
# Backpacking with Code: Software Portability for DHTC

Christina Koch (ckoch5@wisc.edu)
Research Computing Facilitator
University of Wisconsin - Madison

# Goals For This Session

- 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)
- Describe what it means to make software “portable”
- Learn about and use software portability techniques

# An Analogy


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

Open Science Grid

# 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 Problem

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

# Software Programs Are Files

- Principle
  - Software is a set of files.
  - These files have instructions for the computer to execute.
- Implications for DHTC
  - Isolate the specific software files needed for a job and bring them along.

# How Software Works*

*Not to scale

## Program

**(software, code, executable, binary)**

# How Software Works*

*Not to scale

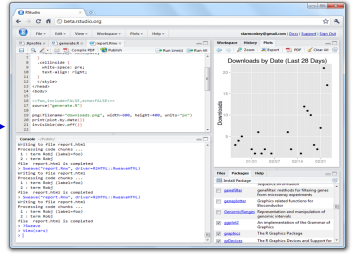
**Program**

**(software, code, executable, binary)**

launches to

**Running Program**

**(process, instance)**



# How Software Works*

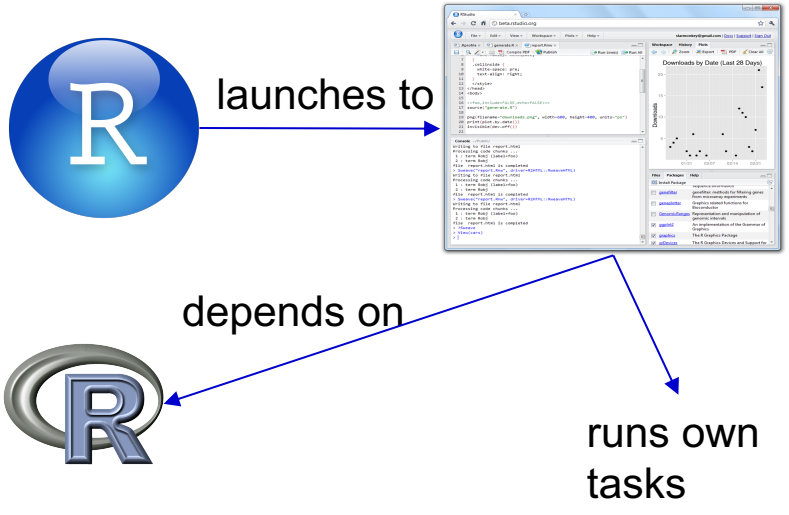
*Not to scale

**Program**

**(software, code, executable, binary)**

**Running Program**

**(process, instance)**



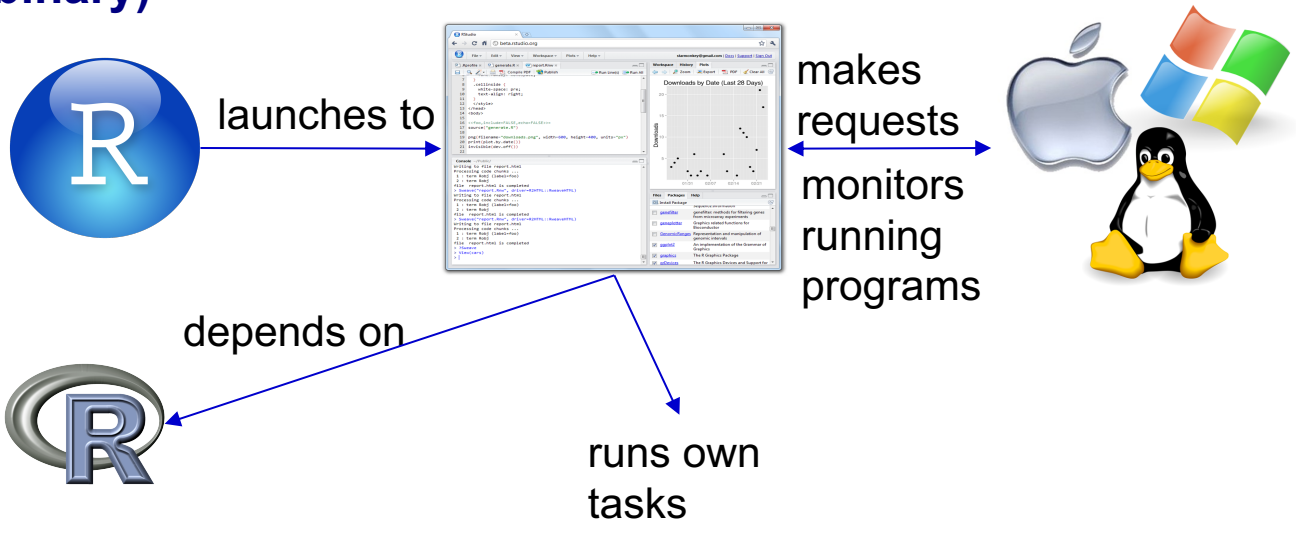
# How Software Works*

*Not to scale

**Program**
**(software, code, executable, binary)**

**Running Program**
**(process, instance)**

**Operating System**



# How Software Works*

*Not to scale

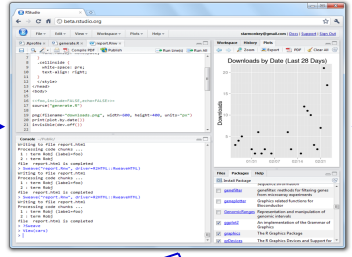
**Program**
**(software, code, executable, binary)**

