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

Agenda

- A recap, C++17, the final report card. Is it great or just OK?
- C++20 and the future of C++
- Networking
- Concepts
- … more

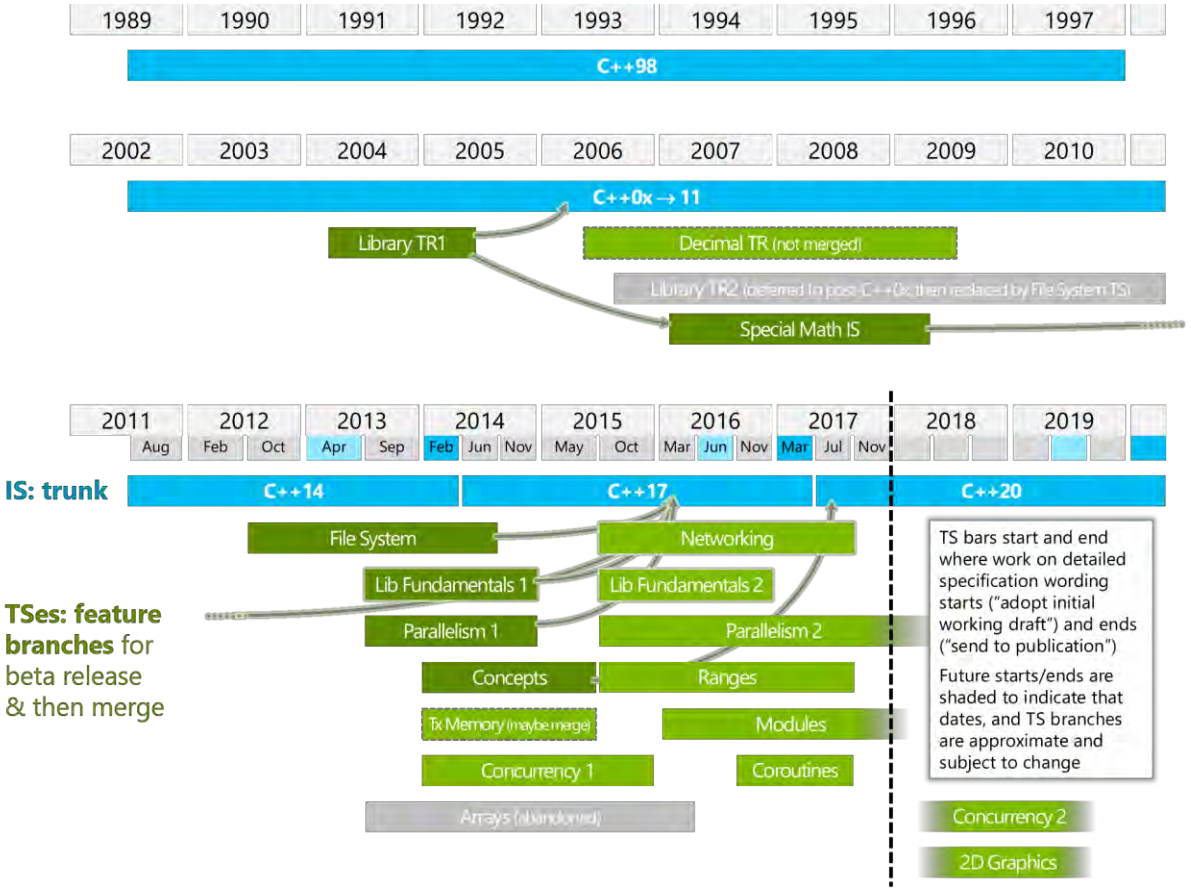
C++ 17 approved in Kona

C++11,14,17“No more Raw Food”

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

C++ Std Timeline/status

https://isocpp.org/std/status



C++ 17 Language features

- static_assert(condition) without a message
- Allowing auto var{expr};
- Writing a template template parameter as template <…> typename Name
- Removing trigraphs
- ***Folding expressions***
- std::uncaught_exceptions()
- Attributes for namespaces and enumerators
- Shorthand syntax for nested namespace definitions
- u8 character literals
- Allowing full constant expressions in non-type template parameters
- Removing the register keyword, while keeping it reserved for future use
- Removing operator++ for bool
- Making exception specifications part of the type system.
- __has_include(),
- Choosing an official name for what are commonly called "non-static data member initializers" or NSDMIs. The official name is "default member initializers".
- A minor change to the semantics of inheriting constructors

- The [[fallthrough]] attribute,
- The [[nodiscard]] attribute,
- The [[maybe_unused]] attribute
- Extending aggregate initialization to allow initializing base subobjects.
- Lambdas in constexpr contexts
- Disallowing unary folds of some operators over an empty parameter pack
- Generalizing the range-based for loop
- ***Lambda capture of *this by value***
- Relaxing the initialization rules for scoped enum types.
- Hexadecimal floating-point literals

C++17 Language features

if constexpr (formerly known as constexpr_if, and before that, static_if)

Template parameter deduction for constructors

template <auto N>

Inline variables

Guaranteed copy elision

Guarantees on expression evaluation order

Dynamic memory allocation for over-aligned data

is_contiguous_layout (really a library feature, but it needs compiler support)

