AMATH 483/583
High Performance Scientific Computing

Lecture 2:
Variables, functions, parameters

Xu Tony Liu, PhD

Paul G. Allen School of Computer Science & Engineering

University of Washington

Seattle, WA

HPC in News

- Jack Dongarra won ACM Turning Award on 03/30
- “FOR PIONEERING CONCEPTS AND METHODS WHICH HAVE RESULTED IN WORLD-CHANGING COMPUTATIONS”
- LINPACK benchmark
  - a measure of a system's floating-point computing power
  - Top500

- LAPACK: Linear Algebra PACKage
- Autotuning: we will learn this in later lectures
- MPI: ditto

Administrivia

- PS1 posted
- 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
- L-value and r-value
- Compilation

Previous 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)

Top500 as of Nov 2021 (top500.org)

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371

7.6M cores

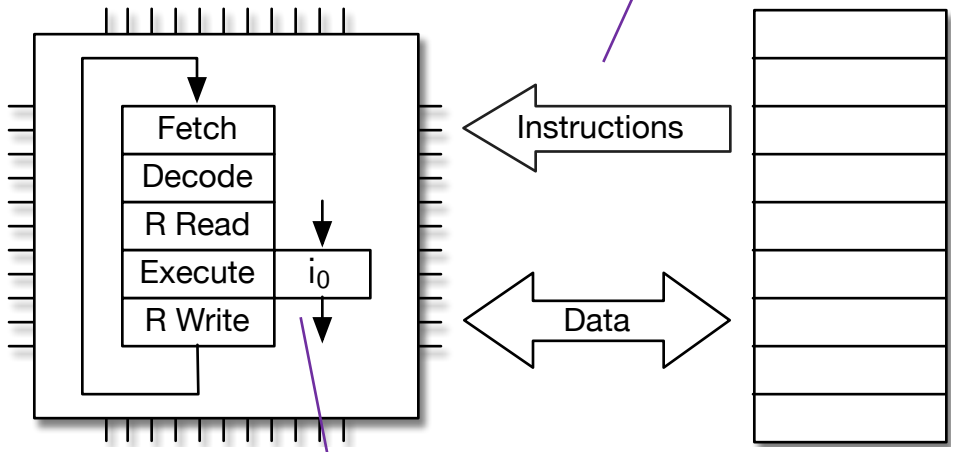
Top500 as of Nov 2021 (top500.org)

5	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70,870.0	93,750.0	2,589
6	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646
7	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
8	**JUWELS Booster Module** - Bull Sequana XH2000 , AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation ClusterSuite, Atos Forschungszentrum Juelich (FZJ) Germany	449,280	44,120.0	70,980.0	1,764

Scaling progression of CPUs

Simplest model

CPU fetches and executes instructions

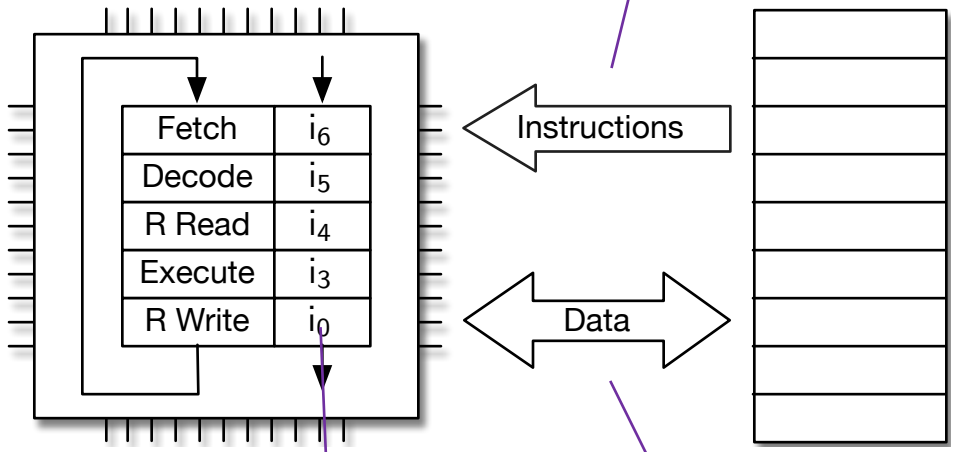


Many cycles per instruction

Pipelining

Pipelining

Instructions are fetched in a stream

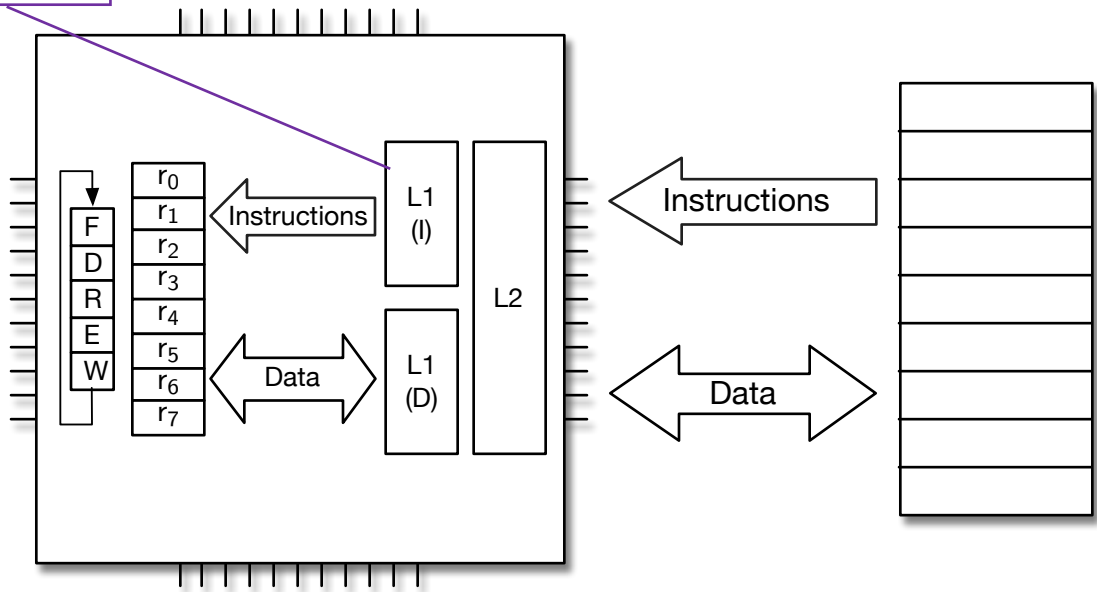


Processed in a pipeline

A long trip from memory

Hierarchical memory

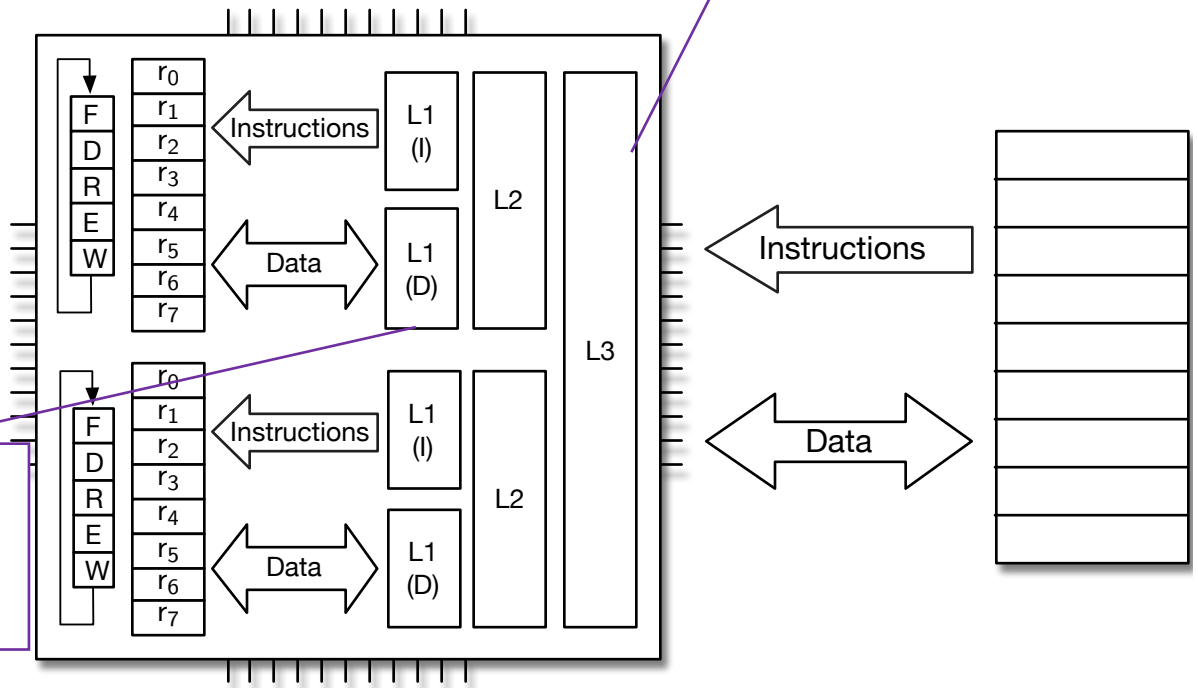
Use special, fast memory to keep data and instructions close



