Design for Dependability

April 22, 2021

Joseph Sifakis

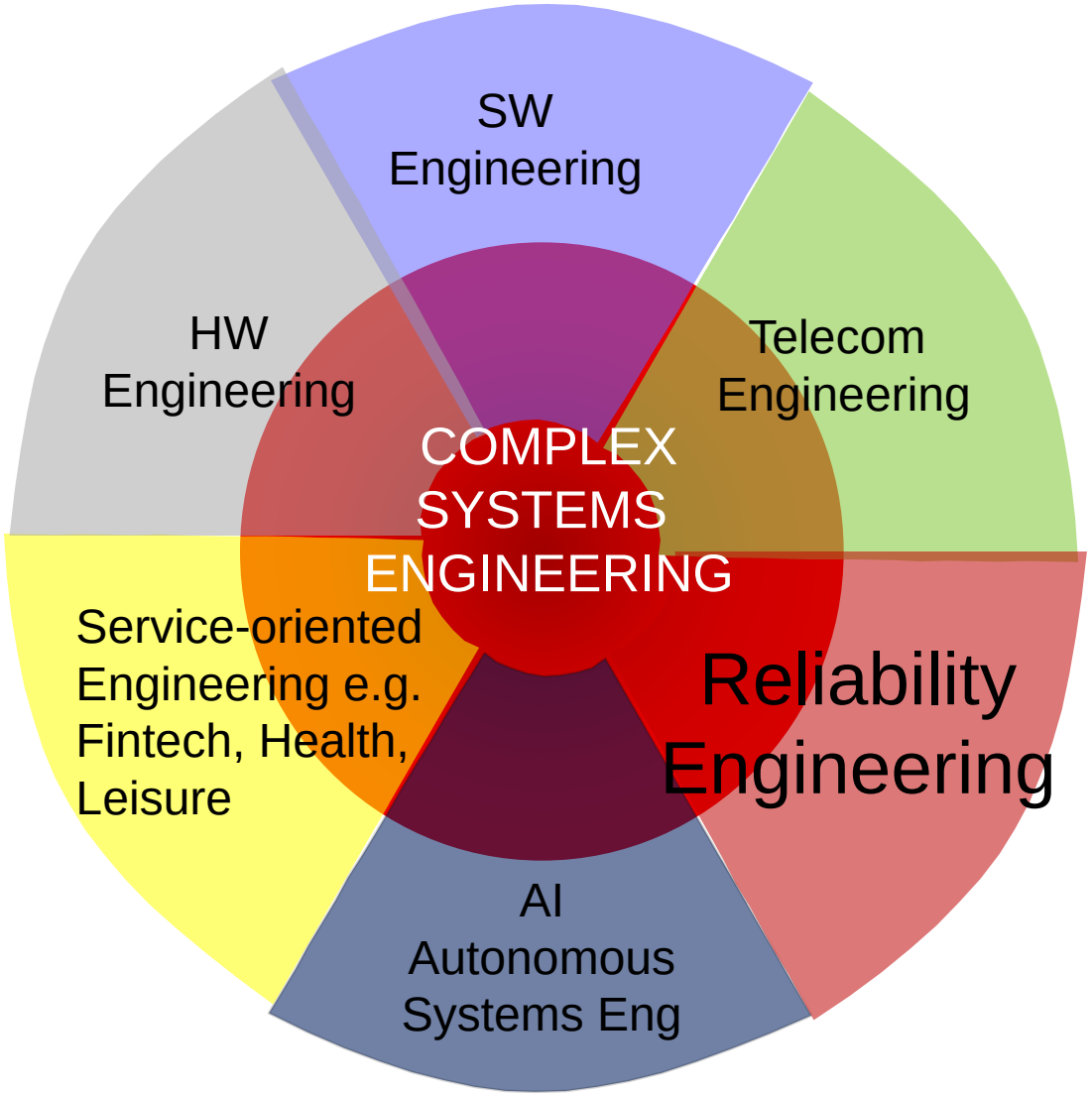
Design for Dependability – Complex Systems Engineering

- Complex Systems Engineering systems needs to integrate methods and tools from other engineering disciplines to face the IoT challenge.

- Complex Systems Engineering is at a turning point facing a huge gap moving from
  - Small Centralized, Automated, Predictable systems to
  - Large Decentralized, Autonomous. Unpredictable Systems

- Reliability Engineering has historically emerged as a sub-discipline of Physical Systems Engineering is often presented as something apart from the other system design activities.

- I will advocate further integration of Reliability Engineering methods and practices in Systems Design.

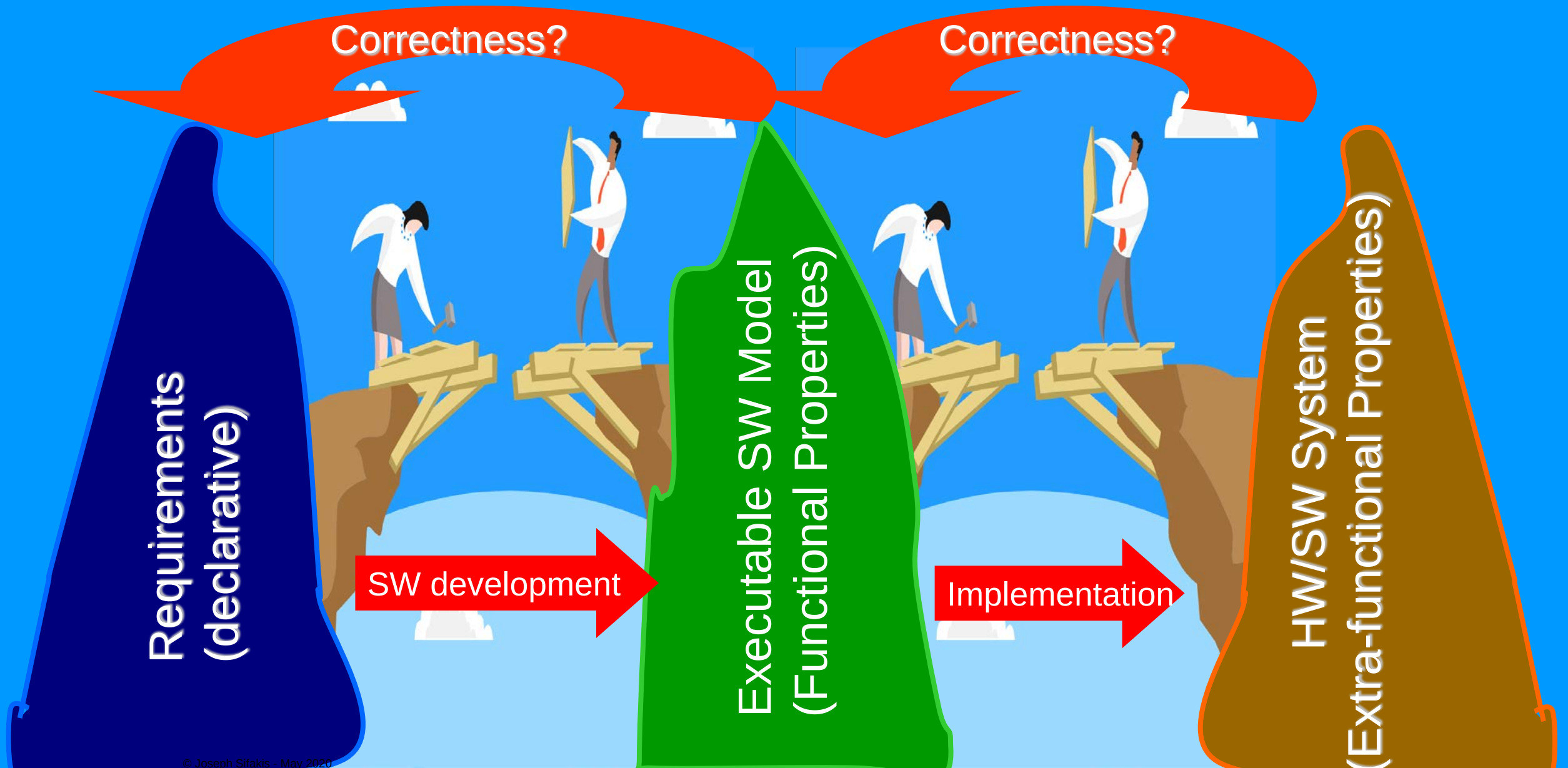


Acknowledgement: The presentation integrates conclusions of joint work with Huawei's Reliability Lab in Shenzhen on Design for Dependability

