Open Science Grid

Backpacking with Code: Software Portability for DHTC

Christina Koch (ckoch5@wisc.edu)

OSG User School 2022

Goals For This Session

- Describe what it means to make software “portable”
- Understand the basics of...
  - how software works
  - where software is installed
  - how software is accessed and run
- ...and the implications for Distributed High Throughput Computing (DHTC) and software portability.
- Learn about and use software portability techniques

An Analogy

Running software on your own computer is like cooking in your own kitchen.

On Your Computer

- You know what you already have.
  - All the software you need is already installed.
- You know where everything is (mostly).
- You have full control.
  - You can add new programs when and where you want.

The Challenge

Running on a shared computer is like cooking in someone else's kitchen.

On Someone Else's Computer

- What's already there?
  - Is R installed? Or Python? What about the packages you need?
- Do you know where anything is?
- Are you allowed to change whatever you want?

The Solution

- Think like a backpacker.
- Take your software with you.
  - Install anywhere
  - Run anywhere
- This is called making software *portable*.

PRELIMINARY CONCEPTS

Running Commands

- When we submit a job, our primary “work” is expressed as a command (or multiple commands) that can be run on the command line*. For example:

```
$ python analysis.py input0.csv
```

```
$ blast -db pdbaa/pdbaa -query mouse.fa -out mouse.result
```

```
$ gmx pdb2gmx -f pro.gro -o mol.gro
```

*prerequisite for running HTC jobs: your work can be run from the command line

Software Is Files

Behind the scenes, any commands we run are actually using software **files** stored somewhere on the computer.

```
$ python analysis.py input0.csv
```

```
$ blast -db pdbaa/pdbaa -query mouse.fa -out mouse.result
```

```
$ gmx pdb2gmx -f pro.gro -o mol.gro
```

Software Is Files

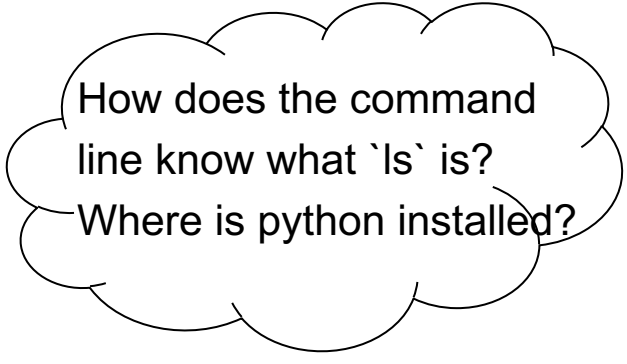
How to see the software program “echo”:

```
$ echo Echo is a command
Echo is a command

$ which echo
/usr/bin/echo

$ ls -lh /usr/bin
```

Command Line and Location

To run a program on the command line, your computer needs to know where the program is located in your computer's file system.

```
$ ls
$ python
$ ~/wrapper.sh
```



Software Locations

The shell keeps a list of software file locations stored in a variable called the PATH. We can print it out using “echo”:

```
$ echo $PATH
```

What if we want to run our *own* program – how do we tell the command line where it is?

Two Location Options

Provide a path (relative or absolute)

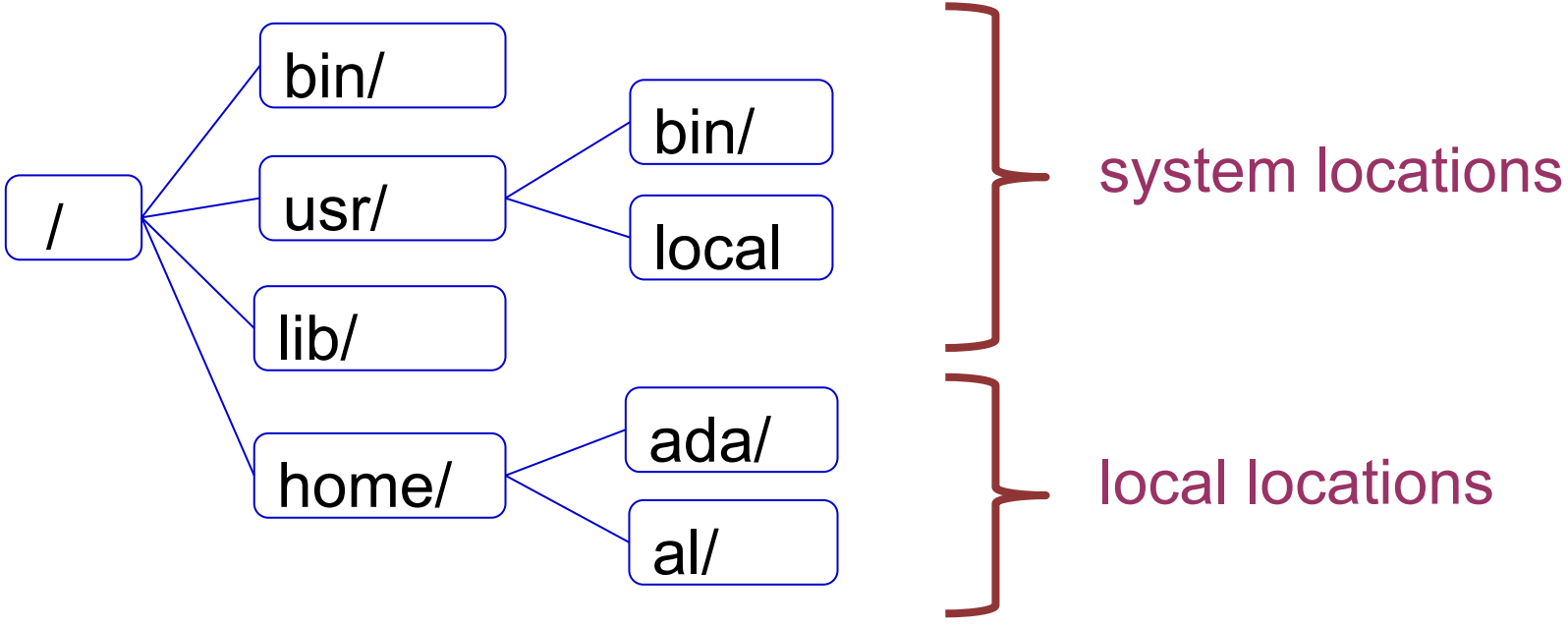
```
[~/Code]$ mypy/bin/python --version
Python 2.7.7
```

Use “the” PATH

```
$ export PATH=/Users/alice/Code/mypy/bin:$PATH
$ echo $PATH
/Users/alice/Code/mypy/bin:/usr/local/bin:/usr/bin:/bin:/usr
/sbin:/sbin
$ which python
/Users/alice/Code/mypy/bin/python
```

