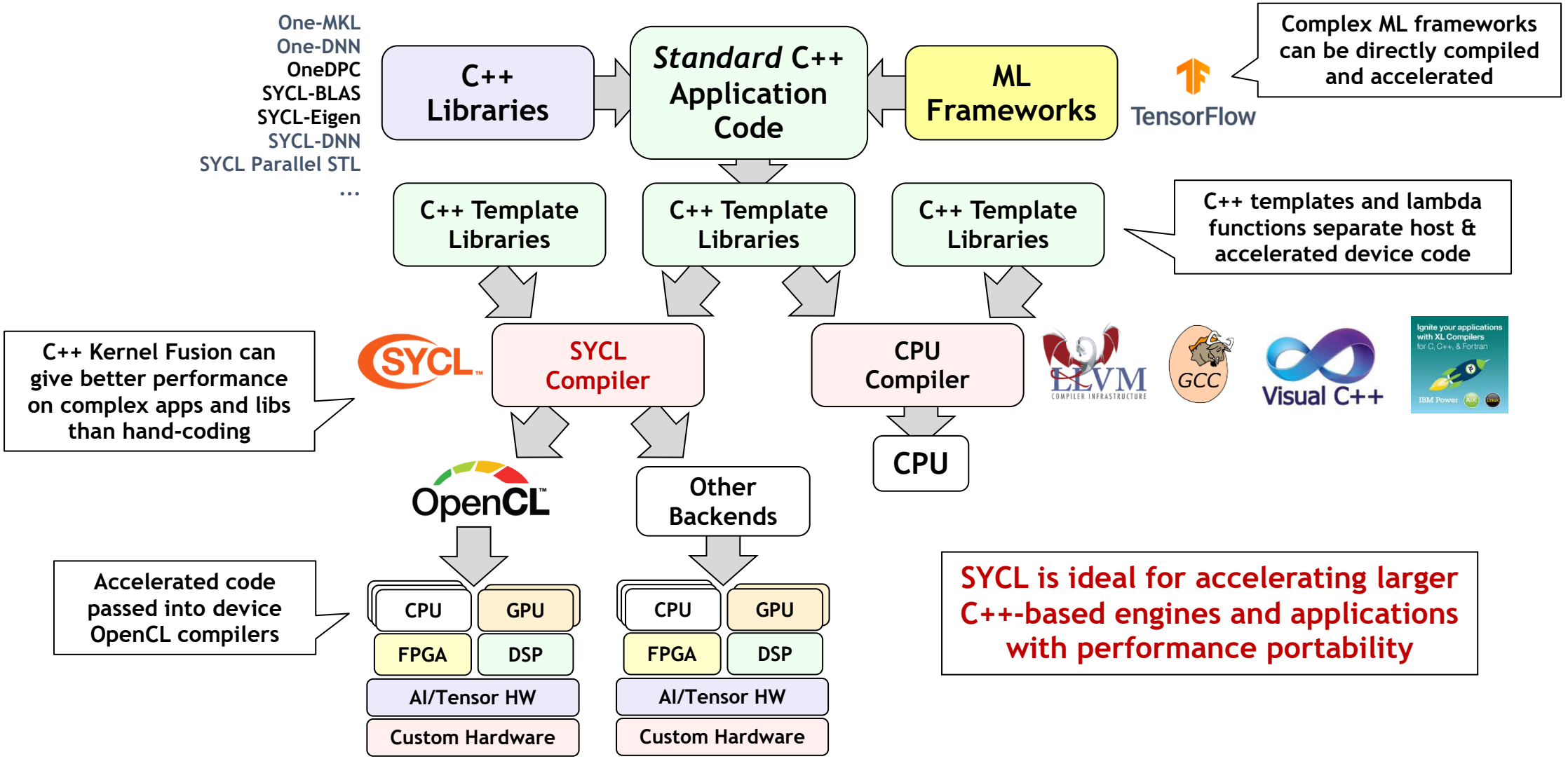
SYCL Academy

# SYCL 2020 and the future

Michael WOng

Acknowledgements:
SYCL WG
Rod Burns

# SYCL Single Source C++ Parallel Programming




# SYCL 2020 is here!

## Open Standard for Single Source C++ Parallel Heterogeneous Programming

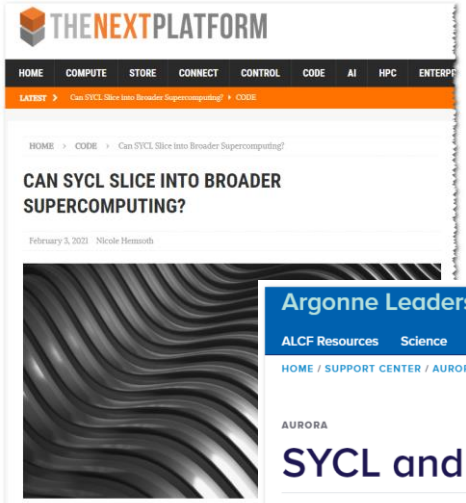
SYCL 2020 is released after 3 years of intense work
Significant adoption in Embedded, Desktop and HPC markets
Improved programmability, smaller code size, faster performance
Based on C++17, backwards compatible with SYCL 1.2.1
Simplify porting of standard C++ applications to SYCL
Closer alignment and integration with ISO C++
Multiple Backend acceleration and API independent

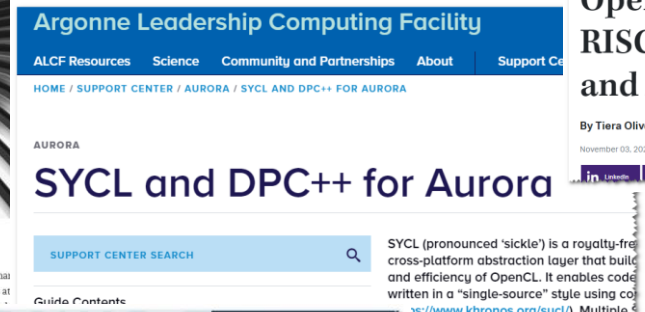
SYCL 2020 increases expressiveness and simplicity
for modern C++ heterogeneous programming

