The future direction of SYCL and ISO C++ heterogeneous programming

Michael Wong

Codeplay Software

VP of Research and Development

http::://wongmichael.com/about

michael@codeplay.com

GPU DAY 2019

VP of R&D of Codeplay

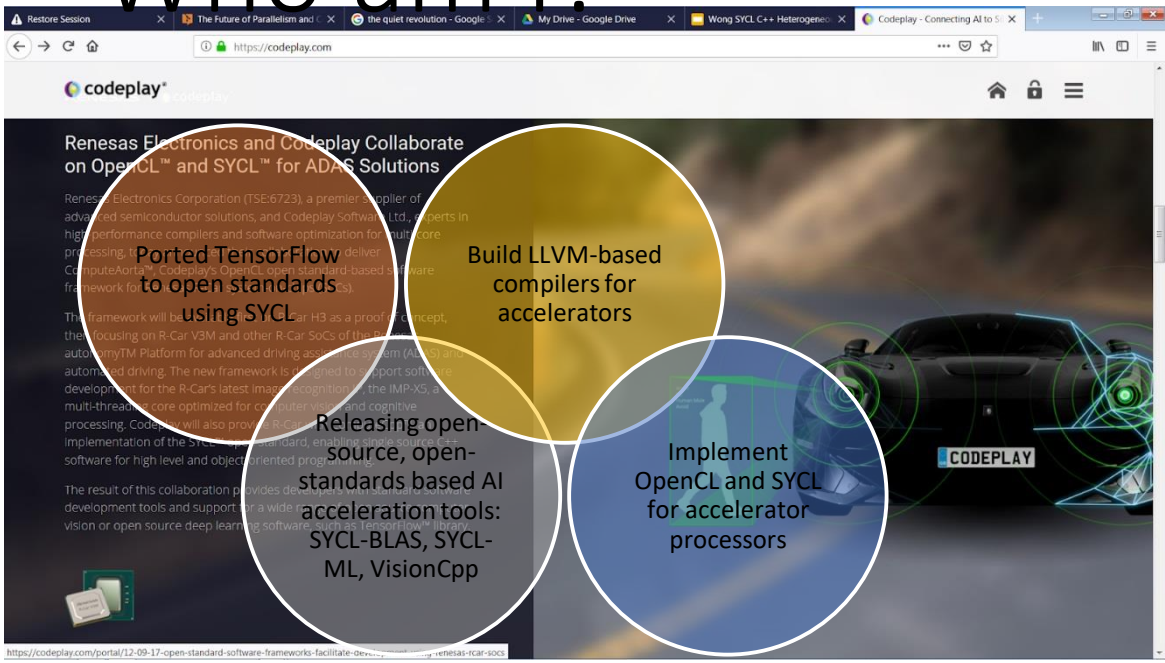
Chair of SYCL Heterogeneous Programming Language
C++ Directions Group
ISOCPP.org Director, VP
http://isocpp.org/wiki/faq/wg21#michael-wong

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

We build GPU compilers for semiconductor companies

- **Now working to make AI/Ml heteroegneous acceleration safe for autonomous vehicle**

Who am I?



Acknowledgement Disclaimer

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

Specifically, Paul Mckenney, Joe Hummel, Bjarne Stroustru, Botond Ballo for some of the slides.

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

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

Legal Disclaimer

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

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

Codeplay - Connecting AI to Silicon

Products

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™, HSA™ and Vulkan™

Addressable Markets

Automotive (ISO 26262)
IoT, Smartphones & Tablets
High Performance Compute (HPC)
Medical & Industrial

Technologies: Vision Processing
Machine Learning
Artificial Intelligence
Big Data Compute

Company

High-performance software solutions for custom heterogeneous systems

Enabling the toughest processor systems with tools and middleware based on open standards

Established 2002 in Scotland

~70 employees

Customers

BROADCOM
RENESAS
Imagination
QUALCOMM
Movidius
intel Partners AMD

3 Act Play

- Is ISO C++ going heterogeneous?
- Is SYCL gaining in the marketplace?
- Is there a direction for C++ and SYCL?

- What gets me up every morning?

C++11,14,17“No more Raw Food”

Don’t use	Don’t use raw numbers, do type-rich programming with UDL
Don’t declare	Don’t declare, use auto whenever possible
Don’t use	Don’t use raw NULL or (void *) 0, use nullptr
Don’t use	Don’t use raw new and delete, use unique_ptr/shared_ptr
Don’t use	Don’t use heap-allocated arrays, use std::vector and std::string, or the new VLA, then dynarray<>
Don’t use	Don’t use functors, use lambdas
Don’t use	Don’t use raw loops; use STL algorithms, ranged-based for loops, and lambdas
Rule	Rule of Three? Rule of Zero or Rule of Five.

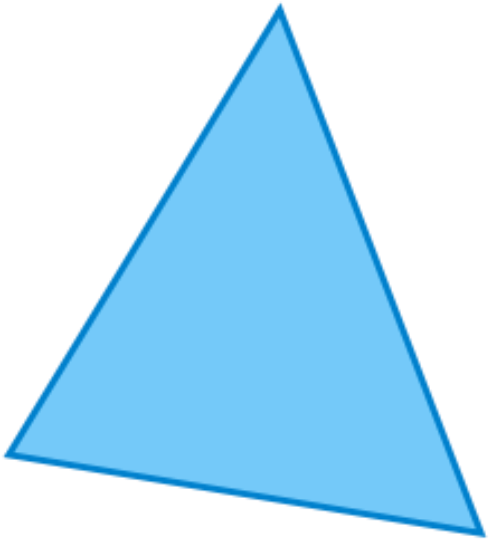
Parallelism “Use the right abstraction”

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, 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
Distributed	HPX, MPI, UPC++
Caches	C++17 false sharing support
Numa	Executors, Execution Context, Affinity
TLS	EALS
Exception handling in concurrent environment	EH reduction properties

Act 1

- Is ISO C++ going heterogeneous?

Iron Triangle of Parallel Programming Language Nirvana



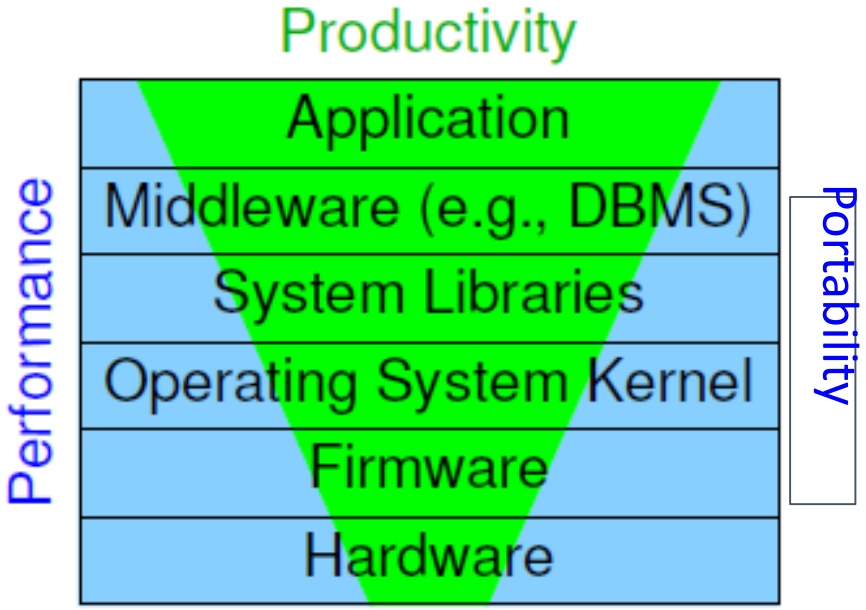
Performance Portability Productivity

OpenCL

OpenMP

CUDA

SYCL



Iron Triangle of Parallel Programming Nirvana is about making engineering tradeoffs

Concurrency vs Parallelism

What makes parallel or concurrent programming harder than serial programming? What's the difference? How much of this is simply a new mindset one has to adopt?

Parallel/concurrency before C++11 (C++98)

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous (GPUs, accelerators, FPGA, embedded AI processors)
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
examples	GUI,background printing, disk/net access	trees, quicksorts, compilation	locked data(99%), lock-free libraries (wizards), atomics (experts)	Pipelines, reactive programming, offload,, target, dispatch
key metrics	responsiveness	throughput, many core scalability	race free, lock free	Independent forward progress,, load-shared
requirement	isolation, messages	low overhead	composability	Distributed, heterogeneous
today's abstractions	POSIX threads, win32 threads, OpenCL, vendor intrinsic	openmp, TBB, PPL, OpenCL, vendor intrinsic	locks, lock hierarchies, vendor atomic instructions, vendor intrinsic	OpenCL, CUDA

Parallel/concurrency after C++11

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous (GPUs, accelerators, FPGA, embedded AI processors)
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
examples	GUI,background printing, disk/net access	trees, quicksorts, compilation	locked data(99%), lock-free libraries (wizards), atomics (experts)	Pipelines, reactive programming, offload,, target, dispatch
key metrics	responsiveness	throughput, many core scalability	race free, lock free	Independent forward progress,, load-shared
requirement	isolation, messages	low overhead	composability	Distributed, heterogeneous
today's abstractions	C++11: thread,lambda function, TLS	C++11: Async, packaged tasks, promises, futures, atomics	C++11: locks, memory model, mutex, condition variable, atomics, static init/term	C++11: lambda

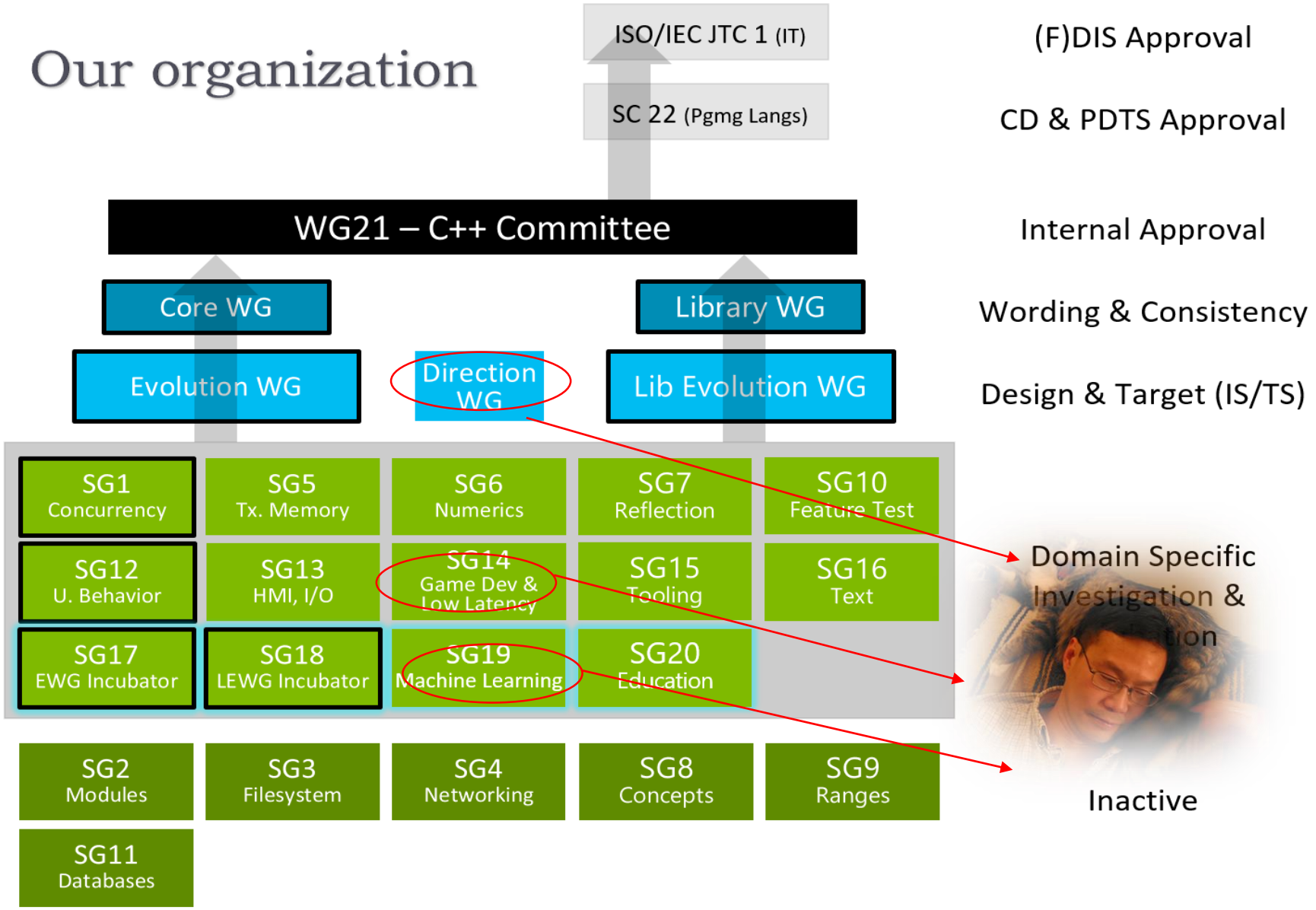
Parallel/concurrency after C++14

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
examples	GUI,background printing, disk/net access	trees, quicksorts, compilation	locked data(99%), lock-free libraries (wizards), atomics (experts)	Pipelines, reactive programming, offload,, target, dispatch
key metrics	responsiveness	throughput, many core scalability	race free, lock free	Independent forward progress,, load-shared
requirement	isolation, messages	low overhead	composability	Distributed, heterogeneous
today's abstractions	C++11: thread,lambda function, TLS, async C++14: generic lambda	C++11: Async, packaged tasks, promises, futures, atomics,	C++11: locks, memory model, mutex, condition variable, atomics, static init/term, C++ 14: shared_lock/shared_timed_mutex, OOTA, atomic_signal_fence,	C++11: lambda C++14: none

