Embedded Systems in a nutshell

Peter Marwedel
TU Dortmund, Informatik 12
& ICD e.V.
Germany

Graphics: © Alexandra Nolte, Gesine Marwedel, 2003

Motivation for the tutorial (1)

According to forecasts characterized by terms such as

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

Basic technologies:

- *Embedded Systems*
- Communication technologies

Motivation for the tutorial (2)

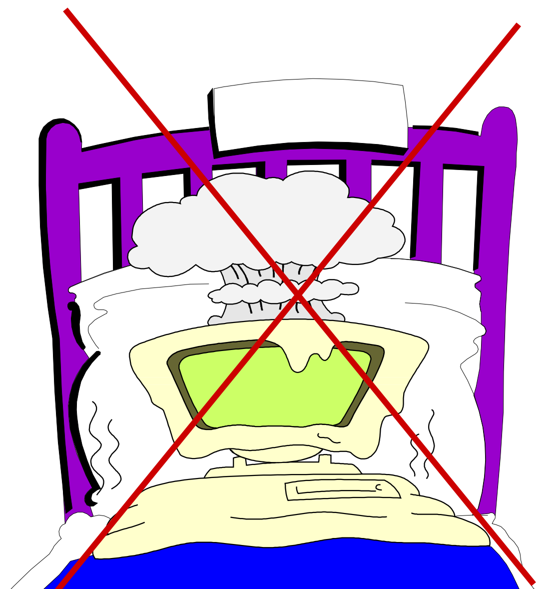
"Information technology (IT) is on the verge of another revolution.... These networked systems of embedded computers ... have the potential to change radically the way people interact with their environment … ... The use of [these embedded computers] throughout society could well dwarf previous milestones in the information revolution."

National Research Council Report (US) Embedded Everywhere

[Source. Ed Lee, UC Berkeley, ARTEMIS Embedded Systems Conference, Graz, 5/2006]

What is an embedded system?

http://www.skywatchers-dragons-fairies.com/kitchen_fairies.htm



Embedded Systems

“Dortmund“ Definition:

Information processing systems embedded into a larger product

Main reason for buying is **not** information processing

Berkeley Modell [Ed Lee]:

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

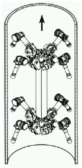
Application areas

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







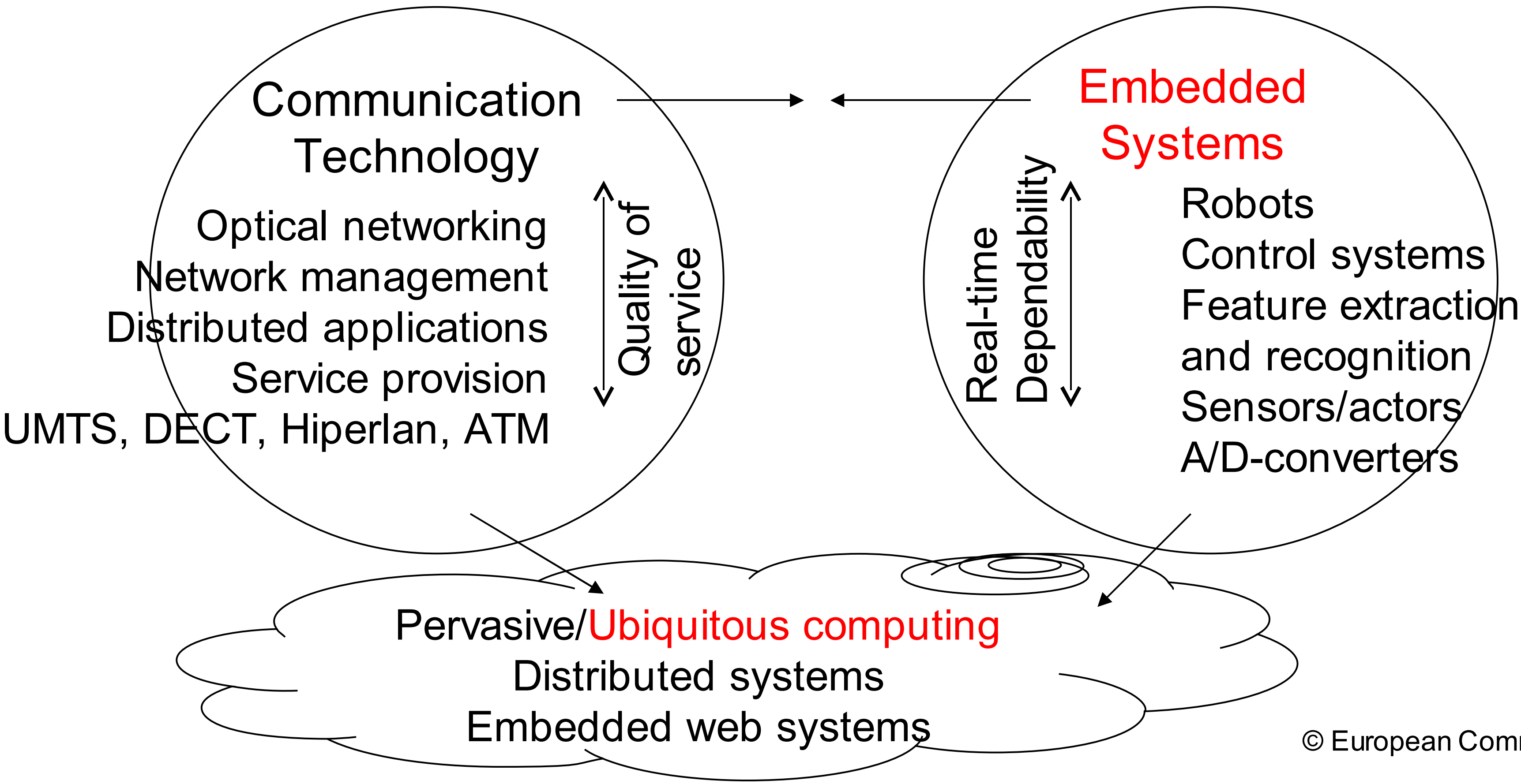
Growing importance of embedded systems

Growing economical importance of embedded systems, e.g.:

- *World market for electronic products was worth some $1.8 trillion in 2006, a figure that is expected to increase to $2.0 trillion in 2007 and $3.2 trillion in 2012, a compound annual growth rate (CAGR) of 9.5% over the next 5 years.* [www.itfacts.biz, Dec. 17th, 2007]
- *Spending on GPS units exceeded $100 mln during Thanksgiving week, up 237% from 2006 holiday season. The average price fell from $322 in 2006 to $171 in 2007. More people bought GPS units than bought PCs, NPD found.* [www.itfacts.biz, Dec. 6th, 2007]
- *With the blessing of government payers in Western Europe and Canada, 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]
- *The automotive sector … ensures the employment of more than 4 million people in Europe.* [OMI bulletin]

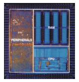
Embedded systems and ubiquitous computing

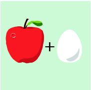
Ubiquitous computing: Information anytime, anywhere.
Embedded systems provide fundamental technology.



Characteristics of Embedded Systems

- Must be **dependable**

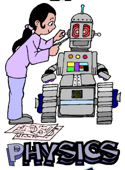






- Must be **efficient**
  (energy, code-size, run-time, weight, cost efficient)

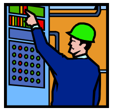
- **Dedicated** towards a certain **application**
- **Dedicated user interface**

- Many ES must meet **real-time constraints**



- Frequently **connected to physical environment** through sensors and actuators,

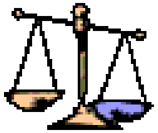
- **Hybrid systems** (analog + digital parts).

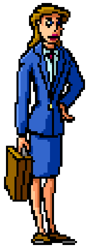


- Typically, ES are **reactive systems**:
  (In continual interaction with is environment)

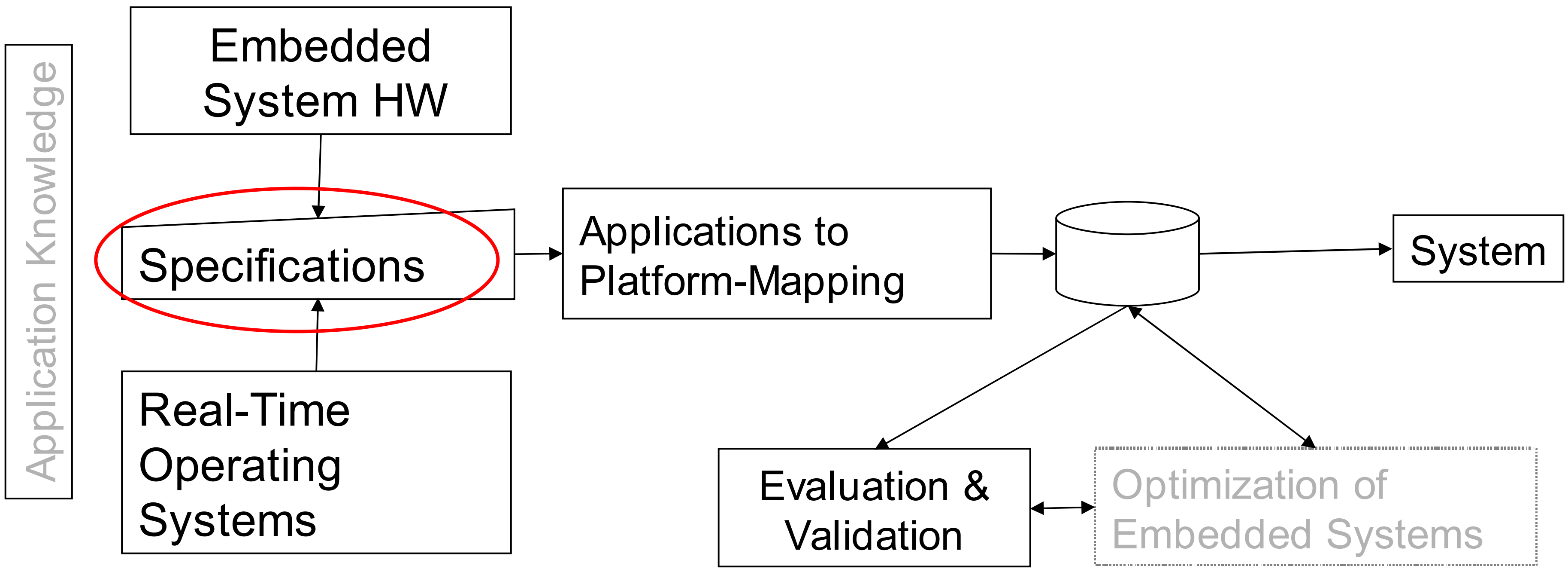
Challenges for Embedded Software

- 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 tutorial

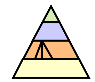
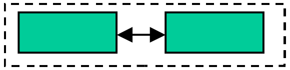
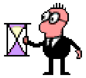
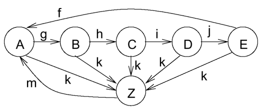


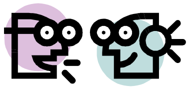
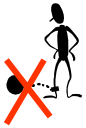
Technische Universität Dortmund

Specification of Embedded Systems

Peter Marwedel
TU Dortmund, Informatik 12
& ICD e.V.
Germany

Specification of embedded systems: Requirements for specification techniques

- 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 language will meet all require-ments ☞ compro-mises

Models of computation
- Definition -

Models of computation define:

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

Components (1)

- Discrete event model

queue

a 6
b 7
c 8

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

- Von Neumann model

Sequential execution, program memory etc.

Example: Observer Pattern With Mutual Exclusion (Mutexes)

```
public synchronized void addListener(listener) {…}

public synchronized void setValue(newValue) {
    myValue = newValue;

    for (int i = 0; i < myListeners.length; i++) {
        myListeners[i].valueChanged(newValue)
    }
}
```

Javasoft recommends against this. What's wrong with it?

Mutexes using Monitors are Minefields

```
public synchronized void addListener(listener) {…}

public synchronized void setValue(newValue) {
    myValue = newValue;

    for (int i = 0; i < myListeners.length; i++) {
        myListeners[i].valueChanged(newValue)
    }
}
```

valueChanged() may attempt to acquire a lock on some other object and stall. If the holder of that lock calls addListener(), deadlock!

Simple Observer Pattern Becomes Not So Simple

```
public synchronized void addListener(listener) {…}

public void setValue(newValue) {
    synchronized(this) {
        myValue = newValue;
        listeners = myListeners.clone();
    }

    for (int i = 0; i < listeners.length; i++) {
        listeners[i].valueChanged(newValue)
    }
}
```

while holding lock, make copy of listeners to avoid race conditions

notify each listener outside of synchronized block to avoid deadlock

This still isn't right.
What's wrong with it?

Simple Observer Pattern: How to Make It Right?

```
public synchronized void addListener(listener) {…}

public void setValue(newValue) {
    synchronized(this) {
        myValue = newValue;
        listeners = myListeners.clone();
    }

    for (int i = 0; i < listeners.length; i++) {
        listeners[i].valueChanged(newValue)
    }
}
```

Suppose two threads call setValue(). One of them will set the value last, leaving that value in the object, but listeners may be notified in the opposite order. The listeners may be alerted to the value changes in the wrong order!

Problems with thread-based concurrency

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

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

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

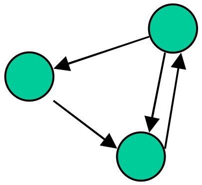
Problems with classical CS theory and von Neumann computing

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

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

Components (2)

- Finite state machines

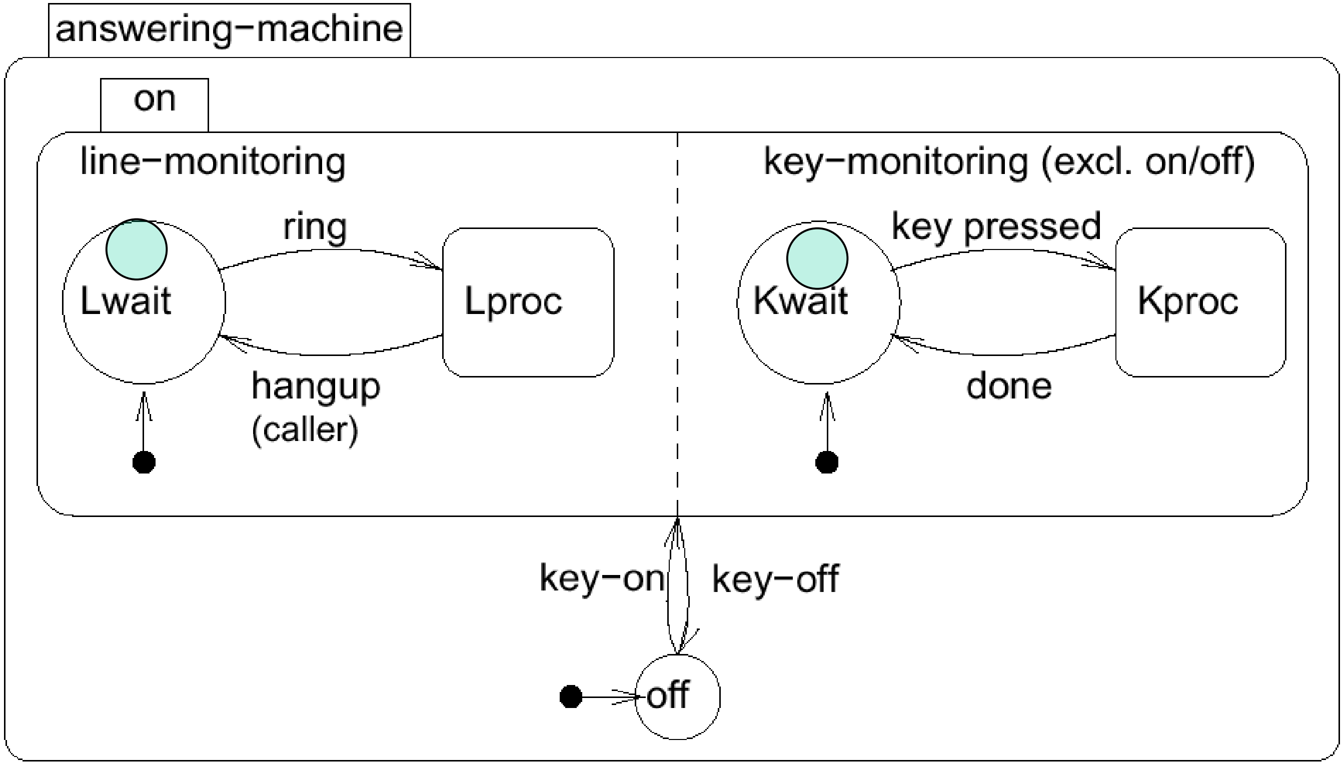


- Differential equations

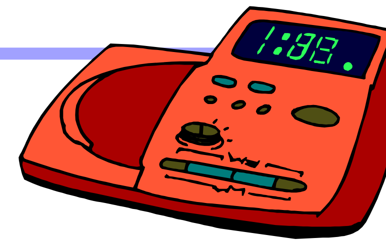
$$\frac{\partial^2 x}{\partial t^2} = b$$

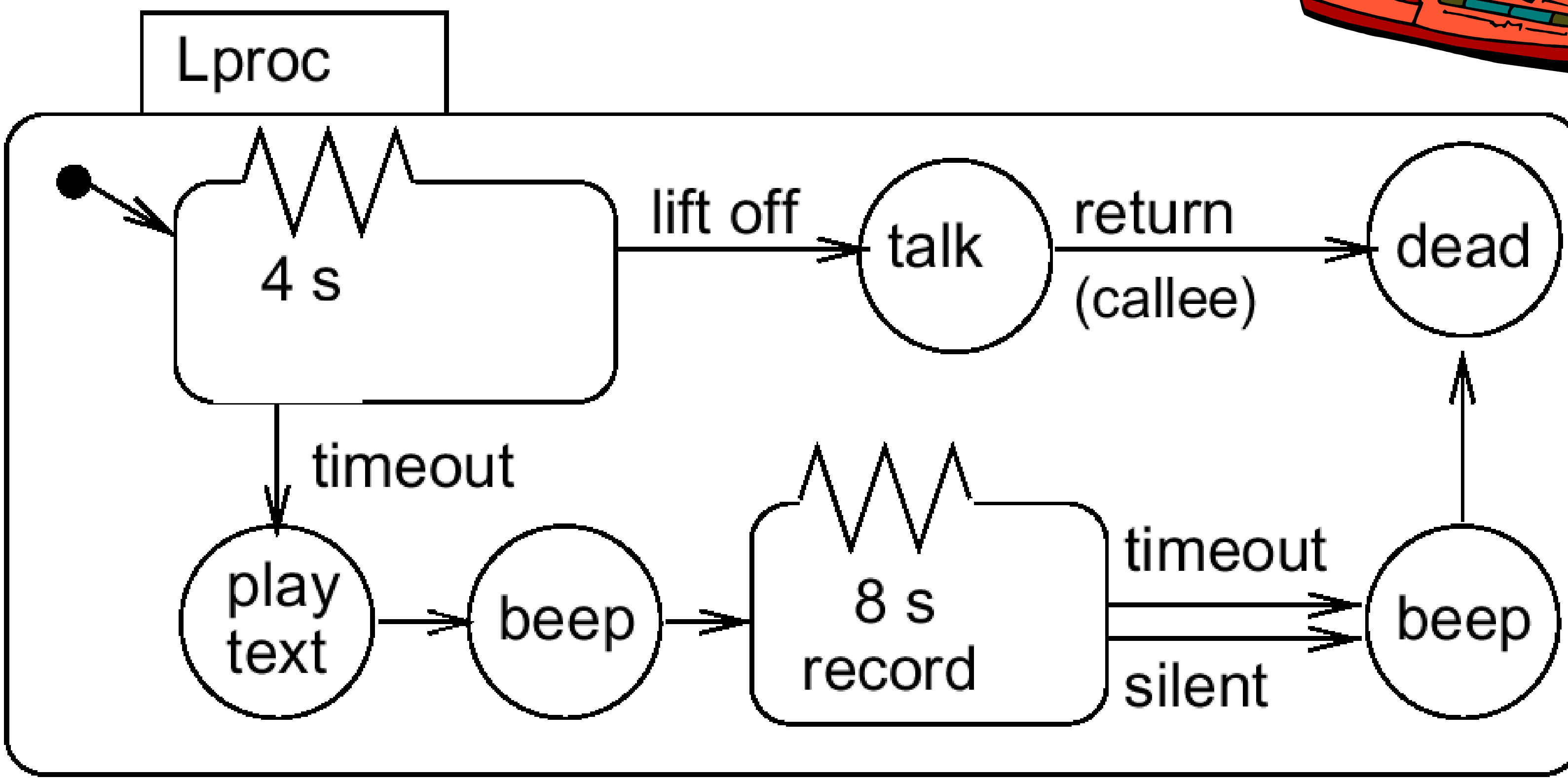
Concurrency

Convenient ways of describing concurrency are required.
AND-super-states: FSM is in **all** (immediate) sub-states of a super-state; Example:



Using timers in an answering machine





Synchronous vs. asynchronous languages (1)

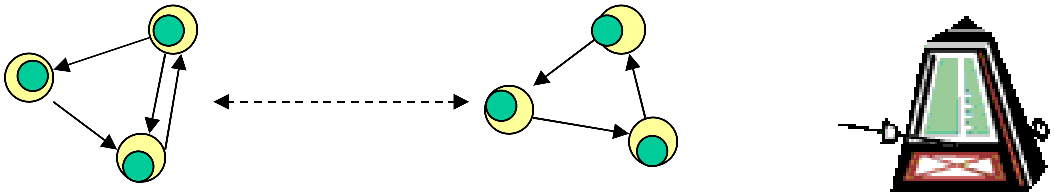
Description of several processes in many languages non-deterministic:
The order in which executable tasks are executed is not specified (may affect result).

Synchronous languages: based on automata models.

„Synchronous languages aim at providing high level, modular constructs, to make the design of such an automaton easier [Halbwachs].

Synchronous languages describe concurrently operating automata. „.. *when automata are composed in parallel, a transition of the product is made of the "simultaneous" transitions of all of them*".

Synchronous vs. asynchronous languages (2)



Synchronous languages implicitly assume the presence of a (global) clock. Each clock tick, all inputs are considered, new outputs and states are calculated and then the transitions are made.

This requires a broadcast mechanism for all parts of the model.

Idealistic view of concurrency.

Has the advantage of guaranteeing deterministic behavior.

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
}
```

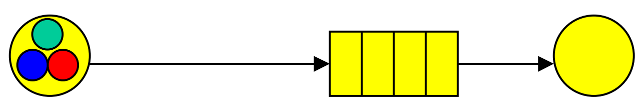
Race-free access to shared memory protected by S possible

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

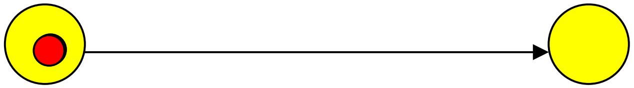


Blocking/synchronous message passing *rendez-vous*

Sender will wait until receiver has received message