OVERVIEW

- System Design and Dependability
- Design for Dependability
  - Risk Analysis
  - Risk Mitigation
  - Risk Evaluation
- Discussion

Design – Two Gaps, Two Steps



Design – The Concept of Dependability

- Dependability expresses assurance that the designed system can be trusted that it will perform as expected despite any kind of hazard due to

HW failures

Design/Programming Errors

Environment Disturbances

Malevolent Actions

- About the concept
  - Dependability is a number quantifying a probability . BUT there is no general agreement on a definition of dependability – other terms such as **trustworthiness** are considered to be equivalent.
  - Usually list of attributes of uneven relevance is given to characterize the concept such as **Reliability, Availability, Maintainability, Integrity, Robustness, Safety, Accountability, Security, Performance**, etc.
- Dependability introduces a distinction between Nominal and Out-of-nominal conditions:
  - Nominal conditions define assumptions about an ideal system environment (execution and physical environment)
  - Out-of-nominal conditions consider situations where rare events violate nominal conditions

Dependability focuses on the system trustworthiness for out-of-nominal conditions

Design – Basic Properties vs. Dependability

NOMINAL CONDITIONS

Basic system properties (imply requirements under nominal conditions)

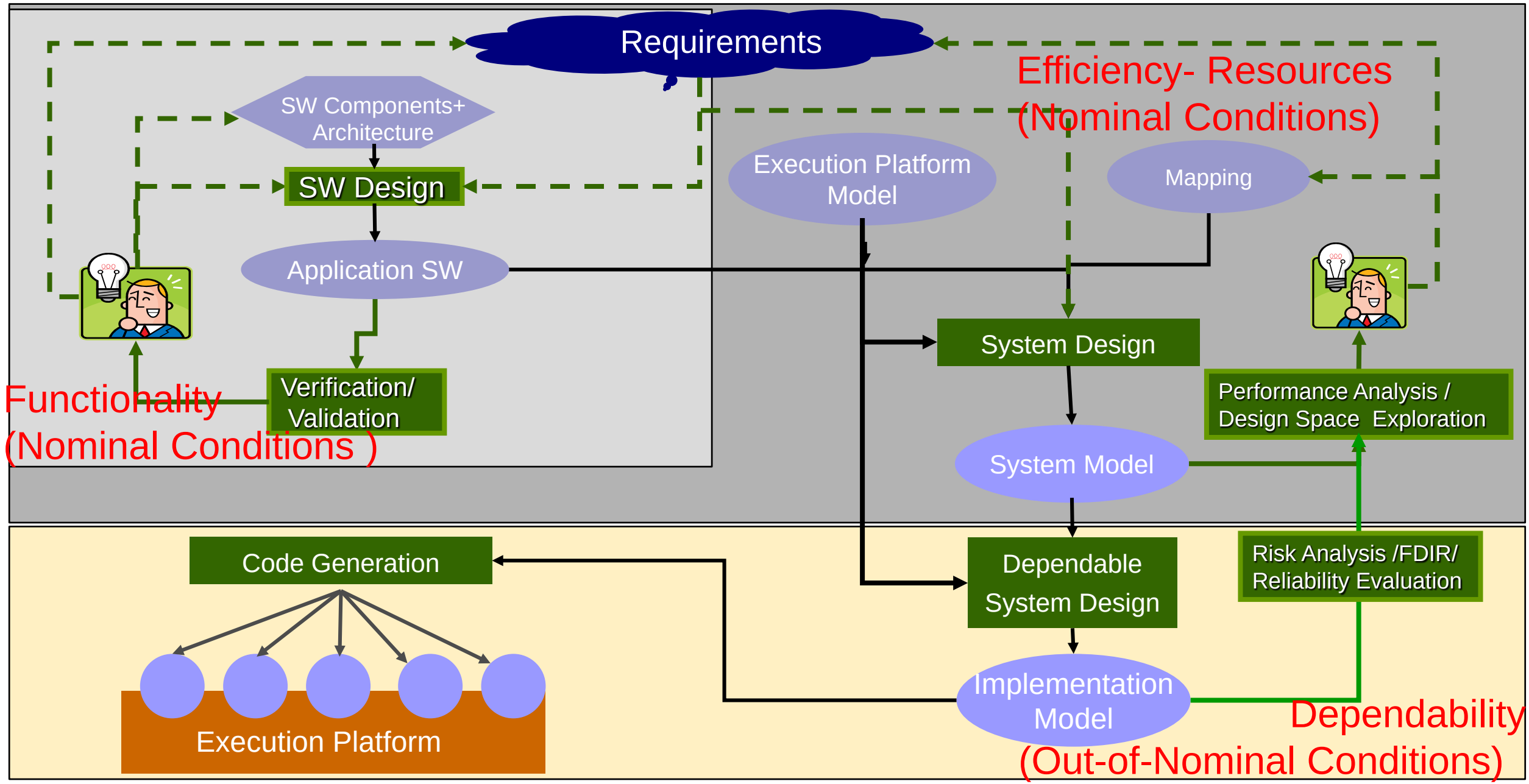
Nothing BAD can happen		Something USEFUL is done
Safety: The system is safe if no bad situations can be reached resulting in the violation of critical constraints e.g. temperature< limit. service_available	Security: The system is secure if it can cope with attacks and malevolent actions – non violation of a set of properties such as privacy, confidentiality and integrity	Efficiency: characterizes the useful work performed by the system using resources (memory, energy, time, money) based on two criteria 1) Performance: how well the system does wrt user demands e.g. throughput, jitter, latency, quality. 1)Cost: how well resources are used wrt economic demands e.g. storage efficiency, energy efficiency, processor utilizability,.

OUT-OF-NOMINAL

Dependability Attributes

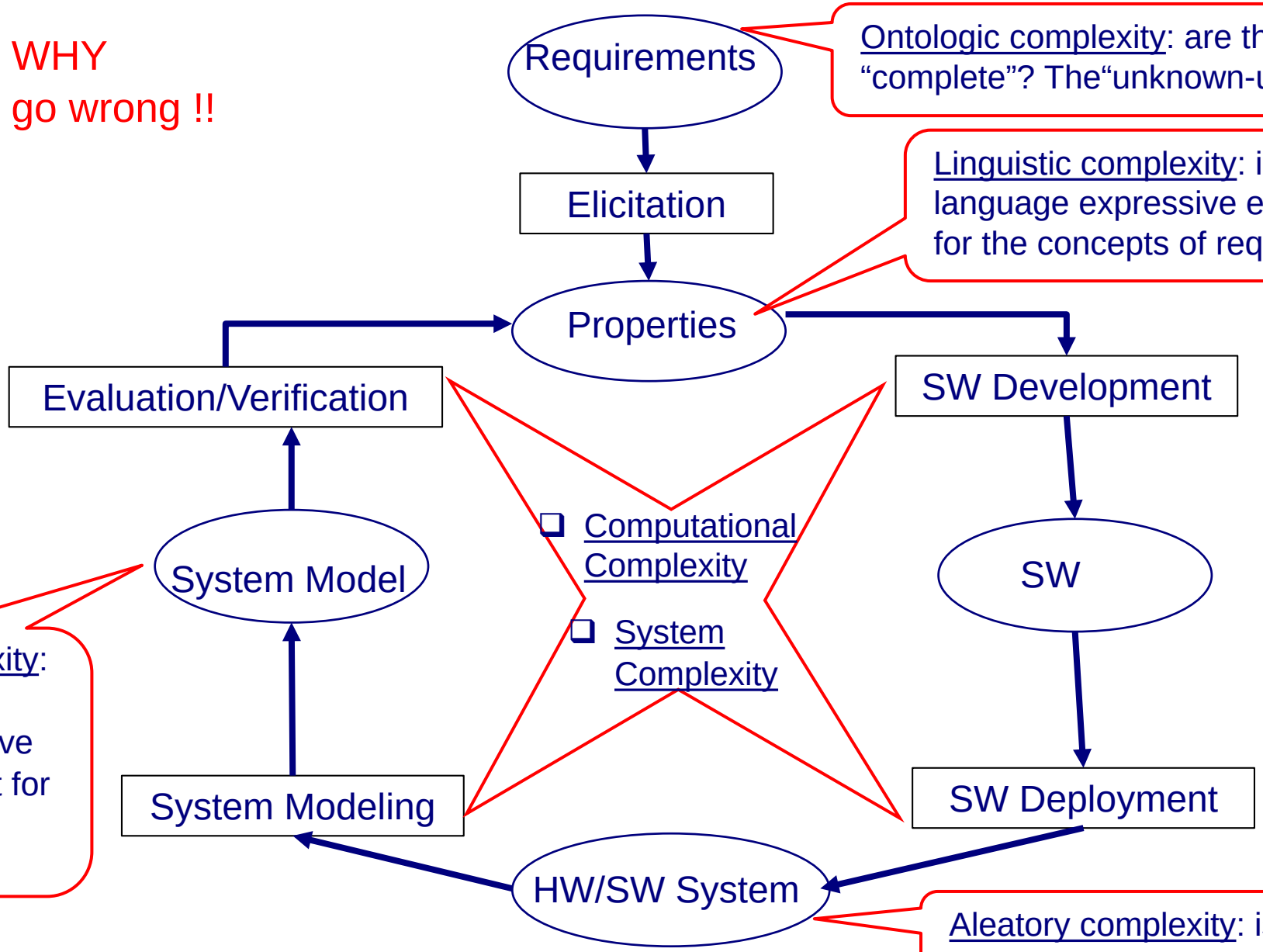
- focus on the estimation of the probability that events violating nominal conditions have an impact on basic system properties;
- reliability is the most important and hard to compute dependability attribute - all the others depend on it and can be derived with moderate effort.

