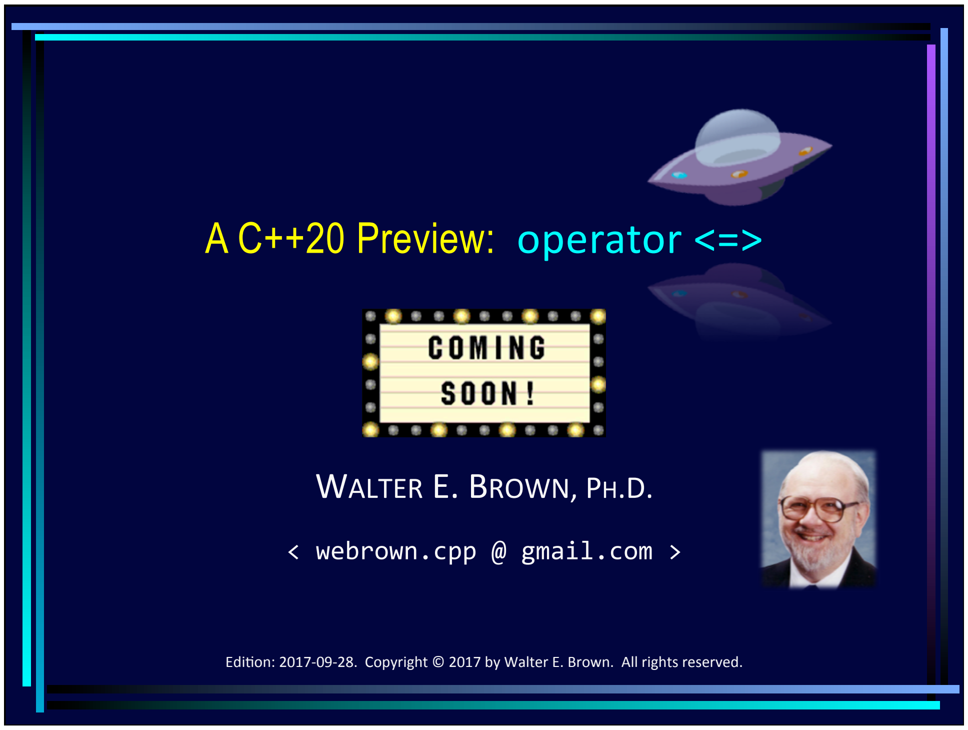


A C++20 Preview: operator <=>

COMING SOON!

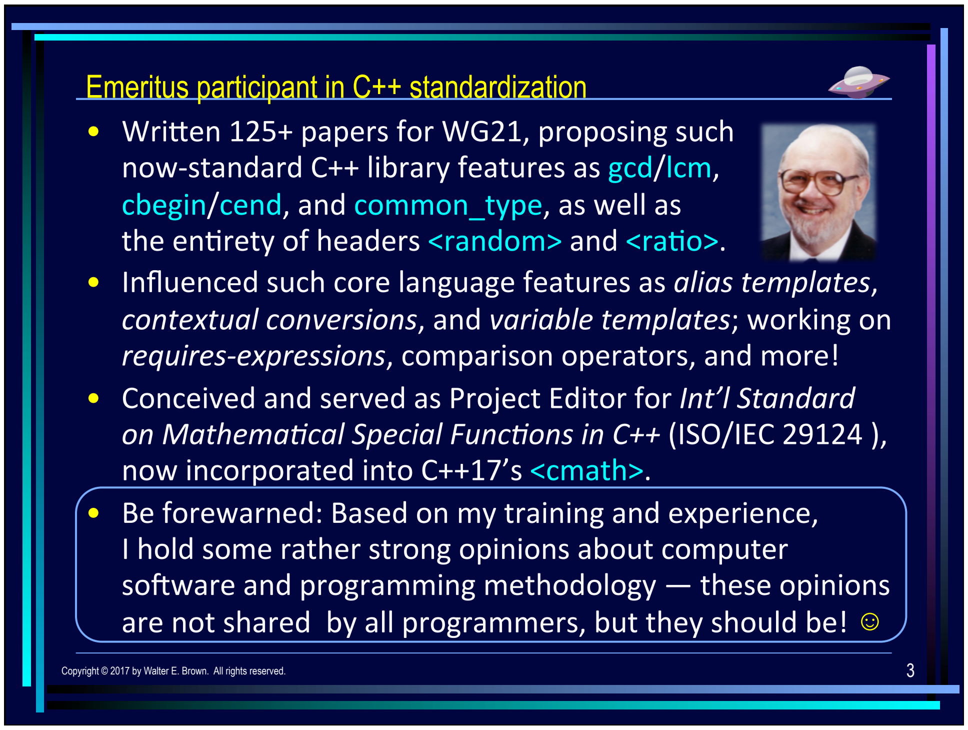
WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >

A little about me

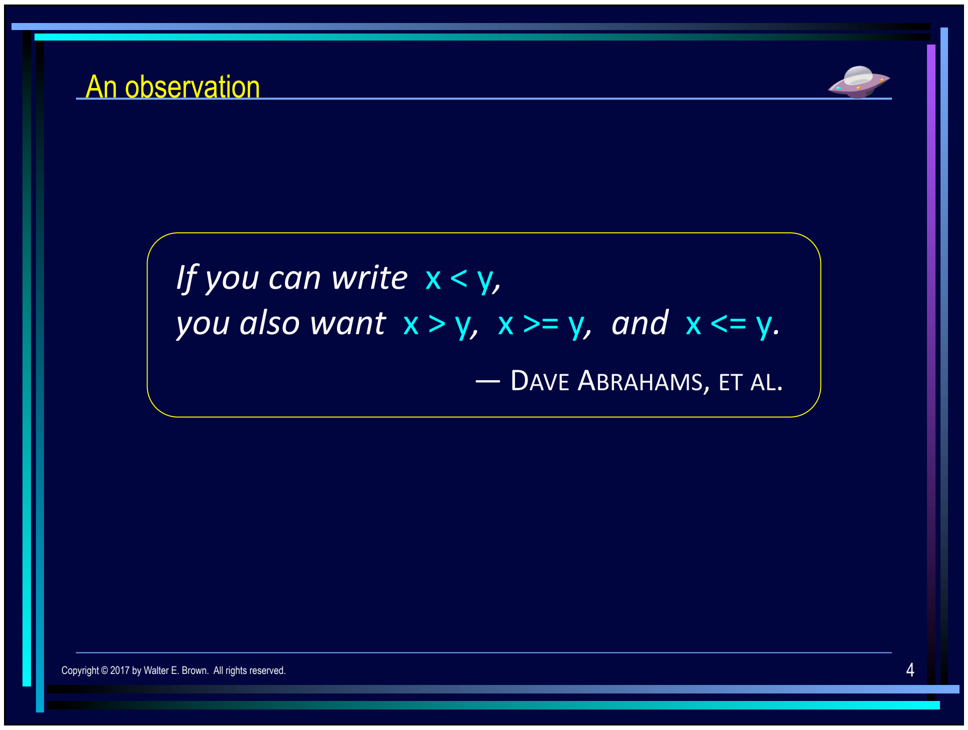
- B.A. (math's); M.S., Ph.D. (computer science).
- Professional programmer for over 50 years, programming in C++ since 1982.
- Experienced in industry, academia, consulting, and research:
  - Founded a Computer Science Dept.; served as Professor and Dept. Head; taught and mentored at all levels.
  - Managed and mentored the programming staff for a reseller.
  - Lectured internationally as a software consultant and commercial trainer.
  - Retired from the Scientific Computing Division at Fermilab, specializing in C++ programming and in-house consulting.
- Not dead — still doing training & consulting. (Email me!)

