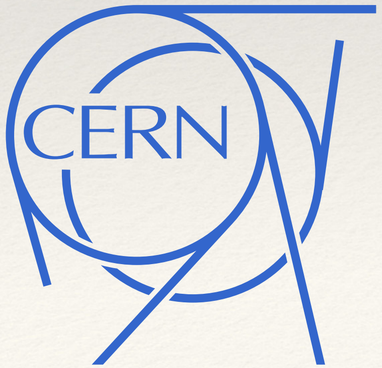
2019-03-25

C++ Meeting Trip Report

Axel Naumann



Background

- C++ evolution defined by ISO committee
- Your CERN representative - but really ours: HENP's!
- Kona was C++20 feature-freeze

- Remember C++98 → C++11?

C++20

C++ 2.0

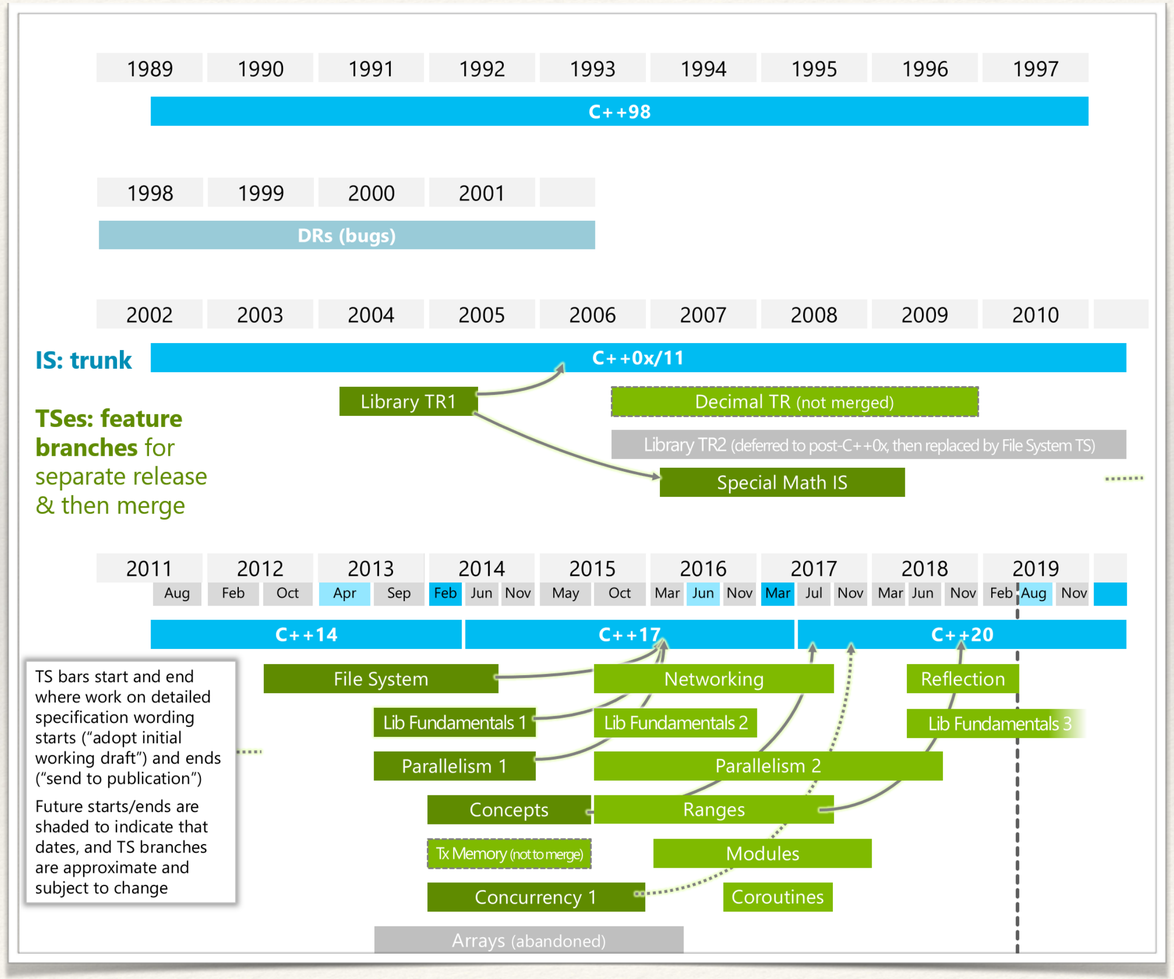
Overview

- Process
- Big features and where they matter
- Little features and where they matter
- What does Axel do?
- The Future
- Conclusion

