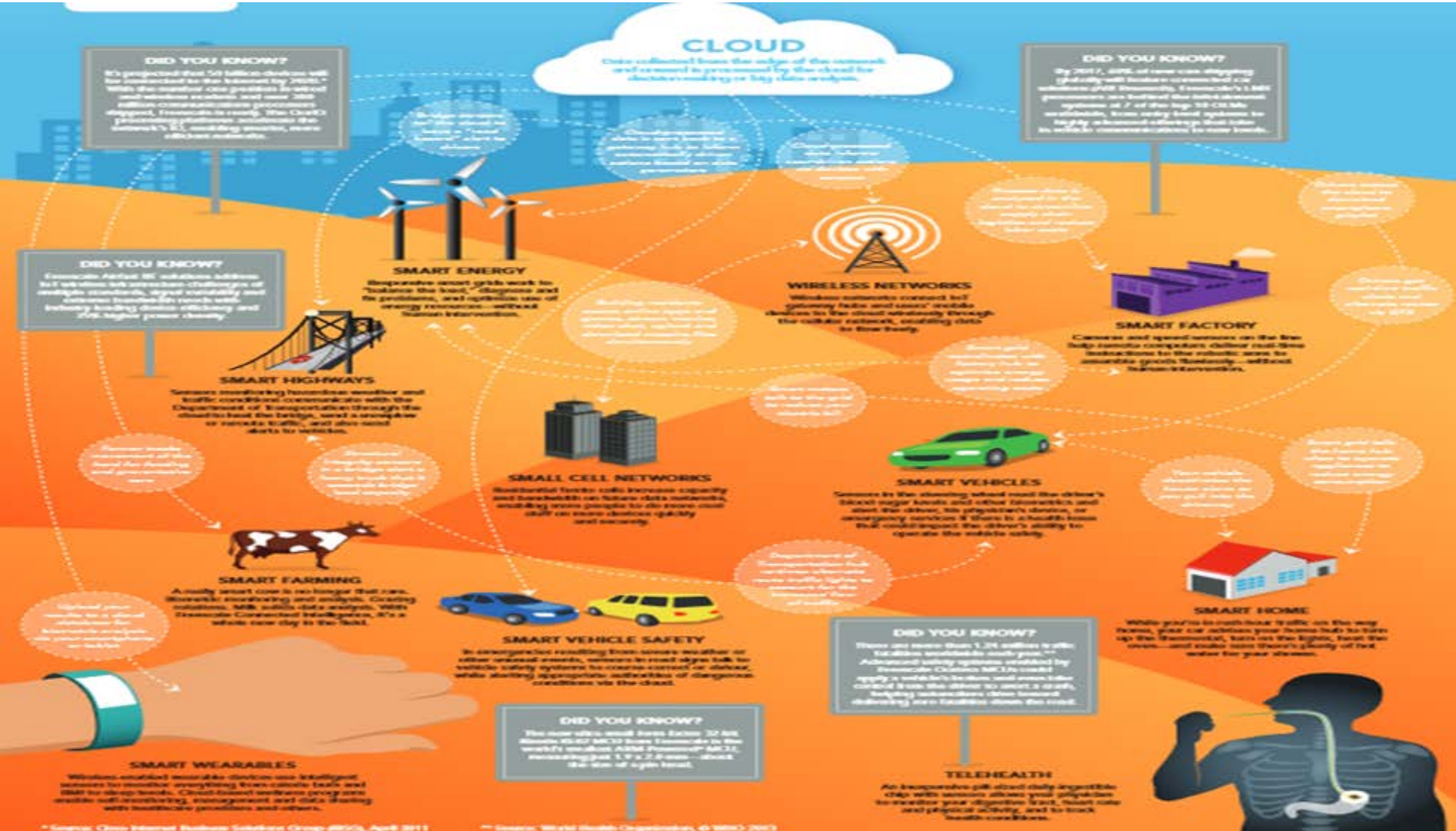
System Design in the Era of IoT

1st International Workshop on
Methods and Tools for Rigorous System Design
Thessaloniki
April 15, 2018

Joseph Sifakis
RSD Team
Verimag Laboratory

System Design Trends&Challenges – The IoT Vision

The IoT allows objects to be sensed or controlled remotely across a network infrastructure, achieving more direct integration of the physical world into computer-based systems, and resulting in improved efficiency and predictability.



System Design Trends&Challenges – The Industrial IoT

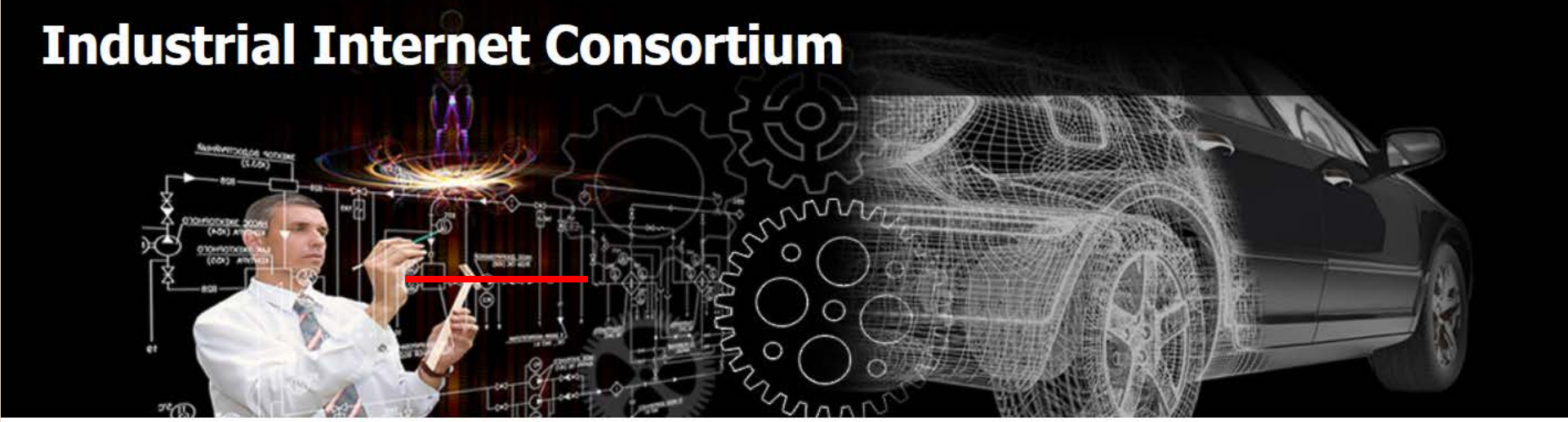
industrial internet CONSORTIUM

Follow us:

http://www.industrialinternetconsortium.org/

Home | About Us | IIC Membership | IIC Committees | Testbeds | Industries | Resource Hub | News & Events | Members Area

Industrial Internet Consortium



IIC Founders Panel – June 18, 2014

Accelerating Innovation In Connected, Intelligent Machines And Processes

Imagine a highway where cars are able to safely navigate to their destinations without a driver. Imagine a home where an elderly patient's health is closely monitored by her hospital physician.

The IIC Blog

Top 10 IoT Security Mishaps 2014 Update

By Brian Witten, Sr Director, IoT Security, Symantec

» Read blog

News Feeds:

Tweets Follow

The IoT Vision – Autonomy

Autonomy is the capacity of an agent (service or system) to achieve a set of coordinated goals by its own means (without human intervention) by adapting to environment variations

- Autonomy of decisions (choosing among possible goals)
- Autonomy of operations planned to achieve the goals
- Autonomy of adaptation e.g. by learning

Autonomous systems are often critical (ITS, Scada, Health, Finance)

The degree of autonomy depends on three factors:

- Complexity of the environment
- Complexity of mission and its implementation as a sequence of feasible tasks
- Non-intervention of human operators

The IoT Vision – SAE Autonomy Levels

Level 0 No automation.

Level 1 Driver assistance required (“hands on”) :
The driver still needs to maintain full situational awareness and control of the vehicle **e.g.** cruise control.

Level 2 Partial automation options available(“hands off”):
Autopilot manages both speed *and* steering under certain conditions, e.g. highway driving.

Level 3 Conditional Automation(“eyes off”):
The car, rather than the driver, takes over actively monitoring the environment when the system is engaged.
However, human drivers must be prepared to respond to a "request to intervene”.

Level 4 High automation (“mind off”):
Self driving is supported only in limited areas (geofenced) or under special circumstances, like traffic jams.

Level 5 Full automation (“steering wheel optional”):
No human intervention is required e.g. a robotic taxi.

- System Design Trends&Challenges

- About System Design

- Two Pillars of System Design

- Component Complexity

- Architecture Complexity

- Knowledge-based Design

- Discussion

OVERVIEW

System Design Trends – Current Limitations

- The requirements for autonomy and increasing integration cannot be met under the current state of the art
  - poor trustworthiness of infrastructures and systems e.g. impossibility to guarantee safety and security;
  - impossibility to guarantee response times in communication thus timeliness which is essential for autonomous reactive systems;
  - Integration of mixed-criticality systems is hard to achieve because critical systems and best-effort systems are developed following two completely different and diverging design paradigms;
  - customization by software updates – Tesla cars software may be updated on a monthly basis.

- Current critical systems standards do not allow any modification after commercialization – trustworthiness is fully established at design time.

  An aircraft is certified as a product that cannot be modified including all its components even HW – aircraft makers purchase and store an advance supply of the microprocessors that will run the software, sufficient to last for the estimated 50 year production!

System Design Trends – Over-optimism

- Many believe that systems can be made secure, dependable etc. without drastically changing the way we design systems and the networking infrastructure
  *"**There is no such thing as a secure system,"** ….. "As we give access to devices around us, from drones to thermostats, we need to make sure they cannot be easily hijacked. **There will be a learning curve before we make them robust, but we'll learn**".* Technology analyst at Davos WEF2016.

- AI is the panacea: extrapolating from recent progress in AI (mainly learning), there is a strong optimism about the ripeness and power of AI techniques on the development of autonomous systems.
  Although AI will be key for achieving autonomy its potential impact to system design seems limited.

- *"I really consider autonomous driving a solved problem. "I think we are probably less than two years away."* Elon Musk June 2, 2016.

System Design Trends – "Resigned Realism"

cerf's up

DOI:10.1145/3130331

Take Two Aspirin a Call Me in the Morr

I use a lot of metaphors in this colur and this one is about security. Secur much on my mind these days along safety and privacy in an increasingly

So where does this leave us? I am fascinated by the metaphor of cyber security as a public health problem. Our machines are infected and they are sometimes also contagious. Our reactions in the public health world involve inoculation and quarantine and we tolerate this because we recognize our health is at risk if other members of society fail to protect themselves from infection. Sadly, virus detection seems to be closing the barn door after the horses have left, to mangle a metaphor. Zero Day attacks cannot be detected with previously cataloged viral signatures, for example. They may help, but perhaps not enough.

System Design Trends – Facing the Challenge

- We need to reassess existing design methodologies in the light of the emerging needs for autonomous adaptable systems
  - understand the limitations to guaranteeing correctness of the designed systems under the current state of the art ;
  - identify relevant research avenues focusing on system design as a process that leads from requirements to mixed HW/SW autonomous systems.

- Systems Engineering comes to a turning point
  - Achieving trustworthiness for autonomous systems requires that we radically change the way we design systems;
  - Adaptive behavior becomes a must as it is impossible to predict all possible mishaps and cope with them at design time.

- What can be reasonably expected from breakthroughs in AI?
  - How can learning and use of knowledge about the designed system help address new design issues?
  - Can we assess safety and security for AI driven systems ?