Removing exception specifications

Using attribute namespaces without repetition

Replacement of class objects containing reference members

Standard and non-standard attributes

Forward progress guarantees: Base definitions

Forward progress guarantees for the Parallelism TS features

- Introducing the term 'templated entity'
- Proposed wording for structured bindings
- Selection statements with initializer
- Explicit default constructors and copy-list-initialization
- Not in C++17
  - **Default comparisons**
    - **For/against/neutral: 16/31/20**
  - **Operator dot**
    - **Not moved as CWG discovered a flaw**

C++17 Library Features

- Removing some legacy library components
- Contiguous iterators
- Safe conversions in unique_ptr<T[]>
- Making std::reference_wrapper trivially copyable
- Cleaning up noexcept in containers
- Improved insertion interface for unique-key maps
- void_t alias template
- invoke function template
- Non-member size(), empty(), and data() functions
- Improvements to pair and tuple
- bool_constant
- shared_mutex
- Incomplete type support for standard containers
- Type traits variable templates.
- as_const()
- Removing deprecated iostreams aliases
- Making std::owner_less more flexible
- Polishing <chrono>
- Variadic lock_guard
- Logical type traits.

- Re-enabling shared_from_this
- ***not_fn***
- constexpr atomic::is_always_lock_free
- Nothrow-swappable traits
- Fixing a design mistake in the searchers interface
- An algorithm to clamp a value between a pair of boundary values
- constexpr std::hardware_{constructive,destructive}_interference_size
- A 3-argument overload of std::hypot
- Adding constexpr modifiers
- Giving std::string a non-const data() member function
- is_callable, the missing INVOKE-related trait

C++17 Library features

- High-performance, locale-independent number <-> string conversions
- make_from_tuple() (like apply(), but for constructors)
- Letting folks define a default_order<> without defining std::less<>
- Splicing between associative containers
- Relative paths
- C11 libraries
- shared_ptr::weak_type
- gcd() and lcm() from LF TS 2
- Deprecating std::iterator, redundant members of std::allocator, and is_literal
- Reserve a namespace for STL v2
- ***std::variant<>***
- Better Names for Parallel Execution Policies in C++17
- Temporarily discourage memory_order_consume
- A <random> Nomenclature Tweak

- Synopses for the C library
- Making Optional Greater Equal Again
- Making Variant Greater Equal
- Homogeneous interface for variant, any and optional
- Elementary string conversions
- Integrating std::string_view and std::string
- has_unique_object_representations
- Extending memory management tools
- Emplace Return Type
- Removing Allocator Support in std::function
- make_from_tuple: apply for construction
- Delete operator= for polymorphic_allocator
- Fixes for not_fn
- Adapting string_view by filesystem paths
- Hotel Parallelifornia: terminate() for Parallel Algorithms Exception Handling

What is not in C++ 17

No Concepts

No Unified Call Syntax

No Default Comparison

No operator dot

- Inline variable stays

Changes voted in the last minute

`Fixes to C+17`

- Removing Deprecated Exception Specifications from C++17
- Added Elementary string conversions
- Std::byte was added

- https://isocpp.org/std/standing-documents/sd-6-sg10-feature-test-recommendations#recs.cpp17

