CPP-Summit 2019

全球C++软件技术大会

C++ Development Technology Summit

Boolan

高端IT互联网教育平台

关注“博览Boolan”服务号

发现更多 会议·课程·活动

Writing Safety Critical  Software for High Performance hardware

Michael Wong
Codeplay Software Ltd.
Vice President of Research & Development

VP of R&D of Codeplay

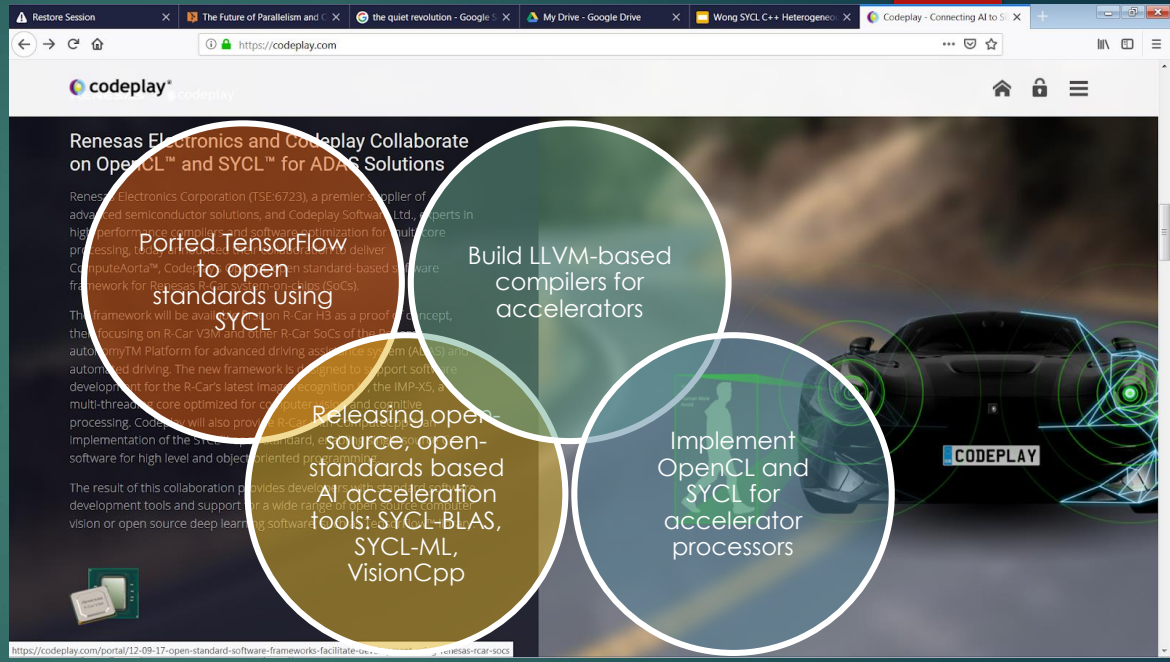
Chair of SYCL Heterogeneous Programming Language
C++ Directions Group
ISOCPP.org Director, VP
http://isocpp.org/wiki/faq/wg21#michael-wong

Head of Delegation for C++ Standard for Canada
Chair of Programming Languages for Standards Council of Canada
Chair of WG21 SG19 Machine Learning
Chair of WG21 SG14 Games Dev/Low Latency/Financial Trading/Embedded
Editor: C++ SG5 Transactional Memory Technical Specification
Editor: C++ SG1 Concurrency Technical Specification
MISRA C++ and AUTOSAR
wongmichael.com/about

We build GPU compilers for semiconductor companies

- Now working to make AI/ML heterogeneous acceleration safe for autonomous vehicle

Who am I?



Acknowledgement Disclaimer

Numerous people internal and external to the original C++/Khronos group, in industry and academia, have made contributions, influenced ideas, written part of this presentations, and offered feedbacks to form part of this talk.

Specifically, Andrew Richards, Marsha Chechik, Paul Mckenney, Joe Hummel, Bjarne Stroustrup, Botond Ballo, Nicolas Becker, Wolfgang Freese, Helen Monkhouse for some of the slides.

I even lifted this acknowledgement and disclaimer from some of them.

But I claim all credit for errors, and stupid mistakes. **These are mine, all mine!**

Legal Disclaimer

THIS WORK REPRESENTS THE VIEW OF THE AUTHOR AND DOES NOT NECESSARILY REPRESENT THE VIEW OF CODEPLAY.

OTHER COMPANY, PRODUCT, AND SERVICE NAMES MAY BE TRADEMARKS OR SERVICE MARKS OF OTHERS.

Codeplay's Business

- Build OpenCL & Vulkan support for new processors
- Building LLVM compiler back-ends
- Building processor-specific drivers, profilers, debuggers
- ComputeAorta product licensing for OpenCL/Vulkan

Silicon Enablement

- Open-source services in AI and graphics
- TensorFlow, Eigen, clSPV, SPIR-V Tools, SYCL-DNN, SYCL-BLAS, SYCL-ML, C++ Parallel STL
- Performance optimizations per-processor
- ComputeCpp SYCL: Free & Pro

Ecosystem

- Functional safety AI acceleration tools
- Working towards: ISO 9001, 16949, ISO 26262, MISRA, AUTOSAR
- Renesas R-Car ADAS OpenCL & SYCL tools

Automotive

3 Act Play

1. What does the industry need to enable safe and secure software for the future?
2. What is ISO2626, MISRA, Autosar, WG23, Core Guidelines …?
3. The two prong approach: C++'s SG12 and MISRA C++

Act 1

1. What does the industry need to enable safe and secure software for the future?

Many Safety Standards

- Which one to use?
- What process to adopt?

Safety and Security

- Safe is not Secure

- Security needs Safety
- A separate topic

Learn from others

- Automotive
- Aircraft

Continual Merging of Language standards and safety standards

Language standards

Augment high performance language features with safety

Safety Standards

Improve Safety Standards to match new language features

Continual change need continual improvement

- Latest Language Standard level changes
- Programmer coding level changes
- Safety Standards usually lags both

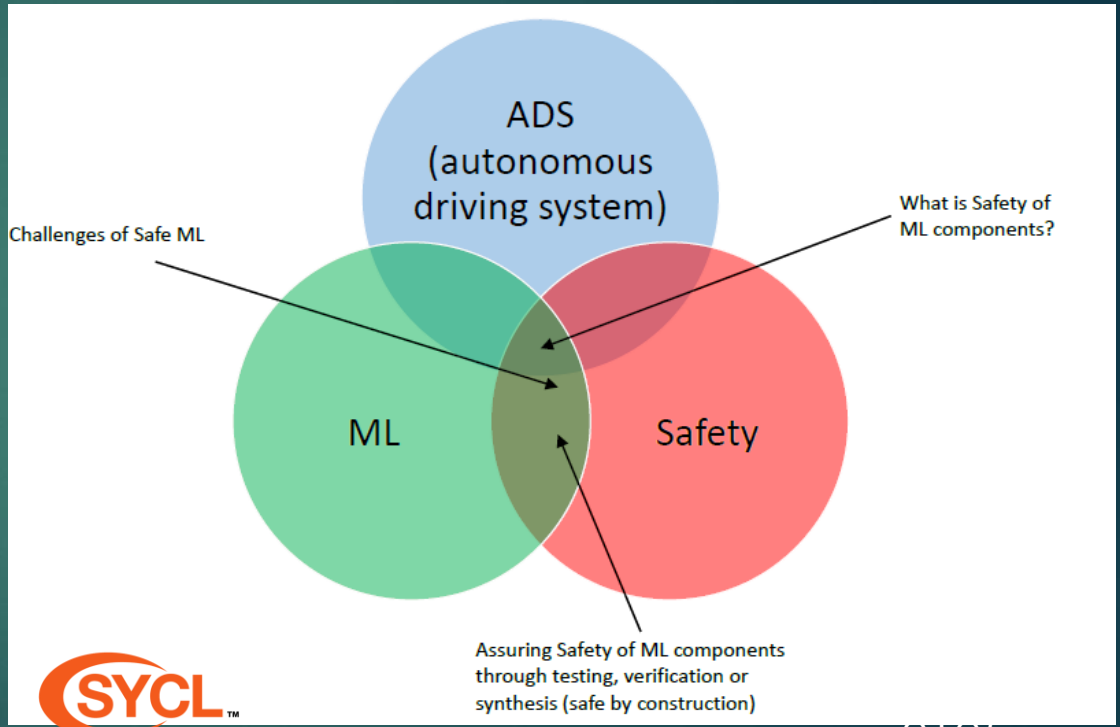
- Education
- Certification
- Automated Tooling
- Root Cause analysis
- Process Control

Look at other Domains' Safety Standards

Domain	Domain Specific Safety Levels				
Automotive (ISO 26262)	**QM**	**ASIL-A**	**ASIL-B/C**	**ASIL-D**	**-**
General (IEC-61508)		SIL-1	SIL-2	SIL-3	SIL-4
Aviation (ED-12/DO-178/DO-254)	DAL-E	DAL-D	DAL-C	DAL-B	DAL-A
Railway (CENELEC 50126/128/129)	-	SIL-1	SIL-2	SIL-3	SIL-4

What is the problem and how can we help make self driving cars safe?

- Autonomous Driving needs to dispatch to GPU/ AI processors
- Most AI software converges on C++/Python
- We need a High level Open Standard programming language that dispatches to Heterogeneous devices
- That is also certified with Safety Standards
- This talk is about that language
- Based on Marsha Chehik's class



The Dream

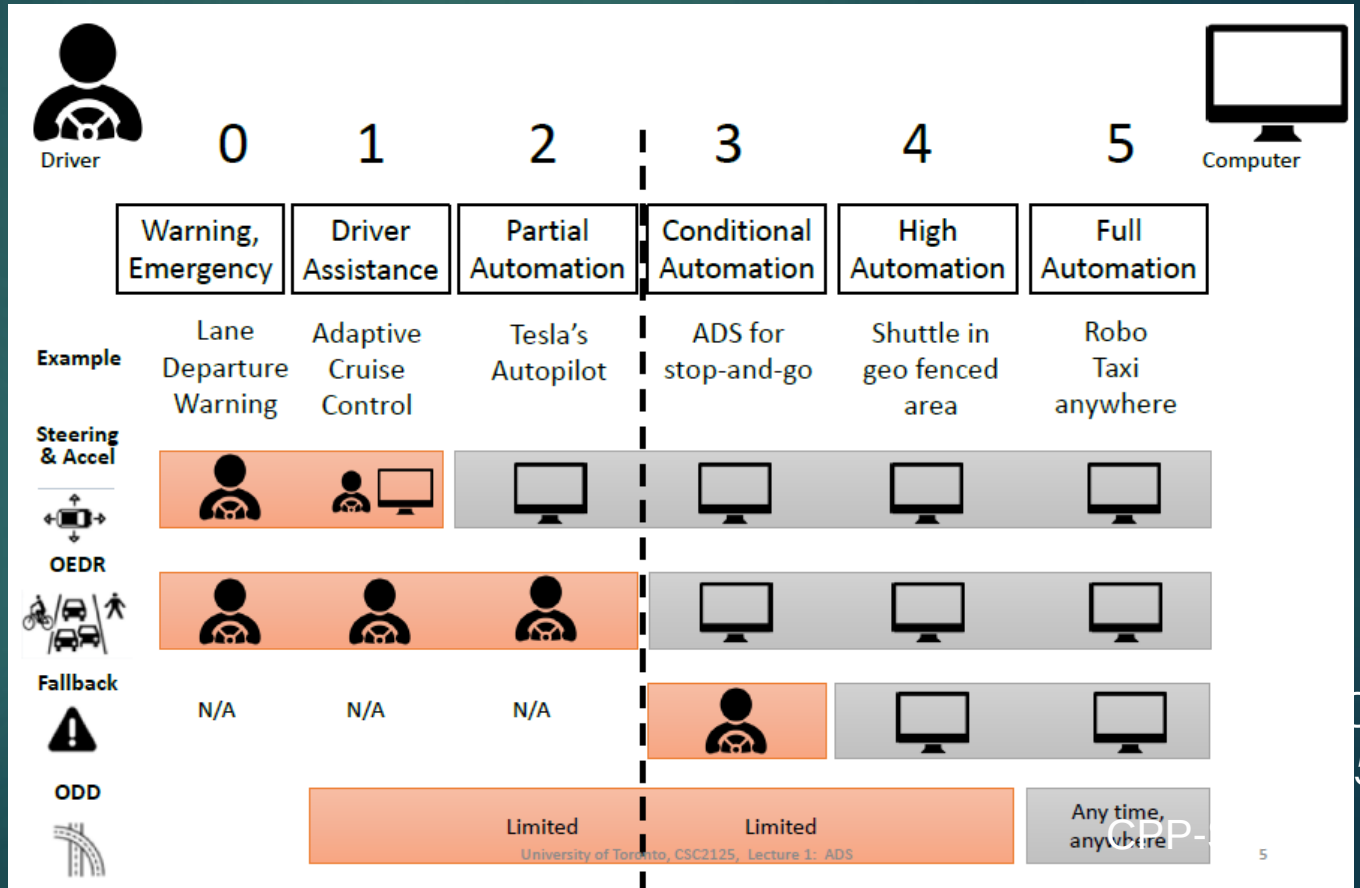
CSC 2125

CPP-Summit 2019

The Reality

https://www.youtube.com/watch?v=gn7UBuuil55

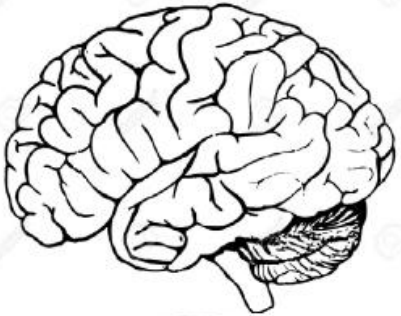
SAE J3016 Self Driving car Automation Levels



How did I get here: What does it take to drive a car?