**Running Program**
**(process, instance)**

**Operating System**

**Hardware**
**(processors, memory, disk)**

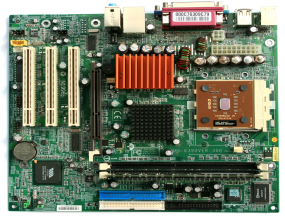
launches to



makes requests

monitors running programs

translates program's request



depends on

runs own tasks

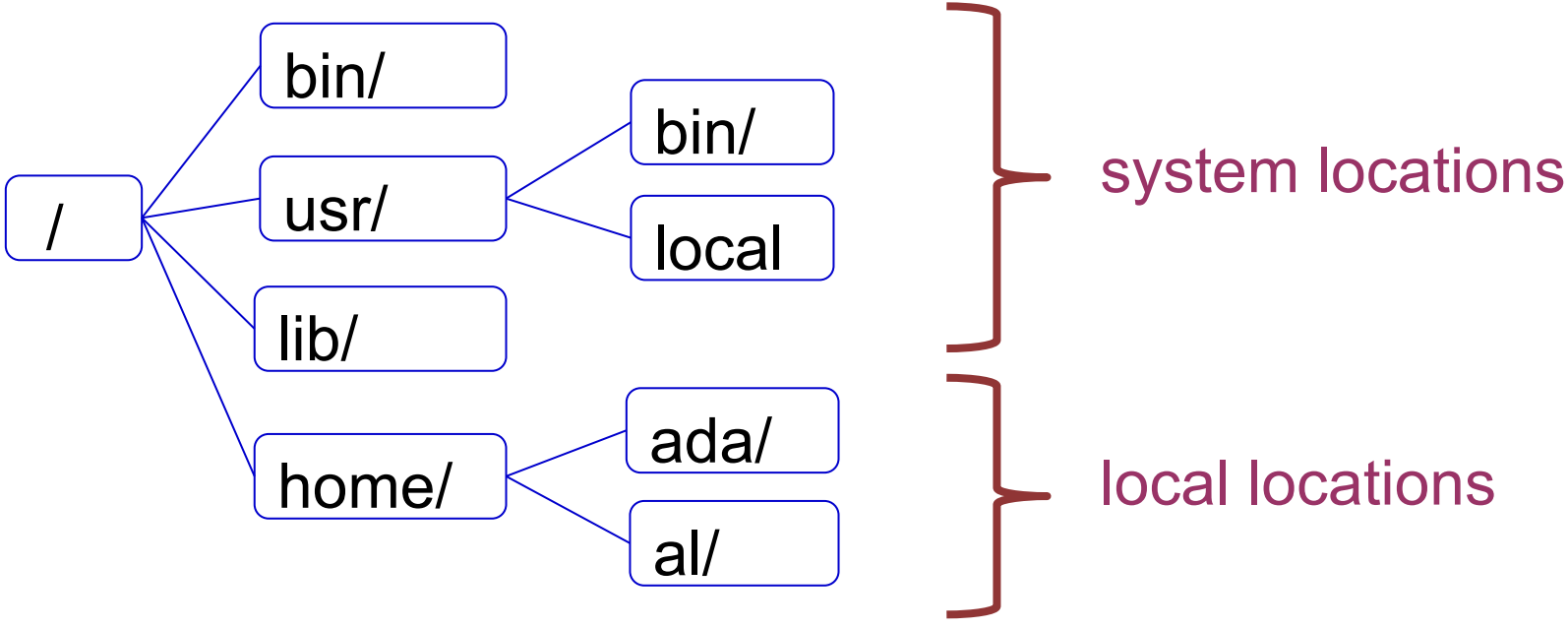
Open Science Grid

# How Software Works

- Principle:
  - Software depends on the operating system, and other installed programs.
- Implications for DHTC:
  - Software must be able to run on target operating system (usually Linux).
  - Know what else your software depends on.

