



C++17: the final report card. Is it Great or just OK?

On the road to heterogeneous Computing: Executors

Michael Wong (Codeplay Software, VP of Research and Development)

Chair of SYCL C++ Heterogeneous Programming Language

ISO C++ 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 SG5 Transactional Memory

Chair of WG21 SG14 Games Dev/Low Latency/Financial Trading/Embedded

Editor: C++ SG5 Transactional Memory Technical Specification

Editor: C++ SG1 Concurrency Technical Specification

<http://wongmichael.com/about>

Acknowledgement and Disclaimer

- Numerous people internal and external to the original C++/Khronos/OpenCL/SYCL group, in industry and academia, have made contributions, influenced ideas, written part of this presentations, and offered feedbacks to form part of this talk.
- 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, IBM, Khronos, OpenMP, or ISOCPP.org.
- Other company, product, and service names may be trademarks or service marks of others.
- This is part of an ongoing keynote presentation for upcoming C/C++ Standard status modified as C and C++ changes. Please attribute credit accordingly.

Thanks to the sponsors!

Bloomberg



think-cell 

 abacogroup.eu

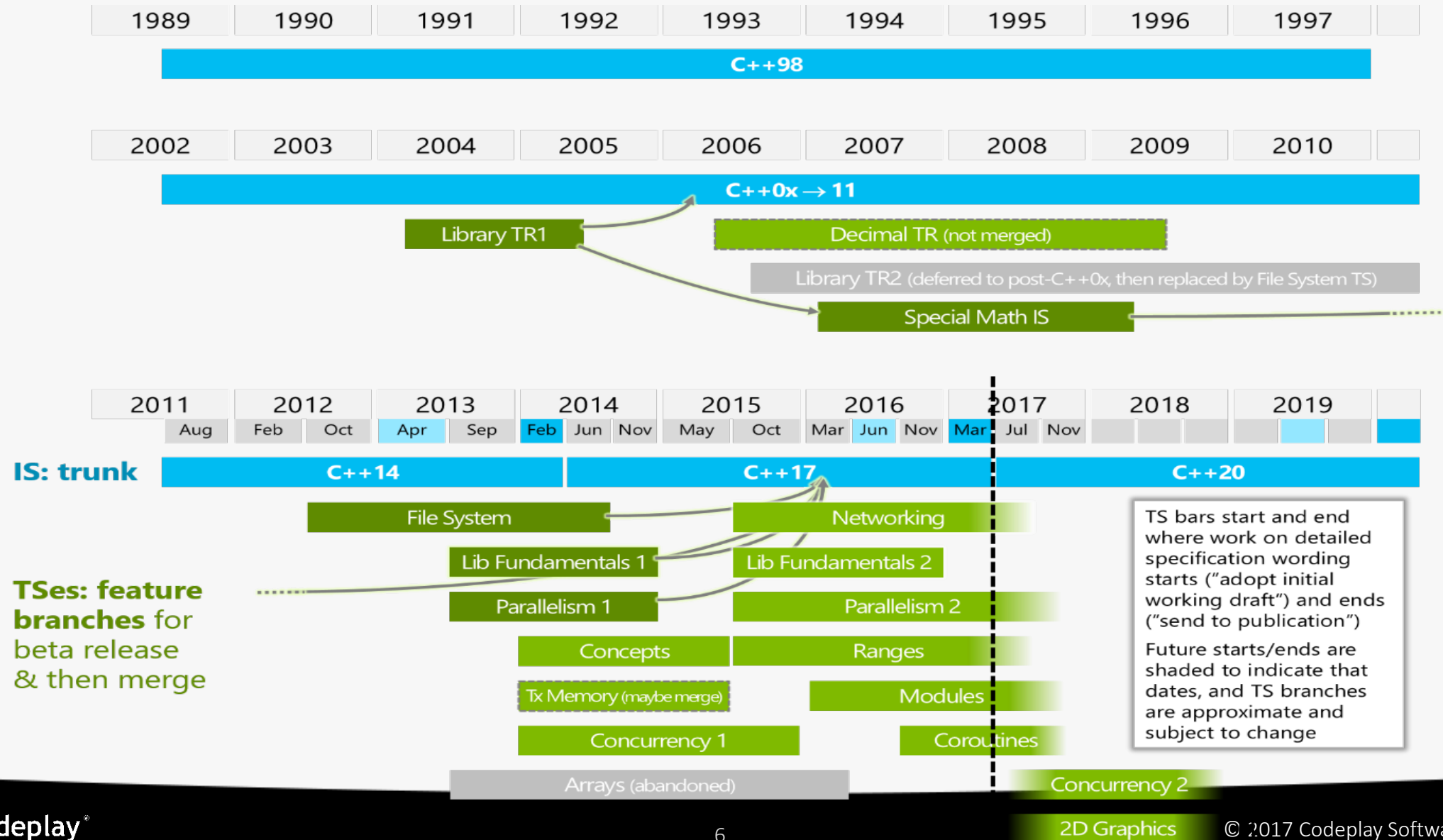


Agenda

- C++17, the final report card. Is it great or just OK?
- The road to Heterogeneous computing: Executors in ISO C++

C++ Std Timeline/status

<https://wongmichael.com/2016/12/09/the-view-from-nov-2016-c-standard-meeting-issaquah/>



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	Published 2011-10-25 (ISO Store) Draft: n2849 (2009-03-06) May be superseded by a future Decimal TS or n3871	Decimal Floating Point decimal32 decimal64	No. Ongoing work in SG6

Status after March Kona 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	No. Already in GCC 6 release and and waiting for subsequent usage experience

Status after March Kona 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	No. 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	In development, Draft n4647 (2017-03-19)	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 March Kona 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 March Kona 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 Heterogeneous 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

C++ 17 Language features already voted in

- [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 voted in Oulu Finland

[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](#)

- [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 already voted in

- [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 voted in Oulu Finland

- [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](#)

