CS4414 Recitation 1

Introduction and C++ basic

08/2024

Alicia Yang

Logistics

- TA Help Session: C++ Coding Environment Setup
  - Session 2: 7:30 PM - 8:30 PM, Tuesday, 09/03 (led by Noam)
  - Location: Uris Hall, Room G01

HW1 will be released this afternoon

Ed discussion announcement

The writeup and starter code are on **Canvas**

Submission to **Gradescope**

No slip days

Overview

- Recitation introduction
- Overview coding environment
- C++ primitive types

Recitation Goals

How to write **good** system program in C++

"Clean code does one thing well.

- The logic should be straightforward to make it hard for bugs to hide,
- the dependencies minimal to ease maintenance,
- error handling complete according to an articulated strategy,
- and performance close to optimal…"

--- Bjarne Stroustrup, the inventor of C++

How to write **good** system program in C++

1. clean and correct code

Write clean and correct code

- The basics: C++ types, variable …
- Classes and functions
- Memory management in C++, RAII principle
- Smart pointers in C++
- C++ templates
- Standard containers – std::vector<T>, std::map<K,V>

How to write **good** system program in C++

1. clean and correct code

2. Develop efficient system

Develop efficient system

- gprof for program profiling, valgrind for memory check
- Make efficient use of hardware
  - Hardware parallelism
  - Multithreading and synchronization

Recitation plan

Learn about how to write good system programs in C++

Assignment introduction and explanation

Exam preparation and reviews

Make the recitations useful

Ask questions

Understand and run the recitation example code

Demystify how C++ system programs work

CPP Reference

https://en.cppreference.com/w/

CPP Reference

Example

Demonstrates how to inform a program about where to find its input and where to write its results.
A possible invocation: `./convert table_in.dat table_out.dat`

Run this code

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

int main(int argc, char *argv[])
{
    std::cout << "argc == " << argc << '\n';

    for (int ndx{}; ndx != argc; ++ndx)
        std::cout << "argv[" << ndx << "] == " << std::quoted(argv[ndx]) << '\n';
    std::cout << "argv[" << argc << "] == "
              << static_cast<void*>(argv[argc]) << '\n';

    /* ... */

    return argc == 3 ? EXIT_SUCCESS : EXIT_FAILURE; // optional return value
}
```

Possible output:

```
argc == 3
argv[0] == "./convert"
argv[1] == "table_in.dat"
argv[2] == "table_out.dat"
argv[3] == 0
```

C++ Coding Environment Setup

CS4414 programming environment

- Use **Ubuntu** (a Linux distribution derived from Debian) environment to write C++ programming assignments.
- Assignments are submitted to Gradescope
  - To standardize the grading environment, we use **Ubuntu22.04** as base image to compile and grade your HW assignments

OS and Compiler matters

OS	macOS	Windows	Ubuntu
Default C++ Compiler	Clang (from Xcode Build System)	Microsoft Visual C++ (MSVC)	GCC (GNU Compiler Collection)

Different C++ compiler may result in different compilation results

```
int foo(int num) {
    if(num % 2 == 1)
        return num * num;
    else
        return num * num +1;
}
```

Clang compiler

% clang++ -S -o main.s main.cpp

```
.globl    __Z3fooi
; -- Begin function _Z3fooi
        .p2align  2
__Z3fooi:                               ;
@_Z3fooi
        .cfi_startproc
; %bb.0:
        and w8, w0,
#0x80000001
        cmpw8, #1
        mul w8, w0, w0
        cinc w0, w8, ne
        ret
```

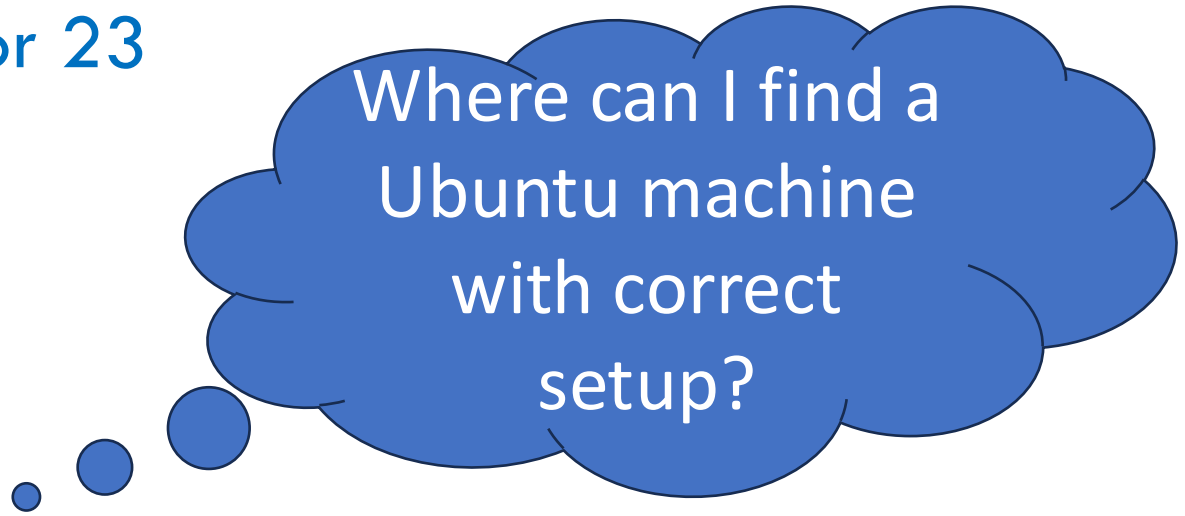
GCC compiler

% g++ -S -o main.s main.cpp

