COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Lecture 4
Elements of Development

Prof. Brendan Kochunas

9/16/2019

NERS 590-004

Outline

- (?) Linear Regression Analysis
- (?) Python
- (?) LaTeX

- Elements of Development
  - Configuring
  - Compiling
  - Linking

- Program Design, Infrastructure, Testing, Debugging will be covered in other lectures ☺

LaTeX

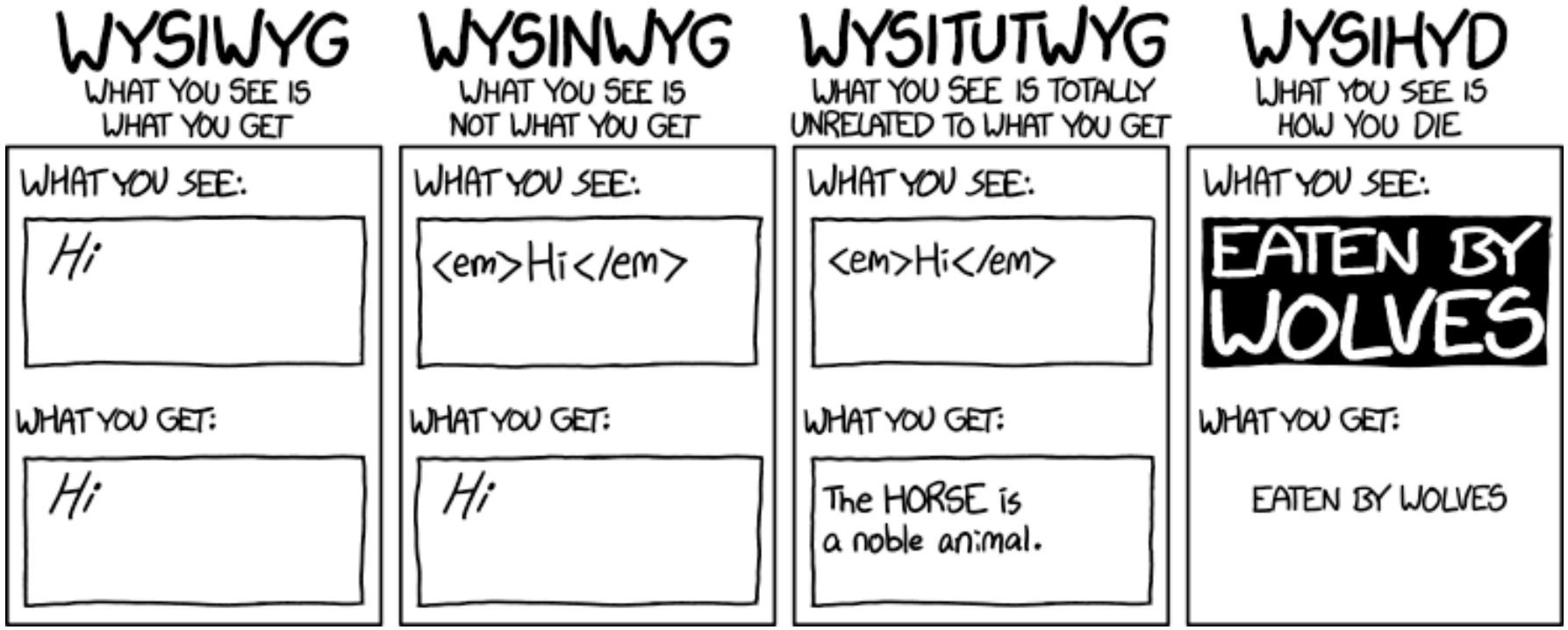
A little bit about LaTeX

- LaTeX (Lamport TeX) is document preparation software with automated typesetting
- Widely used for the publication of scientific documents
  - It allows the writer to follow a template with minimal effort
  - Very straightforward expression of mathematical expressions and equations
  - Easy management of citations and cross-references
  - Portable across many platforms
- Many packages available for specific symbols or layouts
  - Algorithms
  - Graphics
  - Hyper-references
  - Extensions for basic packages like math, tables, etc.
- User can define useful shortcuts for commands that are used frequently
- Basically a programming language all its own.

Downsides of LaTeX

- Installation of LaTeX compilers and packages required
  - Working on different machines may make it difficult to keep packages/documents consistent
- Not a WYSIWYG (what you see is what you get) language
  - Need to “compile” a “code” just to generate a document
  - It can sometimes be REALLY annoying to do something that you think is trivial
    - Usually not an issue unless you want some sort of very special/unique formatting
    - A lot of commands/packages are black boxes that can be difficult to understand/modify
    - e.g., trying get figures to sit exactly where you want them may require many trials and an absurd amount of Googling

Types of Editors: https://xkcd.com/1341/



Compiling and Editing in LaTeX

- A LaTeX PDF can be generated from a .tex file via the command line:
  - `pdflatex <filename>.tex`
    - running the command twice may be necessary to resolve/updated references
- Editors for LaTeX allow users to avoid compiling via the command line
  - Options exist for every platform
    - e.g.: Texmaker, TeXstudio, Sublime Text (via a plugin), TeXShop, TeXworks
    - LyX is a WYSIWYG LaTeX editor that allows users to avoid writing actual LaTeX code
  - We highly recommend https://v2.overleaf.com/ (formerly www.sharelatex.com)
    - Online-based editor (kind of like Google Docs for LaTeX)
      - Free to use unless you want to collaborate with others
    - No need to download any packages or figure out how to install/compile LaTeX
    - Cross-platform in the sense that your work is saved on the cloud and your output will be independent of the computer you use

