The Difficult Relation Between System and Software

by *Gerrit Muller* USN-SE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

Abstract

Today's systems depend heavily on software and data. Despite the central role that software and data play in realizing the system behavior and performance, many organizations suffer from a difficult relationship between systems engineers and software and data engineers. What is causing this difficult relationship?

June 15, 2023
status: draft
version: 0.1

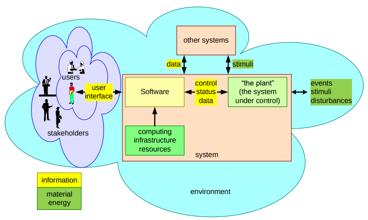
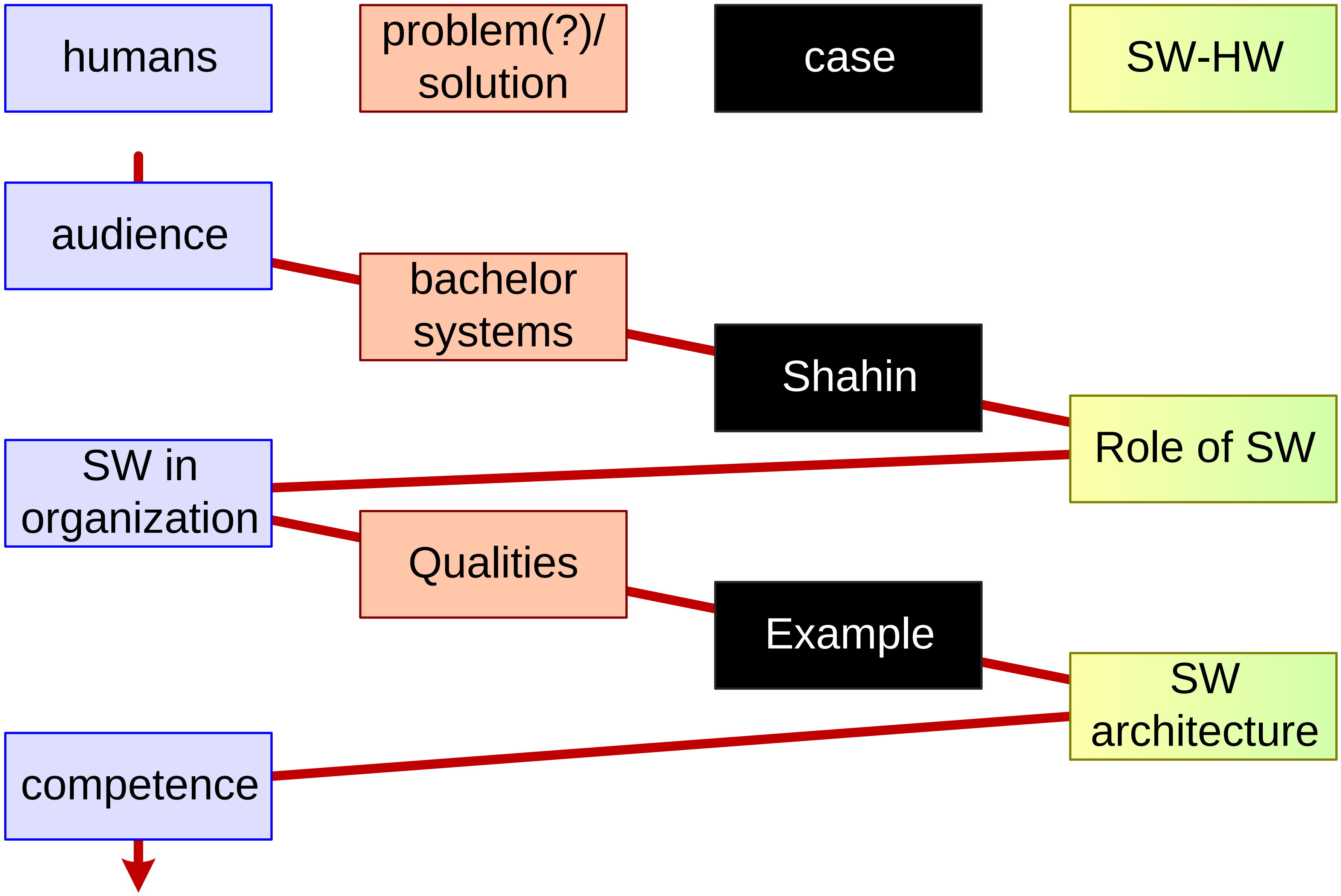


Figure of Content



Who is present at this KSEE?

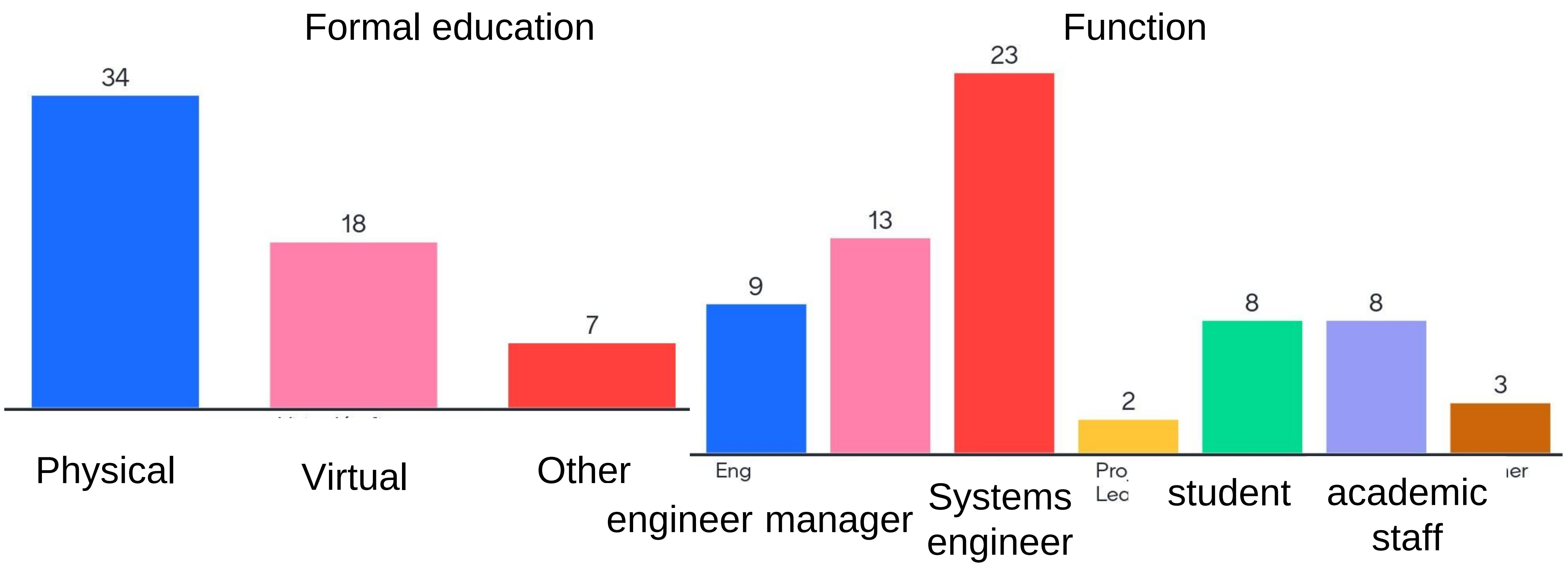
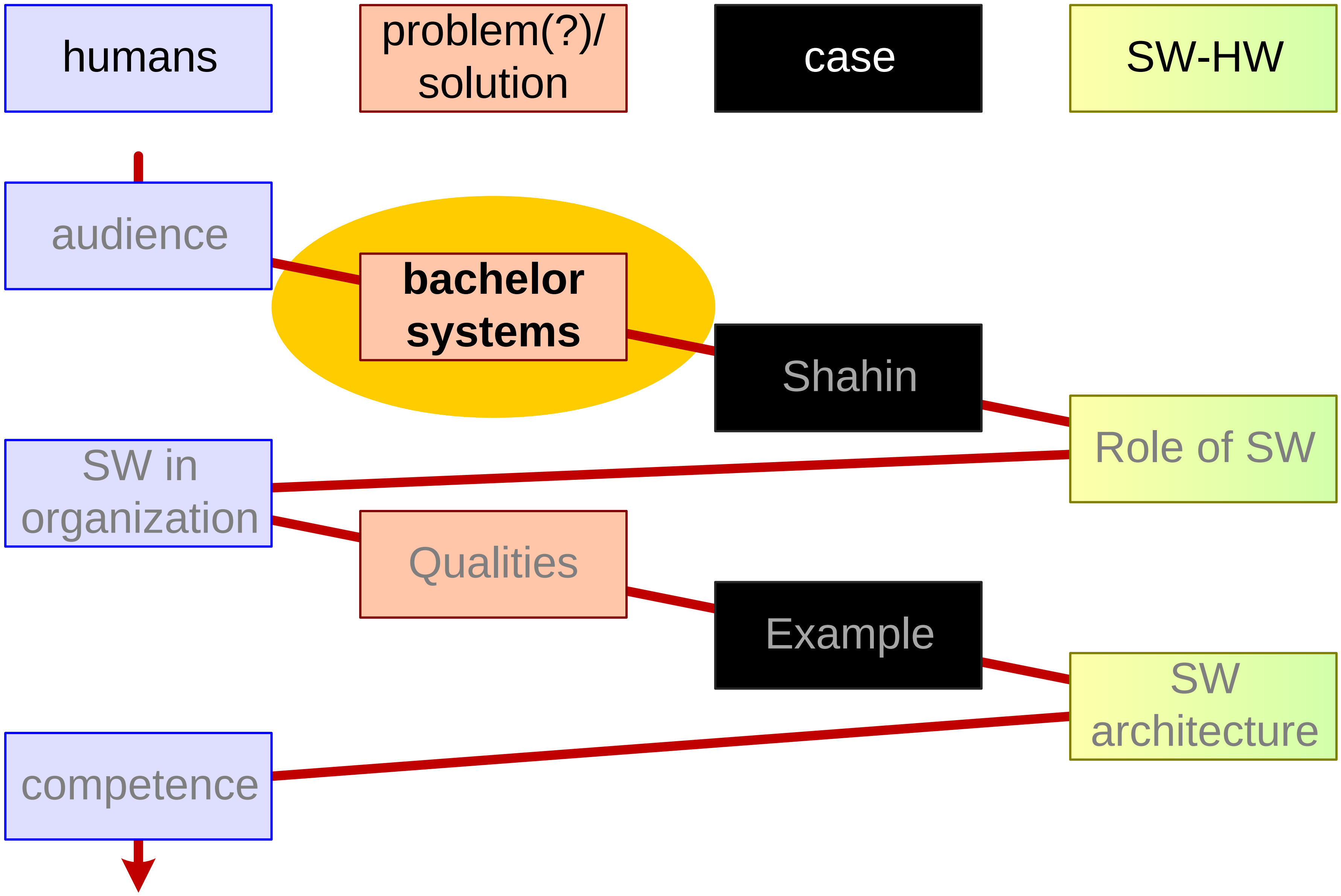
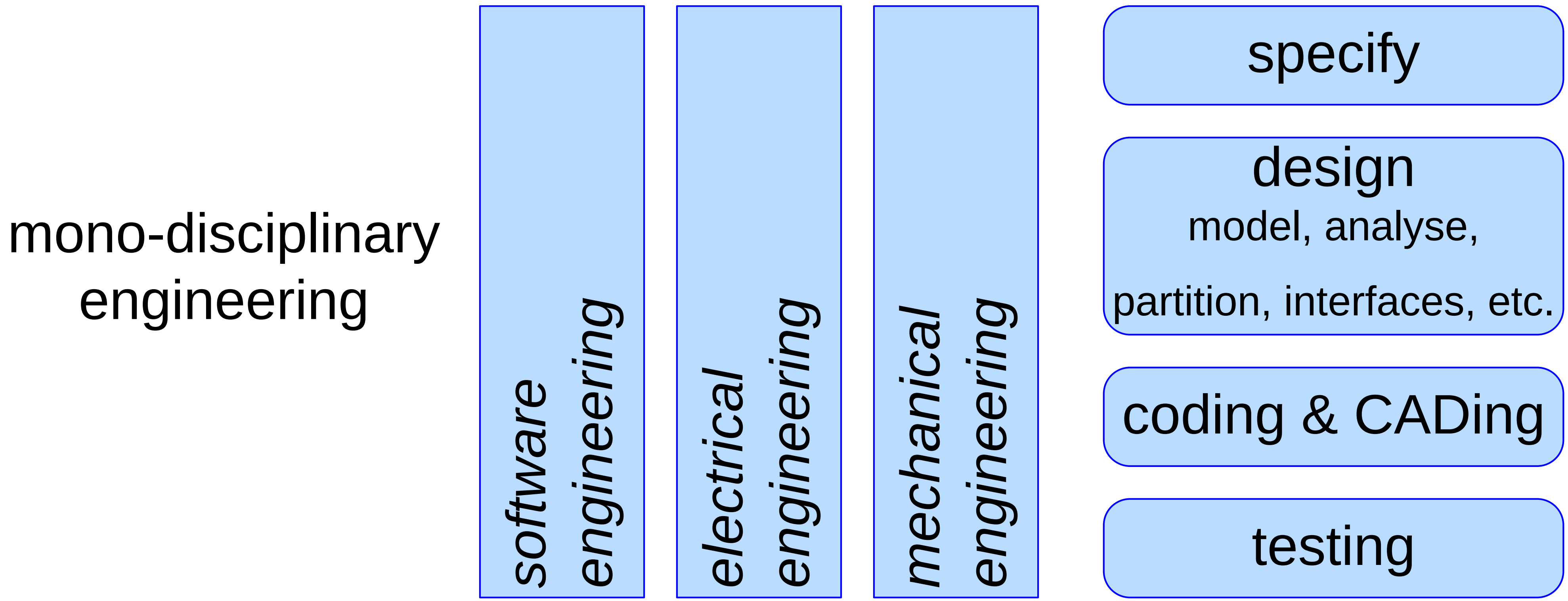


