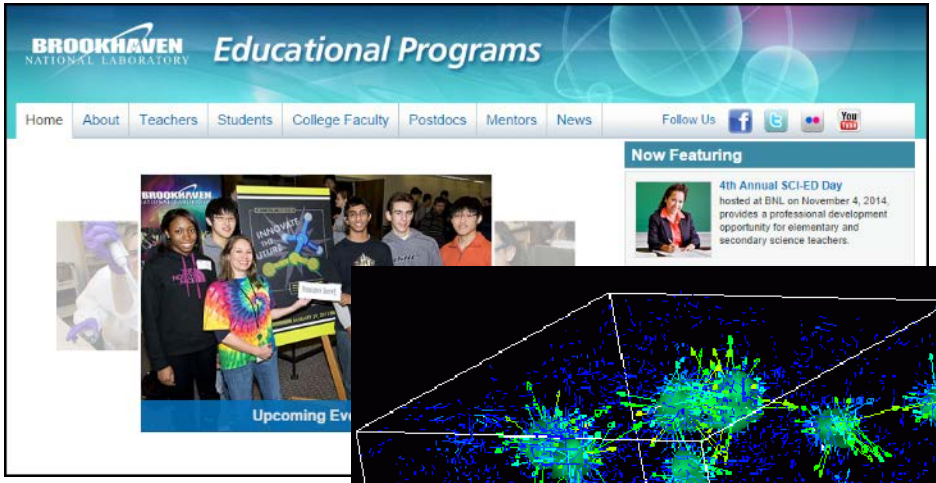
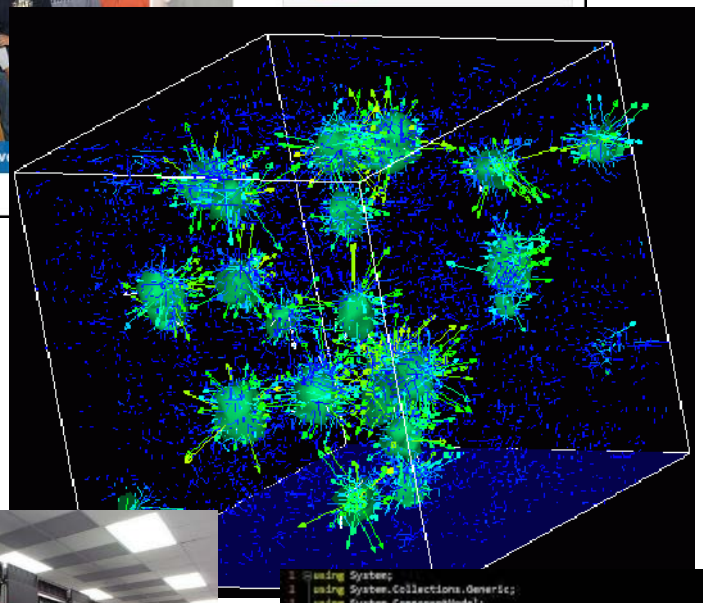
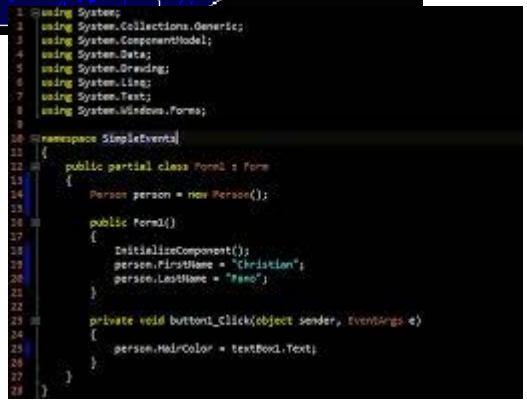






Survey of Scientific Computing (SciComp 301)

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

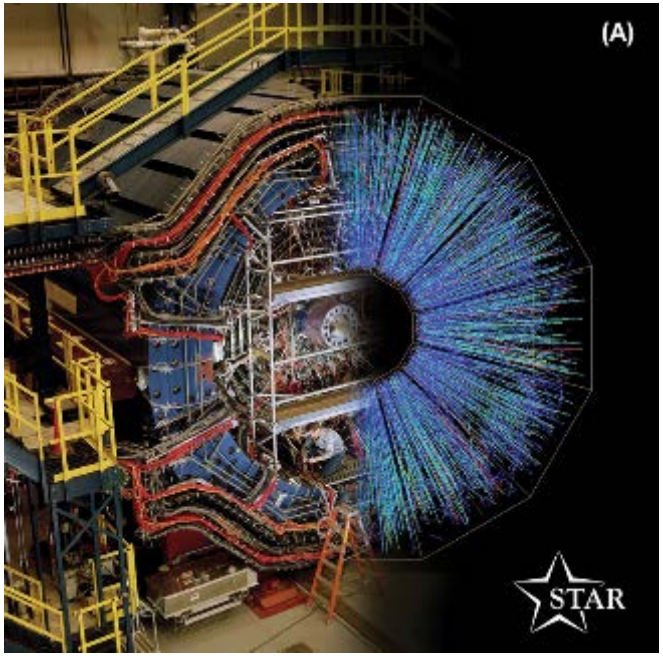
Session 02
Code Structure, Variables

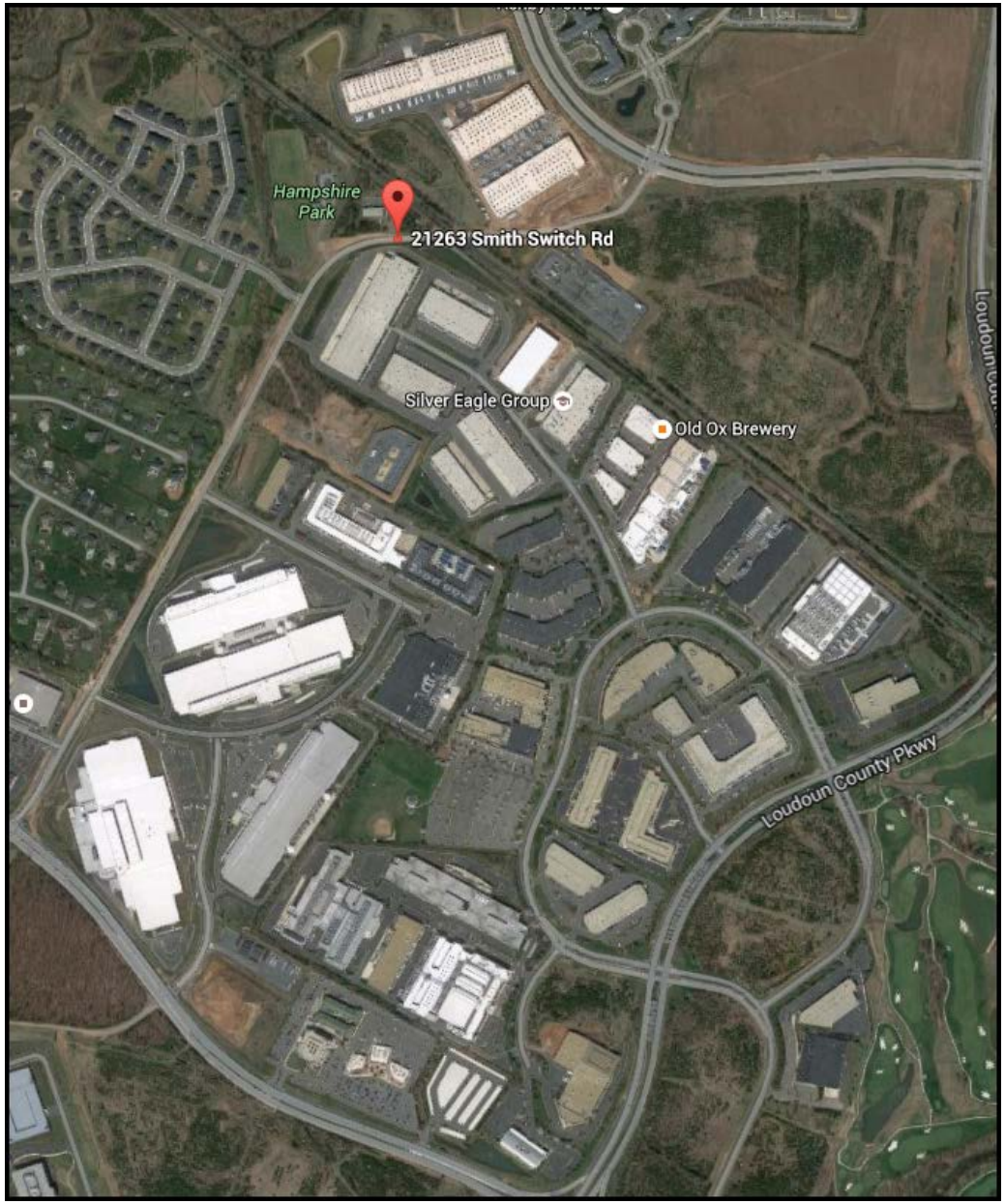
Session Goals

- Access **your** remote PC in Amazon's cloud
- Review the rules for creating C++ **identifiers**
- Understand C++ **statements** and **scopes**
- Create C++ variables using built-in data **types**
- Display the values of variables on the console
- Create simple loops with the **for()** statement

Scientific Computing in the Cloud

- The cloud is a great **equalizer**
- Allows participants to continue learning at home as on campus
- Cloud machines are fully isolated from school networks - eliminates local software installation restrictions & campus cybersecurity concerns
- Schools with low-end PC labs can still access thenewest generation of hardware



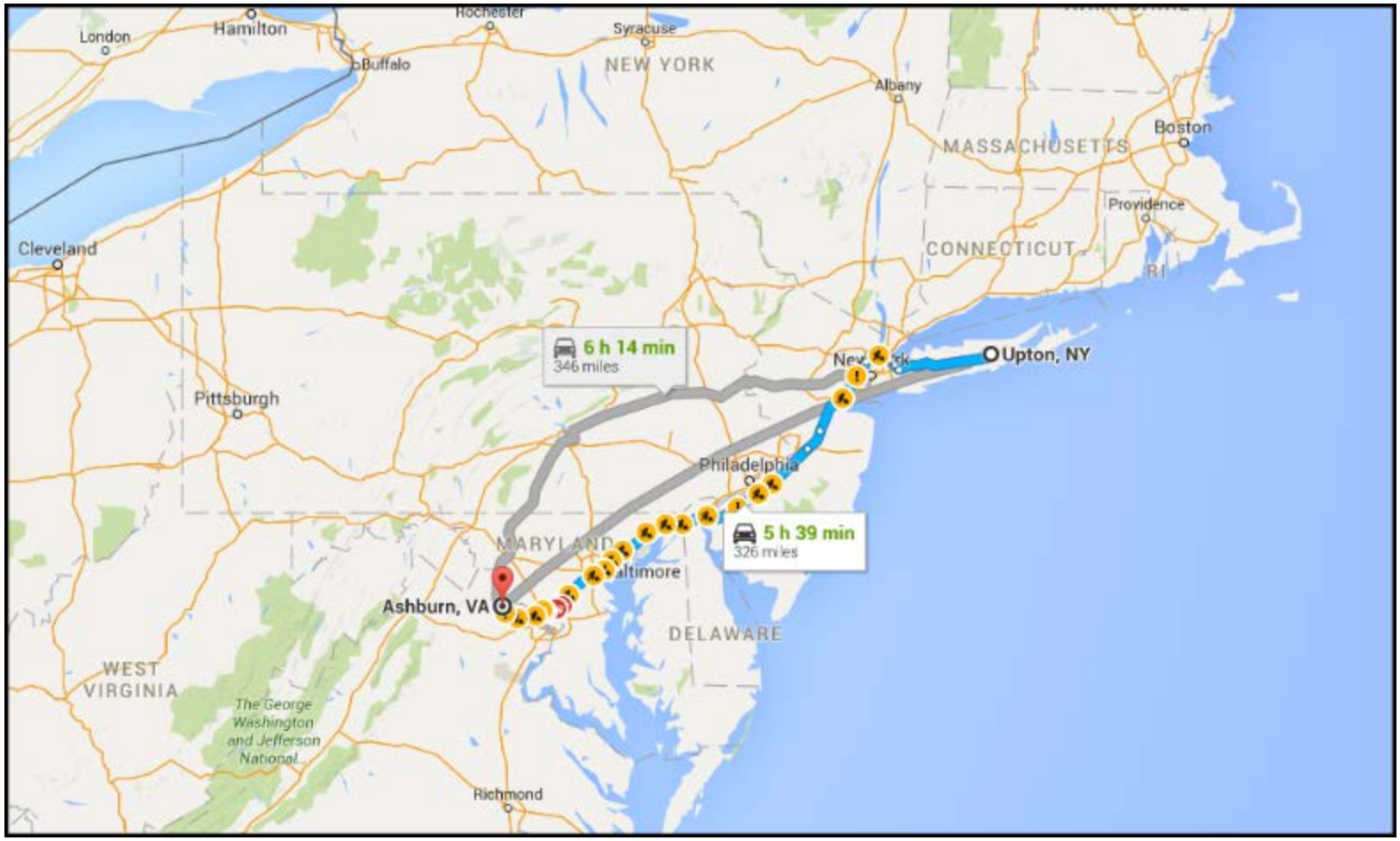


Amazon's
23 data centers in
Ashburn, Virginia

Just two of Amazon's 23 cloud data centers in Ashburn, Virginia

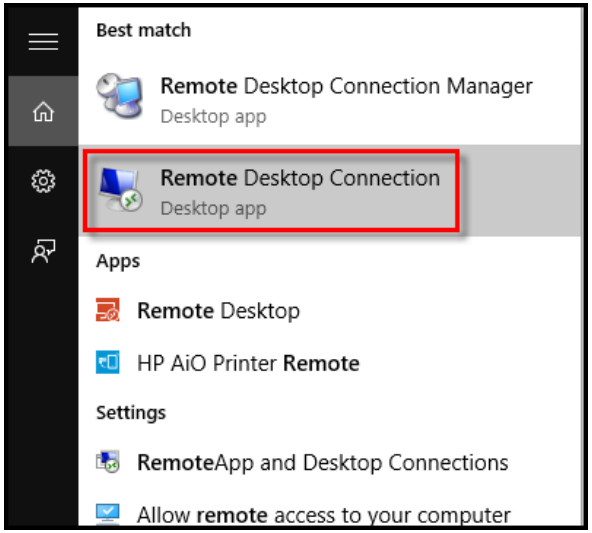
BNL ↔ Ashburn

Roundtrip = 660 miles



Accessing an Amazon Remote PC

- Uses Microsoft's "**Remote Desktop Connection**" application
- Students have assigned credentials
  - The IP address ***(computer name)*** of <u>their</u> specific machine in Amazon's cloud
  - Their user name (case sensitive!)
  - Their password (case sensitive!)
- Students can access their Amazon machine **from their home**
- Educators can remotely monitor student progress **real-time** on the lab exercises

