IWOCL / SYCLcon

codeplay

Can SYCL and OpenCL meet the challenges of functional safety

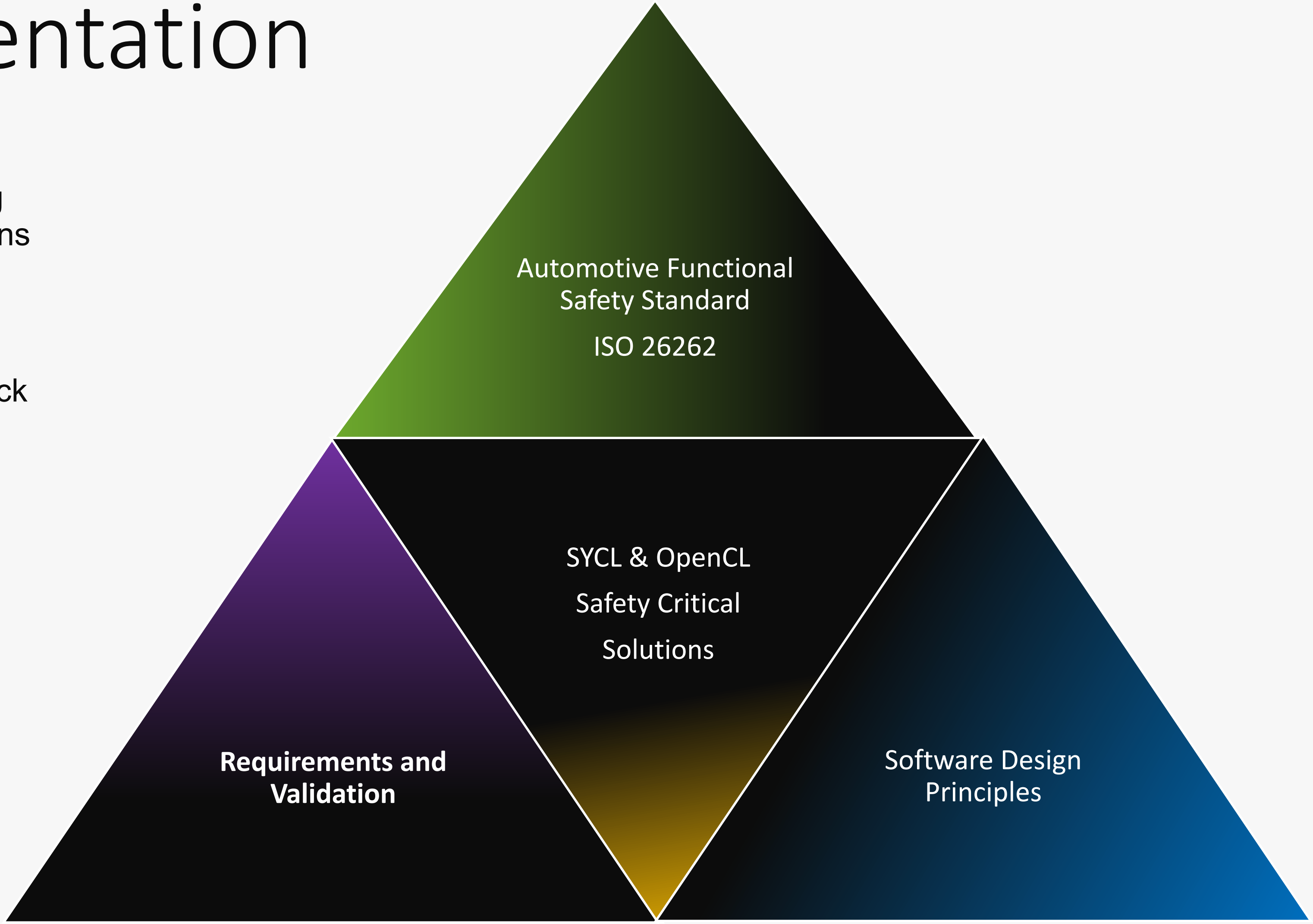
April 9th 2021

Illya Rudkin, Principle Engineer - Lead Safety Practitioner

Aim of this presentation

Look at some of the challenges put to adopting SYCL and OpenCL for safety critical applications