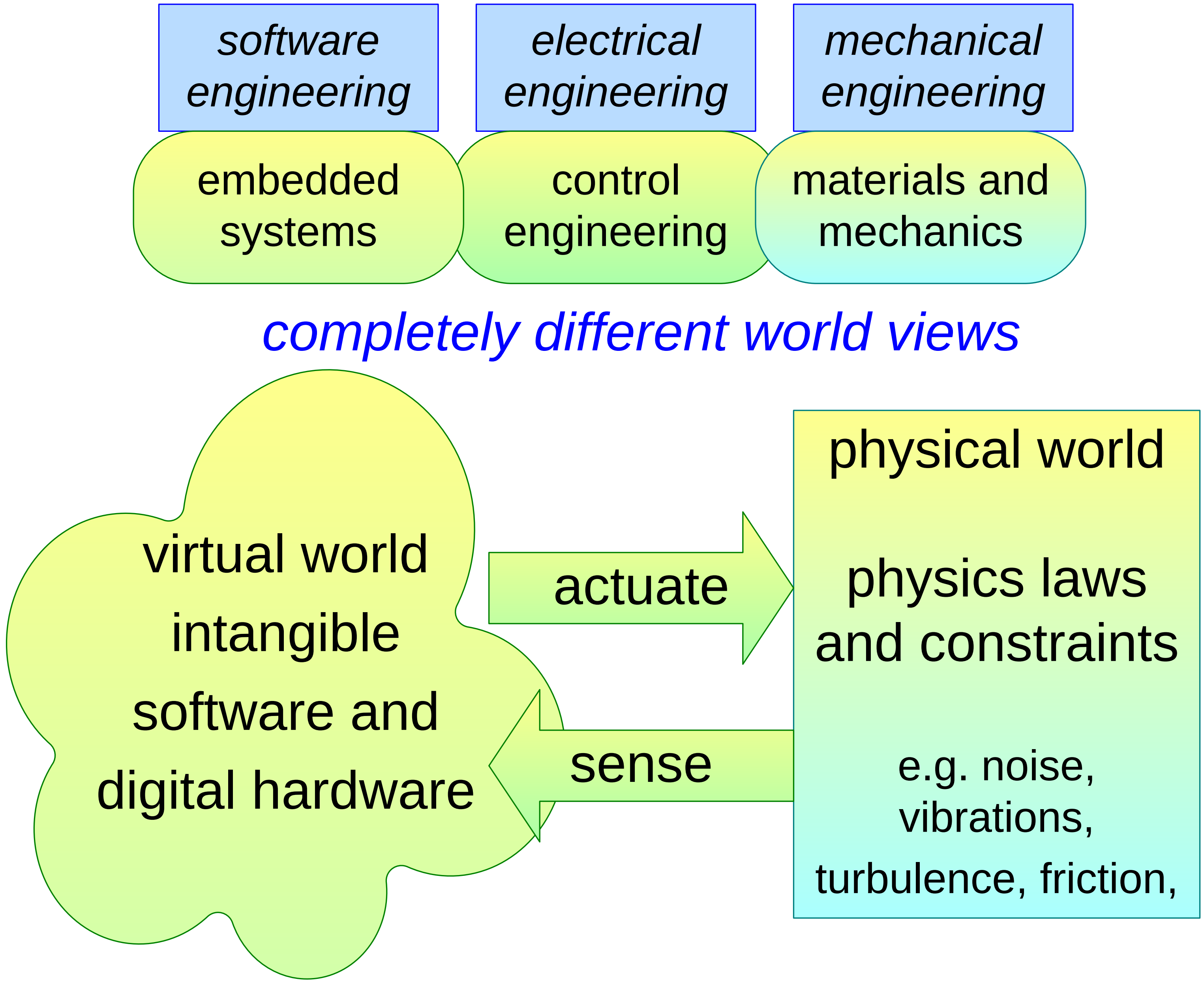
Figure of Content



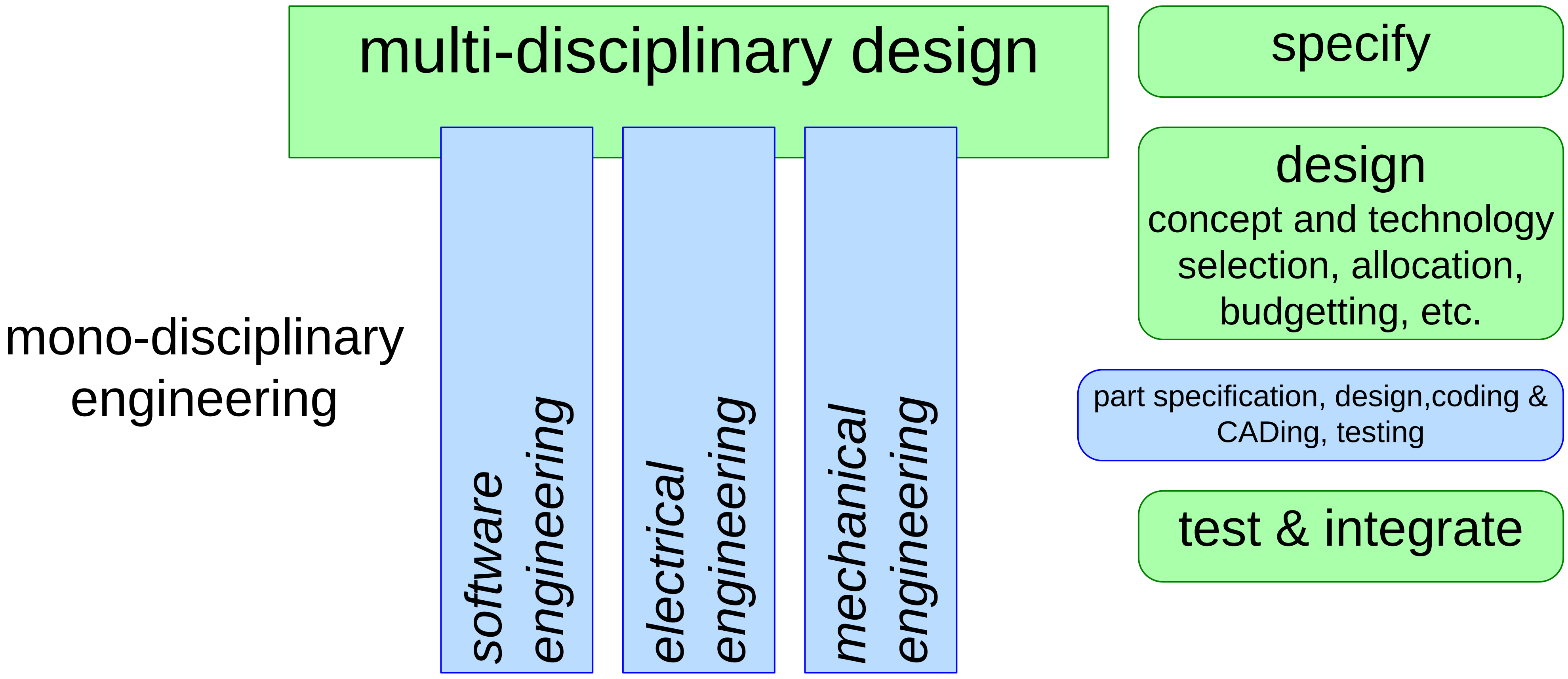
You study mono-disciplinary engineering



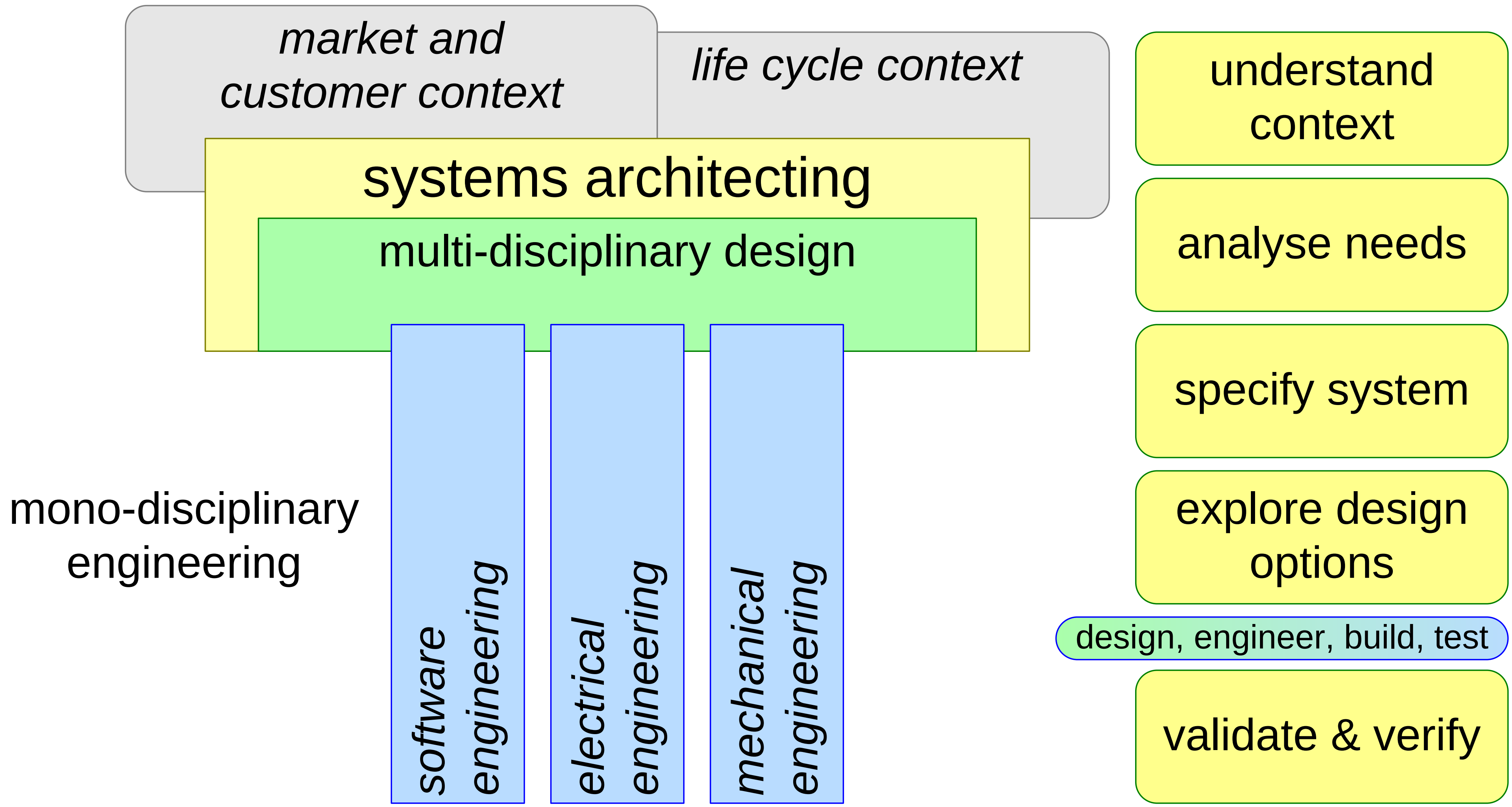
Huge differences in language and way of thinking



Multi-disciplinary design and engineering



Architecting: Fit-For-Purpose



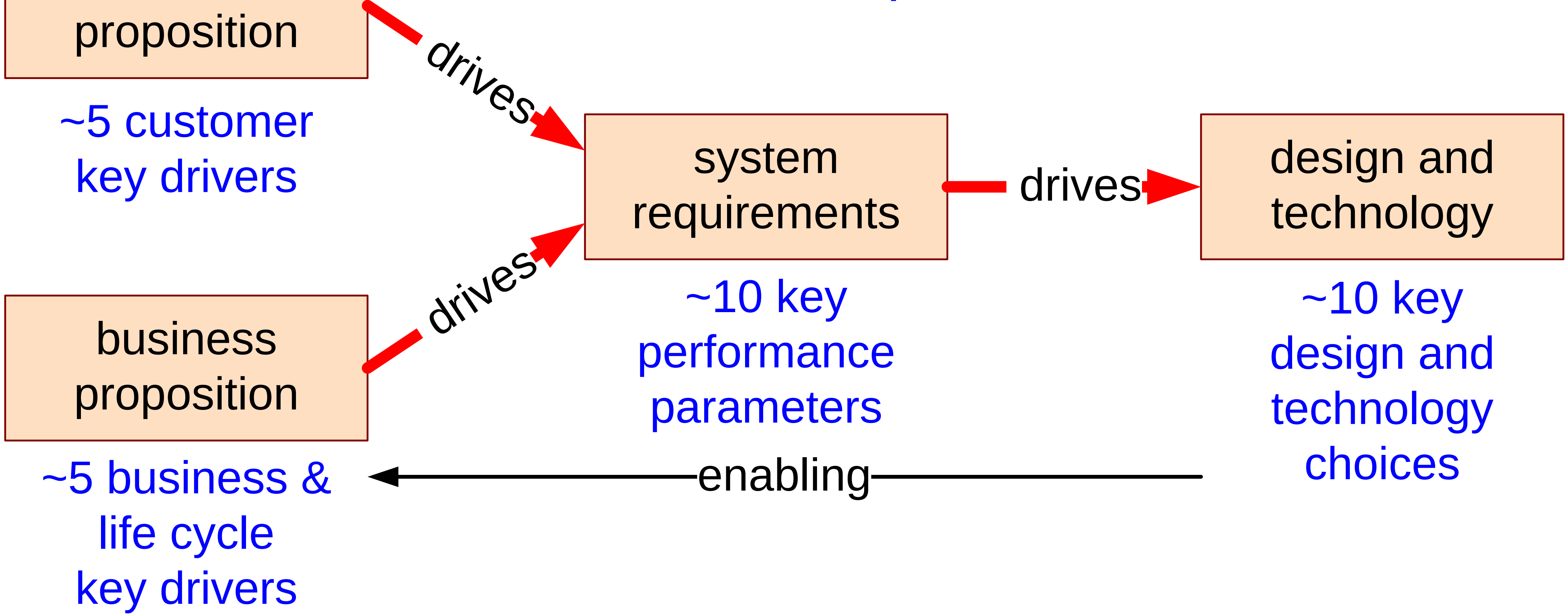
The Single Slide Summary of Systems Engineering

Systems Engineering: ***Fitness-For-Purpose***

Achieving **customer** and **business key drivers**

via **key performance parameters** of system

based on **technical expertise**

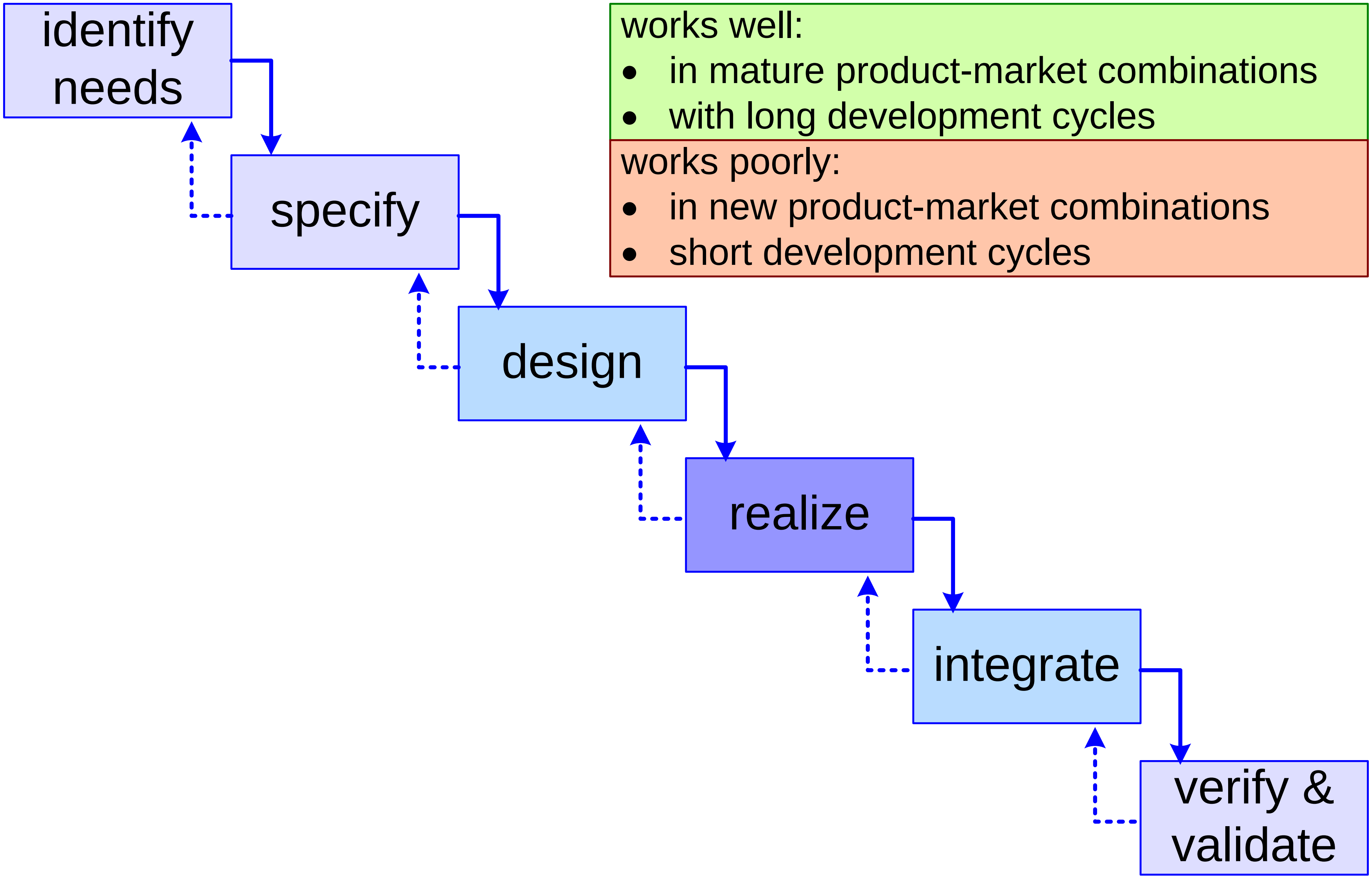


Why Chaotic?