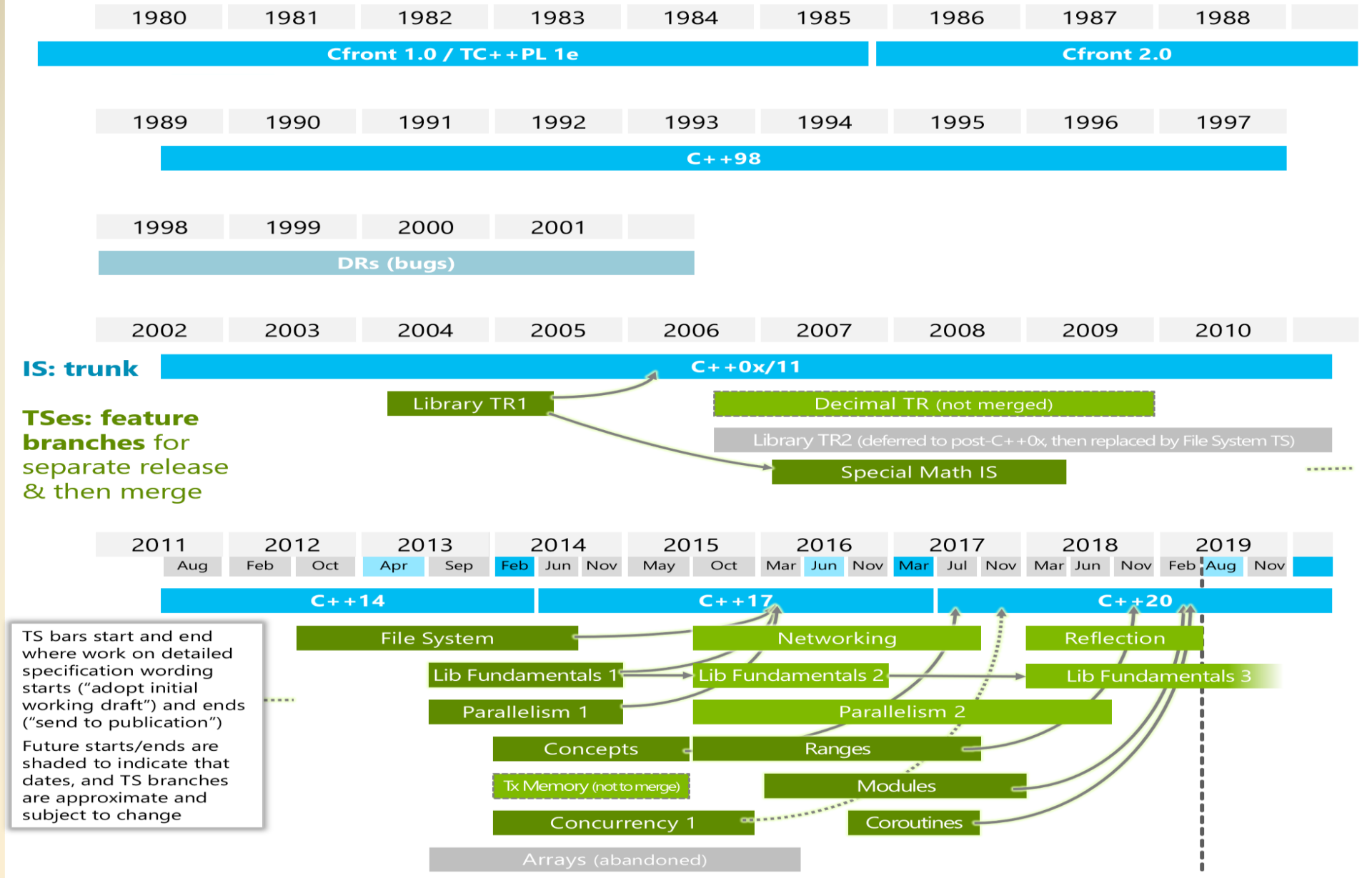
Parallel/concurrency after C++17

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous (GPUs, accelerators, FPGA, embedded AI processors)
summary	tasks that run independently and communicate via messages	operations on groups of things, exploit parallelism in data and algorithm structures	avoid races and synchronizing objects in shared memory	Dispatch/offload to other nodes (including distributed)
today's abstractions	C++11: thread,lambda function, TLS, async C++14: generic lambda	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParallelSTL, control false sharing	C++11: locks, memory model, mutex, condition variable, atomics, static init/term, C++ 14: shared_lock/shared_timed_mutex, OOTA, atomic_signal_fence, C++ 17: scoped _lock, shared_mutex, ordering of memory models, progress guarantees, TOE, execution policies	C++11: lambda C++14: none C++17: progress guarantees, TOE, execution policies

ISO C++ Standard



ISO C++ Timeline https://isocpp.org/std/status



Pre-C++11 projects

ISO number	Name	Status	What is it?	C++17?
ISO/IEC TR 18015:2006	Technical Report on C++ Performance	Published 2006 (ISO store) Draft: TR18015 (2006-02-15)	C++ Performance report	No
ISO/IEC TR 19768:2007	Technical Report on C++ Library Extensions	Published 2007-11-15 (ISO store) Draft: n1745 (2005-01-17) TR 29124 split off, the rest merged into C++11	Has 14 Boost libraries, 13 of which was added to C++11.	N/A (mostly already included into C++11)
ISO/IEC TR 29124:2010	Extensions to the C++ Library to support mathematical special functions	Published 2010-09-03 (ISO Store) Final draft: n3060 (2010-03-06). Under consideration to merge into C++17 by p0226 (2016-02-10)	Really, ORDINARY math today with a Boost and Dinkumware Implementation	YES
ISO/IEC TR 24733:2011	Extensions for the programming language C++ to support decimal floating-point arithmetic	Published 2011-10-25 (ISO Store) Draft: n2849 (2009-03-06) May be superseded by a future Decimal TS or merged into C++ by n3871	Decimal Floating Point decimal32 decimal64 decimal128	No. Ongoing work in SG6

Status after Feb Kona C++ Meeting

ISO NUMBER	NAME	STATUS	LINKS	C++20?
ISO/IEC TS 19841:2015	Transactional Memory TS	Published 2015-09-16, (ISO Store). Final draft: n4514 (2015-05-08)	Composable lock-free programming that scales	No. Already in GCC 6 release and waiting for subsequent usage experience.
ISO/IEC TS 19217:2015	C++ Extensions for Concepts	Published 2015-11-13. (ISO Store). Final draft: n4553 (2015-10-02) Current draft: p0734r0 (2017-07-14) Merged into C++20 (with modifications).	Constrained templates	Merged into C++20, **including abbreviated function templates!**
	Executors		Abstraction for where/how code runs in a concurrent context	Not headed for C++ 20, now retarget for C++23
	Coroutines TS		Resumable functions, based on Microsoft's await design	Published! Merged into C++20
	Reflection TS		Static code reflection mechanisms	PDTS ballot done. Approved for publication.

Concepts: compromised design for Abbreviated Function Template

```
void f(Concept auto x);
Concept auto f(Concept auto x);
```

Status after Feb Kona C++ Meeting

ISO number	Name	Status	What is it?	C++20?
ISO/IEC TS 19571:2016	C++ Extensions for Concurrency	Published 2016-01-19. (ISO Store) Final draft: p0159r0 (2015-10-22)	improvements to future, latches and barriers, atomic smart pointers	Latches, atomic<shared_ptr<t>> merged into C++20. Already in Visual Studio release and Anthony Williams Just Threads! and waiting for subsequent usage experience. Will be withdrawn
ISO/IEC TS 19568:2017	C++ Extensions for Library Fundamentals, Version 2	Published 2017-03-30. (ISO Store) Draft: n4617 (2016-11-28)	source code information capture and various utilities	Published! Parts of it merged into C++17, rest moved to V3
ISO/IEC DTS 21425:2017	Ranges TS	Published 2017-12-05. (ISO Store) Draft: n4685 (2017-07-31)	Range-based algorithms and views	Merged in C++20
ISO/IEC TS 19216:2018	Networking TS	Published 2018-04-24. (ISO Store) Draft n4734 (2017-04-04). Latest draft: n4771 (2018-10-08)	Sockets library based on Boost.ASIO	Published. Not headed to C++20.
ISO/IEC TS 21544:2018	Modules V1	Published 2018-05-16. (ISO Store) Final Draft n4720 (2018-01-29)	A component system to supersede the textual header file inclusion model	Published as a TS
	Modules V2		Improvements to Modules v1, including a better transition path	Merged into C++20
	Contracts		Pre and post conditions	Merged into C++20

Status after Feb Kona C++ Meeting

ISO number	Name	Status	What is it?	C++20?
ISO/IEC DTS 19568:xxxx	Numerics TS	Early development. Draft p0101 (2015-09-27)	Various numerical facilities	Under active development
ISO/IEC DTS 19571:xxxx	Concurrency TS 2	Early development	Exploring , lock-free, hazard pointers, RCU, atomic views, concurrent data structures, fibers Deprecate volatile, add volatile_load/store, TLS?	Under active development. Possible new clause
ISO/IEC TS 19570:2018	Parallelism TS 2	Published 2018-11-15. (ISO Store). Draft: n4773 (2018-10-08)	task blocks, progress guarantees, SIMD<T>, vec, no_vec loop based execution policy	Published. Most are Headed into C++20
ISO/IEC DTS 19841:xxxx	Transactional Memory TS 2	Early development	Exploring on_commit, in_transaction. Lambda-based executor model.	Under active development.
ISO/IEC DTS 19568:xxxx	Graphics TS	Early development. Draft p0267r8 (2018-06-26)	2D drawing API using Cairo interface, adding stateless interfacec	Restarted after being shutdown.
ISO/IEC DTS 19568:xxxx	Library Fundamental V3	Initial draft, early development	Generic scope guard and RAII wrappers	Under development

Status after Feb Kona C++ Meeting

ISO number	Name	Status	What is it?	C++20?
	Linear Algebra	SG14 SIG WIP	Blas, separated into 3 layers	Under active development. Aiming for C++23
	Machine Learning	SG19 WIP	Improve C++ for ML,AI, DNN, Graph programming	Under active development. Aiming for C++23
	Pattern Matching	A match-like facility for C++ WIP		Under active development. Aiming for C++23
	Undefined Behaviour/Safety Critical	SG12 WIP	optimization that cause UB. Pointer provenance, signed integer overflow Validate external C++ Safety APIs: Misra, Autosar	Under active development. Aiming for C++23
	Education	SG20 WIP	Support educating C++, especially new features	Under active development. Aiming for C++23
	Audio	SG13 HMI WIP	Audio drivers	Under active development. Aiming for C++23
	Unicode	SG16 WIP	Compile-time regular expression, source code info capture, charset transcoding	Under active development. Aiming for C++23
	Tooling Ecosystem	SG15 WIP	Build systems;	Under active development. Aiming for C++23 TR

C++ 20 Language Features

- Most notably, the **Concepts Technical Specification has been merged into C++20**!
- Template parameter lists for generic lambdas. T
- Designated initializers.
- Lambda capture [=, *this]
- A __VA_OPT__ macro to make variadic macros easier to use.
- Default member initializers for bitfields
- A tweak to C++17's constructor template argument deduction rules
- Fixing const-qualified pointers to members

- The most significant new feature voted in was **operator<=>**,
- Range-based for statements with initializer.
- Lambdas is unevaluated contexts.
- Default constructible and assignable stateless lambdas.
- Simplifying implicit lambda capture.
- Fixing small functionality gaps in constraints.
- Deprecating the notion of "plain old data" (POD).
- Access checking on specializations.
- const mismatch with defaulted copy constructor.
- ADL and function templates that are not visible.
- Core issue 1581: when are constexpr member functions defined?

More C++20 Language Features

- Language support for empty objects
- Relaxing the structured bindings customization point finding rules.
- Structured bindings in accessible members.
- Allow pack expansion in lambda *init-capture*.
- Symmetry for <=>
- Likely and unlikely attributes
- Down with typename!
- Relaxing range-based for loop's customization point finding rules

- Support for contract-based programming in C++20
- Class types in non-type template parameters.
- Allowing virtual function calls in constant expressions.
- Prohibit aggregates with user-declared constructors.
- Efficient sized deletion for variable-sized classes.

More C++ 20 Language Features

- Consistency improvements for <=> and other comparison operators.
- Conditionally explicit constructors, a.k.a. explicit(bool).
- · Deprecate implicit capture of this via [=].
- · Integrating feature-test macros into the C++ working draft.
- · A tweak to the rules about when certain errors related to a class being abstract are reported.
- · A tweak to the treatment of padding bits during atomic compare-and-exchange operations.
- · Tweaks to the __VA_OPT__ preprocessor feature.
- · Updating the reference to the Unicode standard.

- **Abbreviated function templates** (AFTs).
- Improvements to *return-type-requirements*.
- Immediate functions.
- std::is_constant_evaluated()
- try / catch blocks in constexpr functions.
- Allowing dynamic_cast and polymorphic typeid in constant expressions.
- Changing the active member of a union inside constexpr
- char8_t: a type for UTF-8 characters and strings.
- Access control in contract conditions.
- Revising the C++ memory model.
- Weakening release sequences.
- Nested inline namespaces
- Signed integers are two's complement

More C++ 20 Language Features

- **Modules!**
- **Merging the Coroutines TS into C++20**
- Allow initializing aggregates from a parenthesized list of values

- <=> != ==, an important fix to the default comparisons design.
- Extending structured bindings to be more like variable declarations.
- Reference capture of structured bindings.
- Contract postconditions and return type deduction.
- Array size deduction in *new-expressions*. This is also a Defect Report against previous versions of C++.
- Contra CWG DR1778 (a bugfix related to noexcept and explicitly defaulted functions).
- Make char16_t/char32_t string literals be UTF-16/32.

