NORTHWEST INSTITUTE for ADVANCED COMPUTING

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle for the U.S. Department of Energy

UNIVERSITY of WASHINGTON

AMATH 483/583
High Performance Scientific Computing

Lecture 2:
Variables, functions, parameters

Andrew Lumsdaine
Northwest Institute for Advanced Computing
Pacific Northwest National Laboratory
University of Washington
Seattle, WA

Administrivia

- PS1 to be posted this afternoon
- Sign up for piazza (link on canvas)
- Sign up for course notebook (canvas / onenote)

Overview

- Recap of Lecture 1
- Types and variables
- Namespaces
- Functions and procedural abstraction
- Parameter passing

SC'19 Student Cluster Competition Call-Out!

- Teams work with advisor and vendor to design and build a cutting-edge, commercially available cluster constrained by the 3000-watt power limit
- Cluster run a variety of HPC workflows, ranging from being limited by CPU performance to being memory bandwidth limited to I/O intensive
- Teams are comprised of six undergrad or high-school students plus advisor

https://sc19.supercomputing.org/program/studentssc/student-cluster-competition/

Informational meeting:
Tu 5PM-6PM Allen 203
Th 5PM-6PM Allen 203

In Our Last Exciting Episode

- "High Performance" = bigger, better, faster, more
  - Use resources as efficiently as possible
  - Starting with single core
  - Use multiple cores/sockets/boards/blades/nodes/racks
- Moore's law → miniaturization → higher clock rates → performance
  - "Denard scaling" ended c. 2005
- Moore's law → miniaturization → more transistors → more cores
- Gordon Moore, Seymour Cray, Thomas Sterling
- Top 500 List (top500.org)

Top 500 November 2017

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
4	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890

Top500 November 2017

5	DOE/SC/LBNL/NERSC United States	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray Inc.	622,336	14,014.7	27,880.7	3,939
6	Joint Center for Advanced High Performance Computing Japan	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path Fujitsu	556,104	13,554.6	24,913.5	2,719
7	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
8	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray Inc.	206,720	9,779.0	15,988.0	1,312

Top500 November 2018

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	DOE/SC/Oak Ridge National Laboratory United States	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM	2,397,824	143,500.0	200,794.9	9,783
2	DOE/NNSA/LLNL United States	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM / NVIDIA / Mellanox	1,572,480	94,640.0	125,712.0	7,438
3	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
4	National Super Computer Center in Guangzhou China	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 NUDT	4,981,760	61,444.5	100,678.7	18,482

1.5M cores

10M cores

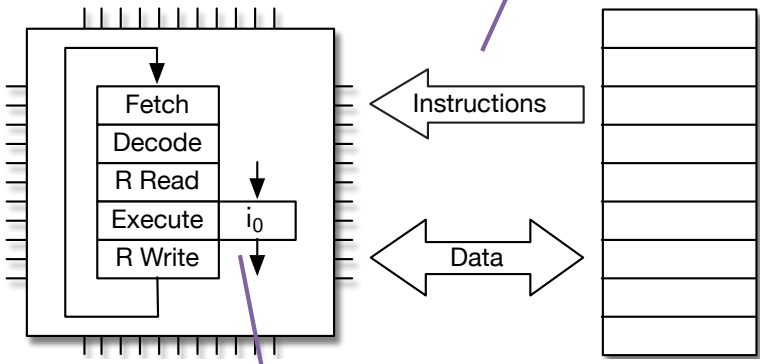
Top500 November 2018

5	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray Inc.	387,872	21,230.0	27,154.3	2,384
6	DOE/NNSA/LANL/SNL United States	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray Inc.	979,072	20,158.7	41,461.2	7,578
7	National Institute of Advanced Industrial Science and Technology (AIST) Japan	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR Fujitsu	391,680	19,880.0	32,576.6	1,649
8	Leibniz Rechenzentrum Germany	**SuperMUC-NG** - ThinkSystem SD530, Xeon Platinum 8174 24C 3.1GHz, Intel Omni-Path Lenovo	305,856	19,476.6	26,873.9	

Scaling progression of CPUs

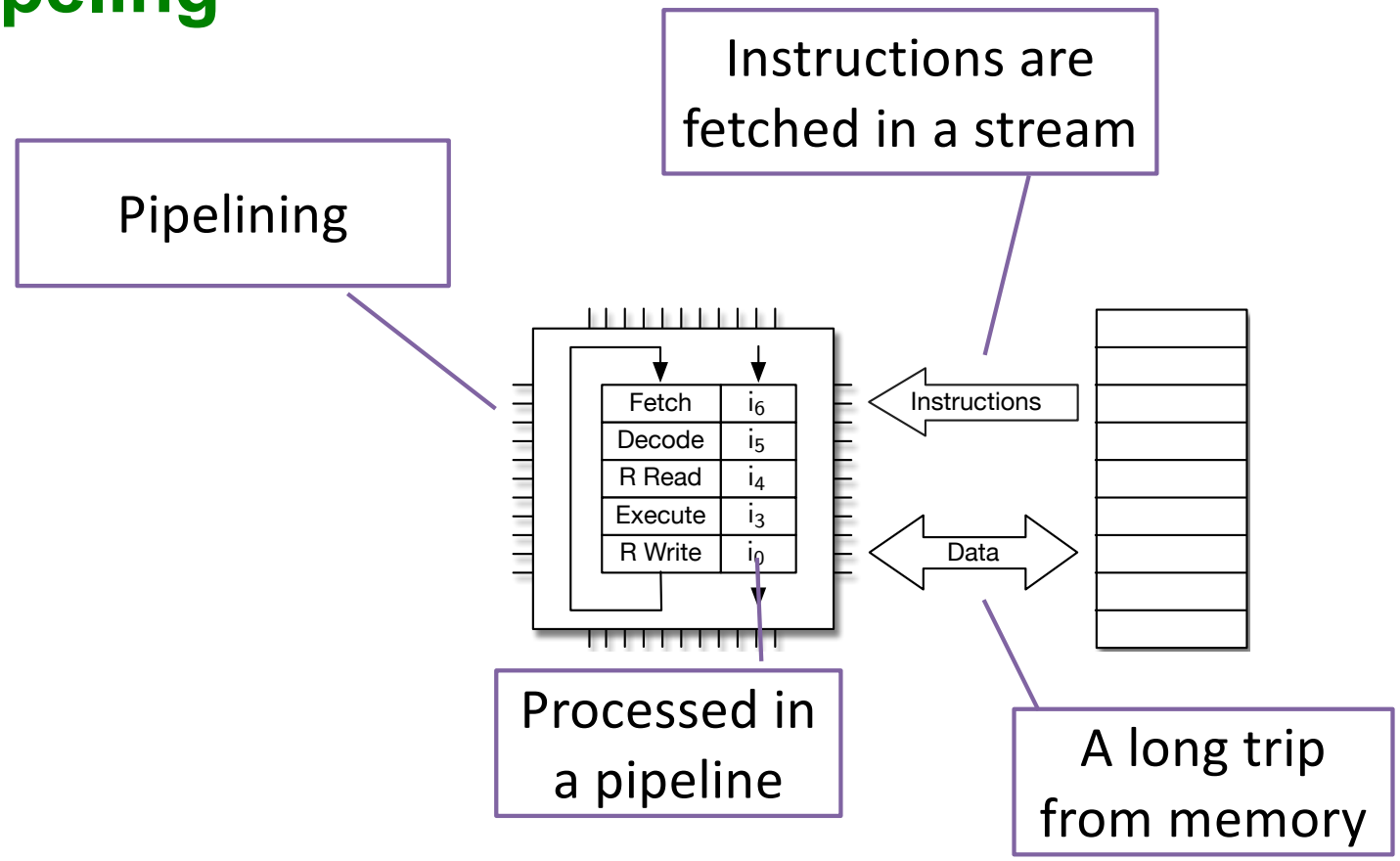
Simplest model

CPU fetches and executes instructions



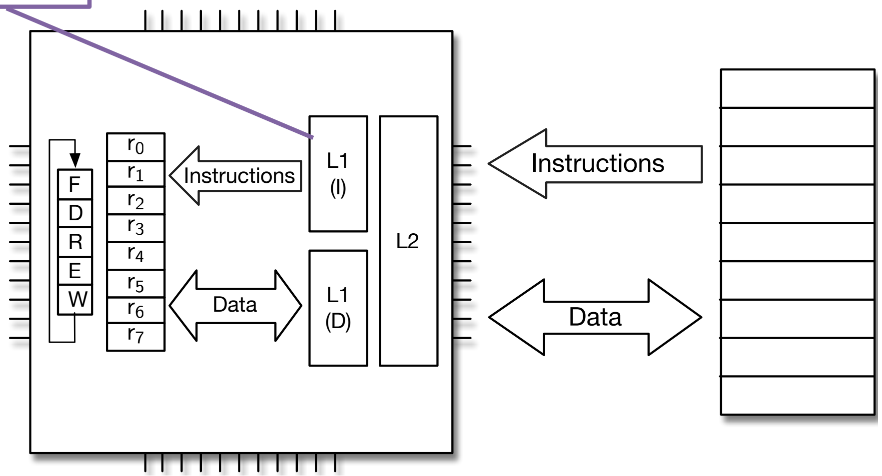
Many cycles per instruction

Pipeling


Instructions are fetched in a stream
Pipelining
Fetch
Decode
R Read
Execute
R Write
i_6
i_5
i_4
i_3
i_0
Instructions
Data
Processed in a pipeline
A long trip from memory

Hierarchical memory

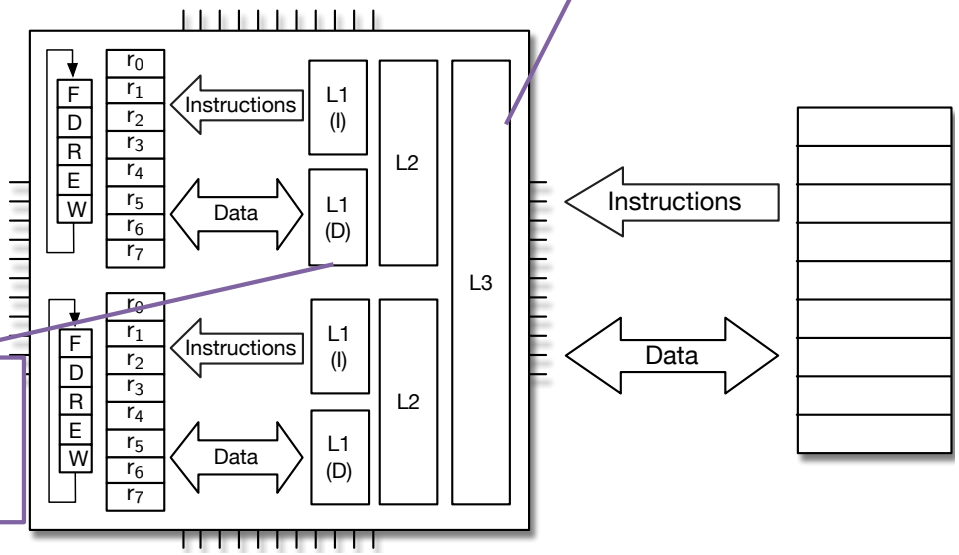
Use special, fast memory to keep data and instructions close



Multicore CPUs

Replicate 2X

Cores share slower memory



Caches need to be kept coherent

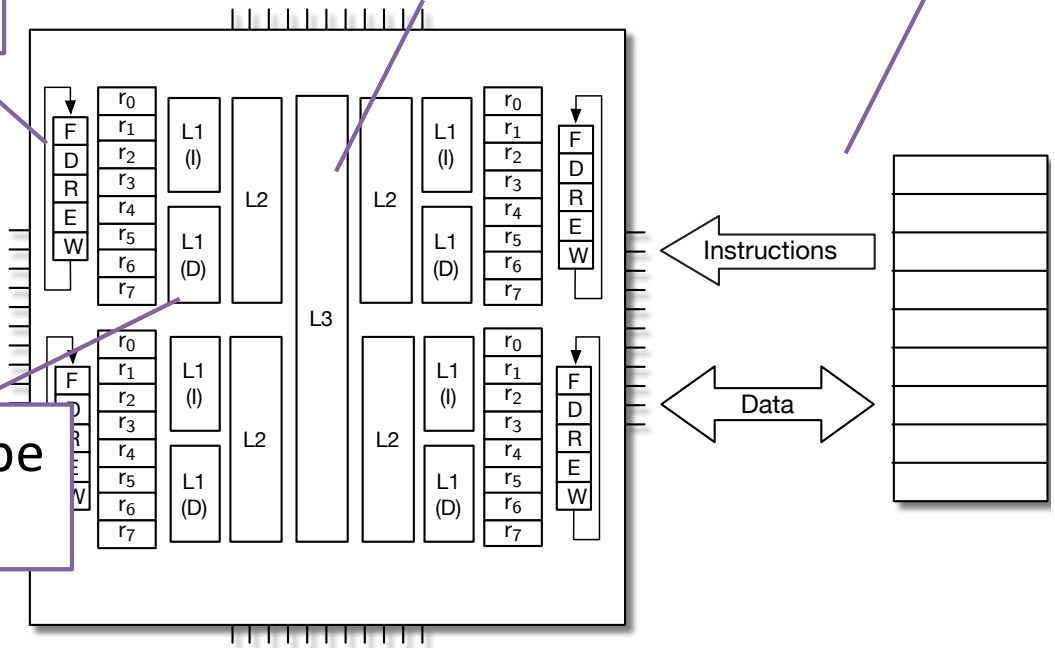
Even more cores

Replicate 4X

Cores share slower memory

Include super-slow DRAM

Caches need to be kept coherent

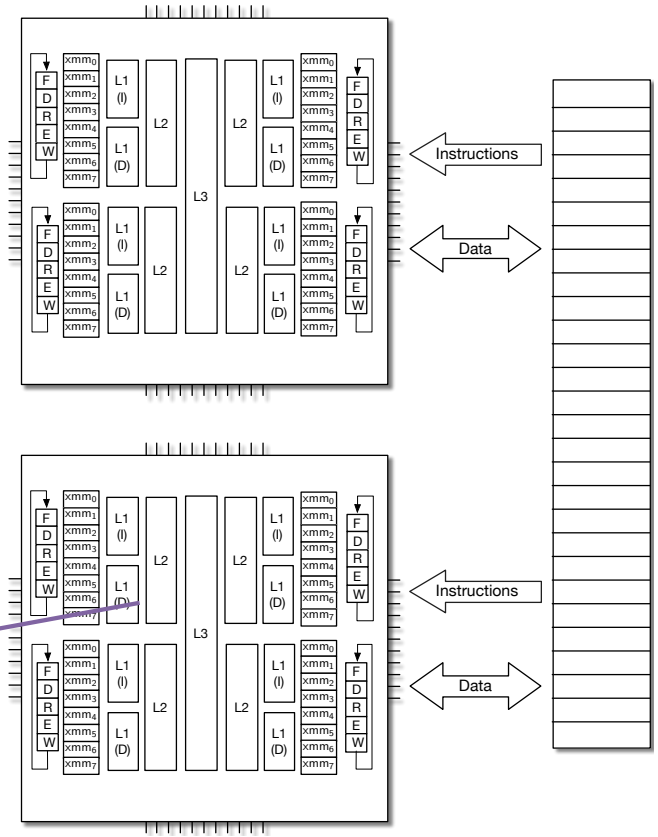


Symmetric Multi-Processor (SMP)