- System Design Trends&Challenges
- About System Design
- Two Pillars of System Design
- Component Complexity
- Architecture Complexity
- Knowledge-based Design
- Discussion

OVERVIEW

System Design – The Concept of Correctness

Trustworthy: the designed system can be trusted that it will perform as expected despite

HW failures

Design Errors

Environment Disturbances

Malevolent Actions

Optimized: the designed meets quantitative constraints on resources such as time, memory and energy characterizing

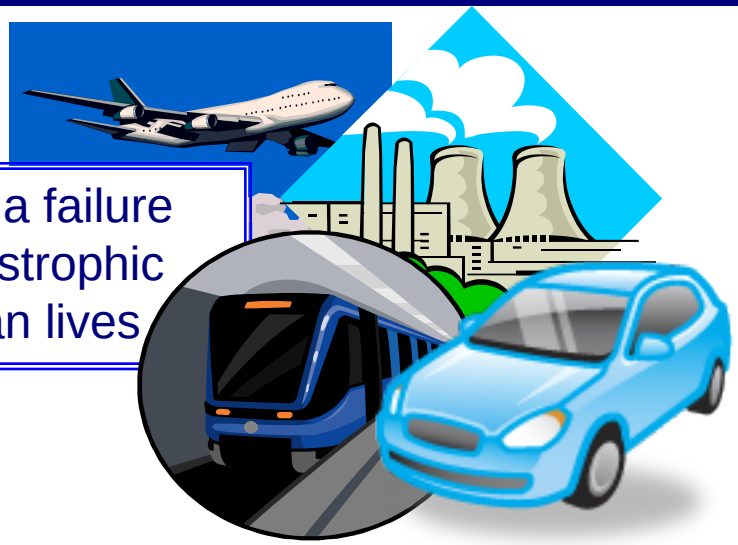
1) performance e.g. throughput, jitter and latency;
2) resources e.g. storage efficiency, processor utilizability

The two types of requirements are antagonistic: System design should determine tradeoffs between cost and quality

System Design – Levels of Criticality

10^{-9}

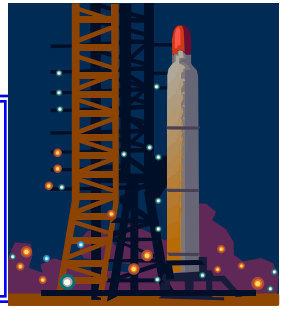
Safety critical: a failure may be a catastrophic threat to human lives

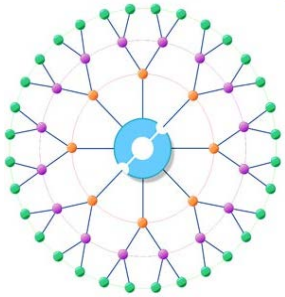


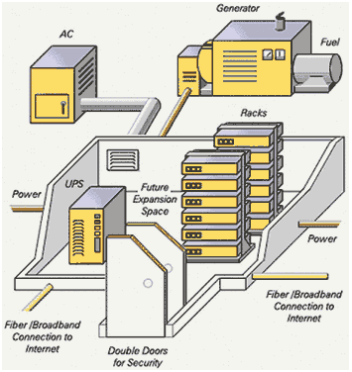
Security critical: harmful unauthorized access

10^{-6}

Mission critical: system availability is essential for the proper running of an organization or of a larger system







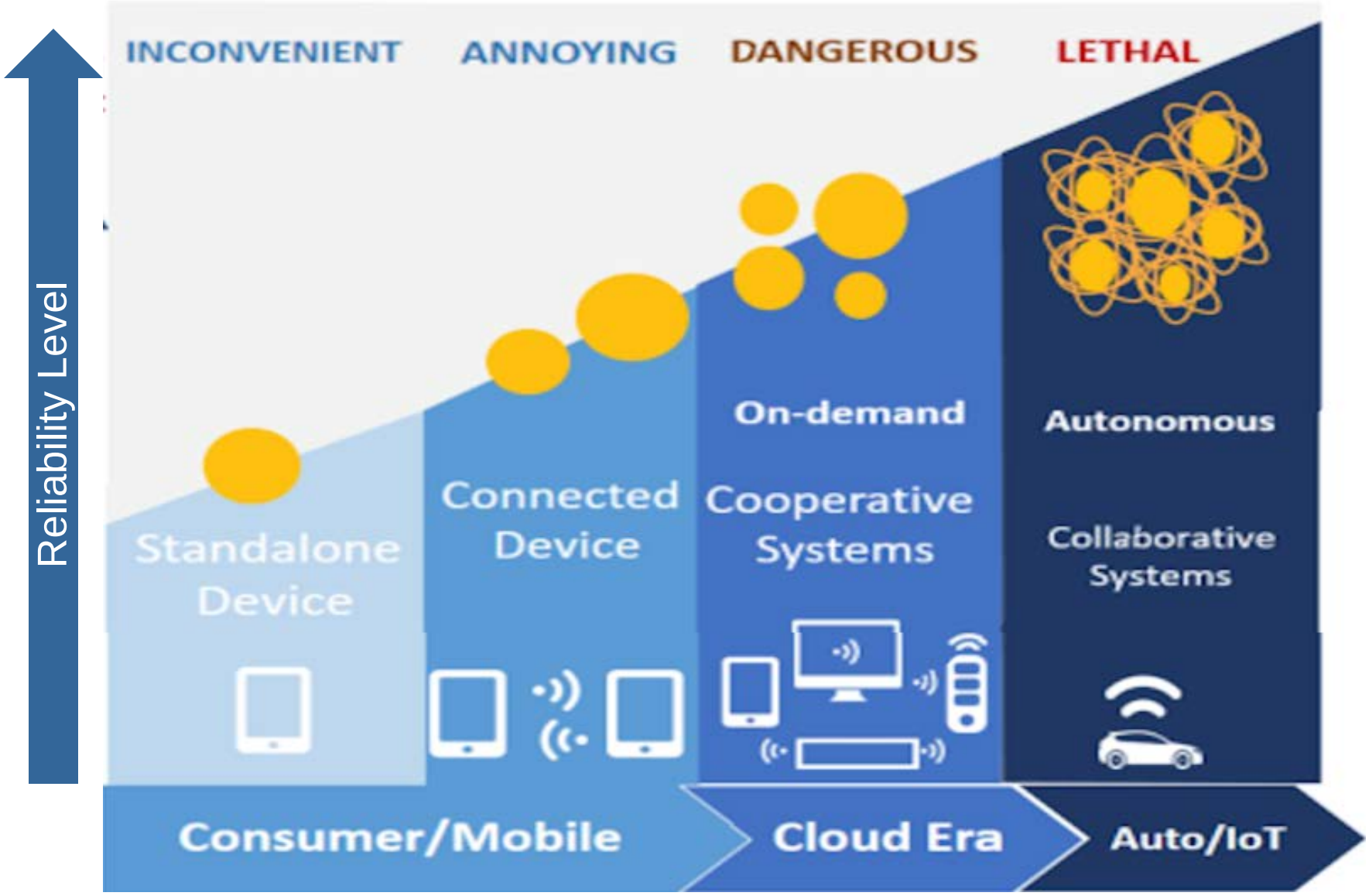
10^{-4}

Best-effort: optimized use of resources for an acceptable level of trustworthiness

amazon.com

Google

System Design – Levels of Criticality



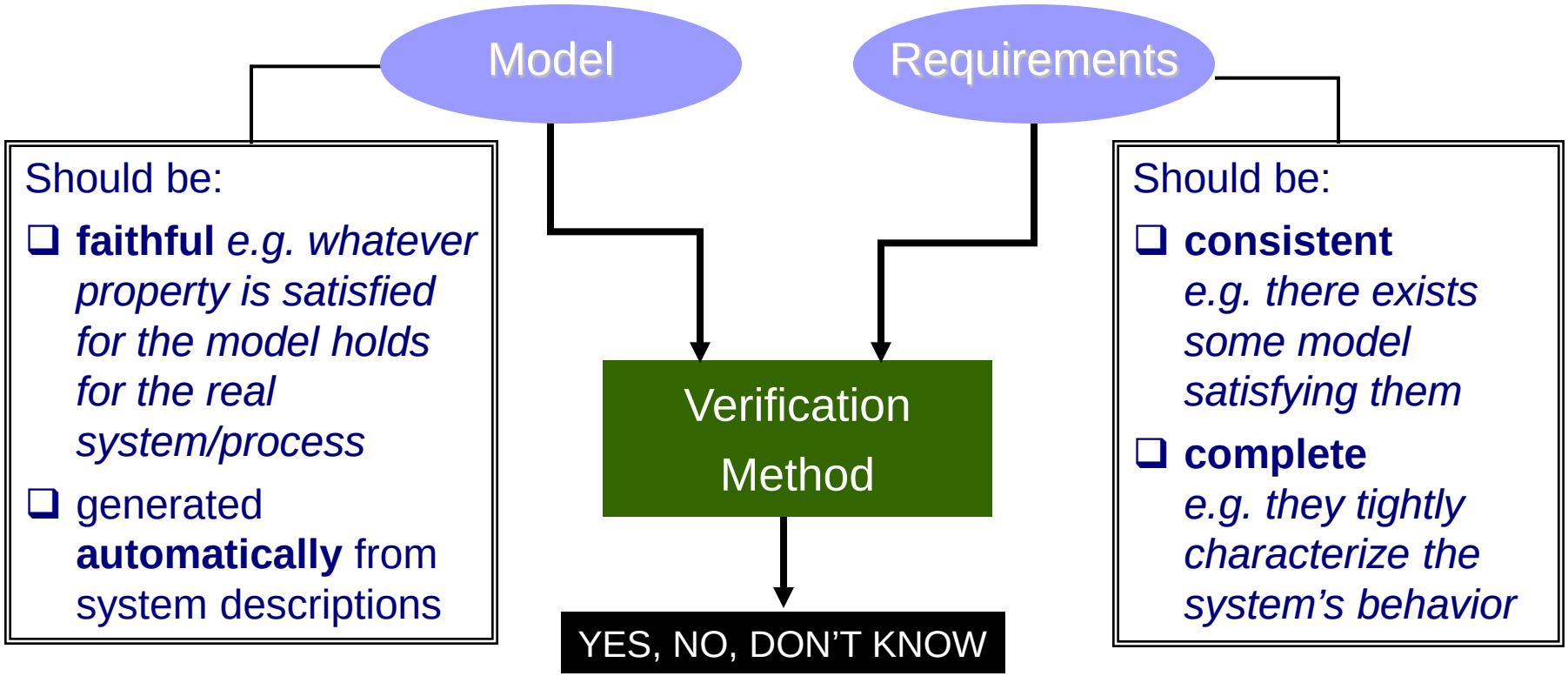
New application trends push requirements in system reliability

OVERVIEW

- System Design Trends&Challenges
- About System Design
- Two Pillars of System Design
- Component Complexity
- Architecture Complexity
- Knowledge-based Design
- Discussion

Two Pillars – Verification

- Verification seeks exhaustive and accountable exploration of system or process models;
- In that respect, it is different from Testing – testing cannot guarantee satisfaction of requirements!



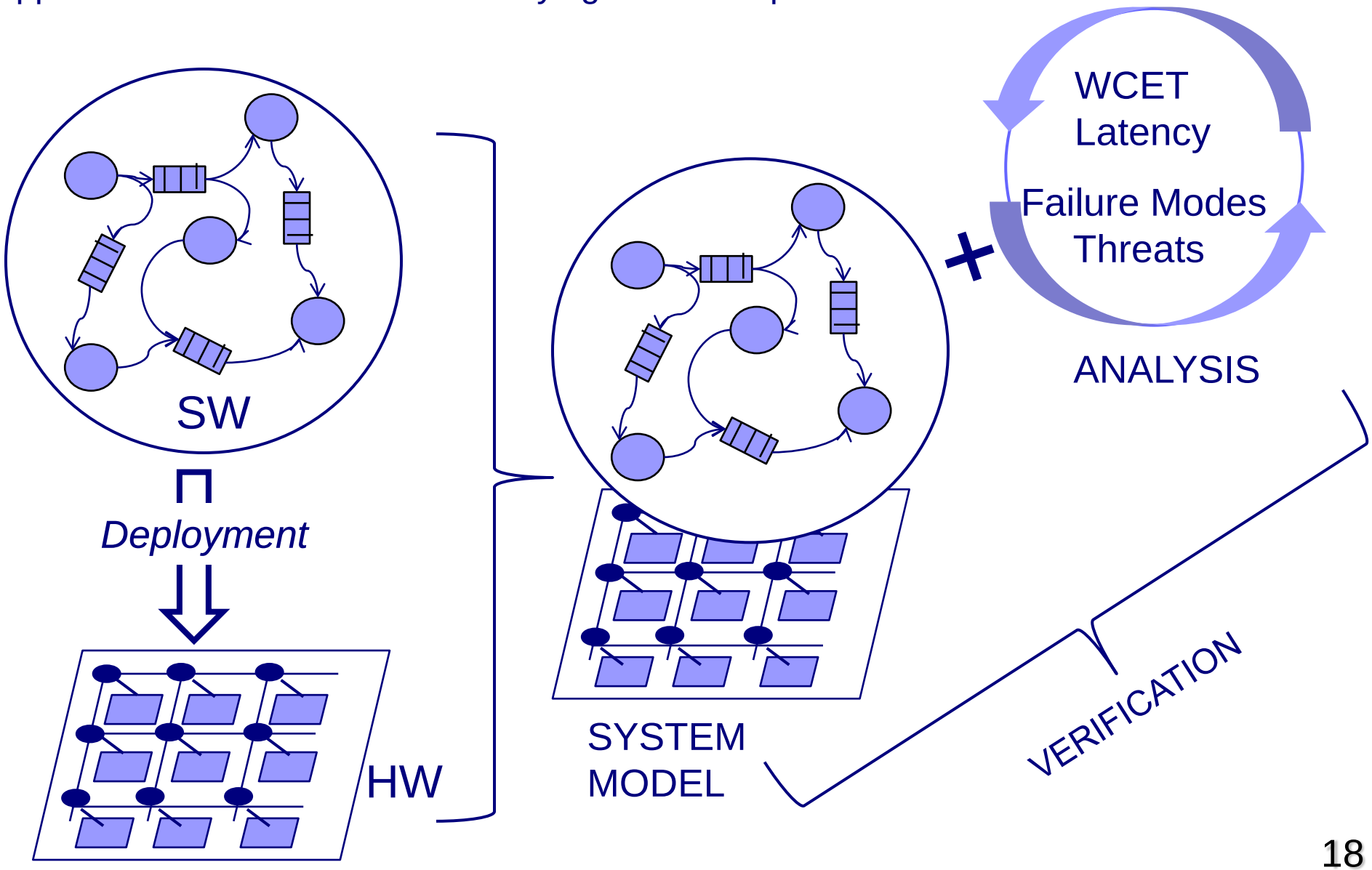
Two Pillars – Verification: Requirements