1. Perception

2. Decision making



3. Control

But if I were a self-driving car …

- Localization and Mapping –Where am I
- Semantic Segmentation or Scene Understanding –Where is Everyone/everything Else?
- Movement Planning –How to get from Point A to Point B
- Driver State –What is the Driver Up to? Essential if driver is part of the loop!
- **Safety Monitoring**

Functional safety

- Safety doesn’t mean the system doesn’t fail

➢ Safety means the system fails safely

- How do you know if the system fails?

➢ You have to detect the failure with a high level of accuracy

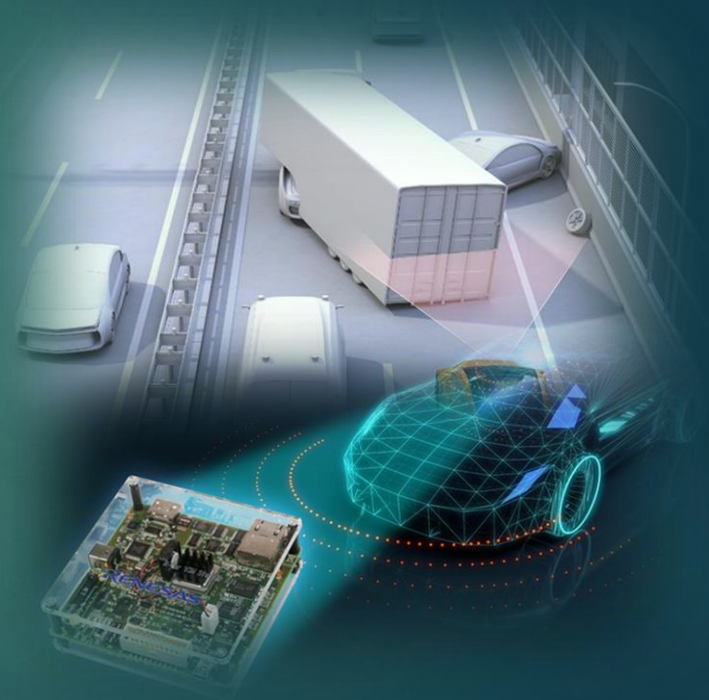
➢ Both incorrect results and late results are a failure

- What do you do if the system fails?

➢ You have to come up with a safe state to return to

Functional Safety

- ISO 26262 (2011) "Absence of *unreasonable* risk due to *hazards* caused by malfunctioning behavior of electrical/electronic systems"
- The standard requires the Development of the Product to be "State of the Art"
- **IEC 61508:** Part of the overall safety related to the equipment under control (EUC) that depends on the correct functioning of the safety-related system



Safety failure types

Systematic Failures:

Result from a failure in design or manufacturing

Often a result of failure to follow best practices

Rate of systematic failures can be reduced through continual and rigorous process improvement

Random Failures:

Result from random defects inherent to process or usage condition

Rate of random failures cannot generally be reduced; focus must be on the detection and handling of random failures in the application

ISO26262 Functional Safety of Road Vehicles

- Standard has 10 parts
- Spans across **~450 pages**
- Require the production of **~120 work products**
  - **…** that are the result of fulfilling a much larger number of **requirements** and **recommendations**

CSC 2125

CPP-Summit 2019

And even if you get it right

You can’t prevent the following recent crashes of self driving cars …

View discretion is advised

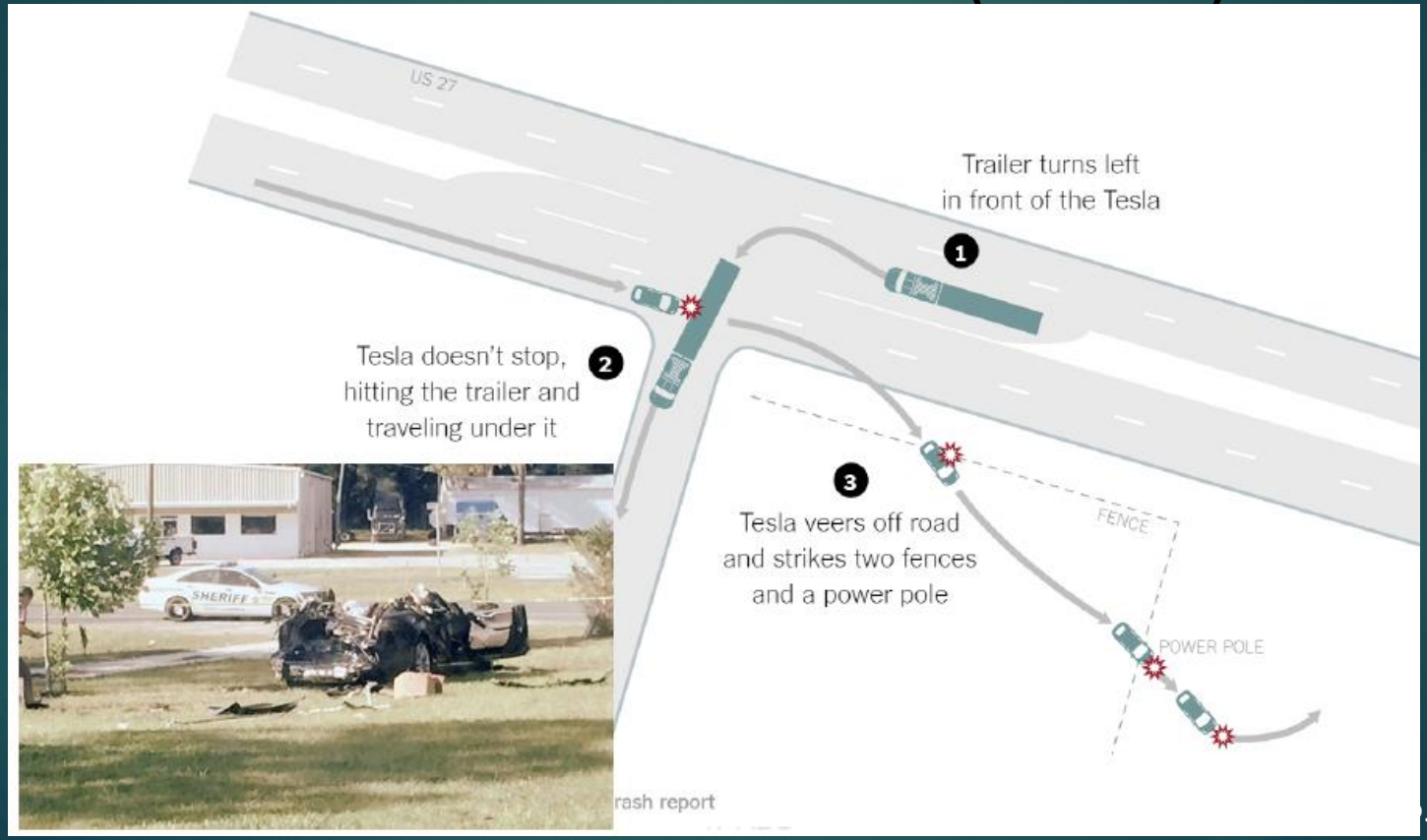
- Some may find the following videos disturbing.
- Look away, you have been warned.

1st Fatal Tesla Autopilot Crash2016 -January 20th-Fatal -Tesla Model S(China)
https://www.youtube.com/watch?v=1cfcumft5n8&t=17s

1st Fatal Tesla Autopilot Crash Analysis

- Model S was equipped with a single forward facing radar,
  - a single forward facing camera,
  - a set of 12 ultrasonic sensors.
- Camera was used by MobileEye's EyeQ3 computing platform implementing a Deep Neural Network (DNN) for its object identification and detection
- Vehicle was also equipped with Tesla's Automatic Emergency Braking (AEB) system
  - AEB system required agreement between ***both*** the camera and the radar before any action was taken.
- Driver monitoring system consisted of a torque sensor in the steering wheel

2nd Fatal Tesla Autopilot Crash2017 -May 7th - Fatal -Tesla Model S (Florida)



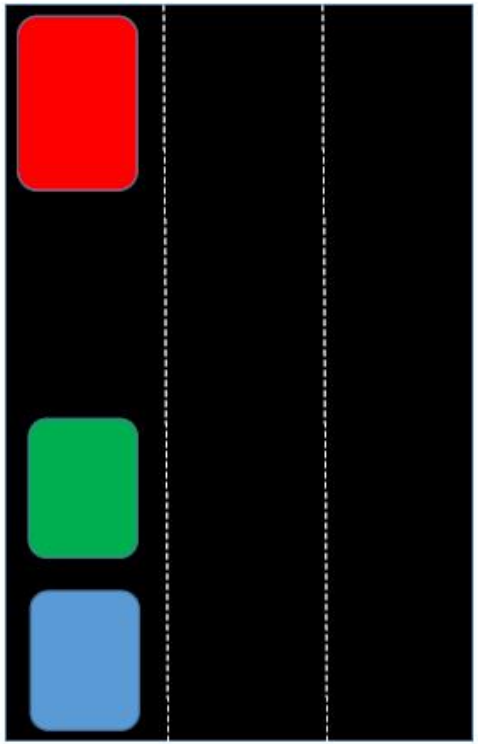
2nd Fatal Tesla Autopilot Crash Analysis

- Similar Model S sensors and features to 1st Tesla Autopilot crash
- No braking or avoidance action prior to collision
- Tesla commented that the camera failed to detect the truck due to white color of the trailer against a brightly lit sky and a high ride height.
- They further commented that the radar filtered out the truck as an overhead road sign to prevent false braking.
- In both cases MobilEye commented that: MobileEye's system was not designed to cover all accident scenarios and that Tesla was using it outside of its intended purpose.

3rd Tesla Autopilot Crash 2018 -January 22nd - Non-Fatal –Tesla Model S (California)

Analysis of Tesla Collision with Fire Truck

- Tesla Model S in Autopilot mode was following a pickup truck in left lane
- Pickup changed lanes to avoid a stationary firetruck
- Tesla accelerated into the back of the firetruck at 65 m.p.h



Why the acceleration?

- ACC is part of Autopilot
- Set Max speed (normal cruising speed) & time gap (headway) when following a lead vehicle @speed < max speed

Hypothesis:

- When pickup changed lane distance to new lead vehicle (firetruck) increased
- ACC commanded acceleration to close the gap

CSC 2125

CPP-Summit 2019

Uber Autonomous Vehicle Crash2018 -March 18th -Fatal – Uber Volvo XC90 (Arizona)
https://www.youtube.com/watch?v=Cuo8eq9C3Ec

Uber Accident Analysis

- Switched off Volvo's standard Aptiva/Intel Mobile Eye collision avoidance/mitigation system Initially detected unknown object 6 seconds before impact
- It decided it was a bicycle 1.3 second before impact and would have started braking
- Why switched off? To reduce interference with their software? Avoid false positives?
  - Think of trying to make a right turn @Time Square in New York
- Also switched off Volvo's Driver Distraction Detection System
  - What's a poor autonomous vehicle to do? Maybe requiring having these features turned on by an industry standard assurance case would help!

