

# Design Objectives and Design Understandability

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

## Abstract

The complexity of systems limits the understanding by the architect of the impact of changes. Many objectives are pursued, from customer needs to implementation lessons learned, while designing a system. From architecting perspective *understandability* of the design is an important issue. Some design choices may create very efficient systems, but might be difficult to grasp. For example simple local autonomy might prove to be efficient and robust, but at the same time other system qualities are emerging and difficult to predict. We discuss the notion of understandability, illustrated by a number of design patterns.

### Distribution

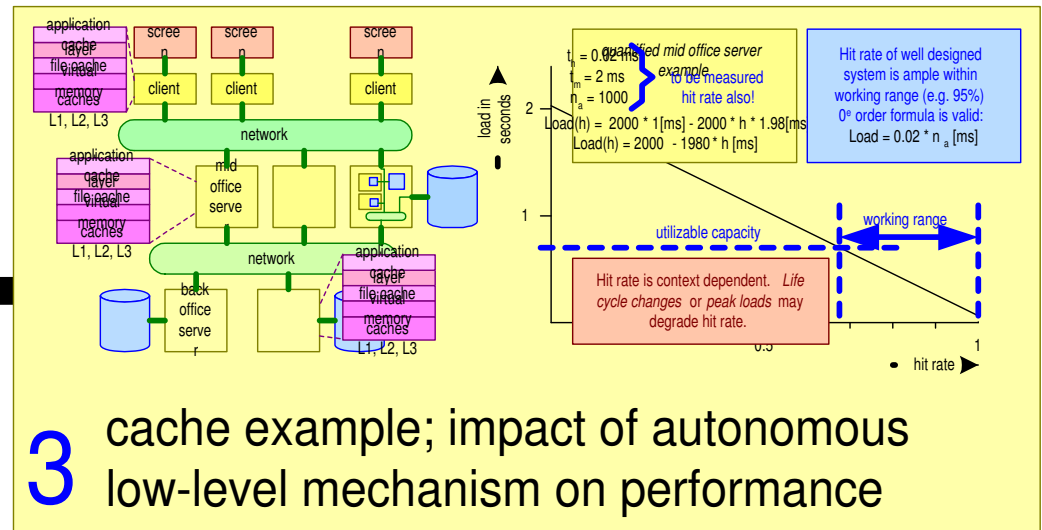
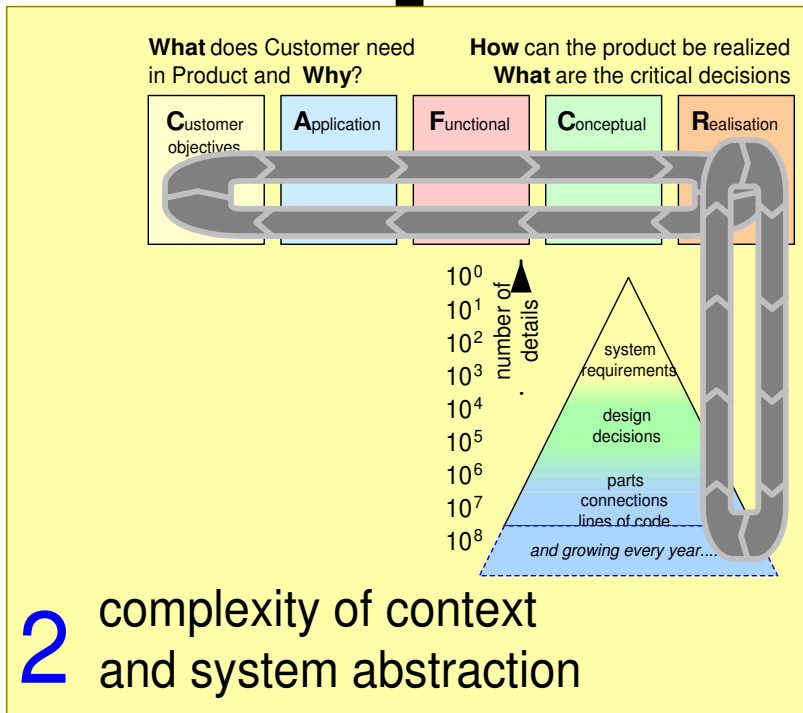
This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution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

July 20, 2011  
status: planned  
version: 0

logo  
TBD

# Figure Of Contents™

1 performance example:  
do we understand our design?



4 discussion and  
conclusion

# Image Retrieval Performance

application need:

at event 3\*3 show 3\*3 images  
instantaneous

design

design

Sample application code:

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

or

alternative application code:

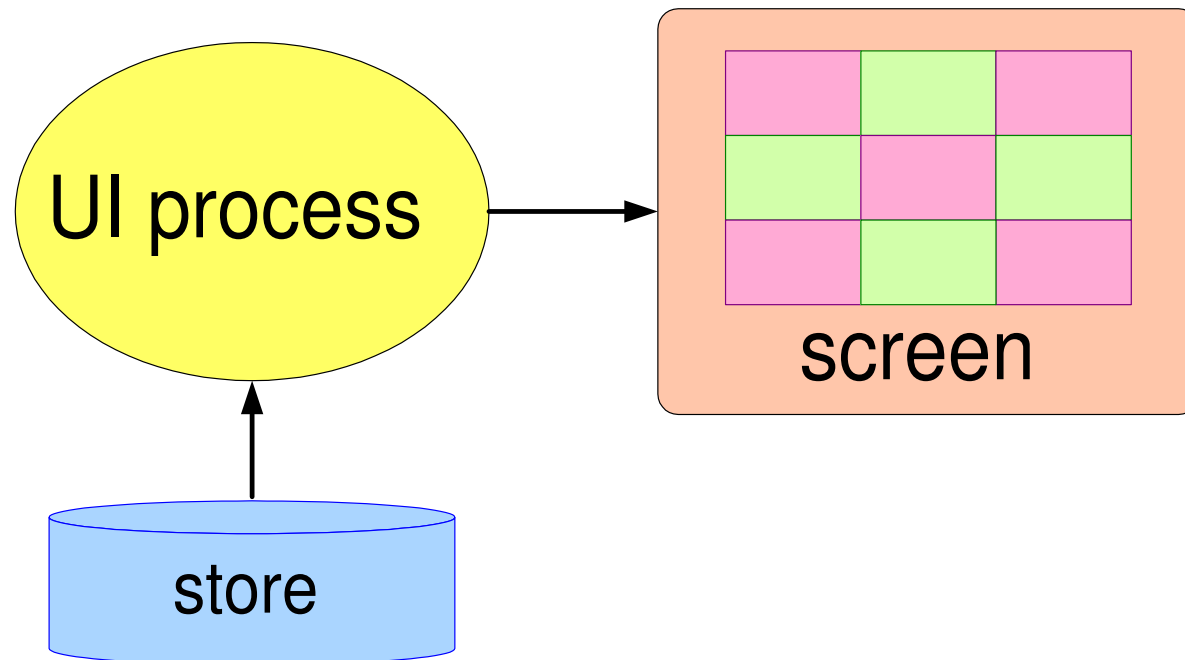
event 3\*3 -> show screen 3\*3

```
<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:

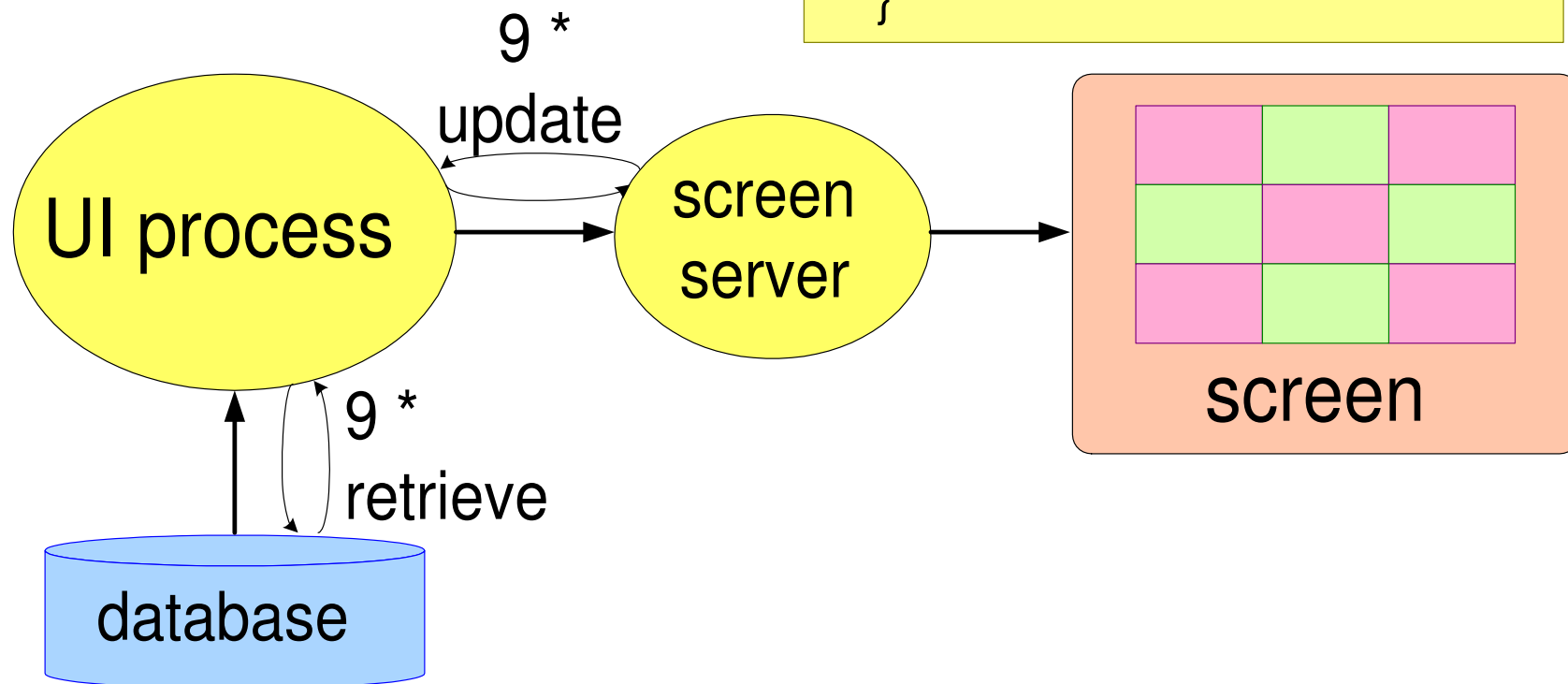
```
for x = 1 to 3 {  
  for y = 1 to 3 {  
    retrieve_image(x,y)  
  }  
}
```



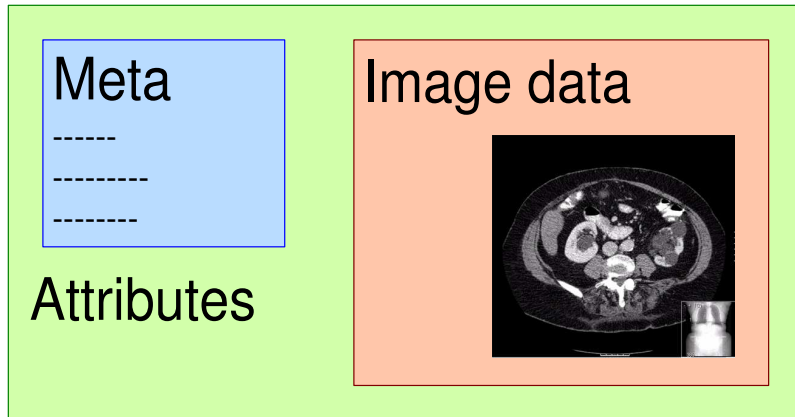
## What If....

Sample application code:

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

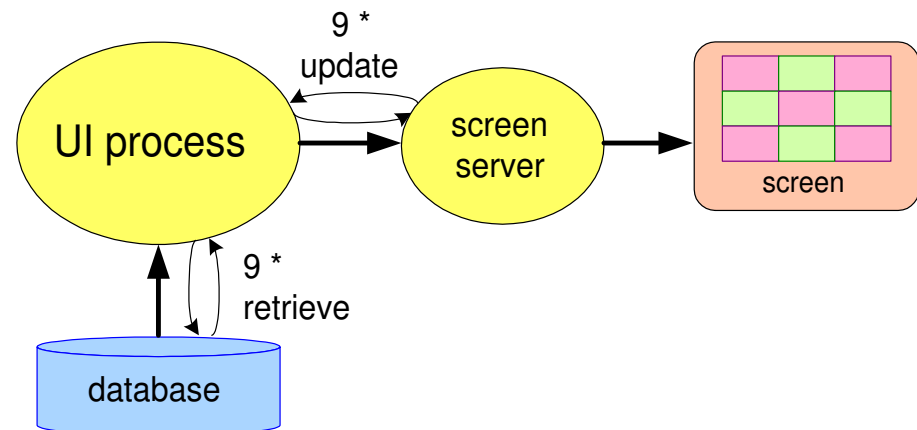


## What If....