C++20 Library Features

- Support for detecting endianness programmatically
- Repairing elementary string conversions (also a Defect Report)
- Improvements to the integration of C++17 class template argument deduction into the standard library (also a Defect Report)
- Extending make_shared to support arrays

- Transformation trait remove_cvref
- Treating unnecessary decay
- Using nodiscard in the standard library
- Make std::memory_order a scoped enumeration
- Synchronized buffered ostream
- A utility to convert pointer-like objects to raw pointers
- Add constexpr modifiers to functions in <algorithm> and <utility> headers.
- constexpr for std::complex
- Atomic shared_ptr
- Floating-point atomics
- De-pessimize legacy <numeric> algorithms with std::move
- String prefix and suffix checking, i.e. starts_with() and ends_with()

More C++20 library Features

- **calendar and timezone library**.
- std::span
- <version> header
- Tweak on how unordered containers are compared
- String::reserve() should not shrink
- User specializations of function templates in namespace std
- Manipulators for C++ synchronized buffer ostream
- constexpr iterator requirements

- The most notable addition at this meeting was **standard library Concepts**.
- atomic_ref
- Bit-casting object representations
- Standard library specification in a Concepts and Contracts world
- Checking for the existence of an element in associative containers
- Add shift() to <algorithm>
- Implicit conversion traits and utility functions
- Integral power-of-2 operations
- The identity metafunction
- Improving the return value of erase()-like algorithms
- constexpr comparison operators for std::array
- constexpr for swap and related functions
- fpos requirements
- Eradicating unnecessarily explicit default constructors
- Removing some facilities that were deprecated in C++17 or earlier

More C++20 Library Features

- The most notable addition at this meeting was **merging the Ranges TS into C++20**!
- Fixing operator>>(basic_istream&, CharT*).
- variant and optional should propagate copy/move triviality.
- visit<R>: explicit return type for visit.
- <chrono> zero(), min(), and max() should be noexcept.
- constexpr in std::pointer_traits.
- Miscellaneous constexpr bits.
- unwrap_ref_decay and unwrap_reference
- reference_wrapper for incomplete types
- A sane variant converting constructor
- std::function move constructor should be noexcept
- std::assume_aligned
- Smart pointer creation with default initialization
- Improving completeness requirements for type traits)
- Remove CommonReference requirement from StrictWeakOrdering (a.k.a fixing relations)
- Utility functions to implement uses-allocator construction
- Should span be Regular?
- Make stateful allocator propagation more consistent for operator+(basic_string))
- Simplified partial function application
- Heterogeneous lookup for unordered containers
- Adopt consistent container erasure from Library Fundamentals v2

More C++20 Library Features

- polymorphic_allocator<> as a vocabulary type.
- Well-behaved interpolation for numbers and pointers., a.k.a. std::midpoint
- Signed ssize() functions, unsigned size() functions in span
- I stream, you stream, we all stream for istream_iterator.
- Ranges design cleanup
- Target vectorization policies (from the Parallelism TS v2)

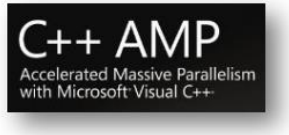
- Usability enhancements for std::span
- Make create_directory() intuitive.
- Precalculated hash values in lookup
- Traits for [un]bounded arrays
- Making std::underlying_type SFINAE-friendly.

Parallel/concurrency aiming for C++20

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous/Distributed
today's abstractions	C++11: thread,lambda function, TLS, async C++ 20: Jthreads +interrupt _token, coroutines	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParallelSTL, control false sharing C++ 20: Is_ready(), make_ready_future() , simd<T>, Vec execution policy, Algorithm un-sequenced policy, span	C++11: locks, memory model, mutex, condition variable, atomics, static init/term, C++ 14: shared_lock/shared_timed_mutex, OOTA, atomic_signal_fence, C++ 17: scoped _lock, shared_mutex, ordering of memory models, progress guarantees, TOE, execution policies C++20: atomic_ref, Latches and barriers, atomic<shared_ptr> Atomics & padding bits Simplified atomic init Atomic C/C++ compatibility Semaphores and waiting Fixed gaps in memory model , Improved atomic flags, Repair memory model	C++17: , progress guarantees, TOE, execution policies C++20: atomic_ref,

Act 2

- Is SYCL winning in the marketplace?



A tale of two cities

OpenMP™

CILK ARTS

SYCL™

C++ AMP
Accelerated Massive Parallelism with Microsoft Visual C++

PPL
stands for
Parallel Patterns Library
Abbreviations.com

OpenACC®
DIRECTIVES FOR ACCELERATORS

OpenCL

NVIDIA.
CUDA.

OH, East is East, and West is West,
and never the twain shall meet...
-Rudyard Kipling

OpenCL

OpenMP

SYCL

CUDA

The Quiet Revolution

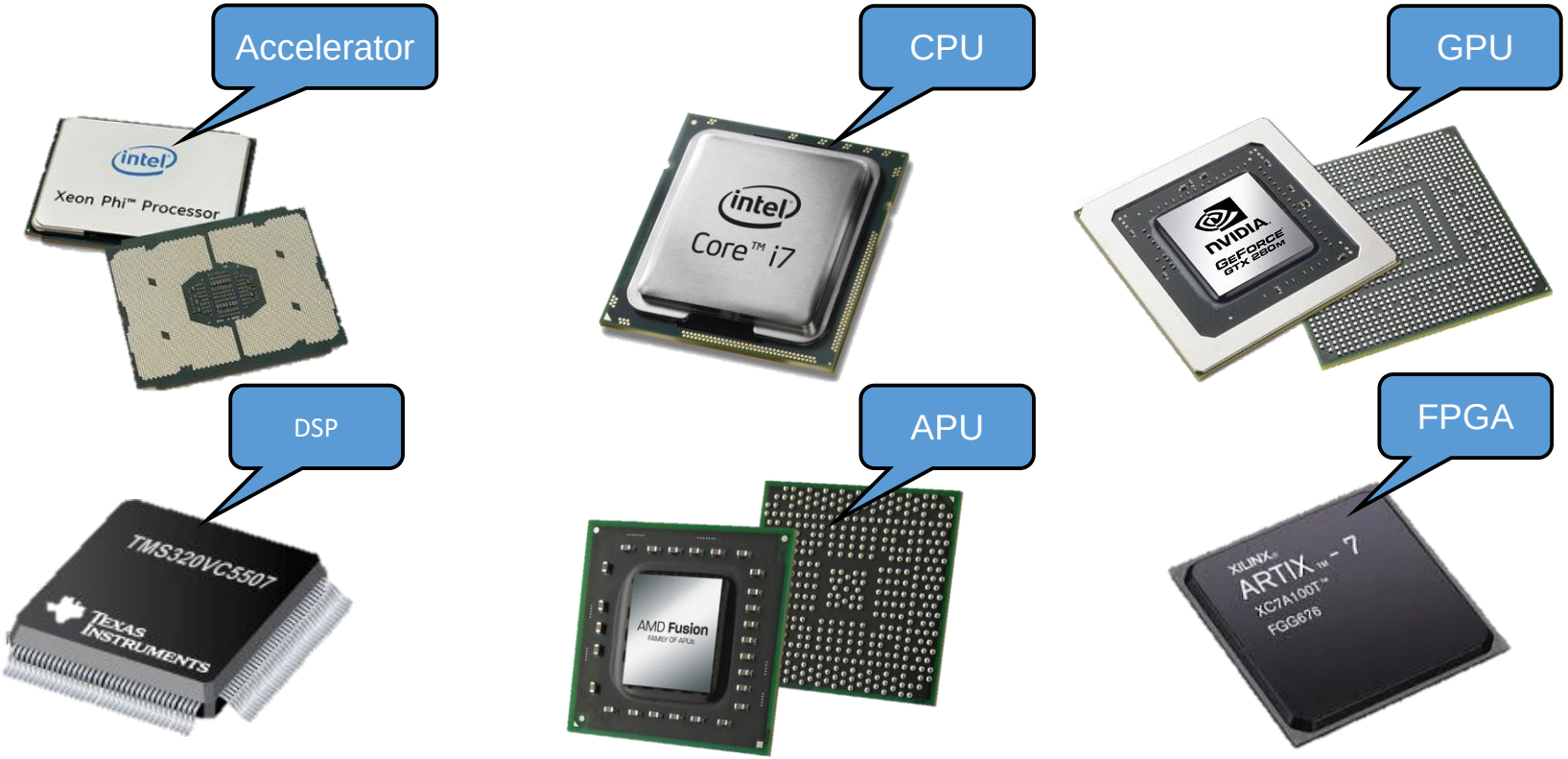
Kokkos

HPX

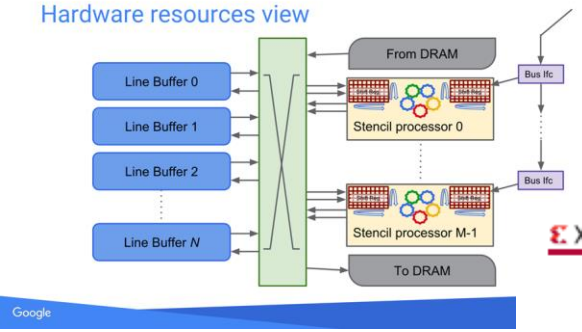
Raja

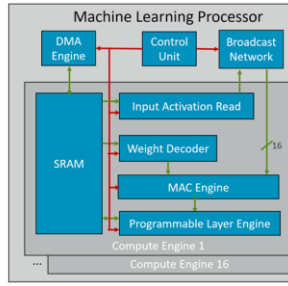
Boost.Compute

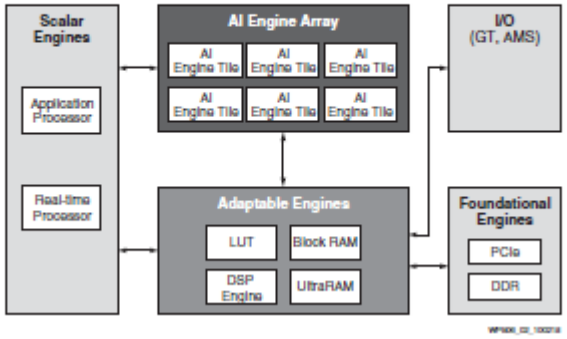
Heterogeneous Devices

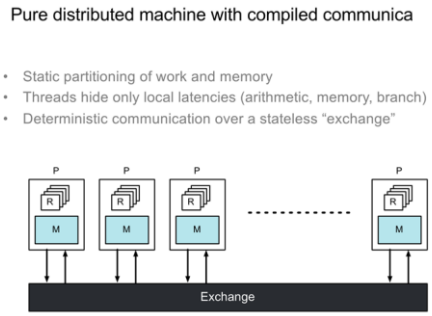


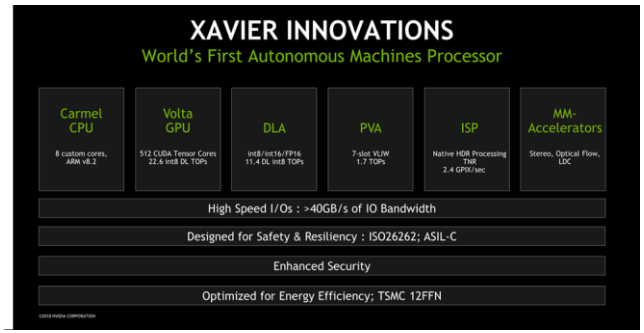
Hot Chips

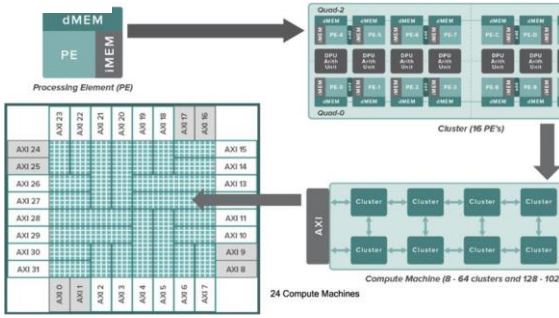

Hardware resources view
From DRAM
Line Buffer 0
Line Buffer 1
Line Buffer 2
Line Buffer N
Bus Ifc
Stencil processor 0
Bus Ifc
Stencil processor M-1
To DRAM
Google


Arm's ML processor: Summary
16 Compute Engines
~ 4 TOP/s of convolution throughput (at 1 GHz)
Targeting > 3 TOP/W in 7nm and ~2.5mm²
8-bit quantized integer support
1MB of SRAM
roid NNAPI and
Machine Learning Processor
DMA Engine
Control Unit
Broadcast Network
SRAM
Input Activation Read
Weight Decoder
MAC Engine
Programmable Layer Engine
Compute Engine 1
Compute Engine 16


Scalar Engines
Application Processor
Real-time Processor
AI Engine Array
AI Engine Tile
AI Engine Tile
AI Engine Tile
AI Engine Tile
AI Engine Tile
AI Engine Tile
I/O (GT, AMS)
Adaptable Engines
LUT
Block RAM
DSP Engine
UltraRAM
Foundational Engines
PCIe
DDR


Pure distributed machine with compiled communica
Static partitioning of work and memory
Threads hide only local latencies (arithmetic, memory, branch)
Deterministic communication over a stateless "exchange"
P
R
M
Exchange
ScaledML 2018


XAVIER INNOVATIONS
World's First Autonomous Machines Processor
Carmel CPU
8 custom cores, ARM v8.2
Volta GPU
512 CUDA Tensor Cores
22.6 int8 DL TOPs
DLA
int8/int16/FP16
11.4 DL int8 TOPs
PVA
7-slot VLIW
1.7 TOPs
ISP
Native HDR Processing
TNR
2.4 GPIX/sec
MM-Accelerators
Stereo, Optical Flow, LDC
High Speed I/Os : >40GB/s of IO Bandwidth
Designed for Safety & Resiliency : ISO26262; ASIL-C
Enhanced Security
Optimized for Energy Efficiency; TSMC 12FFN
13


dMEM
PE
iMEM
Processing Element (PE)
Quad-2
Quad-0
Cluster (16 PE's)
AXI
Cluster
Compute Machine (8 - 64 clusters and 128 - 102
24 Compute Machines