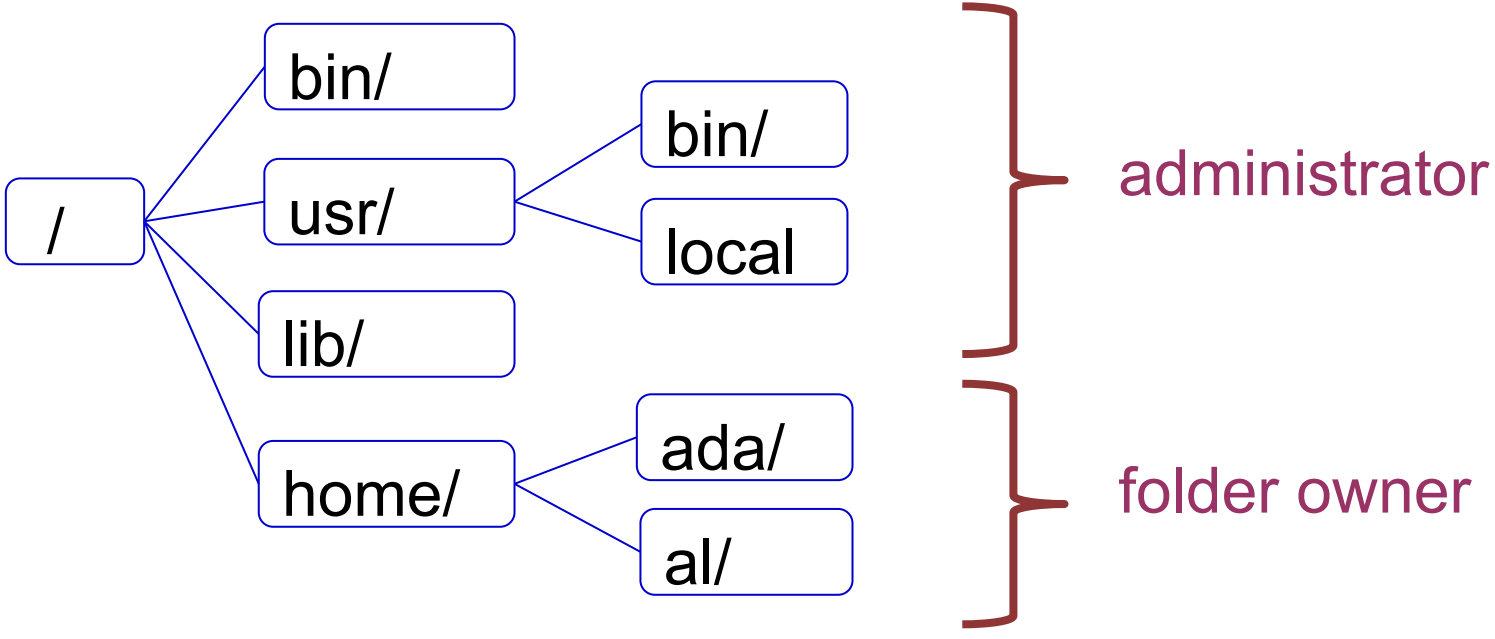
# Location, Location, Location

- Where can software be installed?



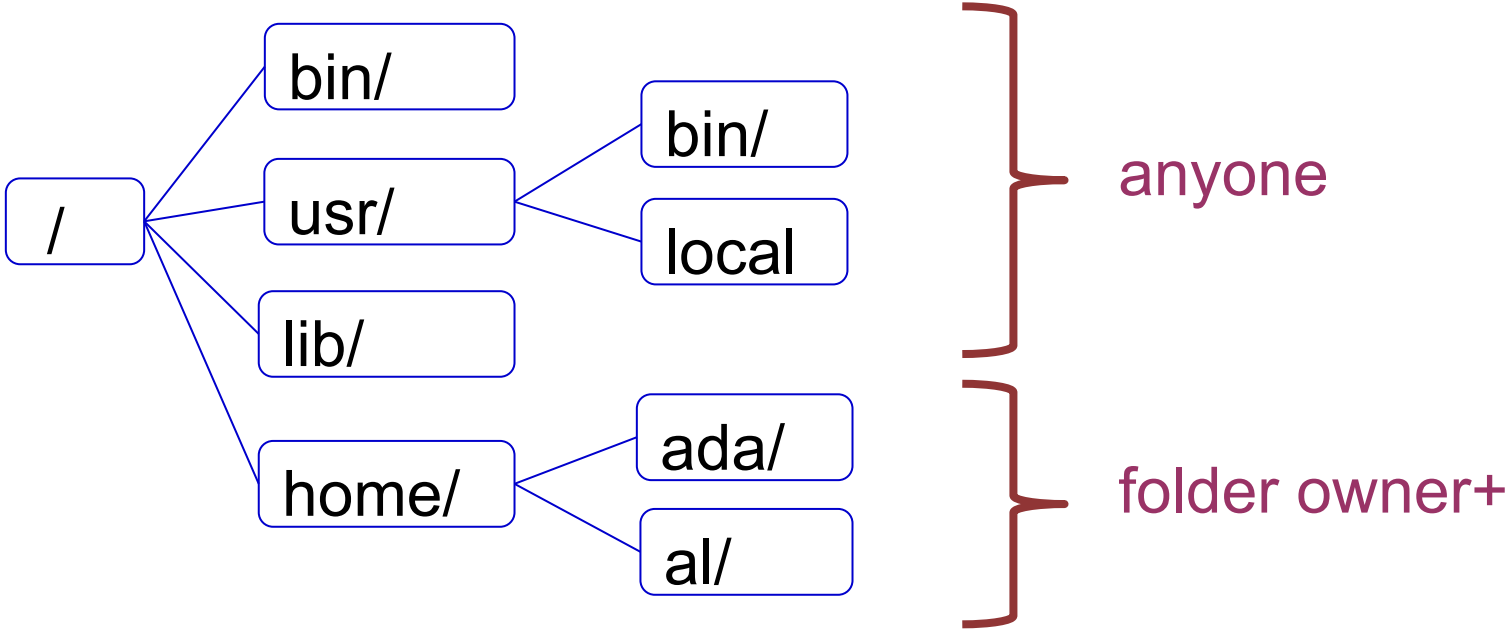
# Location, Location, Location

- Who can install the software?



# Location, Location, Location

- Who can access the software?

