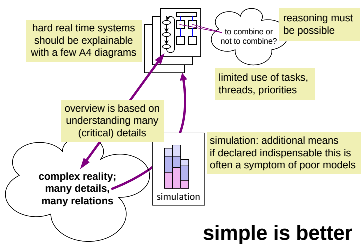
Execution architecture concepts

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

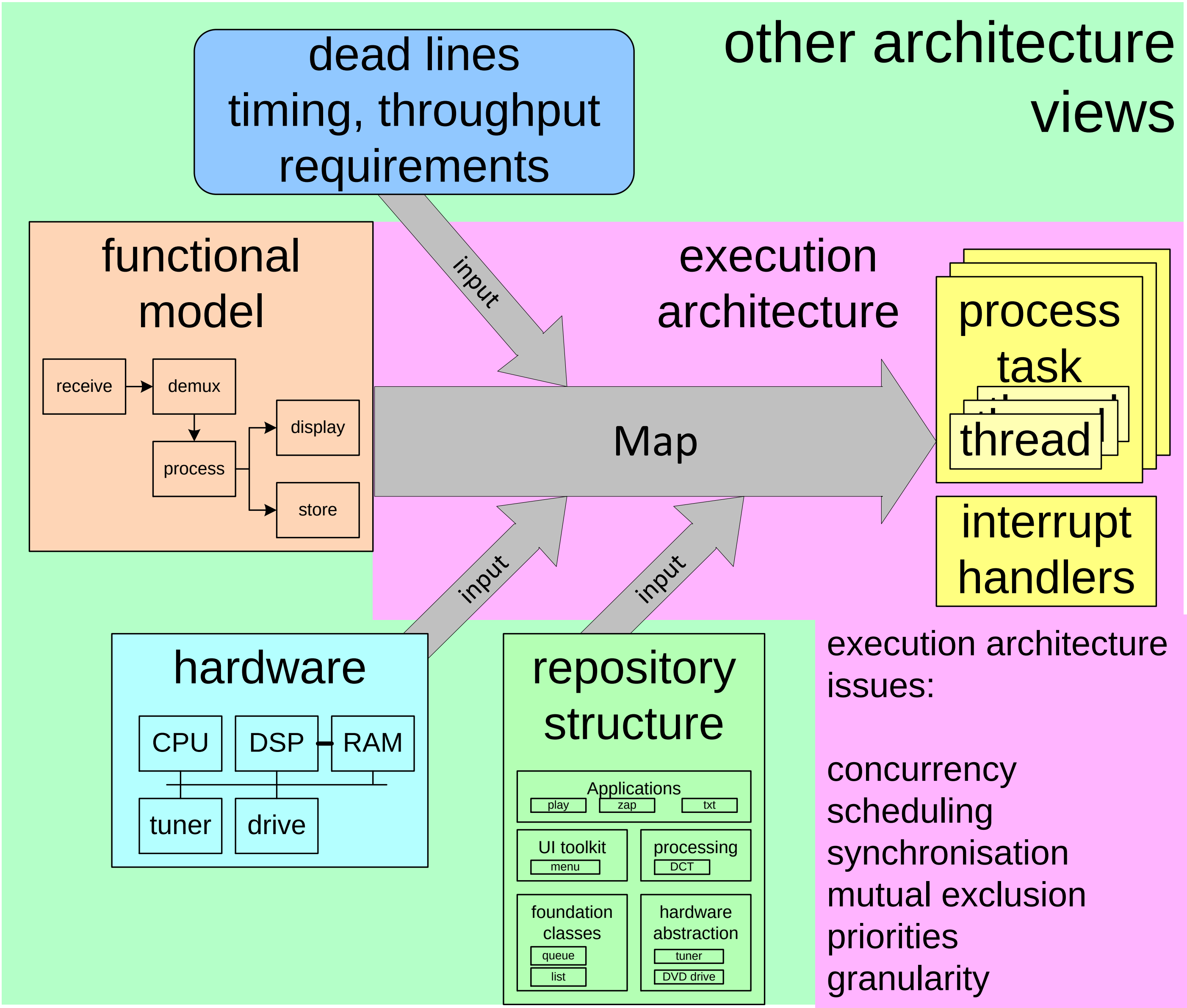
Abstract

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.

