SAFEXPLAIN

Safe and Explainable Critical Embedded Systems based on AI

Safe and explainable critical embedded systems based on AI

Jaume Abella[1], Jon Perez[2], Cristofer Englund[3], Bahram Zonooz[4], Gabriele Giordana[5], Carlo Donzella[6], Francisco J. Cazorla[1], Enrico Mezzetti[1], Isabel Serra[1], Axel Brando[1], Irune Agirre[2], Fernando Eizaguirre[2], Thanh Hai Bui[3], Elahe Arani[4], Fahad Sarfraz[4], Ajay Balasubramaniam[4], Ahmed Badar[4], Ilaria Bloise[5], Lorenzo Feruglio[5], Ilaria Cinelli[5], Davide Brighenti[7], Davide Cunial[7]

[1] Barcelona Supercomputing Center, Spain

[2] Ikerlan Technology Research Centre, Basque Research and Technology Alliance (BRTA), Spain

[3] RISE Research Institutes of Sweden, Sweden

[4] Navinfo Europe, The Netherlands

[5] AIKO s.r.l., Italy

[6] Exida Development s.r.l.

[7] Exida Engineering s.r.l., Italy

Funded by the European Union

This project has received funding from the European Union's Horizon Europe programme under grant agreement number 101069595.

In a nutshell

- The scene
  - **Critical Embedded Systems (CES)** increasingly rely on Artificial Intelligence (AI): automotive, space, railway, avionics, etc.
  - CES must undergo **certification/qualification**
  - AI at odds with functional safety certification/qualification processes (**lack of explainability**, **lack of traceability**, **data-dependent** software, **stochastic** nature)
- SAFEXPLAIN ambition: architecting DL solutions **enabling certification/qualification**
  - Making them **explainable** and **traceable**
  - Preserving **high performance**
  - Tailoring solutions to varying safety requirements by means of **different safety patterns**

OCT 2022 – SEP 2025

SAFEXPLAIN

Safe and Explainable
Critical Embedded Systems based on AI

BARCELONA SUPERCOMPUTING CENTER (BSC)
https://www.bsc.es/

IKERLAN, S. Coop (IKR)
https://www.ikerlan.es/

AIKO SRL (AIKO)
https://www.aikospace.com/

RISE RESEARCH INSTITUTES OF SWEDEN AB (RISE)
https://www.ri.se/

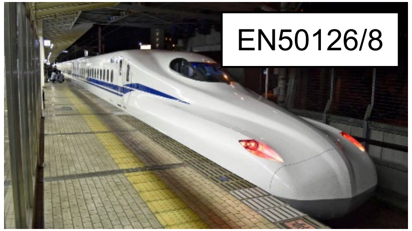
NAVINFO EUROPE BV (NAV)
https://www.navinfo.eu/

EXIDA DEVELOPMENT SRL (EXI)
https://www.exida-eu.com/

Jaume Abella
Project Coordinator

CES

- Failure or malfunction may result **severe harm** (e.g., casualties)
- Exhaustive **Verification and Validation** (V&V) process, and **safety measures** deployed to guarantee the safety goals are met
- Each domain has it's own guidelines and regulations for SW and HW



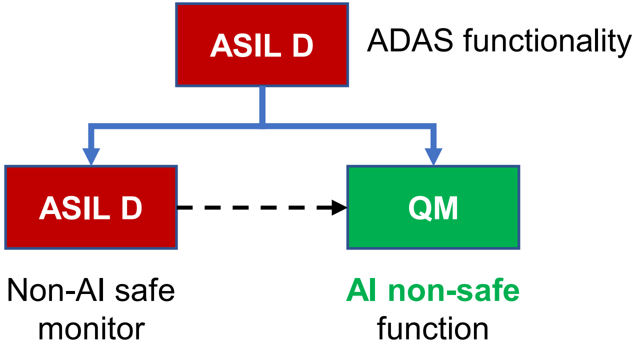
- ISO 26262 and ISO 21448 (SOTIF) for automotive

CES and AI

- The number of mechanical subsystems enhanced or completely replaced by electronic components is increasing
- Advanced software functions are becoming ubiquitous to control all aspects of CES, including safety related systems
- **AI techniques** are at the very heart of the realization of **advanced software functions** such as **computer vision for object detection** and tracking, path planning, driver-monitoring systems,…
  - E.g., You Only Look Once (YOLO) camera-based object detection system builds upon a Neural Network
- Autonomous operation
  - epitome of safety-related applications of AI in CES,
  - exemplifies the need for increasingly **high computing performance** whilst **making AI solutions to comply with FUSA** requirements

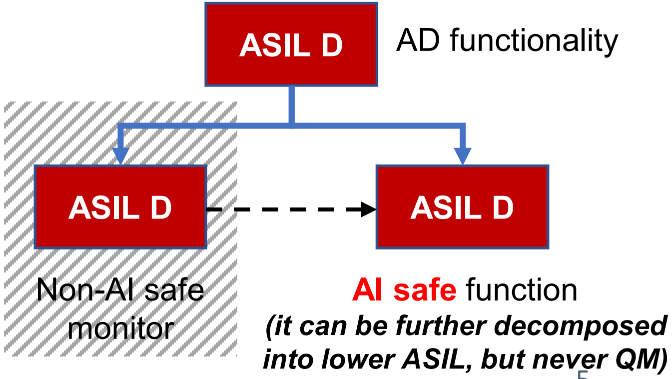
AI in Safety-critical systems so far and in the future

- When software/hardware implements safety-related functionality they inherit safety requirements
- Safety Integrity Level (SIL) decomposition
  - E.g., Automotive SIL (ASIL) from D (highest) to A (lowest), and then QM (no safety)
- **AI used in fail-safe systems** (i.e. systems with a safe state)
  - E.g., Advanced Driving Assistance Systems (ADAS) can notify misbehavior and transfer control to the driver



