



life.augmented

# Functional Safety in Electronic Systems: Principles and Applications

Alessandro Bastoni

Functional Safety Expert

STMicroelectronics

# Lesson #5

Diagnostics (hw and sw), system evolution, hw/sw partitioning

## Summary:

- Safe State
- System evolution in time
- Mode of operations, PST, test frequency
- Elements on diagnostics, hw, sw, system

# De-rating

Hardware components must be operated at levels which are guaranteed by the design of the system to be well below the maximum specification ratings.

De-rating is the practice of ensuring that - under all normal operating circumstances - hardware components are operated well below their maximum stress levels – it can be defined as a safety margin.

IEC61508 recommend derating (2/3 factor) for hardware components

IAO13849-1 explicitly mention derating as one of economized additional techniques to lower the possibility of systematic failures (again 2/3 factor).

De-rating can play a relevant role in guaranteeing that the assumption “failure rate = constant” is still valid.

# Safe State

Safe State is formally defined in both main standards:

IEC61508-4: state of the EUC when safety is achieved

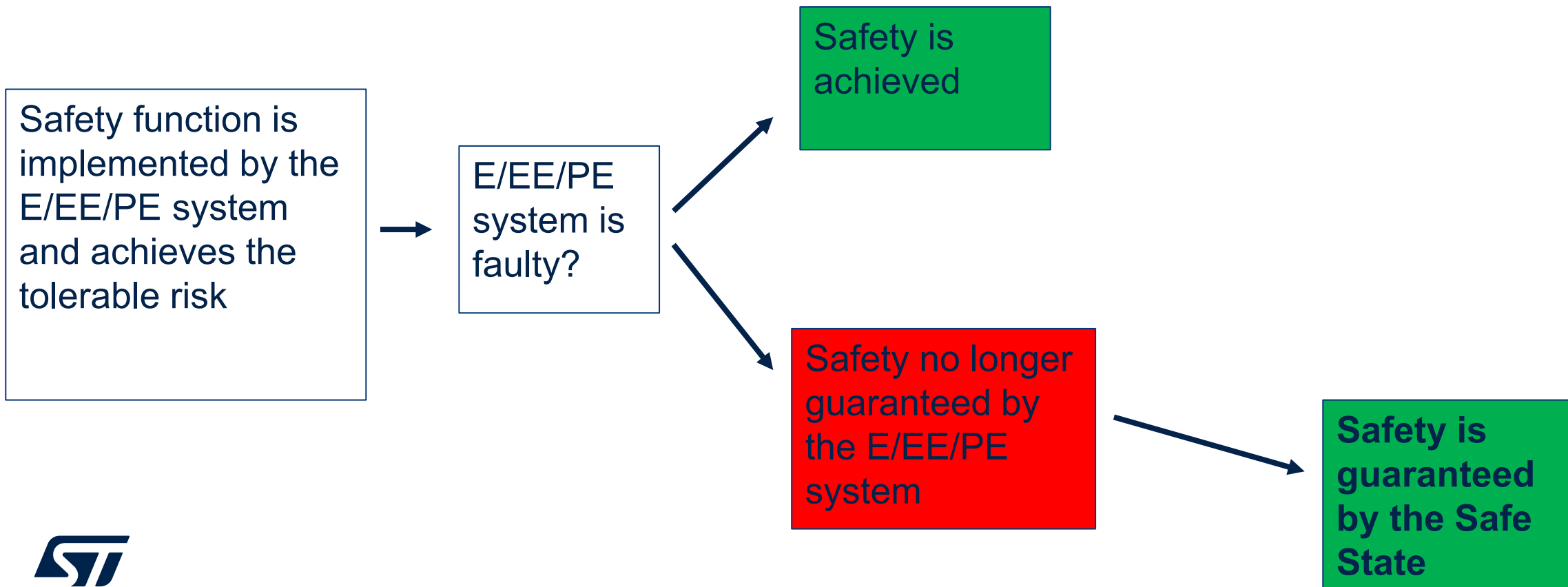
ISO26262-1: operating mode, in case of a failure, of an item without an unreasonable level of risk

In IEC61508 the system must be always in „Safe State“, either when it's perfectly working or faulty.