Formalization of reactive system requirements is problematic e.g. behavioral competencies for self-driving cars (California PATH)

1. 1. Detect and Respond to Speed Limit Changes and Speed Advisories
2. Perform High-Speed Merge (e.g., Freeway)
3. Perform Low-Speed Merge
4. Move Out of the Travel Lane and Park (e.g., to the Shoulder for Minimal Risk)
5. Detect and Respond to Encroaching Oncoming Vehicles
6. 6. Detect Passing and No Passing Zones and Perform Passing Maneuvers
7. Perform Car Following (Including Stop and Go)
8. Detect and Respond to Stopped Vehicles
9. Detect and Respond to Lane Changes
10. Detect and Respond to Static Obstacles in the Path of the Vehicle
11. Detect Traffic Signals and Stop/Yield Signs
12. Respond to Traffic Signals and Stop/Yield Signs
13. 13. Navigate Intersections and Perform Turns
14. Navigate Roundabouts
15. Navigate a Parking Lot and Locate Spaces
16. Detect and Respond to Access Restrictions (One-Way, No Turn, Ramps, etc.)
17. Detect and Respond to Work Zones and People Directing Traffic in Unplanned or Planned Events
18. 18. Make Appropriate Right-of-Way Decisions
19. Follow Local and State Driving Laws
20. Follow Police/First Responder Controlling Traffic (Overriding or Acting as Traffic Control Device)
21. Follow Construction Zone Workers Controlling Traffic Patterns (Slow/Stop Sign Holders).
22. Respond to Citizens Directing Traffic After a Crash
23. Detect and Respond to Temporary Traffic Control Devices
24. Detect and Respond to Emergency Vehicles
25. Yield for Law Enforcement, EMT, Fire, and Other Emergency Vehicles at Intersections, Junctions, and Other Traffic Controlled Situations
26. Yield to Pedestrians and Bicyclists at Intersections and Crosswalks
27. Provide Safe Distance From Vehicles, Pedestrians, Bicyclists, Sides of the Road
28. 28. Detect/Respond to Detours and/or Other Temporary Changes in Traffic Patterns

Two Pillars – Verification: Building Models

Building system models requires understanding intricate interaction between application software and the underlying execution platform



System Design – Verification: Limitations

- System verification

  - formal verification is tractable for moderate complexity
  - only monolithic verification techniques of finite state systems can be automated;
  - research on compositional verification has failed to produce any practically relevant result

  - is not only about that the delivered service in nominal behavior.
  It is also and mainly about understanding all the different mechanisms in which faults occur, and how the system reacts to these faults;

  - cannot address optimization requirements, a more natural approach is to enforce them by design or synthesis

- Proof of correctness by itself is not enough – its acceptance is a social process! Only when proof is substantiated by evidence, checked by a trusted entity e.g. certification authority, does it become believable.
  ("Social processes and proofs of theorems and programs" Alan J. Perlis, Richard A. DeMillo , Richard J. Lipton, CACM, May 1979, Volume 22 Number 5)

Two Pillars – Verification: Limitations

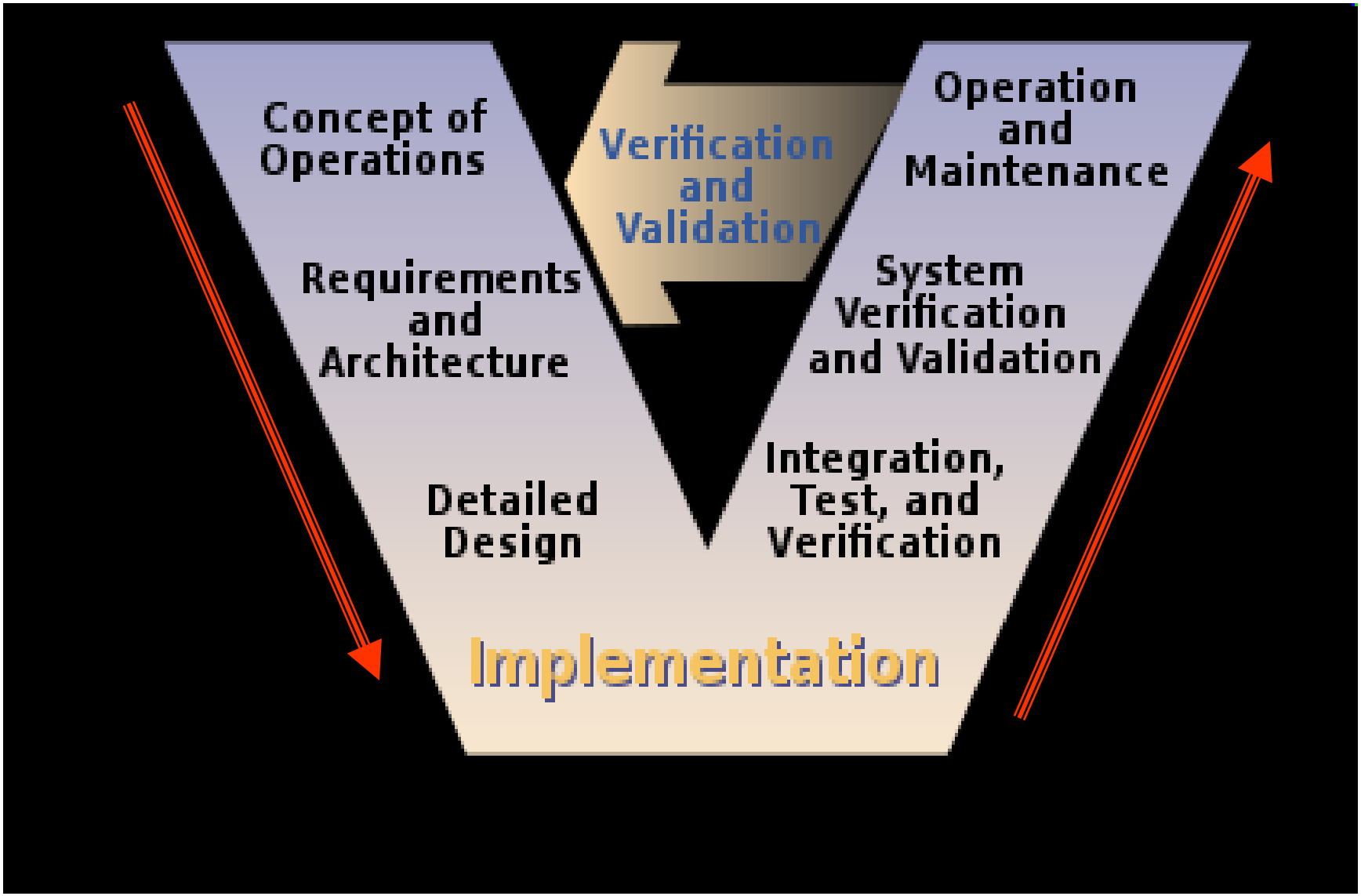
Machine learning techniques – viewed by many experts as the key to building autonomous systems – cannot be verified.
So establishing their safety according to existing standards is problematic.

Neural networks

- are not developed based on requirements e.g. that specify how a dog looks different from a cat
- instead, we are showing a whole bunch of pictures so they can learn just like a human learns the differences between a cat and a dog.

- furthermore the nature of the computation is not procedural -- learning software cannot be verified as an algorithm.

Two Pillars – The V-model



The V-model of the Systems Engineering Process, Source: Wikipedia

Two Pillars – The V-model

The V-model of the traditional Systems Engineering process

1. Assumes that all the system requirements are initially known, can be clearly formulated and understood.

2. Assumes that system development is top-down from a set of requirements. Nonetheless, systems are never designed from scratch; they are built by incrementally modifying existing systems and component reuse.

3. Considers that global system requirements can be broken down into requirements satisfied by system components.

4. Relies mainly on correctness-by-checking (verification or testing).

Safety standards like ISO26262, require a V-model. Nonetheless,

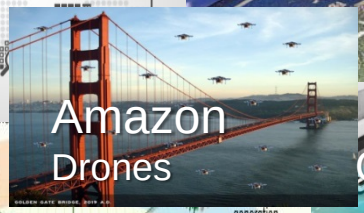
- The V-model cannot be applied machine-learning techniques;
- The V-model is not adopted by most software development methodologies e.g. *agile development* which considers that coding and designing go hand in hand: designs should be modified to reflect adjustments made to the requirements. So, design ideas are shared and improved on during a project.

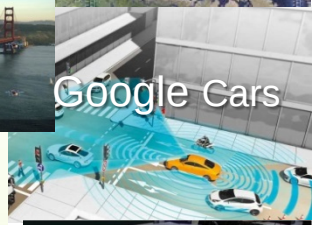
OVERVIEW

- System Design Trends&Challenges
- About System Design
- Two Pillars of System Design
- Component Complexity
- Architecture Complexity
- Knowledge-based Design
- Discussion

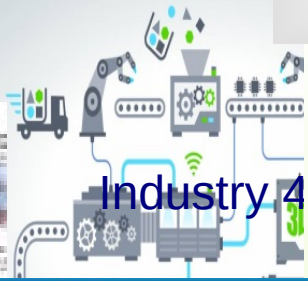
Design Complexity

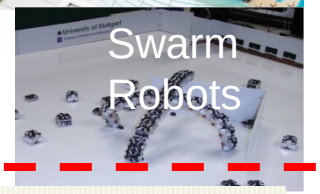
Agricultural Robot

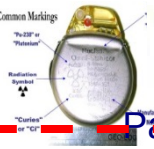




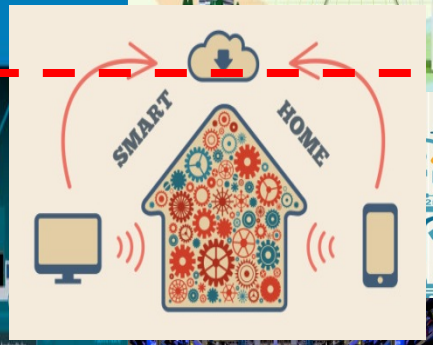
Active safety







Pacemaker



COMPONENT BEHAVIOR COMPLEXITY

CyberPhy

Embedded

Streaming

Transformational

SYSTEMS

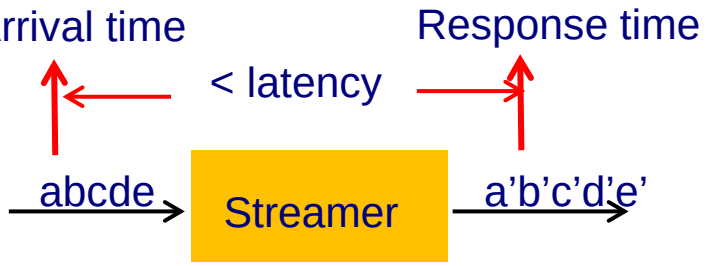
SERVICES

Static parametric distributed reconfigurable mobile self-organizing

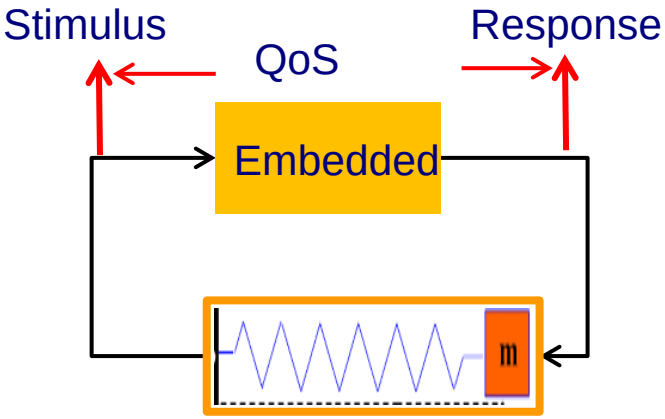
ARCHITECTURAL COMPLEXITY

Component Complexity

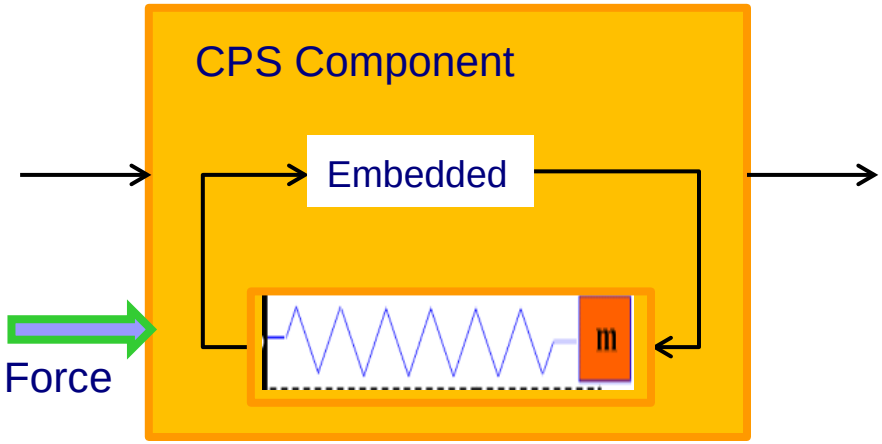
Functions, Methods e.g.
Client-server Systems



