

# Lecture slides course Execution Architecture

material by *Gerrit Muller*

presented by *Ton Kosteljik, Gerrit Muller*

PDSL Eindhoven, Embedded Systems Institute

## **Abstract**

The course Execution Architecture is a joint effort of Ton Kosteljik and Gerrit Muller. Only limited theory is given, most time is spent hands-on. Not all the material is in this bundle, the material made by Ton Kosteljik is in a separate handout.

## **Distribution**

This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

February 11, 2012

status: planned

version: 0

logo  
TBD

# Module Information of the course Execution Architecture

by *Gerrit Muller* Embedded Systems Institute

e-mail: `gerrit.muller@embeddedsystems.nl`

`www.gaudisite.nl`

## Abstract

The course execution architecture is a joint effort of Ton Kostelijck and Gerrit Muller. The intention of the course is to help the participants in the practical aspects of designing an execution architecture. Most time during the course is spent in the normal development environment in exploring, measuring and modifying the current design. In the course setting the results are evaluated and next steps are planned. The amount of theory in the course itself is very limited, plenty of theoretical courses exist already.

## Distribution

This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

February 11, 2012

status: planned

version: 0



# Course Execution Architecture

by *Gerrit Muller* Embedded Systems Institute  
e-mail: `gerrit.muller@embeddedsystems.nl`  
`www.gaudisite.nl`

## Abstract

The course Execution Architecture (EA) is described. The program existing of 2 modules and 3 feedback and plan sessions is described. The course format, based mostly on hands on work in real products being created, is explained.

The course execution architecture is a joint effort of Ton Kostelijk and Gerrit Muller.

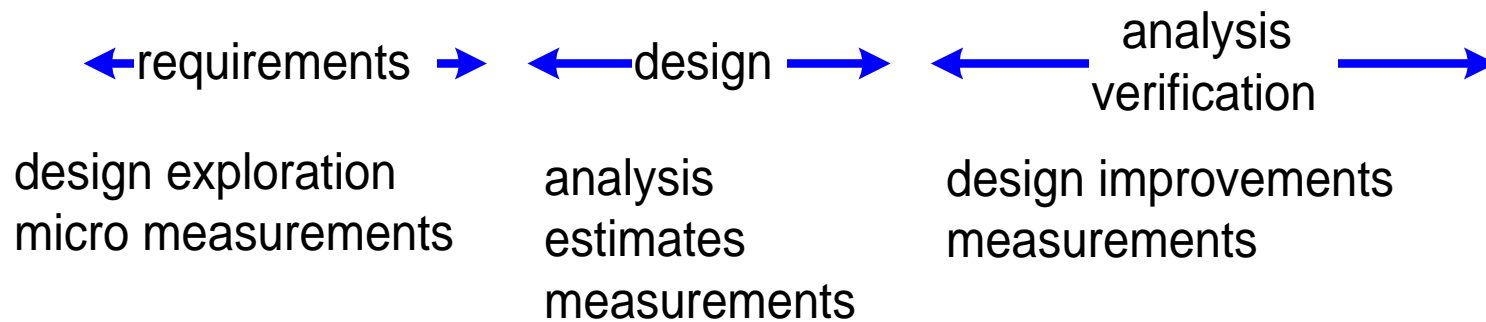
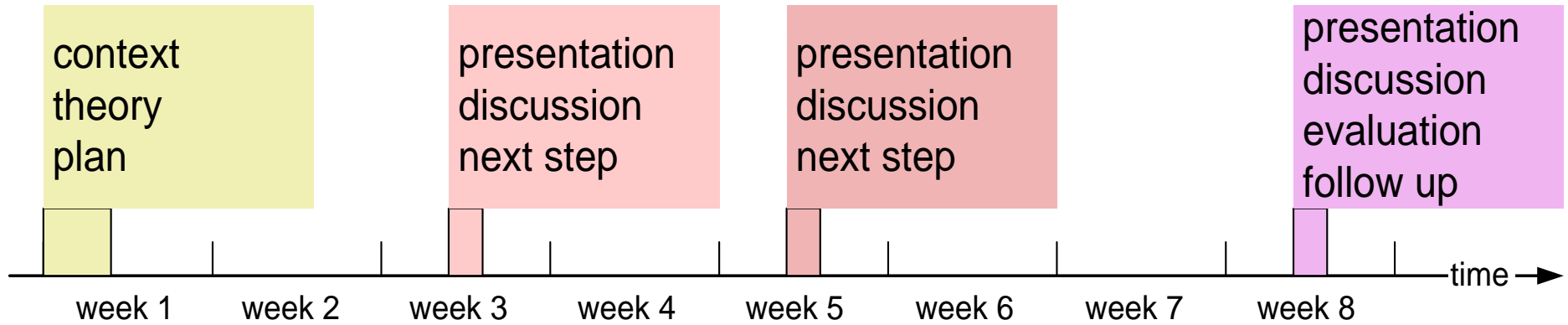
## Distribution

This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

February 11, 2012  
status: planned  
version: 0



# Course Program



# Rules of the Broadcast Part

---

- Please write your questions/remarks/statements on yellow stickers and attach them at the end on the P-flip.

*These will be used in the interactive section for discussion and to increase insight.*

- Short clarification questions are welcome,

*Discussion will take place in the interactive part.*

- Stupid questions don't exist. Learning is based on **safe** and **open** interaction.

*Very individual-oriented questions can be referred to a break or after the session.*

# Rules of the Interactive and the Practice Part

---

- Your contribution is essential.
- Don't monopolize the time. Everyone, also the quiet people, should have the opportunity to contribute.

*The facilitator will intervene if the contribution is limited to a small group of participants.*

- Respect the contribution of others.

*Opinions can't be wrong, difference of opinion is normal and called plurality.*

- The course format is highly experimental and based on improvisation, constructive proposals are welcome.

*It is your course! Regular evaluations will give the opportunity to influence the rest of the course.*

# Evaluation of the Expectations

---

Please write your name and expectations with a marker on one A4 page.

Describe your expectations as one-liner or in a few keywords.

These pages will be displayed on the wall of the room.

At the end of the course we will look back on these expectations, with the purpose of two-way learning.

# Module Execution Architecture approach and concepts

by *Gerrit Muller* Embedded Systems Institute  
e-mail: `gerrit.muller@embeddedsystems.nl`  
`www.gaudisite.nl`

## **Abstract**

The module Execution architecture approach and concepts addresses an incremental approach to design an execution architecture. A set of concepts is introduced and illustrated, which is useful in the hands on phase of the course.

### **Distribution**

This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

February 11, 2012  
status: planned  
version: 0

logo  
TBD

# An incremental execution architecture design approach

by *Gerrit Muller* Embedded Systems Institute  
e-mail: `gerrit.muller@embeddedsystems.nl`  
`www.gaudisite.nl`

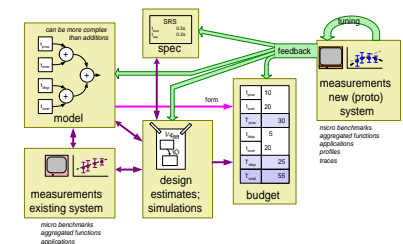
## Abstract

An incremental design approach for the execution architecture is described. The method is based on identification of the most critical requirement from both user as well as technical point of view. The implementation itself is based on quantified budgets. The creation, modification and verification of the budget is discussed.

