C++ Club Meeting Notes

Gleb Dolgich

2017-12-07

Link

Buy your own copy for only CHF 198 / USD 202 / GBP 150 / BTC 0.017363

CLion 2017.3 Released

- Announcement
- What's New video with Phil Nash
- Download
- Improvements in support for list initialization, name lookup, MSVC support, JUCE library
- Support for Valgrind Memcheck
- Support for Boost.Test
- Ability to configure multiple toolchains

Meeting C++ Trip Report

- Jean Guegant

C++ Standards Meeting in Albuquerque, November 2017

Trip report by Botond Ballo

- C++17: GCC and Clang: done, MSVC: by March 2018
- C++20 news
- TS: Coroutines, Ranges (a leading candidate for introducing Concepts into the Standard Library), Networking
- Modules TS, Parallelism TS v2, Concurrency TS v2
- Future: reflection, metaclasses (updated), graphics, numerics
- Concepts: adjective syntax for Abbreviated Function Templates (AFT): `void sort(Sortable& auto s);`
- Rejected proposals, discussion papers
- Modules v2: context-sensitive `module` keyword, module partitions; continue to explore macro support

Article by Julian Templeman

- Structured bindings for tuples, arrays, and structs
- New library types and containers: `std::variant`, `std::byte`, `std::optional`, `std::any`

CppCon 2017: Adrien Devresse “Nix: A functional package manager for your C++ software stack”

- YouTube
- Home

Conan 0.29

- Changelog
- Introduction

Article

Conan

PDF

- That opening Machiavelli quote :-)
- Modules TS introduces a concept of an artifact that depends on the sources and requires a build step, but doesn't specify how to do it
- Turning C++ compiler into a build system: not recommended, but possible
- *build2* understands module dependencies (a separate tool pass required)
- Different binary formats in different compilers (MSVC: open IFC format, Clang: own format, GCC: own format) $\Rightarrow$ translation?
- No packaging support: an opportunity

P0804R0

- The tool must now be able to resolve module import declarations to either the source code for the corresponding module interface unit, or to some module artifact that provides the exported entities for the module.
- Though module artifact are not intended to be a distribution format or an alternative to access to source code, motivation exists to use them in this way.

Modules in Build2

Docs

- From a consumer's perspective, a module is a collection of external names, called module interface, that become visible once the module is imported.
- A module does not provide any symbols, only C++ entity names.
- From the producer's perspective, a module is a collection of module translation units: one interface unit and zero or more implementation units.
- When building a shared library, some platforms (notably Windows) require that we explicitly export symbols that must be accessible to the library users.

Modules in Build2 (cont.)

- Module interface units are by default installed in the same location as headers (for example, `/usr/include`). However, instead of relying on a header-like search mechanism (-I paths, etc.), an explicit list of exported modules is provided for each library in its .pc (pkg-config) file.
- Mega-modules vs. mini-modules: The sensible approach is to create modules of conceptually-related and commonly-used entities possibly complemented with aggregate modules for ease of importation.
- The sensible guideline is to have a separate module implementation unit except perhaps for modules with a simple implementation that is mostly inline/template.

Modules in Build2: module interface unit template

```
1 // Module interface unit.
2 // <header includes>
3 export module <name>;          // Start of module purview.
4 // <module imports>
5 // <special header includes> <- Configuration, export, etc.
6 // <module interface>
7 // <inline/template includes>
```

Modules in Build2: module implementation unit template

```
1 // Module implementation unit.
2 // <header includes>
3 module <name>;              // Start of module purview.
4 // <extra module imports>  <- Only additional to interface.
5 // <module implementation>
```

Modules in Build2 (cont.)

The possible backwards compatibility levels are:

- modules-only (consumption via headers is no longer supported);
- modules-or-headers (consumption either via headers or modules);
- modules-and-headers (as the previous case but with support for consuming a library built with modules via headers and vice versa).

C++Now 2013 Keynote: Chandler Carruth “Optimizing the Emergent Structures of C++”

YouTube

(Terrible clipped sound. Terrible jerky camera tracking and zooming. Good content made unwatchable by unprofessional video. Your eyes and ears will bleed.)

- Antipattern: passing output parameter by reference instead of returning by value
- Value semantics allow compilers apply more optimisations
- Member function calls are evil (need to take address of `this`) *(Caveat: this applies to Clang in 2013)*

  Trust the compiler. You have no idea how smart the compiler is. It’s terrifying!

C++Now 2013 Keynote: Chandler Carruth “Optimizing the Emergent Structures of C++” (cont.)

```
1  struct S {
2      float x, y, z;
3      double delta;
4      double compute();
5  };
6  ...
7  double f() {
8      S s;
9      s.x = /* expensive compute */;
10     s.y = /* expensive compute */;
11     s.z = /* expensive compute */;
12     s.delta = s.x - s.y - s.z;
13      // next line is a killer, says Chandler
14     return s.compute();
15 }
```

C++Now 2013 Keynote: Chandler Carruth “Optimizing the Emergent Structures of C++” (cont.)

