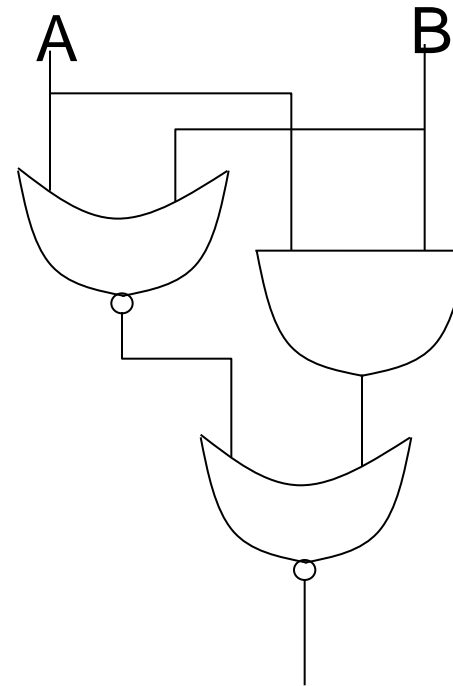
- Models systems structural view
- Often used in the later phases of the design process



System block diagram



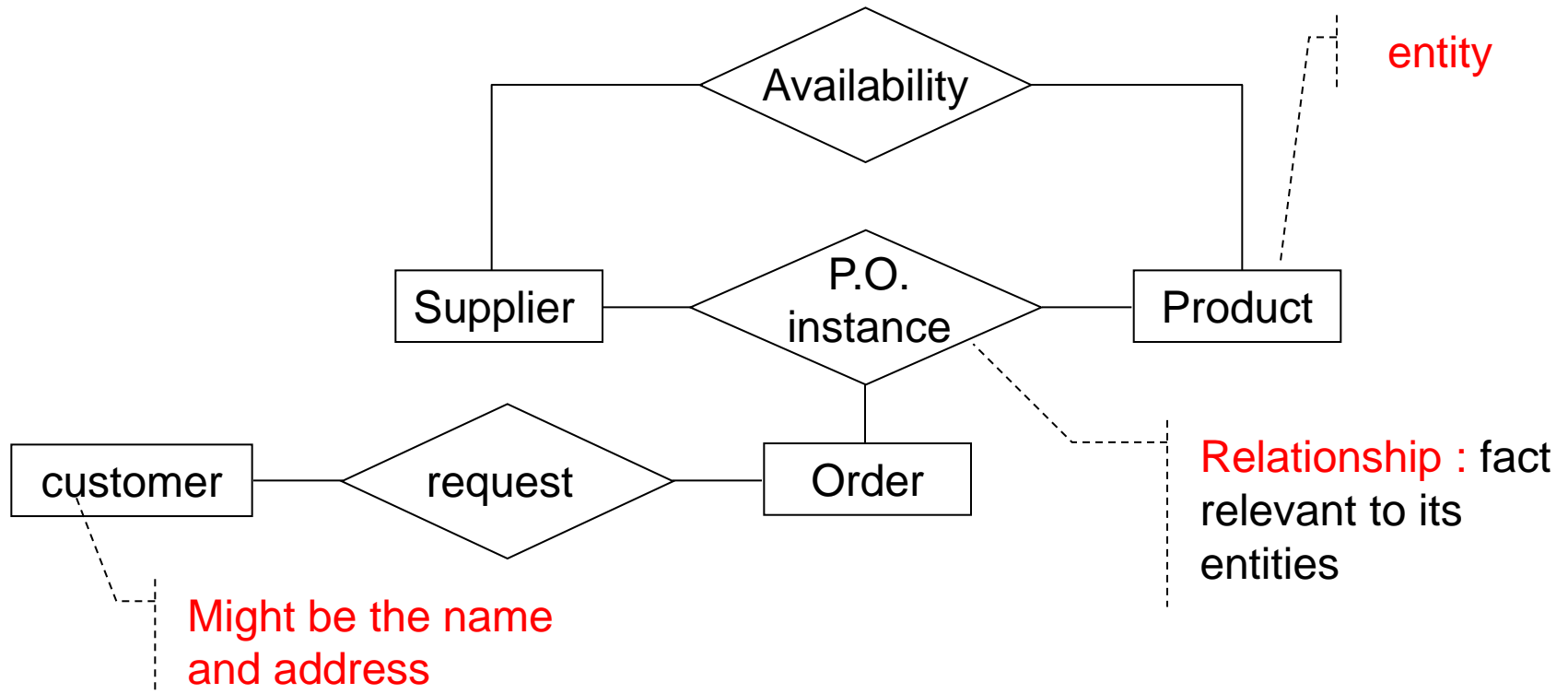
RT-level schematic



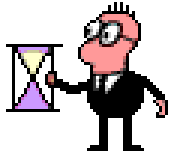
Gate-level schematic

Data-Oriented Models: Entity relationship diagram

Generally used in the design of information systems



Requirements for specification techniques (3): Timing



■ Timing behavior

Essential for embedded 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!**

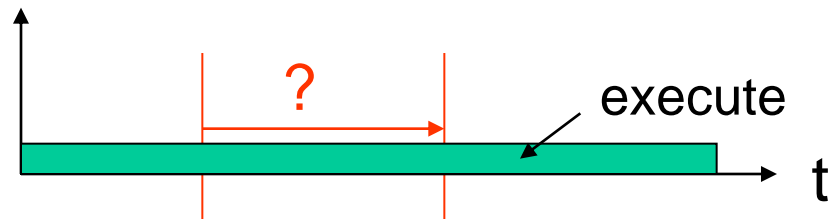