In the common usage, “Safe State“ indicates the specific state where the system guarantees the safety in case of failure (ISO26262 bias) – usually, in a “degraded” mode.

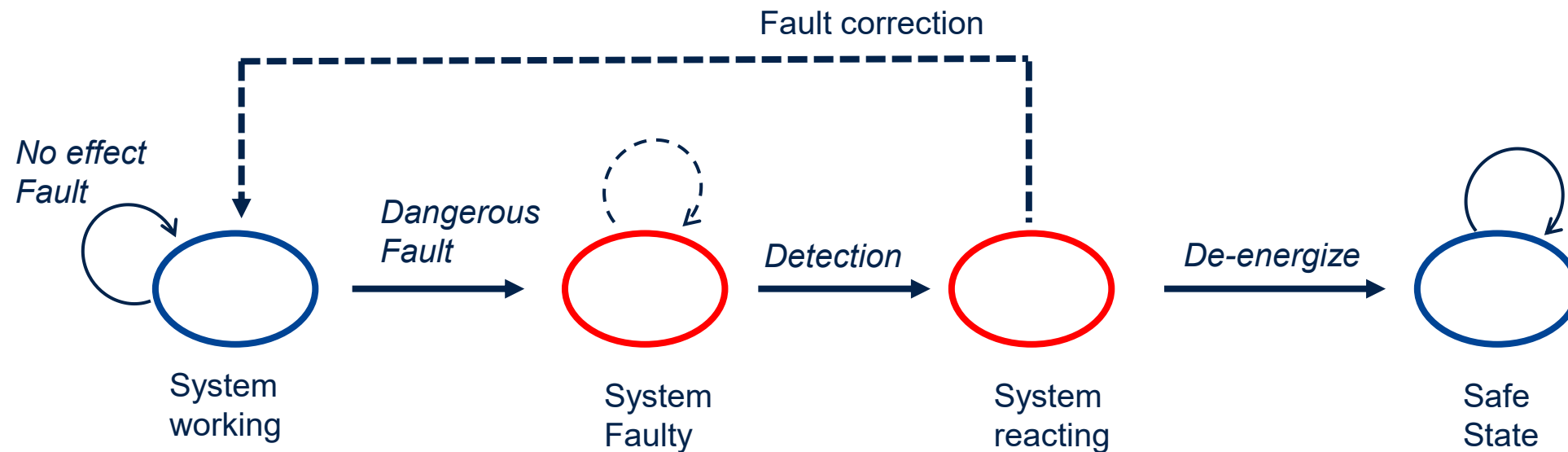
# Safe State

Safe State definition is never generic, as it is strongly connected to the final application i.e. to the way the outputs/decisions are communicated/actuated (the safety function). As per safety function, safe state definition is at system level (a local safe state can be defined as well).