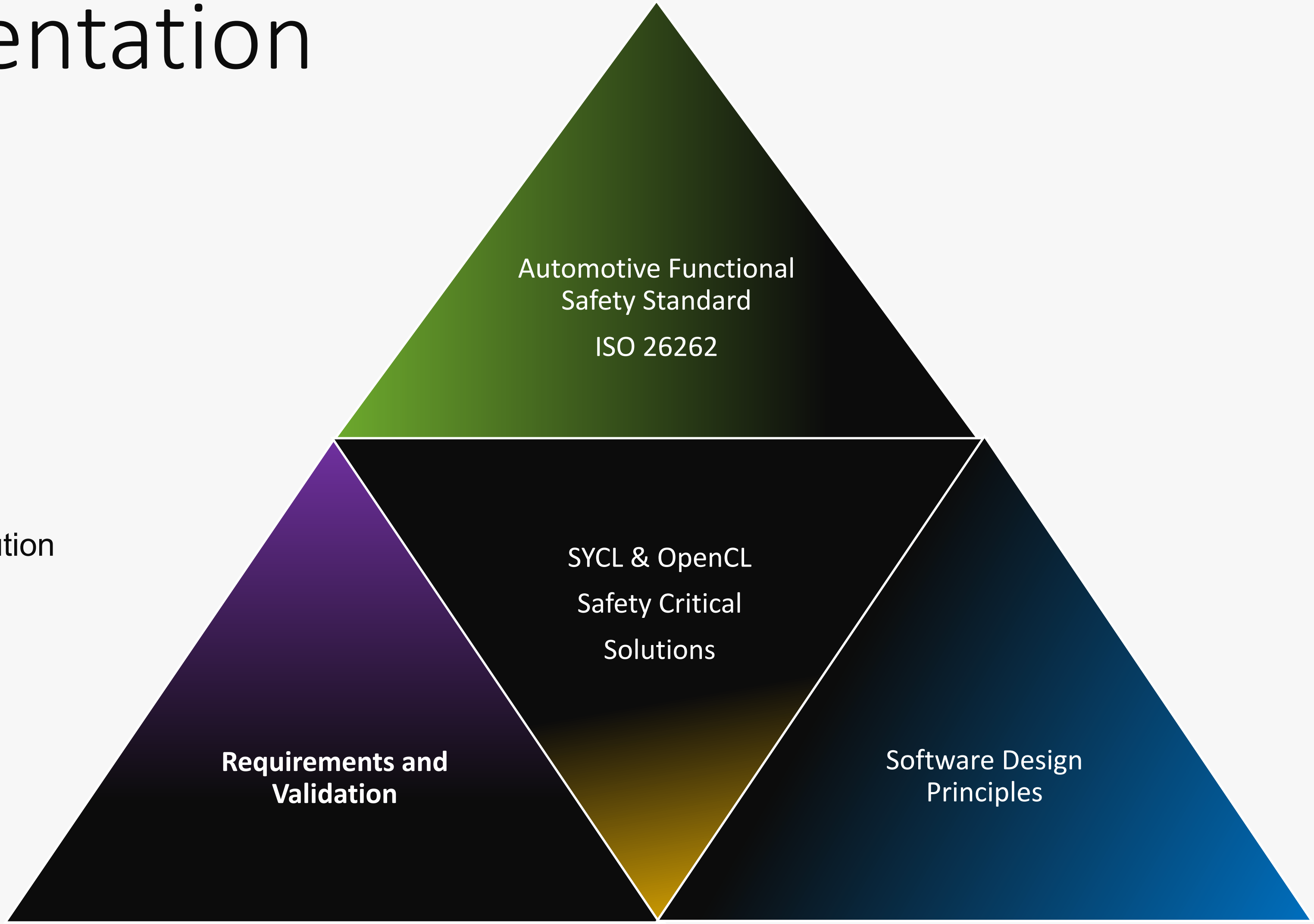
- Give you a context of ISO 26262
- How functional safety affects the SYCL stack
- Example architecture of an automotive platform
- How the SYCL stack today supports safety and some aspects that could be improved
- Quick history of autonomous vehicles



Aim of this presentation

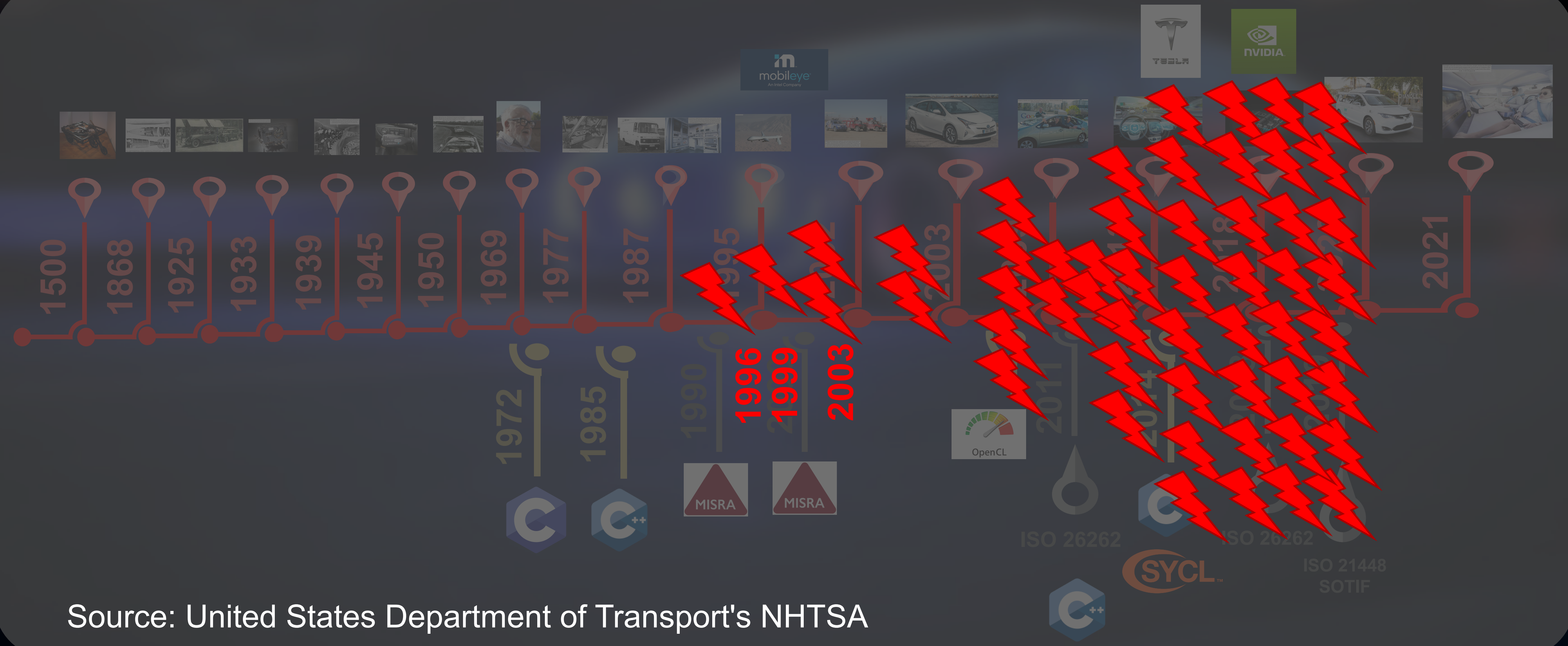
Functional safety is a big topic and so this presentation will not cover:

- The RTOS
- The tool chain and tool qualification
- Statistics / results / reports
- Online offline kernel compilation and execution
- Khronos® safety critical specifications
- Comparison with other parallel computing platforms like CUDA®



History of autonomous vehicles

History of autonomous vehicles



Source: United States Department of Transport's NHTSA

What can the industry do about it?