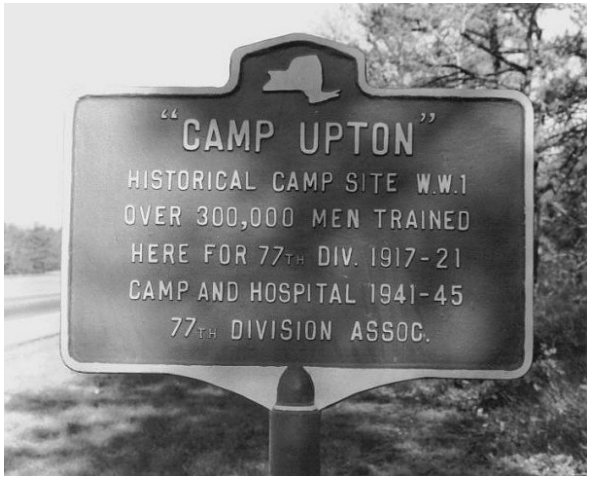


Accessing an Amazon Remote PC

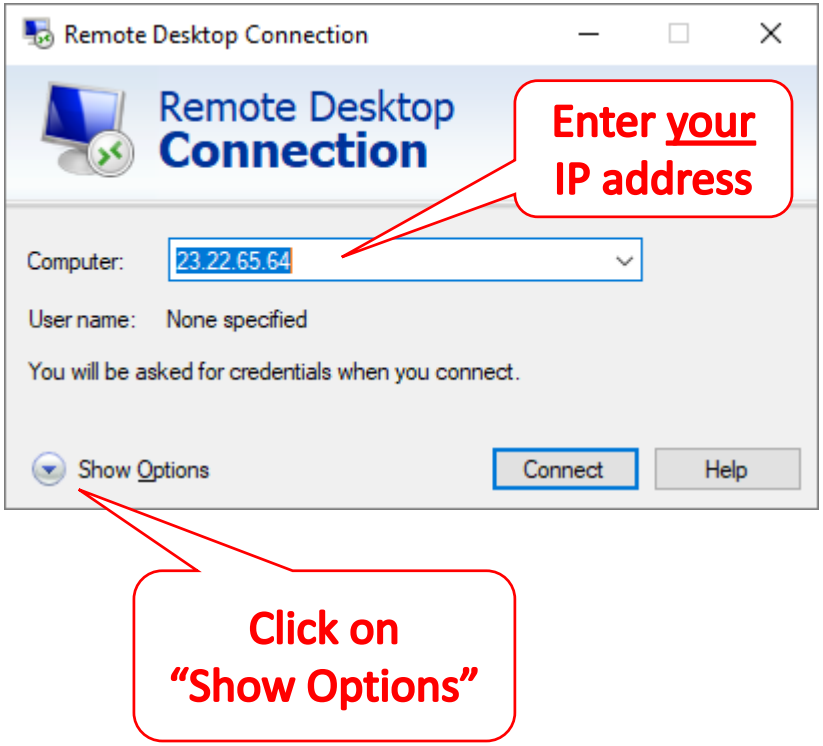
IP Address	Last Name	First Name
23.21.36.204		
23.22.65.64		
23.22.66.177		
23.22.77.143		
23.22.79.115		
23.23.69.92		
50.16.130.78		
50.16.135.127		
50.16.144.96		
50.16.147.112		
50.16.159.220		
50.16.160.201		
50.16.163.25		
50.16.163.117		
50.16.163.174		
50.16.165.70		
50.16.165.235		
50.16.170.168		
50.16.175.58		
50.16.175.233		

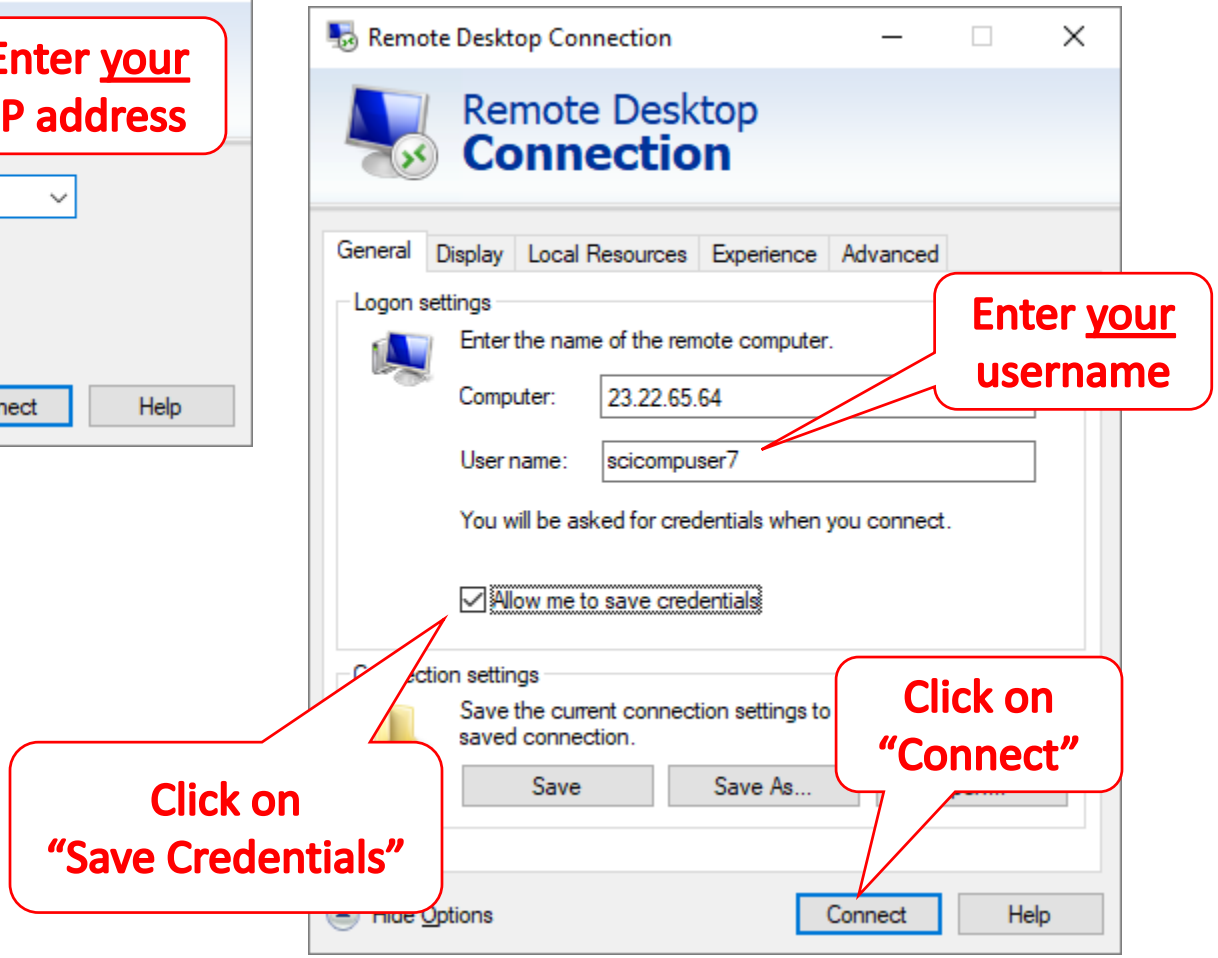
1. Write down your IP address, username, and password in your notebook