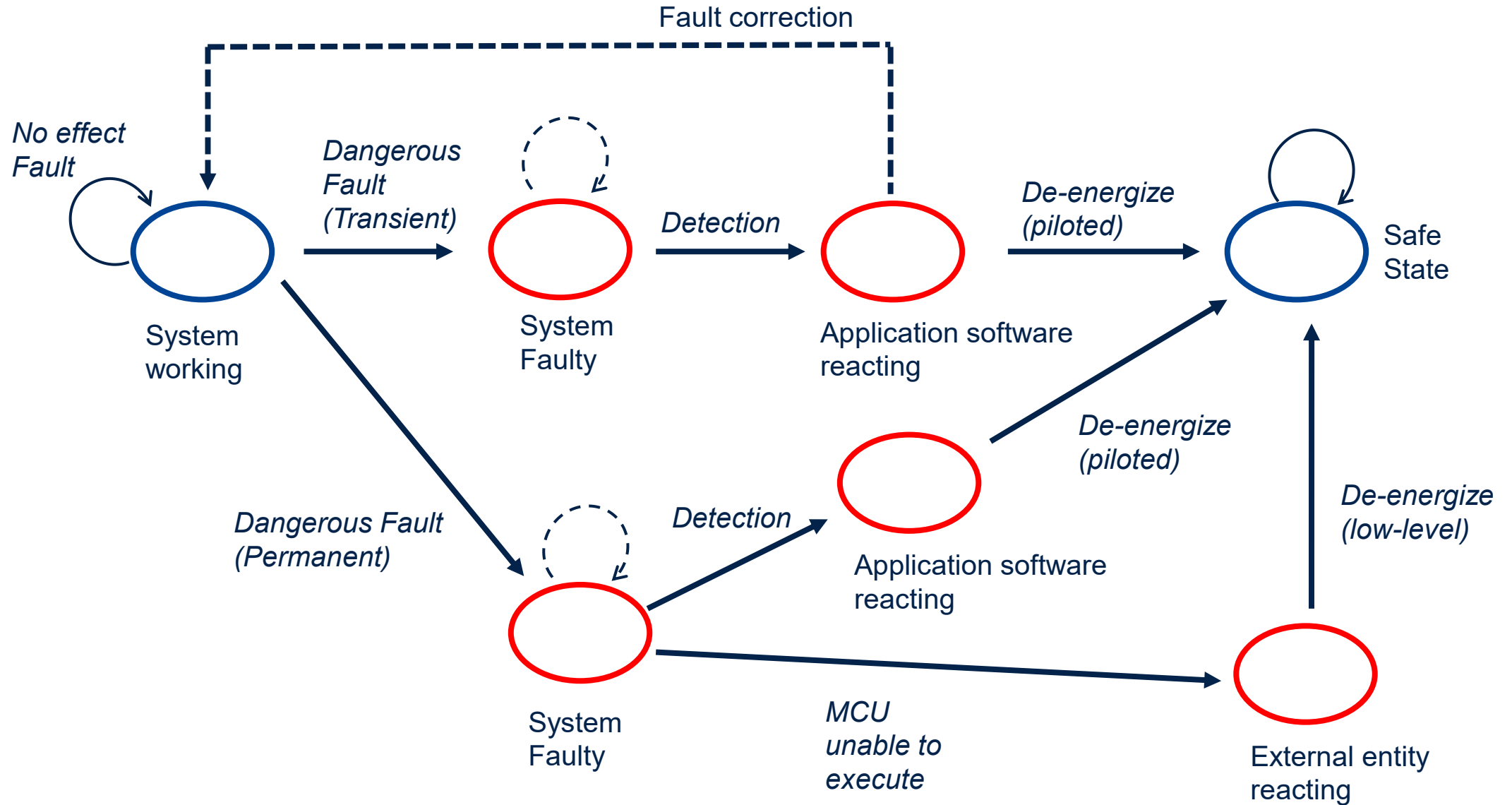


# Safe State and system evolution (general)

**Important:** the Safe State should be always reachable - in an independent way from the actual failure affecting the system.