“The lack of timing in the core abstraction (of computer science) is a flaw, from the perspective of embedded software” [Lee, 2005]

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

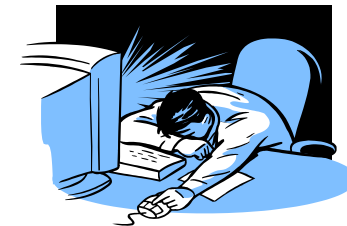
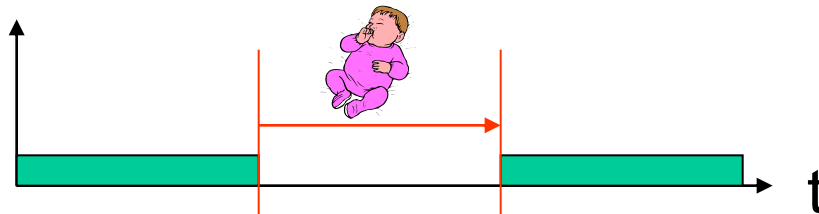
4 types of timing specs required, according to Burns, 1990:

1. Measure elapsed time

Check, how much time has elapsed since last call



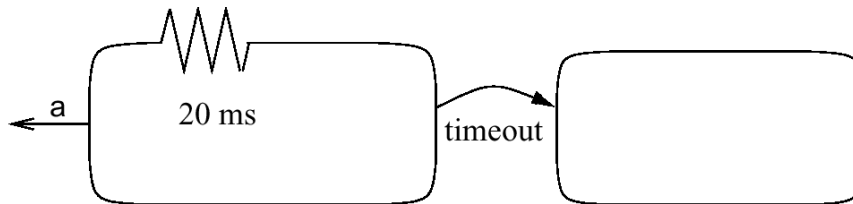
2. Means for delaying processes



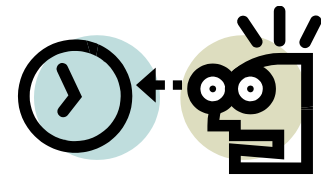
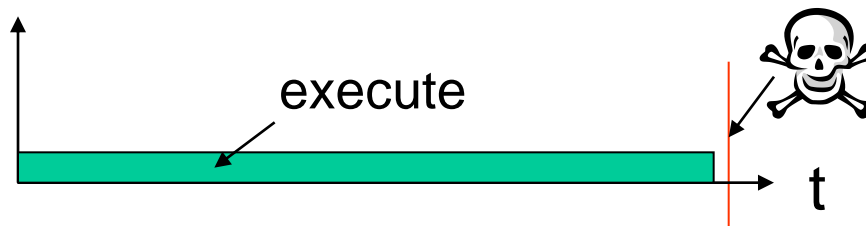
Requirements for specification techniques (3)