Why so chaotic?

Why not follow top-down SE process?

Waterfall model



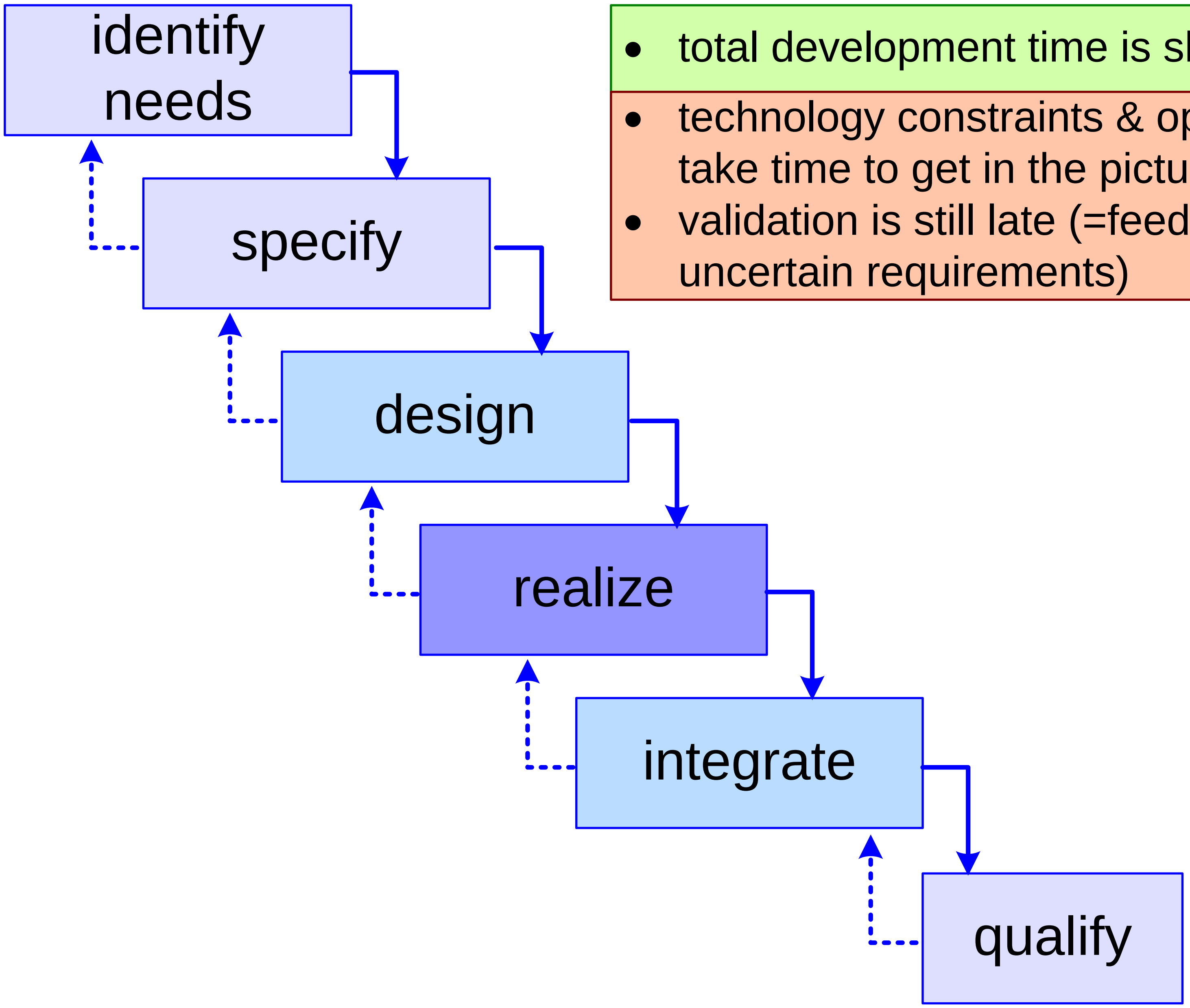
works well:

- in mature product-market combinations
- with long development cycles

works poorly:

- in new product-market combinations
- short development cycles

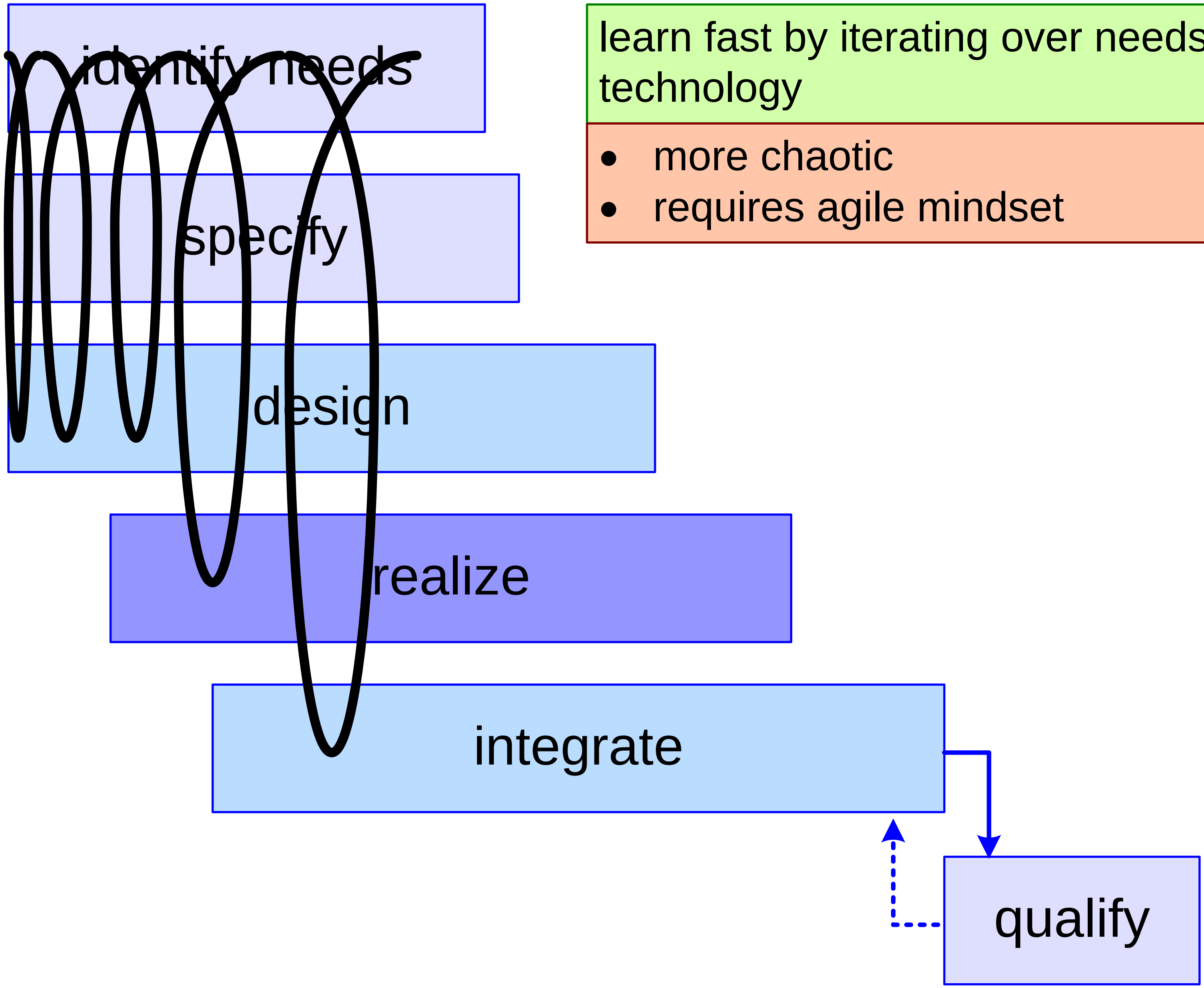
Concurrent Engineering



- total development time is shorter

- technology constraints & opportunities take time to get in the picture
- validation is still late (=feedback on uncertain requirements)

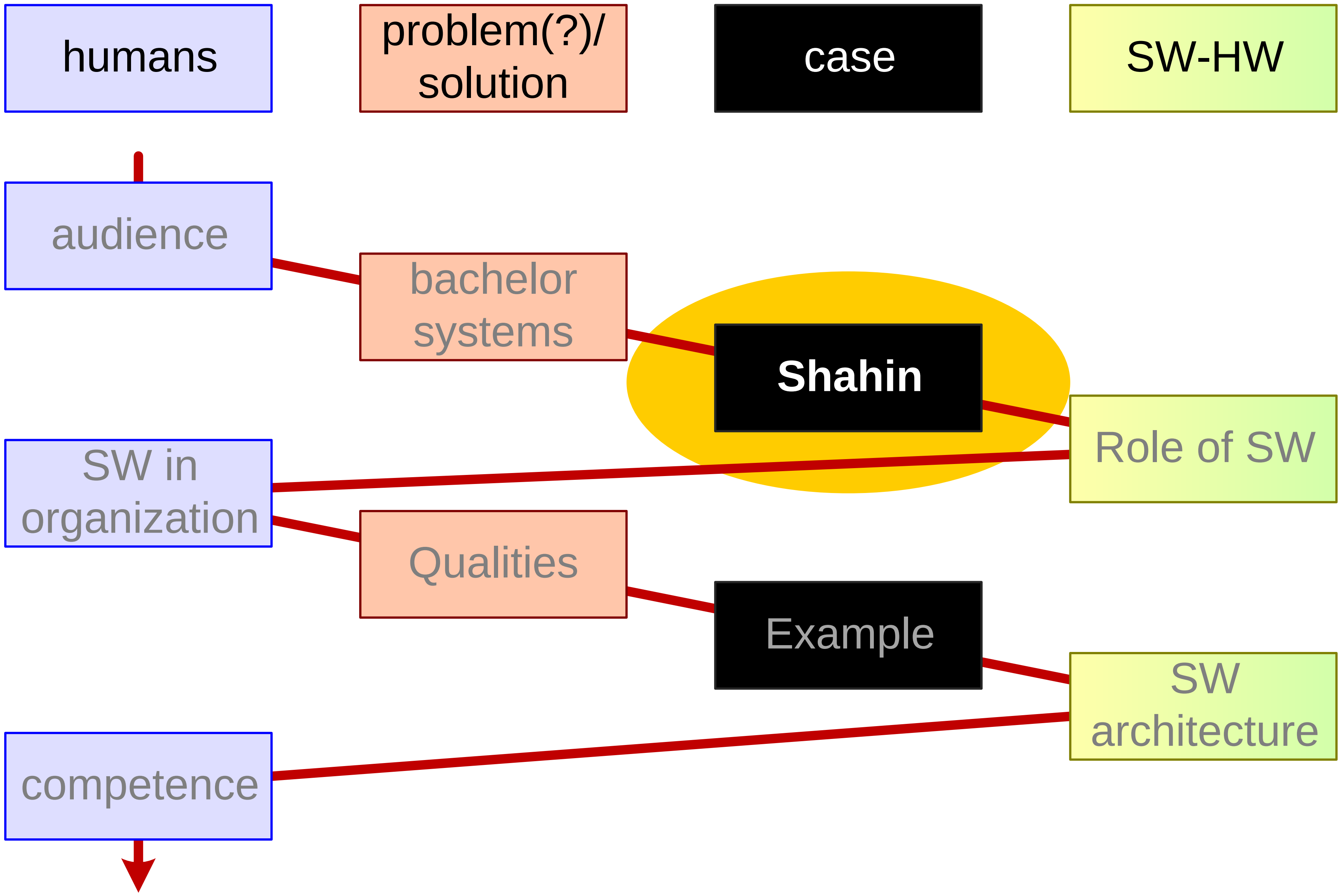
Iterative Approach



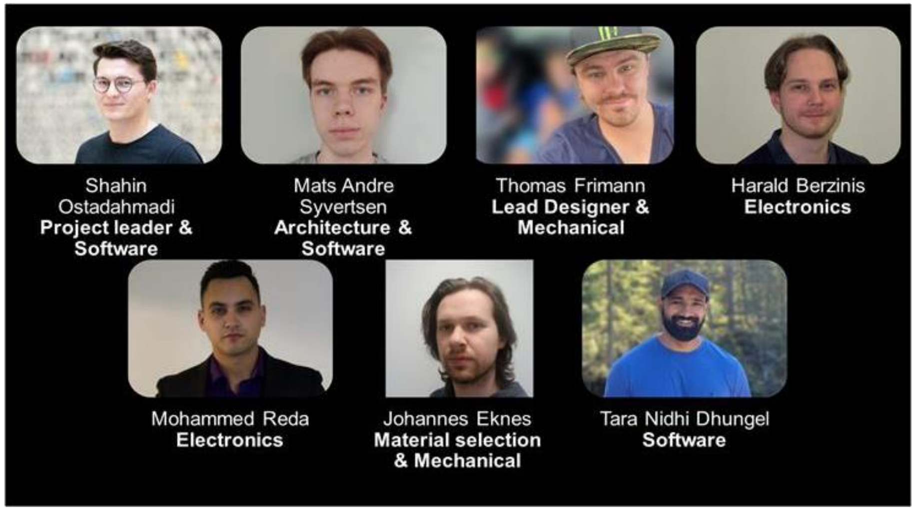
learn fast by iterating over needs and technology

