Contemporary C++: Learning Modern C++ in a Modern Way

الماس فناوری ابری پاسارگاد- آلفا

مدرس: سعید امراللهی بیوکی

ABOUT ME …

$ who am i

- Shahid Beheshti university: BA in Software Engineering
- Azad university: MA in Software Engineering
- 26+ years of experience in C/C++ programming
- I have never had a course on C/C++
- I am the first representative of IRAN at C++ standardization committee at Frankfurt (2009), Rapperswil (2010, 2014), Hawaii (2012) and I paid everything myself.
- In last 9 years, I deeply have focused on embedded System Programming using C/C++, Programming Instant messengers and Cloud-gaming.
- Currently I work for Alpha co.

Agenda 1/24

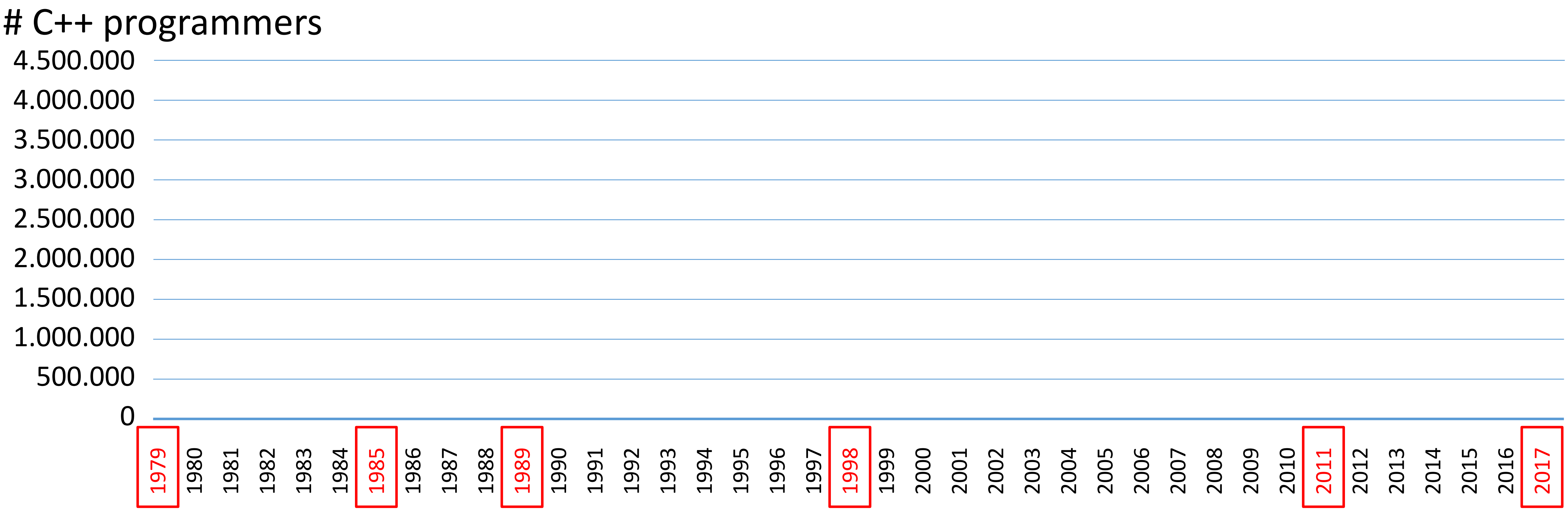
- Course Aims and Outline
- Is C++ dead?
- Four giant C++ Applications
- A brief history of C++
- Software, Programming and Programming languages
- Why C++ or If C++ is the answer, what is the question?
- C++98: Conventional definition
- Hello C++: The first program: Hello, world!
- Install and Setup C++ compilers
- Compile, Link and Execute chain
- 2nd program: Say a greeting to a specific person
- Basic concepts of Stream I/O
- Q & A

150 min (incl. Q & A)

Is C++ dead?

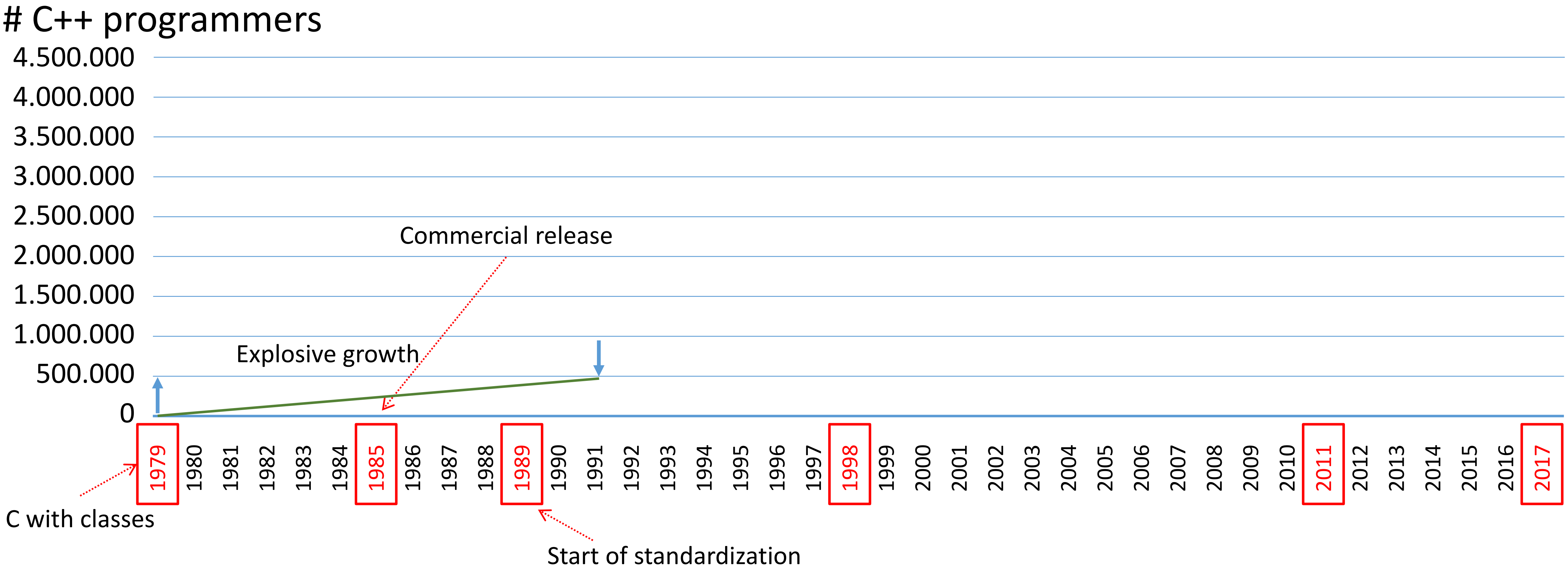
Is C++ dead?

Is C++ dead?



# C++ programmers
4.500.000
4.000.000
3.500.000
3.000.000
2.500.000
2.000.000
1.500.000
1.000.000
500.000
0
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017

Is C++ dead?



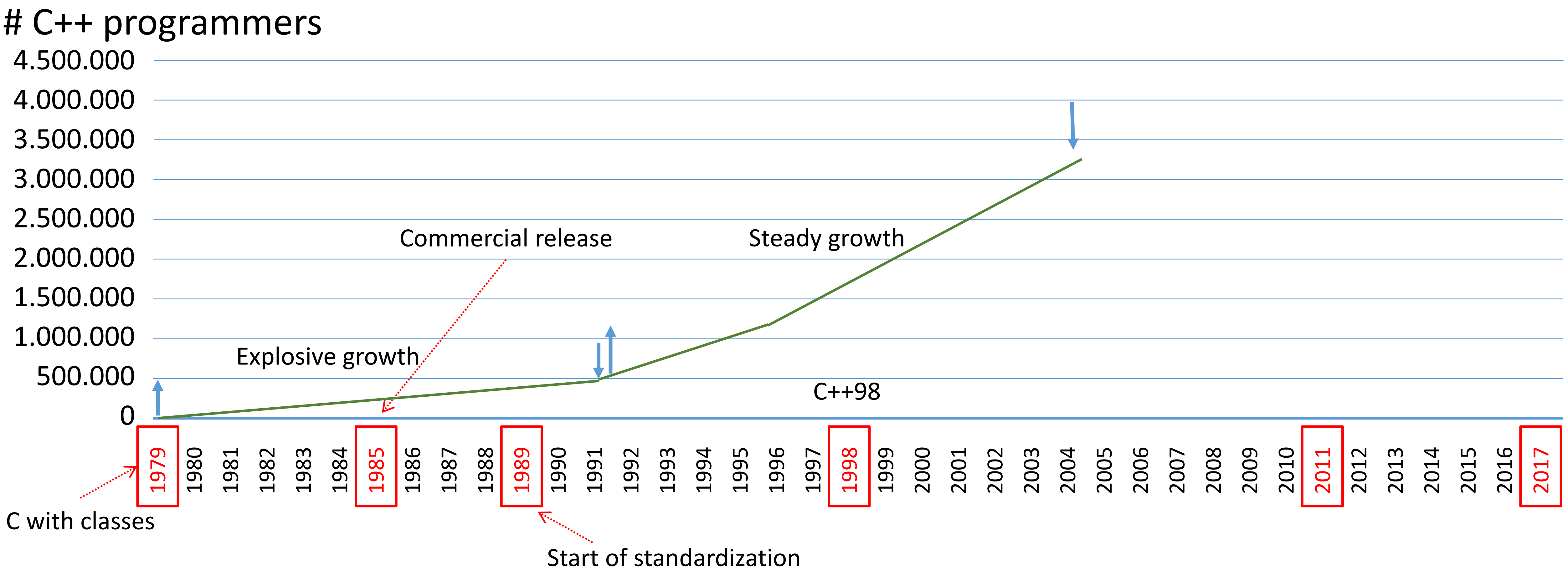
Commercial release

Explosive growth

C with classes

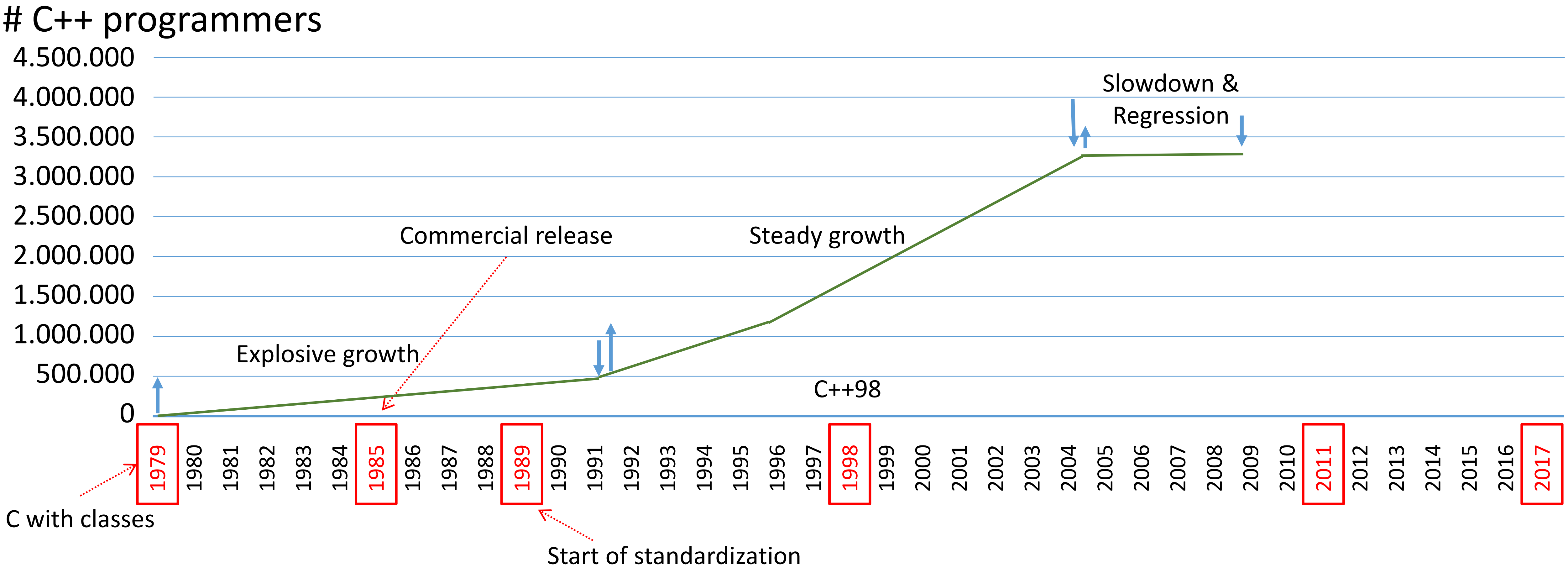
Start of standardization

Is C++ dead?



