CoreAVI

INNOVATING A SAFER TOMORROW

codeplay

Enabling AI & HPC To Be Open Safe & Accessible To All

Exploring SYCL SC

IWOCL 2022

Presented by:

Ken Wenger

www.coreavi.com

Verena Beckham

www.codeplay.com

Date: 5/05/2022

Welcome to IWOCL 2022

Ken Wenger, Senior Director Research & Innovation, CoreAVI

- CoreAVI: Providing safety critical GPU acceleration platforms using open standards
- Long history with Khronos
  - Main contributor to OpenGL® SC 2.0
  - Involved in OpenCL™ early on
  - Main contributor to Vulkan® SC
  - Chaired Vulkan SC working group

- Background in safety critical GPU driver development (graphics and compute)
- Contributing member of OpenGL SC 2.0 and Vulkan SC Working Group

- In my role now:
  - Research the role of functional safety, and determinism in Machine Learning and general computing platforms
  - Design and lead implementation of CoreAVI's Safe AI platform

Welcome to IWOCL 2022

Verena Beckham, VP of Safety Engineering, Codeplay

- Codeplay: Enabling heterogeneous computing using open standards
- Long history with Khronos
  - Involved in OpenCL early on
  - Main contributor to SYCL 1.2, developed first conformant implementation
  - Chaired SYCL WG since the beginning

- We started supporting heterogeneous compute for computer games
- Then mobile phones
- Now high-performance compute
- We want to unlock heterogeneous computing for safety-critical industries as well

Agenda

- Background on the Industry
- Use Cases for Compute/ML
- Non-SC tools and API availability
- SC tools and API availability
- Why SYCL SC
- SYCL SC in the Ecosystem
- SYCL SC Considerations
- The SYCL SC Exploratory Forum

Automotive Software

- There is a surprising amount of software in a car!
  - Controlling the engine
  - Controlling the breaking
  - Controlling the lighting
  - …
- Now, software is starting to determine the value of the car
  - Automatic wind screen wipers, automatic lights
  - “Advanced Driver Assistance” (ADAS) features
    - adaptive cruise control
    - automatic emergency breaking
    - self-parking
    - …
- Holy grail: Autonomous vehicles (AV)
  - This means: AI

Automotive Software is Changing

Traditional	Modern
Simple C/C++ algorithms / assembly / automatically generated from a model	Complex, large code bases, high-level frameworks
SW mostly rewritten for every new HW	Portability becomes important
Many small ECUs	A few central, large processing units
Simple “embedded” HW (homogeneous, single core)	Heterogeneous, multi-core, shared
Different source code / company / framework per ECU	Shared programming platforms

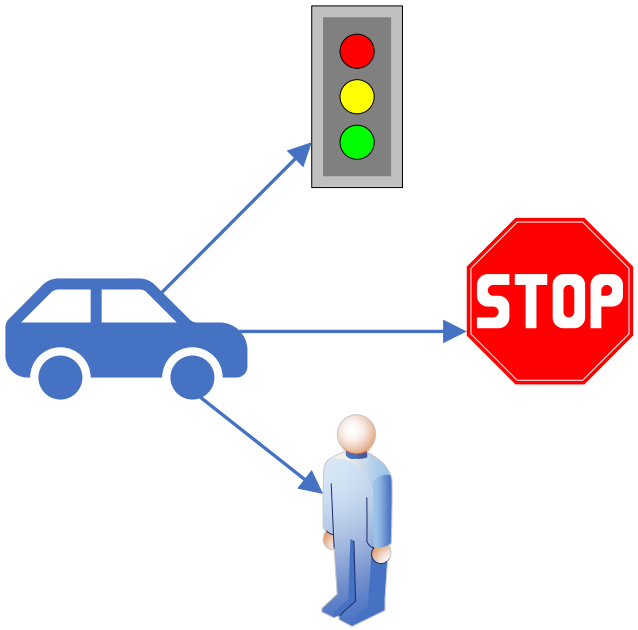
Automotive Software

- Software is safety-critical
  - Software needs to be sufficiently free of bugs
  - If a bug is encountered it still needs to be safe
  - Software needs to cover all use cases
- Standards to ensure this, e.g.
  - ISO 26262 – Functional Safety
  - ISO 21448 – Safety of the Intended Functionality
- ISO 26262 includes guidelines:
  - How to write & track your requirements
  - How to design your software
  - How to (not) write your code
  - How to test your code
- MISRA® guidelines cover a subset of ISO 26262 coding guidelines
  - Guidelines for C - 1998
  - Guidelines for C++ - 2008
  - New guidelines for C++ (in development) – 202x

Other Industries

- Avionics
  - More conservative than automotive
  - Space flight sometimes mission-critical instead
- Nuclear
- Rail
- Medical