```
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 = 80 $\mu$ s  
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<sup>2</sup> image)

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

$$t_{I/O} \approx 1ms$$

- . . .

# 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:

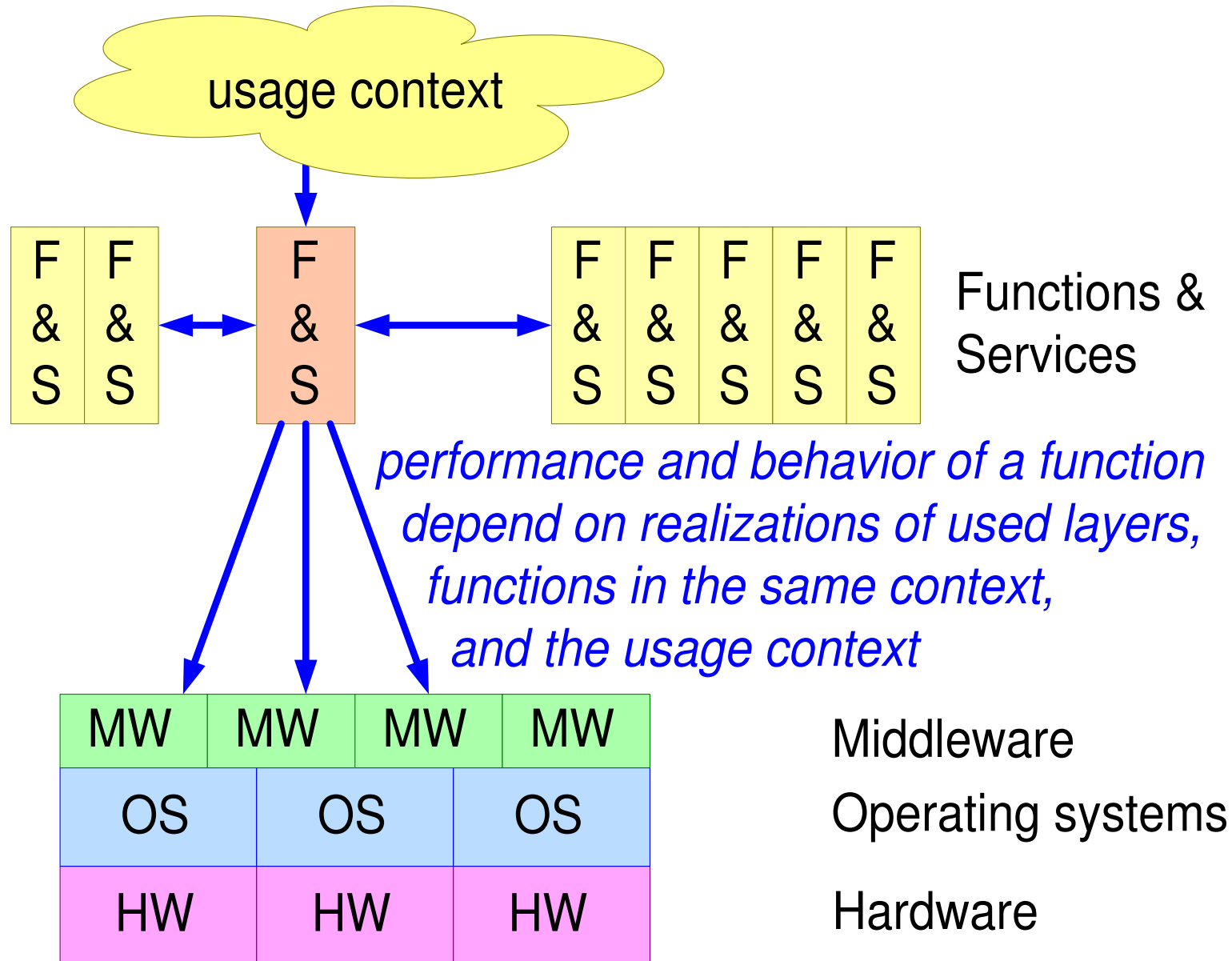
fast, but very local  
slow, but very generic  
slow, but very robust  
fast and robust

...

*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

F	F	F	F	F	F	F	F
&	&	&	&	&	&	&	&
S	S	S	S	S	S	S	S
MW		MW		MW		MW	
OS		OS		OS			
HW		HW		HW			

Functions & Services

Middleware

Operating systems

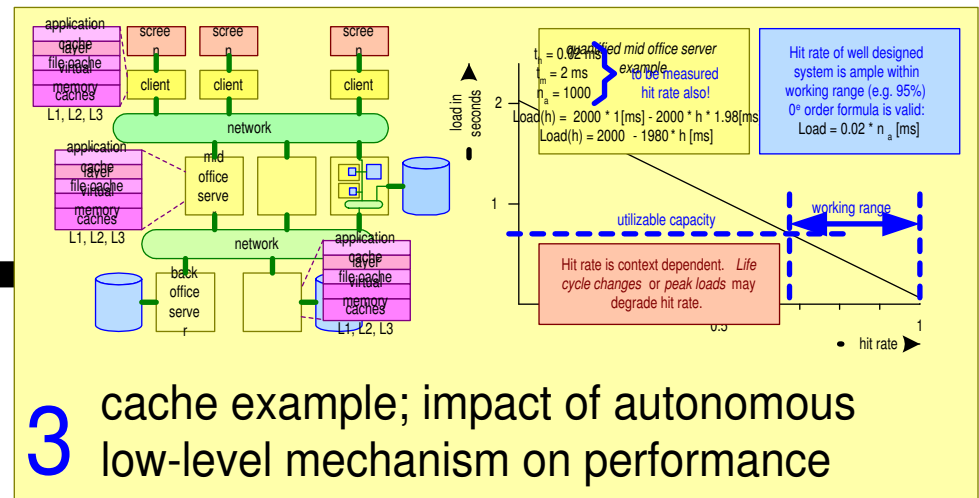
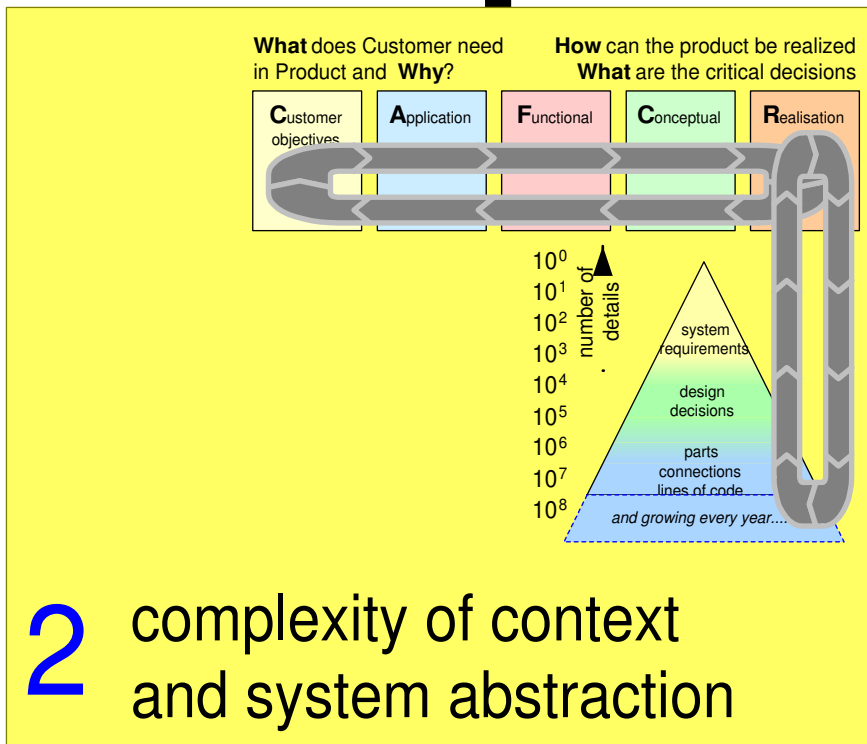
Hardware