Design – Separation of Concerns in the Design Flow



Design – Complexities

Understanding WHY
something can go wrong !!



Ontologic complexity: are the requirements "complete"? The"unknown-unknown"

Linguistic complexity: is the specification language expressive enough to account for the concepts of requirements ?

Epistemic complexity: is the modeling language expressive enough to account for system features?

Aleatory complexity: is the behavior of my system predictable – randomness of events

OVERVIEW

- System Design and Dependability
- Design for Dependability
  - Risk Analysis
  - Risk Mitigation
  - Risk Evaluation
- Discussion

Design for Dependability – General Flow

RISK ANALYSIS

Identify system hazards and estimate their likelihood in terms probabilities of their unintended causes

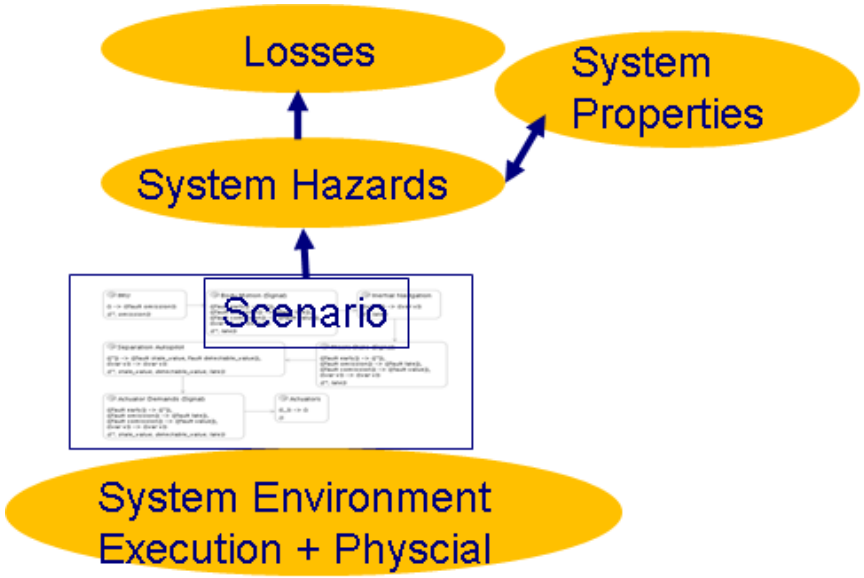
RISK MITIGATION

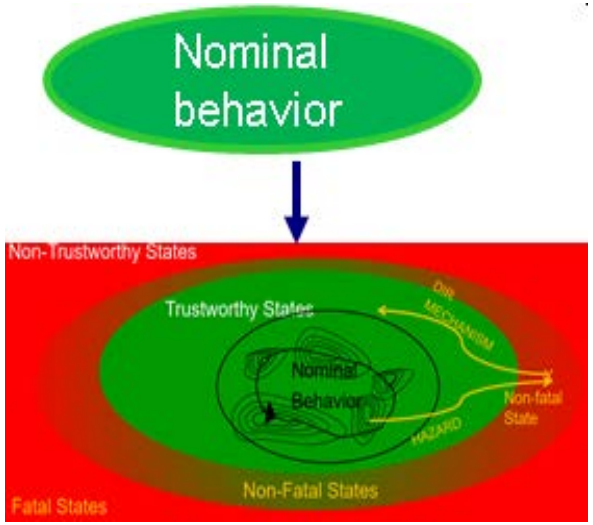
Based on Risk Analysis, design and implement for each hazard corresponding mechanisms for

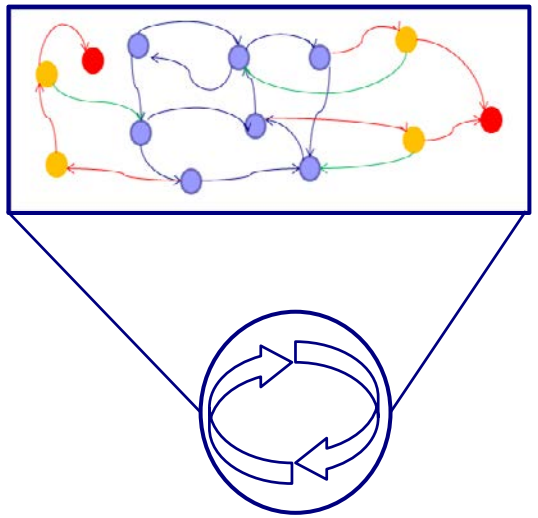
- Detection
- Identification
- Recovery

RISK EVALUATION

Given probabilities of hazards and their effect on the nominal system behavior, estimate the probability that the system meets a set of properties characterizing its correctness.







Risk Analysis – Flow and Terminology

Losses: consequences of system hazards including loss of human life or human injury, property damage. environmental pollution, loss of mission, loss of reputation, loss or leak of sensitive information etc.

System Hazards: any deviation from the expected system nominal behavior e.g. violation of safety or security as well as performance degradation,

EFFECTS

SYSTEM = (Architecture + Components) interacting with an Environment

Hazard (fault or threat)

Fault is a manifestation at component level of failures or errors

Security threat

CAUSES

Failure (random)
- due to aging in a VLSI
- failure of an engine

HW+Controlled Environment

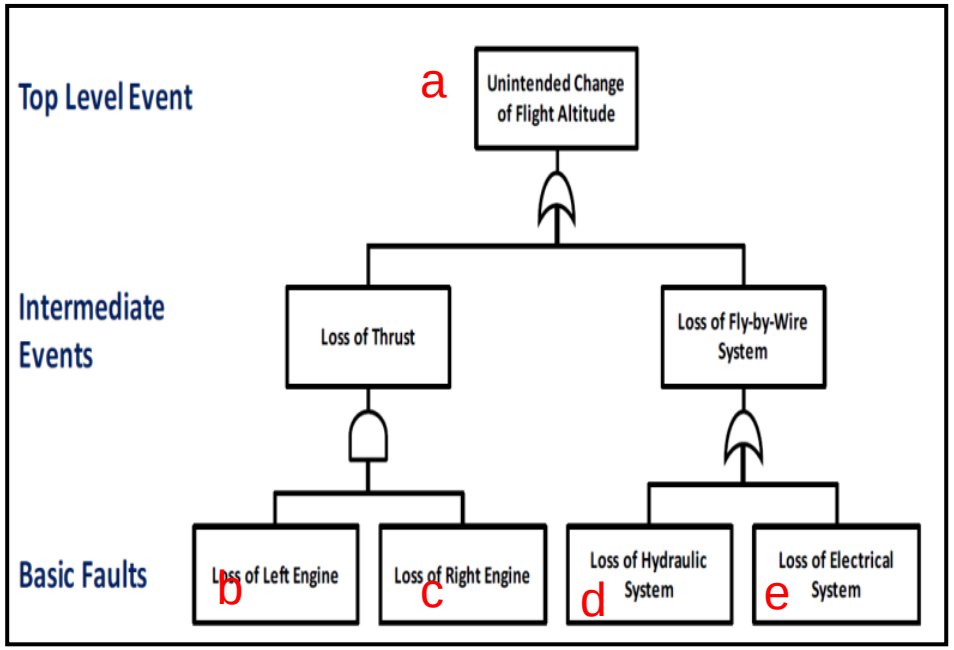
Error (non-intentional)
- Programming error
- Erroneous architecture

Built-in development defects

Security flaw (intentional)
- Exploited by a hacker
- Allowing malevolent action

Malevolent Human Environment

Risk Analysis – Fault-tree Analysis



- A Fault Tree is a Boolean Expression on basic fault variables

$$FT ::= bf \mid tle \rightarrow FT \mid FT \wedge FT \mid FT \vee FT.$$

- We can write $tle = \bigvee_{I \in FCS} \bigwedge_{i \in I} bf_i$

  where FCS= Fault Cut Set a set of faults such that their presence makes the top level event true.

