

Embedded System Foundations of Cyber-Physical Systems

Peter Marwedel
TU Dortmund,
Dortmund, Germany

2011年 08 月 08 日



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

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

Embedded Systems & Cyber-Physical Systems

“Dortmund“ Definition: [Peter Marwedel]

Embedded systems are information processing systems embedded into a larger product


Berkeley: [Edward A. Lee]:

Embedded software is software integrated with **physical processes. The technical problem is managing **time** and **concurrency** in computational systems.**

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

Application area Automotive electronics: clearly cyber-physical

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 ...
- 
- © P. Marwedel, 2011
- Multiple networks
 - Multiple networked processors

Application area avionics: also cyber-physical

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



© P. Marwedel, 2011

Dependability is of outmost importance.

More application areas

- Railways
- Telecommunication
- Consumer electronics
- Robotics
- Public safety
- Smart homes



Mostly cyber-physical

© P. Marwedel, 2011

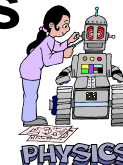
Growing importance of cyber-physical/ embedded systems

- *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]*
- Funding in the 7th European Framework
- Creation of the ARTEMIS Joint Undertaking in Europe
- Funding of CPS research in the US
- Joint education effort of Taiwanese Universities
-



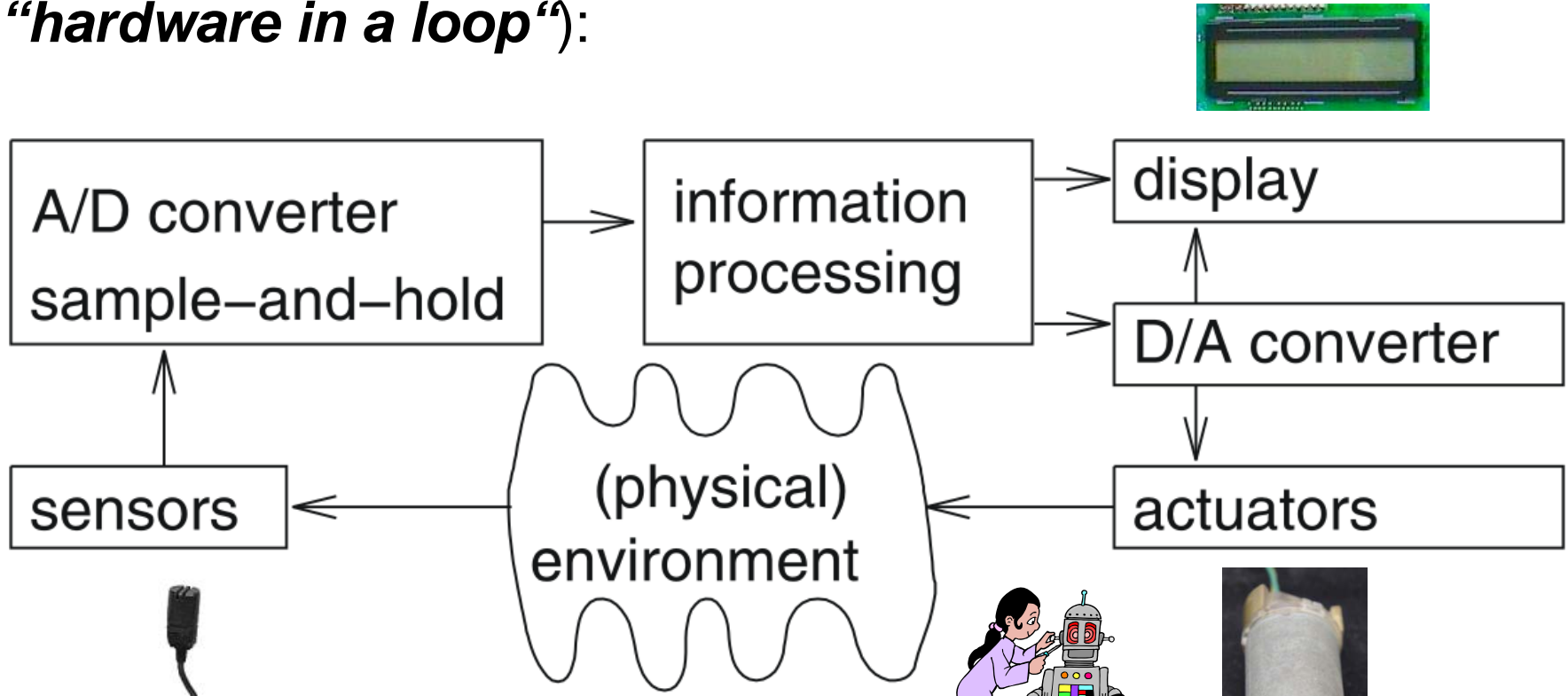
Characteristics of cyber-physical/ embedded systems

- Must be **dependable**
- Must be **efficient**
 - energy, code-size,
 - run-time, weight,
 - cost efficient
- **Dedicated** towards a certain **application**
- **Dedicated user interface**
- Many CPS/ES must meet **real-time constraints**
- **Connected to physical environment**
- **Hybrid systems** (analog + digital parts).
- Typically, CPS/ES are **reactive systems**:
(In continual interaction with its environment)

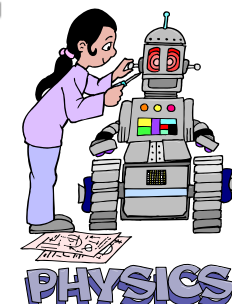


CPS & ES Hardware

CPS & ES hardware is frequently used in a loop (*“hardware in a loop“*):



Cyber-physical systems (!)



© P. Marwedel, 2011

Real-time constraints

- CPS 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

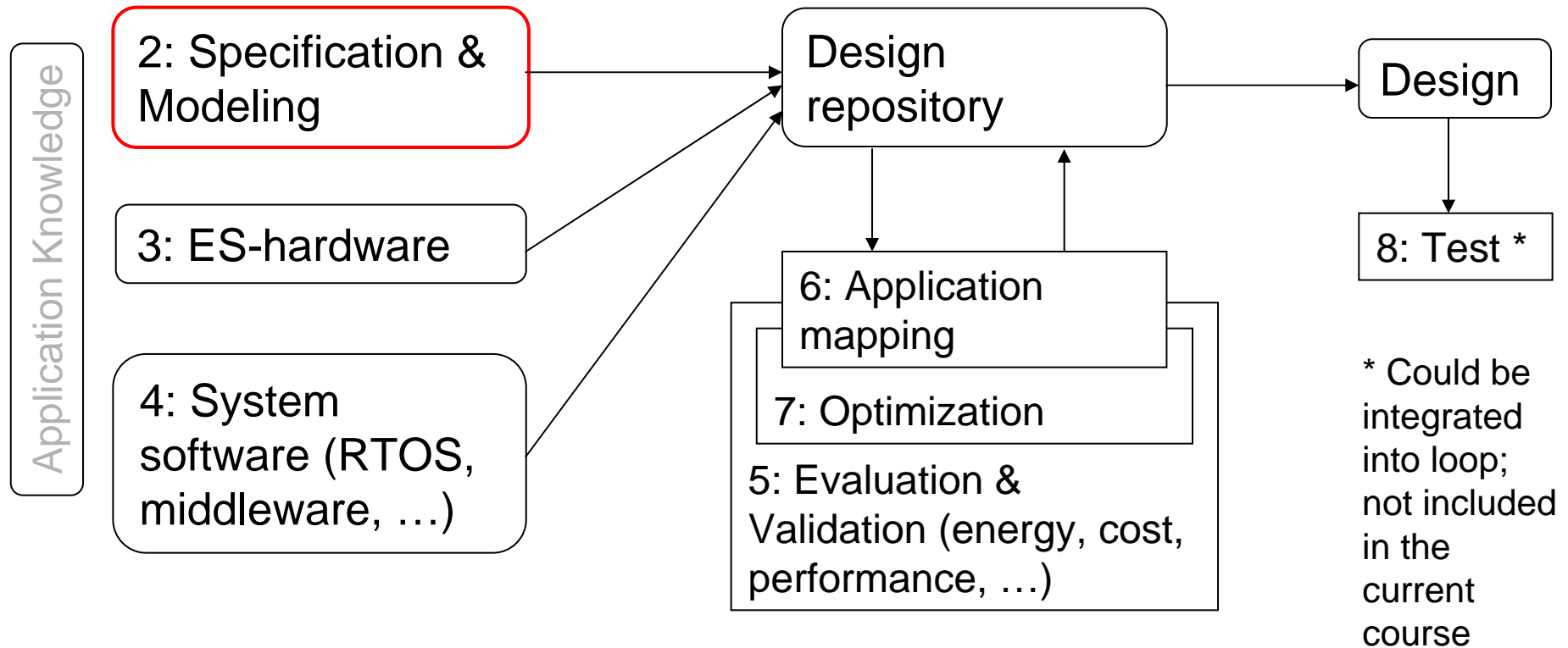


Challenges for Software in CPS

- Dynamic environments
- Capture the required behaviour!
- Validate specifications
- Efficient translation of specifications into implementations!
- How can we check that we meet real-time constraints?
- How do we validate embedded real-time software? (large volumes of data, testing may be safety-critical)

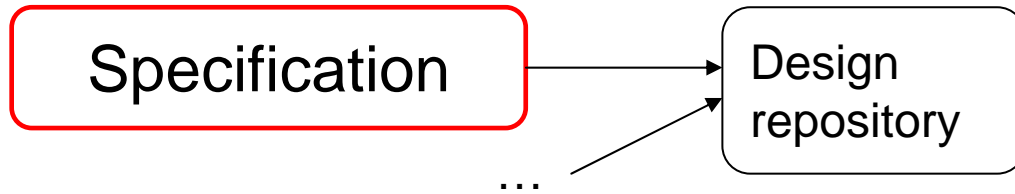


Structure of this course



Generic loop: tool chains differ in the number and type of iterations
Numbers denote sequence of chapters

Motivation for considering specs & models



- Why considering specs & models?
- 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?




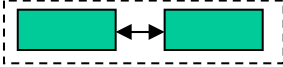

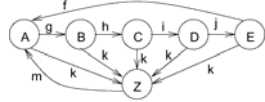


Models

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?

Specification of CPS/ES: Requirements for models

- Hierarchy 
- Compositional behavior 
- Timing behavior 
- State-oriented behavior 
- Event-handling 
- Concurrency
- Synchronization and communication
- Presence of programming elements
- Executability
- Support for the design of large systems
- No obstacles for efficient implementation 
- Domain-specific support
- Non-functional properties

No single model will meet all requirements

 *compromises*

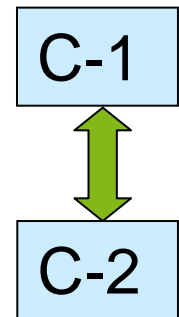
Models of computation

- Definition -

What does it mean, “to compute”?

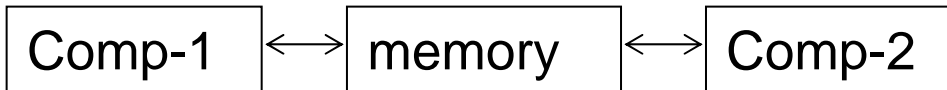
Models of computation define:

- 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.

Communication via shared memory

Several threads access the same memory

- Very fast communication technique (no extra copying)
- Potential race conditions:



```
thread a {  
  u = 1;  
  if u < 5 { u = u + 1; .. }  
}
```

```
thread b {  
  ..  
  u = 5  
}
```

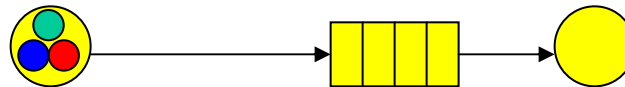
Context switch after the test could result in $u == 6$.

☞ inconsistent results possible

☞ Critical sections = sections at which exclusive access to resource r (e.g. shared memory) must be guaranteed

Non-blocking/ asynchronous message passing

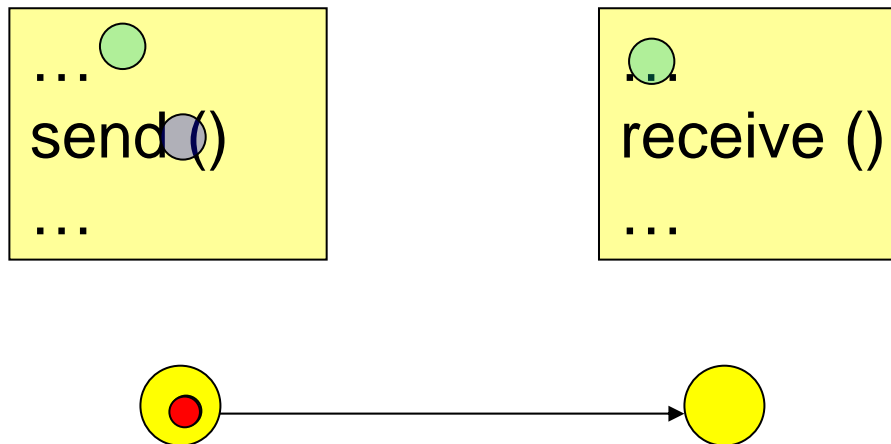
Sender does not have to wait until message has arrived;



Potential problem: buffer overflow

Rendez-vous, Blocking/ synchronous message passing

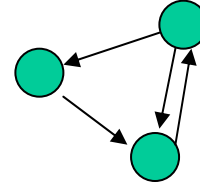
Sender will wait until receiver has received message



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)

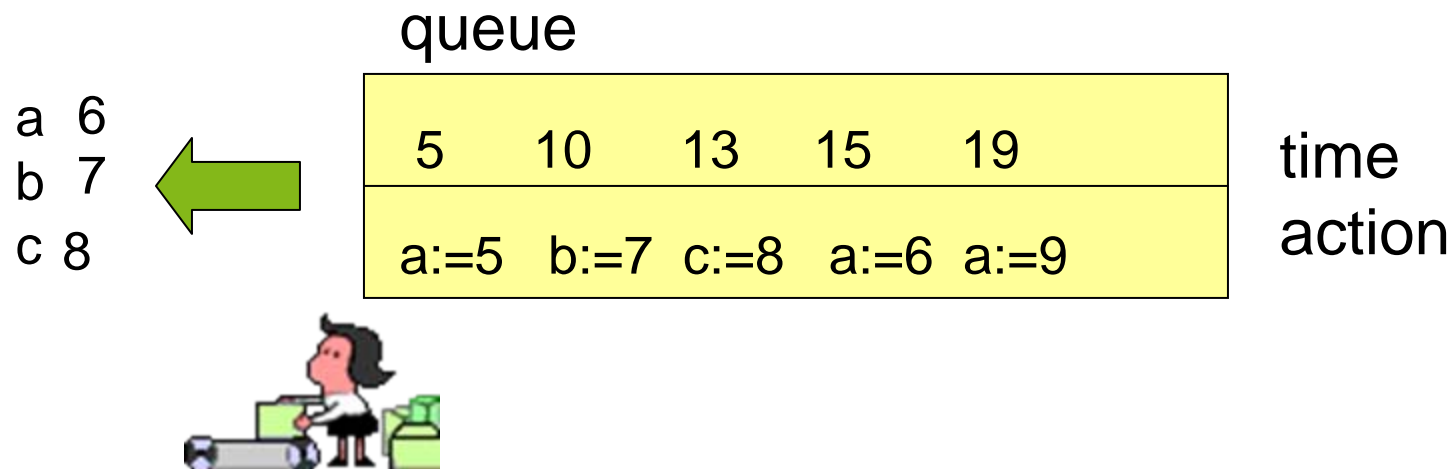
- Differential equations

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



Organization of computations within the components (2)

- Discrete event model



- Von Neumann model

Sequential execution, program memory etc.

Models of computation considered in this course

Communication/ local computation	Shared memory	Message passing	
		Synchronous	Asynchronous
Undefined components	Plain text, use cases (Message) sequence charts		
Communic. 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	

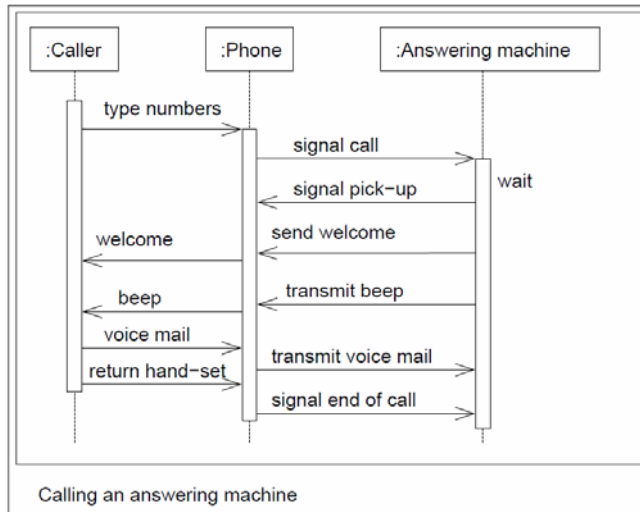
* Classification is based on implementation of VHDL, Verilog, SystemC with central queue

Support for early design phases

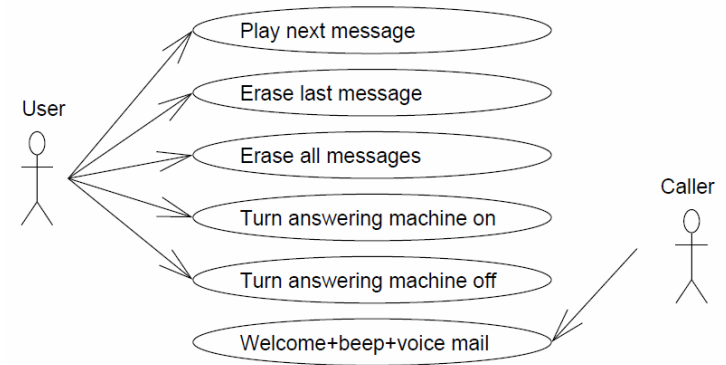
- Informal text

The system must respond to incoming calls. It must play the welcome message followed by a beep and then start recording ...

- (Message) sequence charts



- Uses cases



Similar to SW specification

Models of computation considered in this course

Communication/ local computation	Shared memory	Message passing Synchronous Asynchronous	
Undefined components	Plain text, use cases (Message) sequence charts		
Communic. 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	

* Classification is based on implementation of VHDL, Verilog, SystemC with central queue

StateCharts

Extending classical automata to model ES & CPS

- Adding timing with timed automata (☞ tutorial by Larsen)
- Adding hierarchy:

Complex graphs cannot be understood by humans.

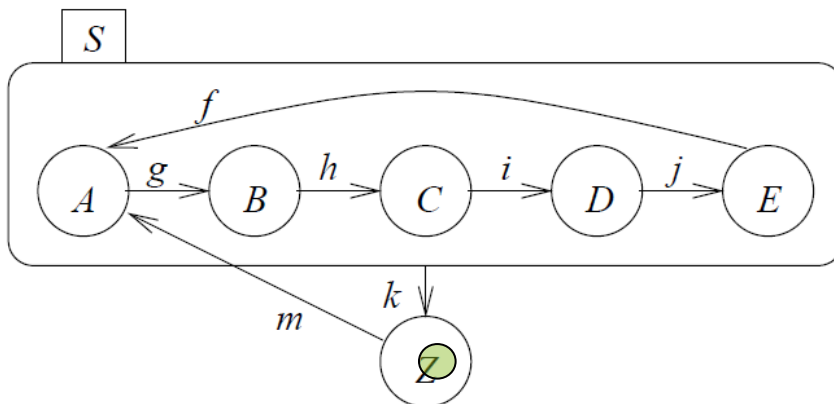
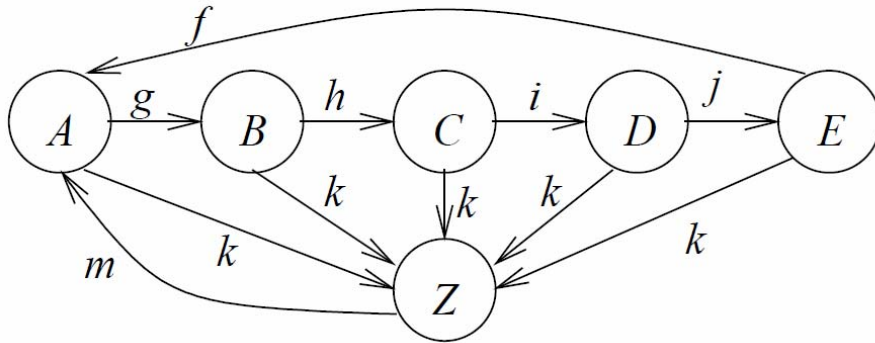
☞ Introduction of hierarchy ☞ StateCharts [Harel, 1987]

StateChart = *the only unused combination of „flow“ or „state“ with „diagram“ or „chart“*

Used here as a (prominent) example of a model of computation based on shared memory communication, appropriate only for local (non-distributed) systems



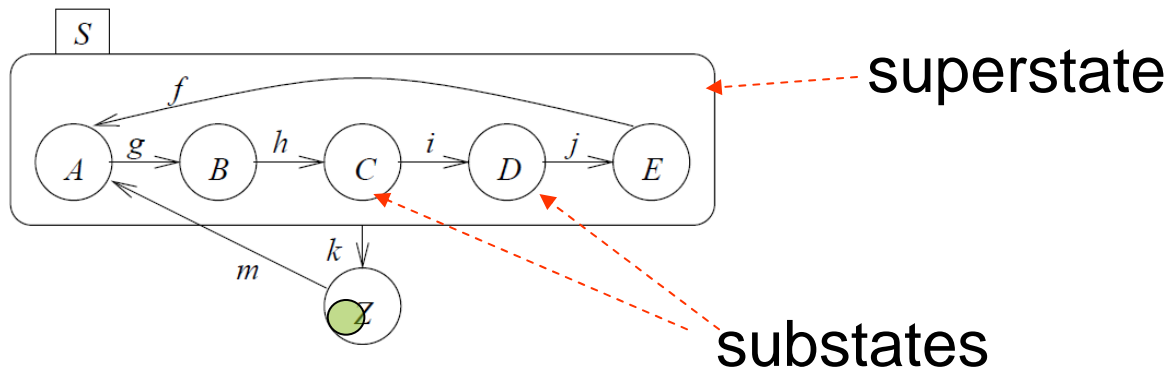
Introducing hierarchy



FSM will be **in** exactly one of the substates of S if S is **active** (either in A or in B or ..)

Definitions

- Current states of FSMs are also called **active** states.
- States which are not composed of other states are called **basic states**.
- States containing other states are called **super-states**.
- Super-states S are called **OR-super-states**, if exactly one of the sub-states of S is active whenever S is active.



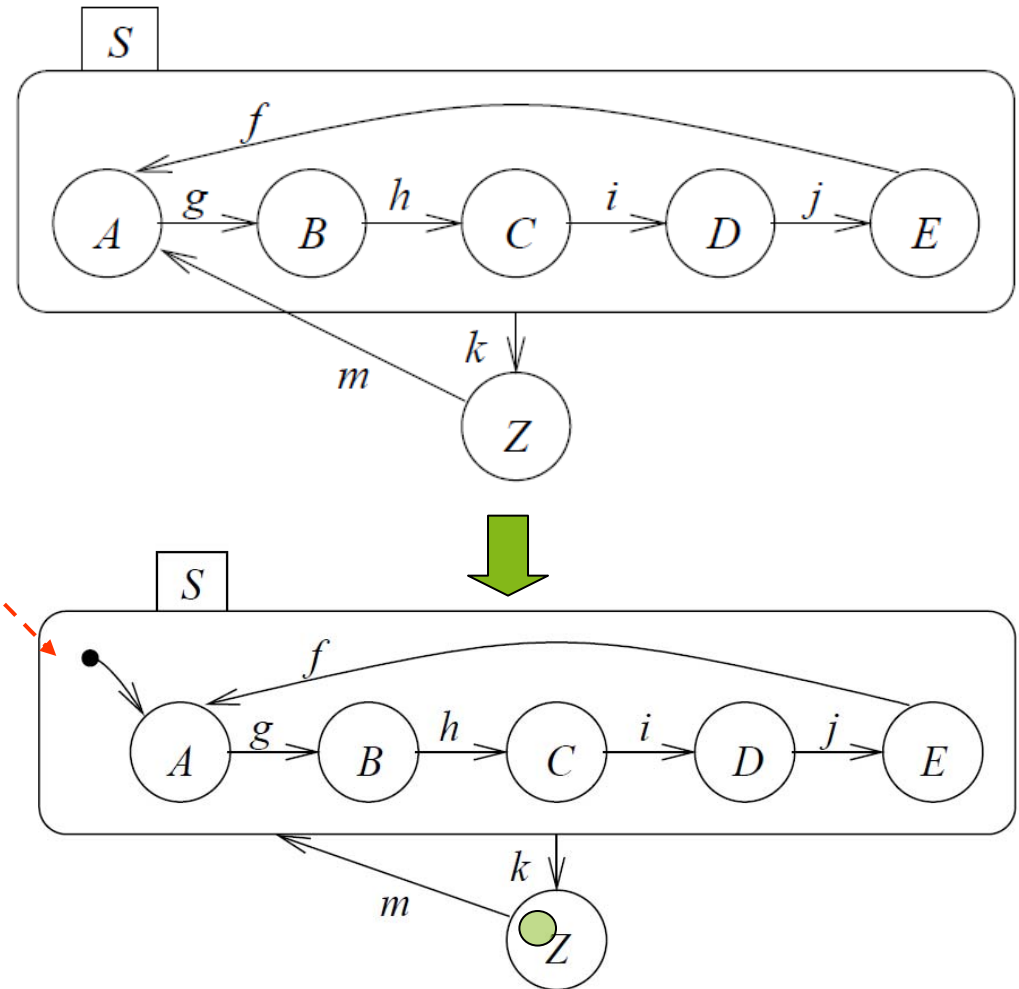
Default state mechanism

Try to hide internal structure from outside world!

☞ Default state

Filled circle indicates sub-state entered whenever super-state is entered.

Not a state by itself!



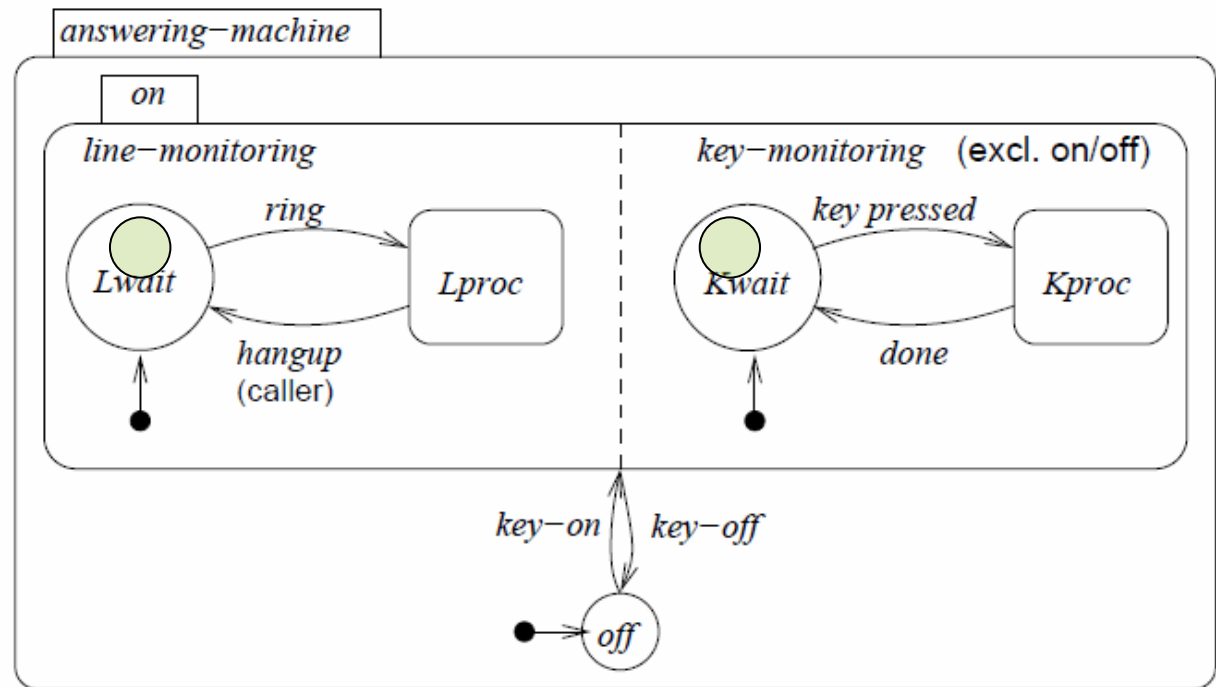
Concurrency

Convenient ways of describing concurrency are required.

AND-super-states:

FSM is in **all** (immediate) sub-states of a super-state.