Performance = Function (F&S, other F&S, MW, OS, HW)

MW, OS, HW >> 100 Manyear : very complex

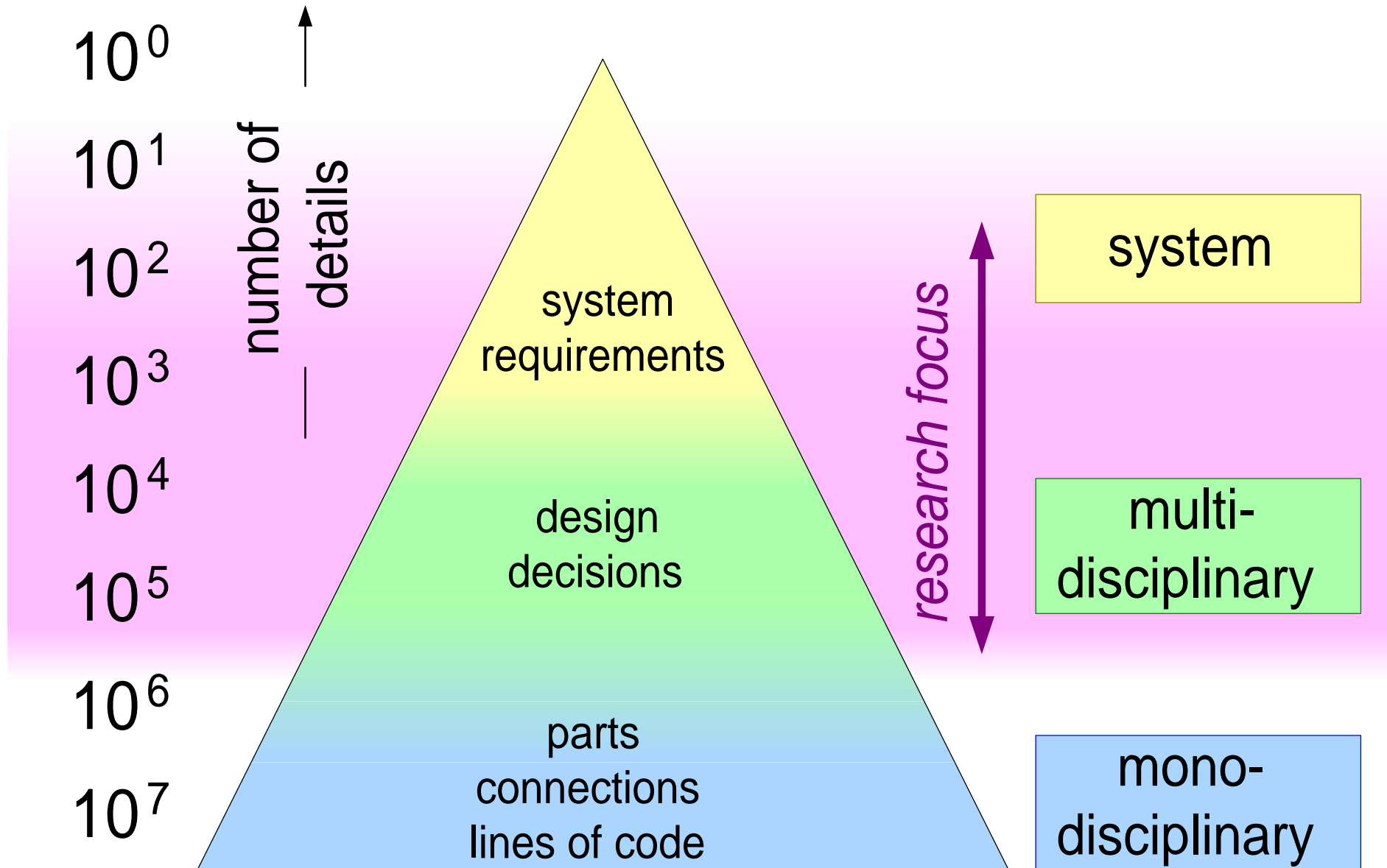
Challenge: How to understand MW, OS, HW  
with only a few parameters

1 performance example:  
do we understand our design?

