What is an embedded system?

→ an electronic device
→ is embedded into another system / product
→ a computer system dedicated to a certain task
→ it has limited HW & SW functionality

limits could be:
- processing power
- memory
- power consumption

} HW

- limited or no OS
- fewer Apps

} SW

Should Know:

Mention limitations in HW/SW

What is an embedded system?

as well (understanding)

- Almost every electrical product now contains an electronic system to control its operation.
- It is called an **embedded (electronic) system** because it improves the function of the product in some way, but is not itself the primary purpose of the product.
- For example, almost every washing machine is now controlled electronically by a microcontroller (MCU or µC) — a ‘computer on a chip’.

This is an embedded system because the purpose of the product is to wash your clothes, not computation.

In contrast, a PC is not an embedded system because it’s main task is computation of some sort. (Actually it contains embedded systems in the keyboard, disk drives and so on).

Nowadays the world is full of embedded systems, from small scale to large scale, and these can be implemented in many ways.

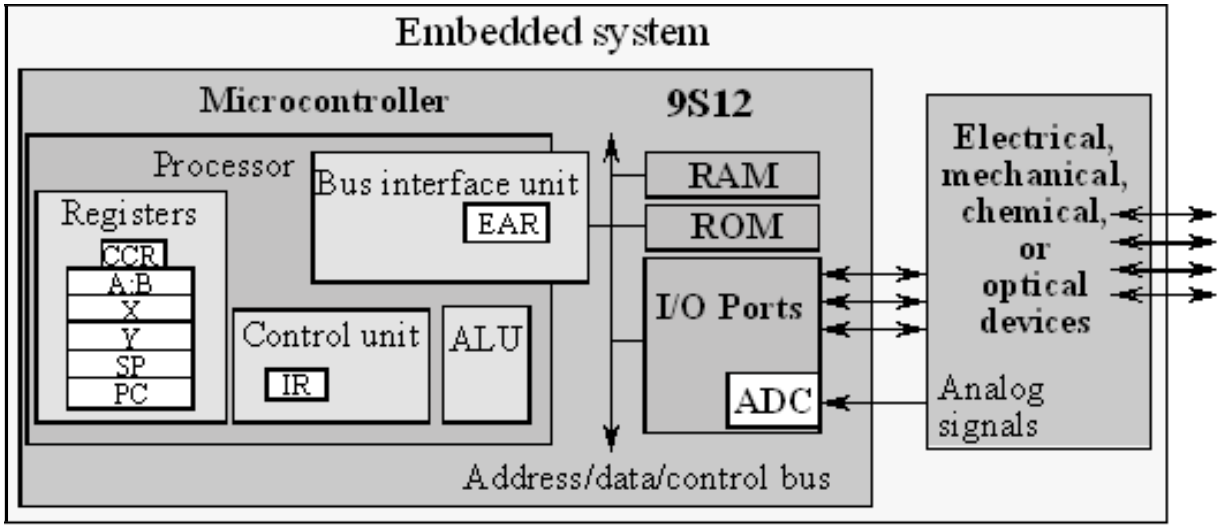
Definition (1): embedded system

“An embedded system **contains a microcontroller** to accomplish its job of processing system inputs and generating system outputs. The link between system inputs and outputs is provided by a coded algorithm stored within the processor’s resident memory. What makes embedded system design so interesting and challenging is the design must also take into account the proper electrical interface for the input and output devices, limited on-chip resources, human interface concepts, the operating environment of the system, cost analysis, related standards, and manufacturing aspects.”

Source: Embedded Systems Design with Atmel AVR Microcontroller Part 1, Steven Barrett , ISBN: 9781608451289, 00/TWQ 82

Definition (2): embedded system

“An embedded computer system is an electronic system that includes a **microcomputer** such as the Freescale 9S12 that is configured to perform a specific dedicated application, drawn in the following figure.”



Source: Embedded Microcomputer Systems: Real Time Interfacing, Jonathan W. Valvano, ISBN: 9781111426255

Definition (3): embedded system

“An embedded system is **a computer system** designed for **specific control functions within a larger system**, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.”

Source: From Wikipedia, retrieved on 24.11.2012

“An **embedded system** is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is *embedded* as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today.”

Source: Michael Barr; Anthony J. Massa (2006). "Introduction". *Programming embedded systems: with C and GNU development tools*. O'Reilly. pp. 1–2. ISBN 978-0-596-00983-0.

Definition (4): embedded system

“Embedded systems are information processing systems embedded into a larger product.”

Source: Peter Marwedel, TU Dortmund

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

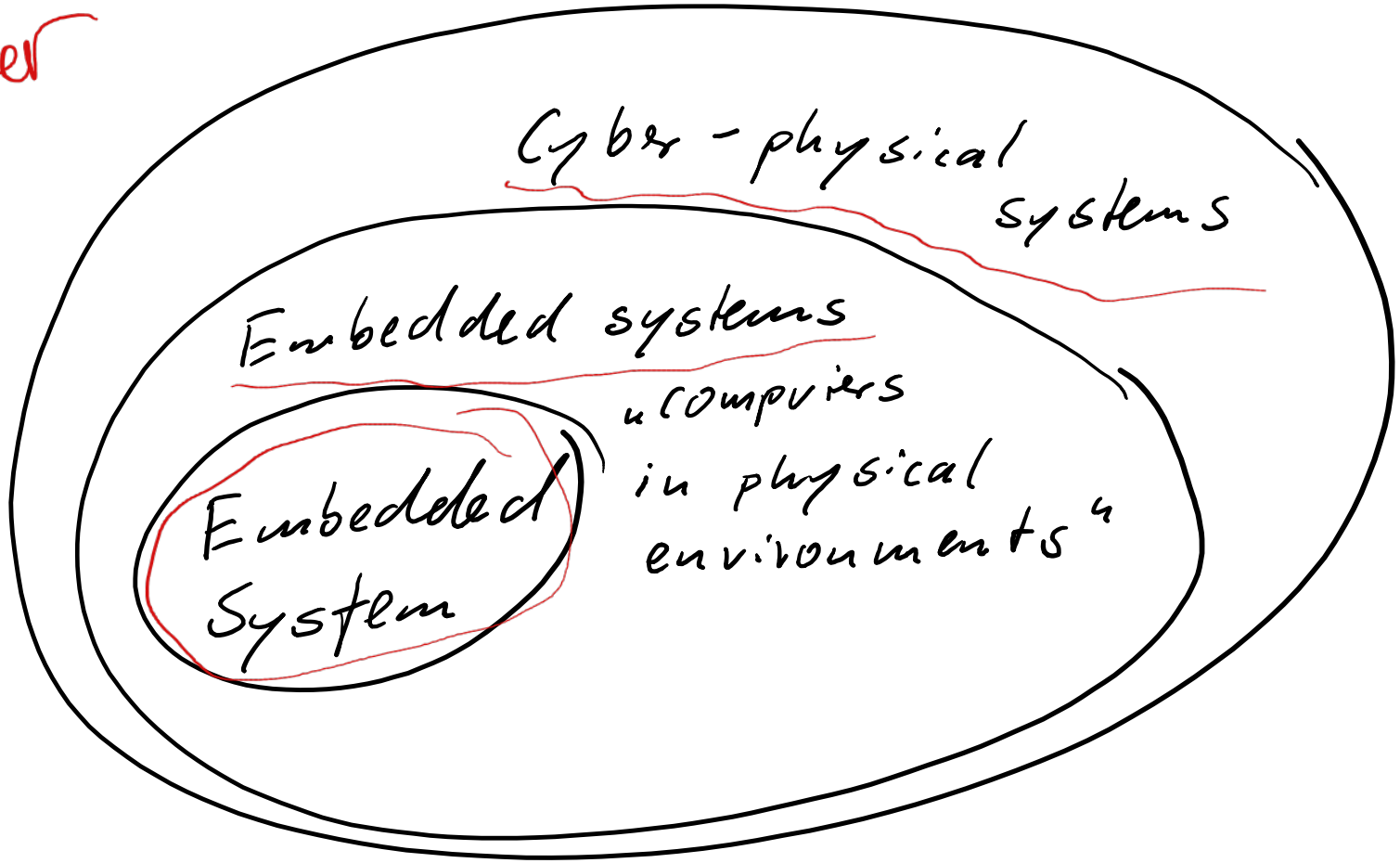
Source: Edward Lee, The Future of Embedded Software, ARTEMIS Conference Graz, University of California, Berkeley, 2006, https://chess.eecs.berkeley.edu/pubs/873/FutureOfEmbeddedSoftware_Lee_Graz.pdf

“Cyber-Physical (cy-phy) Systems (CPS) are integrations of computation with physical processes.”

Source: Edward Lee, Computing foundations and practice for cyber-physical systems: A preliminary report. Technical Report UCB/EECS-2007-72, EECS Department University of California, Berkeley, 2007

Cyber-physical systems (CPS) =
embedded system (ES) + physical environment

Definition (5): cyber-physical system



Embedded systems – Motivation (1)

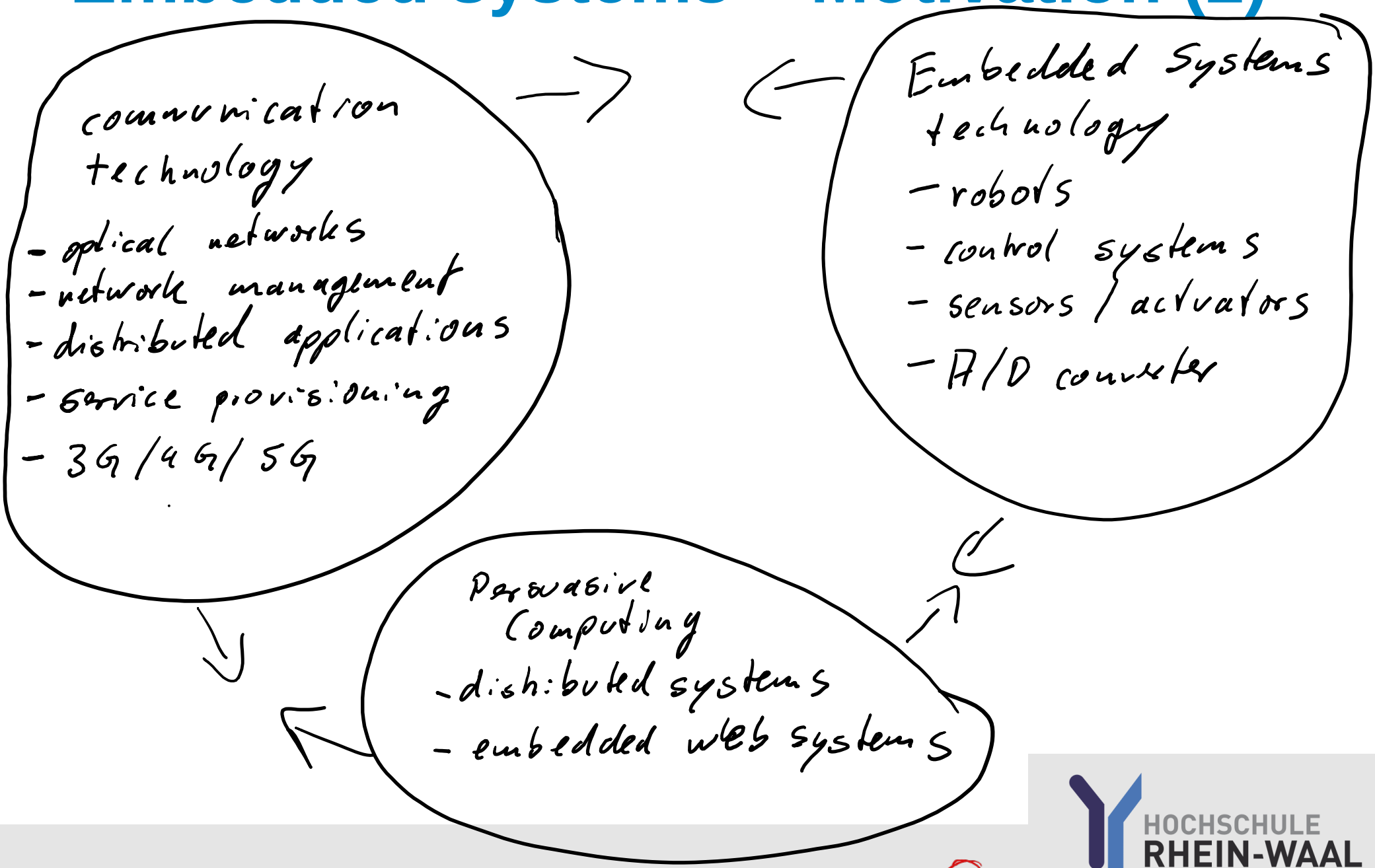
- trend in information processing systems towards:
  - ambient intelligence
  - internet of things
  - ubiquitious computing

why are we doing that?

- requires a holistic approach involving embedded SW & HW and the physical environment
- additional challenges:
  - power / energy
  - cost
  - dependability
  - real time processing
- underrepresented in teaching

⇒ The future is embedded, embedded is the future!

Embedded systems – Motivation (2)



Typical ES constraints

- When we plan an ES we should consider:
  - efficiency *
  - size, weight
  - Environment (humidity, heat, ...)
  - real-time *
  - dependability *
  - extreme cost sensitivity
  - safety critical operation

we looked deeper on these.

need to remember?

ES constraints – customer view

- The customer expects:
  - increased overall dependability
  - reduced cost
  - increased functionality
  - increased performance

ES constrains from the customer's view?

explain depend

Dependability

all

- ES/CPS must be:
  - reliable => r(t) = probability of system working correctly considering it was working correctly at time t=0
  - safety => no harm to be caused, not dangerous
  - security => encrypted communication, authentic communication
  - maintainable => m(d) = probability that the system working correctly d time units after an error occ.
  - available => a(t) = probability that the system works at time t

=> if the system assumptions are wrong, then the system might fail during late use

=> making the system dependable should be one of the first desing considerations

Efficiency

what kind of efficiencies we have?

- ES/CPS must be:
  - energy efficiency (harvesting?)
  - cost "
  - size "
  - weight "
  - code size " (small internal memory)
  - runtime " (should use the available HW as good as possible)

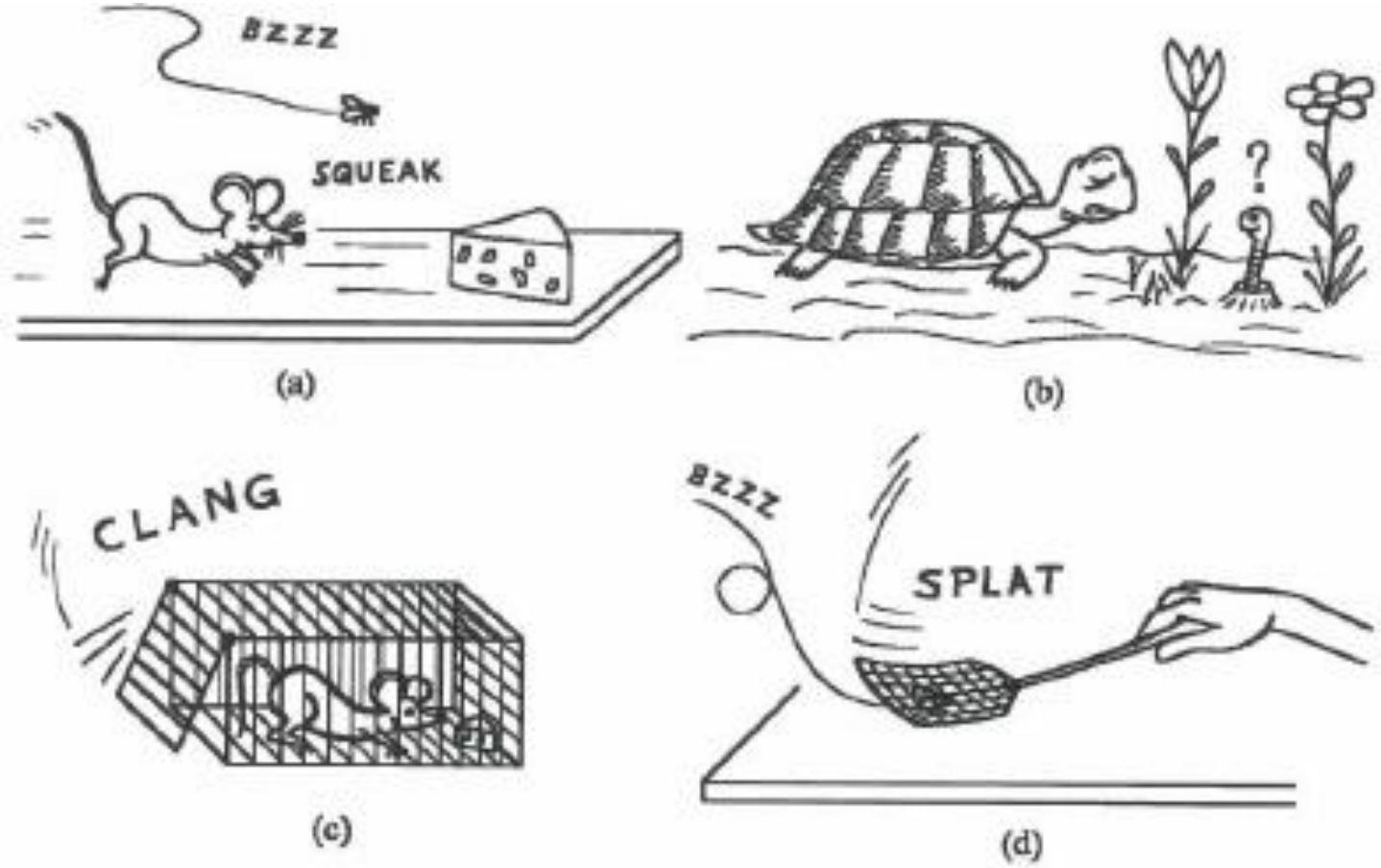
Real-time – Definition (1)

The main characteristic that distinguishes real-time computing from other types of computation is time. Let us consider the meaning of the words **time** and **real** more closely.

