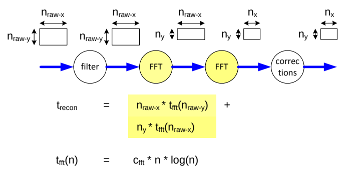
Lecture slides course Architecting System Performance

by *Gerrit Muller*
USN-SE

Abstract

The course Architecting System Performance provides an approach to design performance for software intensive systems. Core to the approach is the combination of measuring and modeling. Models are used for reasoning and analysis of performance, scalability, sensitivity and robustness. The course emphasis is on practice, not on theory. For example patterns and pitfalls from practice are provided.

March 6, 2021
status: draft
version: 0.3

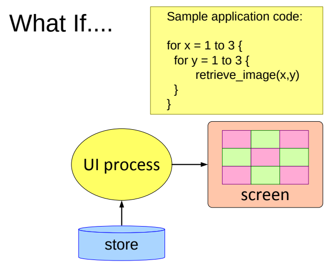


Introduction to System Performance Design

by *Gerrit Muller* University of South-Eastern Norway-NISE
e-mail: `gaudisite@gmail.com`
`www.gaudisite.nl`

Abstract

What is System Performance? Why should a software engineer have knowledge of the other parts of the system, such as the Hardware, the Operating System and the Middleware? The applications that he/she writes are self-contained, so how can other parts have any influence? This introduction sketches the problem and shows that at least a high level understanding of the system is very useful in order to get optimal performance.



March 6, 2021
status: preliminary draft
version: 0.5

Content of Problem Introduction

content of this presentation

Example of problem

Problem statements

Image Retrieval Performance

application need:

at event 3*3 show 3*3 images instanteneous

design

design

Sample application code:

```
for x = 1 to 3 {
   for y = 1 to 3 {
         retrieve_image(x,y)
   }
}
```

or

alternative application code:

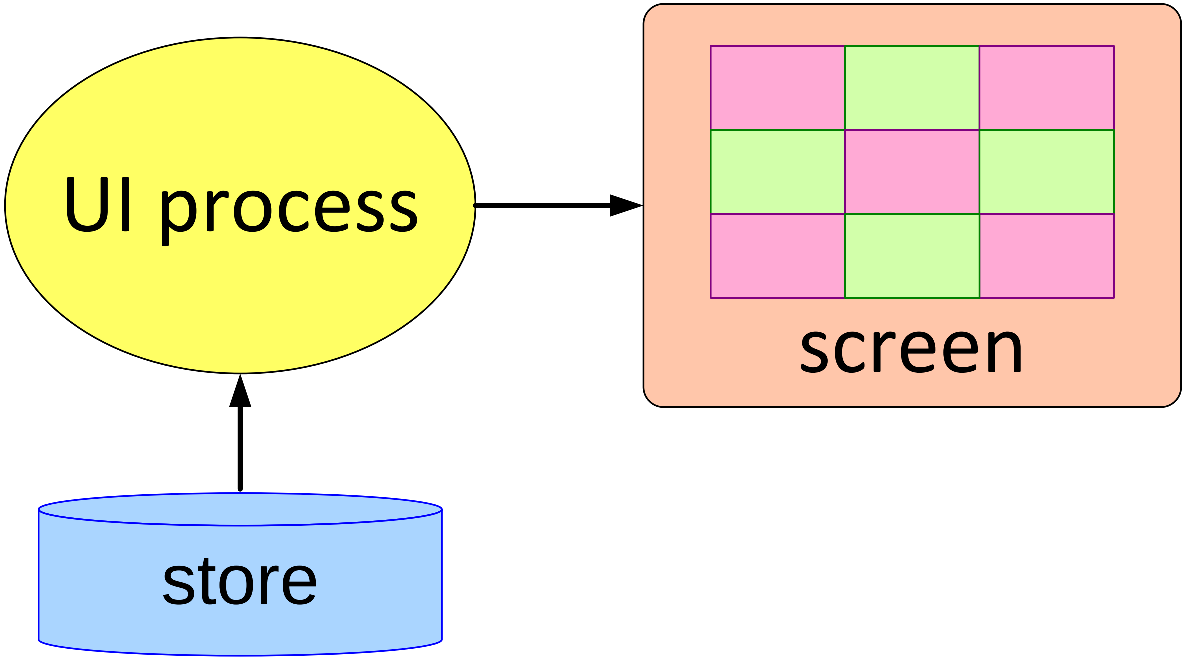
event 3*3 -> show screen 3*3

```
<screen 3*3>
    <row 1>
    <col 1><image 1,1></col 1>
    <col 2><image 1,2></col 2>
    <col 3><image 1,3></col 3>
  </row 1>
  <row 2>
    <col 1><image 1,1></col 1>
    <col 2><image 1,2></col 2>
    <col 3><image 1,3></col 3>
  </row 1>
  <row 2>
    <col 1><image 1,1></col 1>
    <col 2><image 1,2></col 2>
    <col 3><image 1,3></col 3>
  </row 3>
</screen 3*3>
```

What If....

Sample application code:

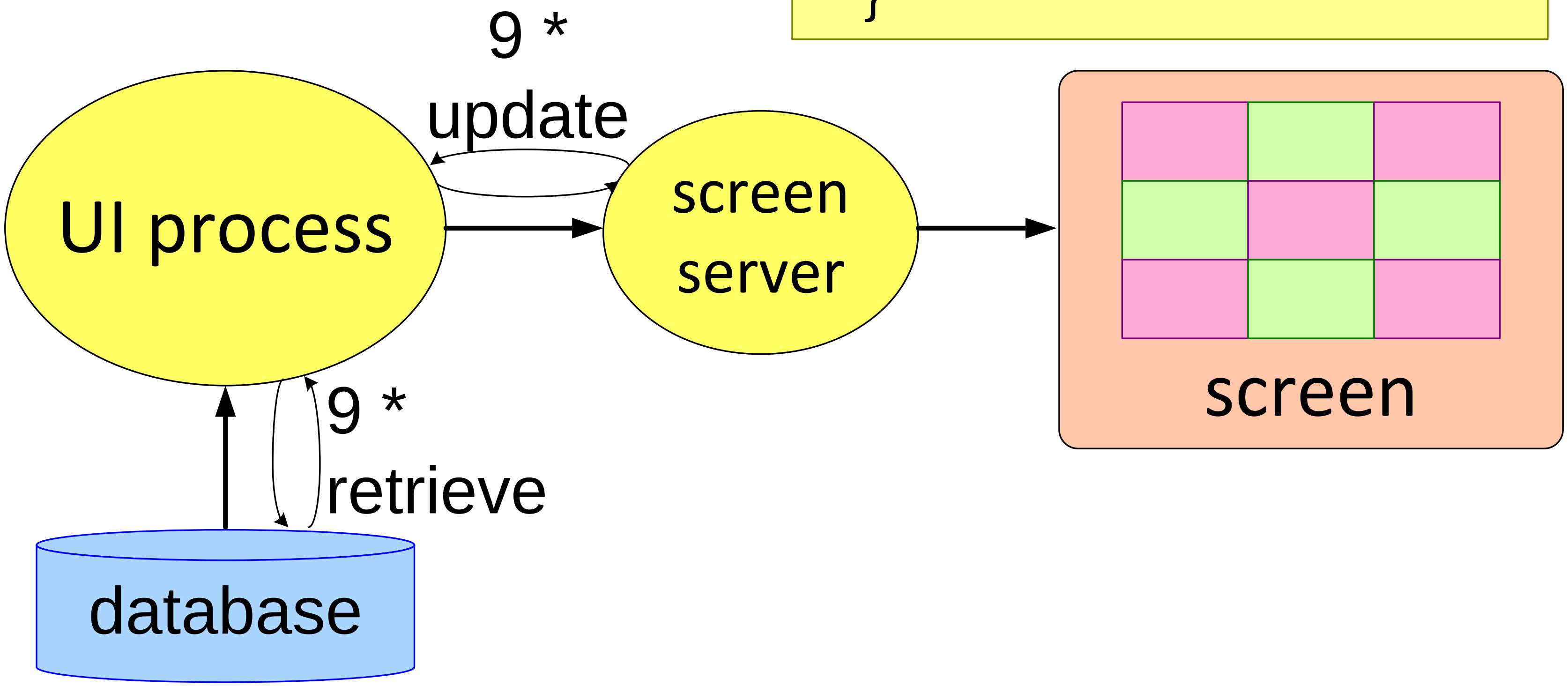
```
for x = 1 to 3 {
  for y = 1 to 3 {
        retrieve_image(x,y)
  }
}
```



What If....

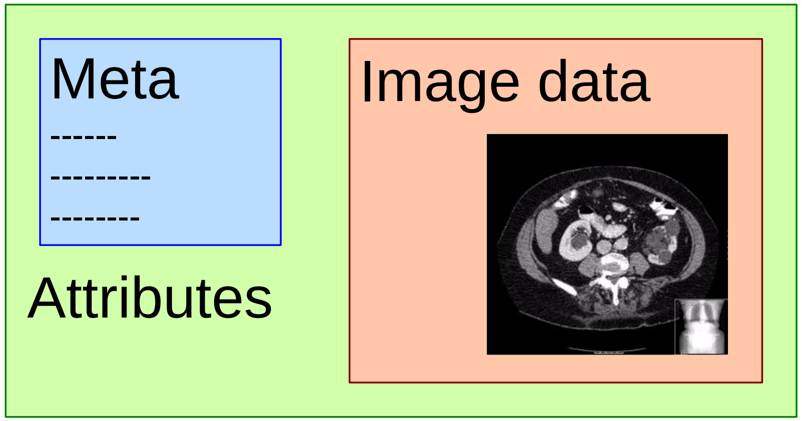
Sample application code:

```
for x = 1 to 3 {
  for y = 1 to 3 {
        retrieve_image(x,y)
  }
}
```



Meta Information Realization Overhead

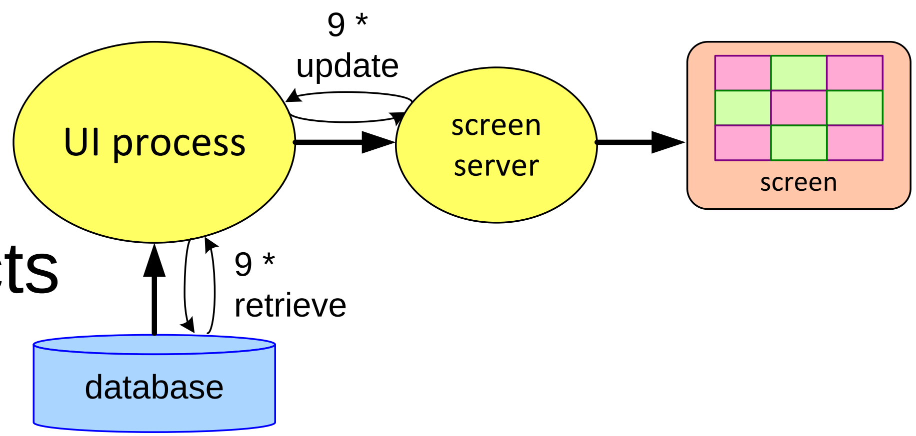
What If....



Sample application code:

```
for x = 1 to 3 {
  for y = 1 to 3 {
        retrieve_image(x,y)
  }
}
```

Attribute = 1 COM object
100 attributes / image
9 images = 900 COM objects
1 COM object = 80µs
9 images = 72 ms



What If....

Sample application code:

```
for x = 1 to 3 {
  for y = 1 to 3 {
        retrieve_image(x,y)
  }
}
```

- I/O on line basis (512^2 image)

$9 * 512 * t_{I/O}$

$t_{I/O} \sim= 1ms$

- . . .

Non Functional Requirements Require System View

Sample application code:

```
for x = 1 to 3 {
  for y = 1 to 3 {
        retrieve_image(x,y)
  }
}
```

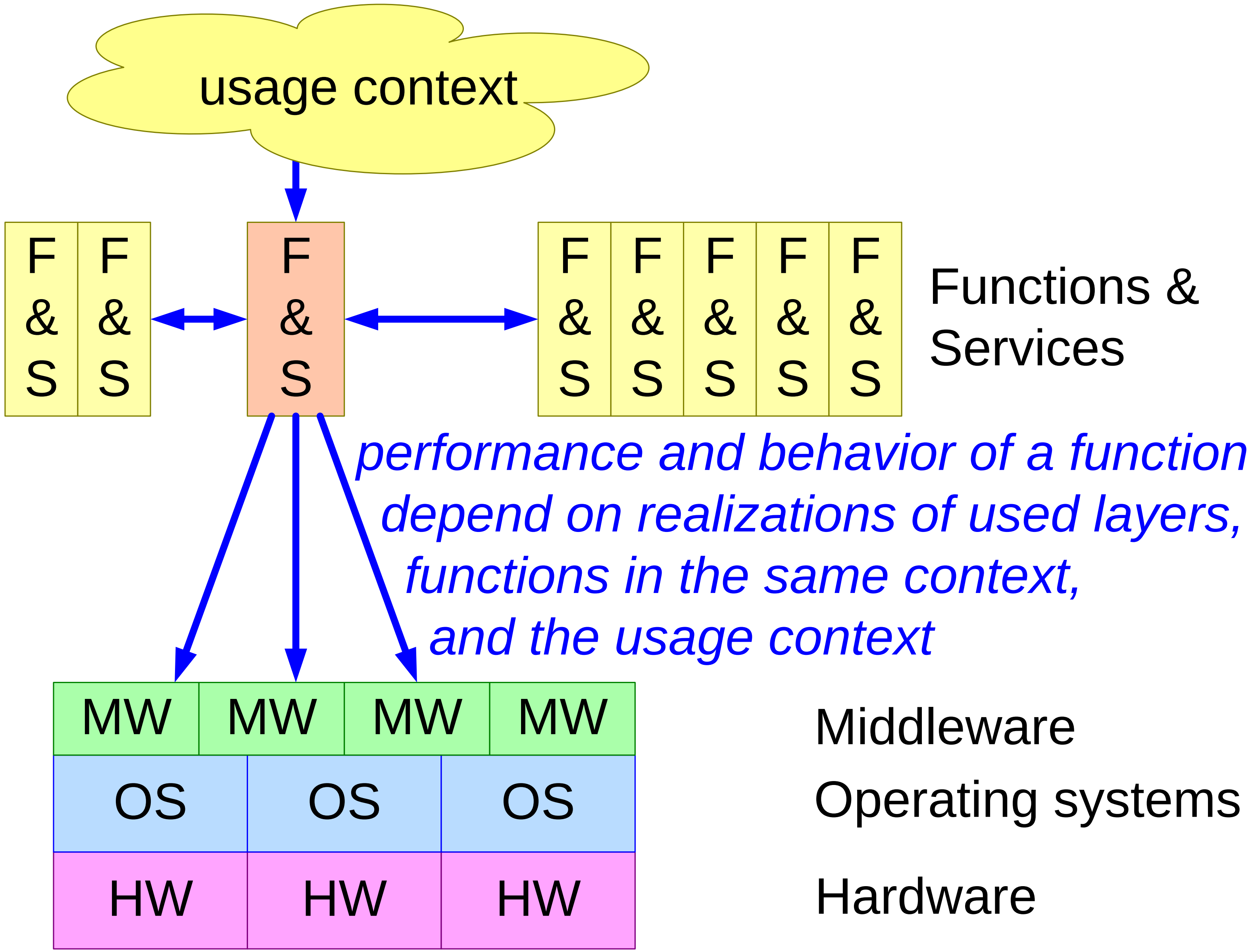
can be:

- fast, but very local
- slow, but very generic
- slow, but very robust
- fast and robust
- ...

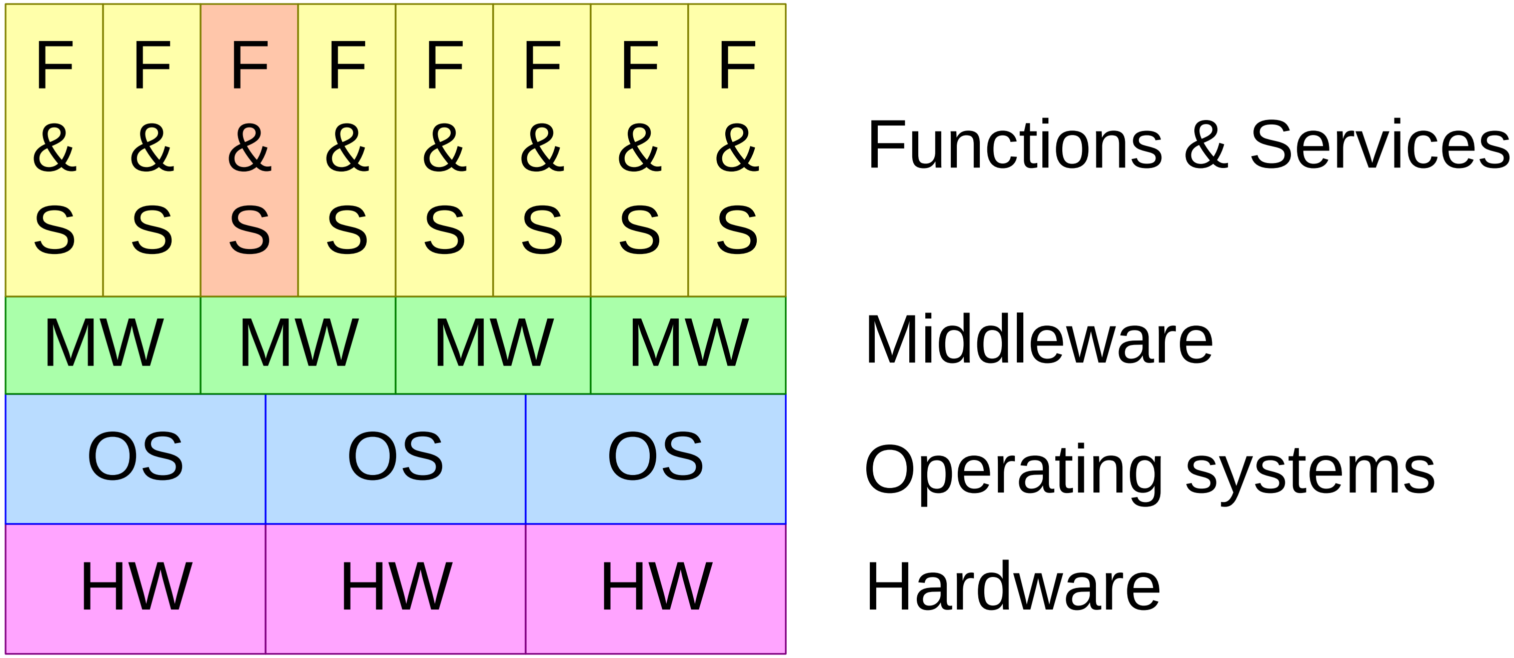
The emerging properties (behavior, performance) cannot be seen from the code itself!

Underlying platform and neighbouring functions determine emerging properties mostly.

Function in System Context



Challenge



Performance = Function (F&S, other F&S, MW, OS, HW)
MW, OS, HW >> 100 Manyear : very complex

Challenge: How to understand MW, OS, HW
with only a few parameters

Summary of Problem Introduction

Summary of Introduction to Problem

Resulting System Characteristics cannot be deduced from local code.

Underlying platform, neighboring applications and user context:

have a big impact on system characteristics

are big and complex

Models require decomposition, relations and representations to analyse.

From Synchronous to Asynchronous Design

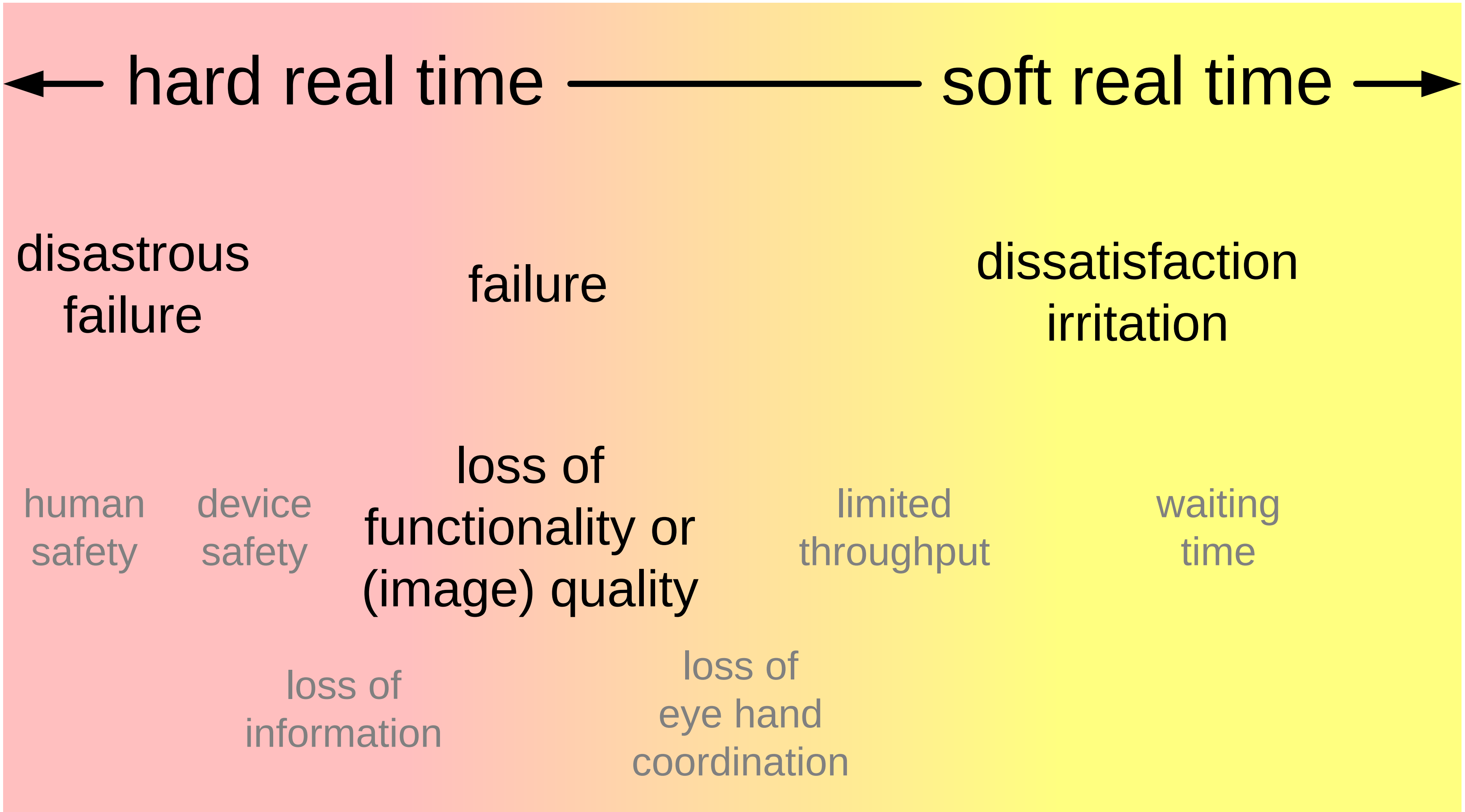
by *Gerrit Muller* HSN-NISE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

Abstract

The most simple real time programming paradigm is a synchronous loop. This is an effective approach for simple systems, but at a certain level of concurrent activities an asynchronous design, based on scheduling tasks, becomes more effective. We will use a conventional television as case to show real time design strategies, starting with a straightforward analog television based on a synchronous design and incrementally extending the television to become a full-fledged digital TV with many concurrent functions.

March 6, 2021
status: preliminary draft
version: 0

Hard Real Time Design



Case Simple Analog TV

Simple Analog TV

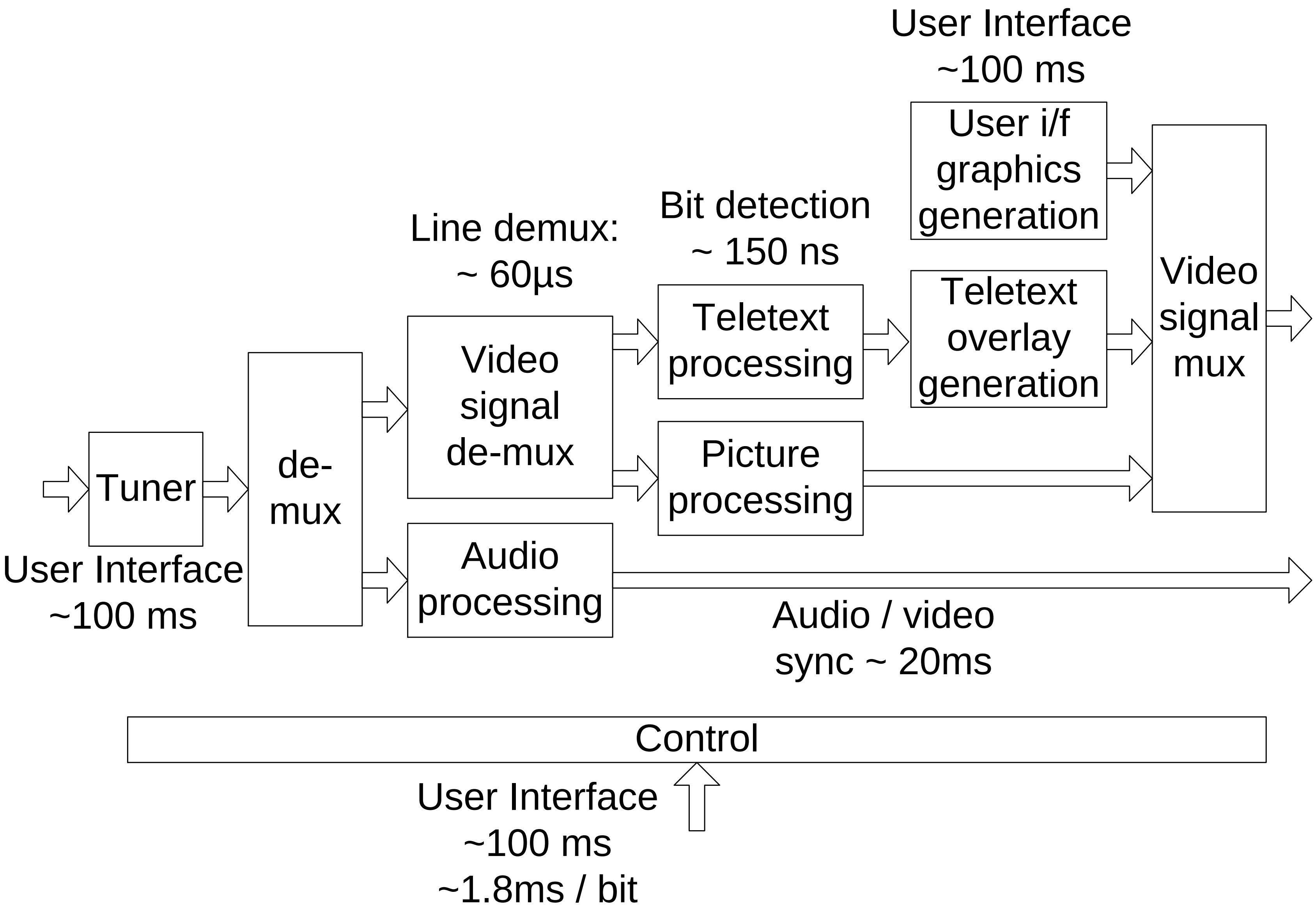
Multiple views on system

Fundamentals of *periodic* or *streaming* Hard Real-Time applications

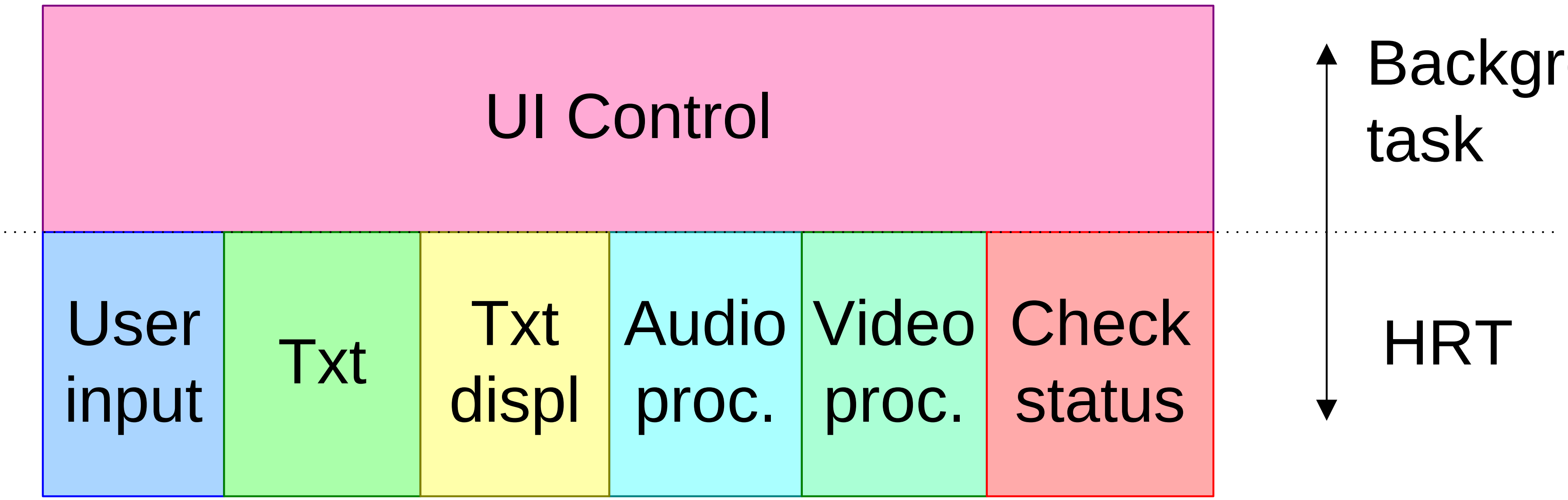
System performance characterisation: Performance model

Synchronous design concept

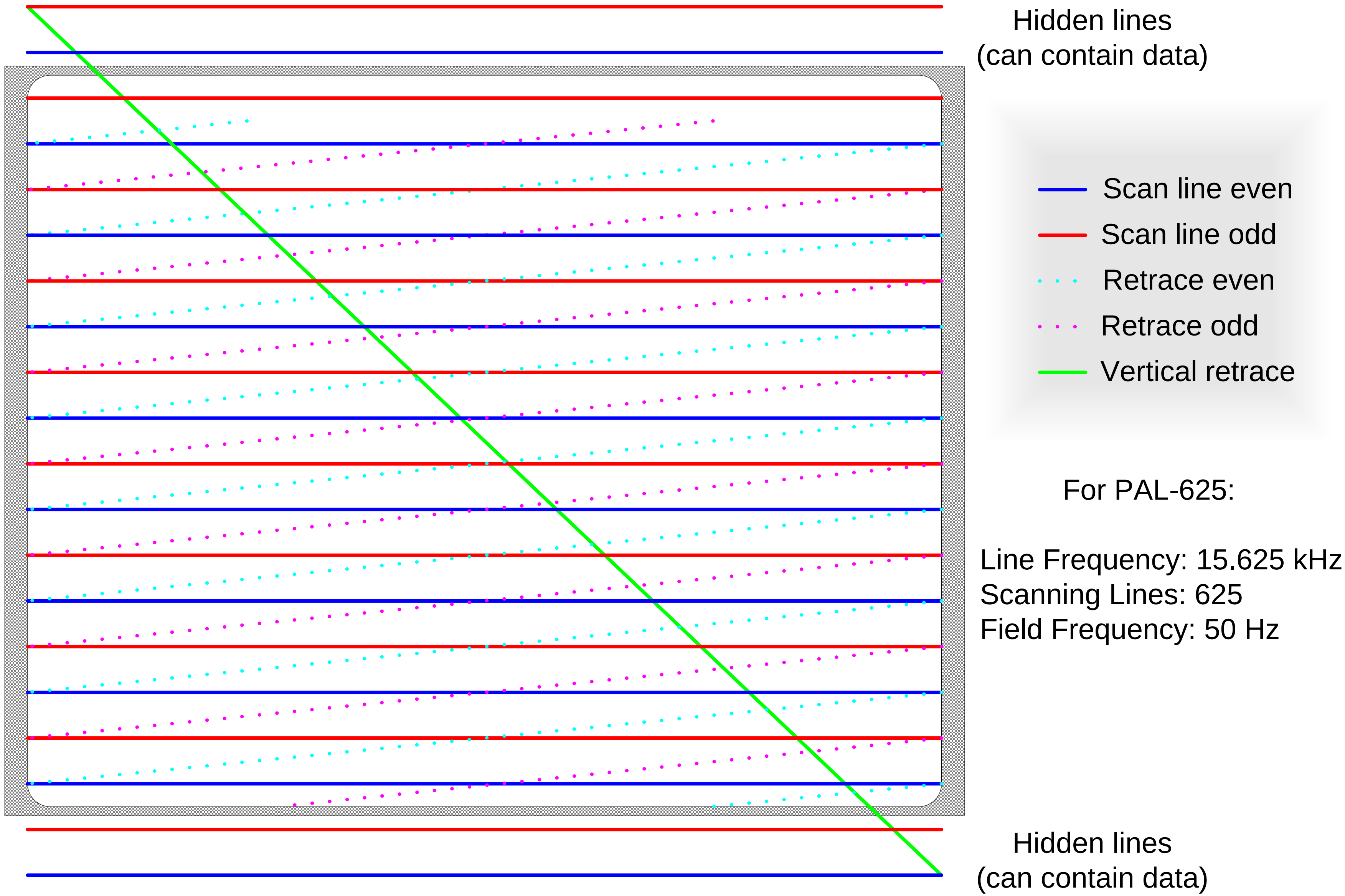
Functional Flow Simple Analog Television



SW Construction Diagram



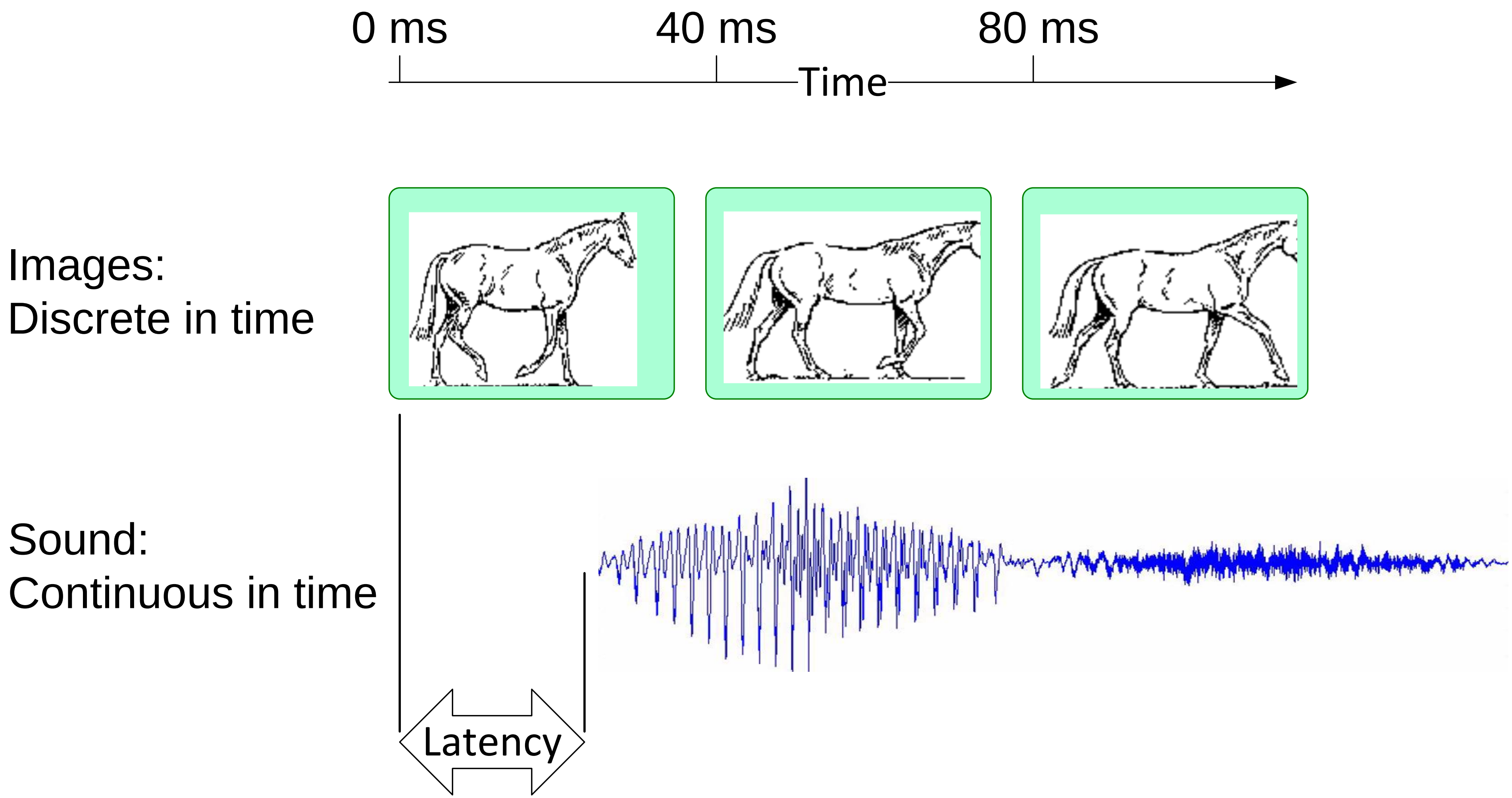
Video Timing



For PAL-625:

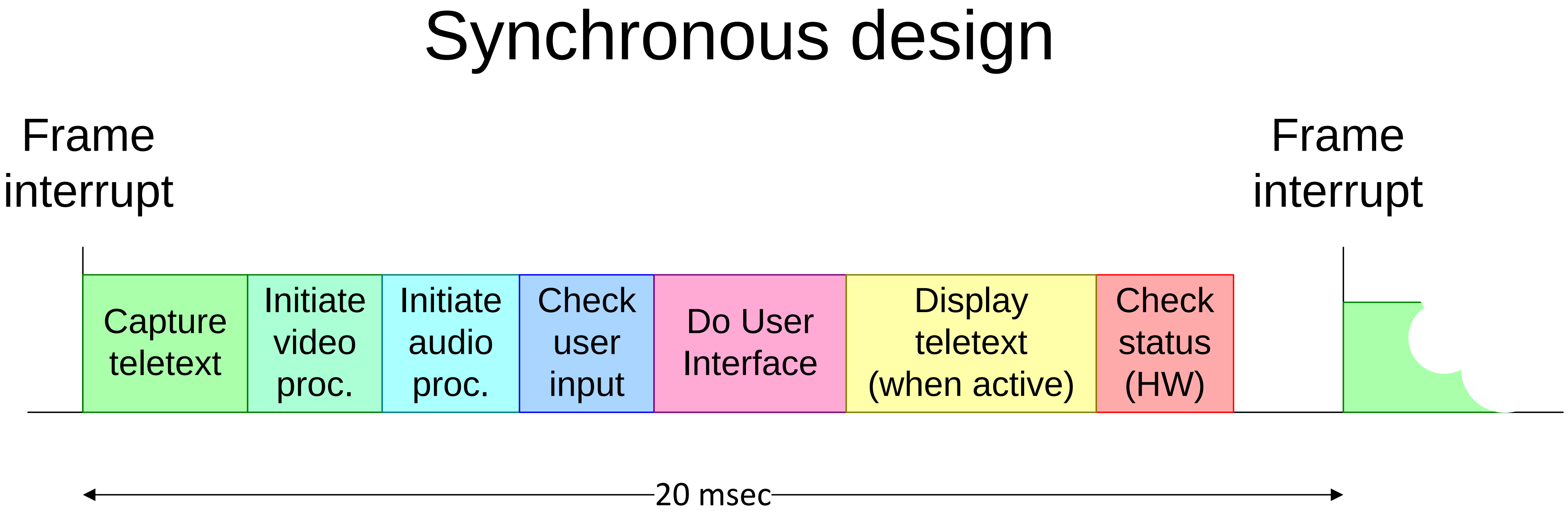
Line Frequency: 15.625 kHz
Scanning Lines: 625
Field Frequency: 50 Hz

Audio-Video Synchronization Requirement

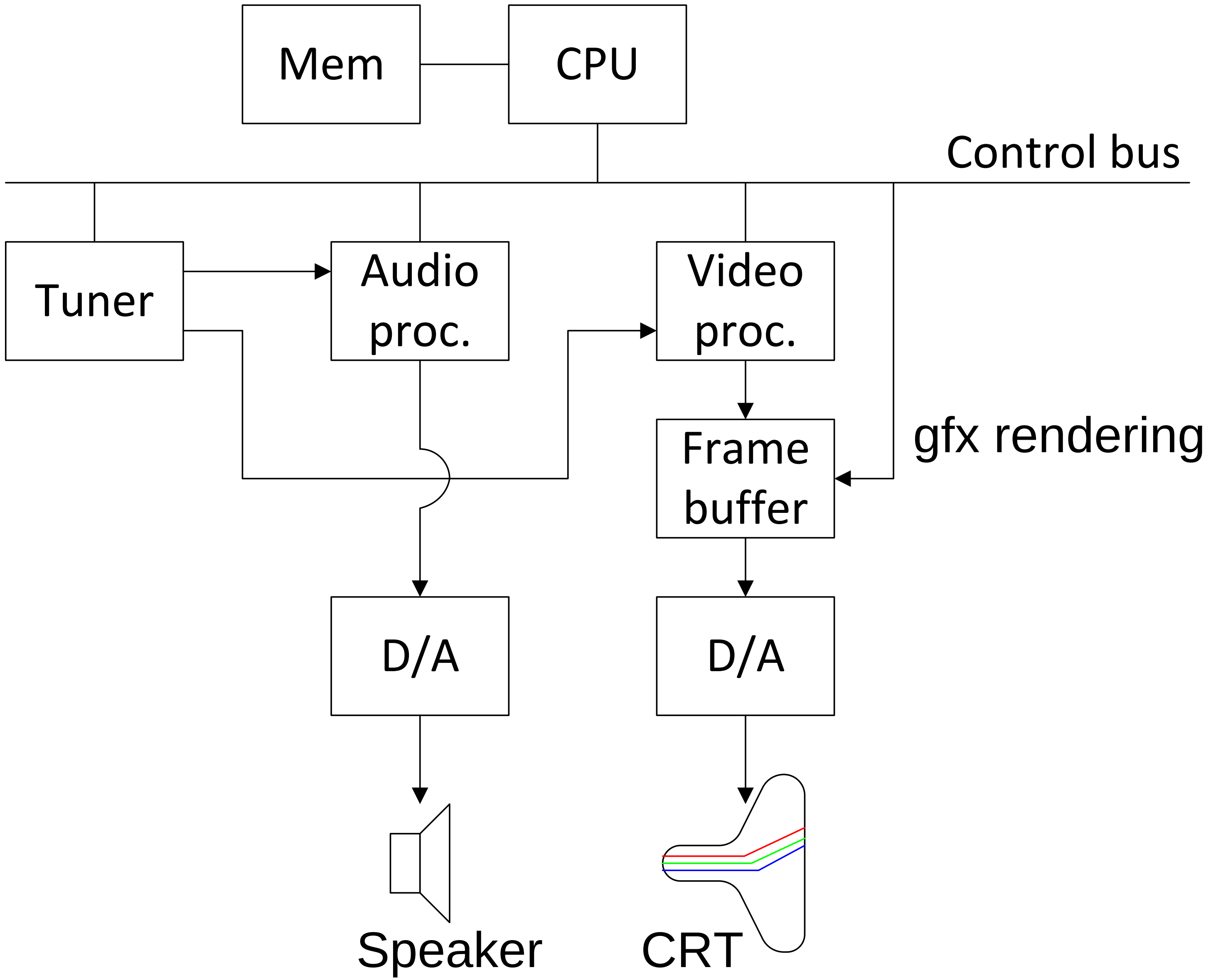


Sound and vision must be lip-sync or better
Maximum latency ~ +/- 100 msec

Synchronous Control Software



HW Diagram


Mem
CPU
Control bus
Tuner
Audio proc.
Video proc.
Frame buffer
gfx rendering
D/A
D/A
Speaker
CRT

Synchronous design questions

Estimate processing time on a 100 MHz ARM core

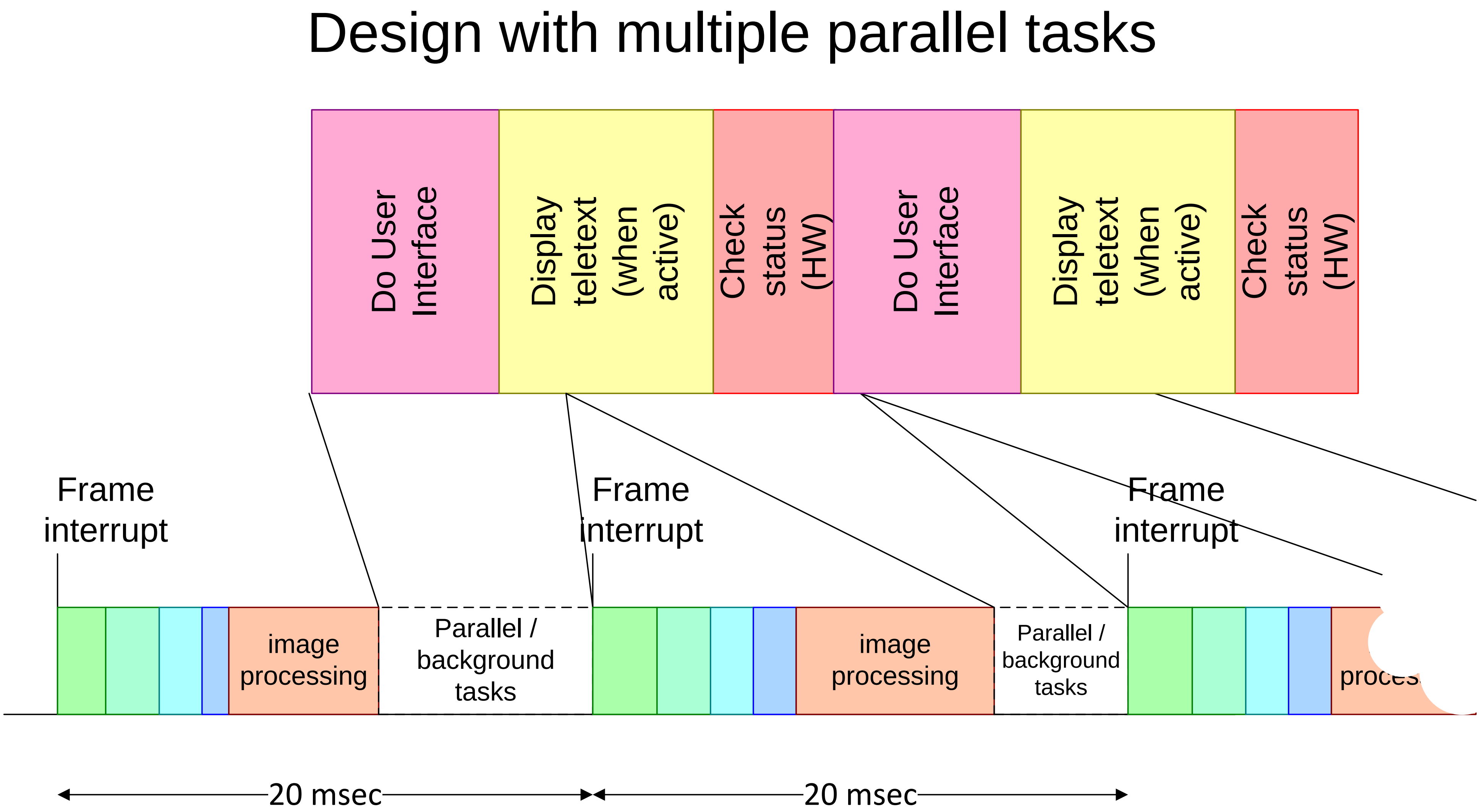
Assuming that all processing and acquisition is done in HW

Graphics rendering (user interface + teletext display) is done in SW

Where do you expect variation?

How feasible and how reliable is this design?

Low Priority Work in the Background



Synchronous or Asynchronous?

Synchronous	=> Map on Highest frequency Constraints: - Processing frequency must be a whole (integer) multiple of the lower frequencies - Each process must be completed within the period of the highest frequency, together with the high-frequency process
A-Synchronous	=> Concurrent processes

Multiple Periods in a Simple TV

Input signal	50 Hz
Processing	100 Hz
User Interface	20 Hz
Power and Housekeeping	0.5 Hz
Output	50, 100 Hz

Summary Case Simple Analog TV

Simple Analog TV

Performance model requires:

- identification of processing steps
- their relation
- critical parameters and values

Synchronous design sufficient for periodic applications with one dominant frequency

Multiple views on system:

- HW diagram
- SW construction diagram
- Functional flow
- Time-line

Case Digital Television

From Analog TV to Digital TV

Adding more input formats and output devices

Multiple heterogenous periods: asynchronous design with concurrent tasks.

Digital Television

Input	Many frequencies
	Video & Audio variable timing
Output	Many frequencies
Processing	Variable

Many video variants (see table)
Many audio variants (quality, number of speakers, ...)

multi task design complex TV

In modern television the format of the image can change (e.g. widescreen)

The user can set the refresh rate to higher values (e.g. 100Hz anti-flicker)

Different displays (CRT, LCD, Plasma) can be attached
that need the image in different formats
(interlaced, non-interlaced, different refresh rates)

Non interlaced images need special filtering of the image
to prevent ragged images

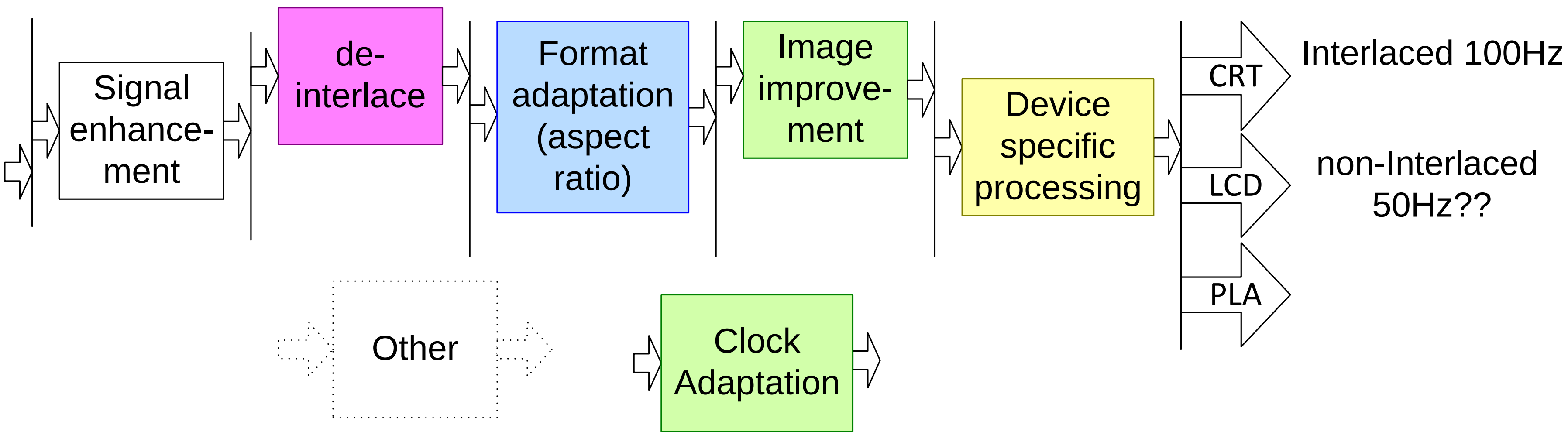


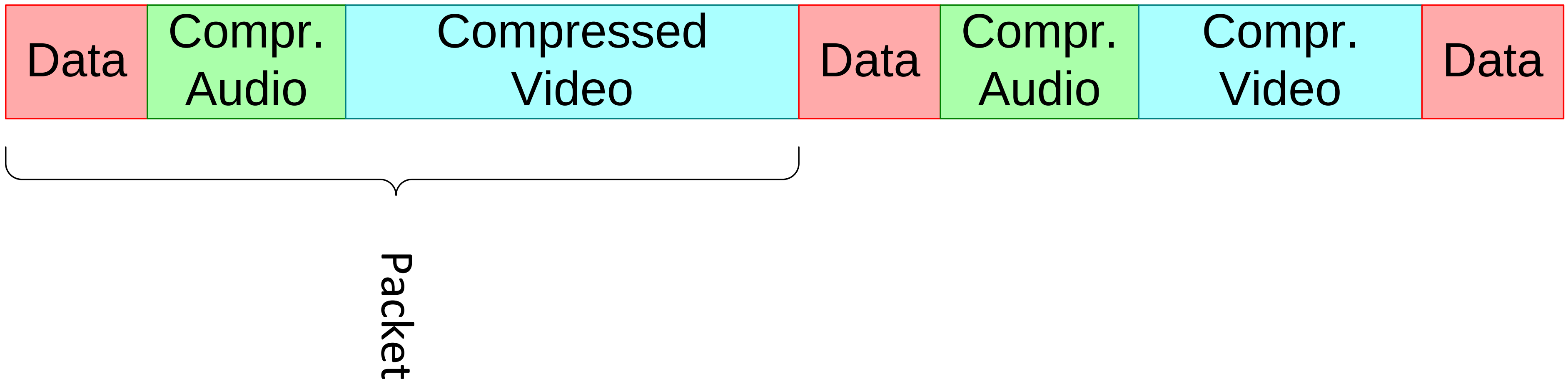
Table with ATSC Video Formats

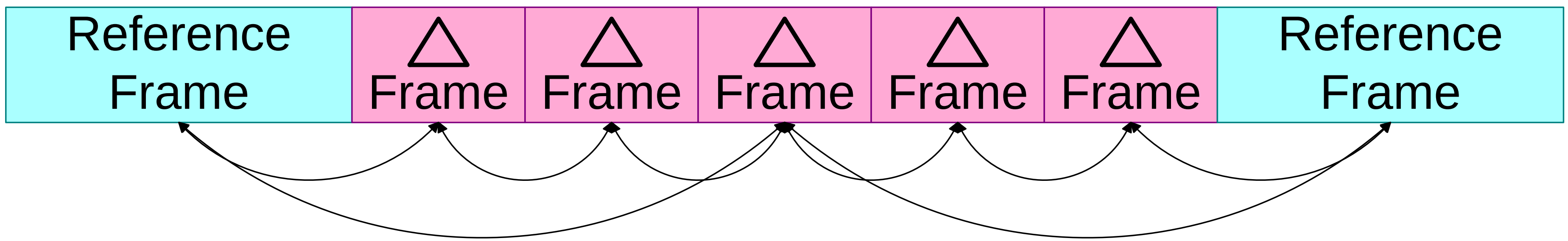
spec	Horizontal pixels	Vertical pixels	Aspect ratio	Monitor interface	Format name	Frames per sec	Fields per sec	Transmitted interlaced
					1080i60	30	60	yes
	1920	1080	16:09	1080i	1080p30	30	30	no
					1080p24	24	24	no
					720p60	60	60	no
	1280	720	16:09	720p	720p30	30	30	no
					720p24	24	24	no
				480p	480p60	60	60	no
	704	480	16:09		480i60	30	60	yes
				480i	480p30	30	30	no
ATSC					480p24	24	24	no
				480p	480p60	60	60	no
	704	480	04:03		480i60	30	60	yes
				480i	480p30	30	30	no
					480p24	24	24	no
				480p	480p60	60	60	no
	640	480	04:03		480i60	30	60	yes
	640			480i	480p30	30	30	no
					480p24	24	24	no
NTSC	»640	483	04:03	Note 1	Note 1	30	60	yes

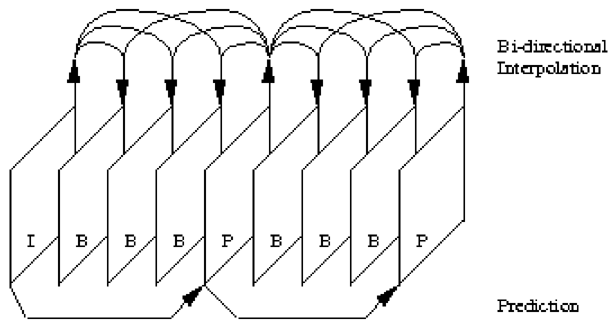
Note 1: Some people refer to NTSC as 480i.

Source: http://www.hdtvprimer.com/ISSUES/what_is_ATSC.html

Data Packets in Digital TV







Summary Case Digital Television

From Analog TV to Digital TV

Real-life applications rapidly introduce all kinds of variations

Concurrent tasks cope with different periods

ASP Python Exercise

by *Gerrit Muller* University of South-Eastern Norway-NISE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

Abstract

A simple measurement exercise is described. Purpose of this exercise is to build up experience in measuring and its many pitfalls. The programming language Python is used as platform, because of its availability and low threshold for use.