Creating Your First LaTeX Document

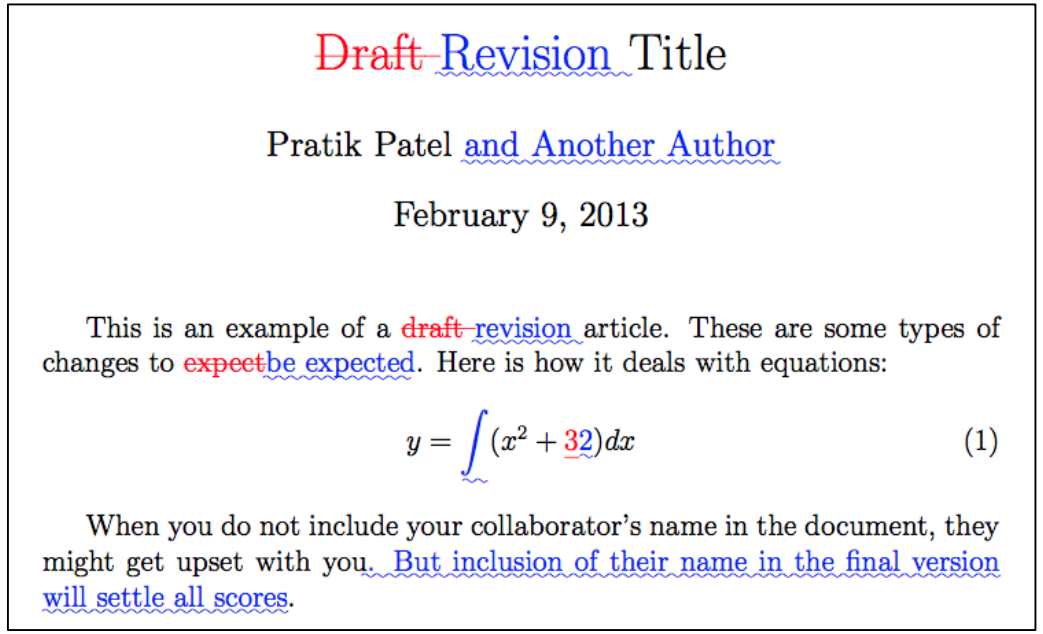
- Very easy to generate a document, even if you've never used LaTeX before
- Most commands are intuitively named and can be found via a simple Google search
  - Examples: `\gamma` produces the Greek letter gamma, `\int` produces an integral sign

```
\documentclass{article}
\usepackage[utf8]{inputenc}
 begin{document}
    your content here…
\end{document}
```

- Example LaTeX document: https://www.sharelatex.com/project/57d858d53414e8de018b3643

latexdiff

- latexdiff is a great tool for comparing the contents of two similar latex files
  - Similar to “track changes” on Microsoft Word
  - `latexdiff file1.tex file2.tex` produces a .tex file which, when compiled, yields a PDF with markups showing all the changes from file1.tex to file2.tex
  - See https://www.sharelatex.com/blog/2013/02/16/using-latexdiff-for-marking-changes-to-tex-documents.html for more information



~~Draft~~ Revision Title

Pratik Patel and Another Author

February 9, 2013

This is an example of a ~~draft~~ revision article. These are some types of changes to ~~expect~~be expected. Here is how it deals with equations:

$$y = \int (x^2 + 32)dx \qquad (1)$$

When you do not include your collaborator's name in the document, they might get upset with you. But inclusion of their name in the final version will settle all scores.

Introduction to Python

Based on 2016 Introduction to Python Workshop
https://goo.gl/uYrc0e

Outline for Intro to python

- What is Python and where does it fit?
- Syntax and arithmetic
- If, elif, and else
- Sequences and for loops
- Data processing (numpy)
- Plotting
- Workflow options
- Further reading

Common Programming Languages

Compiled

- Fortran
- C
- C++
- Java

Interpreted

- bash
- C shell
- Perl
- make
- Python
- Matlab
- R
- SPSS

Scripting

Scientific Calculation

Interpreted Languages for Scientific Calculations

MATLAB

- Based somewhat on Fortran
  - syntax is very much like C.
- Costs
  - Free for students
  - $2150 for regular people
  - +$1000 for each add on package
    - Simulink
    - statistical toolbox
    - parallel computing, etc.

Python

- Based on C/C++
- $0!
  - Comes with most linux distributions and Mac.
  - Some distributions offer add ons and support for $$$
- Python v3 not compatible with Python v2 :(
  - `__future__` package to help transition

Executing python

- Interactively
  - Start a python "shell": `$ python`
- Run a script
  - `$ python myscript.py`
- Run a script interactively
  - `$ python -i myscript.py`

Basic Syntax and Operators

Arithmetic Operators

```
print('Hello world')

x = 10.

print(x + 5.)
print(x - 4.)
print(x * 2.)
print(x / 3.)

print(x ** 2.)
# (x^2 in MATLAB)
```

Boolean Operators

```
x = 10.
# comparison
print(x == 15.)
print(x <= 12.)
print(x != 5.)

# boolean logic
print(not True)
print(True and False)
print(True or False)
```

Intrinsic types

- Python uses “dynamic typing”
  - No need to explicitly declare `int`, `float`, `bool`, etc.
  - Types are *implied*

```
x = 10.              #implied float
i = 1                #implied int
l = True             #implied bool
c = “Hello World”    #implied string
```

- Declare variables anywhere!

```
#Python 2
2/3 == 0
2./3. == 0.66667

#Python 3
2/3 == 2./3. == 0.66667
2//3 == 0
```

Execution Control Constructs

If-Else

```
# indentation required:
if condition1:
    statements
elif condition2:
    other statements
elif condition3:
    other statements
else:
    alt statements
```

Looping

```
# i = 0, 1, ..., n-1
for i in range(n):
    print(i)

# iterate through a
# sequence
x_list = [1, 2, 4, 8]
for x in x_list:
    print(x)
```

Python Lists

- Effectively the implementation of arrays
  - Can be mixed “types”
- But these arrays are fancy
  - Can function like the classical data structure definitions of stacks, queues, or decks
- Operations on lists are very easy compared to other languages.
- Not very fast...
  - For speed you want numpy.

```
VY0 = [2., 3., 4.]
print(VY0)
print(VY0[0],VY0[-1])
print(len(VY0))

VY0.append(5.)
VY0[2] = "cat"
print(VY0)

VY0.pop(0)
print(VY0)
```

Python For Scientific Calculations

- Python is designed for general computing, but it has modules to facilitate scientific computing.
- The **numpy** and **scipy** modules are designed for
  - large N-dimensional numeric arrays
  - efficient loop operations using compiled code (BLAS and LAPACK)
  - transcendental functions, optimization, signal processing, ...
- Trilinos also offers python interfaces in pyTrilinos
- In fact, it is fairly easy to generate python interfaces to compiled code using SWIG.
  - Basically anything that can provide a C interface can be called from another programming language.