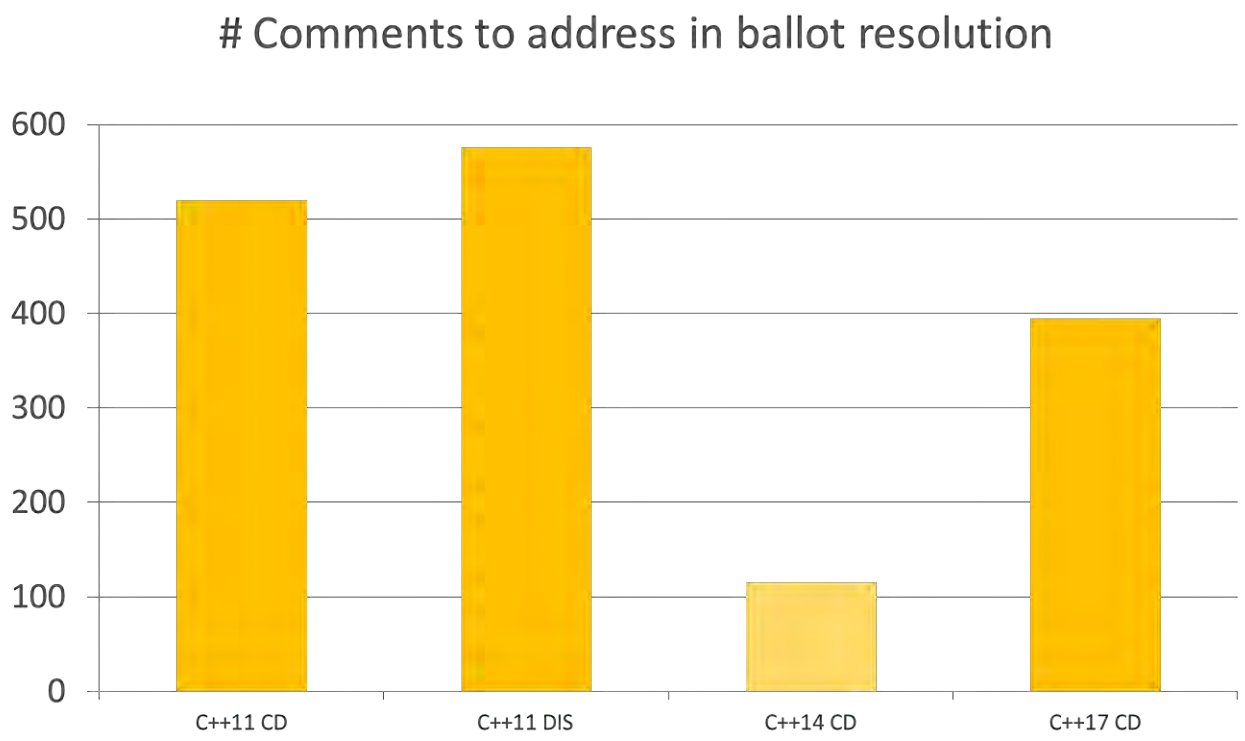
By the number of pages

- C++11 Std is
  - 1353 pages compared to 817 pages in C++03
- C++14 Std is
  - 1373 pages (N3937), n3972 (free)
- The new C++17 CD is
  - N4606: 1572 pages
- C99
  - 550 pages
- C11 is
  - 701 pages compared to 550 pages in C99

- OpenMP 3.1 is
  - 160 pages and growing
- OpenMP 4.0 is
  - 320 pages
- OpenMP 4.5 is
  - 359 pages
- OpenCL 2.0
  - 288 pages
- OpenCL 2.1
  - 300 pages
- OpenCL 2.2
  - 304 pages

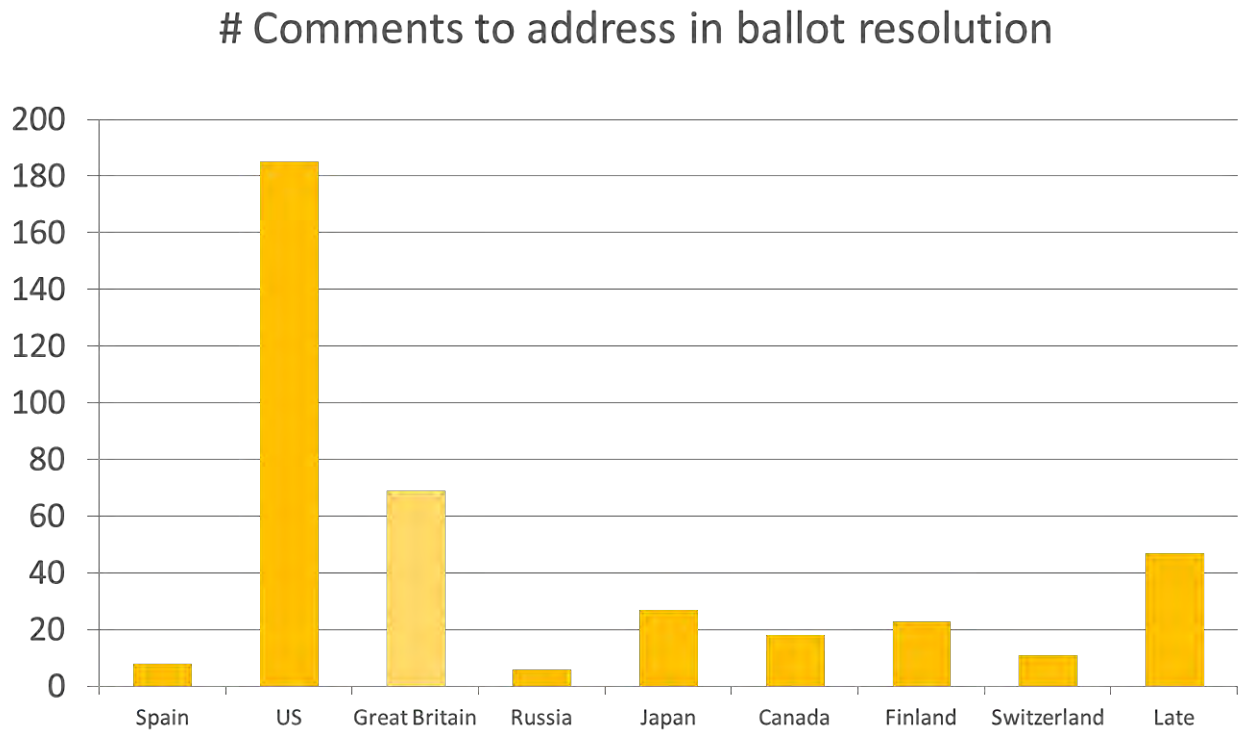
C++11/14/17: Stability

- Each round of international comment ballots generates bugs, tweaks, and requests