The word **time** means that the correctness of the system depends not only on the result of the computation but also on the time at which the results are produced.

The word **real** indicates that the reaction of the system to external events must occur **during** their evolution. As a consequence, the system time (internal time) must be measured using the same time scale used for measuring the time in the controlled environment (external time).
Although the term real time is frequently used in many application fields, it is subject to different interpretations, not always correct. Often, people say that a real-time system is **real time** if it is fast. The term fast, however, has a relative meaning and is NOT correct.

Real-time – Definition (2)



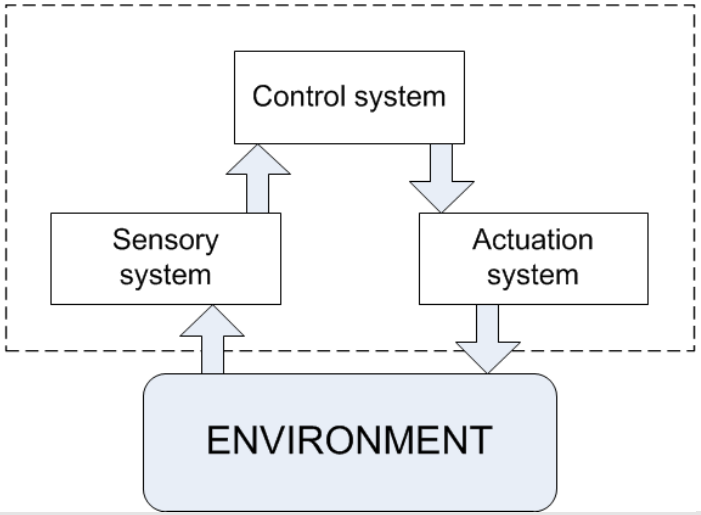
Both the mouse (a) and the turtle (b) behave in real-time with respect to their natural habitat. Nevertheless, the survival of fast animals can be jeopardized by events (c and d) quicker than their reactive capabilities.

Real-time – Definition (3)

Real time ≠ fast!

The previous examples show that the concept of time is not an intrinsic property of a control system, either natural or artificial, but that it is strictly related to the environment in which the system operates. It does not make sense to design a real-time computing system for flight control without considering the timing characteristics of the aircraft.

Block diagram of a generic real-time control system



Real-time – Constraints (1)

- An ES/CPS must usually 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.

- Depending on the consequences that may occur because of a missed deadline, a real-time task can be distinguished in three categories:
  - **Hard:** A real-time task is said to be *hard* if producing the results after its deadline may cause catastrophic consequences on the system under control.
  - **Soft:** A real-time task is said to be *soft* if producing the results after its deadline has still some utility for the system, although causing a performance degradation.
  - **Firm:** A real-time task is said to be *firm* if producing the results after its deadline is useless for the system, but does not cause any damage. (is nowadays categorized into *Soft*)

Real-time – Constraints (2)

What question?

ES, CPS and Real-time systems synonymous?

- For some ES, the real-time behaviour is less important (e.g. tablets, digital cameras, smartphones) → no one will be harmed if you have to wait a few milliseconds until the picture is taken
- For CPS, the real-time behaviour is essential, therefore we can say that real-time systems ≅ CPS
- CPS models also include a model of the physical system

what means Dedicated systems (1)?

- Most ES do not use mouse, keyboard or monitors as user-interfaces.
- Instead they have dedicated user interface such as:
  - LCD or LED
  - push buttons
  - steering wheel
  - pedals

dedicated to one task: otherwise it's computersystem.

- task
- user interface

- These systems are dedicated towards a certain application.
  → Knowledge about behaviour at design time can be used to minimize resources and to maximize robustness of the ES

=> due to this the user can hardly see that information processing is involved

=> because of that this new era of computing is often called „the dissappearing computer"

Dedicated systems (2)

Example

Processors running control software in a car will usually always run the same program. There will be no attempt to run a game or an office program on the same ES.

There will be manly two reasons for that:

1. Running additional programs would make this systems less dependable.
2. Running additional programs is only possible if resources are available. But by design, no unused resources should be available in an ES.

However the situation is slowly changing. Mobile phones for example are more and more becoming PC-likely and can hardly be called CPS.

That means systems become less dedicated!

Challenges for SW implementation

Look at Important

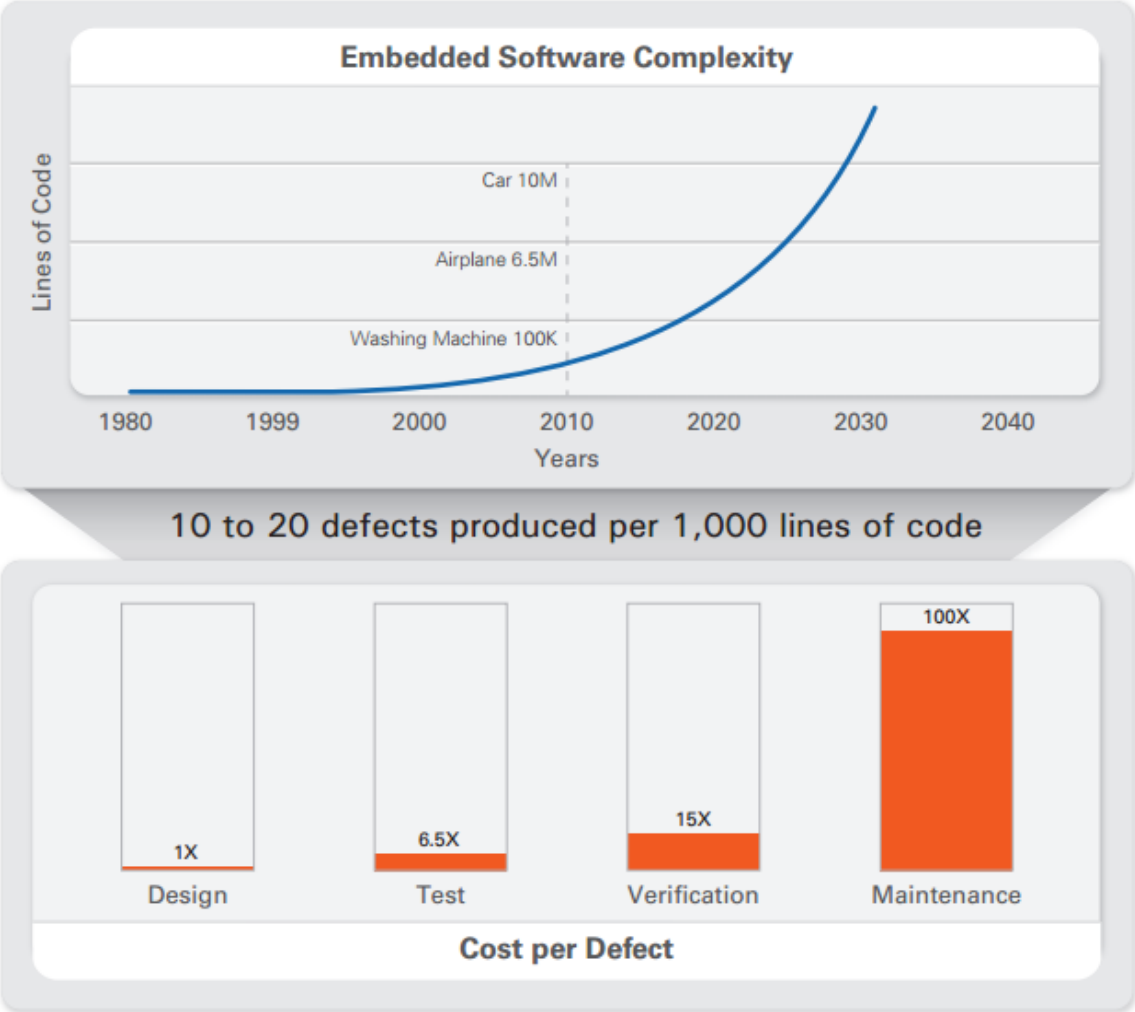
If ES/CPS are mostly implemented in SW, why are we not using approaches used by SW engineers?

- less efficiency in ressource usage
- SW engineers do not have contact with the environment
- SW engineers usually do not have access to the ES/CPS
- knowledge from many other areas must be known
-

⇒ it is not sufficient to consider ES/CPS as a special case of SW engineering

SW complexity

- We can see a exponential increase in SW complexity
- …> 70% of the development costs for complex systems such as automotive electronics and communication systems are due to software development. (A. Sangiovanni-Vincentelli)
- As earlier a SW problem can be found as cheaper is the fixing

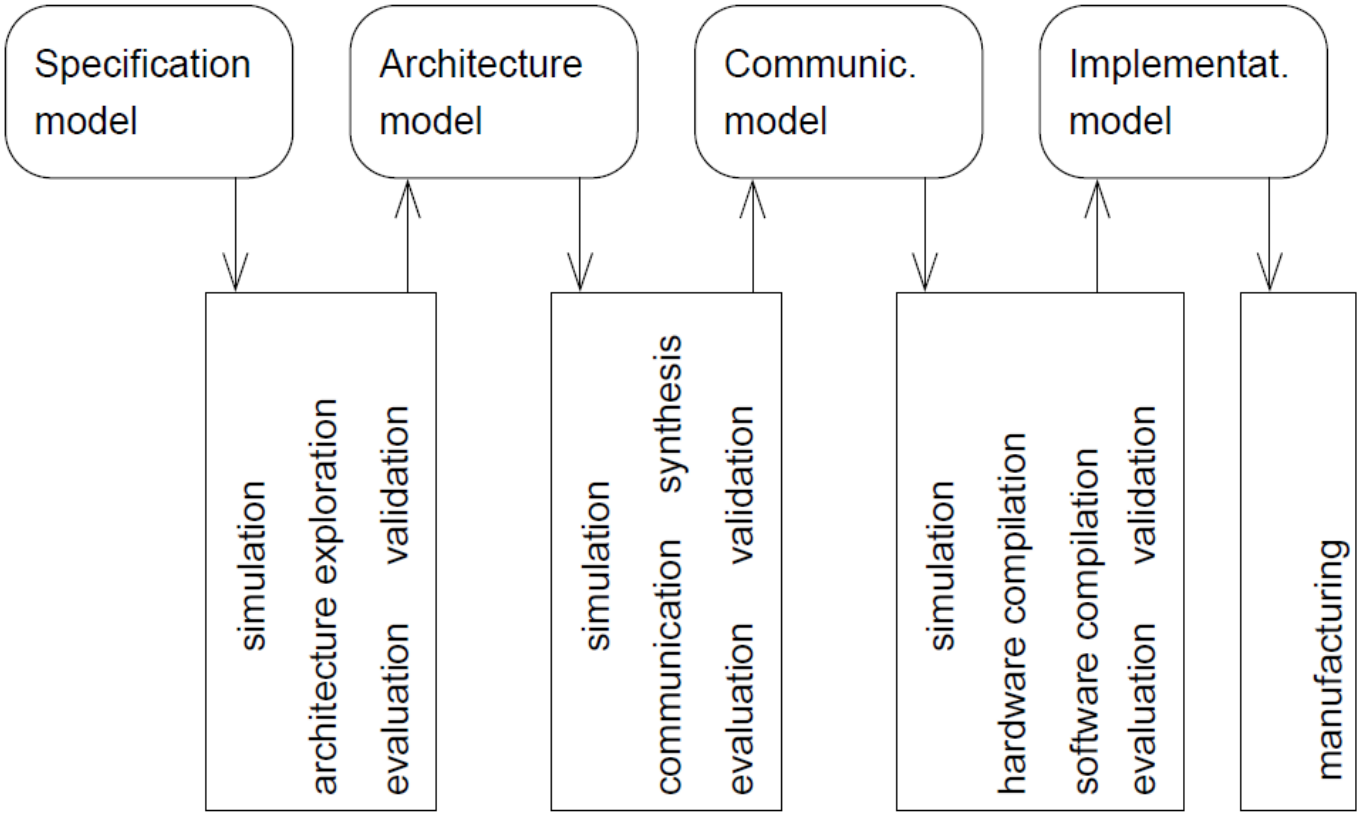


Source: http://www.ni.com/white-paper/52165/en/, Retrieved: 28/08/2015

Iterative design (1)

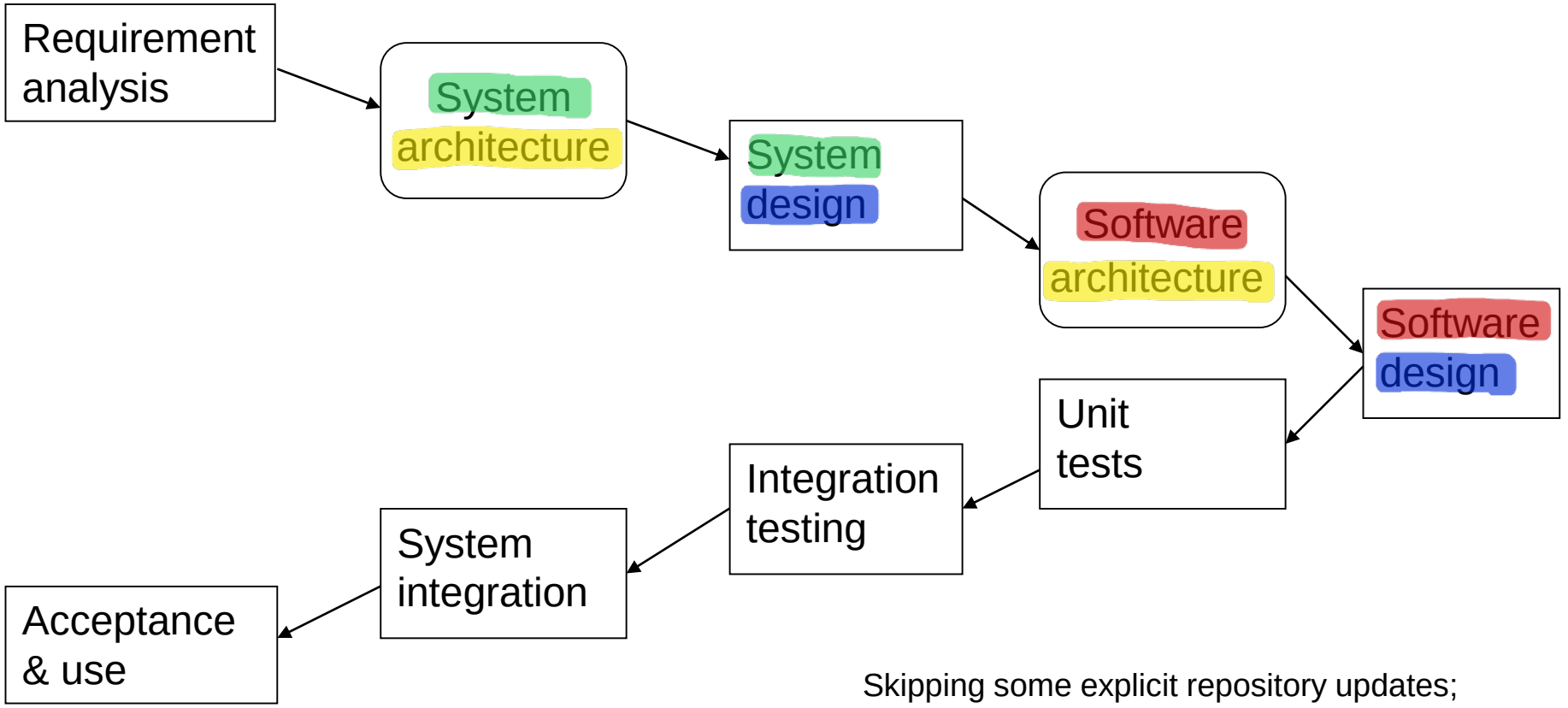
- "unrolling" the loop from the last slide and applying this to a specific design process might end up in different solutions.

