codeplay®

THE HETEROGENEOUS SYSTEMS EXPERTS

The landscape of Parallel Programing Models
Part 2:
The importance of Data

Michael Wong and Rod Burns

Codeplay Software Ltd.

Distiguished Engineer, Vice President of Ecosystem

IXPUG 2020

Products

Acoran

Integrates all the industry standard technologies needed to support a very wide range of AI and HPC

ComputeCpp™

C++ platform via the SYCL™ open standard, enabling vision & machine learning e.g. TensorFlow™

ComputeAorta™

The heart of Codeplay's compute technology enabling OpenCL™, SPIR-V™, HSA™ and Vulkan™

Markets

High Performance Compute (HPC)
Automotive ADAS, IoT, Cloud Compute
Smartphones & Tablets
Medical & Industrial

Technologies: Artificial Intelligence
Vision Processing
Machine Learning
Big Data Compute

codeplay®

Enabling AI & HPC to be Open, Safe & Accessible to All

Company

Leaders in enabling high-performance software solutions for new AI processing systems

Enabling the toughest processors with tools and middleware based on open standards

Established 2002 in Scotland with ~80 employees

Customers

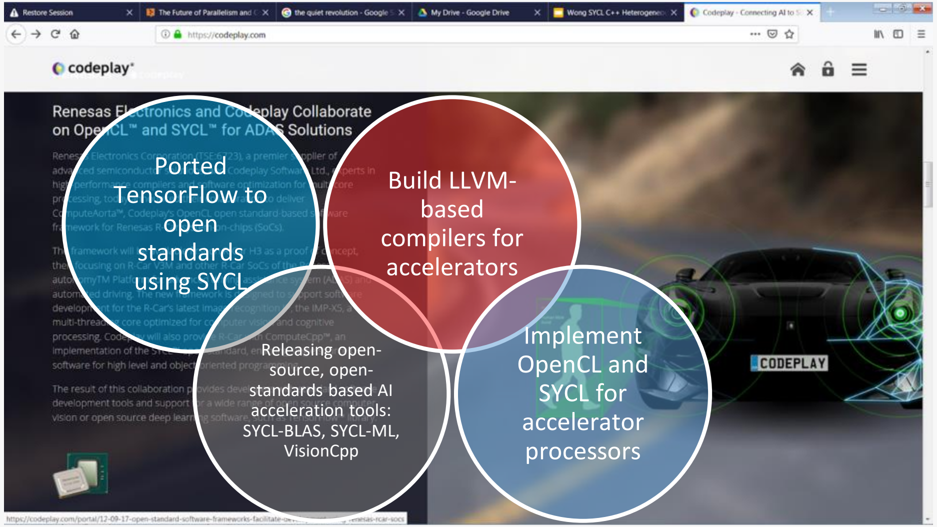
arm
BROADCOM.
CEVA
Imagination
intel
RENESAS
SYNOPSYS®

And many more!

Distinguished Engineer

Michael Wong

- Chair of SYCL Heterogeneous Programming Language
- C++ Directions Group
- ISOCPP.org Director, VP
  http://isocpp.org/wiki/faq/wg21#michael-wong
- michael@codeplay.com
- fraggamuffin@gmail.com
- 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
- Chair of Standards Council Canada TC22/SC32 Electrical and electronic components (SOTIF)
- Chair of UL4600 Object Tracking
- http://wongmichael.com/about
- C++11 book in Chinese:
  https://www.amazon.cn/dp/B00ETOV2OQ


Restore Session
The Future of Parallelism and
the quiet revolution - Google
My Drive - Google Drive
Wong SYCL C++ Heterogeneo
Codeplay - Connecting AI to
https://codeplay.com
codeplay
Renesas Electronics and Codeplay Collaborate on OpenCL™ and SYCL™ for ADAS Solutions
CODEPLAY

Ported TensorFlow to open standards using SYCL

Build LLVM-based compilers for accelerators

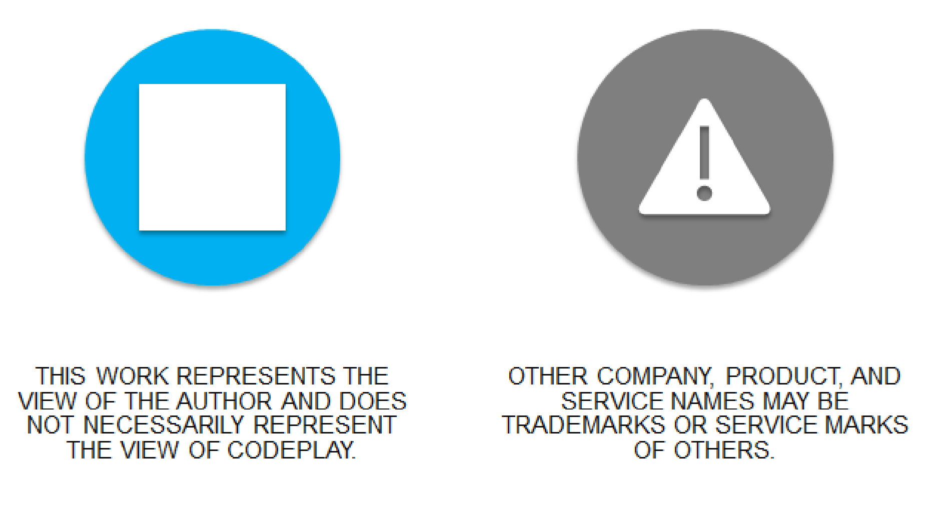
Releasing open-source, open-standards based AI acceleration tools: SYCL-BLAS, SYCL-ML, VisionCpp

Implement OpenCL and SYCL for accelerator processors

We build GPU compilers for semiconductor companies

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

Acknowledgement and 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.

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.

But I claim all credit for errors, and stupid mistakes. **These are mine, all mine! You can’t have them.**

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.

Disclaimers

NVIDIA, the NVIDIA logo and CUDA are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and/or other countries

Codeplay is not associated with NVIDIA for this work and it is purely using public documentation and widely available code

3 Act Play

1. Parallel Heterogeneous Programming Model comparison
2. OpenMP Accelerator and Data Movement
3. SYCL Data Movement: Accessors and USM

Industry Initiative

oneAPI

Intel Product

Act 1

Comparison of Parallel Heterogeneous Programming Models

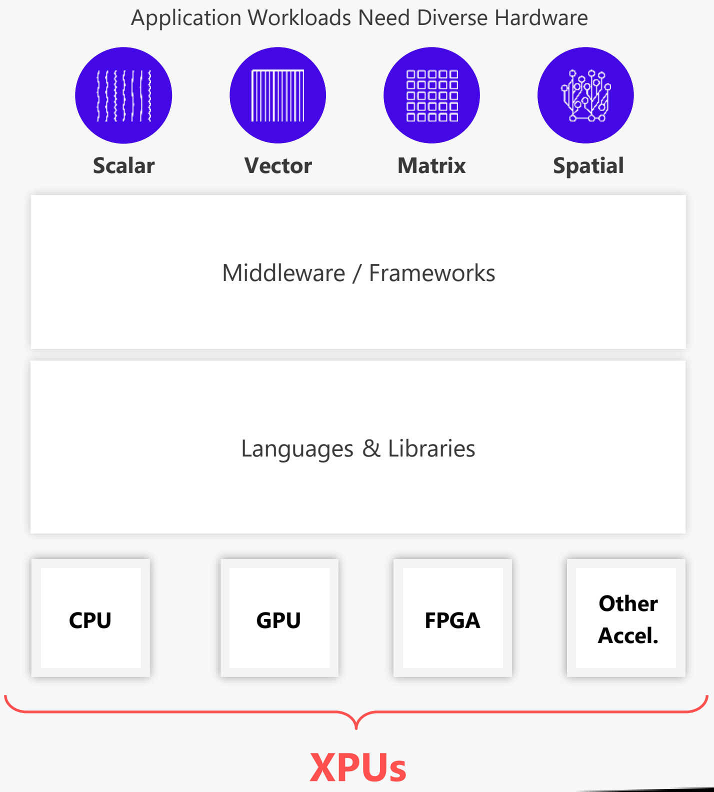
Industry Initiative

oneAPI

Intel Product

Programming Challenges for Multiple Architectures

- Growth in specialized workloads
- No common programming language or APIs
- Inconsistent tool support across platforms
- Each platform requires unique software investment
- Diverse set of data-centric hardware required

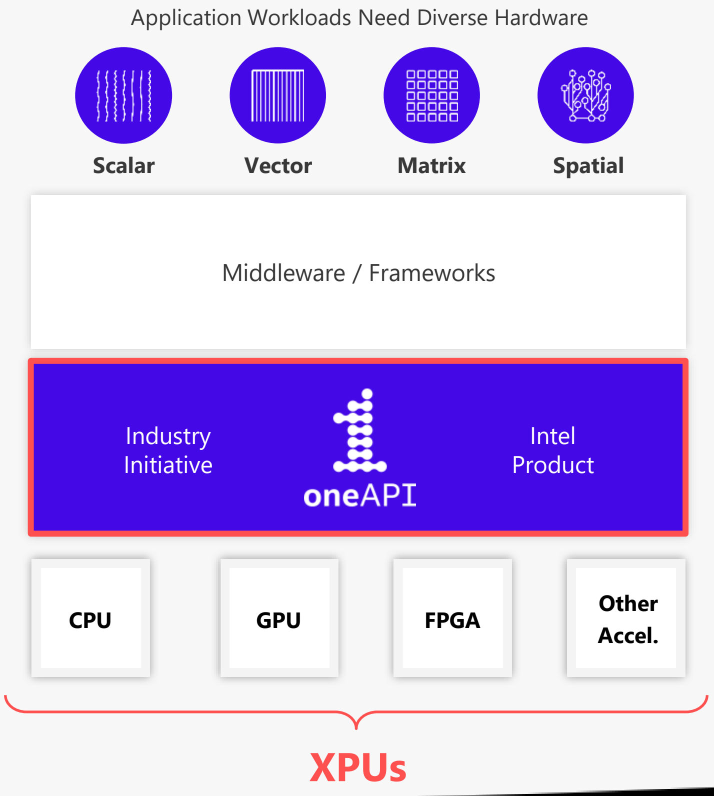


Introducing **oneAPI**

Unified programming model to simplify development across diverse architectures

- Unified and simplified language and libraries for expressing parallelism
- Uncompromised native high-level language performance
- Based on industry standards and open specifications
- Interoperable with existing HPC programming models

Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products.



Vision for **oneAPI Industry Initiative**

A top-to-bottom ecosystem around oneAPI specification

oneAPI Specification

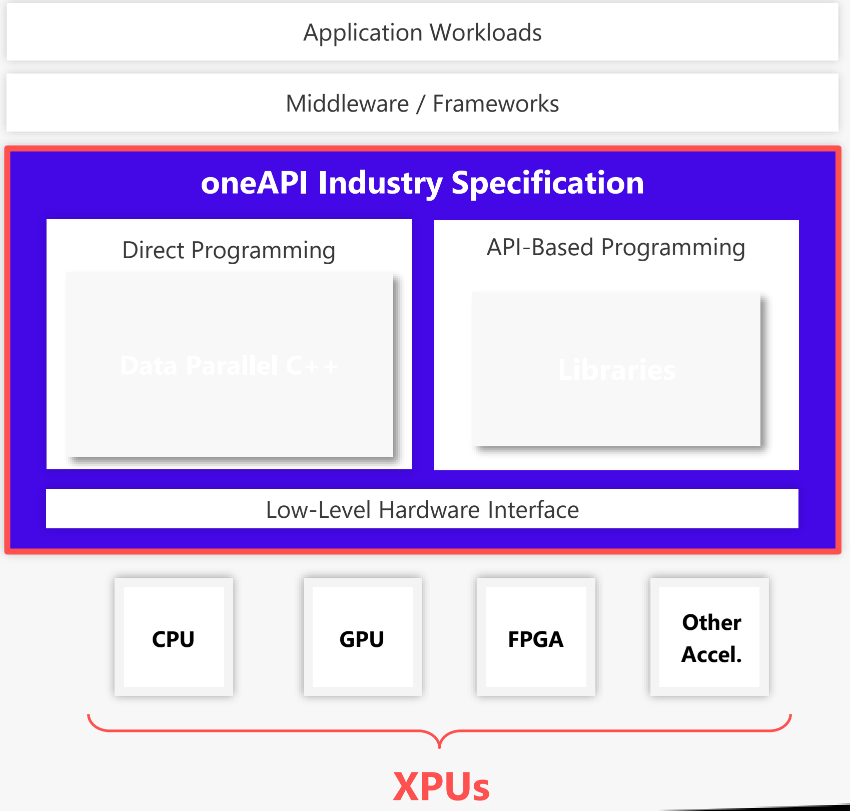
oneAPI Open Source Projects

oneAPI Commercial Products

Applications powered by oneAPI

oneAPI Industry Initiative

- oneAPI Industry Specification
  - A standards based cross-architecture language, DPC++, based on C++ and SYCL
  - Powerful APIs designed for acceleration of key domain-specific functions
  - Low-level hardware interface to provide a hardware abstraction layer to vendors
  - Enables code reuse across architectures and vendors
  - Open standard to promote community and industry support
- Technical Advisory Board
- oneAPI Industry Brand



Application Workloads

Middleware / Frameworks

oneAPI Industry Specification

Direct Programming

Data Parallel C++

API-Based Programming

Libraries

Low-Level Hardware Interface

CPU

GPU

FPGA

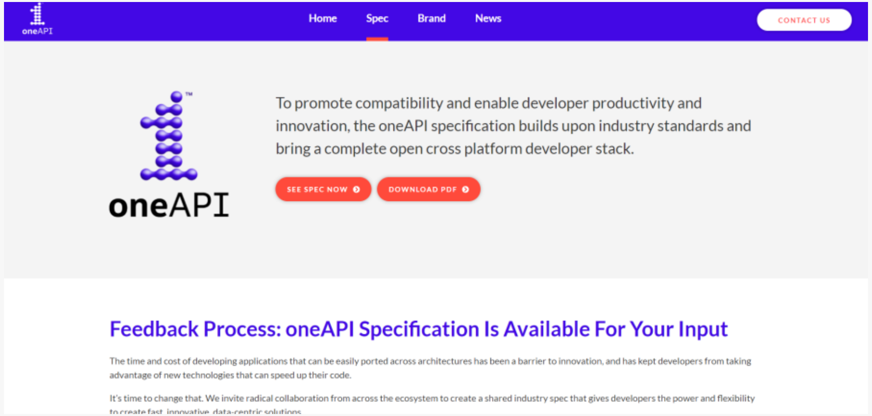
Other Accel.

XPUs

Some capabilities may differ per architecture and custom-tuning will still be required.
Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products.

Visit oneapi.com for more details

oneAPI Specification Feedback Process

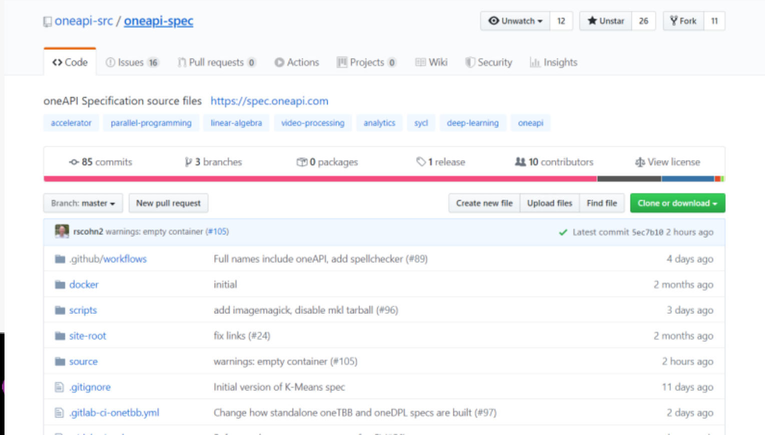


To promote compatibility and enable developer productivity and innovation, the oneAPI specification builds upon industry standards and bring a complete open cross platform developer stack.

Feedback Process: oneAPI Specification Is Available For Your Input

The time and cost of developing applications that can be easily ported across architectures has been a barrier to innovation, and has kept developers from taking advantage of new technologies that can speed up their code.

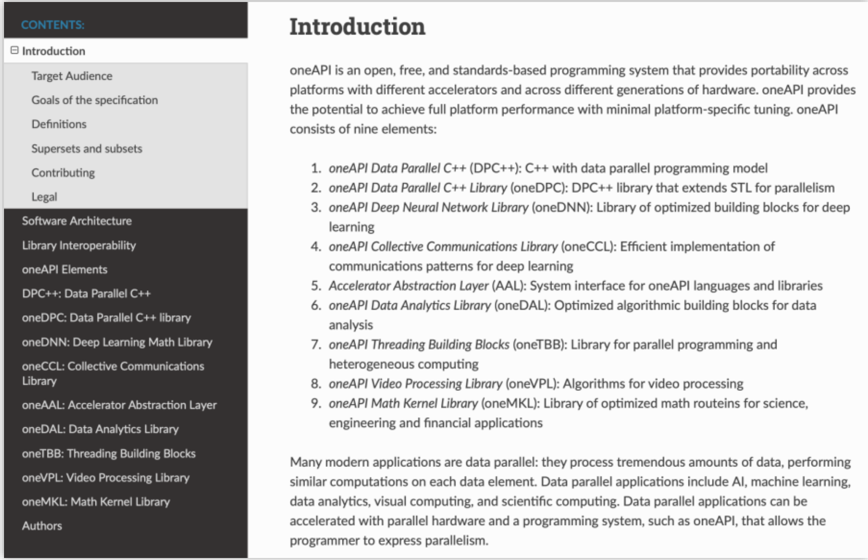
It's time to change that. We invite radical collaboration from across the ecosystem to create a shared industry spec that gives developers the power and flexibility to create fast, innovative, data-centric solutions.



oneAPI Specification source files https://spec.oneapi.com

rscohn2 warnings: empty container (#105) ✓ Latest commit 5ec7b10 2 hours ago

.github/workflows	Full names include oneAPI, add spellchecker (#89)	4 days ago
docker	initial	2 months ago
scripts	add imagemagick, disable mkl tarball (#96)	3 days ago
site-root	fix links (#24)	2 months ago
source	warnings: empty container (#105)	2 hours ago
.gitignore	Initial version of K-Means spec	11 days ago
.gitlab-ci-onetbb.yml	Change how standalone oneTBB and oneDPL specs are built (#97)	2 days ago


CONTENTS:

Introduction

oneAPI is an open, free, and standards-based programming system that provides portability across platforms with different accelerators and across different generations of hardware. oneAPI provides the potential to achieve full platform performance with minimal platform-specific tuning. oneAPI consists of nine elements:

1. *oneAPI Data Parallel C++* (DPC++): C++ with data parallel programming model
2. *oneAPI Data Parallel C++ Library* (oneDPC): DPC++ library that extends STL for parallelism
3. *oneAPI Deep Neural Network Library* (oneDNN): Library of optimized building blocks for deep learning
4. *oneAPI Collective Communications Library* (oneCCL): Efficient implementation of communications patterns for deep learning
5. *Accelerator Abstraction Layer* (AAL): System interface for oneAPI languages and libraries
6. *oneAPI Data Analytics Library* (oneDAL): Optimized algorithmic building blocks for data analysis
7. *oneAPI Threading Building Blocks* (oneTBB): Library for parallel programming and heterogeneous computing
8. *oneAPI Video Processing Library* (oneVPL): Algorithms for video processing
9. *oneAPI Math Kernel Library* (oneMKL): Library of optimized math routeins for science, engineering and financial applications

Many modern applications are data parallel: they process tremendous amounts of data, performing similar computations on each data element. Data parallel applications include AI, machine learning, data analytics, visual computing, and scientific computing. Data parallel applications can be accelerated with parallel hardware and a programming system, such as oneAPI, that allows the programmer to express parallelism.

