Design Principles for Reducing the Complexity of Safety-Critical Embedded Systems

H.Kopetz

July 2023

What are the Trends that shape our Field?

- Hardware Performance is still increasing (3.5 nm Technology).
- Automation of Safety Critical Processes is getting widespread (e.g. Autonomous Vehicles, Robotics, Energy Distribution, etc.).
- Wide use of AI for *object perception* and *categorization* leads to large control systems with millions of lines of code.
- Mind-boggling Complexity prevails and hinders human understanding and explainability—*can we trust the machine*?
- An Intrusion into a safety-critical system is an issue.

It is the objective of this talk to elaborate on the principles that help in the design of these large autonomous control systems.

Outline

- Introduction -- Some Terminology
- The *Challenge* of Design
- Autonomous Driving (AD)
- Safety Assurance (SA) Subsystem
- Byzantine Faults
- A Solution to the *Challenge*
- Conclusion

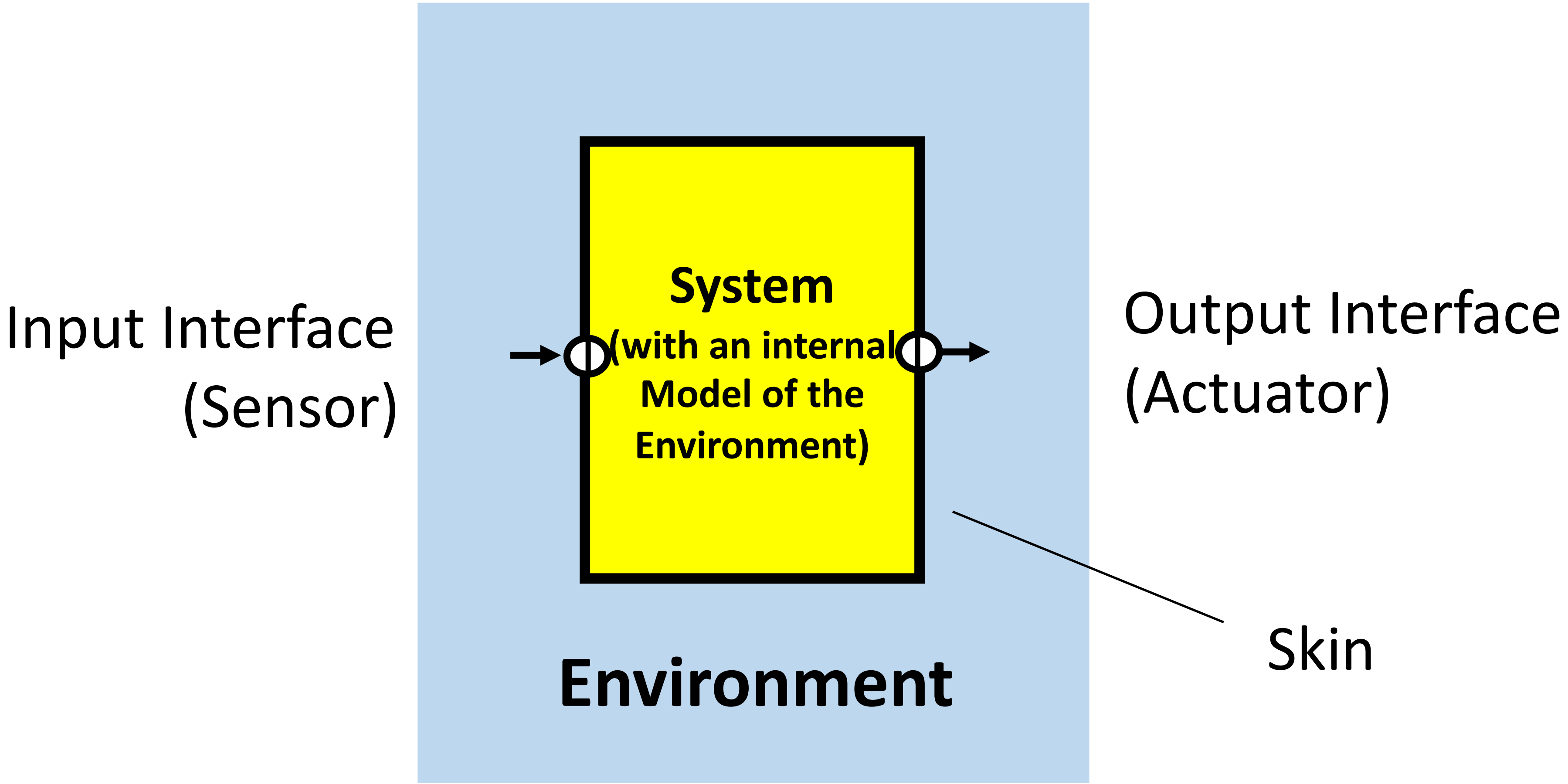
An *Embedded* (Sub) *System*

is *a human-made artefact* that

- has a ***purpose*** and is a ***whole*** that is encapsulated by a physical or virtual ***skin*** that separates the system from its ***environment***.
- The **purpose** of the system is achieved by the **service** (*intended* ***behavior***) of the system to its environment, based on the results of an ***internal model*** of the environment.
- A system has ***interfaces*** in the skin that observe the environment (***sensors*** that provide the input data for the internal model) and act on the environment (***actuators***).
- From the system point of view, the ***perceived environment of a system*** consists of those entities that are observable or can be controlled via the interfaces of the system.

Software *per se* does not qualify as a *system*—it has no temporal properties.

Model of a System



System boundaries imply responsibilities

Decomposable versus *Monolithic* System

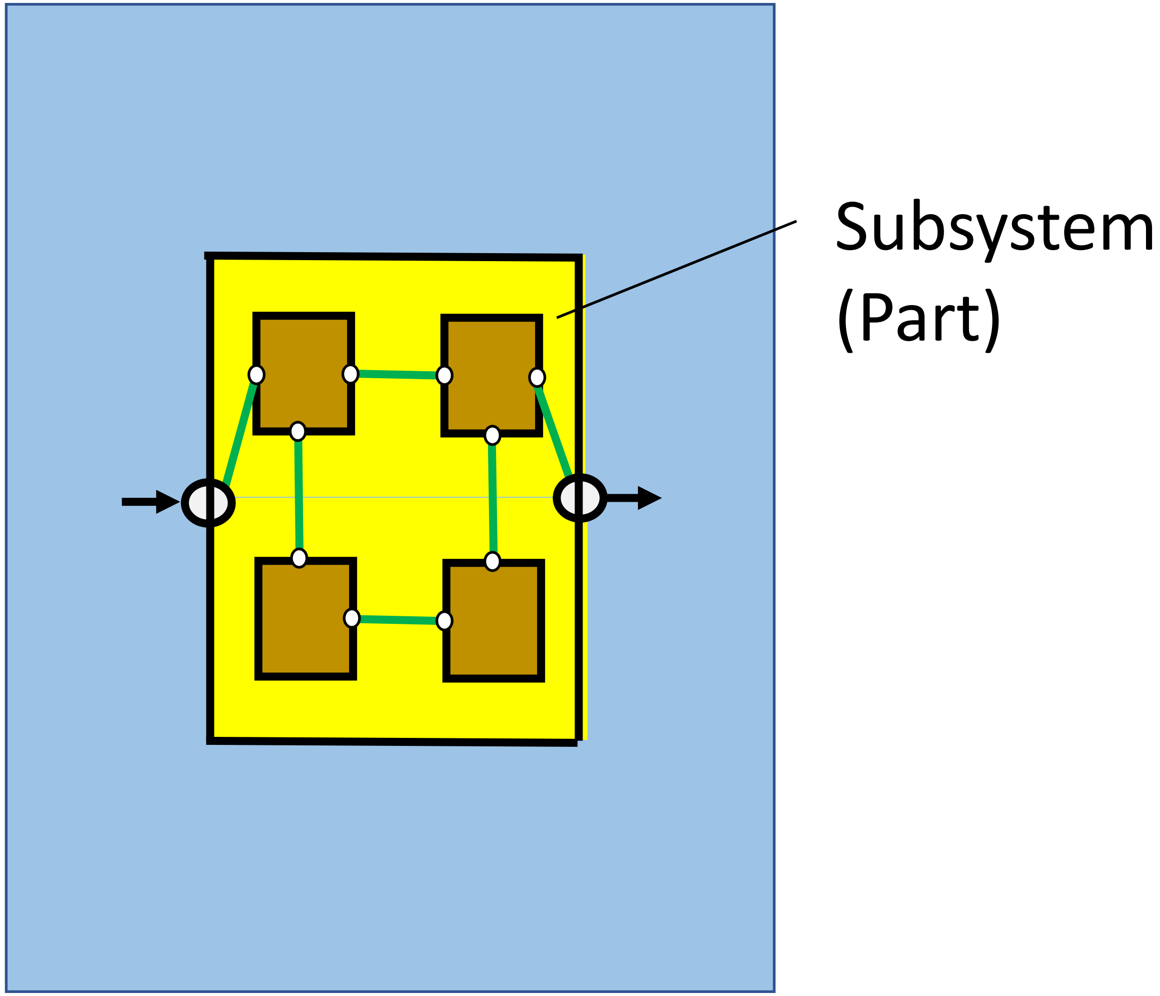
In the *embedded domain*, a large system is called ***decomposable*** if it can be partitioned into a small number of identifiable ***self-contained subsystems, the parts*** (*composed of software **and** hardware),* that interact solely via ***simple message-based interfaces****.*