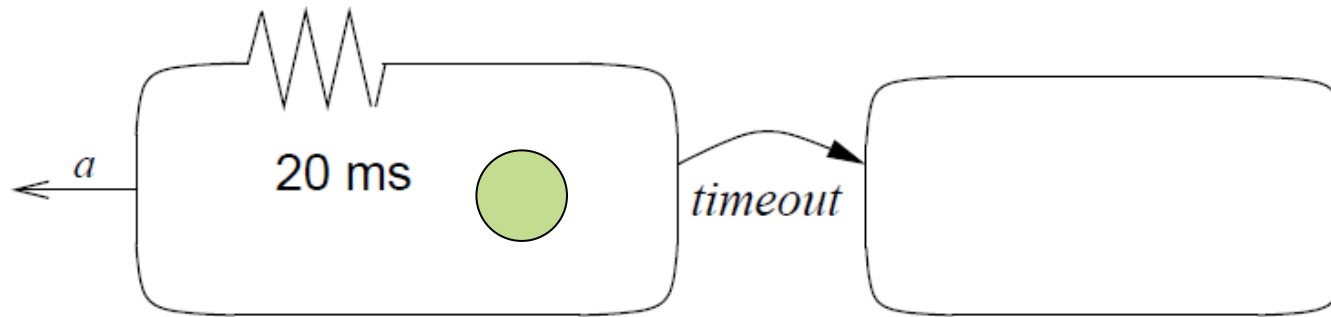
Example:



Timers

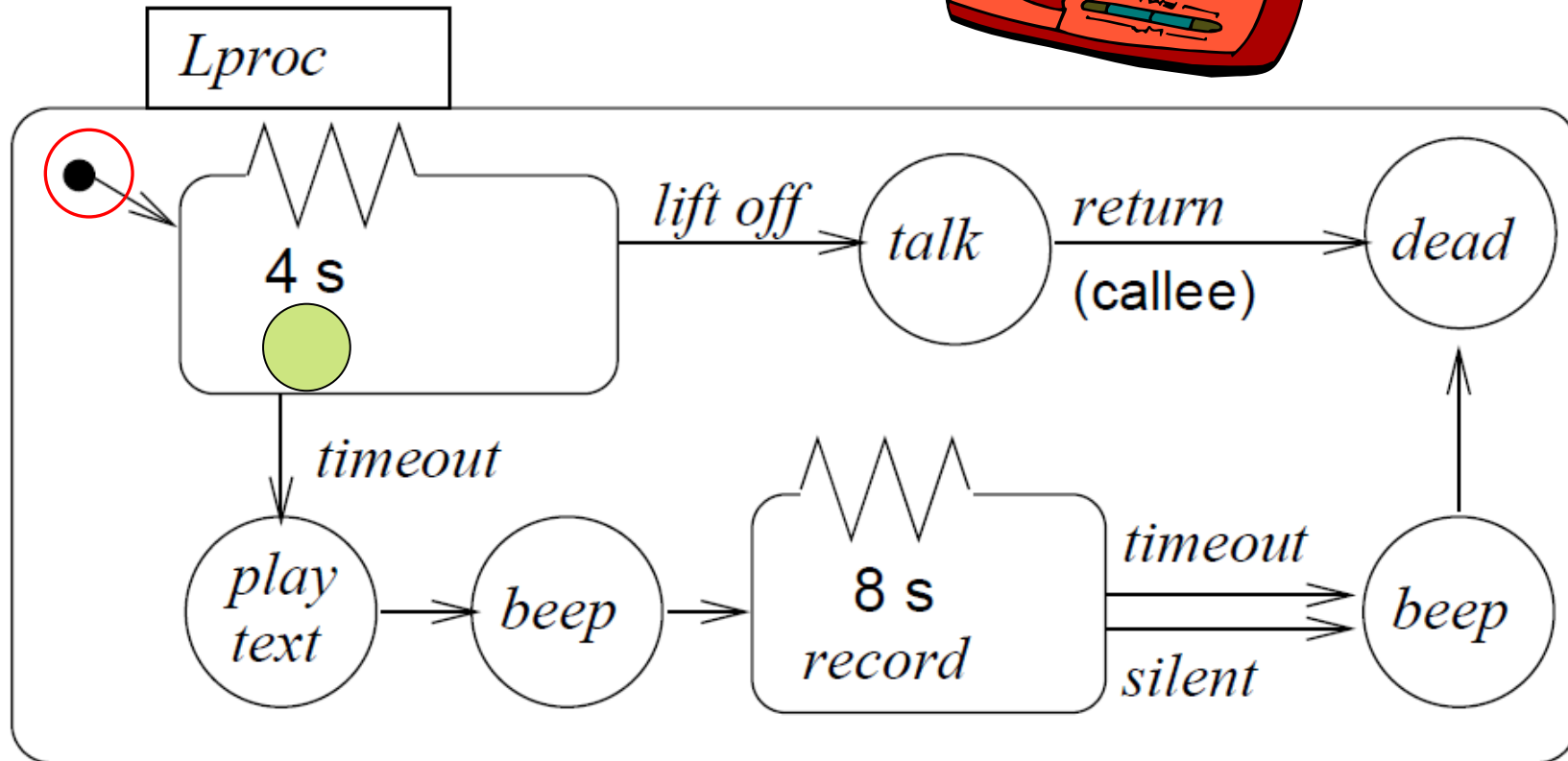
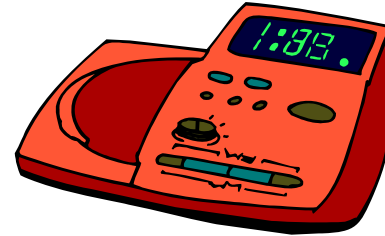
Since time needs to be modeled in embedded systems, timers need to be modeled.

In StateCharts, special edges can be used for timeouts.



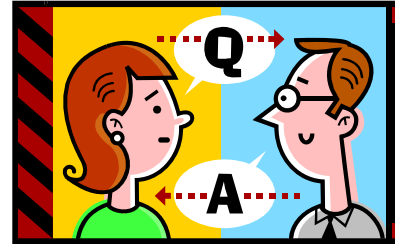
If event *a* does not happen while the system is in the left state for 20 ms, a timeout will take place.

Using timers in an answering machine

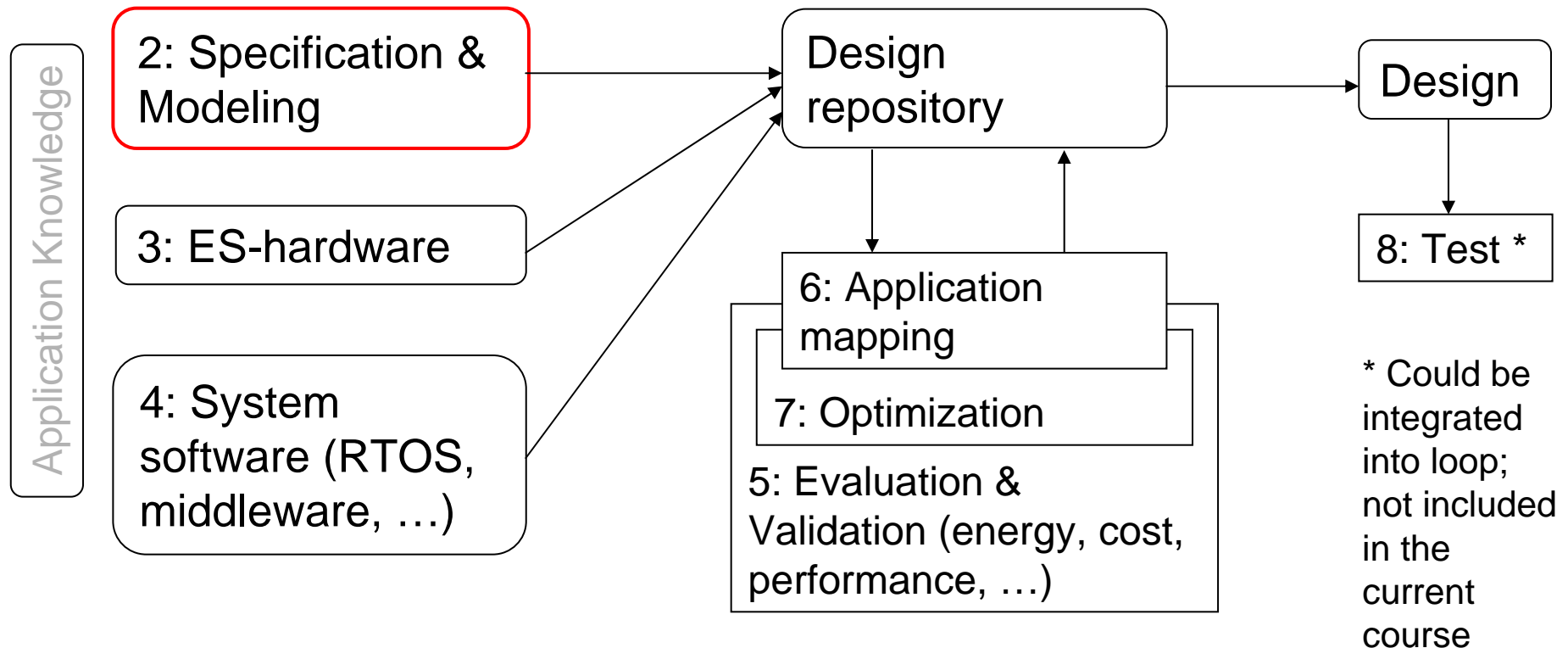


Questions?

Q&A?



Structure of this course



Generic loop: tool chains differ in the number and type of iterations
Numbers denote sequence of chapters

The StateCharts simulation phases (StateMate Semantics)

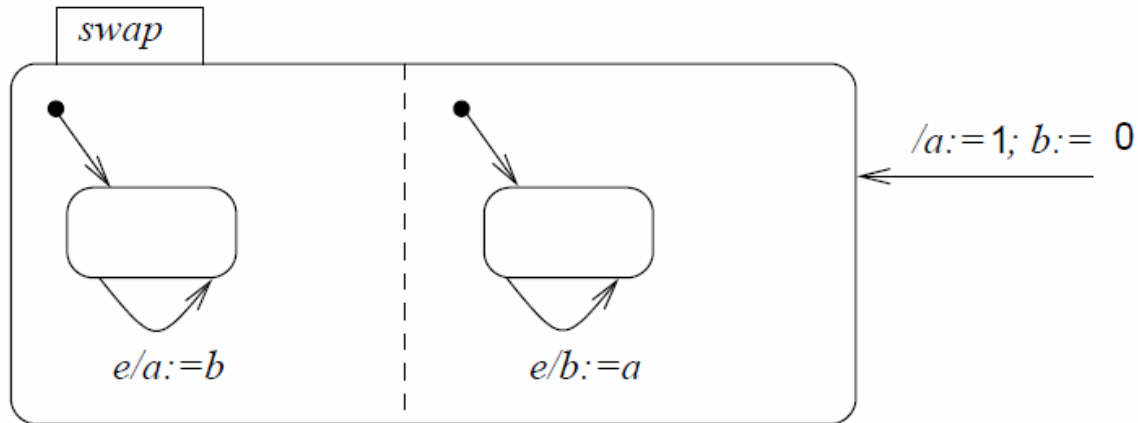
How are edge labels evaluated?

Three phases:

1. Effect of external changes on events and conditions is evaluated,
2. The set of transitions to be made in the current step and right hand sides of assignments are computed,
3. Transitions become effective, variables obtain new values.

Separation into phases 2 and 3 guarantees and reproducible behavior.

Example

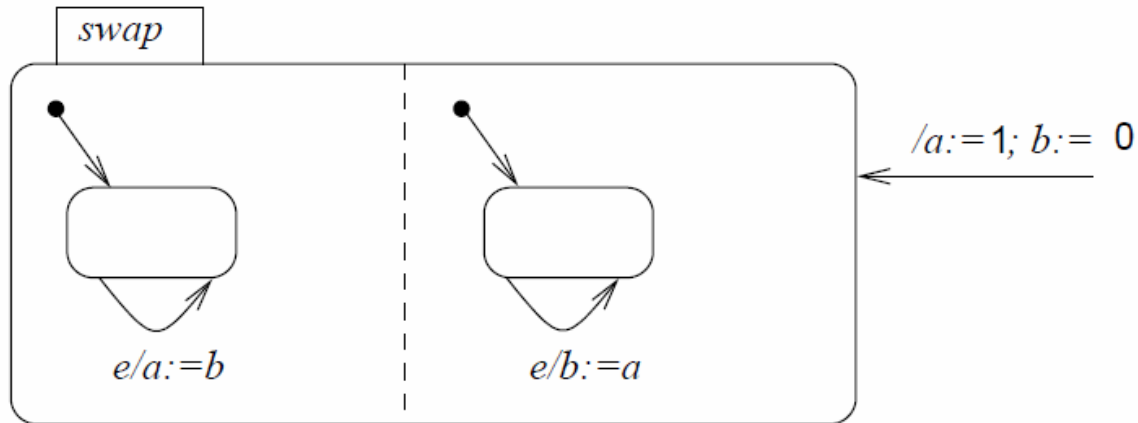


In phase 2, variables a and b are assigned to temporary variables:

In phase 3, these are assigned to a and b .

As a result, variables a and b are swapped.

Example (2)

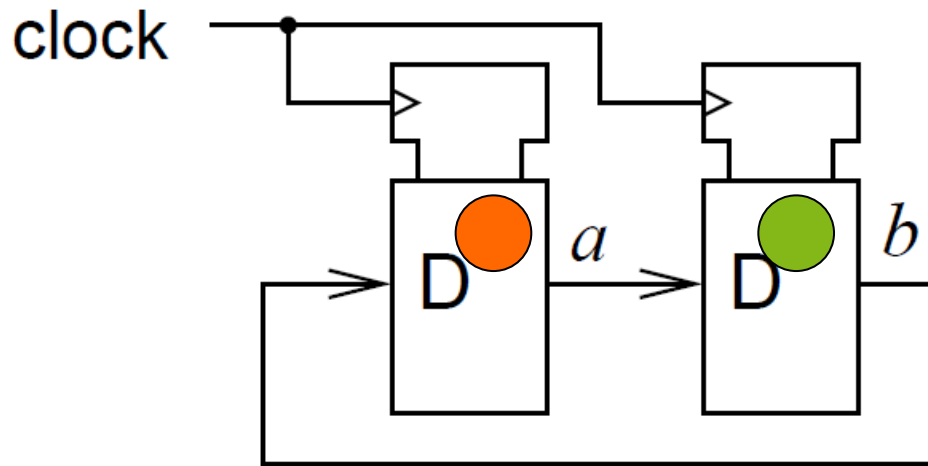


In a single phase environment, executing the left state first would assign the old value of b ($=0$) to a and b :

Executing the right state first would assign the old value of a ($=1$) to a and b .

The result would depend on the execution order.

Reflects model of clocked hardware

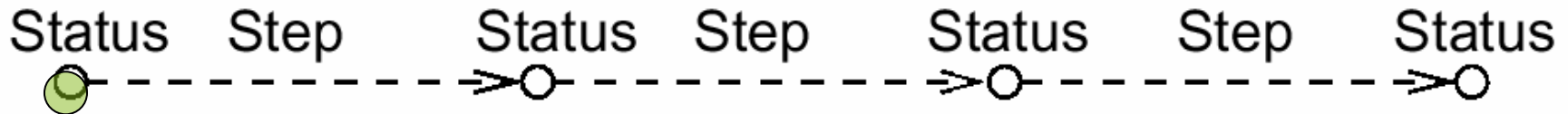


In an actual clocked (synchronous) hardware system, both registers would be swapped as well.

Same separation into phases found in other languages as well, especially those that are intended to model hardware.

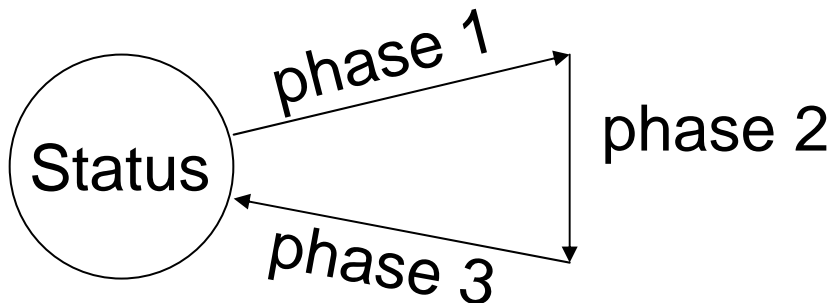
Steps

Execution of a StateMate model consists of a sequence of (status, step) pairs



Status= values of all variables + set of events + current time

Step = execution of the three phases (**StateMate** semantics)



Other implementations of StateCharts do not have these 3 phases!

Determinate vs. deterministic

- Kahn (1974) calls a system **determinate** if we will always obtain the same result for a fixed set (and timing) of inputs
- Others call this property **deterministic**

However, this term has several meanings:

- Non-deterministic finite state machines
- Non-deterministic operators
(e.g. + with non-deterministic result in low order bits)
- Behavior not known before run-time
(unknown input results in non-determinism)
- In the sense of determinate as used by Kahn

In order to avoid confusion, we use the term “determinate” today.

StateCharts determinate or not?

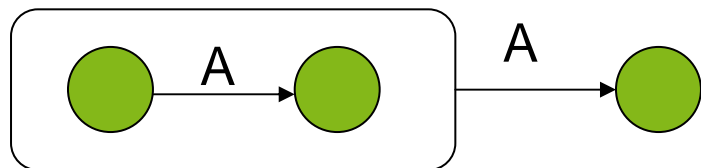
Determinate (in this context) means:

Must all simulators return the same result for a given input?

- Separation into 2 phases a required condition
- Semantics \neq StateMate semantics may be non-determinate

Potential other sources of non-determinate behavior:

- Choice between conflicting transitions resolved arbitrarily



Tools typically issue a warning if such a situation could exist

→ Determinate behavior for StateMate semantics if transition conflicts are resolved and no other sources of undefined behavior exist

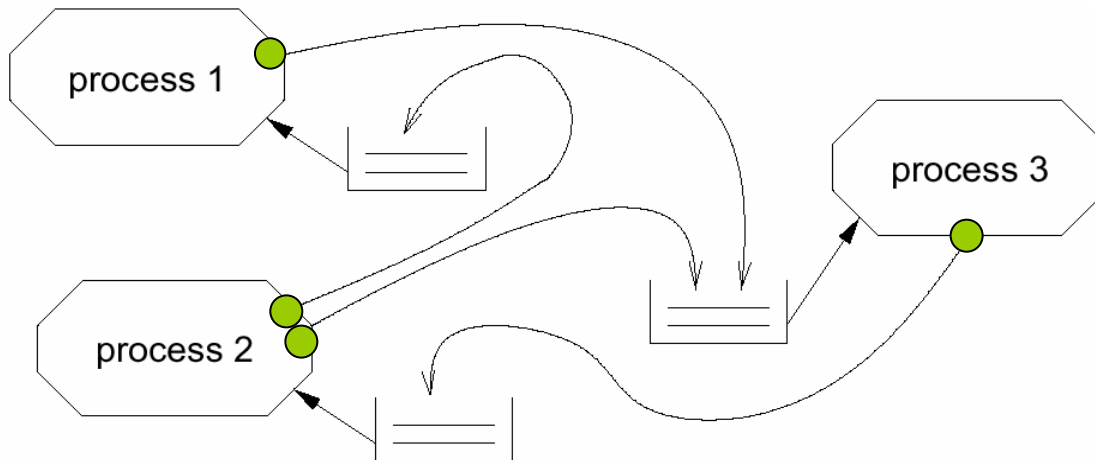
Models of computation considered in this course

Communication/ local computation	Shared memory	Message passing	
		Synchronous	Asynchronous
Undefined components	Plain text, use cases (Message) sequence charts		
Communic. 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	

* Classification is based on implementation of VHDL, Verilog, SystemC with central queue

Message passing in SDL

Communication between FSMs (or “processes“)
is based on **message-passing**, assuming a **potentially indefinitely large FIFO-queue**.

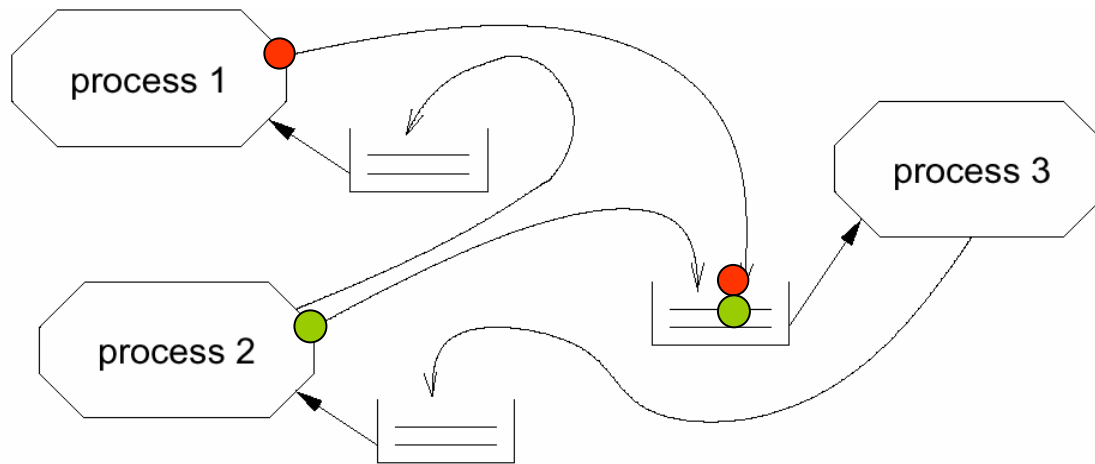


- Each process fetches next entry from FIFO,
- checks if input enables transition,
- if yes: transition takes place,
- if no: input is ignored (exception: SAVE-mechanism).

Determinate?

Let tokens be arriving at FIFO at the same time:

☞ Order in which they are stored, is unknown:



All orders are legal: ☞ simulators can show different behaviors for the same input, all of which are correct.

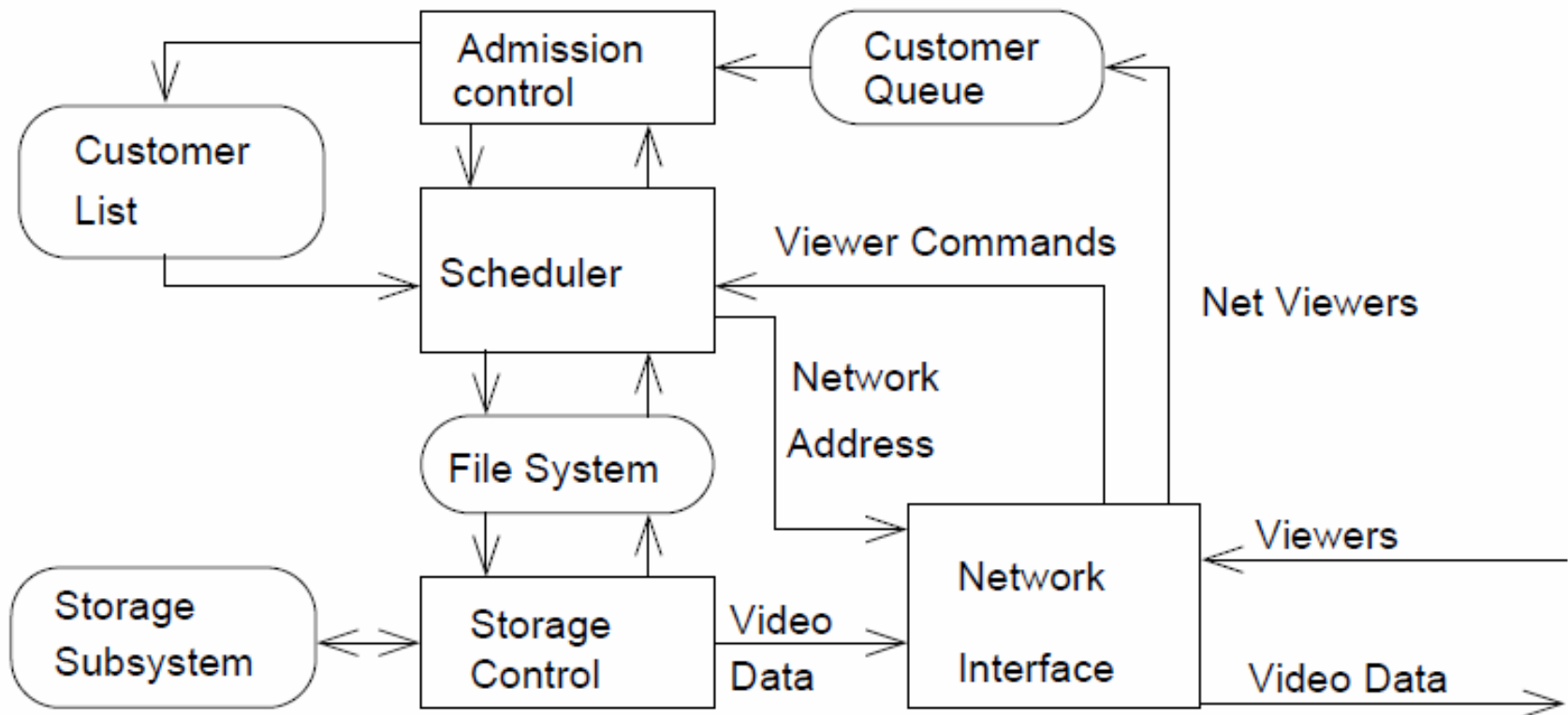
Models of computation considered in this course

Communication/ local computation	Shared memory	Message passing	
		Synchronous	Asynchronous
Undefined components	Plain text, use cases (Message) sequence charts		
Communic. 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	

* Classification is based on implementation of VHDL, Verilog, SystemC with central queue

Data flow as a “natural” model of applications

Example: Video on demand system



www.ece.ubc.ca/~irenek/techpaps/vod/vod.html

Data flow modeling

Definition: Data flow modeling is ...

“the process of identifying, modeling and documenting how data moves around an information system.

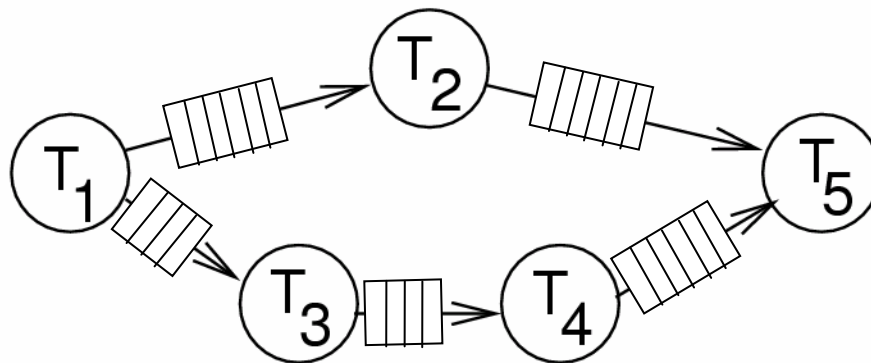
Data flow modeling examines

- *processes (activities that transform data from one form to another),*
- *data stores (the holding areas for data),*
- *external entities (what sends data into a system or receives data from a system, and*
- *data flows (routes by which data can flow)”.*

[Wikipedia: Structured systems analysis and design method.
http://en.wikipedia.org/wiki/Structured_Systems_Analysis_and_Design_Methodology, 2010 (formatting added)].

Kahn process networks

- Each component is a program/task/process, not an FSM
- Communication is by FIFOs; no overflow considered
 - ☞ writes never have to wait,
 - ☞ reads wait if FIFO is empty.



- Only one sender and one receiver per FIFO
 - ☞ no SDL-like conflicts at FIFOs

Example

```
Process f(in int u, in int v, out int w){
```

```
  int i; bool b = true;
```

```
  for (;;) {
```

```
    i = b ? wait(u) : wait(v);
```

//wait returns next token in FIFO, waits if empty

```
    send (i,w); //writes a token into a FIFO w/o blocking
```

```
    b = !b;
```

```
  }
```

© R. Gupta (UCSD), W. Wolf (Princeton), 2003

Key beauty of KPNs

- A process cannot check whether data is available before attempting a read.
- A process cannot wait for data for more than one port at a time.
- Therefore, the order of reads depends only on data, not on the arrival time.
- Therefore, for a given input, for Kahn process networks the result will always be the same, regardless of the speed of the nodes.
- ☞ Many applications in cyber-physical/embedded system design: simplifies emulation of real systems.

Models of computation considered in this course

Communication/ local computation	Shared memory	Message passing	
		Synchronous	Asynchronous
Undefined components	Plain text, use cases (Message) sequence charts		
Communic. 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	

* Classification is based on implementation of VHDL, Verilog, SystemC with central queue

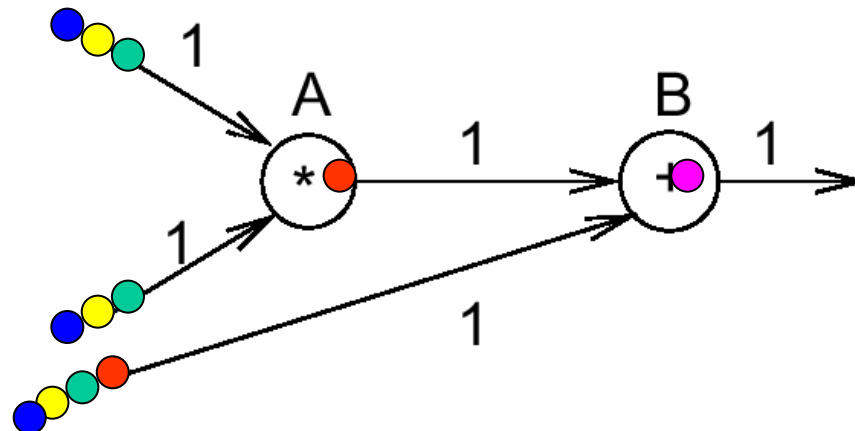
Synchronous data flow (SDF)

Synchronous data flow =

global clock controlling “firing” of nodes

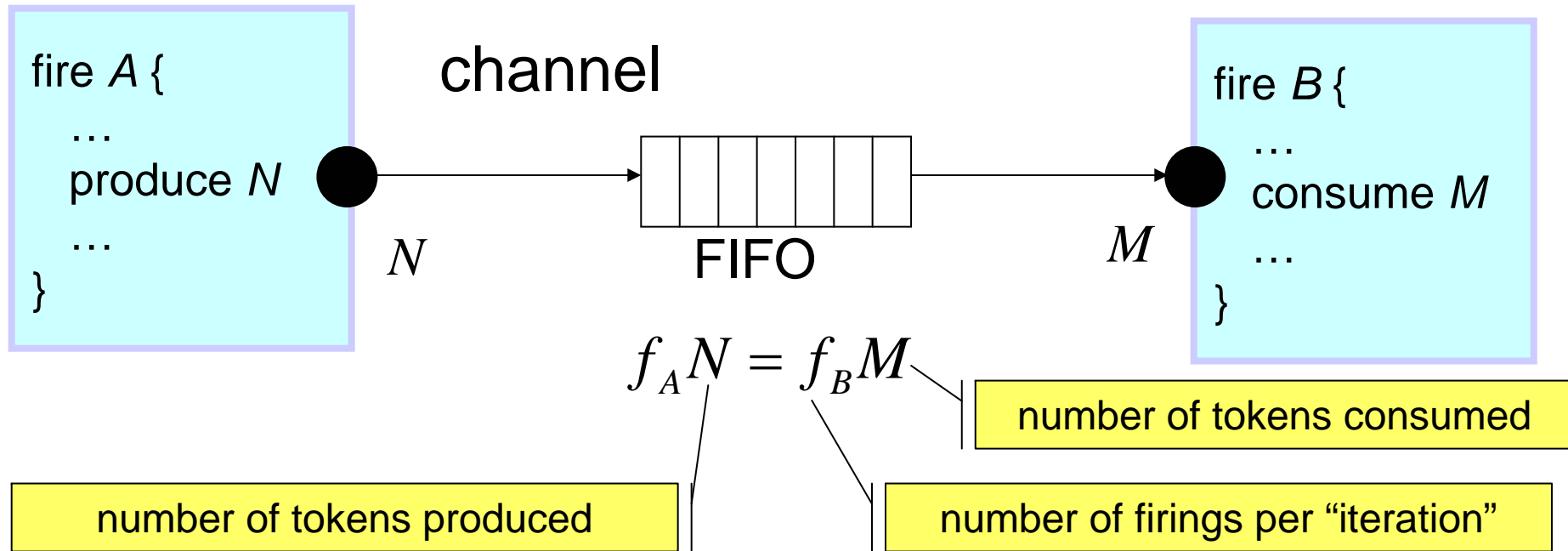
Asynchronous message passing=

tasks do not have to wait until output is accepted.