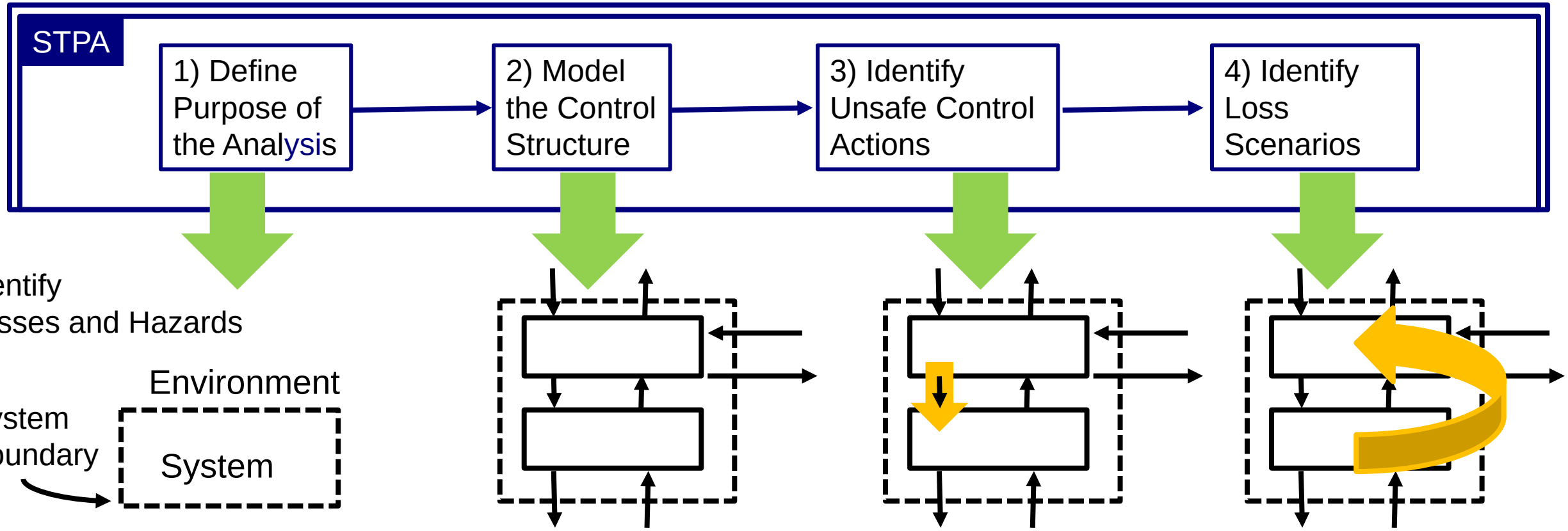
For a= bc $\vee$ d $\vee$ e the cut sets are {bc}, {d}. {e}

Given a Fault Tree we can compute probabilities under the assumption of independence between the basic faults.

- For a = bc $\vee$ d $\vee$ e we can compute Pr(a) as a function of Pr(b), Pr(c), Pr(d), Pr(e) using the rules :
  - $Pr(x \wedge y) = Pr(x)\ Pr(y)$
  - $Pr\ (x \vee y) = Pr(x) + Pr(y) - Pr(x)\ Pr(y)$
  - $Pr\ (\neg x\ ) = 1 - Pr(x)$
- Thus Pr(a) =

  $Pr(\neg\ (\neg\ b \vee \neg\ c)\ \neg\ d\ \neg\ e)) = 1 - Pr(\neg\ b \vee \neg\ c)\ \neg\ d\ \neg\ e)) = 1 - (Pr(\neg\ b) + Pr(\neg\ c) - Pr(\neg\ b)Pr(\neg\ c))\ Pr(\neg\ d)\ Pr(\neg\ e)$

Risk Analysis – System Theoretic Process Analysis (STPA)



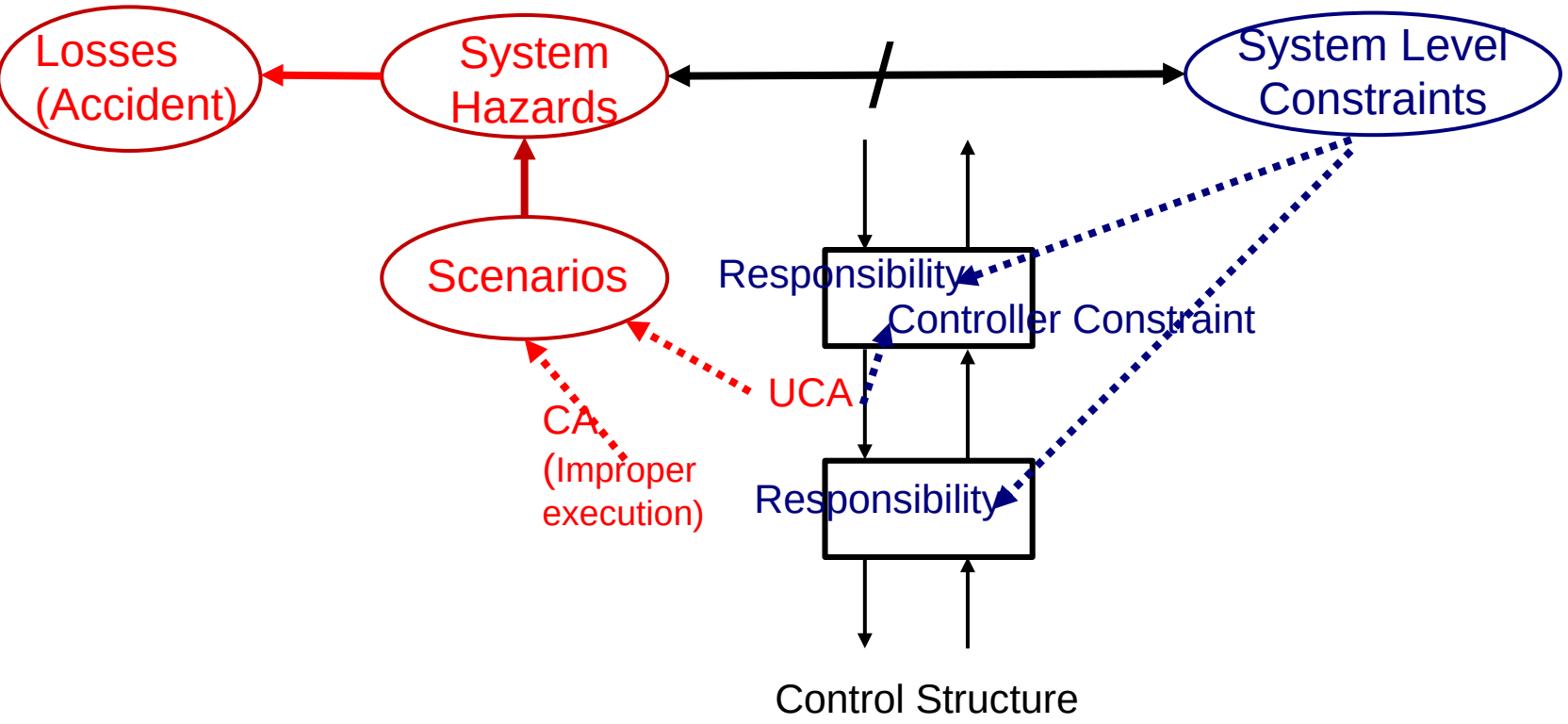
STPA (N. Leveson et al. MIT) is a hazard analysis methodology focusing on unsafe interaction between components in a hierarchical control stricture

- Step 1: what kinds of *losses* will the analysis aim to prevent?
- Step2: build a system model capturing functional relationships and Interactions as a set of feedback control loops.
- Step3: analyse *control actions* in the control structure to examine how they may lead to the losses defined in the Step 1. Unsafe control actions are used to create functional requirements and constraints for the system.
- Step4: identify the reasons why unsafe control actions might occur In the system by creating explicative scenarios.