2



Emeritus participant in C++ standardization

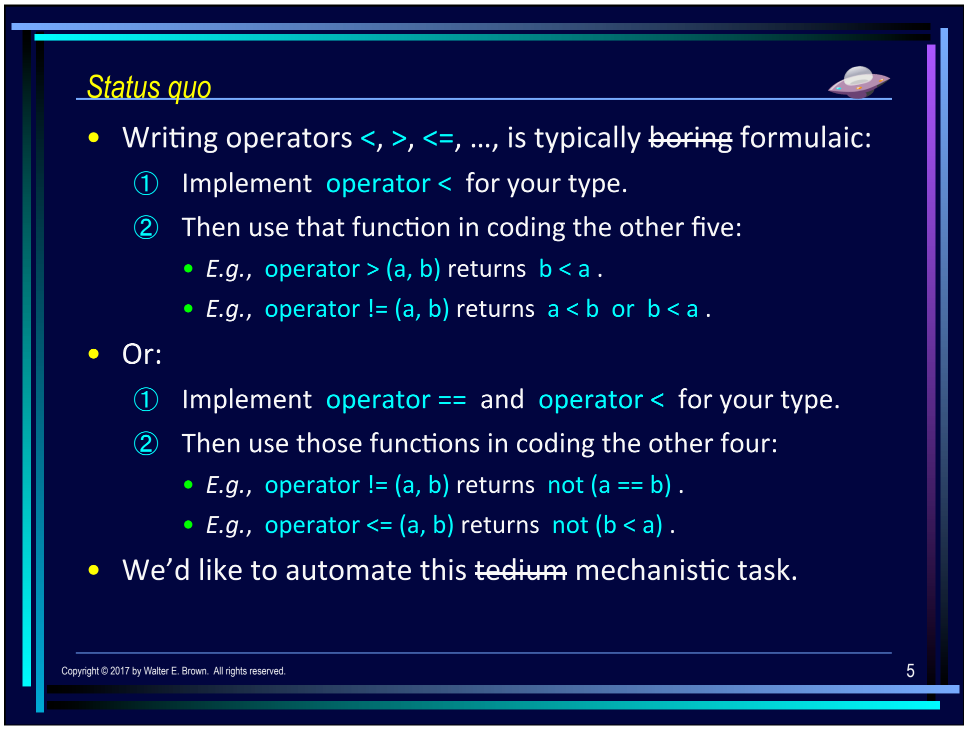
- Written 125+ papers for WG21, proposing such now-standard C++ library features as gcd/lcm, cbegin/cend, and common_type, as well as the entirety of headers <random> and <ratio>.
- Influenced such core language features as *alias templates*, *contextual conversions*, and *variable templates*; working on *requires-expressions*, comparison operators, and more!
- Conceived and served as Project Editor for *Int'l Standard on Mathematical Special Functions in C++* (ISO/IEC 29124), now incorporated into C++17's <cmath>.
- Be forewarned: Based on my training and experience, I hold some rather strong opinions about computer software and programming methodology — these opinions are not shared by all programmers, but they should be! ☺



An observation

If you can write x < y,
you also want x > y, x >= y, *and* x <= y.

