

# 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`

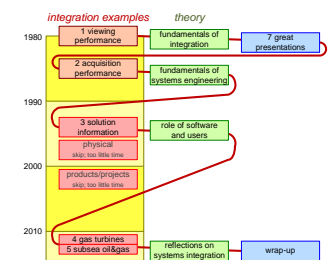
## Abstract

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.

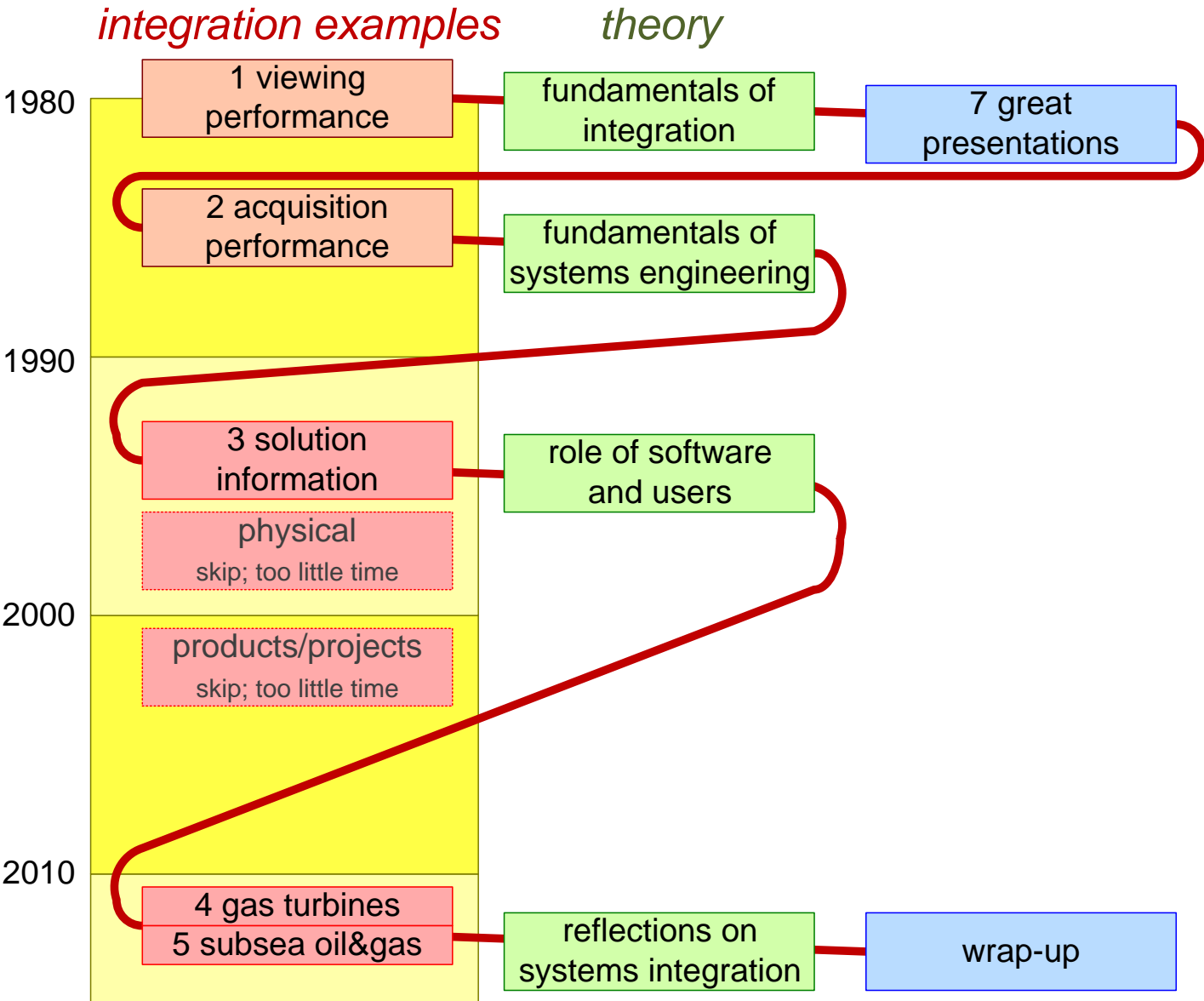
## Distribution

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.

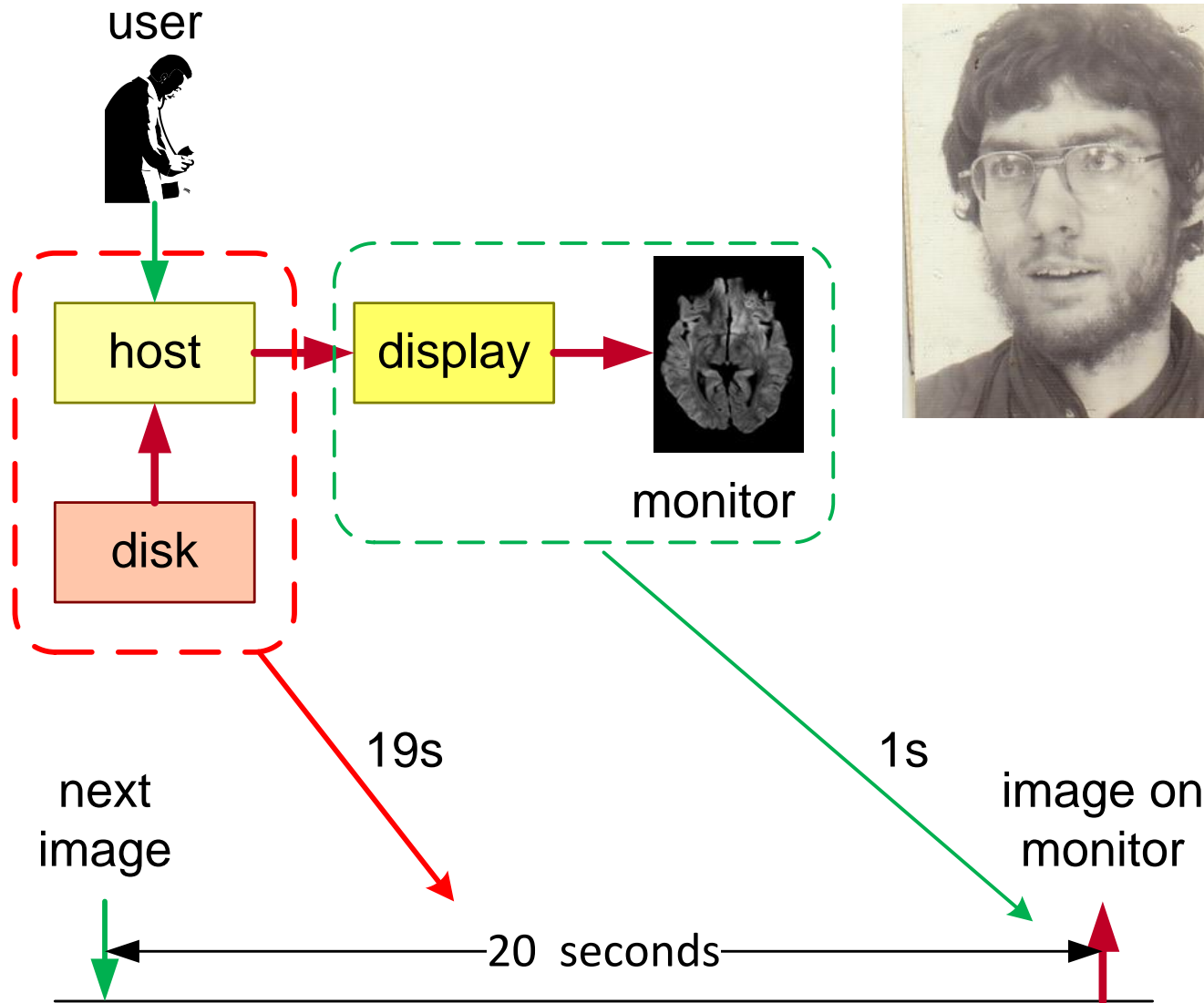
March 4, 2026  
status: planned  
version: 0



# 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:

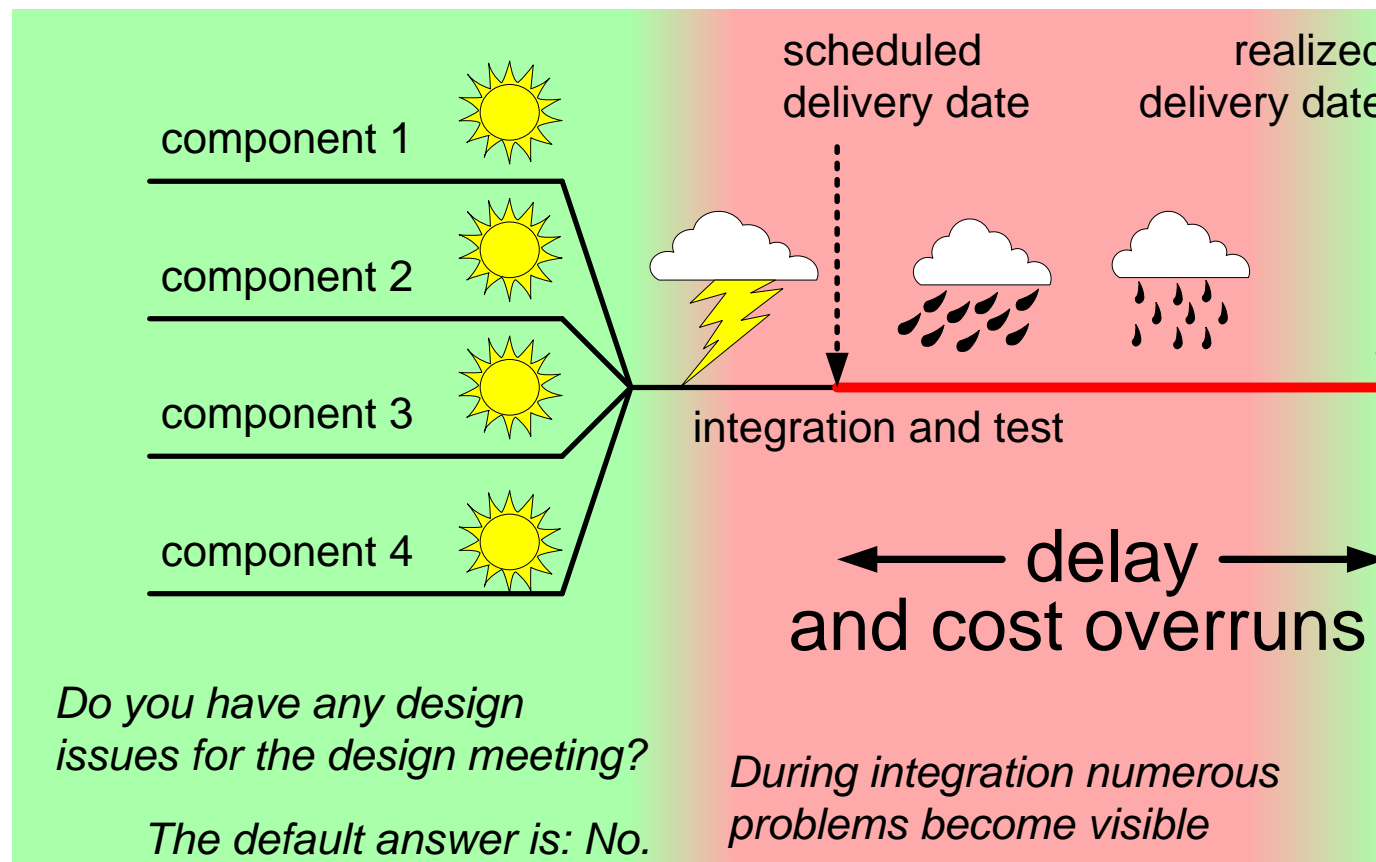
lack of system design

# 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

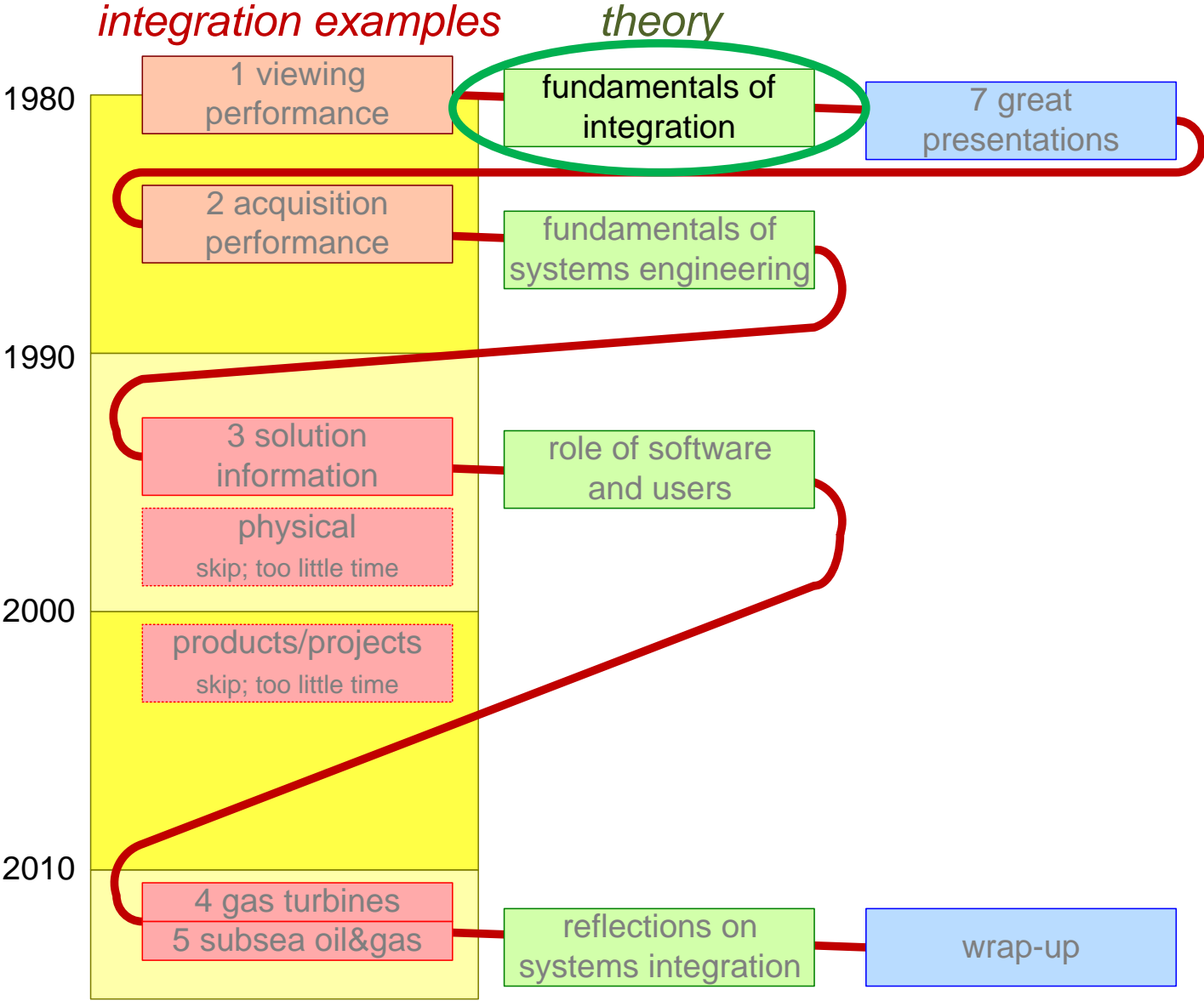
	poor	sufficient	good	very good	excellent	perfect
Outside world					<b>X</b>	
Customers					<b>X</b>	
Lifecycle support					<b>X</b>	
Specifications					<b>X</b>	
Design					<b>X</b>	
Technology					<b>X</b>	
People					<b>X</b>	
Process					<b>X</b>	

