



NVIDIA Approach for Achieving ASIL-B Qualified Linux

Automotive Linux Summit | Dec. 8, 2025



Agenda

- Intro & background

- Why roll your own

- NVIDIA's concept

- What's next?

About Me

Not a safety expert ... but I work with many who are.

Not the creator of this concept ... but I work with those who did.

Tend to speak using automotive terminology ... but this concept is not limited to automotive.

Additional Context



Address Space Isolation for Enhanced Safety of the Linux Kernel

- September 2022



Identifying Safety Weaknesses and Fault Propagation in the Linux Kernel

- August 2025



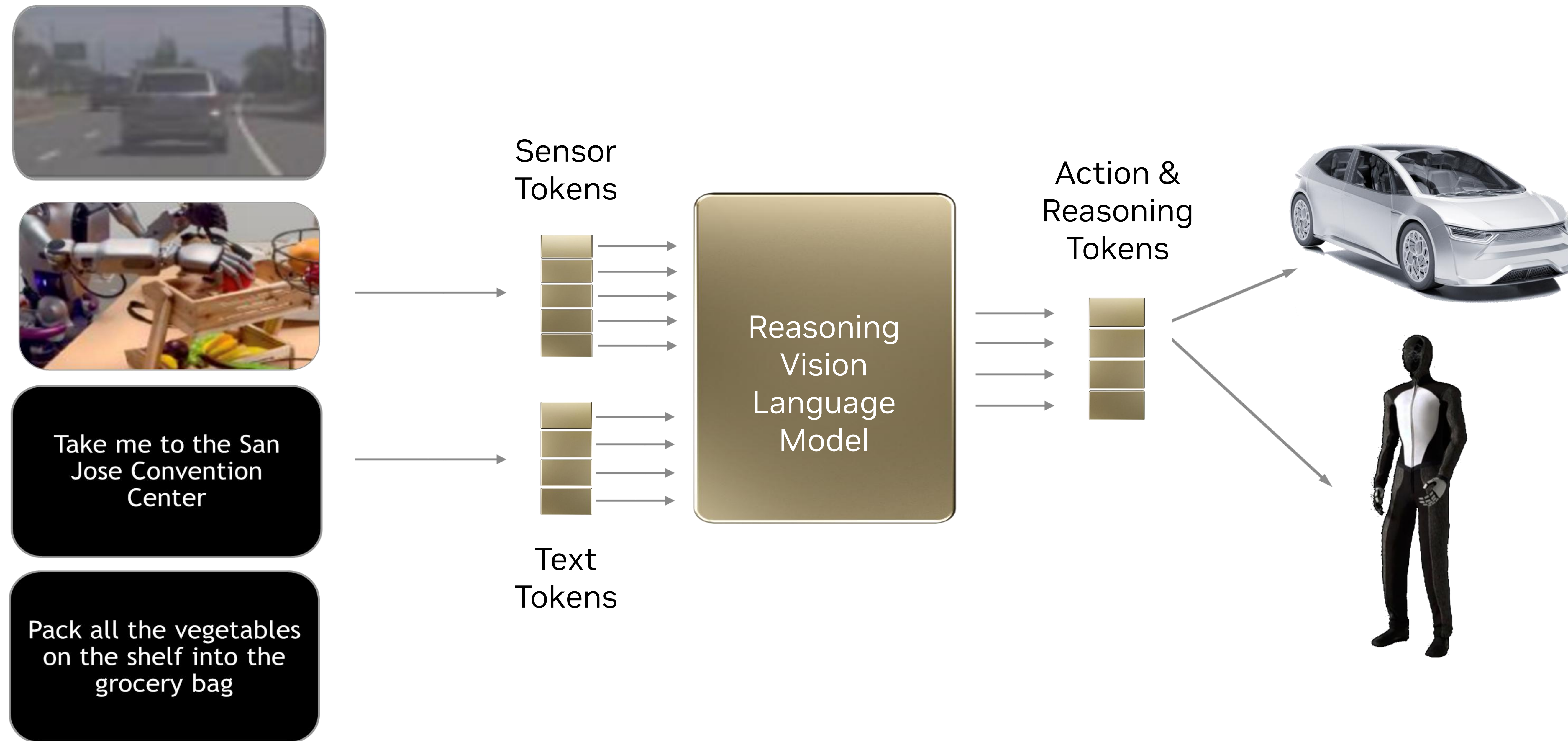
NVIDIA Approach for Achieving ASIL B Qualified Linux: minimizing expectations from upstream kernel processes

- Dec 12, 2025, 10:10 AM
- "Hall B4 (63)" (Toranomon Hills Mori Tower)

Key Takeaways

- Safety is hard – effort, time, cost, etc.
- Safety is contextual – can vary per product, manufacturer, application, industry, etc.
- We need to protect the system from Linux – opposite of things like security
- Lots of approaches for correctness, challenges for completeness