Create a safety standard

Automotive Functional Safety Standard ISO 26262

First published in 2011, and the current second edition was released in 2018.

- Covers all electrical systems in a vehicle
- Widely adopted by most automotive OEMs

ISO 26262
It is about risk of failure management

Fact: Complex software contains bugs and edge cases
Fact: Not all bugs will be found or fixed

Fact: Most bugs or edges cases can be mitigated with **good software engineering practices**, the **analysis of the requirements** backed up with a **verification strategies** appropriate to the consequences of failure.

Eliminate of Software Bugs

There are generally two types of bugs:

- Random failure
- Systematic failure
  - Human error
  - Human misunderstanding or assumptions
  - Limited testing coverage to catch 'non-conformities'

Automotive Functional Safety Standard ISO 26262

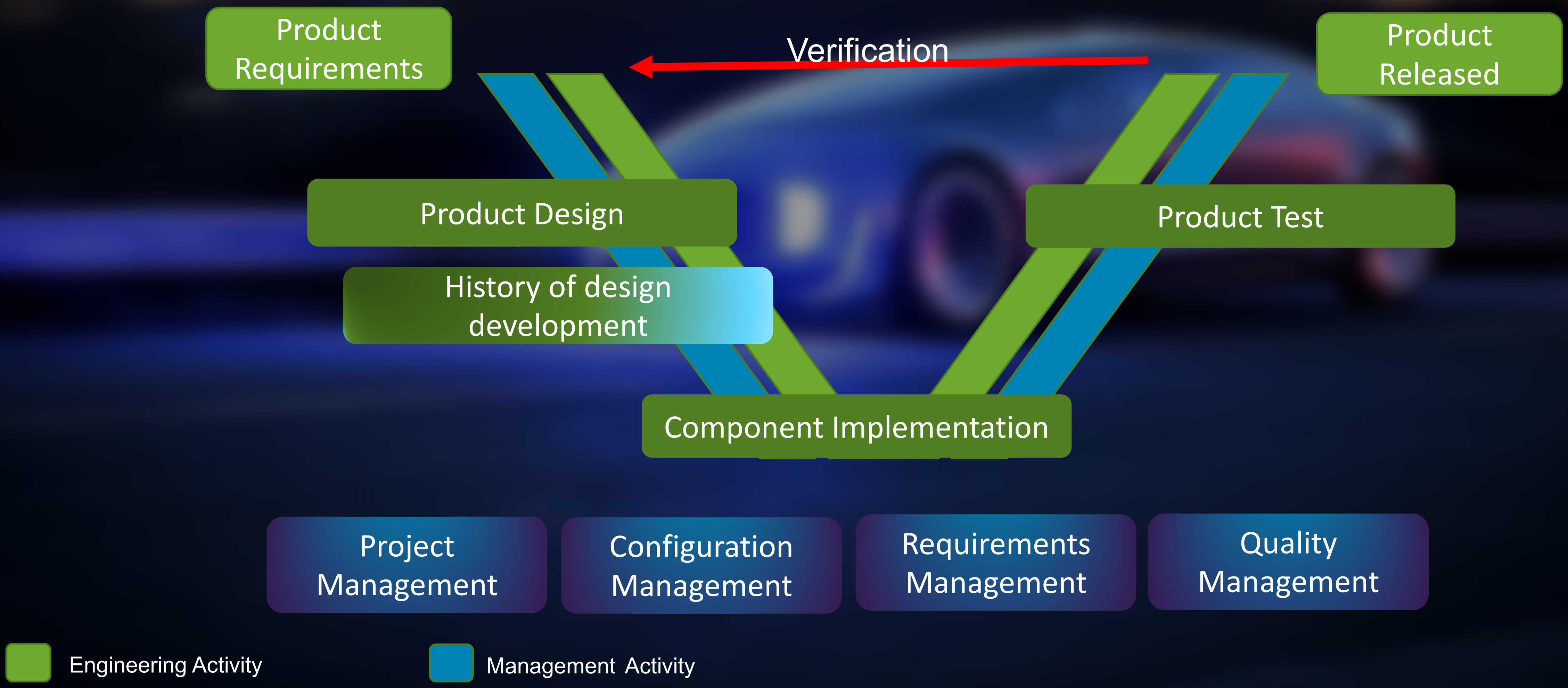
Brief:

- Covers all electrical systems in a vehicle
- Widely adopted by most automotive OEMs
- It consists of two parts:
  - Builds on the ISO 9001 quality management standard
  - Additional safety activities