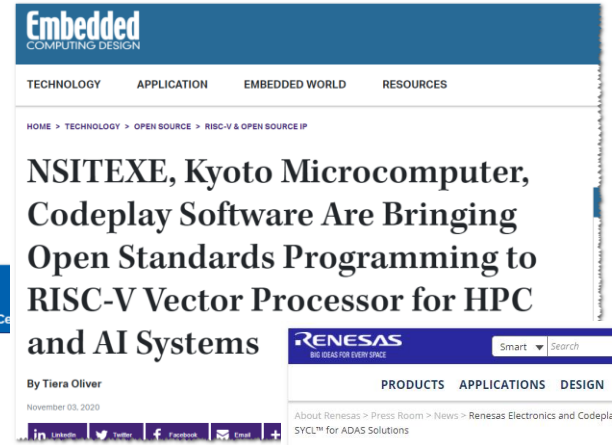

KHRONOS GROUP

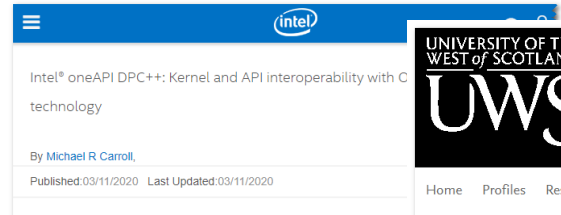
SYCL Academy

# SYCL 2020 Industry Momentum

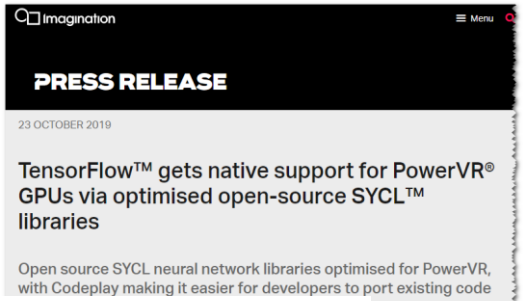


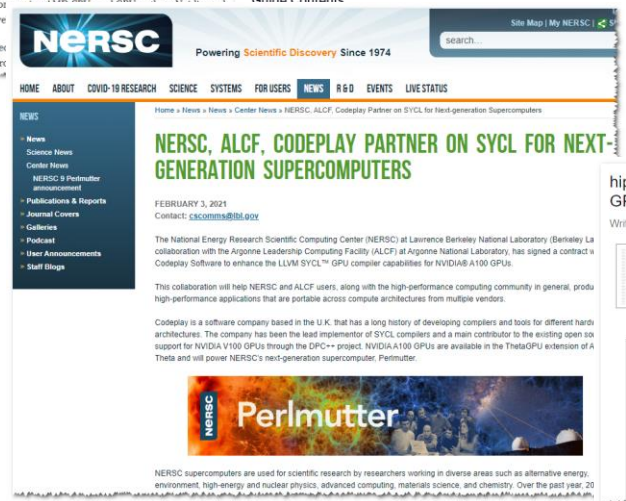



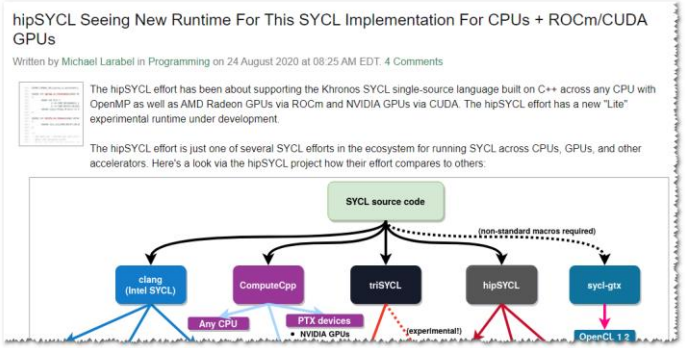







**SYCL support growing from Embedded Systems through Desktops to Supercomputers**

https://www.alcf.anl.gov/support-center/aurora/sycl-and-dpc-aurora
https://www.embeddedcomputing.com/technology/open-source/risc-v-open-source-lp/nsitexe-kyoto-microcomputer-and-codeplay-software-are-bringing-open-standards-programming-to-risc-v-vector-processor-for-hpc-and-ai-systems
https://www.nextplatform.com/2021/02/03/can-sycl-slice-into-broader-supercomputing/
https://www.phoronix.com/scan.php?page=news_item&px=hipSYCL-New-Lite-Runtime
https://software.intel.com/content/www/us/en/develop/articles/interoperability-dpcpp-sycl-opencl.html
https://www.renesas.com/br/en/about/press-room/renesas-electronics-and-codeplay-collaborate-opencl-and-sycl-adas-solutions
https://www.nersc.gov/news-publications/nersc-news/nersc-center-news/2021/nersc-alcf-codeplay-partner-on-sycl-for-next-generation-supercomputers/
https://www.argonne.gov/news/press-releases/2021/argonne-oak-ridge-national-laboratories-award-codeplay-software-further-strengthen-sycl-support-extending-open-standard-software-amd-gpus
https://research-portal.uws.ac.uk/en/publications/trisycl-for-xilinx-fpga
https://www.imaginationtech.com/news/press-release/tensorflow-gets-native-support-for-powervr-gpus-via-optimised-open-source-sycl-libraries/

# SYCL 2020 Major Features

- Unified Shared Memory (USM)
  - Code with pointers can work naturally without buffers or accessors
  - Simplifies porting from most code (e.g. CUDA, C++)
- Parallel Reductions
  - Added built-in reduction operation to avoid boilerplate code and achieve maximum performance on hardware with built-in reduction operation acceleration.
- Work group and subgroup algorithms
  - Efficient parallel operations between work items
- Class template argument deduction (CTAD) and template deduction guides
  - Simplified class template instantiation
- Simplified use of Accessors with a built-in reduction operation
  - Reduces boilerplate code and streamlines the use of C++ software design patterns
- Expanded interoperability
  - Efficient acceleration by diverse backend acceleration APIs
- SYCL atomic operations are now more closely aligned to standard C++ atomics
  - Enhances parallel programming freedom