Multiple CPU chips

AKA “sockets”

Caches still need to be kept (somewhat) coherent



Memory may be uniformly shared among sockets

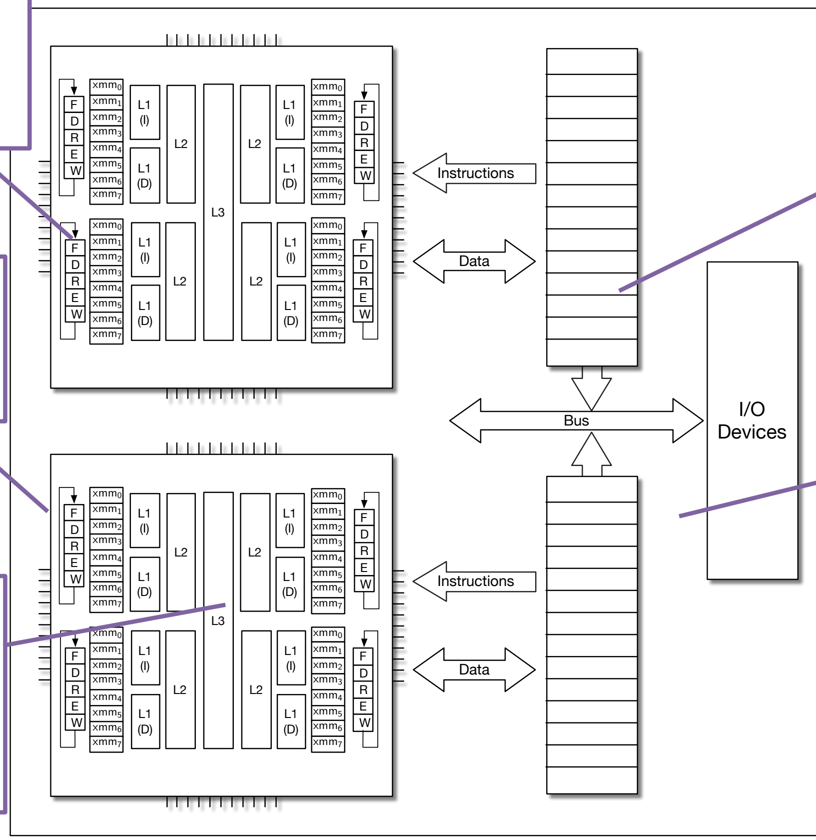
Uniform memory access (UMA)

Asymmetric

Multiple CPU chips

AKA "sockets"

Caches still need to be kept (somewhat) coherent: CC-NUMA



Memory may be non-uniformly shared among sockets

Non-uniform memory access (NUMA – most common)

Putting it All Together

Put sockets on a blade

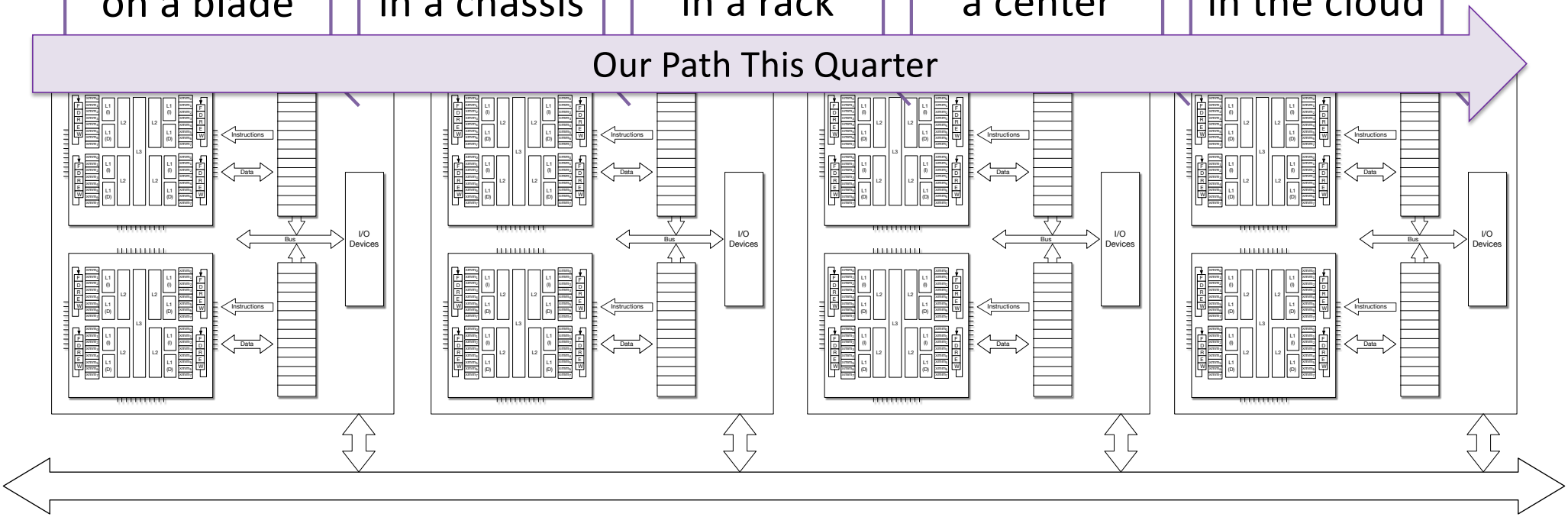
Put blades in a chassis

Put chassis in a rack

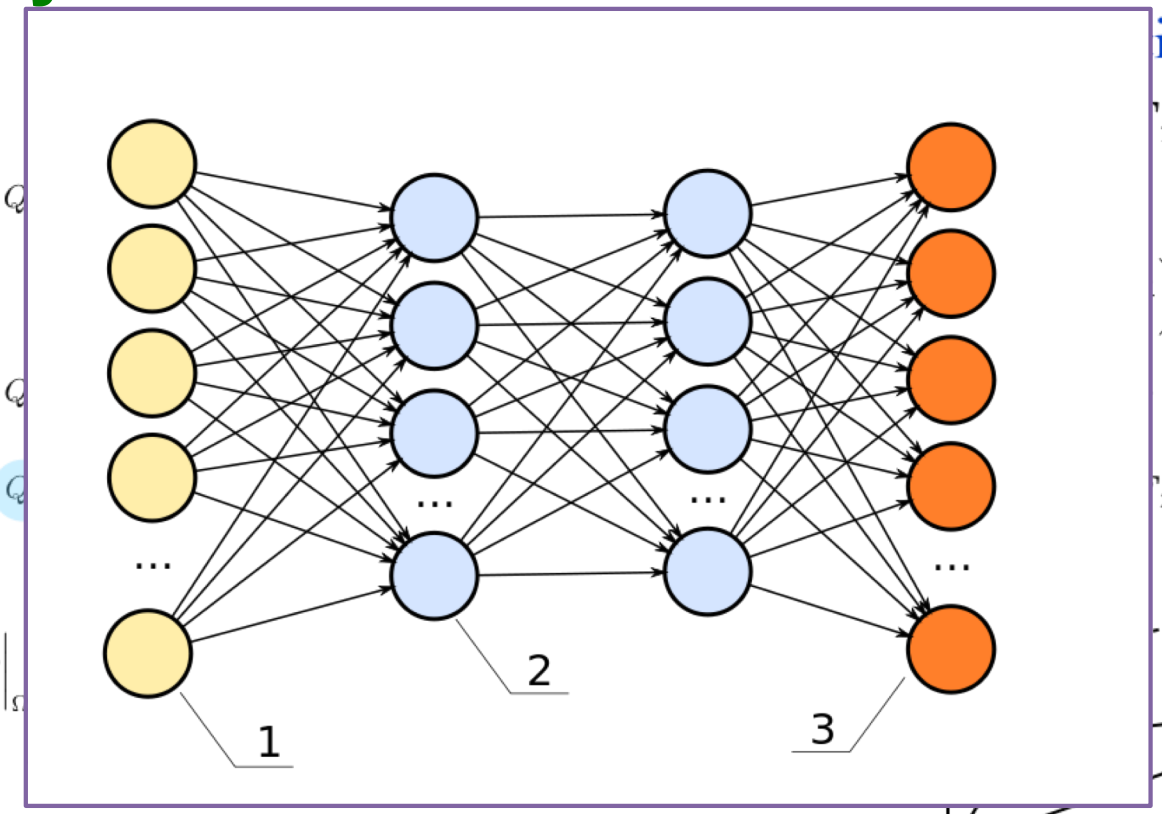
Put racks in a center

Put centers in the cloud

Our Path This Quarter



Multiphysics Solver



ive split

$_p\boldsymbol{F}_T$

$T\rho_0\left(\frac{\partial\dot{\psi}_e}{\partial T}\right)$

$_p\frac{\partial \boldsymbol{F}_T}{\partial T}+$

$\boldsymbol{F}$

Eulerian

$\boldsymbol{u}$

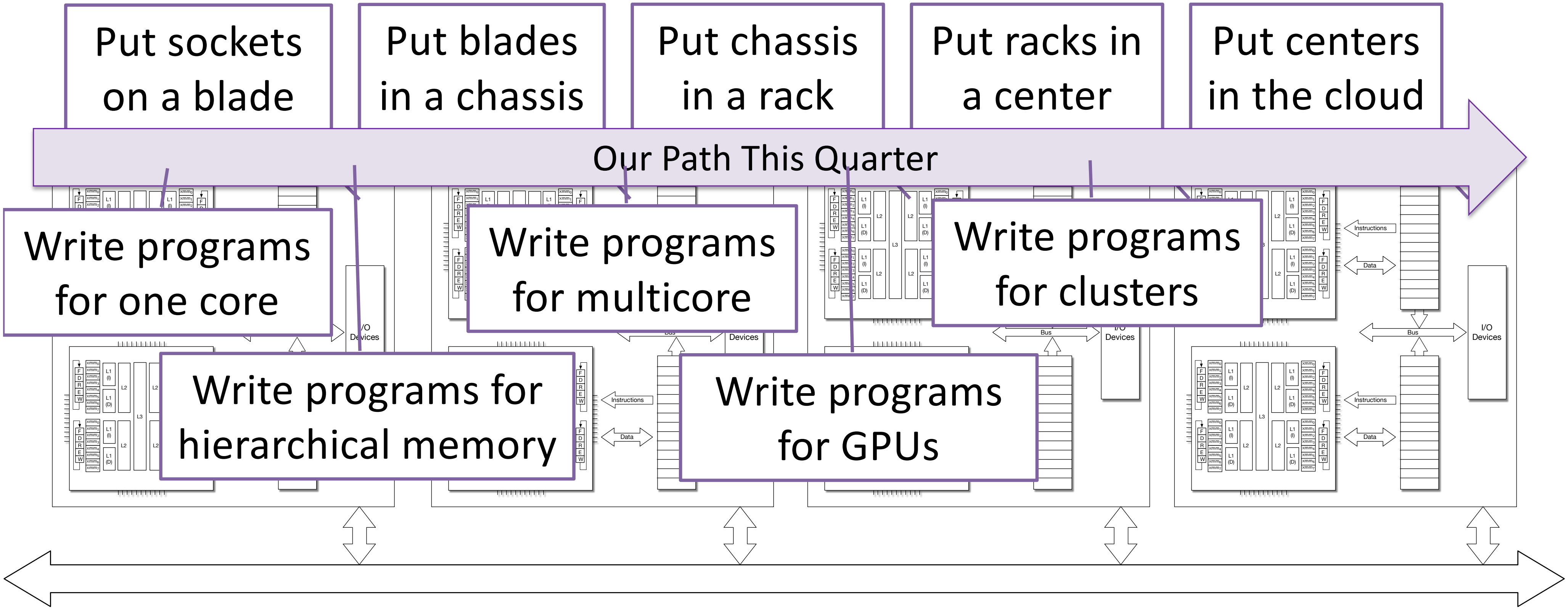
Ω_t

$\boldsymbol{x}=\boldsymbol{\gamma}(\boldsymbol{X},t)$

$\boldsymbol{Q}=-\boldsymbol{F}^{-1}\boldsymbol{\kappa}\boldsymbol{F}^{-1}\nabla_0 T$