The landscape of C++ Heterogeneous computing

- Boost.Compute
- OpenMP 5
- C++AMP (Microsoft)
- TBB (Intel) - parallel threading abstraction for CPU (+OpenCL kernels).
- KOKKOS (Sandia) –parallel execution and data abstraction for CPU and GPU architectures (OpenMP, Pthreads, CUDA, …).
- RAJA (Livermore) –parallel execution for CPU and GPU architectures (OpenMP, TBB, CUDA, …). CHAI adds GPU data abstraction.
- Parallel STL (ISO standard) –parallel execution abstraction for CPU architectures; designed for future extensions for GPU, etc. (e.g. AMD Bolt, Nvidia Thrust, MS PPL).
- SYCL (Khronosstandard) -parallel execution and data abstraction that extends the OpenCL model (supports CPU, GPU, FPGA, …)
- HPX (LSU) – Distributed computing model using modern C++
- CUDA (Nvidia) – proprietary
- HCC/ROcM/Hip – AMD
- Agency (Nvidia research) – testing task dispatch and executors

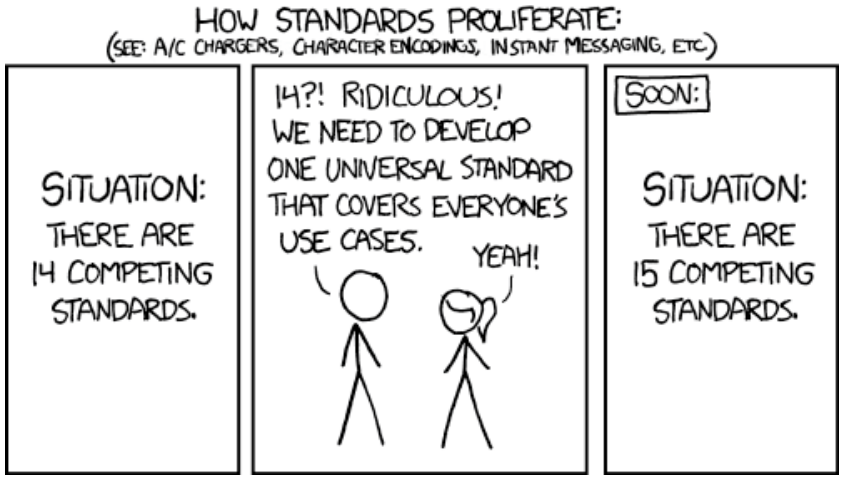
What is SYCL for?

- Modern C++ lets us separate the what from the how :
  - We want to separate **what** the user wants to do: *science, computer vision, AI …*
  - And enable the **how** to be: *run fast on an OpenCL device*
- Modern C++ supports and encourages this separation

What we want to achieve

- We want to enable a C++ ecosystem for OpenCL:
  - Must run on OpenCL devices: GPUs, CPUs, FPGAs, DSPs etc
  - C++ template libraries
  - Tools: compilers, debuggers, IDEs, optimizers
  - Training, example programs
  - Long-term support for current and future OpenCL features

Why a new standard?

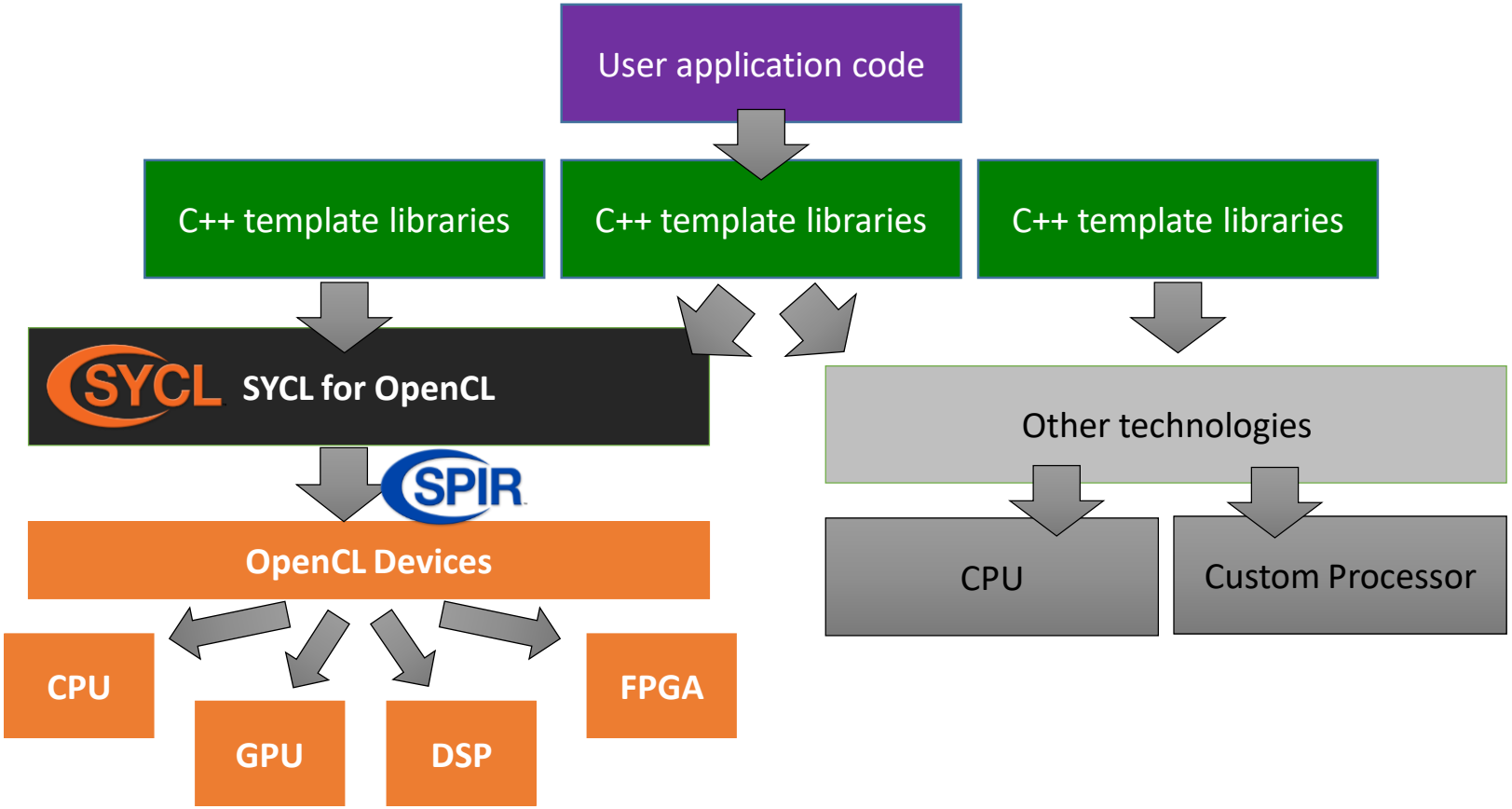


http://imgs.xkcd.com/comics/standards.png

- There are already very established ways to map C++ to parallel processors
  - So we follow the established approaches
- There are specifics to do with OpenCL we need to map to C++
  - We have worked hard to be an *enabler* for other C++ parallel standards
- We add no more than we need to

Where does SYCL fit in?

OpenCL / SYCL Stack



Philosophy

- With SYCL, we wanted to align with the direction the C++ standard is going
  - And we also need to future-proof for future OpenCL device capabilities

- Key decisions:
  - We will not add any language extensions to C++
  - We will work with existing C++ compilers
  - We will provide the full OpenCL feature-set in C++
  - Everything must compile and run on the host as well as an OpenCL device

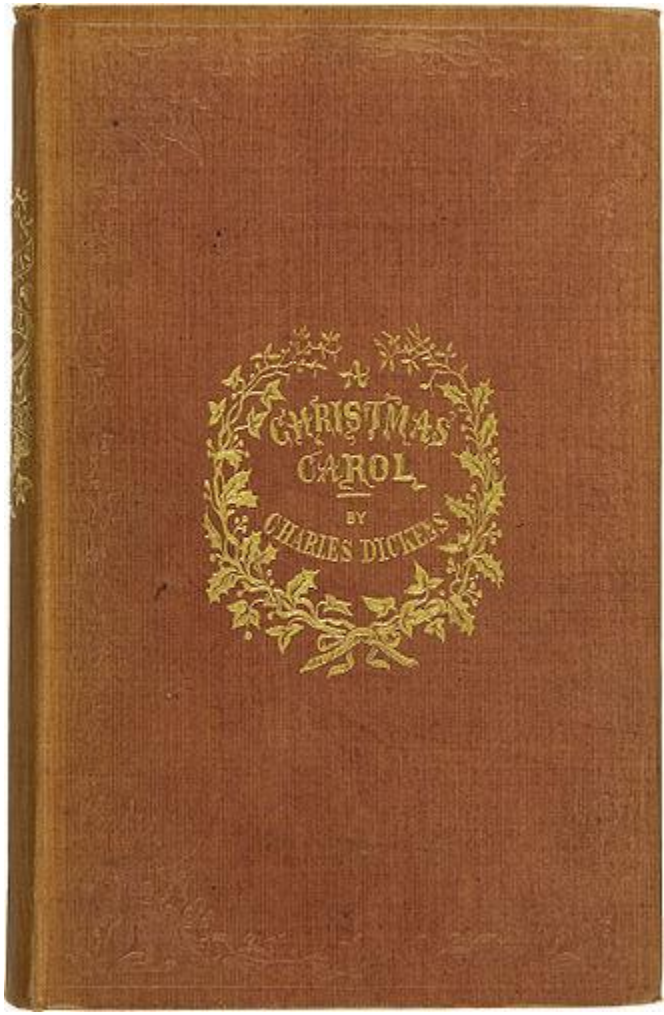
SYCL Ecosystem

- Ecosystem
  - SYCL ParallelSTL
  - SYCLBLAS
  - TensorFlow
  - ParallelSTL with Ranges
  - OpenSource ecosystem at Codeplay
  - CUDA to SYCL
  - Documentation
    - Prototype syclreference
    - https://mmha.github.io/syclreference/libraries/

- U of Bristol Benchmarks
- Parallel Research Kernels
- Adapting to workloads in HPC, Machine Learning, Vision processing

With Apologies to Charles Dickens,

I have NOT a bleak story to tell



SYCL is vibrant and Growing

- Tighter ISO C++ alignment in parallel - injecting our heterogeneous knowledge into ISO and adapting C++ features
- **Membership at all time high**
  - Intel, Xilinx, Codeplay, AMD, ARM, Imagination, Huawei, ETRI, NTHU
- **Actively pursuing new specifications**
  - SYCL 1.2.1 released in November 2017
    - Based on much user feedback from Tensorflow and ~7K downloads of free Codeplay's ComputeCPP
  - SOW approved, now stands alone outside of OpenCL
  - Starting work on SYCL next (tentatively SYCL 2019)
  - Feedback from non-member community through sycl.tech
- **Distributed and Heterogeneous Programming in C and C++ Workshop (DHPCC++) going strong in its 3rd year**
  - 60 people, 6 great talks + 2 great keynotes +panel

Ghost of SYCL Past

SYCL 1.2
C++11 Single source programming

OpenCL

2011

OpenCL 1.2
OpenCL C Kernel Language

OpenCL

2015

OpenCL 2.1
SPIR-V in Core

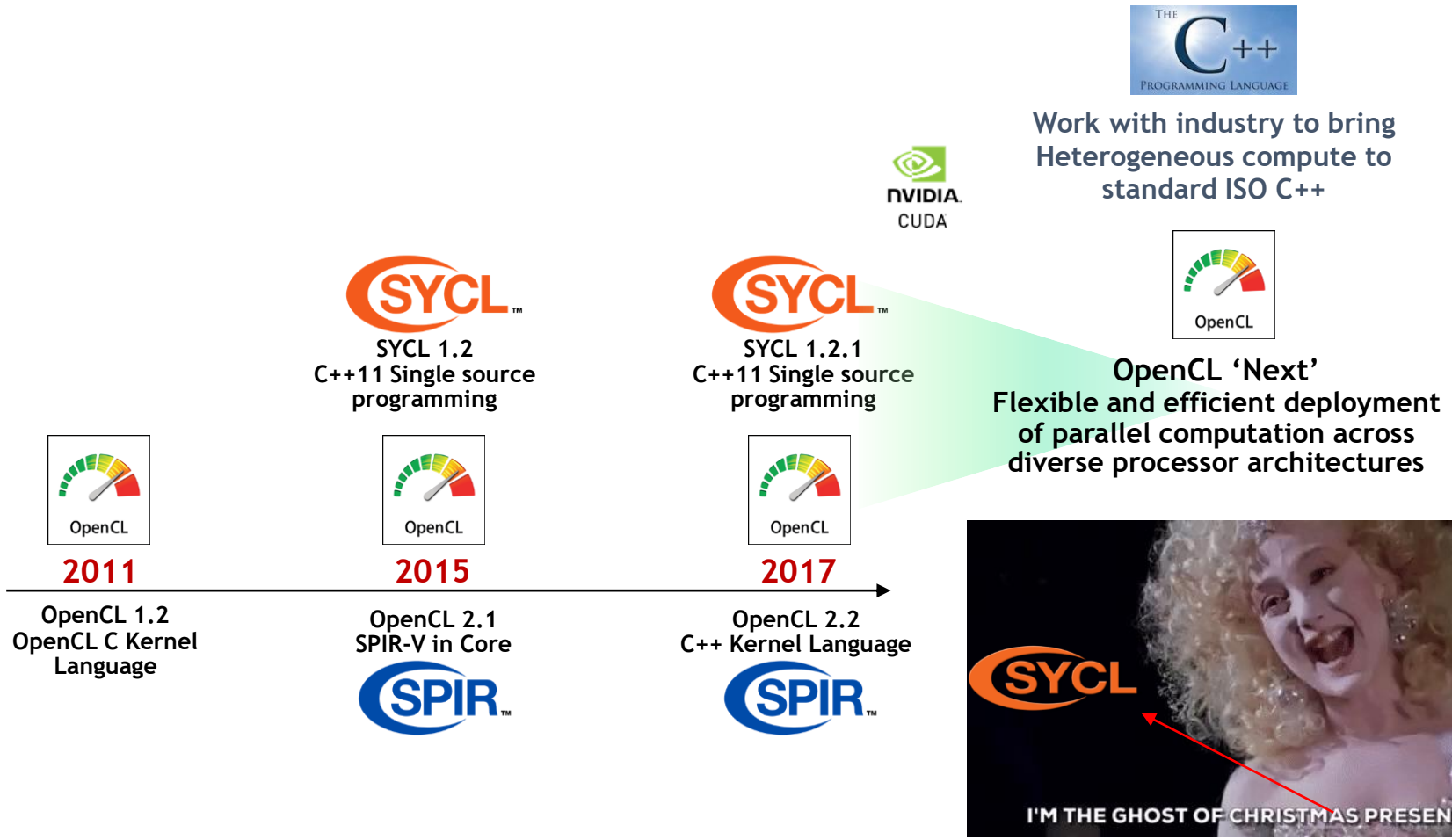
SPIR

KHRONOS GROUP

Our Observations then

- C++ and Heterogeneous are coming together
- Landscape moves really fast
  - SYCL moves was slow, but now even faster then C++
- No major API changes since 2015.
- No major C++-related features since then (e.g, no futures)
- SYCL is ***extremely well received*** on C++ community
- Success at CppCon talks, lots of downloads and user feedback
- Interest from research partners and other standard committee
  - **But Many didn't know OpenCL**