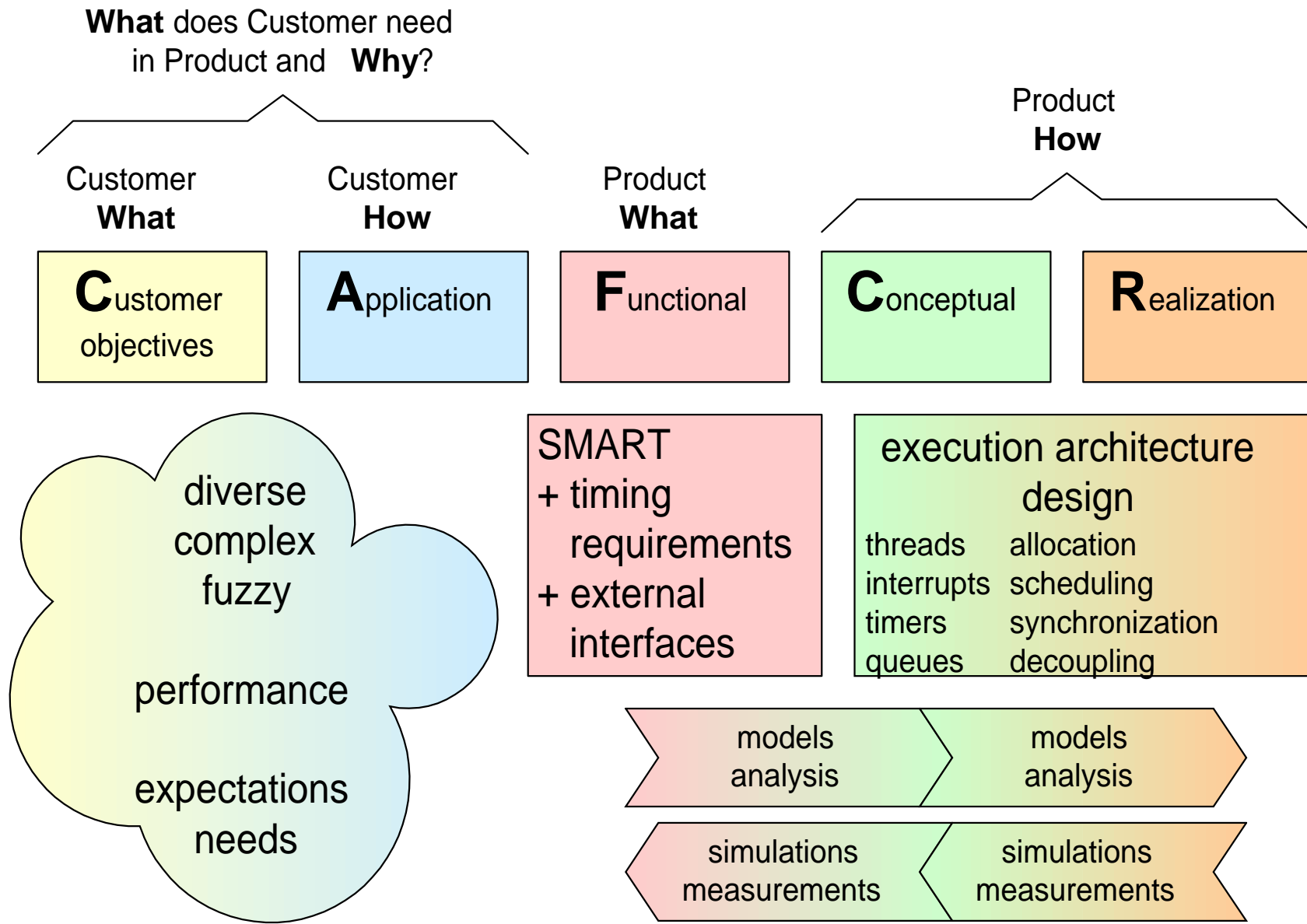
## Distribution

This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

February 11, 2012  
status: draft  
version: 1.0



# Positioning in CAFCR

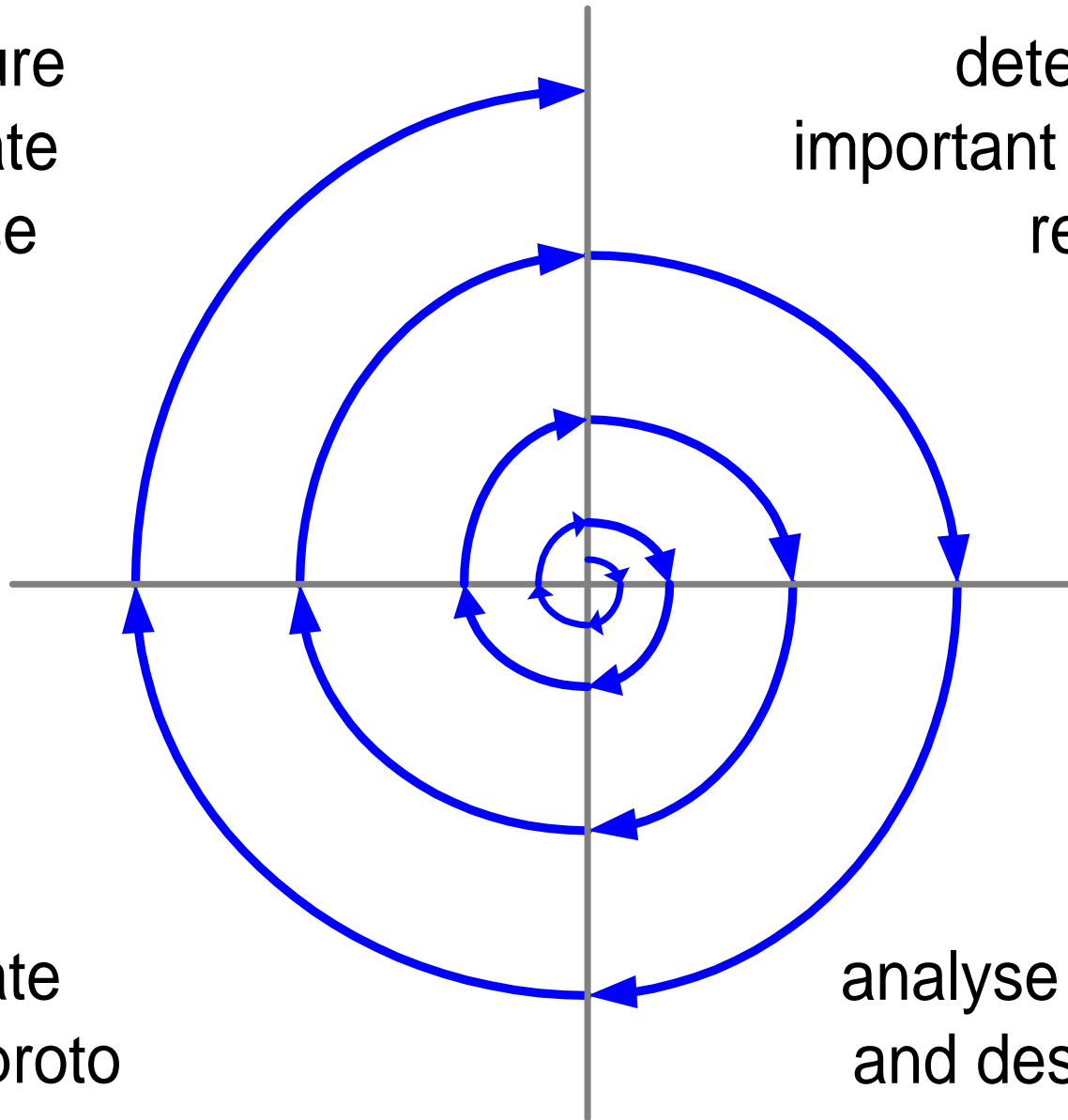


# Incremental approach

---

measure  
evaluate  
analyse

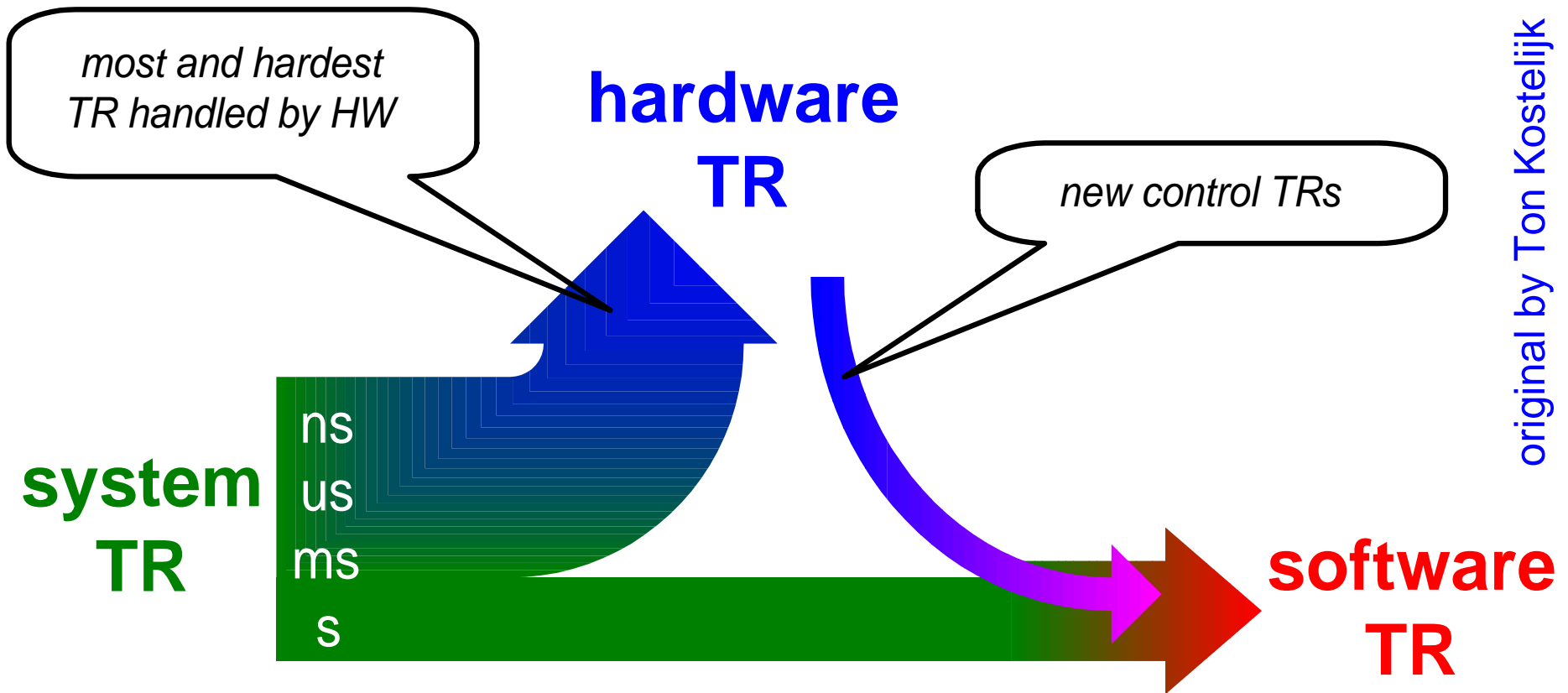
determine most  
important and critical  
requirements