**X** expected answers from Kongsberg industry

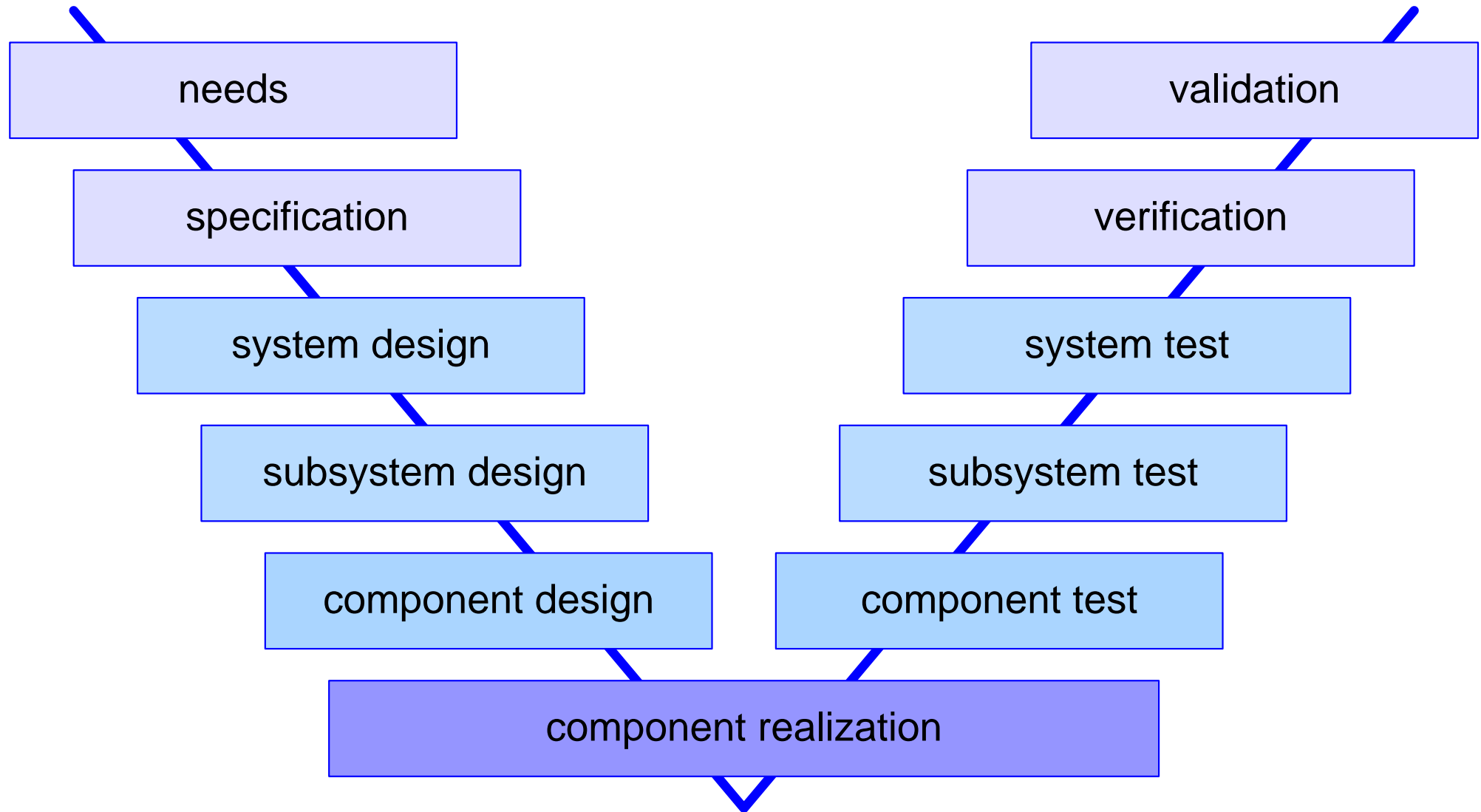
*Perfect processes, people, technologies, designs, or specifications do not exist*

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

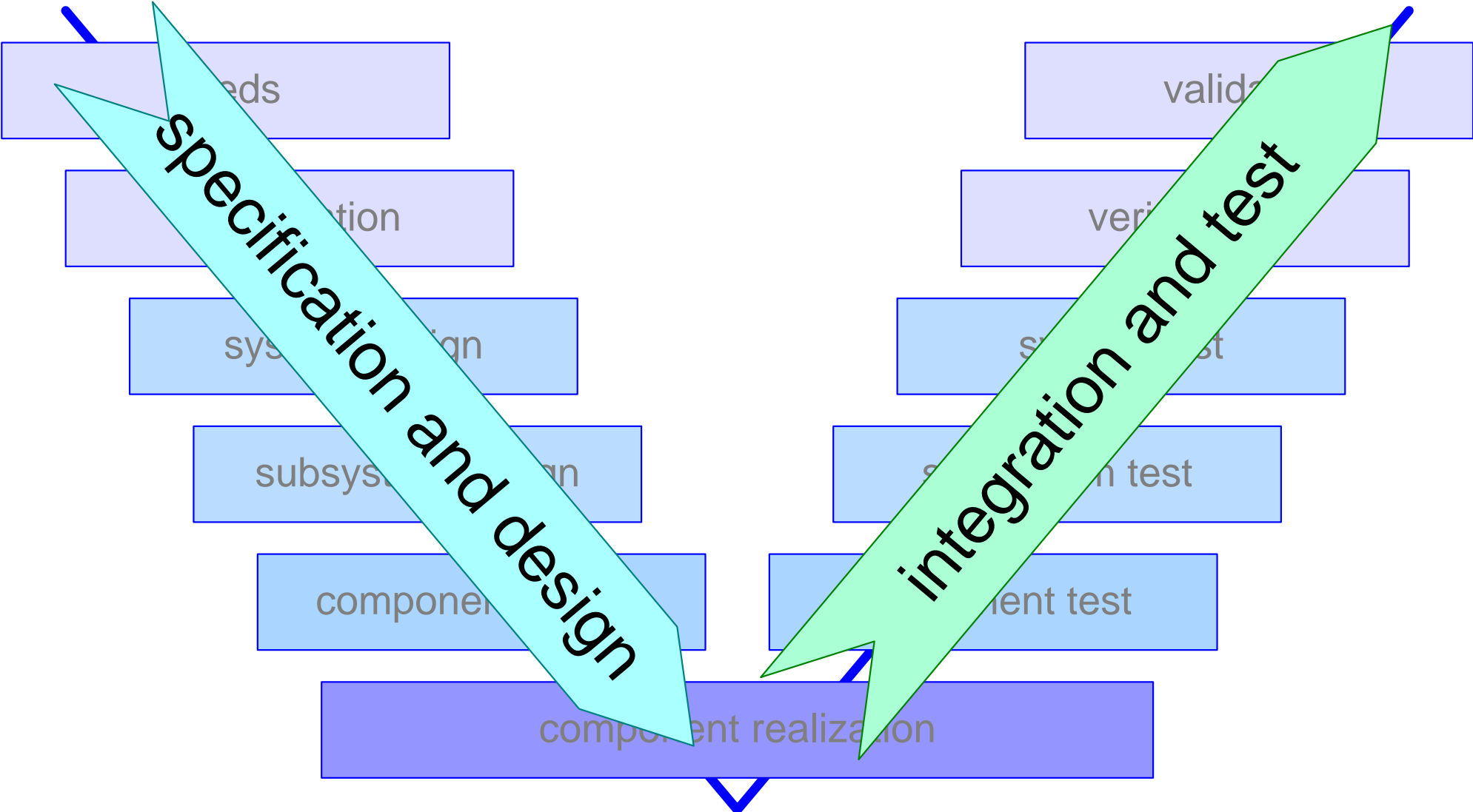
# Fundamentals of Integration



# V-Model

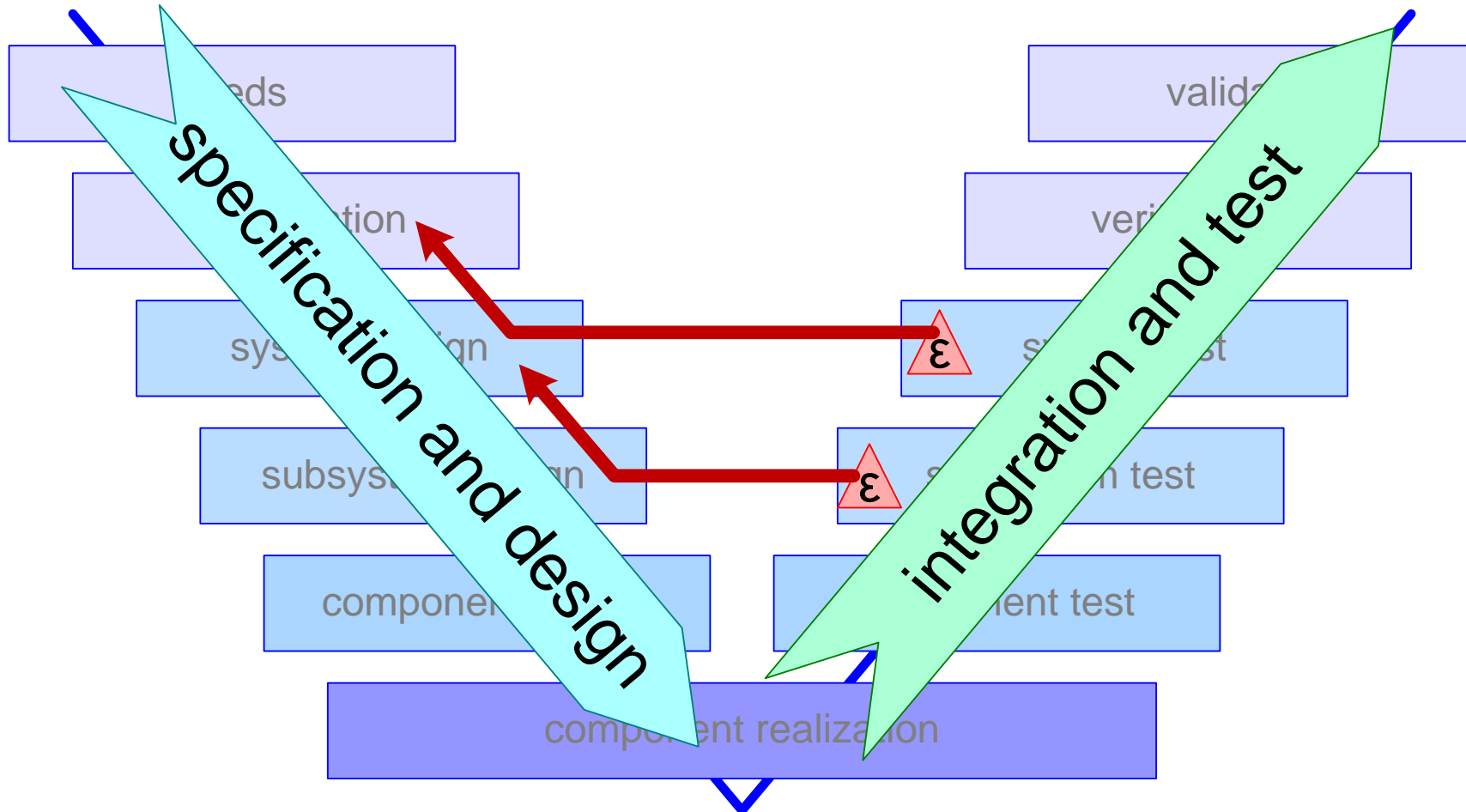


# Conventional Integration View

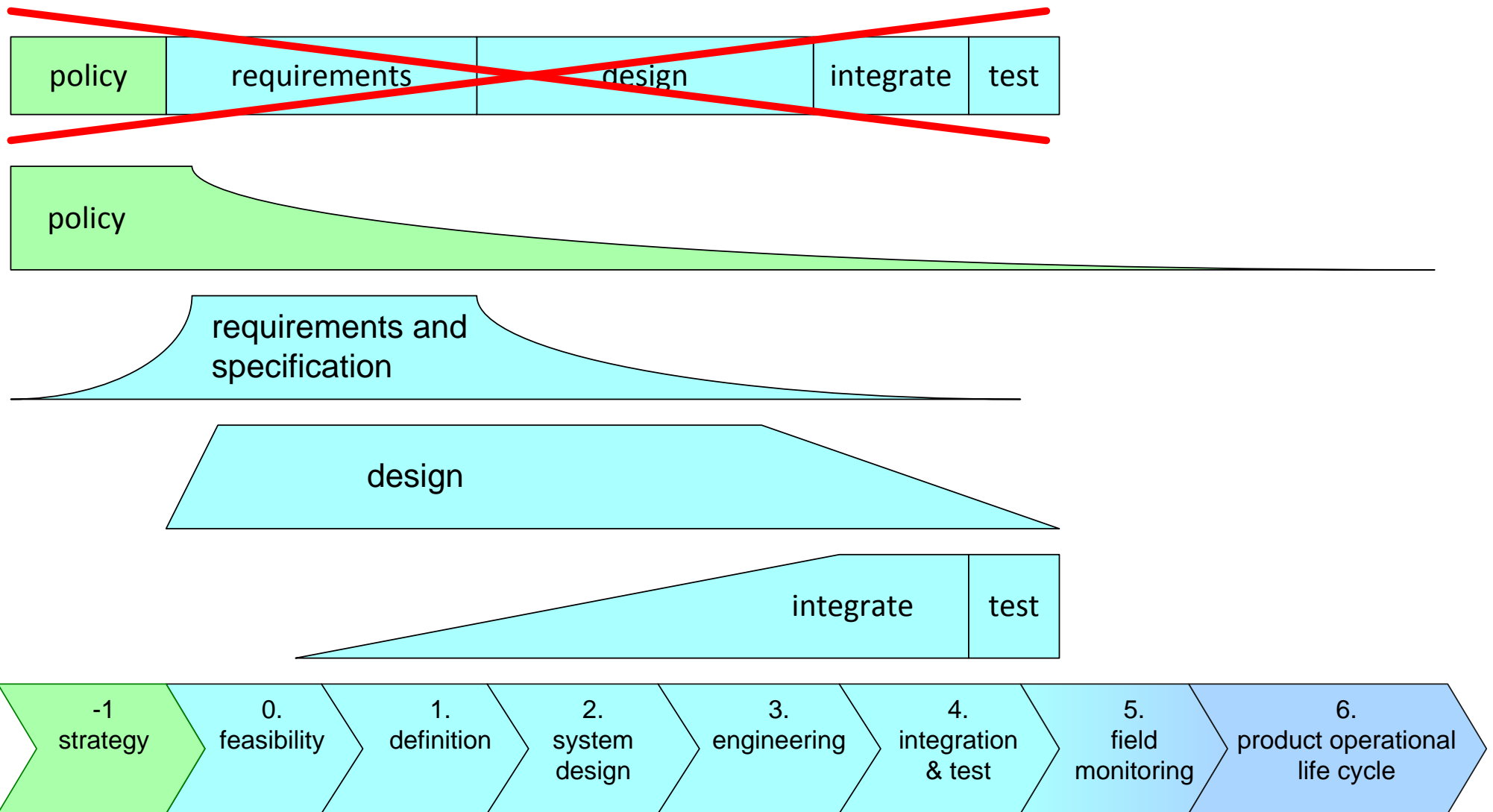


# Limitations in Front-End Cause Failures

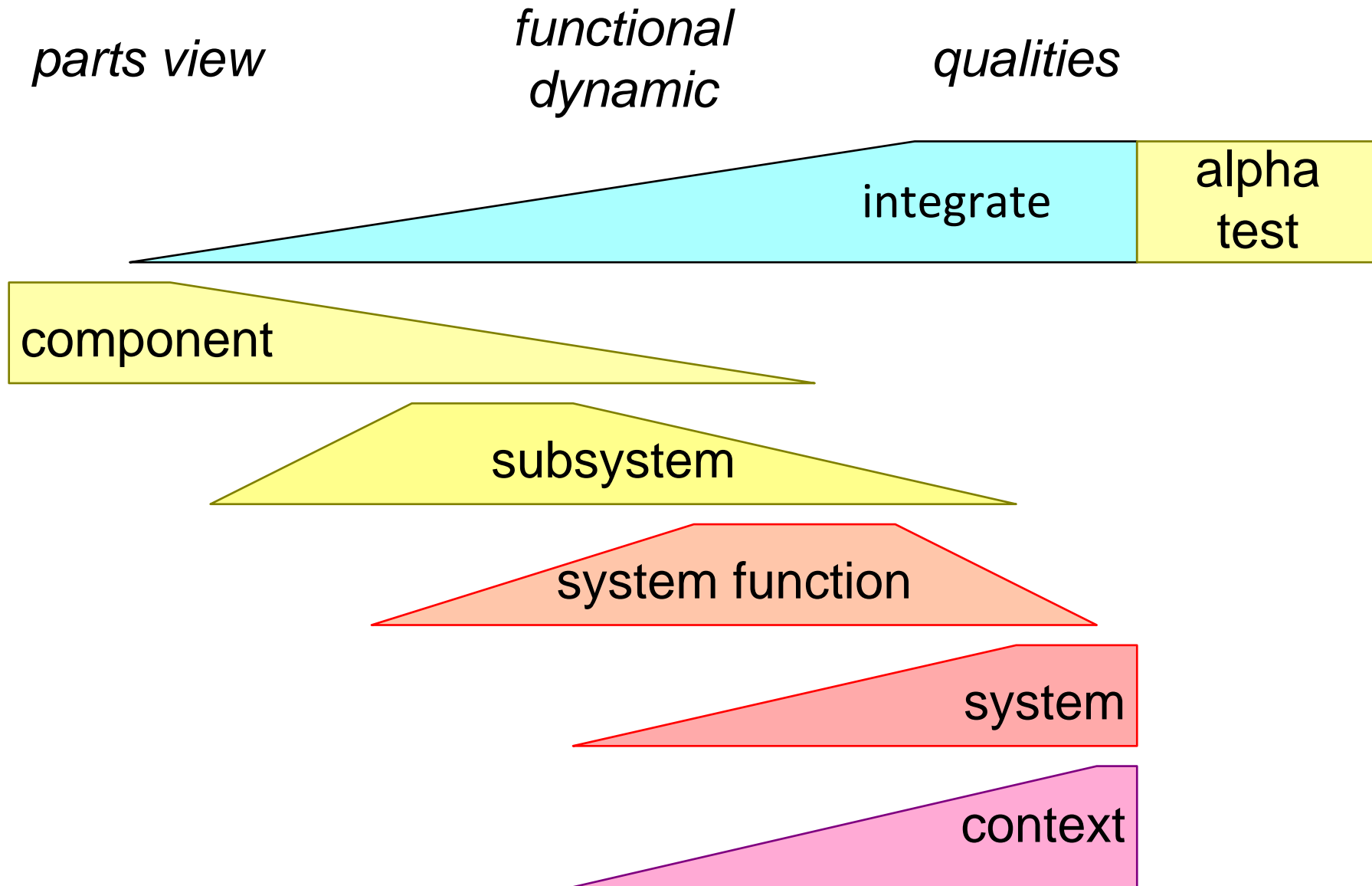
failures found during integration  
can be traced back to *unknowns*,  
*unforeseens*, and *wrong assumptions*