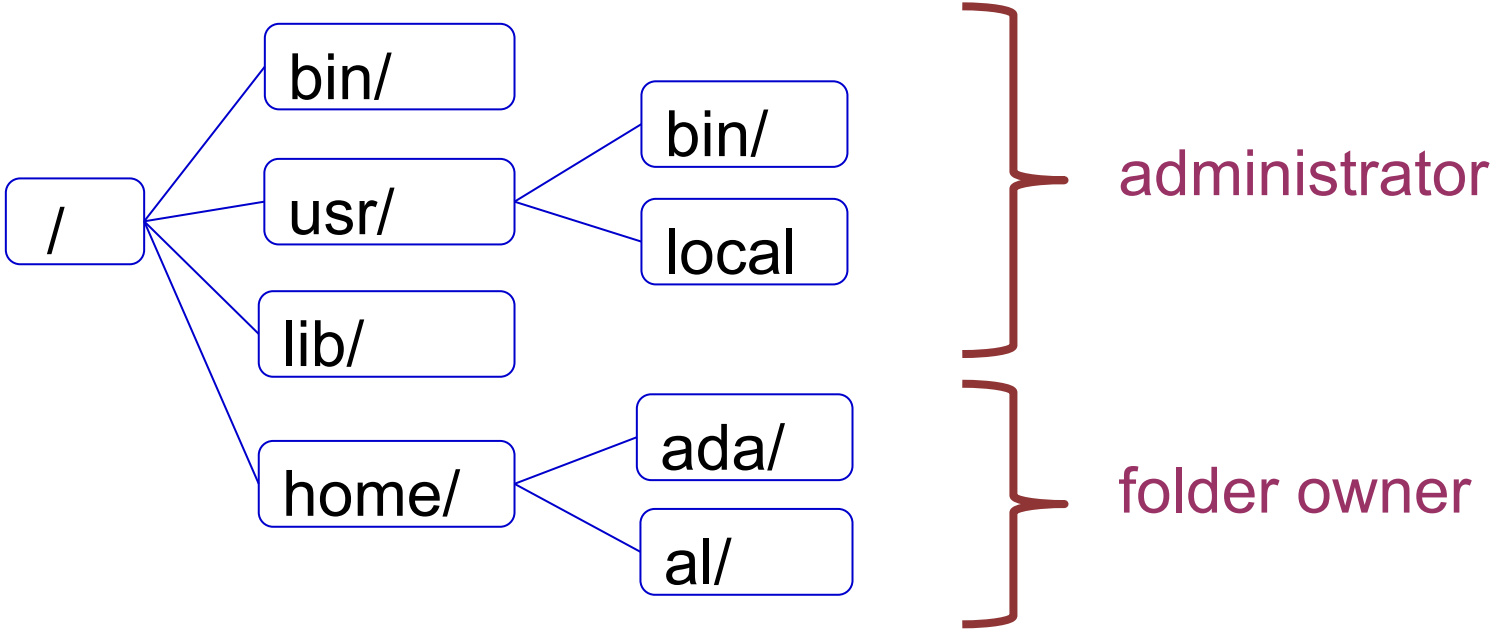
Location, Location, Location

- Where can software be installed?



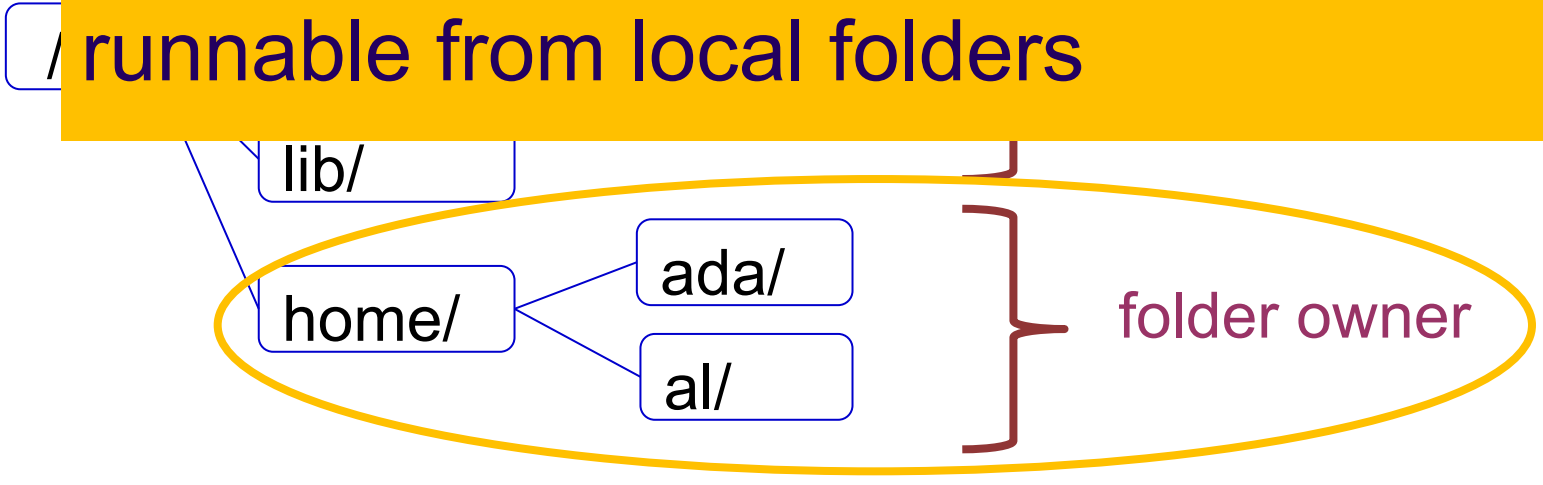
Location, Location, Location

- Who can add to these locations?



Location, Location, Location

On the OSPool, we need to assume that we are *not* administrators and our software needs to be installable and runnable from local folders



Portability Summary

- Run “anywhere” by:
  - bringing along the (Linux-compatible) software files you need…
  - to a location you can access/control…
  - telling the command line where that location is…
  - and using it to run your code.

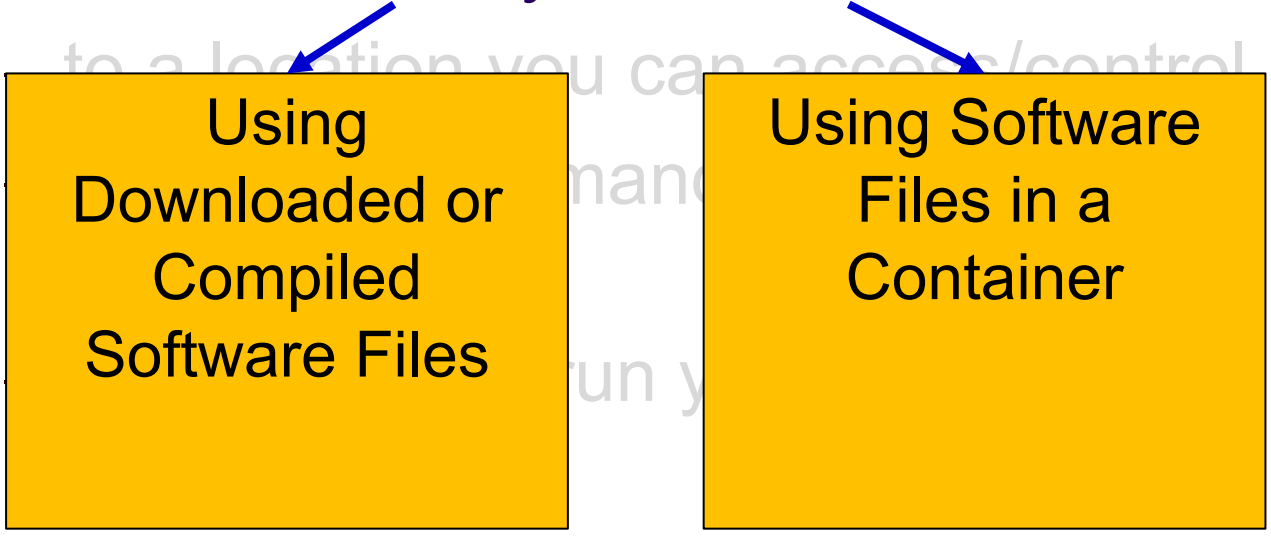
Demo

- GUI → Command line
- Full installation → Bring along files
- Using installation location

