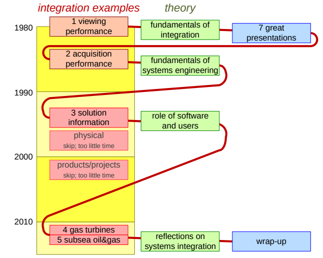
Why is Systems Integration understood so poorly? Reflections on 3 decades of unforeseen failures

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

Abstract

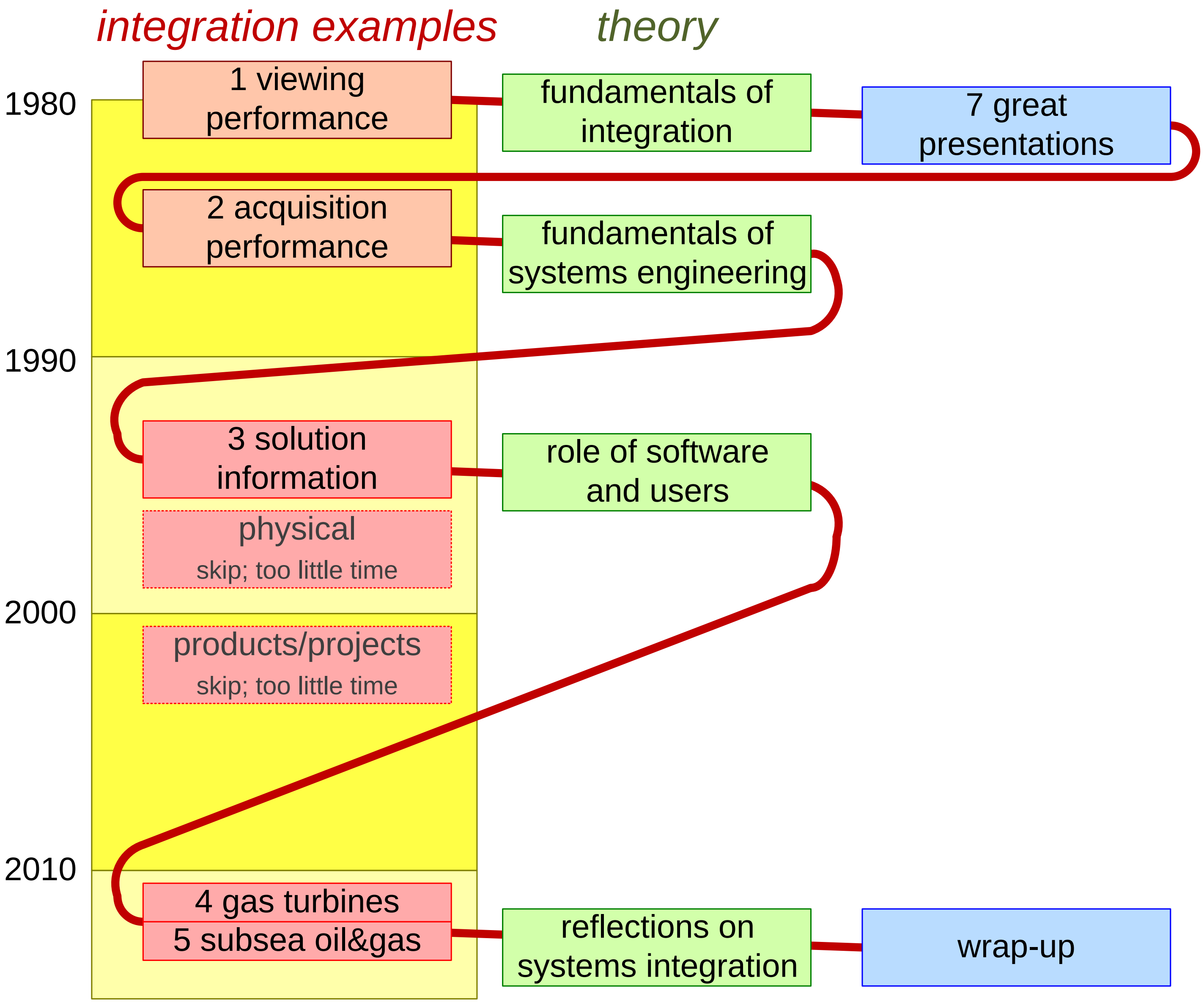
Nearly all systems developments run into problems in the late project phases, where unforeseen surprises disrupt careful planning. We will discuss a framework for systems development and integration and use a number of examples to explore what happens during systems integration. We assert that the entire project plan should be designed in reverse order, taking systems integration as driving concern.



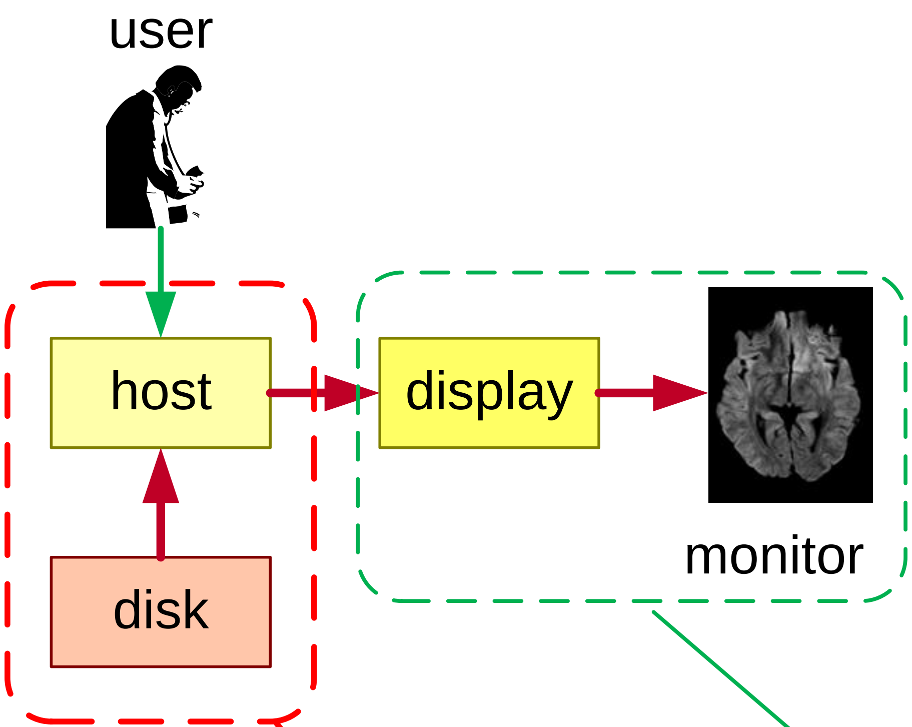
Distribution

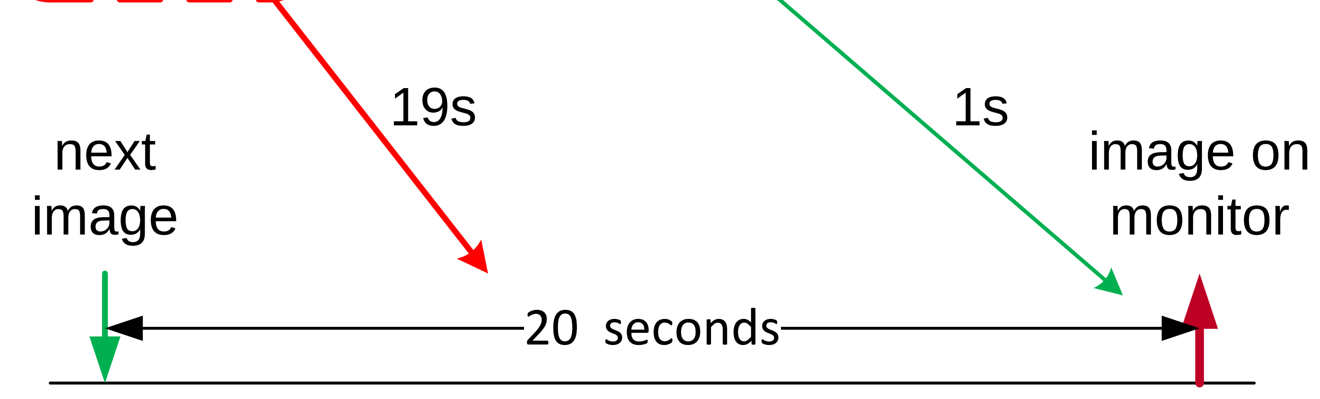
October 4, 2020
status: planned
version: 0

Figure of Contents™



Example 1: Integration of Treatment Planning System





1980, first job:
display firmware

integration drama:
image retrieval **20s**
(spec: less than **1s**)

cause:
- too much overhead
- too many layers
- too much process communication

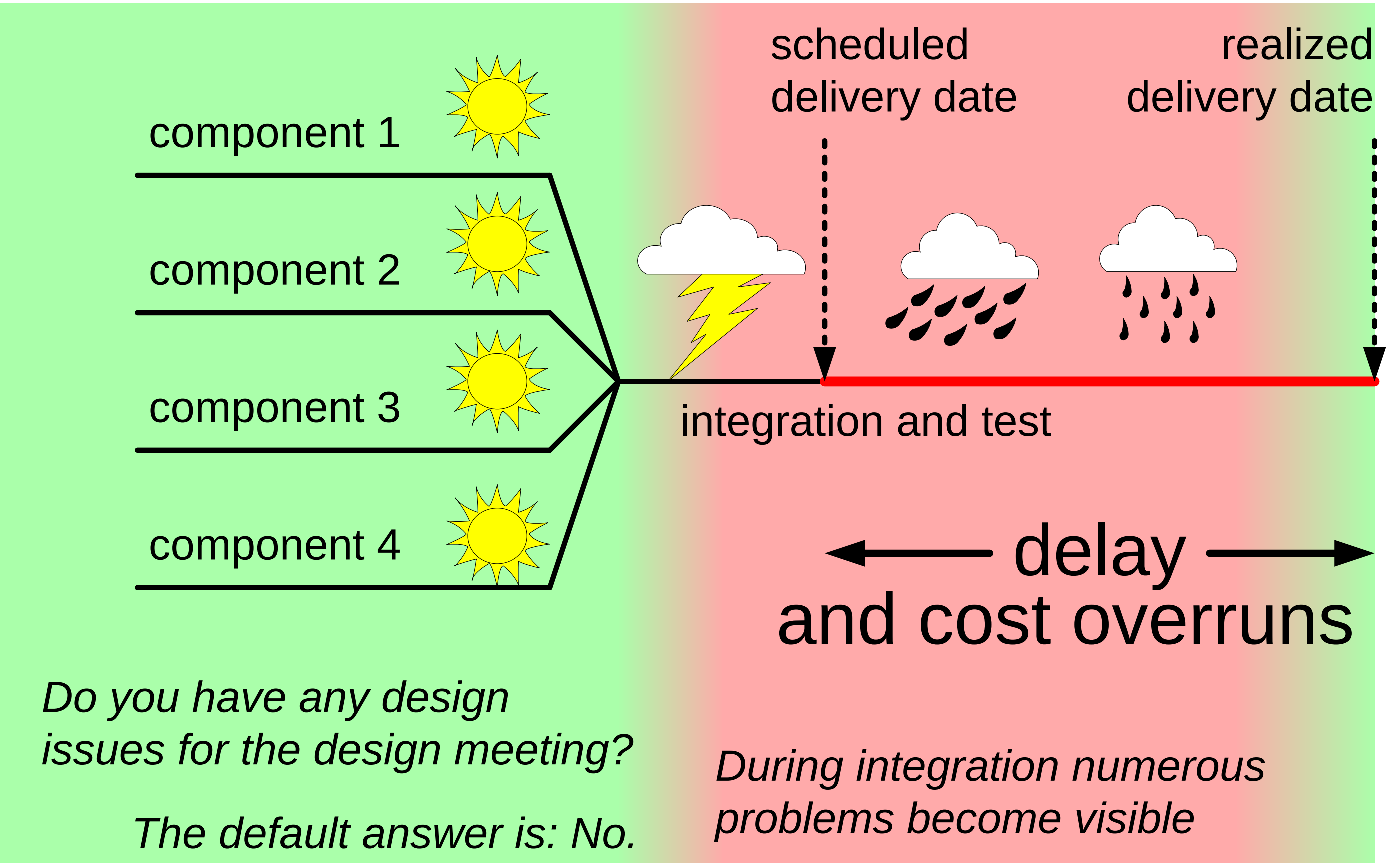
root cause:
lack of system design

Why is Systems Integration so Poorly Understood

Why do we always get delays and cost overruns during integration?

Why seems everything OK until integration?

Why do so few people understand what happens during integration?



How do you rank your project or program?

	poor	sufficient	good	very good	excellent	perfect
Outside world						
Customers						
Lifecycle support						
Specifications						
Design						
Technology						
People						
Process						

Practical Limitations

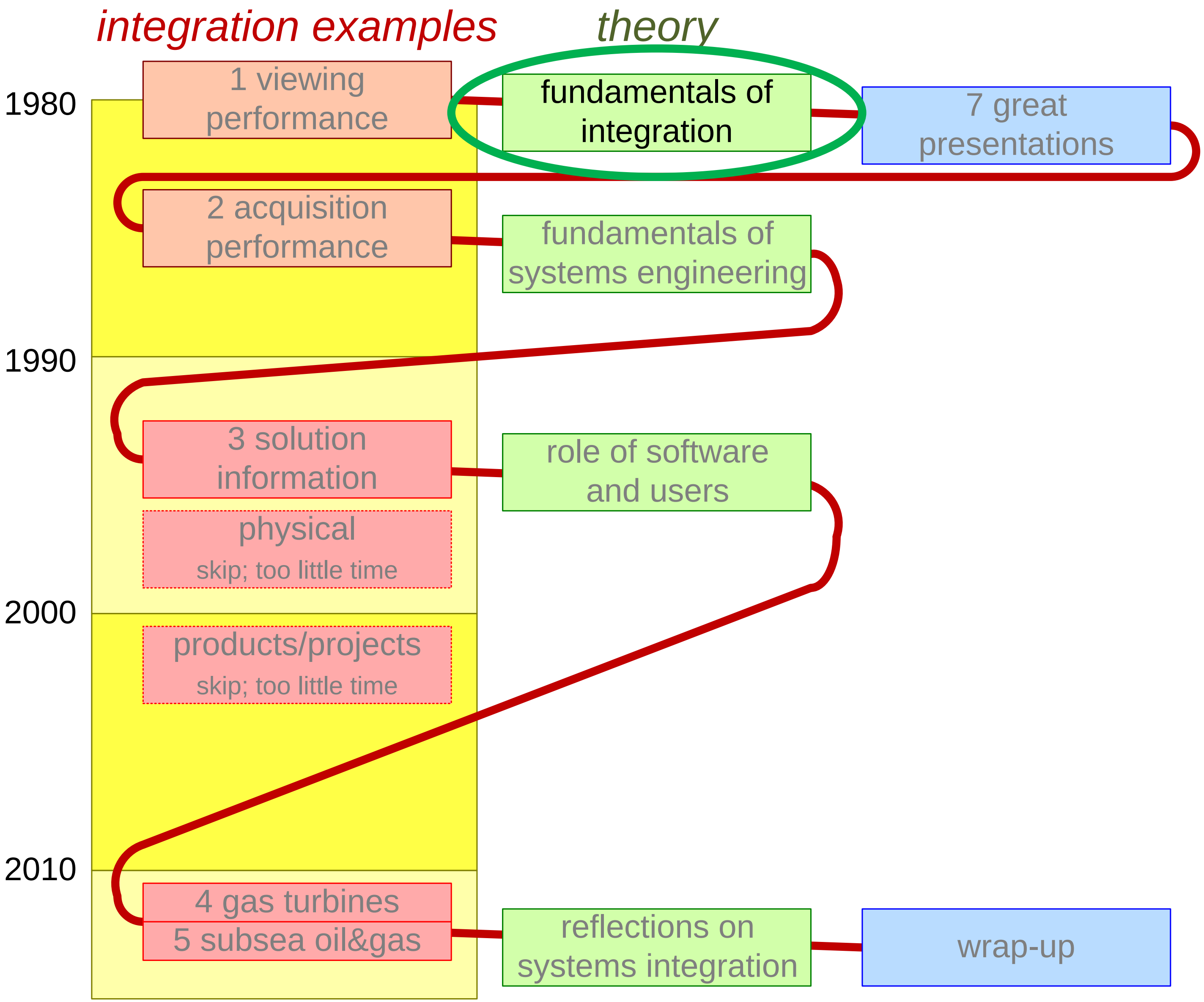
	poor	sufficient	good	very good	excellent	perfect
Outside world					X	
Customers					X	
Lifecycle support					X	
Specifications					X	
Design					X	
Technology					X	
People					X	
Process					X	