Ghost of SYCL Present

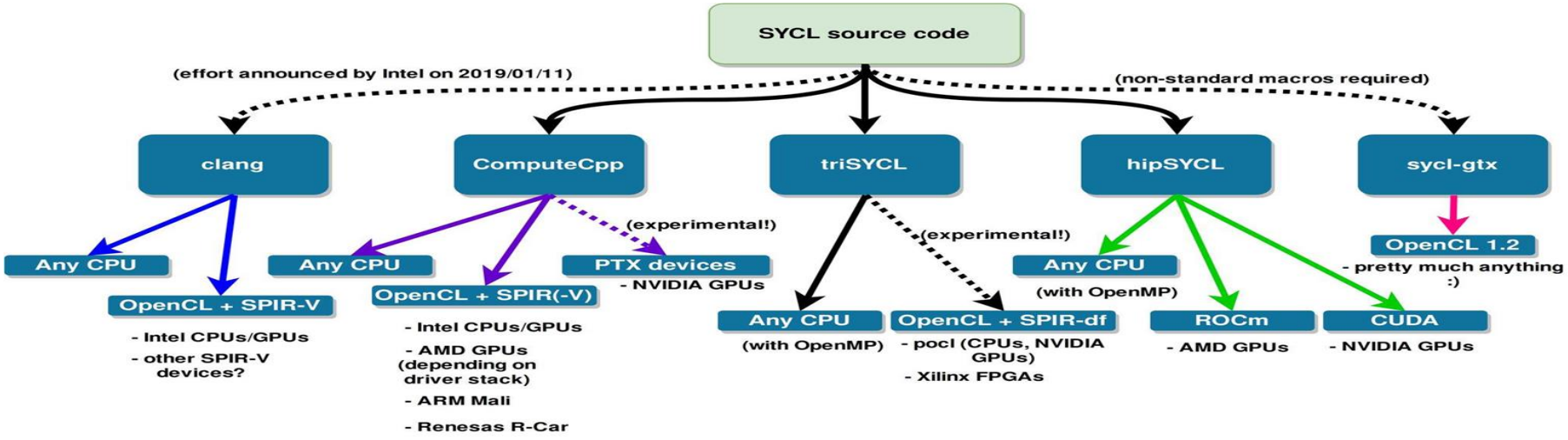


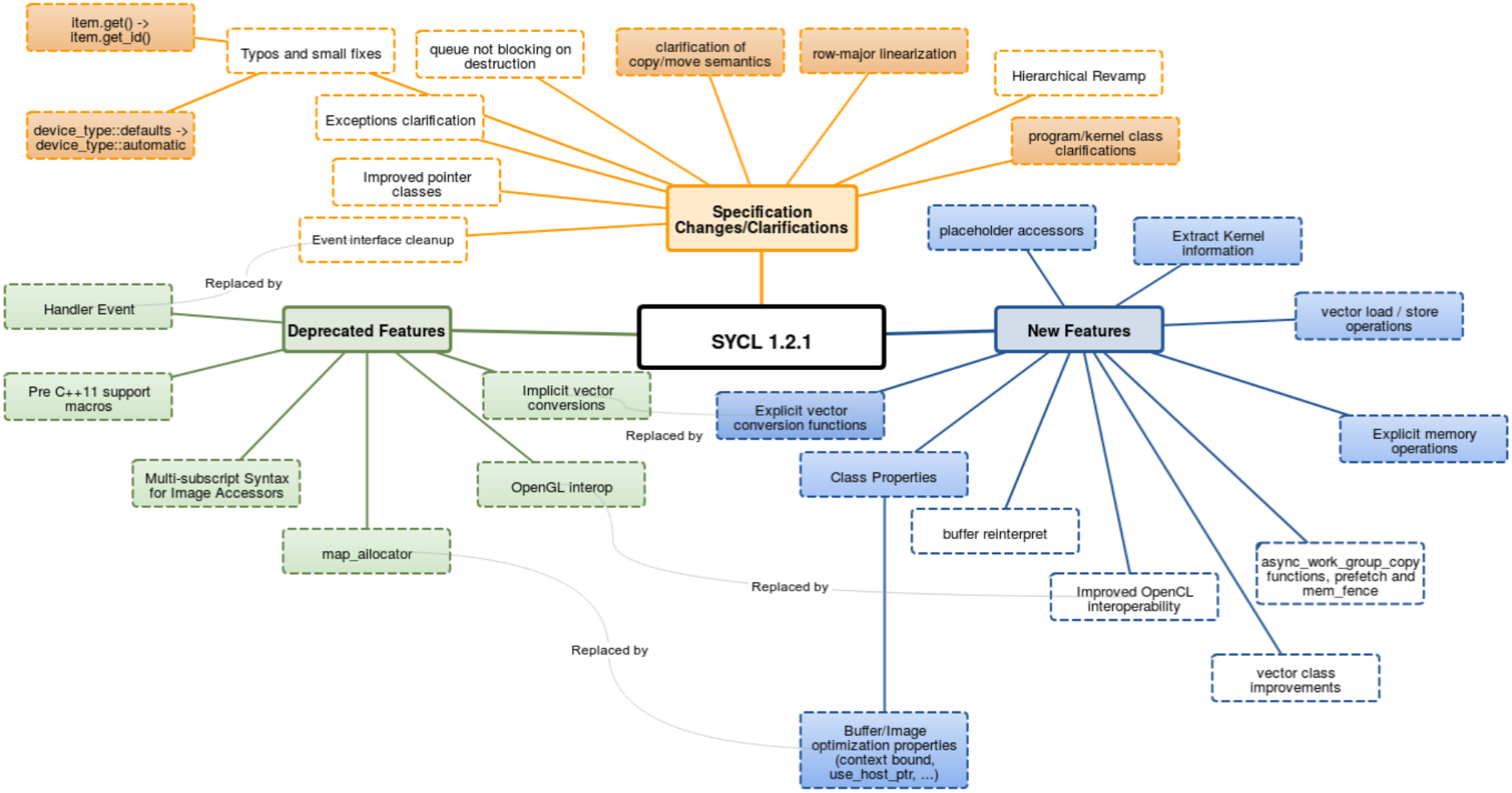
KHRONOS GROUP

Specification Current Status

- SYCL 1.2.1 / CTS Open Source
- SYCL 2019
  - Removed 2.2 provisional
  - Reviewed many MRs and Issues
  - Simpler, more accessible
- ++ WIP Implementations
  - Regular calls on clang collaboration

- ++ Participation + 3 individual
- IWOCL 2019 + 4th DHPCC++
- Part of Khronos Machine Learning
- Tighter alignment with ISO C++
- Plan to have SYCL bofs at SC, ISC, classes at CPPCON, SC, ISC




SYCL 1.2.1
Specification Changes/Clarifications
item.get() -> item.get_id()
Typos and small fixes
queue not blocking on destruction
clarification of copy/move semantics
row-major linearization
Hierarchical Revamp
device_type::defaults -> device_type::automatic
Exceptions clarification
program/kernel class clarifications
Improved pointer classes
Event interface cleanup
Replaced by
Handler Event
Deprecated Features
Pre C++11 support macros
Multi-subscript Syntax for Image Accessors
map_allocator
Implicit vector conversions
OpenGL interop
Replaced by
Replaced by
Replaced by
New Features
placeholder accessors
Extract Kernel information
vector load / store operations
Explicit vector conversion functions
Explicit memory operations
Class Properties
buffer reinterpret
async_work_group_copy functions, prefetch and mem_fence
Improved OpenCL interoperability
vector class improvements
Buffer/Image optimization properties (context bound, use_host_ptr, ...)

What is in an x.x.1? More than you think.

1.2. 

SYCL 1.2.1 for Machine Learning

- Vec load and store operations
  - It's now possible to perform vector load and store operations using the member functions `vec::load` and `vec::store`
- Add variadic function for setting kernel arguments for OpenCL interoperability
  - It is now possible to pass all kernel arguments in a single call using the member function `handler::set_args`
- Placeholder accessors
  - It is now possible to construct an accessor outside of a command group, registering it at a later time, using the placeholder specialisation
- Reinterpretable buffers
  - It is now possible to construct a new buffer from another buffer with a different type, as long as the size of the buffer remains the same
- Explicit copy operations
  - It is possible to direct the SYCL runtime to copy data in or out of the accelerator by triggering explicit copy operations.

Example 1: Reinterpretable Buffers

SYCL 1.2

```
constexpr size_t sizeInBytes = 1024;
auto byteRng = range<1>{sizeInBytes};
auto floatRng = range<1>{sizeInBytes / sizeof(float)};
constexpr size_t sizeInFloats = sizeInBytes / sizeof(float);

/* Construct initial buffer of bytes type */
buffer<byte, 1> byteBuf{data, byteRng};



 queue.submit([&](handler &cgh) {
    auto acc = buf.get_access<access::mode::write>(cgh);
    cgh.parallel_for<kernel>(floatRng, [=](id<1> idx) {
      func(static_cast<float>(acc[idx * sizeof(float)]));
    });
});
```

SYCL 1.2.1

```
constexpr size_t sizeInBytes = 1024;
auto byteRng = range<1>{sizeInBytes};
auto floatRng = range<1>{sizeInBytes / sizeof(float)};
constexpr size_t sizeInFloats = sizeInBytes / sizeof(float);

/* Construct initial buffer of bytes type */
buffer<byte, 1> byteBuf{data, byteRng};

/* Construct a reinterpreted buffer of float type */
auto floatBuf = byteBuf.reinterpret<float>(floatRng);

 queue.submit([&](handler &cgh) {
    auto acc = buf.get_access<access::mode::write>(cgh);
    cgh.parallel_for<kernel>(floatRng, [=](id<1> idx) {
      func(acc[idx]);
    });
});
```

Example 2: Explicit Copy Operations

SYCL 1.2

```
constexpr size_t size = 1024;

buffer<int, 1> buf{range<1>(size)};

std::vector<int> input = some_other_task();
buffer<int, 1> tmpBuf{input.data(), range<1>(size)};

  queue.submit([&](handler& cghh) {
  auto srcAcc=
    buf.get_access<access::mode::read>(cgh);
  auto destAcc =
    buf.get_access<access::mode::write>(cgh);

    /* Enqueue a no-op kernel to perform a copy */
    cgh.parallel_for<cpy>(range<1>(size)[=](id<1> idx){
      destAcc[idx] = src[idx];
    });
  });
```

SYCL 1.2.1

```
constexpr size_t size = 1024;

buffer<int, 1> buf{range<1>(size)};

std::vector<int> input = some_other_task();


  queue.submit([&](handler& cghh) {


  auto destAcc =
    buf.get_access<access::mode::write>(cgh);

    /* Enqueue a copy operation */
    cgh.copy<cpy>(input.data(), destAcc);


  });
```

Example 3: Placeholder Accessors

SYCL 1.2

```
constexpr size_t sizeInBytes = 1024;

buffer<int, 1> buf(data, range<1>(size));

myQueue.submit([&](handler &cgh) {
  auto acc = accessor<int, 1, access::mode::write,
    access::target::global_buffer(buf, cgh);

  cgh.parallel_for<kernel>(
    range<1>(dataSize), [=](id<1> idx) {
      func(acc[idx]);
    });
});
```

SYCL 1.2.1

```
constexpr size_t sizeInBytes = 1024;

buffer<int, 1> buf(data, range<1>(size));

auto acc = accessor<int, 1, access::mode::write,
  access::target::global_buffer,
  access::placeholder::true_t>(buf);

myQueue.submit([&](handler &cgh) {
  cgh.require(acc);

  cgh.parallel_for<kernel>(
    range<1>(dataSize), [=](id<1> idx) {
      func(acc[idx]);
    });
});
```

User feedback from SYCL 1.2.1 release

- SYCL Parallel STL is very well received by C++ community
  - Majority of people still don’t understand the limitations of GPU
- Some prefer migrating CUDA to SYCL than OpenCL
  - Single-source, templates and asynchronous features main drivers

Act 3

Is there a SYCL and C++ future Direction?