```
...
send ()
...
```

```
...
receive ()
...
```



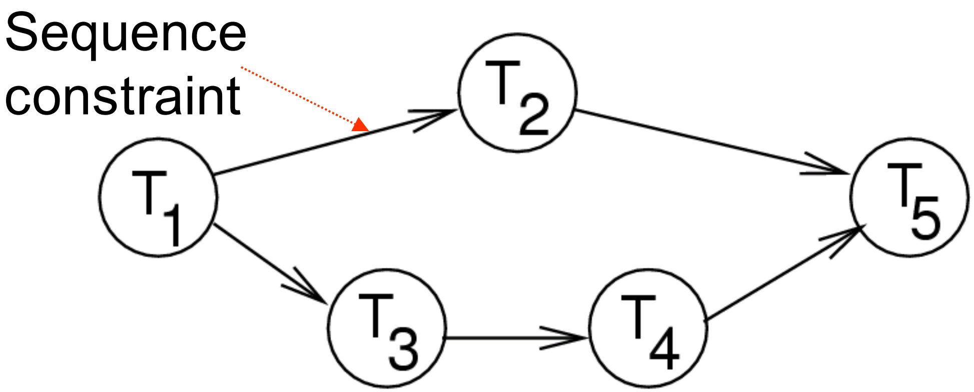
Extended *rendez-vous*

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

…
send ()
…

…
receive ()
…
ack
…

Task graphs

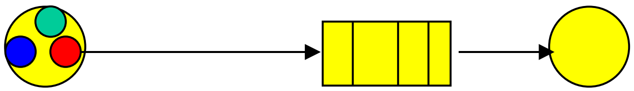


Nodes are assumed to be a "program" described in some programming language, e.g. C or Java.

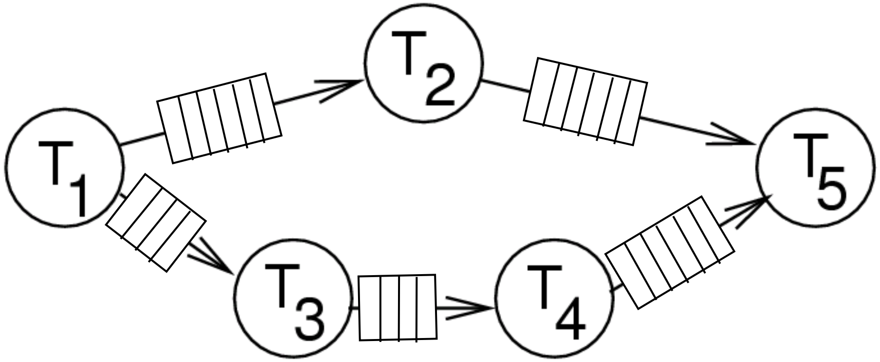
Def.: A **dependence graph** is a directed graph $G=(V,E)$ in which $E \subseteq V \times V$ is a partial order.
If $(v1, v2) \in E$, then $v1$ is called an **immediate predecessor** of $v2$ and $v2$ is called an **immediate successor** of $v1$.
Suppose E^* is the transitive closure of E.
If $(v1, v2)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

Task graphs with asynchronous message passing:
Kahn process networks

For asynchronous message passing:
communication between tasks is buffered



Special case: Kahn process networks:
executable task graphs;
Communication via infinitely large FIFOs



Properties of Kahn process networks (1)

- Each node corresponds to one program/task;
- Communication is only via channels;
- Channels include FIFOs as large as needed;
- Channels transmit information within an unpredictable but finite amount of time;
- Mapping from ≥1 input seq. to ≥1 output sequence;
- In general, execution times are unknown;
- Send operations are non-blocking, reads are blocking.
- One producer and one consumer;
  i.e. there is only one sender per channel;

Properties of Kahn process networks (2)

- There is only one sender per channel.
- 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, Kahn process networks are deterministic (!); for a given input, the result will always the same, regardless of the speed of the nodes.
  SDL-like conflicts at FIFOs do not exist.

Example

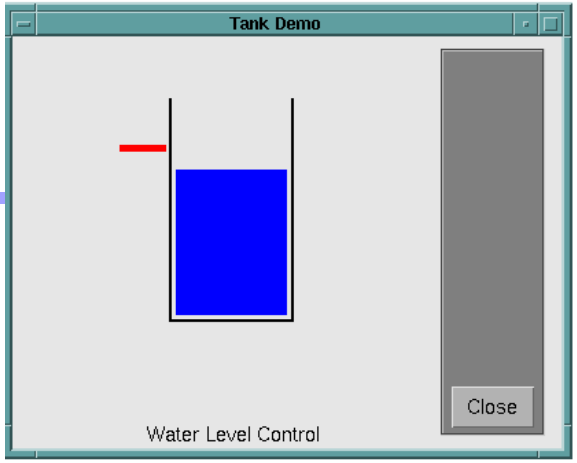
```
Process f (in int u, in int v, out int w) {
    int i; bool b = true;
    for (;;) {
            i = b ? wait (u) : wait (v); // wait returns next token in FIFO, blocks if empty
            printf("%i\n", i);
            send (i, w); // writes a token into a FIFO w/o blocking
            b = !b;
    }
}
```

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

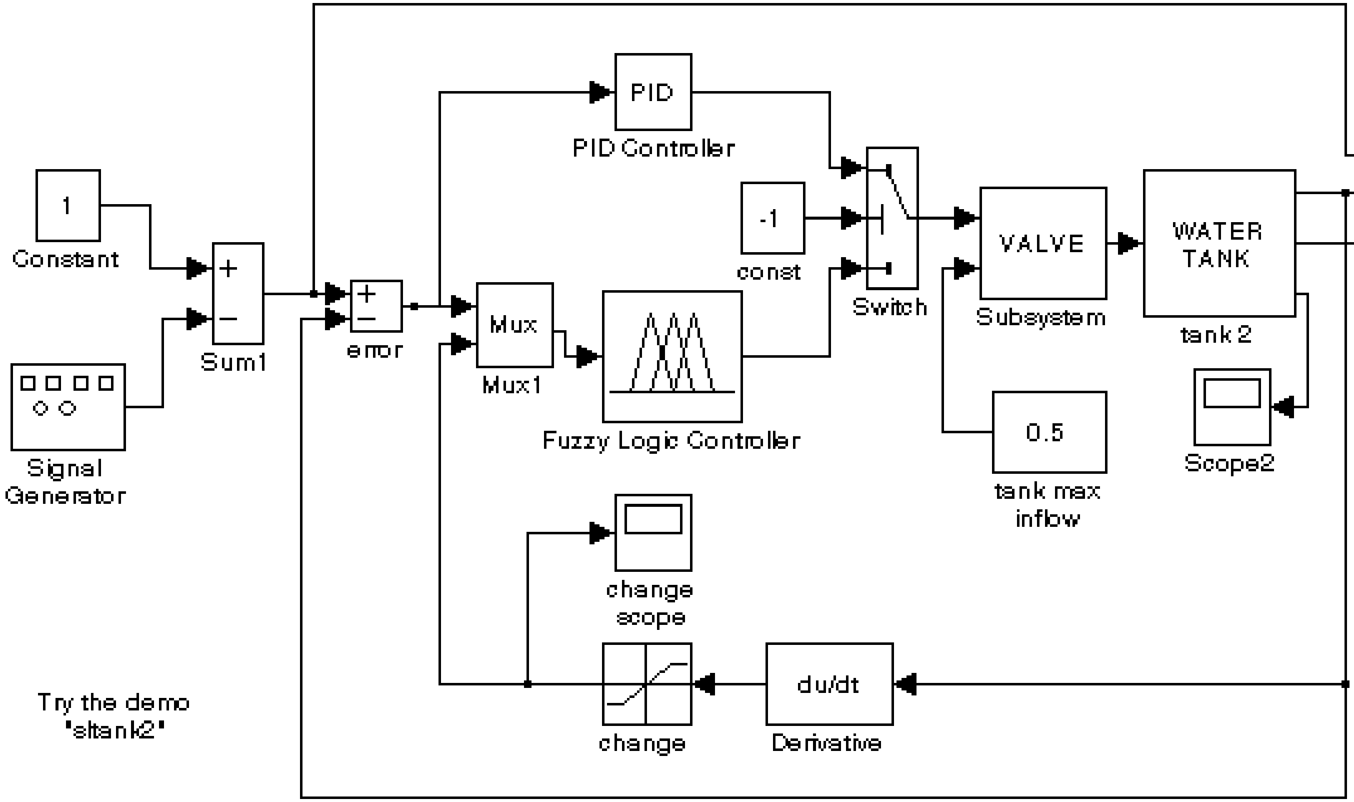
- Model of parallel computations used in practice (e.g. at Philips/NXP).
- It is a challenge to schedule KPNs without accumulating tokens

☞ http://en.wikipedia.org/wiki/Kahn_process_networks

Similar MoC: Simulink
- example -



Simulink uses an idealized timing model for block execution and communication. Both happen infinitely fast at exact points in simulated time. Thereafter, simulated time is advanced by exact time steps. [Nicolae Marian, Yue Ma]



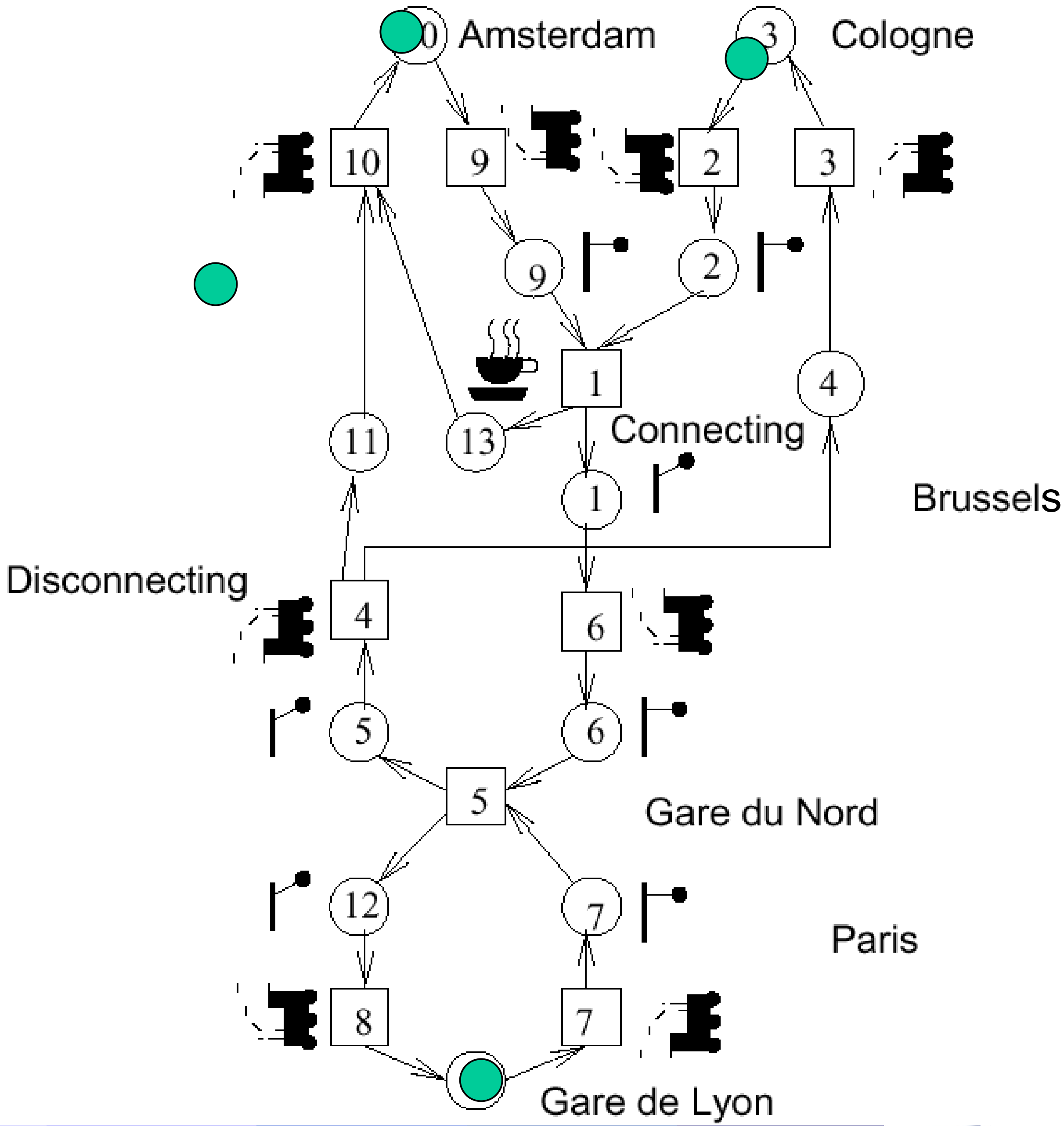
From www.mathworks.co.uk/access/helpdesk/help/toolbox/fuzzy/fuzzyt25.shtml

MATLAB/Simulink

- **MATLAB** (Matrix Laboratory):
  facility for defining matrix-based computations, extending numerical FORTRAN packages LINPACK and EISPACK with a GUI
- **Simulink:** GUI-based specification of control systems, internally using MATLAB for solving these problems.
- **StateFlow:** StateCharts-based tool integrated into MATLAB
  THE environment for (German, at least) car manufacturers

Petrinets

Slightly simplified: Synchronization at Brussels and Paris, using stations “Gare du Nord” and “Gare de Lyon” at Paris



UML for real-time?

Initially not designed for real-time.

Lacking features (1998):

- Partitioning of software into tasks and processes
- specifying timing
- specification of hardware components
- Projects on defining real-time UML based on previous work
- ROOM [Selic] is an object-oriented methodology for real-time systems developed originally at Bell-Northern Research.
- „UML profile for schedulability, performance and time“ http://www.omg.org/cgi-bin/doc?ptc/2002-03-02

Example: Activity diagram with annotations

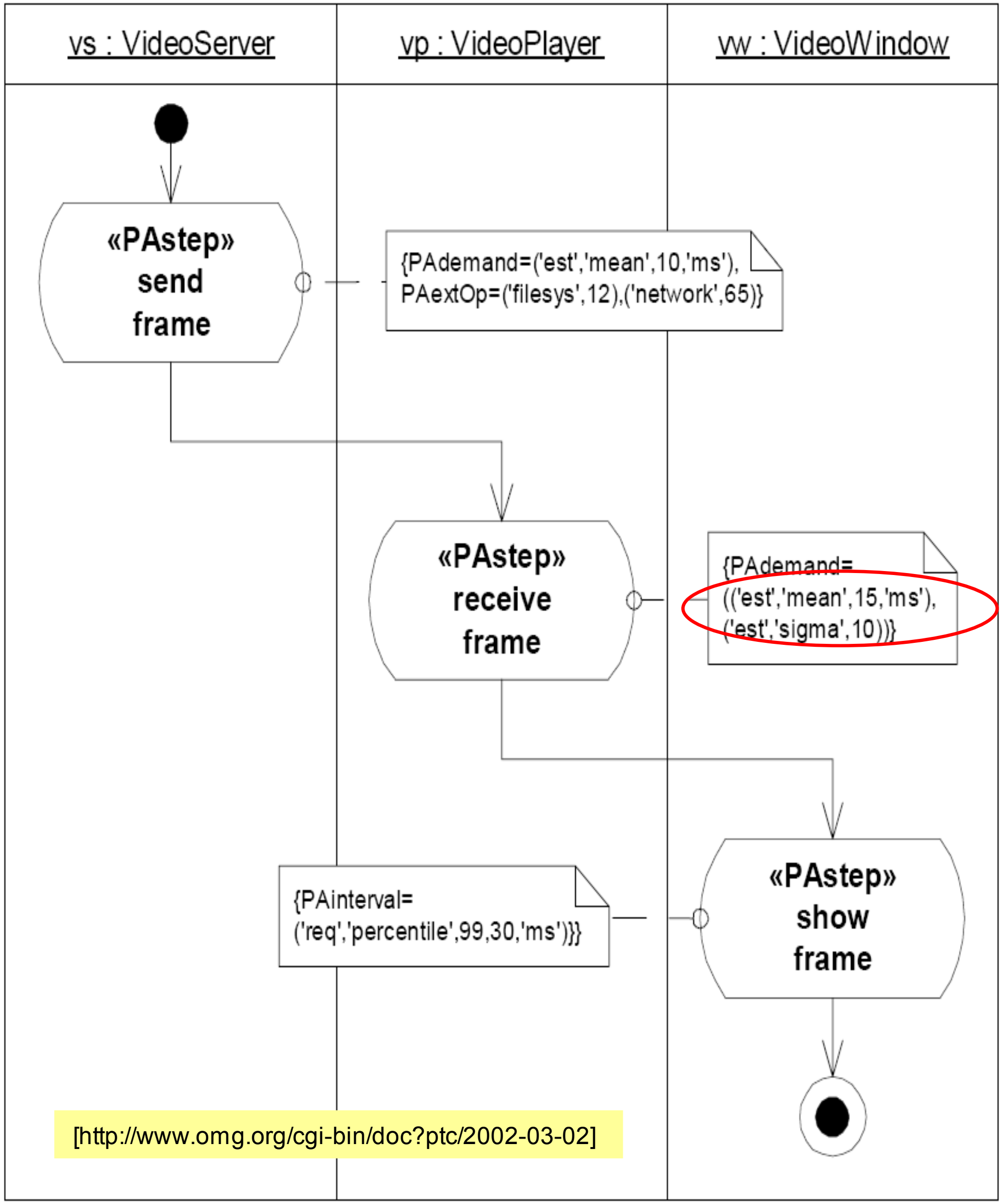


Figure 8-10 Details of the "send video" subactivity with performance annotations

See also W. Müller et al.: UML for SoC, http://jerry.c-lab.de/uml-soc/

Modeling hardware and software in one language: SystemC

Std. SW languages lacking features for HW modeling

- Concurrency, connected blocks, bit vectors
- Time, multi-valued logic
- Plug-and-play connections for intellectual property blocks

☞ **SystemC: required functions ∈ C++ class library**

- *Concurrency:* via processes, controlled by sensivity lists and calls to wait primitives.
- *Time:* Integer values in SystemC 2.0;
  Includes units such as ps, ns, µs etc.
- *Support of bit-datatypes:* bitvectors of different lengths; 2- and 4-valued logic; built-in resolution
- *Communication:* plug-and-play (pnp) channel model

Comparison of languages

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

Ptolemy

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

http://ptolemy.berkeley.edu/

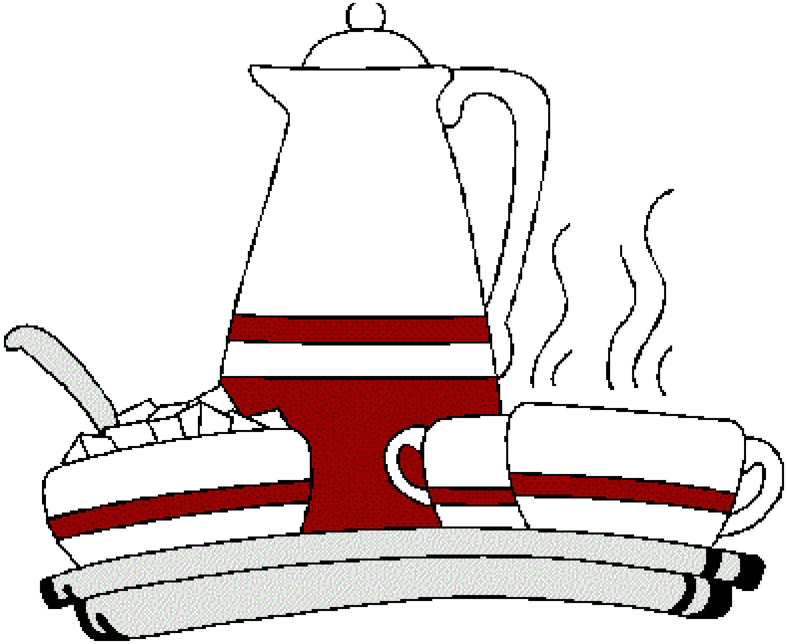
Available examples are restricted to a subset of the supported models of computation.

Newton's craddle

Summary

- Strong trend toward embedded systems (Information processing systems embedded into a larger product)