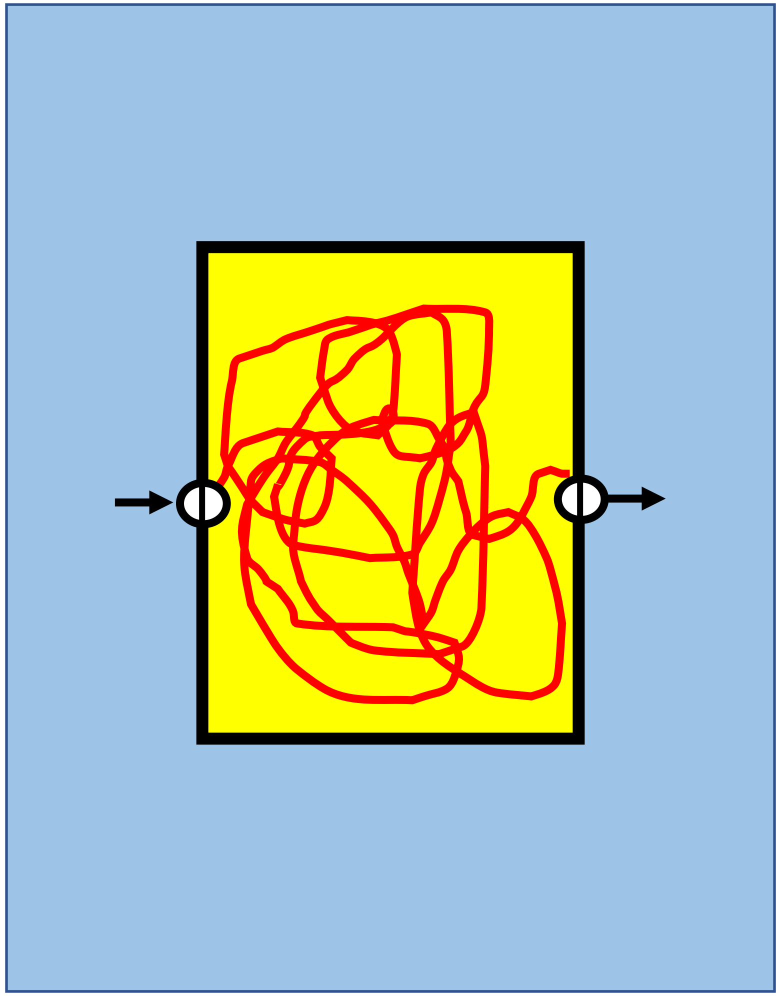
A message-based interface is *simple* if

- the *information items* conveyed by the messages are well defined **in the domains of *data*, *context* and *time***—*time-triggered* messages help!
- the messages can be observed by an independent monitor.
- there are no ***unintended emergent effects*** caused by the message interactions.

If a system is not *decomposable*, it is called a *monolithic* system.

Decomposable versus Monolithic System





flow of simple messages

control flow

Decomposition improves the *Goal Clarity*

- *Goal Clarity* refers to the "*the extent to which the outcome goals and objectives of a job are clearly stated and well defined*" [1]) **.**
- *Goal Clarity* improves the motivation and productivity of a project team and the **quality of the product**.
- A decomposition of a system into nearly independent subsystems— the result of a proper *architectural design*— establishes *goal clarity for every subsystem,* since every subsystem has
  - a well-defined purpose
  - precisely specified interfaces

[1]) Sawyer, J.E. Goal and Process Clarity: Specification of Multiple Constructs of Role Ambiguity and a Structural Equation Model of Their Antecedents and Consequences. Journal of Applied Psychology 1992, Vol.77. No. 2, p. 134.

A Subsystem (Part) is a *Fault-Containment Unit (FCU)*

A hardware-software subsystem is a *fault-containment unit (FCU),* if it has a clear purpose, is self-contained (*hardware plus software*) within its skin, interacts with its environment exclusively by simple messages, and if the direct impact of any fault effects the operation of this subsystem only.

The service of an FCU can be impacted by the following faults:

- permanent hardware fault (e.g. failing transistor, design, etc.)
- transient hardware fault (SEU-single event upset, power outage, etc.)
- specification fault (e.g., incomplete specification of edge cases)
- programming fault
- input fault
- an intrusion.

FCUs must fail independently.

Semi-autonomous vs *Fully-autonomous* Embedded Systems

Semi-autonomous System

- Provides the **specified service** by a ***primary control system*** under *nominal conditions.*
- *The nominal conditions* are part of the specification.
- Requires human intervention to detect and mitigate *off-nominal* conditions.

Fully-autonomous System

- Provides the **specified service** *under nominal conditions* and a **safe exit** under *off-nominal conditions.*
- Requires an ***independent Safety Assurance (SA)*** Subsystem to handle the behavior under *off-nominal conditions.*

Requirement: The safety (probability of a catastrophic event during the lifetime of the system) of a fully autonomous embedded system should be ***better than the safety of a semi-autonomous system***

Functions of the *Safety Assurance Subsystem* (SA)

The *safety assurance (SA) subsystem* must bring the controlled object to a safe state in case a **critical event** has occurred that caused an off-nominal condition.

The SA mitigates the effects of a failure!

Functions of the SA (realized by the *human driver* at level 2):

- **Detection Function**: Detect an off-nominal condition.
- **Decision Function**: Decide to deactivate the faulty subsystem and activate a *fallback subsystem*.
- **Fallback Function**: Bring the *controlled object* to a safe state.

What are *Off-Nominal* Conditions?

Nominal Condition

The *specified design assumptions*
—The Specifications—
about the system in its operational environment **hold.**
(ODD—operational design domain)

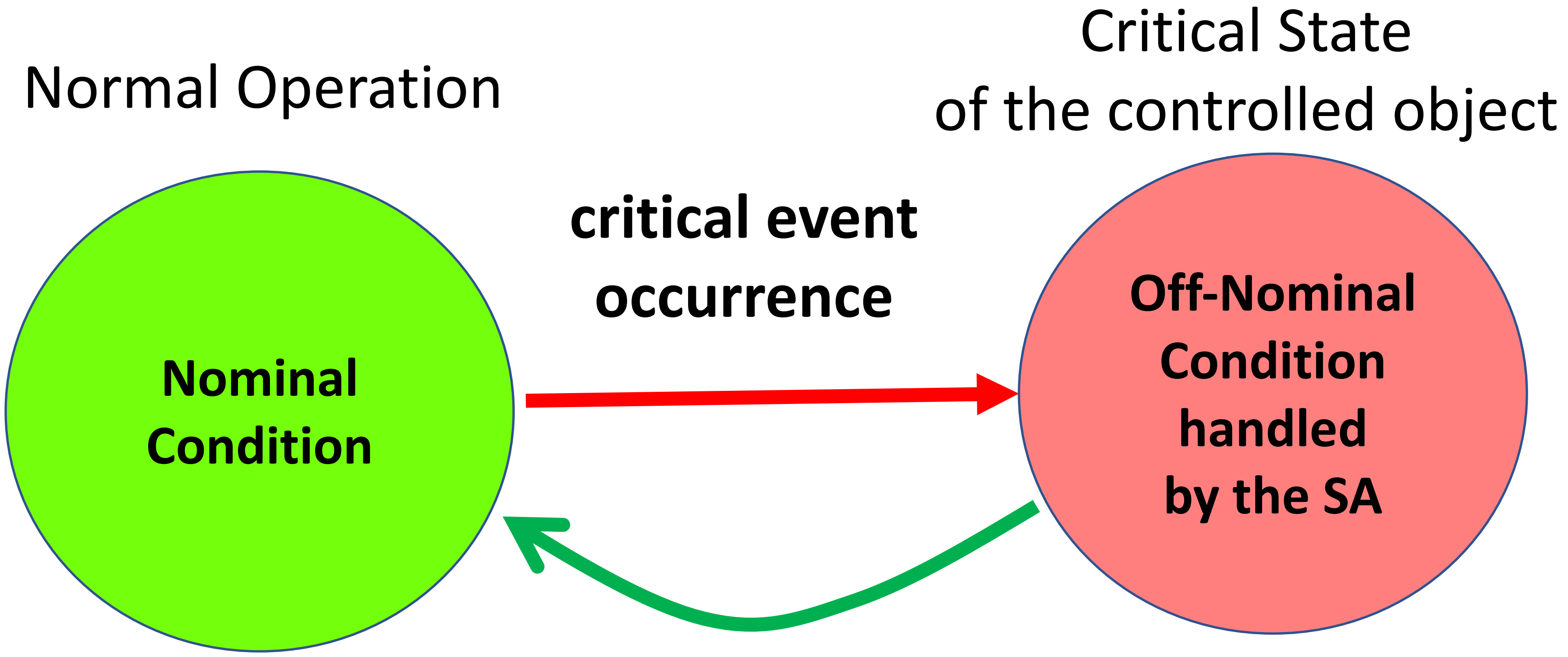
Off-Nominal Condition

Some of the specified design assumptions concerning the nominal conditions about the system or its environment are violated as the consequence of a **critical event.**

The dependability of a *perfect system* is limited by the **assumption coverage.**

Assumption coverage: ***Probability*** **that the assumptions that are made in the *design* and about the *operation* of a system hold during the expected lifetime of the system.**

Nominal condition versus *Off-Nominal* condition



Critical Event Handling by a *Safety Assurance (SA)* subsystem brings the controlled object to a safe state.

At the Start . . .

At the start of the design of a new safety-critical technical system, the following questions should be answered:

- What is the ***purpose*** of the envisioned system?
- What is the ***demanded dependability*** of a safety-critical embedded system?
- What are the relevant ***economic constraints***?
- What are the relevant ***technical constraints***?

The *Challenge in the Design* of a *Large Safety-Critical System*

Find an understandable decomposition of a large safety-critical fully autonomous ultra-dependable embedded system

—e.g. a system for *Autonomous Driving (AD)*—

into independent *Fault-Containment Units (FCU) that* interact by *simple messages only* and where a single failure in anyone of its complex FCUs does not cause an accident.

Example: Autonomous Driving (AD)

Purpose: A car with a fully-autonomous driving (AD) system must transport its passengers safely from a defined start to the selected destination.

AD systems have the following characteristics:

- The safety of an AD system must **be significantly better** than that achieved by a human driver—s**afety is a *tail event*.**
- An AD system is ***complex***: more than ten million lines of code.
- An AD system must handle ***nominal*** and ***off-nominal*** conditions and must mitigate its own faults.

Up to now, more than 100 billion dollars have been spent on AD.

Required Safety of an Autonomous Driving (AD) System:

In Austria there are about 5 Million vehicles on the road. If we assume that every vehicle travels for 200 hours/year with a speed of 60 km/hour, then every vehicle travels 12 000 km/year

In Austria, every year there are about

- 35000 reported traffic accidents, i.e. one accident/150 vehicles or 1,8 Mio km
- 500 traffic fatalities, i.e. one fatality/10 000 vehicles **(Relation 1:70)**

If we mandate that an autonomous car should be *many times* better than a human driver, then an autonomous car must not be involved in

- **a traffic accident for 1 000 000 hours or 60 000 000 km driven**
- **a traffic fatality for 10 000 000 hours or 600 000 000 km driven**.