We encourage feedback on the oneAPI Specification from organizations and individuals

Data parallel C++

Standards-based, Cross-architecture Language

Language to deliver uncompromised parallel programming productivity and performance across CPUs and accelerators

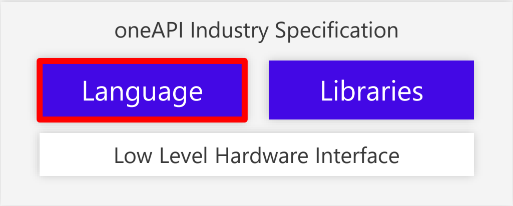
- DPC++ = ISO C++ and Khronos SYCL and Extensions
- Allows code reuse across hardware targets, while permitting custom tuning for a specific accelerator
- Open, cross-industry alternative to single architecture proprietary language

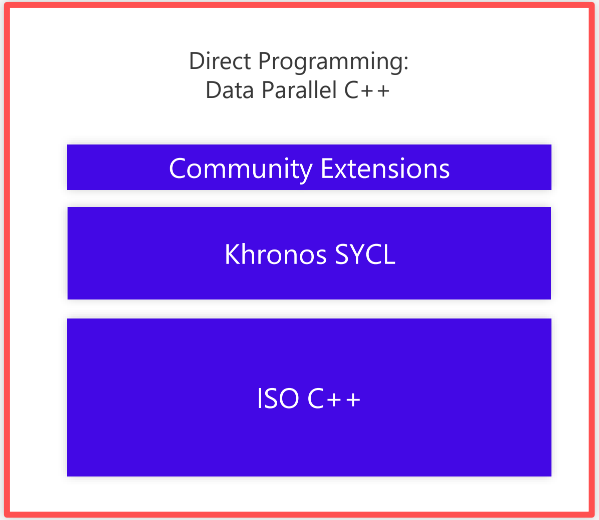
Based on C++

- Delivers C++ productivity benefits, using common and familiar C and C++ constructs
- Incorporates SYCL* from the Khronos Group to support data parallelism and heterogeneous programming

Community Project to drive language enhancements

- Extensions to simplify data parallel programming
- Open and cooperative development for continued evolution
- DPC++ extensions including Unified Shared Memory are being incorporated into upcoming versions of the Khronos SYCL standard.



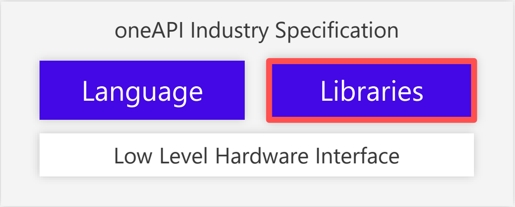


Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products.

oneAPI Specification Libraries

Key domain-specific functions to accelerate compute intensive workloads

Custom-coded for supported architectures



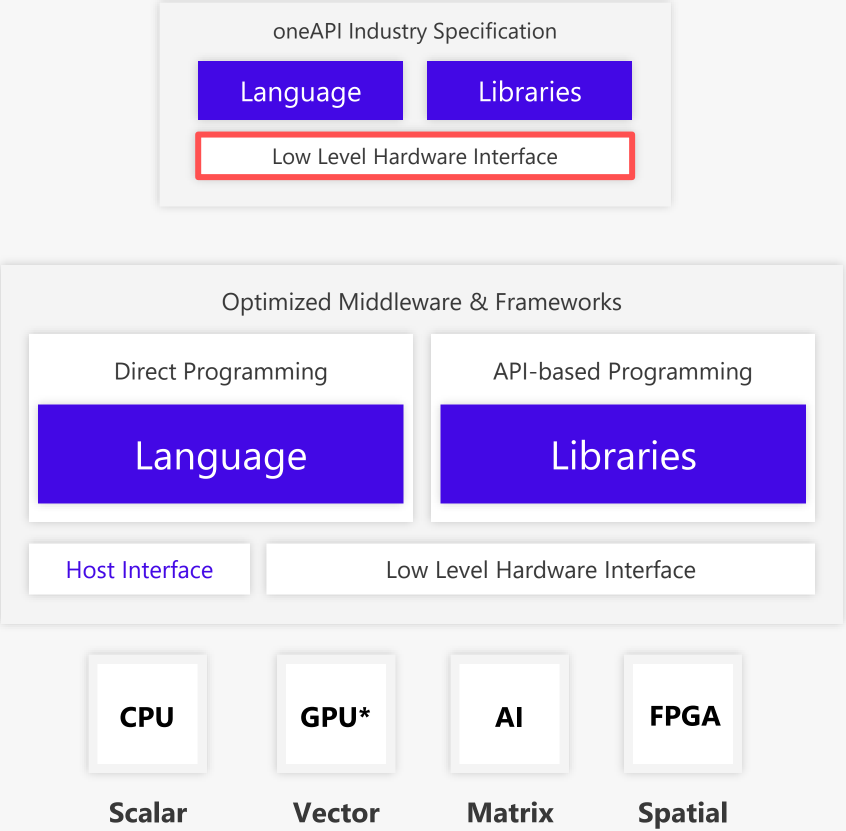
Library Name	Description	Short name
oneAPI DPC++ Library	Key algorithms and functions to speed up DPC++ kernel programming	oneDPC
oneAPI Math Kernel Library	Math routines including matrix algebra, fast Fourier transforms (FFT), and vector math	oneMKL
oneAPI Data Analytics Library	Machine learning and data analytics functions	oneDAL
oneAPI Deep Neural Network Library	Neural networks functions for deep learning training and inference	oneDNN
oneAPI Collective Communications Library	Communication patterns for distributed deep learning	oneCCL
oneAPI Threading Building Blocks	Threading and memory management template library	oneTBB
oneAPI Video Processing Library	Real-time video decoding, encoding, transcoding, and processing functions	oneVPL

Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products

oneAPI Level Zero

Hardware abstraction layer for low-level low-latency accelerator programming control

Target: Hardware and OS vendors who would like to implement oneAPI specification; as well as runtime developers for other languages



oneAPI Industry Specification

Language

Libraries

Low Level Hardware Interface

Optimized Middleware & Frameworks

Direct Programming

Language

API-based Programming

Libraries

Host Interface

Low Level Hardware Interface

CPU

GPU*

AI

FPGA

Scalar

Vector

Matrix

Spatial

Refer to http://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products

* Current version supports GPU

oneAPI initiative – Ecosystem support

allegro.ai

CINECA

GIGASPACES

SAP

Taboola

AI SINGAPORE

Hewlett Packard Enterprise

Lenovo

Tencent 腾讯

rENIAC

Argonne NATIONAL LABORATORY

CERN openlab

codeplay

MEGWARE

PHILIPS

sas

ATLAS EXPERIMENT

सी-डैक CDAC CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

IISER PUNE

Laboratório Nacional de Computação Científica

SENAI CIMATEC SISTEMA FIEB Federação das Indústrias do Estado da Bahia

SUSE We adapt. You succeed.

Ben-Gurion University of the Negev

Indian Institute of Technology Delhi

HCL

MEGH COMPUTING

Tech Mahindra

UNIVERSITY OF CAMBRIDGE

ANSYS

INDIAN INSTITUTE OF SCIENCE

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR 1951 योगः कर्मसु कौशलम्

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

Stockholm University

KTH VETENSKAP OCH KONST

ZUSE INSTITUTE BERLIN

These organizations support the oneAPI initiative ‘concept’ for a single, unified programming model for cross-architecture development. It does not indicate any agreement to purchase or use of Intel’s products

Upcoming relevant DPC++ and SYCL talks

Date Start Time	End Time	Title	Presenter
Wed 14th: 11:30	12:00	SYCL Performance and Portability	Kumudha Narasimhan
Thurs 15th: 12:30	14:00	**Tutorials:** OneAPI/ DPC++ Essential Series hands on (Through Friday) oneAPI Intro Module: (This module is used to introduce oneAPI, DPC++ Hello World and Intel DevCloud) DPC++ Program Structure: (Classes - device, device_selector, queue, basic kernels and ND-Range kernels, Buffers-Accessor memory model, DPC++ Code Anatomy)	Praveen Kundurthy
Thurs 15th: 14:15	15:15	**Tutorial:** DPC++ New Features - Unified Shared Memory (USM), Sub-Groups (Intel oneAPI DPC++ Library -Usage of oneDPL, Buffer Iterators and oneDPL with USM)	Praveen Kundurthy
Friday 16th		Full afternoon of tutorial sessions on developing with SYCL using DPC++ including how to run this code on Nvidia hardware	
Monday 19th		BYOC – Bring your own code along to this Intel workshop and work to bring it to oneAPI and DPC++.	

Describing Parallelism

How do you represent the different forms of parallelism?

- Directive vs explicit parallelism
- Task vs data parallelism
- Queue vs stream execution

Directive vs Explicit Parallelism

Examples:

- OpenMP, OpenACC

Implementation:

- Compiler transforms code to be parallel based on pragmas

Here we're using OpenMP as an example

```
vector<float> a, b, c;

#pragma omp parallel for
for(int i = 0; i < a.size(); i++) {
  c[i] = a[i] + b[i];
}
```

Examples:

- SYCL, CUDA, TBB, Fibers, C++11 Threads

Implementation:

- An API is used to explicitly enqueuer one or more threads

Here we're using C++ AMP as an example

```
array_view<float> a, b, c;
extent<2> e(64, 64);
parallel_for_each(e, [=](index<2> idx)
restrict(amp) {
  c[idx] = a[idx] + b[idx];
});
```

Task vs Data Parallelism

Examples:

- OpenMP, C++11 Threads, TBB

Implementation:

- Multiple (potentially different) tasks are performed in parallel

Here we're using TBB as an example

```
vector<task> tasks = { … };

tbb::parallel_for_each(tasks.begin(),
    tasks.end(), [=](task &v) {
  task();
});
```

Examples:

- C++ AMP, CUDA, SYCL, C++17 ParallelSTL

Implementation:

- The same task is performed across a large data set

Here we're using CUDA as an example

```
float *a, *b, *c;
cudaMalloc((void **)&a, size);
cudaMalloc((void **)&b, size);
cudaMalloc((void **)&c, size);

vec_add<<<64, 64>>>(a, b, c);
```

Queue vs Stream Execution

Examples:

- C++ AMP, CUDA, SYCL, C++17 ParallelSTL

Implementation:

- Functions are placed in a queue and executed once per enqueuer

Here we're using CUDA as an example

```
float *a, *b, *c;
cudaMalloc((void **)&a, size);
cudaMalloc((void **)&b, size);
cudaMalloc((void **)&c, size);

vec_add<<<64, 64>>>(a, b, c);
```

Examples:

- BOINC, BrookGPU

Implementation:

- A function is executed on a continuous loop on a stream of data

Here we're using BrookGPU as an example

```
reduce void sum (float a<>,
                 reduce float r<>) {
  r += a;
}
float a<100>;
float r;
sum(a,r);
```

Data Locality & Movement

One of the biggest limiting factor in parallel and heterogeneous computing

- Cost of data movement in time and power consumption

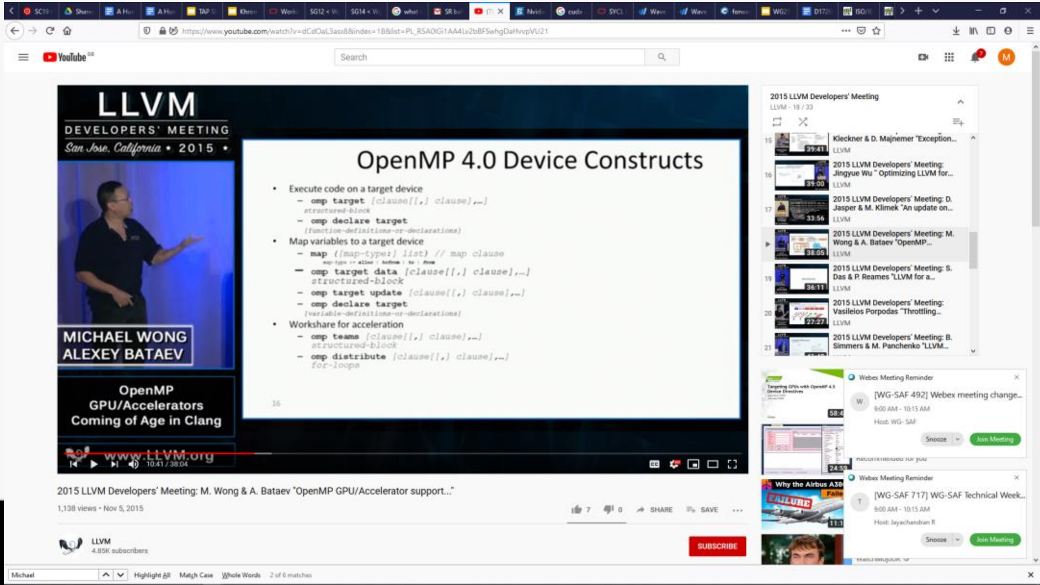
Cost of Data Movement

- It can take considerable time to move data to a device
  - This varies greatly depending on the architecture
- The bandwidth of a device can impose bottlenecks
  - This reduces the amount of throughput you have on the device
- Performance gain from computation > cost of moving data
  - If the gain is less than the cost of moving the data it’s not worth doing
- Many devices have a hierarchy of memory regions
  - Global, read-only, group, private
  - Each region has different size, affinity and access latency
  - Having the data as close to the computation as possible reduces the cost

https://www.youtube.com/watch?v=dCdOaL3asx8&index=18&list=PL_R5A0lGi1AA4Lv2bBFSwhgDaHvvpVU21

How do you move data from the host CPU to a device and back?

➢ Implicit vs explicit data movement



OpenMP 4.0 Device Constructs

- Execute code on a target device
  - **omp target** *[clause[[,] clause],…]* *structured-block*
  - **omp declare target** *[function-definitions-or-declarations]*
- Map variables to a target device
  - **map** *([map-type:] list) // map clause* map-type := **alloc** | **tofrom** | **to** | **from**
  - **omp target data** *[clause[[,] clause],…]* *structured-block*
  - **omp target update** *[clause[[,] clause],…]*
  - **omp declare target** *[variable-definitions-or-declarations]*
- Workshare for acceleration
  - **omp teams** *[clause[[,] clause],…]* *structured-block*
  - **omp distribute** *[clause[[,] clause],…]* *for-loops*

16

2015 LLVM Developers' Meeting: M. Wong & A. Bataev "OpenMP GPU/Accelerator support..."

1,138 views • Nov 5, 2015

LLVM
4.85K subscribers

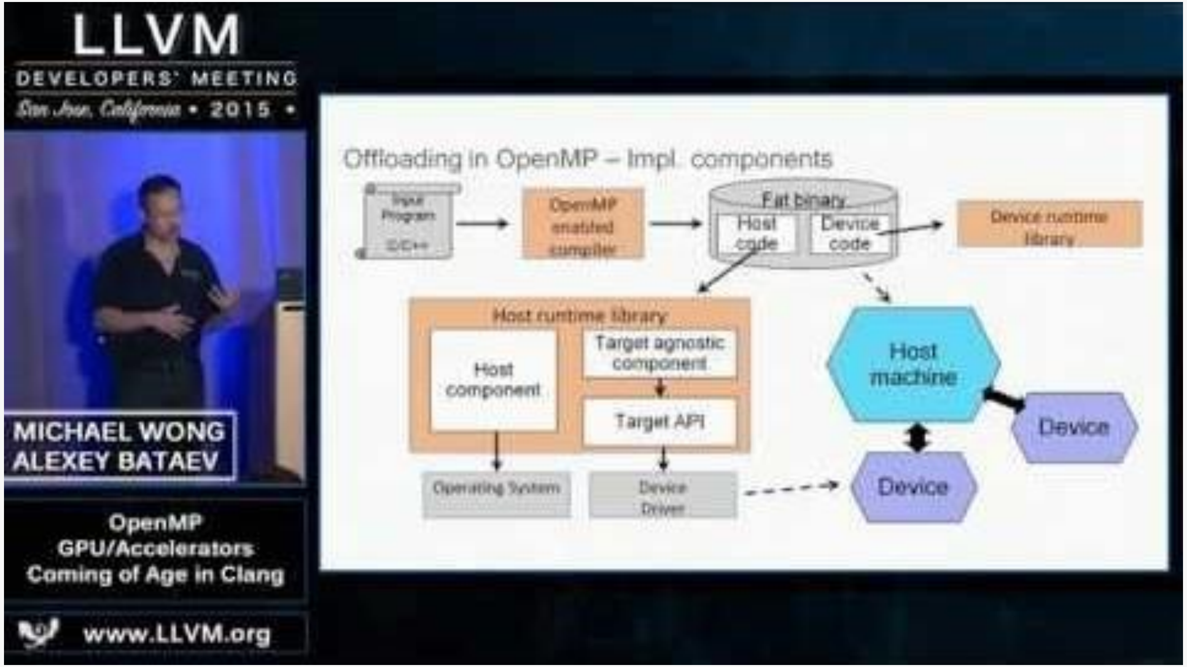
2015 LLVM Developers' Meeting
LLVM - 18 / 33

15 Kleckner & D. Majnemer "Exception... LLVM
16 2015 LLVM Developers' Meeting: Jingyue Wu " Optimizing LLVM for... LLVM
17 2015 LLVM Developers' Meeting: D. Jasper & M. Klimek "An update on... LLVM
▶ 2015 LLVM Developers' Meeting: M. Wong & A. Bataev "OpenMP... LLVM
19 2015 LLVM Developers' Meeting: S. Das & P. Reames "LLVM for a... LLVM
20 2015 LLVM Developers' Meeting: Vasileios Porpodas "Throttling... LLVM
21 2015 LLVM Developers' Meeting: B. Simmers & M. Panchenko "LLVM...

Webex Meeting Reminder
[WG-SAF 492] Webex meeting change...

Webex Meeting Reminder
[WG-SAF 717] WG-SAF Technical Week...

Michael   Highlight All   Match Case   Whole Words   2 of 6 matches



LLVM

DEVELOPERS' MEETING

San Jose, California • 2015 •

MICHAEL WONG
ALEXEY BATAEV

OpenMP
GPU/Accelerators
Coming of Age in Clang

www.LLVM.org

Offloading in OpenMP – Impl. components

Implicit vs Explicit Data Movement

Examples:

- SYCL, C++ AMP

Implementation:

- Data is moved to the device implicitly via cross host CPU / device data structures

Here we’re using C++ AMP as an example

```
array_view<float> ptr;
extent<2> e(64, 64);
parallel_for_each(e, [=](index<2> idx)
restrict(amp) {
  ptr[idx] *= 2.0f;
});
```

Examples:

- OpenCL, CUDA, OpenMP

Implementation:

- Data is moved to the device via explicit copy APIs

Here we’re using CUDA as an example

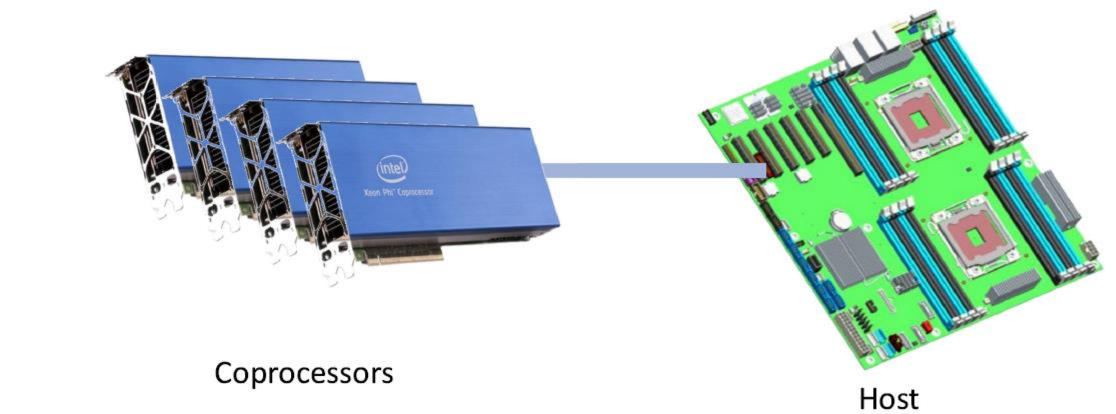
```
float *h_a = { … }, d_a;
cudaMalloc((void **)&d_a, size);
cudaMemcpy(d_a, h_a, size,
  cudaMemcpyHostToDevice);
vec_add<<<64, 64>>>(a, b, c);
cudaMemcpy(d_a, h_a, size,
  cudaMemcpyDeviceToHost);
```