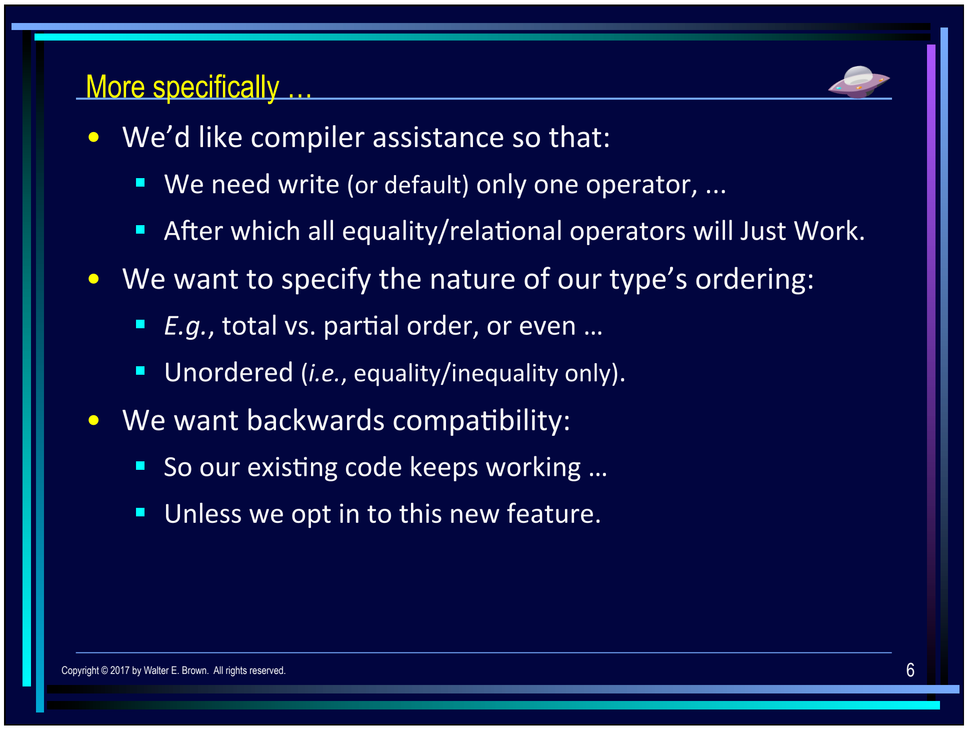
— DAVE ABRAHAMS, ET AL.



Status quo

- Writing operators <, >, <=, …, is typically ~~boring~~ formulaic:
  - ① Implement operator < for your type.
  - ② Then use that function in coding the other five:
    - *E.g.*, operator > (a, b) returns b < a .
    - *E.g.*, operator != (a, b) returns a < b or b < a .
- Or:
  - ① Implement operator == and operator < for your type.
  - ② Then use those functions in coding the other four:
    - *E.g.*, operator != (a, b) returns not (a == b) .
    - *E.g.*, operator <= (a, b) returns not (b < a) .
- We'd like to automate this ~~tedium~~ mechanistic task.