  - Interfacing to physical environment
  - Resource – constrained
- Large set of requirements for specification languages
- Multi-threading not really ideal
- Search for other models of computations
  - Task graphs
  - Data-flow oriented models (KPN, Simulink)
  - Control-flow oriented models (StateCharts)
  - Shared memory vs. message passing
  - Need to consider real-time

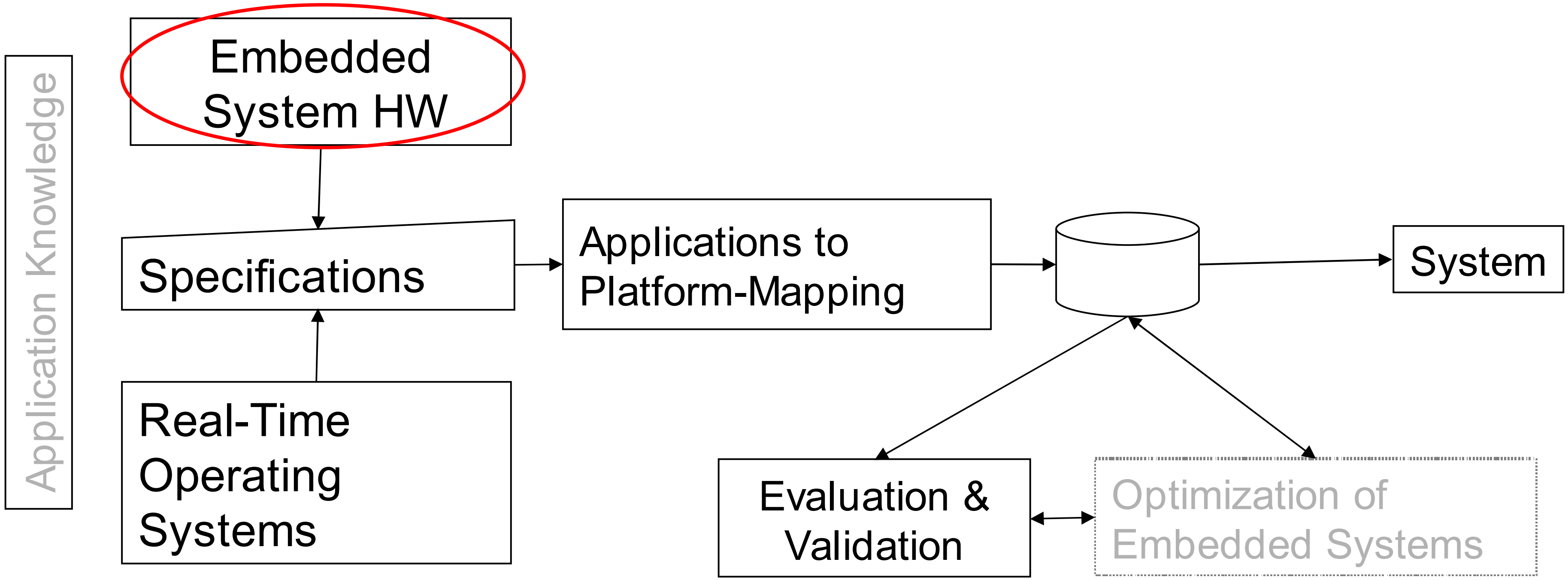
Coffee break (if on schedule)



Embedded Systems Hardware

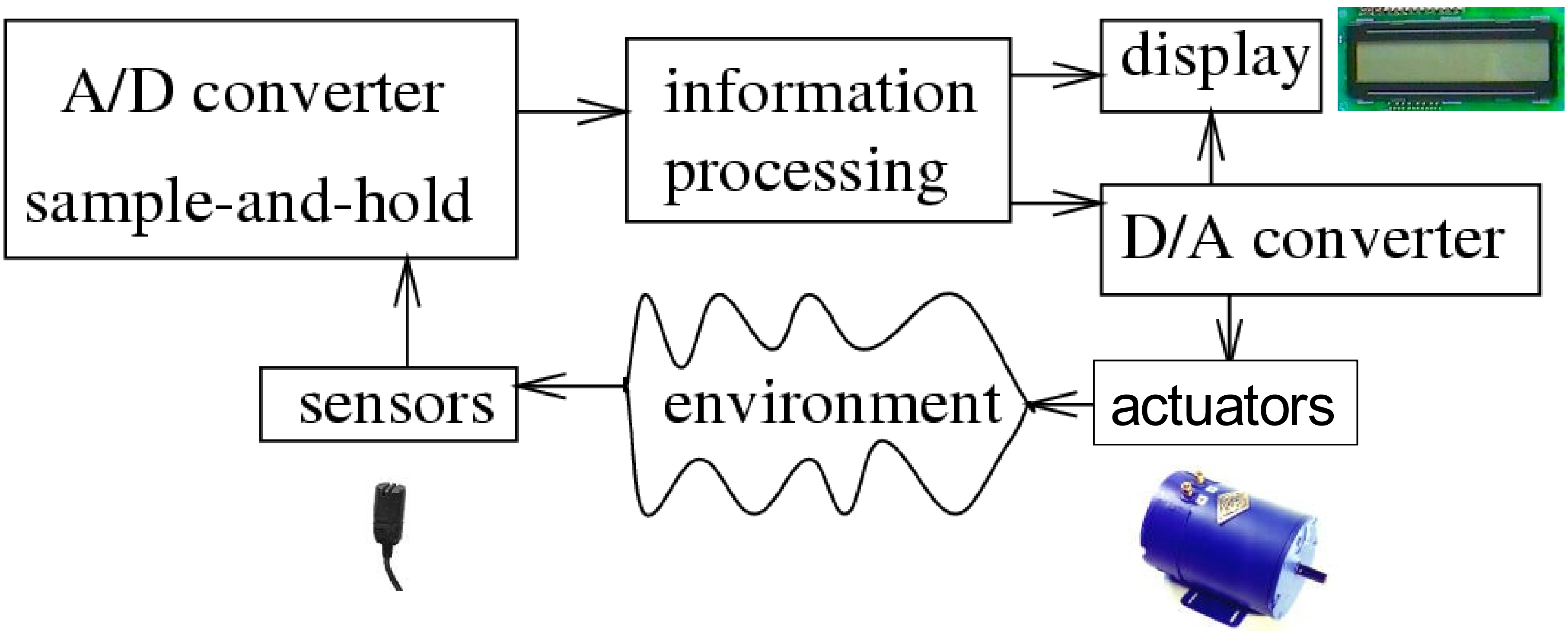
Peter Marwedel
TU Dortmund,
Informatik 12
& ICD e.V.
Germany

Structure of this tutorial



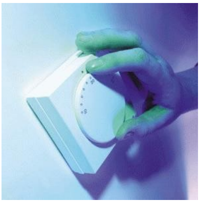
Embedded System Hardware

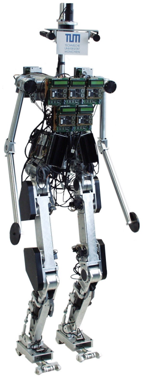
Embedded system hardware is frequently used in a loop („***hardware in a loop***“):



Many examples of such loops

- Heating
- Lights
- Engine control
- Power supply
- …
- Robots





Heating: www.masonsplumbing.co.uk/images/heating.jpg
Robot:: Courtesy and ©: H.Ulbrich, F. Pfeiffer, TU München

Sensors

Processing of physical data starts with capturing it.

Sensors exist for physical/chemical quantities

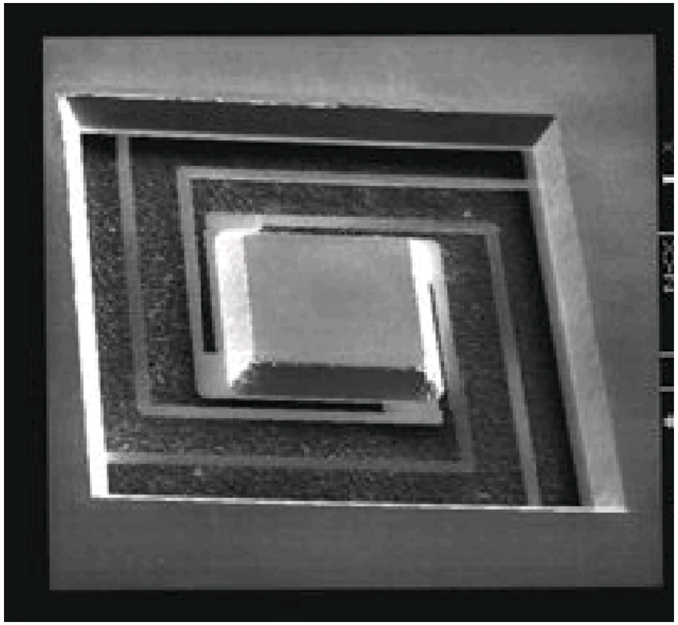
- incl. weight, velocity, acceleration, current, voltage, temperatures etc.
- chemical compounds.

Many physical effects used for construction.

Examples:

- law of induction,
- light-electric effects.

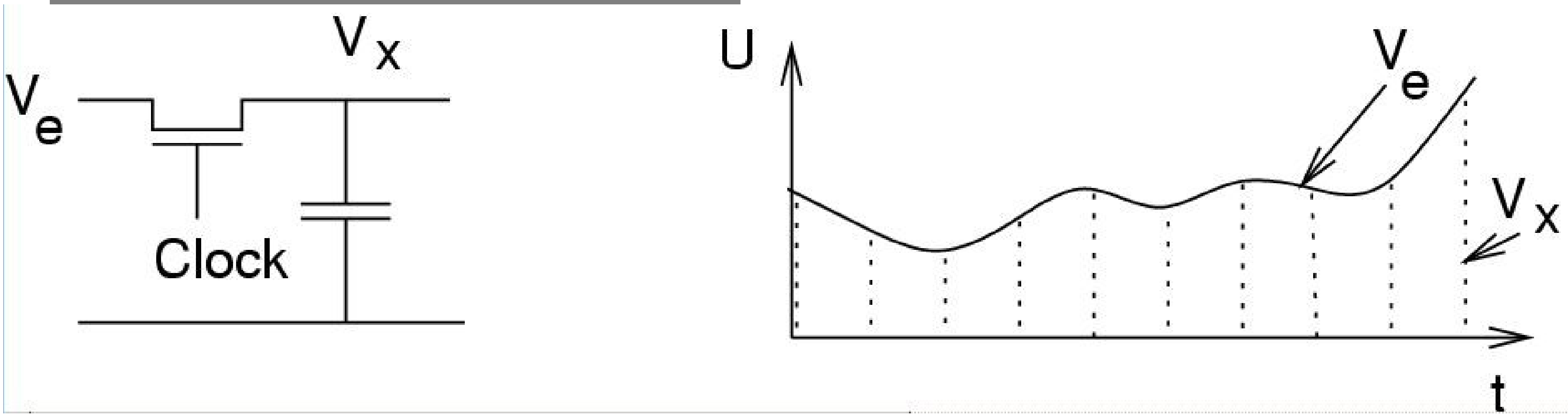
Huge amount of sensors designed.



Courtesy & ©: S. Bütgenbach, TU Braunschweig

Discretization of time

V_e is a mapping $\mathbb{R} \rightarrow \mathbb{R}$



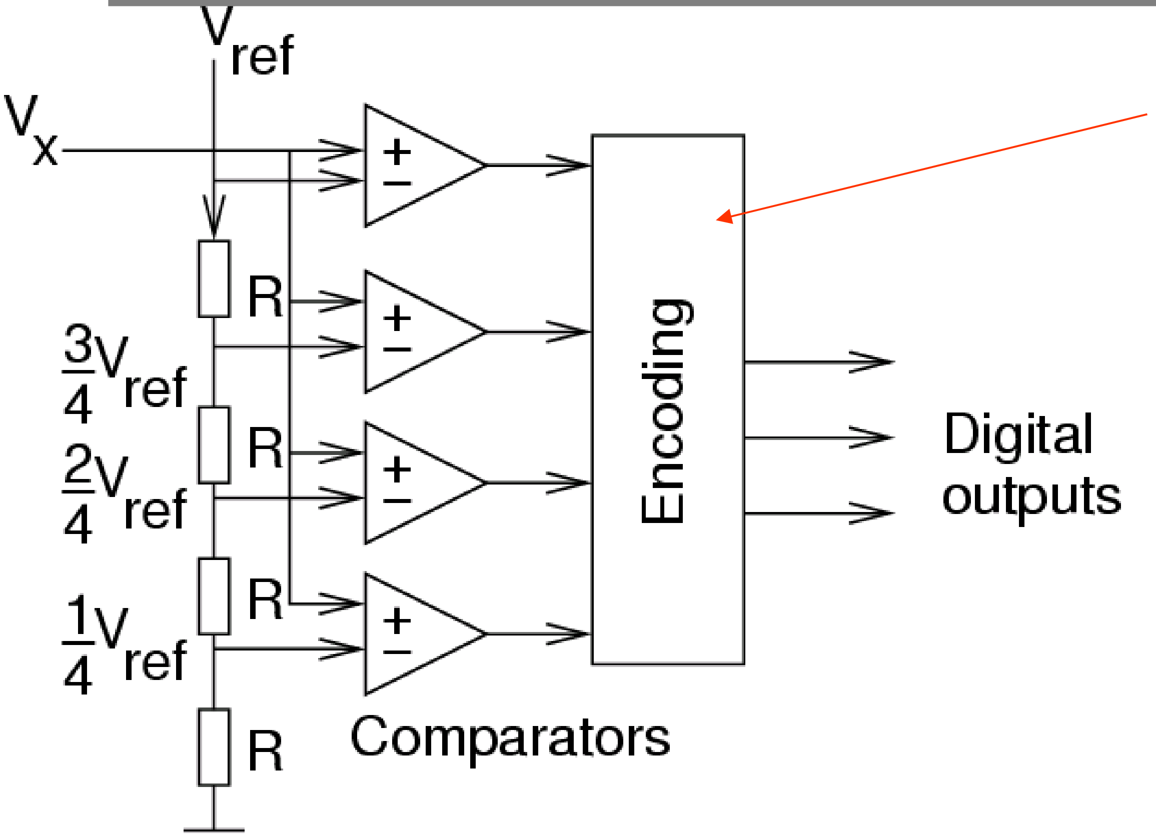
In this course: restriction to digital information processing; Known digital computers can only process discrete time series. ☞ Discrete time; sample and hold-devices.

Ideally: width of clock pulse -> 0

V_x is a **sequence** of values or a mapping $\mathbb{Z} \rightarrow \mathbb{R}$

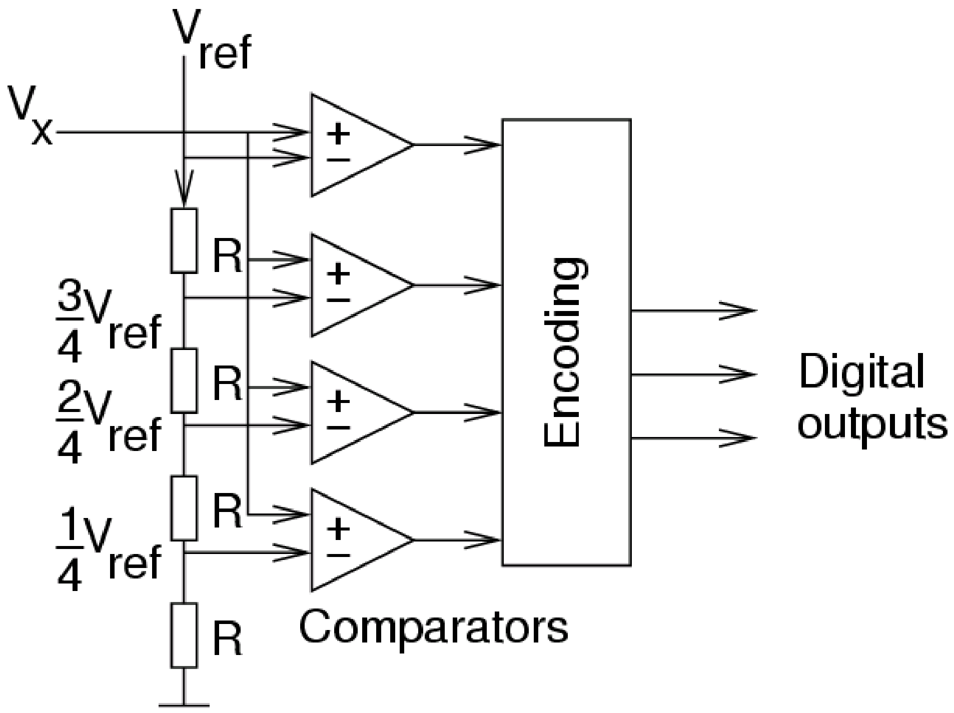
Discretization of values: A/D-converters Flash A/D converter (1)

Digital computers require digital form of physical values
☞A/D-conversion; many methods with different speeds.
Example: 1. Flash A/D converter:



Encodes input number of most significant '1' as an unsigned number, e.g.
"1111" -> "100",
"0111" -> "011",
"0011" -> "010",
"0001" -> "001",
"0000" -> "000"
(Priority encoder).

Discretization of values: A/D-converters
Flash A/D converter (2)



Parallel comparison with reference voltage

Speed: $O(1)$

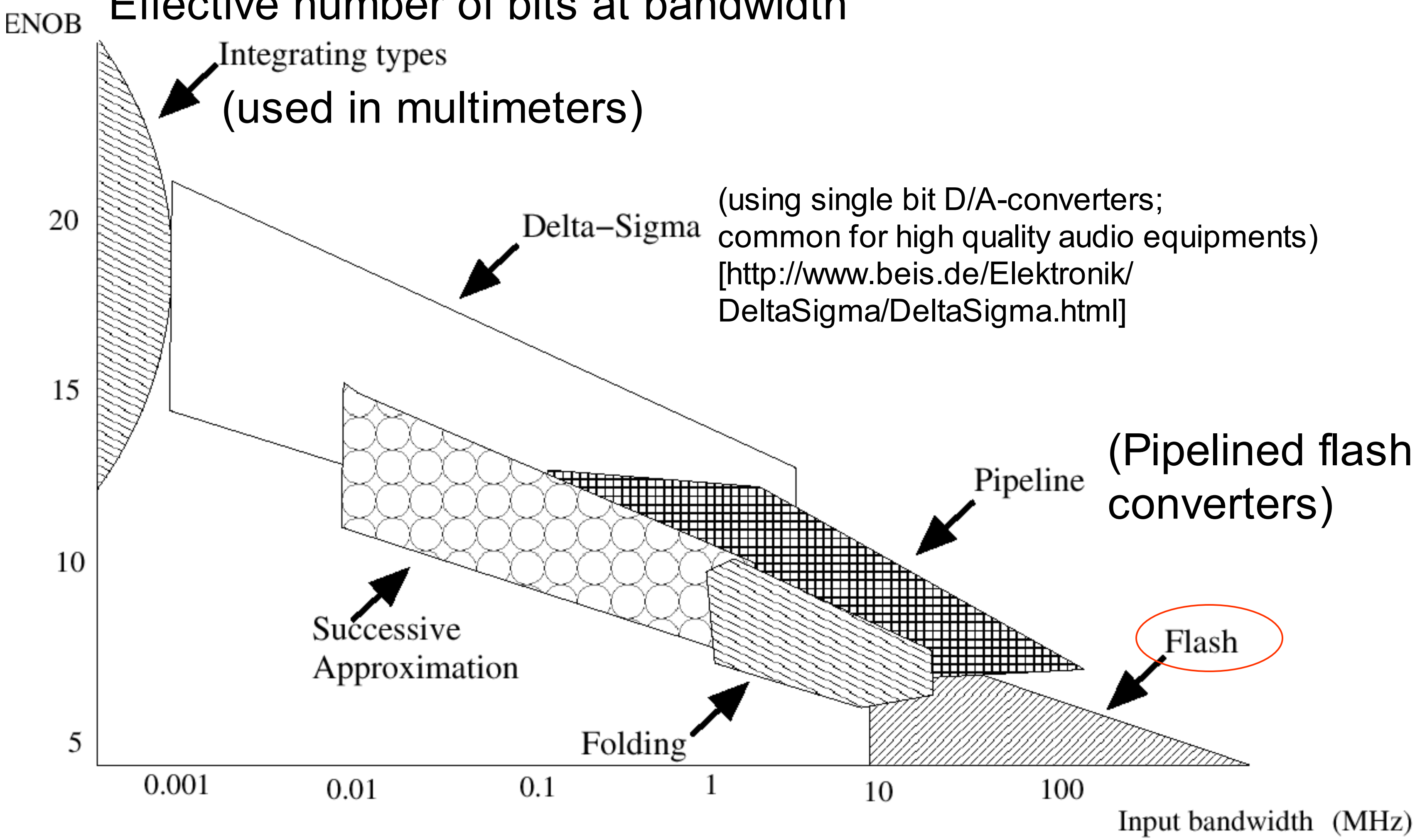
Hardware complexity: $O(n)$

with n= # of distinguished voltage levels

Applications: e.g. in video processing

Application areas for flash and successive approximation converters

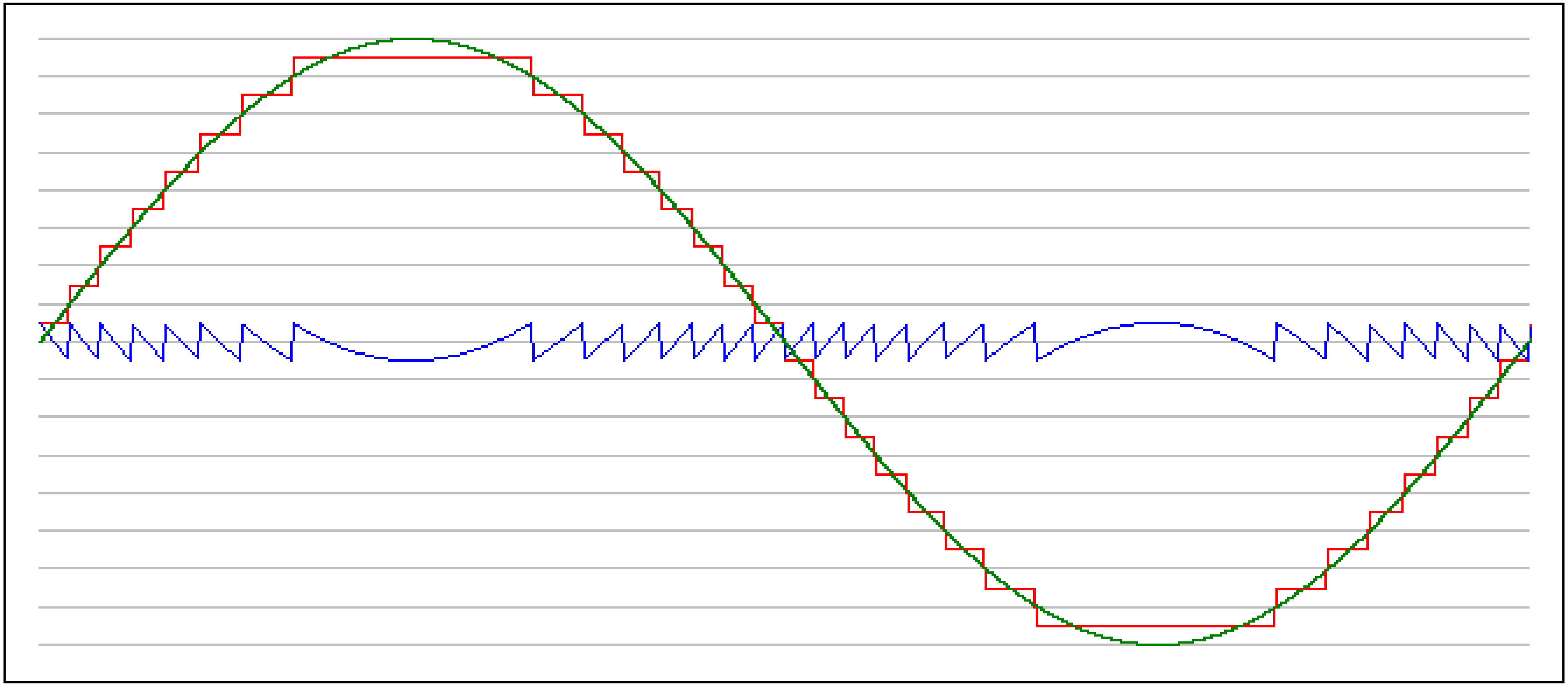
Effective number of bits at bandwidth



[Gielen et al., DAC 2003]

Quantization Noise

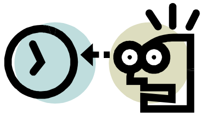
N = (approximated - real signal) called **quantization noise**.
Example: quantization noise for sine wave

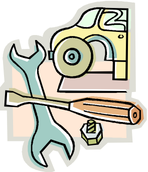
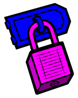


* [http://www.beis.de/Elektronik/DeltaSigma/DeltaSigma.html]

Communication
- Requirements -

- Real-time behavior
- Efficient, economical
  (e.g. centralized power supply)
- Appropriate bandwidth and communication delay
- Robustness
- Fault tolerance
- Maintainability
- Diagnosability
- Security
- Safety



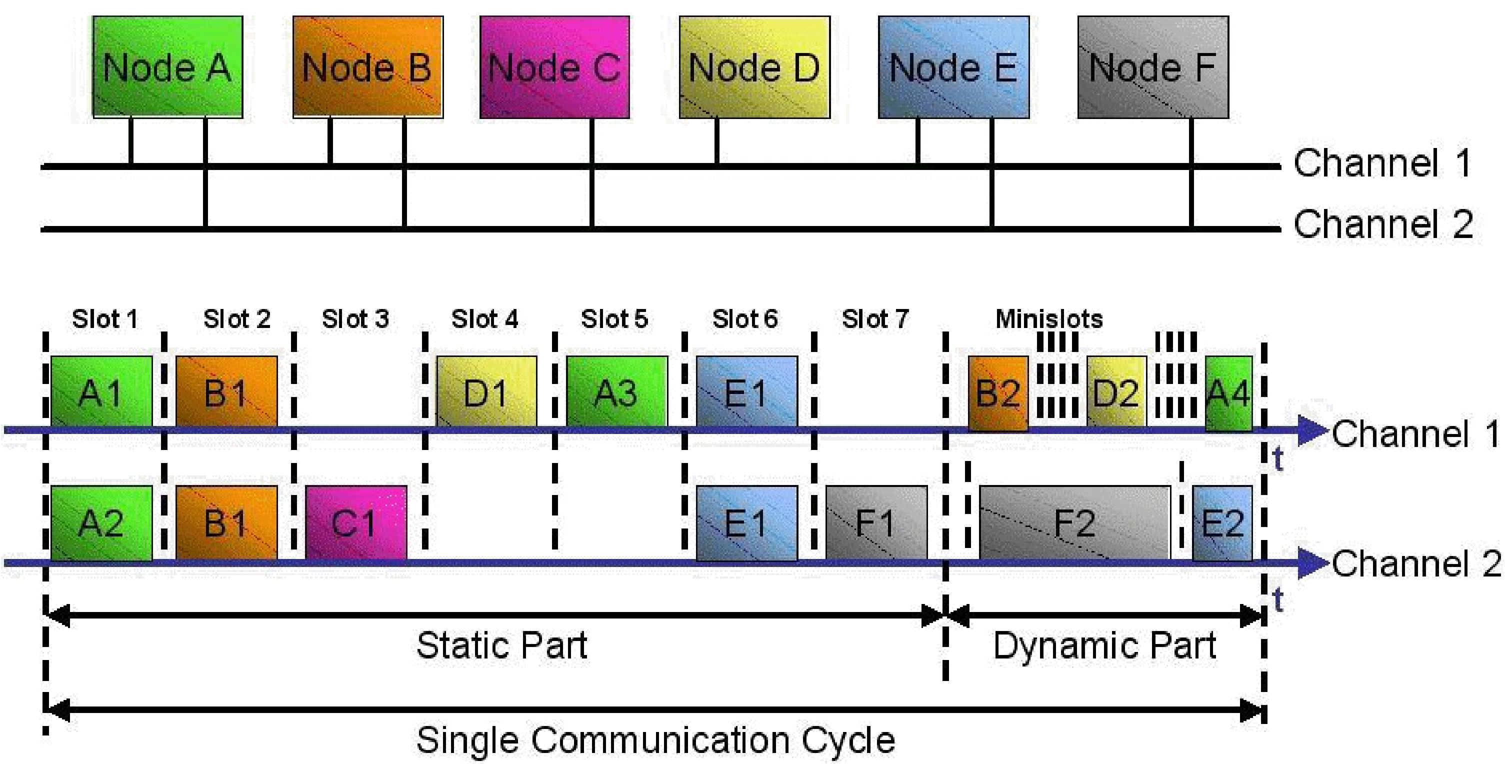




FlexRay

- **D**eveloped by the FlexRay consortium (BMW, Ford, Bosch, DaimlerChrysler, …) Combination of a variant of the TTP and the Byteflight [Byteflight Consortium, 2003] protocol. Specified in SDL.
  - Improved error tolerance and time-determinism
  - Meets requirements with transfer rates >> CAN std. **High data rate can be achieved:**
    – initially targeted for ~ 10Mbit/sec;
    – design allows much higher data rates
  - TDMA (Time Division Multiple Access) protocol: Fixed time slot with exclusive access to the bus
  - Cycle subdivided into a static and a dynamic segment.

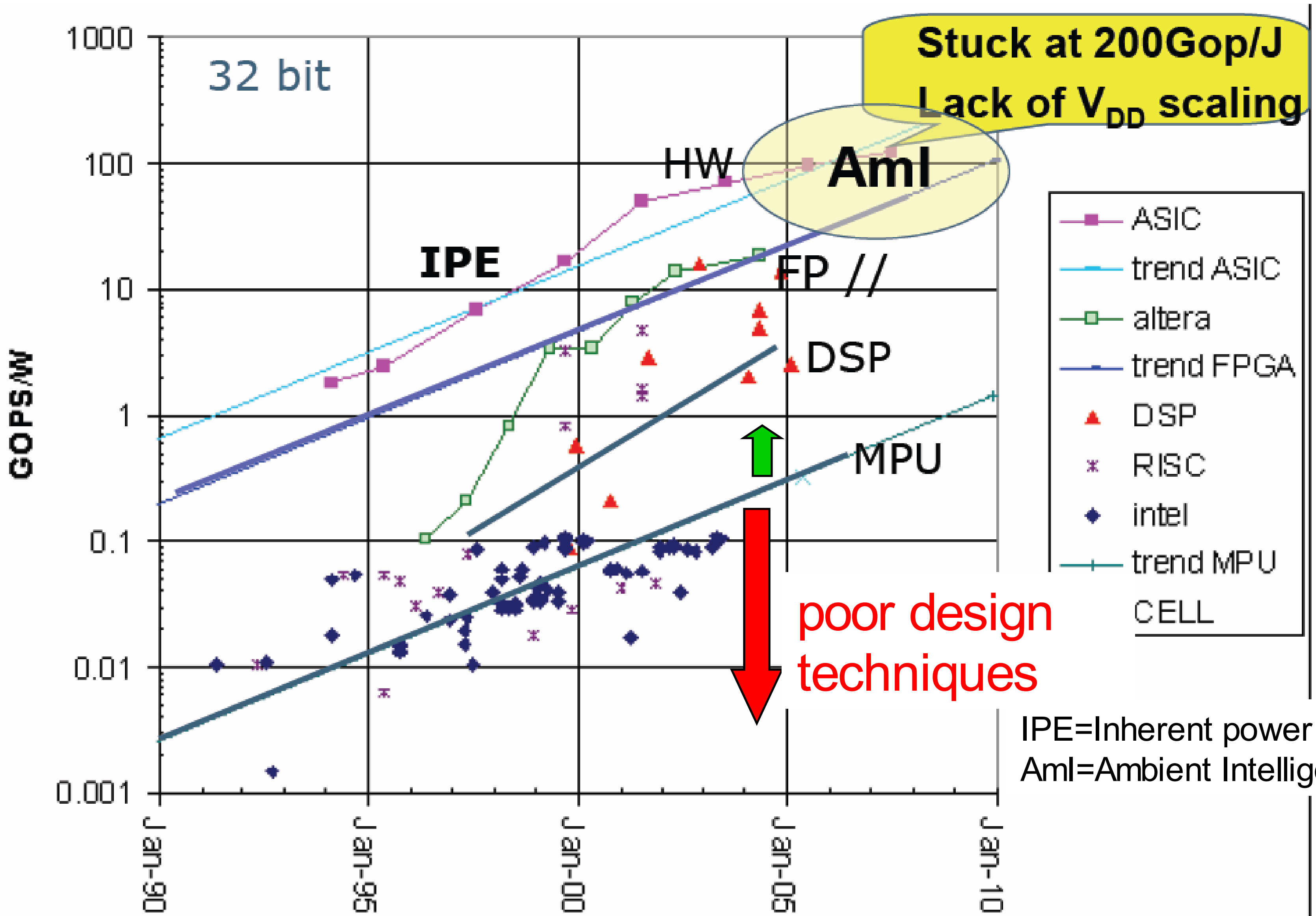
TDMA in FlexRay

Exclusive bus access enabled for short time in each case.
Dynamic segment for transmission of variable length information.
Fixed priorities in dynamic segment: Minislots for each potential sender.
Bandwidth used only when it is actually needed.



http://www.tzm.de/FlexRay/FlexRay_Introduction.html

Importance of Energy Efficiency

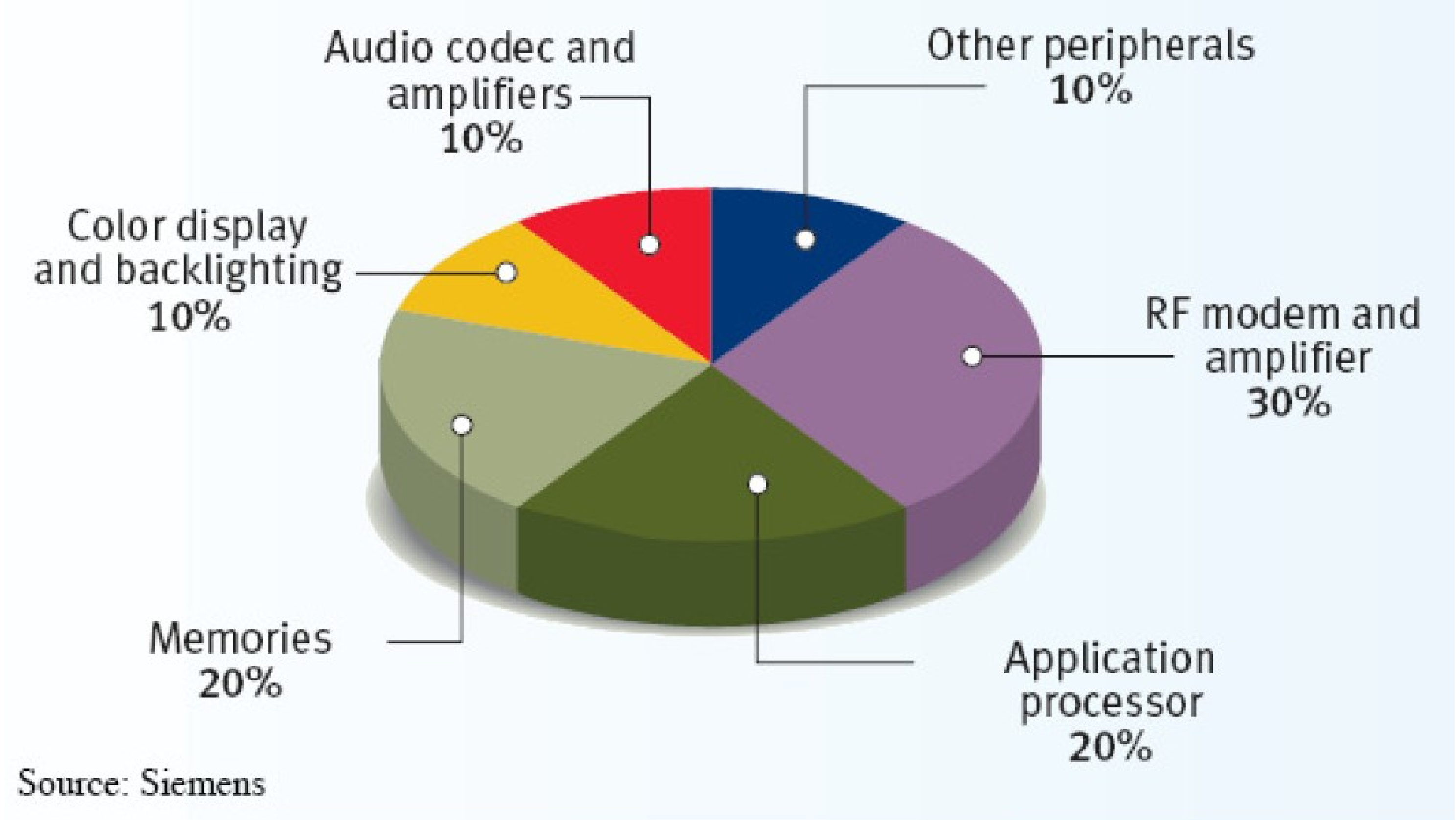


Courtesy: Philips
© Hugo De Man, IMEC, 2007

IPE=Inherent power efficiency
AmI=Ambient Intelligence

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

Energy consumption in mobile devices

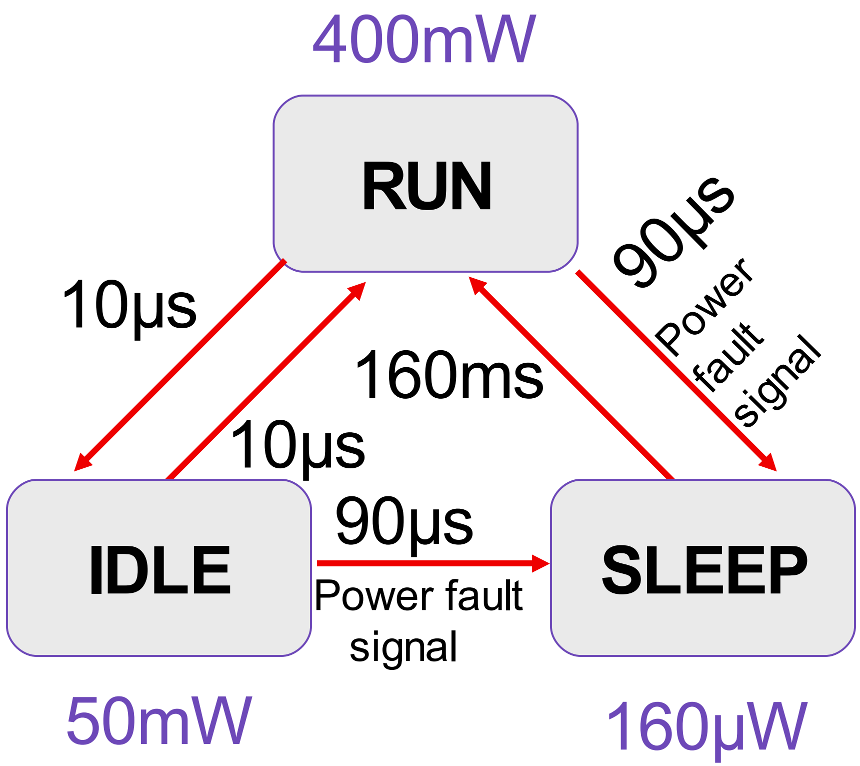


[O. Vargas (Infineon Technologies): Minimum power consumption in mobile-phone memory subsystems; Pennwell Portable Design - September 2005;] Thanks to Thorsten Koch (Nokia/ Univ. Dortmund) for providing this source.

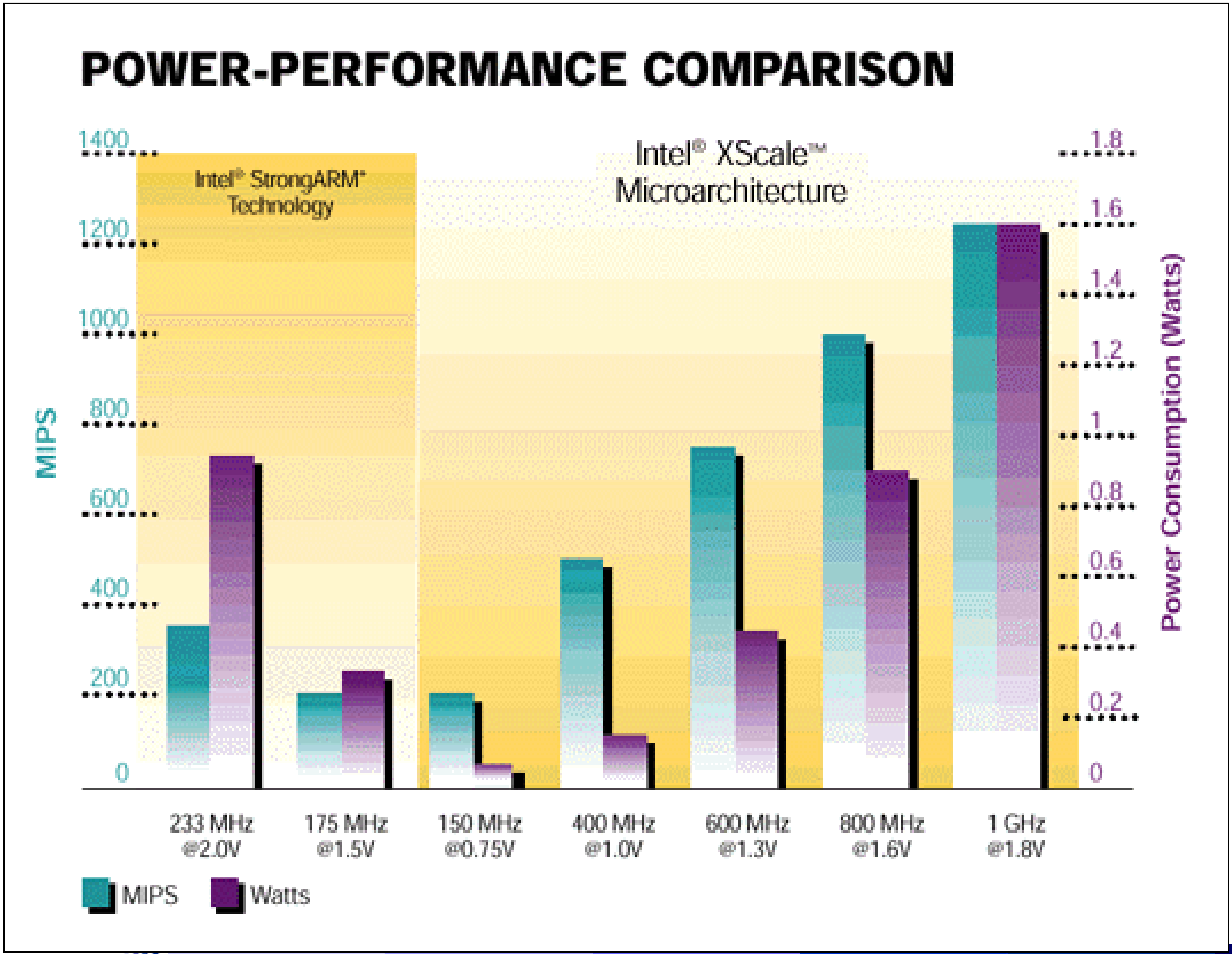
Dynamic power management (DPM)

Example: STRONGARM SA1100

RUN: operational
IDLE: a sw routine may stop the CPU when not in use, while monitoring interrupts