Streaming Systems
e.g. Encoders, Signal processing systems



Embedded Systems
e.g. Flight controller

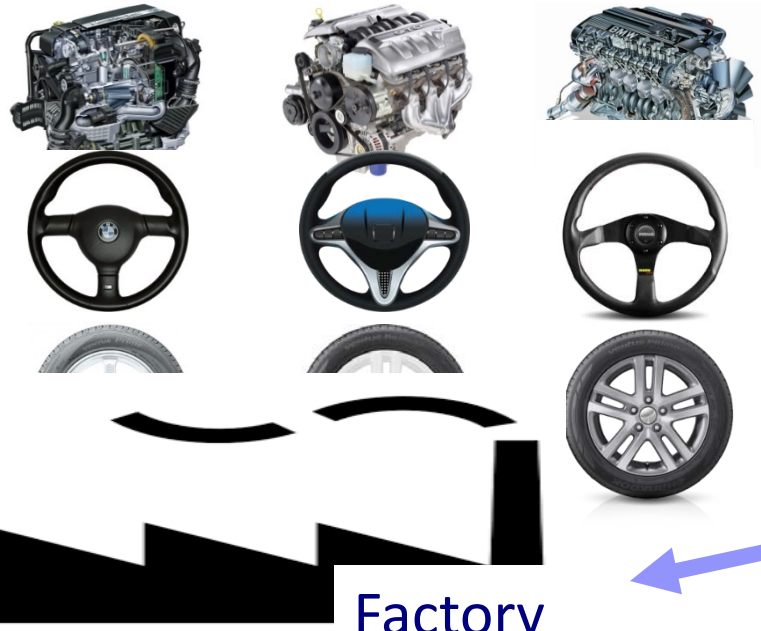


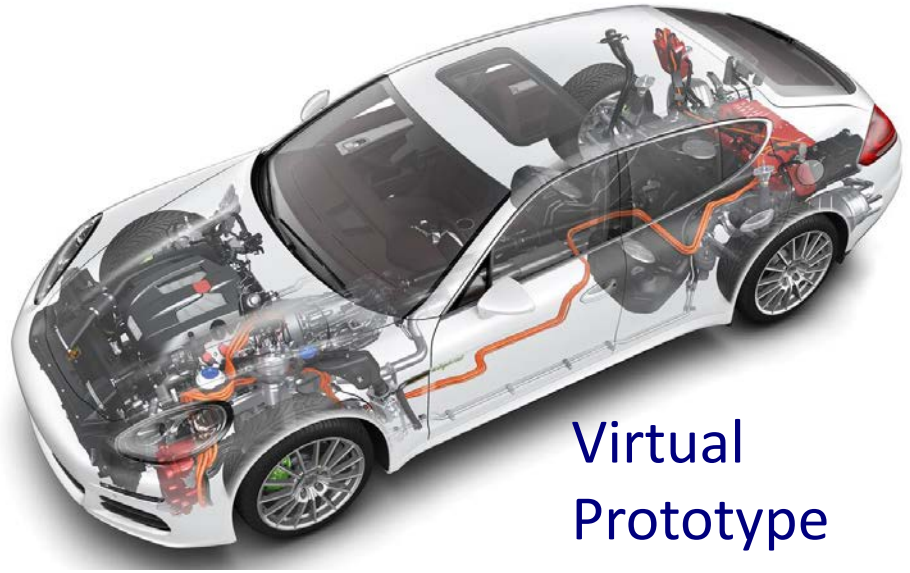
Cyberphysical Systems
e.g. Self-driving cars