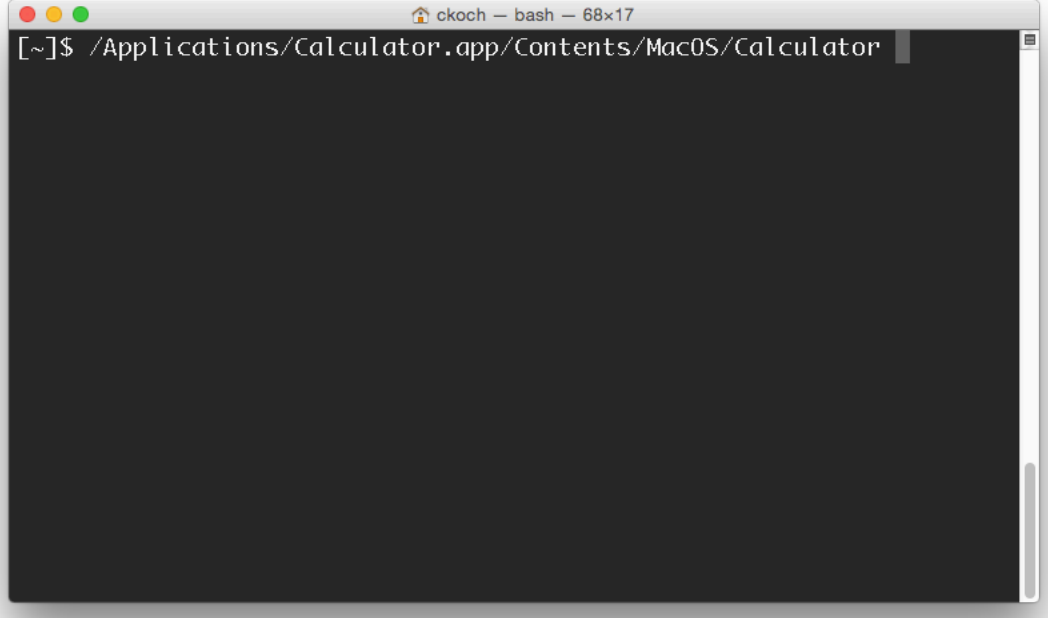


# Location, Location, Location

- Principle:
  - Software files have to be installed somewhere in the file system.
- Implications for DHTC:
  - Software must be installable without administrative privileges.
  - The software's location needs to be accessible to you.

# Command Line

How to automate programs?



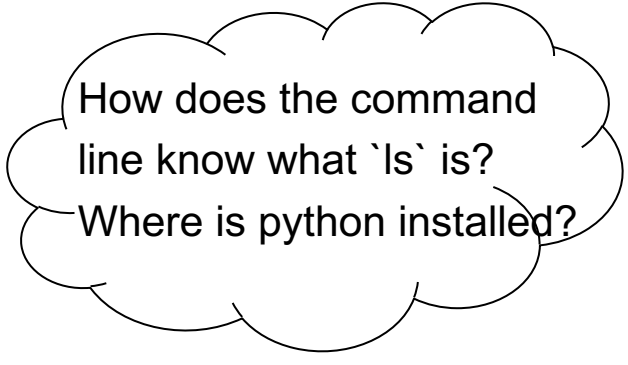
# Command Line

- Principle:
  - To automatically run software, need to use text commands (command line).
- Implications for DHTC:
  - Software must have ability to be run from the command line.
  - Multiple commands are okay, as long as they can be executed in order within a job.

# 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
```



# 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
```

# Command Line

- Principle:
  - To run a program on the command line, the computer has to be able to find it.
- Implications for DHTC:
  - There are different ways to “find” your software on the command line: relative path, absolute path, and PATH variable

# Portability

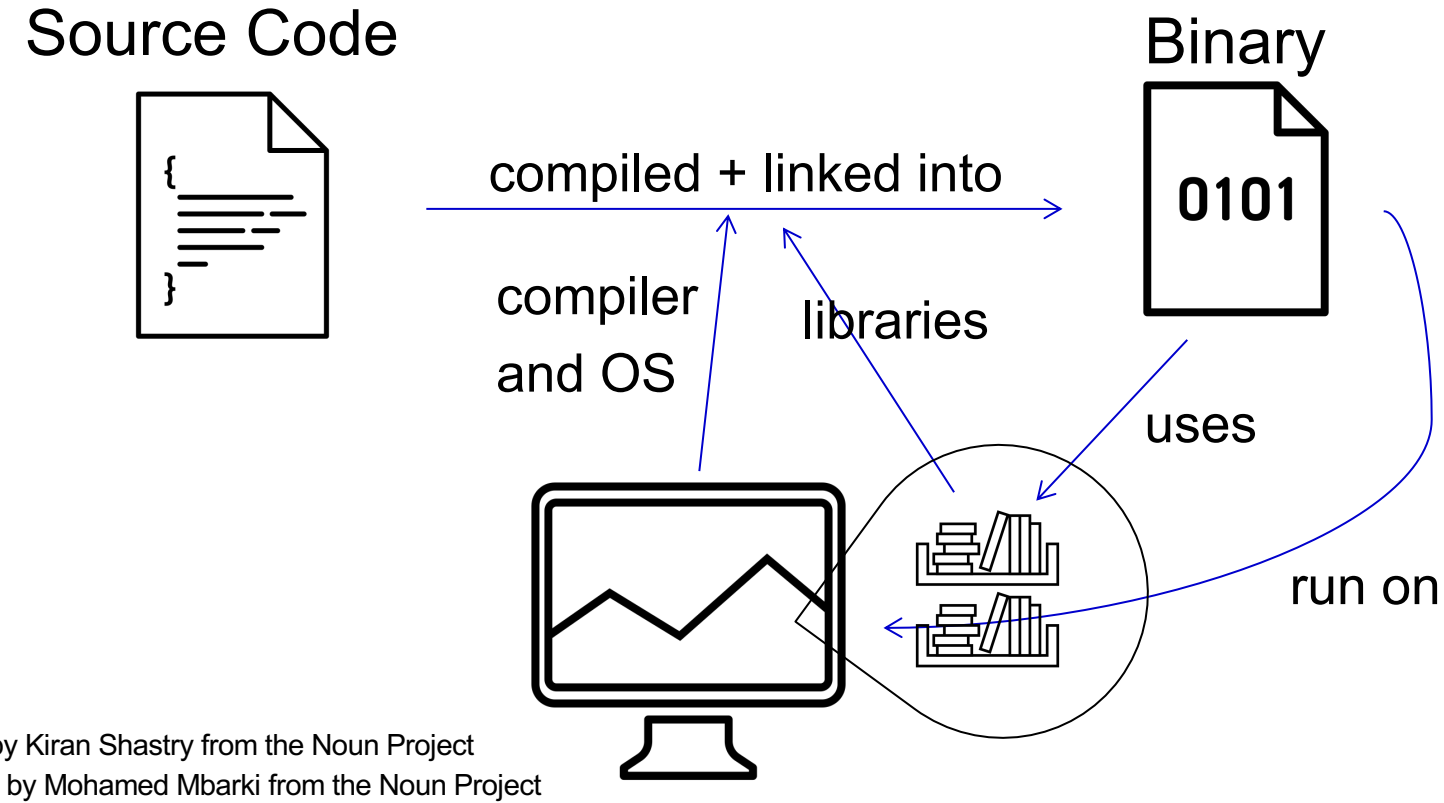
- Run “anywhere” by:
  - bringing along the software files you need…
  - to a location you can access/control…
  - using the command line to run…
  - by providing the correct software location…
  - (on Linux).