Multicore CPUs

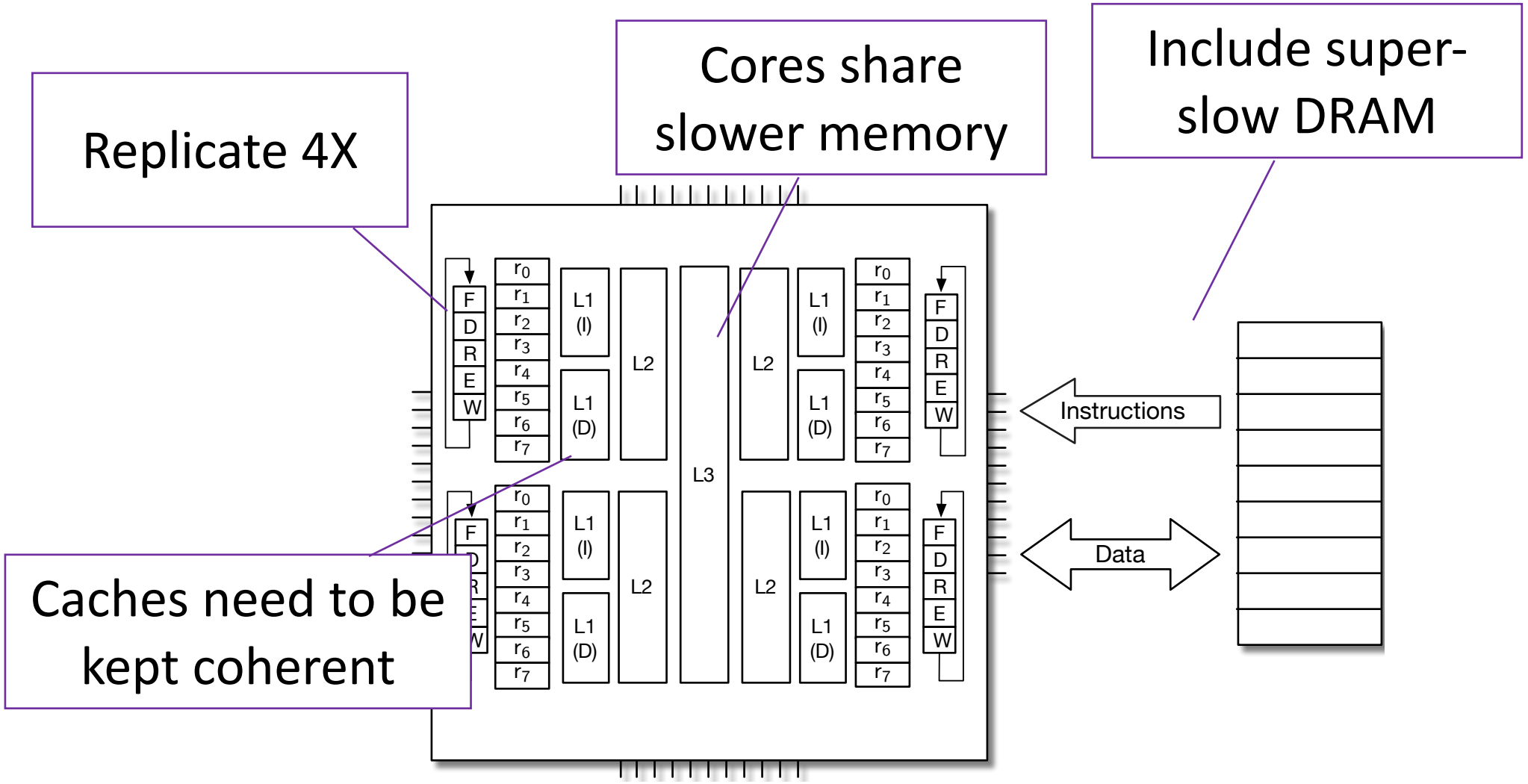
Replicate 2X

Cores share slower memory



Caches need to be kept coherent

Even more cores

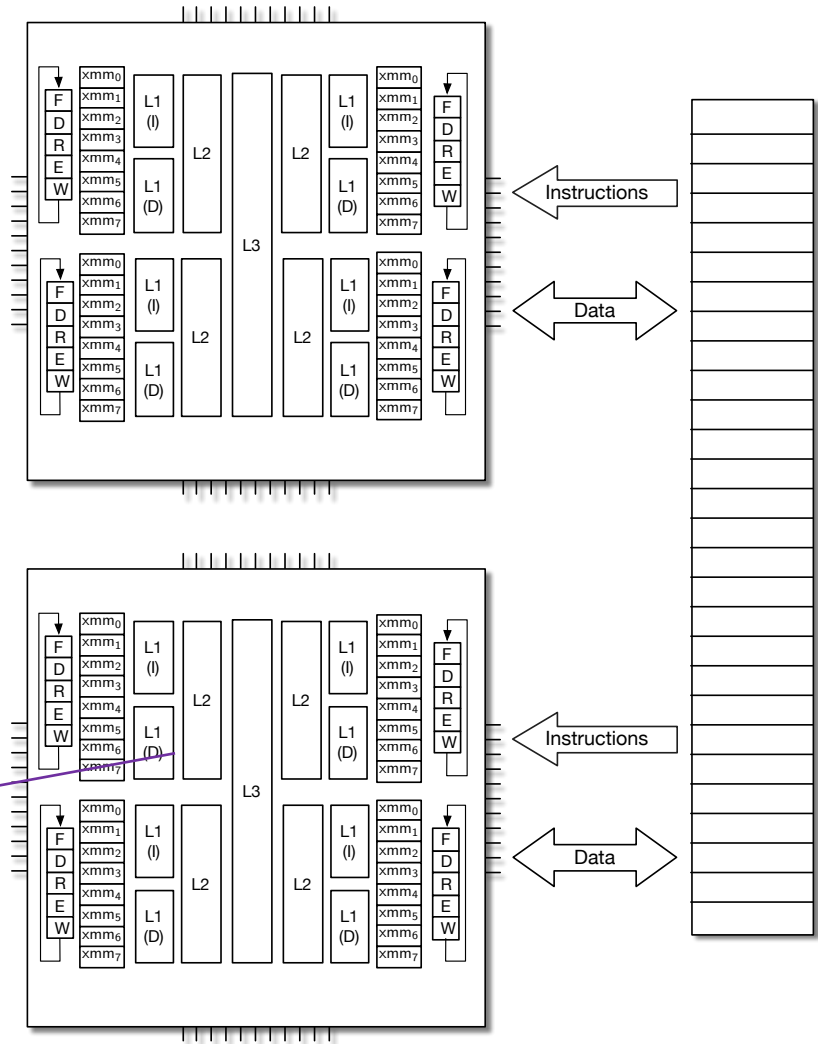


Symmetric Multi-Processor (SMP)

Multiple CPU chips

AKA “sockets”

Caches still need to be kept (somewhat) coherent



Memory may be uniformly shared among sockets

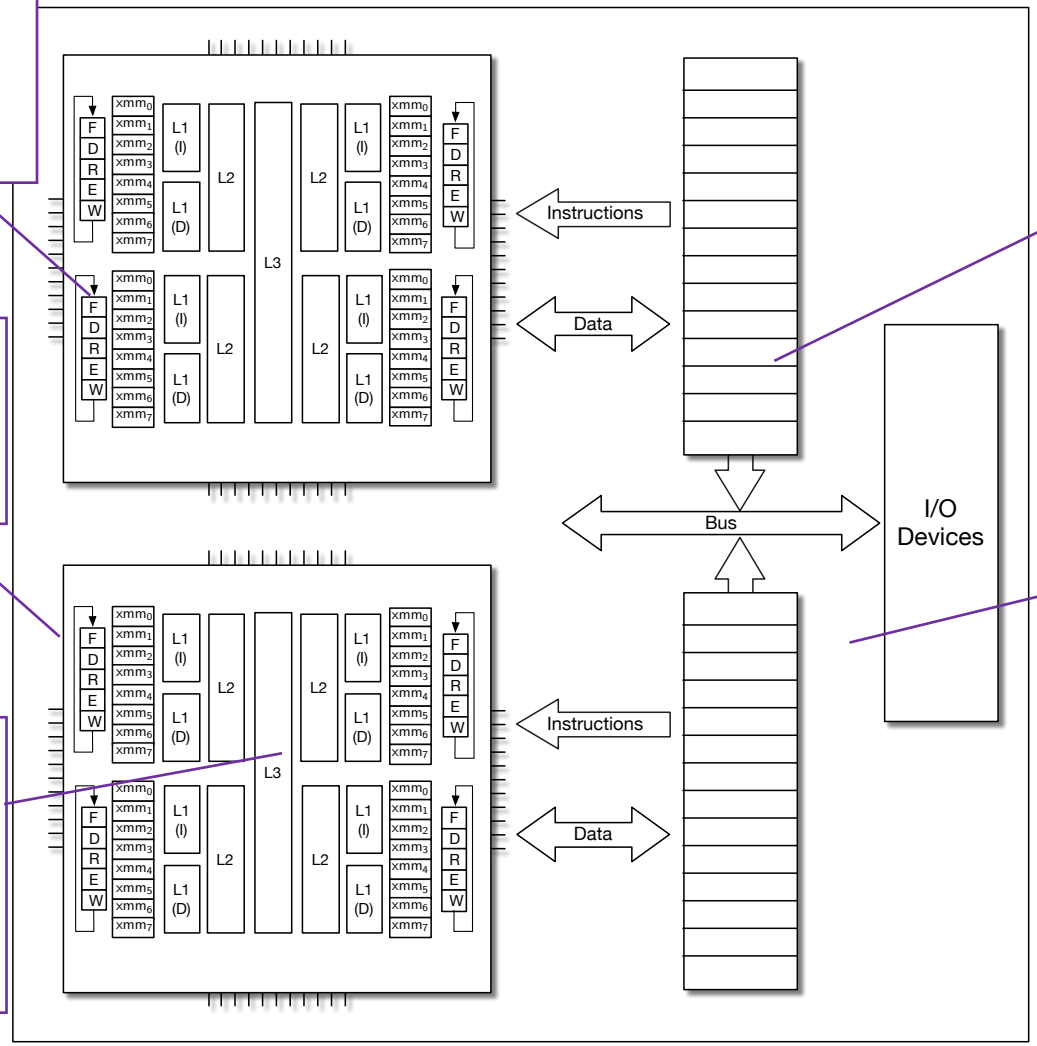
Uniform memory access (UMA)