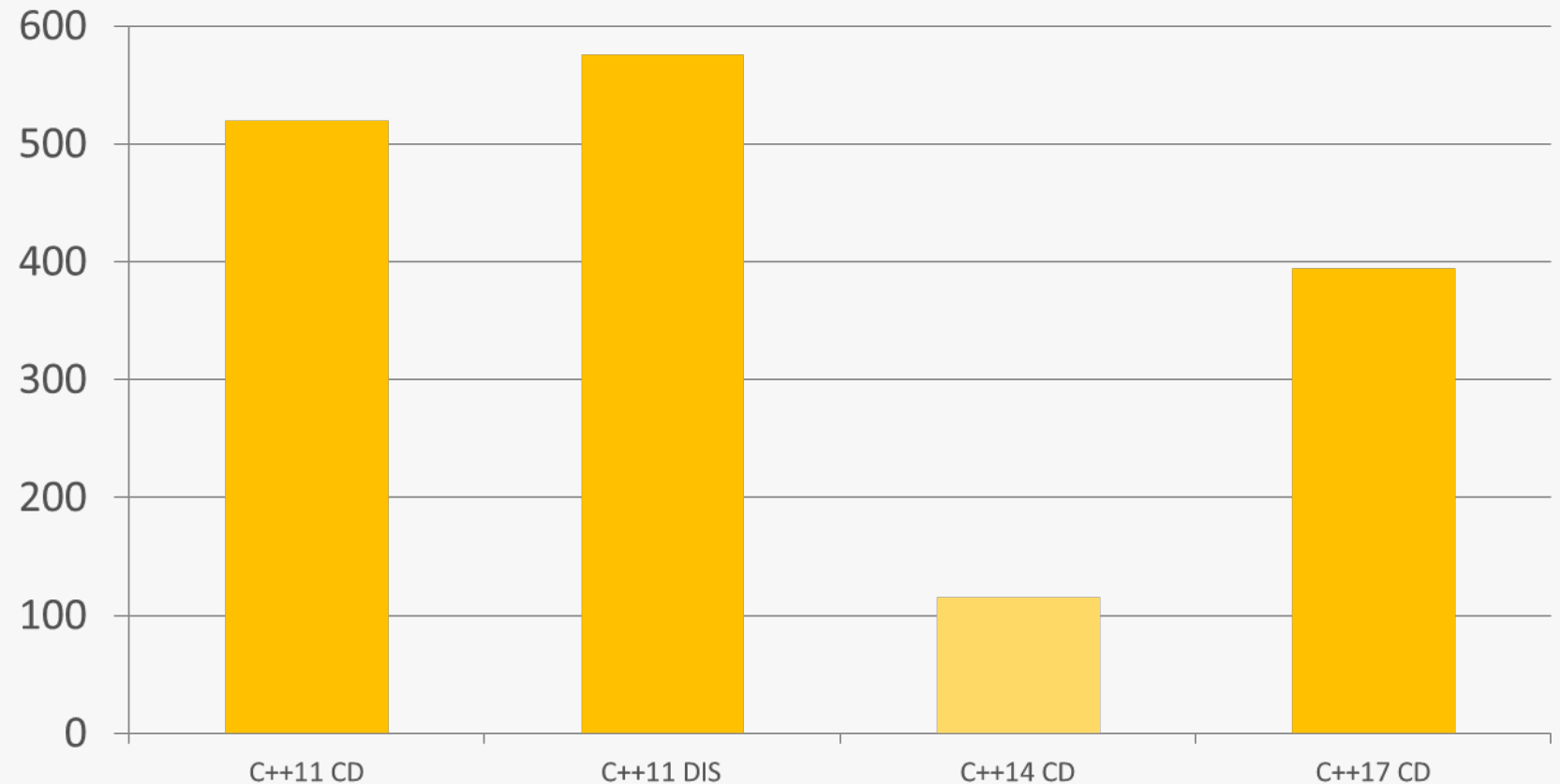
By the number of pages

- 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

Comments to address in ballot resolution

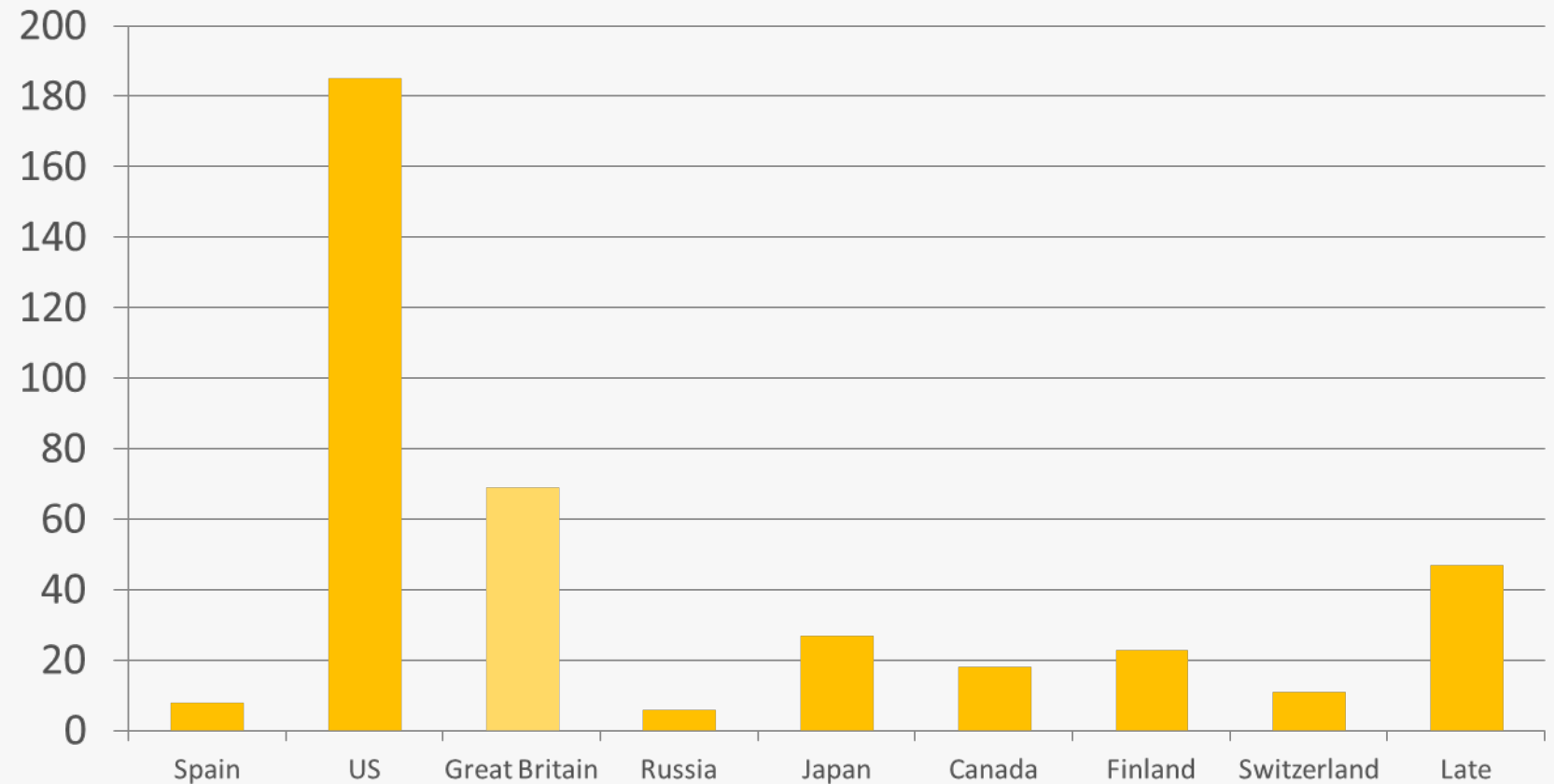
- Each round of international comment ballots generates bugs, tweaks, and requests