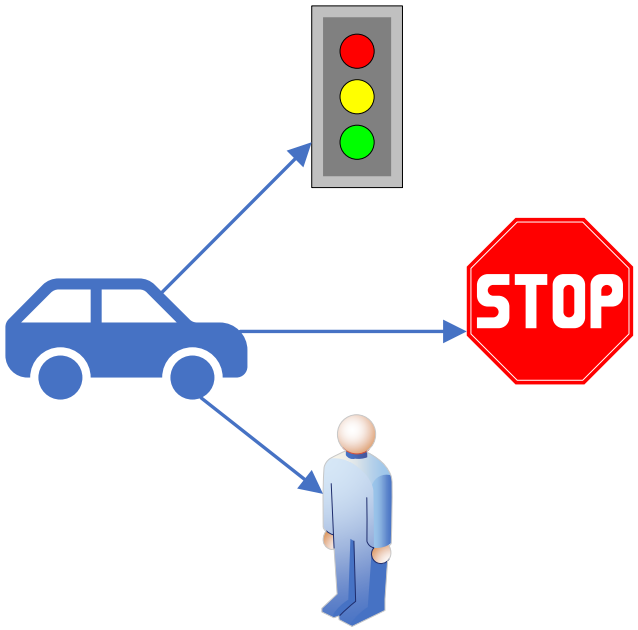
Use Cases



- **Sensor Fusion**
  - Camera
    - Image Processing
    - Computer Vision
    - Deep Learning
  - Radar
    - Signal Processing
    - Deep Learning
  - Combine input sources
    - Compute decision

- **Tools**
  - TensorFlow
  - OpenCV
  - Pytorch
  - Python Sci-Kit
  - Vendor-specific APIs and frameworks
  - SYCL
  - OpenCL
  - Vulkan

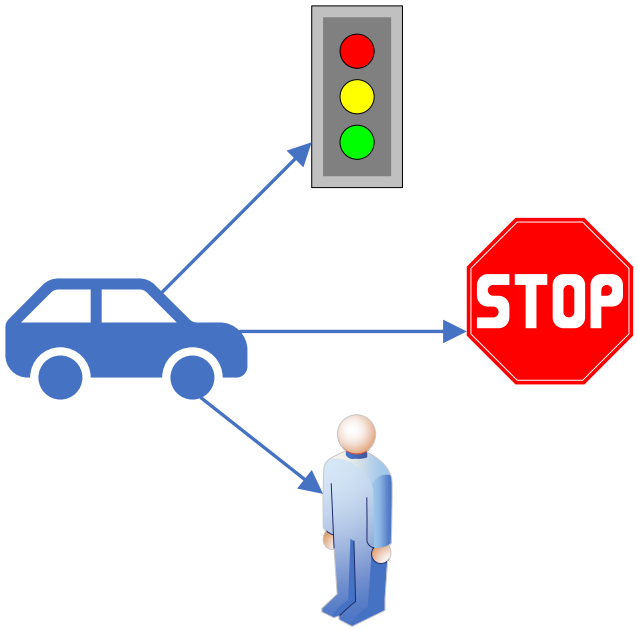
Tools vs Use Case Distribution (Non-SC)



- **Sensor Fusion**
  - Camera
    - Image Processing
    - Computer Vision
    - Deep Learning
  - Radar
    - Signal Processing
    - Deep Learning
  - Combine input sources
    - Compute decision
- **Acceleration**
  - Back-end execution of high-level APIs

- **Tools**
  - TensorFlow
  - OpenCV
  - Pytorch
  - Python Sci-Kit
  - Vendor-specific APIs and frameworks
  - SYCL
  - OpenCL
  - Vulkan

Tools vs Use Case Distribution (SC)



- **Sensor Fusion**
  - Camera
    - Image Processing
    - Computer Vision
    - Deep Learning
  - Radar
    - Signal Processing
    - Deep Learning
  - Combine input sources
    - Compute decision
- **Acceleration**
  - Back-end execution of high-level APIs

- **APIs**
  - OpenVX 1.3
  - Vulkan SC
- **Missing**
  - Tools/Libraries
  - High-Level parallel programming API (SYCL SC)