# BRING ALONG SOFTWARE FILES

# Ways to Prepare Software Files

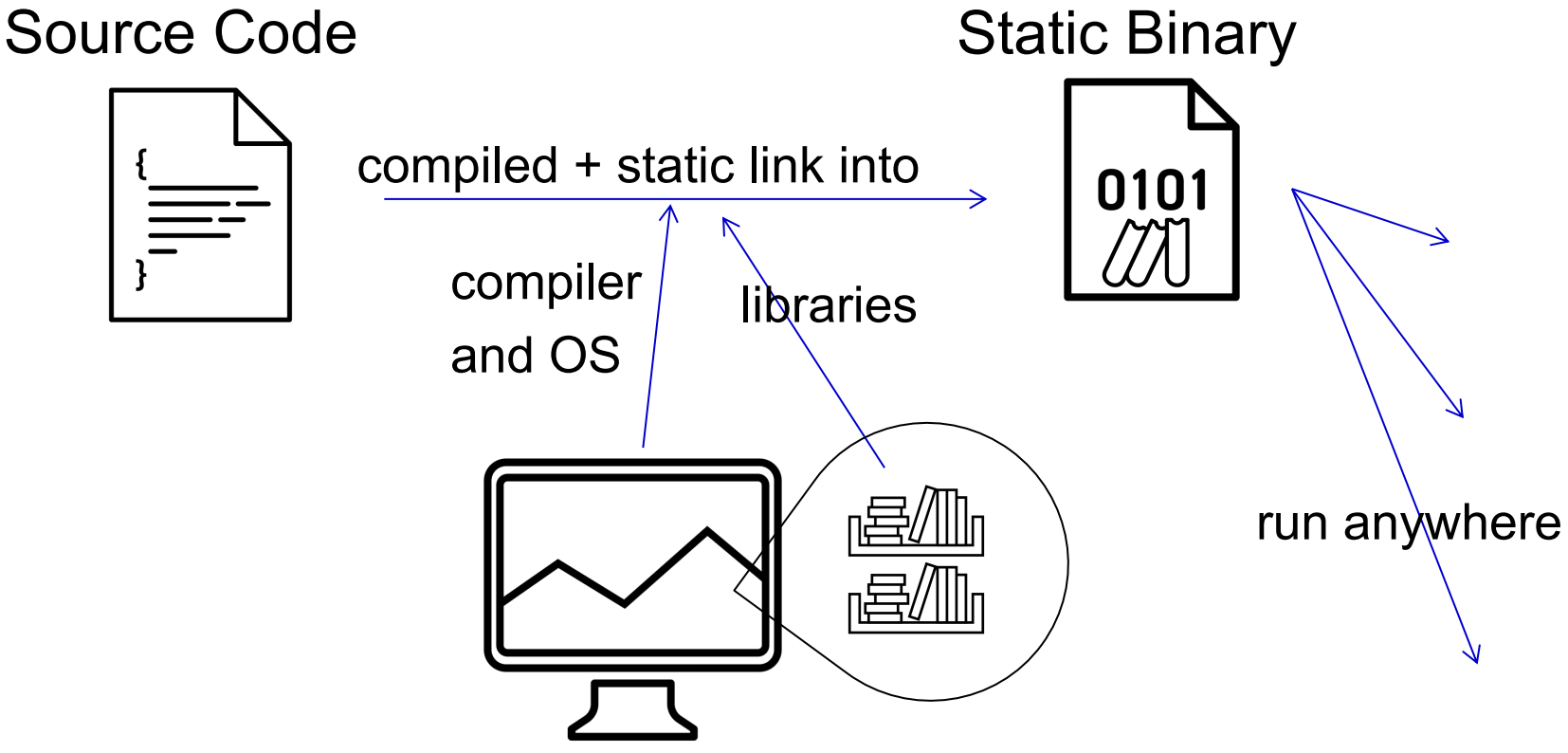
- Download pre-compiled software
- Compile yourself
  - Single binary file
  - Installation contained in a single folder

# 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:

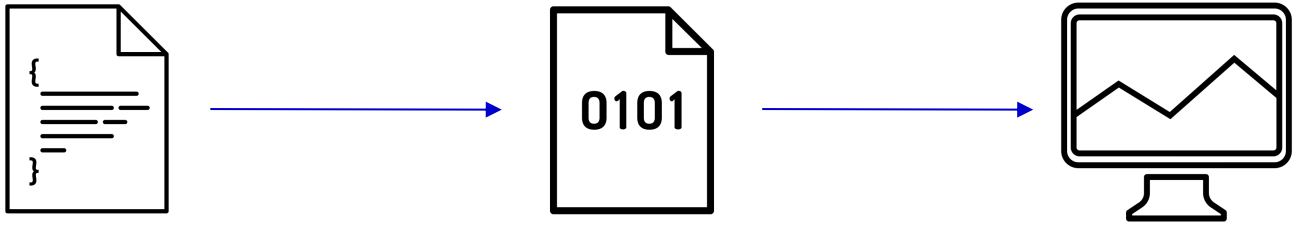
  ./configure # can also include options