# Typical Concurrent Product Creation Process



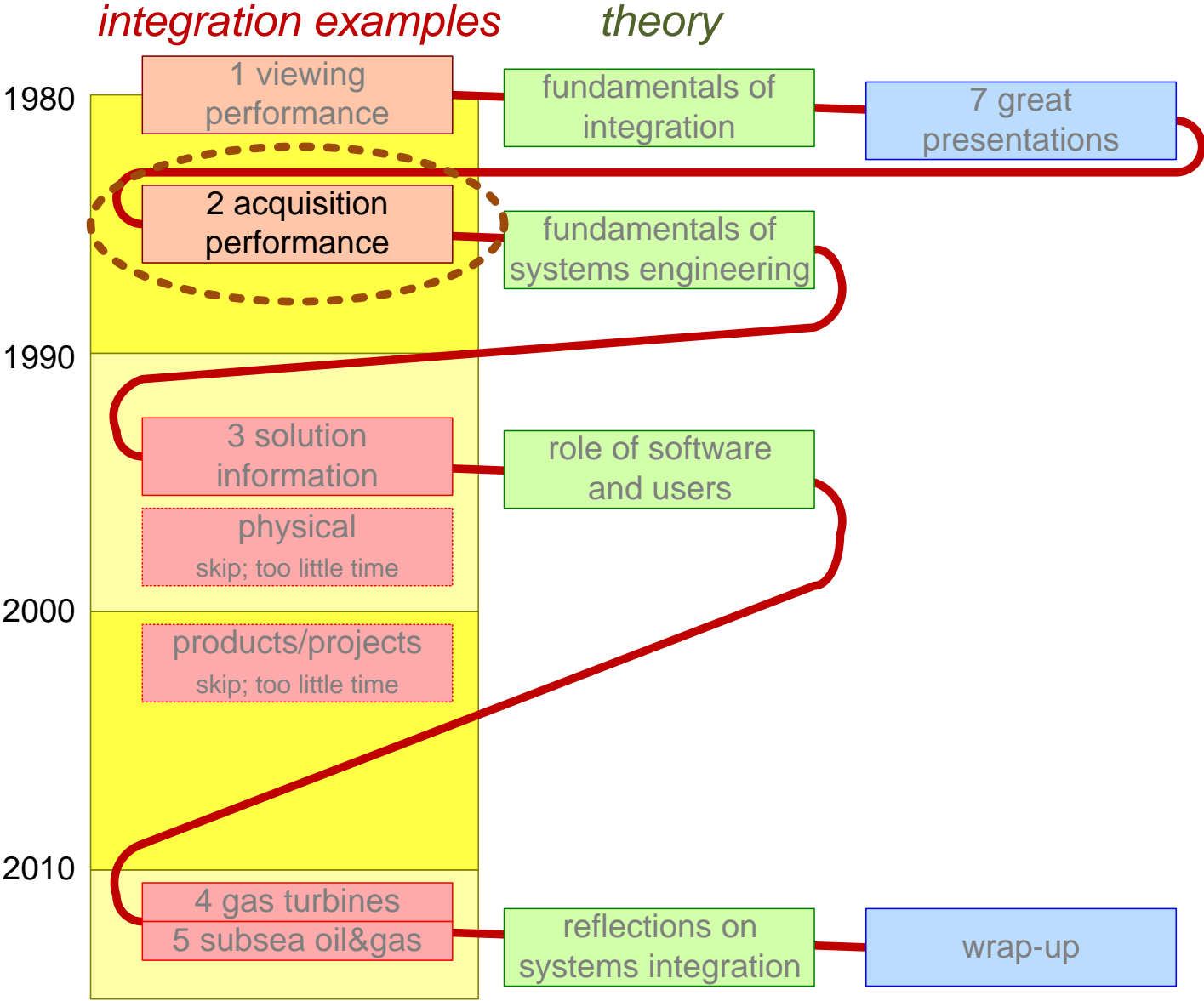
# Integration Takes Place in a Bottom-up Fashion



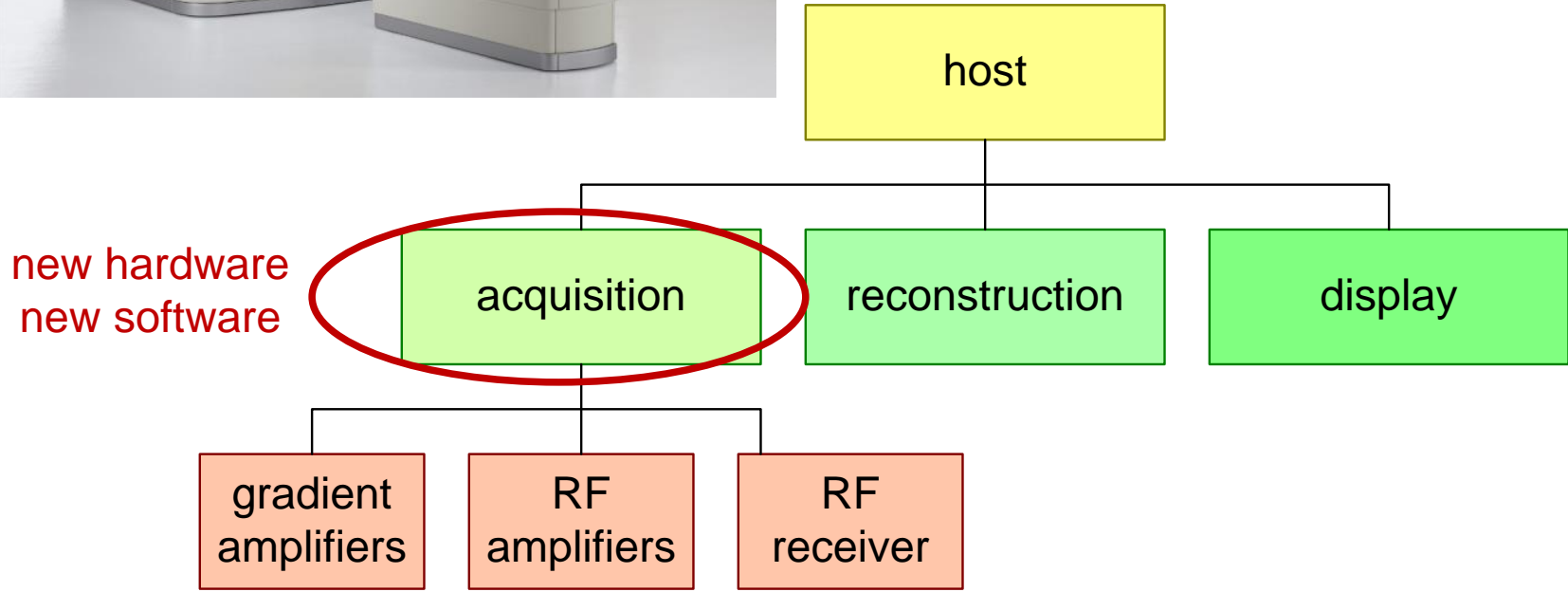
# Fill in this form during KSEE 2013!

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?
<b>Niels Braspenning</b> System Integration at ASML: Linking Technical Content, Test Configurations, Timing... And People!		
<b>Alejandro Salado</b> Validation risks of using development methodologies in a hierarchical fashion - When contracts meet architecture ownership		
<b>Andreas Thorvaldsen</b> Changing A System From Within – And Get Hit By The Unexpected Surprises		
<b>Benoît Le Bihan</b> Laggan Tormore Project System Test: when new Subsea Solutions For Harsh Environment Meet Reality		
<b>Jim Armstrong</b> Systems Integration: What Are We Waiting For?		
<b>Terje Jensvik</b> A software centric approach to Electronic Systems Engineering.		
<b>Eldar Tranøy</b> Early phase need analysis – Can we ease systems integration?		
<b>Gerrit Muller</b> Why is Systems Integration understood so poorly? Reflections on 3 decades of unforeseen failures		

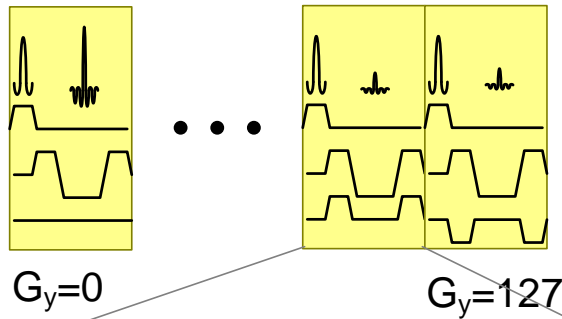
# Example 2: Performance Again



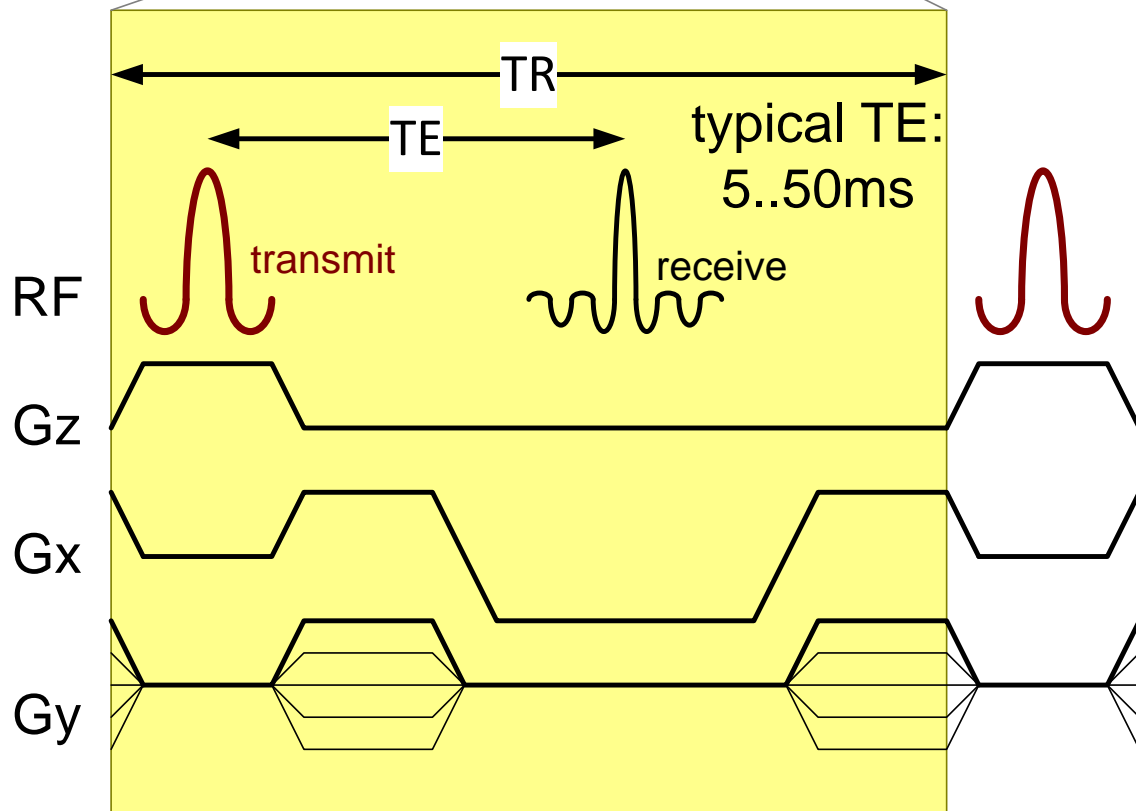
# Example 2: Integration of MRI Acquisition Subsystem