SLEEP: Shutdown of on-chip activity



Variable-voltage/frequency example: INTEL Xscale

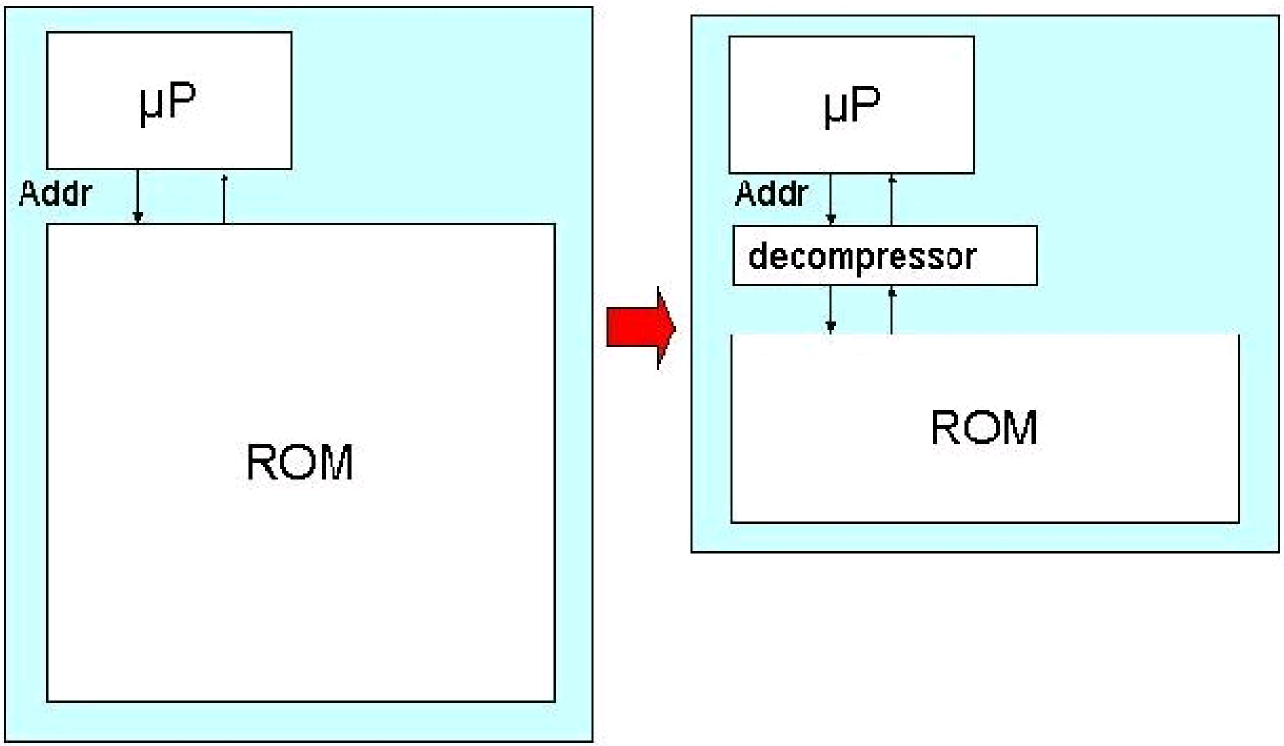


OS should schedule distribution of the energy budget.

From Intel's Web Site

Key requirement #2: Code-size efficiency

- **CISC machines**: RISC machines designed for run-time-, not for code-size-efficiency
- **Compression techniques:** key idea

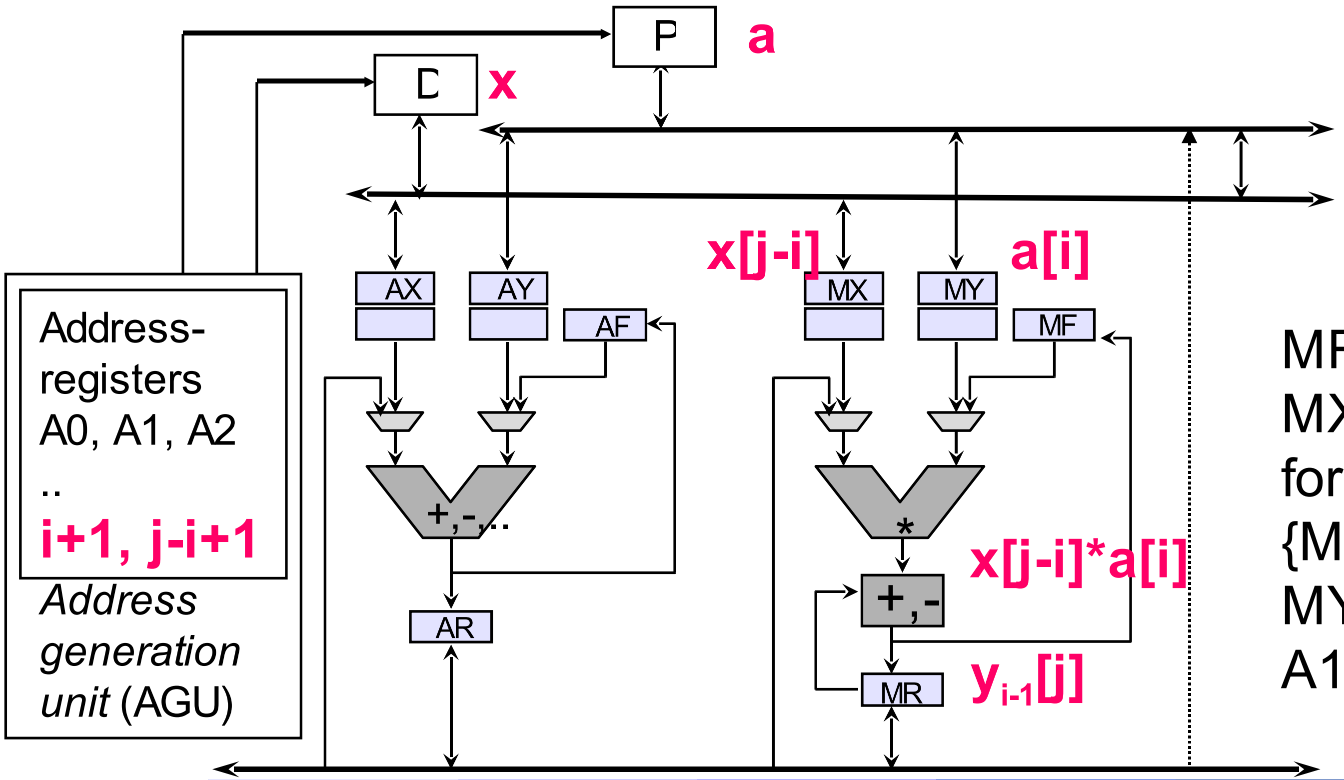


Key requirement #3: Run-time efficiency
- Domain-oriented architectures -

Application: $y[j] = \sum_{i=0}^{n-1} x[j-i]*a[i]$

$\forall i: 0 \leq i \leq n-1: y_i[j] = y_{i-1}[j] + x[j-i]*a[i]$

Architecture: Example: Data path ADSP210x

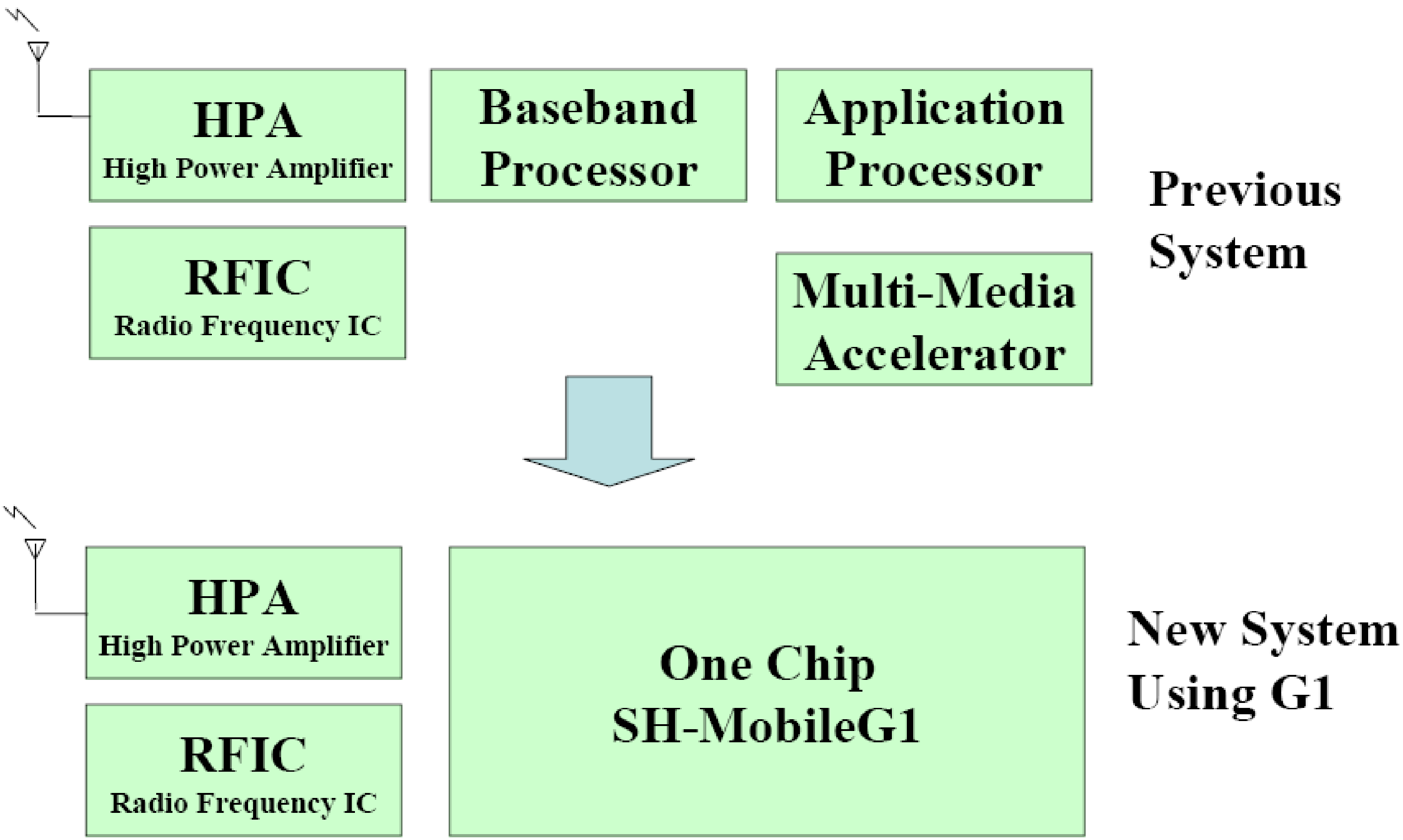


Application maps nicely onto architecture

MR:=0; A1:=1; A2:=n-2;
MX:=x[n-1]; MY:=a[0];
for (j:=1 to n)
{MR:=MR+MX*MY;
MY:=a[A1]; MX:=x[A2];
A1++; A2--}

Trend: multiprocessor systems-on-a-chip (MPSoCs)

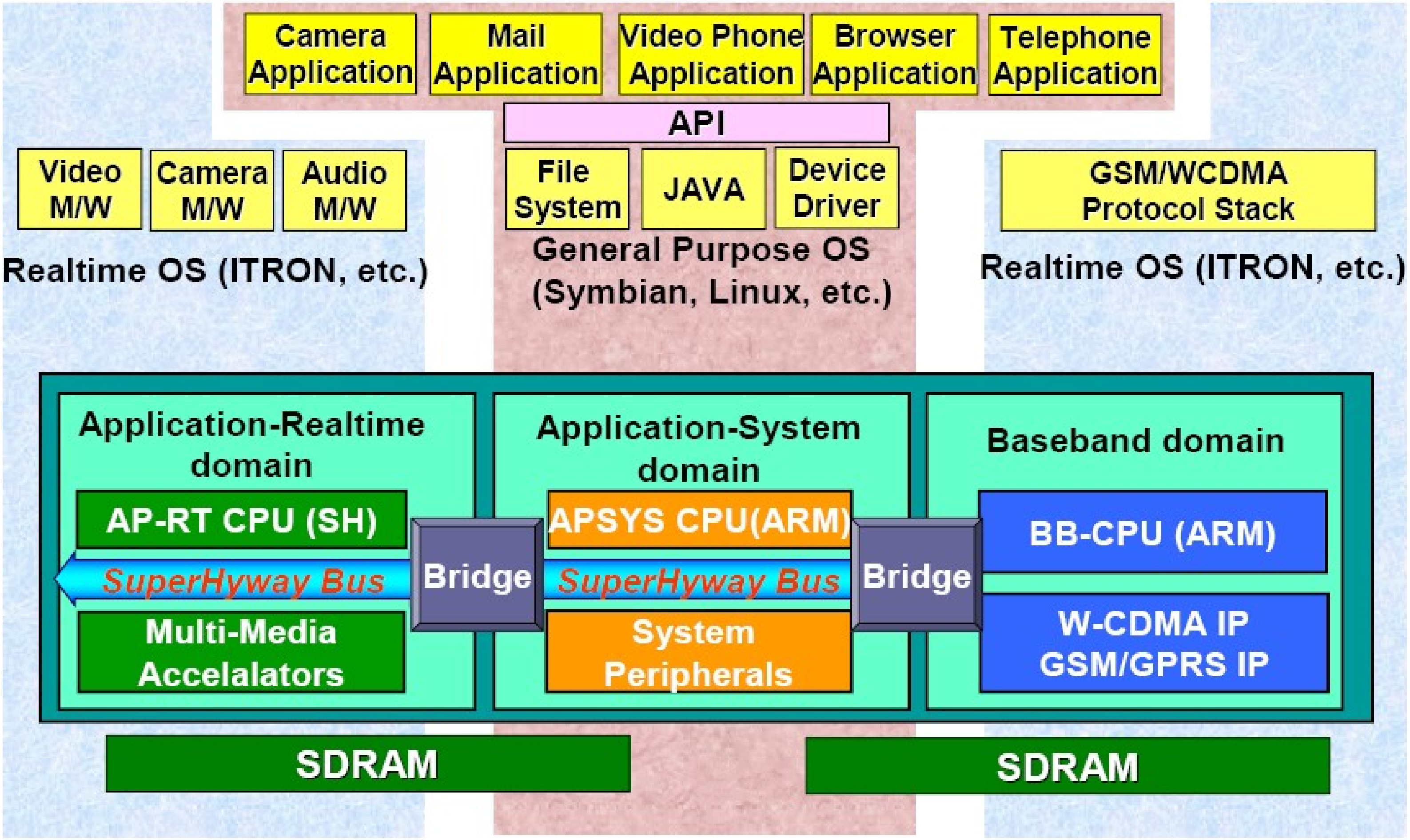
3G Multi-Media Cellular Phone System



http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

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

A Sample of System Architecture using G1



http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Reconfigurable Logic

Full custom chips may be too expensive, software 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)

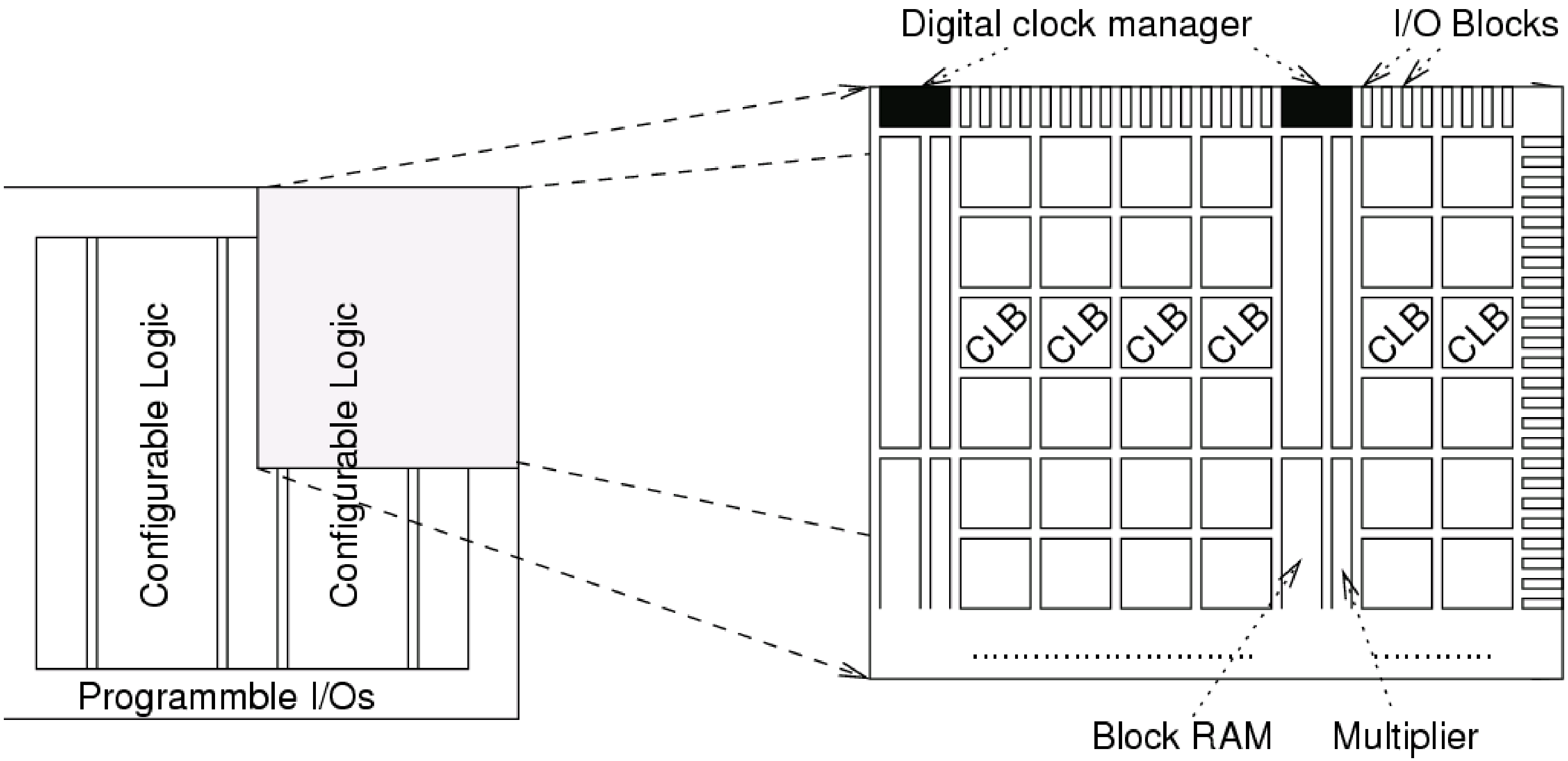
Applications: bit-oriented algorithms like

- 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

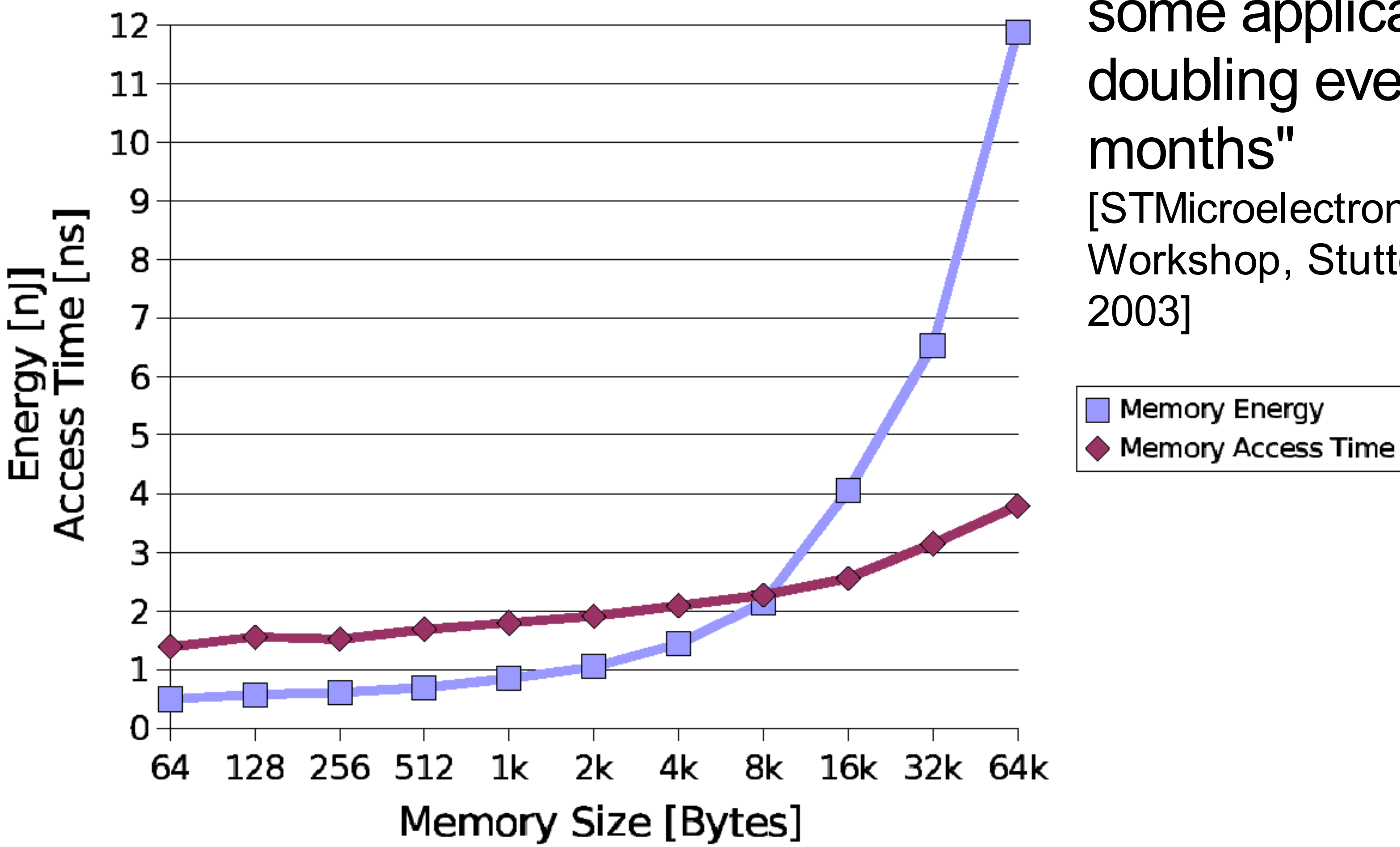
Floor-plan of VIRTEX II FPGAs



CLB's: programmable logic functions + registers

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

Example (CACTI Model):

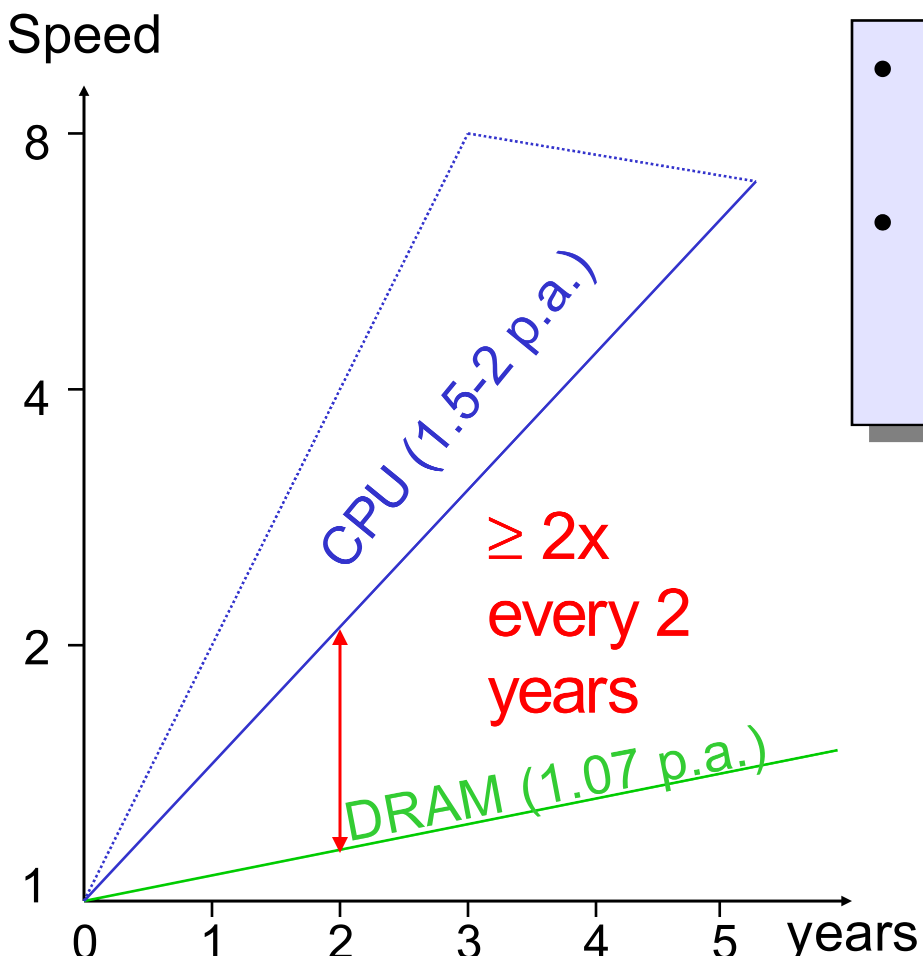


"Currently, the size of some applications is doubling every 10 months"

[STMicroelectronics, Medea+ Workshop, Stuttgart, Nov. 2003]

Access-times will be a problem

Speed gap between processor and main DRAM increases

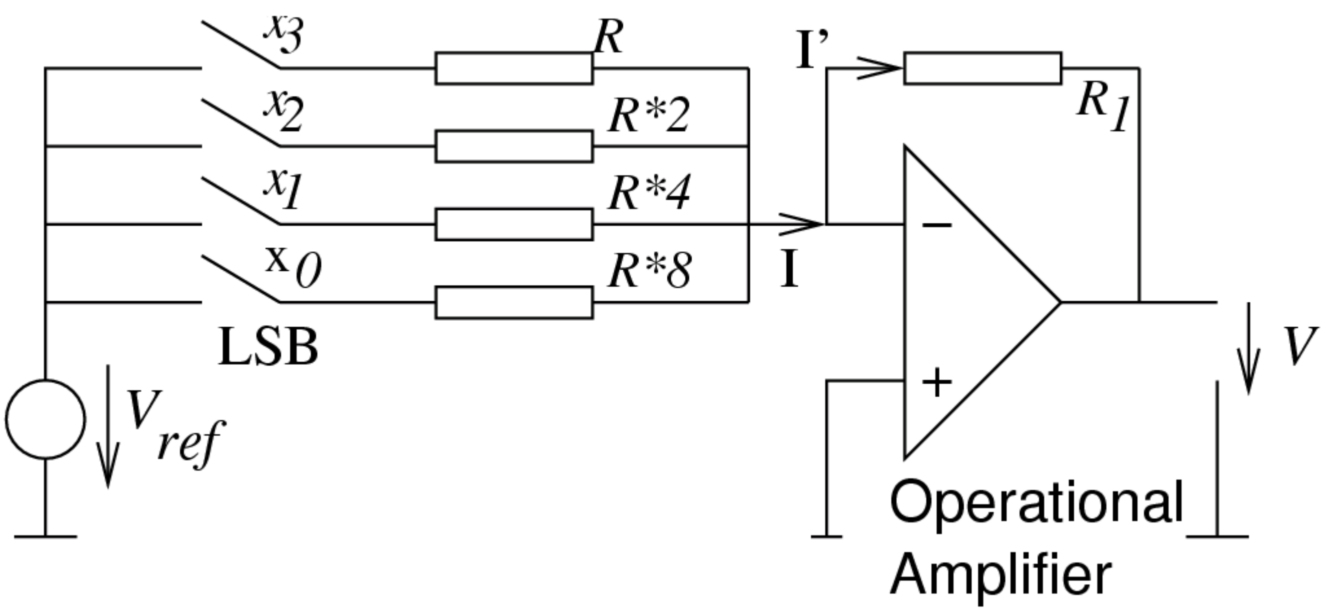


- early 60ties (Atlas): page fault ~ 2500 instructions
- 2002 (2 GHz µP): access to DRAM ~ 500 instructions

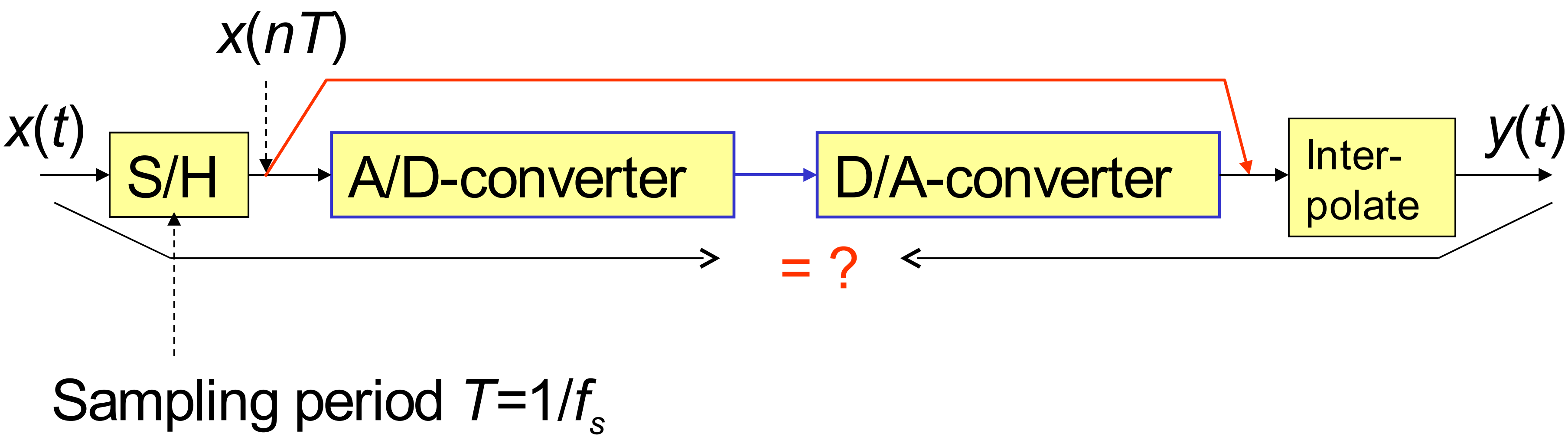
[P. Machanik: Approaches to Addressing the Memory Wall, TR Nov. 2002, U. Brisbane]

Digital-to-Analog (D/A) Converters

Various types, can be quite simple, e.g.:



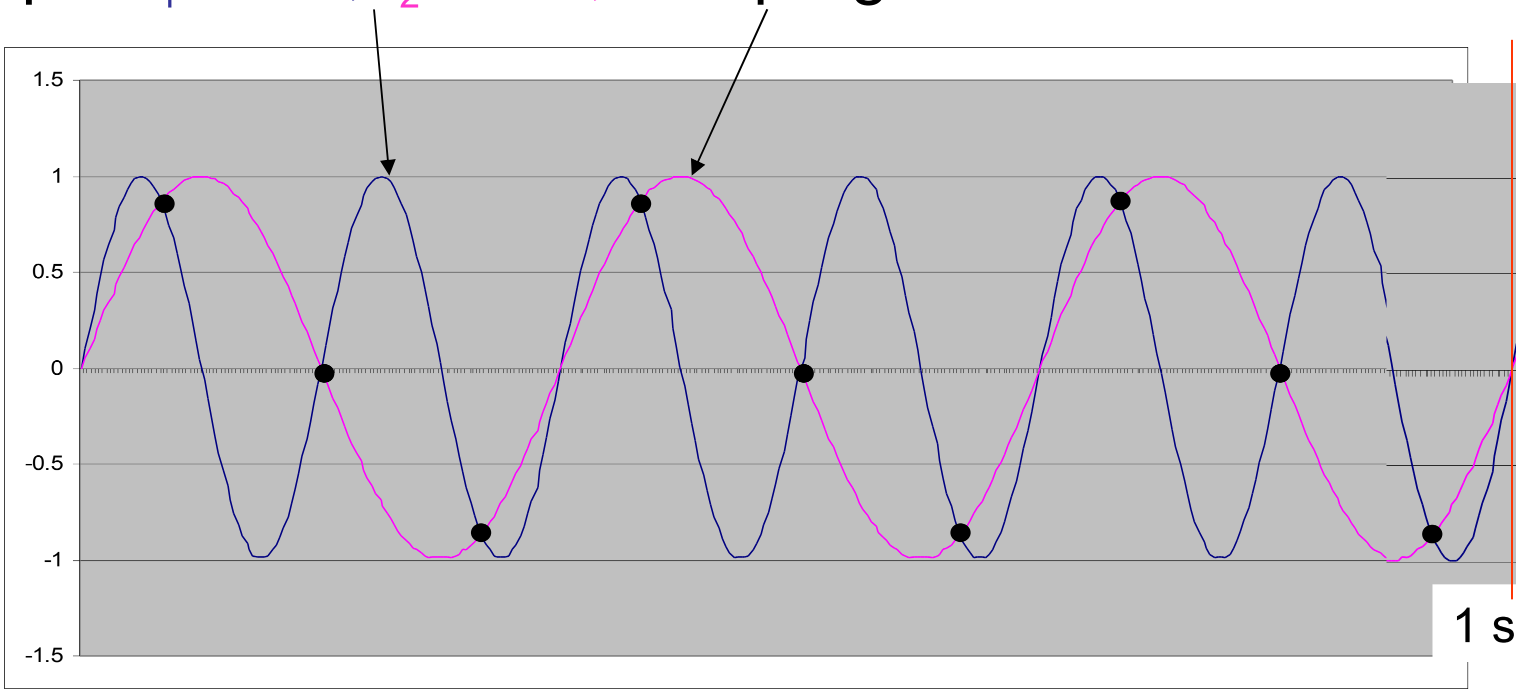
Possible to reconstruct original signal from discrete time series?



Limitations: example

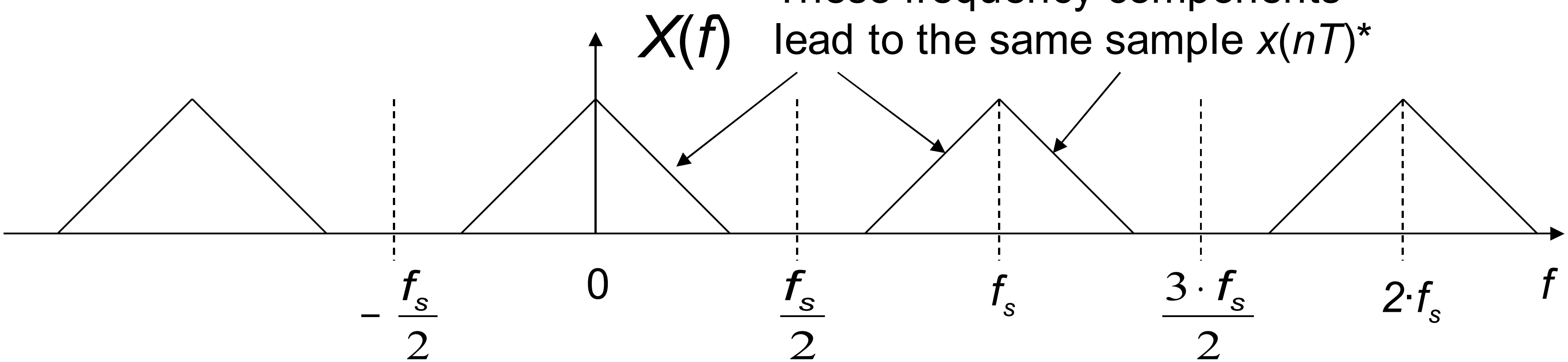
Frequency components > $f_s/2$ cannot be distinguished from frequency components < $f_s/2$.

Example: f_1: 6 Hz; f_2: 3 Hz; Sampling: 9 Hz



$f_s/2$=4.5 Hz; f_1-$f_s/2$=$f_s/2$-f_2=1.5 Hz; samples for frequencies $f_s/2 \pm c$ identical

Frequencies represented by time series *x*(*nT*)



☞Reconstructing a time-continuous signal *x*(*t*) requires that we **know** that only one of these frequency bands is used.

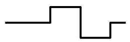
Assumption: we consider signals with $f \in [0..f_u]$ ("*base-band*") only.

☞Reconstructing a time-continuous signal after sampling with a sample frequency of f_s requires the signal to be bandwidth-limited to $f<f_s/2$. $f_s/2$ is the **Nyquist frequency**.

* Can be shown in general by considering symmetries of sine functions

How to generate good values for times $t \neq$ sampling times nT? (1)

A/D-converters do not interpolate between samples.



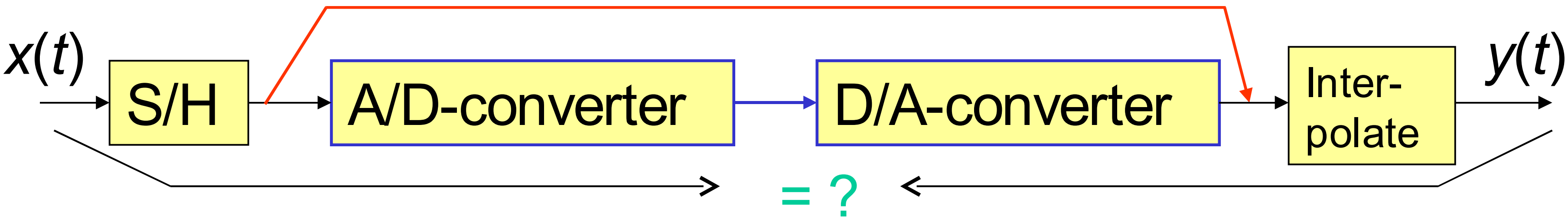
They generate step functions ☞ certainly not the original functions.

Theorem (Shannon and others):

Exact reconstruction of a continuous-time base-band signal from its samples is possible if the signal is band-limited and the sampling frequency is greater than twice the signal bandwidth.

$f = f_s/2$ has to be excluded as well.

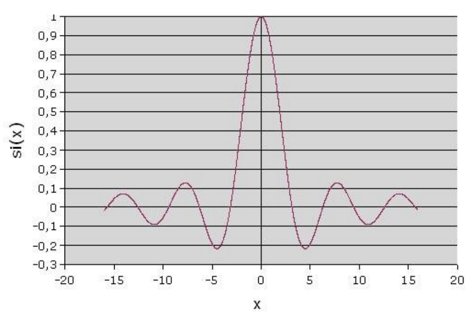
How to generate good values for times $t \neq$ sampling times nT? (2)



The necessary interpolation is based on the *sinc* function:

$$x(t) = y(t) := \sum_{k=-\infty}^{\infty} x_k \prod_{j\in\mathbb{Z},\, j\neq k} \frac{t - jT}{kT - jT} = \sum_{k=-\infty}^{\infty} x(kT)\, \mathrm{sinc}(t/T - k) \quad (1)$$

$$\mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$



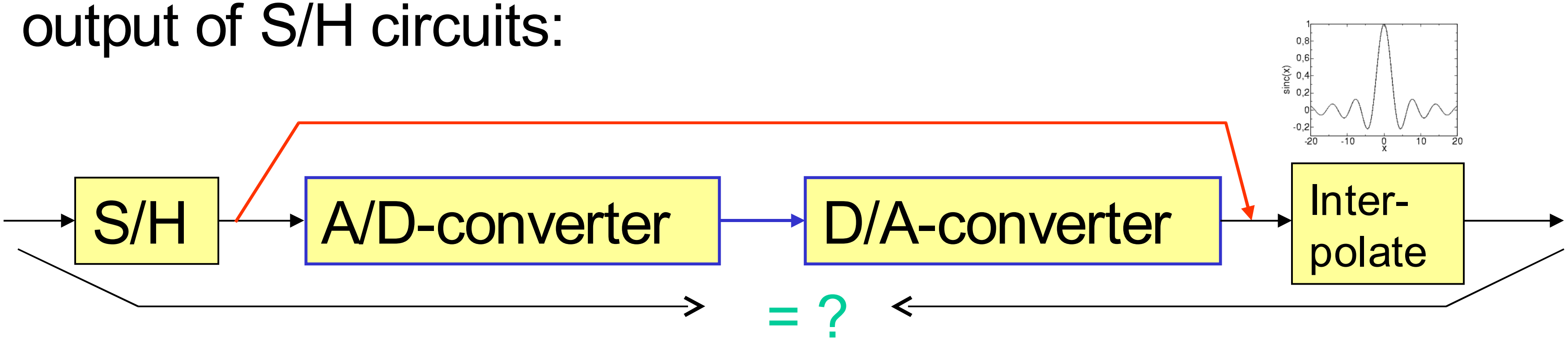
(1) is due to Whittacker (1915)

Each sample $x(nT)$ influences its neighborhood $t \neq nT$ with a weighting factor $sinc(t/T-k)$, which is zero at other sampling times.

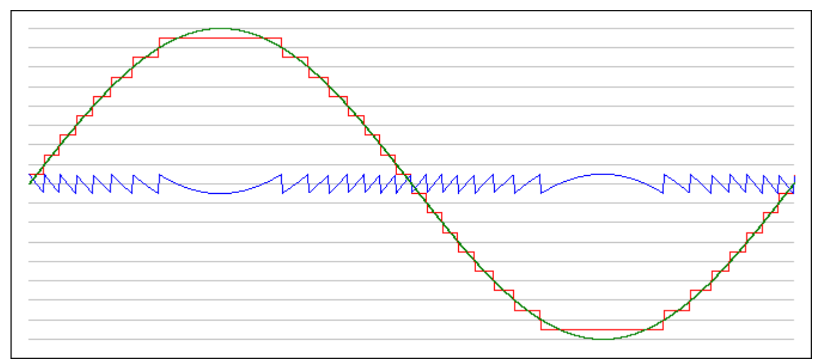
Equations & graphics: de.wikipedia.org

Impact of quantization noise

We can only hope to reconstruct originals signals from the output of S/H circuits:



Signals from the output of the A/D-converter contain quantization noise;



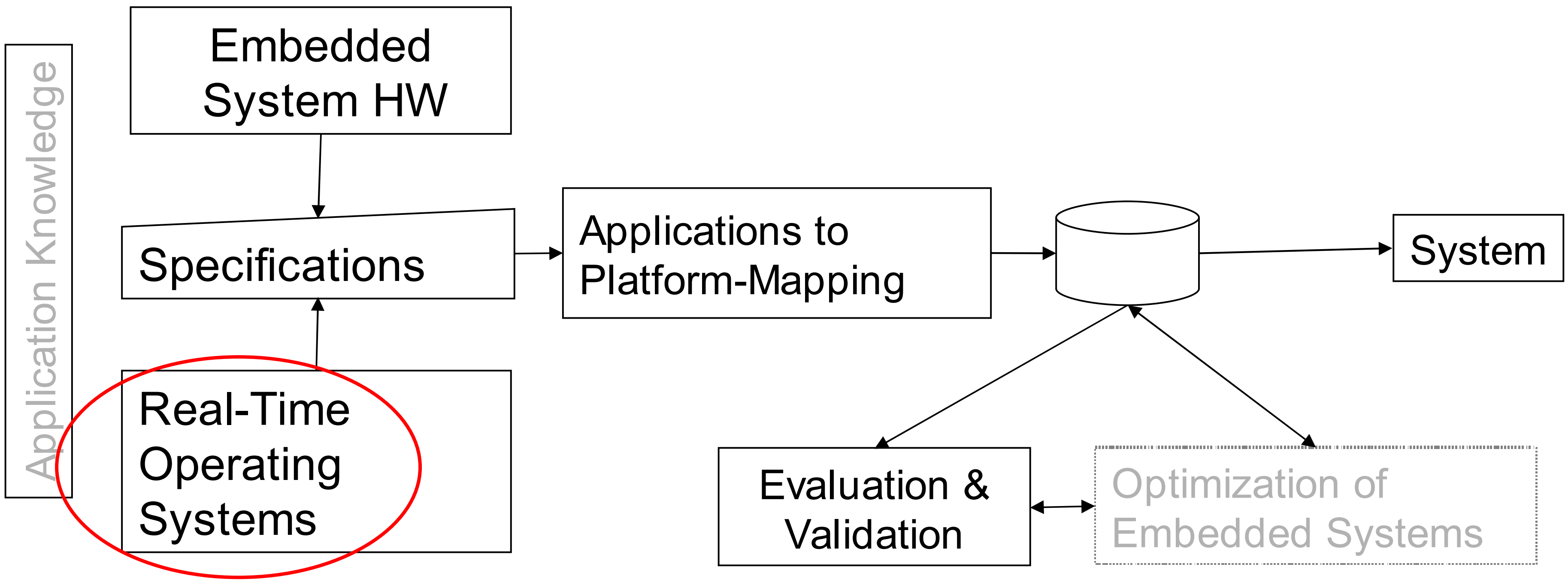
* [http://www.beis.de/Elektronik/DeltaSigma/DeltaSigma.html]

This noise cannot be removed.

Embedded Operating Systems

Peter Marwedel
TU Dortmund,
Informatik 12
& ICD e.V.
Germany

Structure of this tutorial



Embedded operating systems
- Requirement: Configurability -

Configurability

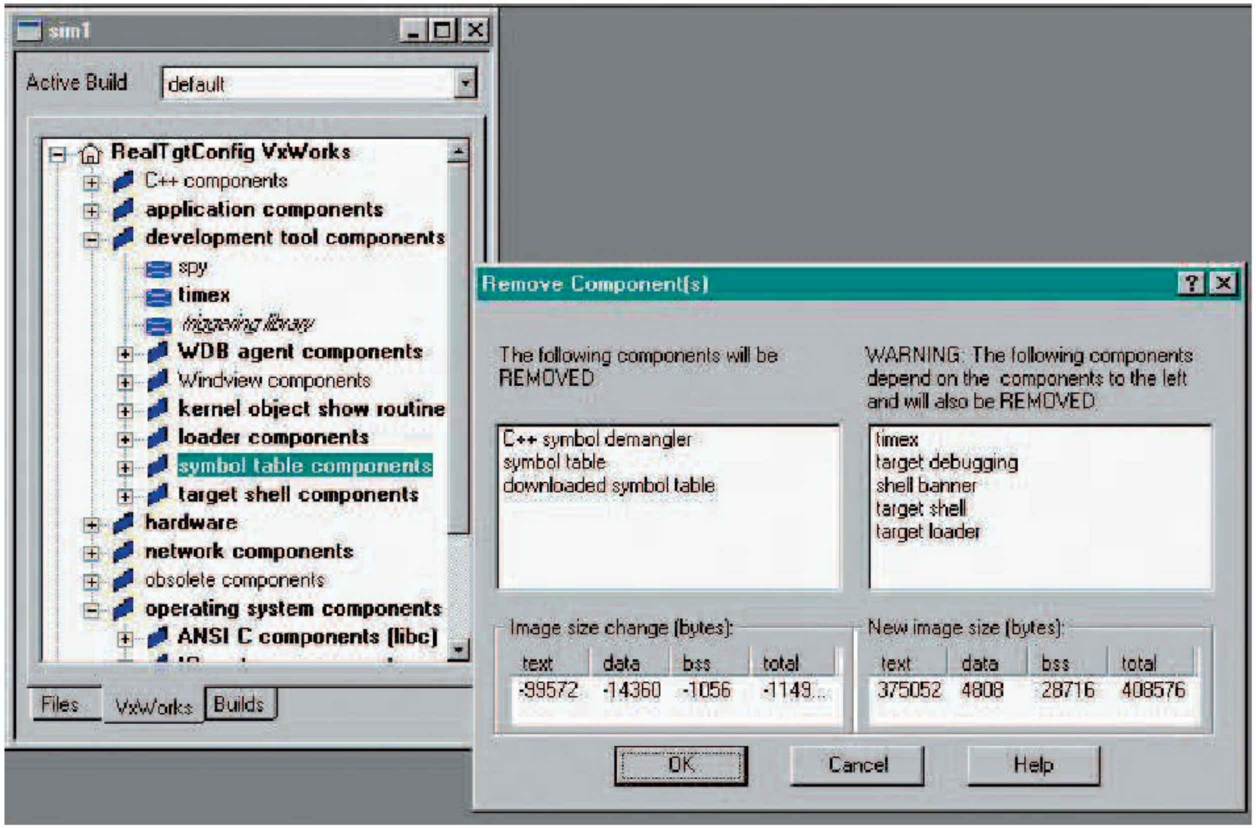
No single RTOS will fit all needs, no overhead for unused functions tolerated ☞ configurability needed.