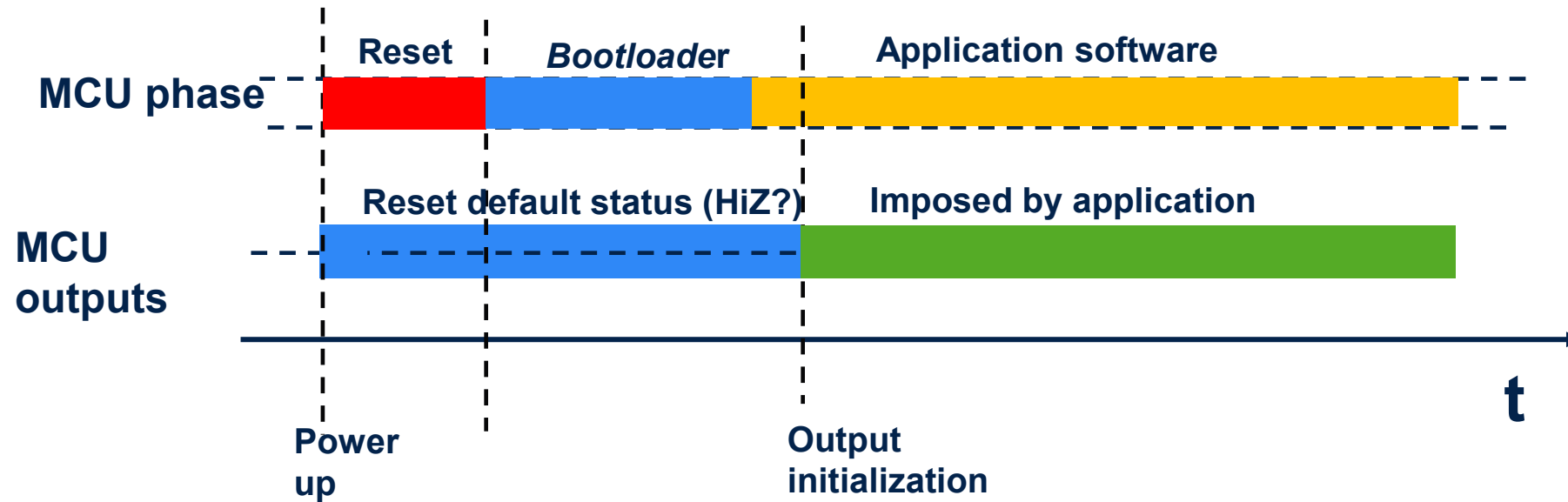


# Safe State and system evolution (adding MCU and fault models)



# Safe State and system boot

Important: the Safe State must be always guaranteed, even when no software execution is possible

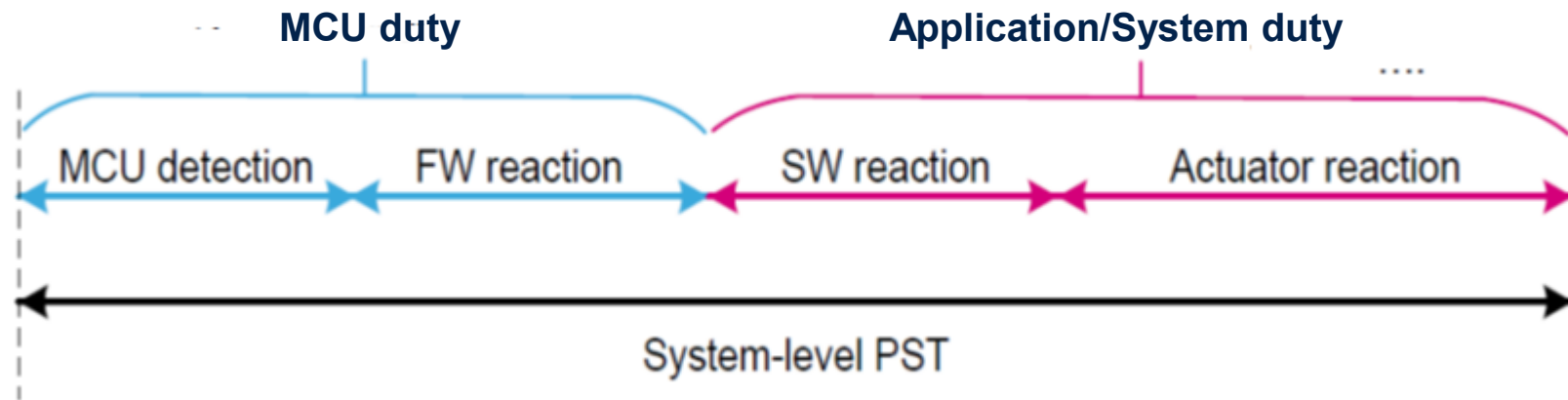


The startup phase (including possible bootloader execution, longer for flash-less devices) requires additional measures external to the MCU to guarantee the safe state

Faults occurring during a power-up phase may let the system hang-up in one of the initial phases.

# Time sequence for PST limitation understanding

This is the typical time/causal sequence from the detection of a fault to the achievement of the Safe State.