Ghost of SYCL Future (May Change)

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

SYCL 1.2
C++11 Single source programming

SYCL 1.2.1
C++11 Single source programming

SYCL 2019
C++17 Single source programming

SYCL 2021
C++20 Single source programming

OpenCL

2011

OpenCL 1.2
OpenCL C Kernel Language

2015

OpenCL 2.1
SPIR-V in Core

SPIR

2017

OpenCL 2.2
C++ Kernel Language

SPIR

201
9

202
1

KHRONOS GROUP

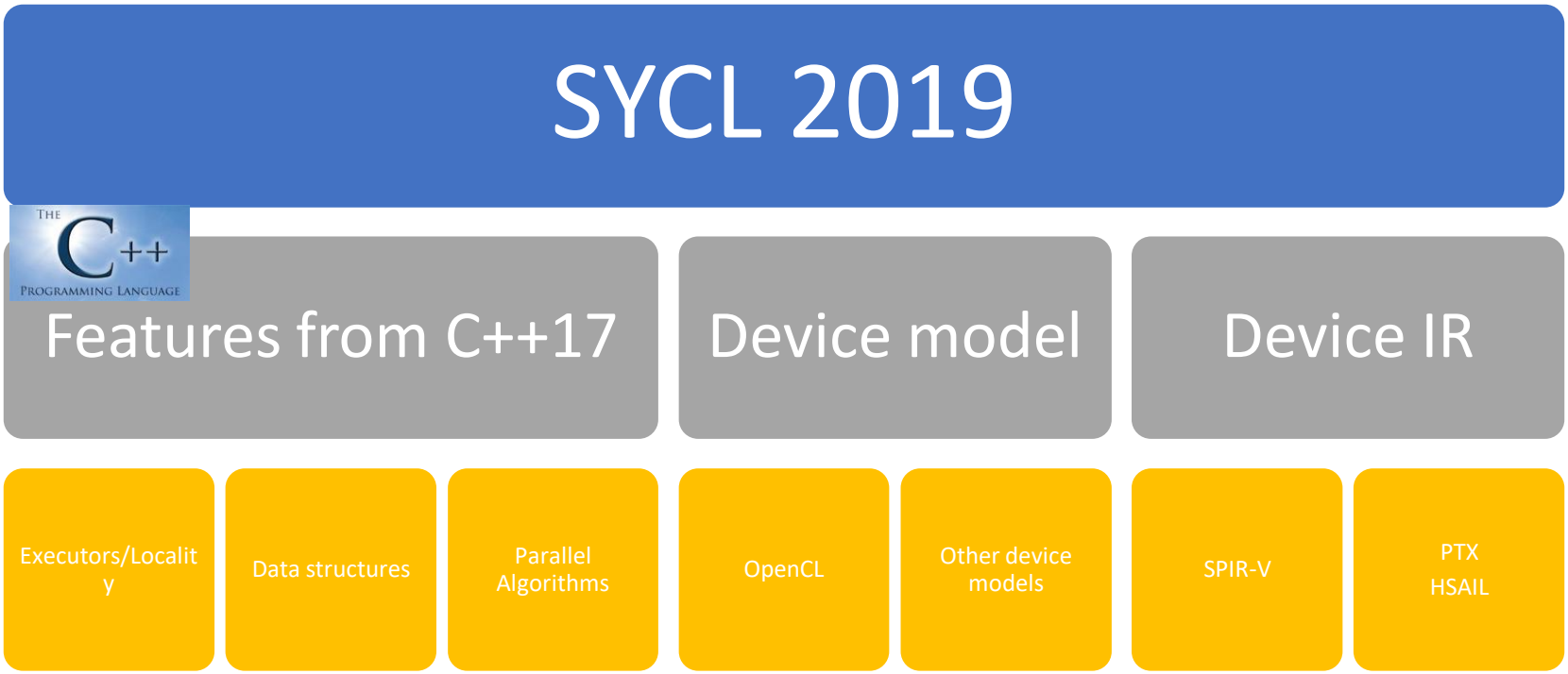
Work on SYCL 2019 is already starting

- Introduce a new naming scheme based on year of release
- Aiming for regular bus-train model of delivery every ~1.5 years
- Build on SYCL 1.2.1 and add user requested features
- Add Safety Critical Support
- Tighter ISO C++ alignment in parallel - injecting our heterogeneous knowledge into ISO and adapting C++ features

What about SYCL 2.2?

- Removed!

SYCL: Heterogeneous C++



Intersection with ISO C++

- Rendered feedback opinion on Modules keyword and Vulkan
- SG12's Cairo 2D Graphics interface may be should use Khronos OpenVG for 2D
- Or Khronos OpenGL/ Vulkan for 3D
- Interact with SG14 Linear Algebra
- SG19 Machine Learning
- Executors
- Affinity
- Futures
- Execution Context
- Concurrent Exception handling
- Pipes
- Execution Agent Local Storage
- C++ Parallelism TS2
  - SIMD

Features only in SYCL

- Single-source standard C++
- Both kernel and host language are in the same source file
- All code is standard C++ without non-conformant extensions
- Host execution + Fallback
- Command groups can be executed on the host if no device available
- Helps debugging
- Possibility of re-enqueue on the host if not possible on device
- Dataflow execution (accessors + requirements)
- Dependencies are used to define order of execution of command groups/kernels/operations
- Implicit memory operations
- Hierarchical parallelism
- Simpler and optimal interface to write algorithms in hierarchical memory architecture
- Other Parallel Patterns?
- Pipelines/Stages? Task farms? PSTL-Algorithms?

Features from OpenCL

- Platform Model
  - Platform/Context/Device/Queue
- Execution Model
  - Queue
  - Kernel execution
  - Synchronization points
- Memory model
  - Buffer/image
  - Global/local/private

OpenCL 1.2

SYCL 1.2.1

- Memory Model
  - Scoped atomics?
- Pipes?
- SVM?
  - Coarse/fine/system?
- Device-Enqueue?

OpenCL 2.2

SYCL 2019 candidates

Increase visibility of SYCL group/spec

Workshops/Tutorials/Events

Periodic reporting of SYCL progress to OpenCL group

Public drafts of the spec

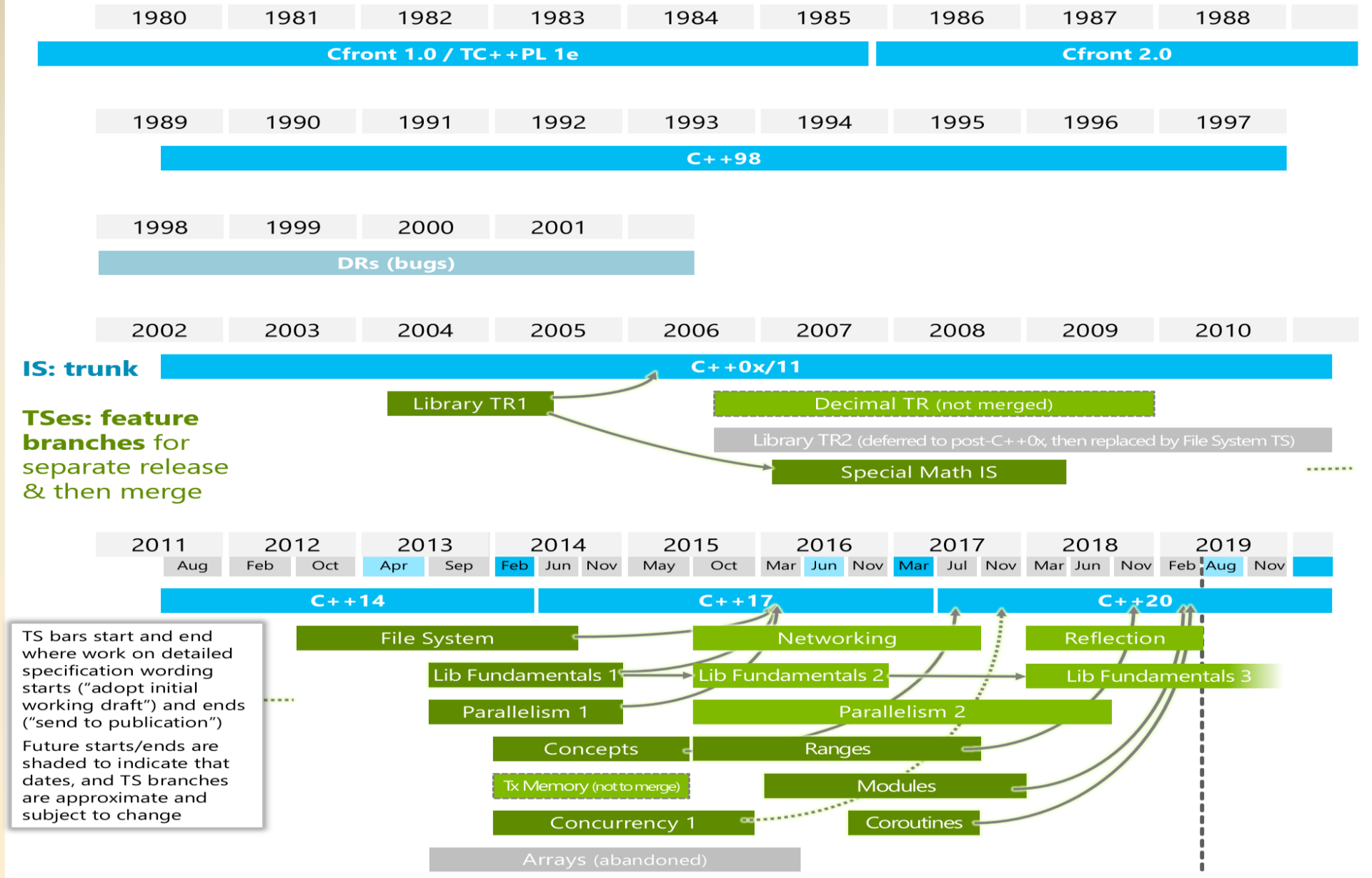
More frequent regular releases (like C++)

Books and other material

Call to Join the Ghost of SYCL Future

- Please join SYCL working group OR
- Join Advisory Board to advise us
  - michael@codeplay.com
  - opencl_sycl@khronos.org
- Visit sycl.tech for information
- What would you like to see in future SYCL
  - What in OpenCL 2.2 you need in SYCL?
  - What features you really need in SYCL?
- Tighter ISO C++ alignment in parallel - injecting our heterogeneous knowledge into ISO and adapting C++ features

ISO C++ Timeline https://isocpp.org/std/status



C++ Directions Group: P0939

Doc. no.: P0939r0
Date: 2018-02-10
Programming Language C++
Audience: All WG21
Reply to: Bjarne Stroustrup (bs@ms.com)

Direction for ISO C++

B. Dawes, H. Hinnant, B. Stroustrup, D. Vandevoorde, M. Wong

Revision History

- This is the initial version.

Main sections

I have a big idea for a big change

- Change gradually building on previous work
- OR
- Provide better alternative to existing feature


Many cooks (photos
by Bjarne Stroustrup)

I have a secret to tell you

Direction Group created as response to Call to Action of

Operating Principles for C++ by Heads of Delegation

C++ in danger of losing coherency due to proposals with differ and contradictory design philosophies

The Direction Group
direction@lists.isocpp.org

We try to represent USERS: the Interest of the larger C++ community

WG 21 Direction Group

What is C++

C++ is a language for defining and using lightweight abstractions

C++ supports building resource constrained applications and software infrastructure

C++ support large-scale software development

How do we want C++ to develop?

Improve support for large -scale dependable software

Improve support for high-level concurrency models

Simplify language use

Address major sources of dissatisfaction

Address major sources of error

C++ rests on two pillars

- **A direct map to hardware (initially from C)**
- Zero-overhead abstraction in production code (initially from Simula, where it wasn't zero-overhead)

Strengthen two pillars

Better support for modern hardware (e.g., concurrency, GPUs, FPGAs, NUMA architectures, distributed systems, new memory systems)

More expressive, simpler, and safer abstraction mechanisms (without added overhead)

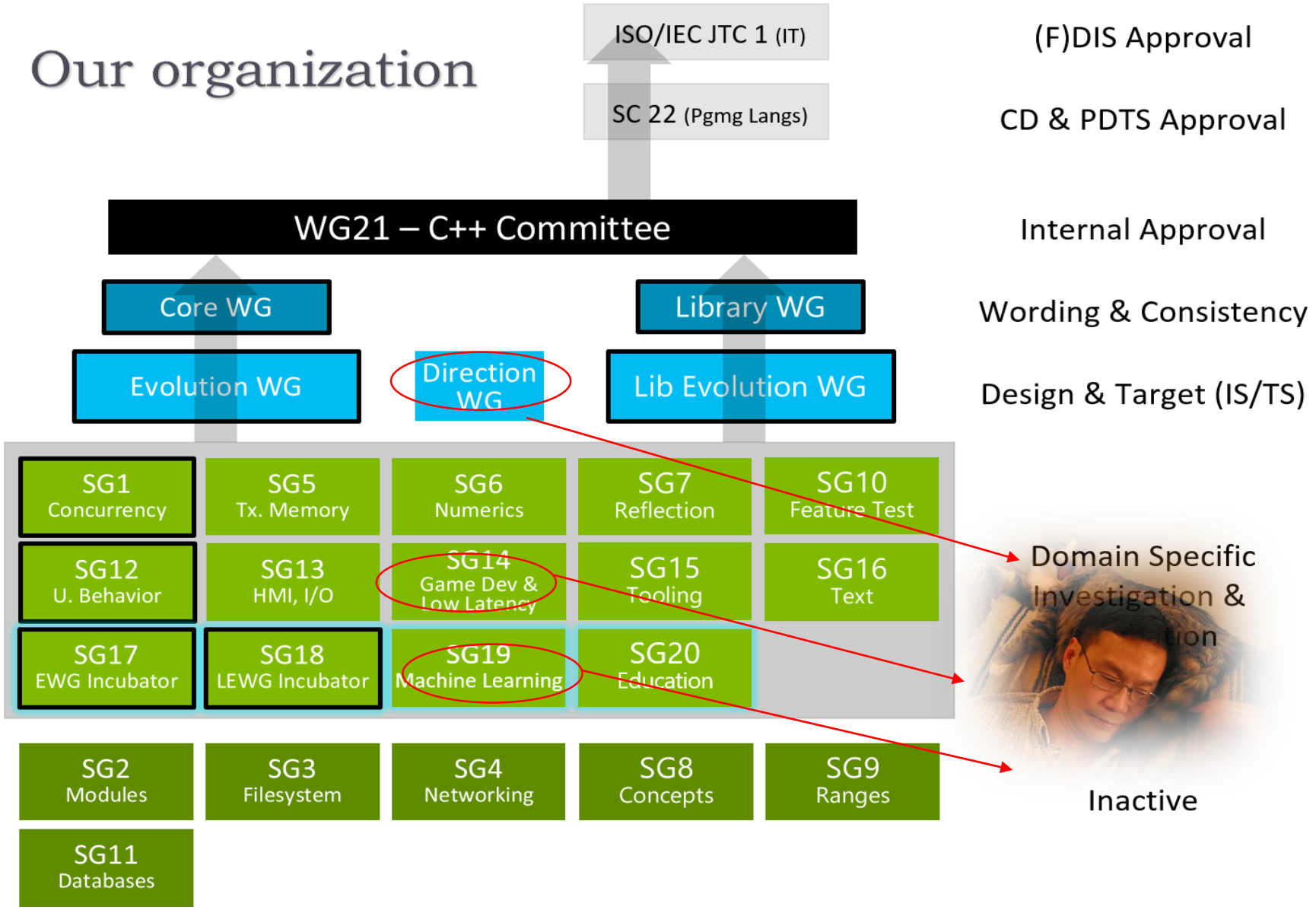
4.3 Concrete Suggestions

- **Pattern matching**
- **Exception and error returns**
- **Static reflection**
- **Modern networking**
- **Modern hardware:**
- *We need better support for modern hardware, such as executors/execution context, affinity support in C++ leading to* ***heterogeneous/distributed*** *computing support, SIMD/task blocks, more concurrency data structures, improved atomics/memory model/lock-free data structures support. The challenge is to turn this (incomplete) laundry list into a coherent set of facilities and to introduce them in a manner that leaves each new standard with a coherent subset of our ideal.*
- **Simple graphics and interaction**
- **Anything from the Priorities for C++20 that didn't make C++20**