Demo files: https://github.com/ChristinaLK/osg-school-sw-demo

Portability Summary

- Run “anywhere” by:
  - bringing along the (Linux-compatible) software files you need…



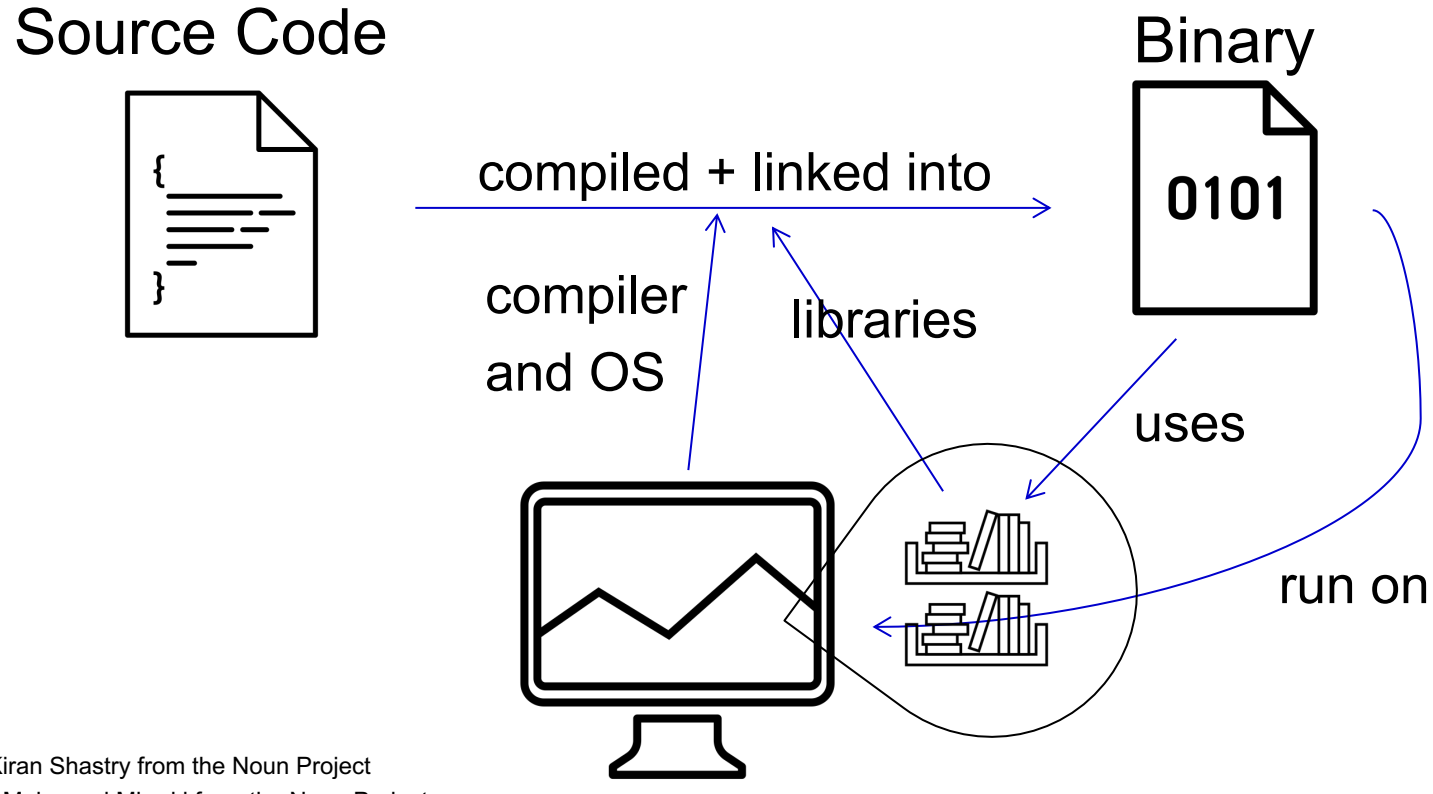
BRING ALONG SOFTWARE FILES

Ways to Prepare Software Files

- Download pre-compiled software
- Compile yourself
  - Single binary file
  - Installation contained in a single folder

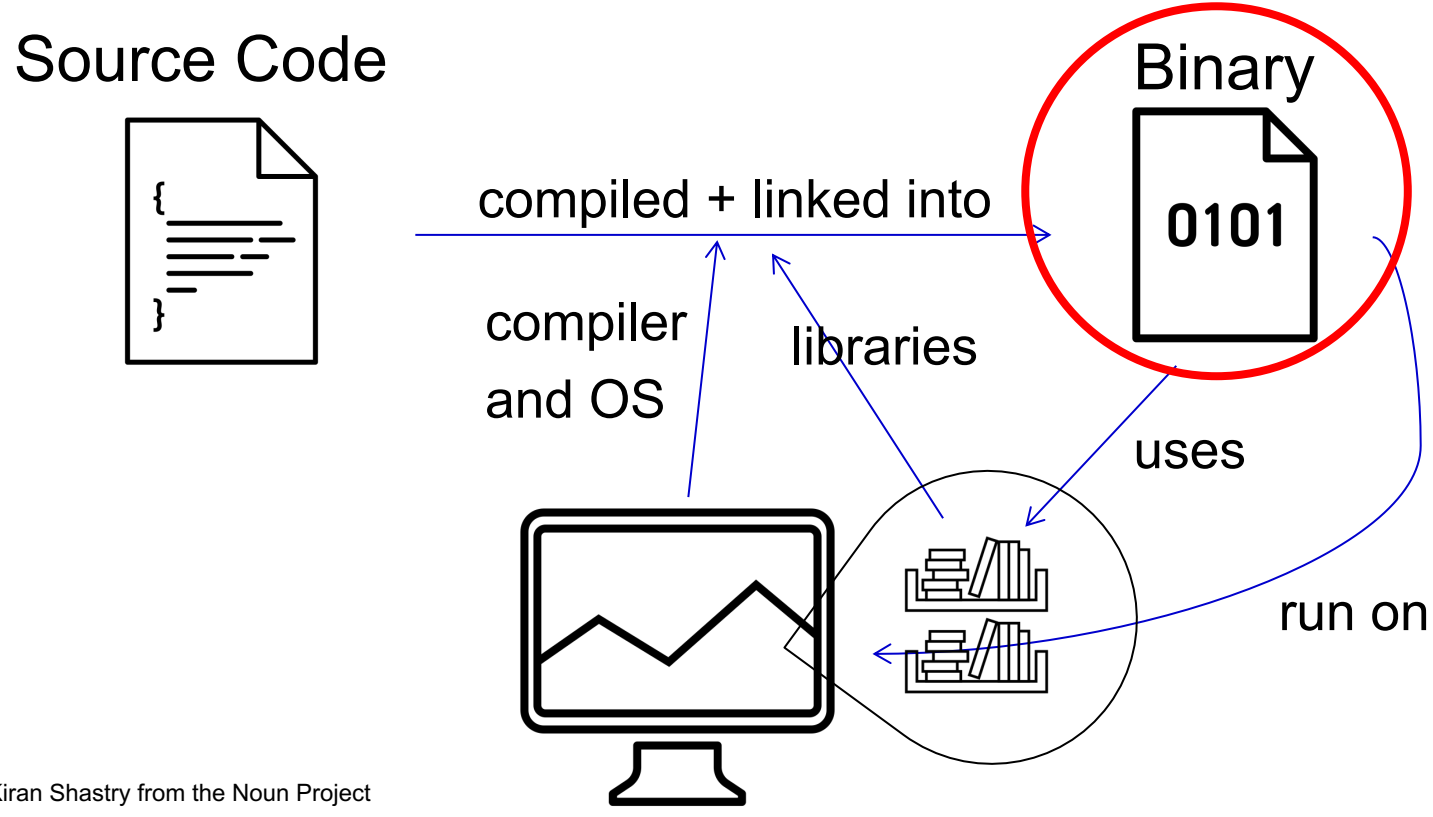
We **always** need a “compiled” file of some kind.