- Example: SpecC tools



Iterative design (2)

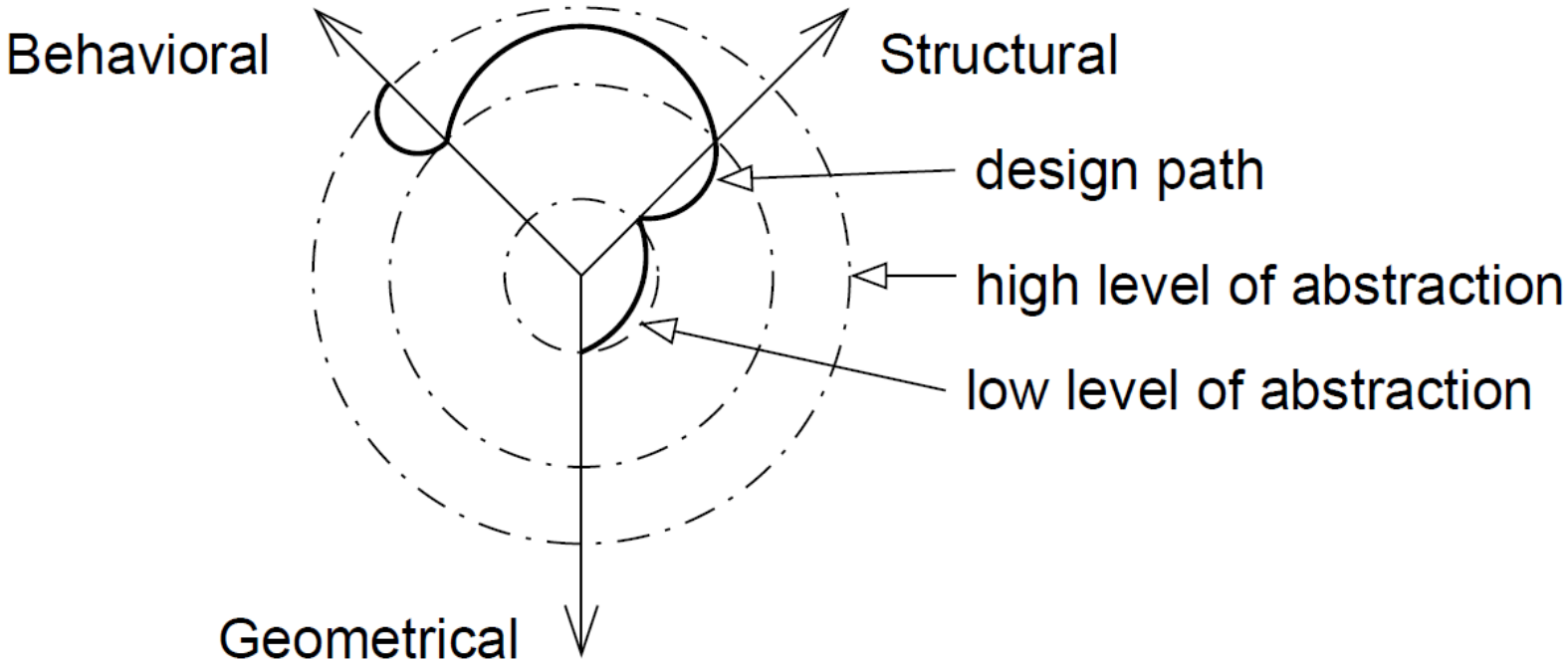
- Example: V-model (version 97)



Skipping some explicit repository updates;
very late integration, problems may be missed ..

Iterative design (3)

- Example: Gajski's Y-chart



Model - definition

what is the model

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)

What are the requirements for a model?

→ contain only relevant information

→ remove all unnecessary information

What are Requirements (1) to the model?

Hierarchy

- Humans are usually not capable of comprehending complex systems (systems with many objects)
- Humans are also not capable to understand complex relations between objects of a complex system

⇒ hierarchy + abstraction is a key mechanism to solve this dilemma
↳ this helps looking at a fewer number of objects at any time

⇒ we differentiate: 1. behavioural hierarchies: - describes objects necessary for the syst. (e.g. states, events, outp.)

2. structural hierarchies: - describes the components of the system (e.g. Processors, registers)

Requirements (2)

Component-based design

- Systems must be designed using “components”
- The behaviour of the system must be discoverable from the behaviour of the components

what means **Concurrency** ?

- Real-life systems are usually distributed and concurrent systems
- Humans are not very good at understanding concurrent systems and its complex behaviour

Synchronization & communication

- Components must be able to communicate and to synchronize
- Without the communication, components would not be able to work together

Requirements (3)

Timing-behaviour

- Timing is essential for ES/CPS as time is one of the key dimensions of physics
- As many ES/CPS are real-time systems, timing requirements must be captured in the specifications
- The underlying platform and its speed must be known
- Additional information such as periods, dependencies and usage scenarios are beneficial to know
- All of these can have an impact on the design process

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

Req. for specs & modeling techniques (1)

- State-oriented behaviour
- Event-handling
- Exception-oriented behaviour
- Presence of programming elements
- Executability & Readability
- Support for the design of large systems
- Domain-specific support

FEETS

name only 5 out of this and the after page

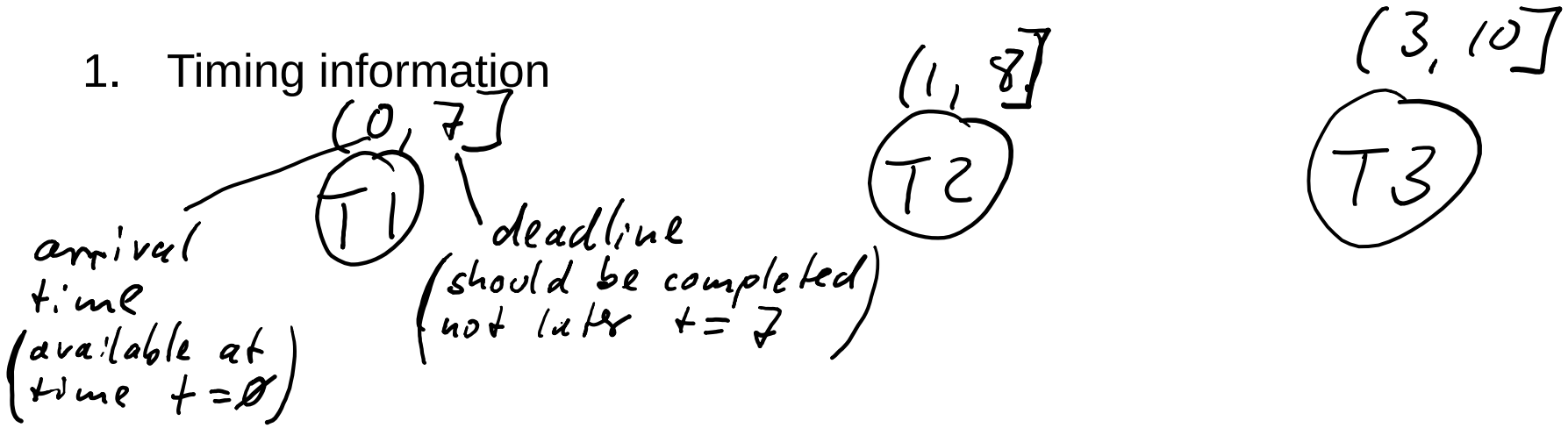
Req. for specs & modeling techniques (2)

- Portability & flexibility
- Termination
- Support for non-standard I/O devices
- Non-functional properties
- Support for the design of dependable systems
- No obstacles for efficient implementation
- Appropriate model of computation

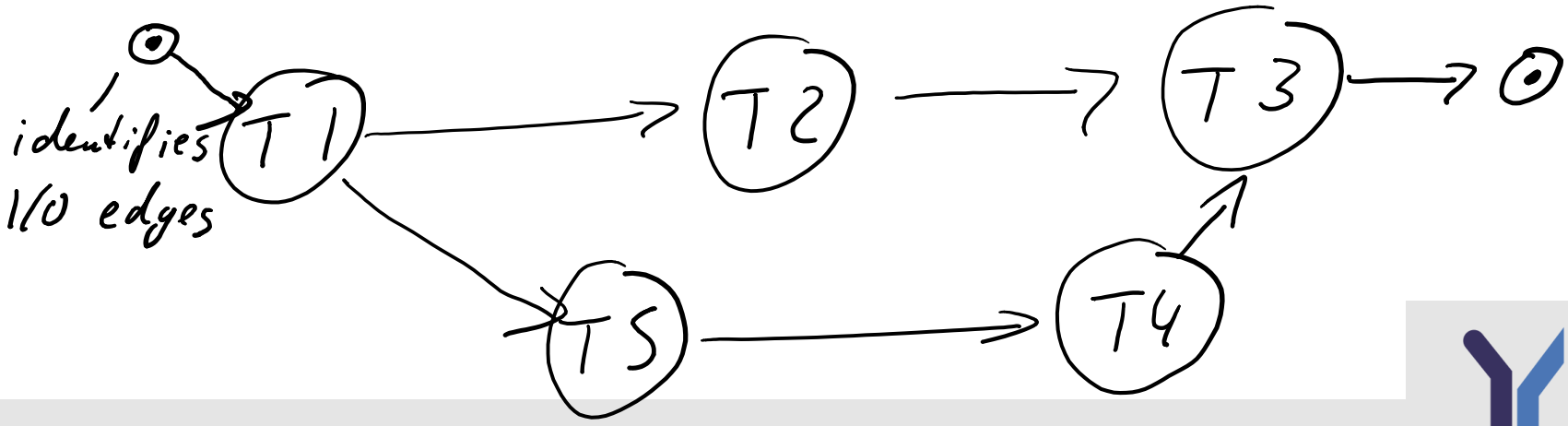
Dependence graph – Definition (2)

Dependence graphs may contain additional information such as:

1. Timing information



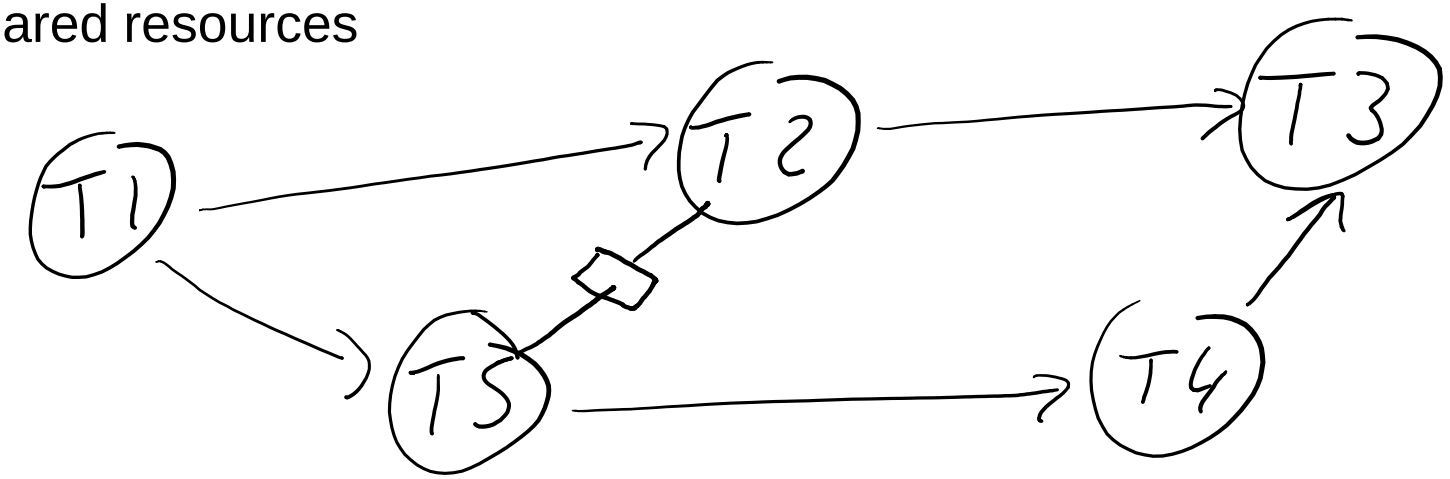
2. I/O information



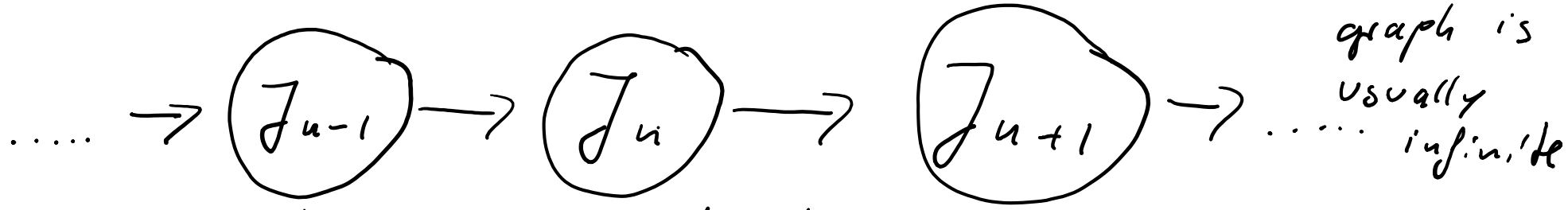
Dependence graph – Definition (3)

Dependence graphs may contain additional information such as: sketch

3. Shared resources



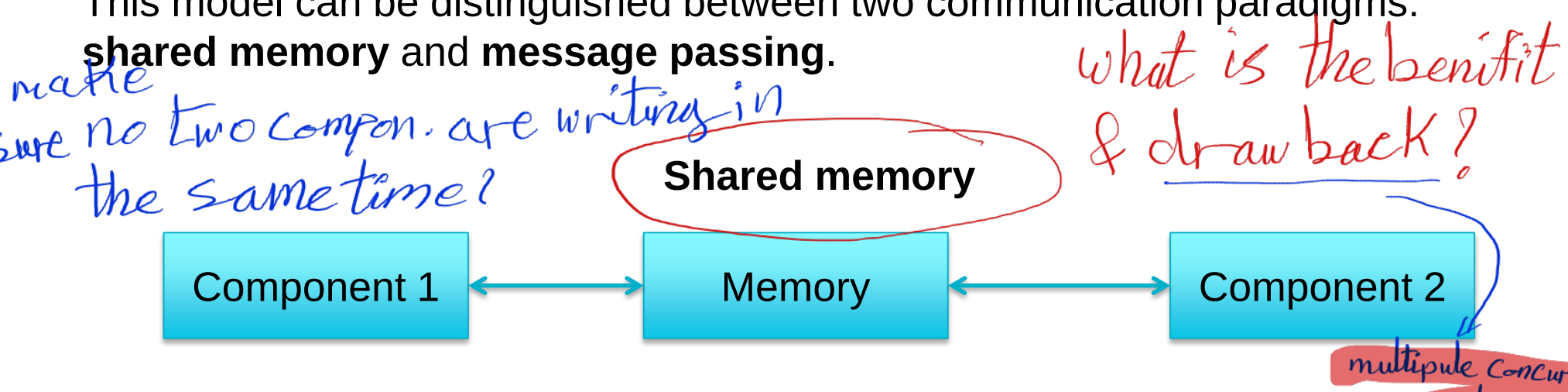
4. Periodic schedules



– many computations are periodically

Models of Communication (1)

This model can be distinguished between two communication paradigms: **shared memory** and **message passing**.