X expected answers from Kongsberg industry

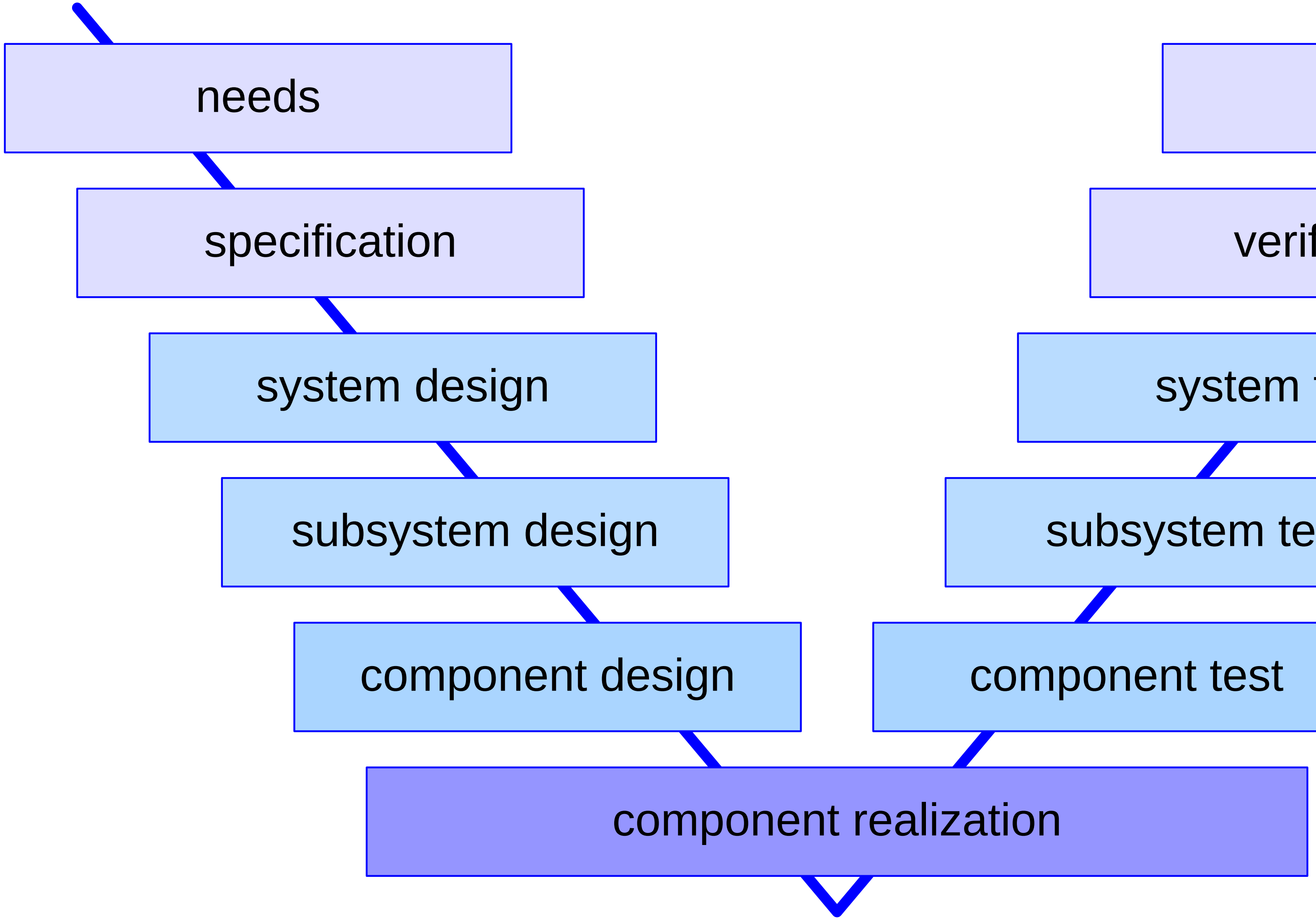
Perfect processes, people, technologies, designs, or specifications do not exist

Imperfections sometime, somewhere, will show up;
always at an inconvenient moment

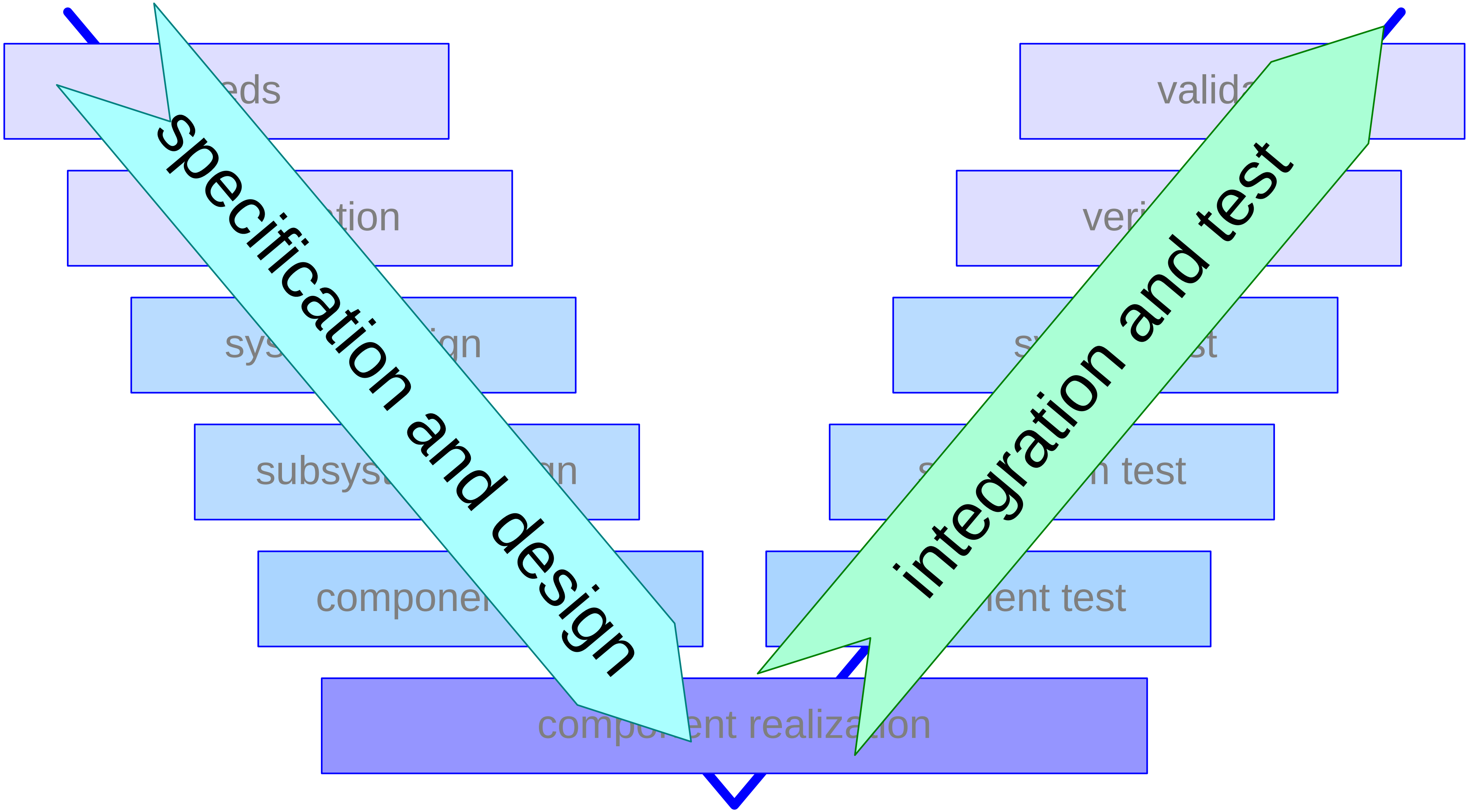
Fundamentals of Integration



V-Model

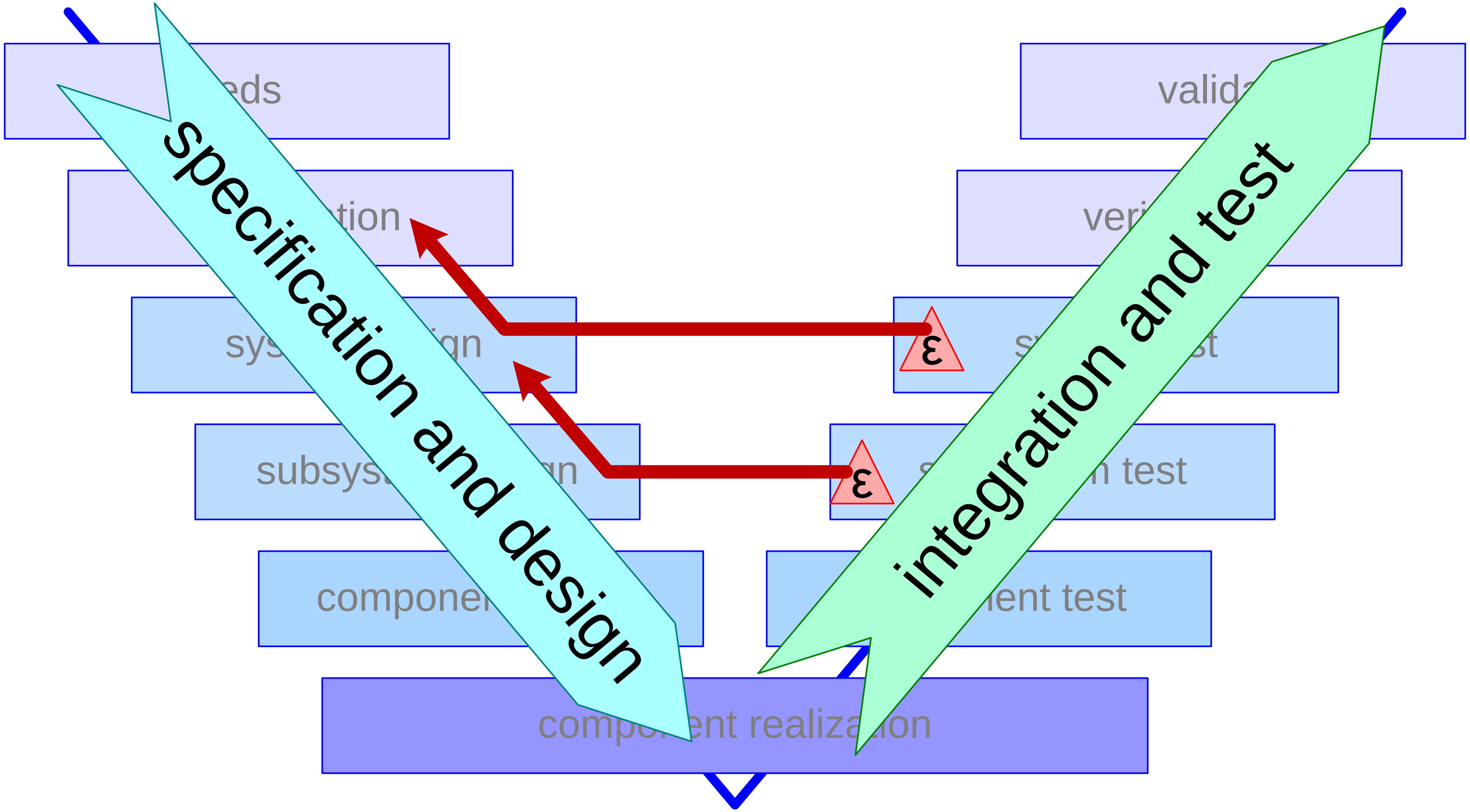

needs
specification
system design
subsystem design
component design
component realization
component test
subsystem test
system test
verification
validation

Conventional Integration View

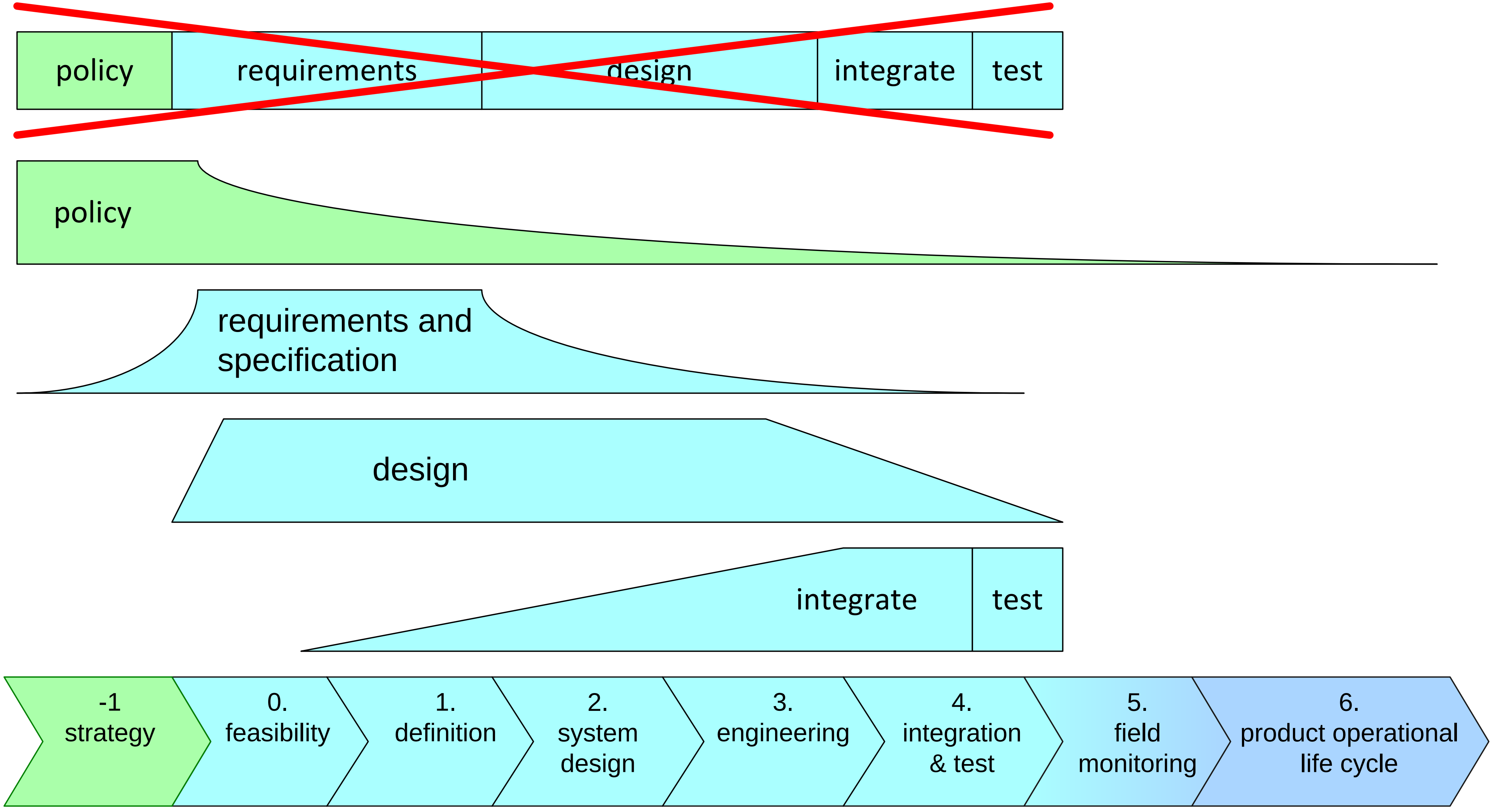


Limitations in Front-End Cause Failures

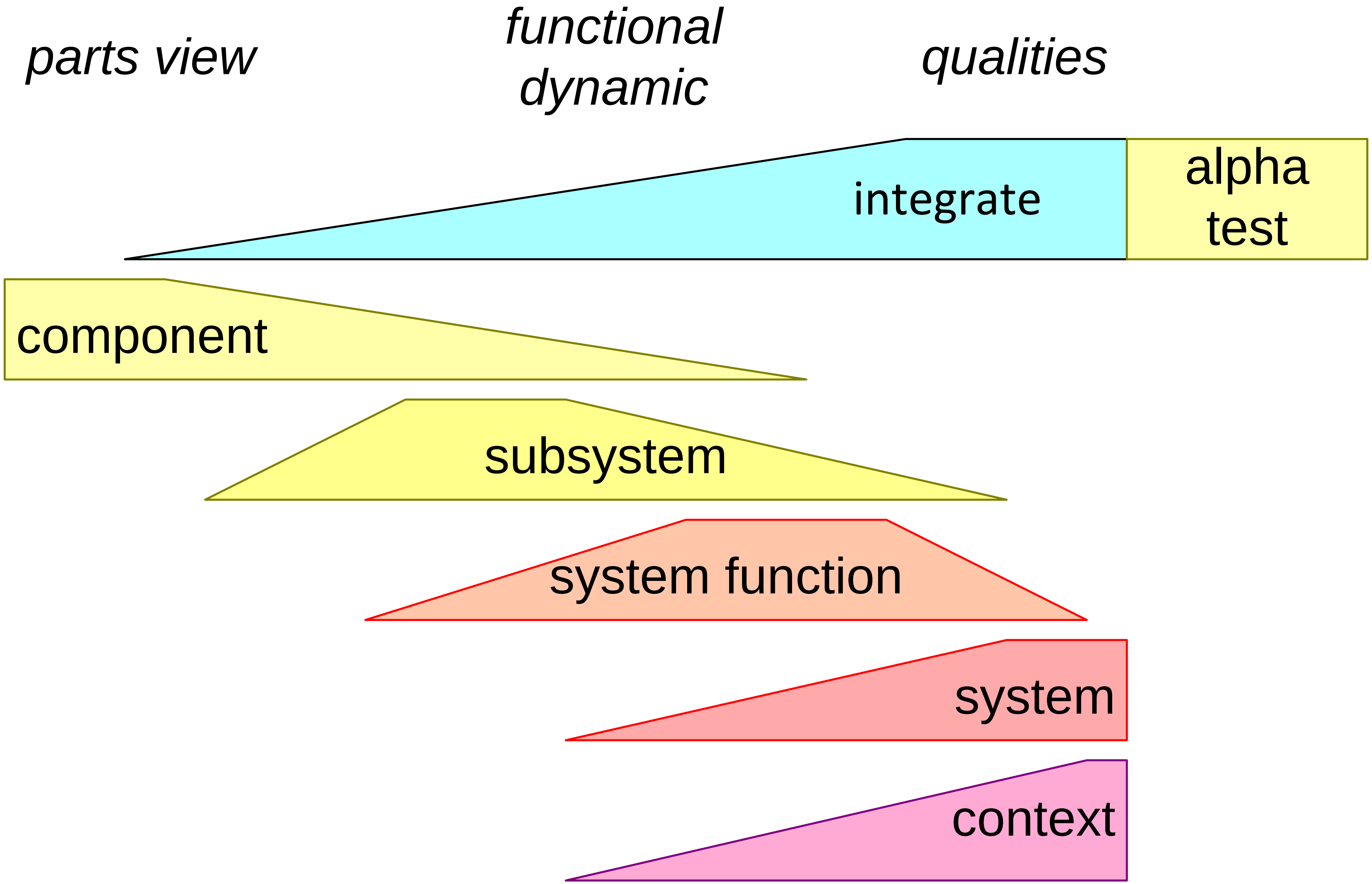
failures found during integration can be traced back to *unknowns*, *unforeseens*, and *wrong assumptions*