```
        .cfi_startproc
        endbr64
        movl    %edi, %edx
        movl    %edi, %eax
        shrl $31, %edx
        imull   %edi, %eax
        addl%edx, %edi
        andl$1, %edi
        subl%edx, %edi
        xorl %edx, %edx
        cmpl    $1, %edi
        setne   %dl
        addl%edx, %eax
        ret
```

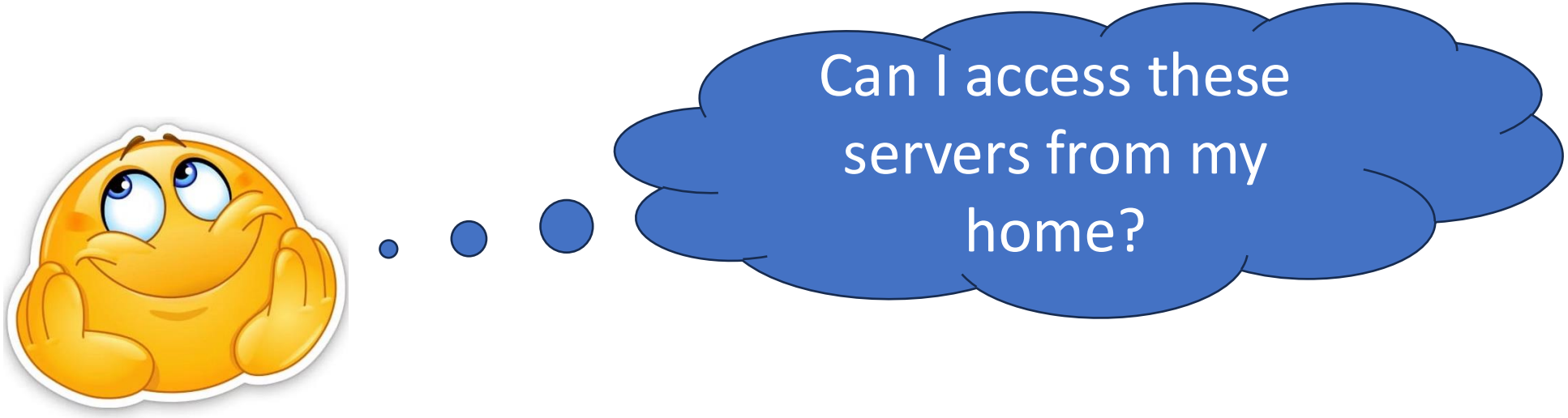
C++ Coding Environment for course assignments

- Compilation tools: GNU Compiler Collection(GCC) with **gcc-8 or recent**
- C++ compiler version: 20 or 23



C++ Coding Environment for course assignments

- Servers from Cornell Engineering cluster:
  - ugclinux server (link)

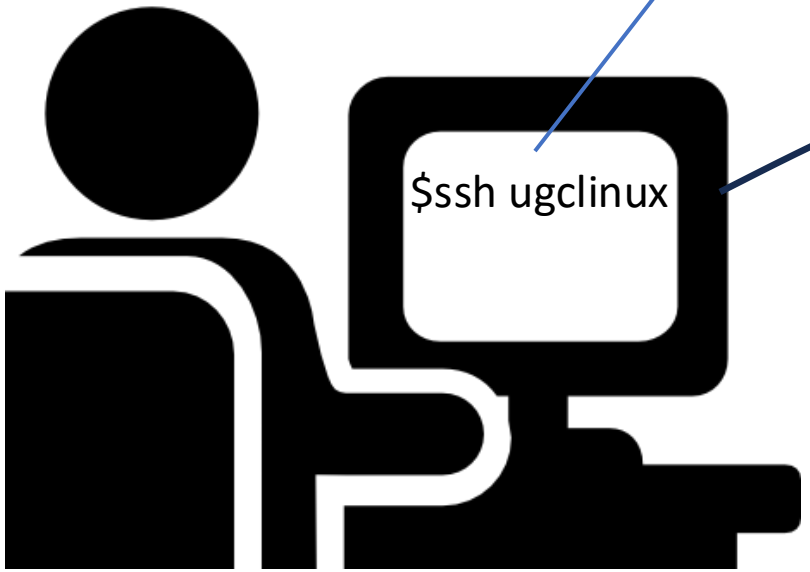


Yes, remote access

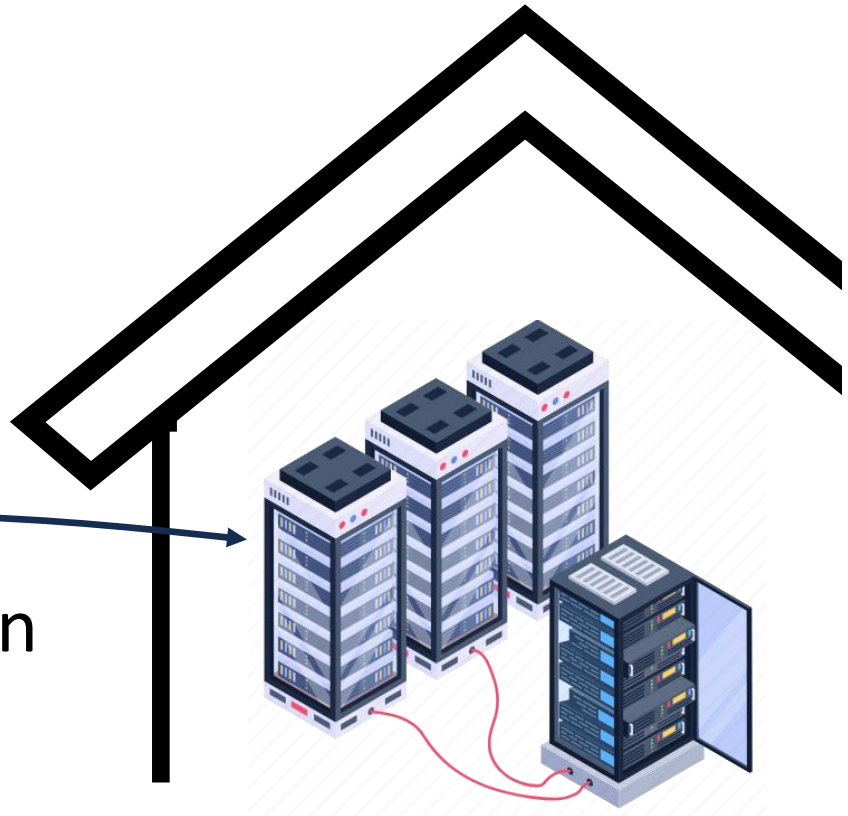
Remote access Linux Ubuntu server via ssh

Command:

$ ssh [net_id]@ugclinux.cs.cornell.edu

