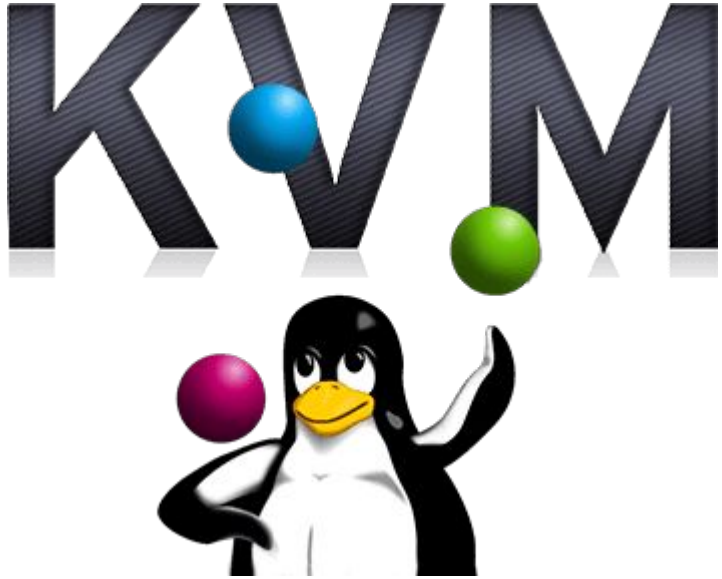
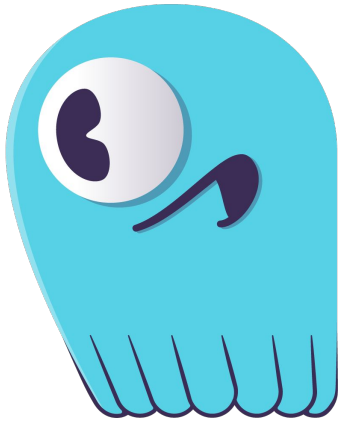

Core C++ 2026
20-21Oct. 2026 :: Tel-Aviv
Trying to stay sane modularizing a large C++ project
Avi Kivity

About Me

- Creator of Linux KVM
- CTO, co-founder @ ScyllaDB
- Seastar, ScyllaDB maintainer





Agenda

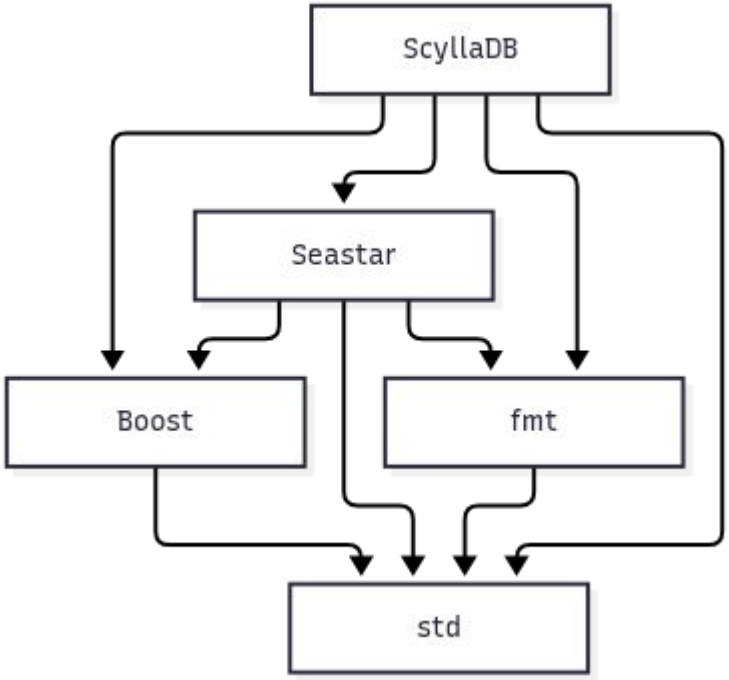
- Goals
- Understanding the problem
- Existing practice
- External vs internal modularization
- Introduction to C++20 modules
- Compiler support, tooling
- ScyllaDB dependency layout
- C++20 module variants
- Attack plan
- Implementation
- Status and future