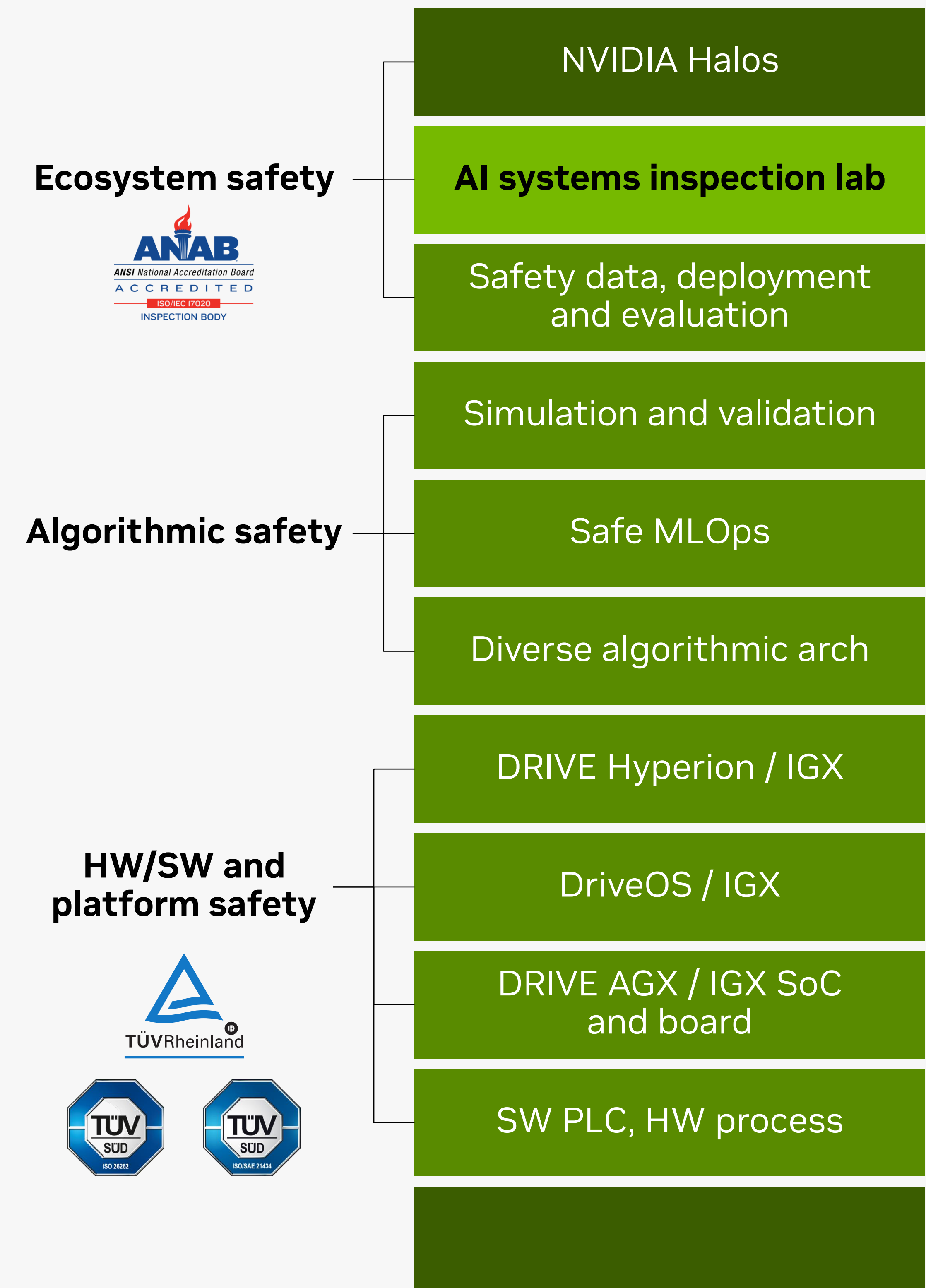
The New Era of Physical AI



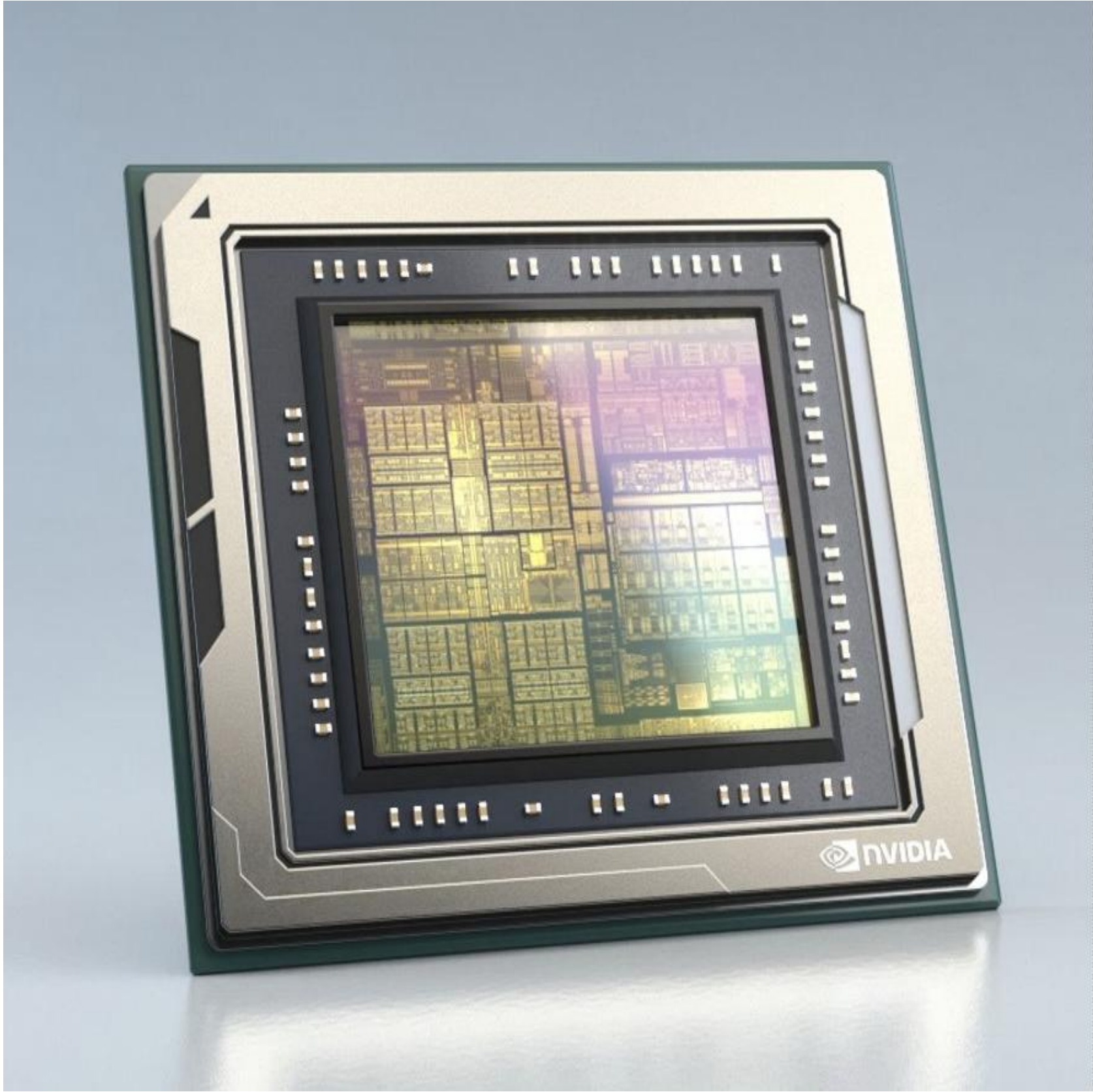
NVIDIA's Halos for Physical AI Safety

A comprehensive safety system for autonomous vehicles and the era of physical AI

- Halos is a **full-stack** comprehensive safety system for Autonomous Machines that unifies safety elements from vehicle and robot architecture to AI models
- It comprises HW and SW elements, tools, models, and the design principles for combining them to safeguard **AI-based, end-to-end AV and robotics stacks**
- The **NVIDIA AI Systems Inspection Lab** is the NVIDIA Halos entry point of **Halos Certification Program**