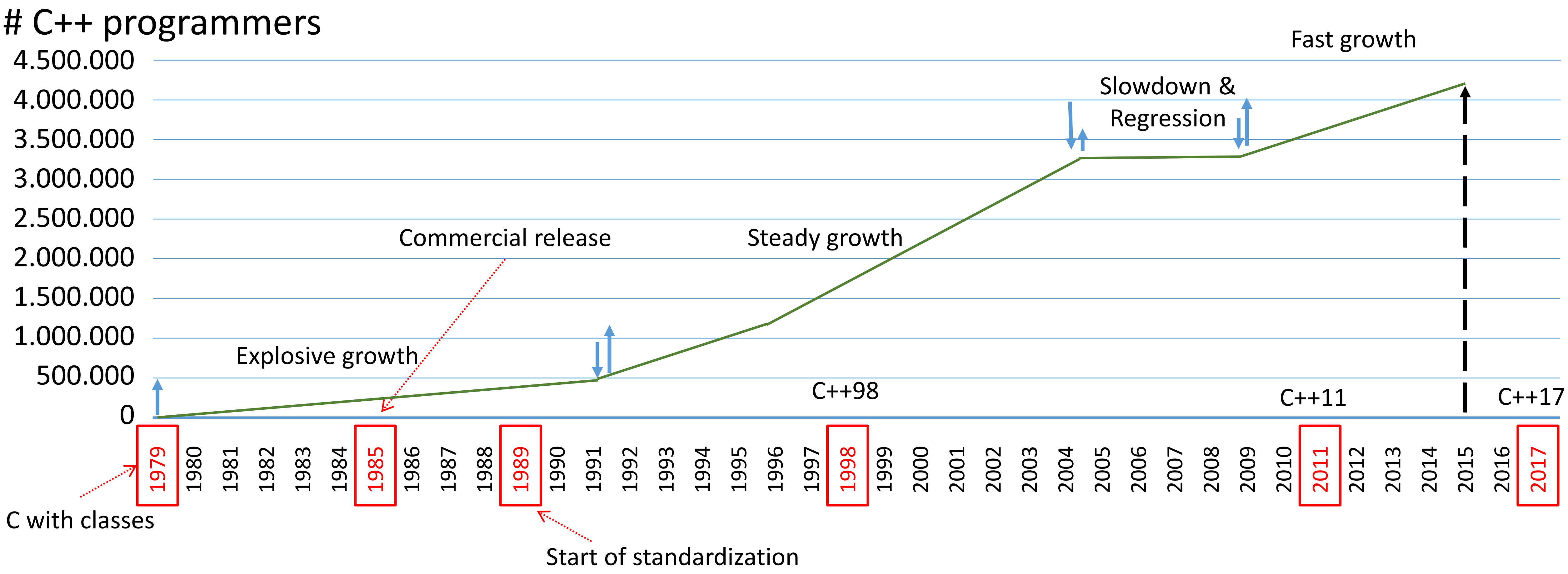
# C++ programmers
4.500.000
4.000.000
3.500.000
3.000.000
2.500.000
2.000.000
1.500.000
1.000.000
500.000
0
Commercial release
Steady growth
Explosive growth
C++98
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
C with classes
Start of standardization

Is C++ dead?



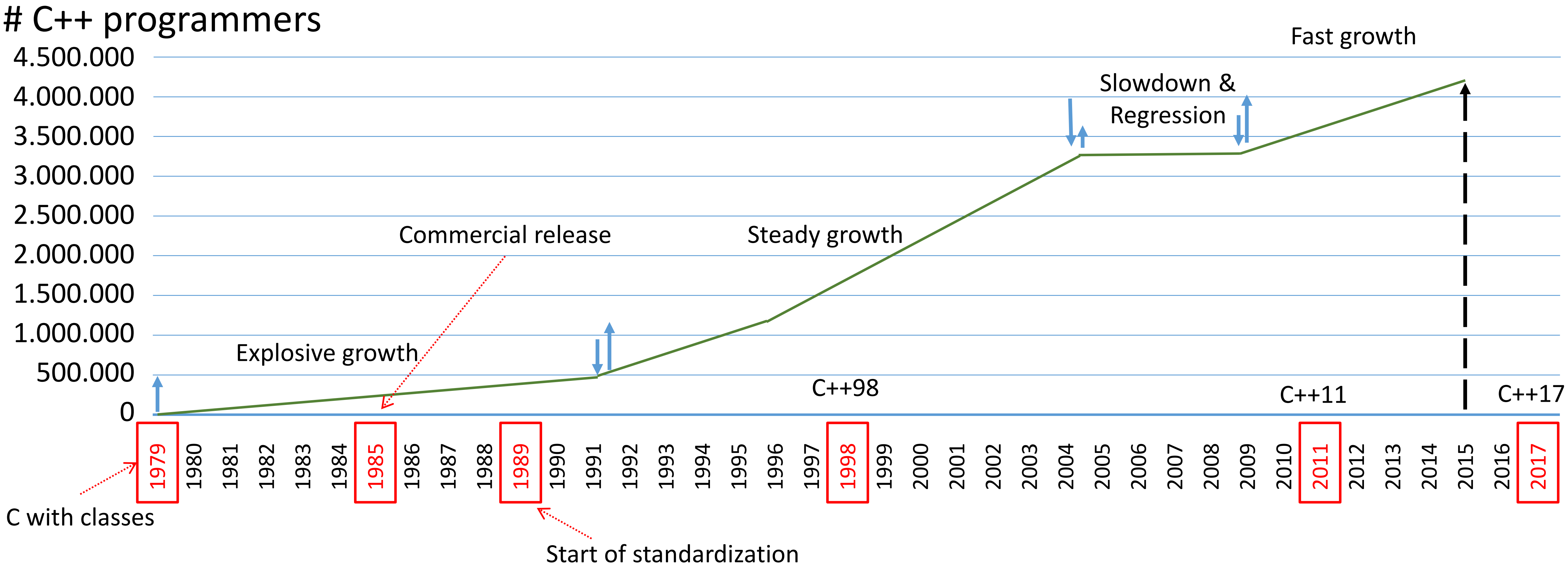
# C++ programmers
4.500.000
4.000.000
3.500.000
3.000.000
2.500.000
2.000.000
1.500.000
1.000.000
500.000
0
Slowdown &
Regression
Commercial release
Steady growth
Explosive growth
C++98
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
C with classes
Start of standardization

Is C++ dead?



# C++ programmers
4.500.000
4.000.000
3.500.000
3.000.000
2.500.000
2.000.000
1.500.000
1.000.000
500.000
0
Fast growth
Slowdown &
Regression
Commercial release
Steady growth
Explosive growth
C++98
C++11
C++17
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
C with classes
Start of standardization

Is C++ dead?



- 35+ years language evolution.
- At 2015: ~ 4.4 million C++ developers
  ~ 1.9 million C developers
- Number of C++ programmers: it's more than to 5,000,000.

Is C++ dead?

http://www.stroustrup.com/applications.html

- Our civilization is on C and C++.
- You use C++ already, even if you write program in Java or C#.

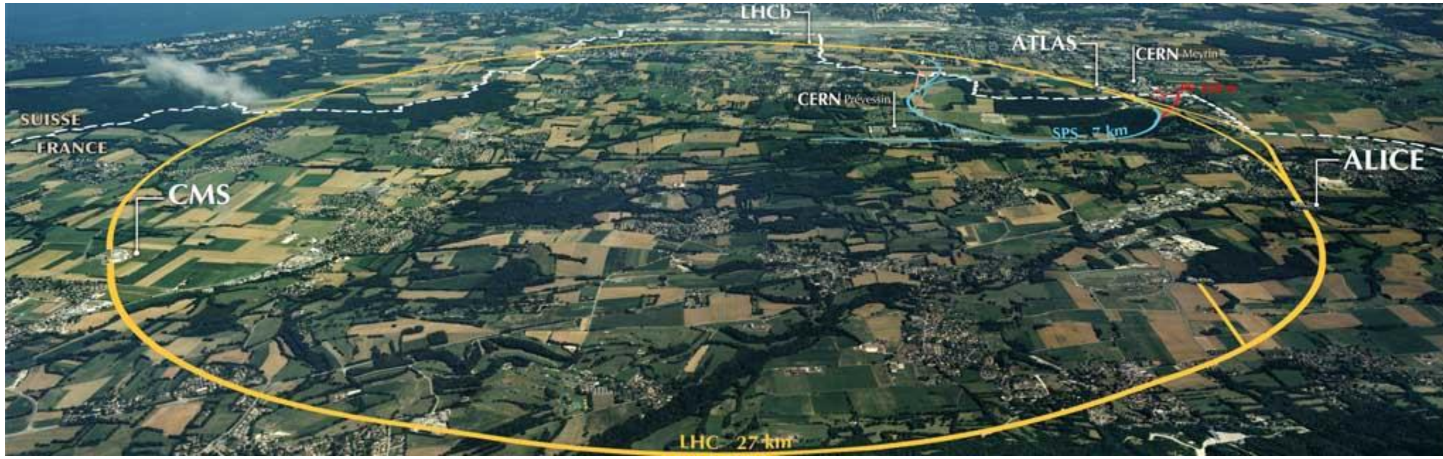
    - C# compiler itself is written in C++.

    - Most Java virtual machines (JVM) are written in C++.

Higgs boson particles

- The European Organization for Nuclear Research known as CERN
- CERN operates the largest particle physics laboratory in the world.

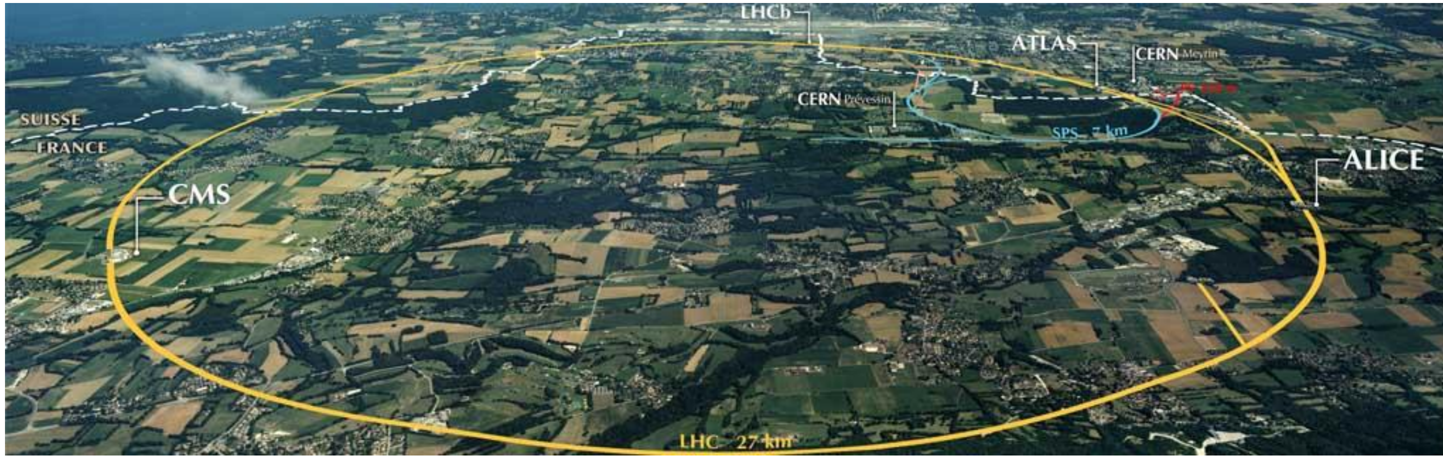
- The European Organization for Nuclear Research known as CERN
- CERN operates the largest particle physics laboratory in the world.