Typical Concurrent Product Creation Process



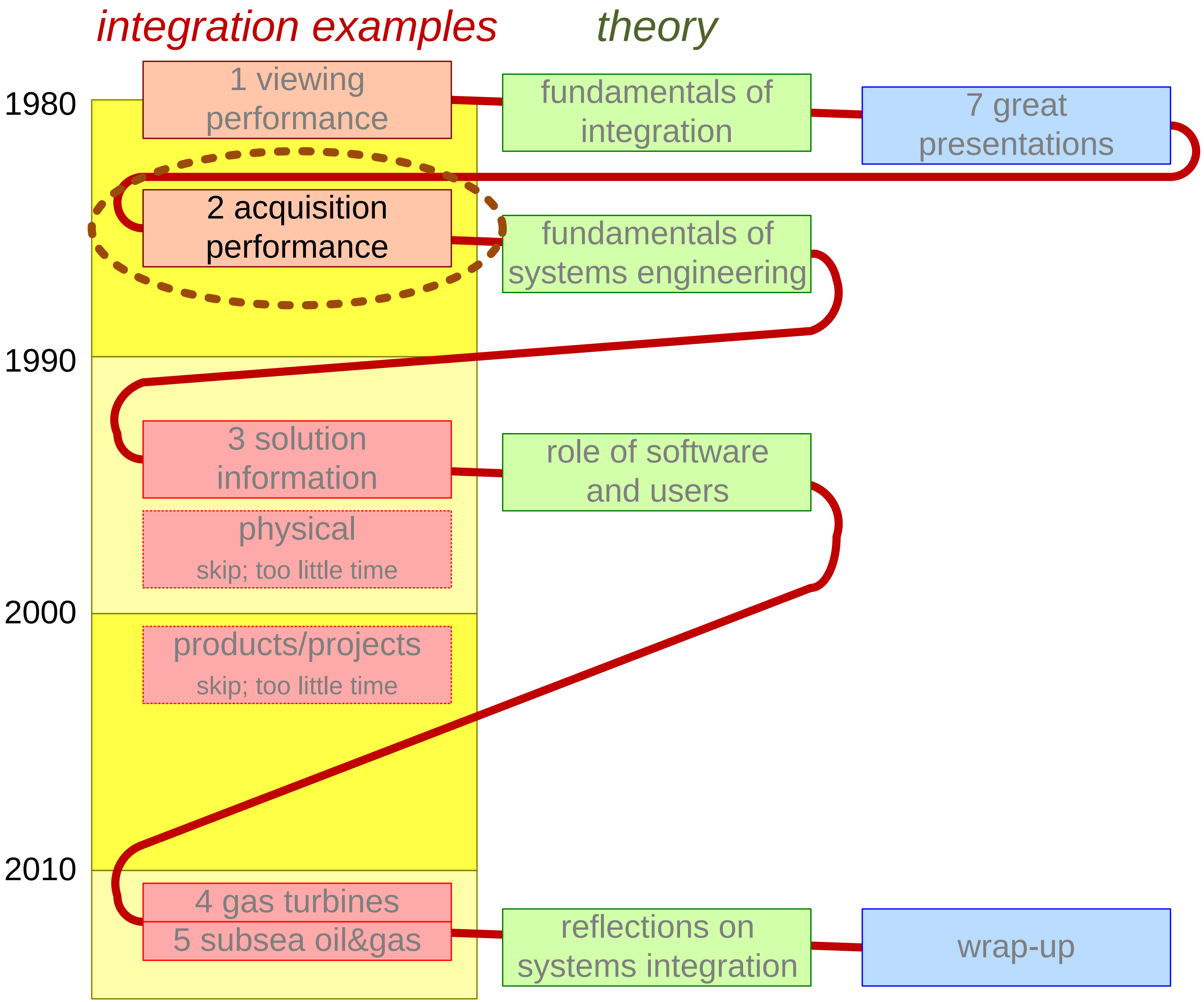
Integration Takes Place in a Bottom-up Fashion



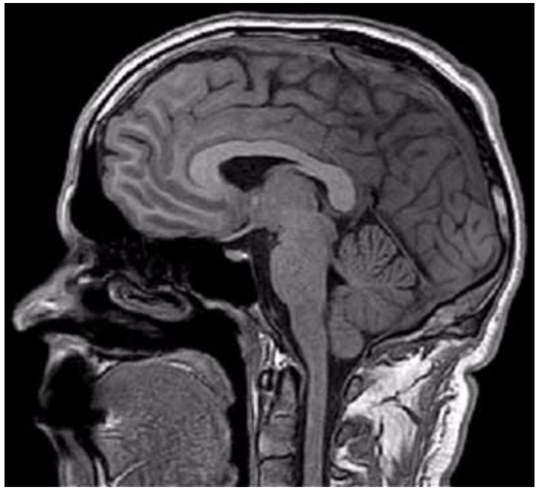
Fill in this form during KSEE 2013!

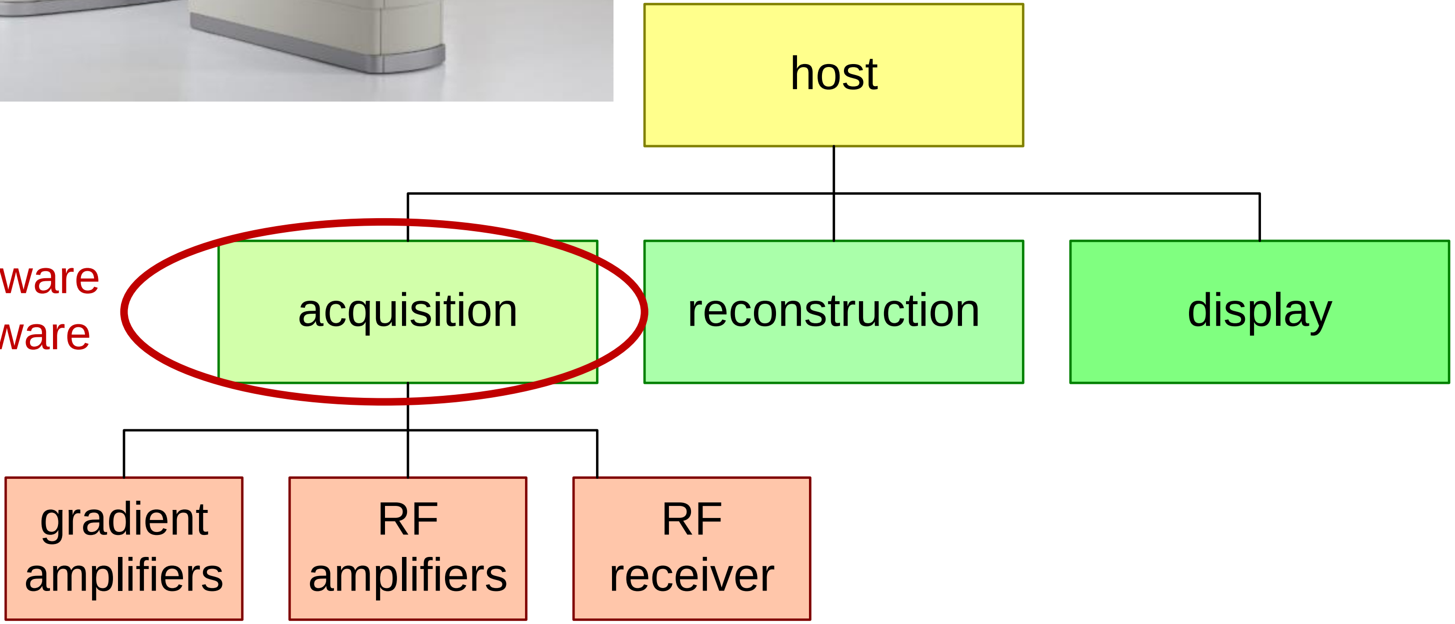
KSEE 2013 work form	*Current Status* What type of failures pop-up during your Integration?	*Potential Improvements* How could these failures be found earlier? What means or strategies can you employ to find them earlier?
Niels Braspenning System Integration at ASML: Linking Technical Content, Test Configurations, Timing... And People!		
Alejandro Salado Validation risks of using development methodologies in a hierarchical fashion - When contracts meet architecture ownership		
Andreas Thorvaldsen Changing A System From Within – And Get Hit By The Unexpected Surprises		
Benoît Le Bihan Laggan Tormore Project System Test: when new Subsea Solutions For Harsh Environment Meet Reality		
Jim Armstrong Systems Integration: What Are We Waiting For?		
Terje Jensvik A software centric approach to Electronic Systems Engineering.		
Eldar Tranøy Early phase need analysis – Can we ease systems integration?		
Gerrit Muller Why is Systems Integration understood so poorly? Reflections on 3 decades of unforeseen failures		

Example 2: Performance Again



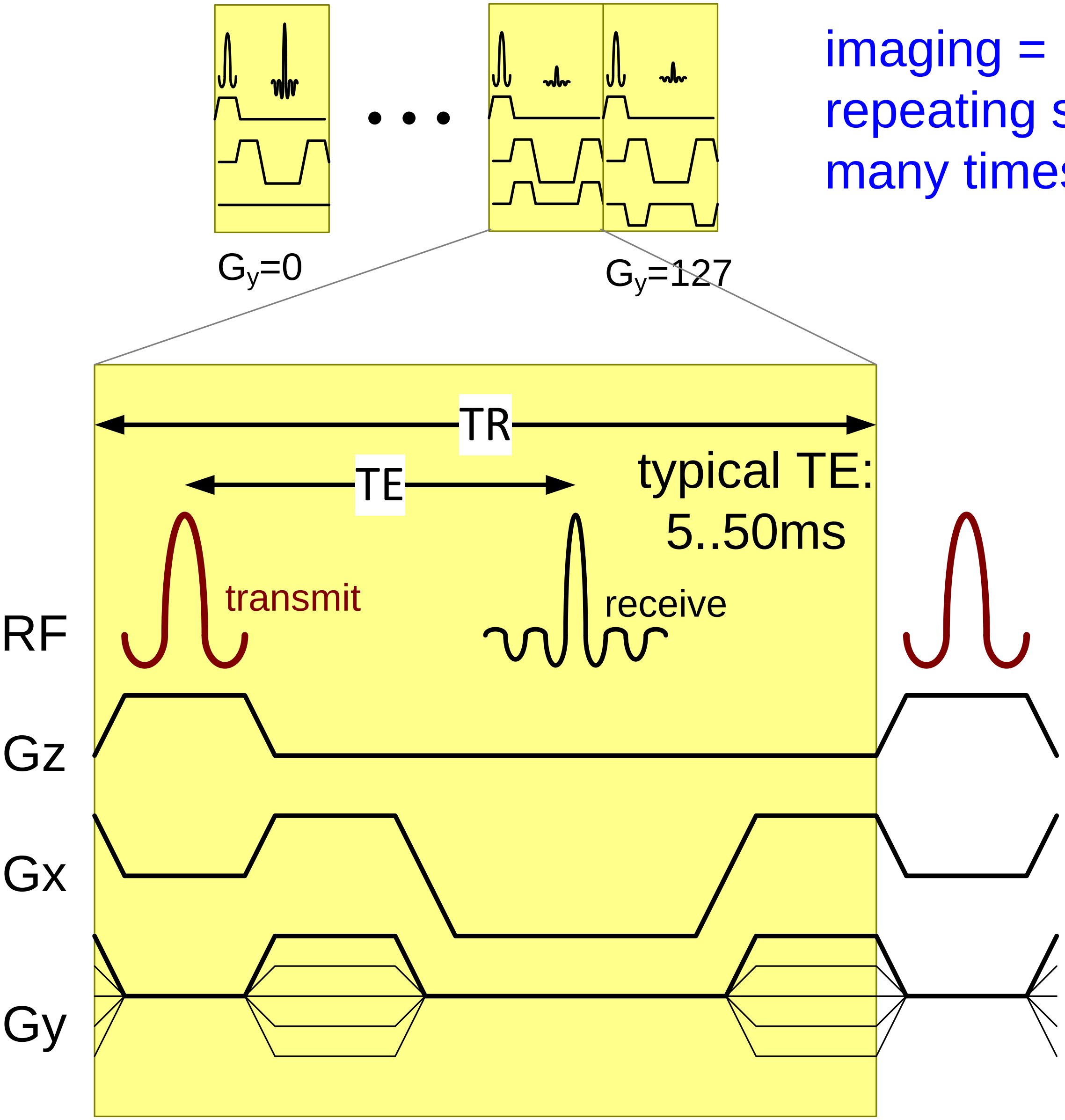
Example 2: Integration of MRI Acquisition Subsystem





new hardware
new software

Repetition Time MRI



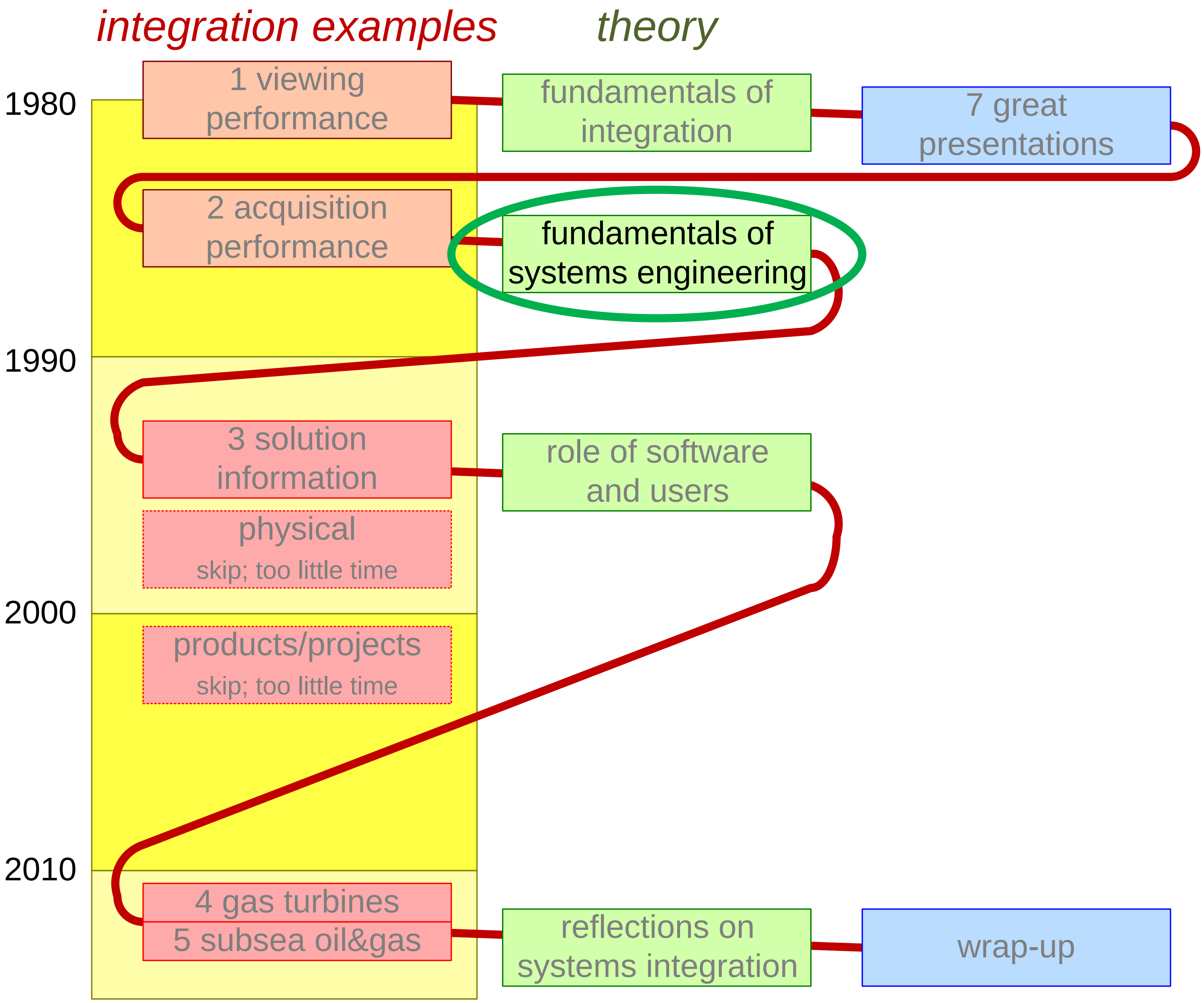
imaging =
repeating similar pattern
many times

problem:
TR > 1 s.
spec less than 10 ms
more than factor 100 off!

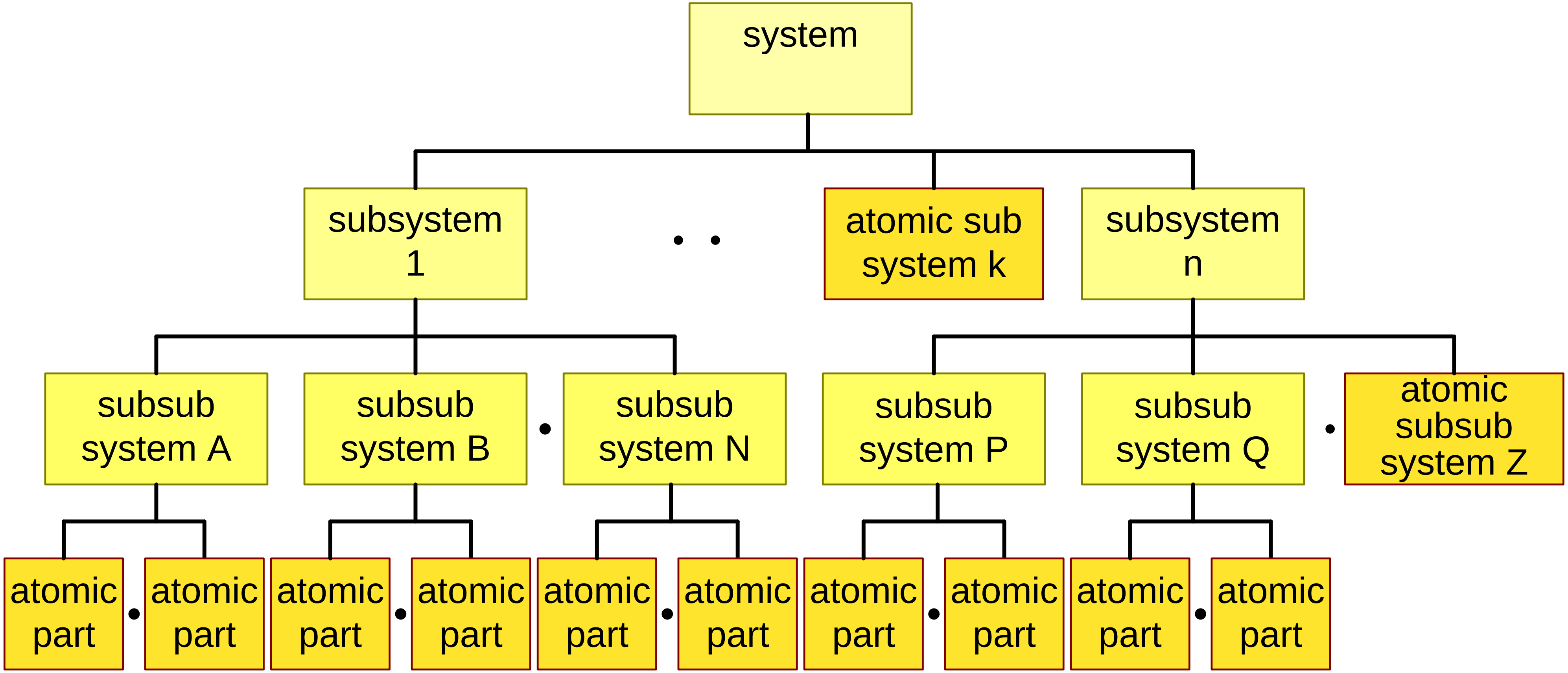
causes:
floating point arithmetic
too many layers