Key Elements of NVIDIA Halos' AV Safety Platform

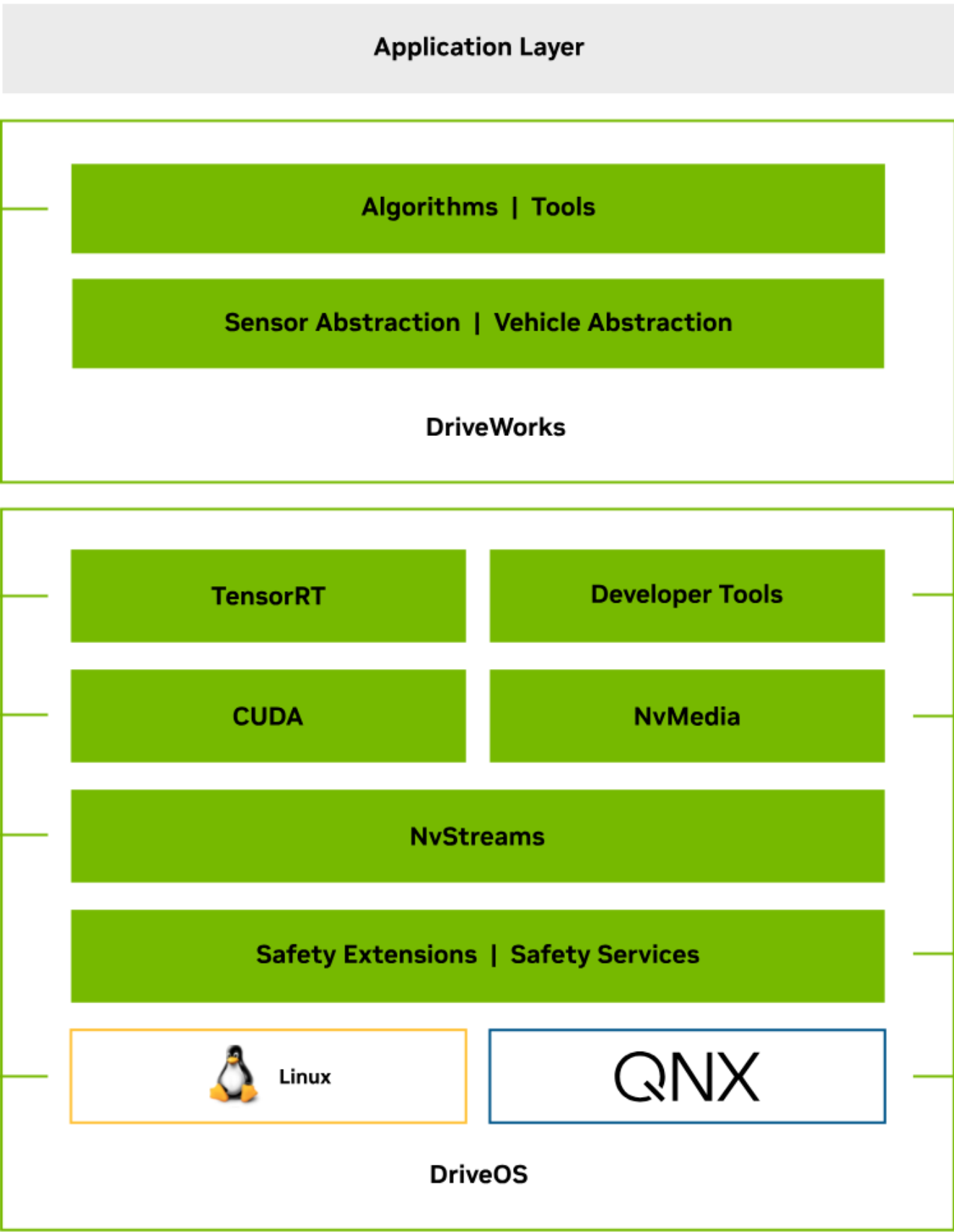


21B

Transistors safety assessed

HW Safety SoC & Reference Board

TüV Safety Assessed
ASIL D Systematics
>ASIL B Random Fault Metrics

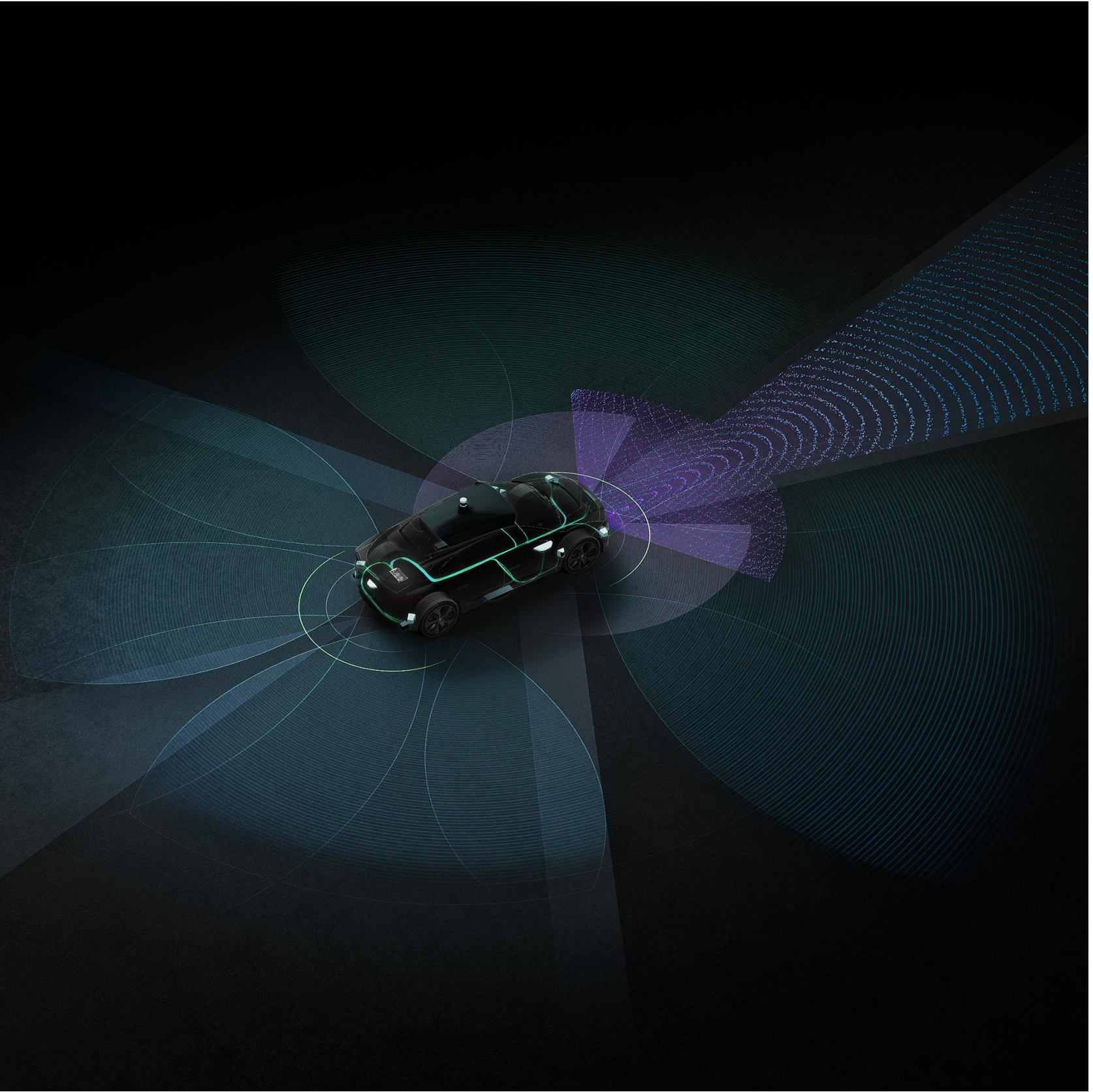


7M

Lines of safety assessed code

SW Safety

TüV Certified DriveOS up to ASIL D



22K+

Platform safety monitors

Platform Safety

TüV Safety Assessed base platform
Sensor Drivers & E2E Authentication

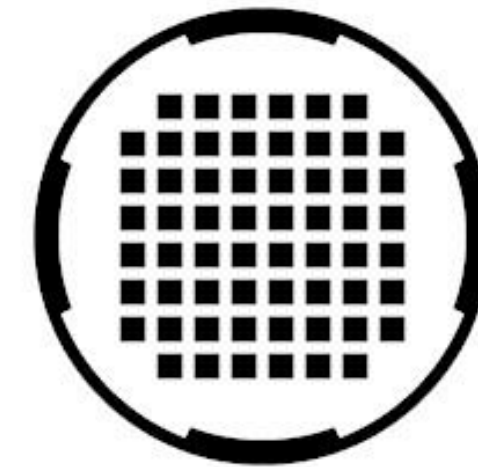
What Makes NVIDIA Halos Unique?

AV Safety Leadership from Research to Engineering



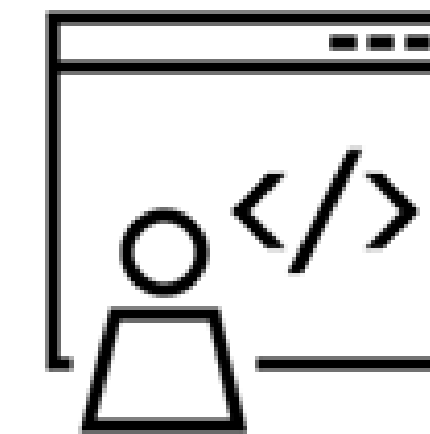
15,000+

Engineering years invested in vehicle safety to date



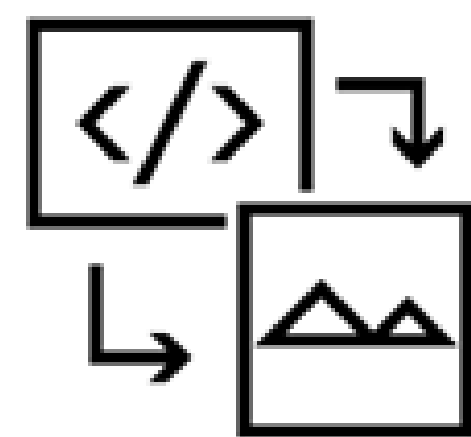
21 Billion+

Safety transistors safety assessed



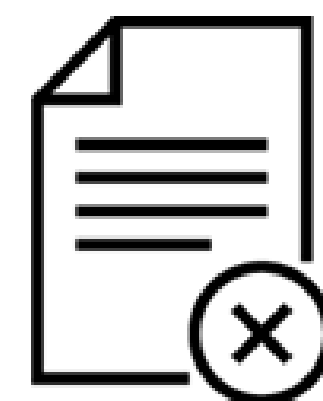
7,000,000

Lines of safety-assessed code



2,000,000

Daily end-to-end integration tests for validation



22,000+

Platform safety monitors



20,000+

Hours of safety test data



1,000+

Patents filed



240+

Research papers published on AV safety



30+

Certificates and assessment reports issued

“

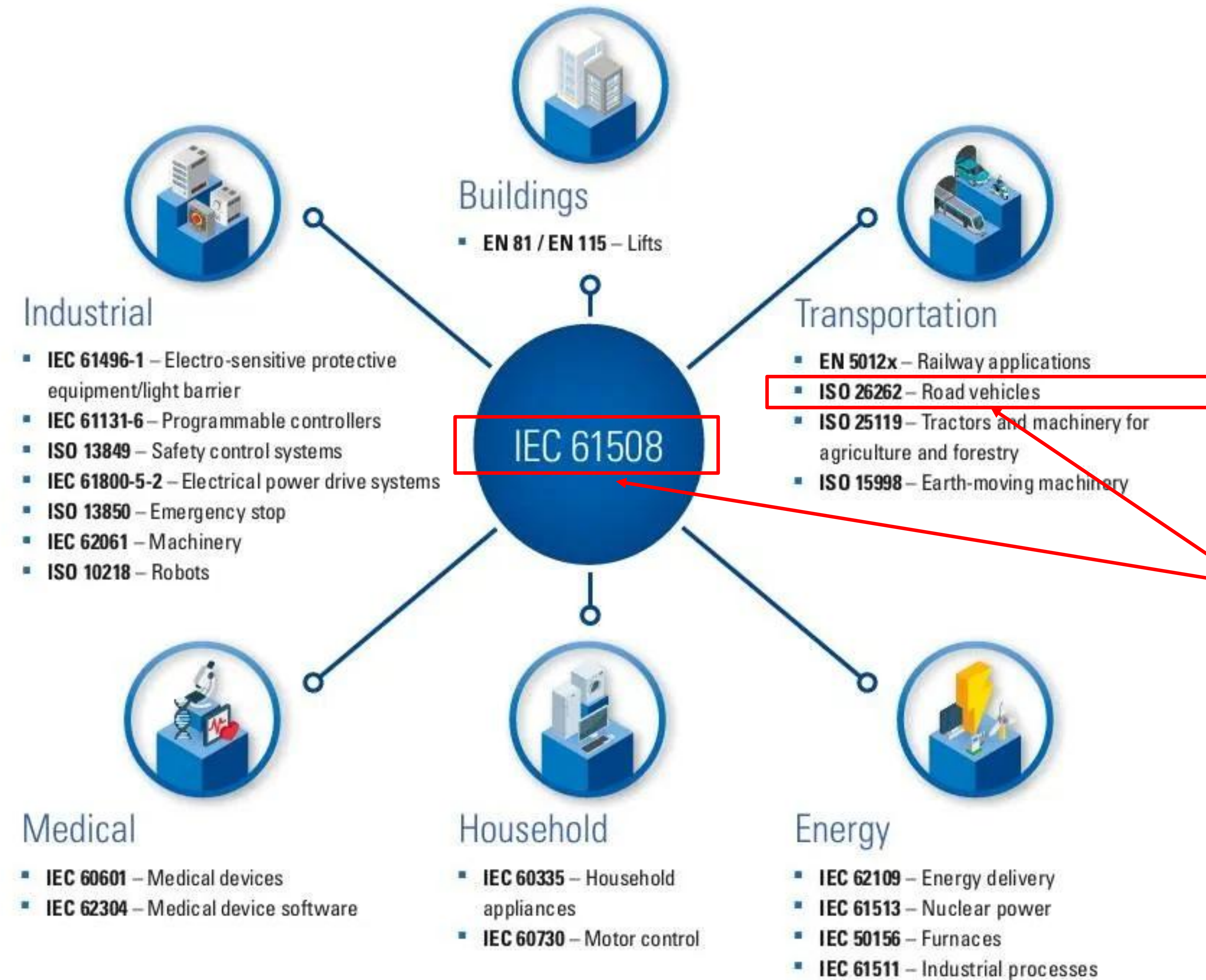
Functional safety is the absence of unreasonable risk due to hazards caused by malfunctioning behavior of electrical/electronic systems.