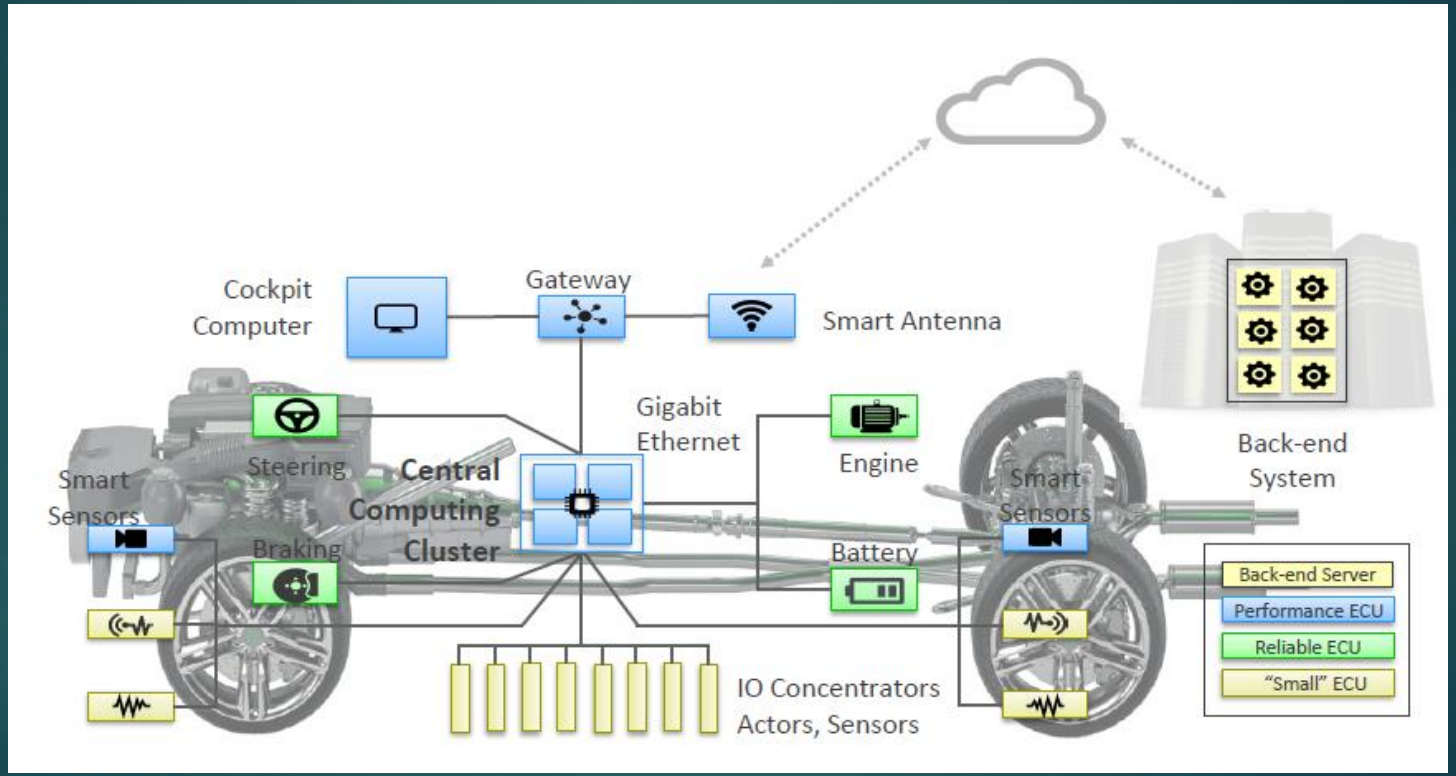
To balance that …

- Many cases where Tesla, Uber and other Self-driving cars may have saved their owner from an accident or fatality
  - One drove its owner having a heart attack to the hospital
  - Prevented a new or timid driver from early crashes or fatal mistakes
- Very useful if you are on a long trip

Getting too (artificially) intelligent with safety

- Object identification is very useful
- Can help predict and plan in addition to help partially meet some safety goals
- Pedestrian detection is an example of how ML fails badly with the key safety requirement: “Don’t hit things!”
- **AI/ML Version:**
  - *“I don’t know what it is so it’s not there.”*
- vs
- **Safety Version:**
  - *“I don’t know what it is BUT IT’S THERE!”*

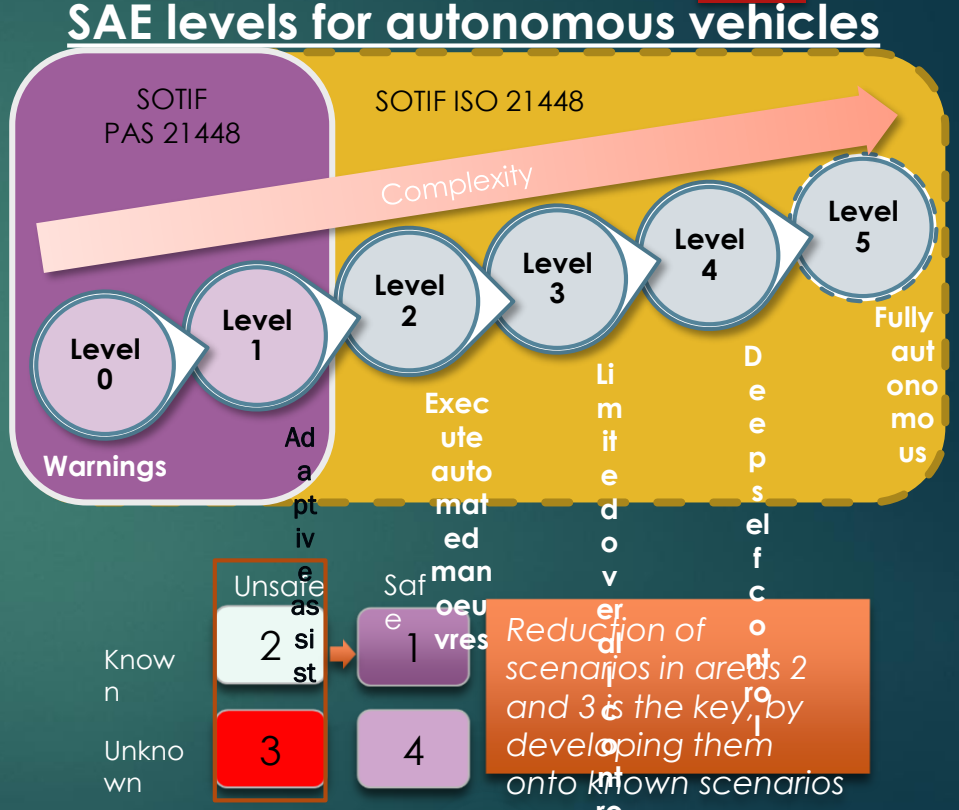
Enter C++ for high performance



The trouble with AI in safety Critical

SOTIF: Safety Of The Intended Function

- Systems or subsystems can cause hazards based on erroneous decision on the environment and not necessarily caused by malfunction of Electrical/Electronic components (Addressed by ISO26262)
- SOTIF answers the question of "**How do you intend to behave**" by utilizing the PAS guidance on design, verification and validation.
- SOTIF intends to address **sensor** limitations (i.e. bad reflection, snow), decision **algorithms** (environment, location, highway construction etc.), misuses by drivers



SOTIF Normative sections

- Each containing a "Shall" objective implying the recommendations which follow.

3. New Vocabulary in addition to ISO 26262 reference.

4.SOTIF activities in the development process

5.Functional and System specification

6.SOTIF Hazard Identification and Evaluation. Risk of harm acceptable?

7.Identification and Evaluation of triggering events

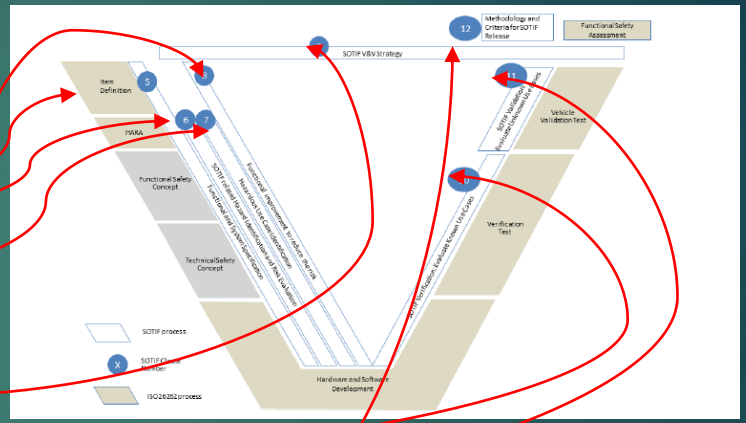
8.Functional modifications to reduce SOTIF risk

9.Definition of the Verification and Validation strategy

10.Verification of the SOTIF (Area 2) – *Known & Unsafe*

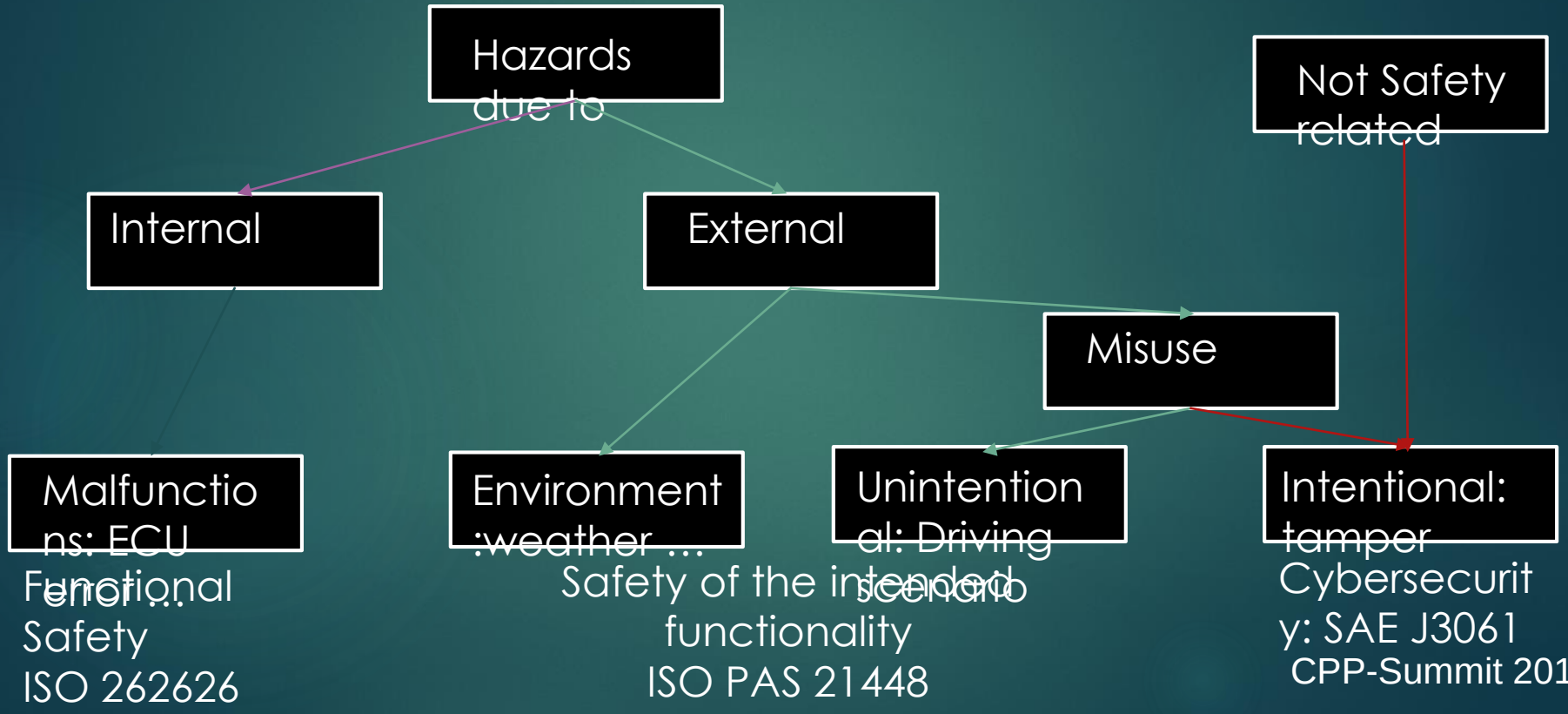
11.Validation of the SOTIF (Area 3) – *Unknown & Unsafe*

12.Methodology and Criteria for SOTIF release



Compliance may be claimed by listing each clause objective and documenting supporting evidence