Process

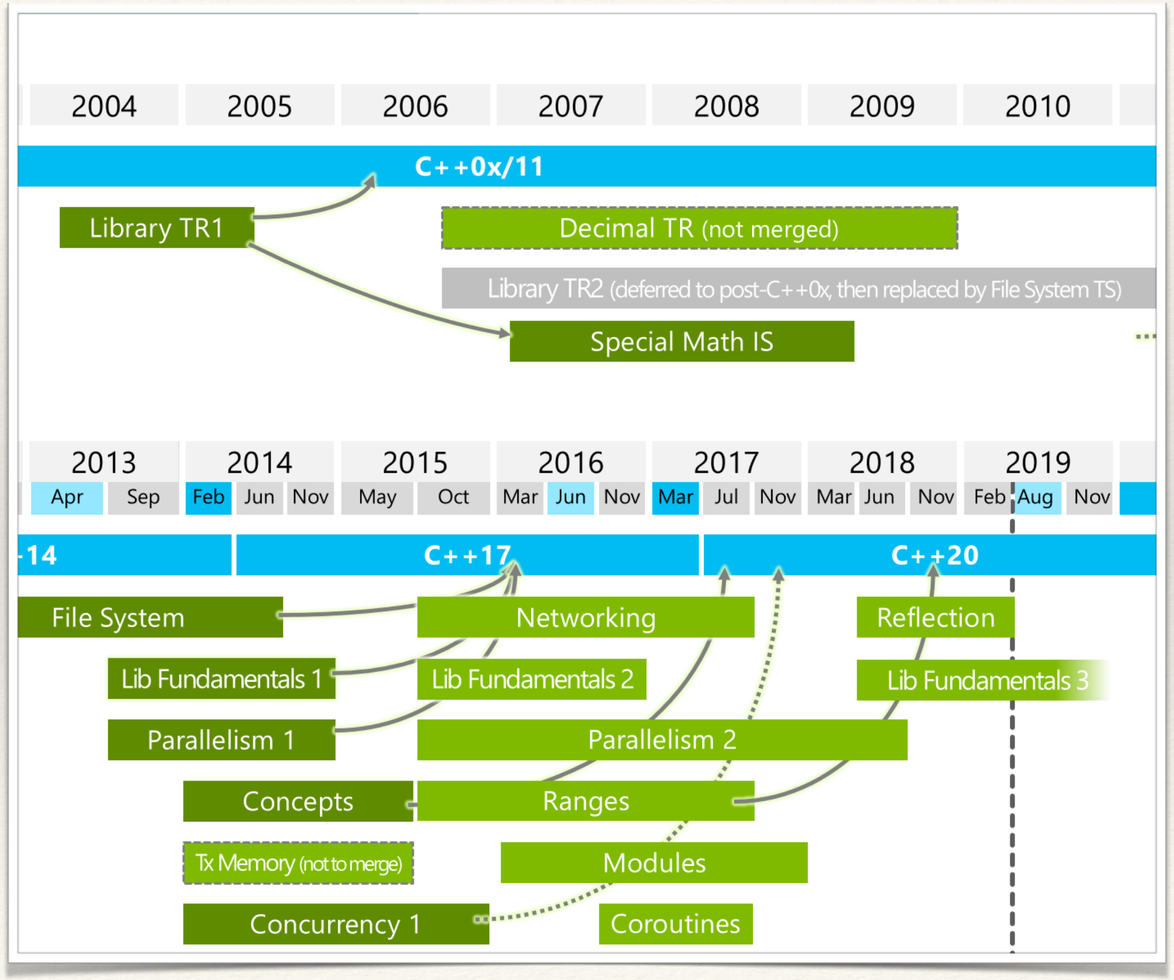
Areas of Work

- Evolution in several areas, in parallel. Several just missed C++17. Many major features ended up in C++20.
- Features are proposals or Technical Specifications (TS)



C++ Technical Specifications

- Major work items are sometimes progressing outside the standard:
  - TS allows to test-drive
  - gain implementation + usage experience before entering the standard



C++ 20 and TSes

- Merging
  - Modules
  - Co-routines
  - Concepts
  - Ranges
- Many almost ready for C++17 - and C++20 became the TS treasure chest!
- … and there is still more!