Act 2

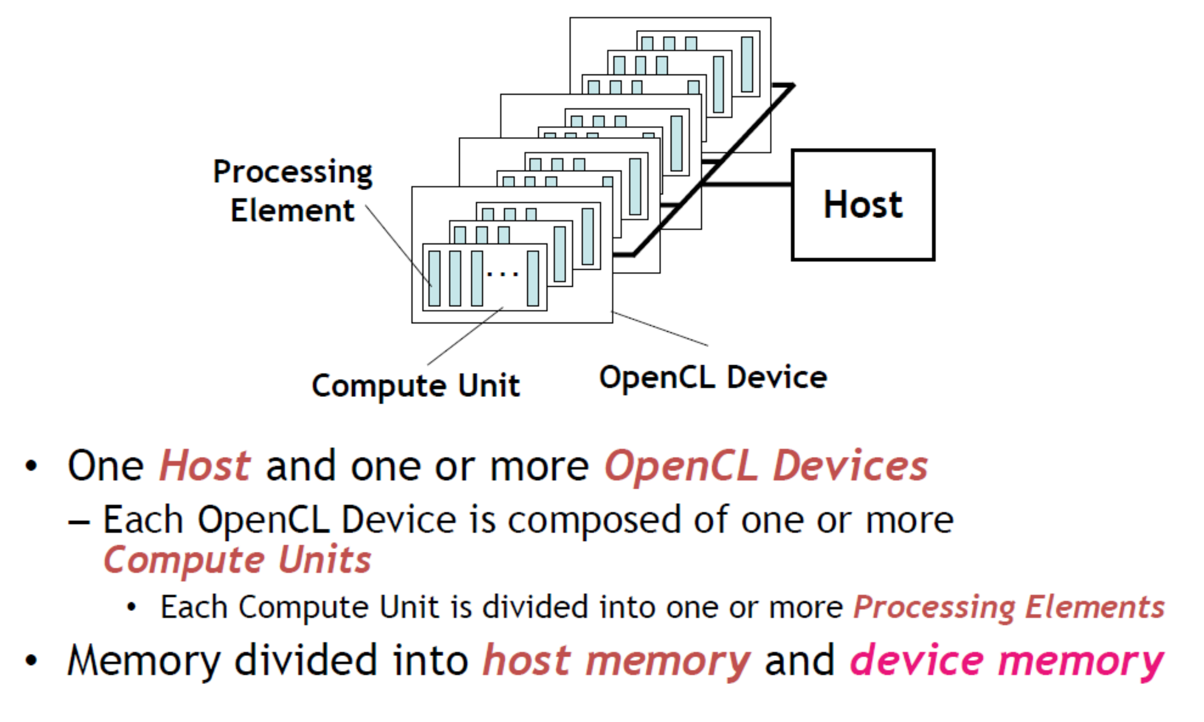
OpenMP Accelerator and Data Movement
(WARNING: this is OpenMP 4, latest OpenMP will have new additions and changes)

Device Model

- One host
- Multiple accelerators/coprocessors

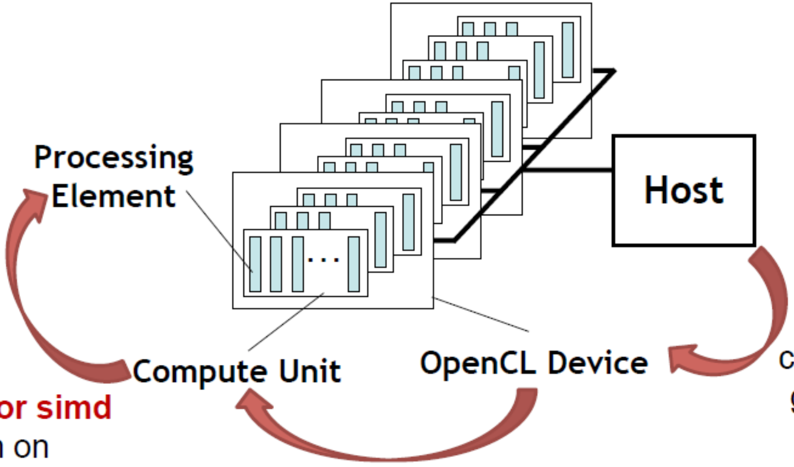


OpenCL Platform Model



- One *Host* and one or more *OpenCL Devices*
  - Each OpenCL Device is composed of one or more *Compute Units*
    - Each Compute Unit is divided into one or more *Processing Elements*
- Memory divided into *host memory* and *device memory*

OpenCL and OpenMP Platform Model



Target construct to get onto a device

Parallel for simd to run on processing elements + vector units

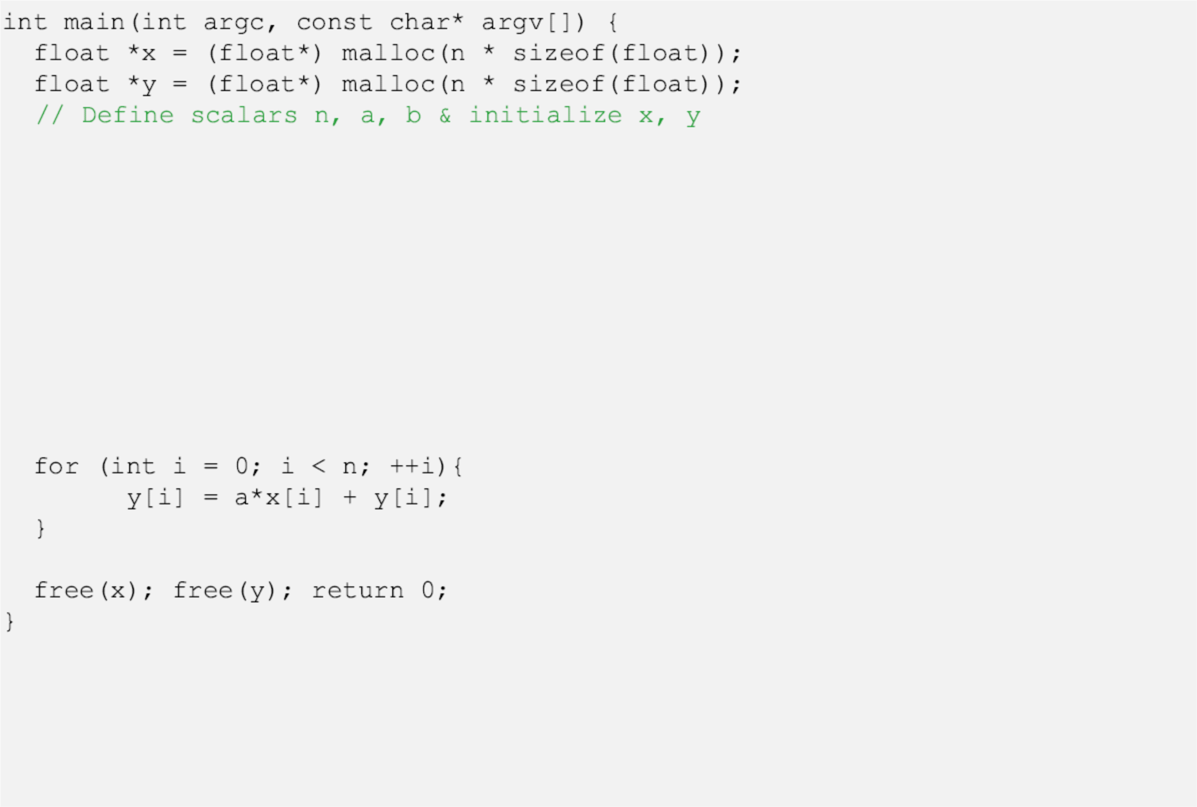
Teams construct to create a league of teams with one team of threads on each compute unit.

Distribute clause to assign work-groups to teams.

OpenMP SAXPY Implementation for CPU

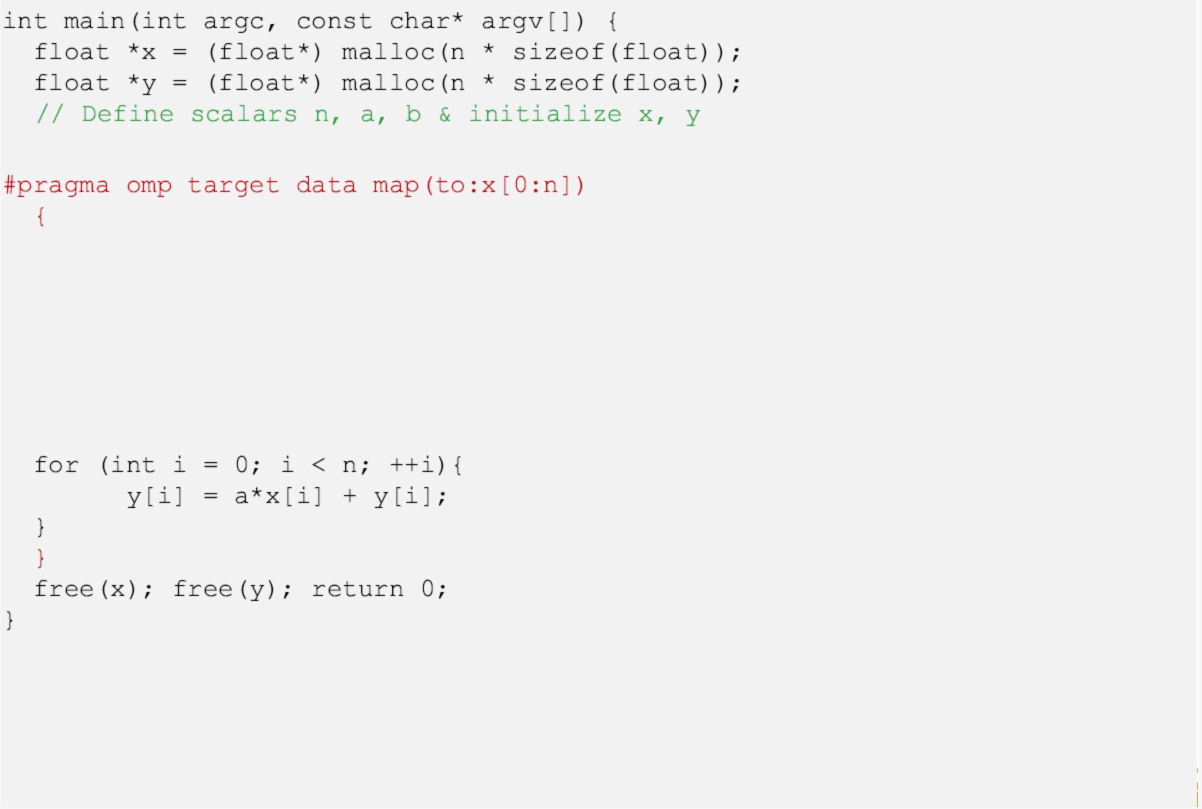
```
void saxpy_openmp(
    int n,      // the number of elements in the vectors
    float a,    // scale factor
    float x[],  // the first input vector
    float y[]   // the output vector and second input vector
) {
#pragma omp parallel for
    for (int i = 0; i < n; ++i)
        y[i] = a * x[i] + y[i];
}
```

SAXPY: Serial (host)



```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y











  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

  free(x); free(y); return 0;
}
```

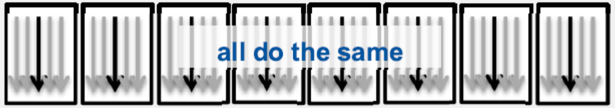
SAXPY: Serial (host)



```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
  {







  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
  }
  free(x); free(y); return 0;
}
```

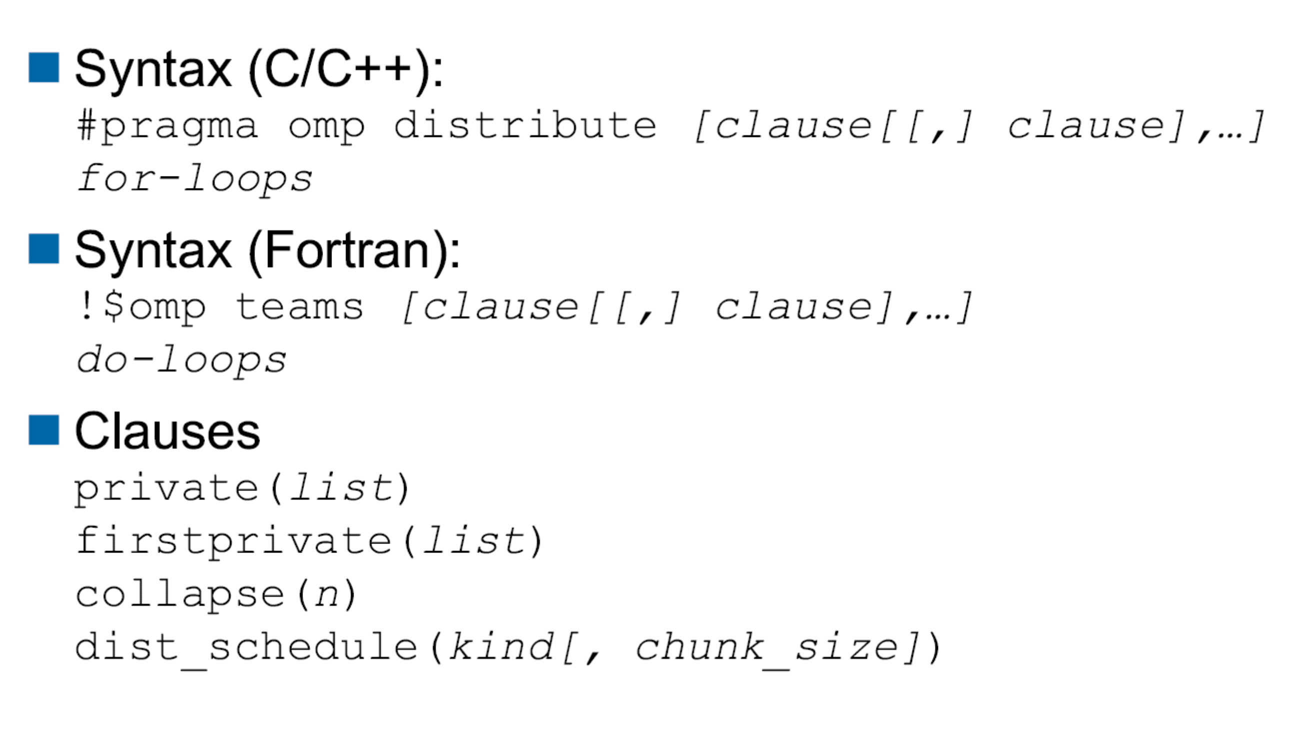
SAXPY: Coprocessor/Accelerator

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
  {
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) num_threads(nthreads)
```



```
  for (int i = 0; i < n; i += num_blocks){
    for (int j = i; j < i + num_blocks; j++) {
        y[j] = a*x[j] + y[j];
  } }
  }
  free(x); free(y); return 0;
}
```

distribute Construct



- Syntax (C/C++):

  `#pragma omp distribute` *`[clause[[,] clause],…]`*
  `for-loops`

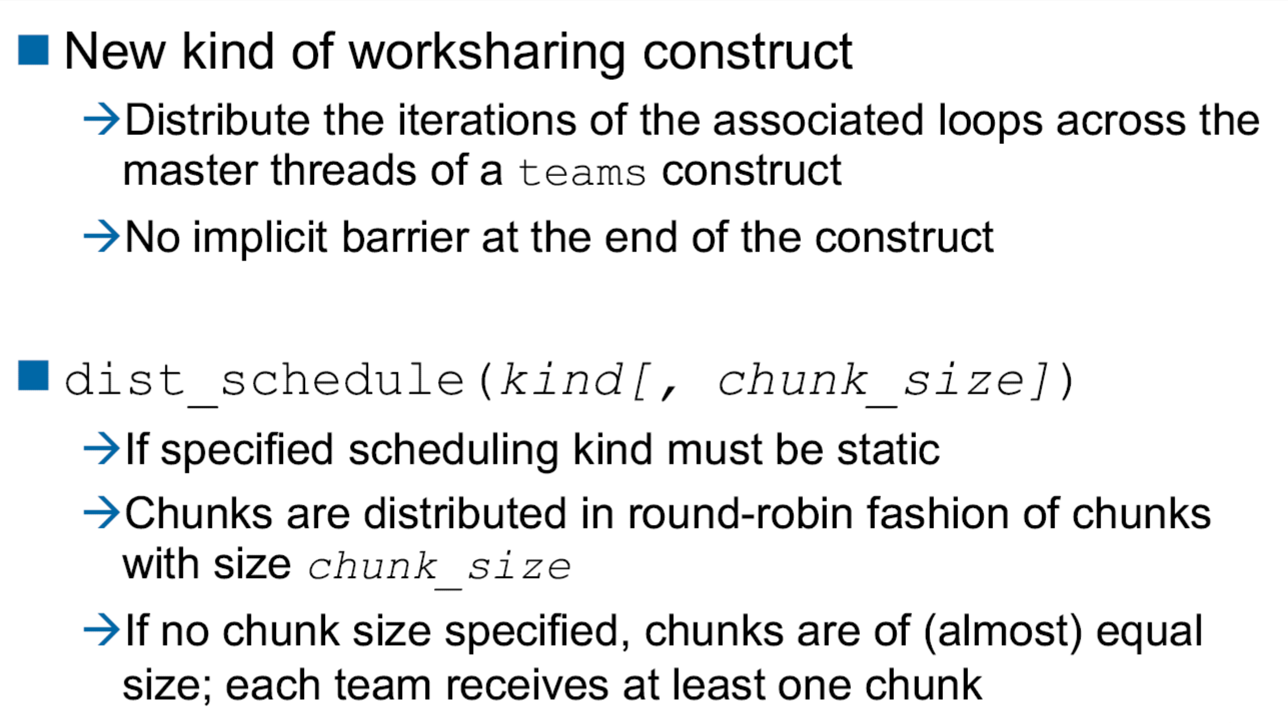
- Syntax (Fortran):

  `!$omp teams` *`[clause[[,] clause],…]`*
  `do-loops`

- Clauses

```
private(list)
firstprivate(list)
collapse(n)
dist_schedule(kind[, chunk_size])
```

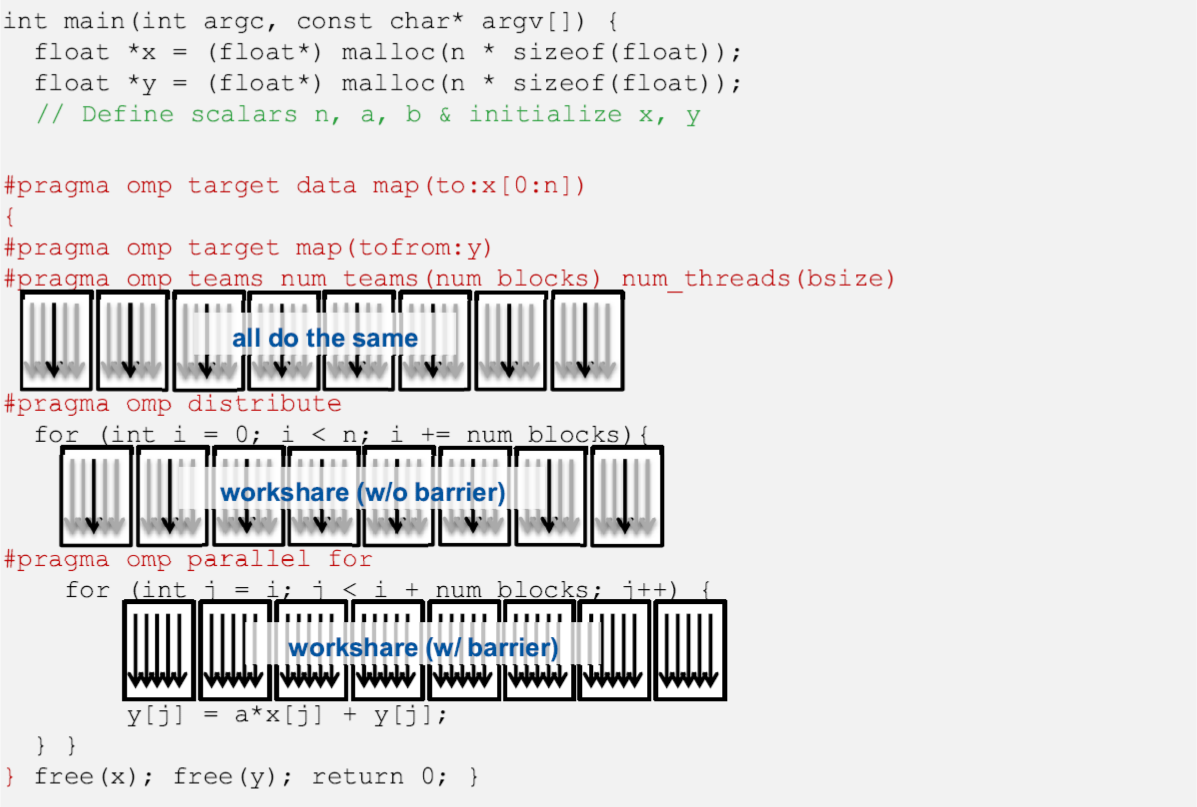
distribute Construct



- New kind of worksharing construct
  - → Distribute the iterations of the associated loops across the master threads of a `teams` construct
  - → No implicit barrier at the end of the construct

- dist_schedule(*kind[, chunk_size]*)
  - → If specified scheduling kind must be static
  - → Chunks are distributed in round-robin fashion of chunks with size *chunk_size*
  - → If no chunk size specified, chunks are of (almost) equal size; each team receives at least one chunk

SAXPY: Coprocessor/Accelerator