X_1 X_3

Putting it All Together



Write A Program

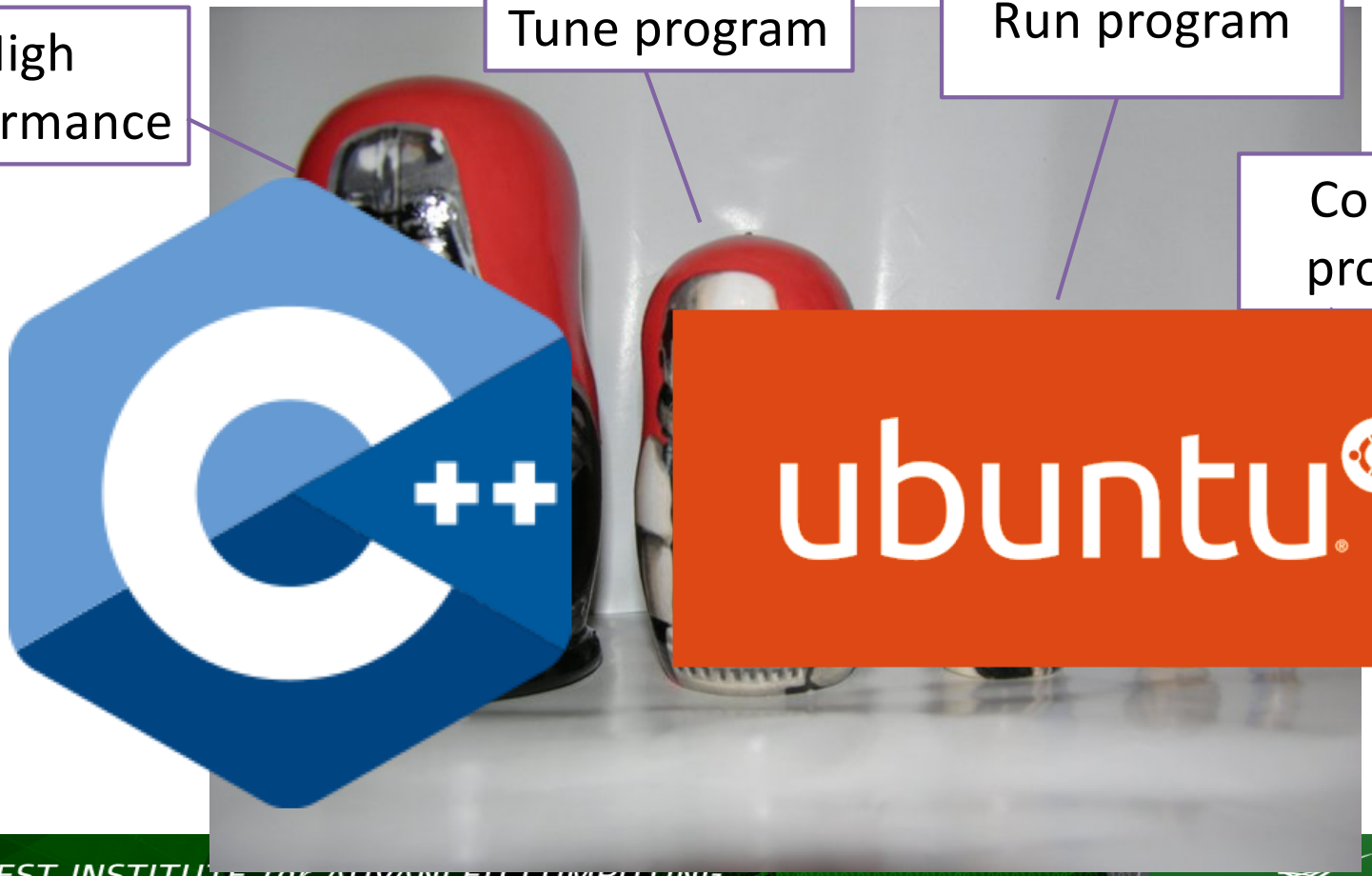
High performance

Measure and Tune program

Run program

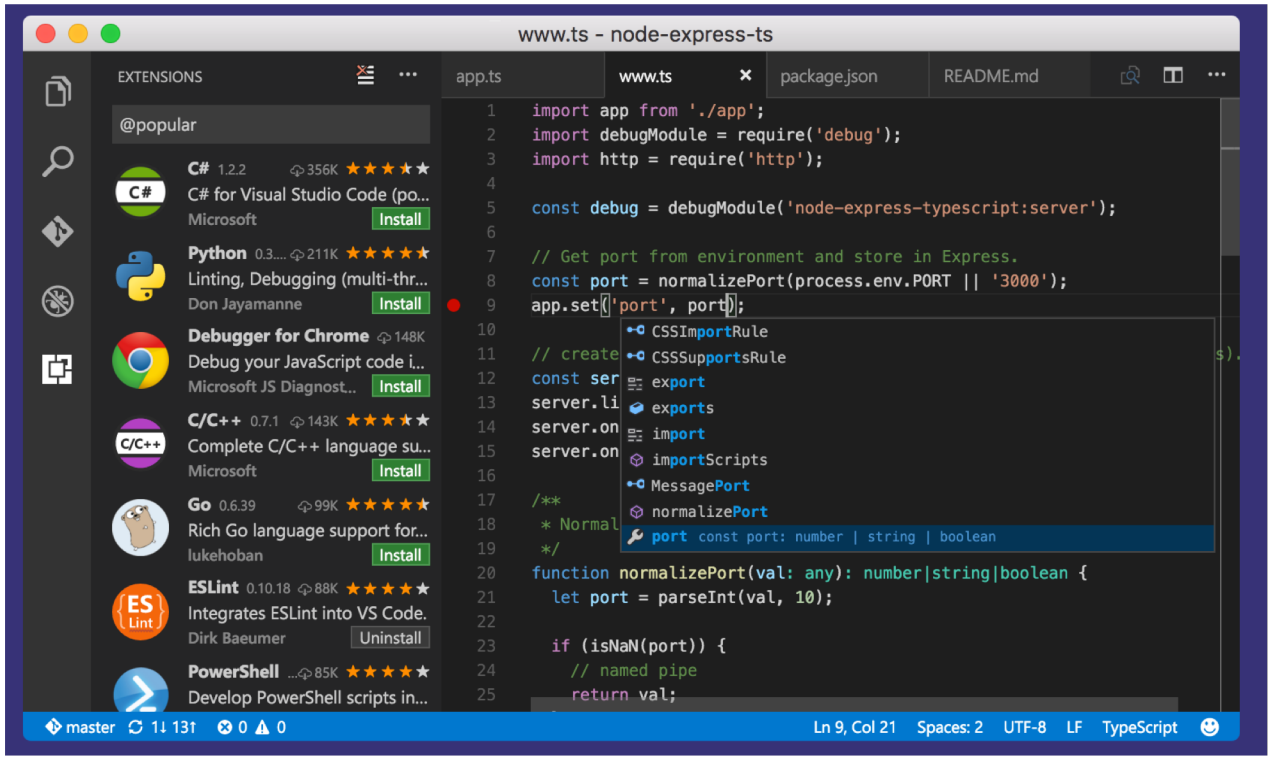
Compile program

Write program



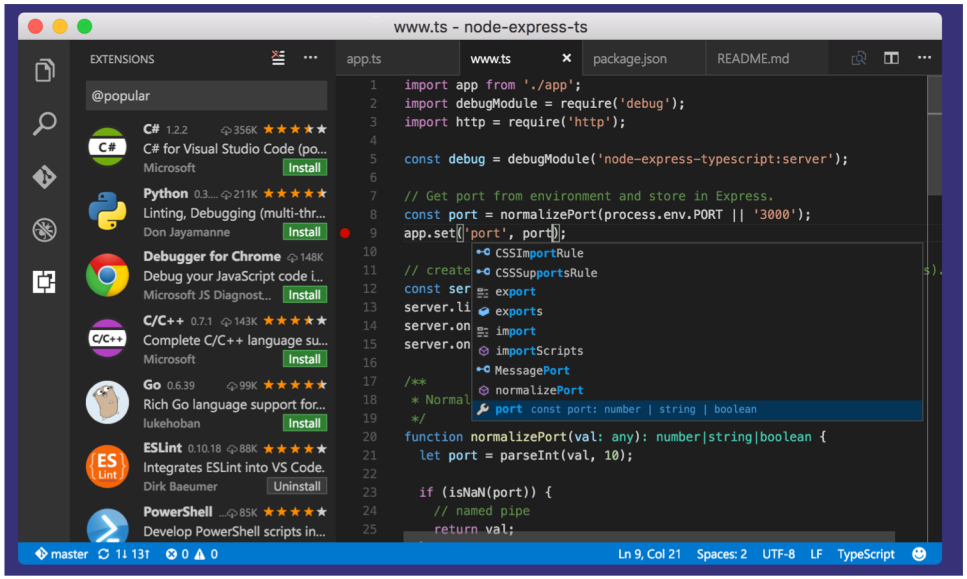
Developing your code

- That includes (especially) mental labor
- Use productivity tools
- ***VS code*** (rec'd), Atom, Eclipse

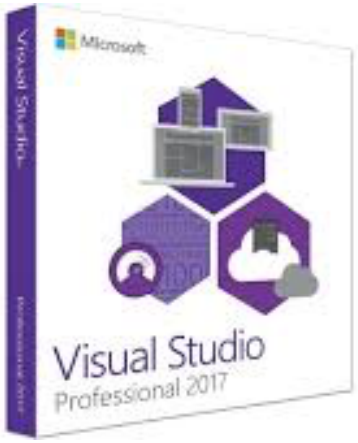


The AMATH 483/583 Development Environment

- Windows 10 (Pro and Education) and Mac OS X can be made to approximate Linux
  - Windows Subsystem for Linux (configured per course guidelines)
  - Xcode command-line tools (configured per course guidelines)
- How-to documentation in problem set and on line



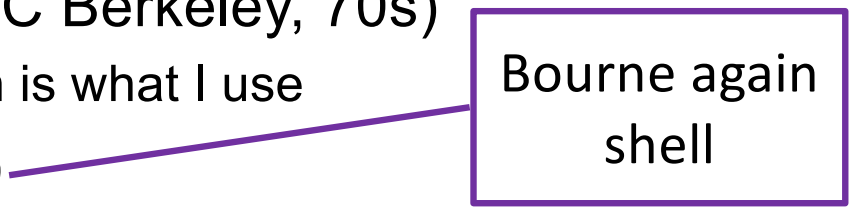
What about …?



- Muscle memory for typing is not the same as productivity (know the difference)
  - Stretch yourself
- Use any environment where you are most productive
- We can only support one (VS code + clang + “Linux”)
- Assignments must work with autograder

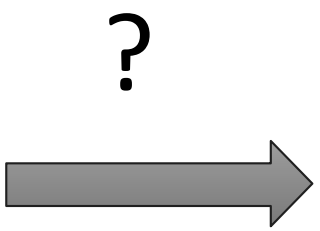
shells

- sh: “Bourne shell” (Stephen Bourne, Bell Labs c.1977)
- ksh: Korn shell (David Korn, Bell Labs, c. 1983)
- csh: C shell (Bill Joy, UC Berkeley, 70s)
  - and cousin tcsh – which is what I use
- bash (Brian Fox, 1989)
  - who knows what this stands for (without searching)
- All are Linux (Unix) processes with read-eval-print loops
- But also complete systems scripting language for dealing with Unix
  - Unix philosophy: data in text format, small programs using text I/O

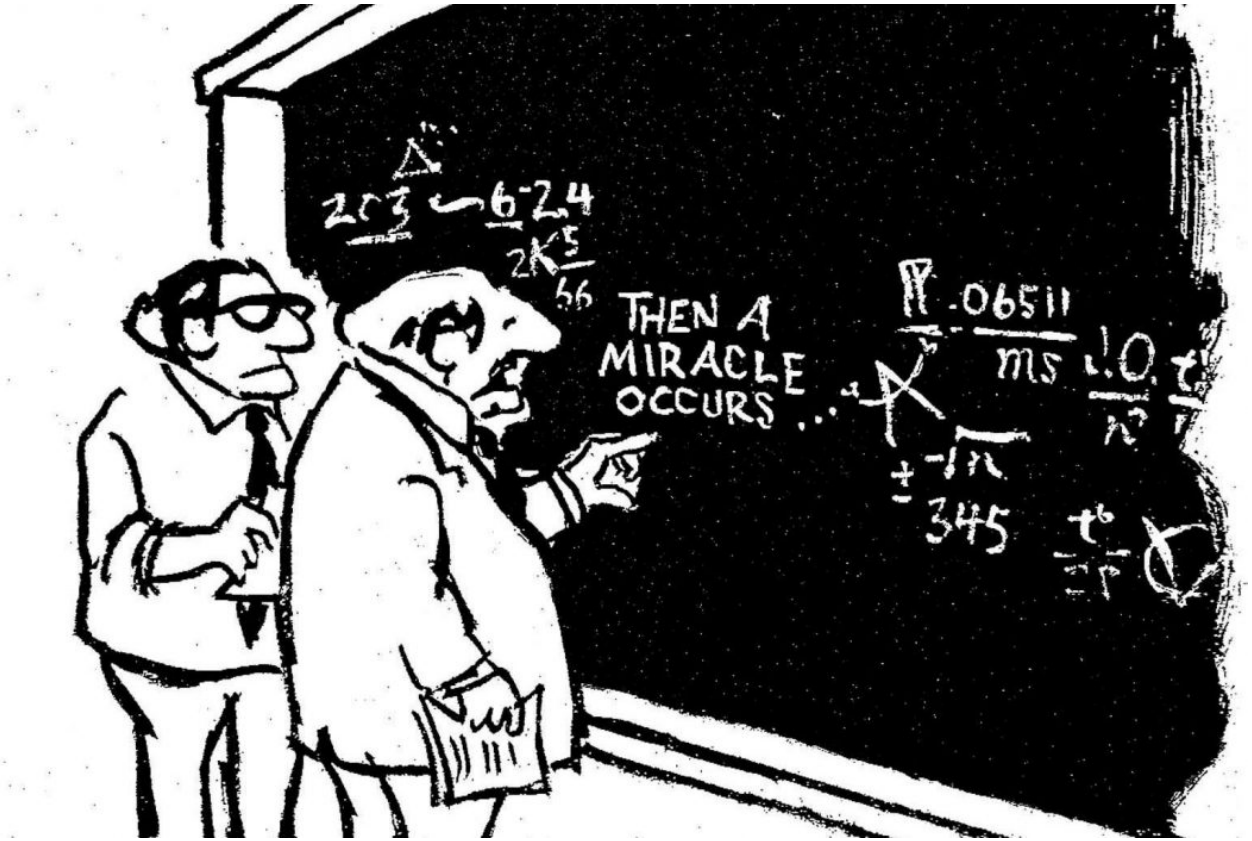


Programs and Programming

```
int main() {
  int a = 1;
  double x = 0.3;
  foo(x,a);
}
```