Functional Safety vs SOTIF: depends on Root Cause

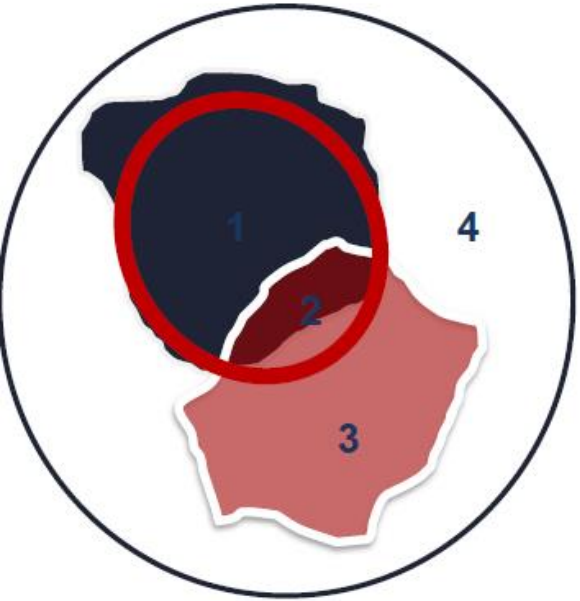


SOTIF Categorization of driving scenarios

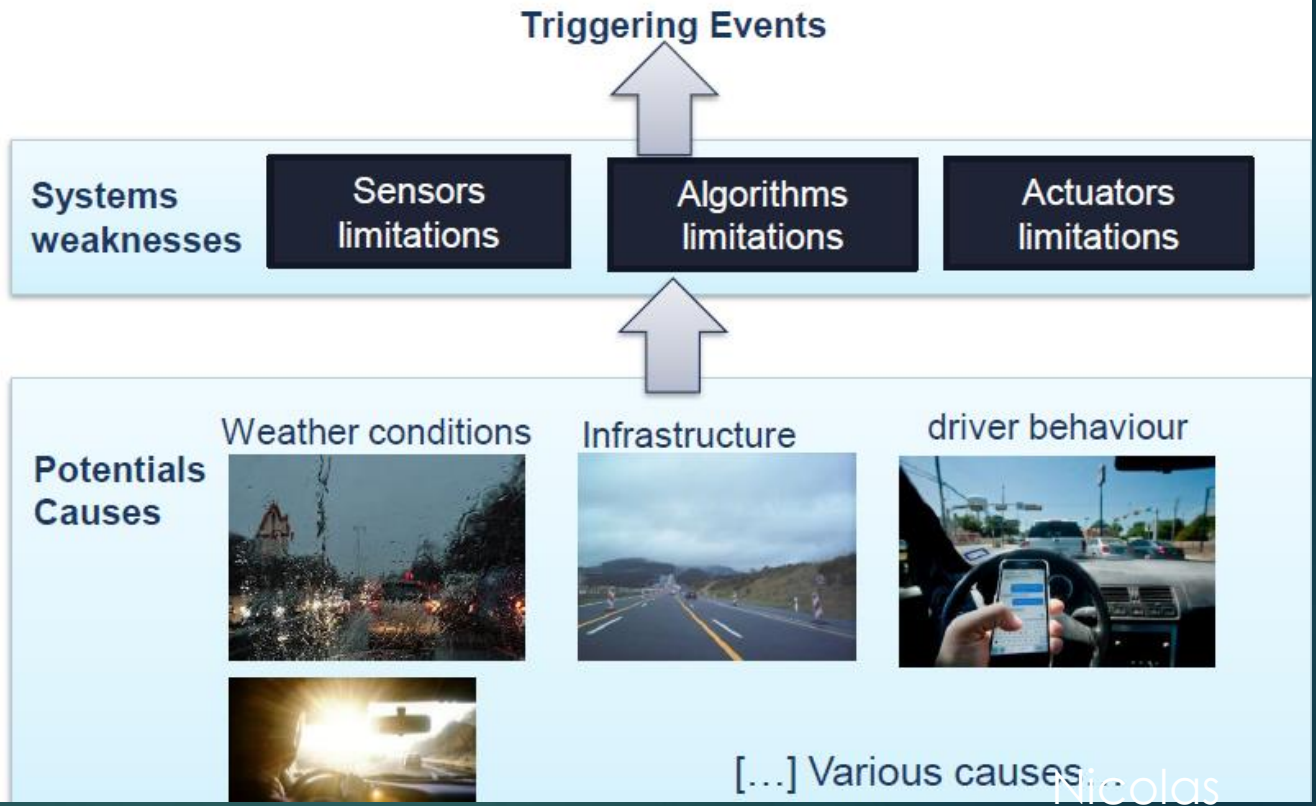
Example of an initial starting point of development

	Known	Unknown
Safe	**Area 1** Nominal behavior	**Area 4** System robustness
Potentially hazardous	**Area 2** Identified system limitations	**Area 3** “Black swans”

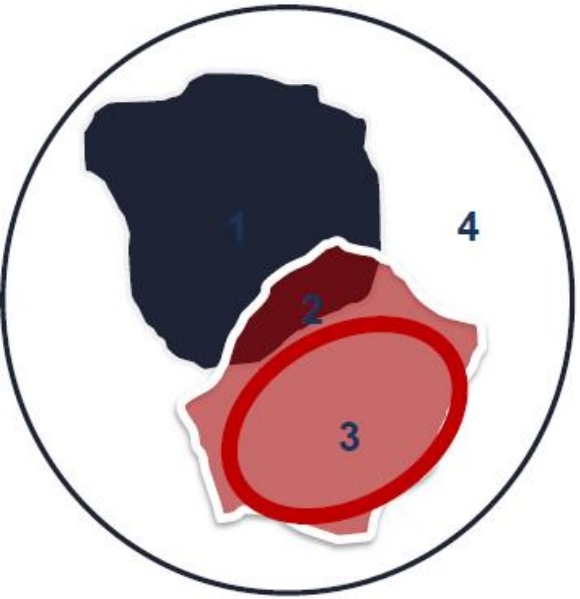
Known Safe (1) and Known Unsafe (2)



Example of an initial starting point of development



Unknown Safe (3) or Unsafe(4)?



Example of an initial starting point of development

Is this Lady crossing?

CSC 2125

CPP-Summit 2019

CSC 2125

CPP-Summit 2019

Traffic light in Toronto

On the way here (from Toronto)

Edge Cases (Unknown Unsafe (4))

CSC 2125

CPP-Summit 2019

Autonomous Vehicle Trap

CSC 2125

CPP-Summit 2019

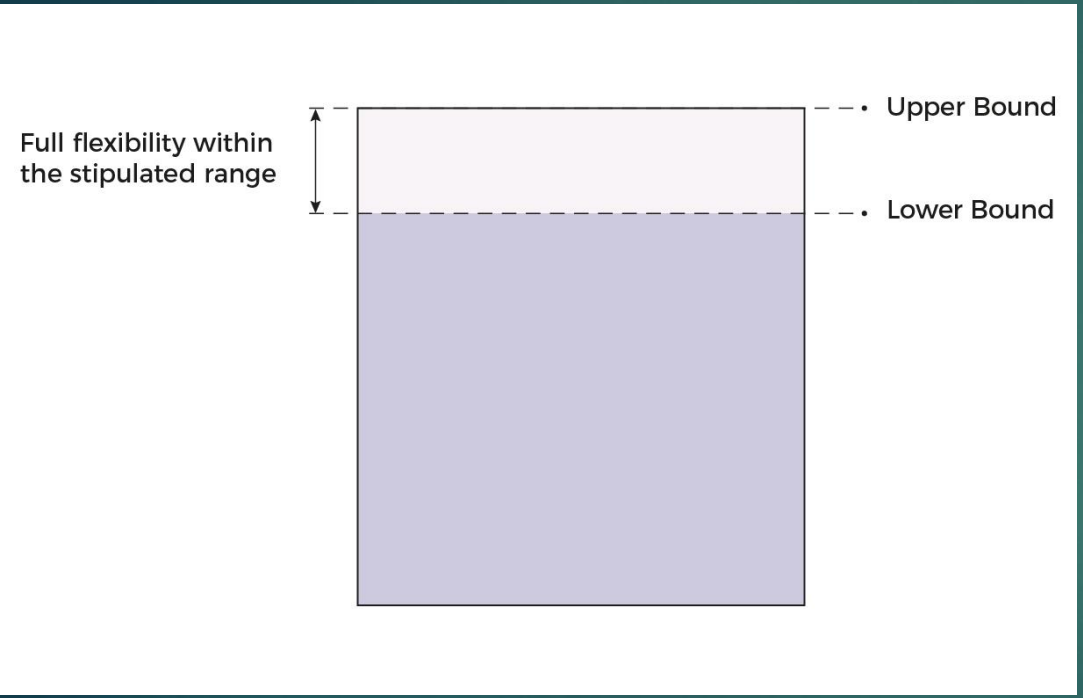
How to create safe software?

- Start with Education, and installing a Safety Culture/certification
- You have a rigorous Safe and secure process
- Identify defects early during development
- By Using coding guidelines
- And a Safe and Secure tool chain, or runtime, or API
- Root cause analysis on Safety Violations to learn to change
- These safety rules are well established for large hosted processors like CPU
- But can it be done for embedded processors, or CPU+GPU or MPSOC or AI processor combination?

Act 2

What is ISO26262, MISRA, Autosar, WG23, Core Guidelines …?

How do Safety guidelines differ from a Coding Guideline like Core Guideline?



Coding Guideline=upper Bound

Safety Guideline=lower bound

ISO 26262 –Static Analysis Verification Methods

ISO 26262 Table 9 – Methods for the verification of software unit design and implementation

Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a. Walk-through	++	+	0	0
1b. Inspection	+	++	++	++
1c. Semi-formal verification	+	+	++	++
1d. Formal verification	0	0	+	+
1e. Control flow analysis	+	+	++	++
1f. Data flow analysis	+	+	++	++
1g. Static code analysis	+	++	++	++
1h. Semantic code analysis	+	+	+	+

8.4.5

"The software unit design and implementation shall be verified in accordance with Clause 9, and by applying the verification methods listed in Table 9 to demonstrate:

...

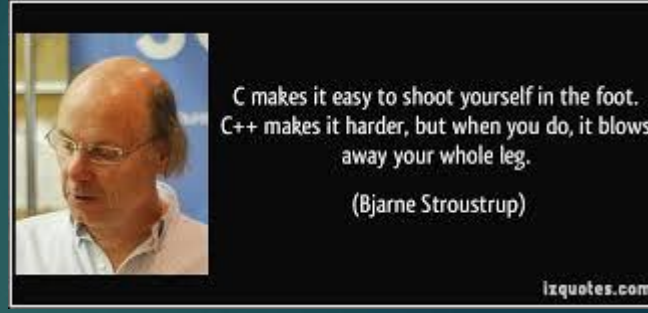
d) the compliance of the source code with the coding guidelines (see 5.5.3); and

e) the compatibility of the software unit implementations with the target hardware

Some unsafe part of C++

- Complex language
- Implementation variance
- Unintended programmer mistakes
- No run-time checking
- Strongly-typed but has loopholes

C++ hard problems



- Single exit single entry
- Exceptions
- Dynamic Memory
- Processes
- Unused variables/code
- Dead code
- Unreachable code
- Global Variables
- Dangling pointers
- Object lifetimes
- Binary verification
- Non Deterministic Behaviour
- Language Ambiguity
- Off-by-one error
- Loop control variables
- Dangling references
- Polymorphism

- Overflow
- Language Gotchas
- Resource/Memory leaks
- Fragmentation
- Type checked
- Template instantiation
- String termination
- Array index
- Arithmetic wrap around
- Shift operations
- Identifier reuse
- Recursion
- Liskov substitution/inheritance
- Deep vs shallow copy
- Overload vs override
- ...

Many Safety Critical APIs

- Misra C and C++:
- Autosar C++ Guidelines
- High Integrity C++
- WG23 Programming Vulnerabilities
- C++ Core Guidelines
- C++ Study Group 12 Vulnerabilities
- C Safe and Secure Study Group
- Carnegie Mellon Cert C and C++
- Joint Strike Fighter ++
- Common Weakness Enumeration

- Khronos Safety Critical Advisory Forum
- OpenCL/SYCL Safety Critical
- Vulkan Safety Critical
- JTC1/SC42 Machine Learning WG3 Trustworthiness
- TC22/SC32 SOTIF WG8 SOTIF

Many intended audience: Safety Critical APIs

- Misra: checkable rules only
- Autosar C++ Guidelines: a mix of meta guidelines and checkable rules
- High Integrity C++: for static checkers
- WG23 Programming Vulnerabilities: for team leads
- C++ Core Guidelines: a mix
- C++ Study Group 12 Vulnerabilities: for standards
- C Safe and Secure Study Group: for standards
- Carnegie Mellon Cert C and C++: a mix
- Joint Strike Fighter ++: checkable rules
- Common Weakness Enumeration: a mix

- Khronos Safety Critical Advisory Forum
- OpenCL/SYCL Safety Critical
- Vulkan Safety Critical
- JTC1/SC42 Machine Learning WG3 Trustworthiness
- ITC22/SC32 SOTIF WG8 SOTIF

Which one to choose and what is the difference?