root cause:
functionality focus

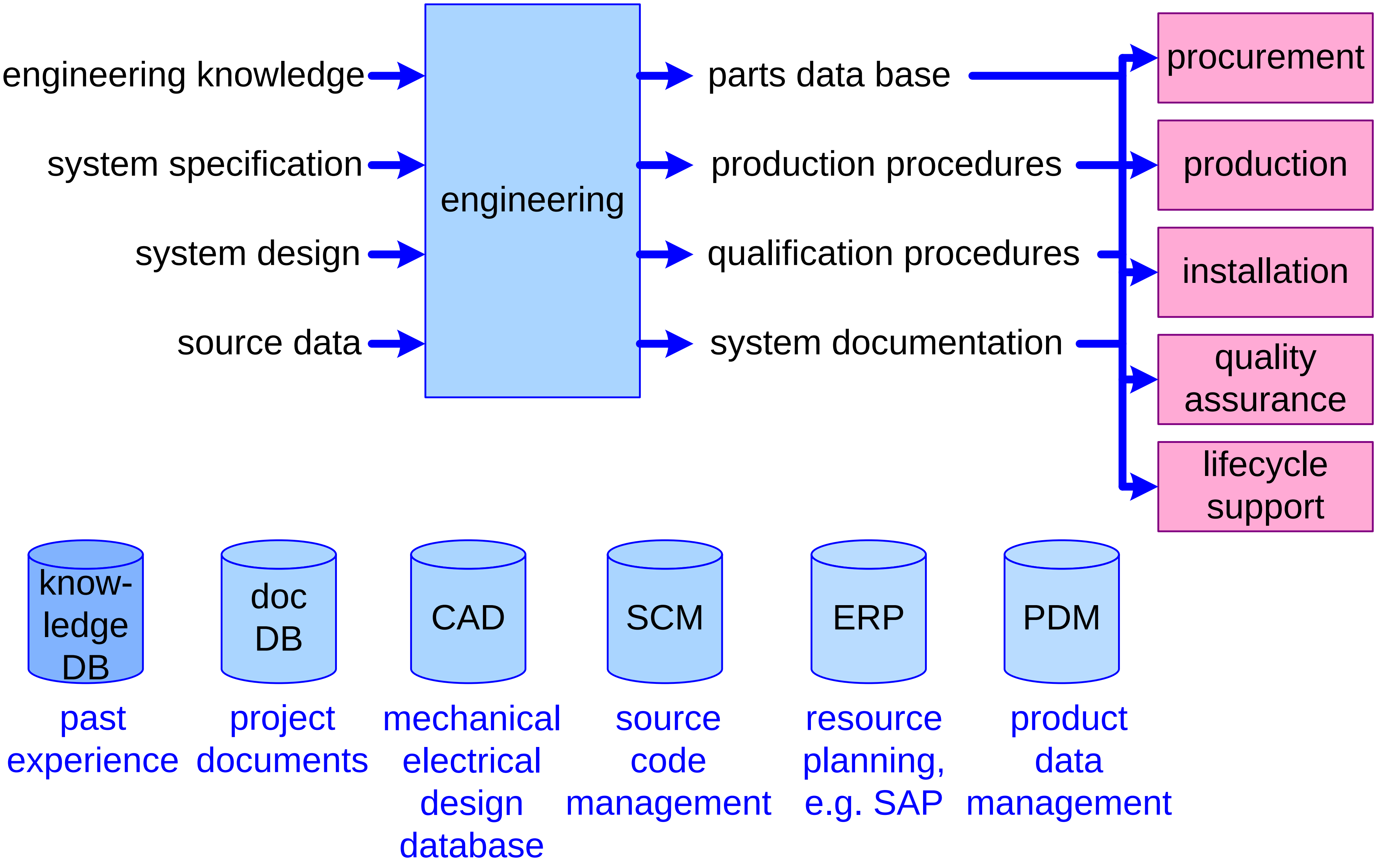
Fundamentals of Systems Engineering



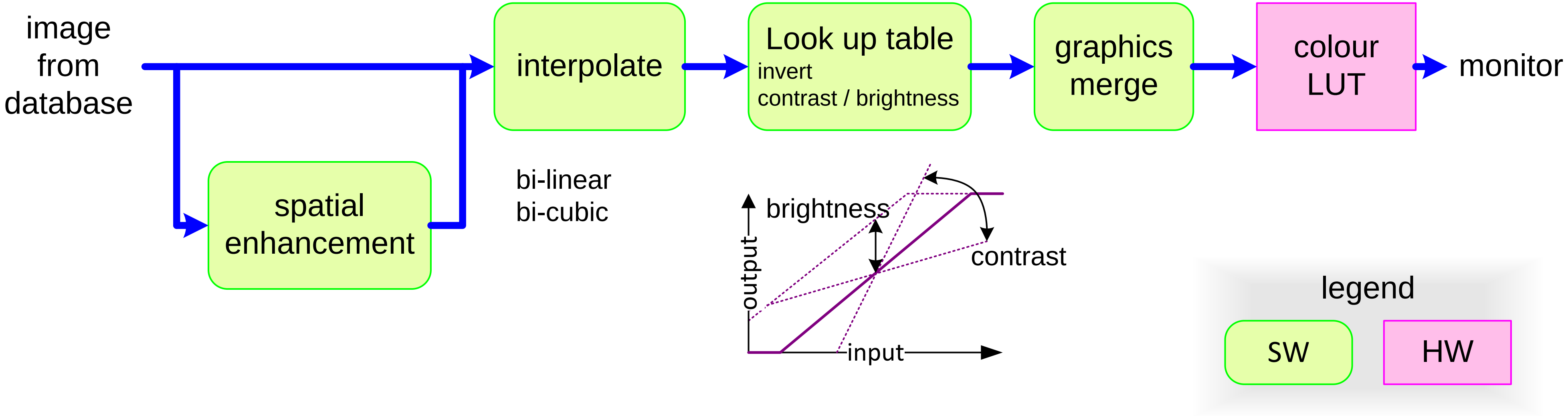
SE Rule 1: Partition and Define Interfaces



99% of Organization has a “Parts” Focus

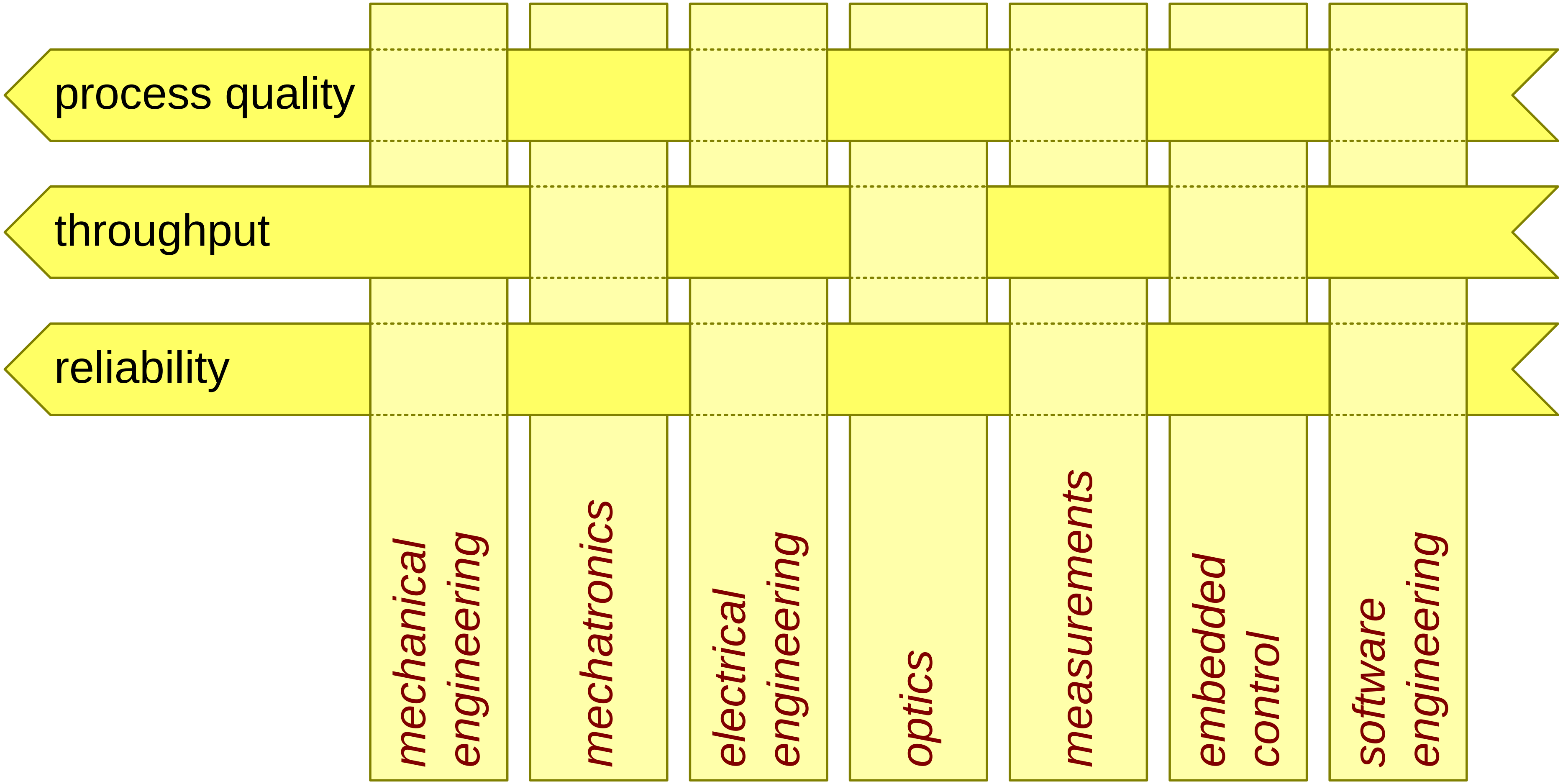


10%? Understands Dynamic Behavior or Functionality



Few Understand Key Performance Parameters

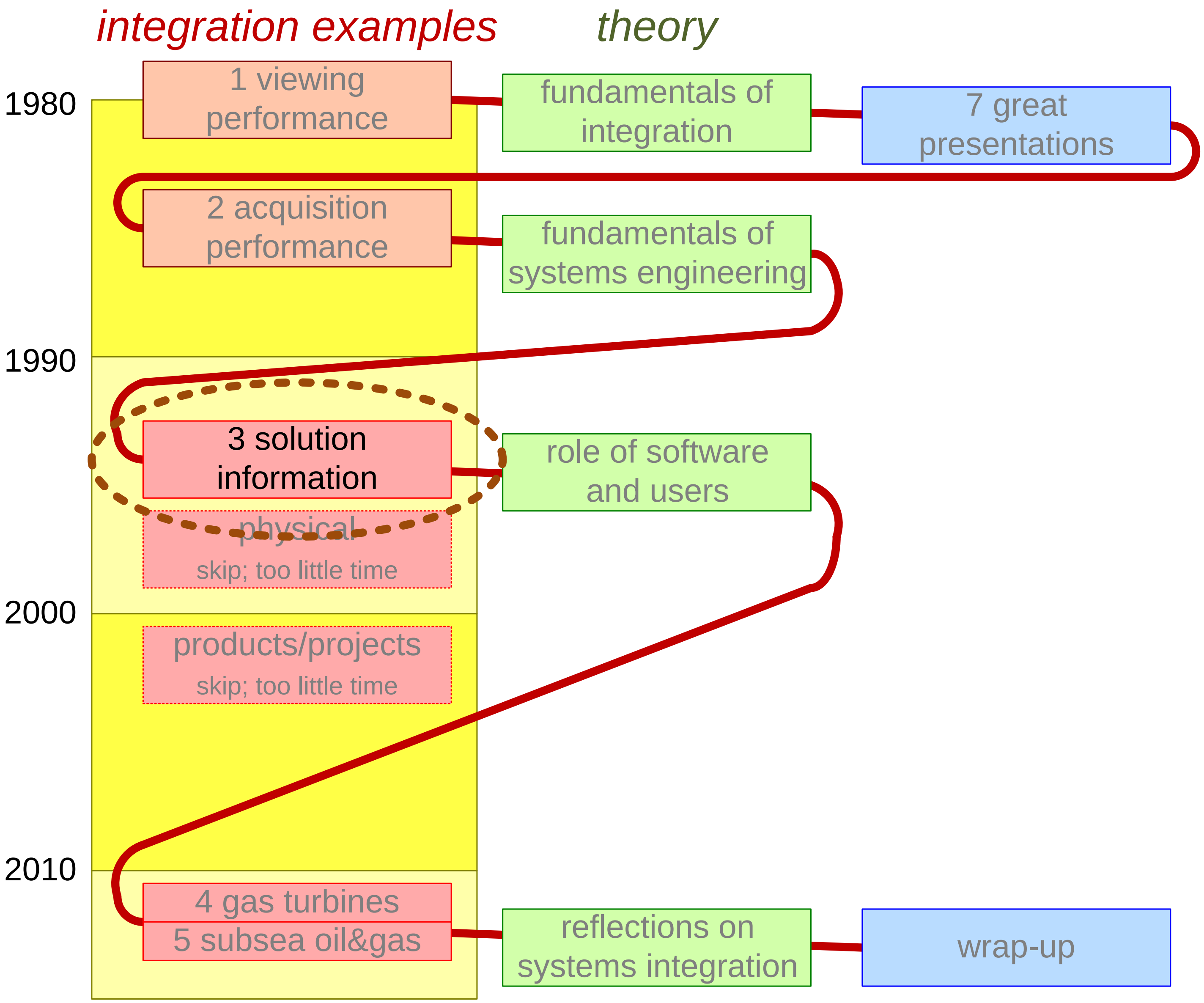
Systems Engineering: responsible for customer key drivers and key performance parameters of system



Typical Order of Integration Problems

1. The (sub)system does not build.
2. The (sub)system does not function.
3. Interface errors.
4. The (sub)system is too slow.
5. Problems with the main performance parameter, such as image quality.
6. The (sub)system is not reliable.

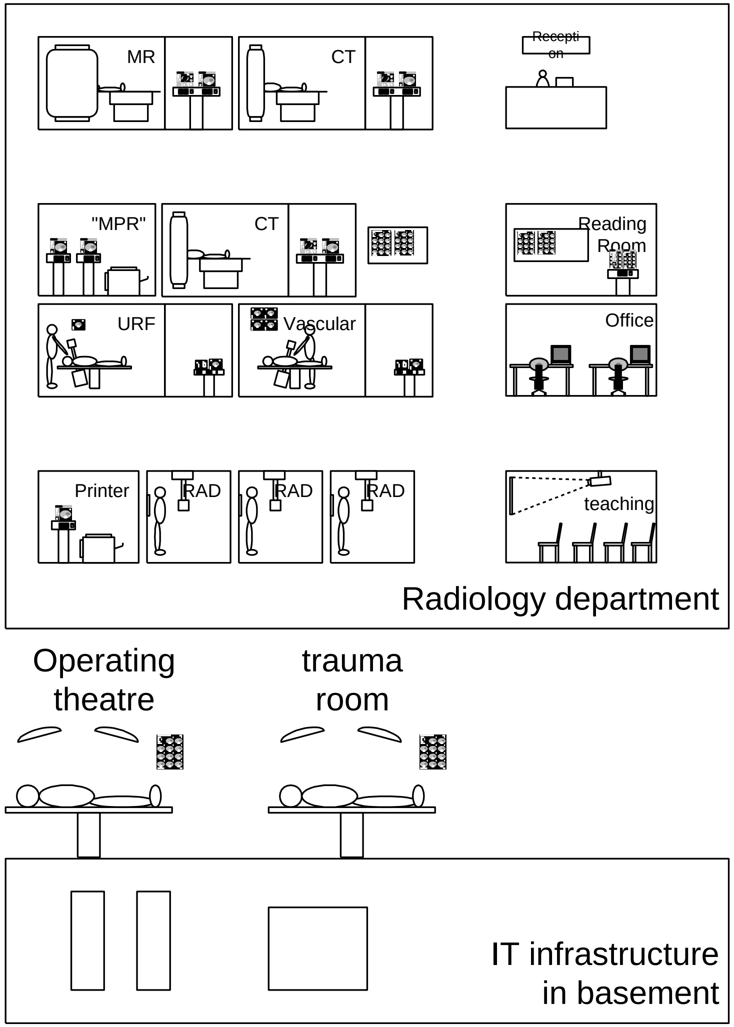
Solutions: Integration of Multiple Products

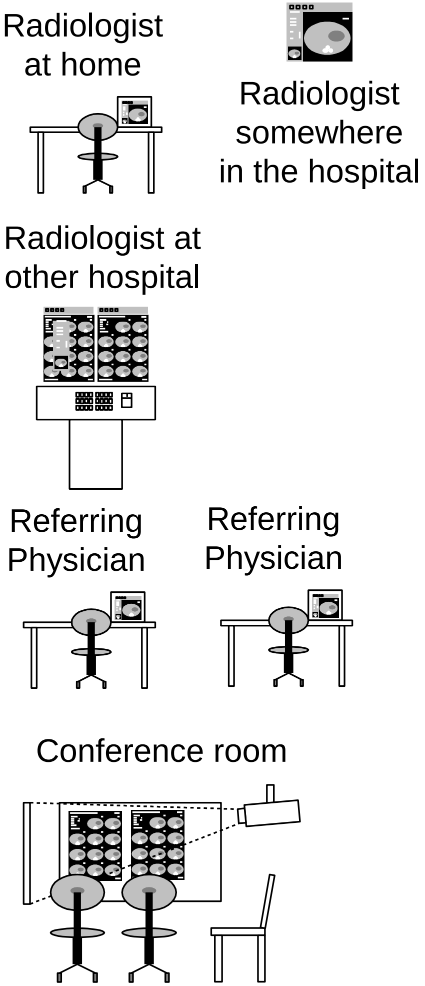


Example 3: Integrated ClinicalSolutions

Integrated Clinical Solutions:
integrate stand-alone products
to offer clinical integrated functionality