What is Compilation?



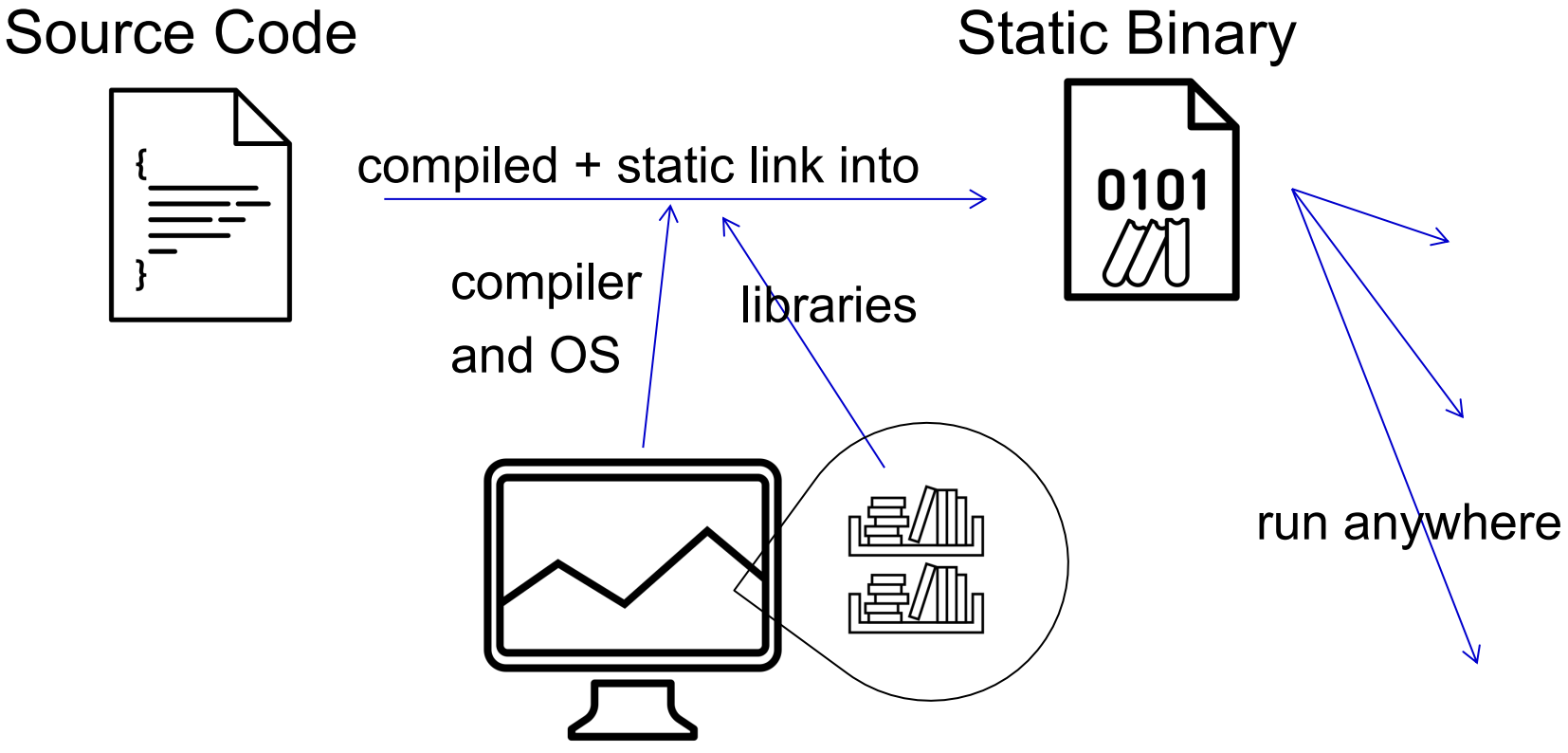
binary code by Kiran Shastry from the Noun Project
Source Code by Mohamed Mbarki from the Noun Project
Computer by rahmat from the Noun Project
books by Viral faisalovers from the Noun Project

What is Compilation?



binary code by Kiran Shastry from the Noun Project
Source Code by Mohamed Mbarki from the Noun Project
Computer by rahmat from the Noun Project
books by Viral faisalovers from the Noun Project

Static Linking



Book by Aleksandr Vector from the Noun Project

Compilation Process

- Use a compiler (like `gcc`) directly
  - Can use options to control compilation process
- More common:

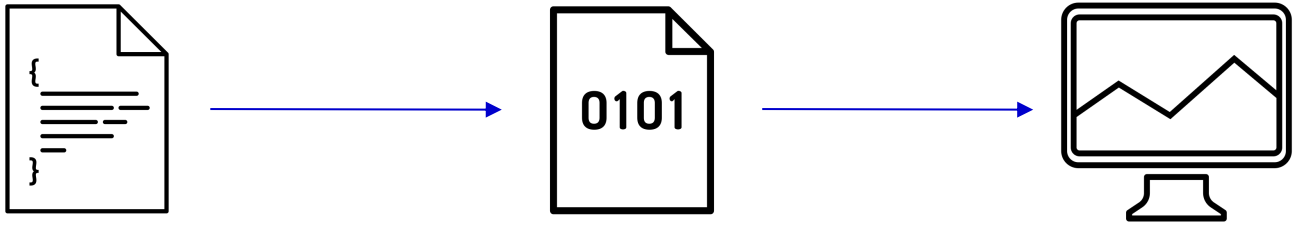
  `./configure` (or `cmake`)

  `make`

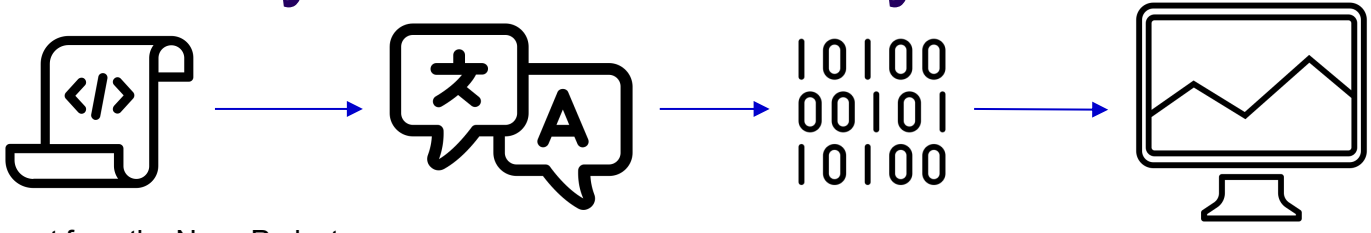
  `make install`
  - Installation options (like where to install) are usually set at the configure/cmake step