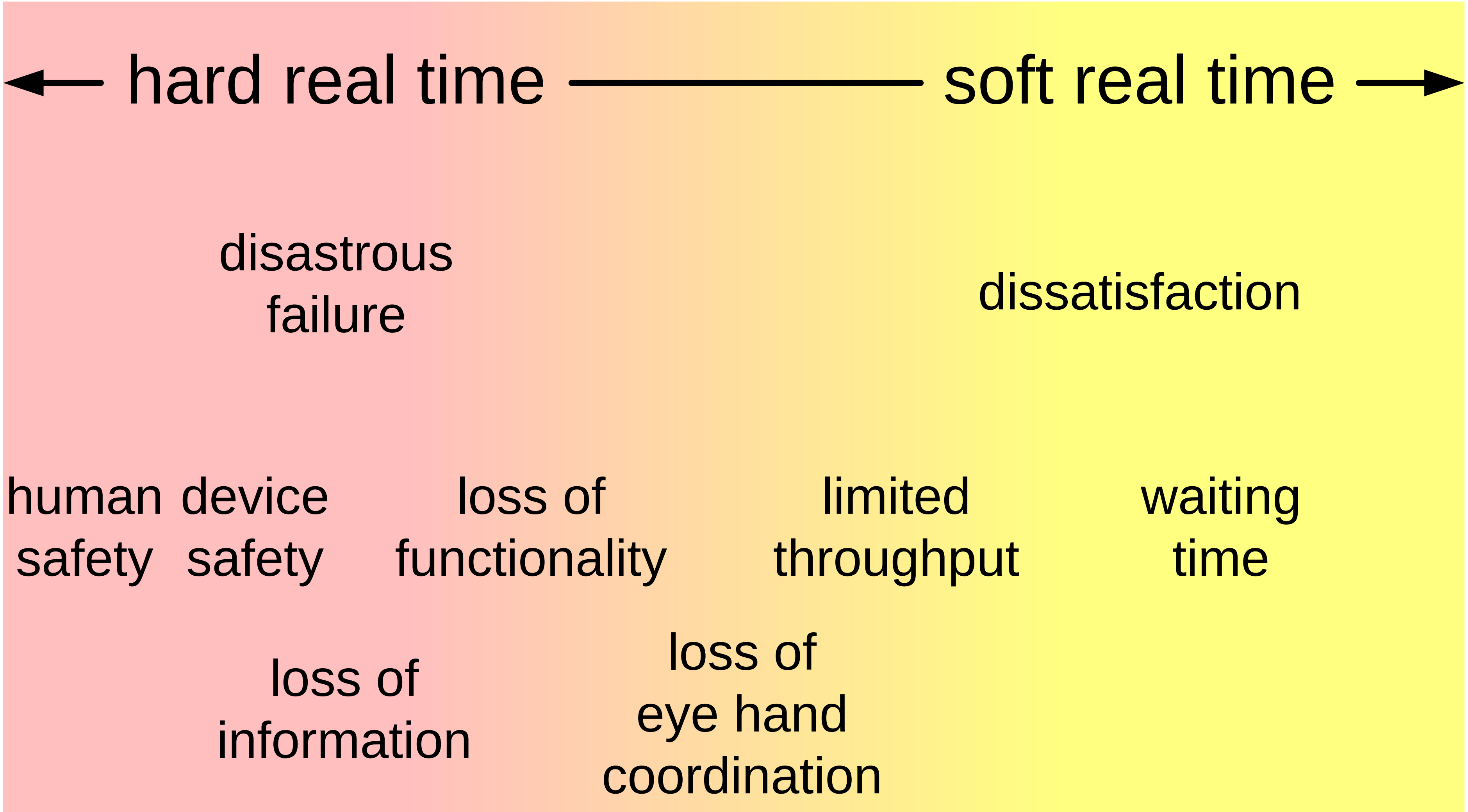


August 21, 2020
status: preliminary draft
version: 1.1

Execution Architecture

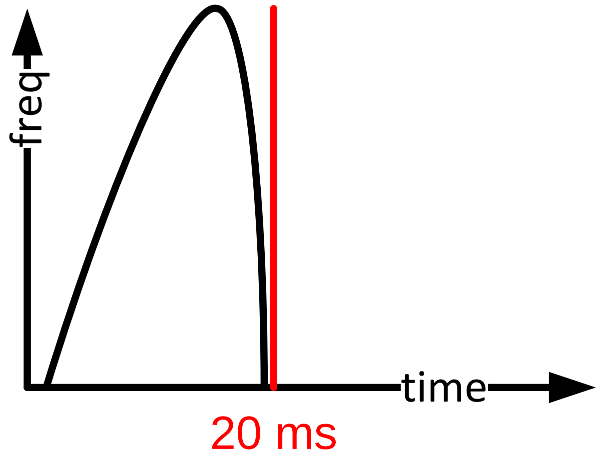


Fuzzy customer view on real time



Smartening requirements

Limited set of hard real time cases

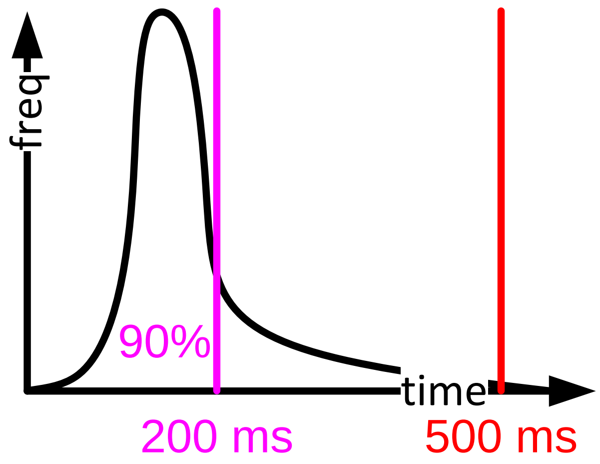


Precise form of the distribution is not important.