In the general case, a number of tokens can be produced/ consumed per firing; firing rate depends on # of tokens ...

Multi-rate models & balance equations (one for each channel)



Schedulable statically

In the general case, buffers may be needed at edges.

Decidable:

- buffer memory requirements
- deadlock

Adopted from: ptolemy.eecs.berkeley.edu/presentations/03/streamingEAL.ppt

Models of computation considered in this course

Communication/ local computation	Shared memory	Message passing Synchronous Asynchronous	
Undefined components	Plain text, use cases (Message) sequence charts		
Communic. 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	

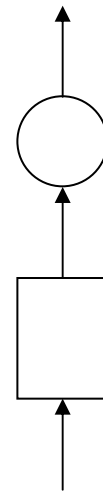
* Classification is based on implementation of VHDL, Verilog, SystemC with central queue

Introduction

Introduced in 1962 by Carl Adam Petri in his PhD thesis. Focus on modeling causal dependencies; no global synchronization assumed (message passing only).

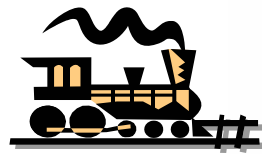
Key elements:

- **Conditions**
Either met or no met.
- **Events**
May take place if certain conditions are met.
- **Flow relation**
Relates conditions and events.



Conditions, events and the flow relation form a **bipartite graph** (graph with two kinds of nodes).

Example: Synchronization at single track rail segment



train entering track from the left

train leaving track to the right

train wanting to go right

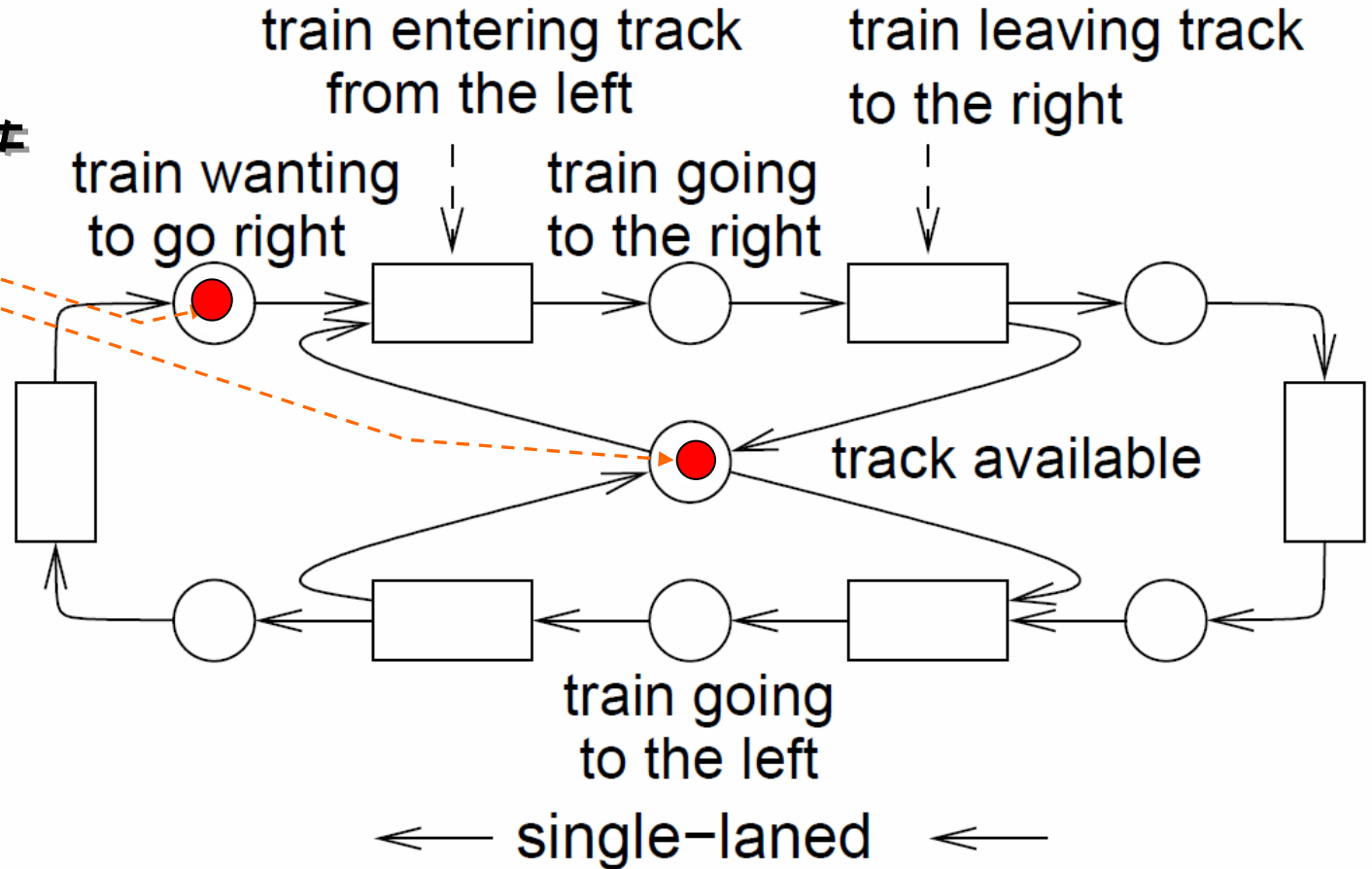
train going to the right

track available

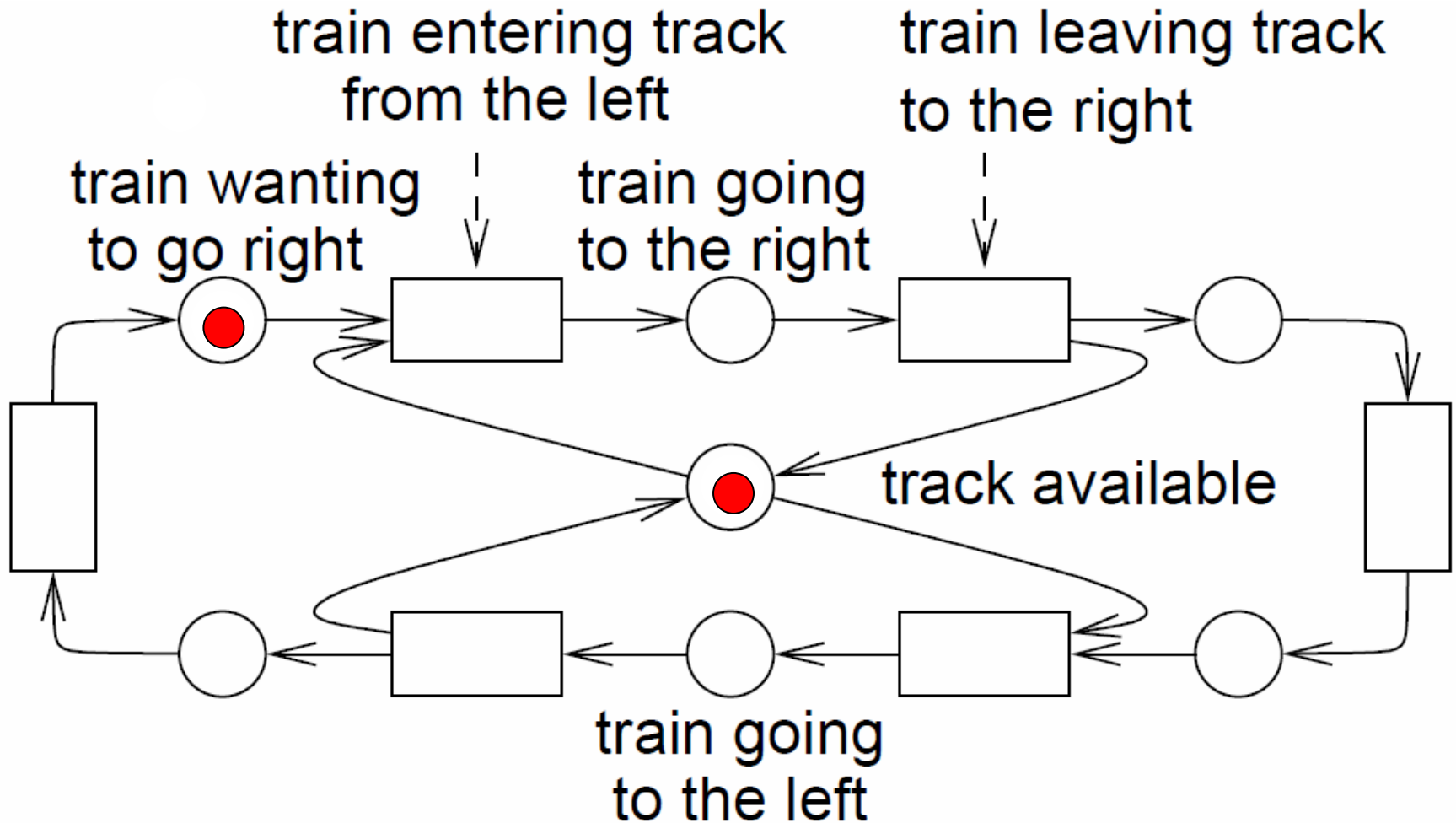
train going to the left

← single-laned ←

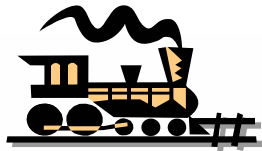
“Preconditions”



Playing the “token game“

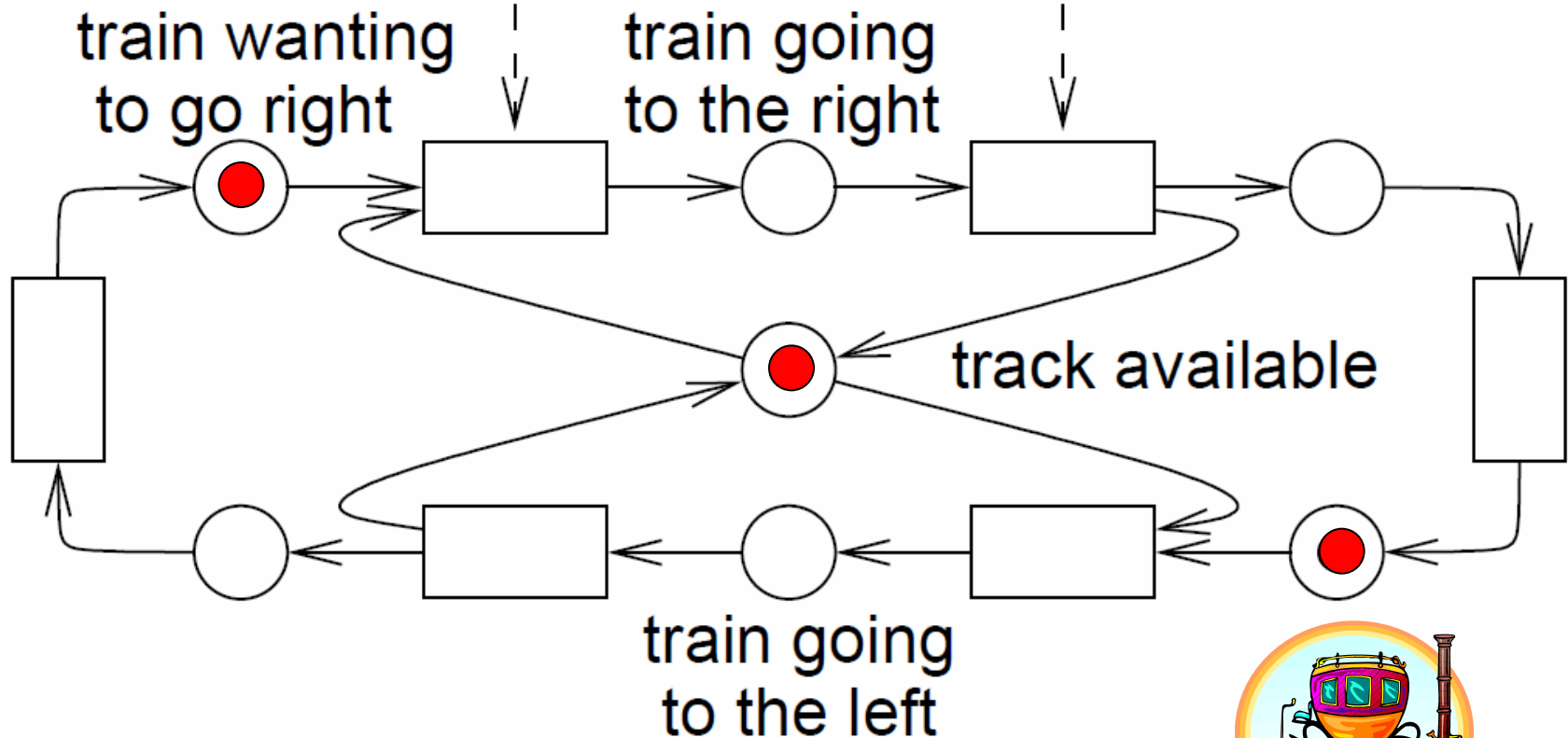


Conflict for resource “track”



train entering track
from the left

train leaving track
to the right



Models of computation considered in this course

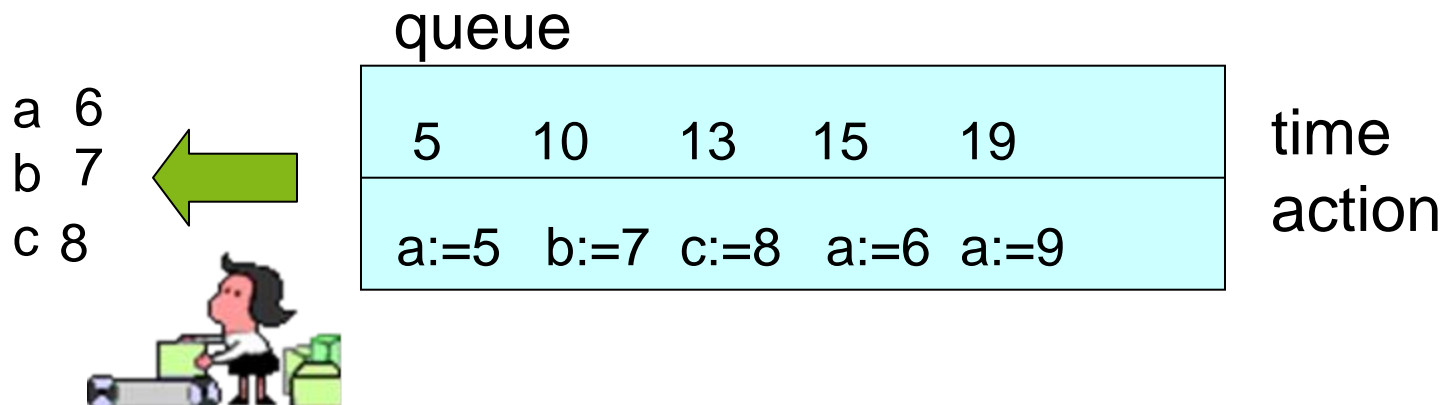
Communication/ local computation	Shared memory	Message passing Synchronous Asynchronous	
Undefined components	Plain text, use cases (Message) sequence charts		
Communic. finite state machines	StateCharts		SDL
Data flow	(Not useful)		Kahn networks, SDF
Petri nets	C/E nets, P/T nets, ...		
Discrete event (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	

* Classification is based on implementation of VHDL, Verilog, SystemC with central queue

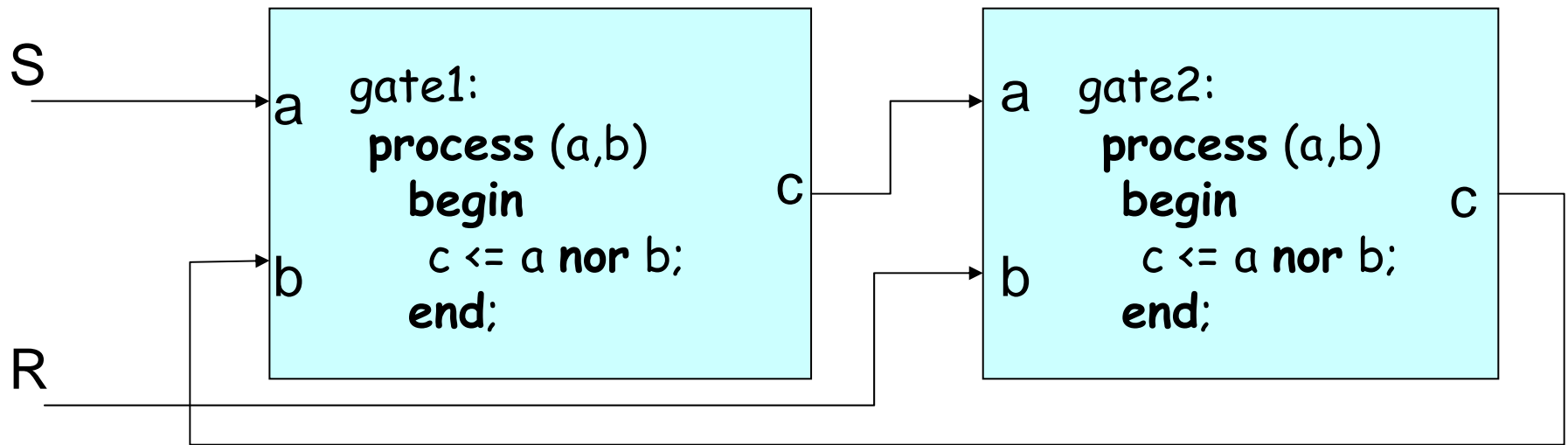
Discrete event semantics

Basic discrete event (DE) semantics

- Queue of future actions, sorted by time
- Loop:
 - Fetch next entry from queue
 - Perform function as listed in entry
 - May include generation of new entries
- Until termination criterion = true



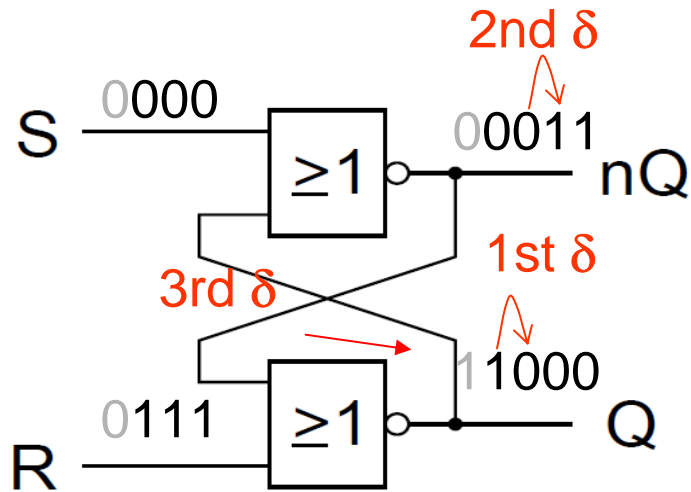
Simple example (VHDL notation)



Processes will wait for changes on their input ports. If they arrive, processes will wake up, compute their code and deposit changes of output signals in the event queue and wait for the next event. If all processes wait, the next entry will be taken from the event queue.

δ -simulation cycles

Simulation of an RS-Flipflop



```

gate1:
  process (S,Q)
  begin
    nQ <= S nor Q;
  end;
gate2:
  process (R,nQ)
  begin
    Q <= R nor nQ;
  end;
  
```

	0ns	0ns+ δ	0ns+2 δ	0ns+3 δ
R	1	1	1	1
S	0	0	0	0
Q	1	0	0	0
nQ	0	0	1	1

δ cycles reflect the fact that no real gate comes with zero delay.

☞ should delay-less signal assignments be allowed at all?

Models of computation considered in this course

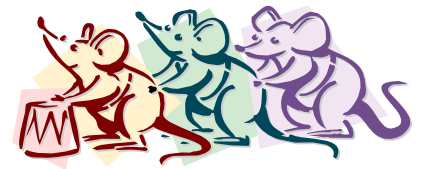
Communication/ local computation	Shared memory	Message passing	
		Synchronous	Asynchronous
Undefined components	Plain text, use cases (Message) sequence charts		
Communic. finite state machines	StateCharts		SDL
Data flow	(Not useful)		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, ...	
Discrete event (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	

* Classification is based on implementation of VHDL, Verilog, SystemC with central queue

Imperative (von-Neumann) model

The von-Neumann model reflects the principles of operation of standard computers:

- Sequential execution of instructions (sequential control flow, fixed sequence of operations)
- Possible branches
- Partitioning of applications into threads
- In most cases:
 - Context switching between threads, frequently based on pre-emption (cooperative multi-tasking or time-triggered context switch less common)
 - Access to shared memory



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); Deadlock possible.



Communication/synchronization

- Special communication libraries for ES & CPS
 - OSEK/VDX COM
 - ...
- Adopted communication libraries for general computing
 - CORBA (Common Object Request Broker Architecture)
 - Message passing interface (MPI)
 - Posix threads (PThreads)
 - OpenMP
 - UPnP, DPWS, JXTA, ...

Frequently not easy to adjust to real-time requirements

What's the bottom line?

- The prevailing technique for writing embedded SW has inherent problems; some of the difficulties of writing embedded SW are not resulting from design constraints, but from the modeling.
- However, there is no ideal modeling technique which fits in all cases.
- The choice of the technique depends on the application.
- Check code generation from non-imperative models
- There is a tradeoff between the power of a modeling technique and its analyzability.
- It may be necessary to combine modeling techniques.
- **In any case, open your eyes & think about the model before you write down your spec! Be aware of pitfalls.**
- You may be forced, to use imperative models, but you can still implement, for example, finite state machines or KPNs in Java.

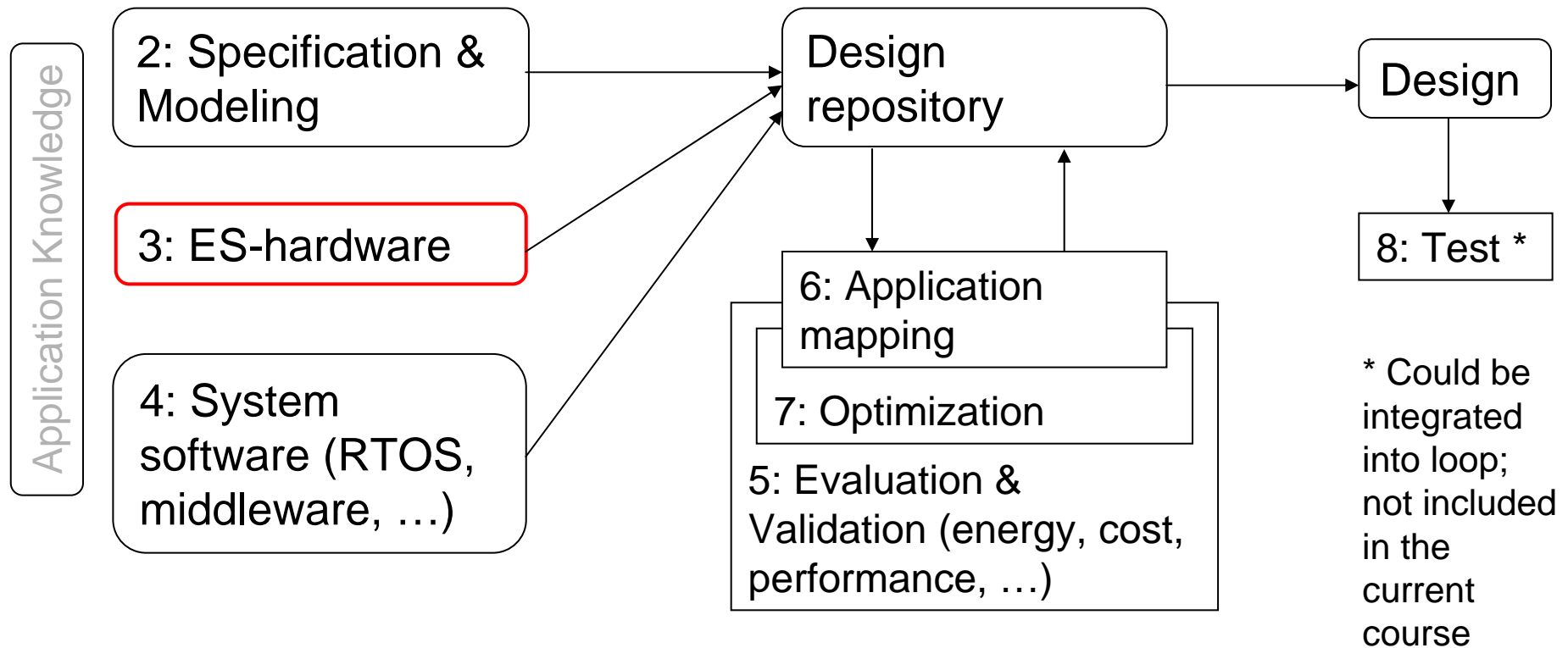


Questions?

Q&A?