- Communication is carried out by access to the same memory from all components
- Access should be restricted (except it is read only)
- For writes, exclusive access to the shared memory must be guaranteed (called critical sections)
- Is a very fast form of communication
- Difficult to implement on multiprocessor systems when no common memory

Models of Communication (3)

Asynchronous message passing (non-blocking communication)

- Components communicate by sending messages through channels
- Each channel has the ability to buffer messages
- The sender does not have to wait for the recipient to be ready for the next message
- Problems: – if buffer is full → buffer overflow

synchronous message passing (blocking communication, rendezvous)

- Components communicate in instantaneous actions called rendezvous
- The components have to wait until both partners are ready to communicate
- Sender will wait until receiver has received the message
- No buffer overflow
- Problems: - reduced performance

Models of Communication (4)

Extended rendezvous

- Sender is only allowed to continue with the communication after an acknowledgement have been received from the recipient
- The acknowledgement can be send after the message has been checked

understand

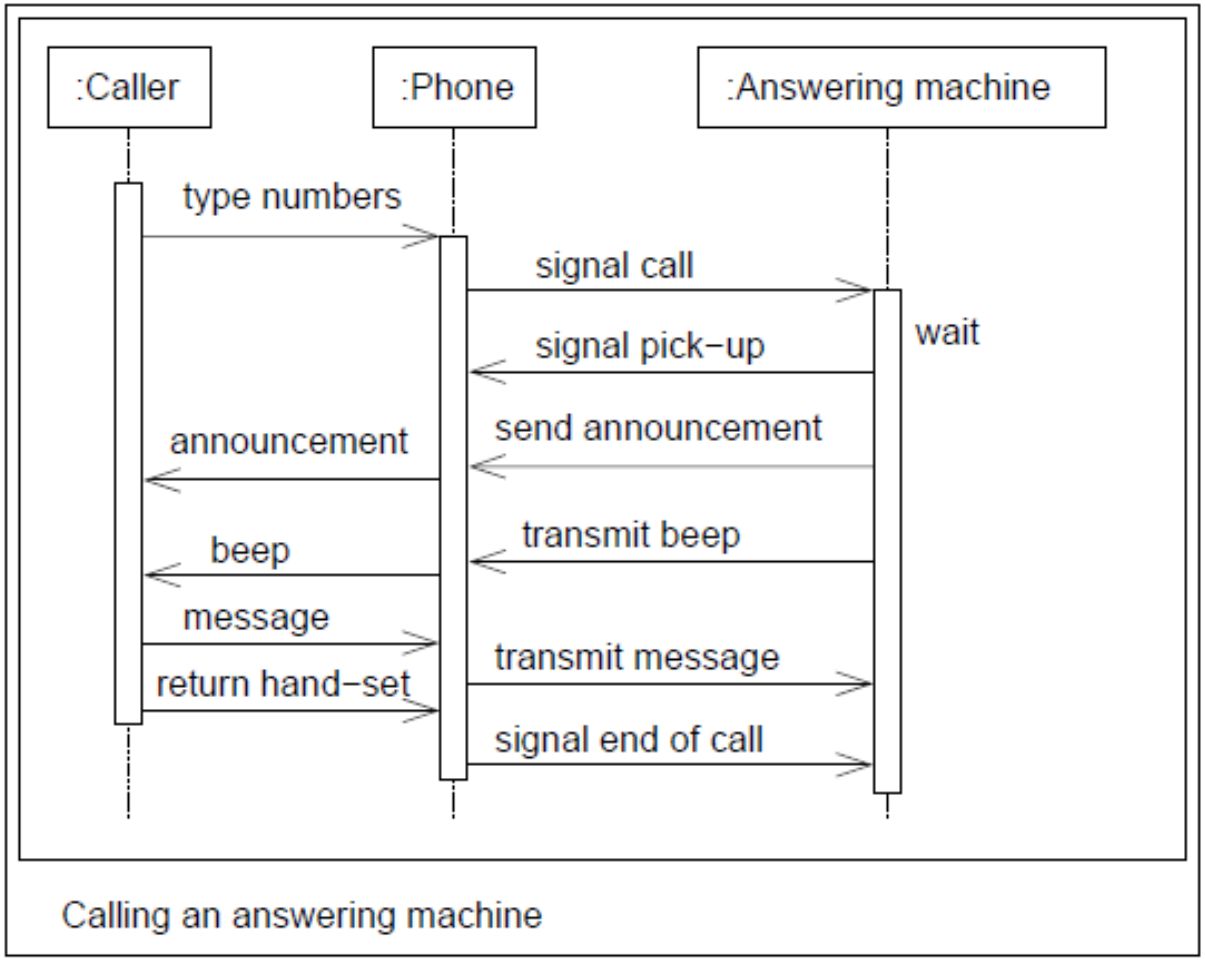
(Message) Sequence charts (1)

- Sequence charts (SC's) are used to describe a more detailed level of messages which must be exchanged between components of a SUD
- Are usually designed in 2-dimensions (vertical dimension denotes time, horizontal dimension denotes the different components)
- SC's describe a possible behaviour of the SUD
- SC's are also described in UML
- Message sequence charts (MSC) is an old term for SC's
- Complex control-dependent actions cannot be described by SC's

(Message) Sequence charts (2)

Example: Answering machine

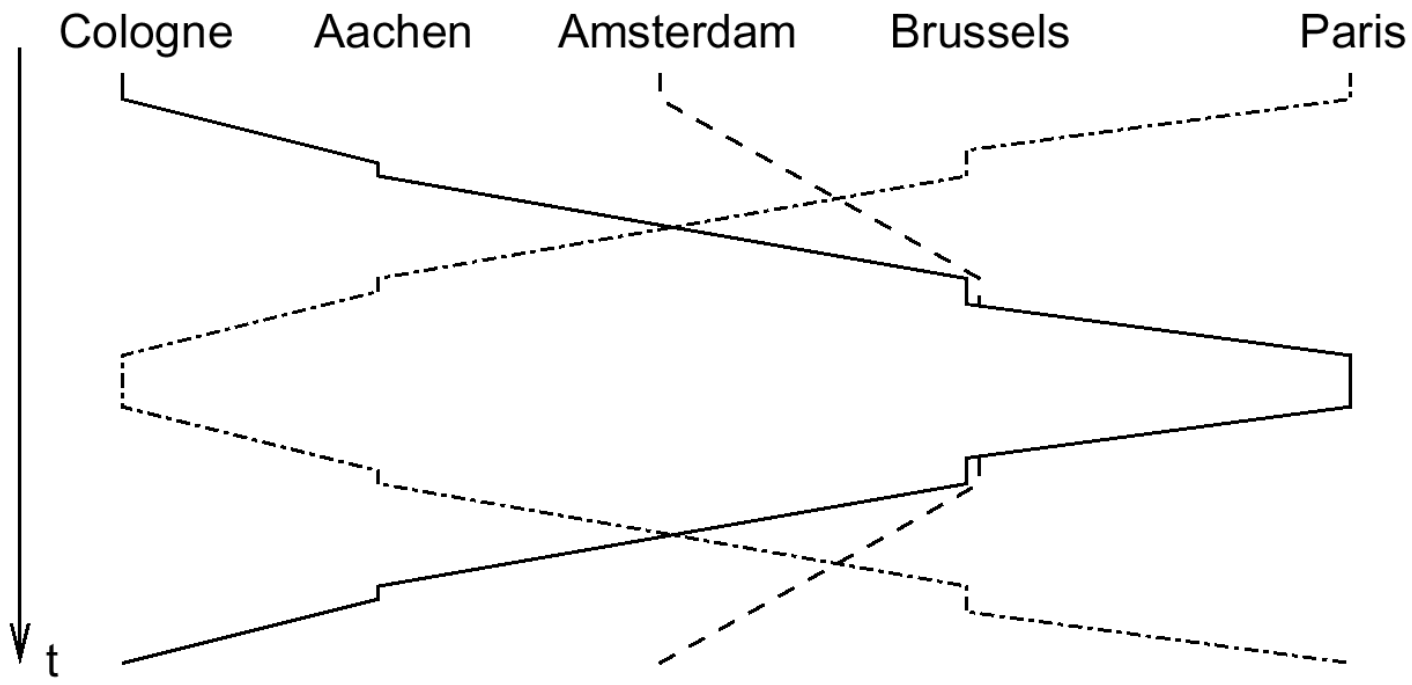
What question?

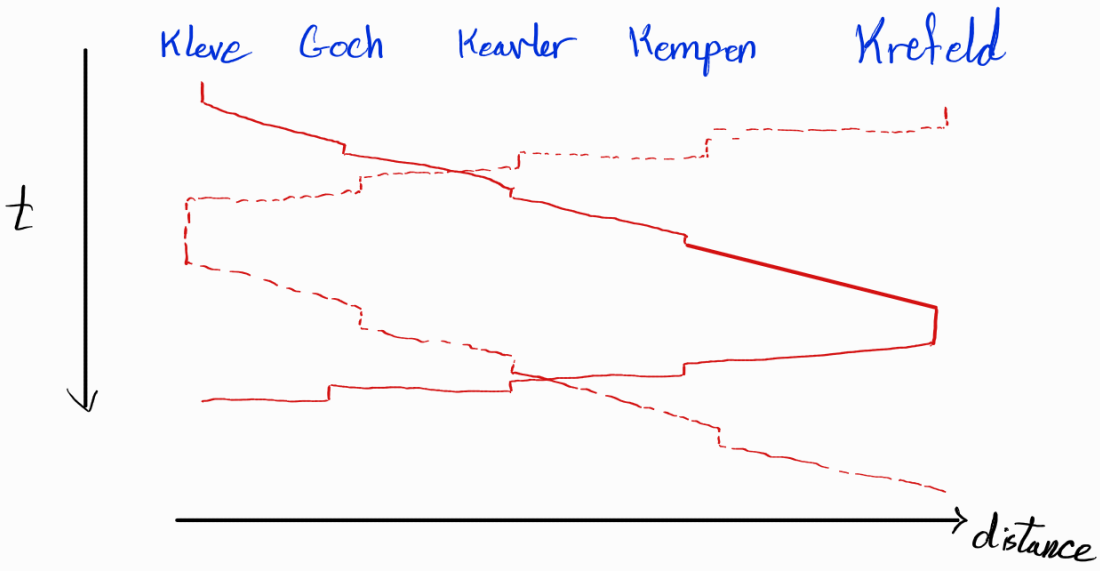


Calling an answering machine

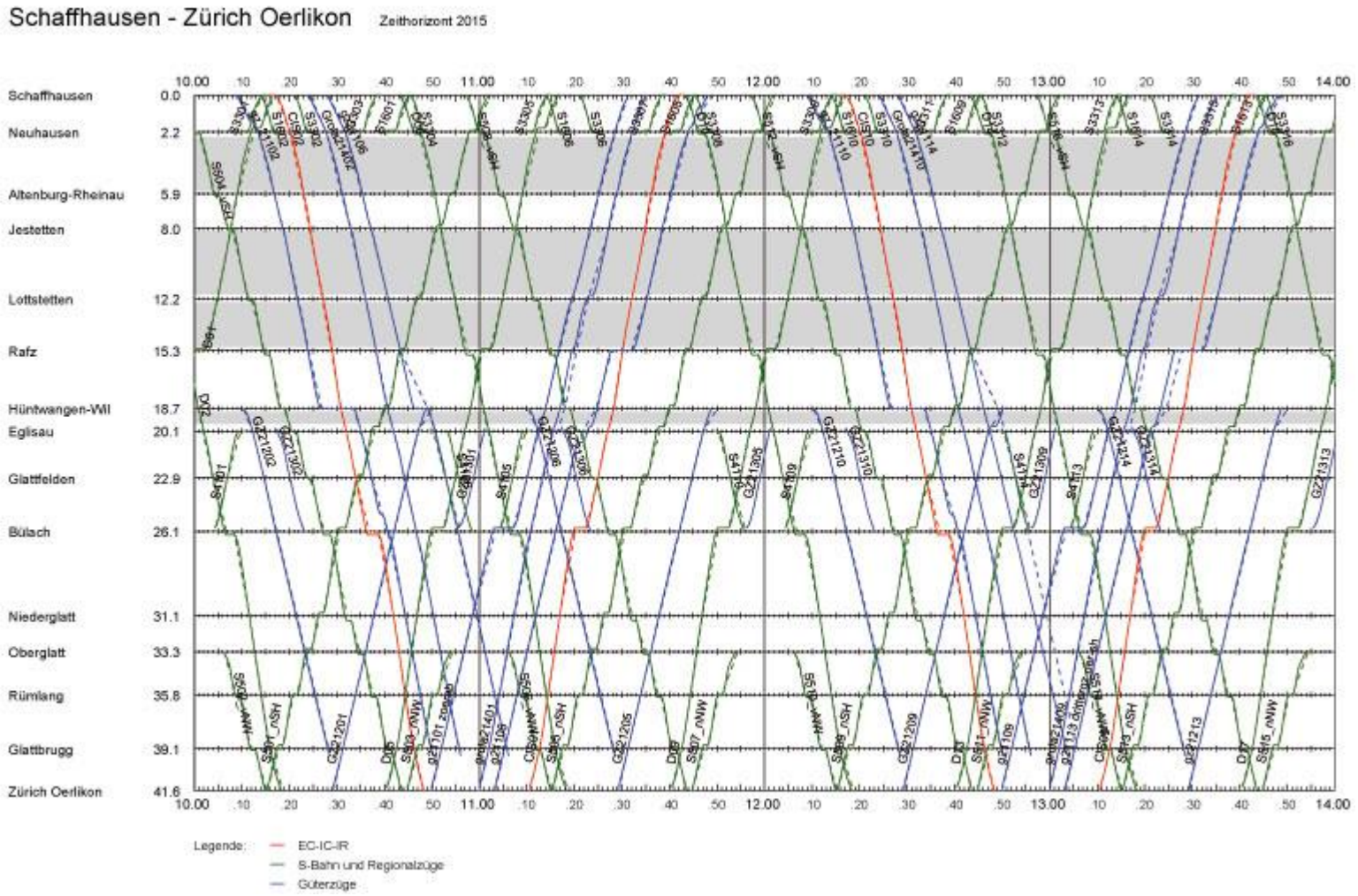
Time distance diagrams (TDD's) (1)

- TDD's are a commonly used variant of an SC's
- The vertical dimension reflects real time, whereby the horizontal dimension (might) reflect real distance
- Is often used to visualize bus or train schedules
- Example: train between Cologne and Paris




Kleve
Goch
Kevelaer
Kempen
Krefeld
t
distance

Time distance diagrams (TDD's) (2)



Source: http://archiv.ethlife.ethz.ch/images/OpenTrack1-l.jpg (02/09/2015)

(Message) sequence charts

- Pro:
  - easy visualization of schedules
  - proven method in public transport (representing schedules)
  - standardized in ITU-TS Z.120
  - semantics are also defined within the standard

- Con:
  - describes just one case
  - no timing tolerances
  - it is not clear if an SC describes all or just a set of sample behaviours of a system

StateCharts - Overview

- FSM – classic automata
- Timed automata
- StateCharts
- Timers & Edge labels
- StateCharts – simulation phases
- Steps
- Broadcast mechanism
- Events & Conflicts
- Synchronous languages
- Summary

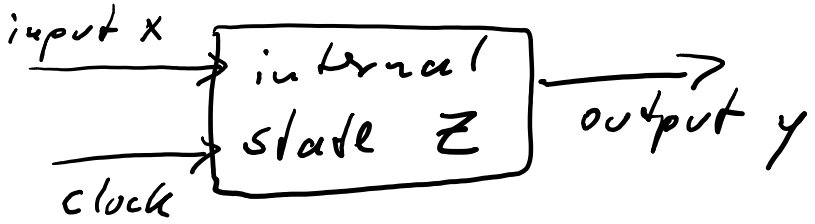
what are classical FSM?