# Parallel Industry Initiatives


**C++11**


**C++14**


**C++17**


**C++20**

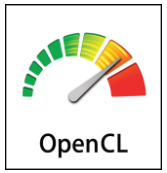

**C++23**


**SYCL 1.2**
**C++11 Single source programming**

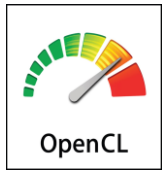

**SYCL 1.2.1**
**C++11 Single source programming**


**SYCL 2020**
**C++17 Single source programming**
**Many backend options**

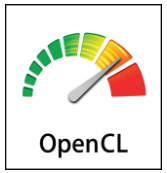

**SYCL 202X**
**C++20 Single source programming**
**Many backend options**




**OpenCL 1.2**
**OpenCL C Kernel Language**




**OpenCL 2.1**
**SPIR-V in Core**




**OpenCL 2.2**


**OpenCL 3.0**

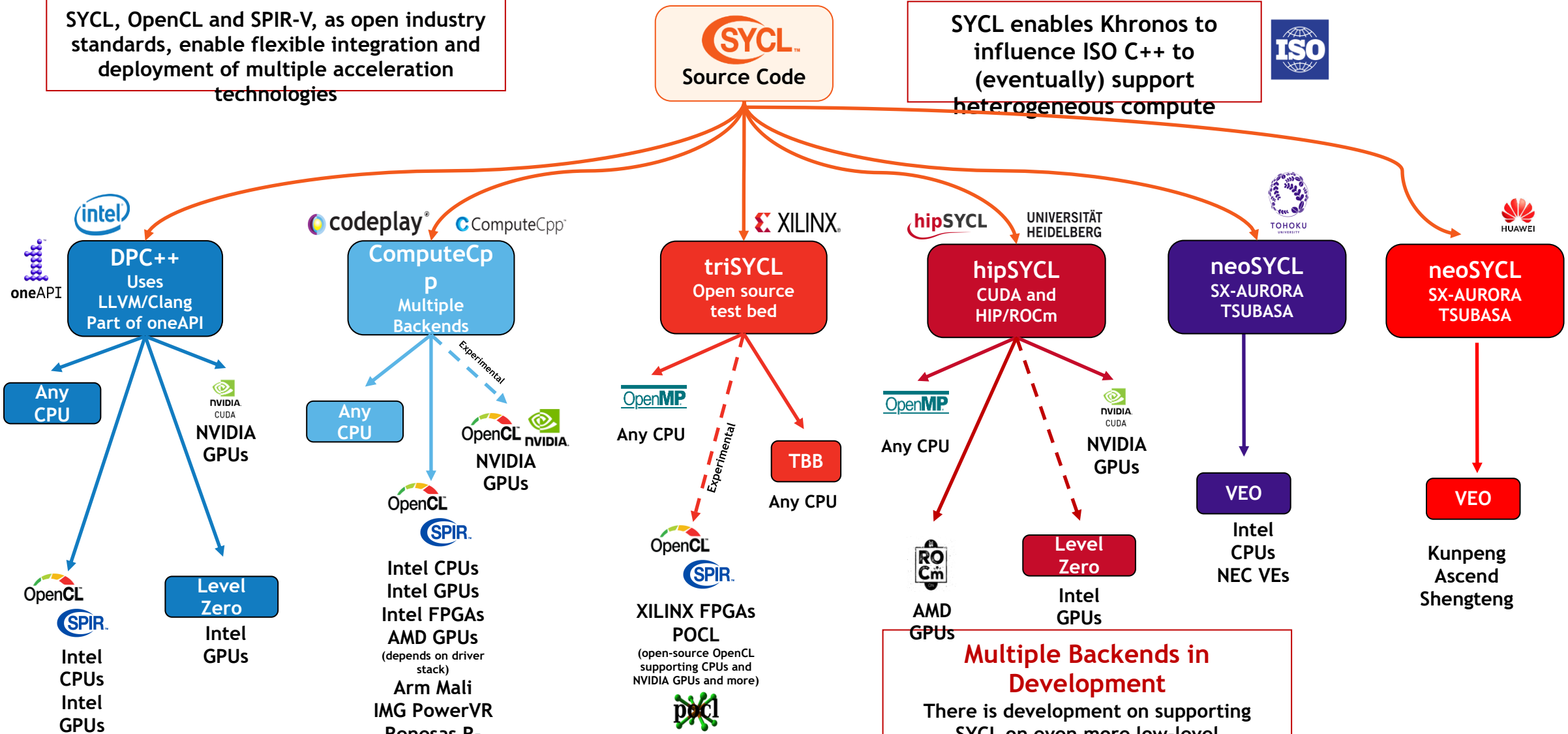





**2011** **2015** **2017** **2020** **202X**

# SYCL Implementations in Development

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies



# SYCL Ecosystem, Research and Benchmarks

## Implementations

triSYCL

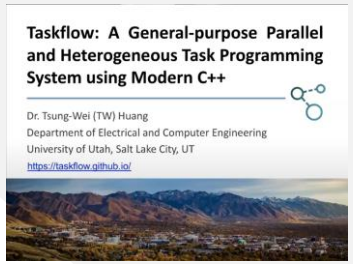
Celerity
High-level C++ for Accelerator Clusters

neoSYCL
SX-AURORA TSUBASA

ECP
EXASCALE COMPUTING PROJECT

kokkos

oneAPI


ComputeCpp

hipSYCL

GROMACS
FAST. FLEXIBLE. FREE.

alpaka

RAJA

ATLAS
EXPERIMENT

## Benchmarks/Books

Direct Programming Benchmark


SYCL-Bench

## Linear Algebra Libraries / Machine Learning Libraries and Parallel Acceleration Frameworks

| BLAS | FFT | Math | RAND |
|---|---|---|---|
| SYCLBLAS<br>oneMKL | oneMKL | oneMKL | oneMKL |
| **SOLVER** | **SPARSE** | **TENSOR** | **STL** |
| oneMKL | oneMKL | SYCL-DNN<br>Eigen<br>oneDNN<br>TensorFlow | SYCL Parallel STL<br>oneDPL |