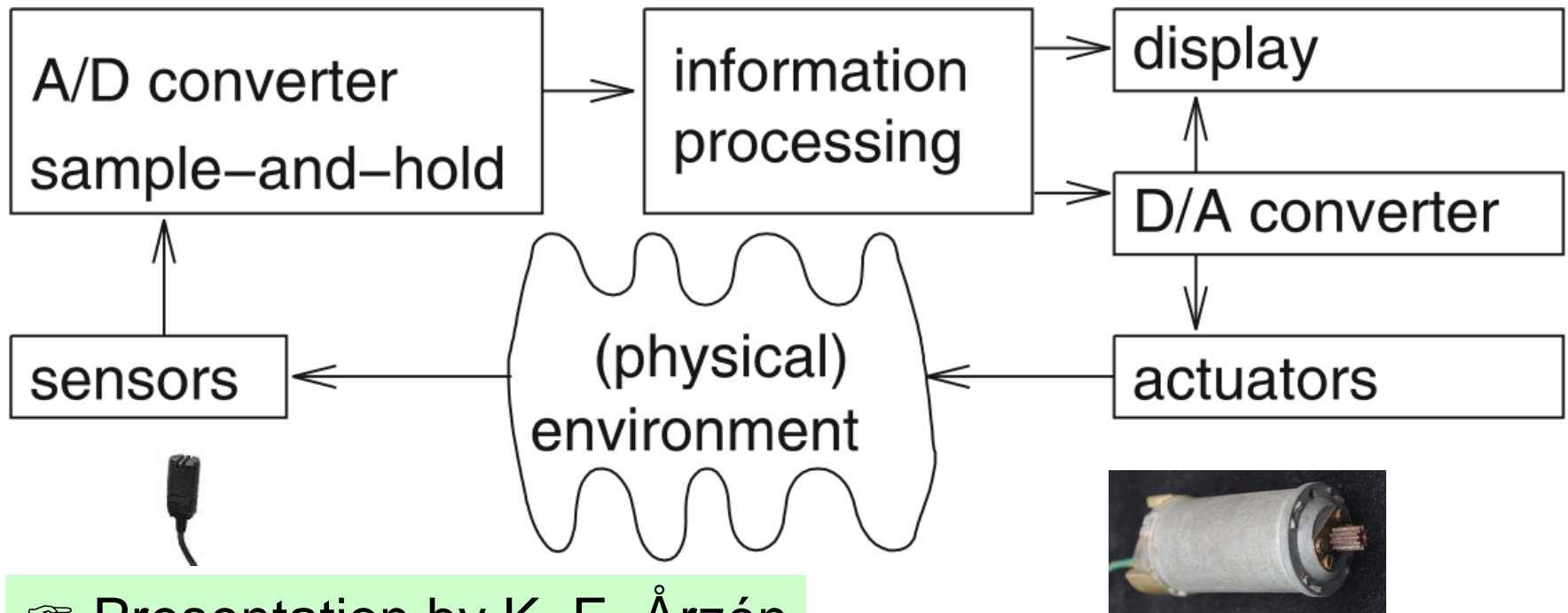


Structure of this course



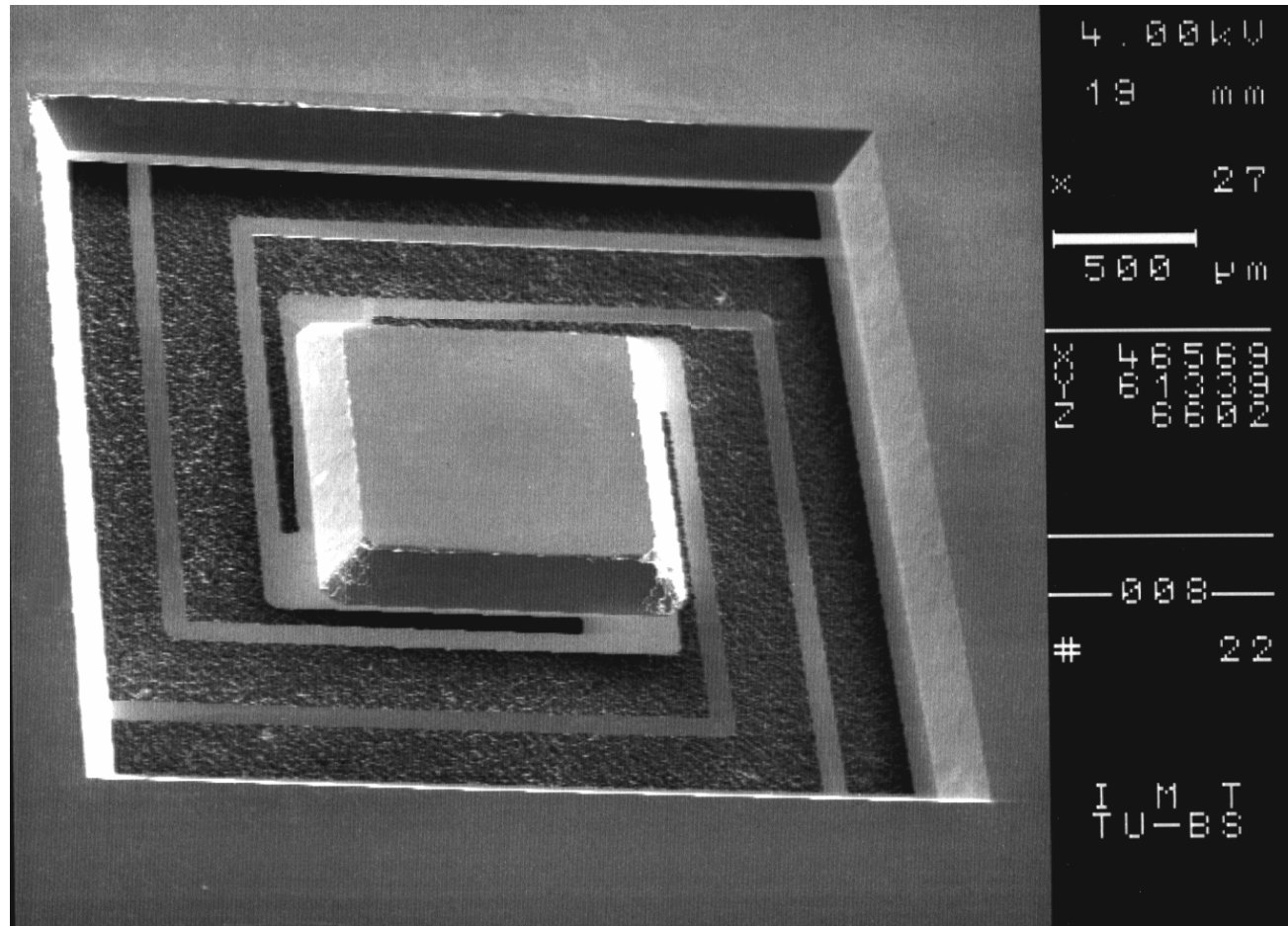
Embedded System Hardware

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



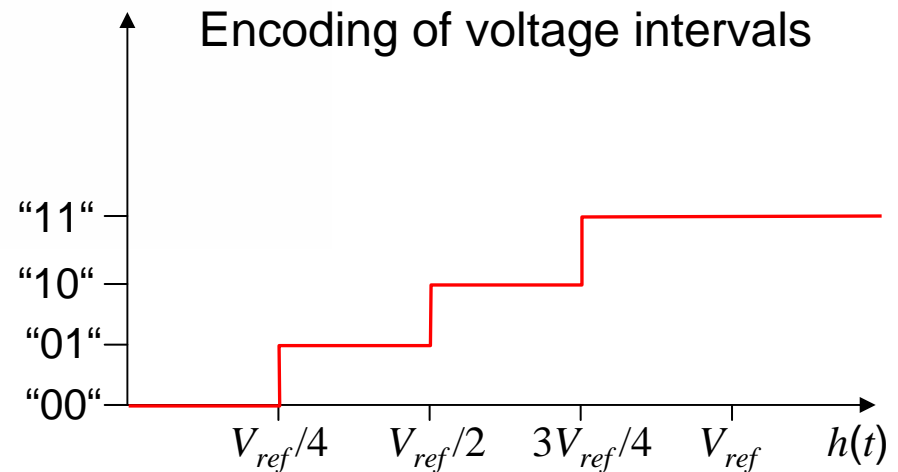
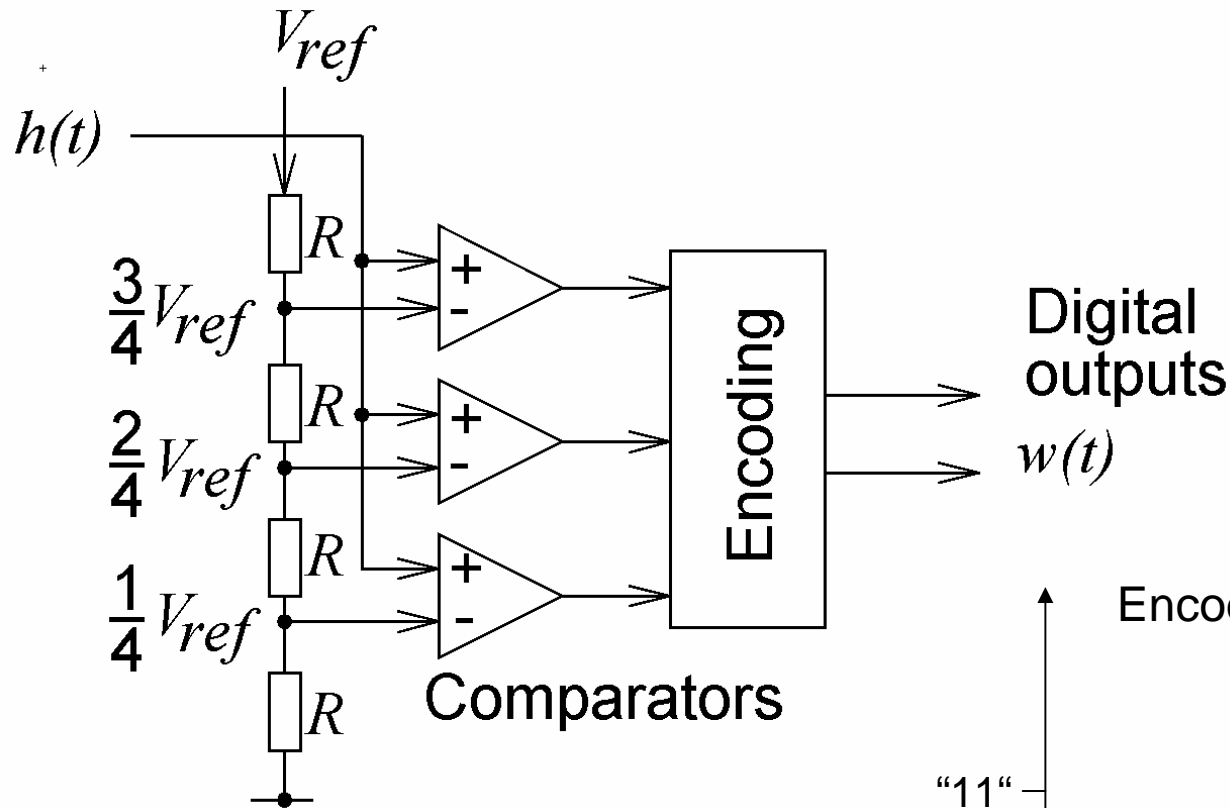
👉 Presentation by K. E. Årzén

Example: Acceleration Sensor



Courtesy & ©: S. Bütgenbach, TU Braunschweig

Assuming $0 \leq h(t) \leq V_{ref}$



Processing units

Need for efficiency (power + energy):

“Power is considered as the most important constraint in embedded systems“

[in: L. Eggermont (ed): Embedded Systems Roadmap 2002, STW]

Why worry about energy and power?

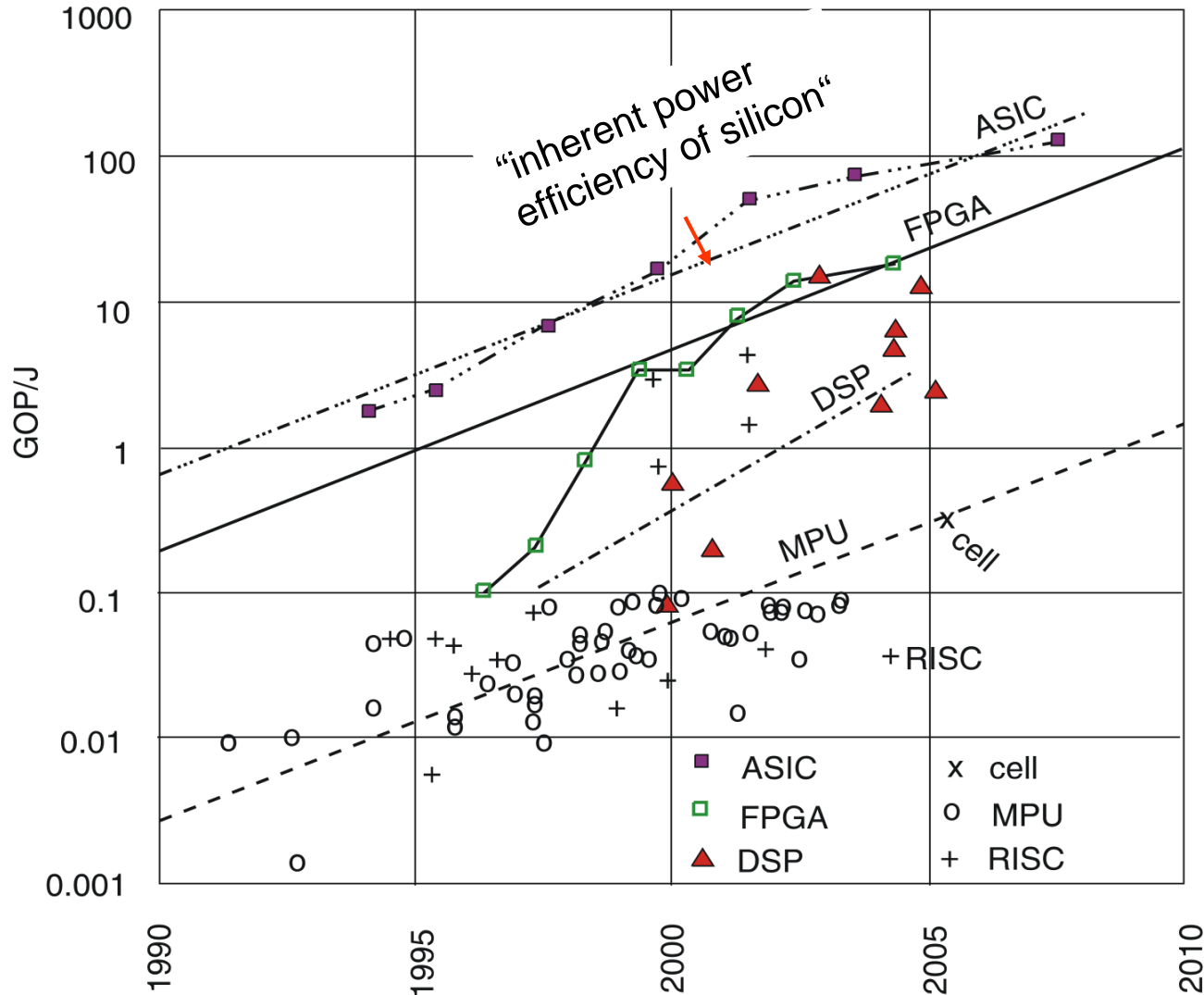


Energy consumption by IT is the key concern of green computing initiatives (**embedded computing leading the way**)



<http://www.esa.int/images/earth,4.jpg>

Importance of Energy Efficiency



© Hugo De Man,
IMEC, Philips, 2007

Fundamentals of dynamic voltage scaling (DVS)

Power consumption of CMOS circuits (ignoring leakage):

$$P = \alpha C_L V_{dd}^2 f \text{ with}$$

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

Delay for CMOS circuits:

$$\tau = k C_L \frac{V_{dd}}{(V_{dd} - V_t)^2} \text{ with}$$

V_t : threshold voltage

($V_t < \text{than } V_{dd}$)

☞ Decreasing V_{dd} reduces P quadratically, while the run-time of algorithms is only linearly increased

Multiprocessor systems-on-a-chip (MPSoCs)

(2) Telephony (W-CDMA)

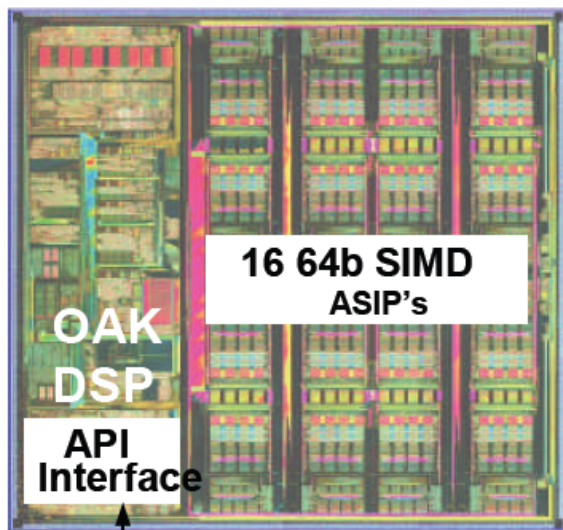


■ Power on
■ Power off

Baseband part	Control	ON
	W-CDMA	ON
	GSM	ON / OFF
Application part	System-domain	ON
	Realtime-domain	OFF
Measured Leakage Current (@ Room Temp, 1.2V)		407 μ A

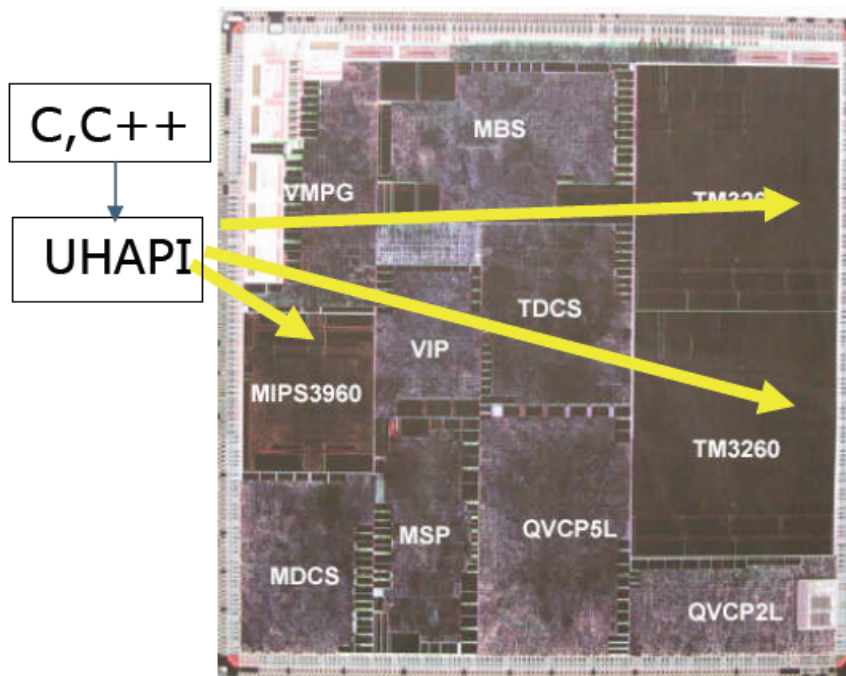
Multiprocessor systems-on-a-chip (MPSoCs) (2)

VIP for car mirrors Infineon



200MHz , 0.76 Watt
100Gops @ 8b
25Gops @ 32b

Nexperia Digital Video Platform NXP



1 MIPS, 2 Trimedia
60 coproc, 250 RAM's
266MHz, 1.5 watt 100 Gops

~50% inherent power efficiency of silicon

Reconfigurable Logic

Custom HW may be too expensive, SW too slow.

Combine the speed of HW with the flexibility of SW

☞ HW with programmable functions and interconnect.

☞ Use of configurable hardware;

common form: field programmable gate arrays (FPGAs)

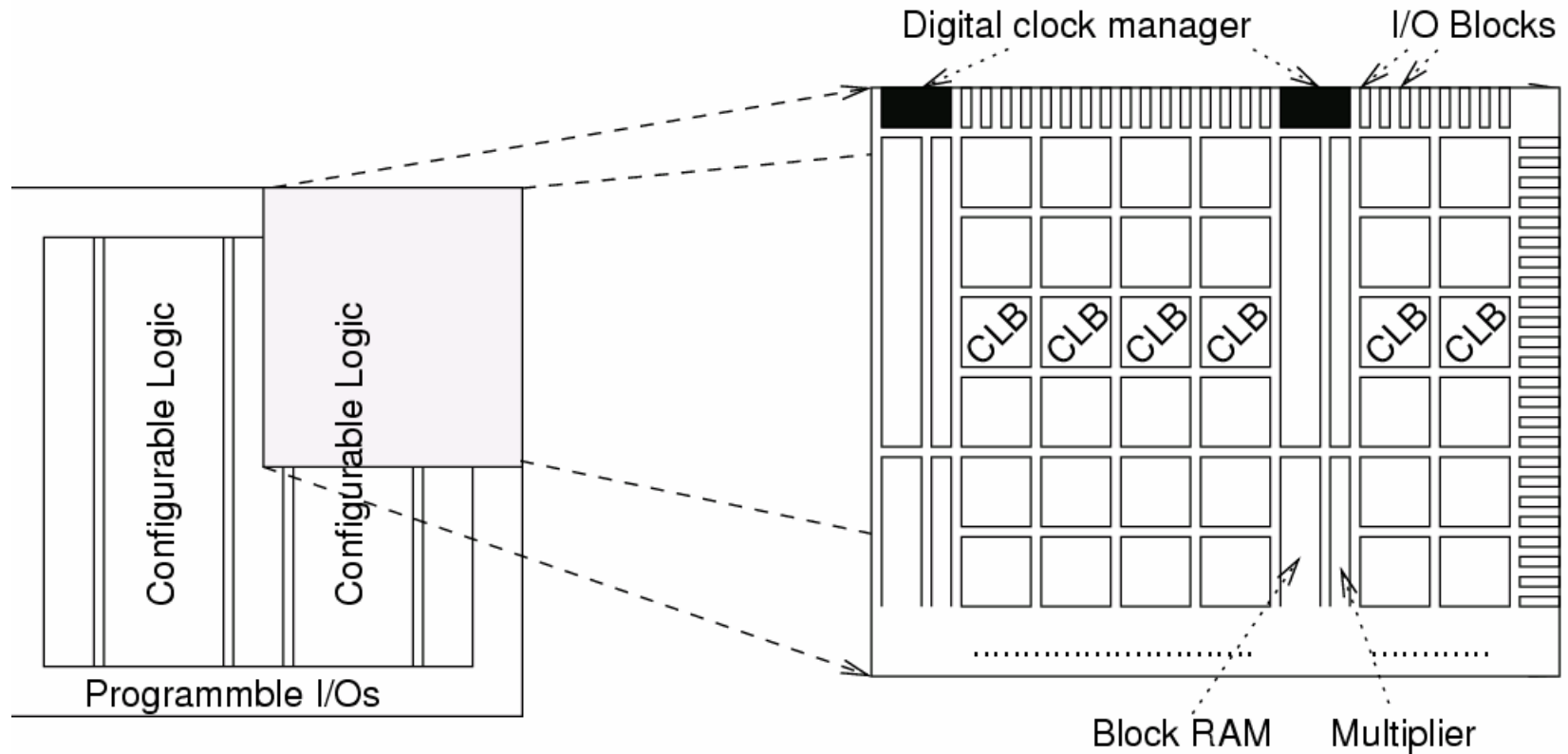
Applications: bit-oriented algorithms like

- encryption,
- fast “object recognition“ (medical and military)
- Adapting mobile phones to different standards.

Very popular devices from

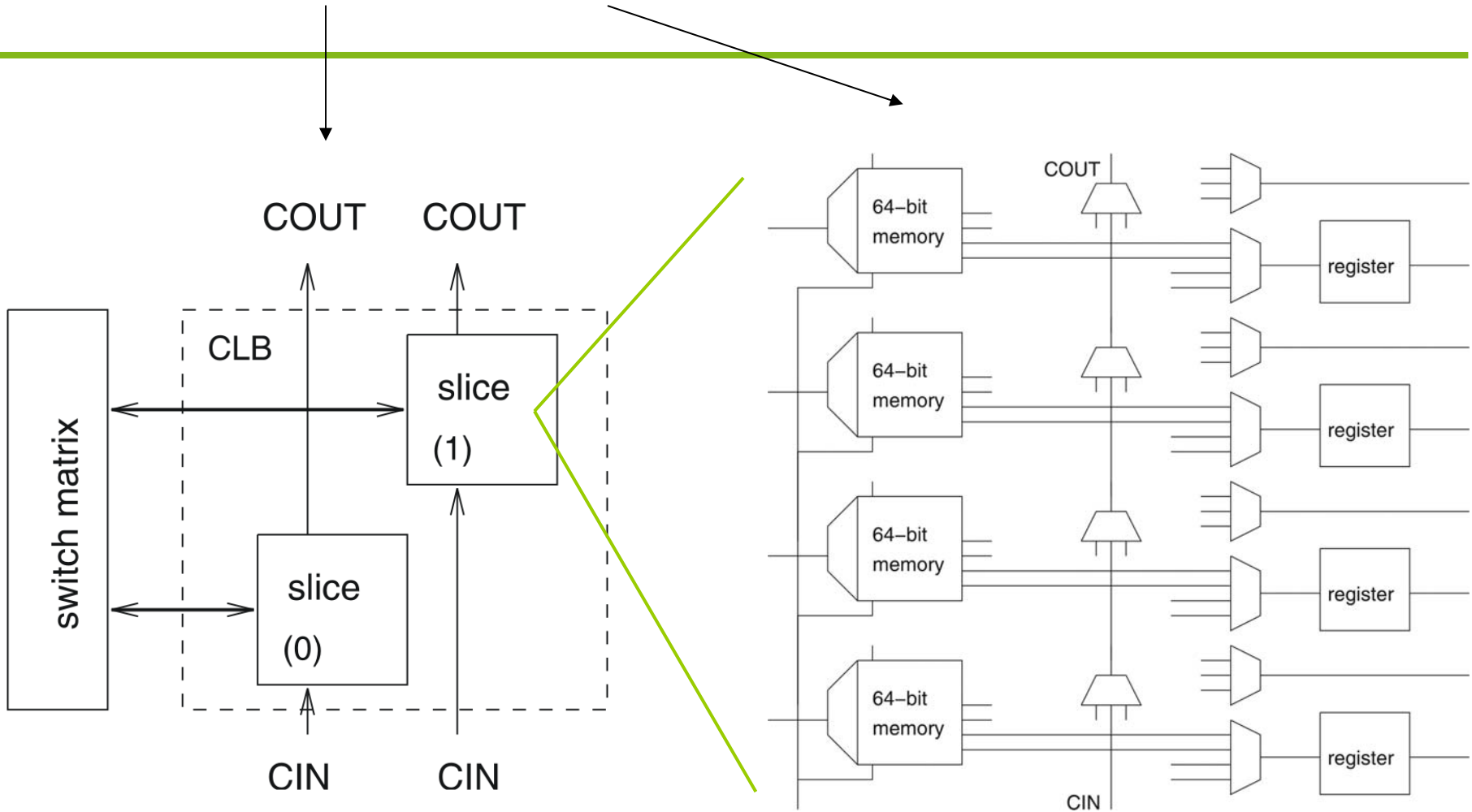
- XILINX (XILINX Vertex II are recent devices)
- Actel, Altera and others

Floor-plan of VIRTEX II FPGAs



More recent: Virtex 5, but no floor-plan found for Virtex 5.

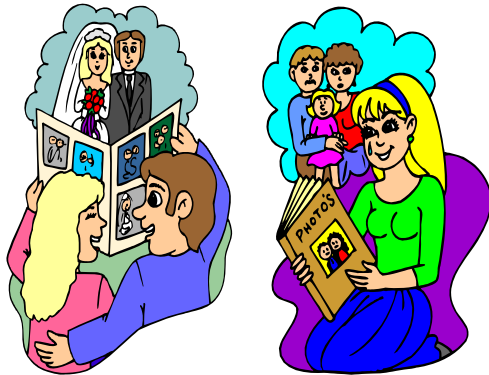
Virtex 5 CLB and slices (simplified)



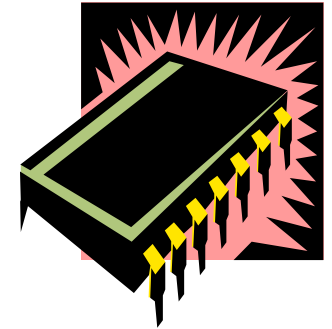
Memories typically used as look-up tables to implement any Boolean function of ≤ 6 variables.

Memory

Memories?



Oops!
Memories!

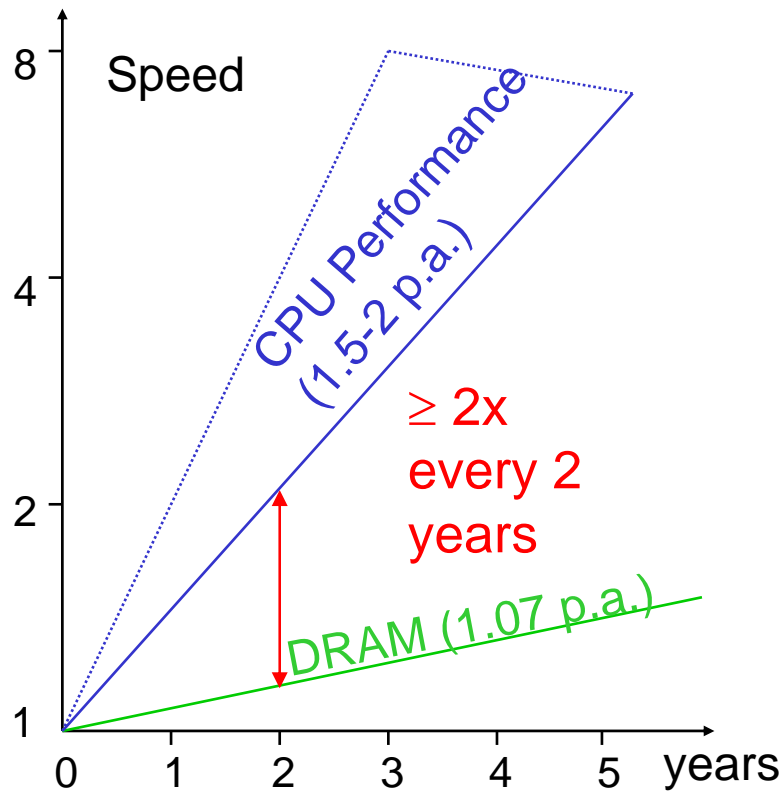


For the memory, efficiency is again a concern:

- speed (latency and throughput); predictable timing
- energy efficiency
- size
- cost
- other attributes (volatile vs. persistent, etc)

Trends for the Speeds

Speed gap between processor and main DRAM increases



Similar problems also for embedded systems & MPSoCs

☞ In the future:

Memory access times \gg processor cycle times

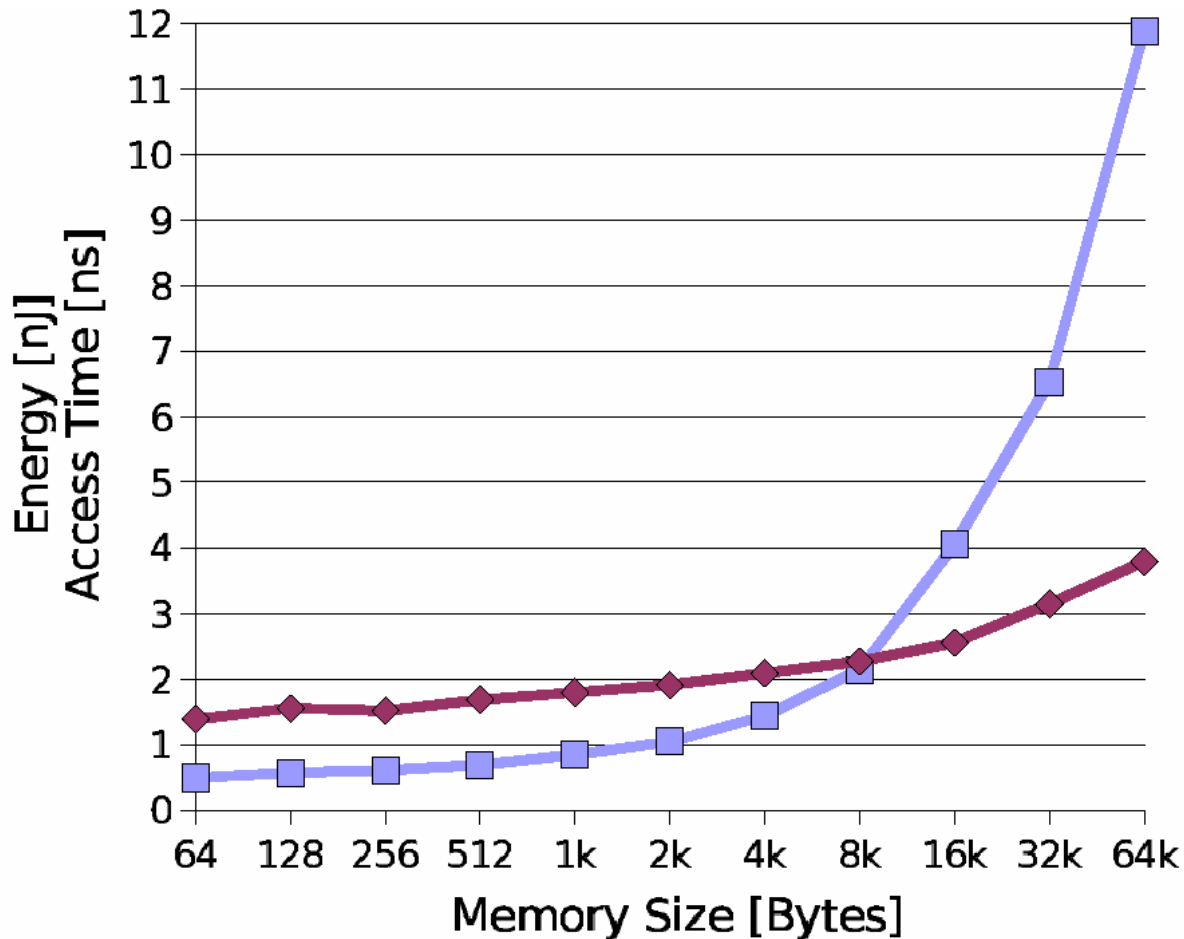
☞ “Memory wall” problem



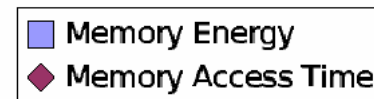
[P. Machanik: Approaches to Addressing the Memory Wall, TR Nov. 2002, U. Brisbane]

Access times and energy consumption increase with the size of the memory

Example (CACTI Model):