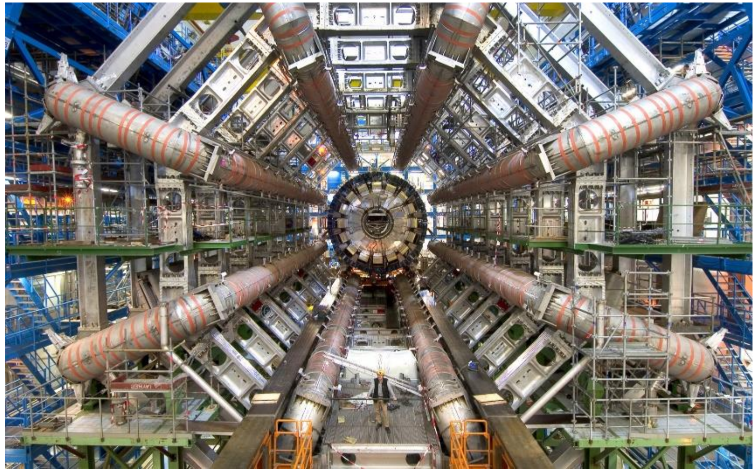


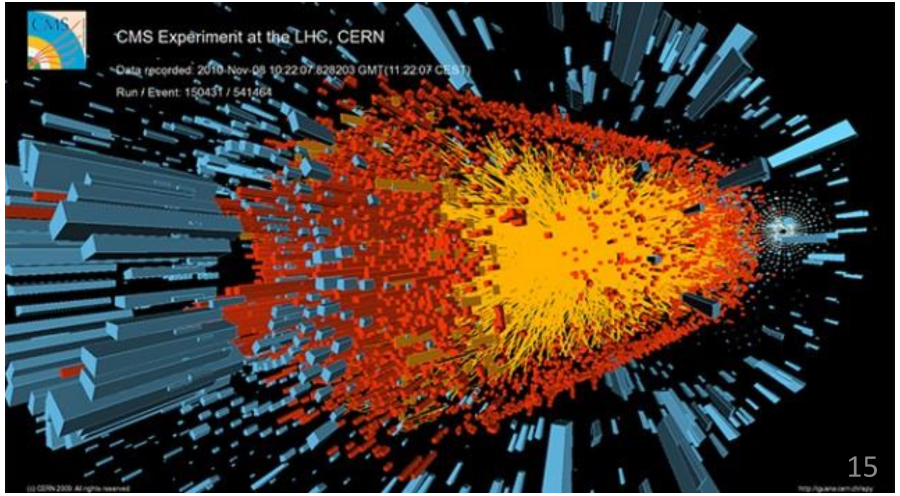
- CERN (Large Hadron Collider)

- The European Organization for Nuclear Research known as CERN
- CERN operates the largest particle physics laboratory in the world.

- CERN (Large Hadron Collider)



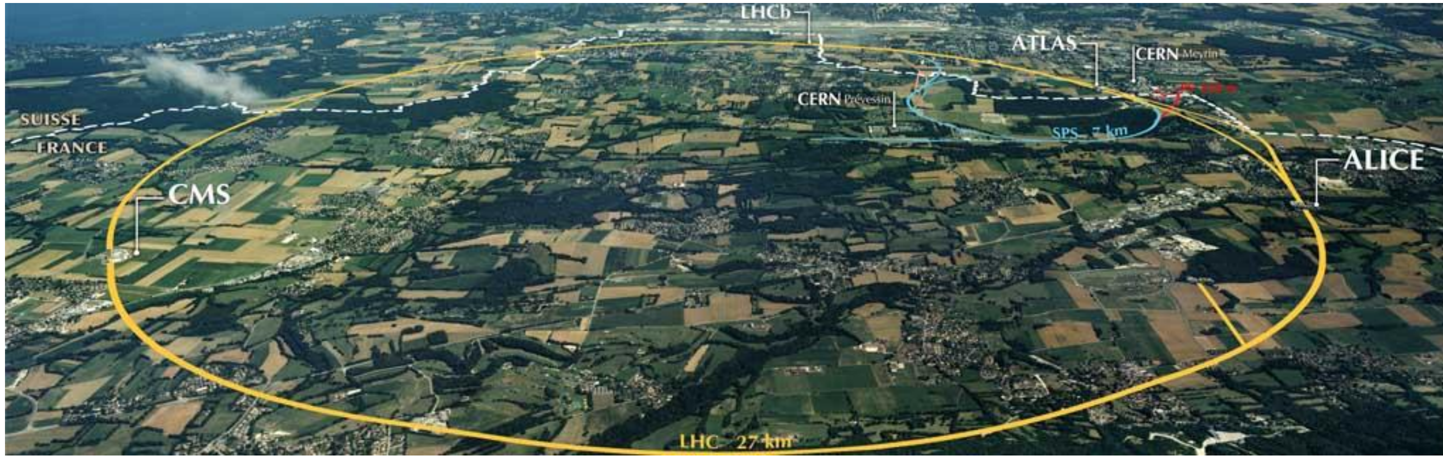


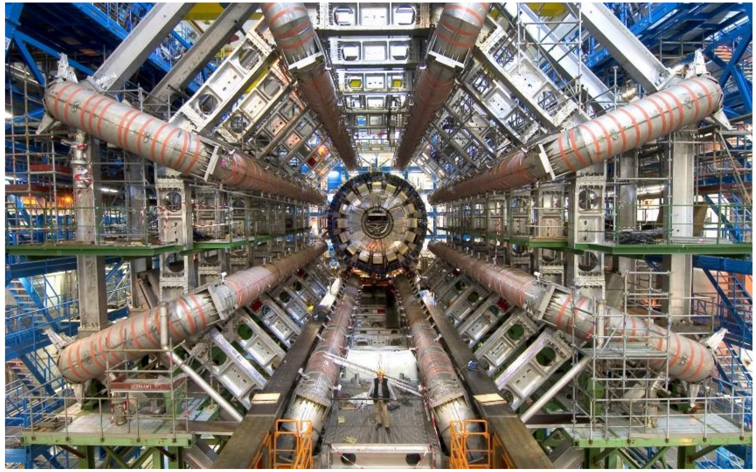

CMS Experiment at the LHC, CERN
Data recorded: 2010-Nov-08 10:22:07.828203 GMT(11:22:07 CEST)
Run / Event: 150431 / 541464

C++ inside

- The European Organization for Nuclear Research known as CERN
- CERN operates the largest particle physics laboratory in the world.
- CERN (Large Hadron Collider)
- -At CERN- all related computing done in C++.

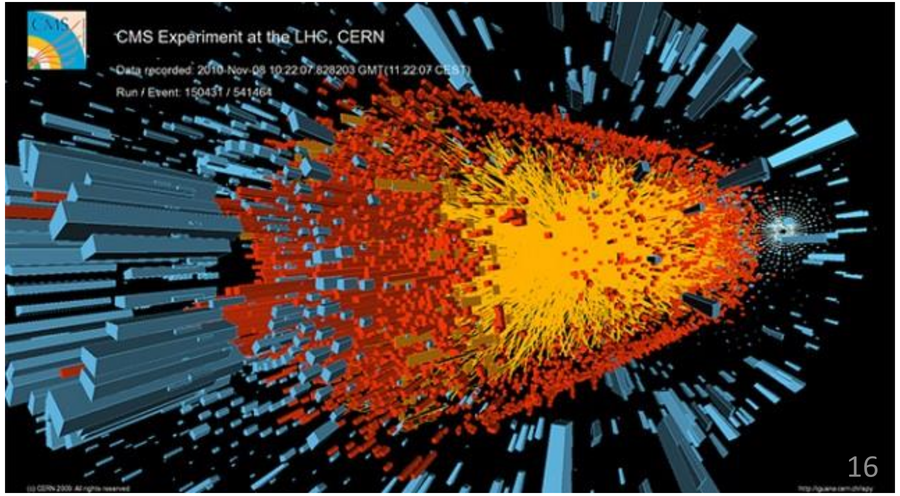
Axel Neumann





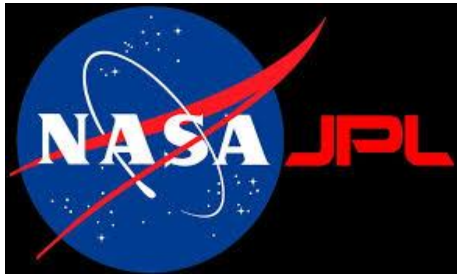
[c++std-core-21966] Higgs boson
Bjarne Stroustrup
Wed 7/4/2012, 5:08 PM
c++std-news@accu.org (c++std-core@accu.org)
From a short note I received from CERN this morning: "all related computing done in C++."

What we do matter.


CMS Experiment at the LHC, CERN
Data recorded: 2010-Nov-08 10:22:07.828203 GMT(11:22:07 CEST)
Run / Event: 150431 / 541464

Mars Rover

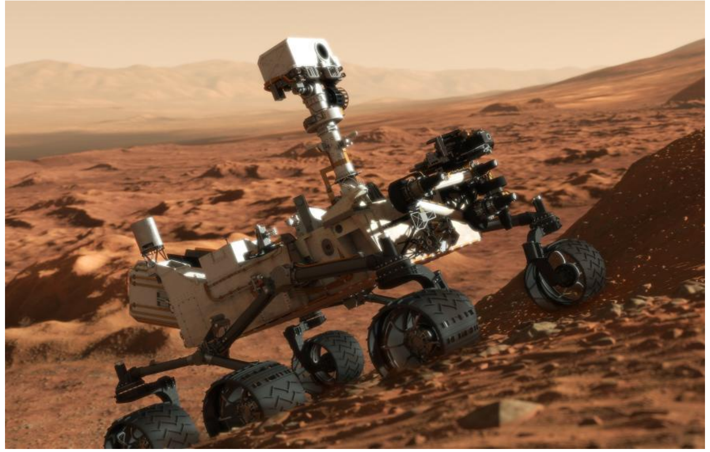
Mars Rover

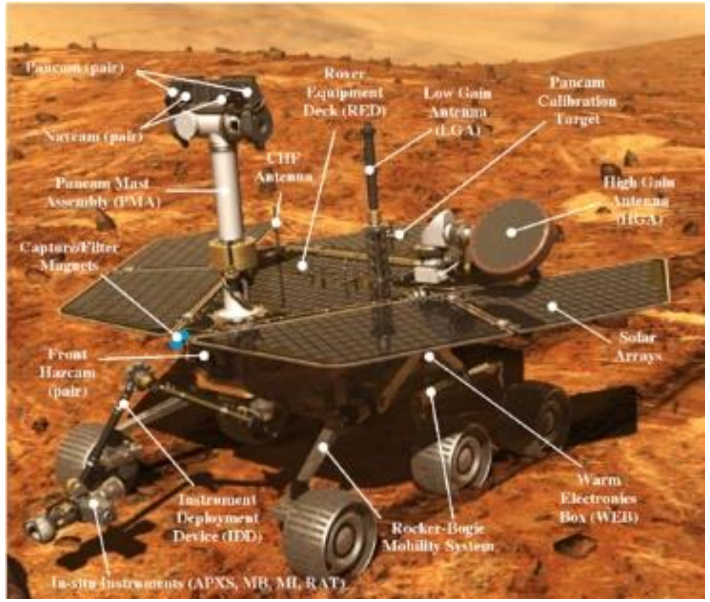


- The Jet Propulsion Laboratory (JPL)
- The JPL is owned by NASA and managed by the nearby California Institute of Technology (Caltech) for NASA.
- https://www.jpl.nasa.gov/

Mars Rover

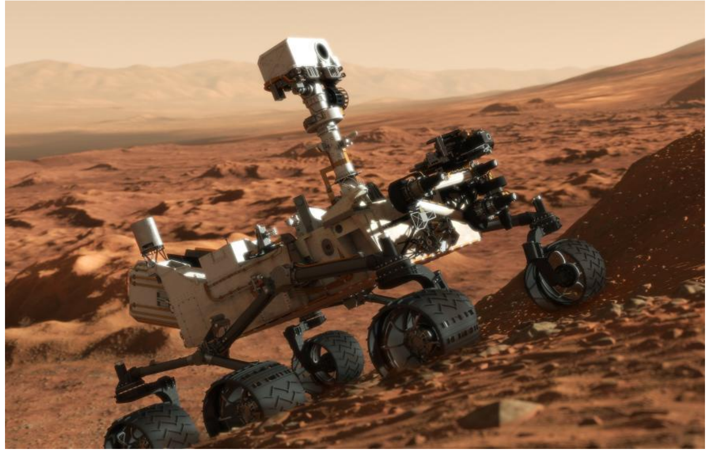
- The Jet Propulsion Laboratory (JPL)
  - The JPL is owned by NASA and managed by the nearby California Institute of Technology (Caltech) for NASA.
- https://www.jpl.nasa.gov/