— *ISO 26262-1:2018, Clause 3.67*

”

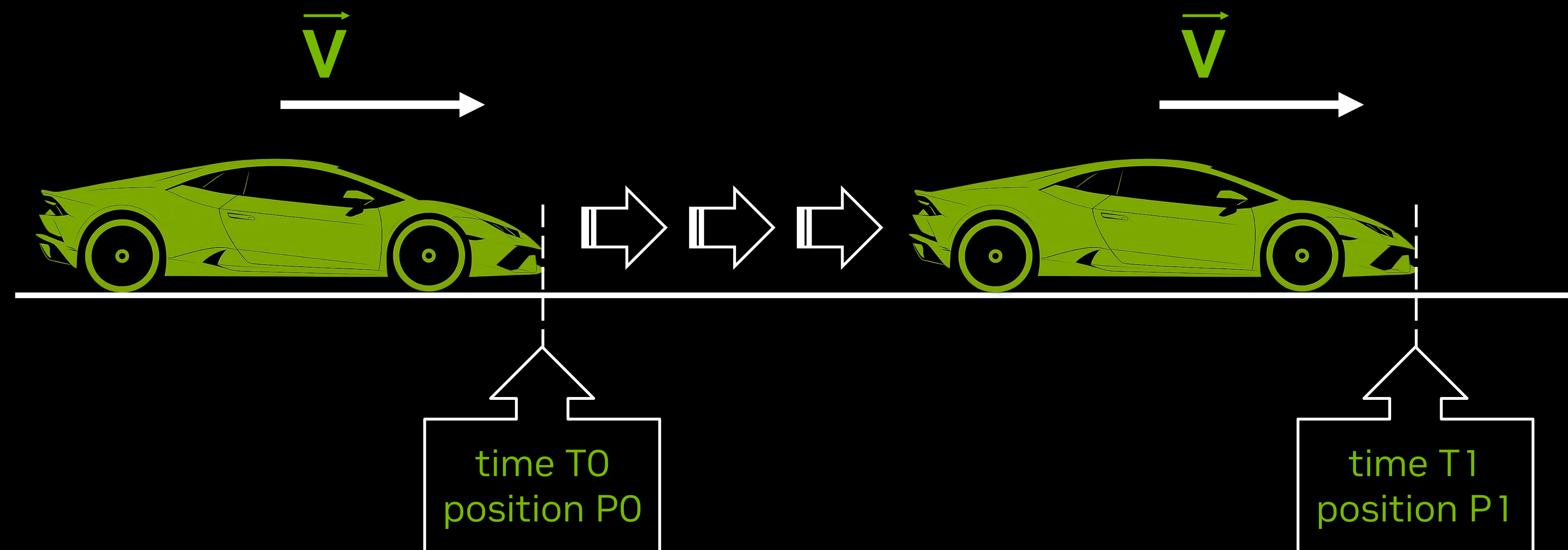
Different Standards For Different Industries

<https://www.tuvsud.com/en-us/services/functional-safety/about>



Most work I've seen are in these areas

Safety Requirements Derive From Physical Constraints



$$V = \frac{P1 - P0}{T1 - T0} = \frac{\Delta P}{\Delta T}$$

$$\Delta T = \frac{\Delta P}{V}$$

$$V = 100 \text{ km/h}$$

$$\Delta P = 1 \text{ m}$$

$$\Delta T = 36 \text{ ms}$$

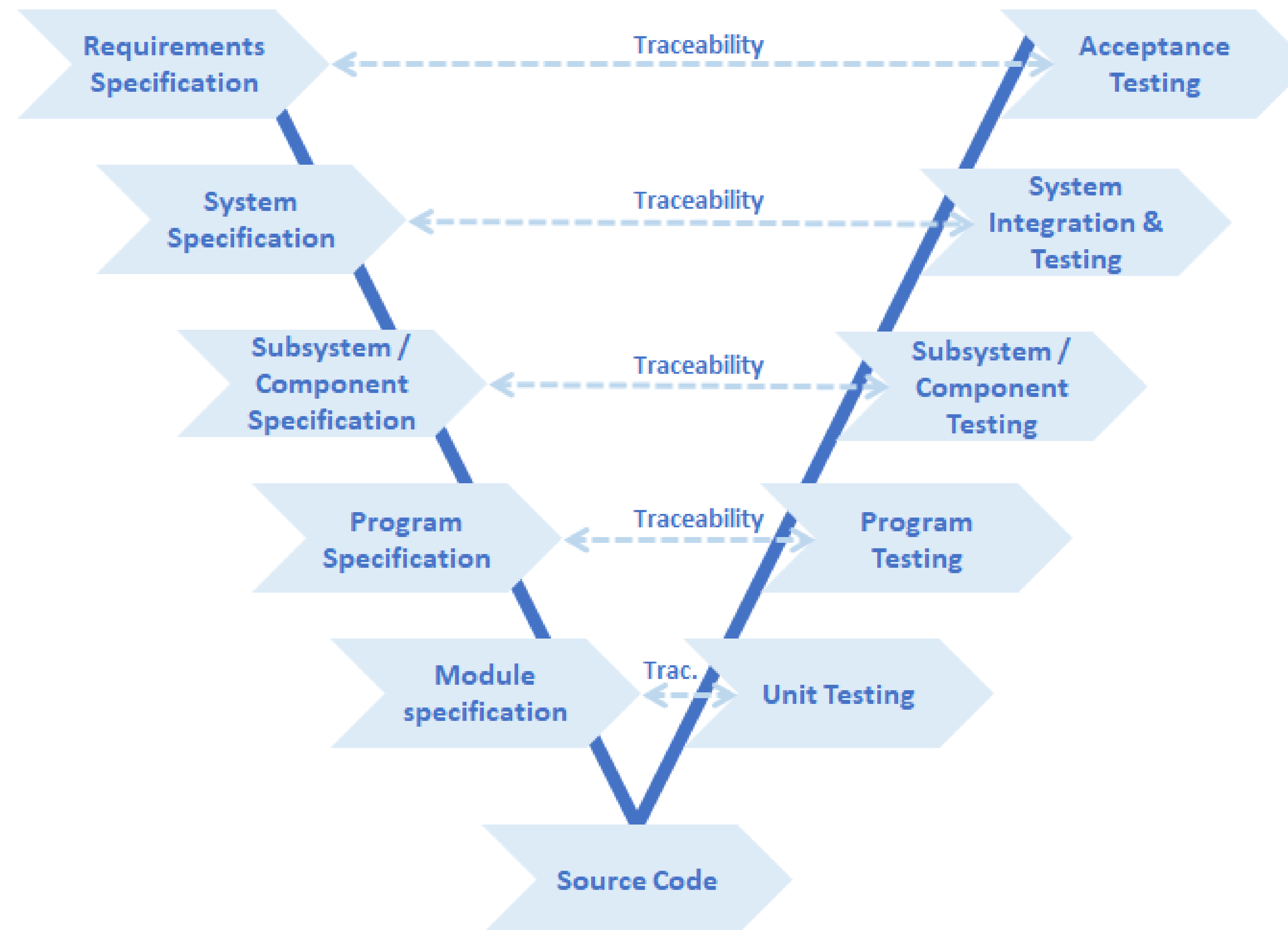
Why Is Functional Safety Challenging For Linux?

Development Process	Architecture & Design	Size & Velocity*
No formal architecture, requirements, design	Monolithic architecture and address space	36,780,000 lines across 81,500 files (as of v6.5.0) <ul style="list-style-type: none">• typically, only 5%-10% used
Heavily dependent on tree of maintainers for review and maintenance	Limited fault isolation	~9 changes per hour → ~78k changes per year
Not developed with safety as a goal	Every component runs with equal privileges	
	Optimized for performance	

All of these translate to increased effort over traditional safety-focused SW.

*<https://github.com/gregkh/kernel-development/blob/master/kernel-development-how-and-security.pdf>