- simplest form: remove unused functions (by linker ?).
- Conditional compilation (using #if and #ifdef commands).
- Dynamic data might be replaced by static data.
- Advanced compile-time evaluation useful.
- Object-orientation could lead to a derivation subclasses.

Verification a problem with many derived OSs:

- Each derived OS must be tested thoroughly;
- potential problem for eCos (open source RTOS from Red Hat), including 100 to 200 configuration points [Takada, 01].

Example: Configuration of VxWorks



Automatic dependency analysis and size calculations allow users to quickly custom-tailor the VxWORKS operating system.

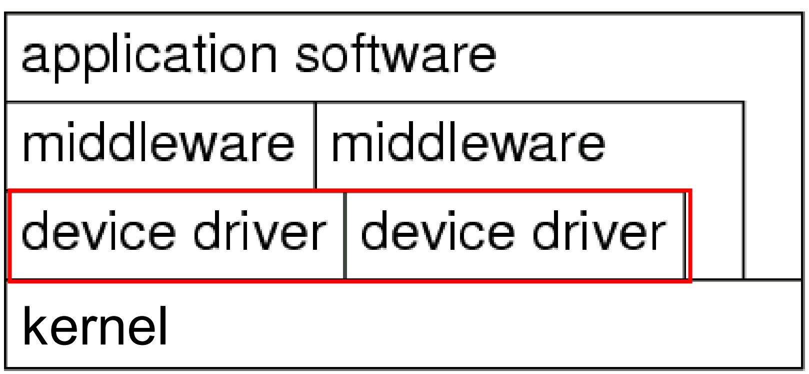
© Windriver

http://www.windriver.com/products/development_tools/ide/tornado2/tornado_2_ds.pdf

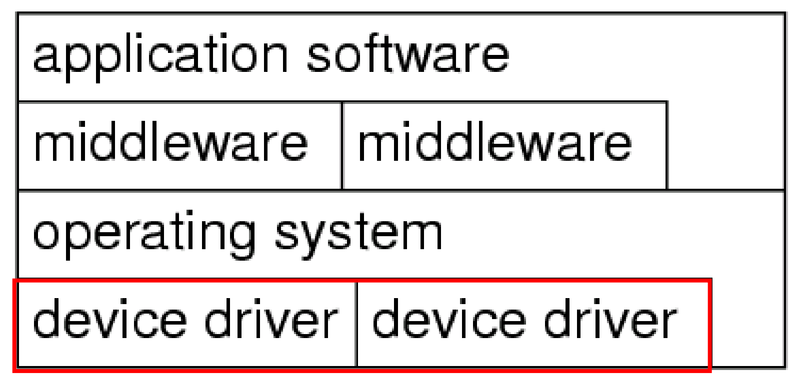
Embedded operating systems
-Requirement: Disc and network handled by tasks-

- Disc & network handled by tasks instead of integrated drivers. Relatively **slow** discs & networks can be handled by tasks.
- Many ES without disc, a keyboard, a screen or a mouse.
- **Effectively no device that needs to be supported by all versions of the OS**, except maybe the system timer.

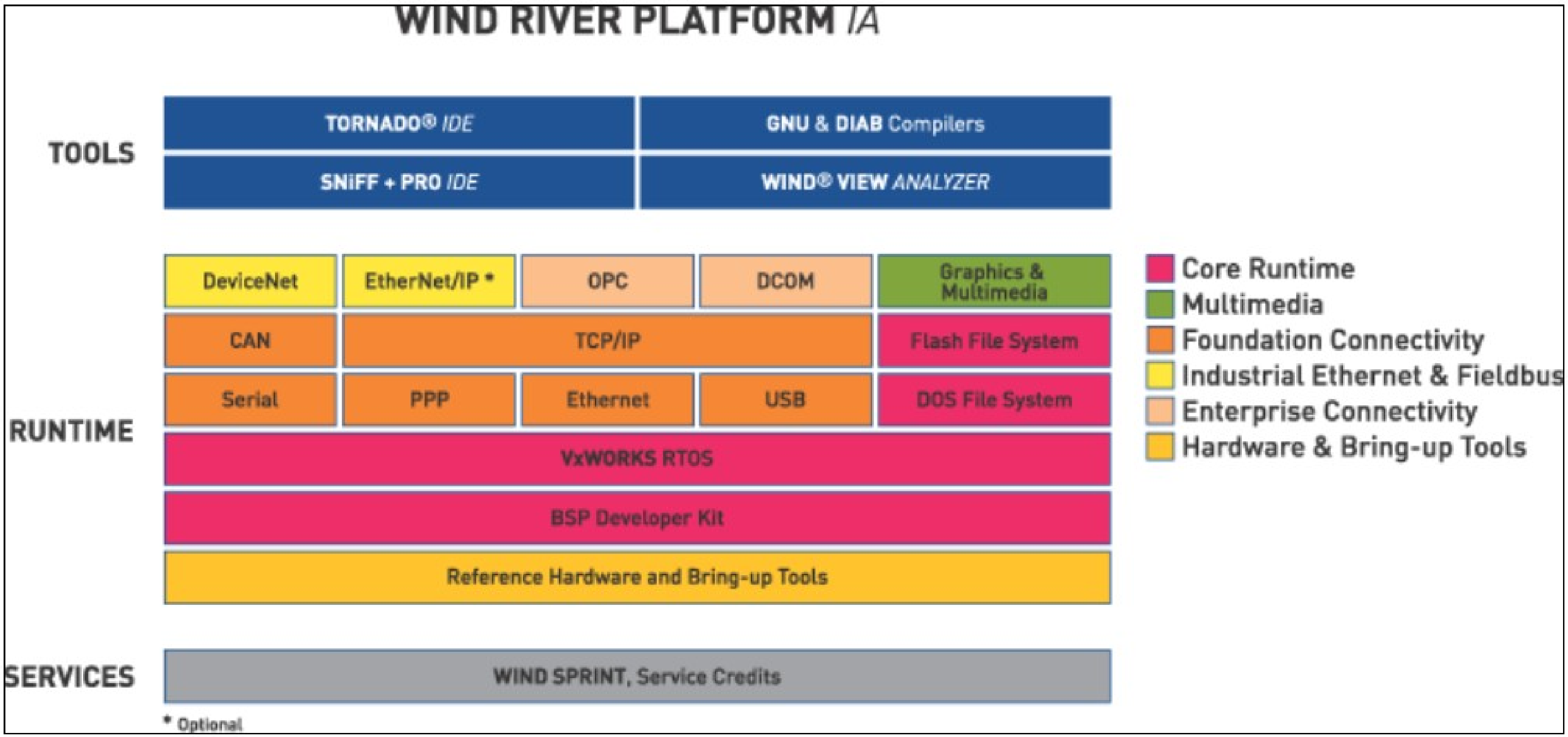
Embedded OS



Standard OS



Example: WindRiver Platform Industrial Automation



© Windriver

Embedded operating systems
- Requirement: Protection is optional-

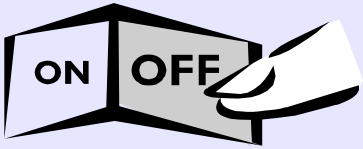
Protection mechanisms not always necessary:
ES typically designed for a single purpose,
untested programs rarely loaded, SW considered reliable.
(However, protection mechanisms may be needed for safety and security reasons).

Privileged I/O instructions not necessary and tasks can do their own I/O.

Example: Let **switch** be the address of some switch
Simply use

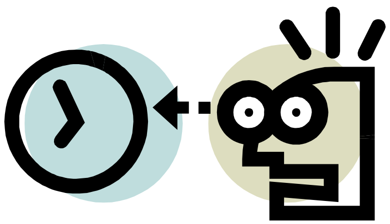
load register,switch

instead of OS call.

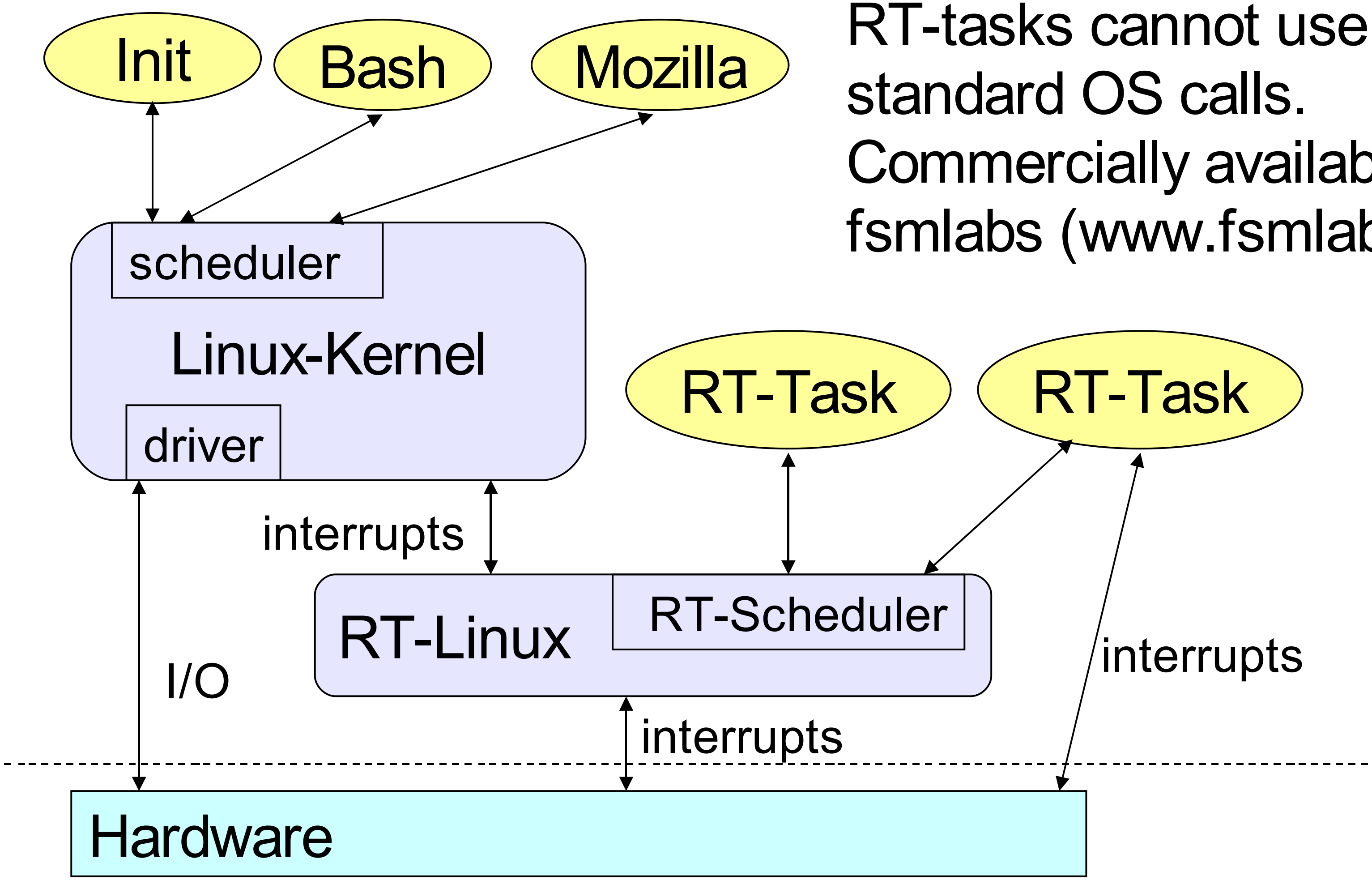


Embedded operating systems - Requirement: Real-time capability-

Many embedded systems are real-time (RT) systems and, hence, the OS used in these systems must be **real-time operating systems (RTOSes).**



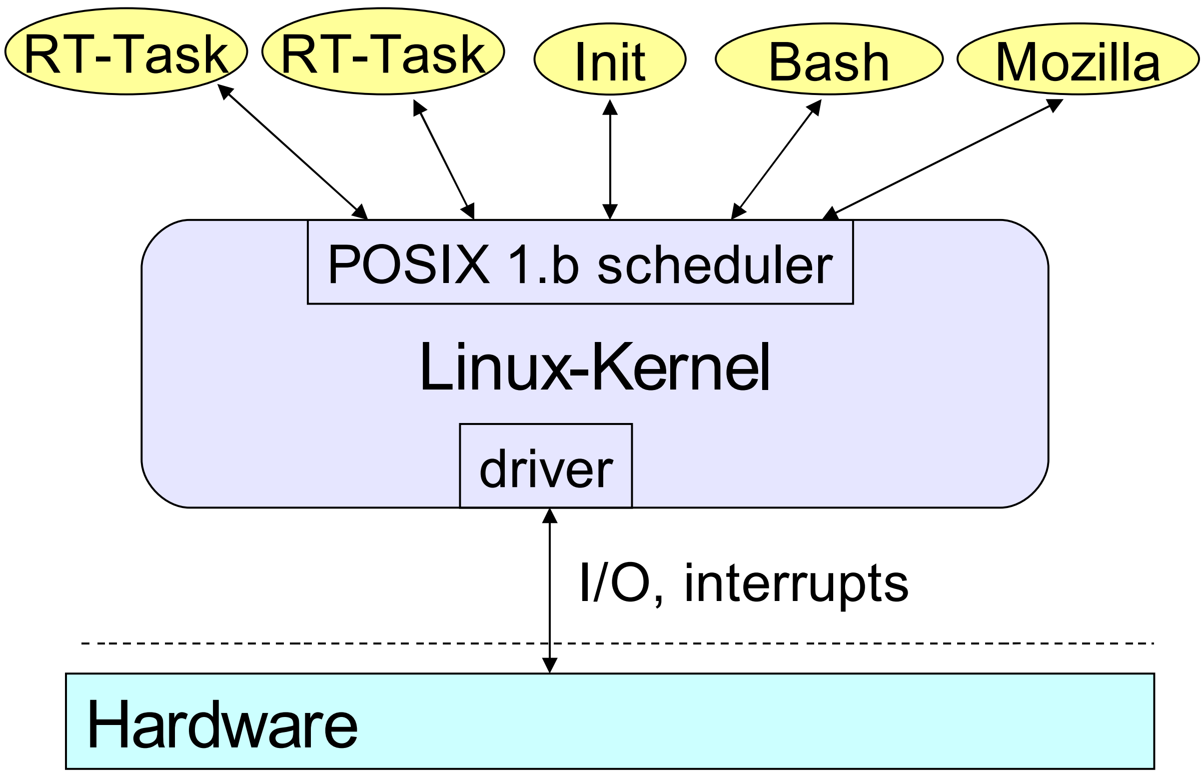
Example: RT-Linux



RT-tasks cannot use standard OS calls.
Commercially available from fsmlabs (www.fsmlabs.com)

Example: Posix 1.b RT-extensions to Linux

Standard scheduler can be replaced by POSIX scheduler implementing priorities for RT tasks



Special RT-calls and standard OS calls available.
Easy programming, no guarantee for meeting deadline

Summary

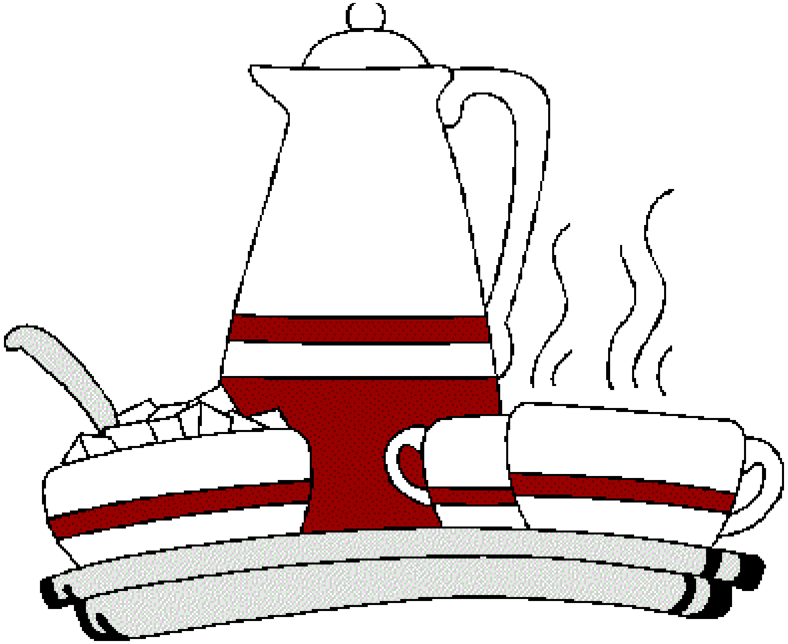
Embedded System Hardware

- Frequent use of “hardware in the loop”: Sensors → Discretization → Processing → D/A-conversion → actuators, using communication
- Importance of code size, energy & run-time efficiency
- Sampling theorem: reconstruction of original signals

Embedded Operating Systems

- Configurability!
- No standard peripherals ☞ I/O on higher layers
- No standard protection mechanism
- Real-time features ☞ may lead to redesign of OS

Coffee break (if on schedule)

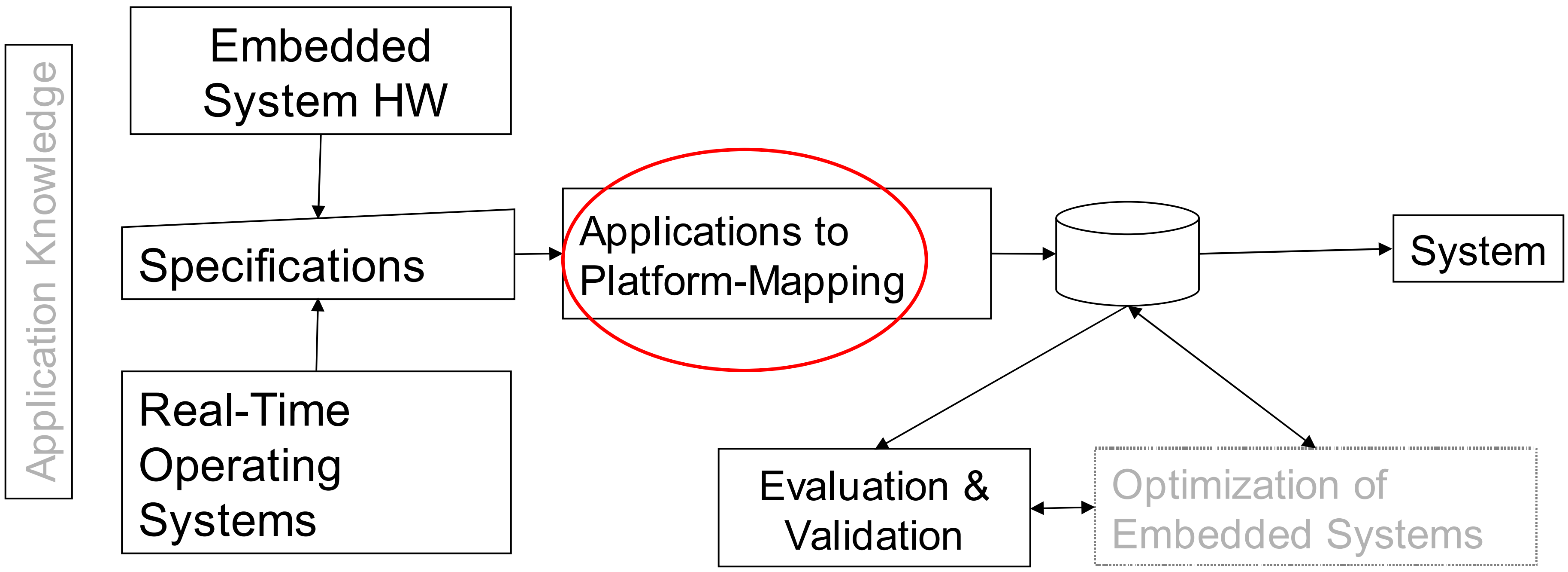


Technische Universität Dortmund

Mapping: Applications → Processors

Peter Marwedel
TU Dortmund,
Informatik 12
& ICD e.V.
Germany

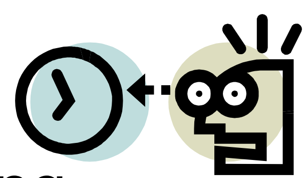
Structure of this tutorial

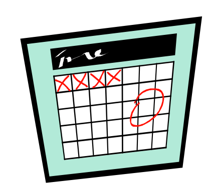


Scope of mapping algorithms

Useful terms from hardware synthesis:

- **Resource Allocation**
  Decision concerning type and number of available resources
- **Resource Assignment**
  Mapping: Task → (Hardware) Resource
- **xx to yy binding:**
  Describes a mapping from behavioral to structural domain, e.g. task to processor binding, variable to memory binding
- **Scheduling**
  Mapping: Tasks → Task start times
  Sometimes, resource assignment is considered being included in scheduling.





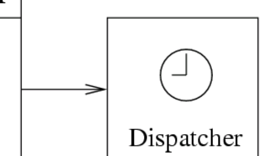
Dynamic/online vs. static/offline scheduling

- **Dynamic/online scheduling:**
  Processor allocation decisions (scheduling) at run-time; based on the information about the tasks arrived so far.

- **Static/offline scheduling:**
  Scheduling taking a priori knowledge about arrival times, execution times, and deadlines into account.

  Dispatcher allocates processor when interrupted by timer. Timer controlled by a table generated at design time.

Time	Action	WCET
10	start T1	12
17	send M5	
22	stop T1	
38	start T2	20
47	send M3	

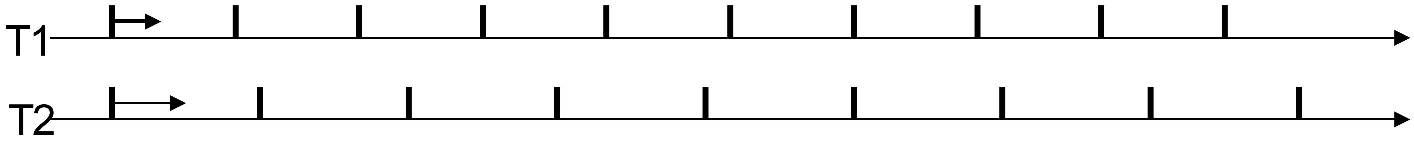


Time-triggered systems

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

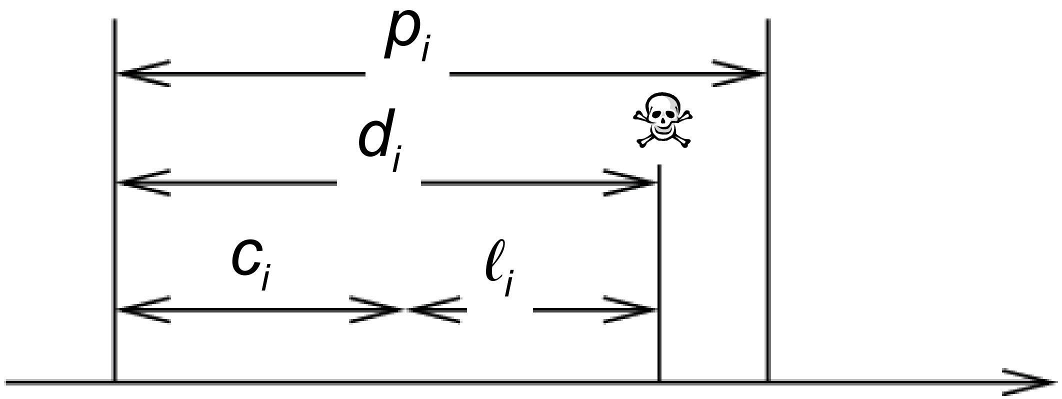
It can be easily checked if timing constraints are met.
The disadvantage is that the response to sporadic events may be poor.

Periodic scheduling



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***
- ℓ_i be the **laxity** or **slack**, defined as $\ell_i = d_i - c_i$



Average utilization

Average utilization:

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i}$$

Necessary condition for schedulability (with *m*=number of processors):

$$\mu \leq m$$

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 accumulated 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.