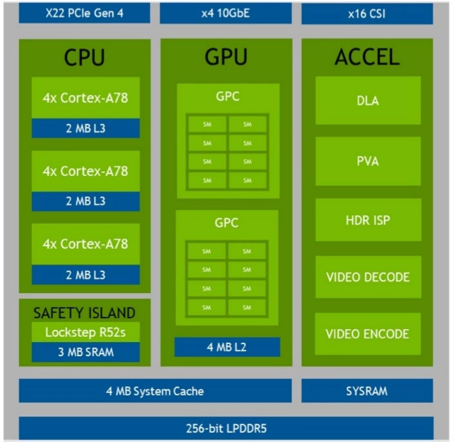
- With **autonomous systems** (e.g., autonomous cars) this is **not yet solved**
  - If no safe state available(*), or non-AI safe monitor is possible, hence AI components inherit safety requirements

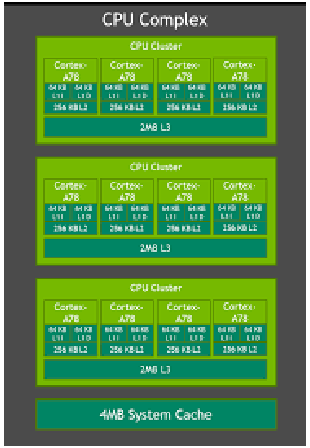
(*) The safe state must not use AI, otherwise we would recursively make AI-based components be fail-operational

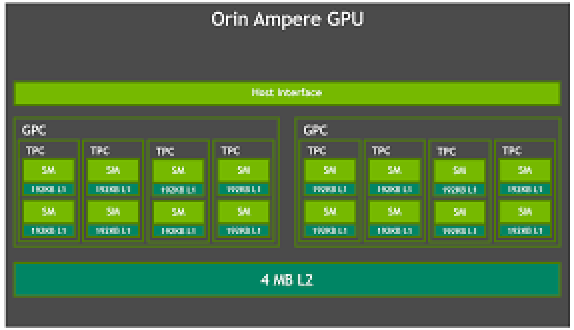


AI impact on the computing platform

- Software implements complex AI algorithms that manage huge amounts of data
- This carries huge computing performance requirements
- Hardware in safety-critical systems: from simple micro-controller to heterogeneous MPSoC with specific accelerators
- Complex MPSoC complicates established software timing V&V



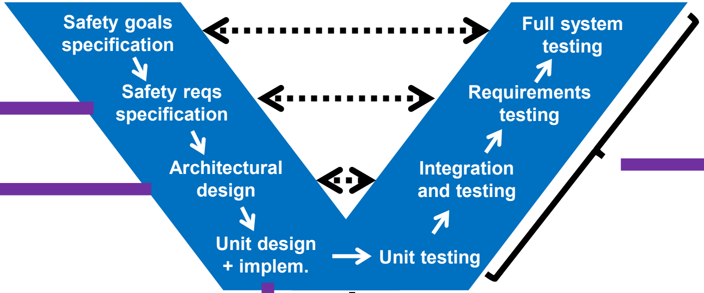




e.g. NVIDIA Orin
Source: NVIDIA

Safety-related Systems Development Process

- ISO 26262 software V-model



Precise and **unambiguous requirements**

- HW: admits failure rates due to random faults
- **SW (algorithm): deterministic. Systematic errors management**

Control SW:
- Data is abstracted (name, source, but not its values)
- **Algorithm is "data-independent"** (defined independently of how the data looks like)

Code (SW) designed based on human expert knowledge and physical laws/operational rules

Data used only for testing purposes: PASS/FAIL tests, diagnostics, quality assessment

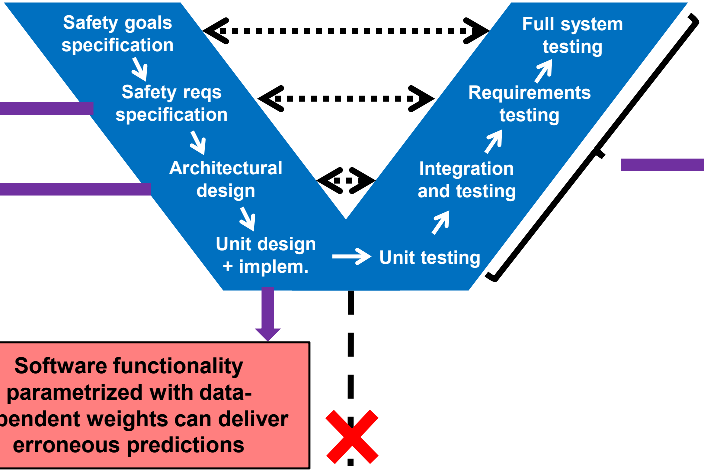
But **data does not determine the design** of the system

NO DATA USED TO DESIGN THE SYSTEM

DATA USED ONLY FOR TESTING

Safety-related Systems Development Process

- AI-related challenges



Precise and **unambiguous requirements**

- HW: admits failure rates due to random faults
- **SW (algorithm): Not always deterministic and limited predictability/explainability**

Data-defined SW:
- Algorithm **behavior is determined by actual data** (e.g., weights of a DNN)
- **Algorithm is fully "data-dependent"**

Software functionality parametrized with data-dependent weights can deliver erroneous predictions

Challenges to **define relevant data for testing**:
- Cannot be **exhaustive**
- **Equivalence** classes
- **Operation modes** cannot be enumerated deterministically
- Etc.