This brings us into the domain of *ultra-dependable systems*.

Ultra-dependable systems are *different* . . .

Most dependability engineers that are working on the design and validation of ***complex ultra-high dependable embedded systems*** would agree that there is strong ***experimental evidence*** that it is ***impossible*** to overcome the constraints that are summed up in the following *four* ***impossibility results***:

(i) It is **impossible** to find all design faults in a large monolithic hardware/ software system that contains millions of lines of software code.

(ii) It is **impossible** to avoid a single event upset (SEU) in non-redundant hardware during on interval of 1 000 000 hours (i.e. 100 years).

(iii) It is **impossible** to establish the ultra-high dependability of a large monolithic system by testing and simulation.

(iv) It is **impossible** to precisely specify all edge cases that can be encountered during 1 000 000 hours of operation of a fielded large ultra-dependable system.

AD Systems are *complex*: the two Facets of Complexity

Complexity is a property of a scenario that is primarily used to denote the mental difficulty of understanding a scenario by a human — complexity generally increases with *perceived size*.

(i) **Object Complexity**: Complexity as a *Property of a Scenario*. A scenario consisting of *many different parts with many peculiar uncontrolled interactions* is considered *complex*.

(ii) **Cognitive Complexity:** Complexity as *a Relation between a Scenario and an Observer*. An expert that has a highly developed conceptual landscape of a domain can consider a scenario as *simple* that is *complex* to a novice. (e.g. *Elo rating* of chess).

In general, a high ob*ject complexity* leads to a high *cognitive complexity*.

Three *Design Principles* for *Complexity Reduction*

The following three design principles, that must be applied *iteratively*, help to reduce the complexity of a system and lead to a multilevel hierarchy:

Top down

(i) Partitioning (Divide and Conquer): Decompose the system into self-contained subsystems with **well-defined interfaces** among the subsystems such that each subsystem can be developed independently.

(ii) Abstraction: Find a *high level conceptualization (a reduced representation)* that supports the ***achievement of the purpose***. In case of a safety-critical system the first-level subsystems should be few ***Fault-Containment Units*** that interact by the exchange of **simple messages**.

Bottom up

(iii) Segmentation: Segment the behavior in the temporal domain between communication and processing. Interactions among subsystems are only allowed to occur at the beginning and the end of a frame..

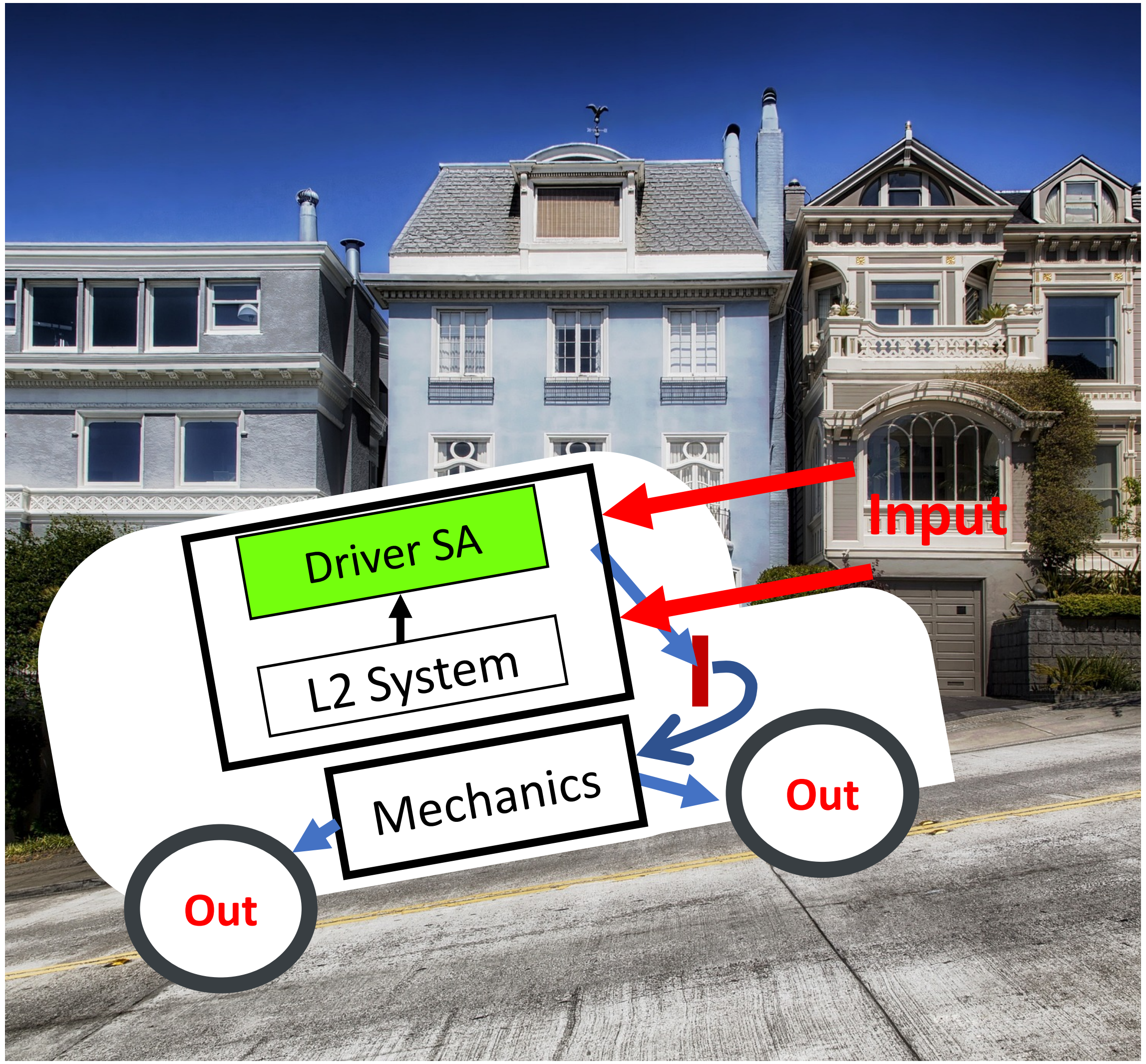
SAE Driving Automation (DA) Levels according to *J3016*

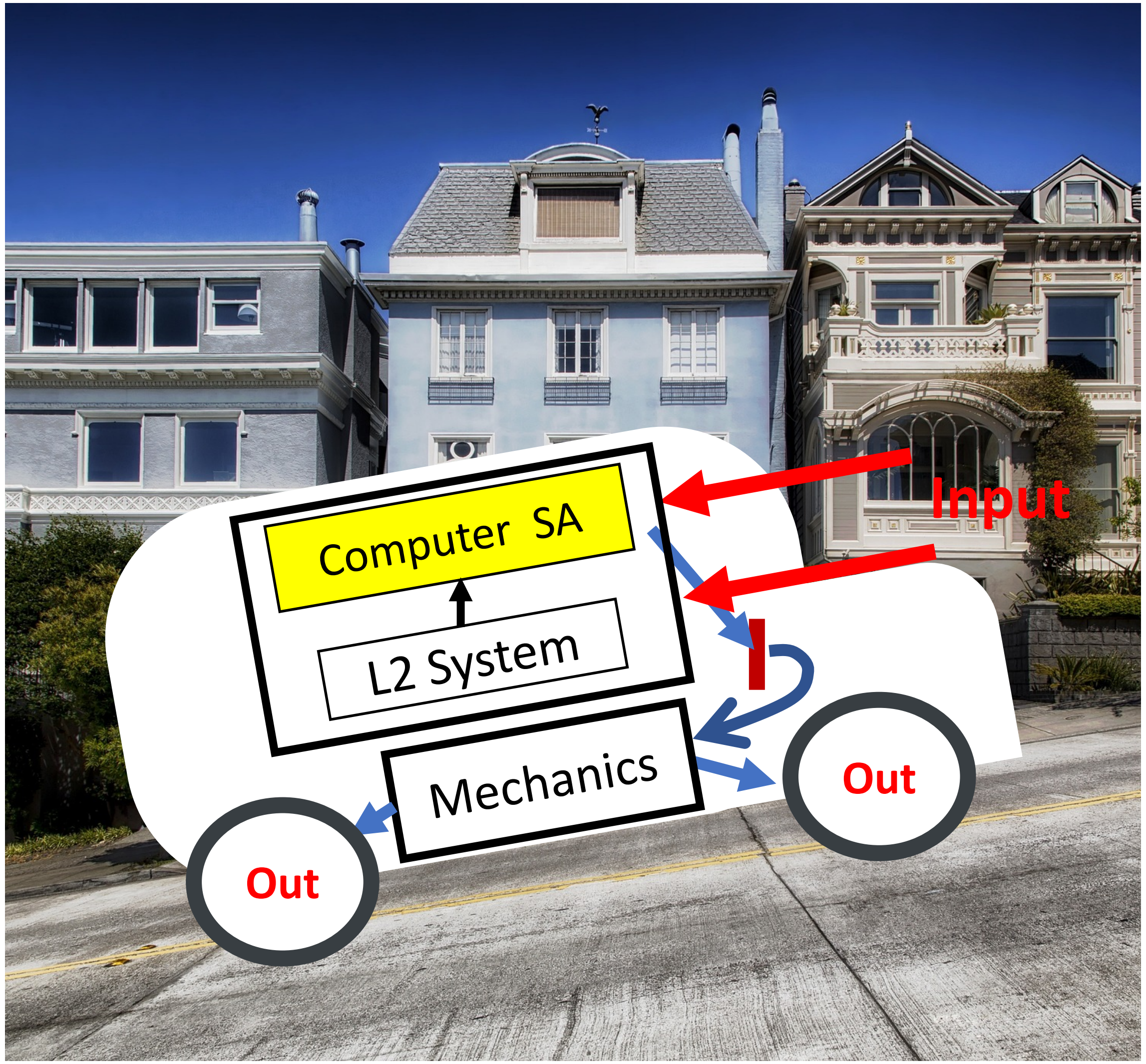
	Role of the Driver	Role of the Computer
Level zero: No (DA)	Performs all Driving Tasks	None
Level one: Driver Assistance	Supervises the DA system and intervenes if necessary	Longitudinal or Lateral Vehicle Control within the ODD
Level two: Partial DA	Supervises the DA system and intervenes if necessary	Longitudinal and Lateral Vehicle Control within the ODD
Level three: Conditional DA	Fallback ready driver is always ready to take over on request from the DA	Longitudinal and Lateral Vehicle Control within the ODD issue of takeover request
Level four: High DA	No Fallback ready driver required	Sustained Vehicle Control in a limited ODD
Level five: Full DA	No Fallback ready driver required	Sustained Vehicle Control in an unlimited ODD

semi-autonomous (Levels zero–two)

fully-autonomous, requires an SA (Levels three–five)

Safety Assurance Subsystem: L2 versus L4





Critical Events in AD **(Leads to an *Disengagement* of an L2 System)**

Internal Events—failure events within the vehicle.