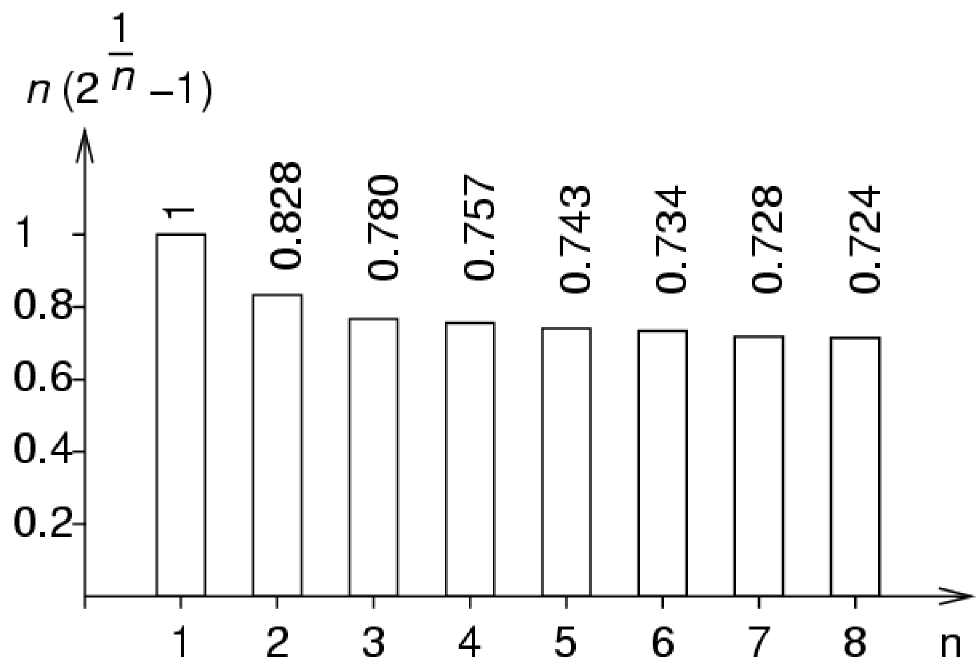
Theorem: If all RM assumptions are met, schedulability is guaranteed.

Maximum utilization for guaranteed schedulability

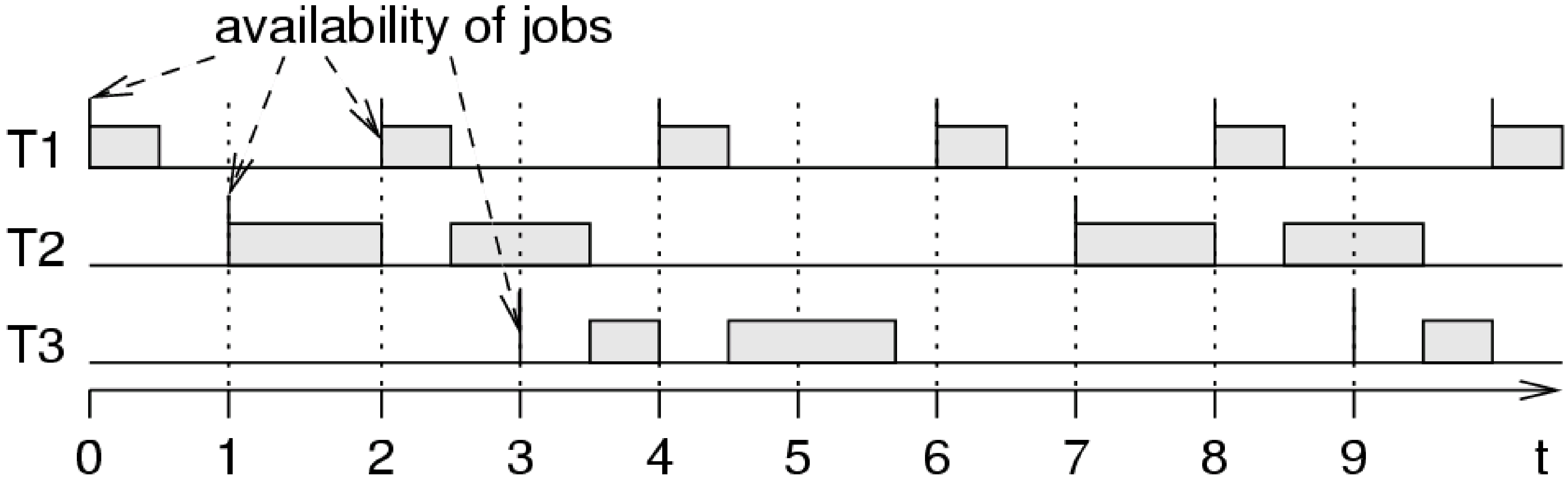
Maximum utilization as a function of the number of tasks:

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

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



Example of RM-generated schedule



T1 preempts T2 and T3.
T2 and T3 do not preempt each other.

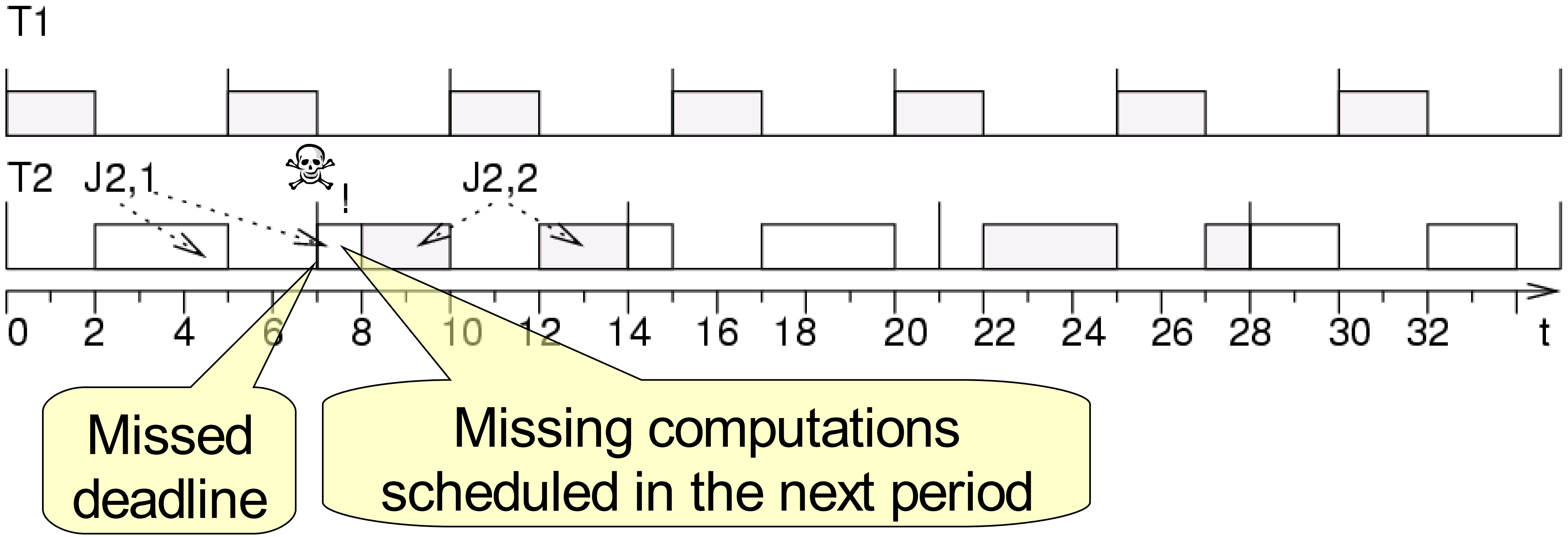
Case of failing RM scheduling

Task 1: period 5, execution time 2

Task 2: period 7, execution time 4

$\mu = 2/5 + 4/7 = 34/35 \approx 0.97$

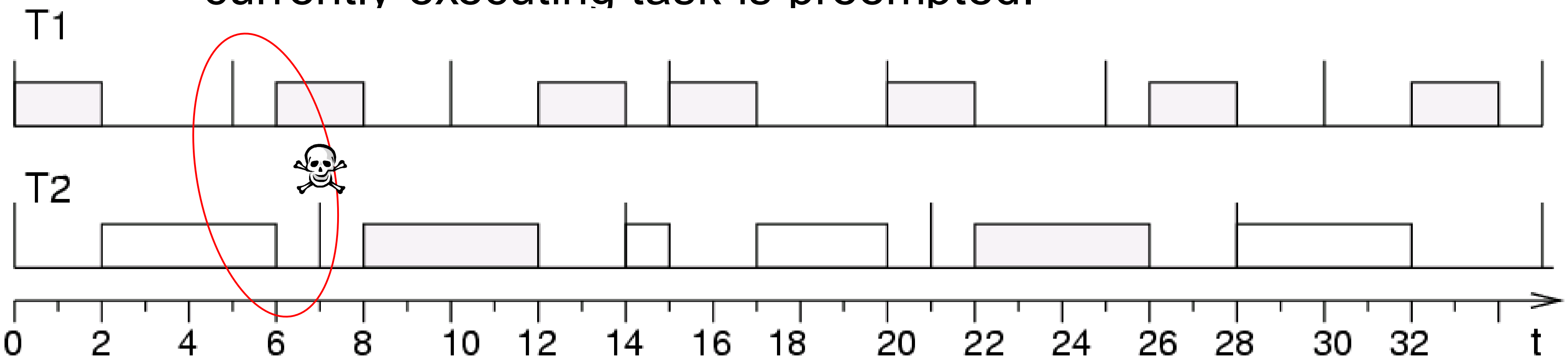
$2(2^{1/2}-1) \approx 0.828$



Earliest Deadline First (EDF) - Algorithm -

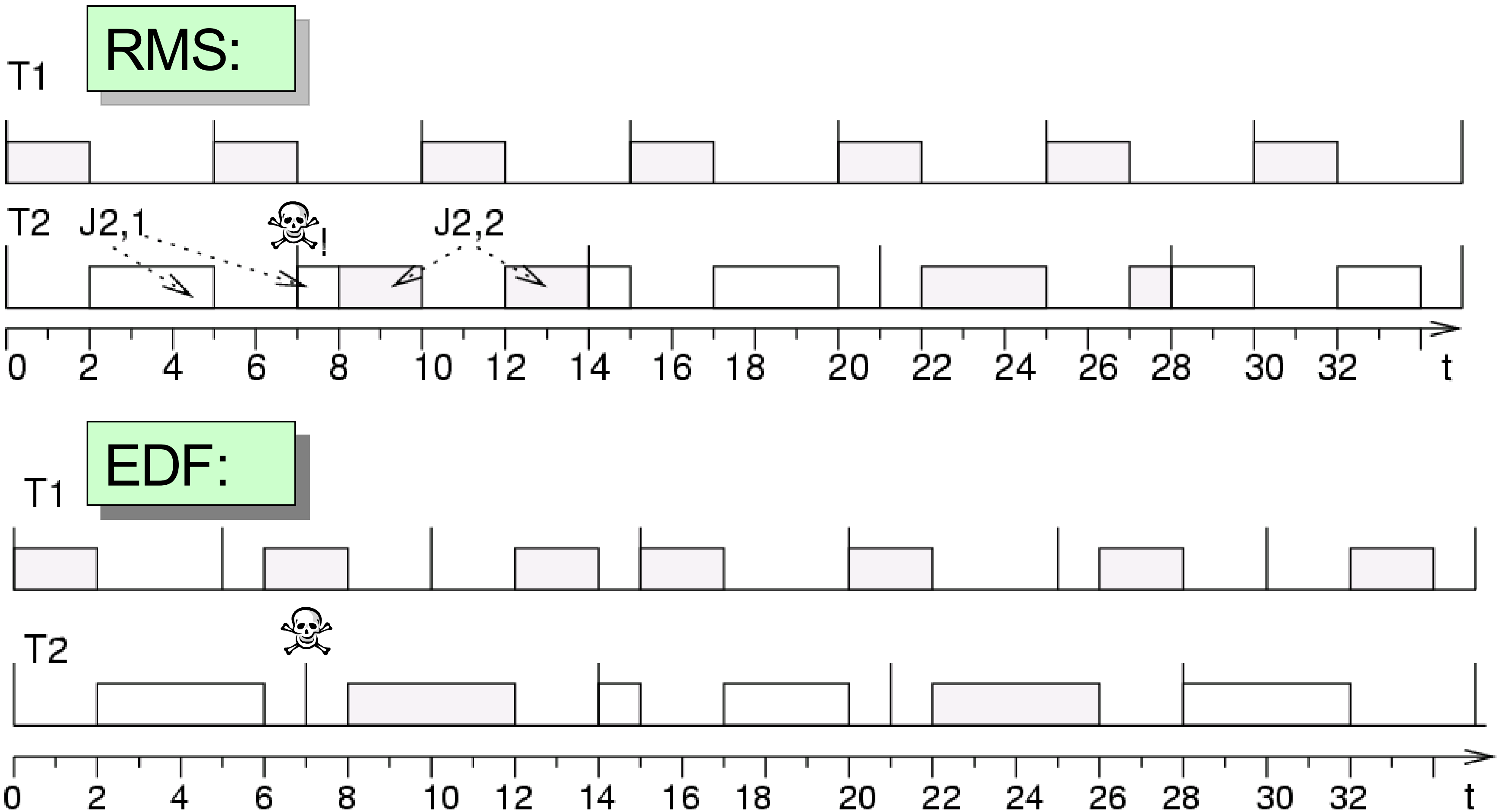
Earliest deadline first (EDF) algorithm:

- Highest priority (dynamic) to task with earliest deadline
- Each time a new ready task arrives:
  - It is inserted into a queue of ready tasks, sorted by their 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.



T2 not preempted, due to its earlier deadline.

Comparison EDF/RMS



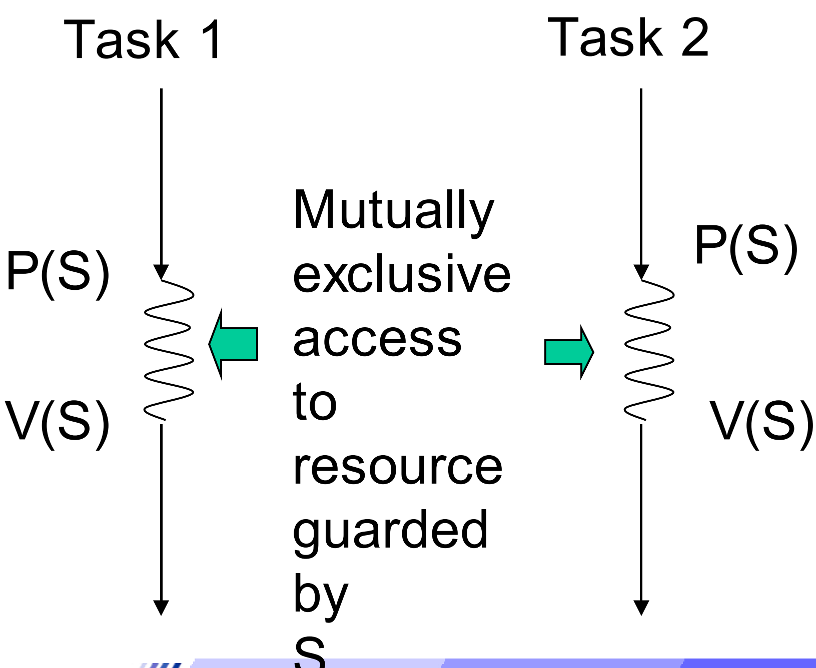
T2 not preempted, due to its earlier deadline.

Comparison RMS/EDF

	RMS	EDF
Priorities	Static	Dynamic
Works with std. OS with fixed priorities	**Yes**	**No**
Uses full computational power of processor	No, just up till $\mu=n(2^{1/n}-1)$	Yes
Possible to exploit full computational power of processor without provisioning for slack	**No**	**Yes**

Resource access protocols

Critical sections: sections of code at which exclusive access to some resource must be guaranteed. Can be guaranteed with semaphores S or “mutexes”.

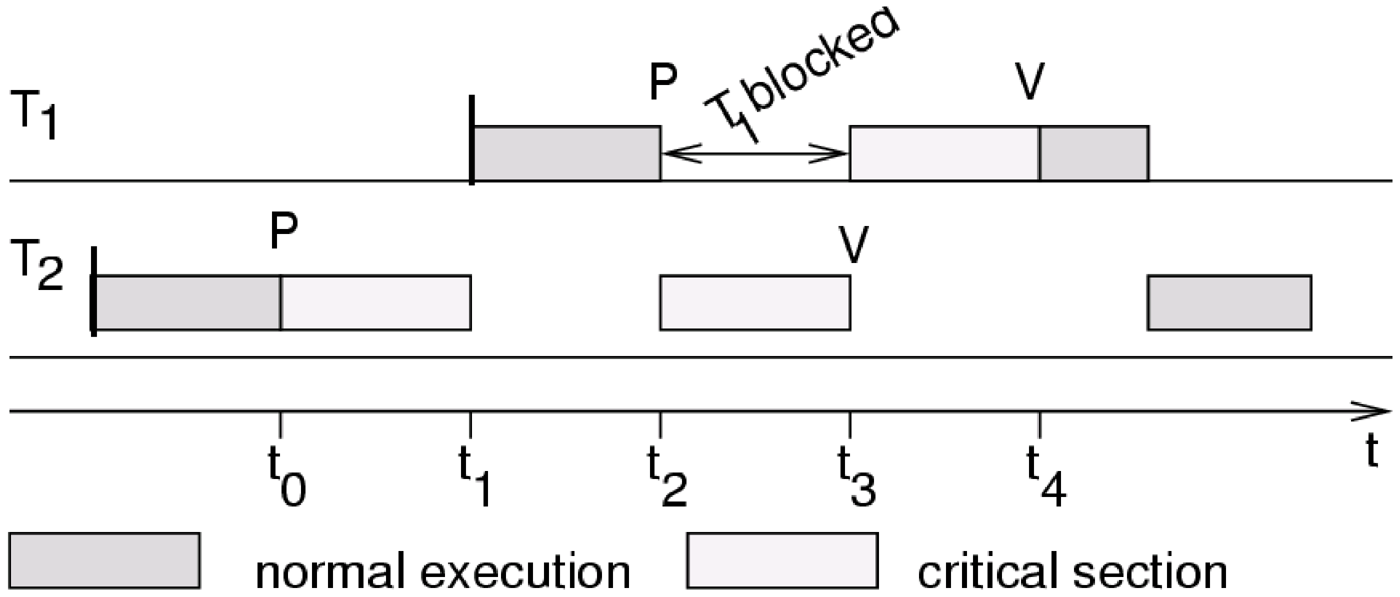


P(S) checks semaphore to see if resource is available and if yes, sets S to „used“. Uninterruptible operations! If no, calling task has to wait.

V(S): sets S to „unused“ and starts sleeping task (if any).

Priority inversion

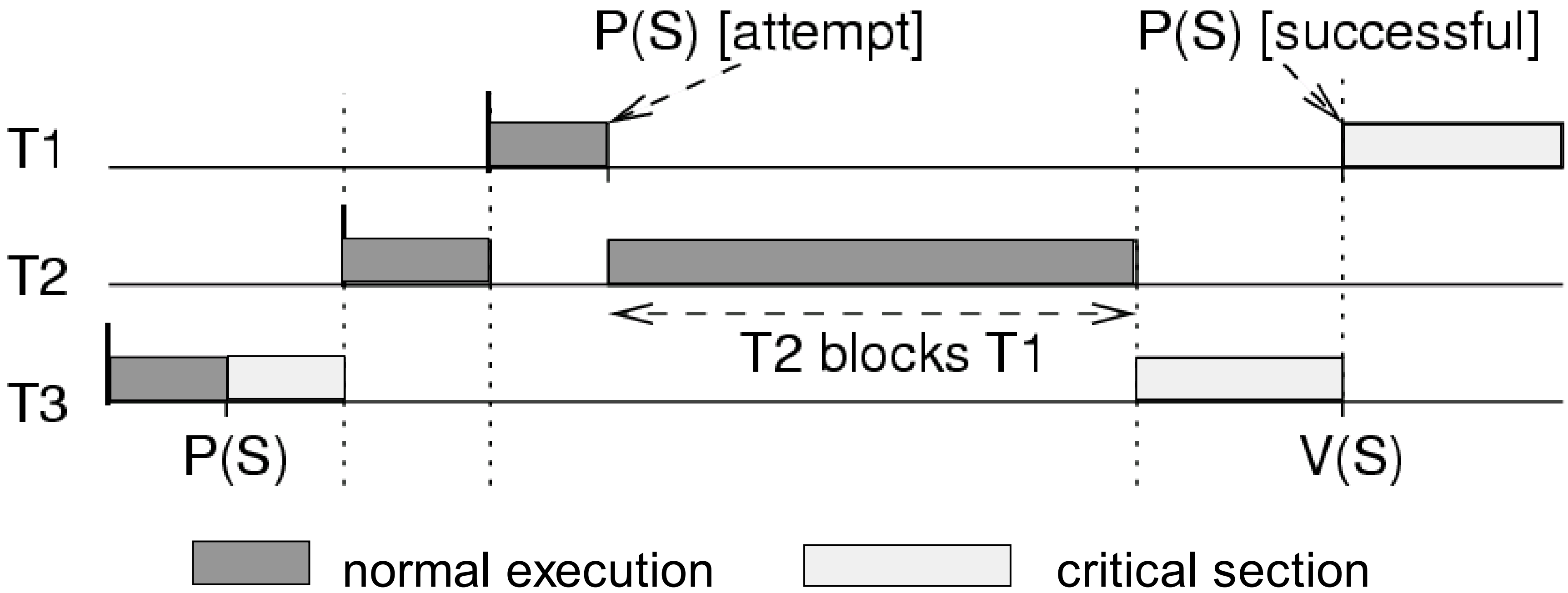
Priority T_1 assumed to be > than priority of T_2.
If T_2 requests exclusive access first (at t_0), T_1 has to wait until T_2 releases the resource (time t_3), thus inverting the priority:



In this example:
duration of inversion bounded by length of critical section of T_2.

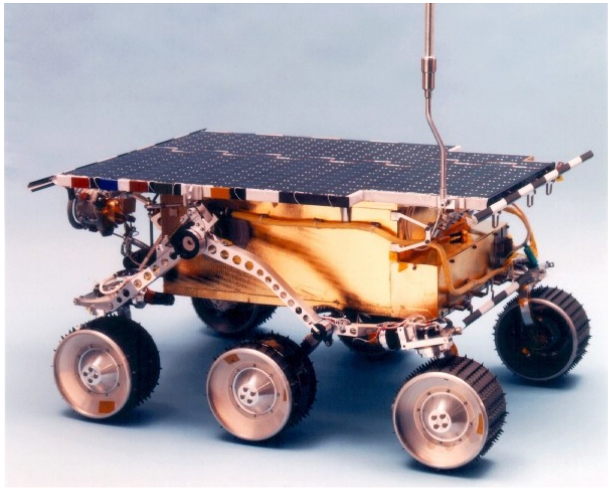
Duration of priority inversion with >2 tasks can exceed the length of any critical section

Priority of T1 > priority of T2 > priority of T3.
T2 preempts T3:
T2 can prevent T3 from releasing the resource.



The MARS Pathfinder problem (1)

- A bus management task ran frequently with high priority, locking the shared memory area.
- A task collecting meteorological data ran as a low priority thread, also locking the shared memory area.
- The spacecraft also contained a communications task that ran with medium priority.

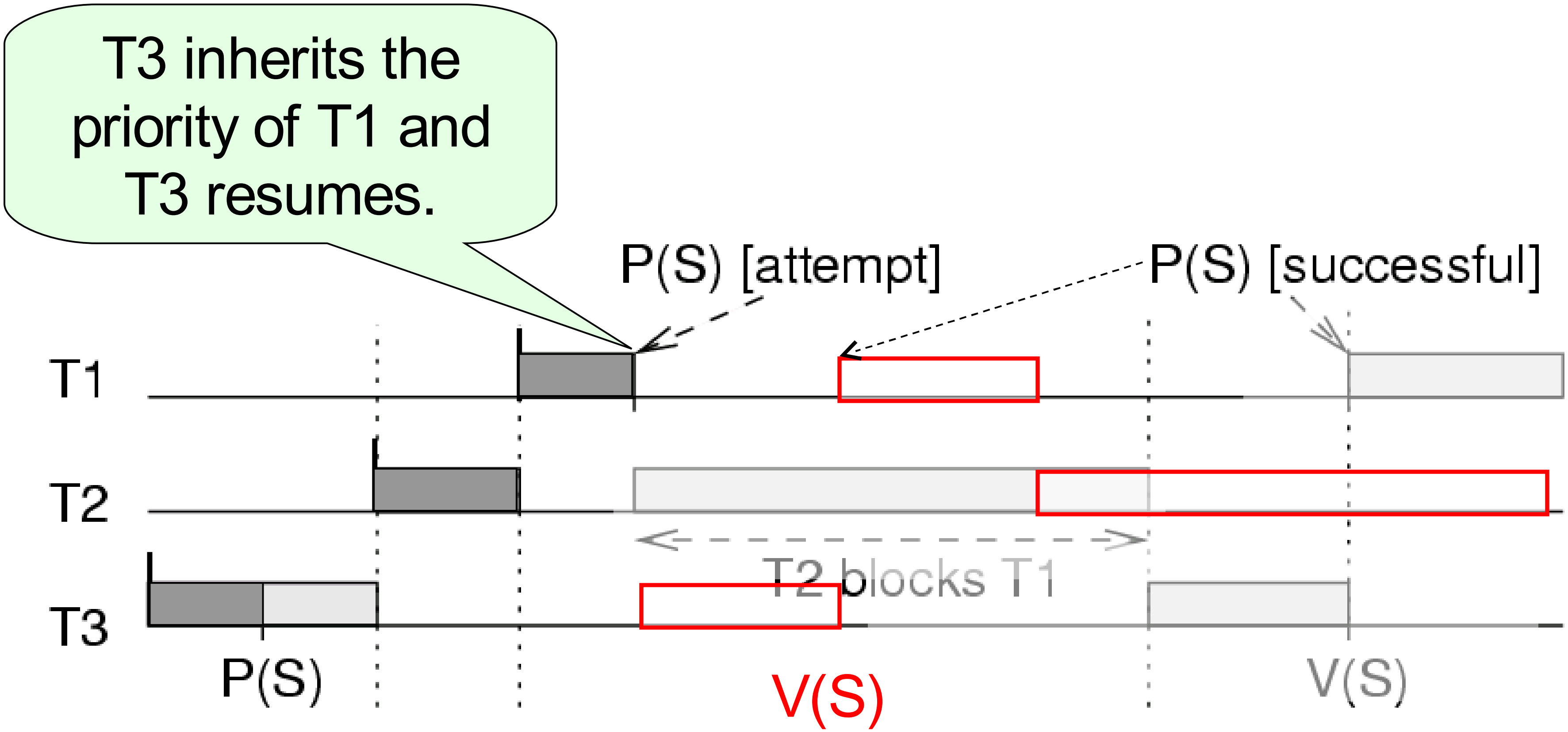


Coping with priority inversion: the priority inheritance protocol

- Tasks are scheduled according to their active priorities. Tasks with the same priorities are scheduled FCFS.
- If task T1 executes P(S) & exclusive access granted to T2: T1 will become blocked.
  If priority(T2) < priority(T1): T2 inherits the priority of T1.
  ☞ T2 resumes.
  Rule: tasks inherit the highest priority of tasks blocked by it.
- When T2 executes V(S), its priority is decreased to the highest priority of the tasks blocked by it.
  If no other task blocked by T2: priority(T2):= original value.
  Highest priority task so far blocked on S is resumed.
- Transitive: if T2 blocks T1 and T1 blocks T0, then T2 inherits the priority of T0.

Example

How would priority inheritance affect our example with 3 tasks?



Mapping Scenario: Overview

Given

1. specification of the task structure (**task model**) = for each flow the corresponding tasks to be executed
2. different usage scenarios (**flow model**)

Sought

processor implementation (**resource model**) =
architecture* + task mapping + scheduling

Objectives:

1. maximize performance
2. minimize cost

Subject to:

1. memory constraints
2. delay constraints

} (**performance model**)

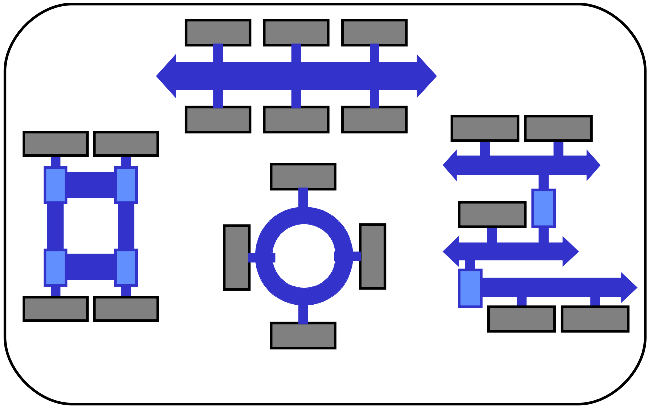
***: 2 cases:**

1. fixed architecture
2. architecture to be designed

based on Thiele's slides

Design Space

Communication Templates



Computation Templates

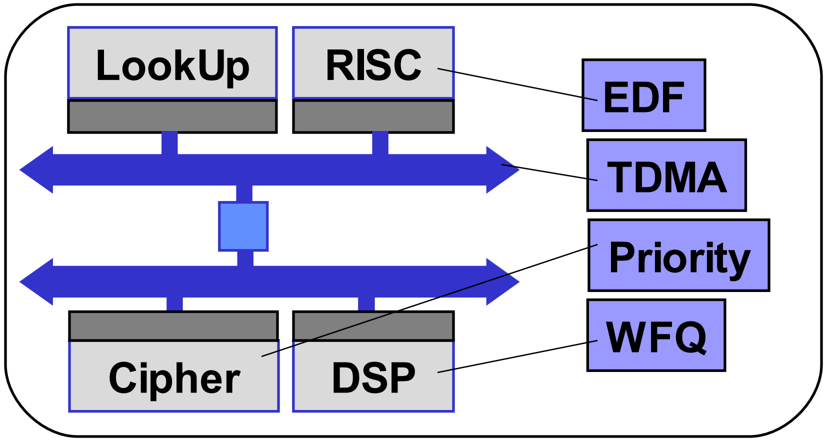
- Cipher
- FPGA
- DSP
- RISC
- SDRAM
- µE
- LookUp

Scheduling/Arbitration

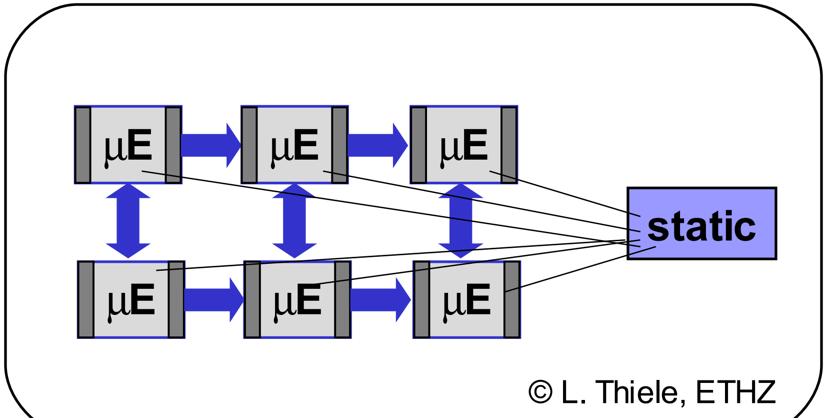
- EDF
- TDMA
- proportional share
- WFQ
- FCFS
- dynamic fixed priority
- static

Which architecture is better suited for our application?

Architecture # 1

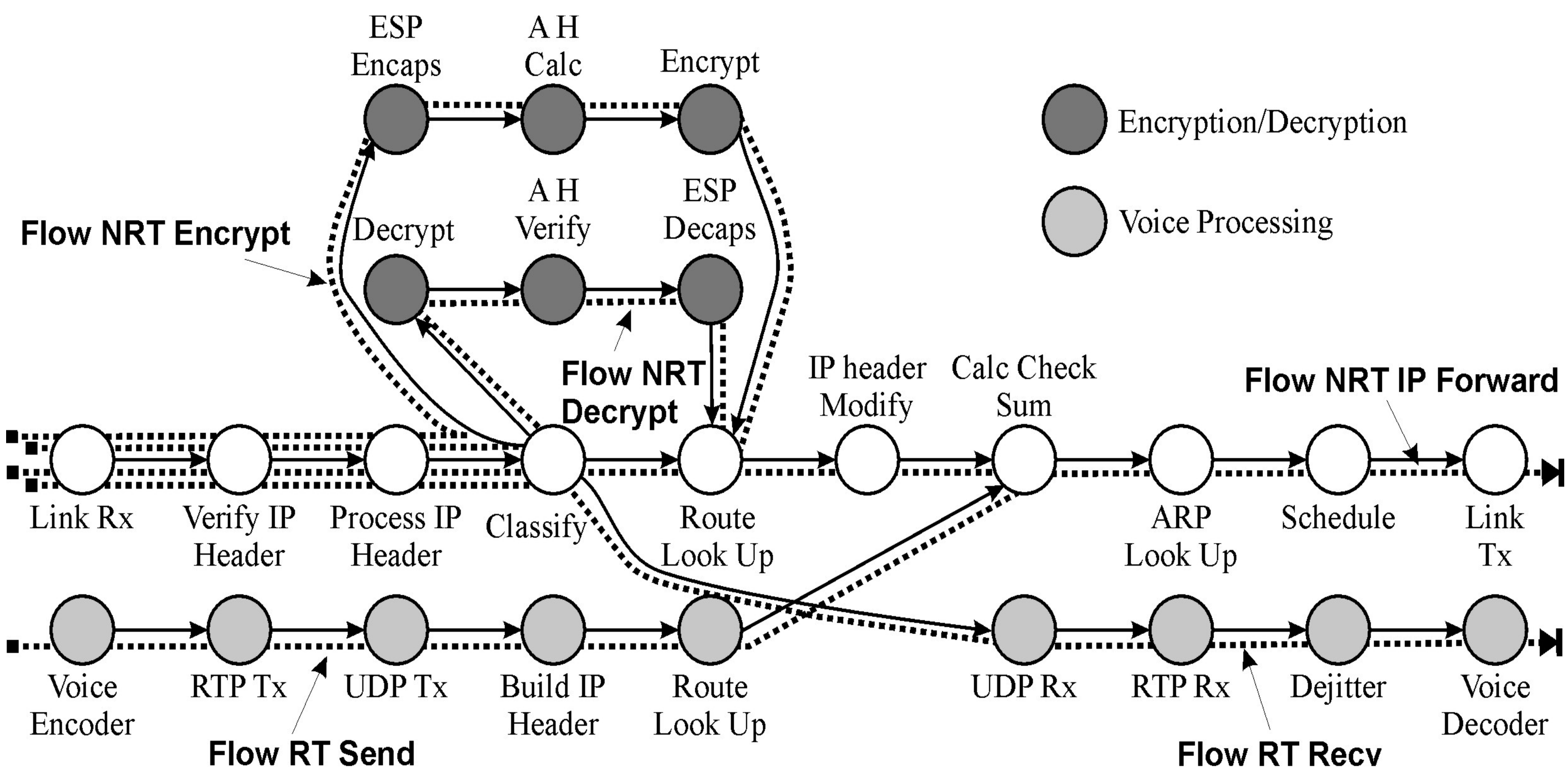


Architecture # 2



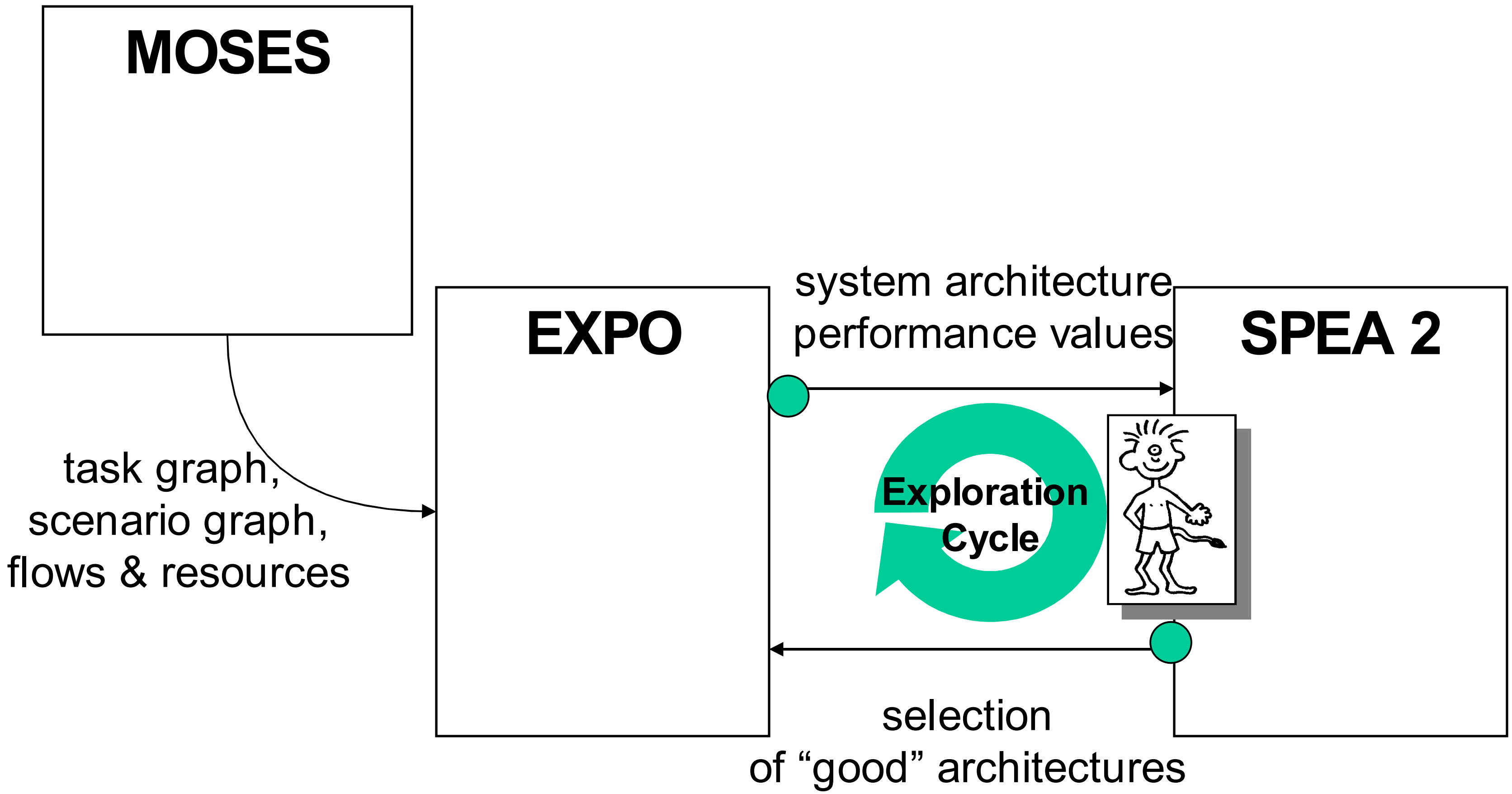
Application Model

Example of a simple stream processing task structure:



© L. Thiele, ETHZ

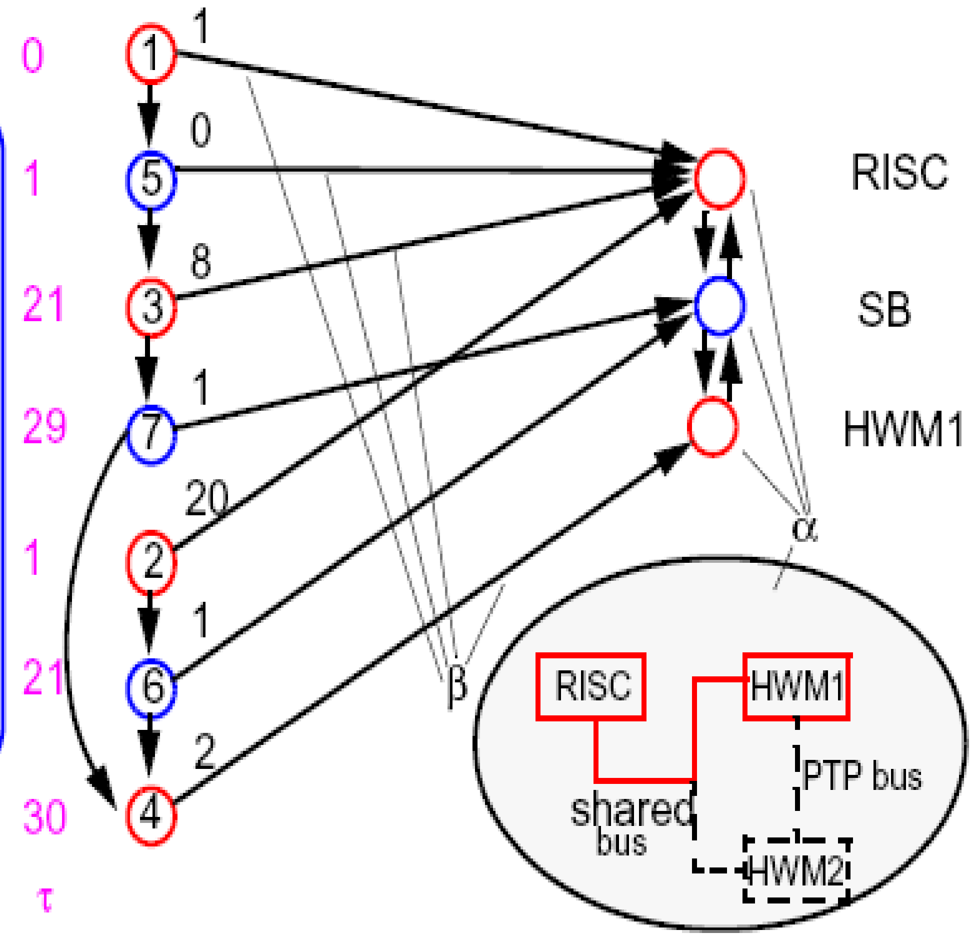
EXPO – Tool architecture (1)



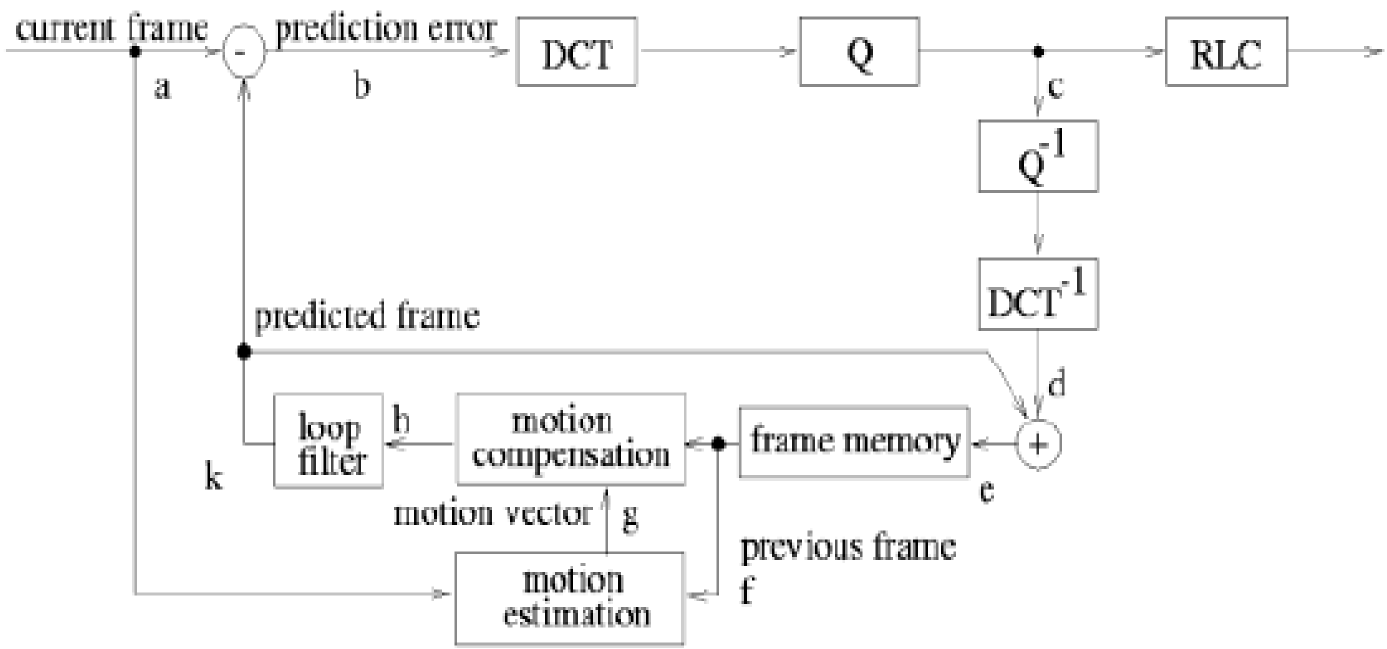
© L. Thiele, ETHZ

Basic model - implementation

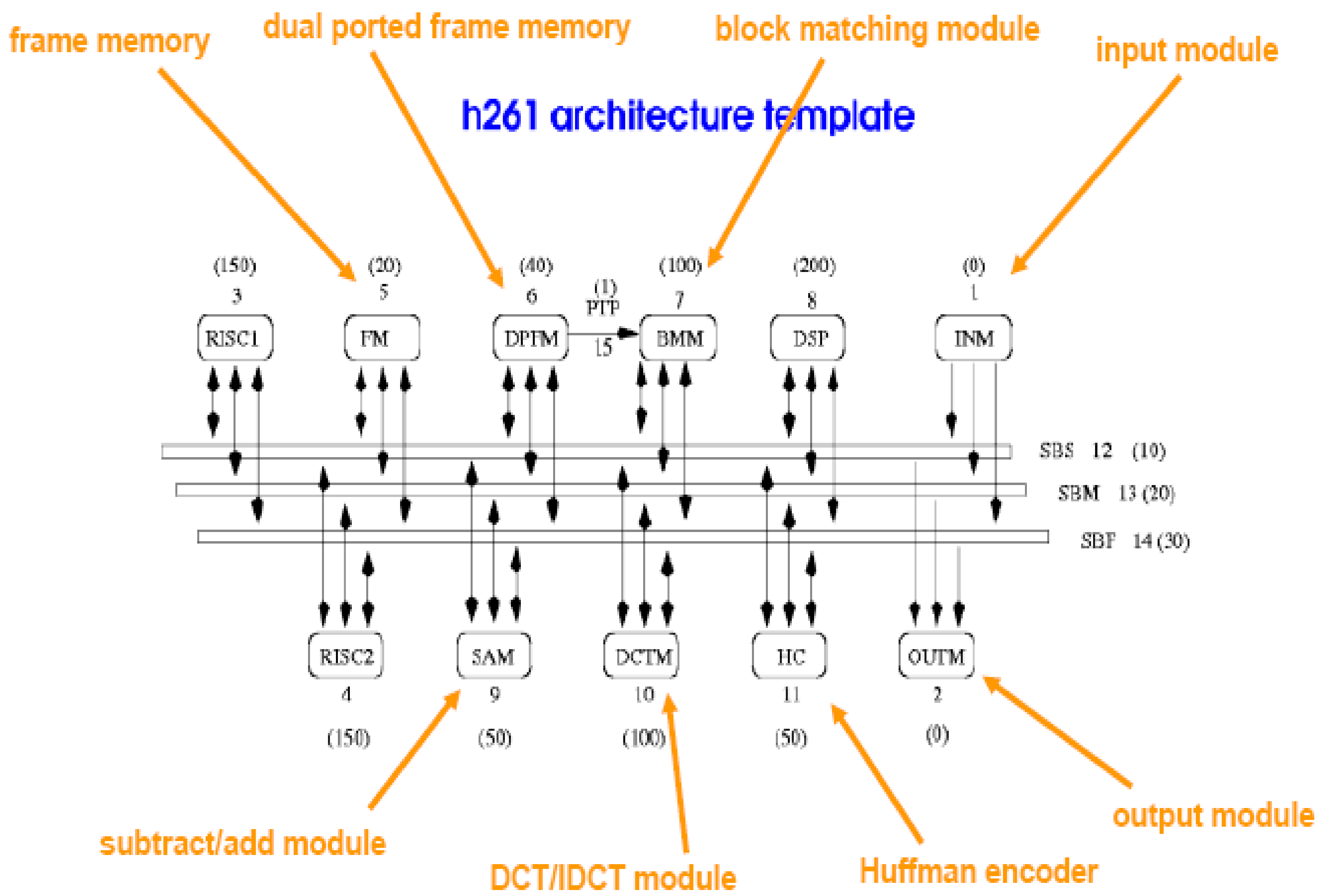