- more chaotic
- requires agile mindset

Figure of Content



Case for Illustration: Fictive and Borrowed

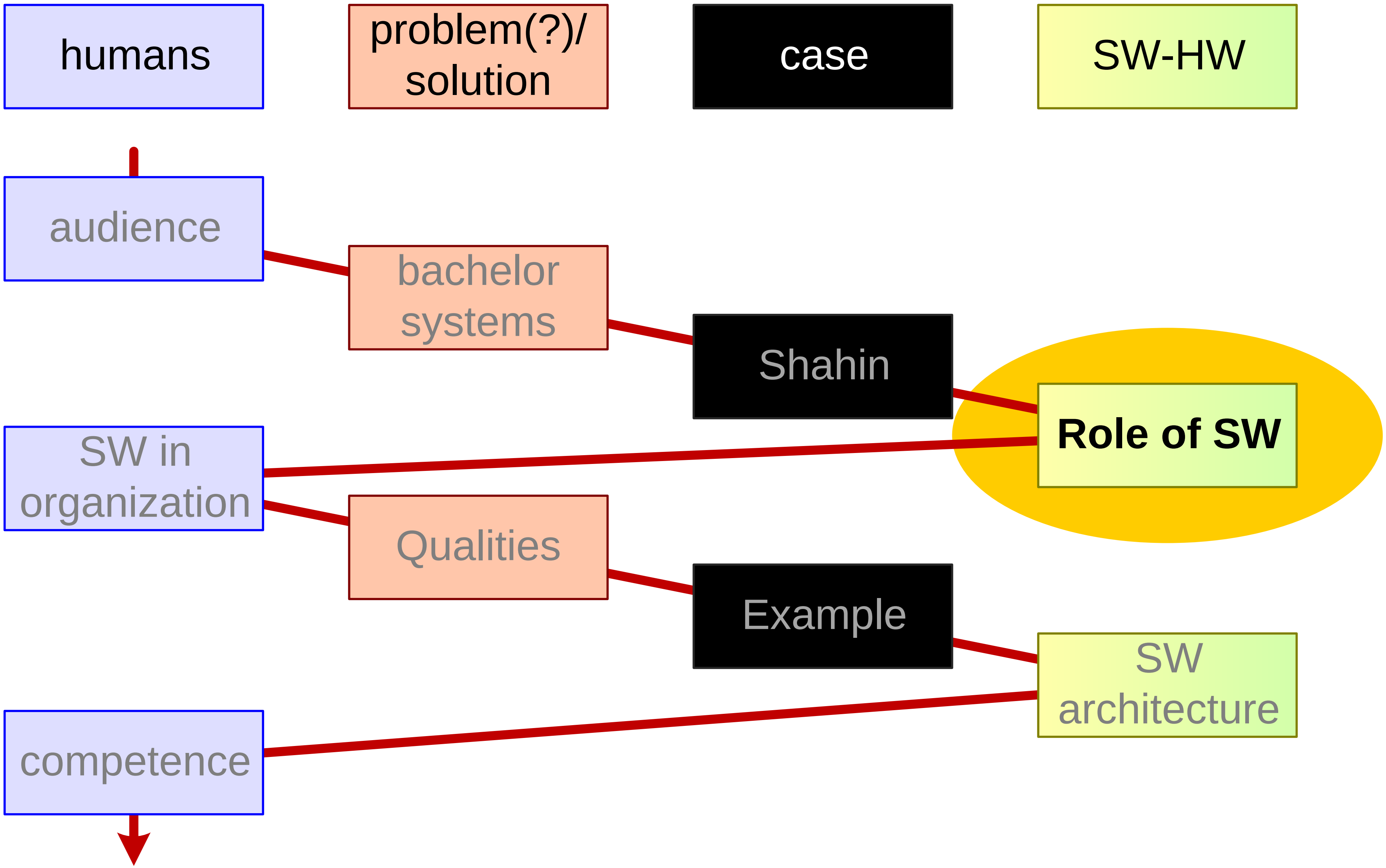


Drone Interception System for Festival Environments

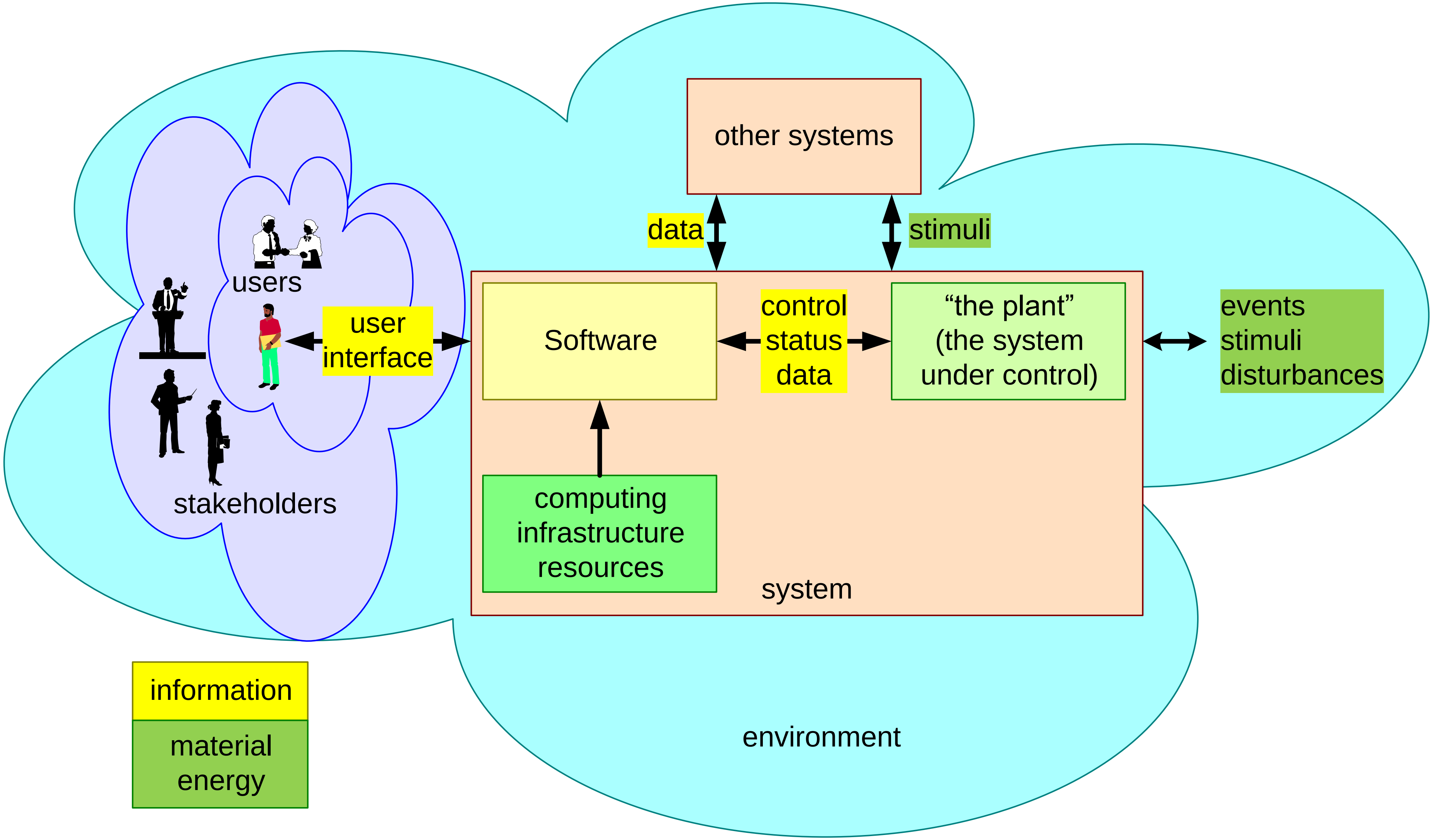
Team 4, Shahin
2023 Course Systems design and Engineering

Specify and design
a **Drone Interception System**.
The goal is to prevent drone misuse, e.g. paparazzi's, terrorist attacks, etc.
You may determine your own scope, such as clients, location, and application