A little more about packages…

- Similar to how C has the C standard library, python has packages
  - Except they are not “standard”
  - You have to install them in addition to python
- Some distributions for python include a bunch of packages
  - e.g. anaconda, canopy, etc.
- Common packages for scientific computing
  - matplotlib – plotting tools
  - numpy - operations on bulk data
    - Compiled C libraries (fast)
  - scipy – statistics, analysis, curve fitting
  - h5py – HDF5 interface for python
  - mpi4py – MPI interface for python

Importing packages

```
# import specific items
from math import sin, pi
print(sin(pi))

# import and rename items
from random import choice as c
print(c([1, 5, 7]))

# import (& rename) package
import numpy as np
print(np.pi)

# import a subpackage
import matplotlib.pyplot as plt
```

A race! (or why you want numpy…)

- This demonstrates problems of overhead with interpreted languages.
  - e.g. having another run time system on top of the operating system.

Native Python

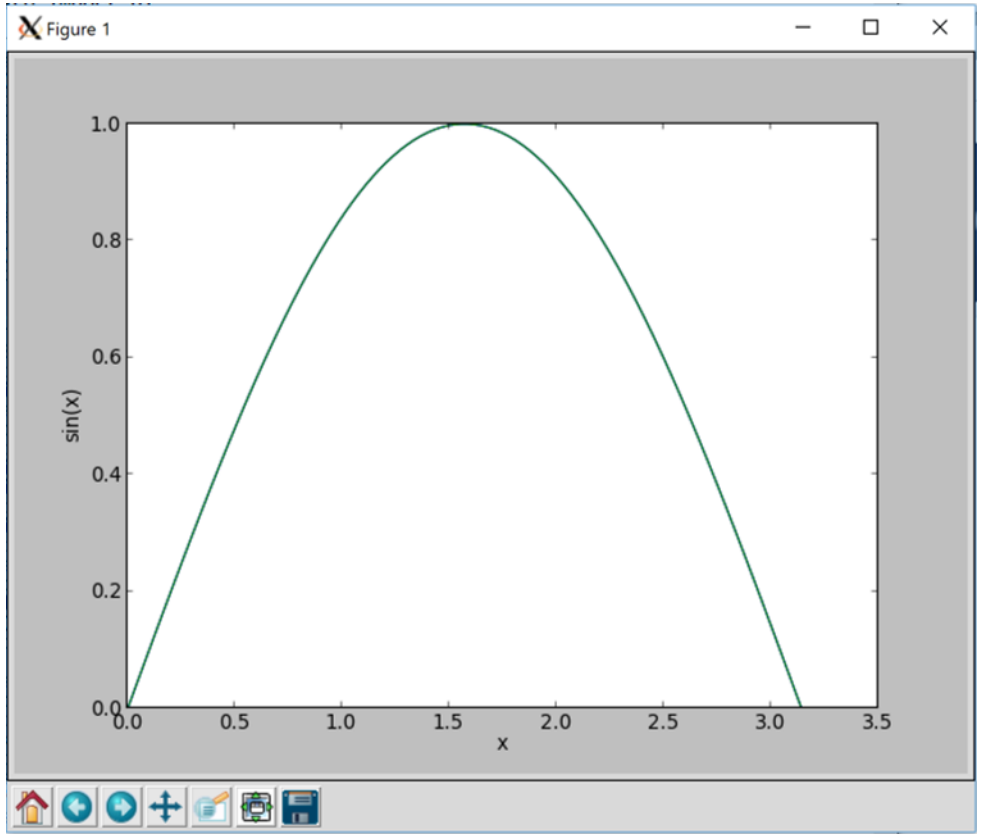
```
N = int(1e7)
x1 = N * [1.]
x2 = N * [0.]
for i in range(N):
 x2[i] = x1[i] **2
print('done with x2')
```

Numpy

```
import numpy as np
x3 = np.ones(N)
x4 = x3 ** 2
print('done with x4')
```

Plotting Example

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0,np.pi,100)
y = np.sin(x)