Be aware of systematic effects

No exception allowed
Worst case must fit

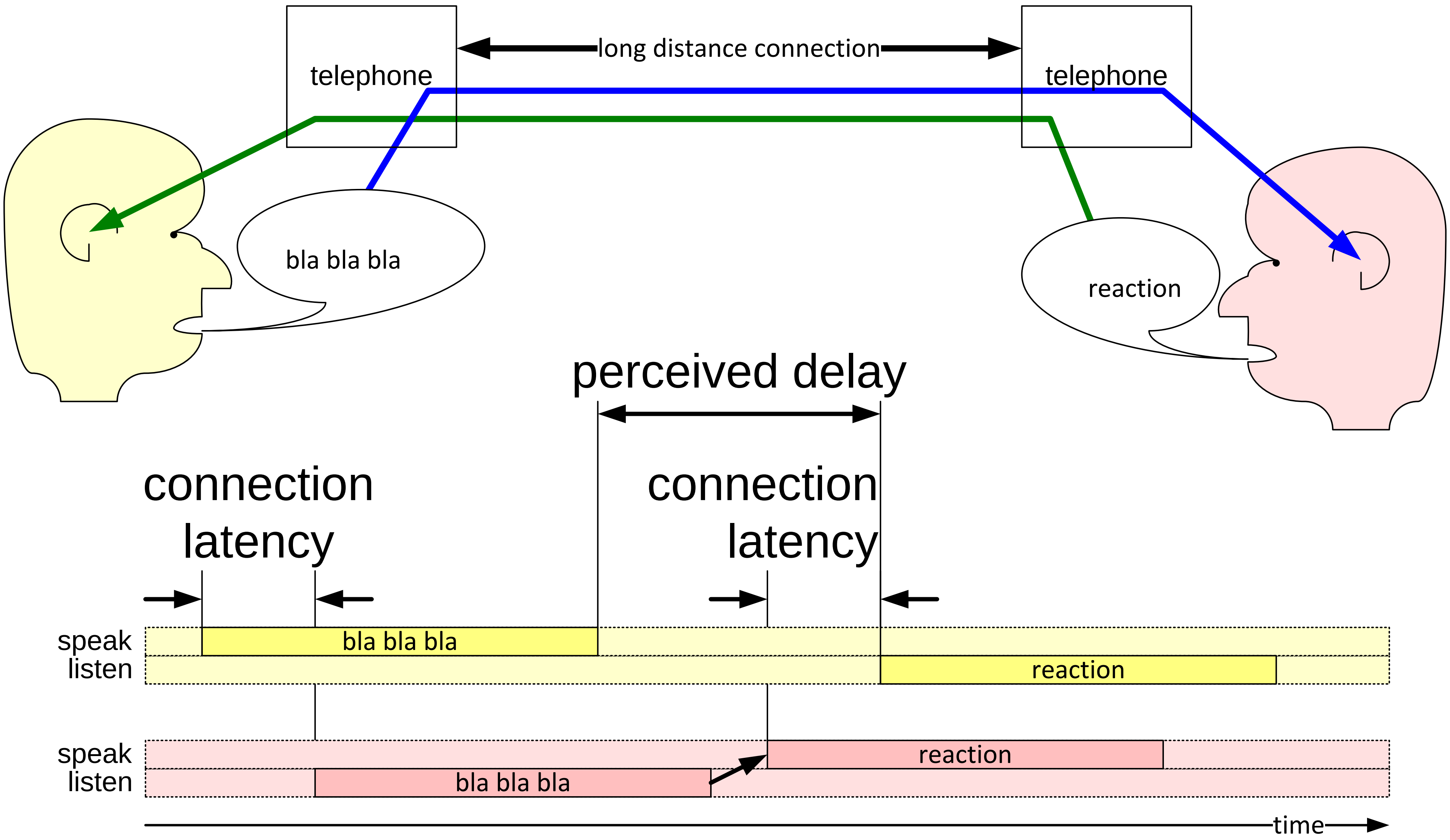
Well defined set of performance critical cases



Typical within desired time, limited exceptions allowed.

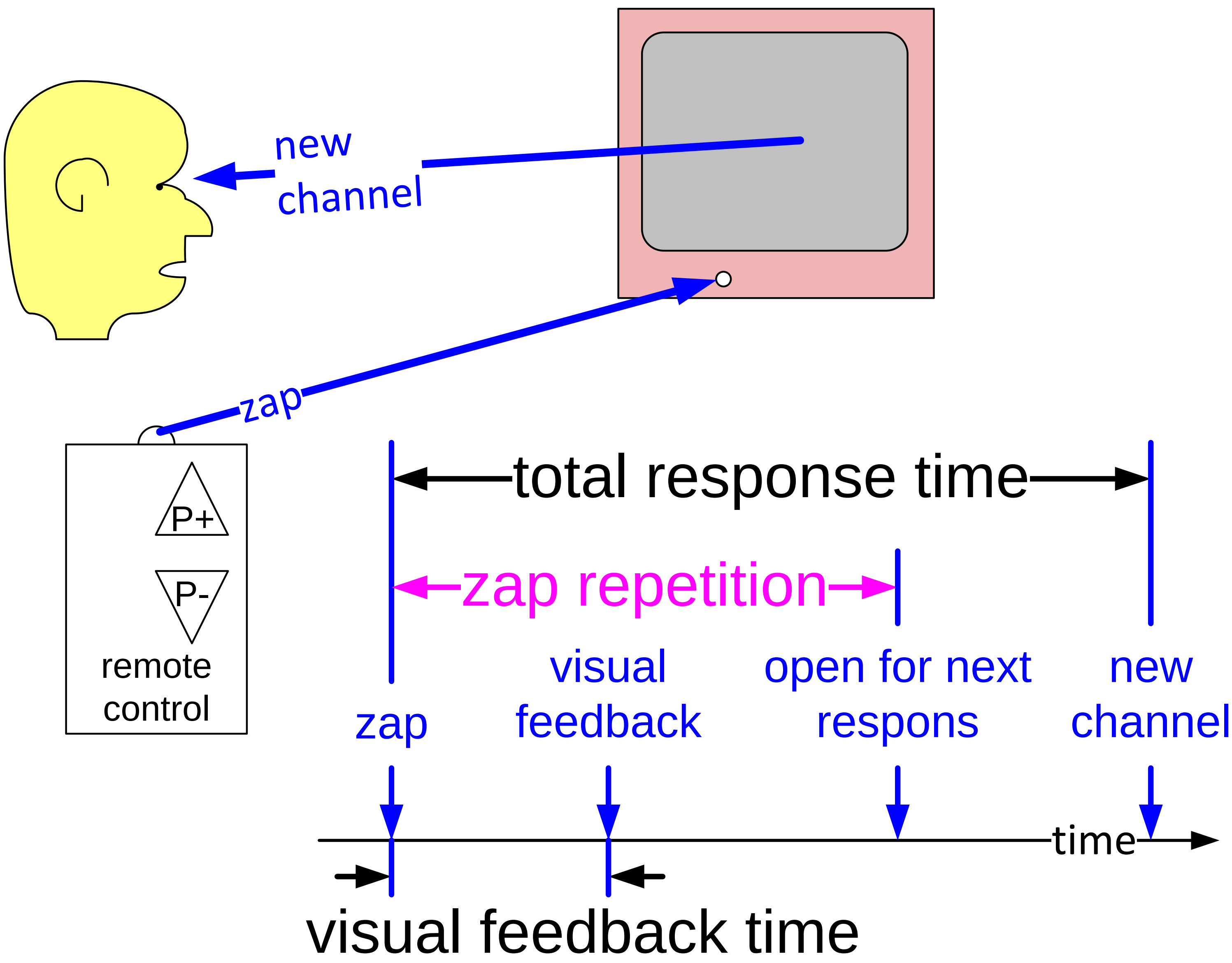
Exceptions may not result in functional failure