Moreover, **independence between training and test data** must be proven

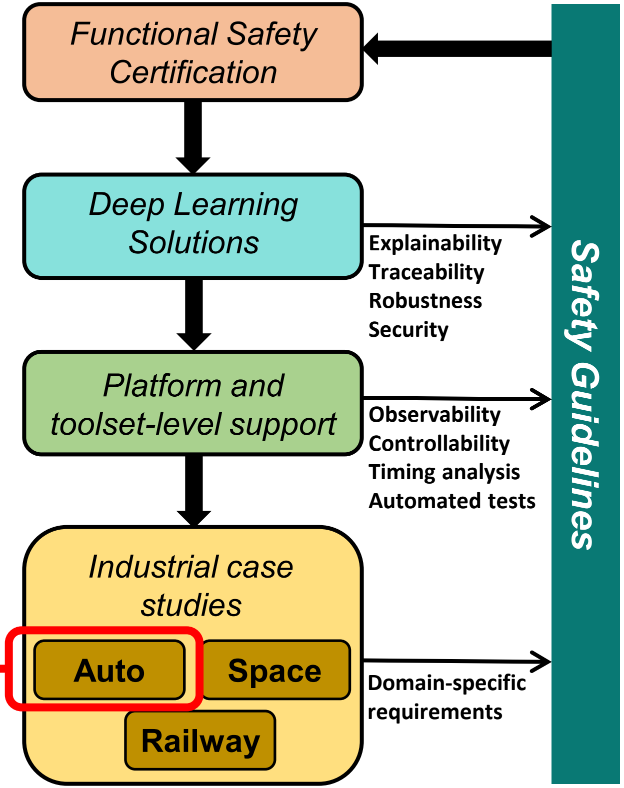
DATA DETERMINES SYSTEM DESIGN

AI (and DL) Specific Challenges

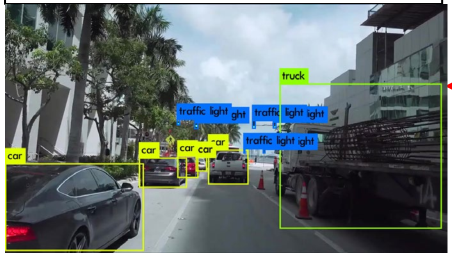
- Current practice in DL frontally clashes with Functional Safety (FUSA)-related processes since:
  - DL software is built as a **combination of**
    - **control** (model configuration such as what layers to use, in which order, etc.) and
    - **data** (algorithm parameters are obtained from training with specific datasets)
      - **stochastic nature**
      - **data-dependent nature**
  - There is a **lack of sufficient explainability and traceability**
    - Why each layer is used and what it does (**semantics**)
    - Why they are deployed in a specific order (**composed semantics**)
    - How safety **requirements can be traced** end-to-end
    - What the scope of application is (e.g. **valid input data range**)
    - What **confidence** can be reached on the predictions obtained (e.g. by detecting occlusions)
  - **Prediction accuracy is stochastic**, and test campaigns deliver, in the best case, success rates linked to specific testing datasets, therefore exposing to **dataset-dependent test conclusions** in many cases

Ambition/objectives

- Ambition: architecting DL solutions **enabling certification/qualification**
  - Making them **explainable** and **traceable**
  - Preserving **high and predictable performance**
  - Tailoring solutions to varying safety requirements by means of **different safety patterns**

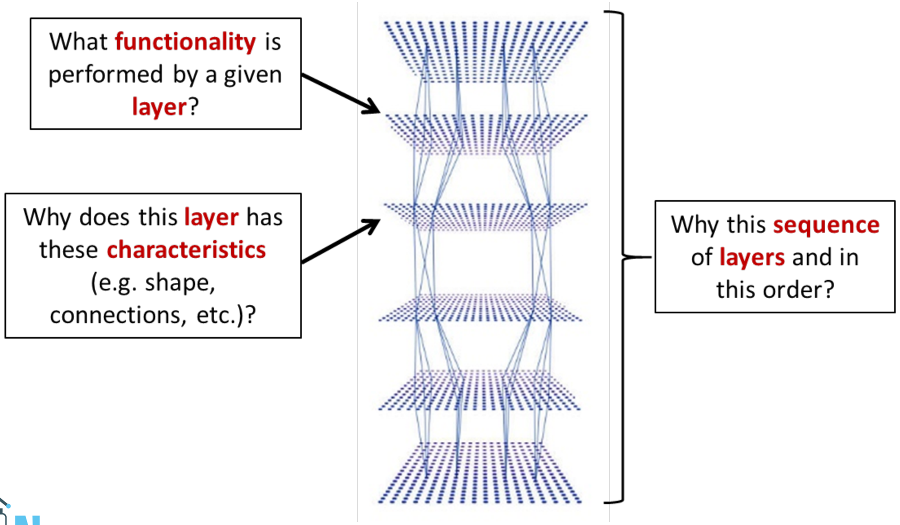


Camera-based object detection (YOLO)



SAFEXPLAIN Goal 1

- Devise new DL components providing explainability and traceability by design
  - Functionally speaking (e.g., a convolution), **software can be developed following the usual process** for automotive systems (i.e., in line with ISO 26262 part 6)
  - **Software architecture** (what layers, what shape), **input data** for training, **training process**, and the **validation test campaign** are the real challenge



Build DNNs so that we can explain
- **What each layer does**
- **Why each layer is as it is**
- **Why layers are arranged this way**
- **How requirements can be traced end-to-end**

SAFEXPLAIN Goal 1 (ctn'd)

- A number of challenges, but some hints on potential approaches to follow
- DL software has "failure rates"
  - This is **not compatible with ISO 26262 for software**
  - But it is **acceptable for hardware** due to random hardware faults
  - Can we extend hardware concept to software?
  - Already foreseen for software timing. We may extend it to software results for DL
- DL software could be assimilated to physical devices
  - Non ASIL-compliant sensors can be used to build some ASIL with proper validation, if their physical principles are diverse(*)

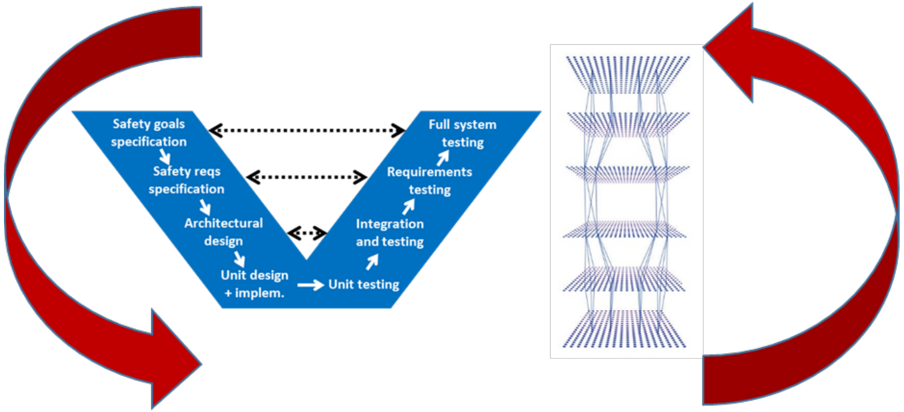
    (*) Further details on this example can be found here: https://doi.org/10.1109/EDCC.2010.34
  - Can we build something similar with **diverse and redundant DNNs**? Where do we have to inject diversity? (training, random inputs, architecture,…)
    - Those are questions to be answered as part of SAFEXPLAIN