March 6, 2021
status: preliminary draft
version: 0

logo TBD

Case for Exercise

Select a programming environment,

where loop overhead and file open

can be measured in 30 minutes.

If this environment is not available,

then use Python.

Python download and information

Active State Python (Freeware distribution, runs directly)

http://www.activestate.com/Products/ActivePython/

Python Language Website

http://www.python.org/

Python Reference Card

http://admin.oreillynet.com/python/excerpt/PythonPocketRef/examples/python.pdf

Python example

```
import time

for n in (1,10,100,1000,10000,100000,1000000):
    a = 0
    tstart = time.time()
    for i in xrange(n):
        a = a+1
    tend=time.time()

    print n, tend-tstart, (tend-tstart)/n

def example_filehandling():
    f = open("c:\\temp\\test.txt")
    for line in f.readlines():
        print line
    f.close()

tstart = time.time()
example_filehandling()
tend=time.time()
print "file open, read & print, close: ",tend-tstart,"s"
```

```
>>>
1 0.0 0.0
10 0.0 0.0
100 0.0 0.0
1000 0.0 0.0
10000 0.00999999046326 9.99999046326e-007
100000 0.039999961853 3.9999961853e-007
1000000 0.44000005722 4.4000005722e-007
test line 1


line 2


line 3


file open, read, close:  0.039999961853 s
```

Exercise

- Perform the following measurements
  1. loop overhead
  2. file open
- Determine for every measurement:

  What is the expected result?

  What is the measurement error?

  What is the result?

  What is the credibility of the result?

  Explain the result.

  (optional) What is the variation? Explain the variation.

Reflection on Exercise

+ measuring is easy
+ measuring provides data and understanding

~ result and expectation often don't match

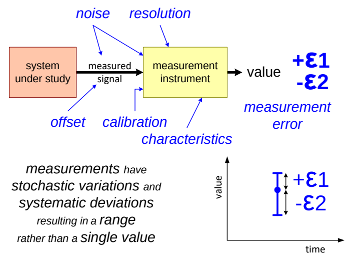
- sensible measuring is more difficult

Modeling and Analysis: Measuring

by *Gerrit Muller* University of South-Eastern Norway-SE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

Abstract

This presentation addresses the fundamentals of measuring: What and how to measure, impact of context and experiment on measurement, measurement errors, validation of the result against expectations, and analysis of variation and credibility.



March 6, 2021
status: preliminary draft
version: 1.2

Presentation Content

content

What and How to measure

Impact of experiment and context on measurement

Validation of results, a.o. by comparing with expectation

Consolidation of measurement data

Analysis of variation and analysis of credibility

Measuring Approach: What and How

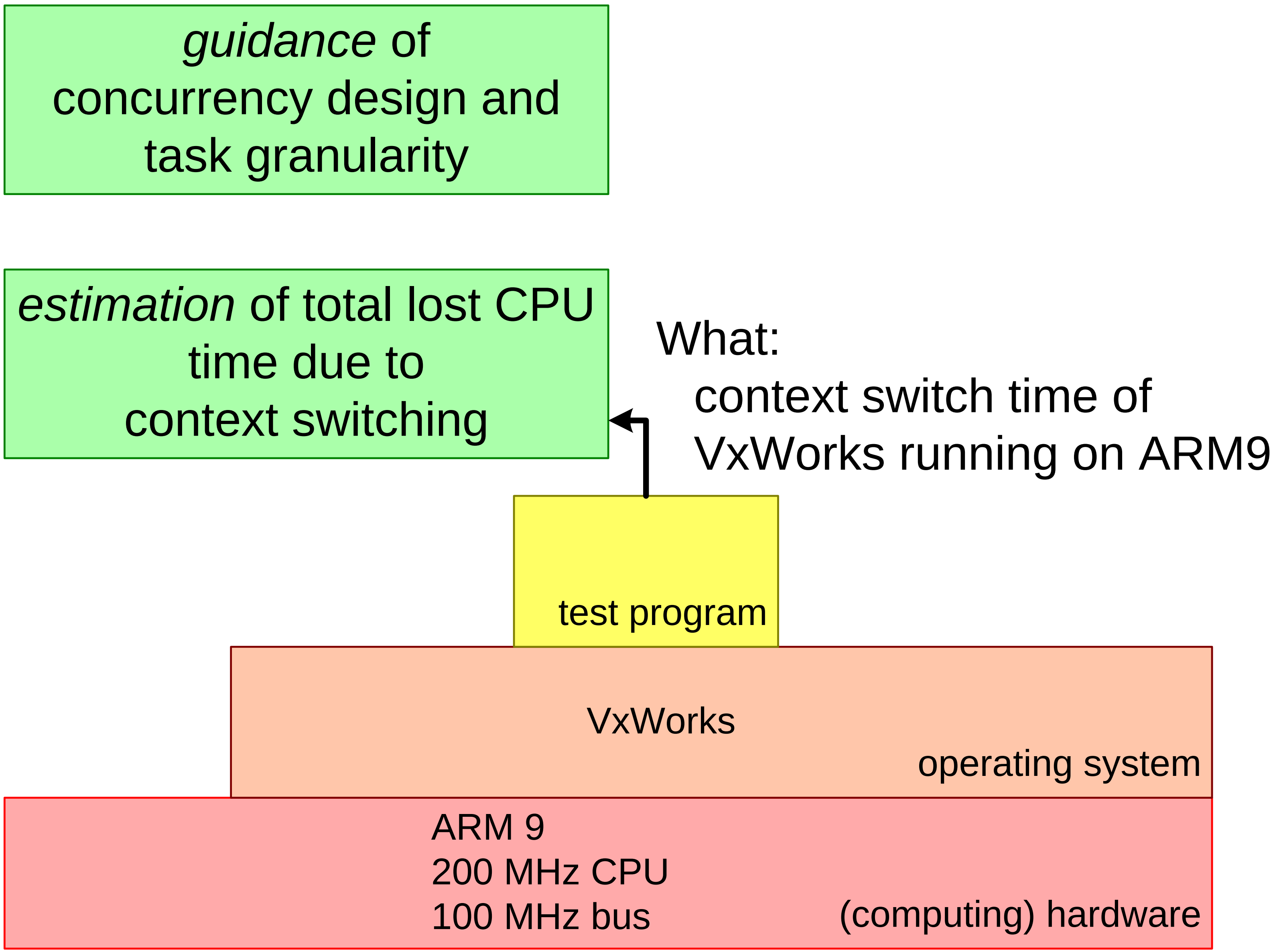
what

1. What do we need to know?	
2. Define quantity to be measured.	initial model
3. Define required accuracy	purpose
4A. Define the measurement circumstances	fe.g. by use cases
4B. Determine expectation	historic data or estimation
4C. Define measurement set-up	
5. Determine actual accuracy	uncertainties, measurement error
6. Start measuring	
7. Perform sanity check	expectation versus actual outcome

how

iterate

1. What do We Need? Example Context Switching

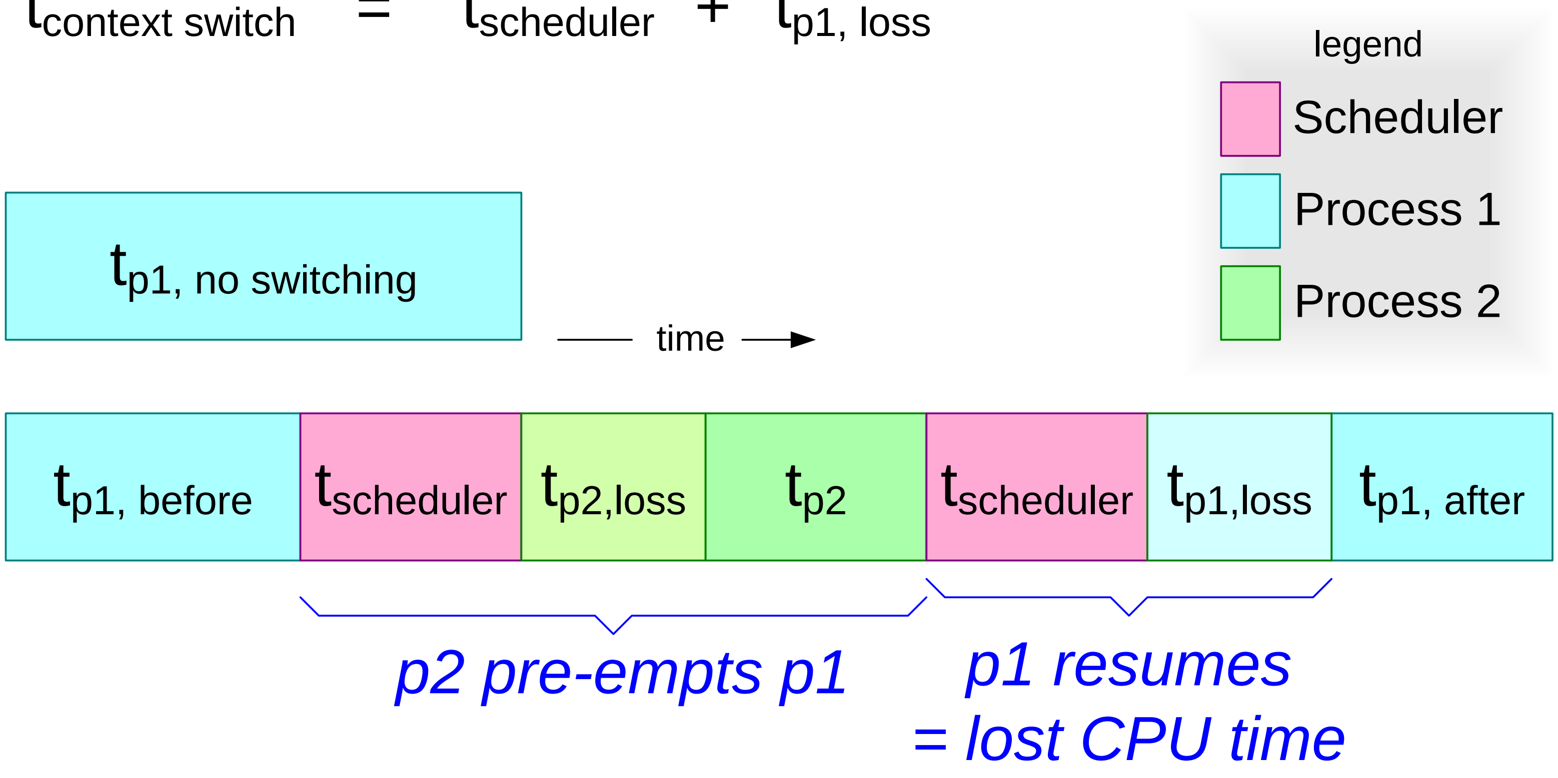


2. Define Quantity by Initial Model

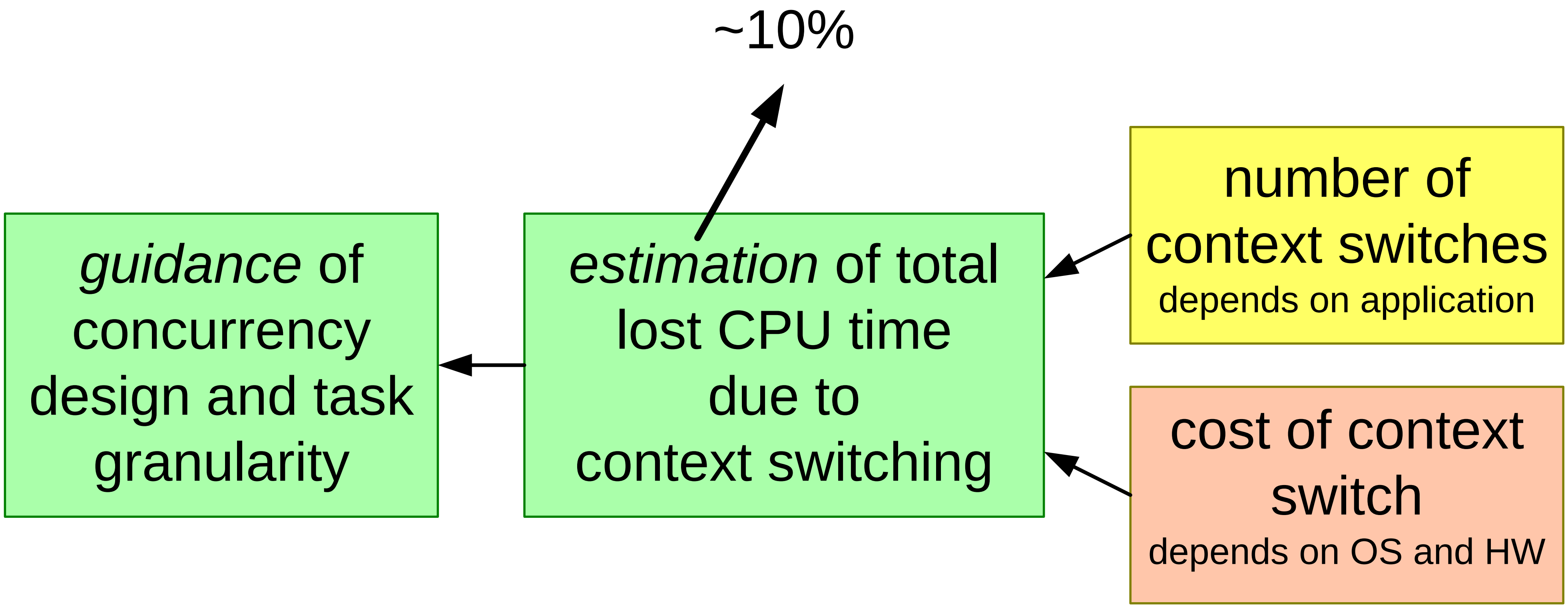
What (original):
context switch time of
VxWorks running on ARM9

What (more explicit):
The amount of lost CPU time,
due to context switching on
VxWorks running on ARM9
on a heavy loaded CPU

$$t_{\text{context switch}} = t_{\text{scheduler}} + t_{\text{p1, loss}}$$

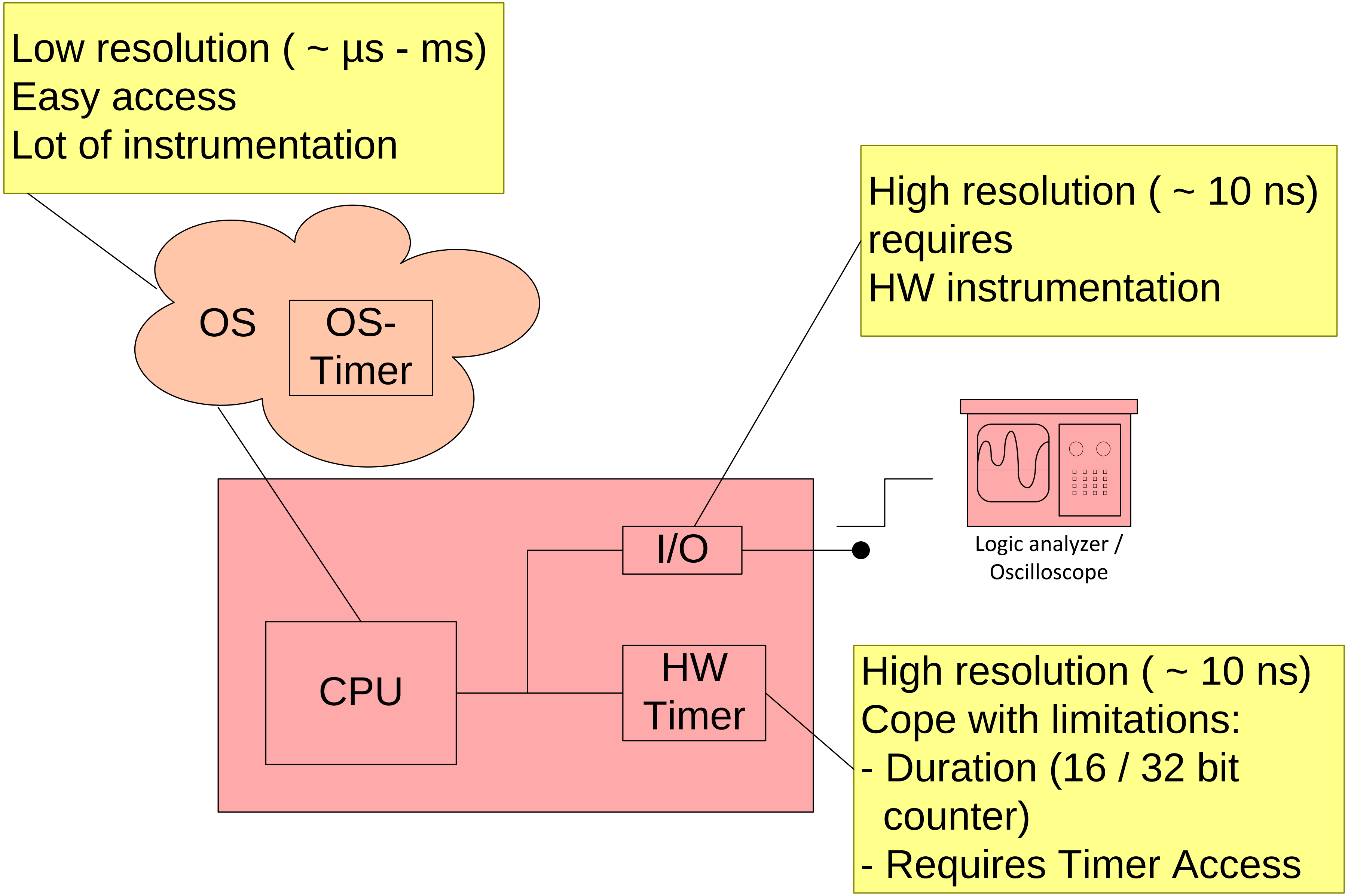


3. Define Required Accuracy



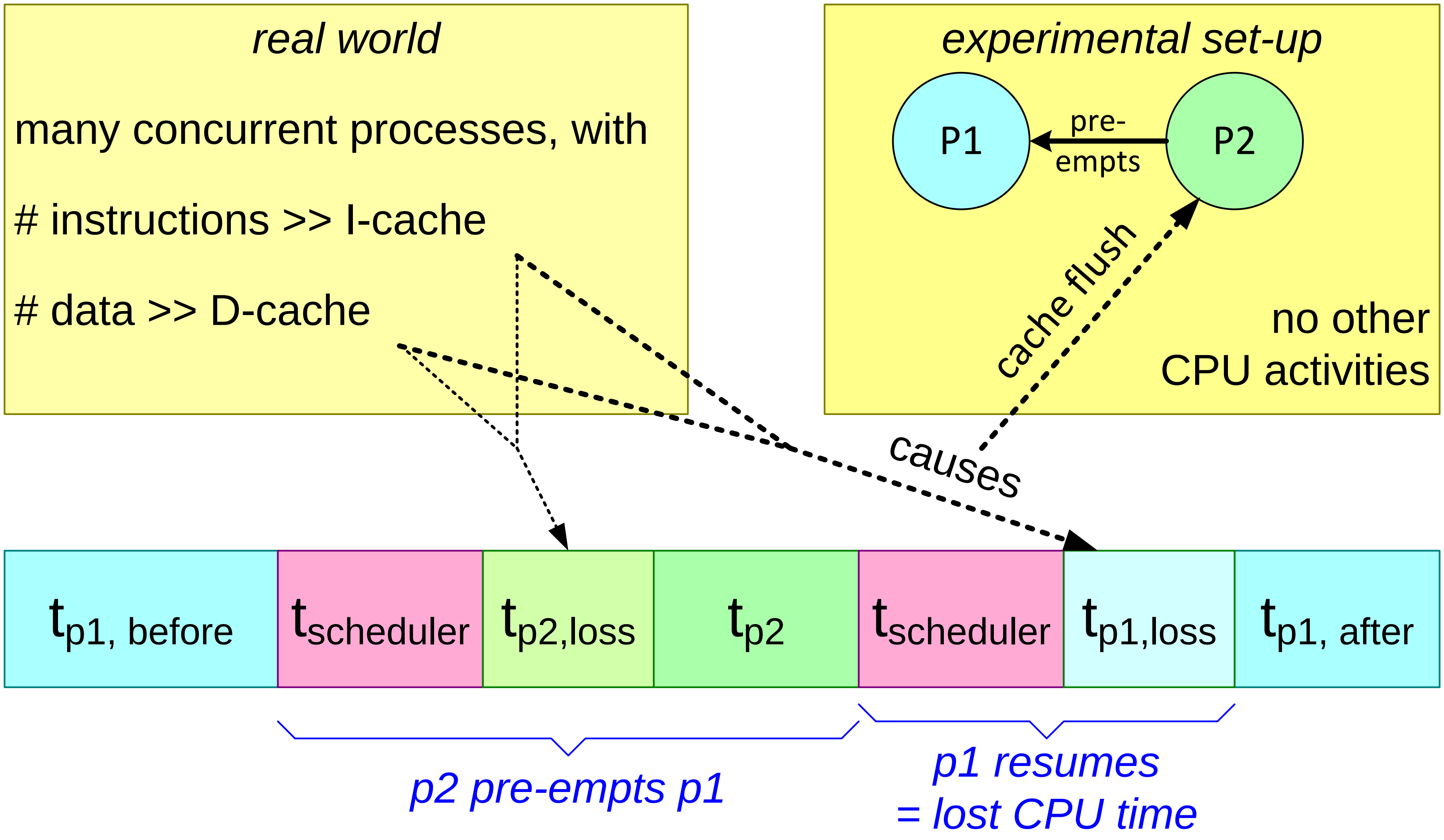
purpose drives required accuracy

Intermezzo: How to Measure CPU Time?

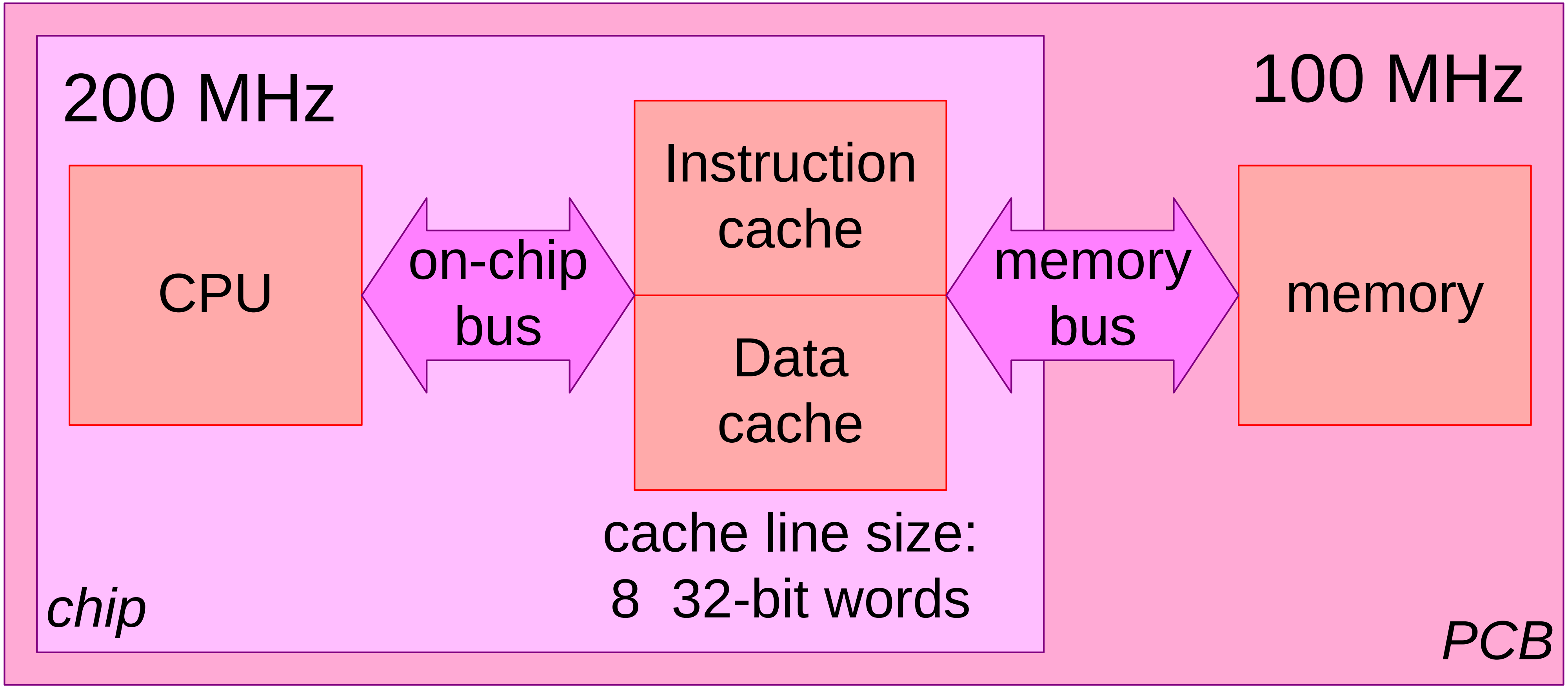


4A. Define the Measurement Set-up

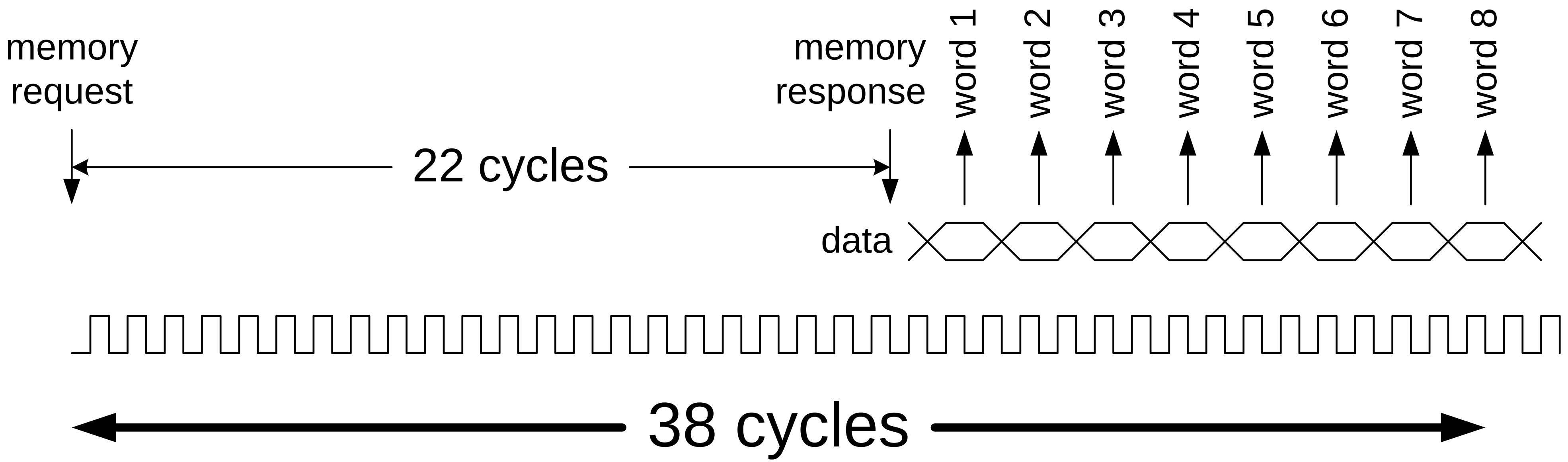
Mimick relevant real world characteristics



4B. Case: ARM9 Hardware Block Diagram

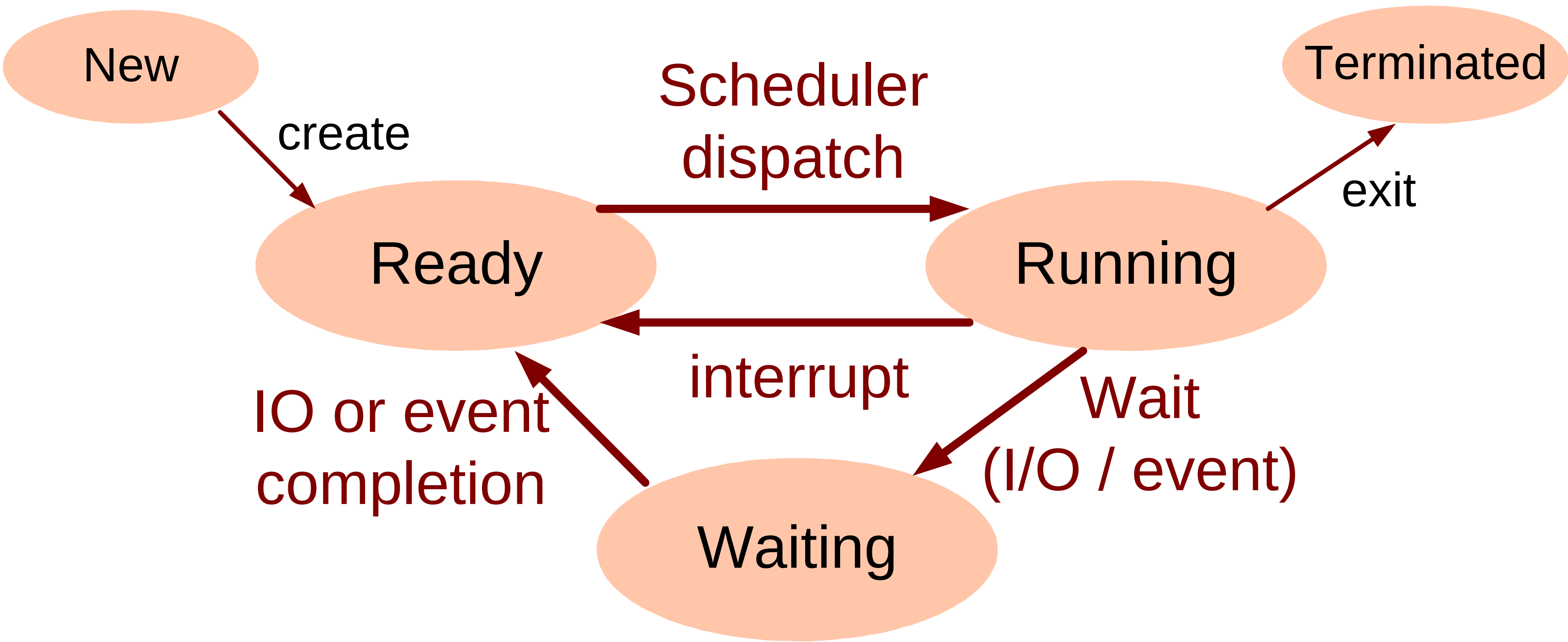


Key Hardware Performance Aspect



memory access time in case of a cache miss
200 Mhz, 5 ns cycle: 190 ns

OS Process Scheduling Concepts



Determine Expectation

simple SW model of context switch:

- save state P1
- determine next runnable task
- update scheduler administration
- load state P2
- run P2

Estimate how many **instructions** and **memory accesses** are needed per **context switch**

input data HW:

$t_{\text{ARM instruction}} = 5$ ns

$t_{\text{memory access}} = 190$ ns

Calculate the **estimated time** needed per **context switch**

Determine Expectation Quantified

instructions	memory accesses	simple SW model of context switch:
10	1	save state P1
50	2	determine next runnable task
20	1	update scheduler administration
10	1	load state P2
10	1 +	run P2
100	6	

Estimate how many instructions and memory accesses are needed per context switch

input data HW:

$t_{ARM\ instruction}$ = 5 ns

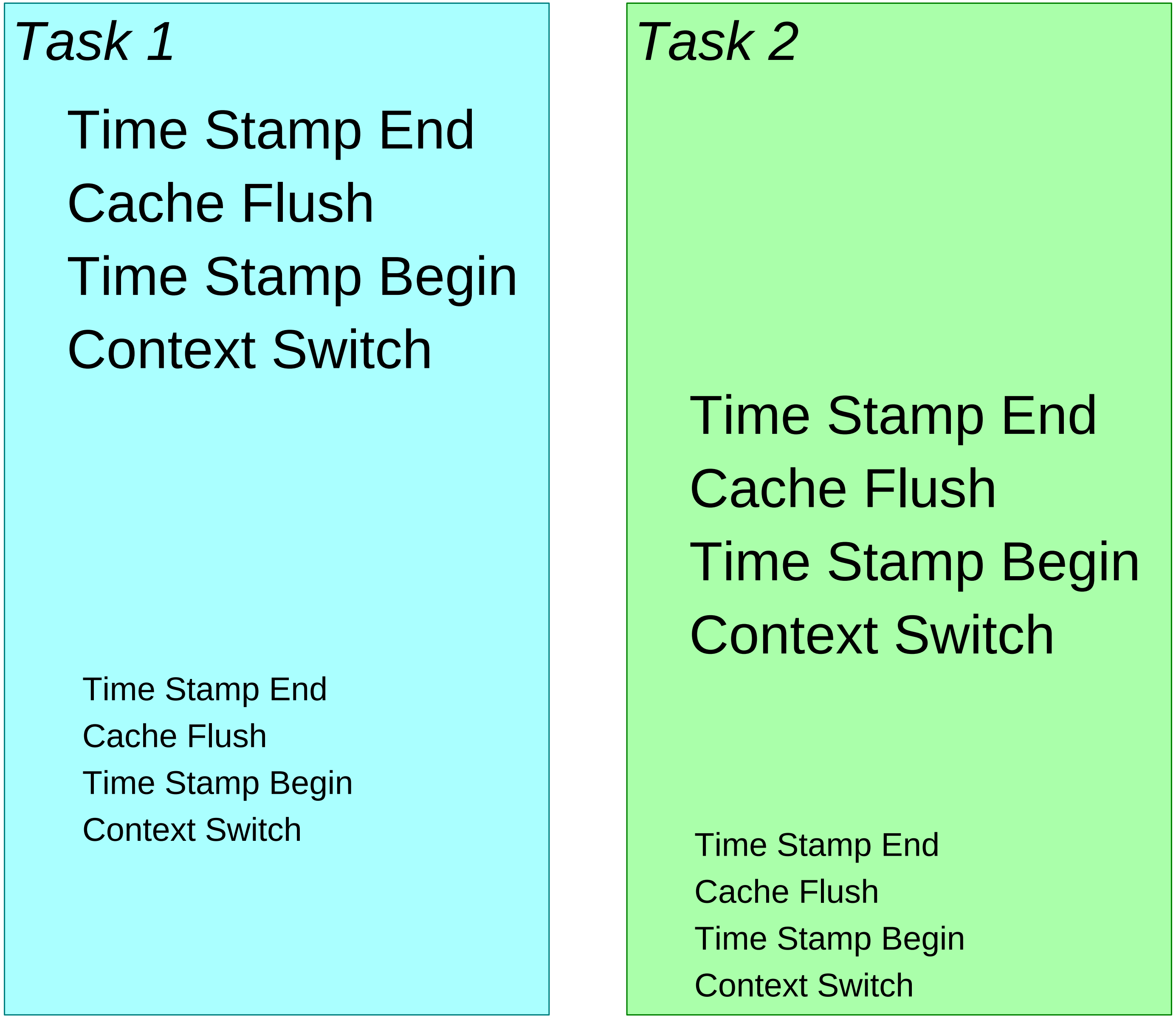
$t_{memory\ access}$ = 190 ns

500 ns
1140 ns +
1640 ns

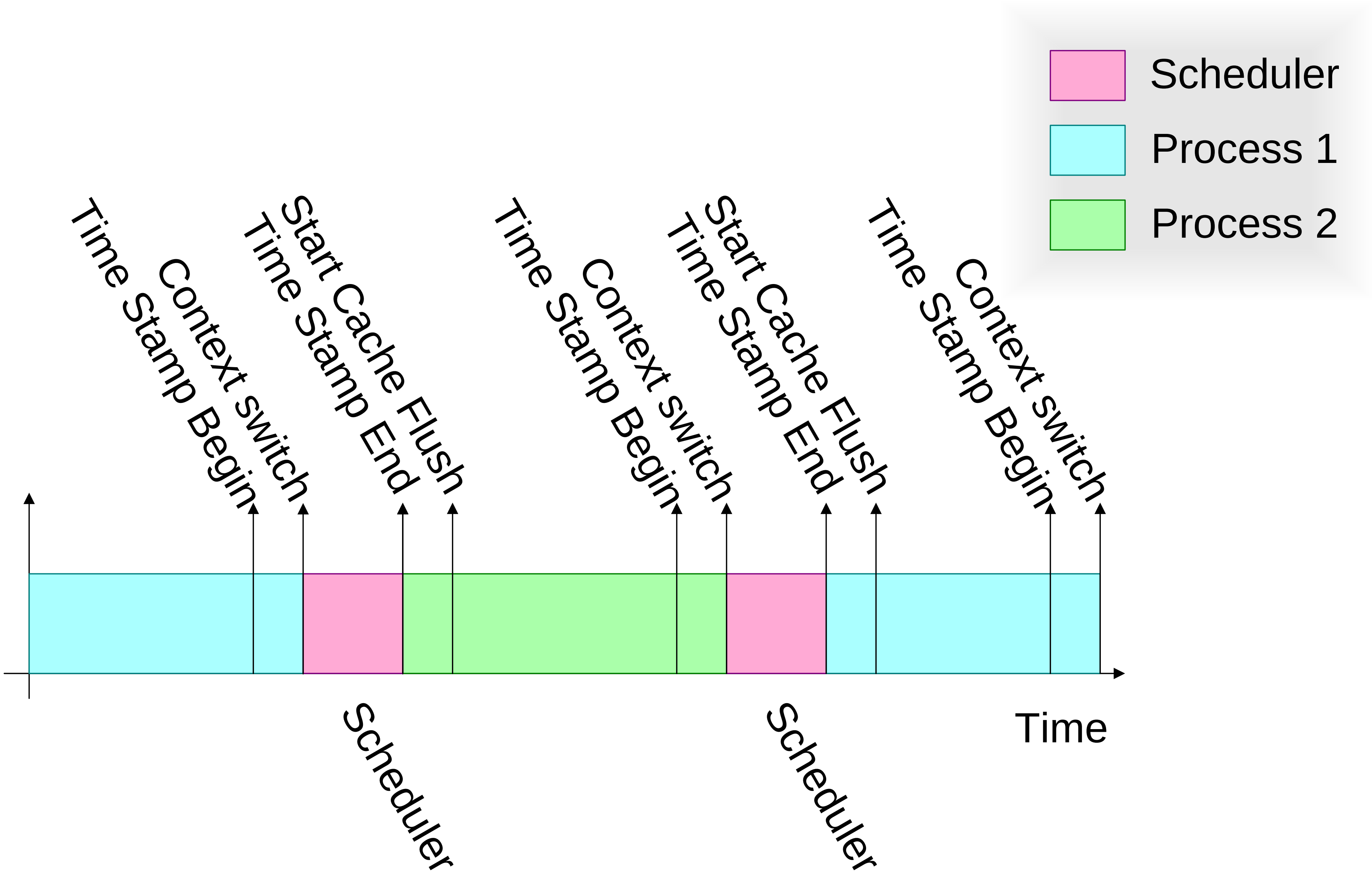
Calculate the estimated time needed per context switch

round up (as margin) gives expected $t_{context\ switch}$ = 2 µs

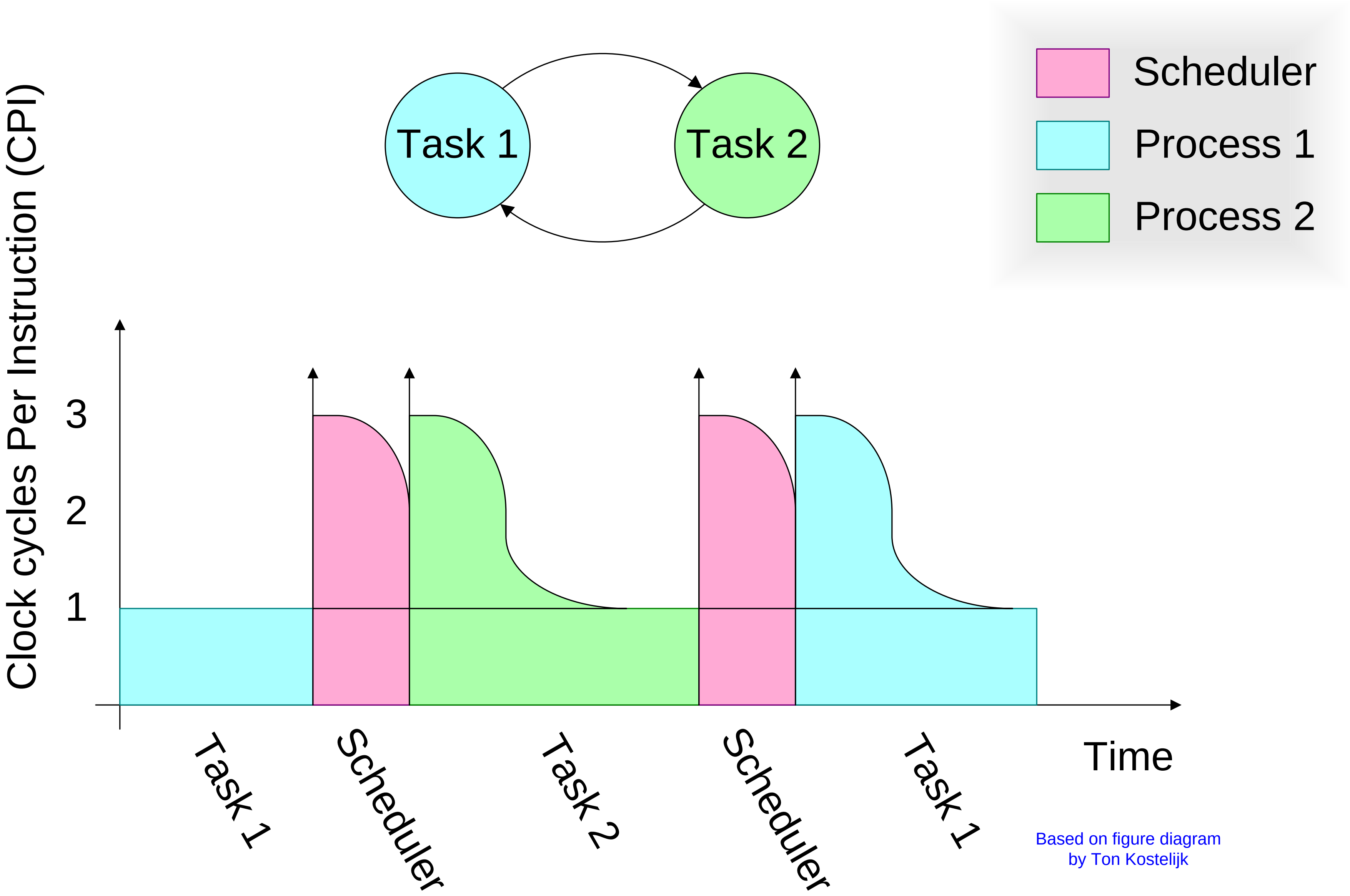
4C. Code to Measure Context Switch



Measuring Task Switch Time

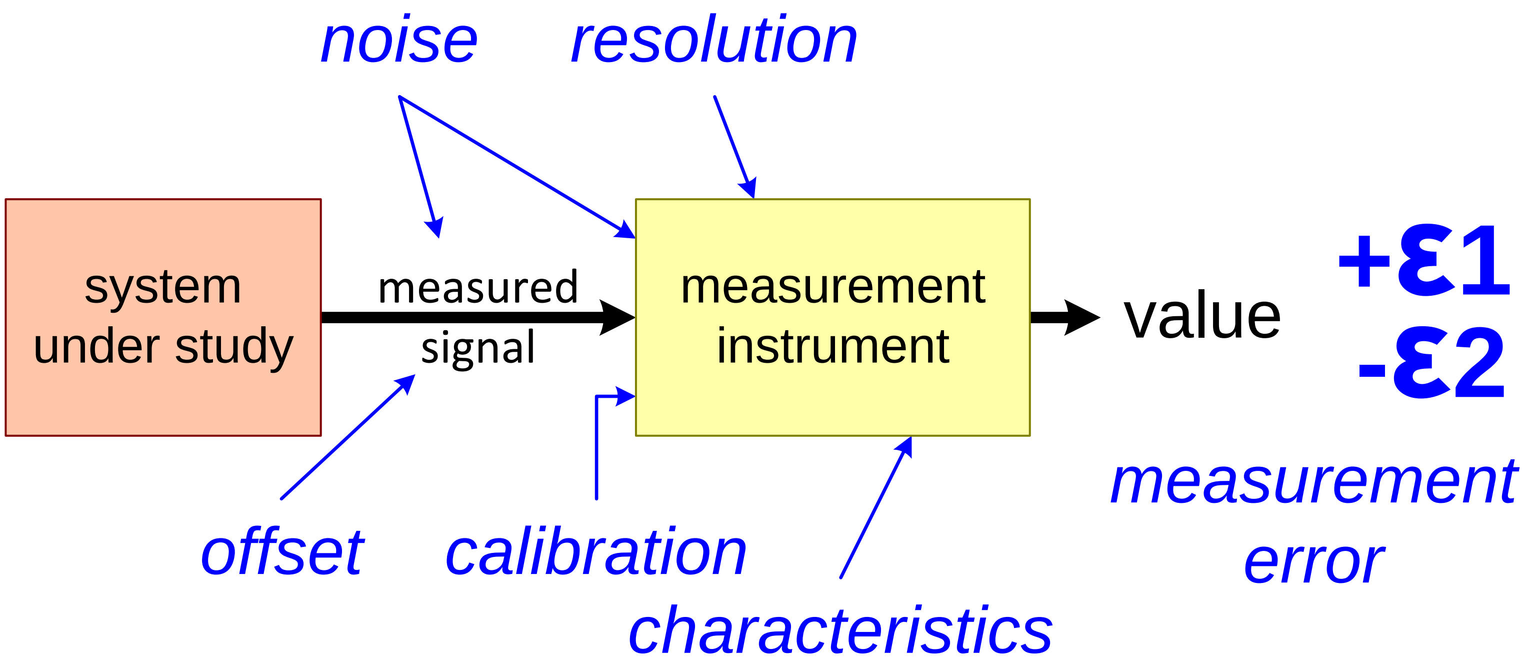


Understanding: Impact of Context Switch

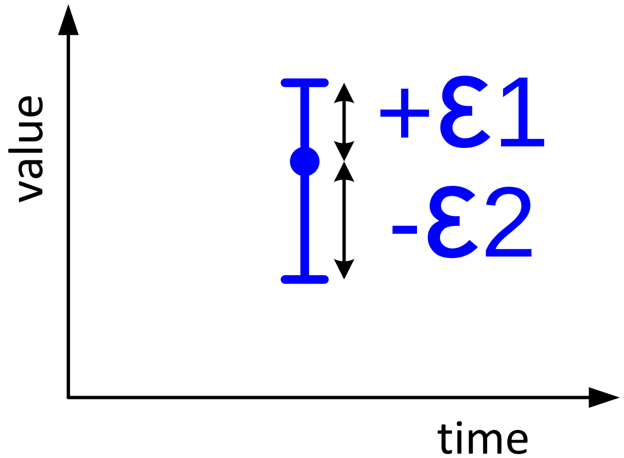


Based on figure diagram
by Ton Kostelijk

5. Accuracy: Measurement Error



measurements have stochastic variations and systematic deviations resulting in a range rather than a single value



Accuracy 2: Be Aware of Error Propagation

$$t_{duration} = t_{end} - t_{start}$$

$$t_{start} = 10 \text{ +/- } 2\ \mu s$$

$$t_{end} = 14 \text{ +/- } 2\ \mu s$$

$$t_{duration} = 4 \text{ +/- } ?\ \mu s$$

systematic errors: add linear

stochastic errors: add quadratic

Intermezzo Modeling Accuracy

Measurements have

stochastic variations and systematic deviations

resulting in a range rather than a single value.

The inputs of modeling,

"facts", assumptions, and measurement results,

also have stochastic variations and systematic deviations.

Stochastic variations and systematic deviations

propagate (add, amplify or cancel) through the model

resulting in an output range.

ARM9 200 MHz $t_{context\ switch}$ as function of cache use

cache setting	$t_{context\ switch}$
From cache	2 µs
After cache flush	10 µs
Cache disabled	50 µs

7. Expectation versus Measurement

	$t_{context\ switch}$
expected	$t_{context\ switch} = 2\ \mu s$
measured	$t_{context\ switch} = 10\ \mu s$

How to explain?

instructions	memory accesses	simple SW model of context switch:
10	1	save state P1
50	2	determine next runnable task
20	1	update scheduler administration
10	1	load state P2
10	1 +	run P2
100	6	

input data HW:

$t_{ARM\ instruction} = 5$ ns

$t_{memory\ access} = 190$ ns

500 ns

1140 ns +

1640 ns

potentially missing in expectation:

- memory accesses due to instructions
  - ~10 instruction memory accesses ~= 2 µs
- memory management (MMU context)
- complex process model (parents, permissions)
- bookkeeping, e.g performance data
- layering (function calls, stack handling)
- the combination of above issues

However, measurement seems to make sense

Conclusion Context Switch Overhead

$$t_{overhead} = n_{context\ switch} * t_{context\ switch}$$

$n_{context\ switch}$ (s^{-1})	$t_{context\ switch} = 10\mu s$		$t_{context\ switch} = 2\mu s$	
	$t_{overhead}$	CPU load overhead	$t_{overhead}$	CPU load overhead
500	5ms	0.5%	1ms	0.1%
5000	50ms	5%	10ms	1%
50000	500ms	50%	100ms	10%

Summary Context Switching on ARM9

goal of measurement

Guidance of concurrency design and task granularity

Estimation of context switching overhead

Cost of context switch on given platform

examples of measurement

Needed: context switch overhead ~10% accurate

Measurement instrumentation: HW pin and small SW test program

Simple models of HW and SW layers

Measurement results for context switching on ARM9

Summary Measuring Approach

Conclusions

Measurements are an important source of factual data.

A measurement requires a well-designed experiment.

Measurement error, validation of the result determine the credibility.

Lots of consolidated data must be reduced to essential understanding.

Techniques, Models, Heuristics of this module

experimentation

error analysis

estimating expectations

Colophon

This work is derived from the EXARCH course at CTT developed by *Ton Kostelijk* (Philips) and *Gerrit Muller*.

The Boderc project contributed to the measurement approach. Especially the work of

Peter van den Bosch (Océ),

Oana Florescu (TU/e),

and *Marcel Verhoef* (Chess)

has been valuable.

Home work

Introductory discussion

Formula Based Performance Design

by *Gerrit Muller* HSN-NISE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

Abstract

Performance models are mostly simple mathematical formulas. The challenge is to model the performance at an appropriate level. In this presentation we introduce several levels of modeling, labeled zeroth order, second order, et cetera. AS illiustration we use the performance of MRI reconstruction.