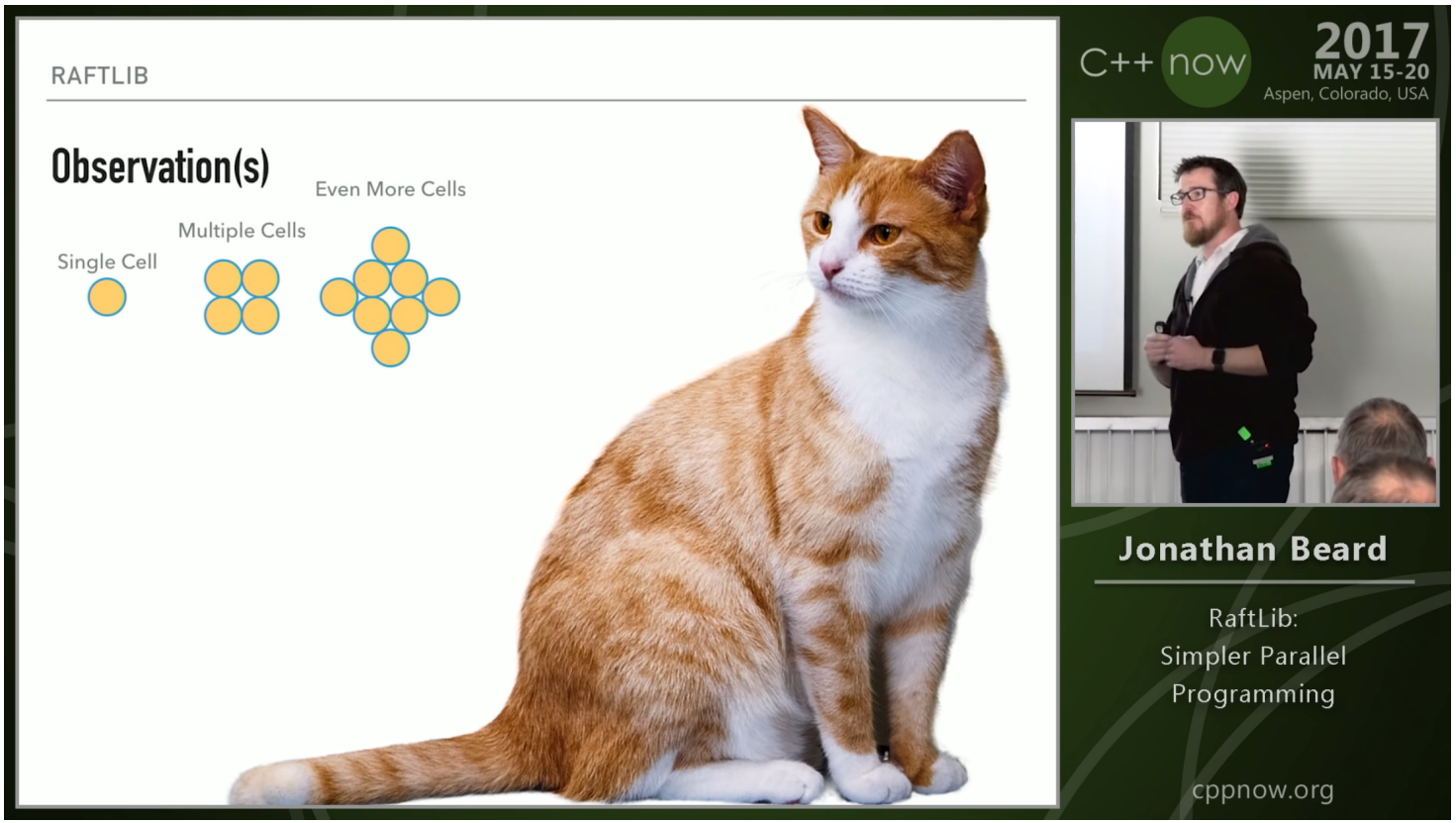
Tips for optimizable APIs

- Use value semantics.
- Don’t create unneeded abstractions. Sometimes, a function parameter is plenty.
- Partition **all** logic away from template-expanded deeply nested constructs *(YMMV)*

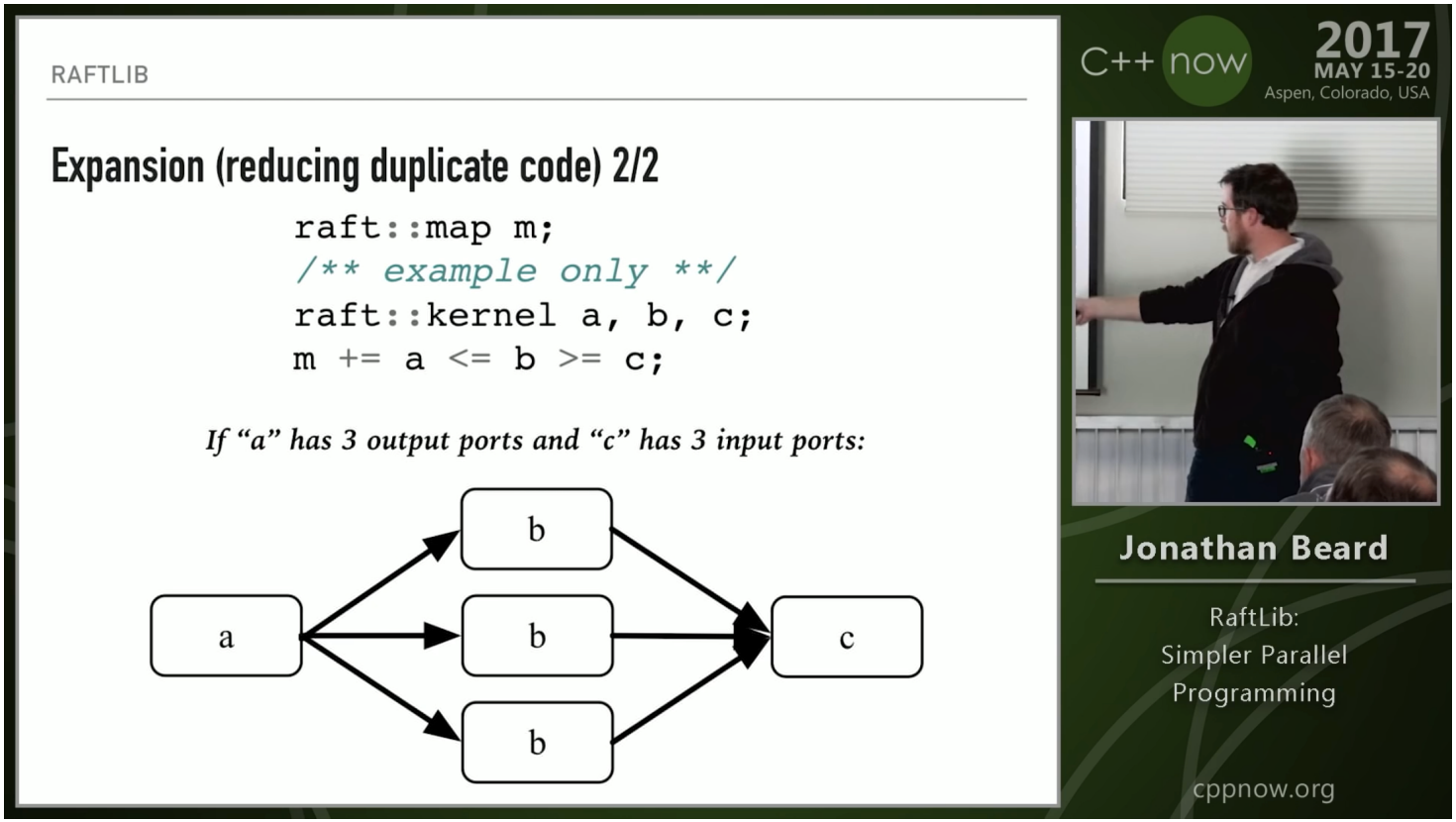
Use abstractions, but also consider how they will look to the optimizer.

C++Now 2017: Jonathan Beard “RaftLib: Simpler Parallel Programming”

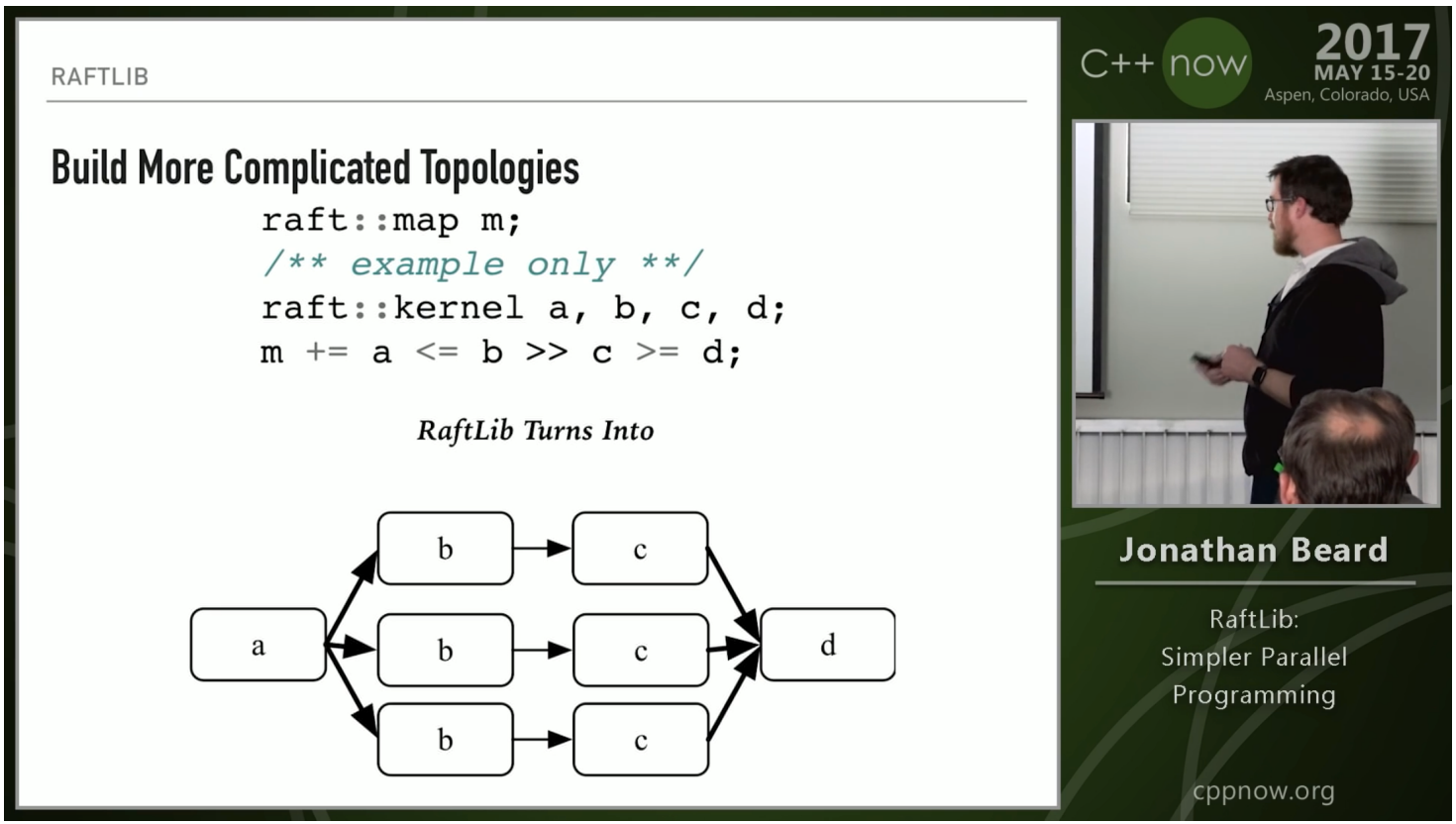
YouTube * GitHub (Apache 2.0)



C++Now 2017: Jonathan Beard “RaftLib: Simpler Parallel Programming” (cont.)



C++Now 2017: Jonathan Beard “RaftLib: Simpler Parallel Programming” (cont.)



CppCon 2017: Billy Baker “Almost Unlimited Modern C++ in Kernel-Mode Applications”

- YouTube

Linus Torvalds, 2004:

> *Trust me - writing kernel code in C++ is a BLOODY STUPID IDEA.*

CppCon 2017: Billy Baker “Almost Unlimited Modern C++ in Kernel-Mode Applications” (cont.)

Windows (2012): Visual C++ `/kernel` option

- No exceptions (compiler error on `try`/`catch`)
- No RTTI (compiler error on `dynamic_cast` and `typeid`)
- Users must replace `new` and `delete`

CppCon 2017: Billy Baker “Almost Unlimited Modern C++ in Kernel-Mode Applications” (cont.)

Options:

- VxWorks (WindRiver) – GCC
- Linux – GCC, Clang
- On Time – Visual C++, Borland C++
- Windows/TenaSys InTime – Visual C++, Intel ICC, Clang
- Windows/IntervalZero RTX – Visual C++, Intel ICC, Clang
- bare metal – Borland C++

Windows is not reliable enough for mission-critical systems (frequent reboots, intermittent misbehavious, instability, background tasks, OOM errors).

CppCon 2017: Billy Baker “Almost Unlimited Modern C++ in Kernel-Mode Applications” (cont.)

Not just C++:

- Fortran
- D (but: GC?)
- Lua

CppCast: Oculus Research with Dave Moore

- Episode page
- Everything is written in C++
- Targeting 90 FPS
- The “Nope” reaction

Quote

Unknown:

> *If you put a million monkeys on a million keyboards, one of them will eventually write a Java program. The rest of them will write Perl programs.*

Quote

Michael Hartung:

> *Hardware eventually fails. Software eventually works.*