Component Complexity – Cyber physical Systems

ON CLOUD

Library of Components

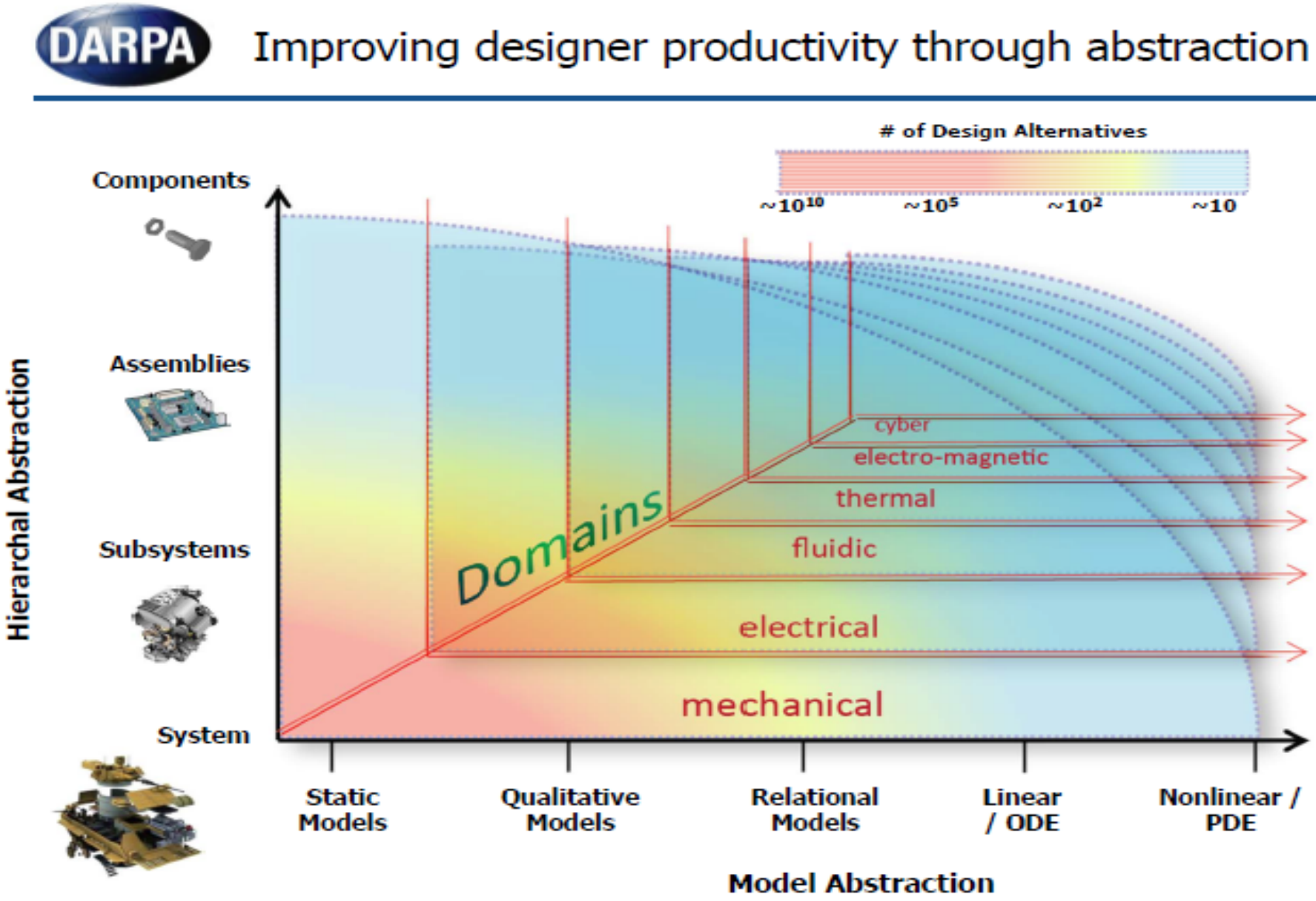




Factory

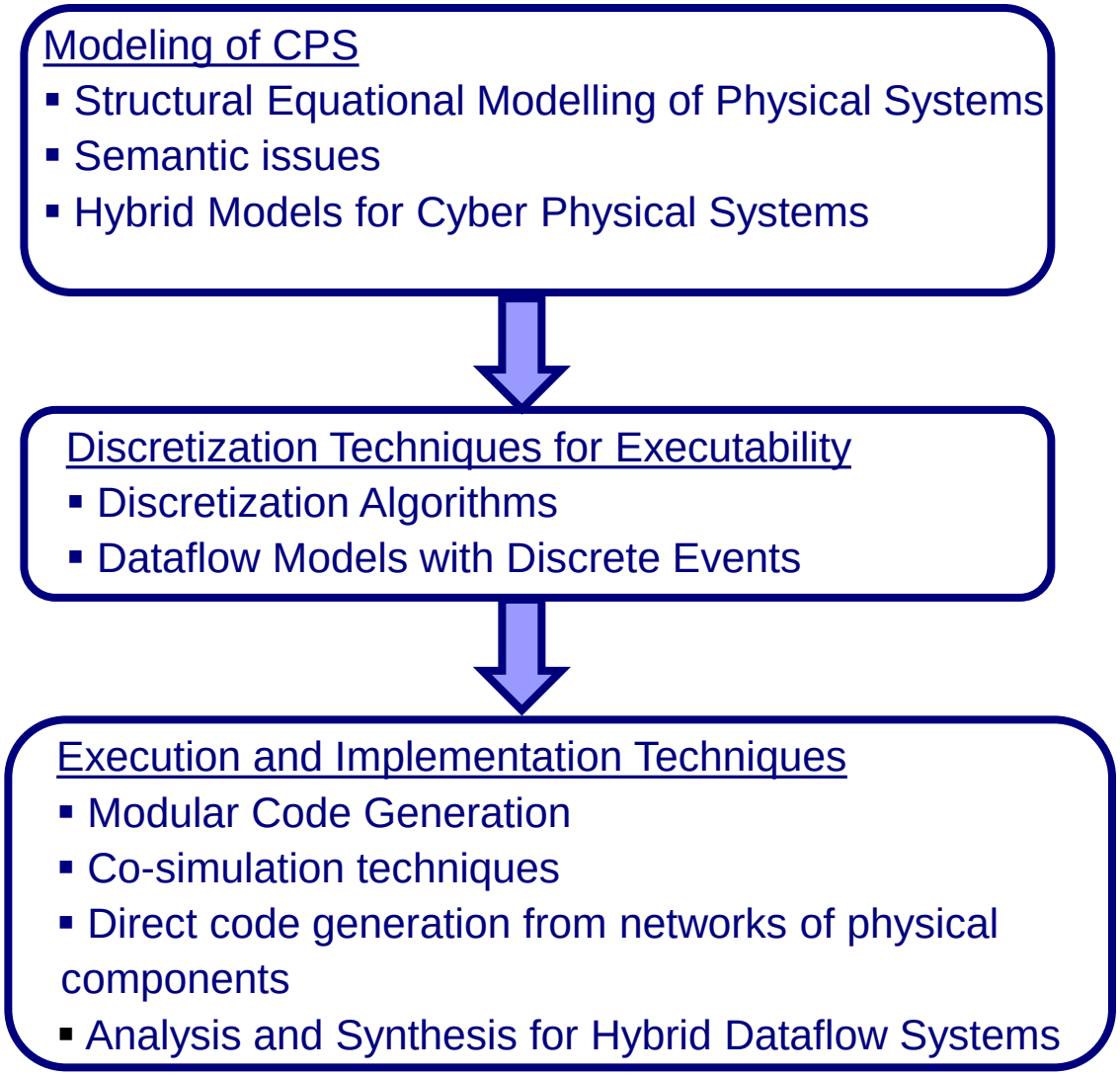
Virtual Prototype

Component Complexity – Cyber physical Systems

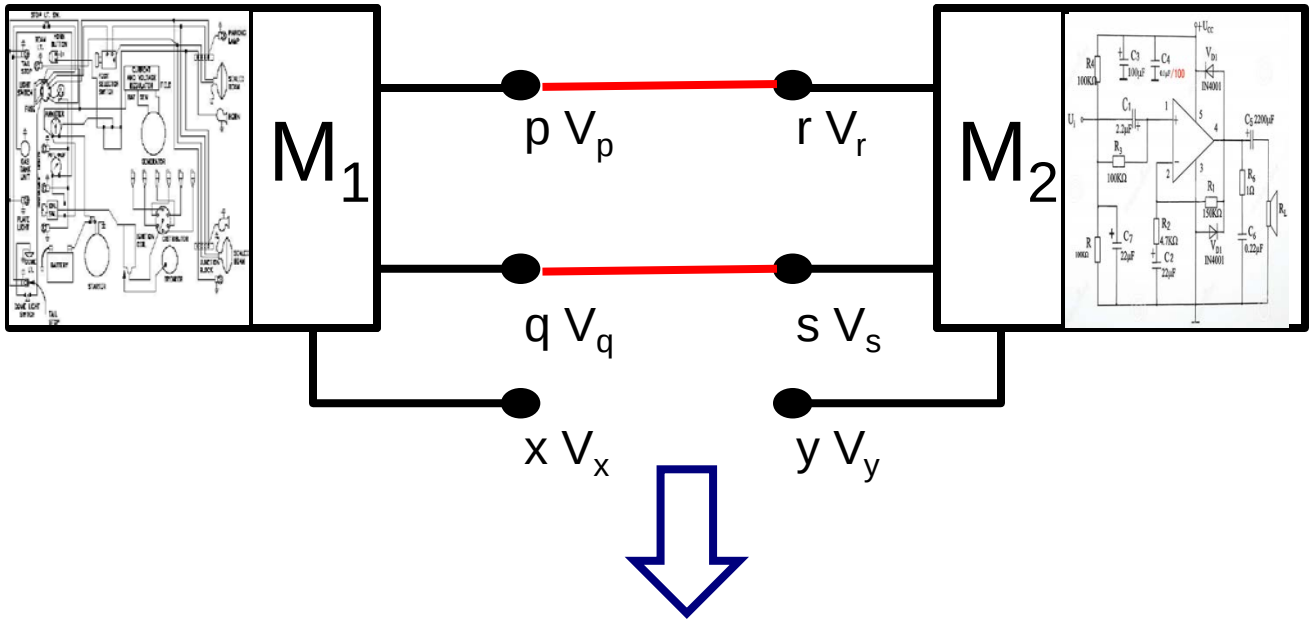


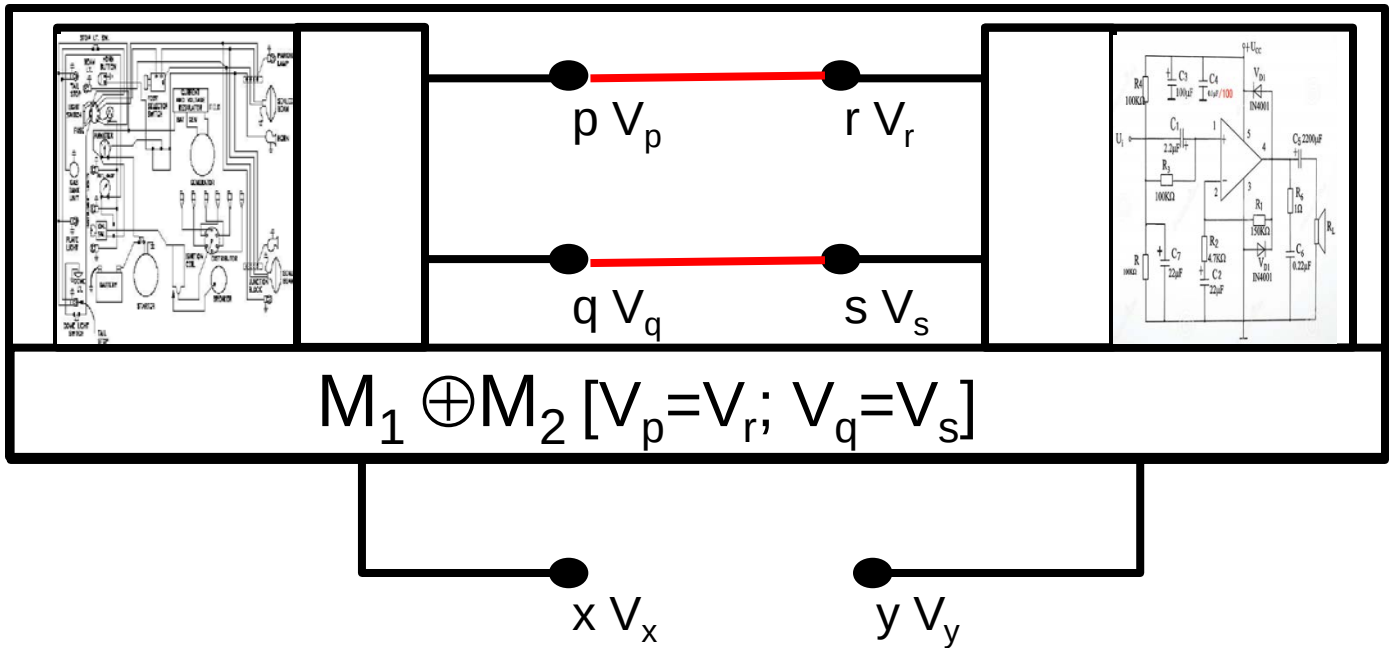
Multiscale multidomain integration of theories!

Component Complexity – Cyber physical Systems



Component Complexity – Cyber physical Systems



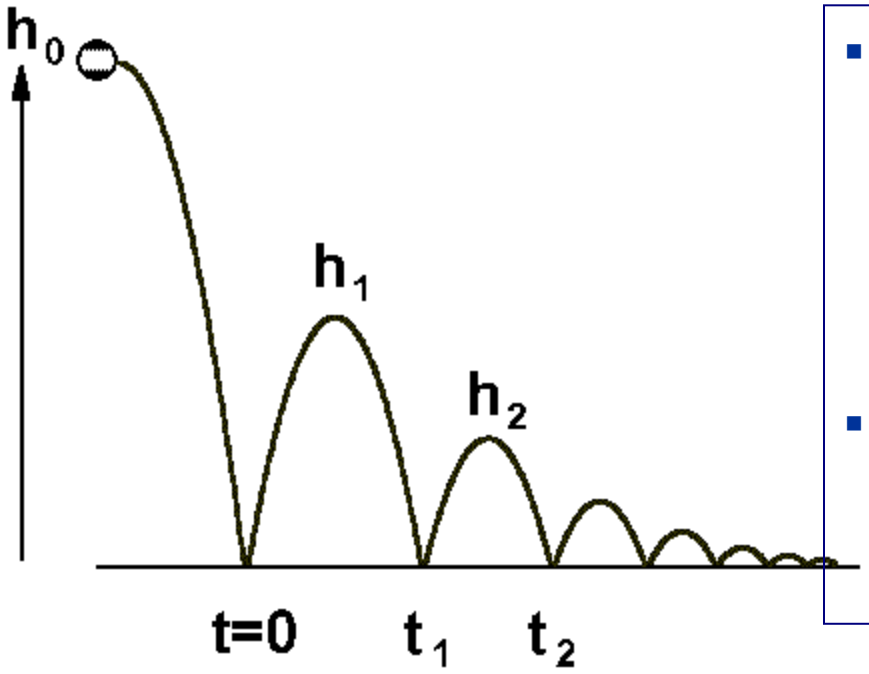


If M1and M2 are well-formed is the resulting component well-formed?

Component Complexity – Cyber physical Systems

One difficulty in simulating CPS is the existence of converging sequences of discrete events

- Consider a ball falling of an attitude h_0 and losing a percentage of its speed due to non elastic shock with the ground;
- The physical model is described by : $v' = -g$, $x'=v$, $v_{i+1}=0.8\ v_i$, $t_{i+1} - t_i = (2/g)v_i$
- $\lim_{tn\to\infty} (t_n - t_0)$ is bounded, so time will not exceed the Zeno point.



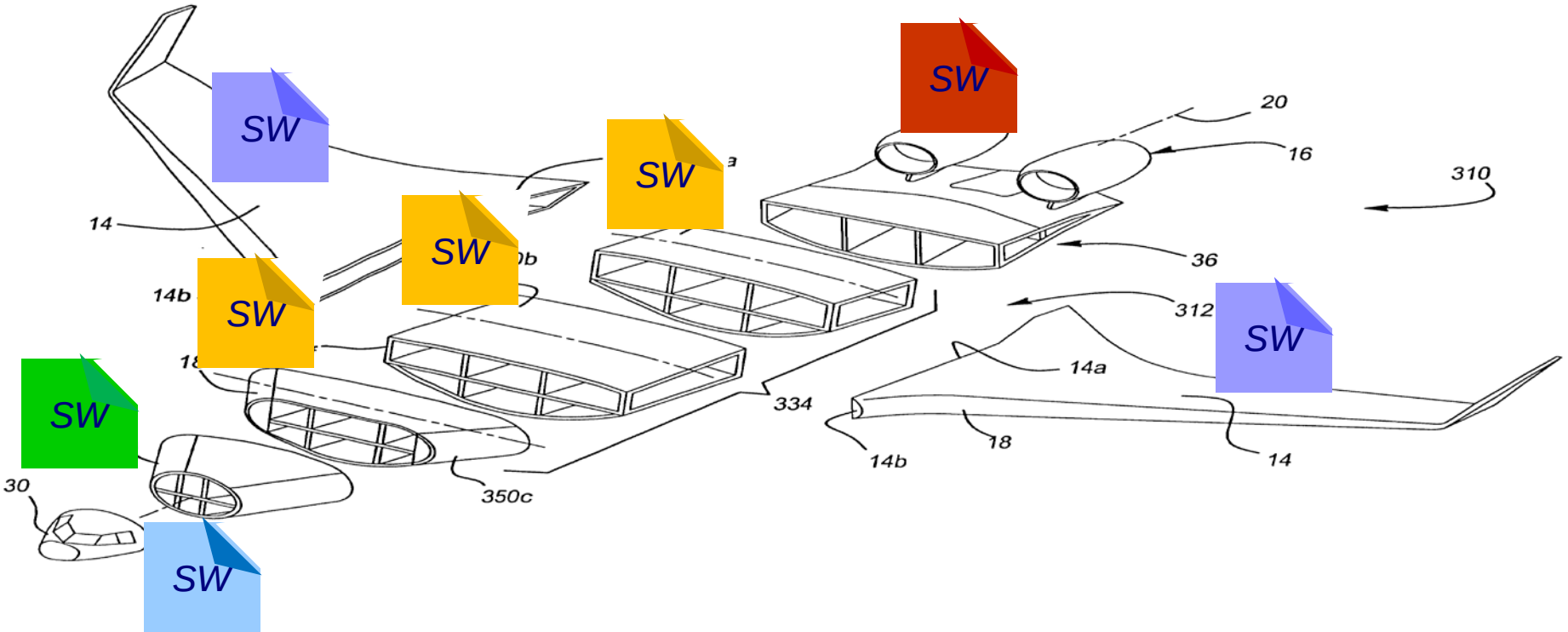
- Simulators provided with the physical model only will not converge – convergence should be enforced by the programmer!
- Proving convergence is undecidable as it requires discovery and application of an induction hypothesis

Component Complexity – Cyber physical Systems

Modular simulation: Can physical systems be effectively simulated compositionally in a distributed manner i.e. by replacing

- each physical component by a solver simulating its behavior
- each physical connector by a protocol involving the connected components

If at all possible, the communication overhead is prohibitive even for simple systems

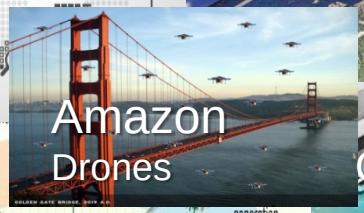


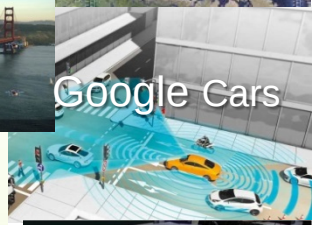
OVERVIEW

- System Design Trends&Challenges
- About System Design
- Two Pillars of System Design
- Component Complexity
- Architecture Complexity
- Knowledge-based Design
- Discussion

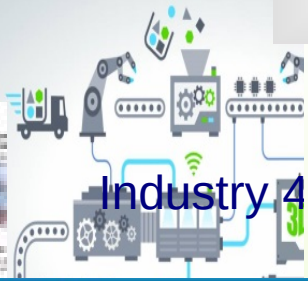
Design Complexity

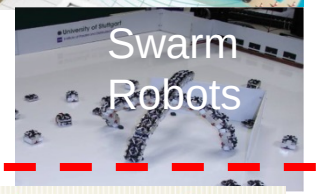
Agricultural Robot

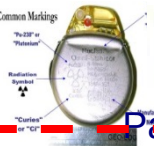




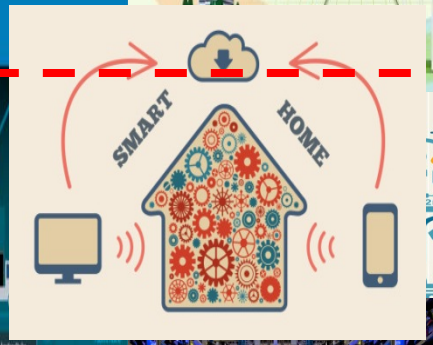
Active safety







Pacemaker



COMPONENT BEHAVIOR COMPLEXITY

CyberPhy

Embedded

Streaming

Transformational

SYSTEMS

SERVICES

Static parametric distributed reconfigurable mobile self-organizing

ARCHITECTURAL COMPLEXITY

Architecture Complexity – A Key Concept

Architectures

- depict design principles, paradigms that can be understood by all, allow thinking on a higher plane and avoiding low-level mistakes
- are a means for ensuring global properties characterizing the coordination between components – correctness for free
- Using architectures is key to ensuring trustworthiness and optimization in networks, OS, middleware, HW devices etc.
- We need to formalize and apply these design principles consciously from the start instead of rediscovering them each time.