Risk Analysis – STPA: Some Conclusions

STPA is the systematization of a risk analysis methodology. It reflects practice in the domain of aeronautics and gets acceptance in automotive industry

- The analysis is manual and hard to be formalized as it is – it is applicable to systems with a limited number of possible losses (accidents)
- It distinguishes between different types of faults (provided, not provided, too early or too late, stopped too soon or applied too long) but the component behavior is implicit.



- The methodology distinguishes between GOOD (Constraints, Responsibilities) and BAD (Hazards, Losses);
- Responsibilities are invariants (monitors) of the behavior of the components;
- Projecting System Level Constraints into responsibilities (local constraints) is not trivial : The conjunctions of the Responsibilities should imply the System Level Constraints.

Risk Analysis – Fault Propagation and Transformation Calculus (FPTC)

- FTPC (Wallace et al. University of York, 2005) goes one step further than STPA, in an attempt to analyze the causality relation between types faults in hierarchical data-flow networks.

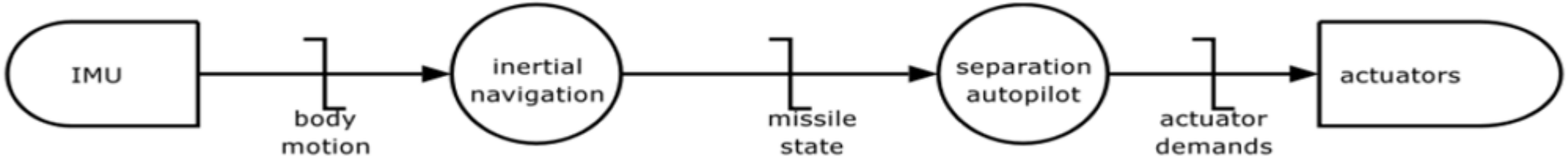
- Components and connectors are specified by an I/O relation → between fault values where
  - * : denotes the absence of fault;
  - f : is a variable denoting any type of fault .

- Components are specified by sets of rules. They can:
  - be source of faults : * → f (produces a fault f)
  - be sink of faults: f → * (tolerates or corrects a fault f)
  - propagate a fault f : f → f
  - transform a fault f1 into fault f2: f1 → f2

- A fault ontology defines types of faults for the variable f (this gives a lot of freedom):
  - Value faults: cause the components to respond at the correct time interval but with wrong values (Detectably wrong, Undetectably wrong)
  - Timing faults: cause components respond with correct value but outside the time interval (Early or Late)
  - Service provision faults: component fails to produce an appropriate output (Omission) or produces an inappropriate output (Commission)

Risk Analysis – FPTC (Fault Propagation and Transformation Calculus)

The system comprises four software components, connected using three instances of a signalling communication protocol. This protocol uses a destructive (non-blocking) write, and a destructive (blocking) read.
(From Paige et al. “Automated Safety Analysis for Domain-Specific Languages,” 2008)



Component	Behaviour
Inertial Navigation	$v \longrightarrow v$
Separation Autopilot	$* \longrightarrow stale\_value$ $* \longrightarrow detectable\_value$ $v \longrightarrow v$
Signal Comms Protocol	$early \longrightarrow *$ $omission \longrightarrow late$ $commission \longrightarrow value$ $v \longrightarrow v$

Table 1. Behavioural properties of components.

Risk Analysis – A Synthesis

Behavioral analysis of the out-of-nominal behavior (AltaRica. AADL, Statistical Model-Checking at Verimag)

- Relies on component-based representation of the system with probabilistic hazard events and modes;
- Algorithmic analysis techniques can be applied to estimate probabilities of loses.

FPTC:

- Analysis based on the systems hierarchical control data-flow structure;
- Components have known I/O behavior with respect to predefined fault types;
- The analysis can be automated and has been implemented in tools.

STPA:

- Reasoning on a hierarchical control structure independent from the actual implementation and distinguishing between the system and its environment;
- Local reasoning at each component level and synthesis of the analysis via scenarios – Nonetheless components have no explicitly specified behavior.

FTA, FMECA (Failure Mode, Effects & Criticality Analysis), ETA (Event Tree Analysis): global reasoning at system level in terms of failure modes and their effects

- determine AND/OR combinations of causality relations;
- components have no behavior – they can be OK or KO with some probability.

OVERVIEW

- System Design and Dependability
- Design for Dependability
  - Risk Analysis
  - Risk Mitigation
  - Risk Evaluation
- Discussion

Risk Mitigation – Fault Detection, Isolation, Recovery (FDIR)

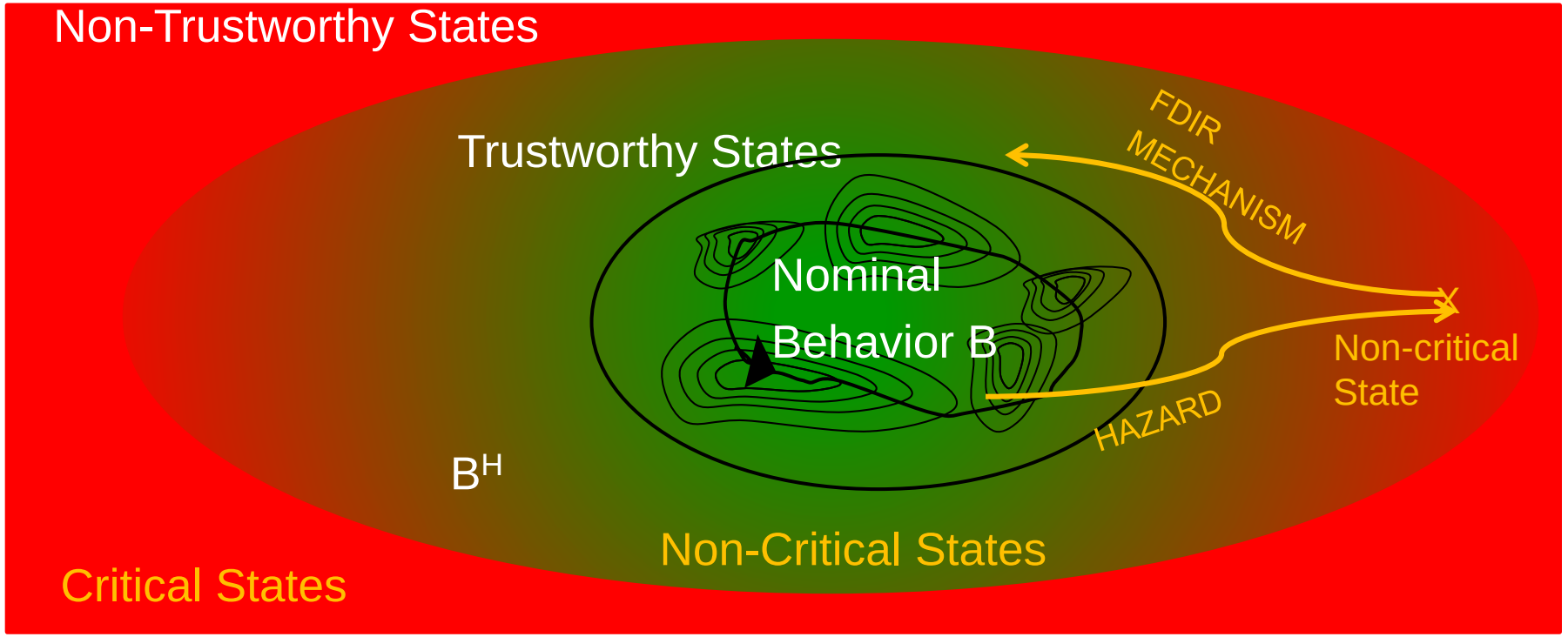
Fault detection: achieved
- on line without disturbing service delivery e.g. using monitors, redundancy, error detecting codes;
- off line by applying testing and diagnostics techniques.