Note that PST Process Safety Time is defined as the time between the rise of a dangerous failure and the moment a real hazard occurs. In CM systems, diagnostics must be able to intervene within the PST

*Attention: not comprehensive of corner cases related to CPU hang up or impossibility to correctly execute software actions. Because of that, Safe State transition by external entities (e.g. a watchdog) is needed.*

# Mode of operations (IEC61508)

In IEC61508 the Mode of operation is related to which frequency the Safety Function is demanded; it drives the target metric (PFD/PFH) and testing frequency:

Low demand mode: safety function is only performed on demand, to transfer the EUC into a specified safe state, and where the frequency of demands is no greater than one per year

High demand mode: safety function is only performed on demand, to transfer the EUC into a specified safe state, and where the frequency of demands is greater than one per year

Continuous mode: where the safety function retains the EUC in a safe state as part of normal operation

LD → PFD Probability of Failure on Demand: probability

HD/CM → PFH (Probability of Failure per Hour: probability/time)

# Mode of operations (IEC61508)

The frequency of periodic diagnostic execution depends on the Mode: DC can be claimed just for safety mechanisms executed within the limits specified here below.

- ❑ On LD systems, proof test concept apply (refer to related slide).
- ❑ HD systems: test frequency is linked to the frequency of safety function demands (100x faster). This allows software-based concepts.
- ❑ CM systems require that each periodic diagnostic is executed at least once per PST (Process Safety Time), introducing related concept

# Mode of operations (IEC61508)

Examples of LD/HD/CM safety functions:

Mode	Safety Function	Description
LD	Emergency Shutdown System (ESD)	Shuts down the process safely in case of a hazardous event (e.g., gas leak, fire).
HD	Safety Interlock Systems	Frequently invoked to prevent unsafe operations (e.g., opening a valve only under safe conditions)
CM	Fire and Gas Detection System (continuous monitoring)	Continuously monitors for fire or gas presence and triggers alarms or shutdowns immediately upon detection

*Note that in the same safety system it is possible to have coexisting safety functions with different Mode of operations (e.g. in a Fire systems, one CM SF for fire/smoke detection and one LD SF for sprinklers deployment)*