2. Take a picture of your login information to keep on your smart phone

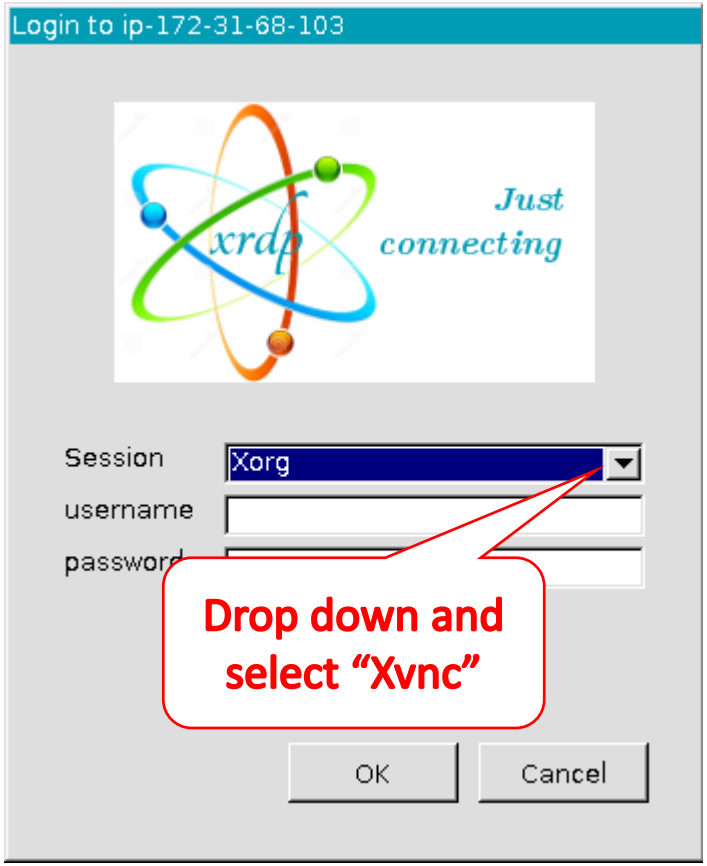


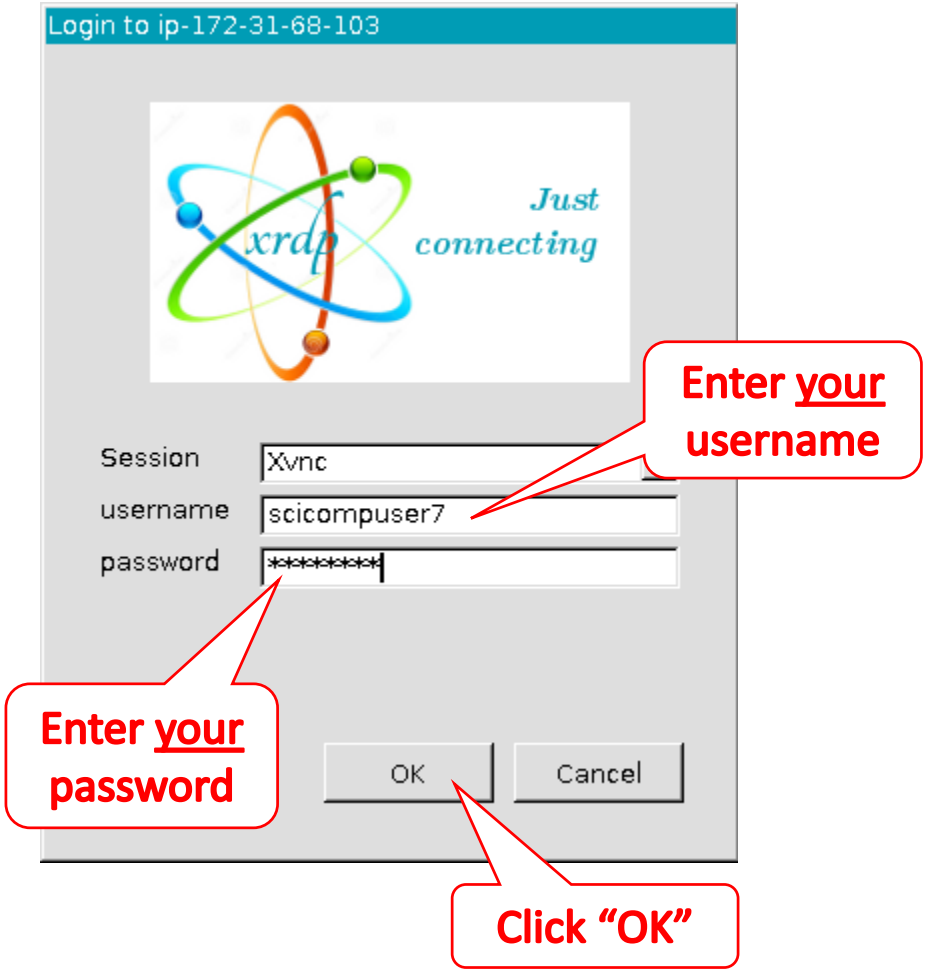
Accessing an Amazon Remote PC



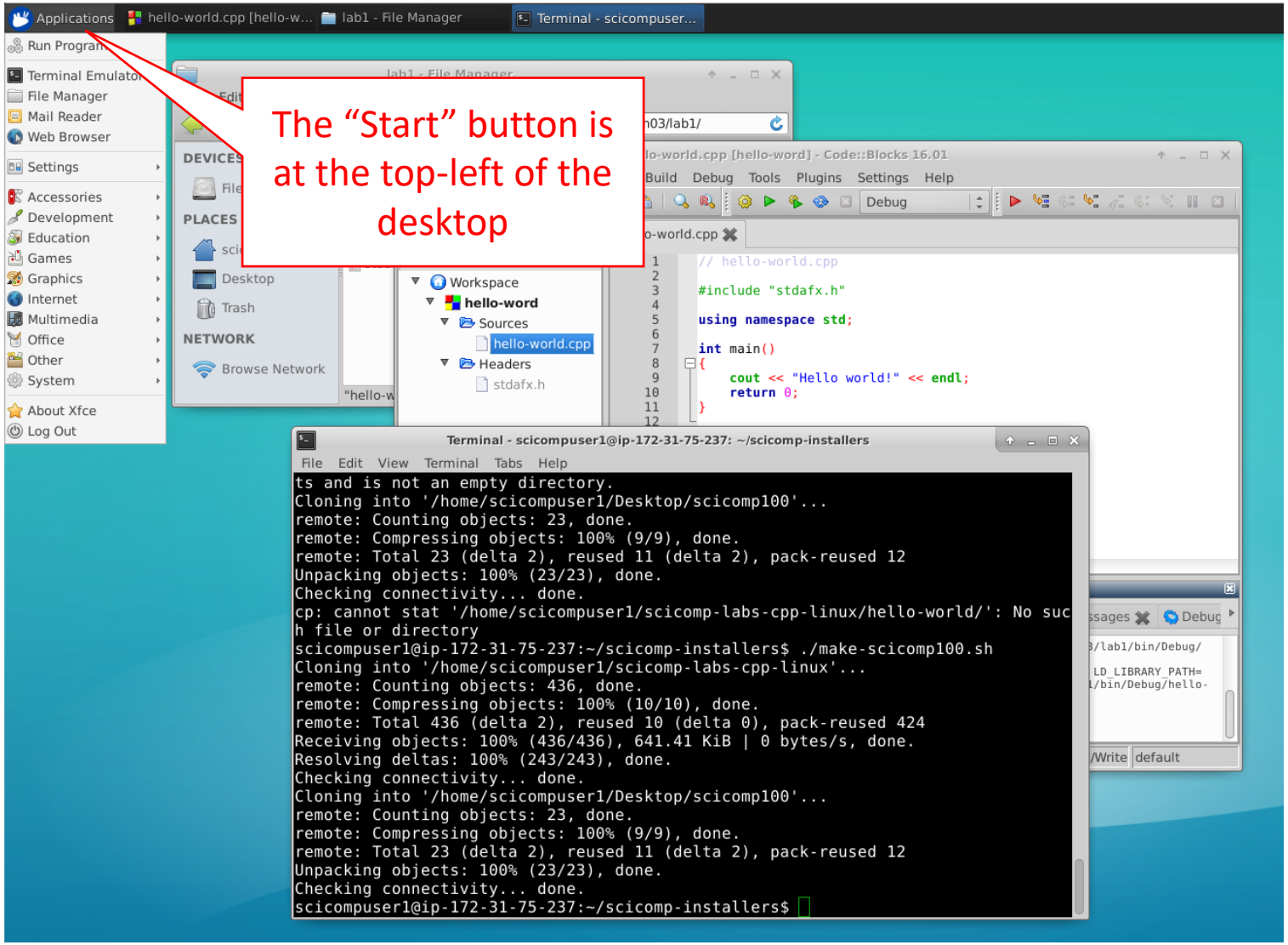


Accessing an Amazon Remote PC

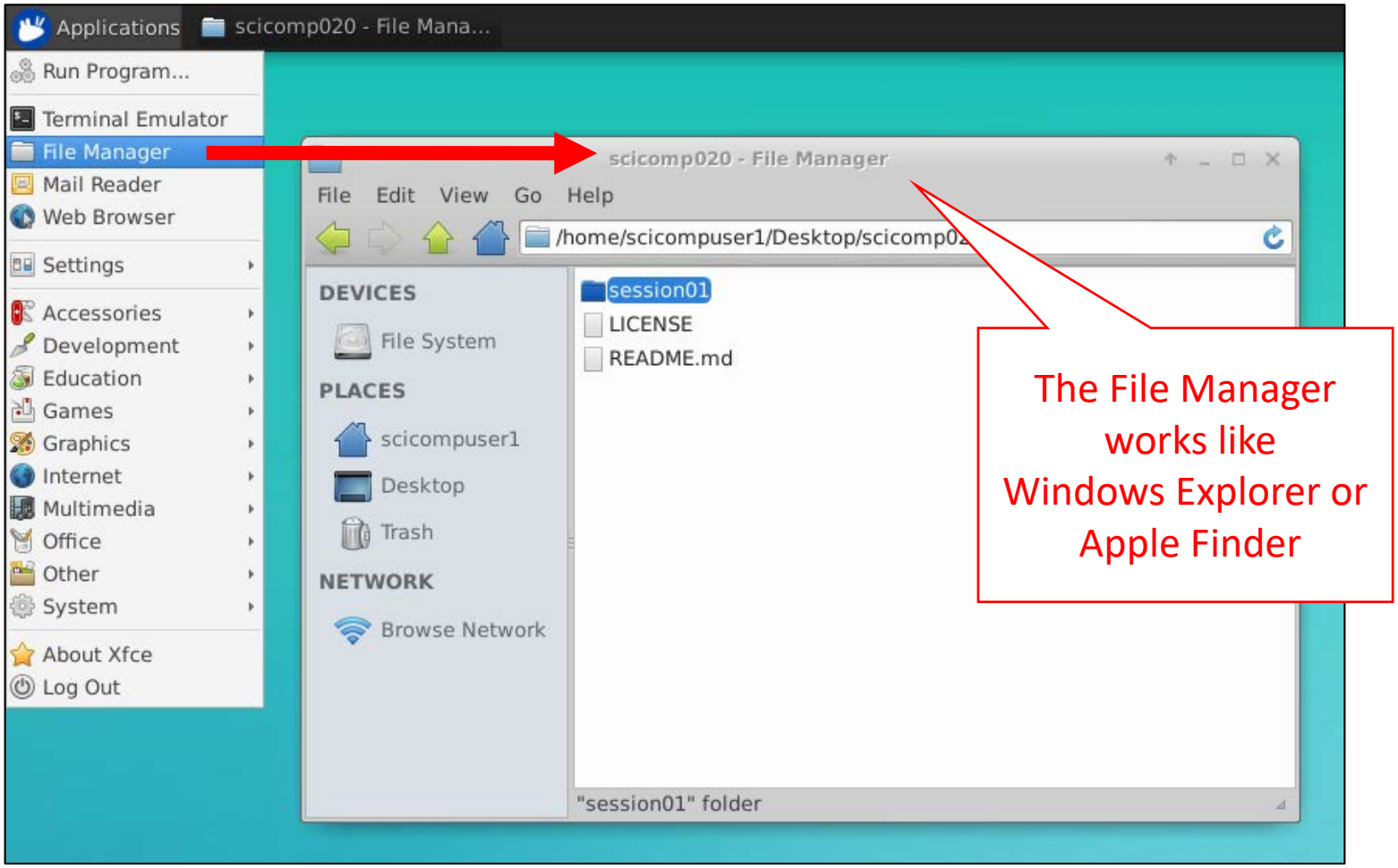




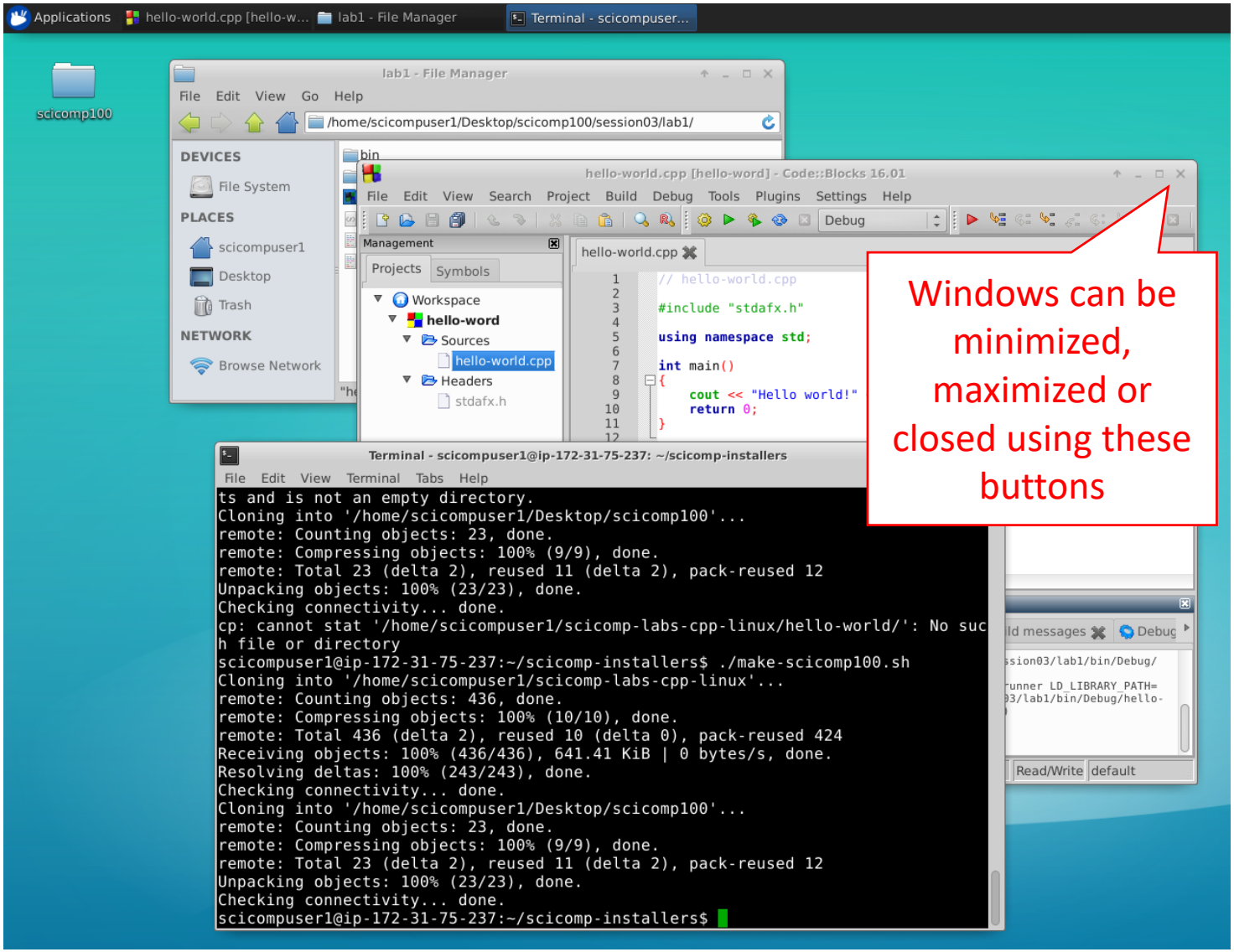
Ubuntu Linux with Xfce Desktop



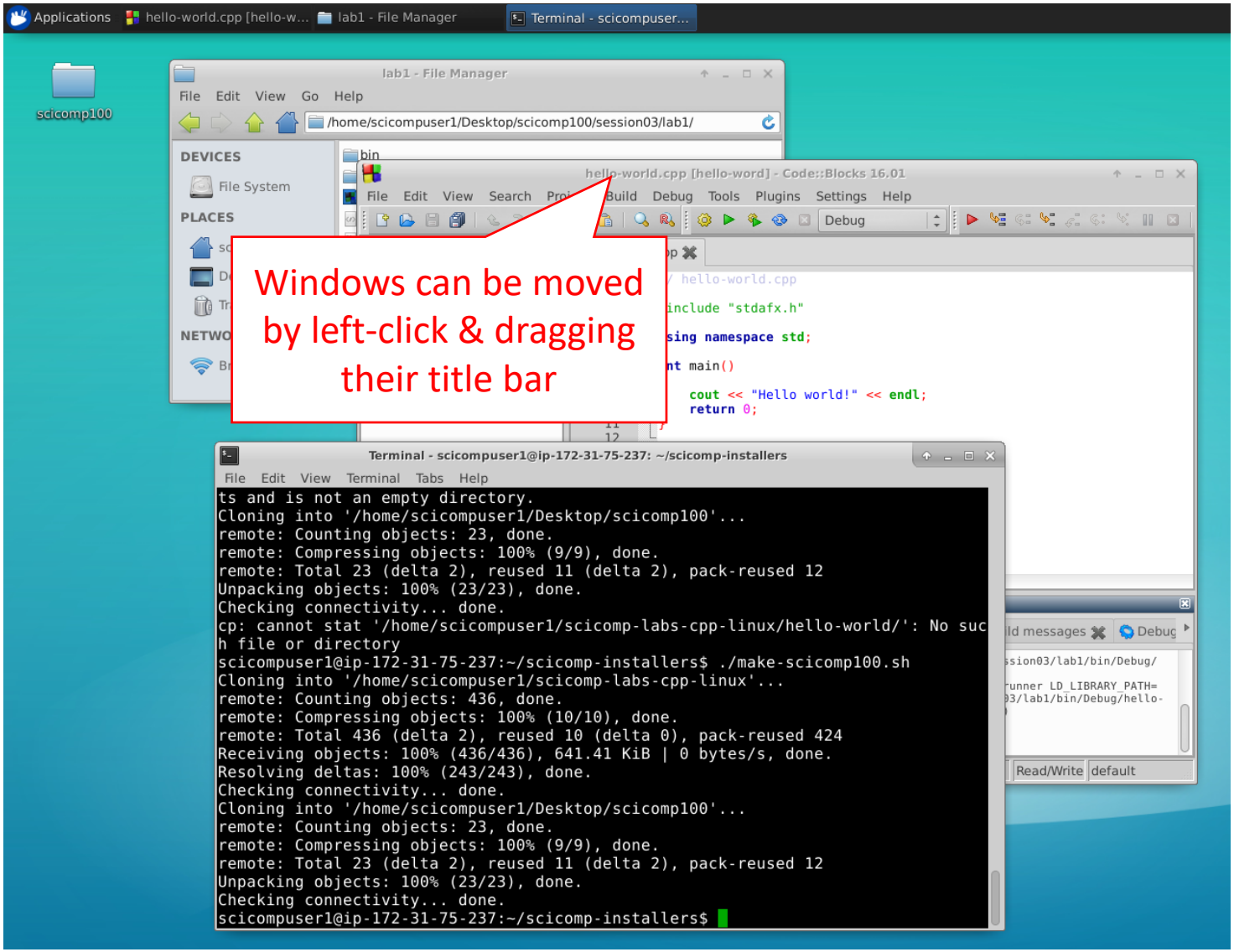
Ubuntu Linux with Xfce Desktop



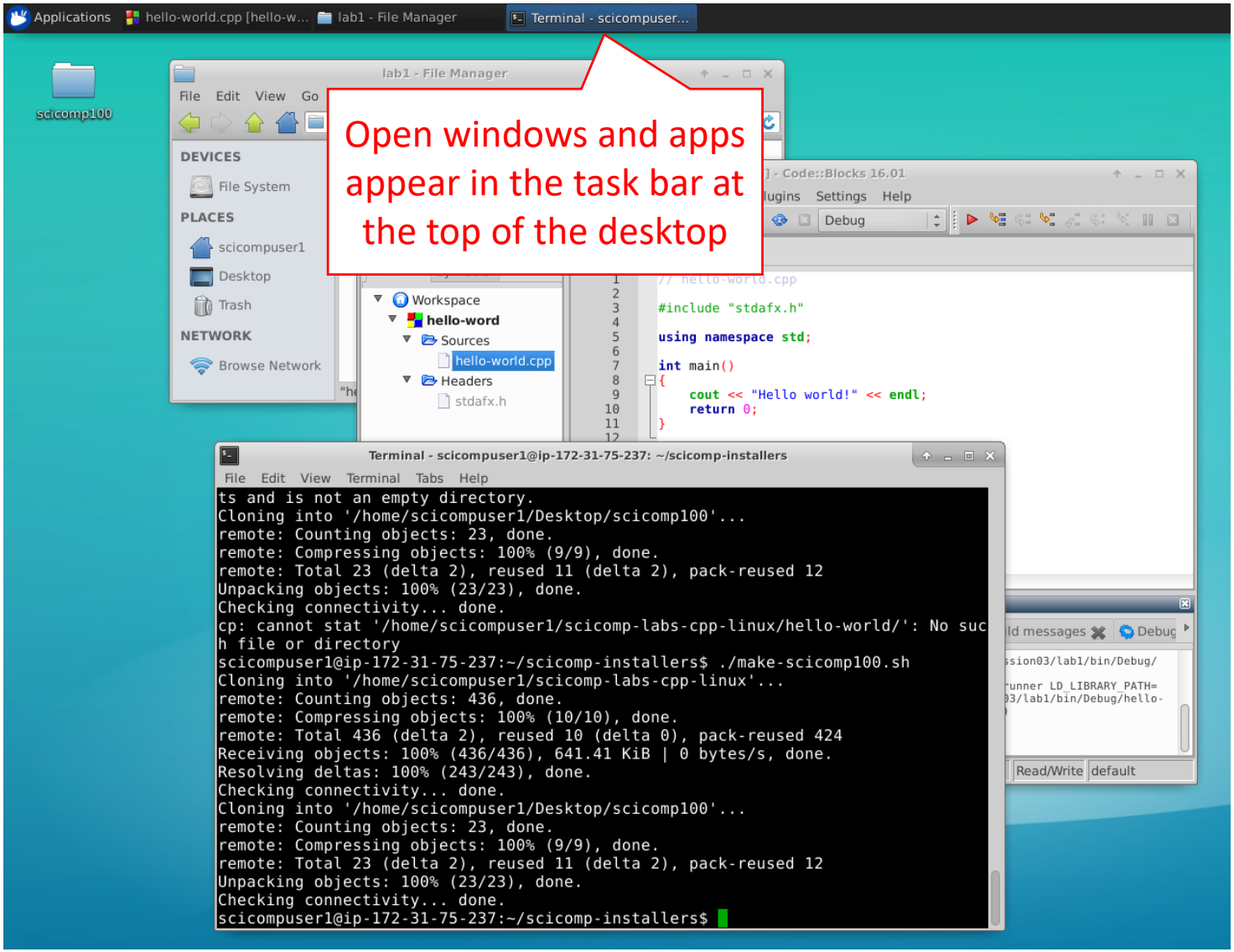
Ubuntu Linux with Xfce Desktop