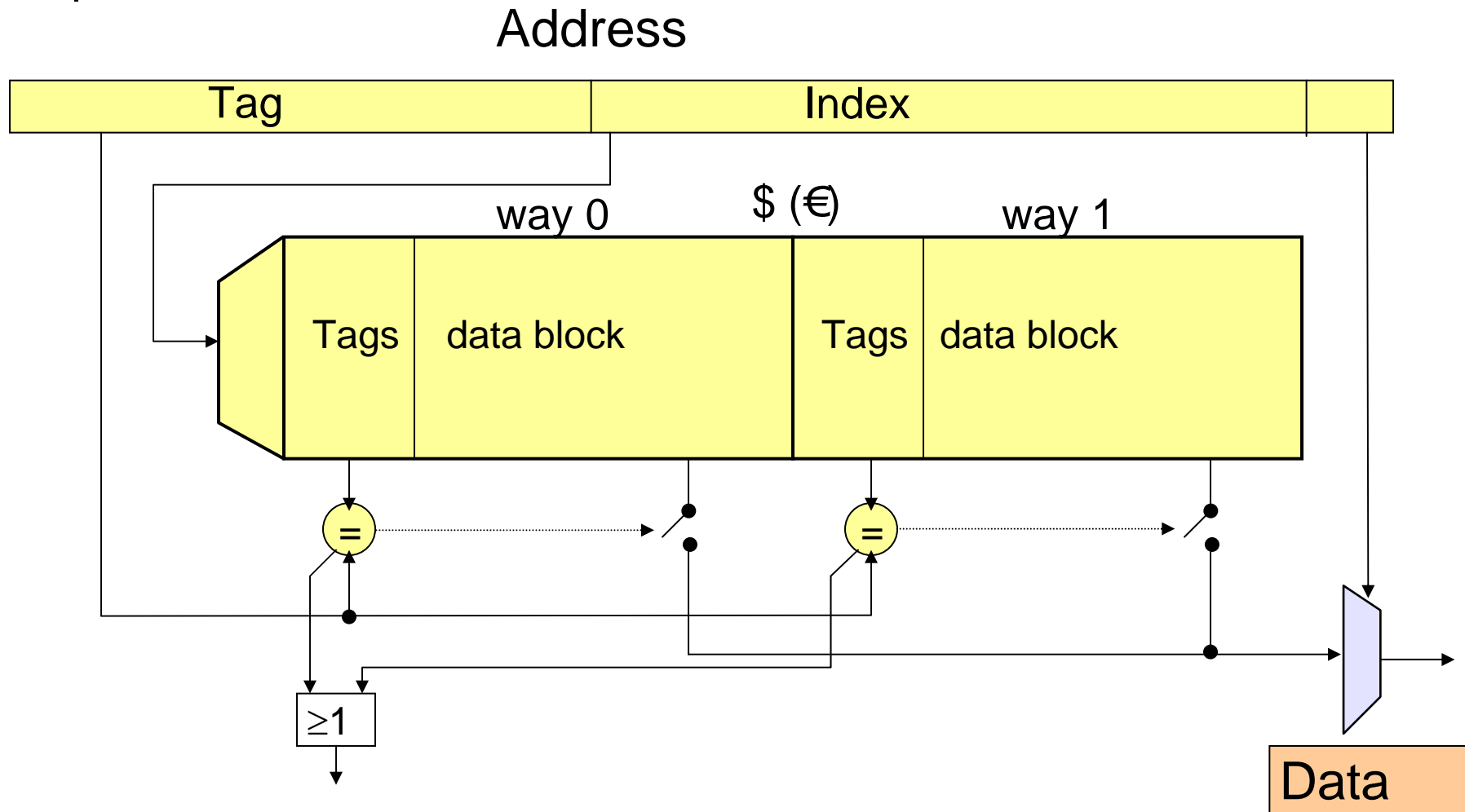


"Currently, the size of some applications is doubling every 10 months" [STMicroelectronics, Medea+ Workshop, Stuttgart, Nov. 2003]



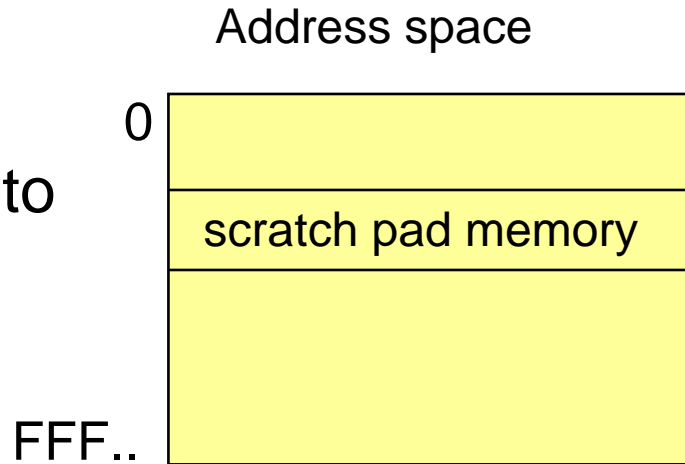
Set-associative cache n -way cache

$|\text{Set}| = 2$

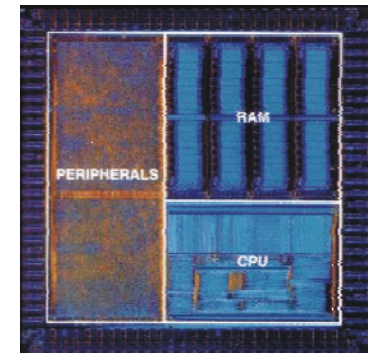


Hierarchical memories using scratch pad memories (SPM)

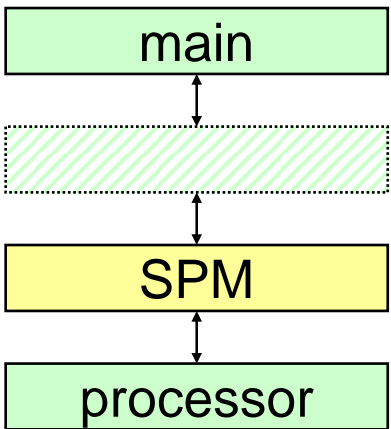
SPM is a small, physically separate memory mapped into the address space



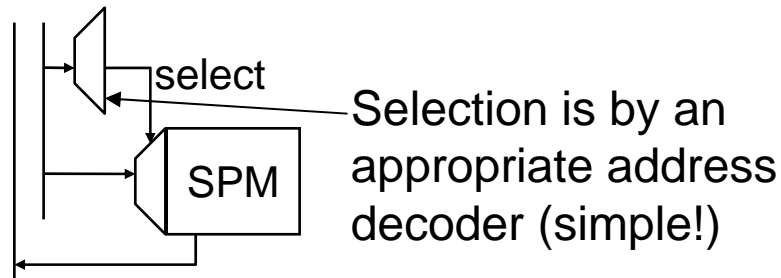
Example



Hierarchy



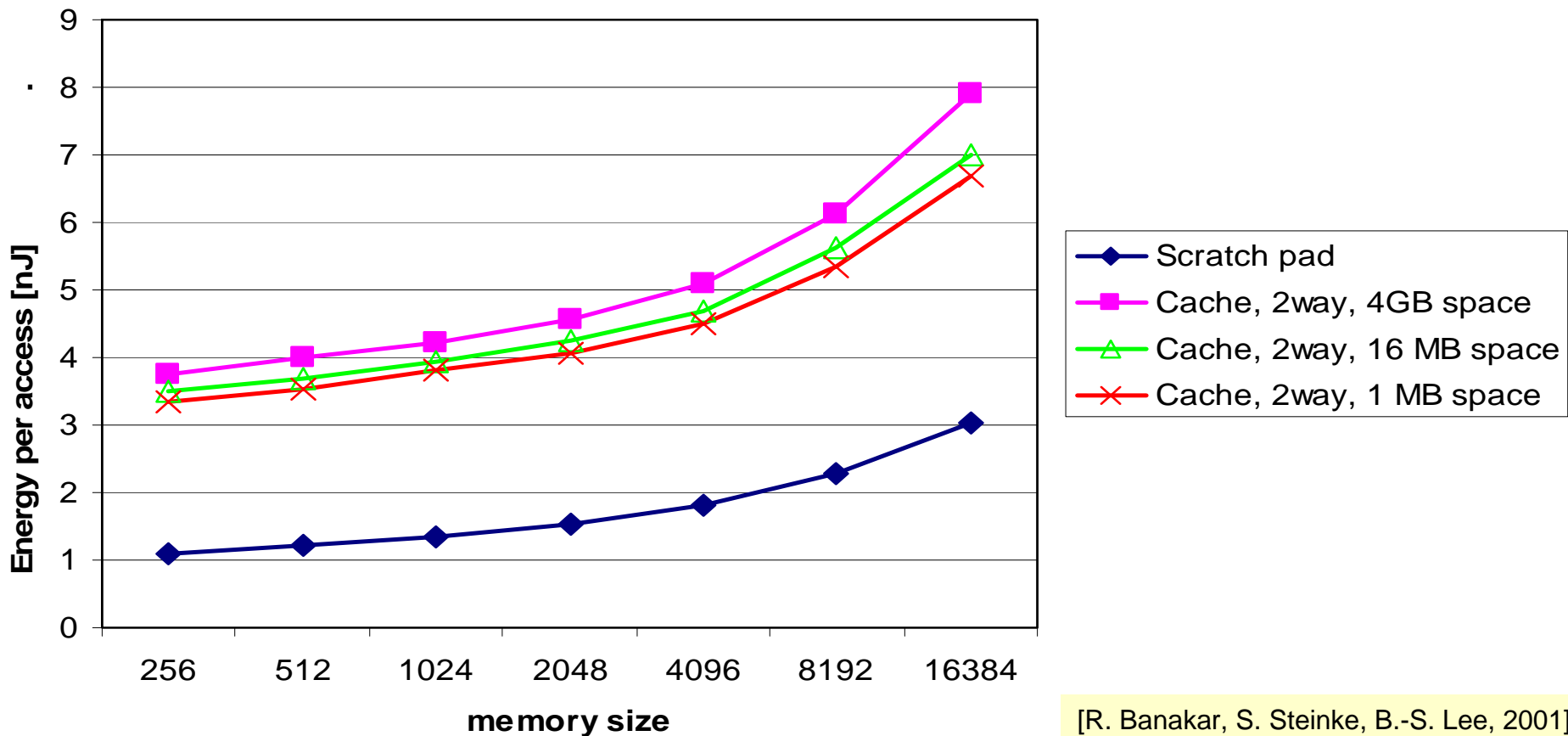
no tag memory



ARM7TDMI cores, well-known for low power consumption

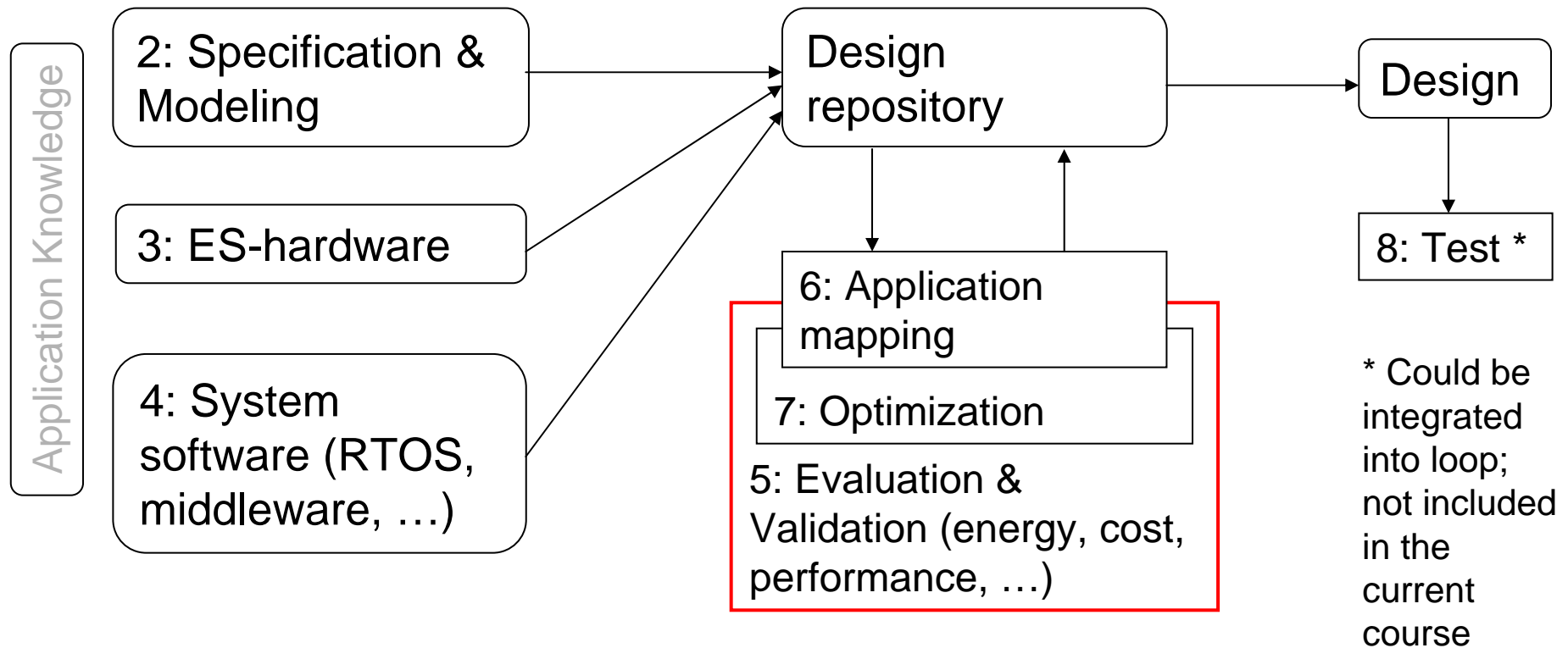
Why not just use a cache ?

2. Energy for parallel access of sets, in comparators, muxes.



[R. Banakar, S. Steinke, B.-S. Lee, 2001]

Structure of this course



Validation and Evaluation

Definition: Validation is the process of checking whether or not a certain (possibly partial) design is appropriate for its purpose, meets all constraints and will perform as expected (yes/no decision).

Definition: Validation with mathematical rigor is called (formal) verification.

Definition: Evaluation is the process of computing quantitative information of some key characteristics of a certain (possibly partial) design.

How to evaluate designs according to multiple criteria?

In practice, many different criteria are relevant for evaluating designs:

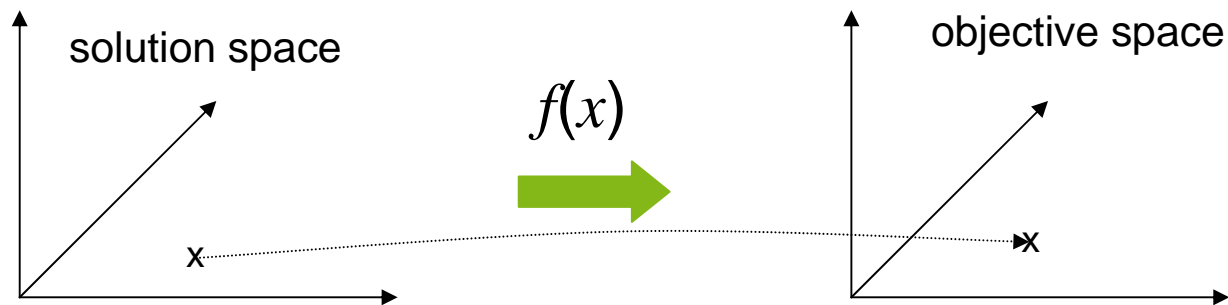
- (average) speed
- worst case speed
- power consumption
- cost
- size
- weight
- radiation hardness
- environmental friendliness

How to compare different designs?

(Some designs are “better” than others)

Definitions

- Let X : m -dimensional **solution space** for the design problem. Example: dimensions correspond to # of processors, size of memories, type and width of busses etc.
- Let F : n -dimensional **objective space** for the design problem. Example: dimensions correspond to speed, cost, power consumption, size, weight, reliability, ...
- Let $f(x) = (f_1(x), \dots, f_n(x))$ where $x \in X$ be an **objective function**. We assume that we are using $f(x)$ for evaluating designs.



Pareto points

- We assume that, for each objective, a total order $<$ and the corresponding order \leq are defined.

- **Definition:**

Vector $u=(u_1, \dots, u_n) \in F$ **dominates** vector $v=(v_1, \dots, v_n) \in F$

\Leftrightarrow

u is “better” than v with respect to one objective and not worse than v with respect to all other objectives:

$$\forall i \in \{1, \dots, n\} : u_i \leq v_i \wedge$$

$$\exists i \in \{1, \dots, n\} : u_i < v_i$$

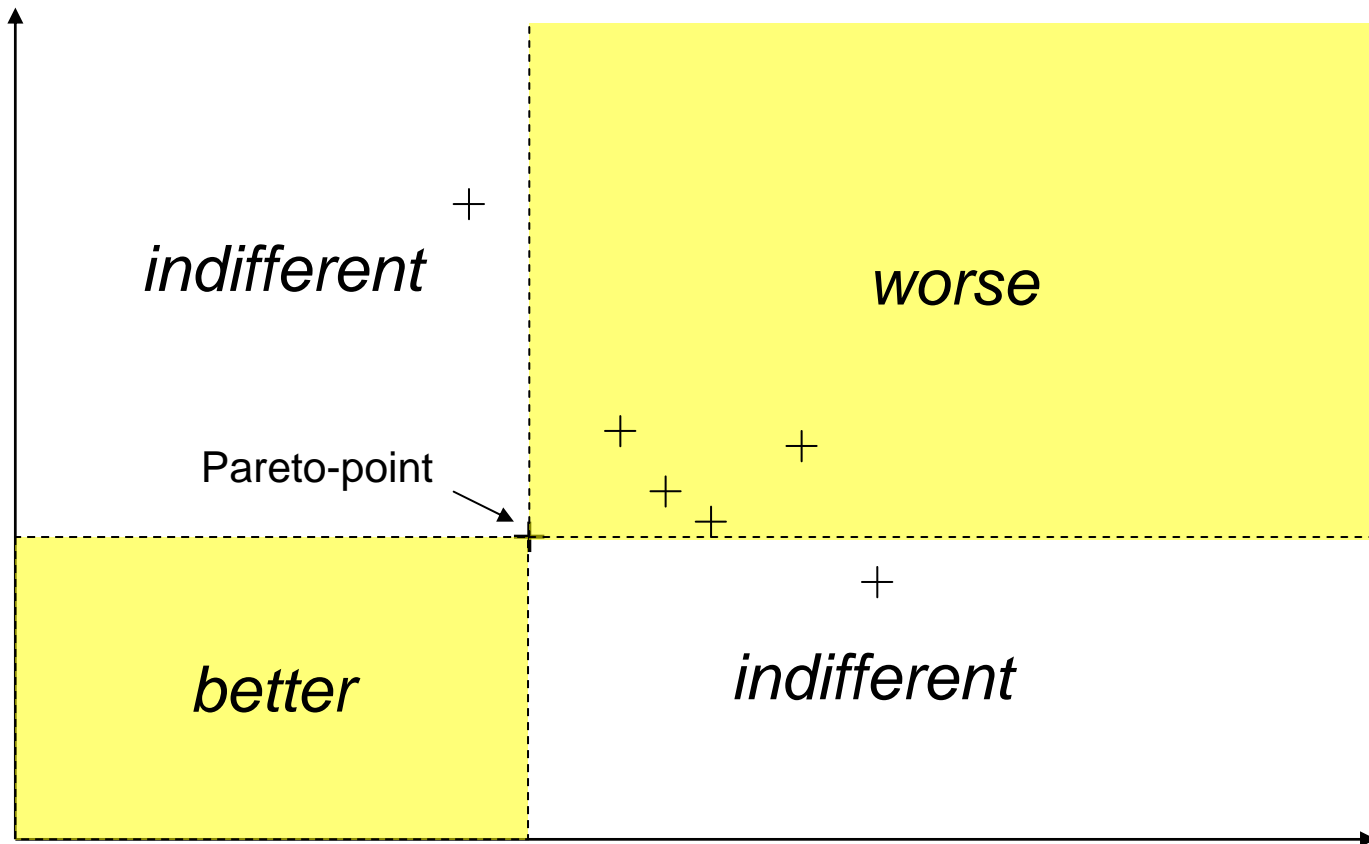
- **Definition:**

Vector $u \in F$ is **indifferent** with respect to vector $v \in F$

\Leftrightarrow neither u dominates v nor v dominates u

Pareto Point

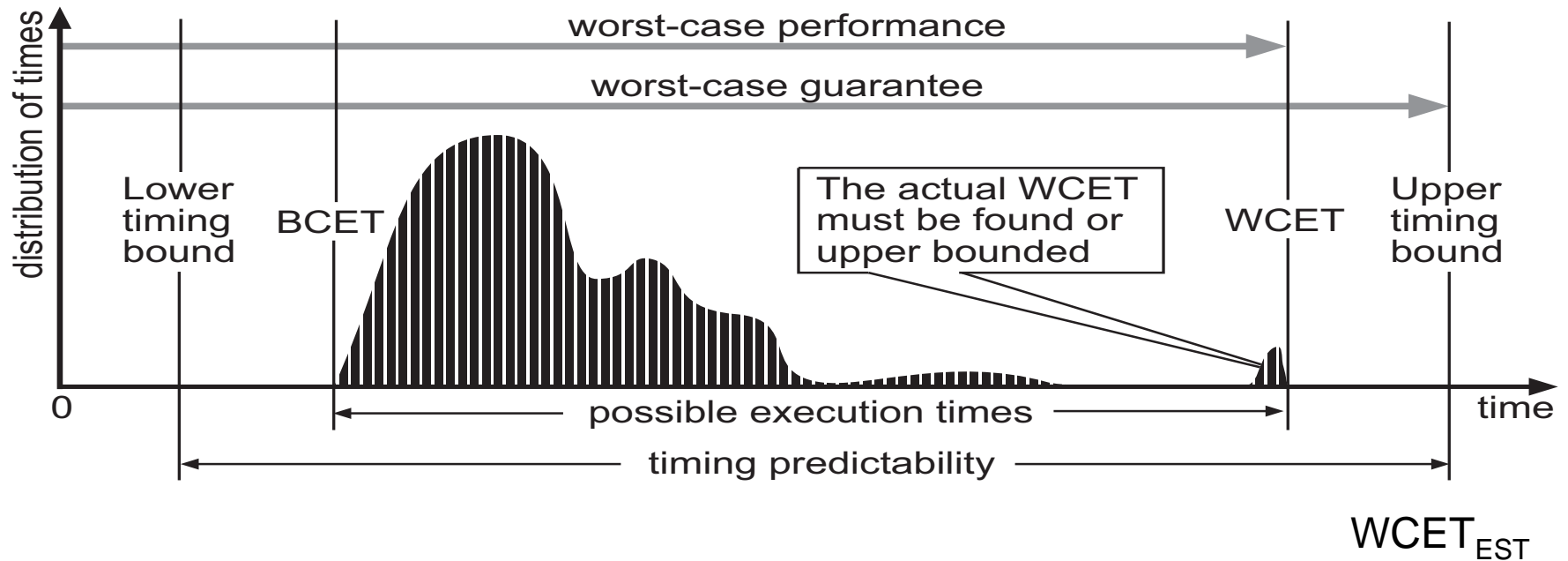
Objective 1
(e.g. energy consumption)



(Assuming *minimization* of objectives)

Objective 2
(e.g. run time)

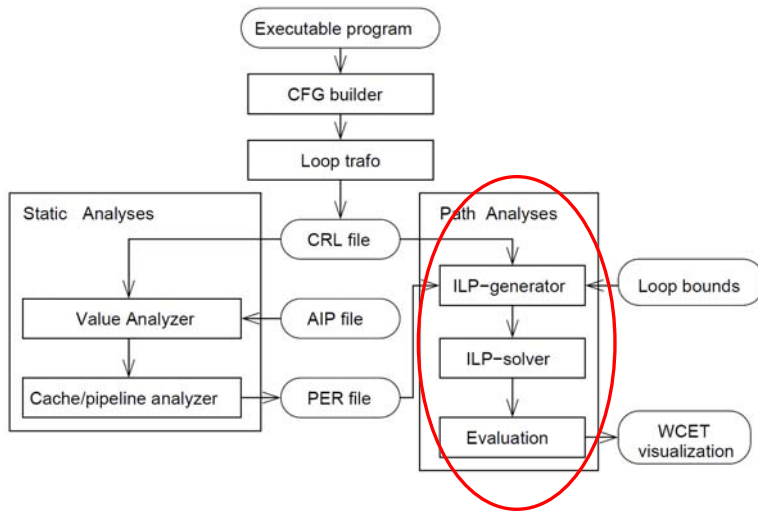
Worst/best case execution times (WCET/BCET)



Requirements on WCET estimates:

- *Safeness*: $WCET \leq WCET_{EST}$!
- *Tightness*: $WCET_{EST} - WCET \rightarrow \text{minimal}$

ILP model



- Objective function reflects execution time as a function of the execution time of blocks. To be **maximized**.
 - Constraints reflect dependencies between blocks.
 - Avoids explicit consideration of all paths
- ☞ Called **implicit path enumeration** technique.

Example (1)

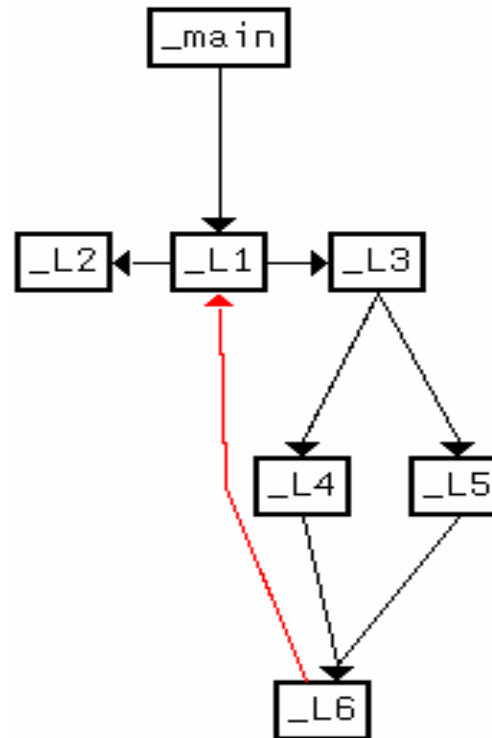
Program

```
int main()
{
    int i, j = 0;

    _Pragma( "loopbound min
            100 max 100" );
    for ( i = 0; i < 100; i++ ) {
        if ( i < 50 )
            j += i;
        else
            j += ( i * 13 ) % 42;
    }

    return j;
}
```

CFG



WCETs of BB (aiT 4 TriCore)

```
_main: 21 cycles
_L1: 27
_L3: 2
_L4: 2
_L5: 20
_L6: 13
_L2: 20
```

Example (2)

ILP

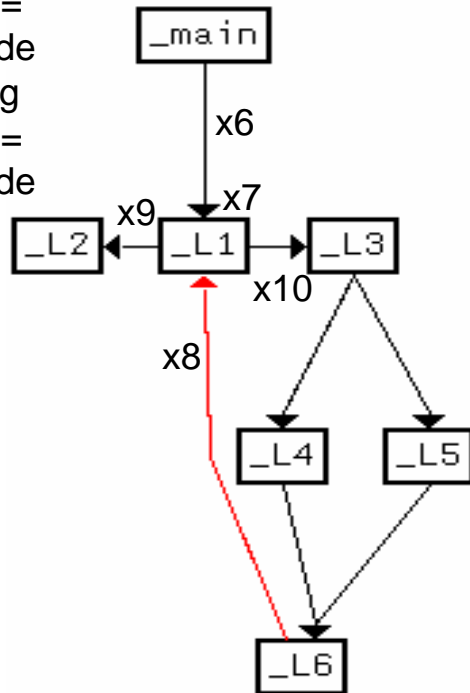
- Virtual start node
- Virtual end node
- Virtual end node per function

Variables:

- 1 variable per node
- 1 variable per edge

Constraints: „Kirchhoff“ equations per node

- Sum of incoming edge variables = flux through node
- Sum of outgoing edge variables = flux through node



_main: 21 cycles
 _L1: 27
 _L3: 2
 _L4: 2
 _L5: 20
 _L6: 13
 _L2: 20

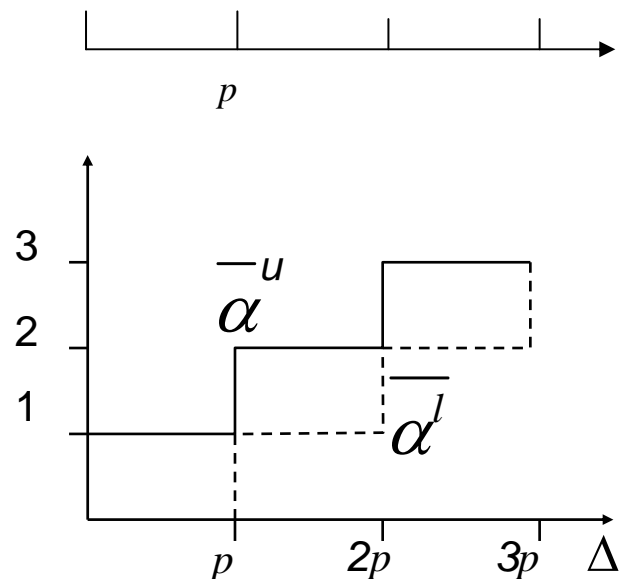
```

/* Objective function = WCET to be maximized*/
21 x2 + 27 x7 + 2 x11 + 2 x14 + 20 x16 + 13 x18 + 20 x19;
/* CFG Start Constraint */ x0 - x4 = 0;
/* CFG Exit Constraint */ x1 - x5 = 0;
/* Constraint for flow entering function main */
x2 - x4 = 0;
/* Constraint for flow leaving exit node of main */
x3 - x5 = 0;
/* Constraint for flow entering exit node of main */
x3 - x20 = 0;
/* Constraint for flow entering main = flow leaving main */
x2 - x3 = 0;
/* Constraint for flow leaving CFG node _main */
x2 - x6 = 0;
/* Constraint for flow entering CFG node _L1 */
x7 - x8 - x6 = 0;
/* Constraint for flow leaving CFG node _L1 */
x7 - x9 - x10 = 0;
/* Constraint for lower loop bound of _L1 */
x7 - 101 x9 >= 0;
/* Constraint for upper loop bound of _L1 */
x7 - 101 x9 <= 0; ....
  