# Repetition Time MRI



imaging =  
repeating similar pattern  
many times

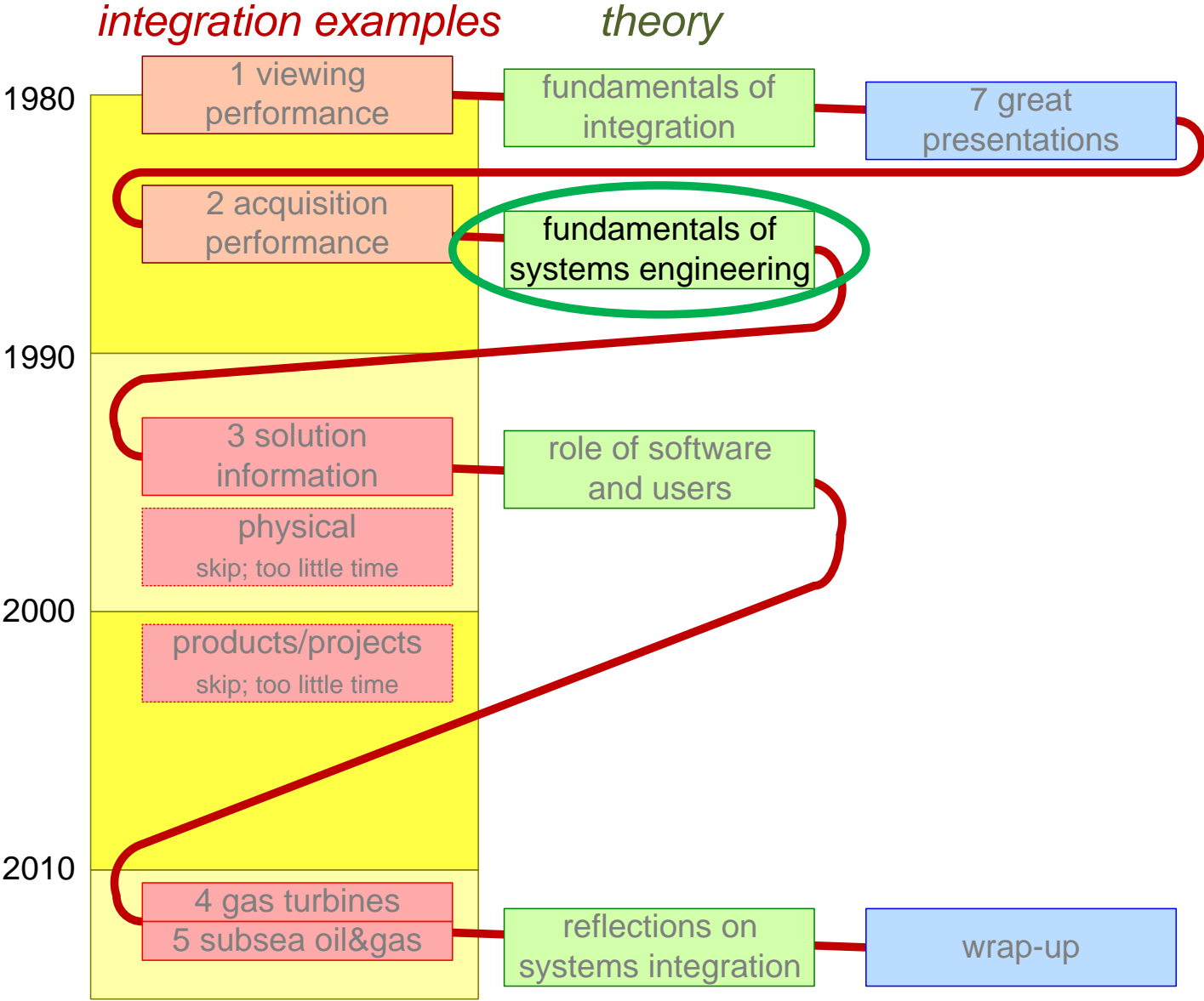


problem:  
TR > 1 s.  
spec less than 10 ms  
*more than factor 100 off!*

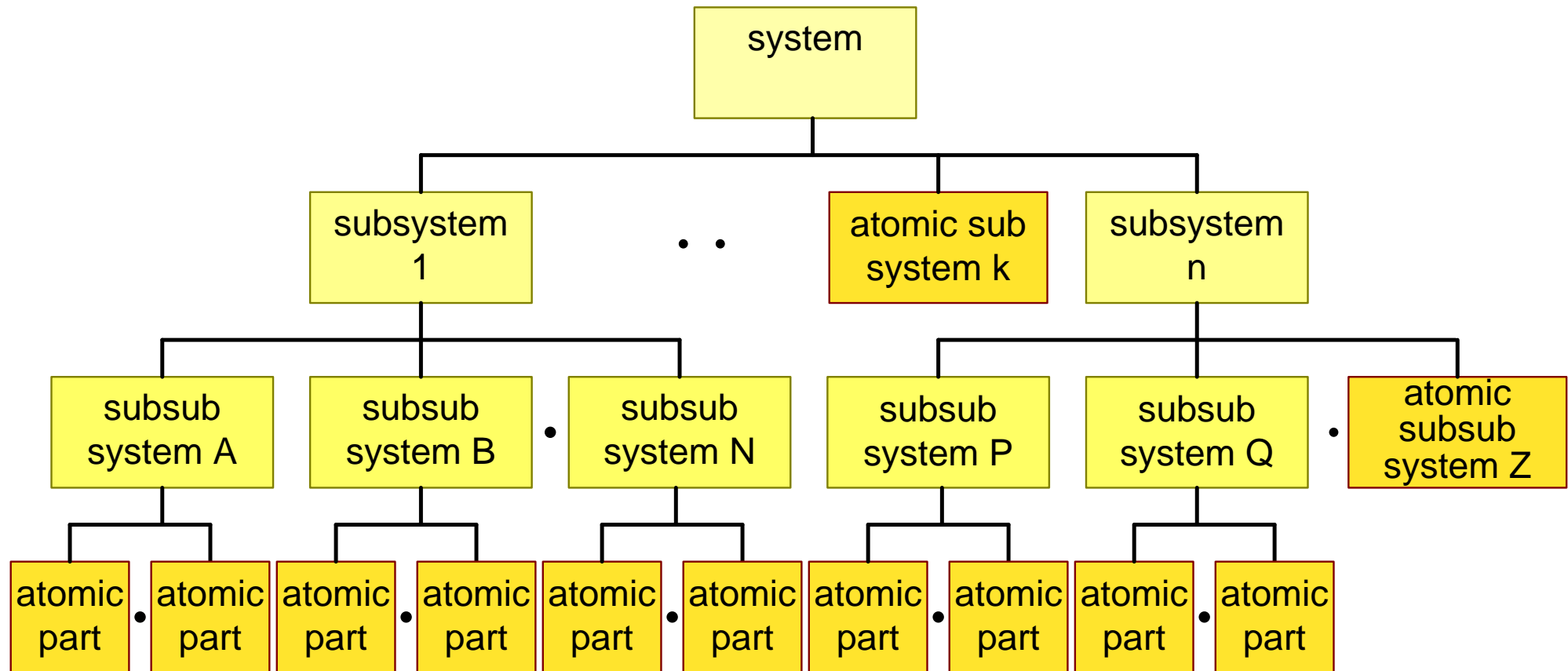
causes:  
floating point arithmetic  
too many layers

root cause:  
functionality focus

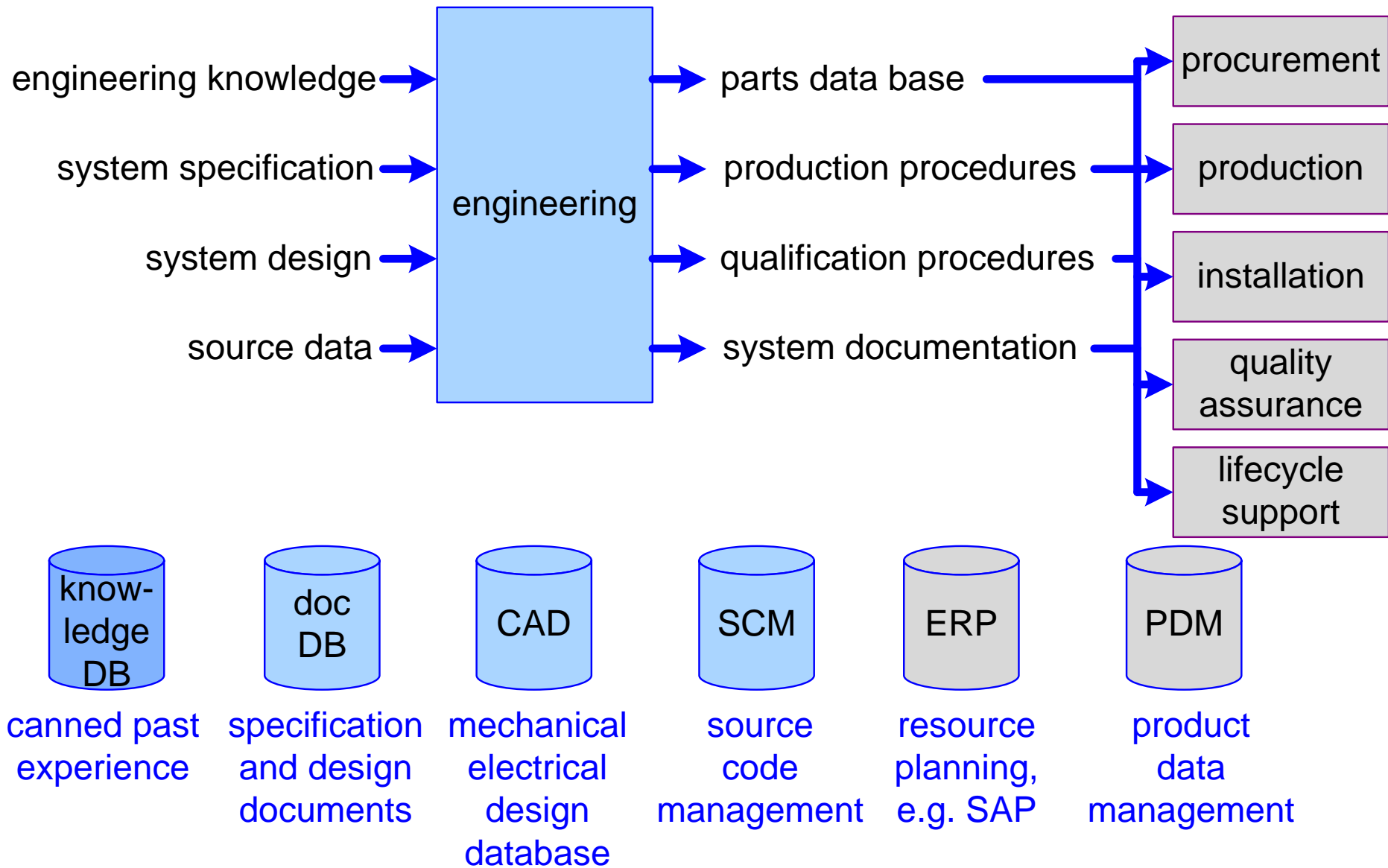
# Fundamentals of Systems Engineering



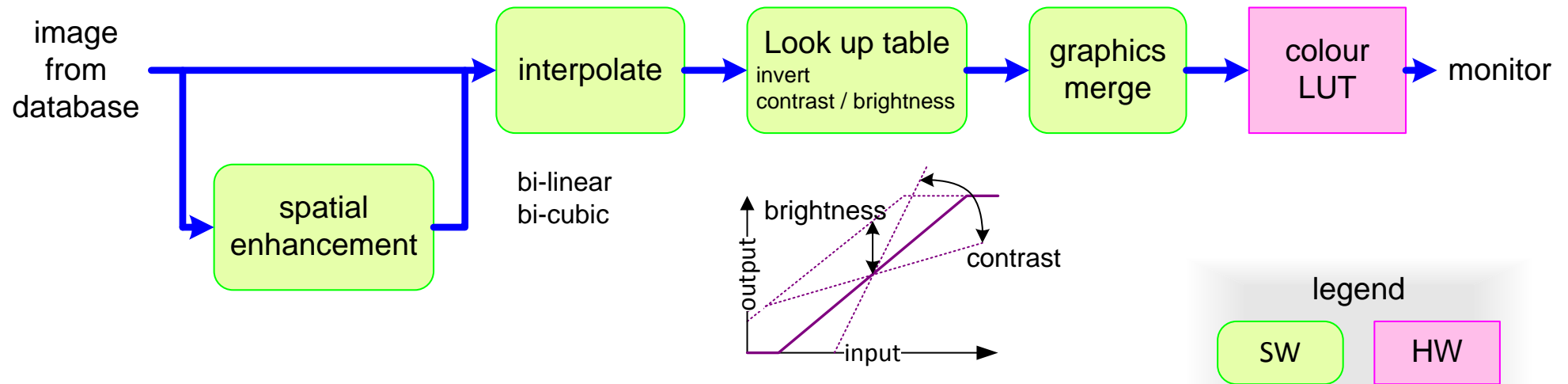
# SE Rule 1: Partition and Define Interfaces