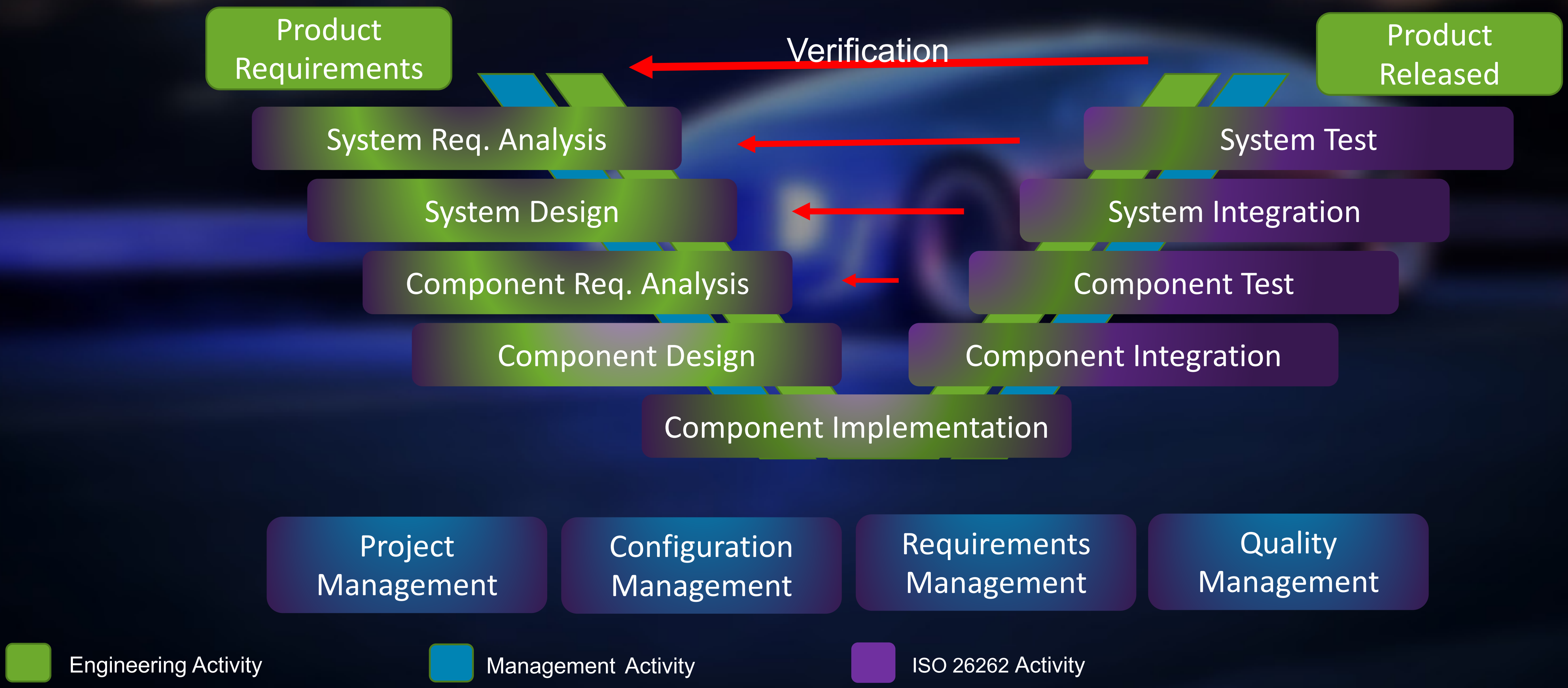
How does ISO 26262 achieve this?

Through Analysis + Verification

Quality Managed Software Development - ISO 9001



Automotive Functional Safety Standard ISO 26262



ISO 26262

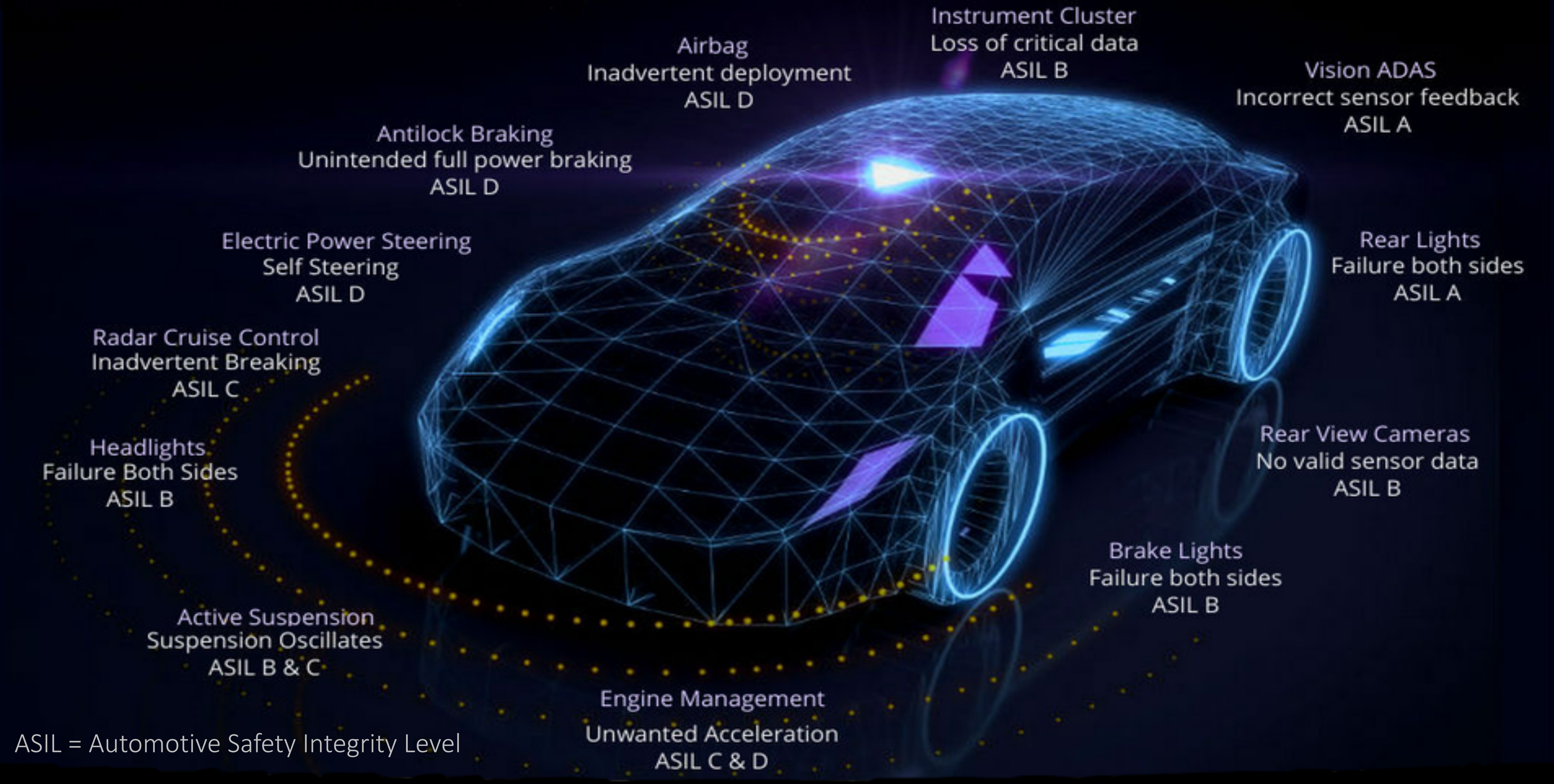
The management of risk

Factors:

- Severity - the potential of harm
- Exposure – the probability of occurrence
- Controllability – the ability to avoid injury

Requirements + Analysis

Typical Automotive ASIL Classifications



ASIL = Automotive Safety Integrity Level

Image credit: Mentor

ISO 26262 – It is about risk of failure management

Determining the ASIL levels for Electrical Systems

Malfunction	Brake ABS system failure
Hazard Analysis	State the unintended situations (hazards) that could occur
Risk Analysis	How likely is the hazard to happen?
ASIL Determination	What level of safety (risk) reduction does the system need?