```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
{
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
#pragma omp distribute
  for (int i = 0; i < n; i += num_blocks){
#pragma omp parallel for
    for (int j = i; j < i + num_blocks; j++) {
        y[j] = a*x[j] + y[j];
  } }
} free(x); free(y); return 0; }
```

Act 3

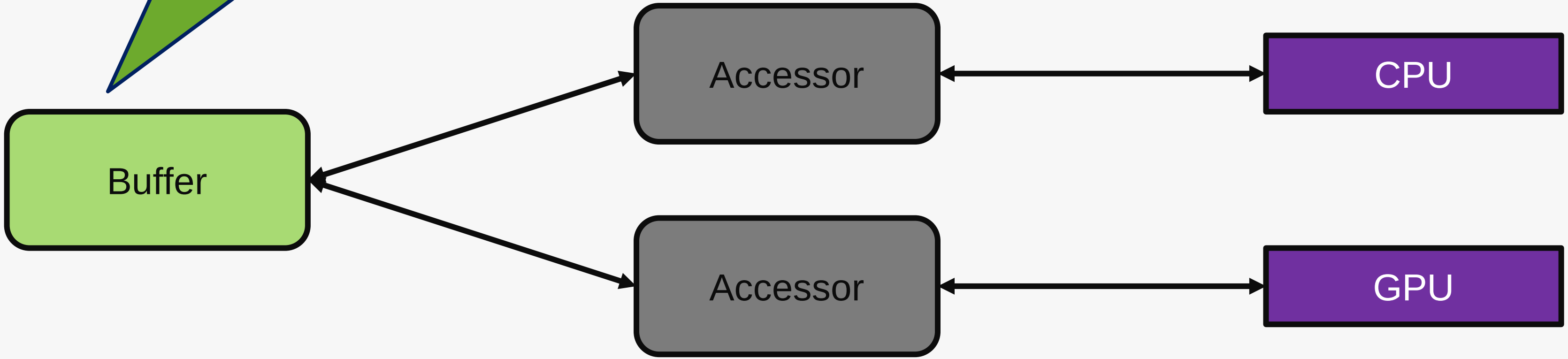
SYCL Accelerator and Data Movement: Accessors and USM

SYCL aims to make data locality and movement efficient

- SYCL separates data storage from data access
- SYCL has separate structures for accessing data in different address spaces
- SYCL allows you to create data dependency graphs

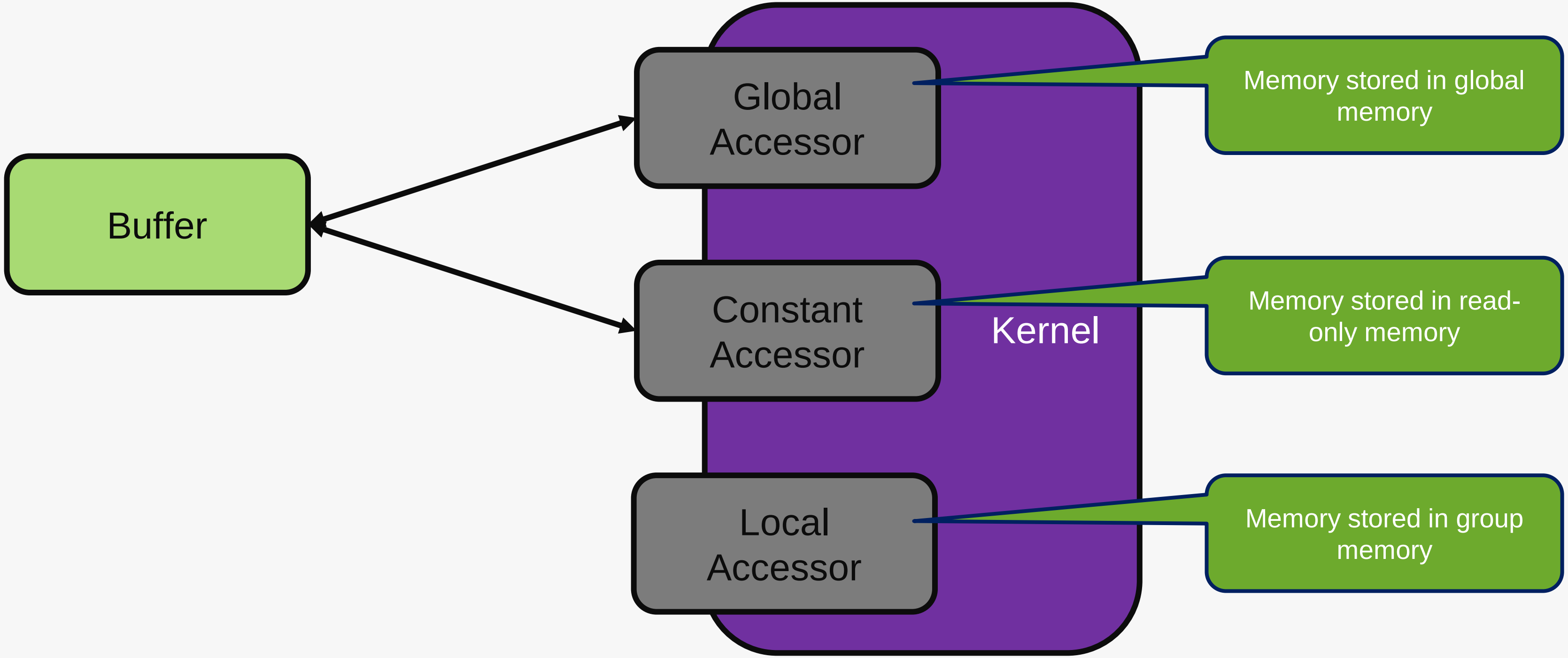
Separating Data & Access

Buffers allow type safe access across host and device



Accessors are used to detect dependencies

Copying/Allocating Memory in Address Spaces

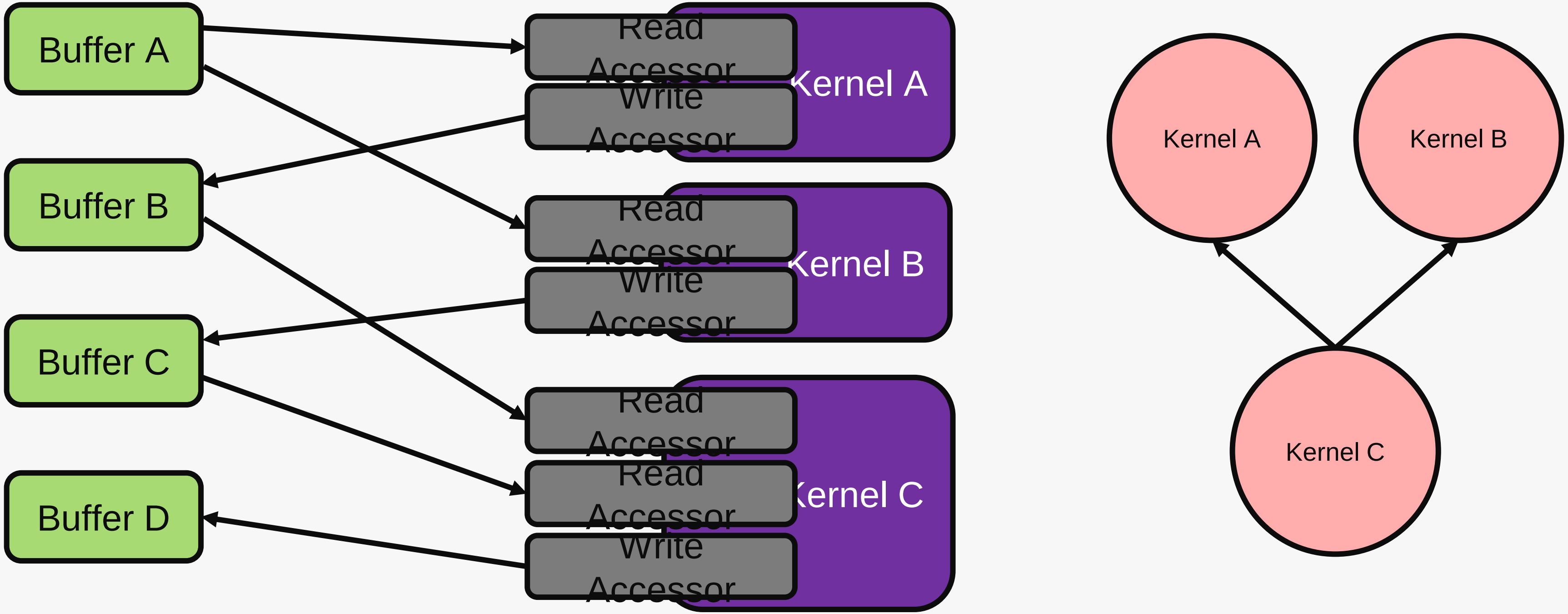


Memory stored in global memory

Memory stored in read-only memory

Memory stored in group memory

Data Dependency Task Graphs



Benefits of Data Dependency Graphs

- Allows you to describe your problems in terms of relationships
  - Don’t need to en-queue explicit copies
- Synchronisation can be performed using RAII
  - Automatically copy data back to the host if necessary
- Removes the need for complex event handling
  - Dependencies between kernels are automatically constructed
- Allows the runtime to make data movement optimizations
  - Pre-emptively copy data to a device before kernels
  - Avoid unnecessary copying data back to the host after kernels

So what does SYCL look like?

- Here is a simple example SYCL application; a vector add

Example: Vector Add

Example: Vector Add

```
#include <CL/sycl.hpp>


template <typename T>
void parallel_add(T *inputA, T *inputB, T *output, size_t size) {













}
```

Include sycl.hpp for the whole SYCL runtime

Example: Vector Add

```
#include <CL/sycl.hpp>


template <typename T>
void parallel_add(T *inputA, T *inputB, T *output, size_t size) {
  cl::sycl::buffer<T, 1> inputABuf(inputA, size);
  cl::sycl::buffer<T, 1> inputBBuf(inputB, size);
  cl::sycl::buffer<T, 1> outputBuf(output, size);









}
```

Create buffers to maintain the data across host and device

The buffers synchronise upon destruction

Example: Vector Add

```
#include <CL/sycl.hpp>


template <typename T>
void parallel_add(T *inputA, T *inputB, T *output, size_t size) {
  cl::sycl::buffer<T, 1> inputABuf(inputA, size);
  cl::sycl::buffer<T, 1> inputBBuf(inputB, size);
  cl::sycl::buffer<T, 1> outputBuf(output, size);
  cl::sycl::queue defaultQueue;








}
```

Create a queue to en-queue work

Example: Vector Add

```
#include <CL/sycl.hpp>


template <typename T>
void parallel_add(T *inputA, T *inputB, T *output, size_t size) {
  cl::sycl::buffer<T, 1> inputABuf(inputA, size);
  cl::sycl::buffer<T, 1> inputBBuf(inputB, size);
  cl::sycl::buffer<T, 1> outputBuf(output, size);
  cl::sycl::queue defaultQueue;
  defaultQueue.submit([&] (cl::sycl::handler &cgh) {







  });
}
```

Create a command group to define an asynchronous task

The scope of the command group is defined by a lambda

Example: Vector Add

```
#include <CL/sycl.hpp>


template <typename T>
void parallel_add(T *inputA, T *inputB, T *output, size_t size) {
  cl::sycl::buffer<T, 1> inputABuf(inputA, size);
  cl::sycl::buffer<T, 1> inputBBuf(inputB, size);
  cl::sycl::buffer<T, 1> outputBuf(output, size);
  cl::sycl::queue defaultQueue;
  defaultQueue.submit([&] (cl::sycl::handler &cgh) {
    auto inputAPtr = inputABuf.get_access<cl::sycl::access::read>(cgh);
    auto inputBPtr = inputBBuf.get_access<cl::sycl::access::read>(cgh);
    auto outputPtr = outputBuf.get_access<cl::sycl::access::write>(cgh);




  });
}
```

Create accessors to give access to the data on the device

Example: Vector Add

```
#include <CL/sycl.hpp>
template <typename T> kernel;

template <typename T>
void parallel_add(T *inputA, T *inputB, T *output, size_t size) {
  cl::sycl::buffer<T, 1> inputABuf(inputA, size);
  cl::sycl::buffer<T, 1> inputBBuf(inputB, size);
  cl::sycl::buffer<T, 1> outputBuf(output, size);
  cl::sycl::queue defaultQueue;
  defaultQueue.submit([&] (cl::sycl::handler &cgh) {
    auto inputAPtr = inputABuf.get_access<cl::sycl::access::read>(cgh);
    auto inputBPtr = inputBBuf.get_access<cl::sycl::access::read>(cgh);
    auto outputPtr = outputBuf.get_access<cl::sycl::access::write>(cgh);
    cgh.parallel_for<kernel<T>>(cl::sycl::range<1>(size)),
                                         [=](cl::sycl::id<1> idx) {

    }));
  });
}
```

Create a parallel_for to define a kernel

Example: Vector Add

```
#include <CL/sycl.hpp>
template <typename T> kernel;

template <typename T>
void parallel_add(T *inputA, T *inputB, T *output, size_t size) {
  cl::sycl::buffer<T, 1> inputABuf(inputA, size);
  cl::sycl::buffer<T, 1> inputBBuf(inputB, size);
  cl::sycl::buffer<T, 1> outputBuf(output, size);
  cl::sycl::queue defaultQueue;
  defaultQueue.submit([&] (cl::sycl::handler &cgh) {
    auto inputAPtr = inputABuf.get_access<cl::sycl::access::read>(cgh);
    auto inputBPtr = inputBBuf.get_access<cl::sycl::access::read>(cgh);
    auto outputPtr = outputBuf.get_access<cl::sycl::access::write>(cgh);
    cgh.parallel_for<kernel<T>>(cl::sycl::range<1>(size)),
                                [=](cl::sycl::id<1> idx){
      outputPtr[idx] = inputAPtr[idx] + inputBPtr[idx];
    }));
  });
}
```

You must provide a name for the lambda

Access the data via the accessor’s subscript operator

Example: Vector Add

```
template <typename T>
void parallel_add(T *inputA, T *inputB, T *output, size_t size);

int main() {

  float inputA[count] = { /* input a */ };
  float inputB[count] = { /* input b */ };
  float output[count] = { /* output */ };

  parallel_add(inputA, inputB, output, count);
}
```

The result is stored in output upon returning from parallel_add

How I learn to stop worrying and love pointers

- Pointers are a fact of life in many existing C/C++ codes.
  - Interesting programs operate on more than just Arrays of POD.
  - Rewriting C/C++ programs to augment with buffers/accessors is a pain point for new programmers and large programs.
- Also this along with in-order queues enables porting to from CUDA or any explicit data movement type C++ framework, especially one that is pointer-based program much easier
  - when you have a simple program or don’t care about data dependency
  - when you are building some other framework on top of SYCL that requires explicit control of data movement

Pointers: Deconstruct SYCL Vector Add

```
#include <CL/sycl.hpp>
template <typename T> kernel;