Note the similarity with Kongsberg Maritime’s achievements with K-master and operator stations

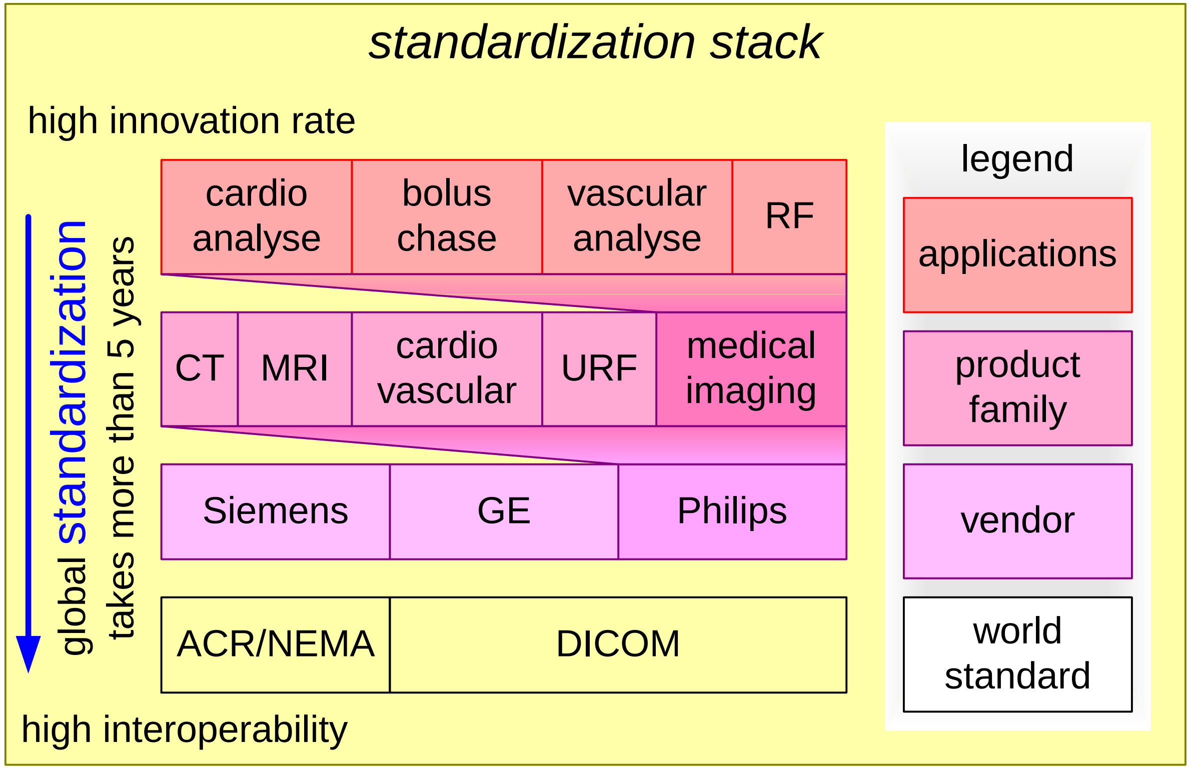




The Information Model Swamp

Every application, release, product, product family, and vendor has its particular interpretation of information, despite standardization.

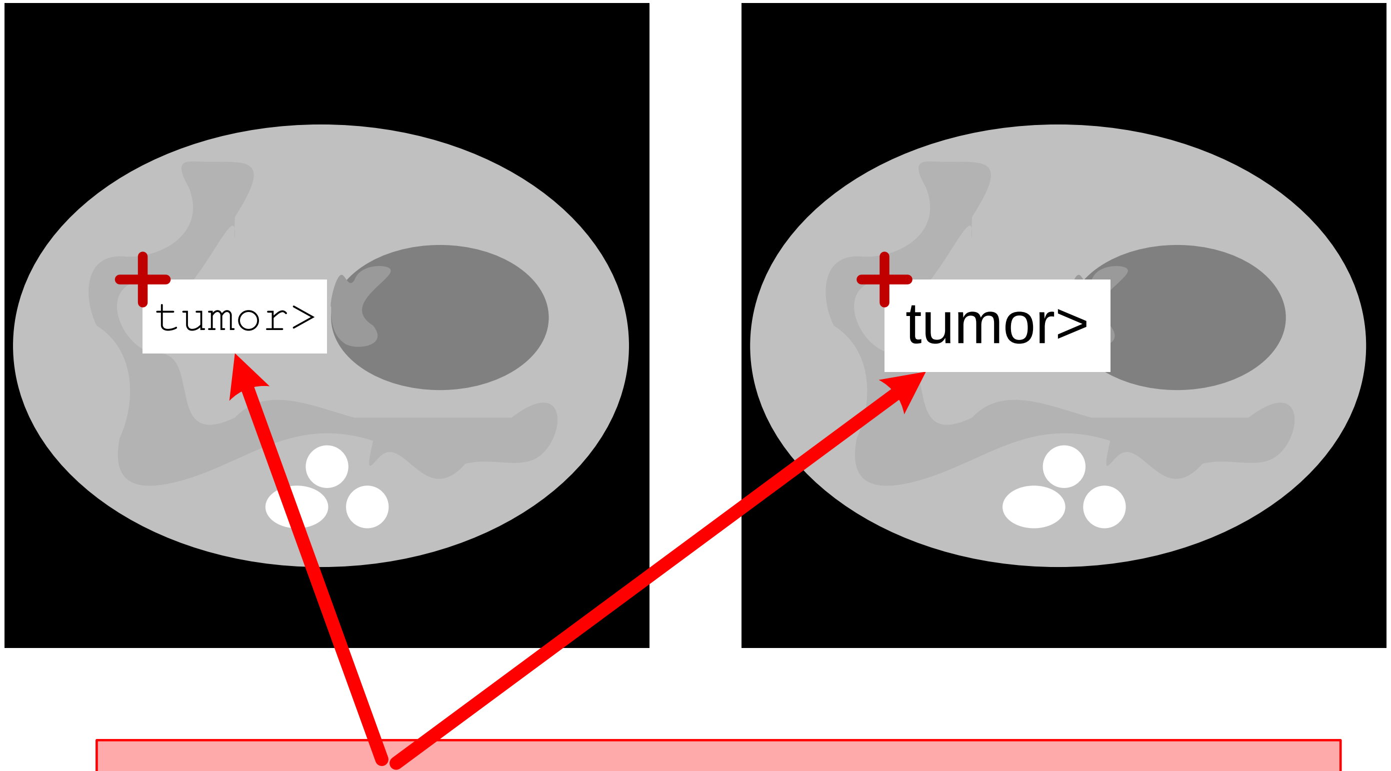
Convertors, wrappers, and adapters are nearly everywhere.
The cynical name of our product was *Shit Concentrator*
since the integrating product has to resolve any inconsistency



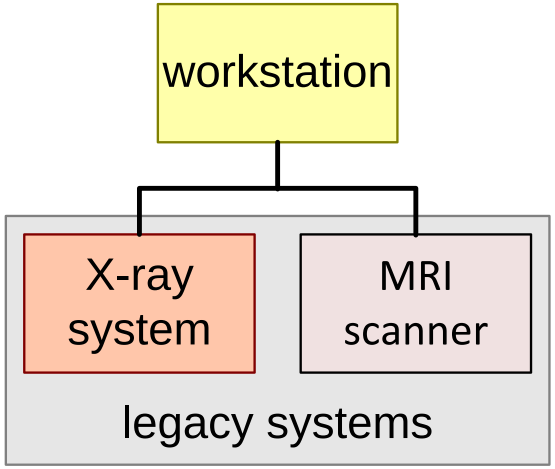
Risks of “Near Identical” Data Models

URF monitor output:
fixed size letters at fixed grid

Workstation

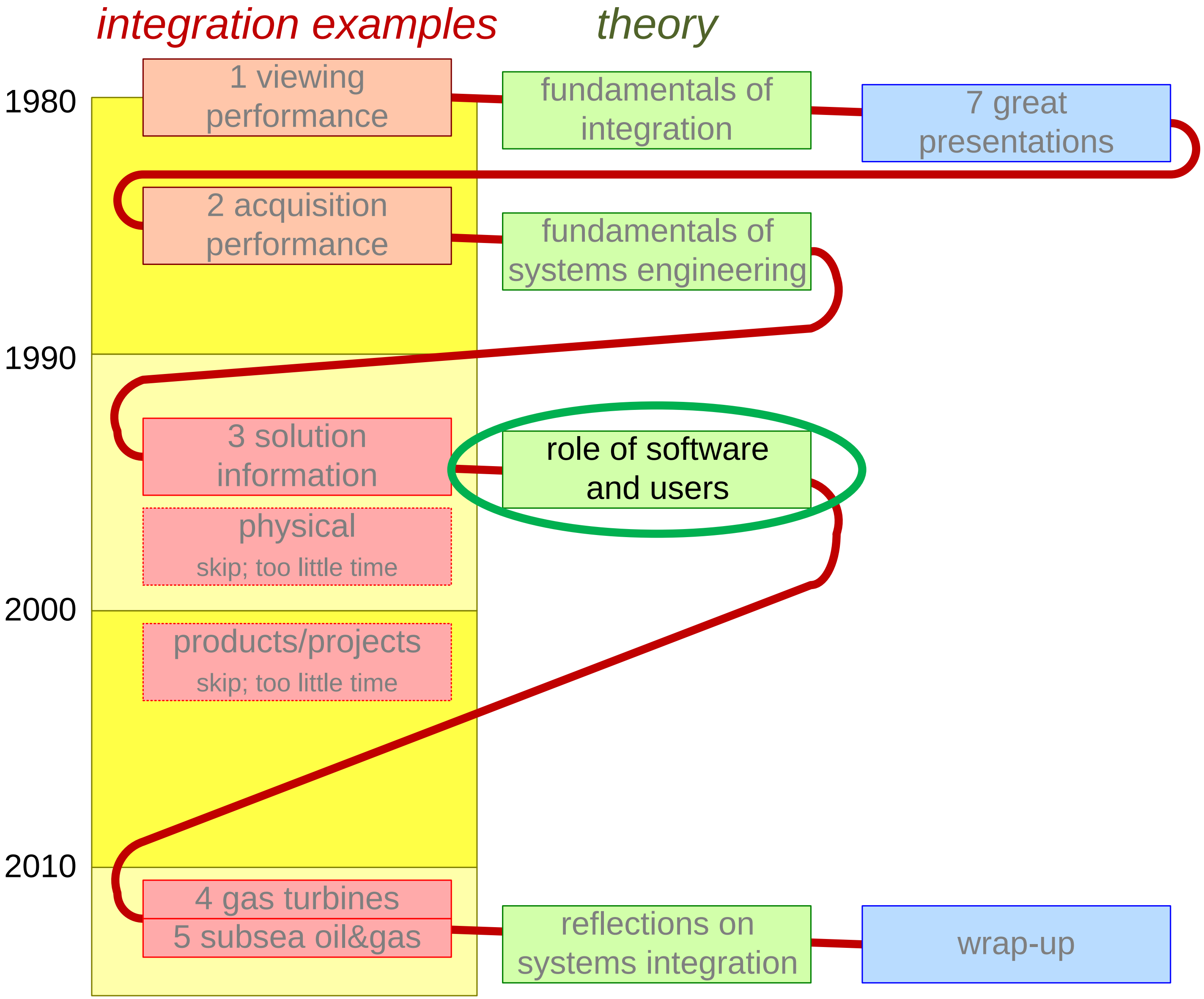


other rendering causing a dangerous mismatch between text and image



multiple near-identical data models with near-identical interpretations

Role of Software

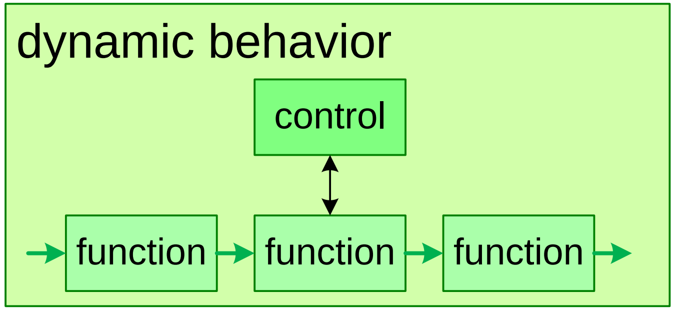


Software Characteristics and Role

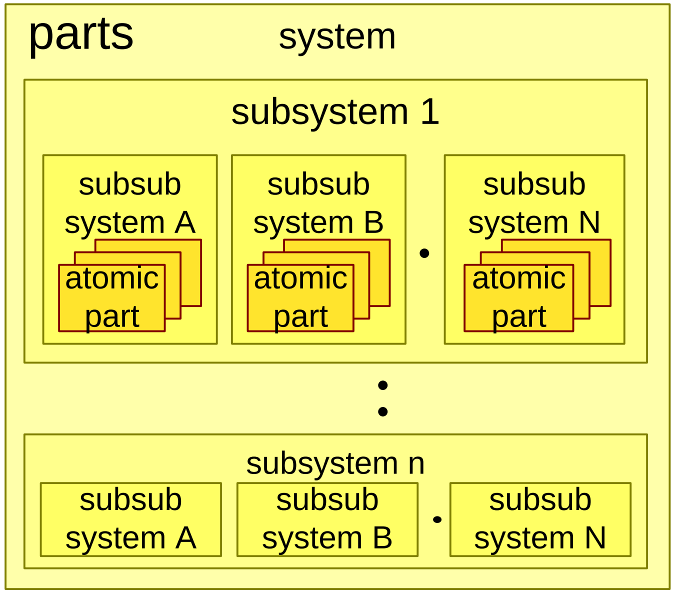
quantified properties

productivity: 100.000 images per hour
speed: 100 frames/second
max latency: 50ms
max down time: 4 hrs/year

SW
determines and limits properties



SW
- defines functionality and dynamic behavior
- captures applications
- conducts all technologies

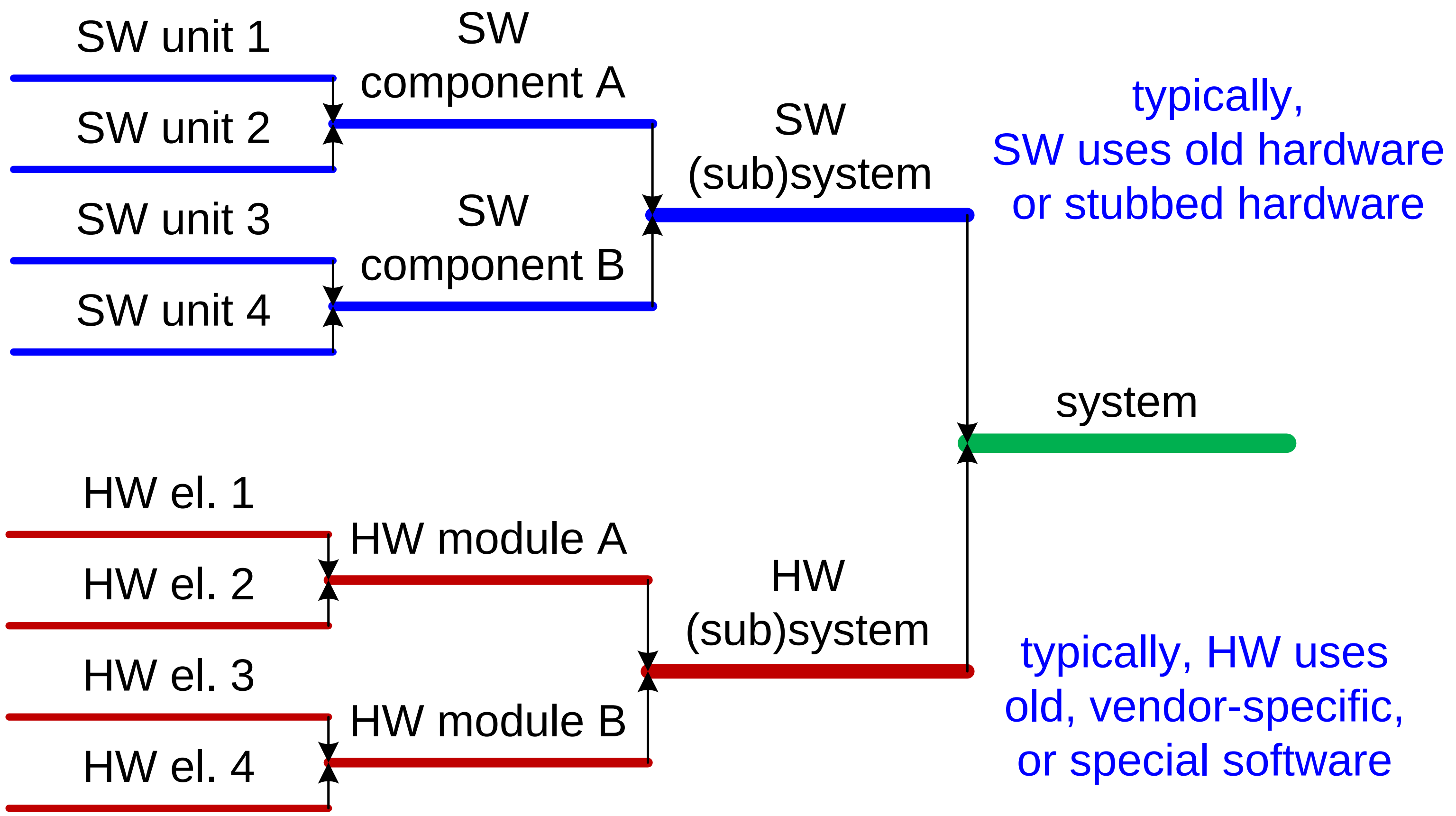


SW has its own partitioning
in e.g. components, units

SW
- has zero delivery time
- production is costless
- is ideal to solve last minute problems

