Installing software on HPC

Erin Shaw and Cesar Sul

Advanced Cyberinfrastructure Research and Education Facilitation

USC Center for High-Performance Computing

Welcome HPC Workshop - Setup

- Sign in at goo.gl/Qan5Kg

- Best to have X display forwarding enabled
  - On Mac, download and install XQuartz from www.quartz.org
  - On PC, check that X forwarding is enabled within your ssh client

- Log in to hpc-login2 or hpc-login3 with -X or –Y option
  - Go to your /staging directory and create workshop space

```
$ cd /staging/<project>/<user>
$ mkdir workshop
$ cd workshop
```

Welcome HPC Workshop - Setup

- Slides and files used are here:
  - /home/rcf-proj/workshop/handouts/HPCInstallingSoftware*.pdf
  - /home/rcf-proj/workshop/installing-software/

- To copy slides to your laptop:
  - On your laptop, open an xterminal and go to, e.g., your Desktop
  - Type the following. This is all one command and there is a ”dot” at the end!

```
scp <your_NetId>@hpc-login3.usc.edu:/home/rcf-proj/
   workshop/handouts/HPCInstallingSoftware*.pdf <space> .
```

#”dot” is a Unix symbol that resolves to/is replaced by the path to your ”current directory”.

HPC Facilitation

- Request assistance
  - Email hpc@usc.edu (email again!)
  - Drop-in to office hours
    - *UPC: every Tuesday, 2:30p (LVL 3M)*
    - *HSC: 1st (NML 203) & 3rd Thursdays (SSB106)*
- Learn more!
  - Webpages online at https://hpcc.usc.edu
  - Request a consultation (Getting Help => Office Hours & Consultations)
  - Attend a workshop

Outline

Introduction
Setting your environment
Downloading files
Precompiled binary
Compiling source code
Compiling source code with dependencies
HPC compilers and libraries
Installing Python packages
Installing R libraries
Using Singularity
Licensed software

Introduction

- You are encouraged to install all software you need in your HPC project directory!
  - If you are here today, we assume that's what you want to do
- Installing and running software typically requires knowledge of the computer it will run on, at some level
  - That's why there are different downloads for different machines

Introduction

- For example, HPC computers feature:
  - Intel 64 bit processors (x86_64)
  - CentOS 7.x operating systems
    - *CentOS is a derivative of Red Hat Enterprise Linux*
    - *Linux is a variant of Unix for personal computers*

- But not all HPC “nodes” are the same:
  - Processor type and speed
  - Advanced hardware support
  - GPUs (graphical processing units)
- These ”node features” are important to software applications

Introduction

- Installing software can be quick and painless
  - With precompiled binaries for your specific operating system, it can be as easy as unzipping a file
- … or neither quick nor painless
  - If you have to compile the software yourself using compilers, linkers, Makefiles, external libraries, etc.
- (or even worse...)
  - If it's from an academic lab from 1999 and requires old versions of multiple libraries which have multiple dependencies!

Introduction

- Generally speaking, software can be installed globally or locally
  - On your laptop, you are the system administrator
  - On HPC, you are not the system administrator
- Globally means system-wide
  - Software is installed to system locations like /usr/bin or /usr/local
  - Global installs require root privileges
- System-wide installations will not work on HPC
  - Only systems administrators have root privileges on HPC
  - E.g., ”yum install” and “apt install” will not work

Introduction

- HPC users must perform local, or “user”, installs
  - Software installed to 'local' folders
    - */home/rcf-proj/<project>/<user>*
    - *Always work in project directory!*
  - Requires write privileges, which you have in your own directories
  - Software will be accessible by you, even on compute nodes
- It is not always obvious how to perform a user install
  - Depends on software
  - You may have to google a bit
- Occasionally it is not possible and HPC admins will install

Outline

Introduction
Setting your environment
Downloading files
Precompiled binary
Compiling source code
Compiling source code with dependencies
HPC compilers and libraries
Installing Python packages
Installing R libraries
Using Singularity
Licensed software

Setting your environment

- Environment variables!
  - Are *named, string variables*, e.g.
    - *HOME=/home/rcf-proj/erinshaw, USER=erinshaw*
    - *mydata=/home/rcf-proj/erinshaw/ess/data*
  - By convention, system-set environment variables are all UPPERCASE and user-set environment variables are at least partially lowercase
  - The value is accessed by preceding the variable with a ”$”
    - *Type: echo $HOME, echo $SHELL, echo $mydata*
    - *Type: env (to print all envs)*
- Environment variables can be set by the system, programs, and users; and are available system-wide, on command line, and inside code

Setting your environment

- $PATH contains a list of directories (locations)
  - The locations are searched sequentially by the shell, when you type a command, to find the specified command
    - *The first match will be executed*
    - *When there is no match, the shell will produce an error*

```
$ python
$ which python
/usr/bin/python
$ echo $PATH
/usr/lib64/qt-
3.3/bin:/opt/moab/bin:/opt/mam/bin:/usr/local/bin:/usr/bin:/usr/
local/sbin:/usr/sbin
```