## Working Group Members

STREAM HPC

Argonne NATIONAL LABORATORY

QUALCOMM

University of BRISTOL

arm

AMD

SYCL

codeplay

XILINX

universität innsbruck

intel

TOHOKU UNIVERSITY

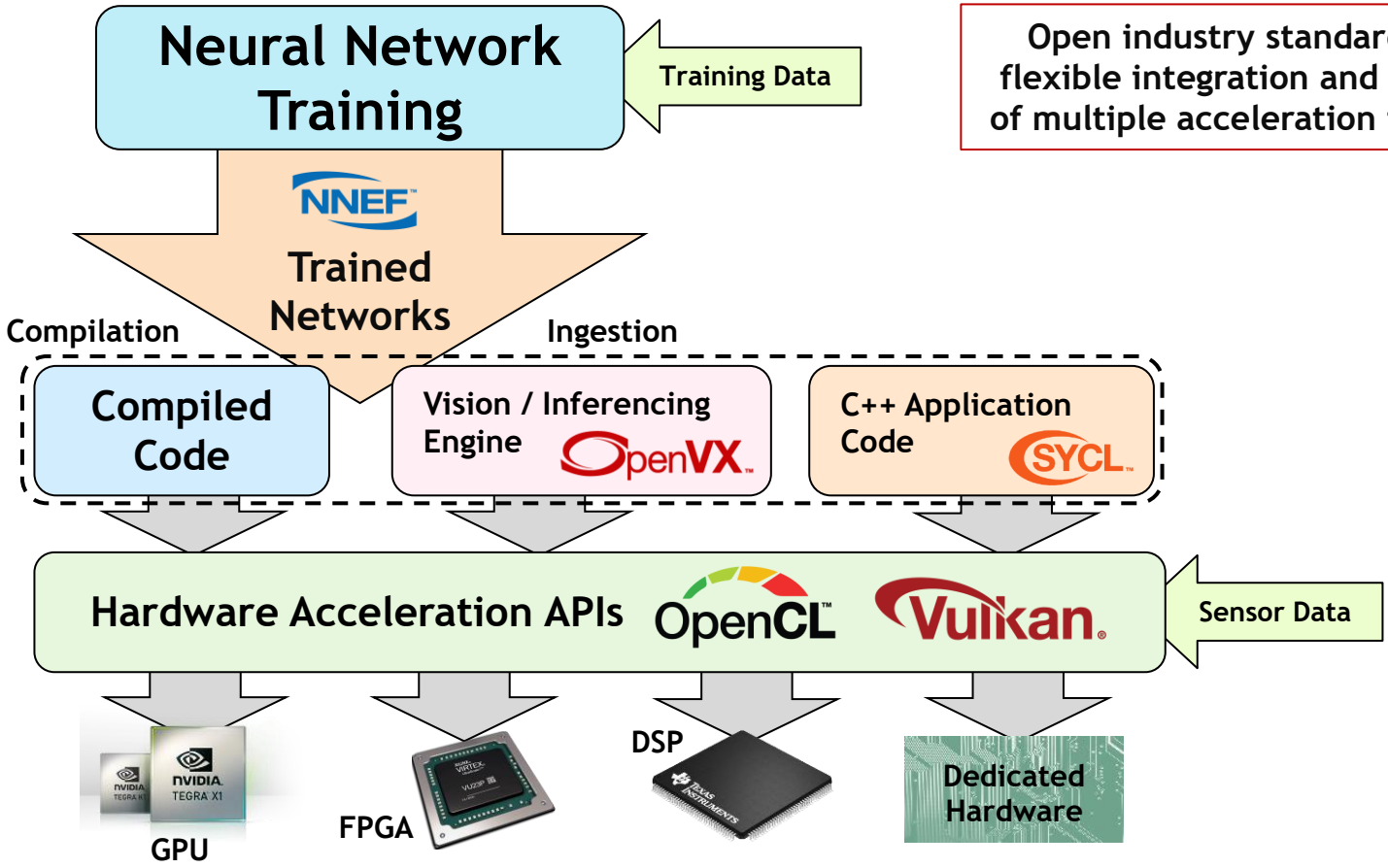
HIPPOCRATICA CIVITAS STUDIUM SALERNI

# SYCL in Embedded Systems, Automotive, and AI

**Networks trained on high-end desktop and cloud systems**

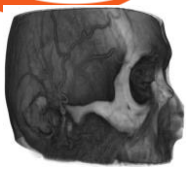
**Applications link to compiled inferencing code or call vision/inferencing API**

**Diverse Embedded Hardware**
Multi-core CPUs, GPUs
DSPs, FPGAs, Tensor Cores
* Vulkan only runs on GPUs




Open industry standards, enable flexible integration and deployment of multiple acceleration technologies

# Safety Critical API Evolution



New Generation Safety Critical APIs for Graphics, Compute and Display

OpenCL and SYCL SC work will minimize API surface area , reduce ambiguity, UB, increase determinism

OpenCL

Rendering Compute Display

Industry Need
for GPU Acceleration APIs designed to ease system safety certification is increasing
ISO 26262 / ASIL-D