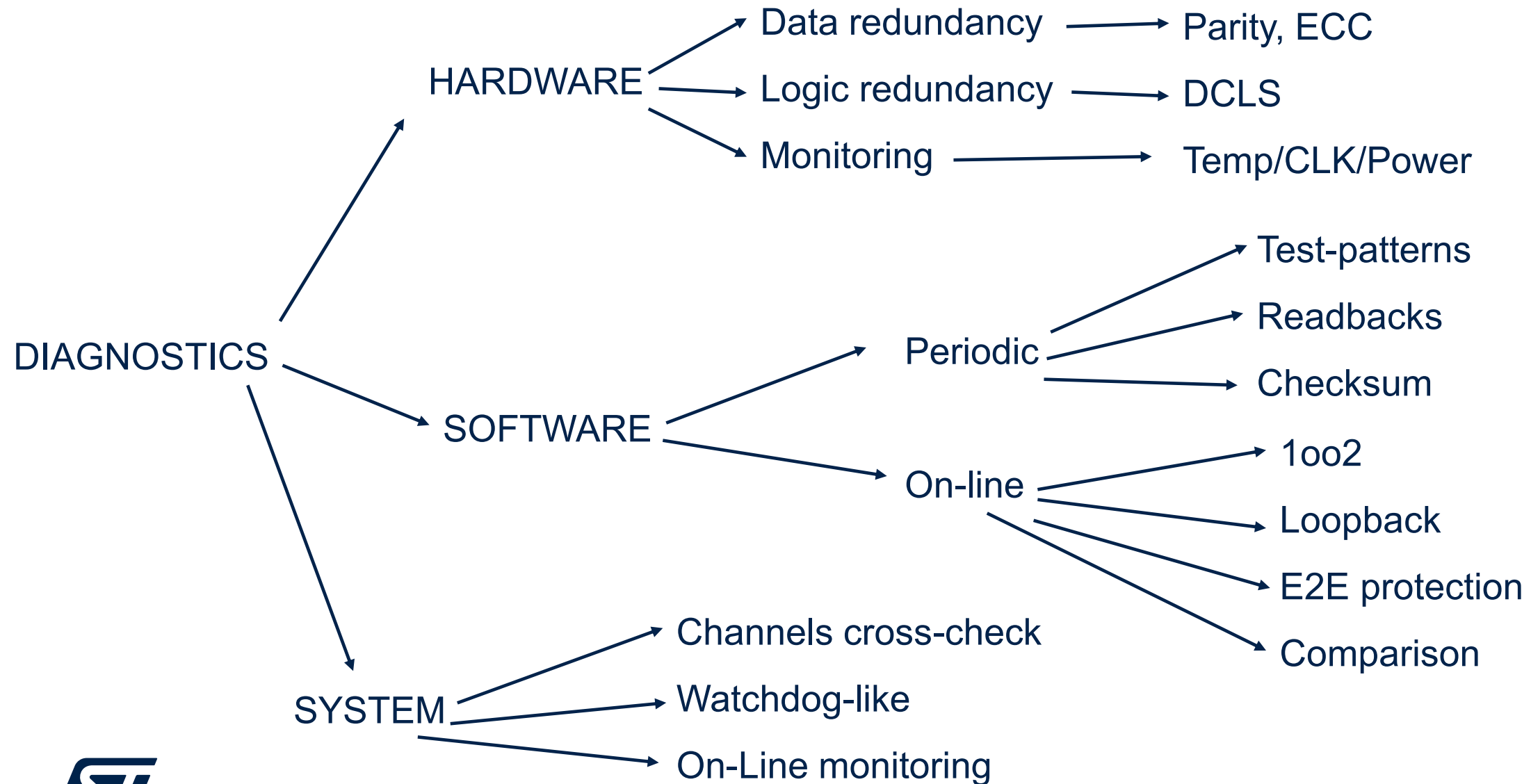
# The proof test concept

Proof test is defined as a periodic test performed to detect dangerous hidden failures in a safety-related system, to allow (if necessary) a repair which can restore the system to an “as new” condition or as close as practical to that condition.

Proof test is the main way to ensure safety on Low Demand systems, where PFD is the dominant metric. The periodicity of the test is imposed by the target SIL level and the device failure rate (the higher  $\lambda_{DU}$  is, the shorter is the interval)

Proof test can be applied to HD/CM systems as well with the intention to address “hidden” failures related to structures difficult to be tested during operating time like diagnostic functions, error chain (reporting and reaction), partially corrected faults. Usually, the effect on PFH is negligible.

# Safety mechanisms classification



# Pro/Cons per categories



# Safety mechanisms characteristics

There are common characteristics to be defined when dealing with diagnostics/safety mechanisms

- ☐ Addressed fault model (permanent/transient/both?)
- ☐ Periodicity (continuous/on demand/periodic)
- ☐ Error reaction (message/flag/interrupt)
- ☐ Error correction (yes/no/partial)
- ☐ Test/multiple fault protection (listing alternative diagnostic(s) for faults preventing the correct functioning of the safety mechanisms itself)
- ☐ Initialization/configuration (some diagnostics are always on, others may need configuration by sw, etc)

# Achieved DC – how to establish

There are multiple patterns to establish the achieved Diagnostic Coverage for a given safety mechanism

- ❑ Reference Tables from safety standards: many safety standards include reference table where for a set of high-level specified diagnostics a range/indication for achievable (\*) DC is provided. Usually, enumerated values (High=99%, Medium=90%, Low=60%)
- ❑ Fault injection/simulation: the component is modelled inside a tool able to emulate the faults affecting the hardware, and the reaction capability of diagnostics. DC is computed on statistical way

(\*) pay attention: “achievable” is not “achieved”. Accordingly, those values are considered the maximum achievable DC for such a diagnostics (!).