Moore:

Mealy:

FSM – classical automata

- A classical automata is for example a Moore and Mealy automata
- Next state Z^+ is computed by function δ
- Output computed by function λ



- Moore:

  $Y = \lambda\,(Z);\quad Z^+ = \delta\,(X, Z)$

  → output is only based on current state

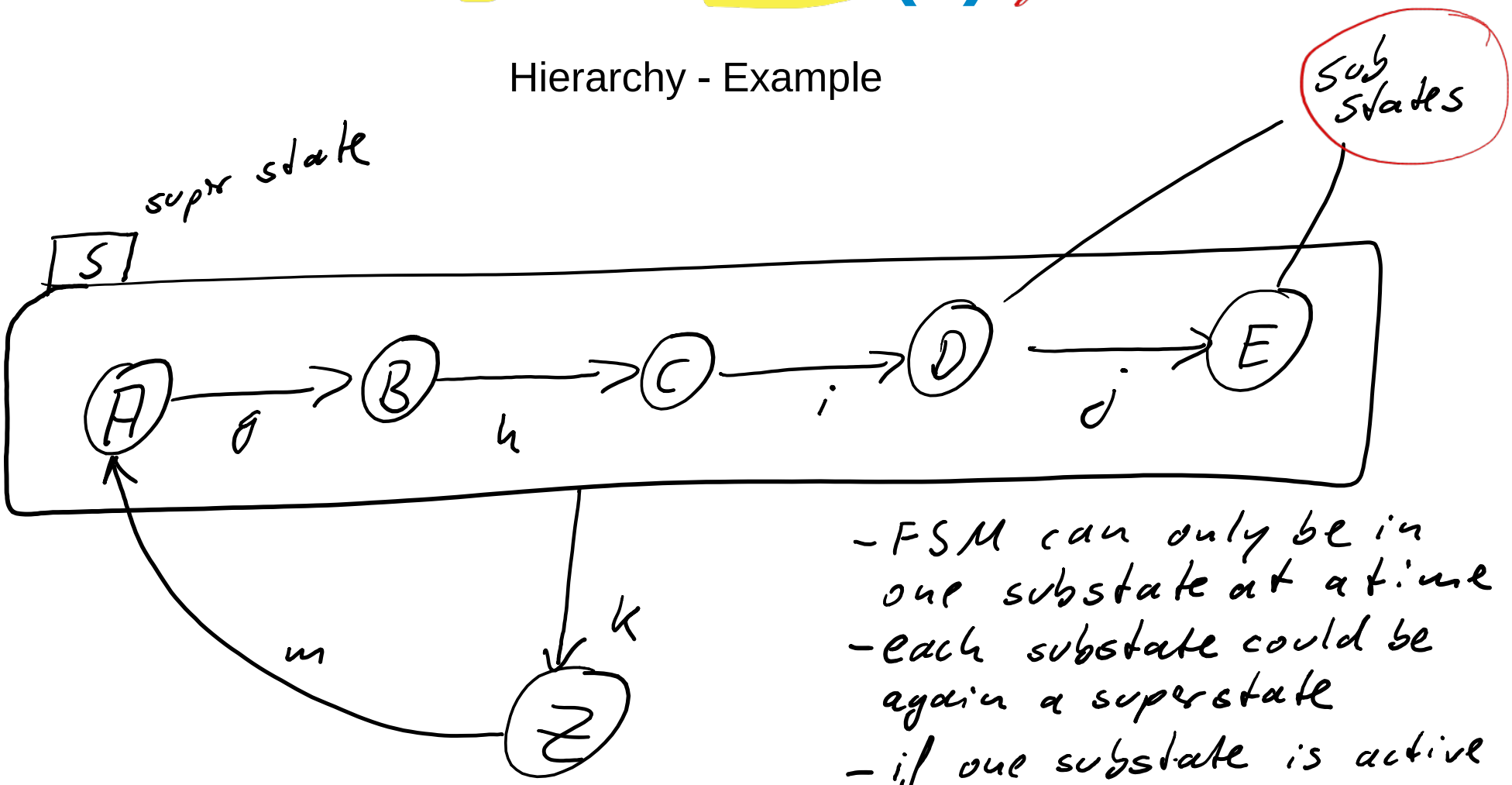
- Mealy:

  $Y = \lambda\,(X, Z);\quad Z^+ = \delta\,(X, Z)$

  → output is based on current state & input

what is StateCharts (2)?

Hierarchy - Example



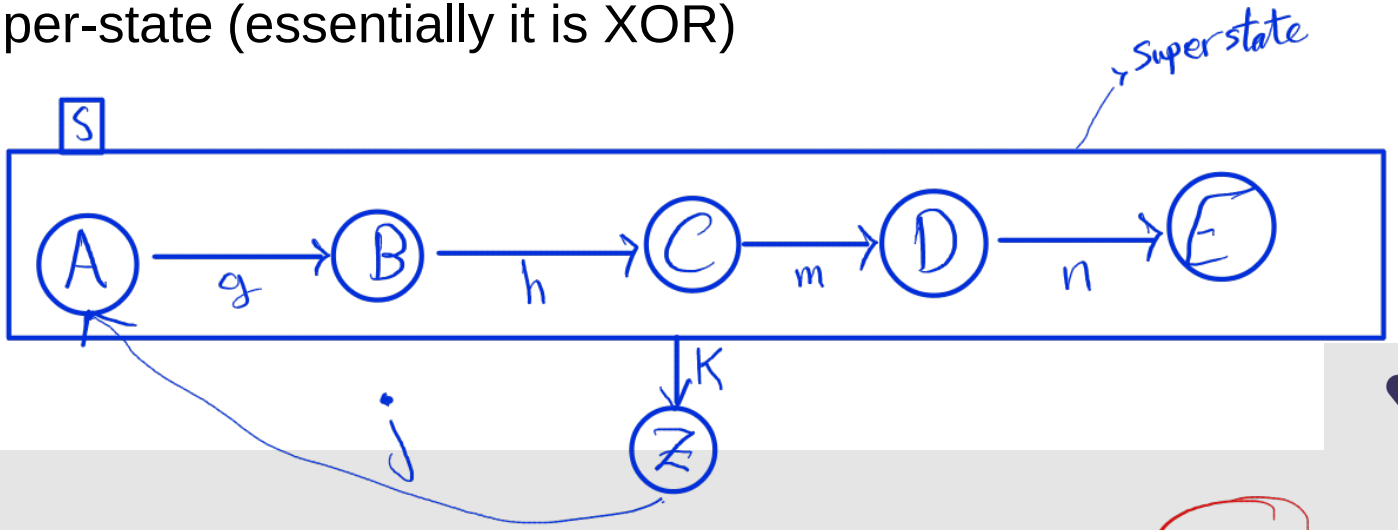
- FSM can only be in one substate at a time
- each substate could be again a superstate
- if one substate is active
  ↳ superstate is active too

Sketch statechart which cotains 5 basic state 5 sub-states & 2 super state?

StateCharts (3)

Definitions

- Current states are called *active* states
- States which are not composed of other states are called *basic* states
- States containing sub-states are called super-states
- If only one sub-state can be active at any time we call the super-state OR-super-state (essentially it is XOR)

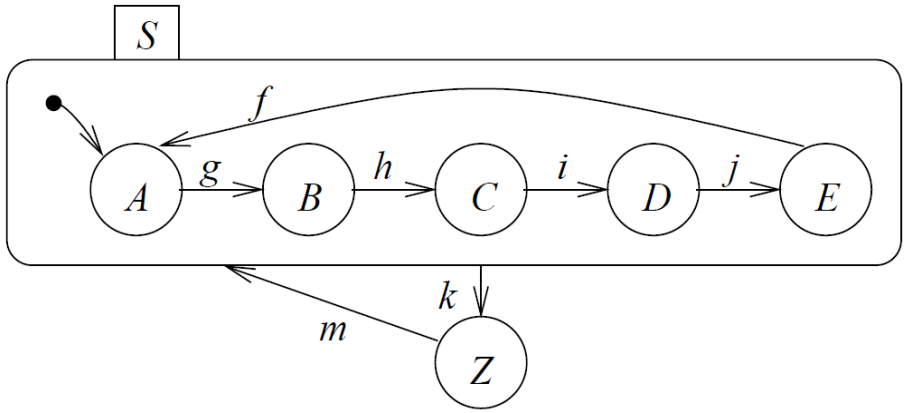


StateCharts (4)

mechanism of sketching

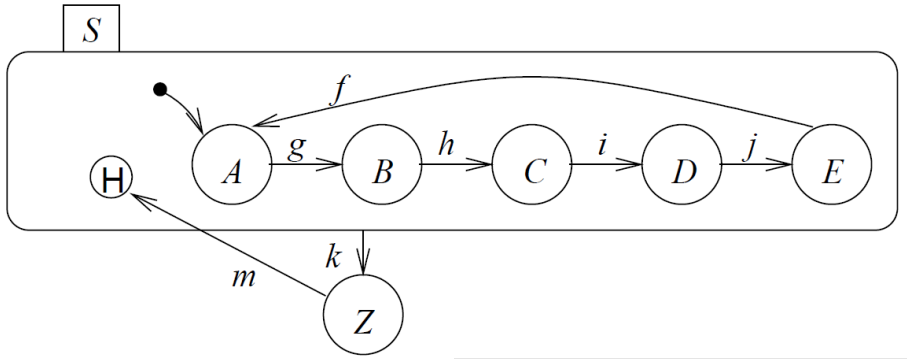
Default state

- The default state can be indicated by a small filled circle
- This small circle is no state itself
- Default state is entered when super-state is entered



History mechanism

- For a transition from Z to S (if input m occurs), S will enter the last sub-state
- Default state might still be present for the situation that S is entered the first time



StateCharts simulation phases

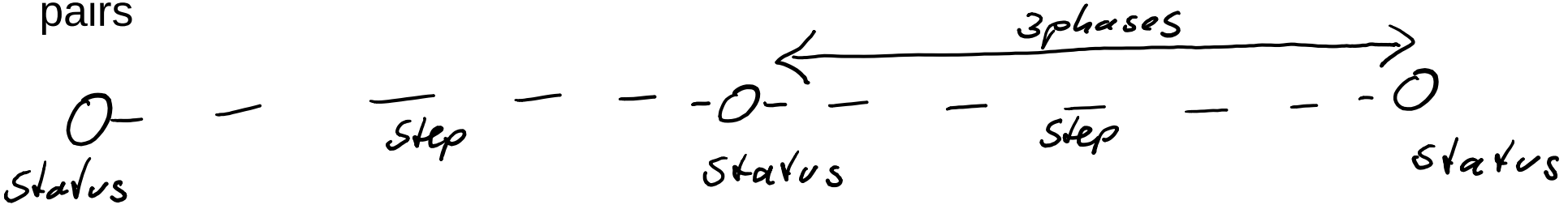
Edge labels are evaluated in three phases (StateMate[Drusinsky and Harel, 1989]):

1. The impact of external changes on events and conditions is evaluated,
2. Calculation of the set of transitions needed for the next step & evaluation of variable assignments (new values are not assigned yet)
3. State transition become active & variables obtain their new values

The separation into phases 2 & 3 allows to guarantee a reproducible behaviour

Steps

- The execution of a StateMate model consists of a sequence of status-step pairs

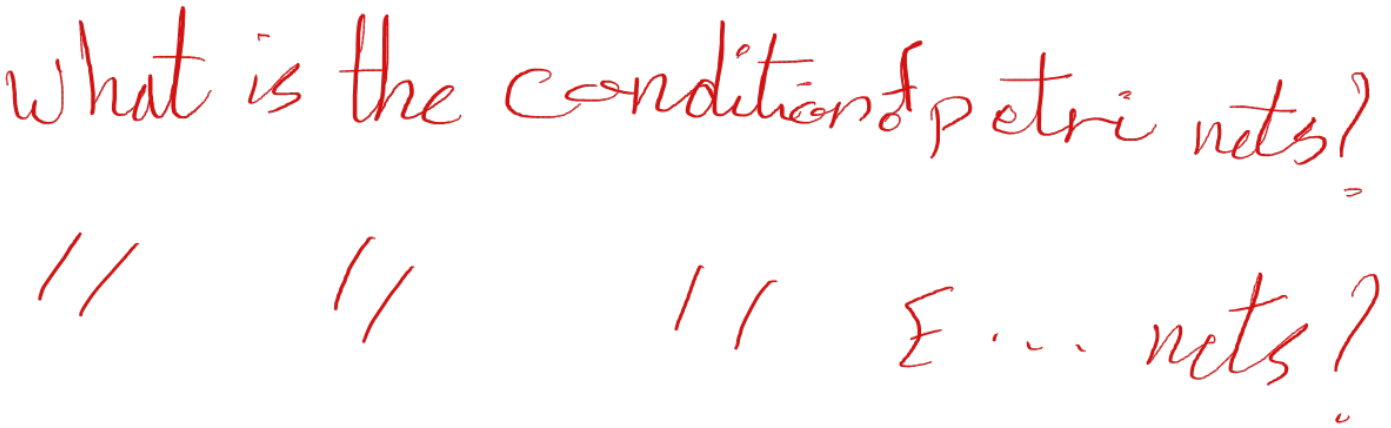


- Status: values of all variables & current time & set of events
- Step: consists of the execution of the 3 phases (StateMate)

- Not all implementations of StateCharts will have these 3 phases, which might lead to different results!

Embedded Systems

Petri Nets



Petri nets

- Can be used to create very comprehensive descriptions of control flows
- They model only control and control dependencies
- For modeling data, petri nets must be extended
- Introduced by Carl Petri in his PhD thesis in 1962, focusing on modeling casual dependencies.
- They do not assume a global synchronization and are therefore especially suited to model distributed systems.
- Consists of three key elements: **conditions**, **events** and **flow relation**

Petri nets – key elements

- **Conditions:**
  - are satisfied or not satisfied

applicable to the train line

- **Events:**
  - may happen if certain conditions are met

- **Flow relation:**
  - describes the conditions that must be met before events can happen, describes conditions which become true if events happen

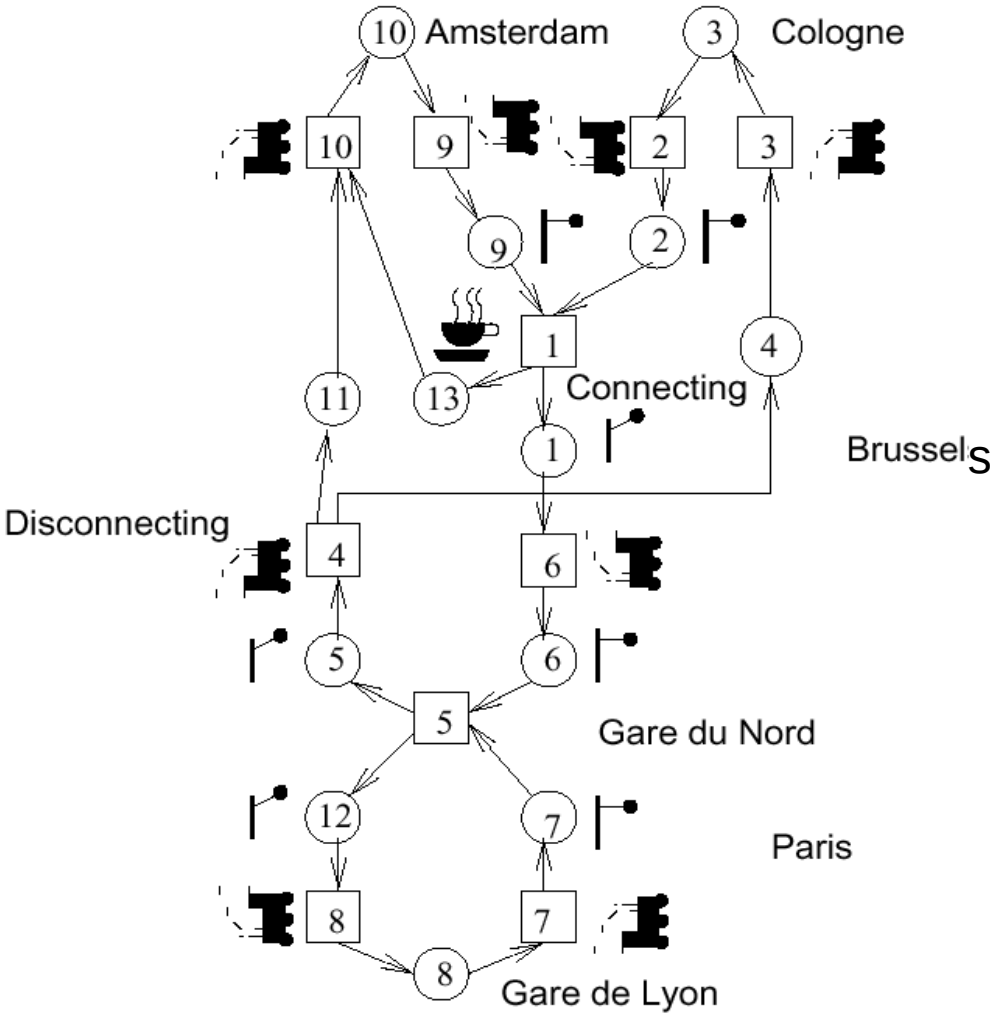
- Conditions, events and the flow relation form a bipartite graph (two kinds of nodes).