# Commands are very
#   similar to MATLAB
# Basic idea:
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.show()
```



Further Reading

- Numpy and Scipy
  - https://docs.scipy.org/doc/
- Gorelick and Ozsvald, “High Performance Python,” O’Reilly Media, 2014.
  - https://search.lib.umich.edu/articles/record/FETCH-LOGICAL-a20226-f73b8557ff448d247627a886c1ffc2c90a965f101c963ddd998713efc6af39573
- MOOC: https://www.sololearn.com/Course/Python/
- A great new tool for productivity is the Jupyter Notebook
  - https://jupyter-notebook-beginner-guide.readthedocs.io
- Scientific computing distributions of Python
  - Anaconda and miniconda
  - Canopy/Enthought

Motivation

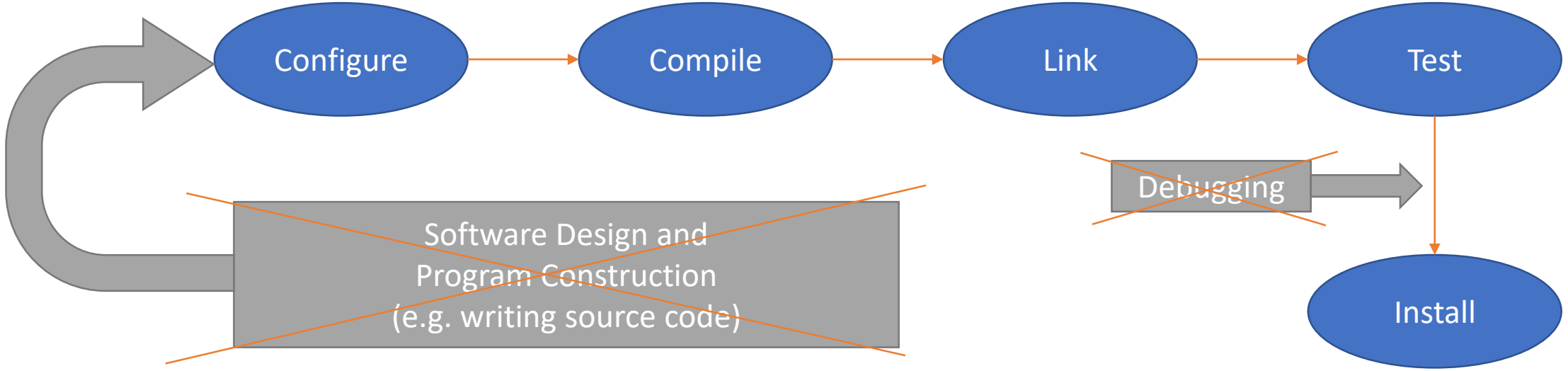
- Learn skill to solve your own problems
  - not fully dependent on “tech support”
- Helps you be more adept at using software distributed by others
- Automate developer workflow

Teaching Objective

- Understand the difference between configuring, compiling, and linking
- Understand the subtleties in these steps which you will need to know to complete homework 1.
- *Really* understand how a compiler works
  - e.g. We are “going to learn how the sausage is made”
  - Should give you everything you need to know for homework 1

Elements to Development

- Program Design, Testing, and Debugging are all integral parts of development.
  - They each will get their own lecture later in the semester.
- This lecture is focused on the tools and steps needed to build software:



The "Toolchain"

Definitions

- The tools (e.g. programs and their libraries) typically used by the developer that are needed to take your source files (as input) and produce executables.
- Frequently the definition of a tool chain will also include:
  - a program for "configuring"
  - a "make" script and program
  - a distribution of MPI
- Third Party Libraries (TPLs) are typically not considered part of the "toolchain"
  - MPI is sort of the exception to this
- Toolchains represent part of the "Minimum Requirements" for a software package.
  - Software packages can also support multiple toolchains

Examples

- In the CASL project
  - CMake 2.8.11
  - autoconf-2.69 (auto tools)
  - gnu-4.8.3 compilers
  - mpich-3.1.3 (MPI library)
  - Assumes GNU Make or Make is available on the system
    - Almost always available on linux since it may be needed to install (parts of) the OS
- Other examples
  - Trilinos – supports many
  - PETSc – supports many
  - xSDK – being defined for future computational science and engineering applications

Some Terminology about "Time"

- *configure-time* – when you are configuring
- *compile-time/build-time* – when you are compiling
- *link-time* – when you are linking (very last step in "compiling")
- *run-time* – when you are running the executable
- Important for communicating "when things went wrong"

What is Configuring?

For Scientific Software Development

- Purpose is to probe the system for to determine
  - the computer architecture (e.g. x86, AIX, etc.)
  - what compilers are installed, and where they are installed
  - what additional system software is installed
  - what third party libraries are available
- Basically creates “Makefiles” for a specific computer and environment for use in compiling (the next step)
- Specialized programs exist to perform the “configure” step.
  - autotools (autoconf/automake) → `configure`
  - CMake → `cmake`
- Most difficult part to establishing complex software systems.
- Considered part software infrastructure
- AUTOMATE as much as possible

Imperfect Cooking Analogy

- Look around and find all the necessary ingredients and cookware (pots, pans, knives) to make dinner
  - Your stomach
  - Appliances
  - Spatula, spoon and knives
  - Salt, Pepper, Spices and Sauces
- Detailed instructions to cook in your kitchen
- Cookbooks
- Writing your own cookbook for others
- It’s a part of life
- Use pre-packaged frozen meals

Things that Configuring Can Control

- Where the libraries and executables produced from compilation are installed
- Compiler options:
  - e.g. whether the libraries and executables are “debug” (slow) or “release” (fast)
  - e.g. whether the compilation produces “dynamic” or “static” binaries
- What features of the library are enabled
  - e.g. with HDF5 you can specify whether you want the compiled library to include Fortran interfaces or just C interfaces.
  - e.g. what third party libraries to include (often provide additional capability in the software package
    - In HDF5 the “Z” library can be included to provide data compression.
- Various other options that would be specific to the software package.

Configuration Options

The Cliff's Notes

- Depends on the tool!
- You will *always have to read documentation*
  - Usually files are named `INSTALL` or `README`
- Hopefully libraries and programs you work with are well documented
- Make sure you document your configuration steps well (or make them "robust") in software you produce
  - People won't use your software if it is difficult to install or build.

"Common" Usage and Options

- Autotools (autoconf/automake)
  - `./configure [options]`
    - `--prefix=<path_to_install>`
    - `CC=<c_compiler>`
    - `FC=<fortran_compiler>`
- CMake
  - `cmake [options] <path_source_dir>`
    - `-DCMAKE_BUILD_TYPE="Release"`
    - `-DCMAKE_C_COMPILER`
    - `-DCMAKE_CXX_COMPILER`
    - `-DCMAKE_Fortran_COMPILER`

Troubleshooting the Configure Step

For Scientific Software Development

- My configure failed! What do I do?
  - Uh oh… Usually difficult to resolve
- At best, problem is solved by modifying your environment.
- At worst, may require some other software be installed
  - And you may not have privileges to do so!
  - In this case you're likely 
  - …unless you're willing to put in way more time than is appropriate

For Cooking

- I can't find everything I need to cook dinner!
  - I'm hungry now, but I need to grill this steak!
- Ask your neighbor for a cup of sugar and some eggs or a melon baller
- You mean I need an grill!?
  - But I live in an apartment!
  - Guess you'll go hungry
  - …unless I can make friends with someone who has a grill

Examples of Configuration Scripts

CMake (CMakeLists.txt) for METIS

```
cmake_minimum_required(VERSION 2.8)
project(METIS)

set(GKLIB_PATH "GKlib" CACHE PATH "path to GKlib")
set(SHARED FALSE CACHE BOOL "build a shared library")
if(MSVC)
  set(METIS_INSTALL FALSE)
else()
  set(METIS_INSTALL TRUE)
endif()
# Configure libmetis library.
if(SHARED)
  set(METIS_LIBRARY_TYPE SHARED)
else()
  set(METIS_LIBRARY_TYPE STATIC)
endif(SHARED)
include(${GKLIB_PATH}/GKlibSystem.cmake)
# Add include directories.
include_directories(${GKLIB_PATH})
# Recursively look for CMakeLists.txt in subdirs.
add_subdirectory("include")
add_subdirectory("libmetis")
add_subdirectory("programs")
```

Autoconf (configure.in) for PAPI