Fault isolation: various techniques such as
- In SWE use of partitioned architectures such that the memory and processing time of a partition is not affected by another faulty partition;
- In Control Theory “model-based techniques”.

Fault recovery:
- using fault-tolerance mechanisms that mask the detected fault e.g. TMR;
- online using roll-back techniques to a trustworthy saved state that existed prior to the occurrence of the fault (caused by SW error);
- online using roll-forward techniques to a to a trustworthy new state (caused by SW error);
- online through reconfiguration that preserves some minimal service leaving out the faulty components ;
- offline with system re-initialization after checking that the causes of the detected fault are not present anymore.

Risk Mitigation – Principle

Mitigation :
1) develop a model B^H relating nominal behavior B to a set of hazards H (from Risk Analysis);
2) design and implement failure detection isolation and recovery mechanisms.

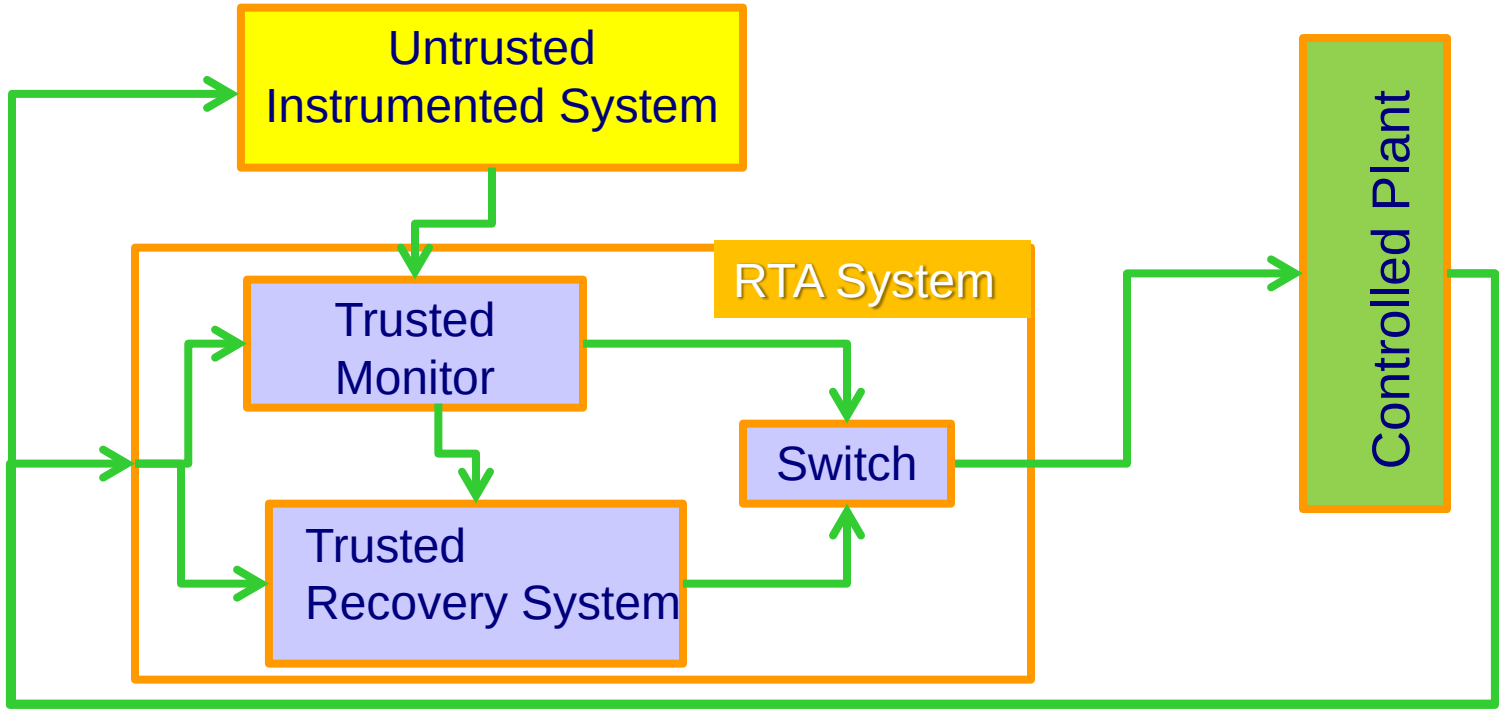


- Worst-case analysis and static resource reservation is defeated by the overwhelming complexity of modern systems and may lead to over-provisioned systems
- Idea: move to Run-time Assurance Techniques

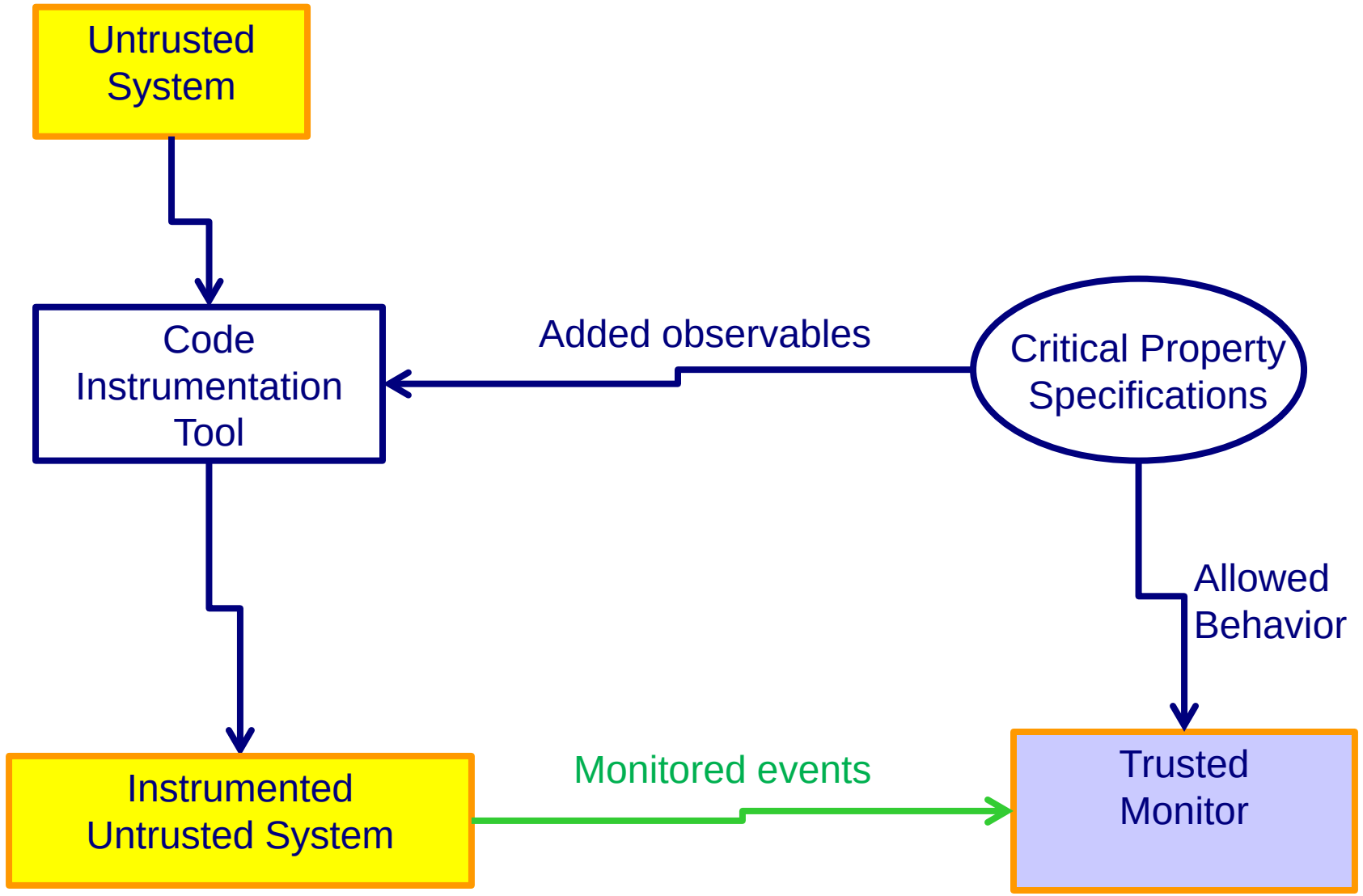
Risk Mitigation – Run-time Assurance

- Replace redundancy techniques and detailed risk analysis by Runt-time Assurance:

- Run in parallel the Untrusted System and a Run-time Assurance System consisting of a Trusted Monitor and a Trusted Recovery System.
  - The Trusted Monitor detects discrepancies from the nominal behavior;
  - The Trusted Recovery System can cope with hazards detected by the Trusted Monitor;
  - The Switch provides the output of the Untrusted System as long as no hazard is detected.