Timing (3)

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



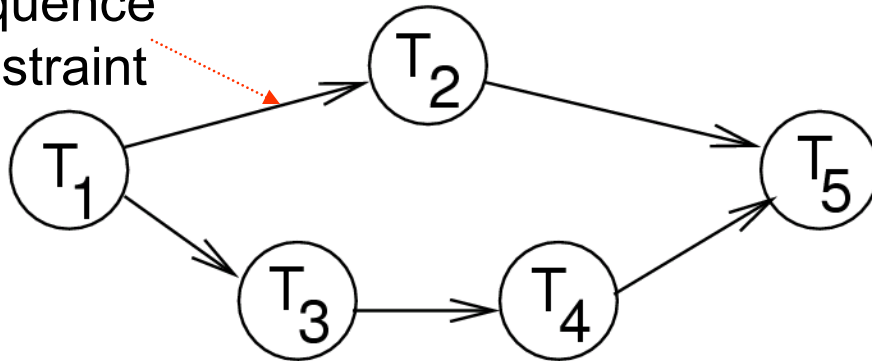
4. Methods for specifying deadlines
Not available or in separate control file.



Dependence graph

Definition

Sequence
constraint



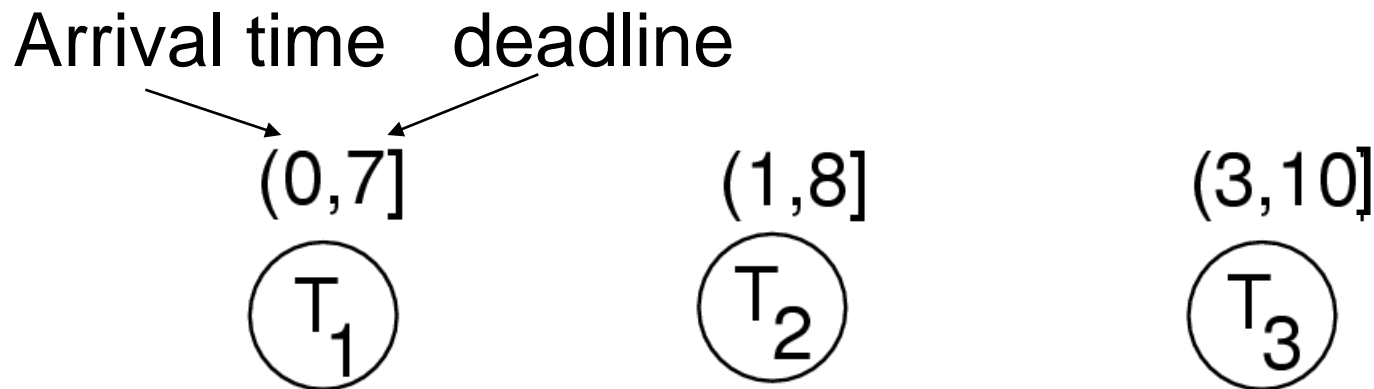
Nodes could be programs
or simple operations

- **Def.:** A **dependence graph** is a directed graph $G=(V,E)$ in which $E \subseteq V \times V$ is a partial order.
- If $(v1, v2) \in E$, then $v1$ is called an **immediate predecessor** of $v2$ and $v2$ is called an **immediate successor** of $v1$.
- Suppose E^* is the transitive closure of E .
If $(v1, v2) \in E^*$, then $v1$ is called a **predecessor** of $v2$ and $v2$ is called a **successor** of $v1$.