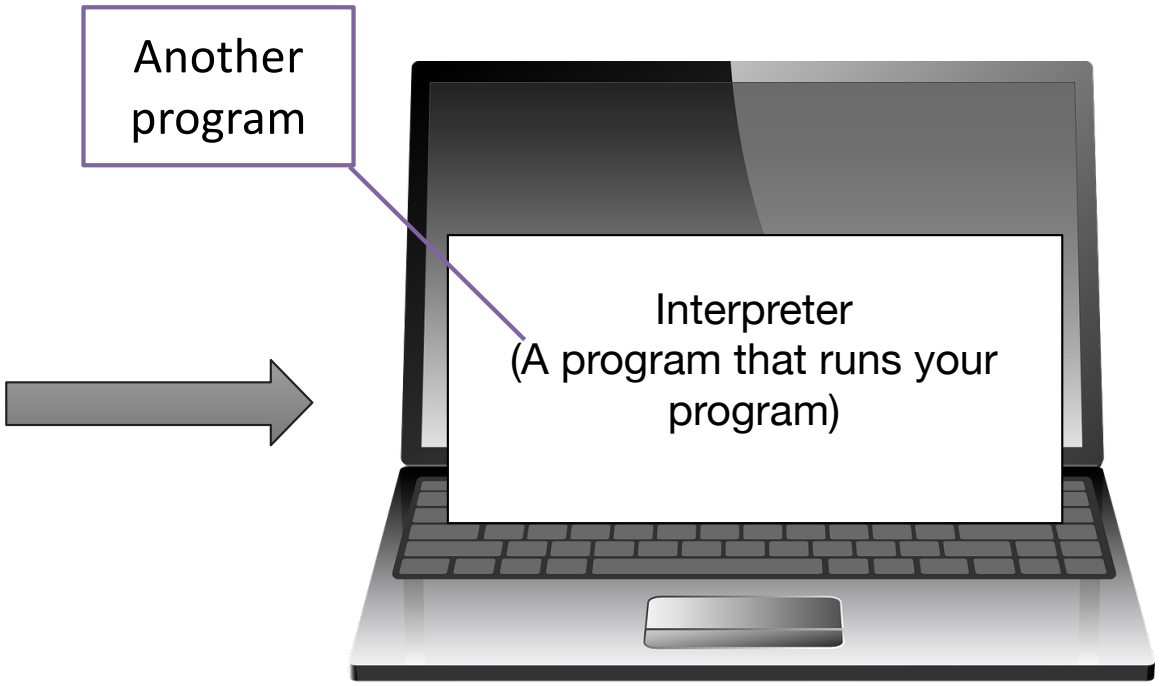
Programs and Programming



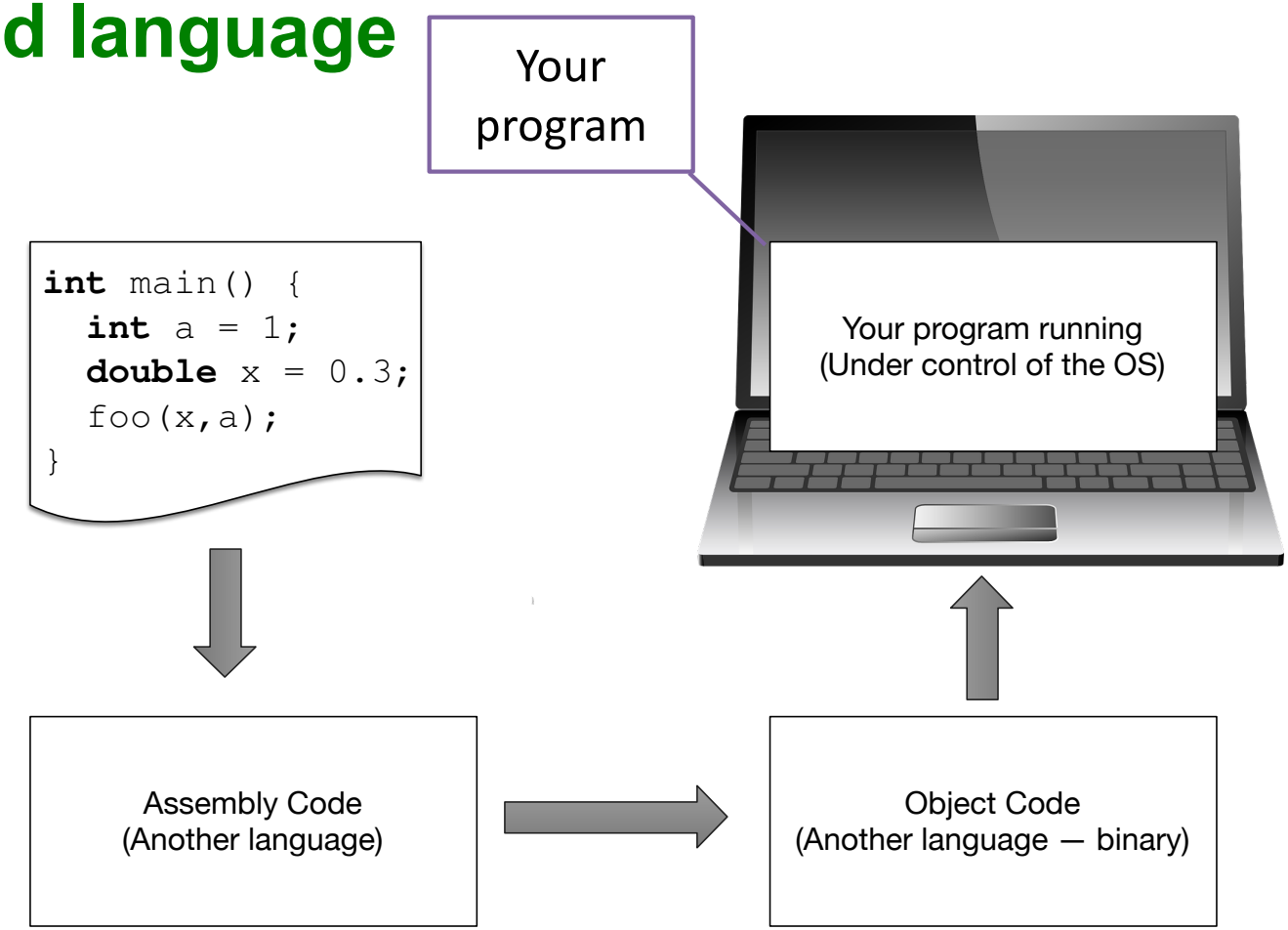
© S. Harris

Interpreted language (Python)

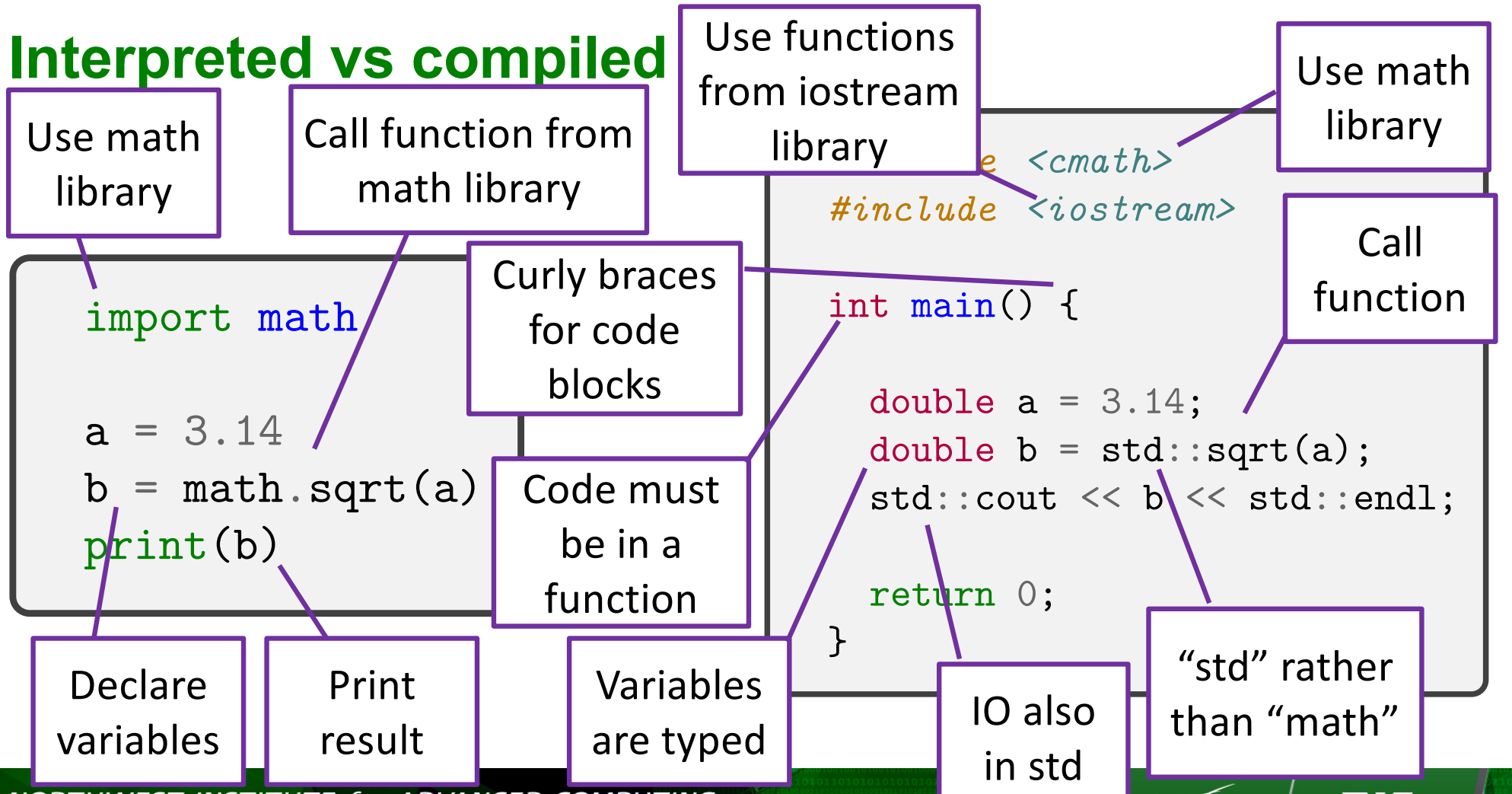
```
import math

a = 3.14
b = math.sqrt(a)
print(b)
```



Compiled language



Interpreted vs compiled



Compilation

```
#include <cmath>
#include <iostream>

int main() {

  double a = 3.14;
  double b = std::sqrt(a);
  std::cout << b << std::endl;

  return 0;
}
```

You can't run this code

An "executable"

It needs to be turned into code that can run

Multi-step process

Compile to **object file**

Then link in libraries for sqrt and IO

Bits just for this code

Compiling

- To compile one source file to an executable
  - $ c++ filename.cpp
  - (What is the name of the executable?)
- To compile multiple source files to an executable
  - $ c++ one.cpp two.cpp three.cpp
- To create an object file
  - $ c++ -c one.cpp –o one.o
- To create an executable from multiple object files
  - $ c++ one.o two.o three.o –o myexecutable

Slice of C++

- C++11 (C++14, C++17, C++20) are quite modern languages
- But C++11 (et al) and libraries are ***huge***
- We will use a focused slice of C++11
- Use some modern features
- Avoid legacy features (such as pointers)
- Avoid modern features (OO)

```
#include <cmath>
#include <iostream>

int main() {

  double a = 3.14;
  double b = std::sqrt(a);
  std::cout << b << std::endl;

  return 0;
}
```