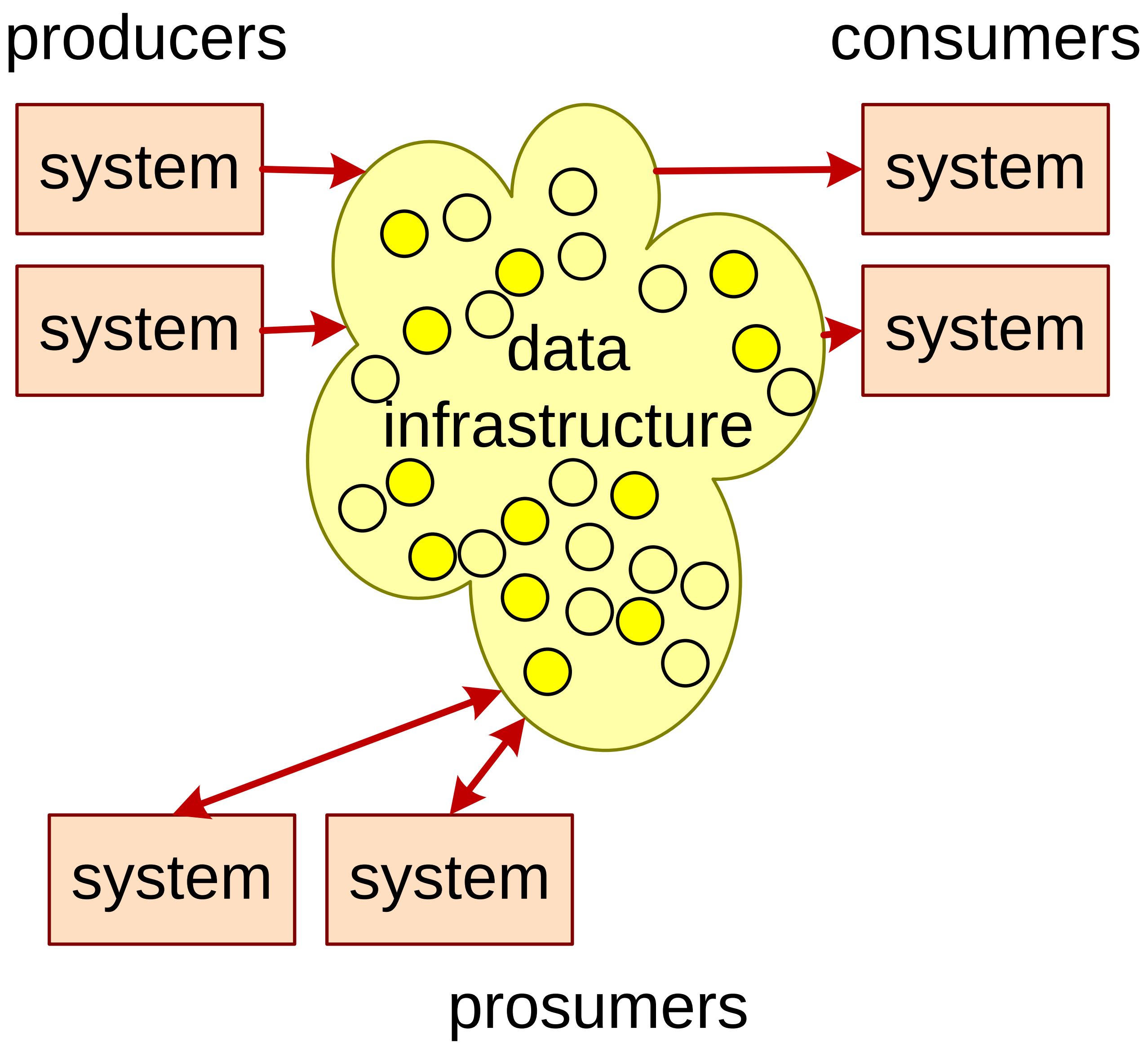
Figure of Content



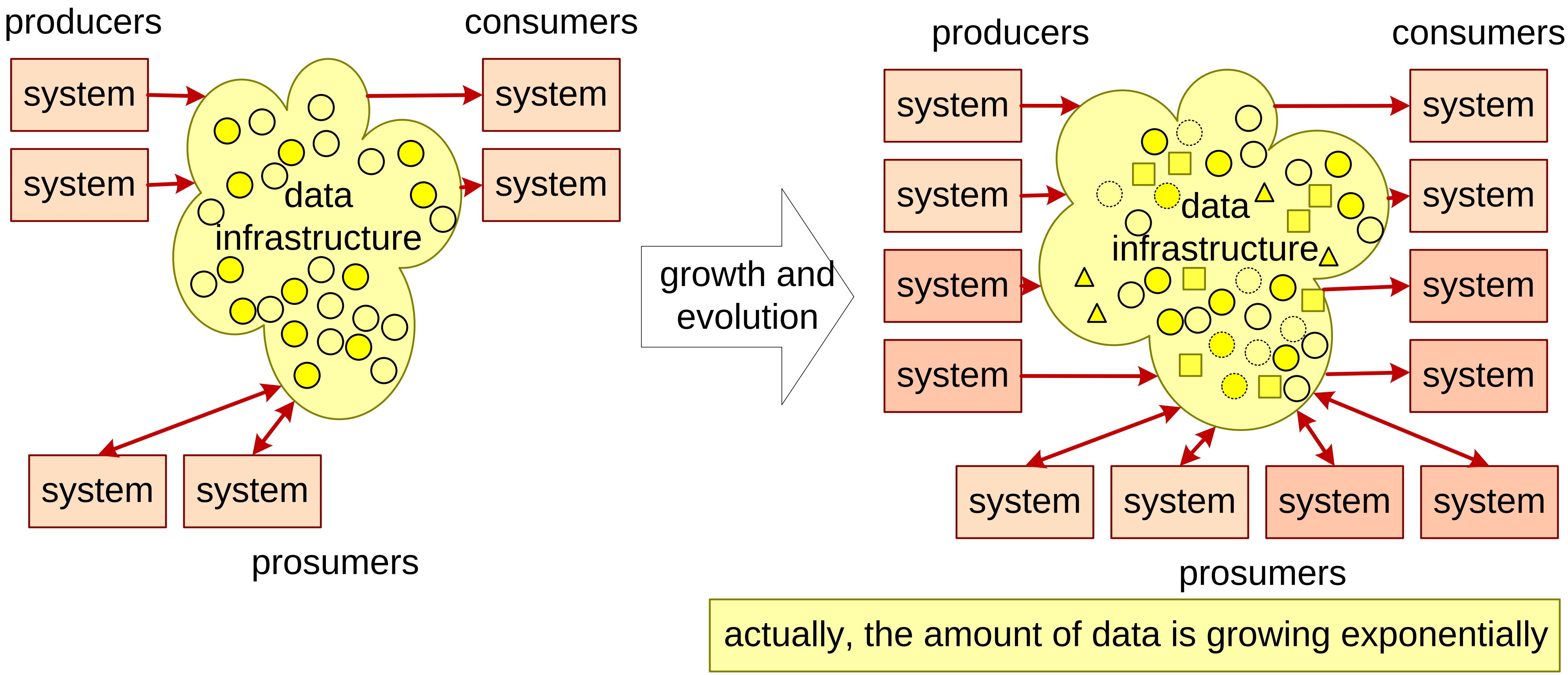
The Context of Software



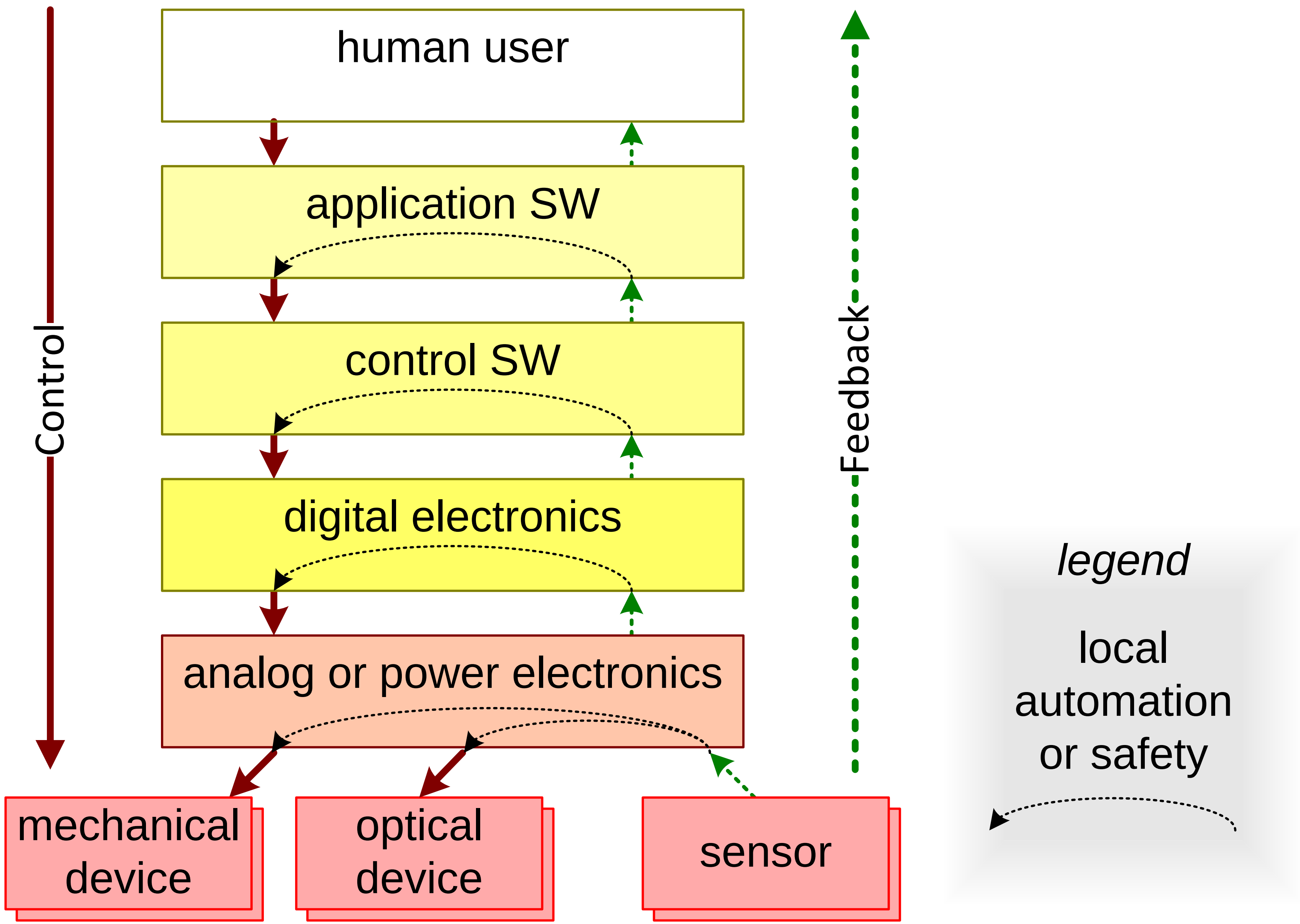
The Context of Data



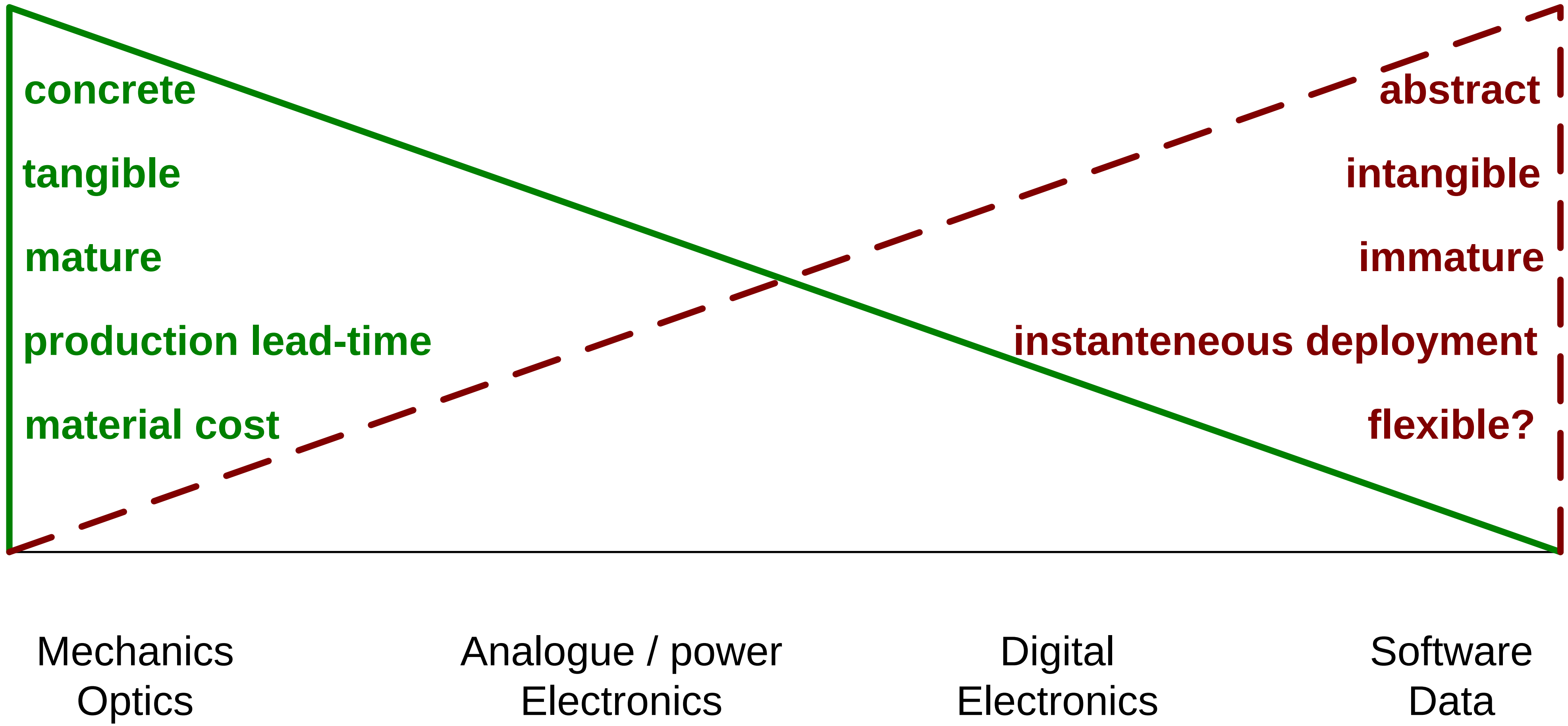
The Context of Data keeps Growing and Evolving



Control Hierarchy along Technology axis



Characterization of disciplines



Role of Sofware in System

integration technology

captures *application* functionality

defines lot of *system* behavior

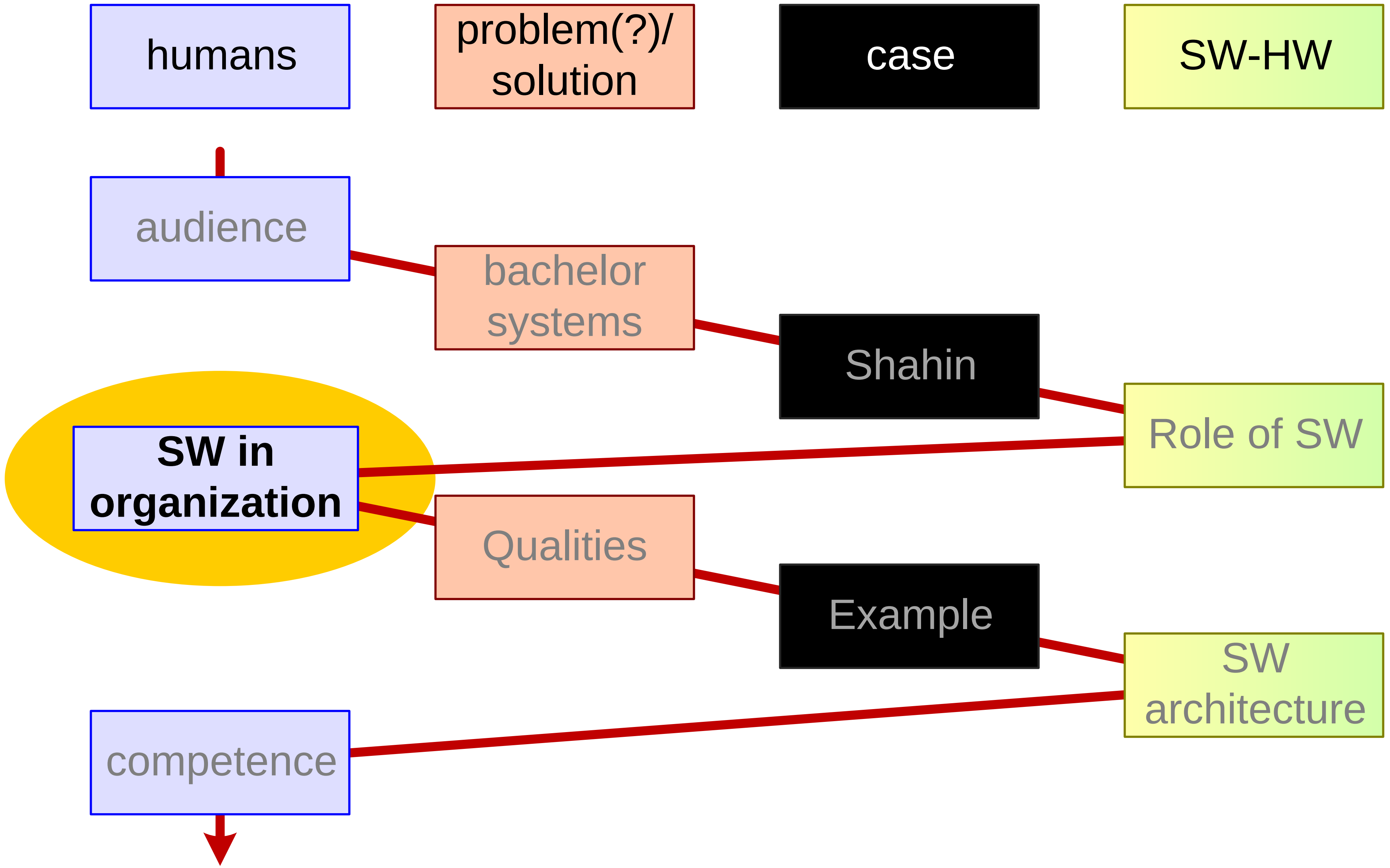
determines how much of potential *system* performance is achieved

acts as director

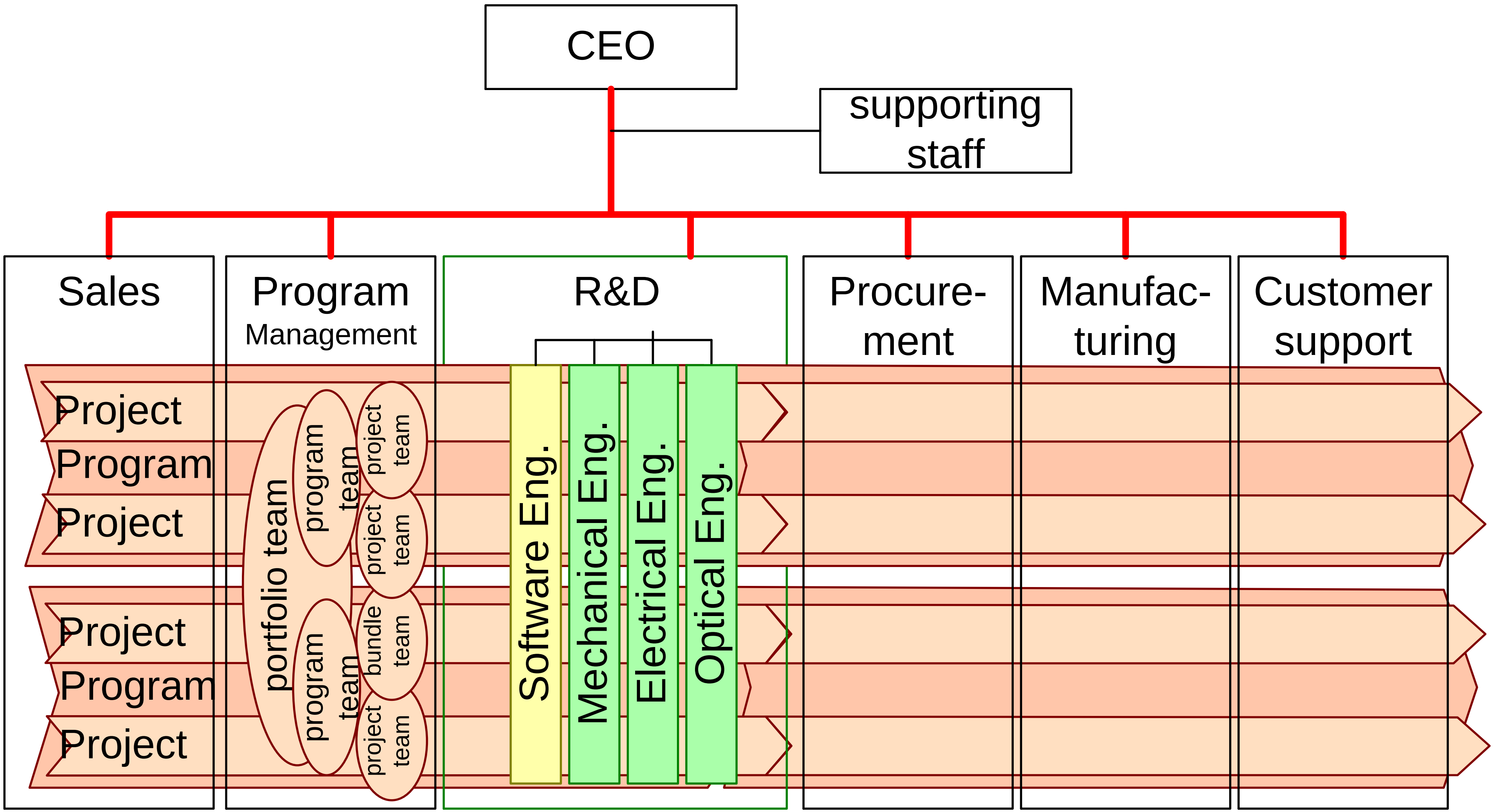
interface to users/*humans*

interface to other systems (producer or consumer of *data*)