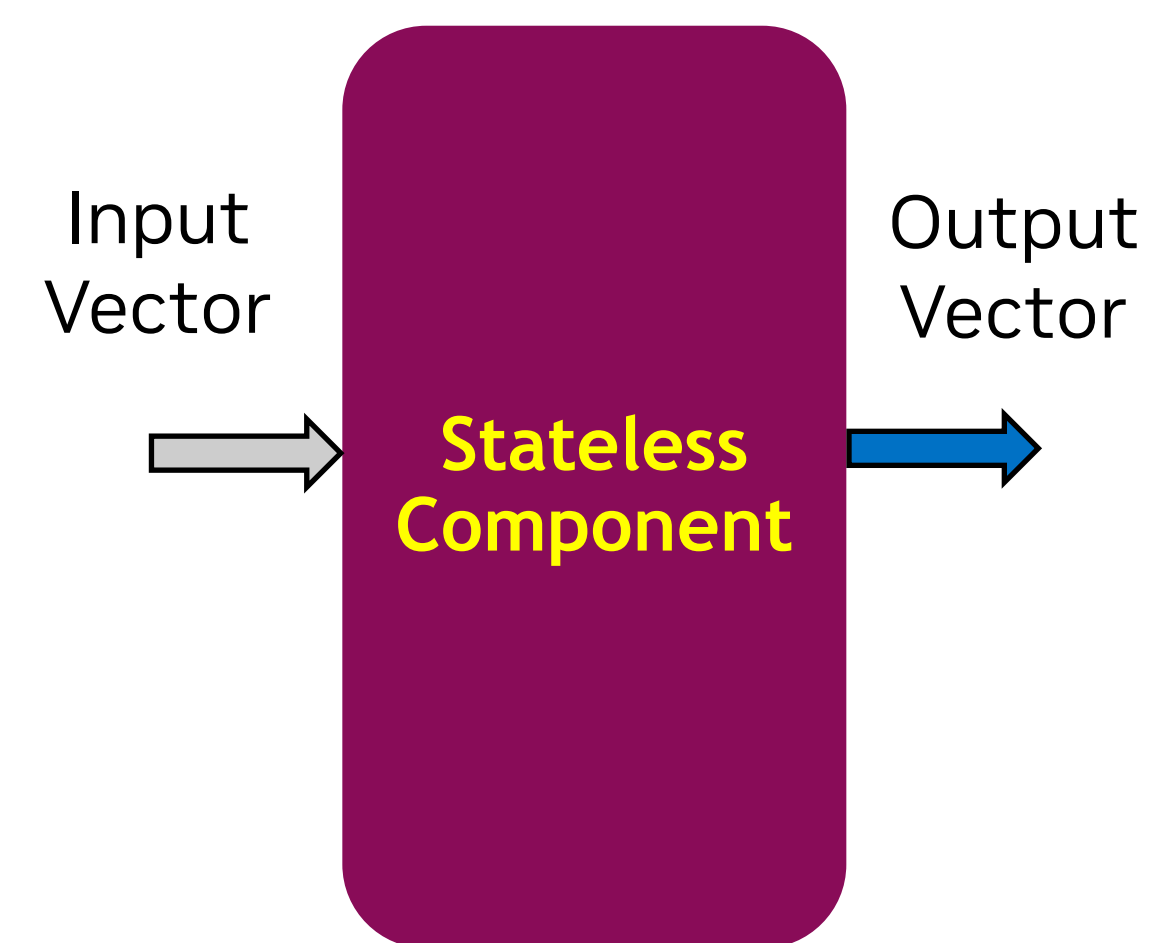
Traditional V Model



Component Complexity vs. Qualification Accuracy

As complexity increases, qualification accuracy decreases

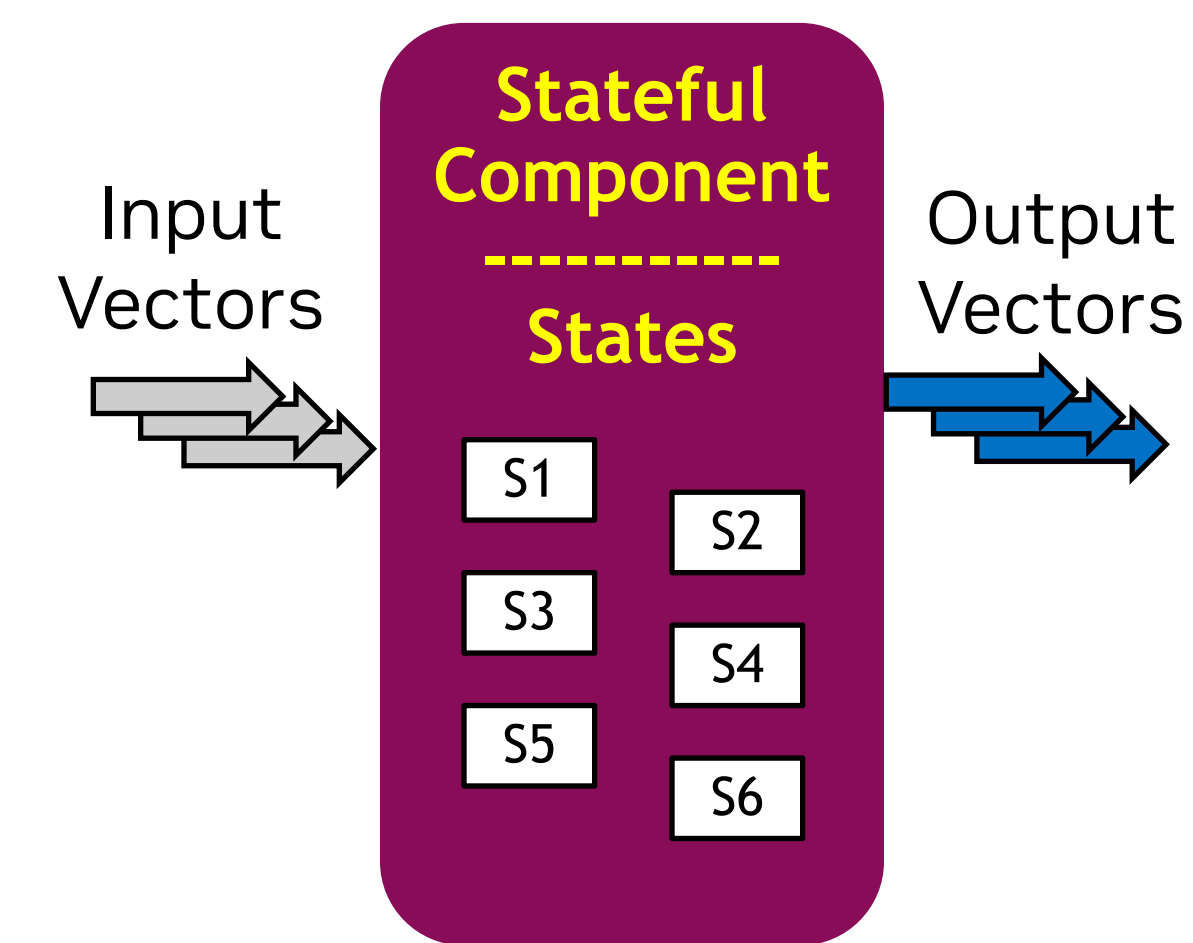
High Confidence



Low complexity → High accuracy

Examples – crypto algorithms, memset, memcpy

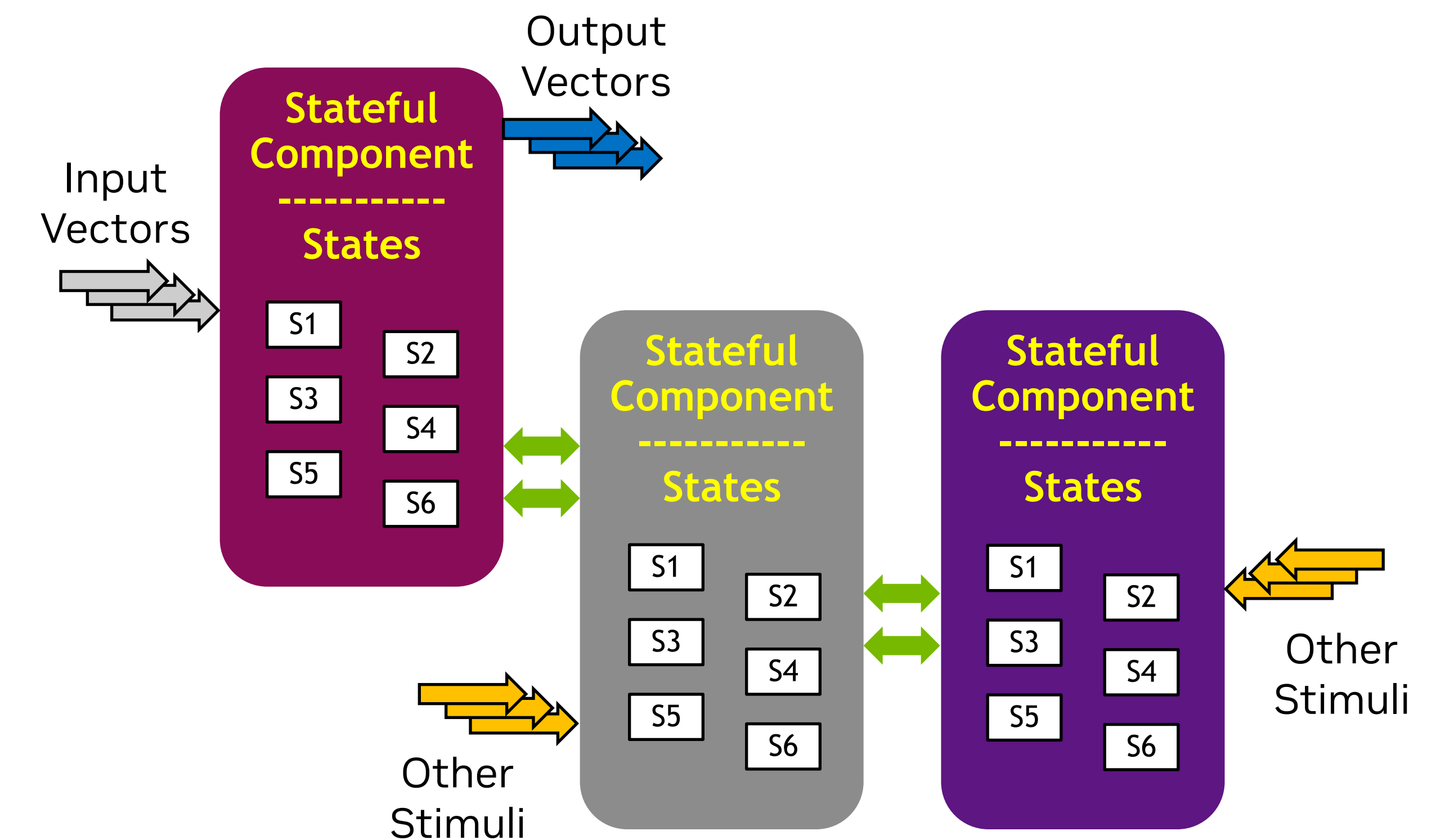
Borderline



Medium complexity → Medium accuracy

Examples – Simple security modules (Apparmor)

Low Confidence



High complexity → Low accuracy

Examples – Memory, Scheduling, Networking, cgroups

High Effort & Cost Using Traditional Methods

Approximations (YMMV)

1,500

Lines per engineer per year

1.8M

Lines to qualify for Linux Kernel (5% of v6.5)

1,226

Engineers required to complete in 1 year

\$300M

Cost per ASIL-B qualification

Common Solution Patterns We've Seen

Many efforts over many years

“Dive In”	“Don’t Touch It”	“Zoom Out”
Safety Element Out of Context (SEOOc)	Safety Element Out of Context (SEOOc)	In-context
Minimize scope to qualify using traditional methods	Heavy dependence on external monitoring	Analyze the system as a whole
Tailor qualification process	Intimate knowledge and coupling with external monitors	Challenging to scale and replicate
Still very labor-intensive		

All have tradeoffs in functionality, latency, scalability.