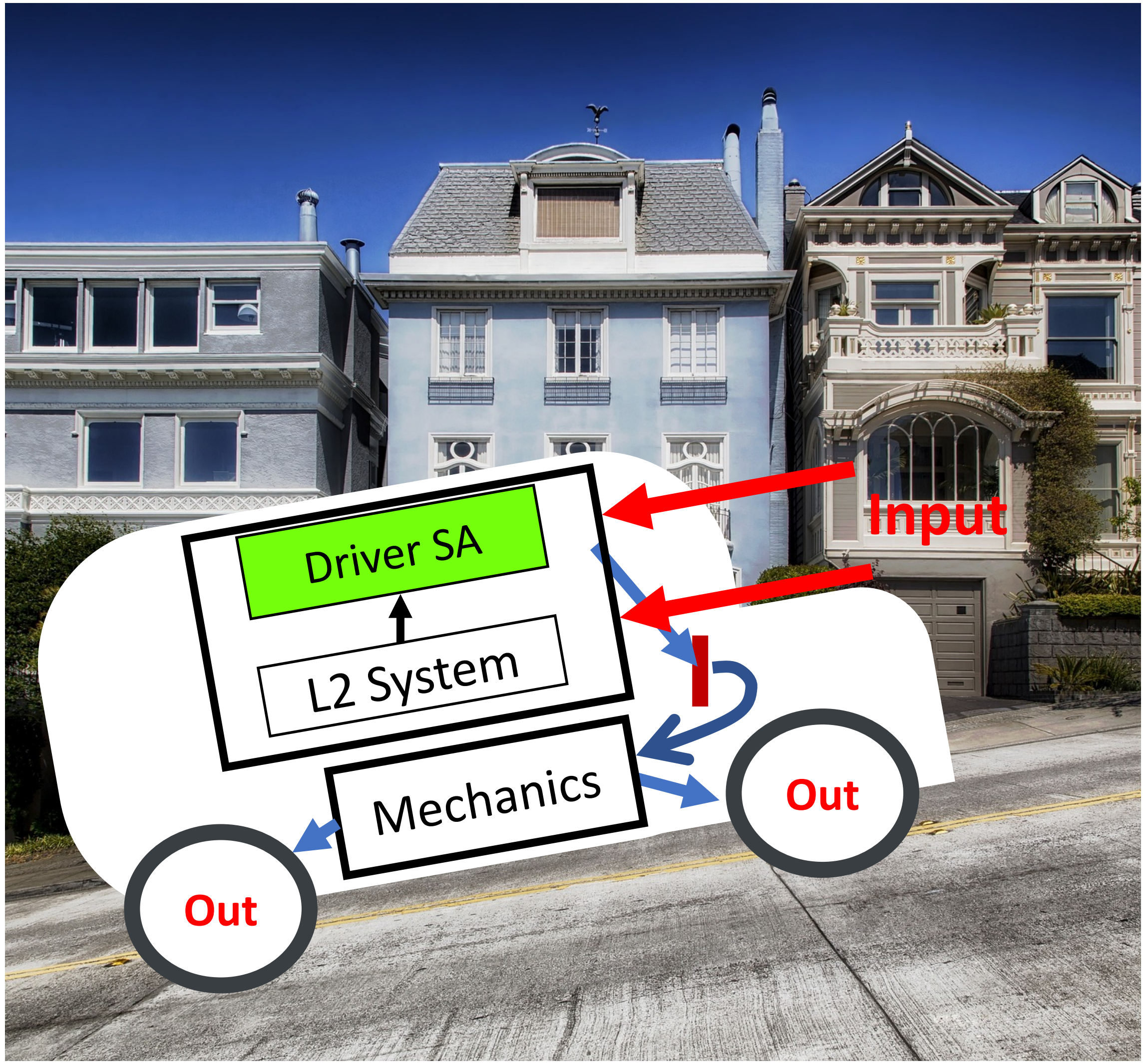
- Software specification error (*wrong* nominal conditions).
- Programming error (e.g. *Heisenbug* → bit flip).
- Not specified computer hardware failure (e.g. SEU → bit flip).
- Not specified failure of a mechanical part of the car.

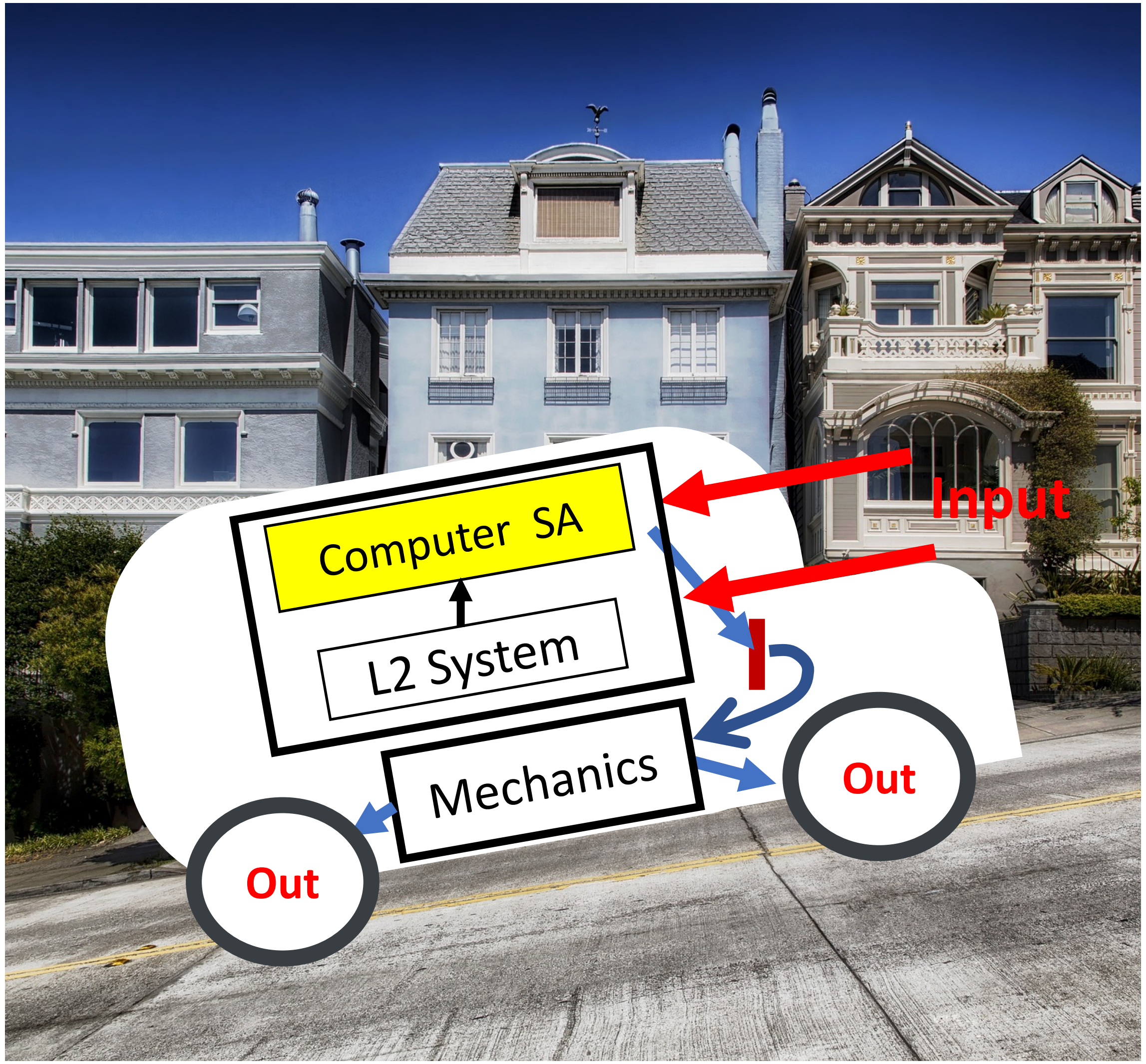
External Events—unspecified events outside the vehicle.

- Not recognized ODD exit   (e.g., snowfall, ice, road condition)
- Traffic participants behave outside the specification (e.g. children)
- Intrusion
- and many more (e.g. suicide driver).