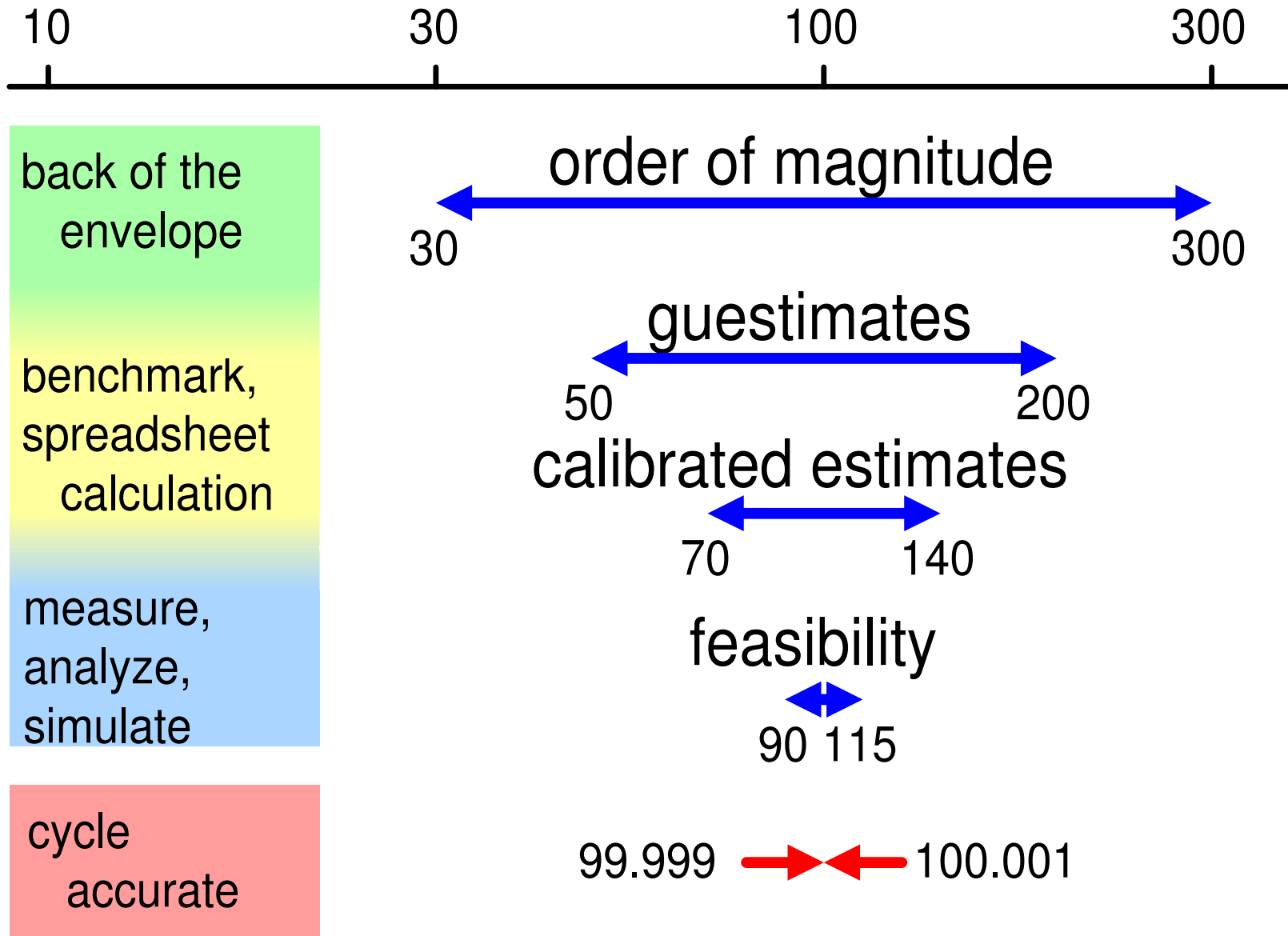
simulate  
build proto

model  
analyse constraints  
and design options

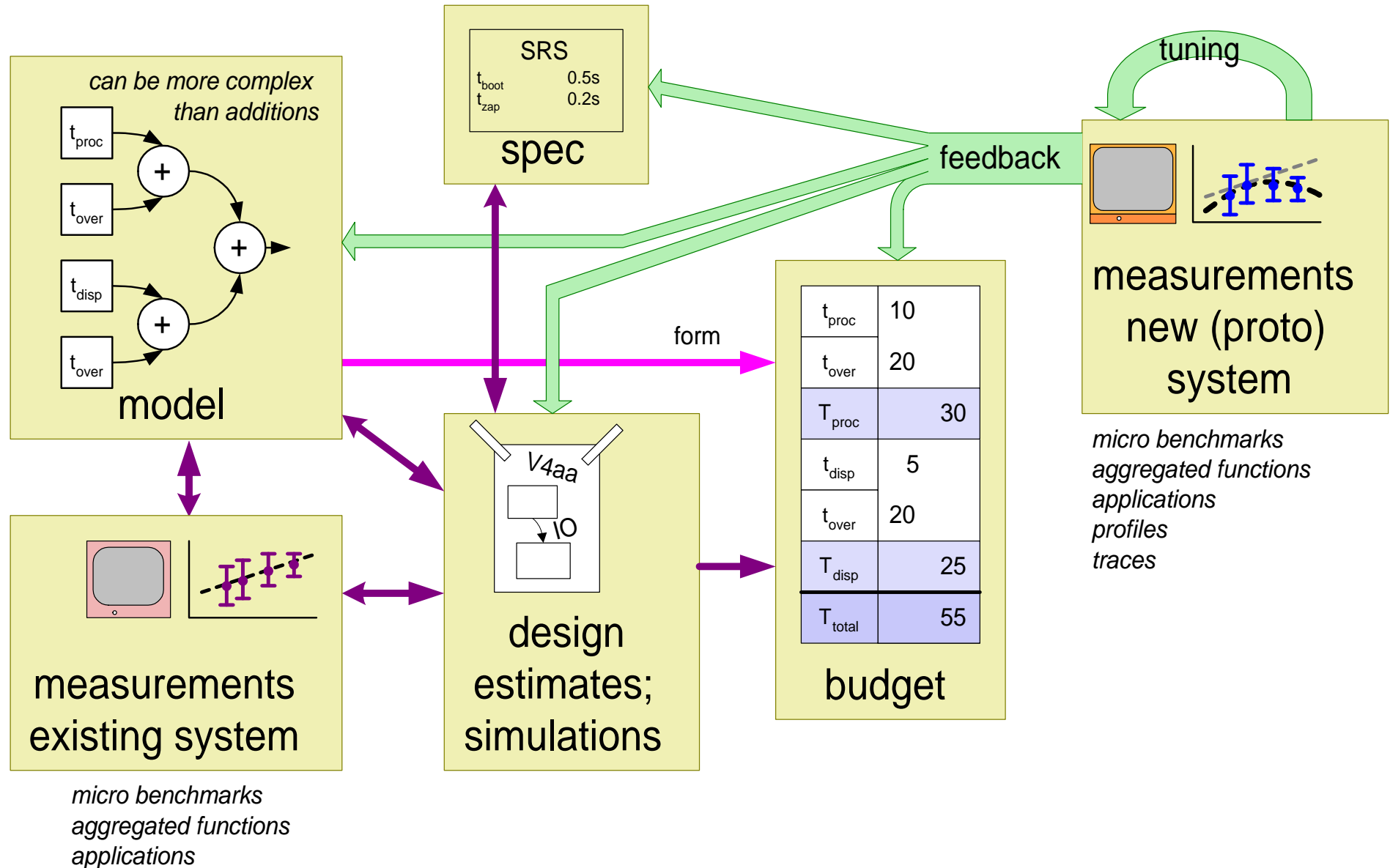
# Decomposition of system TR in HW and SW



# Quantification steps



# Budget based design



# Execution architecture concepts

by *Gerrit Muller* Embedded Systems Institute

e-mail: `gerrit.muller@embeddedsystems.nl`

`www.gaudisite.nl`

## Abstract

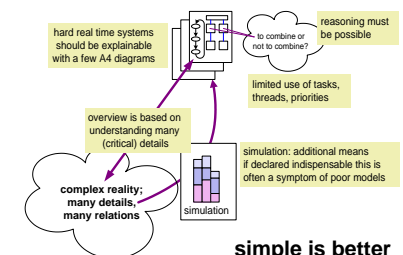
The execution architecture determines largely the realtime and performance behavior of a system. Hard real time is characterized as "missing a deadline" will result in system failure, while soft real time will result "only" in dissatisfaction.