- Safe but not C++11/14/17
  - Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program, 2005
    - With the help of Bjarne Stroustrup
  - **MISRA C++:2008 Guidelines for the use of the C++ language in critical systems, The Motor Industry Software Reliability Association, 2008**
    - **Continues to be the reference despite its age**
    - **For automated static analysis tools**
    - **Aimed for embedded domains**
- C++11/14/17 but not safe
  - High Integrity C++ Coding Standard Version 4.0, Programming Research Ltd, 2013
    - Some parallelism
  - Software Engineering Institute CERT C/C++ Coding Standard, Software Engineering Institute Division at Carnegie Mellon University, 2016
    - Most recent effort based on C 11 and C++ 14
  - C++ Core Guidelines, http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines, 2017
    - Most recent effort based on C++17
    - More style guide
    - No specific domains, also for static analysis and guidance
  - WG23 Vulnerabilities ISO for C, C++, Ada, Fortran, …
    - Guidelines for teamleads
    - Reviewed with each ISO C, C++, Ada, Fortran help

Comparing coding standards

Coding Standard	C++ Versions
Autosar	C++14
Misra	C++03 (working to C++17)
High Integrity CPP	C++11
JSF	C++03
C++ CG	C++11/14/17/20/latest
CERT C++	C++14

Pedigree

Coding Standard	Number of Rules	Number of rules in common with Autosar			% of rules in common
		Identical	Small Diff	Big Diff	
Misra C++	229	138	38	32	91%
High Integrity C++	155	0	99	25	80%
JSF ++	226	0	143	28	76%
C++ CG	412	0	174	49	54%
CERT C++	156	0	75	33	69%

Adherence to MISRA Standards

MISRA – Motor Industry Software Reliability Association

MISRA C

- ☑ 1998 - Guidelines for the use of the C language in vehicle based software
  - ☑ MISRA C:1998 (MISRA C1)
- ☑ 2004 - MISRA C:2004 Guidelines for the use of the C language in critical systems
  - ☑ MISRA C:2004 (MISRA C2)
- ☑ 2013 - MISRA C:2012 Guidelines for the use of the C language in critical systems
  - ☑ MISRA C:2012 (MISRA C3)
  - ☑ 159 rules of which 138 are statically enforceable

MISRA C++

- ☑ 2008 - Guidelines for the use of the C++ language in critical systems
  - ☑ 228 rules of which 219 are statically enforceable

C++ in Safety Critical

The Good

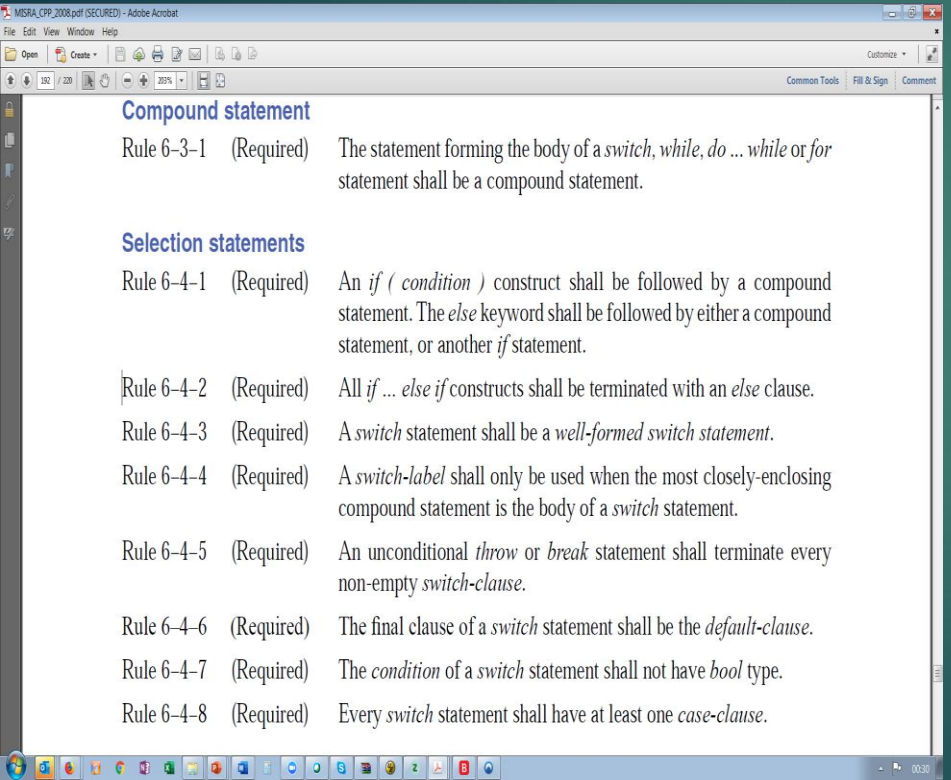
- RAII
- Templates
- Regular Types
- Exceptions/Error Codes
- C++11 Smart Pointers
- C++11/14 Lambdas
- C++17 Parallel Algorithm
- C++20 Contracts, Concepts

Loopholes fixed by tools

- Preprocessor
- Implementation-Defined, unspecified, and undefined behaviour
- Error-prone language constructs
- Type system
- Multiple Inheritance
- Dynamic Memory

Some Examples that need safety net

If else switch



MISRA_CPP_2008.pdf (SECURED) - Adobe Acrobat

File Edit View Window Help

Open Create Customize

192 / 220 283% Common Tools Fill & Sign Comment

Compound statement

Rule 6–3–1 (Required) The statement forming the body of a *switch*, *while*, *do ... while* or *for* statement shall be a compound statement.

Selection statements

Rule 6–4–1 (Required) An *if (condition)* construct shall be followed by a compound statement. The *else* keyword shall be followed by either a compound statement, or another *if* statement.

Rule 6–4–2 (Required) All *if ... else if* constructs shall be terminated with an *else* clause.

Rule 6–4–3 (Required) A *switch* statement shall be a *well-formed switch statement*.

Rule 6–4–4 (Required) A *switch-label* shall only be used when the most closely-enclosing compound statement is the body of a *switch* statement.

Rule 6–4–5 (Required) An unconditional *throw* or *break* statement shall terminate every non-empty *switch-clause*.

Rule 6–4–6 (Required) The final clause of a *switch* statement shall be the *default-clause*.

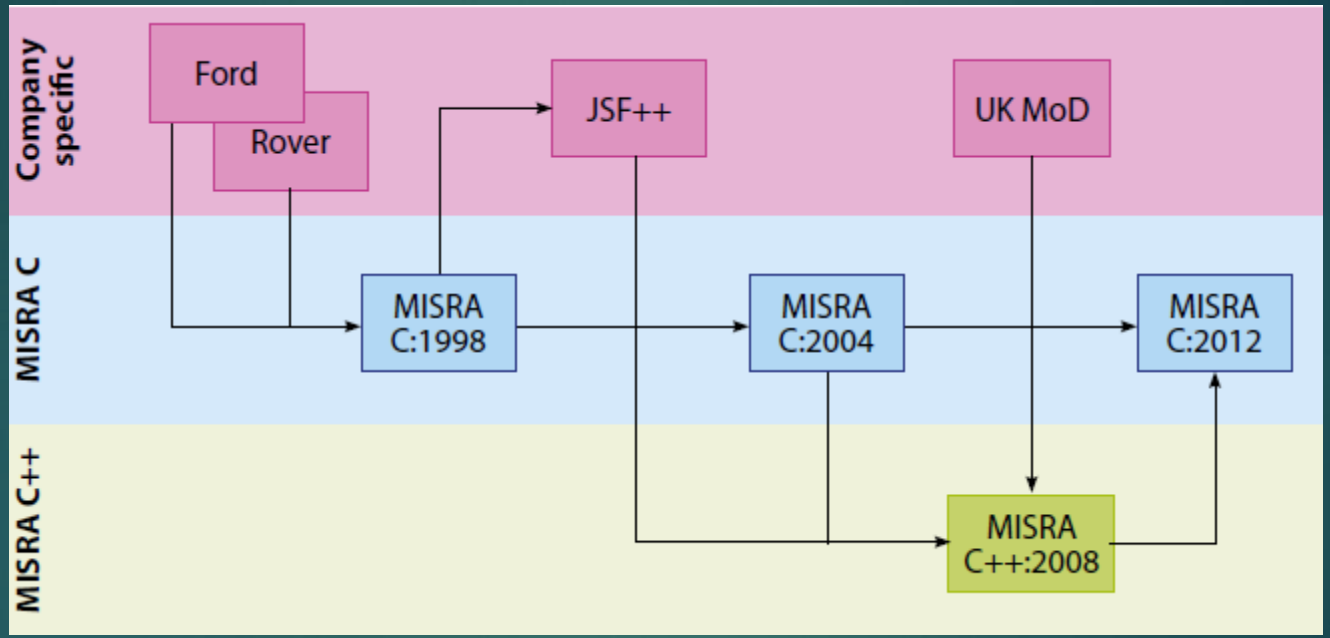
Rule 6–4–7 (Required) The *condition* of a *switch* statement shall not have *bool* type.

Rule 6–4–8 (Required) Every *switch* statement shall have at least one *case-clause*.

Type system

```
short i = 10000;
short j = 8;
int32 result = i * j;
```

A few related safety standards



C++ with Safety Critical API

C++98/03

Inheritance
Exception
Templates
STL

C++11

Auto
Lambda
Concurrency
Move
Futures

2008

C++14

2011

2015

C++17

Template deduction
Compile time if
Parallel STL

2017

C++20

Modules
Concept
Co-routine
Ranges

201

2021

C++ 2008

What will be in C++ 20

	Depends on	Current target (estimated, could slip)
Concepts		C++20 (adopted, including convenience syntax)
Contracts		C++20 (adopted)
Ranges		C++20 (adopted)
Coroutines		C++20
Modules		C++20
Reflection		TS in C++20 timeframe, IS in C++23
Executors		Lite in C++20 timeframe, Full in C++23
Networking	Executors, and possibly Coroutines	C++23
future.then, async2	Executors	

Why does MISRA 2008 forbids Dynamic memory?

C++98 /03

C++11

C++14

C++17

C++20

auto:_ptr
New/delete

Auto_ptr
auto_ptr deprecated
Unique_ptr
Shared_ptr
Weak_ptr

Auto_ptr removed

atomic<shared_ptr<T>>
Atomic<weak_ptr<T>>

2008 2011 2015 2017 201 202 1

MISRA

C++ 2008

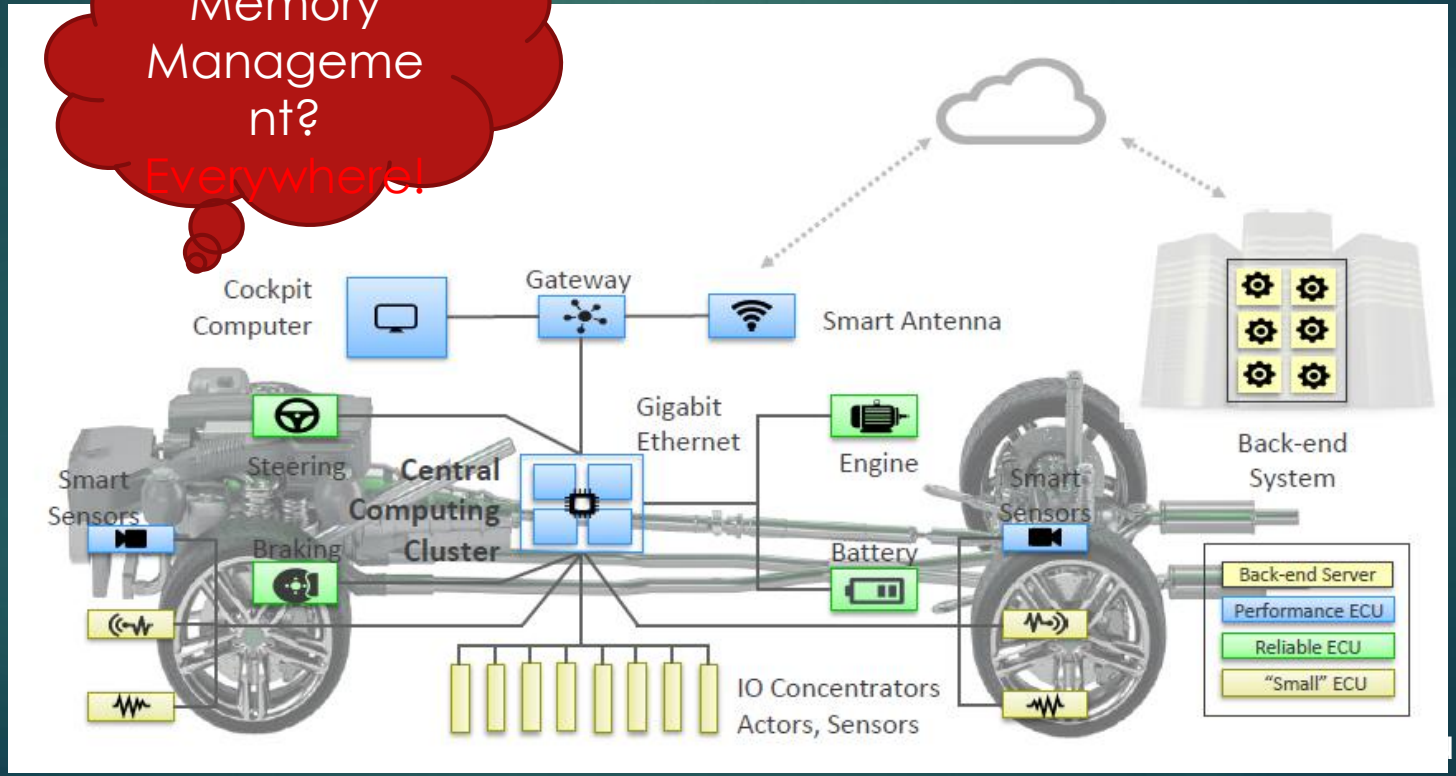
AUTOSAR

MISRA AUTOSAR

Adaptive Autosar and Memory Management

Where is the Memory Management?