Definition: Given a specification graph G_S an implementation is a triple (α,β,τ), where α is a feasible allocation, β is a feasible binding, and τ is a schedule.



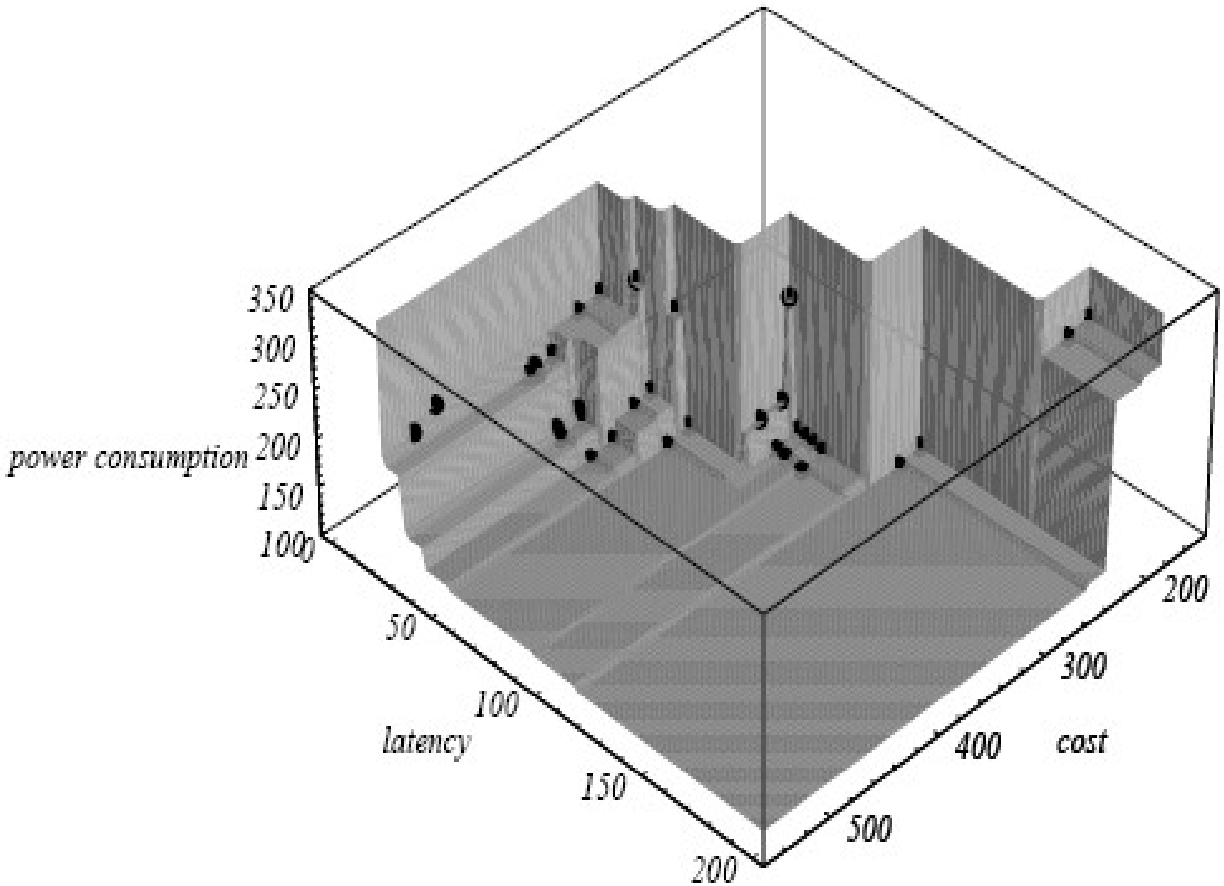
behavioral specification of a video codec for video compression



Exploration - Case Study (3)



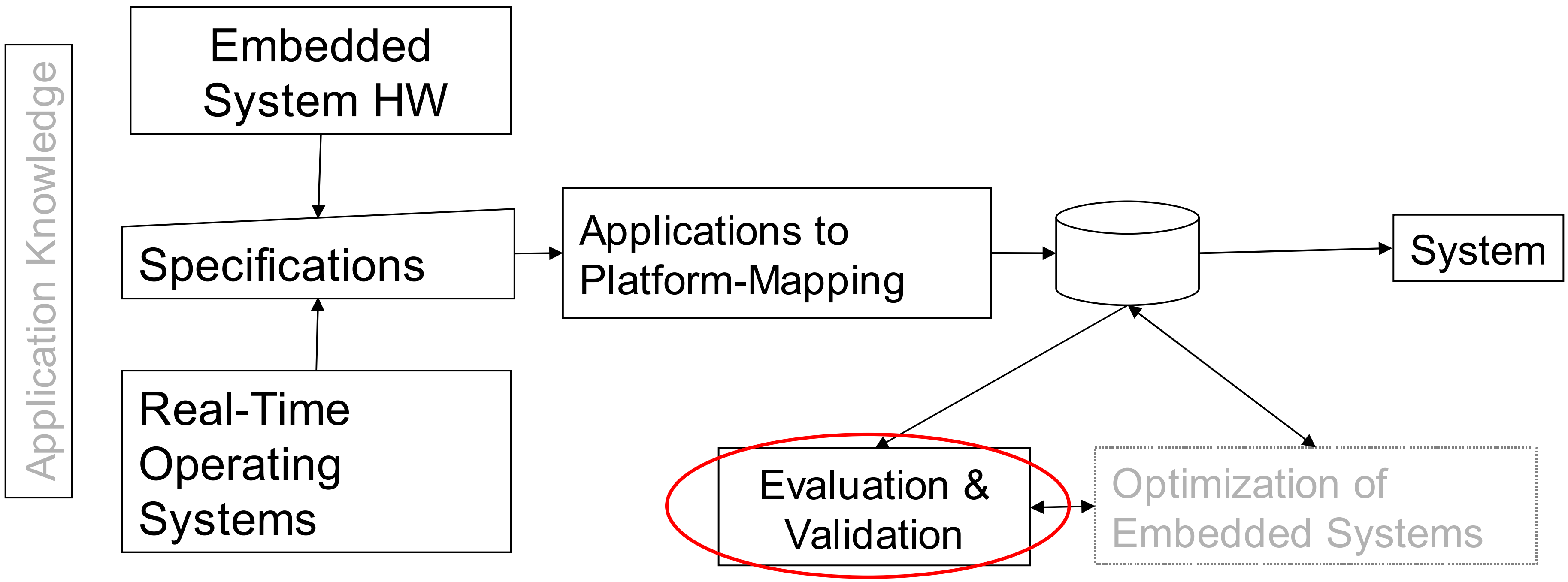
EA Case Study - Design Space



Evaluation and Validation

Peter Marwedel
TU Dortmund, Informatik 12
& ICD e.V.
Germany

Structure of this tutorial



Validating functional behavior by simulation

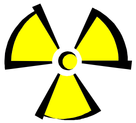
Various levels of abstractions used for simulations:

- High-level of abstraction: fast, but sometimes not accurate
- Lower level of abstraction: slow and typically accurate
- Choosing a level is always a compromise

Simulations Limitations

- Typically slower than the actual design.
  ☞ **Violations of timing constraints** likely if simulator is connected to the actual environment

- Simulations in the real environment may be **dangerous**



- There may be huge amounts of data and it may be impossible to simulate enough data in the available time.

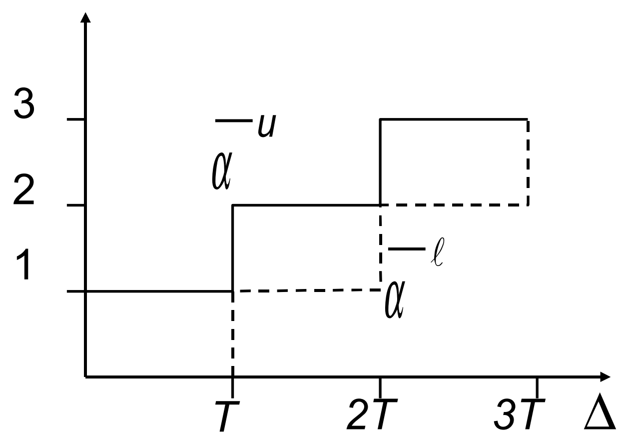
- Most actual systems are too complex to allow simulating all possible cases (inputs).
  Simulations can help finding errors in designs, but they cannot guarantee the absence of errors.

Thiele's real-time calculus - Arrival curves -

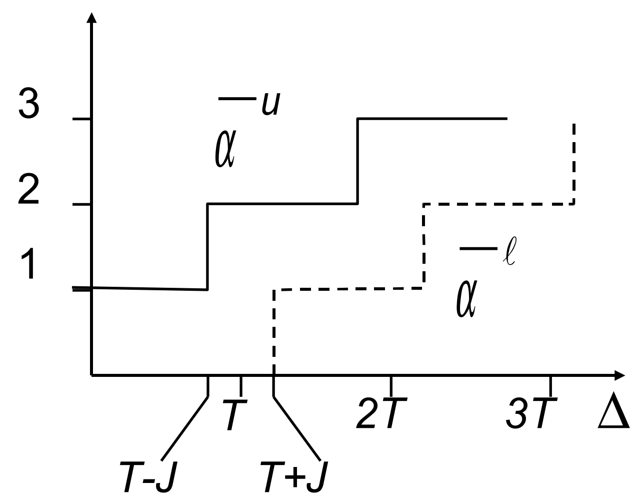
Arrival curves describe the maximum and minimum number of events arriving in some time interval Δ

Examples

periodic event stream



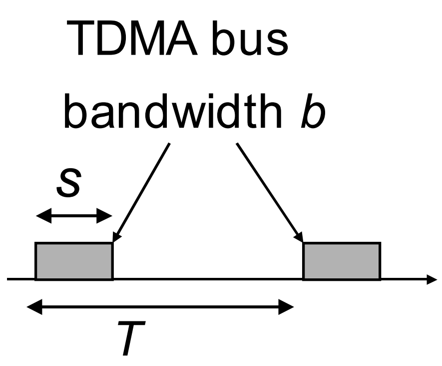
periodic event stream with jitter

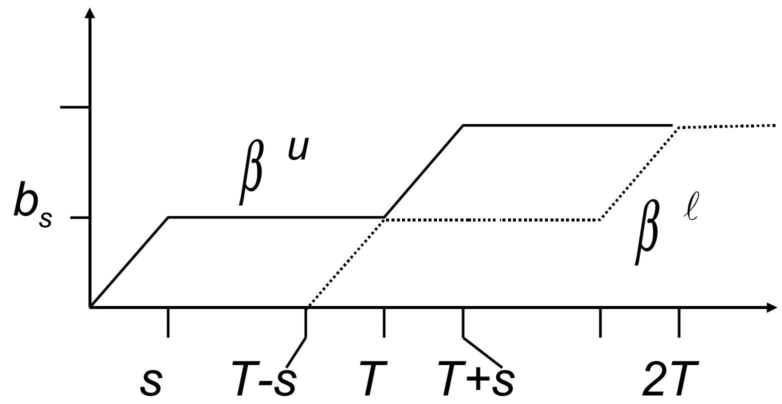


Thiele's real-time calculus - Service curves -

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

Example:

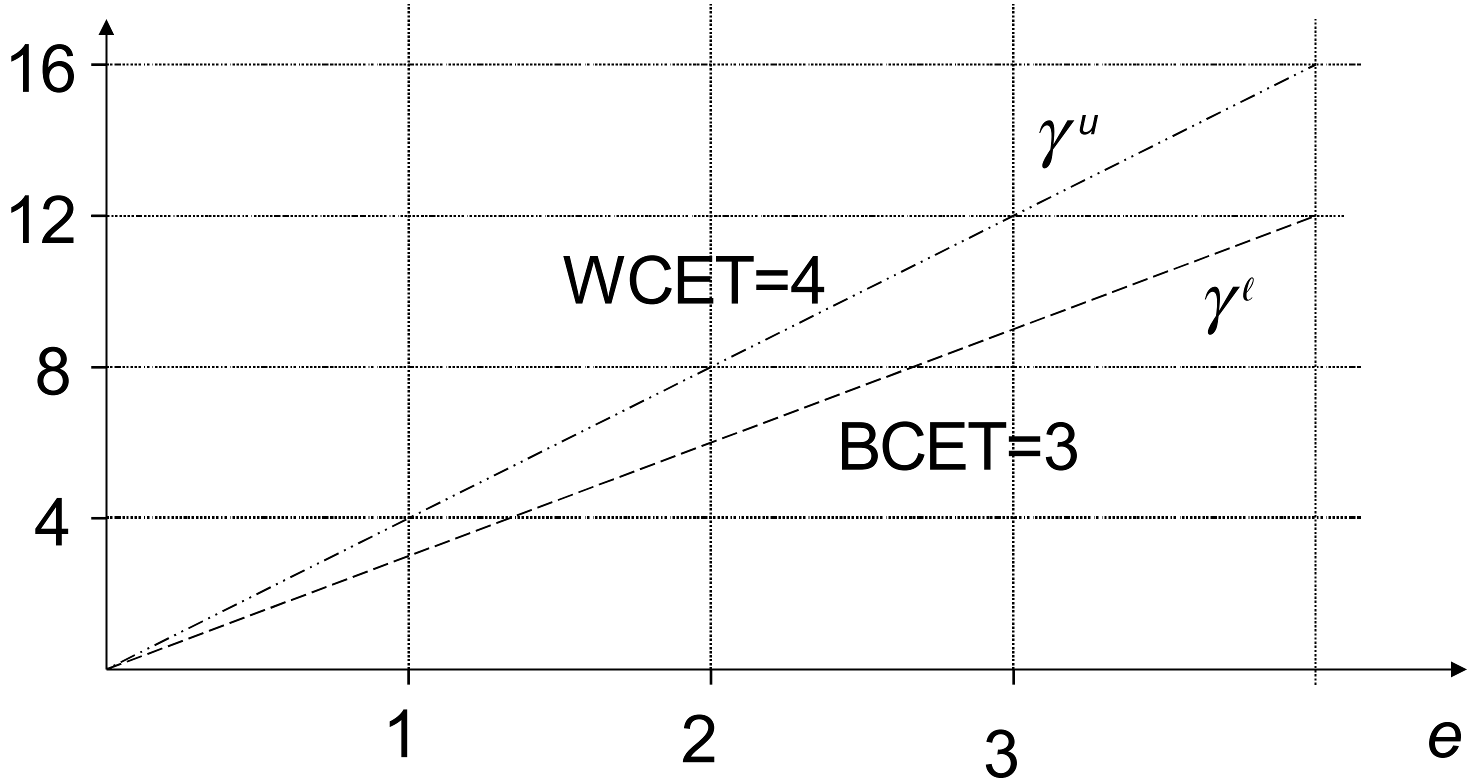




Thiele's real-time calculus - Workload characterization -

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

Example



Workload required for incoming stream

Incoming workload

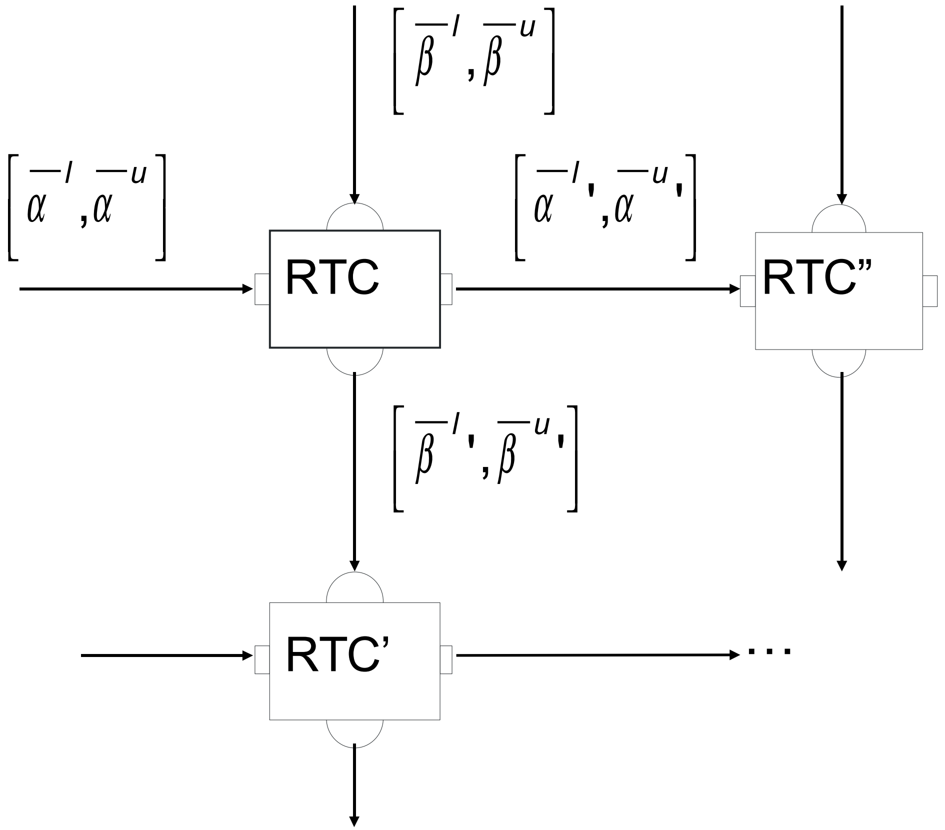
$$\alpha^{u}(\Delta) = \gamma^{u}\left(\overline{\alpha^{u}}(\Delta)\right) \qquad \alpha^{\ell}(\Delta) = \gamma^{\ell}\left(\overline{\alpha^{\ell}}(\Delta)\right)$$

Upper and lower bounds on the number of events

$$\overline{\beta}^{u}(\Delta) = \gamma^{-1}\left(\beta^{u}(\Delta)\right) \qquad \overline{\beta}^{\ell}(\Delta) = \gamma^{-1}\left(\beta^{\ell}(\Delta)\right)$$

Thiele's real-time calculus - 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 ☞

Thiele's real-time calculus
- Transformation of arrival and service curves -

Resulting arrival curves:

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

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

Remaining service curves:

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

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

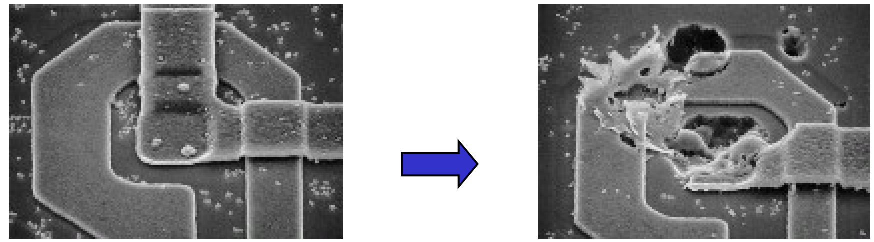
Where:

$$\left(f \underline{\otimes} g\right)(t) = \inf_{0 \le u \le t}\left\{f(t-u) + g(u)\right\} \qquad \left(f \overline{\otimes} g\right)(t) = \sup_{0 \le u \le t}\left\{f(t-u) + g(u)\right\}$$

$$\left(f \underline{\oplus} g\right)(t) = \inf_{u \ge 0}\left\{f(t+u) - g(u)\right\} \qquad \left(f \overline{\oplus} g\right)(t) = \sup_{u \ge 0}\left\{f(t+u) - g(u)\right\}$$

Risk- and dependability analysis

Example : metal migration @ Pentium 4



www.jrwhipple.com/computer_hangs.html

„10^{-9}": For many systems, probability of a catastrophe has to be less than 10^{-9} per hour ≡ one case per 100,000 systems for 10,000 hours.

FIT: failure-in-time unit for failure rate (=1/MTTF≈1/MTBF);

1 FIT: rate of 10^{-9} failures per hour

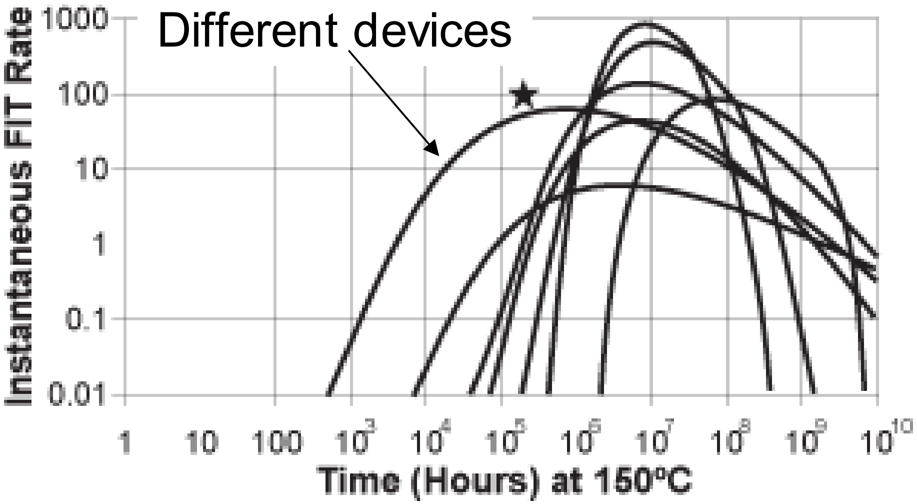
Damages are resulting from hazards.

For every damage there is a severity and a probability.

Several techniques for analyzing risks.

Actual failure rates

Example: failure rates less than 100 FIT for the first 20 years of life at 150°C @ TriQuint (GaAs)
[www.triquint.com/company/quality/faqs/faq_11.cfm]



Target: Failures rates of systems ≤ 1FIT

Reality: Failures rates of circuits ≤ 100 FIT

☞redundancy is required to make a system more reliable than its components

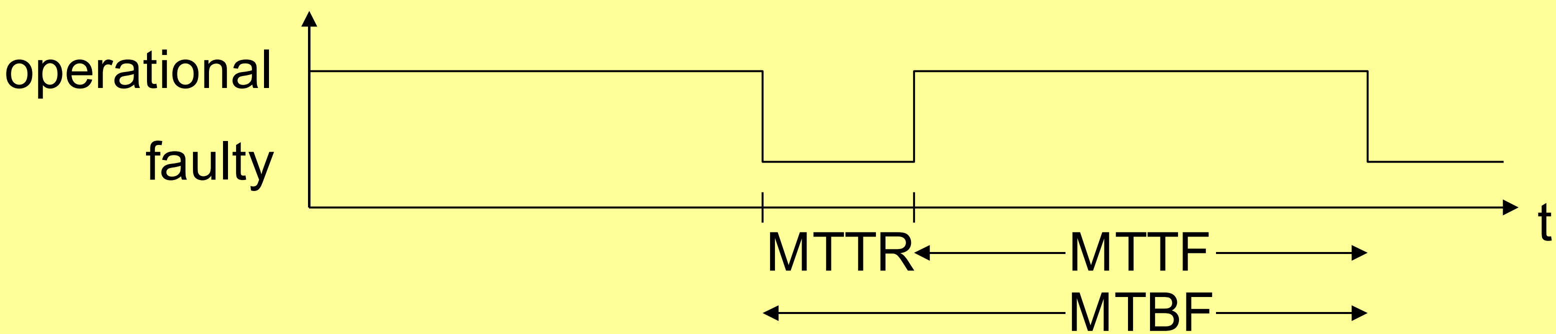
Analysis frequently works with simplified models ☞

MTTF, MTTR and MTBF

MTTR = mean time to repair
(average over repair times using distribution *M*(*d*))
MTBF* = mean time between failures = MTTF + MTTR

$$\text{Availability } A = \lim_{t \to \infty} A(t) = \frac{\text{MTTF}}{\text{MTBF}}$$

Ignoring the statistical nature of faults …

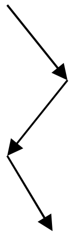


---

* Mixed up with MTTF, if starting in operational state is implicitly assumed

Fault tree Analysis (FTA)

- FTA is a top-down method of analyzing risks. Analysis starts with possible damage, tries to come up with possible scenarios that lead to that damage.
- FTA typically uses a graphical representation of possible damages, including symbols for AND- and OR-gates.
- OR-gates are used if a single event could result in a hazard.
- AND-gates are used when several events or conditions are required for that hazard to exist.