Risk Mitigation – Run-time Assurance: Trusted Monitor Generation



OVERVIEW

- System Design and Dependability
- Design for Dependability
  - Risk Analysis
  - Risk Mitigation
  - Risk Evaluation
- Discussion

Risk Evaluation – Reliability, Availability etc.

Reliability:

- Reliability is the probability R(t) that a system fails at time t
- For fixed failure rate λ we have $R(t)= e^{-\lambda t}$ and probability of failure $F(t)=1- e^{-\lambda t}$

Statistical Analysis: For given failure rate λ (failures per unit e.g. time, km) the confidence C to the system after n units of failure-free operation is $C=1-(1- \lambda)^{n}$

Therefore, the number of units required to operate without failure to achieve confidence C:

$$n = \ln (1-C)/\ln(1-\lambda)$$

For instance, for 1-λ=99.9999989% with a C=95% confidence we need n = 275 million failure-free units.

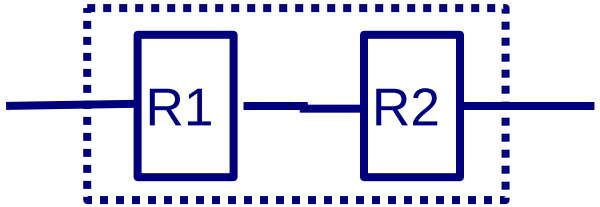
Availability = Uptime /Uptime+Downtime

- MTBF (Mean Time Between Failure) = 1/λ
- MTTR (Mean Time To Repair) = Σ λi ti / Σλi where
  λi is the failure rate for the ith failure mode
  ti is the time to repair the system after the ith failure has occurred

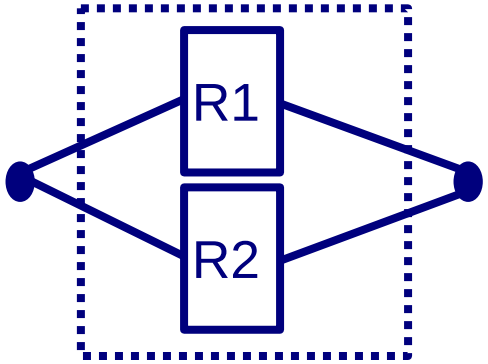
❑ We can also define MTBM (Mean Time Between Maintenance), MMT (Mean Maintenance Time), MDT (Mean Downtime) and of course maintainability, serviceability, manageability etc.

Risk Evaluation – Estimating System Reliability

1. Composition of a functional model obtained as the serial and parallel composition of components

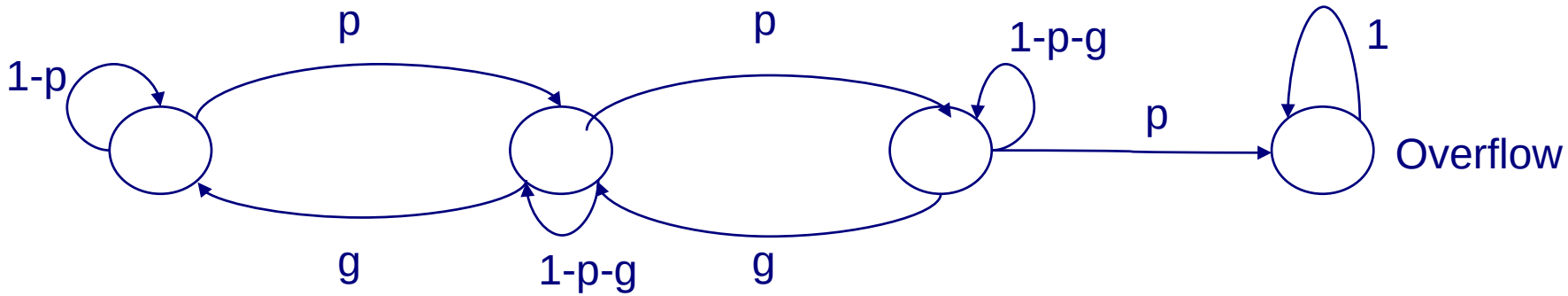


R= R1R2



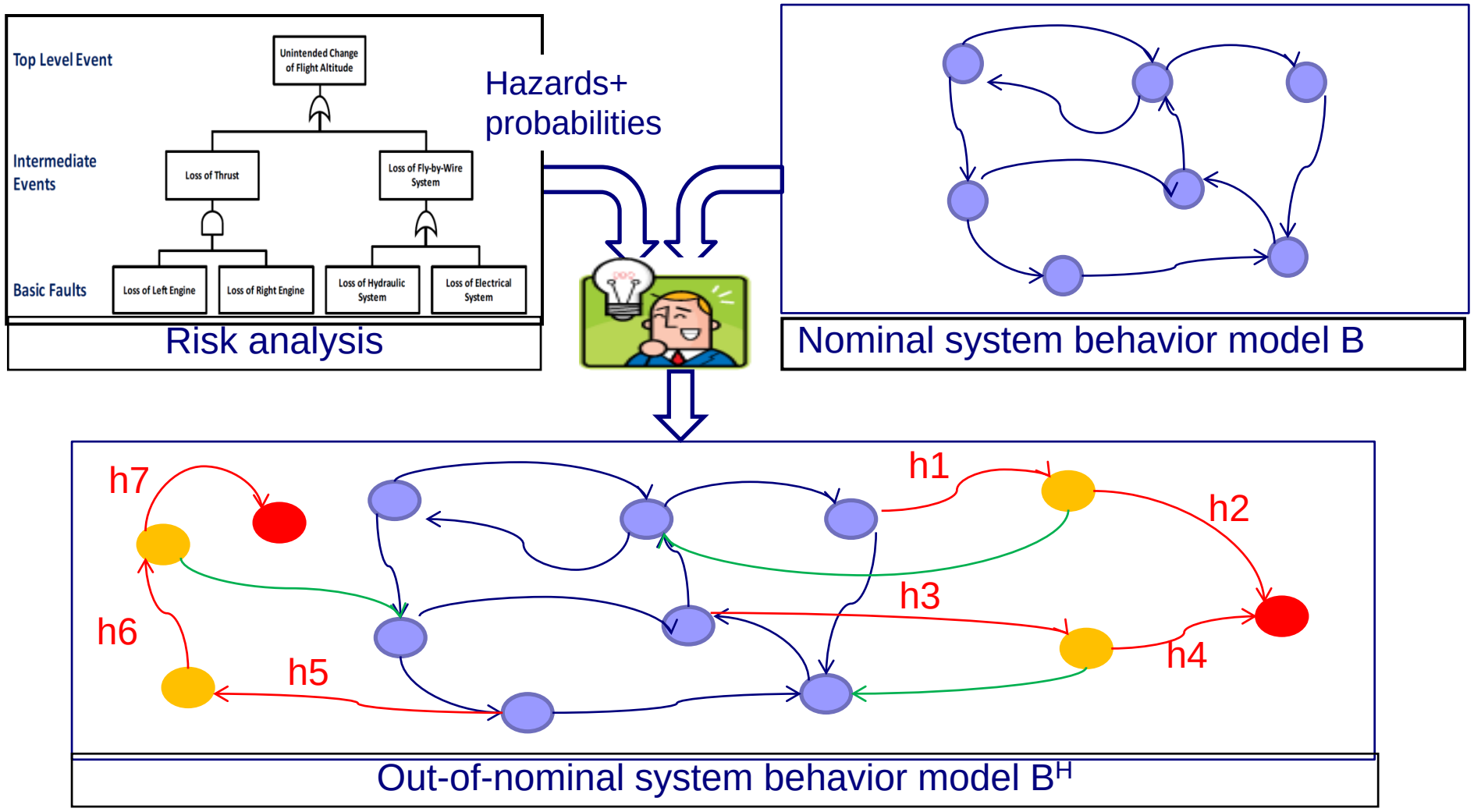
R= 1-(1-R1)(1-R2)