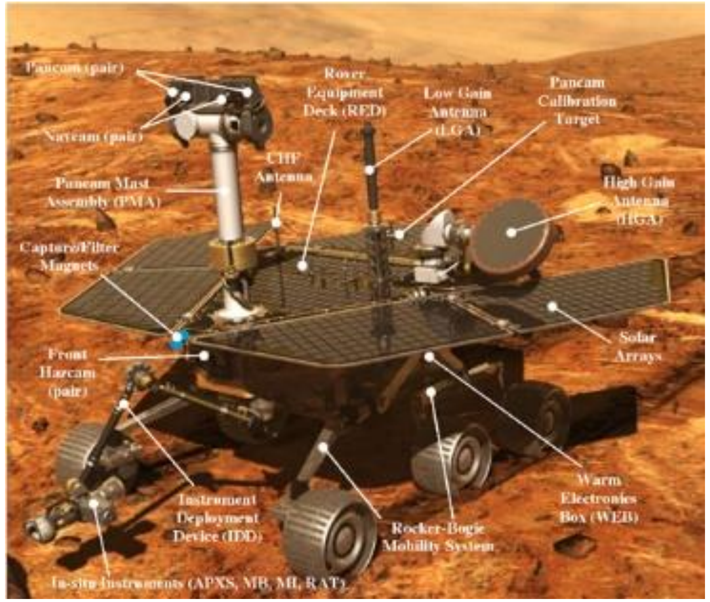



Pancam (pair)
Navcam (pair)
Pancam Mast Assembly (PMA)
Capture/Filter Magnets
Front Hazcam (pair)
Instrument Deployment Device (IDD)
In-situ Instruments (APXS, MB, MI, RAT)
Rover Equipment Deck (RED)
UHF Antenna
Low Gain Antenna (LGA)
Pancam Calibration Target
High Gain Antenna (HGA)
Solar Arrays
Warm Electronics Box (WEB)
Rocker-Bogie Mobility System

Mars Rover

- The Jet Propulsion Laboratory (JPL)
  - The JPL is owned by NASA and managed by the nearby California Institute of Technology (Caltech) for NASA.
- https://www.jpl.nasa.gov/




Pancam (pair)
Navcam (pair)
Pancam Mast Assembly (PMA)
Capture/Filter Magnets
Front Hazcam (pair)
Rover Equipment Deck (RED)
UHF Antenna
Low Gain Antenna (LGA)
Pancam Calibration Target
High Gain Antenna (HGA)
Solar Arrays
Warm Electronics Box (WEB)
Rocker-Bogie Mobility System
Instrument Deployment Device (IDD)
In-situ Instruments (APXS, MB, MI, RAT)

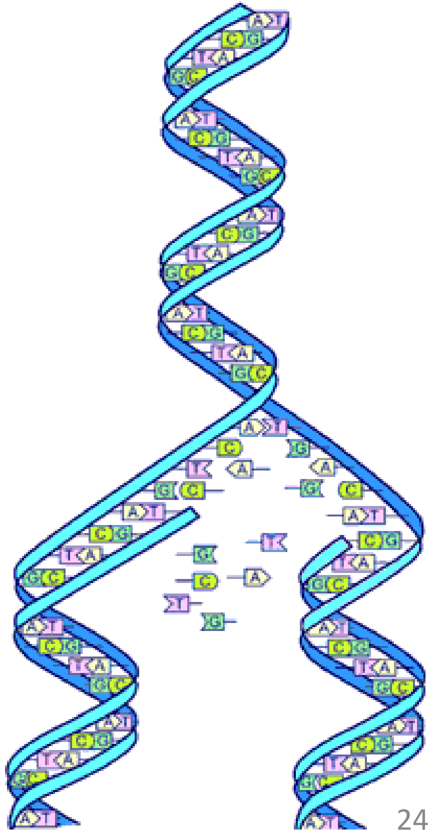
- Mars Rover autonomous driving system: Scene analysis & Route planning
- Spirit, Curiosity & Opportunity

Human Genome Sequencing Project

Human Genome Sequencing Project

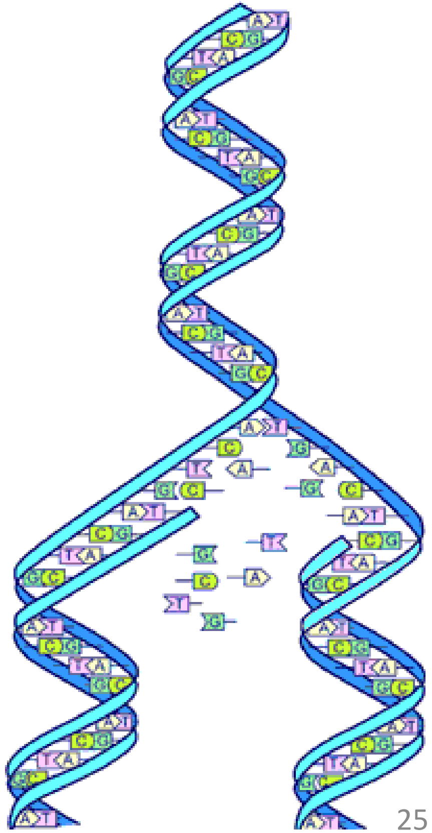
Human Genome Sequencing Project

CELERA
an Applera Corporation Business



Human Genome Sequencing Project

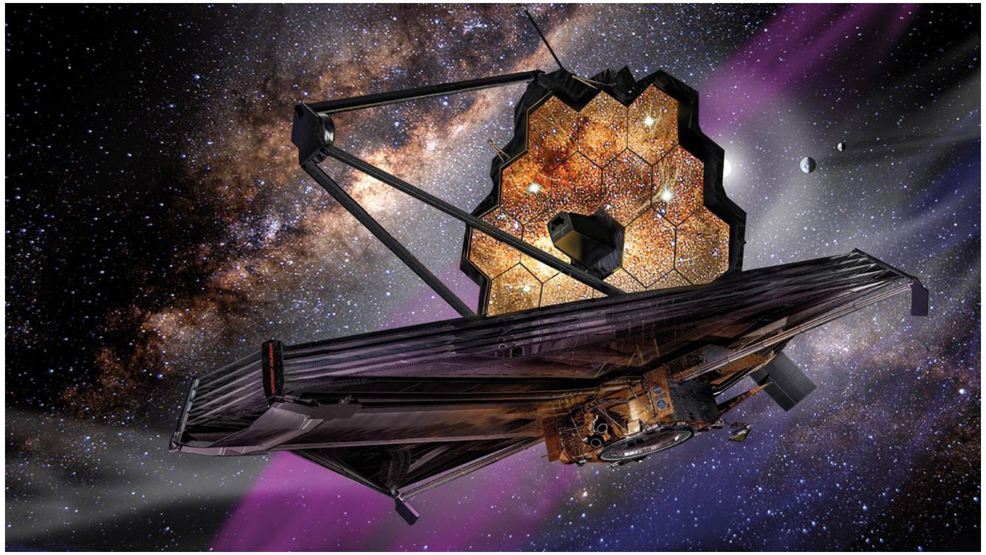
CELERA
an Applera Corporation Business

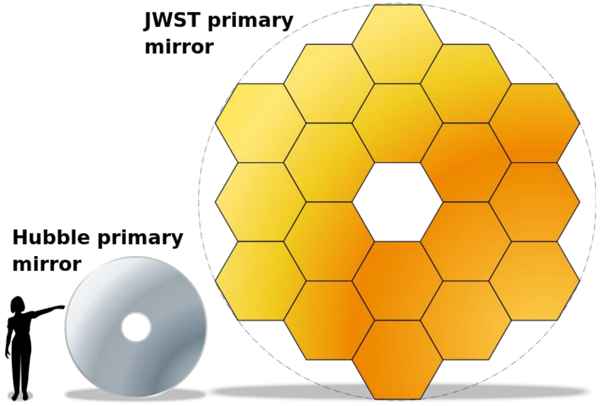


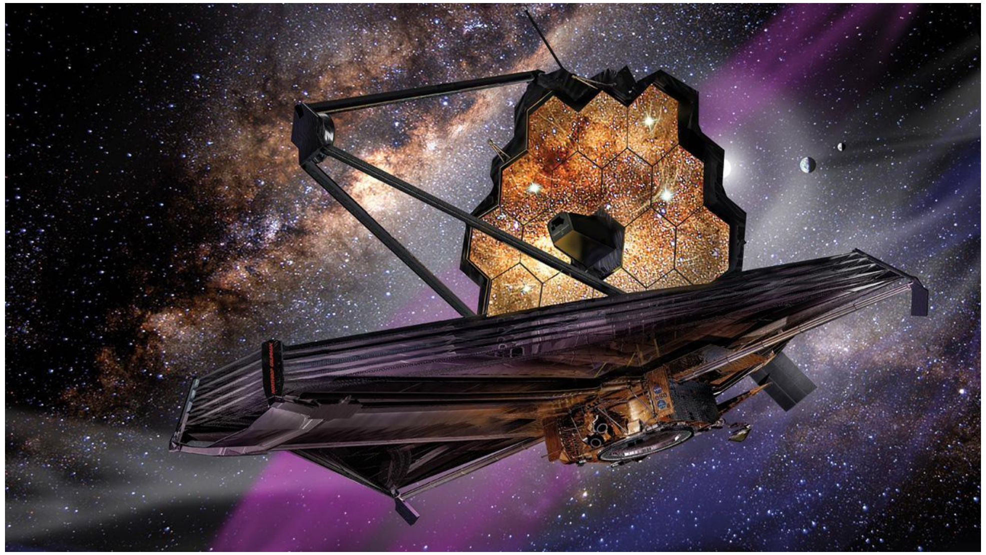
C++ inside

- The Jet Propulsion Laboratory (JPL)
- The JPL is owned by NASA and managed by the nearby California Institute of Technology (Caltech) for NASA.
- https://www.jpl.nasa.gov/

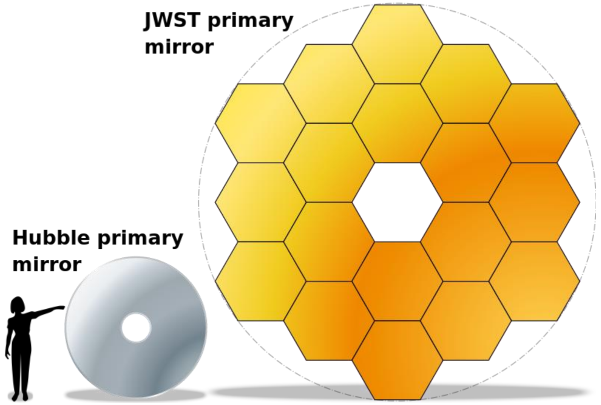
- James Webb is the successor to the Hubble Space Telescope.







- The Jet Propulsion Laboratory (JPL)
- The JPL is owned by NASA and managed by the nearby California Institute of Technology (Caltech) for NASA.
- https://www.jpl.nasa.gov/



- James Webb is the successor to the Hubble Space Telescope.

What’s a Programming Language?

- A tool for instructing machines
- A notation for algorithms
- A means for communications among programmers
- A tool for experimentation
- A means for controlling computer controlled gadgets
- A mean for controlling computerized devices
- A way of expressing relationships among concepts
- A means for expressing high-level design
- All of the above!

  - and more

▪ Is study of programming language important?

- Yes! You can′t learn to program without a programming language.

The prehistory of C++

The prehistory of C++

- Bjarne Stroustrup
- Computing Laboratory at Cambridge university
- Ph.D. Thesis: Organization of system software for distributed systems.

  Simulating software running on a distributed system.

The prehistory of C++

- Bjarne Stroustrup
- Computing Laboratory at Cambridge university, 1977
- Ph.D. Thesis: Organization of system software for distributed systems.

  Simulating software running on a distributed system.
- A simulator for simulating software running on a distributed system.
- The simulator was written using *Simula* programming language.

The Simula programming language

The Simula programming language

- It is based on Algol-60.
- Simula: discrete event simulation language.

The Simula programming language

- It is based on Algol-60.
- Simula: discrete event simulation language.
- Class concept: mapping application concepts into language constructs.

  Example: class computer

The Simula programming language

- It is based on Algol-60.
- Simula: discrete event simulation language.
- Class concept: mapping application concepts into language constructs.
  Example: class computer
- Class hierarchies were used to express variants of application-level concepts.
  Example: different types of computers derived from class Computer.