Place/Transition nets (1)

- Can allow more than one token per condition (in contrast to C/E nets which allow only one token)
- **Places** correspond to conditions (in C/E nets)
- **Transitions** correspond to events (in C/E nets)
- Number of tokens per place is called a **marking**
- A marking is a mapping from the set of places to the set of natural numbers extended by the symbol ω (describes infinity)

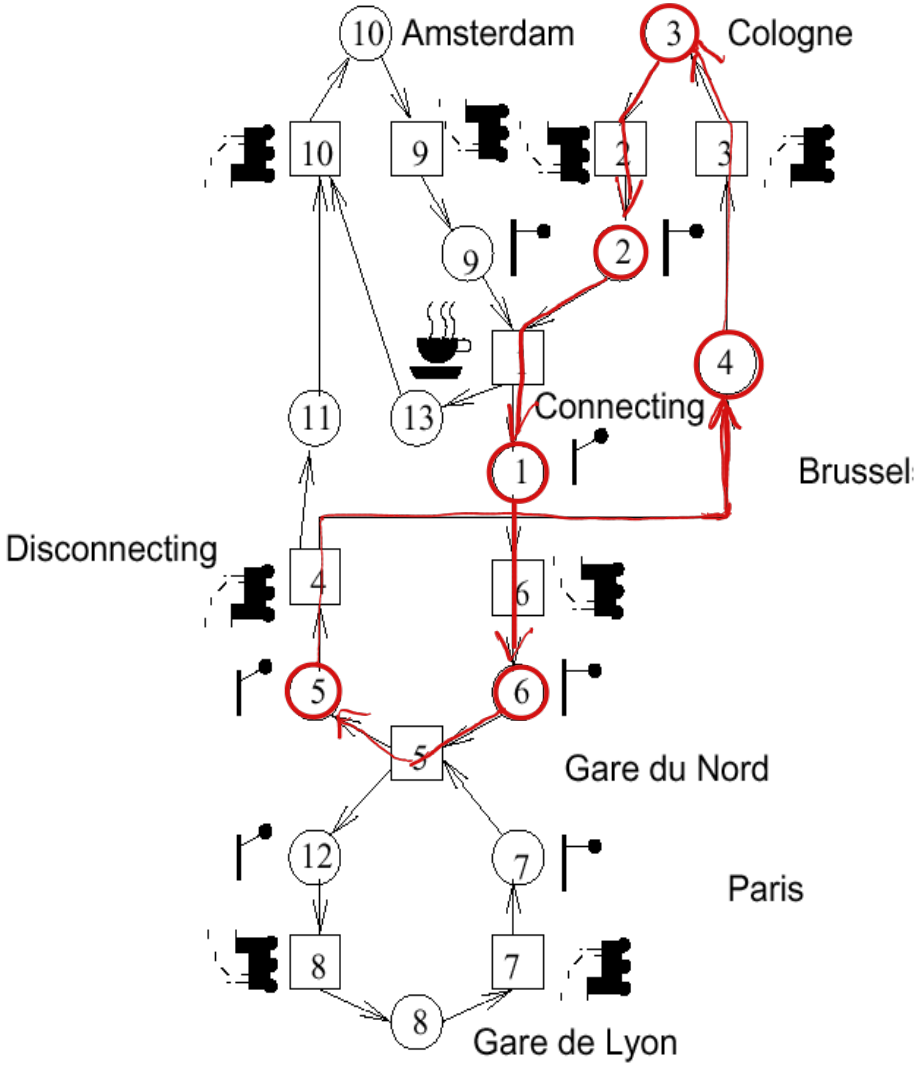
Place/Transition nets – Example (1)

- Synchronization of trains between Amsterdam, Cologne, Brussels and Paris
- Trains run independently the first part of the track and join at Brussels
- Trains will be connected at Brussels, running as one train to Paris
- On the way back they will be disconnected at Brussels and run as independent trains to Amsterdam and Cologne
- We assume a synchronization at Paris (must)
- The petri net could look as follow:



Place/Transition nets – Example (6)

Interpretation – 1st invariant



$c_{R,1}$ = (1,1,1,1,1,1,0,0,0,0,0,0,0)

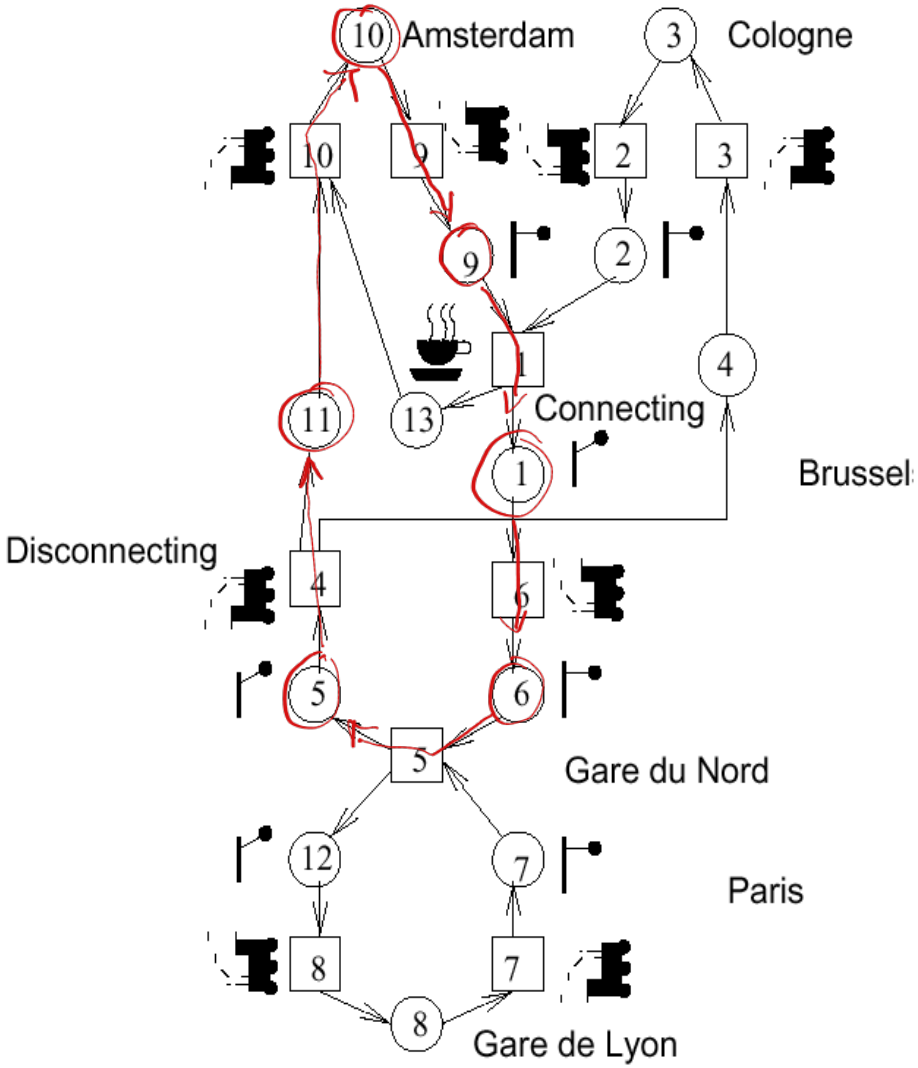
1 2 3 4 5 6 7 8 9 10 11 12 13

c_R describes places for cologne train

⟹ prooved that the number of trains is constant on that path

Place/Transition nets – Example (7)

Interpretation – 2nd invariant



$c_{R,2}$ = (1,0,0,0,1,1,0,0,1,1,1,0,0)

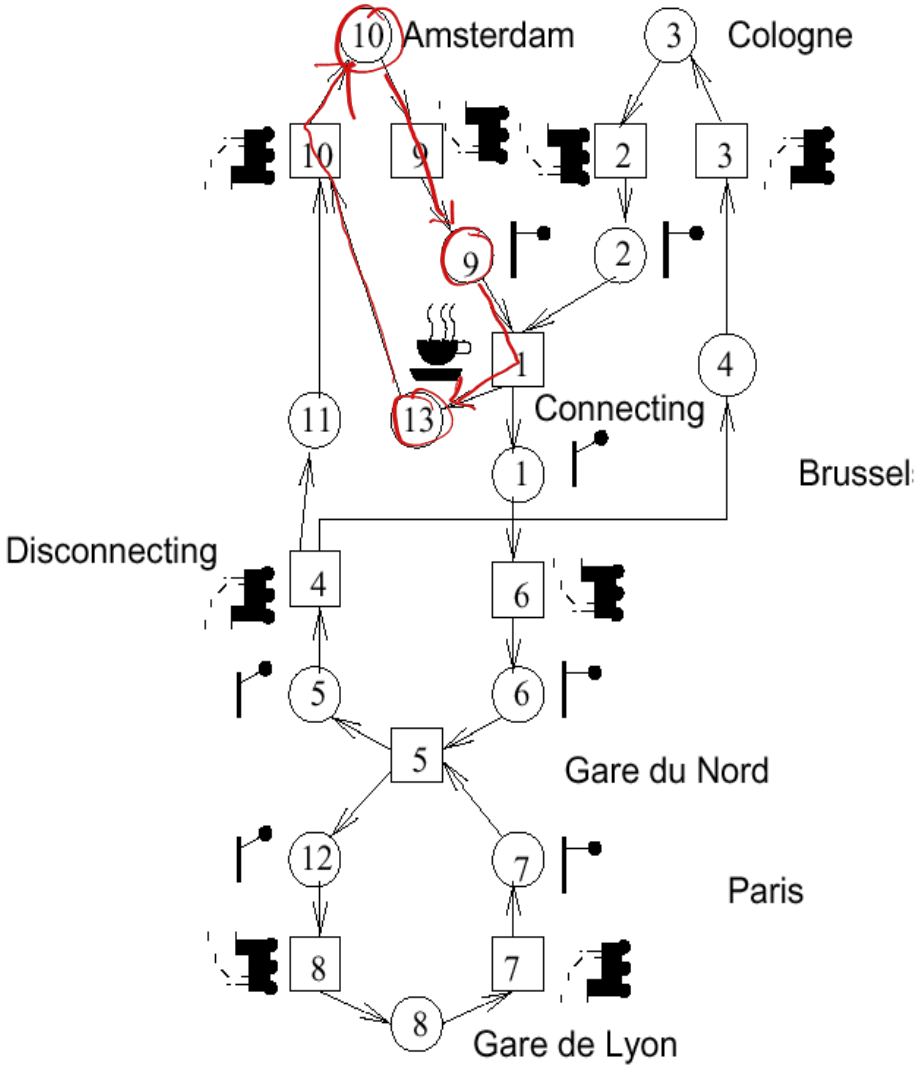
(1 2 3 4 5 6 7 8 9 10 11 12 13)

c_R describes the Amsterdam train.

→ we proved that no Amsterdam train gets lost

Place/Transition nets – Example (8)

Interpretation – 3rd invariant



$c_{R,3}$ = (0,0,0,0,0,0,0,0,1,1,0,0,1)

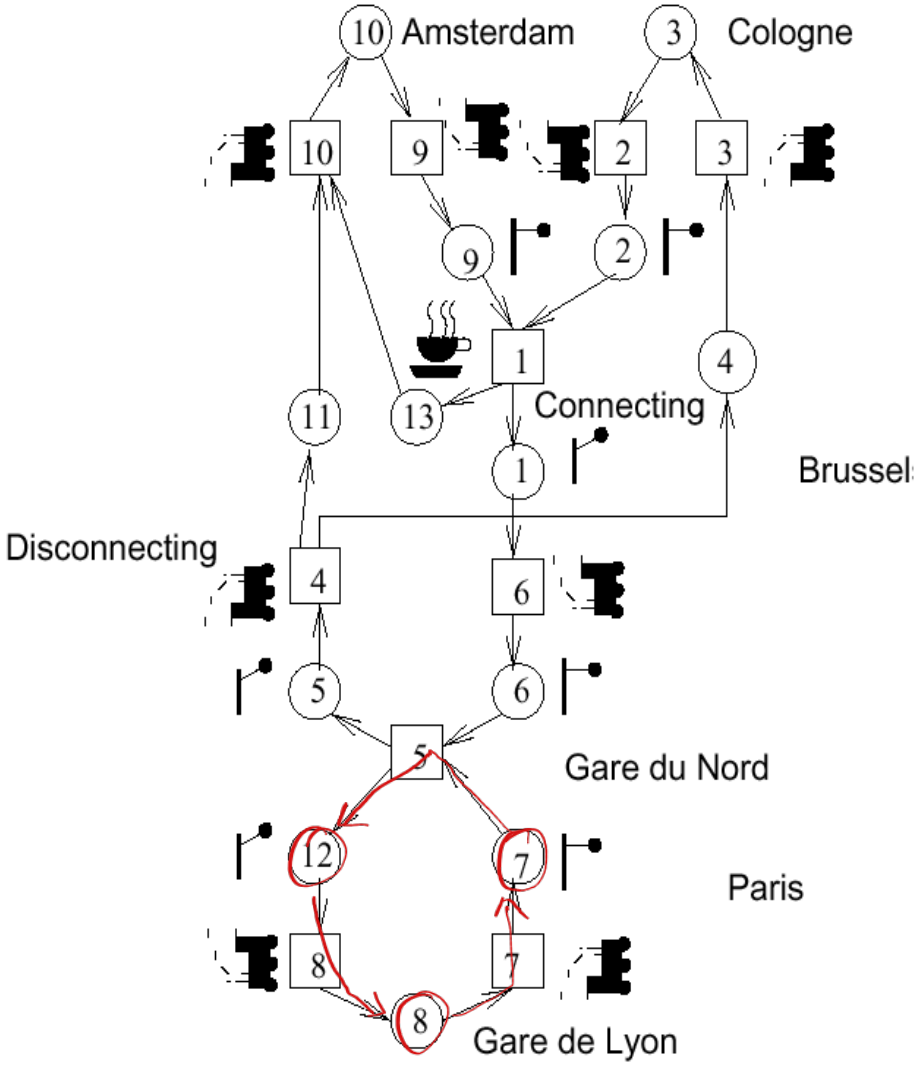
1 2 3 4 5 6 7 8 9 10 11 12 13

c_R describes the path of the train driver II

⇒ prooved that the number of train drivers remains constant.

Place/Transition nets – Example (9)

Interpretation – 4th invariant



$c_{R,4} = (0,0,0,0,0,0,1,1,0,0,0,1,0)$

Exam question

(you will be asked to describe the vectors as well)

C_R describes the paris train

⟹ The number of trains serving Amsterdam/Cologne and paris remains constant.

Predicate/transition nets (3)

Example: The dining philosophers problem (2)

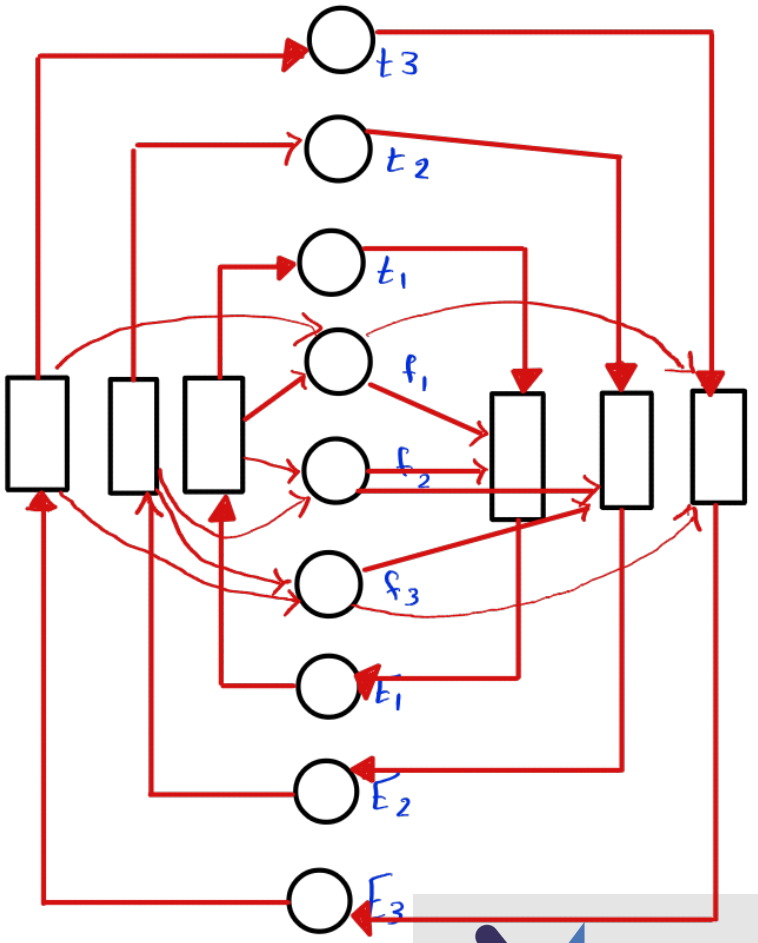
Conditions ⇒ ○ Events ⇒ ▭ Flow ⇒ →

- Condition/event net for this problem

- t_j is the thinking state of philosopher j
- e_j is the eating state of philosopher j
- f_j is the fork at place j

- This is an already quite large model of this relatively small problem

- It is complicated to expand this model to more philosophers



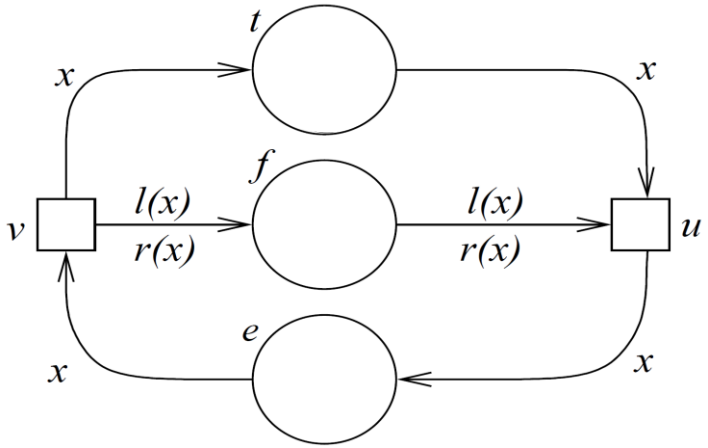
Which net is better for dining philosophers problem?

ans:

Predicate/transition nets (5)

Example: The dining philosophers problem (4)

- Two forks are required as a pre-condition for transition *u*
- These two forks are returned as a post-condition by transition *v*



- These model can easily be extended by a forth philosopher (just add an individual token)
- No change of the structure of the net is required

Petri Nets - Evaluation

Pros:

- well suited for modeling distributed systems
- strong theoretical foundation for formally proving system properties
- they become more popular due to the increasing amount of distributed systems

Cons:

- not necessarily determinate (different firing sequences can lead to different results)
- problems with modeling time
- no programming elements
- no hierarchy
- it is difficult to represent data

Extensions:

- large amount of efforts to remove some limitations

What is Von-Neumann model ?