Latency

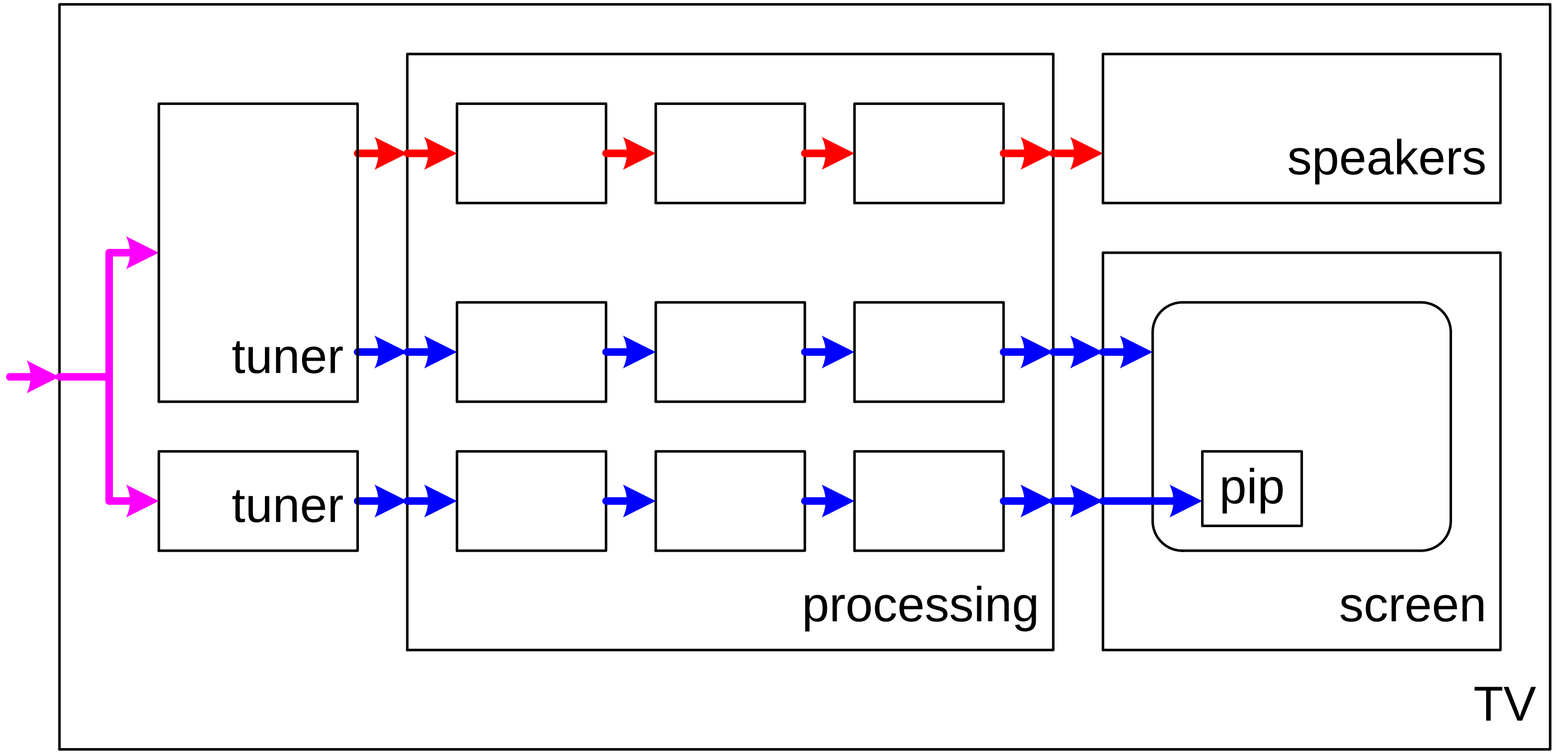


telephone
long distance connection
telephone
bla bla bla
reaction
perceived delay
connection latency
connection latency
speak
listen
bla bla bla
reaction
speak
listen
reaction
bla bla bla
time