The Simula programming language

- It is based on Algol-60.
- Simula: discrete event simulation language.
- Class concept: mapping application concepts into language constructs.
  Example: class computer
- Class hierarchies were used to express variants of application-level concepts.
  Example: different types of computers derived from class Computer.
- Concurrency mechanism: Pseudo-parallel Co-routine.

The Simula programming language

- It is based on Algol-60.
- Simula: discrete event simulation language.
- Class concept: mapping application concepts into language constructs.
  Example: class computer
- Class hierarchies were used to express variants of application-level concepts.
  Example: different types of computers derived from class Computer.
- Concurrency mechanism: Pseudo-parallel Co-routine.
- Type system: compiler’s ability to catch type errors.
- Simula *flexible* type system vs. Pascal *rigid* and *primitive* strong type system.

The Simula programming language

- It is based on Algol-60.
- Simula: discrete event simulation language.
- Class concept: mapping application concepts into language constructs.
  Example: class computer
- Class hierarchies were used to express variants of application-level concepts.
  Example: different types of computers derived from class Computer.
- Concurrency mechanism: Pseudo-parallel Co-routine.
- Type system: compiler's ability to catch type errors.
- Simula *flexible* type system vs. Pascal *rigid* and *primitive* strong type system.
- Simula is the first *Object-oriented* programming language.

Simula language vs. Simula implementation

- Scalability problem
- The Simula implementation was geared to small programs and was inherently unsuitable for larger programs.
- Simula implementation:

    - run-time type checking

    - Guaranteed initialization

    - *Concurrency support*

    - *Garbage collection*.

Simula language vs. Simula implementation

- Scalability problem
- The Simula implementation was geared to small programs and was inherently unsuitable for larger programs.
- Simula implementation:
  - run-time type checking
  - Guaranteed initialization
  - *Concurrency support*
  - *Garbage collection*.
- Simula implementation drawbacks:
  - long link-time
  - Poor runtime performance
  - Garbage collector: %80 of runtime was spent in garbage collector.

Simula language vs. Simula implementation

- Scalability problem
- The Simula implementation was geared to small programs and was inherently unsuitable for larger programs.
- Simula implementation:

  - run-time type checking

  - Guaranteed initialization

  - *Concurrency support*

  - *Garbage collection*.

- Simula implementation drawbacks:

  - long link-time

  - Poor runtime performance

  - Garbage collector: %80 of runtime was spent in garbage collector.

- Writing simulator with BCPL programming language.
- BCPL makes C look like a very high-level language and provides absolutely no type checking or run-time support.

Bell Laboratories

Bell Laboratories

Brian Kernighan
Bjarne Stroustrup

Douglas McIlroy

Dennis Ritchie

Ken Thompson

- B, C, BCPL, C++, UNIX, Plan 9

Unix kernel distribution

Unix kernel distribution

- UNIX Kernel Distribution over a network of computers connected by a LAN:

  - How to analyze the network traffic

  - How to modularize the kernel

Unix kernel distribution

- UNIX Kernel Distribution over a network of computers connected by a LAN:

  - How to analyze the network traffic

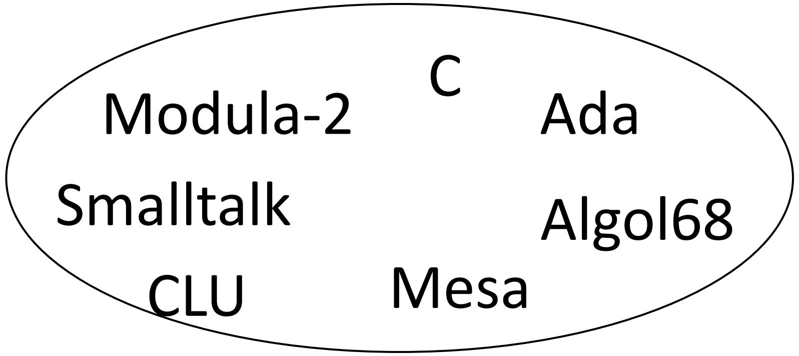
  - How to modularize the kernel

- This is exactly the kind of problem that I had become determined never again to attack without proper tools.

Why C?

Why C?

Choosing Base programming language:



C

Modula-2

Ada

Smalltalk

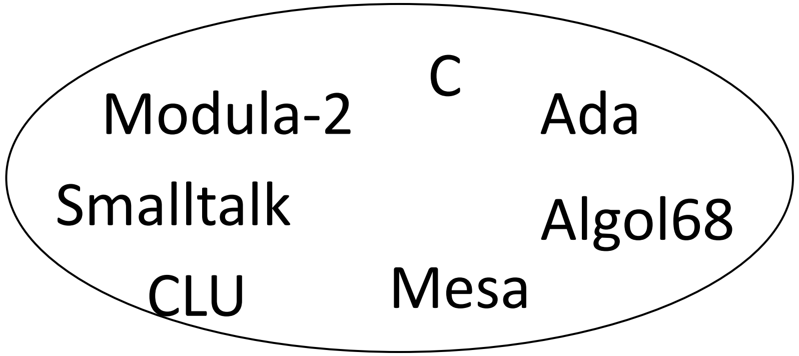
Algol68

CLU

Mesa

Why C?

Choosing Base programming language:



C

Modula-2

Ada

Smalltalk

Algol68

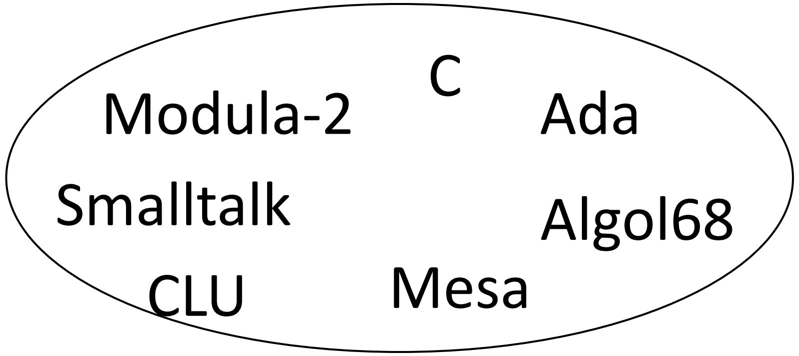
CLU

Mesa

- C is clearly not the cleanest language ever designed nor the easiest to use so why do so many people use it?

Why C?

Choosing Base programming language:



C

Modula-2

Ada

Smalltalk

Algol68

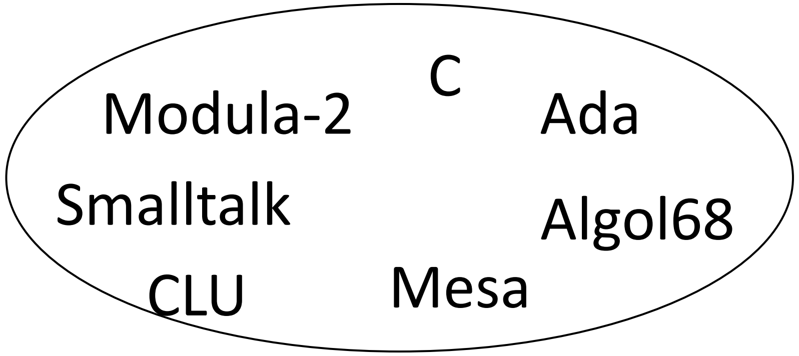
CLU

Mesa

- C is clearly not the cleanest language ever designed nor the easiest to use so why do so many people use it?
    - C is *flexible*

Why C?

Choosing Base programming language:



C

Modula-2

Ada

Smalltalk

Algol68

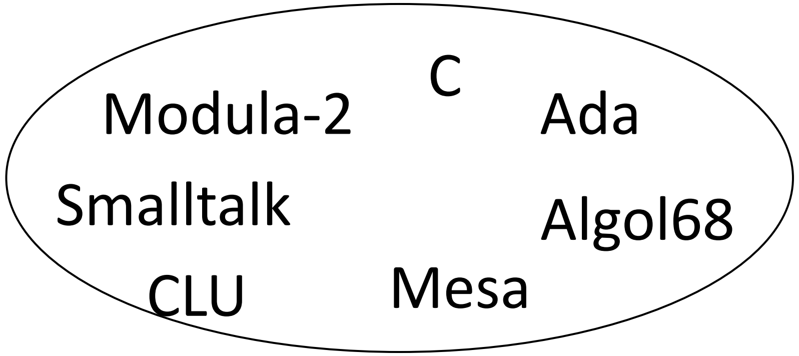
CLU

Mesa

- C is clearly not the cleanest language ever designed nor the easiest to use so why do so many people use it?
  - C is *flexible*
  - C is *efficient*

Why C?

Choosing Base programming language:



C

Modula-2

Ada

Smalltalk

Algol68

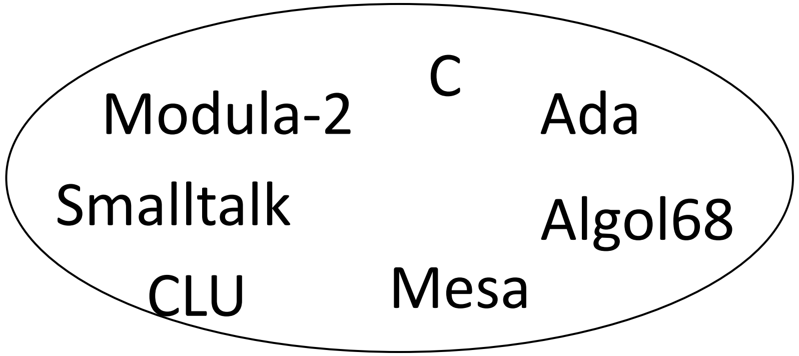
CLU

Mesa

- C is clearly not the cleanest language ever designed nor the easiest to use so why do so many people use it?
  - C is *flexible*
  - C is *efficient*
  - C is *available*

Why C?

Choosing Base programming language:



- C is clearly not the cleanest language ever designed nor the easiest to use so why do so many people use it?

  - C is *flexible*
  - C is *efficient*
  - C is *available*
  - C is *portable*

The C programming language?

The C programming language?

C is a small language.

- Brian W. Kernighan & Dennis M. Ritchie:The C Programming Language

The C programming language?

C is a small language.

- Brian W. Kernighan & Dennis M. Ritchie:The C Programming Language

- C : System programming - Procedural programming

The C programming language?

C is a small language.

- Brian W. Kernighan & Dennis M. Ritchie:The C Programming Language

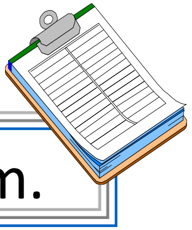
- C : System programming - Procedural programming
- C: machine-architecture-independent notions that directly map to the key hardware notions

The C programming language?

C is a small language.

- Brian W. Kernighan & Dennis M. Ritchie:The C Programming Language

- C : System programming - Procedural programming
- C: machine-architecture-independent notions that directly map to the key hardware notions



Every C program should be a valid C^{++} program.

The C programming language?

C is a small language.

- Brian W. Kernighan & Dennis M. Ritchie:The C Programming Language

- C : System programming - Procedural programming
- C: machine-architecture-independent notions that directly map to the key hardware notions

Every C program should be a valid C^{++} program.

- As close to C as Possible, but no closer!

C with classes

C with classes

- October 1979:

  - A preprocessor: Cpre

  - The language accepted by the preprocessor was called “C with Classes”.

C with classes

- October 1979:

  - A preprocessor: Cpre

  - The language accepted by the preprocessor was called “C with Classes”.

- October 1980:

  - Language rather than a tool

C with classes

- October 1979:

  - A preprocessor: Cpre

  - The language accepted by the preprocessor was called “C with Classes”.

- October 1980:

  - Language rather than a tool

  - A General Purpose Language not a Special Purpose one.

    Concurrency, Specific application areas

C with classes

- October 1979:

  - A preprocessor: Cpre

  - The language accepted by the preprocessor was called “C with Classes”.

- October 1980:

  - Language rather than a tool

  - A General Purpose Language not a Special Purpose one.

    Concurrency, Specific application areas

  - No compromise between Efficiency and Elegance.

C with classes

- October 1979:

  - A preprocessor: Cpre

  - The language accepted by the preprocessor was called “C with Classes”.

- October 1980:

  - Language rather than a tool

  - A General Purpose Language not a Special Purpose one.