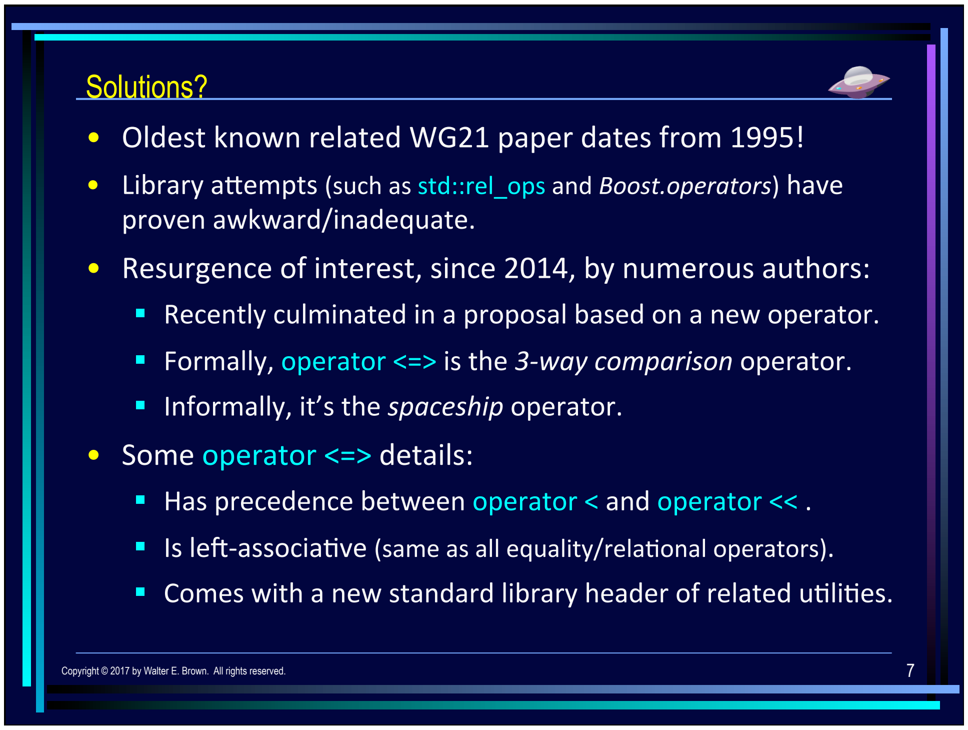
5



More specifically …

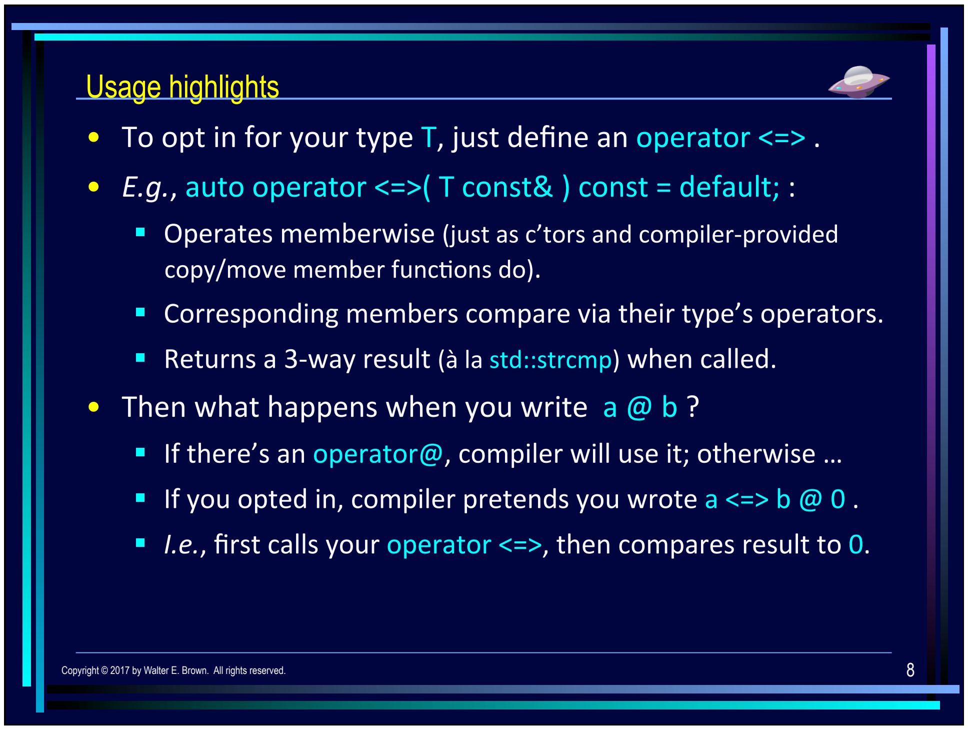
- We'd like compiler assistance so that:
  - We need write (or default) only one operator, …
  - After which all equality/relational operators will Just Work.
- We want to specify the nature of our type's ordering:
  - *E.g.*, total vs. partial order, or even …
  - Unordered (*i.e.*, equality/inequality only).
- We want backwards compatibility:
  - So our existing code keeps working …
  - Unless we opt in to this new feature.