UNECE WP.29

ISO 26262

ISO/PAS 21448

UL 4600

MISRA

SAE INTERNATIONAL

ISO/IEC JTC 1/SC 42
Artificial intelligence

SCSC FOR EVERYONE WORKING IN SYSTEM SAFETY WWW.SCSC.UK

ISO International Organization for Standardization

RISC-V

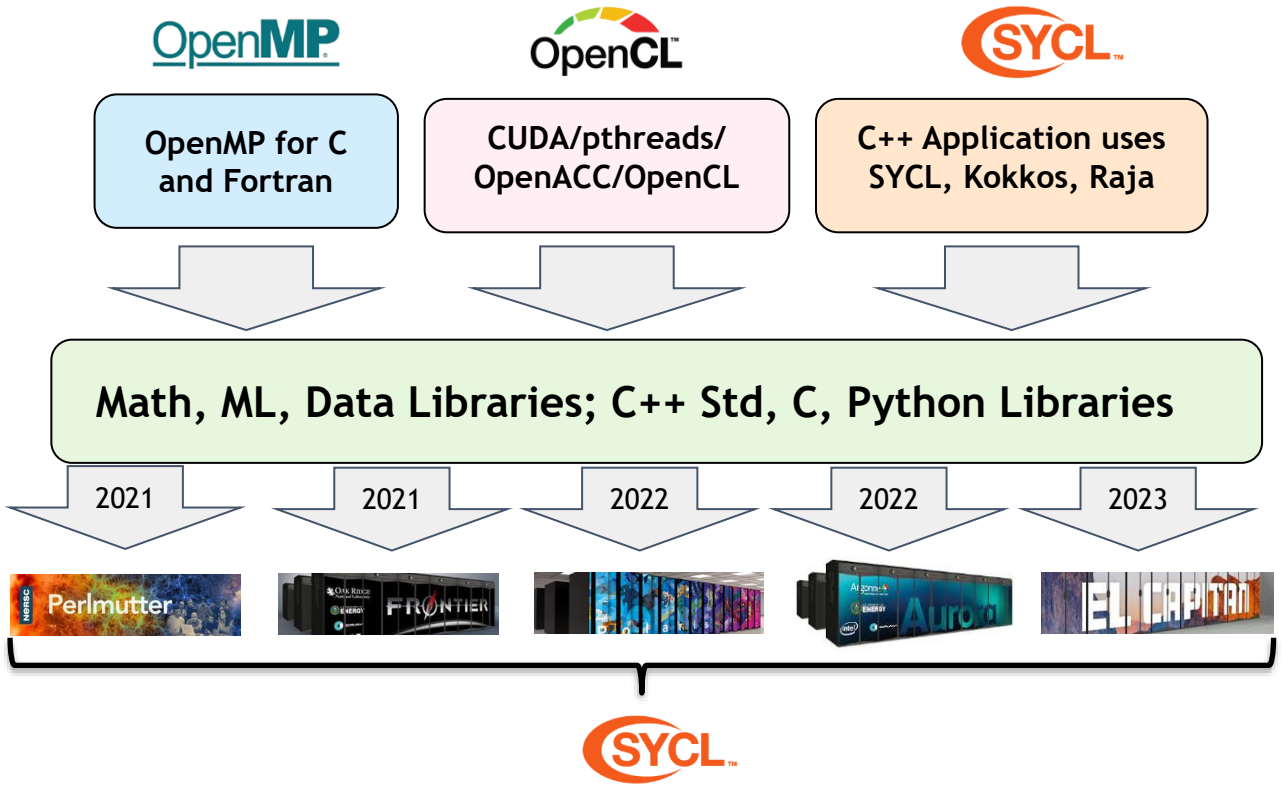
# SYCL in HPC/Supercomputers

**Simulation**
HPC Languages
Solver Libraries, Parallel RT

**Data**
Productivity Languages
Big Data Stack, Stats Lib, Databases

**Learning**
Productivity Languages
Deep Learning, Linear Alg, ML

Three Pillars of Science Problem

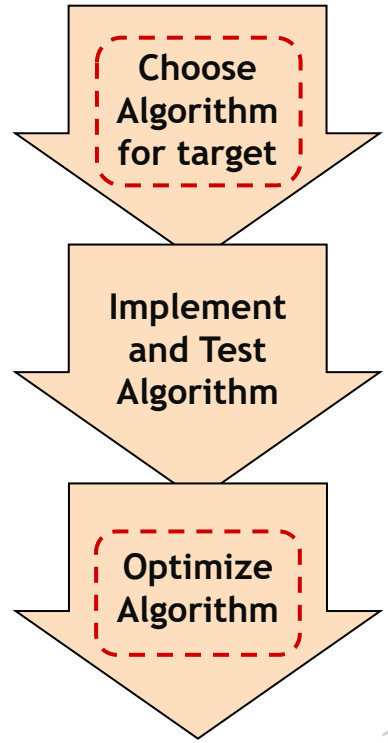



Need Languages that allow control of these Data Issues
Set Data affinity, Data Layout, Data movement, Data Locality, highly Parameterized Code and dynamically compose the algorithms (C++ templates, parallel STL, inlining and fusion, abstractions)

Libraries augment compiler optimizations for Performance Portable programs

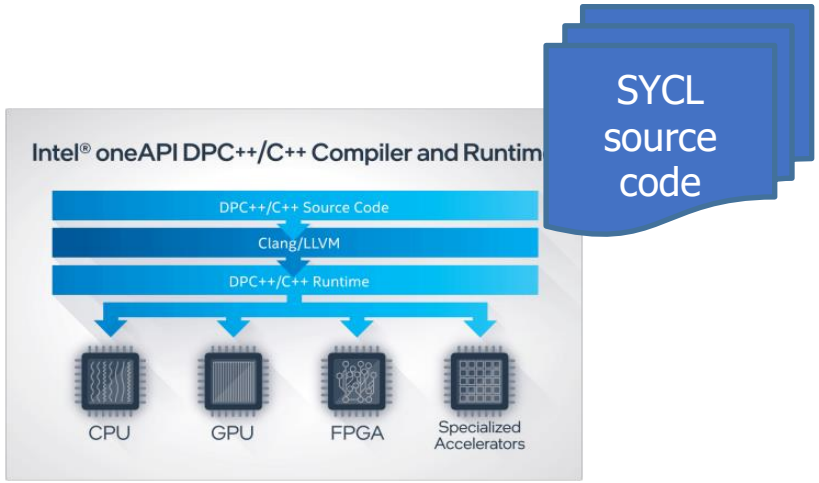
Use open standards to run Performance Portable code on new generation, or different vendor's, hardware with compiler optimization, explicit parametrization and dynamically composed algorithm

Today's Supercomputing Development Workflow needs knowledge of system architecture and tools that control data