Figure of Content



Software as Discipline in a Matrix Organization



Physical an Virtual Disciplines Fit Different Processes

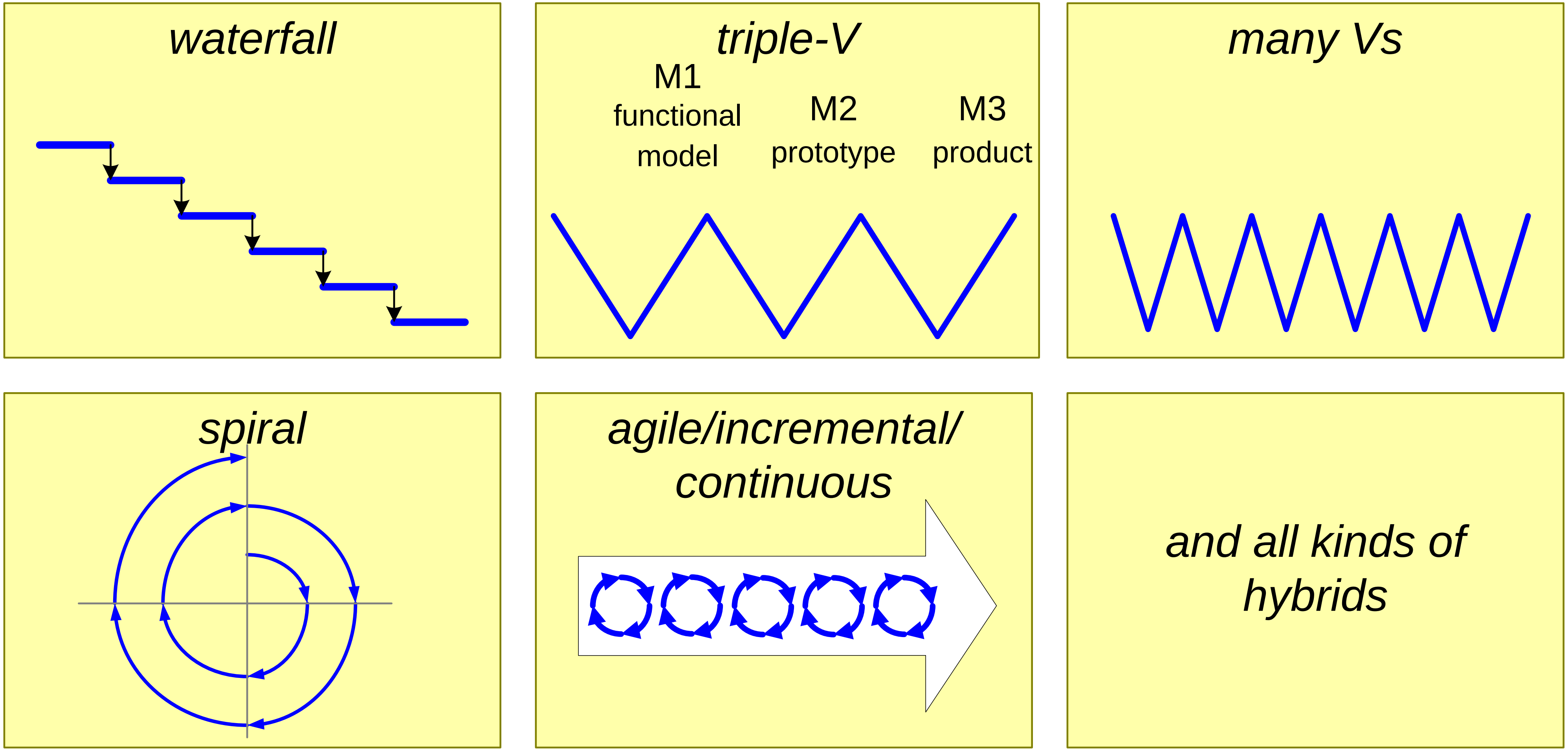
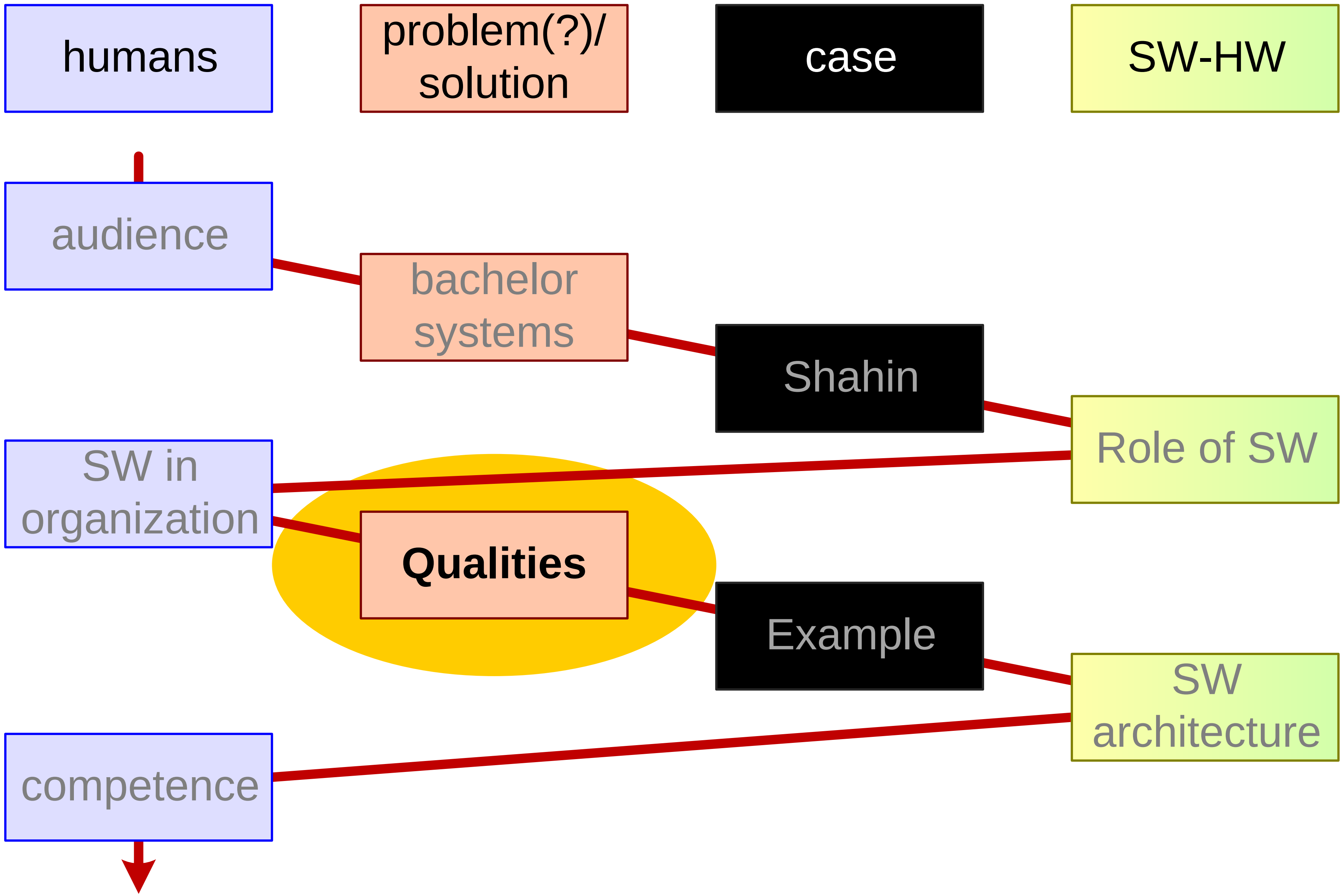
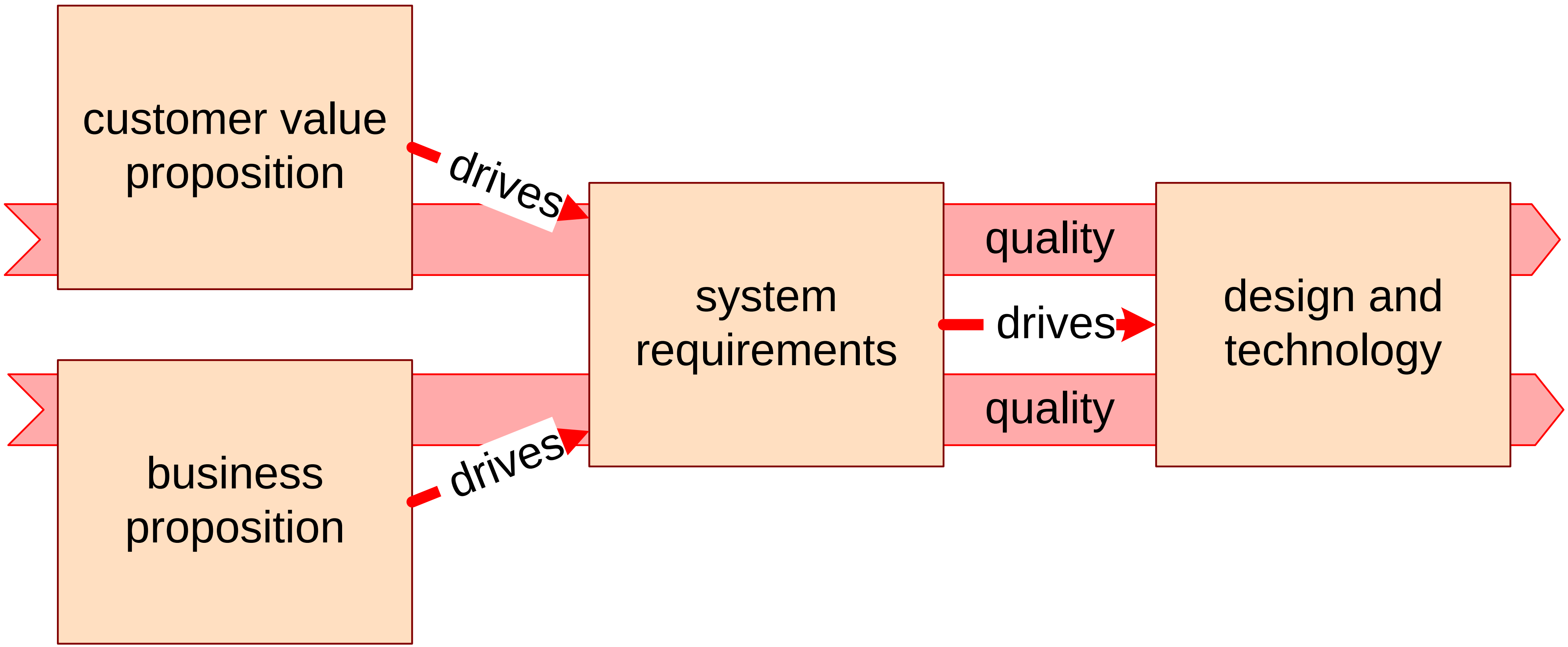


Figure of Content



Qualities Cross all Views



Qualities Checklist

usable
usability
attractiveness
responsiveness
image quality
wearability
storability
transportability

dependable
safety
security
reliability
robustness
integrity
availability

effective
throughput or
productivity

interoperable
connectivity
3rd party extendible

liable
liability
testability
traceability
standards compliance

efficient
resource utilization
cost of ownership

consistent
reproducibility
predictability

serviceable
serviceability
configurability
installability

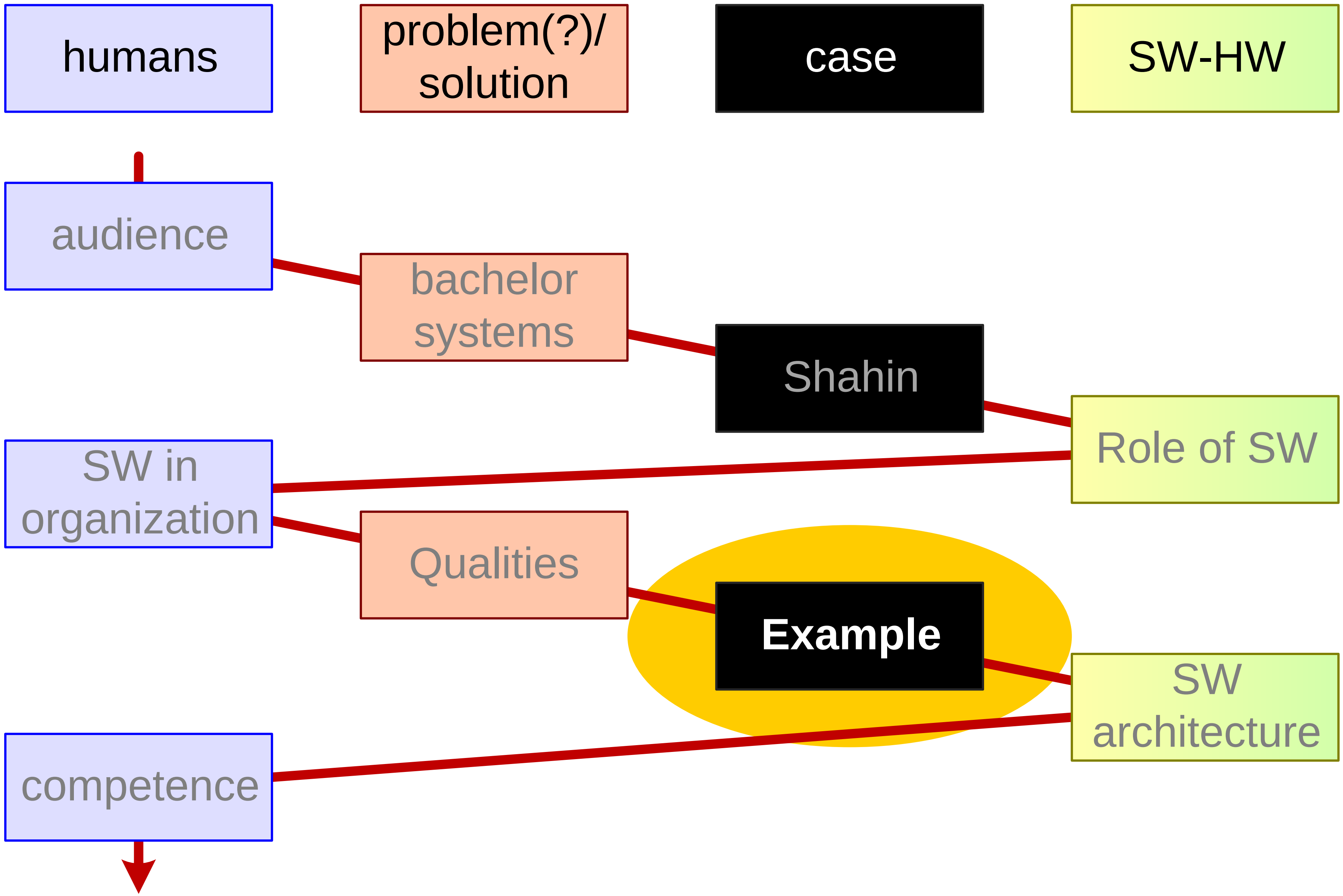
future proof
evolvability
portability
upgradeability
extendibility
maintainability

logistics friendly
manufacturability
logistics flexibility
lead time

ecological
ecological footprint
contamination
noise
disposability

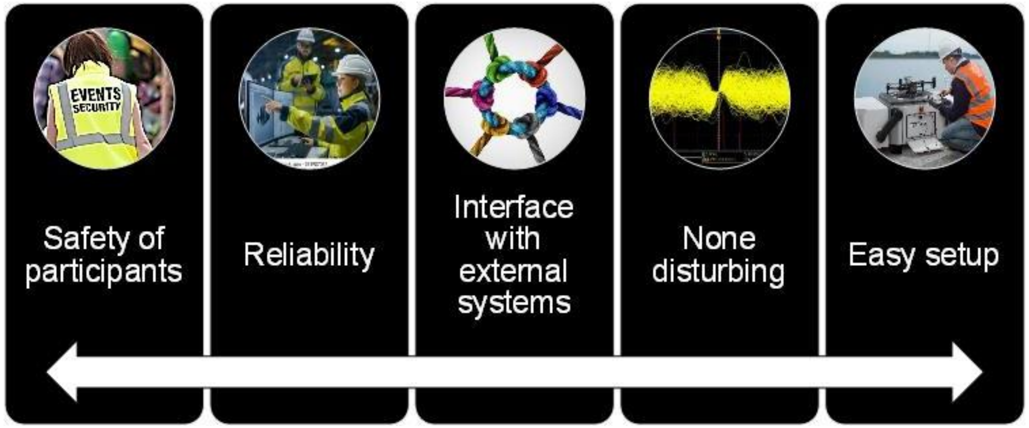
down to earth attributes
cost price
power consumption
consumption rate
(water, air,
chemicals,
et cetera)
size, weight
accuracy