Asymmetric Multi-Processor

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

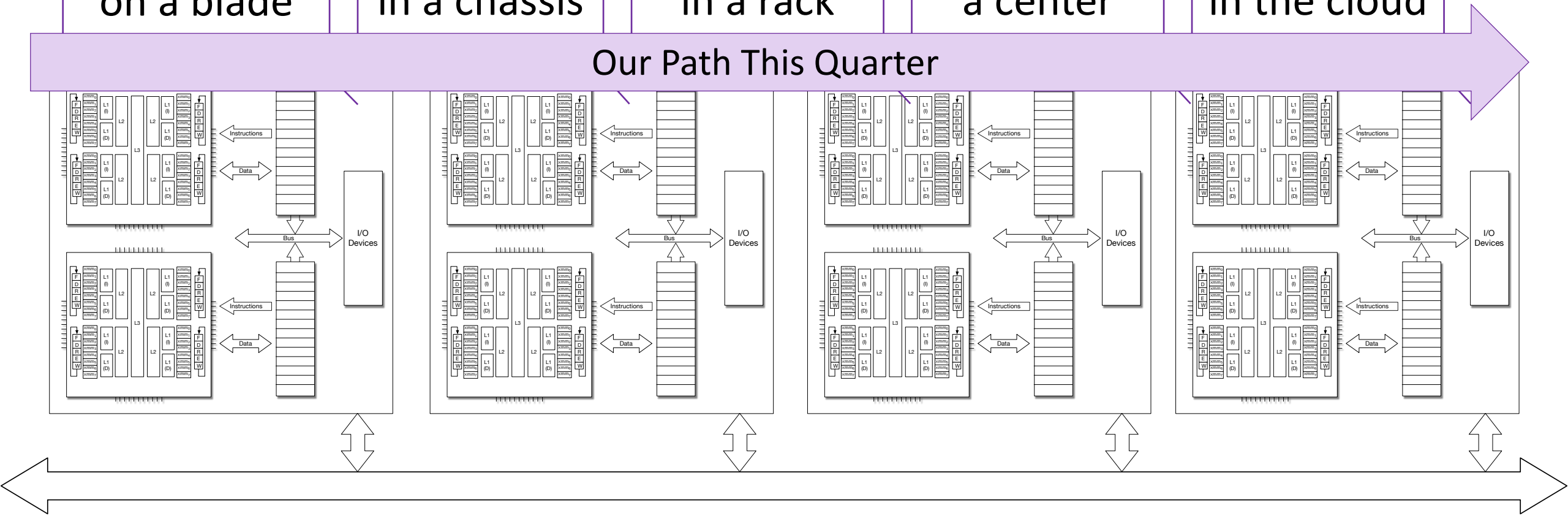
Put blades in a chassis

Put chassis in a rack

Put racks in a center

Put centers in the cloud

Our Path This Quarter



Computational Science

System of Partial Differential Eqns

$$\begin{aligned} \nabla \cdot \boldsymbol{P} &= \boldsymbol{f}_0 \quad \text{in} \quad \Omega_0 \\ [\![\boldsymbol{P} \cdot \boldsymbol{N}_0]\!] &= [\![\boldsymbol{t}_c]\!] \quad \text{on} \quad S_0 \\ \boldsymbol{P} \cdot \boldsymbol{N}_0 &= \boldsymbol{t}_0 \quad \text{on} \quad \partial\Omega_{t_0} \\ \boldsymbol{u} &= \boldsymbol{u}_p \quad \text{on} \quad \partial\Omega_{u_0} \end{aligned}$$

Find P that satisfies this

(too hard)

discretize

System of Nonlinear Eqns

$F(x) = 0$

Find x that satisfies this

(too hard)

linearize

System of Linear Eqns

$Ax = b$

Find x that satisfies this

All of scientific computing is this

A problem we can solve

Computational Science

- The fundamental computation at the core of many (mo
  computational science programs is solving $Ax = b$

- Assume $x, b \in R^N$ and $A \in R^{N \times N}$
- i.e., x and b are vectors with N real elements and A is a matrix with N by N real elements