- Reflects the principles of operation of standard computers by the sequential execution of instructions

- It also allows possible branches and an almost unrestricted access to global memory (shared variables)

- Example languages include:
  - machine (binary) languages
  - assembly languages
  - imperative languages providing limited abstraction of machine languages such as C, C++ or Java

- Follows the bottom up process

Shared memory communication (1)

- Allows several threads to access the same memory and is therefore very fast

- Might end up in potential race conditions:

```
thread 1 {
        u = 1;
        if u < 10 { u = u +1; ...}
}
```

```
thread 2 {
        …..
        u = 20;
}
```

- A context switch after the if check could change the result to u = 21

- That means an inconsistent result might be possible

- For critical sections, exclusive access to memory must be guaranteed

what is a race condition?

Shared memory communication (2)

- Critical sections:

```
thread 1 {
      u = 1;
   P(s)   //obtaining mutex
      if u < 10 { u = u +1; ...}
  V(s)   //releasing mutex

}
```

```
thread 2 {
      .....
       P(s)
      u = 20;
       V(s)

}
```

- P(S) grants up t *n* concurrent accesses to resource (here is n=1)
- V(S) increases number of allowed accesses to resource

- Imperative model should be supported by:
  - mutual exclusion for critical sections what is
  - cache coherency protocols

How to make the sys. Deadlocks & ree !

Deadlocks can happen, if the following four conditions (Coffman, 1971) are met:

- Mutual exclusion: a resource that cannot be used by >1 threads
- Hold and wait: thread already holding resources may request new resources
- No pre-emption: resources can only be released by the withholding tread
- Circular wait: ≥ 2 threads form a circular chain, each thread waits for the resource from the next thread which holds the thread
- Techniques for turning one of these conditions false degrade performance/increase resource requirements seriously

Modeling levels (2)

Levels at which modeling can be done include:

- System level
- Algorithmic level
- Instruction set level (ISA)
- Transaction level modeling (TML)
- Register transfer level (RTL)
- Gate-level models
- Processor/memory/switch level
- Layout level

Know at least 5 (in any order)

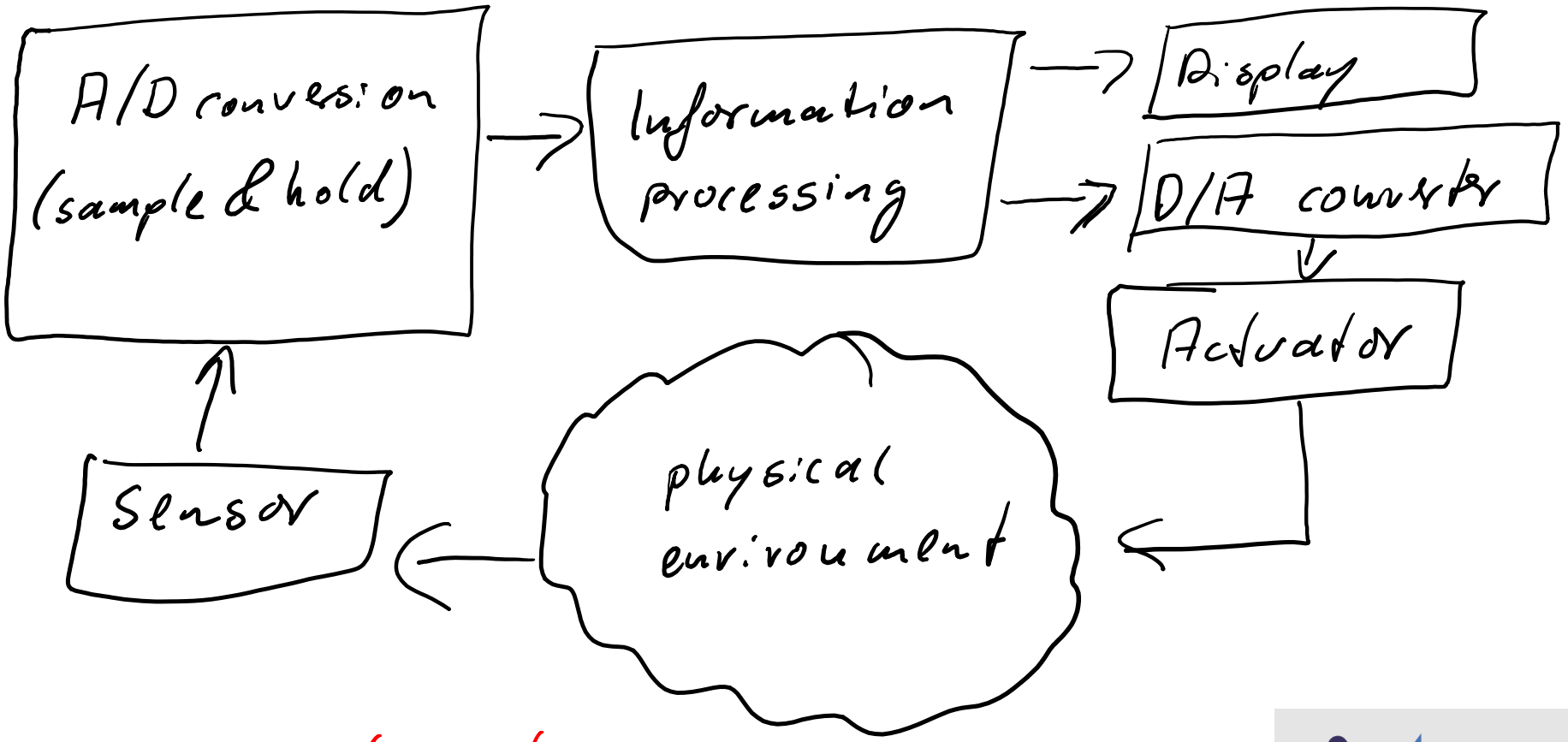
Gaits

Gate-level, Algorithmic level, Instruction set level, Transaction level, System level

ES Hardware (3)

- Embedded systems HW is often used as “hardware in a loop”
- Example:

sketch HW in a loop circuit

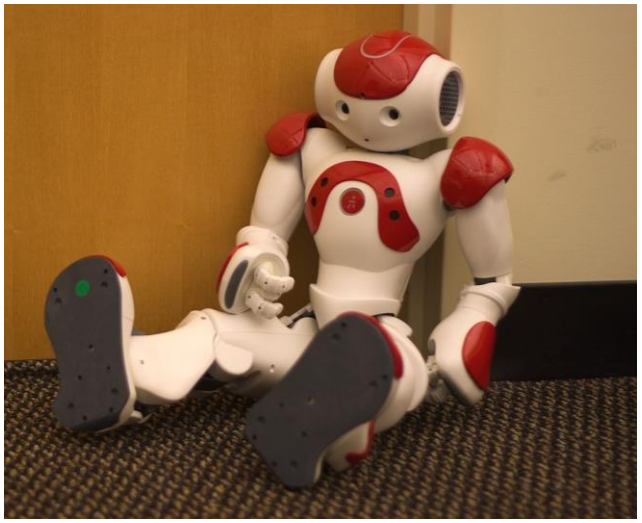


cyber physical system

nothing_men.

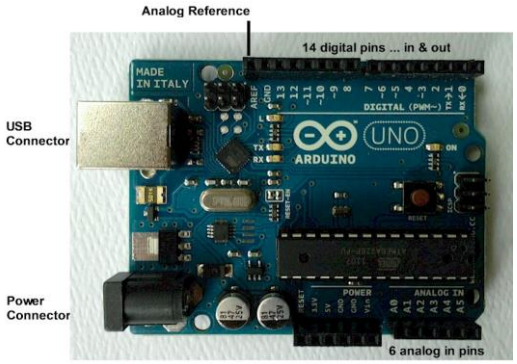
ES Hardware (4)

- There are many examples of such loops:
  - heating
  - lights
  - engine control
  - power supply
  - ….
  - robots



Source: https://de.wikipedia.org/wiki/Nao_(Roboter)#/media/File:Nao_humanoid_robot.jpg, Author: Jiuguang Wang, Retrieved: 20/09/2016, Licence: CC BY-SA 3.0

Source: https://de.wikipedia.org/wiki/Thermostatventil#/media/File:Honeywell_Rondostat_HR20_resized.jpg, Author: Phrontis, Retrieved: 20/09/2016, Licence: CC BY-SA 3.0



Source: https://en.wikipedia.org/wiki/Arduino#/media/File:UnoConnections.jpg, Author: 1sfoerster, Retrieved: 20/09/2016, Licence: CC BY-SA 3.0

Sensors (1)

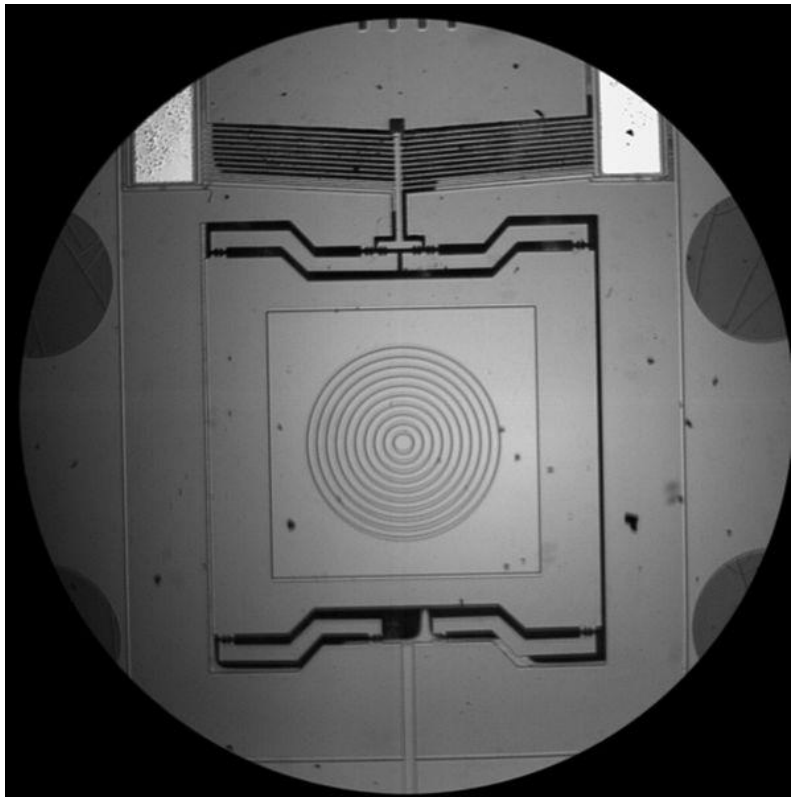
- Capture physical/chemical quantity and convert it to electrical quantity
- Captured quantities include:
  - temperature
  - light
  - distance
  - acceleration
  - velocity
  - magnetic fields
  - weight
  - chemical compounds

mention 5 diff sensors

- Many physical effects are applied in sensor production, e.g.:
  - law of induction (generation of voltages in a magnetic field)
  - hall effect (see https://en.wikipedia.org/wiki/Hall_effect for more details)

Sensors (2)

- Huge amount of sensors developed in recent years
- Example: MEMS accelerometer



Source: https://commons.wikimedia.org/wiki/File:MEMS_(594047927 7).jpg, Author: MEMS, Retrieved: 14/12/2016, Licence: United States Government Framework

- Other sensors:
  - rain sensor sketch
  - rotation sensor
  - biometric sensor
  - artificial eyes
  - radio frequency identification – RFID
  - pressure sensor
  - proximity sensor
  - image sensor – charged-coupled device (CCD)

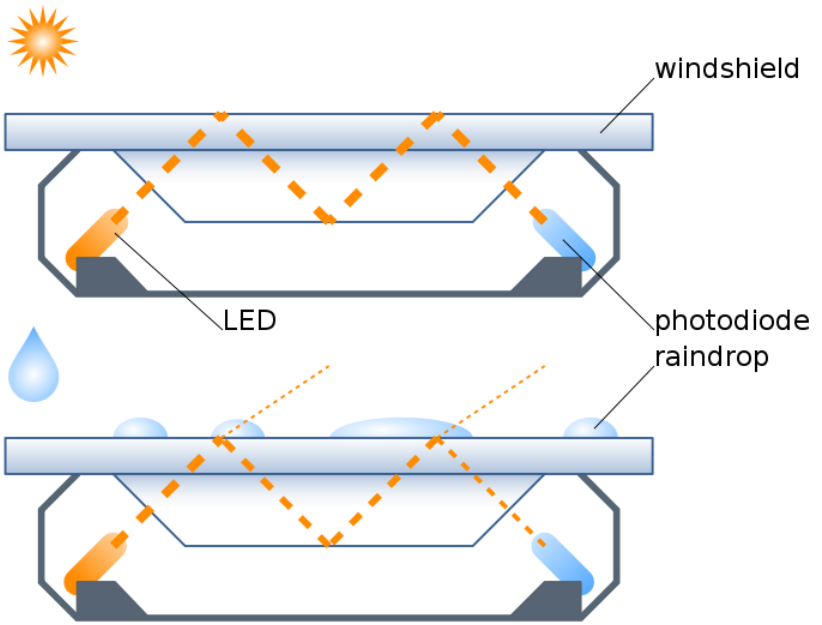
Sensors (3)

Rain sensor

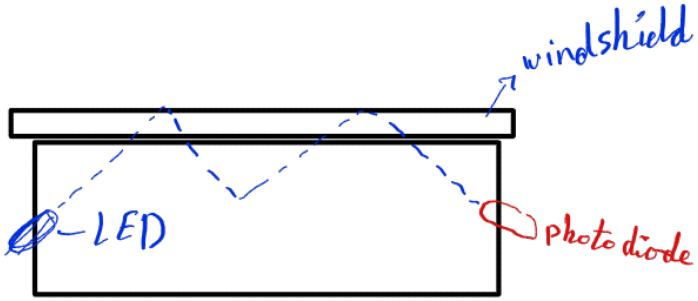
sketch

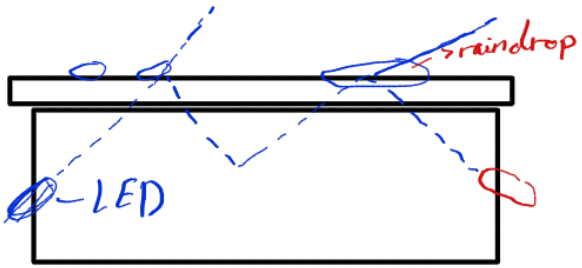
- Based on internal reflection

- IR light is projected at a 45 degree angle into the windshield
- Nearly all the light makes its way to the sensor if the glass is dry
- If the glass is wet, less light finds its way to the sensor, which then generates a different signal



Source: https://en.wikipedia.org/wiki/File:Rain_sensor_en.svg (14/09/2015)





Signals (1)

what is a signal?
how to megar?
sensors
Can we work with a signal continous in time?