```

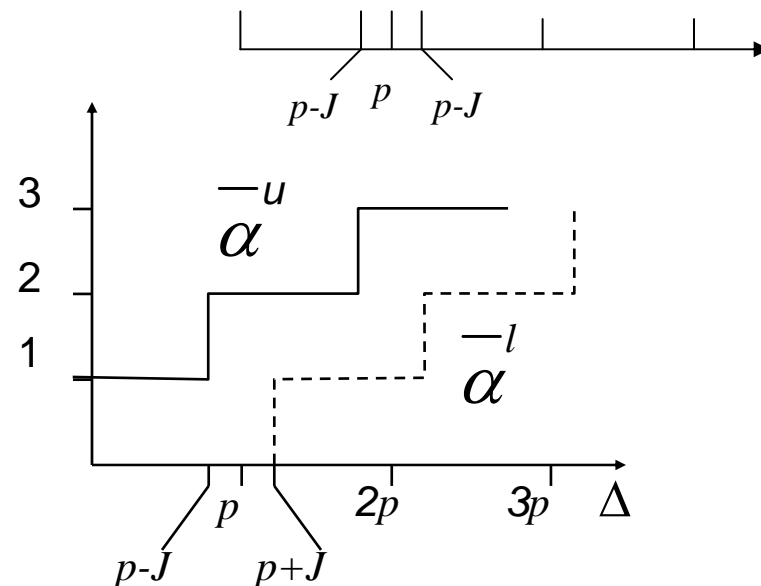
Real-time calculus (RTC)/ Modular performance analysis (MPA)

Thiele et al. (ETHZ): Extended **network calculus**: **Arrival curves** describe the maximum and minimum number of events arriving in some time interval Δ .
Examples:

periodic event stream



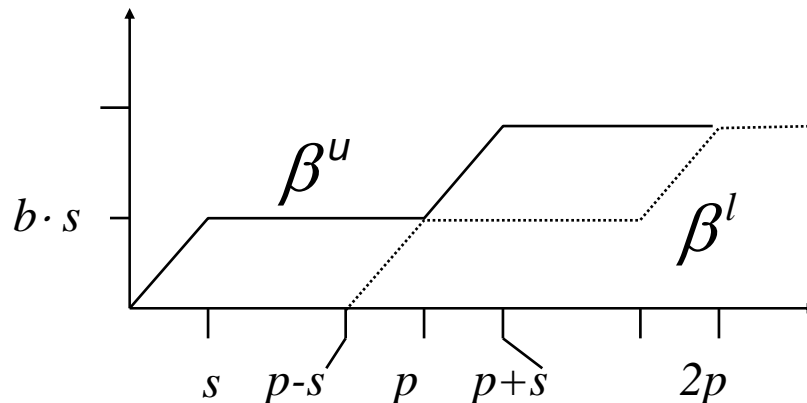
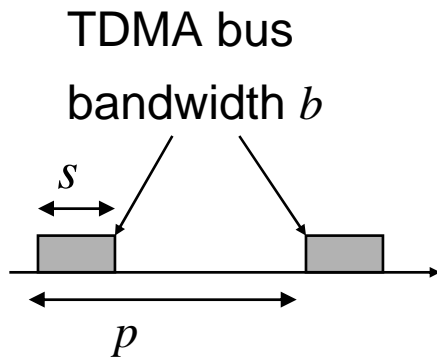
periodic event stream with jitter



RTC/MPA: Service curves

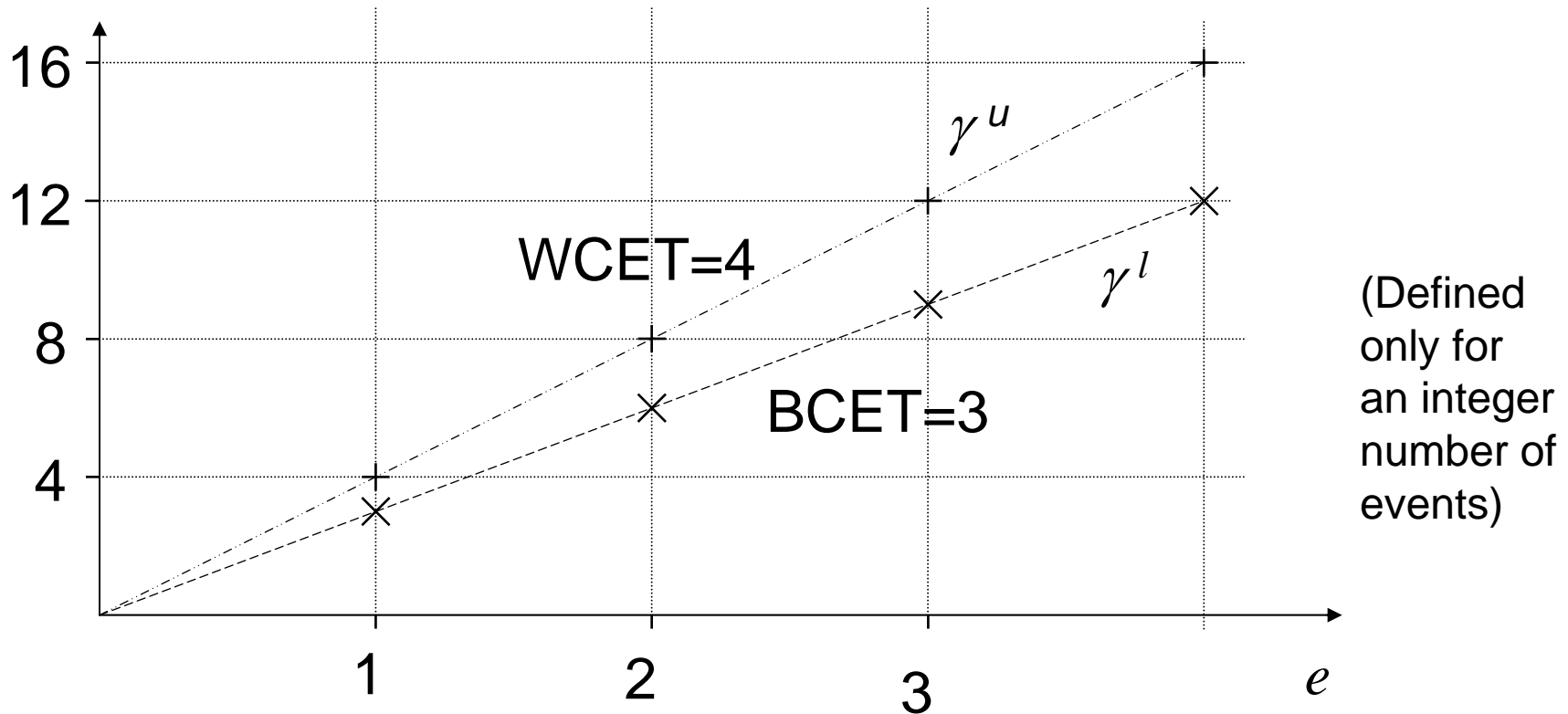
Service curves β^u resp. β^l describe the maximum and minimum service capacity available in some time interval Δ

Example:



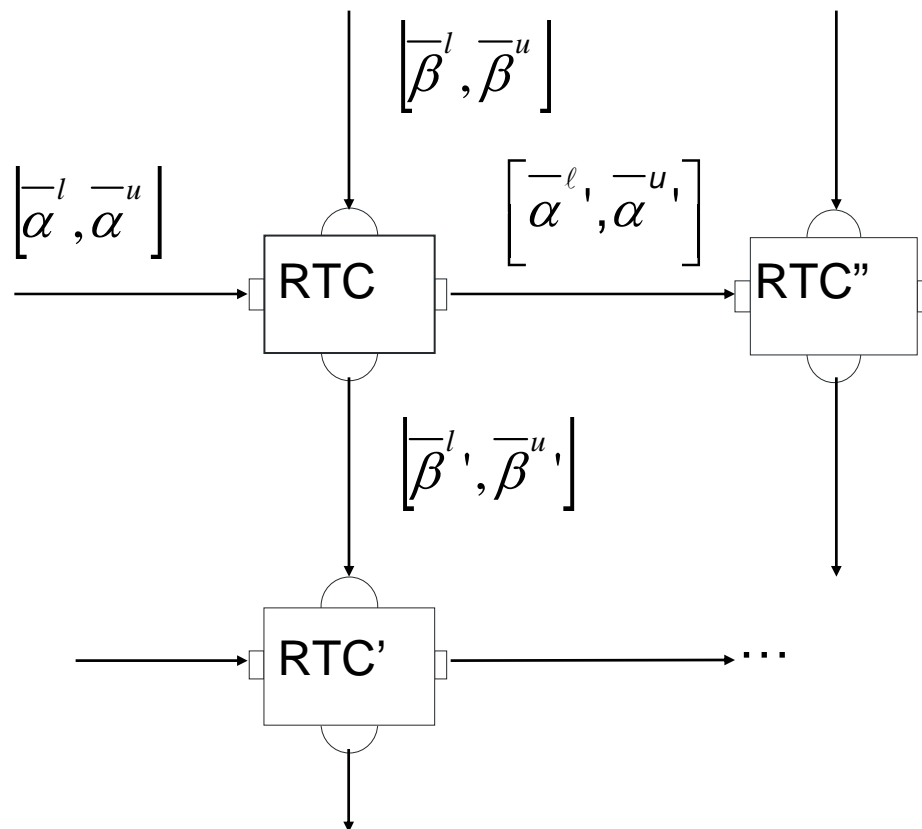
RTC/MPA: Workload characterization


γ^u resp. γ^l describe the maximum and minimum service capacity required as a function of the number e of events. Example:



RTC/MPA: System of real time components

Incoming event streams and available capacity are transformed by real-time components:



Theoretical results allow the computation of properties of outgoing streams 

RTC/MPA: Transformation of arrival and service curves

Resulting arrival curves:

$$\bar{\alpha}^u{}' = \min \left(\left[\left(\bar{\alpha}^u \otimes \bar{\beta}^u \right) \oplus \bar{\beta}^l \right], \bar{\beta}^u \right)$$

$$\bar{\alpha}^l{}' = \min \left(\left[\left(\bar{\alpha}^l \oplus \bar{\beta}^u \right) \otimes \bar{\beta}^l \right], \bar{\beta}^l \right)$$

Remaining service curves:

$$\bar{\beta}^u{}' = \left(\bar{\beta}^u - \bar{\alpha}^l \right) \oplus 0$$

$$\bar{\beta}^l{}' = \left(\bar{\beta}^l - \bar{\alpha}^u \right) \otimes 0$$

Where:

$$(f \otimes g)(t) = \inf_{0 \leq u \leq t} \{f(t-u) + g(u)\} \quad (f \bar{\otimes} g)(t) = \sup_{0 \leq u \leq t} \{f(t-u) + g(u)\}$$

$$(f \oplus g)(t) = \inf_{u \geq 0} \{f(t+u) - g(u)\} \quad (f \bar{\oplus} g)(t) = \sup_{u \geq 0} \{f(t+u) - g(u)\}$$

Summary

ES hardware

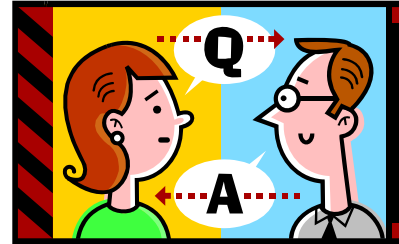
- HW in a loop
- Sensors, discretization
- Processors
- FPGAs
- Memories
- Communication (👉 presentation by L. Almeida)
- Back to the analog world

Evaluation and validation

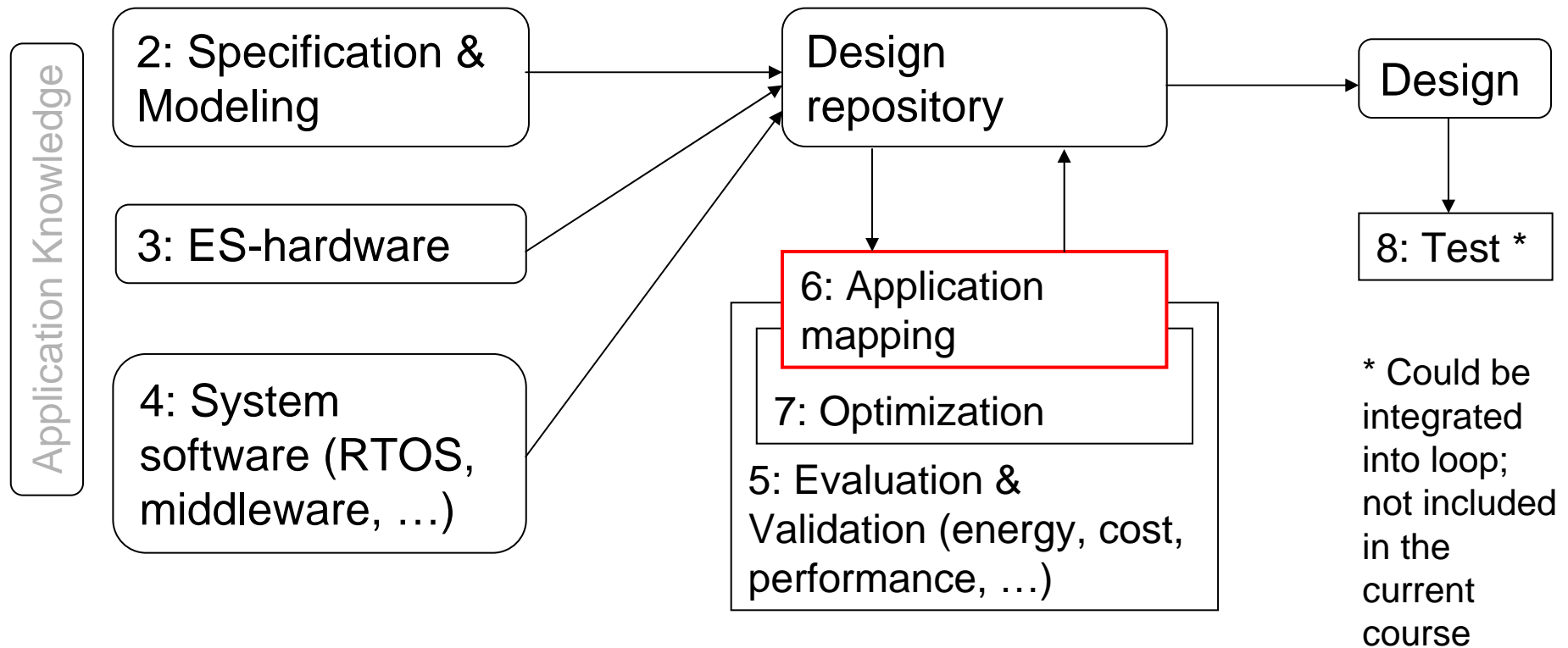
- Multiple objectives, Pareto optimality
- Computation of worst case execution times (WCETs)
- Real-time calculus

Questions?

Q&A?



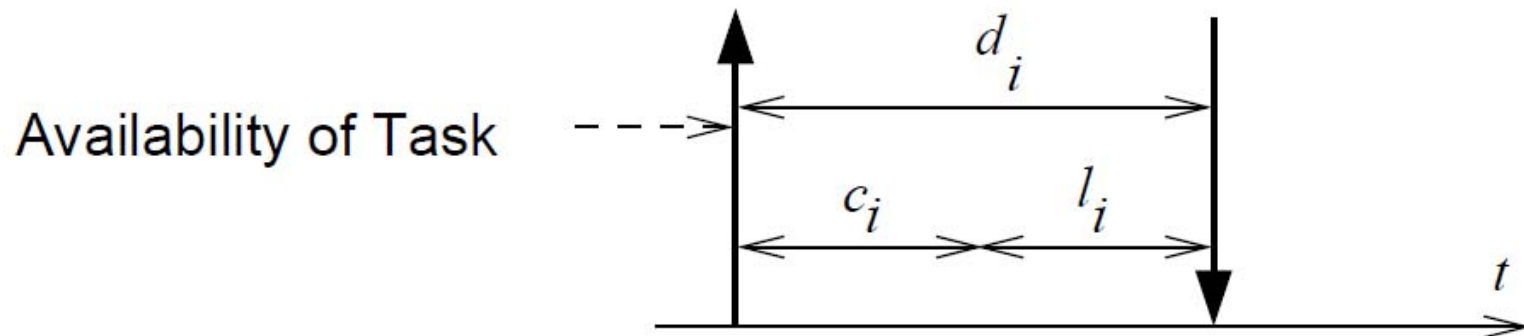
Structure of this course



Aperiodic scheduling; - Scheduling with no precedence constraints -

Let $\{T_i\}$ be a set of tasks. Let:

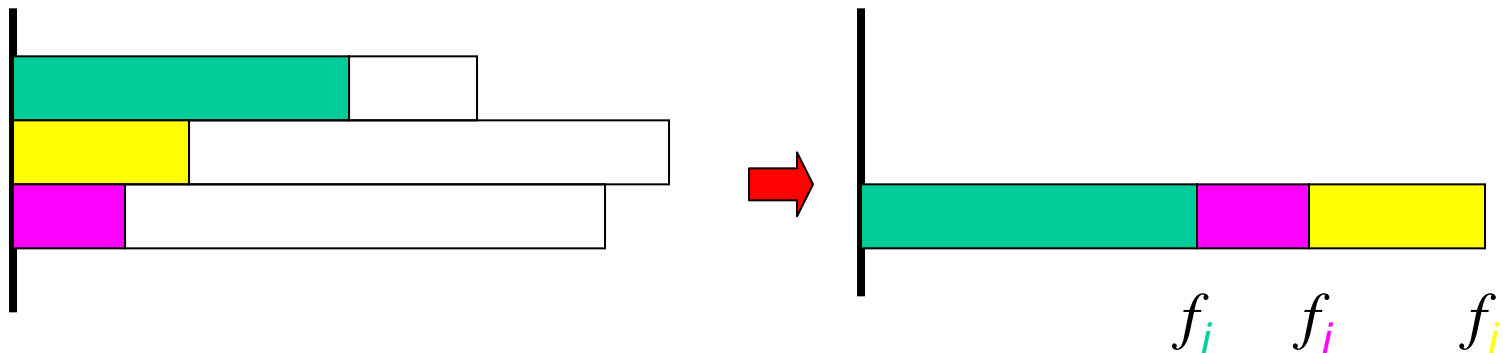
- c_i be the execution time of T_i ,
- d_i be the **deadline interval**, that is, the time between T_i becoming available and the time until which T_i has to finish execution.
- l_i be the **laxity** or **slack**, defined as $l_i = d_i - c_i$
- f_i be the finishing time.



Uniprocessor with equal arrival times

Preemption is useless.

Earliest Due Date (EDD): Execute task with earliest due date (deadline) first.



EDD requires all tasks to be sorted by their (absolute) deadlines. Hence, its complexity is $O(n \log(n))$.

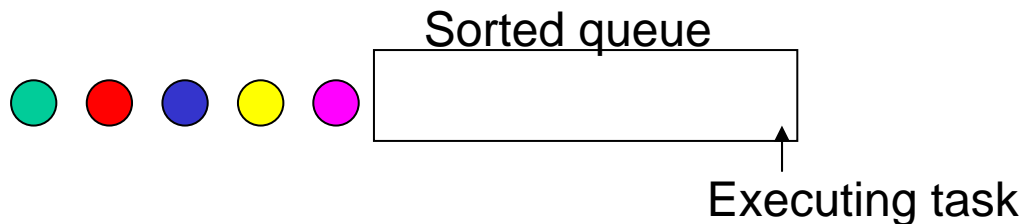
Earliest Deadline First (EDF)

- Algorithm -

Earliest deadline first (EDF) algorithm:

- Each time a new ready task arrives:
- It is inserted into a queue of ready tasks, sorted by their **absolute** deadlines. Task at head of queue is executed.
- If a newly arrived task is inserted at the head of the queue, the currently executing task is preempted.

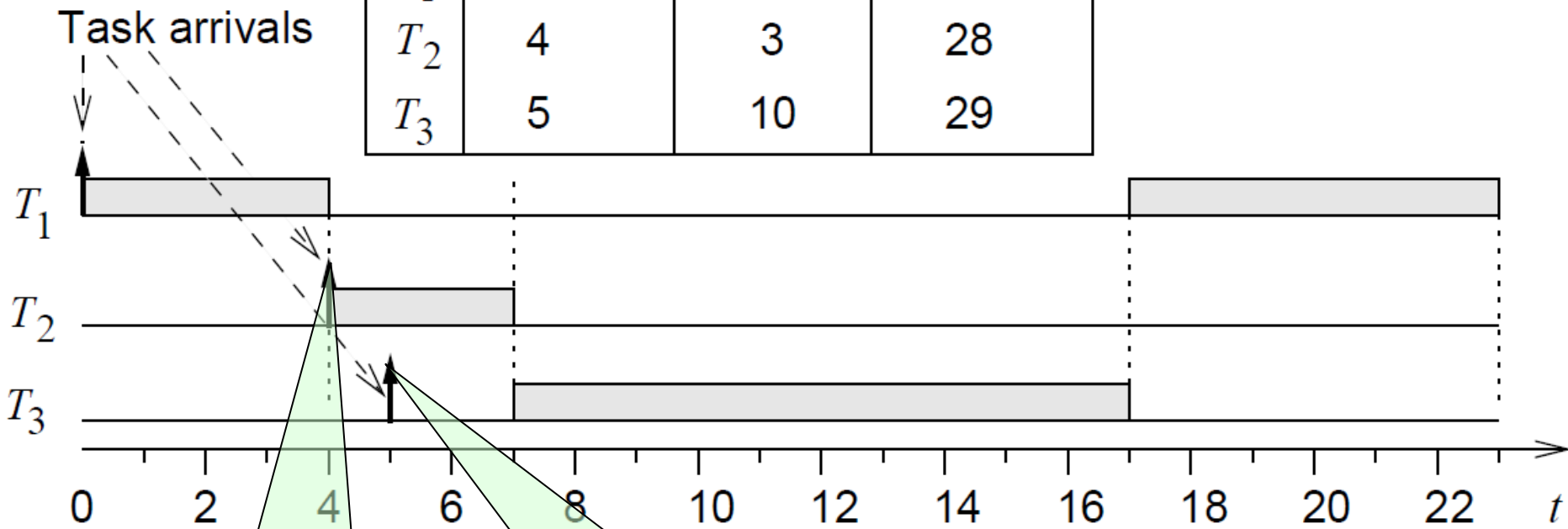
Straightforward approach with sorted lists (full comparison with existing tasks for each arriving task) requires run-time $O(n^2)$; (less with binary search or bucket arrays).



Earliest Deadline First (EDF)

- Example -

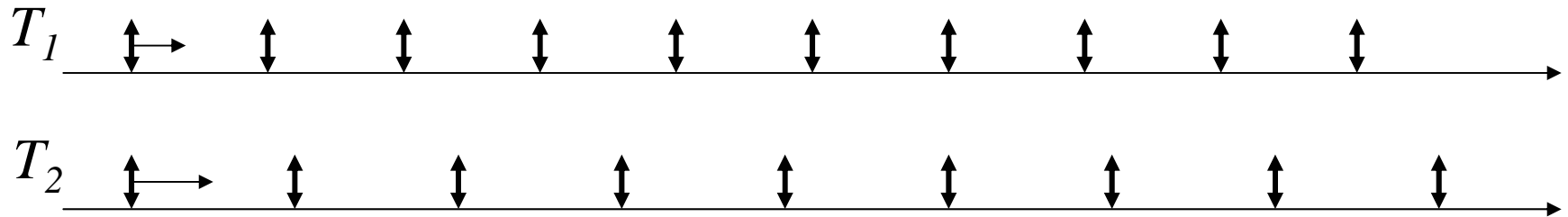
	arrival	duration	deadline
T_1	0	10	33
T_2	4	3	28
T_3	5	10	29



Earlier deadline
preemption

Later deadline
no preemption

Periodic scheduling



Each execution instance of a task is called a **job**.

Notion of optimality for aperiodic scheduling does not make sense for periodic scheduling.

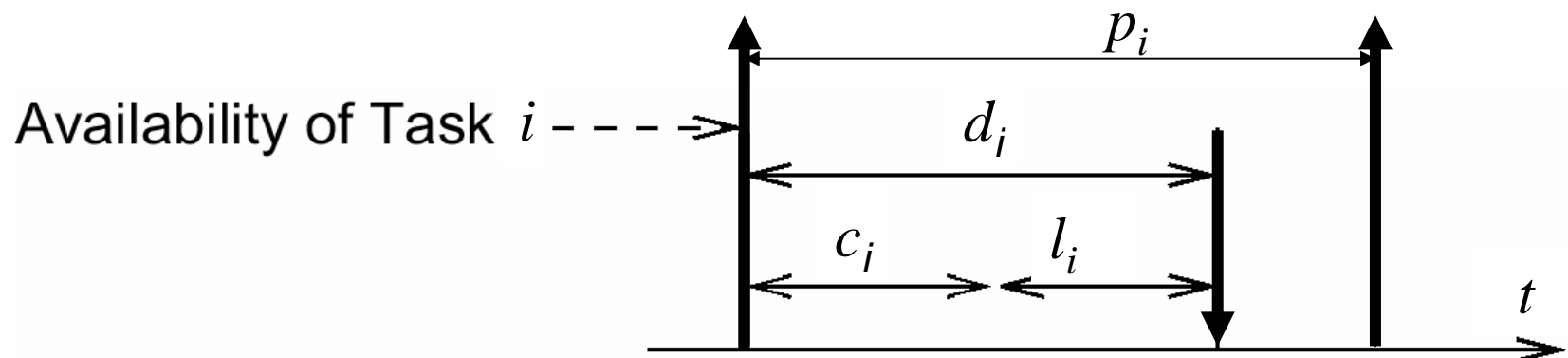
For periodic scheduling, the best that we can do is to design an algorithm which will always find a schedule if one exists.

☞ A scheduler is defined to be **optimal** iff it will find a schedule if one exists.

Periodic scheduling: Scheduling with no precedence constraints

Let $\{T_i\}$ be a set of tasks. Let:

- p_i be the period of task T_i ,
- c_i be the execution time of T_i ,
- d_i be the **deadline interval**, that is, the time between T_i becoming available and the time until which T_i has to finish execution.
- l_i be the **laxity** or **slack**, defined as $l_i = d_i - c_i$
- f_i be the finishing time.



Independent tasks:

Rate monotonic (RM) scheduling

Most well-known technique for scheduling independent periodic tasks [Liu, 1973].

Assumptions:

- All tasks that have hard deadlines are periodic.
- All tasks are independent.
- $d_i = p_i$, for all tasks.
- c_i is constant and is known for all tasks.
- The time required for context switching is negligible.
- For a single processor and for n tasks, the following equation holds for the average utilization μ :

$$\mu = \sum_{i=1}^n \frac{c_i}{p_i} \leq n(2^{1/n} - 1)$$



Rate monotonic (RM) scheduling

- The policy -

RM policy: The priority of a task is a monotonically decreasing function of its period.



At any time, a highest priority task among all those that are ready for execution is allocated.

Theorem: If all RM assumptions are met, schedulability is guaranteed.

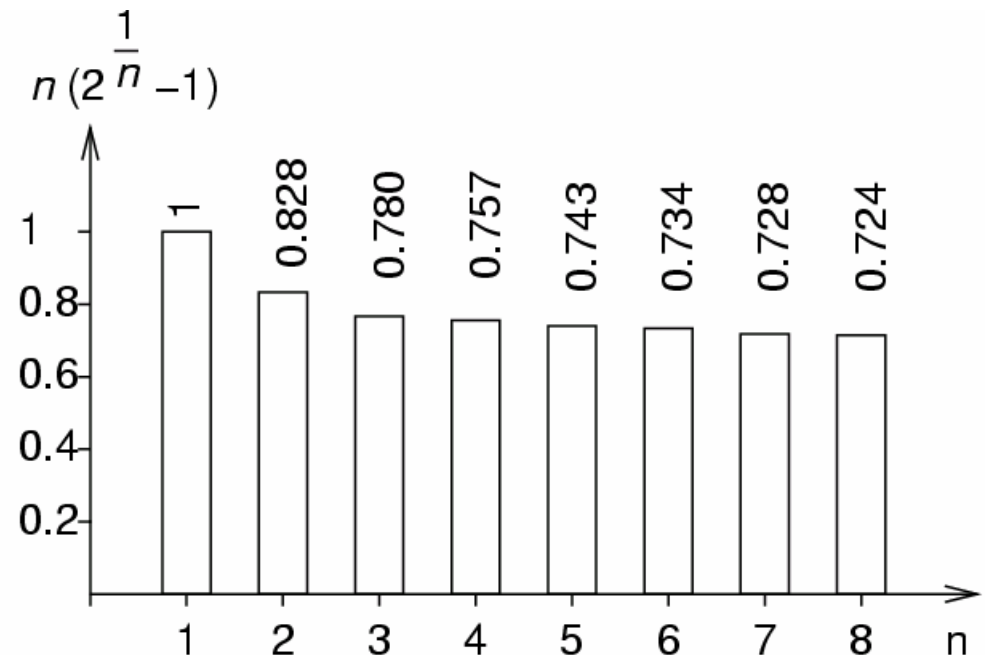


Maximum utilization for guaranteed schedulability

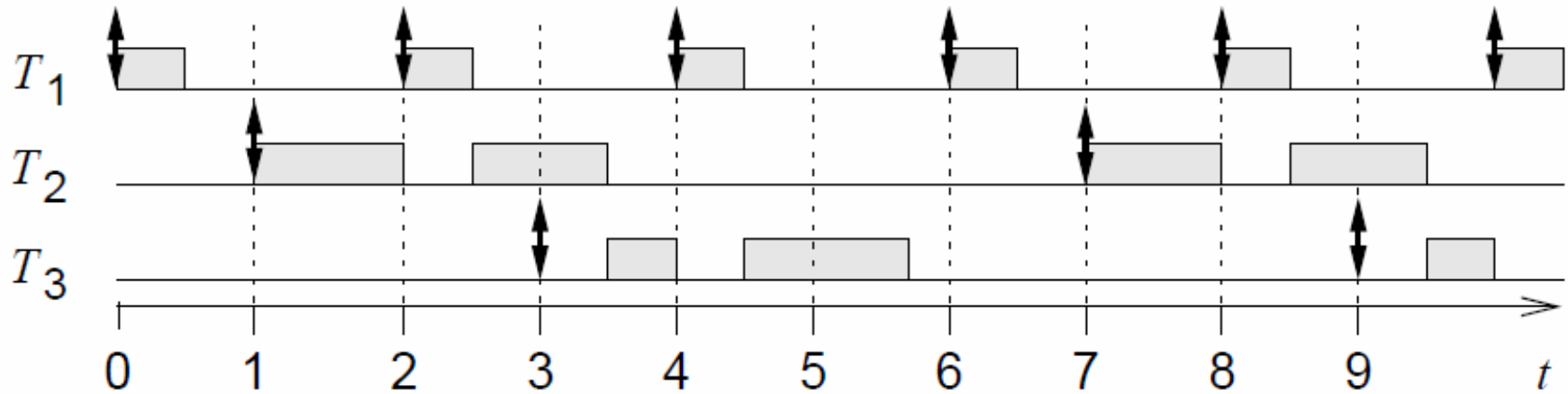
Maximum utilization as a function of the number of tasks:

$$\mu = \sum_{i=1}^n \frac{c_i}{p_i} \leq n(2^{1/n} - 1)$$

$$\lim_{n \rightarrow \infty} (n(2^{1/n} - 1)) = \ln(2)$$



Example of RM-generated schedule



T_1 preempts T_2 and T_3 .