Low — ASIL — High

A B C D

ASIL = Automotive Safety Integrity Level

ISO 26262

Get to a safe state

Reach your safety goal requirement

ISO 26262:2018 - Fault Tolerant Time Interval (FTTI)

“The minimum timespan from the occurrence of a fault in an item to a possible occurrence of a hazardous event, if the safety mechanisms are not activated.”

Hazard occurs
(Safety goal violated)

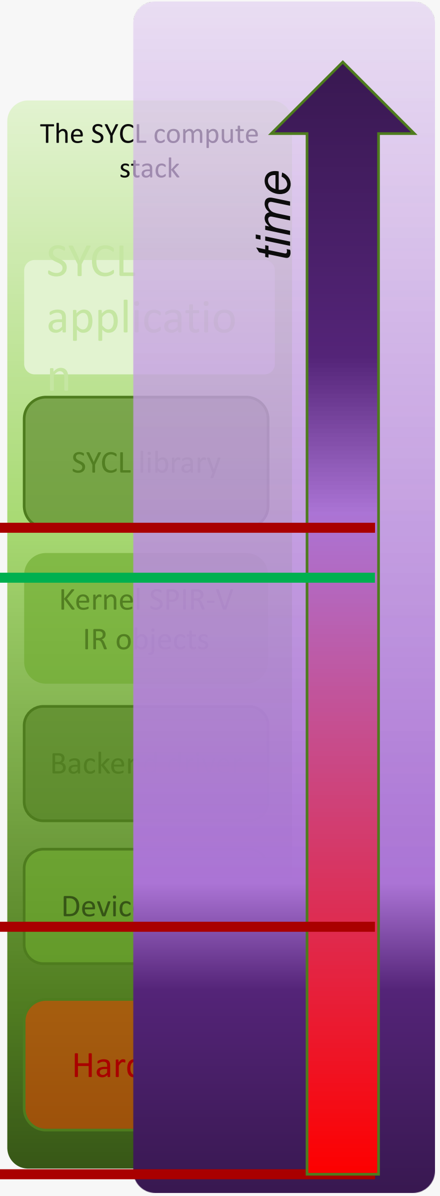
Stack reached safe state

Hazard manifestation period

Malfunctioning behavior

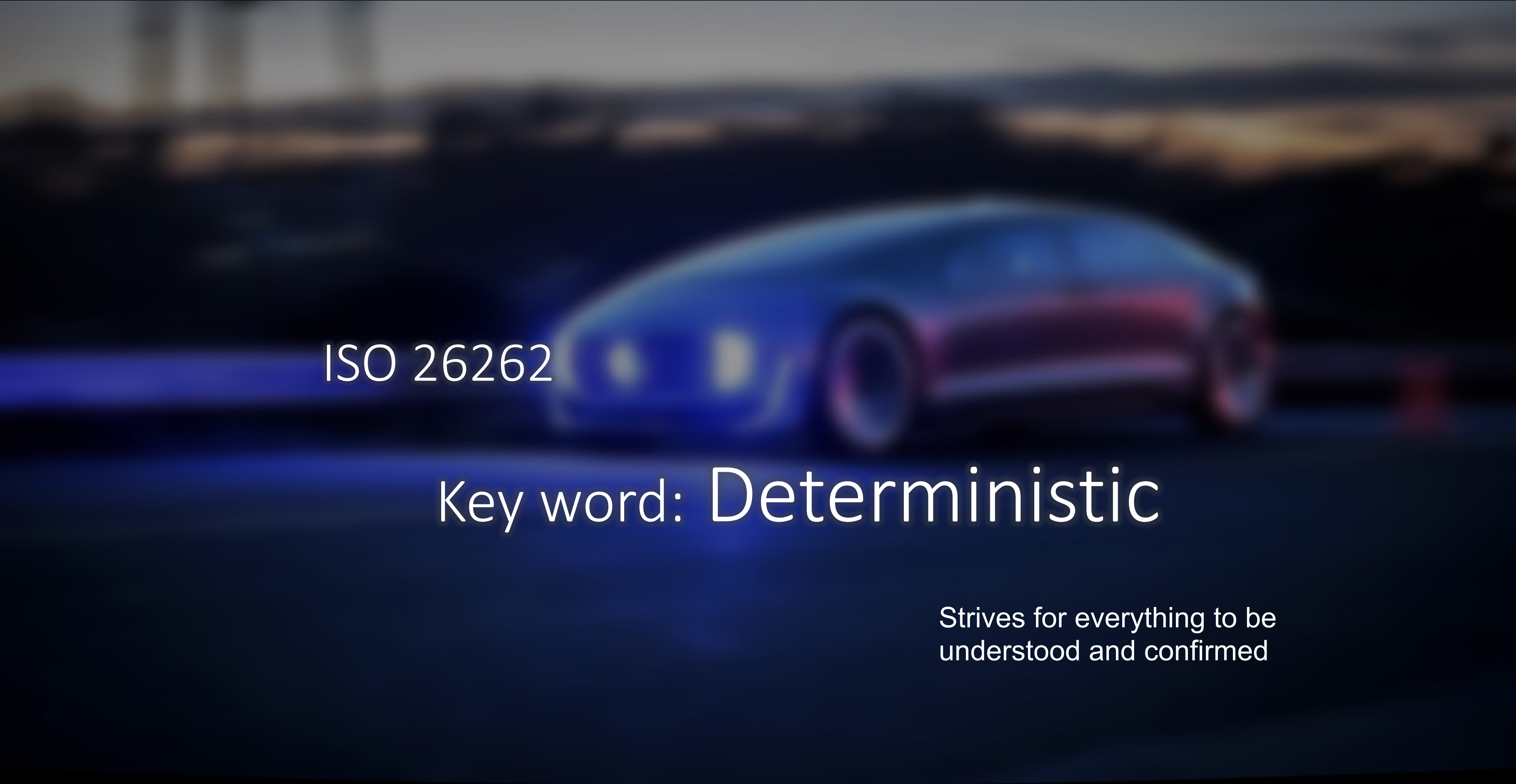
Malfunction manifestation period

Fault occurs



Safety mechanism can be:

- A limited operation mode
- Output an error status
- Shutdown
- Reset
- Activate safety systems
- Deactivate until within operational parameters
- An external operational monitor



ISO 26262

Key word: Deterministic

Strives for everything to be understood and confirmed

What is not 'deterministic' in the stack?