Interpreted Code

- Instead of being compiled and then run…



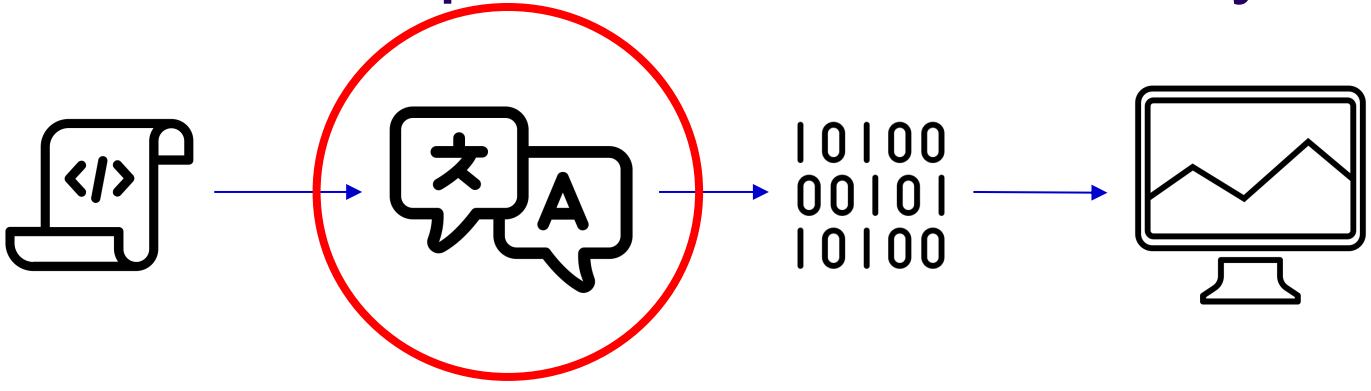
- …interpreted languages are translated into binary code “on the fly.”



script by Adrien Coquet from the Noun Project
translate by Adrien Coquet from the Noun Project
coding by Vectorstall from the Noun Project

Interpreted Code

- And the interpreter is itself a binary file.



script by Adrien Coquet from the Noun Project
translate by Adrien Coquet from the Noun Project
coding by Vectorstall from the Noun Project

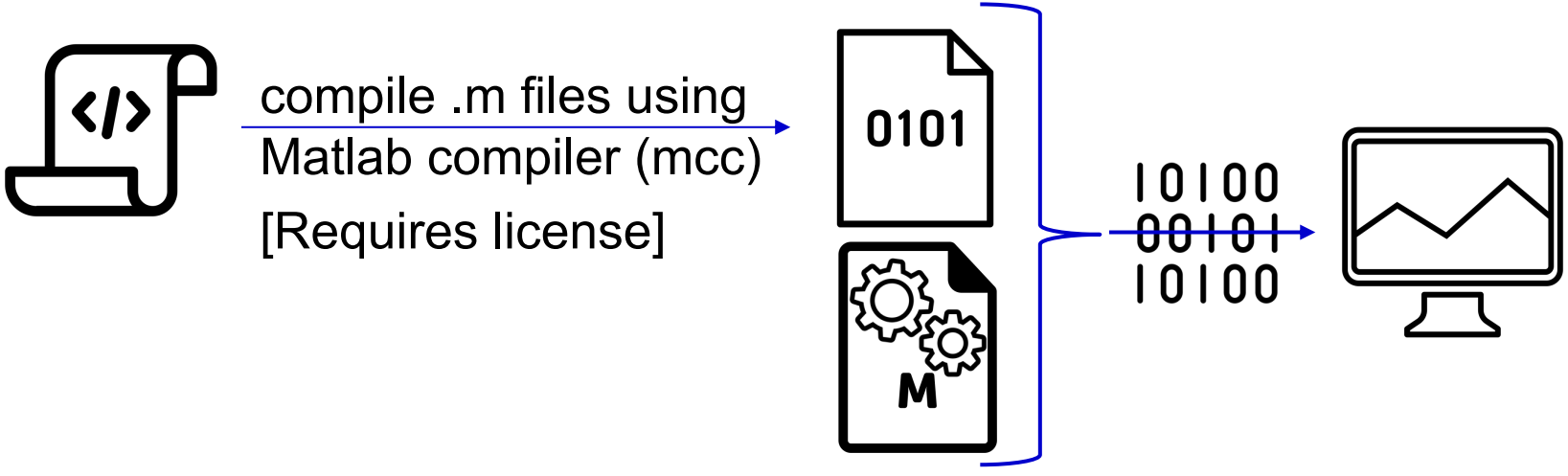
Open Science Grid

What Kind of Code?

- Programs written in C, C++ and Fortran are typically compiled.
- For interpreted (scripting) languages like perl, Python, R, or Julia:
  - Don’t compile the scripts, but *do* use a compiled copy of the underlying language interpreter.

Matlab

- Matlab is a scripting language…but can also be compiled.



compiled file and Matlab Runtime work together to run program.

[no license needed]

M File by Viktor Vorobyev from the Noun Project
Gears by Trevor Dsouza from the Noun Project

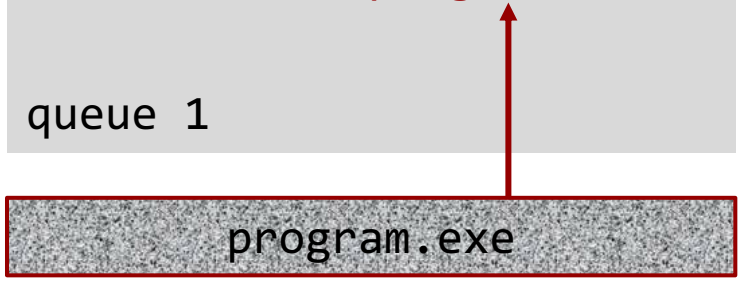
RUN "BROUGHT-ALONG" SOFTWARE FILES

Ways to Run Software

Executable

- Software must be a single compiled binary file or single script.

```
executable = program.exe

queue 1
```