- Solution process only requires basic arithmetic operatio

Requirements for machine learning are changing this

We will see this a lot

This is what computers can do

Write A Program

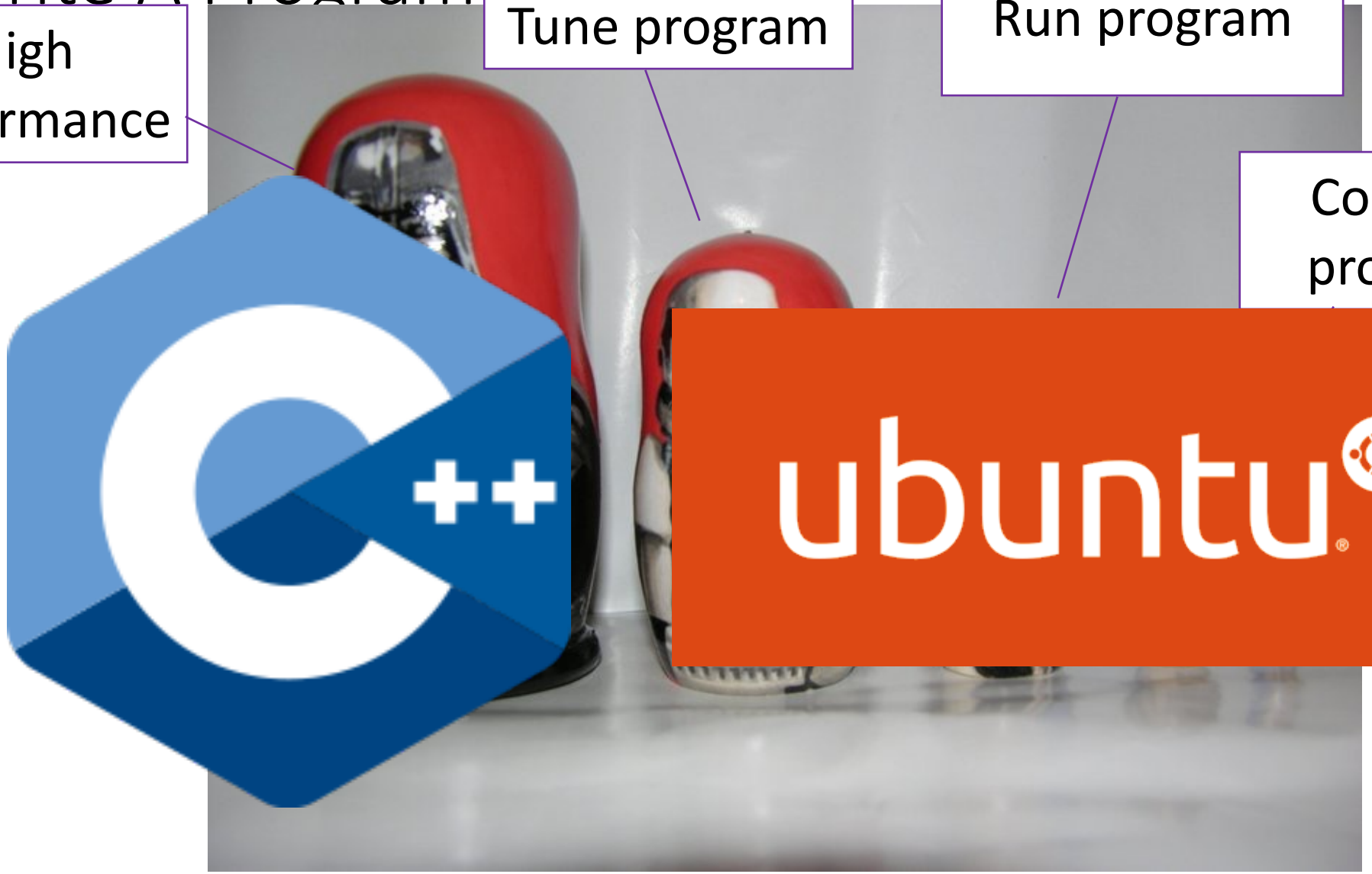
High performance

Measure and Tune program

Run program

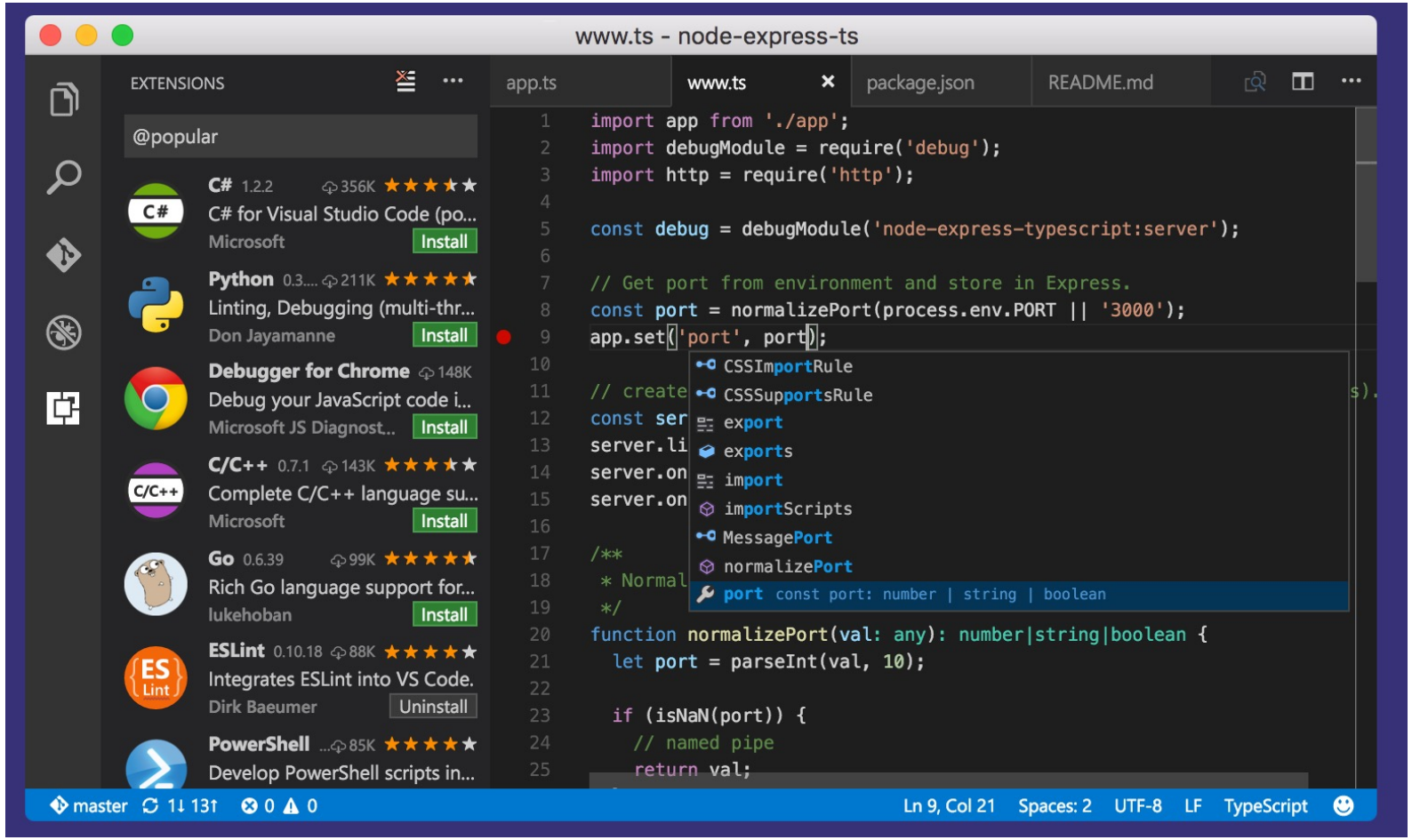
Compile program

Write program



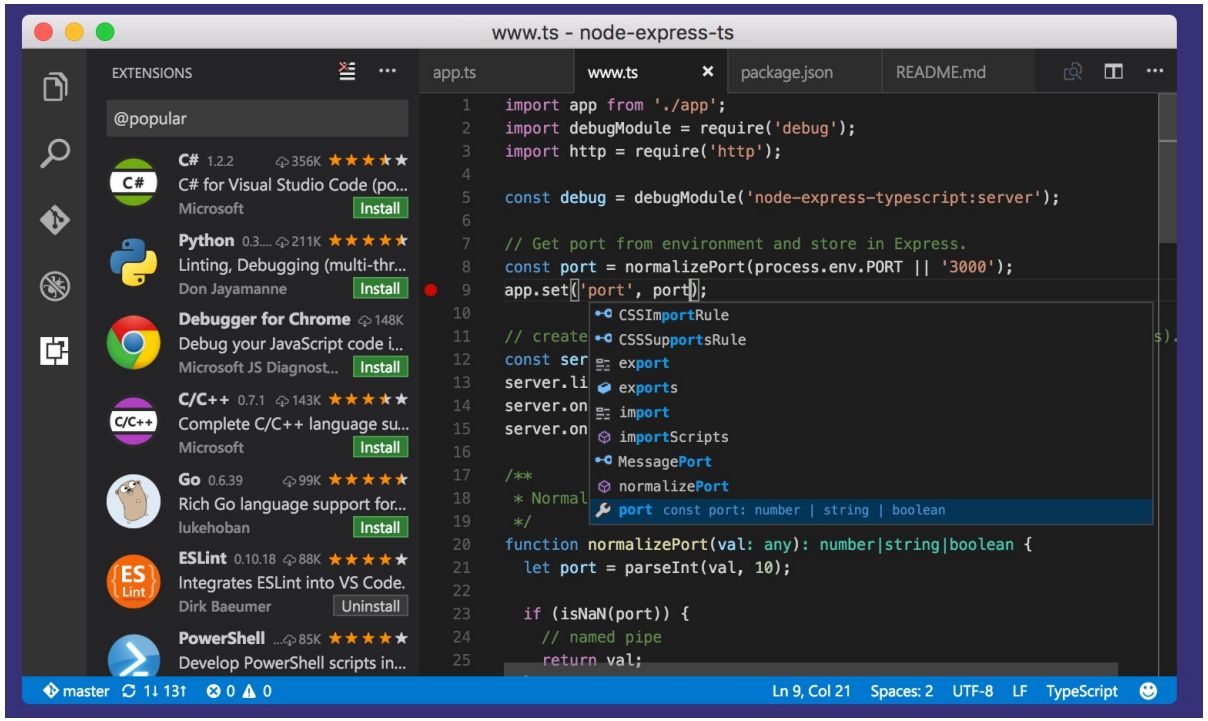
Developing your code

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



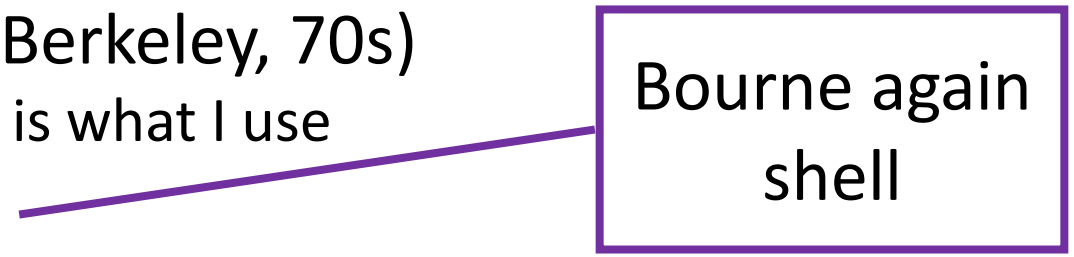
The AMATH 483/583 Development Environment

- Windows 10 (Pro and Education) and Mac OS X can be made to approximate Linux
  - Uniform docker environment for everyone
- How-to documentation in problem set and online