C++ 17: by Country

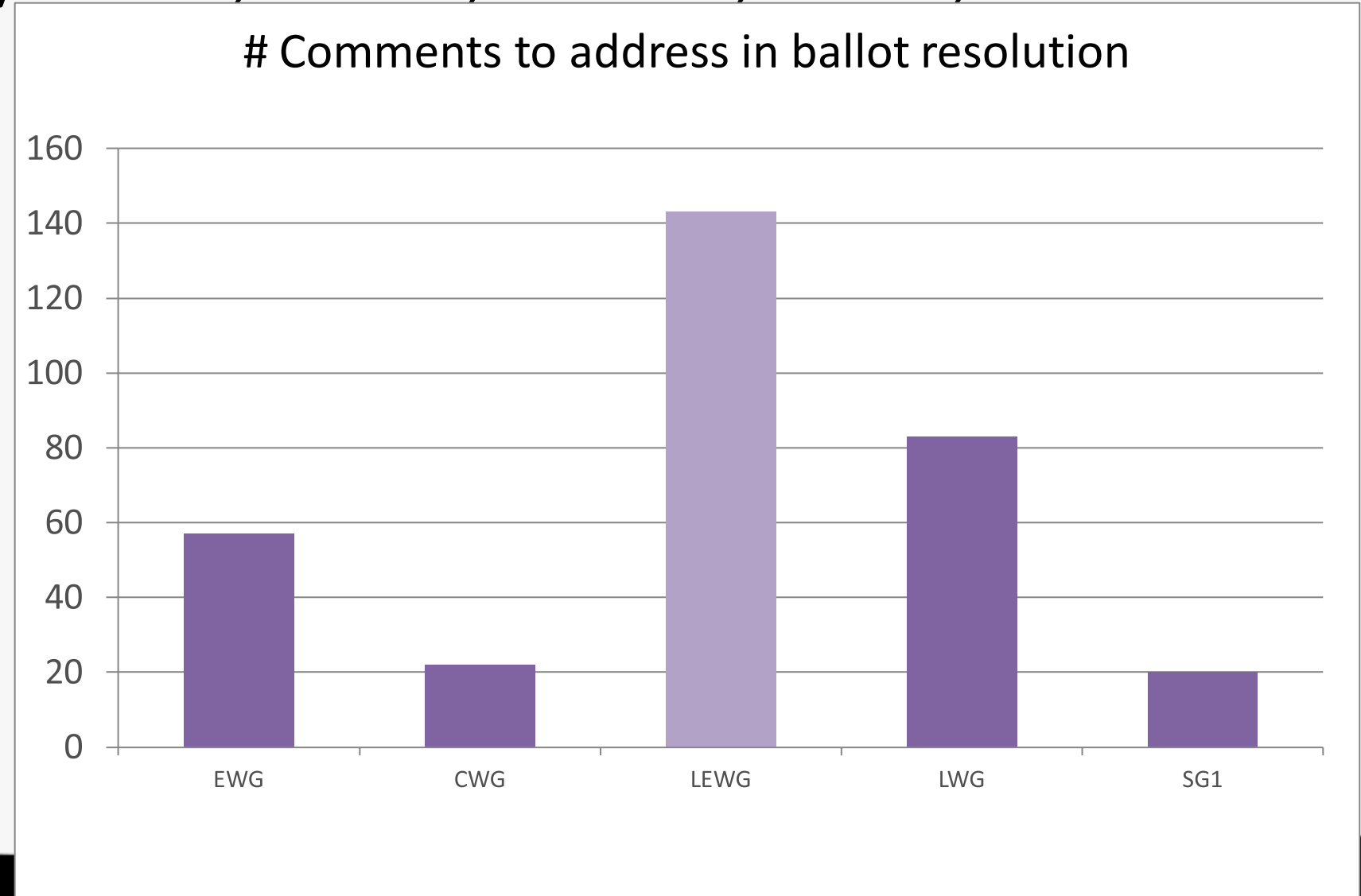
Comments to address in ballot resolution

- 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



What did not change from Issaquah

No Concepts

- Inline variable stays

No Unified Call Syntax

No Default Comparison

No operator dot

Changes voted in Issaquah+Kona

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>

C++20 new features

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.](#)


Future C++ Standard schedules

- 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
- 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

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](#)

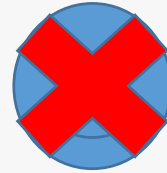


- Parallel algorithms (incl. parallel versions of some of the STL

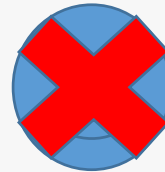
- [n4409](#)

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
- Bus Train Model, sometimes you miss the bus
- 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