An incremental design approach is described. Concepts such as latency, response time and throughput are illustrated. Design considerations and recommendations are given such as separation of concerns, understandability and granularity. The use of budgets for design and feedback is discussed.

## Distribution

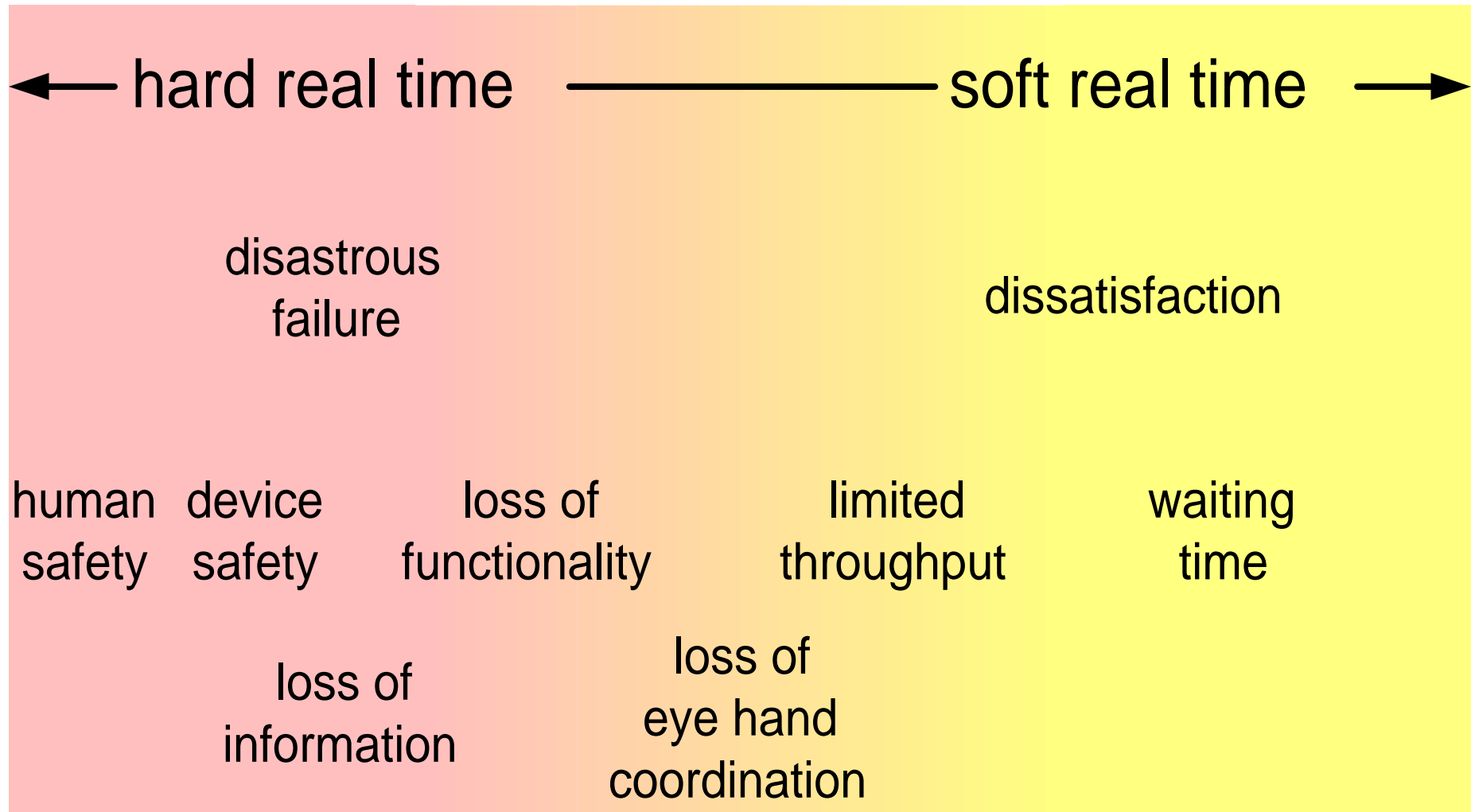
This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

February 11, 2012  
status: preliminary  
draft  
version: 1.1



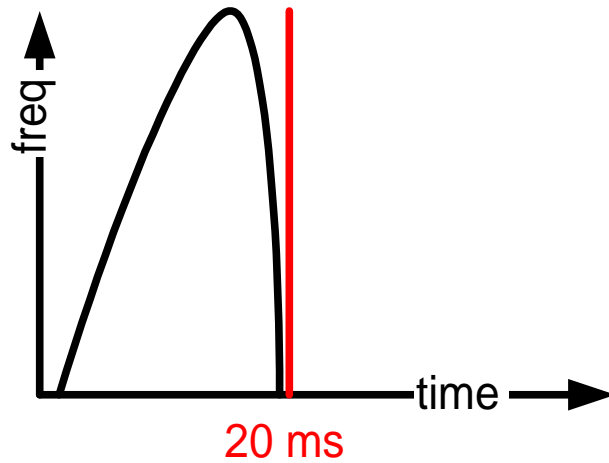


# Fuzzy customer view on real time



# Smartening requirements

Limited set of hard real time cases

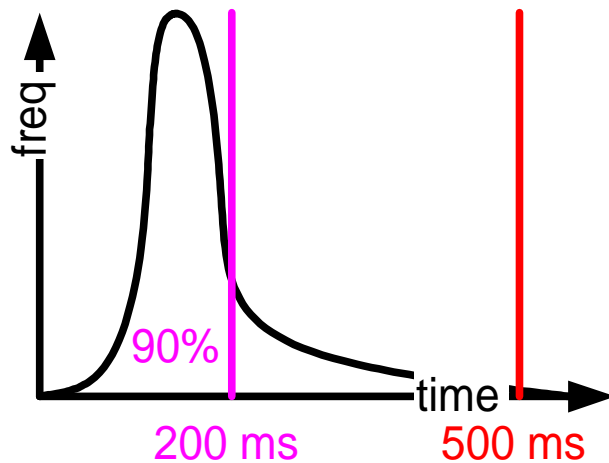


Precise form of the distribution is not important.

Be aware of systematic effects

No exception allowed  
Worst case must fit

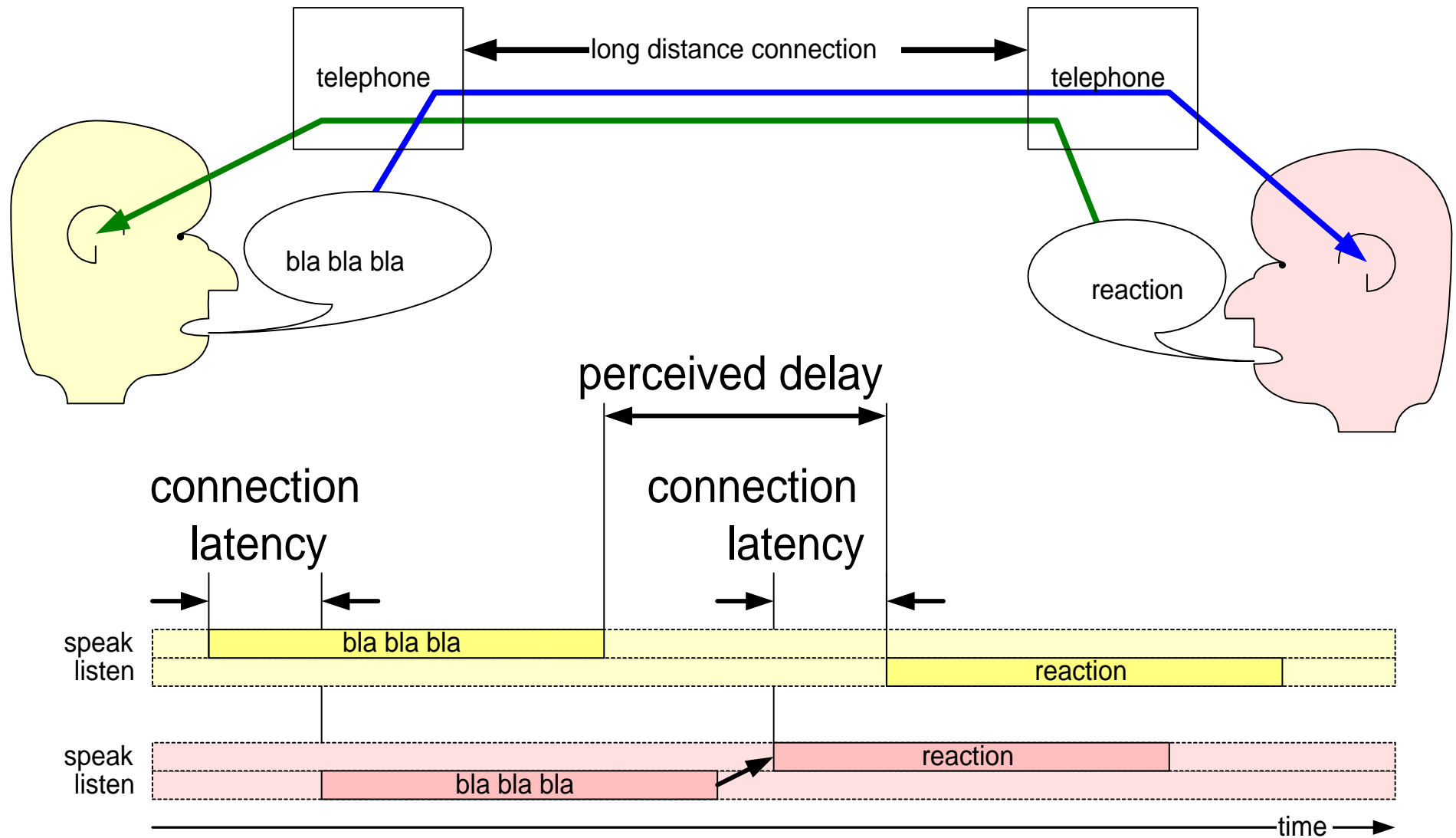
Well defined set of performance critical cases