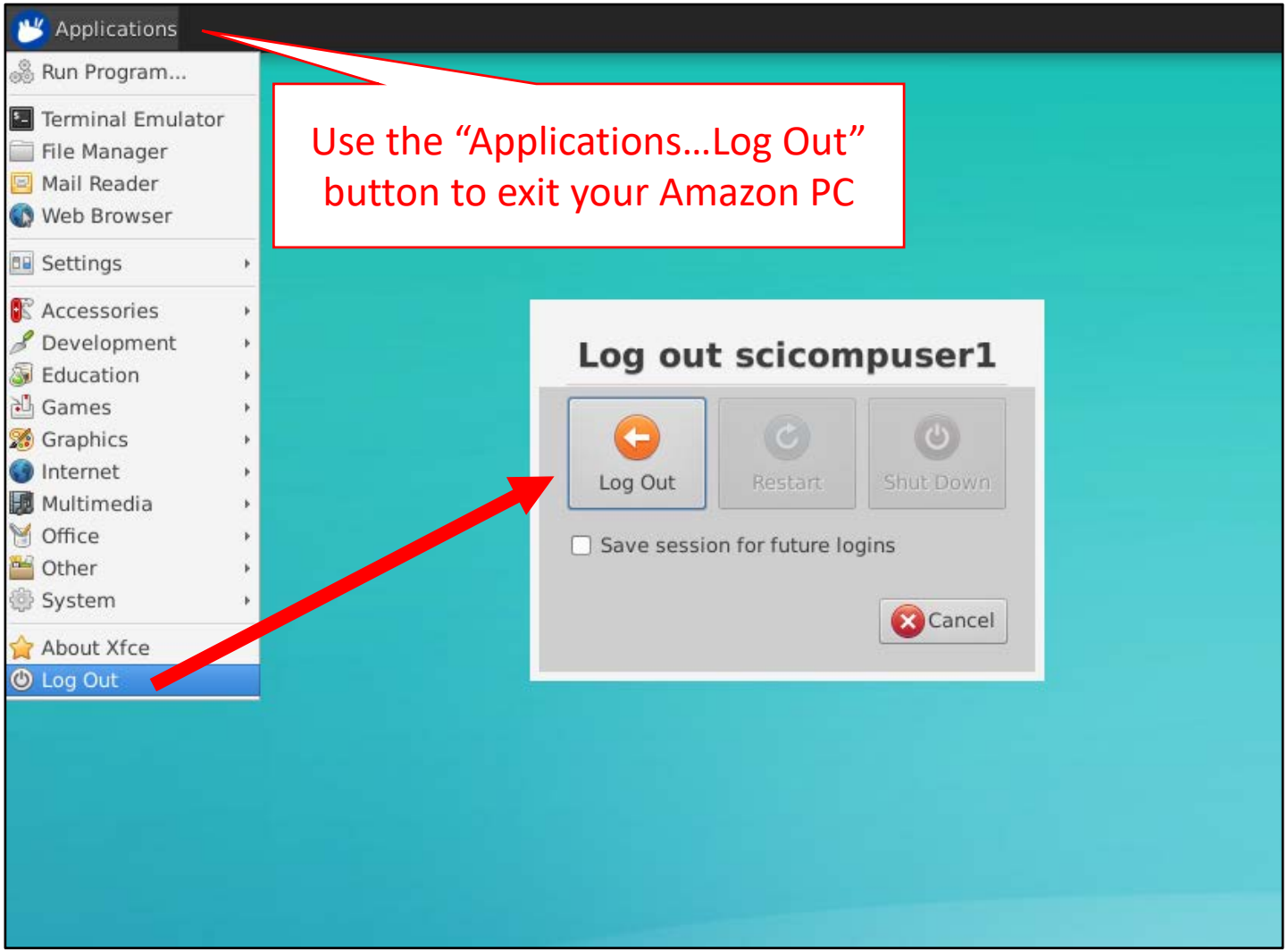
Ubuntu Linux with Xfce Desktop



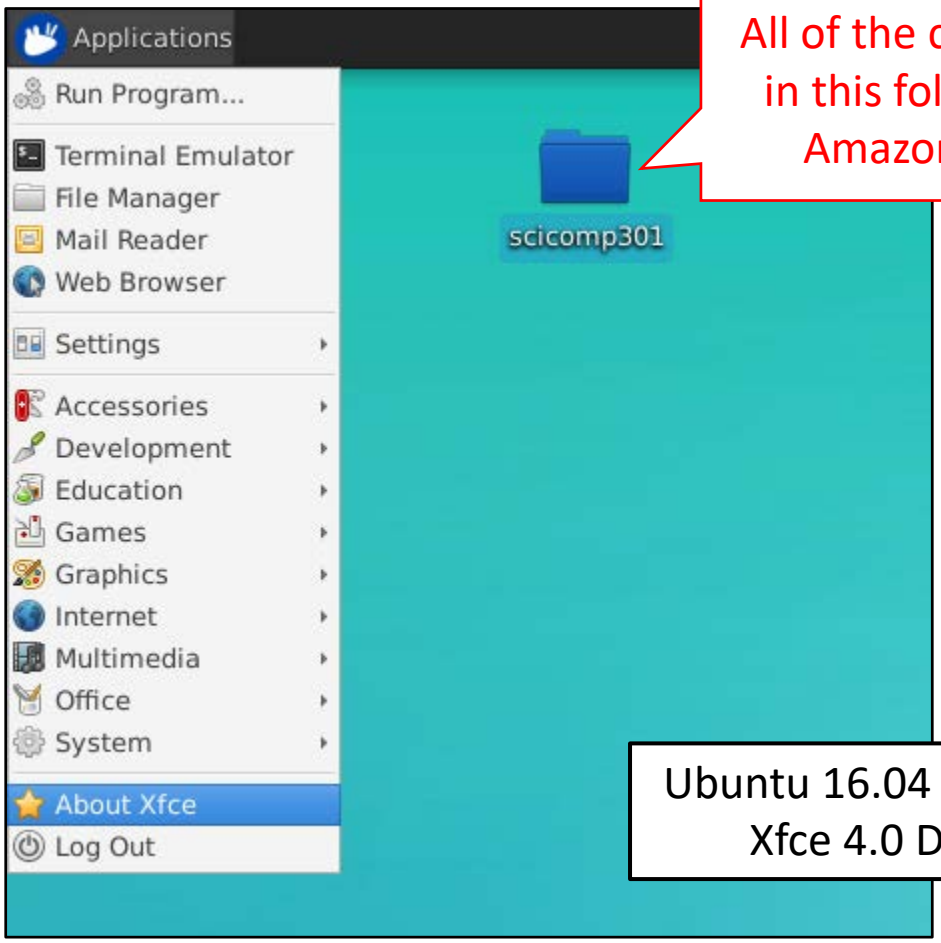
Ubuntu Linux with Xfce Desktop



Ubuntu Linux with Xfce Desktop



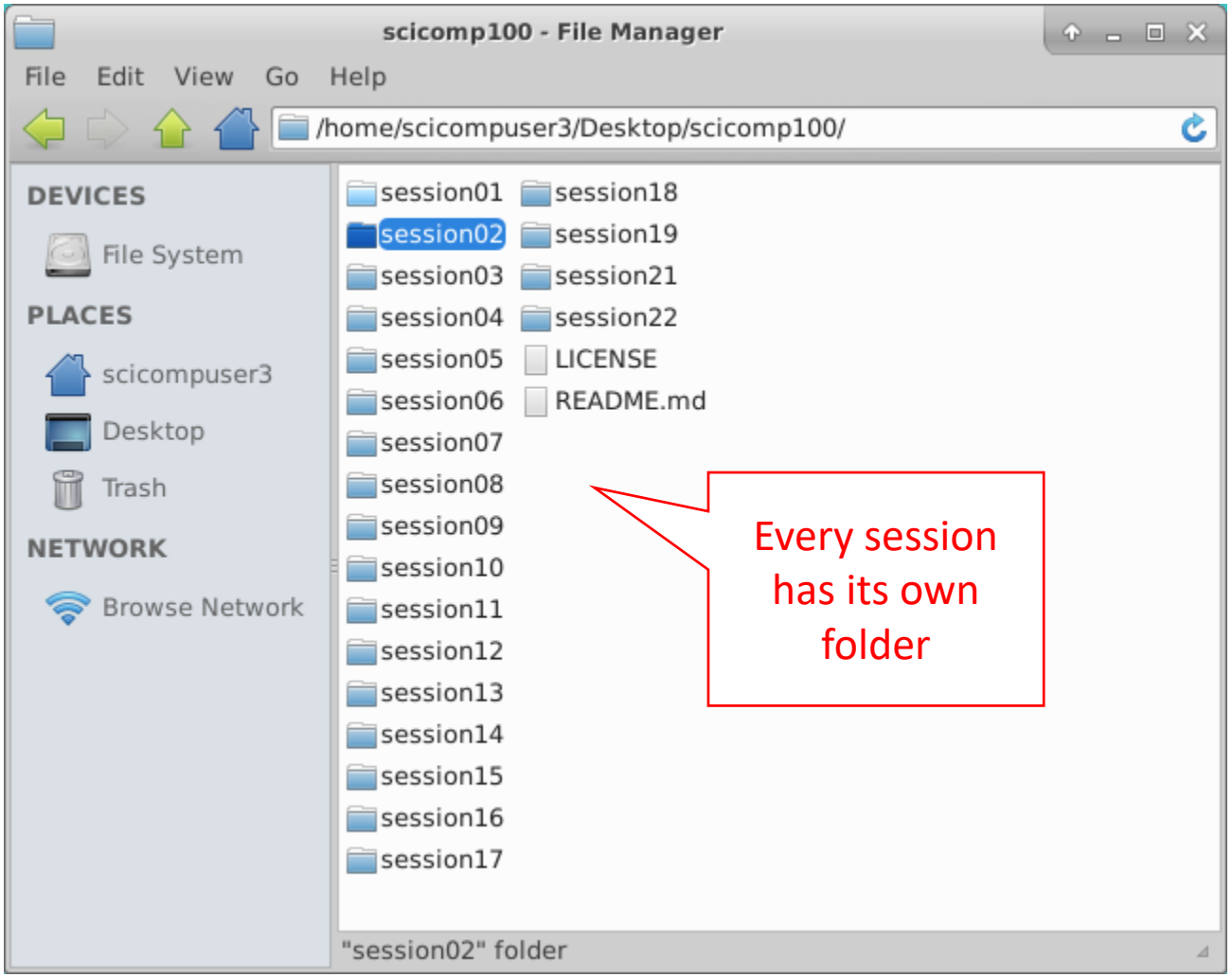
SciComp Courseware Layout



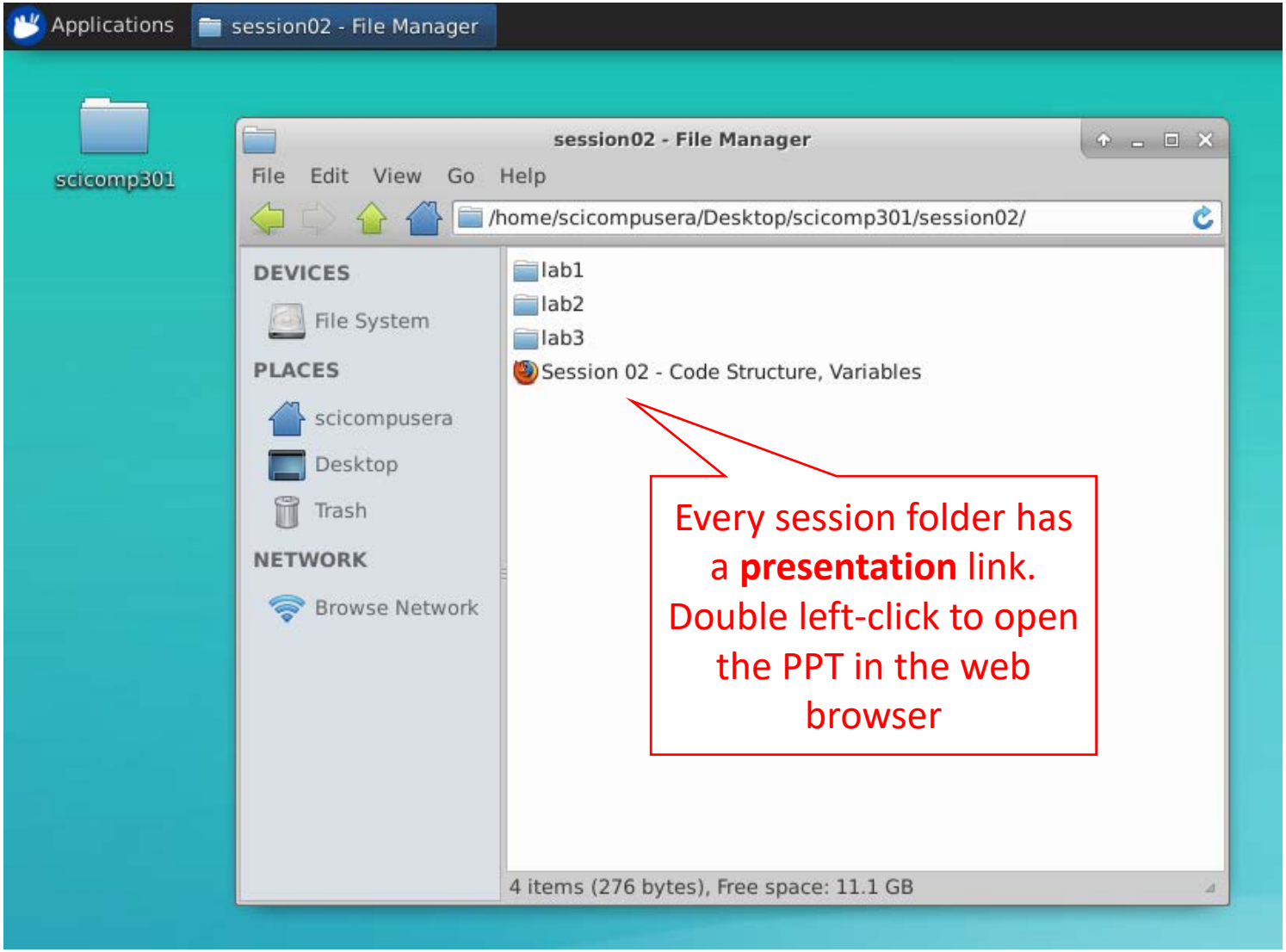
All of the courseware is in this folder on your Amazon Desktop

Ubuntu 16.04 LTS (64-Bit)
Xfce 4.0 Desktop

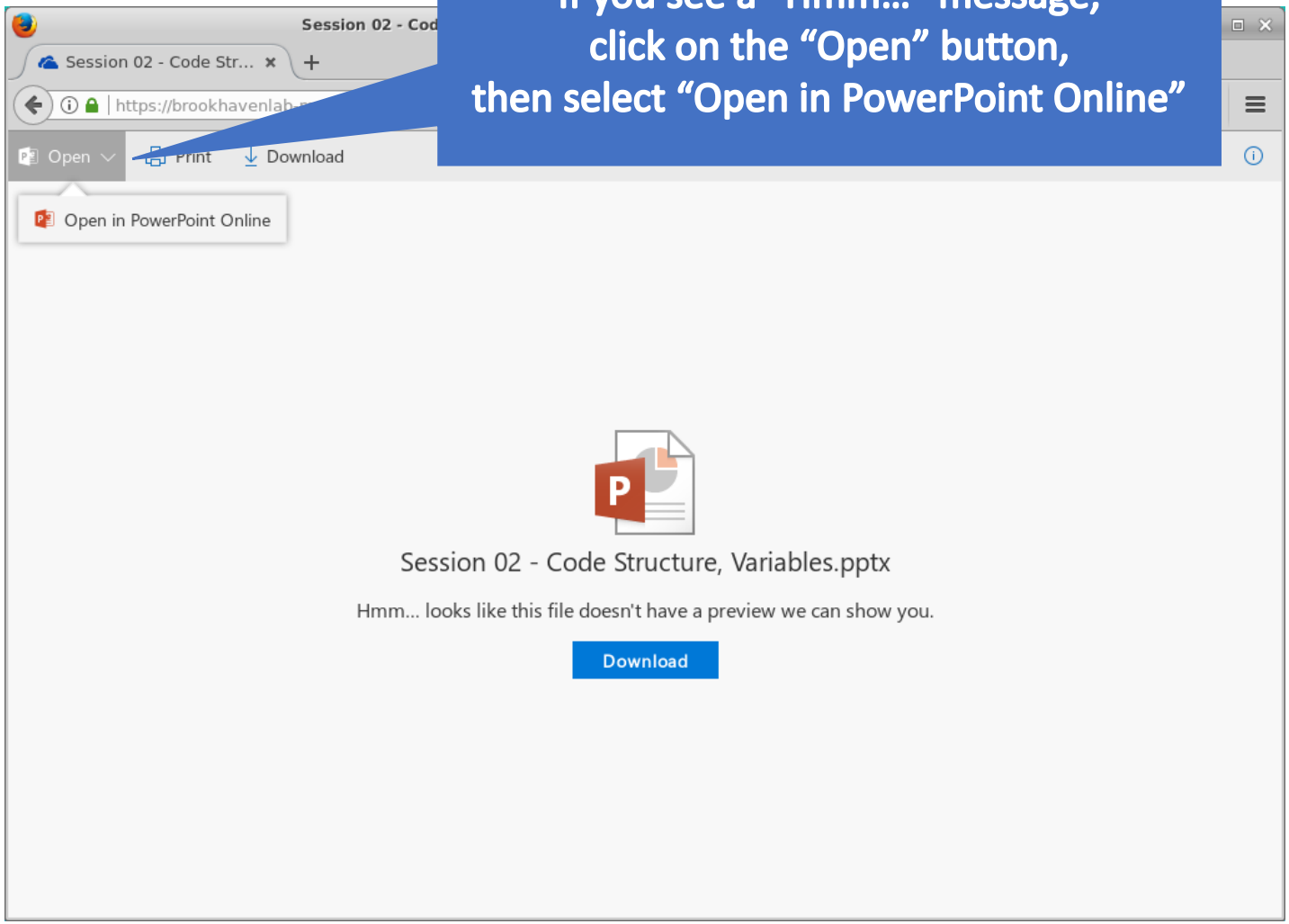
SciComp Courseware Layout



SciComp Courseware Layout