# oneAPI and SYCL

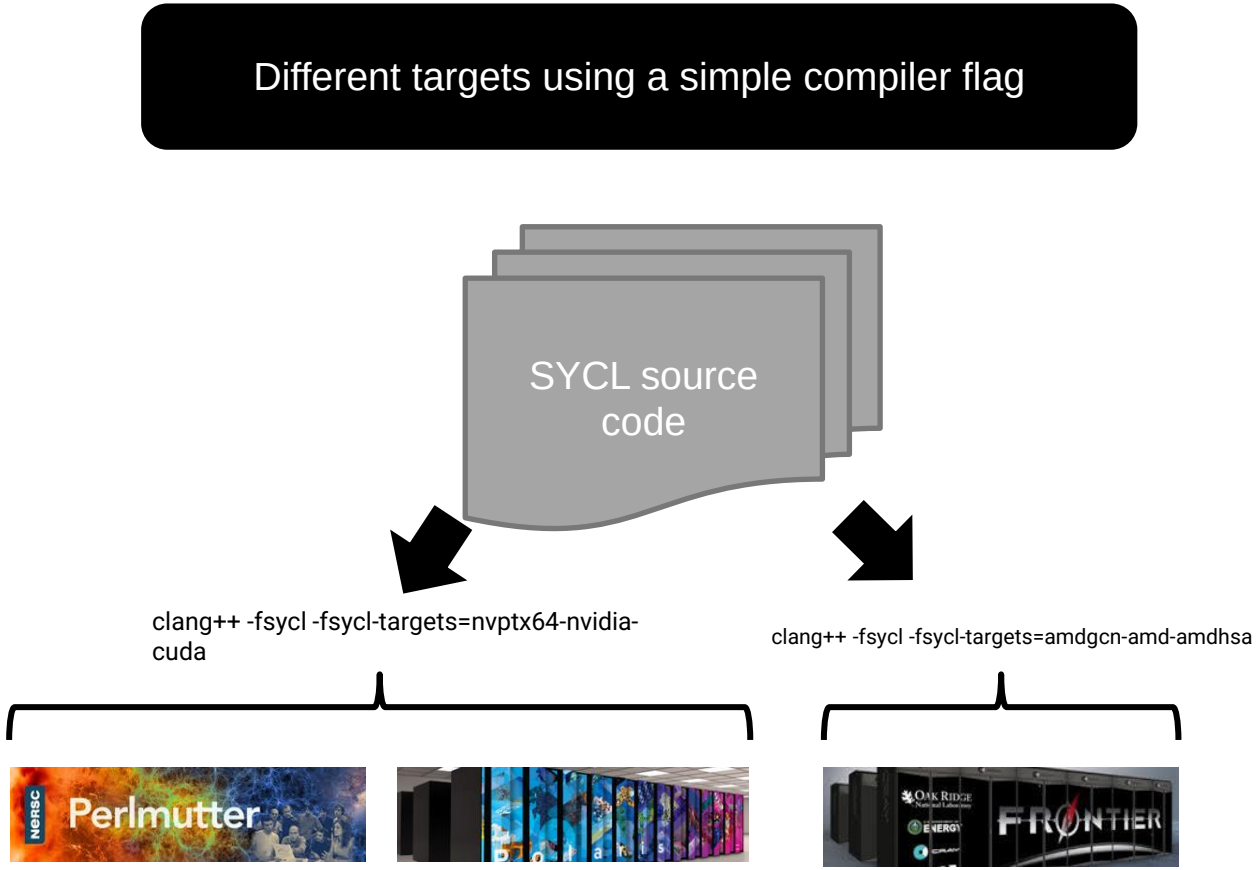





- SYCL sits at the heart of oneAPI
- Provides an open standard interface for developers
- Defined by the industry

# Nvidia and AMD Support in oneAPI

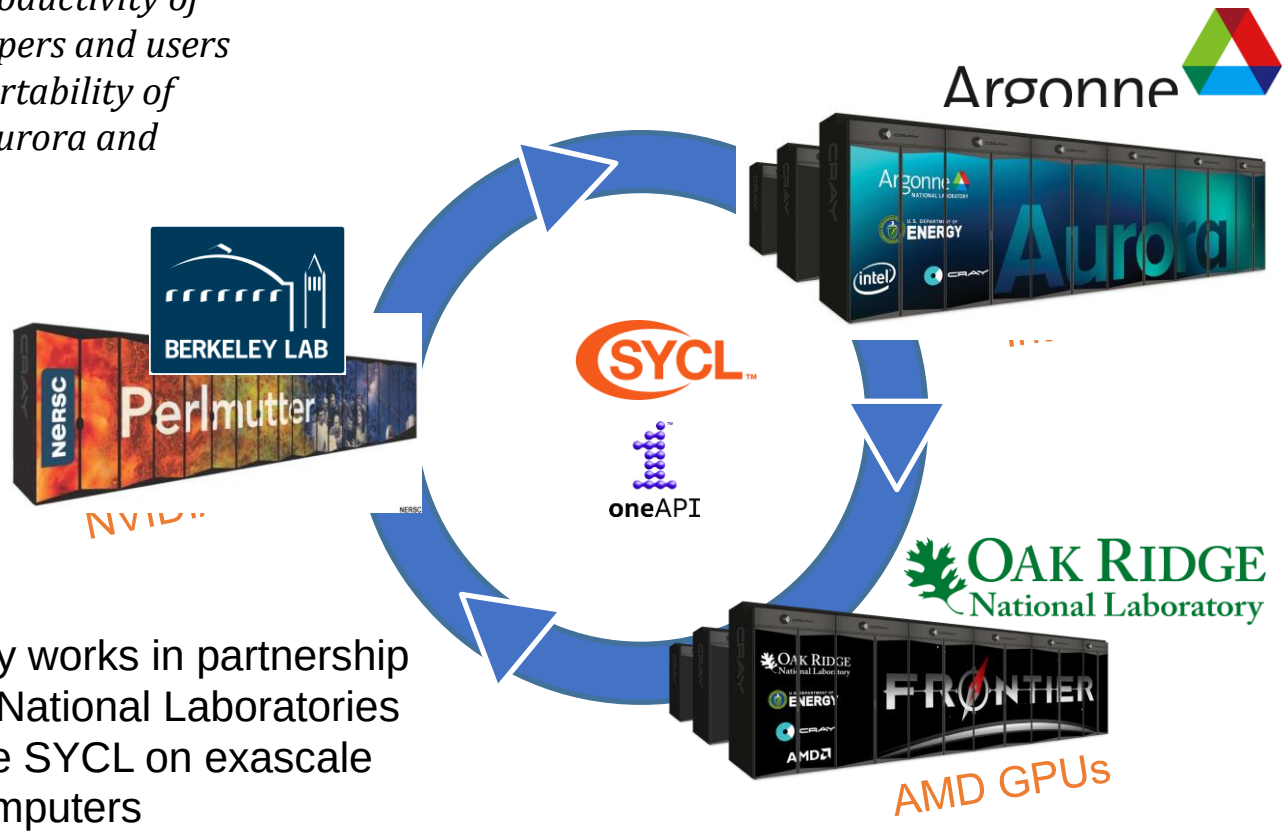
- Extending DPC++ to target Nvidia and AMD GPUs
- Supporting Perlmutter, Polaris and Frontier supercomputers
- Open source and available to everyone




**https://www.codeplay.com/oneapiforcuda**
**Resources for AMD coming soon**

# SYCL Enables Supercomputers

*"this work supports the productivity of scientific application developers and users through performance portability of applications between Aurora and Perlmutter."*




Codeplay works in partnership with US National Laboratories to enable SYCL on exascale supercomputers

Enables a broad range of software frameworks and applications

# SYCL Future Evolution

## SYCL 2020 compared with SYCL 1.2.1

- Easier to integrate with C++17 (CTAD, Deduction Guides...)
- Less verbose, smaller code size, simplify patterns
- Backend independent
- Multiple object archives aka modules simplify interoperability
- Ease porting C++ applications to SYCL
- Enable capabilities to improve programmability
- Backwards compatible but minor API break based on user feedback

SYCL Future Roadmap (MAY CHANGE)

SYCL 2020

**Over 40 Selected Features for SYCL 2020**

Unified Shared Memory)
Parallel Reductions adds a built in reduction operation
Work-group and sub-group algorithms
Improvements to atomic operations
Class template argument deduction (CTAD) and deduction guides
Simplification of accessors
Expanded interoperability with different backends
Extension mechanism
Address spaces
Vector rework
Specialization Constants
...