```
AC_PREREQ(2.59)
AC_INIT(PAPI, 5.5.0.0, ptools-perfapi@eecs.utk.edu)
AC_CONFIG_SRCDIR([papi.c])
AC_CONFIG_HEADER([config.h])

AC_MSG_CHECKING(for architecture)
AC_ARG_WITH(arch,
                [  --with-arch=<arch>   Specify architecture (uname -m)],
                [arch=$withval],
                [arch=`uname -m`])
AC_MSG_RESULT($arch)

AC_ARG_WITH(bitmode,
           [  --with-bitmode=<32,64>  Specify bit mode of library],
           [bitmode=$withval])

AC_MSG_CHECKING(for OS)
```

Autotools Configure Example Cont.

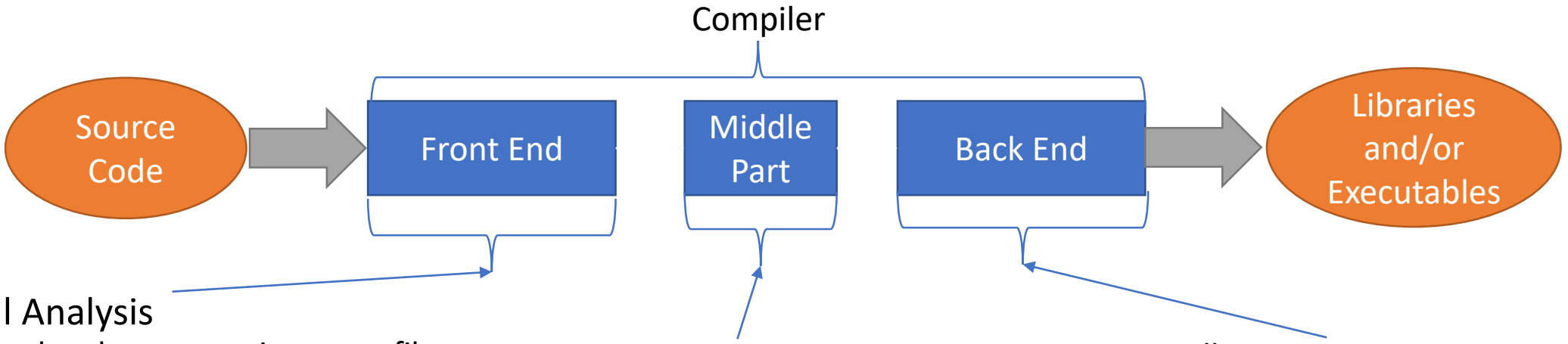
```
#!/bin/sh
# Guess values for system-dependent variables and create Makefiles.
# Generated by GNU Autoconf 2.59 for PAPI 5.5.0.0.

# Identity of this package.
PACKAGE_NAME='PAPI'
PACKAGE_TARNAME='papi'
PACKAGE_VERSION='5.5.0.0'
PACKAGE_STRING='PAPI 5.5.0.0'
PACKAGE_BUGREPORT='ptools-perfapi@eecs.utk.edu'

ac_unique_file="papi.c"
# Factoring default headers for most tests.
ac_includes_default="\
#...
#if STDC_HEADERS
# include <stdlib.h>
# include <stddef.h>
#else
# if HAVE_STDLIB_H
#  include <stdlib.h>
# endif
#endif
```

- As a user you typically will not run `autoconf`
  - But as a developer you might
- As a user you will run configure shell script produced by autoconf.
- Conventional wisdom for developers
  - CMake >> ~~Autohell~~ Autotools
- **If starting new, start with CMake**.

What is a Compiler?

- It's a program written by people
  - so it has bugs
  - And different versions and the behavior between versions can vary significantly.
- Translates a high-level programming language (suitable for humans) into a low level machine language required by the computer.
- Typically have several common features
  - Checking for syntax and programming errors
  - Supply debugging information
  - Perform optimizations

Modern Compiler Program Architectures



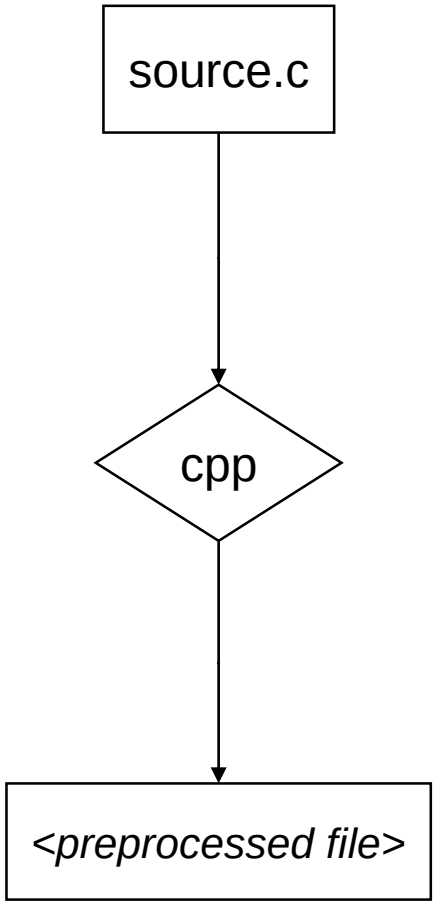
- Lexical Analysis
  - Read and parse text in source files into *tokens*
- Syntax Analysis
  - Arrange tokens in syntax tree to reflect program structure
- Type checking
  - Checks syntax tree for mistakes (e.g. undefined variables)

- Intermediate coded generation
  - Translation to simple machine independent language
  - Typically will vary between compilers
- Optimization
  - Apply algorithms for optimization to intermediate language

- Register allocation
  - Translate variables to machine registers (memory locations)
- Machine code generation
  - Translate intermediate language to machine code (assembly)
- Assembly and linking
  - Convert assembly to binary and resolve addresses for variables and functions

Other things a compiler does (sort of)

- Preprocess (separate program executed by compiler during compilation, "**cpp**")
  - ***Modifies source files***
  - A part of C, but can be used in the compilation of C++ & Fortran
    - In Fortran file extensions control default behavior:
    - *.F, *.F90 are automatically preprocessed, and *.f, *.f90 are not
  - Based on "directives", start with "#" in first column.
    - Include files, macro expansion, conditional compilation
  - Compilers will predefine some symbols for you
- Link (separate program executed by compiler during compilation, "**ld**")
  - We'll discuss linking in a few slides…



Some Preprocessor Examples

C