Response Time



Throughput



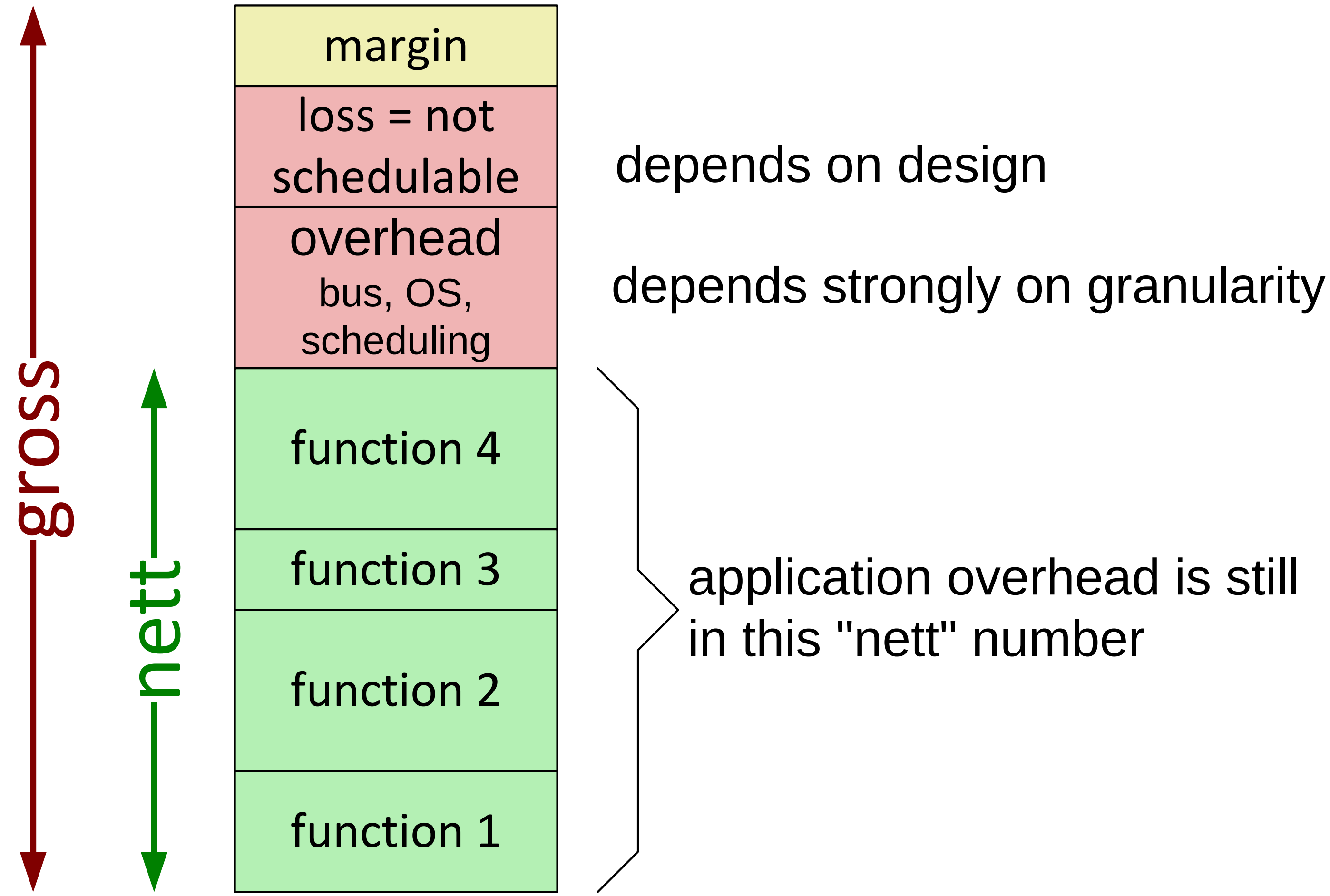
throughput:

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

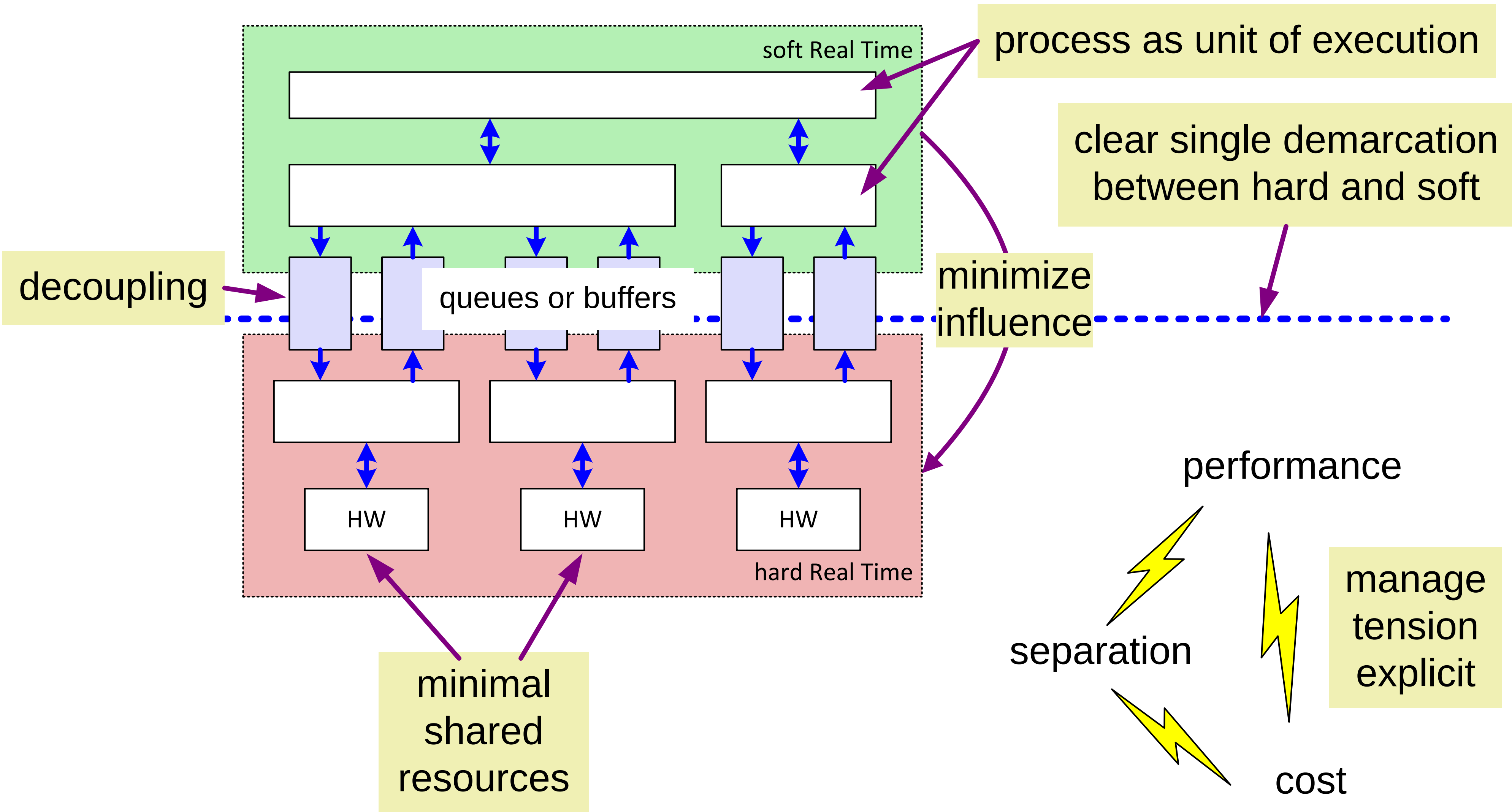
Gross versus Nett

bus bandwidth, processor load [memory usage]