March 6, 2021
status: draft
version: 1.0

Theory Block: n Order Formulas

0th order — main function parameters — *order of magnitude*
relevant for main function

1st order — add overhead secondary function(s) — *estimation*

2nd order — interference effects circumstances
main function, overhead and/or secondary functions
more accurate, understanding

CPU Time Formula Zero Order

$$t_{cpu\ total} = t_{cpu\ processing} + t_{UI}$$

$$t_{cpu\ processing} = n_x * n_y * t_{pixel}$$

CPU Time Formula First Order

$$t_{\text{cpu total}} = t_{\text{cpu processing}} + t_{\text{UI}}$$

$$+ \; t_{\text{context switch overhead}}$$

CPU Time Formula Second Order

$$t_{\text{cpu total}} = t_{\text{cpu processing}} + t_{\text{UI}} + t_{\text{context switch overhead}} + t_{\text{stall time due to cache efficiency}} + t_{\text{stall time due to context switching}}$$

signal processing: high efficiency
control processing: low/medium efficiency

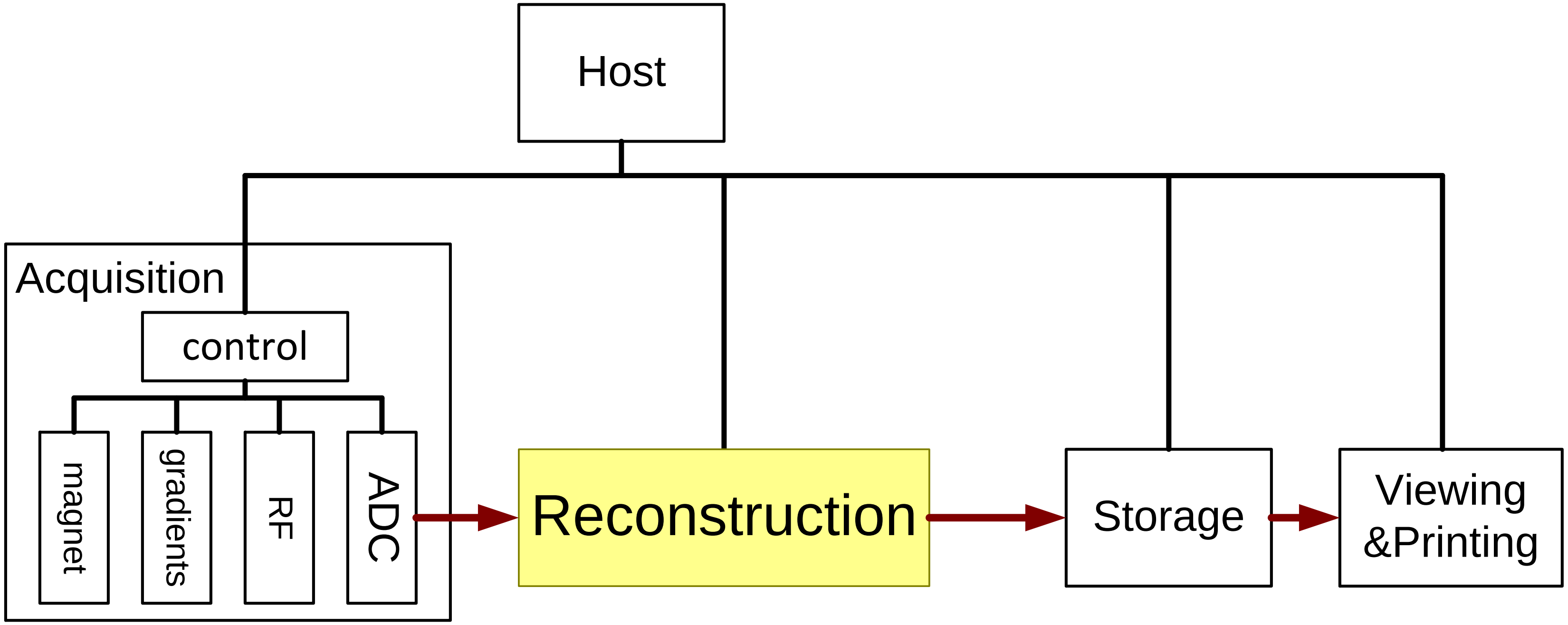
Case MRI Reconstruction

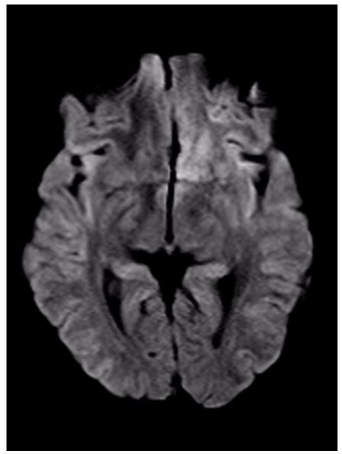
MRI reconstruction

"Test" of performance model on another case

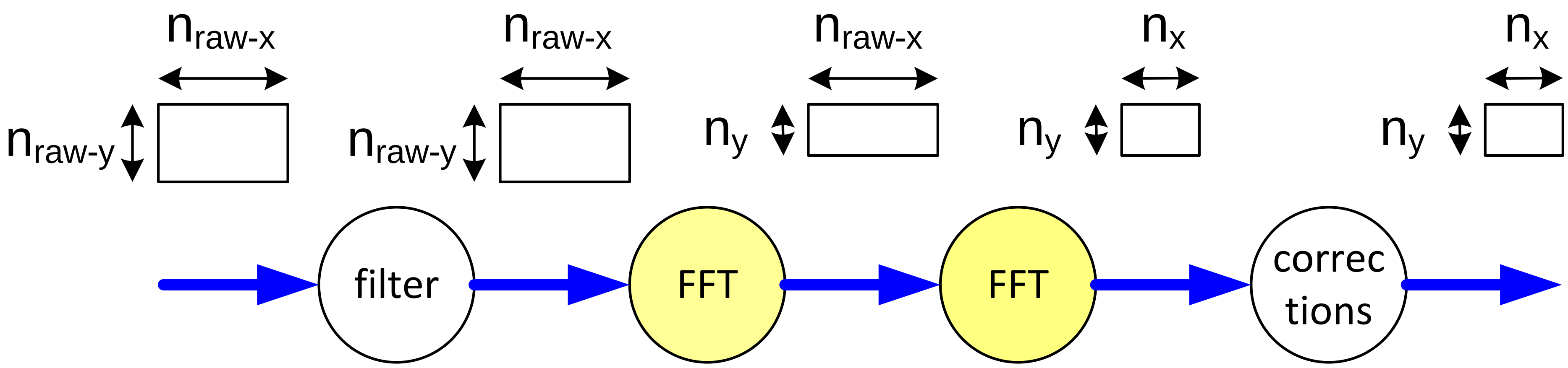
Scope of performance and significance of impact

MR Reconstruction Context





MR Reconstruction Performance Zero Order



$$t_{recon} = n_{raw-x} * t_{fft}(n_{raw-y}) + n_y * t_{fft}(n_{raw-x})$$

$$t_{fft}(n) = c_{fft} * n * \log(n)$$

Typical FFT, 1k points ~ 5 msec

(scales with 2 * n * log (n))

using:

$n_{raw\text{-}x} = 512$

$n_{raw\text{-}y} = 256$

$n_x = 256$

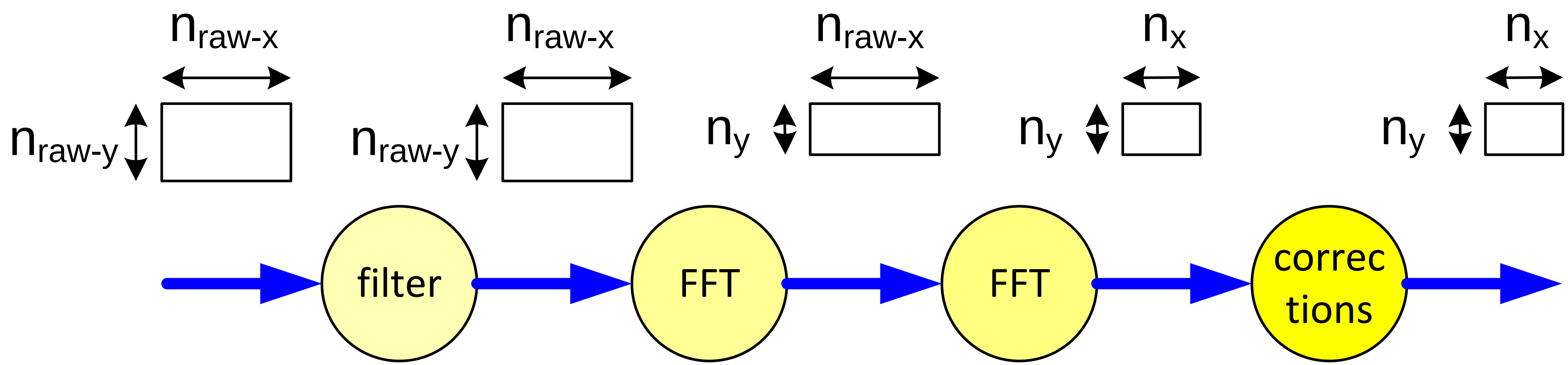
$n_y = 256$

$$t_{recon} = n_{raw\text{-}x} * t_{fft}(n_{raw\text{-}y}) + n_y * t_{fft}(n_{raw\text{-}x}) +$$

512 * 1.2 + 256 * 2.4

~= 1.2 s

MR Reconstruction Performance First Order



$$t_{recon} = t_{filter}(n_{raw-x}, n_{raw-y}) + n_{raw-x} * t_{fft}(n_{raw-y}) + n_y * t_{fft}(n_{raw-x}) + t_{corrections}(n_x, n_y)$$

$$t_{fft}(n) = c_{fft} * n * \log(n)$$

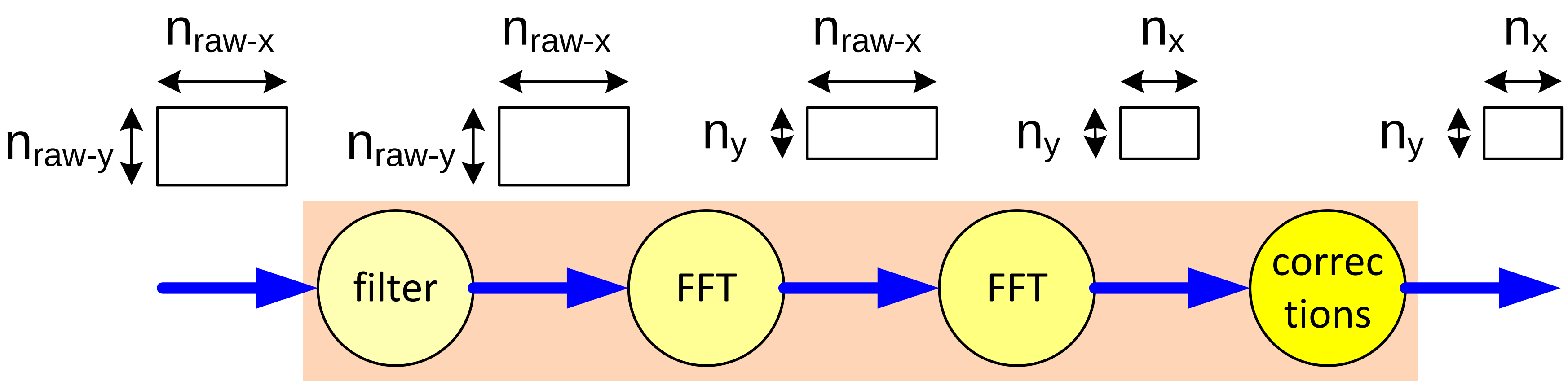
First Order Quantitative Example

Typical FFT, 1k points ~ 5 msec
(scales with 2 * n * log (n))

Filter 1k points ~ 2 msec
(scales linearly with n)

Correction ~ 2 msec
(scales linearly with n)

MR Reconstruction Performance Second Order



$$
\begin{aligned}
t_{recon} = \; & t_{filter}(n_{raw\text{-}x}, n_{raw\text{-}y}) \; + \\
& n_{raw\text{-}x} * (\, t_{fft}(n_{raw\text{-}y}) \; + \; t_{col\text{-}overhead} \,) \; + \\
& n_y * (\, t_{fft}(n_{raw\text{-}x}) \; + \; t_{row\text{-}overhead} \,) \; + \\
& t_{corrections}(n_x, n_y) \; + \\
& t_{control\text{-}overhead}
\end{aligned}
$$

$$t_{fft}(n) = c_{fft} * n * \log(n)$$

Second Order Quantitative Example

Typical FFT, 1k points ~ 5 msec
(scales with 2 * n * log (n))

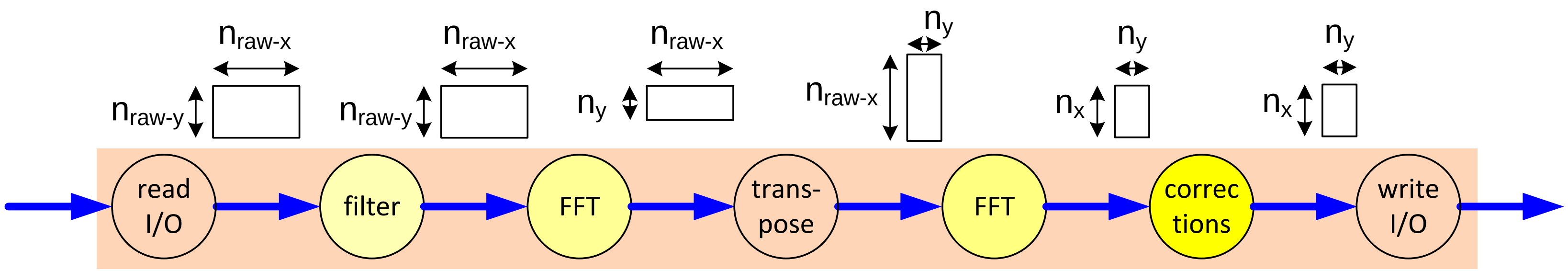
Filter 1k points ~ 2 msec
(scales linearly with n)

Correction ~ 2 msec
(scales linearly with n)

Control overhead = $n_y * t_{row\ overhead}$

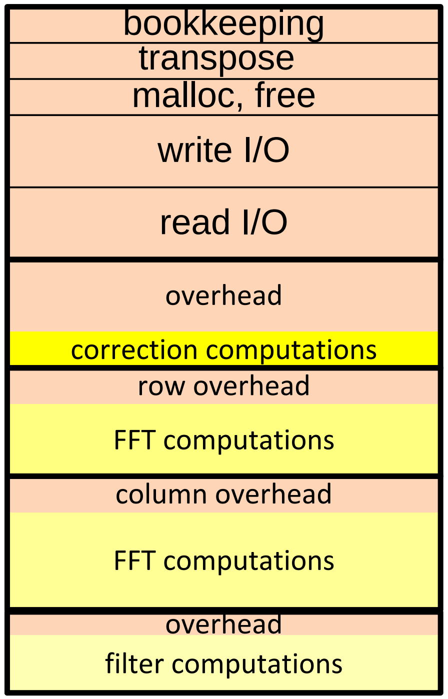
10 .. 100 µs

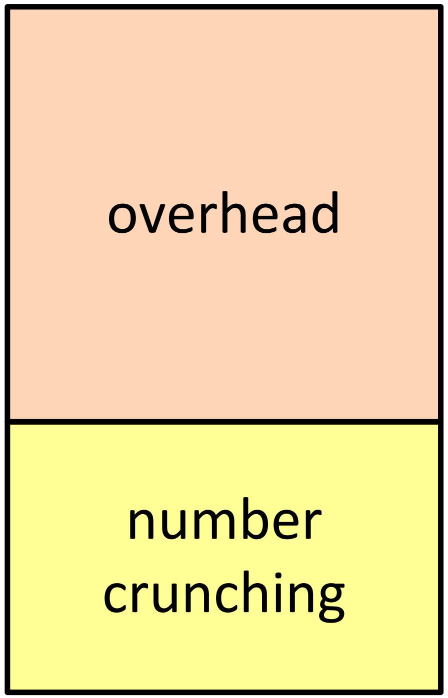
MR Reconstruction Performance Third Order



$$t_{recon} = t_{filter}(n_{raw-x}, n_{raw-y}) + n_{raw-x} * (t_{fft}(n_{raw-y}) + t_{col-overhead}) +$$
$$n_y * (t_{fft}(n_{raw-x}) + t_{row-overhead}) + t_{corrections}(n_x, n_y) + t_{read\ I/O} + t_{transpose} + t_{write\ I/O} + t_{control-overhead}$$

$$t_{fft}(n) = c_{fft} * n * \log(n)$$





focus on overhead reduction

is more important

than faster algorithms

this is not an excuse for sloppy algorithms

Summary Case MRI Reconstruction

MRI reconstruction

System performance may be determined by other than standard facts

E.g. more by overhead I/O rather than optimized core processing

==> Identify & measure what is performance-critical in application

Soft Real Time Design

by *Gerrit Muller* HSN-NISE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

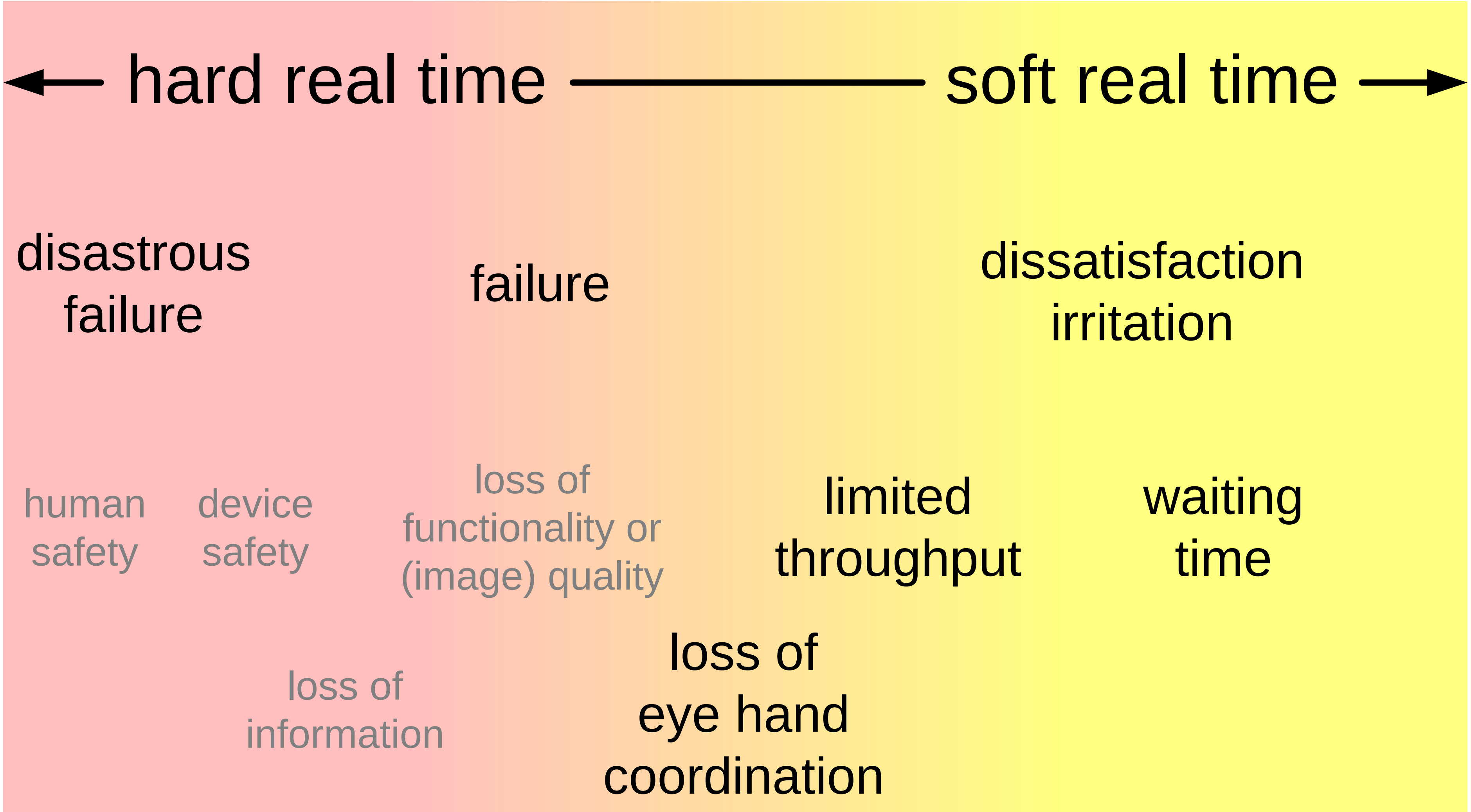
Abstract

Soft Real Time design addresses the performance aspects of the system design, under the assumption that the hard real time design is already well-covered. Core decisions in soft real time design are:

- granularity
- synchronization
- prioritization
- allocation
- resource management

March 6, 2021
status: preliminary draft
version: 0.2

Soft Real Time Design



Case 1

TV zapping

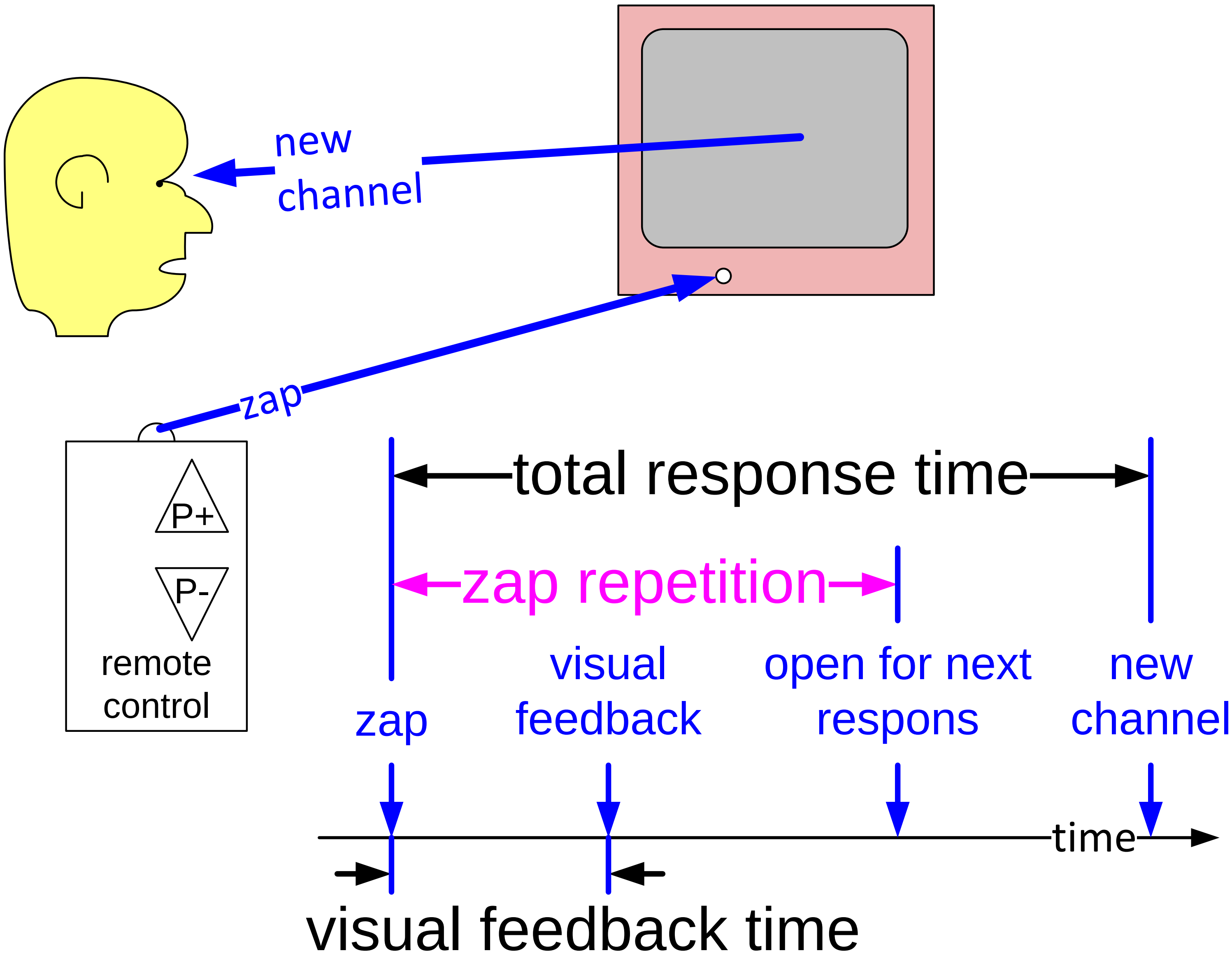
Problem introduction

Approach for solving response time problems

Revised functional model

Measuring and modelling

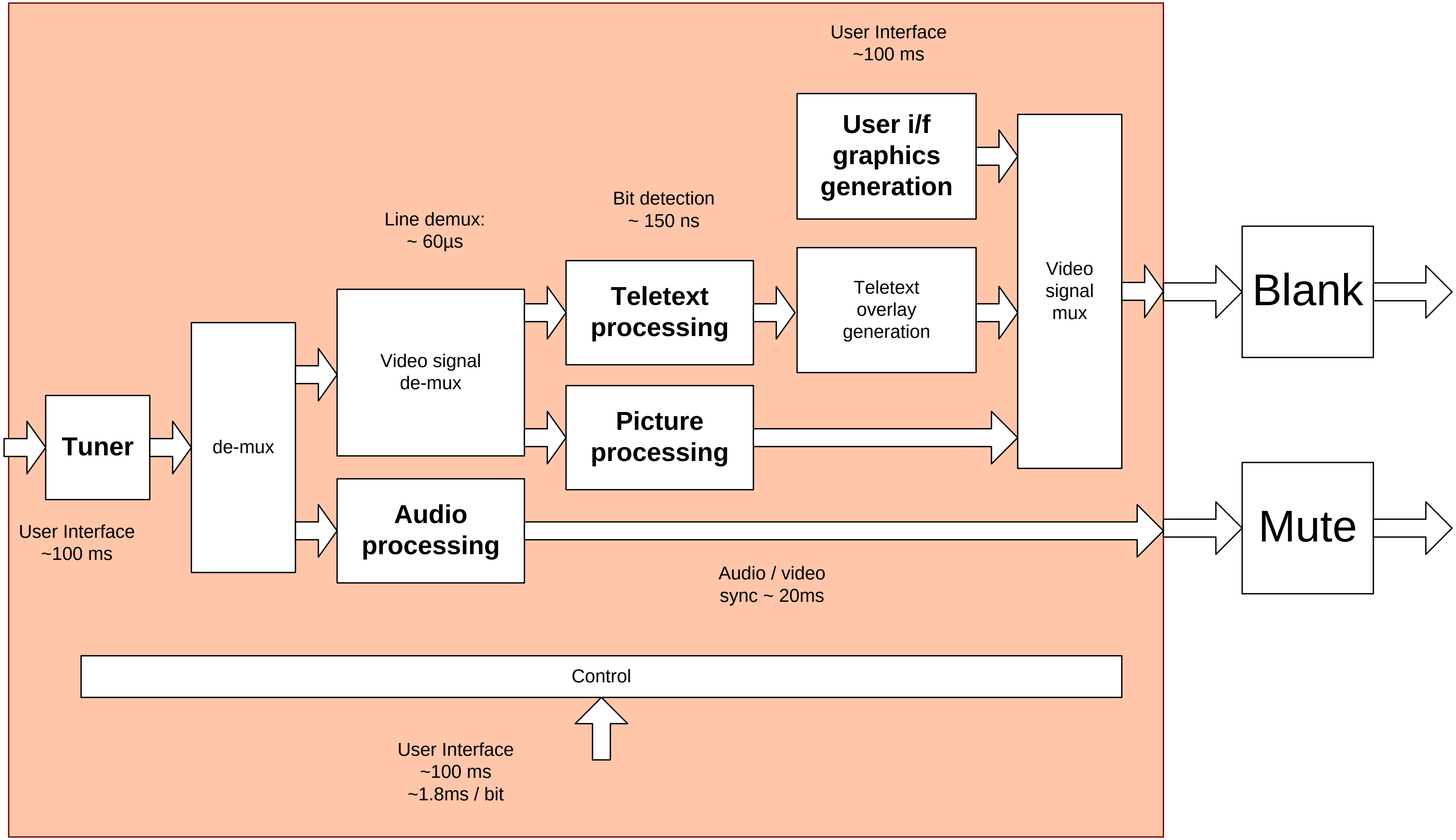
Zap timing: What is the Requirement?



Approach

1) Measure the end-to-end time
2) Decompose the processes based on expected outcome
3) Measure the individual components use previous decomposition (2)
4) Clarify the unknown parts and make them explicit
5) Further divide the major posts
6) Aggregate the smaller posts

Functional Model



Expectated and Measured Values

Expected values:

Mute	: 50 ms
Blank	: 40 ms
Flush AV pipeline	: 160 ms
Set tuner	: 200 ms
Fill AV pipeline	: 160 ms
Unmute	: 50 ms
Unblank	: 40 ms

1 frame

4 frames

4 frames

Measured values:

Mute	: 60 ms
Blank	: 120 ms
Flush AV pipeline	: 0 ms
Set tuner	: 180 ms
Fill AV pipeline	: 40 ms
Format detection	: 200 ms
Unmute	: 60 ms
Unblank	: 120 ms
Summing	: ~ 900 ms
Total time measured	: 2000 ms

1 frame

5 frames

Zapping Problem step 4

Somewhere 1000 ms are missing

Detection of frame size takes a long time!
+ Lots of software overhead
Analyze frame size detection and SW overhead

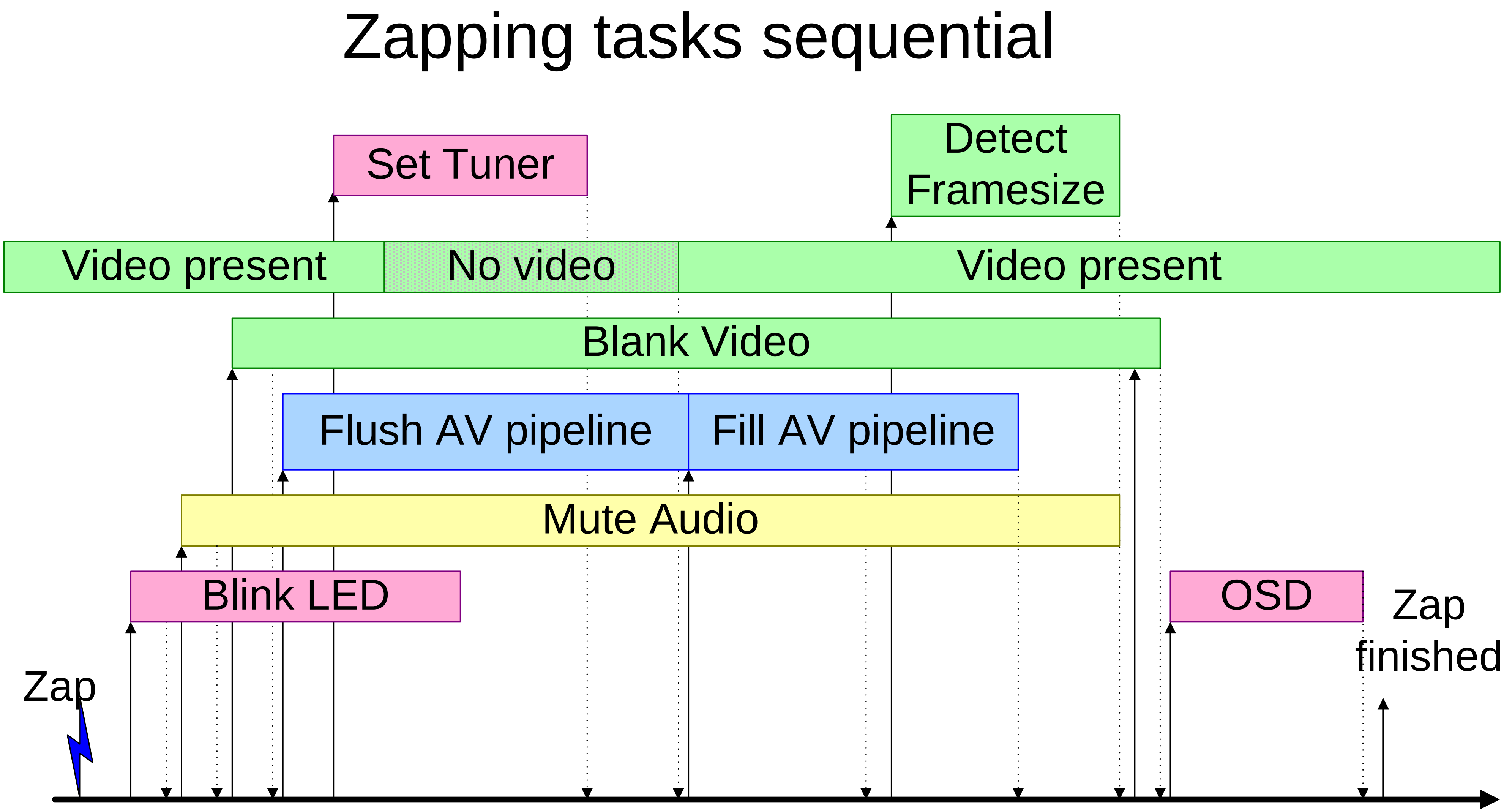
Zapping Problem step 5

Subdivide / analyze format detection (200 ms)

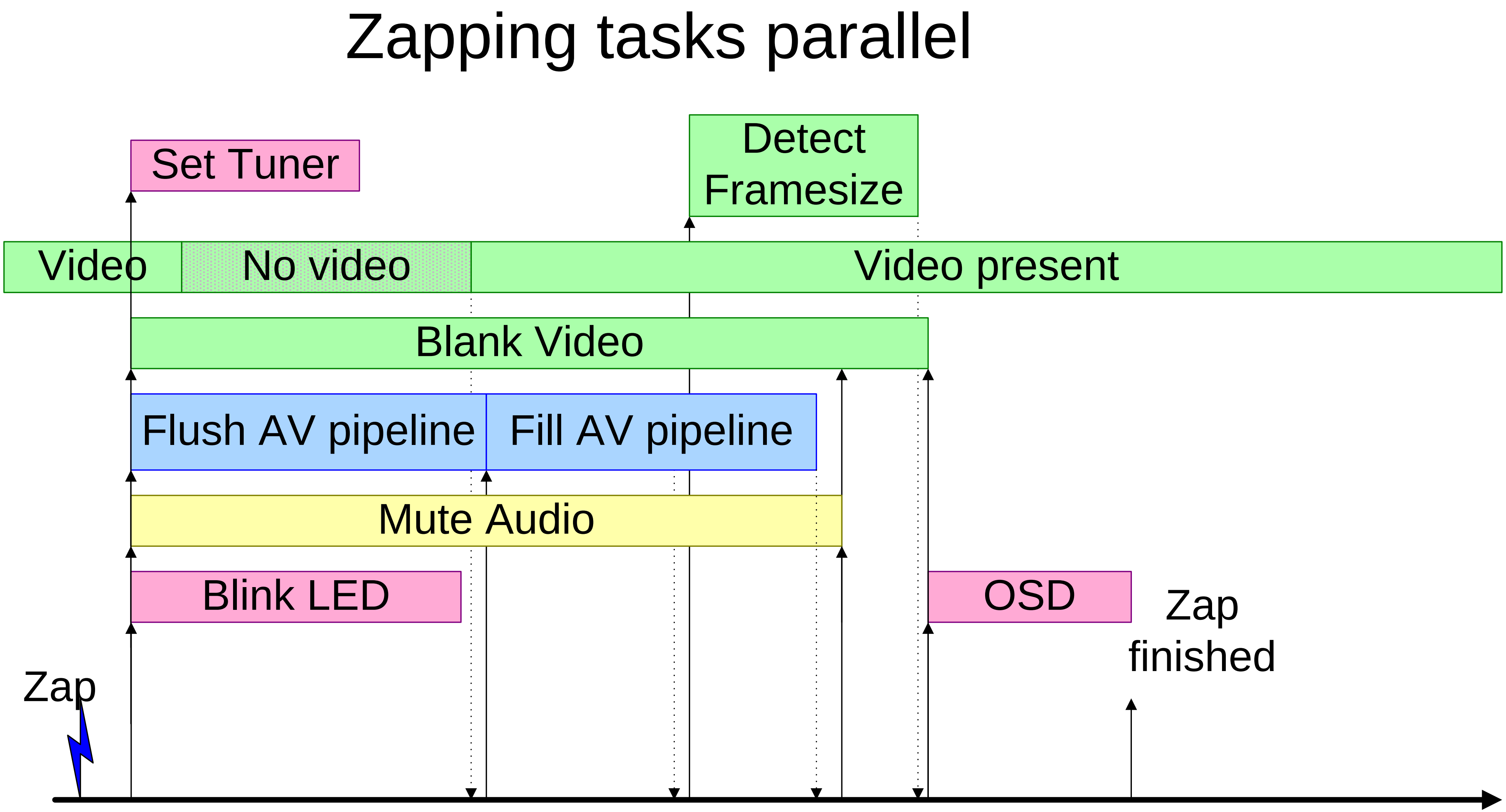
Zapping Problem step 6

Ignore pipeline effects

Simple Concurrency Model (with waits)



Simple Concurrency Model (optimized)



Case 1 Summary

TV zapping

Understanding of the problem is crucial

Iterate over modelling and measuring to build balanced performance model

EasyVision: Resource Management

Introduction to application

SW design

Memory and performance

Memory design

CPU load and Performance

Introduction to Medical Imaging Application

Easyvision

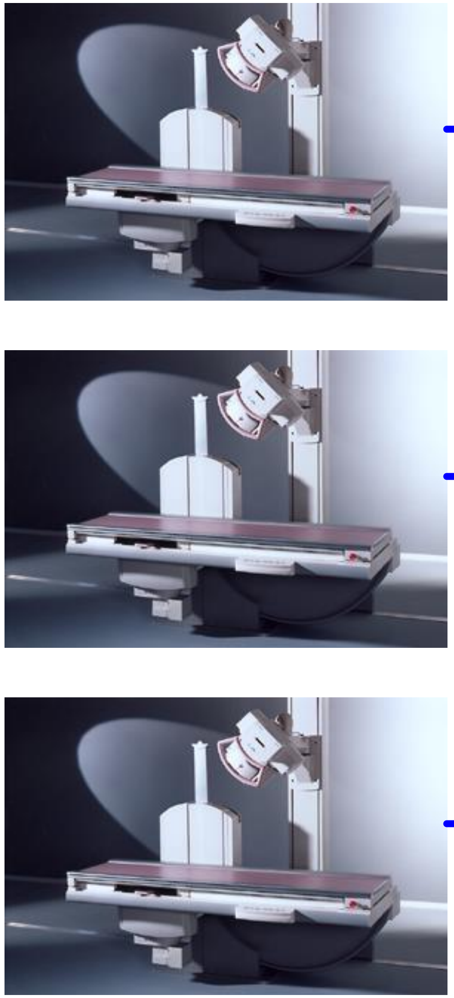
Medical Imaging Workstation

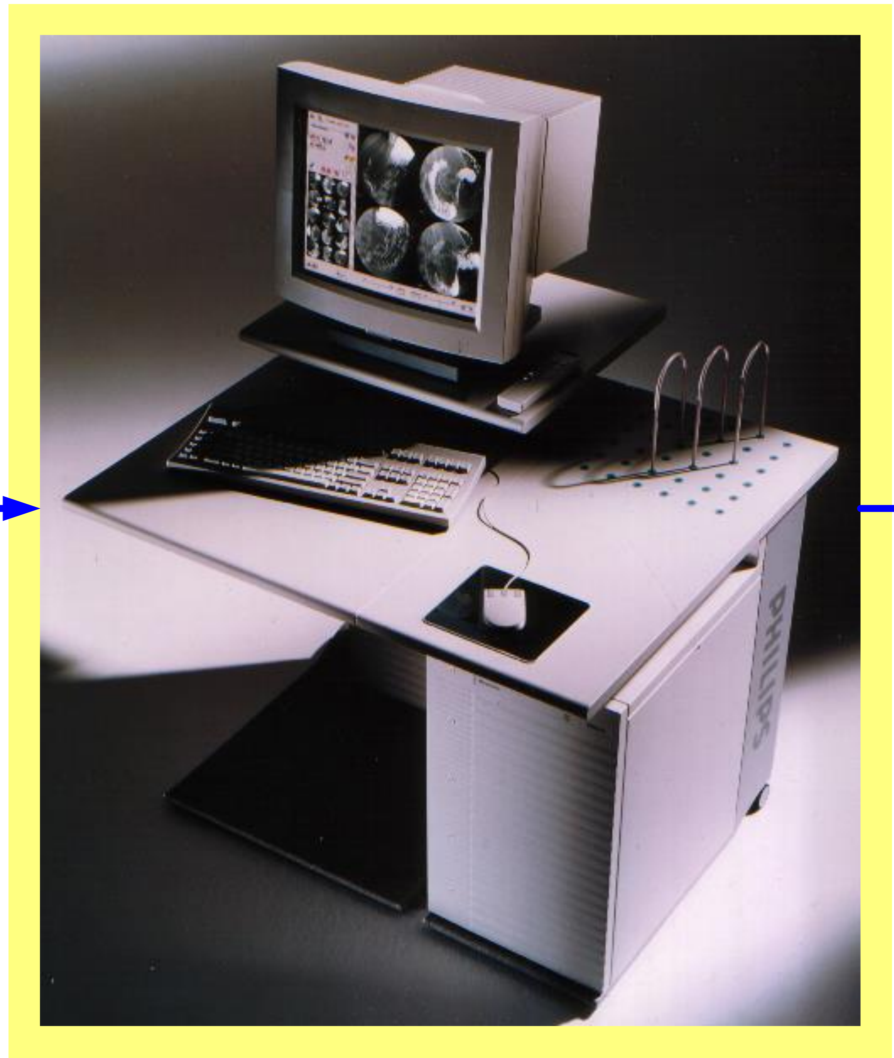
serving 3 X-ray examination rooms

providing interactive viewing and printing on high resolution film

Challenge: interoperability and WYSIWYG over different products

Easyvision Serving Three URF Examination Rooms



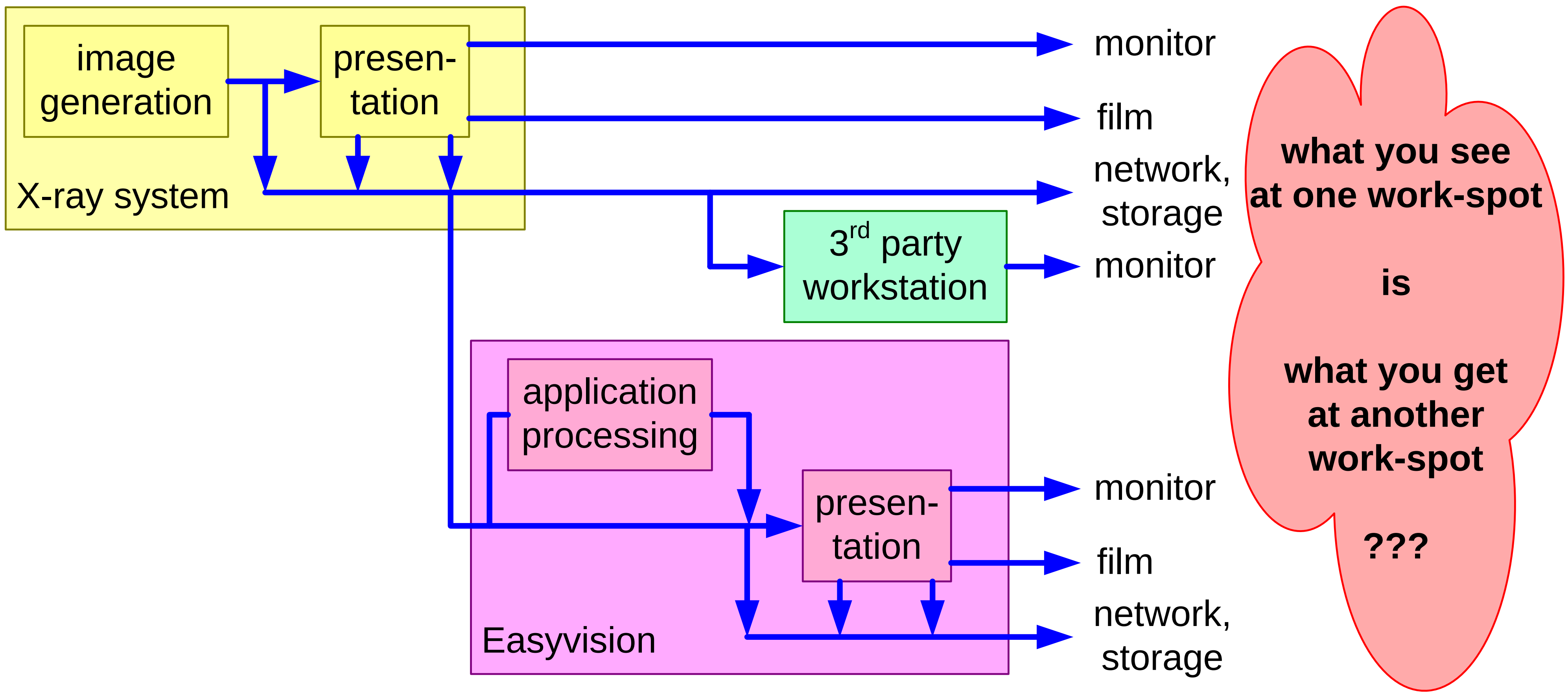


typical clinical image (intestines)

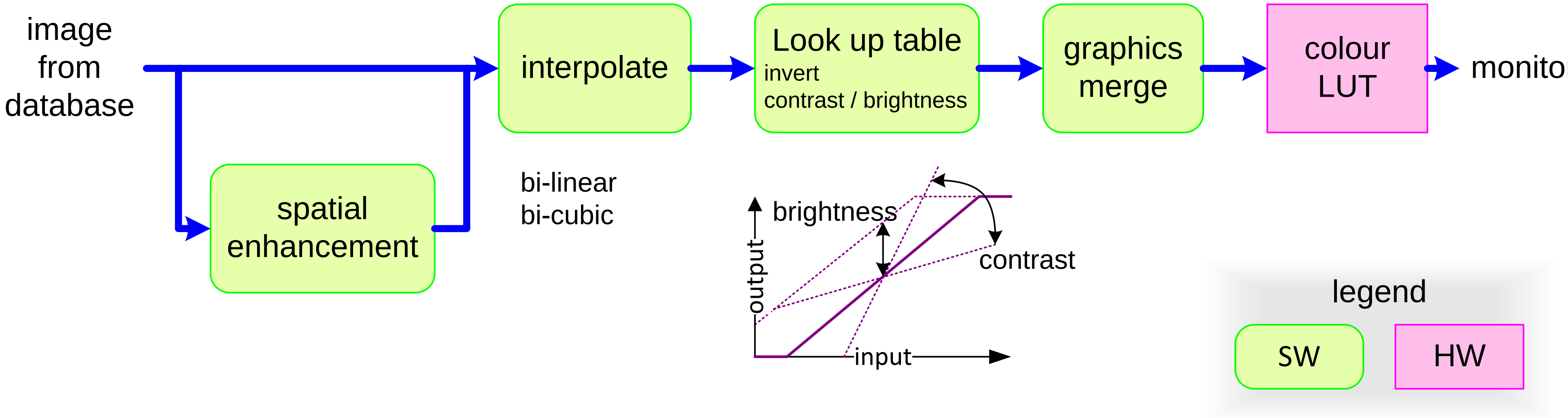
URF-systems

EasyVision: Medical Imaging Workstation

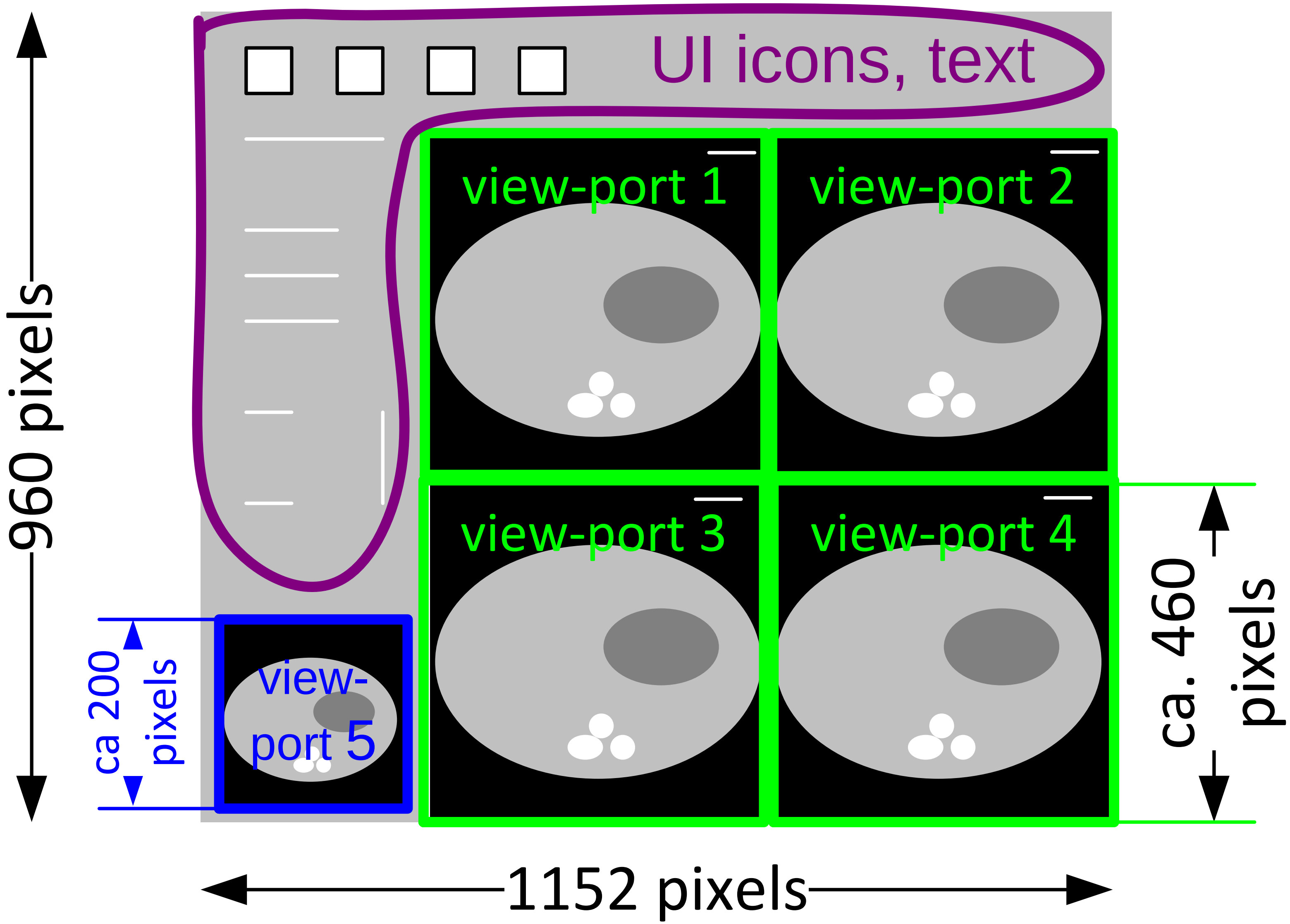
Image Quality Expectation WYSIWYG



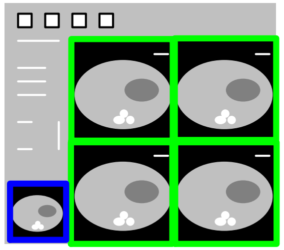
Presentation Pipeline for X-ray Images



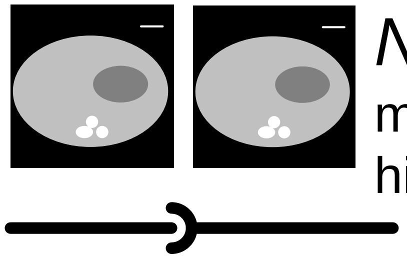
Quadruple View-port Screen Layout



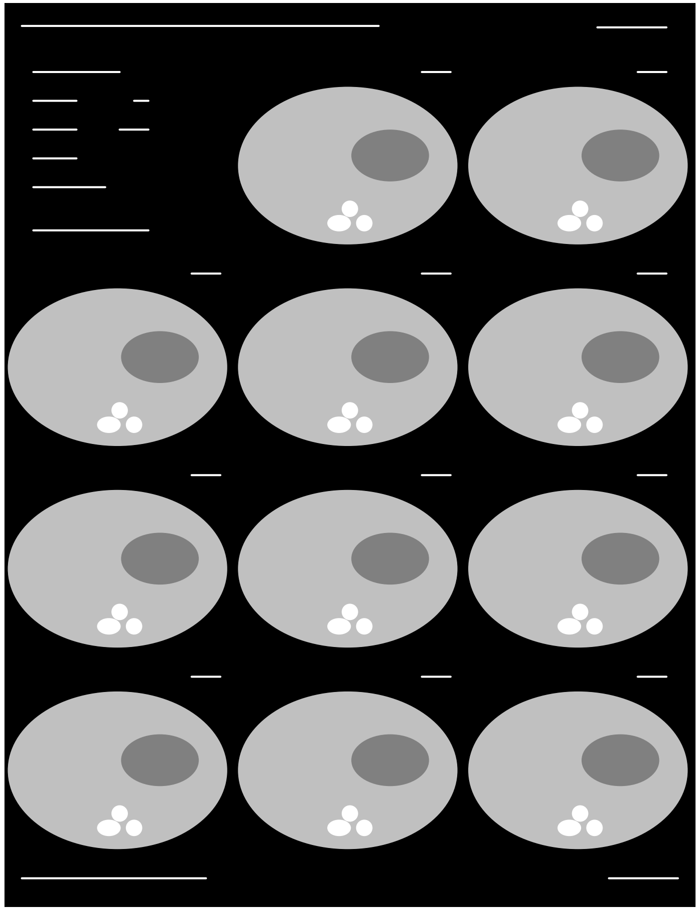
Rendered Images at Different Destinations



Screen:
low resolution
fast response



Network:
medium resolution
high throughput



Film:
high resolution
high throughput

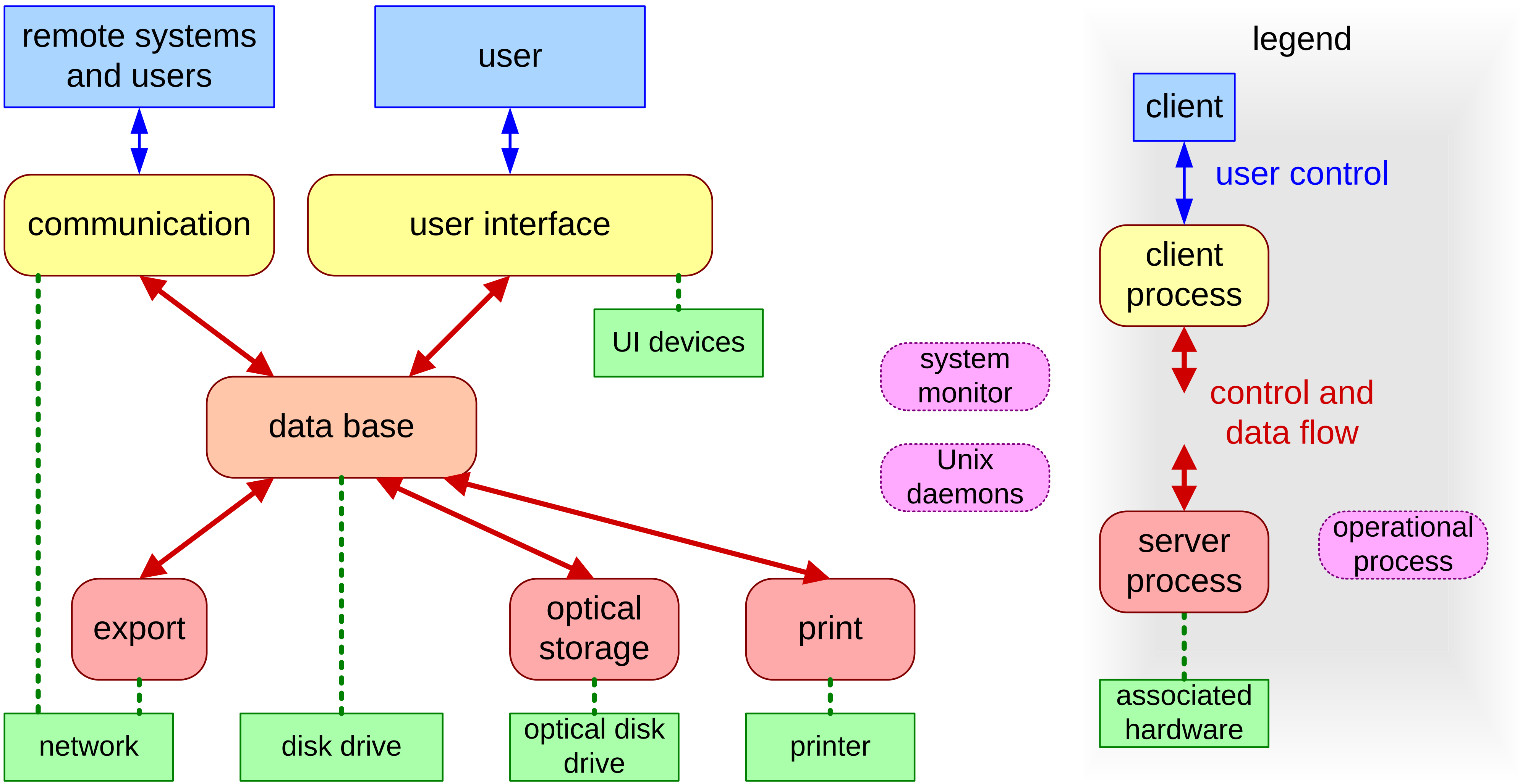
SW Design

Easyvision SW design

Concurrency design

SW layers

Concurrency via Software Processes

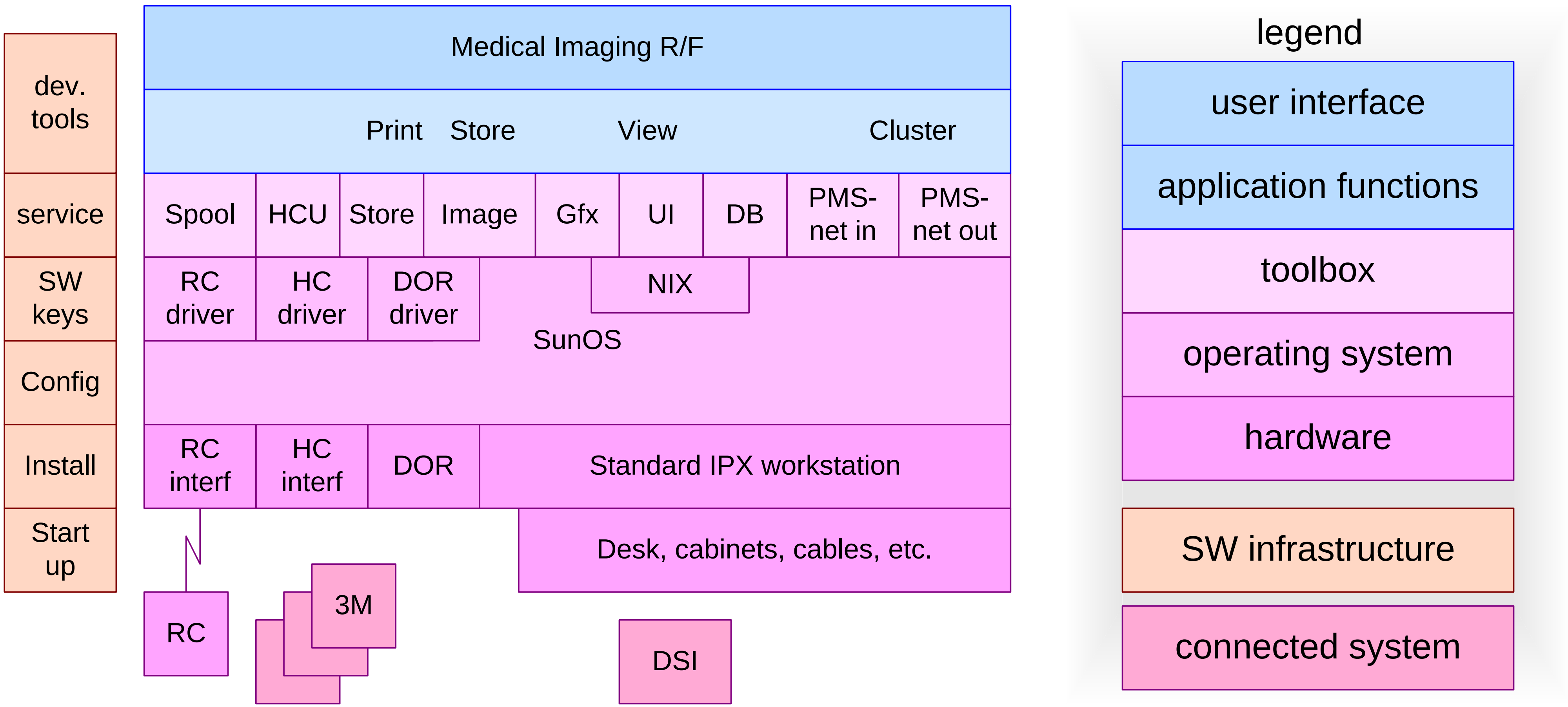


Criteria for Process Decomposition

- management of concurrency
- management of shared devices
- unit of memory budget (easy measurement)
- enables distribution over multiple processors
- unit of exception handling: fault containment and watchdog monitor

Processes are a facility provided by the Operating System (OS) to manage concurrency, resources and exceptions

Simplified Layering of the SW (Construction Decomposition)