I. Agirre, F.J. Cazorla, J. Abella, C. Hernandez, E. Mezzetti, M. Azkarate-Askasua, T. Vardanega, "**Fitting Software Execution-Time Exceedance into a Residual Random Fault in ISO-26262**," in IEEE Transactions on Reliability, vol. 67, no. 3, pp. 1314-1327, Sept. 2018, doi: 10.1109/TR.2018.2828222.

A. Brando, E. Mezzetti, I. Serra, F.J. Cazorla, J. Perez, J. Abella, "**On Neural Networks Redundancy and Diversity for Their Use in Safety-Critical Systems**" in IEEE Computer (special Issue on Trustworthy AI), vol. 56, no. 6, pp.41-50, May 2023, doi: 10.1109/MC.2023.3236523

SAFEXPLAIN Goal 2

- Adapt software safety lifecycle steps and the architecture of solutions based on DL components so that certification is viable
  - E.g., add additional lifecycle steps to contemplate model training, and adapt requirement specification, data management and testing approaches

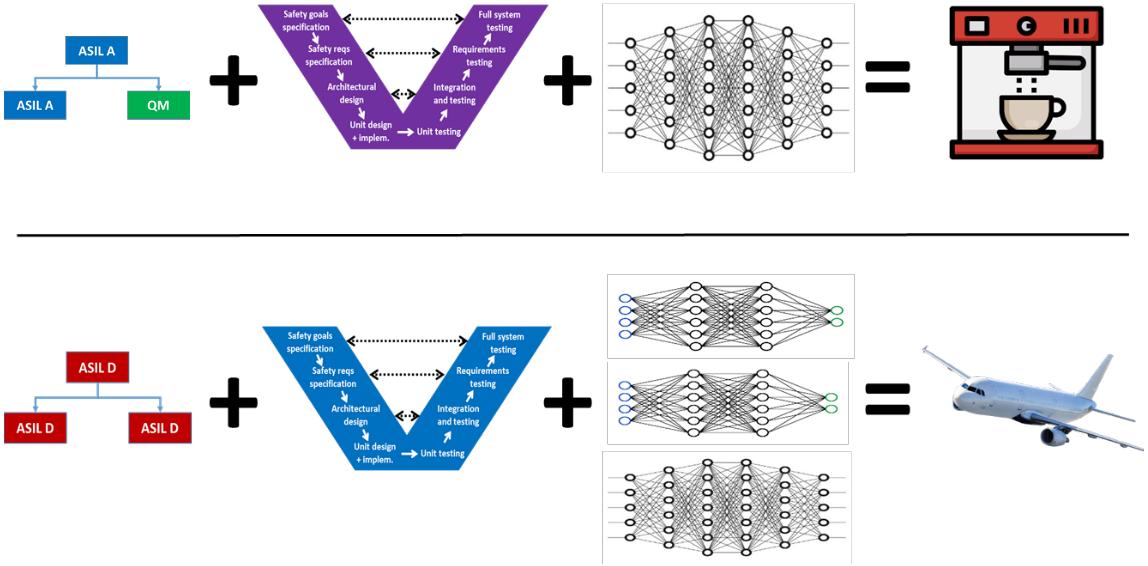


Tailor safety life cycle to enable DNN certification

Tailor DNNs to match properties needed by functional safety standards

SAFEXPLAIN Goal 3

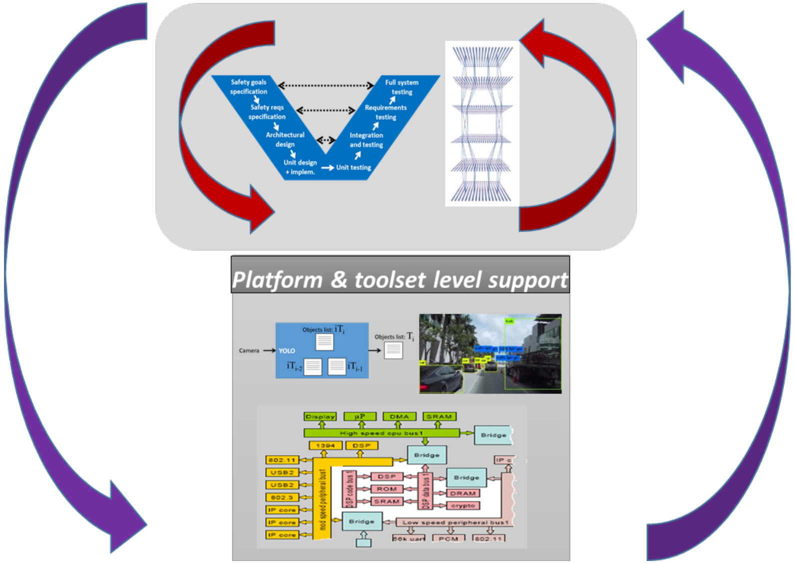
- Provide complementary safety patterns with different safety, cost, and reliability tradeoffs
  - E.g., architecture is different for ASIL-A or ASIL-D, for fail-safe or fail-operational
  - Perhaps a practical example comparable to the "E-gas monitoring concept" would be convenient



Safety patterns include:
- **SIL decomposition**
- **SIL allocation to DL items**
- **Development process**
- **DL architecture**
- **Etc.**

SAFEXPLAIN Goal 4

- Tailor DL architectures to achieve sufficient performance on relevant high-performance platforms
  - Be careful with “performance insufficiencies” in line with SOTIF