Study Groups

- Focus on topics before sending recommended proposals to rest of committee
- New-ish ones: GUI, Low Latency, Tooling, Unicode, Machine Learning (incl linear algebra), Education

Small Features

Selection of small features in C++17, 20

- span
- format
- optional + variant (C++17)
- structured binding (C++17)
- if constexpr (C++17)
- if with variable declaration (C++17)
- (string) literals as template parameters
- more constexpr, consteval
- class template argument deduction

Too much!

- Selected most relevant ones
- For the rest: check CppCon + MeetingCpp + CppNow

Selection of small features in C++17, 20

✓ = covered here

- ✓ span
- ✓ format
- ✓ optional + variant (C++17)
- ✓ structured binding (C++17)
- ✓ if constexpr (C++17)
- ✓ if with variable declaration (C++17)

- ❖ (string) literals as template parameters
- ❖ more constexpr, consteval
- ❖ class template argument deduction

if with variable declaration (C++17)

```
if (auto v = f(); !v.get())
```

- Fixes scope of if-condition variables, compared to earlier:

```
auto v = f();
if (!v.get()) {
  // but I need v only in here...
}
```

if with variable declaration (C++17)

```
if (auto v = f(); !v.get())
```

- Fixes scope of if-condition variables, compared to earlier:

```
auto v = f();
if (!v.get()) {
  // but I need v only in here...
}
```

if constexpr

```
template <class T>
void *begin_if(T& v) {
  if constexpr (is_container<T>)
    return &*v.begin();
  return nullptr;
}
```

- Does not compile the branch if false
- Valid code for T being bool despite v.begin()

Structured Binding

```
for (auto && [k,v] : myMap) {
    cout << "key: " << k
         << "val: " << v << '\n';
}
```

- Great way of “receiving” multiple (2, 3, 4,…) struct members or tuple<> elements
- Handle multiple values being passed in for or if statements

optional, variant

```
variant<double, string> v{17.};
assert(get<double>(v) > 16);
v.emplace<string>("ABC");
cout << get<1>(v); // good!
get<0>(v); // throws!
```

- std::optional: holds one or none
- std::variant: holds one of a set (union)
- C++20 will possibly also have std::expected: value or error
- Extremely powerful for writing safe, compact code

span

- Whether std::vector, std::array or C-style array
- Refers to a contiguous array of given size
- Wonderful as function parameter
- Fixed-size or runtime-size

```
void func(span<double,4> lv);
array<double, 4> jetLV{...};
func(jetLV);
```

format

```
string message = format("The answer is {}.", 42);
```

- Not yet guaranteed for C++20 - but expected!
- An efficient and nice way to format strings in C++, finally
- printf-format plus so much more
- Incl. user-extensible: format your classes

Big Features

BIG FEATURES

Big C++ features since C++14

✓ = covered here

- ✓ Contracts
- ✓ Concepts
- ✓ Ranges
- ✓ Modules
- ✓ <=>

- ❖ Coroutines
- ❖ std::filesystem (C++17)

Contracts (1/3): Intro

- Specify what your function expects
- Can be checked by compiler
- Check can be turned off
- Also enables optimizations

Contracts (2/3): Setting Expectations

- expects (or “pre”?): condition on arguments
- ensures (or “post”): post-condition, a guarantee by the function
- assert: a check to be performed within the function

```
int f(int i) [[expects: i > 0]];
int g(string& s) [[ensures: !s.empty()]] {
    [[assert: s.empty()]]
    s = "ABC";
}
```

Contracts (3/3): Validating Expectations

- Contract levels **default**, **audit**, **axiom**; compiler flag selects what to check:
  - `--default`: checks **default**
  - `--off`: nothing
  - `--audit`: **default** and **audit**
- **axiom** is thus never checked: good for optimizer hints; expensive checks can be **audit**

```
int f(int i) [[default pre: i > 0]];
int g(int i) [[audit pre: i > 0]];
int h(int i) [[axiom pre: i > 0]];
```

Concepts

- Document and restrict template parameters
- Better error messages
- User-Oriented feature for library authors, i.e. us!

```
template <ConvertibleTo<string> T>
  class DoesSomethingWithAString;
```

Ranges

- Generalized iterators working on everything that has a begin
- Much, *much* nicer syntax
- More than just syntax: a way of writing algorithms without mentioning data!

```
if (auto evens = vec | view::filter(is_even)) {
  // Do something with an even number.
}
```

- Note use of | to pipe into filter; use of range as boolean expression

Modules

- Dramatic build time reduction
- Hides implementation details, similar to header / source
- See “Migrating large codebases to C++ Modules" by ROOT team’s Yuka Takahashi on Wednesday, 19:00, Track 1!