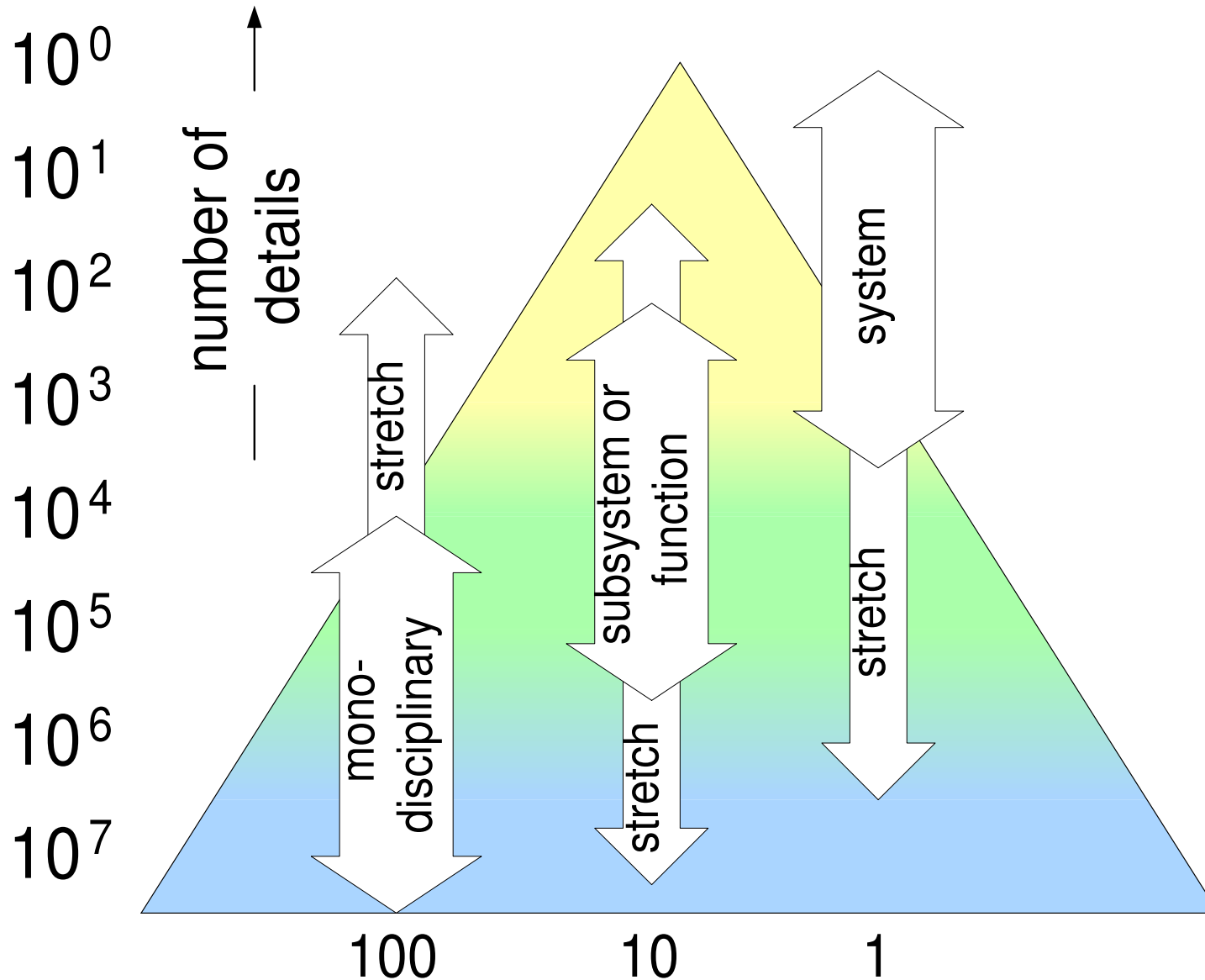


4 discussion and conclusion

# Exponential Pyramid, from requirement to bolts and nuts

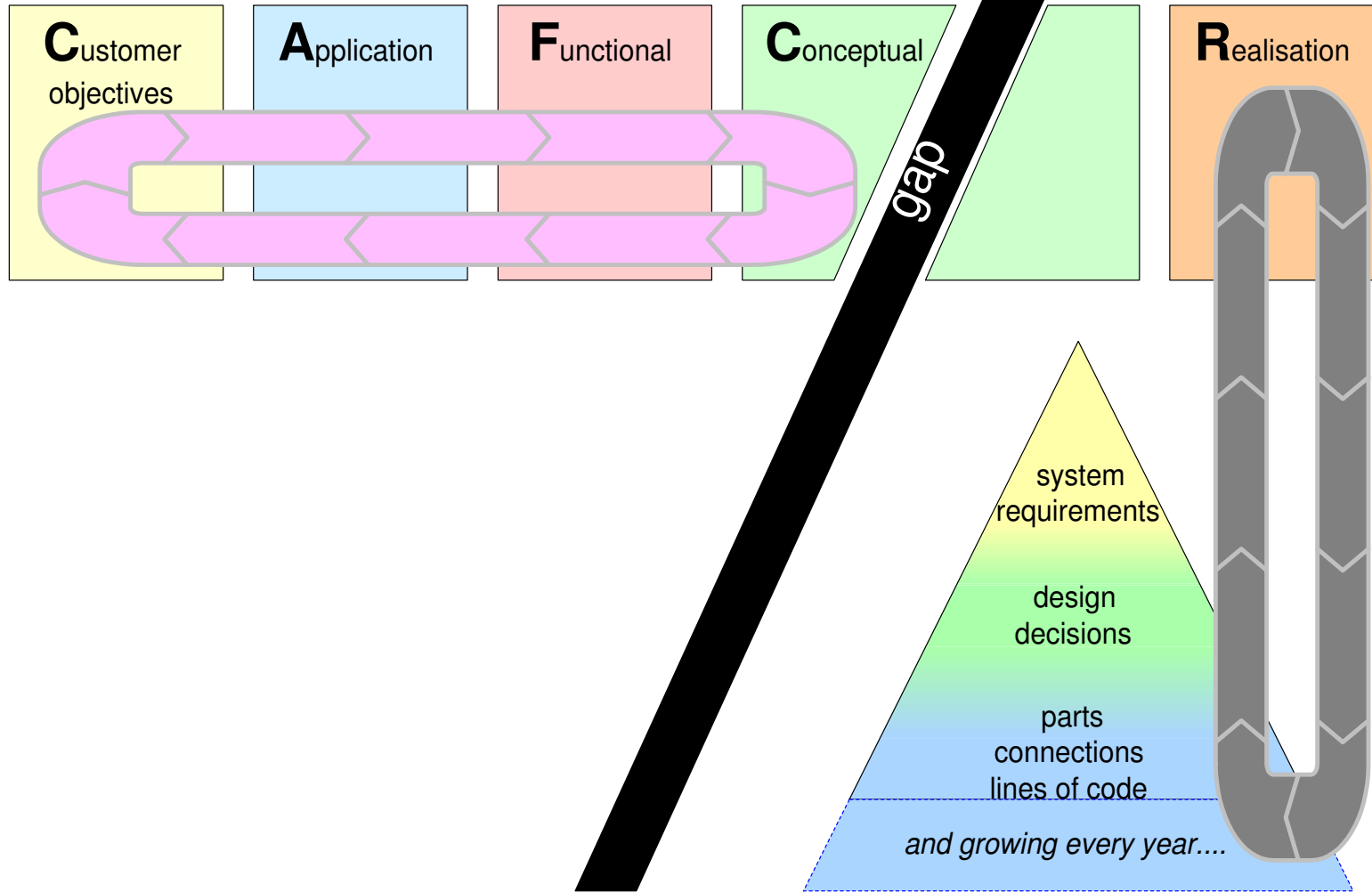


# Major Bottleneck: Mental Dynamic Range



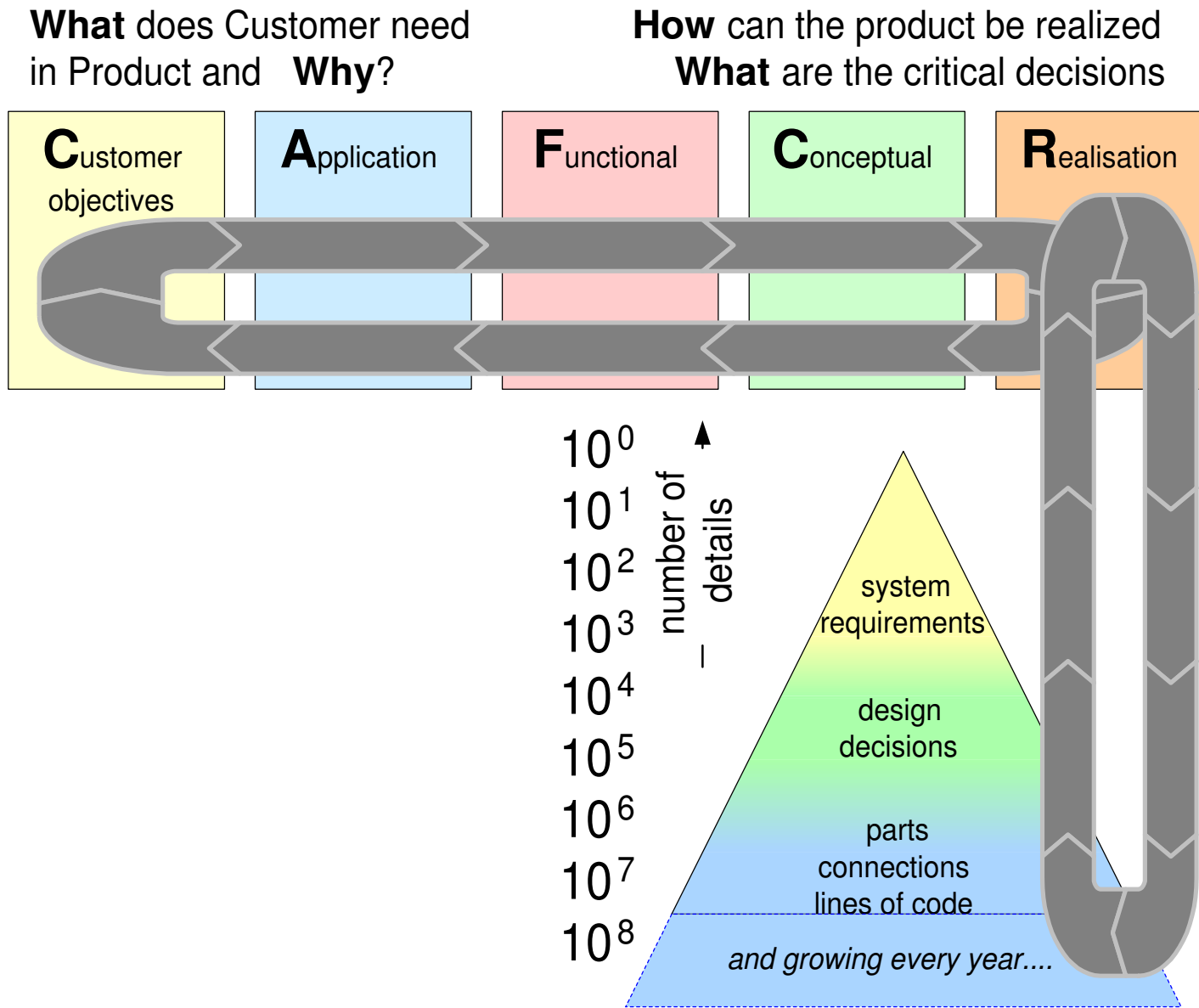
# Organizational Problem: Disconnect

**What** does Customer need  
in Product and **Why?**

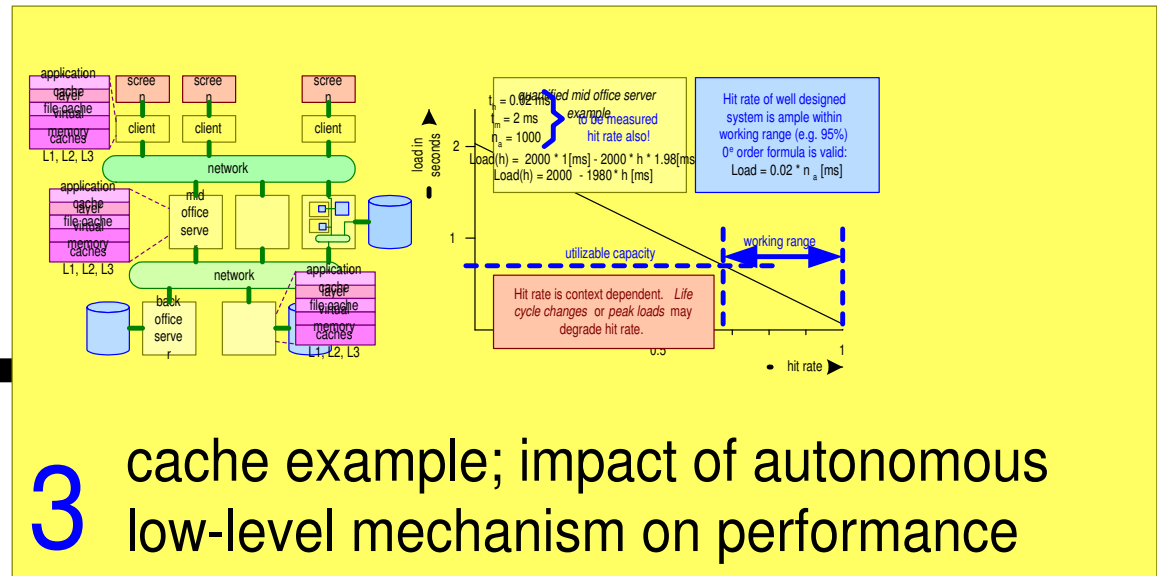
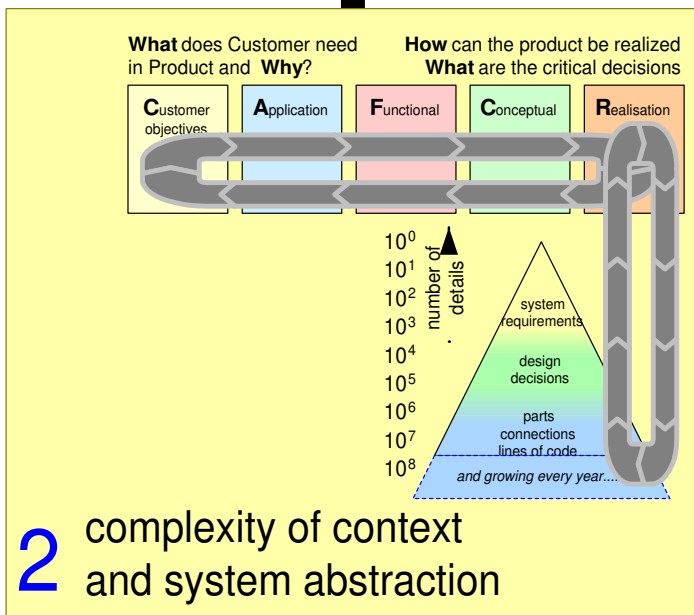