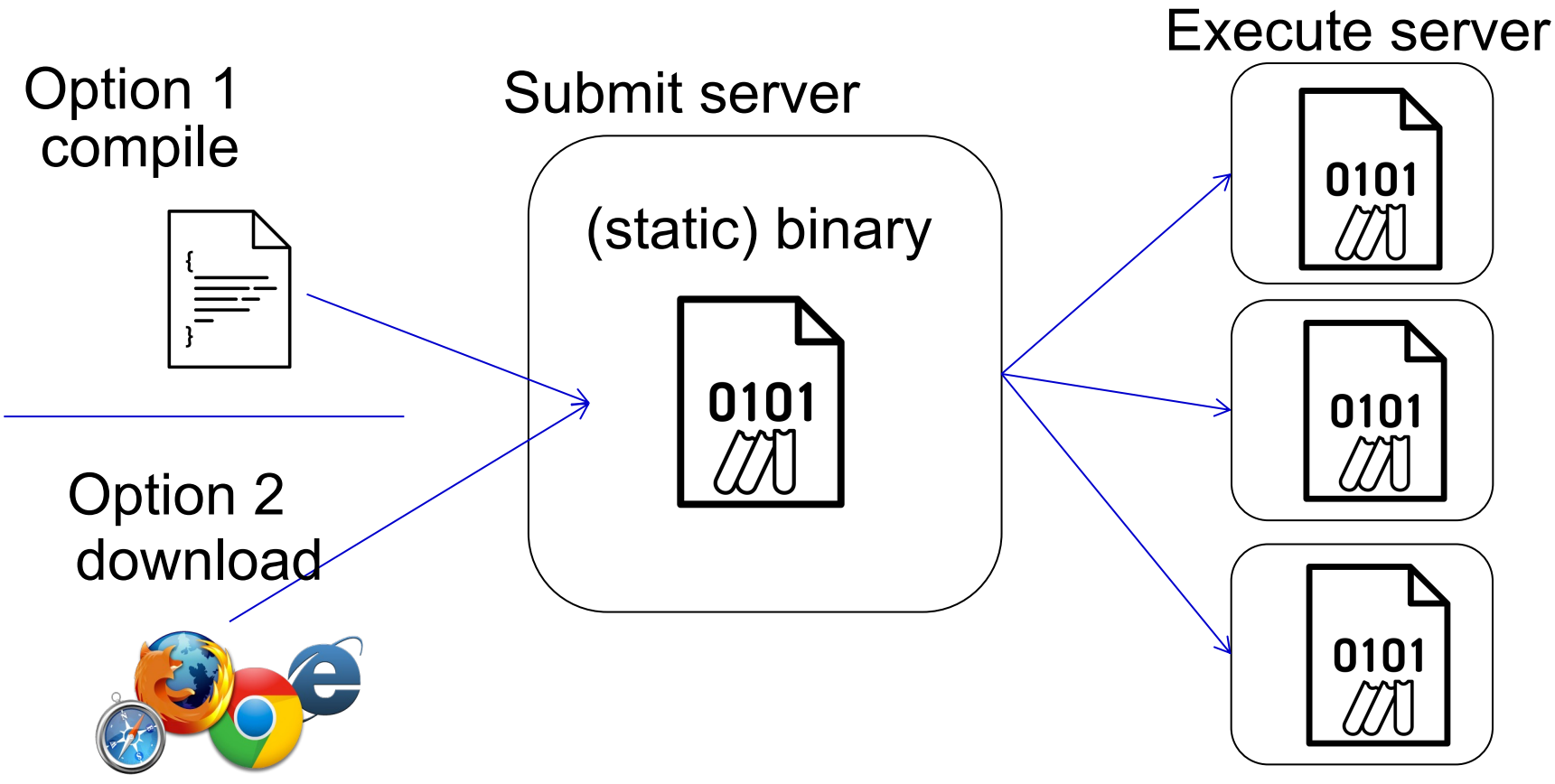
Wrapper Script

- Software can be in any compiled format.

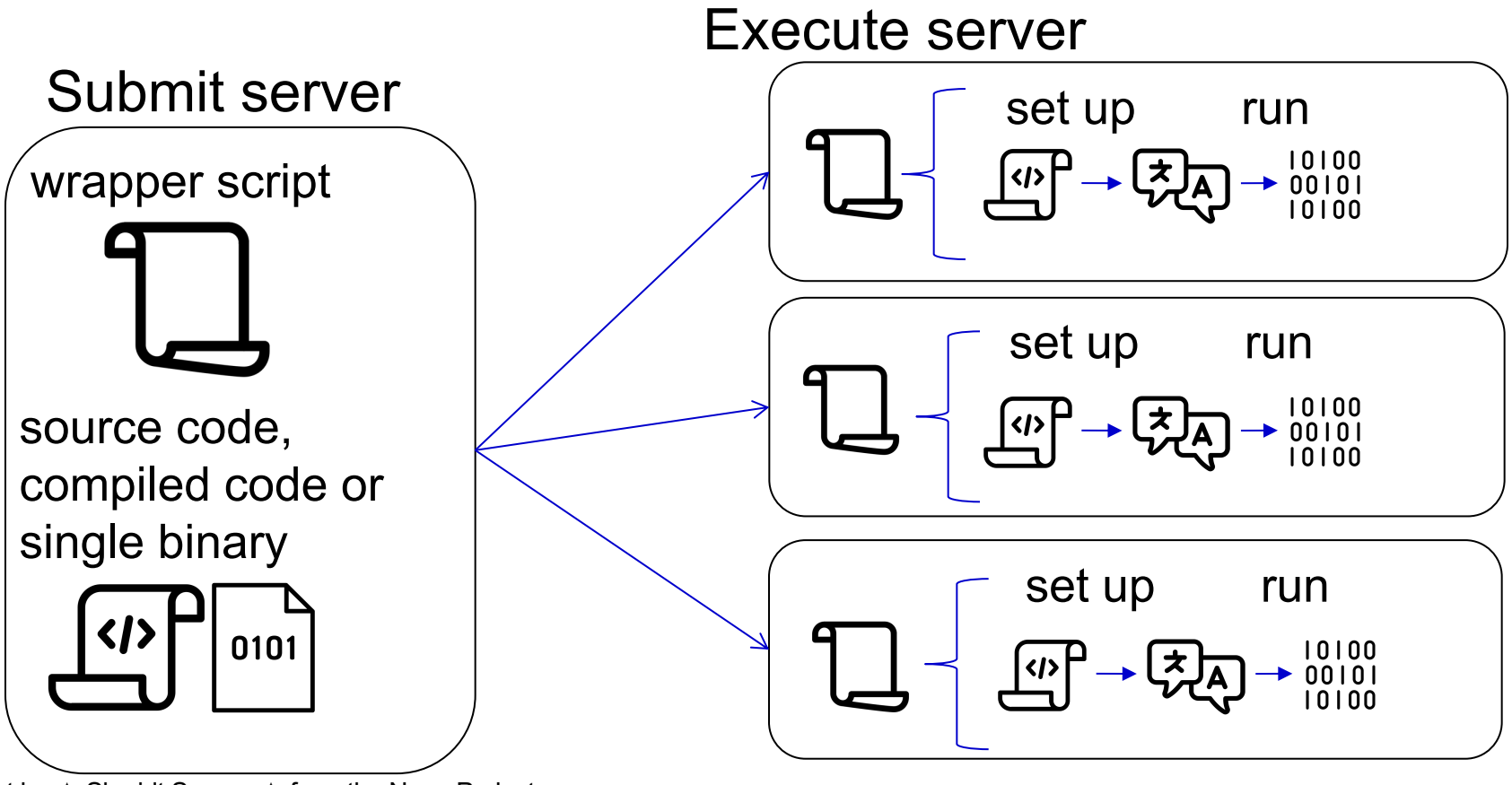
```
executable = run_program.sh
transfer_input_files =
             program.tar.gz
queue 1
```

```
#!/bin/bash
# run_program.sh

tar –xzf program.tar.gz
program/bin/run in.dat
```