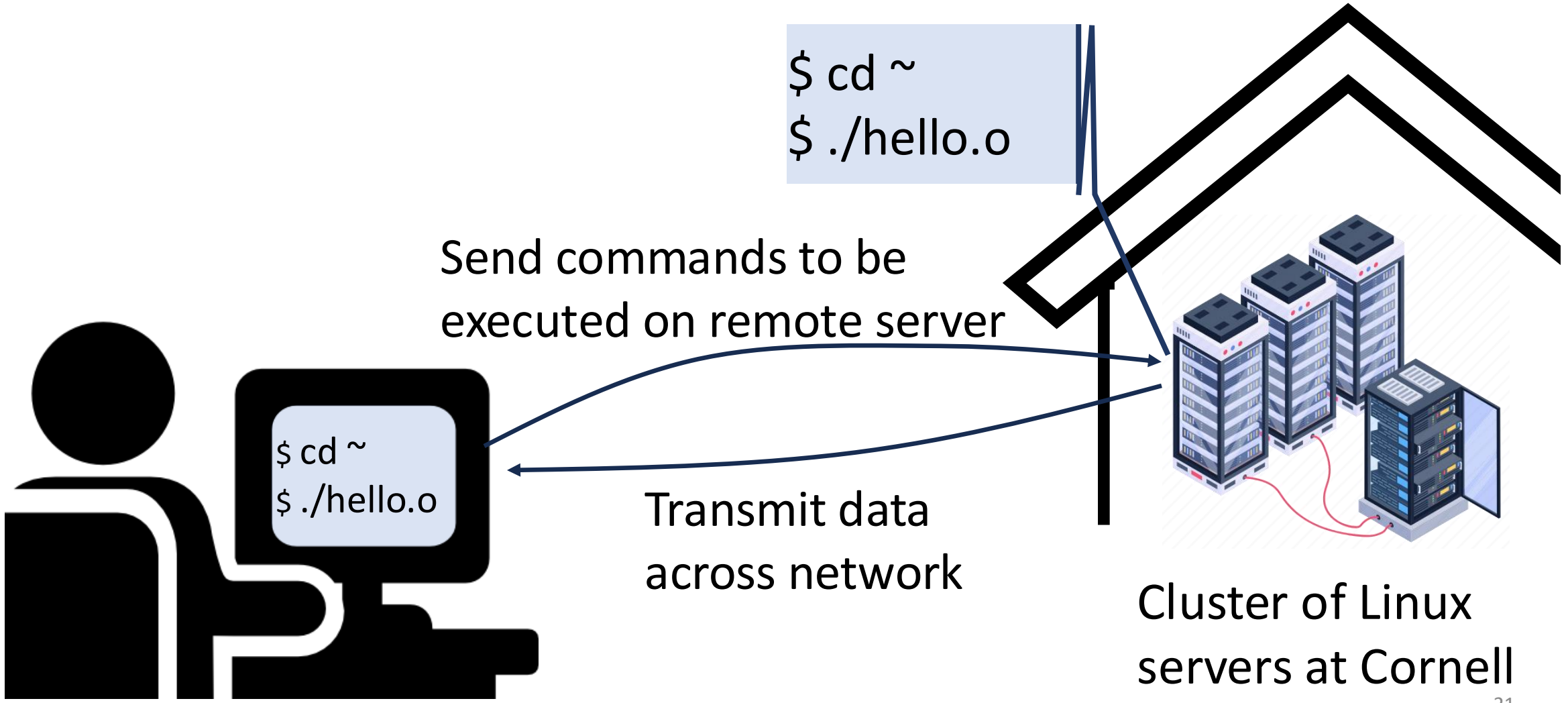


Set up connection to the remote server



Cluster of Linux servers at Cornell

Remote access Linux Ubuntu server via ssh

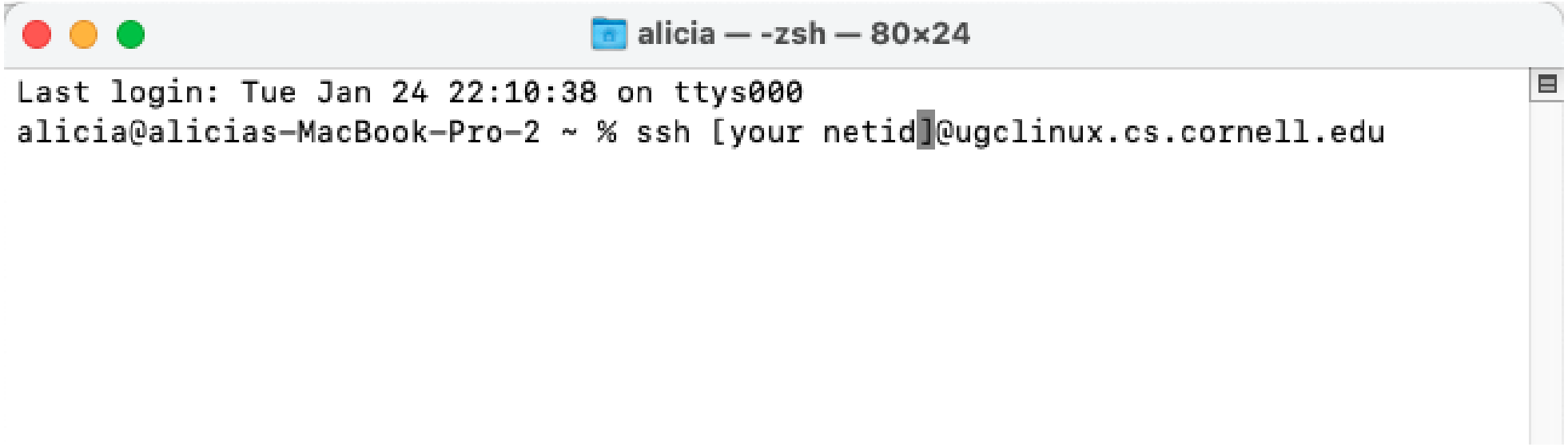


Connect to remote ugclinux server

Command line

From terminal login to ugclinux server, via ssh tunnel

% ssh [your netid]@ugclinux.cs.cornell.edu

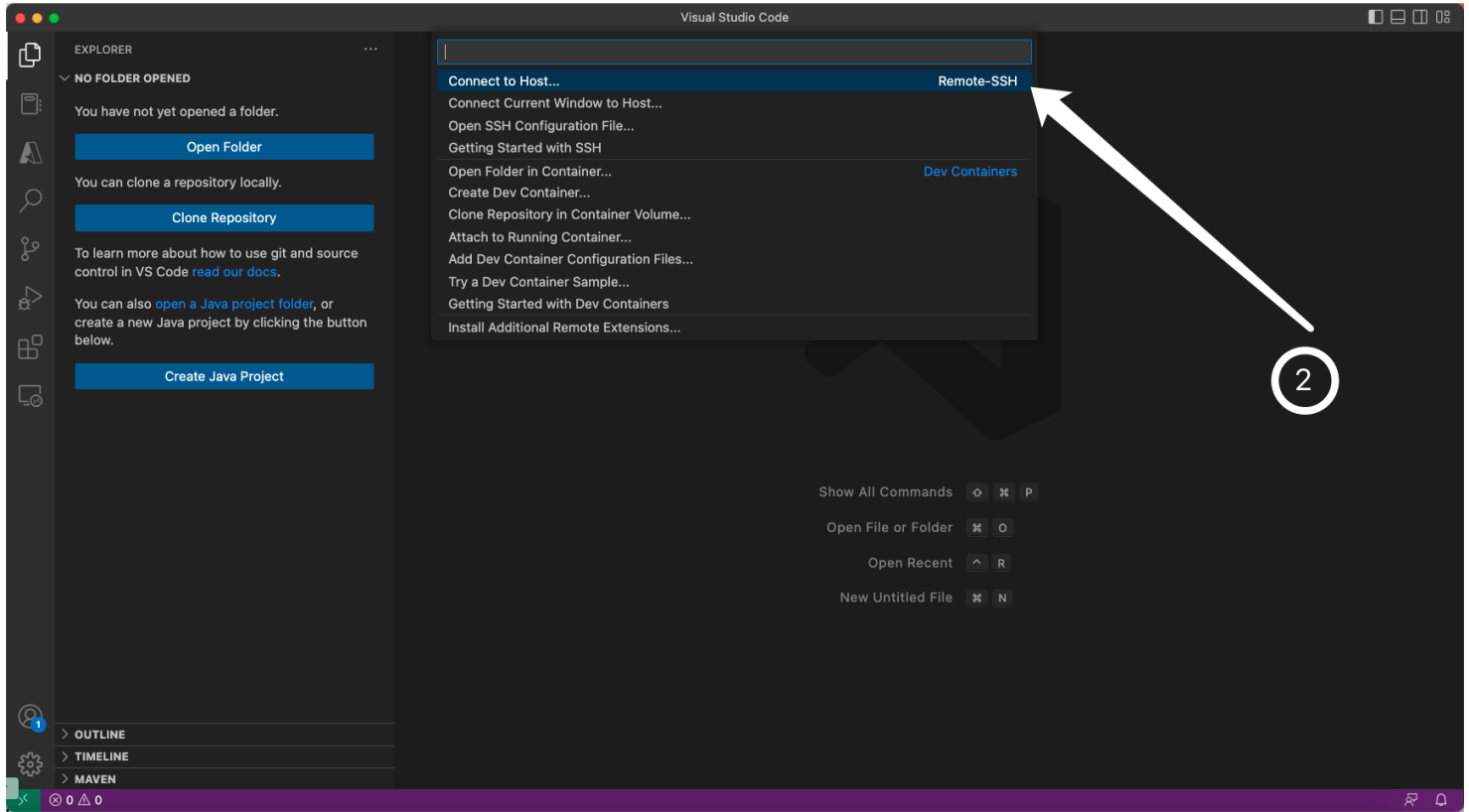


Connect to remote ugclinux server

VS Code

Download Visual Studio Code on your computer (link)

Use Remote-ssh on VS code to access ugclinux



More detailed step-by-step tutorial

- TA Help Session: C++ Coding Environment Setup
  - Session 1: 7:30 PM - 8:30 PM, Thursday, 08/29 (led by Austin)
  - Session 2: 7:30 PM - 8:30 PM, Tuesday, 09/03 (led by Noam)
  - Location: Uris Hall, Room G01

Running C++ programs

Helloworld.cpp example

demo