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

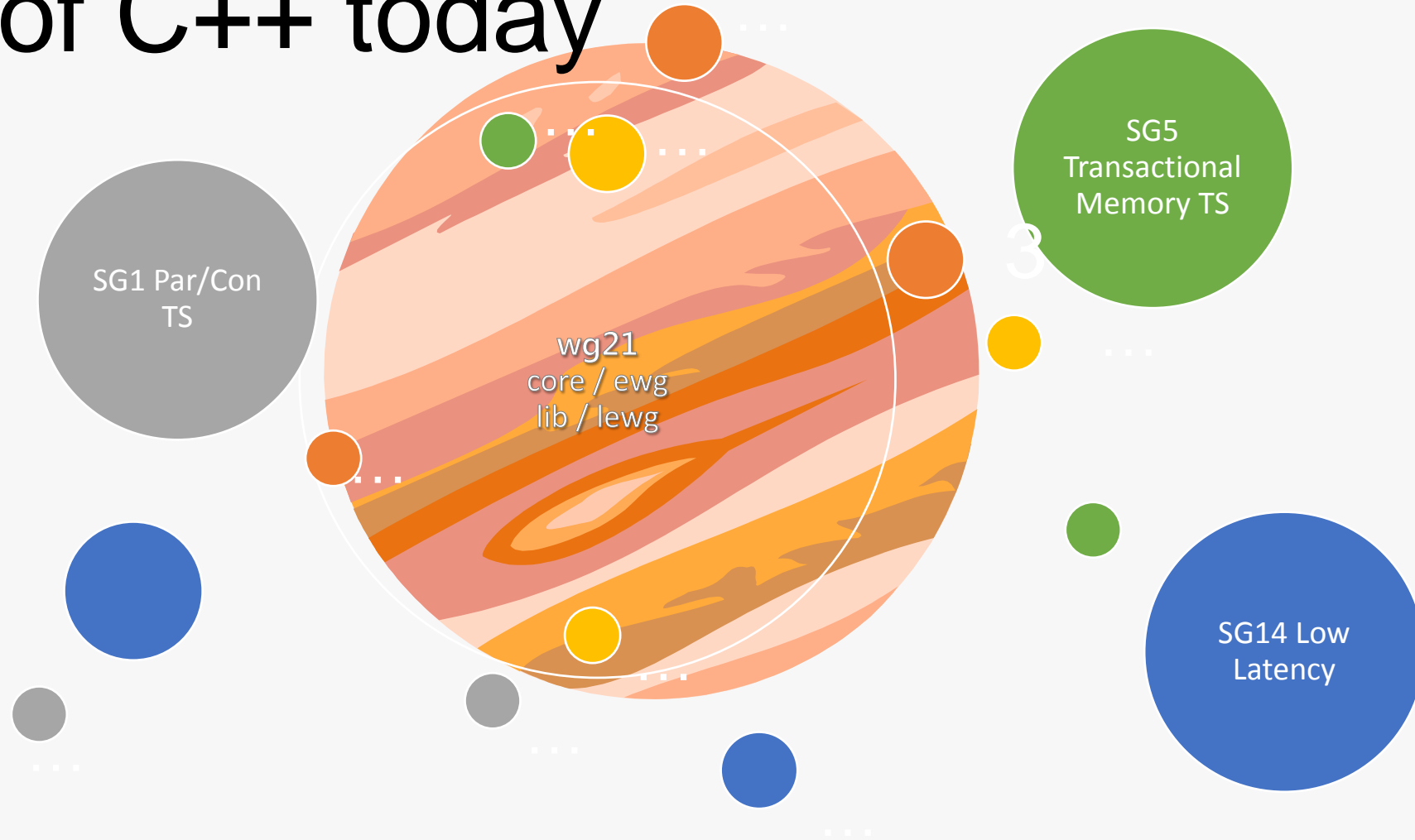
- You blew it
- Not a Major release
- No risk, no gain
- Nobody implement TSs
- Tethering tower of Babel of TSs
- Bus Train Model, sometimes you miss the bus
- Did a nice job
- But not Minor either
- Safe and conservative wins
- TSs are implemented
- Followed the rules of a bus train model, how to get 110 people to work together

A Medium/OK
Release

Agenda

- C++17, the final report card. Is it great or just OK?
- The road to Heterogeneous computing: Executors in ISO C++

The Parallel and concurrency planets of C++ today



C++1Y(1Y=17/20/22) SG1/SG5/SG14 Plan

red=C++17, blue=C++20? Black=future?

Parallelism

- **Parallel Algorithms:**
- **Data-Based Parallelism.**
(Vector, SIMD, ...)
- **Task-based parallelism**
(cilk, OpenMP, fork-join)
- **Execution Agents**
- **Progress guarantees**
- **MapReduce**
- **Pipelines**

Concurrency

- **Future++ (then, wait_any, wait_all):**
- **Executors:**
- **Resumable Functions, await (with futures)**
- **Lock free techniques/Transactions**
- **Synchronics**
- **Atomic Views**
- **Co-routines**
- **Counters/Queues**
- **Concurrent Vector/Unordered Associative Containers**
- **Latches and Barriers**
- **upgrade_lock**
- **Atomic smart pointers**

Executors

- Executors are to function execution what allocators are to memory allocation
- If a control structure such as `std::async()` or the parallel algorithms describe work that is to be executed
- An executor describes where and when that work is to be executed
- <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0443r0.html>

The Idea Behind Executors

Diverse
Control
Structures

```
async(...)      for_each(...)
define_task_block(...)  defer(...)
your_favorite_control_structure(...)
```

Unified Interface for Execution

Diverse
Execution
Resources

Operating
System Threads

SIMD vector
units

Thread pool
schedulers

GPU
runtime

OpenMP
runtime

Fibers

Several Competing Proposals

- P0008r0 (Mysen): Minimal interface for fire-and-forget execution
- P0058r1 (Hoberock *et al.*): *Functionality needed for foundations of Parallelism TS*
- P0113r0 (Kohlhoff): Functionality needed for foundations of Networking TS
- P0285r0 (Kohlhoff): Executor categories & customization points

What Are Executors? (Gordon Brown)

invoke	async	parallel algorithms	future::then	post
defer	define_task_block	dispatch	asynchronous operations	strand<>

Unified interface for execution

OpenCL

OpenMP

CUDA

C++ Threads



SAXPY – Computation (Gordon Brown)

```
{  
  ...  
  
  #pragma omp parallel for  
  for (int i = 0; i < SIZE; i++) {  
    y[i] = a * x[i] + y[i];  
  }  
  
  ...  
}
```

OpenMP

```
{  
  ...  
  
  cgh.parallel_for(range<1>(SIZE), [=](id<1>  
idx){  
    y[i] = a * x[i] + y[i];  
  });  
});  
  
  ...  
}
```

SYCL

```
{  
  ...  
  
  for (int i = 0; i < SIZE; i++) {  
    std::thread([=]() {  
      y[i] = a * x[i] + y[i];  
    });  
  }  
  
  ...  
}
```

C++11 Threads

```
__global__ void saxpy(float a, float * restrict x,  
                      float * restrict y) {  
  int i = blockIdx.x * blockDim.x + threadIdx.x;  
  y[i] = a * x[i] + y[i];  
}  
  
{  
  ...  
  
  saxpy<<< SIZE >>>(a, x, y);  
  
  ...  
}
```

CUDA

SAXPY – Computation (Gordon Brown)

```
{  
  ...  
  
  executor exec;  
  exec.bulk_execute(shape<1>(SIZE), [=](index<1> i) {  
    y[i] = a * x[i] + y[i];  
  }));  
  
  ...  
}
```

Executors

Purpose 1 of executors:where/how execution

- Placement is, by default, at discretion of the system.

```
for_each(par, I.begin(), I.end(), [](int i) { y[i] += a*x[i]; });
```