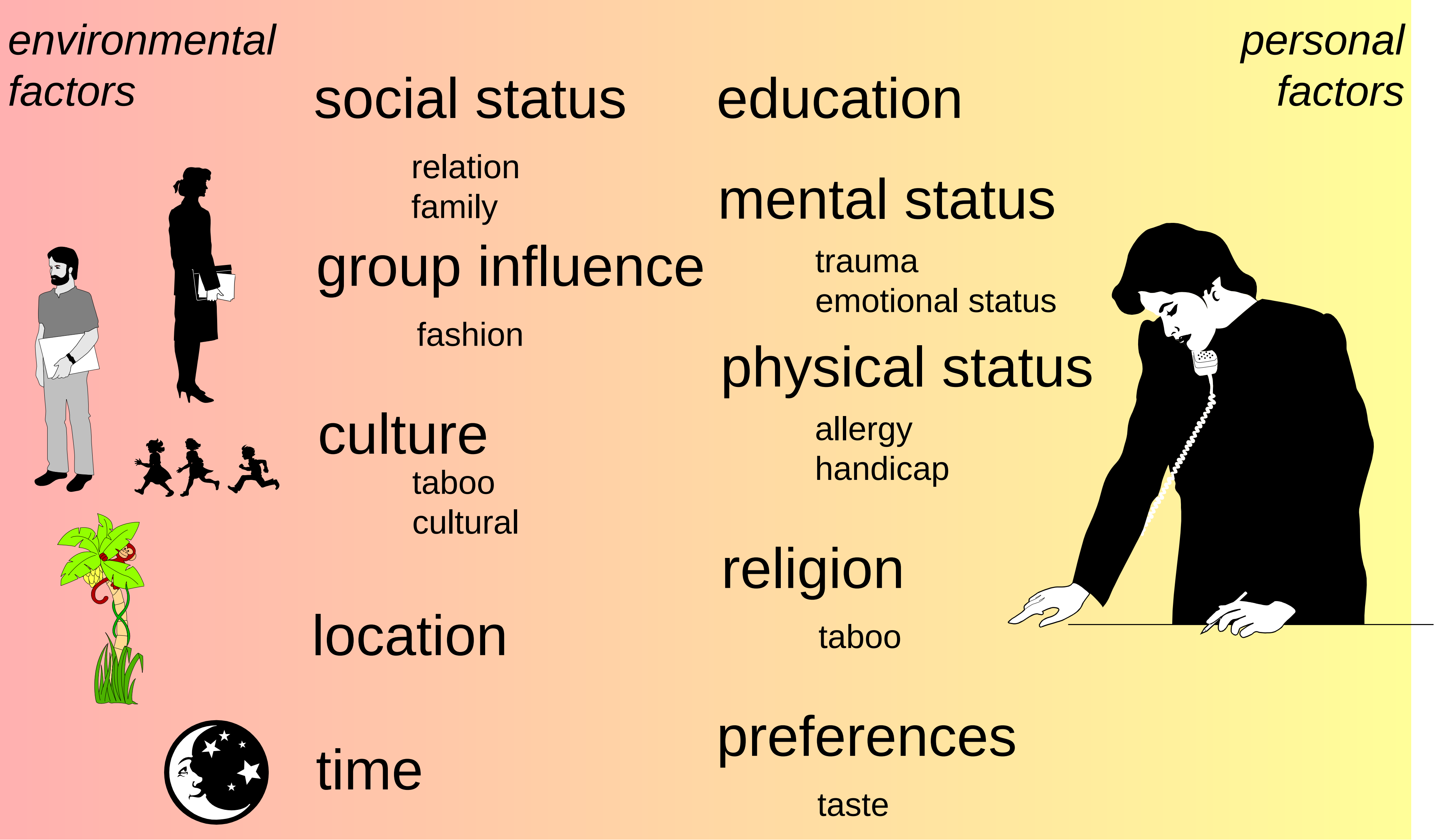
SW
- is abstract and intangible
- is alien to “physical” engineers

Hardware and Software Typically Meet at the End



Segregation of hardware and software is a typical organizational problem. Such segregation ignores close coupling of hardware and software.	Erroneous assumptions about hardware are discovered late. Key performance parameters are visible late.

User Behavior is a.o. Determined by



environmental factors

personal factors

social status

- relation
- family

group influence

- fashion

culture

- taboo
- cultural

location

time

education

mental status

- trauma
- emotional status

physical status

- allergy
- handicap

religion

- taboo

preferences

- taste

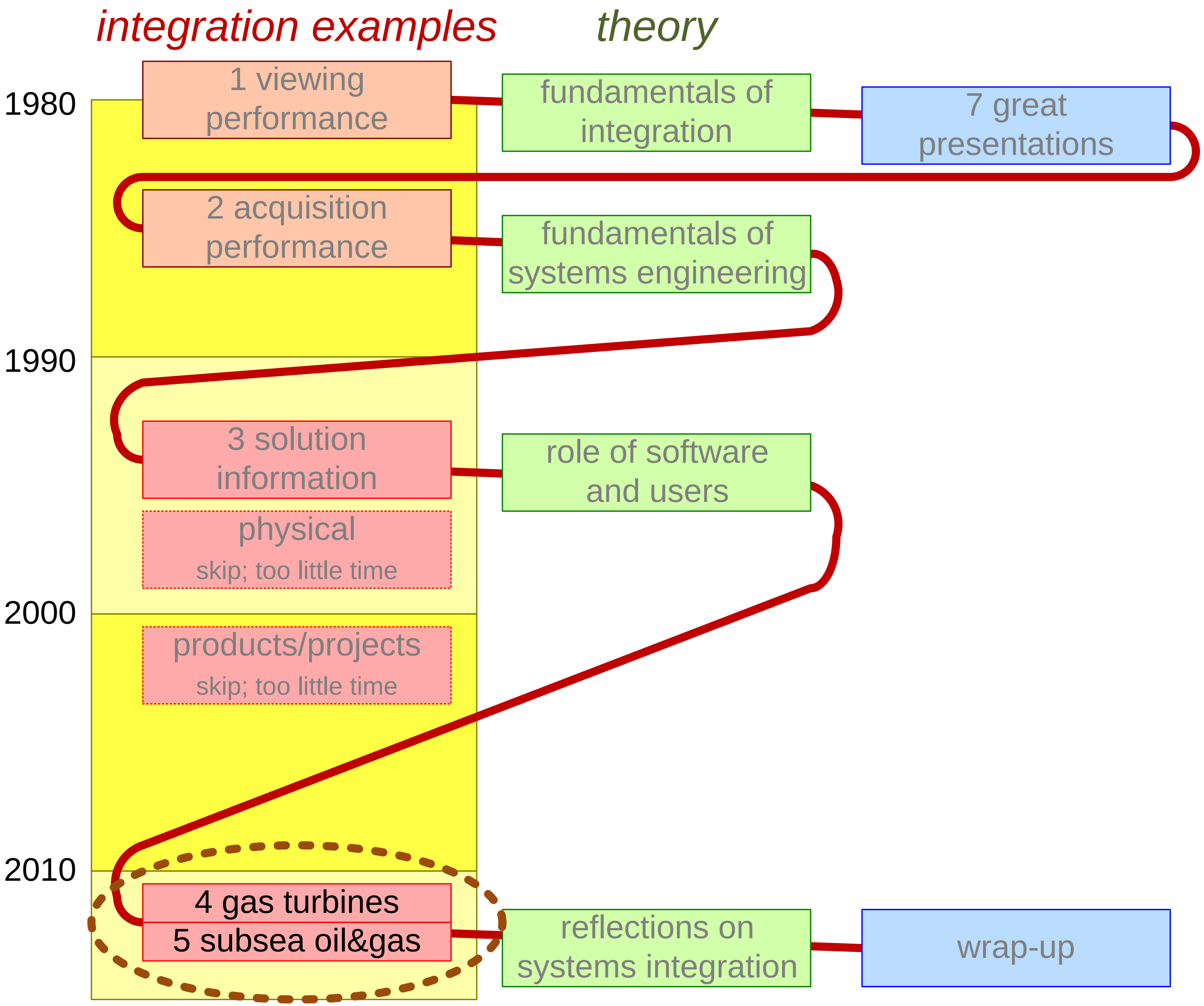
Role of Users

Users:

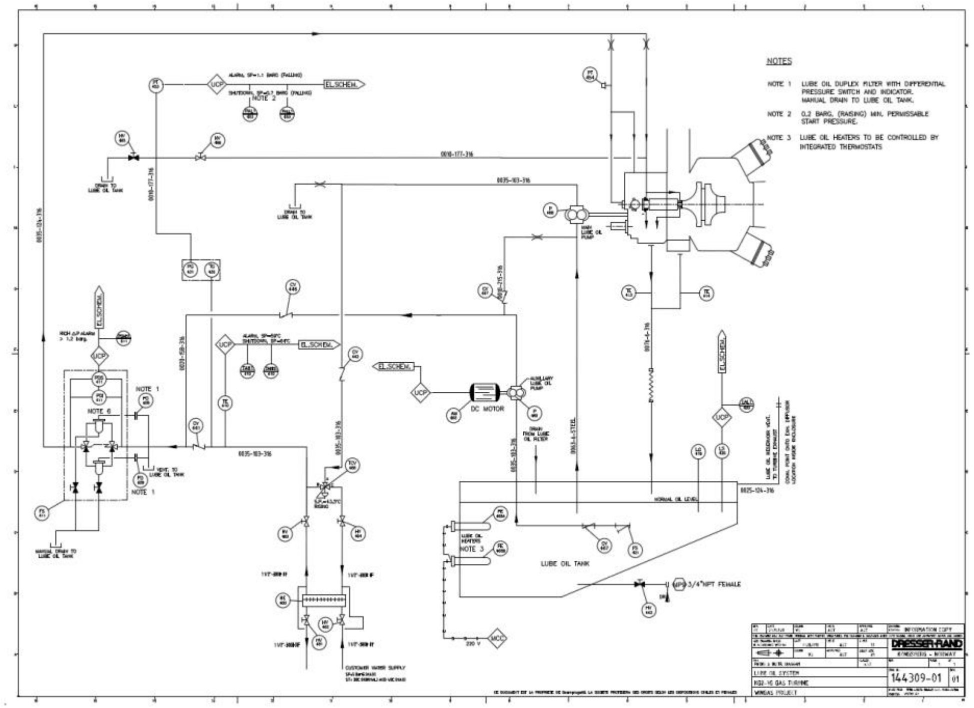
- are autonomous
- behave under influence of internal and external drivers
- are creative
- “solve” problems
- have limited knowledge of the system
- have limited insight in their impact on the system

Users do the unexpected

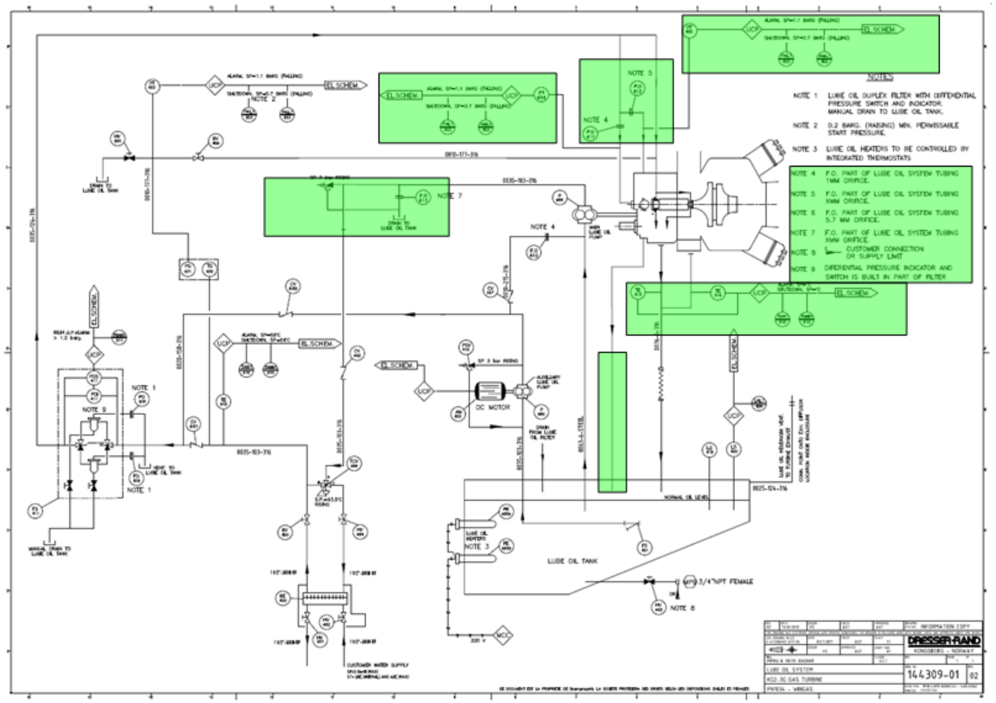
Today in Kongsberg



Errors Found after Functional Analysis and Quantification



PID before

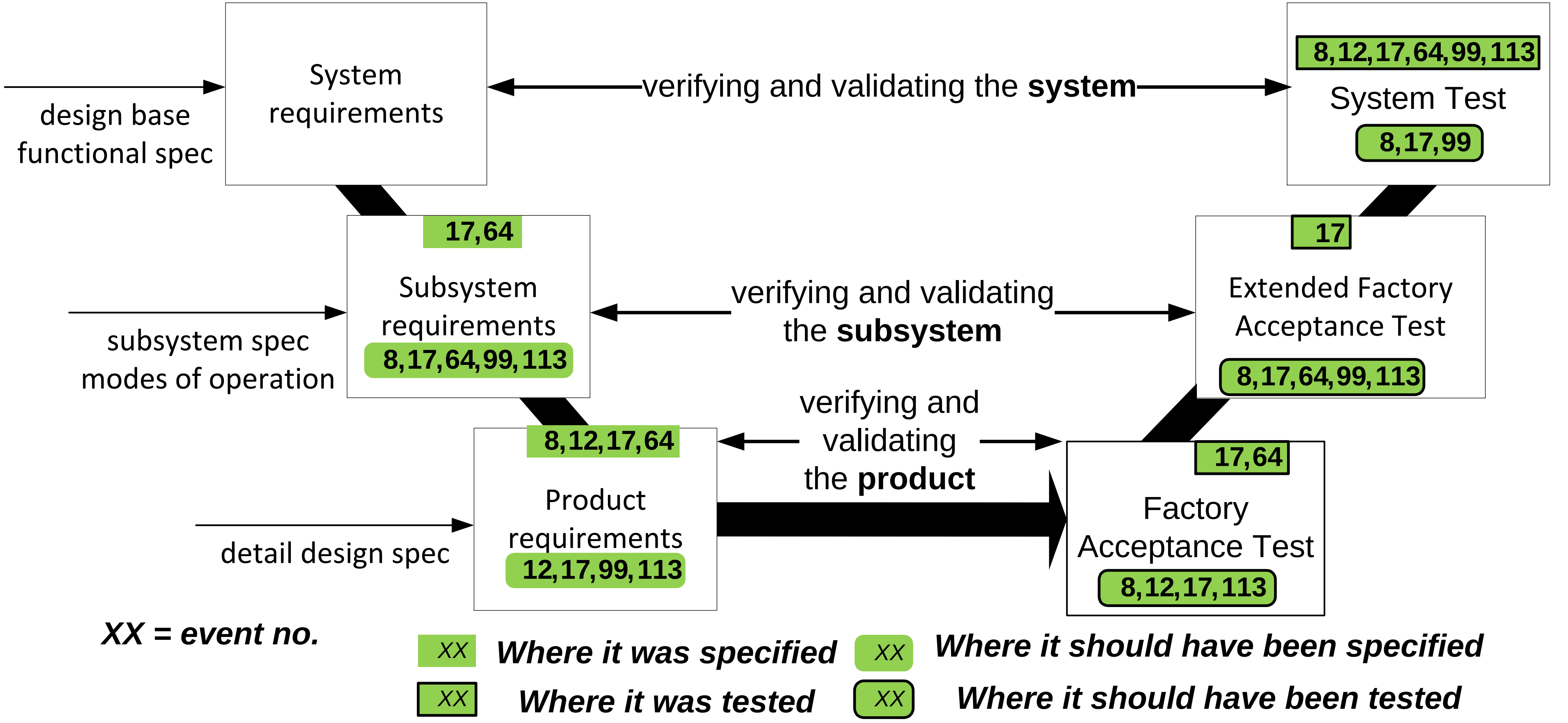


PID after

🟩 changed due to functional analysis and quantification

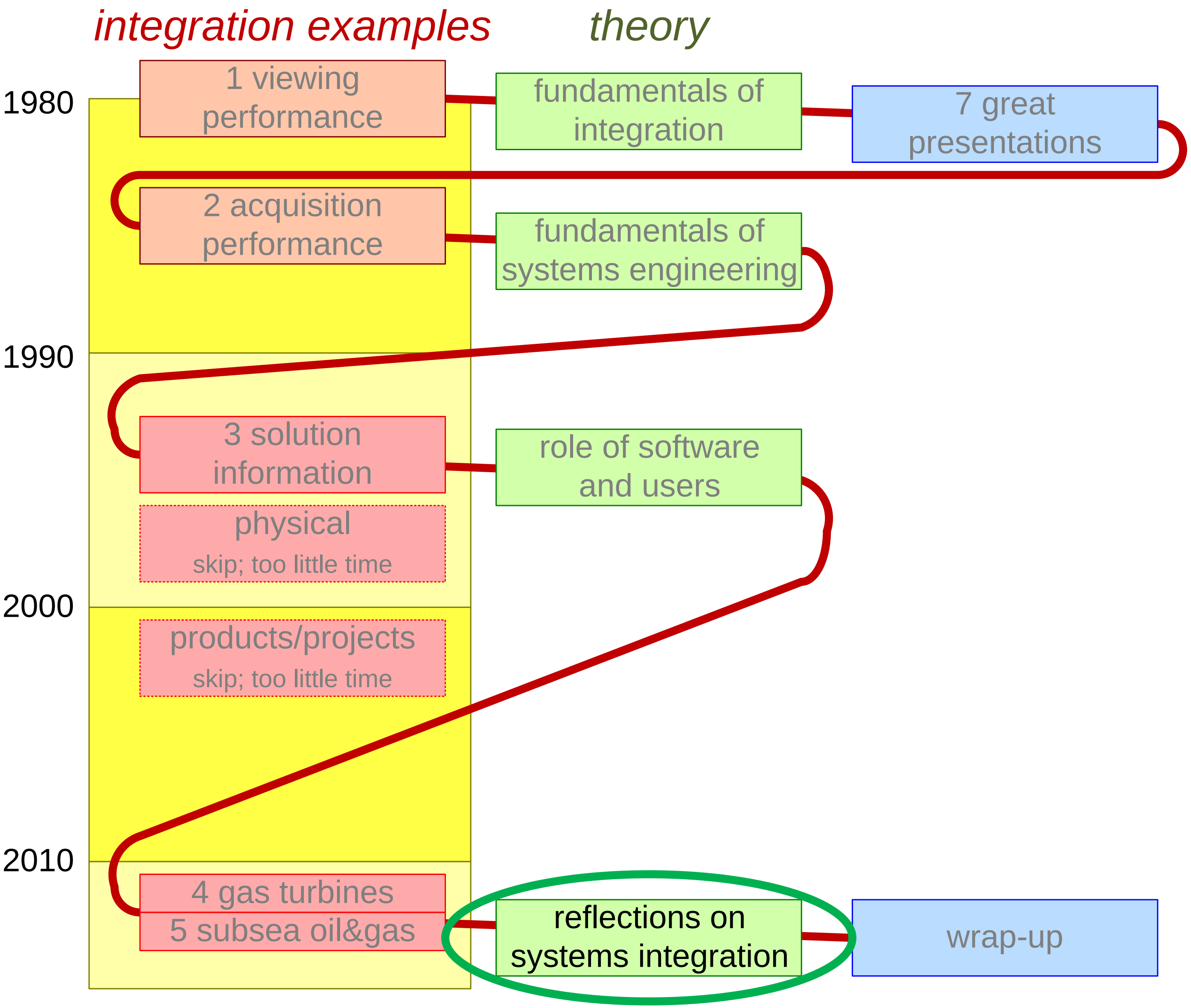
from Knowledge Capture, Cross Boundary Communication and Early Validation with Dynamic A3 Architectures
by Vickram Singh http://www.gaudisite.nl/INCOSE2013_Singh_Muller_DynamicA3.pdf

