constexpr type ordering (P2830R4)

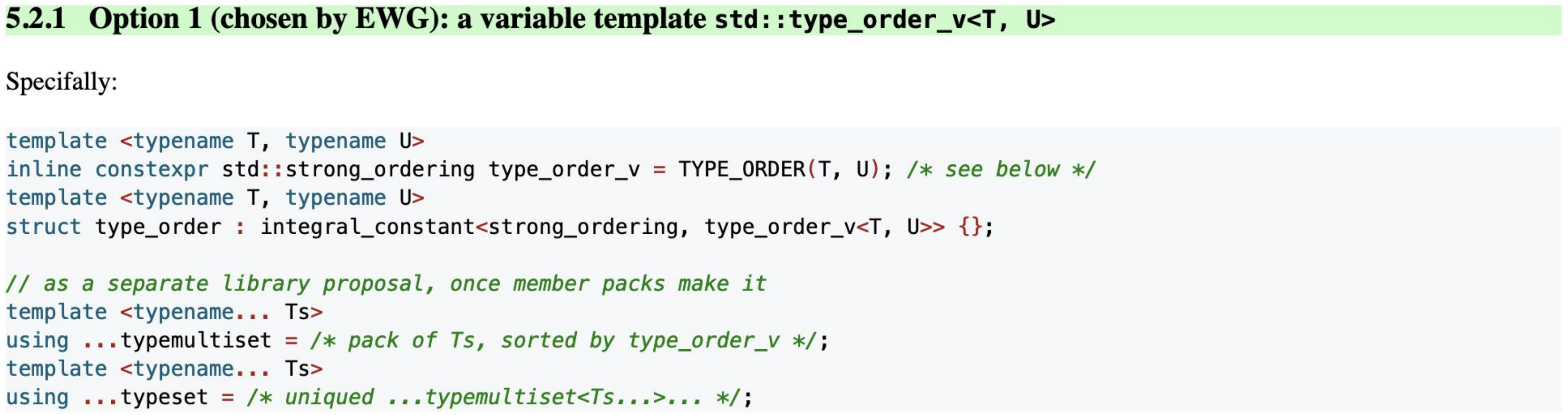
WHY C++26

- std::execution-style code really needs typesets for efficient (code-size) implementations

We know this because there are **pretty_function** based implementations in most implementations I know of.

- Ordering is more fundamental than sets.

Syntax (EWG already confirmed this)



5.2.1 Option 1 (chosen by EWG): a variable template `std::type_order_v<T, U>`

Specifally:

```
template <typename T, typename U>
inline constexpr std::strong_ordering type_order_v = TYPE_ORDER(T, U); /* see below */
template <typename T, typename U>
struct type_order : integral_constant<strong_ordering, type_order_v<T, U>> {};

// as a separate library proposal, once member packs make it
template <typename... Ts>
using ...typemultiset = /* pack of Ts, sorted by type_order_v */;
template <typename... Ts>
using ...typeset = /* uniqued ...typemultiset<Ts...>... */;
```

Desired properties

Stability

Order should not change between compilation units (crucial for API compatibility)

Free-standing

Type ordering should not rely on non-free-standing features

Self-consistency

`type_order_v<T, U> == type_order_v<some_template<T>, some_template<U>>`.

Reflection compatibility

Any `operator<=>(std::meta::info, std::meta::info)` should be consistent with this one.

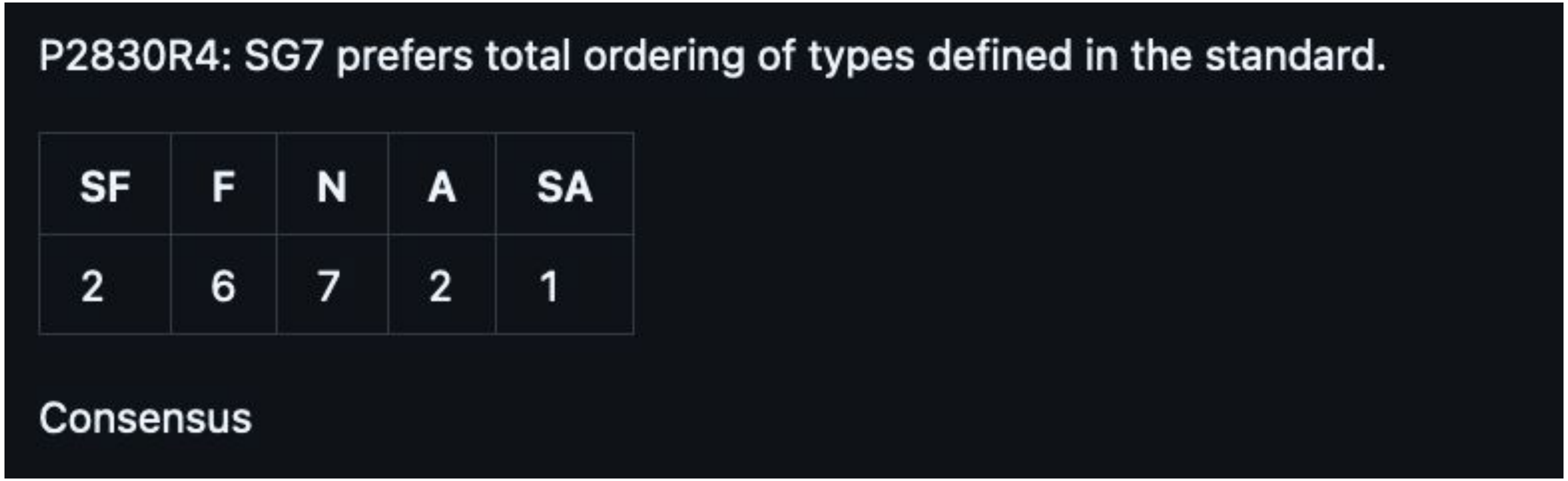
- Can’t have this syntax because this `operator⇔` would need to return a partial order (it reflects more than types)

non-goals

Consistency with `type_info::before()`

- Impossible: some implementations don’t have consistent type_info::before() even between runs of the same application
- type_info ignores cv-ref qualifiers

SG7 recommendation



P2830R4: SG7 prefers total ordering of types defined in the standard.

SF	F	N	A	SA
2	6	7	2	1

Consensus

Main question

Implementation-defined or fully specified by the standard?

- Implementation defined:
  - Pro: ABIs already did all the work
  - Cons:
    - ABIs don’t agree
    - Frontend doesn’t know ABI for static analysis tools
    - Layering violation
    - Compilers need to agree to have compatible ABI
    - Not self-consistent (name mangling uses compression)
- Fully specified:
  - Pros:
    - Fully portable, including static analysis tools
    - Faster than mangling during constexpr evaluation
      - (comparison does not require stringifying long symbol names, it short-circuits quickly)
    - Does not require the frontend to know the ABI (helps IDEs)
  - Cons:
    - Lots of work
    - Anonymous entities require a completely new-to-standard notion of a “declaration scope” with all the template **arguments** of all enclosing scopes
    - We need to continue to specify ordering for every change to language entities