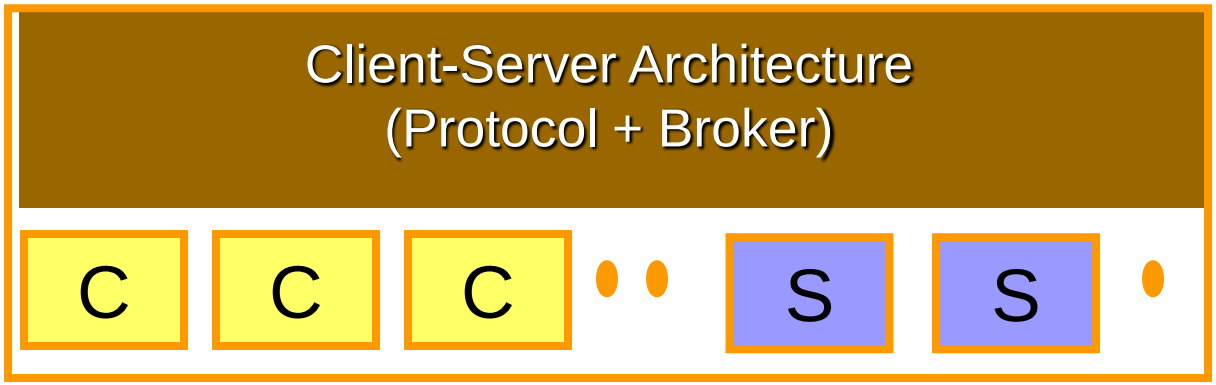
System developers extensively use libraries of standard reference architectures -- nobody ever again have to design from scratch a banking system, an avionics system, a satellite ground system, a web-based e-commerce system, or a host of other varieties of systems.

- Time-triggered architectures
- Security architectures
- Fault-tolerant architectures
- Adaptive Architectures
- SOAP-based architecture, RESTful architecture

Architecture Complexity – What is an Architecture?

An architecture can be viewed as a parametric operator that take as arguments instances of components types and builds a compound component meeting a characteristic property.

- A client-server architecture should enforce atomicity and resilience of transactions for any numbers of clients and servers.



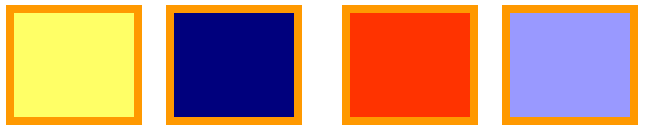
Architectures are characterized by:

- The type of components involved, in particular synchronous or asynchronous
- The coordination primitives and mechanisms used
- The structure of the coordination flow e.g. topology and dynamism

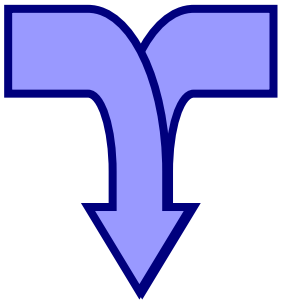
Architecture Complexity – Correctness by Construction

Rule1: Property Enforcement

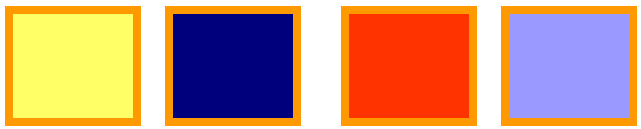
Architecture
for Mutual Exclusion



Components



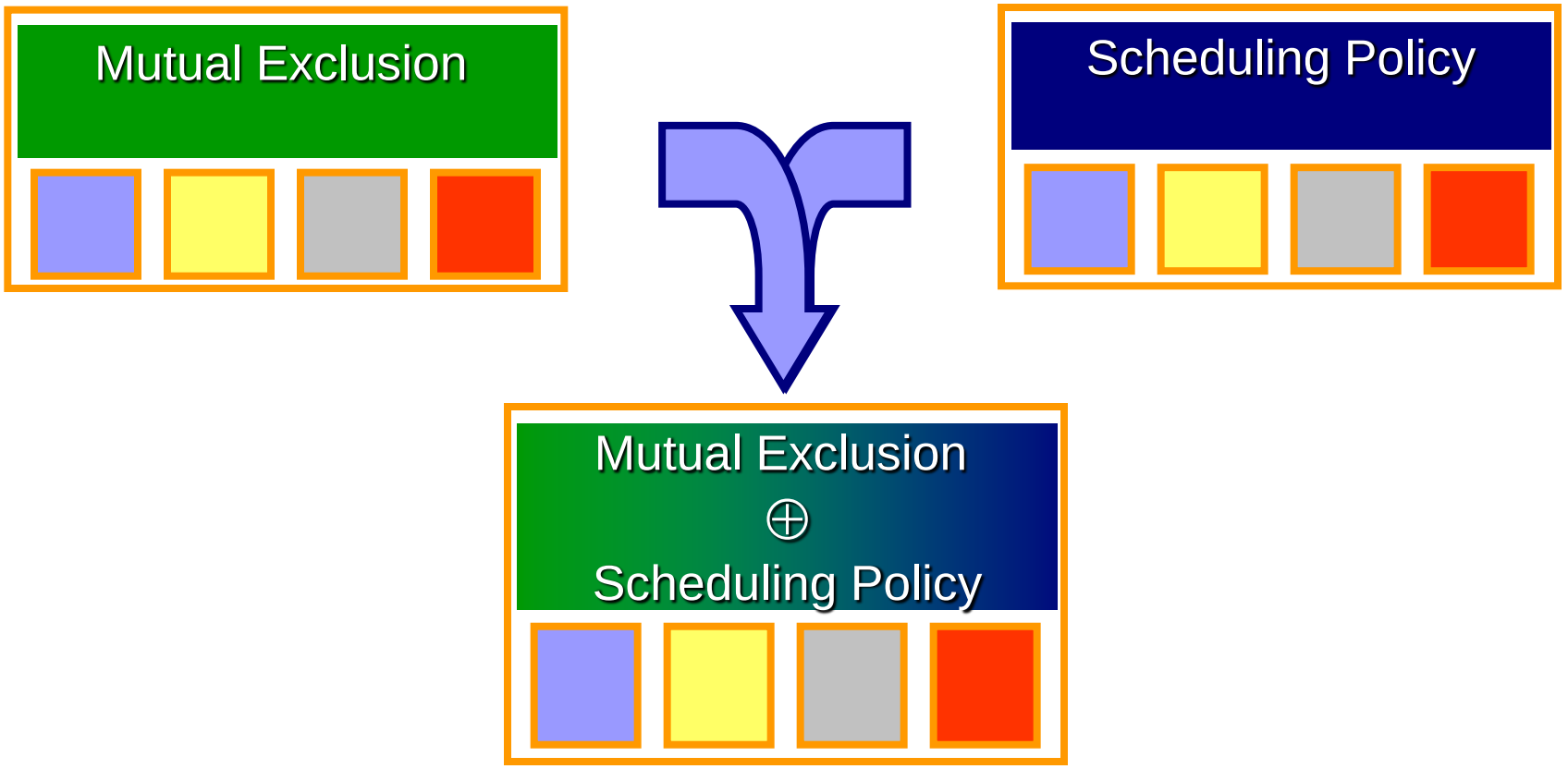
Architecture
for Mutual Exclusion



satisfies Mutex

Architecture Complexity – Correctness by Construction

Architecture Composability



Feature interaction in telecommunication systems, interference among web services and interference in aspect programming are all manifestations of lack of composability

Architecture Complexity – Correctness by Construction

Fully customizable smartphones *:*

- *Google's Project Ara was being designed to be utilized by "6 billion people":*
- *The smartphone consists of an endoskeleton and modules. A module can be anything, from a new application processor to a new display or keyboard, an extra battery, a pulse oximeter or something not yet thought of!*

TECH INNOVATION

Project Ara: Inside Google's Bold Gambit to Make Smartphones Modular

Tech Guru Daily

WILL PROJECT ARA WORK? THE GOOD AND BAD OF GOOGLE'S PLAN TO TURN PHONES INTO LEGOS

By **Simon Hill** — April 26, 2014

Debated: Google's Project Ara Will Likely Fail

May 2, 2014 - 00:31 by Rob Enderle

Google confirms the end of its modular Project Ara smartphone

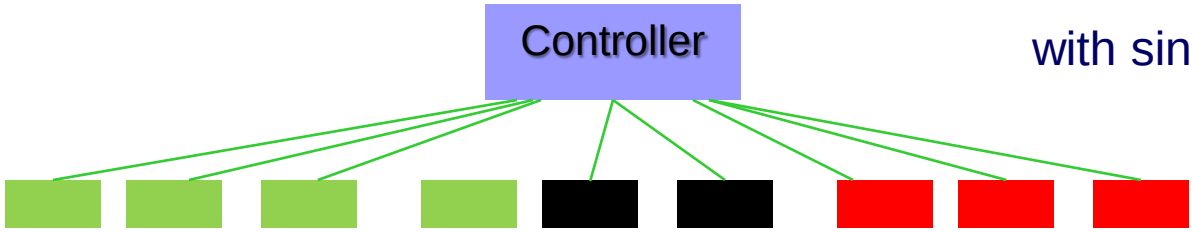
49

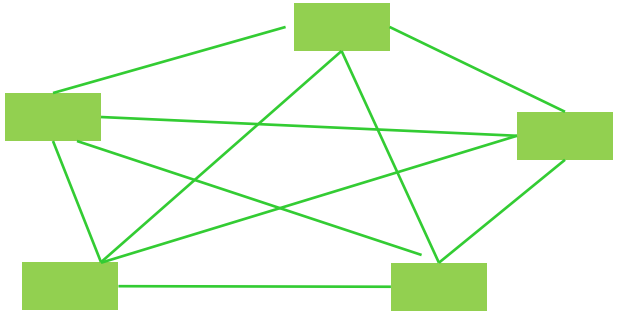
No mods for you

by Nick Statt | @nickstatt | Sep 2, 2016, 3:30pm EDT

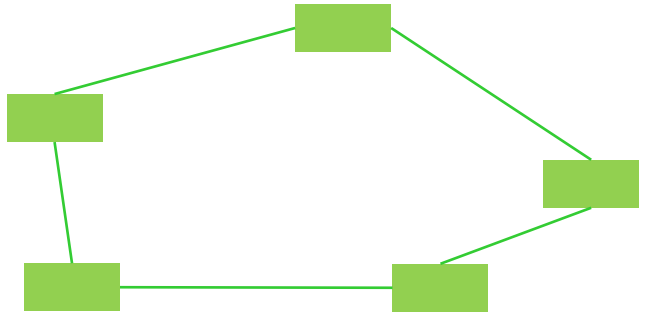
Architecture Complexity – Topology

Centralized Architecture
with single Controller





Clique Architecture
e.g. distributed mutual exclusion

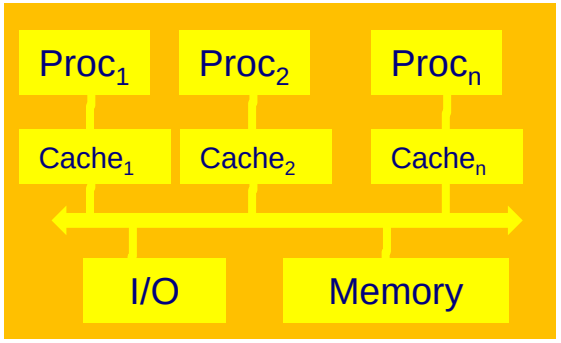


Ring Architecture
e.g. token ring, dinning philosophers, pipelines

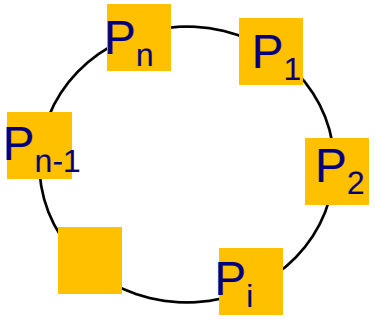
- Parametric systems can be formalized using 1st order or 2nd Order Interaction Logics
- In contrast to static verification, there exist no general methods and tools for parametric verification
- Most theoretical results concern non decidability even for simple properties and finite-state components

Architecture Complexity – Degree of Dynamism

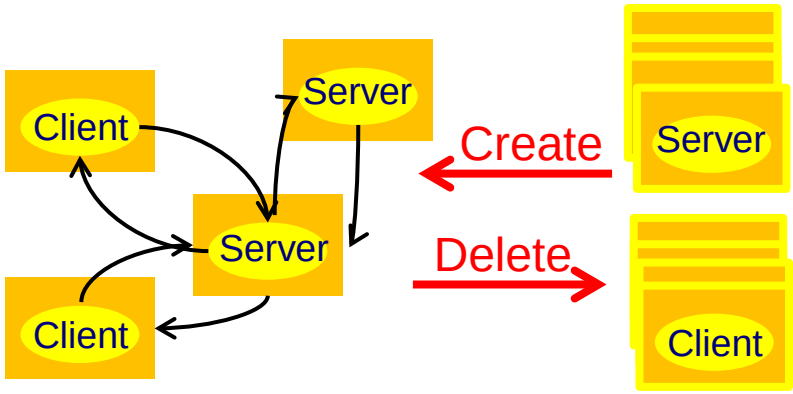
How much involved is the coordination between components?