# 99% of Organization has a “Parts” Focus

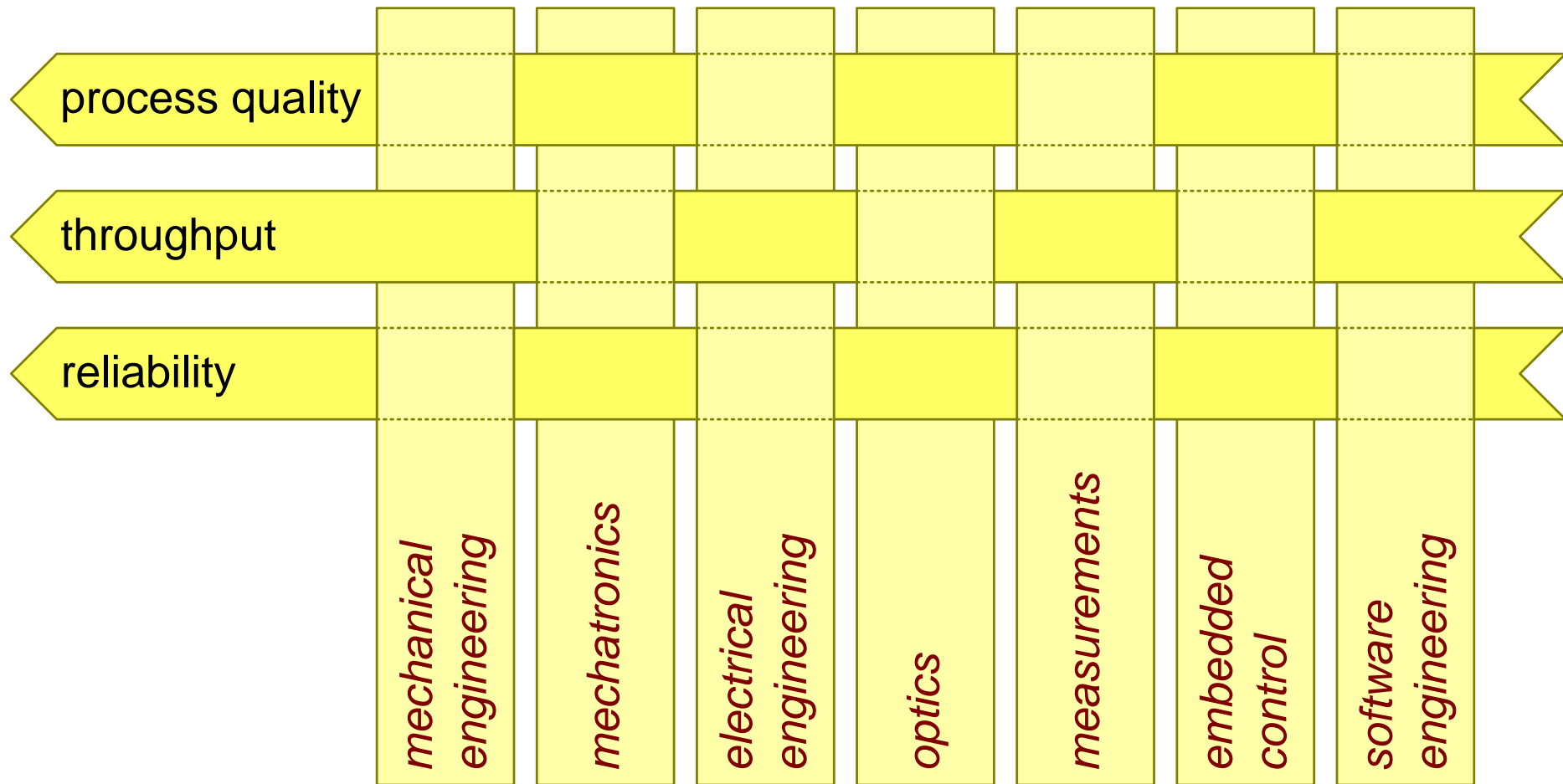


# 10%? Understands Dynamic Behavior or Functionality



# Few Understand Key Performance Parameters

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

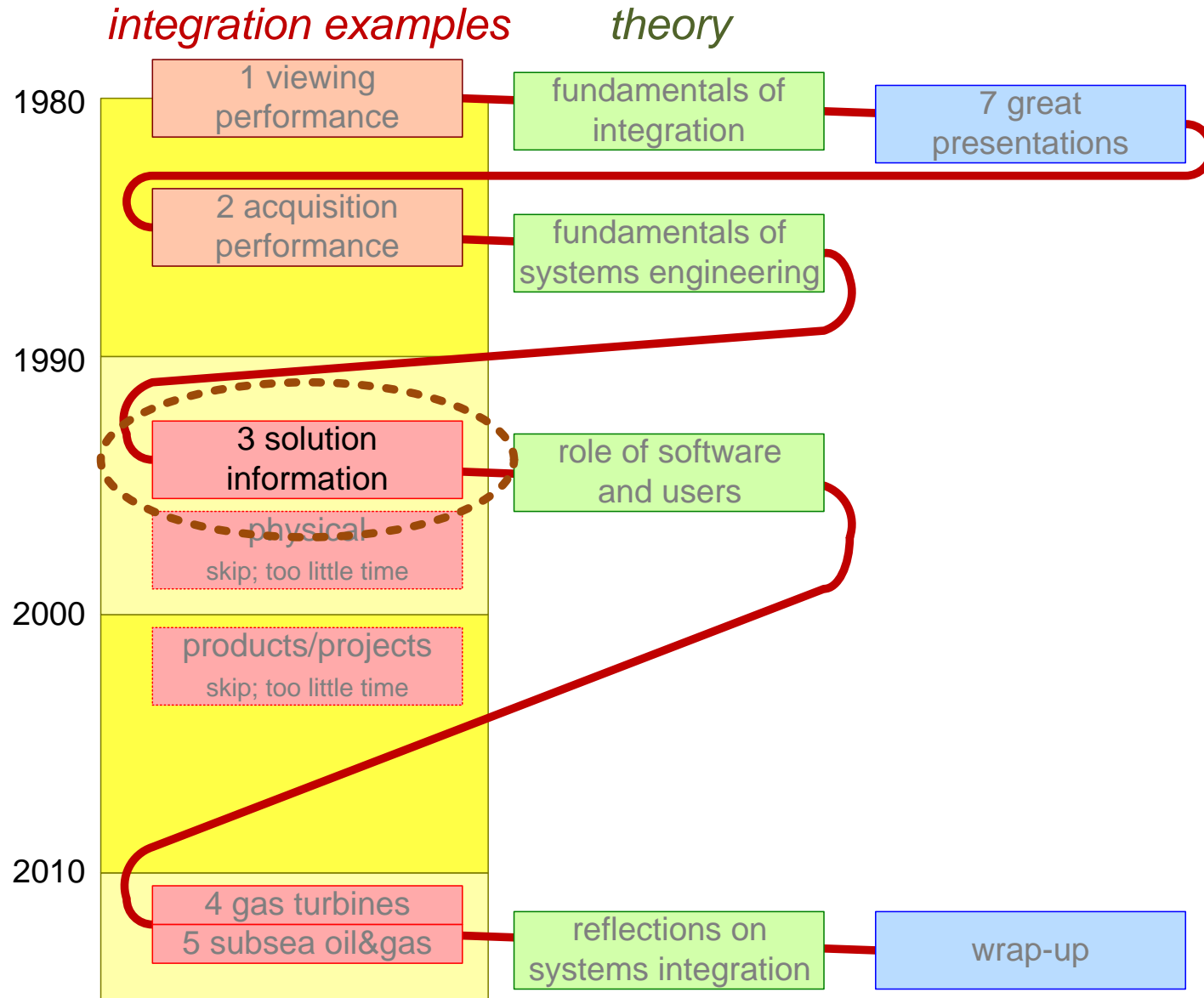


# Typical Order of Integration Problems

---

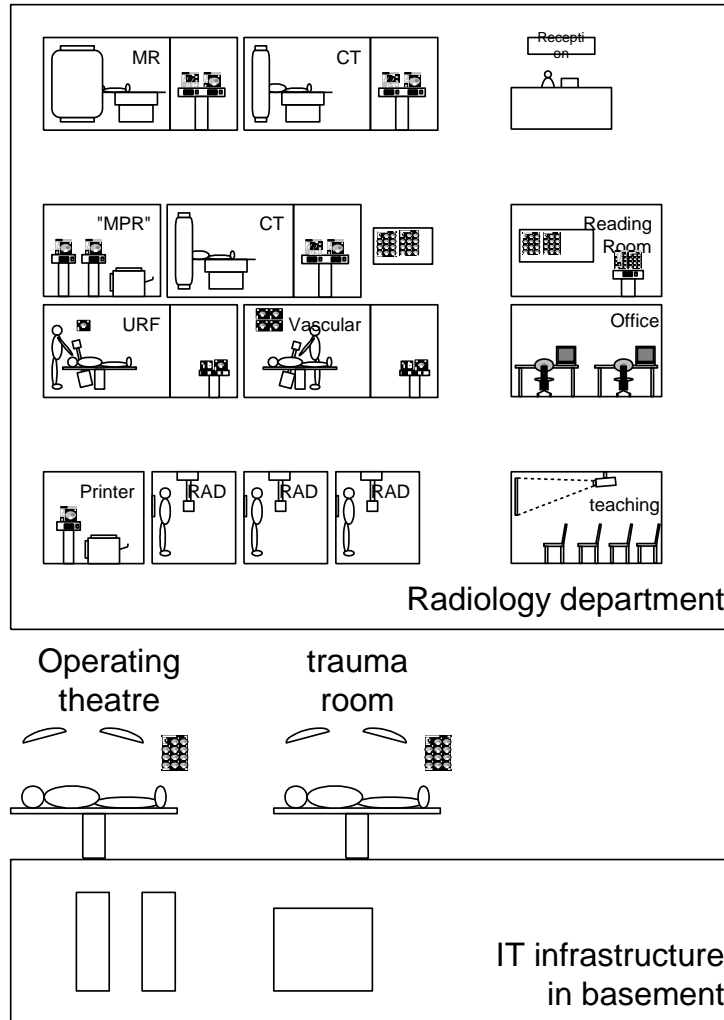
1. The (sub)system does not build.
2. The (sub)system does not function.
3. Interface errors.
4. The (sub)system is too slow.
5. Problems with the main performance parameter, such as image quality.
6. The (sub)system is not reliable.

# Solutions: Integration of Multiple Products



# Example 3: Integrated Clinical Solutions

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



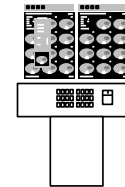
Note the similarity with Kongsberg Maritime's achievements with K-master and operator stations

Radiologist at home

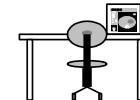


Radiologist somewhere in the hospital

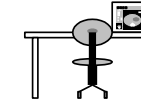
Radiologist at other hospital



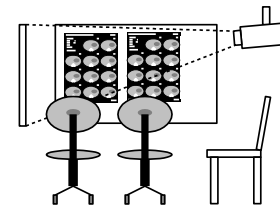
Referring Physician



Referring Physician



Conference room

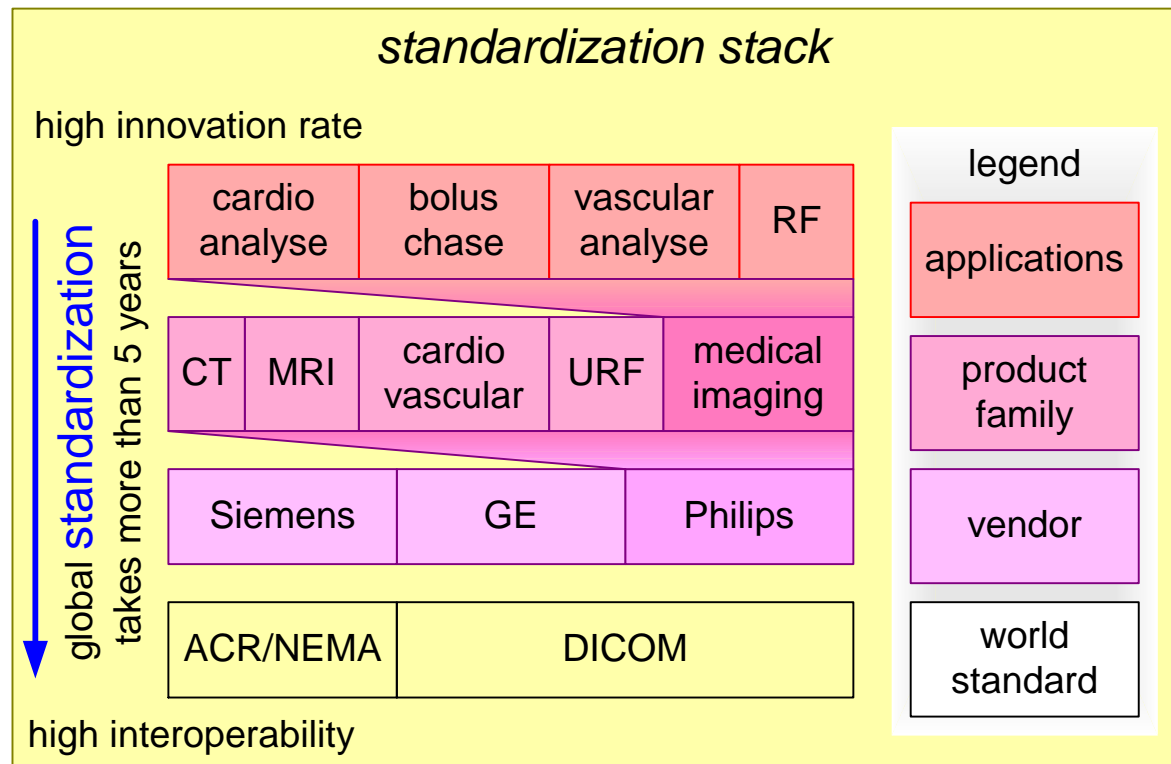


# The Information Model Swamp

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

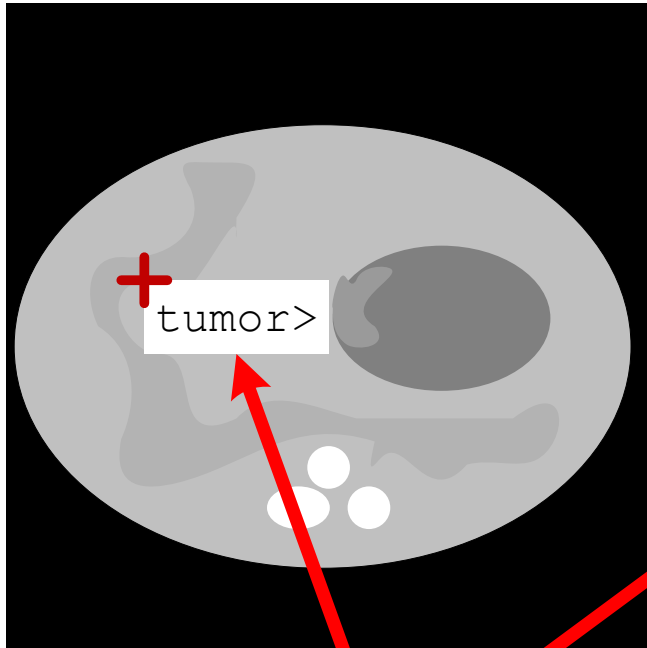
Convertors, wrappers, and adapters are nearly everywhere.

The cynical name of our product was *Shit Concentrator* since the integrating product has to resolve any inconsistency

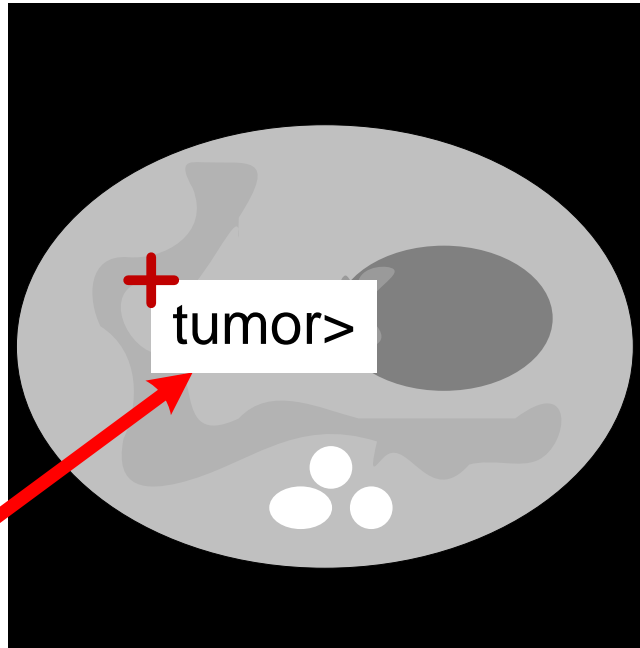


# Risks of “Near Identical” Data Models

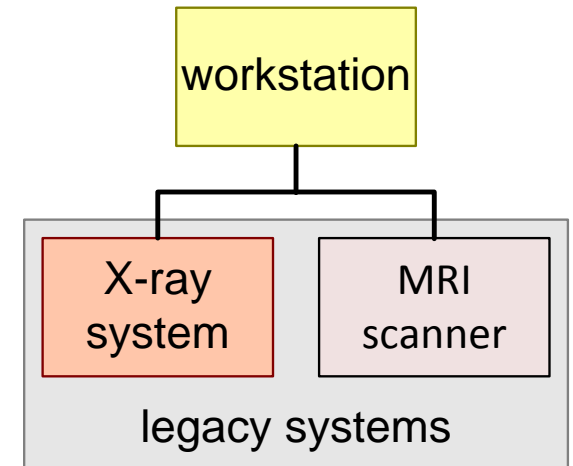
URF monitor output:  
fixed size letters at fixed grid



Workstation

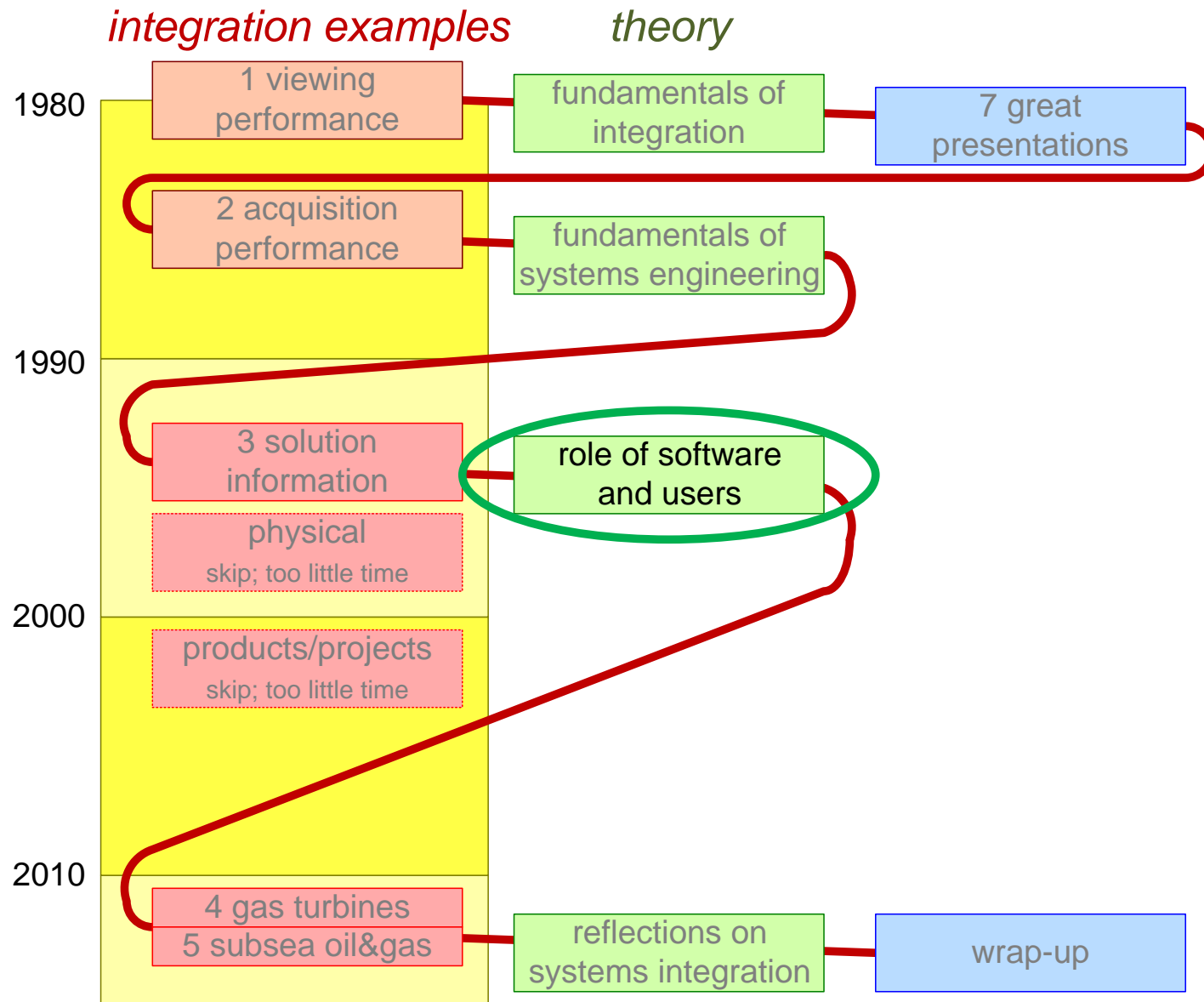


other rendering causing a dangerous mismatch between text and image

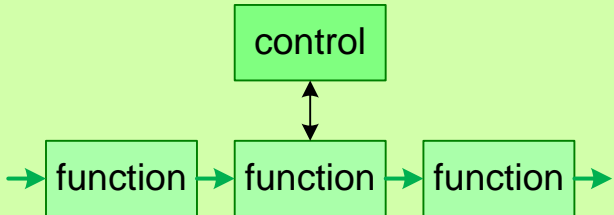
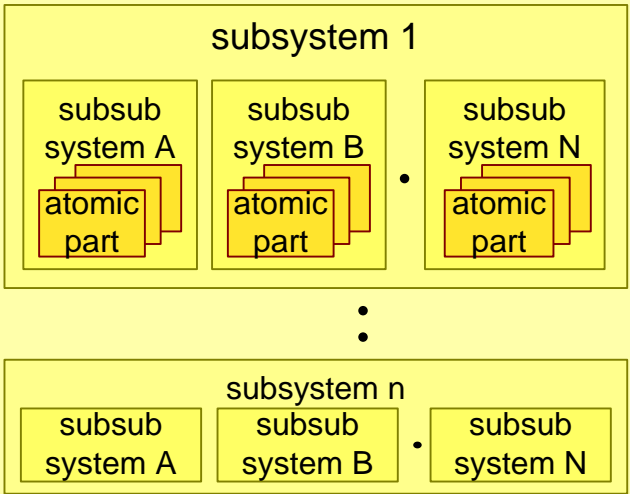