    Concurrency, Specific application areas

  - No compromise between Efficiency and Elegance.

General rule

C++ is a language not a complete system.

References and further readings

- Bjarne Stroustrup. The Design and Evolution of C++. Addison-Wesley, 1994.

Original C++ Semantics

Original C++ Semantics

- C++ was designed to combine the strengths of C as a systems programming language with Simula’s facilities for organizing programs.

Dennis Ritchie
1941-2011

Kristen Nygaard
1926-2002

Original C++ Semantics

- C++ was designed to combine the strengths of C as a systems programming language with Simula’s facilities for organizing programs.

Dennis Ritchie
1941-2011

Kristen Nygaard
1926-2002

- C Semantic: Close to the machine, Machine level facilities, *efficient* code

Original C++ Semantics

- C++ was designed to combine the strengths of C as a systems programming language with Simula's facilities for organizing programs.

Dennis Ritchie
1941-2011

Kristen Nygaard
1926-2002

- C Semantic: Close to the machine, Machine level facilities, *efficient* code
- Simula is the first object-oriented programming language.
- Simula Semantic: close to the problem to be solved, abstraction, *elegant* code

Original C++ Semantics

- C++ was designed to combine the strengths of C as a systems programming language with Simula's facilities for organizing programs.

Dennis Ritchie
1941-2011

Kristen Nygaard
1926-2002

- C Semantic: Close to the machine, Machine level facilities, *efficient* code
- Simula is the first object-oriented programming language.
- Simula Semantic: close to the problem to be solved, abstraction, *elegant* code
- No compromise between efficiency & elegance.
- C++ vs. C: As close to C as possible but no closer!

Conventional definition

Conventional definition

- Many C^{++} design decisions have their roots in my dislike for forcing people to do things in some particular way.

- Bjarne Stroustrup

Conventional definition

- Many C^{++} design decisions have their roots in my dislike for forcing people to do things in some particular way.

  - Bjarne Stroustrup

1997-2010

C^{++} is a *general-purpose programming language* with a bias towards *systems programming* that

- is a better C
- supports data abstraction
- supports object-oriented programming
- supports generic programming

Conventional definition

- Many C^{++} design decisions have their roots in my dislike for forcing people to do things in some particular way.

  - Bjarne Stroustrup

1997-2010

C^{++} is a *general-purpose programming language* with a bias towards *systems programming* that

- is a better C
- supports data abstraction
- supports object-oriented programming
- supports generic programming

Conventional definition

- Many C^{++} design decisions have their roots in my dislike for forcing people to do things in some particular way.

- Bjarne Stroustrup

1997-2010

C^{++} is a *general-purpose programming language* with a bias towards *systems programming* that

- is a better C
- supports data abstraction
- supports object-oriented programming
- supports generic programming

Conventional definition

- Many C^{++} design decisions have their roots in my dislike for forcing people to do things in some particular way.

  - Bjarne Stroustrup

1997-2010

C^{++} is a *general-purpose programming language* with a bias towards *systems programming* that

- is a better C → Macros, structures & functions
- supports data abstraction → Classes
- supports object-oriented programming → Inheritance & Polymorphism
- supports generic programming → Templates

Conventional definition

- Many C^{++} design decisions have their roots in my dislike for forcing people to do things in some particular way.

- Bjarne Stroustrup

1997-2010

C^{++} is a *general-purpose programming language* with a bias towards *systems programming* that

- is a better C → Macros, structures & functions
- supports data abstraction → Classes
- supports object-oriented programming → Inheritance & Polymorphism
- supports generic programming → Templates

C++ general rule:

General rule

C++ is a language not a complete system.

Conventional definition

- Many C^{++} design decisions have their roots in my dislike for forcing people to do things in some particular way.

- Bjarne Stroustrup

1997-2010

C^{++} is a *general-purpose programming language* with a bias towards *systems programming* that

- is a better C → Macros, structures & functions
- supports data abstraction → Classes
- supports object-oriented programming → Inheritance & Polymorphism
- supports generic programming → Templates

C++ general rule:

General rule

C++ is a language not a complete system.

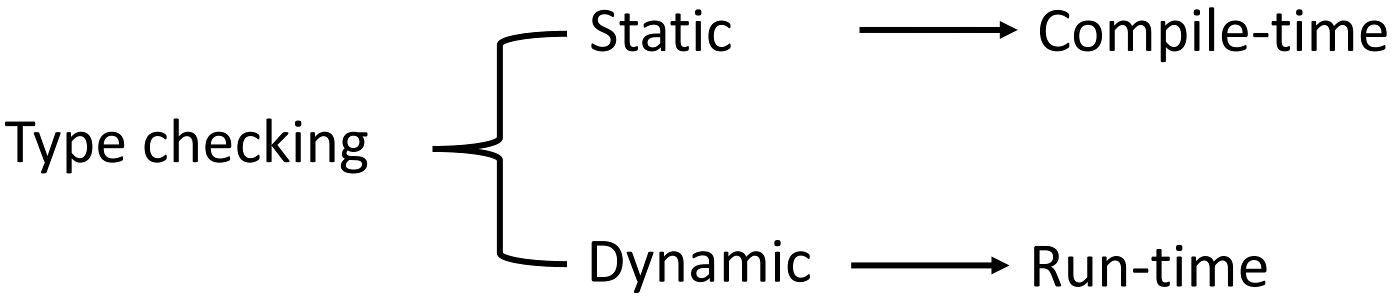
- C++ is a multi-paradigm/multi-style programming language.
- It’s old, but still very useful definition.

If C++ is the answer, what is the question?

- Static type checking
- Light-weight abstractions
- The zero overhead principle: What you don't use, you don't pay for it!
- Talk to hardware directly
- Multi-paradigm/Multi-style programming
- Compatibility with C programming language
- C++ is Multi-threaded programming language
- Compile-time computation

Type system and type checking

- Type system
- Type checking: the process of checking that every expression is used according to its type.



Type checking

Static → Compile-time

Dynamic → Run-time

Static type checking

Static type checking

- A language is *statically typed* if the type of a variable is known at compile time.

  Example: C, C++, Java, Go, Rust

- Static type checking means *compile-time* type checking. All type checking is done by compiler.

> Static type: Type of an expression resulting from analysis of the program without considering execution semantics.
>
> from *Committee Draft*

Static type checking

- A language is *statically typed* if the type of a variable is known at compile time.

  Example: C, C++, Java

- Static type checking means *compile-time* type checking. All type checking is done by compiler.

> Static type: Type of an expression resulting from analysis of the program without considering execution semantics.
>
> from *Committee Draft*

- A language is *dynamically typed* if the type is associated with *run-time values*, and not named variables/fields/etc.

  Example: Lisp, Smalltalk, Perl, Ruby, Python

Static type checking

- A language is *statically typed* if the type of a variable is known at compile time.

  Example: C, C++, Java

- Static type checking means *compile-time* type checking. All type checking is done by compiler.

> Static type: Type of an expression resulting from analysis of the program without considering execution semantics.
>
> from *Committee Draft*

- A language is *dynamically typed* if the type is associated with run-time values, and not named variables/fields/etc.

  Example: Lisp, Smalltalk, Perl, Ruby, Python

- C++ follow the Simula model of type checking and inheritance, *not* the Smalltalk or Lisp models.

Static type checking

- A language is *statically typed* if the type of a variable is known at compile time.

  Example: C, C++, Java

- Static type checking means *compile-time* type checking. All type checking is done by compiler.

> Static type: Type of an expression resulting from analysis of the program without considering execution semantics.
>
> from *Committee Draft*

- A language is *dynamically typed* if the type is associated with run-time values, and not named variables/fields/etc.

  Example: Lisp, Smalltalk, Perl, Ruby, Python

- C++ follow the Simula model of type checking and inheritance, *not* the Smalltalk or Lisp models.

- Other terminologies: Strong type checking, type safety, …

Static type checking- more details

Static type checking- more details

- The type of every variable should be known at compile time.
- All function calls are checked at compile time.
- The full type (both return type and the argument types) of a function should be known/fixed at compile time.
- The interface of classes should be known/fixed at compile time.

  - Simula, C++, Java vs. Smalltalk, Lisp
- …

Static type checking- advantages

Static type checking- advantages

- A tool for good design

Static type checking- advantages

- A tool for good design
- early error detection

Static type checking- advantages

- A tool for good design
- early error detection
- Run-time efficiency

X safety

- Type safety
- Resource safety
- Exception safety
- Resource safety

C++ and C type System

Performance story

Performance story

- Efficiency vs. Performance

Performance story

- Efficiency vs. Performance

Performance story:

… Two hikers who had taken off their shoes and were resting, when they saw a bear. One of them slowly started putting his shoes back on. The other said, "Why are you putting your shoes on? If that bear sees us, you can't outrun it." The first one replied, "I don't have to outrun the bear; I just have to outrun you.")

Performance story

- Efficiency vs. Performance

Performance story:

… Two hikers who had taken off their shoes and were resting, when they saw a bear. One of them slowly started putting his shoes back on. The other said, "Why are you putting your shoes on? If that bear sees us, you can't outrun it." The first one replied, "I don't have to outrun the bear; I just have to outrun you.")

- A program that is too slow will have no users and might as well not exist.

Performance story

- Efficiency vs. Performance

Performance story:
… Two hikers who had taken off their shoes and were resting, when they saw a bear. One of them slowly started putting his shoes back on. The other said, "Why are you putting your shoes on? If that bear sees us, you can't outrun it." The first one replied, "I don't have to outrun the bear; I just have to outrun you.")

- A program that is too slow will have no users and might as well not exist.
- Improving efficiency involves *doing less work or using less resources.*
- Improving performance involves doing work faster.
- C++ doesn’t give you performance, but it gives you control over performance.

Performance story

- Efficiency vs. Performance

Performance story:

… Two hikers who had taken off their shoes and were resting, when they saw a bear. One of them slowly started putting his shoes back on. The other said, "Why are you putting your shoes on? If that bear sees us, you can't outrun it." The first one replied, "I don't have to outrun the bear; I just have to outrun you.")

- A program that is too slow will have no users and might as well not exist.
- Improving efficiency involves *doing less work or using less resources.*
- Improving performance involves doing work faster.
- C++ doesn't give you performance, but it gives you control over performance.

"If you're not at all interested in performance, shouldn't you be in the Python room down the hall?"
– Scott Meyers

Light-weight abstractions

- Compact data structures

  - *Mission-critical* & *Resource-constrained* infrastructure software

- light-weight abstraction:

  abstractions that do not impose space or time overheads in excess of what would be imposed by careful hand coding of a particular example of the abstraction.

- The characteristics of light-weight abstractions:

  - A simple and Direct mapping to hardware

  - Zero-overhead abstractions

- Fundamental types: bool, char, int, double, …
- derived types: pointers, references, arrays, …

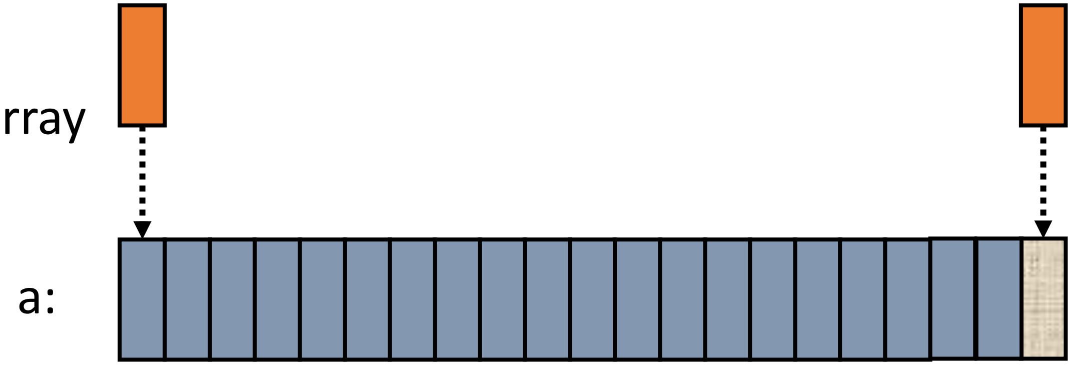
- There is no “abstract”, “virtual” or mathematical model between the C++ programmer’s expressions and the machine’s facilities.