**How** can the product be realized  
**What** are the critical decisions

# Architect: Connecting Problem and Technical Solution

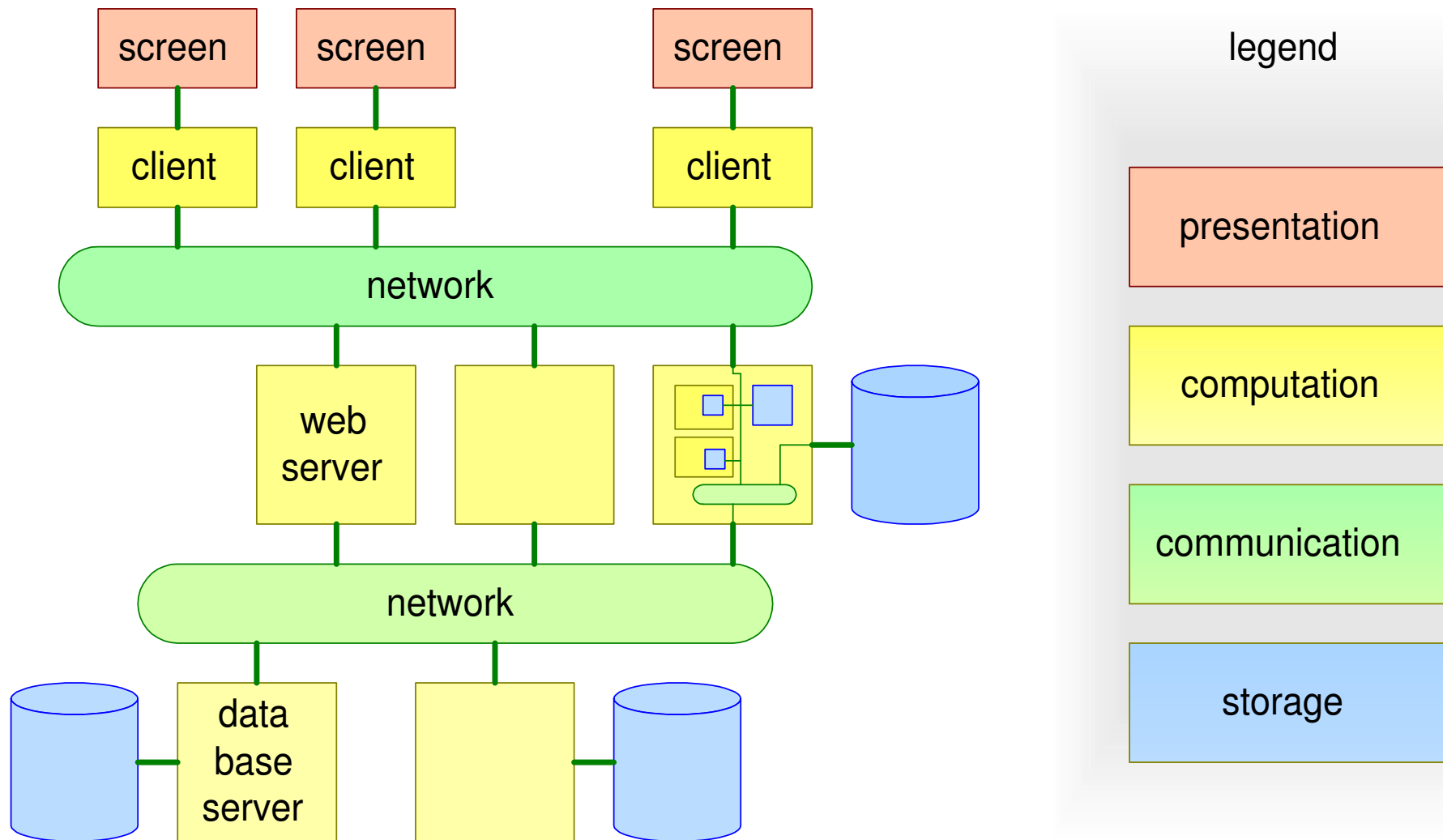


1 performance example:  
do we understand our design?



4 discussion and  
conclusion

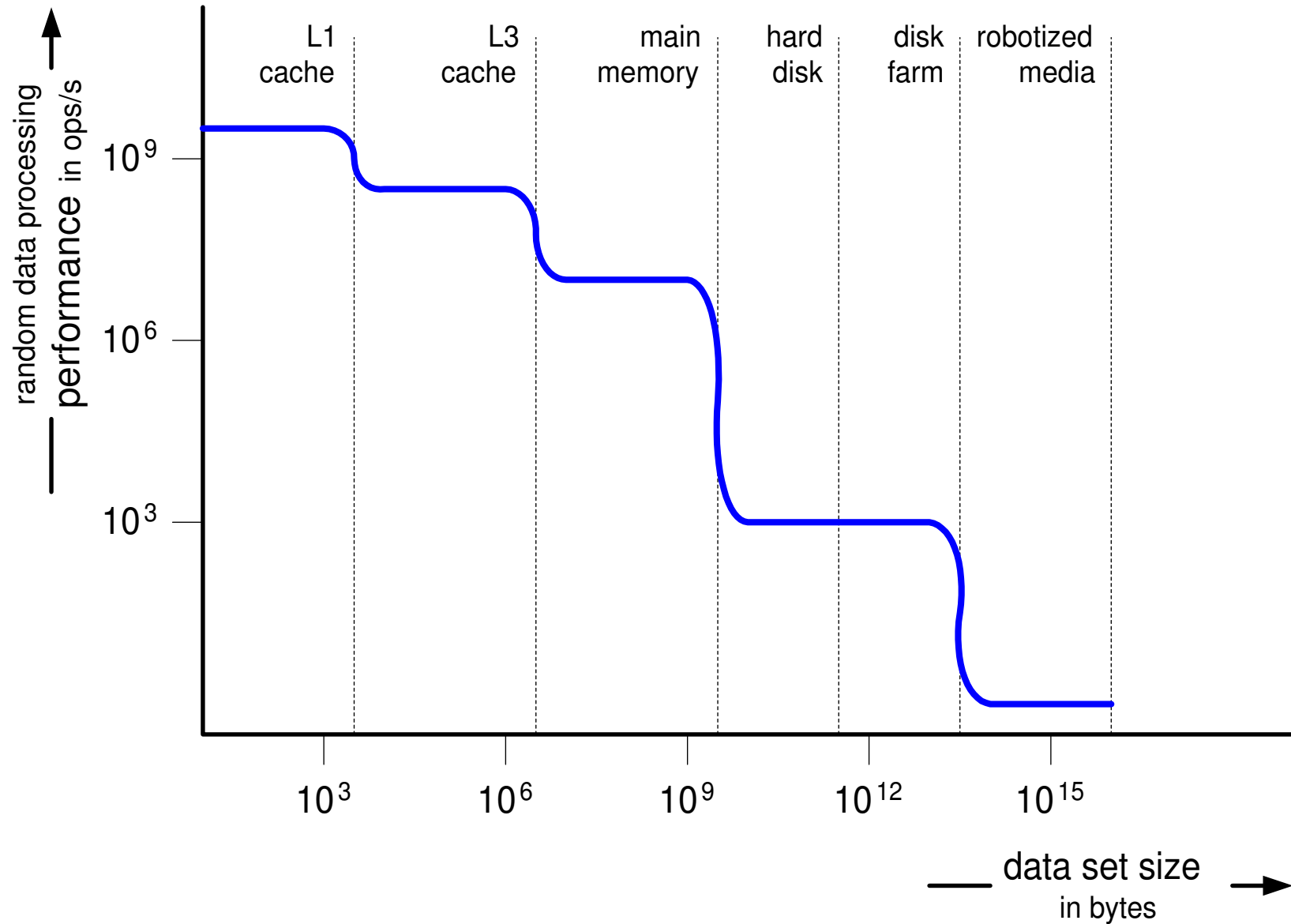
# Typical Block Diagram and Typical Resources in IT



# Hierarchy of Storage Technology Figures of Merit

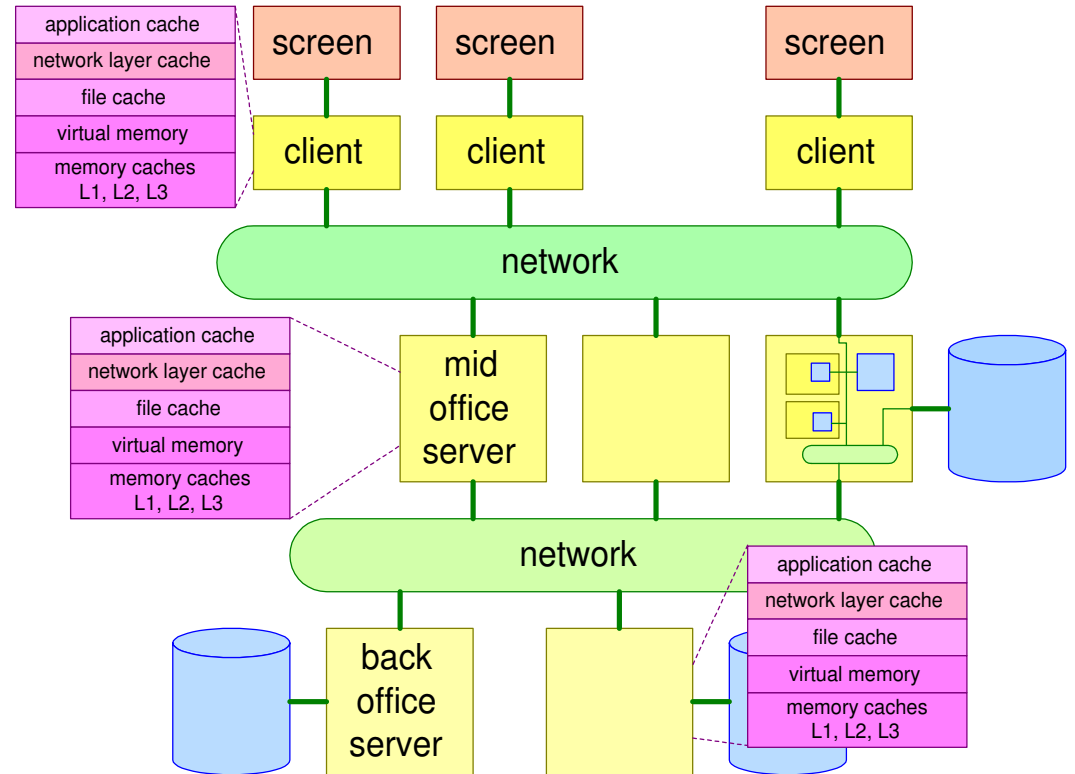
		latency	capacity
processor cache	<i>L1 cache</i>	sub ns	n kB
	<i>L2 cache</i>		
	<i>L3 cache</i>	ns	n MB
fast volatile	<i>main memory</i>	tens ns	n GB
persistent	<i>disks</i>		n*100 GB
	<i>disk arrays</i>	ms	
	<i>disk farms</i>		n*10 TB
archival	<i>robotized optical media tape</i>	>s	n PB