```
#include <iostream>


int main() {
    std::cout << "Hello world!" << std::endl;
    return 0;
}
```

Program starting point
Every C++ program must have exactly one main() function.

Helloworld.cpp example

Instruct the compiler to include the declaration of the standard stream I/O facilities in iostream

```
#include <iostream>


int main() {
    std::cout << "Hello world!" << std::endl;
    return 0;
}
```

Helloworld.cpp example

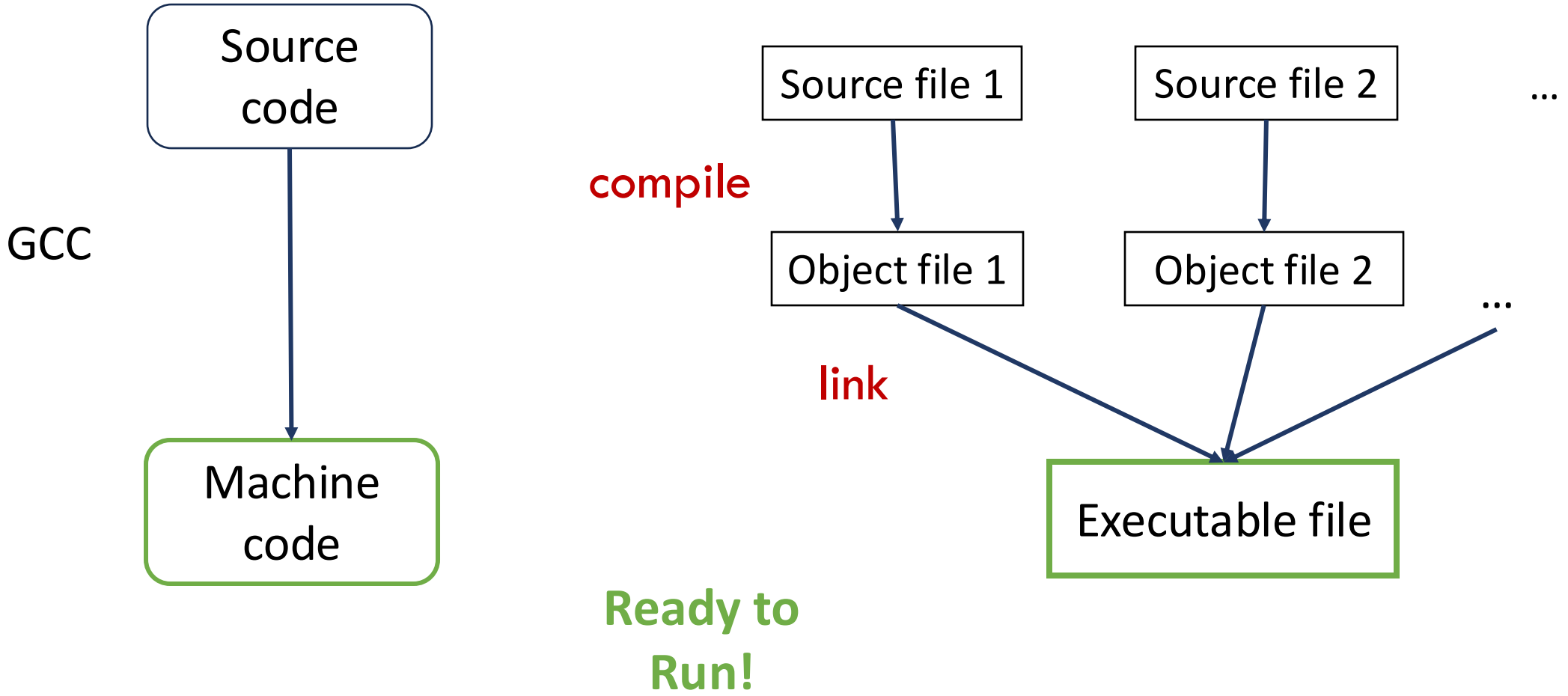
```
#include <iostream>


int main() {
    std::cout << "Hello world!" << std::endl;
    return 0;
}
```

std:: (standard library)
specifies that the name cout to be found
in the standard library namespace

Operator << , writes its second argument to its first.
(write “Hello world” to
the standard output stream std::cout)

C++ is a compiled language



Compile and Run your C++ Code

Command line

1. Compile your C++ program with simple line below

```
%   g++  -std=c++20  -Wall  helloworld.cpp -o helloworld
```

- Flags:
  - -std=c++20: specify the compiler version to use C++20
  - -Wall: allow all compiler warnings to be printed out
  - -o: specify the name of the output executable

How to debug my code?

demo

1. Compile your C++ program with line below

```
%   g++  -std=c++20 -g  -Wall  helloworld.cpp -o helloworld
```

- Flags:
  - -g flag: include debug symbols

C++ Built-in Types

What is C++?

A federation of related languages, with four primary sublanguages

- **C**: C++ is based on C, while offering approaches superior to C. Blocks, statements, processor, built-in data types, arrays, pointers, etc., all come from C
- **Object-Oriented C++:** "C with Classes", classes including constructor, destructors, inheritance, virtual functions, etc.
- **Template C++:** generic programming language. Gives a template, define rules and pattern of computation, to be used across different classed.
- **STL(standard template library):** a special template library with conventions regarding containers, iterators, algorithms, and function objects

C++ types

- Primitive(fundamental) data types
  - bool / bool*
  - char / char*
  - int / int*
  - float / float*
  - double / double*
  - ……
- Derived data types
  - pointer
  - array
  - function
- User-defined data types
  - class
  - struct

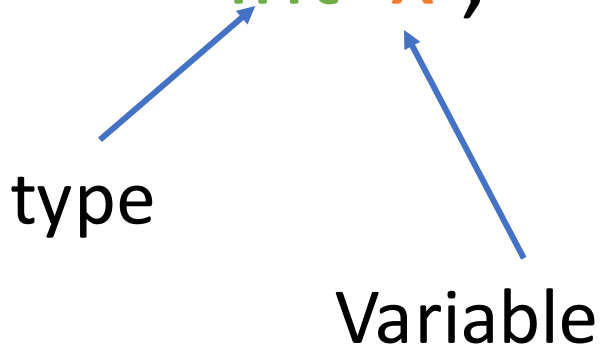
C++ types

- bool // boolean, possible values are true and false
- char // character, such as ‘a’, ‘z’, ‘9’, ‘\” ..
- int // integer, such as 36, -273, 10006, ..
- double//double-prevision floating-point number, such as 3.14, 230421.0, ..
- unsigned // non-negative integer, such as 0, 365,…
- uint8_t // 8-bit(1-byte) unsigned integer, such as 0, .. 200, .. 255

C++ is strongly typed

- A **declaration** is a statement that introduce a name to the program with a specified type

```
int  x ;    // declaration
```



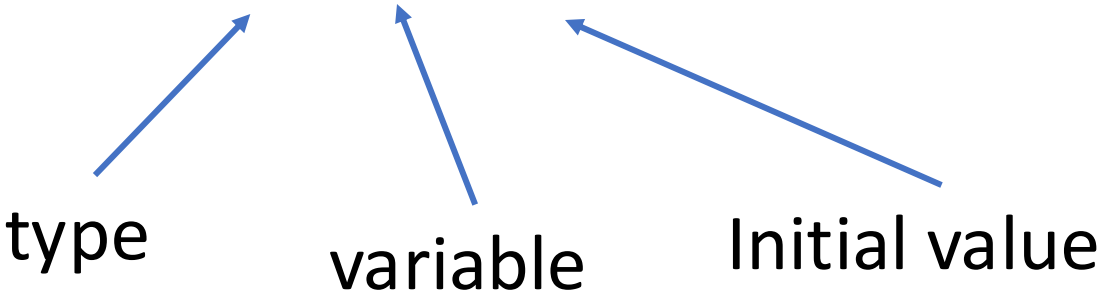
C++ is strongly typed

- A declaration is a statement that introduce a name to the program, with a specified type

```
int  x ;        // declaration
```

- A **declaration** can also follow with an **initialization**

```
int  x = 5;     // declaration + initialization
```



C++ is strongly typed

- A declaration is a statement that introduce a name to the program with a specified type

```
int x ;        // declaration
```

- A declaration can also follow with an initialization

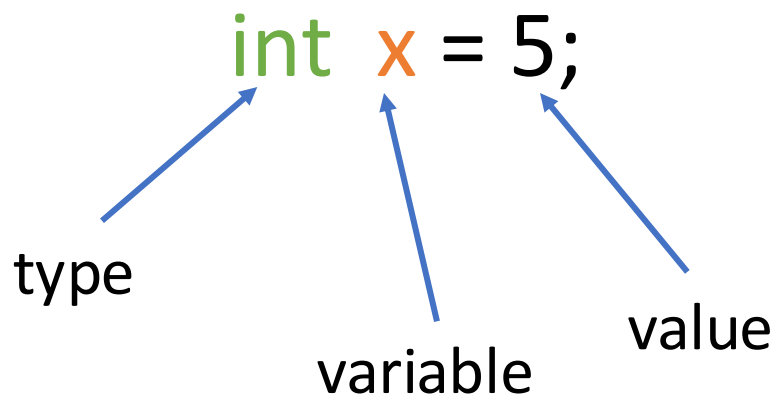
```
int x = 5;        // declaration + initialization
```

- Later, you can use variable x in **expressions** such as

```
int y = x + 1;        // initialization of y using x
x = 7;                // reassignment
```

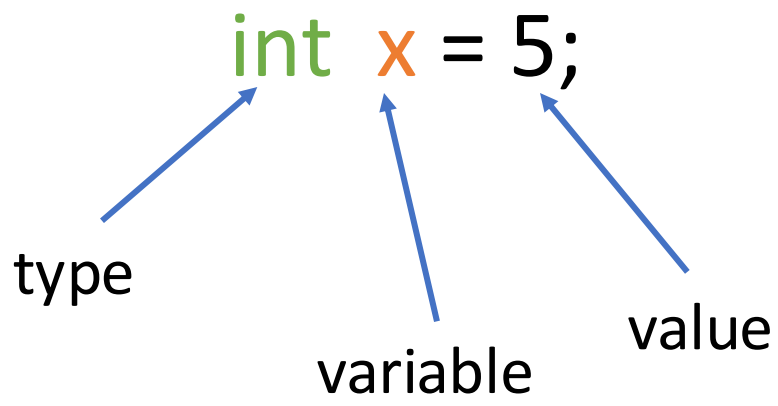
C++ is strongly typed

- A C++ variable has a name, a type, a value and an address in memory
  - A type: defines a set of **possible values** and **operations** that this variable can do



C++ is strongly typed

- A C++ variable has a name, a type, a value and an address in memory
  - A type: defines a set of **possible values** and **operations** that this variable can do
  - A value: a set of bits to be interpreted by its type



C++ fundamental data type

- Integer types with different sizes and signedness
  - int, short, unsigned int, long, long long, unsigned long, …
  - int8_t, int16_t, int32_t, int64_t, …
  - uint8_t, uint16_t, uint32_t, uint64_t, …

C++ fundamental type correspond to fixed sizes

- bool  // each boolean variable has 1 byte(8 bit)
- char 
- int 
- double 
- uint8_t 

Demo

C++ fundamental data type

- How do I find out the **size of a built-in type**?
  - Use the built-in function sizeof(variable name) or sizeof(<type>) to find out the size of the variable's **type**

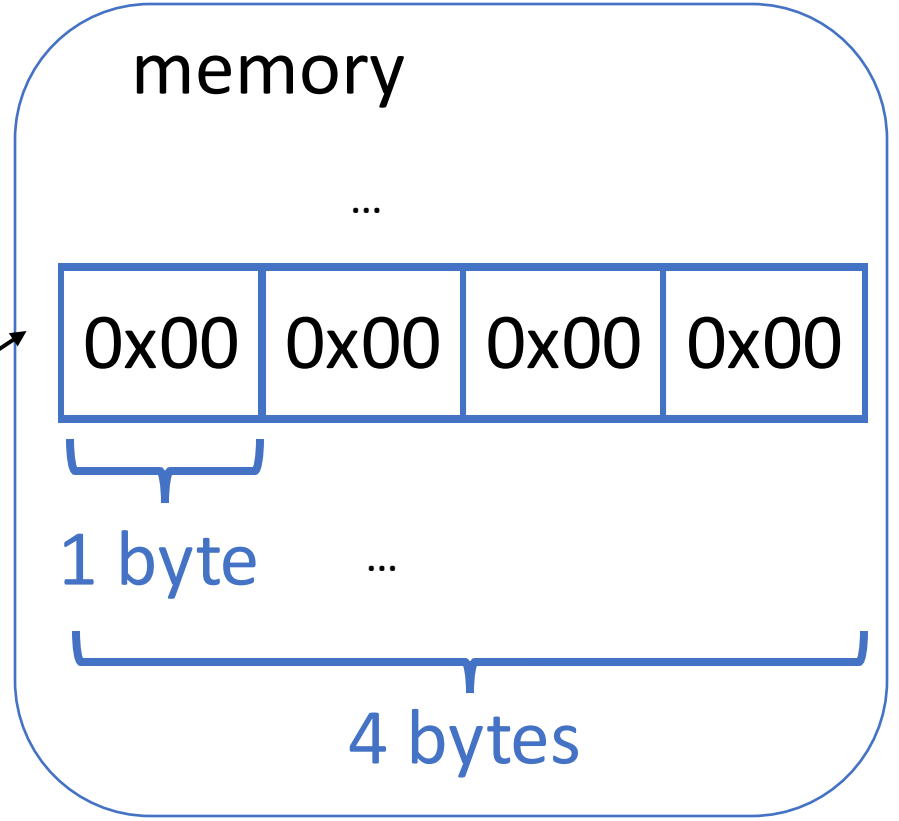
```
int x = 0;

std::cout << sizeof(x) << std::endl;                    // print 4

std::cout << sizeof(long long int) << std::endl;        // print 4
```

& | Address

```
int32_t  x  = 0;
```

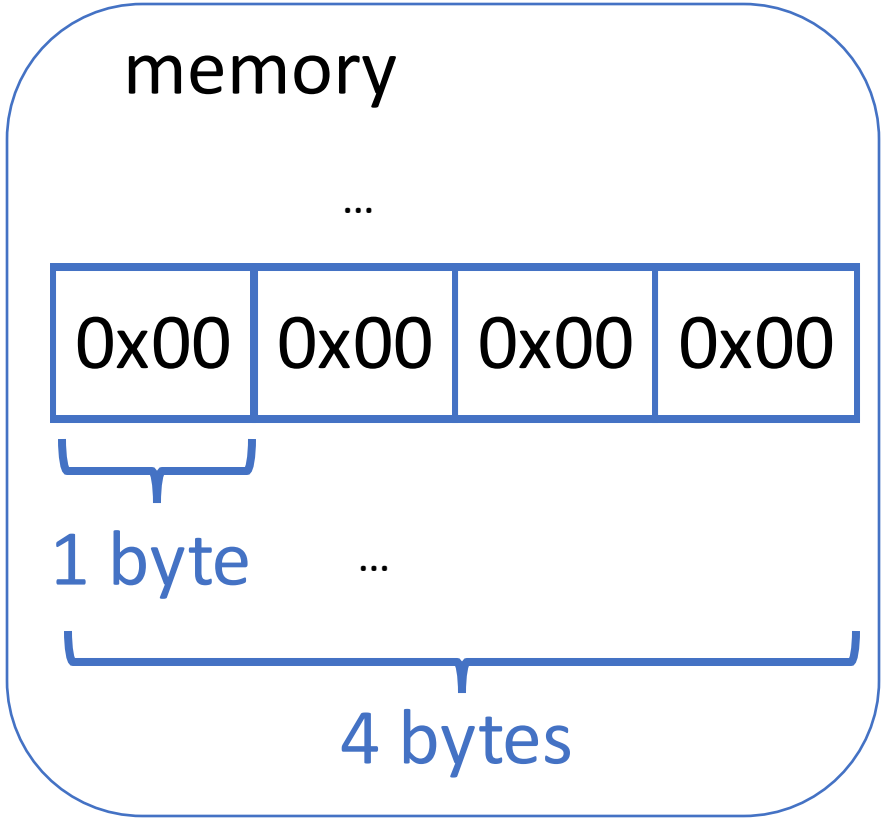


0x00 is a pair of hex number
(0x is the prefix, 00 is hex digits)

& Address

Where does x live in memory exactly?

```
int32_t x = 0;
```



& | Address

- Can obtain the **address** (represented in hex) with the **&** operator

```
int32_t x = 0;
std::cout <<  &x << std::endl;
                    // prints to the address of x
                       for example, 0x7ffd55bdaa4
```

& | Address

- Can obtain the **address** (represented in hex) with the **&** operator

```
int32_t x = 0;
std::cout <<  &x << std::endl;
                    // prints to the address of x
                       for example, 0x7ffd55bdaa4
```

Demo

& | Address

- Can obtain the **address** (represented in hex) with the **&** operator

```
std::cout <<  &x << std::endl;
```

- What happens if you use an **uninitialized** variable**?**

```
int32_t  x ;

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

& Address

- Can obtain the **address** (represented in hex) with the **&** operator

```
std::cout <<  &x << std::endl;

                                    // prints 0x7ffd55bdaa4
```

- What happens if you use an **uninitialized** variable?

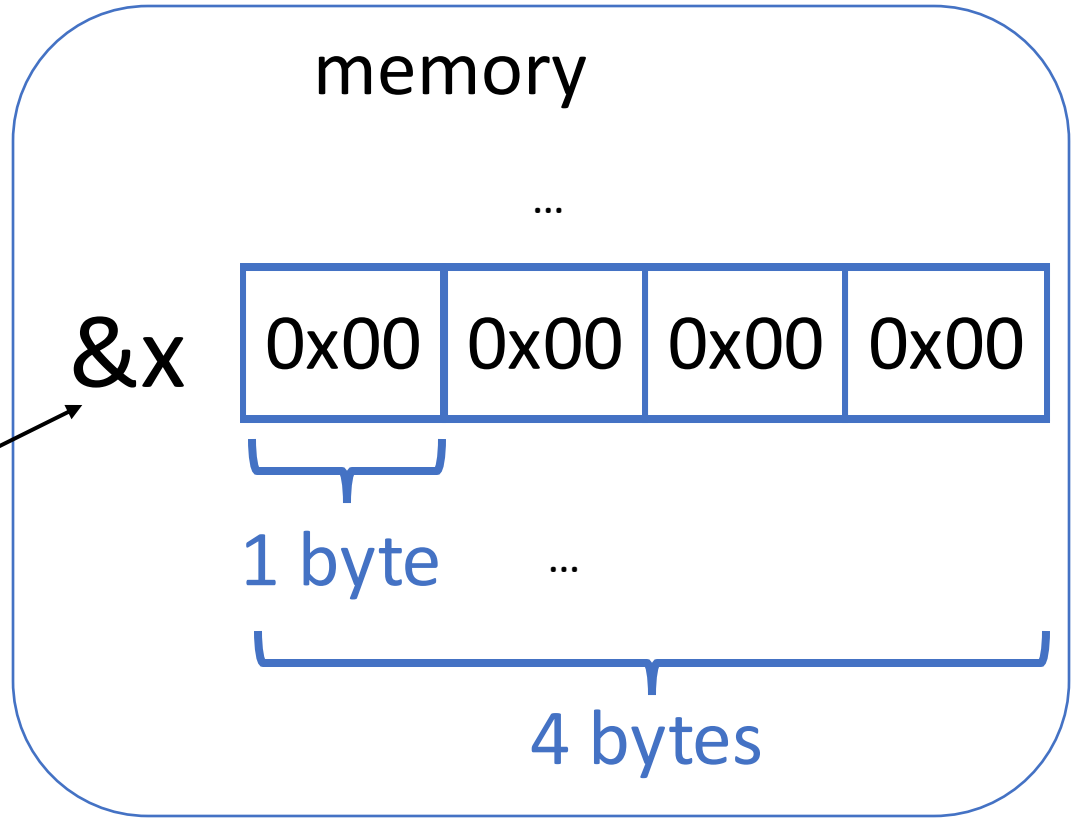
```
int32_t  x ;              // uninitialized value

std::cout << x << std::endl;
                    // the value of x is undefined
```

& | Address

Can I store &x in a variable to use in the future?

```
int32_t x = 0;
```



&x is the memory address of x

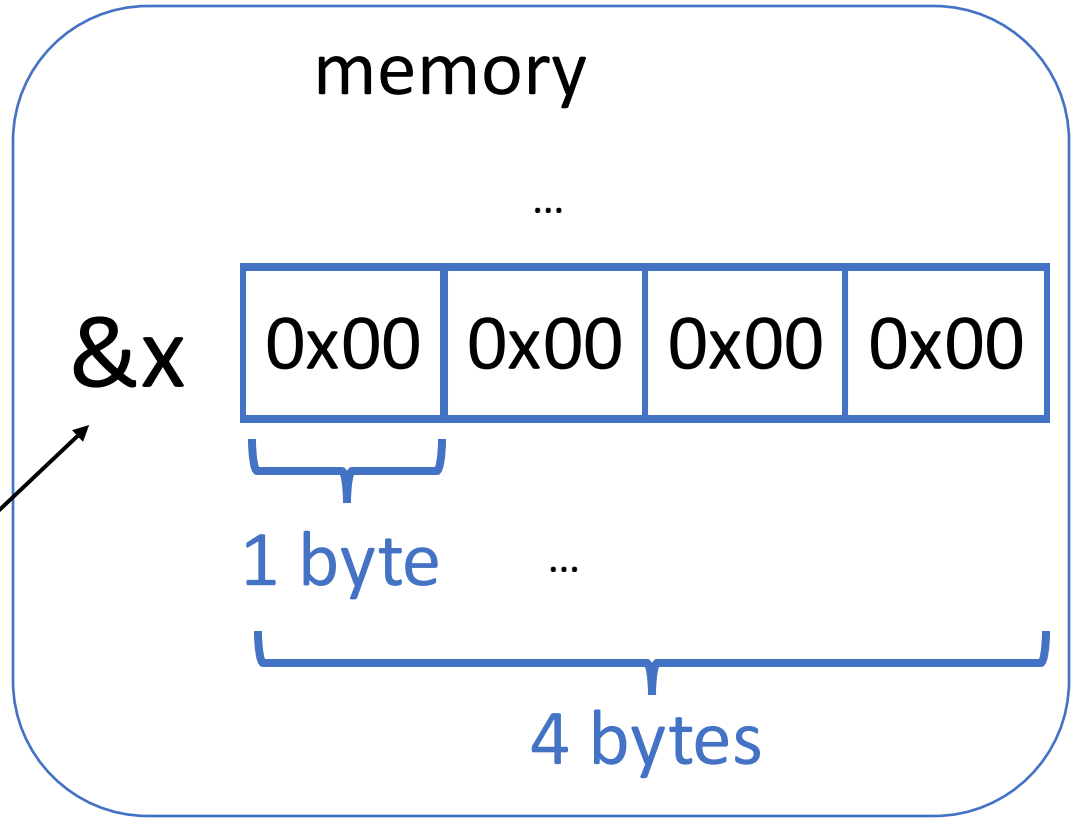
* Pointers

- A pointer is a variable that stores a memory address.

```
int32_t  x  = 0;

int32_t* px;

px = &x;
```



* Pointers

- A pointer is a variable that stores a memory address.
- A pointer is declared just like a variable but with * **after the type**

```
int32_t* px;
```

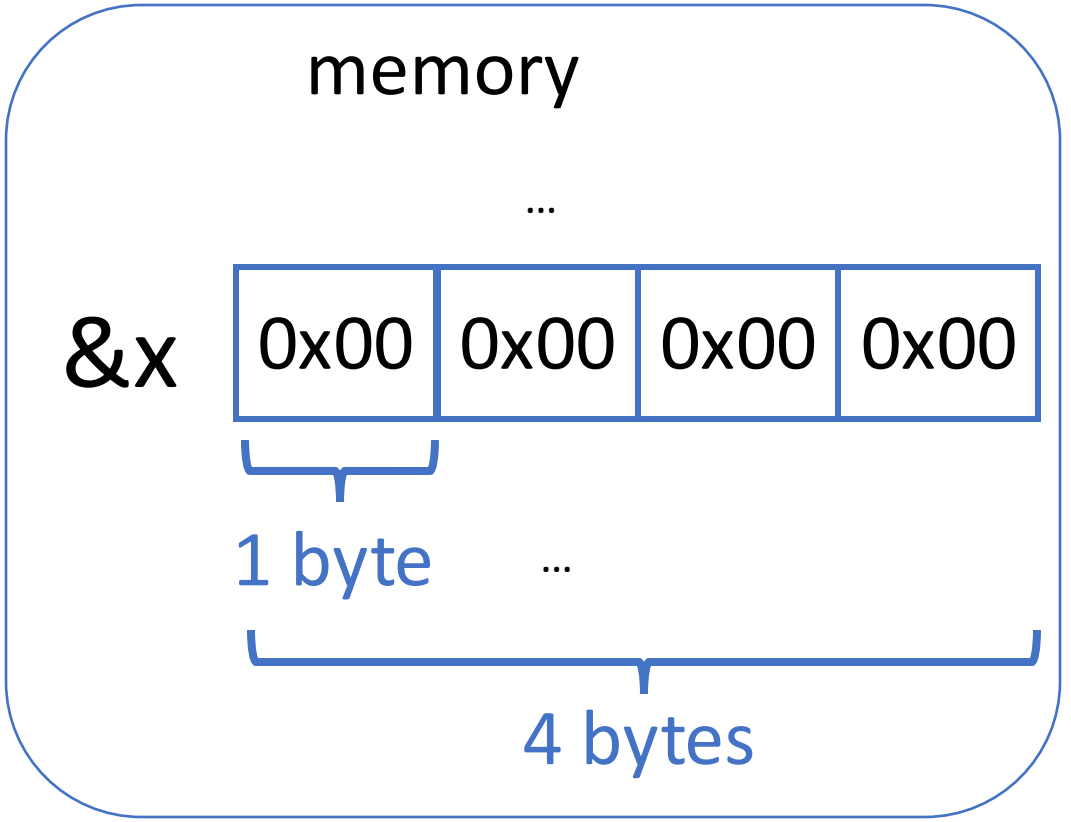
A pointer that could point to an integer

* Pointers

- **Dereferencing** the pointer, could give us the value stored in that memory address

```
int32_t  x  = 0;
int32_t* px;
px = &x;

std::cout <<  *px << std::endl;
                // prints 0
```

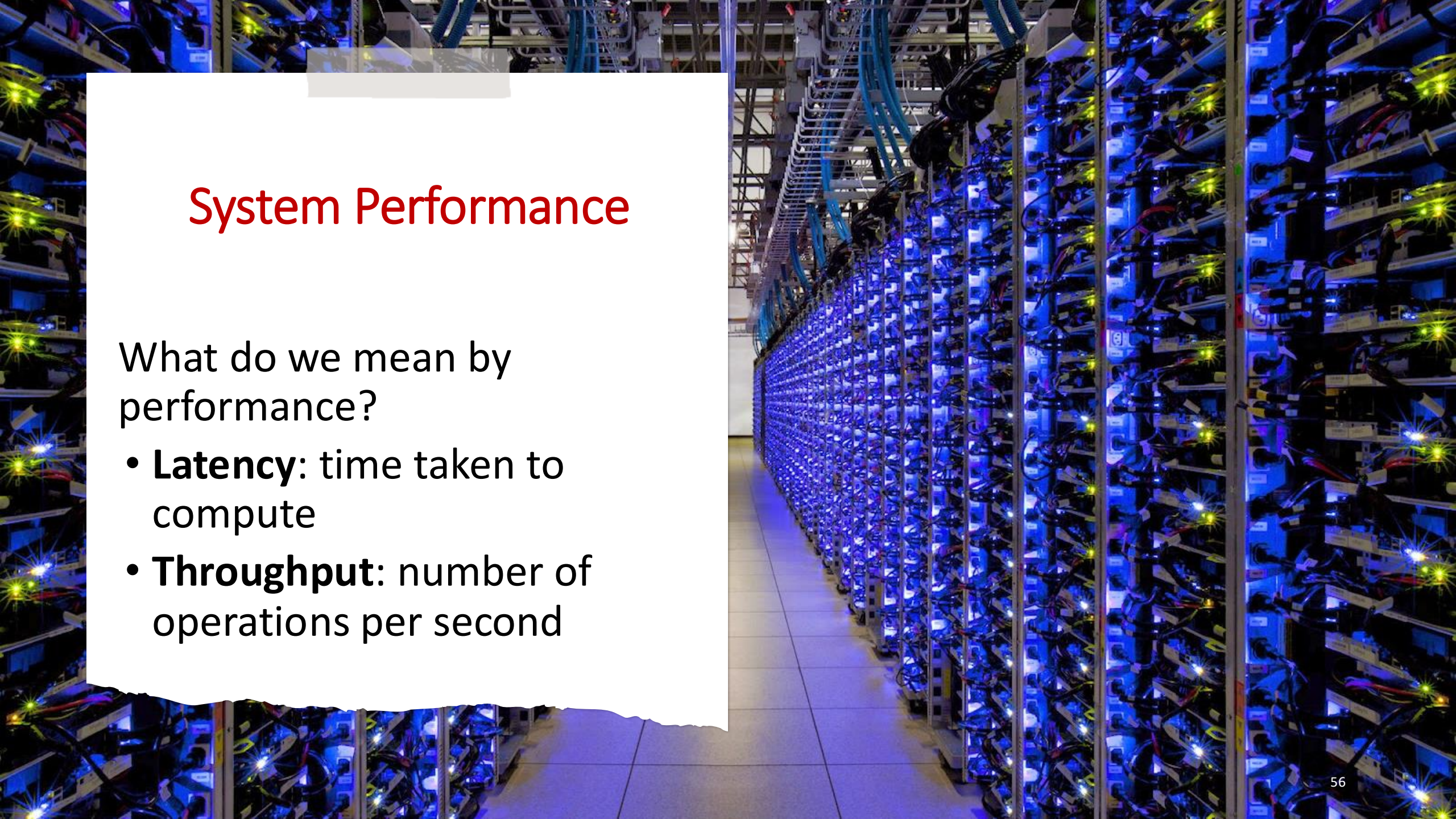


Systems Performance

Why C++?

C++ is an efficient and fast language

- Performance benefit
- Fine-grained memory management



System Performance

What do we mean by performance?

- **Latency**: time taken to compute
- **Throughput**: number of operations per second

Reasoning about system performance

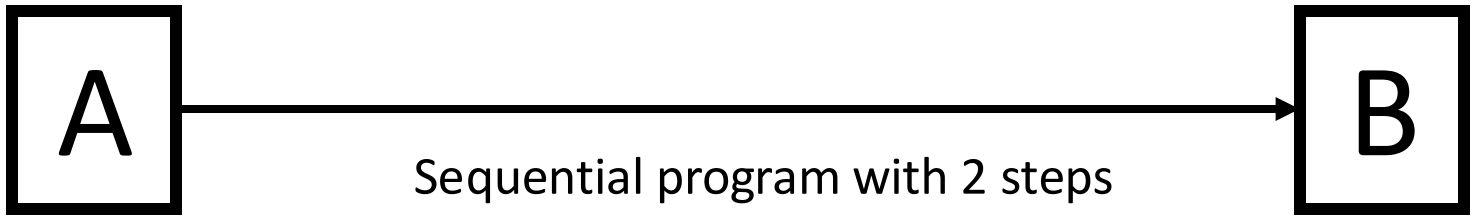
- Theoretical improvements don't always translate to better

  runtimes

  Insertion sort outperforms quick sort in some cases
  Why?
  1. Insertion sort is iterative – no overhead from recursive calls (good for sorting a small set)
  2. Insertion sort is fast when data is nearly sorted

Reasoning about system performance

- Theoretical improvements don't always translate to better runtimes
- Which algorithm? A system can be very complex with many features



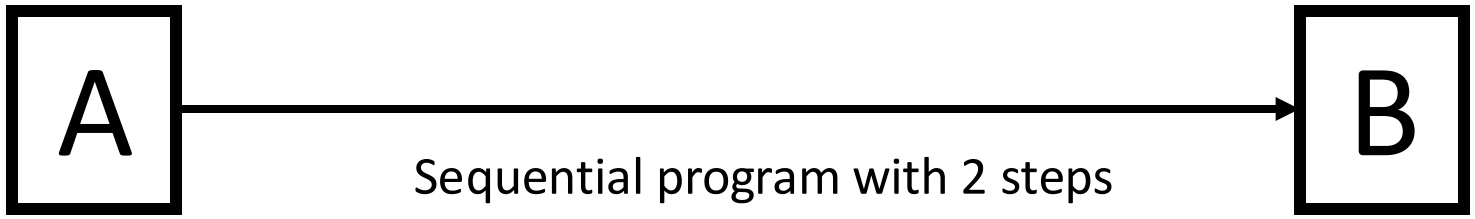
Fairly optimized code

Highly inefficient code

- A = processing files, B = printing 1 million lines of output

Reasoning about system performance

- Theoretical improvements don't always translate to better runtimes
- Which algorithm? A system can be very complex with many features



Fairly optimized code

Highly inefficient code

- What if step A takes about 99% of the total time? We need to profile and understand performance characteristics of code we write

Reasoning about system performance

- Theoretical improvements don't always translate to better runtimes
- Which algorithm? A system can be very complex with many features
- What if the code that implements the algorithm is inefficient?
- Sometimes heuristics work better

References

- A Tour of C++, Bjarne Stroustrup, 2nd edition
- Effective C++: 55 specific ways to improve your programs and designs, Scott Meyers, 3rd edition
- Large Scale C++, Process and Architecture, John Lakos, Volume 1
- GDB documentation: https://www.sourceware.org/gdb/
- https://www.geeksforgeeks.org/gdb-step-by-step-introduction/
- GDB quickstart tutorial: https://web.eecs.umich.edu/~sugih/pointers/gdbQS.html
- How does gbd work? https://www.aosabook.org/en/gdb.html