If you see a “Hmm...” message,
click on the “Open” button,
then select “Open in PowerPoint Online”



Session 02 - Code Structure, Variables.pptx - Mozilla Firefox

Session 02 - Code Str... | Session 02 - Code Str...

https://brookhavenlab-my.sharepoint.com/:p:/r/personal/dbiersach_bn

Search

PowerPoint Online

Guest Contributor

Download | Start Slide Show | Print to PDF | Comments | •••

Survey of Scientific Computing (SciComp 301)

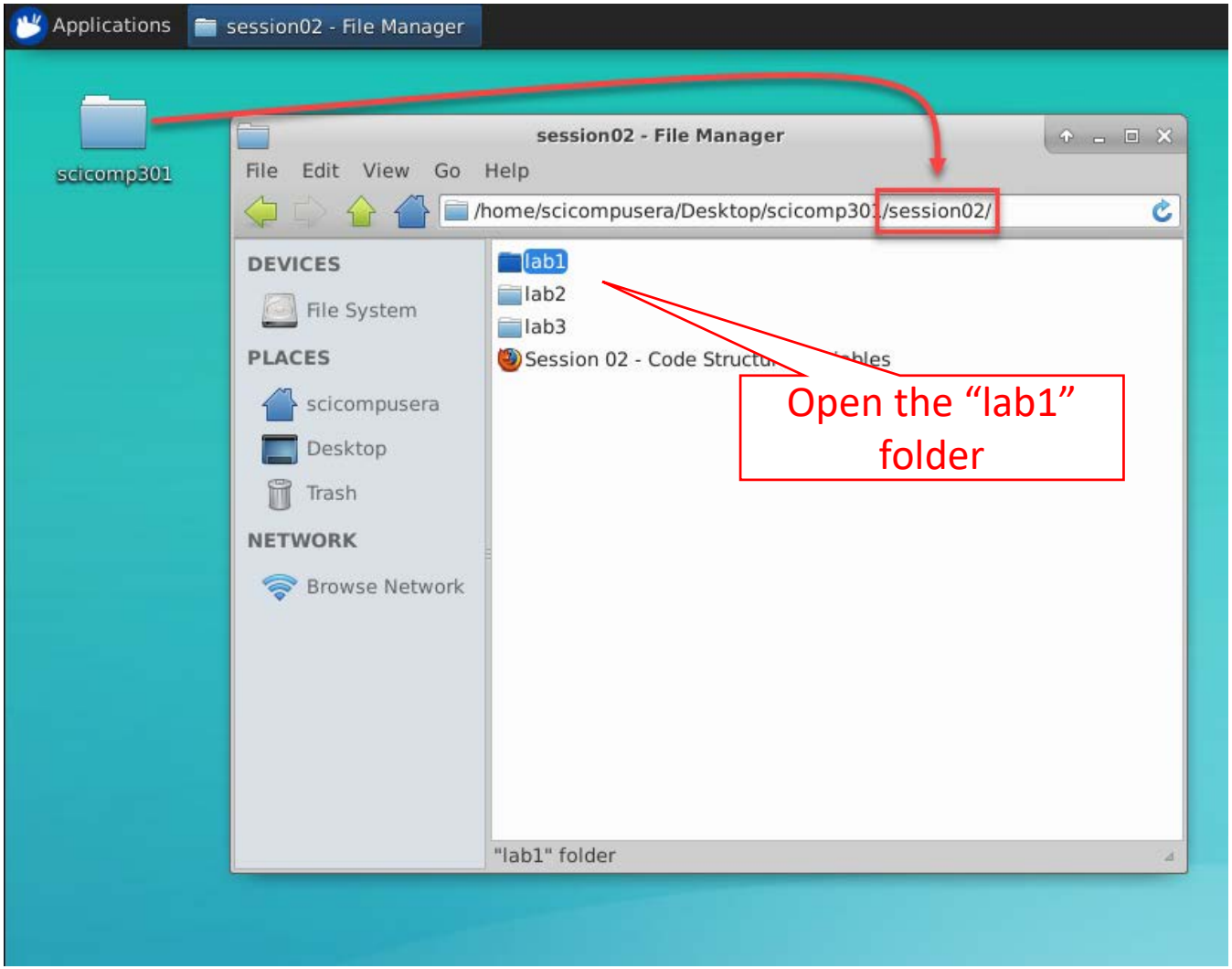
Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