Thread dead lock

Random bit flip

Data integrity

Kernel execution time

Scheduler

Memory read write order

Single point code failure

Interrupt kernel execution

Compiler

Auto optimized code

Scheduler

Denial of Service call

Rogue kernels

Freedom from interference

Concurrency

Resource availability

Dynamic memory latency

Code execution paths

Application resource management

C++ exception handling

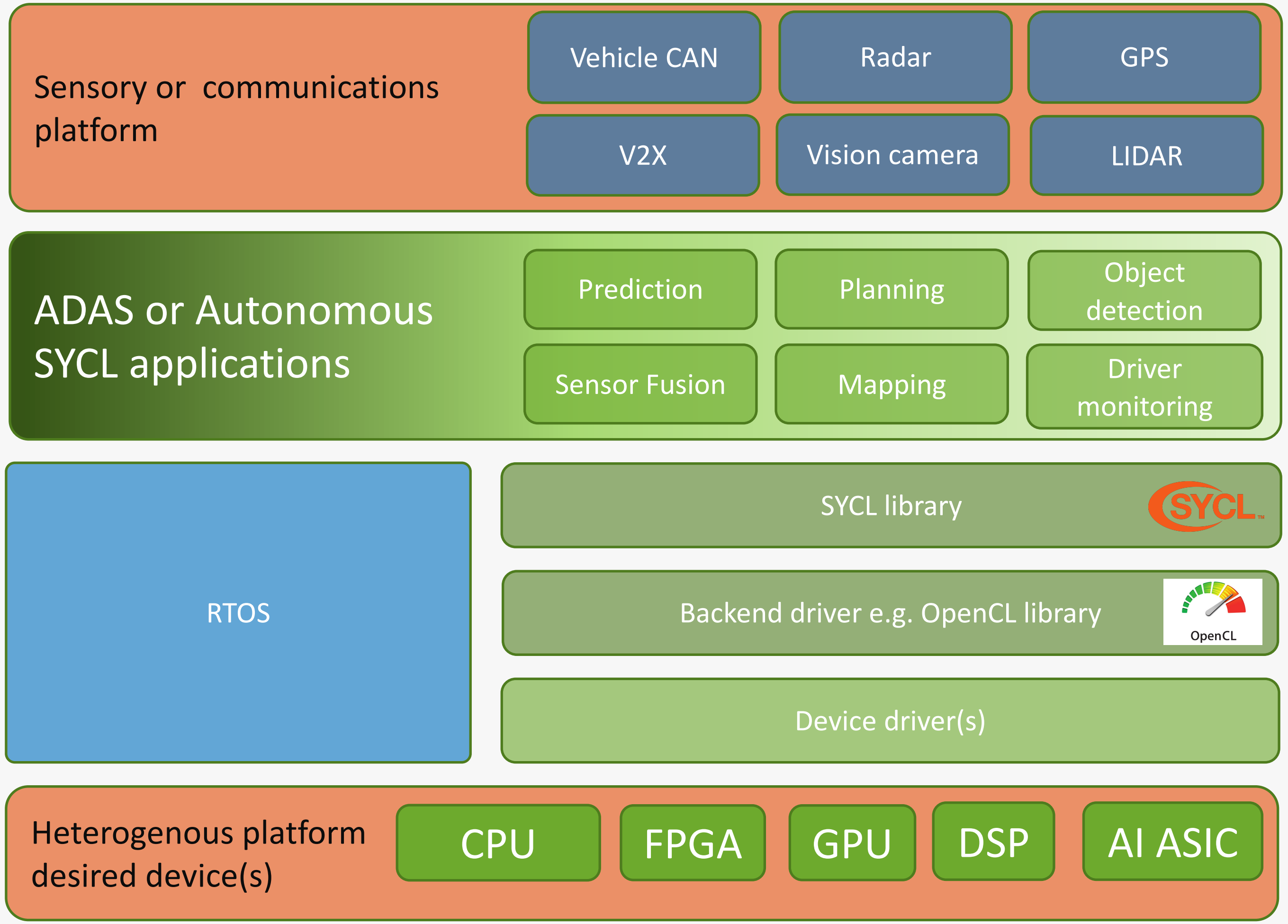
Execution performance

Kernel execution hang

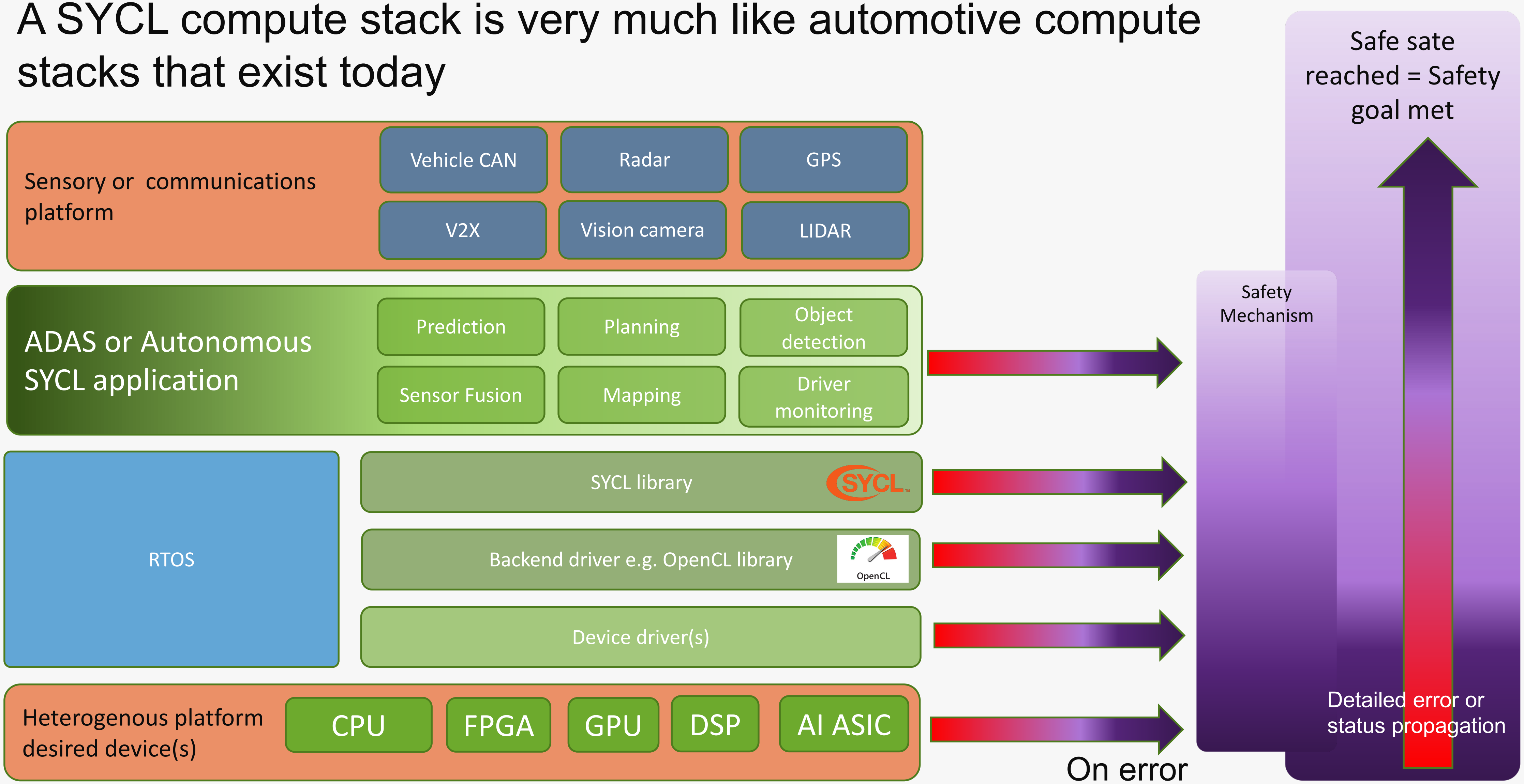
Can you provide assurances these will not cause a failure?

The automotive SYCL compute stack

A SYCL automotive compute stack is very much like typical automotive compute stacks that exist today



A SYCL compute stack is very much like automotive compute stacks that exist today



So, what is ISO 26262 asking you to do?

1st By removing the issues

We reduce the chance of failure by
- Having clear requirements
- Performing analysis
- Verification of the implementation

+

2nd On a failure occurring

Initiate the safety mechanism

→ It runs you through a set of design and software requirements

ISO 26262 Part 6: Software Development

As a developer you would have to demonstrate:

- How you would carry out the activity
- Validate the activity
- Produce evidence of the activity

The ISO 26262 requirements – The levels of rigor to apply

Table 1 — Topics to be covered by modelling and coding guidelines

	Topics	**ASIL**			