**Improving Software Ecosystem**

Books, Tutorials, Tool, libraries, GitHub

**Expanding Implementations**

DPC++
ComputeCpp
triSYCL
hipSYCL
neoSYCL

**Regular Maintenance Updates**

Spec clarifications, formatting and bug fixes
https://www.khronos.org/registry/SYCL/

**Repeat The Cycle every 1.5-3 years**

NEXT

**Conformance Tests**

**Working on Implementations**

**Future SYCL NEXT Proposals**

...

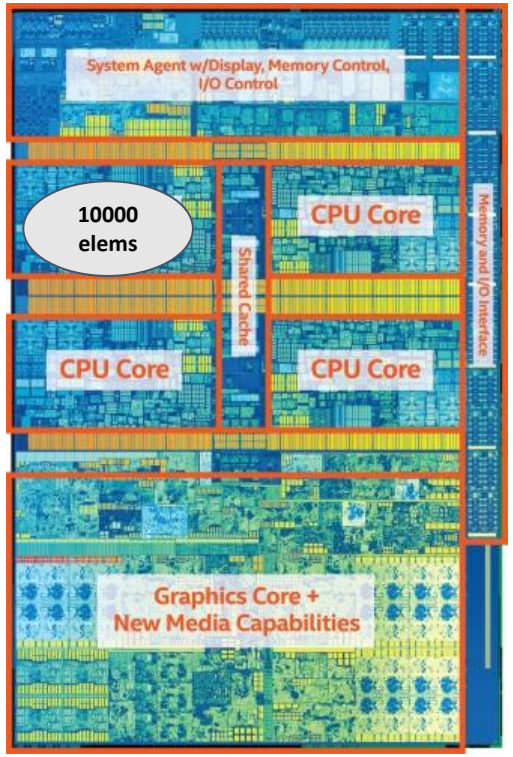
**Integration of successful Extensions plus new Core functionality**

**Converge SYCL with ISO C++ and continue to support OpenCL to deploy on more devices**

CPU
GPU
FPGA
AI processors
Custom Processors

# A Demo with C++ Parallel STL

# What can I do with a Parallel For Each?

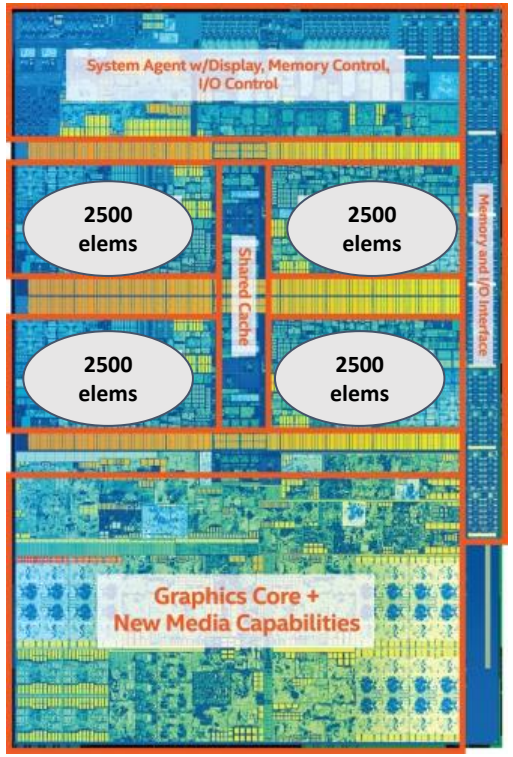



**Intel Core i7 7th generation**

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(std::begin(v1), nElems, 1);

std::for_each(std::begin(v), std::end(v),
                    [=](float f) { f * f + f });
```

**Traditional for each uses only one core, rest of the die is unutilized!**

# What can I do with a Parallel For Each?



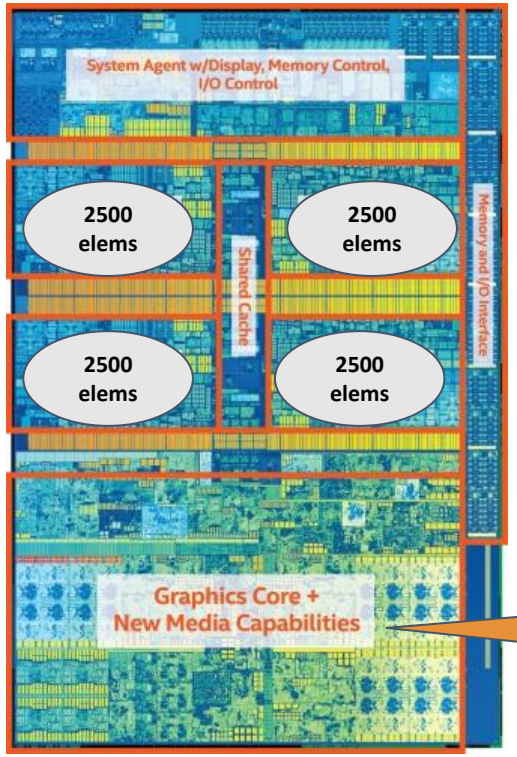

**Intel Core i7 7th generation**

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(std::execution_policy::par,
            std::begin(v1), nElems, 1);

std::for_each(std::execution_policy::par,
              std::begin(v), std::end(v),
              [=](float f) { f * f + f });
```

**Workload is distributed across cores!**

(mileage may vary, implementation-specific behaviour)