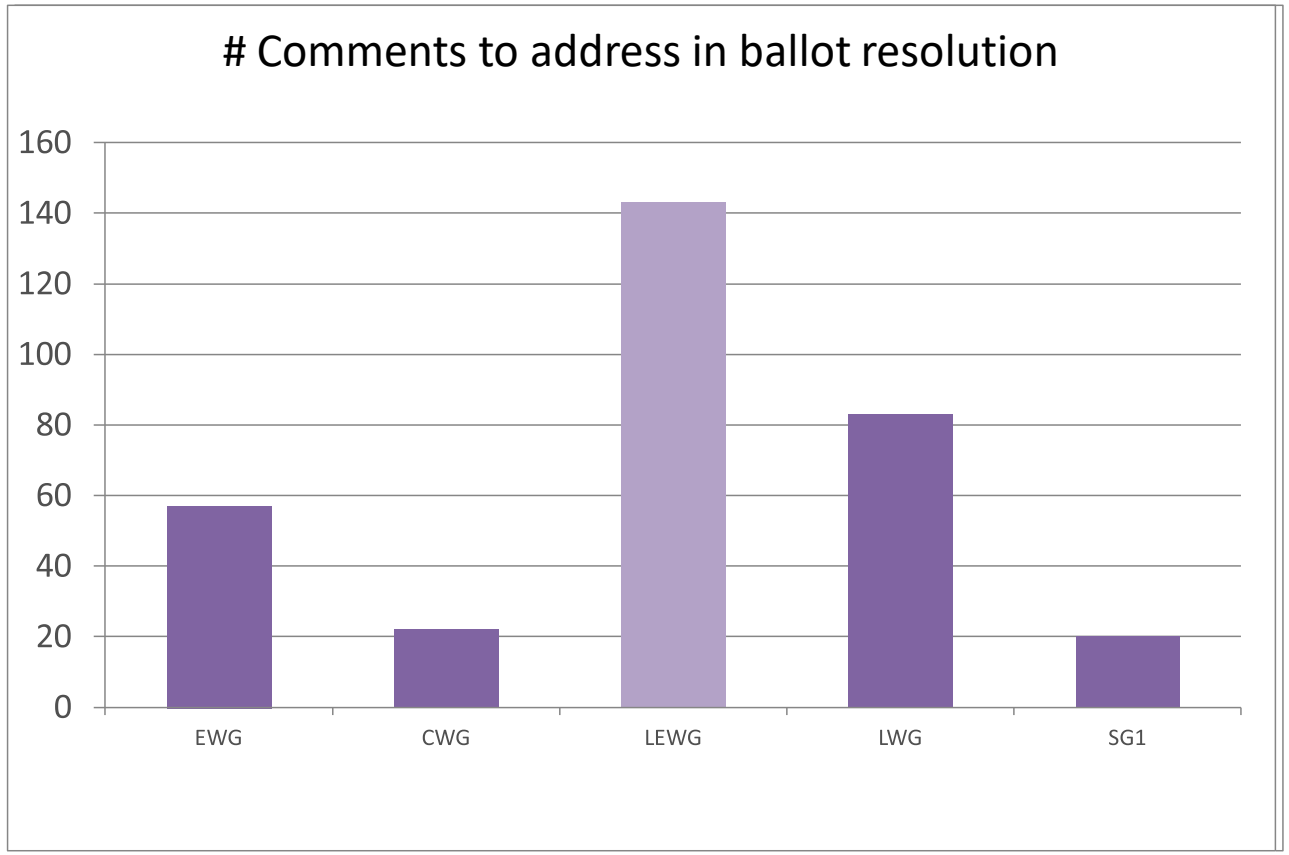
C++ 17: by Country

- Spain
- US
- Great Britain
- Russia
- Japan
- Canada
- Finland
- Switzerland
- Late



C++ 17: by EWG, CWG, LEWG, LWG, SG1

- Evolution
- Core
- Library Evolution
- Library
- Parallel/Concurrency



C++ 18 Goals

Improve support for large-scale dependable software

- Modules
  - to improve locality and improve compile time; n4465 and n4466
- Contracts
  - for improved specification; n4378 and n4415
- A type-safe union
  - probably functional-programming style pattern matching; something based on my Urbana presentation, which relied on the Mach7 library: Yuriy Solodkyy, Gabriel Dos Reis and Bjarne Stroustrup: Open Pattern Matching for C++. ACM GPCE'13.

Provide support for higher-level concurrency models

- Basic networking
  - asio n4478
- A SIMD vector
  - to better utilize modern high-performance hardware; e.g., n4454 but I'd like a real vector rather than just a way of writing parallelizable loops
- Improved futures
  - e.g., n3857 and n3865
- Co-routines
  - finally, again for the first time since 1990; N4402, N4403, and n4398
- Transactional memory
  - n4302
- Parallel algorithms (incl. parallel versions of some of the STL)
  - n4409

Simplify core language use and address major sources of errors

- Concepts (n3701 and n4361)
- concepts in the standard library
  - based on the work done in Origin, The Palo Alto TR, and Ranges n4263, n4128 and n4382
- default comparisons
  - to complete the support for fundamental operations; n4475 and n4476
- uniform call syntax
  - among other things: it helps concepts and STL style library use; n4474
- operator dot
  - to finally get proxies and smart references; n4477
- array_view and string_view
  - better range checking, DMR wanted those: "fat pointers"; n4480
- arrays on the stack
  - "stack_array" anyone? But we need to find a safe way of dealing with stack overflow; n4294
- optional
  - unless it is subsumed by pattern matching, and I think not in time for C++17, n4480

C++ 17 Report Card

Improve support for large-scale dependable software

- Modules
  - to improve locality and improve compile time; n4465 and n4466
- Contracts
  - for improved specification; n4378 and n4415
- A type-safe union
  - functional-programming style pattern matching; something based on my Urbana presentation, which relied on the Mach7 library: Yuriy Solodkyy, Gabriel Dos Reis and Bjarne Stroustrup: Open Pattern Matching for C++. ACM GPCE'13.