6



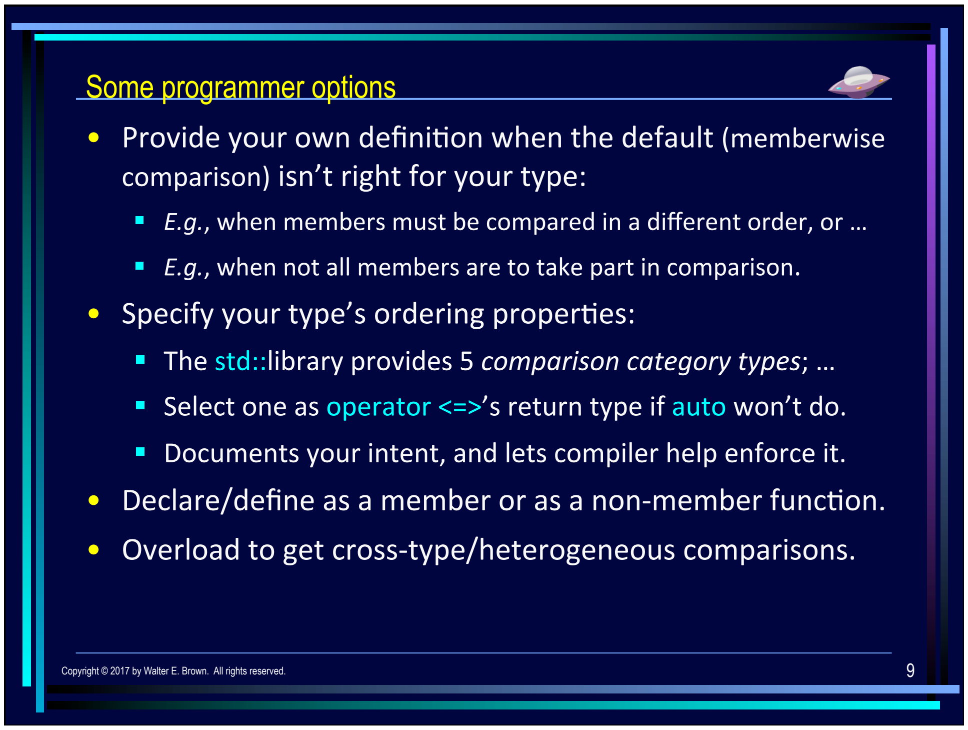
Solutions?

- Oldest known related WG21 paper dates from 1995!
- Library attempts (such as std::rel_ops and *Boost.operators*) have proven awkward/inadequate.
- Resurgence of interest, since 2014, by numerous authors:
  - Recently culminated in a proposal based on a new operator.
  - Formally, operator <=> is the *3-way comparison* operator.
  - Informally, it's the *spaceship* operator.
- Some operator <=> details:
  - Has precedence between operator < and operator << .
  - Is left-associative (same as all equality/relational operators).
  - Comes with a new standard library header of related utilities.