Memory Use and Performance

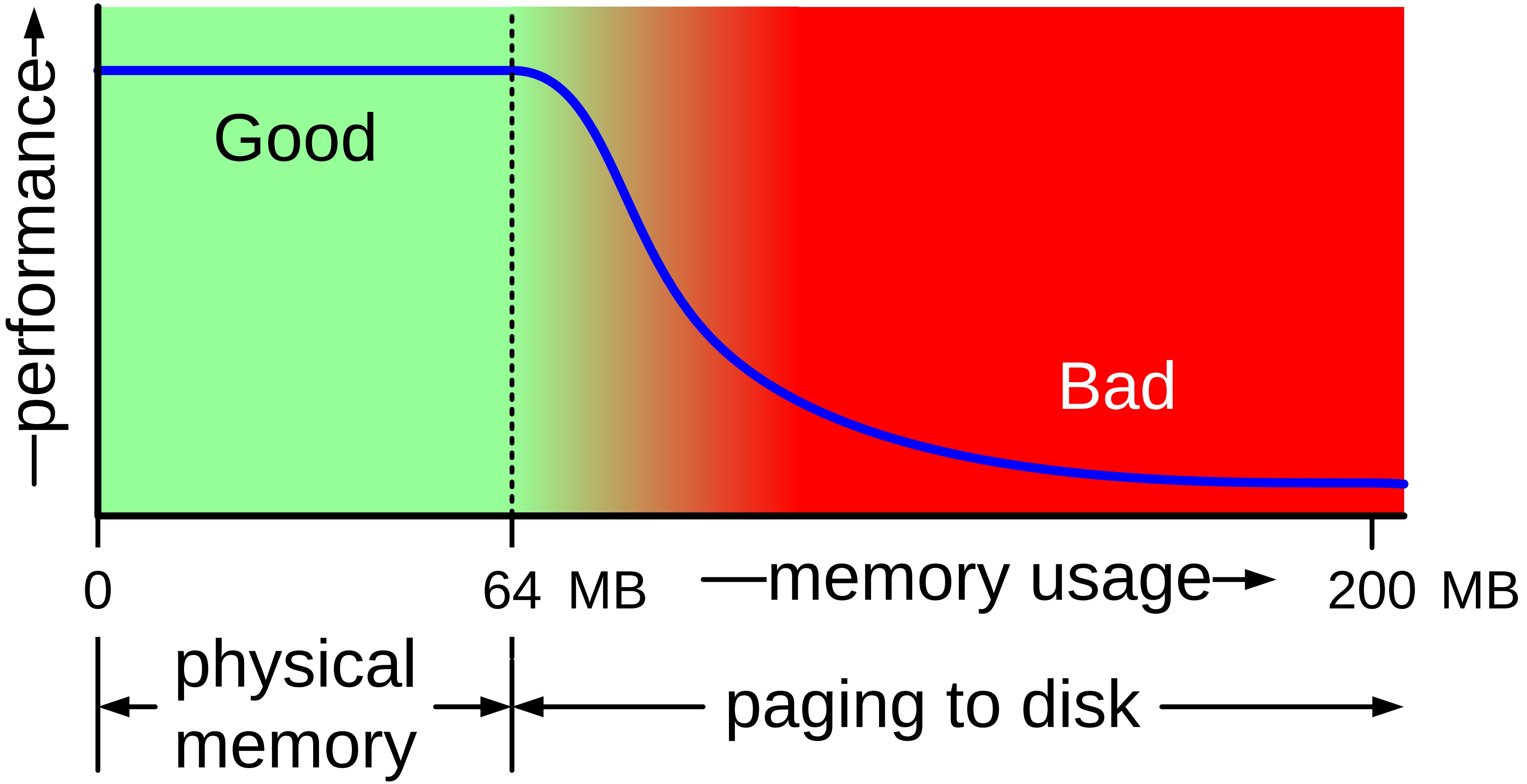
Easyvision Memory and Performance

Performance problems

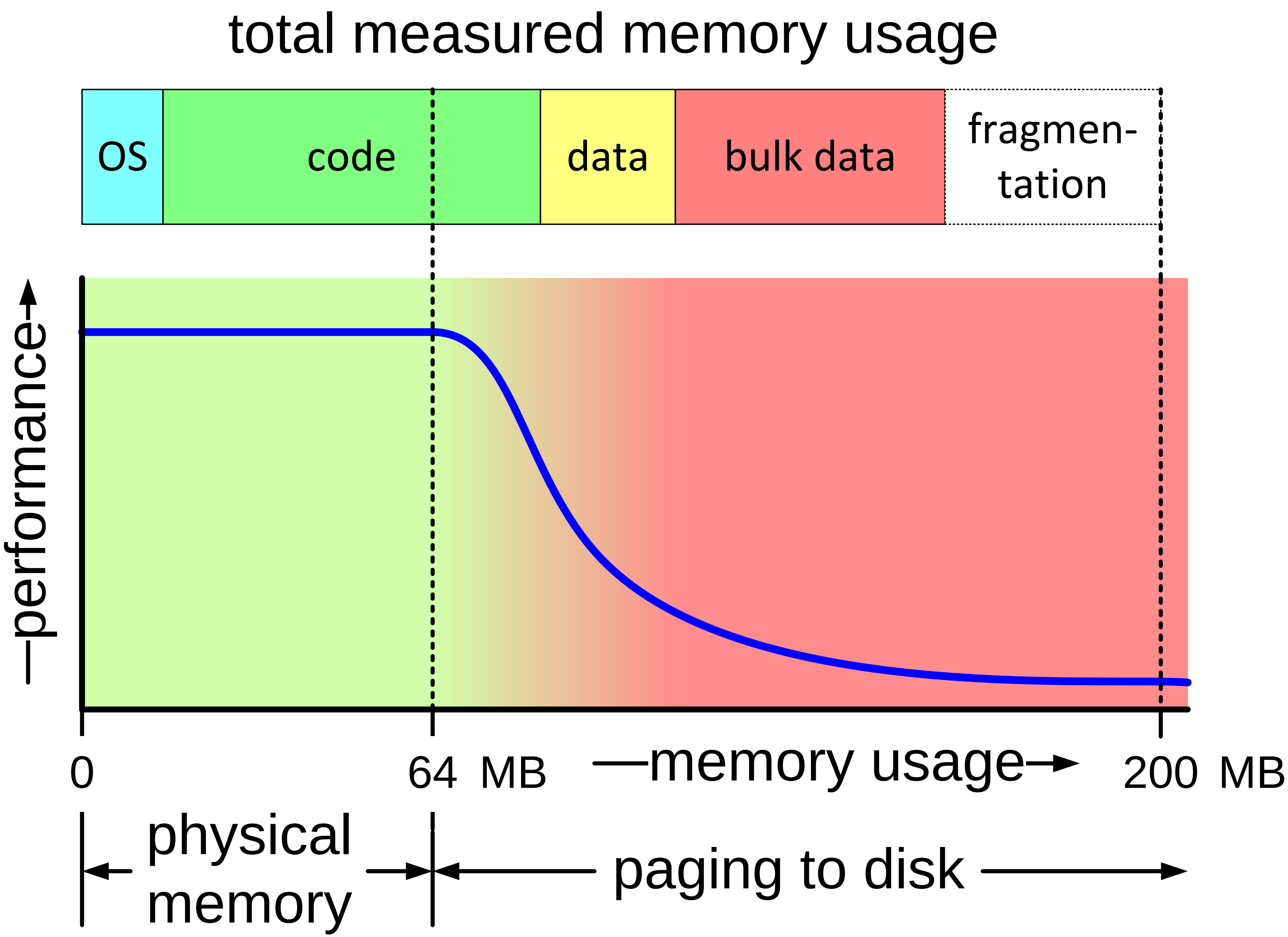
Analysis of memory use

Memory budget

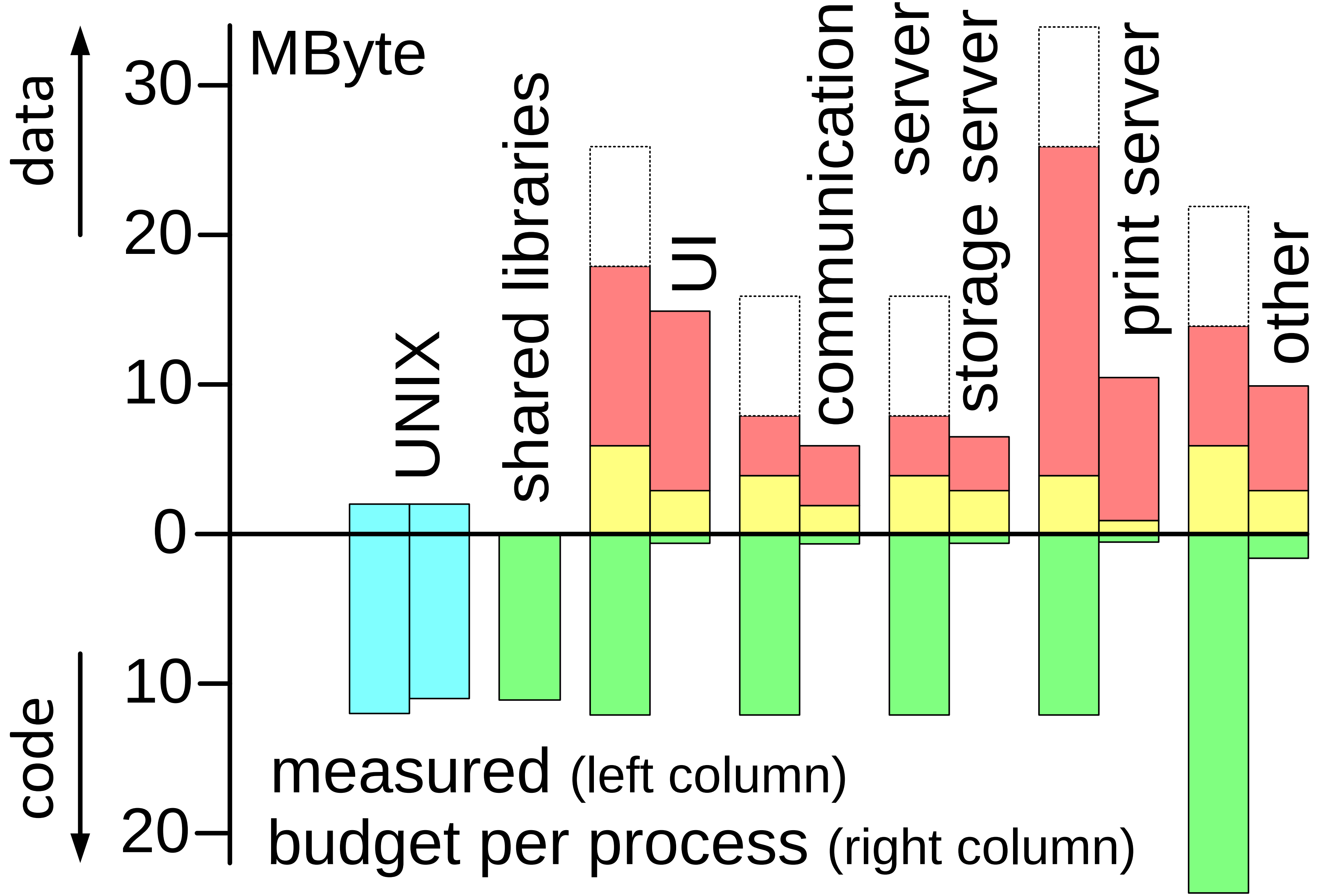
Performance as a Function of Memory Use



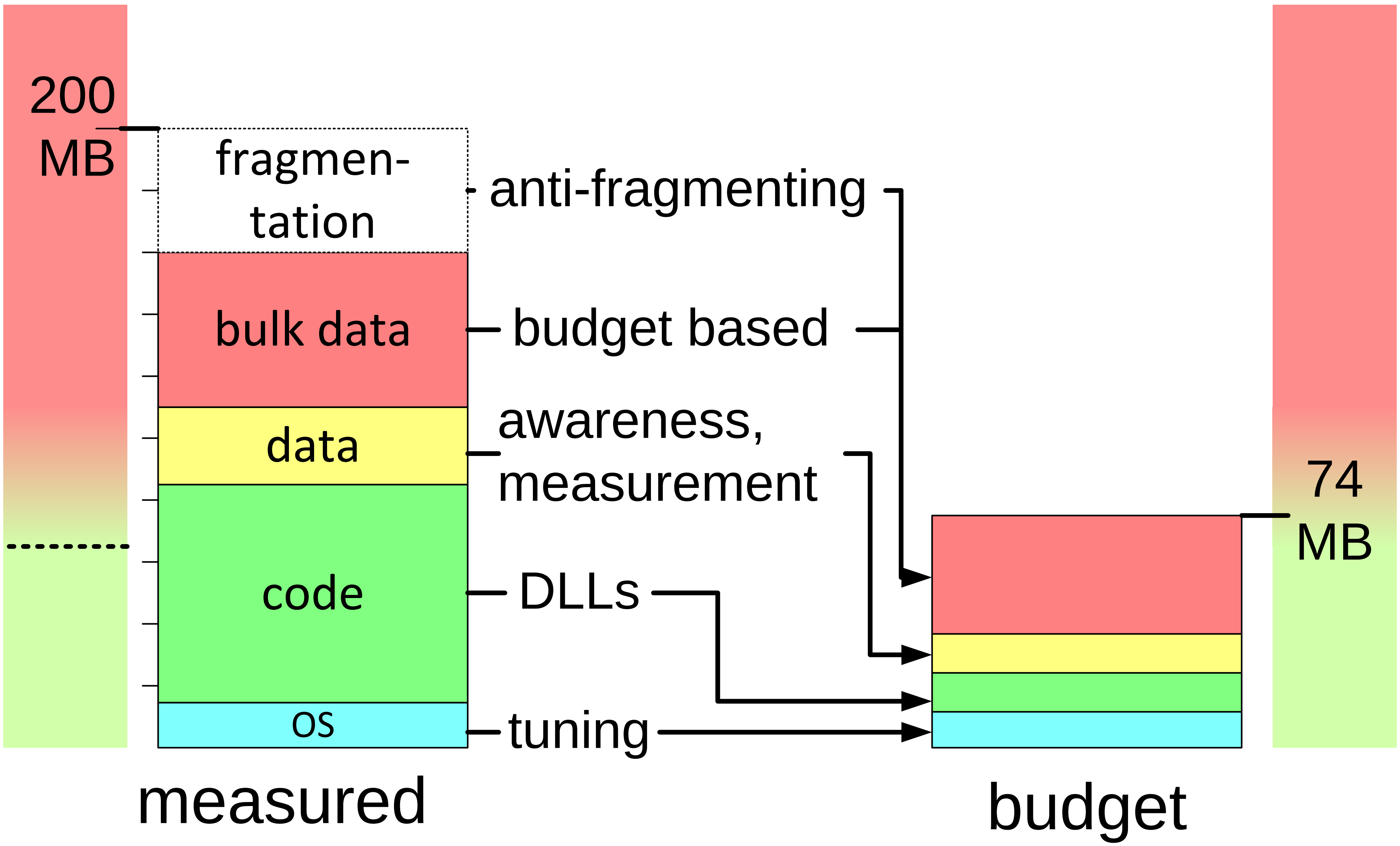
Problem: Unlimited Memory Consumption (1992)



Measurement Per Process



Solution: Measure and Iterative Redesign

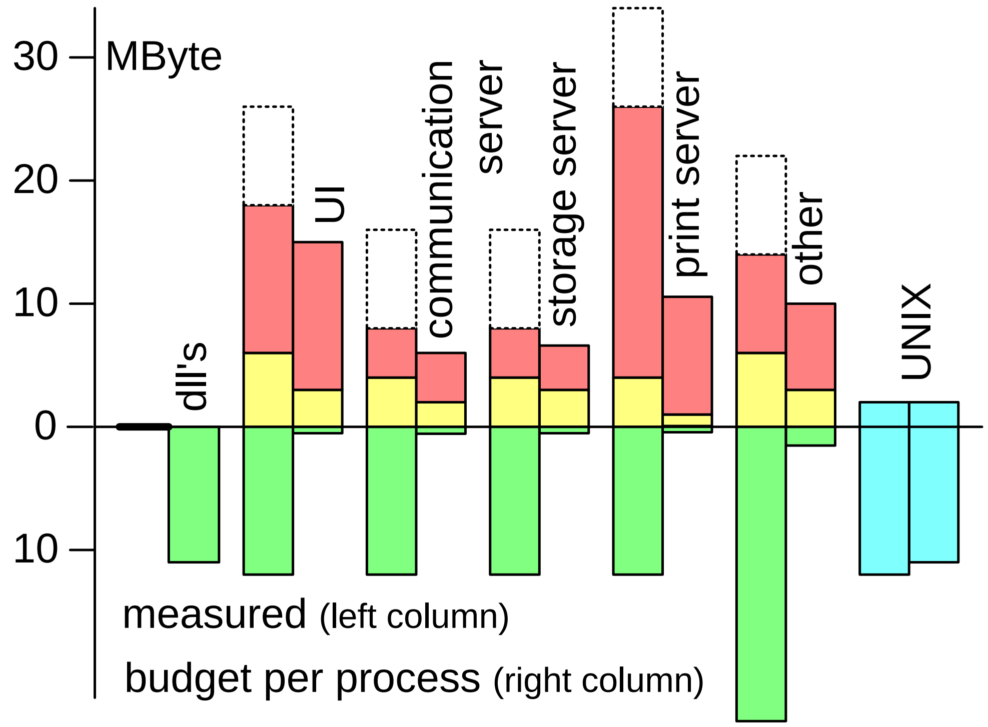


Budget:

+ measurable

+ fine enough to provide direction

+ coarse enough to be maintainable



Example of a Memory Budget

memory budget in Mbytes	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

Exercise: Bulk Data Capacity

Memory block

12MByte

How many blocks of
1024 x 1024 8-bits data
can be stored?

How many blocks of
1024 x 1024 16-bits data
can be stored?

Exercise: Object Data Capacity

Object Data
3MByte

Frequency		Description		Typical size
1		Large objects (e.g. dictionary)		20 kB
20		Medium object, e.g. UI data		200 Bytes
1000		Small object, e.g. image attributes		20 Bytes
Total				

How many objects with this distribution fit in the 3MByte Object data store?

Memory Budget of Easyvision RF R1 and R2

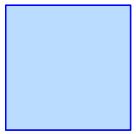
	code		object data		bulk data		total	
memory budget in Mbytes	R1	R2	R1	R2	R1	R2	R1	R2
shared code	6.0	11.0					6.0	11.0
UI process	0.2	0.3	2.0	3.0	12.0	12.0	14.2	15.3
database server	0.2	0.3	4.2	3.2		3.0	4.4	6.5
print server	0.4	0.3	2.2	1.2	7.0	9.0	9.6	10.5
DOR server	0.4	0.3	4.2	2.0	2.0	1.0	6.6	3.3
communication server	1.2	0.3	15.4	2.0	10.0	4.0	26.6	6.3
UNIX commands	0.2	0.3	0.5	0.2			0.7	0.5
compute server		0.3		0.5		6.0		6.8
system monitor		0.3		0.5				0.8
application total	8.6	13.4	28.5	12.6	31.0	35.0	66.1	61.0
UNIX							7.0	10.0
file cache							3.0	3.0
total							76.1	74.0

Answer: Bulk Data Capacity

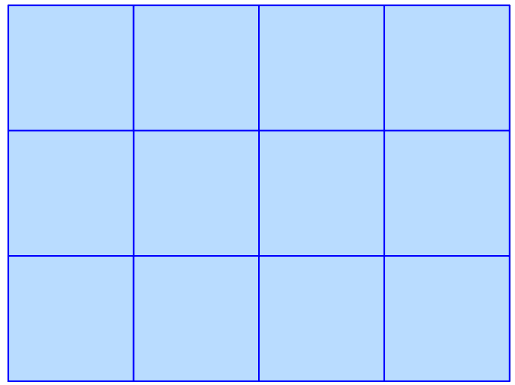
Memory block

12MByte

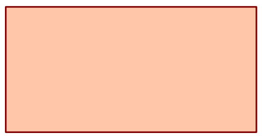
How many blocks of
1024 x 1024 8-bits data
can be stored?



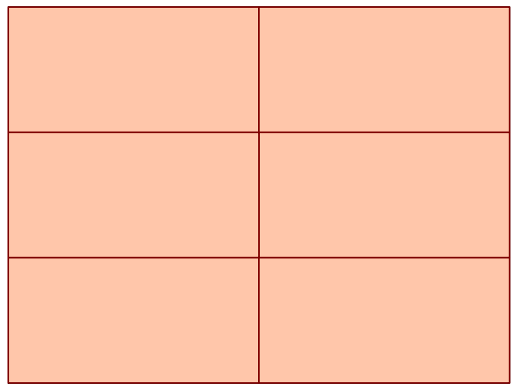
12



How many blocks of
1024 x 1024 16-bits data
can be stored?



6



* Assuming that 8-bit data is stored as 8-bit (char)
Assuming that 16-bit data is stored as 16-bit (short int)

Answer: Object Data Capacity

Object Data
3MByte

Frequency	Description	Typical size	Size * Freq
1	Large objects (e.g. dictionary)	20 kB	20 kB
20	Medium object, e.g. UI data	200 Bytes	4kB
1000	Small object, e.g. image attributes	20 Bytes	20kB
Total			44kB

44kByte fits approximately 68 times in 3MByte
Expect to store at most 68 large objects
(1360 Medium sized objects, 68000 small objects)

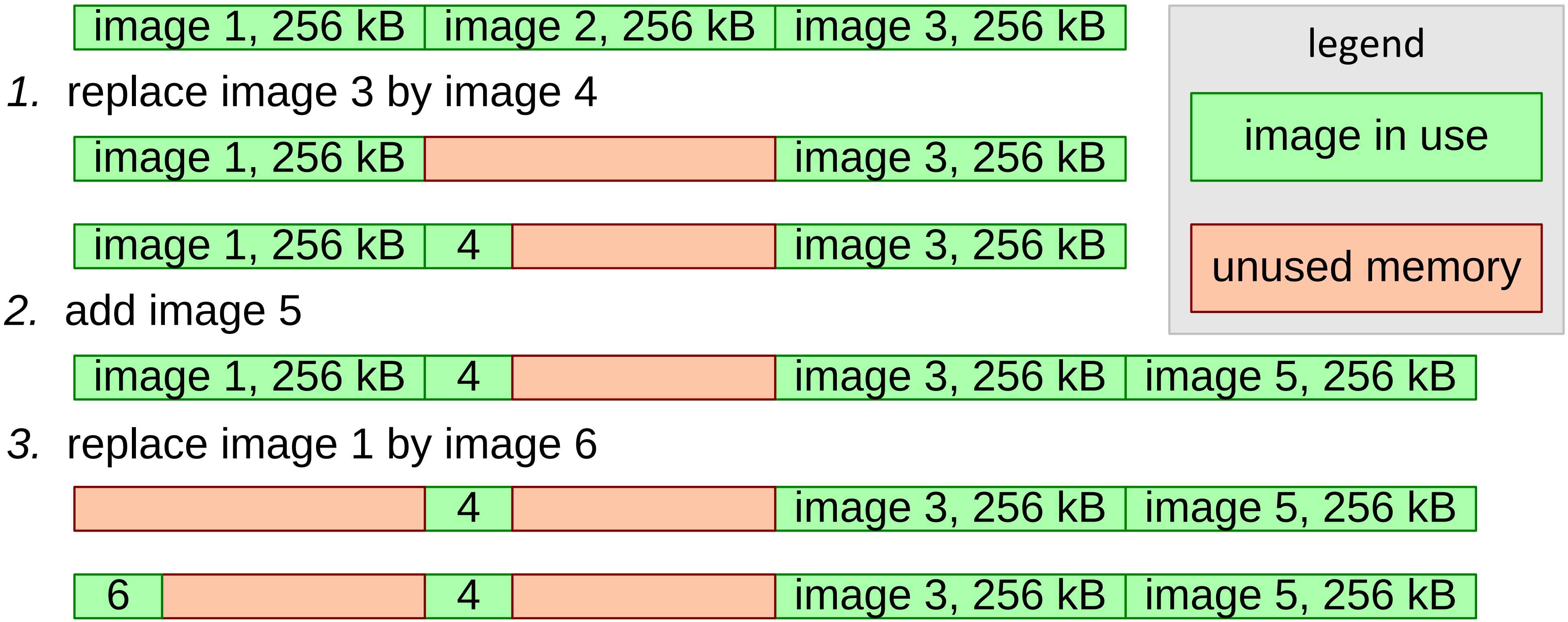
Easyvision Memory Design

Fragmentation and consequences

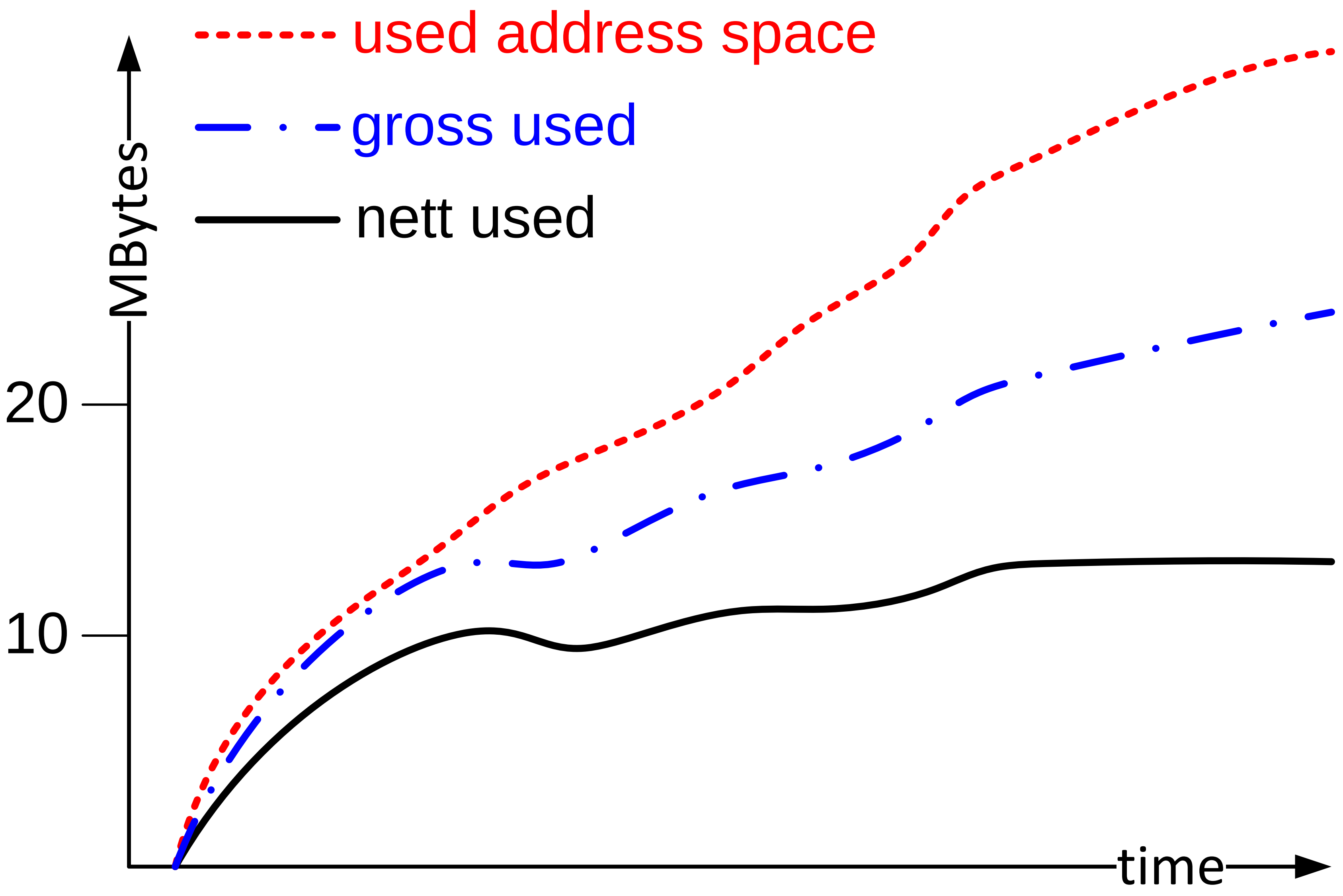
Application caches

Memory design applied

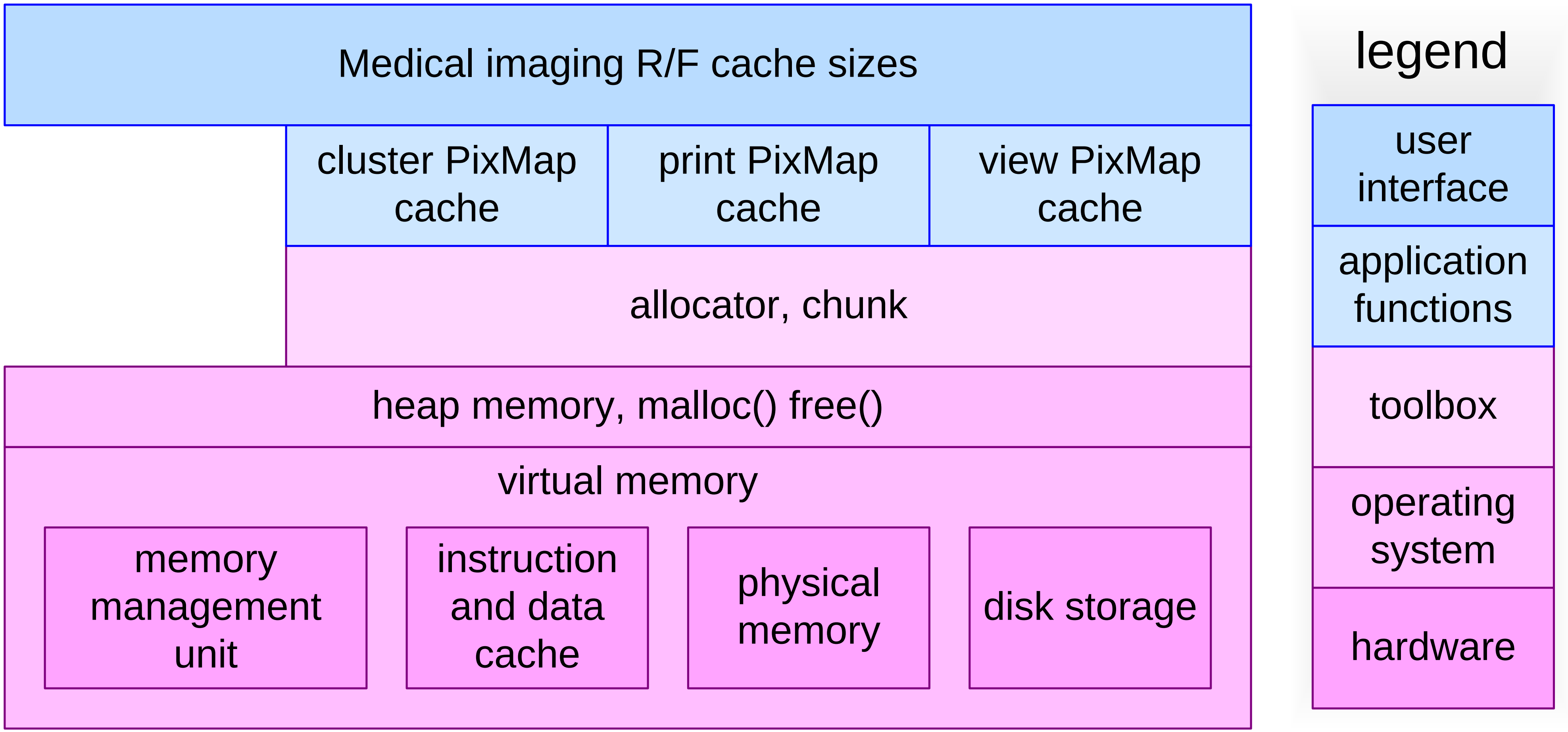
Memory Fragmentation



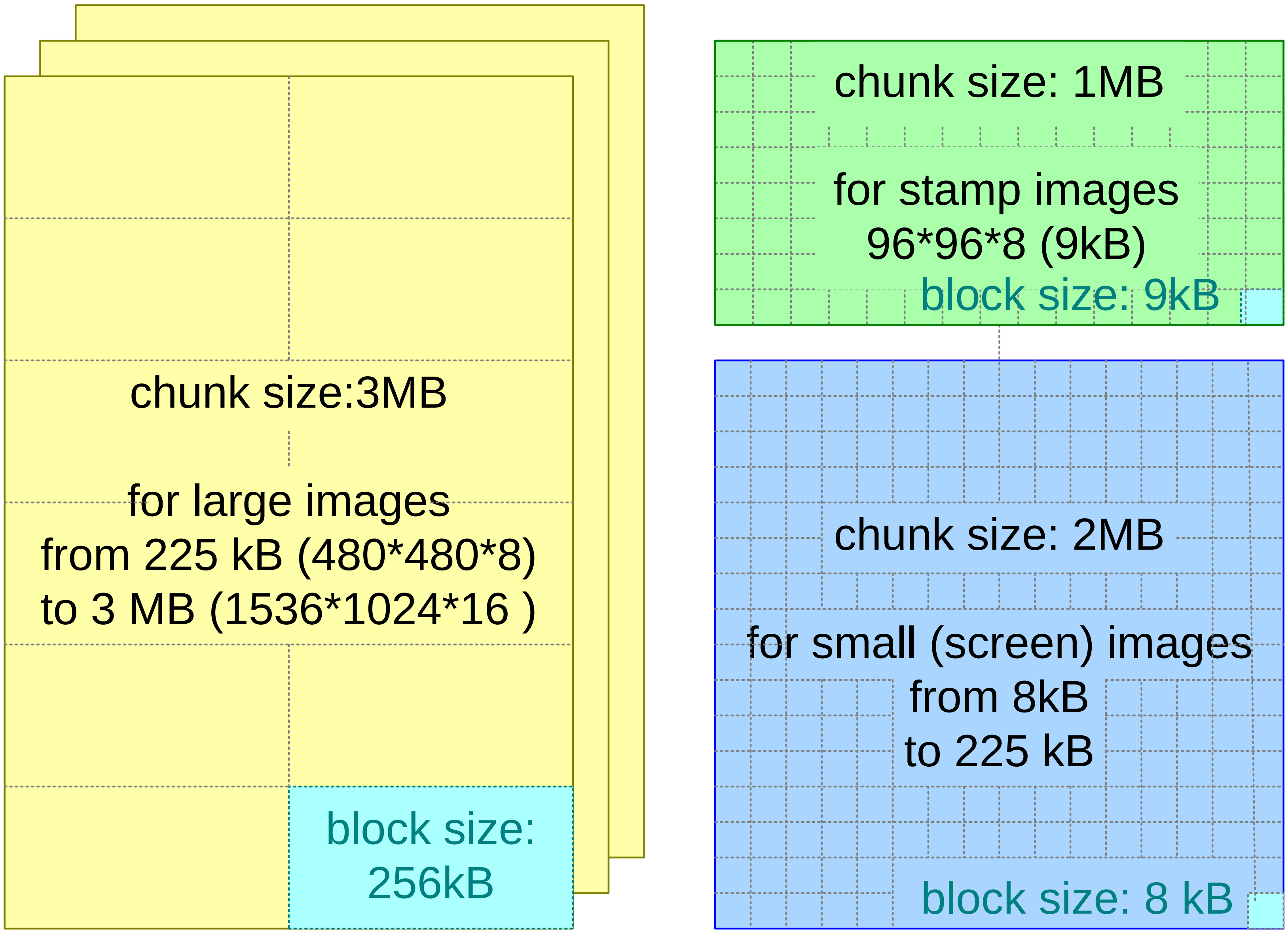
Memory Fragmentation Increase



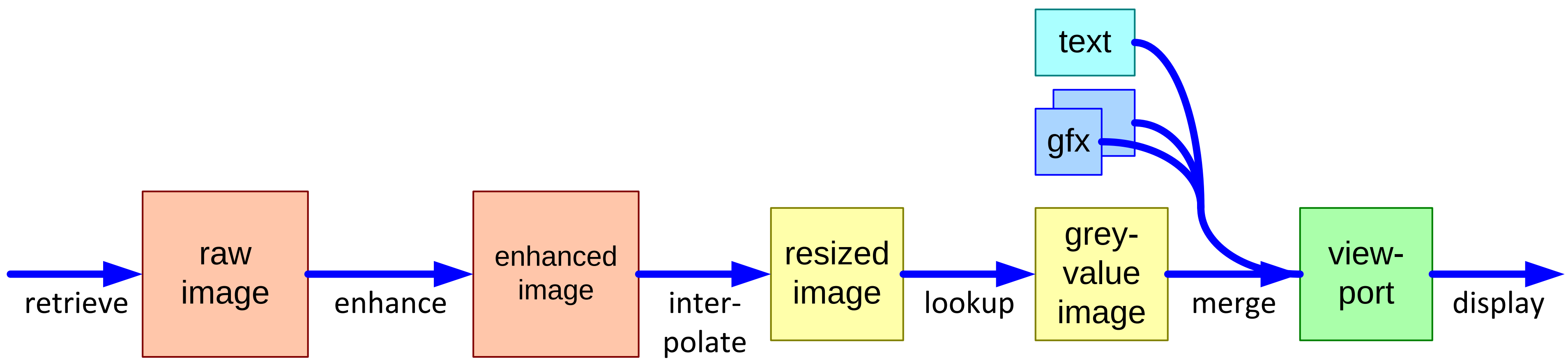
Cache Layers



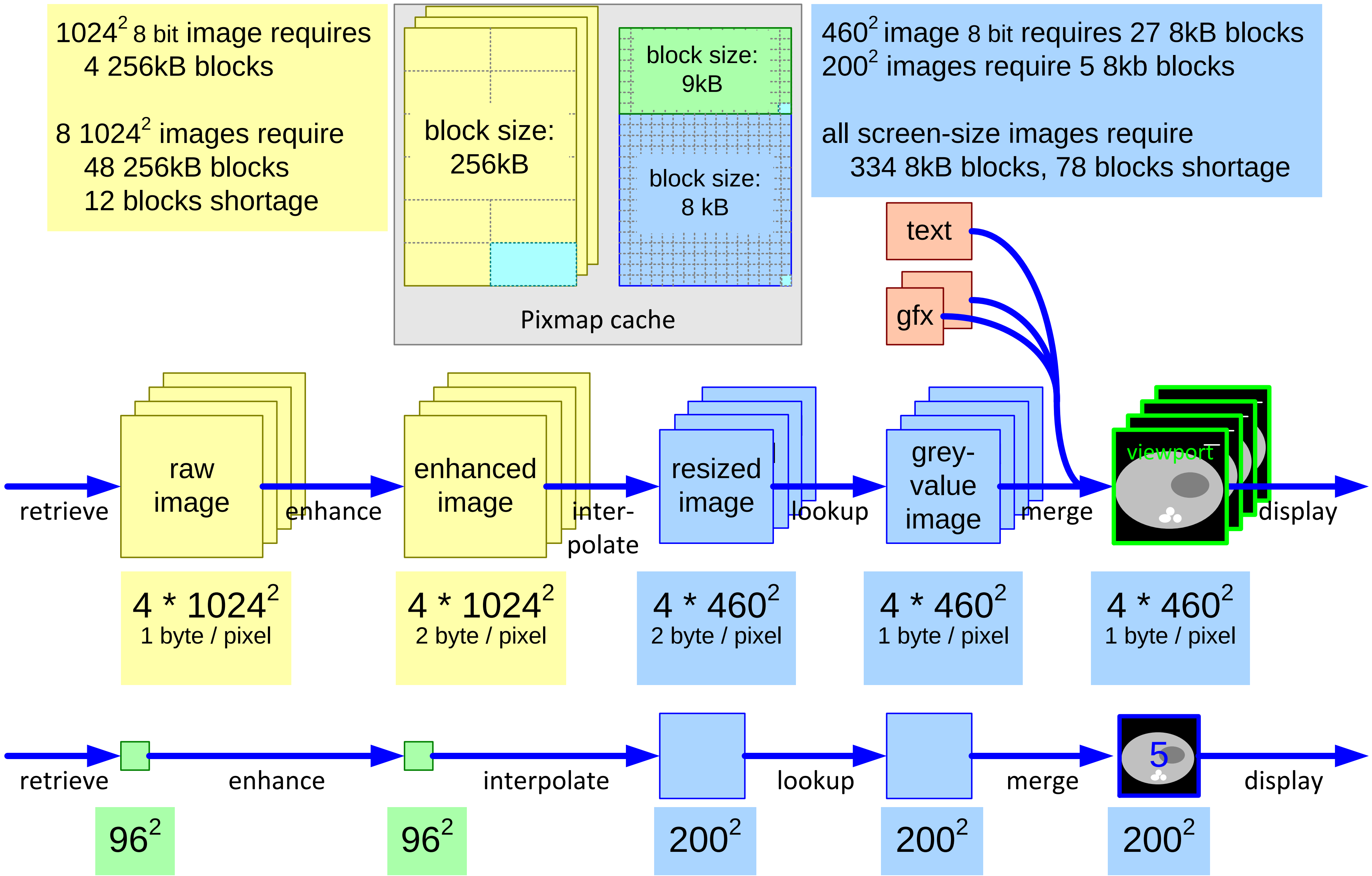
Bulk Data Memory Management Memory Allocators



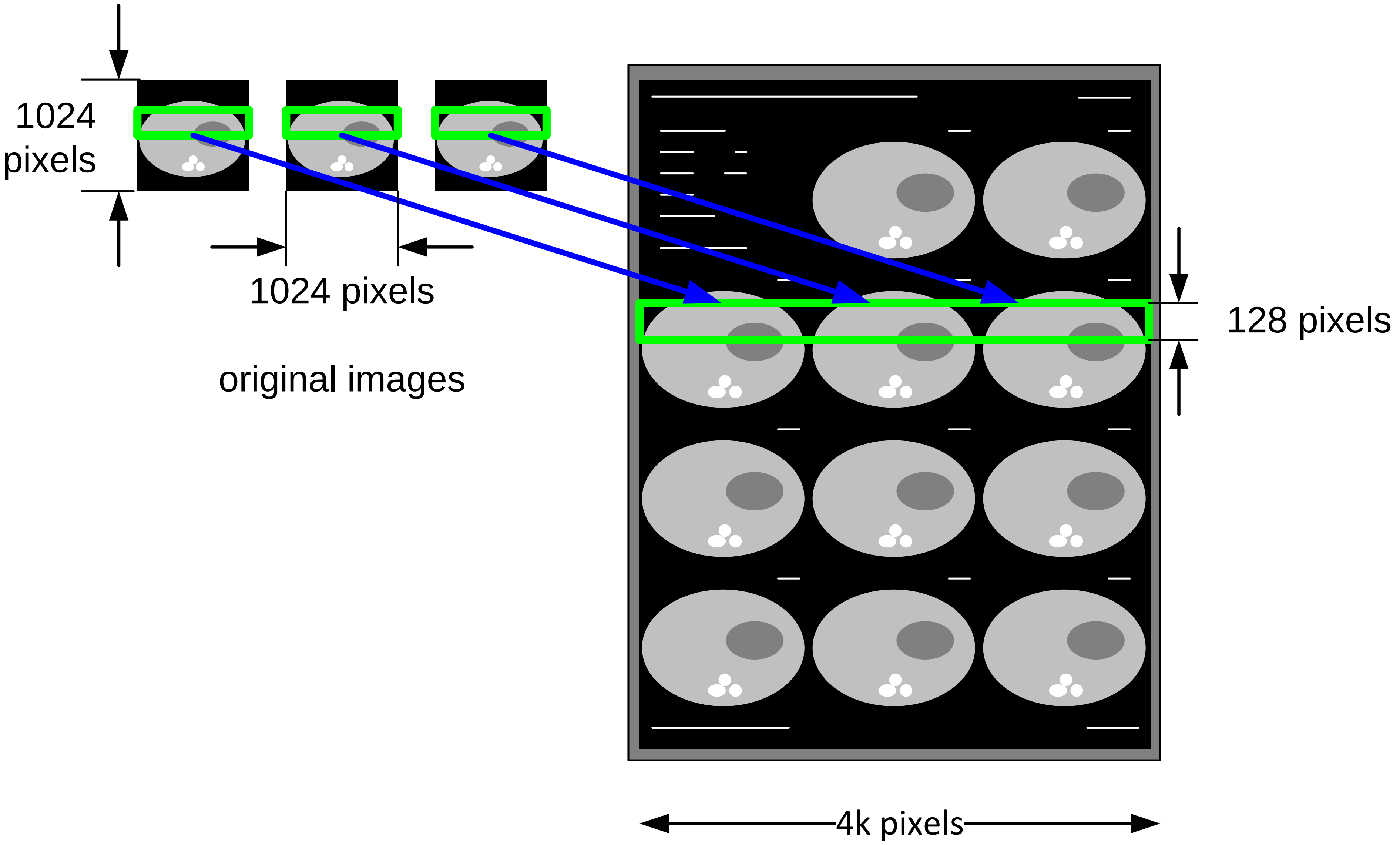
Cached Intermediate Processing Results



Example of Allocator and Cache Use



Print Server is Based on Banding



CPU Load and Performance

Easyvision Memory CPU load and performance

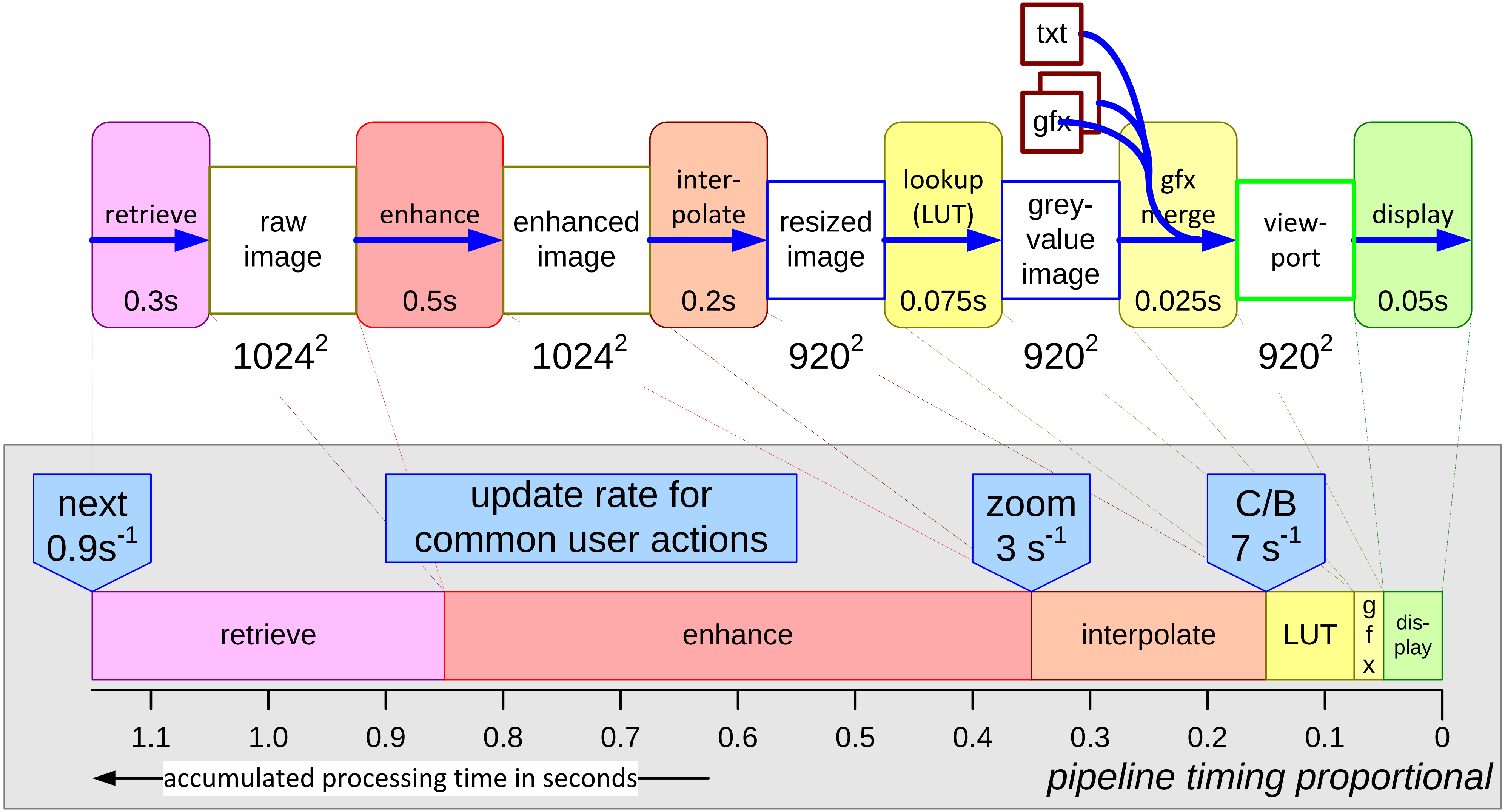
CPU load analysis

response time

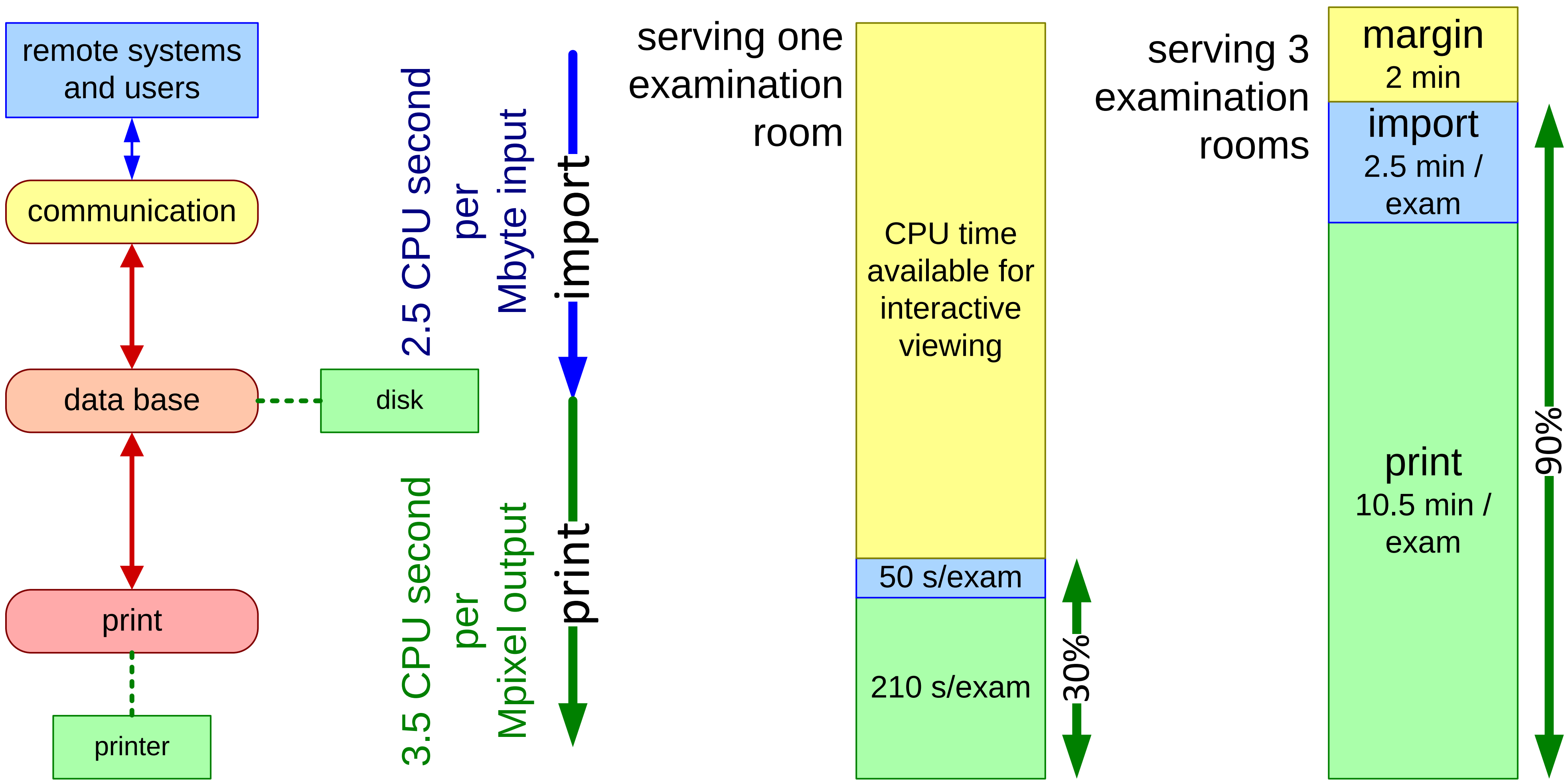
throughput

measurement tools

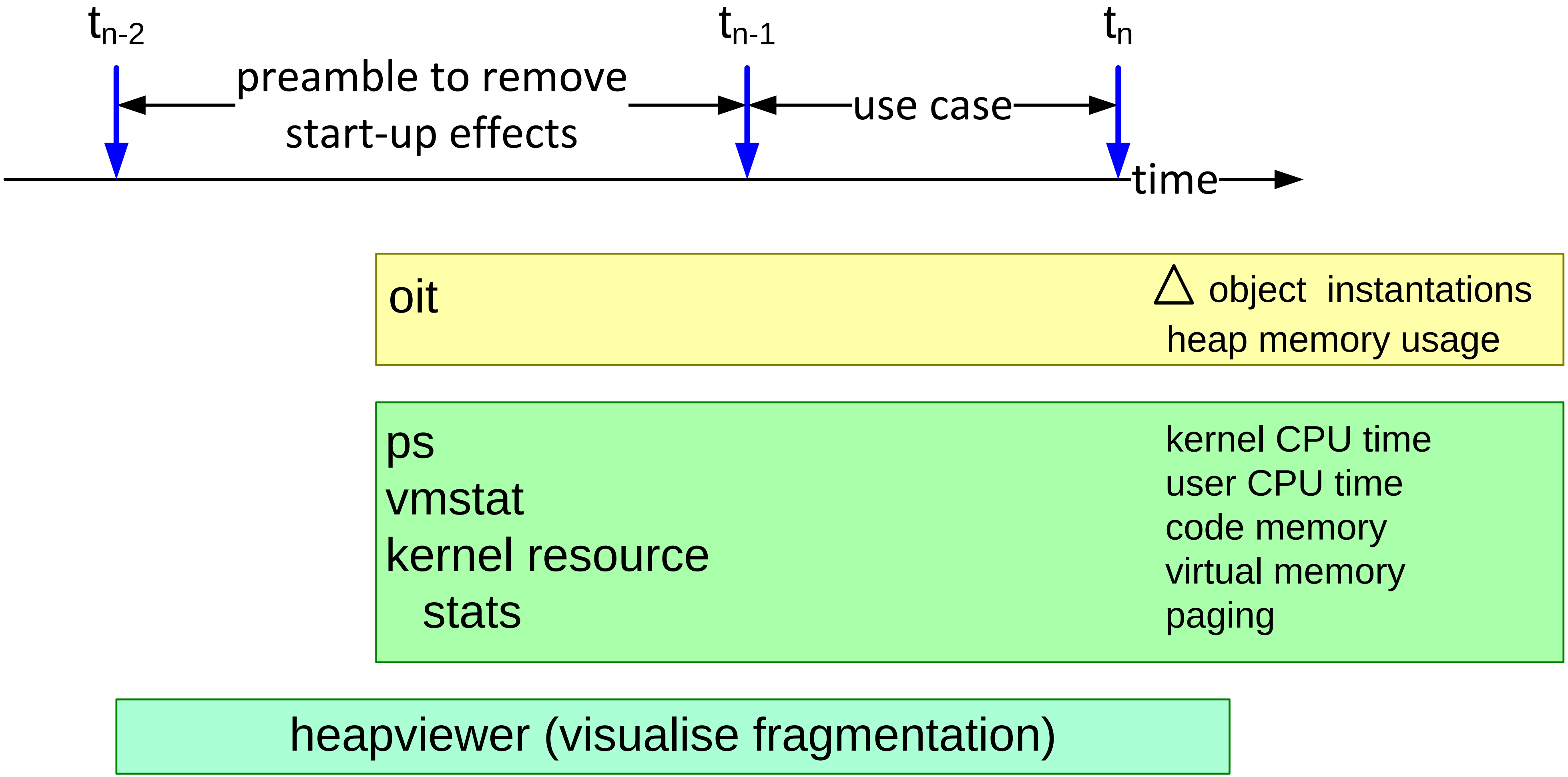
CPU Processing Times and Viewing Responsiveness



Server CPU Load



Resource Measurement Tools



Object Instantiation Tracing

class name	current nr of objects	deleted since t_{n-1}	created since t_{n-1}	heap memory usage
AsynchronousIO	0	-3	+3	
AttributeEntry	237	-1	+5	
BitMap	21	-4	+8	
BoundedFloatingPoint	1034	-3	+22	
BoundedInteger	684	-1	+9	
BtreeNode1	200	-3	+3	[819200]
BulkData	25	0	1	[8388608]
ButtonGadget	34	0	2	
ButtonStack	12	0	1	
ByteArray	156	-4	+12	[13252]

Overview of Benchmarks and Other Measurement Tools

	test / benchmark	what, why	accuracy	when
public	SpecInt (by suppliers)	CPU integer	coarse	new hardware
	Byte benchmark	computer platform performance OS, shell, file I/O	coarse	new hardware new OS release
self made	file I/O	file I/O throughput	medium	new hardware
	image processing	CPU, cache, memory as function of image, pixel size	accurate	new hardware
	Objective-C overhead	method call overhead memory overhead	accurate	initial
	socket, network	throughput CPU overhead	accurate	ad hoc
	data base	transaction overhead query behaviour	accurate	ad hoc
	load test	throughput, CPU, memory	accurate	regression