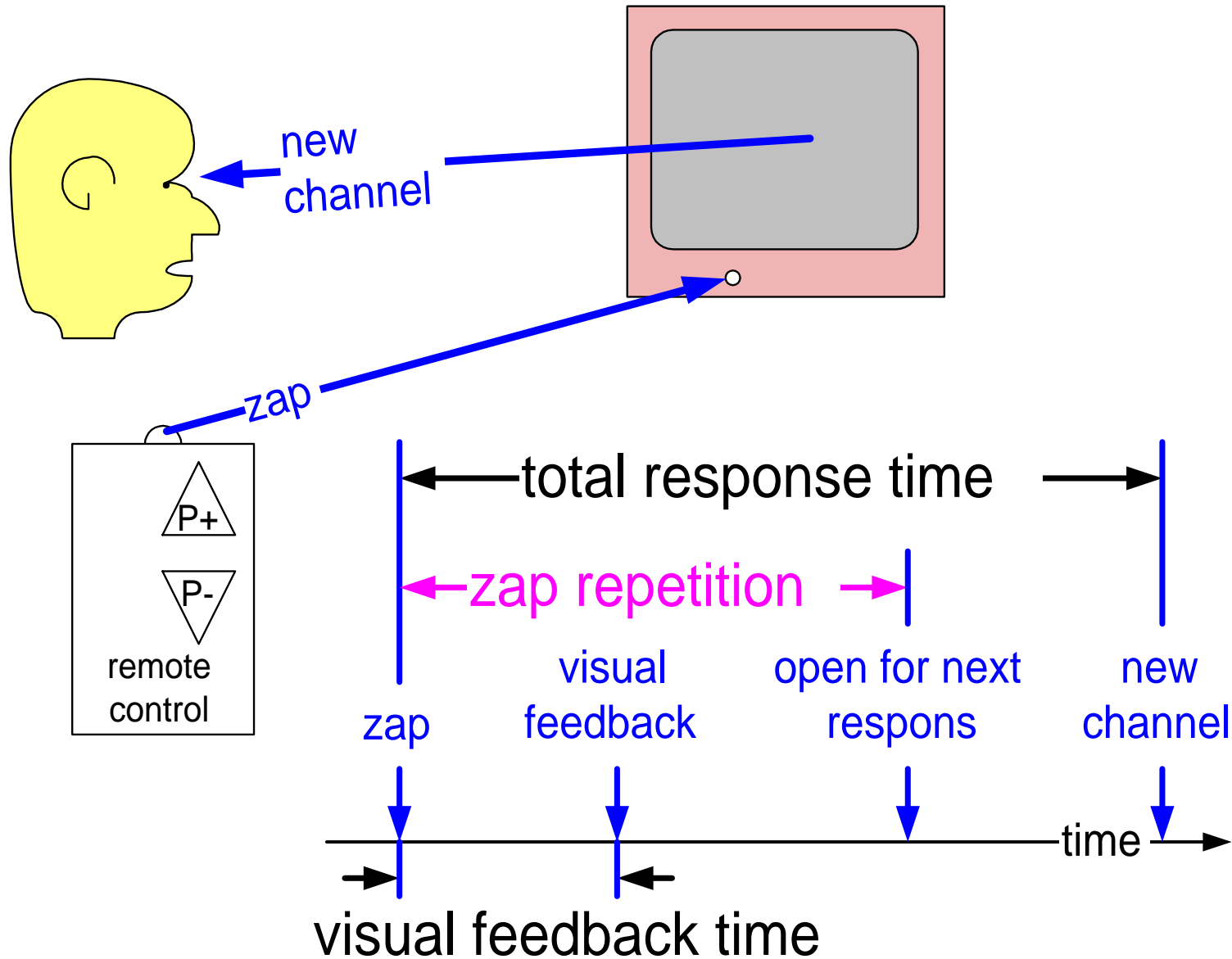
Typical within desired time,  
limited exceptions allowed.