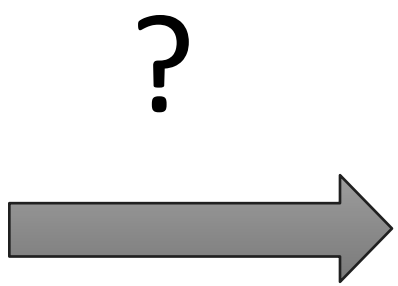
shells

- 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

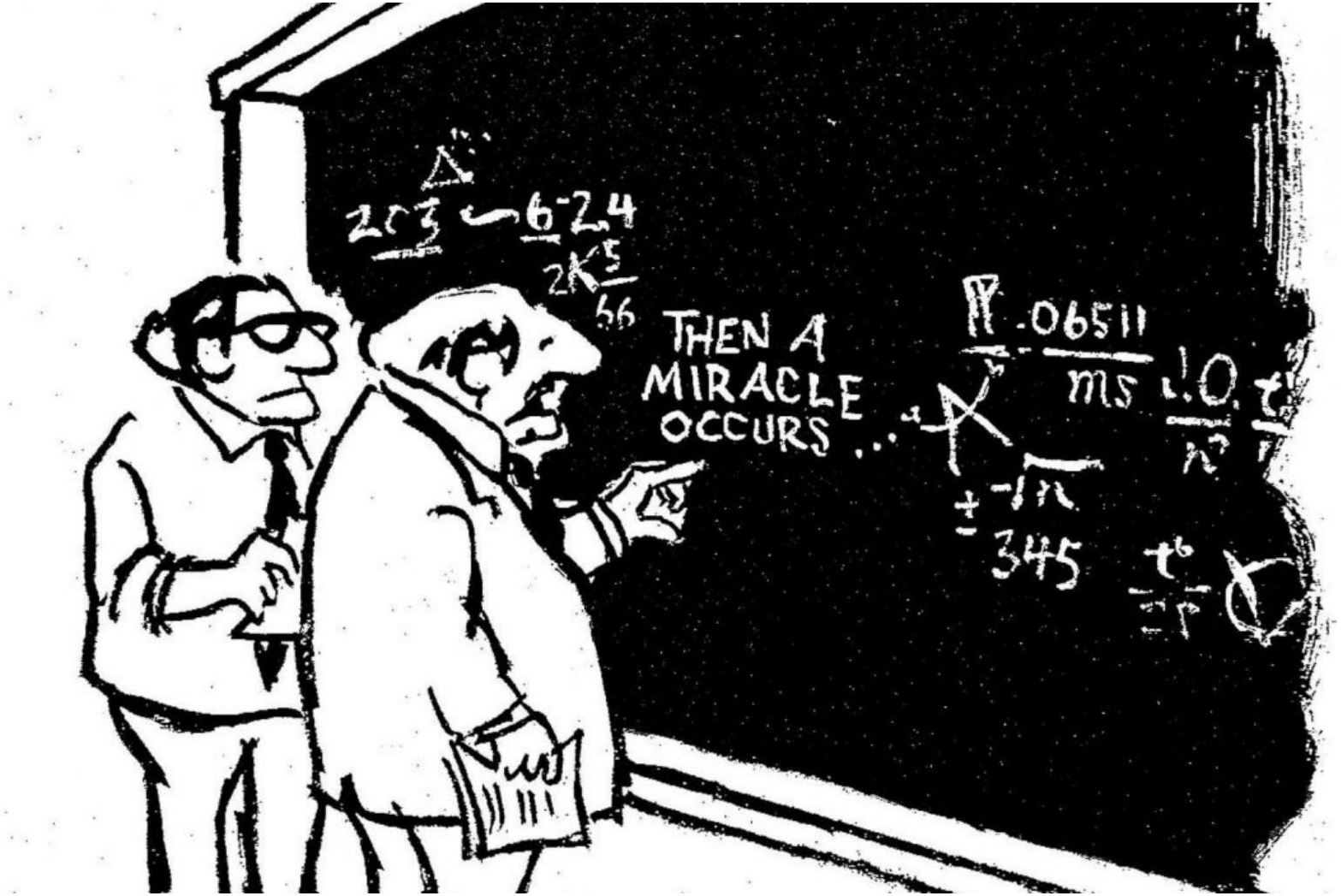


Programs and Programming

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



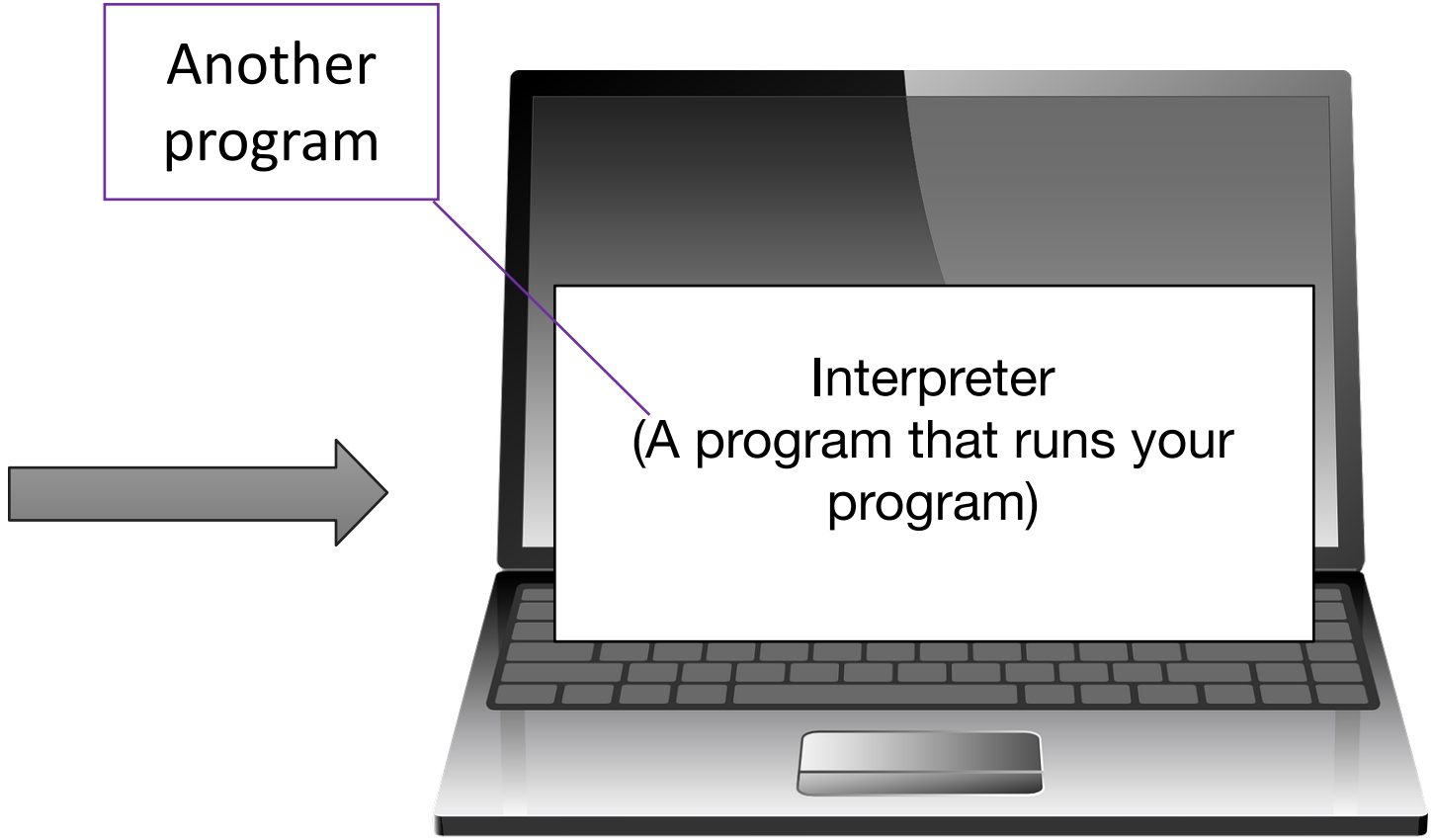
Programs and Programming



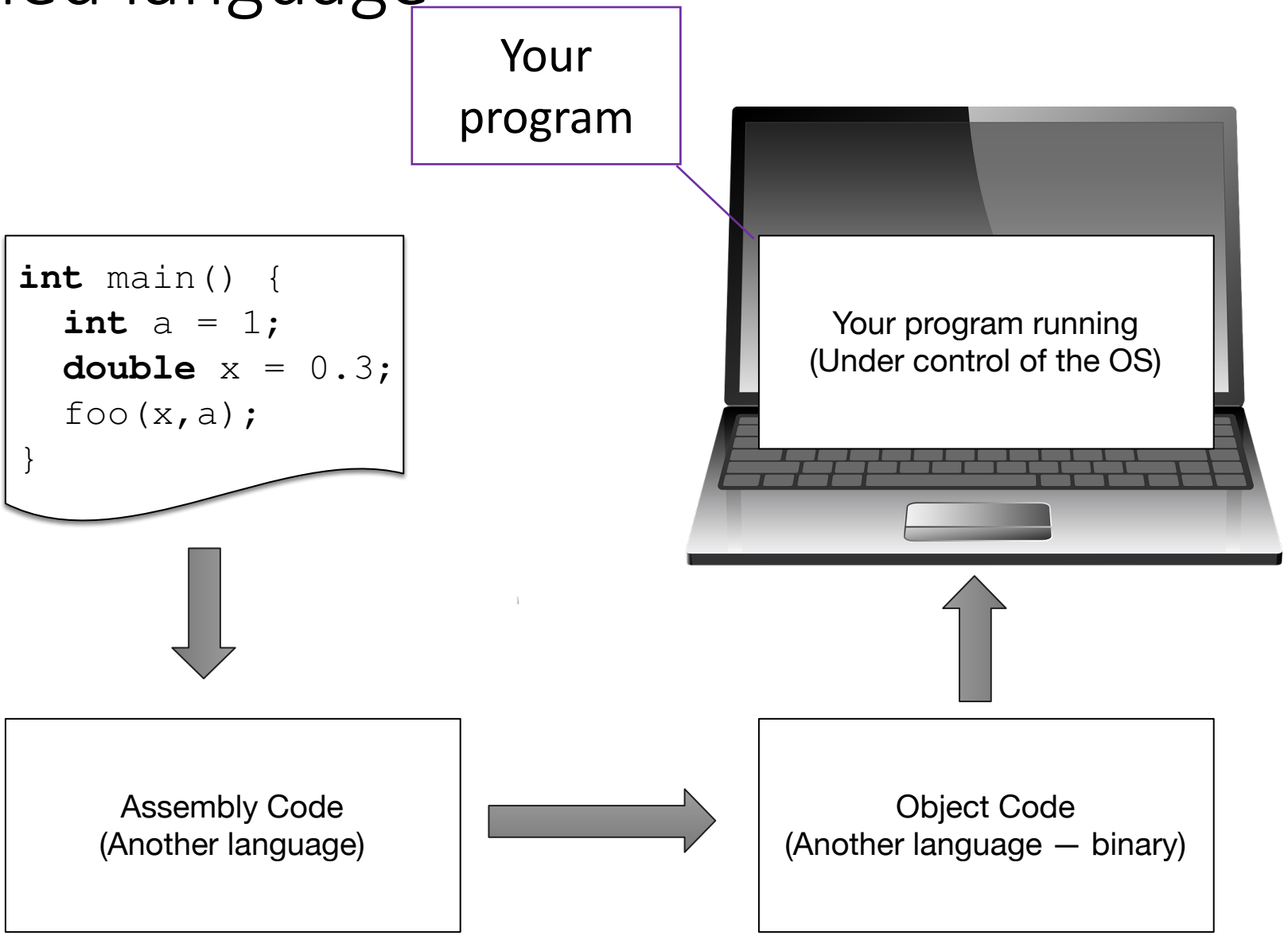
© S. Harris

Interpreted language (Python)

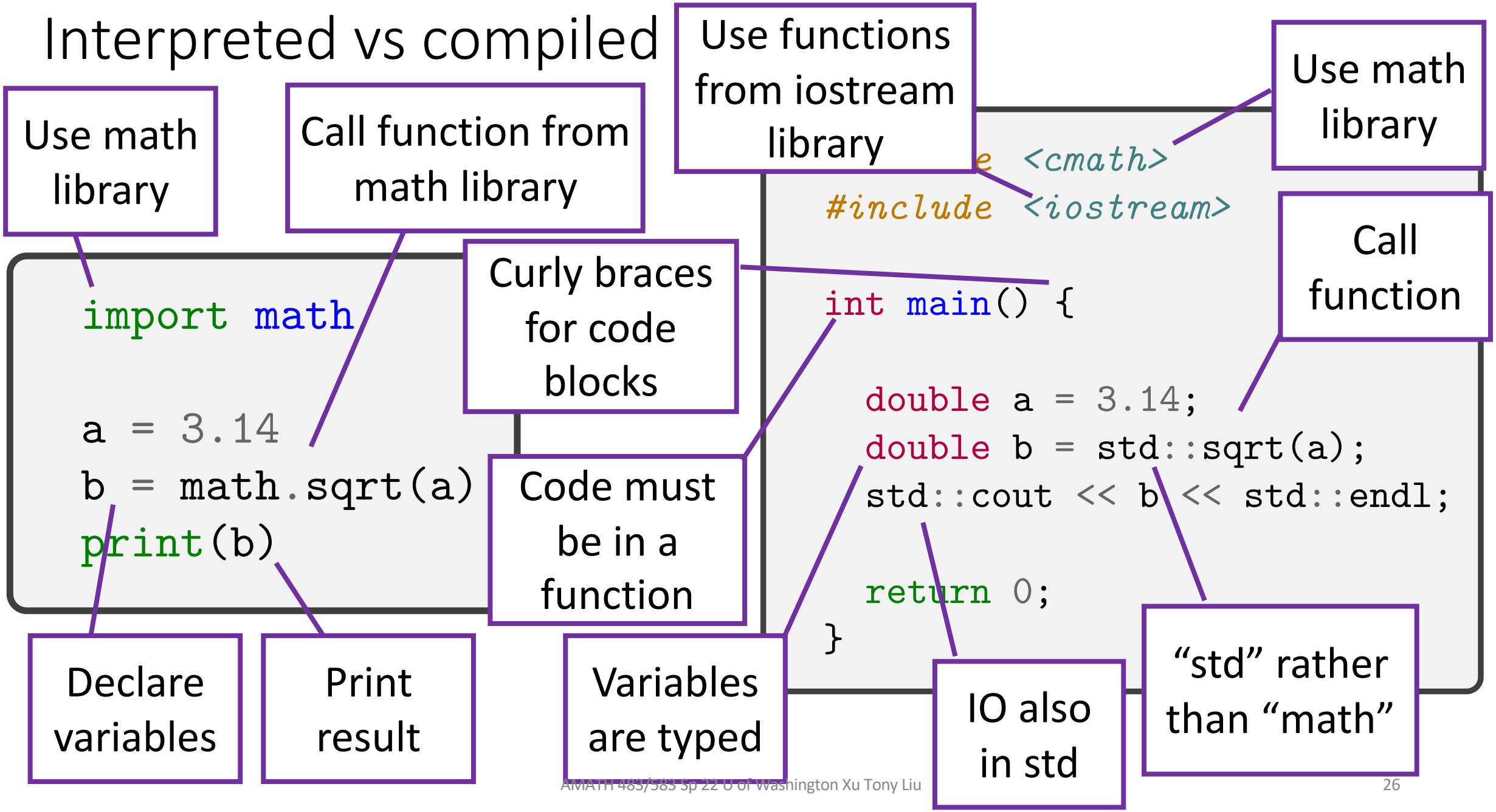
```
import math

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

It needs to be turned into code that can run

An "executable"

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)