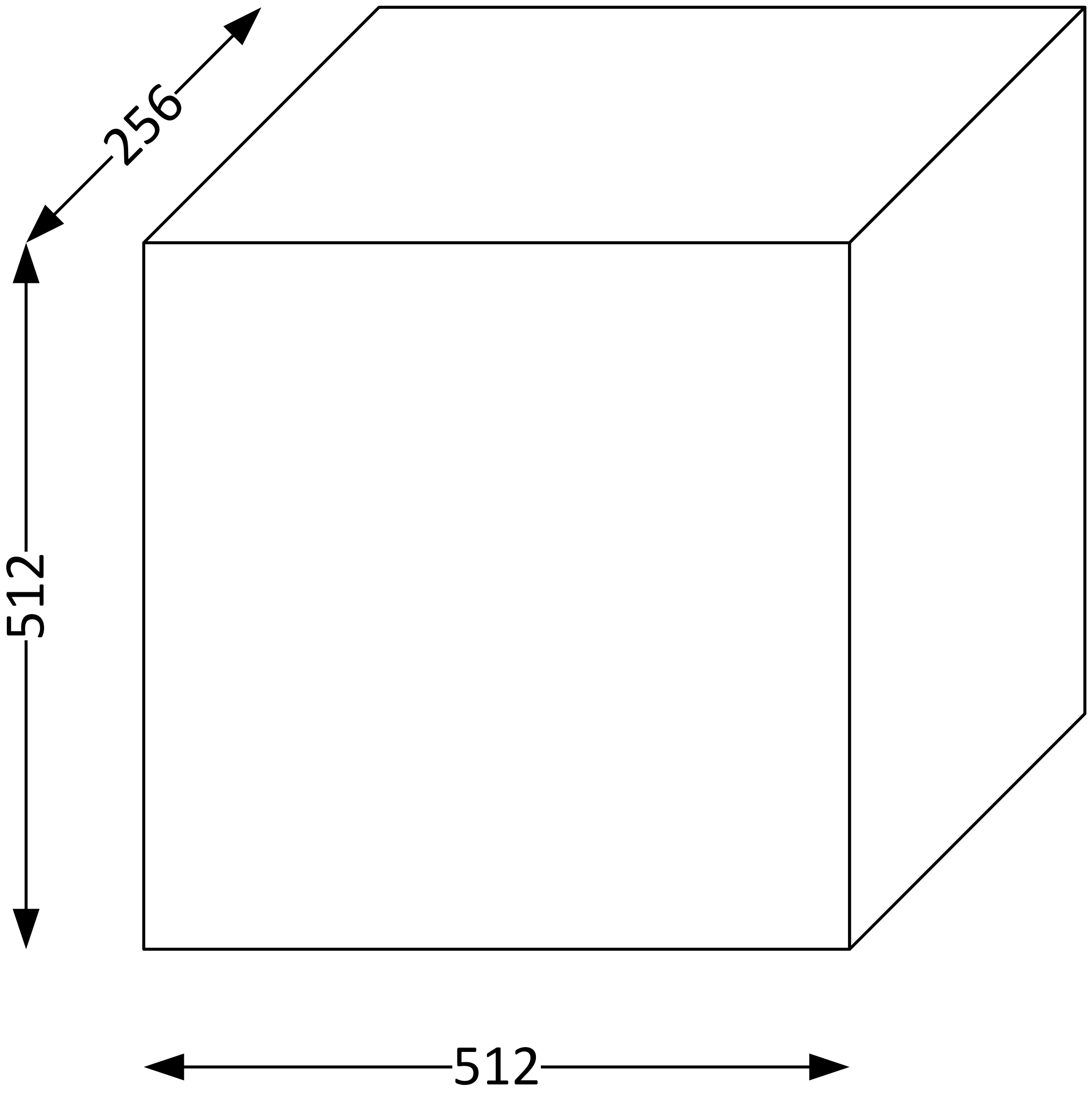
Case 4

MRI Volume Reconstruction and Viewing

Usage patterns as impact on performance

Resource model and requirements identification for usage patterns

Volume Acquisition and Reconstruction



Data in bytes =

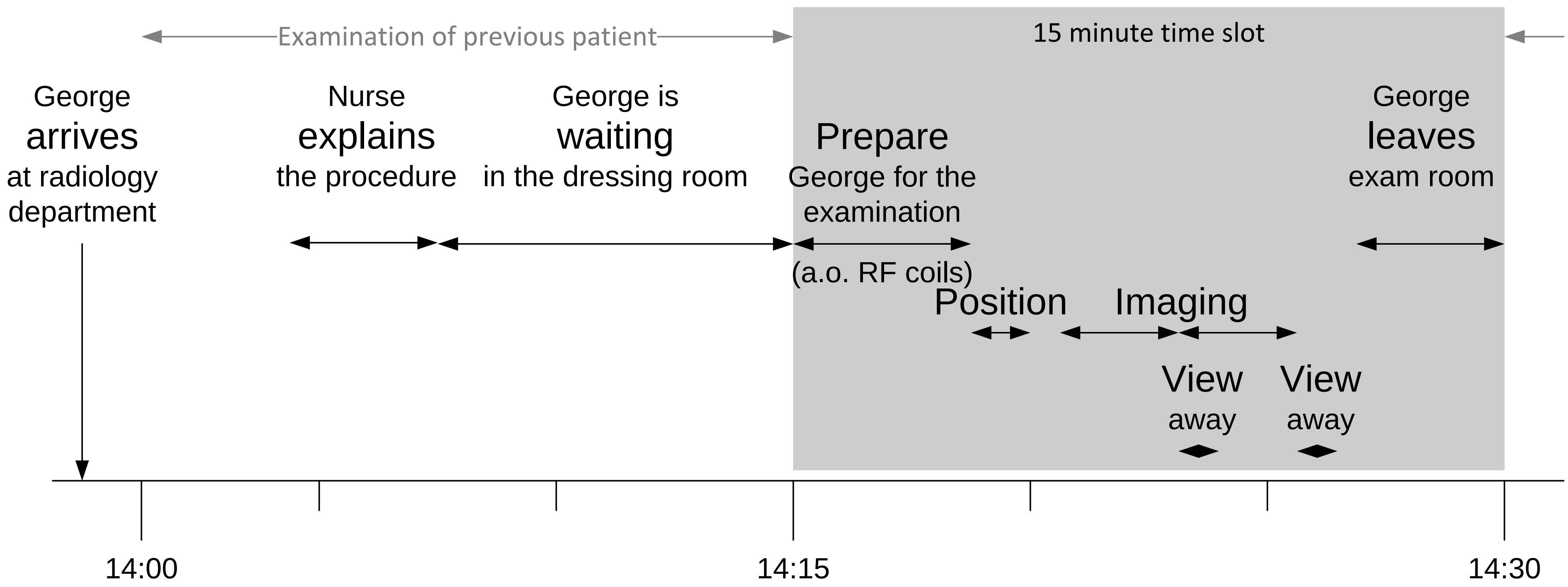
2 * 512 * 512 * 256 * 2 =

Volumes x y z bytes per pixel

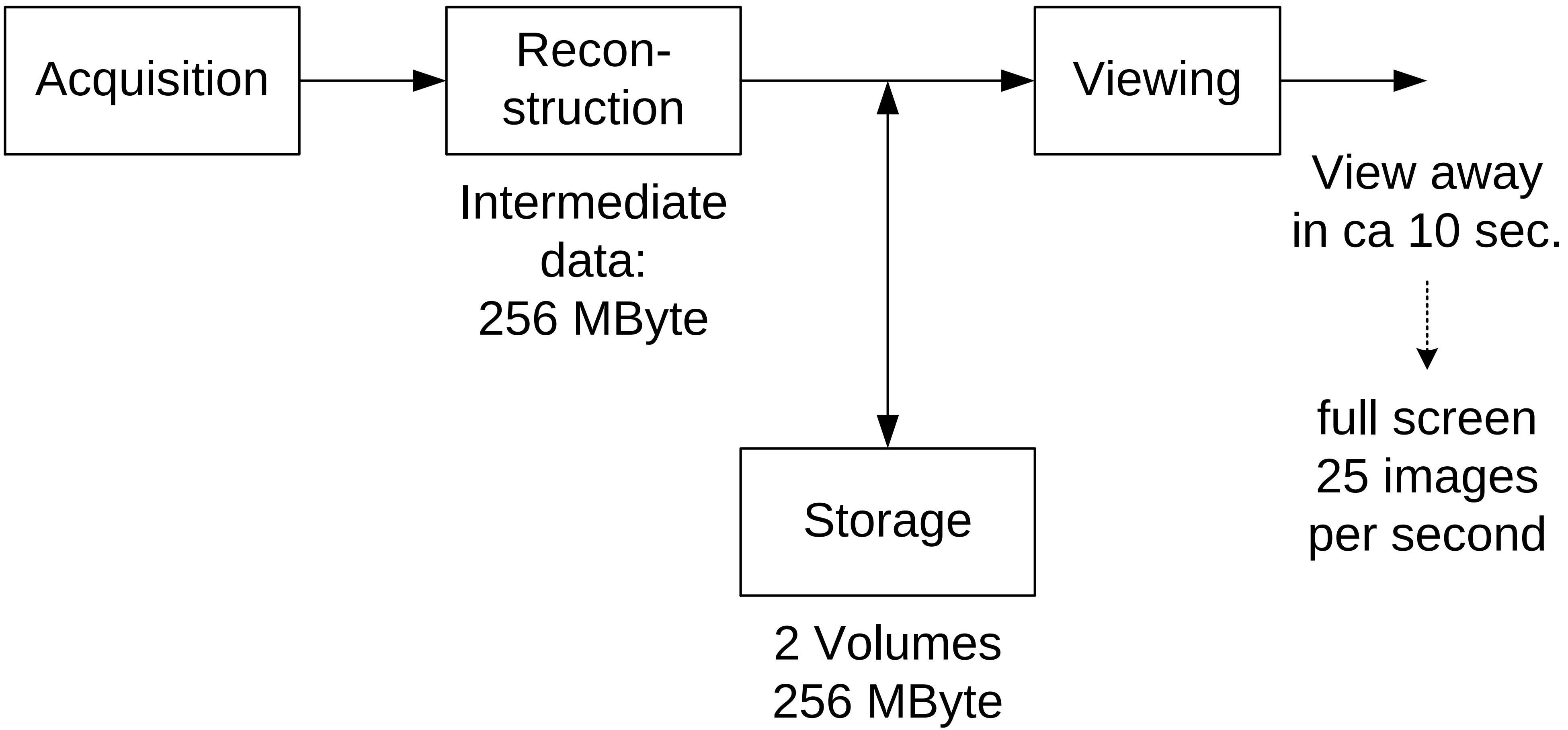
256 MBytes

in 2 * 2 minutes =
240 seconds

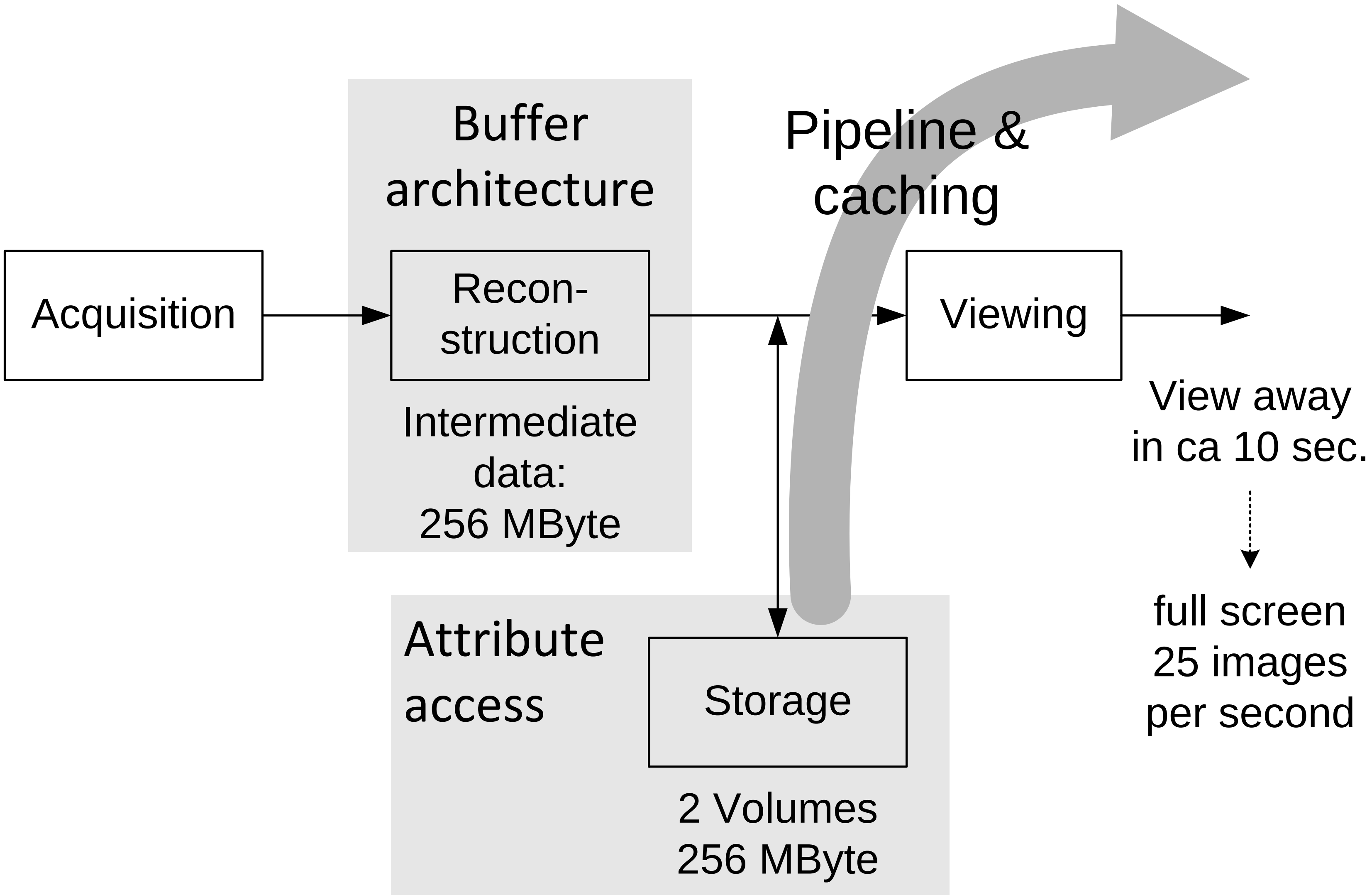
Performance Requirements



Resource Model



Critical Resources



Case 4

MRI Volume Reconstruction and Viewing

Operational usage pattern drives (implicit/explicit) system performance requirements

Resource / cost trade-off must support operational usage patterns

Case 5

Mobile Display Appliances

Modelling external environment

End-to-end performance

Allocation choices

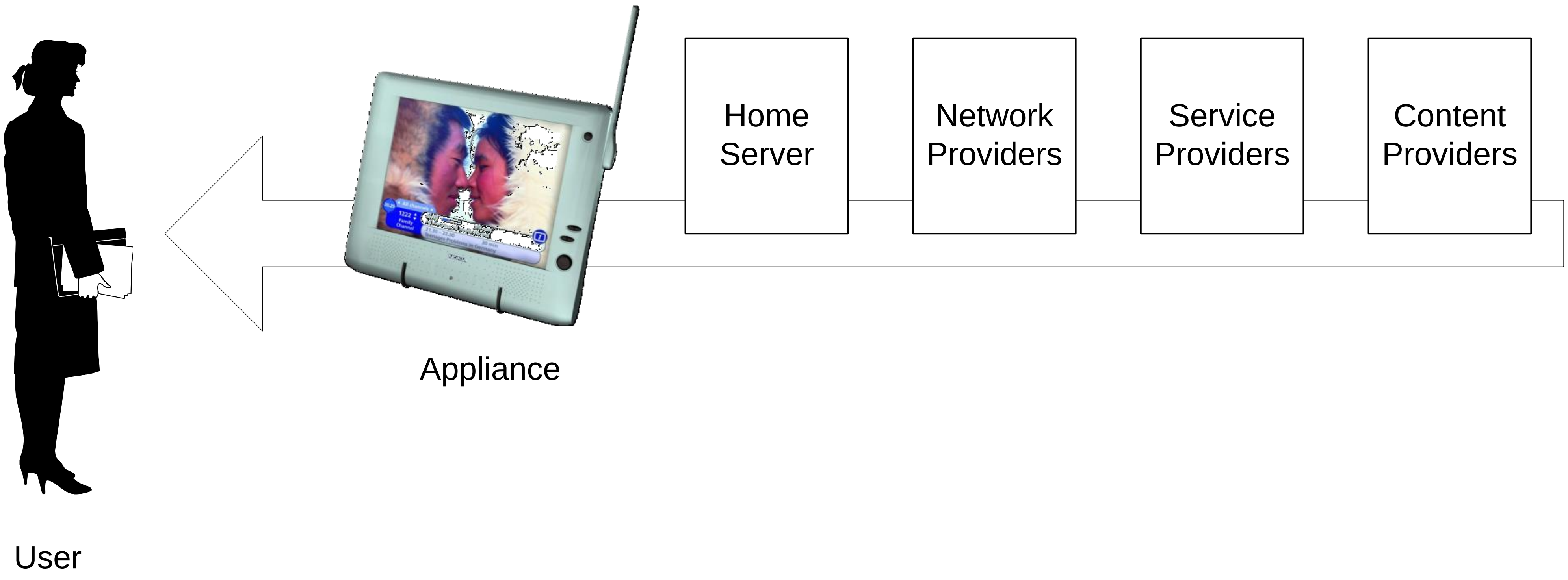
Mobile Display Appliances

Mobile Display Appliance

Mediascreen

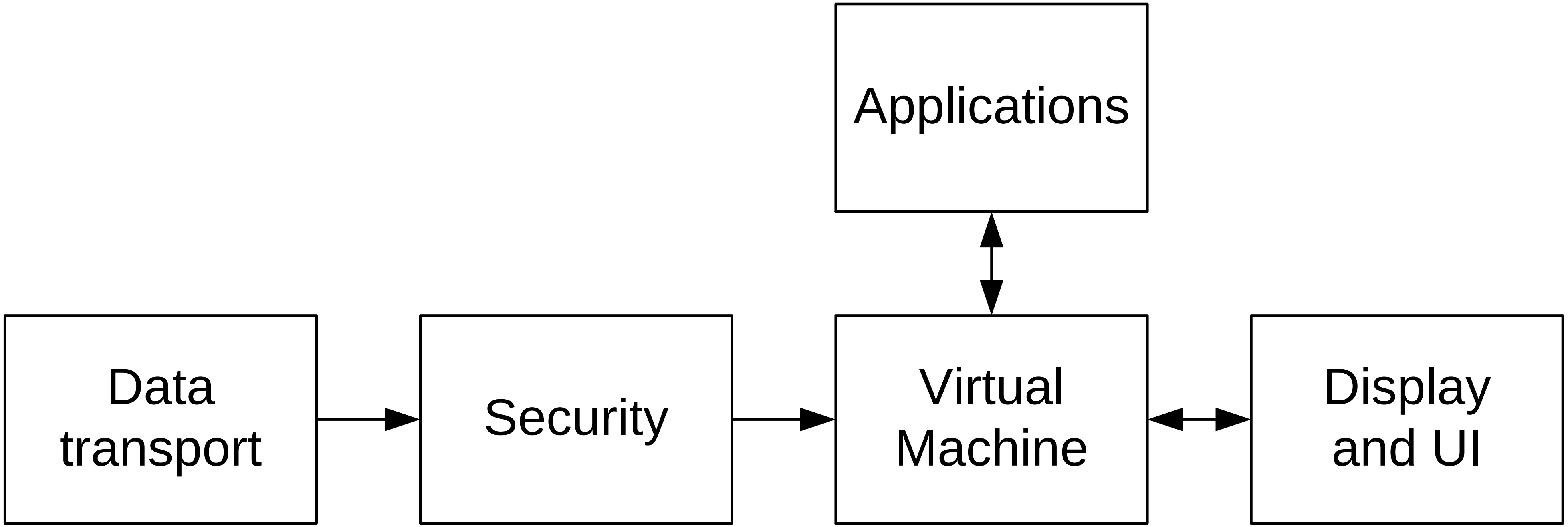
Original pictures from Nokia

User Access Point to a Long Foodchain



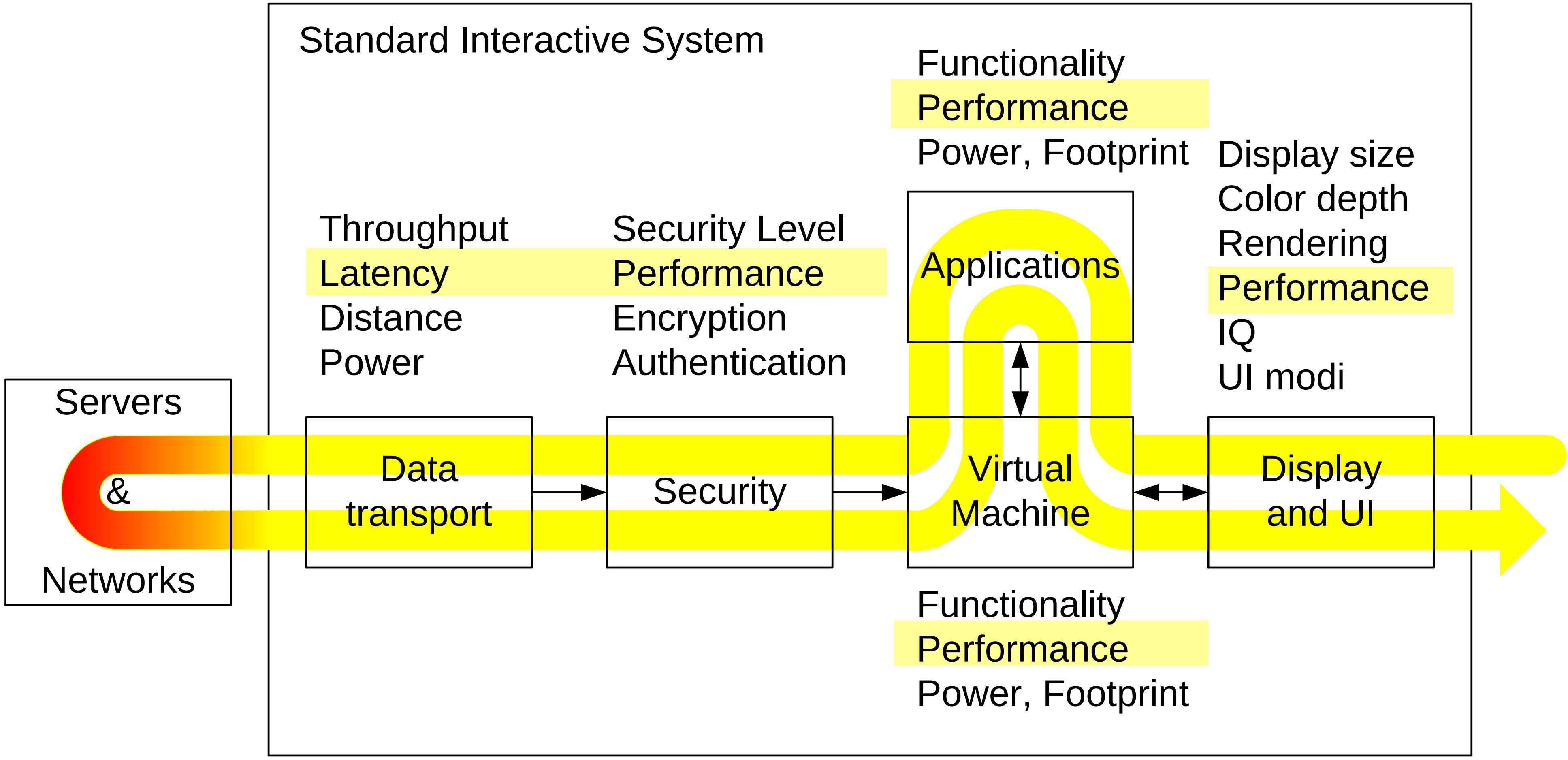
The "SMART" World of the Design

Standard Interactive System



free after Nick Thorne, Philips Semiconductors,
Systems Laboratory Southampton UK,
as presented at PSAVAT April 2001

Specifiable Characteristics

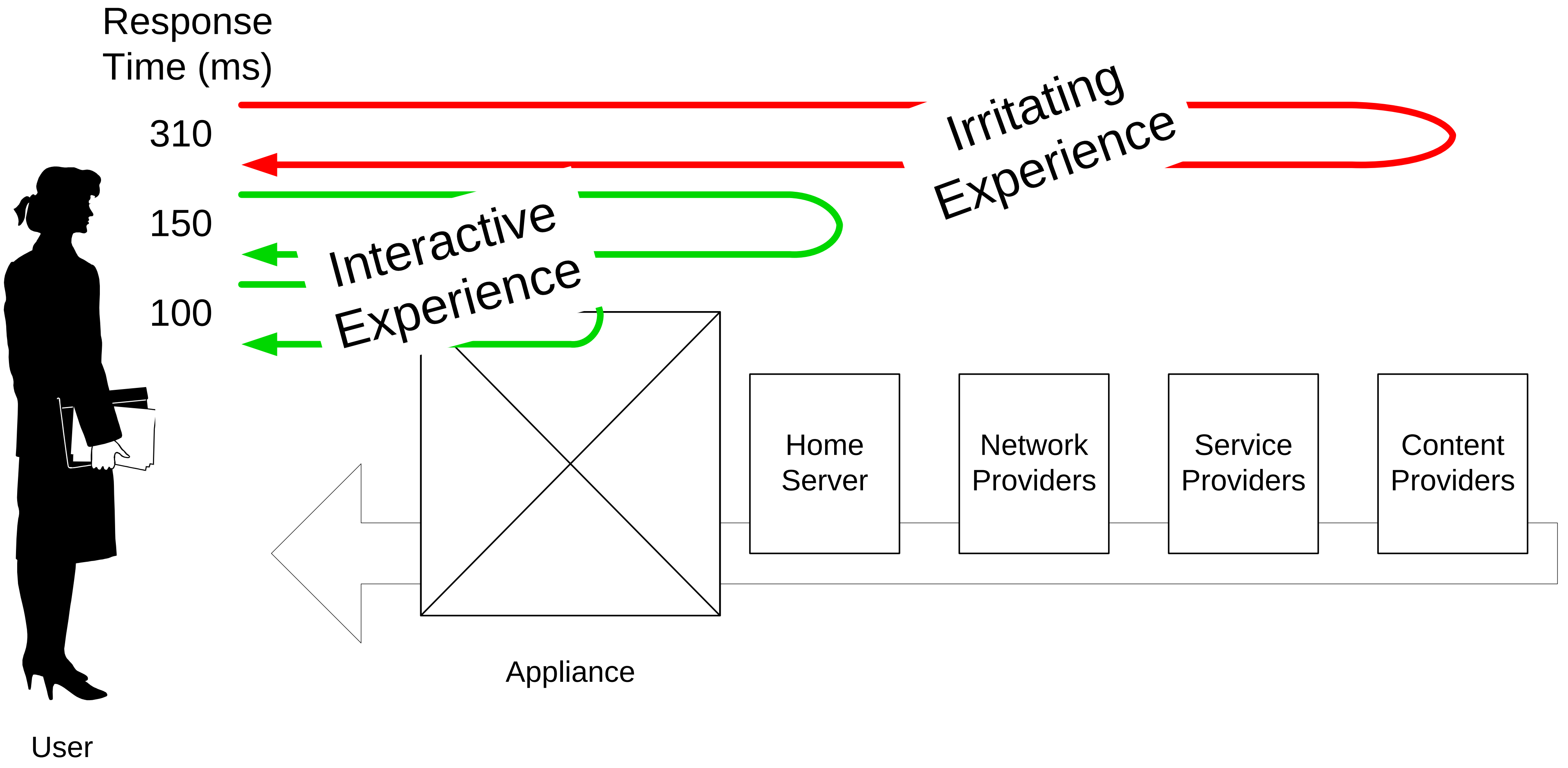


Response Time: Latency Budget

times in milliseconds		Message Latency		Response Time
Appliance		40		100
Data transport	10		20	
Security	10		20	
Virtual Machine	10		20	
Application	10		30	
Graphics and UI	0		10	
Home Network		20		50
Home Server	10		30	
Network contention	10		20	
Provider Infrastructure		50		160
Last-Mile network	10		20	
Backbone network	20		40	
Service server	10		50	
Content server	10		50	
Total		110		310
User need				200

All numbers are imaginary and for illustration purposes only

Interaction or Irritation?



Case 5 Summary

Mobile Display Appliances

Modelling external environment: make assumptions

End-to-end performance:

- large part of performance budget is not controlled

User perceived performance determines function allocation

Exercise

Explore “Fast Browser” product specification, design options and performance issues

Scheduling Techniques and Analysis

by *Gerrit Muller* HSN-NISE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

Abstract

The choice of scheduling technique and it's parametrization impacts the performance of systems. This is an area where quite some theoretical work has been done. In this presentation we address Earliest Deadline First and Rate Monotolic Scheduling (RMS). We provide how-to information for RMS, based on Rate Monotonic Analysis (RMA).

March 6, 2021
status: preliminary draft
version: 0

Embedded Systems INSTITUTE

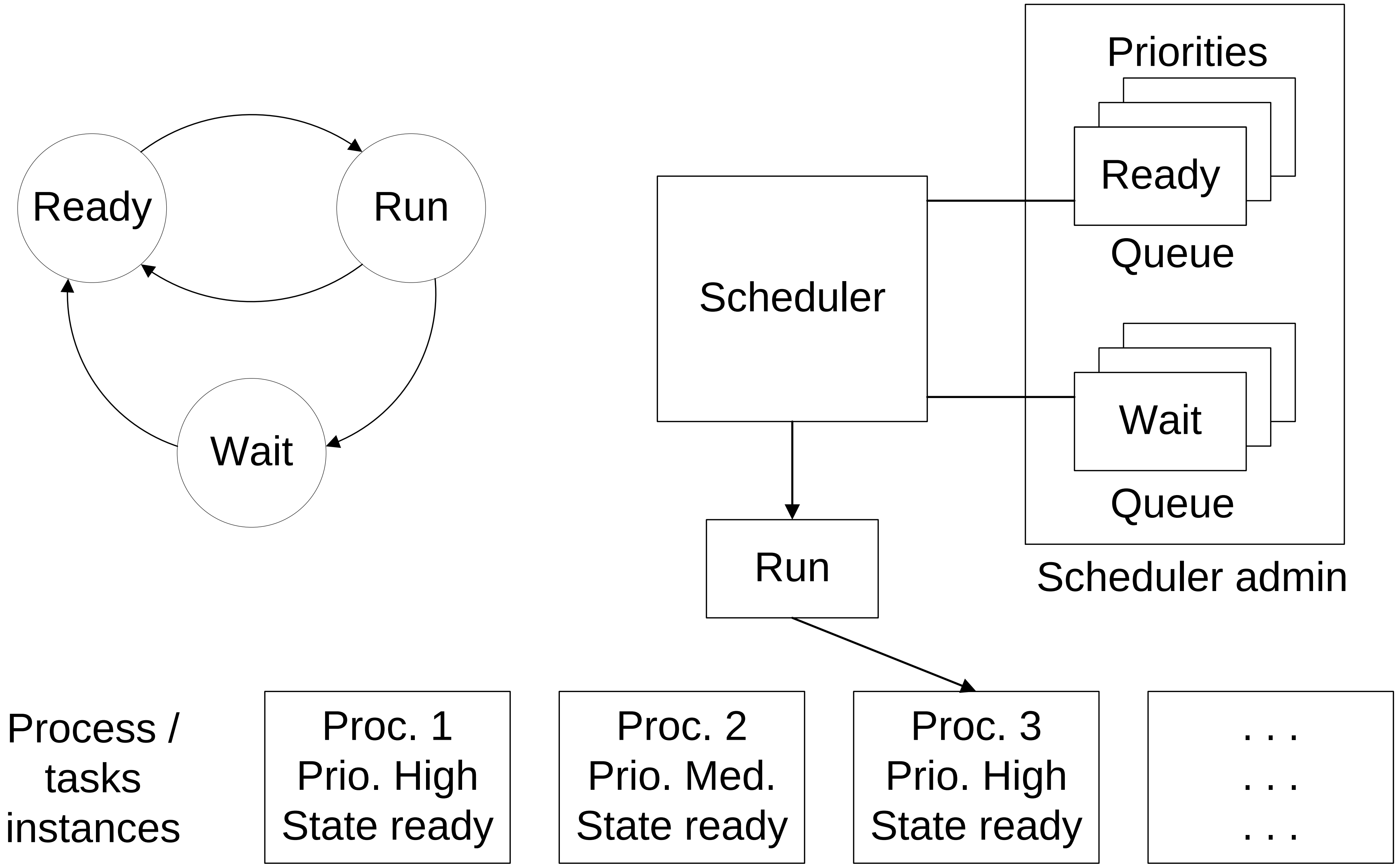
Theory Block Scheduling Techniques and Analysis

Theory Hard Real Time Scheduling

Earliest Deadline First (EDF)

Rate Monotonic Scheduling (RMS)

Real Time Scheduling



Earliest Deadline First

• Determine deadlines	in Absolute time (CPU cycles or msec, etc.)
• Assign priorities	Process that has the earliest deadline gets the highest priority (no need to look at other processes)
• Constraints	Smart mechanism needed for Real-Time determination of deadlines Pre-emptive scheduling needed

EDF = Earliest Deadline First

Earliest Deadline based scheduling
for (a-)periodic Processing

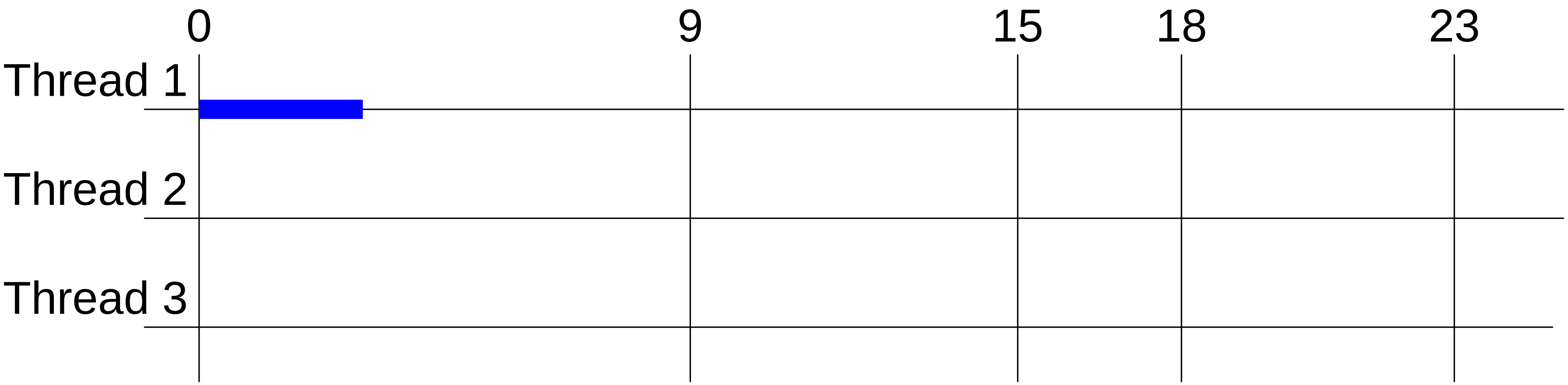
The theoretical limit for any number of processes
is 100% and so the system is schedulable.

Exercise Earliest Deadline First (EDF)

Calculate loads and determine thread activity (EDF)

Thread	Period = deadline	Processing	Load
Thread 1	9	3	33.3%
Thread 2	15	5	
Thread 3	23	5	

Suppose at t=0, all threads are ready to process the arrived trigger.



Source: Ton Kostelijk - EXARCH course

Rate Monotonic Scheduling

• Determine deadlines (period)	in terms of Frequency or Period (1/F)
• Assign priorities	Highest frequency (shortest period) ==> Highest priority
• Constraints	Independent activities Periodic Constant CPU cycle consumption Assumes Pre-emptive scheduling

RMS = Rate Monotonic Scheduling

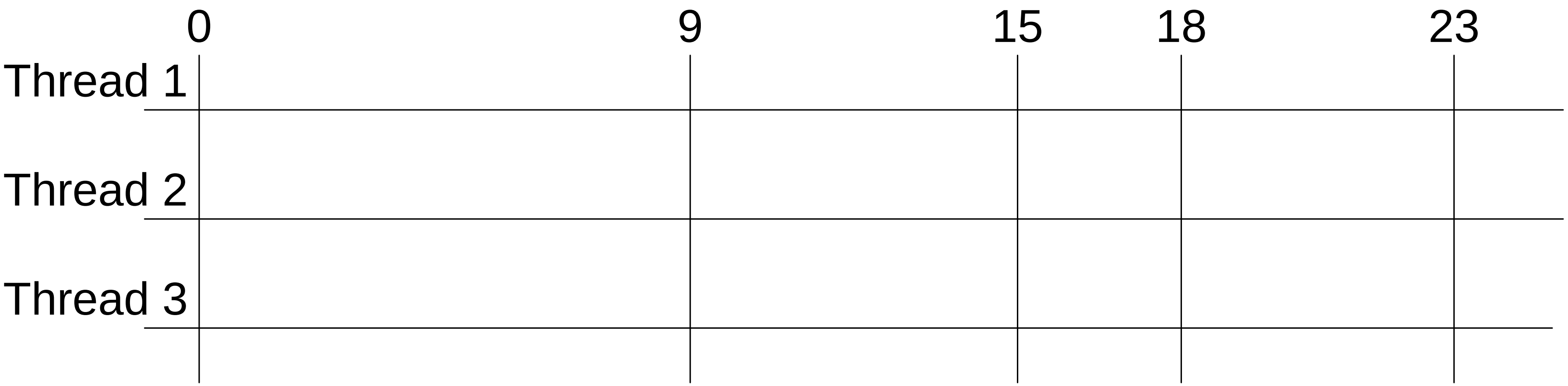
Priority based scheduling for Periodic Processing of tasks with a guaranteed CPU - load

Exercise Rate Monotonic Scheduling (RMS)

Calculate loads and determine thread activity (RMS)

Thread	Period = deadline	Processing	Load
Thread 1	9	3	33.3%
Thread 2	15	5	
Thread 3	23	5	

Suppose at t=0, all threads are ready to process the arrived trigger.



Source: Ton Kostelijk - EXARCH course

Real-time scheduling theory, utilization bound

- Set of tasks with periods T_i, and process time P_i: load $u_i = P_i / T_i$
- Schedule is at least possible when tasks are independent and:

$$Load \equiv \sum_i U_i \leq n\left(2^{\frac{1}{n}} - 1\right)$$

- 1.00 , 0.83 , 0.78 , 0.76 , ... log(2) = 0.69

Source: Ton Kostelijk - EXARCH course

RMS Evaluation

- RMS cannot utilize 100% (1.0) of CPU, but for 1, 2, 3, 4, ... ∞ processes: 1.00 , 0.83 , 0.78 , 0.76 , ... log(2) = 0.69
- RMS guarantees that all processes will always meet their deadlines, for any interleaving of processes.
- With fixed priorities, context switch overhead is limited

Source: Ton Kostelijk - EXARCH course

RMS Evaluation (continued)

- For specific cases the utilization bound can be higher:
  up to 0.88 load for large n

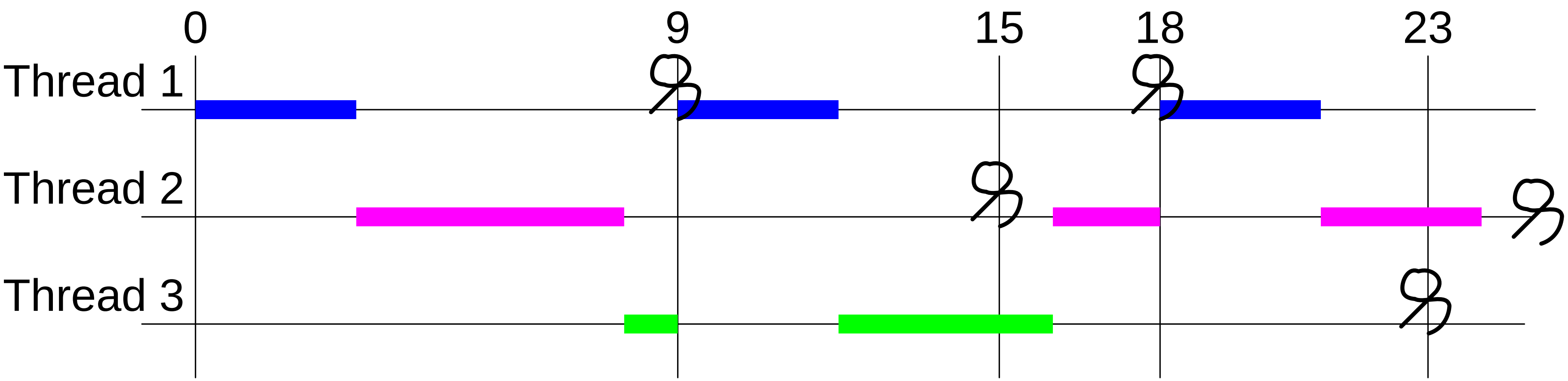
- A processor running only hard-real-time processes is rare.
  For soft-RT less of a problem

- A lot of additional theory exists.
  Meeting deadlines in hard-real-time systems
  (L.P. Briand & D.M. Roy)

Source: Ton Kostelijk - EXARCH course

Answers: loads and thread activity (EDF)

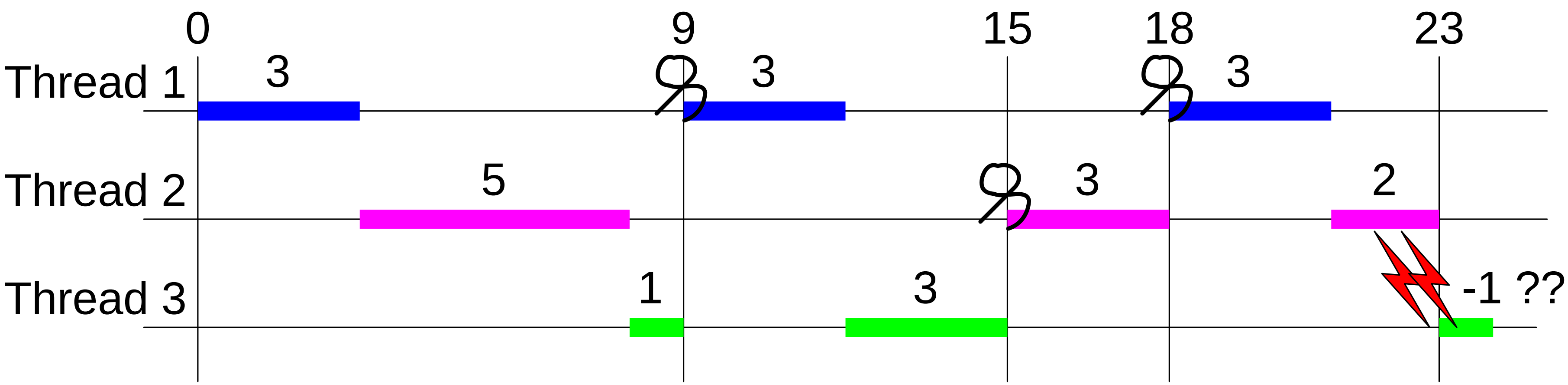
Thread	Period = deadline	Processing	Load
Thread 1	9	3	33.3%
Thread 2	15	5	33.3%
Thread 3	23	5	21.7%
			88.3%



Source: Ton Kostelijk - EXARCH course

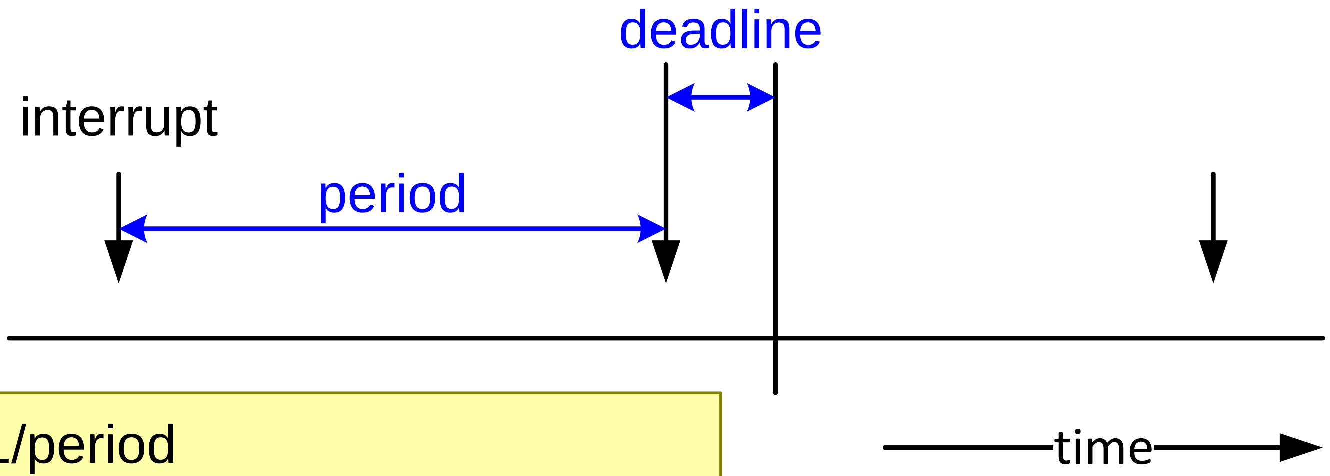
Answers: loads and thread activity (RMS)

Thread	Period = deadline	Processing	Load
Thread 1	9	3	33.3%
Thread 2	15	5	33.3%
Thread 3	23	5	21.7%
			88.3%



Source: Ton Kostelijk - EXARCH course

Extensions of the Application of RMS



if deadline <> 1/period

then use period = 1/deadline

if CPU consumption varies

then use worst case CPU consumption

More advanced techniques are available, for instance in case of "nice" frequencies

Summary

Theory Hard Real Time Scheduling

Earliest Deadline First (EDF):

optimal according theory, but practical not applicable due to overhead

Rate Monotonic Scheduling (RMS):

provides recipe to assign priorities to tasks

results in predictable real time behavior

works well, even outside theoretical constraints

Exercise

Measurement of file transfers with different HTTP, FTP, Windows filesystem, on fast and slow networks

Soft Real Time

Navigation Case to be inserted here

Home work

Assignment for next block

Summary

to be inserted here

Home work reporting

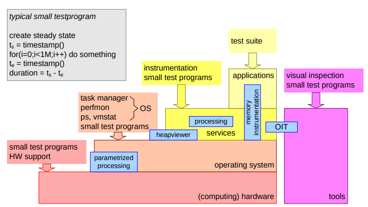
Exploring an existing code base: measurements and instrumentation

by *Gerrit Muller* University of South-Eastern Norway-NISE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

Abstract

Many architects struggle with a given large code-base, where a lot of knowledge about the code is in the head of people or worse where the knowledge has disappeared. One of the means to recover insight from a code base is by measuring and instrumenting the code-base. This presentation addresses measurements of the static aspects of the code, as well as instrumentation to obtain insight in the dynamic aspects of the code.

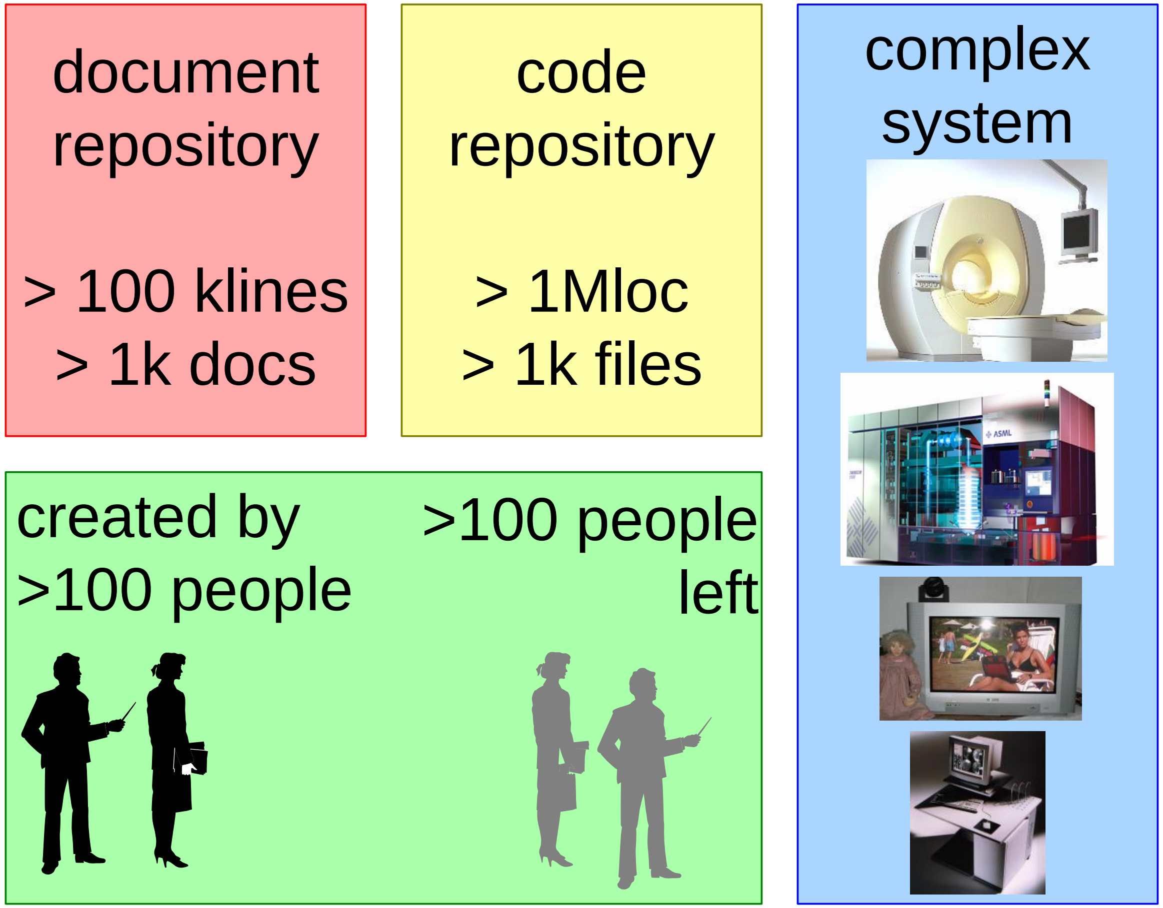
March 6, 2021
status: draft
version: 0.4



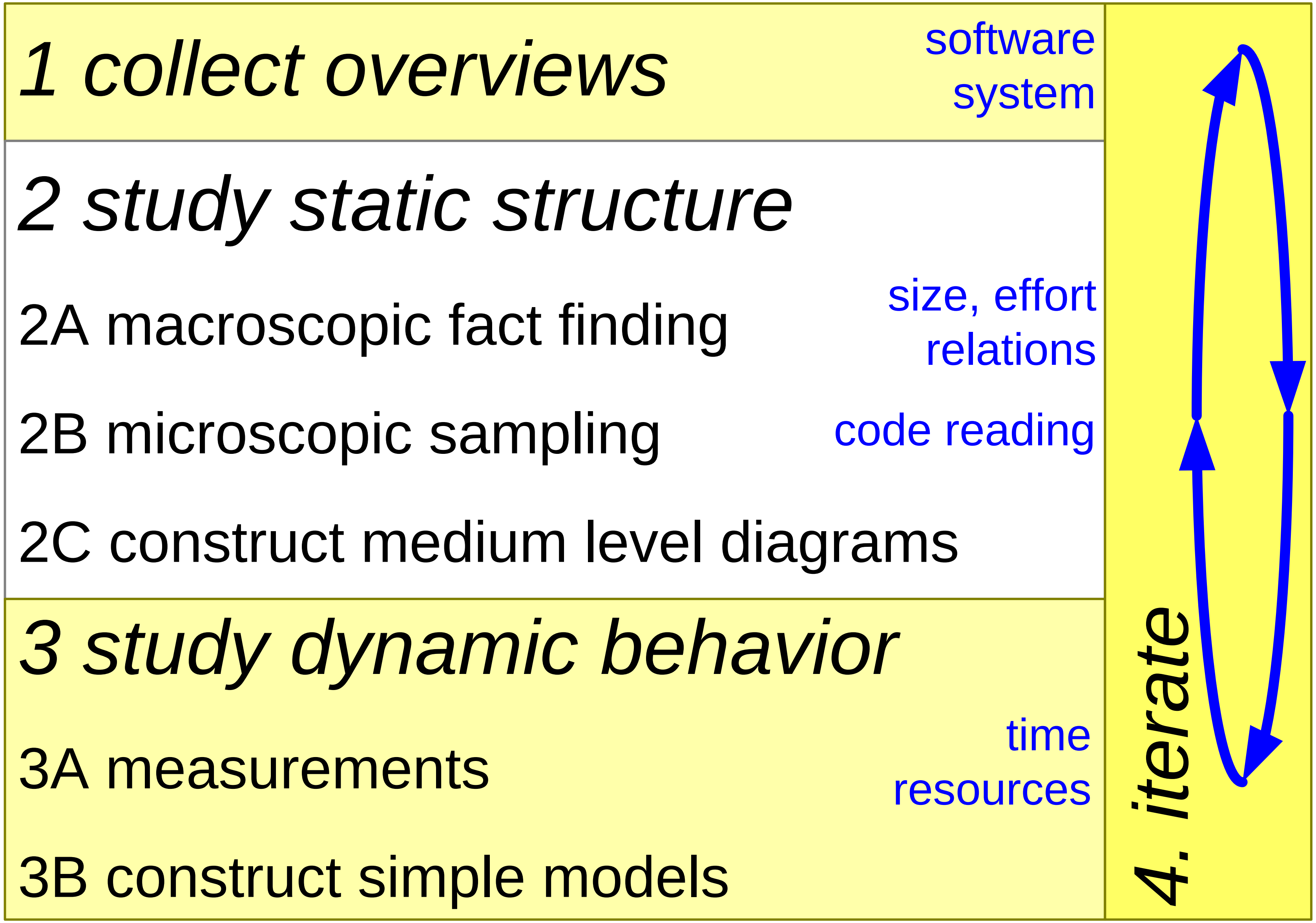
Problem Statement

wanted:
new functions and interfaces, higher performance levels, improvements, et cetera

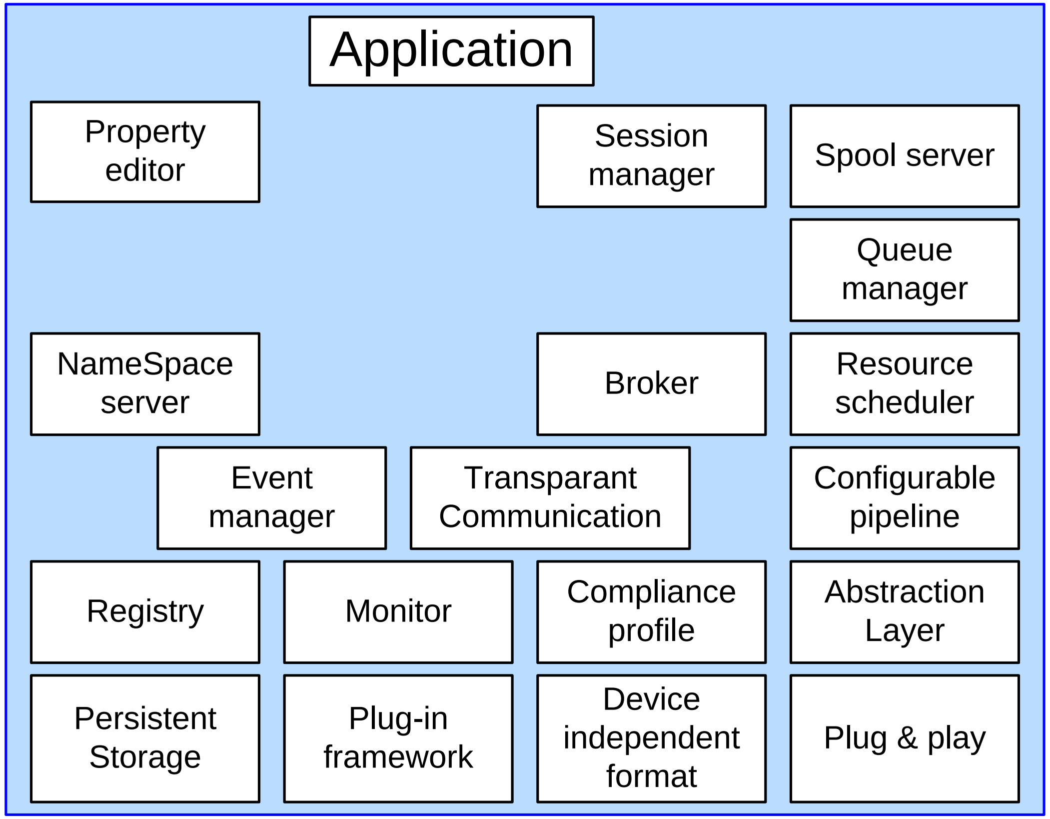
given:



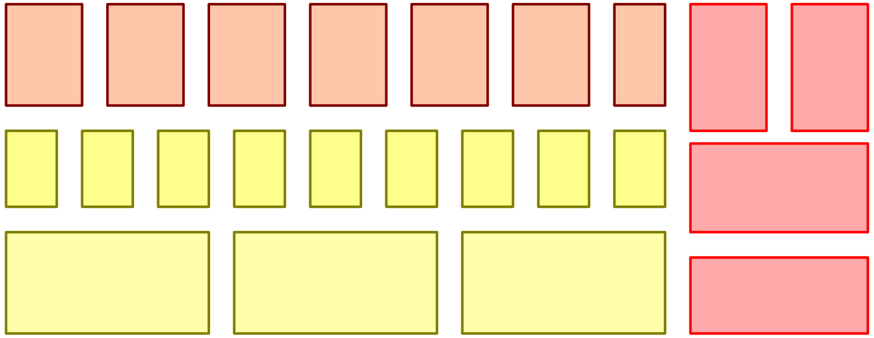
Overview of Approach and Presentation Agenda



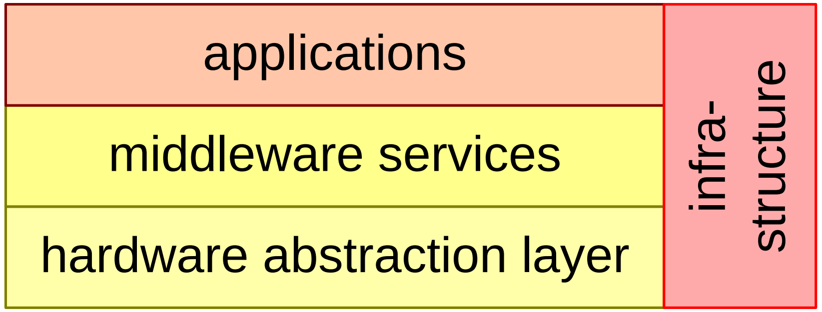
SW Overview(s)



mechanism centric



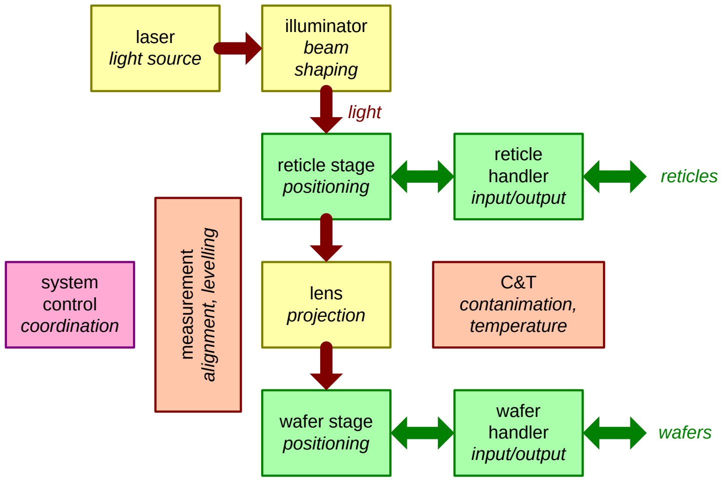
delivery centric



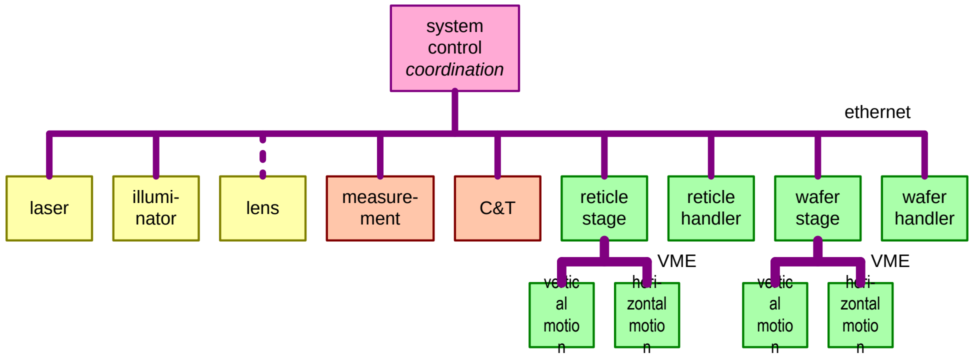
(over)simplistic

System Overviews

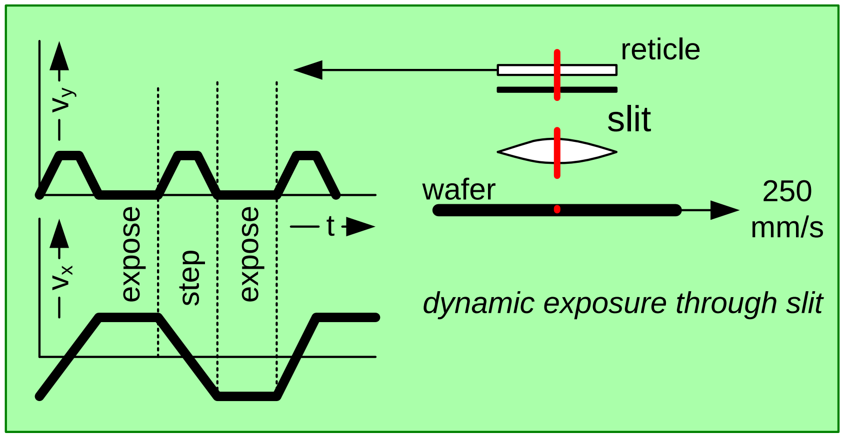
subsystems



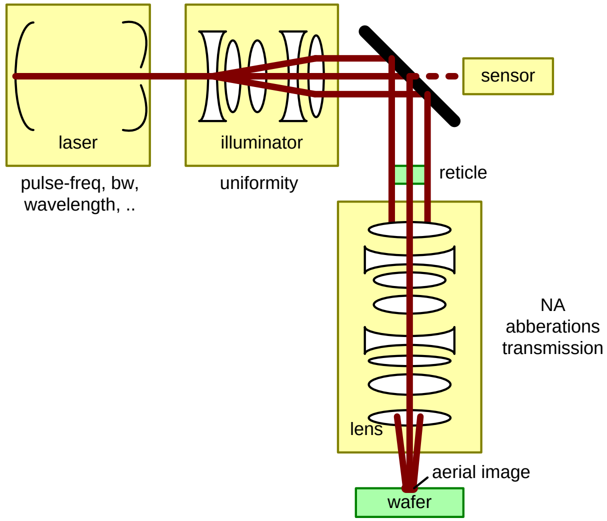
control hierarchy



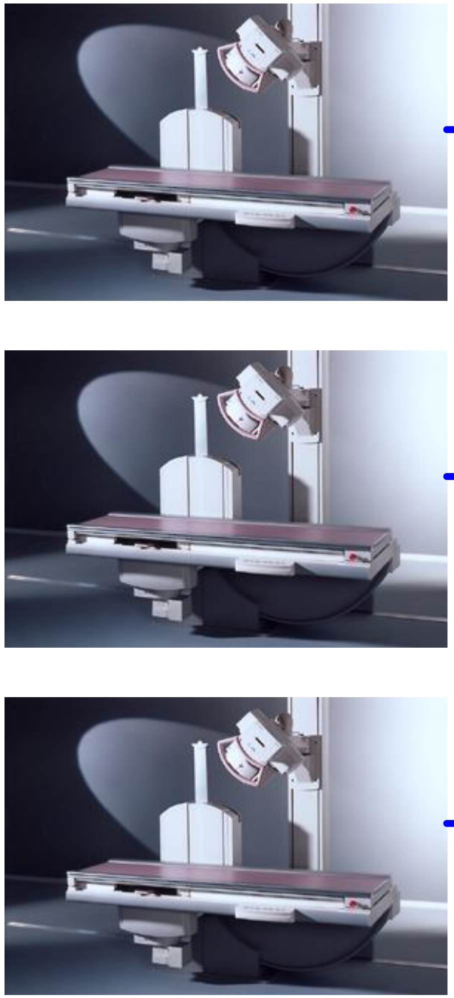
kinematic

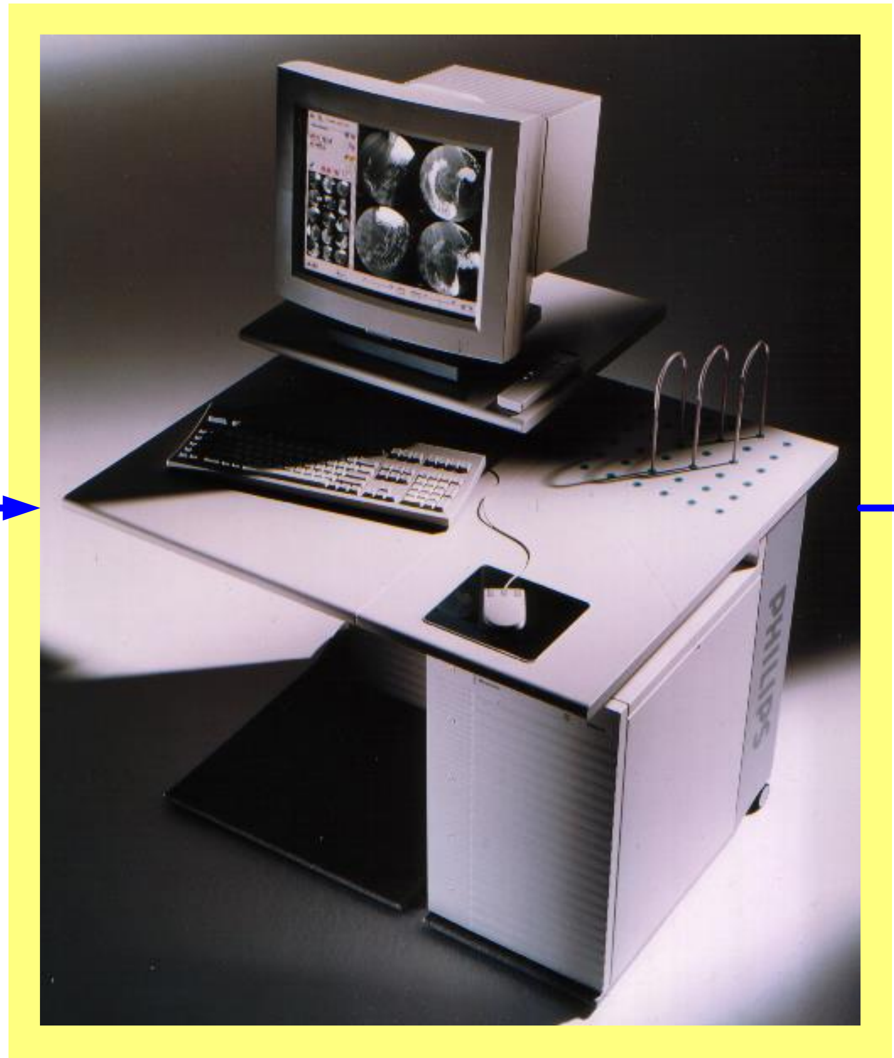


physics/optics



Case 1: EasyVision (1992)





typical clinical image (intestines)

URF-systems

EasyVision: Medical Imaging Workstation

Examples of Macroscopic Fact Finding

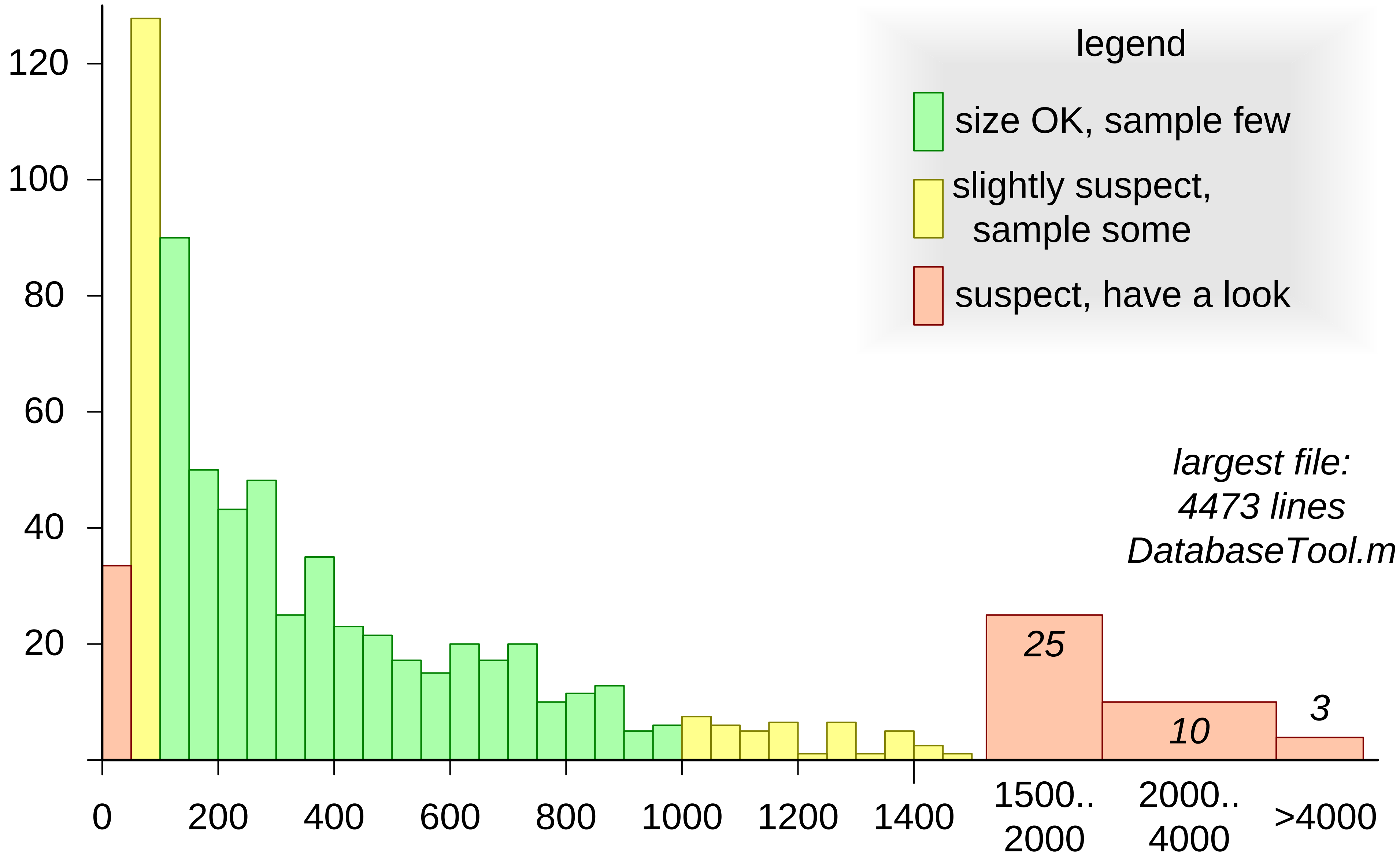
```
> wc -l  *.m
72 Acquisition.m
13 AcquisitionFacility.m
330 ActiveDataCollection.m
132 ActiveDataObject.m
304 Activity.m
281 ActivityList.m
551 AnnotateParser.m
1106 AnnotateTool.m
624 AnyOfList.m
466 AsyncBulkDataIO.m
264 AsyncDeviceIO.m
261 AsyncLocalDbIO.m
334 AsyncRemoteDbIO.m
205 AsyncSocketIO.m
```

version control information:
#new files
#deleted files
#changes per file since ...

package information:
files

metrics:
QAC type information
methods
globals

Histogram of File Sizes EV R1.0



Microscopic Sampling (Code Reading)

Example of small classes due to database design; These classes are only supporting constructs

13 IndexBtree.m
12 IndexInteriorNode.m
13 IndexLeafNode.m
13 ObjectStoreBtree.m
12 ObjectStoreInteriorNode.m
13 ObjectStoreLeafNode.m

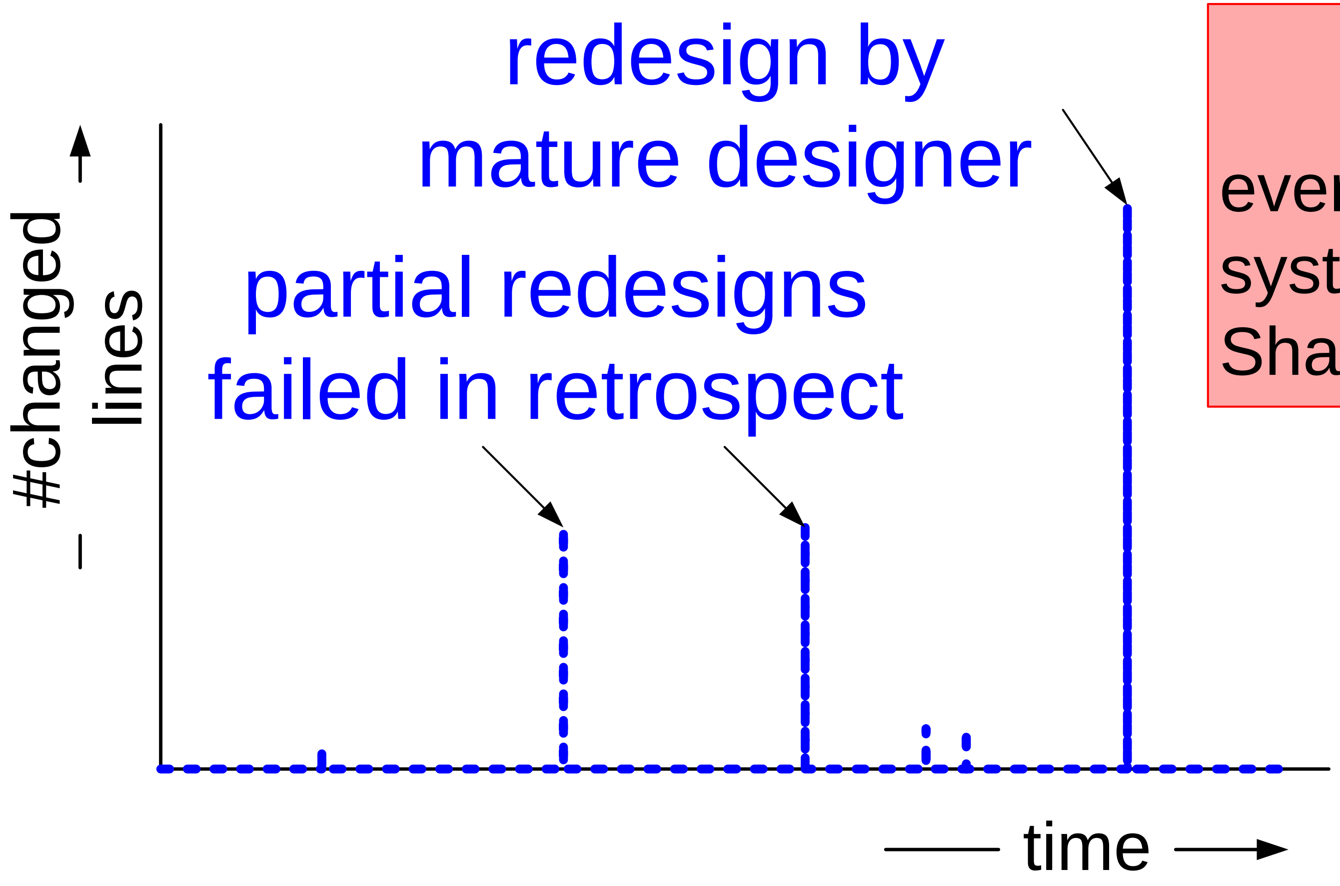
Example of large classes due to large amount of UI details

4473 DatabaseTool.m
1291 EnhancementTool.m
1106 AnnotateTool.m
1291 EnhancementTool.m
3471 GreyLevelTool.m
1639 HCConfigurationTool.m
1007 HCQueueViewingTool.m
1590 HardcopyTool.m

Example of large classes due to inherent complexity; some of these classes are really suspect

1541 GenericRegion.m
1415 GfxArea.m
1697 GfxFreeContour.m
4095 GfxObject.m
1714 GfxText.m
1374 CVObject.m
1080 ChartStack.m
1127 Collection.m
1651 Composite.m
1725 CompositeProjectionImage.m
1373 Connection1.m
1181 Database1.m
3707 DatabaseClient.m
3240 Image.m
1861 ImageSet.m

Changes Over Time

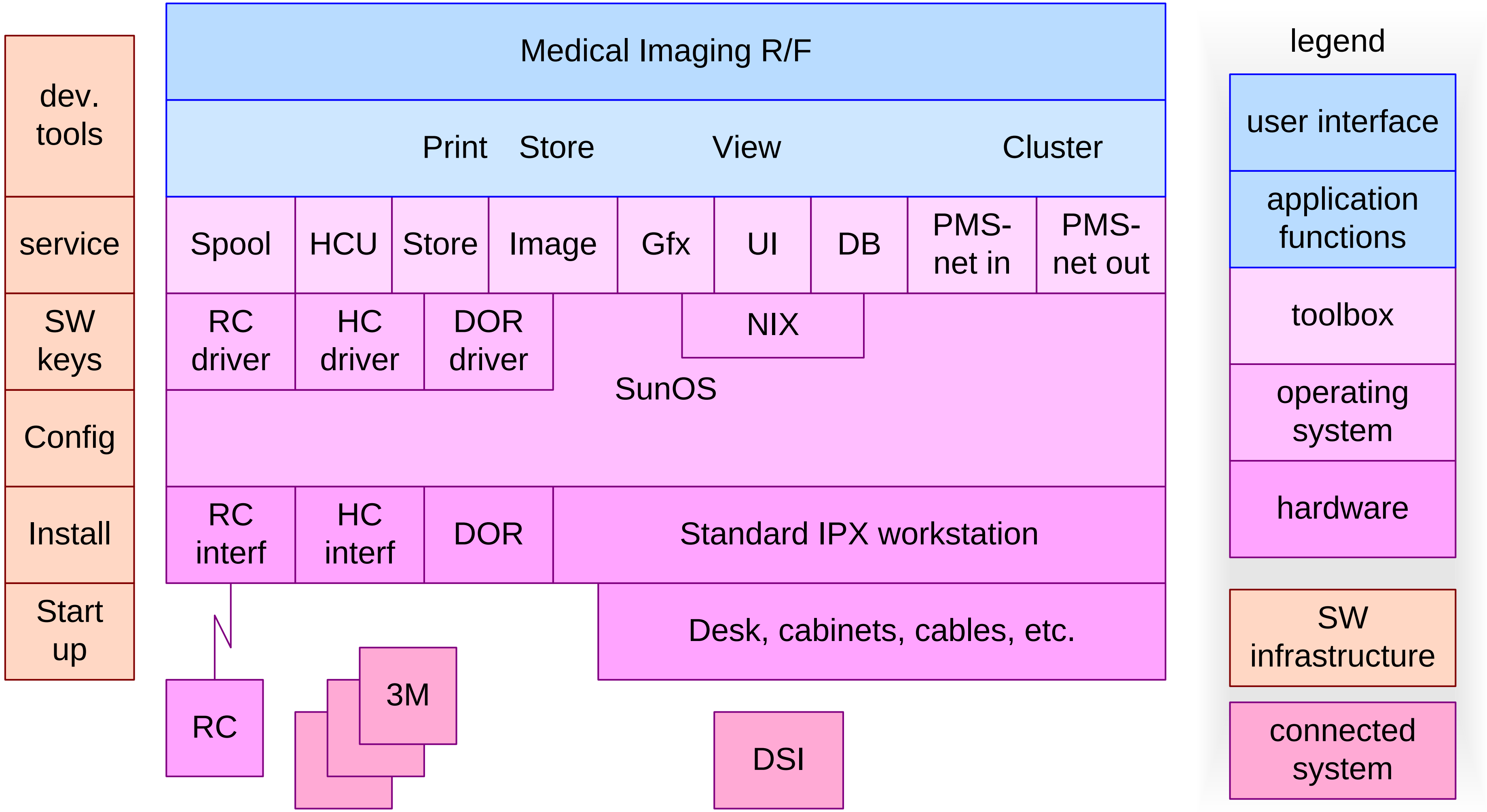


hot spots

ever changing files e.g.:
systemConstants.h
ShakyImplementation.m

Simplified Medium Level Diagram

The real layering diagram did have >15 layers



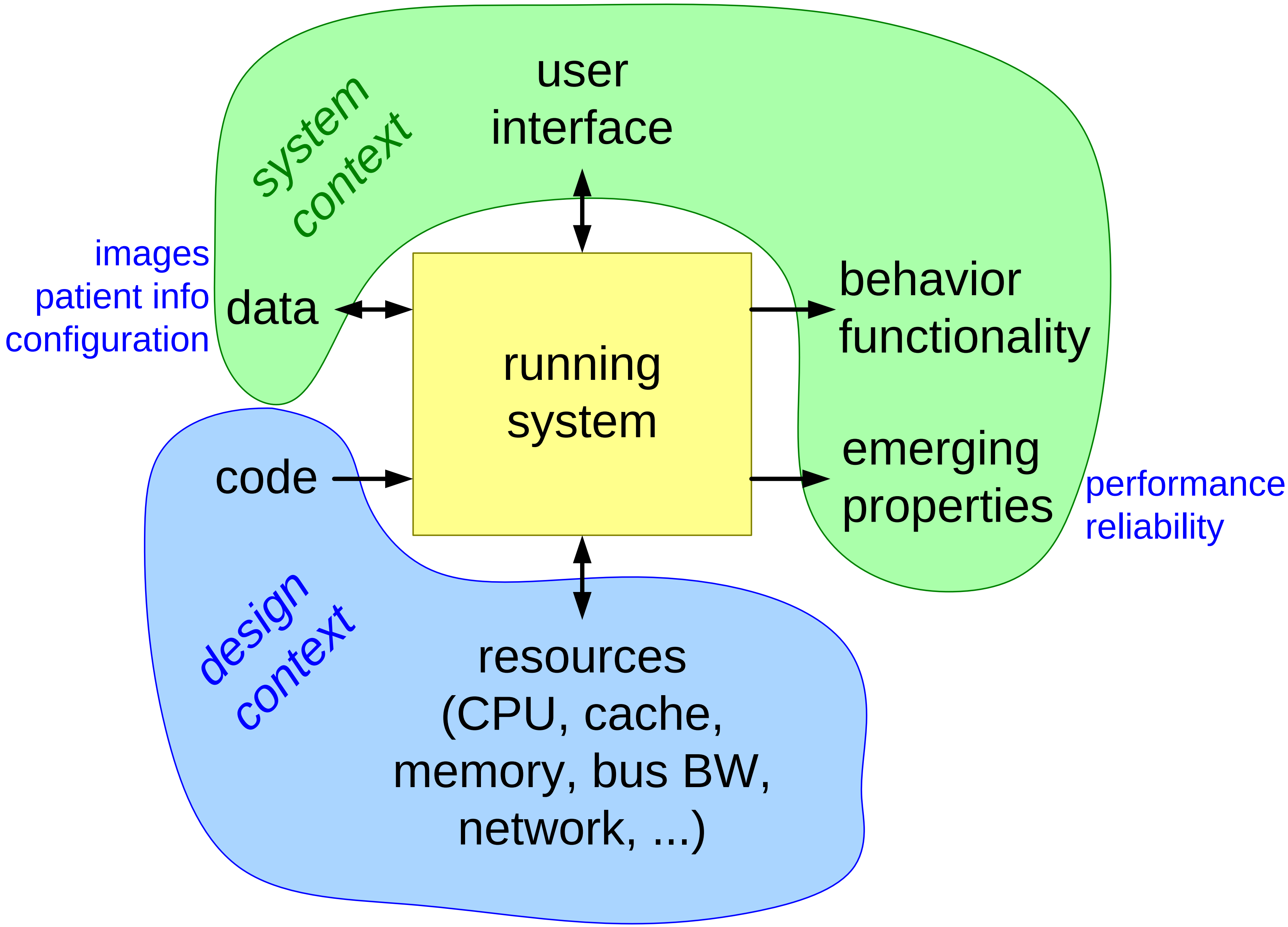
Conclusions Static Exploration

Quantification helps to *calibrate* the *intuition* of the architect

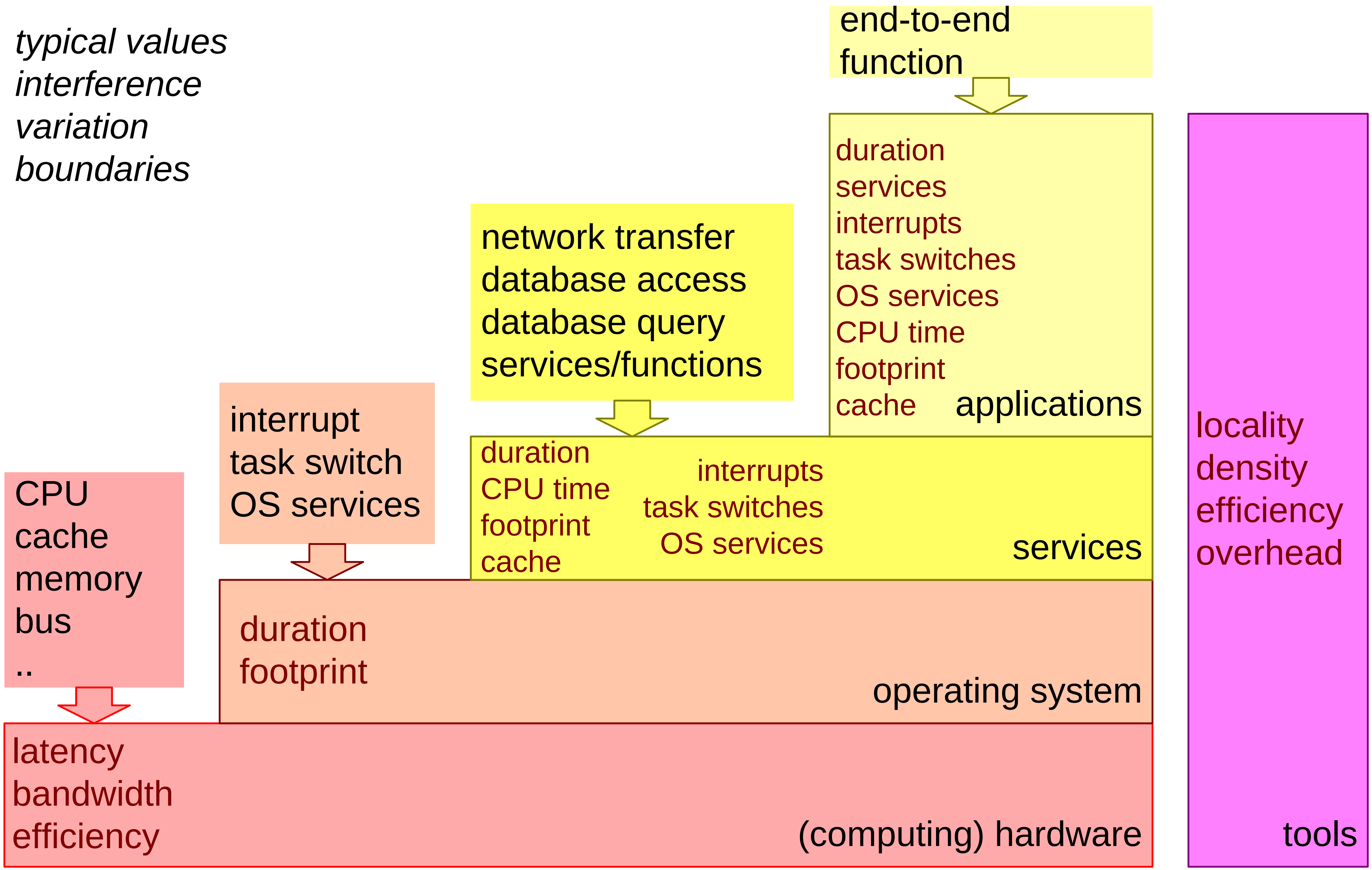
Macroscopic numbers related to *code level* understanding provides insight

+ relative complexity
+ relative effort
+ hot spots
+ (static) dependencies and relations

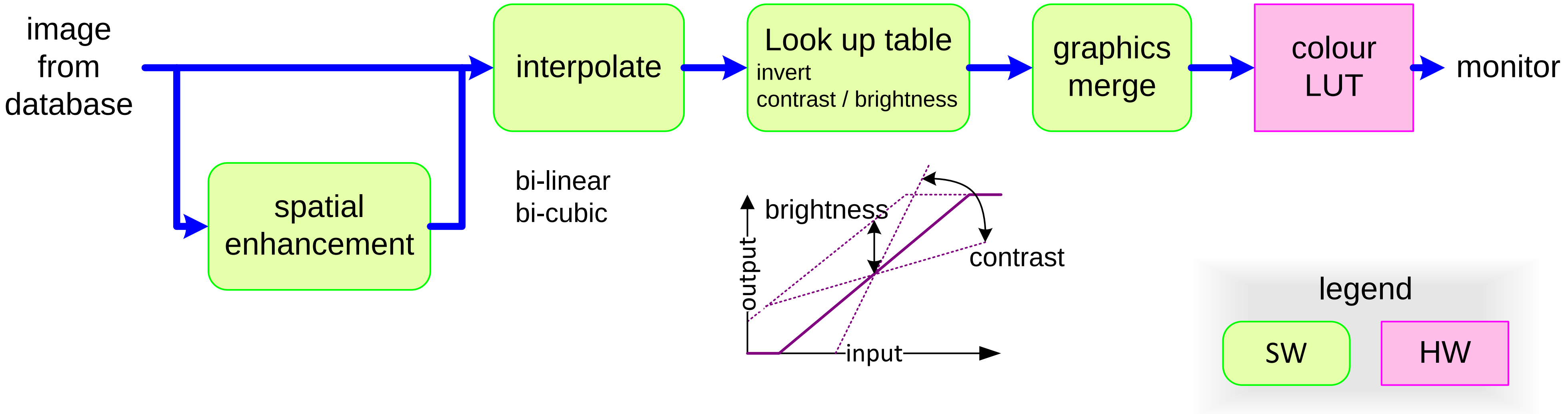
Dynamics ≫ Static



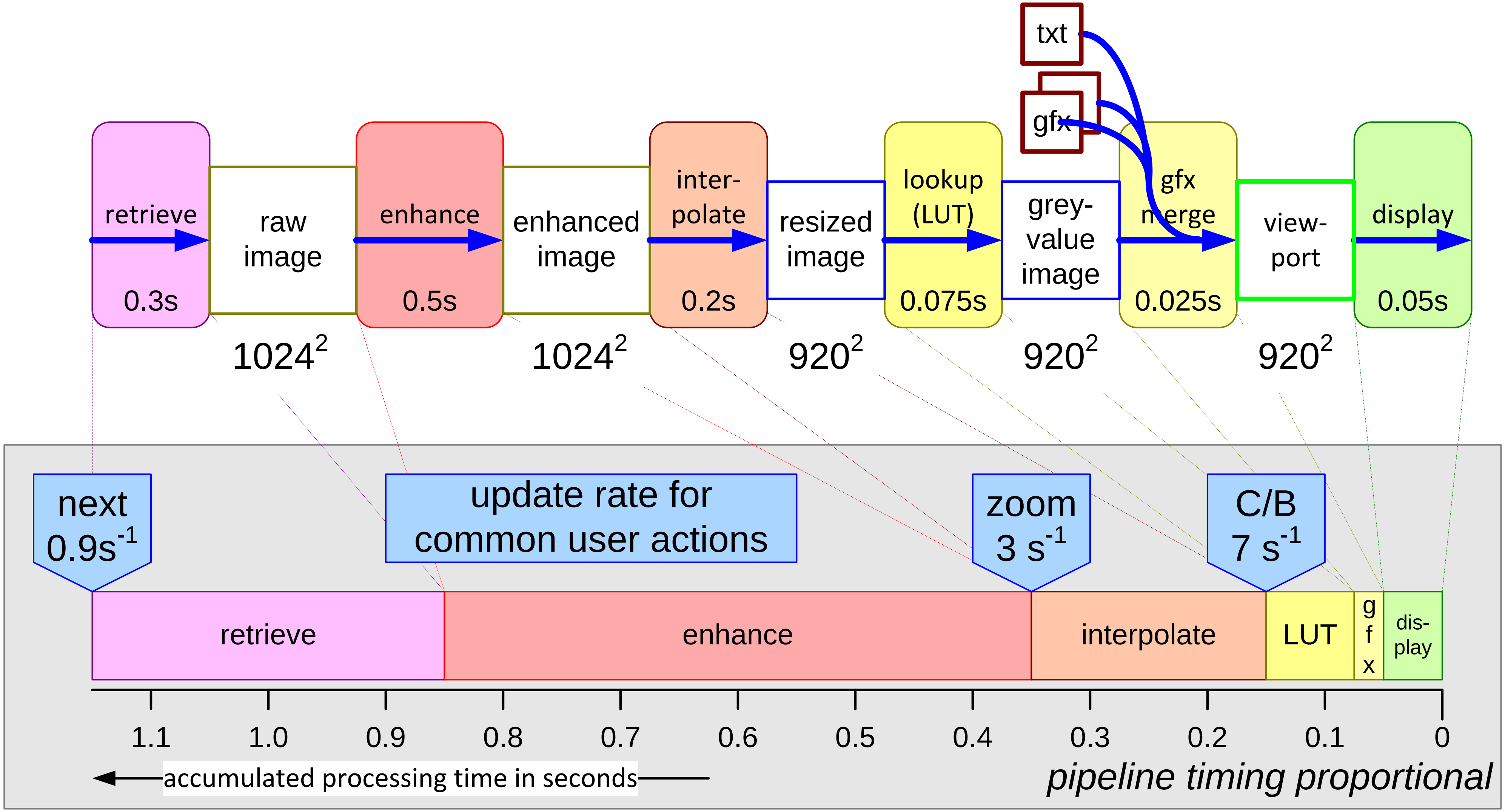
Layered Benchmarking



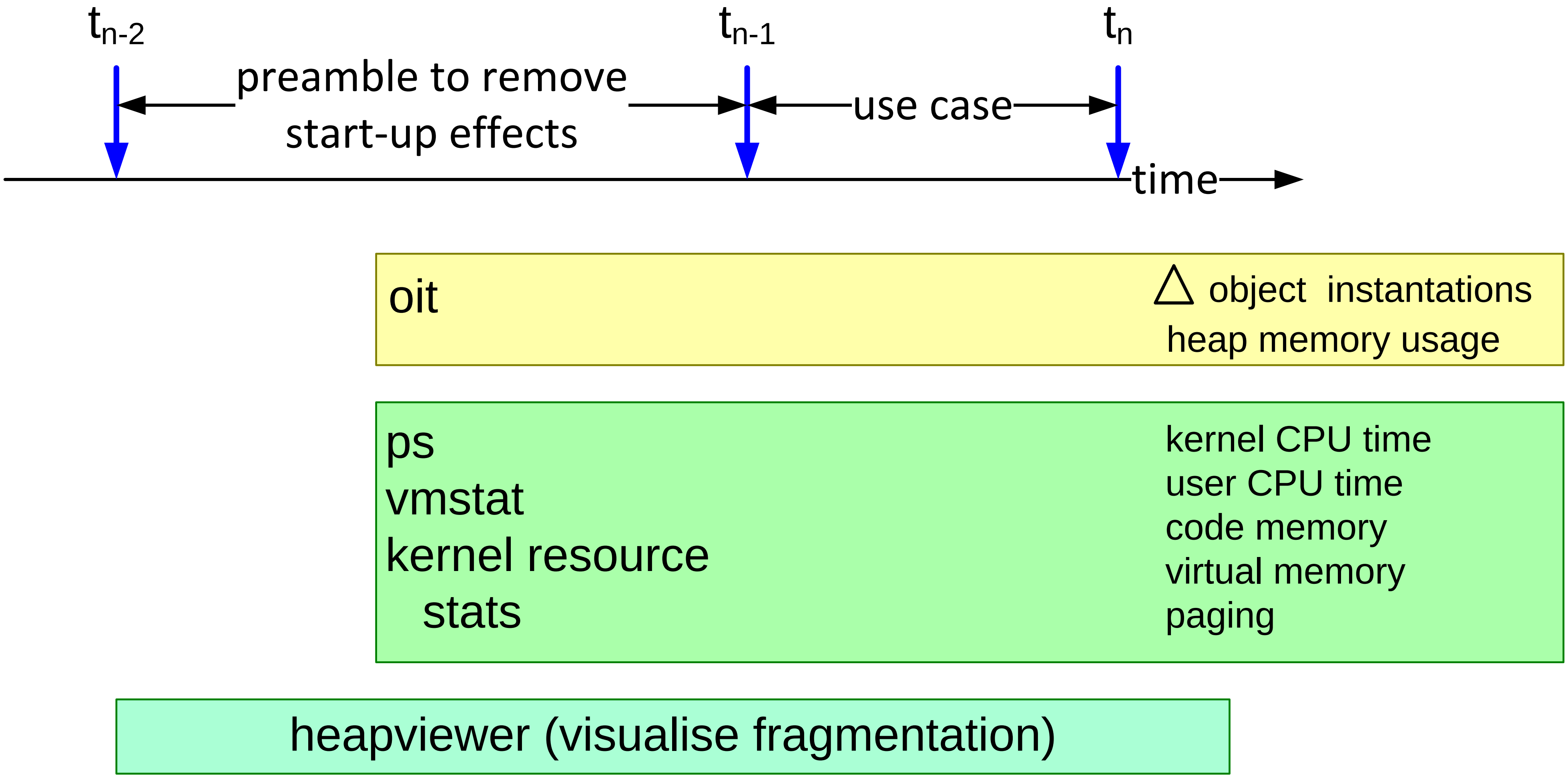
Example: Processing HW and Service Performance



Processing Performance



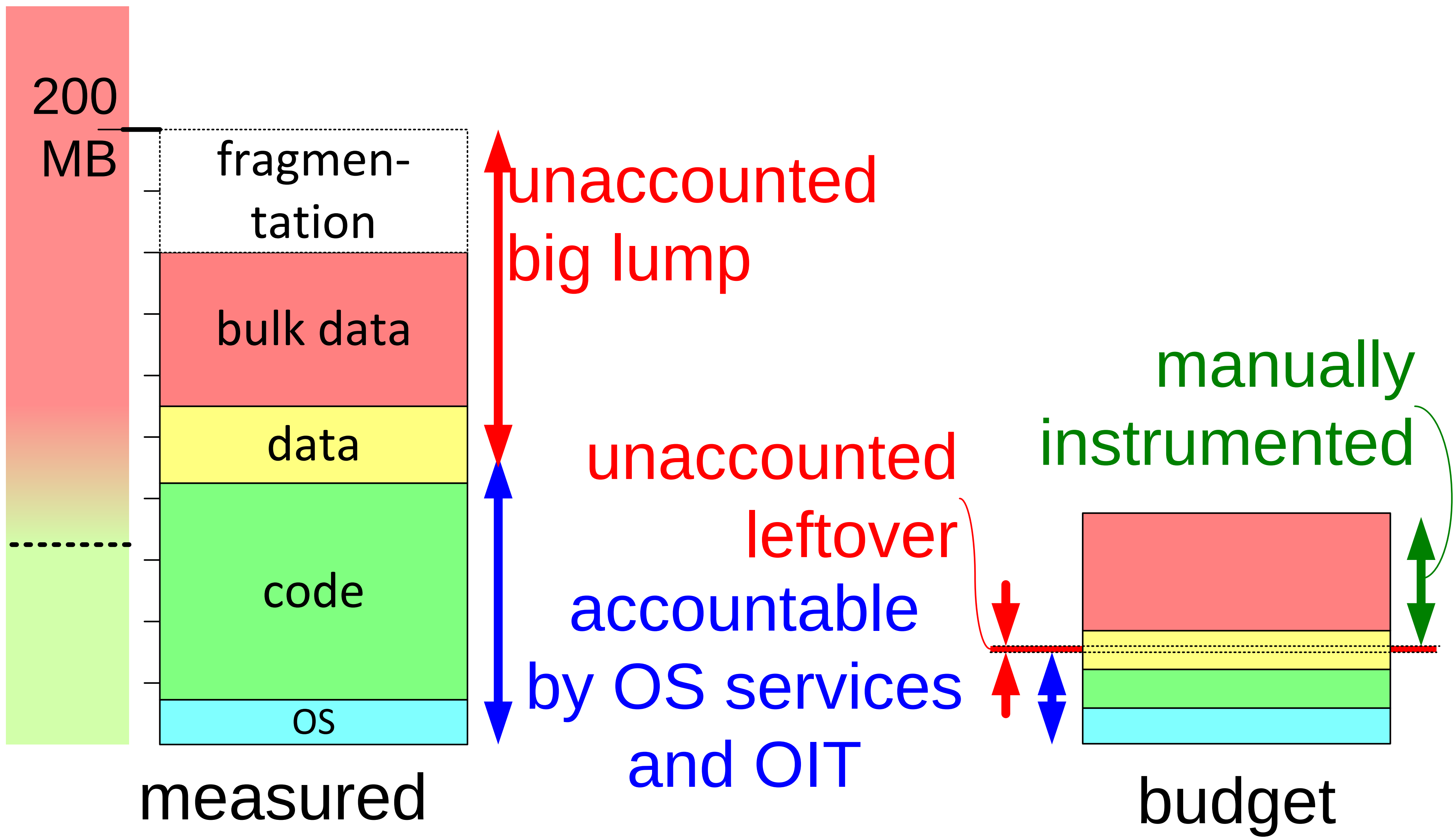
Resource Measurement Tools



Object Instantiation Tracing

class name	current nr of objects	deleted since t_{n-1}	created since t_{n-1}	heap memory usage
AsynchronousIO	0	-3	+3	
AttributeEntry	237	-1	+5	
BitMap	21	-4	+8	
BoundedFloatingPoint	1034	-3	+22	
BoundedInteger	684	-1	+9	
BtreeNode1	200	-3	+3	[819200]
BulkData	25	0	1	[8388608]
ButtonGadget	34	0	2	
ButtonStack	12	0	1	
ByteArray	156	-4	+12	[13252]

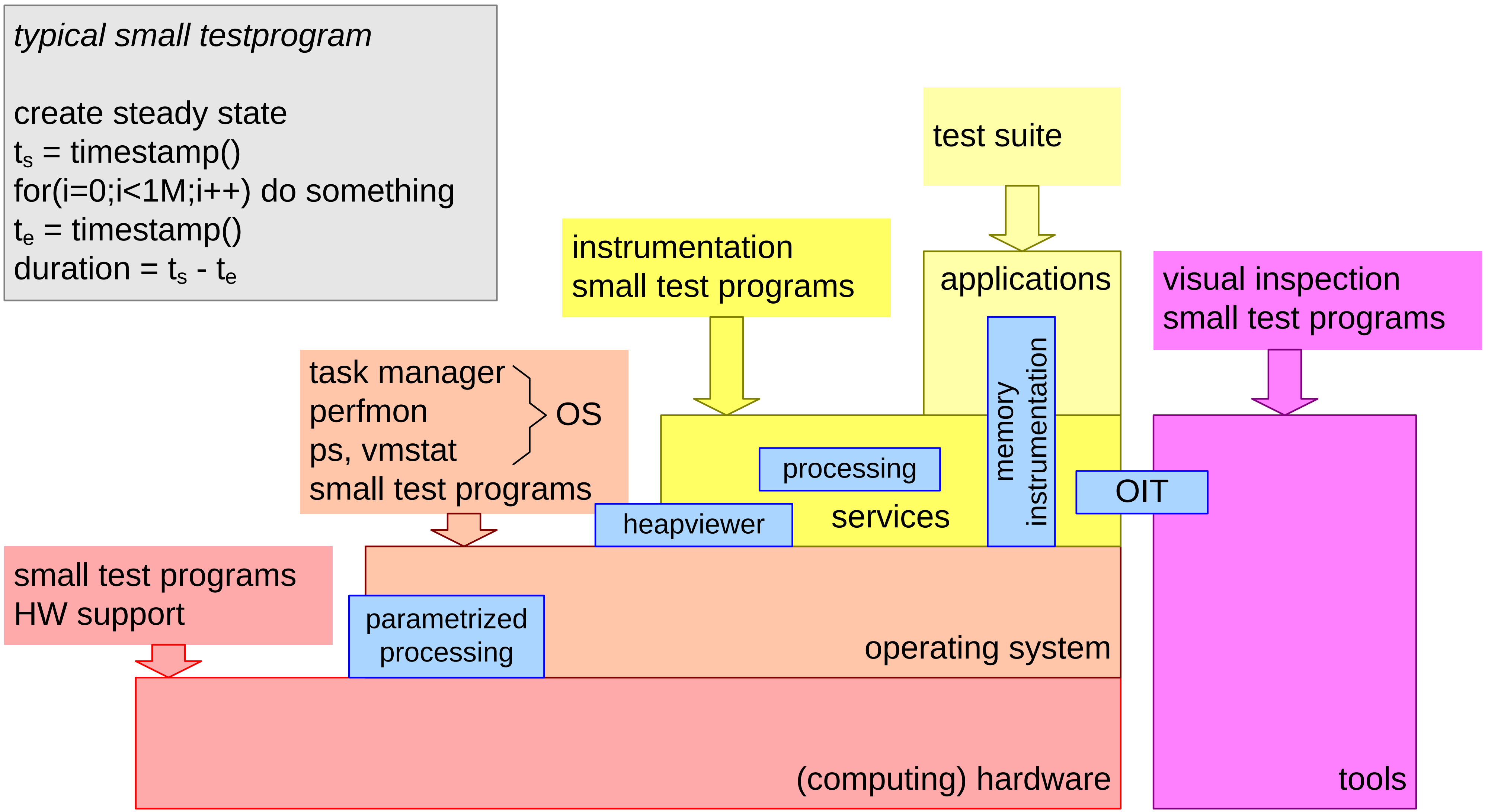
Memory Instrumentation



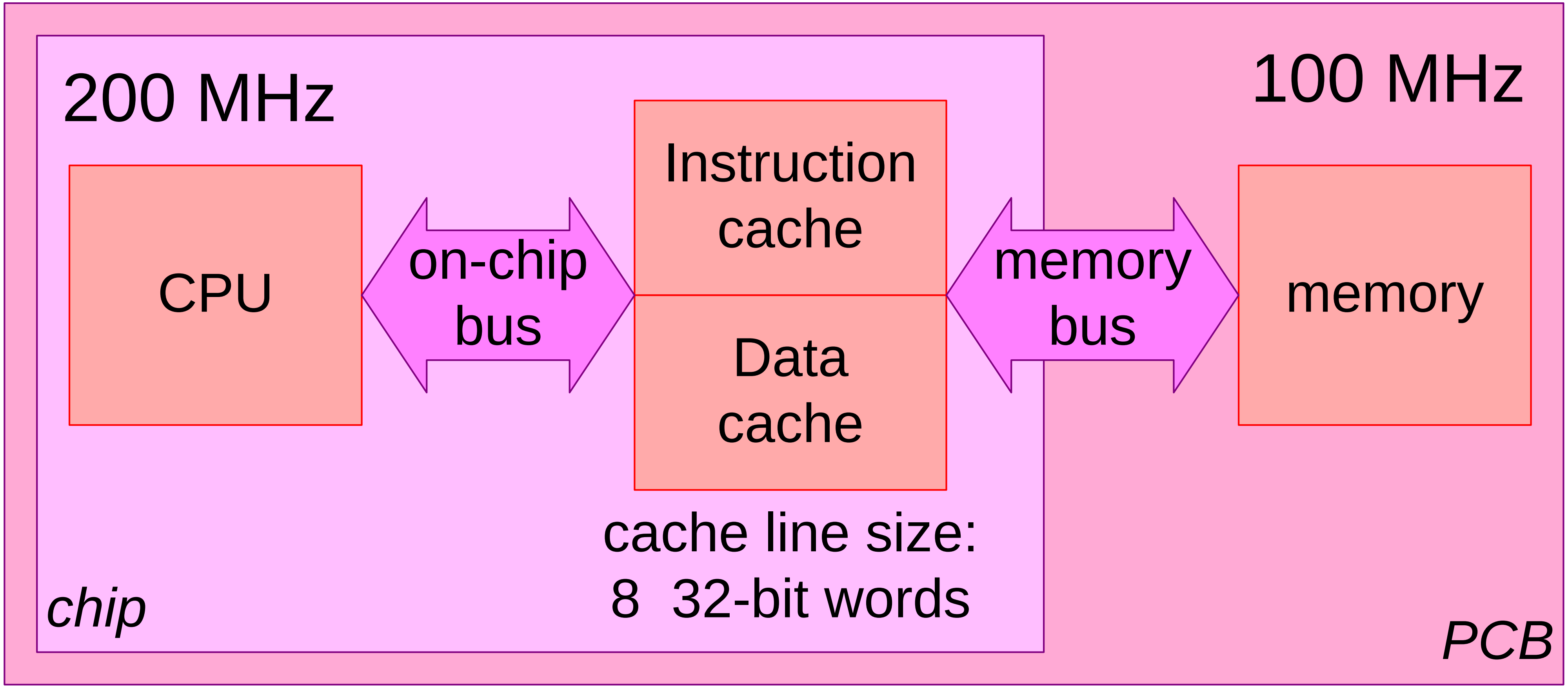
Overview of Benchmarks and Other Measurement Tools

	test / benchmark	what, why	accuracy	when
public	SpecInt (by suppliers)	CPU integer	coarse	new hardware
public	Byte benchmark	computer platform performance OS, shell, file I/O	coarse	new hardware new OS release
self made	file I/O	file I/O throughput	medium	new hardware
self made	image processing	CPU, cache, memory as function of image, pixel size	accurate	new hardware
self made	Objective-C overhead	method call overhead memory overhead	accurate	initial
self made	socket, network	throughput CPU overhead	accurate	ad hoc
self made	data base	transaction overhead query behaviour	accurate	ad hoc
self made	load test	throughput, CPU, memory	accurate	regression

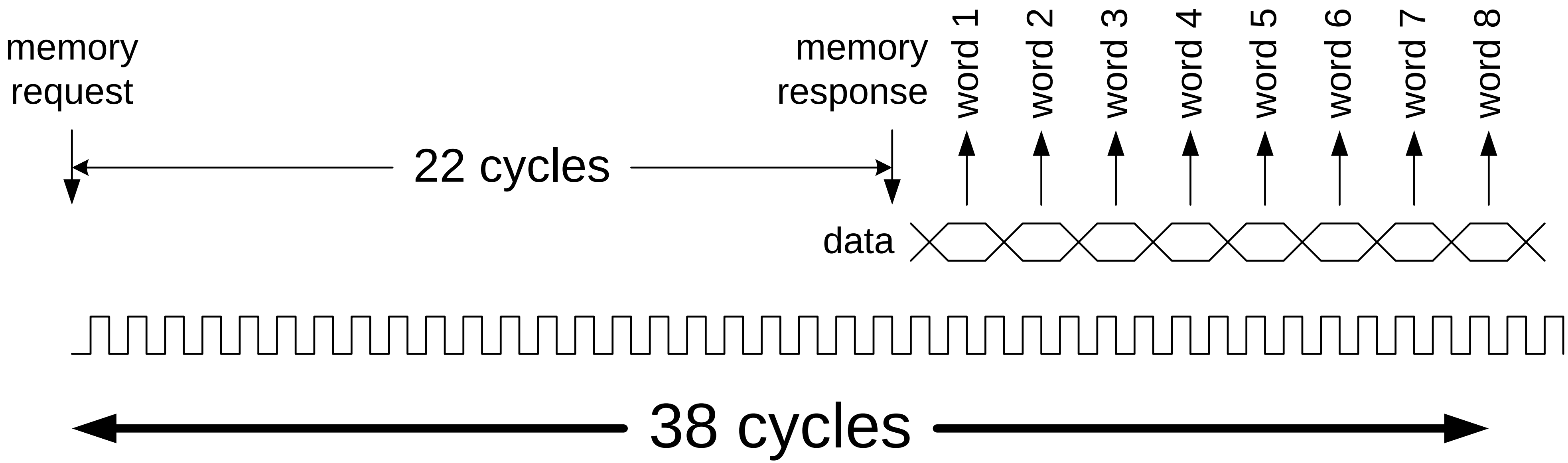
Tools and Instruments Positioned in the Stack