Provide support for higher-level concurrency models

- Basic networking
  - asio n4478
- A SIMD vector
  - to better utilize modern high-performance hardware; e.g., n4454 but I'd like a real vector rather than just a way of writing parallelizable loops
- Improved futures
  - e.g., n3857 and n3865
- Co-routines
  - finally, again for the first time since 1990; N4402, N4403, and n4398
- Transactional memory
  - n4302
- Parallel algorithms (incl. parallel versions of some of the STL
- n4409

Simplify core language use and address major sources of errors

- Concepts (n3701 and n4361)
- concepts in the standard library
  - based on the work done in Origin, The Palo Alto TR, and Ranges n4263, n4128 and n4382
- default comparisons
  - to complete the support for fundamental operations; n4475 and n4476
- uniform call syntax
  - among other things: it helps concepts and STL style library use; n4474
- operator dot
  - to finally get proxies and smart references; n4477
- array_view and string_view
  - better range checking, DMR wanted those: "fat pointers"; n4480
- arrays on the stack
  - "stack_array" anyone? But we need to find a safe way of dealing with stack overflow; n4294
- optional
  - unless it is subsumed by pattern matching, and I think not in time for C++17, n4480

~~May come back in limited form with National Body comment~~

~~May come back in limited form with National Body comment~~

The Verdict on C++17? (from reddit)

- You blew it
- Not a Major release
- No risk, no gain
- Nobody implement TSs
- Tethering tower of Babel of TSs

- Did a nice job
- But not Minor either
- Safe and conservative wins
- TSs are implemented
- Followed the rules of a bus train model, how to get 110 people to work together

A Medium/OK Release

Agenda

- A recap, C++17, the final report card. Is it great or just OK?
- C++20 and the future of C++
- Networking
- Concepts
- … more

short float

C++20 new features Kona

Overall direction plan:

- Concepts
- Modules
- Ranges
- Networking

- Pack expansions in *using-declarations*
- Lifting Restrictions on requires-Expressions
- Allowing attributes on template instantiations.
- Simplifying implicit lambda capture.
- Consistent comparisons.
- Static reflection.
- Implicit moving from rvalue references in return statements
- Contracts.

C++17 DIS

- In Kona
  - Address additional returned comments in February Kona
  - Issue DIS after Kona, Feb 2017, send it to National Body for final approval ballot; this is just an up/down vote, no comments
  - Will not be approved in time for July 2017 Toronto Meeting due to translation time
  - Then send it to ISO Geneva for publication, likely by EOY 2017

C++20 features Toronto

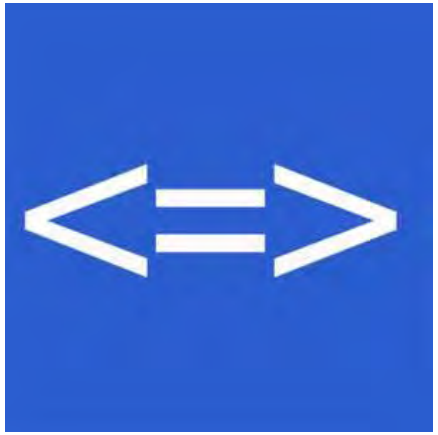
- Template parameter lists for generic lambdas.
- Designated initializers.
- Default member initializers for bitfields
- tweak to C++17's constructor template argument deduction rules
- Lambda capture [=, *this]
- Fixing const-qualified pointers to members
- __VA_OPT__ macro
- language defect related to defaulted copy constructors.
- allowing the template keyword in unqualified-ids
- attribute to mark unreachable code
- Down with typename!
- Removing throw().
- Ranged-based for statement with initializer.
- changes to the Modules TS and Concepts

- detecting endianness programmatically
- Repairing elementary string conversions
- Extending make_shared to support arrays
- Improvements to the integration of C++17 class template argument deduction into the standard library

C++ 20 Features just added in ABQ

- All Modules PDTS comment processed
- Range based for
- Simplify implicit lambda capture
- Spaceship operator for consistent comparison
- C++ ostream synchronized buffer
- Atomic<Shared_ptr>
- Floating point atomics
- Memory order is an enumeration

- After C++17
  - Default is 3 yr cycle: C++20, 23
- C++20 prediction
  - Concepts, ranges, Concurrency TS1/TS2, Parallelism TS2, Executor TS1, Reflection TS1, Coroutine TS1, Networking TS1, Modules TS1, Transactional Memory TS1, Numerics TS1, Heterogeneous TS1



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 Nov ABQ C++ Meeting

ISO number	Name	Status	links	C++17?
ISO/IEC TS 18822:2015	C++ File System Technical Specification	Published 2015-06-18. (ISO store). Final draft: n4100 (2014-07-04)	Standardize Linux and Windows file system interface	YES
ISO/IEC TS 19570:2015	C++ Extensions for Parallelism	Published 2015-06-24. (ISO Store). Final draft: n4507 (2015-05-05)	Parallel STL algorithms.	YES but removed dynamic execution policy, exception_lists, changed some names
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 19568:2015	C++ Extensions for Library Fundamentals	Published 2015-09-30, (ISO Store). Final draft: n4480 (2015-04-07)	optional, any, string_view and more	YES but moved Invocation Traits and Polymorphic allocators into LF TS2
ISO/IEC TS 19217:2015	C++ Extensions for Concepts	Published 2015-11-13. (ISO Store). Final draft: n4553 (2015-10-02)	Constrained templates	Merged into C++20 without terse syntax. . Already in GCC 6 release and and waiting for subsequent usage experience.