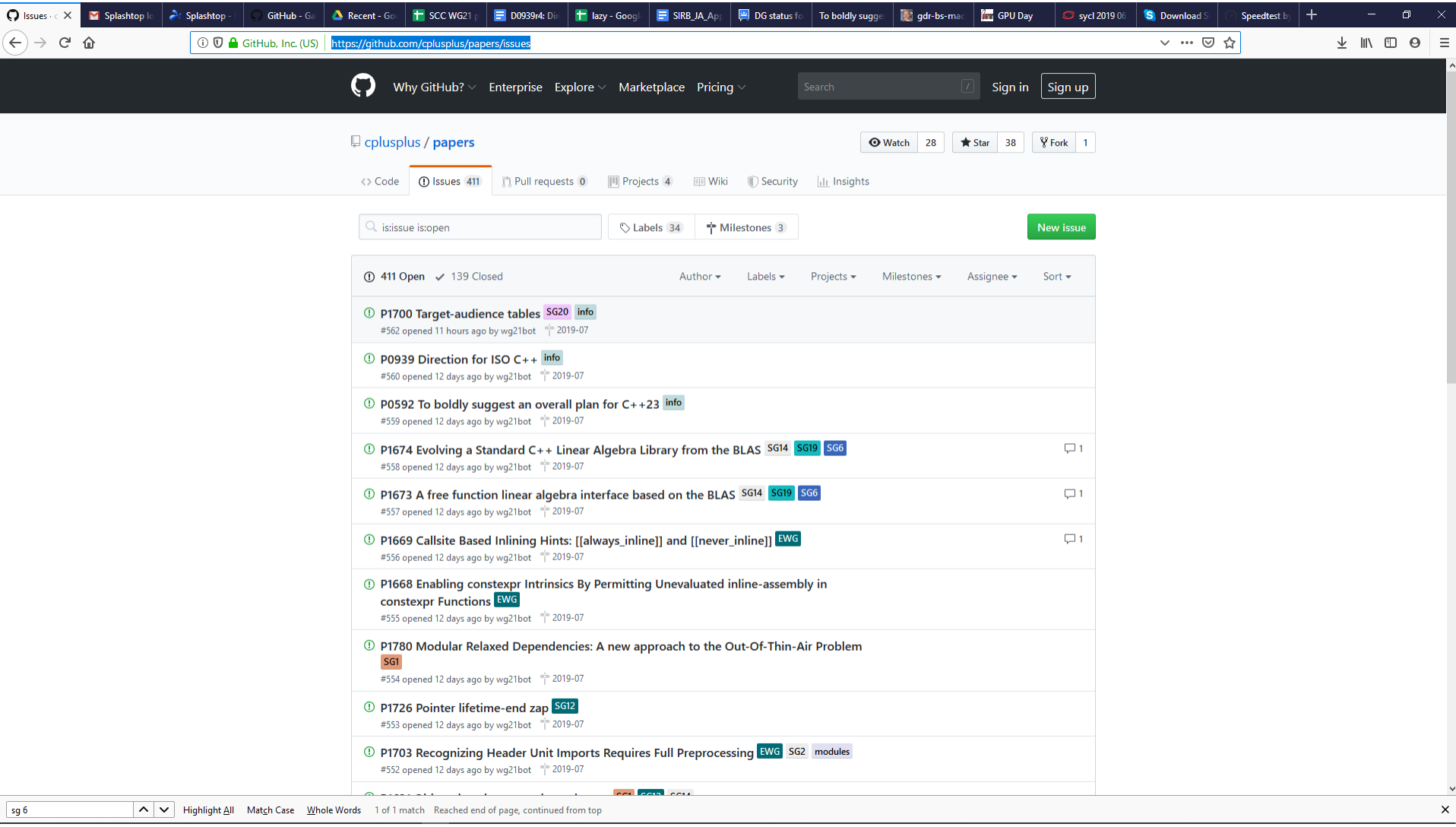
Modern hardware

- We need better support for modern hardware, such as executors/execution context, affinity support in C++ leading to heterogeneous/distributed computing support, ...

ISO C++ Standard



https://github.com/cplusplus/papers/issues



Take away

- C++ is pushing towards Heterogeneous device programming
- Adding Study Groups for Machine Learning, Graphics, Education, Linear Algebra, Low Latency
- C++ is good for AI and ML and still works for Legacy code
- C++20 will be MAJOR MAJOR release

What will be in C++ 20

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

Use the Proper Abstraction with C++

Abstraction	How is it supported
Cores	C++11/14/17 threads, async
HW threads	C++11/14/17 threads, async
Vectors	Parallelism TS2->C++20
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 P0796 on affinity
Distributed	HPX, MPI, UPC++ P0796 on affinity
Caches	C++17 false sharing support
Numa	Executors, Execution Context, Affinity, P0443->Executor TS
TLS	EALS, P0772
Exception handling in concurrent environment	EH reduction properties P0797

invoke async parallel algorithms future::then post

defer define_task_block dispatch asynchronous operations strand<>

Unified interface for execution

SYCL / OpenCL / CUDA / HCC

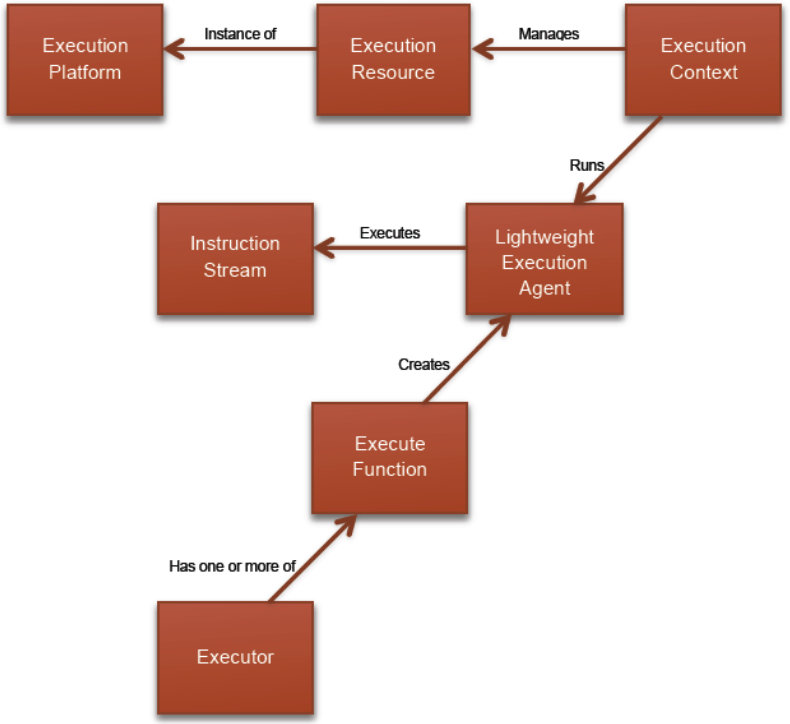
OpenMP / MPI

C++ Thread Pool

Boost.Asio / Networking TS

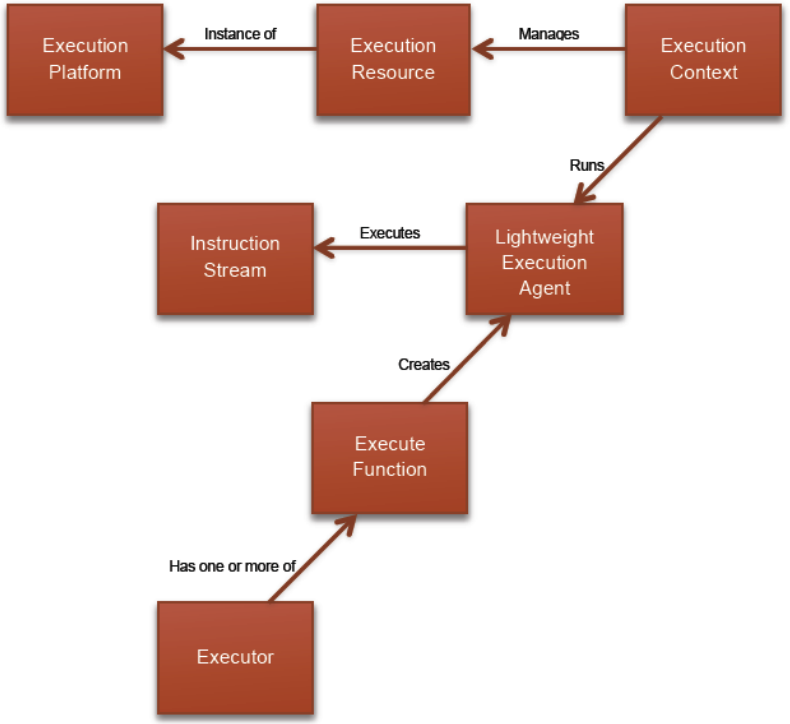
Current Progress of Executors

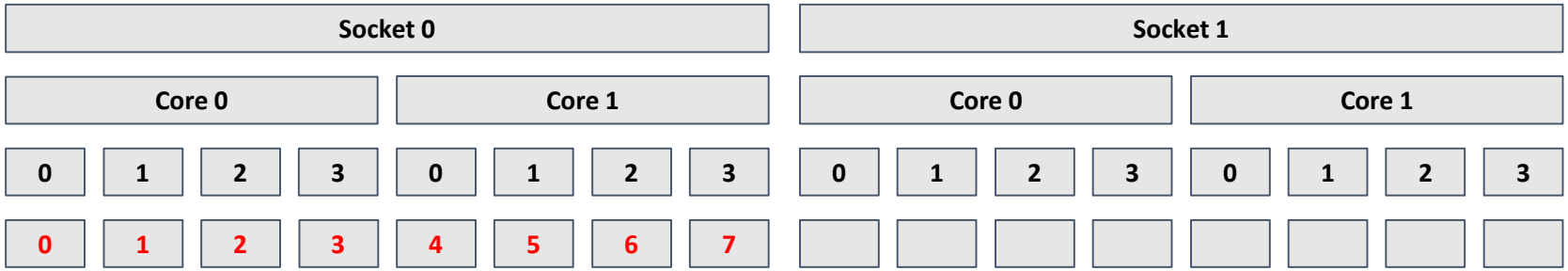
- An ***instruction stream*** is the function you want to execute
- An ***executor*** is an interface that describes where and when to run an ***instruction stream***
- An ***executor*** has one or more ***execute functions***
- An ***execute function*** executes an ***instruction stream*** on light weight ***execution agents*** such as threads, SIMD units or GPU threads



Current Progress of Executors

- An ***execution platform*** is a target architecture such as linux x86
- An ***execution resource*** is the hardware abstraction that is executing the work such as a thread pool
- An ***execution context*** manages the light weight ***execution agents*** of an ***execution resource*** during the execution






```
{
  auto exec = execution::execution_context{execRes}.executor();

  auto affExec = execution::require(exec, execution::bulk,
    execution::bulk_execution_affinity.compact);

  affExec.bulk_execute([](std::size_t i, shared s) {
    func(i);
  }, 8, sharedFactory);
}
```

Parallel/Concurrency beyond C++20: C++23

	Asynchronus Agents	Concurrent collections	Mutable shared state	Heterogeneous/DIstributed
today's abstractions	C++11: thread,lambda function, TLS, async C++14: generic lambda C++ 20: Jthreads +interrupt _token C++23: networking, asynchronous algorithm, reactive programming, EALS, async2, executors	C++11: Async, packaged tasks, promises, futures, atomics, C++ 17: ParallelSTL, control false sharing C++ 20: Is_ready(), make_ready_future(), simd<T>, Vec execution policy, Algorithm un-sequenced policy Executors Lite, span C++23: new futures, concurrent vector,task blocks, unordered associative containers, two-way executors with lazy sender-receiver models, concurrent exception handling, executors, mdspan	C++11: … C++ 14: … C++ 17: … C++20: atomic_ref, Latches and barriers atomic<shared_ptr> Atomics & padding bits Simplified atomic init Atomic C/C++ compatibility Semaphores and waiting Fixed gaps in memory model , Improved atomic flags , Repair memory model C++23: hazard_pointers, rcu/snapshot, concurrent queues, counters, upgrade lock, TM lite, more lock-free data structures, asymmetric fences	C++17: , progress guarantees, TOE, execution policies C++20: atomic_ref, mdspan, executors Lite C++23: affinity, pipelines, EALS, freestanding/embedded support well specified, mapreduce, ML/AI, reactive programming executors, mdspan

C++23

- Library support for coroutines
- Executors
- Networking
- A modular standard library

After C++20

- Much more libraries
  - Audio
  - Linear Algebra
  - Graph data structures
  - Tree Data structures
  - Task Graphs
  - Differentiation
  - Reflection
    - IPR paper
    - https://github.com/GabrielDosReis/ipr

A Principled, Complete, and Efficient Representation of C++

Gabriel Dos Reis and Bjarne Stroustrup

Abstract. We present a systematic representation of C++, called IPR, for complete semantic analysis and semantics-based program transformations. We describe the ideas and design principles that shaped the IPR. In particular, we describe how general type-based unification is key to minimal compact representation, fast type-safe traversal, and scalability. For example, the representation of a fairly typical non-trivial C++ program in GCC 3.4.2 was 32 times larger than its IPR representation; this led to significant improvements to GCC. IPR is general enough to handle real-world programs involving many translation units, archaic programming styles, and generic programming using C++0x extensions that affect the type system. The difficult issue of how to represent irregular (ad hoc) features in a systematic (non ad hoc) manner is among the key contributions of this paper. The IPR data structure can represent all of C++ with just 157 simple node types; to compare the ISO C++ grammar has over 700 productions. The IPR is used for a variety of program analysis and transformation tasks, such as visualization, loop simplification, and concept extraction. Finally, we report impacts of this work on existing C++ compilers.