# Performance as Function of Data Set Size

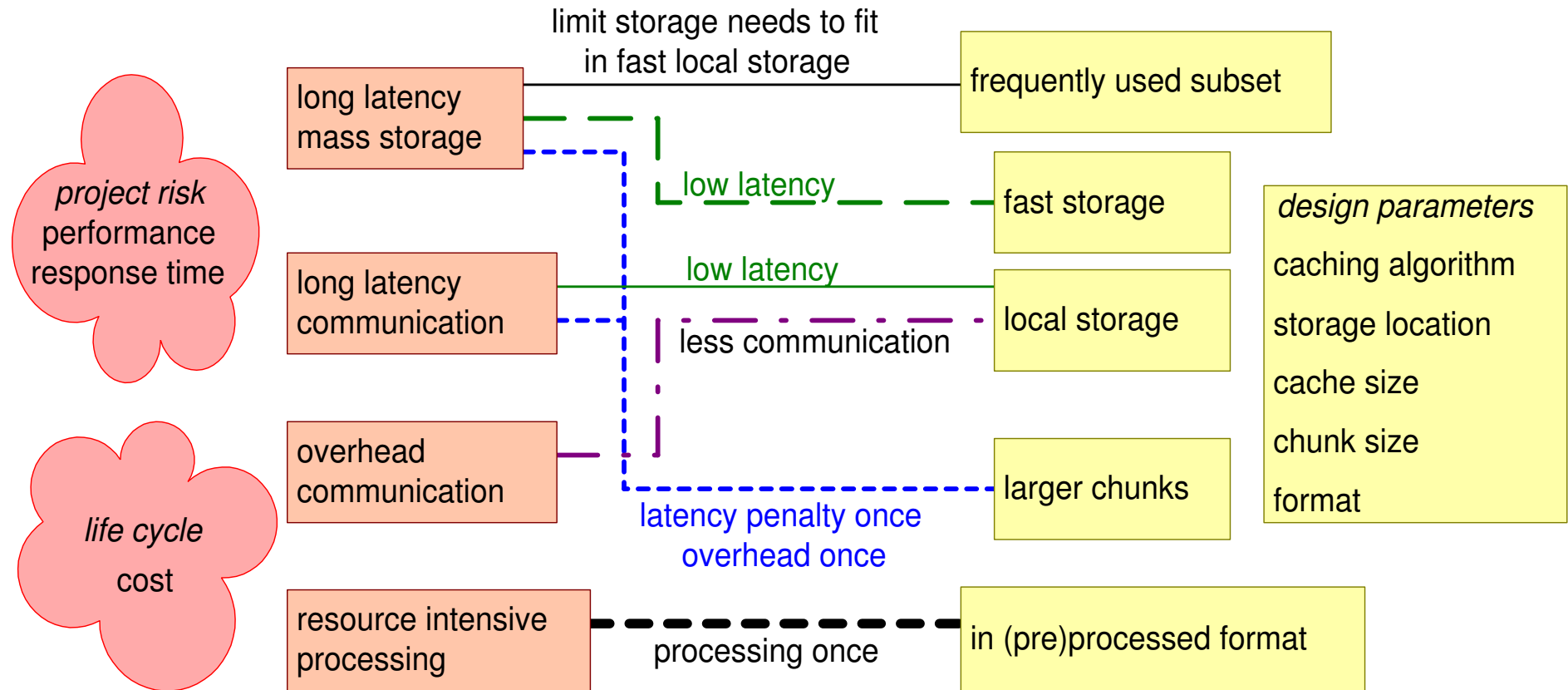


# Multiple Layers of Caching

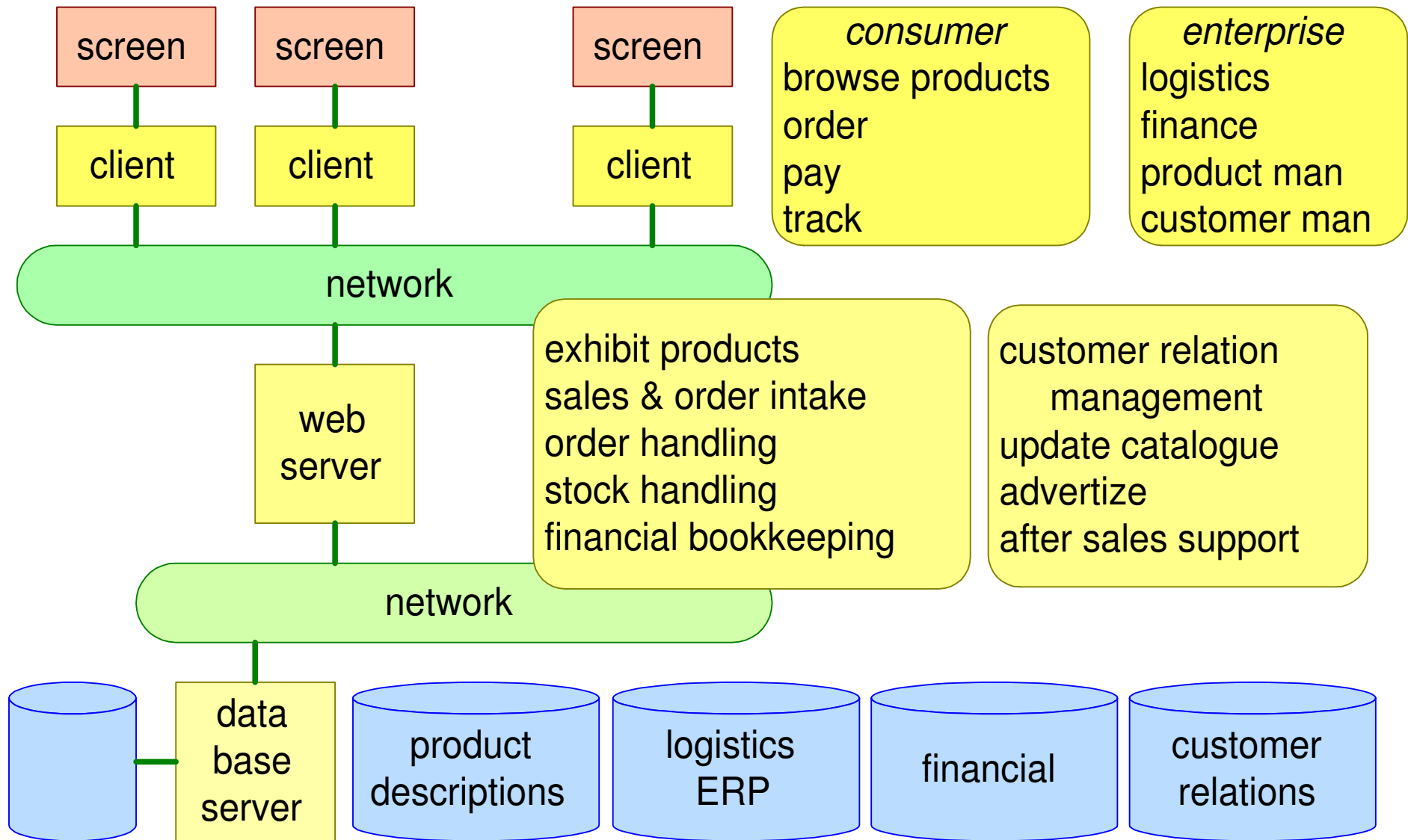
	cache miss penalty	cache hit performance
application cache	1 s	10 ms
network layer cache	100 ms	1 ms
file cache	10 ms	10 us
virtual memory	1 ms	100 ns
memory caches L1, L2, L3	100 ns	1 ns



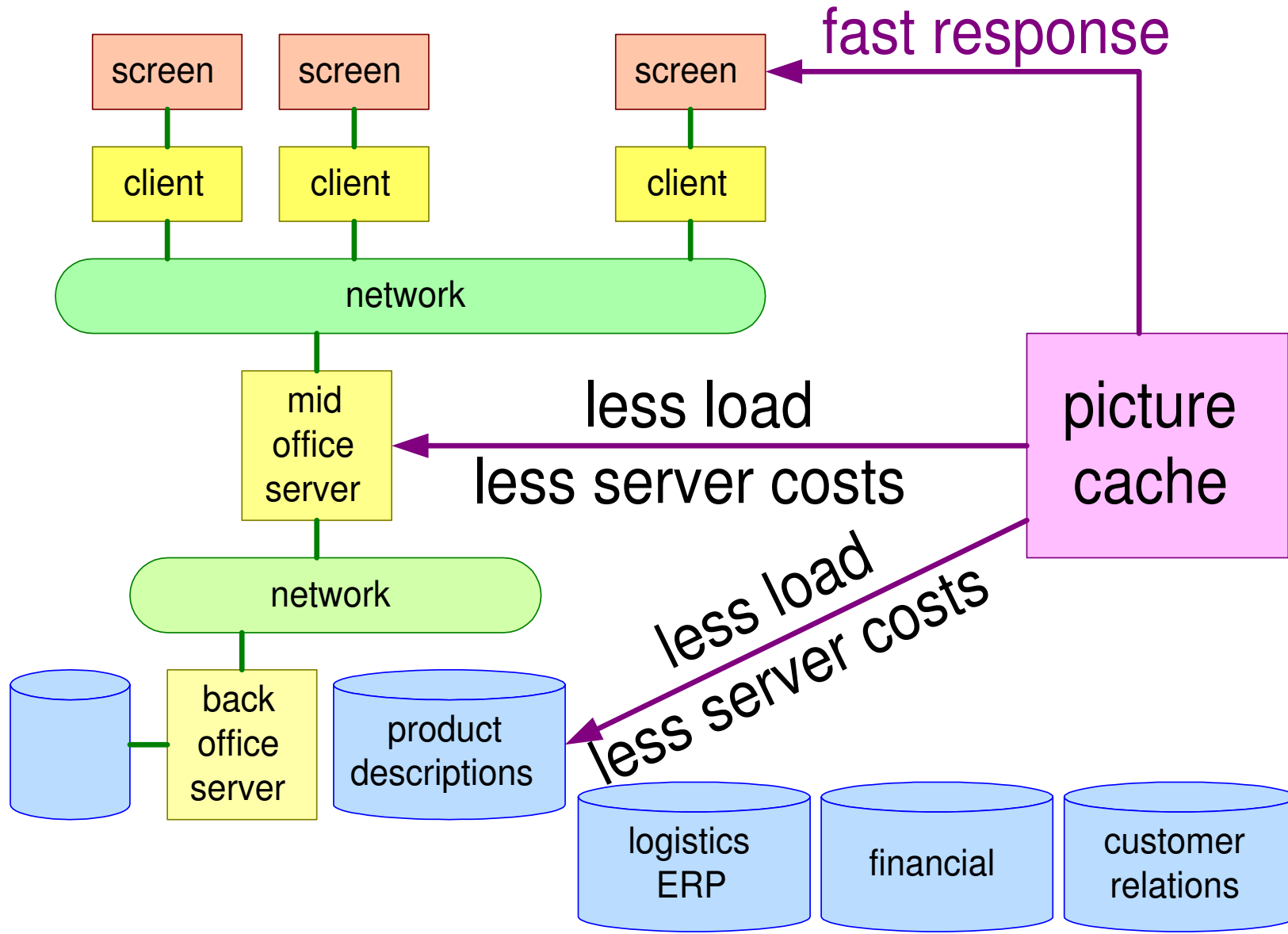
# Why Caching?