Session 02
Code Structure, Variables

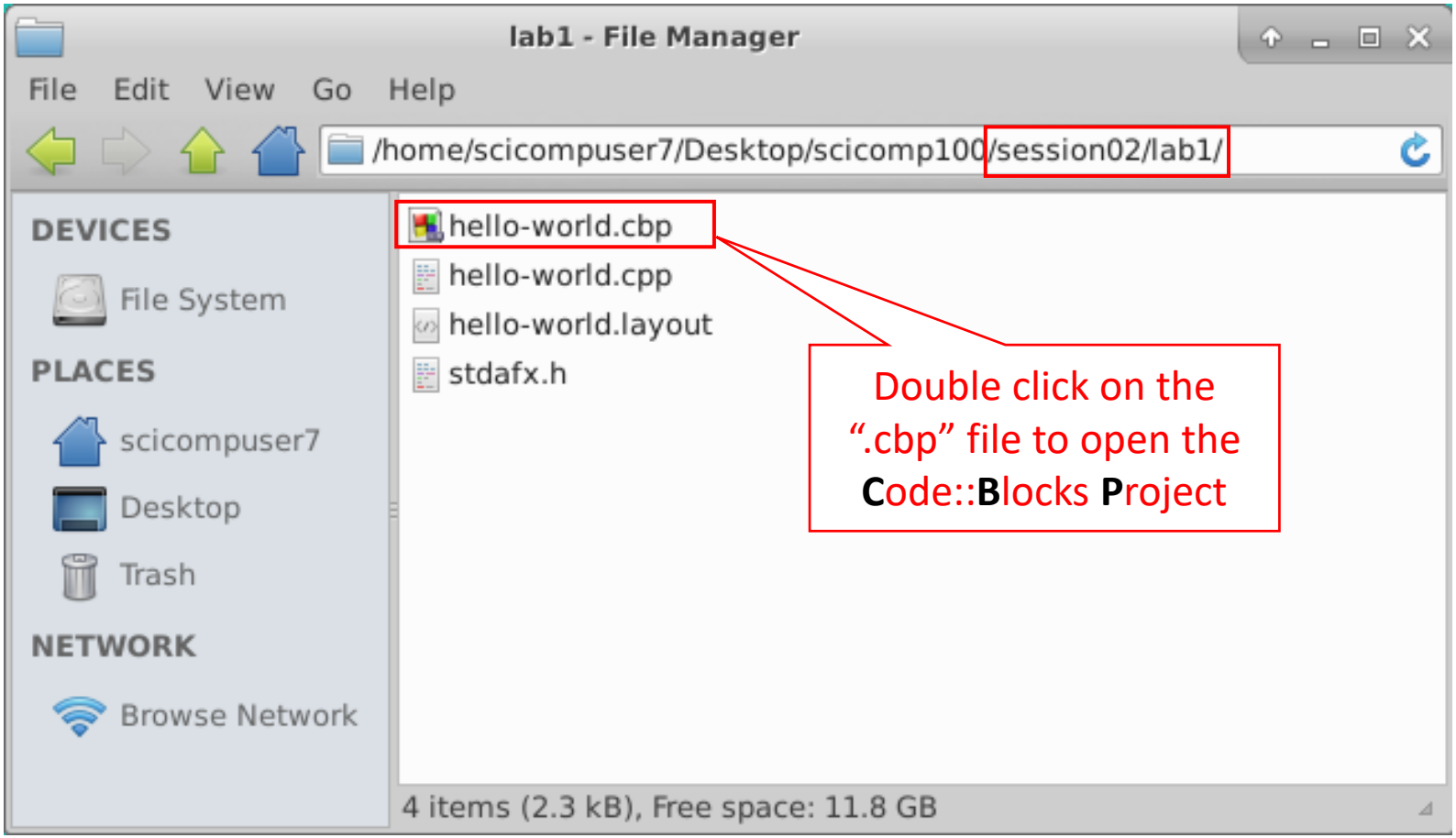
Left click on "Slide # of #" to jump to a specific slide

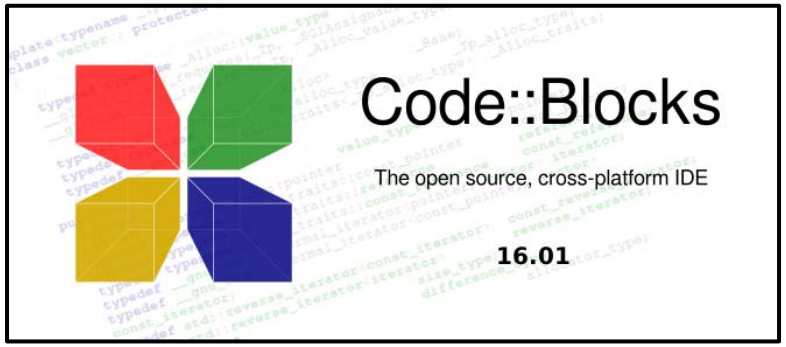
SLIDE 1 OF 48 | HELP IMPROVE OFFICE | NOTES

Lab 1 – Hello World!



Lab 1 – Hello World!





http://www.codeblocks.org

Highlights:

- **Open Source!** GPLv3, no hidden costs.
- **Cross-platform**. Runs on Linux, Mac, Windows (uses wxWidgets).
- Written in C++. No interpreted languages or proprietary libs needed.
- Extensible through plugins

Compiler:

- **Multiple compiler support**:
  - GCC (MingW / GNU GCC)
  - MSVC++
  - clang
  - Digital Mars
  - Borland C++ 5.5
  - Open Watcom
  - ...and more
- **Very fast** custom build system (no makefiles needed)
- Support for **parallel builds** (utilizing your CPU's extra cores)
- Multi-target projects
- Workspaces to combine multiple projects
- Inter-project dependencies inside workspace
- Imports MSVC projects and workspaces (NOTE: assembly code not supported yet)
- Imports Dev-C++ projects

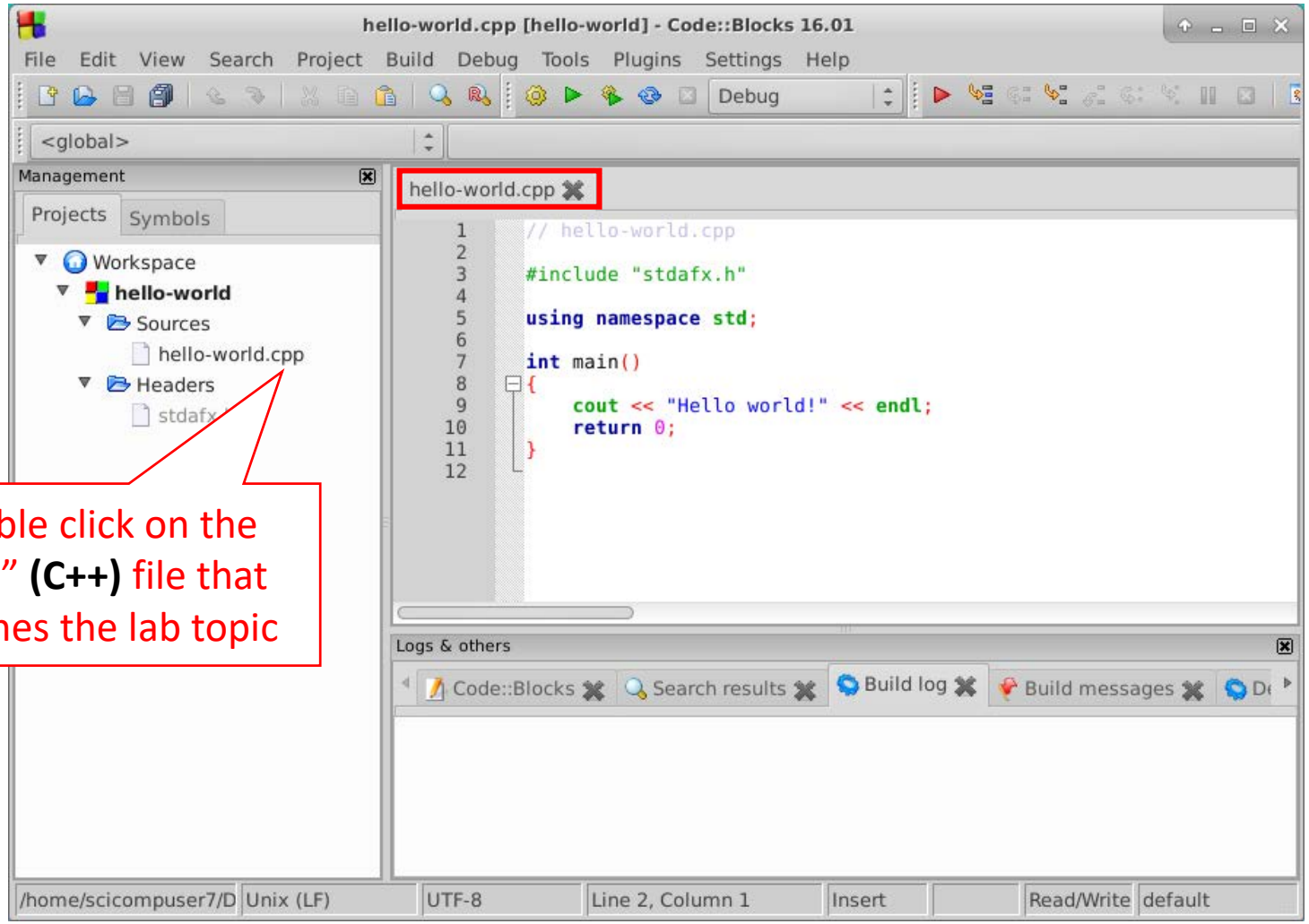
Debugger:

- Interfaces GNU GDB
- Also supports MS CDB (not fully featured)
- **Full breakpoints support**:
  - Code breakpoints
  - Data breakpoints (read, write and read/write)
  - Breakpoint conditions (break only when an expression is true)
  - Breakpoint ignore counts (break only after certain number of hits)
- Display local function symbols and arguments
- User-defined watches (support for watching user-defined types through scripting)
- Call stack
- Disassembly
- Custom memory dump
- Switch between threads
- View CPU registers

Interface:

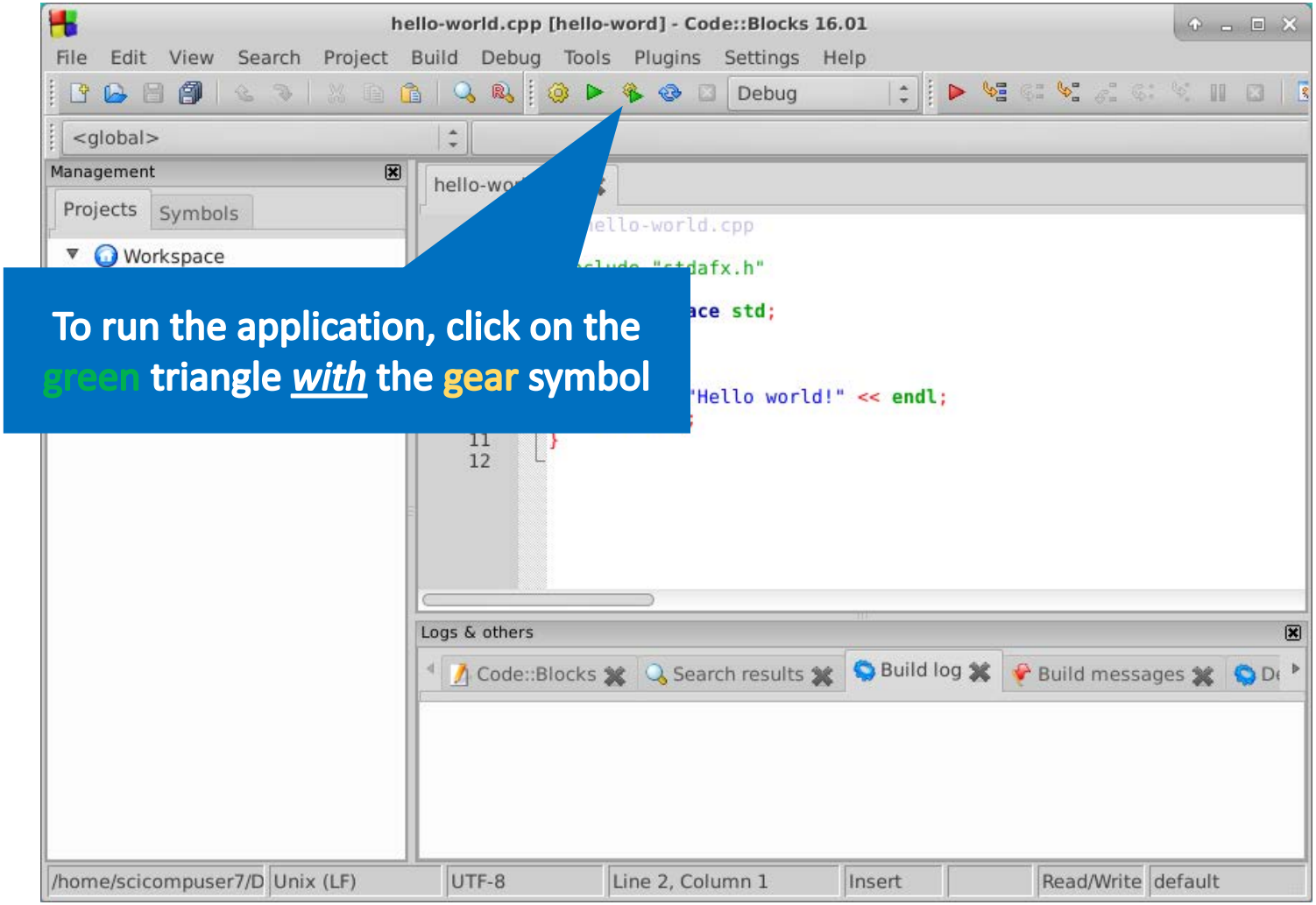
- Syntax highlighting, customizable and extensible
- Code folding for C, C++, Fortran, XML and many more files.
- Tabbed interface
- Code completion
- Class Browser
- Smart indent
- One-key swap between .h and .c/.cpp files
- Open files list for quick switching between files (optional)
- External customizable "Tools"
- To-do list management with different users

Lab 1 – Hello World!



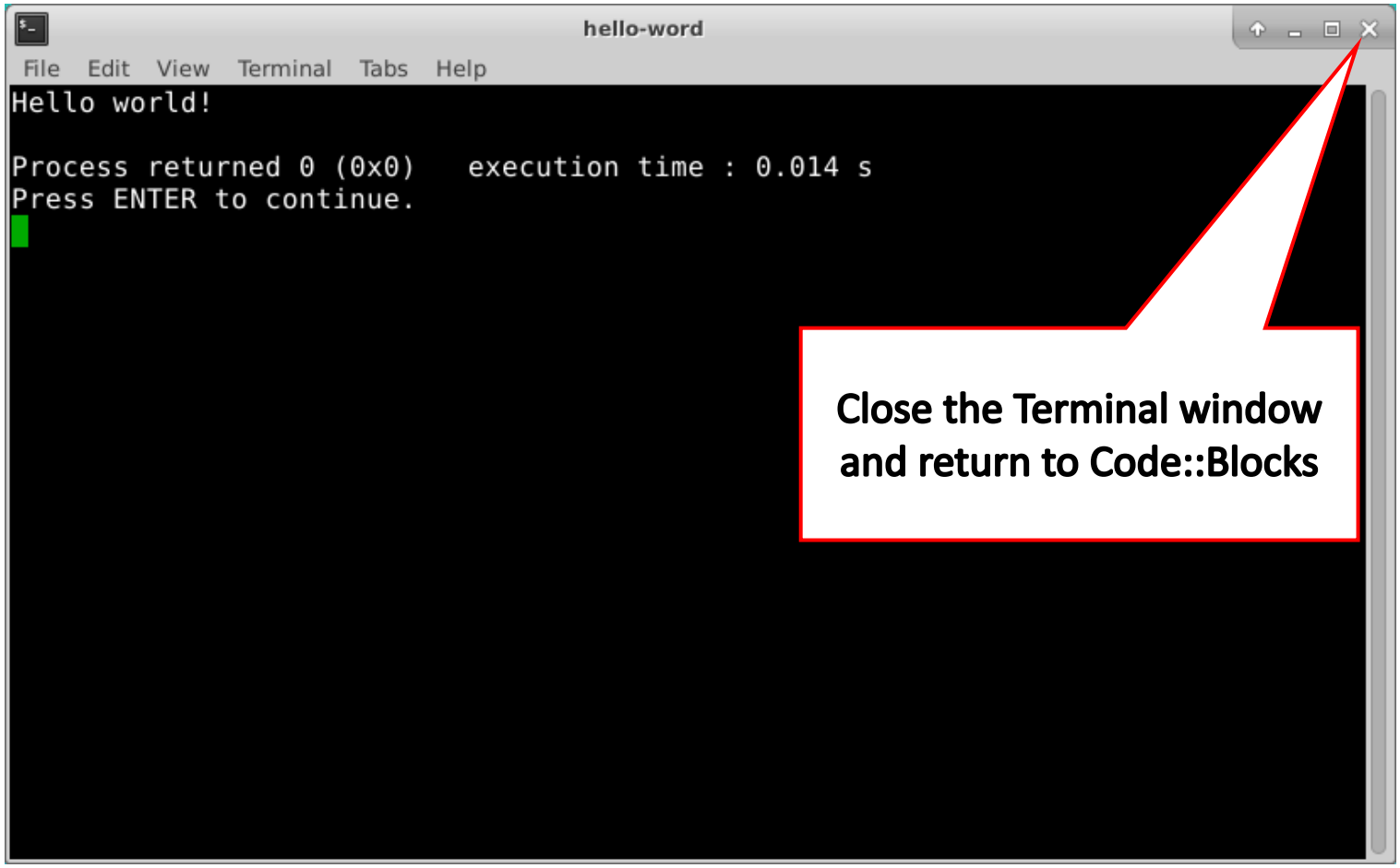
Double click on the ".cpp" **(C++)** file that matches the lab topic

Lab 1 – Hello World!



To run the application, click on the green triangle *with* the gear symbol

Lab 1 – Hello World!



Hints on Using Xfce & Code::Blocks

- Terminal = Shell = Console = Command Line
  - The shell is used to interact with character mode applications
  - Linux programmers mostly use the command line
  - Shell commands and file names are **case-sensitive**
  - Don't accumulate open **terminal windows** – close them!
- Only keep one (1) instance of **Code::Blocks** open
  - After each lab, close all open instances of Code::Blocks
  - To open a lab, be sure to double-click on the **.cbp** file (.~~cpp~~!)
- Your session will auto-logout after an hour of of inactivity

Identifiers

- Identifiers are just **names** – everything in code has a name!
  - Name must be < 64 chars in length, upper or lower case, numbers
  - Identifiers must start with a letter and **cannot contain spaces!**
- Three types of identifier "casing"
  1. CamelCaseEachWord (first letter is Capitalized)
  2. camelCaseEachWord (first letter is not capitalized)
  3. all lowercase **(most common in C++)**
- Identifiers in C++ are **case sensitive!!**
  - ***n is not the same as N***
  - *Never* use ALLCAPS

Identifiers

```
// AgeInSeconds.cpp

#include "stdafx.h"

using namespace std;

int main()
{
    double ageInYears = 49;

    double ageInSeconds =
        ageInYears * 60 * 60 * 24 * 365;

    cout.imbue(std::locale(""));
    cout << fixed << setprecision(2);

    cout << "Age (years) = "
        << ageInYears << endl;

    cout << "Age (secs) = "
        << ageInSeconds << endl;

    system("pause");
    return 0;
}
```

Function Name

Variable Name

Object Name

Method Name

C++ Keyword

Statements & Scopes

- A **statement** does something
  - A statement is either a declaration, a keyword, or a function
  - Statements are executed from top to bottom of the program
  - **Statements must be terminated with a semicolon!!**
- A **scope** contains one or more statements
  - A scope with **more than one statement** must be enclosed within open (left) and close (right) curly braces **{}**
  - If you only want one statement in a scope, you don't need **{}**
  - Scopes can be *nested* – each inner scope is further <u>indented</u>
  - Scope braces are like parenthesis – they must be balanced!
  - **You don't need a ; after a closing right brace }**

Scopes

```
// AgeInSeconds.cpp

#include "stdafx.h"

using namespace std;

int main()
{
    double ageInYears = 49;

    double ageInSeconds =
        ageInYears * 60 * 60 * 24 * 365;

    cout.imbue(std::locale(""));
    cout << fixed << setprecision(2);

    cout << "Age (years) = "
        << ageInYears << endl;

    cout << "Age (secs) = "
        << ageInSeconds << endl;

    system("pause");
    return 0;
}
```

Opening of Scope

Statements are terminated with a semicolon ;

Closing of Scope

No semicolon needed after close of scope }

Variable Types

- **Variables** store data in memory to be used later
  - Variables can be called whatever <u>you</u> want
  - Create variable names that mean something to a programmer
  - Use **camelCase #2** (first letter is lower case)
- Intrinsic (built-in) **types** for variables:
  - **int** = Stores integers (–2,147,483,648 to 2,147,483,647)
  - **double** = Stores real numbers with 15 digits of precision
  - **bool** = Stores “true” or “false” (Boolean data type)
  - **string** = Stores zero or more letters & numbers

Creating a Variable

- You must ***declare*** <u>and</u> ***define*** a variable before you can use it
- **Declaring** a variable means to specify a <u>type</u> for an identifier
  - The data type always *precedes* the name of the variable
  - You can ***declare*** a variable **only once** per scope
- **Defining** a variable means to give it a specific <u>value</u>
  - The value type must match the variable type
  - You can ***define*** a variable **multiple times** per scope

```
int totalVotes = 123;
double hourlyPay = 9.75;
string firstName = “Dave”;
```

Operators

- C++ operators obey normal **PEMDAS** precedence
  - Expressions are evaluated left to right in your source code
  - Use **=** to assign a value to a variable
  - Use ***** for multiplication and **/** for division operators
  - Use parenthesis to explicit specify the order of operations
  - "Greater than or equal to" operator is **>=**
  - "Less than or equal to" is operator **<=**

```
double degC = (degF - 32) * 5. / 9;
```

The period (decimal point) forces the numerator to be evaluated as a **double**

Displaying Variables

- **cout** is used to display the value of variables
  - Represents "console output" (pronounced see-out, not coot)
  - The variables to display follow the stream insertion *operator* **<<**
  - Don't forget the first and last quotation marks for literals!
- To specify the # of digits to show to **right** of decimal point:

```
cout << fixed << setprecision(2);
```

- To **right justify** a column:

```
cout << setw(7) << right;
```

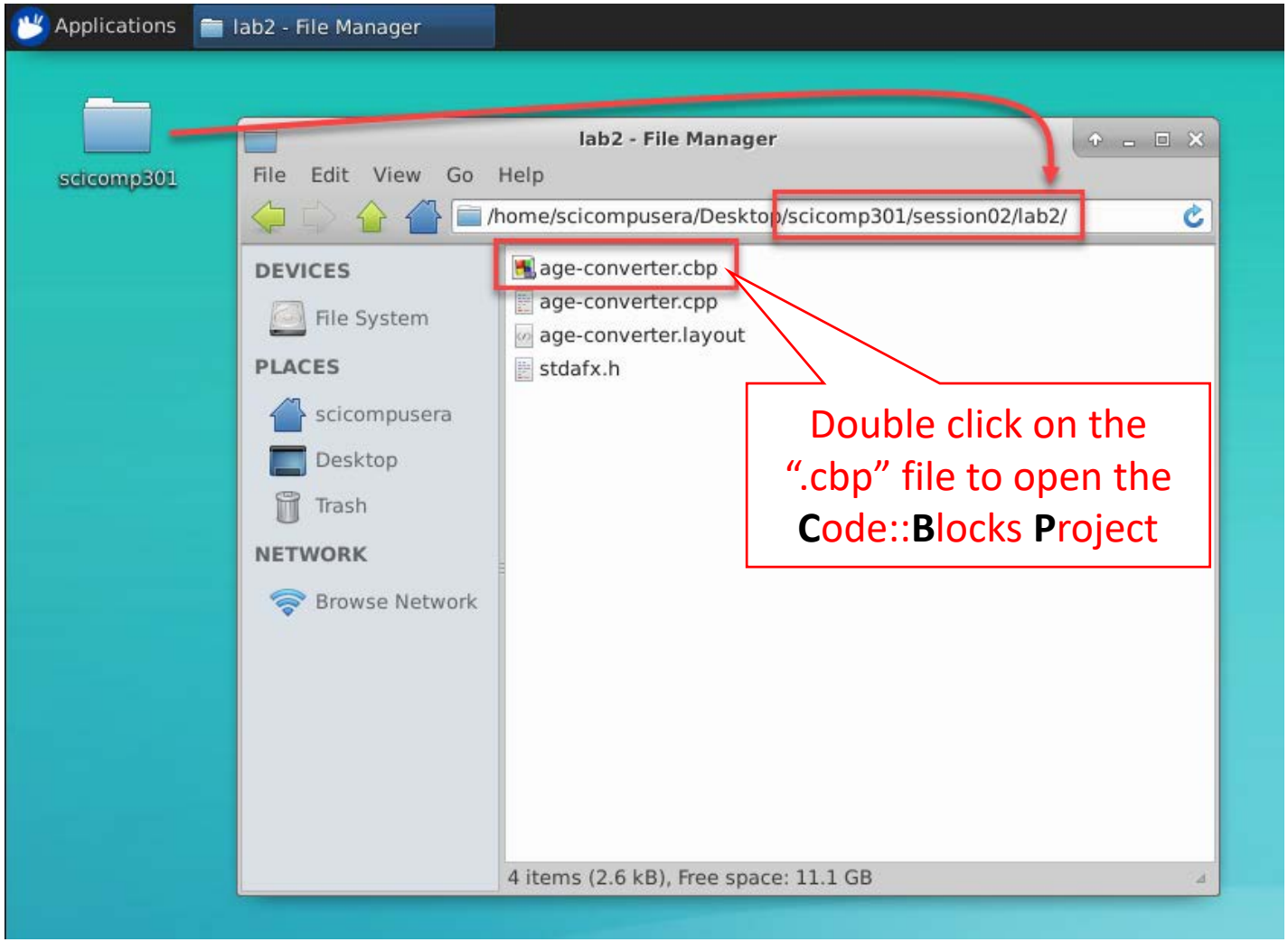
- To adopt local language preference for digit separators:

```
cout.imbue(std::locale(""));
```