```
#include <stdio.h>

//Define macros
#define PI 3.14159
#define RADTODEG(x) ((x) * 57.29578)

int main()
{
    float x;
    x=RADTODEG(PI*0.5);
    printf("PI/2 radians in degrees is %.6f\n",x);
    return 0;
}
```

```
int main()
{
    float x;
    x=((3.14159*0.5) * 57.29578);
    printf("PI/2 radians in degrees is %.6f\n",x);
    return 0;
}
```

Fortran

```
PROGRAM main

#ifdef __GFORTRAN__
  WRITE(0,'(a,i2,a)') 'File: "'//__FILE__//'", line ',__LINE__, &
    " was compiled with gfortran!"
#else
  WRITE(0,'(a,i2,a)') 'File: "'//__FILE__//'", line ',__LINE__, &
    " was NOT compiled with gfortran!"
#endif

ENDPROGRAM
```

```
PROGRAM main


  WRITE(0,'(a,i2,a)') 'File: "'//"hello.F90"//'", line ',4, &
    " was compiled with gfortran!"


ENDPROGRAM
```

Compiler options for debugging

GCC compiler option	Meaning
`-g`	Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF). GDB can work with this debugging information.
`-fsanitize=<opt>`	Enable AddressSanitizer, a fast memory error detector.
`-fbounds-check`	Generate additional code to check that indices used to access arrays are within the declared range during run time.
`-fcheck-pointer-bounds`	Each memory reference is instrumented with checks of the pointer used for memory access against bounds associated with that pointer.
`-fstack-check`	Generate code to verify that you do not go beyond the boundary of the stack.

Pretty much only `-g` is important.

For the other run-time checks, significant overhead in run time may be observed.

Other compiler options

GCC compiler option	Meaning
`-c`	Compile without linking
`-o`	Name of output file.
`-I`	Search path for included header files (there are predefined system paths)
`-L`	Search path for libraries
`-l`	Library name to link in
`-D` ***<symbol>***	Define preprocessor symbol ***<symbol>*** during compilation
`-E`	Output preprocessed source file
`-S`	Output assembly from compilation
`-fPIC`	Compile **P**osition **I**ndependent **C**ode (necessary for shared objects)
`-fopenmp`	Process OpenMP directives
`-p`	Generate profiling information during run time for profiling analysis tools (e.g. `gprof`)
`-ftest-coverage`	Generate coverage information during run time for coverage analysis tools (e.g. `gcov`)

Object code & Binary output

- Compiling a source file `source.F90` (e.g. –c) produces an object file `source.o`
  - Object files are *relocatable* machine code.
  - Typical object file format for linux is ELF.
  - Cannot view object files in text editors
- ELF files contain
  - Program header table describing 0 or more segments
    - Contains run-time information
  - Section header table describing 0 or more sections
    - Contains link-time information
  - Data referred to by segments and sections
- How can you inspect ELF files/object code/object files?

Snippet of object file from program on slide 15 in vi

```
^?ELF^B^A^A^@^@^@^@^@^@^@^@^@^A^@>^@^A^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@¸^B^@^@^@^@^@^@^
@^@^@^@@^@^@^@^@^@@^@^M^@
^@UH<89>åH<81>ìà^A^@^@HÇ<85>(þÿÿ^@^@^@^@Ç<85>0
þÿÿ^E^@^@^@HÇ<85>hþÿÿ^@^@^@^@Ç<85>pþÿÿ^H^@^@^@
Ç<85> þÿÿ^@^P^@^@Ç<85>$þÿÿ^@^@^@^@H<8d><85>
þÿÿH<89>Çè^@^@^@^@H<8d><85>
þÿÿ°^X^@^@^@¾^@^@^@^@H<89>Çè^@^@^@^@H<8d><85>
þÿÿ°^D^@^@^@¾^@^@^@^@H<89>Çè^@^@^@^@H<8d><85>
þÿÿ°^\^@^@^@¾^@^@^@^@H<89>Çè^@^@^@^@H<8d><85>
þÿÿH<89>Çè^@^@^@^@ÉÃUH<89>åH<83>ì^P<89>}üH<89>
uðH<8b>Uð<8b>EüH<89>Ö<89>Çè^@^@^@^@¾^@^@^@^@¿^
H^@^@^@è^@^@^@^@è^Xÿÿÿ¸^@^@^@^@ÉÃ^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@hello.F90^@(a,i2,a)File:
"hello.F90", line ^@^@^D^@^@^@ was compiled
with
gfortran!^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@D^@^@^@ÿ^A^@^@^@^@^@^@^@^@^@^@^@^@^@^A^@^
@^@^@^@^@^@^A^@^@^@^@GCC: (Ubuntu/Linaro
4.6.4-1ubuntu1~12.04)
4.6.4^@^@^@^@^@^@^@^@^T^@^@^@^@^@^@^@^AzR^@^Ax
^P^A^[^L^G^H<90>^A^@^@^\^@^@^@^\^@^@^@^@^@^@^@
´^@^@^@^@A^N^P<86>^BC^M^F^B¯^L^G^H^@^@^\^@^@^@
<^@^@^@^@^@^@^@;^@^@^@^@A^N^P<86>^BC^M^Fv^L^G^
H^@^@^@^@.symtab^@.strtab^@.shstrtab^@.rela.te
xt^@.data^@.bss^@.rodata^@.comment^@.note.
```

Inspecting Object Files

readelf -a source.o