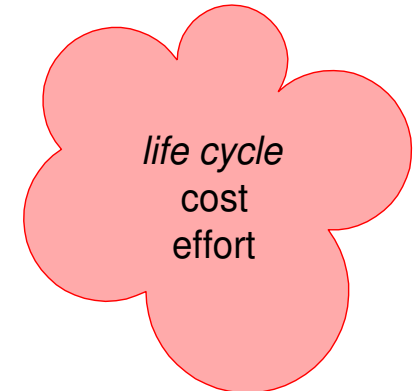
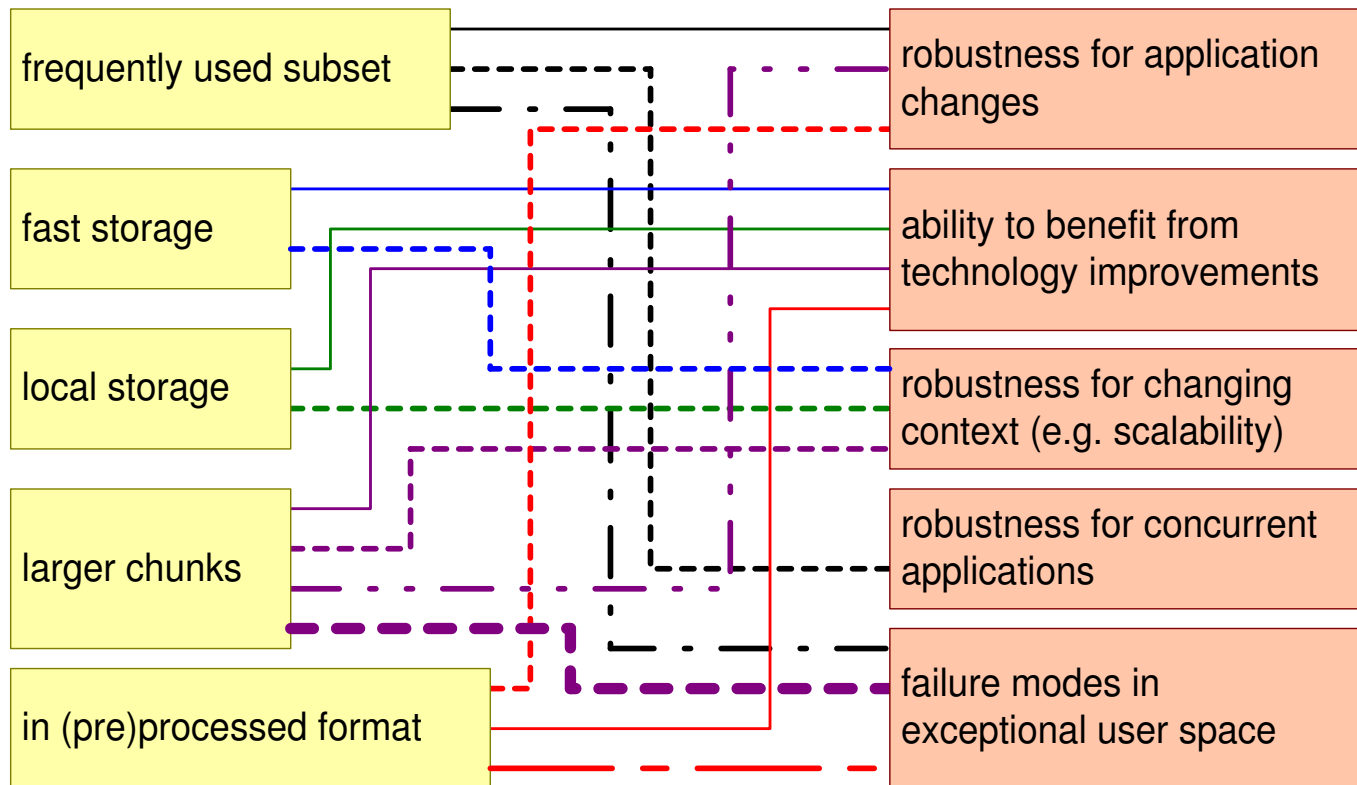
# Example Web Shop