		A	**B**	**C**	**D**
1a	Enforcement of low complexity[a]	++	++	++	++
1b	Use of language subsets[b]	++	++	++	++
1c	Enforcement of strong typing[c]	++	++	++	++
1d	Use of defensive implementation techniques[d]	+	+	++	++
1e	Use of well-trusted design principles[e]	+	+	++	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++
1i	Concurrency aspects[f]	+	+	+	+

ASIL = Automotive Safety Integrity Level

o = need not apply
+ = recommended practices or 'should be using anyway'
++ = highly recommended practices or 'mandatory'. A justification is needed if not used.

The ISO 26262 requirements – The levels of rigor to apply

Table 3 — Principles for software architectural design

	Principles	ASIL A	ASIL B	ASIL C	ASIL D
1a	Appropriate hierarchical structure of the software components	++	++	++	++
1b	Restricted size and complexity of software components[a]	++	++	++	++
1c	Restricted size of interfaces[a]	+	+	+	++
1d	Strong cohesion within each software component[b]	+	++	++	++
1e	Loose coupling between software components[b,c]	+	++	++	++
1f	Appropriate scheduling properties	++	++	++	++
1g	Restricted use of interrupts[a,d]	+	+	+	++
1h	Appropriate spatial isolation of the software components	+	+	+	++
1i	Appropriate management of shared resources[e]	++	++	++	++

ASIL = Automotive Safety Integrity Level

o = need not apply
+ = recommended practices or 'should be using anyway'
++ = highly recommended practices or 'mandatory'. A justification is needed if not used.