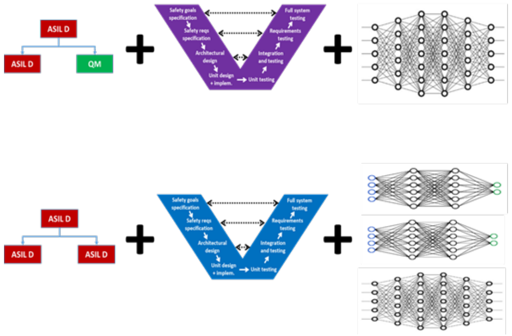


Coordinated actions:
- **Tailor DL architecture to the platform**
- **Keep DL architecture compatible with the safety life cycle**
- **Configure platform to achieve required performance**

SAFEXPLAIN Goal 5

- Demonstrate the long-term viability of the SAFEXPLAIN approach
  - Automotive is the largest target market of the project

SAFEXPLAIN technologies (safety patterns, DL libraries, etc.) → **Integrate in commercial and public tools** → **Assess against mixed-criticality industrial use cases**



- Open source DL libraries
- Commercial toolset integration

Putting it all together \1

- On the FUSA side
  - **Identify patterns** meaningful for AI-based functions
  - Focus on **patterns with varying requirements** (e.g., ASIL-A or ASIL-D, fail-safe or fail-operational, etc.) on AI-based functions
  - Identify **FUSA relevant properties** for DL components and ensembles (e.g., failure rates, diverse redundancy, etc.)

- On the DL side
  - Investigate **DL organizations** that make explainability and traceability emerge by construction while preserving accuracy
  - Investigate **combinations (ensembles) of DL models** that provide FUSA-relevant properties (e.g., diverse redundancy)

Putting it all together \2

- On the platform/tooling side
  - Consider DL solution deployments providing sufficiently **high and stable performance**
  - Iterate with FUSA and DL people to find FUSA patterns and DL solutions that can be run efficiently
  - Devise ways to (automatically or semi-automatically) **provide FUSA-relevant evidence** based on DL-based results using appropriate tools

- On the case study side
  - Consider **varying FUSA requirements** for different AI-based components
    - Within a single use case
    - Across different use cases
  - Consider heterogeneous requirements across use cases (e.g., **varying degrees of performance, accuracy**, etc.)

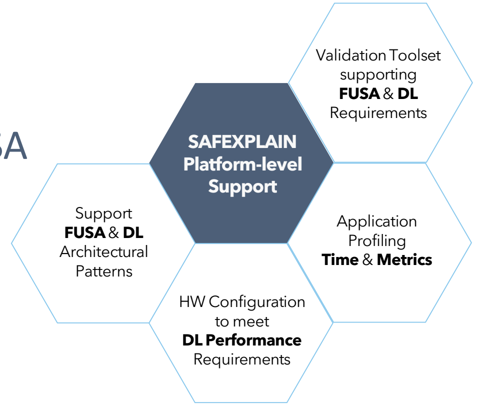
Conclusions

- AI needed to realize autonomous systems
- But **AI challenges common practice for FUSA-related software**
  - Failure rates, data used for software design, etc.
- SAFEXPLAIN goals
  - Make **DL components explainable and traceable** by design
    - DL components built with FUSA in mind
  - **Adapt FUSA standards** to allow certifying DL software
    - Make standards amenable to intrinsic DL characteristics (e.g., failure rates, data used for design)
  - **Preserve sufficiently high levels of performance** to meet safety goals (e.g., 25 FPS)
- **Do not consider each part on its own, but keep a continuous dialogue among DL, FUSA and platform experts, along with end users to make all pieces fit together**

Focus on SAFEXPLAIN Platform

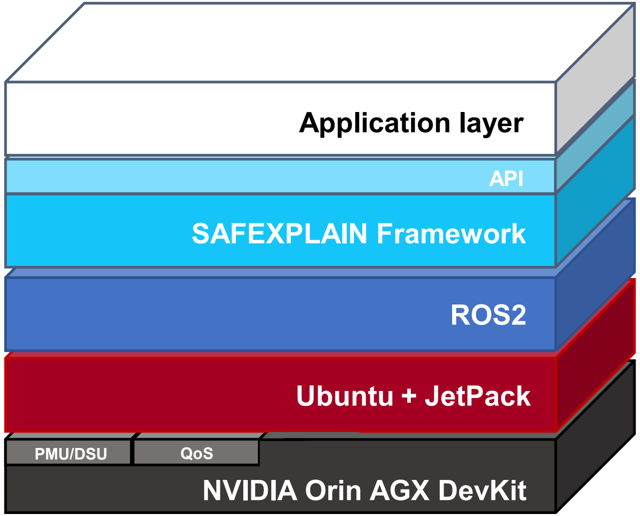
SAFEXPLAIN Platform drivers

- **Support SAFEXPLAIN FUSA & DL patterns**
  - Deploy necessary HW/SW support to map identified FUSA patterns to concrete platform
- **Guarantee DL performance requirements**
  - At the same time exploit computational power of selected target platform
- **Tailor an industrial-quality validation toolset**
  - Support monitoring and test reproducibility/automation
- **Provide timing characterization of DL functions**
  - Profiling of execution time and relevant metrics
  - Deploy statistical methods for timing predictions