Case 2: ARM9 Cache Performance



Example Hardware Performance



memory access time in case of a cache miss
200 Mhz, 5 ns cycle: 190 ns

ARM9 200 MHz $t_{context\ switch}$ as function of cache use

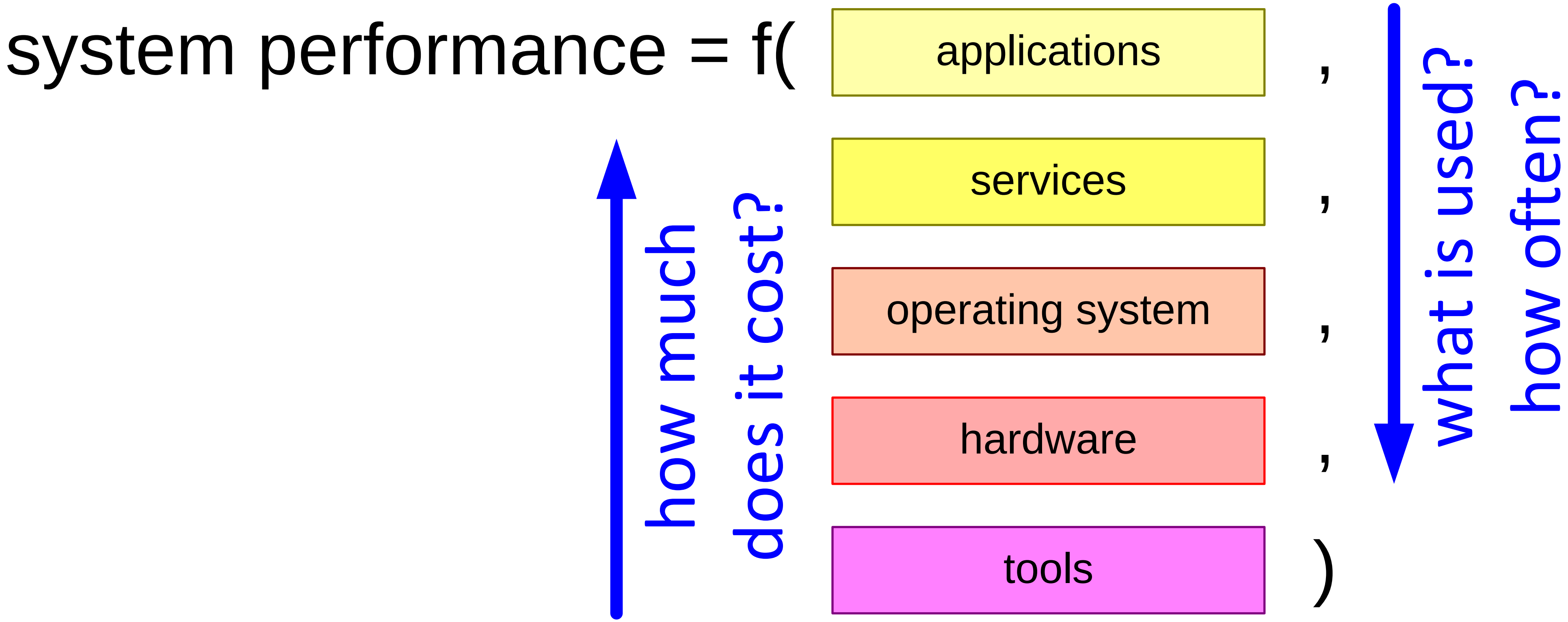
cache setting	$t_{context\ switch}$
From cache	2 µs
After cache flush	10 µs
Cache disabled	50 µs

Context Switch Overhead

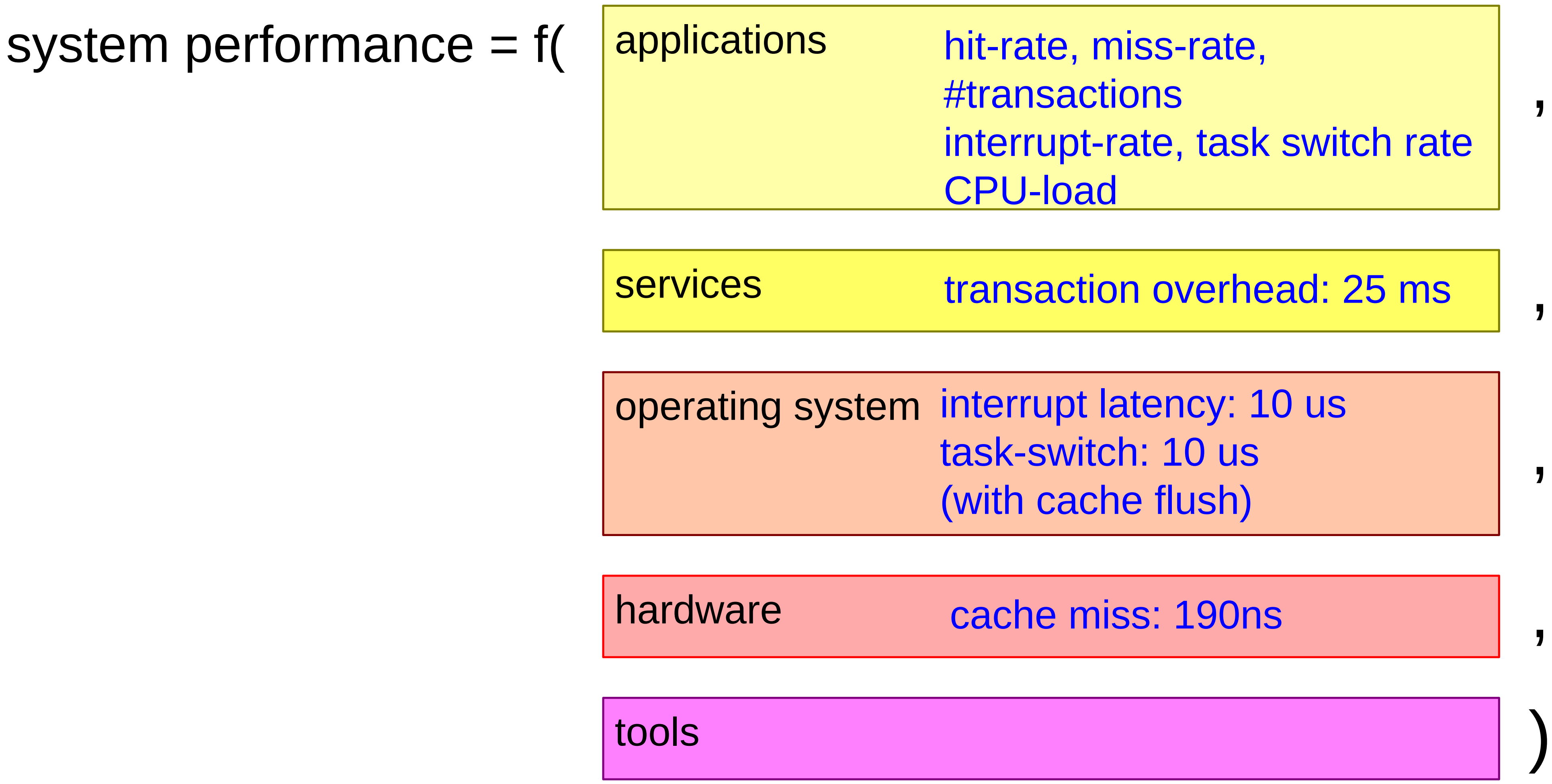
$$t_{overhead} = n_{context\ switch} * t_{context\ switch}$$

$n_{context\ switch}$ (s^{-1})	$t_{context\ switch} = 10\mu s$		$t_{context\ switch} = 2\mu s$	
	$t_{overhead}$	CPU load overhead	$t_{overhead}$	CPU load overhead
500	5ms	0.5%	1ms	0.1%
5000	50ms	5%	10ms	1%
50000	500ms	50%	100ms	10%

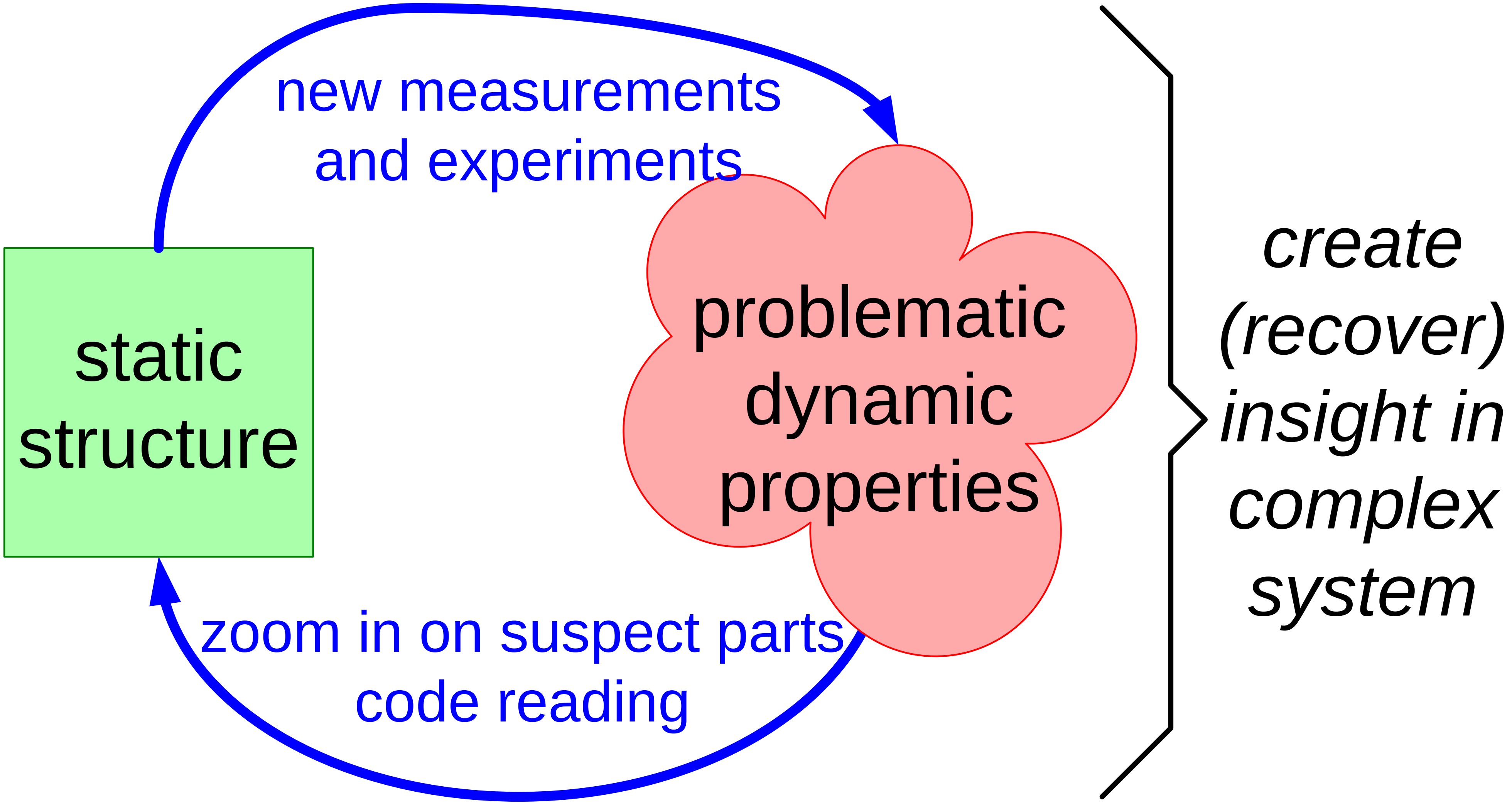
Performance as Function of all Layers



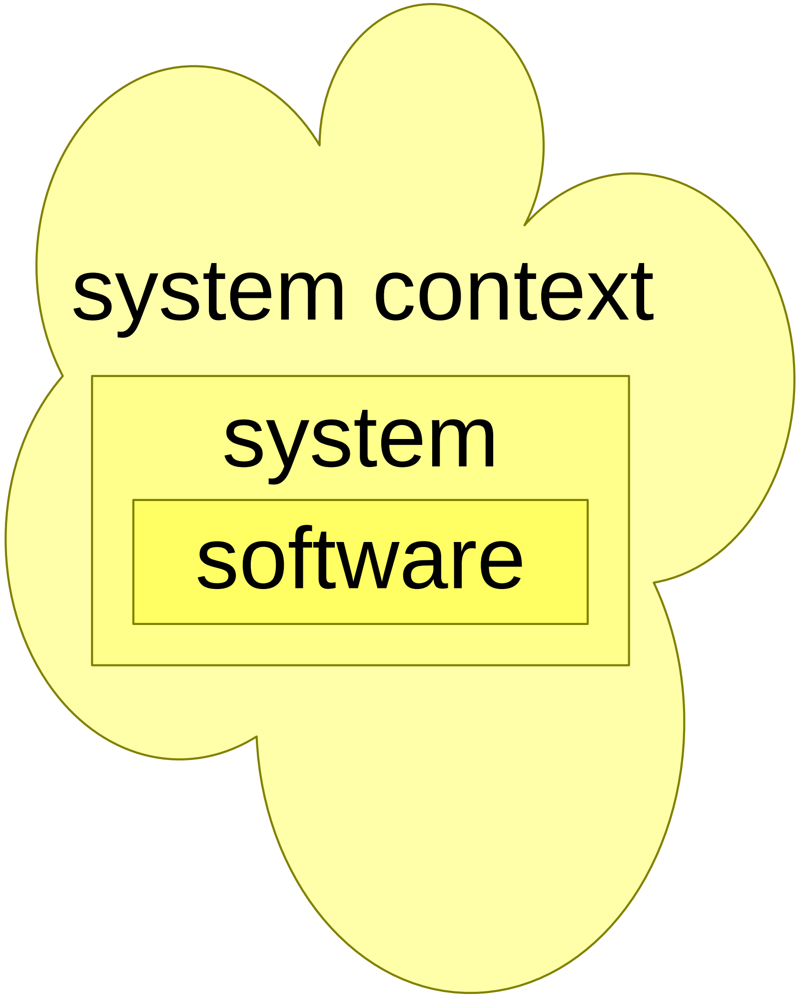
Annotated Performance Formule



Keep iterating!



Discussion propositions



0. many design teams have lost the overview of the system

1. a good (sw) architect has a quantified understanding of system context, system and software

2. a good design facilitates measurements of critical aspects for a small realization effort

Performance Patterns, Pitfalls, and Approach

by *Gerrit Muller* HSN-NISE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

Abstract

Performance Design is based on the application on many performance oriented patterns. Patterns are a way are to consolidate experience: what solution fits to what problem in what situation? Pitfalls are also a way to consolidate experience: what are common design mistakes?

March 6, 2021
status: preliminary draft
version: 0.1

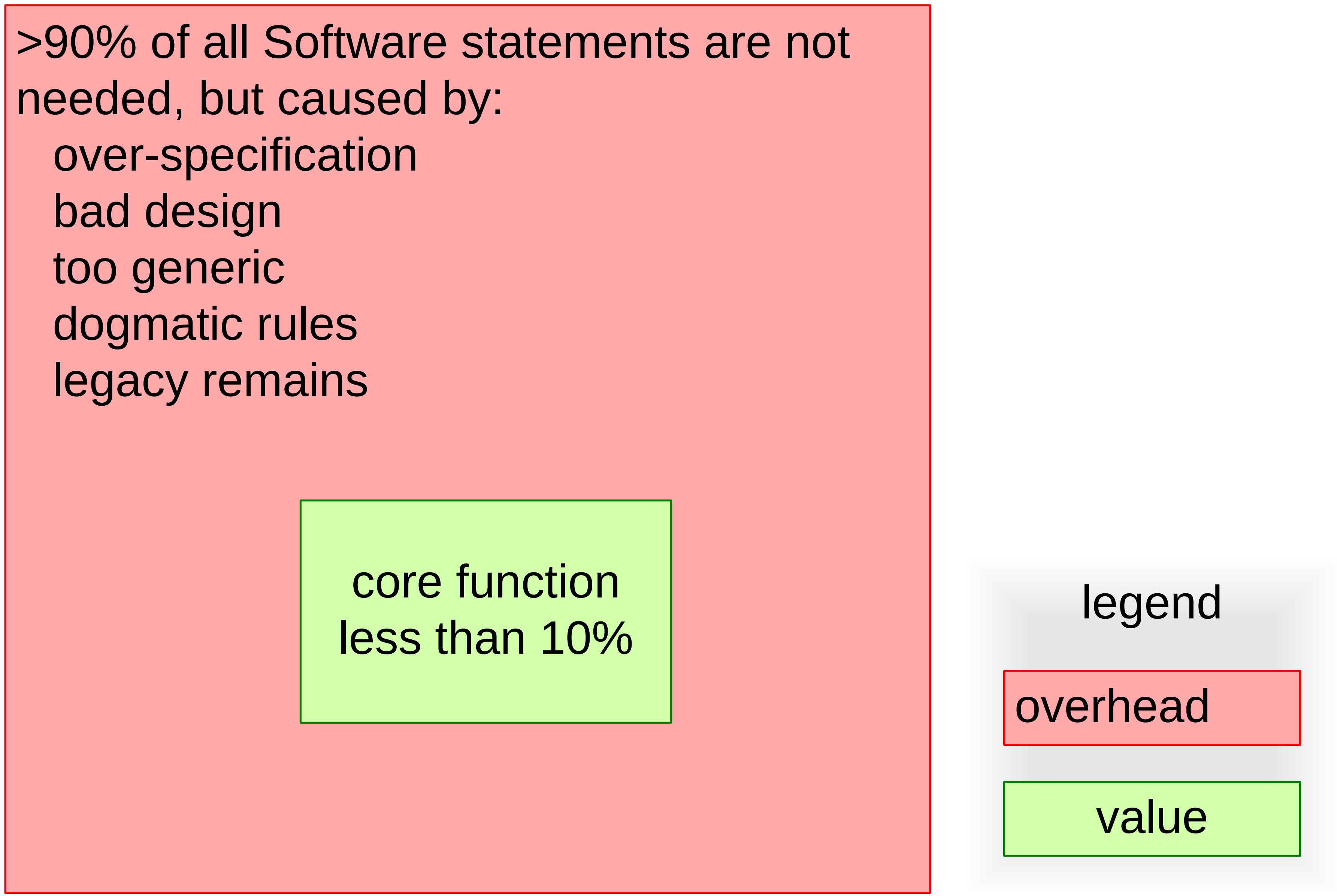
Common Platforms and Bloating

Generic nature of platforms

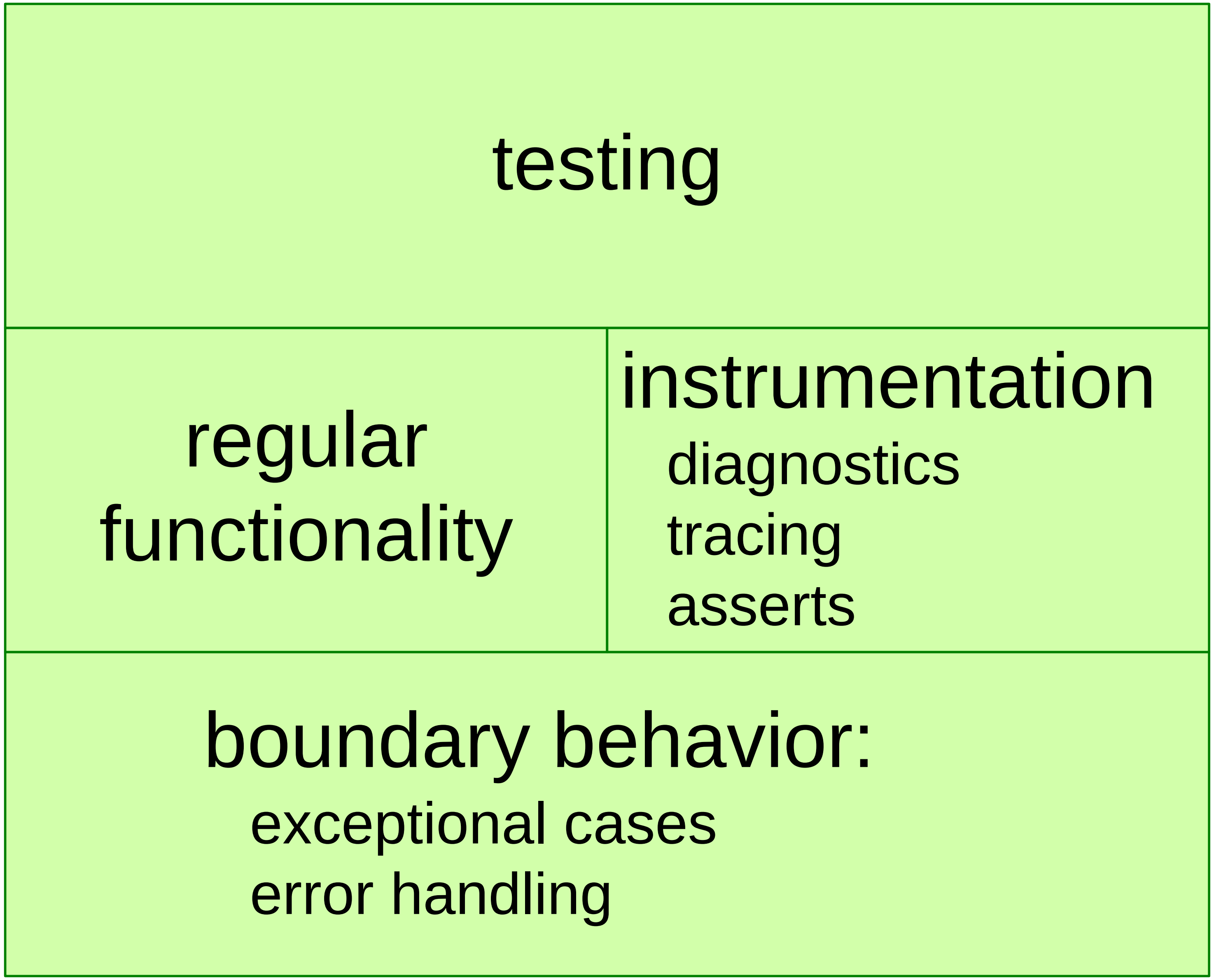
Most SW implementations are way too big

Performance suffers from oversize and generic provisions

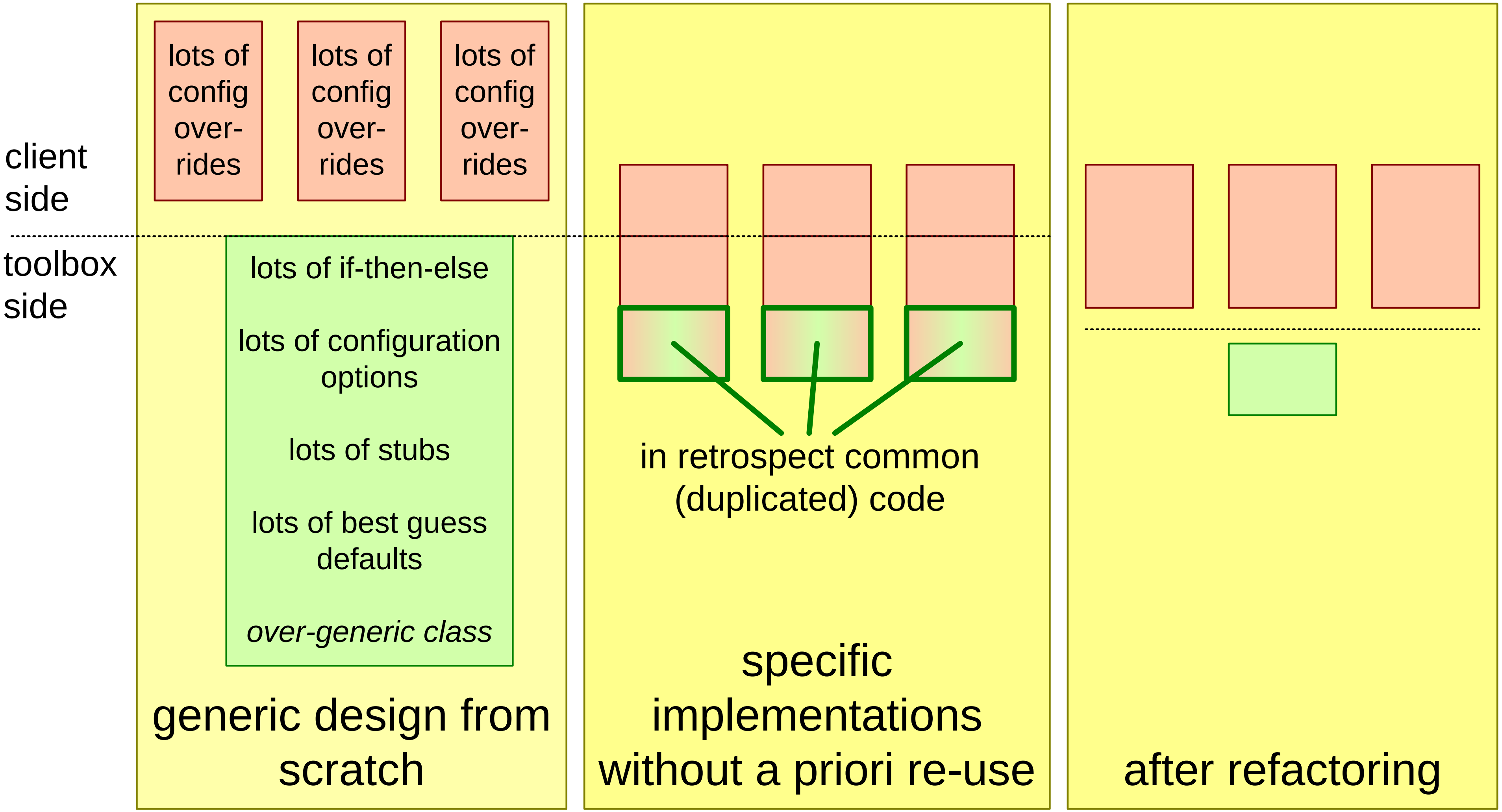
Exploring Bloating: Main Causes



Necessary Functionality ≫ Intended Regular Function

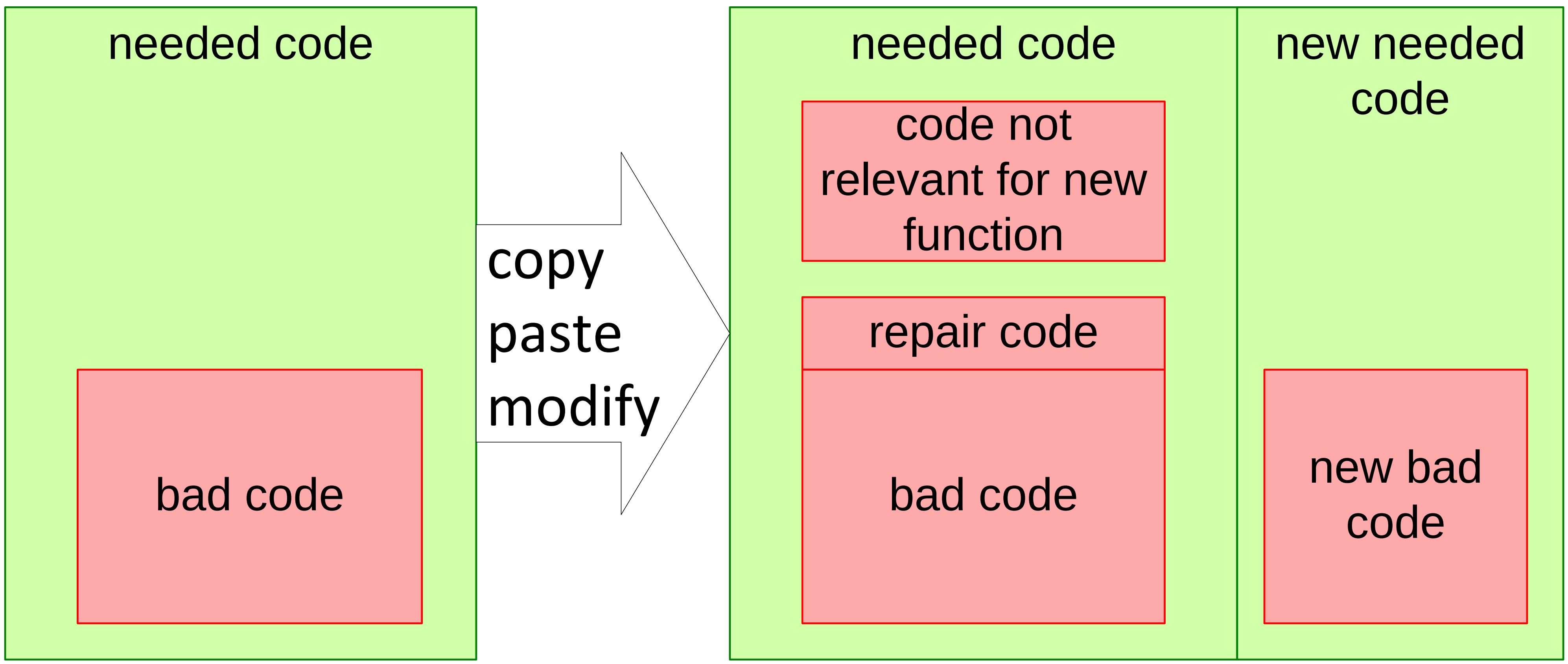


The Danger of Being Generic: Bloating

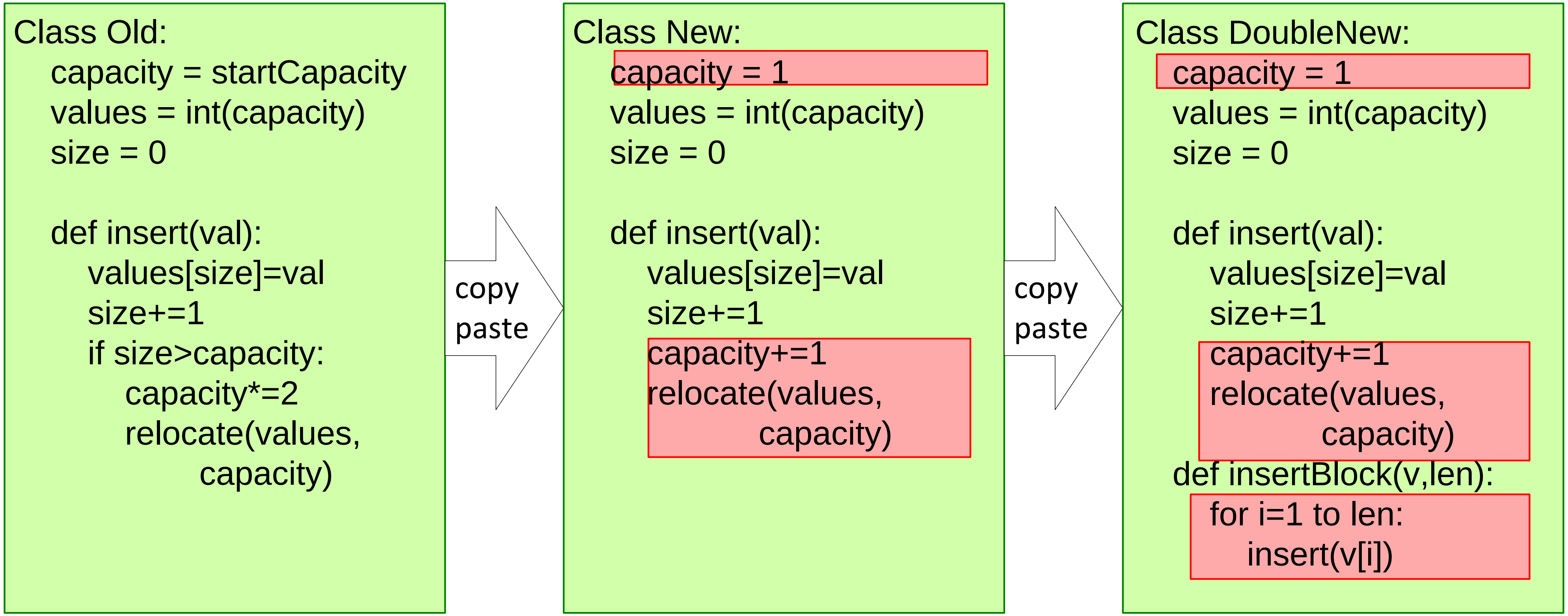


"Real-life" example: redesigned *Tool* super-class and descendants, ca 1994

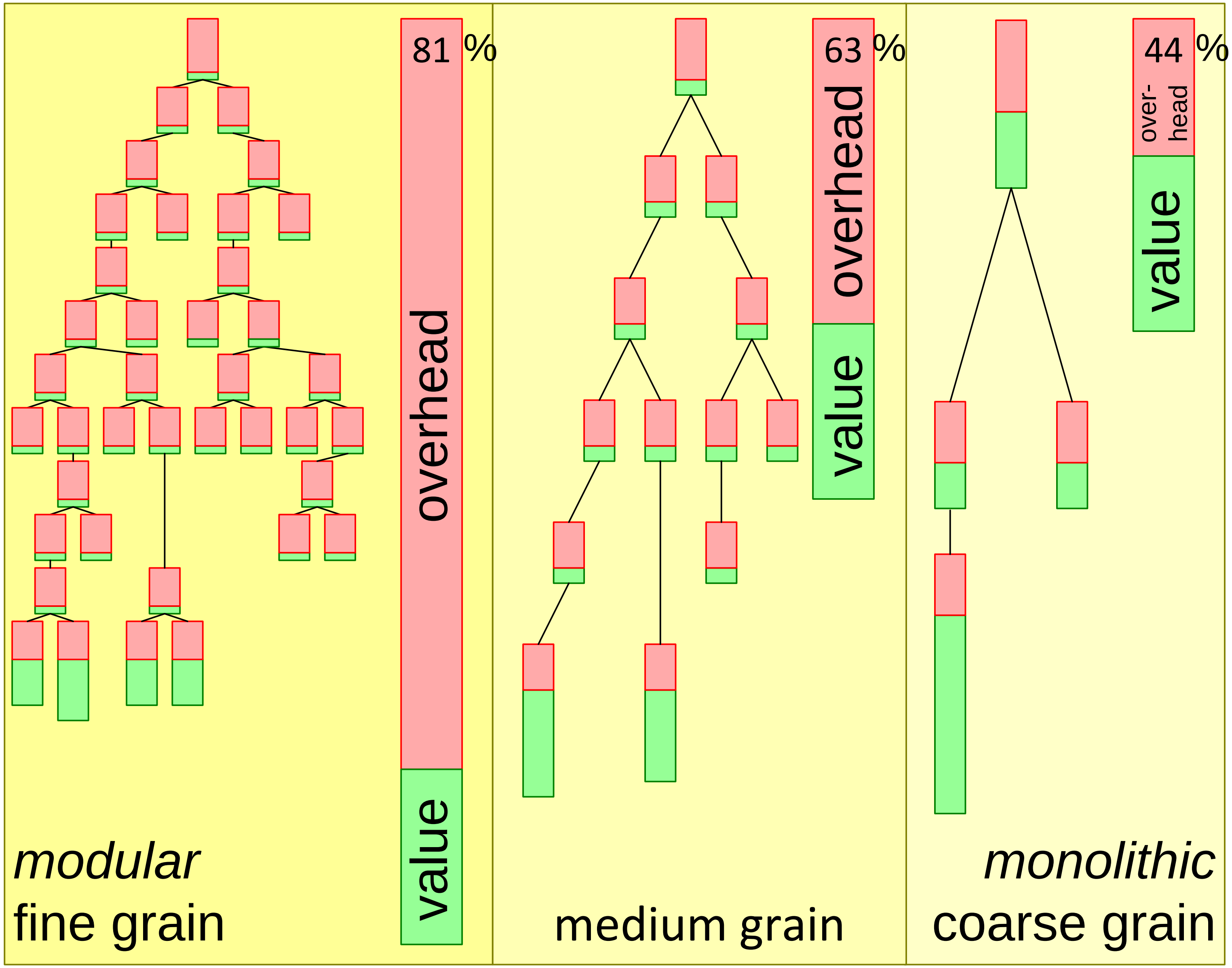
Problem Propagation via Copy & Paste



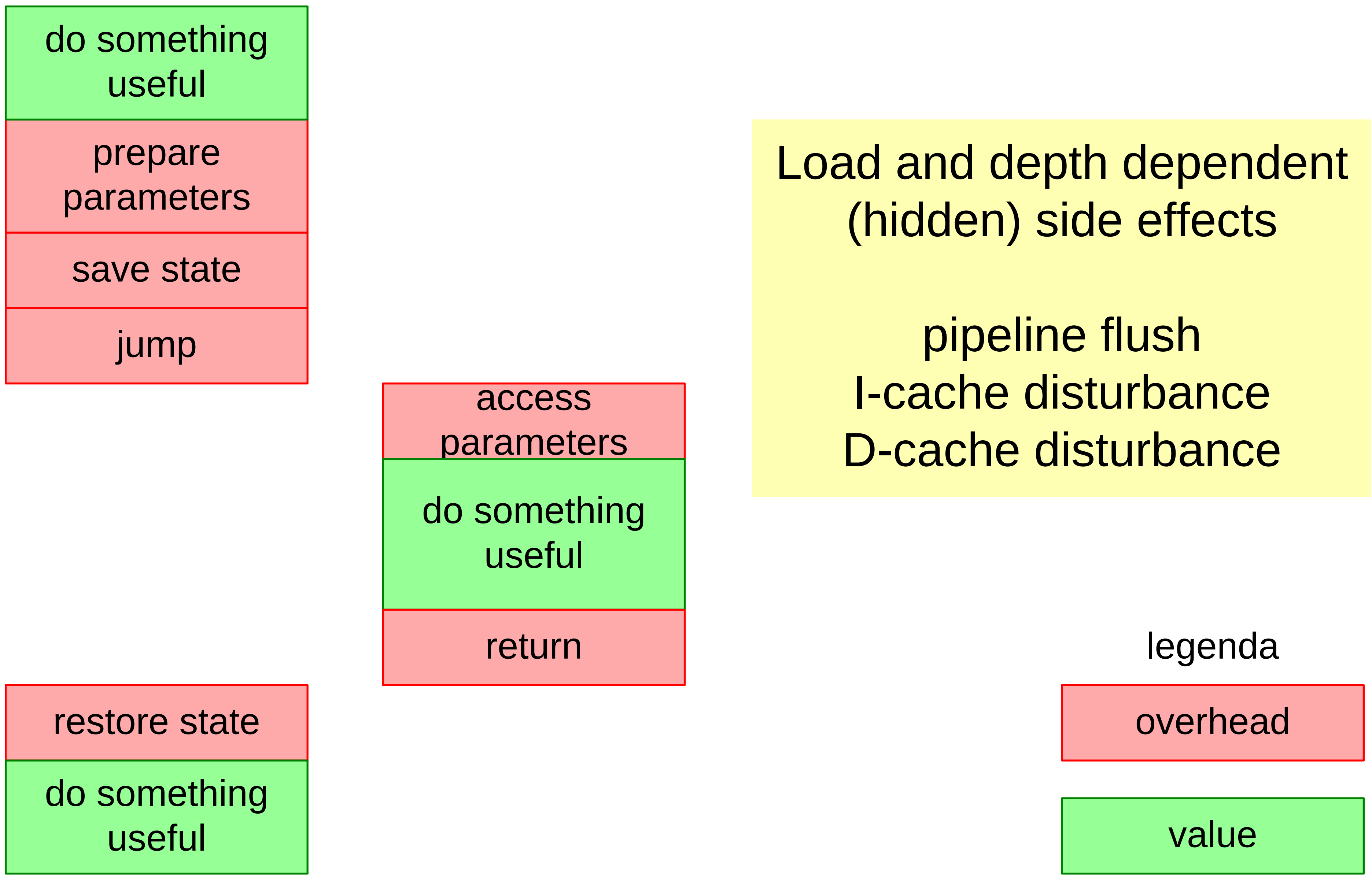
Example of Problem Propagation



Overhead Penalty of Modularity



Function Call Overhead



Exercise Call Tree Overhead

Suppose:

Call Overhead = 10µs
Call graph branching factor = 2
Depth = 12

What is the Call overhead
when all branches are followed?

Exercise Frame Rate for Layered SW

Suppose:

Function call = 10µs
Call layer depth = 20
1024 calls per image

What is the maximum frame rate possible assuming that the complete CPU time is available for function calls?

Case 6

Common Platforms and Bloating

Platforms are overprovisioned and very generic

Are benefits > disadvantages?

Performance loss is significant and can be measured and modelled

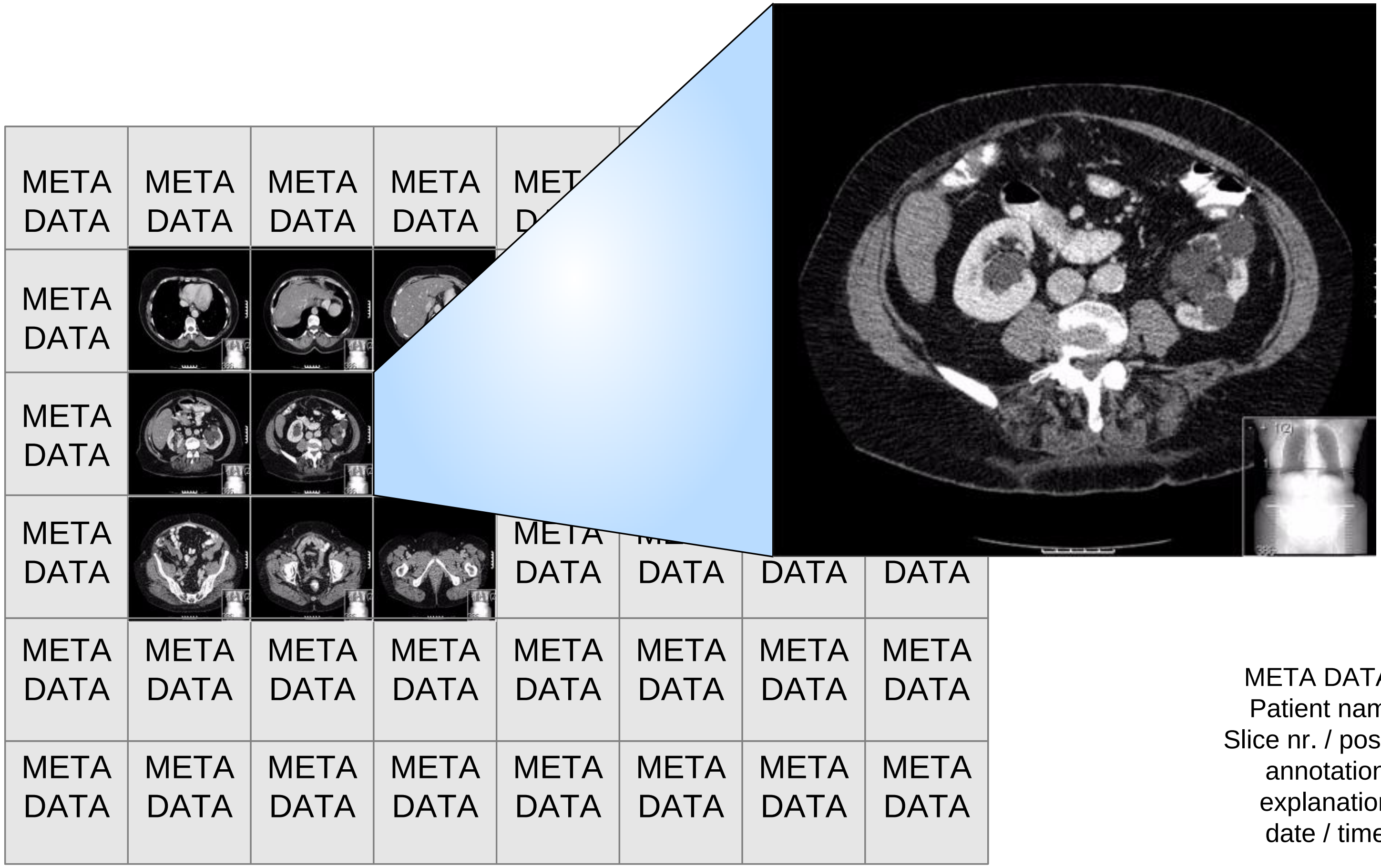
Multi-Dimensional Viewing of many Images: Greedy and Lazy Design Patterns

Greedy and Lazy systems

Greedy: pre-fetched lots of data:
System tries to have data available for the requesting system

Lazy: hardly of no pre-fetching of data:
System tries to set data available for the requesting system
only when asked for

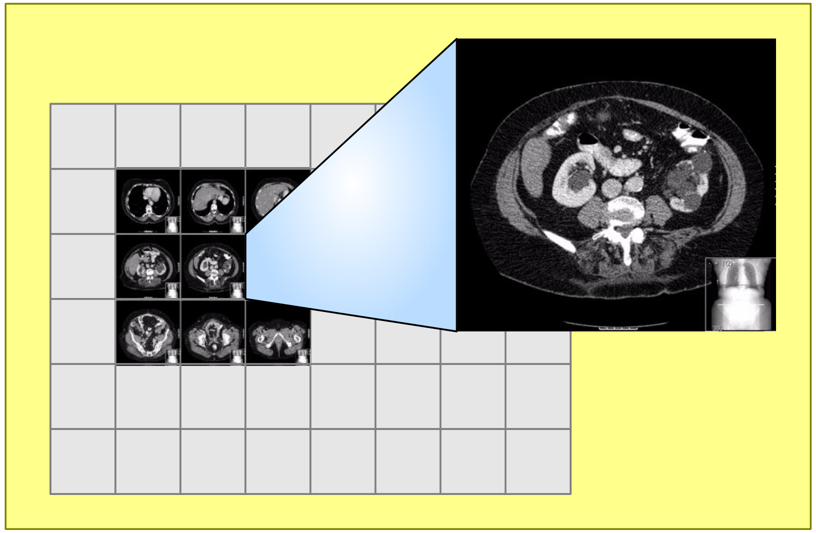
Example Greedy / Lazy (1)



META DATA=
Patient name
Slice nr. / position
annotation
explanation
date / time

Example Greedy / Lazy (2)

Lazy: Fetch only
the requested image



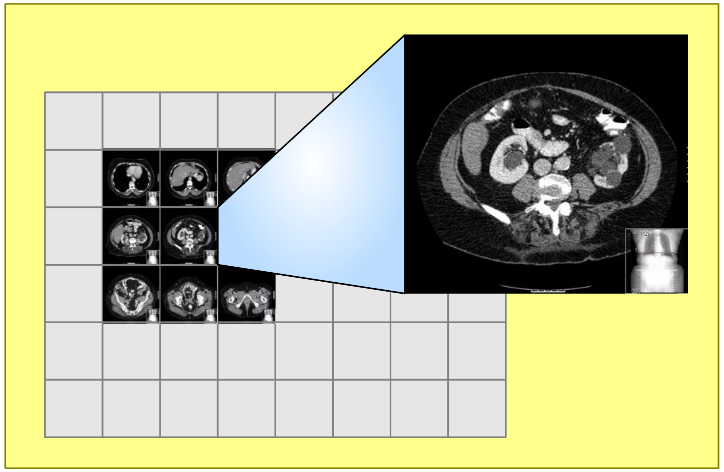
Greedy: Fetch all the images
in the set

In between options:

- Fetch requested image + surrounding images
- Fetch requested image + only meta information of images

Example Greedy / Lazy (3)

Lazy:
- low load on system
- long waiting time for next image



Greedy:
- high load on system
- possible long initial wait
- short response time insteady state

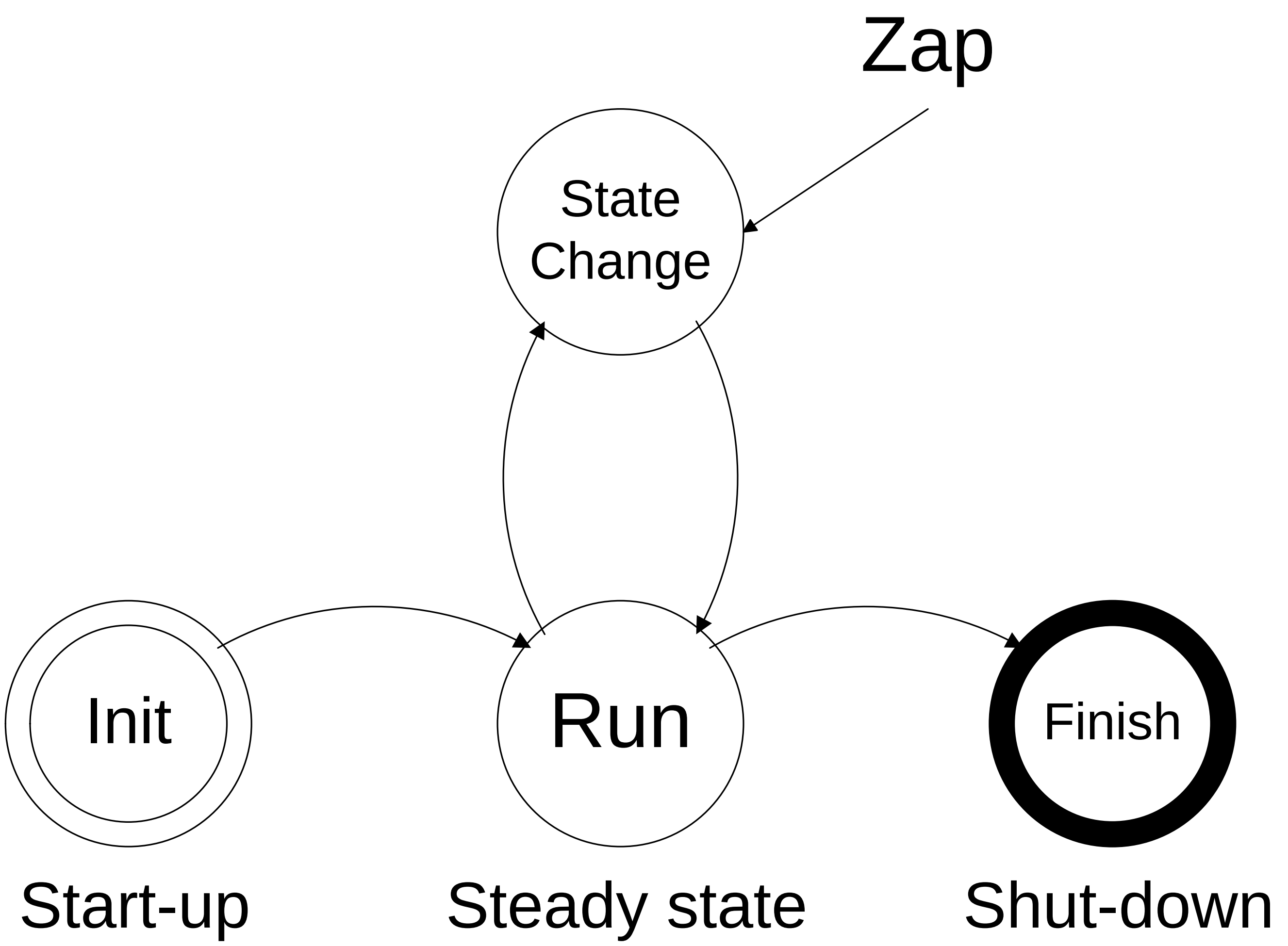
In between options:

- medium system load

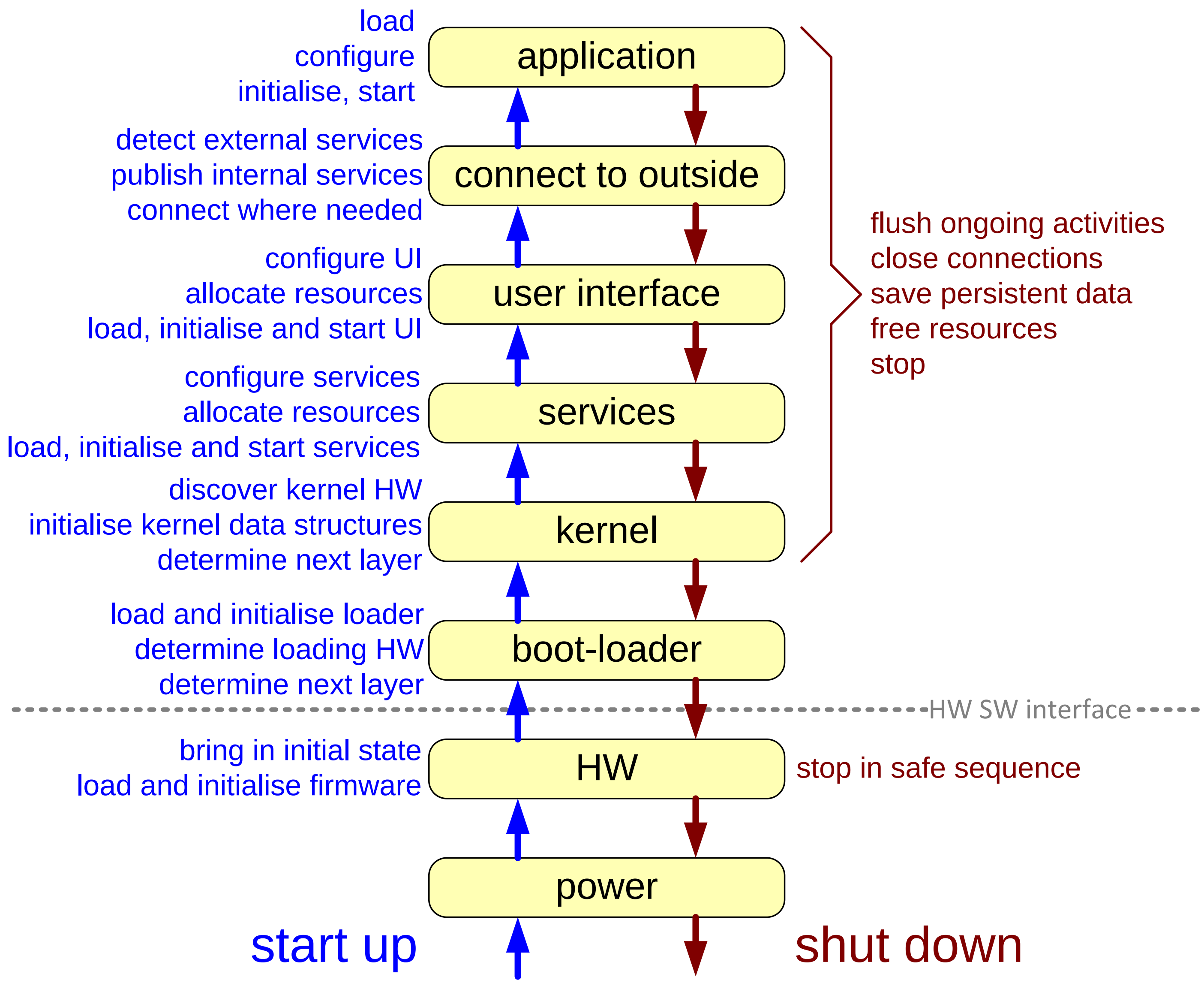
- fast response for initialization and common image fetches

Initialization, Steady State and Finalization

Start-up, Steady State, Shut Down



Start-up, Steady State, Shut Down Scheme

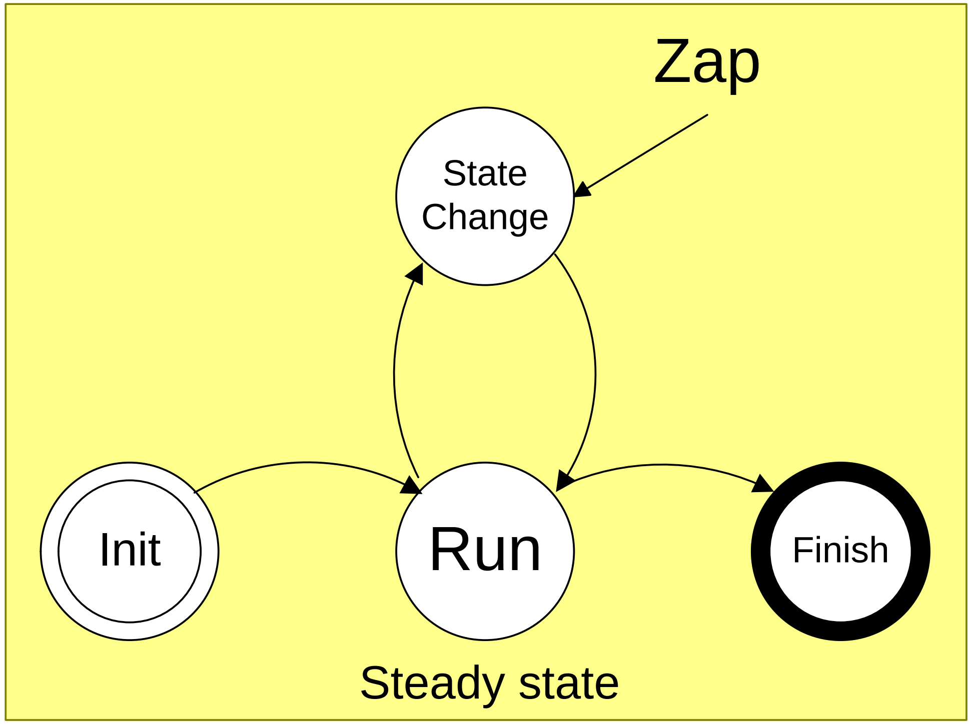


Start-up, Steady State, Shut Down Trade off

Trade-off:

Optimize on steady state
may result in
poor performance for initialization
and process finish

Optimize on Initialization and/or finish
may result in
poor steady state performance



Common Performance Pitfalls

- Overhead
- Data bloating
- Cache thrashing
- Layering
- Process communication
- Conversions
- Serialization
- Backfiring optimalisations
- Hidden loads (bus, DMA etc)
- Poor algorithms
- Wrong dimensioning

Performance Design of Streaming Systems

by *Gerrit Muller* HSN-NISE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

Abstract

Video and audio content is a continuous stream of data. Video and audio systems have to be designed in such a way that these streams are processed and delivered continuously. We discuss the pipelining of multiple functions and the impact on bus bandwidth, memory use and CPU overhead.