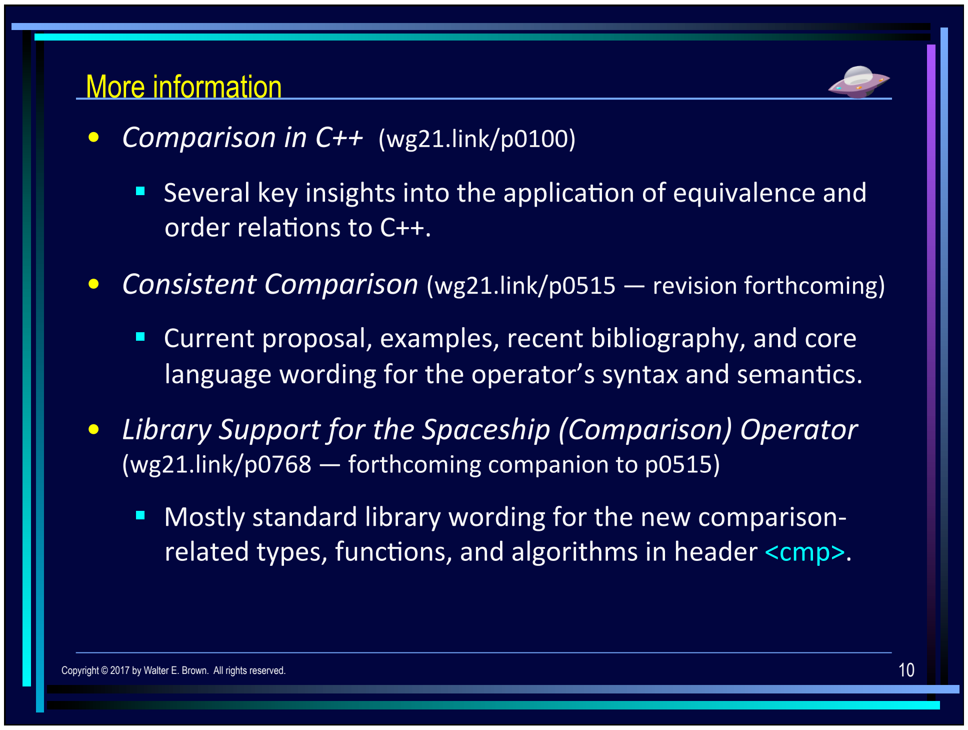
Usage highlights

- To opt in for your type T, just define an operator <=> .
- *E.g.*, auto operator <=>(T const&) const = default; :
  - Operates memberwise (just as c'tors and compiler-provided copy/move member functions do).
  - Corresponding members compare via their type's operators.
  - Returns a 3-way result (à la std::strcmp) when called.
- Then what happens when you write a @ b ?
  - If there's an operator@, compiler will use it; otherwise …
  - If you opted in, compiler pretends you wrote a <=> b @ 0 .
  - *I.e.*, first calls your operator <=>, then compares result to 0.



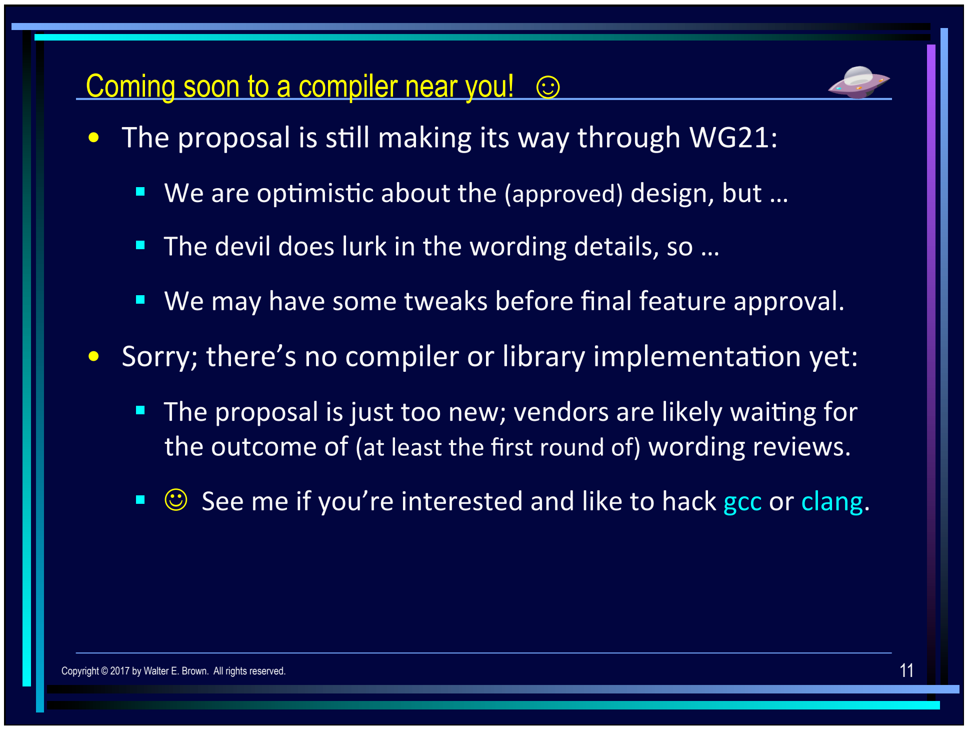
Some programmer options

- Provide your own definition when the default (memberwise comparison) isn't right for your type:
  - *E.g.*, when members must be compared in a different order, or …
  - *E.g.*, when not all members are to take part in comparison.
- Specify your type's ordering properties:
  - The std::library provides 5 *comparison category types*; …
  - Select one as operator <=>'s return type if auto won't do.
  - Documents your intent, and lets compiler help enforce it.
- Declare/define as a member or as a non-member function.
- Overload to get cross-type/heterogeneous comparisons.