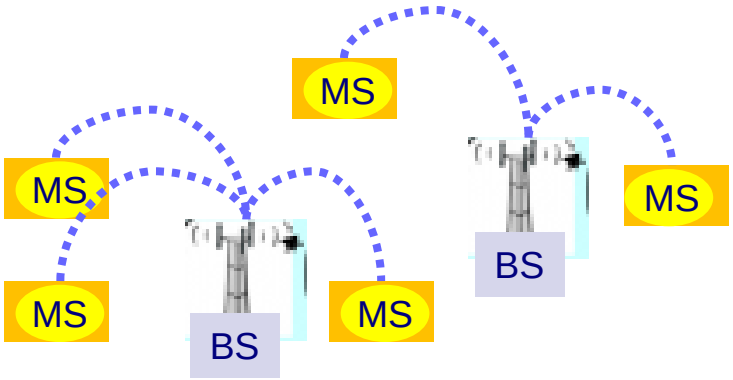
Static Interaction:
Multiprocessor System



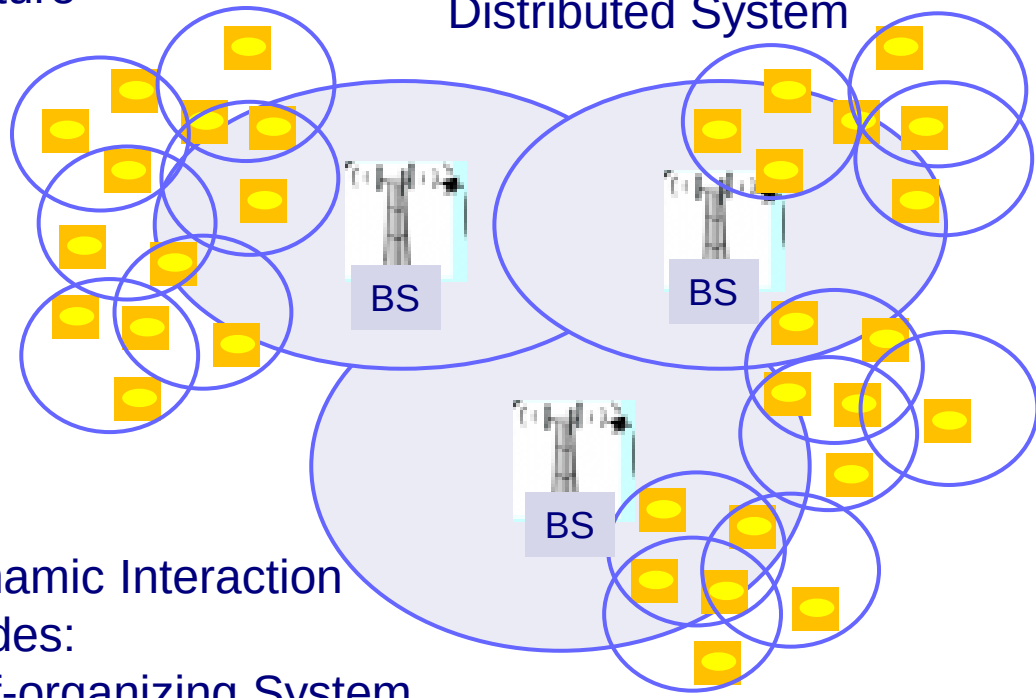
Parametric Interaction:
Ring Architecture



Dynamic Interaction:
Distributed System



Dynamic
Interaction+ Environment:
Mobile System



Dynamic Interaction
Modes:
Self-organizing System

OVERVIEW

- System Design Trends&Challenges
- About System Design
- Two Pillars of System Design
- Component Complexity
- Architecture Complexity
- Knowledge-based Design
- Discussion

KB Design – Sources of Uncertainty

- Uncertainty is the situation involving ambiguous or missing information.
- Systems must ensure a service meeting given requirements despite uncertainty:

❑ In the external environment

- non deterministic behavior e.g. varying throughput, workload
- attacks, malevolent actions
- dynamically changing neighborhood for mobile systems
- ambiguity of stimuli and inputs e.g. reduced using learning techniques

❑ In the execution environment

- variability of HW due to manufacturing errors or aging
- varying execution times due to layering, caches, speculative execution
- timing anomalies (decreasing performance for increasing platform speed)
- uncertainty due to distributed execution (partial state, message losses)
- updates, reconfiguration, dynamically changing architecture

How can we use knowledge to cope with uncertainty and compensate the lack of predictability in system design?

KB Design – What is Knowledge?

Knowledge is information that is a truthful relation that can be used to understand a situation and predict an event, or can be used to solve a problem.

- A priori or posteriori
  - A priori knowledge is independent from experience such as mathematics such as the Pythagorean Theorem
  - A posteriori knowledge depends on observation and experiments such as scientific knowledge but also empirical knowledge developed by neural systems

- Declarative or procedural
  - Declarative knowledge is truthful system properties in a logic e.g. Newton's laws, program invariant
  - Procedural knowledge can be used to solve problems e.g. decision procedure, algorithm, design technique, cooking recipe

- Theoryful or theoryless (Leslie Valiant, PAC)
  - Theoryful knowledge is theoretically justified based on the use of rigorous models and methods
  - Theoryless knowledge is empirically acquired and validated knowledge e.g. machine learning, driving a car, knowledge used by automated thinking

KB Design – Predictability

Model

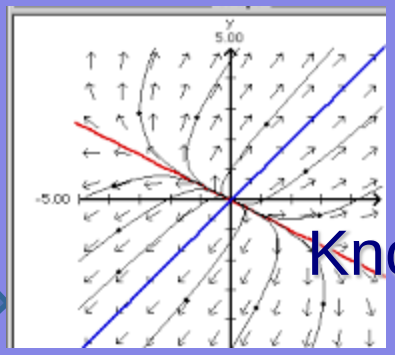
$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2} + \frac{Q(x,t)}{c\rho}$$

$$\frac{\partial^2 u}{\partial t^2} = c^2\frac{\partial^2 u}{\partial x^2}$$

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \nabla^2 u = 0$$

$$\frac{\partial u}{\partial t} - 4\frac{\partial^2 u}{\partial t^2} = \frac{\partial^3 u}{\partial x^3} + 8u - g(x,t)$$

Computational Complexity

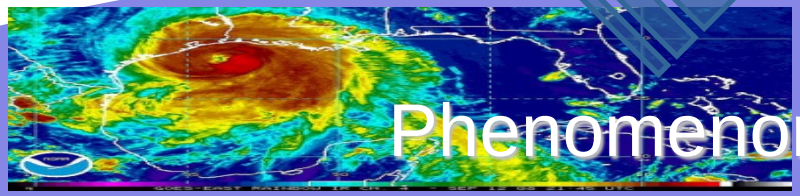


Knowledge

Information

Epistemic Complexity

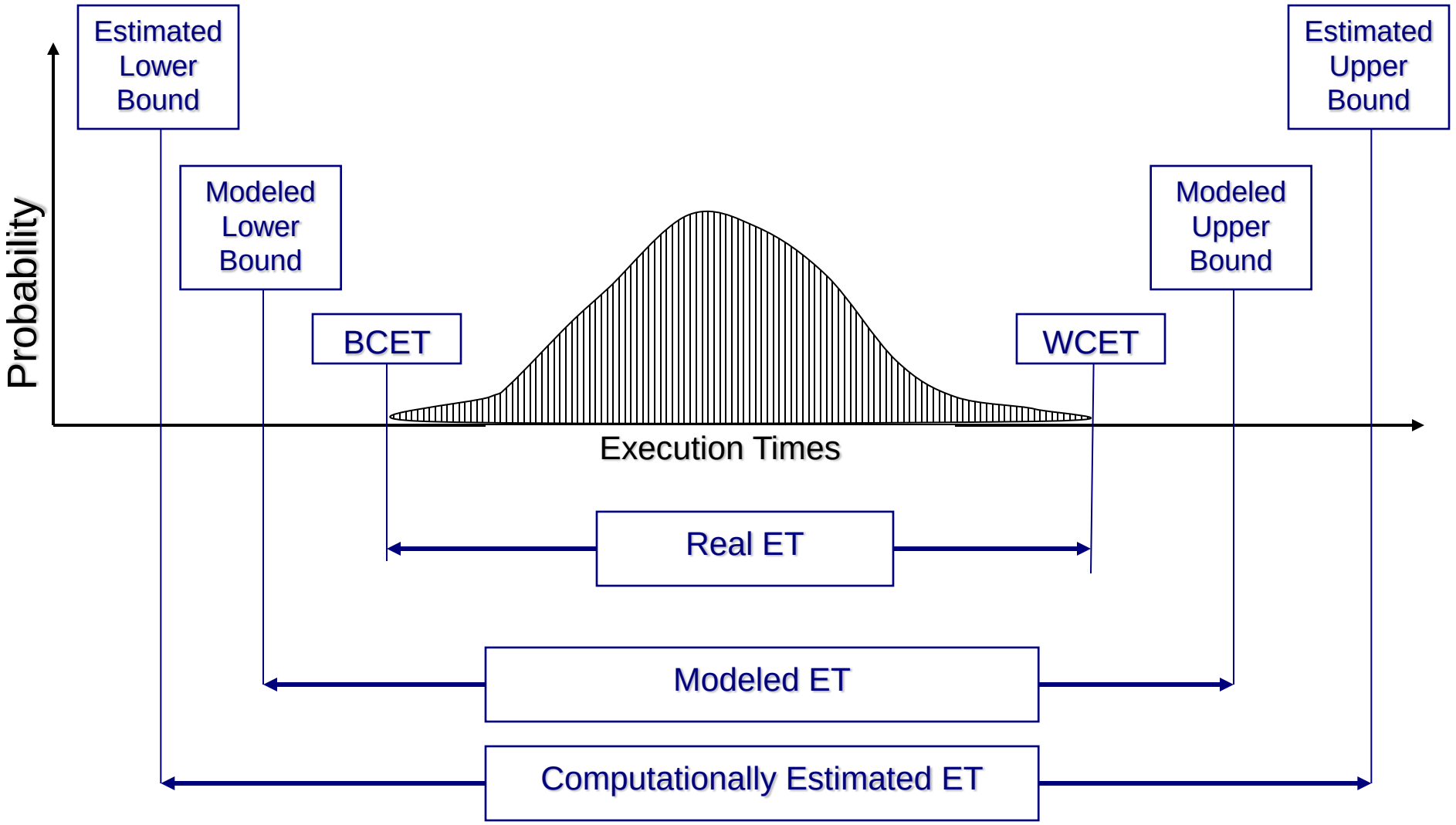
Predictability



Phenomenon

Material World

KB Design – Predictability of WCET

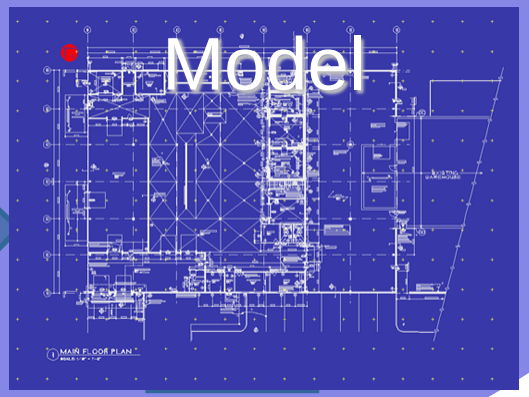


WCET : Worst Case Execution Time
BCET: Best Case Execution Time

KB Design – Designability

Needs in Natural Language

Linguistic Complexity

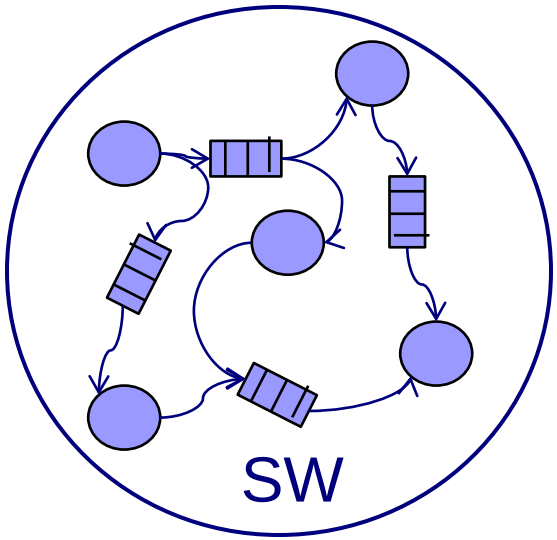


Designability

Computational Complexity

Material World

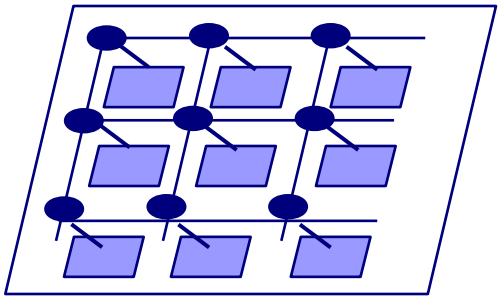
KB Design – Designability



⊓

Deployment

⇓⇓



Platform

Problem:
find a trustworthy and optimal SW deployments (mapping and scheduling) for a given platform

Cyclic Dependency :

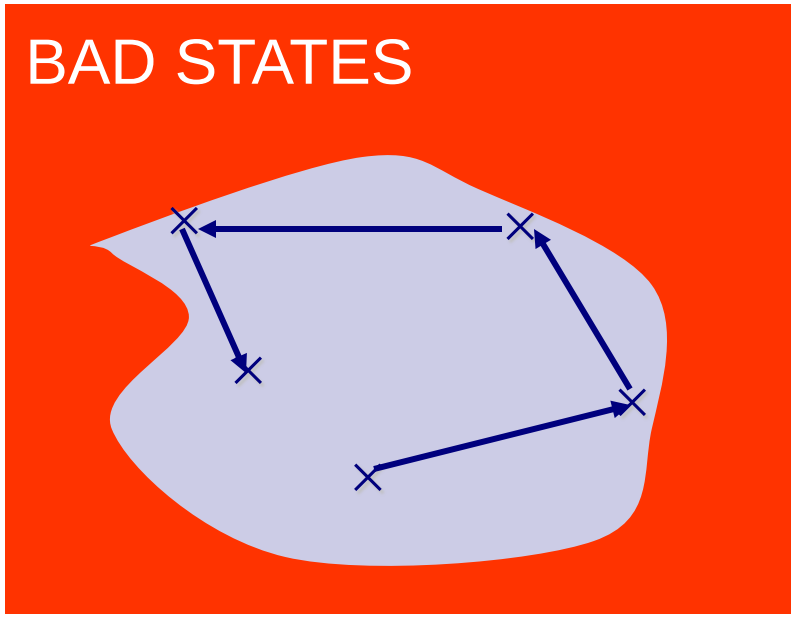
- Deployment trustworthiness (safety , security) and optimization can be assessed only for known WCET, latency, failures modes and threats
- WCET, latency, failure modes and threats can be analyzed only for known deployments, for example.