March 6, 2021
status: preliminary draft
version: 0

Case Video Streaming

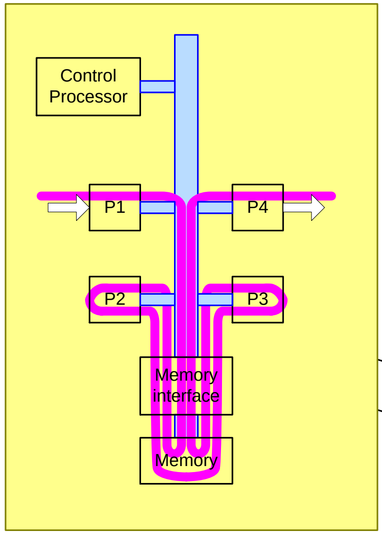
Video Streaming

Hard real-time performance for distributed system with memory-bus

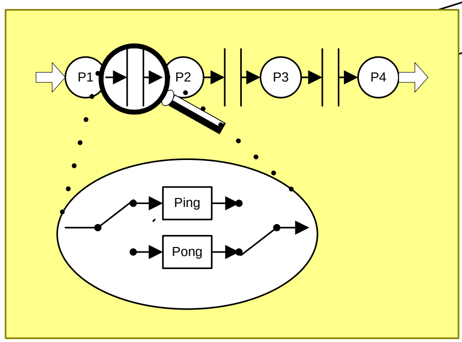
Trade-off of between latency, memory and overhead

Performance consideration in increasing detail

Case Video Streaming: Performance Design



Fixed HW diagram



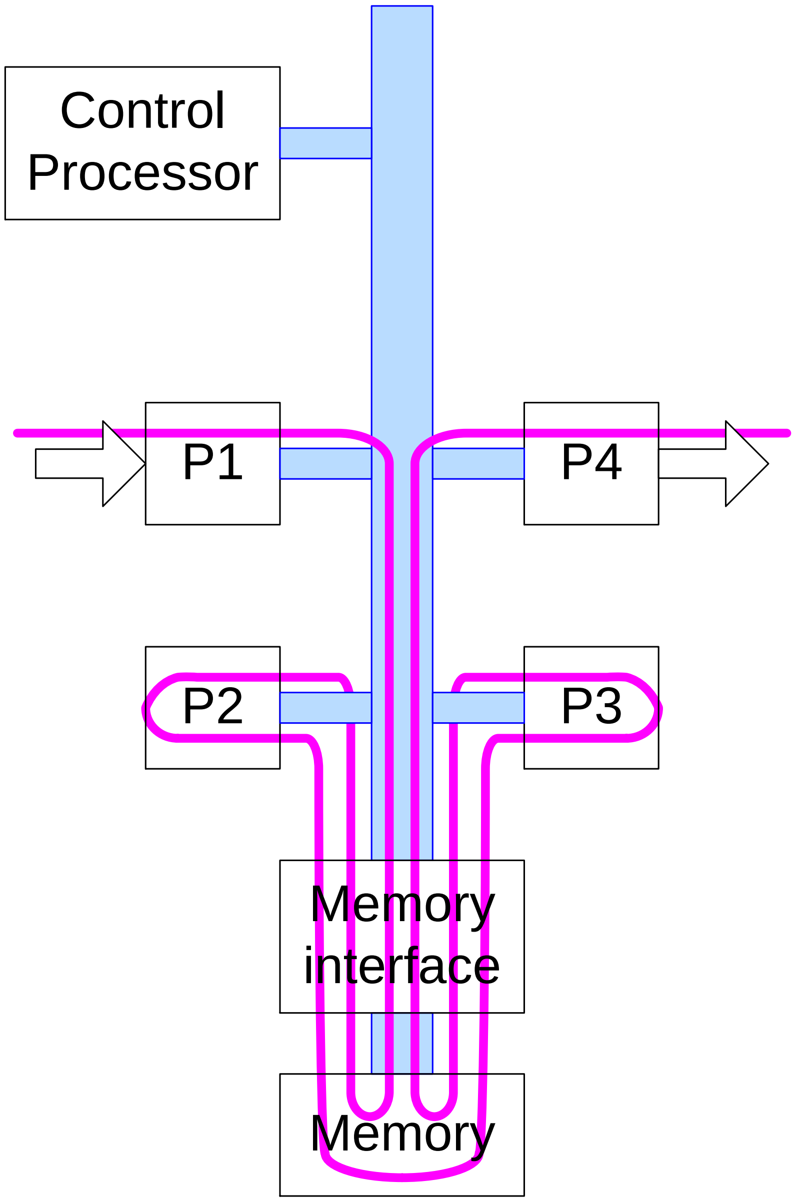
Fixed algorithmic flow

Design process

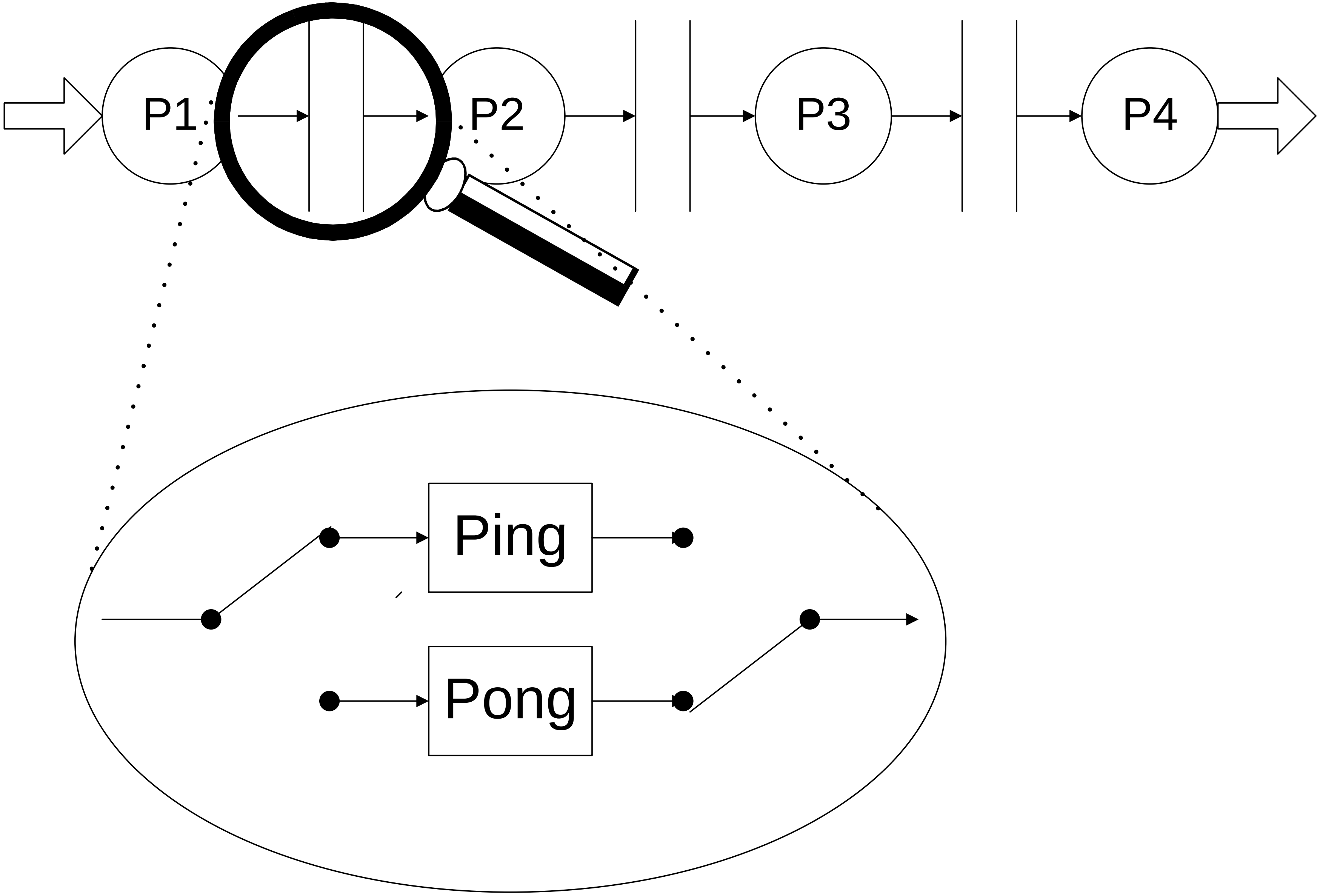
- Latency
- CPU Overhead
- Memory use
- Bus load

- Granularity
- Synchronisation strategy

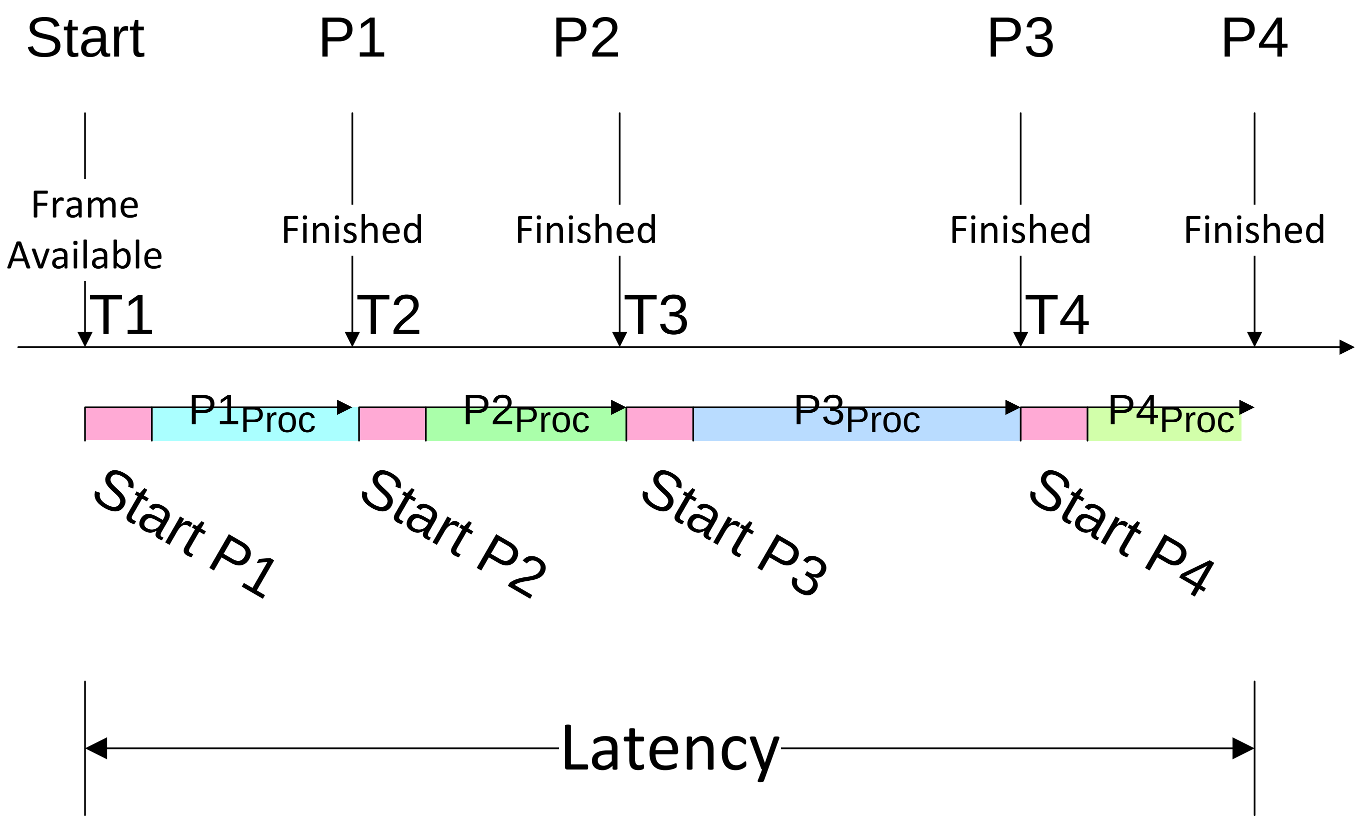
Video Streaming: HW Diagram



Video Streaming Pipeline



Video Streaming: Latency



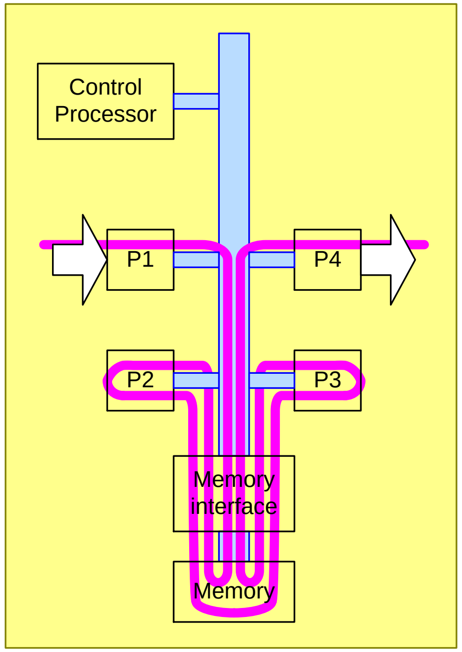
$P3_{Proc} \lesssim ½\ T_{frame}$

$Latency \approx 2\ T_{frame}$

Legenda

- Task switch
- Process 1
- Process 2
- Process 3
- Process 4

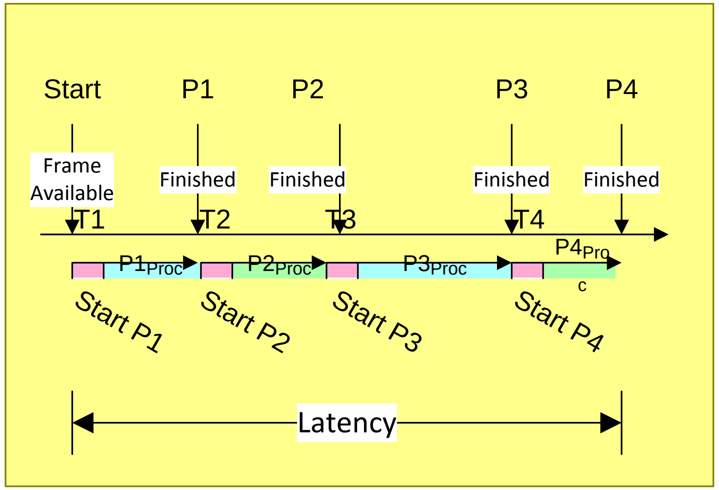
Video Streaming: Resources



Overhead = (T1 + T2 + T3 + T4) * Frame rate

Memory usage = 3 * 2 * Frame size

$$\text{Bus load} = \frac{3 * 2 * \text{Frame size} * \text{Frame rate}}{\text{Bus capacity}} \quad \%$$



T1 .. T4 = Overhead
to start P1 .. P4

Latency Calculation

Latency = Nr. of Proc. blocks (4) * processing time per block (0.01s) * frame fragment (1)

Memory = (Nr. of Proc. blocks (4) - 1) * 2 * pixels per frame (414720) * frame fragment (1)

Overhead = Nr. of Proc. blocks (4) * task switch time (10 µs)

Overhead (%) = Overhead (40 µs) / Latency (40 ms)

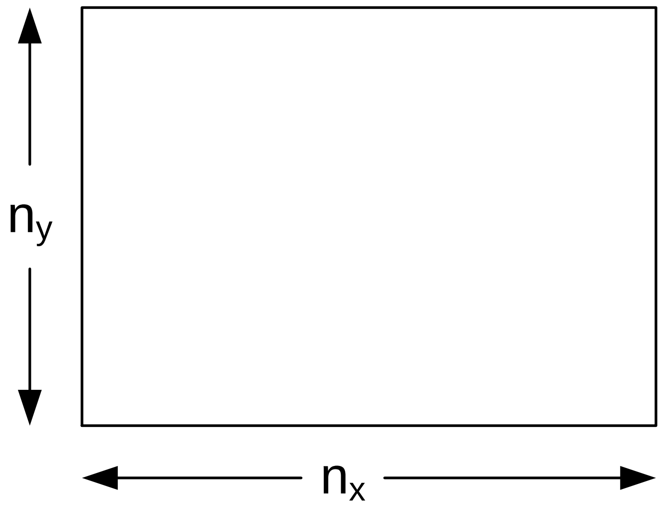
Busload = Memory usage (2430 kB) * frame fragment (1) * (frames/s) (25) / BusCapacity (500MB/s)

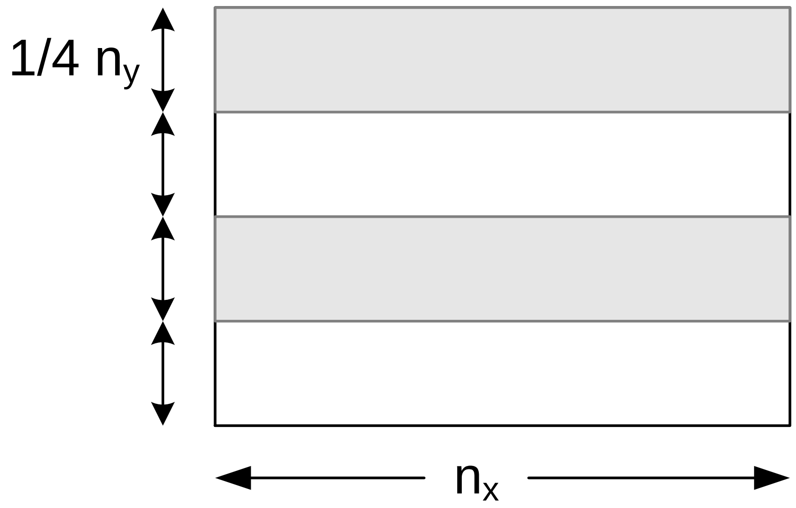
(mind the units, ms vs. µs and kB vs MB!)

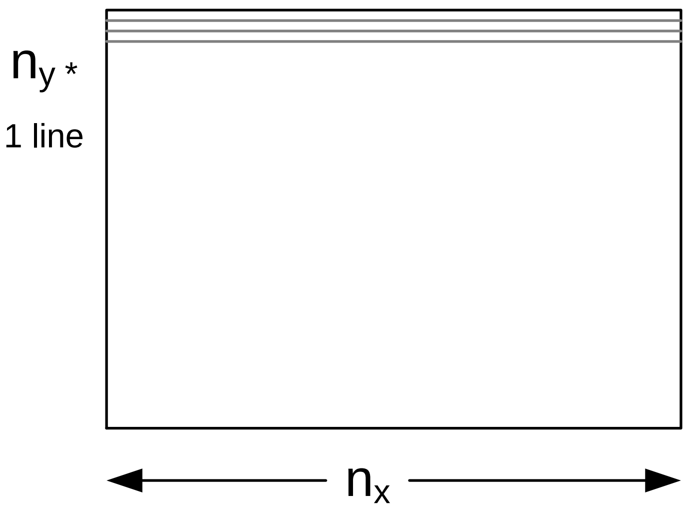
lines	576		pixels per frame	414720
pixels per line	720		Memory in kB	405
			Memory in MB	0.40
frame time	0.04		frame time in µs	40000
task switch time (µs)	10			
Processing per block	0.01		Processing in µs	10000
Bus capacity (MB/s)	500			
Line time (µs)	69			
Frame fragment	Full frame : 1			

Nr of Processing Blocks	4
Latency (ms)	40
Memory (kB)	2430
Overhead (µs)	40
Overhead (%)	0
Busload (%)	12.15

Exercise







Calculate:
Processing time
Overhead
Memory Use
Latency
for buffer size = 1/4 frame size
and for
buffer size = 1 video line

Exercise Worksheet

Nr of Processing Blocks		4	20
Block size			
	Latency (ms)	40	200
Frame	Memory (kB)	2430	15390
1	Overhead (µs)	40	200
	Overhead (%)	0	0
	Busload (%)	12.15	76.95
	Latency (ms)		
½ Frame	Memory (kB)		
2	Overhead (µs)		
	Overhead (%)		
	Busload (%)		
	Latency (µs)		
Line	Memory (kB)		
576	Overhead (µs)		
	Overhead (%)		
	Busload (%)		

lines	576
pixels per line	720
pixels per frame	414720
Memory in kB	405
Memory in MB	0.395508
frame time	0.04
frame time in µs	40000
task switch time (µs)	10
Processing per block	0.01
Processing in µs	10000
Bus capacity (MB/s)	500
Line time (µs)	69

Changing the Buffer Size

buffersize = ¼ frame

Processing time = ¼ * original (per fragment)
Latency ~ ¼ * original
Overhead = 4 * original
Memory use = ¼ original

buffersize = 1 line

Processing time = $^{1}/_{576}$ * original (per fragment)
Latency ~ $^{1}/_{576}$* original **+ overhead**
Overhead = 576 * original
Memory use = $^{1}/_{576}$ original

Summary Case Video Streaming

Video Streaming

Properly designing distributed HRT systems requires trade-off between latency, overhead, and memory needs

Performance model detailing dependent on significance of impact factors

Home work reporting

Exercise

Measure functions or platform characteristics needed for “Fast Browser”. Select most critical characteristics

Home work reporting

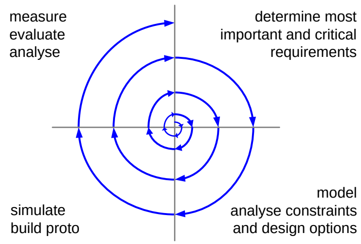
Performance Method Fundamentals

by *Gerrit Muller*   HSN-NISE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

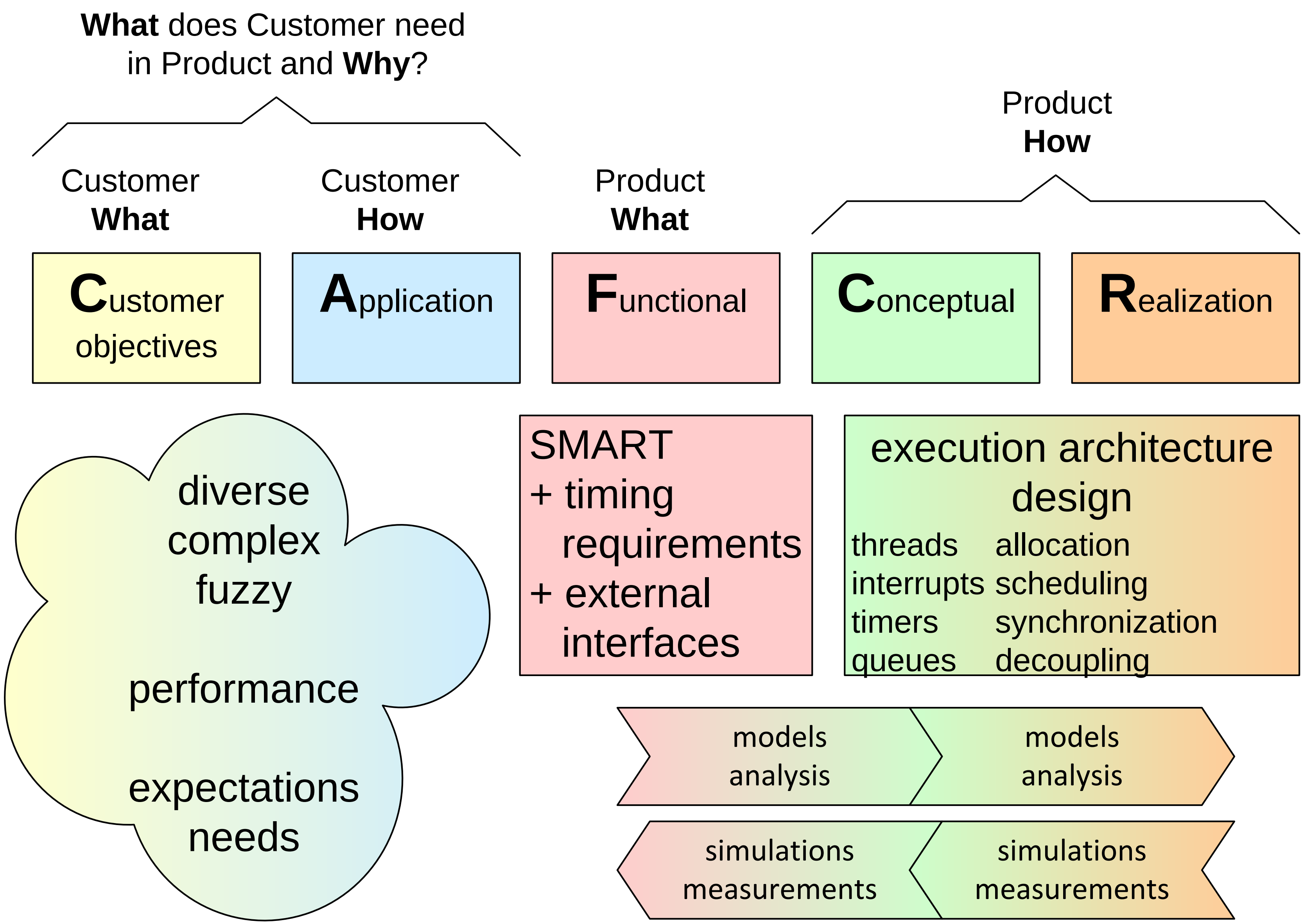
Abstract

The Performance Design Methods described in this article are based on a multi-view approach. The needs are covered by a requirements view. The system design consists of a HW block diagram, a SW decomposition, a functional design and other models dependent on the type of system. The system design is used to create a performance model. Measurements provide a way to get a quantified characterization of the system. Different measurement methods and levels are required to obtain a usable characterized system. The performance model and the characterizations are used for the performance design. The system design decisions with great performance impact are: granularity, synchronization, priorization, allocation and resource management. Performance and resource budgets are used as tool.

March 6, 2021
status: draft
version: 0.2



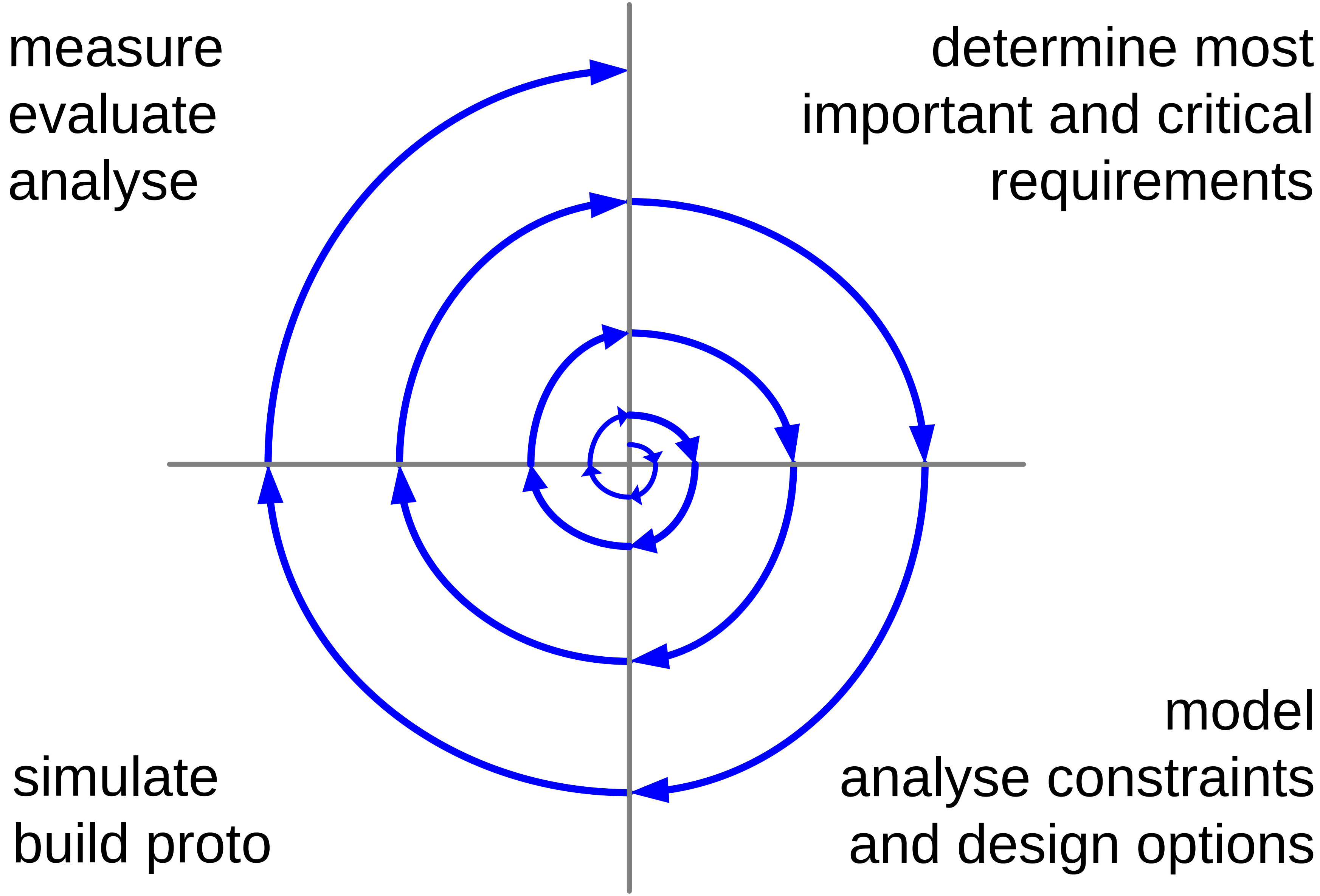
Positioning in CAFCR



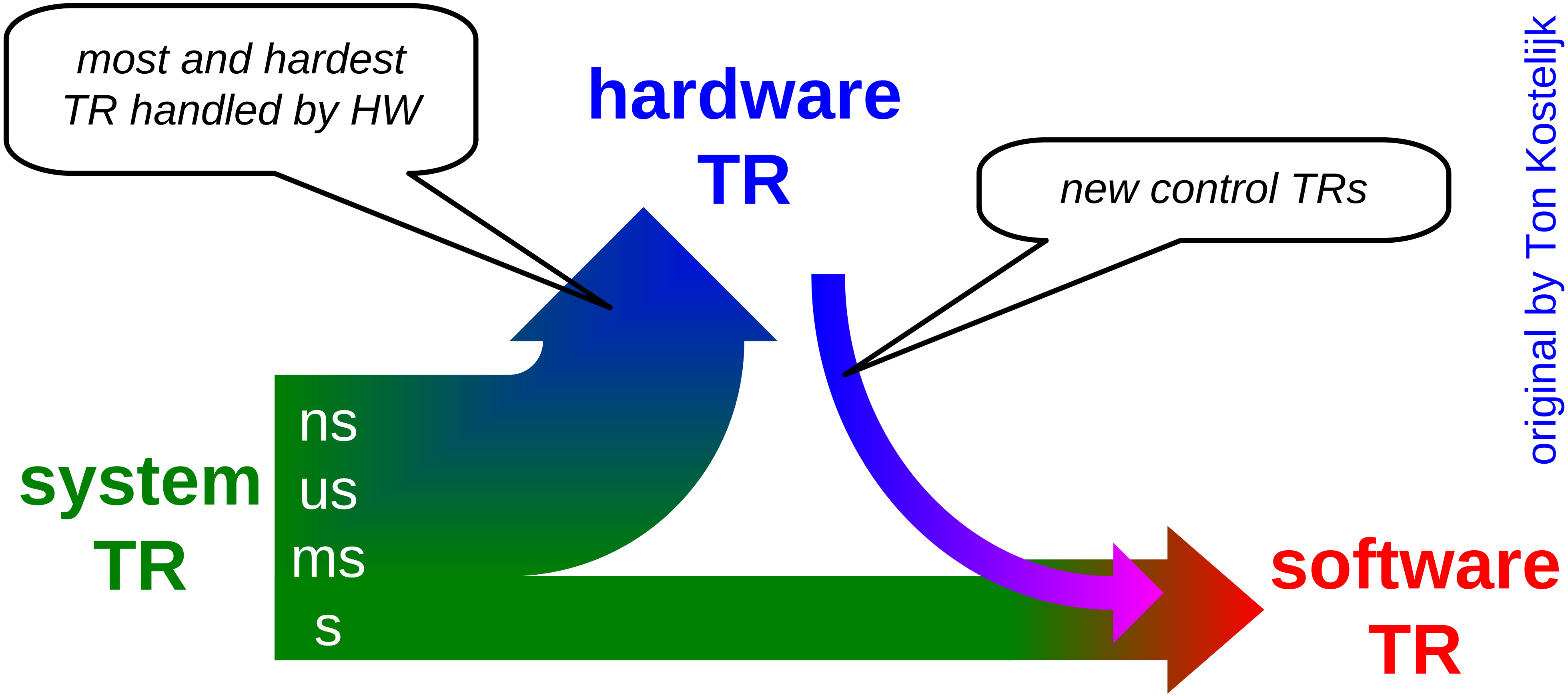
Toplevel Performance Design Method

1A Collect most critical performance and timing requirements	
1B Find system level diagrams	HW block diagram, SW diagram, functional model(s) concurrency model, resource model, time-line
2A Measure performance at 3 levels	application, functions and micro benchmarks
2B Create Performance Model	
3 Evaluate performance, identify potential problems	
4 Performance analysis and design	granularity, synchronization, priorization, allocation, resource management
Re-iterate all steps	are the right requirements addressed, refine diagrams, measurements, models, and improve design

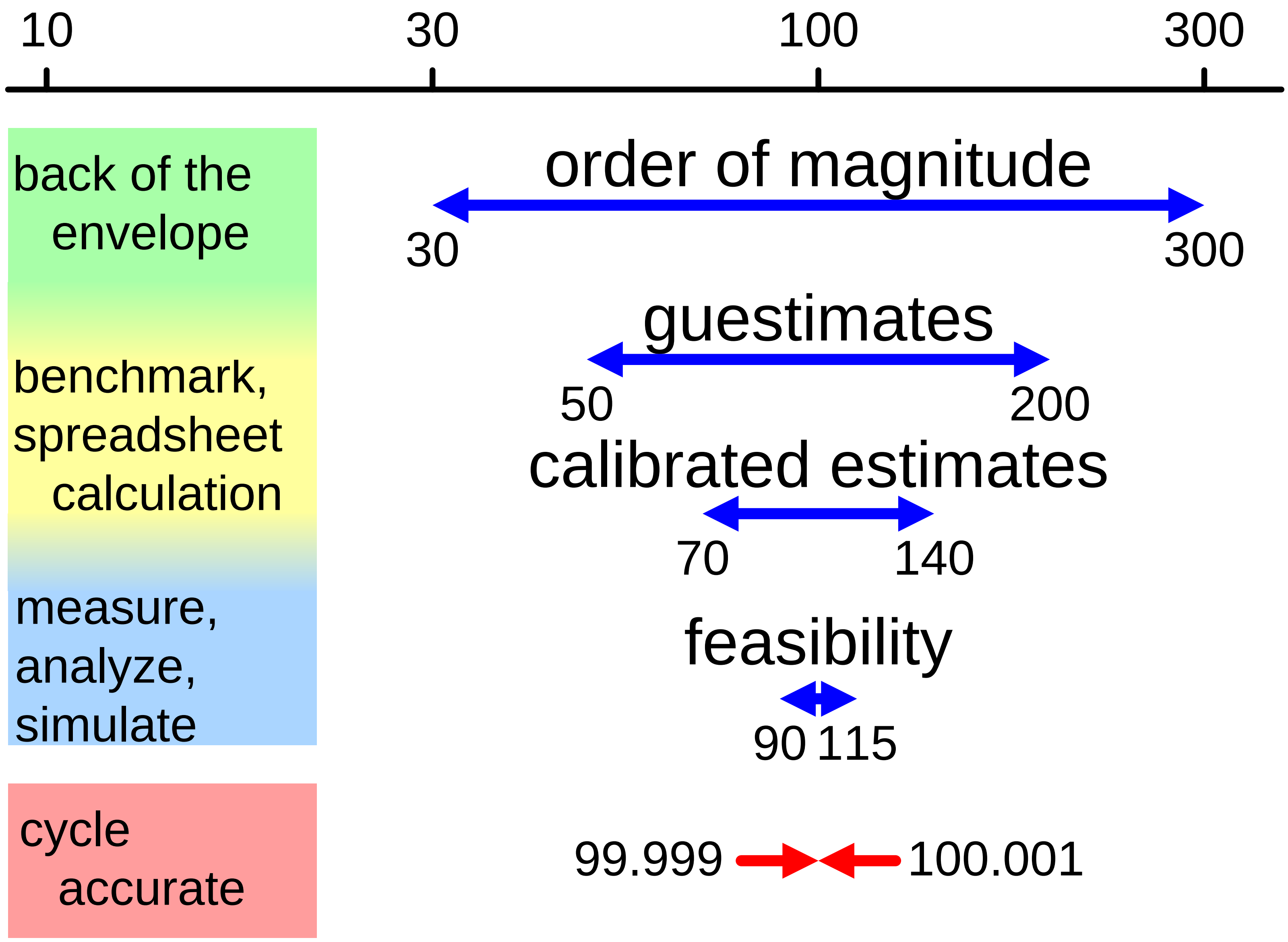
Incremental Approach



Decomposition of System TR in HW and SW



Quantification Steps



Iteration

zoom in on detail

aggregate to end-to-end performance

from coarse guestimate to reliable prediction

from typical case to boundaries of requirement space

from static understanding to dynamic understanding

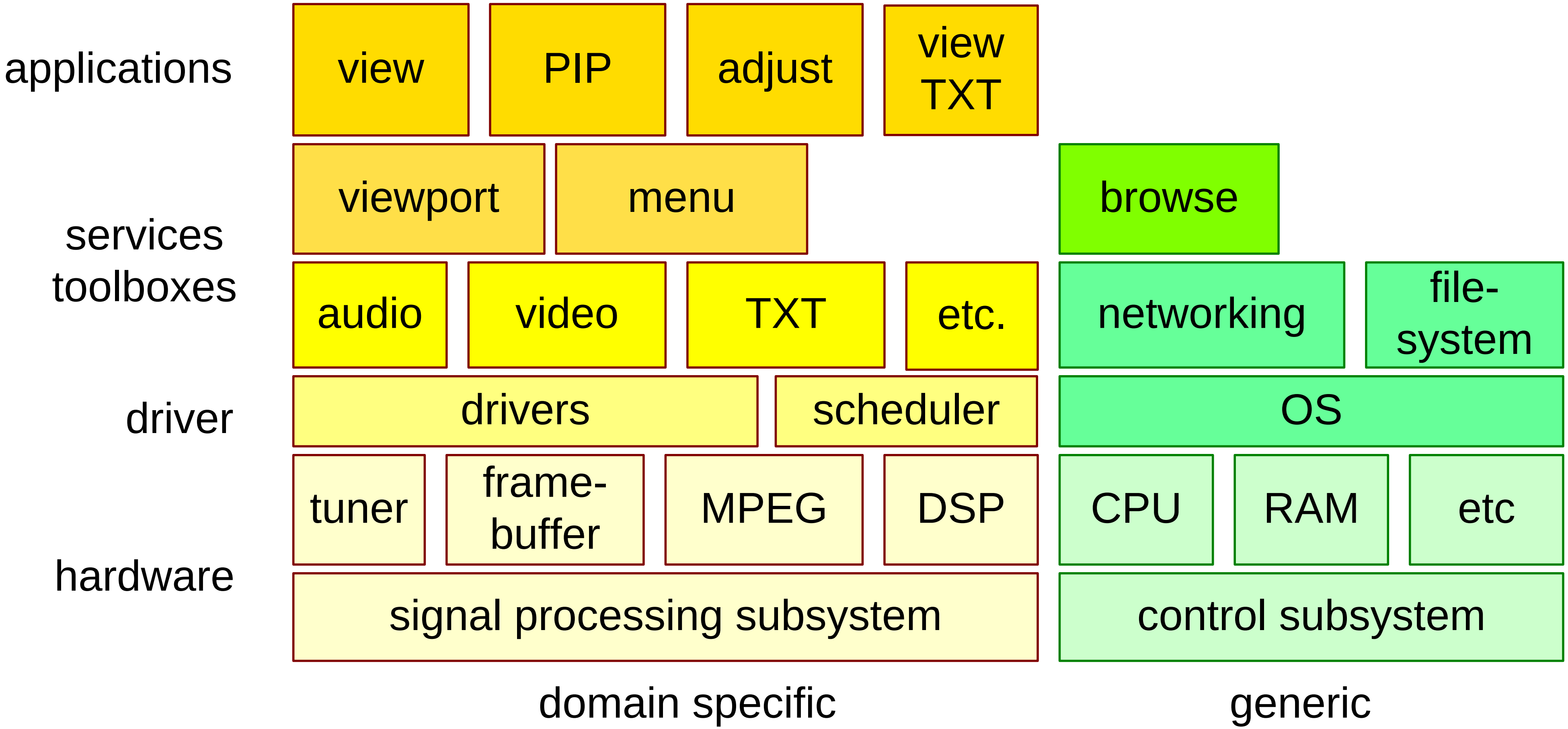
from steady state to initialization, state change and shut down

discover unforeseen critical requirements

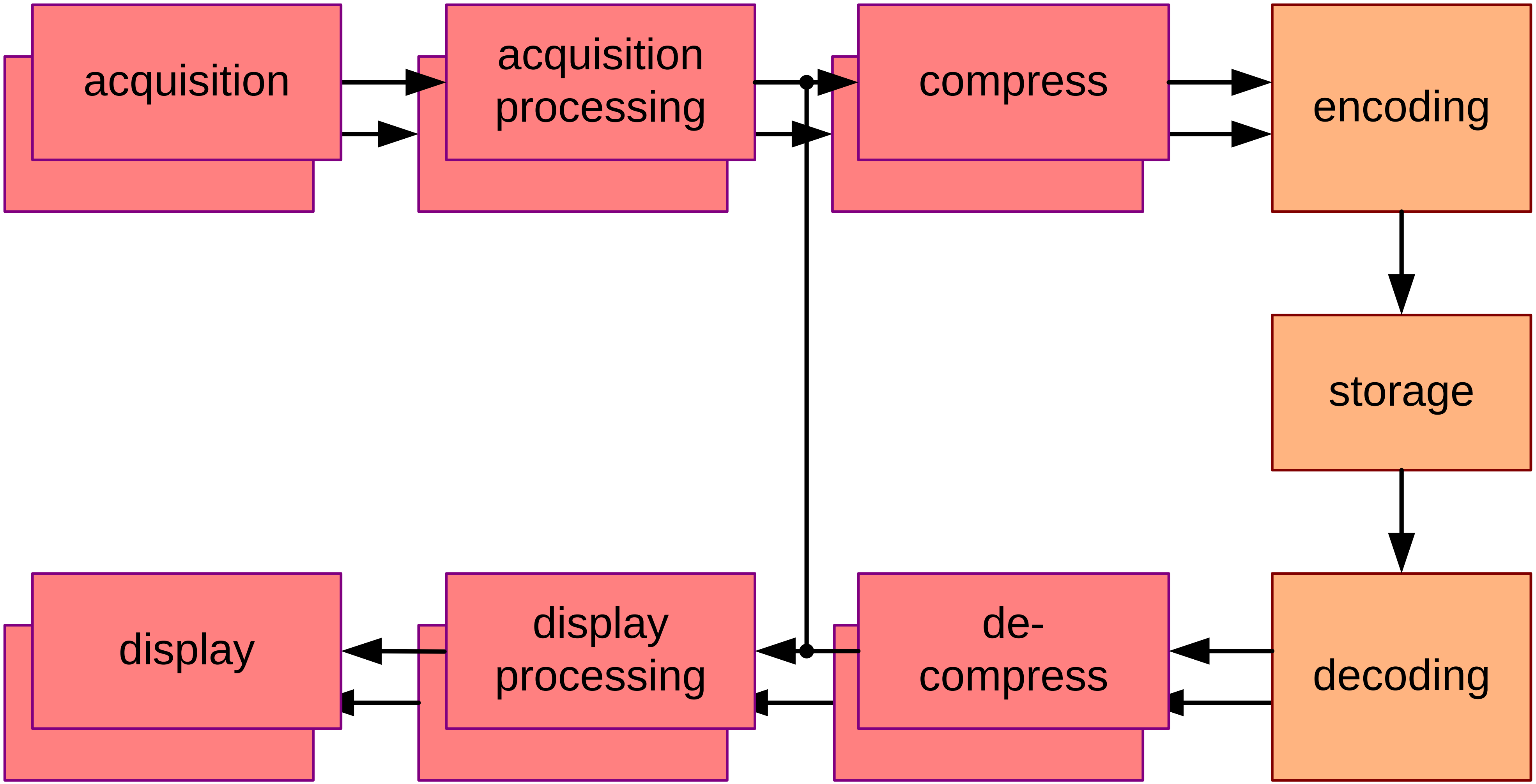
improve diagrams and designs

from old system to prototype to actual implementation

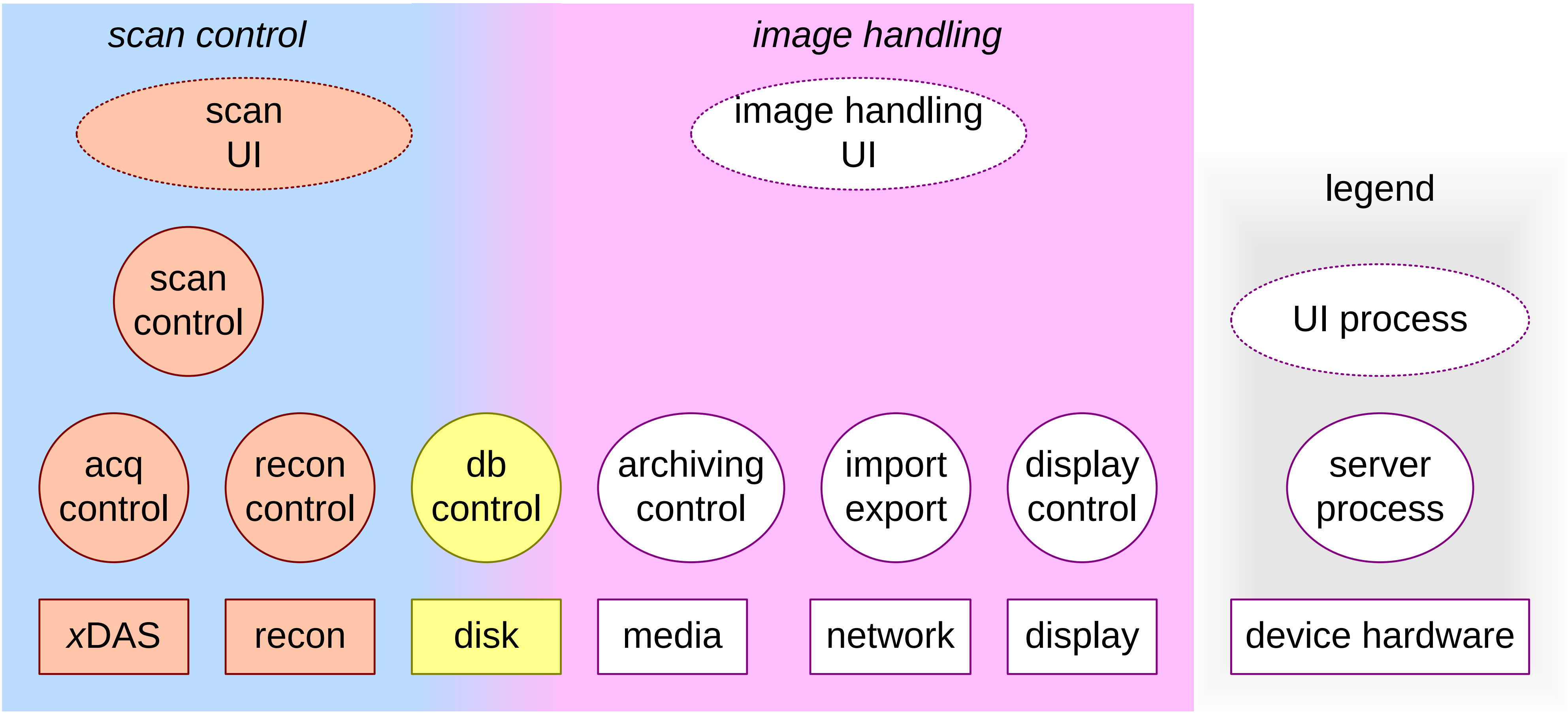
Construction Decomposition



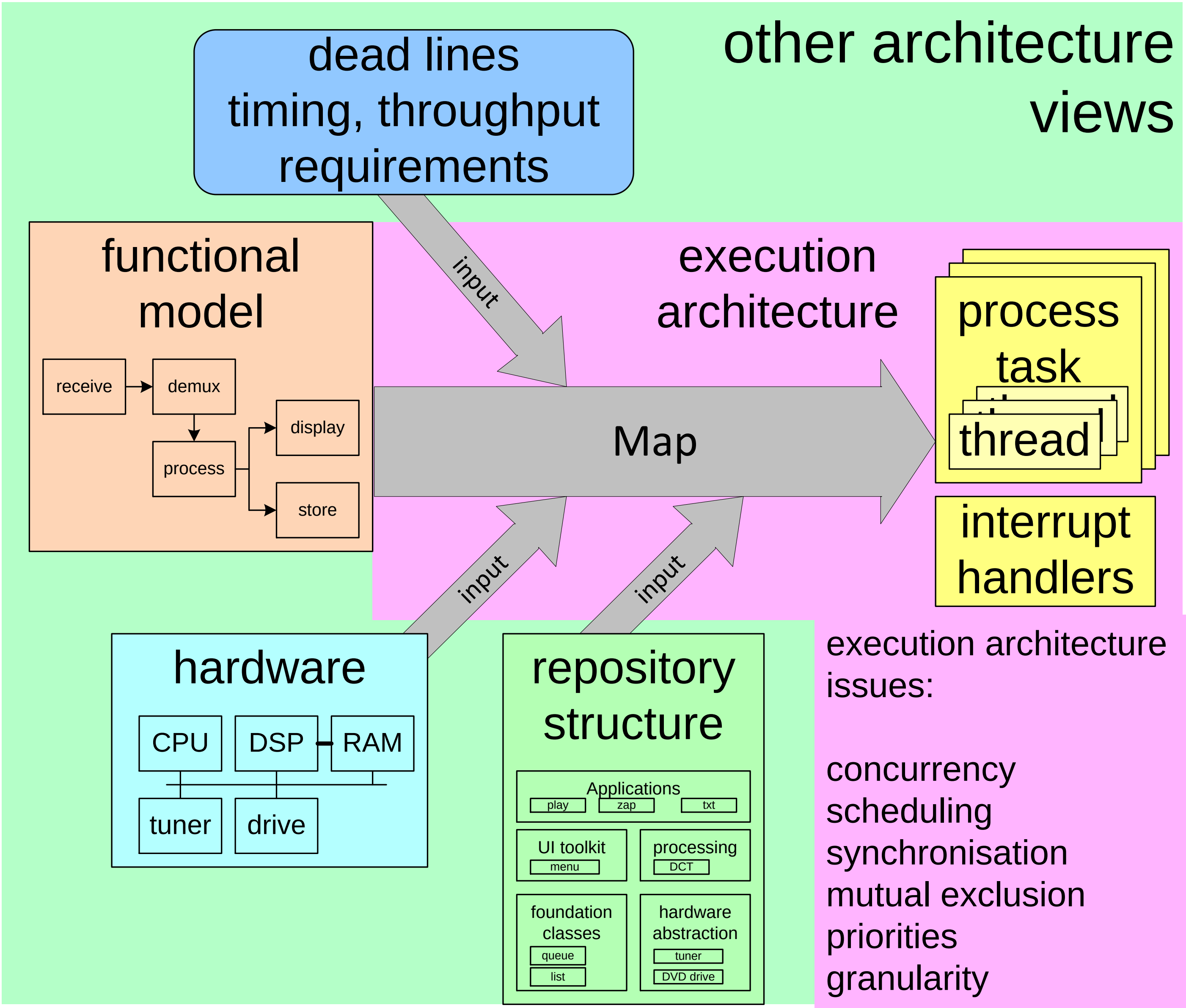
Functional Decomposition



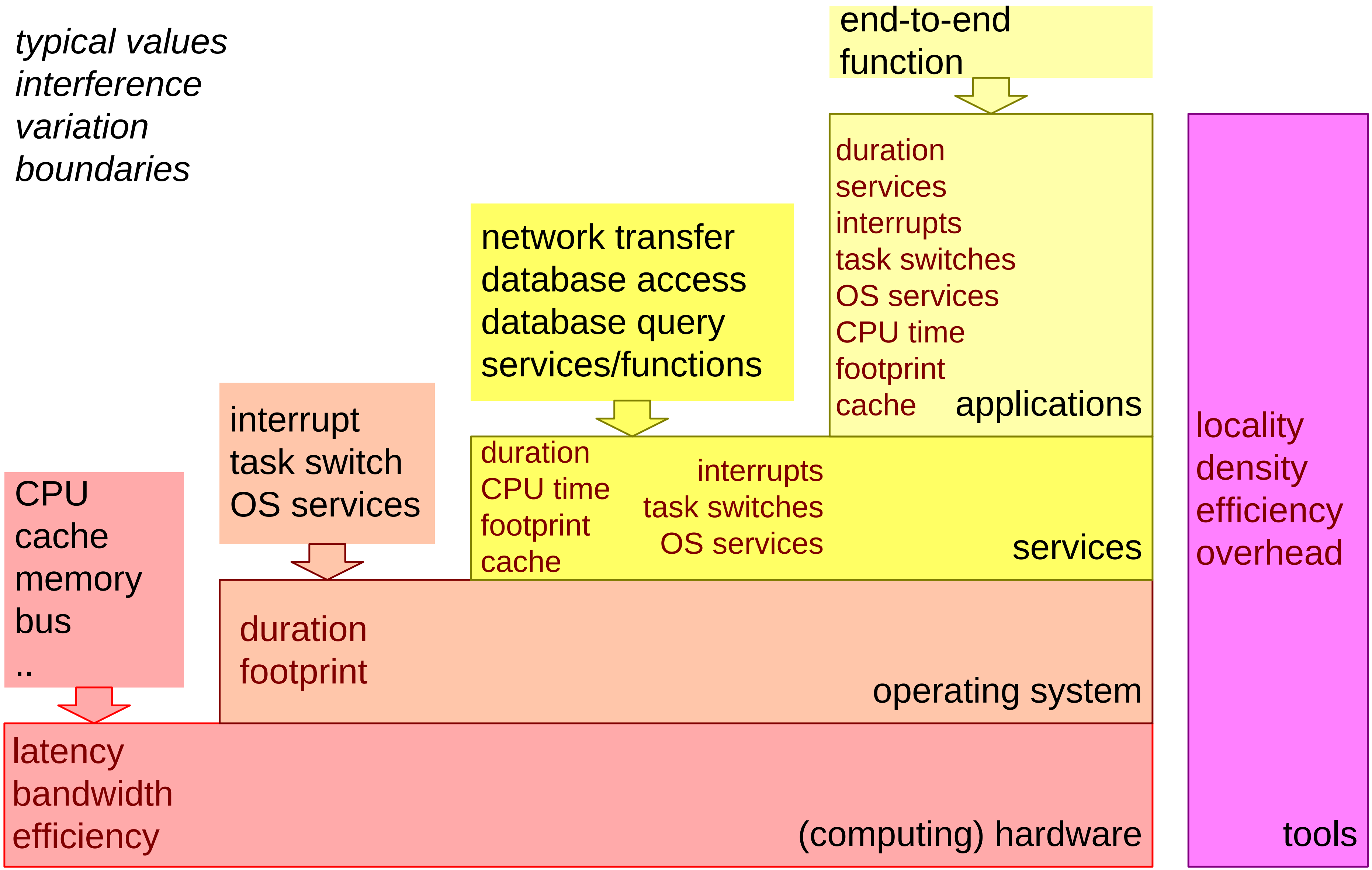
An example of a process decomposition of a MRI scanner.



Combine views in Execution Architecture



Layered Benchmarking Approach



Micro Benchmarks

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

Home work reporting

Performance and Reliability

To be inderted here

Exercise

Create “fast Browser” performance model. Finish measurements where needed