# What can I do with a Parallel For Each?




**Intel Core i7 7th generation**

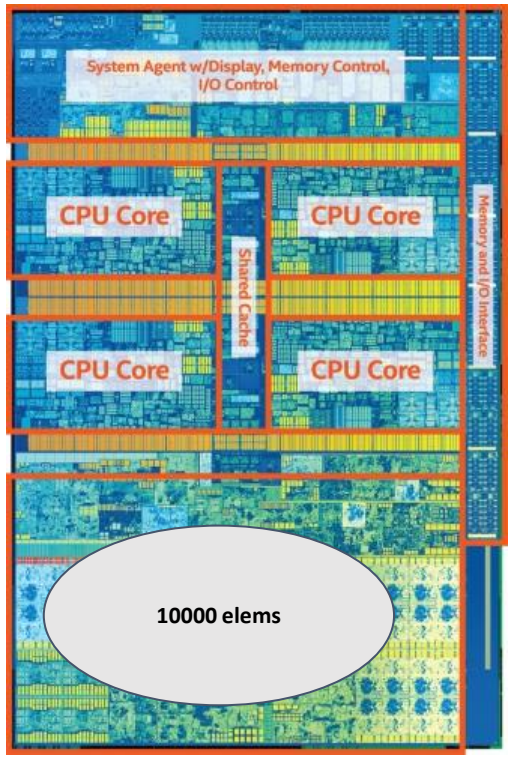
What about this part?

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(std::execution_policy::par,
            std::begin(v1), nElems, 1);

std::for_each(std::execution_policy::par,
              std::begin(v), std::end(v),
              [=](float f) { f * f + f });
```

**Workload is distributed across cores!**

(mileage may vary, implementation-specific behaviour)

# What can I do with a Parallel For Each?



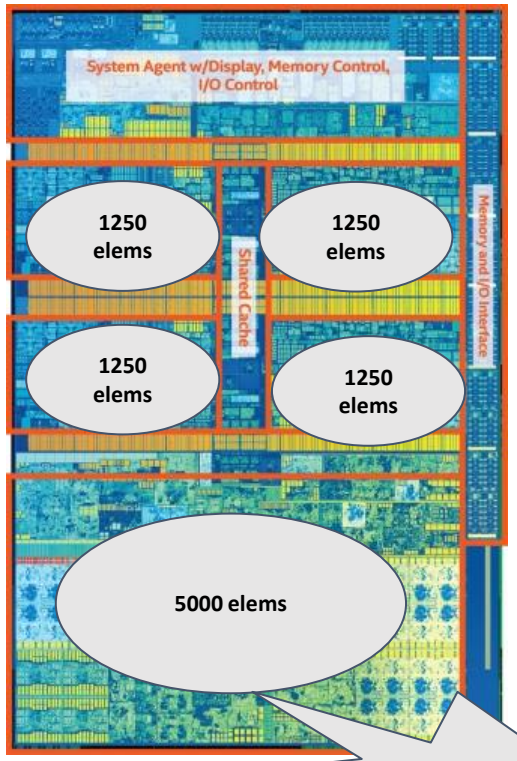

**Intel Core i7 7th generation**

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(sycl_policy,
            std::begin(v1), nElems, 1);

std::for_each(sycl_named_policy
                <class KernelName>,
              std::begin(v), std::end(v),
               [=](float f) { f * f + f });
```

**Workload is distributed on the GPU cores**

(mileage may vary, implementation-specific behaviour)

# What can I do with a Parallel For Each?




Intel Core i7 7th

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(sycl_heter_policy(cpu, gpu, 0.5),
            std::begin(v1), nElems, 1);

std::for_each(sycl_heter_policy<class kName>
              (cpu, gpu, 0.5),
              std::begin(v), std::end(v),
              [=](float f) { f * f + f });
```

**Workload is distributed on all cores!**

**Experimental!**

(mileage may vary, implementation-specific behaviour)

# Demo Results - Running std::sort
# (Running on Intel i7 6600 CPU & Intel HD  Graphics 520)

| size | 2^16 | 2^17 | 2^18 | 2^19 |
|---|---|---|---|---|
| std::seq | 0.27031s | 0.620068s | 0.669628s | 1.48918s |
| std::par | 0.259486s | 0.478032s | 0.444422s | 1.83599s |
| std::par_unseq | 0.24258s | 0.413909s | 0.456224s | 1.01958s |
| sycl_execution_policy | 0.273724s | 0.269804s | 0.277747s | 0.399634s |


# SYCL 2020 is here!

## Open Standard for Single Source C++ Parallel Heterogeneous Programming

SYCL 2020 is released after 3 years of intense work
Significant adoption in Embedded, Desktop and HPC markets
Improved programmability, smaller code size, faster performance
Based on C++17, backwards compatible with SYCL 1.2.1
Simplify porting of standard C++ applications to SYCL
Closer alignment and integration with ISO C++
Multiple Backend acceleration and API independent

SYCL 2020 increases expressiveness and simplicity
for modern C++ heterogeneous programming


KHRONOS GROUP

# Enabling Industry Engagement

- SYCL working group values industry feedback
  - https://community.khronos.org/c/sycl
  - https://sycl.tech
- SYCL FAQ
  - https://www.khronos.org/blog/sycl-2020-what-do-you-need-to-know
- What features would you like in future SYCL versions?

- **Advisory Panel Chaired by Tom Deakin of U of Bristol**
- **Quarterly SYCL Advisory Panel**
- **Regular meetings to give feedback on roadmap and draft specifications**

**Public contributions to Specification, Conformance Tests and software**
https://github.com/KhronosGroup/SYCL-CTS
https://github.com/KhronosGroup/SYCL-Docs
https://github.com/KhronosGroup/SYCL-Shared
https://github.com/KhronosGroup/SYCL-Registry
https://github.com/KhronosGroup/SyclParallelSTL
https://github.com/intel/llvm

**Invited Experts**
https://www.khronos.org/advisors/

**Khronos members**
https://www.khronos.org/members/
https://www.khronos.org/registry/SYCL/

**Open to all!**
https://community.khronos.org/www.khr.io/slack
https://app.slack.com/client/TDMDFS87M/CE9UX4CHG
https://community.khronos.org/c/sycl/
https://stackoverflow.com/questions/tagged/sycl
https://www.reddit.com/r/sycl
https://github.com/codeplaysoftware/syclacademy
https://sycl.tech/