template <typename T>
void parallel_add(T *inputA, T *inputB, T *output, size_t size) {
  cl::sycl::buffer<T, 1> inputABuf(inputA, size);
  cl::sycl::buffer<T, 1> inputBBuf(inputB, size);
  cl::sycl::buffer<T, 1> outputBuf(output, size);
  cl::sycl::queue defaultQueue;
  defaultQueue.submit([&] (cl::sycl::handler &cgh) {
    auto inputAPtr = inputABuf.get_access<cl::sycl::access::read>(cgh);
    auto inputBPtr = inputBBuf.get_access<cl::sycl::access::read>(cgh);
    auto outputPtr = outputBuf.get_access<cl::sycl::access::write>(cgh);
    cgh.parallel_for<kernel<T>>(cl::sycl::range<1>(size)),
                                        [=](cl::sycl::id<1> idx)
      outputPtr[idx] = inputAPtr[idx] + inputBPtr[idx];
    }));
  });
}
```

All our pointers became buffers

Access have to be declared and used instead of pointers

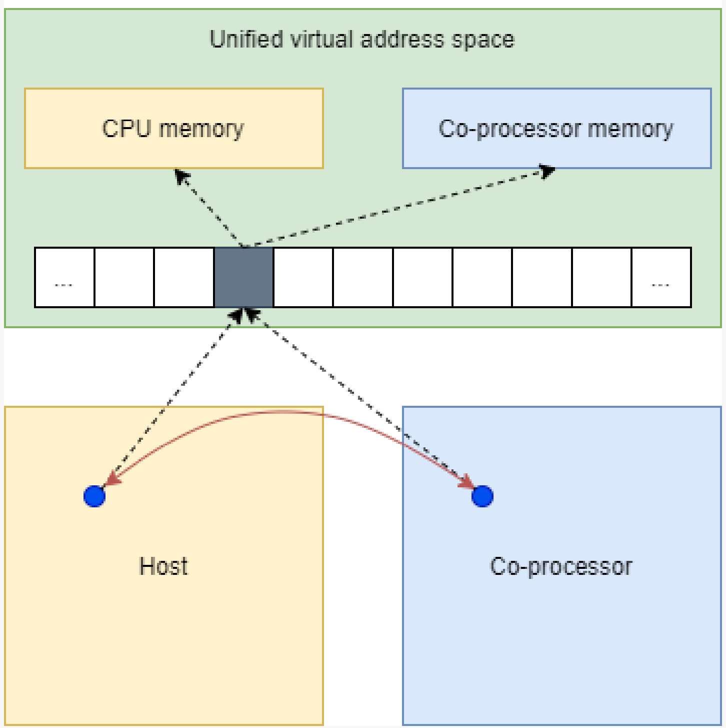
codeplay

What is USM?

- Unified shared memory (USM) is an alternative pointer-based data management model to the accessor-buffer model.
    - Unified virtual address space
    - Pointer-based structures
    - Explicit memory management
    - Shared memory allocations

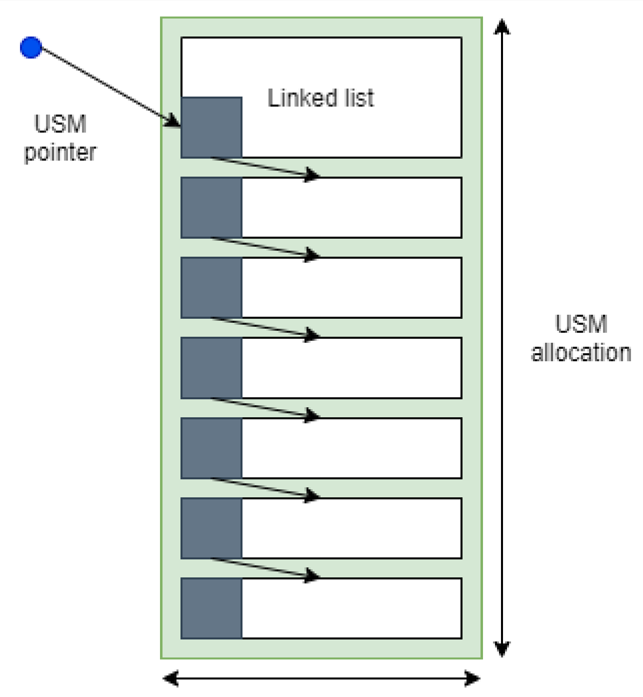
Unified Virtual Address Space

- USM memory allocations return pointers which are consistent between the host application and kernel functions on a device.
- Representing data between the host and device(s) does not require creating accessors.
- Pointer-based API more familiar to C or C++ programmers.



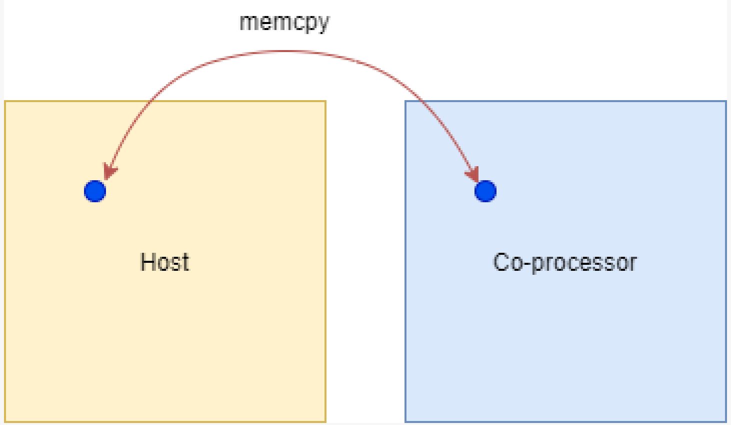
Pointer based structures

- Data is moved between the host and device(s) in a span of memory in bytes rather than a buffer of a specific type.
- Pointers within that region of memory can freely point to any other address in that region.
- Easier to port existing C or C++ code to use SYCL.



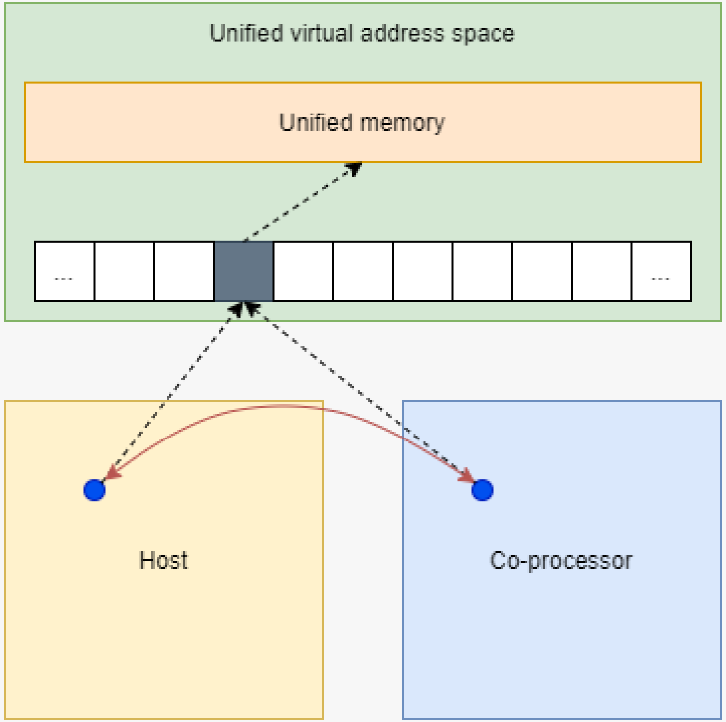
Explicit Memory Management

- Memory is allocated and data is moved using explicit routines.
- Moving data between the host and device(s) does not require accessors or submitting command groups.
- The SYCL runtime will not perform any data dependency analysis, dependencies between commands must be managed manually.



Shared memory allocations

- Some platforms will support variants of USM where memory allocations share the same memory region between the host and device(s).
- No explicit routines are required to move the data between the host and device(s).

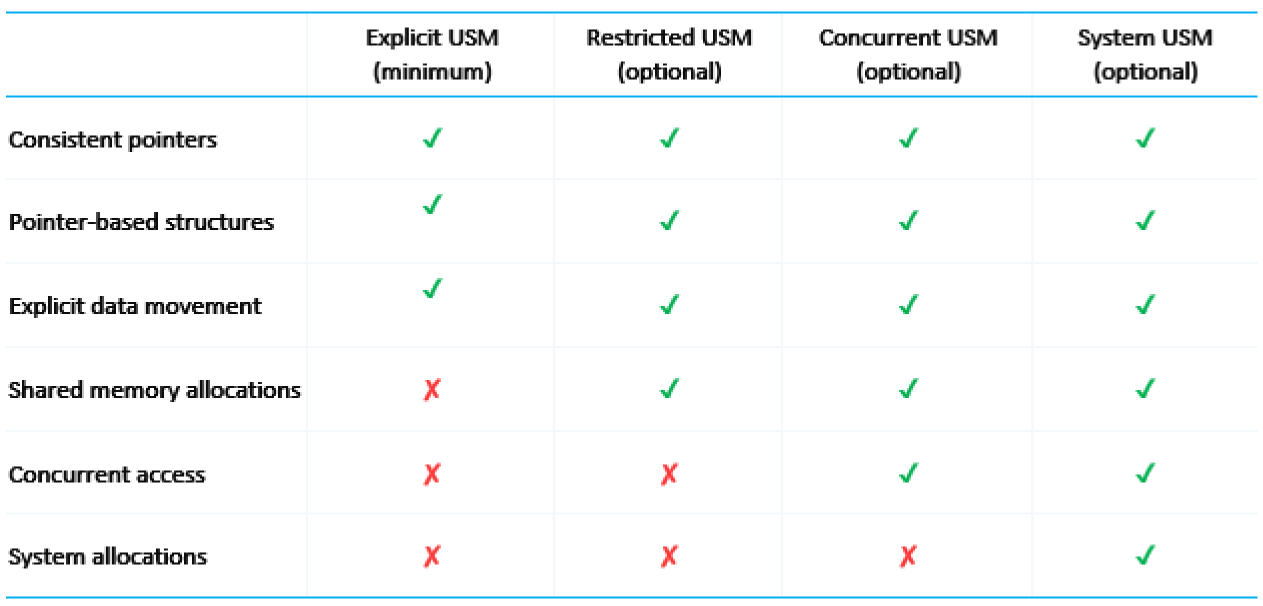


USM allocation types

- USM has three different kinds of memory allocation.
  - A **host** allocation is allocated in host memory.
  - A **device** allocation is allocation in device memory.
  - A **shared** allocation is allocated in shared memory and can migrate back and forth.

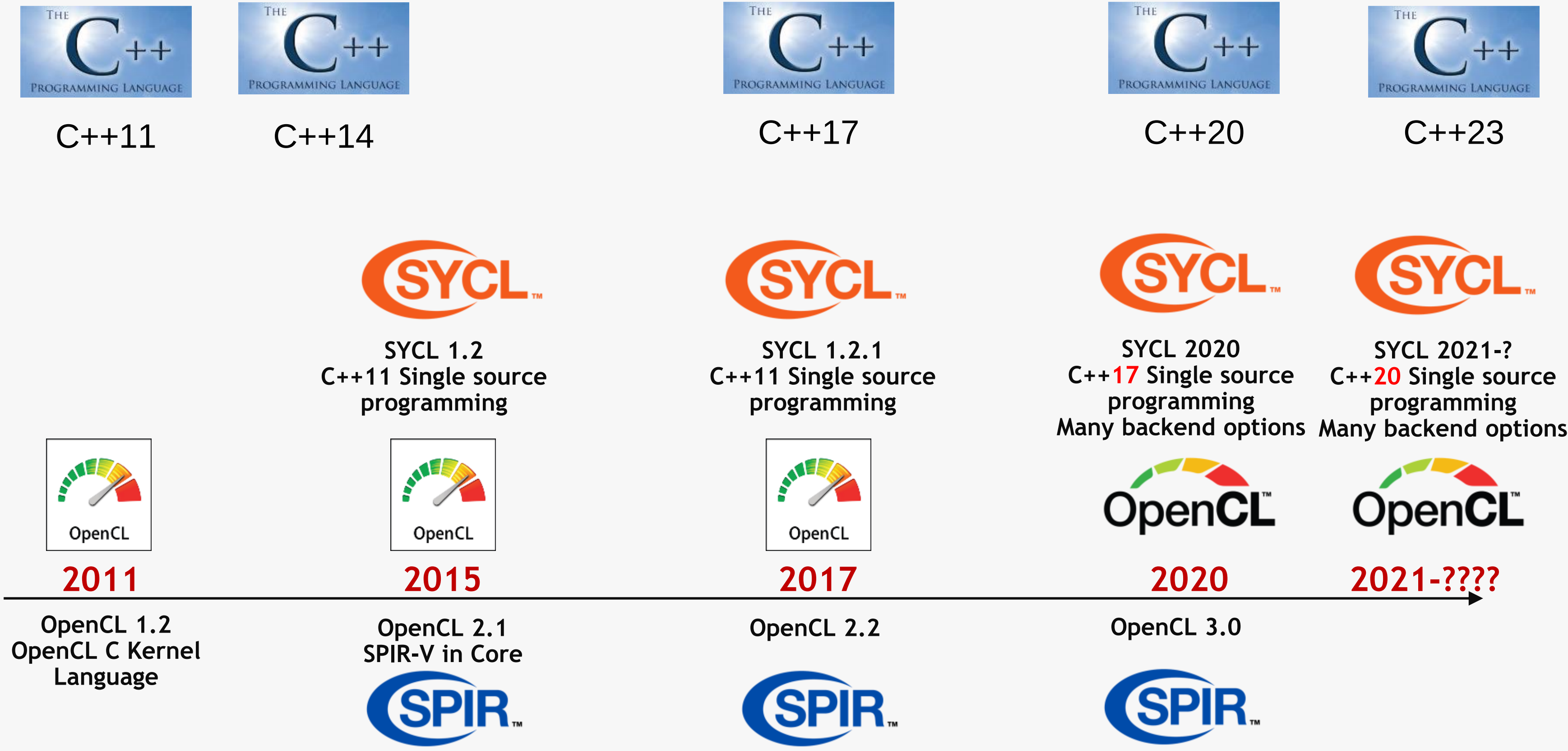
USM variants

- USM has four variants which a platform can support with varying levels of support.
- Each SYCL platform and it's device(s) will support different variants of USM and different kinds of memory allocation.



	Explicit USM (minimum)	Restricted USM (optional)	Concurrent USM (optional)	System USM (optional)
Consistent pointers	✓	✓	✓	✓
Pointer-based structures	✓	✓	✓	✓
Explicit data movement	✓	✓	✓	✓
Shared memory allocations	✗	✓	✓	✓
Concurrent access	✗	✗	✓	✓
System allocations	✗	✗	✗	✓

SYCL Present and Future Roadmap (May Change)



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 2021-?
C++20 Single source programming
Many backend options

2011
OpenCL 1.2
OpenCL C Kernel Language

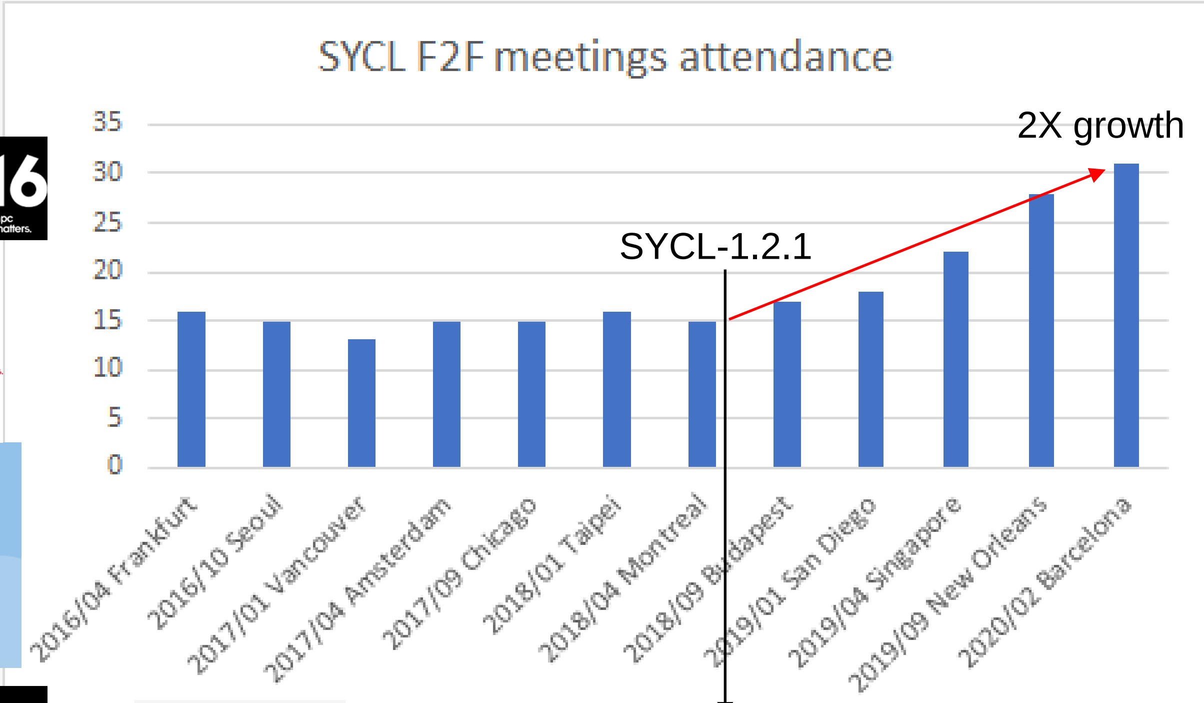
2015
OpenCL 2.1
SPIR-V in Core

2017
OpenCL 2.2

2020
OpenCL 3.0

2021-????

SYCL community is vibrant



SC16 Salt Lake City, Utah | hpc matters.

SC17 Denver, CO | hpc connects.

SC18 Dallas, TX | hpc inspires.

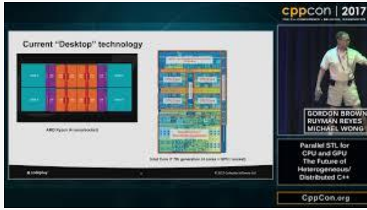
SC19 Denver, CO | hpc is now.

DHPC C++
Distributed & Heterogeneous Programming in C/C++

IWOCL
Toronto, Canada
May 16-18, 2017

IWOCL 2018
6th International Workshop on OpenCL
14-16 May, 2018 | Oxford, UK
Including DHPCC++ 2018 | 14 May
OpenCL
SPIR
SYCL

IWOCL
The 7th International Workshop on OpenCL
Incorporating DHPCC++ 2019 - 4th workshop on Distributed & Heterogeneous Programming in C/C++
DHPC C++


cppcon | 2017
Current "Desktop" technology
GORDON BROWN
RUYMAN REYES
MICHAEL WONG
Parallel STL for CPU and GPU The Future of Heterogeneous/Distributed C++
CppCon.org

SYCL Evolution

SYCL 2020 Potential Features

Generalization (a.k.a the Backend Model) presented by Gordon Brown
Unified Shared Memory (USM) presented by James Brodman
Improvement to Program class Modules presented by Gordon Brown
Host Task with Interop presented by Gordon Brown
In order queues, presented by James Brodman

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

2017
SYCL 1.2.1

SYCL 2020 Roadmap (WIP, MAY CHANGE)

Integration of successful Extensions plus new Core functionality

Target 2020
Provisional Q3 then Final Q4

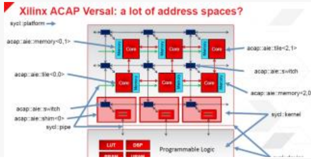
Converge SYCL with ISO C++ and continue to support OpenCL to deploy on more devices
CPU
GPU
FPGA
AI processors
Custom Processors

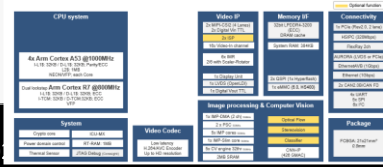
Improving Software Ecosystem
Tool, libraries, GitHub

Expanding Implementation
DPC++
ComputeCpp
triSYCL
hipSYCL

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

Selected Extension Pipeline aiming for SYCL 2020 Provisional Q3
Reduction
Subgroups
Accessor simplification
Atomic rework
Extension mechanism
Address spaces


Xilinx ACAP Versal: a lot of address spaces?



Repeat The Cycle every 1.5-3 years

SYCL Ecosystem, Research and Benchmarks

Implementations

oneAPI

oneAPI

oneAPI

DATA PARALLEL C++

Performant, Productive Programming for Heterogeneous Architecture

ComputeCpp™

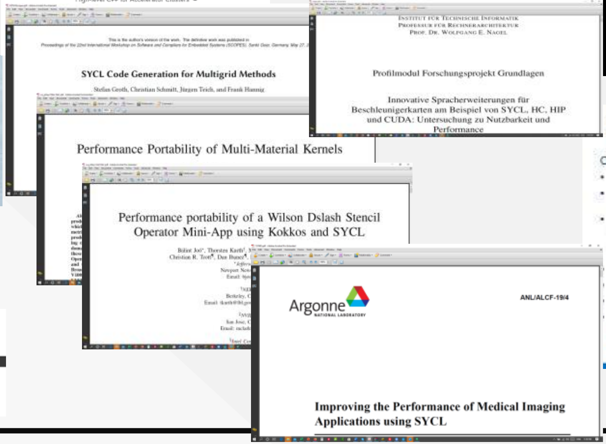
hipSYCL

Research

Celerity

High-level C++ for Accelerator Clusters

EXASCALE COMPUTING PROJECT



This is the author's version of the work. The definitive work was published in Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems (SCOPES), Sankt Goar, Germany, May 27,

SYCL Code Generation for Multigrid Methods

Stefan Groth, Christian Schmitt, Jürgen Teich, and Frank Hannig

Institut für Technische Informatik
Professur für Rechnerarchitektur
Prof. Dr. Wolfgang E. Nagel

Profilmodul Forschungsprojekt Grundlagen

Innovative Spracherweiterungen für Beschleunigerkarten am Beispiel von SYCL, HC, HIP und CUDA: Untersuchung zu Nutzbarkeit und Performance

Performance Portability of Multi-Material Kernels

Performance portability of a Wilson Dslash Stencil Operator Mini-App using Kokkos and SYCL

Bálint Joó, Thorsten Kurth, Christian R. Trott, Dan Ibanez

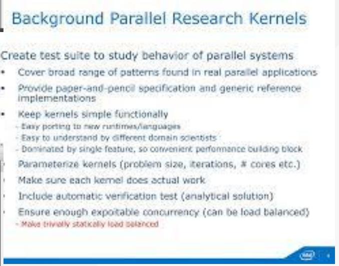
Argonne NATIONAL LABORATORY

ANL/ALCF-19/4

Improving the Performance of Medical Imaging Applications using SYCL

Benchmarks

BabelStream



Background Parallel Research Kernels

Create test suite to study behavior of parallel systems

- Cover broad range of patterns found in real parallel applications
- Provide paper-and-pencil specification and generic reference implementations
- Keep kernels simple functionally
  - Easy porting to new runtimes/languages
  - Easy to understand by different domain scientists
  - Dominated by single feature, so convenient performance building block
- Parameterize kernels (problem size, iterations, # cores etc.)
- Make sure each kernel does actual work
- Include automatic verification test (analytical solution)
- Ensure enough expoitable concurrency (can be load balanced)
  - Make trivially statically load balanced

RSBench

Linear Algebra Libraries

SYCL-BLAS

Eigen

oneMKL

SYCL Parallel STL

Machine Learning Libraries and Parallel Acceleration Frameworks

SYCL-ML

TensorFlow

SYCL-DNN

STREAM HPC

Argonne NATIONAL LABORATORY

QUALCOMM

arm

SYCL™

intel®

XILINX®

HIPPOCRATICA CIVITAS STUDIUM SALERNI

AMD

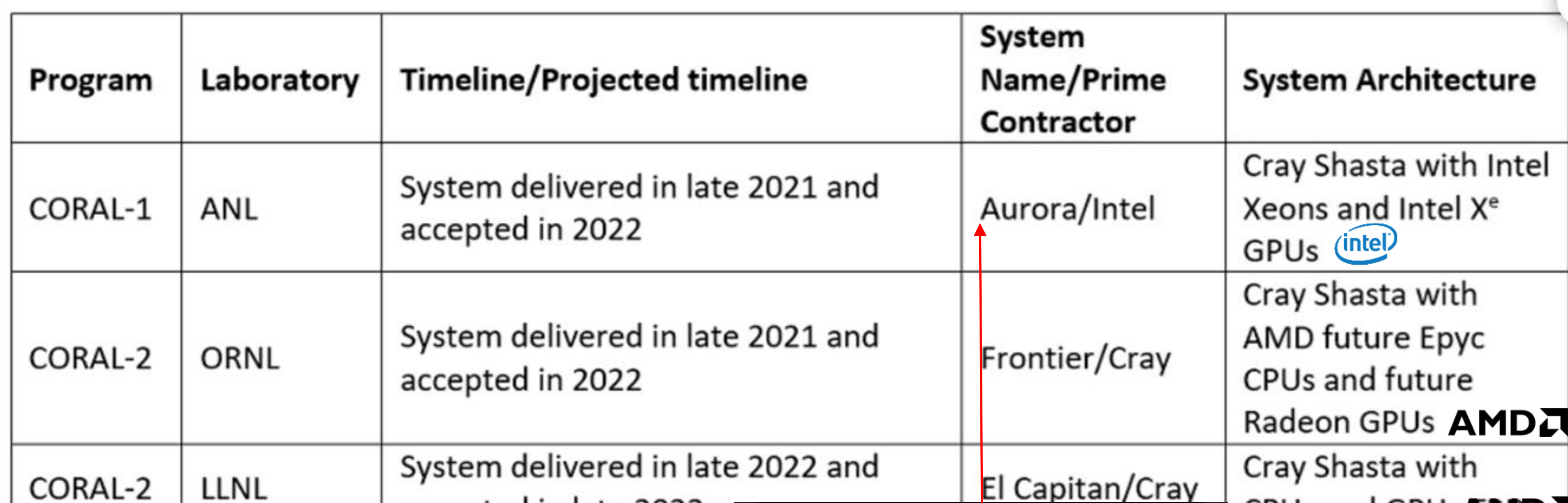
University of BRISTOL

codeplay®

Active Working Group Members

KHRONOS® GROUP

SYCL, Aurora and Exascale computing