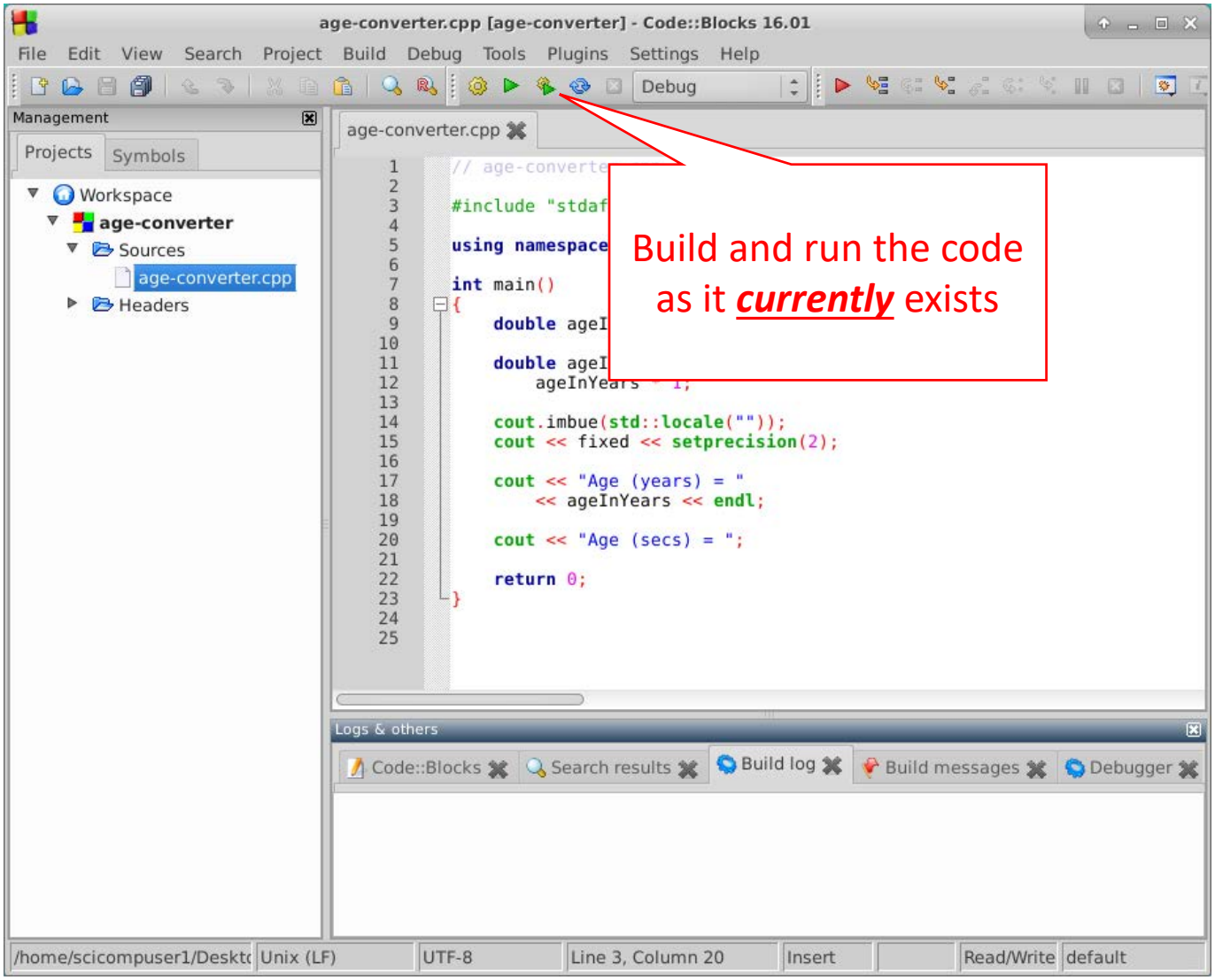
Lab 2 – My Age In Seconds

- Your scientist has asked you to fix an existing buggy C++ console application using Code::Blocks
- Initialize a variable to hold your current age in years
- Develop and implement an equation that performs the correct **dimensional analysis** (factor label method) to convert years to seconds
- There is no need to accommodate **leap years** in the equation
- Display in the terminal window **both** your age in years ***and*** your age in seconds

Lab 2 – My Age In Seconds



Lab 2 – My Age In Seconds



Lab 2 – My Age In Seconds

age-converter

File Edit View Terminal Tabs Help

```
Age (years) = 0.00
Age (secs) =
Process returned (0x0)   execution time : 0.012 s
Press ENTER to con
```

The code (as written) has 3 bugs:

1. It doesn't use the correct years
2. It doesn't actually convert years to seconds
3. It doesn't display the value of the age in seconds variable

Lab 2 – My Age In Seconds

age-converter.cpp [age-converter] - Code::Blocks 16.01

File Edit View Search Project Build Debug Tools Plugins Settings Help

Management

Projects Symbols

- Workspace
  - age-converter
    - Sources
      - age-converter.cpp
    - Headers

Fix the buggy code

age-converter.cpp

```cpp
// age-converter.cpp

#include "stdafx.h"

using namespace std;

int main()
{
    double ageInYears = 0;

    double ageInSeconds =
        ageInYears * 1;

    cout.imbue(std::locale(""));
    cout << fixed << setprecision(2);

    cout << "Age (years) = "
        << ageInYears << endl;

    cout << "Age (secs) = ";

    return 0;
}
```

Logs & others

Code::Blocks | Search results | Build log | Build messages | Debugger

/home/scicompuser1/Deskt | Unix (LF) | UTF-8 | Line 3, Column 20 | Insert | Read/Write | default

Lab 2 – My Age In Seconds

age-converter.cpp [age-converter] - Code::Blocks 16.01

Build Debug Tools Plugins Settings Help

age-converter.cpp

```
1  // age-converter.cpp
2
3  #include "stdafx.h"
4
5  using namespace std;
6
7  int main()
8  {
9      double ageInYears = 49;
10
11     double ageInSeconds =
12
13
14     cou
15     cou
16
17     cou
18
19
20     cou
21
22
23     ret
24 }
25
26
```

Your numbers will be different based upon your current age

File Edit View Terminal Tabs

```
Age (years) = 49.00
Age (secs) = 1,545,264,000.00

Process returned 0 (0x0)   execution time : 0.011 s
Press ENTER to continue.
```

Logs & others

for() Loops

- **for**() loops execute all the statements within their scope as long as the 2nd part of the loop definition remains **true**

```
// Sum the numbers 1 to 10
double sum = 0;
for (double n = 1; n <= 10; n = n + 1)
{
    sum = sum + n;
}
```

The 3-part loop definition

- The <u>three</u> parts of the **for**() loop definition are:
  1. A statement to **declare the loop counter** variable
  2. A Boolean condition to define **how long** the loop should run
  3. An iterator statement to **adjust the loop counter** <u>*after*</u> each pass

Lab 3 –Temperature Converter

- Fix the code to calculate the correct Celsius temperature for a given Fahrenheit temperature
- Display values between **-44°F** and **216°F** ***inclusive***
- Your code should increment in steps of **4°F**
- The research question your scientists wants you to solve:

What is the **one temperate that is the <u>same</u>** in both Fahrenheit and Celsius?

Lab 3 – Temperature Converter

temperature-converter.cpp [temperature-converter] - Code::Blocks 16.01

File Edit View Search Project Build Debug Tools Plugins Settings Help

Management

Projects Symbols

- Workspace
  - temperature-c
    - Sources
      - temperatur
    - Headers

temperature-converter.cpp

```
1   // temperature-converter.cpp
2
3   #include "stdafx.h"
4
5   using namespace std;
6
7   int main()
8   {
9       cout.imbue(std::locale(""));
10      cout << fixed << setprecision(2);
11
12      for (double degF{ -44 }; degF <= 216; degF += 4) {
13
14          double degC = 0;
15
16          cout << setw(7) << right << degF << " F = "
17              << setw(7) << right << degC << " C"
18              << endl;
19      }
20
21      return 0;
```

Same as degF = 44

Same as degF = degF + 4

Set next column width

Right justify next column

Lab 3 – Temperature Converter

temperature-converter

File Edit View Terminal Tabs Help

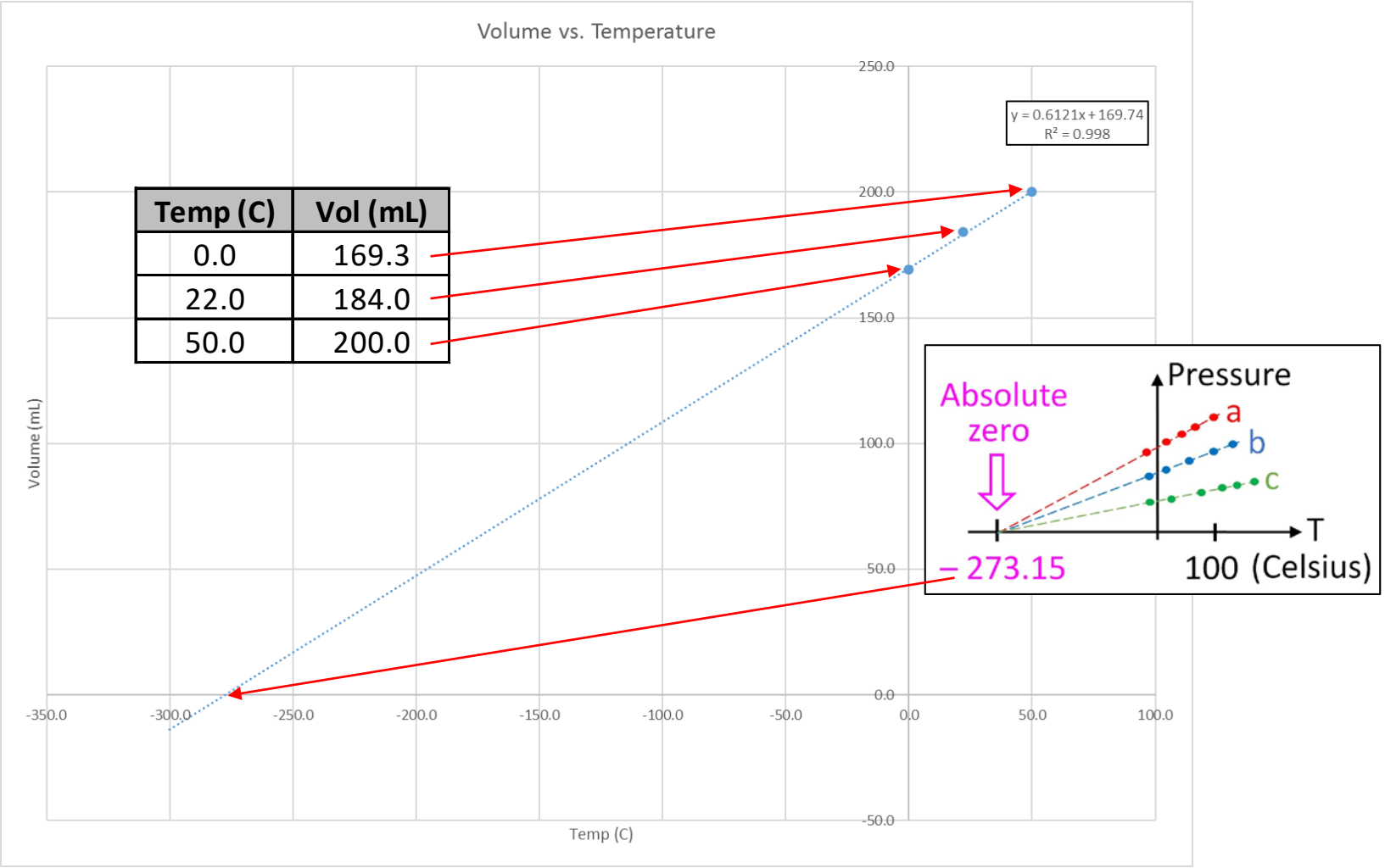
```
 140.00 F =   60.00 C
 144.00 F =   62.22 C
 148.00 F =   64.44 C
 152.00 F =   66.67 C
 156.00 F =   68.89 C
 160.00 F =   71.11 C
 164.00 F =   73.33 C
 168.00 F =   75.56 C
 172.00 F =   77.78 C
 176.00 F =   80.00 C
 180.00 F =   82.22 C
 184.00 F =   84.44 C
 188.00 F =   86.67 C
 192.00 F =   88.89 C
 196.00 F =   91.11 C
 200.00 F =   93.33 C
 204.00 F =   95.56 C
 208.00 F =   97.78 C
 212.00 F =  100.00 C
 216.00 F =  102.22 C

Process returned 0 (0x0)   execution time : 0.016 s
Press ENTER to continue.
```

Scroll up to see the initial values

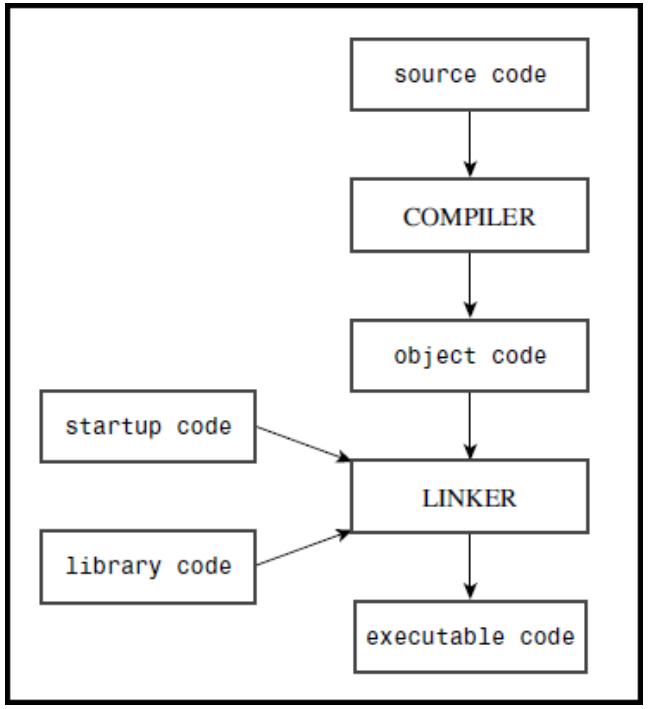
What is the **one temperate that is the same** in both Fahrenheit and Celsius?

How did we calculate absolute zero in 1779? (PV = nRT)



Temp (C)	Vol (mL)
0.0	169.3
22.0	184.0
50.0	200.0

C++ Build Process



.h	header files - function declarations
.cpp	function definitions
.obj	output of compiler
.lib	output of linker - static library, embedded in final EXE
.dll	Windows shared object - dynamic link library
.exe	Windows executable - final output of linker

Header File Naming Conventions

Kind of Header	Convention	Example	Comments
C++ old style	Ends in `.h`	`iostream.h`	Usable by C++ programs
C old style	Ends in `.h`	`math.h`	Usable by C and C++ programs
C++ new style	No extension	`iostream`	Usable by C++ programs, uses `namespace std`
Converted C	`c` prefix, no extension	`cmath`	Usable by C++ programs, might use non-C features, such as `namespace std`

A header between <> brackets (preferred) adds symbols to the compiler's **std** namespace, not the user's global namespace like "header.h" will

Now you know…

- How to access your remote Amazon PC
- Linux with Xfce is quite similar to Microsoft Windows
- How to create a open a Code::Blocks project
- How to build & run a C++ console (shell) application
- The C++ build process is two phase: **compiling** then **linking**
- How to log out of your Linux (Ubuntu) desktop session

Now you know…

- Rules for identifier casing
- C++ intrinsic **data types**
- Declaring and defining variables
- Statements and scopes **{}**
- Operators & Precedence
- Assignment Operator
- How to write messages and variables to the console window
- **for**() loops
- -40° F = -40° C
- Absolute Zero = -273.15° C