The ISO 26262 requirements – The levels of rigor to apply

Table 4 — Methods for the verification of the software architectural design

	Methods	**ASIL A**	**ASIL B**	**ASIL C**	**ASIL D**
1a	Walk-through of the design[a]	++	+	o	o
1b	Inspection of the design[a]	+	++	++	++
1c	Simulation of dynamic behaviour of the design	+	+	+	++
1d	Prototype generation	o	o	+	++
1e	Formal verification	o	o	+	+
1f	Control flow analysis[b]	+	+	++	++
1g	Data flow analysis[b]	+	+	++	++
1h	Scheduling analysis	+	+	++	++

ASIL = Automotive Safety Integrity Level

o = need not apply
+ = recommended practices or 'should be using anyway'
++ = highly recommended practices or 'mandatory'. A justification is needed if not used.

The ISO 26262 requirements – The levels of rigor to apply

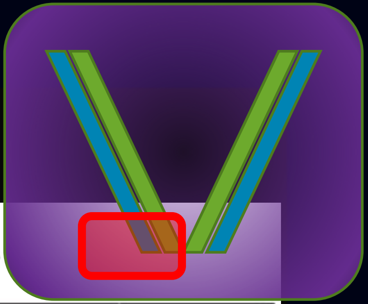


Table 6 — Design principles for software unit design and implementation

	Principle	ASIL A	ASIL B	ASIL C	ASIL D
1a	One entry and one exit point in subprograms and functions[a]	++	++	++	++
1b	No dynamic objects or variables, or else online test during their creation[a]	+	++	++	++
1c	Initialization of variables	++	++	++	++
1d	No multiple use of variable names[a]	++	++	++	++
1e	Avoid global variables or else justify their usage[a]	+	+	++	++
1f	Restricted use of pointers[a]	+	++	++	++
1g	No implicit type conversions[a]	+	++	++	++
1h	No hidden data flow or control flow	+	++	++	++
1i	No unconditional jumps[a]	++	++	++	++
1j	No recursions	+	+	++	++

ASIL = Automotive Safety Integrity Level

o = need not apply
+ = recommended practices or 'should be using anyway'
++ = highly recommended practices or 'mandatory'. A justification is needed if not used.

The ISO 26262 requirements – The levels of rigor to apply

Table 7 — Methods for software unit verification

Methods		ASIL			
		A	B	C	D
1a	Walk-through[a]	++	+	o	o
1b	Pair-programming[a]	+	+	+	+
1c	Inspection[a]	+	++	++	++
1d	Semi-formal verification	+	+	++	++
1e	Formal verification	o	o	+	+
1f	Control flow analysis[b, c]	+	+	++	++
1g	Data flow analysis[b, c]	+	+	++	++
1h	Static code analysis[d]	++	++	++	++
1i	Static analyses based on abstract interpretation[e]	+	+	+	+
1j	Requirements-based test[f]	++	++	++	++
1k	Interface test[g]	++	++	++	++
1l	Fault injection test[h]	+	+	+	++
1m	Resource usage evaluation[i]	+	+	+	++
1n	Back-to-back comparison test between model and code, if applicable[j]	+	+	++	++

ASIL = Automotive Safety Integrity Level

o = need not apply
+ = recommended practices or ‘should be using anyway’
++ = highly recommended practices or ‘mandatory’. A justification is needed if not used.

The ISO 26262 requirements – The levels of rigor to apply

Table 8 — Methods for deriving test cases for software unit testing

Methods		ASIL			
		A	B	C	D
1a	Analysis of requirements	++	++	++	++
1b	Generation and analysis of equivalence classes[a]	+	++	++	++
1c	Analysis of boundary values[b]	+	++	++	++
1d	Error guessing based on knowledge or experience[c]	+	+	+	+

ASIL = Automotive Safety Integrity Level

o = need not apply
+ = recommended practices or ‘should be using anyway’
++ = highly recommended practices or ‘mandatory’. A justification is needed if not used.

The ISO 26262 requirements – The levels of rigor to apply

Table 10 — Methods for verification of software integration

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Requirements-based test[a]	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[b]	+	+	++	++
1d	Resource usage evaluation[c, d]	++	++	++	++
1e	Back-to-back comparison test between model and code, if applicable[e]	+	+	++	++
1f	Verification of the control flow and data flow	+	+	++	++
1g	Static code analysis[f]	++	++	++	++
1h	Static analyses based on abstract interpretation[g]	+	+	+	+

ASIL = Automotive Safety Integrity Level

o = need not apply
+ = recommended practices or 'should be using anyway'
++ = highly recommended practices or 'mandatory'. A justification is needed if not used.

The ISO 26262 requirements – The levels of rigor to apply

Table 15 — Methods for deriving test cases for the test of the embedded software

Methods		ASIL			
		A	B	C	D
1a	Analysis of requirements	++	++	++	++
1b	Generation and analysis of equivalence classes	+	++	++	++
1c	Analysis of boundary values	+	+	++	++
1d	Error guessing based on knowledge or experience	+	+	++	++
1e	Analysis of functional dependencies	+	+	++	++
1f	Analysis of operational use cases[a]	+	++	++	++

ASIL = Automotive Safety Integrity Level

o = need not apply
+ = recommended practices or 'should be using anyway'
++ = highly recommended practices or 'mandatory'. A justification is needed if not used.

Conclusion

SYCL and OpenCL specifications and current implementations are not designed with safety in mind.

With regards to the SYCL application stack meeting the challenge of ISO 26262, it has:

- Good integration design principles
  - Separation of concerns
  - Clearly identifiable API boundaries, call hierarchy and responsibilities
  - Supports integration testing
- High quality Khronos specifications with requirements

To support safety mechanisms and deterministic behavior throughout the stack, it needs:

- Resource management models and verification methods
- Concurrency and scheduling of task models
- Improved support for communicating propagating up status and error conditions
- Improved support for negative testing
- Data integrity checking
- Operational self test support and reporting

Thank you

Contact: illya@codeplay.com

Enable AI and HPC to be open, safe and accessible to all

DOI: 10.1145/3456669.3456688 ISBN: 978-1-4503-9033-0/21/04