- If the Programmer want to control placement:

```
auto exec1 = choose_some_executor();  
auto exec2 = choose_another_executor();  
  
for_each(par.on(exec1), I.begin(), I.end(), ...);  
for_each(par.on(exec2), I.begin(), I.end(), ...);
```

Purpose 2 of executors

- Control relationship with Calling threads
 - `async(launch_flags, function);`
 - `async(executor, function);`

Purpose 3 of executors

- Uniform interface for scheduling semantics across control structures
 - `for_each(P.on(executor), ...);`
 - `async(executor, ...);`
 - `future.then(executor, ...);`
 - `dispatch(executor, ...);`

SHORT TERM GOALS

- Compose with existing control structures
 - In C++17:
 - `async()`, `invoke()`, `for_each()`, `sort()`, ...
 - In technical specifications:
 - `define_task_block()`, `future.then()`, Networking TS, asynchronous operations, Transactional memory

POSSIBLE EXTENSIONS

Out of scope of minimal proposal

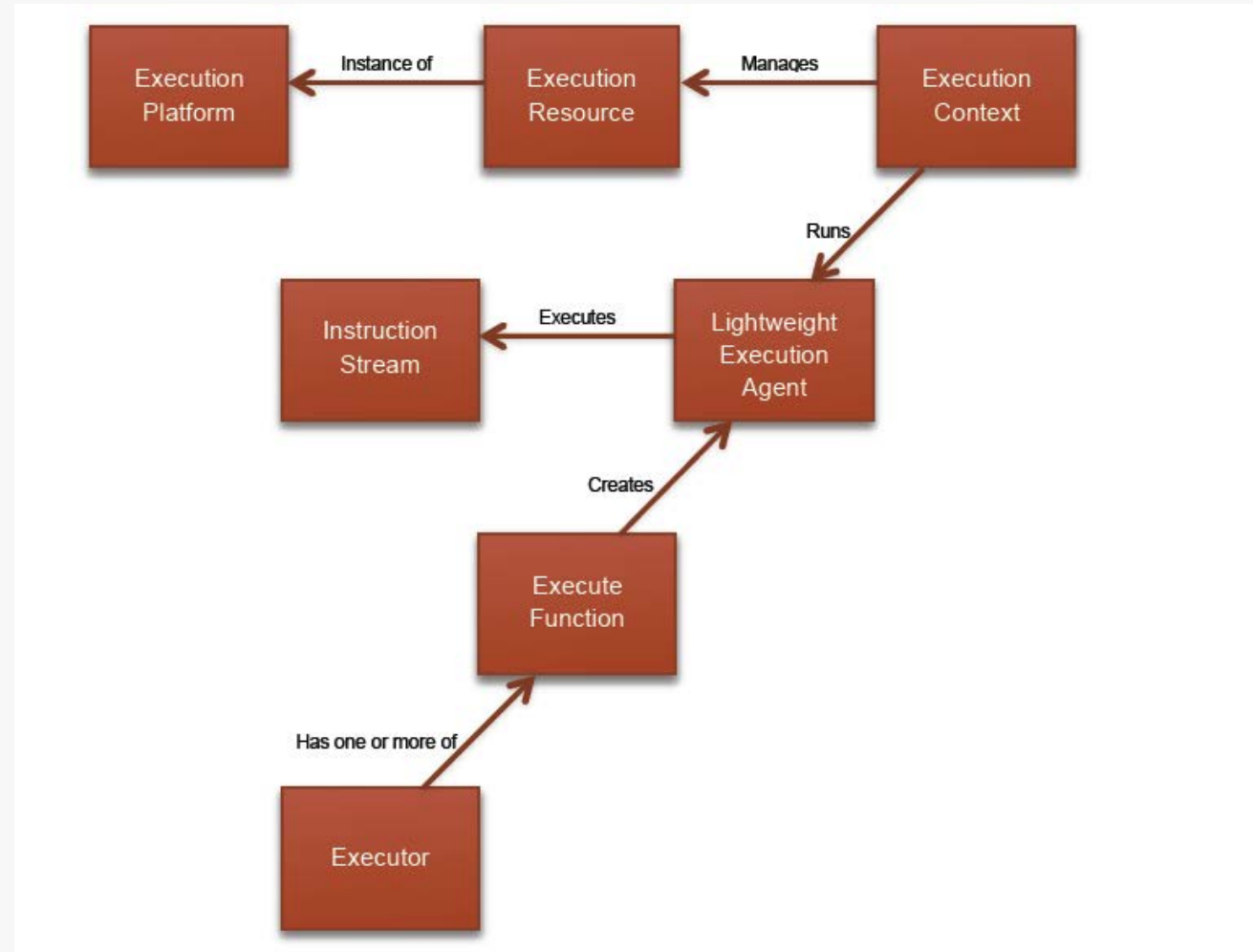
- Error handling
- Requirements on user-defined Future types
- Heterogeneity
- Distributed memory
- Additional abstractions for bulk execution
- Higher-level variadic abstractions
- Remote execution
- Additional thread pool functionality
- System resources
- Syntactic sugar for contexts + control structures

UNIFIED DESIGN

- Distinguish **executors** from **execution contexts**
- **Categorize** executors
- Enable **customization**
- Describe composition with existing control structures

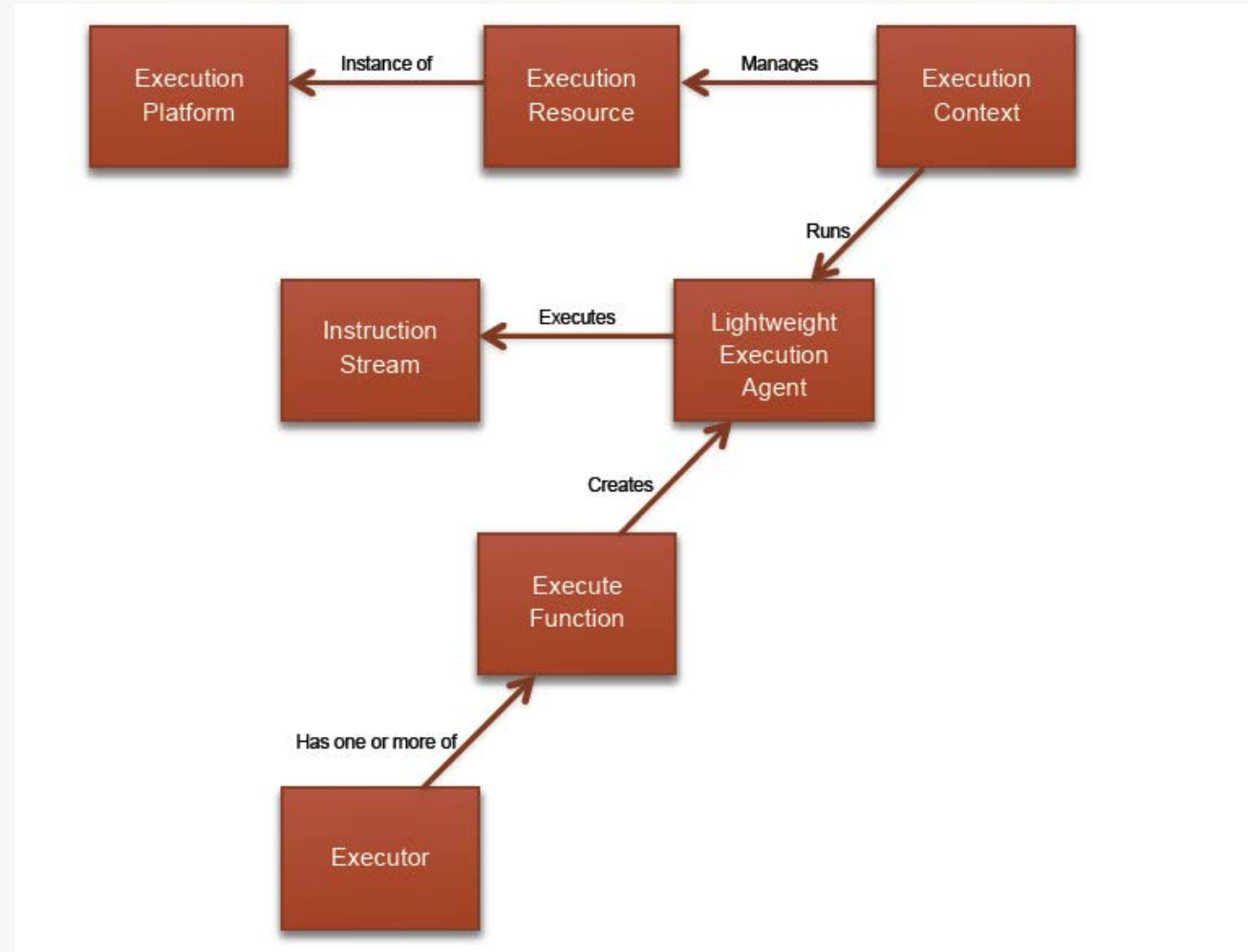
Current Progress of Executors

- Closing in on minimal proposal
- A foundation for later proposals (for heterogeneous computing)
- Still work in progress



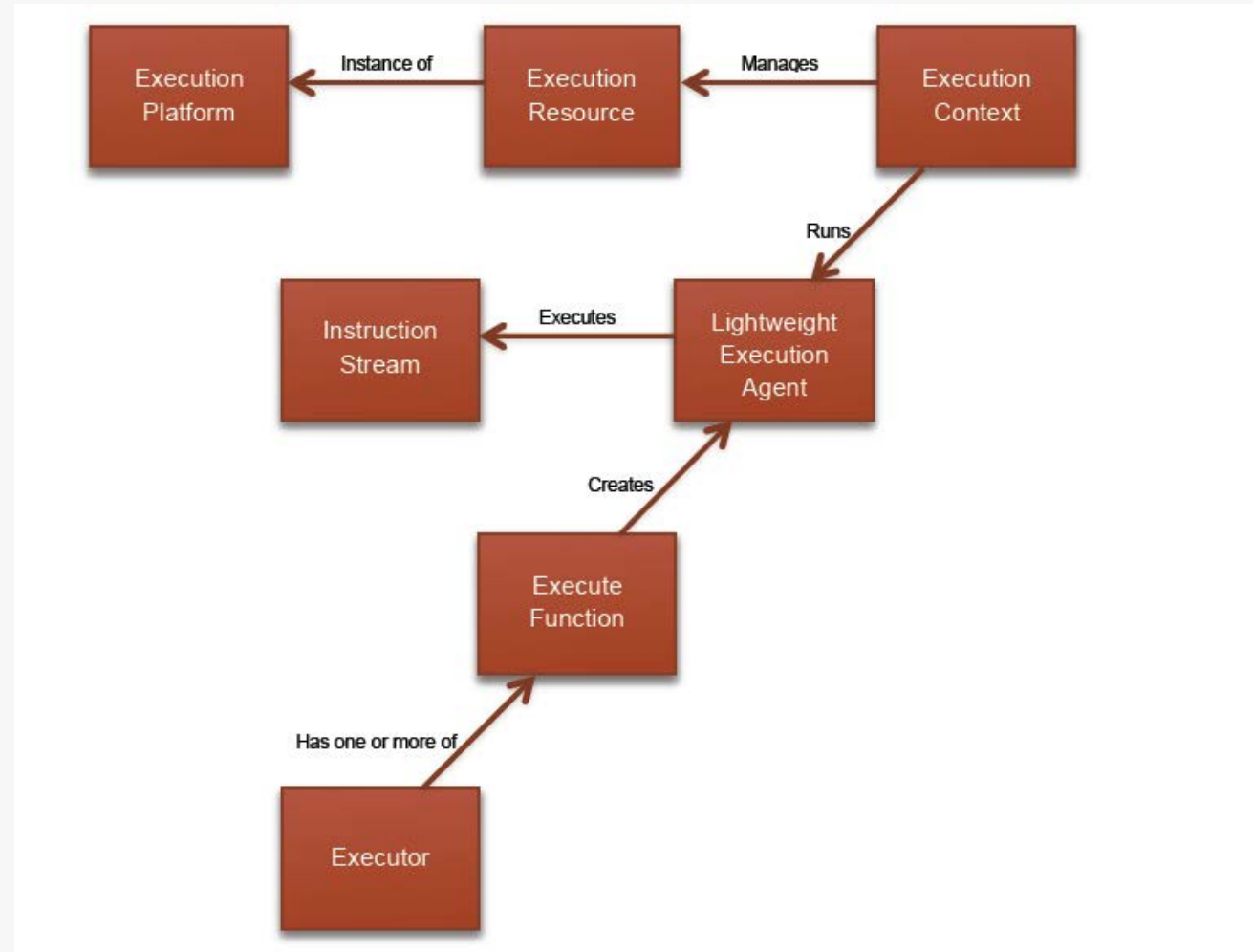
Current Progress of Executors

- 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*; decouple control structures from resources
- 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 and/or software abstraction that is executing the work such as a thread pool
- An *execution context* **are** objects representing a handle to specific collections of resources; manages the light weight *execution agents* of an *execution resource* during the execution



EXECUTION FUNCTIONS

Fundamental functions for creating execution agents

- Name encodes characteristics

- Blocking
- Directionality
- Cardinality

	Cardinality	Directionality	Blocking
Execute	Single	One-way	Possibly
async_defer	single	Two-way	No
bulk_sync_execute	bulk	Two-way	Yes
...			