Analysis of Subsea System Test

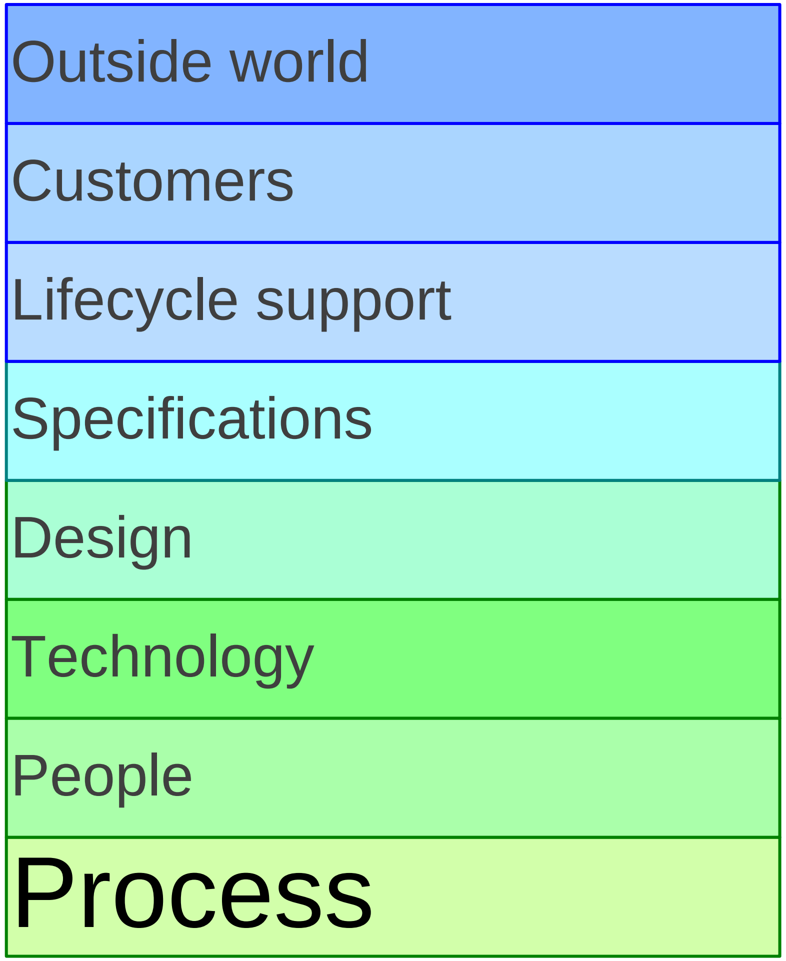

design base
functional spec
System requirements
verifying and validating the **system**
8,12,17,64,99,113
System Test
8,17,99
subsystem spec
modes of operation
17,64
Subsystem requirements
8,17,64,99,113
verifying and validating the **subsystem**
17
Extended Factory Acceptance Test
8,17,64,99,113
detail design spec
8,12,17,64
Product requirements
12,17,99,113
verifying and validating the **product**
17,64
Factory Acceptance Test
8,12,17,113
***XX* = event no.**
XX ***Where it was specified***
XX ***Where it should have been specified***
XX ***Where it was tested***
XX ***Where it should have been tested***

from master project by Åke Törnlycke and Rune Henden, FMC, 2012

Reflections on Systems Integration

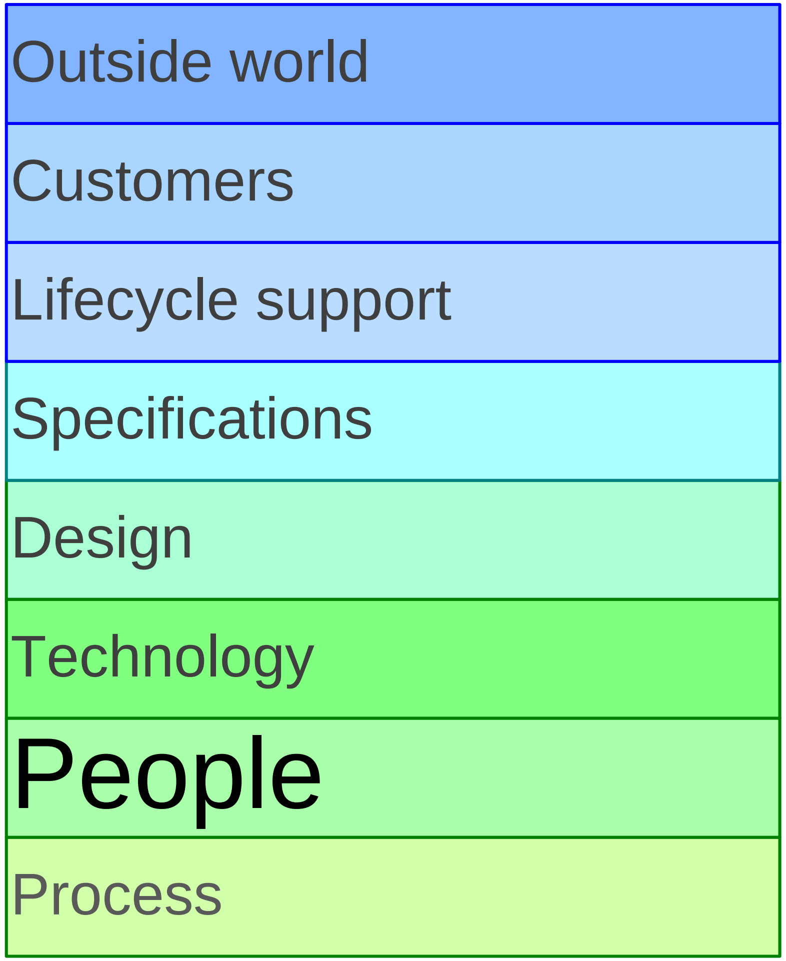


Imperfect Processes



- result and delivery oriented
- artifact oriented (documents!)
- “check mark” syndrome

Imperfect People



Outside world

Customers

Lifecycle support

Specifications

Design

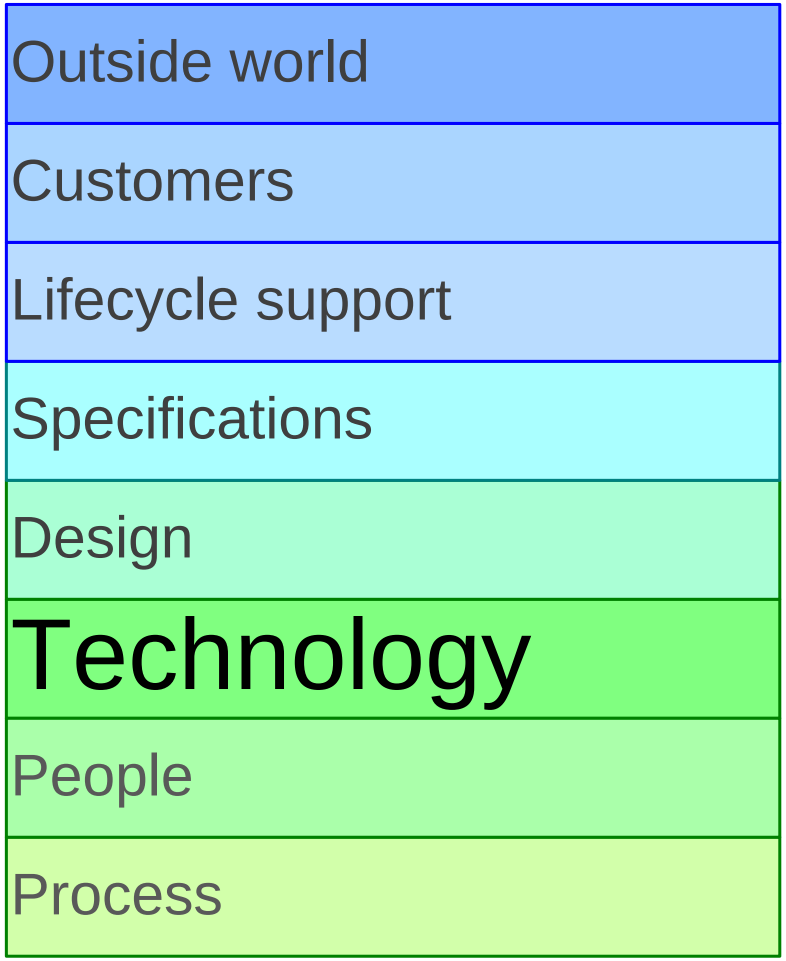
Technology

People

Process

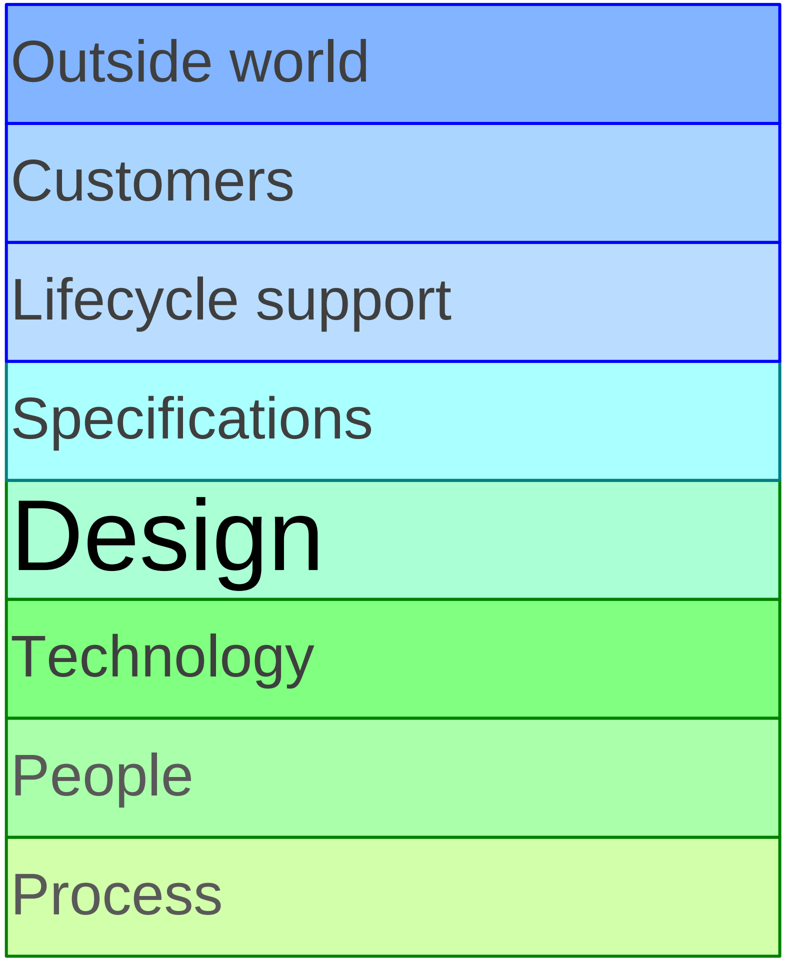
- see only a small part of the big picture
- are unaware of their blind spots
- are adaptable and intelligent

Imperfect Technology



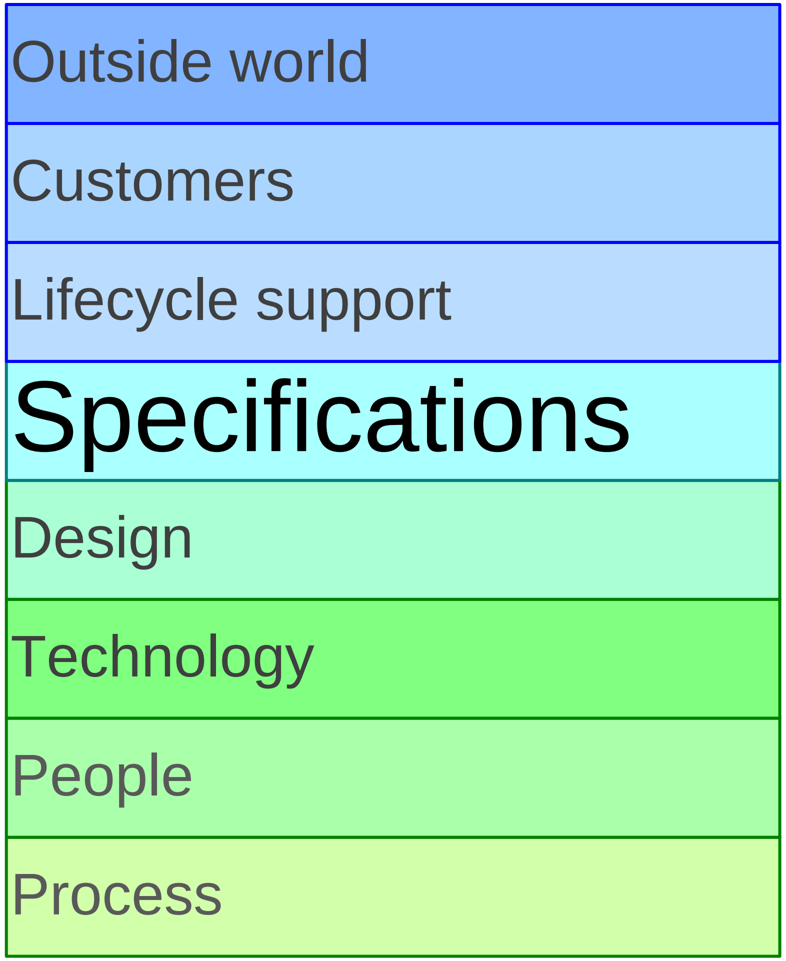
- builds on math, physics, etc.
- even experts do not understand all
- vendors may supply it

Imperfect Design



- multi-disciplinary
- many faceted (parts, functions, qualities)

Imperfect Specifications


Outside world

Customers

Lifecycle support

Specifications

Design

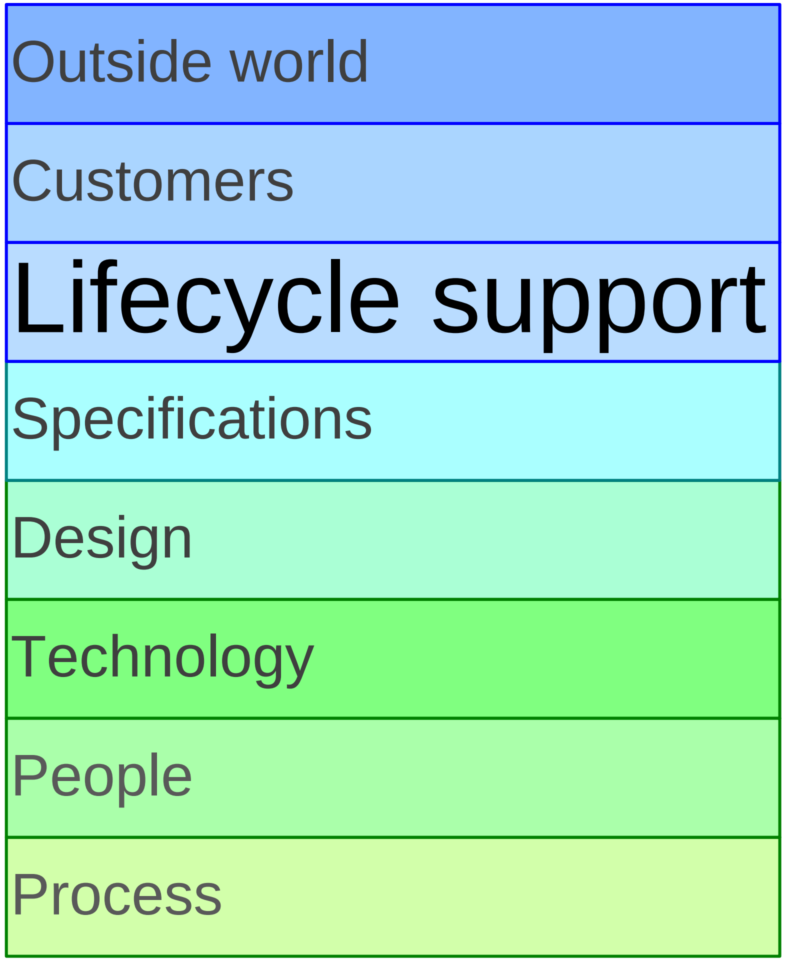
Technology

People

Process

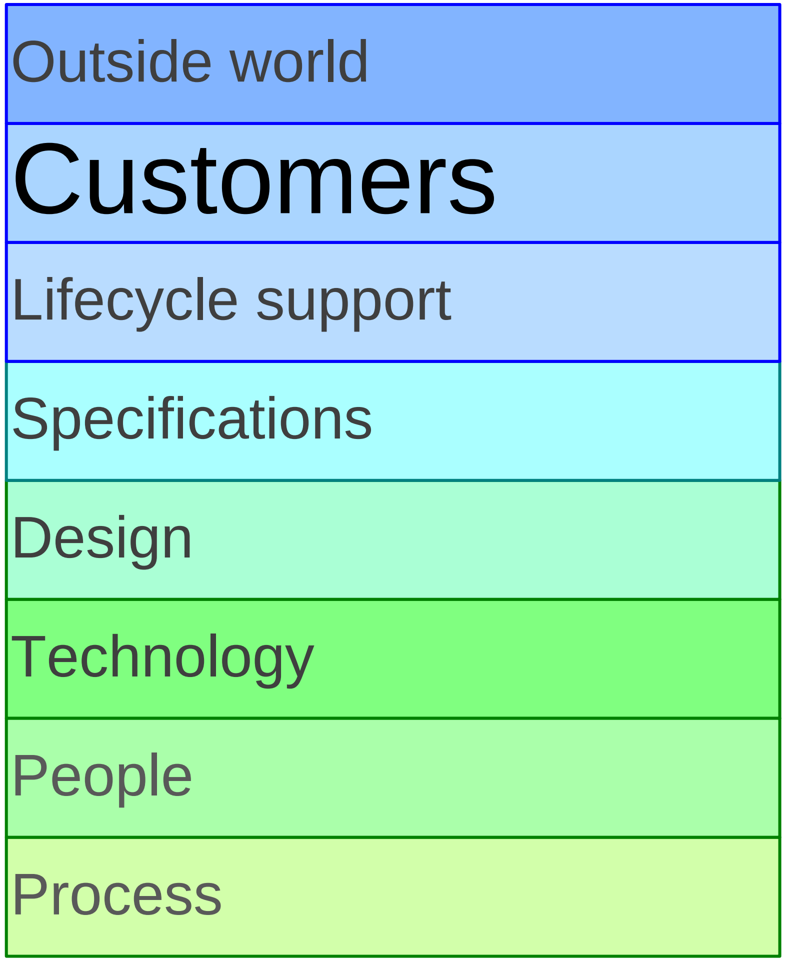
- are never complete
- are often polluted with solutions
- are often internally inconsistent
- tend to lack sharpness