  make

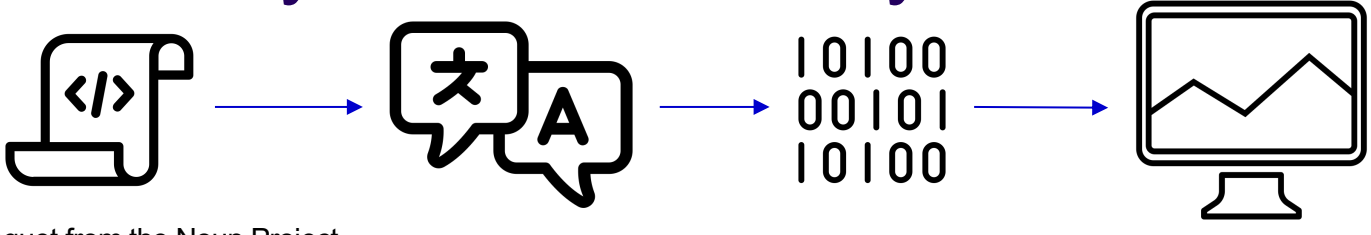
  make install

# 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

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.

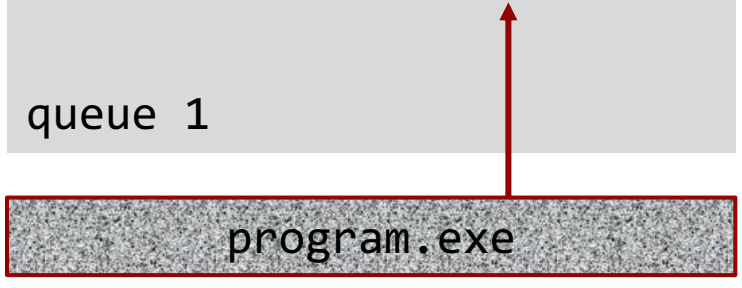
# Ways to Run Software

## Executable

- Software must be a single compiled binary file.

```
executable = program.exe

queue 1
```




## Wrapper Script

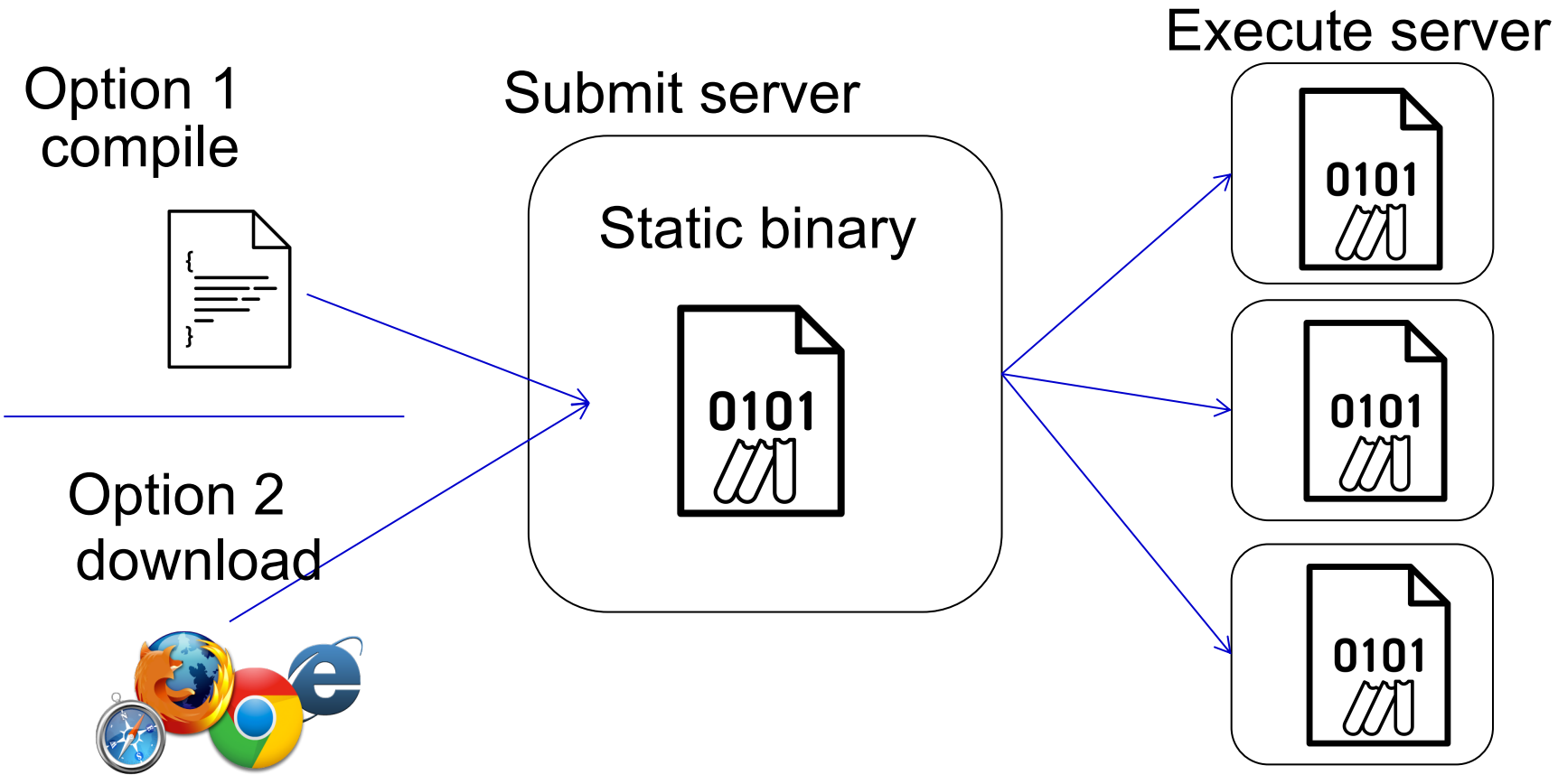
- Software can be in any compiled format.