Status after Nov ABQ C++ Meeting

ISO number	Name	Status	What is it?	C++17?
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>> headed into C++20. Already in Visual Studio release and Anthony Williams Just Threads! and waiting for subsequent usage experience.
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	No.
ISO/IEC DTS 21425:xxxx	Ranges TS	PDTS, Draft n4651 (2017-03-15)	Range-based algorithms and views	No. Resolution of comments on Preliminary Draft in progress
ISO/IEC DTS 19216:xxxx	Networking TS	PDTS, Draft n4656 (2017-03-17)	Sockets library based on Boost.ASIO	No. Resolution of comments on Preliminary Draft in progress
ISO/IEC DTS 21544:xxxx	Modules	Proposed Draft n4689 (2017-07-31) out for ballot	A component system to supersede the textual header file inclusion model	No. First version based largely on Microsoft's design; hope to vote out Preliminary Draft at next meeting.

Status after Nov ABQ C++ Meeting

ISO number	Name	Status	What is it?	C++17?
	Numerics TS	Early development. Draft p0101 (2015-09-27)	Various numerical facilities	No. Under active development
ISO/IEC DTS 19571:xxxx	Concurrency TS 2	Early development	Exploring , lock-free, hazard pointers, RCU, atomic views, concurrent data structures	No. Under active development
ISO/IEC DTS 19570:xxxx	Parallelism TS 2	Early development. Draft n4578 (2016-02-22)	Exploring task blocks, progress guarantees, SIMD.	No. Under active development
ISO/IEC DTS 19841:xxxx	Transactional Memory TS 2	Early development	Exploring on_commit, in_transaction.	No. Under active development.
	Graphics TS	Early development. Draft p0267r0 (2016-02-12)	2D drawing API using Cairo interface, adding stateless interfacec	No. Wording review of the spec in progress
ISO/IEC DTS 19569:xxxx	Array Extensions TS	Under overhaul. Abandoned draft: n3820 (2013-10-10)	Stack arrays whose size is not known at compile time	No. Withdrawn; any future proposals will target a different vehicle

Status after Nov ABQ C++ Meeting

ISO number	Name	Status	What is it?	C++17?
ISO/IEC DTS 22277:xxxx	Coroutine TS	PDTS. Draft n4663 (2017-03-25)	Resumable functions, based on Microsoft's await design	Preliminary Draft voted out for balloting by national standards bodies
	Reflection TS	Early development. Draft p0194r2 (2016-10-15) with rationale in p0385r2 (2017-02-06). Alternative: p0590r0 (2017-02-05)	Code introspection and (later) reification mechanisms	No. Introspection proposal passed core language design review; next stop is design review of the library components. Targeting a Reflection TS.
	Contracts TS	Unified proposal reviewed favourably.)	Preconditions, postconditions, etc.	No. Proposal passed core language design review; next stop is design review of the library components. Targeting C++20.
	Executor TS	Separated from Concurrency TS. have a unified proposal .	Describes how, where, when of execution. Enables distributed and heterogeneous computing.	No. bi-weekly calls
	Heterogeneous Device TS	Managed_ptr and Channels proposal.	Support Hetereogeneous Devices	No. Under active development.
	C++17	Draft International Standard published; on track for final publication by end of 2017	Filesystem TS, Parallelism TS, Library Fundamentals TS I, if constexpr, and various other enhancements are in. See slide 44-47 for details.	YES

Agenda

- A recap, C++17, the final report card. Is it great or just OK?
- C++20 and the future of C++
- Networking
- Concepts
- … more

1. Using the executors library: a two minute introduction

Run a function asynchronously.

```
#include <experimental/executor>
using std::experimental::post;
int main()
{
    post([]{
      // ...
    });
}
```

Using the executors library: a two minute introduction

Run a function asynchronously on your own thread pool

```
#include <experimental/executor>
#include <experimental/thread_pool>
using std::experimental::post;
using std::experimental::thread_pool;
int main()
{
    thread_pool pool;
    post(pool, []{
        // ...
    });
    pool.join();
}
```

Using the executors library: a two minute introduction

Run a function asynchronously.

Wait for the result.

```
#include <experimental/executor>
#include <experimental/future>
#include <iostream>
using std::experimental::post;
using std::experimental::package;
int main()
{
	std::future<int> f =
		post(package([]{
			// ...
			return 42;
		}));
	std::cout << f.get() << std::endl;
}
```

Using the executors library: a two minute introduction

Run a function asynchronously on your own thread pool.

Wait for the result.

```
#include <experimental/executor>
#include <experimental/future>
#include <experimental/thread_pool>
#include <iostream>
using std::experimental::post;
using std::experimental::package;
using std::experimental::thread_pool;
int main()
{
	thread_pool pool;
	std::future<int> f =
		post(pool, package([]{
			// ...
			return 42;
	}));
	std::cout << f.get() << std::endl;
}
```

Using the executors library: a two minute introduction

Run a function in the future.