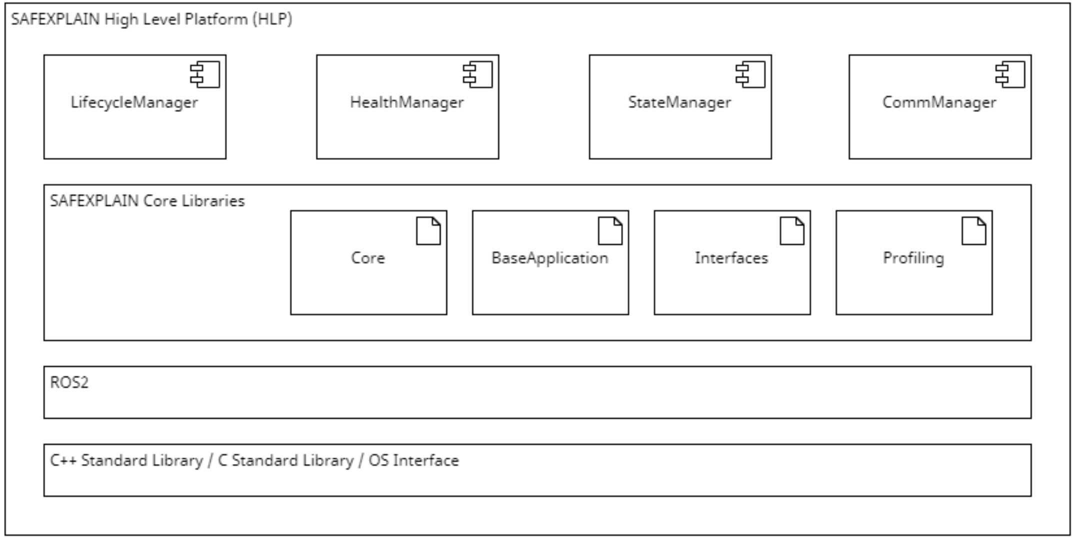
SAFEXPLAIN framework

- **Deep reusable SW stack**
  - Inheriting Ubuntu and JetPack libraries
  - Selected ROS-2 as standardized layer
    - Middleware, libraries, communication
    - Client interface for users' application
    - Users define *nodes* and *data flow*
  - Make ROS-2 transparent to SAFEXPLAIN applications
    - Wrapper API for users' applications
    - The API implements the toolset functionalities with minimal configuration overhead



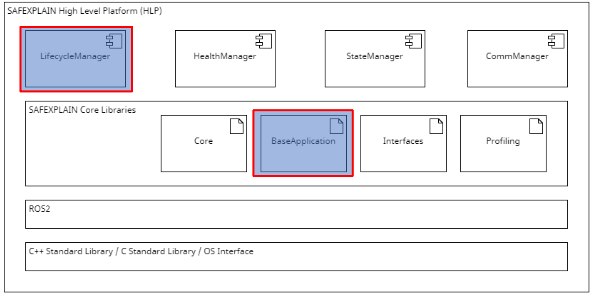
SAFEXPLAIN Platform Framework Overview

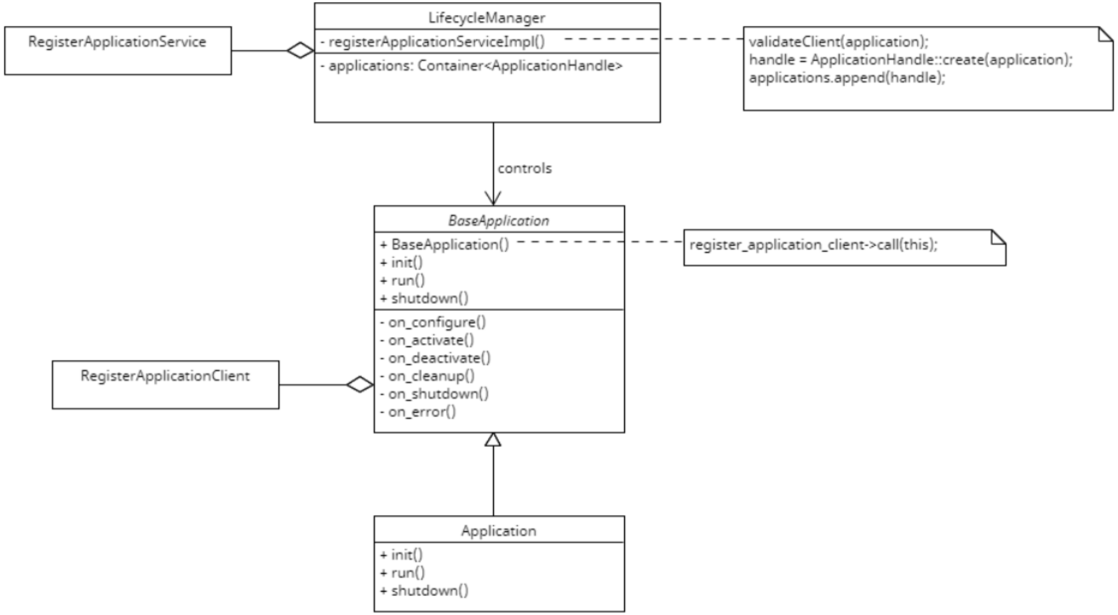
- **The main goals are:**
  - To build observability channels, facilities for testing and monitoring
  - To centralize control of the platform resources
  - To bridge the gap between the application layer and the Low Level Platform
- The HLP design is inspired from the AUTOSAR Adaptive standard

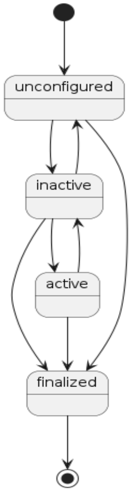


Example: Lifecycle Management

- The *LifecycleManager* component is responsible for initialization, configuration, and termination of platform applications.