```cpp
#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

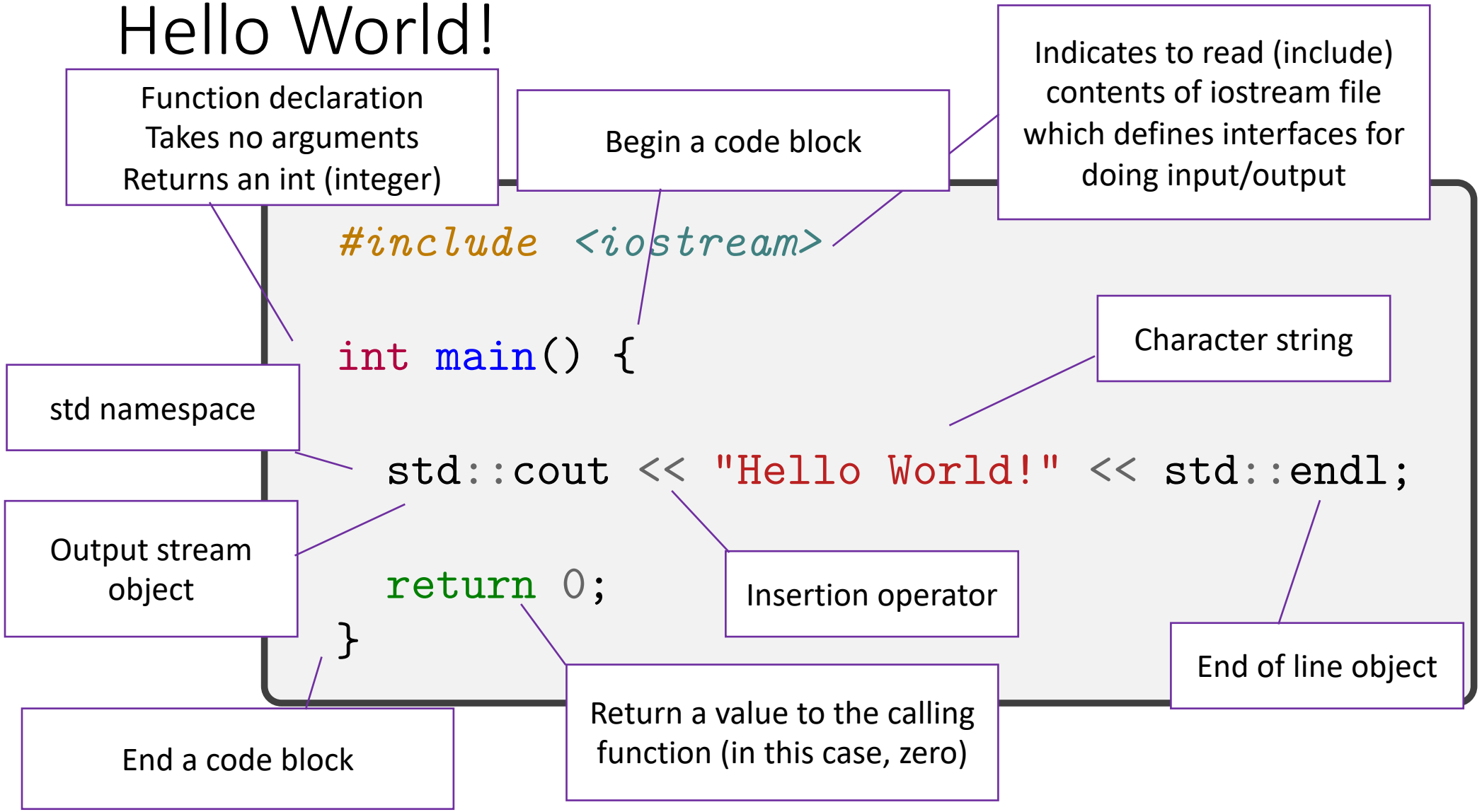
From C++ Core Guidelines

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() {
Character string
std namespace
std::cout << "Hello World!" << std::endl;
Output stream object
return 0;
}
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 two slashes //

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

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

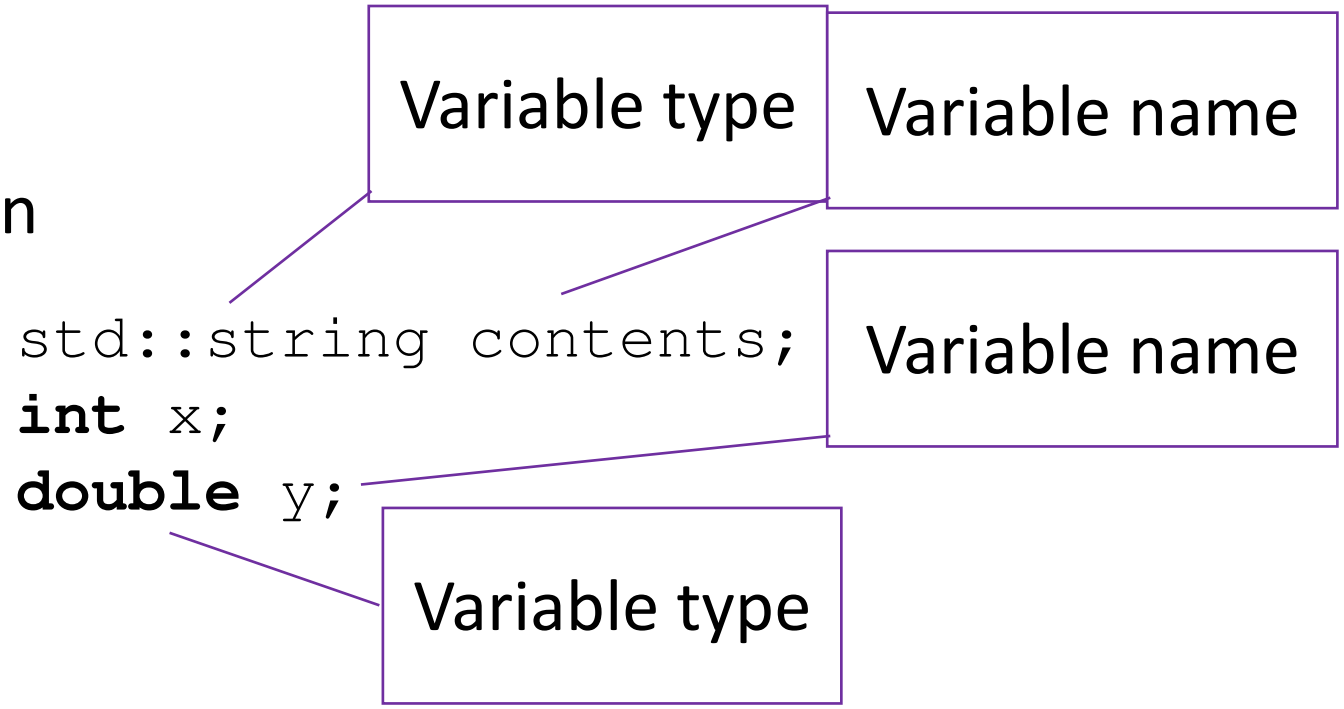
```
#include <algo

int main() {

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

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

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 mean 'count'?
  std::cout << message << std::endl;
  ~~~~~^~~~
       count
/usr/bin/../lib/gcc/x86_64-linux-gnu/5.4.0/../../../../include/c++/5.4.0/bits/stl_algo.h:3959:5:
      '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

- provides a scope to the identifiers (the names of types, functions, variables, etc) inside it
- organize code into logical groups
- prevent name collisions that can occur especially when your code base includes multiple libraries

Namespaces

```cpp
#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

```cpp
#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

```cpp
#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

```cpp
#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

```cpp
#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

```cpp
#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

```cpp
#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 = 3.14159265358979323846264338327950288419 7;

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 namespa

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

```cpp
#include <iostream>

namespace amath483 {
  double pi = 3.14159;
};

namespace amath583 {
  double pi = 3.14;
};

double pi = 3.14159265358

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 namespa

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

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

namespace amath483 {
  double pi = 3.14159;
};

namespace amath583 {
  double pi = 3.14;
};

double pi = 3.14159265358979323846264338327950288419716939937510;

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;
}
```

Include iomanip

Set precision to 15 digits

So what about namespace std?

```cpp
#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
  - Why not?
- Don’t write using namespace in a header file
  - Why?

First option for AMATH 483/583

```cpp
#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

```cpp
#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

```cpp
#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

```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;
}
```

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

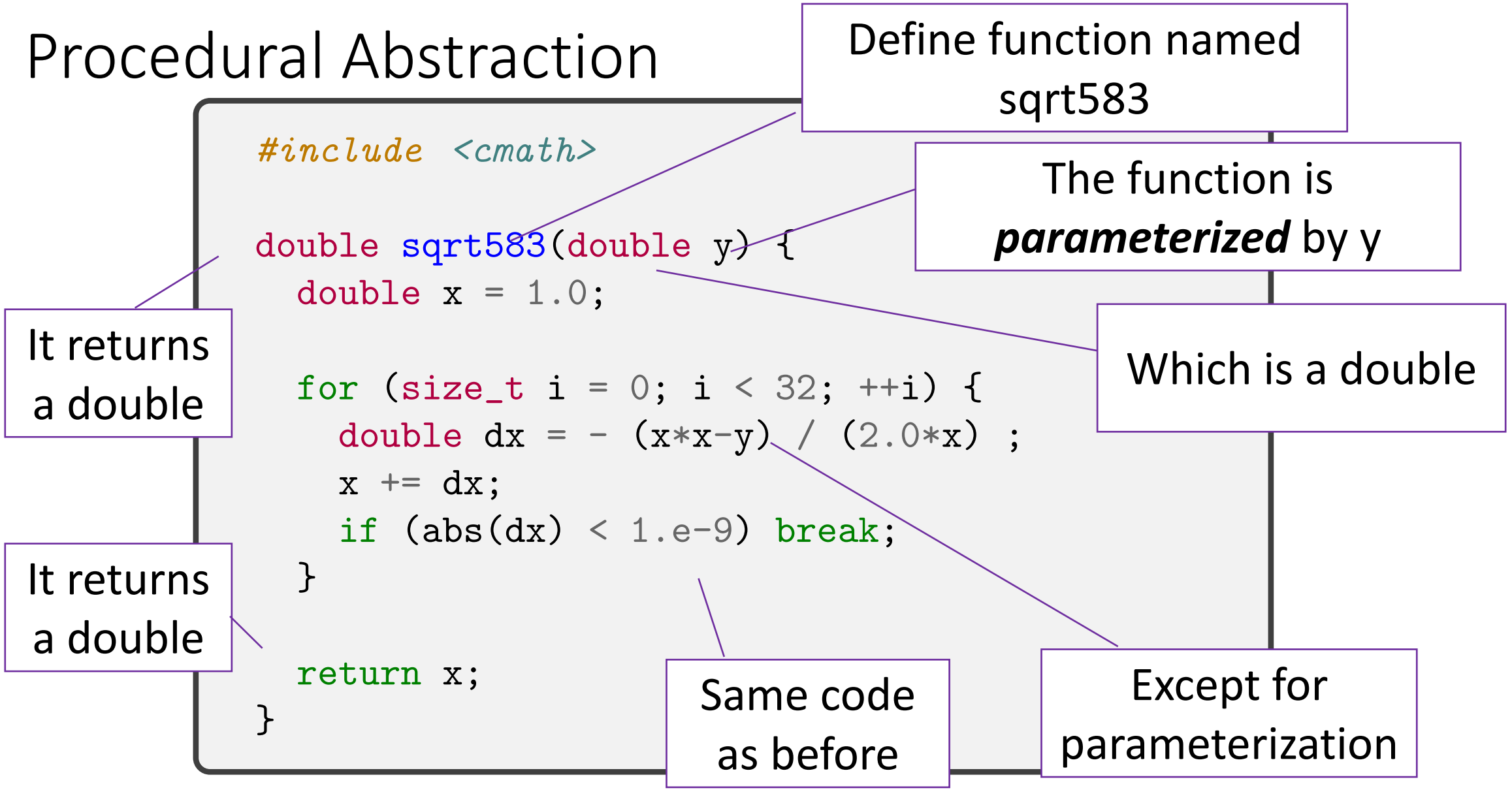
This is the only difference