Wait for the result.

```
#include <experimental/executor>
#include <experimental/future>
#include <experimental/timer>
#include <iostream>
using std::experimental::post_after;
using std::experimental::package;
int main()
{
	std::future<int> f =
		post_after(
			std::chrono::seconds(1),
			package([]{
				// ...
				return 42;
			}));
	std::cout << f.get() << std::endl;
}
```

Networking Executor

- *Executors are to function execution as allocators are to memory allocation*
- An executor is a set of rules governing where, when and how to run a function object.
- Like allocators, executors are lightweight and cheap to copy.
- Examples:
  - The system executor
  - A strand

Execution Context

- An execution context is a place where function objects are executed.
- Examples:
  - A fixed-size thread pool
  - A loop scheduler
  - An asio::io_service
  - The set of all threads in the process

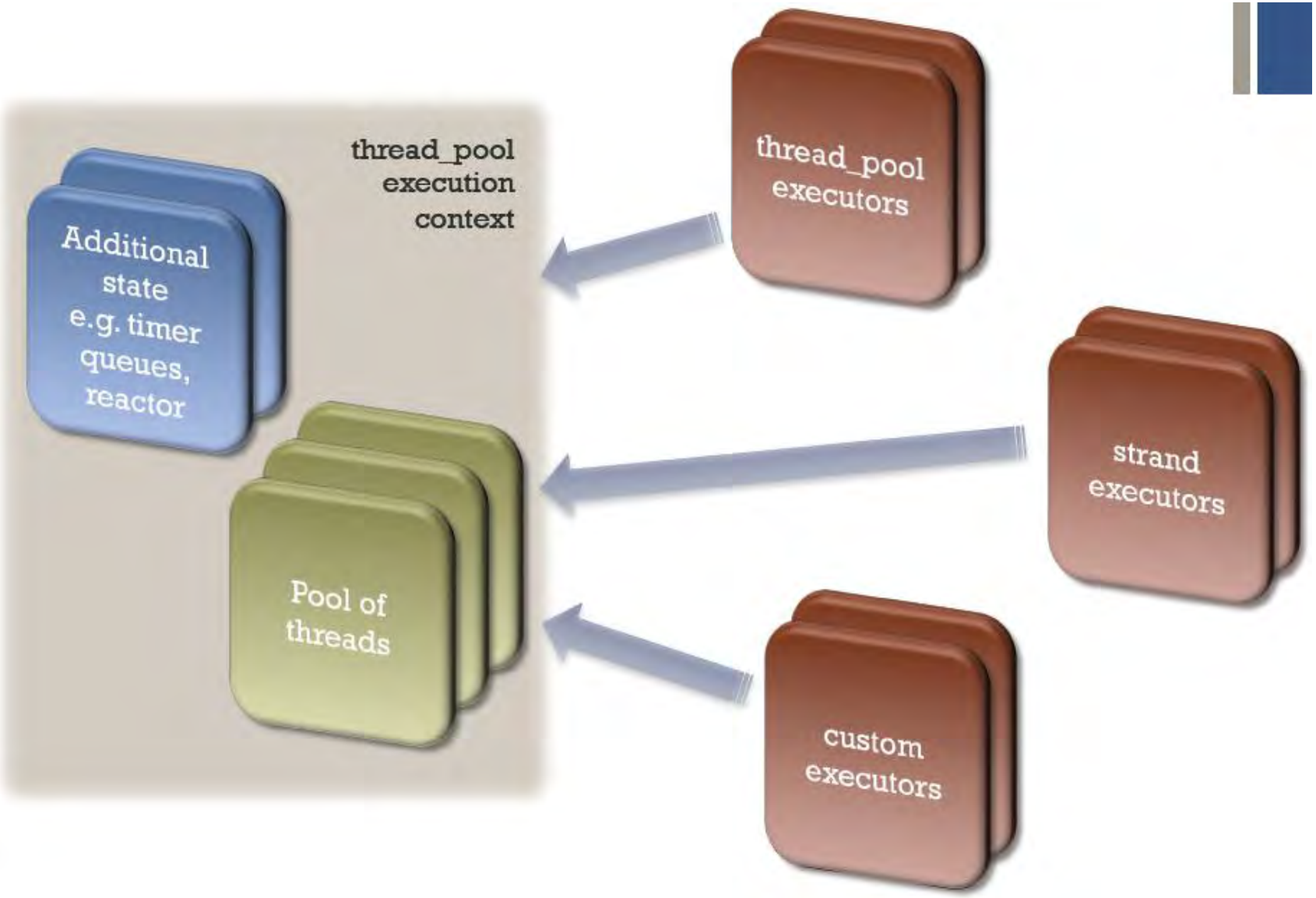
Example: a thread pool

- A thread pool *is an execution context.*
- A thread pool *has an executor.*
- A thread pool's executor embodies this rule:
  - Run function objects *in the pool and nowhere else.*

Example: a strand

- A strand *is an executor.*
- A strand is an adapter for an underlying executor.
- A strand embodies this rule:
  - Run function objects according to the underlying executor's rules, but also run them *in FIFO order and not concurrently.*

Execution contexts and executors



Execution contexts and executors

Execution Contexts

- Usually long lived.
- Non-copyable.
- May contain additional state.
  - Timer queues.
  - Socket reactors.
  - Hidden threads to emulate asynchronous functionality

Executors

- May be long or short lived.
- Lightweight and copyable.
- May be customized on a finegrained basis.
- Example: an executor to capture exceptions generated by an asynchronous operation into an exception_ptr.