Figure of Content

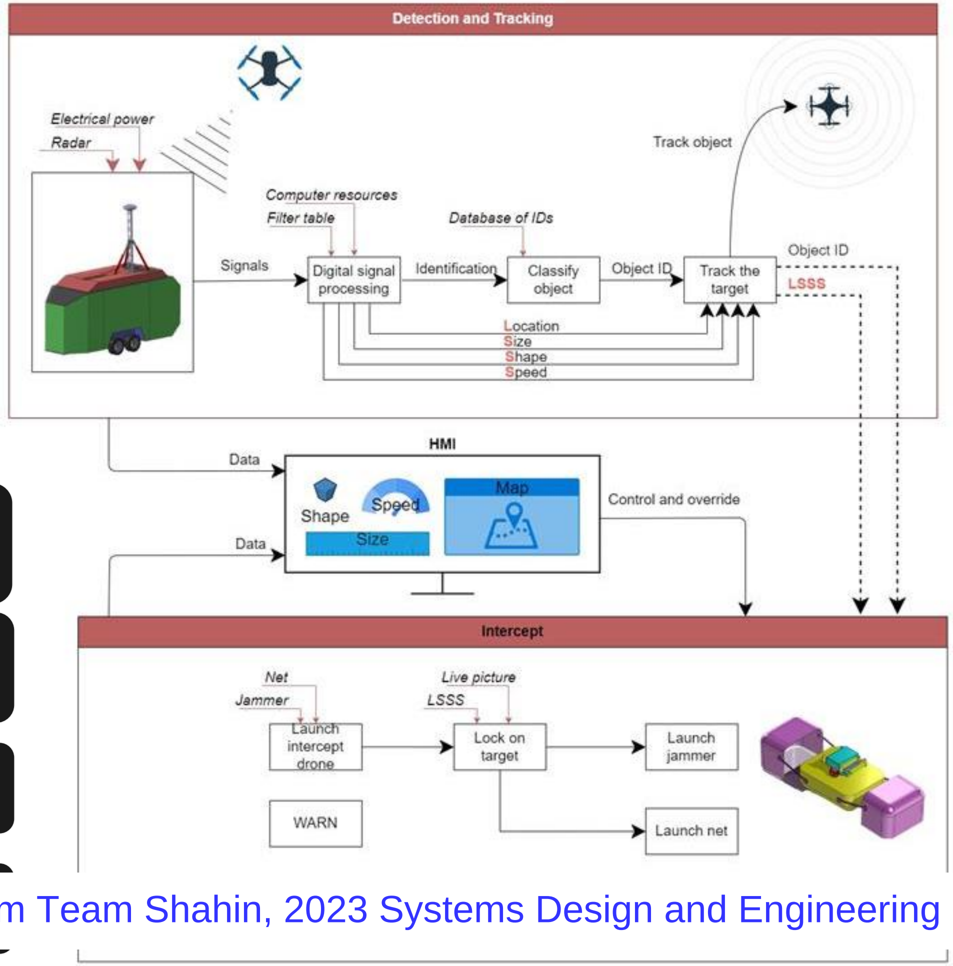


From Key Drivers to System

Customer Key Drivers



System Overview

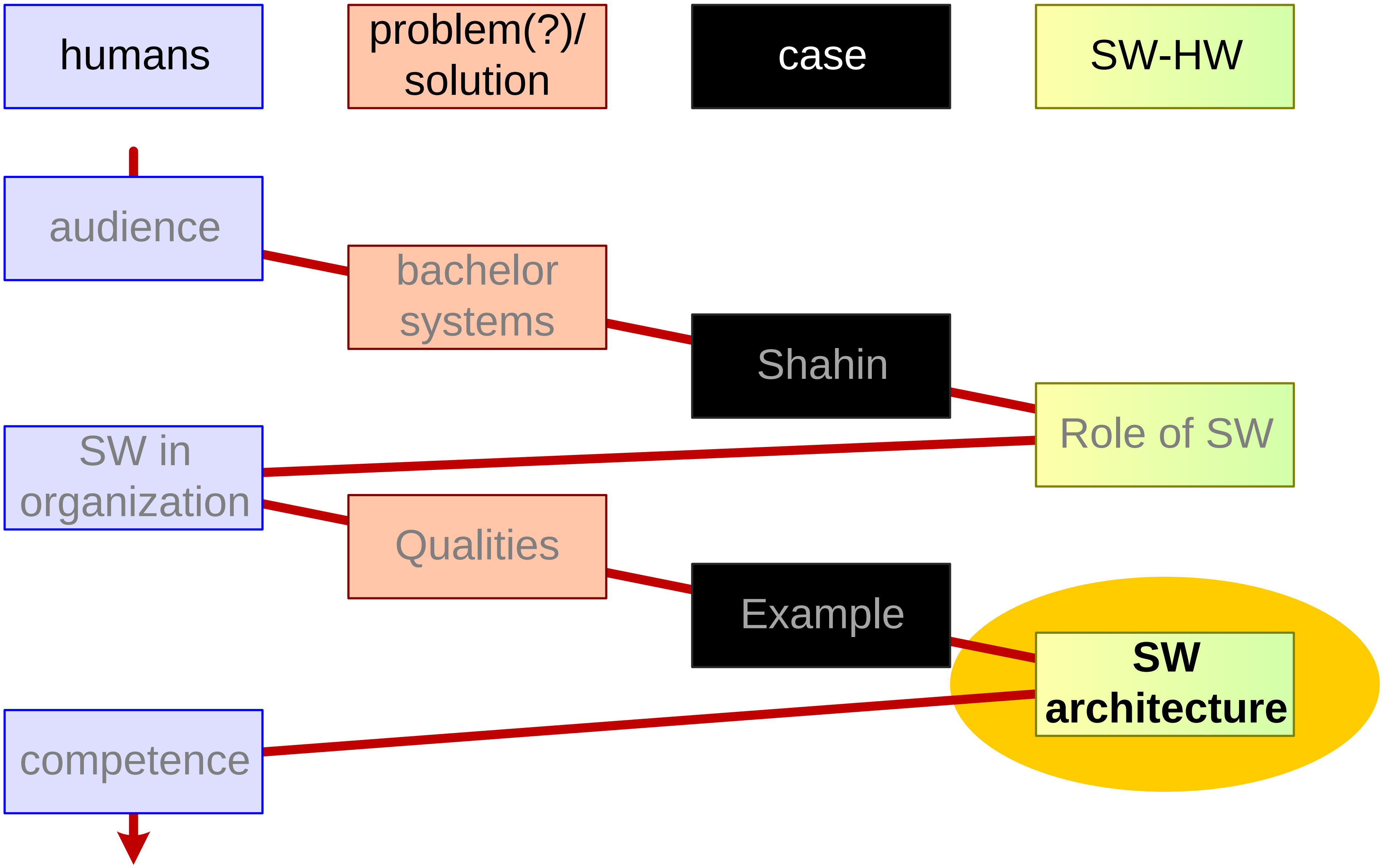


From Team Shahin, 2023 Systems Design and Engineering

System Requirements

Detection	• Drone of size larger than 10 cm • Within 1 km of the site • Non-military drones
Elimination	• Intercept drones • Control it and keep it out • Attempt to ground it
Weather	• Operating at -10 to 50 degree • Rain and snow
System	• Sustainable and future-proof • Avoid disturbance of festival systems

Figure of Content



Quality Attributes annotated with SW relation

usable
- usability
- attractiveness
- responsiveness
- image quality
- wearability
- storability
- transportability

dependable
- safety
- security
- reliability
- robustness
- integrity
- availability

effective
- throughput or productivity

interoperable
- connectivity
- 3rd party extendible

liable
- liability
- testability
- traceability
- standards compliance

efficient
- resource utilization
- cost of ownership

consistent
- reproducibility
- predictability

serviceable
- serviceability
- configurability
- installability

future proof
- evolvability
- portability
- upgradability
- extendibility
- maintainability

logistics friendly
- manufacturability
- logistics flexibility
- lead-time

ecological
- ecological footprint
- contamination
- noise
- disposability

down-to-earth attributes
- cost price
- power consumption
- consumption rate (water, air, chemicals, etc.)
- size, weight
- accuracy

legend
- weak SW relation
- strong SW relation

Design Aspects related to SW

design philosophy per quality attribute — performance, safety, security, ...

granularity, scoping, containment, cohesion, coupling — e.g., distributed or centralized control

interfaces, allocation, budgets

information model (entities, relations, operations)

identification, naming

static characteristics, dynamic behavior

system-level infrastructure

software development process, environment, repository, and tools

life cycle, configuration management, upgrades, obsolescence

feedback tools, for instance monitoring, statistics, and analysis

persistence

licensing, SW-keys

setup sequence, initialization, start-up, shutdown

technology choices

make, outsource, buy, or interoperability decisions

SW Mechanisms

error handling, exception handling, logging

processes, tasks, threads

configuration management; packages, components, files, objects, modules, interfaces

automated testing: special methods, harness, suites

signaling, messaging, callback scheduling, notification, active data, watchdogs, timeouts

locking, semaphores, transactions, checkpoints, deadlock detection, rollback

identification, naming, data model, registry, configuration database, inheritance, scoping