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

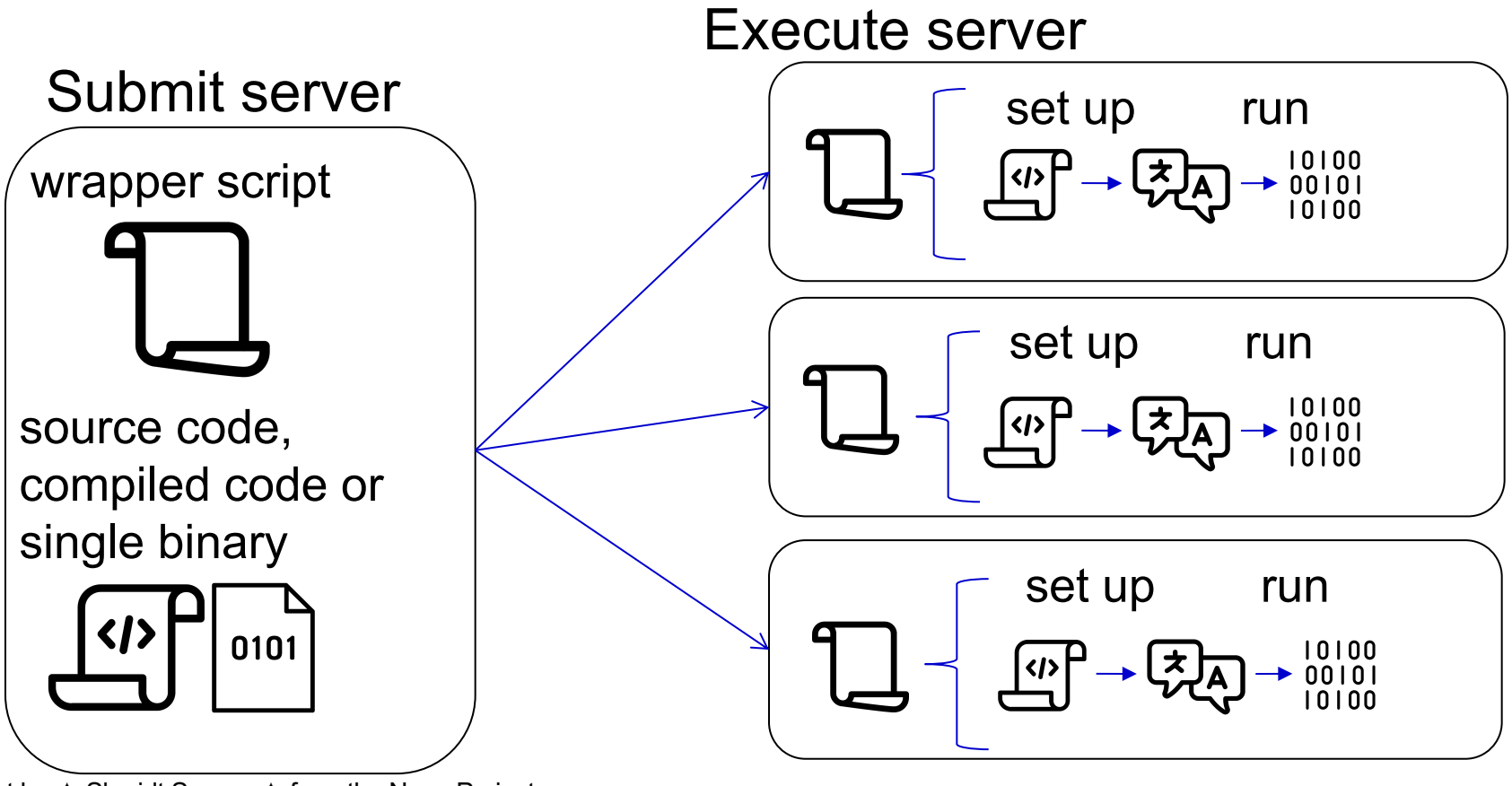
```
#!/bin/bash
# run_program.sh

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

# Single Binary Workflow



# Wrapper Script Workflow

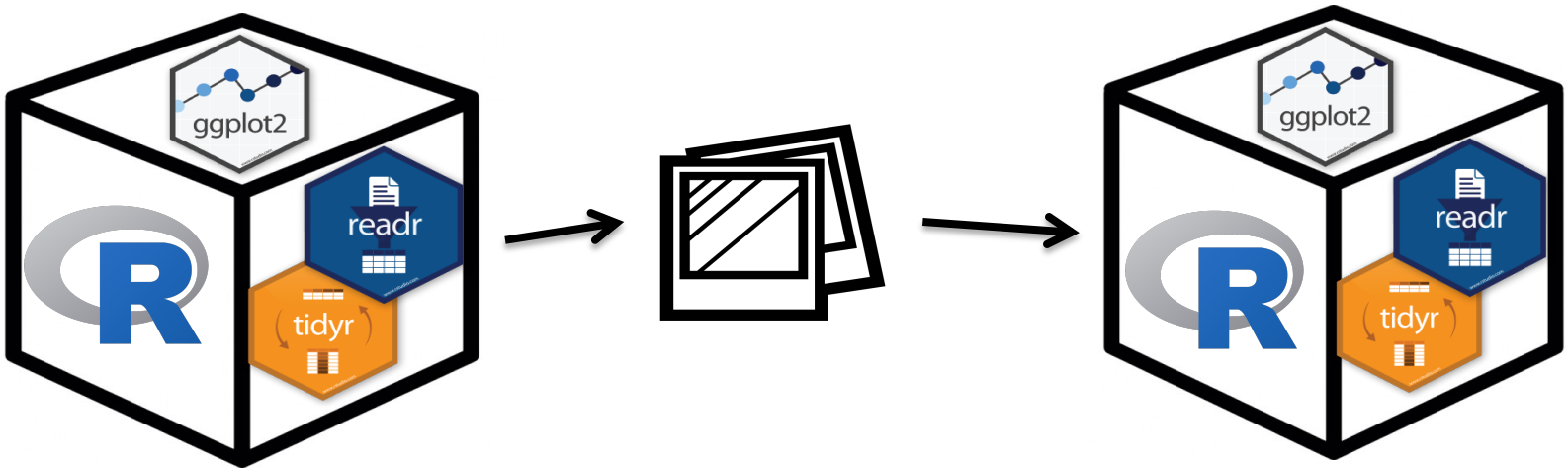



script by ✦ Shmidt Sergey ✦ from the Noun Project

# 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

Open Science Grid

# Why Containers?

Why use containers instead of the methods we just discussed?

- Complex installations: software that has a lot of dependencies or components.
- Software that can't be moved: do files or libraries have to be at a specific path?
- Sharing with others: one container can be used by a whole group that's doing the same thing.
- Reproducibility: save a copy of your environment.

# Using 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 | Singularity |
| --- | --- |