C++ development philosophy

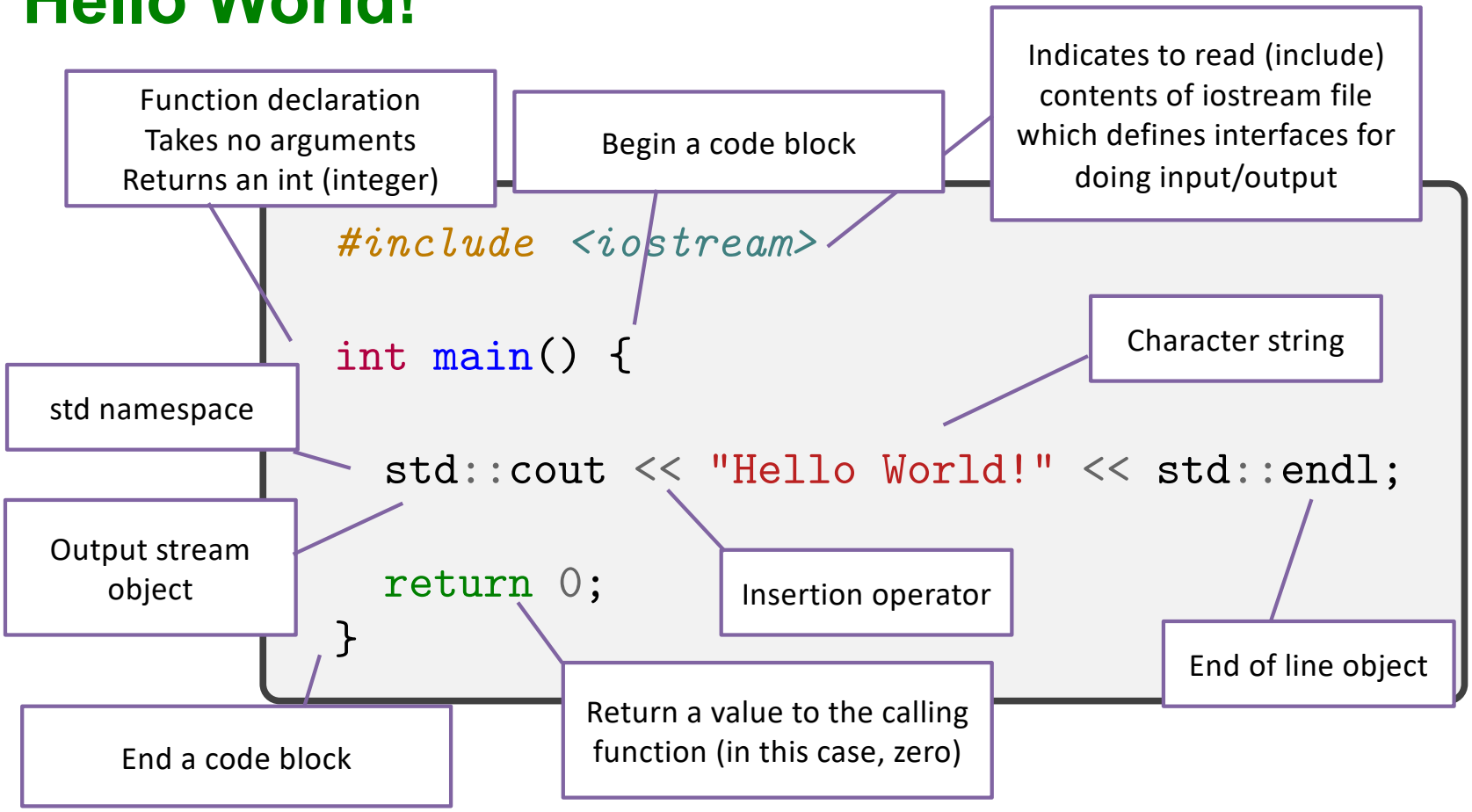
From C++ Core Guidelines

Only one rule about C++

[P.1: Express ideas directly in code]
[P.2: Write in ISO Standard C++]
[P.3: Express intent]
[P.4: Ideally, a program should be statically type safe]
[P.5: Prefer compile-time checking to run-time checking]
[P.6: What cannot be checked at compile time should be checkable at run time]
[P.7: Catch run-time errors early]
[P.8: Don't leak any resources]
[P.9: Don't waste time or space]
[P.10: Prefer immutable data to mutable data]
[P.11: Encapsulate messy constructs, rather than spreading through the code]
[P.12: Use supporting tools as appropriate]
[P.13: Use support libraries as appropriate]

Many follow from the two simple rules

Hello World!


Function declaration
Takes no arguments
Returns an int (integer)
Begin a code block
Indicates to read (include) contents of iostream file which defines interfaces for doing input/output
#include <iostream>
int main() {
std::cout << "Hello World!" << std::endl;
return 0;
}
Character string
std namespace
Output stream object
Insertion operator
End of line object
End a code block
Return a value to the calling function (in this case, zero)

#include

- Pulls text of <file> into source file
- Usually contain declarations and definitions of types, functions, etc
  - External libraries or other source files

```
//

int main() {

  std::cout << "Hello World!" << std

  return 0;
}
```

No includes

```
$ g++ hello.cpp
hello.cpp:3:3: error: use of undeclared identifier 'std'
  std::string message = "Hello World";
  ^
hello.cpp:4:3: error: use of undeclared identifier 'std'
  std::cout << message << std::endl;
  ^
hello.cpp:4:27: error: use of undeclared identifier 'std'
  std::cout << message << std::endl;
                          ^
3 errors generated.
```

Comments

```
// This is a comment
/*
  This is also a comment
*/
//  This is a
//  multiline comment
int main() {

  std::cout << "Hello World!" << std::endl

  return 0;  // main should always return 0
}
```

Delimit to end of line using //

Can also use C-style /* */ (disouraged)

Style for multiline comments

In-line comments

#include

- Pulls text of <file> into source file
- Usually contain declarations and definitions of types, functions, etc
  - External libraries or other source files

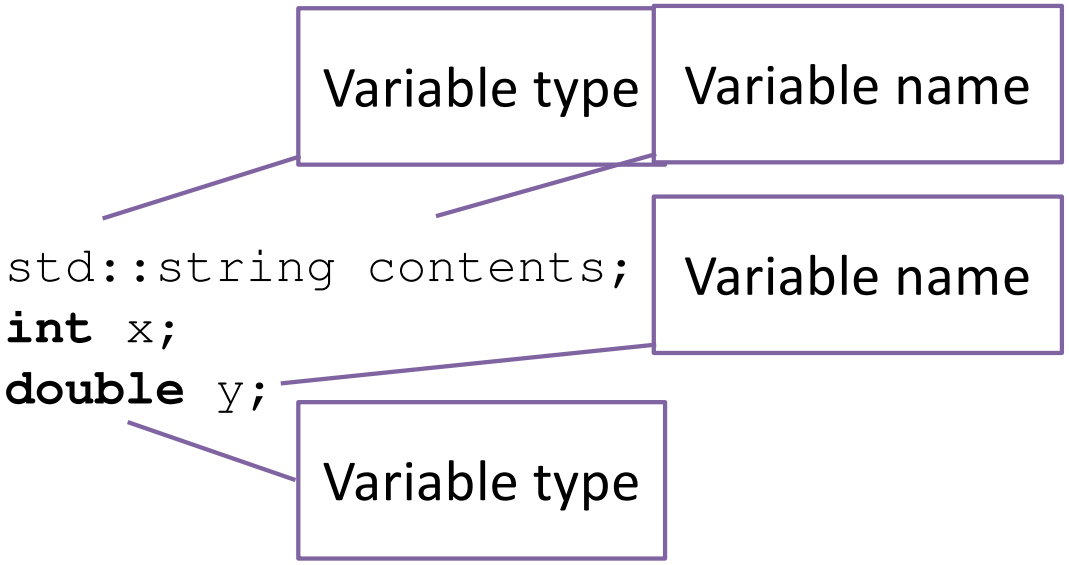
```
#include <algo

int main() {

  std::cout << "Hello World!" << std::end

  return 0;  // main should always return
}
```

Wrong include

```
$ c++ aa.cpp
aa.cpp:5:8: error: no type named 'string' in namespace 'std'
 std::string message = "Hello World";
 ~~~~~^
aa.cpp:6:8: error: no member named 'cout' in namespace 'std'; did you mea
 std::cout << message << std::endl;
 ~~~~~^~~~
     count
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/bits/
    'count' declared here
   count(_InputIterator __first, _InputIterator __last, const _Tp& __value)
   ^
aa.cpp:6:32: error: no member named 'endl' in namespace 'std'
 std::cout << message << std::endl;
                         ~~~~~^
3 errors generated.
```

Types

- Variable definition



- C++ has many **built-in** types: int, double, char, etc
- Other types are defined for **libraries** (accessed via #include)
- Almost always **class** definitions

Declaring and Initializing Variables

- In the old days variables were declared at the beginning of a block

```
int main() {
  double x, y;
  // ...
  x = 3.14159;
  y = x * 2.0;
  // ...
  return 0;
}
```

Declaration

Use

- Now they can be defined anywhere in the block

```
int main() {
  // ...
  double x = 3.14159;
  double y = x * 2.0;
  // ...
  return 0;
}
```

Declaration with initialization

- Best practice: Don't declare variables before they are needed and **always** initialize if possible

Namespaces

```
#include <iostream>

double pi = 3.14;

int main() {

  std::cout << "The legislated value of pi is ";
  std::cout << pi << std::endl;

  return 0;
}
```

To print this value

```
$ ./a.out
The legislated value of pi is 3.14
```

Compiler needs to ***resolve*** this symbol

Namespaces

```cpp
#include <iostream>

namespace amath583 {
  double pi = 3.14;
};

int main() {

  std::cout << "The legislated value of pi is "
  std::cout << pi << std::endl;

  return 0;
}
```

```
ns.cpp:11:16: error: use of undeclared identifier 'pi'; did you mean 'amath583::pi'?
  std::cout << pi << std::endl;
               ^~
               amath583::pi
ns.cpp:5:10: note: 'amath583::pi' declared here
  double pi = 3.14;
         ^
1 error generated.
```

pi is hidden in a namespace

But can't

Compiler needs to resolve this symbol

Namespaces

```
#include <iostream>

namespace amath583 {
  double pi = 3.14;
};

int main() {

  std::cout << "The legislated value of pi is ";
  std::cout << amath583::pi << std::endl;

  return 0;
}
```

```
$ ./a.out
The legislated value of pi is 3.14
```

Look for the variable pi in the amath583 namespace

Namespaces

```
#include <iostream>

namespace amath583 {
  double pi = 3.14;
};

using amath583::pi;

int main() {

  std::cout << "The legislated value of pi is ";
  std::cout << pi << std::endl;

  return 0;
}
```

We can also tell the compiler to look for pi in the namespace this way

```
$ ./a.out
The legislated value of pi is 3.14
```

Use “pi” where it can be found (no specified namespace)

Namespaces

```
#include <iostream>

namespace amath583 {
  double pi = 3.14;
};

using amath583;

int main() {

  std::cout << "The legislated value of pi is ";
  std::cout << pi << std::endl;

  return 0;
}
```

Everything in this namespace can be found

Open the namespace

Use “pi” where it can be found

Namespaces

```cpp
#include <iostream>

namespace amath483 {
  double pi = 3.14159;
};

namespace amath583 {
  double pi = 3.14;
};

int main() {

  std::cout << "The legislated value of pi is ";
  std::cout << pi << std::endl;

  return 0;
}
```

```
n2.cpp:18:16: error: reference to 'pi' is ambiguous
  std::cout << pi << std::endl;
               ^
n2.cpp:5:10: note: candidate found by name lookup is 'amath483::pi'
  double pi = 3.14159;
         ^
n2.cpp:9:10: note: candidate found by name lookup is 'amath583::pi'
  double pi = 3.14;
         ^
1 error generated.
```

Can't find pi

How to resolve ambiguity?

What happens here?

Namespaces

```
#include <iostream>

namespace amath483 {
  double pi = 3.14159;
};

namespace amath583 {
  double pi = 3.14;
};

using namespace amath483::pi;

int main() {

  std::cout << "The legislated value of pi is ";
  std::cout << pi << std::endl;

  return 0;
}
```

```
$ ./a.out
The legislated value of pi is 3.14159
```

Specify that this pi is in the global namespace

Namespaces

```
#include <iostream>

namespace amath483 {
  double pi = 3.14159;
};

namespace amath583 {
  double pi = 3.14;
};

using namespace amath483::pi;

int main() {

  std::cout << "The legislated value of pi is ";
  std::cout << amath583::pi << std::endl;

  return 0;
}
```

```
$ ./a.out
The legislated value of pi is 3.14
```

(This one)

Specify exactly which pi

Namespaces

```
#include <iostream>

namespace amath483 {
  double pi = 3.14159;
};

namespace amath583 {
  double pi = 3.14;
};

using namespace amath483::pi;

int main() {

  std::cout << "The legislated value of pi is ";
  std::cout << amath583::pi << std::endl;

  return 0;
}
```

(This one)

Specify exactly which pi

Disambiguating

```
#include <iostream>

namespace amath483 {
  double pi = 3.14159;
};

namespace amath583 {
  double pi = 3.14;
};

double pi = 3.14159265358979323846264338327950288419716939937510;

int main() {

  std::cout << "The value of pi is ";
  std::cout << pi << std::endl;

  return 0;
}
```

```
$ ./a.out
The value of pi is 3.14159
```

This one. (Why?)

Which pi?

Disambiguating

```cpp
#include <iostream>

namespace amath483 {
  double pi = 3.14159;
};

namespace amath583 {
  double pi = 3.14;
};

double pi = 3.14159265358979323846264338327950288419 7;

using namespace amath483;
using namespace amath583;

int main() {

  std::cout << "The value of pi is ";
  std::cout << pi << std::endl;

  return 0;
}
```

```
$ ./a.out

pi4.cpp:19:16: error: reference to 'pi' is ambiguous
  std::cout << pi << std::endl;
               ^

pi4.cpp:11:8: note: candidate found by name lookup is 'pi'
double pi = 3.14159265358979323846264338327950288419 7;
       ^

pi4.cpp:8:10: note: candidate found by name lookup is 'amath583::pi'
  double pi = 3.14;
         ^

pi4.cpp:4:10: note: candidate found by name lookup is 'amath483::pi'
  double pi = 3.14159;
         ^

1 error generated.
```

We know how to disambiguate and pick these

What about the global pi?

Which pi?