Light-weight abstractions- examples

- C-style Array: *continuous* sequence of bytes

```
int a[20];
```

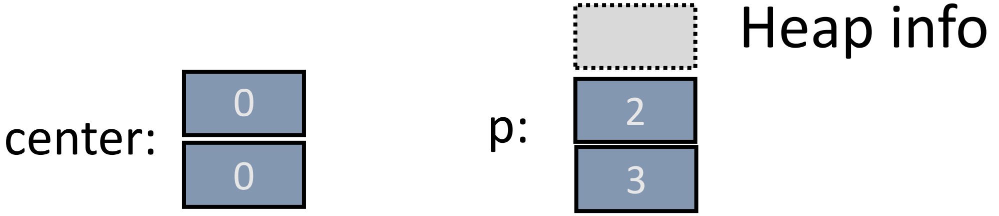
Pointer p:
beginning of the array



Pointer q
to one beyond the last
element of the array

- Pointers: machine addresses
- Simple user-defined data types:

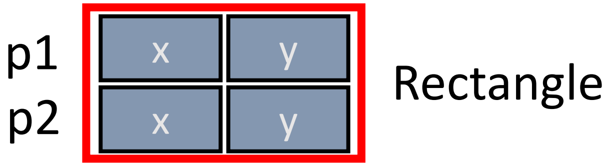
```
class Point { // 2D point concept
    int x, y; // implementation
    // …
};
Point center{0, 0};
Point* p = new Point{2, 3};
```



- A Point is simply the concatenation of its data members, so the size of the Point center is simply two times the size of an int.

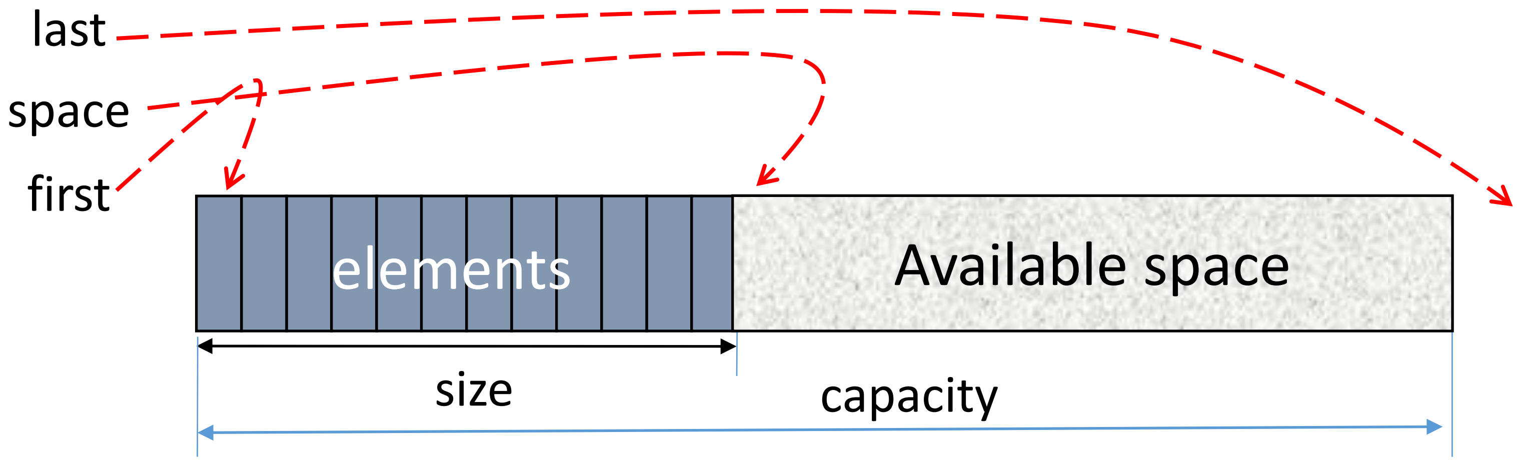
Light-weight abstractions- more examples

```
class Rectangle {
    Point p1_;
    Point p2_;
public:
    Rectangle(const Point& p1, const Point& p2) :
        p1_{p1}, p2_{p2}
    {}
    Rectangle(int x1, int y1, int x2, int y2) :
        p1_{ x1, y1 }, p2_{ x2, y2 }
    {}
};
```



Light-weight abstractions- vector

- Vector has two associated sizes: size and capacity. size is the number of elements that it contains. The other, called its capacity, is the total amount of memory that is available for storing elements.



```
template <class T /*, allocator */ >
class vector {
   T* first;
   T* space;
   T* last;
}; // 24 bytes (on 64 bits systems)
```

- vector size

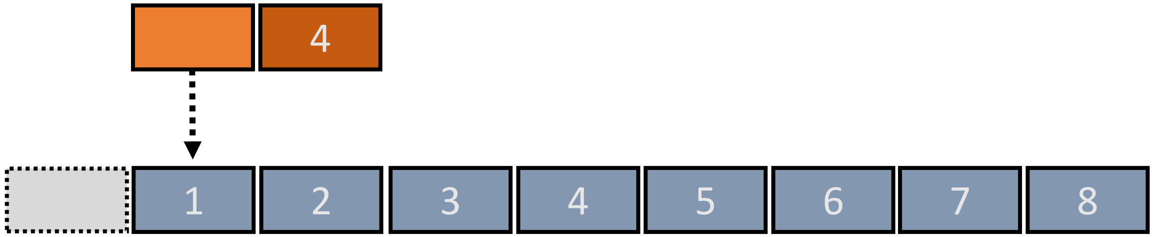
Returns: the number of elements in the container.

- vector capacity

Returns: The total number of elements that the vector can hold without requiring reallocation.

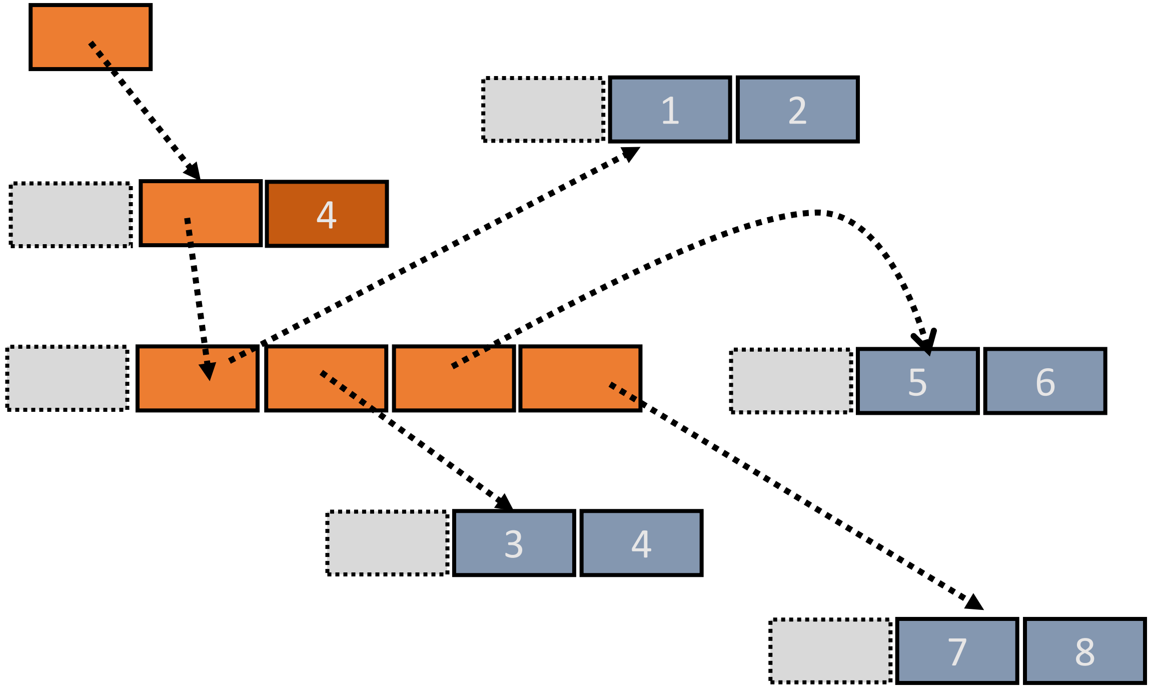
Compact representation

```
vector<Point> vp = {
   Point{1,2}, Point{3,4}, Point{5,6}, Point{7,8}
};
```



- Linked representation

  - Java, Python



Classes without virtual functions

- … I always maintained a clear view of what an object looked like in memory.

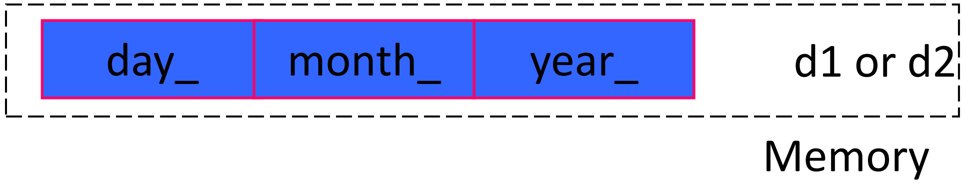
  - Bjarne Stroustrup

- A class without a virtual function is a simple C struct.
- Each class object maintains its copy of the class data members.

```
class Date {
  int day_, month_, year_;
public:
  Date(int = 0, int = 0, int = 0);
  int get_day() const { return day_; }
  void add_year(int);
  void add_month(int);
  void add_day(int);
  // Other member function(s)
} d1;
```

=

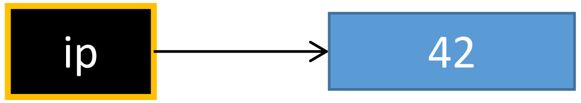
```
struct Date {
  int day_, month_, year_;
} d2;
```



- Inline methods, inline data
- A class without a virtual function requires exactly as much space to represent as a struct with the same data members.
- A compiler may add some “padding” between and after the members for alignment.

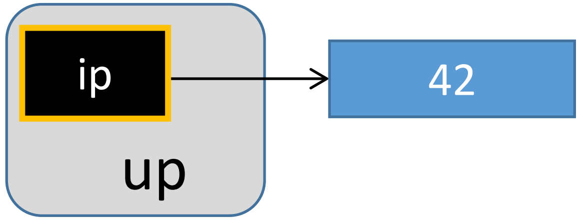
Light-weight abstractions- unique_ptr

- Raw pointer



```
{
    // …
    int* ip = new int{42};
    delete ip;
}
```

- Unique pointer



```
#include <memory>
{
        std::unique_ptr<int> up{new int{42}};
        // …
}
```

Talk to hardware directly

- Something has to talk to hardware. Not everything can be a virtual machine, someone has to write virtual machines.