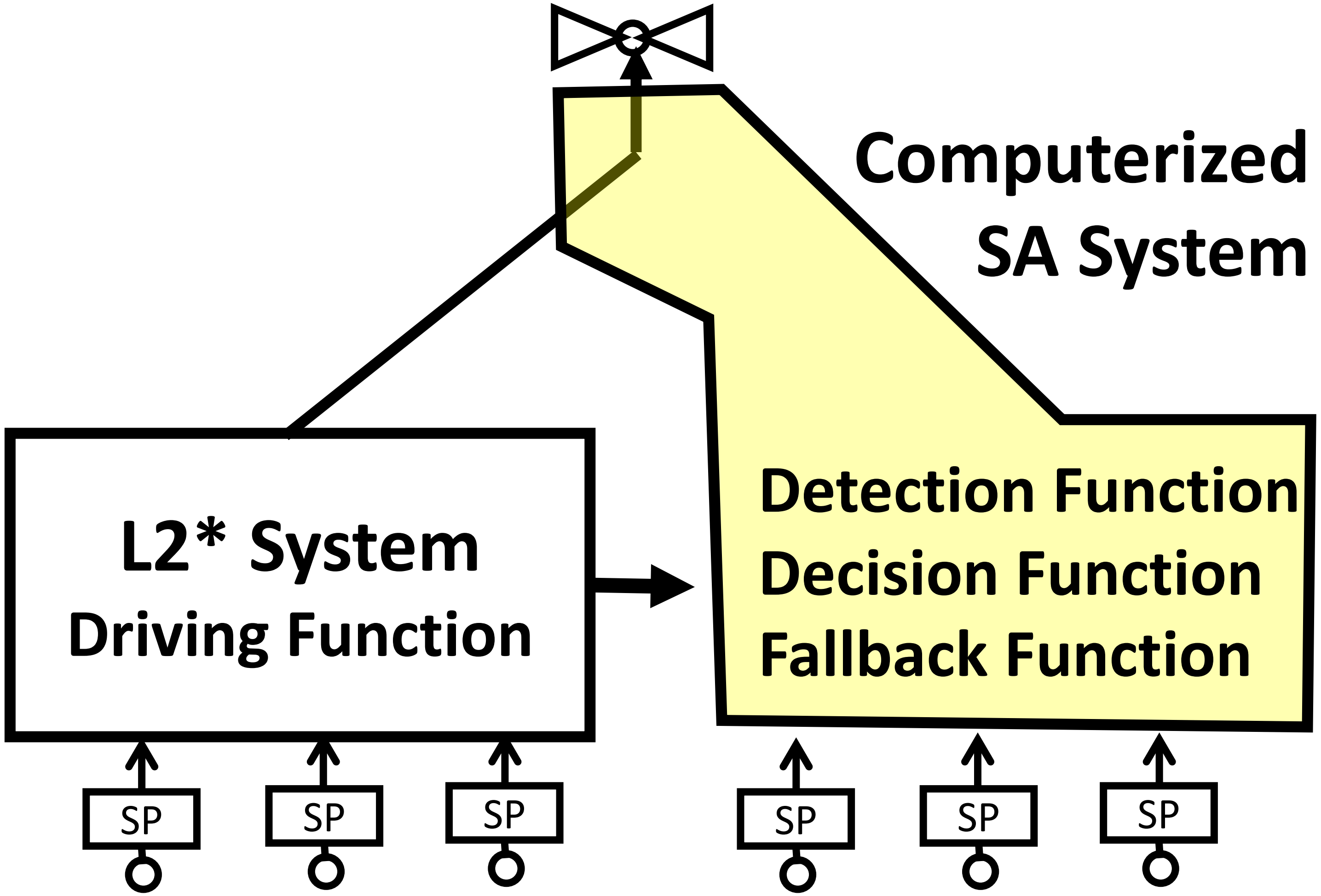
Distinguish between the cause of failure and the effect of a failure!— first mitigate the effect and later eliminate the cause.

First Try: *Computerized SA System* replaces *the Human Driver*

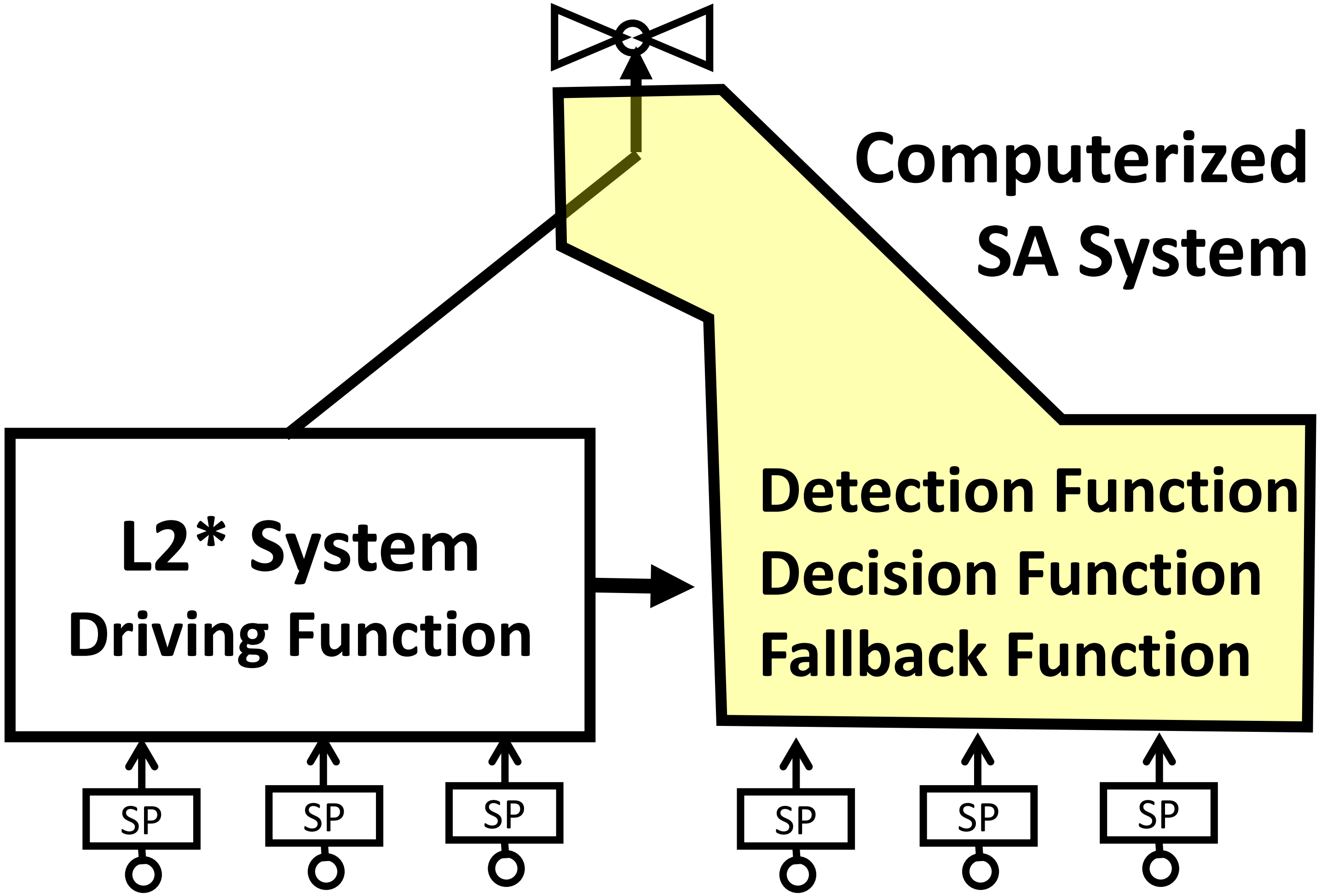




First Try: Two *Fault-Containment Units (FCUs)*



First Try: Two *Fault-Containment Units (FCUs)*



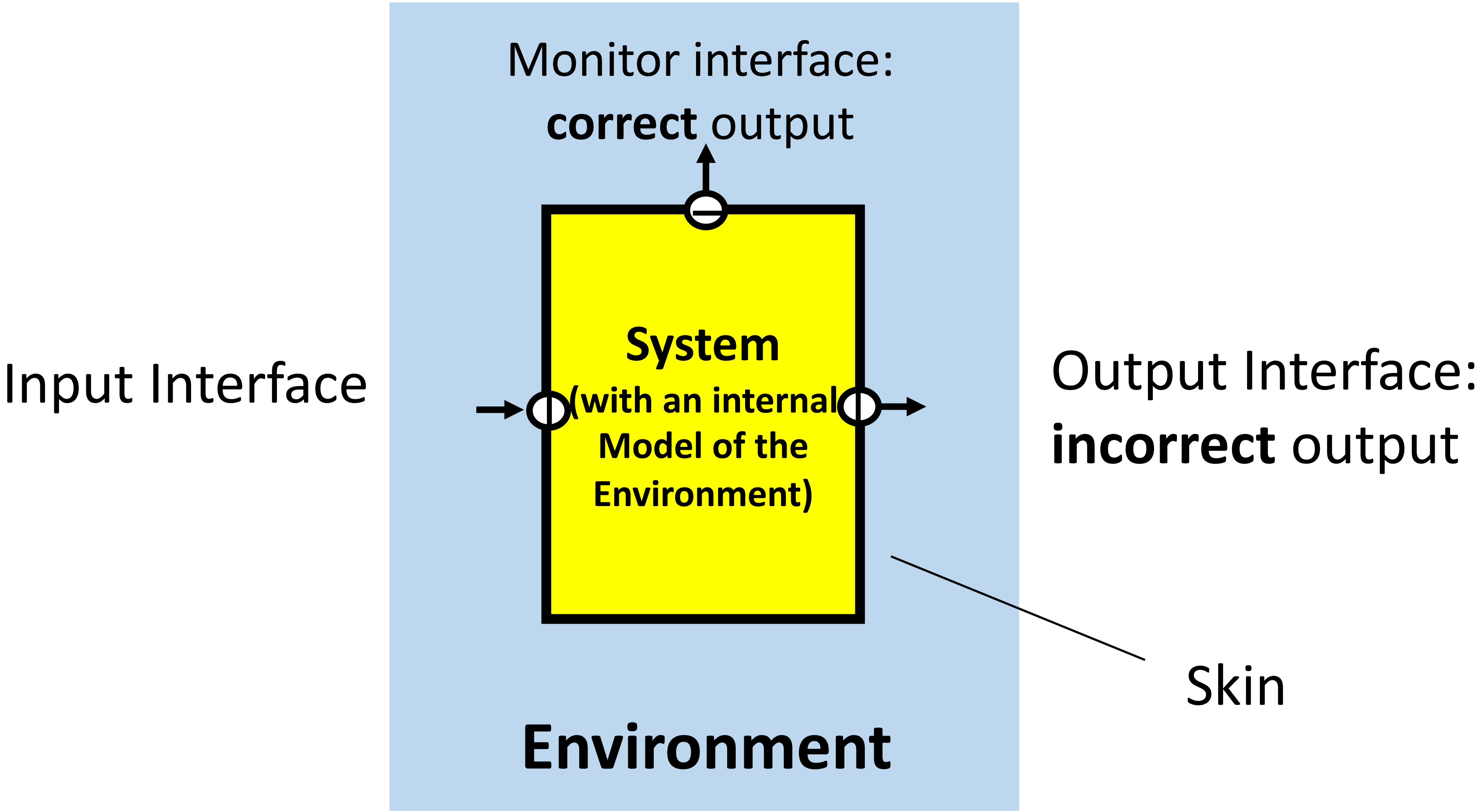
This is not a good idea!
If the computerized SA System is faulty, then this *faulty system* can take control of the vehicle and cause an accident.