Goals

- Reduce compile time
- Promote logical modularity
- Dispel the shame

Setting the stage

- ScyllaDB
  - Sources: ~0.9k translation units, 1.2k headers
- Major C++ dependencies
  - C++ Standard Library
  - Seastar
  - fmt
  - Boost
- Influence
  - Co-maintainer of Seastar
  - Contributor to fmt
  - Contributor to libstdc++
    - But low release frequency



The NxM Problem

- N translation units (.cpp, .cc)
- M headers (.h, .hpp, .hh)
- O(N*M) files parsed for a full build
- Project grows -> N and M grow
- Despair

External vs internal modularization

- External: consume third-party libraries as modules instead of headers
- Internal: convert internal sources to a module graph

Existing practice: precompiled headers

- Precompile a set of relatively fixed headers into one precompiled header
- Non-standard extension
- Source is no longer source-of-truth
- Works with relatively stable external dependencies
- Tooling: ccache, not distcc or sccache
- Can give 10% improvement
- Our case: std, fmt, boost, seastar

Introduction to C++20 modules

- Introduced 32,663.42 years too late
- 79 different ways of accomplishing anything
- Obfuscated terminology
- Compiler support lagging
- Tooling support lagging++

Introduction to C++20 modules

- Introduced 32,663.42 years too late
- 79 different ways of accomplishing anything
- Obfuscated terminology
- Compiler support lagging
- Tooling support lagging++

- But we’re addicted

Introduction to C++20 modules

- Keyword **module**
- Keyword **export**
- Keyword **import**
- Keyword **export import**
- Keyword **constexpr static inline module import export**

What do modules do?

- Concatenate headers into a long source
- Parse the source

- Parse module into Binary Module Interface (BMI)
- Import BMI into consuming module
  - Import is fast*
  - Multiple consumers can import a BMI

Limitations

- No macros
  - Mostly okay as best practice is to avoid macros
  - Boost.Test a glaring exception
  - Beware of headers that expect to be configured via macros
- Global linkage only
  - No ‘static inline’

Compiler support

- gcc 11 (2021) - based on earlier draft, incomplete
- gcc 15 (2025) - standard compliant, more or less
- Clang 3 (2012) - nonstandard modules
- Clang 5 (2017) - early draft
- Clang 15 (2022) - C++20 module support, incomplete
- Many incremental improvements
- Many, many subtle bugs

Tooling support

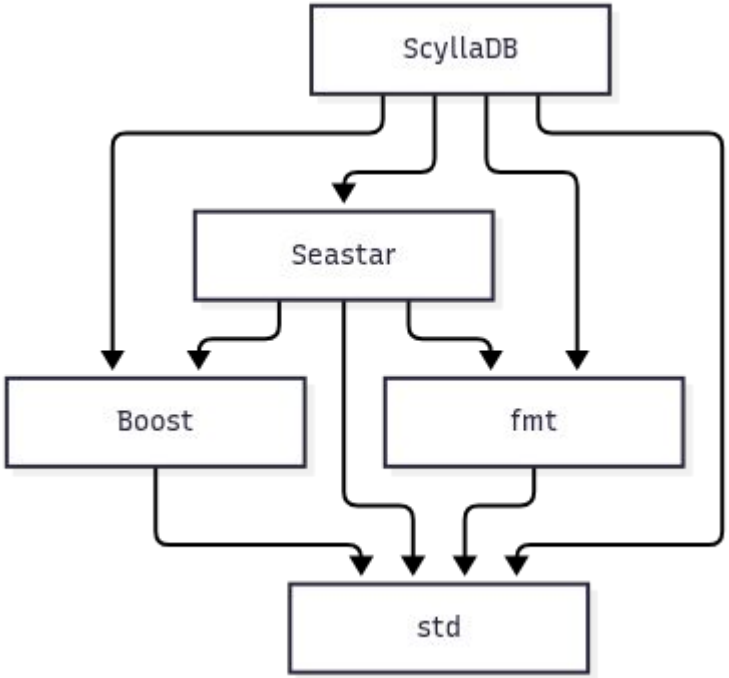
- ccache: draft pull request
  - Implement support for C++20 modules- #1523 #1523
- distcc: open ticket
  - Support with C++20 modules #470
- icecream: wants to merge into ccache/distcc
- sccache: merged
  - partial c++20 module support- #2516
  - Follow-up improvements
  - Made the project viable
- https://arewemodulesyet.org/
- clangd