Everywhere!



Dynamic Memory Management

MISRA C++ Rule 18-4-1 (Required) Dynamic heap memory allocation shall not be used.

AUTOSAR Rule A18-5-5 (required, toolchain, partially automated)

Memory management functions shall ensure the following: (a) deterministic

behavior resulting with the existence of worst-case execution time, (b)

avoiding memory fragmentation, (c) avoid running out of memory, (d)

avoiding mismatched allocations or deallocations, (e) no dependence on

non-deterministic calls to kernel.

Why do we need dynamic memory?

- Size of data only known at runtime
- Lifetime of data independent from object lifetimes
- Sharing/transmitting data across threads (**promise** – **future**)
- Type erasure, e.g. **std::function**
- Some language/library features use dynamic memory implicitly
  - Exception handling
  - Containers (can be customized)
  - **std::function** (customization deprecated in C++17)
- *Few programs manage to avoid some of these facilities*

Dynamic memory issues

- Memory leaks
- Memory fragmentation
- Non-deterministic execution time
- Out of memory

Memory leaks

- Use RAII
- Do not call **new** and **delete** explicitly

- Easy to achieve with
- **std::vector**, **std::string**, and other containers
- **std::unique_ptr**, **std::make_unique**
- **std::shared_ptr**, **std::make_shared**

Memory Fragmentation

- Allocator must minimize fragmentation
- Usually OS **malloc/free** is pretty good at that
- Techniques/implementations for custom allocators are available

Non-deterministic execution time

- Allocators must guarantee deterministic WCET

- Either
  - OS **malloc/free** makes this guarantee
- or
  - Roll your own: **malloc/free**, **new/delete**, custom allocators
- or
  - Allocate/deallocate only during non-realtime phases of the program

Out of Memory

- Define maximum memory needs, use pre-allocated storage

C++ Exceptions: a brief history

- integration of exception handling with constructor/destructor resource managements in the form of RAII is one of the major achievements of C++
- applications for which the use of exceptions was unsuitable
- problems for which error codes provide no good solution:

Exception Safety

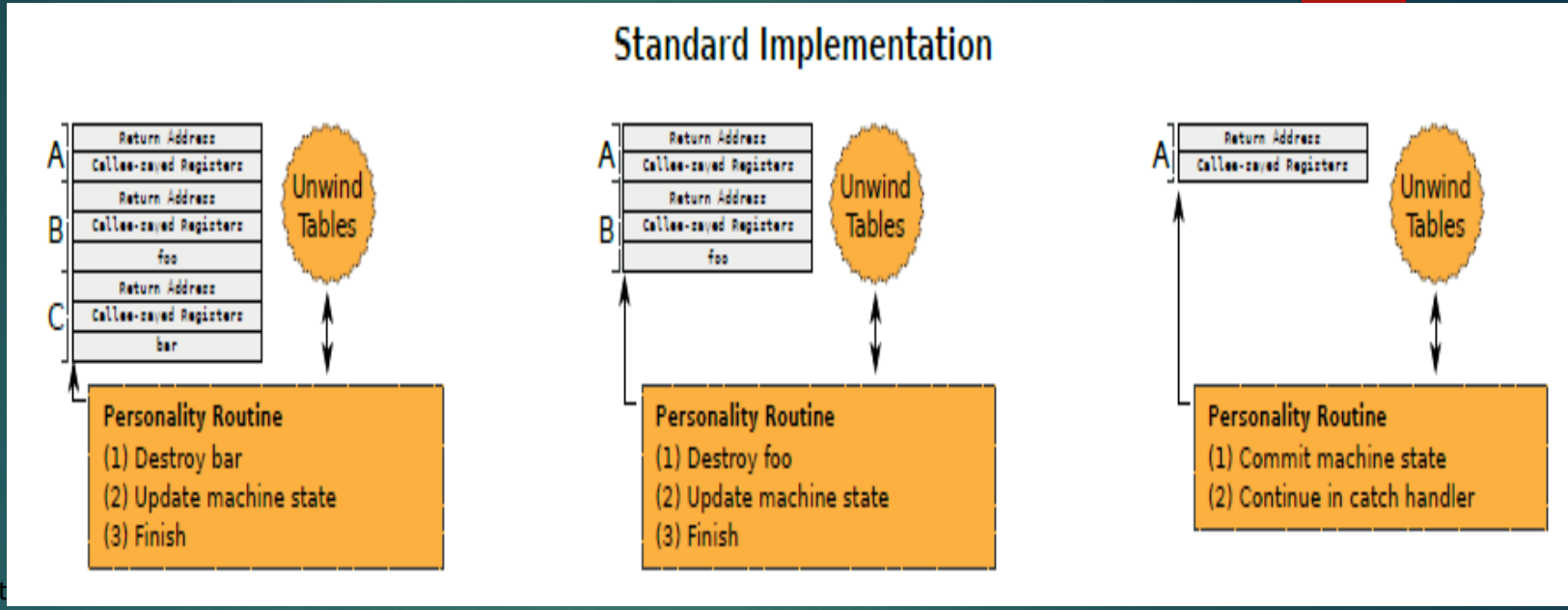
- Why is it disallowed in many coding standards
  1. Perceived violation of zero overhead principle
     - Does it really? Or is it that we have sped up other code?
     - C++ exceptions designed for the 1:100 case
  2. Violates Determinism principle
     - C++ Exceptions were never meant for HARD real-time code
- Why don’t people use exceptions?
  - Many, but mainly current implementation is BAD!
  - Prioritize performance for when there is no exception

Cost of error handling

- Error propagation costs
- Error recognition costs
- Error identification costs

Problems with current implementations

```
void C() {
  Bar bar;
  throw 0;
}
void B() {
  Foo foo;
  C();
}
void A() {
  try {
  B();
  } catch (int
  // Catch exception
  }
}
```



Autosar advice on C++ exceptions

- Exceptions are permitted conditionally, *assuming the following are addressed:*
  - Worst time execution time (WCET):
  - Follow Java's checked und unchecked exception paradigm (via Doxygen, tooling)
  - All operations provide either a basic exception guarantee (no resource leak, no object invariant violated upon throw), a strong guarantee (function succeeds or no effect), or a nothrow guarantee.
  - Compiler supplier or project to perform analysis for typical failure cases: correct stack unwinding, exception not thrown, other exception thrown, wrong catch activated, memory not available while handling exception.
  - Prefer last-ditch *"catch all"* for all exceptions (incl. those from external libraries) to avoid the case of uncaught exceptions (because behavior is unclear: may unwind stack or not before termination)
  - If a function's noexcept specification can not be determined, then always declare it to be noexcept(false) (otherwise risk of calling std::terminate()). Conversely, a known non-throwing function shall always have a noexcept specification.

What about deterministic exception

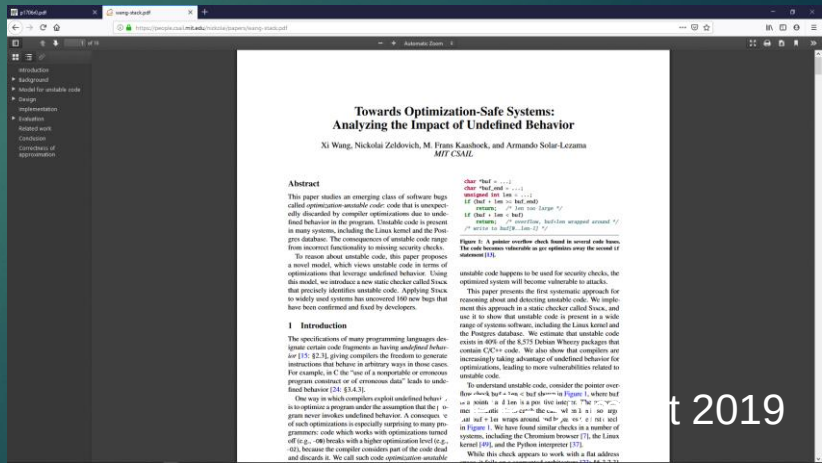
- Herbception is a promising start but …
  - Use of registers
  - All exceptions be the same type
- What we want:
  - Maintain existing exception model
  - A new way to store and match exception at runtime, with no performance impact when exceptions are not thrown
  - But also scales well with exception depth, deterministically in size and time

Act 3

The two prong approach:
C++'s SG12 and MISRA C++

WG21 SG12 needs to reduce vulnerabilities

- Started as a group to handle optimizations that can cause unsafe code
  - Towards Optimization-Safe systems
- Expanded in 2017 Toronto meeting to help with other Safety Critical documents:
  - WG23 Vulnerabilities
  - MISRA
  - AUTOSAR
- Autosar coding guideline folded


Towards Optimization-Safe Systems:
Analyzing the Impact of Undefined Behavior

Xi Wang, Nickolai Zeldovich, M. Frans Kaashoek, and Armando Solar-Lezama
MIT CSAIL

Abstract

This paper studies an emerging class of software bugs called *optimization-unstable code*: code that is unexpectedly discarded by compiler optimizations due to undefined behavior in the program. Unstable code is present in many systems, including the Linux kernel and the Postgres database. The consequences of unstable code range from incorrect functionality to missing security checks.

To reason about unstable code, this paper proposes a novel model, which views unstable code in terms of optimizations that leverage undefined behavior. Using this model, we introduce a new static checker called Stack that precisely identifies unstable code. Applying Stack to widely used systems has uncovered 160 new bugs that have been confirmed and fixed by developers.

1 Introduction

The specifications of many programming languages designate certain code fragments as having *undefined behavior* [15: §2.3], giving compilers the freedom to generate instructions that behave in arbitrary ways in those cases. For example, in C the "use of a nonportable or erroneous program construct or of erroneous data" leads to undefined behavior [24: §3.4.3].

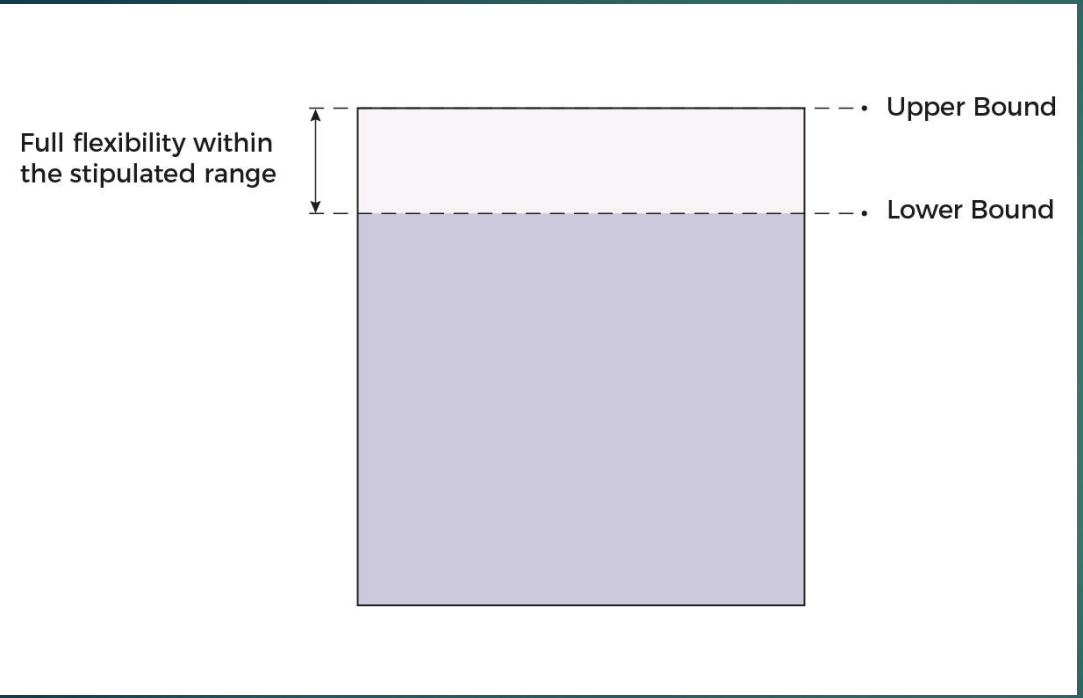
Additional WG21 SG12 charter

- SC 22/WG 23 and SC 22/WG 21/SG 12 agree to work together to develop TR 24772-9 Programming Language Vulnerabilities in C++.