Program	Laboratory	Timeline/Projected timeline	System Name/Prime Contractor	System Architecture
CORAL-1	ANL	System delivered in late 2021 and accepted in 2022	Aurora/Intel	Cray Shasta with Intel Xeons and Intel X^e GPUs intel
CORAL-2	ORNL	System delivered in late 2021 and accepted in 2022	Frontier/Cray	Cray Shasta with AMD future Epyc CPUs and future Radeon GPUs AMD
CORAL-2	LLNL	System delivered in late 2022 and accepted in late 2023	El Capitan/Cray	Cray Shasta with CPUs and GPUs AMD

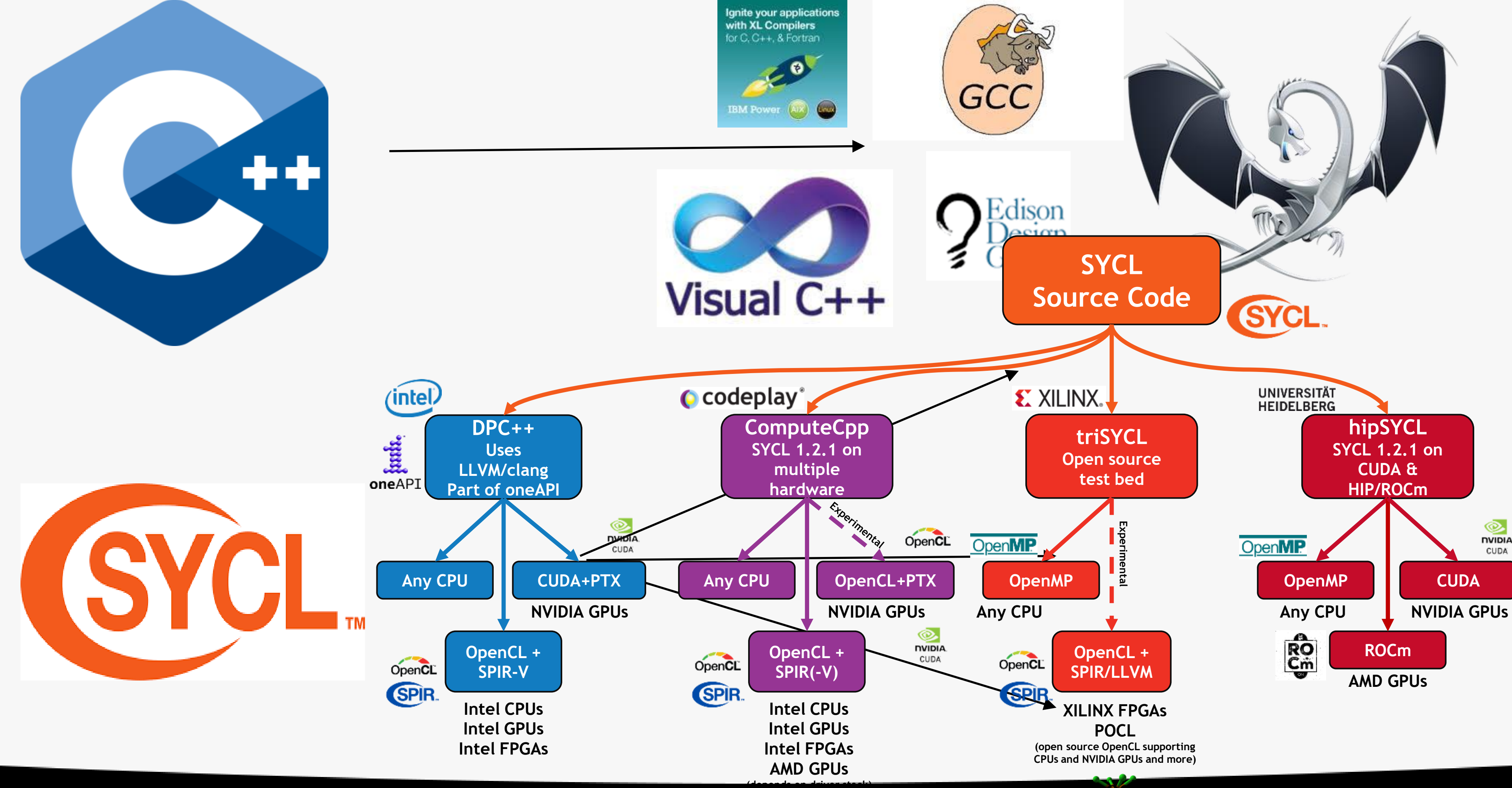
Argonne

A Roadmap for SYCL/DPC++ on Aurora

Thomas Applencourt - [illegible]
Kevin Harms - [illegible]

ALCF

SYCL can run on AMD ROCM

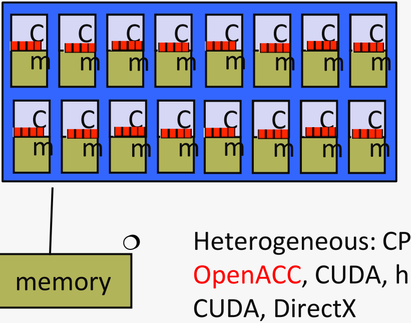

Ignite your applications with XL Compilers for C, C++, & Fortran
IBM Power
AIX
Linux
GCC
Visual C++
Edison Design G
SYCL Source Code
intel
oneAPI
DPC++ Uses LLVM/clang Part of oneAPI
Any CPU
CUDA+PTX
NVIDIA CUDA
NVIDIA GPUs
OpenCL + SPIR-V
OpenCL
SPIR
Intel CPUs
Intel GPUs
Intel FPGAs
codeplay
ComputeCpp SYCL 1.2.1 on multiple hardware
Experimental
Any CPU
OpenCL+PTX
OpenCL
NVIDIA GPUs
OpenCL + SPIR(-V)
Intel CPUs
Intel GPUs
Intel FPGAs
AMD GPUs
(depends on driver stack)
XILINX
triSYCL Open source test bed
OpenMP
Any CPU
Experimental
OpenCL + SPIR/LLVM
XILINX FPGAs
POCL
(open source OpenCL supporting CPUs and NVIDIA GPUs and more)
UNIVERSITÄT HEIDELBERG
hipSYCL SYCL 1.2.1 on CUDA & HIP/ROCm
OpenMP
Any CPU
CUDA
NVIDIA GPUs
ROCm
AMD GPUs
SYCL

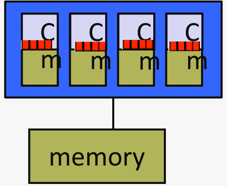
Oh, and one more thing

Which Programming model works on all the Architectures?Is there a pattern?

❒ Multicore Manycore

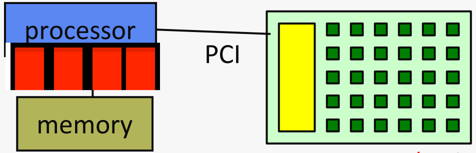
- Manycore vs Multicore CPU: OpenCL, OpenMP, SYCL, C++11/14/17/20, TBB, Cilk, pthread



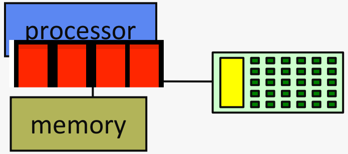


cores can be hardware multithreaded (hyperthread)

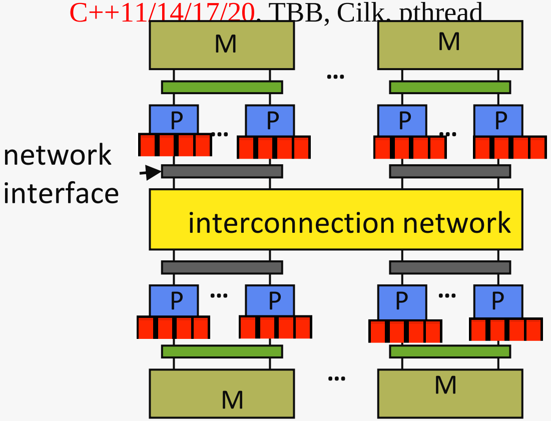
- Heterogeneous: CPU + GPU: OpenCL, OpenMP, SYCL, C++17/20, OpenACC, CUDA, hip, RocM, C++ AMP, Intrinsics, OpenGL, Vulkan, CUDA, DirectX



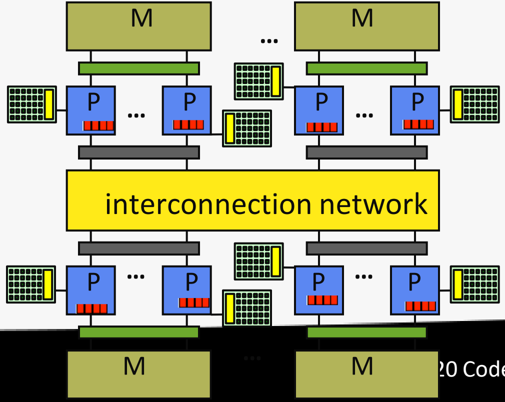
- Heterogeneous: “Fused” CPU + GPU: OpenCL, OpenMP, SYCL, C++17/20, hip, RocM, Intrinsics, OpenGL, Vulkan, DirectX



- Heterogeneous: CPU+Manycore CPU: OpenCL, OpenMP, SYCL, C++11/14/17/20, TBB, Cilk, pthread



- Heterogeneous: Multicore SMP+GPU Cluster: OpenCL, OpenMP, SYCL, C++17/20



To support all the different parallel architectures

- With a single source code base
- And if you also want it to be an International Open Specification
- And if you want it to be growing with the architectures

- You really only have a few choices

OpenCL™

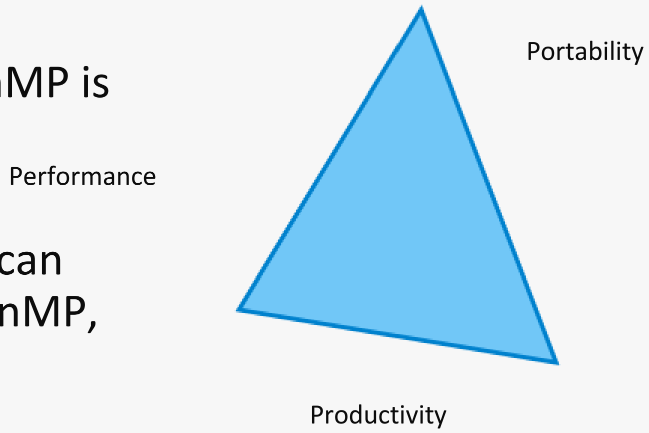
C++

SYCL™

OpenMP®

Summary of Programming models features

- SYCL is entirely standard C++, OpenCL is C99, OpenMP is C, Fortran, C++
- SYCL and OpenCL compiles to SPIR, but SYCL 2020 can also compile to other backends such as Vulkan, OpenMP, Nvidia PTX/CUDA, or some proprietary device ISA.
- SYCL and OpenCL supports a multi compilation model



Summary of Programming models features

- SYCL separates the storage and access of data and has both implicit and explicit data movement; OpenCL, OpenMP, C++ has explicit data movement

- SYCL, OpenMP, C++ are single source; OpenCL is separate source for host and device

- SYCL creates automatic data dependency graphs;

- C++ parallelism is still fairly low level from which all parallel patterns can be built; OpenCL is higher level then C++; SYCL is the highest level; But this means some parallel patterns are not yet available. Higher level means greater productivity.

Summary of Programming models features

- SYCL, OpenCL, C++ are an explicit parallelism model, OpenMP is a directive based programming model

- SYCL and OpenCL are the most ideal for any kind of platforms in an open environment and follows C++ and C closely. C++ allows this separation of concerns and is ideal for general programming purposes. OpenMP is mostly ideal for Fortran and older C code base and does not allow separation of concerns.

Use the Proper Abstraction in the future

Abstraction	How is it supported
Cores	C++11/14/17 threads, async
HW threads	C++11/14/17 threads, async
Vectors	Parallelism TS2-
Atomic, Fences, lockfree, futures, counters, transactions	C++11/14/17 atomics, Concurrency TS1->C++20, Transactional Memory TS1
Parallel Loops	Async, TBB:parallel_invoke, C++17 parallel algorithms, for_each
Heterogeneous offload, fpga	OpenCL, SYCL, HSA, OpenMP/ACC, Kokkos, Raja, CUDA P0796 on affinity
Distributed	HPX, MPI, UPC++ P0796 on affinity
Caches	C++17 false sharing support
Numa	OpenMP/ACC, Executors, Execution Context, Affinity, P0443->Executor TS
TLS	EALS, P0772
Exception handling in concurrent environment	EH reduction properties

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

TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

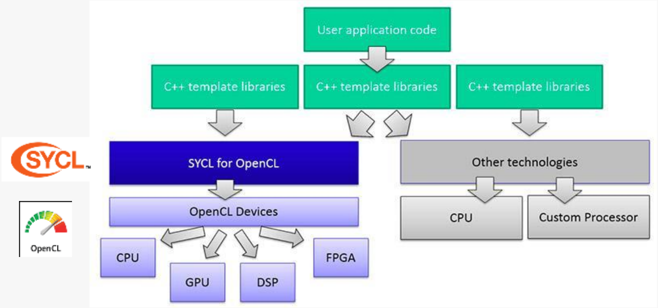
SYCL Ecosystem

- 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, ComputeCpp SDK …
- More information at http://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	**Single-source C++**
Low Level Control	**Programmer Familiarity**
'GPGPU'-style separation of device-side kernel source code and host code	**Approach also taken by C++ AMP and OpenMP**
OpenCL	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

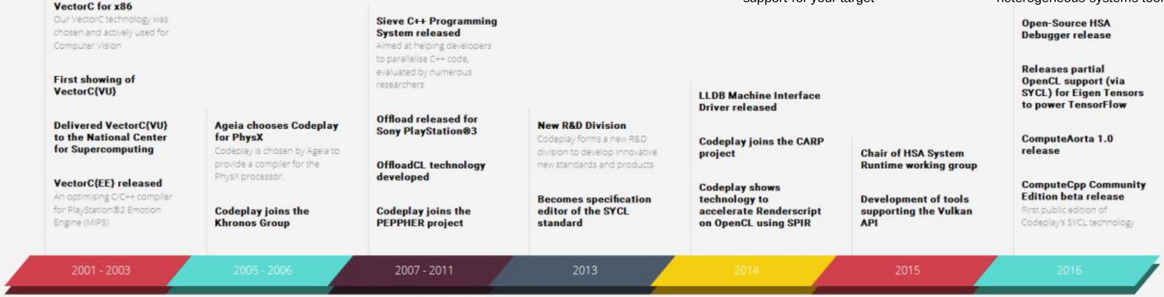
- HSA LLDB Debugger
- SPIR-V tools
- RenderScript debugger in AOSP
- LLDB for Qualcomm Hexagon
- TensorFlow for OpenCL
- C++ 17 Parallel STL for SYCL
- VisionCpp: C++ performance-portable programming model for vision

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



2001 - 2003

VectorC for x86
Our VectorC technology was chosen and actively used for Computer Vision

First showing of VectorC{VU}

Delivered VectorC{VU} to the National Center for Supercomputing

VectorC{EE} released
An optimising C/C++ compiler for PlayStation®2 Emotion Engine (MIPS)

2005 - 2006

Ageia chooses Codeplay for PhysX
Codeplay is chosen by Ageia to provide a compiler for the PhysX processor.

Codeplay joins the Khronos Group

2007 - 2011

Sieve C++ Programming System released
Aimed at helping developers to parallelise C++ code, evaluated by numerous researchers

Offload released for Sony PlayStation®3

OffloadCL technology developed

Codeplay joins the PEPPHER project

2013

New R&D Division
Codeplay forms a new R&D division to develop innovative new standards and products

Becomes specification editor of the SYCL standard

2014

LLDB Machine Interface Driver released

Codeplay joins the CARP project

Codeplay shows technology to accelerate Renderscript on OpenCL using SPIR

2015

Chair of HSA System Runtime working group

Development of tools supporting the Vulkan API

2016

Open-Source HSA Debugger release

Releases partial OpenCL support (via SYCL) for Eigen Tensors to power TensorFlow

ComputeAorta 1.0 release

ComputeCpp Community Edition beta release
First public edition of Codeplay's SYCL technology

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

SYCL™

ComputeCpp™

Community Edition

Available now for free!

Visit:

computecpp.codeplay.com

SYCL™

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

We're Hiring!
codeplay.com/careers/

codeplay®
THE HETEROGENEOUS SYSTEMS EXPERTS

Thanks

@codeplaysoft

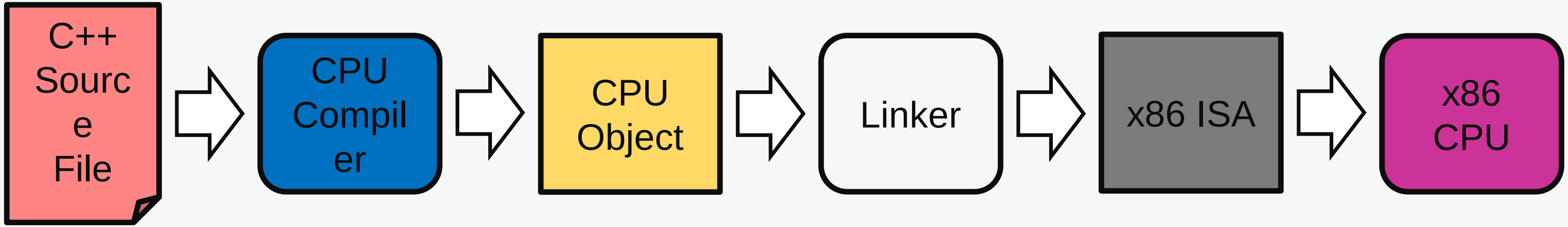
info@codeplay.co
m

www
codeplay.com

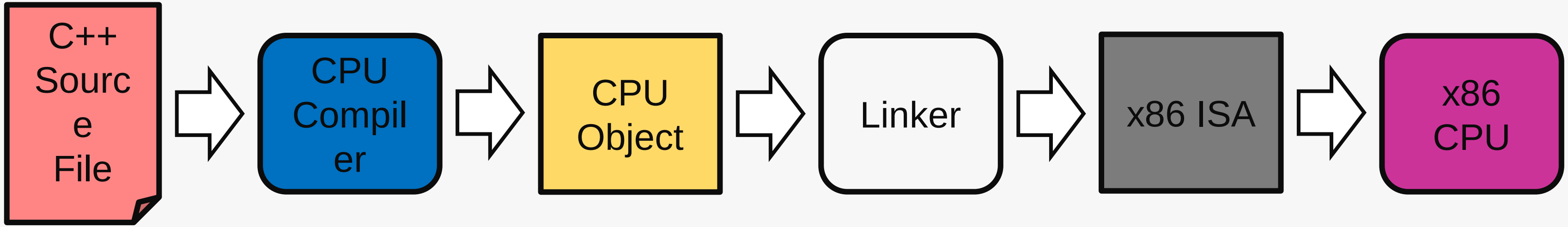
So if you can’t write a single program to run everywhere

- You need a programming model which allows you to compose your problem in different ways

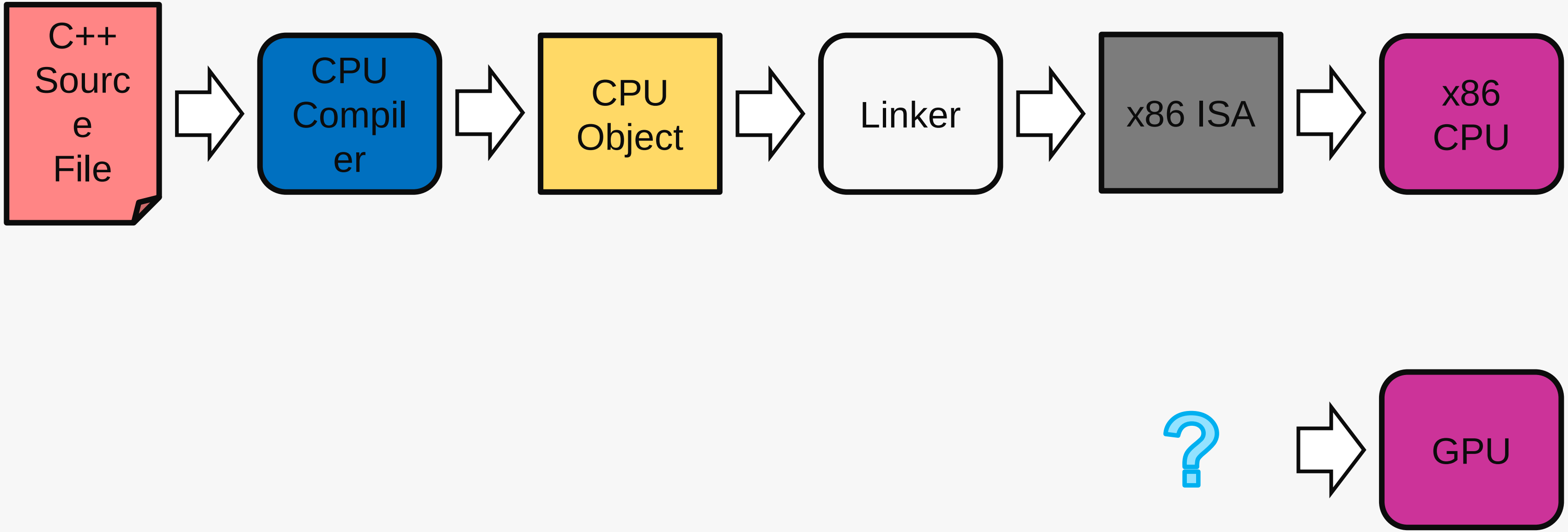
C++ Compilation Model



C++ Compilation Model



C++ Compilation Model



C++ Source File

CPU Compiler

CPU Object

Linker

x86 ISA

x86 CPU

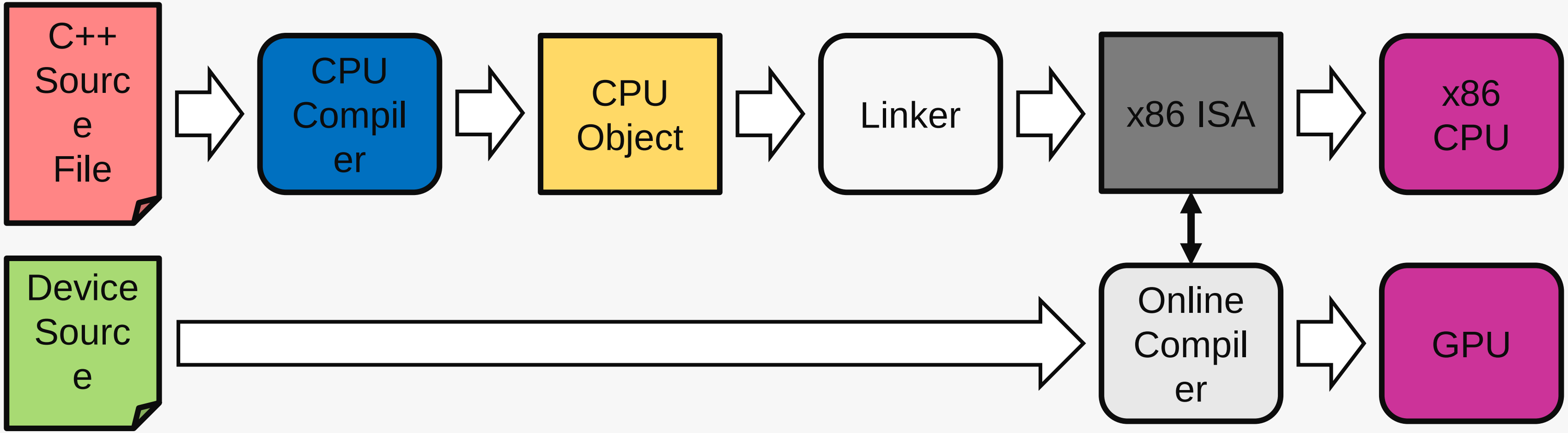
?

GPU

How can we compile source code for a sub architectures?

- ➢ Separate source (OpenCL C, OpenCL C++, GLSL)
- ➢ Single source (SYCL, C++, CUDA, OpenMP, C++ AMP)
- ➢ Embedded DSLs (RapidMind, Halide)

Separate Source Compilation Model

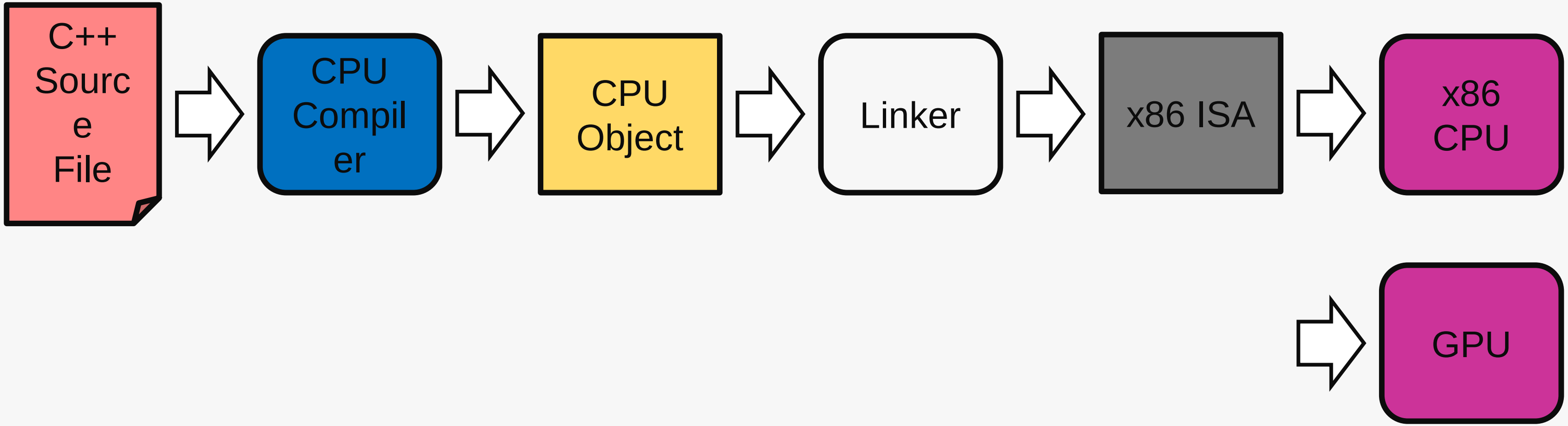