T_2 and T_3 do not preempt each other.

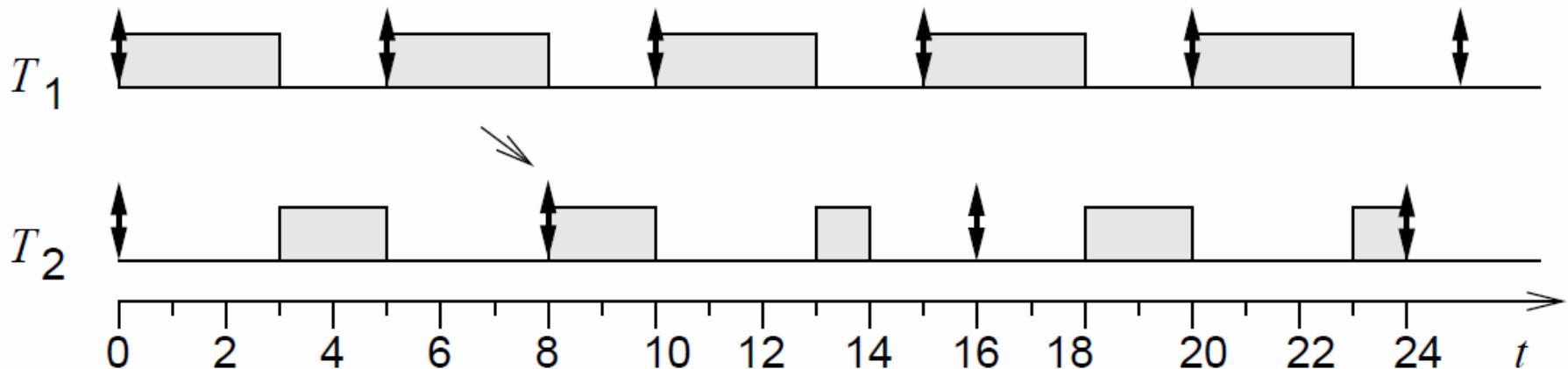
Failing RMS

Task 1: period 5, execution time 3

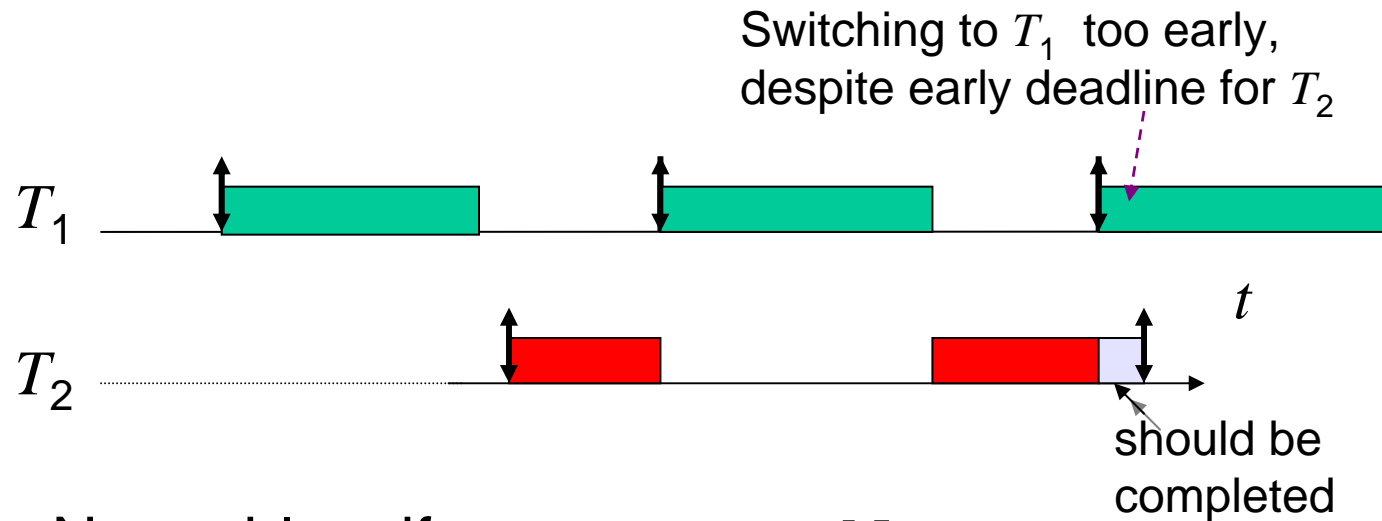
Task 2: period 8, execution time 3

$$\mu = 3/5 + 3/8 = 24/40 + 15/40 = 39/40 \approx 0.975$$

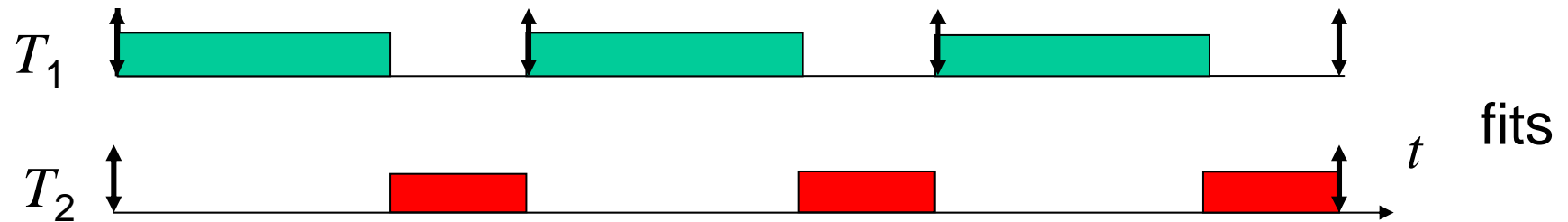
$$2(2^{1/2} - 1) \approx 0.828$$



Intuitively: Why does RM fail ?

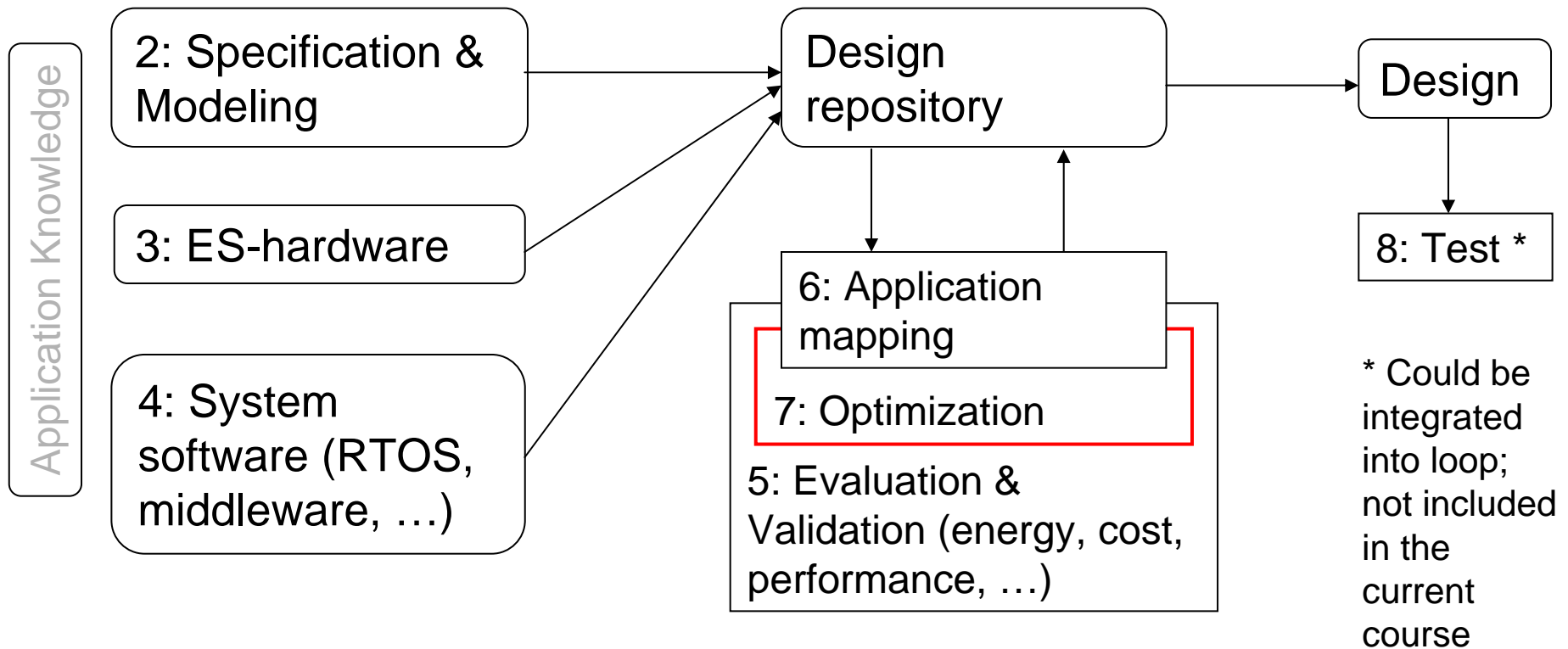


No problem if $p_2 = m p_1$, $m \in \mathbb{N}$:

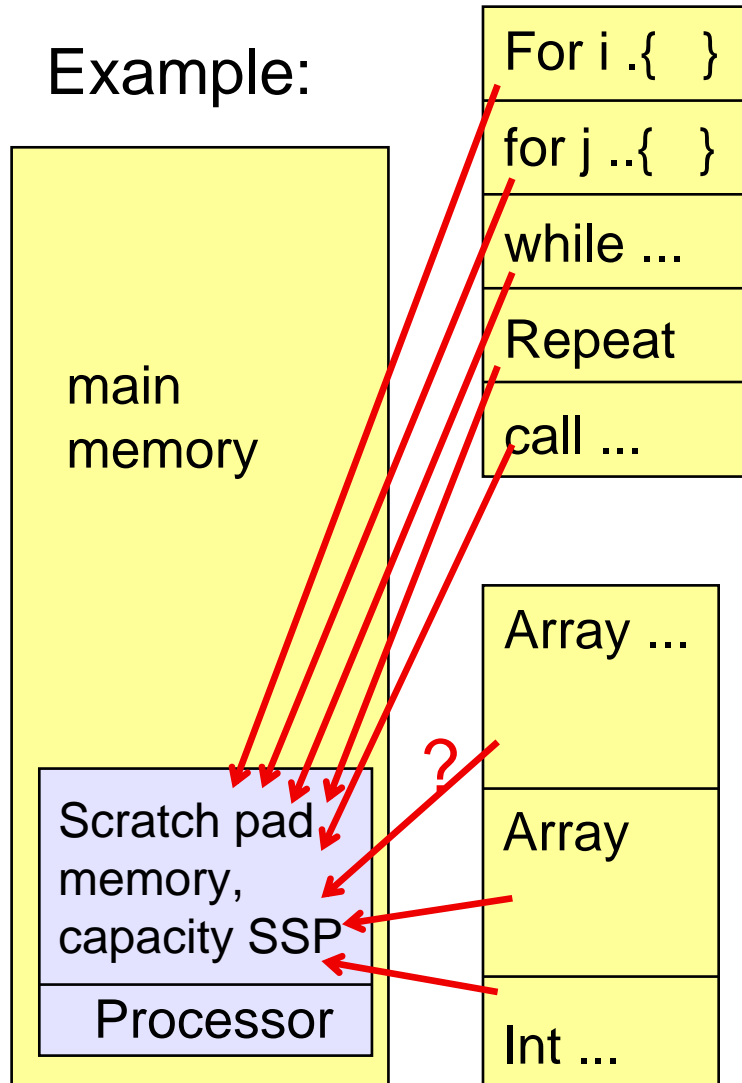


Structure of the course

More on validation/verification:  tutorial by Larsen



Migration of data & instructions, global optimization model (TU Dortmund)



Which memory object (array, loop, etc.) to be stored in SPM?

Non-overlapping (“Static”) allocation:

Gain g_k and size s_k for each object k . Maximise gain $G = \sum g_k$, respecting size of SPM $SSP \geq \sum s_k$.

Solution: knapsack algorithm.

Overlaying (“dynamic”) allocation:

Moving objects back and forth

IP representation

- migrating functions and variables-

Symbols:

$S(var_k)$ = size of variable k

$n(var_k)$ = number of accesses to variable k

$e(var_k)$ = energy **saved** per variable access, if var_k is migrated

$E(var_k)$ = energy **saved** if variable var_k is migrated (= $e(var_k) n(var_k)$)

$x(var_k)$ = decision variable, =1 if variable k is migrated to SPM,
=0 otherwise

K = set of variables; Similar for functions I

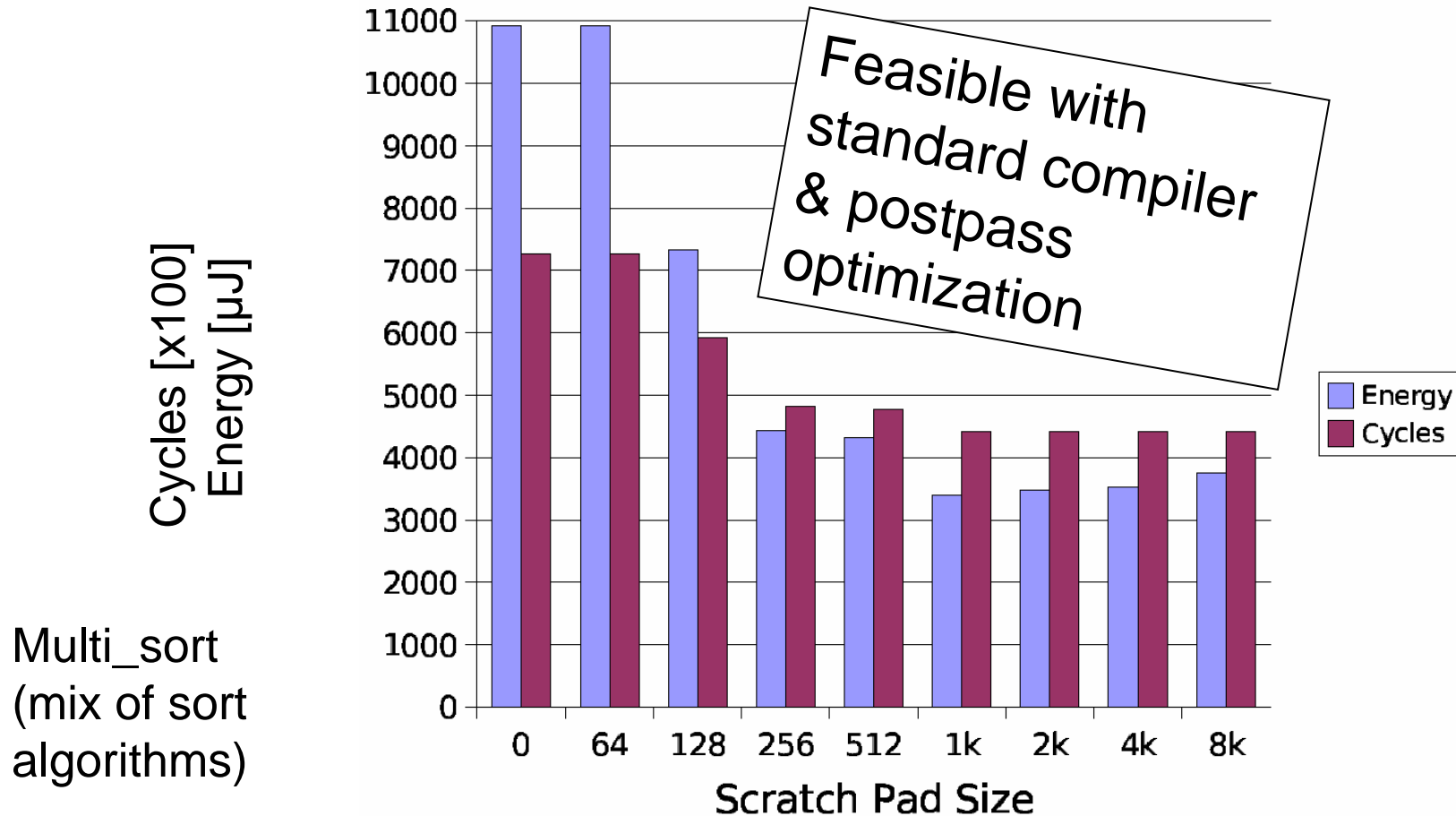
Integer programming formulation:

Maximize $\sum_{k \in K} x(var_k) E(var_k) + \sum_{i \in I} x(F_i) E(F_i)$

Subject to the constraint

$\sum_{k \in K} S(var_k) x(var_k) + \sum_{i \in I} S(F_i) x(F_i) \leq SSP$

Reduction in energy and average run-time



Measured processor / external memory energy + CACTI values for SPM (combined model)

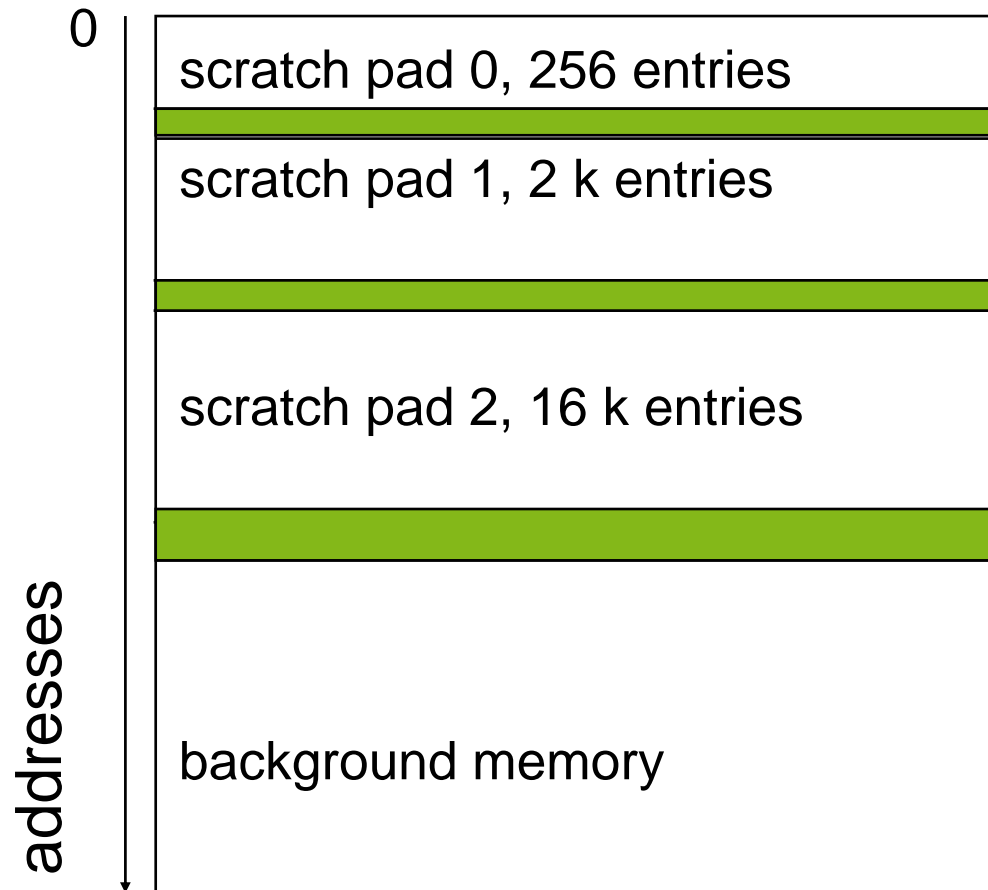
Numbers will change with technology, algorithms remain unchanged.

Multiple scratch pads

Small is beautiful:

One small SPM is beautiful (😊).

May be, several smaller SPMs are even more beautiful (😊 😊 😊)?



Optimization for multiple scratch pads

Minimize
$$C = \sum_j e_j \cdot \sum_i x_{j,i} \cdot n_i$$

With e_j : energy per access to memory j ,
and $x_{j,i} = 1$ if object i is mapped to memory j , $=0$ otherwise,
and n_i : number of accesses to memory object i ,
subject to the constraints:

$$\forall j: \sum_i x_{j,i} \cdot S_i \leq SSP_j$$

$$\forall i: \sum_j x_{j,i} = 1$$

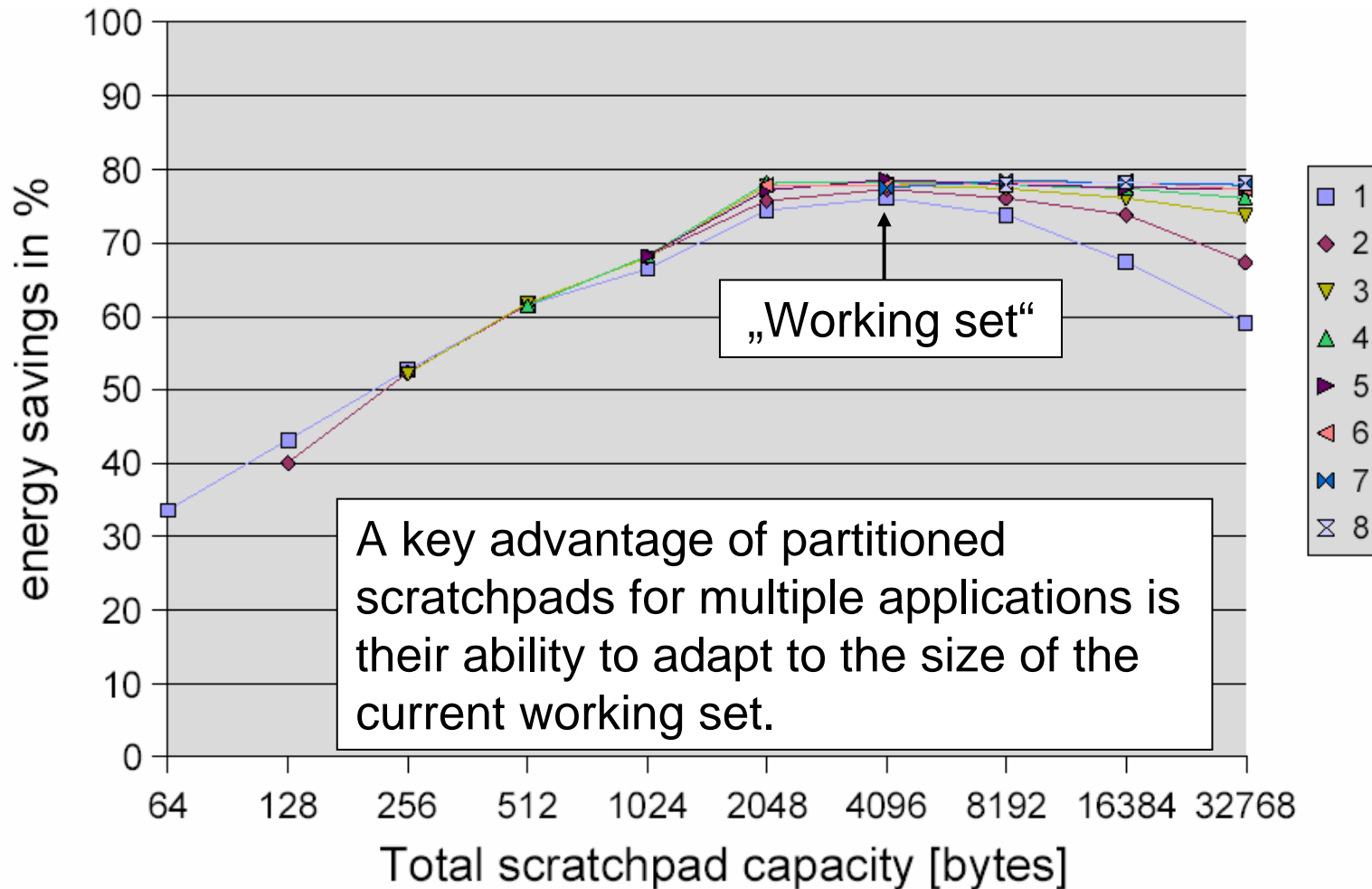
With S_i : size of memory object i ,
 SSP_j : size of memory j .

Considered partitions

Example of considered memory partitions for a total capacity of 4096 bytes

# of partitions	number of partitions of size:						
	4k	2k	1k	512	256	128	64
7	0	1	1	1	1	1	2
6	0	1	1	1	1	2	0
5	0	1	1	1	2	0	0
4	0	1	1	2	0	0	0
3	0	1	2	0	0	0	0
2	0	2	0	0	0	0	0
1	1	0	0	0	0	0	0

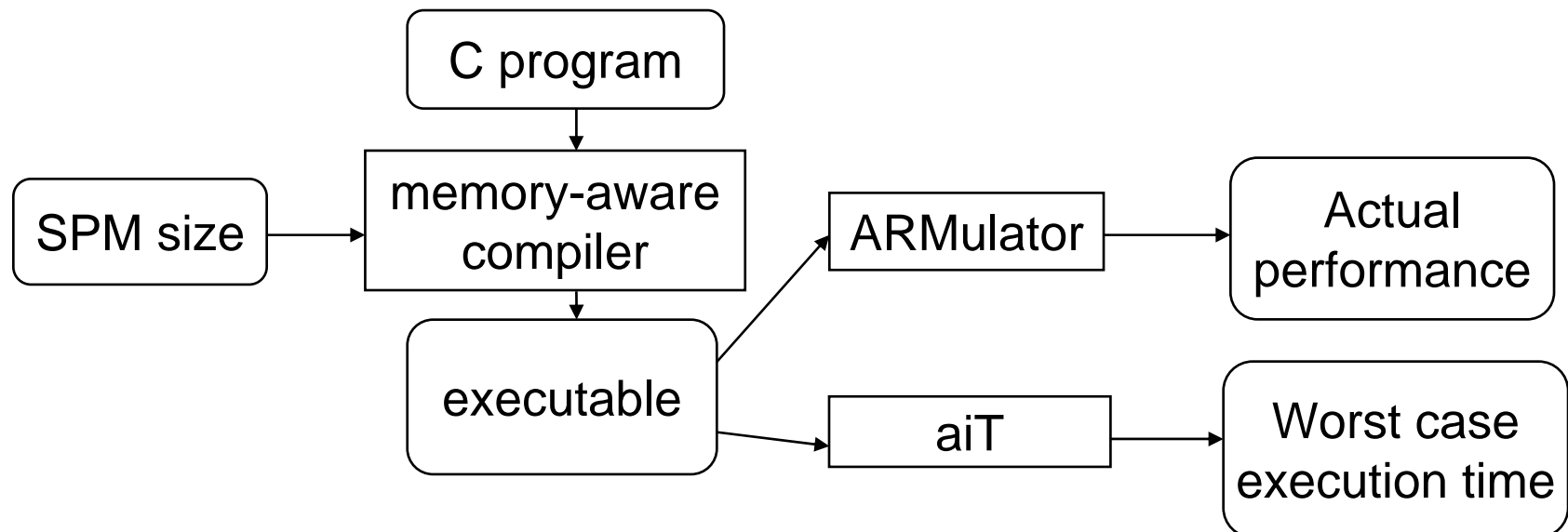
Results for parts of GSM coder/decoder



Scratch-pad/tightly coupled memory based predictability

Pre run-time scheduling is often the only practical means of providing predictability in a complex system [Xu, Parnas].