C++ Module Styles

- Modules
  - Similar to non-insane languages
  - Hermetic
  - Name mangling suffix “@foo”
  - “export void ns::f();”
- Wrapper modules
  - Closer to traditional headers
  - Same names can arrive from headers and modules
  - No name mangling changes
  - “export namespace ns { using ns::f; }”
- Wrapper modules are critical for incremental transition
  - Wacky
  - Many subtle compiler bugs

Plan of attack

- libstdc++: wrapper (vendor)
- fmt: module (vendor)
- Seastar: wrapper (vendor/me)
- Boost: wrapper (me)



- Wrapper modules crucial for partially modularized source
- Expect partial modularization to last a long time

Macros

- FLT_EPSILON and similar: convert to std::numeric_limits
  - Mostly easy to solve
- Boost.Test: keep as headers
- Seastar-testing: keep as headers (depends on Boost.Test)
- Library version macros: export via build system (-D)

Seastar modularization

- One seastar.cppm file that exports everything
- Using wrapper style

```
224   #include <seastar/coroutine/generator.hh>
225   #include <seastar/coroutine/maybe_yield.hh>
226   #include <seastar/coroutine/parallel_for_each.hh>
227   #include <seastar/coroutine/switch_to.hh>
228   #include <seastar/coroutine/try_future.hh>
229
230   export module seastar;
231
232   export namespace seastar {
233
234   using seastar::abort_on_expiry;
235   using seastar::abort_requested_exception;
236   using seastar::abort_source;
237   using seastar::accept_result;
```

Boost modularization

- Create .cppm files for each important Boost library
- Supermodule so “`import boost`” works
- Not possible to modularize Boost.Test
- A few symbols don’t work (static linkage)

Fmt, std modularization

- Use vendor modules
- Change Seastar to optionally consume fmt via a module
- Use `import std.compat` to use C symbols standardized via C++

Clang bugs

- Clang crashes in `Sema::IsValueInFlagEnum` on a `switch` over a flag enum that came from a PCH **clang:frontend** **crash**
  Bug llvm/llvm-project#223243 · by avikivity was closed on Sep 14 — 1 — 4
- Duplicate global-module-fragment entities from two imported modules fail to merge, causing spurious "different kind of symbol" / ambiguous-conversion errors **clang:modules**
  llvm/llvm-project#207581 · by avikivity was closed on Jul 9 — 1 — 16
- [clang][modules] ambiguity when an inline variable is visible both through a header and through a module **clang:modules** **release:backport**
  llvm/llvm-project#190333 · by avikivity was closed on May 8 · LLVM 22.x Release — 3 — 11
- [clang][C++20 modules] textual `#include` sandwiched between two `import` declarations loses enum declarations **clang:modules** **release:backport**
  Bug llvm/llvm-project#188853 · by avikivity was closed on Apr 2 · LLVM 22.x Release — 2 — 12

Clang bugs

- Clang checks for a symbol reachable from both a header and a module
- Legal (with wrapper modules) but must verify
- Many, many subtle bugs in this area

Solving clang bugs

- Ask Claude to create a minimal reproducer
- Encourage it when the going is tough
- Refuse to apply workarounds
- Mix of looking at the original reproducer and looking at clang sources
- File bug with Claude analysis
- File pull request with Claude fix
- Close pull request when maintainer comes up with right fix
- Shout out @ChuanqiXu9 for guidance and for fixing the bugs
    - Also great with coroutines

Everything compiles

- It’s ~30% slower

Panic

- Modularize “minor dependencies”
  - ~4 new wrapper modules
  - Avoids header-module contention via leakage
- Use `perf diff` to understand source of slowness
- Patch clang to avoid premature desertialization
  - Complex patch series
  - Yet to be upstreamed

BMI Structure

- Symbol directory
- Serialized representation of parse tree
  - Includes inline bodies
- Indirectly used symbols
  - Function parameter names
  - Symbols used in inline bodies
- Template instantiations
  - Can bloat BMIs
  - NxM problem (but in space, not time)

Closing words

- Can not yet claim success
- Very hard to get improvement without modularizing internals
- But series is already complex enough
- Need at least break-even
- Could not do without Claude

Thanks / Q&A