Name	Cardinality	Directionality	Blocking
execute	single	one-way	possibly
post	single	one-way	no
defer	single	one-way	no
async_execute	single	two-way	possibly
then_execute	single	two-way	possibly
sync_execute	single	two-way	yes
async_post	single	two-way	no
async_defer	single	two-way	no
bulk_execute	bulk	one-way	possibly
bulk_post	bulk	one-way	no
bulk_defer	bulk	one-way	no
bulk_async_execute	bulk	two-way	possibly
bulk_then_execute	bulk	two-way	possibly
bulk_sync_execute	bulk	two-way	yes
bulk_async_post	bulk	two-way	no
bulk_async_defer	bulk	two-way	no

Possible simplification

- oneway_execute
- twoway_execute
- bulk_oneway_execute
- Bulk_twoway_execute

Execute, 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.

Execute

- 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. Wait till I am done**
- 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.**

EXECUTION FUNCTIONS

Name encodes characteristics

- Users see what is natively supported by an executor
 - Look at member/free functions
- Users see how a call to an executor behaves
 - Look at which function was called
- Executor Authors simultaneously implement and guarantee behavior
 - Select the appropriate function and implement it

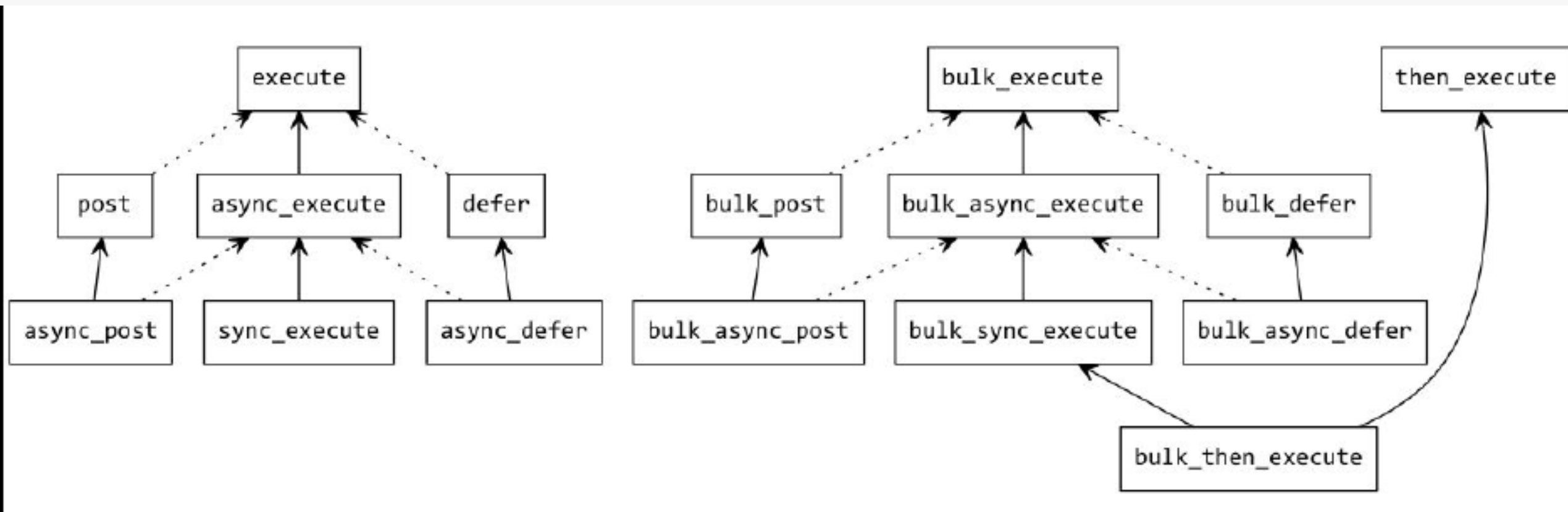
CUSTOMIZATION POINTS

For fundamental executor interactions

- Provides a taxonomy of behaviors through unique functions which provides concrete execution guarantees
- Enable uniform use of executors by generic code without requiring a full interface implementation by any given executor implementer
- Adapt native behavior to desired behavior as needed
- Still allows code to implement directly to the executor interface (for performance or efficiency reasons)

CUSTOMIZATION POINTS

Visualizing adaptations



CROSS-CUTTING CONCERNS

Introspecting guarantees

- Invariants preserved by adaptation
- Exposed via type traits
- Enables compile-time decisions

Type Traits

- blocking behavior
- execution agent mapping
- execution function detection
- bulk execution semantics
- associated types
 - Context
 - Future
 - Index
 - shape

CATEGORIES

	Oneway	NonBlockingOneway	Twoway	BulkTwoway
can_execute				
can_post				
can_defer				
can_async_post				
can_async_defer				
can_sync_execute				
can_async_execute				
can_then_execute				
can_bulk_execute				
can_bulk_sync_execute				
can_bulk_async_execute				
can_bulk_then_execute				
can_bulk_post				
can_bulk_defer				
can_bulk_async_post				
can_bulk_async_defer				

Executor Framework: Abstract Platform details of execution.

Create execution agents

Manage data they share

Advertise semantics

Mediate dependencies

```
class sample_executor
{
public:
    using execution_category = ...;
    using shape_type = tuple<size_t,size_t>;
    template<class T> using future = ...;
    template<class T> future<T>
        make_ready_future(T&& value);
    template<class Function, class Factory1,
              class Factory2> future<...>
        bulk_async_execute(Function f,
                           shape_type shape, Factory1 result_factory,
                           Factory2 shared_factory);...
}
```

EXECUTORS & CONTEXTS

Light-weight views on long-lived resources

- Distinguish **executors** from **execution contexts**
- **Categorize** executors
- Enable **customization**
- Describe composition with existing control structures

EXECUTORS & CONTEXTS

Light-weight views on long-lived resources

- **Executors** are (potentially **short-lived**) **objects** that **create** execution agents on execution contexts.
- **Execution contexts** are (potentially **long-lived**) **objects** that manage the lifetime of underlying execution resources.