The Linux Kernel Spatial Interference Problem

Code running in kernel mode

can write to anything in the kernel address space

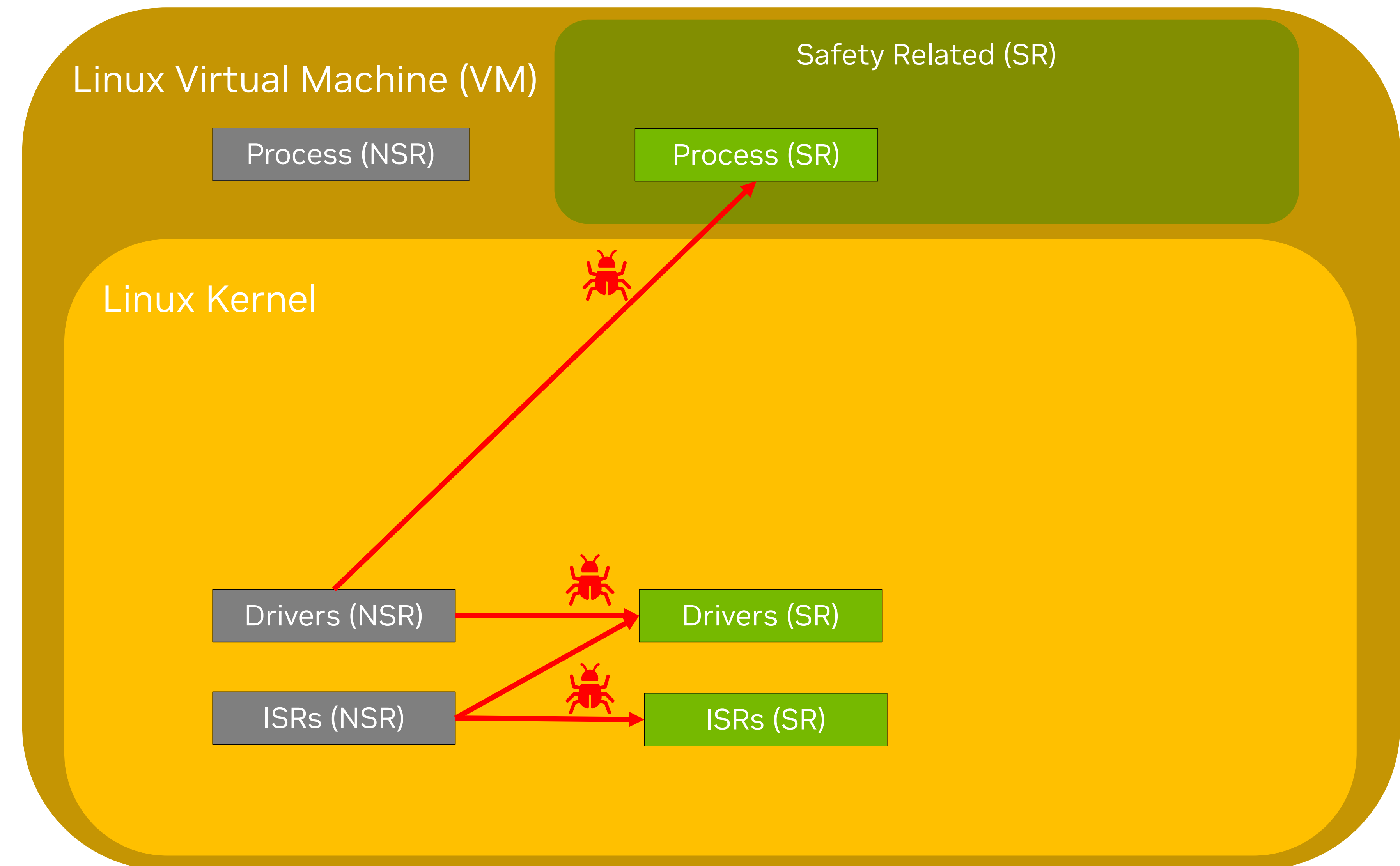
that is not write protected there, including:

- **ALL kernel data (e.g. drivers) and metadata (e.g. stacks)**
- **ALL processes memory, code and constants too.**
- **ALL memory mapped peripherals (e.g. interrupt controller)**

Spatial Interference Paths From Linux Kernel

Monolithic Linux kernel has broad access to system resources

- Freedom From Interference (FFI) should fully isolate SR & NSR
- Sources of interference in NSR Linux Kernel (bugs) have the ability to interfere with SR code
- Analysis and testing on their own are insufficient for achieving strict ASIL integrity and availability requirements



Solution To The Linux Kernel Spatial Interference Problem

1. **Create additional contexts, one for each ASIL**
 - An ASIL context can write only to same-or-lower ASIL data
2. **Qualified code can write only to same-or-lower ASIL data**
 - Threads switch context independently from each other
 - Threads can affect only other ones of same-or-lower ASIL
3. **Processes receive memory aligned with their ASIL**
4. **Peripherals are protected in a similar fashion**
5. **Access to mixed criticality resources is gated and vetted**

MMU Enforcement For Context Protection

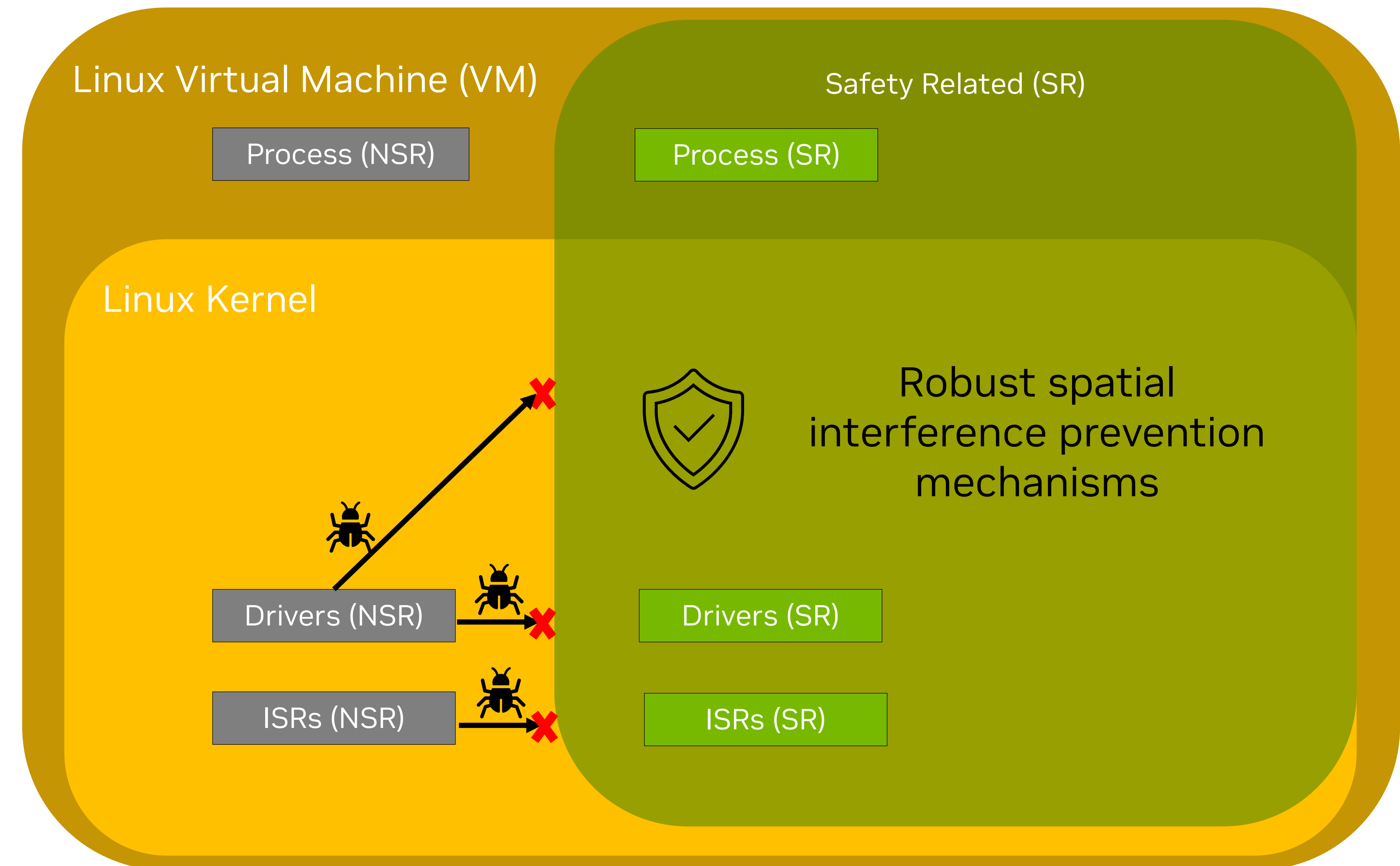
Userspace & Linux Kernel

From \ To	QM	ASIL-A	ASIL-B	ASIL-C	ASIL-D
QM	READ: Yes WRITE: Yes	READ: Yes WRITE: No	READ: Yes WRITE: No	READ: Yes WRITE: No	READ: Yes WRITE: No
ASIL-A	READ: Yes WRITE: Yes	READ: Yes WRITE: Yes	READ: Yes WRITE: No	READ: Yes WRITE: No	READ: Yes WRITE: No
ASIL-B	READ: Yes WRITE: Yes	READ: Yes WRITE: Yes	READ: Yes WRITE: Yes	READ: Yes WRITE: No	READ: Yes WRITE: No
ASIL-C	READ: Yes WRITE: Yes	READ: Yes WRITE: Yes	READ: Yes WRITE: Yes	READ: Yes WRITE: Yes	READ: Yes WRITE: No
ASIL-D	READ: Yes WRITE: Yes	READ: Yes WRITE: Yes	READ: Yes WRITE: Yes	READ: Yes WRITE: Yes	READ: Yes WRITE: Yes

Robust Spatial Interference Prevention Mechanisms Are Needed

Architecture Overview

- Fully isolate SR code from NSR code at all levels of the Linux stack
- New mechanisms to prevent, not just detect, entire classes of spatial interference from Linux Kernel
- Scalable to the highest levels of ASIL integrity and availability
- No change to NSR code! Safety opt-in only where needed.



