Listing 5.1 and Listing 6

```
// P0800 Listing 5 revised: consult https://godbolt.org/g/ytP6fw for diagnostics
#include <regex>
#include <type_traits>
#include <range/v3/all.hpp>

int main()
{
  std::enable_if_t<ranges::Regular<std::regex>(), std::regex> foo{};
}
```

```
#include <experimental/ranges/algorithm>
#include <experimental/ranges/iterator>
#include <regex>

using std::experimental::ranges::Regular;

int main()
{
  Regular foo = std::regex{};
}
```

Tag dispatching (Listing 11)

```cpp
#include <iterator>

template <typename I>
void advance_helper(I i, typename std::iterator_traits<I>::difference_type n, std::random_access_iterator_tag) {
   i += n;
}

template <typename I>
void advance_helper(I i, typename std::iterator_traits<I>::difference_type n, std::bidirectional_iterator_tag) {
   for (; n > 0; --n)
      ++i;
   for (; n < 0; ++n)
      --i;
}

template <typename I>
void advance_helper(I i, typename std::iterator_traits<I>::difference_type n, std::input_iterator_tag) {
   for (; n > 0; --n)
      ++i;
}

template <typename I>
void advance(I i, typename std::iterator_traits<I>::difference_type n) {
   advance_helper(i, n, typename std::iterator_traits<I>::iterator_category{});
}
```

Killing tag dispatching (Listing 12)

```
#include <experimental/ranges/iterator>

namespace ranges = std::experimental::ranges;

template <ranges::RandomAccessIterator I>
void advance_helper(I i, ranges::difference_type_t<I> n) {
   i += n;
}

template <ranges::BidirectionalIterator I>
void advance_helper(I i, ranges::difference_type_t<I> n) {
   for (; n > 0; --n)
      ++i;
   for (; n < 0; ++n)
      --i;
}

template <ranges::InputIterator I>
void advance_helper(I i, ranges::difference_type_t<I> n) {
   for (; n > 0; --n)
      ++i;
}

template <typename I>
void advance(I i, ranges::difference_type_t<I> n) { advance_helper(i, n); }
```

constexpr-if (Listing 13.1)

```
// P0800 Listing 13 revised
#include <range/v3/all.hpp>

template <typename I>
void advance(I i, ranges::difference_type_t<I> n)
{
   if constexpr (ranges::RandomAccessIterator<I>()) {
      i += n;
   }
   else if constexpr (ranges::BidirectionalIterator<I>()) {
    for (; n > 0; --n)
       ++i;
    for (; n < 0; ++n)
       --i;
   }
   else {
      for (; n > 0; --n)
         ++i;
   }
}
```

constexpr-if (Listing 14)

```cpp
namespace ranges = std::experimental::ranges;

template <template <typename...> typename C, typename T>
C<T> from_file(const std::string& path)
{
   if (auto in = std::ifstream{path}) {
      const ranges::SignedIntegral size = [&in]{
         ranges::SignedIntegral i = 0;
         in >> i;
         return i;
      }();

      auto c = [size]{
         auto c = C<T>{};
         if constexpr (std::is_same_v<C<T>, std::vector<T>>)
            c.reserve(size);
         return c;
      }();
      // ...

      return c;
   }
}
```

constexpr bool objects

```
auto v = std::vector<int>{}; static_assert(ranges::Regular<decltype(v)>());
```

constexpr-if (Listing 14)

```
for (auto i : v) {
  static_assert(ranges::Regular<decltype(i)>());
  // ...
}
```

```
for (const auto& i : v) {

static_assert(ranges::Regular<std::remove_const_t<std::remove_reference_t<decltype(
i)>>>());
  // ...
}
```

```
for (const Regular& i : v) {
  // ...
}
```

constexpr bool objects

```
auto v = std::vector<int>{}; static_assert(ranges::Regular<decltype(v)>());
```

constexpr-if (Listing 14)

```
for (auto i : v) {
  static_assert(ranges::Regular<decltype(i)>());
  // ...
}
```

```
for (const auto& i : v) {

static_assert(ranges::Regular<std::remove_const_t<std::remove_reference_t<decltype(
i)>>>());
  // ...
}
```

```
for (const Regular& i : v) {
  // ...
}
```

Concepts make C++ better

What is C++20?

- What’s the elevator pitch?
  - We ***must*** have an answer
- It’s a major release
  - Like C++98 and C++11
  - Not minor like C++03 and C++14
  - Not Medium like C++17
- If we deliver nothing major
  - The C++ community will be disappointed and angry
  - Other languages will benefit
- We must ship something coherent
  - A simple list of features is not good enough

Conclusion

C++17 will change the way we write C++ code, just as C++11 and C++14 did. For example, string_view and optional are expected to be heavily used in writing interfaces. And with parallel STL often you can just add *std::par* or *std::par_vec*, and your algorithm will speed up by a factor of 2-4 on ordinary hardware; we had a compelling story with C++11 move semantics where we could say “just recompile your code and it’ll often be noticeably faster,” and this is likely to be an even bigger improvement.

Codeplay

Standards bodies

- HSA Foundation: Chair of software group, spec editor of runtime and debugging
- Khronos: chair & spec editor of SYCL. Contributors to OpenCL, Safety Critical, Vulkan
- ISO C++: Chair of Low Latency, Embedded WG; Editor of SG1 Concurrency TS
- EEMBC: members

Research

- Members of EU research consortiums: PEPPHER, LPGPU, LPGPU2, CARP
- Sponsorship of PhDs and EngDs for heterogeneous programming: HSA, FPGAs, ray-tracing
- Collaborations with academics
- Members of HiPEAC

Open source

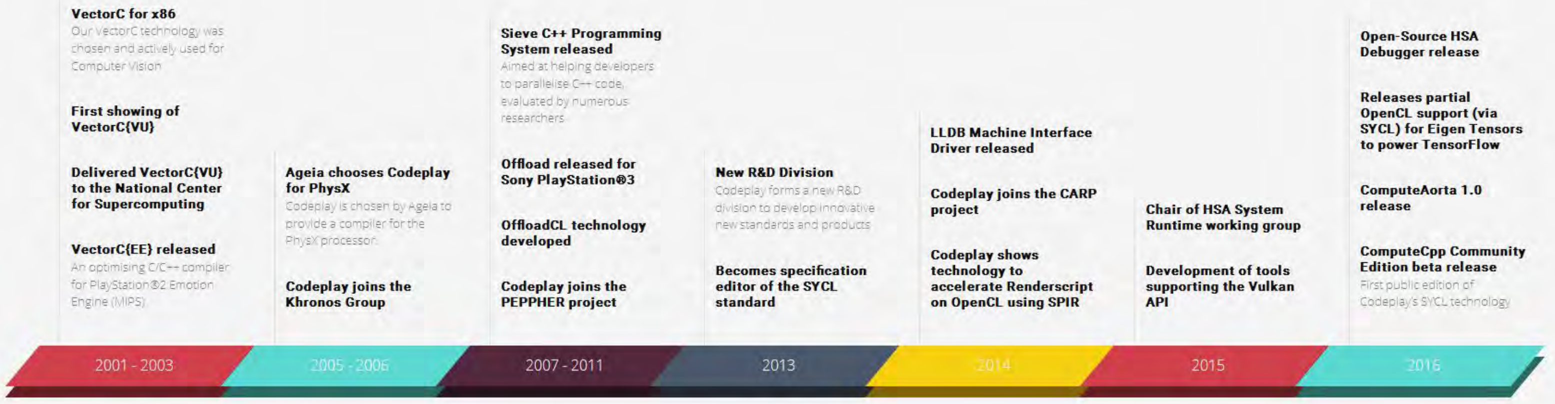
- HSA LLDB Debugger
- SPIR-V tools
- RenderScript debugger in AOSP
- LLDB for Qualcomm Hexagon
- TensorFlow for OpenCL
- C++ 17 Parallel STL for SYCL
- VisionCpp: C++ performance-portable programming model for vision

Presentations

- Building an LLVM back-end
- Creating an SPMD Vectorizer for OpenCL with LLVM
- Challenges of Mixed-Width Vector Code Gen & Scheduling in LLVM
- C++ on Accelerators: Supporting Single-Source SYCL and HSA
- LLDB Tutorial: Adding debugger support for your target

Company

- Based in Edinburgh, Scotland
- 57 staff, mostly engineering
- License and customize technologies for semiconductor companies
- ComputeAorta and ComputeCpp: implementations of OpenCL, Vulkan and SYCL
- 15+ years of experience in heterogeneous systems tools



2001 - 2003

VectorC for x86
Our VectorC technology was chosen and actively used for Computer Vision

First showing of VectorC{VU}

Delivered VectorC{VU} to the National Center for Supercomputing

VectorC{EE} released
An optimising C/C++ compiler for PlayStation®2 Emotion Engine (MIPS)

2005 - 2006

Ageia chooses Codeplay for PhysX
Codeplay is chosen by Ageia to provide a compiler for the PhysX processor

Codeplay joins the Khronos Group

2007 - 2011

Sieve C++ Programming System released
Aimed at helping developers to parallelise C++ code, evaluated by numerous researchers

Offload released for Sony PlayStation®3

OffloadCL technology developed

Codeplay joins the PEPPHER project

2013

New R&D Division
Codeplay forms a new R&D division to develop innovative new standards and products

Becomes specification editor of the SYCL standard

2014

LLDB Machine Interface Driver released

Codeplay joins the CARP project

Codeplay shows technology to accelerate Renderscript on OpenCL using SPIR

2015

Chair of HSA System Runtime working group

Development of tools supporting the Vulkan API

2016

Open-Source HSA Debugger release

Releases partial OpenCL support (via SYCL) for Eigen Tensors to power TensorFlow

ComputeAorta 1.0 release

ComputeCpp Community Edition beta release
First public edition of Codeplay's SYCL technology

Codeplay build the software platforms that deliver massive performance

What our ComputeCpp users say about us

Benoit Steiner – Google TensorFlow engineer

“We at Google have been working closely with Luke and his Codeplay colleagues on this project for almost 12 months now. Codeplay's contribution to this effort has been tremendous, so we felt that we should let them take the lead when it comes down to communicating updates related to OpenCL. … we are planning to merge the work that has been done so far… we want to put together a comprehensive test infrastructure”

ONERA

“We work with royalty-free SYCL because it is hardware vendor agnostic, single-source C++ programming model without platform specific keywords. This will allow us to easily work with any heterogeneous processor solutions using OpenCL to develop our complex algorithms and ensure future compatibility”

Hartmut Kaiser -HPX

“My team and I are working with Codeplay's ComputeCpp for almost a year now and they have resolved every issue in a timely manner, while demonstrating that this technology can work with the most complex C++ template code. I am happy to say that the combination of Codeplay's SYCL implementation with our HPX runtime system has turned out to be a very capable basis for Building a Heterogeneous Computing Model for the C++ Standard using high-level abstractions.”

WIGNER Research Centre for Physics

It was a great pleasure this week for us, that Codeplay released the ComputeCpp project for the wider audience. We've been waiting for this moment and keeping our colleagues and students in constant rally and excitement. We'd like to build on this opportunity to increase the awareness of this technology by providing sample codes and talks to potential users. We're going to give a lecture series on modern scientific programming providing field specific examples.“

Further information

OpenCL	https://www.khronos.org/opencl/
OpenVX	https://www.khronos.org/openvx/
HSA	http://www.hsafoundation.com/
SYCL	http://sycl.tech
OpenCV	http://opencv.org/
Halide	http://halide-lang.org/
VisionCpp	https://github.com/codeplaysoftware/visioncpp

SYCL™

ComputeCpp™

Community Edition

Available now for free!

Visit:

computecpp.codeplay.com

- Open source SYCL projects:
  - ComputeCpp SDK - Collection of sample code and integration tools
  - SYCL ParallelSTL – SYCL based implementation of the parallel algorithms
  - VisionCpp – Compile-time embedded DSL for image processing
  - Eigen C++ Template Library – Compile-time library for machine learning

All of this and more at: http://sycl.tech

Questions ?

2017 CPP-Summit

DynaMix: A New Take on Polymorphism

Borislav Stanimirov

Video game programmer

Hello, World

```
#include <iostream>

int main()
{
    std::cout << "你好 I'm Borislav.";
    return 0;
}
```

Borislav Stanimirov

- Mostly a C++ programmer
- Mostly a game programmer since 2006
- Open-source programmer
- Currently employed at Chobolabs

DynaMix: A New Take on Polymorphism

DynaMix: A New Take on Polymorphism

OOP and Polymorphism

- OOP has come to imply dynamic polymorphism
    - Dynamic polymorphism is when the compiler can see a function call but can't know which actual piece of code will be executed next
    - It's in the category of things which are slower and can't have good compilation errors
- Totally anti modern C++
- OOP has been criticized a lot
- OOP can be useful for business logic
- People forget that C++ is an OOP language
- Out of the box in an OOP context C++ only gives us virtual functions for polymorphism

C++ and Business Logic

- Is C++ is a bad choice for business logic?
- Many projects have chosen other languages: Lua, Python, JavaScript, Ruby...
  - C++ has poor OOP capabilities
  - You can hotswap
  - You can delegate to non-programmers
- However:
  - The code is slower
  - There is more complexity in the binding layer
  - There are duplicated functionalities (which means duplicated bugs)

DynaMix: A New Take on Polymorphism

DynaMix: A New Take on Polymorphism

Polymorphism in Modern C++

- Polymorphic type-erasure wrappers
  - Boost.TypeErasure, Dyno, Folly.Poly

```cpp
using Drawable = Library_Specific_Magic;
struct Square {
    void draw(std::ostream& out) const { out << "Square\n"; }
};
struct Circle {
    void draw(std::ostream& out) const { out << "Circle\n"; }
};
void f(const Drawable& d) {
    d.draw(std::cout);
}
int main() {
    f(Square{});
    f(Circle{});
}
```

Polymorphic Wrappers

- Better than classic virtual functions
  - Information hiding (PIMPL)
  - Non-intrusive
  - More extensible
  - Potentially faster

- … but more or less the same
  - Interface types
  - Implementation types
  - Basically improved virtual functions
  - Don’t seem compelling enough to ditch scripting languages

Other C++ Polymorphism

- Signals/slots (Multicasts)
  - Very popular
  - Especially in GUI libraries (say Qt)
  - Boost.Signals2, FastDelegate, ...
- Multiple dispatch
  - `collide(obj1, obj2);`
  - Obscure feature
  - Relatively easy to mimic
  - Folly.Poly, yomm11
- Functional programming libraries

DynaMix: A New Take on Polymorphism

DynaMix: A New Take on Polymorphism

DynaMix

- Open source, MIT license, C++ library
  - github.com/iboB/dynamix
- This talk is an introduction to the library
  - Focus on the what and why
  - Hardly even mention the "how"
  - There will also be a small demo
- History
  - 2007: Interface. Zahary Karadjov
  - 2013: Rebirth as Boost.Mixin
  - 2016: Bye, Boost. Hello, DynaMix

Earthrise

Epic Pirate Story 2

War Planet Online