Dependence graph

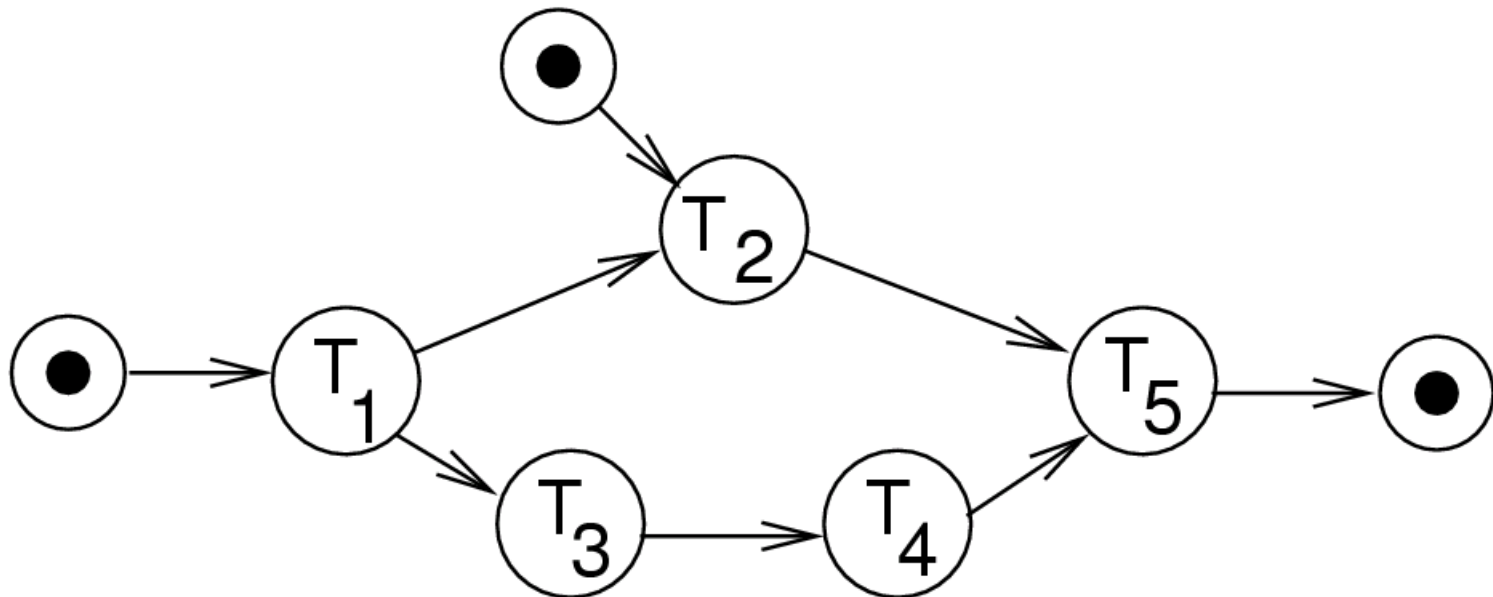
Timing information

Dependence graphs may contain additional information,
for example: Timing information



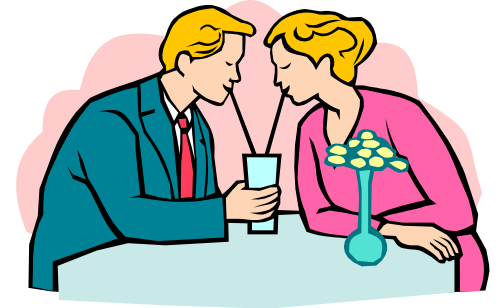
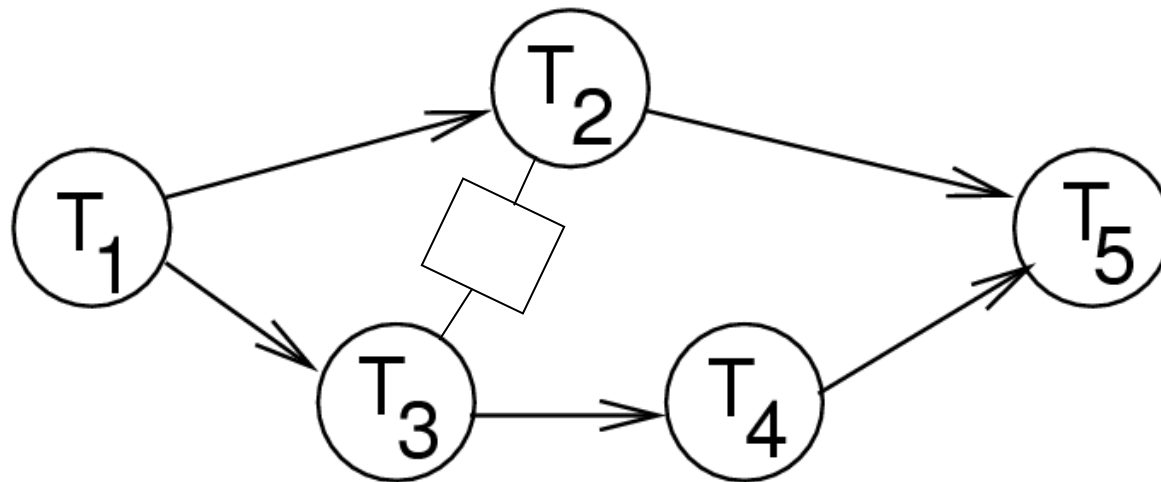
Dependence graph