The Linux Kernel Temporal Interference Problem

Code running in kernel mode

can write to anything in the kernel address space

that is not write protected there, including:

- **scheduler metadata**, disrupting expected execution timing
- **Interrupts configuration**, preventing asynchronous events

Solution To The Linux Kernel Temporal Interference Problem

Create independent monitors

- SW-defined, multi-context, challenge-response watchdog
- Per-safe-context opt-in process-watchdog
- Per-safe-context opt-in in-process thread-watchdog
- In-kernel thread-watchdog
- In-kernel interrupt-watchdog: from ISR to bottom-half

Applying This Concept

Code Changes

- Add a safety dimension to memory allocation
- Add context switching to functions that have safety requirements

Performance

- Increase in memory utilization due to additional safety pools
- Increase in CPU utilization due to additional context switching

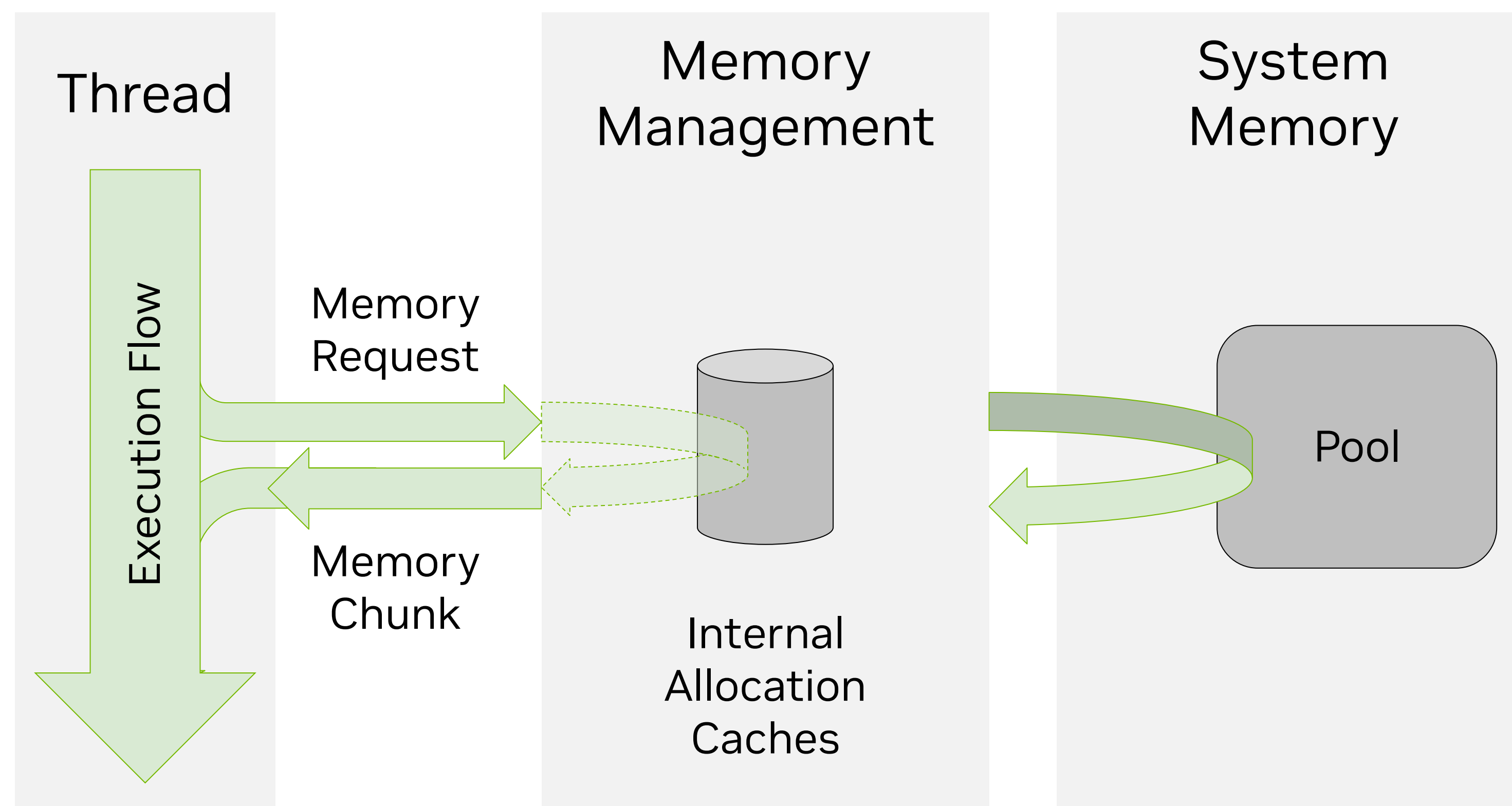
Tradeoffs

- Fine-grained application possible based on safety requirements and detailed call chain analysis
- More context switching vs. more safety analysis – you can dial either up/down

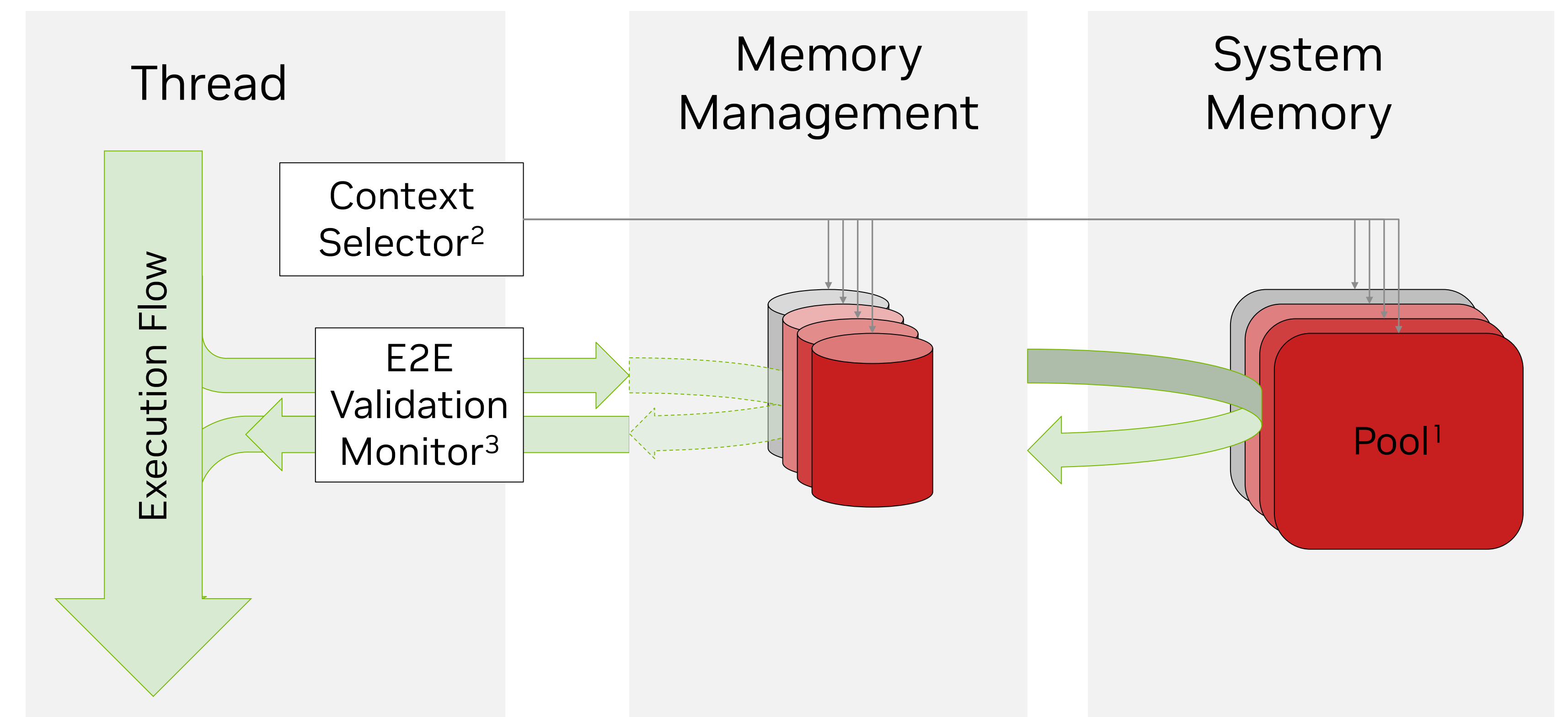
Isolated Memory Pools Enable Safe Contexts