Bjarne Stroustrup, GoingNative 2012, The Importance of being native panel

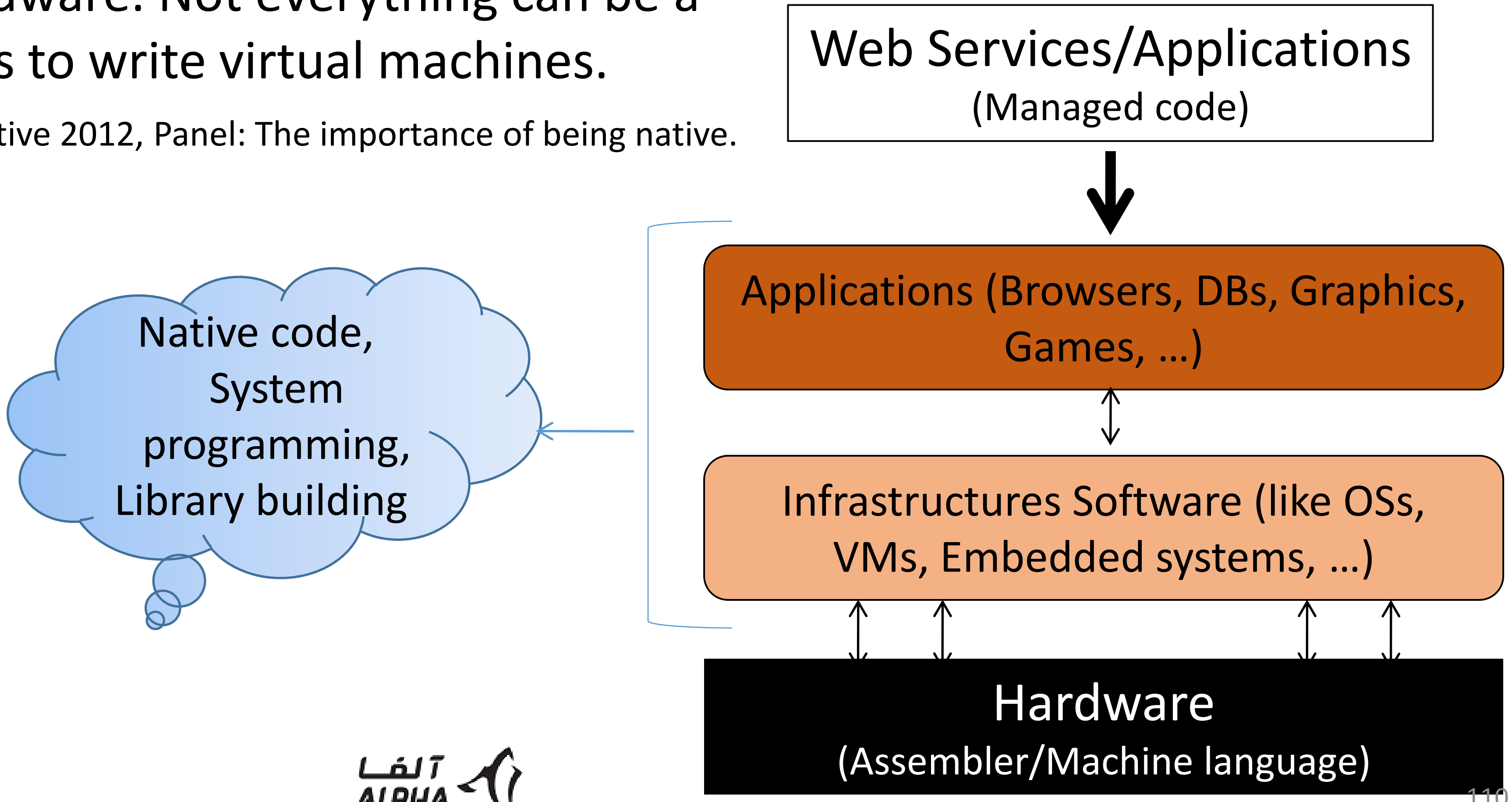
Talk to hardware directly

Low-level programming support rules:

Leave no room for a lower-level language below C++ (except assembler).

- Something has to talk to hardware. Not everything can be a virtual machine, somebody has to write virtual machines.

  - Bjarne Stroustrup. GoingNative 2012, Panel: The importance of being native.



Web Services/Applications (Managed code)

Applications (Browsers, DBs, Graphics, Games, …)

Infrastructures Software (like OSs, VMs, Embedded systems, …)

Hardware (Assembler/Machine language)

Native code, System programming, Library building

Light-weight abstractions

C++ is a suitable tool for implementation of *light-weight abstractions* in “small-scale” and *software infrastructure* in “large scale”.

- Light-weight abstractions: compact and efficient data structures

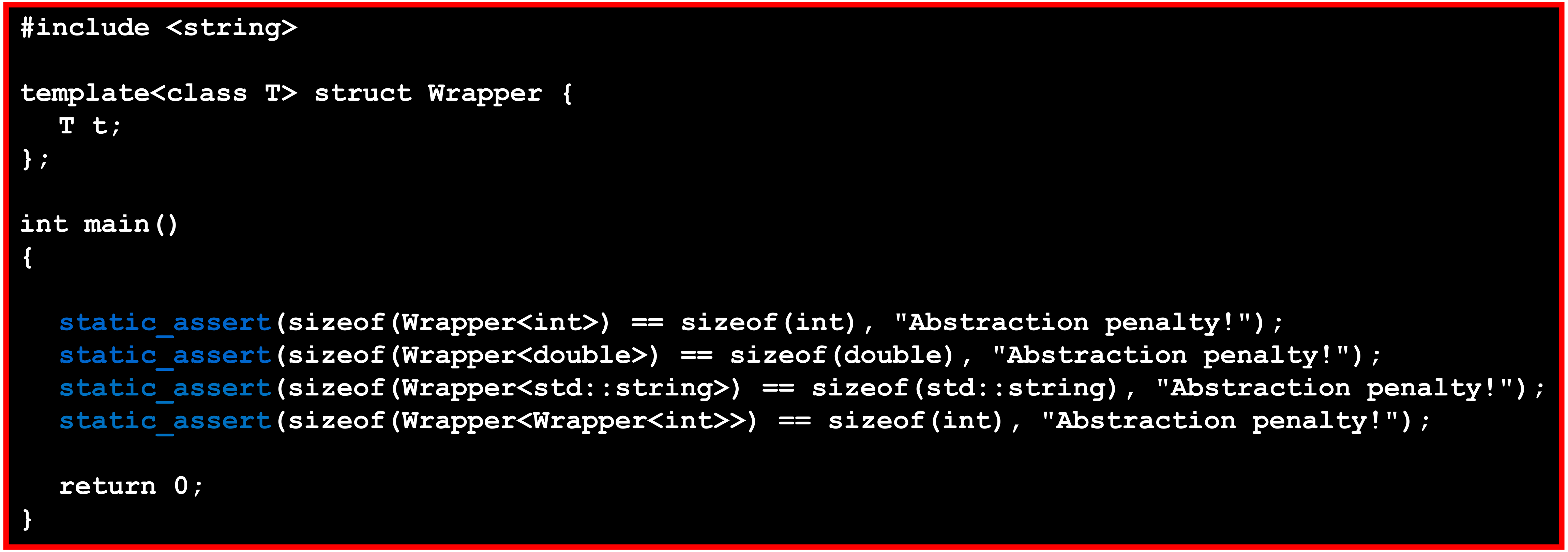
Zero-overhead abstractions

From Bjarne Stroustrup:

“C++ enables zero-overhead abstraction to get us away from the hardware without adding cost”

Zero-overhead abstractions- an example

- Type wrapper



```
#include <string>

template<class T> struct Wrapper {
   T t;
};

int main()
{

   static_assert(sizeof(Wrapper<int>) == sizeof(int), "Abstraction penalty!");
   static_assert(sizeof(Wrapper<double>) == sizeof(double), "Abstraction penalty!");
   static_assert(sizeof(Wrapper<std::string>) == sizeof(std::string), "Abstraction penalty!");
   static_assert(sizeof(Wrapper<Wrapper<int>>) == sizeof(int), "Abstraction penalty!");

   return 0;
}
```

How C++ combats global Warming?

How C++ combats global Warming?

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

Bjarne Stroustrup

How C++ combats global Warming?

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

Bjarne Stroustrup

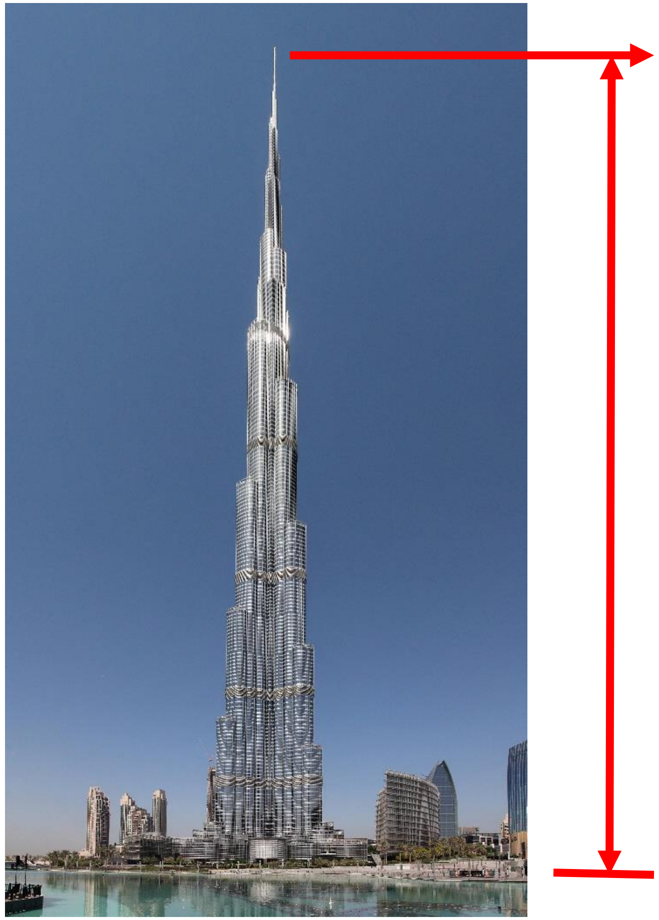
How C++ combats global Warming?

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

Bjarne Stroustrup

- My contribution to the fight against global warming is C++’s efficiency: Just think if Google had to have twice as many server farms! Each uses as much energy as a small town. And it’s not just a factor of two… Efficiency is not just running fast or running bigger programs, it’s also running using less resources.

The importance of low-latency: An example



- Burj Khalifa

Height:
- 828 meters
- 2,722 feet

- Speed of light:

- ~ 1 foot per ns

CPPCON 2017

When a Microsecond Is an Eternity: High Performance Trading Systems in C++

Carl Cook

- A very good minimum time (wire-to-wire) for a software-based trading system is around 2.5 us.
- That’s less than the time its takes light to travel from the top of the spire to the ground.

Low-level != Efficient

- Array elements accumulations: accumulate algorithm, traditional for loop, range-based for loop, for_each + lambda
- C++ sort algorithm vs. C qsort
- Language features + compiler + optimizer deliver performance

  for_each()+lambda vs. for-loop Examples like these give identical performance on several compilers:

```
sum = 0;
for(vector<int>::size_type i=0; i<v.size(); ++i) // conventional loop
    sum += v[i];
```

```
sum = 0;
for_each(v.begin(),v.end(),
[&sum](int x) {sum += x; }); // algorithm + lambda
```

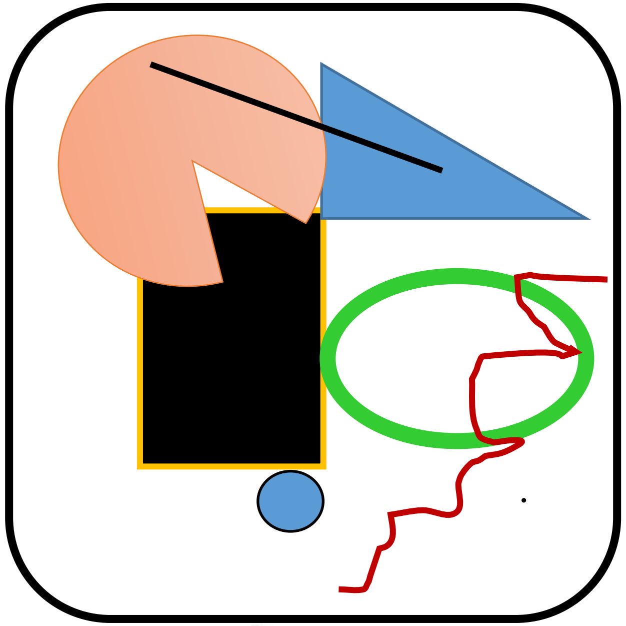
Compatibility with C

“As close as possible to C, but no closer.”

Using several abstractions in a 5 line of code

```
void rotate_and_draw(vector<Shape*>& vs, int r)
{
  for_each(vs.begin(),vs.end(), [](Shape* p) { p->rotate(r); }); // rotate all elements of vs
  for (Shape* p : vs) p->draw(); // draw all elements of vs
}
```

from Bjarne Stroustrup. Five popular myths about C++. http://www.stroustrup.com/Myths-final.pdf



- Is this *object-oriented*? Of course it is; it relies critically on a class hierarchy with virtual functions.
- It is *generic*? Of course it is; it relies critically on a parameterized container (**vector**) and the generic function **for_each**.
- Is this *functional*? Sort of; it uses a lambda (the **[]** construct).
- Is this *Procedural*? Of course it is, the function rotate_and_draw.
- It is modern C++: C++11.

References and further readings

- Bjarne Stroustrup. Foundations of C++. Proc. 22nd European Symposium on Programming (ESOP). Springer LNCS 7211. April 2012.
- Bjarne Stroustrup. Software Development for Infrastructure. Computer, vol. 45, no. 1, pp. 47-58, Jan. 2012.

Thanks for your patience …

A man who asks a question is a fool for minute,
The man who does not ask, is a fool for a life.
- Confucius

Learning to ask the right (often hard) questions is an essential part of learning to think as a programmer.
- Bjarne Stroustrup *programming Principles and Practice Using C++, page 4.*

There is no stupid question, but there is stupid answer.
- Howard Hinnant

Q & A