```
float *a, *b, *c;
…
kernel k = clCreateKernel(…, "my_kernel", …);
clEnqueueWriteBuffer(…, size, a, …);
clEnqueueWriteBuffer(…, size, a, …);
clEnqueueNDRange(…, k, 1, {size, 1, 1}, …);
clEnqueueWriteBuffer(…, size, c, …);
```

Here we're using OpenCL as an example

```
void my_kernel(__global float *a, __global float *b,
               __global float *c) {
  int id = get_global_id(0);
  c[id] = a[id] + b[id];
}
```

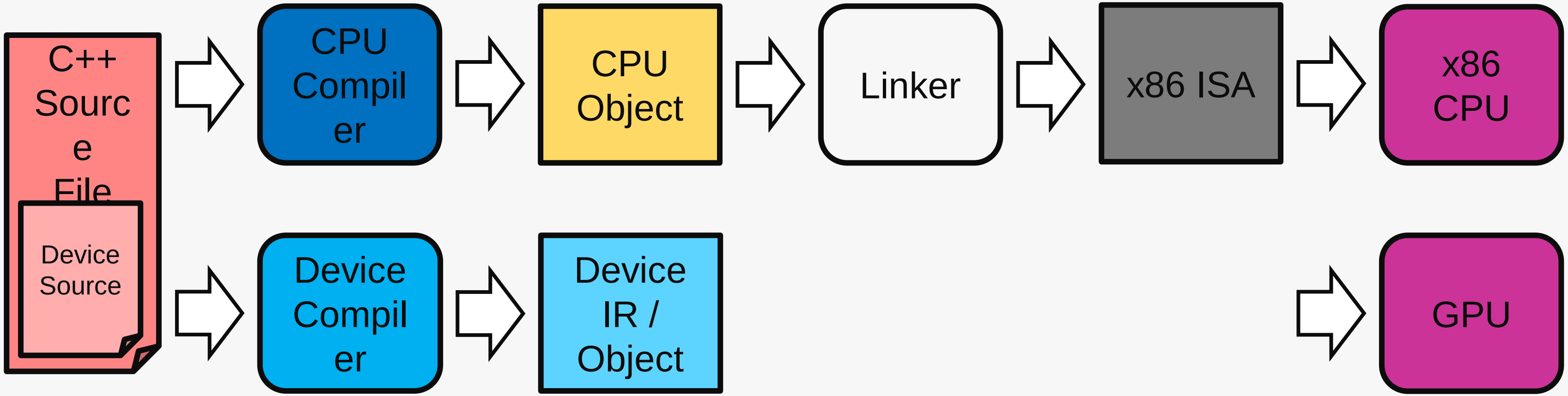
Single Source Compilation Model



```
array_view<float> a, b, c;
extent<2> e(64, 64);

parallel_for_each(e, [=](index<2> idx) restrict(amp) {
  c[idx] = a[idx] + b[idx];
});
```

Here we are using C++ AMP as an example

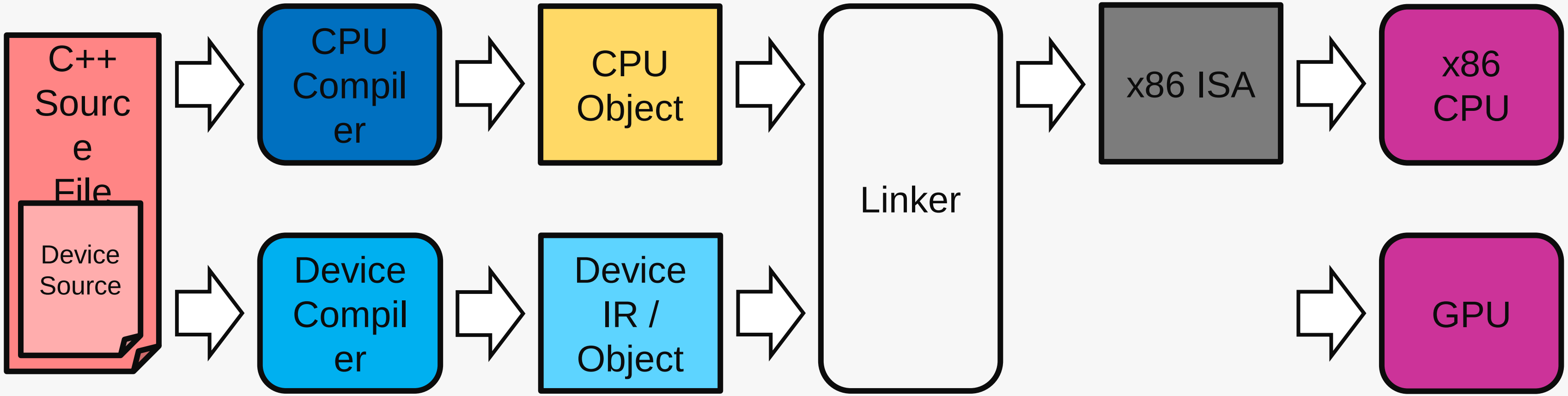
Single Source Compilation Model



```
array_view<float> a, b, c;
extent<2> e(64, 64);

parallel_for_each(e, [=](index<2> idx) restrict(amp) {
  c[idx] = a[idx] + b[idx];
});
```

Here we are using C++ AMP as an example

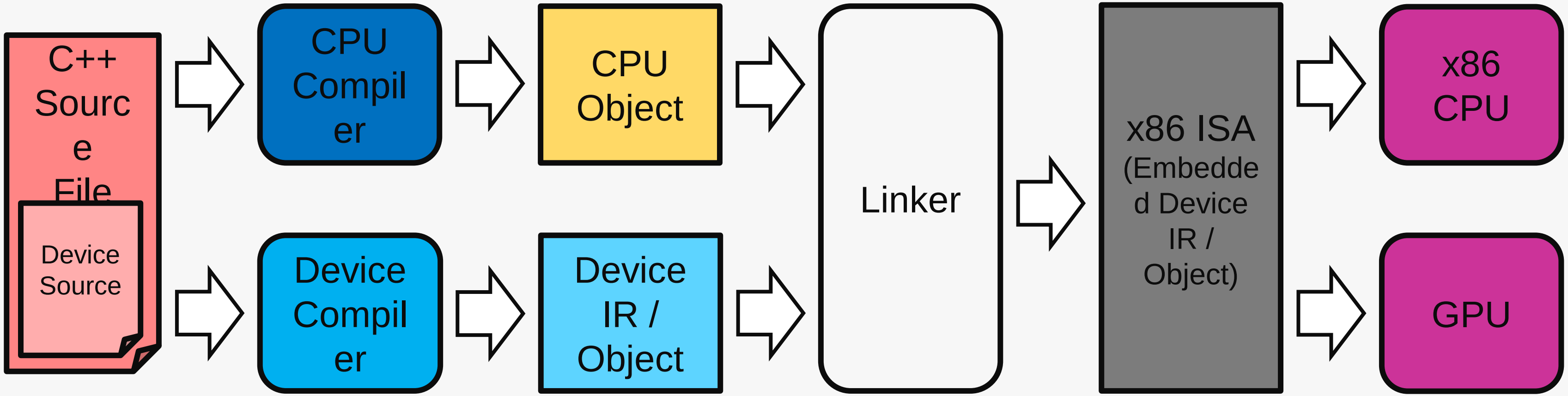
Single Source Compilation Model



```
array_view<float> a, b, c;
extent<2> e(64, 64);

parallel_for_each(e, [=](index<2> idx) restrict(amp) {
  c[idx] = a[idx] + b[idx];
});
```

Here we are using C++ AMP as an example

Single Source Compilation Model



```
array_view<float> a, b, c;
extent<2> e(64, 64);

parallel_for_each(e, [=](index<2> idx) restrict(amp) {
  c[idx] = a[idx] + b[idx];
});
```

Here we are using C++ AMP as an example

Benefits of Single Source

- Device code is written in C++ in the same source file as the host CPU code

- Allows compile-time evaluation of device code

- **Supports type safety across host CPU and device**

- **Supports generic programming**

- Removes the need to distribute source code

SYCL aims to easily integrate with existing C++ libraries

- SYCL is completely standard C++ with no language extensions
- SYCL provides a limited subset of C++ features

Standard C++

```
~~__global__~~ vec_add(float *a, float *b, float *c)
{
  return c[i] = a[i] + b[i];
}

float ~~*a,~~
vec_add<<
```

```
vector<float> a, b, c;

~~#pragma parallel_for~~
                .size(); i++)

                ;
```

```
array_view<float> a, b, c;

parallel_for_each(extent, [=](index<2> idx) restrict(amp)
{
  c[idx] = a[idx] + b[idx];
});
```