Exceptions may not result in  
functional failure

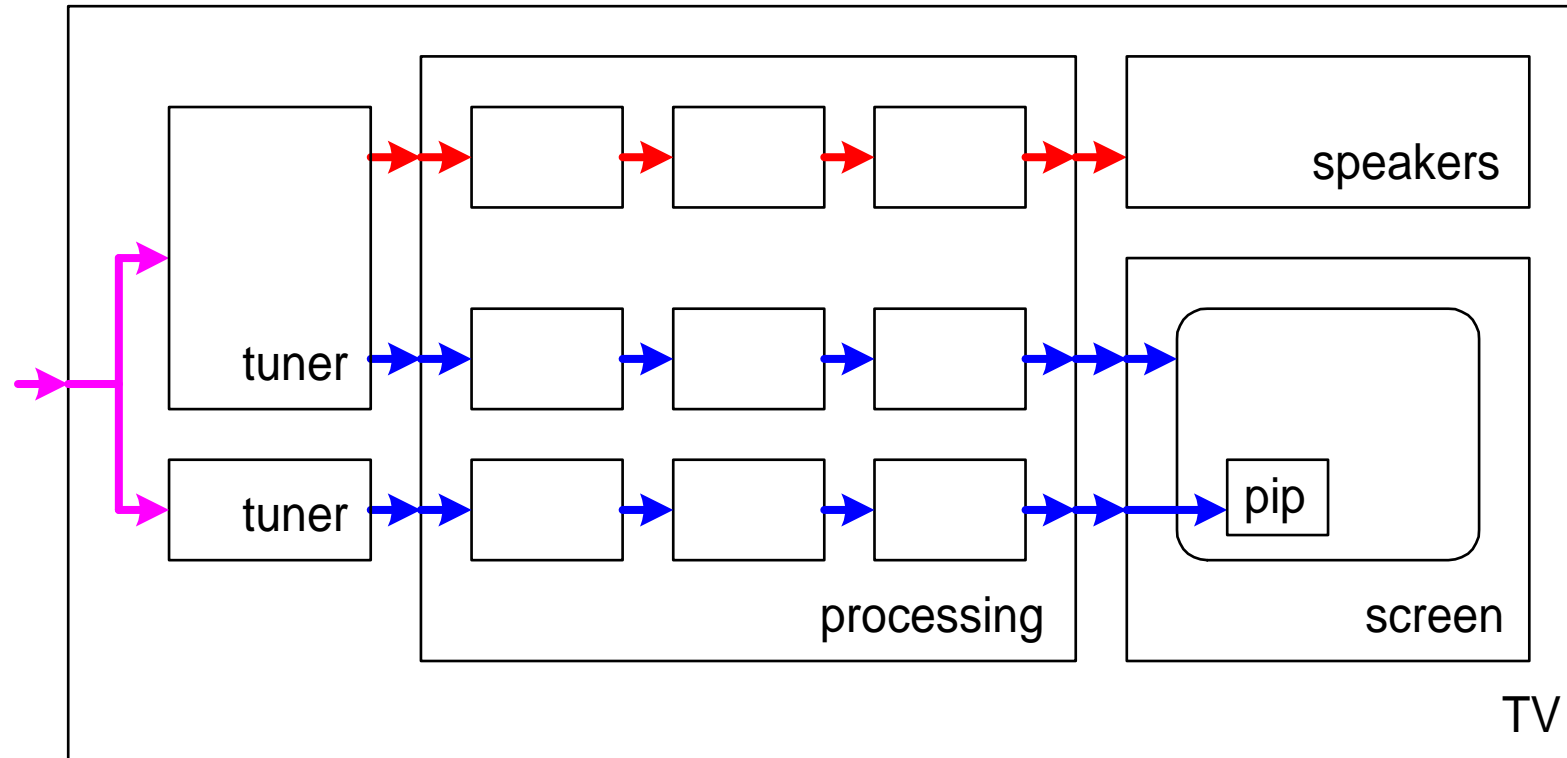
# Latency



# Response Time



# Throughput

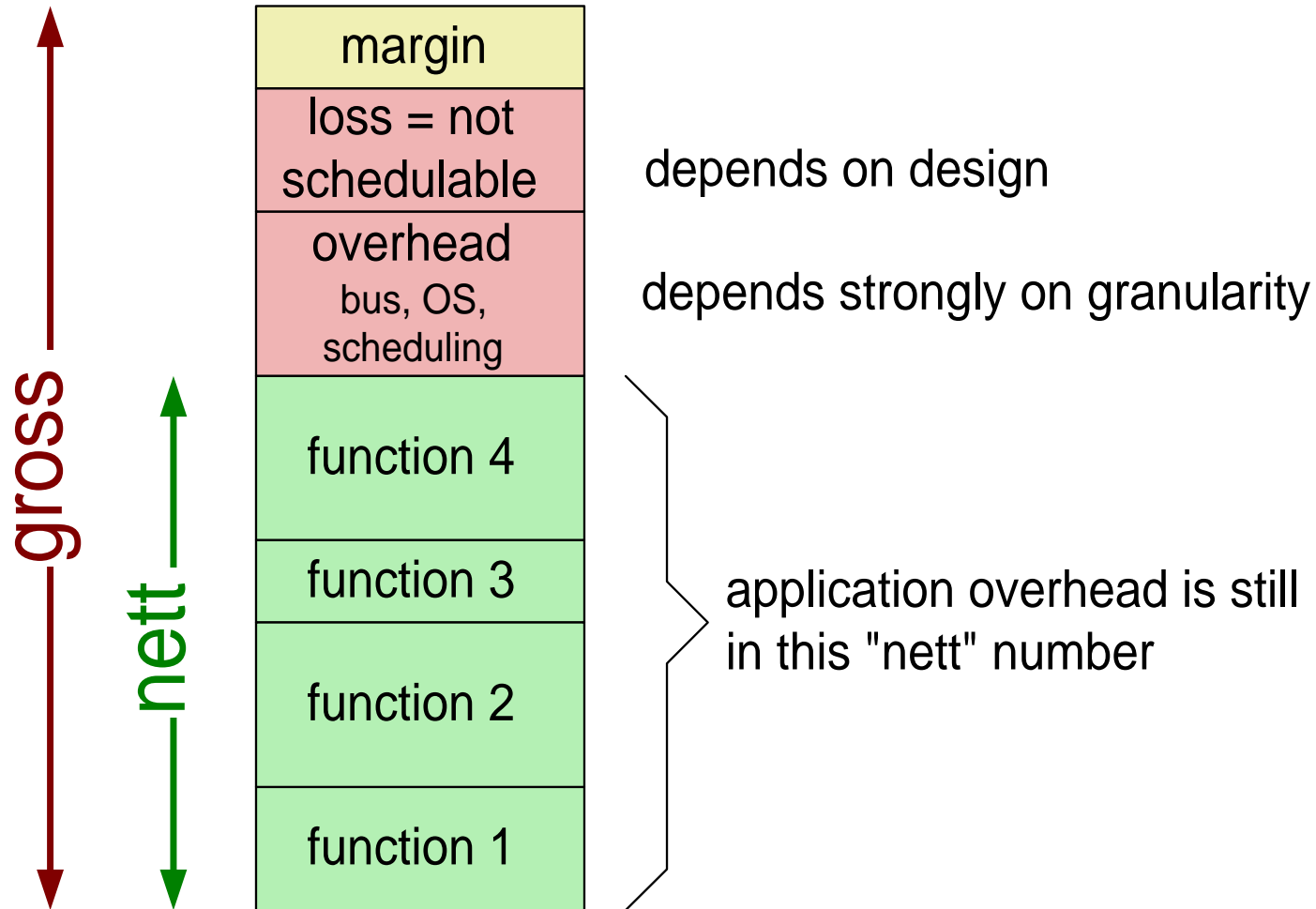


throughput:

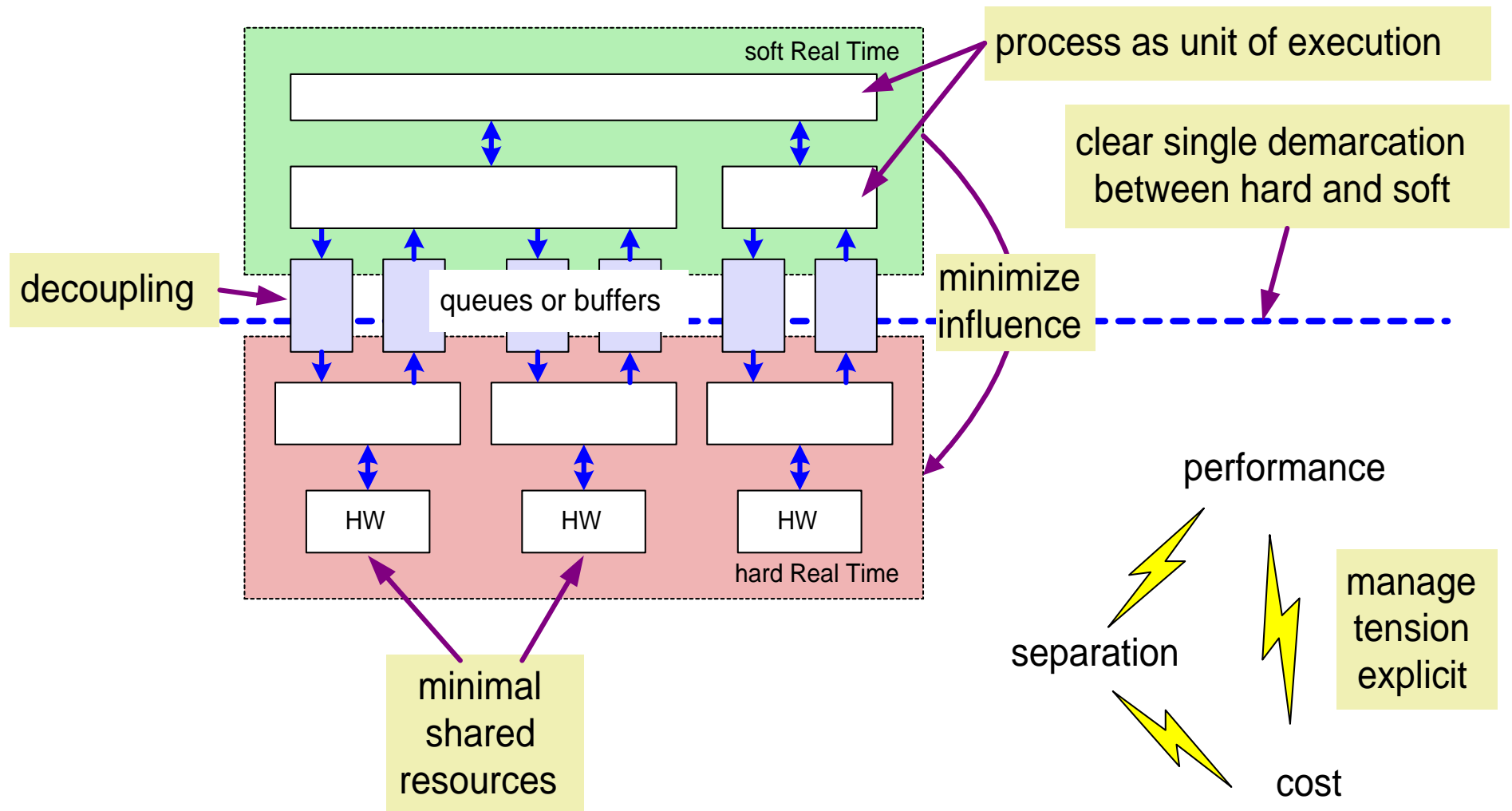
- + processing steps/frame
- + frames/second
- + concurrent streams