# Impact of Picture Cache



# Risks of Caching



*zero order web server load model*

$$\text{Load} = n_a * t_a$$

$n_a$  = total requests

$t_a$  = cost per request

# First Order Load Model

*first order web server load model*

$$\text{Load} = n_{a,h} * t_h + n_{a,m} * t_m$$

$n_{a,h}$  = accesses with cache hit

$n_{a,m}$  = accesses with cache miss

$t_h$  = cost of cache hit

$t_m$  = cost of cache miss

$$n_{a,h} = n_a * h$$

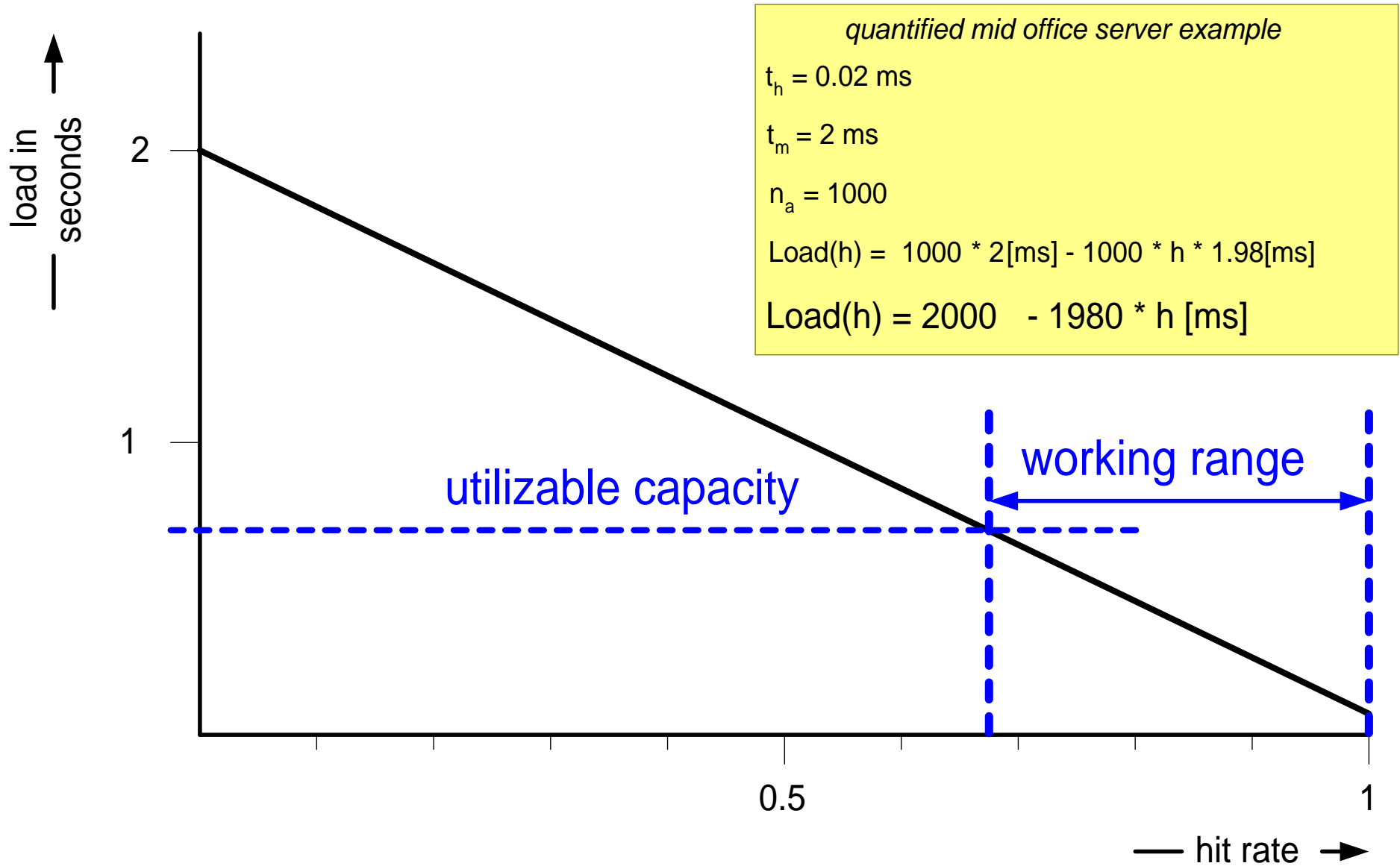
$$n_{a,m} = n_a * (1-h)$$

$n_a$  = total accesses

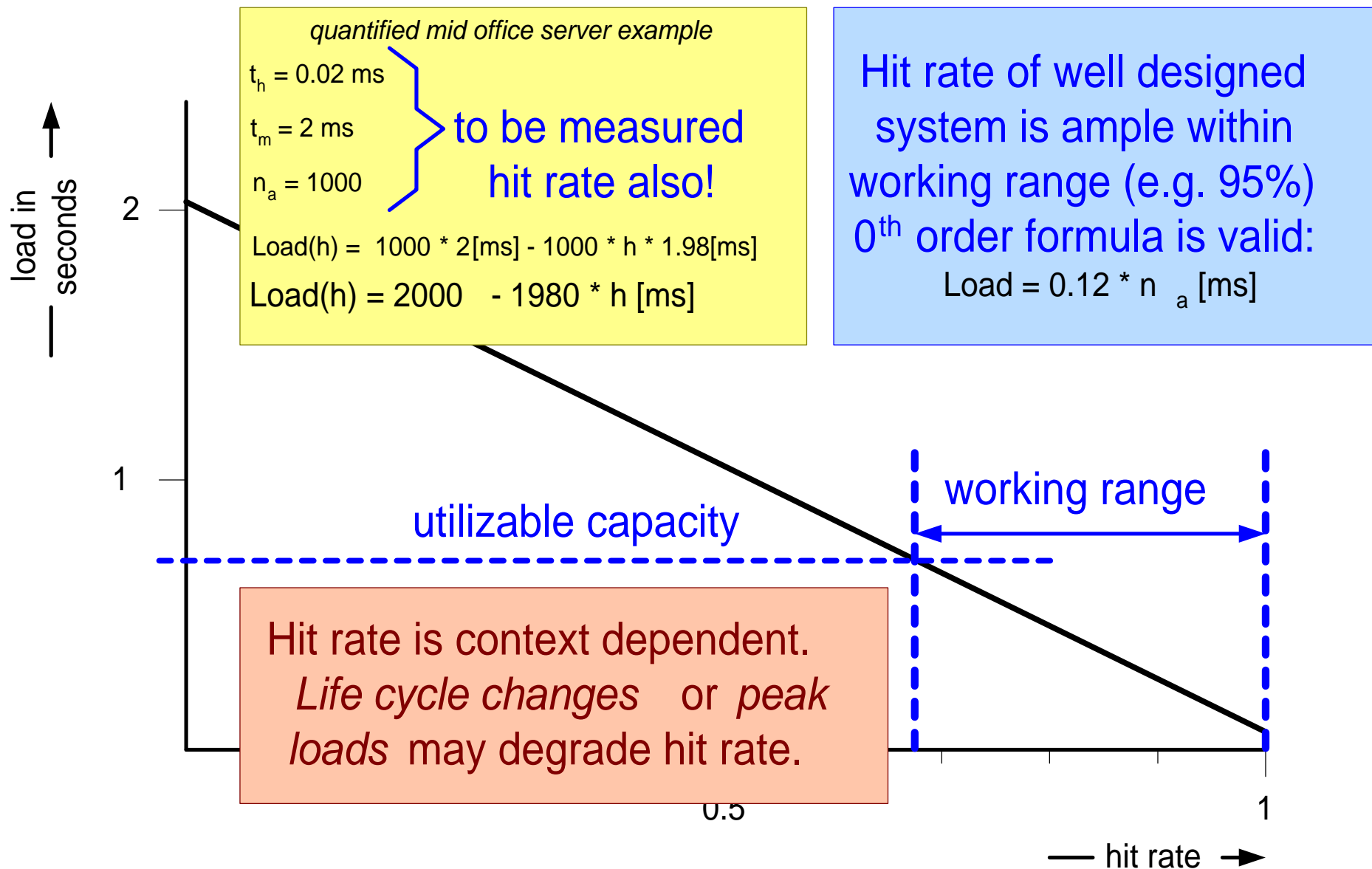
$h$  = hit rate

$$\text{Load}(h) = n_a * h * t_h + n_a * (1-h) * t_m = n_a * t_m - n_a * h * (t_m - t_h)$$

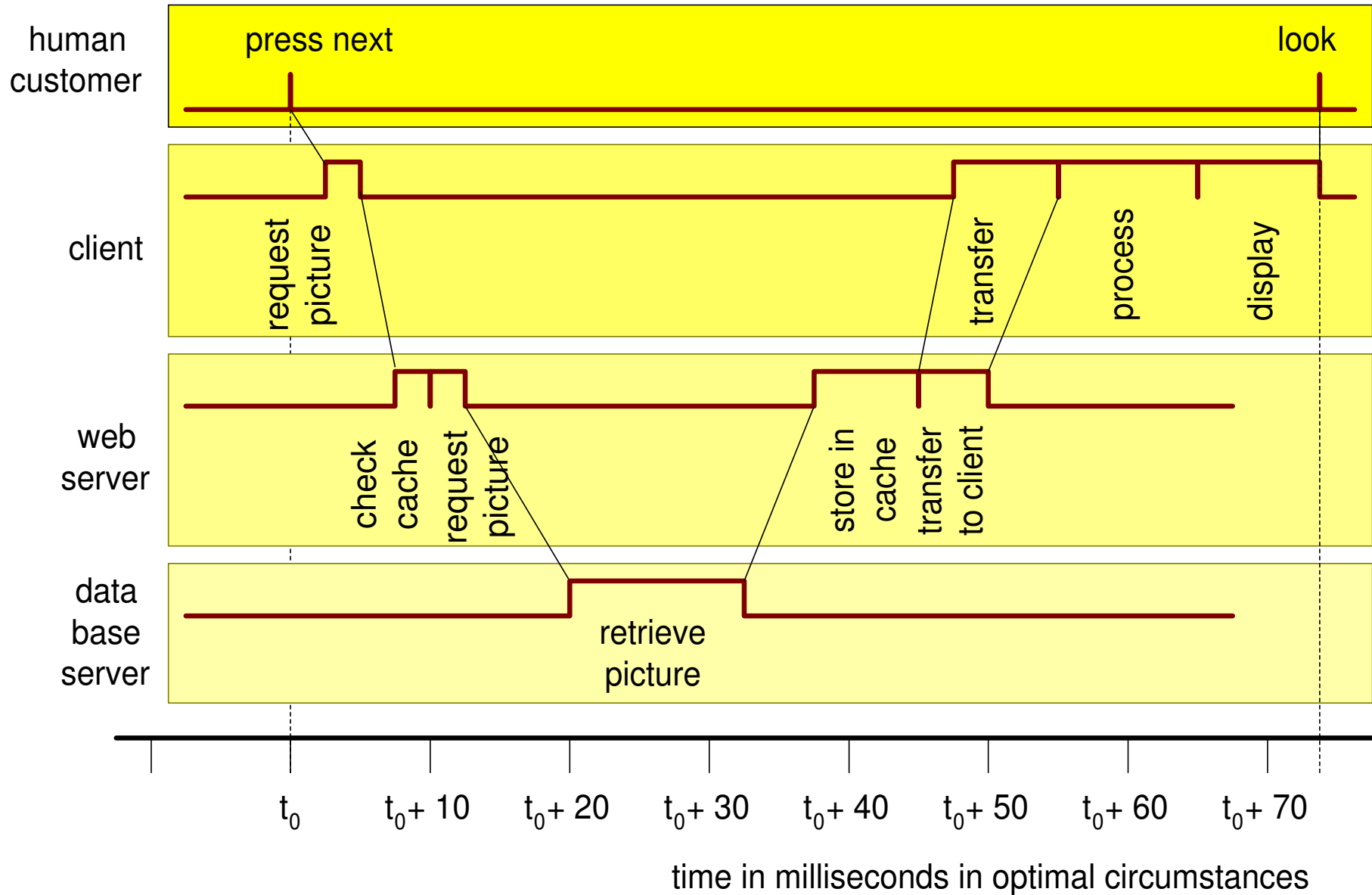
# Quantification: From Formulas to Insight



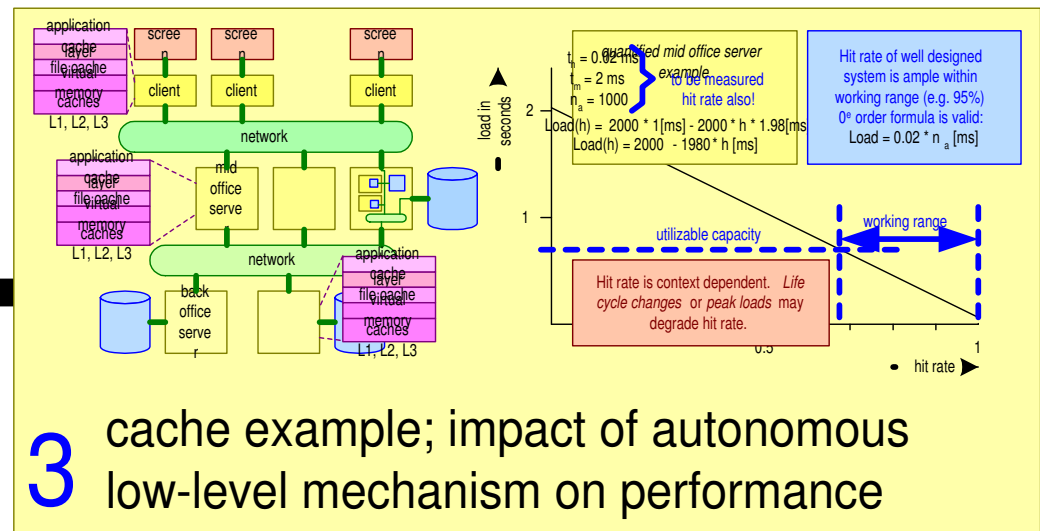
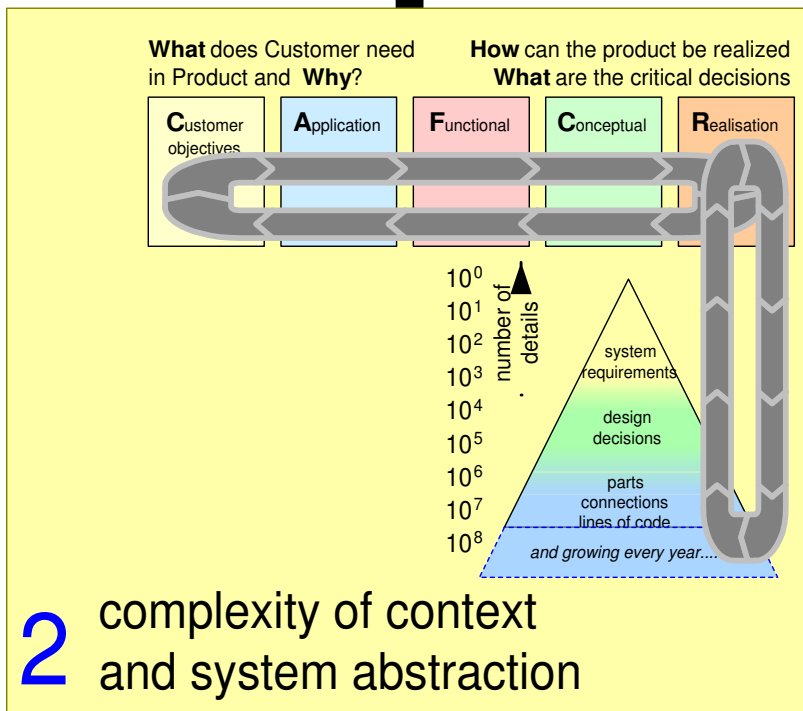
# Hit Rate Considerations



# Response Time



# 1 performance example: do we understand our design?



# 4 discussion and conclusion

# Some *Understandability* Propositions

---

central full control does not imply understandability

delegated autonomous behavior does not imply understandability

a few simple rules can create very complex behavior

understanding does not imply determinism or predictability

valid abstractions facilitate understanding

simulations provide numbers, not understanding

only humans understand!

# Conclusions

---

*control* , *predictability* , and *determinism* are **illusions**

*simple* rules can create *complex* **non-understandable**  
systems

challenge: to *model* systems at " **right** " *abstraction* level