```
cgh.parallel_for<class vec_add>(range, [=](cl::sycl::id<2> idx) {
  c[idx] = a[idx] + c[idx];
}));
```

C++ Features

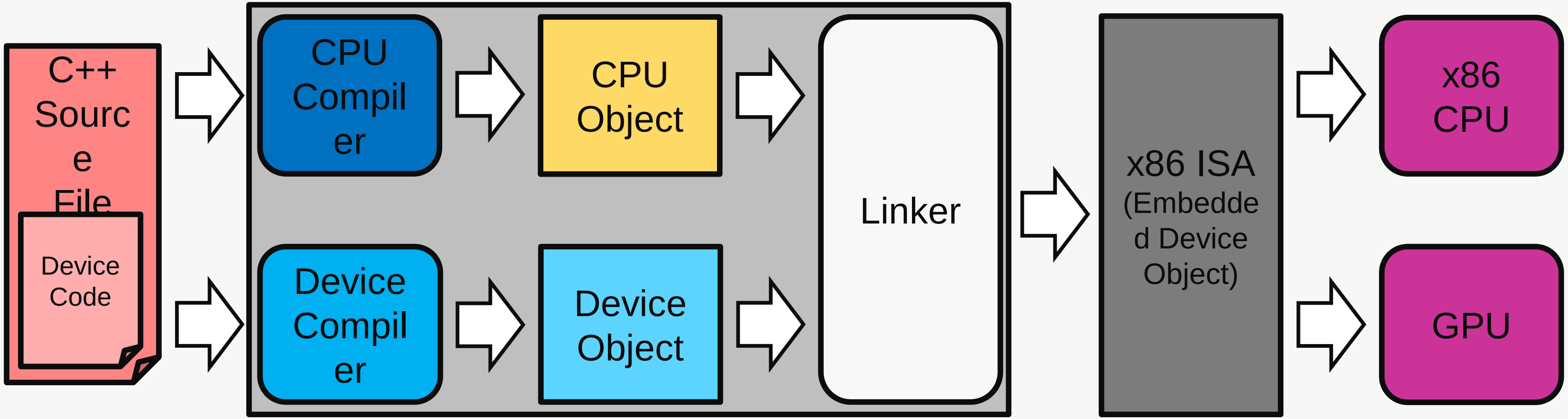
- Supported:
  - Classes
  - Operator overloading
  - Lambdas
  - Static polymorphism
  - Placement allocation
  - Template recursion
- Unsupported:
  - Recursion
  - Exception handling
  - RTTI
  - Dynamic allocation
  - Dynamic polymorphism
  - Function pointers
  - Virtual functions
  - Static variables

Some features cannot be supported on device due to hardware restrictions

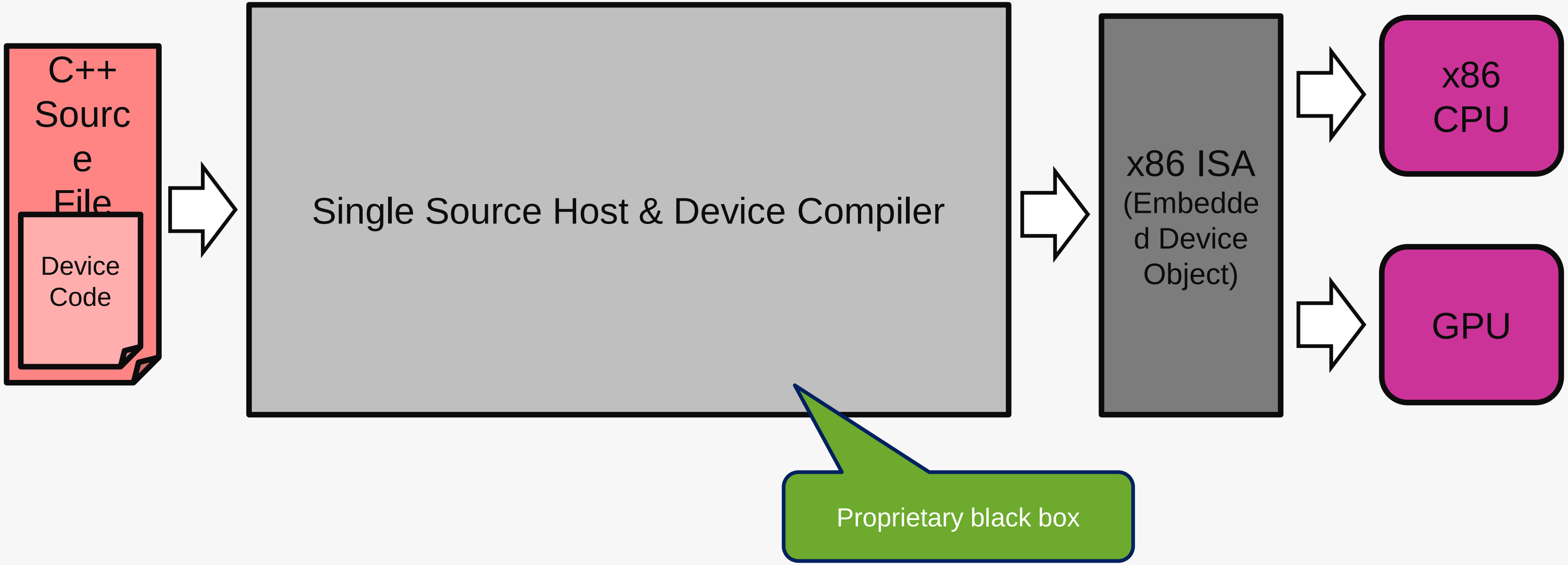
SYCL aims to be open, portable and flexible

- SYCL offers a single source programming model with multi pass compilation

Single Pass Compilation



Single Pass Compilation



C++ Source File

Device Code

Single Source Host & Device Compiler

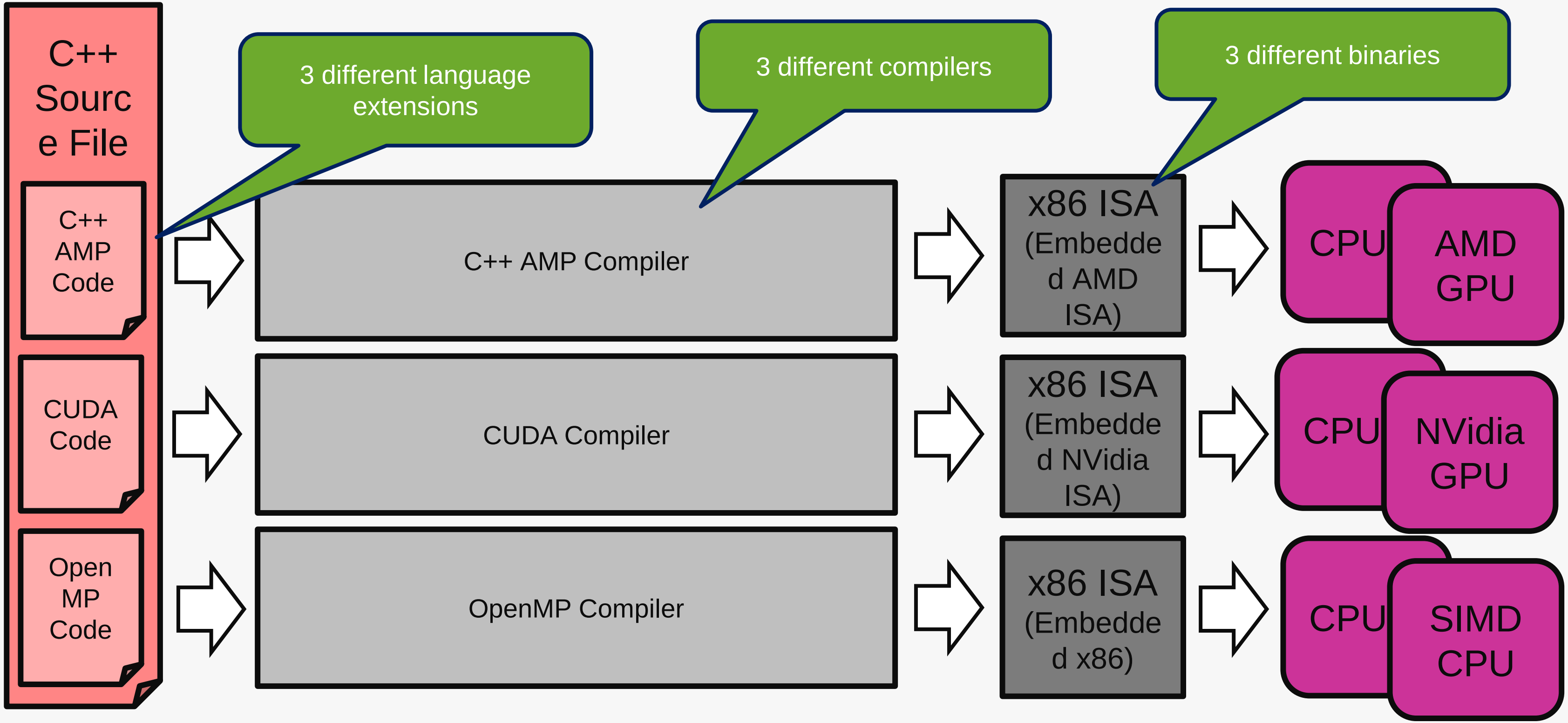
x86 ISA (Embedded Device Object)

x86 CPU

GPU

Proprietary black box

Single Pass Compilation



C++ Source File

C++ AMP Code

CUDA Code

OpenMP Code

3 different language extensions

3 different compilers

3 different binaries

C++ AMP Compiler

CUDA Compiler

OpenMP Compiler

x86 ISA (Embedded AMD ISA)

x86 ISA (Embedded NVidia ISA)

x86 ISA (Embedded x86)

CPU

AMD GPU

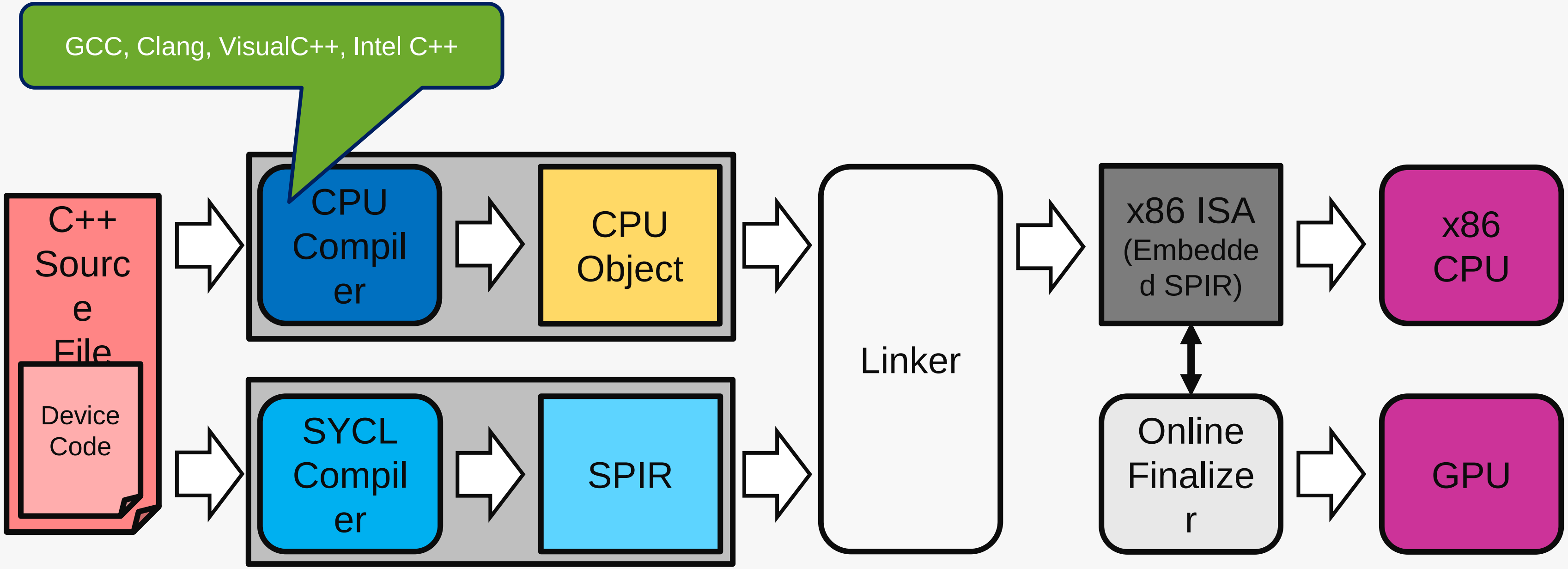
CPU

NVidia GPU

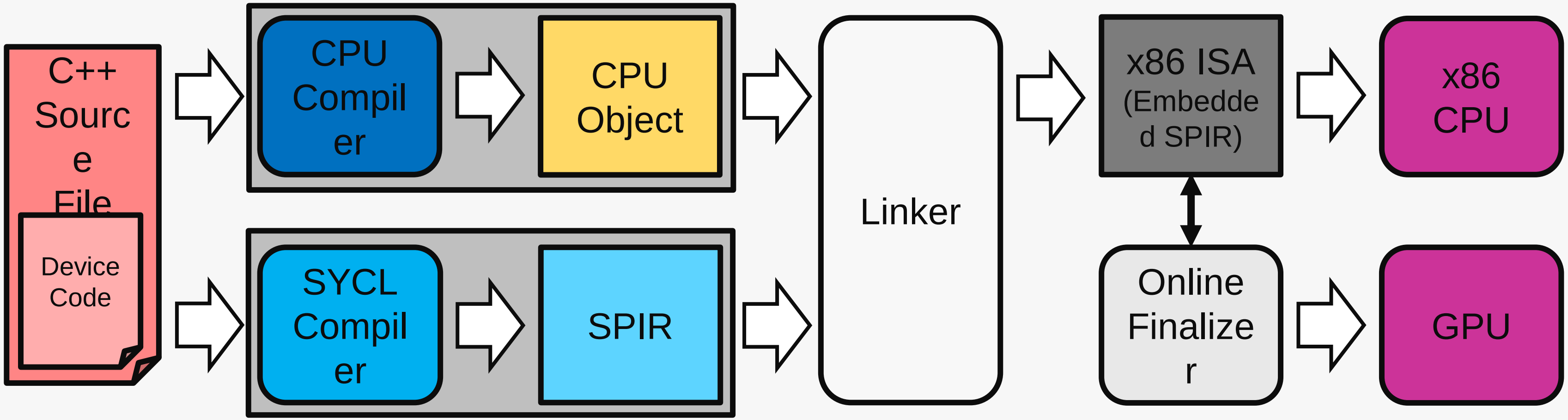
CPU

SIMD CPU

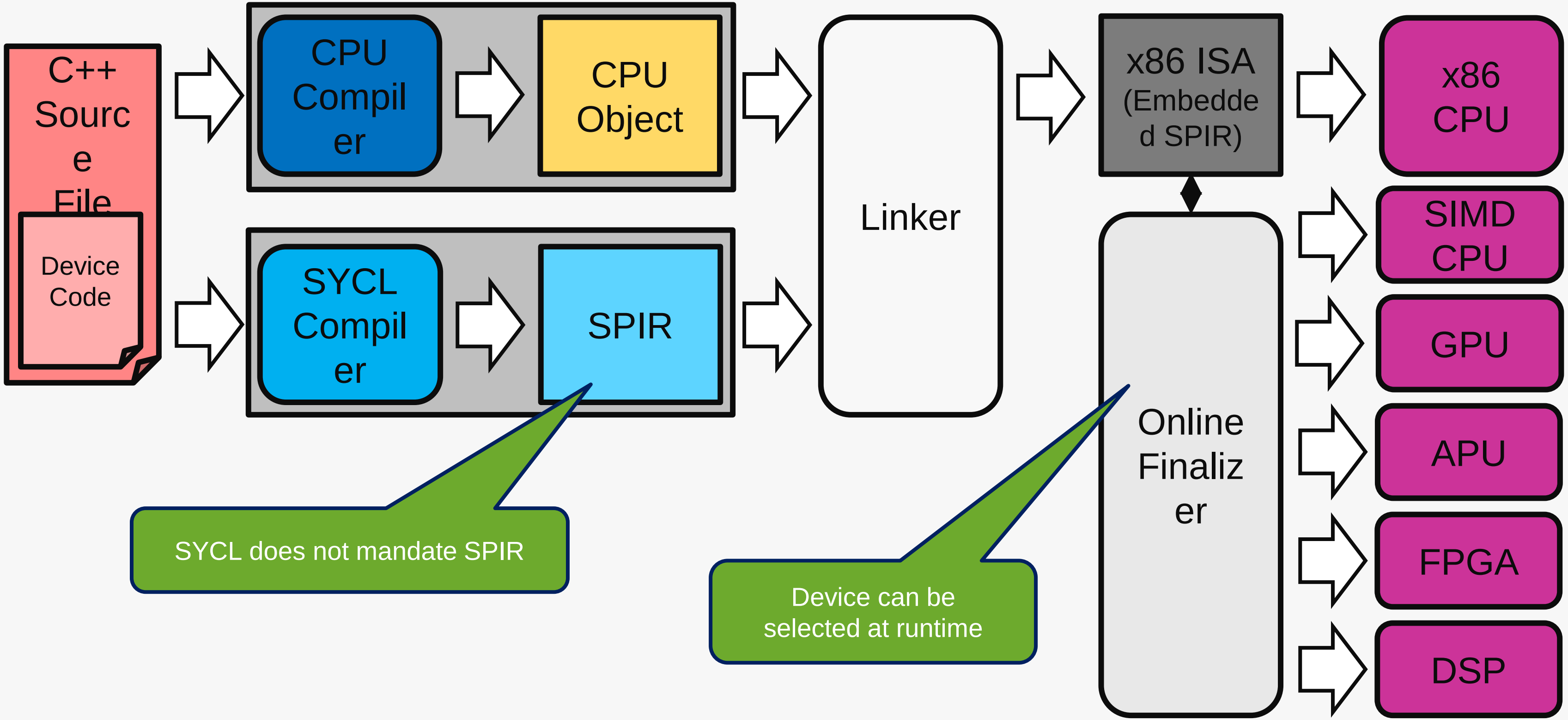
Multi Pass Compilation



Multi Pass Compilation



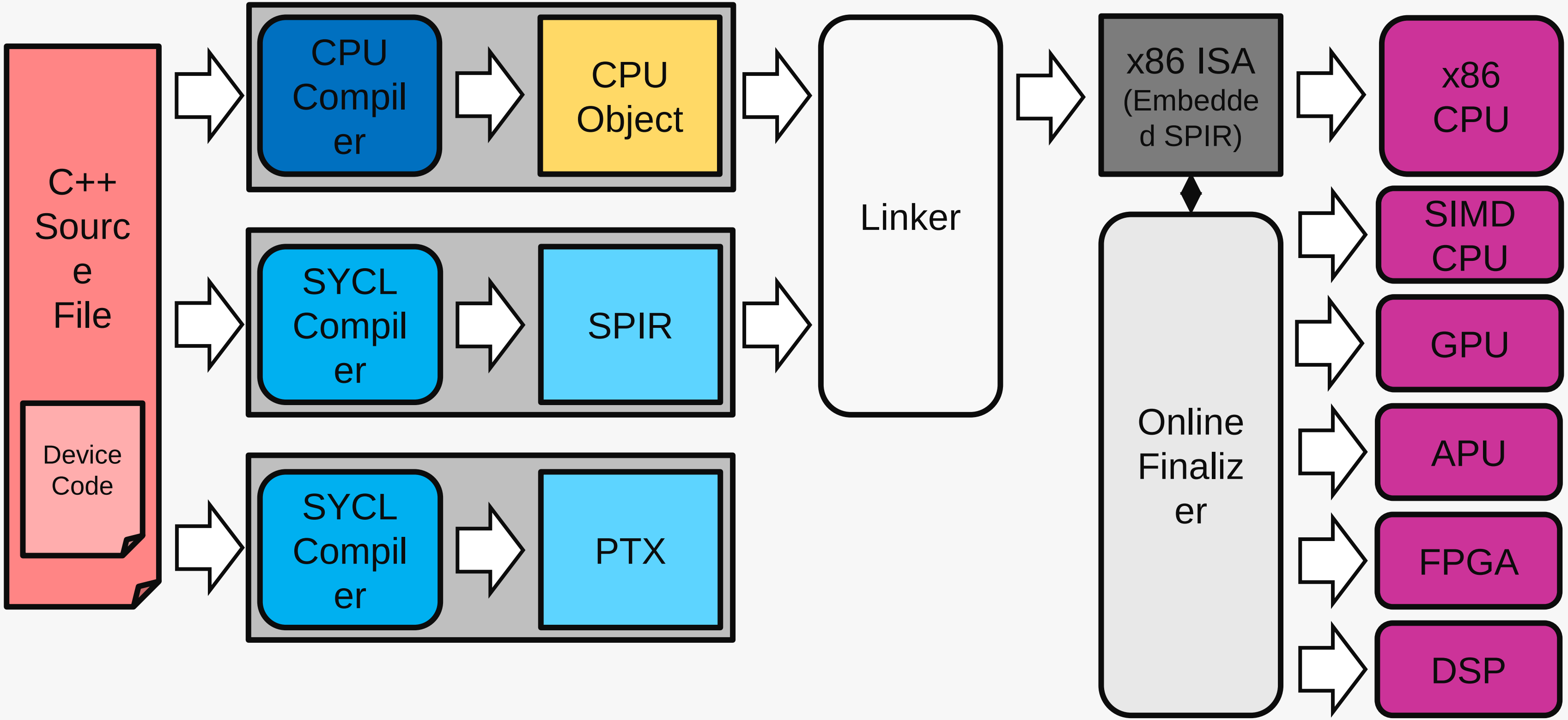
Multi Pass Compilation



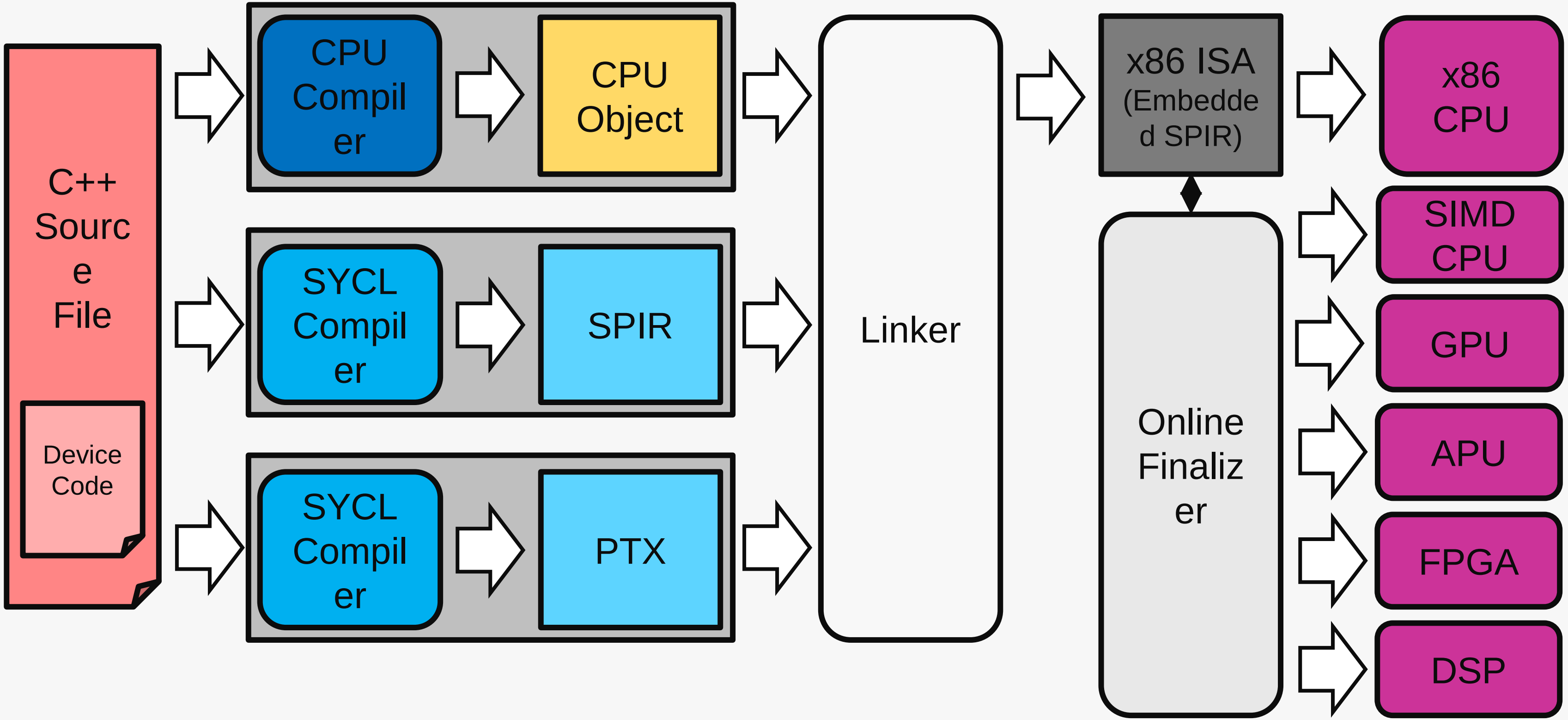
SYCL does not mandate SPIR

Device can be selected at runtime

Multi Pass Compilation



Multi Pass Compilation


C++ Source File
Device Code
CPU Compiler
CPU Object
SYCL Compiler
SPIR
SYCL Compiler
PTX
Linker
x86 ISA (Embedded SPIR)
Online Finalizer
x86 CPU
SIMD CPU
GPU
APU
FPGA
DSP