Single Binary Workflow



Wrapper Script Workflow

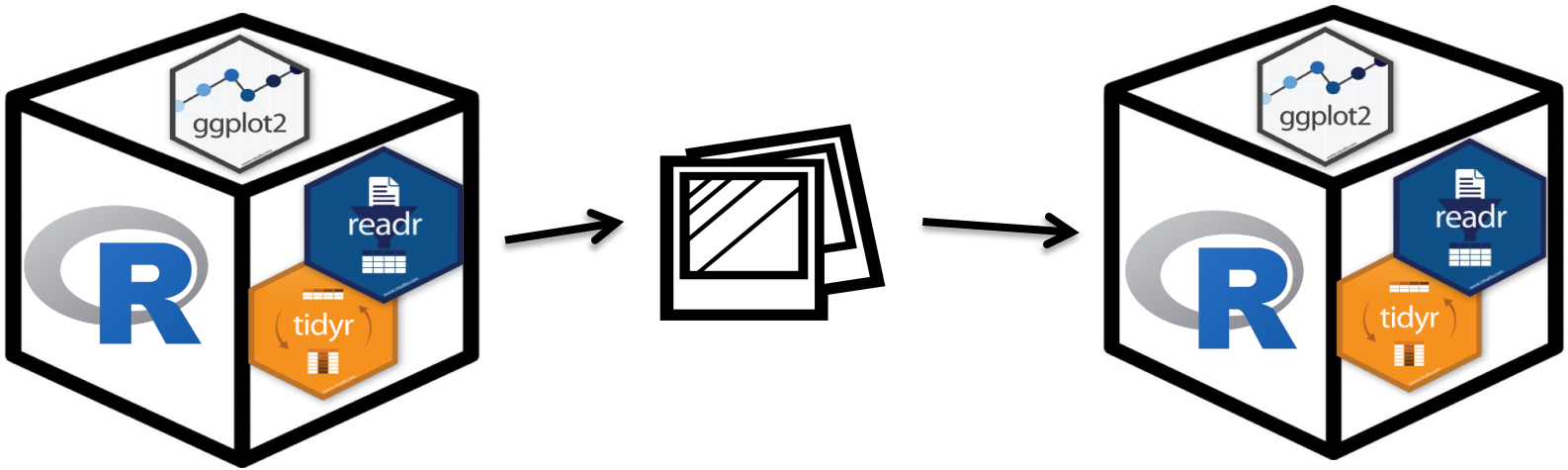


script by ✦ Shmidt Sergey ✦ from the Noun Project

BRING ALONG CONTAINERS

Containers

- Containers are a tool for capturing an entire job “environment” (software, libraries, operating system) into an “image” that can be used again.



Credit: polaroid photos by Nick Bluth from the Noun Project

Returning to Our Analogy…

- Using a container is kind of like bringing along a whole kitchen…

Why Containers?

Why use containers instead of the methods we just discussed?

Why Containers?

- Complex installations: software that has a lot of dependencies or components.

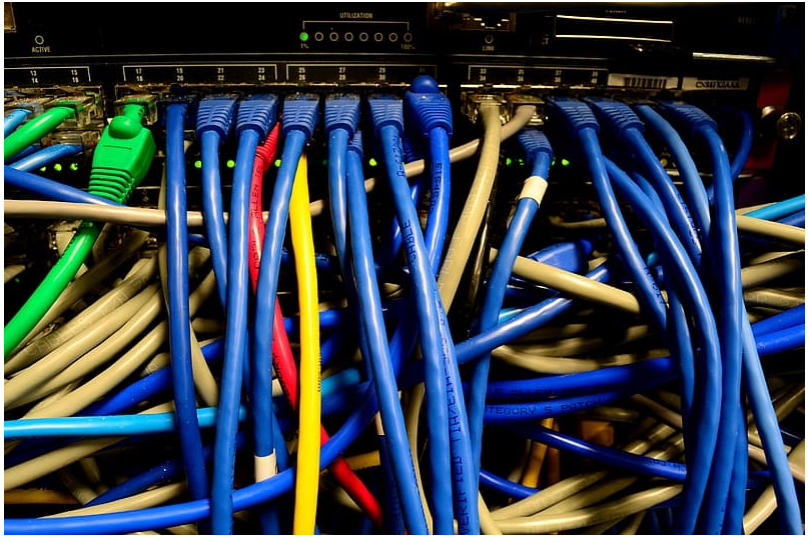


Photo on pikrepo

Why Containers?

- Software that can't be moved: do files or libraries have to be at a specific path?

Photo by R. Miller on Flickr, CC-BY

Why Containers?

- Sharing with others:
  one container can be used by a whole group that's doing the same thing.

Why Containers?

- Running on different systems: The same container can run on Linux, Mac and Windows

Photo by Ilona Frey on Unsplash

Why Containers?

- Reproducibility: save a copy of your environment.

Getting Containers

- To use a container as your software portability tool, need to either:
  - Find a pre-existing container with what you need.
  - Build your own container.*

* not covered today

Container Types

- Two common container systems:

Docker

https://www.docker.com/

Singularity

https://sylabs.io/

Container Types

- Container system =
  - Container **image format**
  - Container "**engine**" for running
- **Image Format**
  - Always Linux-based
  - Docker images can be converted to Singularity images
- "**Engine**" capabilities
  - Singularity "engine" can run both Docker + Singularity images
  - Docker "engine" installs on Linux, Mac, Windows, meaning Docker containers can be run on any OS

RUN CONTAINERS

Submit File Requirements

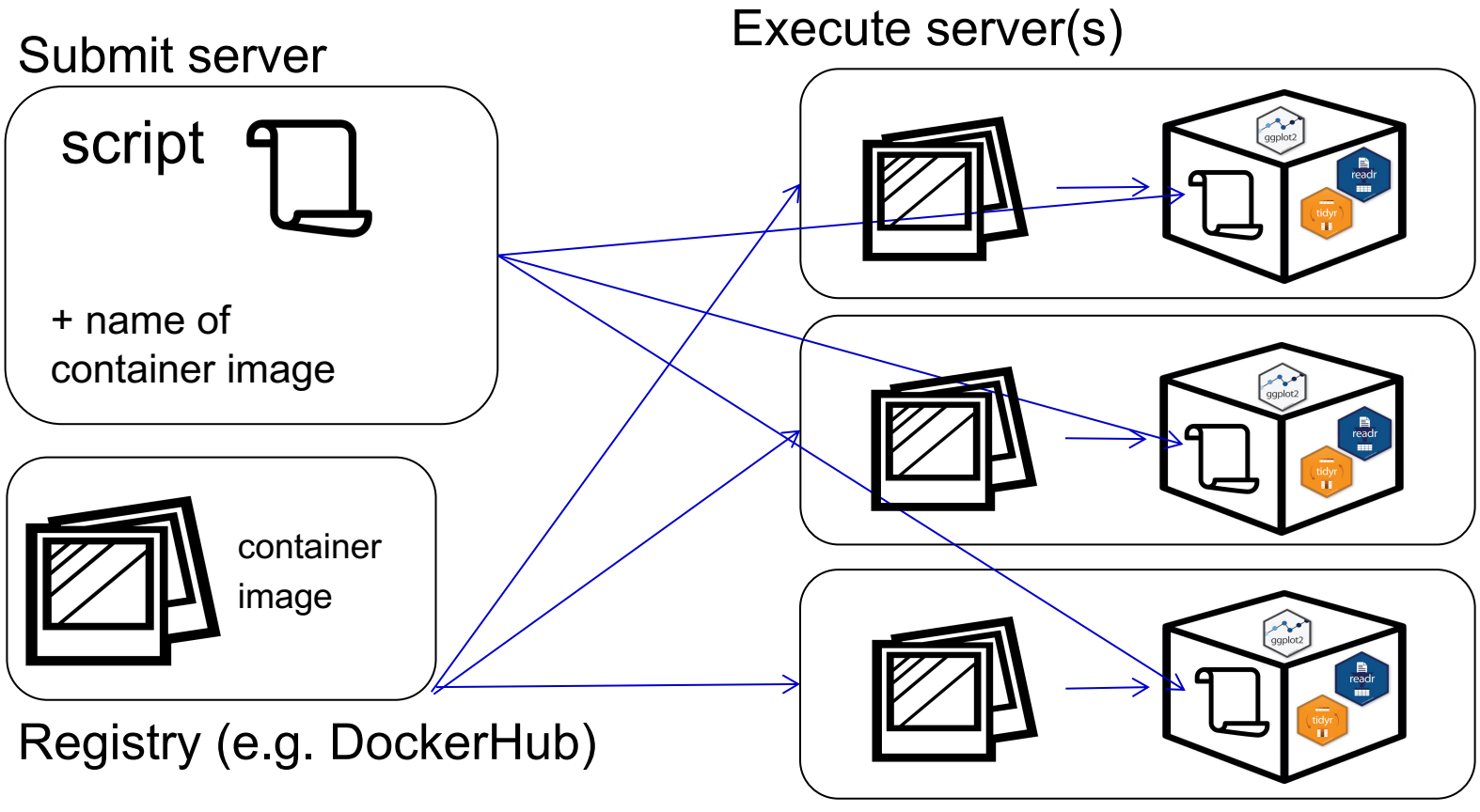
- Docker (from CHTC Access Point)

```
universe = docker
docker_image = centos/python-34-centos7:latest
```