The global namespace

```
$ ./a.out
The value of pi is 3.14159
```

```
#include <iostream>

namespace amath483 {
  double pi = 3.14159;
};

namespace amath583 {
  double pi = 3.14;
};

double pi = 3.1415926535

using namespace amath483
using namespace amath583

int main() {

  std::cout << "The value of pi is
  std::cout << ::pi << std::endl;

  return 0;
}
```

Explicitly state which pi

Global namespace

The global namespace

```
#include <iostream>
#include <iomanip>

namespace amath483 {
  double pi = 3.14159;
};

namespace amath583 {
  double pi = 3.14;
};

double pi = 3.14159265358979323846264338327950288419 7;

using namespace amath483;
using namespace amath583;

int main() {

  std::cout << "The value of pi is ";
  std::cout << std::setprecision(15) << ::pi << std::endl;

  return 0;
}
```

```
$ ./a.out
The value of pi is 3.14159265358979
```

Include iomanip

Set precision to 15 digits

So what about namespace std?

```
#include <iostream>

namespace amath483 {
  double pi = 3.14159;
};

namespace amath583 {
  double pi = 3.14;
};

void foo() {
  using namespace amath583;
  std::cout << "The legislated value of pi is ";
  std::cout << pi << std::endl;
}

void bar() {
  using namespace amath483;
  std::cout << "The legislated value of pi is ";
  std::cout << pi << std::endl;
}

int main() {
  foo() ; bar() ;
  return 0;
}
```

Local scope

Local scope

- C++ core guidelines: Use using namespace directives for transition, for foundation libraries (such as std), or within a local scope
- Resist using namespace directives globally
- Don't write using namespace in a header file

First option for AMATH 483/583

```
#include <iostream>
#include <string>

int main() {

  std::string message = "Hello World";
  std::cout << message << std::endl;

  return 0;
}
```

Explicitly use variables from namespace std

Second option for AMATH 483/583

```
#include <iostream>

using std::cout;  using std::endl;
using std::string;

int main() {

  string message = "Hello World";
  cout << message << endl;

  return 0;
}
```

Using for just the variables you want

Third options for AMATH 483/583

```
#include <iostream>

using namespace std;

int main() {

  string message = "Hello World";
  cout << message << endl;

  return 0;
}
```

Open all of namespace std

Which option for AMATH 483/583

P.3: Express intent

```
#include <iostream>
#include <string>

int main() {

  std::string message = "Hello World";
  std::cout << message << std::endl;

  return 0;
}
```

Organizing your programs

- Software development is difficult
- How do humans attack complex problems?
- Apply the same principles to software

- Modular / reusable
- Well defined interfaces and functionality
- Understandable

Newton's Method for Square Root

- To solve $f(x) = 0$ for x
- Linearize (approximate the nonlinear problem with a linear one) and solve the linear problem
- Iterate
- Taylor: $f(x + \Delta x) \approx f(x) + \Delta x f'(x) = \Delta x f'(x)$

$$\Delta x = -\frac{f(x)}{f'(x)}$$

$$f(x) = x^2 - y = 0 \rightarrow y = \sqrt{x} \qquad f'(x) = 2x \qquad \Delta x = -\frac{x^2 - y}{2x}$$

Compute square root of 2

```
#include <iostream>
#include <cmath>

int main () {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-2.0) / (2.0*x) ;
    x += dx;
    if (std::abs(dx) < 1.e-9) break;
  }

  std::cout << x << std::endl;

  return 0;
}
```

Compute square root of 3

```cpp
#include <iostream>
#include <cmath>

int main () {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-3.0) / (2.0*x) ;
    x += dx;
    if (std::abs(dx) < 1.e-9) break;
  }

  std::cout << x << std::endl;

  return 0;
}
```

Compute square root of 2 and 3

```cpp
#include <iostream>
#include <cmath>

int main () {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-2.0) / (2.0*x) ;
    x += dx;
    if (std::abs(dx) < 1.e-9) break;
  }

  std::cout << x << std::endl;

  return 0;
}
```

Don't do the same thing twice in different places

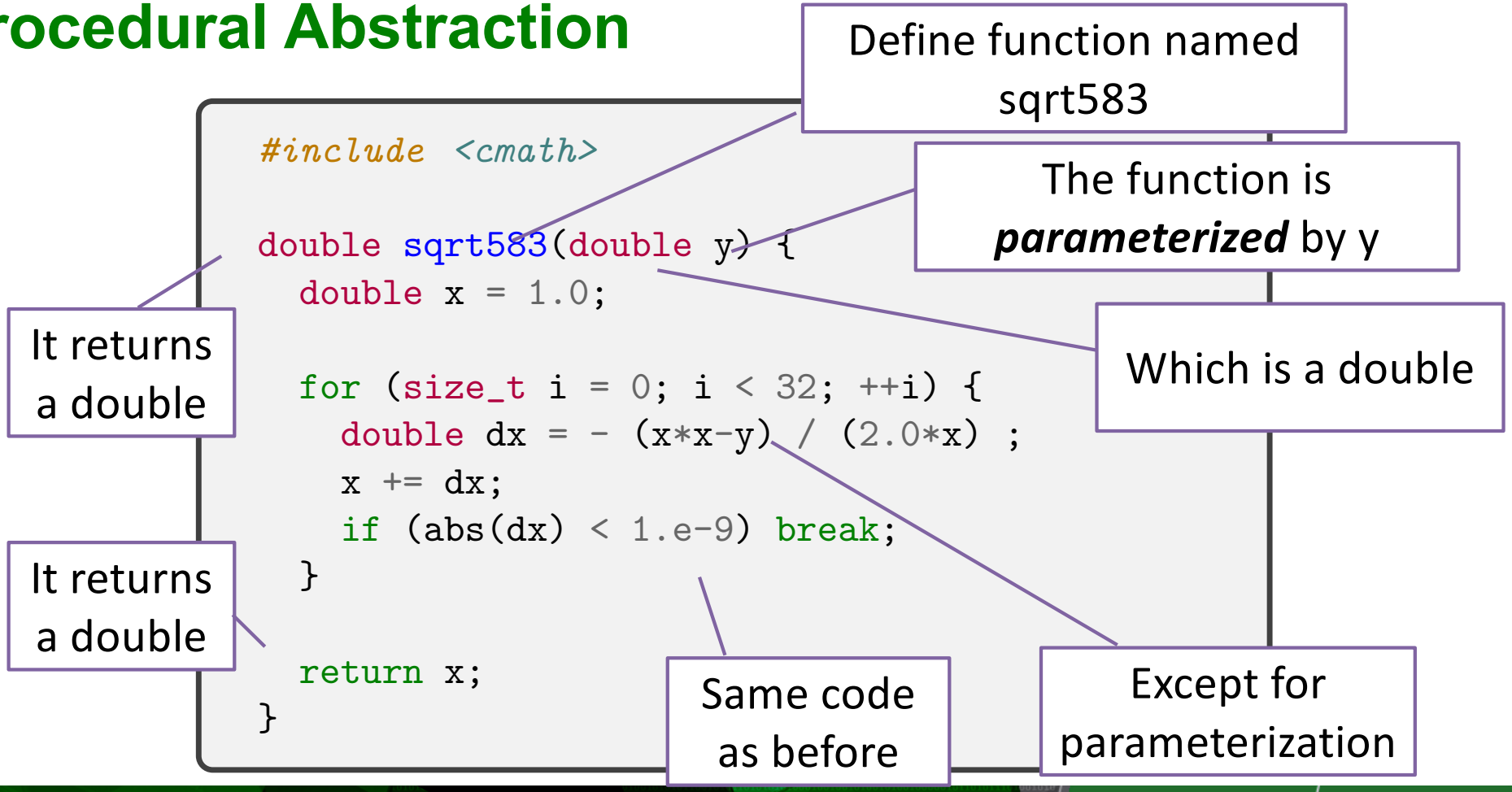
```cpp
#include <iostream>
#include <cmath>

int main () {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-3.0) / (2.0*x) ;
    x += dx;
    if (std::abs(dx) < 1.e-9) break;
  }

  std::cout << x << std::endl;

  return 0;
}
```

But they're not exactly the same

This is the only difference

This is the only difference

Procedural Abstraction



Procedural Abstraction

Redundant?

```
#include <cmath>

double sqrt583(double y) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-y) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  return x;
}
```

It returns a double

It returns a double

Compiler can deduce return types

Note auto is a C++14 feature!

It returns a double

It returns a double

```
#include <cmath>

auto sqrt583(double y) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-y) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  return x;
}
```

Square root of 2 and 3

Note initialization and declaration of i

What is a size_t?

Pass parameter 2

Pass parameter 3

```
#include <iostream>
#include <cmath>

double sqrt583(double y) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-y) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;

int main () {
                    sqrt583(2.0) << std::endl;
                    sqrt583(3.0) << std::endl;
```

Thought experiment

Change value of y

Print y

What will print?

```
#include <iostream>
#include <cmath>

double sqrt583(double y) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-y) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  y = x;

  return x;
}


int main () {
  double y = 2.0;
  std::cout << sqrt583(y) << std::endl;
  std::cout << y << std::endl;

  return 0;
}
```

```
$ ./a.out
1.41421
2
```

Parameter Passing in C++

y is passed **by value** (copied), so only the copy is changed, not the original

C++ has “pass by value” semantics

```
#include <iostream>
#include <cmath>

double sqrt583(double y) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-y) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  y = x;

  return x;
}


int main () {
  double y = 2.0;
  std::cout << sqrt583(y) << std::endl;
  std::cout << y << std::endl;

  return 0;
}
```

Parameter Passing in C++

y is passed **by value** (copied), so only the copy is changed, not the original

C++ has “pass by value” semantics

Just to be clear, the parameter can have any name (don’t confuse with y declared in main)

```
#include <iostream>
#include <cmath>

double sqrt583(double z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  z = x;

  return x;
}


int main () {
  double y = 2.0;
  std::cout << sqrt583(y) << std::endl;
  std::cout << y << std::endl;

  return 0;
}
```

Before

```
$ ./a.out
1.41421
2
```

```cpp
#include <iostream>
#include <cmath>

double sqrt583(double z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  z = x;

  return x;
}


int main () {
  double y = 2.0;
  std::cout << sqrt583(y) << std::endl;
  std::cout << y << std::endl;

  return 0;
}
```

After

```
$ ./a.out
1.41421
1.41421
```

```cpp
#include <iostream>
#include <cmath>

double sqrt583(double& z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  z = x;

  return x;
}


int main () {
  double y = 2.0;
  std::cout << sqrt583(y) << std::endl;
  std::cout << y << std::endl;

  return 0;
}
```

After

```
$ ./a.out
1.41421
1.41421
```

y is passed **by reference** (not copied), so the original is changed

This variable

Is this variable

```
#include <iostream>
#include <cmath>

double sqrt583(double& z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  z = x;

  return x;
}


int main () {
  double y = 2.0;
  std::cout << sqrt583(y) << std::endl;
  std::cout << y << std::endl;

  return 0;
}
```

Thought experiment

This variable

Is this variable

Which isn't a variable

```
sqrtr2.cpp:21:16: error: no matching function for call to 'sqrt583'
  std::cout << sqrt583(2.0) << std::endl;
               ^~~~~~~
sqrtr2.cpp:4:8: note: candidate function not viable: expects an l-value for 1st argument
double sqrt583(double &z) {
       ^
1 error generated.
```

```
#include <iostream>
#include <cmath>

double sqrt583(double &z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  z = x;

  return x;
}


int main () {

  std::cout << sqrt583(2.0) << std::endl;

  return 0;
```

Thought experiment

Why would we want to pass a reference?

"Out parameters"

Efficiency (no copy)

How can we do this?

```
#include <iostream>
#include <cmath>

double sqrt583(double &z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  z = x;

  return x;
}


int main () {

  std::cout << sqrt583(2.0) << std::endl;


  return 0;
}
```

Before

```cpp
#include <iostream>
#include <cmath>

double sqrt583(double &z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  z = x;

  return x;
}


int main () {

  std::cout << sqrt583(2.0) << std::endl;


  return 0;
}
```

After

```
#include <iostream>
#include <cmath>

double sqrt583(const double &z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  z = x;

  return x;
}


int main () {

  std::cout << sqrt583(2.0) << std::endl;


  return 0;
}
```

After

Promise not to change z

A reference to a constant is okay

```
#include <iostream>
#include <cmath>

double sqrt583(const double &z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  z = x;

  return x;
}


int main () {

  std::cout << sqrt583(2.0) << std::endl;


  return 0;
}
```

Functions

- F.2: A function should perform a single logical operation
- F.3: Keep functions short and simple
- F.16: For “in” parameters, pass cheaply-copied types by value and others by reference to const
- F.17: For “in-out” parameters, pass by reference to non-const
- F.20: For “out” output values, prefer return values to output parameters

Thought experiment

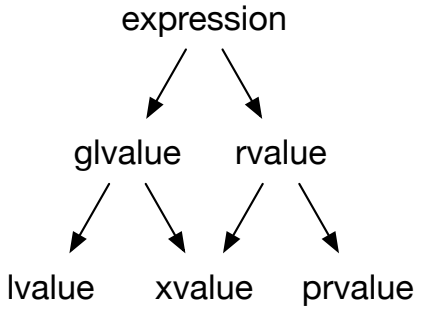
What's an l-value?

```
sqrtr2.cpp:21:16: error: no matching function for call to 'sqrt583'
  std::cout << sqrt583(2.0) << std::endl;
               ^~~~~~~
sqrtr2.cpp:4:8: note: candidate function not viable: expects an l-value for 1st argument
double sqrt583(double &z) {
       ^
1 error generated.
```

```
#include <iostream>
#include <cmath>

double sqrt583(double &z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  z = x;

  return x;
}


int main () {

    l::cout << sqrt583(2.0) << std::endl;

    urn 0;
```

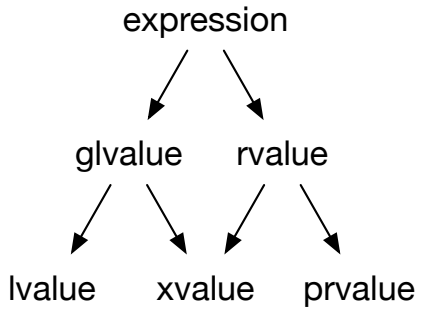
l-values and r-values

- Section 3.10 of C++ standard
  - A *glvalue* is an expression whose evaluation determines the identity of an object, bit-field, or function.
  - A *prvalue* is an expression whose evaluation initializes an object or a bit-field, or computes the value of the operand of an operator, as specified by the context in which it appears.
  - An *xvalue* is a glvalue that denotes an object or bit-field whose resources can be reused (usually because it is near the end of its lifetime).
  - An *lvalue* is a glvalue that is not an xvalue.
  - An *rvalue* is a prvalue or an xvalue



l-values and r-values

- More intuitively
- Ignore glvalue, xvalue, prvalue
- lvalue is something that can go on the ***left*** of an assignment (correctly)
  - "Lives" beyond an expression
- Rvalue is something that can go on the ***right*** of an assignment (correctly)
  - Does not "live" beyond an expression

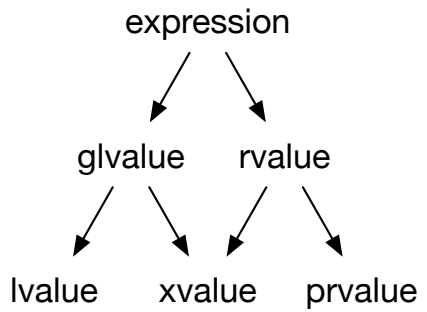


l-values and r-values

```
double x, y, z;

x = y;
x = 1.0;
y = x + z;
```

lvalue rvalue

```
double x, y, z;

x = y;
1.0 = x;
x + z = y;
```




```
% c++ sl7.cpp
c++ sl7.cpp
sl7.cpp:7:9: error: expression is not assignable
  x + z = y;
  ~~~~~ ^
1 error generated.
```

l-values and r-values

- Consider following program:

```
int main() {
  double y = 2.0, z = 3.0;

  double x = sqrt583(y+z);

  return 0;
}
```

rvalue

Copy: x = y + z (lvalue = rvalue

- With following declarations:

```
double sqrt583(double x);
```

OK to copy rvalue

```
double sqrt583(double& x);
```

Not OK to reference rvalue

```
double sqrt583(const double& x);
```

OK to reference const rvalue

l-values and r-values

- How is the value used in the function

```
double sqrt583(double y) {
  double x = 0.0, dx;

  do {
    dx = - (x*x-y) / (2.0*x);
    x += dx;
  } while (abs(dx) > 1.e-9);

  y = x;

  return x;
}
```

lvalue

lvalue

- With following declarations:

```
double sqrt583(double x);
```

OK to copy rvalue

```
double sqrt583(double& x);
```

Not OK to reference rvalue

```
double sqrt583(const double& x);
```

OK to reference const rvalue

l-values and r-values

- How is the value used in the function

```
double sqrt583(double& y) {
  double x = 0.0, dx;

  do {
    dx = -(x*x-y) / (2.0*x);
    x += dx;
  } while (abs(dx) > 1.e-9);

  y = x;

  return x;
}
```

lvalue ref

lvalue ref

- With following declarations:

```
double sqrt583(double x);
```

Temp result of x+y is references

```
double sqrt583(double& x);
```

Not OK to reference rvalue

```
double sqrt583(const double& x);
```

l-values and r-values

- How is the value used in the function

```
double sqrt583(const double& y) {
  double x = 0.0, dx;

  do {
    dx = -(x*x-y) / (2.0*x);
    x += dx;
  } while (abs(dx) > 1.e-9);

  return x;
}
```

Const lvalue ref

- With following declarations:

```
double sqrt583(double x);
```

```
double sqrt583(double& x);
```

Temp x+y can be referenced if read only

```
double sqrt583(const double& x);
```

OK to reference const rvalue

l-values and r-values

- How is the value used in the function

```
double sqrt583(const double& y) {
  double x = 0.0, dx;

  do {
    dx = -(x*x-y) / (2.0*x);
    x += dx;
  } while (abs(dx) > 1.e-9);

  y = x;

  return x;
}
```

Const lvalue ref

```
$ c++ -c sl11.cpp
error: cannot assign to variable 'y' with
const-qualified type 'const double &'
  y = x;
  ~ ^
sl11.cpp:3:30: note: variable 'y' declared const here
double sqrt583(const double& y) {
               ~~~~~~~~~~~~~~~^
1 error generated.
```

Reusing functions

```
#include <iostream>
#include <cmath>

double sqrt583(double z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  return x;
}


int main () {

  std::cout << sqrt583(2.0) << std::endl;


  return 0;
}
```

```
$ c++ main.cpp
$ ./a.out
1.4142
```

Compile main.cpp

```
$ c++ main.cpp
```

Translate it into a language the cpu can run

```
$ ./a.out
```

The executable (program that the cpu can run)

Reusing in other programs

Put this function in its own file amath583.cpp

Many programs (mains) can call it

```cpp
#include <cmath>

double sqrt583(double& z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  return x;
}
```

```cpp
#include <iostream>
#using namespace std;
int main () {

  cout << sqrt583(3.0) << e

  return 0;
}
```

```cpp
#include <iostream>
#using namespace std;
int main () {

  cout << sqrt583(3.14) <

  return 0;
}
```

```cpp
#include <iostream>
#using namespace std;
int main () {

  cout << sqrt583(42.0) << endl;

  return 0;
}
```

Reusing in other programs

Many mains can call it

```
#include <iostream>
#using namespace std;
int main () {

  cout << sqrt583(42.0) << endl;

  return 0;
}
```

```
sqrt3.cpp:8:11: error: use of undeclared identifier 'sqrt583'
  cout << sqrt583(2.0) << endl;
          ^
sqrt3.cpp:9:11: error: use of undeclared identifier 'sqrt583'
  cout << sqrt583(3.0) << endl;
          ^
2 errors generated.
```

Undeclared identifier

Defined in a different file

```
#include <cmath>

double sqrt583(double z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  return x;
}
```

Didn't we declare it here?

This is ***definition***

Reusing functions

Doesn't know how to translate this

```
$ c++ main.cpp
$ ./a.out
1.4142
```

Compile main.cpp

```
#include <iostream>
#using namespace std;
int main () {

  cout << sqrt583(42.0) << endl;

  return 0;
}
```

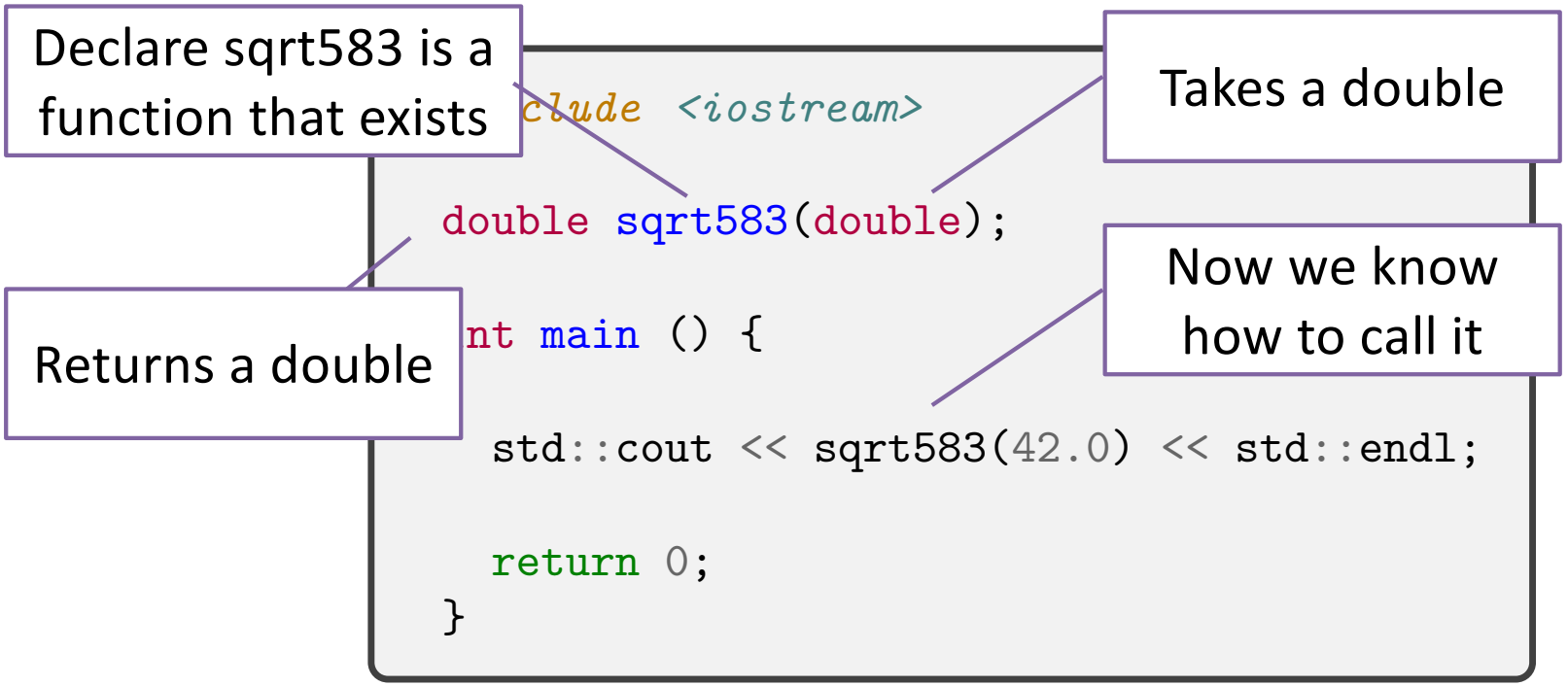
```
$ c++ main.cpp
```

Translate it into a language the cpu can run

```
$ ./a.out
```

The executable (program that the cpu can run)

Reusing functions across programs



Reusing in other programs

```
#include <iostream>

double sqrt583(double);

int main () {

  std::cout << sqrt583(42.0) << std::e

  return 0;
}
```

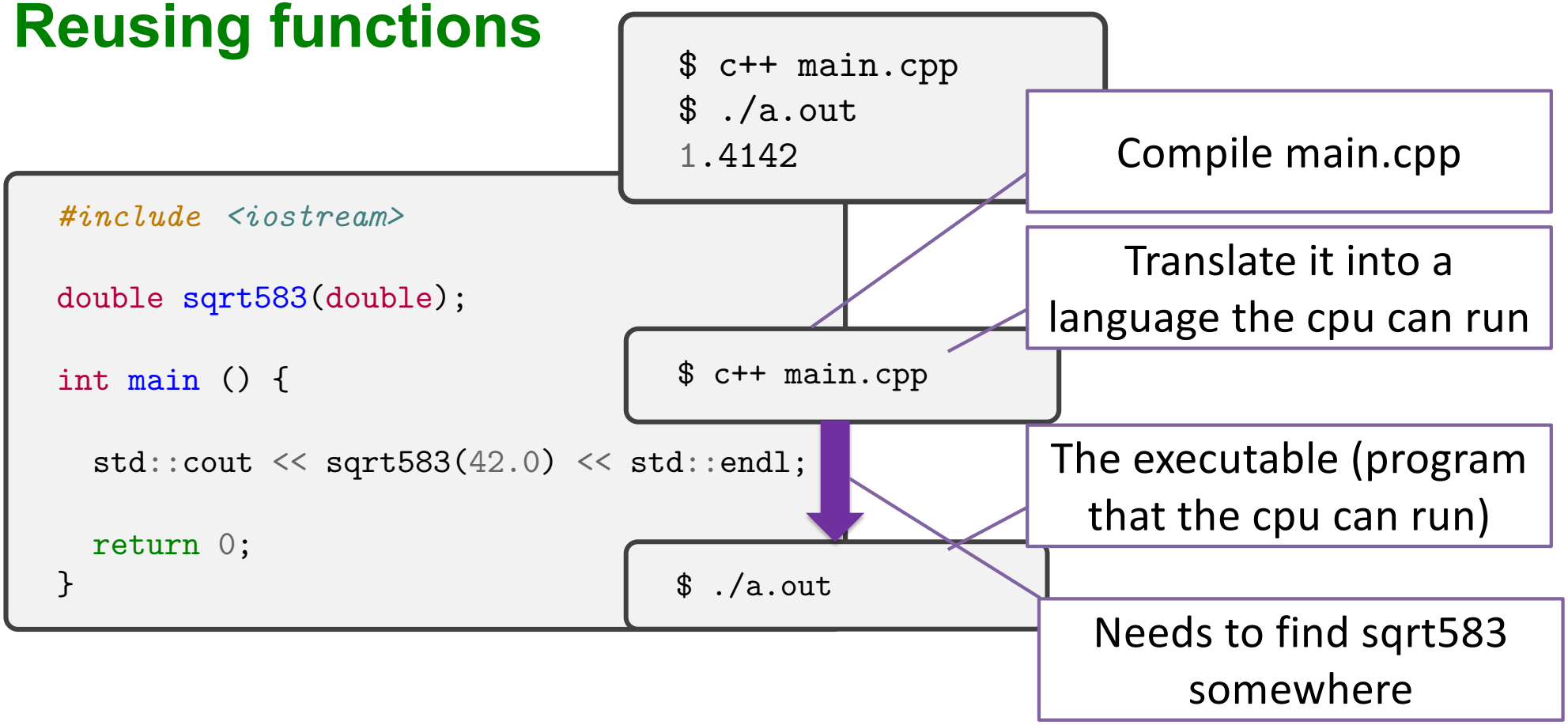
Many mains can call sqrt583

Undefined symbol

Linker command failed

```
Undefined symbols for architecture x86_64:
  "sqrt583(double const&)", referenced from:
      _main in sqrt3-1d1d35.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

Reusing functions



Reusing in other programs

```
#include <iostream>
double sqrt583(const double& x);

using namespace std;

int main () {

  cout << sqrt583(2.0) << endl;
  cout << sqrt583(3.0) << endl;

  return 0;
}
```

Declare sqrt583

```
Undefined symbols for architecture x86_64:
  "sqrt583(double const&)", referenced from:
      _main in sqrt3-1d1d35.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

Reusing in other programs

```cpp
#include <iostream>
#using namespace std;
int main () {

  cout << sqrt583(42.0) << endl;

  return 0;
}
```

```cpp
#include <cmath>

double sqrt583(double z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  return x;
}
```

```
$ c++ main.cpp sqrt583.cpp
```

Compile main.cpp ***with*** sqrt583.cpp

Translate it into a language the cpu can run

```
$ ./a.out
```

The executable (program that the cpu can run)

Reusing in other programs