Why SYCL SC

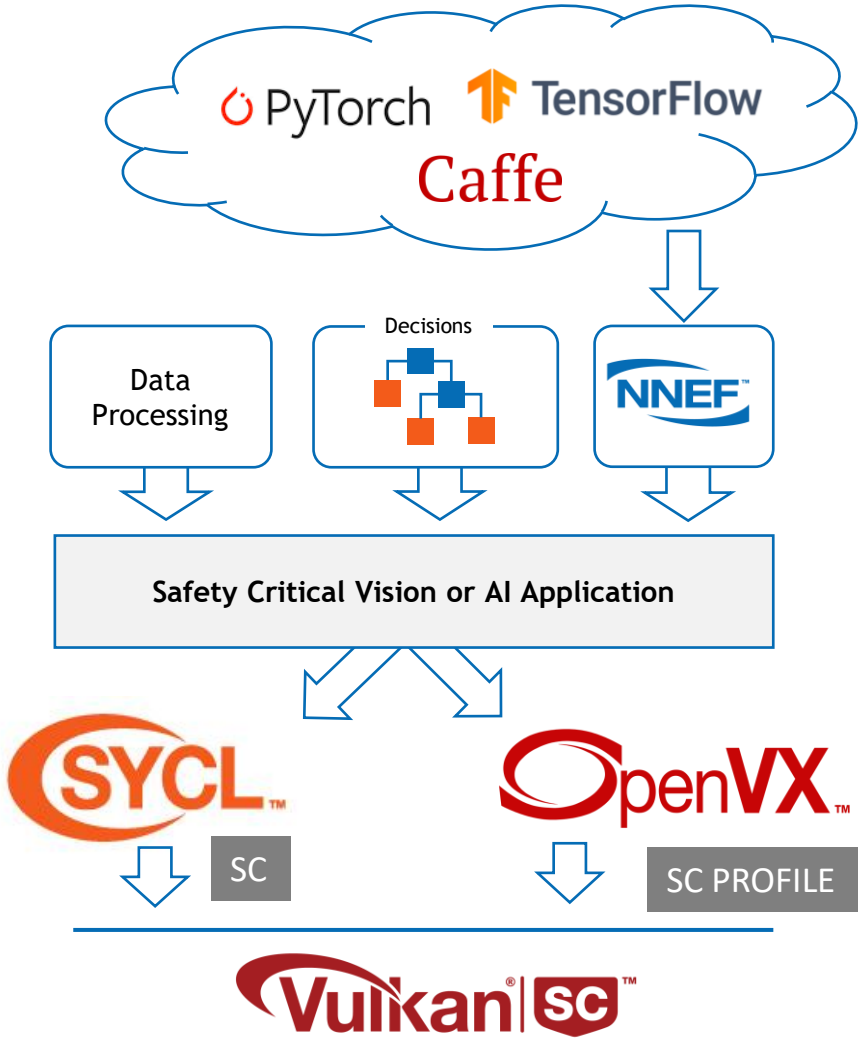
- **Most embedded developers are familiar with C++**
  - Non-sc parallel compute dominated by C++
  - Barrier to entry for C++ much lower than OpenCL or Vulkan SC

- **Khronos already defines SYCL**
  - A parallel programming API for C++
  - Khronos has a long track record of producing SC APIs from existing ones (OpenGL SC, Vulkan SC, OpenVX).
  - SYCL SC is the next logical step

- **SC ecosystem based on open standards must have APIs for each level of abstraction**

- **A single API cannot address every problem**

SYCL SC in the Khronos SC Ecosystem



Neural network models are trained in the cloud using a variety of platforms.

Once the model is trained it is exported and converted to NNEF before being passed to a safety critical API for inferencing.

OpenVX provides high level APIs for Vision and AI with a safety critical profile, enabling applications to quickly deploy trained NN models.

SYCL SC provides a general parallel programming API for accelerated compute at the C++ level. A typical AI application pipeline will combine the discreet functionality exposed by OpenVX with proprietary algorithms written using SYCL SC involving data pre-processing and post-processing, as well as complex decision making.

Vulkan SC is a lower, execution-level API that could be used to accelerate higher-level APIs like SYCL SC & OpenVX

SYCL SC – Considerations

- **SYCL SC should follow/monitor Vulkan SC progress**
  - Should be possible to implement SYCL SC on top of Vulkan SC

- **Definition of "Safety Critical" similar to Vulkan SC's definition**
  - Enable deterministic execution of software (in time, and space)
    - Deterministic management of memory resources
      - Dis-allow dynamic memory allocations?
      - SYCL SC instance must allocate all memory resources at init time?
    - Exception handling – can it be removed?

  - Reduce complexity of runtime
    - Move shader/kernel compilation step to offline
    - Remove redundant or dev/debuging APIs

- Reduce undefined behavior (document usage that leads to UB)
- Robust error handling

Memory Allocation

- Memory allocations influence execution time behaviour
  - Dynamic memory allocations lead to memory fragmentation
  - Memory fragmentation leads to non-deterministic execution
- Static memory allocations
- Application to define memory resources to be used in its lifetime
  - Done during initialization
- Dis-allowing freeing of resources at runtime

Exceptions

- Typically allocated on the heap -> dynamically
- Typically use RTTI -> non-deterministic run time