```
Symbol table '.symtab' contains 18 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS hello.F90
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     6: 0000000000000000   180 FUNC    LOCAL  DEFAULT    1 MAIN__
     7: 0000000000000060    32 OBJECT  LOCAL  DEFAULT    5 options.1.1538
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT    7
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT    8
    10: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _gfortran_st_write
    12: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _gfortran_transfer_charac
    13: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _gfortran_transfer_intege
    14: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _gfortran_st_write_done
    15: 00000000000000b4    59 FUNC    GLOBAL DEFAULT    1 main
    16: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _gfortran_set_args
    17: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _gfortran_set_options
```

nm source.o

```
0000000000000000 t MAIN__
                 U _gfortran_set_args
                 U _gfortran_set_options
                 U _gfortran_st_write
                 U _gfortran_st_write_done
                 U _gfortran_transfer_character_write
                 U _gfortran_transfer_integer_write
00000000000000b4 T main
0000000000000060 r options.1.1538
```

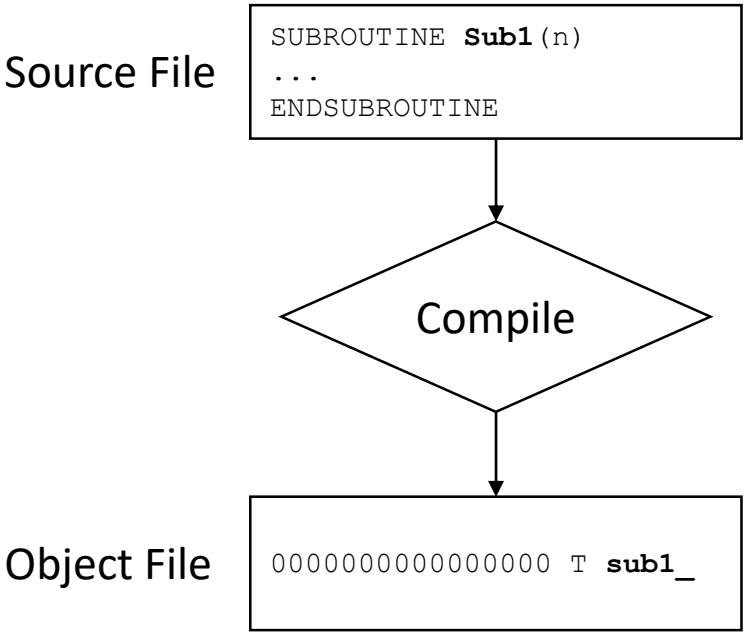
Wait… Why do I need to know what's in my object files?

97% of the time you don't need to know. However, this can be useful in resolving link errors and multi-language programs

Name Mangling (Fortran)

Fortran

- Binary symbol name is different from high level programming language name
- Variants:
  - Lower, Lower_, Lower__
  - Upper, Upper_, Upper__
- Used to be critical for calling Fortran from C.
  - Now the Fortran standard provides features that give programmer more control over name mangling.



This example is “Lower_”

```
SUBROUTINE Sub1(n) BIND(C,NAME="Sub1")
...
ENDSUBROUTINE
```

```
0000000000000000 T Sub1
```

Name Mangling (C++)

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

using namespace std;

template <typename T>
inline T const& Max (T const& a, T const& b)
{
    return a < b ? b:a;
}

int main ()
{
    int i = 39; int j = 20;
    cout << "Max(i, j): " << Max(i, j) << endl;

    double f1 = 13.5; double f2 = 20.7;
    cout << "Max(f1, f2): " << Max(f1, f2) << endl;

    string s1 = "Hello"; string s2 = "World";
    cout << "Max(s1, s2): " << Max(s1, s2) << endl;
    return 0;
}
```

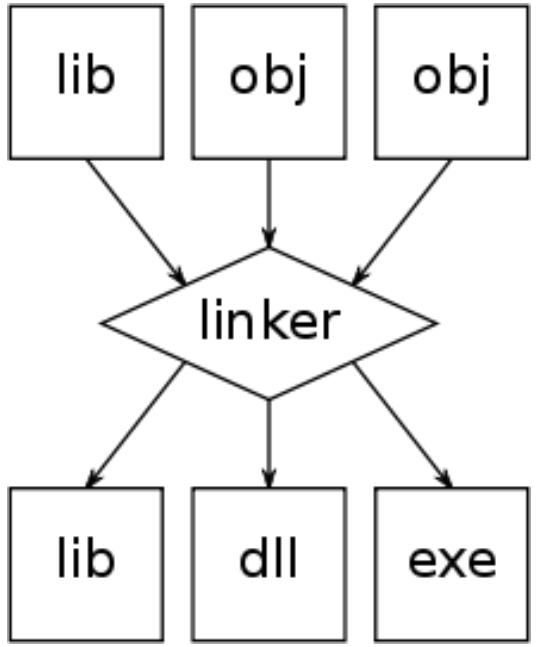
- Shows up with templating
  - Have to produce different binary code for each templated type
  - Necessary for linking

```
00000000000001c6 t _Z41__static_initialization_and_destruction_0ii
                 U _ZNKSs7compareERKSs
                 U _ZNSaIcEC1Ev
                 U _ZNSaIcED1Ev
                 U _ZNSolsEPFRSoS_E
                 U _ZNSolsEd
                 U _ZNSolsEi
                 U _ZNSsC1EPKcRKSaIcE
                 U _ZNSt8ios_base4InitC1Ev
                 U _ZNSt8ios_base4InitD1Ev
                 U _ZSt4cout
                 U _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
0000000000000000 b _ZStL8__ioinit
                 U _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
0000000000000000 W _ZStltIcSt11char_traitsIcESaIcEEbRKSbIT_T0_T1_ES8_
```

Example from: https://www.tutorialspoint.com/cplusplus/cpp_templates.htm

What is Linking?