Dispatch, post and defer

- The three fundamental operations for submitting function objects for execution.
- They differ in the level of eagerness to execute a function.
- May be used to submit function objects to an executor or an execution context.

Dispatch

- Run the function object immediately if the rules allow it.
- Otherwise, submit for later execution.
- Example: a thread pool
  - Rule: run function objects *in the pool and nowhere else.*
  - If we are on a thread in the pool, run the function object immediately.
  - If we are *not on a thread in the pool, queue the function object for* later and wake up a thread to process it.

Post

- Submit the function for later execution.
- Never run the function object immediately.
- Example: a thread pool
  - Whether or not we are on a thread in the pool, queue the function object for later and wake up a thread to process it.

Defer

- Submit the function for later execution.
- Never run the function immediately.
- Implies a continuation relationship between caller and function object.
- Example: a thread pool
  - If we are *not on a thread in the pool, queue the function object for* later and wake up a thread to process it.
  - If we are on a thread in the pool, queue the function object for later, but don't wake up a thread to process it until control returns to the pool.

Use cases

1. Replacing std::async
2. active objects
3. parallelism in application data flow
4. asynchronous operations

Agenda

- A recap, C++17, the final report card. Is it great or just OK?
- C++20 and the future of C++
- Networking
- Concepts
- … more

Agenda

1. Definitions
2. Diagnostics
3. Generic programming with Concepts
4. Generic programming with the C++17
5. Conclusion

Definitions

- Modern C++
- Concepts TS
- Ranges TS
- Novice
- Average
- Expert

Constraints

```
// pre-conditions:
//   ++i must be possible
//   decltype(++i) is I&
template <typename I>
I successor(I i, int n)
{
   while (--n > 0)
      ++i;
   return i;
}
```

Constraints

```
// pre-conditions:
//   ++i must be possible
template <typename I>
auto successor(I i, int n) -> std::enable_if_t<
   std::is_same_v<decltype(++i), I&>, I>
{
   while (--n > 0)
      ++i;
   return i;
}
```

Constraints

```
template <typename I>
requires requires(I i) {
   {++i} -> I&;
}
I successor(I i, int n)
{
   while (--n > 0)
      ++i;
   return i;
}
```

Concepts

Concepts

```
template <typename T>
concept bool Equality_comparable = requires(T t) {
   {t == t} -> bool;
   {t != t} -> bool;
}

template <typename T>
concept bool Regular = std::is_destructible_v<T> && std::is_default_constructible_v<T> &&
   std::is_move_constructible_v<T> && std::is_move_assignable_v<T> &&
   std::is_copy_constructible_v<T> && std::is_copy_assignable_v<T> &&
   Equality_comparable<T>;

template <Regular T>
class Regular_vector : public std::vector<T> {};
```

Diagnostics

Listing 1

```
// without concepts
#include <algorithm>
#include <iterator>
#include <list>

int main()
{
   auto l = std::list{1, 2, 3, 4, 5};
   std::sort(std::begin(l), std::end(l));
}
```

Listing 2

```
// with concepts
#include <experimental/ranges/algorithm>
#include <experimental/ranges/iterator>
#include <list>

int main()
{
   auto l = std::list{1, 2, 3, 4, 5};
   std::experimental::ranges::sort(l);
}
```

Listing 3

```
// without concepts
#include <algorithm>
#include <iterator>
#include <vector>

class Foo {};

int main()
{
   auto v = std::vector<Foo>{};
   std::sort(std::begin(v), std::end(v));
}
```

Listing 4

```
// with concepts
#include <experimental/ranges/algorithm>
#include <experimental/ranges/iterator>
#include <vector>

class Foo {};

int main()
{
   auto v = std::vector<Foo>{};
   std::experimental::ranges::sort(v);
}
```

Generic programming with Concepts

Simple for experts?

- Nope.
- This is good!
- Descriptive concepts aren’t trivially composable.
- `Sortable` aims to mathematically capture what it means for a type to be sortable.
- Intuitively easy to understand, proof not-so-much

Generic programming in C++17

std::enable_if

```
#include <type_traits>

template <class T, std::enable_if_t<std::is_integral_v<T>>* = nullptr>
void foo(T) {}

int main()
{
  foo(42.0);
}
```

```
<source>: In function 'int main()':
7 : <source>:7:11: error: no matching function for call to 'foo(double)'
   foo(42.0);
           ^
4 : <source>:4:6: note: candidate: template<class T, std::enable_if_t<is_integral_v<T> >* <anonymous> > void foo(T)
 void foo (T){}
      ^~~
4 : <source>:4:6: note:   template argument deduction/substitution failed:
3 : <source>:3:63: note: invalid template non-type parameter
 template <class T, std::enable_if_t<std::is_integral_v<T>>* = nullptr>
                                                              ^~~~~~~
Compiler exited with result code 1
```

std::enable_if

```
#include <type_traits>

template <class T, class = void>
struct foo;

template <class T>
struct foo<T, std::enable_if_t<std::is_integral_v<T>>> {
    //impl
};

int main() {
  foo<double> a;
}
```

```
<source>: In function 'int main()':
12 : <source>:12:15: error: aggregate 'foo<double> a' has incomplete type and cannot be defined
   foo<double> a;
               ^
Compiler exited with result code 1
```