Setting your environment

- Changing the $PATH variable, by 'sourcing' (running/executing) HPC's setup.sh script changes, will change which version of python is executed

```
$ source /usr/usc/python/2.7.8/setup.sh
$ echo $PATH
/usr/usc/python/2.7.8/bin:/usr/lib64/qt-
3.3/bin:/opt/moab/bin:/opt/mam/bin:/usr/local/bin:/usr/bin:/usr/
local/sbin:/usr/sbin
$ which python
/usr/usc/python/2.7.8/bin/python
```

Setting your environment

- Don't add environment variables to shell files like .bashrc, .bash_profile, .login, or .profile!
  - The wrong environment variables can lead to conflicting settings.
  - You'll need to be able to 'turn off' environment variables/settings.
  - This happens all the time!
- Your .bashrc and .bash_profile are for run commands and configuration settings.
  - Add aliases for commands you like to use
    - *alias rm='rm –i'*
    - *alias cdp='cd /home/rcf-proj/tt/tommy*
  - Or change your prompt

Setting your environment

- Best to create environment setup scripts that initialize the settings you need
  - e.g., env.sh, setup.sh (we will be creating these)
  - Especially if you are doing work for multiple projects
  - Simple settings can go right into your SLURM scripts
- Enables you to keep track of how software is installed
  - You might have to re-run or reconfigure environment
  - Environment settings could influence your results
  - You want your results to be reproducible!

Outline

Introduction
Setting your environment
Downloading files
Precompiled binary
Compiling source code
Compiling source code with dependencies
HPC compilers and libraries
Installing Python packages
Installing R libraries
Using Singularity
Licensed software

Downloading files

- Various ways to acquire code from internet
  - Can only do this from a head node – use hpc-transfer.usc.edu
- Code often comes in form of a tarball (tar)
  - Unix archive file, much like a zip file
  - It is usually compressed to .gz or .bz
  - Use tar xvf <filename> to extract files, or “untar”
- Sources include
  - Web pages
  - Github
  - Source Forge

Outline

Introduction
Setting your environment
Downloading files
Precompiled binary
Compiling source code
Compiling source code with dependencies
HPC compilers and libraries
Installing Python packages
Installing R libraries
Using Singularity
Licensed software

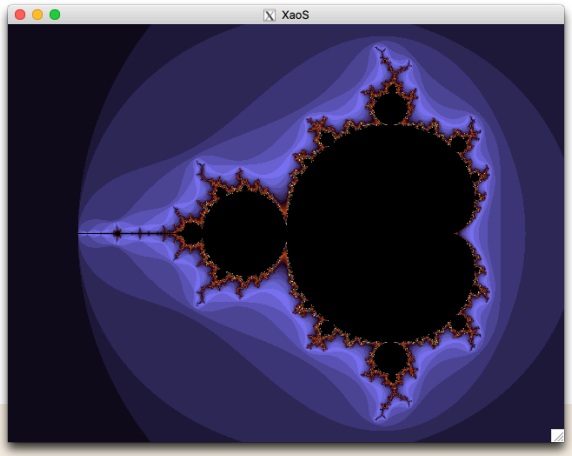
Precompiled binary (Setup)

- Best to have X display forwarding enabled
  - On Mac, download and install XQuartz from www.quartz.org
  - On PC, check that X forwarding is enabled within your ssh client

- Log in to hpc-login2 or hpc-login3 with -X or –Y option
  - Go to your /staging directory and create workshop space

```
$ cd /staging/<project>/<user>
$ mkdir workshop
$ cd workshop
```

Precompiled binary



Example: XaoS

```
#Copy tarball
$ cp /home/rcf-proj/workshop/installing-software/xaos-3.6-bin.tar.gz .

#Extract files
$ tar xvf xaos-3.6-bin.tar.gz

#Set your environment (adds a new location to your path)
$ export PATH=/staging/<project>/<user>/workshop/xaos/bin:${PATH}

#Test installation (requires that X11 forwarding be enabled)
$ xaos
```

Outline

Introduction
Setting your environment
Downloading files
Precompiled binary
Compiling source code
Compiling source code with dependencies
HPC compilers and libraries
Installing Python packages
Installing R libraries
Using Singularity
Licensed software

~~Compiling~~ Interpreting

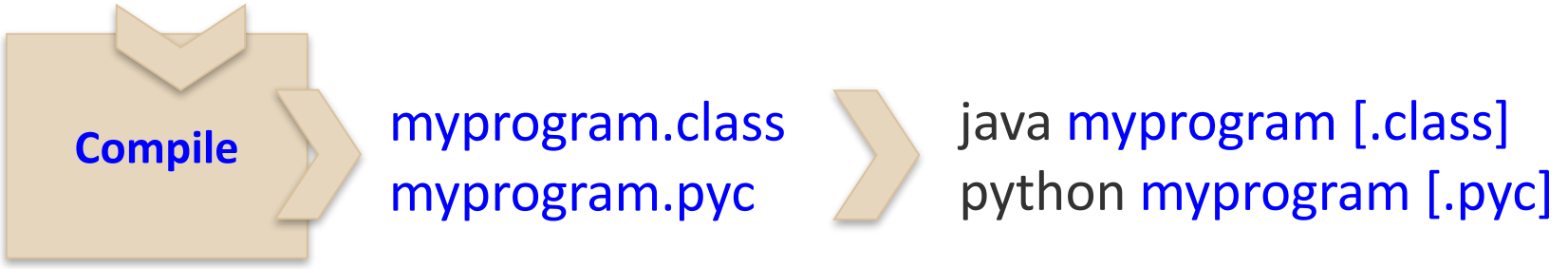
- Scripts can be edited, interpreted, and run
  - Text is run line-by-line -- no formal compiling is done
  - The interpreters themselves must be compiled for each OS

myscript.sh	bash myscript.sh
myscript.pl	perl myscript.pl
myscript.m	matlab –r myscript.m
myscript.R	R myscript.R

Compiling to bytecode

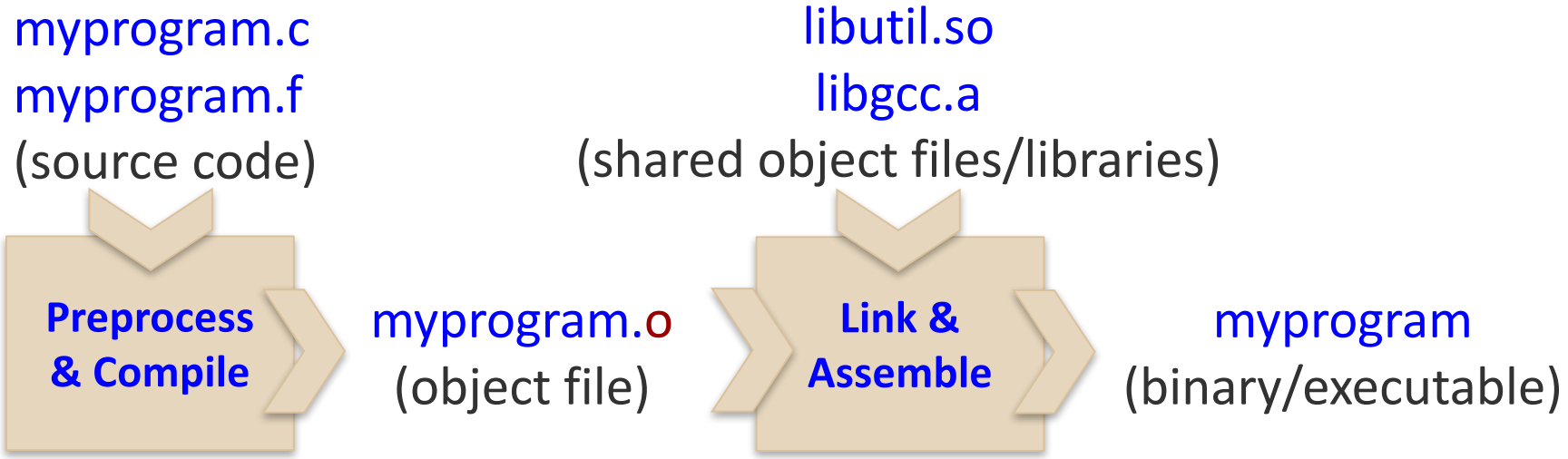
- Code can be compiled to “byte code” and run by a virtual machine (the VMs themselves are compiled for each OS)

myprogram.java
myprogram.py



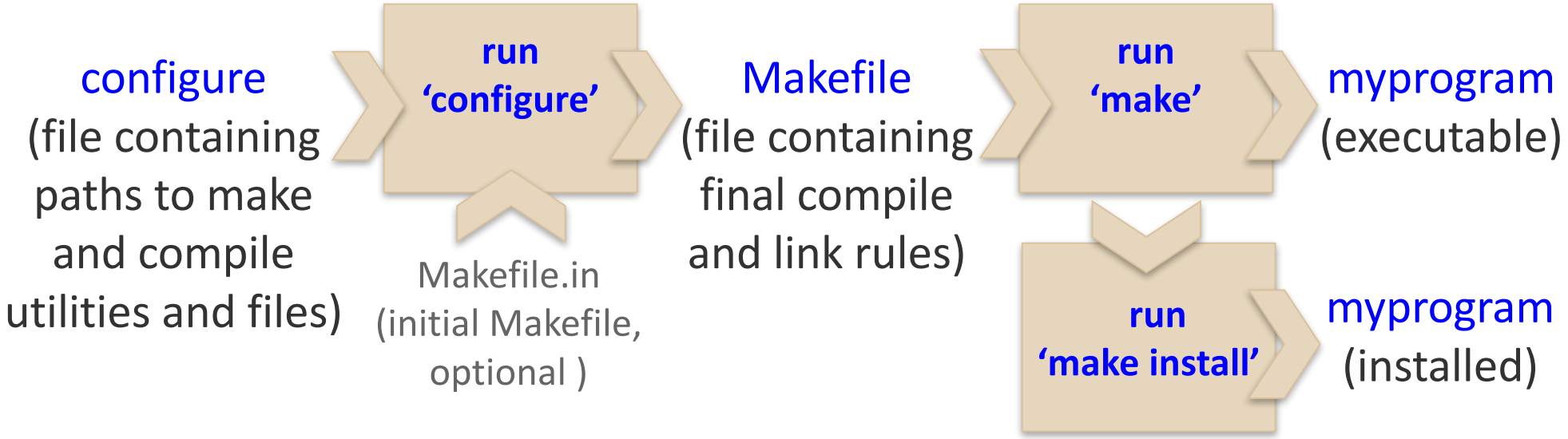
Compiling to machine code

- C/C++ and Fortran programs are compiled and assembled
  - The output is machine-executable object code



With configure/make

- Applications may have hundreds of files and so manually typing compile and link commands is not feasible
- Software build utilities, like gnu's autotools, are used for this



HPC Compilers

- Environment variables are heavily used in makefiles, e.g.:
  - CCFLAGS Flags to pass the C Compiler
  - CPPFLAGS Where the C PreProcessor can find include (.h) files
  - LDFLAGS The Library Directory paths and which libraries to include (.so, .a files)
- A typical compile command for C code might be:

  gcc ${CCFLAGS} source.c ${CPPFLAGS} ${LDFLAGS} –o myprogram

  where the environment variables are pre-defined:

  CCFLAGS='-g -Wall -O3'

  CPPFLAGS='-I/path/to/include'

  LDFLAGS='-L/path/to/lib -lgsl -lgslcblas –lm'

Compiling source code

Example: XaoS

```
#Download tarball (normally this would require a 'wget')
$ cp /home/rcf-proj/workshop/installing-software/xaos-
3.6.tar.gz .
$ tar xvf xaos-3.6.tar.gz
$ cd xaos-3.6

#Read install file
$ less INSTALL

#Run configure script
$ ./configure --prefix=/staging/<prj>/<usr>/workshop/XaoS

#Run makefile
$ make
$ make install
```

Compiling source code

To set your environment, create this script and 'source' it.

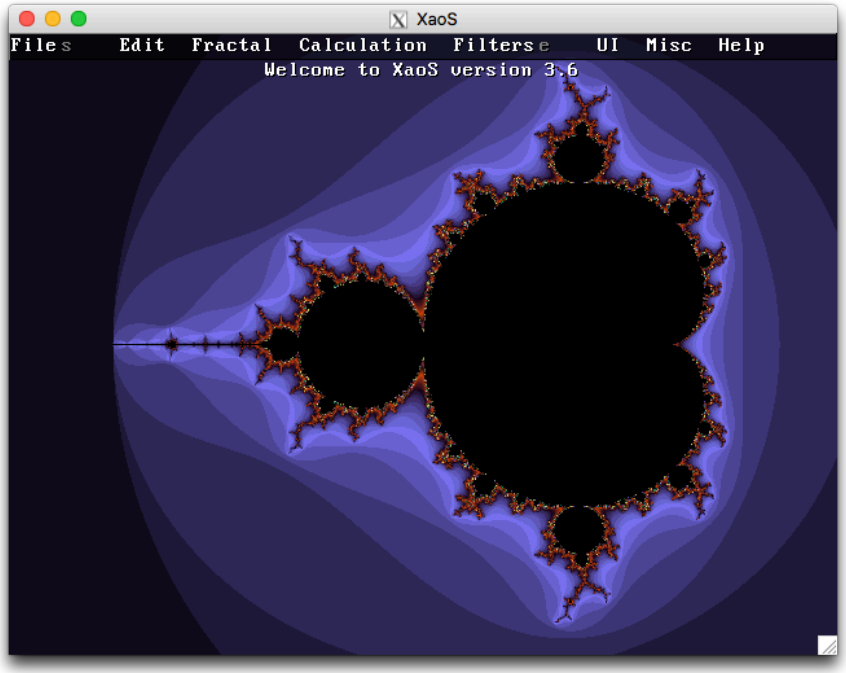
```
#Display the contents of the script
$ cat env.sh
#!/bin/bash
PATH=/staging/<prj>/<usr>/workshop/XaoS/bin:${PATH}
MANPATH=/staging/<prj>/<usr>/workshop/Xaos/share/man:${MANPATH
export PATH MANPATH

#source (run) the script
$ source env.sh
```

Compiling source code

Test installation

```
#Run XaoS (Need X11 forwarding enabled)
$ xaos
```



Click on part of the image to zoom in

Troubleshooting

Error	What it means	How to fix it
Permission denied Read-only file system	You are trying to install somewhere you don't have access to.	Install to a location where you have write permissions. e.g., use: ./configure --prefix=path/to/dir
Library (.so,.a) file not found Undefined reference to… Cannot find shared object file	Your program needs to link to libraries that the installer can't find.	Set your library search path. LDFLAGS=-L/path/to/lib ./configure
Header (.h) files not found Undefined reference to...	Your program needs to include header files that the installer can't find.	Set your Include file path. CPPFLAGS= -I/path/to/include ./configure

Outline

Introduction
Setting your environment
Downloading files
Precompiled binary
Compiling source code
Compiling source code with dependencies
HPC compilers and libraries
Installing Python packages
Installing R libraries
Using Singularity
Licensed software

Compiling source with dependencies

- Example: NetCDF

Requires HDF5, which is already in /usr/usc/hdf5/

```
#Download tarball
wget https://github.com/Unidata/netcdf-c/archive/v4.4.1.tar.gz

#Untar and read install directions
$ tar xvf v4.4.1.tar.gz
$ cd v4.4.1.tar.gz
$ less INSTALL

#Edit compile,include,library flags: point to HPC locations
$ CC=mpicc
$ CPPFLAGS=-I/usr/usc/hdf5/1.8.18/parallel/include
$ LDFLAGS=-L/usr/usc/hdf5/1.8.18/parallel/lib

#Run configure script
$./configure --prefix=/path/to/install/dir'
```

Compiling source with dependencies

```
#Run makefile
$ make
$ make install

#Create an environment script
$ nano env.sh

#!/bin/bash

source /usr/usc/hdf5/default/setup.sh

PATH=/path/to/install/dir/bin:${PATH}
LD_LIBRARY_PATH=/path/to/install/dir/lib:${LD_LIBRARY_PATH}

export PATH LD_LIBRARY_PATH
```

Outline

Introduction
Setting your environment
Downloading files
Precompiled binary
Compiling source code
Compiling source code with dependencies
HPC compilers and libraries
Installing Python packages
Installing R libraries
Using Singularity
Licensed software

HPC Compilers

- HPC supports open source and commercial compilers for the C, C++ and Fortran programming languages

Vendor	C/C++	Fortran	Path
GNU (open source)	gcc/g++	gfortran	/usr/bin /usr/usc/gnu/gcc
Intel*	icc/icpc	ifort	/usr/usc/intel
PGI*	pgcc/pgc++	pgfortran	/usr/usc/pgi
OpenMPI**	mpicc/mpicxx	mpifort	/usr/usc/openmpi

*Commercial compilers

**MPI compilers are "wrappers" that can use GNU/Intel/PGI

Compiling on HPC

- HPC has the GNU, Portland Group, and Intel compilers, and MPI compiler wrappers installed in the **/usr/usc** directory.
  - To use one of these compilers, you must source the appropriate setup.sh file each time you log in.

    source /usr/usc/pgi/default/setup.csh
- GNU compilers come installed on OS and are, by default, in your path

```
$ which gfortran
/usr/bin/gfortran
$ gfortran --version
GNU Fortran (GCC) 4.8.5 20150623
$ which f95
/usr/bin/f95
$ f95 --version
GNU Fortran (GCC) 4.8.5 20150623
```

```
$ which gcc
/usr/bin/gcc
$ gcc --version
gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-4)

$ which g++
/usr/bin/g++
$ g++ --version
g++ (GCC) 4.8.5 20150623 (Red Hat 4.8.5-4)
```

Compiling on HPC

- We also have versions of gnu compilers under **/usr/usc/gnu/gcc**
  - Good if you need reproducibility
  - You must 'source' the appropriate setup.sh file

- Intel compilers are available under **/usr/usc/intel**
  - Choose the version of the compiler you want to use and source its setup file to access it.

    $ source /usr/usc/intel/16.0/setup.sh

    $ which ifort
    /usr/usc/intel/16.0/compilers_and_libraries_2016.0.109/linux/bin/intel64/ifort

- Java is available under **/usr/usc/java** and **/usr/usc/jdk**

HPC Libraries

- Scientific software usually requires low level math libraries
  - BLAS (Basic Linear Algebra Subprograms) is the de-facto standard for linear algebra libraries for C and Fortran
  - Set of low-level routines for performing common linear algebra operations, such as vector addition and matrix multiplication
- HPC maintains optimized BLAS libraries for user installs
  - OpenBLAS, Intel MKL – Math Kernel Library
  - source /usr/usc/intel/default/setup.sh
- HPC-installed software in /usr/usc/ is compiled with OpenBLAS
  - NumPy, Matlab, R

Viewing linked libraries

```
$ cd xoas/bin
$ ldd  xaos
linux-vdso.so.1 =>  (0x00007ffce1fea000)
libpng15.so.15 => /lib64/libpng15.so.15 (0x00007f39846b3000)
libz.so.1 => /lib64/libz.so.1 (0x00007f398449d000)
libm.so.6 => /lib64/libm.so.6 (0x00007f398419b000)
libXext.so.6 => /lib64/libXext.so.6 (0x00007f3983f89000)
libX11.so.6 => /lib64/libX11.so.6 (0x00007f3983c4b000)
libgsl.so.0 => /lib64/libgsl.so.0 (0x00007f3983822000)
libgslcblas.so.0 => /lib64/libgslcblas.so.0 (0x00007f39835e5000)
libc.so.6 => /lib64/libc.so.6 (0x00007f3983218000)
/lib64/ld-linux-x86-64.so.2 (0x00007f39848de000)
libxcb.so.1 => /lib64/libxcb.so.1 (0x00007f3982ff0000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007f3982dec000)
libsatlas.so.3 => /usr/lib64/atlas/libsatlas.so.3 (0x00007f398219f000)
libXau.so.6 => /lib64/libXau.so.6 (0x00007f3981f9b000)
libgfortran.so.3 => /lib64/libgfortran.so.3 (0x00007f3981c79000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f3981a5d000)
libquadmath.so.0 => /lib64/libquadmath.so.0 (0x00007f3981821000)
```

Finding libraries on HPC

- Use the RPM package manager to locate programs or libraries that have been installed on a node.
  - $ man rpm
  - Note: The command, **rpm**, will not find programs and libraries installed under /usr/usc.
- Packages available on head nodes might not be available on compute nodes.
- Example: My application requires the atlas library. Is it available on HPC?
  - What exactly does your application require? (Maybe the BLAS library?)
  - First, check software available under /usr/usr/
  - Next, check using rpm

Finding libraries on HPC

$ rpm -qa | fgrep atlas

atlas-3.8.4-2.el6.x86_64

atlas-devel-3.8.4-2.el6.x86_64

$ rpm -q --info atlas

Name : atlas Version : 3.8.4

Release : 2.el6

Build Date: Tue 20 Mar 2012 07:03:13 PM PDT

:

URL : http://math-atlas.sourceforge.net/

Summary : Automatically Tuned Linear Algebra Software Description : The ATLAS (Automatically Tuned Linear Algebra Software) project is an

:

Finding libraries on HPC

```
$ rpm -q --filesbypkg atlas
atlas      /etc/ld.so.conf.d/atlas-x86_64.conf
atlas      /usr/lib64/atlas
atlas      /usr/lib64/atlas/libatlas.so.3
:
$ rpm -q --filesbypkg atlas | fgrep blas
atlas      /usr/lib64/atlas/libcblas.so.3
atlas      /usr/lib64/atlas/libcblas.so.3.0
atlas      /usr/lib64/atlas/libf77blas.so.3
:
```

Outline

Introduction
Setting your environment
Downloading files
Precompiled binary
Compiling source code
Compiling source code with dependencies
HPC compilers and libraries
Installing Python packages
Installing R libraries
Using Singularity
Licensed software

Installing Python packages

- Don't forget to source the version of python you want to use

```
$ source /usr/usc/python/2.7.8/setup.sh
```

- To check what packages are available use the command

```
$ pip freeze
```

- Python's default installation location is ${HOME}/.local
  - You have only 1GB disk quota and 100,000 file quota in your home directory and a python package install can easily be larger
  - Solution is to create a symbolic link, or 'symlink', which is a pointer (named .local) to another directory with more disk space

Initial Python package

- Create a directory outside of your home directory, for storing python packages, then create a symbolic link to it called .local

```
$ cd ~
$ mkdir -p /home/rcf-proj/<proj>/<user>/python_packages
$ ln -s /home/rcf-proj/<proj>/<user>/python_packages .local
```

- Install package (bash shell)

```
$ source /usr/usc/python/<version>/setup.sh
$ pip install <package_name> --user
```

- Sometimes you'll need to install the latest version of a package that is already installed

```
$ pip freeze
$ pip install <package_name> --upgrade --user
```

Dependencies for Python packages

- Some packages are Python wrappers for C/C++ libraries
- The installer needs to know where these libraries are
- The h5py package is one example

```
$ HDF5_DIR=/path/to/hdf5
$ HDF5_VERSION=X.Y.Z
$ CC="mpicc"
$ pip install h5py
```

- You might have to download the package tarball and edit some files like setup.py

Outline

Introduction
Setting your environment
Downloading files
Precompiled binary
Compiling source code
Compiling source code with dependencies
HPC compilers and libraries
Installing Python packages
Installing R libraries
Using Singularity
Licensed software

Installing R packages

- R's default installation location is ${HOME}/R
  - You have only 1GB disk quota and 100,000 file quota in your home directory and a python package install can easily be larger
  - Solution is to create a symbolic link, or 'symlink', which is a pointer (named R) to another directory with more disk space
- Create a directory outside of your home directory, for storing python packages, then create a symbolic link to it called "R"

```
$ cd ~
$ mkdir -p /home/rcf-proj/<proj>/<user>/R_packages
$ ln -s /home/rcf-proj/<proj>/<user>/R_packages R
```

**note: there is a space between "R_packages" and "R" on the last line*

Installing R packags

- Source the version of R you want to use and start R

```
$ source /usr/usc/R/default/setup.sh
$ R
```

- Install package syntax (you may have to specify a path)

```
> install.packages('<package_name>')
> install.packages('<package_name>", lib="/path/to/packages")
```

- Then load the library when you want to use

```
> library('<package_name>’)
> library('<package_name>’, lib.loc="/path/to/packages")
```

Installing R packags

- Sometimes, the package you need in not available through CRAN
  - To query the available repositories and inspect available packages:

```
> setRepositories()
> ap <-available_packages ()
> View(available_packages)
```

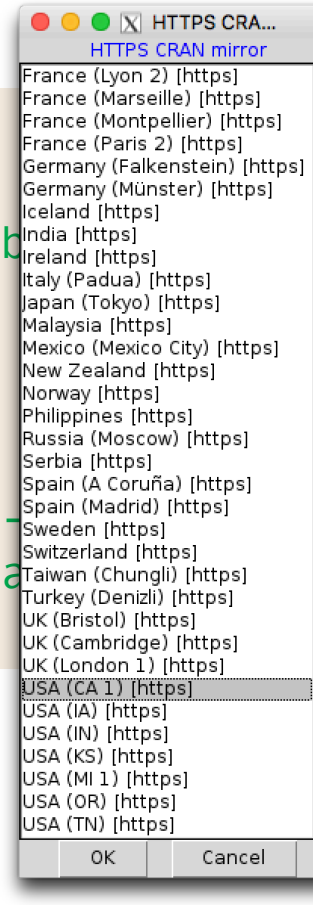
- Now install and add

```
> install.packages('<package_name>')
> library (<package_name>)
```

Installing R packages

- Example

```
> install.packages("gplots")
Warning in install.packages("gplots") :
  'lib = "/auto/usc/R/3.3.1/lib64/R/library"' is not writab
Would you like to use a personal library instead?  (y/n) y
Would you like to create a personal library
~/R/x86_64-pc-linux-gnu-library/3.3
to install packages into?  (y/n) y
--- Please select a CRAN mirror for use in this session ---
also installing the dependencies ‘bitops’, ‘gtools’, ‘gdata
‘caTools’
```



- Displays a list (or window on right if -X enabled)
  - Select USA (CA 1) as download site

Installing R packages

- You can display packages that are installed in the USC versions of R, or that you have installed.

```
> installed.packages(lib.loc="/usr/usc/R/default/lib64/R/library")
> installed.packages(lib.loc="~/R/x86_64-pc-linux-gnu-library/3.3")
```

- It is more efficient to check one package at a time

```
> system.file(package="gplots")
```

- Don't forget to load the library!

```
> library("gplots")
> library("gplots", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.3")
```

Installing R packages

- Display your library paths in R with .libPaths()

```
>.libPaths()
[1] "/auto/rcf-proj/ess/erinshaw/homeR/x86_64-pc-linux-gnu-
library/3.3"
[2] "/auto/usc/R/3.3.1/lib64/R/library"
```

- You can set R's library path
  - By setting R_LIBS_USER in your env.sh or slurm script
  - By creating an .Rprofile file in your home directory and adding a path

```
$ export R_LIBS_USER=~/R/x86_64-pc-linux-gnu-
library/3.3:${R_LIBS_USER}
```

- You can tell R to look in multiple locations

```
> libPaths( c(.libPaths(), "~/R/x86_64-pc-linux-gnu-
library/3.3"))
```

Dependencies for R packages

- Some packages are R wrappers for C/C++ libraries
  - The installer needs to know where these libraries are
  - You might have to download the package tarball and edit some files
- You can set compilation environment variables like LDFLAGS in the file ${HOME}/.R/Makevars

Outline

Introduction
Setting your environment
Downloading files
Precompiled binary
Compiling source code
Compiling source code with dependencies
HPC compilers and libraries
Installing Python packages
Installing R libraries
Using Singularity
Licensed software

Licensed software

Singularity

- For difficult installations
- Singularity provides packaged "computing environments"
- Works best with complex dependency chains
- Compatible with Docker

Singularity

Example: lolcow (fortune|cowsay|lolcat)

```
#Download container image
$ singularity pull shub://GodloveD/lolcow

#Test
$ singularity run lolcow_latest.sif
```

See this page for more ways to interact with a container:
https://sylabs.io/guides/3.0/user-guide/quick_start.html#interact-with-images

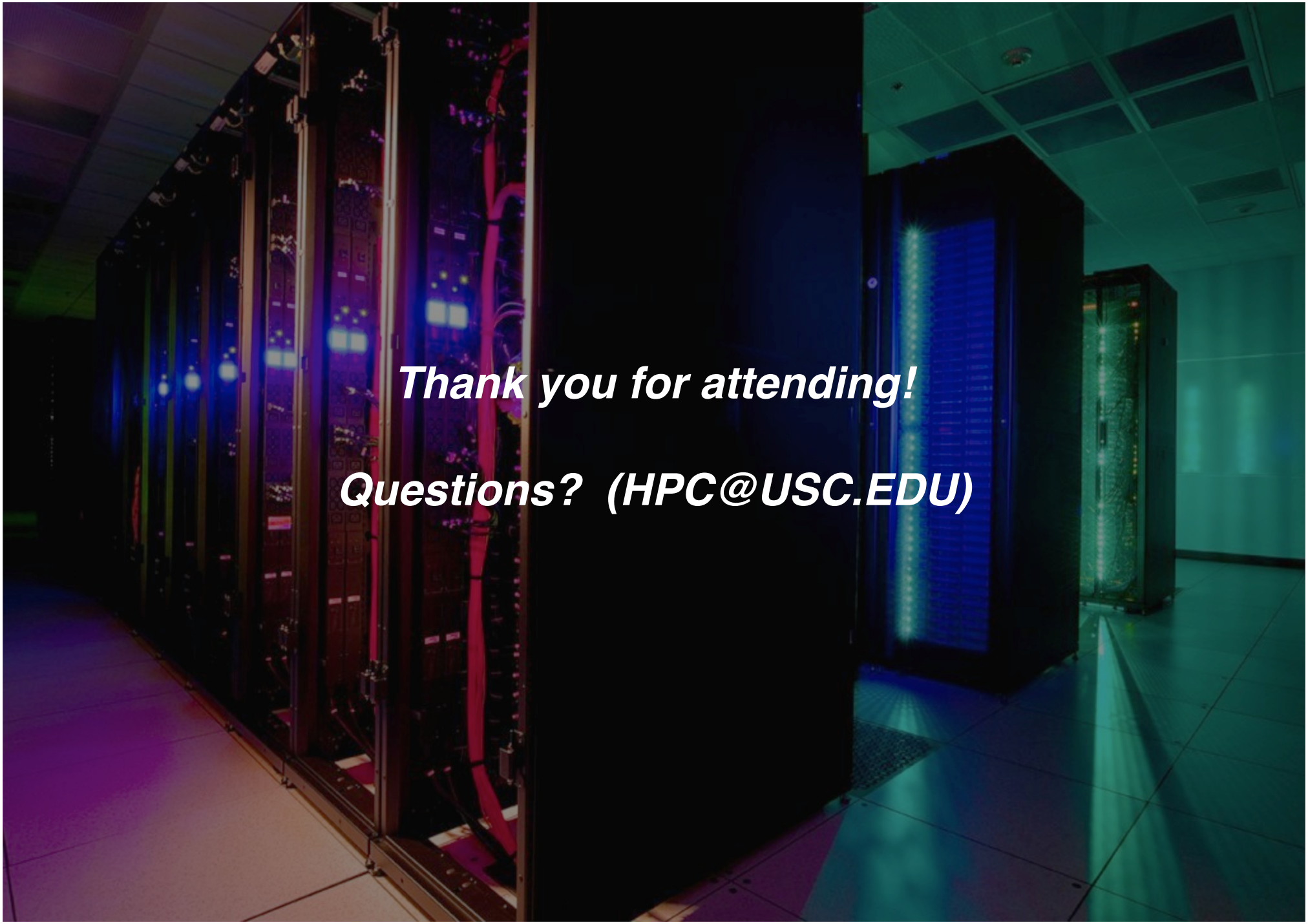
```
[ttroj@hpc-login3 playground]$ singularity run
lolcow_latest.sif
 ______________________________________
/ You will give someone a piece of your \
\ mind, which you can ill afford.       /
 --------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Outline

Introduction
Setting your environment
Downloading files
Precompiled binary
Compiling source code
Compiling source code with dependencies
HPC compilers and libraries
Installing Python packages
Installing R libraries
Using Singularity
Licensed software

Licensed software

- Some of the software in /usr/usc is licensed
  - E.g., MATLAB, which is available USC-wide, and MATLAB Parallel Computing Toolbox, of which we have 1000 “seats”, and SAS, etc.
- HPC will help you set up and install licensed software
  - We recommend you get a floating ‘Flex’ license
  - HPC will host the license server
  - Compute nodes will have access to this server
  - HPC can set restrictions so only you or your user group can use it
- There are situations when licenses must reside outside HPC
  - Mixed use licenses for desktops and HPC
  - HPC can accommodate these situations, too
- Recall: HPC compute nodes live in a private network
  - There is no public internet access


Thank you for attending!
Questions? (HPC@USC.EDU)