-> behaviour depends on the compiler, not standard

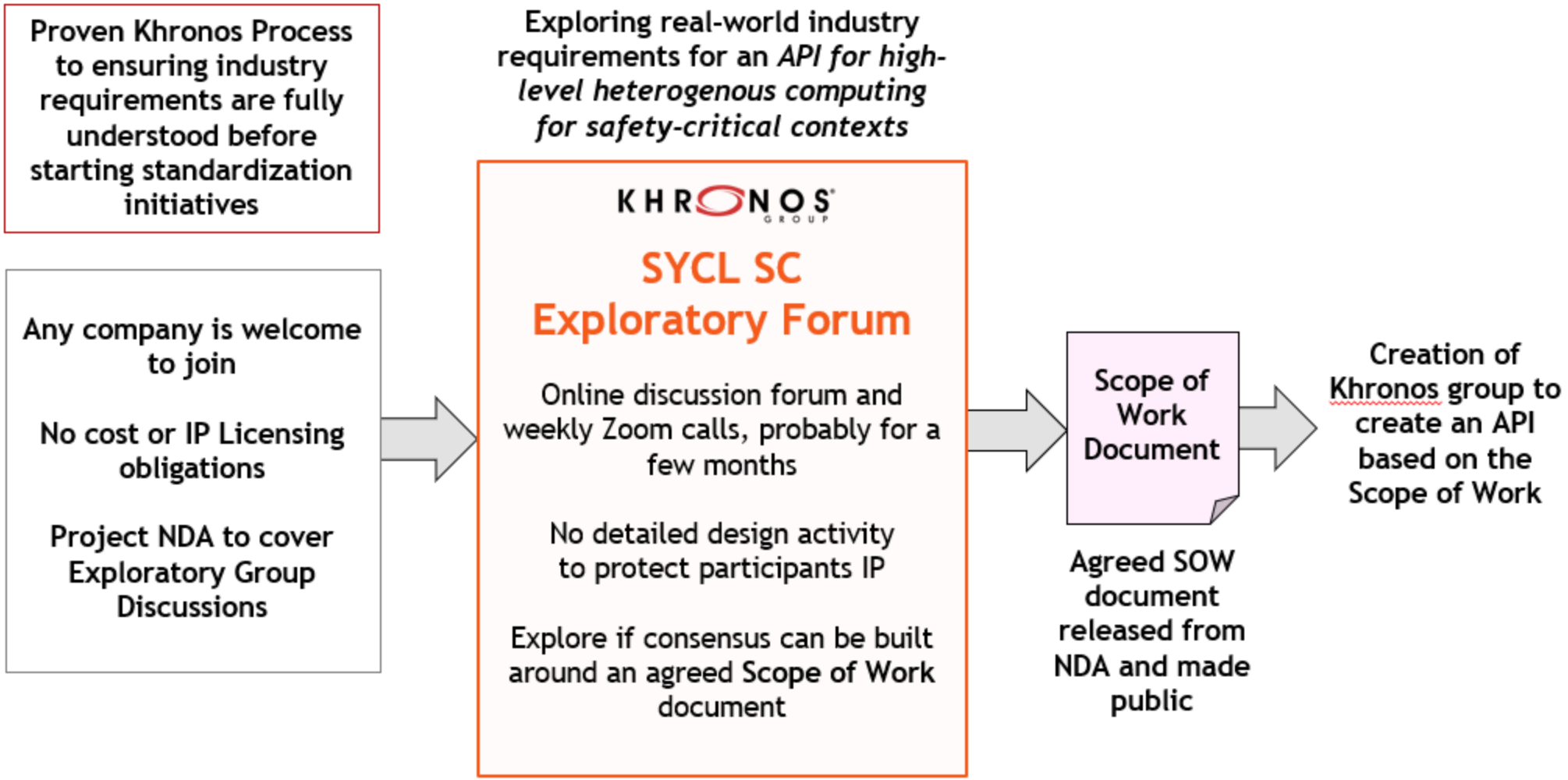
Not necessarily bad by themselves

- Solutions:
  - Implement exceptions as abort, perhaps with callback
  - Error codes
  - std::expected

Undefined Behaviour

- Ideally API has no undefined behaviour
- Undefined Behaviour should be minimized and constrained to specific areas
  - Failure may crash GPU but not entire system
- APIs must have well defined valid usage
  - Situations where undefined behaviour might occur should be well understood and documented
  - API functions should document valid usage
  - Valid usage must never lead to undefined behaviour
  - Breaking valid usage may lead to undefined behaviour

Exploratory Forum Process



Thank you!

- Join us at the SYCL SC Exploratory Forum
  - Visit https://www.khronos.org/syclsc
  - Or email sycl_sc_ef-chair@lists.khronos.org

- Meeting Mondays at 9:00 Pacific Time

CoreAVI

INNOVATING A SAFER TOMORROW

Thank you!