- Linking is the process of combining the various objects and libraries output from compilation into a single executable (or library or object).
  - May also include binaries (e.g. libraries) already installed on the system
- Sometimes performed by external program called by compiler (e.g. `ld`)
- Sometimes part of compiler (depends on the vendor)
- Key steps in linking are
  - *Resolving external symbols* that the linker uses to figure out how to piece together the executable
  - *Relocating load addresses* of various program parts (e.g. function addresses and variable addresses) to reflect the assigned addresses in the whole program.
- Linking can produce targets thatare ***statically*** linked or ***dynamically*** linked



Dynamic Linking vs. Static Linking

Static Linking

- Probably what you think of when you think "linking"
- Copy all binary code from all libraries and objects then package into a single executable image
  - Usually results in larger executable file sizes
- A little more portable since all the binary code is packaged together
- Requires all libraries that are linked to be static libraries (e.g. `lib<name>.a`)
- Sometimes a requirement on large clusters
  - Compute nodes and login nodes are different

Dynamic Linking

- Symbol resolution is delayed until executable is run
  - Executable code has undefined symbols
  - Requires all libraries that are linked to be dynamic libraries (e.g. `lib<name>.so`)
- Some advantages
  - For system libraries used by every program, no need to copy into every executable (e.g. `libc`)
  - If there is a bug in a library, and a new version of the library that fixes the bug is installed, all programs benefit.
    - Statically linked executables need to be re-linked
- Some disadvantages
  - Libraries that are updated that break backwards compatibility, might break your executable.
  - Need to have the correct environment.
  - Not necessarily portable, OS and environment need to consistent.

What link errors look like

Static Link Error

```
PROGRAM hello_main

WRITE(*,*) "Hello World!"
CALL some_undefined_routine()

ENDPROGRAM
```

```
$ gfortran -c hello.F90
$ gfortran hello.o -o hello.exe
hello.o: In function `MAIN__':
hello.F90:(.text+0x71): undefined reference to
`some_undefined_routine_'
collect2: ld returned 1 exit status
```

Dynamic Link Error

```
$ ./some_mpi_program.exe
./mpi_program.exe: error while loading shared
libraries: libmpi.so: cannot open shared object file:
No such file or directory
```

When you attempted to run the executable, The OS could not find the library using the information in your current environment

The command given to the linker did not include the library or object (or the correct path to the library or object) that defines the named symbol.

How to trouble shoot link errors

Static Link Error

- Most likely you are missing the correct entries on the following options passed to the linker:
  - `-l<library_name_with_symbol>`
  - `-L<path_to_library>`
- Could also be a typo in your source code
- Generally easy to resolve
  - If you know where the missing library is located.
- Can be difficult if you have no idea why the symbol is trying to be linked (where is it used, where is it defined
  - More likely to happen when you are linking third party libraries

Dynamic Link Error

- Most likely your environment is not the same as when you compiled
  - Check your environment
  - Environment variable is `LD_LIBRARY_PATH`
- Useful command: `ldd`
  - Shows you *exactly* what libraries are dynamically linked to your executable

```
$ ldd ./some_mpi_program.exe
        linux-vdso.so.1 =>  (0x00007ffcf2be8000)
        libmpi.so => not found
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
(0x00007f4d12878000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f4d12c38000)
```

Dynamic ***Loading***: Linking in code at run time

- Start your executable *then load a library* into memory.
  - Use case is “plugins”. An example might be linking proprietary correlations for material properties.
  - Can be done interactively. User could specify library name and function name as an input.
  - Challenging to list “available symbols” in library, although this can be done. But basically need to know what routine you want to call
- In Linux requires “`dl`” library.

```
#include <dlfcn.h>

void* sdl_library = dlopen("libSDL.so", RTLD_LAZY);
if (sdl_library == NULL) {
   // report error ...
} else {
   void* initializer = dlsym(sdl_library,"SDL_Init"); //extract library contents
   if (initializer == NULL) {
      // report error ...
   } else {
      // cast initializer to its proper type and use
      typedef void (*sdl_init_function_type)(void);
      sdl_init_function_type init_func = (sdl_init_function_type) initializer;
   }
}
```

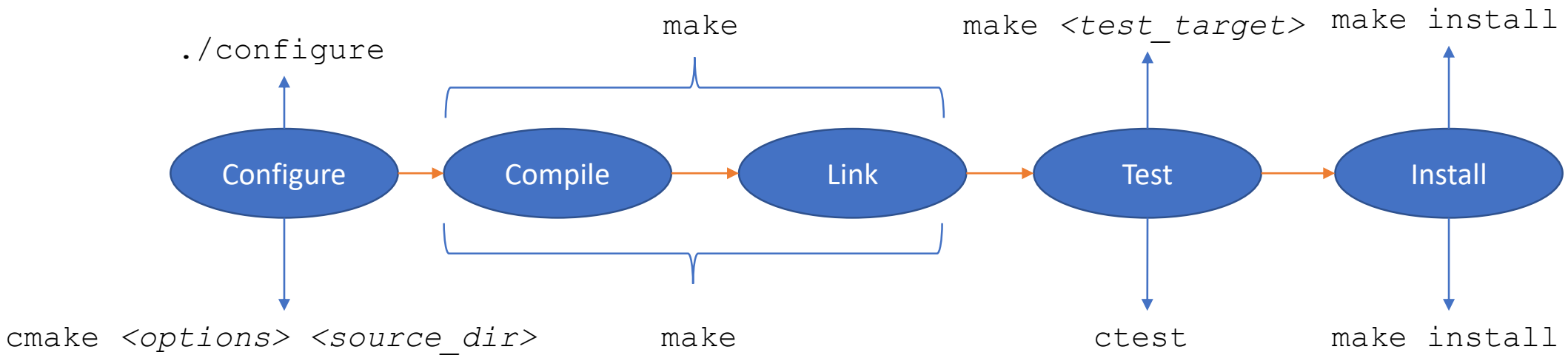
https://en.wikipedia.org/wiki/Dynamic_loading

Multi-language Programs

- **The key is linking!**
  - Linker does not care what high-level language produced your object code. It could have been generated from Fortran or C or C++.
    - Linker just has to resolve symbols in object code.
  - Well one subtlety, you must have a compatible application binary interface (ABI)
    - Usually not an issue unless you are compiling on one machine and linking on another.
- If a programming language or environment (e.g. Python) supports linking of C interfaces than you can link any code that provides a C interface
  - Most languages support C interfaces (because they were probably implemented in C or the compiler was)
  - Therefore, *C is the de-factor language of interoperability*.
- By "C interface" I mean a binary symbol that is producible from the C high-level language and a C compiler.

Summary: Using the Toolchain

Autotools



CMake