multiple near-identical data models with near-identical interpretations

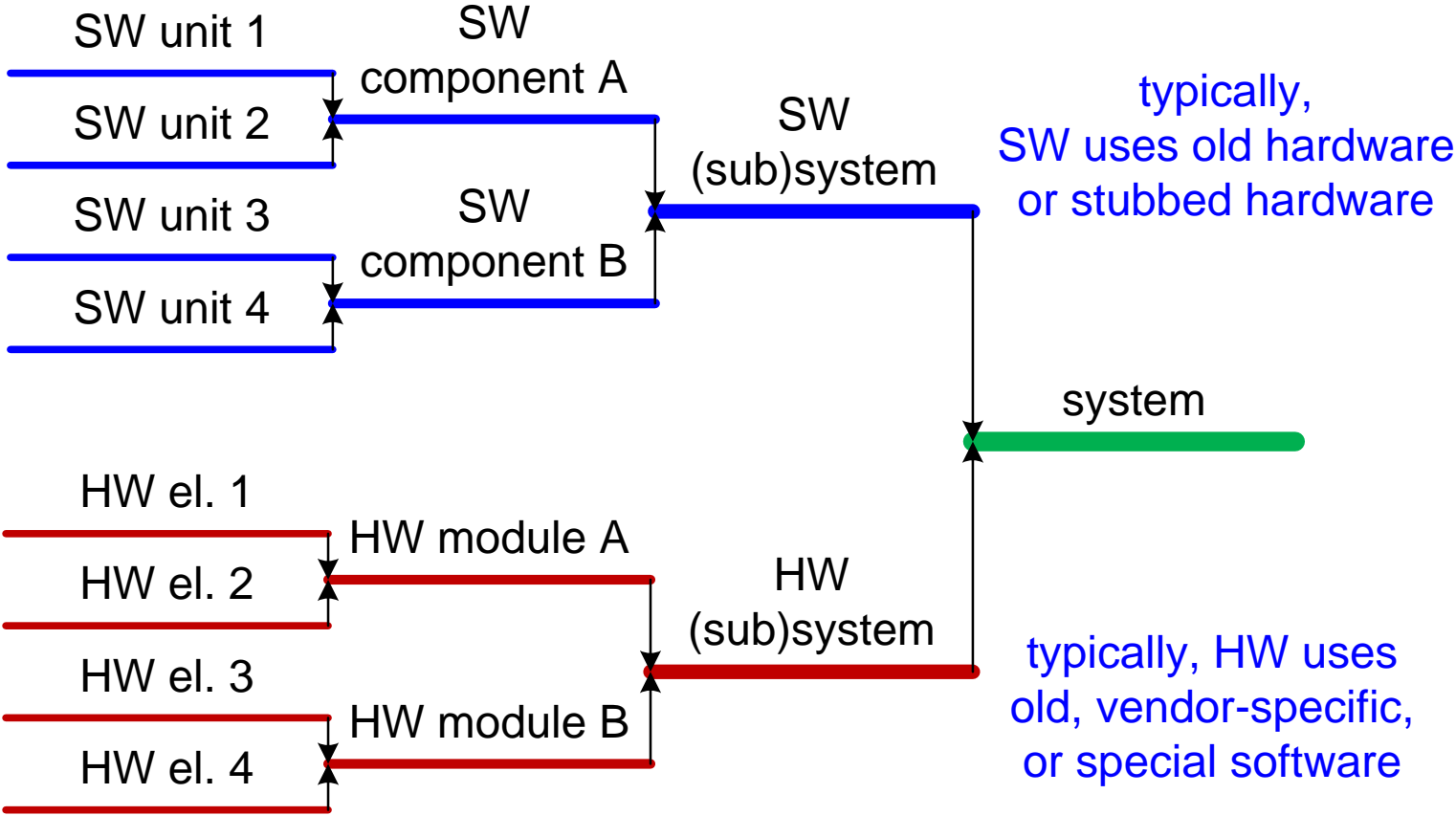
# Role of Software



# Software Characteristics and Role

<p><b>quantified properties</b>          productivity: 100.000 images per hour          speed: 100 frames/second          max latency: 50ms          max down time: 4 hrs/year</p>	<p>SW          determines and limits properties</p>
<p><b>dynamic behavior</b></p>  <pre>         graph TD             control[control] &lt;--&gt; f1[function]             f1 --&gt; f2[function]             f2 --&gt; f3[function]             style control fill:#90EE90             style f1 fill:#90EE90             style f2 fill:#90EE90             style f3 fill:#90EE90             </pre>	<p>SW          defines functionality and dynamic behavior          captures applications          conducts all technologies</p>
<p><b>parts</b>      <b>system</b></p>  <pre>         graph TD             subgraph system                 subgraph subsystem_1 [subsystem 1]                     subgraph subsub_A [subsub system A]                         direction TB                         ap_A1[atomic part]                         ap_A2[atomic part]                         ap_A3[atomic part]                     end                     subgraph subsub_B [subsub system B]                         direction TB                         ap_B1[atomic part]                         ap_B2[atomic part]                         ap_B3[atomic part]                     end                     subgraph subsub_N [subsub system N]                         direction TB                         ap_N1[atomic part]                         ap_N2[atomic part]                         ap_N3[atomic part]                     end                 end                 subgraph subsystem_n [subsystem n]                     subgraph subsub_A_n [subsub system A]                     end                     subgraph subsub_B_n [subsub system B]                     end                     subgraph subsub_N_n [subsub system N]                     end                 end             end             </pre>	<p>SW has its own partitioning          in e.g. components, units</p> <p>SW          has zero delivery time          production is costless          is ideal to solve last minute problems</p> <p>SW          is abstract and intangible          is alien to “physical” engineers</p>

# Hardware and Software Typically Meet at the End



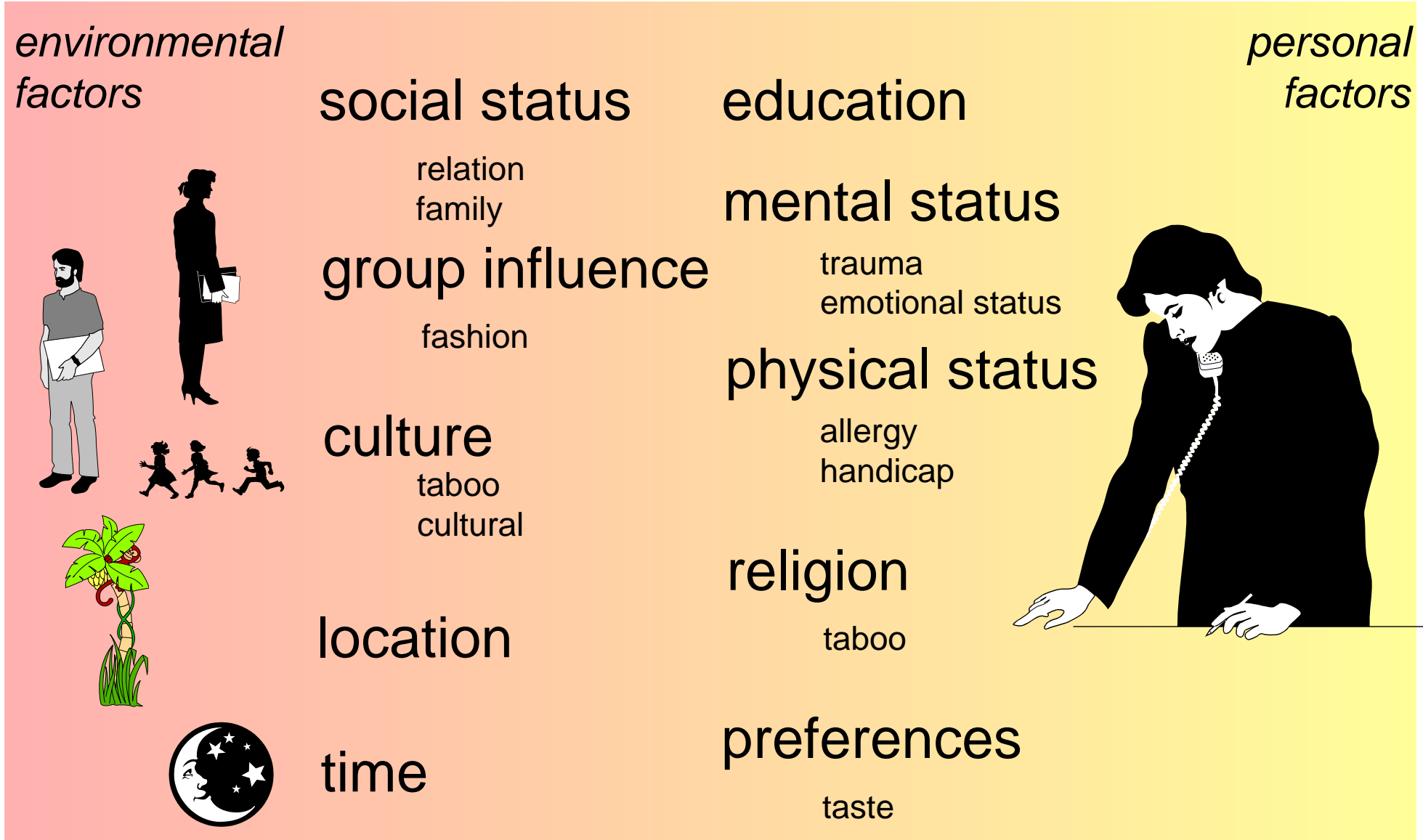
Segregation of hardware and software is a typical organizational problem.

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

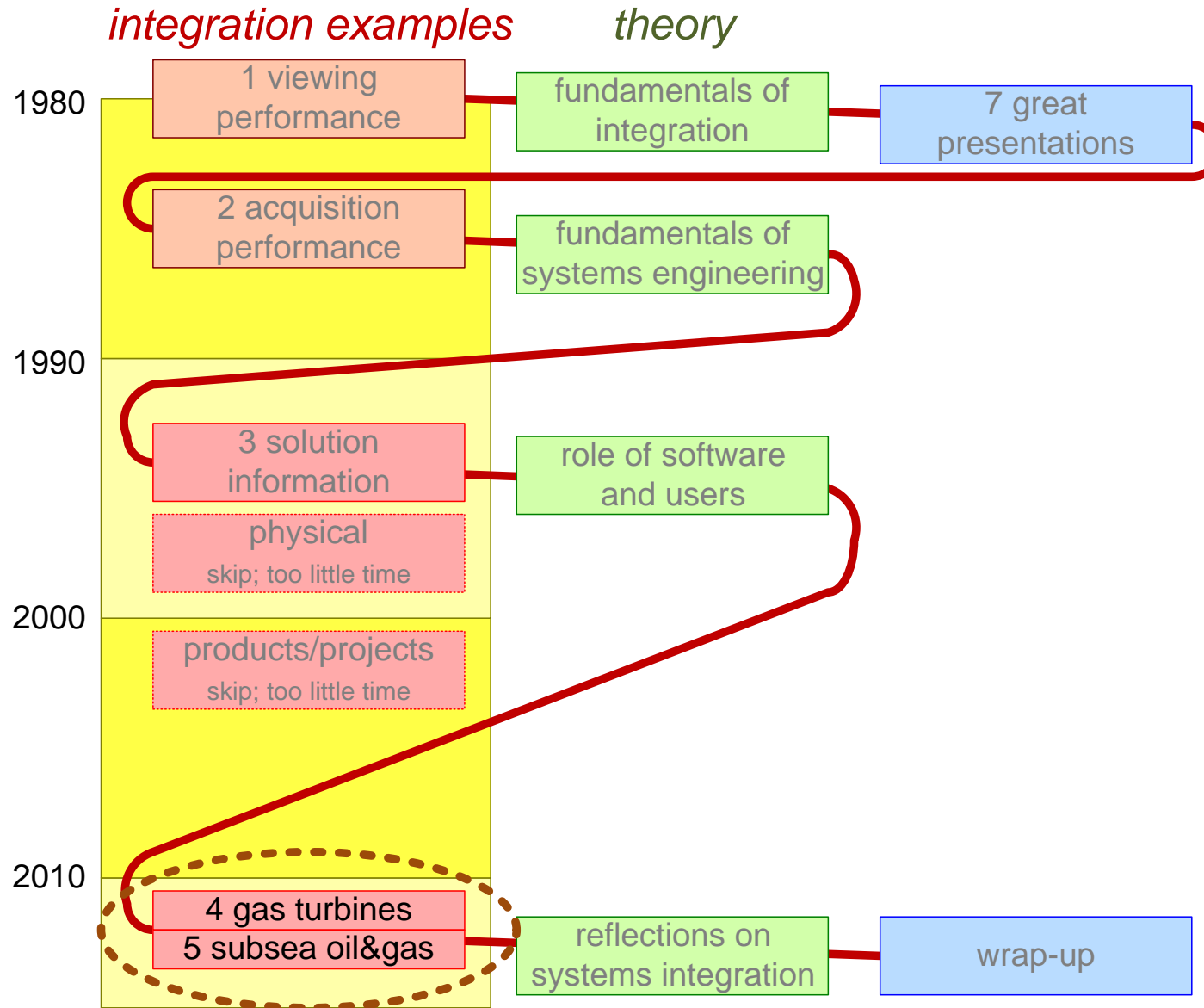


## Users:

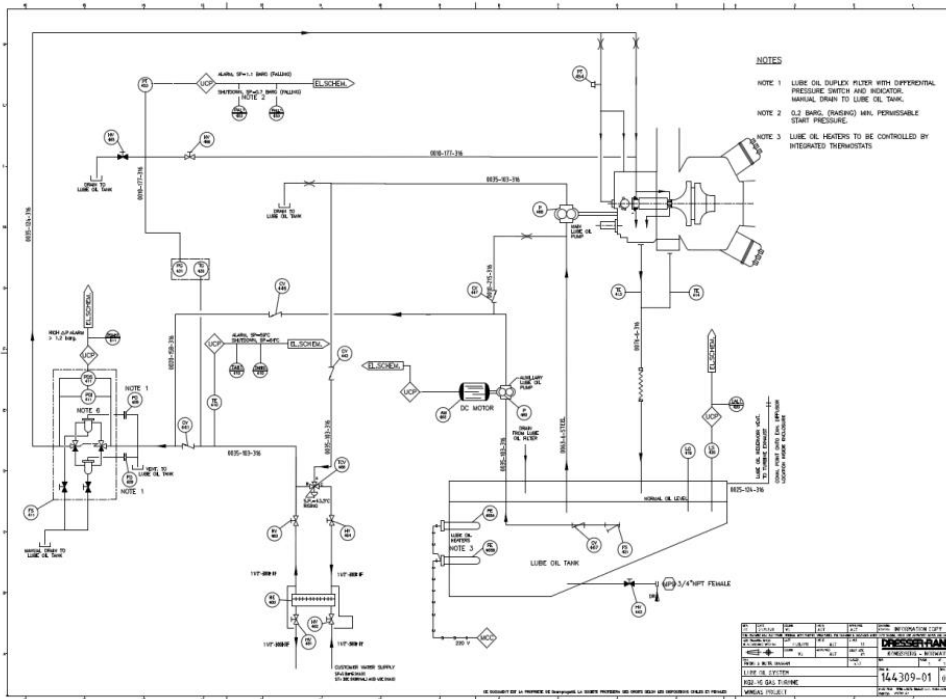
- are autonomous
- behave under influence of internal and external drivers
- are creative
- “solve” problems
- have limited knowledge of the system
- have limited insight in their impact on the system