```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 the same

This is the only difference

Procedural Abstraction




```
#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;
}
```

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

```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;
}
```

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?

```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;
}
```

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

```cpp
#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

```cpp
#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

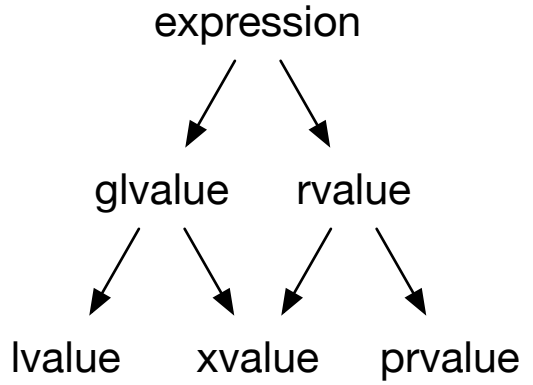
What's an l-value?

```
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;

    turn 0;
```

l-values and r-values

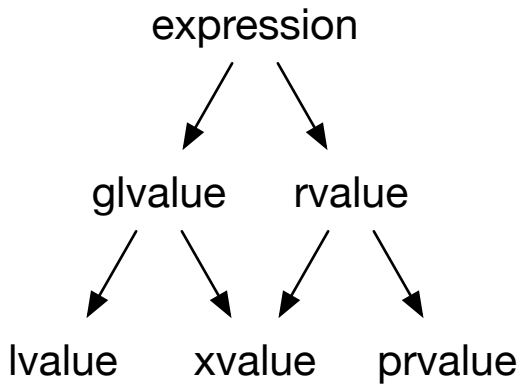
- Section 3.10 of C++ standard
  - A *glvalue* is an expression whose evaluation determines the identity of an object, bit-field, or function.
  - A *prvalue* is an expression whose evaluation initializes an object or a bit-field, or computes the value of the operand of an operator, as specified by the 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



glvalue: “generalized” lvalue
prvalue: “pure” rvalue
xvalue: “an “expiring” value

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

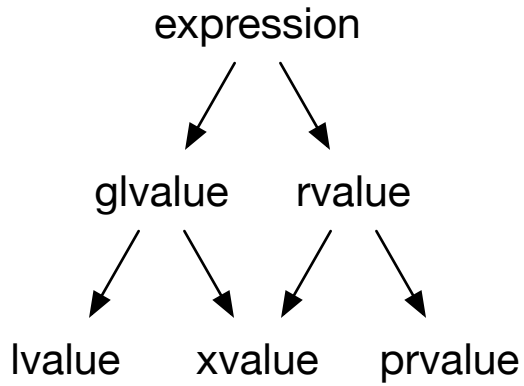


l-values and r-values

```
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);

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

Temp x+y can be referenced if read only

OK to reference const rvalue

l-values and r-values

- How is the value used in the function

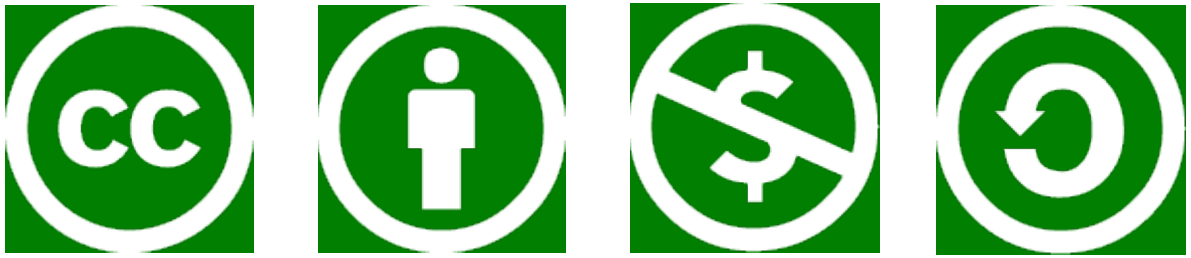
```
double sqrt583(const double& y) {
  double x = 0.0, dx;

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

  y = x;

  return x;
}
```

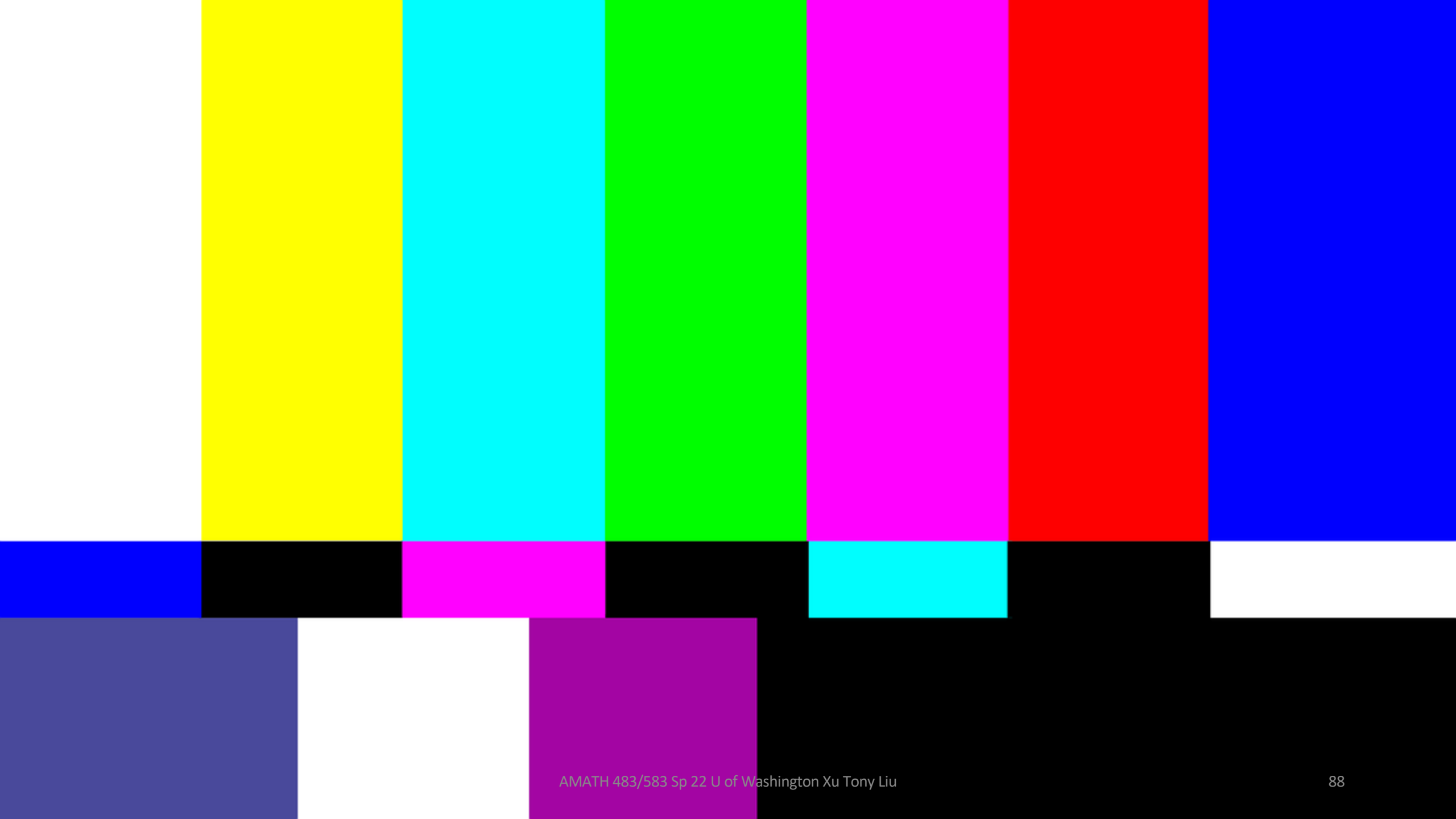
Const lvalue ref

```
$ 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.
```

Thank you!

Creative Commons BY-NC-SA 4.0 License





Example with Input

All variables in C++ must be typed!

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

int main() {

    std::string contents;

    std::cin >> contents;

    std::cout << contents << std::endl;

    return 0;
}
```