Failure Modes of a Fault Containment Unit (FCU)

A *fault-containment unit (FCU)* interacts with its environment solely by the output of timed messages. It can exhibit one of the following three *external failure modes* due to a fault:

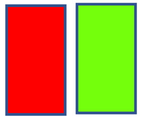
- ***fail silent:*** The faulty FCU detects all faults internally and does not deliver any message. *Assumption Coverage: < 1*
- ***fail consistent:*** In case of a fault, the FCU sends the same message to all its partners. *Assumption Coverage: < 1*
- ***fail Byzantine:*** No assumptions about the messages of a failing FCU (e.g. a *security incident, when the faulty FCU deceives its monitor).*
  Assumption Coverage = 1

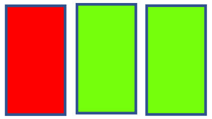
An Example for *Byzantine Behavior*

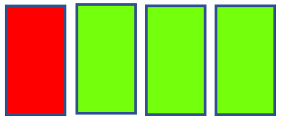


Number of FCUs for Fault Tolerance

Number of FCUs needed to tolerate a **single** faulty FCU:

- fail *silent FCUs*  *Assumption Coverage: < 1*

- fail *consistent FCUs*  *Assumption Coverage: < 1*

- **fail *Byzantine FCUs***  ***Assumption Coverage = 1***

FCUs must fail independently.

The Byzantine Fault-Model covers *Intrusions*

- If an intruder is successful and gets full control of a system, then he can produce *any kind of inconsistent behavior.* This is exactly the definition of a Byzantine fault.
- If a system is designed to mitigate a single Byzantine fault in any one of its subsystems, then the system will also handle any single intrusion.
- An intrusion in a (single) fault-tolerant system is only successful, if two different subsystems are compromised at the same time. Design diversity implies the need for intrusion diversity.

***Security* and *Safety* are thus two sides of the same coin.**

Do we have to assume a *Byzantine Fault* of an FCU?

- Digital Computer have the ugly property that ***small changes in structure*** *can* lead to ***catastrophic changes in behavior*** (e.g. a *bit flip* in a computer program can have unpredictable consequences).
- ISO 26262, requires in Section 7.4.3.1 an *inductive analysis* to determine the effects of a hardware fault—*at what level is this doable*?
- What are the worst-case consequences of a single bit flip (caused by an SEU in Hardware or a *Heisenbug* in the Software) for the service of an FCU?

If we assume only *non-Byzantine Failures* of an FCU, then intrusions are not covered and the *assumption coverage* is less than one.

Look at the 2003 paper by Kevin Driscoll!

Mitigating a Byzantine Failure:

In order to mitigate a **single** Byzantine failure we need

- four subsystems that are FCUs
- Byzantine Agreement Protocols among the four subsystems to mitigate the failure of one of the subsystems.

If none of the four subsystems can be trusted (assumed to be correct), then the Byzantine Agreement Protocols are expensive with respect to number of message exchanges and required time.

If one of the subsystems is assumed to be correct (simple enough to be free of a software error and executed on fault-tolerant hardware) then the speed and the complexity of the Byzantine Agreement Protocols can be substantially reduced.

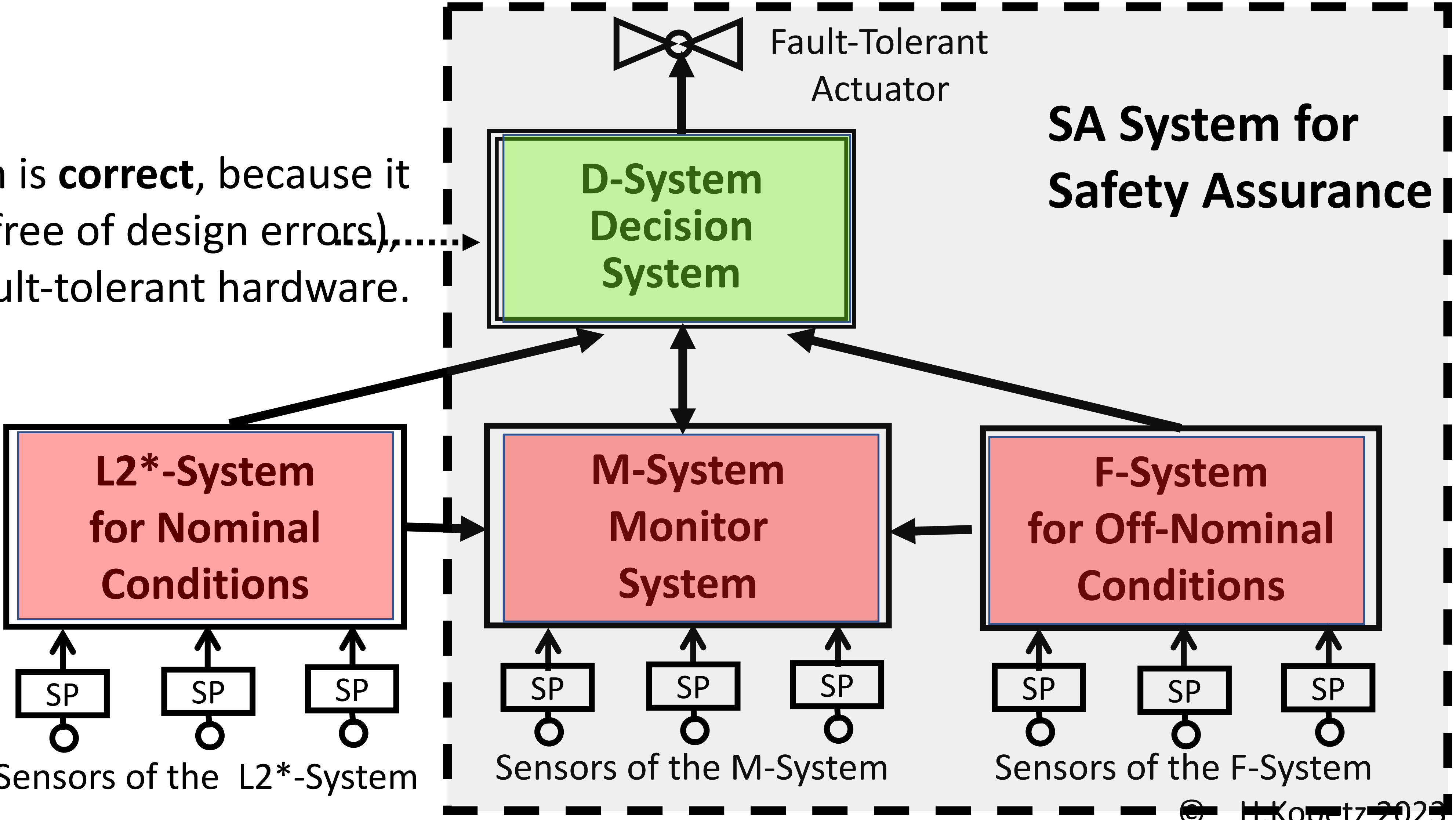
See the paper by Lamport (1982) : *The Byzantine Generals Problem.*

One Solution: Four FCUs, one of them is correct!

D system is **correct**, because it is ***simple*** (free of design errors), executed on fault-tolerant hardware.

One Complex FCU can fail

SP: Sensor Preprocessing



L2*-System: The Computer Controlled Driving Subsystem—*complex*

Purpose: To autonomously control the vehicle under nominal conditions (Basically an extended SAE Level 2 System).

Interfaces: Periodic Transmission of the Setpoints for Acceleration (Braking) and Steering to the D-System and the planned Trajectory to the M-System. (*A Trajectory is a sequence of timed waypoints.)*

Assumptions: Vehicle o.k., ODD o.k, All drivers adhere to highway code, etc.

Addition to Current L2 Systems: Run time error detection — e.g., Detection of cases that are *Out of Distribution (OOD)* of the ML network.

Possible Failure Modes: Byzantine

Estimated MTTF: 1000 operational hours

F-System: Fallback Subsystem—*complex*

Purpose: To bring the vehicle from the current state to a safe state.

Interfaces: Periodic Transmission of the Setpoints for Acceleration (Braking) and Steering to the D-System and a life-sign to the M-System.

Assumptions: Vehicle not o.k., ODD violated, Drivers may not observe the traffic code, etc.

Possible Failure Modes: Byzantine

Estimated Failure Rate on Demand: 1 Failure in 200 demands.

The F-System must be able to properly handle a scenario that has not been encountered up to now. The logic of the F-System must support know*ledge-based reasoning* and ***transfer learning***.

M-System: Monitoring Subsystem—*complex*

Purpose:

- Detect an unsafe trajectory of the L2*-System,
- Check if the F-System is o.k and if *nominal conditions* prevail.

A *fail positive failure* of the M-System is critical.

Interfaces: Reception of the planned trajectory from the L2*-System.
Periodic Transmission of safety assessment of the trajectory to the D-System and the state of the F-System to the L2*-Sysem.

Assumptions: Sensors and Computer o.k.

Possible Failure Modes: Byzantine

Estimated MTTF: 1000 operational hours

D-System: Fault Tolerant Decision Subsystem—*simple*

Purpose: To decide which setpoints are handed to the actuators.