One pool per context

Standard Linux Memory Management



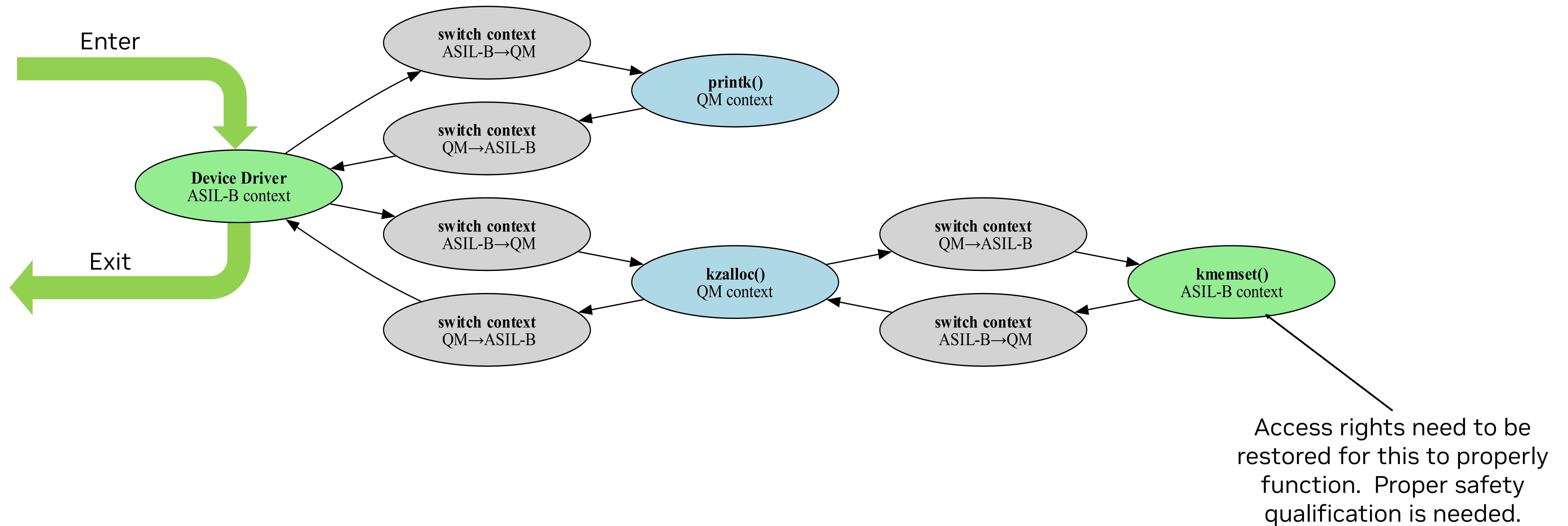
Multi-Context Isolated Memory Pools



1. Pools & caches dimensioned for safety
2. Context-aware memory allocation
3. Monitor to confirm proper allocation

Device Driver Context Switching

Simplified example to demonstrate the expected behaviors



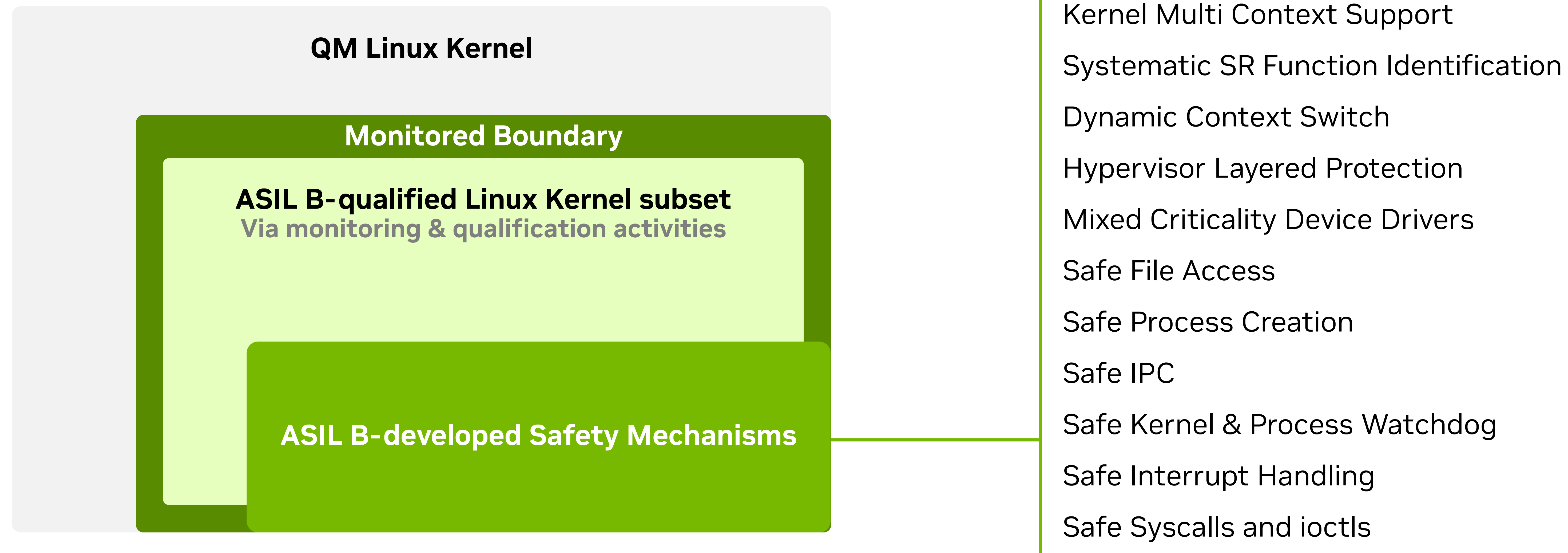
From Boot To Running

Starting the system
Kernel + Safe Initramfs loaded into RAM
Kernel core initialization: <ul style="list-style-type: none">• creation of pools
ASIL-dependent Device Drivers Initialization
ASIL-dependent Processes Initialization
End of Initialization: <ul style="list-style-type: none">• Denial of further ASIL allocations• Prevention of further init-only ASIL-mode operations
Engaging into Safety-Relevant Activity (e.g. the autonomous driving cycle can start)

Running the system
Active Safety Mechanisms: Periodic ASIL Functional Validation
Passive Safety Mechanisms: Catch crossing of safety Tripwires <ul style="list-style-type: none">• Forbidden write operations to higher ASIL• Attempts to allocate more ASIL resources• Deviations from expected timing• Deviation of specific parameters from ranges specified by device drivers at initialization time (range checking)

Use Cases & Mechanisms

Safety Perspective





Summary of Our Approach To Safety For Linux

Yes, we believe it can be done

Components

- Linux Kernel
- Userspace libraries (e.g. libc)
- Other userspace components (e.g. init processes)

Qualification


- Up to ASIL B, according to ISO 26262:2018, and SIL2 per IEC 61508
- Expecting seamless transition to ISO 26262 3rd edition

Path

- ISO/PAS 8926: standard tailoring of ISO 26262 Part 6 for pre-existing software elements
- ISO 26262 Part 8-12
- Additional safety measures (e.g. safety monitoring)

NVIDIA Linux for Safety Concept

TÜV SÜD Technical Report



Add value.
Inspire trust.

Technical Report

on the

Concept

of the

NVIDIA Linux for Safety

Applicant

NVIDIA Corporation
2788 San Tomas Expressway
Santa Clara, CA 95051
USA

Report No.: NS104835T
Version 1.0 of 2025-05-13

Testing Laboratory for Safety Components

TÜV SÜD Rail GmbH
Rail Automation
Barthstraße 16
D-80339 München

(Page 1 of 13)

Development and Qualification Approach:

- An intermediate result of the review of the development and qualification approach is documented in [R1] Review Protocol List of Open Points.
- In additional meetings during the project’s runtime further details on the planned approach were checked by TÜV SÜD Rail GmbH. As explained above, the CPA is one of the key activities to allow for claiming completeness of the monitors, but these analyses have not yet been conducted for NVIDIA Linux for Safety and could therefore not be checked. The shown approach via the two phases and the different processes and the CPA as a verification activity is generally suitable for ASIL B. The correct application of these processes as well as the correctness and completeness of the planned safety analysis must be demonstrated during the project detailed phase and verified by TÜV SÜD Rail GmbH.

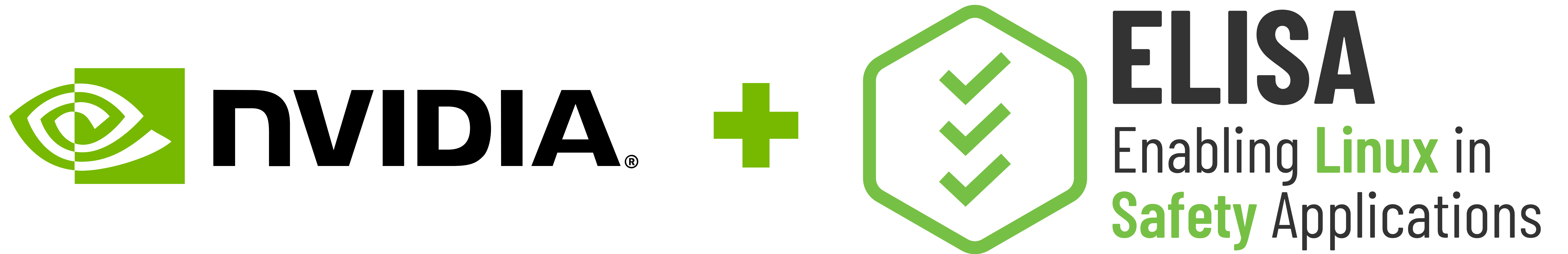
Safety Concept and Initial Safety Mechanisms:

- An intermediate result of the review of the development and qualification approach is documented in [R1] Review Protocol List of Open Points. In additional meetings during the project’s runtime further details on the planned approach were checked by TÜV SÜD Rail GmbH. As explained above, the CPA is one of the key activities to allow for claiming completeness of the monitors, but these analyses have not yet been conducted for NVIDIA Linux for Safety and could therefore not be checked. The shown approach via the two phases, processes to apply and the CPA as a verification activity is generally suitable for ASIL B. The effectiveness of these measures must be demonstrated and verified during the project detailed phase by TÜV SÜD Rail GmbH.

Final Summary:

- The approach for the NVIDIA Linux for Safety from NVIDIA Corporation is in principle able to fulfil the requirements in accordance to ASIL B of the standards mentioned in chapter 2.1. The planned process and approach as outlined in Safe Linux Safety Concept Document and Safe Linux Tailored Process Strategy shall be applied and met in the following detailed phase.

NVIDIA Joins ELISA!



Goals

- Share our safety expertise with the Linux community
- Continue to develop our safety concept with community input
- Help develop common tools for safety like fault injection, static analysis, call chain analysis, etc.
- Work on advanced architectures and processes with a goal of achieving ASIL-D / SIL-3 safety