I/O-information



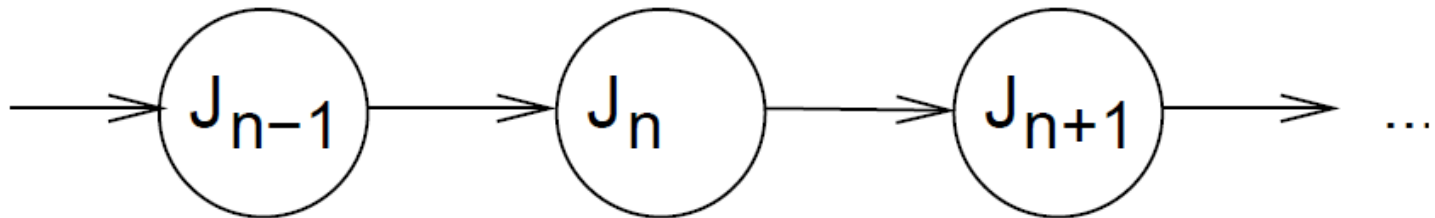
Dependence graph

Shared resources



Dependence graph

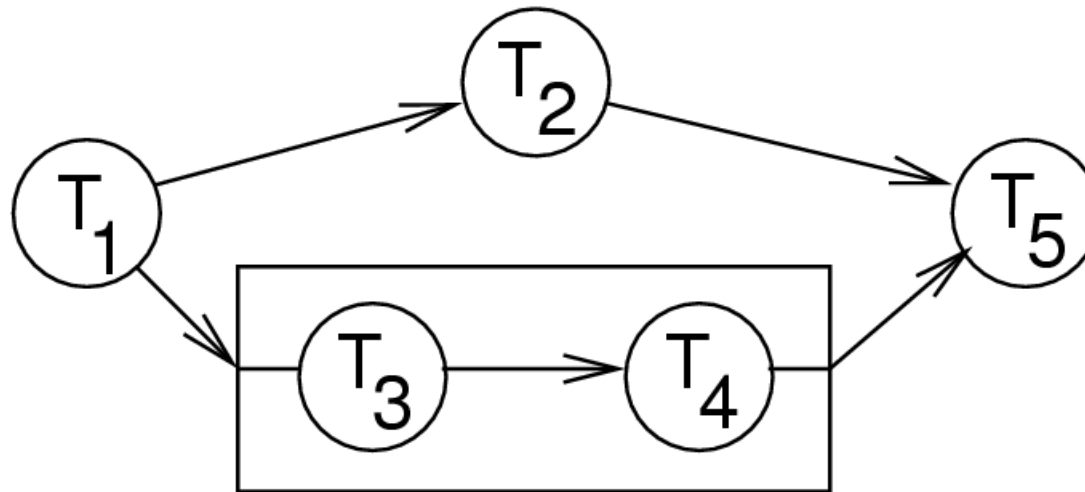
Periodic schedules



- A **job** is single execution of the dependence graph
- Periodic dependence graphs are infinite