Spaceship <=>

- finally a default comparison!
- reduces code clutter and bugs

```
class A { int i; };
bool operator==(A, A) {...}
bool operator!=(A, A) {...}
bool operator>(A, A) {...}
bool operator<(A, A) {...}
bool operator>=(A, A) {...}
bool operator<=(A, A) {...}
```

Spaceship <=>

- finally a default comparison!
- reduces code clutter and bugs

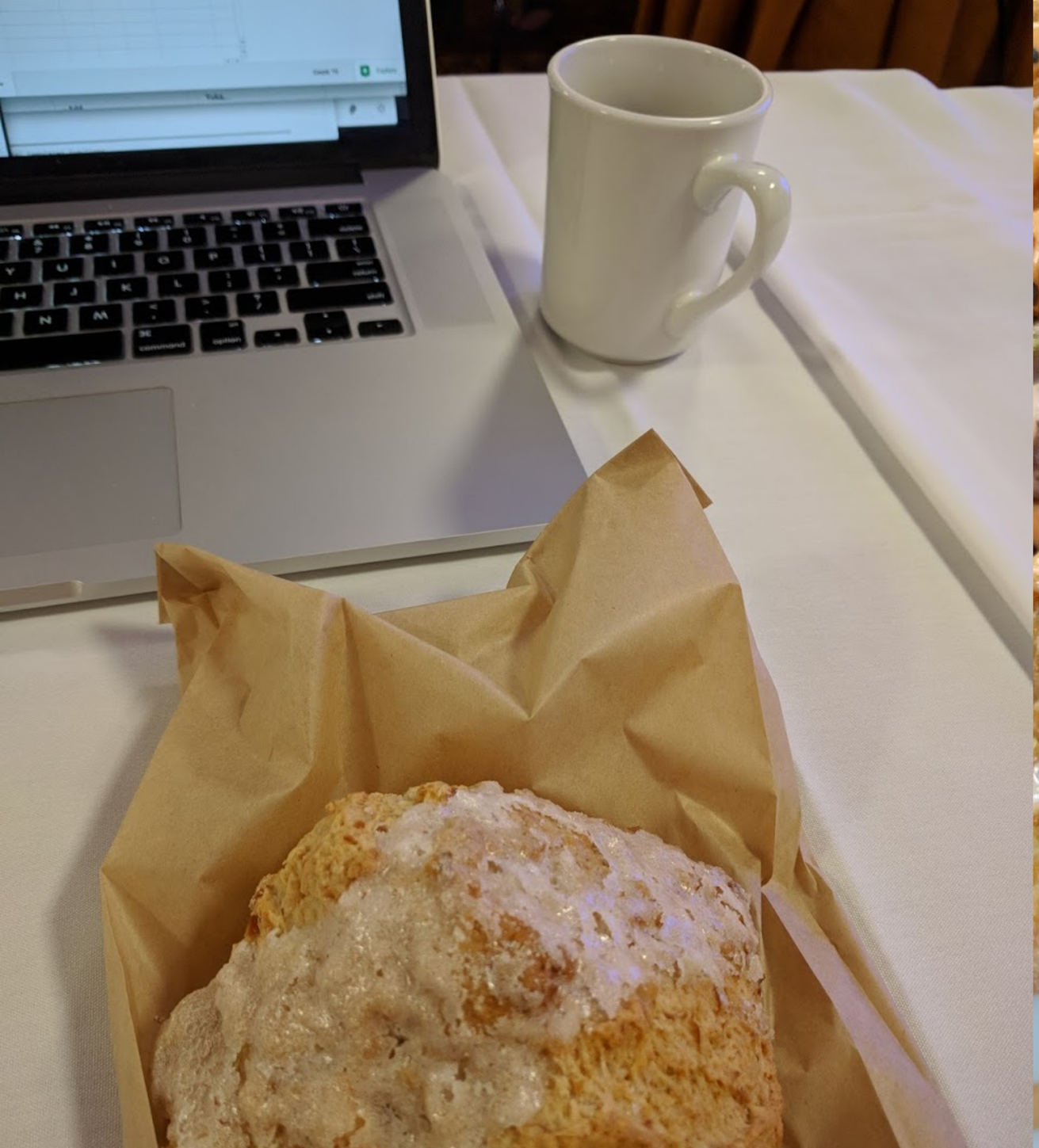
```
class A { int i; };
bool operator==(A, A) {...}
bool operator!=(A, A) {...}
bool operator>(A, A) {...}
bool operator<(A, A) {...}
bool operator>=(A, A) {...}
bool operator<=(A, A) {...}
```

```
class A {
  auto operator<=>(A) const = default;
  int i;
};
```