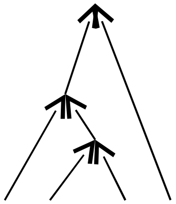


Failure mode and effect analysis (FMEA)

- FMEA starts at the components and tries to estimate their reliability. The first step is to create a table containing components, possible faults, probability of faults and consequences on the system behavior.

Component	*Failure*	*Consequences*	*Probability*	*Critical?*
Processor	metal migration	no service	10^{-6} /h	yes
...	...	...	...	...

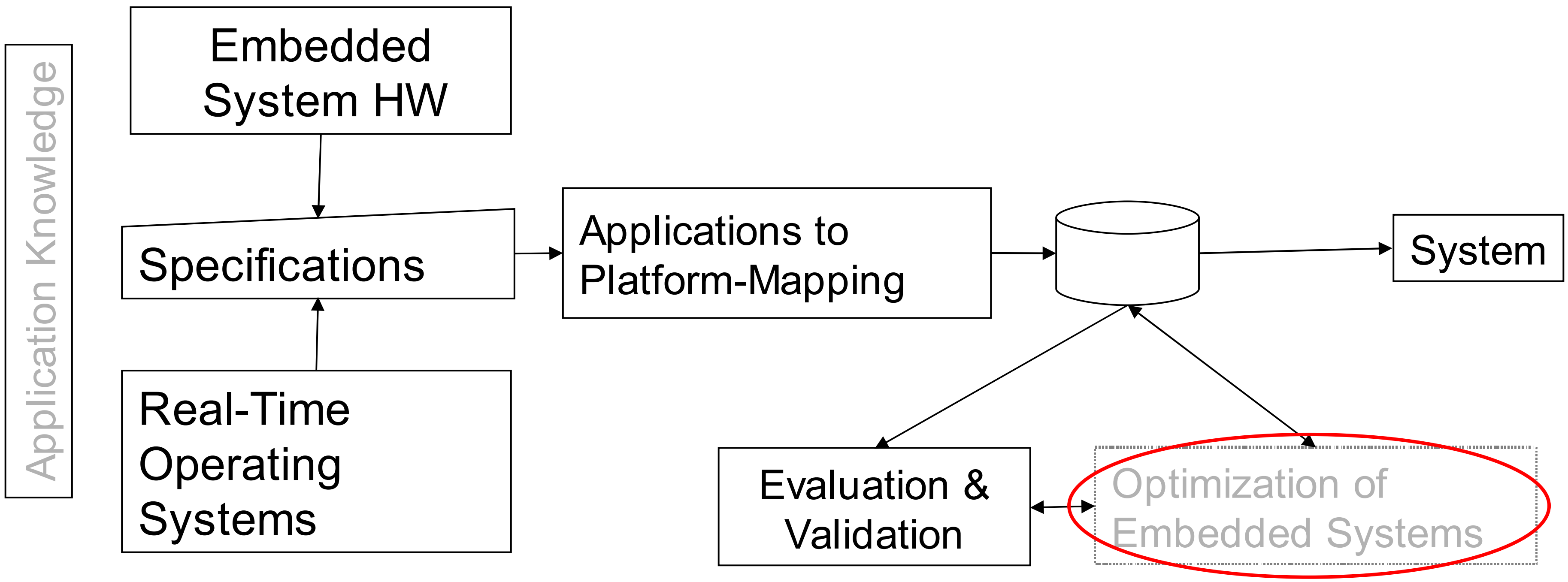
- Using this information, the reliability of the system is computed from the reliability of its parts (corresponding to a bottom-up analysis).



Optimization

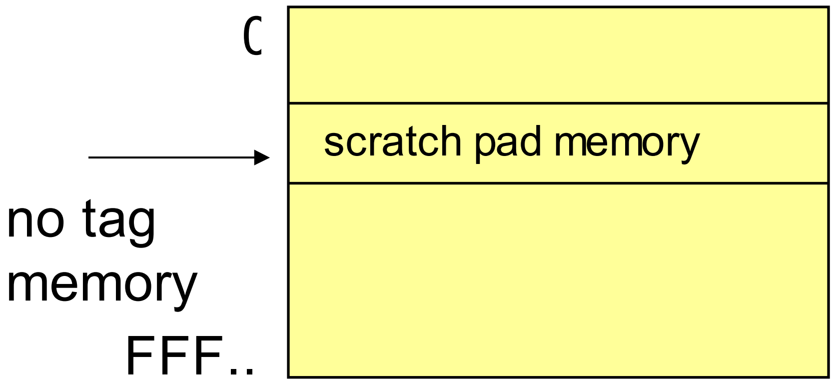
Peter Marwedel
TU Dortmund, Informatik 12
& ICD e.V.
Germany

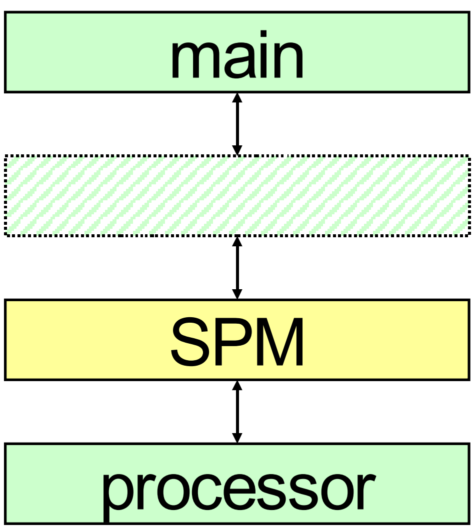
Structure of this tutorial



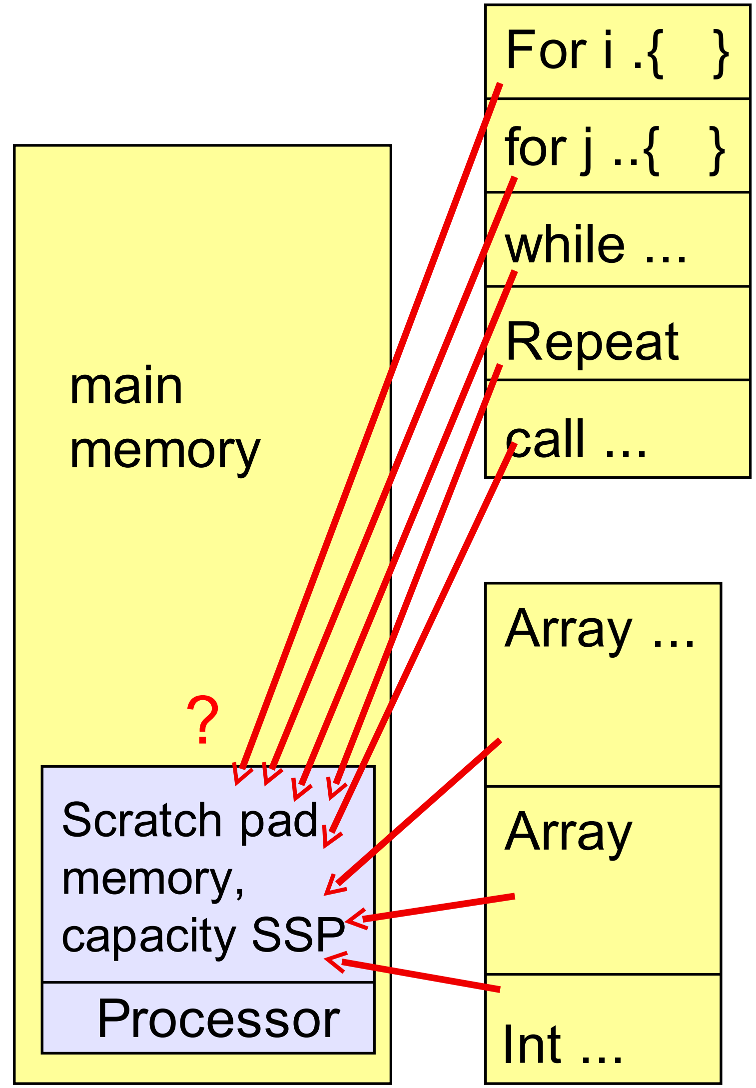
(just a very brief look at our work)

Optimization of hierarchical memories using scratch pad memories (SPM)



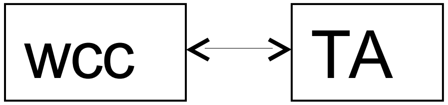


fast,
energy efficient
& timing
predictable



Optimization of WCET
- Integration of compilers and timing analysis -

Reconciliation of compilers and timing analysis



Opportunities:

- Not just post-compilation analysis of WCET
- Precise WCET information for run-time optimizations
- Pass additional information (flow facts) to timing analysis
- Aggressive optimizations for code on WCET path
- Respecting WCET constraints during compilation
- Reduction of jitter in multimedia applications
- Avoids rerunning compilers again and again with different options to meet real-time constraint.

Exploitation of mobile platforms

Communication between mobile platforms using public communication techniques

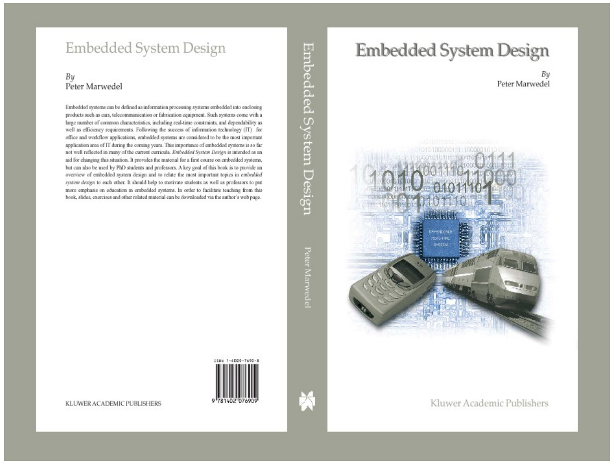
- ☞ messaging
- ☞ information sharing

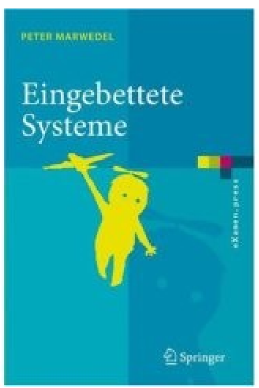
without using any commercial access provider

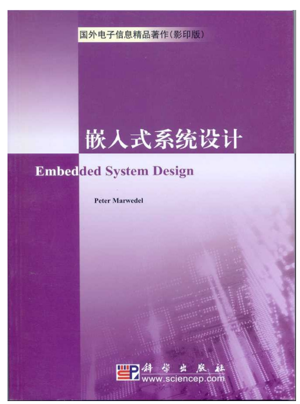
Also active in project on health care and forest mitigation.

Considering resource management.

Textbook(s)







Several Editions:

- Original hardcover version, Kluwer, 2003, >100 $/€
- Reprint, lighter cover borders, same price/content; Corrections available on web site
- 2nd edition, soft cover, with corrections, Springer, end of Dec.2005/Jan.2006, 37-39€
- German edition, March 2007, 29 €
- Chinese edition, April 2007, only preface in Chinese, not for sale outside China
- Russian edition (to be publ.d)

Web site: //ls12-www.cs.tu-dortmund.de/~marwedel/es-book

Summary

Mapping Applications → Processors

- Standard scheduling theory for real-time scheduling
  - Rate monotonic vs. earliest deadline first
  - Nasty priority inversion ☞ protocols like priority inheritance
- Mapping for complex multi-processors systems
  - allocation (if hardware is not fixed)
  - binding of tasks to resources, scheduling
  - multi-objective optimization problems ☞ Evolutionary algorithms

Evaluation

- of performance, energy consumption, reliability, …
- real-time calculus as an example

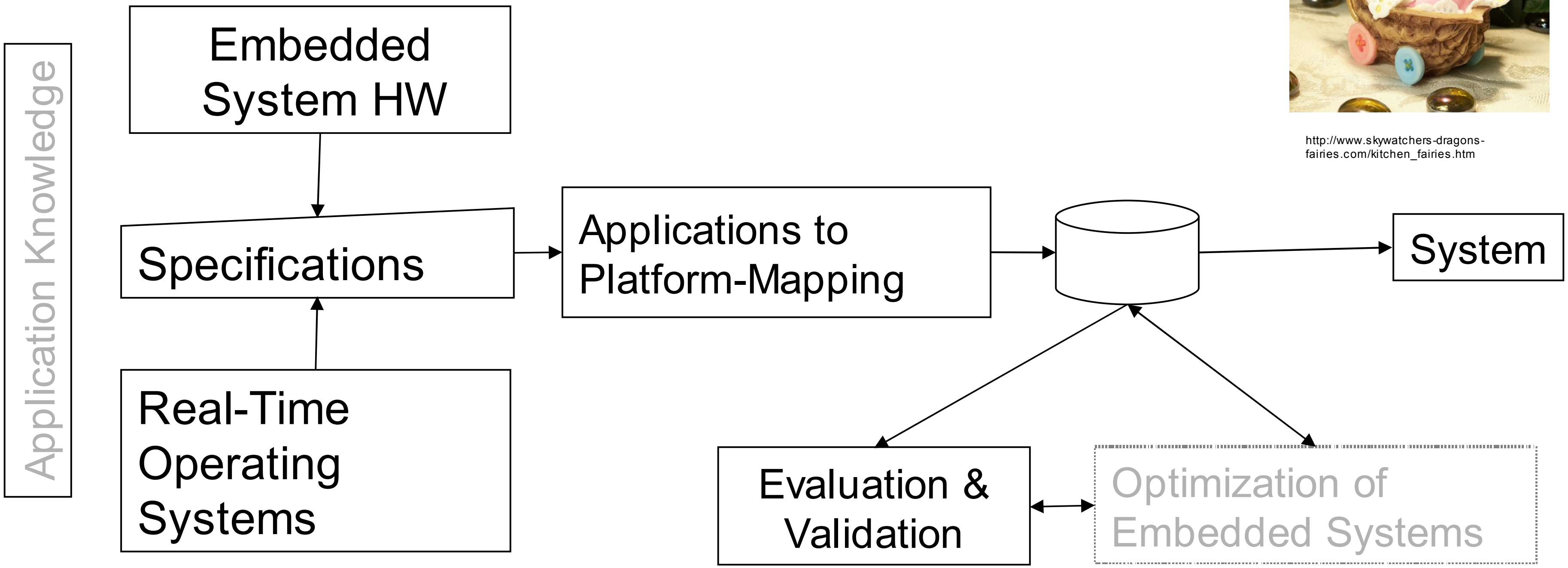
Optimization:

- Huge area, e.g. SPM optimization, WCET aware compilation

Global summary

http://www.skywatchers-dragons-fairies.com/kitchen_fairies.htm

☞ Brief walk-through ends here …



Assignment 1

Start the *levi* learning module leviRTS.

a) Use leviRTS to simulate the example of a failing rate monotonic schedule shown on the slides.

b) Demonstrate that EDF schedules this example without any problem.

Assignment 2

Start the *levi* learning module *leviRTS*.
Model a task set comprising the following tasks:

Task	Priority	Arrival	c_i	Printer		Comm line	
				Δt P	Δt V	Δt P	Δt V
T1	4 (low)	0	20	1	14	4	5
T2	3	2	10	-	-	1	6
T3	2	4	5	-	-	-	-
T4	1 (high)	4	5	1	3	-	-

Use priority-based, pre-emptive scheduling
Which problem occurs? How can it be solved?

Possible Extensions

- Use learning component *leviKPN* to model a Kahn process network computing Fibonacci numbers
- Use component *leviFR* (Flexray™) to model a Flexray bus
- Use Berkeley's *Ptolemy* tools to compare different models of computation
- Use the *EXPO* framework from ETH Zürich to map applications to multi-processor systems

Fibonacci number

From Wikipedia, the free encyclopedia

In mathematics, the **Fibonacci numbers** are a sequence of numbers named after Leonardo of Pisa, known as Fibonacci, whose *Liber Abaci* published in 1202 introduced the sequence to Western European mathematics.

The sequence is defined by the following recurrence relation:

$$F(n) := \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

That is, after two starting values, each number is the sum of the two preceding numbers. The first Fibonacci numbers (sequence A000045 in OEIS), also denoted as F_n, for $n = 0, 1, 2, \ldots$, are:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, ...[1]

Each third number of the series is an *even* number.

The sequence named after Fibonacci was first described in Indian mathematics.[2][3]

The sequence extended to negative index n satisfies $F_n = F_{n-1} + F_{n-2}$ for *all* integers n, and $F_{-n} = (-1)^{n+1} F_n$:

.., -8, 5, -3, 2, -1, 1, followed by the sequence above.