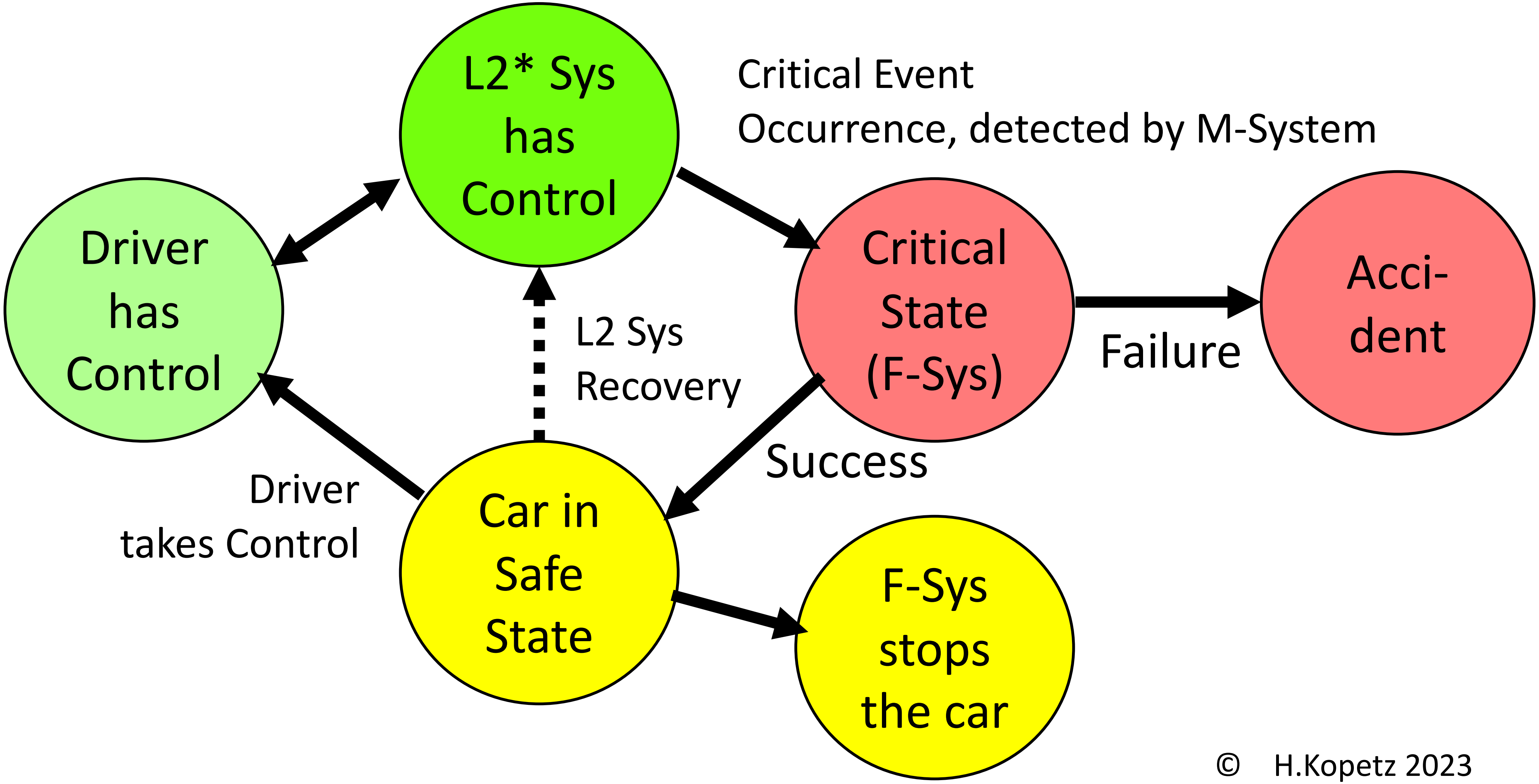
Interfaces: Periodic Reception of the Setpoints from the the L2*-System and the F-System and the safety assessment from the M-System. *Periodic Transmission of the received setpoints from the L2* system to the M-System.*

Assumptions: hardware *fault-tolerant* and software *correct*.

Possible Failure Modes: none

Estimated MTTF: meets ultra-dependable requirement.

State Transitions



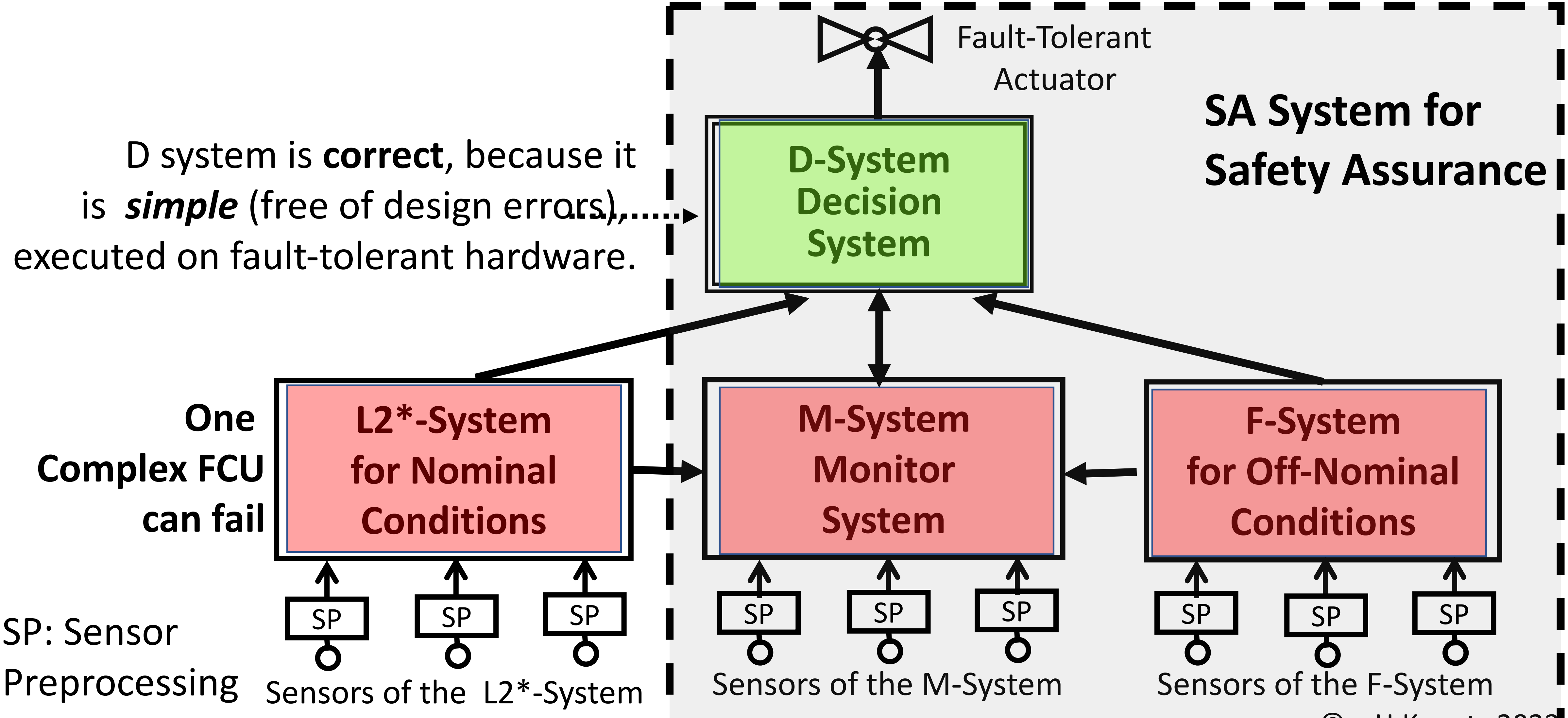
One Important Assumption: No Correlated Failures

In this architecture proposal, the probability of correlated failures of FCUs is reduced by:

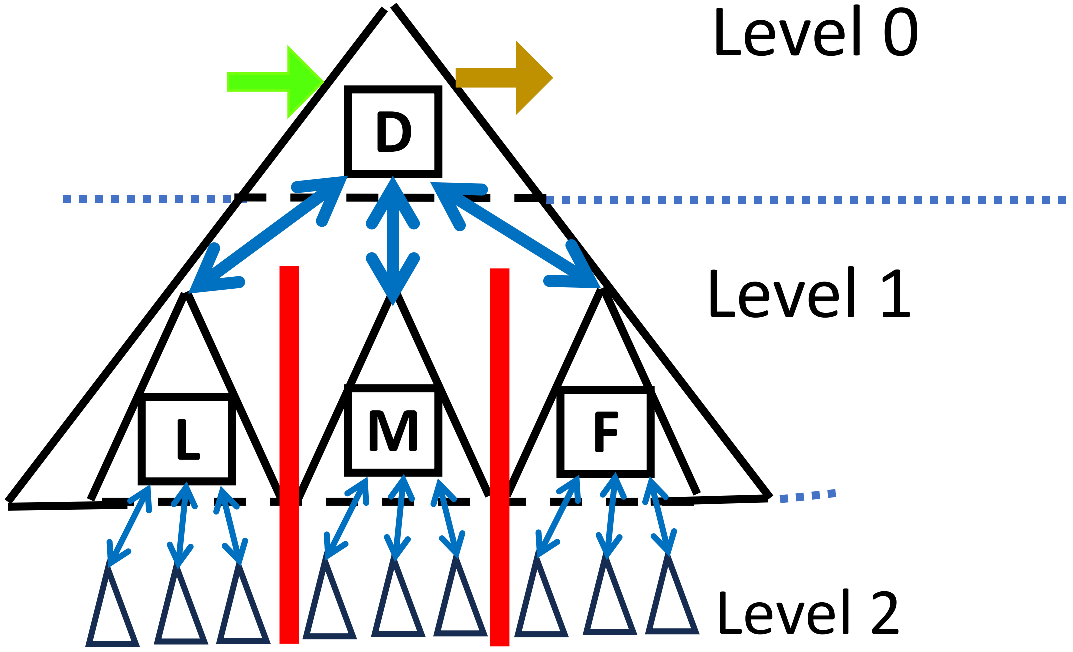
- **Diversity of purpose** of the internal models and of the algorithms in the three complex subsystems (the L2* system, the M-system and the F-system—it is not *TMR*).
- **Diverse execution environments** (hardware, sensors, operating systems, power supply, etc.) of the three complex subsystems.
- **Diverse Sensors and physical viewpoints**
- **Diverse design teams** that do not communicate beyond the establishment of the common system-level interfaces.

There will always be some correlation due to the same external environment. This correlation must be assessed experimentally.