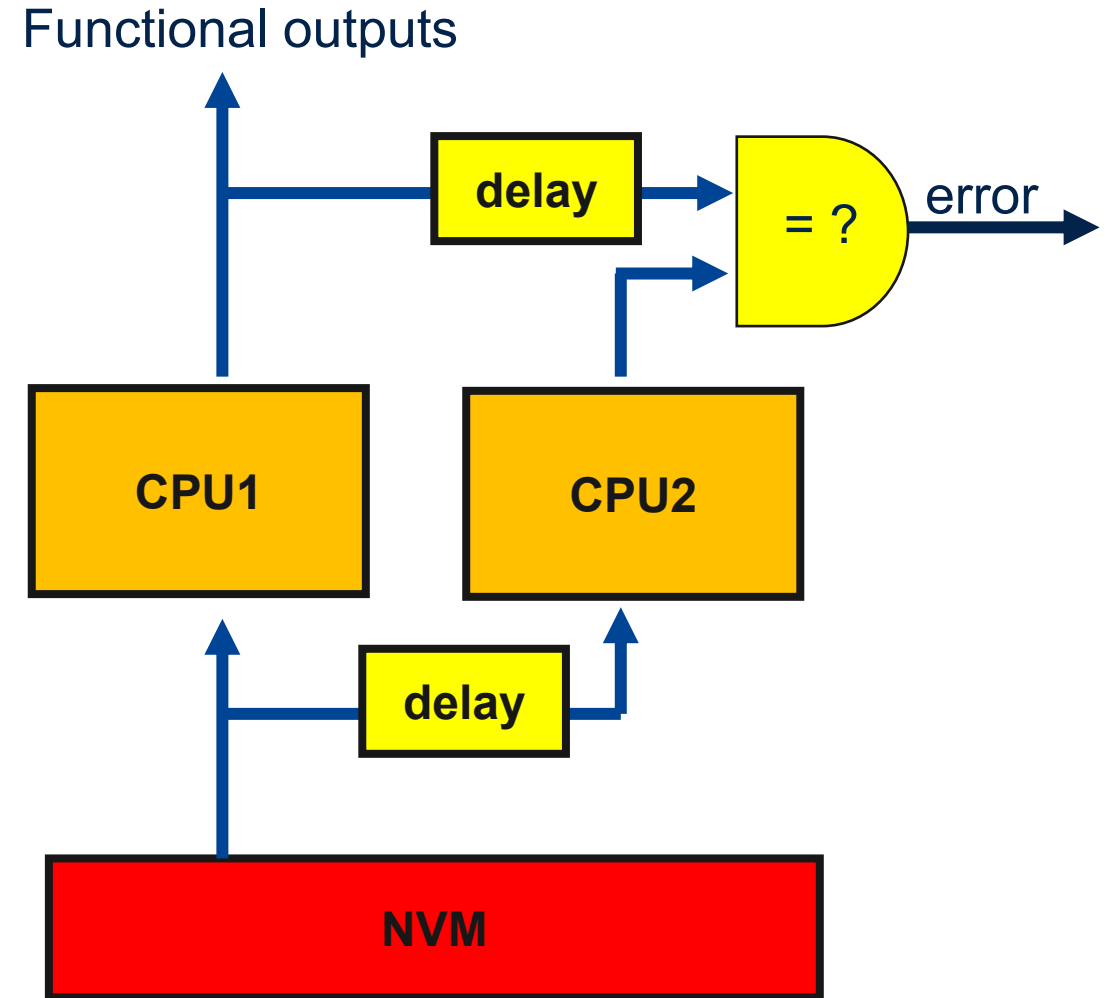
# Dual Core Lock Step (DCLS)

## PROS

- Fast fault detection
- Detects both permanent and transient failures
- High achieved DC
- Application independent

## CONS

- Cost and complexity so available just on safety-ready devices
- Second CPU is just for monitoring, so HFT=0
- Still needs additional entities to manage failures preventing software execution



# Parity bits

A parity bit is added to each word (multiple schemes are possible), enabling single bit error detection when data are read.