2. Analysis of behavioral stochastic models e.g. Markov Chains suffer from well-known limitations e.g. state explosion, lack of theory for model composition



3. General model-based analysis techniques on models representing the faulty system behavior.

Risk Evaluation – Behavioral Aproach

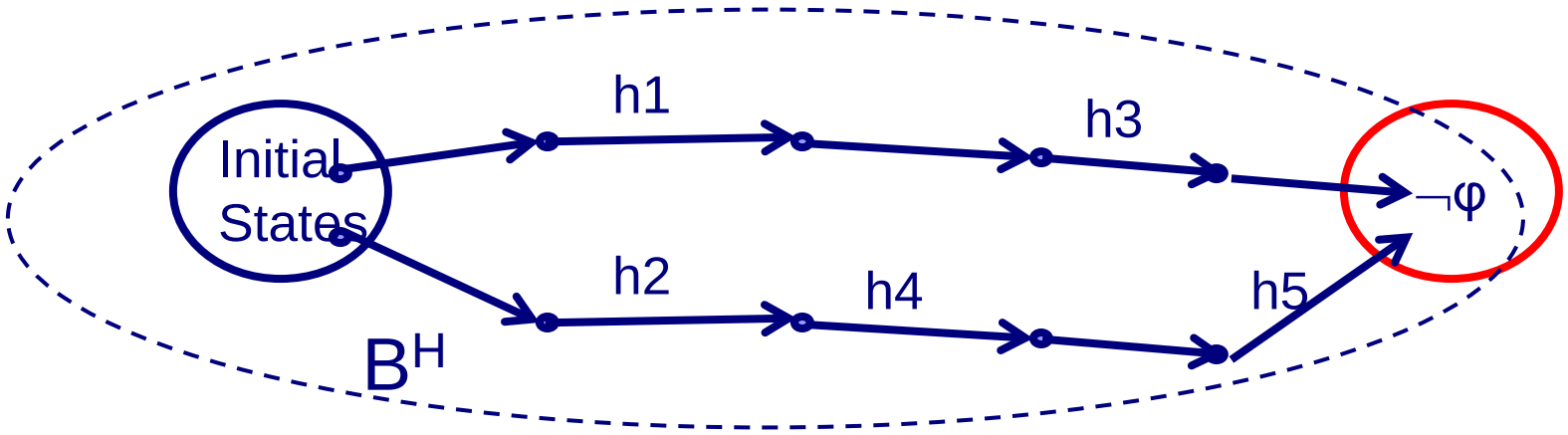


- Estimating “likelihood” to reach bad states (in red)
- Strategies for avoiding bad states
- Design space exploration of parametric models for risk minimization

Evaluation – Behavioral Aproach

Given a system with nominal behavior B=(Q, →) where Q is its set of states and → is the transition relation find the system B^H modeling the out-of-nominal behavior B^H =(Q^H, $→^H$) for a set of hazards H such that

- $Q^H = Q \cup C$ where C is a new set of critical states $Q \cap C = \varnothing$
- B^H contains the behavior of B ($→ \subseteq →^H$)
- B^H has transitions of the form:
  q –h→ c (hazard h occurs), c1—h→c2(transition between critical states)



φ is a safety constraint - so ¬φ characterizes critical states

- Given B^H and the probabilities that the hazards happen we can estimate the probability to violate a constraint φ by application of algorithmic analysis techniques e.g., statistical model-checking;
- Note that the approach can be used for Risk Analysis – for instance to computing cutsets as the sets of hazards of a path leading from initial states to violation of φ, e.g. h1 h3 ∨h2h4h5.

OVERVIEW

- System Design and Dependability
- Design for Dependability
  - Risk Analysis
  - Risk Mitigation
  - Risk Evaluation
- Discussion

Discussion – The Big Picture

MATHEMATICAL MODELS

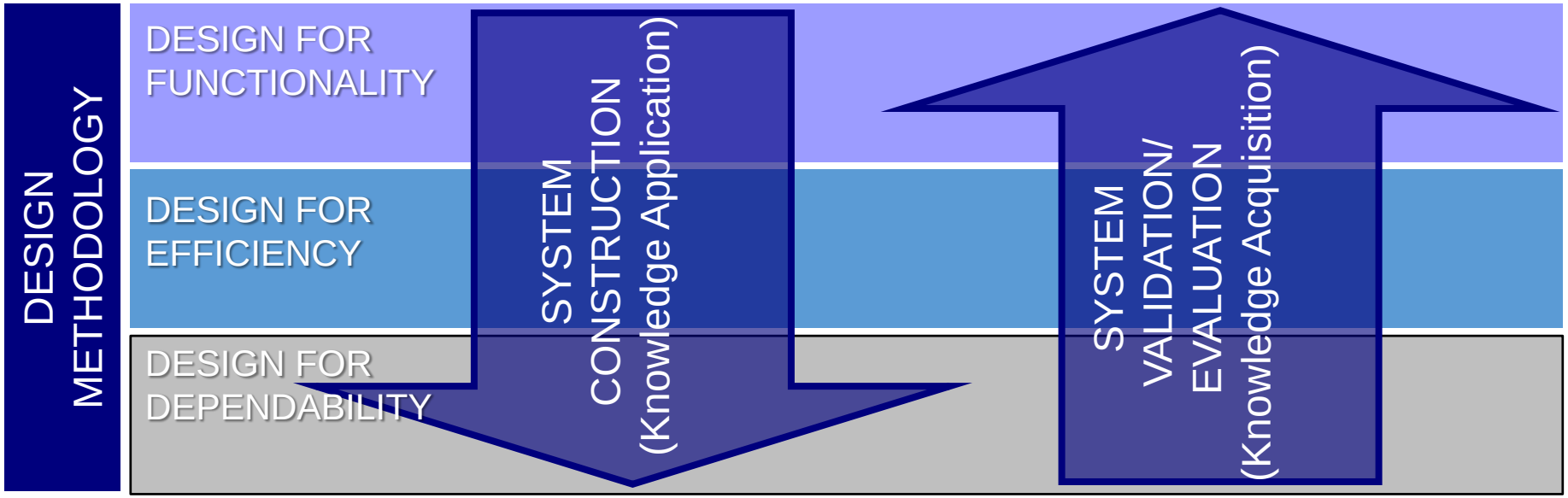
- Dynamic systems
- Dynamic model execution
- Logic-based property description

METHODS & TOOLS

- Transformation methods
- Synthesis methods
- Experimentation/validation/Evaluation methods

LANGUAGES

- General purpose programming languages e.g. imperative, functional, logical ….
- Domain Specific: Synchronous; Modeling languages e.g. UML. Query languages …..
- Non-executable languages: Natural Languages, Logic-based languages e.g. Temporal logics,.



SYSTEMS

- Application Domains: Industry e.g. data, production; Services e.g. communications, health,….
- Computing Infrastructure: Execution platform e.g. physical vs virtual; architecture e.g. centralized vs..
- Types of systems: Neural nets, HW, HW/SW, OS, Middleware, Data Base, Service Systems, ….

Discussion

- Dependability terminology
  - Terminology is useless if it is not connected to methodologies describing how methods, tools and competences can be combined to produce systems - Reliability engineering is often presented as something apart from the other system design activities.
  - What matters is not so much the choice of terms themselves but the coverage of all the useful concepts
  - We need a dependability ontology integrated in a general system design ontology and relying on a minimal and complete set of core concepts:- current terminologies are vague, with overlapping concepts.

- We should integrate dependability techniques into systems design flows
  - Design for dependability should be a concern from the beginning and should take into account all the aspect of system development and validation
  - For risk analysis we should move from Boolean FTA to techniques that take into account system architecture e.g. STPA and behavior e.g. FPTC
  - Traditional FDIR techniques require a detailed static analysis often defeated by the overwhelming system environment complexity and lead to overprovisioned solutions – we should move to Runtime Assurance Techniques.
  - Traditional reliability evaluation is limited to data-flow functional systems – we should explore component-based techniques modeling the effect of failures on nominal behavior and amenable to statistical analysis.

THANK YOU