1. Introduction

The C++ programming language [16] is a general-purpose programming language, with bias toward system programming. It has, for the last two decades, been widely used in diverse application areas [32, 33, 31]. Besides traditional applications of general-purpose programming languages, it is being used in high-performance com-

After C++23

- Reflection
- Pattern matching
- C++ ecosystem

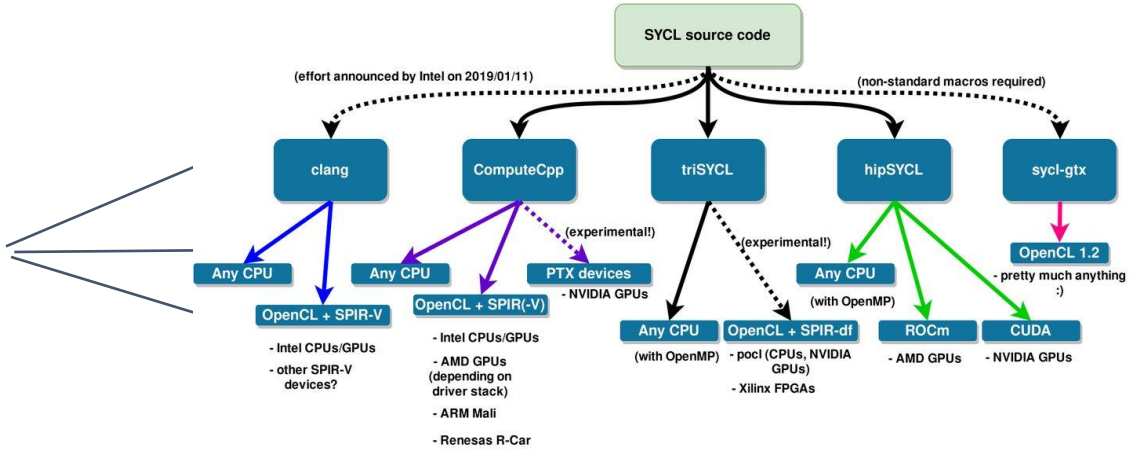

Ignite your applications
with XL Compilers
for C, C++, & Fortran
IBM Power
AIX
Linux

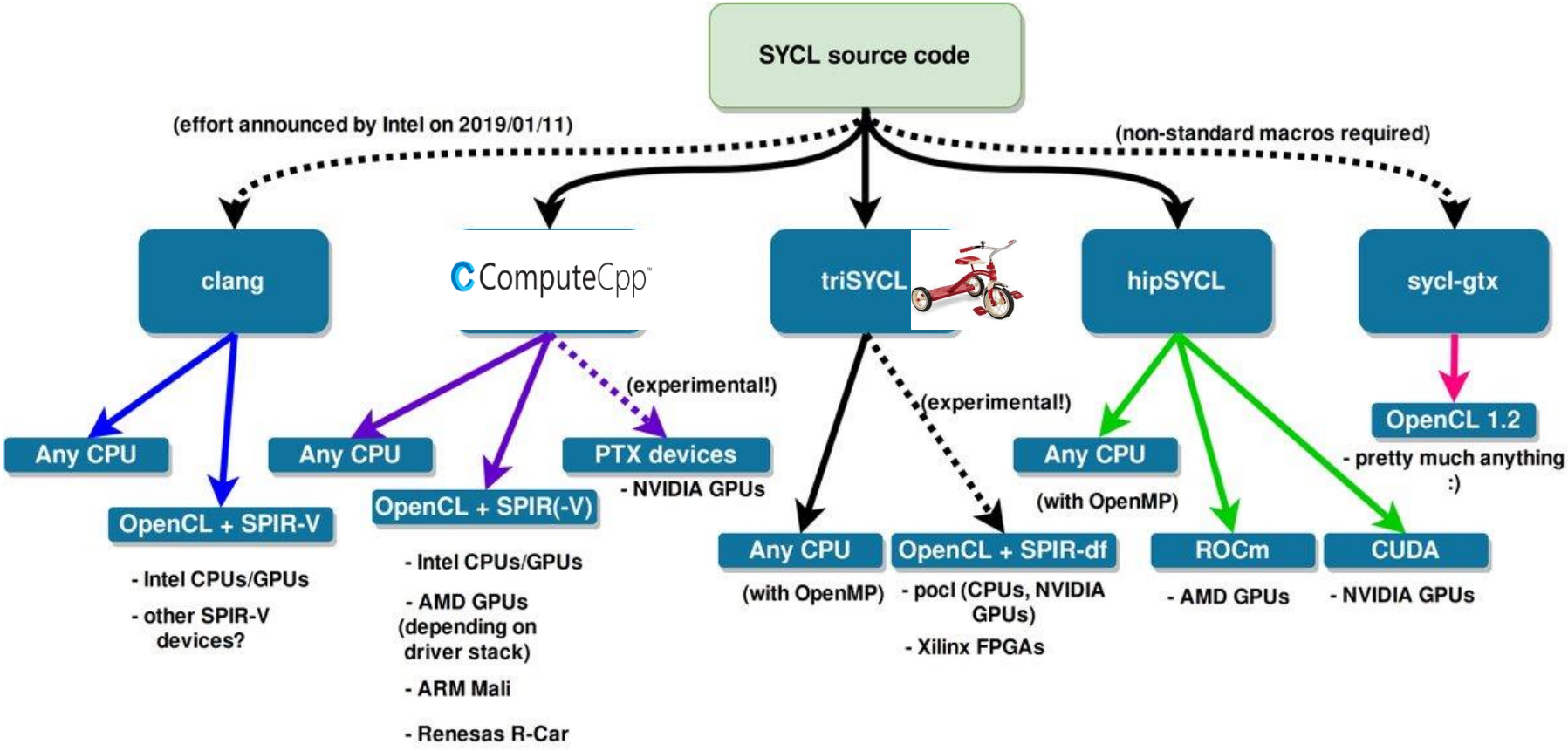

GCC


Visual C++


Edison
Design
Group


SYCL
TM


SYCL source code
(effort announced by Intel on 2019/01/11)
(non-standard macros required)
clang
ComputeCpp
triSYCL
hipSYCL
sycl-gtx
Any CPU
OpenCL + SPIR-V
- Intel CPUs/GPUs
- other SPIR-V devices?
Any CPU
OpenCL + SPIR(-V)
(experimental!)
PTX devices
- NVIDIA GPUs
- Intel CPUs/GPUs
- AMD GPUs (depending on driver stack)
- ARM Mali
- Renesas R-Car
(experimental!)
Any CPU
(with OpenMP)
OpenCL + SPIR-df
- pocl (CPUs, NVIDIA GPUs)
- Xilinx FPGAs
Any CPU
(with OpenMP)
ROCm
- AMD GPUs
OpenCL 1.2
- pretty much anything :)
CUDA
- NVIDIA GPUs


SYCL source code
(effort announced by Intel on 2019/01/11)
(non-standard macros required)
clang
ComputeCpp
triSYCL
hipSYCL
sycl-gtx
Any CPU
OpenCL + SPIR-V
- Intel CPUs/GPUs
- other SPIR-V devices?
Any CPU
OpenCL + SPIR(-V)
- Intel CPUs/GPUs
- AMD GPUs (depending on driver stack)
- ARM Mali
- Renesas R-Car
(experimental!)
PTX devices
- NVIDIA GPUs
Any CPU
(with OpenMP)
(experimental!)
OpenCL + SPIR-df
- pocl (CPUs, NVIDIA GPUs)
- Xilinx FPGAs
Any CPU
(with OpenMP)
ROCm
- AMD GPUs
CUDA
- NVIDIA GPUs
OpenCL 1.2
- pretty much anything :)

IWOCL DHPCC++ 2019 feedback

- reduction
- extension
- common address space
- A unified common address space.
- Common parallel algorithms such as reduce and sort as built-in commands in the SYCL API.
- A mechanism for supporting extensions and a flexible profile.
- Support for more than 3 dimensions in buffers, accessors and kernel invocations.
- A SYCL standard library; consisting of math functions, utilities and BLAS routines.
- A mechanism for pre-baking graphs in SYCL that can be executed multiple times, similar to CUDA graphs.
- A solution for the problem of having to name lambdas used as SYCL kernel functions.
- Support for more advanced access patterns for accessors, such as strided access and views adapters.
- A generalization of the different levels of iteration space sub-division. Re-introducing the multi_ptr subscript operator.

SYCL in 2019

- SYCL is being investigated for HPC
  - Keynote at DOE Performance Portability Shootout
    - **Xilinx FPGA and Codeplay PTX backend demonstrations at last SC17!**
    - **Plan SC19 BoF**
    - **Classes at CPPCON 2019, ISC 2020 SC2020**
    - **Intel One API and DPC++**
    - **Planned SYCL conference in 2020**
    - **NEED more HPC features!**

SYCL and C++
Convergence and Continued Research

Use the Proper Abstraction with C++

Abstraction	How is it supported
Cores	C++11/14/17 threads, async
HW threads	C++11/14/17 threads, async
Vectors	Parallelism TS2->C++20
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 P0796 on affinity
Distributed	HPX, MPI, UPC++ P0796 on affinity
Caches	C++17 false sharing support
Numa	Executors, Execution Context, Affinity, P0443->Executor TS
TLS	EALS, P0772
Exception handling in concurrent environment	EH reduction properties P0797

If you have to remember 3 things

1

Expose more parallelism

2

Increase Locality of reference

3

Use Heterogeneous C++ today

SYCL Ecosystem

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

Eigen Linear Algebra Library

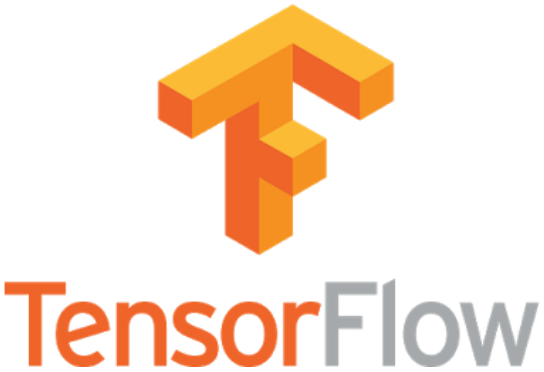
- SYCL backend in mainline
- Focused on Tensor support, providing support for machine learning/CNNs
- Equivalent coverage to CUDA
- Working on optimization for various hardware architectures (CPU, desktop and mobile GPUs)

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

TensorFlow

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

https://github.com/tensorflow/tensorflow



SYCL Ecosystem

- **Single-source heterogeneous programming using STANDARD C++**
  - Use C++ templates and lambda functions for host & device code
  - Layered over OpenCL
- **Fast and powerful path for bring C++ apps and libraries to OpenCL**
  - C++ Kernel Fusion - better performance on complex software than hand-coding
  - Halide, Eigen, Boost.Compute, SYCLBLAS, SYCL Eigen, SYCL TensorFlow, SYCL GTX
  - 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

Low Level Control

'GPGPU'-style separation of device-side kernel source code and host code

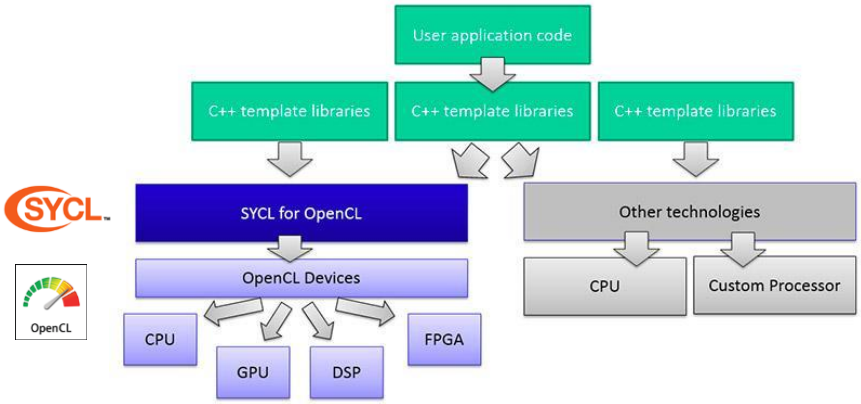
OpenCL

Single-source C++

Programmer Familiarity

Approach also taken by C++ AMP and OpenMP

SYCL™



Codeplay

Standards bodies

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

Research

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

Open source

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

Presentations

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

Company

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

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

Community Edition

Available now for free!

Visit:

computecpp.codeplay.com

- 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

codeplay

THE HETEROGENEOUS SYSTEMS EXPERTS

Questions ?

@codeplaysoft

/codeplaysoft

codeplay.com