- 👉 Time-triggered, statically scheduled operating systems
- 👉 Let's do the same for the memory system
 - 👉 Are SPMs really more timing predictable?
 - 👉 Analysis using the aiT timing analyzer



Architectures considered

ARM7TDMI with 3 different memory architectures:

1. Main memory

LDR-cycles: (CPU,IF,DF)=(3,2,2)

STR-cycles: (2,2,2)

* = (1,2,0)

2. Main memory + unified cache

LDR-cycles: (CPU,IF,DF)=(3,12,6)

STR-cycles: (2,12,3)

* = (1,12,0)

3. Main memory + scratch pad

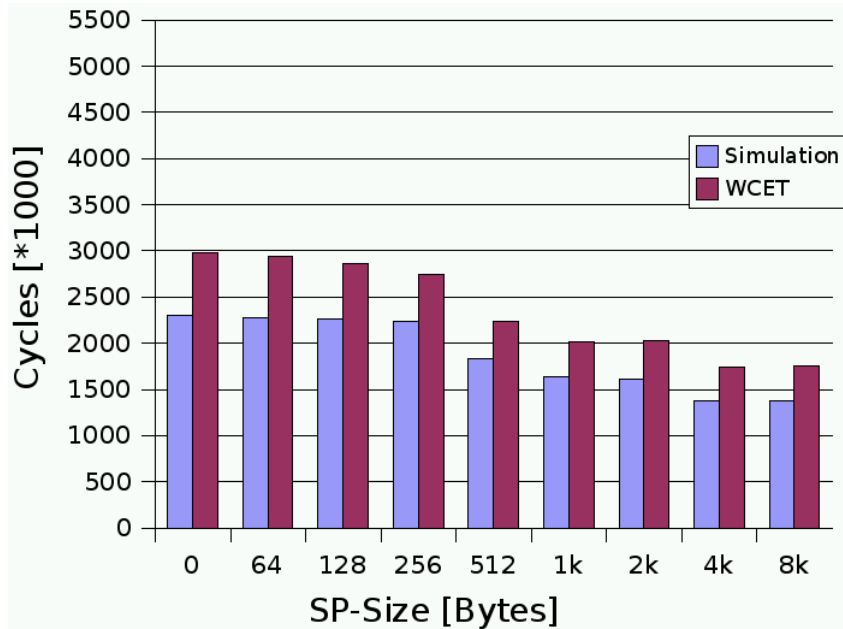
LDR-cycles: (CPU,IF,DF)=(3,0,2)

STR-cycles: (2,0,0)

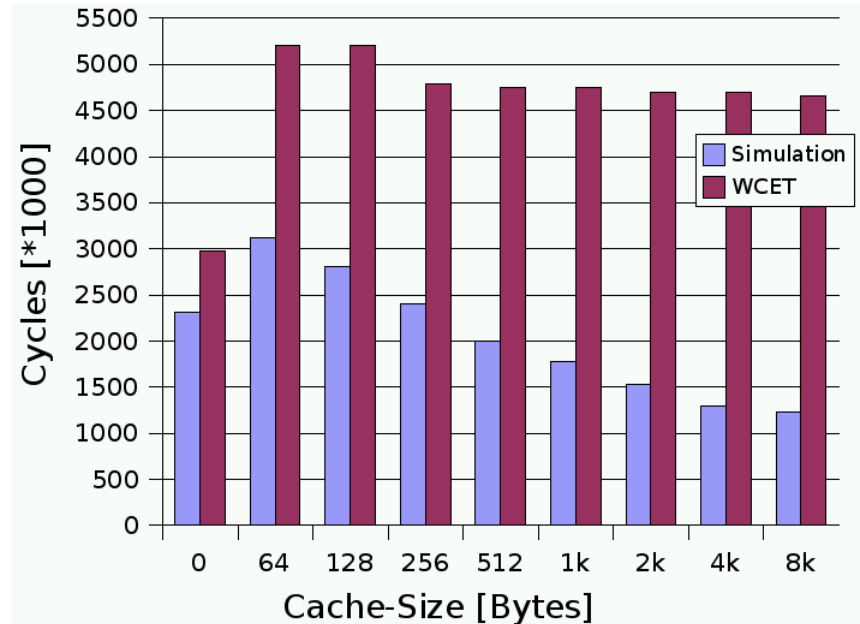
* = (1,0,0)

Results for G.721

Using Scratchpad:



Using Unified Cache:

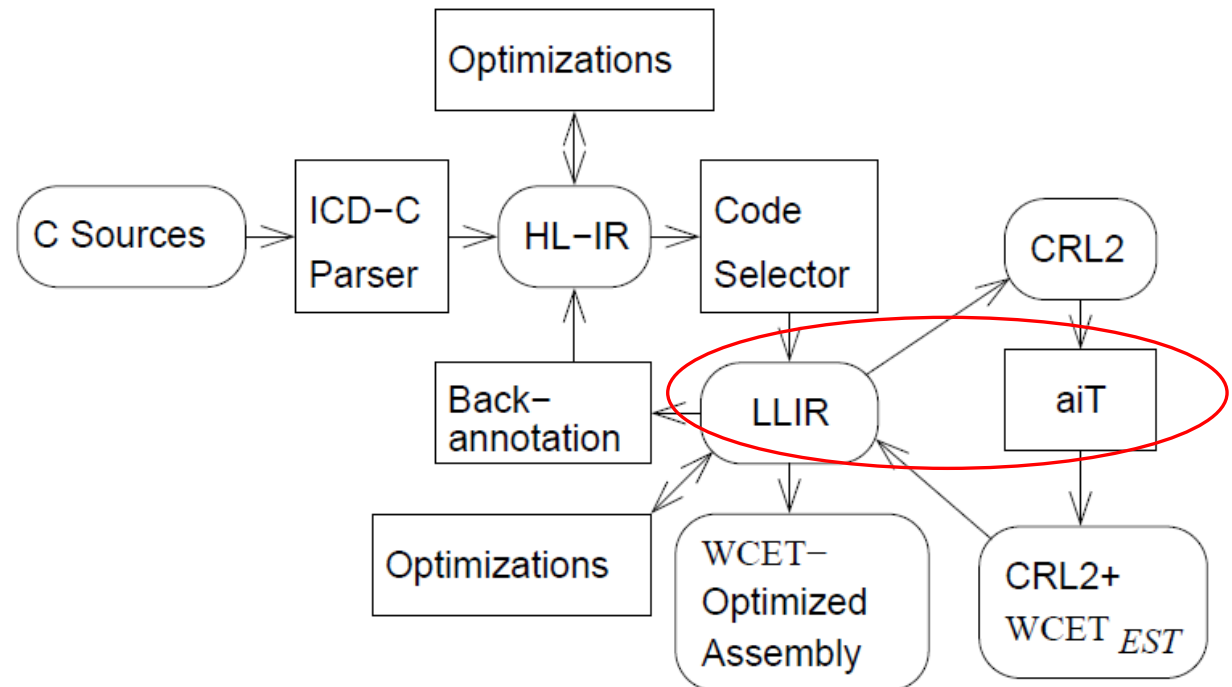


References:

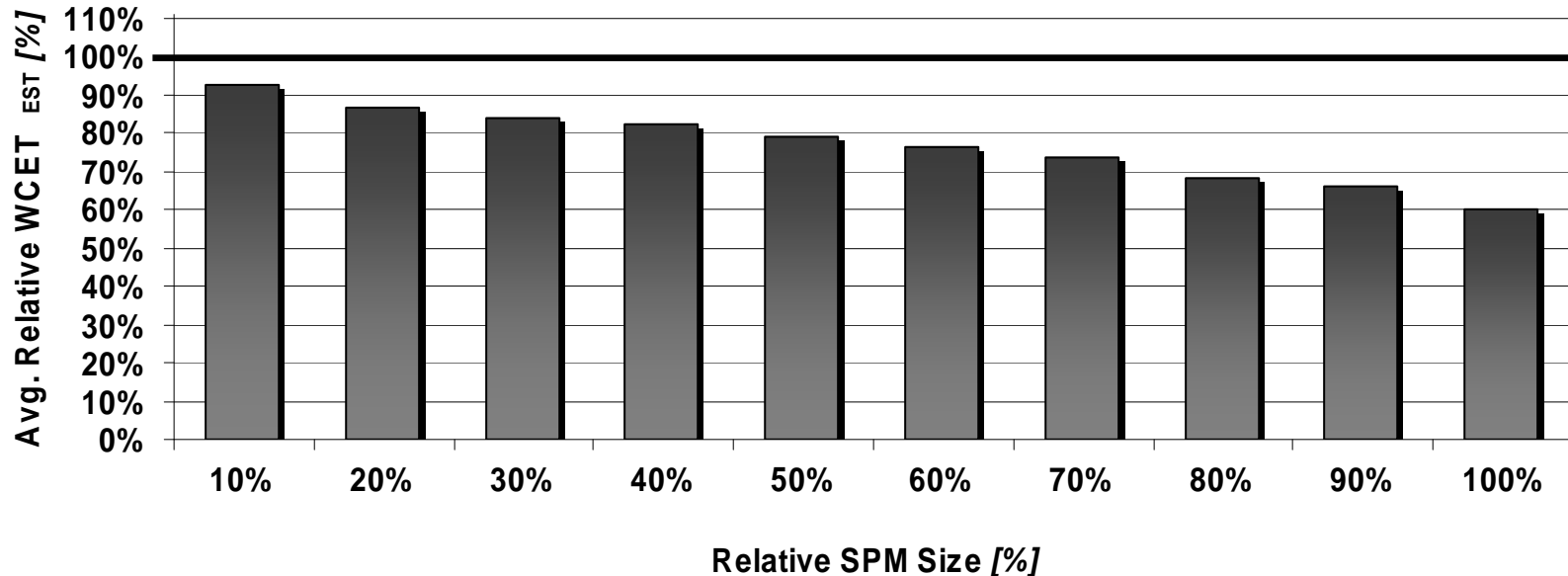
- Wehmeyer, Marwedel: Influence of Onchip Scratchpad Memories on WCET: 4th Intl Workshop on worst-case execution time (WCET) analysis, Catania, Sicily, Italy, June 29, 2004
- Second paper on SP/Cache and WCET at DATE, March 2005

Tight integration of compilation and timing analysis

- Computation of the WCET **after** compilation does not give us optimum results
- Let's optimize for the WCET **during** compilation
- Tight integration of aiT WCET analyzer from AbsInt into experimental WCET aware compiler WCC



Average WCET_{EST} for 73 Benchmarks

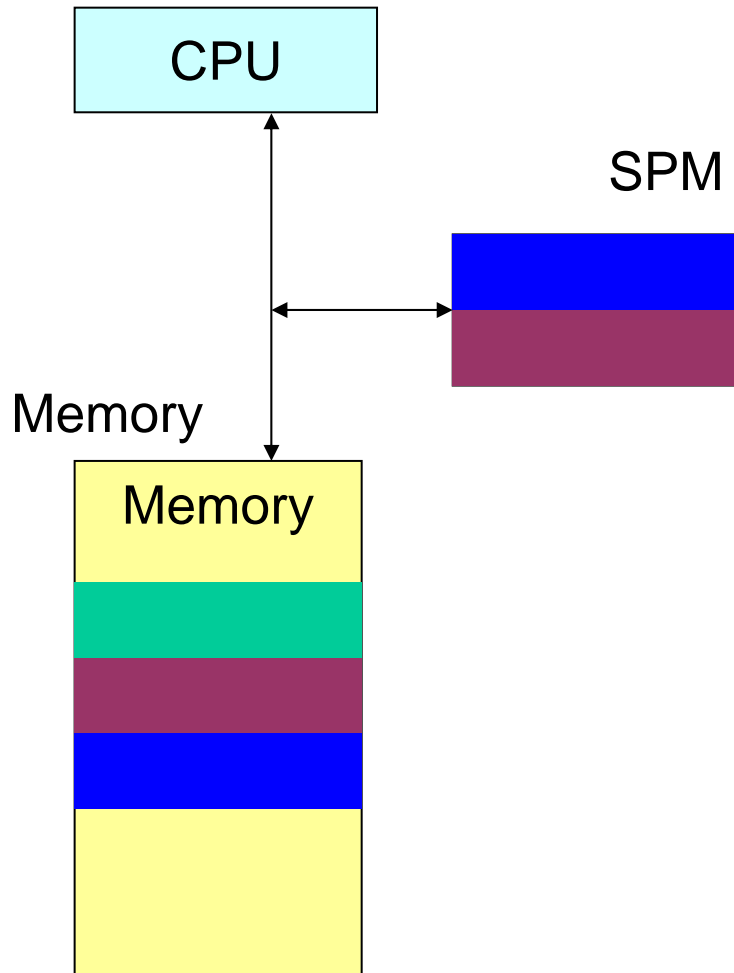


Steady WCET_{EST} decreases for increasing SPM sizes
WCET_{EST} reductions from 7% – 40%

X-Axis: SPM size = x% of benchmark's code size
Y-Axis: 100% = WCET_{EST} when not using SPM at all

H. Falk, J. Kleinsorge: Optimal Static WCET-aware Scratch-pad Allocation of Program Code, *46th Design Automation Conference (DAC)*, 2009

Dynamic replacement within scratch pad

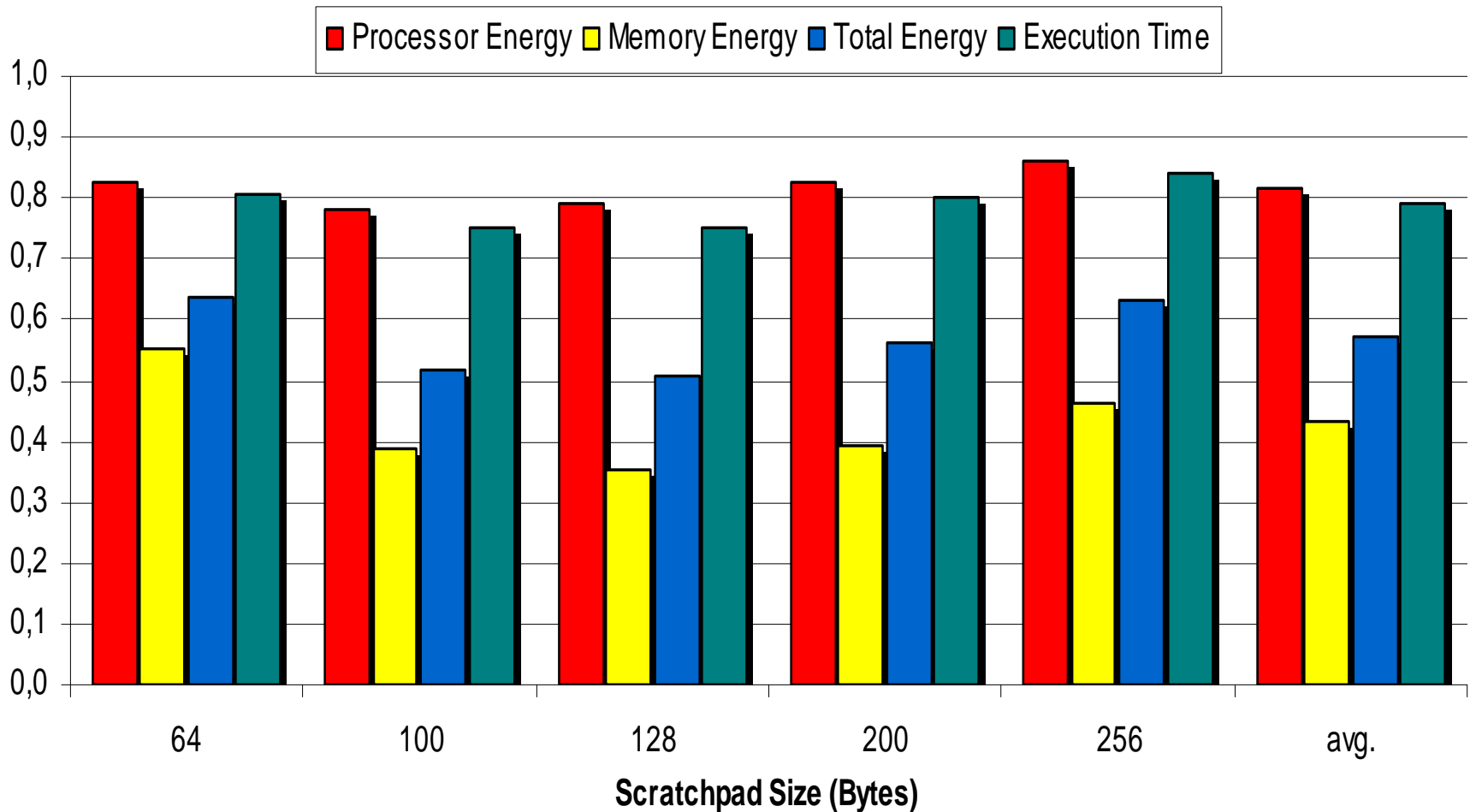


- Effectively results in a kind of **compiler-controlled segmentation/ paging** for SPM
- Address assignment within SPM required (paging or segmentation-like)

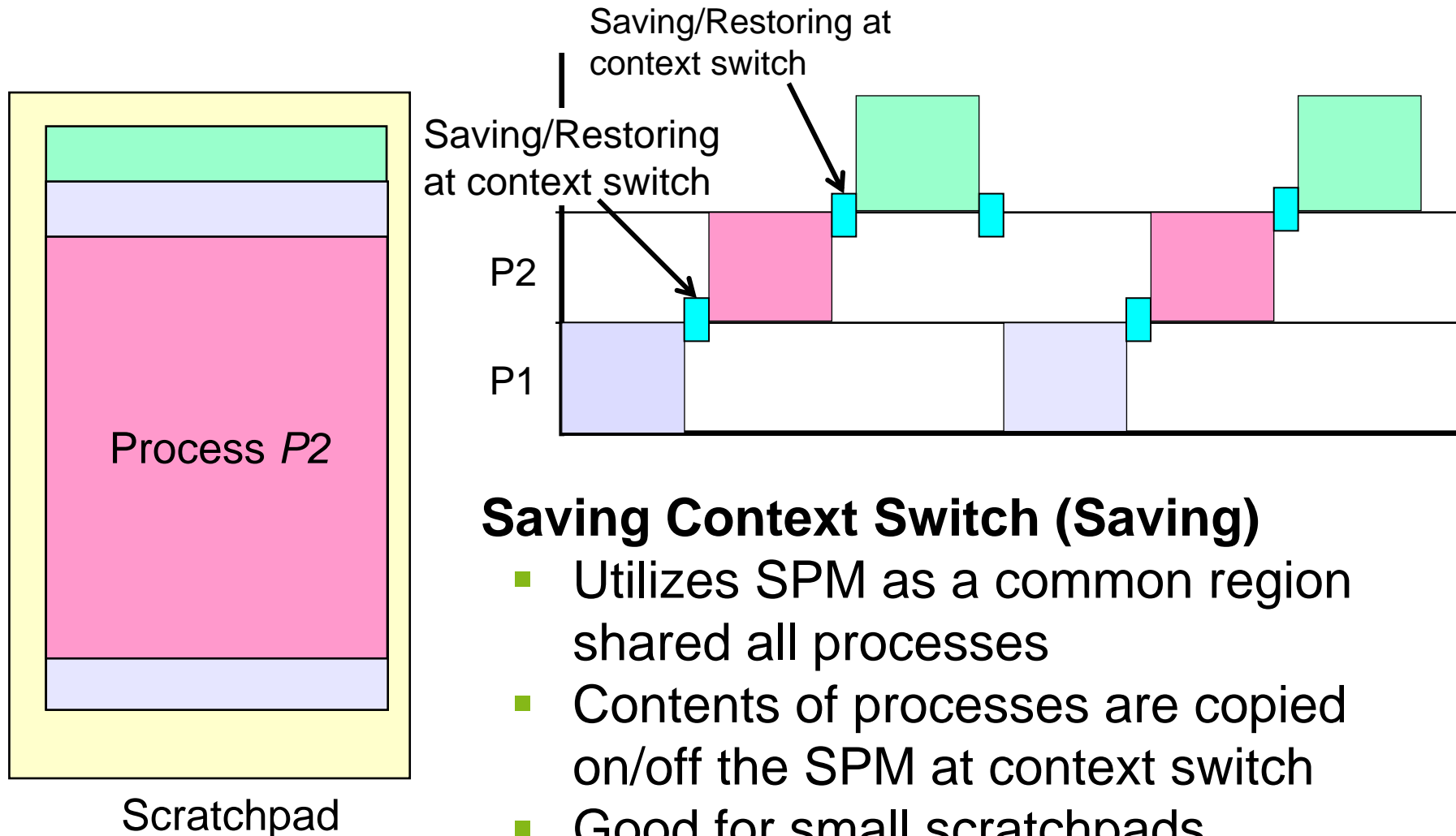
Reference: Verma, Marwedel: Dynamic Overlay of Scratchpad Memory for Energy Minimization, ISSS 2004

Dynamic replacement within scratch pad

- Results for edge detection relative to static allocation -



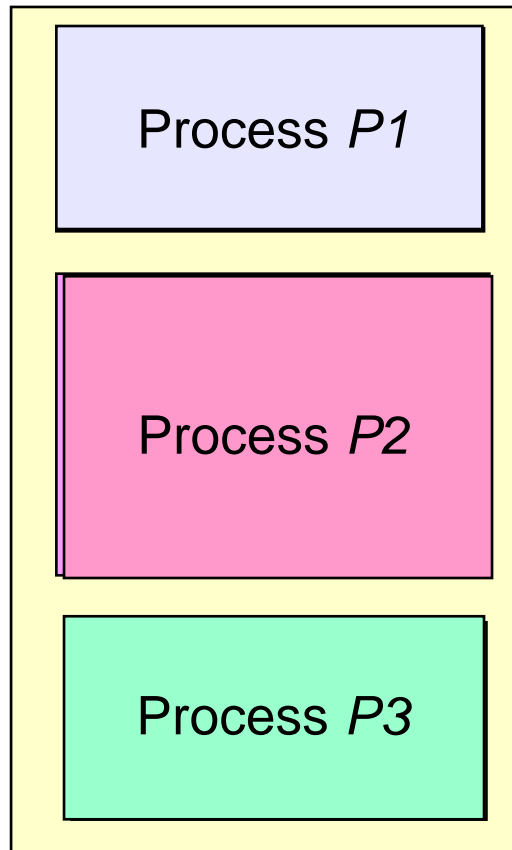
Saving/Restoring Context Switch



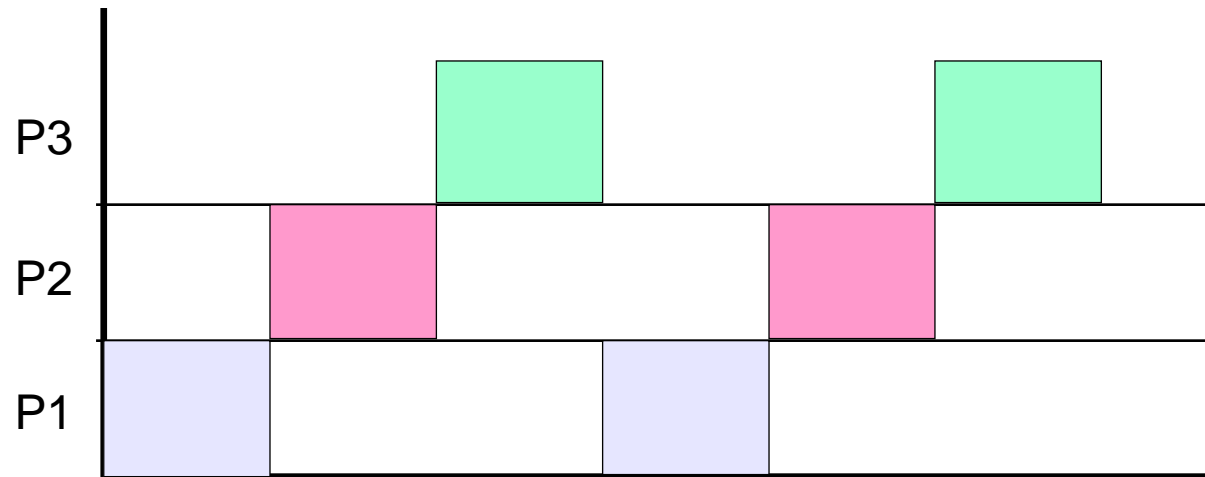
Saving Context Switch (Saving)

- Utilizes SPM as a common region shared all processes
- Contents of processes are copied on/off the SPM at context switch
- Good for small scratchpads

Non-Saving Context Switch



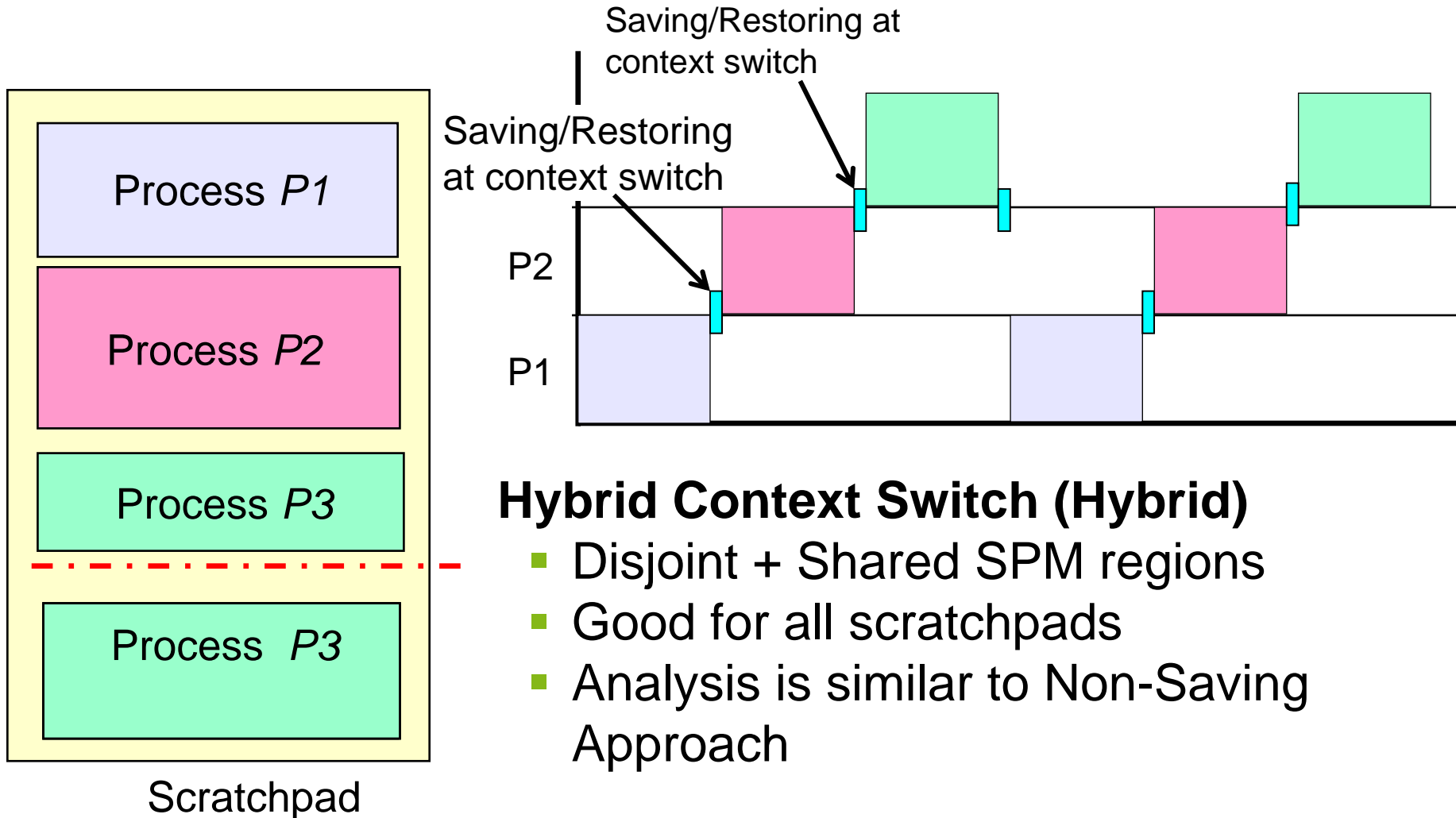
Scratchpad



Non-Saving Context Switch

- Partitions SPM into disjoint regions
- Each process is assigned a SPM region
- Copies contents during initialization
- Good for large scratchpads

Hybrid Context Switch

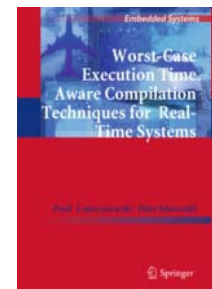
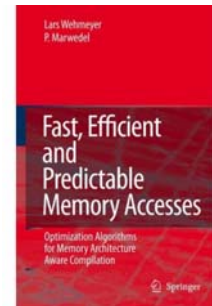


Hybrid Context Switch (Hybrid)

- Disjoint + Shared SPM regions
- Good for all scratchpads
- Analysis is similar to Non-Saving Approach

Research monographs

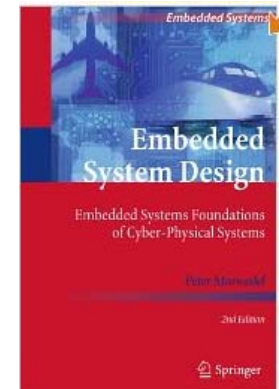
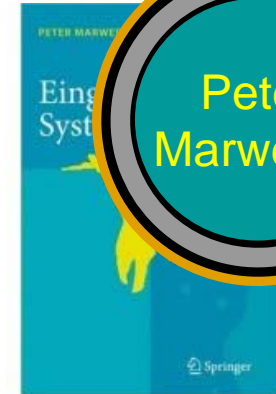
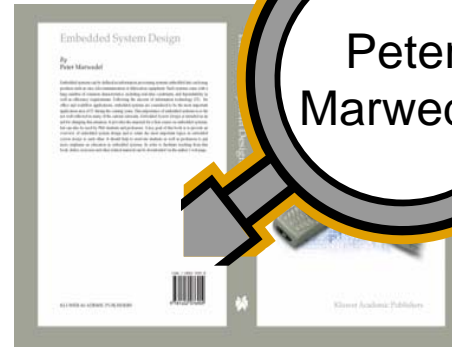
- Lars Wehmeyer, Peter Marwedel: Fast, Efficient and Predictable Memory Accesses, *Springer*, 2006
- Manish Verma, Peter Marwedel: Advanced Memory Optimization Techniques for Low-Power Embedded Processors, *Springer*, May 2007
- Paul Lokuciejewski, Peter Marwedel: WCET-aware Source Code and Assembly Level Optimization Techniques for Real-Time Systems, *Springer*, 2010



Textbook(s)

Several editions/translations:

- 1st edition
 - English
 - Original hardcover version
 - Reprint, soft cover, 2006
 - German, 2007
 - Chinese, 2006
 - Macedonian, 2010
- 2nd edition, with CPS
 - English, Dec. 2010/Jan. 2011
 - German, TBA
 - Plans for Portuguese & Greek edition

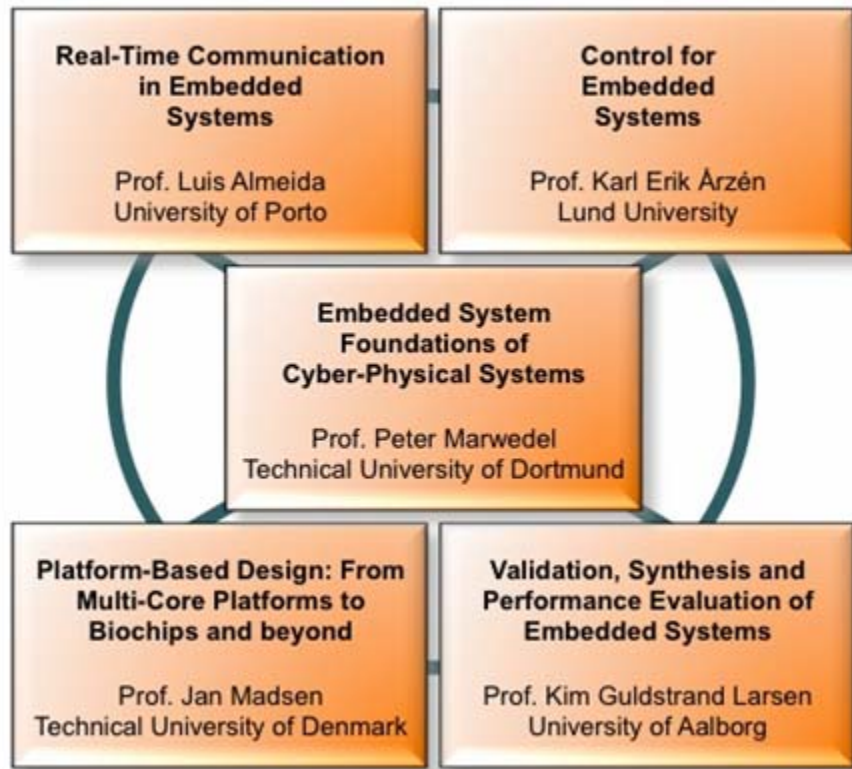


Slides available at:
<http://ls12-www.cs.tu-dortmund.de/~marwedel/es-book>

Overall Summary

- Introduction, Motivation and Overview
 - Motivation
 - Common characteristics
- Specifications and Modeling
 - Models of computation
 - Early phases
 - FSM-based models, Data flow, Petri nets, discrete event-based models, Von-Neumann models
 - Comparison
- Exploitation of the memory hierarchy
 - Scratch pad memories
 - Non-overlapping allocation
 - Overlapping allocation

Links to the rest of the course



	Morning	Afternoon
Monday	Marwedel	Marwedel
Tuesday	Madsen	Larsen
Wednesday	Madsen	Larsen
Thursday	Almeida	Arzen
Friday	Almeida	Arzen