EXECUTORS & CONTEXTS

Example: simple thread pool

```
• struct my_thread_pool
• {
•     template<class Function>
•     void submit(Function&& f);
•     struct executor_t
•     {
•         my_thread_pool& ctx;
•         template<class Function>
•         void execute(Function&& f) const
•         {
•             // forward the function to the thread pool
•             ctx.submit(std::forward<Function>(f));
•         }
•         my_thread_pool& context() const noexcept {return ctx;}
•         bool operator==(const executor_t& rhs) const noexcept {return ctx == rhs.ctx;}
•         bool operator!=(const executor_t& rhs) const noexcept {return ctx != rhs.ctx;}
•     };
•     executor_t executor() { return executor_t{*this}; }
•     ...
• };
```

- Context: `my_thread_pool`
- Executor: `my_thread_pool::executor_t`
- `.execute()` submits a task to the thread pool
- The executor is created by the context

Executor Interface: semantic types exposed by executors

Type	Meaning
execution_category	Scheduling semantics amongst agents in a task. (sequenced, vector-parallel, parallel, concurrent)
shape_type	Type for indexing bulk launch of agents. (typically n-dimensional integer indices)
future<T>	Type for synchronizing asynchronous activities. (follows interface of std::future)

EXECUTORS & THE STANDARD LIBRARY

Composition with control structures

- `my_executor exec = get_my_executor(...);`
- `using namespace std;`
- `auto fut1 = async(exec, task);`
- `sort(execution::par.on(exec), vec.begin(), vec.end());`
- `auto fut2 = fut1.then(exec, continuation);`
- Most programmers use higher-level control structures
- Need to compose with user-defined executors

EXECUTORS & THE STANDARD LIBRARY

Possible implementation of `std::for_each`

```
template<class Policy, class Iterator>
using __enable_if_bulk_sync_executable<Policy,Iterator> =
    enable_if_t<
        is_same_v<
            typename Policy::execution_category,
            parallel_execution_tag
        > &&
        is_convertible_v<
            typename iterator_traits<Iterator>::iterator_category,
            random_access_iterator_tag
        >
    >;
```


Summary Executors

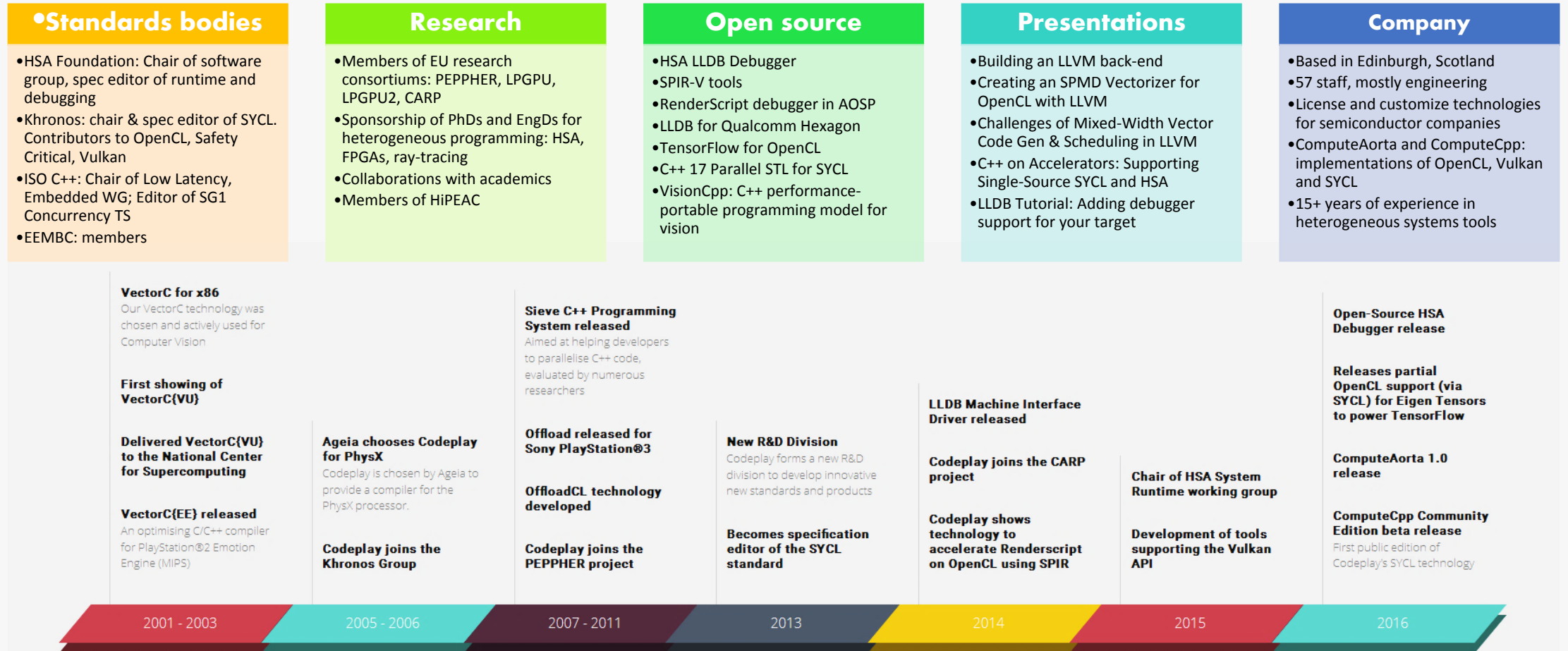
Executors **decouple** control structures from work creation

Short-term goal: **compose with existing** control structures

P0443 is the **minimal** proposal to achieve short-term goal

Provides a foundation for **extensions** to build on

Codeplay



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



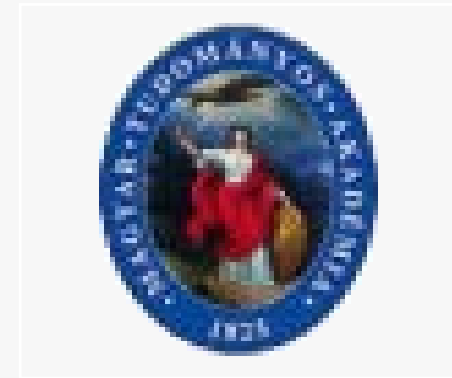
"We work with royalty-free SYCL because it is hardware vendor agnostic, single-source C++ programming model without platform specific keywords. This will allow us to easily work with any heterogeneous processor solutions using OpenCL to develop our complex algorithms and ensure future compatibility"

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:

compute.cpp.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>



Questions ?



@codeplaysoft



/codeplaysoft



codeplay.com