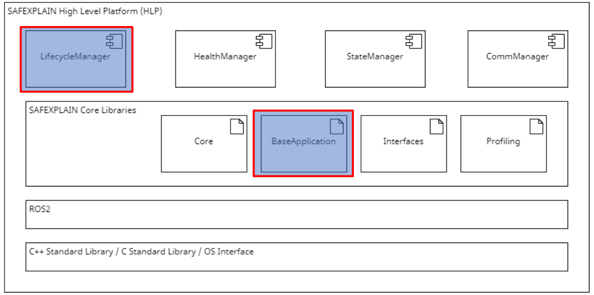




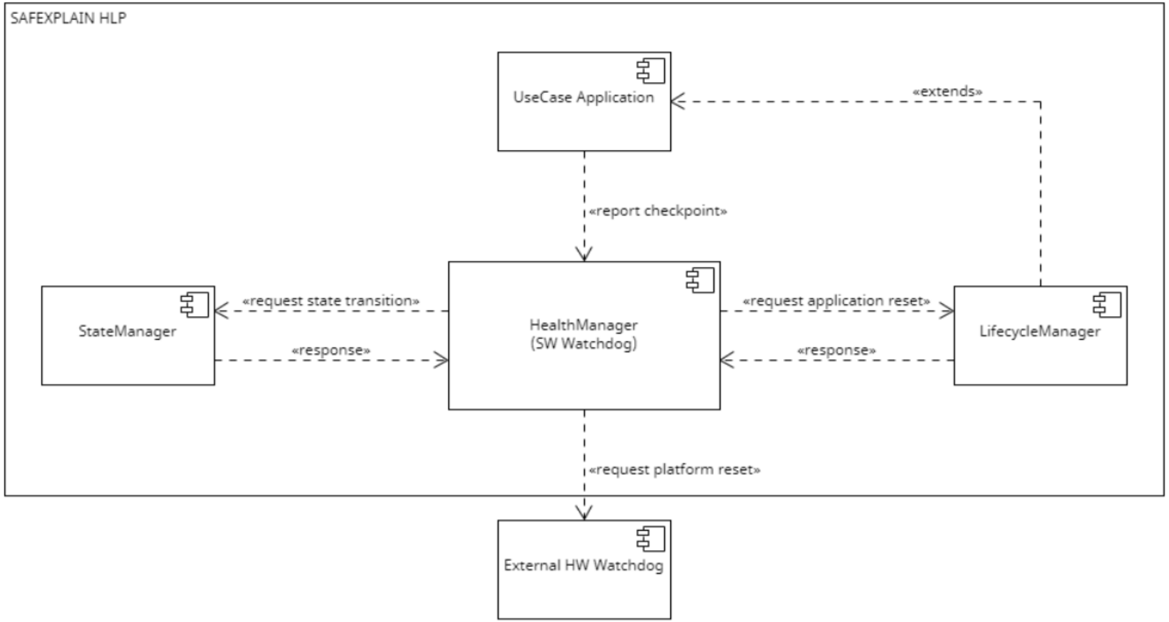


Application (internal) states

Example: Lifecycle Management



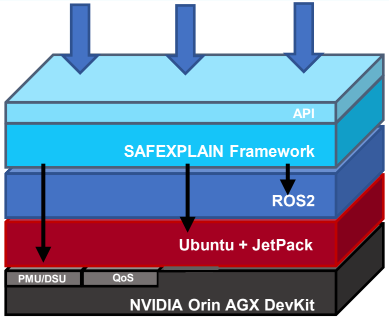
- Offers a possible reaction path to unexpected events.
  - Events will be defined as part of the monitoring concept and implemented by the *HealthManager*.



SAFEXPLAIN HW profiling solution

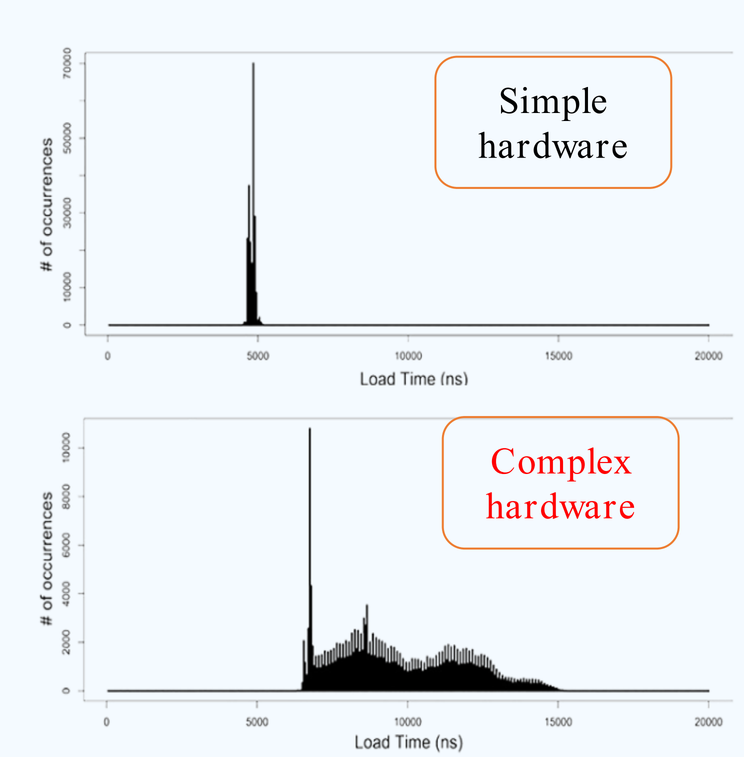
- **Observability support**
    - Collect timing information and relevant HW events
        - Cache statistics, HW resource usage, etc.
    - *CPU Clusters*
        - Standard support available in A78 cores- PMUv3 (
            - Accessible via standard tools or memory mapped PMCs
        - Also, Coresight (v3) and Embedded Trace Macrocell (v4.2)
    - *GPU Cluster*
        - No open support for monitors
        - Wrapping or integrate with NVIDIA proprietary Nsight tools
- **SAFEXPLAIN application interface**
    - Profiling API can be:
        - Implicitly attached to a *node* or
        - Explicitly invoked from within the *node*
    - Minimal API requirements:
        - *init() run() shutdown()*
        - Each may implicitly call the profiling API
    - Extended API for profiling:
        - *init_perf() configure_perf() start_perf() stop_perf()*
    - API will transparently access and configure the right layer
        - HW PMU, Linux tools, ROS2 library
    - Information is saved to text device and retrieved for offline processing

```
class App : BaseApplication {
    void init() {
        ... // App initialization directives
        init_perf();
        configure_perf(config);
    }
    void run() {
        start_perf();
        ... // Work to be profiled
        stop_perf();
        ... // Other work
    }
    void shutdown() { ... }
};
```