Dependence graph

Hierarchical task graphs



Models of 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  
}
```

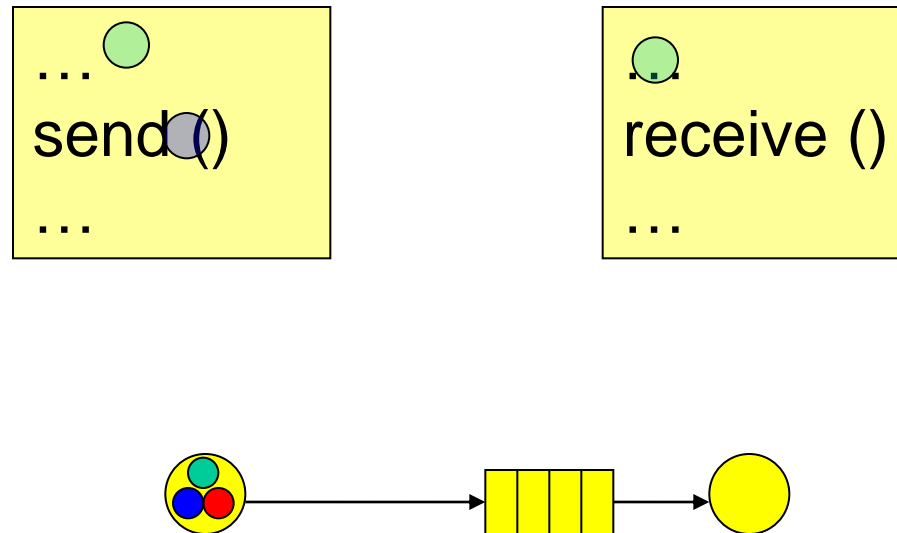
Race-free access
to shared memory
protected by S
possible

This model may be supported by:

- mutual exclusion for critical sections
- cache coherency protocols

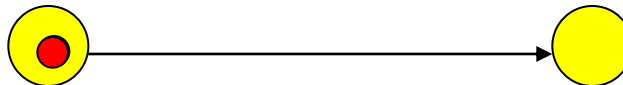
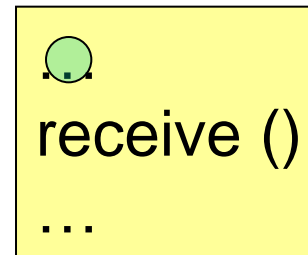
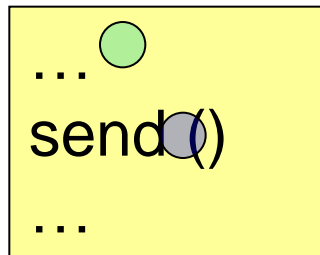
Non-blocking/asynchronous message passing

Sender does not have to wait until message has arrived;
potential problem: buffer overflow



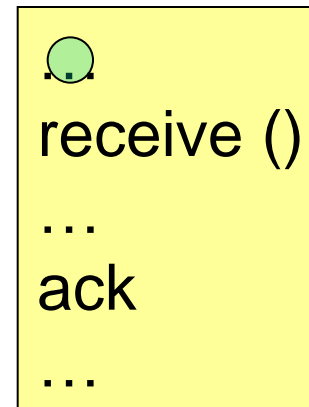
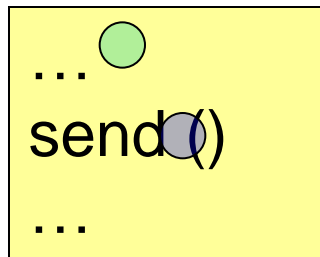
Blocking/synchronous message passing *rendez-vous*

Sender will wait until receiver has received message



Extended *rendez-vous*

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



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 Edward A. Lee, UCB
- Axel Jantsch: Modeling Embedded Systems and Soc's: Concurrency and Time in Models of Computation, Morgan-Kaufman, 2004

Ptolemy

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

<http://ptolemy.berkeley.edu/>



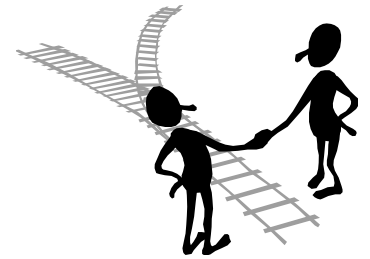
Available examples are restricted to a subset of the supported models of computation.

Newton's cradle



Facing reality

No language that meets all language requirements
☞ using compromises



Summary

Search for other models of computation =

- models of components
 - finite state machines (FSMs)
 - data flow,
- models for communication
 - Shared memory
 - Message passing