Imperfect Lidecycle Support



- many lifecycles
- many stakeholders
- many rhythms

Imperfect Customers



Outside world

Customers

Lifecycle support

Specifications

Design

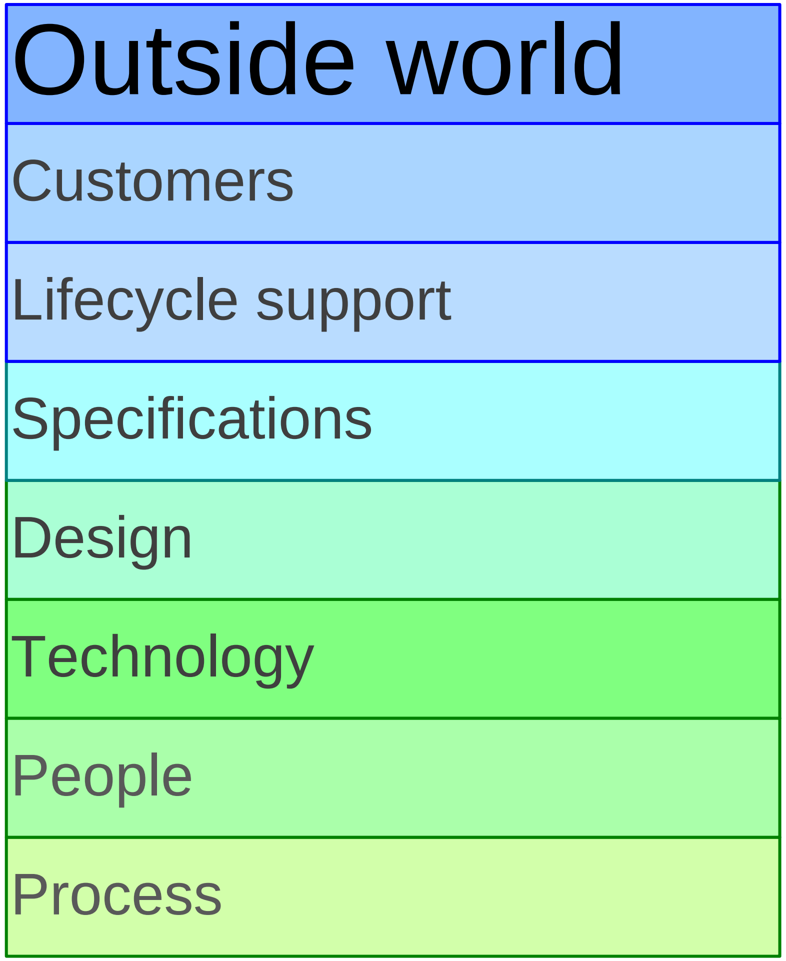
Technology

People

Process

- complicated environment
- politics
- do not know what they need
- do the unexpected

Imperfect Outside World



Outside world

Customers

Lifecycle support

Specifications

Design

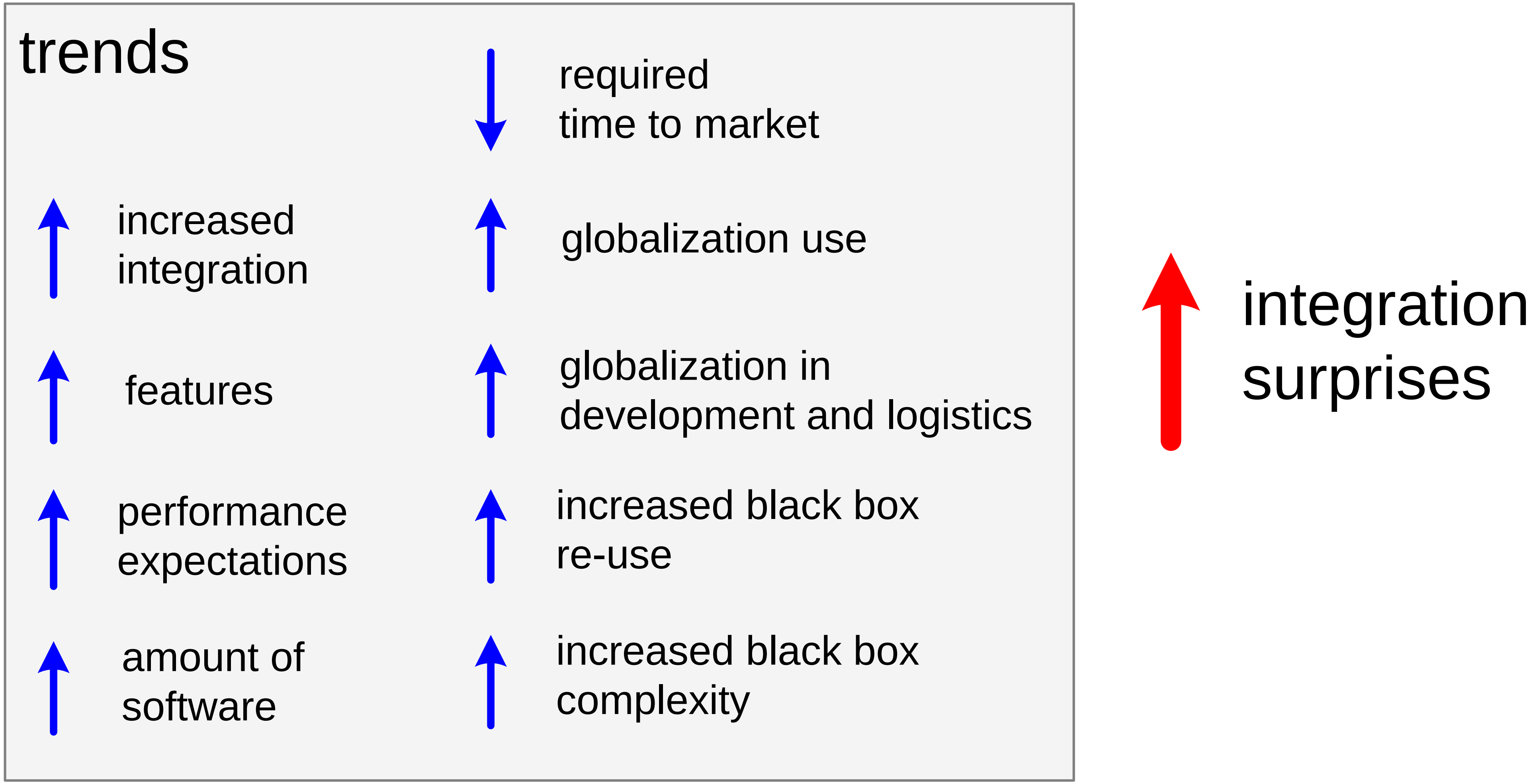
Technology

People

Process

- social complexity (humans)
- natural complexity
- interaction between natural and artificial world

Without Measures it only gets Worse...



trends

↓ required time to market

↑ increased integration

↑ globalization use

↑ features

↑ globalization in development and logistics

↑ performance expectations

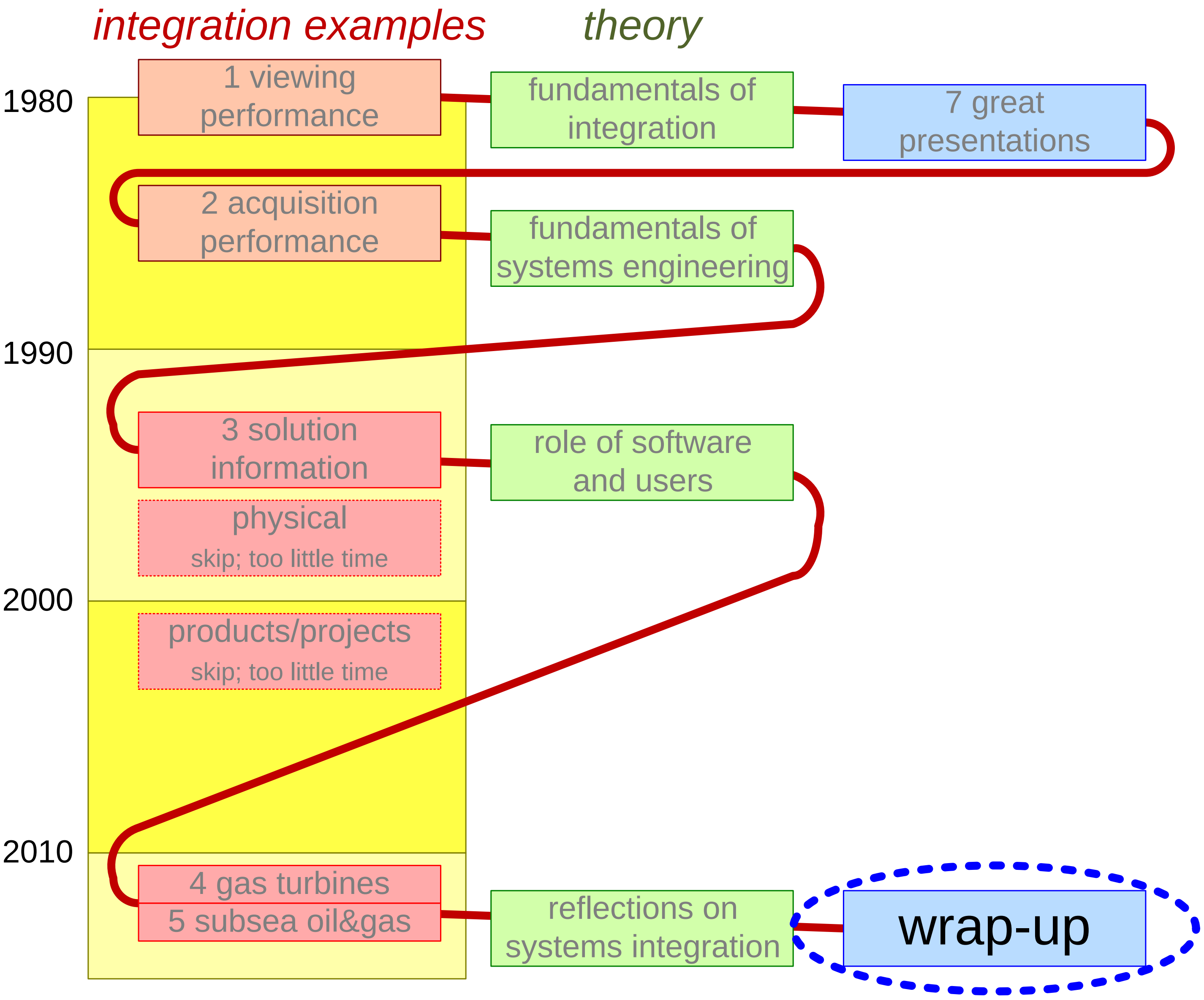
↑ increased black box re-use

↑ amount of software

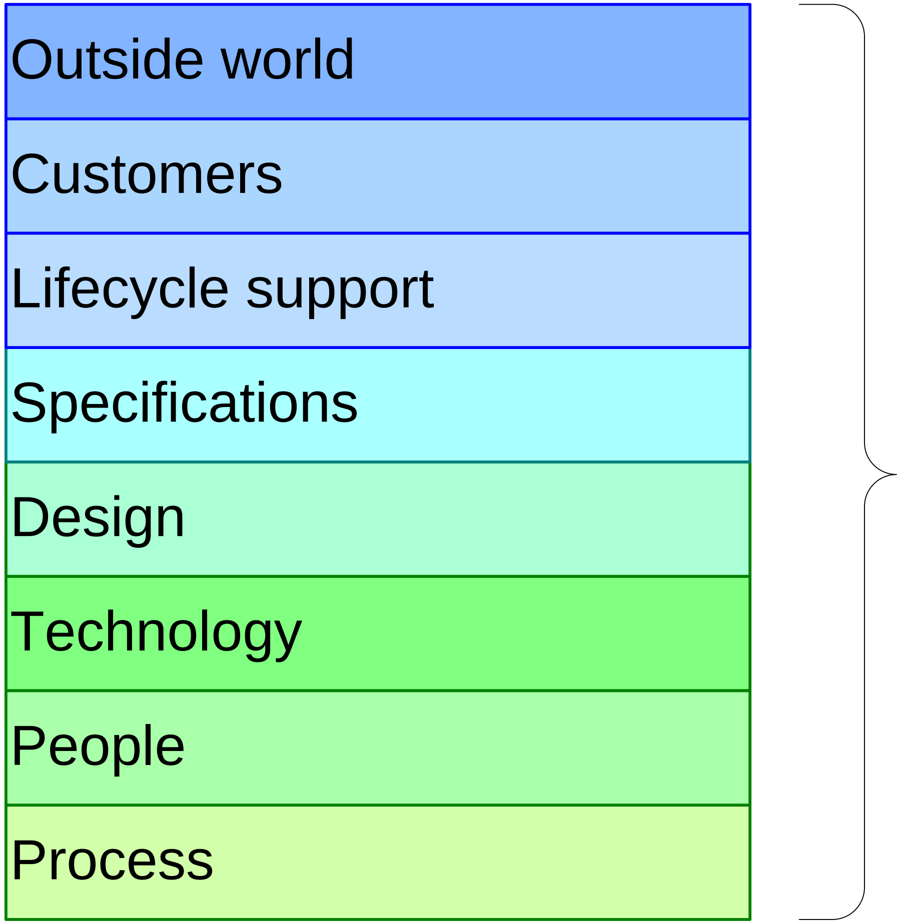
↑ increased black box complexity

↑ integration surprises

Wrap-up

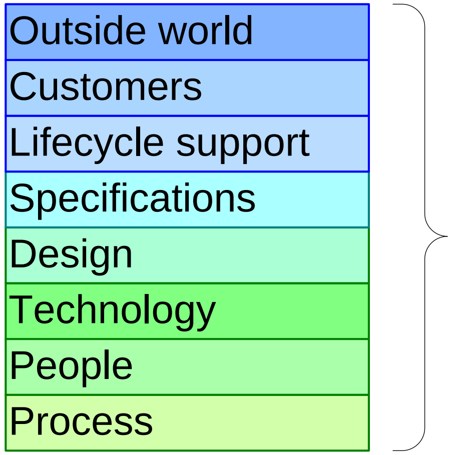

integration examples
theory
1980
1990
2000
2010
1 viewing performance
fundamentals of integration
7 great presentations
2 acquisition performance
fundamentals of systems engineering
3 solution information
role of software and users
physical
skip; too little time
products/projects
skip; too little time
4 gas turbines
5 subsea oil&gas
reflections on systems integration
wrap-up

Conclusion on Reflections



plenty of imperfections!

How to Counter all of this?



Outside world

Customers

Lifecycle support

Specifications

Design

Technology

People

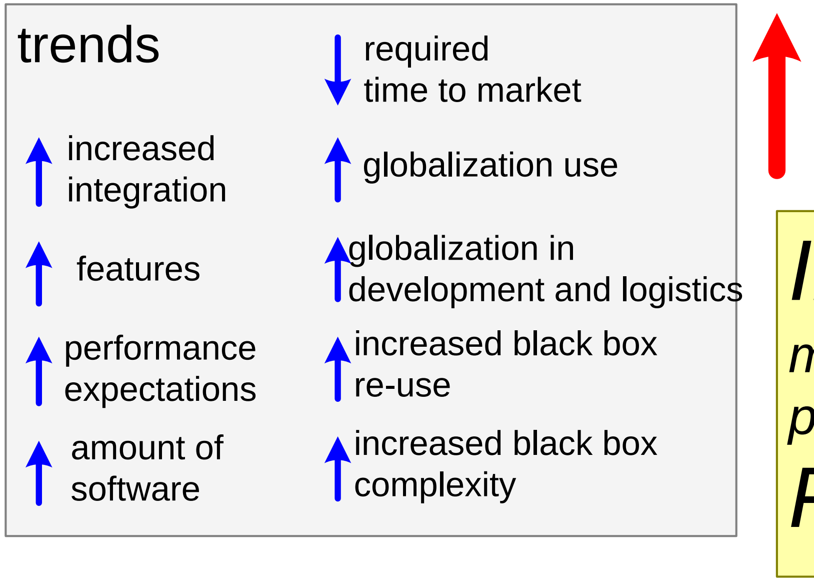
Process

plenty of imperfections!

Fail Early:

“proof” key performance ASAP

use partial integrations



trends

↓ required time to market

↑ increased integration

↑ globalization use

↑ features

↑ globalization in development and logistics

↑ performance expectations

↑ increased black box re-use

↑ amount of software

↑ increased black box complexity

↑ integration surprises

Improve System Development:

modeling, analysis, tools

process, people

Focus on Systems Engineering