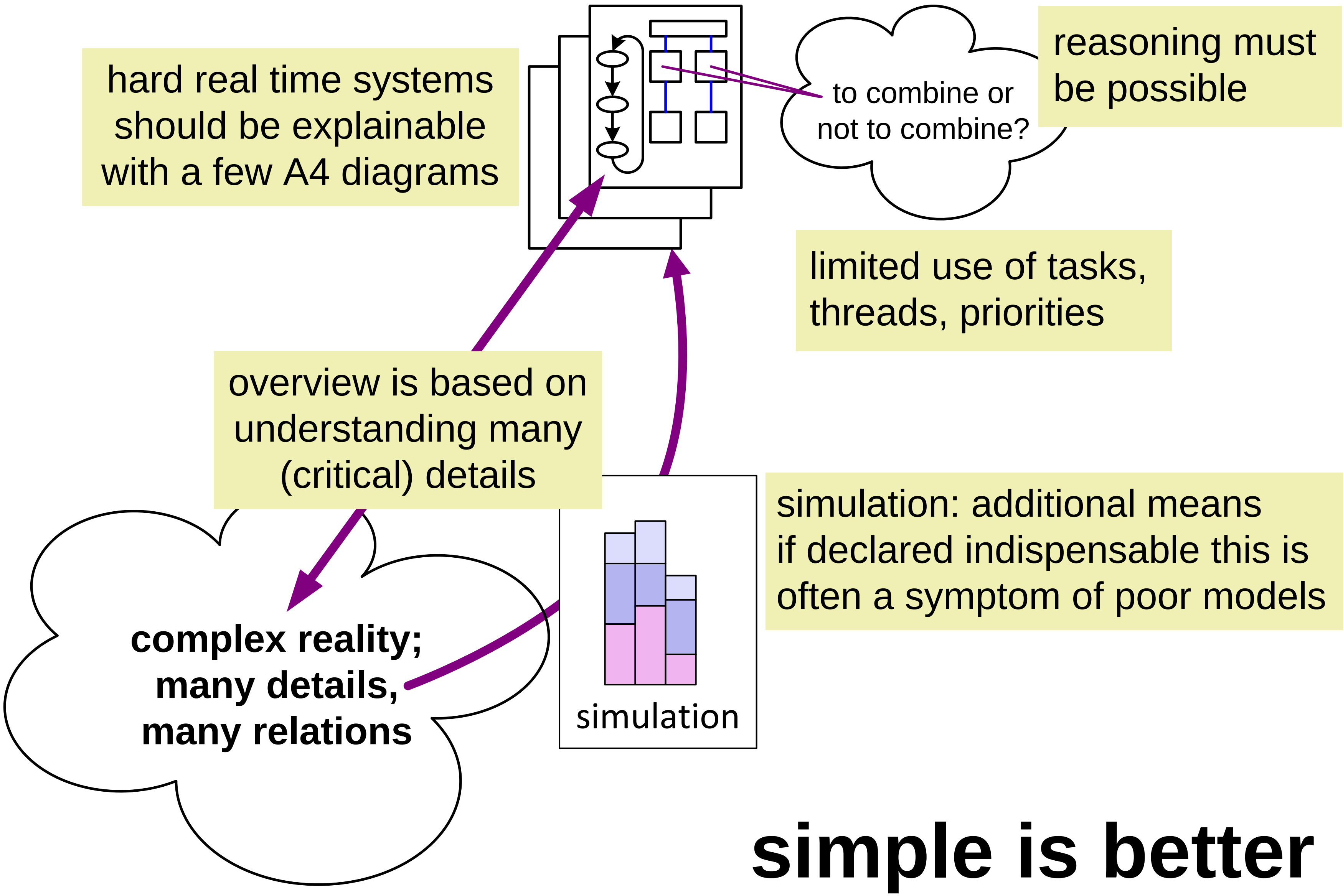
useful macroscopic views, be aware of microscopic behavior



Design recommendations separation of concerns



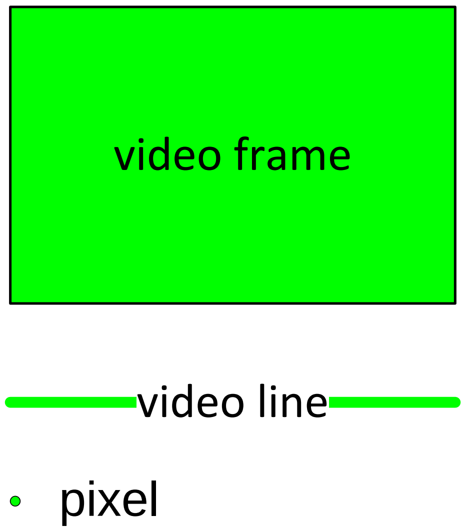
Design recommendations understandability