resource management, allocation, fragmentation prevention, garbage collection

persistence, caching, versioning, prefetching, lazy evaluation

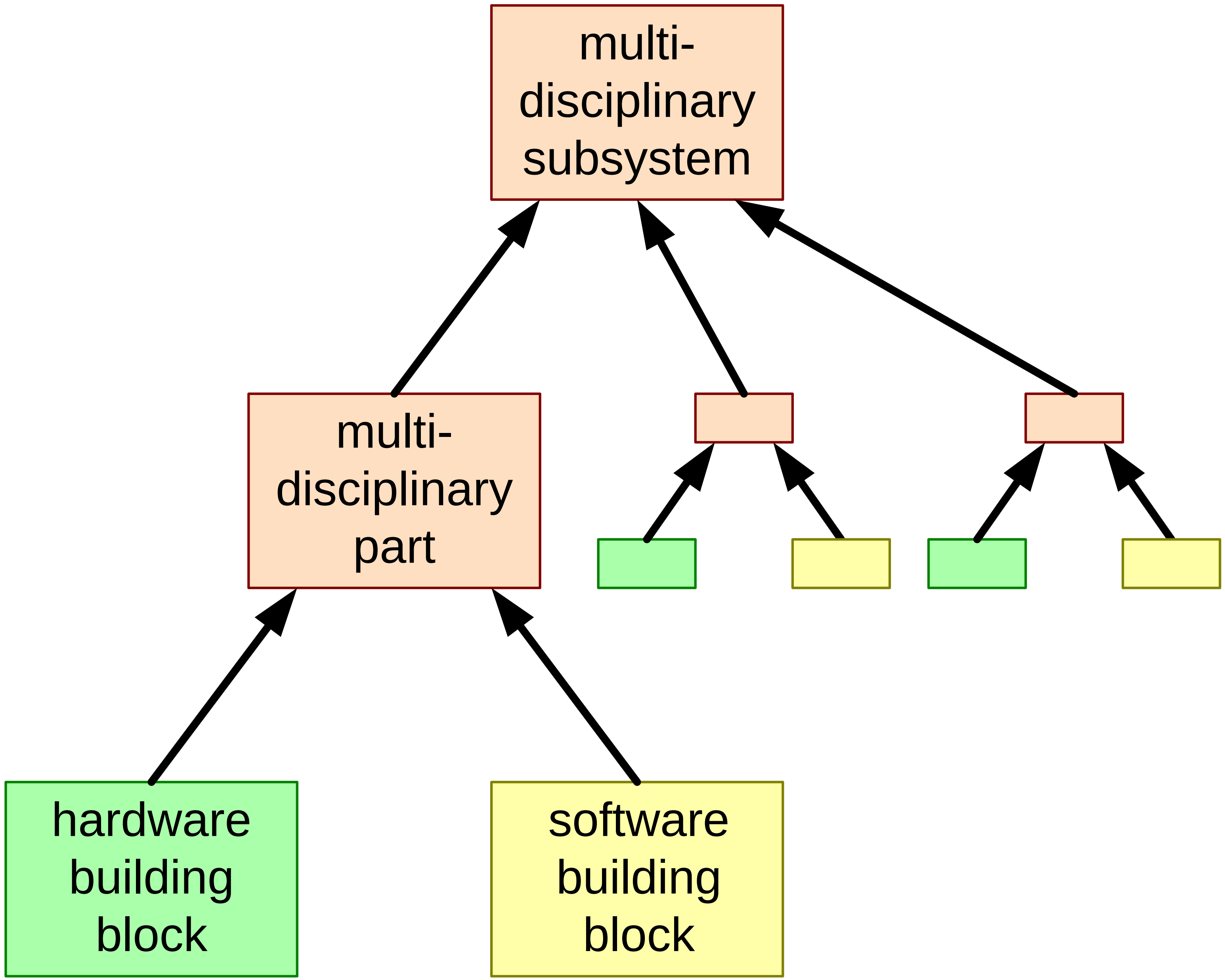
licensing, SW-keys

bootstrap, discovery, negotiation, introspection

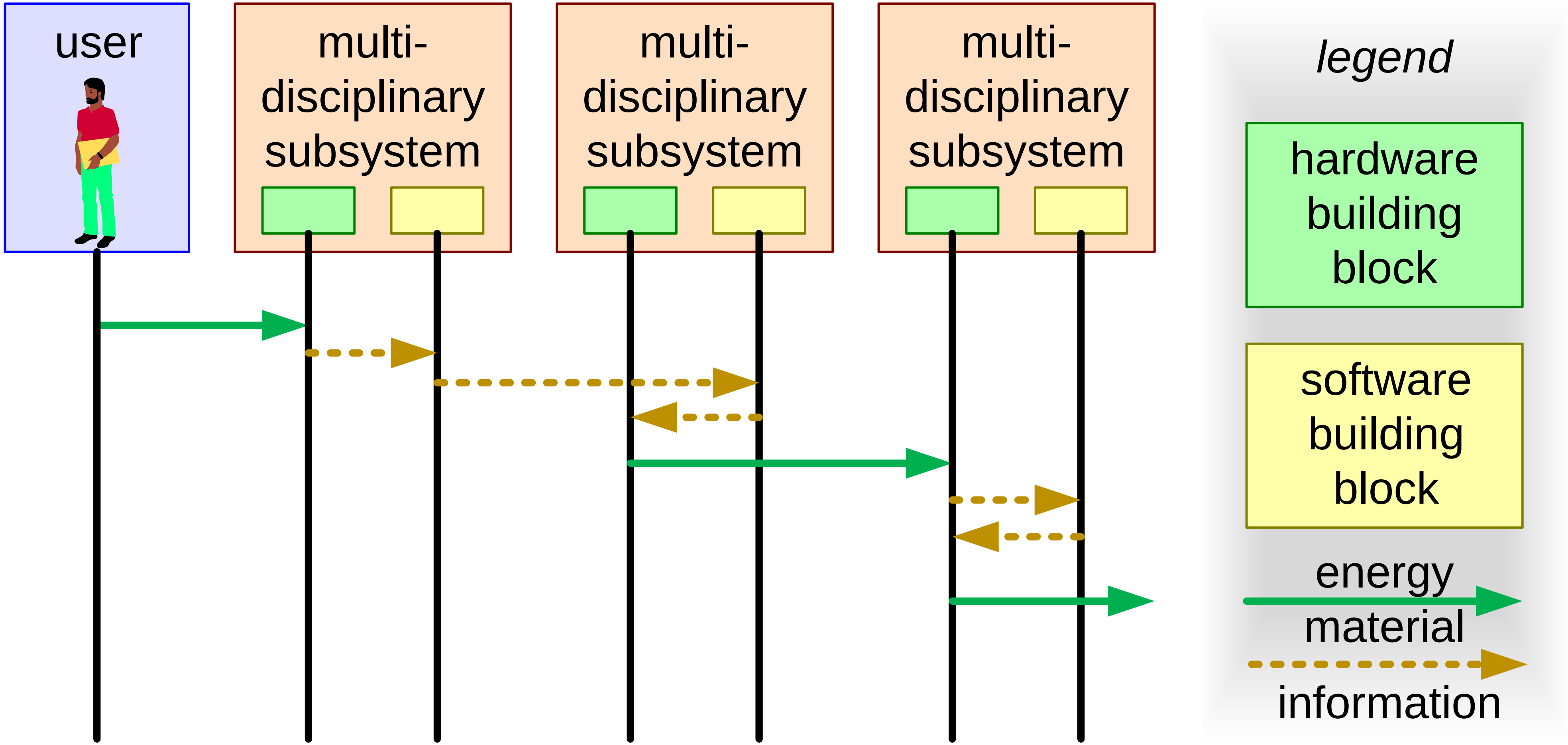
call graphs, message tracing, object tracing, etc.

distribution, allocation, transparency; component, client/server, multitier model

Mono-disciplinary Parts Aggregate into Multi-disciplinary Parts



Behavior Emerges from Interacting Parts



The Perspective changes when Zooming out

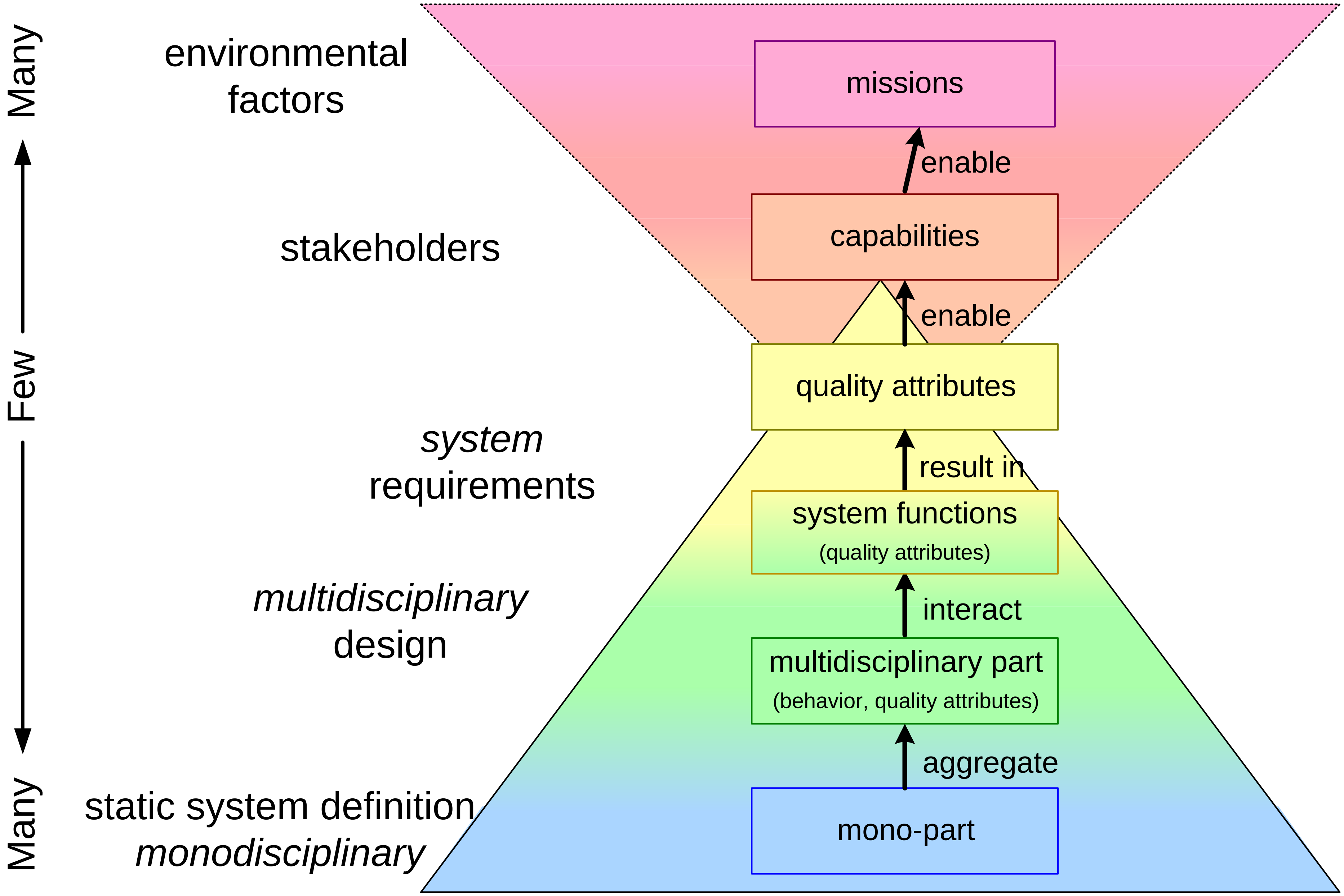
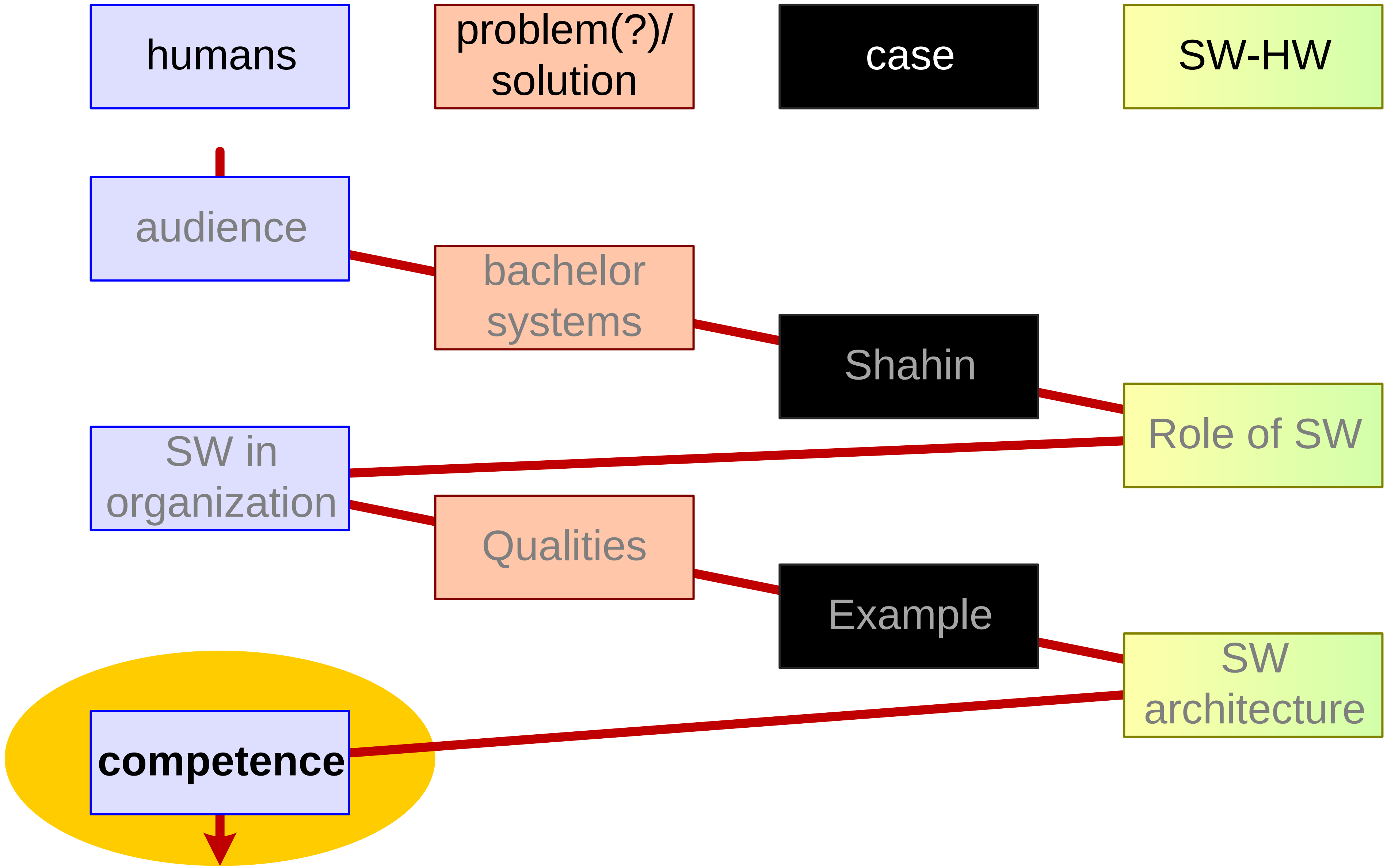
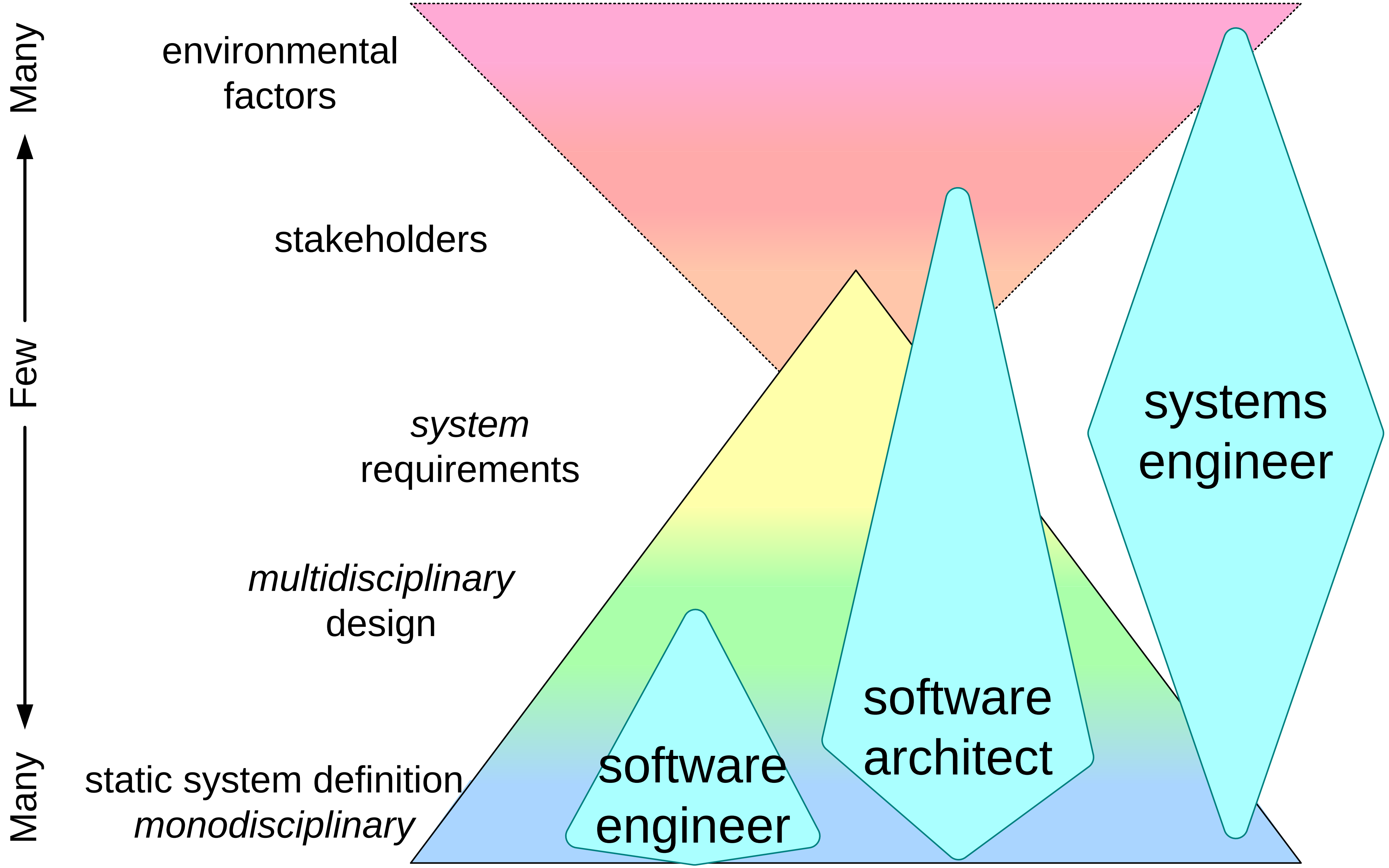


Figure of Content



System and Software Roles (Ideally)



Wish List for Competences

systems engineer	experience the "nature" of virtual technologies cope with abstraction, intangibility combine big picture and agile/iterative develop leadership (soft) competences
software architect	develop outward focus work towards (quantified) qualities own the software design, especially the aspects develop soft skills, grow to leadership
software engineer	understand the other side of the fence (plant, humans, …) dare to quantify, measure, reason about aspects communicate with direct stakeholders develop soft skills