Users do the unexpected

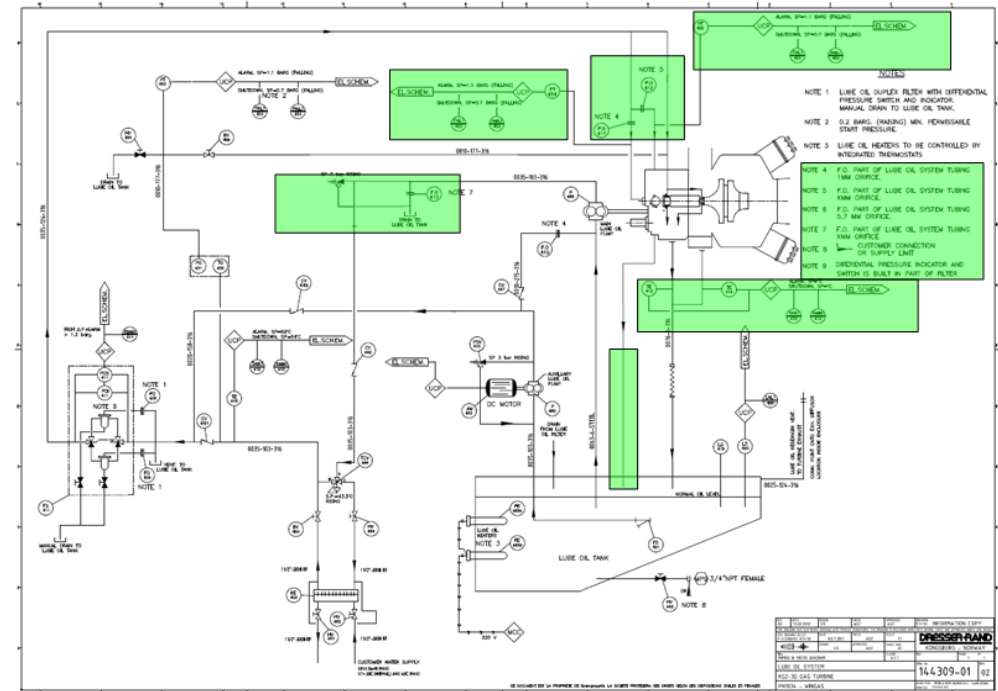
# Today in Kongsberg



# Errors Found after Functional Analysis and Quantification



PID before

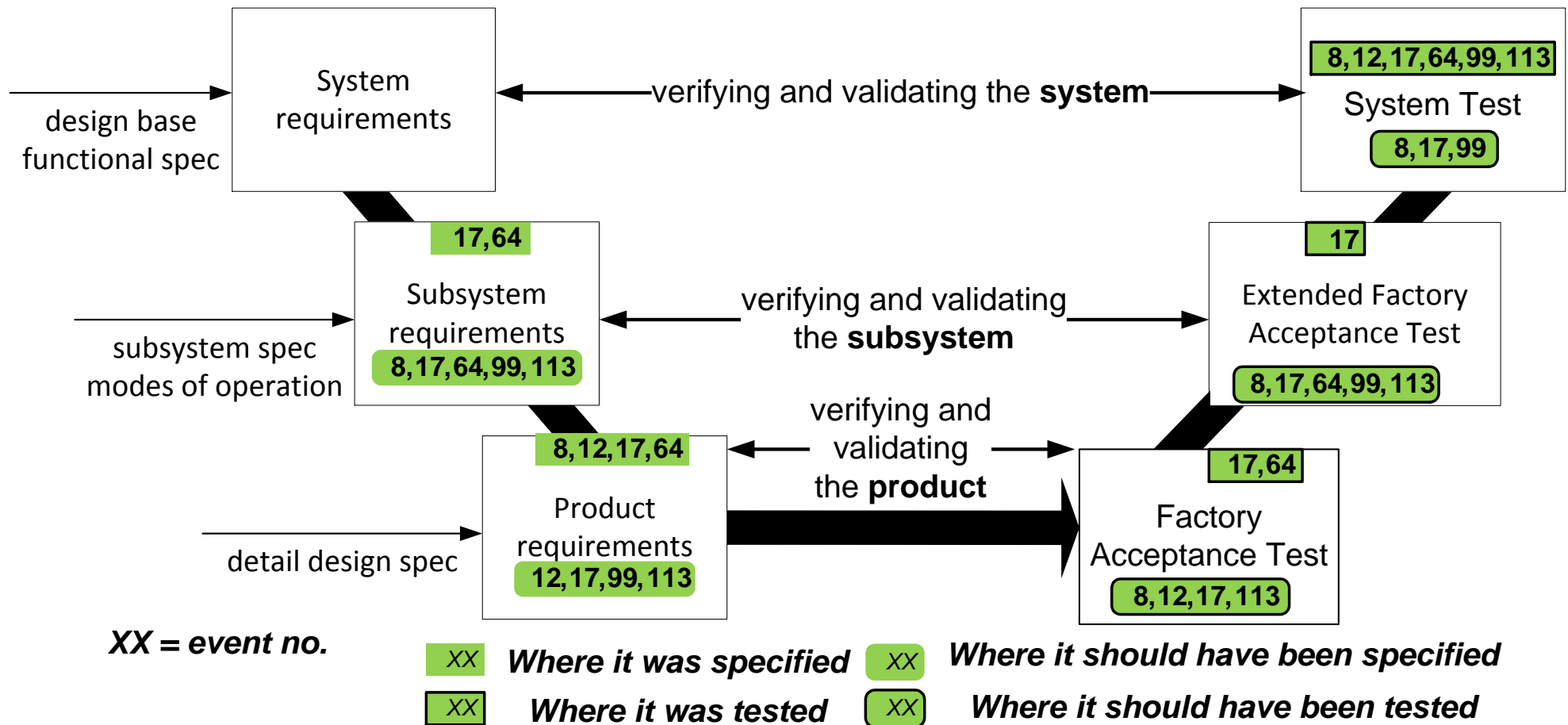


PID after

changed due to functional analysis and quantification

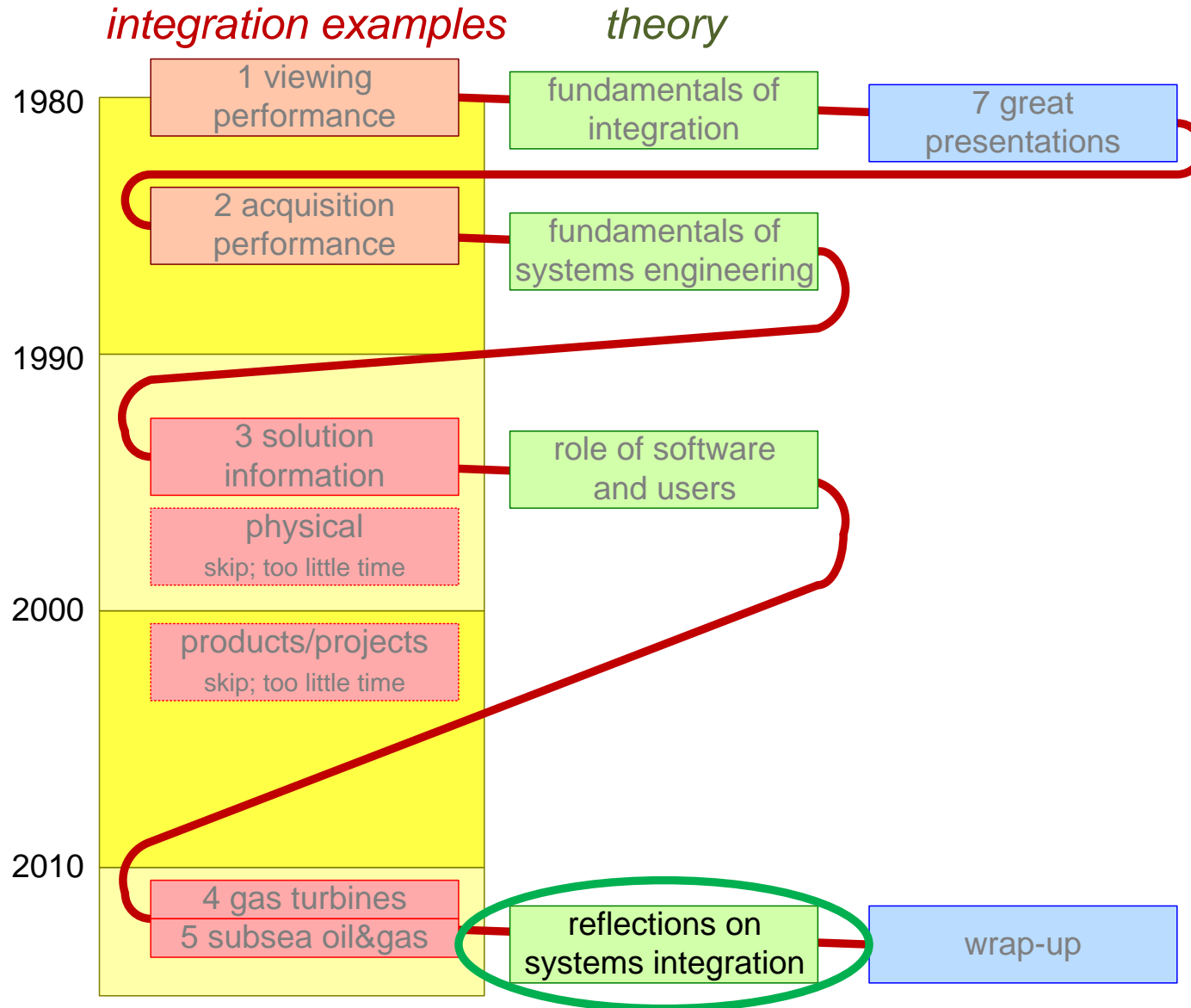
from Knowledge Capture, Cross Boundary Communication and Early Validation with Dynamic A3 Architectures by Vickram Singh [http://www.gaudisite.nl/INCOSE2013\\_Singh\\_Muller\\_DynamicA3.pdf](http://www.gaudisite.nl/INCOSE2013_Singh_Muller_DynamicA3.pdf)