Variable declaration

Variable type is a std::string

Variable name is contents

Input Object

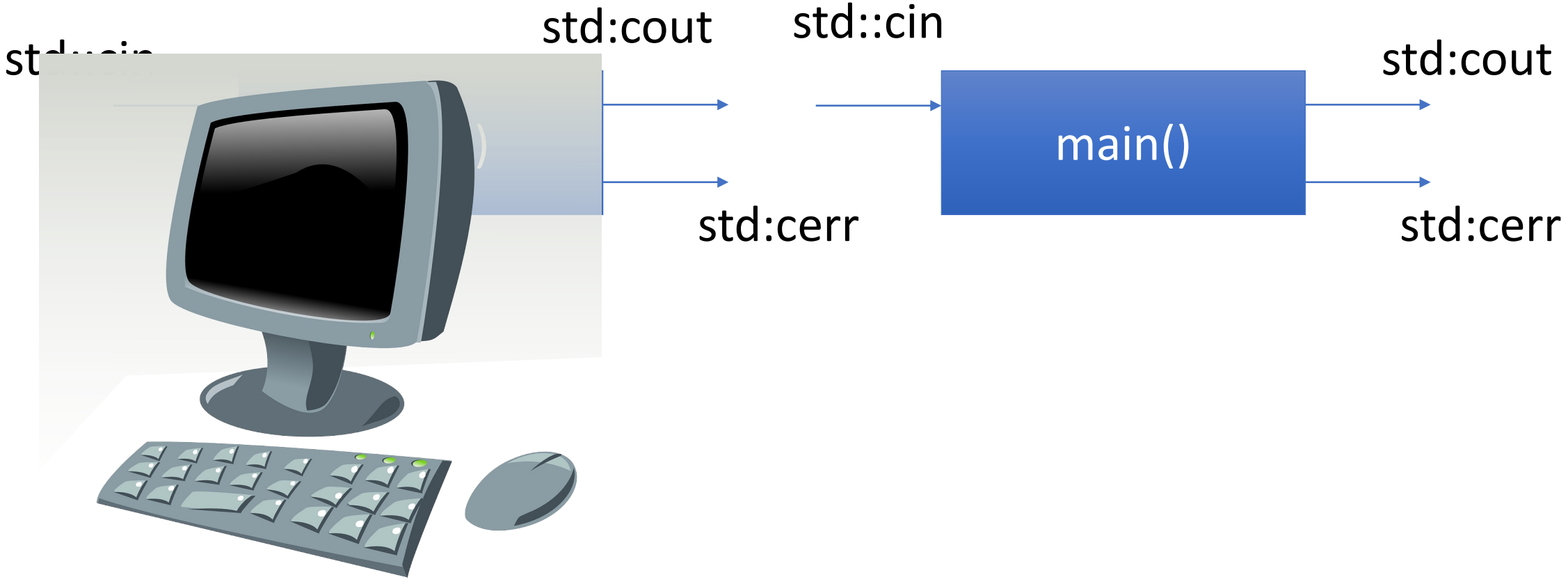
Result

```
$ c++ demo.cpp
$ ./a.out
Today is a good day for HPC!
Today
```

- Explain

Aside (Standard I/O)

- When text is entered into bash, it is accumulated and sent to the program after CR is entered (there are ways to change this: stty)



Example

```
$ wc
int main() {
  std::cout << "Hello World" << std::endl;
  return 0;
}
      4      12      70
$ cat b.cpp | wc
      4      12      70
$ wc b.cpp
 4 12 70 b.cpp
```

Word count (man wc)

Tty input (all the hello world text)

pipe

4 lines, 12 words, 70 characters

Pipe the text from b.cpp into wc

Read contents from b.cpp

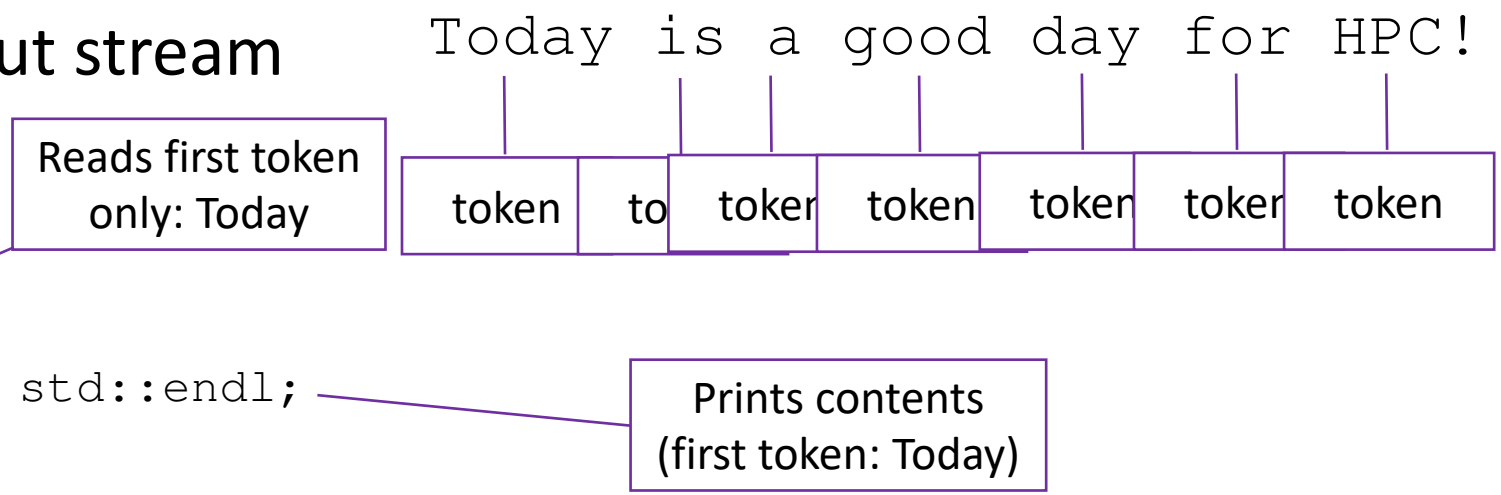
Explanation

- When text is entered into bash, it is accumulated and sent to the program after CR is entered (there are ways to change this: stty)
- This entire string is put into the input stream of the program

```
Today is a good day for HPC!
```

- cin **tokenizes** the input stream

```
int main() {
  std::string contents;

  std::cin >> contents;
  std::cout << contents << std::endl;

  return 0;
}
```



Next Attempt

```
int main() {
  std::string contents;

  std::cin >> contents;
  std::cout << contents;

  std::cin >> contents;
  std::cout << contents;

  std::cin >> contents;
  std::cout << contents;

  std::cin >> contents;
  std::cout << contents;

  std::cin >> contents;
  std::cout << contents;

  std::cin >> contents;
  std::cout << contents;

  std::cin >> contents;
  std::cout << contents;

  std::cin >> contents;
  std::cout << contents;

  std::cin >> contents;
  std::cout << contents << std::endl;

  return 0;
}
```

```
$ c++ demo2.cpp
$ ./a.out
Today is a good day for HPC!
TodayisagooddayforHPC!
```

- Explain

Yet Another Attempt

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

int main() {

  std::string contents;

  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << std::endl;


  return 0;
}
```

```
$ ./a.out
Today is a good day for HPC!
Today is a good day for HPC!

$ ./a.out
Today is a good day for
Today is a good day for

$ ./a.out
Today is a good day for
Today is a good day for HPC
HPC
```

Stuck

One more token

Final token

What Else is Wrong?

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

int main() {

  std::string contents;

  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << "␣";
  std::cin >> contents;
  std::cout << contents << std::endl;


  return 0;
}
```

```
$ ./a.out
Today    is  a  good day for    HPC!
Today is a good day for HPC!
```

Getting a Line of Input

- Use std::getline()

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

int main() {

  std::string contents;
  std::getline(std::cin, contents);
  std::cout << contents << std::endl;

  return 0;
}
```

getline() function

Stream to get line from

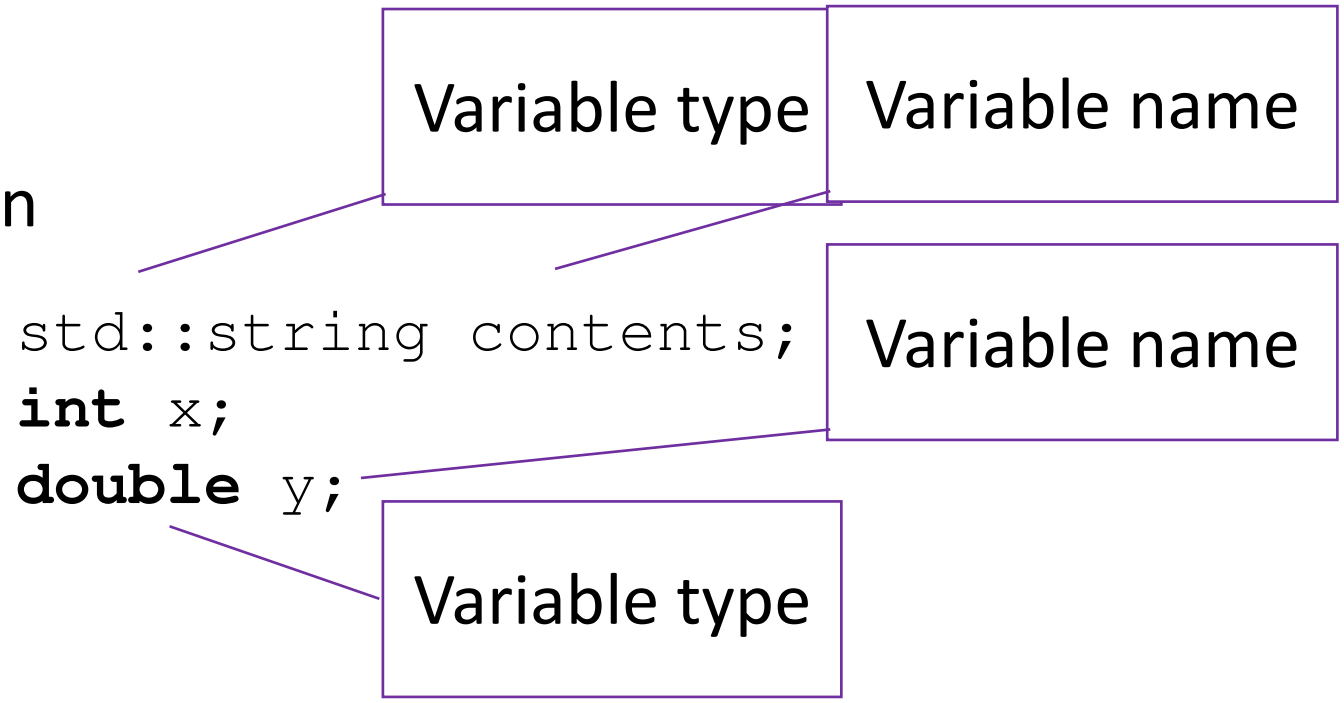
Where to put the line

```
$ ./a.out
Today is a good day for HPC!
Today is a good day for HPC!
```

- Gets entire line of text, with no tokenization
- Make sure you understand getline() vs >>

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

More about string

```
std::string s;

std::string t = "Hello World";

std::string u = t;

std::string v = s + t;

int length = v.size();
```

Declare empty string

Declare string object and initialize with characters (Note “Hello World” is not a C++ string object)

Declare string and copy from t

+ operator concatenates two string objects

size member function returns length of string

Example

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

int main() {
  std::string msg_1   = "Hello";
  std::string msg_2   = "World";
  std::string message = msg_1 + "␣" + msg_2;
  int msg_length      = message.size();

  std::cout << "There␣are␣" << msg_length << "␣characters␣in␣";
  std::cout << "\"" << message << "\"" << std::endl;

  return 0;
}
```