# Gross versus Nett

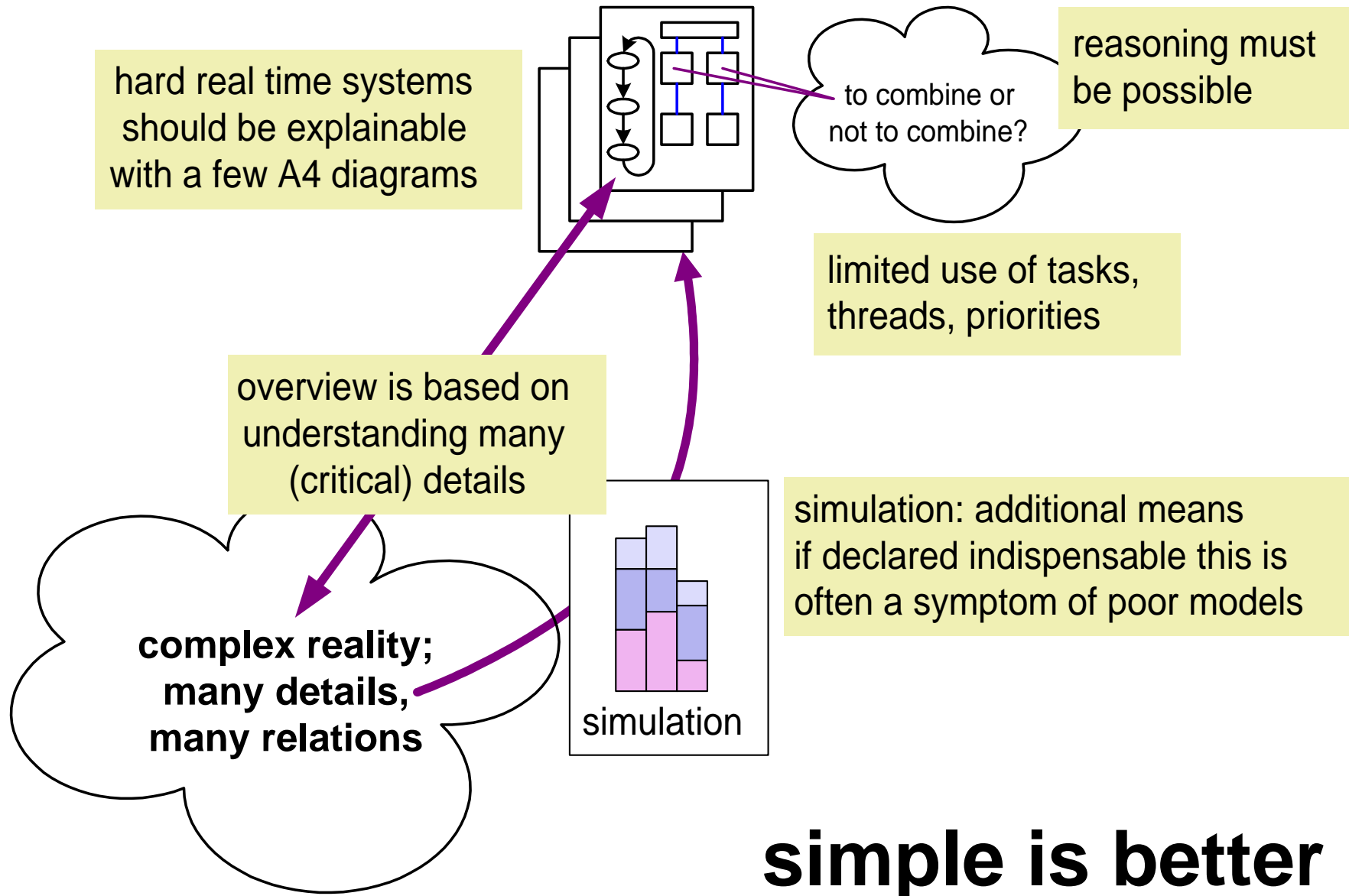
bus bandwidth, processor load [memory usage]  
useful macroscopic views, be aware of microscopic behavior



# Design recommendations separation of concerns

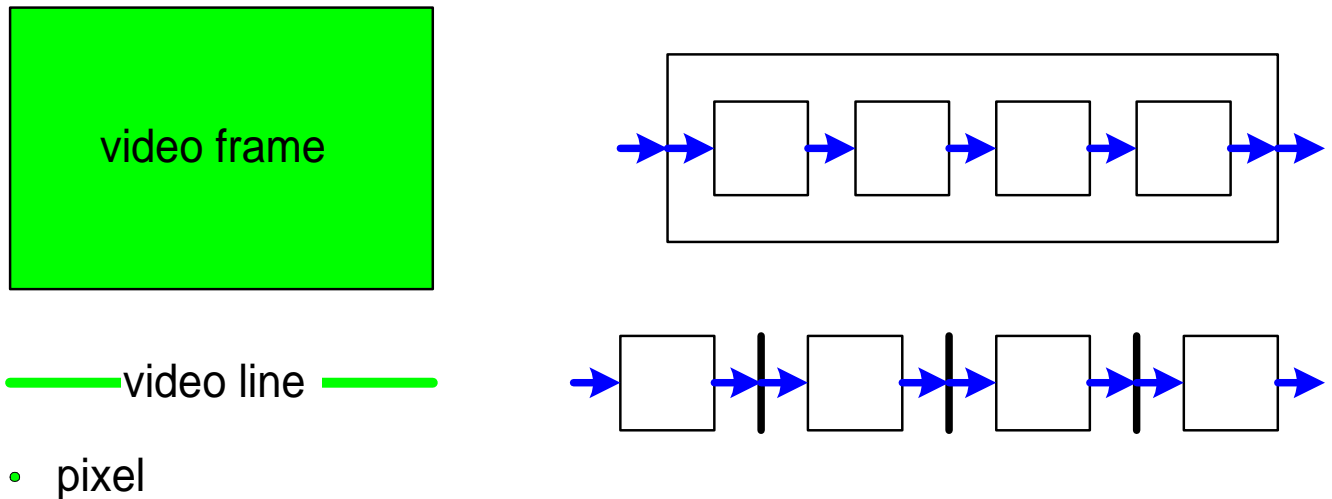


# Design recommendations understandability



# Granularity considerations

*unit of buffering*    ==  
or  
<>    *unit of synchronization*    ==  
or  
<>    *unit of processing*    ==  
or  
<>    *unit of I/O*



*fine grain:*  
flexible  
high overhead

*coarse grain:*  
rigid  
low overhead

# Design patterns

## synchronous

safety critical, reliable, subsystems

very low overhead  
predictable  
understandable

works best in total separation  
does not work for multiple rhythms

## thread based

Asynchronous applications and services

separation of timing concerns  
sharing of resources (no wait)

poor understanding of concurrency  
danger of high overhead

## timer based

regular rhythm;  
e.g. monitor HW status, update time,  
status display

low "tunable" overhead  
understandable

fast rhythms significant overhead

## interrupt based

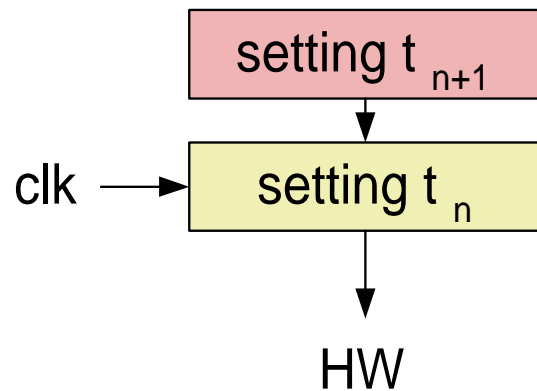
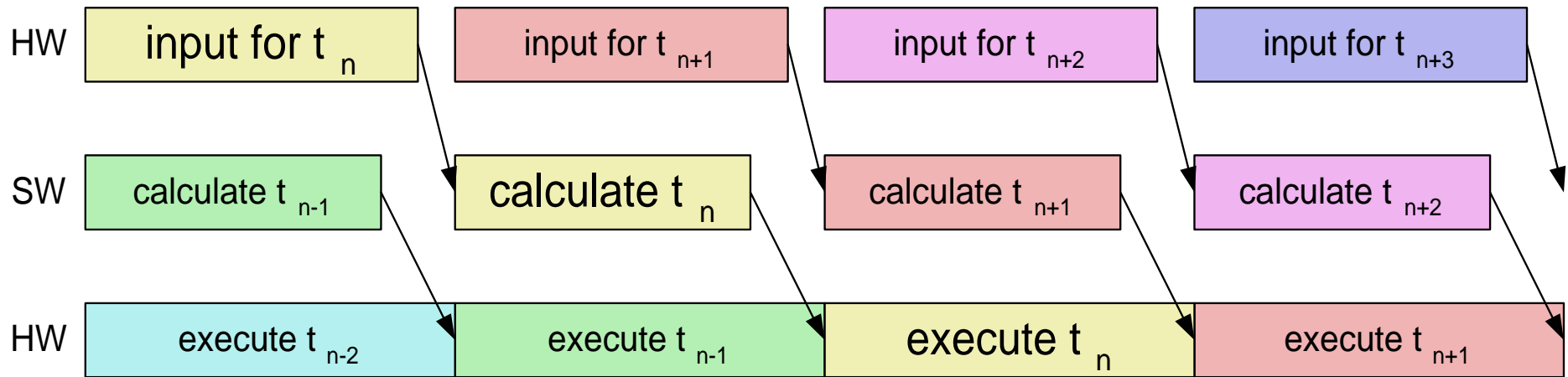
I/O and HW events  
data available, display frame sync