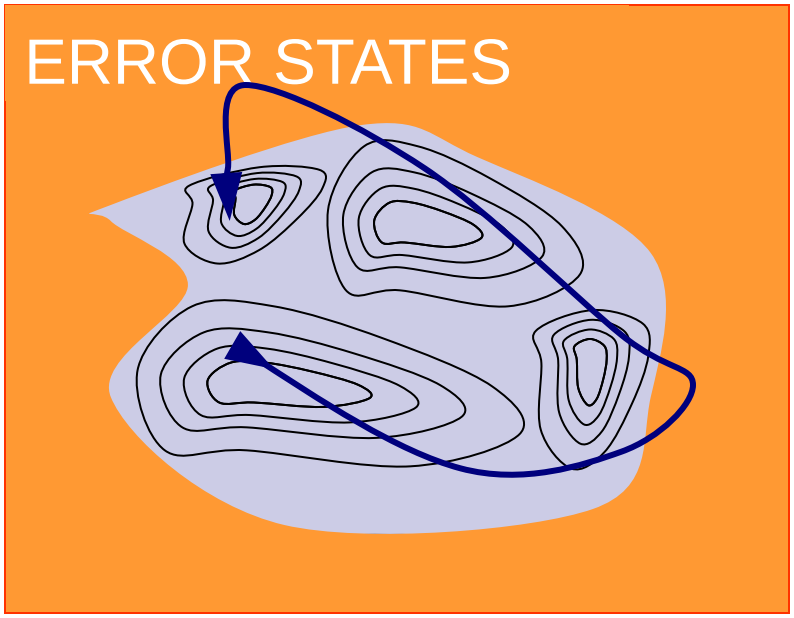
WCET=WCET_in_isol + Interference delay

Cyclic dependency can be broken for simple monolithic systems e.g. flight control SW for all Airbus types except A380 run on bare metal

KB Design – Critical vs. Best Effort Engineering

The divide between Critical and Best-effort systems engineering is not affordable anymore e.g. in car industry





Critical systems design based on worst-case analysis and static resource reservation e.g. hard real-time approaches, massive redundancy.

Leads to over-provisioned systems.

Best effort design
based on average case analysis and dynamic resource management e.g. QoS for optimization of speed, memory, bandwidth, power;
- no guaranteed availability.

KB Design – Adaptive Systems

Non predictability of all bad situations due to inherent uncertainty in the design process implies that it is impossible to guarantee at design time trustworthiness and optimality of systems.

- One avenue is to try to make critical systems more predictable by reducing intrinsic and estimated uncertainty
  - Simplify HW architectures e.g. no caches, no out-of-order execution
  - Enforce time-deterministic observable behavior e.g. time triggered systems

- The other avenue would be to integrate in designs adaptive monitoring and control mechanisms. For example,
  - To ensure security, early warning mechanisms based on machine learning are used to detect abnormal situations e.g. intrusion, and take measures to mitigate their effect.
  - For systems integrating both critical and best effort services adaptive controllers handle a sufficient amount of global resources to
    1) satisfy first and foremost critical properties;
    2) and secondarily, to handle optimally the available resources for best-effort services.

Adaptive techniques find application in many areas including networks, data mining, multimedia systems.

KB Design – Adaptive Controller

Adaptive systems use control-based techniques to ensure correctness despite unpredictable events and hazards.

Adaptive techniques need to be further studied and improved:

- false positives,
- varying latency,
- costly learning and data analysis,
- lack of theory to assess their dependability.

KB Design – Principles

Knowledge-based design aims at building systems such that the overall knowledge about their actual and intended behavior is used to ensure their correctness

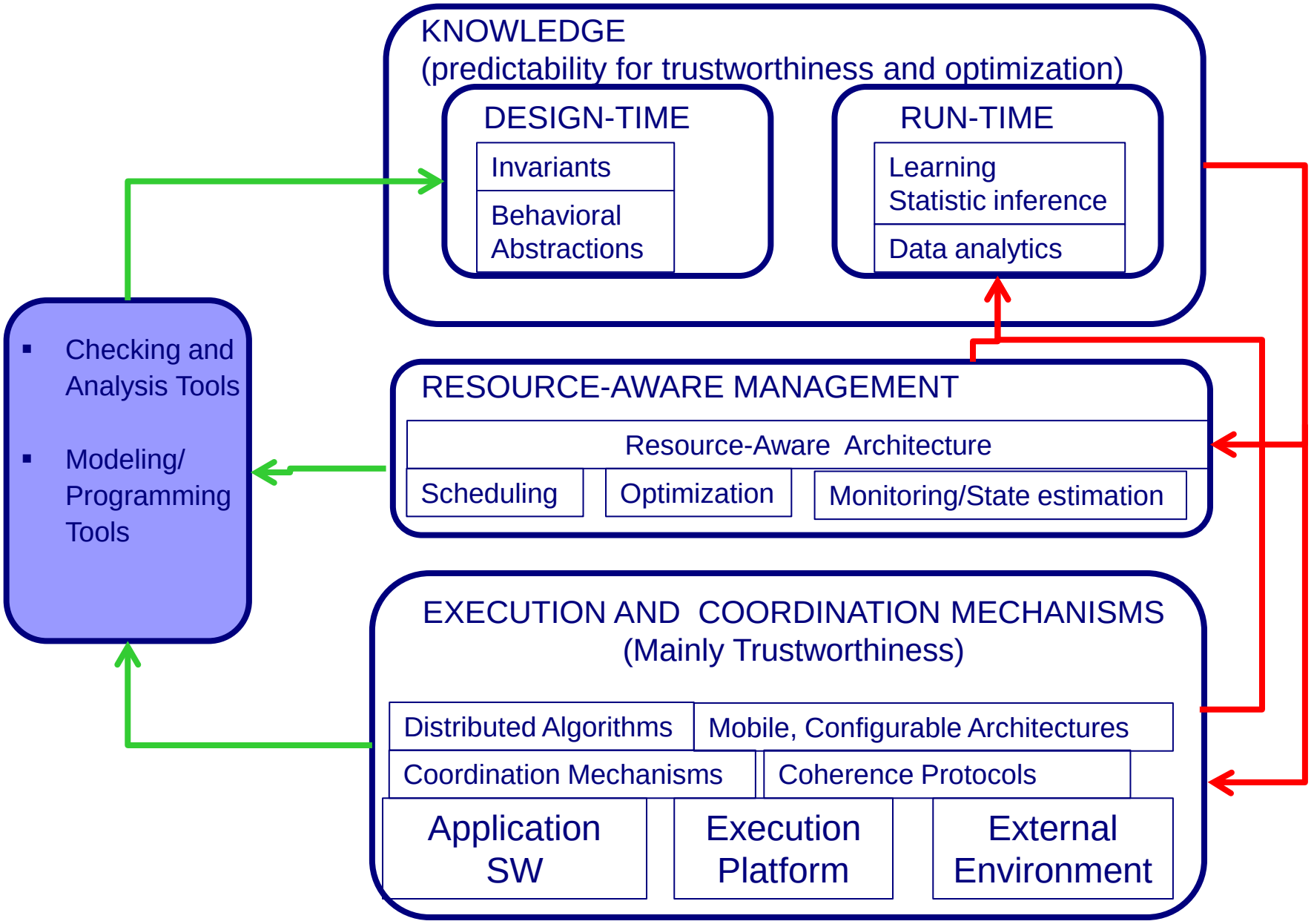
- Design-time Knowledge:
  Knowledge about the intended behavior e.g. properties established at some step of the design flow and are guaranteed to remain true forever e.g. invariants or procedures, architectures that have been proven correct.

- Run-time Knowledge: Observed relationships and measurements used to
  - complement design time knowledge e.g. more precise estimates of execution times,
  - detect abnormal behavior e.g. early warning security system

- Challenge:

  - design techniques allowing to clearly distinguish between properties enforced by design and properties are left to be enforced at run time
  - analyzing semantic information and upgrade it to knowledge
  - effective and efficient use of both design and run-time knowledge e.g. low computational overhead for significant gain in trustworthiness and optimality

KB Design – Architecture



- System Design Trends&Challenges
- About System Design
- Two Pillars of System Design
- Component Complexity
- Architecture Complexity
- Knowledge-based Design
- Discussion

OVERVIEW

Discussion – Standards for Autonomous Systems

- Current safety standards such as ISO26262 and DO178B, allow a quality check of the development process and require *conclusive evidence* that the system can cope with any type of mishap that can cause catastrophes.
  Although they cannot guarantee absence of bugs.
- Nonetheless,
  - they cannot handle machine learning software – the learning process is inductive describing inferences from a finite sample of data to a generalization about certain objects;
  - they cannot handle design flows for autonomous systems – they give a system credit for a human driver ultimately being responsible for safety.
  - Stringent predictability requirements make their application impossible to IoT autonomous systems.

- Consequently, there is no Independent safety certification for autonomous systems
  - Although it is customary for even home appliances like toasters to be certified by independent labs, the automotive and medical device industry are exempted from certification obligations.
  - Products are self-certified by manufacturers following guidelines that determine how to provide *sufficient evidence* that the developed system is reliable enough e.g. Federal Automated Vehicles Policy issued by DOT and NHTSA.

Discussion – The Way Forward

- Future autonomous systems will not be designed as classical critical systems
- "How good is good enough ? - *Is it possible to guarantee safety by testing?*
- Interesting discussion in *"On a Formal Model of Safe and Scalable Self-driving Cars"* Shai Shalev-Shwartz, Shaked Shammah, Amnon Shashua, Mobileye, 2017

  The paper shows *"why a statistical approach to validation of an AV system is infeasible, even for validating a simple claim such as "the system makes N accidents per hour". This implies that a model-based safety definition is the only feasible tool for validating an AV"*
- We need rigorous novel design methodologies for open autonomous interconnected systems involving embedded supercomputers, AI algorithms and receiving updated data from the Cloud.
- We need new trustworthiness assessment techniques and standards for third party certification

Discussion – The Way Forward

Two possible scenarios

1) Gradual transition

- from few robo-cars to many robo-cars
  - mixed traffic of robo-cars and human-driven cars is too hard to manage!
- from SAE safety level 3 to levels 4 and 5.
  - what it means that a car is partially autonomous?

  At level 3 *"the human driver must be ready to take back control when the automated system requests"*
- *"The irony of automation" : as tasks are automated, and hence taken away from the operators 1) this reduces their manual control skills in factories, stock markets, aircraft; 2) this reduces their skills in diagnosing problems and mitigating their consequences*

2) Forced transition toward autonomous systems based on statistic arguments – systems can manage better than humans dangerous situations.

> *"In the distant future, I think people may outlaw driving cars because it's too dangerous" (*Elon Musk).

Discussion – Summary

The trend for autonomous systems in the framework of IoT renders completely obsolete current critical systems design practice and standards

- Validation by testing and purely statistic-based approaches seem to be the last resort – but by their nature they cannot provide conclusive evidence and guarantees.

- The alternative would be to develop rigorous design flows supported by standards that pragmatically distinguish between correctness at design time and correctness at run time based on two pillars

1. Replace the V-model by “agile techniques” relying on both verification and “correctness by construction” through the use of predefined and provably correct architectures
2. Extensive use of knowledge
    - integration of design-time and run-time knowledge in autonomous system architectures
    - “Knowledge-aware” design flow,

Thank You