Summary

- C++20 will change how we write code
- Goals are simplicity, 0-cost, faster programs, common features in the library
- Implementations are on their way, most of C++17 already available

What did Axel do?



Reflection TS

- N4766

21.12.4.6 Member operations **[reflect.ops.member]**

1 A specialization of any of these templates with a meta-object type that is reflecting an incomplete type renders the program ill-formed. Such errors are not in the immediate context (17.9.2).

```
template <ScopeMember T> struct get_scope;
```

2 All specializations of `get_scope<T>` shall meet the `TransformationTrait` requirements (23.15.1). The nested type named `type` is the `Scope` reflecting a scope *S*. With *ST* being the scope of the declaration of the entity, typedef or value reflected by `T`, *S* is found as the innermost scope enclosing *ST* that is either a namespace scope (including global scope), class scope, enumeration scope, function scope (for the function's parameters), or immediately enclos-

- Accepted by ISO members, to be published in 2019

Hours of Discussions

- Mike: "where T reflects a cv-unqualified type or a non-type
- Axel: split 2.4 sub-bullet in type and non-type cases
- Needs work.

CH 072: detecting class-key struct vs. class

- EWG likes this change.
- John: This is an ugly change, because grammar term class-key covers class, struct, union.
- Axel: EWG did not want the correct phrasing.
- Richard: We require with this wording that implementers track struct/class. This is not a property of the type, it's a property of the declaration. Same issue as locations and parameter name.
- Richard: Is it valid for uses_class_key and uses_struct_key to both be false or both be true? Yes.
- Hubert: So, need to refer to a specification declaration, not to a "type".
- John: What about a template specialization using a differnent class-key?
- Richard: It is not clear that the properties of the primary template matter?
- Richard, Jens: "If T reflects a class with class-key class" is a category error (rescue attempts in later sentences notwithstanding)
- Richard: "If T reflects a class for which a declaration uses class-key "class".... Otherwise, the result is unspecified."
- Jens: snapshotting for reflexpr not required by the design;

And Evenings / Nights?

- Spent my nights updating the wording

```
> \pnum{8} All specializations of [`is_class`]{.rm}[`uses_class_key`]{.add}`<T>
.add}`<T>` shall meet the `UnaryTypeTrait` requirements (23.15.1). If `T` refle
ration uses]{.add}-]{+all declarations use]{.add}+} *class-key* `class` (for [`
) or `struct` (for [`is_struct`]{.rm}[`uses_struct_key`]{.add}`<T>`), the base
ecialization is [-`true_type`,-]{+`true_type`[,+} otherwise it is [-`false_type
ts a class for which no declaration uses *class-key* `class` (for `uses_class_k
`), the base characteristic of the respective template specialization is `false
 the base characteristic is `true_type` or `false_type`]{.add}. [If+} the same
y* `class` and *class-key* `struct`, the base characteristic of the template sp
 and `is_struct<T>` can be `true_type`, the other template specialization is `f
unspecified.-]{+unspecified.]{.rm}+}
```

- And preparing my straw poll vote for modules, co-routines

Conclusion

Outlook

- C++23 will be much smaller
- Implementing language features for library where needed: stdlib modules, more concepts
- Several major features did not make it: networking, executors
- More features in the works, e.g. reflection

Conclusion (1/2)

- C++ learned a lesson: evolve or be dead
- Relevant to us: better, more maintainable code
- We won't need all features, but we have a palette to select from

Conclusion (2/2)

- Many features are targeted at us: math special functions (C++17), ranges, concurrency, compile times of large-scale code
- We should benefit from what the language and its tools provide
- We need to evolve tooling and code to benefit
- Need to upgrade our coding guidelines, selecting “allowed” features: contracts? concepts? coroutines?