separation of timing concerns

definition of interrupts determines:  
overhead, understandability

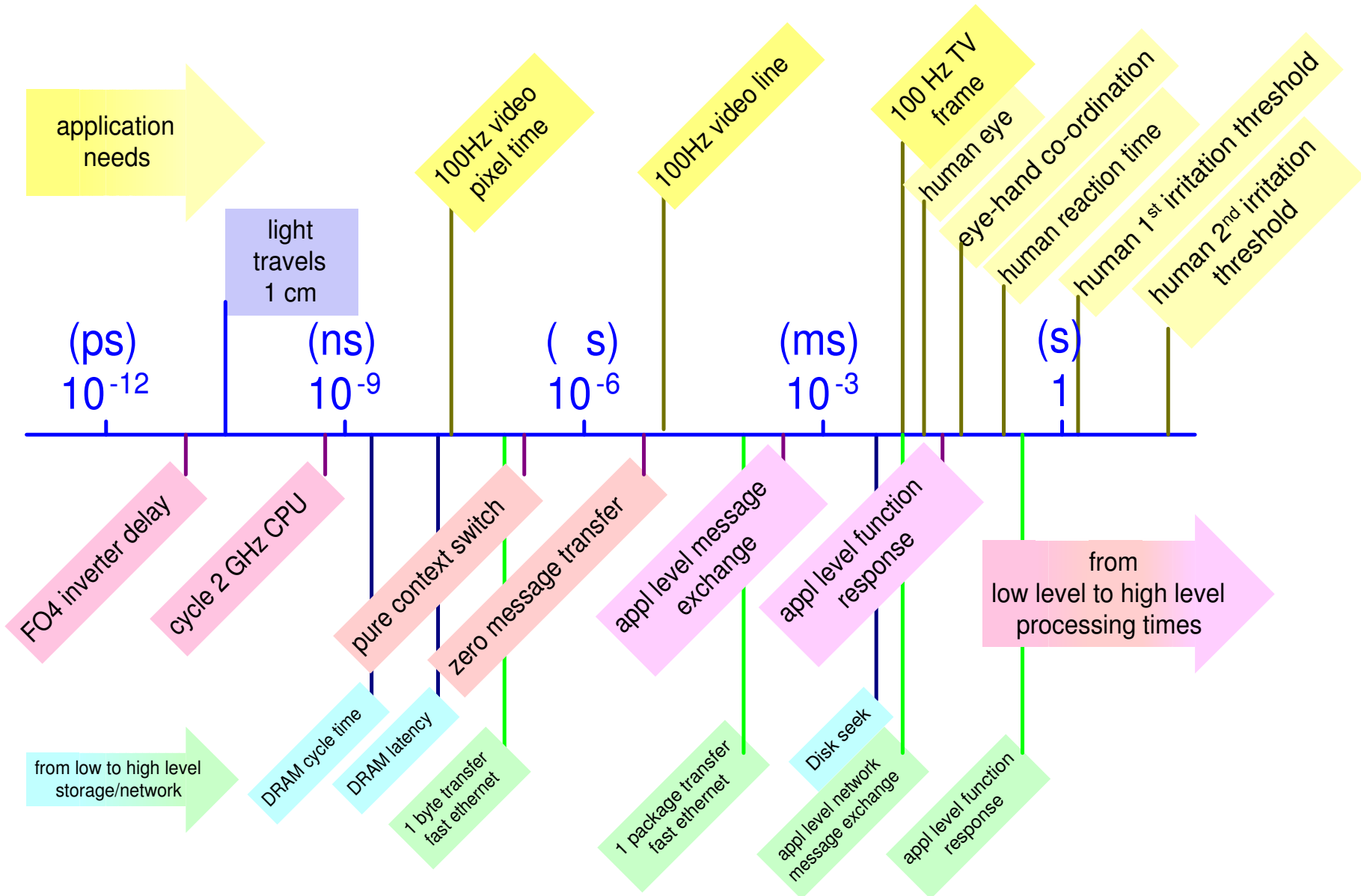


# Synchronous design



double buffer:  
full decoupling of calculation and execution

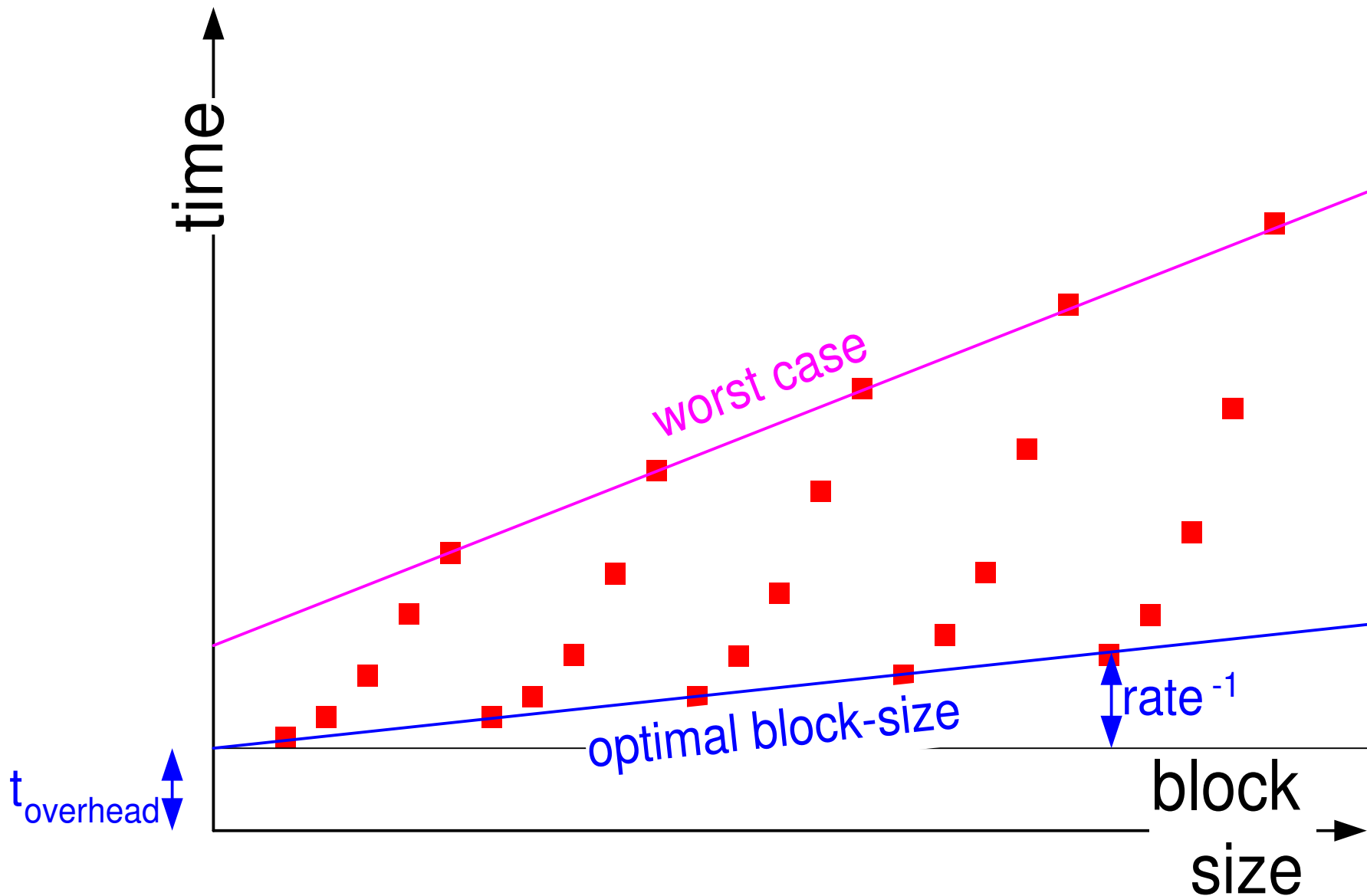
# Actual timing on logarithmic scale



# Typical micro benchmarks for timing aspects

	<i>infrequent operations, often time-intensive</i>	<i>often repeated operations</i>
<i>database</i>	start session finish session	perform transaction query
<i>network, I/O</i>	open connection close connection	transfer data
<i>high level construction</i>	component creation component destruction	method invocation same scope other context
<i>low level construction</i>	object creation object destruction	method invocation
<i>basic programming</i>	memory allocation memory free	function call loop overhead basic operations (add, mul, load, store)
<i>OS</i>	task, thread creation	task switch interrupt response
<i>HW</i>	power up, power down boot	cache flush low level data transfer

# The transfer time as function of blocksize



# Example of a memory budget

---

<i>memory budget in Mbytes</i>	code	obj data	bulk data	total
shared code	11.0			11.0
User Interface process	0.3	3.0	12.0	15.3
database server	0.3	3.2	3.0	6.5
print server	0.3	1.2	9.0	10.5
optical storage server	0.3	2.0	1.0	3.3
communication server	0.3	2.0	4.0	6.3
UNIX commands	0.3	0.2	0	0.5
compute server	0.3	0.5	6.0	6.8
system monitor	0.3	0.5	0	0.8
application SW total	13.4	12.6	35.0	61.0
UNIX Solaris 2.x				10.0
file cache				3.0
total				74.0

# Complicating factors and measures

---

## *complications*

cache

bus allocation

memory management

garbage collection

memory (buffer, storage) fragmentation

non preemptable OS activities

"hidden" dependencies (ie [dead]locks)

systematic "coincidences", avalanche triggers

instable response, performance

## *measures*

considered margin

explicit behavior

architecture rules

monitoring, logging

pool management

feedback to architect

flipover simulation