- Singularity (from OSPool Access Point)

```
+SingularityImage =
"/cvmfs/singularity.opensciencegrid.org/centos/pyt
hon-34-centos7:latest"
```

Container Workflow



WRAPPING UP

Conclusion

To use any software in a DHTC system:

1. Create/find software files:
   - download pre-compiled code, compile your own, create/find a container
2. Account for all dependencies, files, and requirements in the submit file.
3. If needed, write a wrapper script to set up the environment when the job runs.

Acknowledgements

- This work is supported by NSF under Cooperative Agreement OAC-2030508 as part of the PATh Project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

OLD SLIDES

How Software Works*

*Not to scale

Program

(software, code, executable, binary)

How Software Works*

*Not to scale

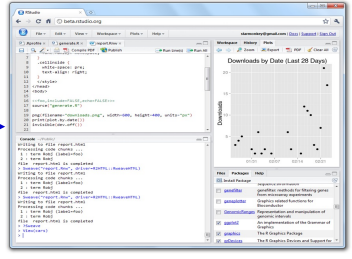
Program

(software, code, executable, binary)

launches to

Running Program

(process, instance)



How Software Works*

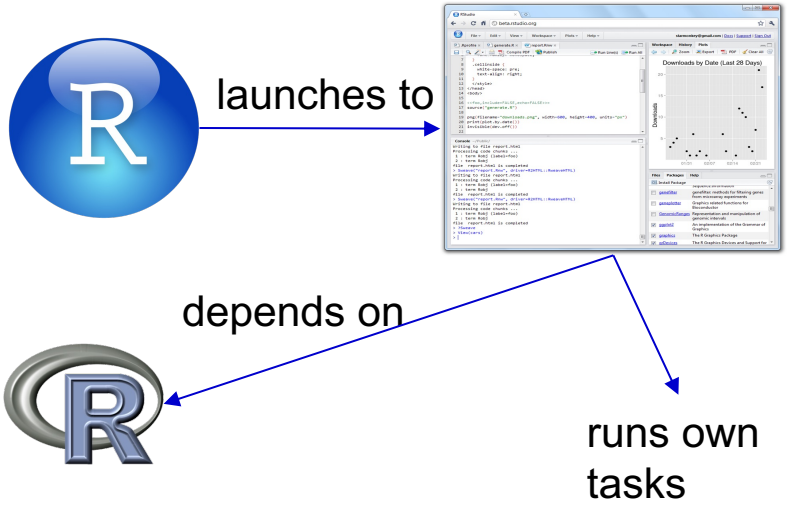
*Not to scale

Program

(software, code, executable, binary)

Running Program

(process, instance)



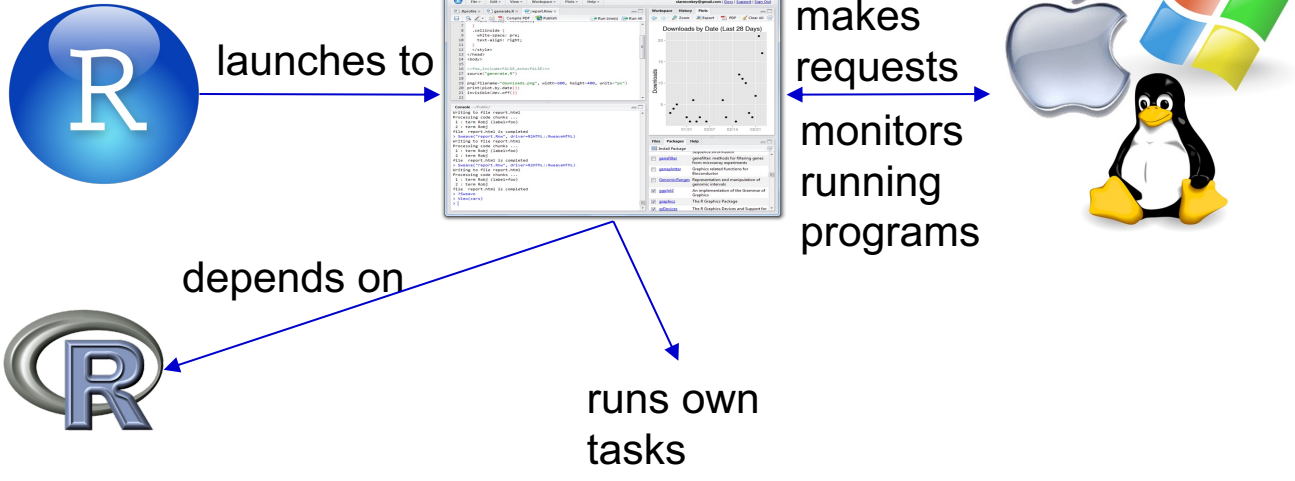
How Software Works*

*Not to scale

Program
(software, code, executable, binary)

Running Program
(process, instance)

Operating System



How Software Works*

*Not to scale

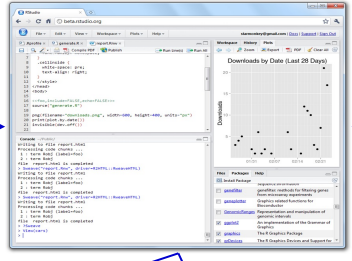
Program
(software, code, executable, binary)

Running Program
(process, instance)

Operating System

Hardware
(processors, memory, disk)

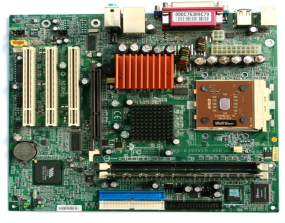
launches to



makes requests

monitors running programs

translates program's request



depends on

runs own tasks