## PROS

- Fast fault detection
- Detects both permanent and transient failures
- Best compromise between device cost and medium achieved DC
- Application independent
- No time penalties from end user perspectives

## CONS

- Coverage guaranteed just on single bit failures (50% on dual, etc...)
- Achieved coverage is questionable because of differences between safety standard guidance
- Usually, check is done on reading → error accumulation must be managed (scrubbing)
- If address lines are not included, additional tests for address decoder needed

A multi-bit redundant code is added to each word (different schemes are possible), enabling single error correction and double error detection when data are read..

## PROS

- Fast fault detection
- Allows single error correction so increasing system availability
- Detects both permanent and transient failures
- High achieved DC
- Application independent

- No time penalties from end user perspectives

## CONS

- Usually, check is done on reading → error accumulation must be managed (scrubbing)
- Usually, correction is done just on data sent to CPU and not on cells → error still there
- If address lines are not included, additional tests for address decoder needed

# Internal watchdog

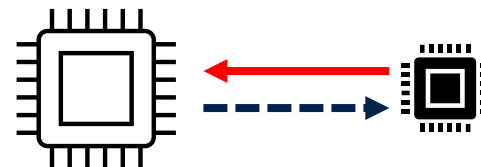
Forces a CPU reset when the required action from software (e.g. register write with a key) is not executed within the programmed period. Timing policy can be enforced by window requirement.

## PROS

- Manages permanent or transient failures affecting correct software execution capability
- Contributes to systematic capability of the software by intercepting wrong control flow or timing

## CONS

- Lack of hardware diversity as it shares with the CPU the same silicon substrate, and often power/clock as well
- Unable to completely manage failures leading to software execution inability
- Often overlapped by external watchdog as required by IEC61508-2, Table A.1/Table A.14.



# Useful references on safety mechanisms

Reference [6] lists in section “7 Brief Description of Diagnostics “ more than 100 different safety mechanisms defined in a automotive safety ASIL D microcontroller (TI). Check also “Appendix A Summary of Recommended Safety Feature Usage” where an exhaustive table provides a synoptic view of all characteristics for the listed diagnostics.

Reference [7] provides similar descriptions in section “3.6 Hardware and software diagnostics”, in this case an IEC 61508 wording is adopted across the document. The target is also in this case a MCU with intermediate safety level SIL 2.

Reference [8] offers a different perspective on a “simpler” device, a PMIC. Refer to section “5 TPS65919-Q1 Architecture Safety Mechanisms and Assumptions of Use” for a view of the very different set of dedicated diagnostics.

# Bibliography



# Reference documents 1/2

[R1]: Microelectronics Reliability: Physics-of-Failure Based Modeling and Lifetime Evaluation - Jet Propulsion Laboratory California Institute of Technology Pasadena, California

[R2]: : Semiconductor Reliability Handbook – Renesas Electronics, Rev.2.50 Jan. 2017

[R3]: ExoMars 2016 - Schiaparelli Anomaly Inquiry (ESA) downloaded from <https://exploration.esa.int/web/mars/-/59176-exomars-2016-schiaparelli-anomaly-inquiry>

[R4]: Fault Tree Handbook with Aerospace Applications - NASA Office of Safety and Mission Assurance, V 1.1 , 2002

[R5]: open FTA software can be found on the web, e.g. <https://www.fault-tree-analysis.com/free-fault-tree-analysis-software>, or check for OpenFTA download

# Reference documents 2/2

[R6]: Safety Manual for TMS570LS31x and TMS570LS21x Hercules™ ARM®-Based Safety Critical Microcontrollers

[R7]: UM2331- STM32H7 singlecore series safety manual STMicroelectronics – from <https://www.st.com/en/embedded-software/x-cube-stl.html#documentation>

[R8]: Safety Manual for TPS65919-Q1 Power Management Unit (PMU)

# Thank you

© STMicroelectronics - All rights reserved.

ST logo is a trademark or a registered trademark of STMicroelectronics International NV or its affiliates in the EU and/or other countries.

For additional information about ST trademarks, please refer to [www.st.com/trademarks](http://www.st.com/trademarks).

All other product or service names are the property of their respective owners.



life.augmented