More information

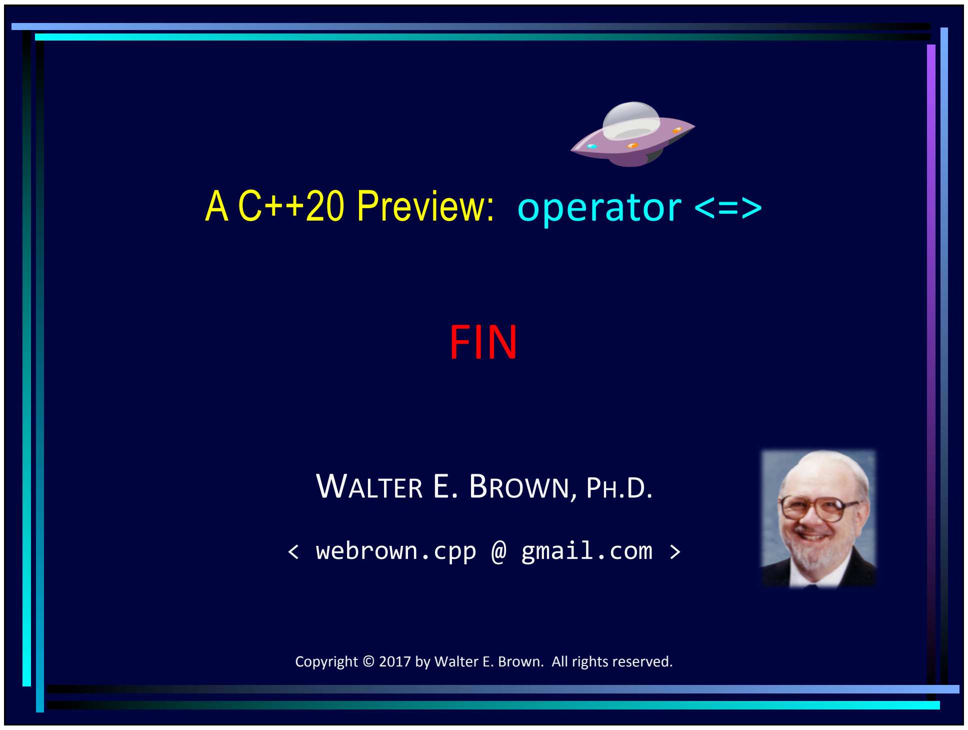
- *Comparison in C++* (wg21.link/p0100)
  - Several key insights into the application of equivalence and order relations to C++.
- *Consistent Comparison* (wg21.link/p0515 — revision forthcoming)
  - Current proposal, examples, recent bibliography, and core language wording for the operator's syntax and semantics.
- *Library Support for the Spaceship (Comparison) Operator* (wg21.link/p0768 — forthcoming companion to p0515)
  - Mostly standard library wording for the new comparison-related types, functions, and algorithms in header <cmp>.



Coming soon to a compiler near you! ☺

- The proposal is still making its way through WG21:
  - We are optimistic about the (approved) design, but …
  - The devil does lurk in the wording details, so …
  - We may have some tweaks before final feature approval.
- Sorry; there's no compiler or library implementation yet:
  - The proposal is just too new; vendors are likely waiting for the outcome of (at least the first round of) wording reviews.
  - ☺ See me if you're interested and like to hack gcc or clang.

11



A C++20 Preview: operator <=>

FIN

WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >