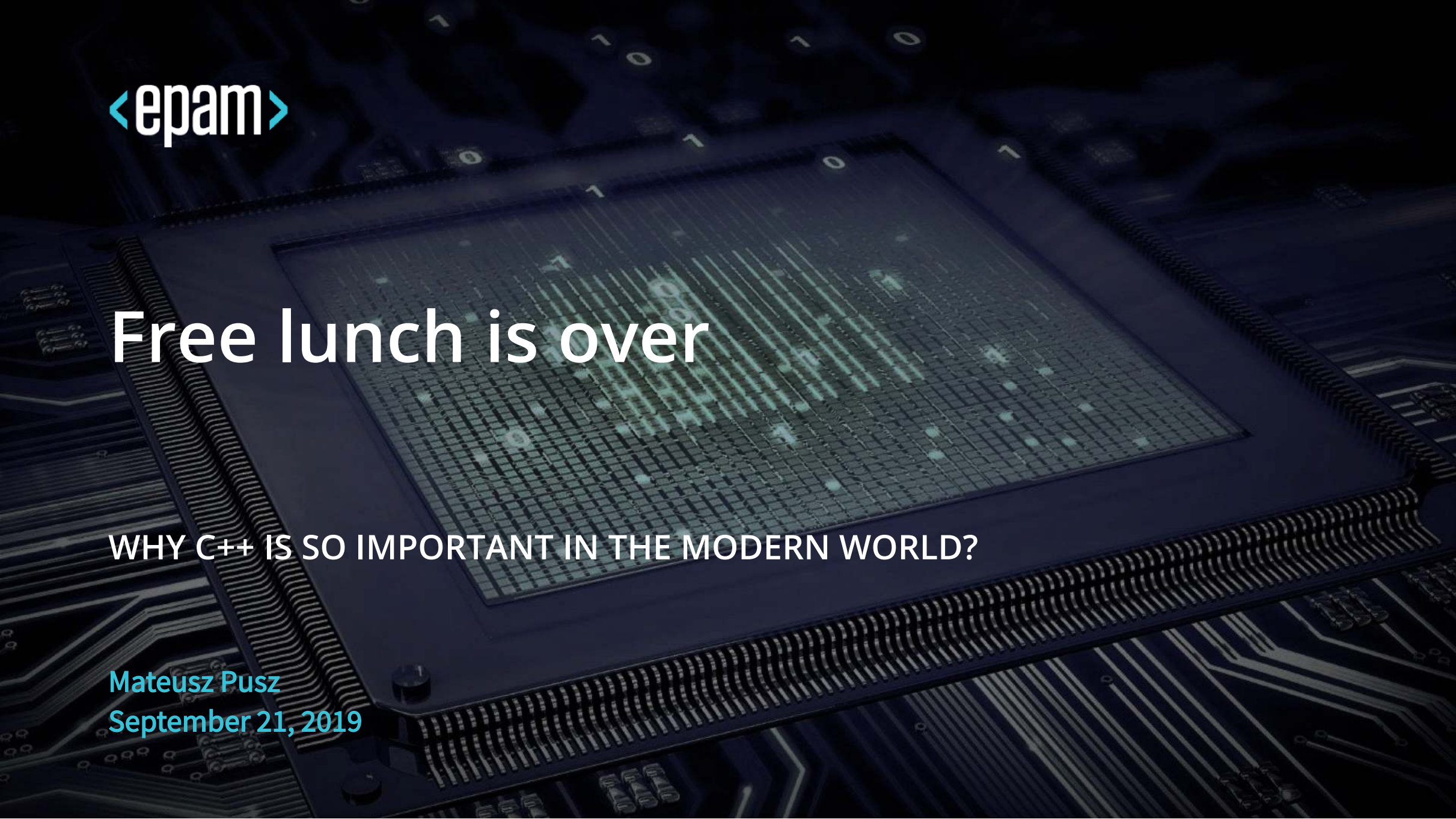


Free lunch is over

WHY C++ IS SO IMPORTANT IN THE MODERN WORLD?

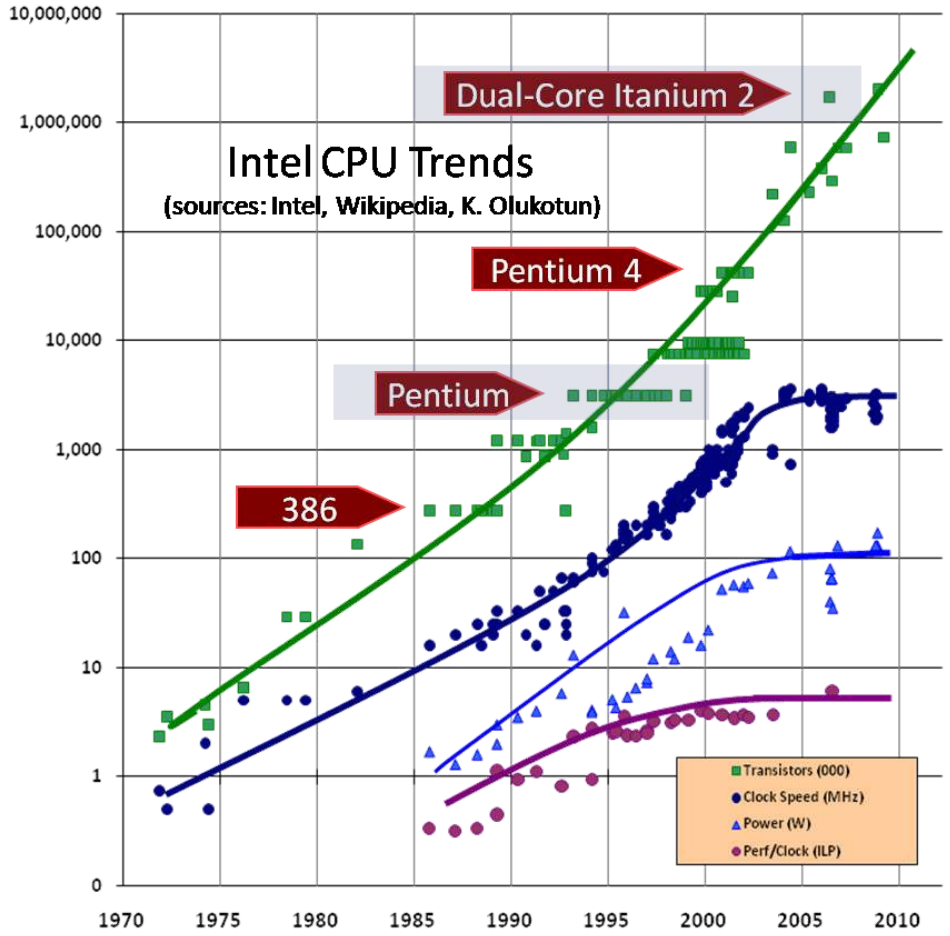
Mateusz Pusz
September 21, 2019

Confidential Information

WHY C++?

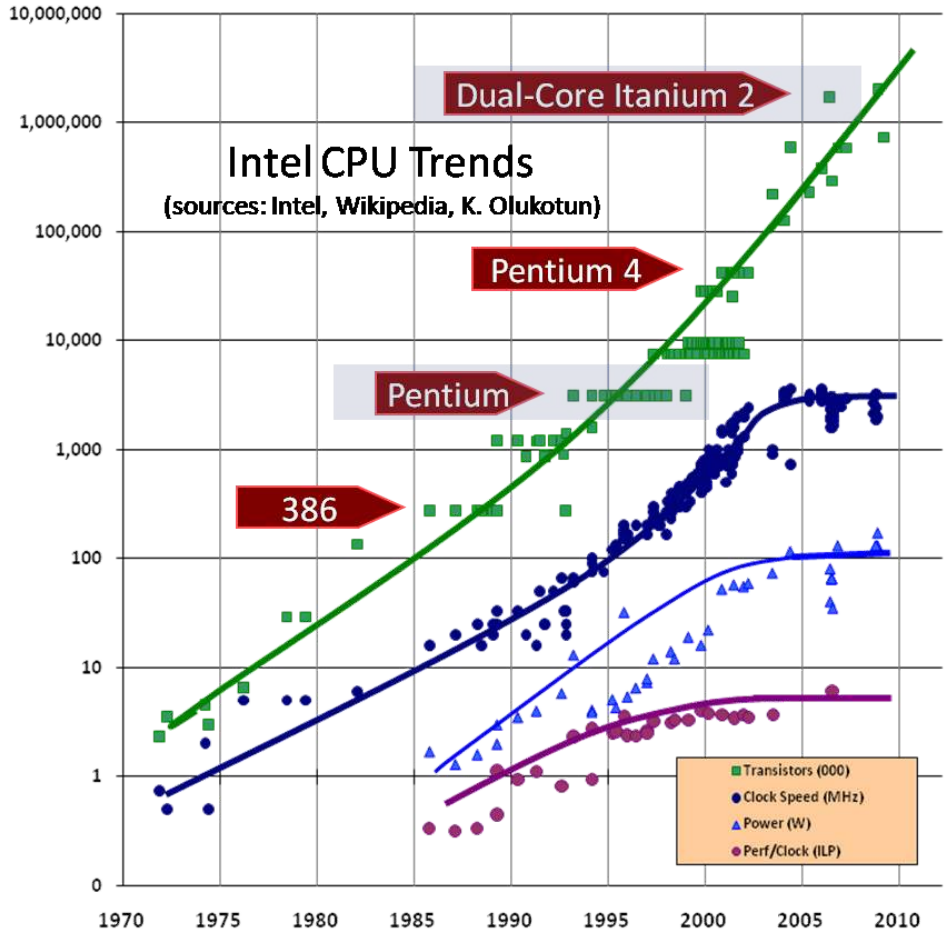
Herb Sutter 2005: Free Lunch is Over

- No matter how fast processors get, *software consistently finds new ways to eat up the extra speed*



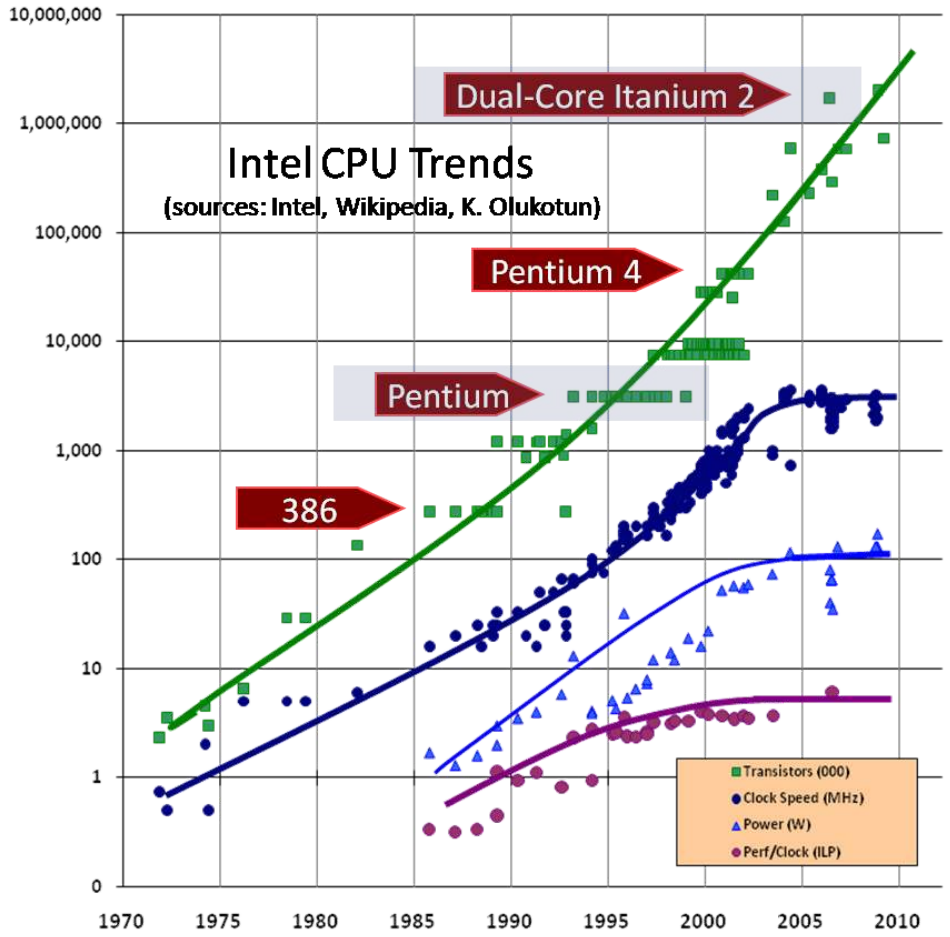
Herb Sutter 2005: Free Lunch is Over

- No matter how fast processors get, *software consistently finds new ways to eat up the extra speed*
- Applications have enjoyed *free and regular performance gains for several decades*
  - even without releasing new versions



Herb Sutter 2005: Free Lunch is Over

- No matter how fast processors get, *software consistently finds new ways to eat up the extra speed*
- Applications have enjoyed *free and regular performance gains for several decades*
  - even without releasing new versions
- *Around 2003 we run out of room with approaches to boost CPU clocks*
  - heat (too much and hard to dissipate)
  - power consumption (too high)
  - leakage problems



Where the Focus Is?

LANGUAGE	EFFICIENCY	FLEXIBILITY	ABSTRACTION	PRODUCTIVITY
C	YES!	YES!	non-goal	non-goal
C++	YES!	YES!	YES!	*WIP*
JAVA, C#	at the expense of	at the expense of	YES!	YES!

Where the Focus Is?

LANGUAGE	EFFICIENCY	FLEXIBILITY	ABSTRACTION	PRODUCTIVITY
C	YES!	YES!	non-goal	non-goal
C++	YES!	YES!	YES!	*WIP*
JAVA, C#	at the expense of	at the expense of	YES!	YES!

C++ motto:

You don't pay for what you don't use

Why C++?

Why C++?

PERFORMANCE PER TRANSISTOR

- Hardware costs
- Limited resources on mobiles

Why C++?

PERFORMANCE PER TRANSISTOR

- Hardware costs
- Limited resources on mobiles

PERFORMANCE PER CLOCK

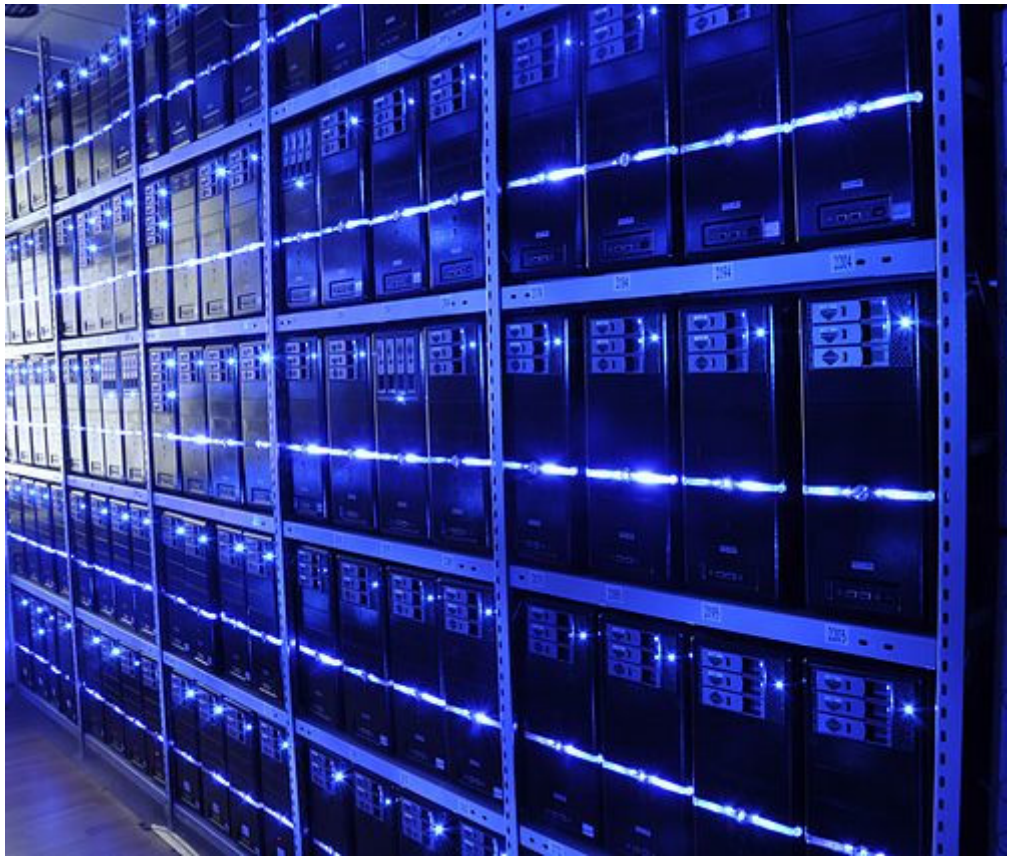
- Get the most from every CPU core
- Bigger experiences on a smaller hardware

Why C++?

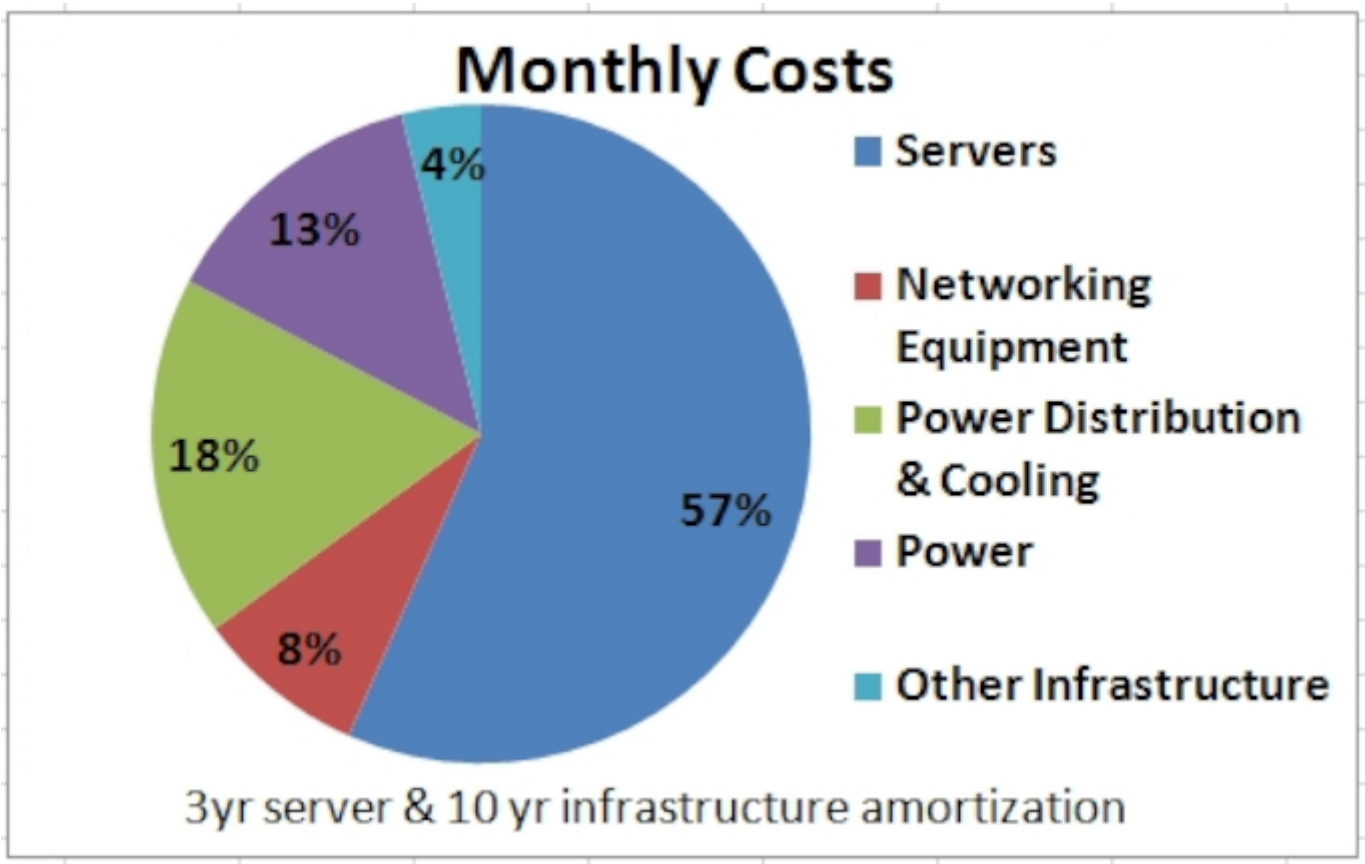
PERFORMANCE PER WAT

Why C++?

PERFORMANCE PER WAT

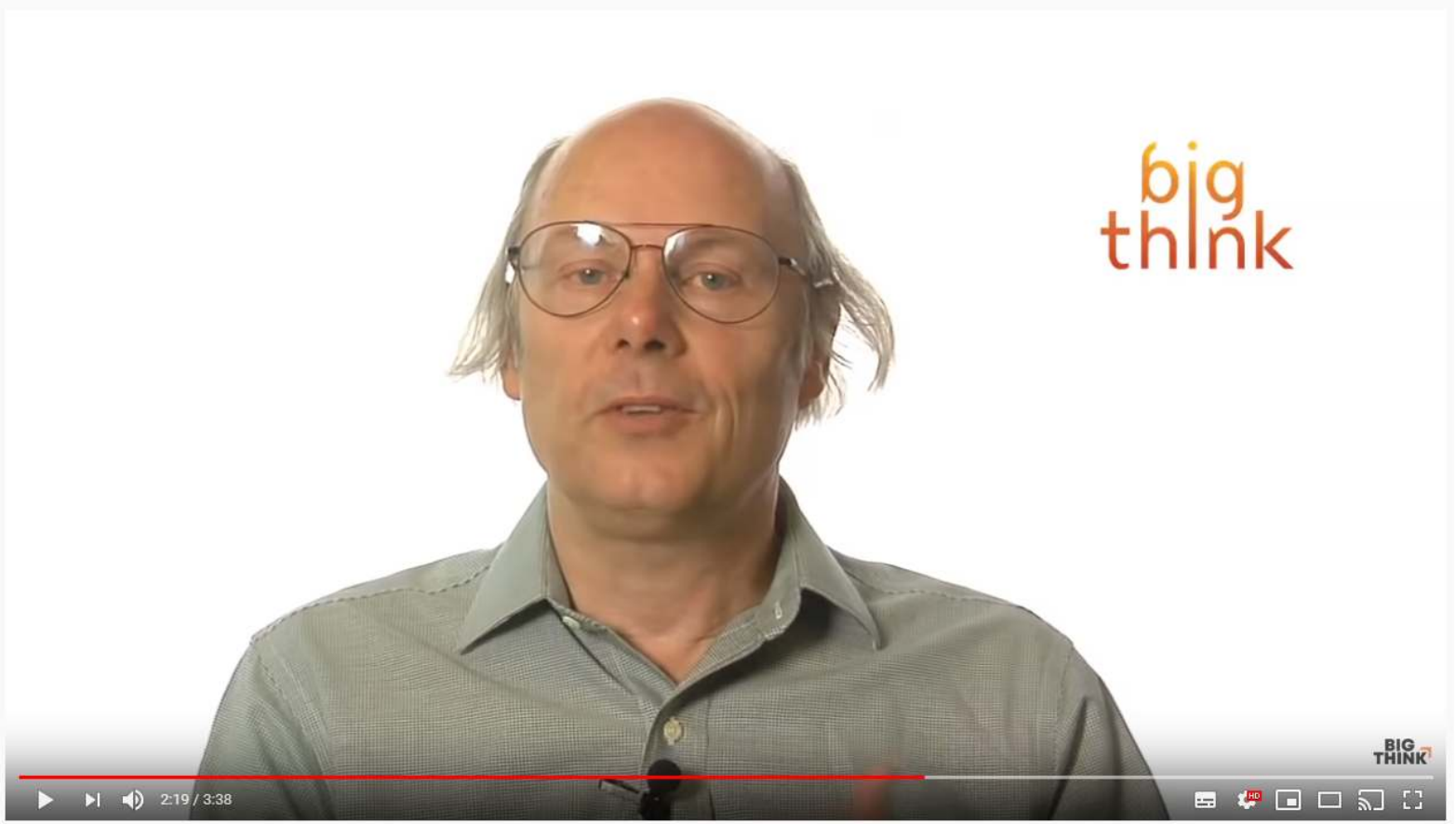


Data center costs



https://perspectives.mvdirona.com/2010/09/overall-data-center-costs

How C++ Combats Global Warming



Bjarne Stroustrup: How C++ Combats Global Warming

How C++ Combats Global Warming

I guess in some way of saying here's my contribution to dealing with global warming because **if you can double the efficiency of those systems you don't need yet another server farm** that use as much energy as a small town.

-- Bjarne Stroustrup

Focus shift

People costs shift from top to nearly irrelevant.

James Hamilton

VP & Distinguished Engineer, Amazon Web Services

Efficiency

Efficiency is not just running fast or running bigger programs, it's also **running using less resources**.

-- Bjarne Stroustrup

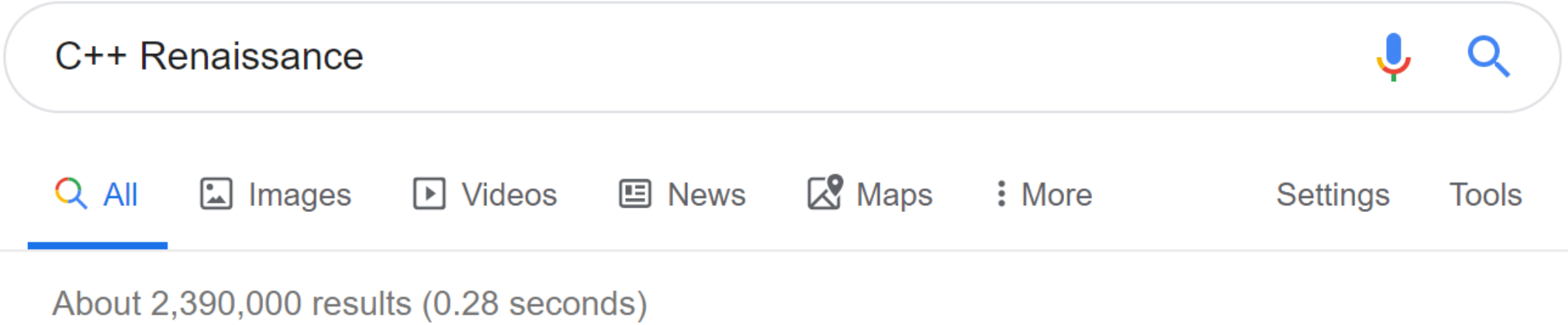
Efficiency

Efficiency is not just running fast or running bigger programs, it's also **running using less resources**.

-- Bjarne Stroustrup

Performance-oriented corporations are *willing to pay more for software development* if due to its better efficiency it will save lots of money during product lifetimes

C++ Renaissance



C++ Renaissance: Google

Monday, May 23, 2011

C++ at Google: Here Be Dragons

Google has one of the largest monolithic C++ codebases in the world. We have thousands of engineers working on millions of lines of C++ code every day.

http://google-engtools.blogspot.com/2011/05/c-at-google-here-be-dragons.html

C++ Renaissance: Microsoft

Pete Brown is a XAML and Blinky lights guy at Microsoft who focuses on Windows XAML (WinRT), WPF, Silverlight, .NET Micro Framework and other "code on the client" and "code on a device" technologies. This is his personal blog.

Now, you can actually accomplish creating a basic add-in simply by storing a little hunk of script on the drive and adding the right registry keys. However, I specifically wanted to do this in C++. No, not because I hate myself, but because I'm starting to see a resurgence of interest in C++. You can create add-ins using .NET and Script, but both have significant limitations as well as performance concerns. If you want to write an add-in of any complexity, you'll almost certainly want to write it in C++. So, that's what I decided to do.

http://10rem.net/blog/2011/02/22/creating-an-internet-explorer-add-in-toolbar-button-using-cplusplus-and-atl

C++ Renaissance: Facebook (2012)

> Facebook consumes very little Wats per user. Actually fractions of Wat. ...
> Everything that is backend, high-performance is C++ because it must be responsive and well done. ...
> Facebook is literally running on C++. We also have JIT for PHP that is obviously written in C++ and generates machine code during runtime. ...
> Using the native language such as C++ means more features for the same power and less power for the given set of features. It is a very simple equation.

-- *Andrei Alexandrescu*, Research Scientist at Facebook, Lang.Next 2012

C++ Renaissance: Facebook (Today)

It turns out Facebook is already using JIT'ed C++ code in production as their own "efficient scripting framework" for dealing with their HTTP request handling logic in their L7 reverse proxies. Their C++ scripts are compiled/linked/executed at run-time and built off the LLVM/Clang infrastructure.

Facebook engineers say their C++ scripting framework was faster than another previously used scripting language by four times. This isn't straight-up unmodified C++ sources they are JIT'ing but are relying upon pre-compiled headers and other changes to suit their workflow/requirements.

https://www.phoronix.com/scan.php?page=news_item&px=Facebook-JIT-Cpp-Scripting

C++ Renaissance: Bloomberg

Applications and Programming

How Bloomberg is advancing C++ at scale

Bloomberg Professional August 23, 2016

https://www.bloomberg.com/professional/blog/bloomberg-advancing-c-scale

WHAT IS C++?

What is C++?

What is C++?

- *C++ is no longer C with classes*

What is C++?

- *C++ is no longer C with classes*
- C++ is a **general-purpose programming language**
  - *imperative*, *object-based*, *object-oriented*, *generic*, and *functional programming* features
  - developer is not tied to any specific programming paradigm by the language

What is C++?

- *C++ is no longer C with classes*
- C++ is a **general-purpose programming language**
  - *imperative*, *object-based*, *object-oriented*, *generic*, and *functional programming* features
  - developer is not tied to any specific programming paradigm by the language
- **High-level abstractions at low cost**

What is C++?

- *C++ is no longer C with classes*
- C++ is a **general-purpose programming language**
  - *imperative*, *object-based*, *object-oriented*, *generic*, and *functional programming* features
  - developer is not tied to any specific programming paradigm by the language
- **High-level abstractions at low cost**
- **Low-level access** when needed

What is C++?

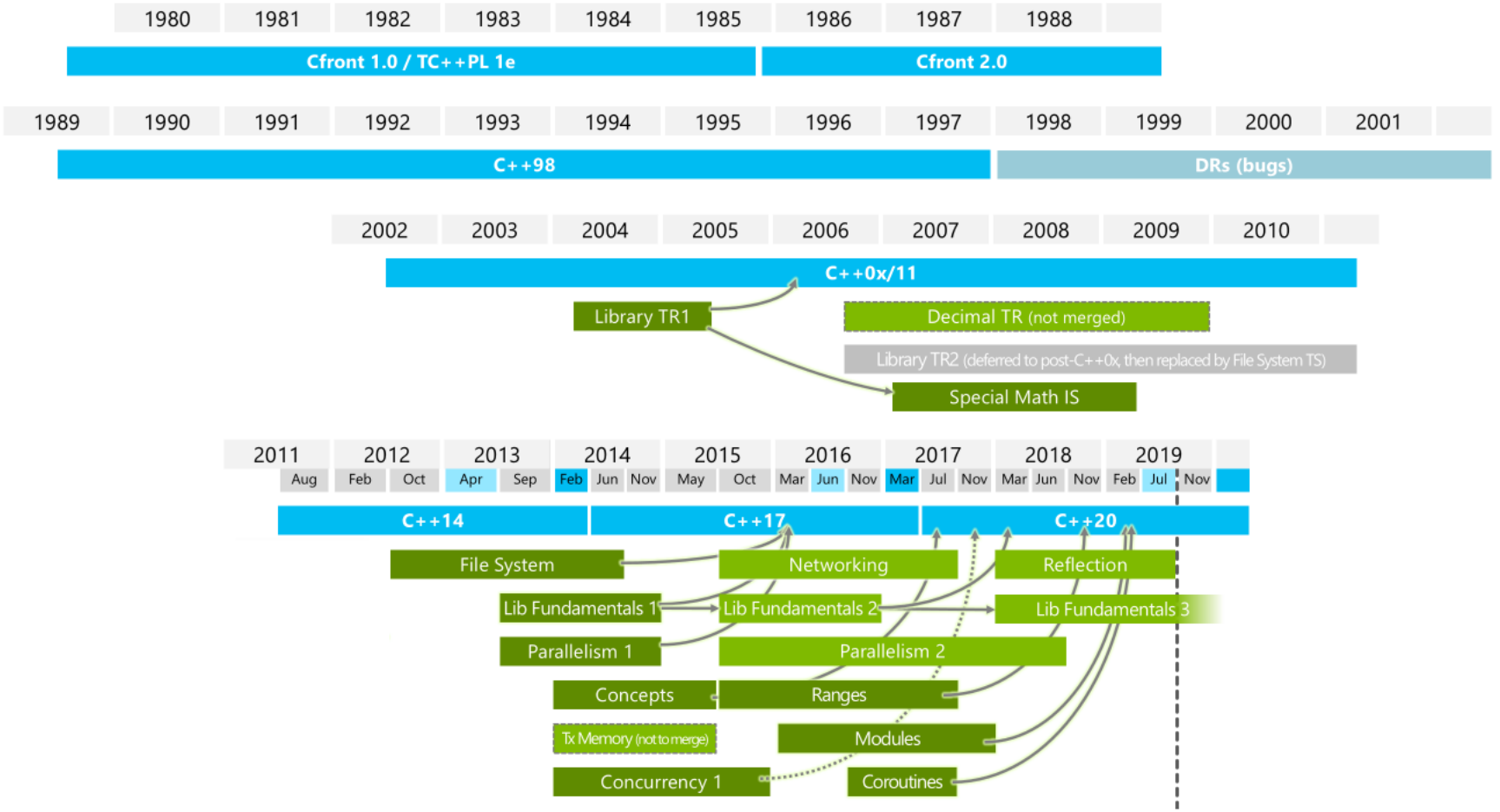
- *C++ is no longer C with classes*
- C++ is a **general-purpose programming language**
  - *imperative*, *object-based*, *object-oriented*, *generic*, and *functional programming* features
  - developer is not tied to any specific programming paradigm by the language
- **High-level abstractions at low cost**
- **Low-level access** when needed
- If used correctly, provides hard to beat **performance**

What is C++?

- *C++ is no longer C with classes*
- C++ is a **general-purpose programming language**
  - *imperative*, *object-based*, *object-oriented*, *generic*, and *functional programming* features
  - developer is not tied to any specific programming paradigm by the language
- **High-level abstractions at low cost**
- **Low-level access** when needed
- If used correctly, provides hard to beat **performance**

C++ language evolves now faster than ever

C++ Timeline


1980 1981 1982 1983 1984 1985 1986 1987 1988
Cfront 1.0 / TC++PL 1e
Cfront 2.0
1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001
C++98
DRs (bugs)
2002 2003 2004 2005 2006 2007 2008 2009 2010
C++0x/11
Library TR1
Decimal TR (not merged)
Library TR2 (deferred to post-C++0x, then replaced by File System TS)
Special Math IS
2011 2012 2013 2014 2015 2016 2017 2018 2019
Aug Feb Oct Apr Sep Feb Jun Nov May Oct Mar Jun Nov Mar Jul Nov Mar Jun Nov Feb Jul Nov
C++14
C++17
C++20
File System
Networking
Reflection
Lib Fundamentals 1
Lib Fundamentals 2
Lib Fundamentals 3
Parallelism 1
Parallelism 2
Concepts
Ranges
Tx Memory (not to merge)
Modules
Concurrency 1
Coroutines

ISO C++ Committee (WG21)

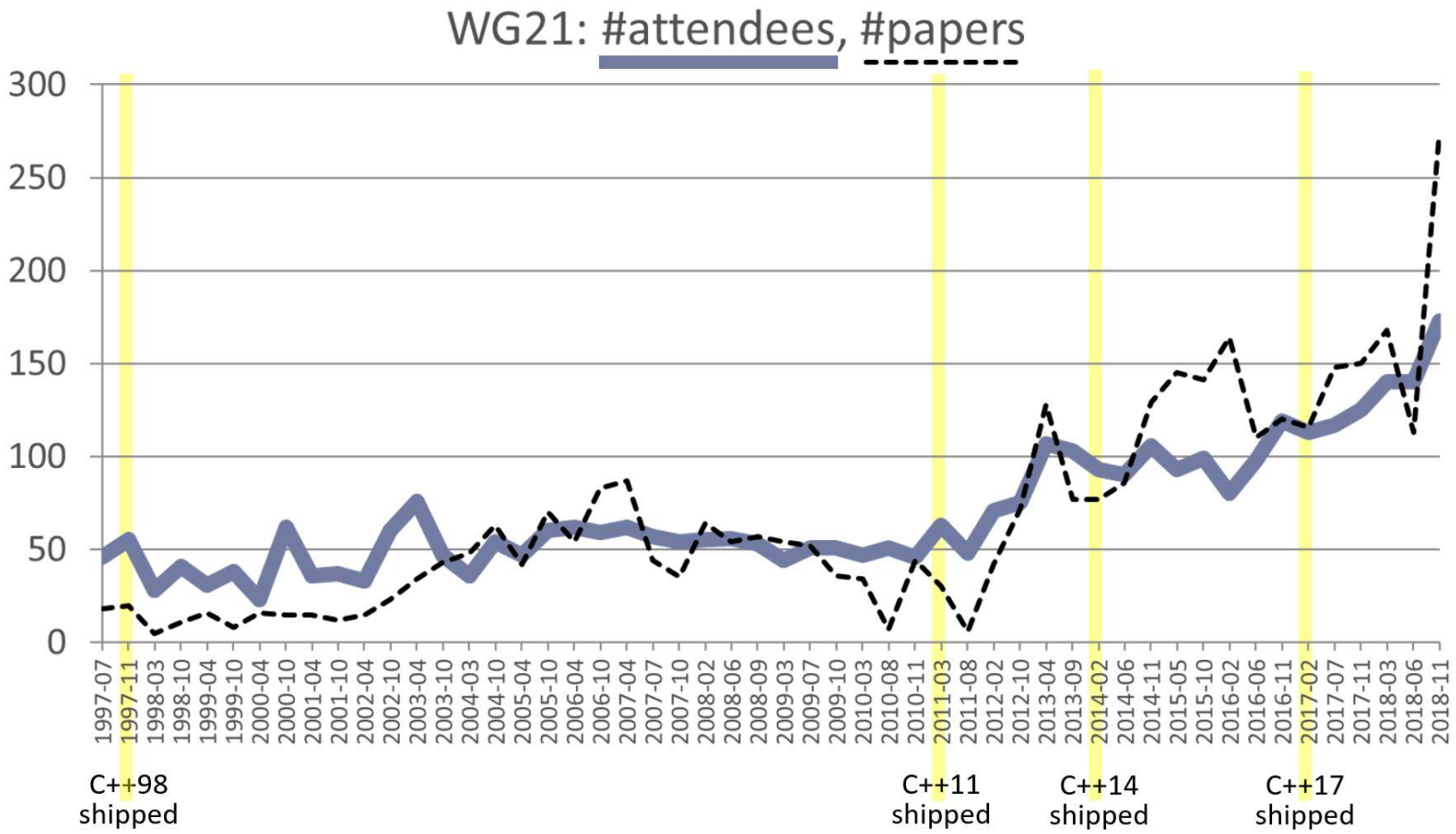
- ISO/IEC JTC1/SC22/WG21 (Joint Technical Committee 1/Subcommittee 22/Working Group 21)
- Formed in 1990-91
- Consists of accredited experts from member nations of ISO/IEC JTC1/SC22 who are interested in the C++ work
- Includes *13 member countries*: Bulgaria, Canada, Czech Republic, Finland, France, Germany, Netherlands, Poland, Russia, Spain, Switzerland, United Kingdom, and the United States
  - Bulgaria, Czech Republic, and Poland *joined during last year*

ISO C++ Committee (WG21)

- ISO/IEC JTC1/SC22/WG21 (Joint Technical Committee 1/Subcommittee 22/Working Group 21)
- Formed in 1990-91
- Consists of accredited experts from member nations of ISO/IEC JTC1/SC22 who are interested in the C++ work
- Includes *13 member countries*: Bulgaria, Canada, Czech Republic, Finland, France, Germany, Netherlands, Poland, Russia, Spain, Switzerland, United Kingdom, and the United States
  - Bulgaria, Czech Republic, and Poland *joined during last year*

EPAM joined "The Committee" in February 2017!

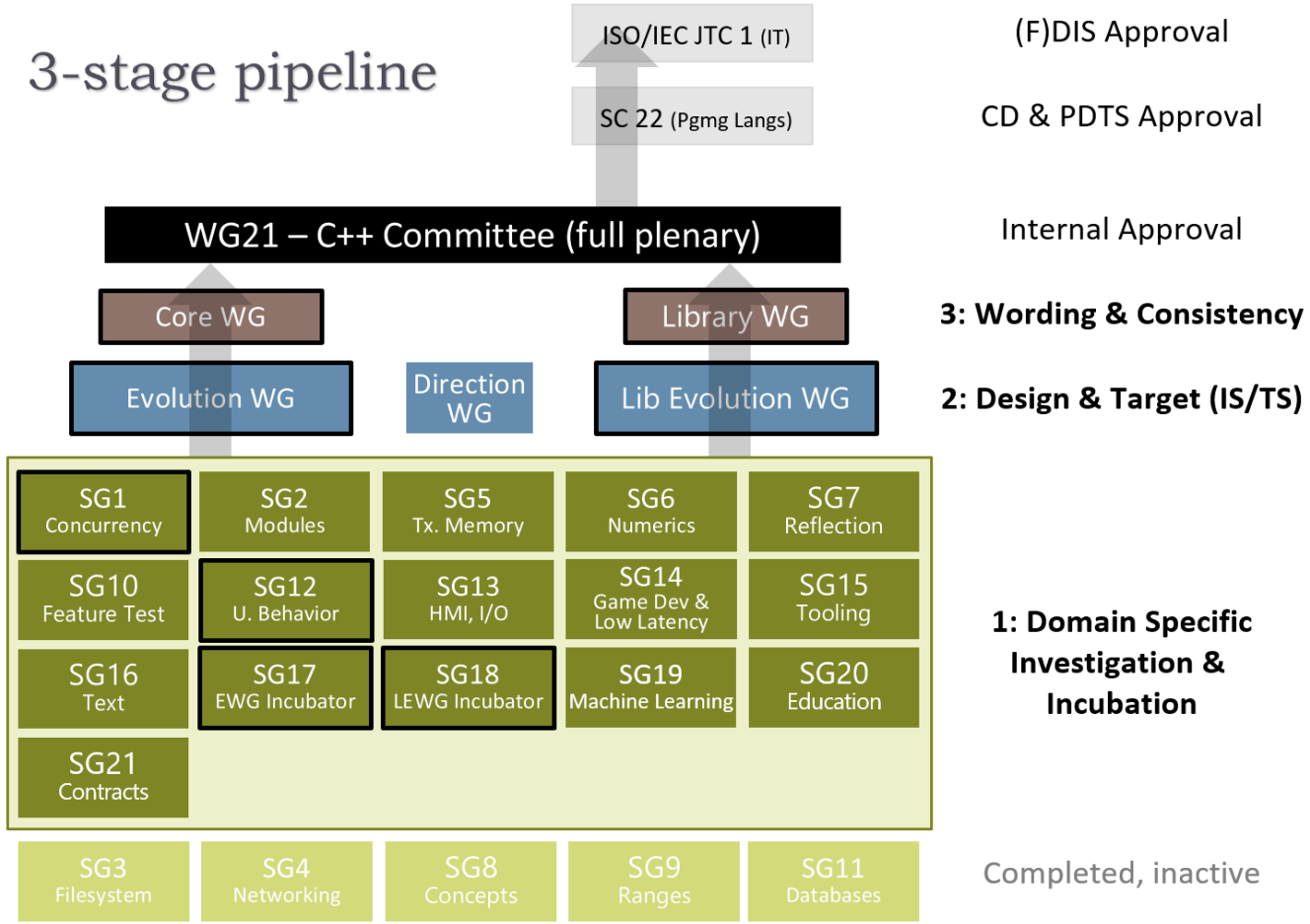
C++ Momentum


WG21: #attendees, #papers
300
250
200
150
100
50
0
1997-07
1997-11
1998-03
1998-10
1999-04
1999-10
2000-04
2000-10
2001-04
2001-10
2002-04
2002-10
2003-04
2003-10
2004-03
2004-10
2005-04
2005-10
2006-04
2006-10
2007-04
2007-07
2007-10
2008-02
2008-06
2008-09
2009-03
2009-07
2009-10
2010-03
2010-08
2010-11
2011-03
2011-08
2012-02
2012-10
2013-04
2013-09
2014-02
2014-06
2014-11
2015-05
2015-10
2016-02
2016-06
2016-11
2017-02
2017-07
2017-11
2018-03
2018-06
2018-11
C++98 shipped
C++11 shipped
C++14 shipped
C++17 shipped

Not just yet another ISO committee

SC 22 SUBGROUP	NAME	RECENT FACE-TO-FACE ATTENDANCE	NOTES
WG 4	COBOL	15	
WG 5	Fortran	26	
WG 9	Ada	0	
WG 14	C	24	
WG 17	Prolog	3	
WG 21	C++	224	
WG 23	Programming Language Vulnerabilities	3	+ *~10 WG21 experts* when meeting co-located with WG21
WG 24	Linux Standard Base	(none yet)	

ISO C++ Committee structure



Performance vs type safety and usability

Performance vs type safety and usability

JAVA

```
class Price { /* ... */ };
```

```
Price buy = new Price(10);
Price threashold = new Price(11);
if(buy.less(threashold))
  execute();
buy.incr(1);
```

In most "modern" languages (i.e. Java, C#)

user-defined types

- have to be *allocated on the heap*
- do not have *value semantics*
- do not work with *built-in operators*

Performance vs type safety and usability

JAVA

```
class Price { /* ... */ };
```

```
Price buy = new Price(10);
Price threashold = new Price(11);
if(buy.less(threashold))
  execute();
buy.incr(1);
```

In most "modern" languages (i.e. Java, C#)

user-defined types

- have to be *allocated on the heap*
- do not have *value semantics*
- do not work with *built-in operators*

C++

```
class price { /* ... */ };
```

```
price buy(10);
price threashold(11);
if(buy < threashold)
  execute();
buy += 1;
```

Modern C++

- *type safety without sacrificing performance*
- power and expressiveness of *built-in types*
- prefers *value semantics*
- *limits* dynamic allocations
- *discourages* dynamic polymorphism

Because type matters

Because type matters

C

```c
void qsort(void* base, size_t el_count, size_t el_size,
           int (*compare)(const void*, const void*));
```

Because type matters

C

```c
void qsort(void* base, size_t el_count, size_t el_size,
           int (*compare)(const void*, const void*));
```

C++

```cpp
template<class RandomIt, class Compare>
constexpr void sort(RandomIt first, RandomIt last, Compare comp);
```

Because type matters

C

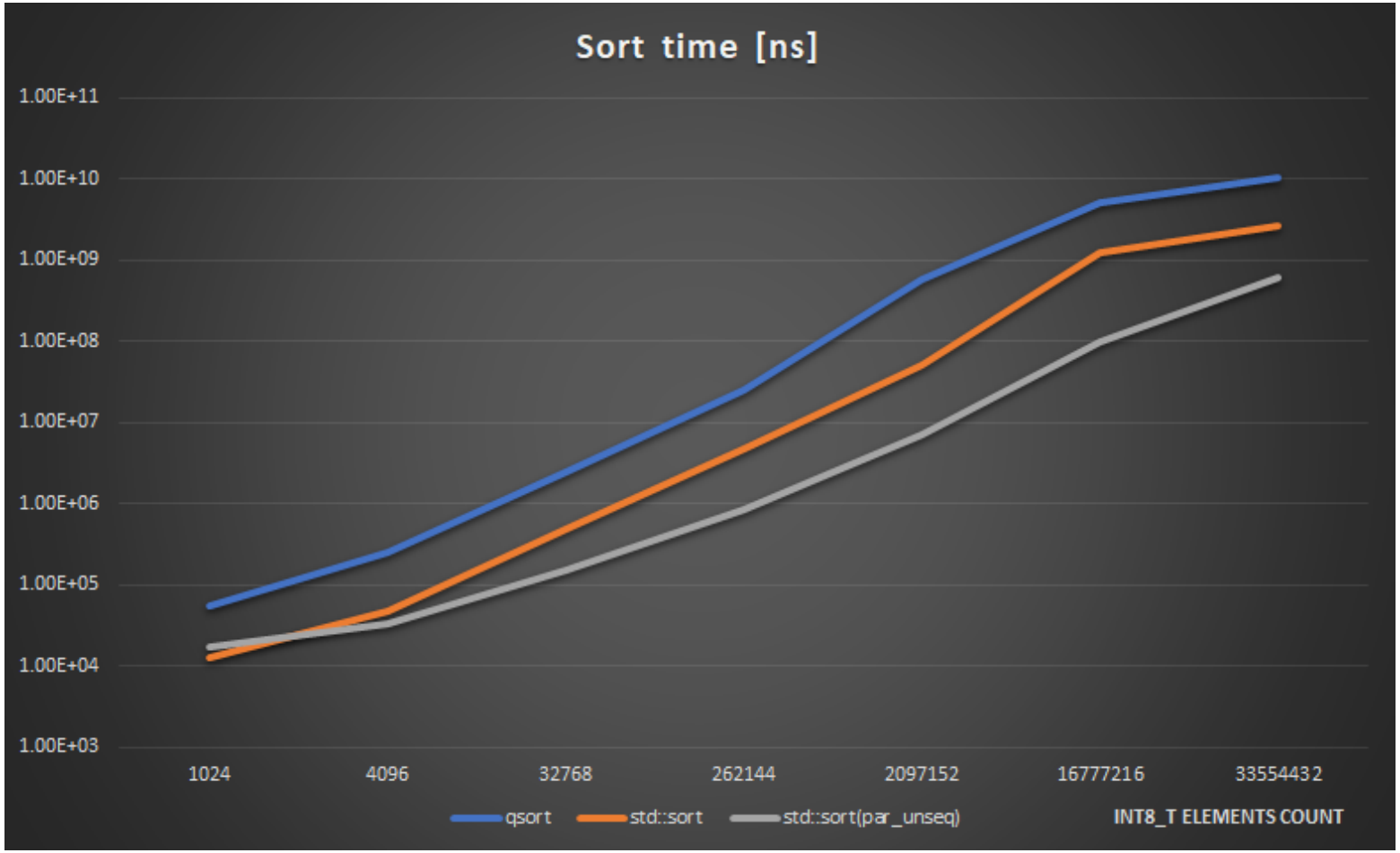
```
void qsort(void* base, size_t el_count, size_t el_size,
           int (*compare)(const void*, const void*));
```

C++

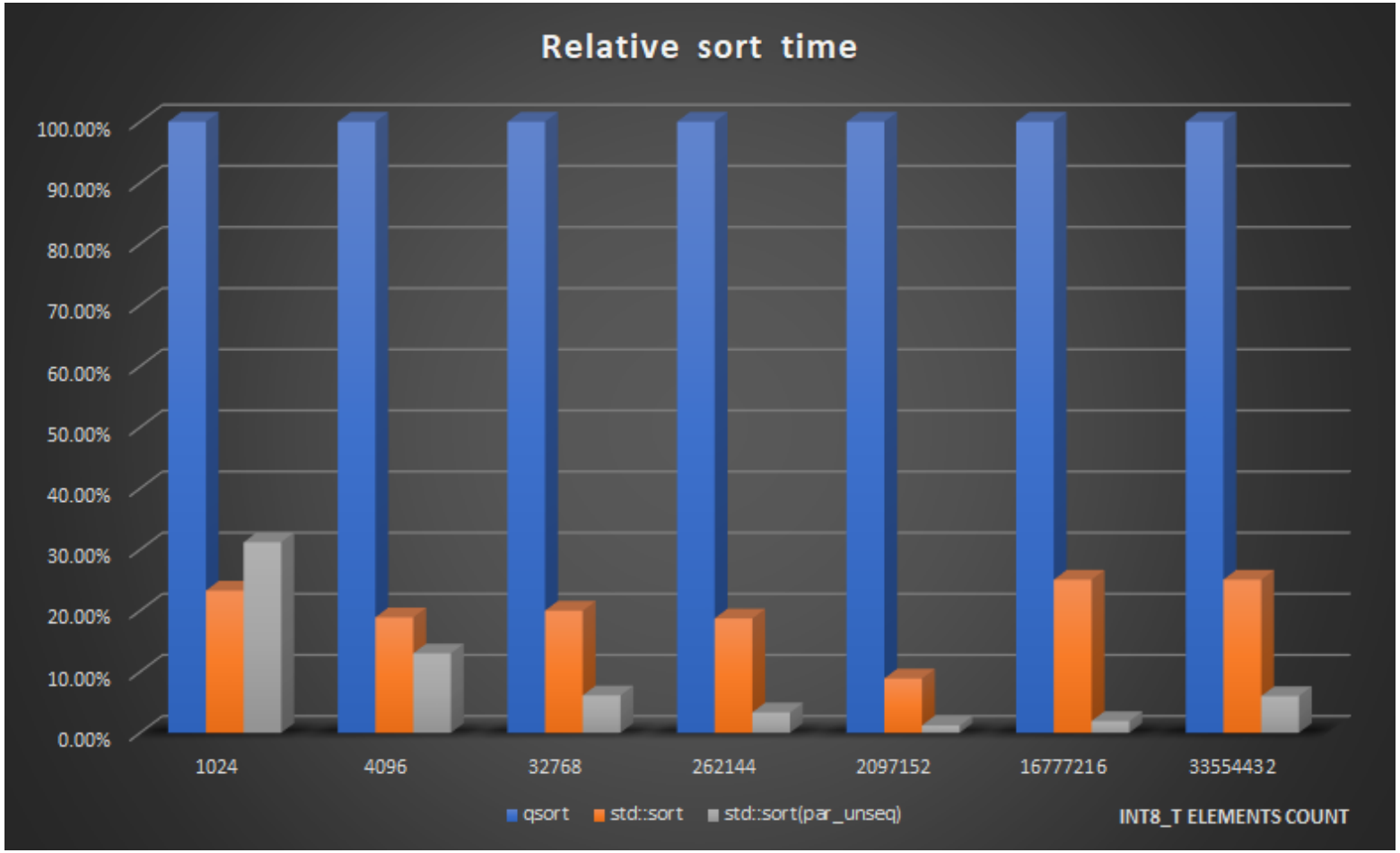
```
template<class RandomIt, class Compare>
constexpr void sort(RandomIt first, RandomIt last, Compare comp);
```

```
template<class ExecutionPolicy, class RandomIt, class Compare>
void sort(ExecutionPolicy&& policy, RandomIt first, RandomIt last, Compare comp);
```

Because type matters



Because type matters



Unified approach to resource handling

Unified approach to resource handling

MEMORY

```
{
  auto smart_ptr = std::make_unique<BigData>();
  // use ptr
}
```

Unified approach to resource handling

MEMORY

```
{
  auto smart_ptr = std::make_unique<BigData>();
  // use ptr
}
```

THREAD SYNCHRONIZATION

```
{
  auto lock = std::scoped_lock(mtx);
  // critical section
}
```

Unified approach to resource handling

MEMORY

```
{
  auto smart_ptr = std::make_unique<BigData>();
  // use ptr
}
```

FILE HANDLES

```
{
  auto file = fhandle(fopen("demo.txt", "r"));
  // use the file
}
```

THREAD SYNCHRONIZATION

```
{
  auto lock = std::scoped_lock(mtx);
  // critical section
}
```

Unified approach to resource handling

MEMORY

```
{
  auto smart_ptr = std::make_unique<BigData>();
  // use ptr
}
```

FILE HANDLES

```
{
  auto file = fhandle(fopen("demo.txt", "r"));
  // use the file
}
```

THREAD SYNCHRONIZATION

```
{
  auto lock = std::scoped_lock(mtx);
  // critical section
}
```

NETWORK SOCKET

```
{
  auto socket = connect("www.epam.com");
  // use the socket
}
```

Unified approach to resource handling

MEMORY

```
{
  auto smart_ptr = std::make_unique<BigData>();
  // use ptr
}
```

FILE HANDLES

```
{
  auto file = fhandle(fopen("demo.txt", "r"));
  // use the file
}
```

THREAD SYNCHRONIZATION

```
{
  auto lock = std::scoped_lock(mtx);
  // critical section
}
```

NETWORK SOCKET

```
{
  auto socket = connect("www.epam.com");
  // use the socket
}
```

No need for a garbage collector as memory is not the only resource to handle

Compile time safety without sacrificing performance

Compile time safety without sacrificing performance

```
constexpr double avg_speed(double length,
                           double time)
{
  return length / time;
}
```

```
/// l - distance in kilometers
/// t - time in hours
void foo(double l, double t)
{
  auto speed = avg_speed(l, t);
  int mps = int(speed * 1000 / 3600);

  printf("Average speed in mps is: %u\n",
         mps);
}
```

Compile time safety without sacrificing performance

```
constexpr double avg_speed(double length,
                           double time)
{
  return length / time;
}
```

```
constexpr Velocity auto avg_speed(Length auto d,
                                  Time auto t)
{
  return d / t;
}
```

```
/// l - distance in kilometers
/// t - time in hours
void foo(double l, double t)
{
  auto speed = avg_speed(l, t);
  int mps = int(speed * 1000 / 3600);

  printf("Average speed in mps is: %u\n",
         mps);
}
```

```
void foo(Length auto l, Time auto t)
{
  auto speed = avg_speed(l, t);
  auto mps = quantity_cast<metre_per_second,
                           int>(speed);
  printf("Average speed in mps is: %u\n",
         mps.count());
}
```

Compile time safety without sacrificing performance

```
constexpr double avg_speed(double length,
                           double time)
{
  return length / time;
}
```

```
constexpr Velocity auto avg_speed(Length auto d,
                                  Time auto t)
{
  return d / t;
}
```

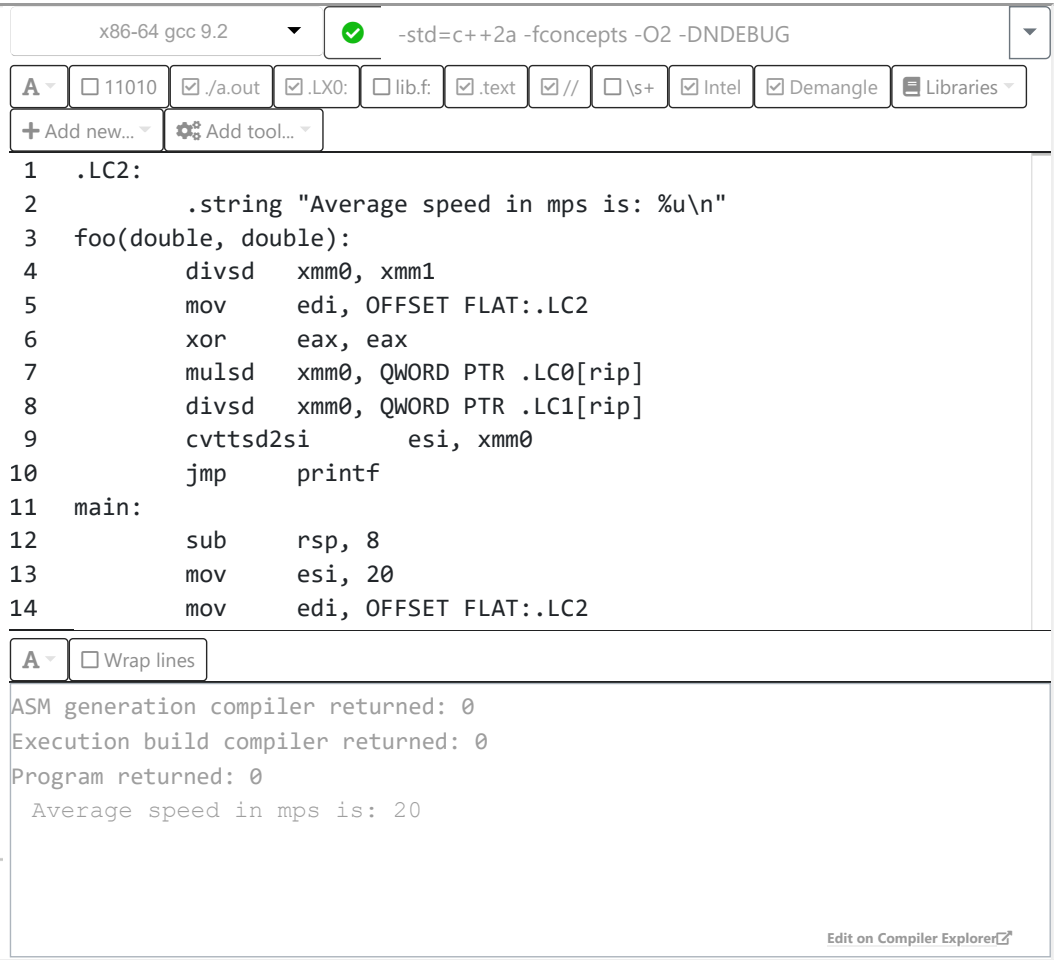
```
/// l - distance in kilometers
/// t - time in hours
void foo(double l, double t)
{
  auto speed = avg_speed(l, t);
  int mps = int(speed * 1000 / 3600);

  printf("Average speed in mps is: %u\n",
         mps);
}
```

```
void foo(Length auto l, Time auto t)
{
  auto speed = avg_speed(l, t);
  auto mps = quantity_cast<metre_per_second,
                           int>(speed);
  printf("Average speed in mps is: %u\n",
         mps.count());
}
```

How much do we pay for such advanced and safe abstractions?

Compile time safety without sacrificing performance

```cpp
#include <cstdio>




constexpr double avg_speed(double length, double time)
{
  return length / time;
}

void foo(double l, double t)
{
  auto speed = avg_speed(l, t);
  printf("Average speed in mps is: %u\n",
         int(speed * 1000 / 3600));
}

int main()
{
  foo(144., 2.);
}

```



Compile time safety without sacrificing performance

```cpp
#include <units/dimensions/velocity.h>

using namespace units;

template<Length L, Time T>
constexpr Velocity avg_speed(L length, T time)
{
  return length / time;
}

void foo(double l, double t)
{
  auto speed = avg_speed(quantity<kilometre>(l), quantity<hour>(t));
  printf("Average speed in mps is: %u\n",
         quantity_cast<metre_per_second, int>(speed).count());
}

int main()
{
  foo(144., 2.);
}

```

x86-64 gcc 9.2 | -std=c++2a -fconcepts -O2 -DNDEBUG

A | 11010 | ./a.out | .LX0: | lib.f: | .text | // | \s+ | Intel | Demangle | Libraries

Add new... | Add tool...

```
1   .LC2:
2           .string "Average speed in mps is: %u\n"
3   foo(double, double):
4           divsd   xmm0, xmm1
5           mov     edi, OFFSET FLAT:.LC2
6           xor     eax, eax
7           mulsd   xmm0, QWORD PTR .LC0[rip]
8           divsd   xmm0, QWORD PTR .LC1[rip]
9           cvttsd2si       esi, xmm0
10          jmp     printf
11  main:
12          sub     rsp, 8
13          mov     esi, 20
14          mov     edi, OFFSET FLAT:.LC2
```

A | Wrap lines

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
 Average speed in mps is: 20
```

Edit on Compiler Explorer

Compile time safety without sacrificing performance

```cpp
#include <units/dimensions/velocity.h>

using namespace units;

template<Length L, Time T>
constexpr Velocity avg_speed(L length, T time)
{
  return length / time;
}

void foo(Length l, Time t)
{
  auto speed = avg_speed(l, t);
  printf("Average speed in mps is: %u\n",
         quantity_cast<metre_per_second, int>(speed).count());
}

int main()
{
  using namespace units::literals;
  foo(144.km, 2.h);
}
```

x86-64 gcc 9.2 -std=c++2a -fconcepts -O2 -DNDEBUG

A | 11010 | ./a.out | .LX0: | lib.f: | .text | // | \s+ | Intel | Demangle | Libraries

Add new... | Add tool...

```
.LC0:
        .string "Average speed in mps is: %u\n"
main:
        sub     rsp, 8
        mov     esi, 20
        mov     edi, OFFSET FLAT:.LC0
        xor     eax, eax
        call    printf
        xor     eax, eax
        add     rsp, 8
        ret
```

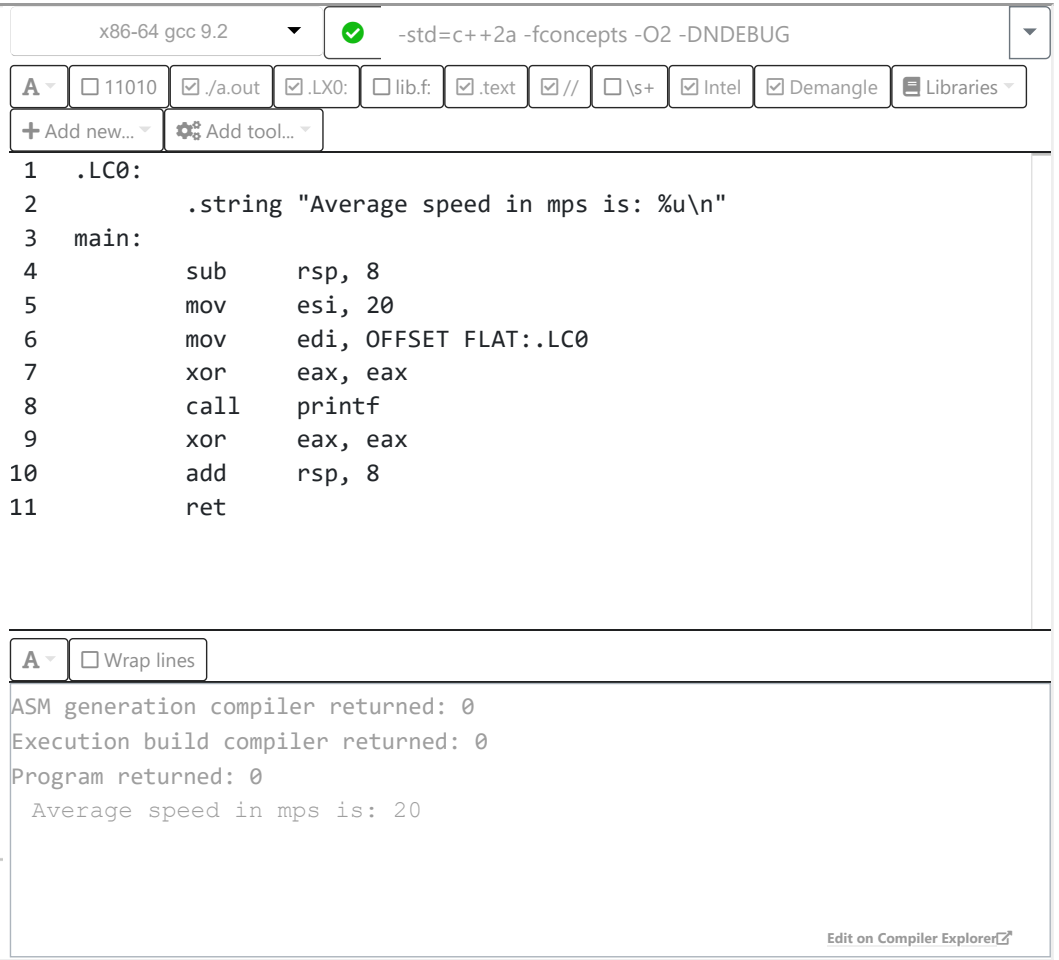
A | Wrap lines

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
 Average speed in mps is: 20
```

Edit on Compiler Explorer

Compile time safety without sacrificing performance

```
1  #include <cstdio>
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18 int main()
19 {
20   printf("Average speed in mps is: %u\n", 20);
21 }
22
```



Omnipresence of compile-time usage

Omnipresence of compile-time usage

DATE HANDLING

```
constexpr auto d = 2019_y/September/21;
static_assert(weekday{d} == Saturday);
```

Omnipresence of compile-time usage

DATE HANDLING

```
constexpr auto d = 2019_y/September/21;
static_assert(weekday{d} == Saturday);
```

```
std::cout << "'SEC 2019' starts" <<
             " on " << weekday{d} <<
             " at " << d + 9h << "\n";
```

Omnipresence of compile-time usage

DATE HANDLING

```
constexpr auto d = 2019_y/September/21;
static_assert(weekday{d} == Saturday);
```

```
std::cout << "'SEC 2019' starts" <<
             " on " << weekday{d} <<
             " at " << d + 9h << "\n";
```

```
static_assert(d == September/21/2019);
static_assert(d == 21_d/September/2019);
static_assert(d == Saturday[3]/September/2019);
```

Omnipresence of compile-time usage

DATE HANDLING

```
constexpr auto d = 2019_y/September/21;
static_assert(weekday{d} == Saturday);
```

```
std::cout << "'SEC 2019' starts" <<
             " on " << weekday{d} <<
             " at " << d + 9h << "\n";
```

```
static_assert(d == September/21/2019);
static_assert(d == 21_d/September/2019);
static_assert(d == Saturday[3]/September/2019);
```

UNITS AND DIMENSIONAL ANALYSIS

```
// simple numeric operations
static_assert(10km / 2 == 5km);

// unit conversions
static_assert(1h == 3600s);
static_assert(1km + 1m == 1001m);

// dimension conversions
static_assert(1km / 1s == 1000mps);
static_assert(2kmph * 2h == 4km);
static_assert(2km / 2kmph == 1h);
static_assert(1000 / 1s == 1kHz);
static_assert(10km / 5km == 2);
```

Omnipresence of compile-time usage

DATE HANDLING

```
constexpr auto d = 2019_y/September/21;
static_assert(weekday{d} == Saturday);
```

```
std::cout << "'SEC 2019' starts" <<
             " on " << weekday{d} <<
             " at " << d + 9h << "\n";
```

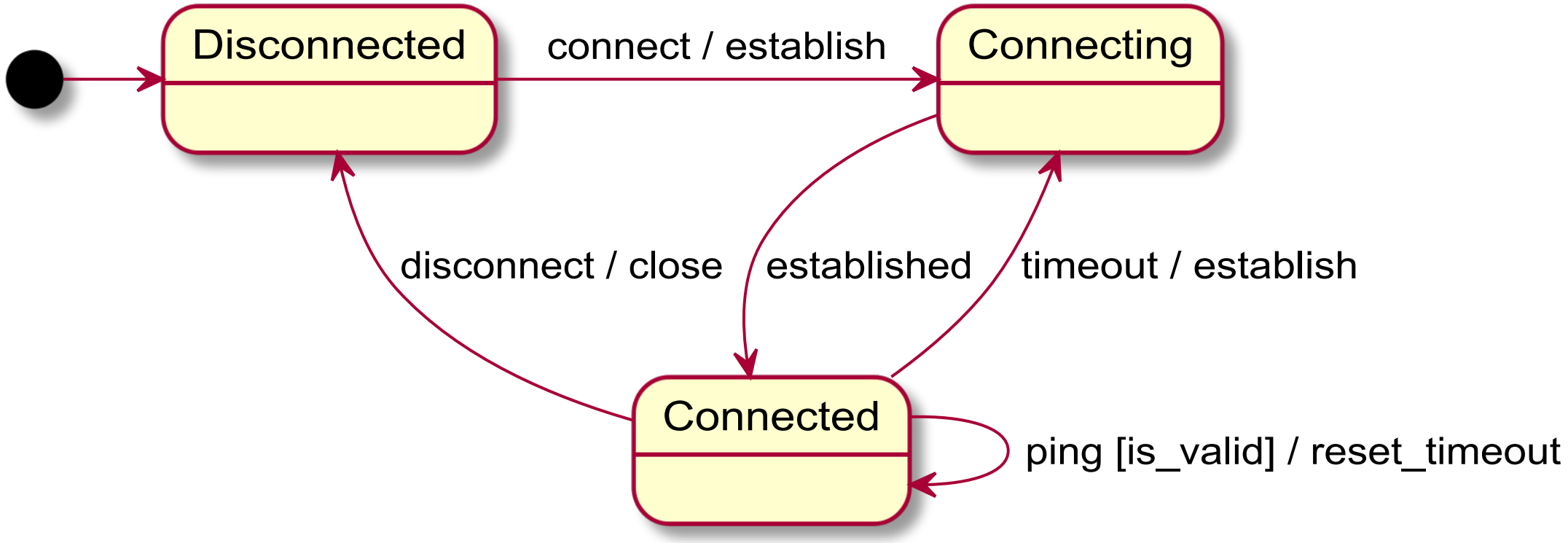
```
static_assert(d == September/21/2019);
static_assert(d == 21_d/September/2019);
static_assert(d == Saturday[3]/September/2019);
```

UNITS AND DIMENSIONAL ANALYSIS

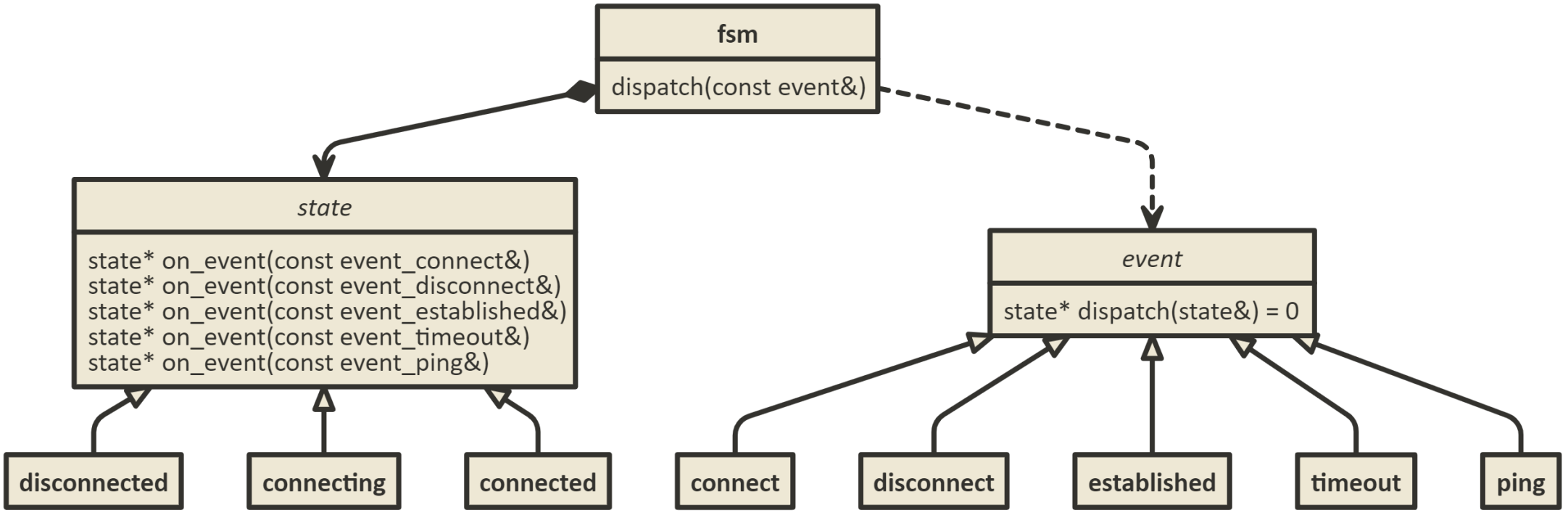
```
// simple numeric operations
static_assert(10km / 2 == 5km);

// unit conversions
static_assert(1h == 3600s);
static_assert(1km + 1m == 1001m);

// dimension conversions
static_assert(1km / 1s == 1000mps);
static_assert(2kmph * 2h == 4km);
static_assert(2km / 2kmph == 1h);
static_assert(1000 / 1s == 1kHz);
static_assert(10km / 5km == 2);
```

The fastest programs are those that do nothing

Finite State Machine (FSM)



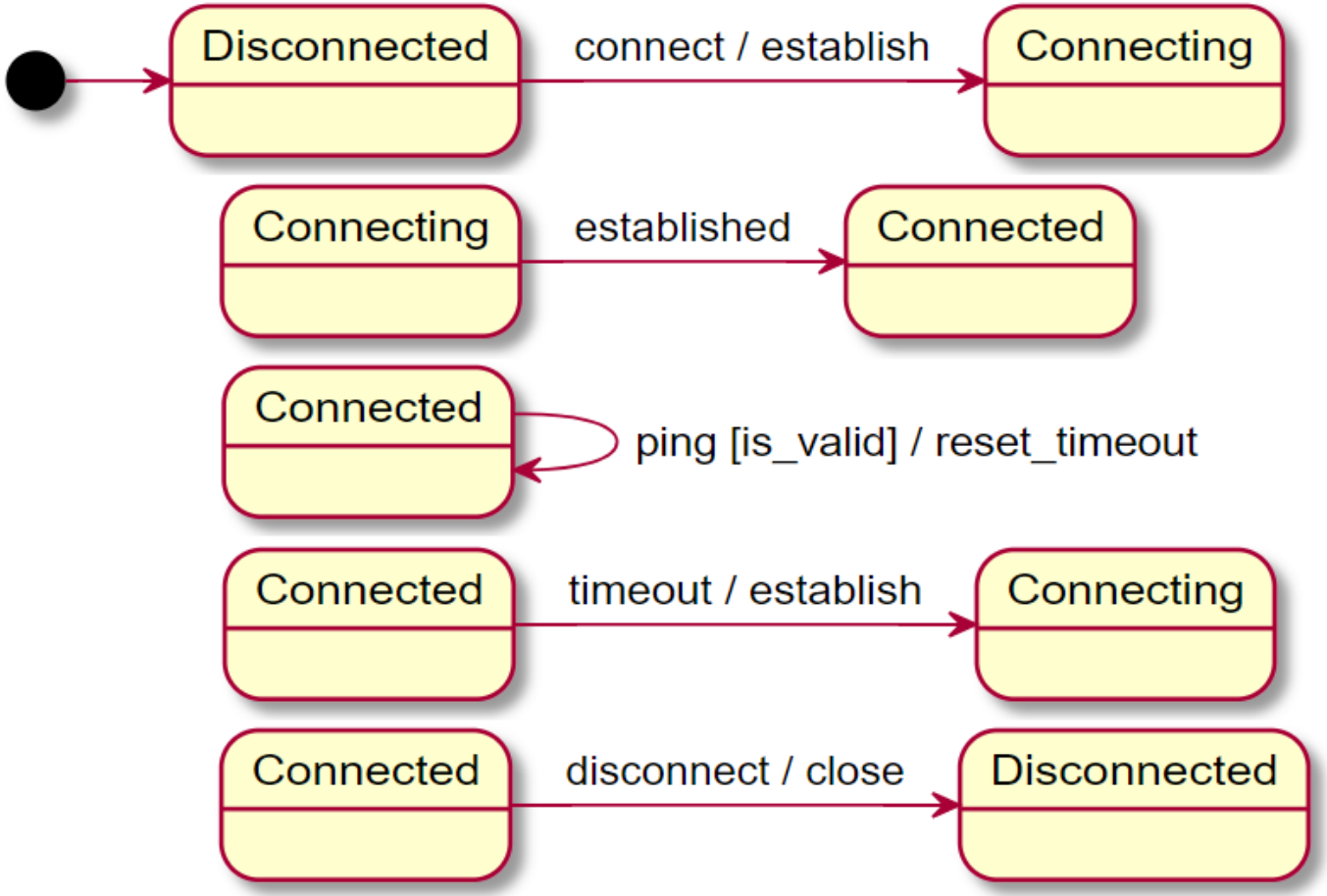
FSM: A classical approach



FSM: A naive approach

```
class fsm {
  enum class state : char { disconnected, connecting, connected } state_;
public:
  constexpr void on_event(const connect&)
  {
    if(state_ == state::disconnected) {
      establish();
      state_ = state::connecting;
    }
  }

  constexpr void on_event(const disconnect&) { /* ... */ }
  constexpr void on_event(const established&) { /* ... */ }
  constexpr void on_event(const ping& event) { /* ... */ }
  constexpr void on_event(const timeout&) { /* ... */ }
};
```

FSM: Alternative syntax



FSM: Domain Specific Language

```
struct connection {
  auto operator()() const
  {
    using namespace boost::sml;
    return make_transition_table(
      * "Disconnected"_s + event<connect> / establish    = "Connecting"_s,
        "Connecting"_s   + event<established>            = "Connected"_s,
        "Connected"_s    + event<ping> [ is_valid ] / reset_timeout,
        "Connected"_s    + event<timeout> / establish    = "Connecting"_s,
        "Connected"_s    + event<disconnect> / close     = "Disconnected"_s
    );
  }
};
```

FSM: Domain Specific Language

```
struct connection {
  auto operator()() const
  {
    using namespace boost::sml;
    return make_transition_table(
      * "Disconnected"_s + event<connect> / establish    = "Connecting"_s,
        "Connecting"_s   + event<established>            = "Connected"_s,
        "Connected"_s    + event<ping> [ is_valid ] / reset_timeout,
        "Connected"_s    + event<timeout> / establish    = "Connecting"_s,
        "Connected"_s    + event<disconnect> / close     = "Disconnected"_s
    );
  }
};
```

Program that says **WHAT** rather than **HOW** is more likely to be correct

FSM: A classical approach

```cpp
// state base class
struct state {
  virtual ~state() = default;
  virtual std::unique_ptr<state> on_event(const connect&) { return nullptr; }
  virtual std::unique_ptr<state> on_event(const ping&) { return nullptr; }
  virtual std::unique_ptr<state> on_event(const established &) { return nullptr; }
  virtual std::unique_ptr<state> on_event(const timeout&) { return nullptr; }
  virtual std::unique_ptr<state> on_event(const disconnect&) { return nullptr; }
};

// events
struct event {
  virtual ~event() = default;
  virtual std::unique_ptr<state> dispatch(state&) const = 0;
};

struct connect final : event {
  std::unique_ptr<state> dispatch(state& s) const override { return s.on_event(*this); }
};

struct established final : event {
  std::unique_ptr<state> dispatch(state& s) const override { return s.on_event(*this); }
};

struct ping final : event {
  std::unique_ptr<state> dispatch(state& s) const override { return s.on_event(*this); }
};

struct disconnect final : event {
  std::unique_ptr<state> dispatch(state& s) const override { return s.on_event(*this); }
};

struct timeout final : event {
  std::unique_ptr<state> dispatch(state& s) const override { return s.on_event(*this); }
```

x86-64 gcc (trunk) -std=c++2a -O2 -Wno-subobject-linkage

A 11010 ./a.out .LX0: lib.f: .text // \s+ Intel Demangle Libraries

Add new... Add tool...

```
        .quad   0
        .quad   typeinfo for connected
        .quad   connected::~connected() [complete object destructor]
        .quad   connected::~connected() [deleting destructor]
        .quad   state::on_event(connect const&)
        .quad   connected::on_event(ping const&)
        .quad   state::on_event(established const&)
        .quad   connected::on_event(timeout const&)
        .quad   connected::on_event(disconnect const&)
vtable for disconnected:
        .quad   0
        .quad   typeinfo for disconnected
        .quad   disconnected::~disconnected() [complete object destructor]
        .quad   disconnected::~disconnected() [deleting destructor]
        .quad   disconnected::on_event(connect const&)
        .quad   state::on_event(ping const&)
        .quad   state::on_event(established const&)
        .quad   state::on_event(timeout const&)
        .quad   state::on_event(disconnect const&)
vtable for connecting:
        .quad   0
        .quad   typeinfo for connecting
        .quad   connecting::~connecting() [complete object destructor]
        .quad   connecting::~connecting() [deleting destructor]
        .quad   state::on_event(connect const&)
        .quad   state::on_event(ping const&)
        .quad   connecting::on_event(established const&)
        .quad   state::on_event(timeout const&)
        .quad   state::on_event(disconnect const&)
```

Edit on Compiler Explorer

FSM: A naive approach

```cpp
#include <cstdio>

// actions
constexpr auto establish = []{ std::puts("establish"); };
constexpr auto close = []{ std::puts("close"); };
constexpr auto is_valid = [](auto const&) { return true; };
constexpr auto reset_timeout = []{ std::puts("reset_timeout"); };

// events
struct connect {};
struct established {};
struct ping {};
struct disconnect {};
struct timeout {};

// fsm
class fsm {
  enum class state : char { disconnected, connecting, connected } state_;
public:
  constexpr void on_event(const connect&)
  {
    if(state_ == state::disconnected) {
      establish();
      state_ = state::connecting;
    }
  }

  constexpr void on_event(const disconnect&)
  {
    if(state_ == state::connecting || state_ == state::connected) {
      close();
      state_ = state::disconnected;
    }
  }
```

x86-64 gcc (trunk) -std=c++2a -O2 -Wno-subobject-linkage

A | 11010 | ./a.out | .LX0: | lib.f: | .text | // | \s+ | Intel | Demangle | Libraries

Add new... | Add tool...

```
.LC0:
        .string "establish"
.LC1:
        .string "reset_timeout"
.LC2:
        .string "close"
main:
        sub     rsp, 8
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     edi, OFFSET FLAT:.LC1
        call    puts
        mov     edi, OFFSET FLAT:.LC2
        call    puts
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     edi, OFFSET FLAT:.LC1
        call    puts
        xor     eax, eax
        add     rsp, 8
        ret
```

Edit on Compiler Explorer

FSM: Domain Specific Language

```cpp
#include <https://raw.githubusercontent.com/boost-experimental/sml/master/include/boost/sml.hpp>
#include <cstdio>

// actions
constexpr auto establish = []{ std::puts("establish"); };
constexpr auto close = []{ std::puts("close"); };
constexpr auto is_valid = [](auto const&) { return true; };
constexpr auto reset_timeout = []{ std::puts("reset_timeout"); };

// states
struct connect {};
struct established {};
struct ping {};
struct disconnect {};
struct timeout {};

// fsm
struct connection {
  auto operator()() const
  {
    using namespace boost::sml;
    return make_transition_table(
      * "Disconnected"_s + event<connect> / establish               = "Connecting"_s,
        "Connecting"_s   + event<established>                       = "Connected"_s,
        "Connected"_s    + event<ping> [ is_valid ] / reset_timeout,
        "Connected"_s    + event<timeout> / establish               = "Connecting"_s,
        "Connected"_s    + event<disconnect> / close                = "Disconnected"_s
    );
  }
};

int main()
{
  boost::sml::sm<connection> connection{};
```

x86-64 gcc (trunk) -std=c++2a -O2 -Wno-subobject-linkage

A | 11010 | ./a.out | .LX0: | lib.f: | .text | // | \s+ | Intel | Demangle | Libraries

Add new... | Add tool...

```
.LC0:
        .string "establish"
.LC1:
        .string "reset_timeout"
.LC2:
        .string "close"
main:
        sub     rsp, 8
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     edi, OFFSET FLAT:.LC1
        call    puts
        mov     edi, OFFSET FLAT:.LC2
        call    puts
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     edi, OFFSET FLAT:.LC1
        call    puts
        xor     eax, eax
        add     rsp, 8
        ret
```

Edit on Compiler Explorer

FSM: Summary

	CLASSICAL	NAIVE	DSL
HEAP USAGE	Lots	None	None
DYNAMIC POLYMORPHISM	Yes	No	No
INLINING	No	Yes	Yes
CPU CACHE FRIENDLY	No	Yes	Yes
ERROR PRONE	Yes	Yes	No
REUSE	Moderate	Hard	Easy
ASSEMBLY LINES	350	21	21
MEMORY FOOTPRINT	High	1B	1B
PERFORMANCE	SLOW	FAST	FAST

The best language for Data Oriented Design (DOD)

The best language for Data Oriented Design (DOD)

1. Keep data that is not used together apart, on separate cache lines
   - avoids the invisible convoying of false sharing (ping-pong)

The best language for Data Oriented Design (DOD)

1. **Keep data that is not used together apart**, on separate cache lines
   - avoids the invisible convoying of false sharing (ping-pong)
2. **Keep data that is frequently used together close together**
   - if a thread that uses **A** frequently also needs **B**, try to put them in one cache line

The best language for Data Oriented Design (DOD)

1. **Keep data that is not used together apart**, on separate cache lines
   - avoids the invisible convoying of false sharing (ping-pong)
2. **Keep data that is frequently used together close together**
   - if a thread that uses **A** frequently also needs **B**, try to put them in one cache line
3. **Keep “hot” and “cold” data that is not used with the same frequency apart**
   - fit “hot” data in the fewest possible cache lines and memory pages
   - reduces the cache footprint and cache misses, the memory footprint, and virtual memory paging

The best language for Data Oriented Design (DOD)

1. **Keep data that is not used together apart**, on separate cache lines
   - avoids the invisible convoying of false sharing (ping-pong)
2. **Keep data that is frequently used together close together**
   - if a thread that uses **A** frequently also needs **B**, try to put them in one cache line
3. **Keep “hot” and “cold” data that is not used with the same frequency apart**
   - fit “hot” data in the fewest possible cache lines and memory pages
   - reduces the cache footprint and cache misses, the memory footprint, and virtual memory paging

CPU instructions to execute is also data sitting in L1 cache

Modern C++

Modern C++ is not the language we used to learn 20 years ago!

Modern C++

Modern C++ is not the language we used to learn 20 years ago!

- **Move Semantics** to efficiently handle resources and improve performance

Modern C++

Modern C++ is not the language we used to learn 20 years ago!

- **Move Semantics** to efficiently handle resources and improve performance
- **`constexpr`** to move execution of the code from run-time to compile-time

Modern C++

Modern C++ is not the language we used to learn 20 years ago!

- **Move Semantics** to efficiently handle resources and improve performance
- **`constexpr`** to move execution of the code from run-time to compile-time
- **Deterministic** destruction

Modern C++

Modern C++ is not the language we used to learn 20 years ago!

- **Move Semantics** to efficiently handle resources and improve performance
- **`constexpr`** to move execution of the code from run-time to compile-time
- **Deterministic** destruction
- Better **resource handling**

Modern C++

Modern C++ is not the language we used to learn 20 years ago!

- **Move Semantics** to efficiently handle resources and improve performance
- **`constexpr`** to move execution of the code from run-time to compile-time
- **Deterministic** destruction
- Better **resource handling**
- **Value semantics** to improve performance and development experience

Modern C++

Modern C++ is not the language we used to learn 20 years ago!

- **Move Semantics** to efficiently handle resources and improve performance
- **`constexpr`** to move execution of the code from run-time to compile-time
- **Deterministic** destruction
- Better **resource handling**
- **Value semantics** to improve performance and development experience
- Full control over **concurrency and parallelism** (task-based, parallel, vectorized, lock-free, ...)

Modern C++

Modern C++ is not the language we used to learn 20 years ago!

- **Move Semantics** to efficiently handle resources and improve performance
- **`constexpr`** to move execution of the code from run-time to compile-time
- **Deterministic** destruction
- Better **resource handling**
- **Value semantics** to improve performance and development experience
- Full control over **concurrency and parallelism** (task-based, parallel, vectorized, lock-free, ...)
- Many **productivity-related improvements** (i.e. lambdas, range-for loops, **`auto`**, variadic templates, ...)

Modern C++

Modern C++ is not the language we used to learn 20 years ago!

- **Move Semantics** to efficiently handle resources and improve performance
- **`constexpr`** to move execution of the code from run-time to compile-time
- **Deterministic** destruction
- Better **resource handling**
- **Value semantics** to improve performance and development experience
- Full control over **concurrency and parallelism** (task-based, parallel, vectorized, lock-free, ...)
- Many **productivity-related improvements** (i.e. lambdas, range-for loops, **`auto`**, variadic templates, ...)
- Lots of **new toys in the C++ Standard Library**

C++20 is going to be a game-changer

C++20 is going to be a game-changer

- Concepts to provide even better compile-time safety

C++20 is going to be a game-changer

- **Concepts** to provide even better compile-time safety
- **Coroutines** to enhance asynchronous operations

C++20 is going to be a game-changer

- **Concepts** to provide even better compile-time safety
- **Coroutines** to enhance asynchronous operations
- **Ranges** to build better algorithms with higher level abstractions

C++20 is going to be a game-changer

- **Concepts** to provide even better compile-time safety
- **Coroutines** to enhance asynchronous operations
- **Ranges** to build better algorithms with higher level abstractions
- **Modules** to fix C build system and improve large-scale architecture

C++20 is going to be a game-changer

- **Concepts** to provide even better compile-time safety
- **Coroutines** to enhance asynchronous operations
- **Ranges** to build better algorithms with higher level abstractions
- **Modules** to fix C build system and improve large-scale architecture
- **<=>** to simplify implementation of value types

C++20 is going to be a game-changer

- **Concepts** to provide even better compile-time safety
- **Coroutines** to enhance asynchronous operations
- **Ranges** to build better algorithms with higher level abstractions
- **Modules** to fix C build system and improve large-scale architecture
- **<=>** to simplify implementation of value types
- **New tools** in the C++ Standard Library

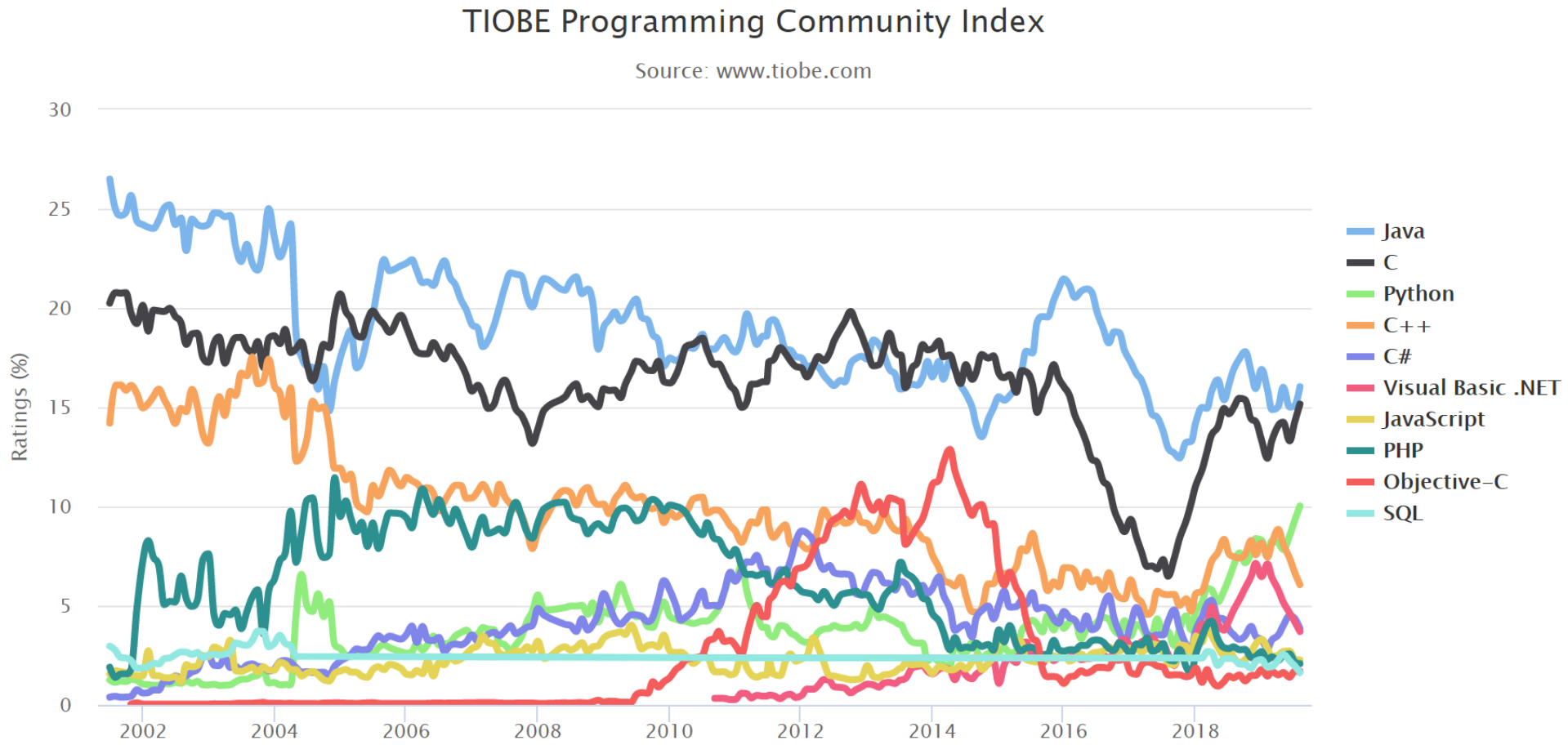
C++20 is going to be a game-changer

- **Concepts** to provide even better compile-time safety
- **Coroutines** to enhance asynchronous operations
- **Ranges** to build better algorithms with higher level abstractions
- **Modules** to fix C build system and improve large-scale architecture
- **<=>** to simplify implementation of value types
- **New tools** in the C++ Standard Library

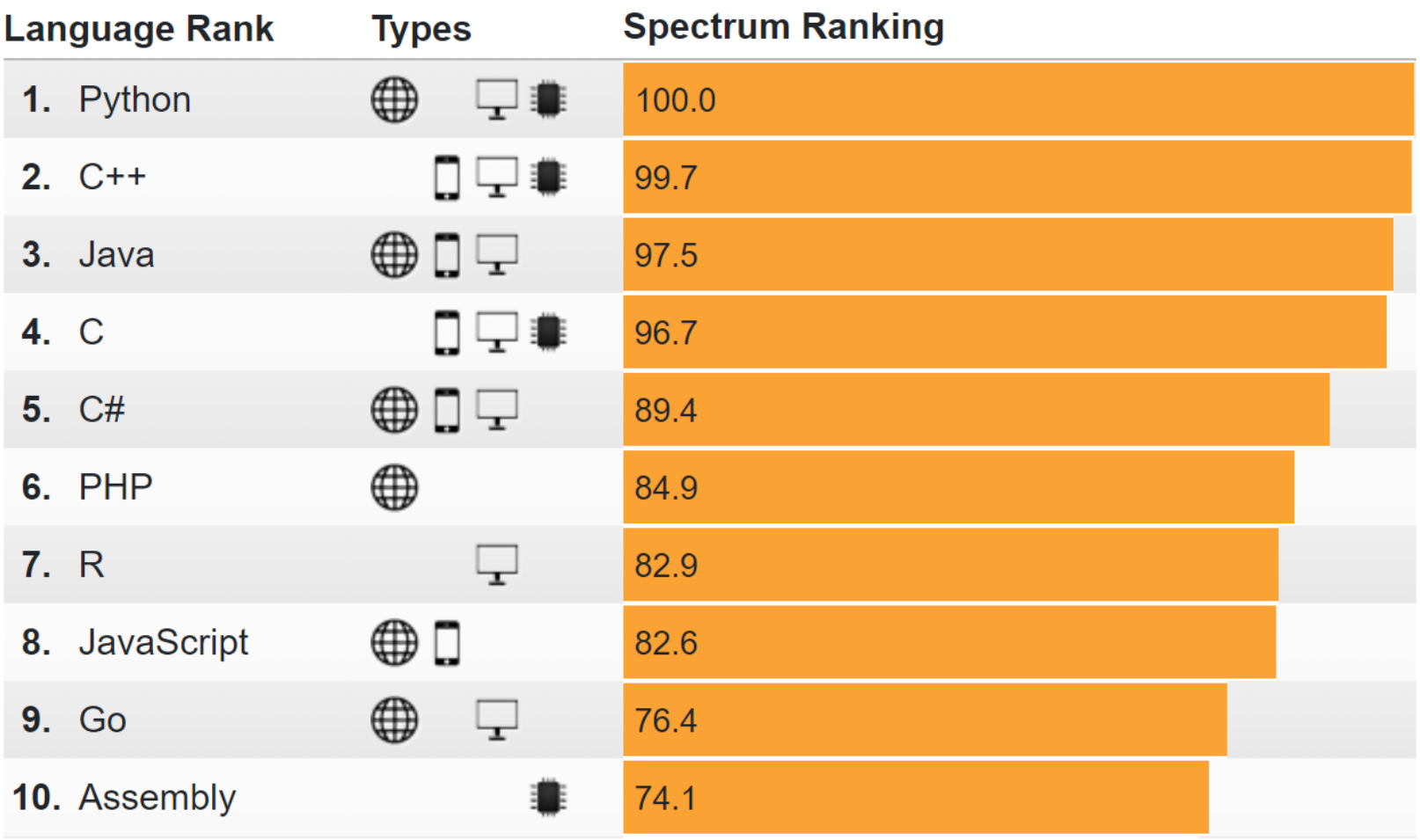
EPAM is an *active participant* of the C++ standardization process and *contributed* some performance-related features to C++20 :-)

C++ ON THE MARKET

Tiobe Index: August 2019



IEEE Spectrum: The 2018 Top Programming Languages



IEEE Spectrum: The 2018 Top Programming Languages

WEB

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. Java		97.5
3. C#		89.4
4. PHP		84.9
5. JavaScript		82.6

ENTERPRISE

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		99.7
3. Java		97.5
4. C		96.7
5. C#		89.4

MOBILE

Language Rank	Types	Spectrum Ranking
1. C++		99.7
2. Java		97.5
3. C		96.7
4. C#		89.4
5. JavaScript		82.6

EMBEDDED

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		99.7
3. C		96.7
4. Assembly		74.1
5. Arduino		69.0

Top 10 GitHub languages (# of pushes)

# Ranking	Programming Language	Percentage (Change)	Trend
1	JavaScript	27.652% (+3.768%)	
2	Python	14.915% (+0.623%)	
3	Java	11.023% (+0.831%)	
4	C++	7.068% (-0.227%)	^
5	PHP	5.718% (-1.811%)	v
6	C#	4.453% (-1.978%)	
7	Shell	4.146% (-0.627%)	
8	Ruby	4.118% (+0.001%)	
9	Go	4.082% (-0.015%)	
10	TypeScript	3.760% (+0.510%)	^

Facts about C++

https://blog.jetbrains.com/clion/2015/07/infographics-cpp-facts-before-clion

Facts about C++

~4.4 million C++ devs
~1.9 million C devs

#1

There are 4.4 million C++ developers and nearly 2 million C developers in the world.

Facts about C++

~4.4 million C++ devs
~1.9 million C devs

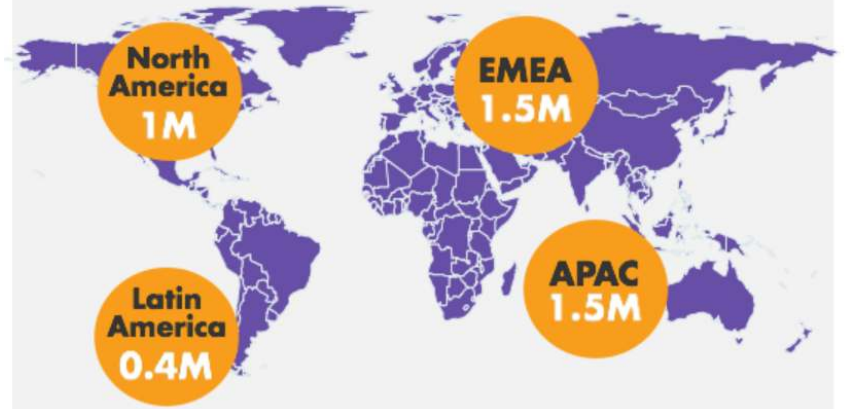
#1

There are 4.4 million C++ developers and nearly 2 million C developers in the world.

#3

C++ developers by world region

EMEA and Asia-Pacific are the most densely populated regions with regard to developers in general and C++ developers in particular.

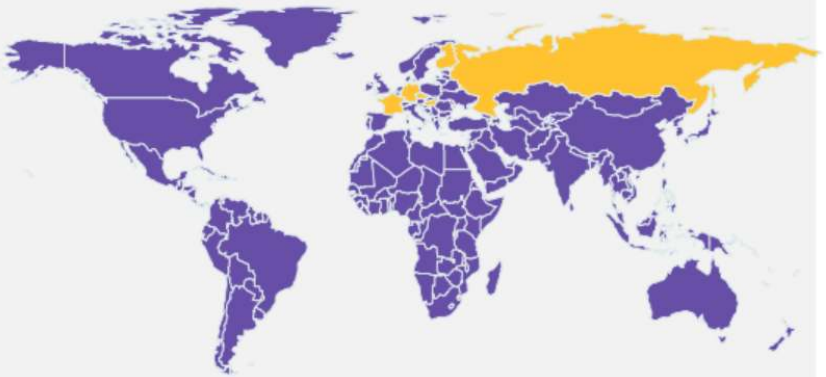


Facts about C++

#4

Where C++ is relatively ahead of other languages

C++ is relatively more popular than other languages and technologies in Russia, Czech Republic, Hungary, France, Singapore, Finland, Israel and Germany.

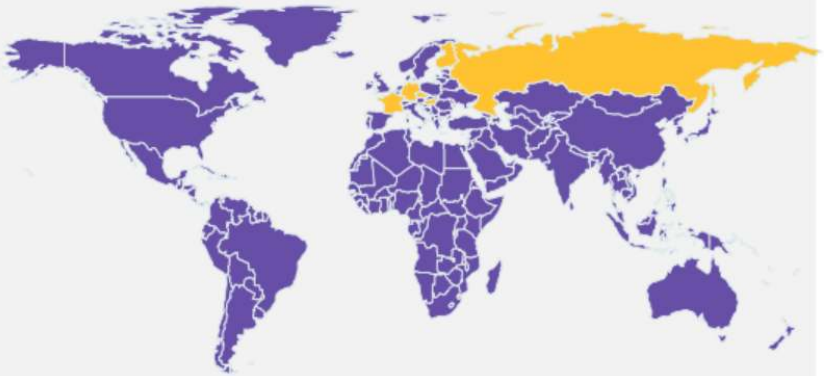


Facts about C++

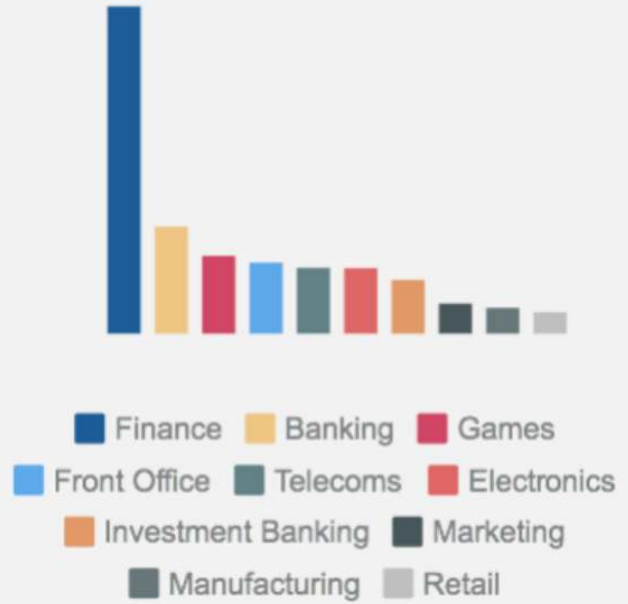
#4

Where C++ is relatively ahead of other languages

C++ is relatively more popular than other languages and technologies in Russia, Czech Republic, Hungary, France, Singapore, Finland, Israel and Germany.



#5



Industry distribution

Based on the C/C++ job ads we analyzed, C++ is most used in industries like Finance and Banking.

The Modern IT world is built on C++

The Modern IT world is built on C++

- Low-Level
  - Operating systems and drivers
  - Embedded devices (from tiny to huge scale)
  - Autonomous cars

The Modern IT world is built on C++

- Low-Level
  - Operating systems and drivers
  - Embedded devices (from tiny to huge scale)
  - Autonomous cars
- High performance libraries
  - Scientific (NASA, CERN, ...)
  - Hardware accelerated calculations
  - AI, machine learning
  - Image recognition and mapping
  - Graphics
  - Maps and navigation
  - All kinds of frameworks (Electron, ...)

The Modern IT world is built on C++

- Low-Level
  - Operating systems and drivers
  - Embedded devices (from tiny to huge scale)
  - Autonomous cars
- High performance libraries
  - Scientific (NASA, CERN, ...)
  - Hardware accelerated calculations
  - AI, machine learning
  - Image recognition and mapping
  - Graphics
  - Maps and navigation
  - All kinds of frameworks (Electron, ...)
- High performance servers
  - Stock Exchanges
  - High Frequency Trading
  - Data centers (Google, Facebook, Amazon, ...)
  - Databases (MySQL, ...)
  - Telecom

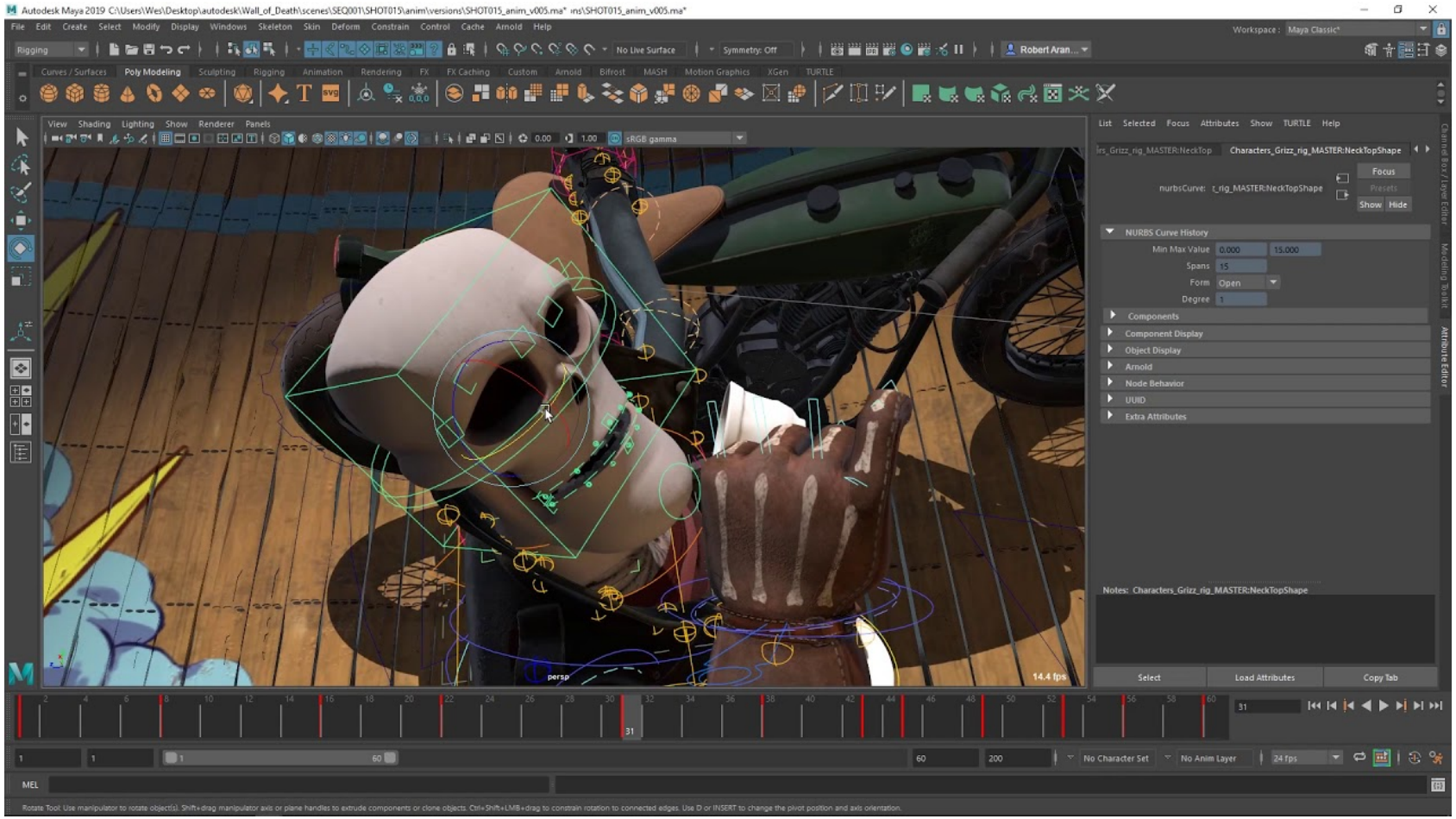
The Modern IT world is built on C++

- **Low-Level**
  - Operating systems and drivers
  - Embedded devices (from tiny to huge scale)
  - Autonomous cars
- **High performance libraries**
  - Scientific (NASA, CERN, ...)
  - Hardware accelerated calculations
  - AI, machine learning
  - Image recognition and mapping
  - Graphics
  - Maps and navigation
  - All kinds of frameworks (Electron, ...)
- **High performance servers**
  - Stock Exchanges
  - High Frequency Trading
  - Data centers (Google, Facebook, Amazon, ...)
  - Databases (MySQL, ...)
  - Telecom
- **End-user applications**
  - Games
  - Powerful modeling and simulation software (Adobe, Maya, Autodesk, Pixar, ...)
  - Compilers

Windows

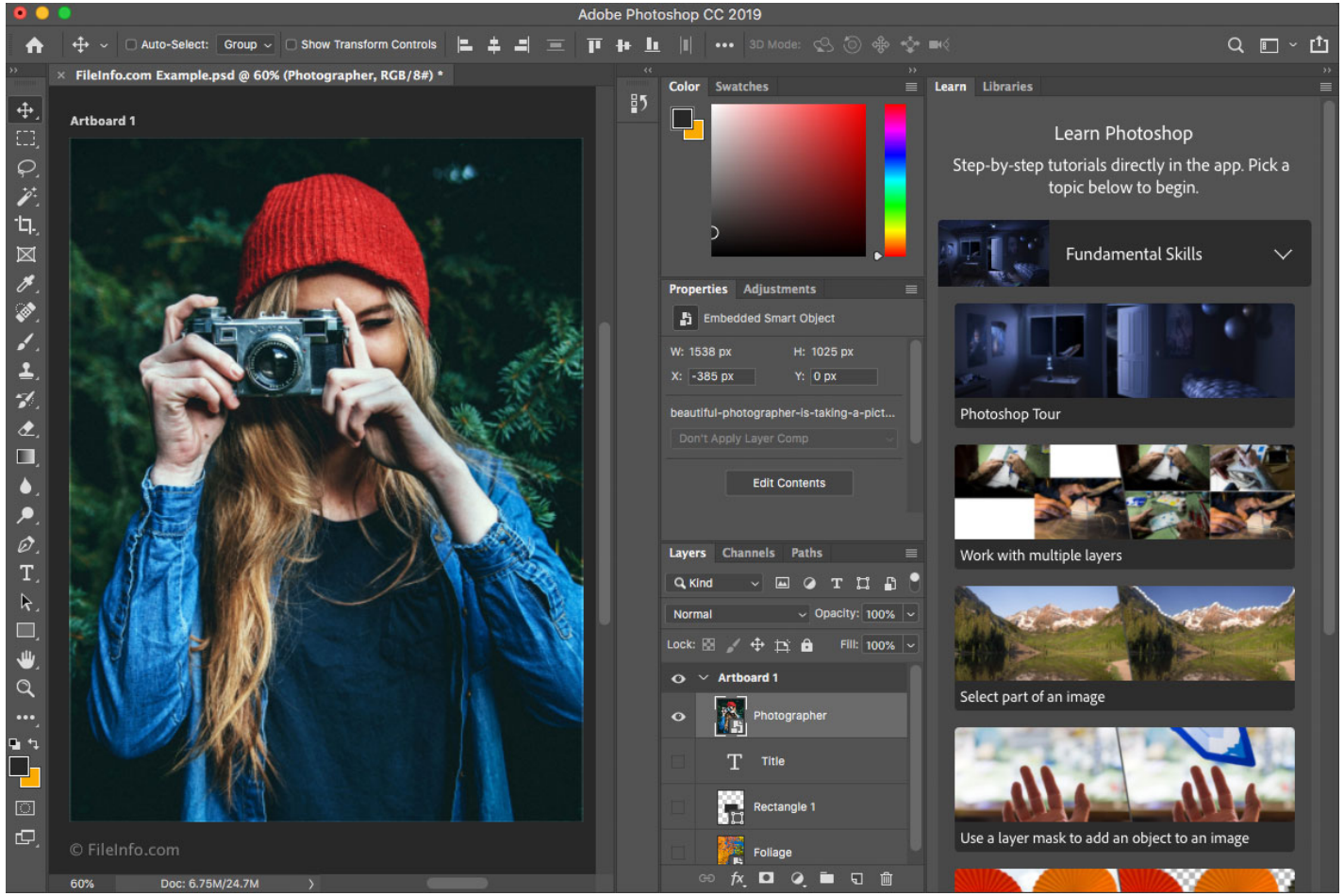

Suggested
Microsoft Mahjong
Free
A
Afterlight
Allrecipes
C
Calculator
Calendar
Camera
D
Dragon Mania Legends
Dropbox
E
Edmodo
Excel 2016
G
Welcome Daniela!
How can I help you today?
Cortana
Mike Kong
School Project
Notes
Mail
20
Photos
Sunny
68°F
88°
67°
Los Angeles
Store
Maps
Xbox
MINECRAFT
NETFLIX
Ask me anything
11:20 AM
8/18/2017

3D Computer Animation



Movies

Raster graphics editors



Light Projections

Games

C++ is the most efficient programming language in the world

C++ is the most efficient programming language in the world

1. **Learn Modern C++ and stay current** with a fast evolving language
   - follow and *learn C++* releases
   - follow *C++ conferences and blogs*
   - hire *C++ trainers*

C++ is the most efficient programming language in the world

1. **Learn Modern C++ and stay current** with a fast evolving language
   - follow and *learn C++* releases
   - follow *C++ conferences and blogs*
   - hire *C++ trainers*
2. **Use C++ in your projects** to
   - *decrease money expenditure* on computational resources
   - boost *performance*
   - limit *jitter* (improve response time)
   - limit *power usage*
   - limit *memory footprint*
   - have *more control over hardware*