# Analysis of Subsea System Test



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

# Reflections on Systems Integration



# Imperfect Processes

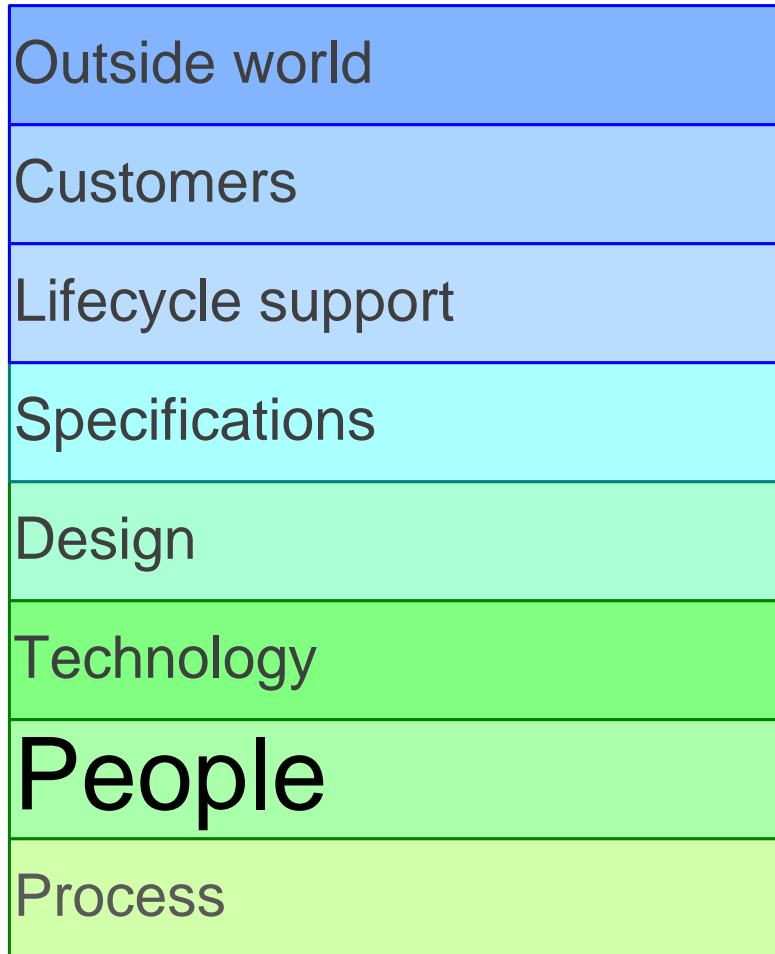
---



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

# Imperfect People

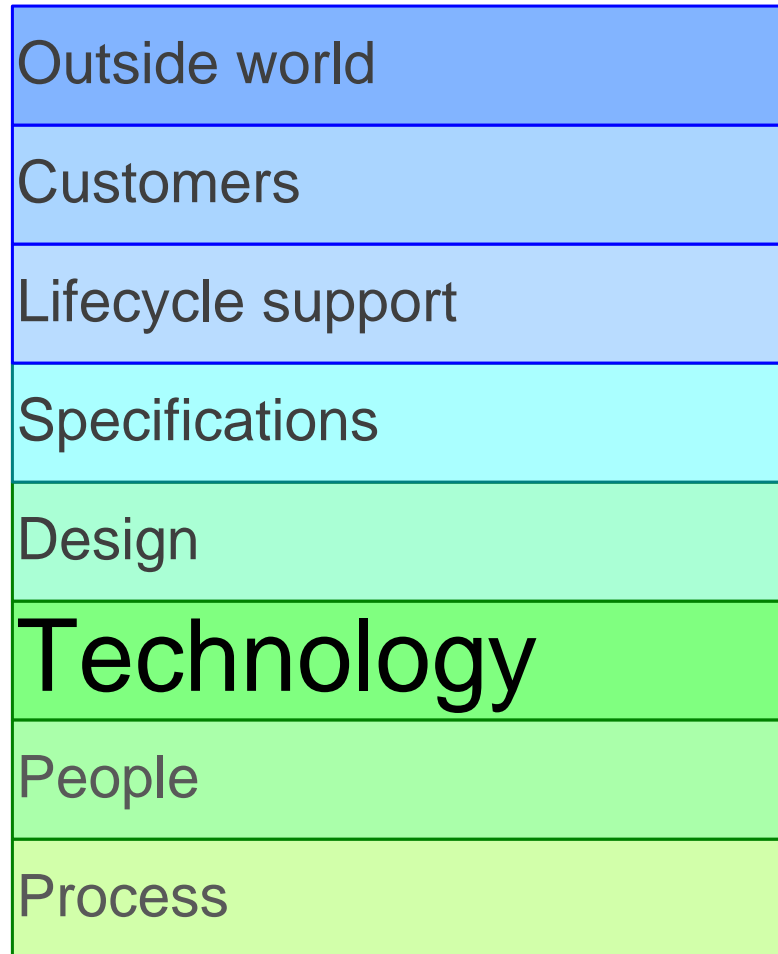
---



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

# Imperfect Technology

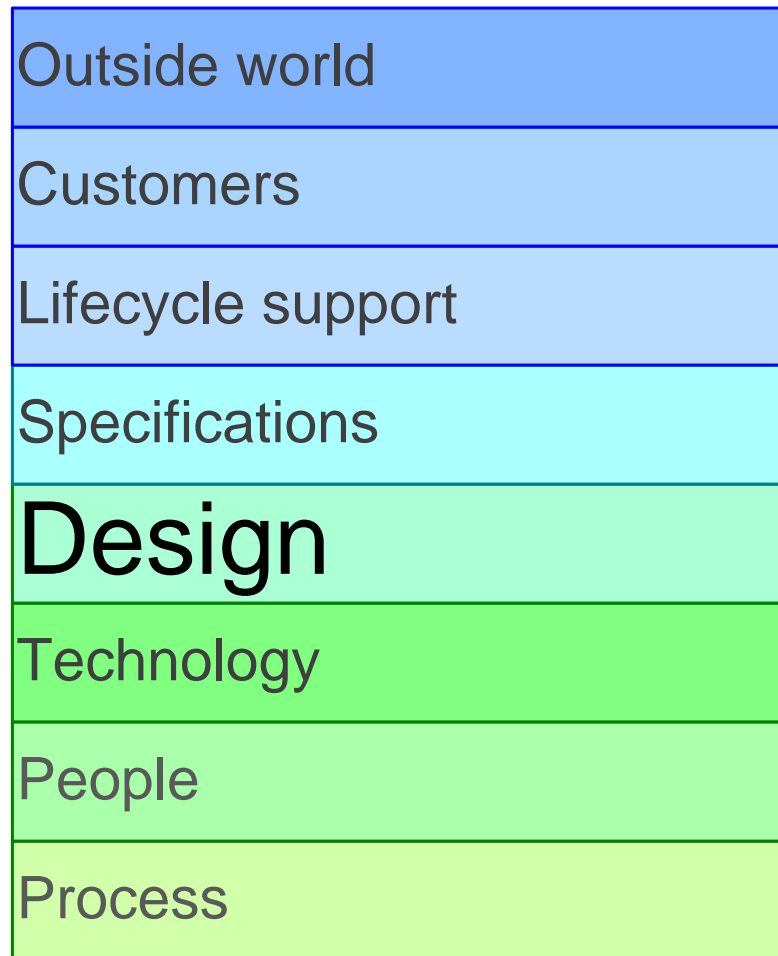
---



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

# Imperfect Design

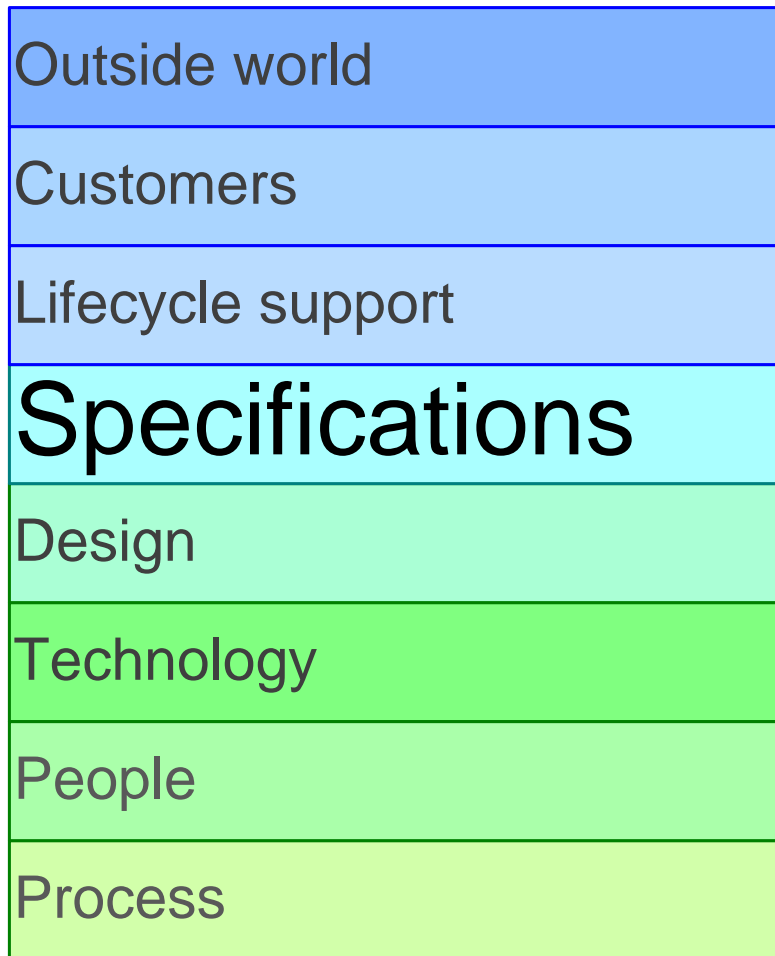
---



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

# Imperfect Specifications

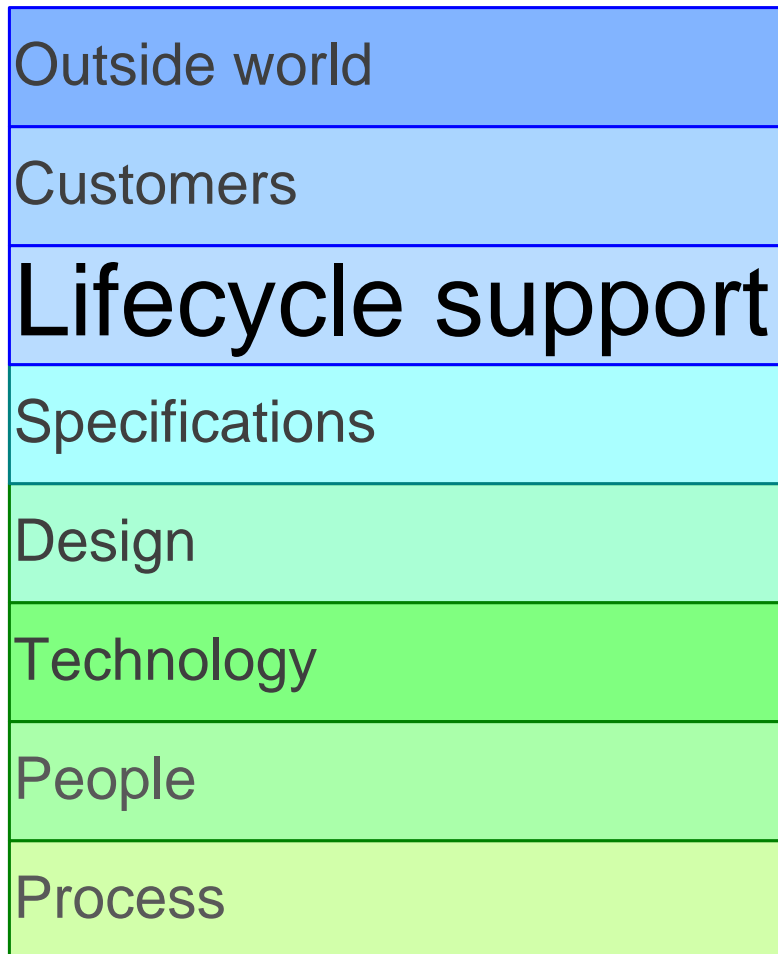
---



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

# Imperfect Lifecycle 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

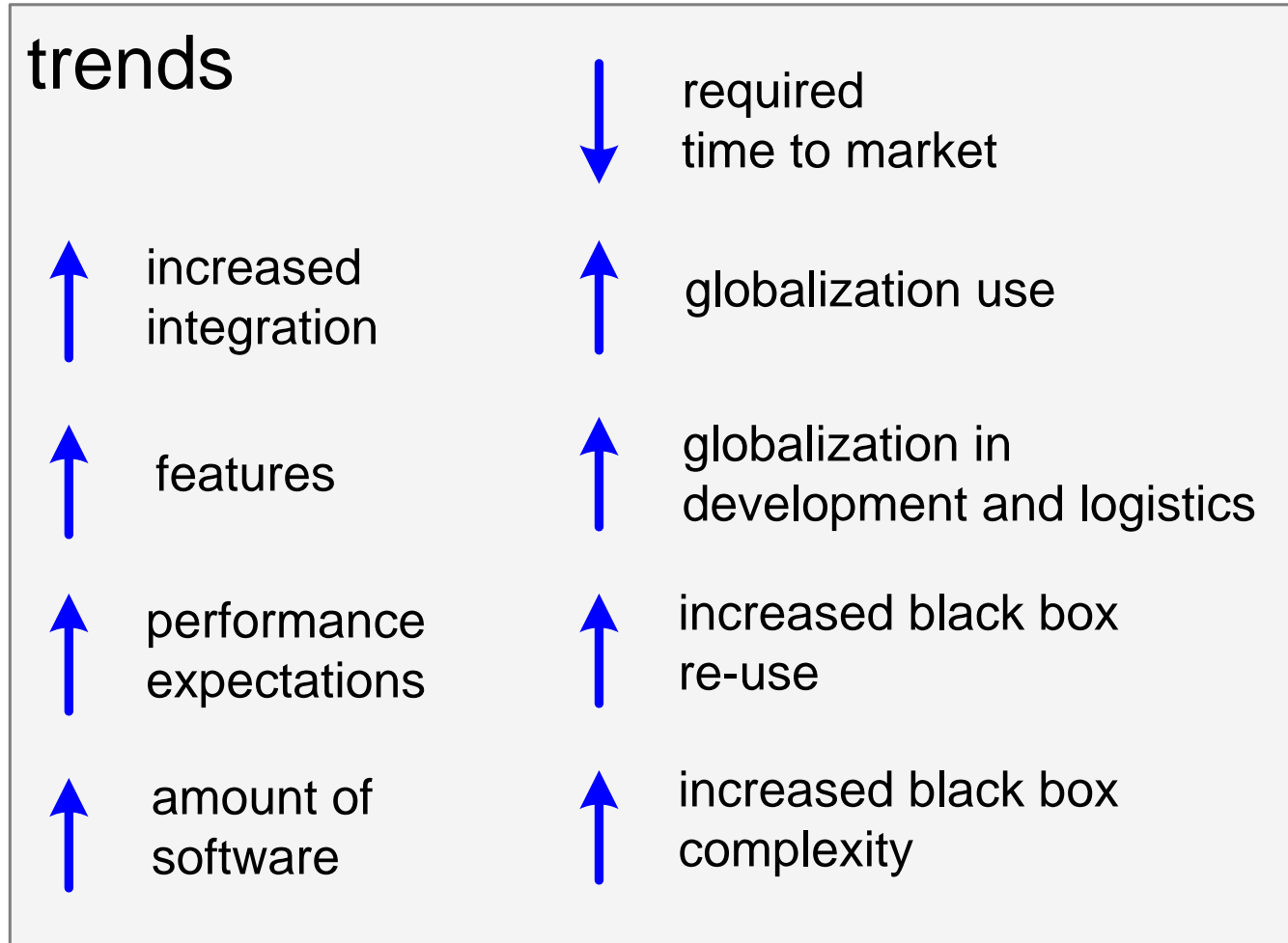
Technology

People

Process

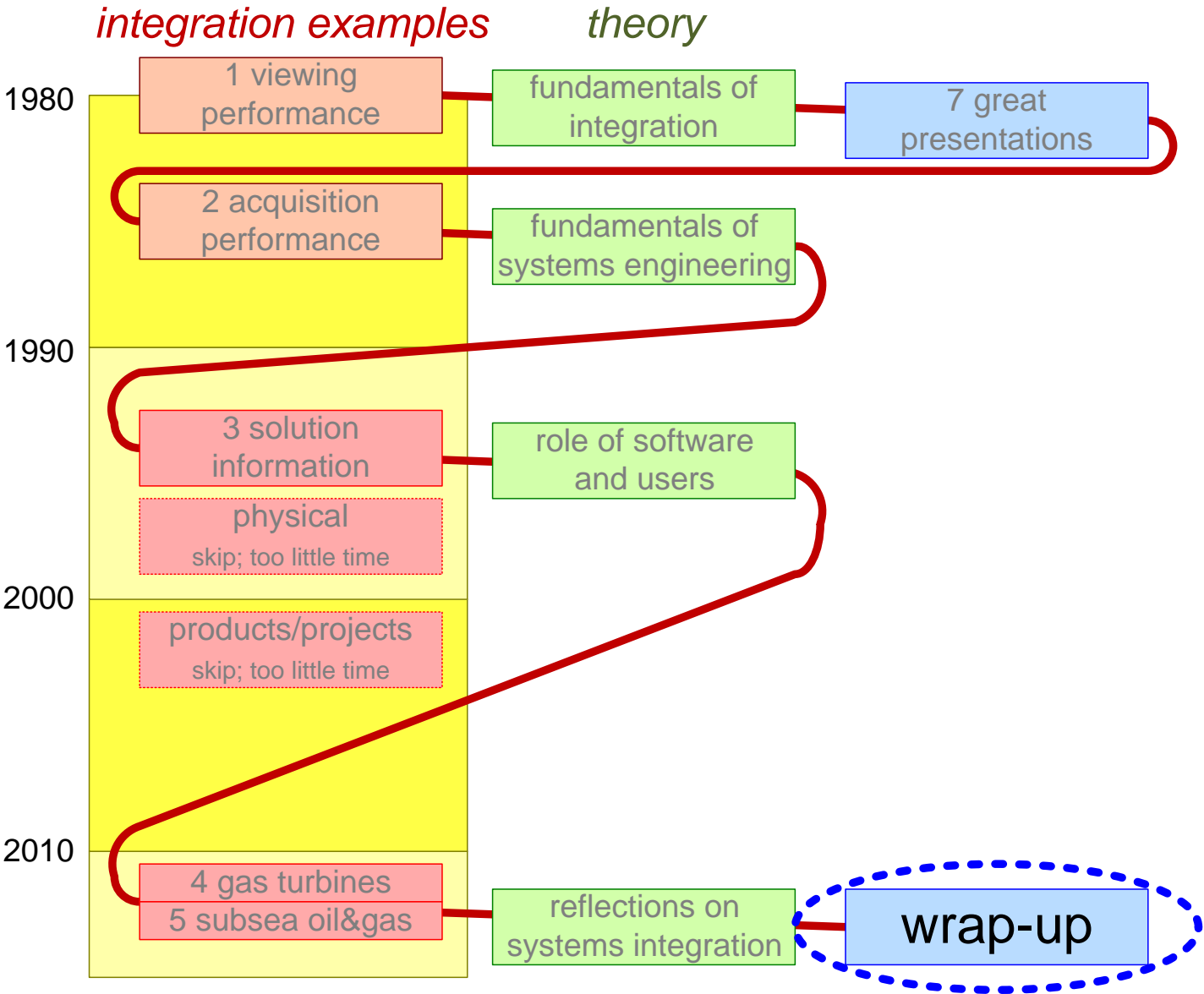
- social complexity (humans)
- natural complexity
- interaction between natural and artificial world

# Without Measures it only gets Worse...



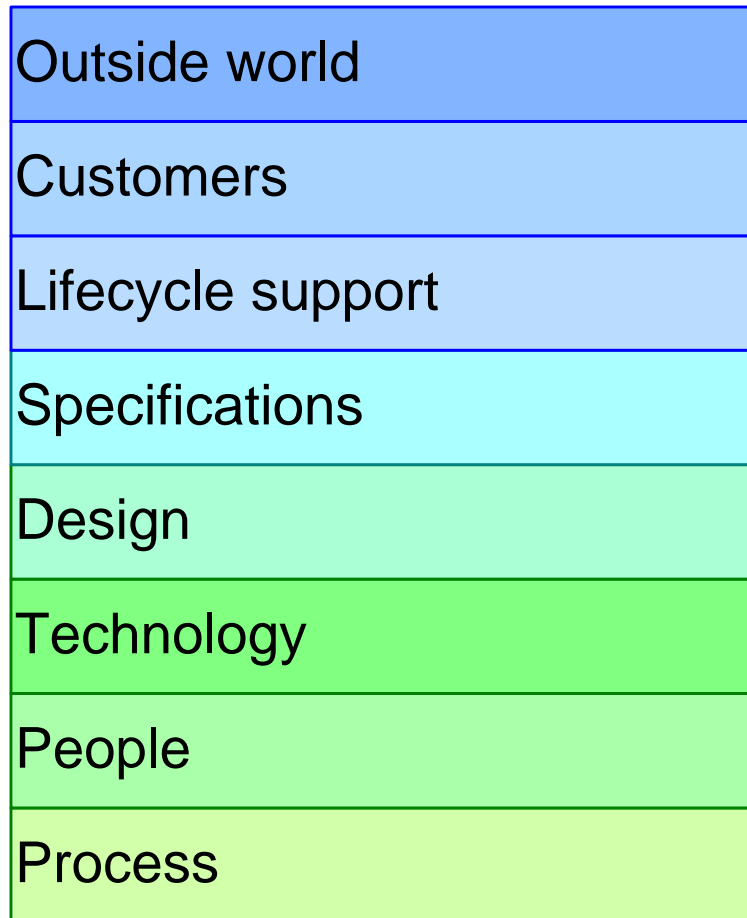
↑ integration surprises

# 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

↑ increased integration

↑ features

↑ performance expectations

↑ amount of software

↓ required time to market

↑ globalization use

↑ globalization in development and logistics

↑ increased black box re-use

↑ increased black box complexity

↑ integration surprises

***Improve System Development:***

*modeling, analysis, tools*

*process, people*

***Focus on Systems Engineering***