Emerging cpp safety Critical proto study group

- Work on what we can change to C++ Standard proposals to support safety
- Analyze existing C++ features from Safety point of view

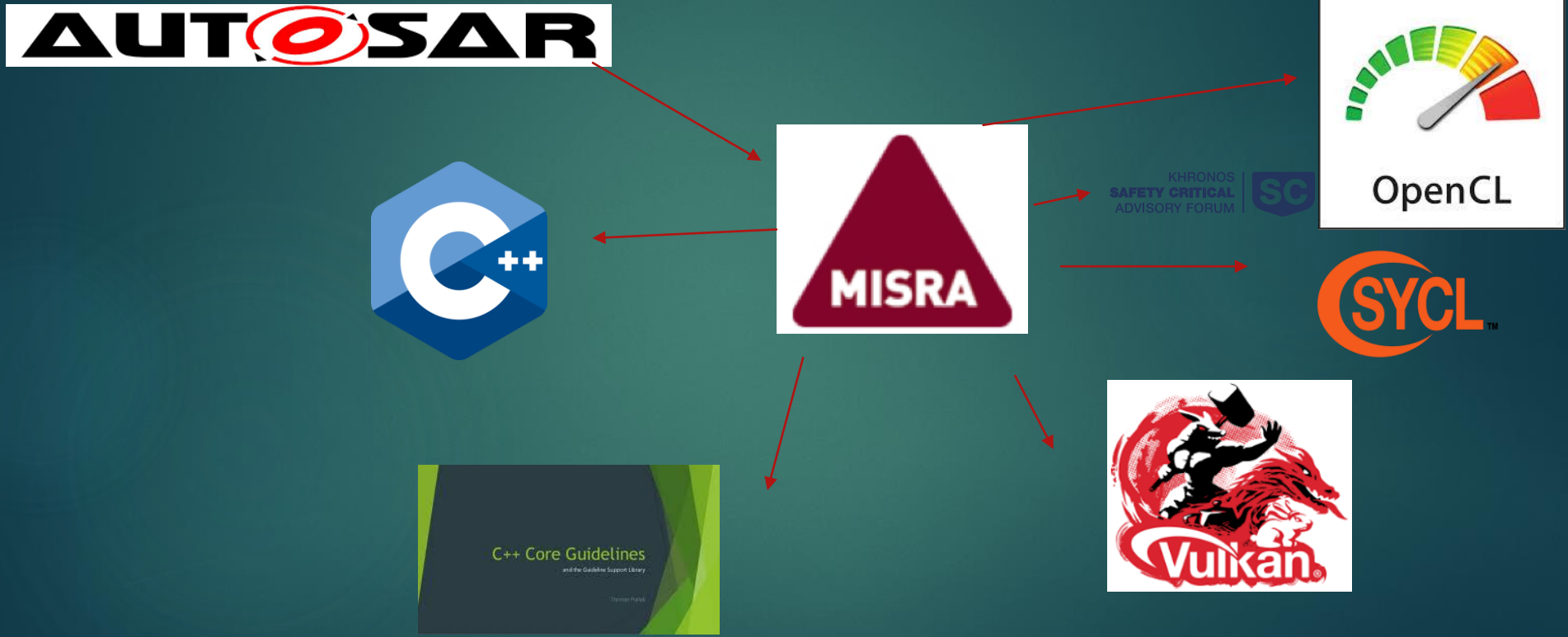
How do Safety guidelines differ from a Coding Guideline like Core Guideline?



Coding Guideline=upper Bound (everything above is elegant great code)

Safety Guideline=lower bound (everything below is unsafe, everything above is safe but may not be elegant great code)

The future: towards full Safety Critical



What is still missing?

So far most only deal with Sequential code

Very few deal with Parallel code

Even fewer deal with Concurrent, event driven code

None deal with Heterogeneous dispatch code

MISRAC++ParallelConcurrencyHeteroRulesOverview

File Edit View Insert Format Tools Add-ons Help Last edit was made seconds ago by Michael Wong

Share

Outline

Note: this is an early draft WIP. It's known to be incomplet and incorrekt, and it has lots of badformatting.

Table of Content

Table of contents

Heading numbers format 1.2.3

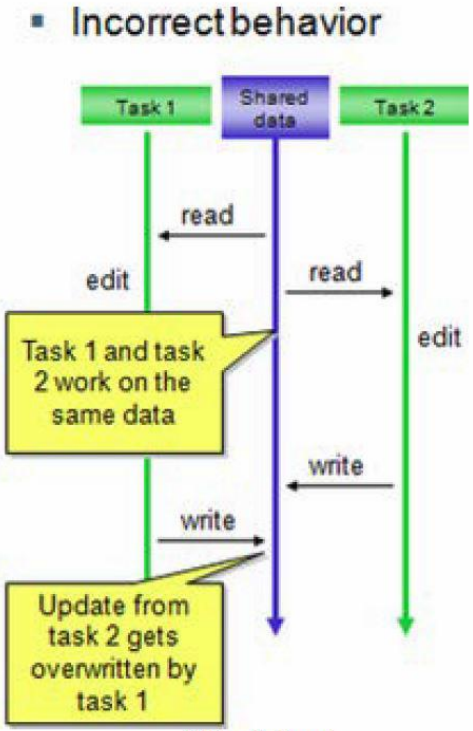
Display until level 6

Always open for this document

Powered by Josué MOËNS @ LumApps

Concurrency Bugs/Race conditions

- One CPU can have many tasks (e.g., with Real Time Operating System)
  - Tasks take turns, sharing the CPU and memory ("multi-tasking")
- Concurrency defects often come from incorrect data sharing
  - One way to fix this is to **disable interrupts** before reading to ensure one task reads/writes at a time
  - Defects may be due to subtle timing differences, and are often **difficult to reproduce**



[Wind River]

Audi Unintended Acceleration

- NASA identified a specific concurrency defect

  This rule is based on the fact that equal priority tasks cannot interrupt each other. Both can still be interrupted by a higher priority task. If because of this interruption the second task does not complete, and the first task restarts in the next time interval, it could still overwrite the result of the interrupted second task.

  [NASA App. A pp. 33-34]

- Nested scheduler unlocks [Bookout 2013-10-14PM 21:10-22:1]
- Shared global variables not all "volatile" [NASA App. A pp. 33-34]
- Shared globals not always access with interrupts masked

[NASA App. A pp. 33-34]

If two tasks running at different priority levels access the same data, then the lower priority task must use interrupt masking to protect against interference from the higher-priority task.

This rule is not always followed in the code. In a few cases, the lower priority task merely sets a flag before entering its critical section and the higher priority task checks this flag before accessing the same data.

There are cases in the code where tasks of different priority levels (e.g., levels 14, 12, and 4) access the same global variables without using interrupt masks (pattern: read, store local copy, update, write new value). An example is the use of variable *s2s_eafsfb_gaind*, which is declared as a *non*-volatile static variable. This use would appear to be in violation of coding rule 651.

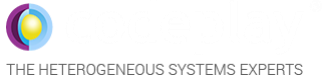
...

Despite the rigor in the use of the *volatile* qualifier on *constant* data, other shared global variables are not always declared *volatile*. The team counted **11,528** non-constant, shared global variables in the code.

There are only **865** uses of interrupt masking in the code, in **194** different source files. This indicates that access to global variables is not always done under protection of interrupt masks.

Many Safety Standards

- Which one to use?
- Use MISRA for now
- What process to adopt?
- Follow one of the safety process e.g. automotive, airline

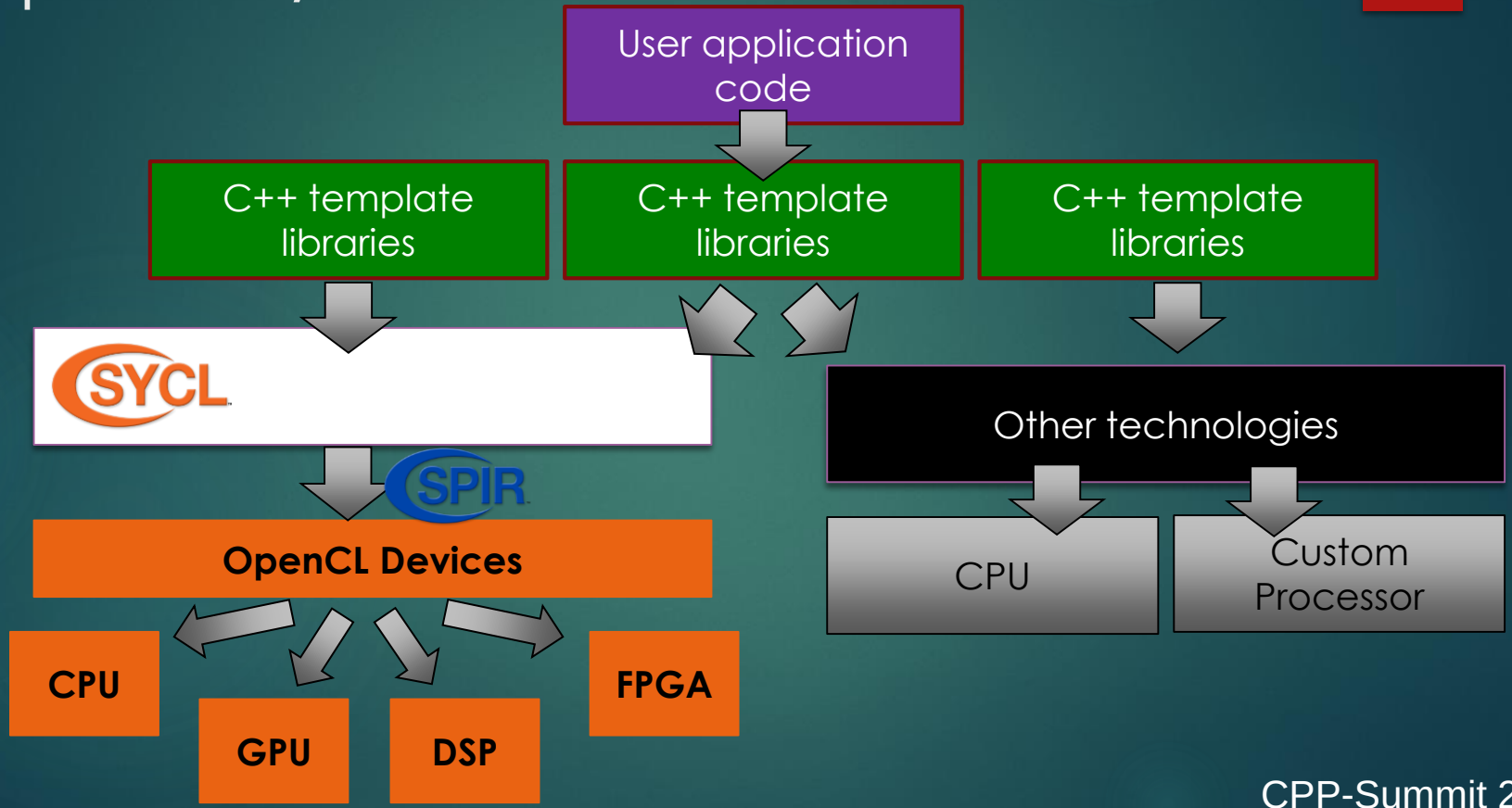


Oh one more thing!
SYCL and Codeplay Automotive AI Acceleration Tools & Roadmap

What is SYCL for?