Probabilistic Timing Analysis

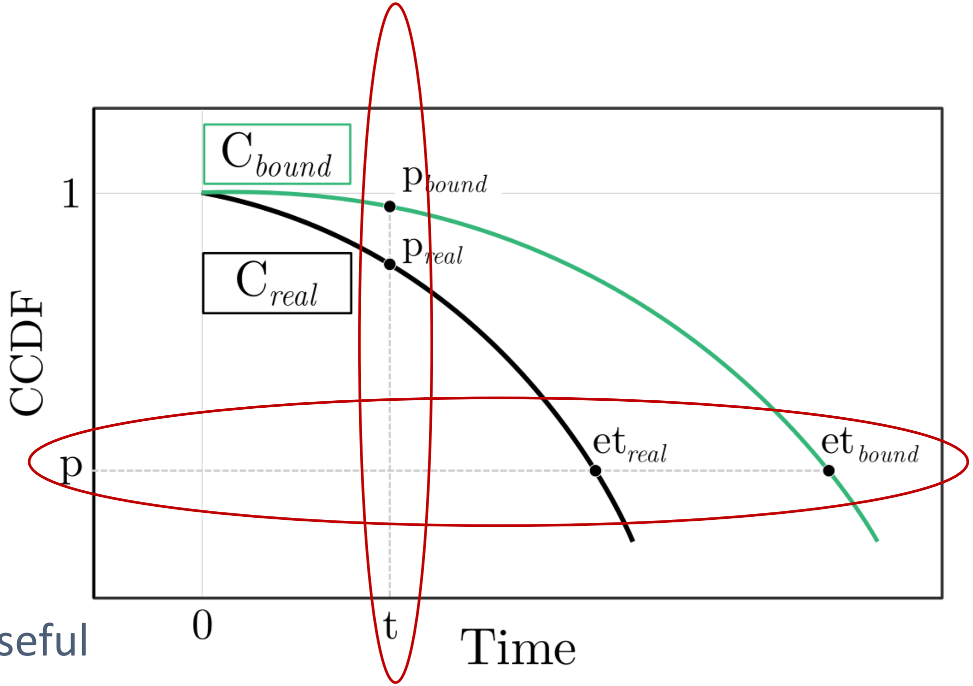
- **Probabilistic Timing Analysis (PTA)**
  - Increasingly and successfully deployed for deriving trustworthy and tight estimates of software timing
  - Especially for *Measurement-Based variant* (**MBPTA**)
- MBPTA helps dealing with the **increased complexity** of hardware and software in real-time systems
  - From micro-controllers to MPSoCs
  - From simple control SW to AI-based software
- Increased complexity causes
  - Variable timing behavior
  - Unobvious dispersion (multi-modal distribution)



Source: Lynx Software

MBPTA

- Produces a **probabilistic WCET** (pWCET) estimate
- The CCDF denotes the probability of exceeding a certain *execution time* value (*et*)
- The pWCET required properties
  - ✗ **Optimistic**:
    - ✗ $p_{bound} < p_{real}$
    - ✗ $et_{bound} < et_{real}$
  - ✓ **Conservative**:
    - ✓ $p_{bound} \geq p_{real}$
    - ✓ $et_{bound} \geq et_{real}$



- Exceedingly pessimistic pWCET are not useful
- pWCET estimates should **tightly** model the real distribution

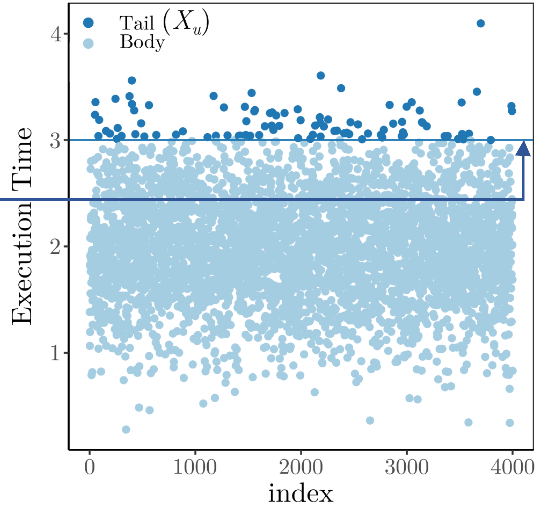
Extreme Value Theory (EVT)

- EVT provides two fundamental **theorems for the distribution of extremes (tails)**
  - The excess random variable is the variable X from a **threshold u** onward
  - The excess distribution function is the distribution from a threshold u onward
- It converges in probability to the Generalised Pareto Distribution (GPD)

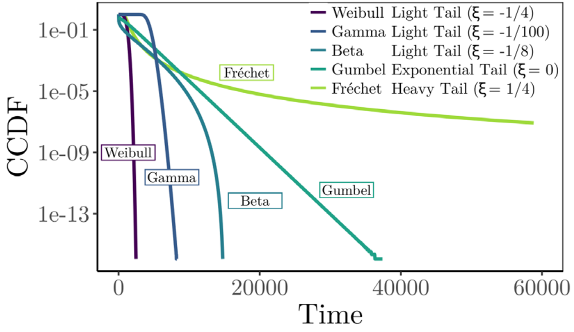
$$F_u(y) = \frac{F(u+y) - F(u)}{1 - F(u)}, \quad y \geq 0$$

$$F_u \xrightarrow{\mathcal{L}} G(y;\xi,\sigma) \quad as \quad u \to \infty$$

$$G(y;\sigma,\xi) = \begin{cases} 1 - \left(1 + \frac{\xi y}{\sigma}\right)^{-\frac{1}{\xi}} & if \quad \xi \neq 0 \\ 1 - \exp\left(-\frac{y}{\sigma}\right) & if \quad \xi = 0 \end{cases}$$



- The extreme value index ξ determines the shape of the tail
  - Because programs must finish, they are modelled as light tails
  - The good model is GPD or other distributions with ξ < 0
  - A generally safer but possibly pessimistic model is the exponential (ξ = 0)



SAFEXPLAIN

Safe and Explainable
Critical Embedded Systems based on AI

Follow us on social media:

www.safexplain.eu

Funded by
the European Union

This project has received funding from the European Union's Horizon Europe programme under grant agreement number 101069595.