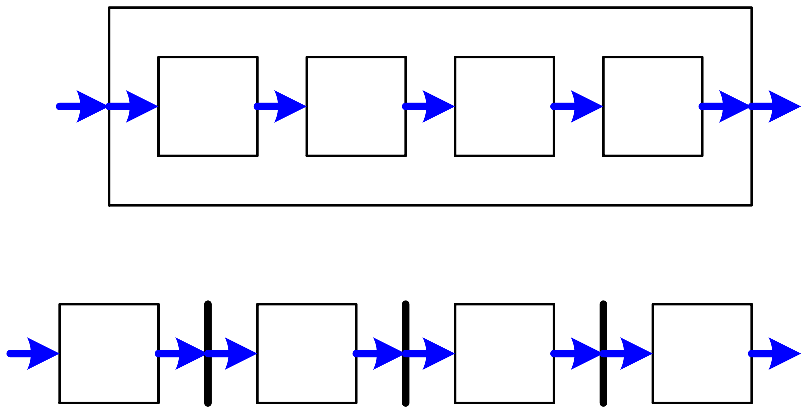


simple is better

Granularity considerations

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





fine grain:	*coarse grain:*
flexible	rigid
high overhead	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;

low "tunable" overhead
understandable

fast rhythms significant overhead

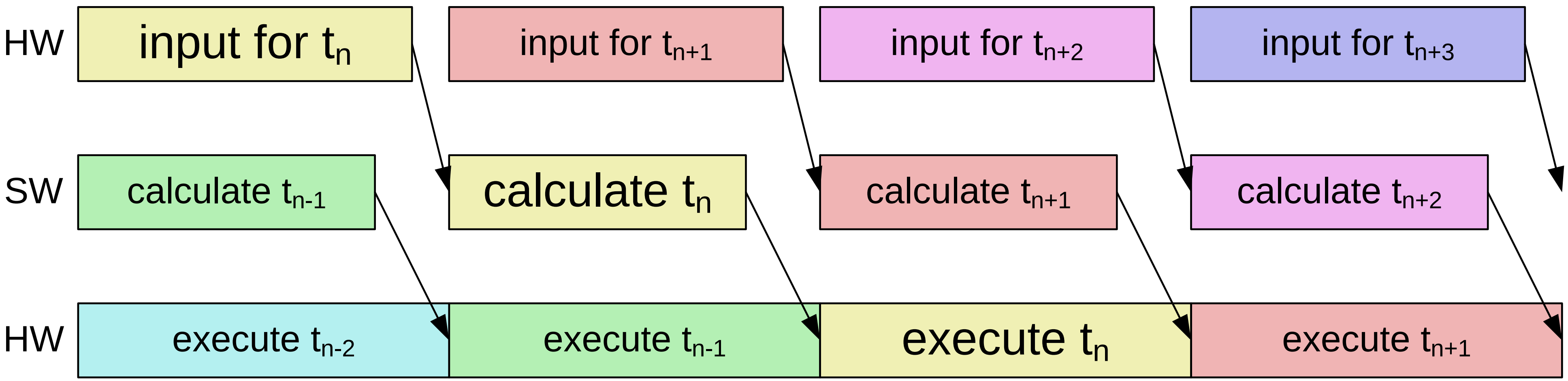
interrupt based

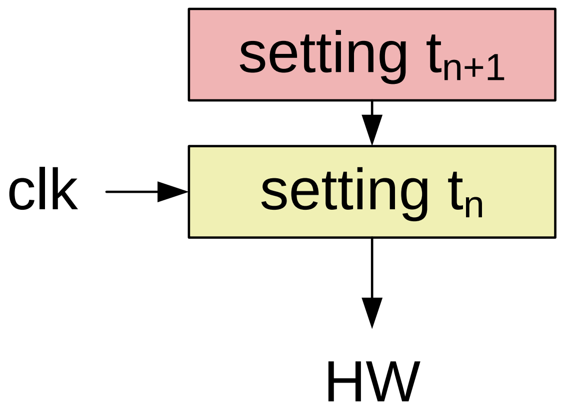
I/O and HW events

separation of timing concerns

definition of interrupts determines:
overhead, understandability

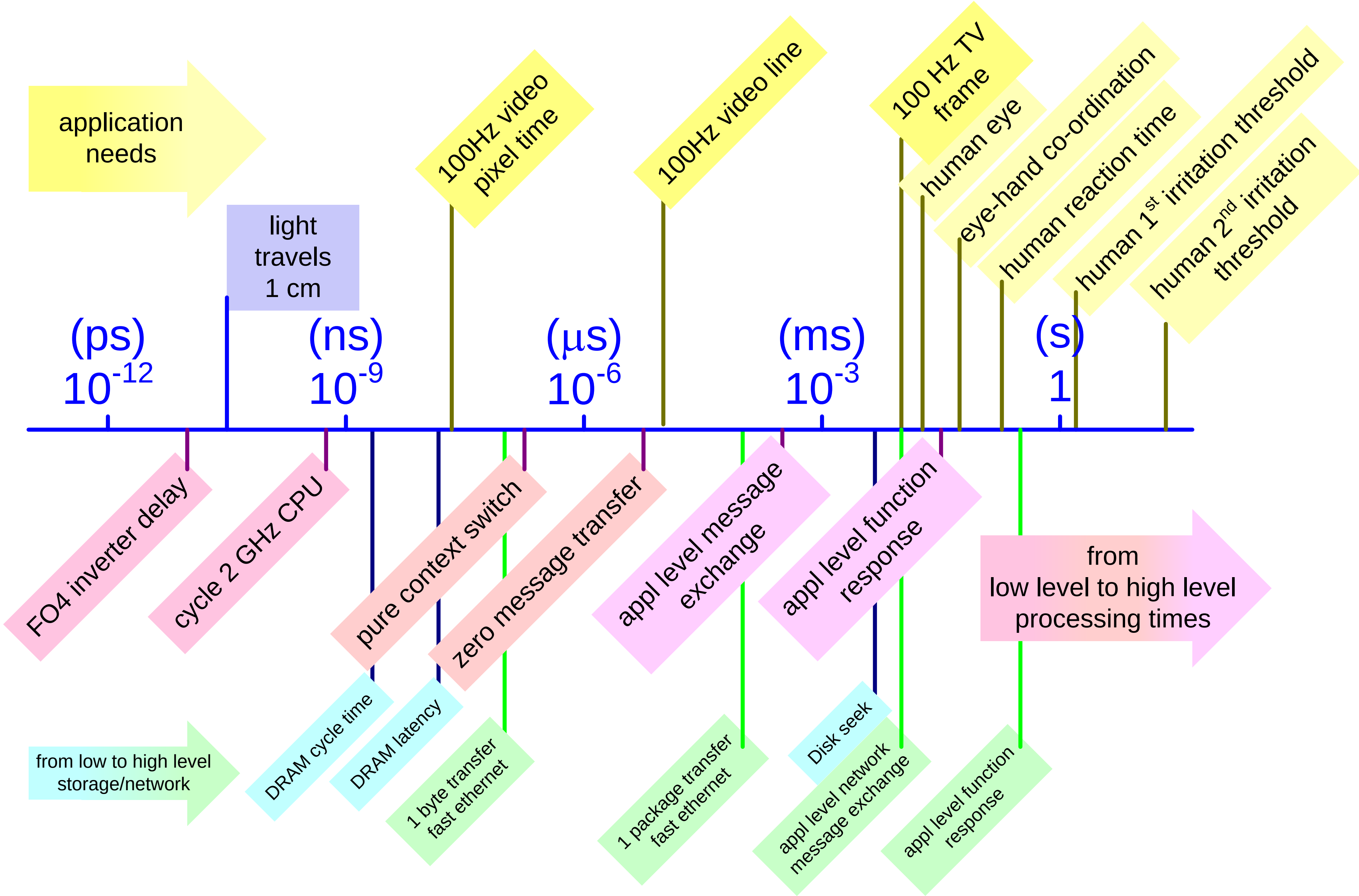
Synchronous design





double buffer:
full decoupling of calculation and execution

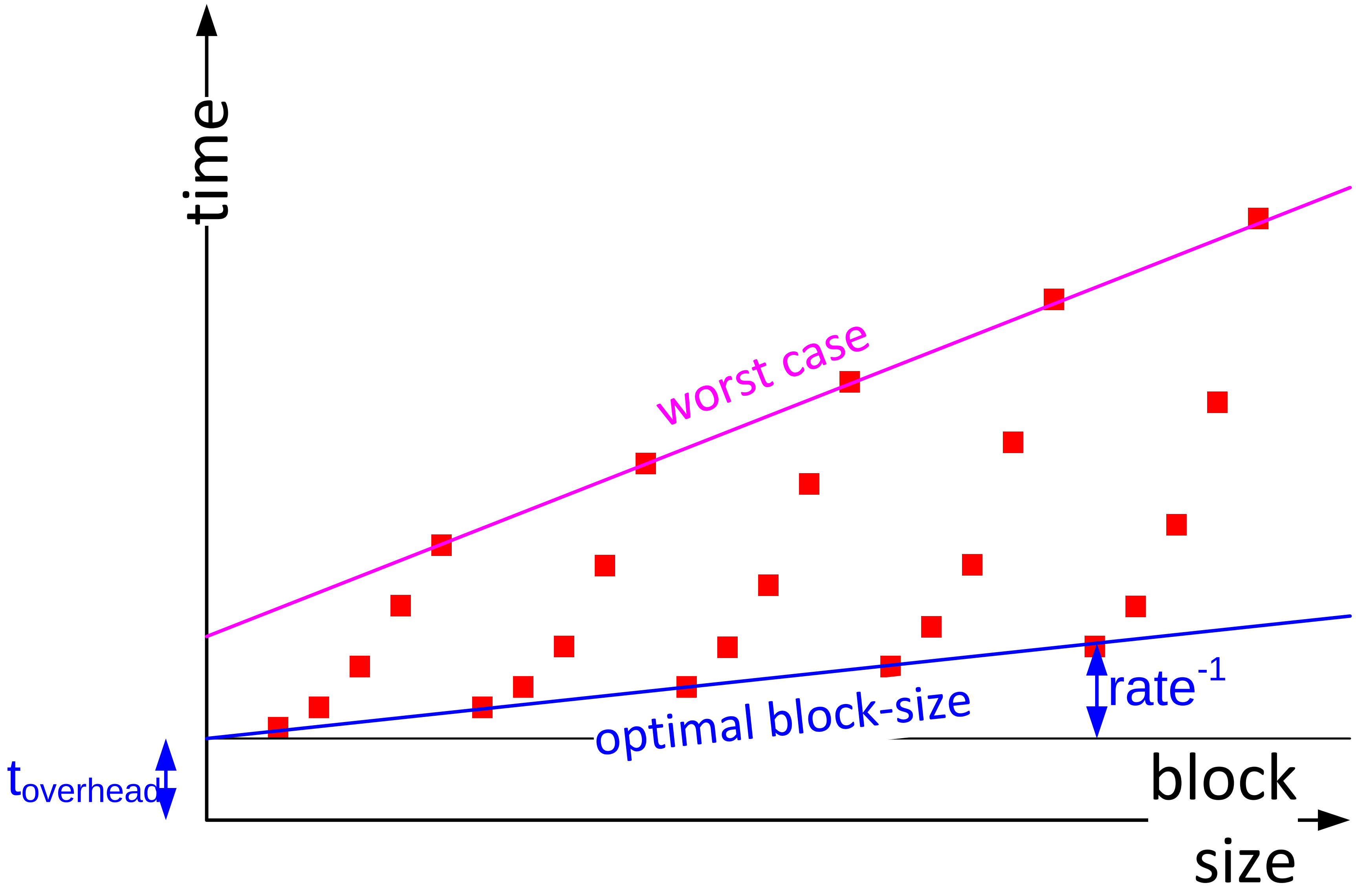
Actual timing on logarithmic scale



Typical micro benchmarks for timing aspects

	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

The transfer time as function of blocksize



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

Complicating factors and measures

complications	*measures*
cache	considered margin
bus allocation	explicit behavior
memory management	architecture rules
garbage collection	monitoring, logging
memory (buffer, storage) fragmentation	pool management
non preemptable OS activities	feedback to architect
"hidden" dependencies (ie [dead]locks)	flipover simulation
systematic "coincidences", avalanche triggers	
instable response, performance	