The Four FCUs form an *Open Holonic Hierarchy*



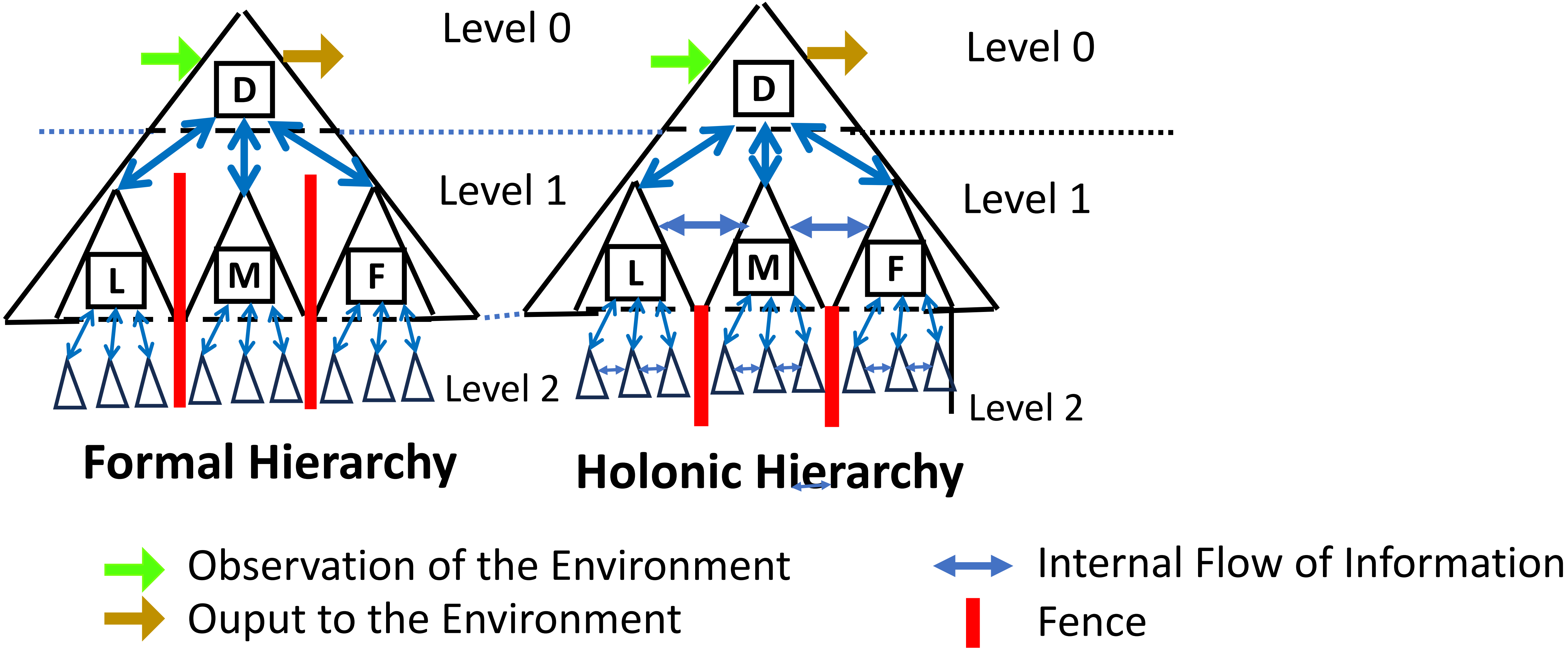
Multi-level Hierarchies



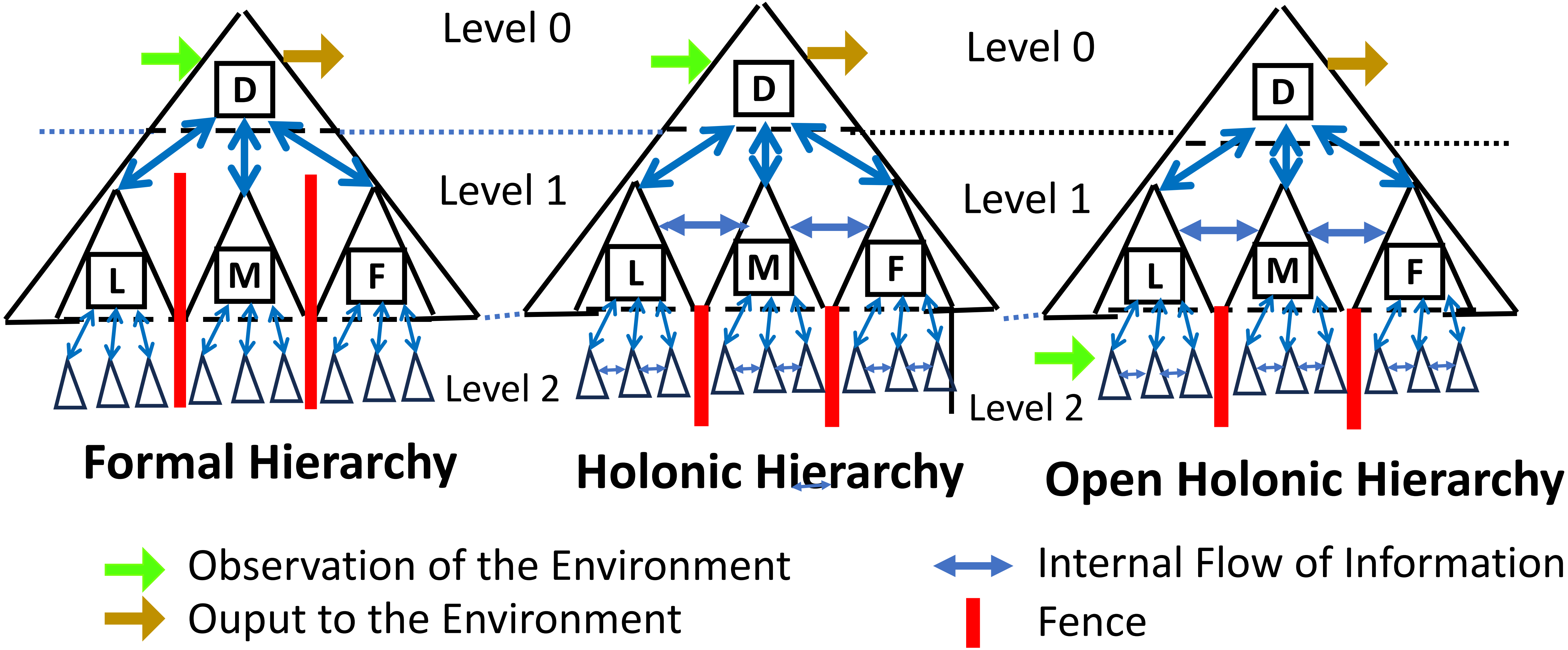
Formal Hierarchy

- Observation of the Environment
- Ouput to the Environment
- Internal Flow of Information
- Fence

Multi-level Hierarchies



Multi-level Hierarchies



Open Holonic Hierarchies

Open Holonic Hierarchies are well suited to structure the design of a large safety critical embedded system:

- The structure supports the interactions of the level-1 subsystems that leads to the *emergence of intended new properties.*
- The structure support the *diversity of implementation* to mitigate design errors.
- The *diversity of observation* of the system environment reduces the probability of errors in perception.
- The level-1 subsystems are self-contained Fault Containment Units with well-defined interfaces at level-1 and no interaction below level-1, supporting *Goal Clarity*.

Sketch of a Safety-Case

Safety Goal: Time to an unmitigated critical event 100 000 hours.

Assumptions, must be justified by experimental evidence:

- L2* Sys and the M-Sys fail in a Byzantine failure mode every 1000 hours.
- F-system fails in one out of 200 demands.
- D-Sys is correct

Safety Argument: A single Byzantine failure of or a single intrusion into one the three complex systems (L2*-System, M-System, F-System) is mitigated.

System Failure: If (L2* fails .or. M fails) .and. (F fails)

demand/time x failure/demand = failure/time

System Failure if two complex sub systems fail at about the same time!

Conclusions

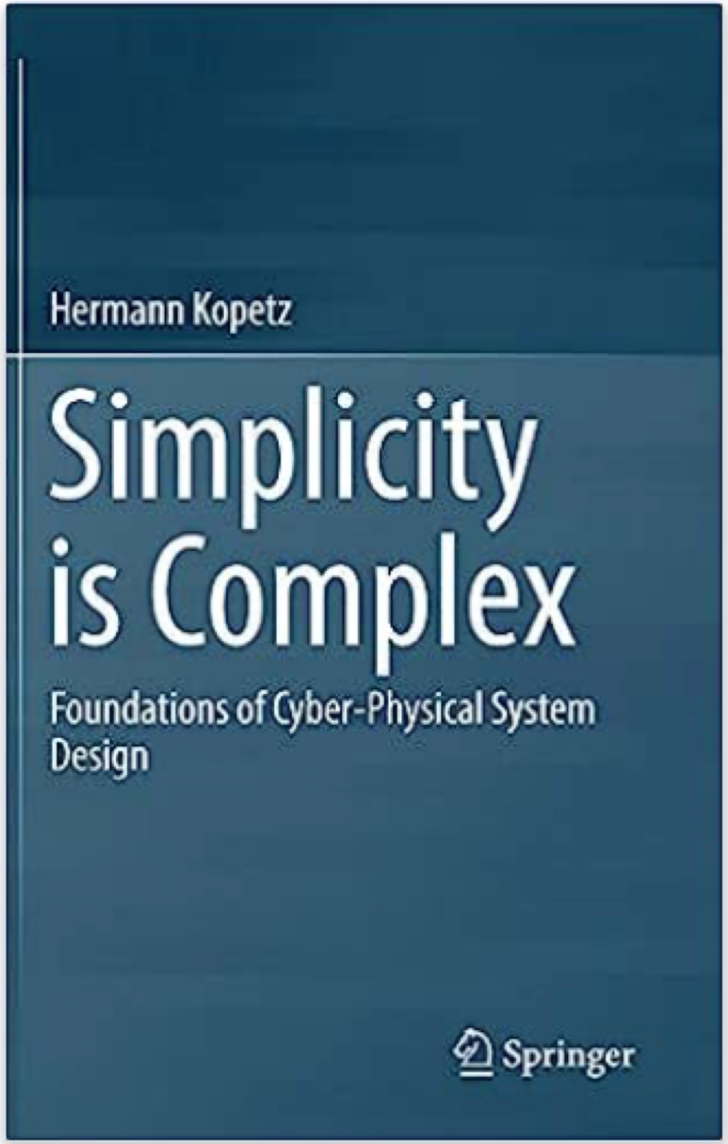
It the domain of fully autonomous large ultra-dependable embedded computer applications

- ***Fault-tolerance with design diversity* is absolutely essential**
- ***Open Holonic Hierarchies* are well suited to structure the design**
- ***An independent safety assurance (SA) subsystem* must be provided to mitigate *off-nominal* conditions.**


Thank you—
Any Questions?

Further Reading . . .

Published:
July 18
2019



Published:
March 18
2022