- Sensors generate signals from the measured quantities

- **Definition**:

  A signal *s* is a mapping from the time domain D_T to a value domain D_V:

  $$s: D_T \rightarrow D_V$$

  D_T : continuous or discrete time domain

  D_V : continuous or discrete value domain

- Digital computers usually work in a discrete time domain, which means that they can process discrete sequences/streams of values

  → continuous time domain signals must be converted to discrete time domain signals

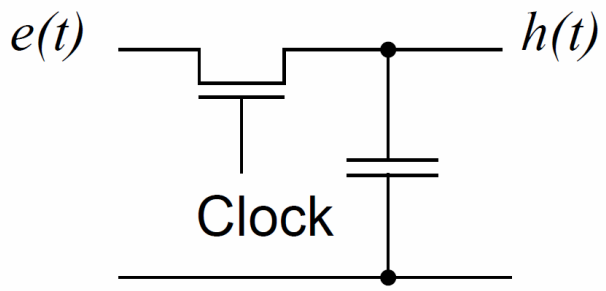
- Sample-and-hold circuits are designed to do this job

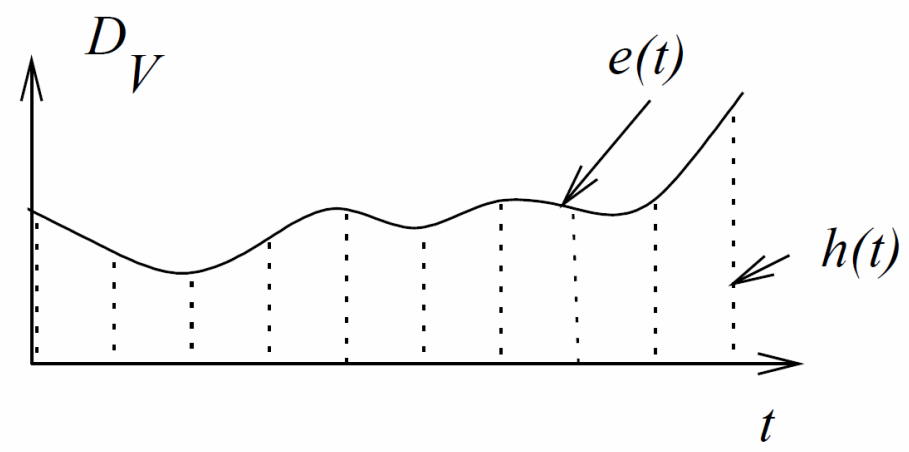
What do we use to go from Analog to digital signal?

Signals (2)

Discretization of time

- Clocked transistor and capacitor
- Transistor operates as a switch, the capacitor is charged up to the *e(t)* if the transistor is closed [so *h(t)=e(t)*]
- *h(t)* remains the same when the transistor is opened
- Each value stored on the capacitor can be considered as one discrete value
- The discrete values will be only defined at the clock times

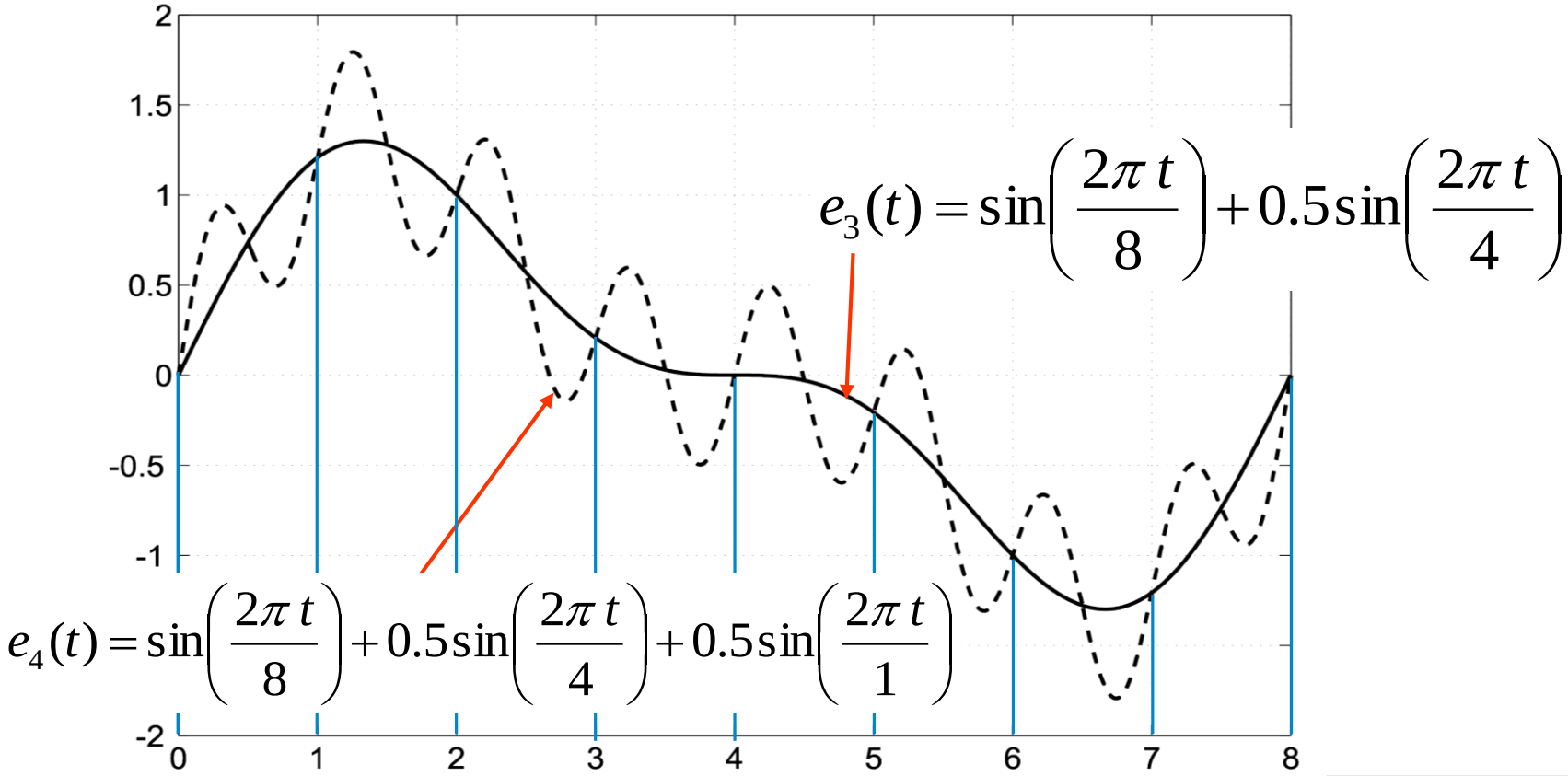




Why do we have Aliasing (1)? when we don't sample enough

- Are we able to reconstruct the original signal?

we want to reconstruct the signal

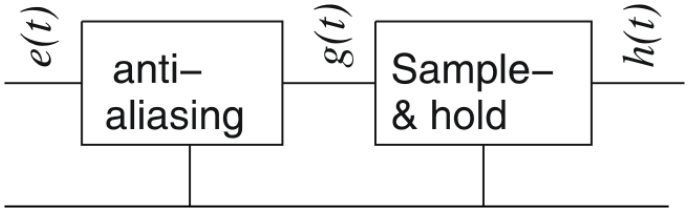


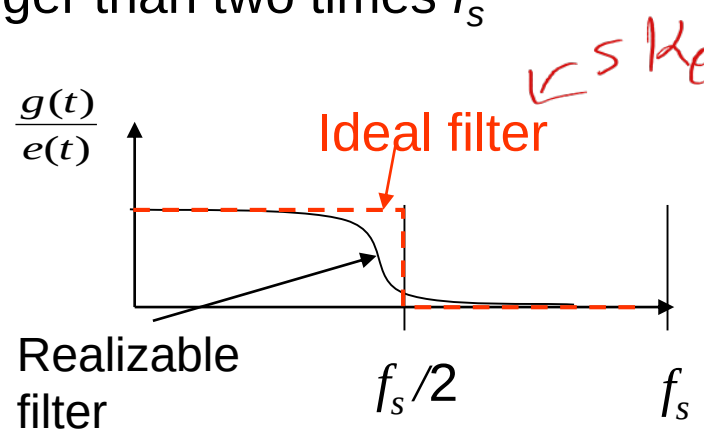
What can we do to prevent

Aliasing (3)

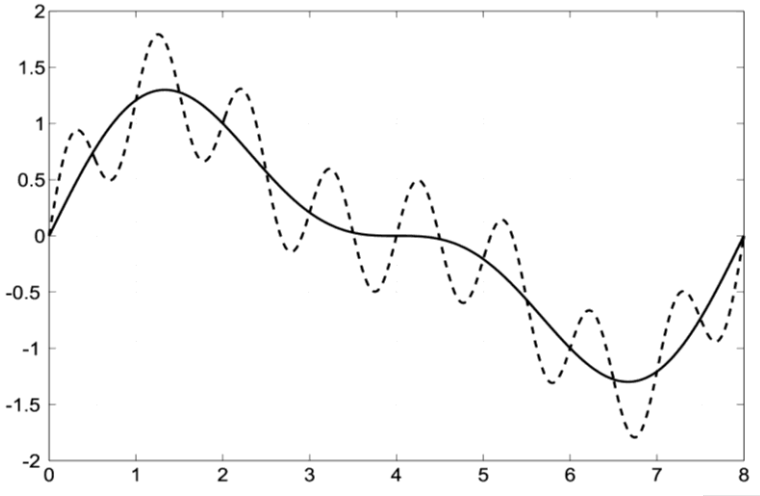
Anti-Aliasing filter

- To remove aliasing we can place a low-pass filter in front of the signal which removes all frequency components larger than two times f_s





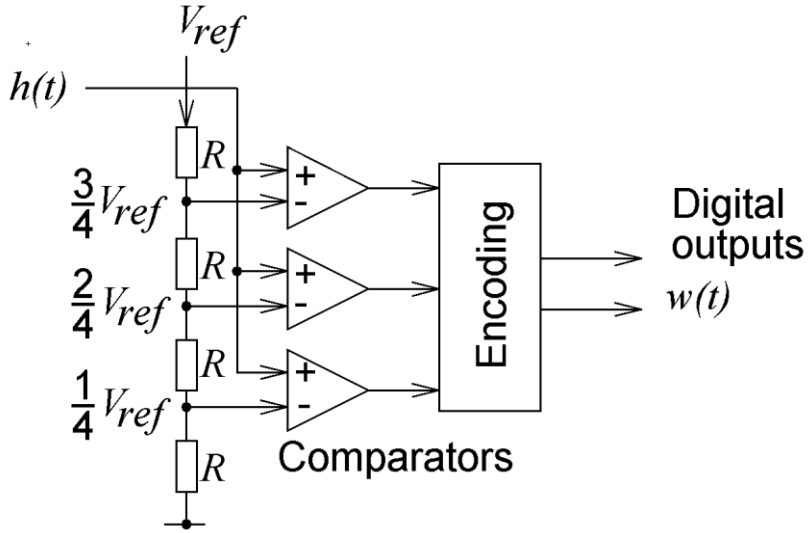
- A low-pass filter at f_s /2 will remove $e_4(t)$

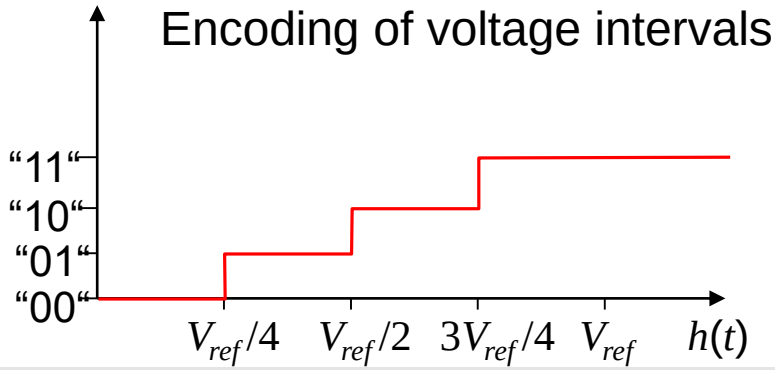


A/D converter (1)

Example: Flash A/D converter

- Utilises comparators to convert the values
- Each comparator has two inputs (+ & -)
- When the voltage on input + exceeds the voltage on input -, the output corresponds to a logical **1**, otherwise to a logical **0**
- All inputs are connected to a voltage divider
- For *h(t)* = 80% of V_{ref} all three comparators will be logical **1**
- the encoder will encode this into a digital signal

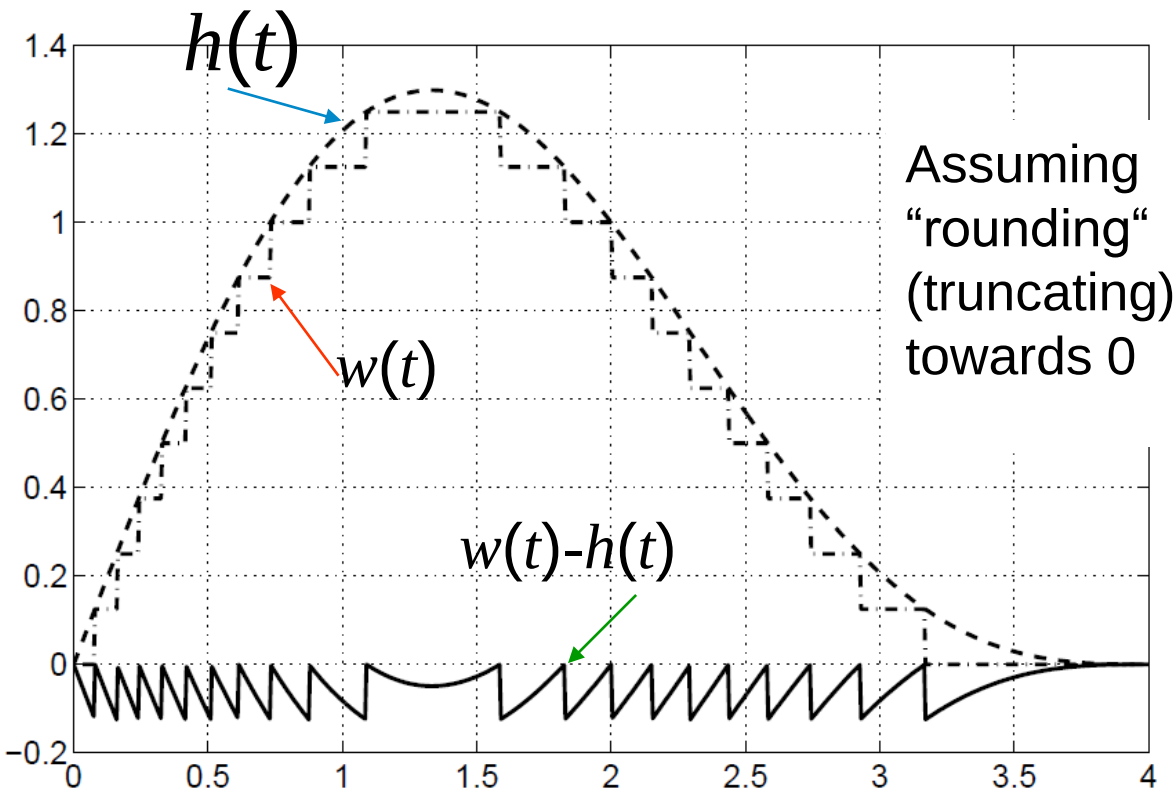




What is

Quantization noise

- The Figure shows the original voltage *h(t)*,
- the voltage corresponding to the digital value *w(t)* and
- the difference between the two *w(t)-h(t)*
- This difference is called: *quantization noise(t)*
- It is possible to decrease the quantization noise by increasing the resolution (bits)



- The impact of the quantization noise is captured in the definition of the signal-to-noise ratio (SNR)

Signal to Noise Ratio (SNR)

- The SNR can be calculated by: Could ask to calculate !

$$SNR(dB) = 10 \cdot \log \frac{P_S}{P_N} \quad \text{or} \quad SNR(dB) = 20 \cdot \log \frac{V_S}{V_N} \quad \text{where}$$

remember

P_S : is the power of the signal

P_N : is the power of the noise

V_S : is the voltage of the signal

V_N : is the voltage of the noise

- Recap: the power of a signal is equal to the square of the voltage for any given R

- Example: SNR of a 16-bit audio CD is in the order of 96 dB

Embedded systems HW