```cpp
#include <iostream>
#using namespace std;
int main () {

  cout << sqrt583(42.0) << endl;

  return 0;
}
```

```cpp
#include <cmath>

double sqrt583(double z) {
  double x = 1.0;

  for (size_t i = 0; i < 32; ++i) {
    double dx = - (x*x-z) / (2.0*x) ;
    x += dx;
    if (abs(dx) < 1.e-9) break;
  }

  return x;
}
```

```
$ c++ main.cpp
```

```
$ c++ sqrt583.cpp
```

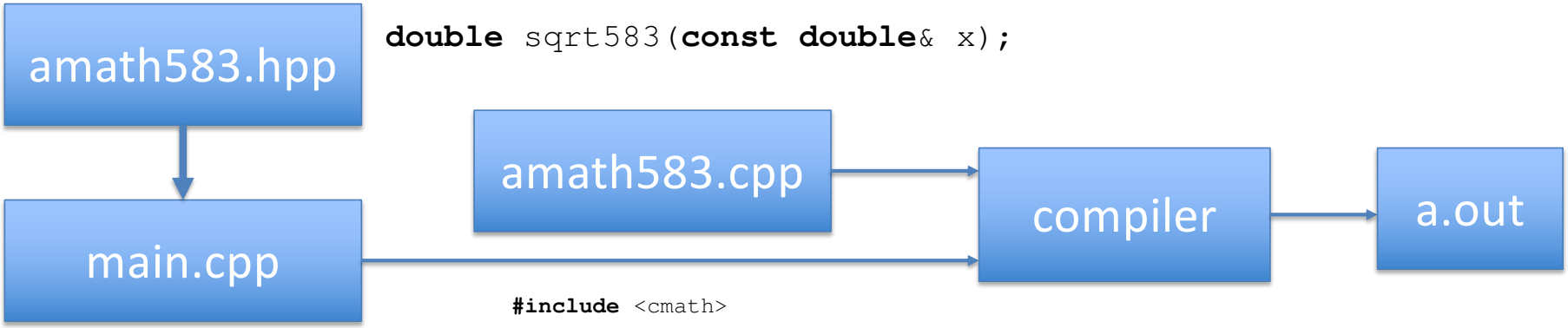
```
$ ./a.out
```

Compile main.cpp by itself

Compile sqrt583.cpp by itself

Generate executable

One more refinement in organization



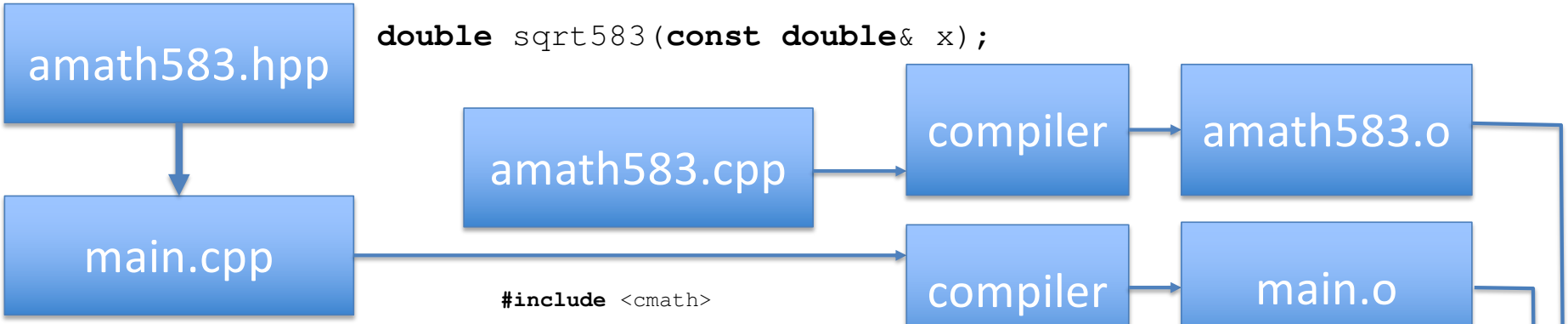

```
double sqrt583(const double& x);
```

```
#include <iostream>
#include "amath583.hpp"

using namespace std;

int main () {

  cout << sqrt583(2.0) << endl;
  cout << sqrt583(3.0) << endl;

  return 0;
}
```

```
#include <cmath>

using std::abs;

double sqrt583(const double& y) {
  double x = 0.0, dx;

  do {
    dx = - (x*x-y) / (2.0*x);
    x += dx;
  } while (abs(dx) > 1.e-9);

  return x;
}
```

C++ main.cpp amath583.cpp

One more refinement in organization

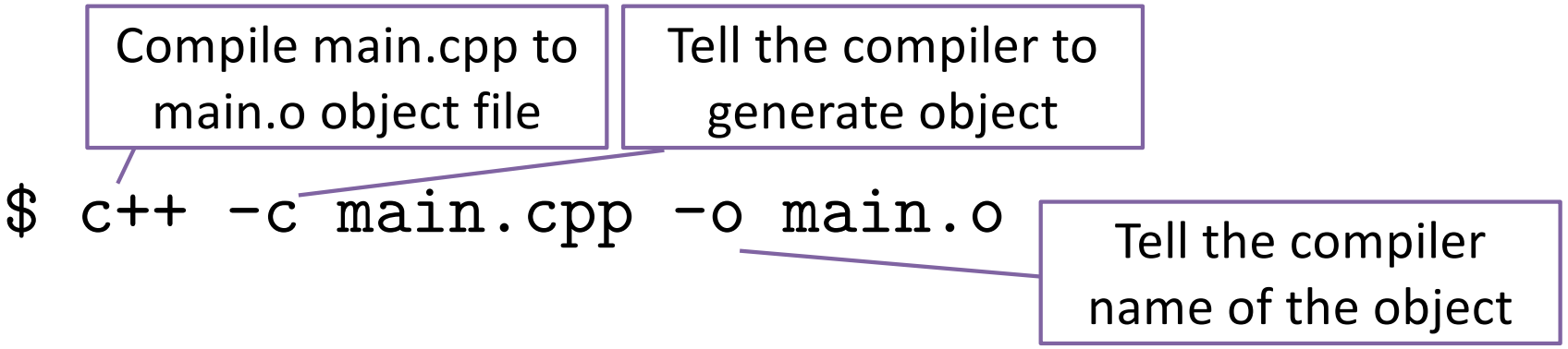



```
double sqrt583(const double& x);
```

```
#include <iostream>
#include "amath583.hpp"

using namespace std;

int main () {

  cout << sqrt583(2.0) << endl;
  cout << sqrt583(3.0) << endl;

  return 0;
}
```

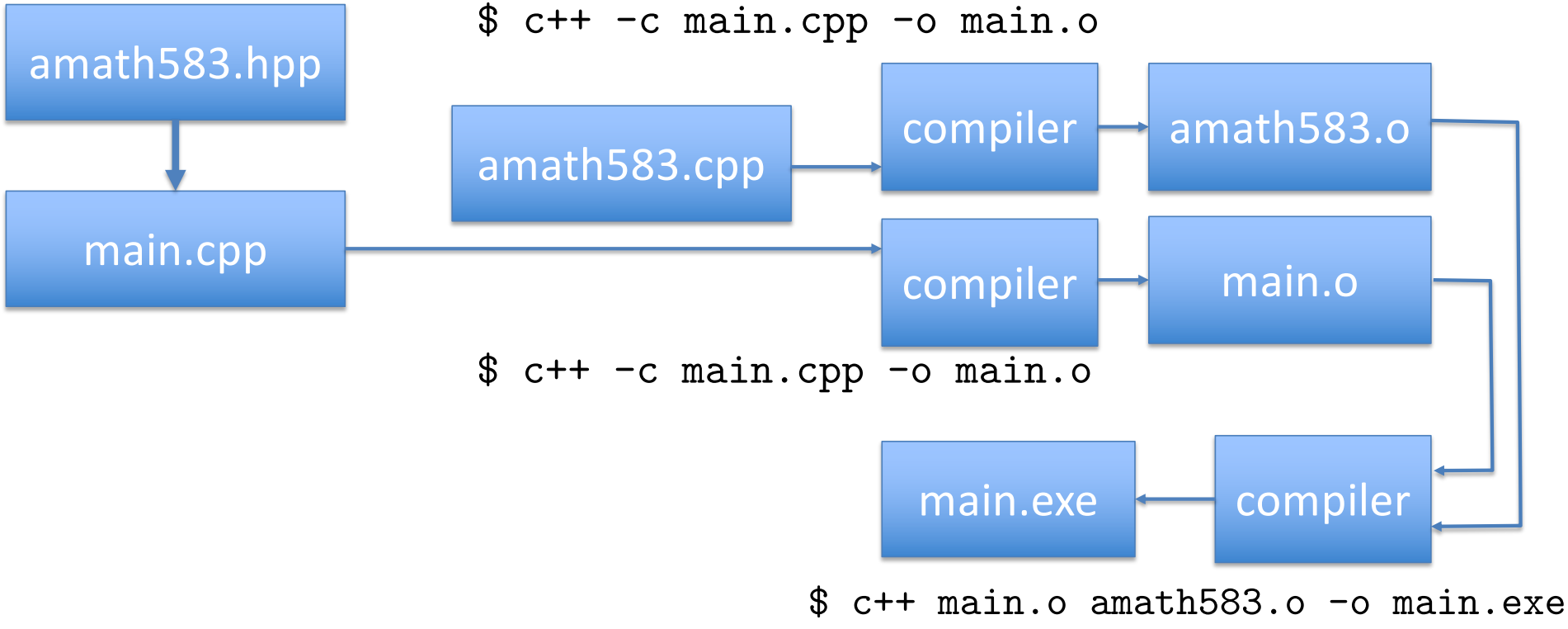
```
#include <cmath>

using std::abs;

double sqrt583(const double& y) {
  double x = 0.0, dx;

  do {
    dx = - (x*x-y) / (2.0*x);
    x += dx;
  } while (abs(dx) > 1.e-9);

  return x;
}
```

Multifile Multistage Compilation

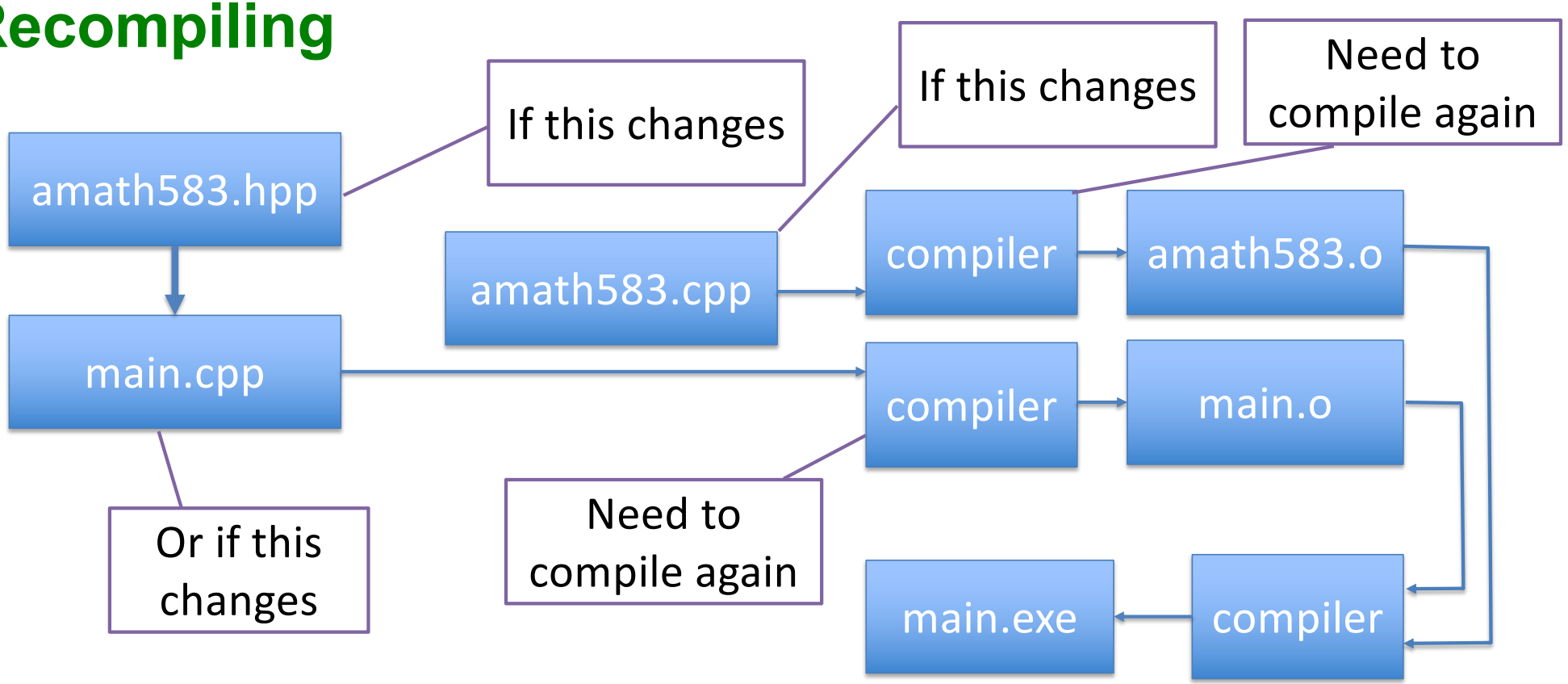



```
$ c++ -c main.cpp -o main.o

$ c++ -c amath583.cpp -o amath583.o

$ c++ main.o amath583.o -o main.exe
```

One more refinement in organization



Recompiling



Dependencies

- main.o depends on main.cpp and amath583.hpp
- amath583.o depends on amath583.cpp
- main.exe depends on amath583.o and main.o

Automating: The Rules

- If main.o is newer than main.exe → recompile main.exe
- If amath583.o is newer than main.exe → recompile main.exe
- If main.cpp is newer than main.o → recompile main.o
- If amath583.cpp is newer than amath583.o → recompile amath583.o
- If amath583.hpp is newer than main.o → recompile main.o

Make

- Tool for automating compilation (or any other rule-driven tasks)
- Rules are specified in a makefile (usually named “Makefile”)
- Rules include
  - Dependency
  - Target
  - Consequent

```
main.exe: main.o amath583.o
    c++ main.o amath583.o -o main.exe

main.o: main.cpp amath583.hpp
    c++ -c main.cpp -o main.o

amath583.o: amath583.cpp
    c++ -c amath583.cpp -o amath583.o
```

Dependencies

Consequent

Target

Make

- Tool for automating compilation (or any other rule-driven tasks)
- Rules are specified in a makefile (usually named “Makefile”)
- Rules include
  - Dependency
  - Target
  - Consequent

```
$ make
c++ -c main.cpp -o main.o
c++ -c amath583.cpp -o amath583.o
c++ main.o amath583.o -o main.exe
```

- Edit amath583.hpp

```
$ make
c++ -c main.cpp -o main.o
c++ main.o amath583.o -o main.exe
```

Thank you!

Creative Commons BY-NC-SA 4.0 License