- Modern C++ lets us separate the what from the how :
  - We want to separate **what** the user wants to do: *science, computer vision, AI …*
  - And enable the **how** to be: *run fast on an OpenCL device*
- Modern C++ supports and encourages this separation
- See the talk Tuesday 3:15 Efficient GPU Programming with Modern C++
- Tutorial this weekend: Parallelism in Modern C++: From CPU to GPU

OpenCL / SYCL Stack



Automotive AI Acceleration Tools & Roadmap

- ComputeAorta & ComputeCpp for Renesas R-Car: delivers OpenCL and SYCL with automotive optimizations for Renesas R-Car platform
- Developer guides for performance and porting from CUDA
- Performance profiler
- SYCL for PowerVR with ComputeCpp

Codeplay exclusive technology

ComputeSuite

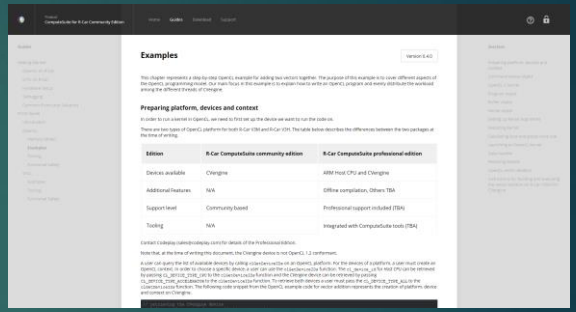
- Functional Safety experts for automotive AI acceleration
- Can bring AI algorithms to high performance on R-Car, using open-standards and easy to maintain
- Can bring OpenCL and SYCL with functional safety to new AI accelerators

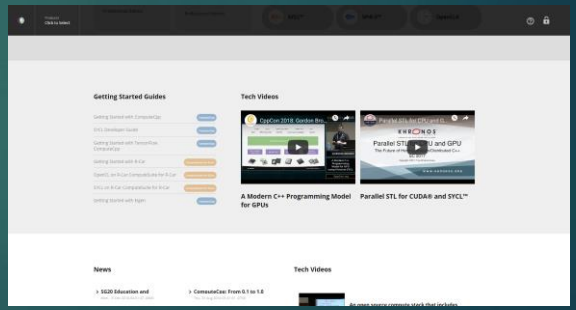
Codeplay specialist services

R-Car OpenCL documentation

- Getting started guides
- Performance optimization guides
- Online Stack-overflow type support for free Community Edition
- Private support for Professional Edition
- Sample code
- Constantly updated with new releases and new optimization experience

http://developer.codeplay.com





SYCL in automotive

- SYCL is a royalty-free industry standard from Khronos, with the same IP position as OpenCL
- C++ programming model for HPC and AI
- Codeplay's ComputeCpp is a conformant implementation of the SYCL standard
- Equivalent programming model to CUDA, but standard: easy to port from CUDA to SYCL

ComputeCpp™

R-Car V3M: Front Camera

R-Car V3H: Advanced Vision

PowerVR by Imagination

mali Visual Computing by ARM

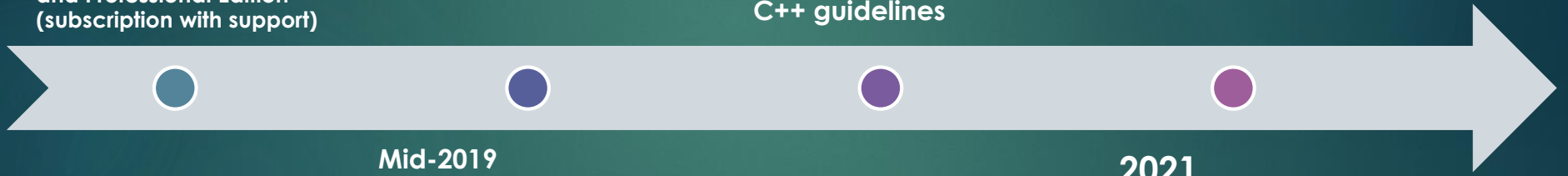
Also supports: Intel CPU & GPU, and AMD GPU

Intel	hipSYCL	triSYCL
• Open-source SYCL implementation in development for Intel processors • Basic features, right now • Requires OpenCL 2.1	• Open-source SYCL implementation for AMD and NVIDIA GPUs • Relies on CUDA or AMD's HIP implementation providing compiler support	• First open-source SYCL implementation from Xilinx • Also supported by RedHat • Limited capabilities

Codeplay Automotive Roadmap

Available today

- R-Car H3, V3M, V3H: preconformant OpenCL/SYCL
- CVengine for programmable kernels
- CUDA-to-SYCL porting guide
- Performance optimization guides for R-Car
- SYCL-BLAS, SYCL-DNN on CVengine (limited optimization)
- 100% SYCL SDK passing on R-Car
- Community Edition (free) and Professional Edition (subscription with support)



Mid-2019

- PowerVR GPU support

2020

- MISRA C++ - Needs a new, updated MISRA C++ to support SYCL
- Adaptive AUTOSAR C++ guidelines

2021

- ISO 26262 ASIL B

Benchmarking phase

Production development phase

SYCL Present and Future (May Change)

C++98 C++11 C++14 C++17 C++20

SYCL 1.2
C++11 Single source programming

SYCL 1.2.1
C++11 Single source programming

SYCL 2019
C++17 Single source programming

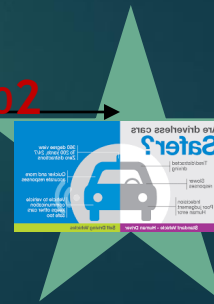
SYCL 2021
C++20 Single source programming

1998 2011 2015 2017 2019 2021

MISRA C++ 2008

AUTOSAR

MISRA AUTOSAR



Conclusion

- Misra C/C++ remains the defacto coding Standard for automotive and the base others build upon
- In 2019, Misra integrated Autosar updates for next Misra with Misra 2008 rules updated for C++17
- But there is still work needed to add additional features from C++11/14/17
- I lead the effort in Misra to add Parallelism Concurrency and heterogeneous support
- Soon SYCL will integrate this combined Misra/Autosar for Safety Critical application

The Dream

SYCL Ecosystem

- ComputeCpp - https://codeplay.com/products/computesuite/computecpp
- triSYCL - https://github.com/triSYCL/triSYCL
- SYCL - http://sycl.tech
- SYCL ParallelSTL - https://github.com/KhronosGroup/SyclParallelSTL
- VisionCpp - https://github.com/codeplaysoftware/visioncpp
- SYCL-BLAS - https://github.com/codeplaysoftware/sycl-blas
- TensorFlow-SYCL - https://github.com/codeplaysoftware/tensorflow
- Eigen http://eigen.tuxfamily.org

Eigen Linear Algebra Library

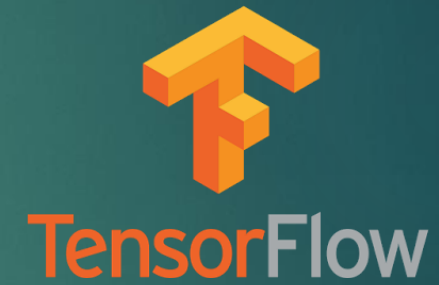
- SYCL backend in mainline
- Focused on Tensor support, providing support for machine learning/CNNs
- Equivalent coverage to CUDA
- Working on optimization for various hardware architectures (CPU, desktop and mobile GPUs)

https://bitbucket.org/eigen/eigen/

TensorFlow

- SYCL backend support for all major CNN operations
- Complete coverage for major image recognition networks
  - GoogLeNet, Inception-v2, Inception-v3, ResNet, ….
- Ongoing work to reach 100% operator coverage and optimization for various hardware architectures (CPU, desktop and mobile GPUs)

https://github.com/tensorflow/tensorflow



SYCL Ecosystem

- Single-source heterogeneous programming using STANDARD C++
  - Use C++ templates and lambda functions for host & device code
  - Layered over OpenCL
- Fast and powerful path for bring C++ apps and libraries to OpenCL
  - C++ Kernel Fusion - better performance on complex software than hand-coding
  - Halide, Eigen, Boost.Compute, SYCLBLAS, SYCL Eigen, SYCL TensorFlow, SYCL GTX
  - Clang, triSYCL, ComputeCpp, VisionCpp, Compute
- More information at https://sycl.tech

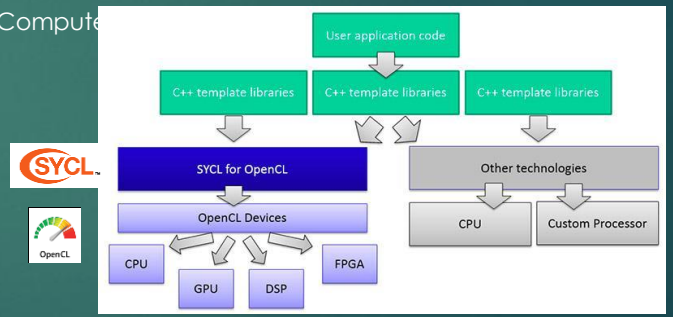
Developer Choice
The development of the two specifications are aligned so code can be easily shared between the two approaches

C++ Kernel Language
Low Level Control
'GPGPU'-style separation of device-side kernel source code and host code

OpenCL

Single-source C++
Programmer Familiarity
Approach also taken by C++ AMP and OpenMP

SYCL



Codeplay

Standards bodies

- HSA Foundation: Chair of software group, spec editor of runtime and debugging
- Khronos: chair & spec editor of SYCL. Contributors to OpenCL, Safety Critical, Vulkan
- ISO C++: Chair of Low Latency, Embedded WG; Editor of SG1 Concurrency TS
- EEMBC: members

Research

- Members of EU research consortiums: PEPPHER, LPGPU, LPGPU2, CARP
- Sponsorship of PhDs and EngDs for heterogeneous programming: HSA, FPGAs, ray-tracing
- Collaborations with academics
- Members of HiPEAC

Open source

- HSA LLDB Debugger
- SPIR-V tools
- RenderScript debugger in AOSP
- LLDB for Qualcomm Hexagon
- TensorFlow for OpenCL
- C++ 17 Parallel STL for SYCL
- VisionCpp: C++ performance-portable programming model for vision

Presentations

- Building an LLVM back-end
- Creating an SPMD Vectorizer for OpenCL with LLVM
- Challenges of Mixed-Width Vector Code Gen & Scheduling in LLVM
- C++ on Accelerators: Supporting Single-Source SYCL and HSA
- LLDB Tutorial: Adding debugger support for your target

Company

- Based in Edinburgh, Scotland
- 57 staff, mostly engineering
- License and customize technologies for semiconductor companies
- ComputeAorta and ComputeCpp: implementations of OpenCL, Vulkan and SYCL
- 15+ years of experience in heterogeneous systems tools

Codeplay build the software platforms that deliver massive performance

What our ComputeCpp users say about us

Benoit Steiner – Google TensorFlow engineer

"We at Google have been working closely with Luke and his Codeplay colleagues on this project for almost 12 months now. Codeplay's contribution to this effort has been tremendous, so we felt that we should let them take the lead when it comes down to communicating updates related to OpenCL. … we are planning to merge the work that has been done so far… we want to put together a comprehensive test infrastructure"

ONERA

"We work with royalty-free SYCL because it is hardware vendor agnostic, single-source C++ programming model without platform specific keywords. This will allow us to easily work with any heterogeneous processor solutions using OpenCL to develop our complex algorithms and ensure future compatibility"

Hartmut Kaiser - HPX

"My team and I are working with Codeplay's ComputeCpp for almost a year now and they have resolved every issue in a timely manner, while demonstrating that this technology can work with the most complex C++ template code. I am happy to say that the combination of Codeplay's SYCL implementation with our HPX runtime system has turned out to be a very capable basis for Building a Heterogeneous Computing Model for the C++ Standard using high-level abstractions."

WIGNER Research Centre for Physics

It was a great pleasure this week for us, that Codeplay released the ComputeCpp project for the wider audience. We've been waiting for this moment and keeping our colleagues and students in constant rally and excitement. We'd like to build on this opportunity to increase the awareness of this technology by providing sample codes and talks to potential users. We're going to give a lecture series on modern scientific programming providing field specific examples."

Further information

- OpenCL https://www.khronos.org/opencl/
- OpenVX https://www.khronos.org/openvx/
- HSA http://www.hsafoundation.com/
- SYCL http://sycl.tech
- OpenCV http://opencv.org/
- Halide http://halide-lang.org/
- VisionCpp https://github.com/codeplaysoftware/visioncpp

Community Edition

Available now for free!

Visit:

computecpp.codeplay.com

ComputeCpp™

- Open source SYCL projects:
  - ComputeCpp SDK - Collection of sample code and integration tools
  - SYCL ParallelSTL – SYCL based implementation of the parallel algorithms
  - VisionCpp – Compile-time embedded DSL for image processing
  - Eigen C++ Template Library – Compile-time library for machine learning

All of this and more at: http://sycl.tech

Thanks