| https://www.docker.com/ | https://sylabs.io/ |


The container itself will always be some version of Linux - but can be run on Linux / Mac / Windows if Docker or Singularity is installed

# Submit File Requirements

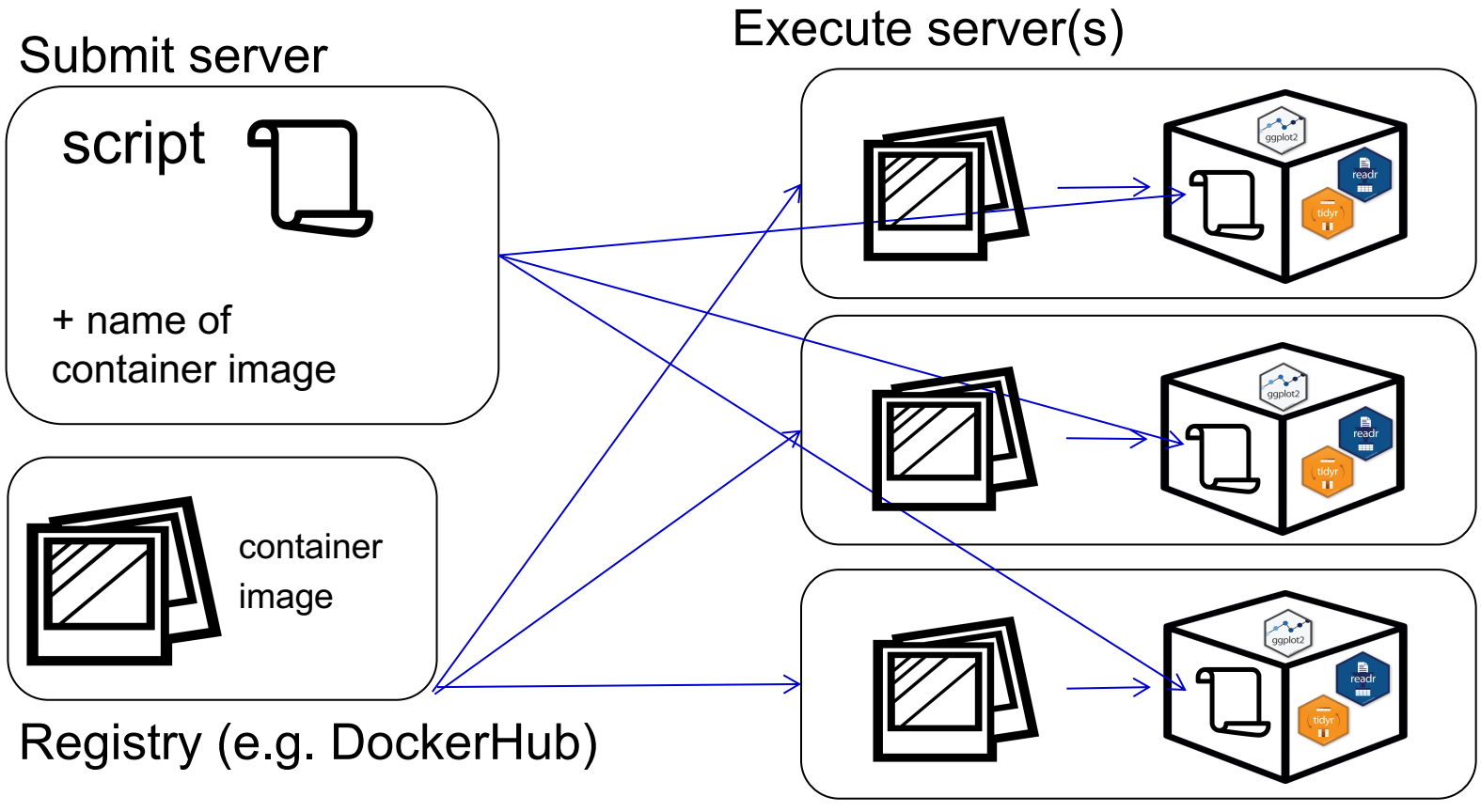
- Docker (from CHTC submit server)

```
universe = docker
docker_image = python:3.7.0
requirements = (HasDocker == true)
```

- Singularity (from OSG submit server)

```
+SingularityImage =
"/cvmfs/singularity.opensciencegrid.org/centos/python-34-centos7:latest"
requirements = (HAS_SINGULARITY == true)
```

# Container Workflow



# Conclusion

To use any software in a DHTC system:

1. Create/find software package:
   - 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 script to set up the environment when the job runs.

# PRE-INSTALLED SOFTWARE

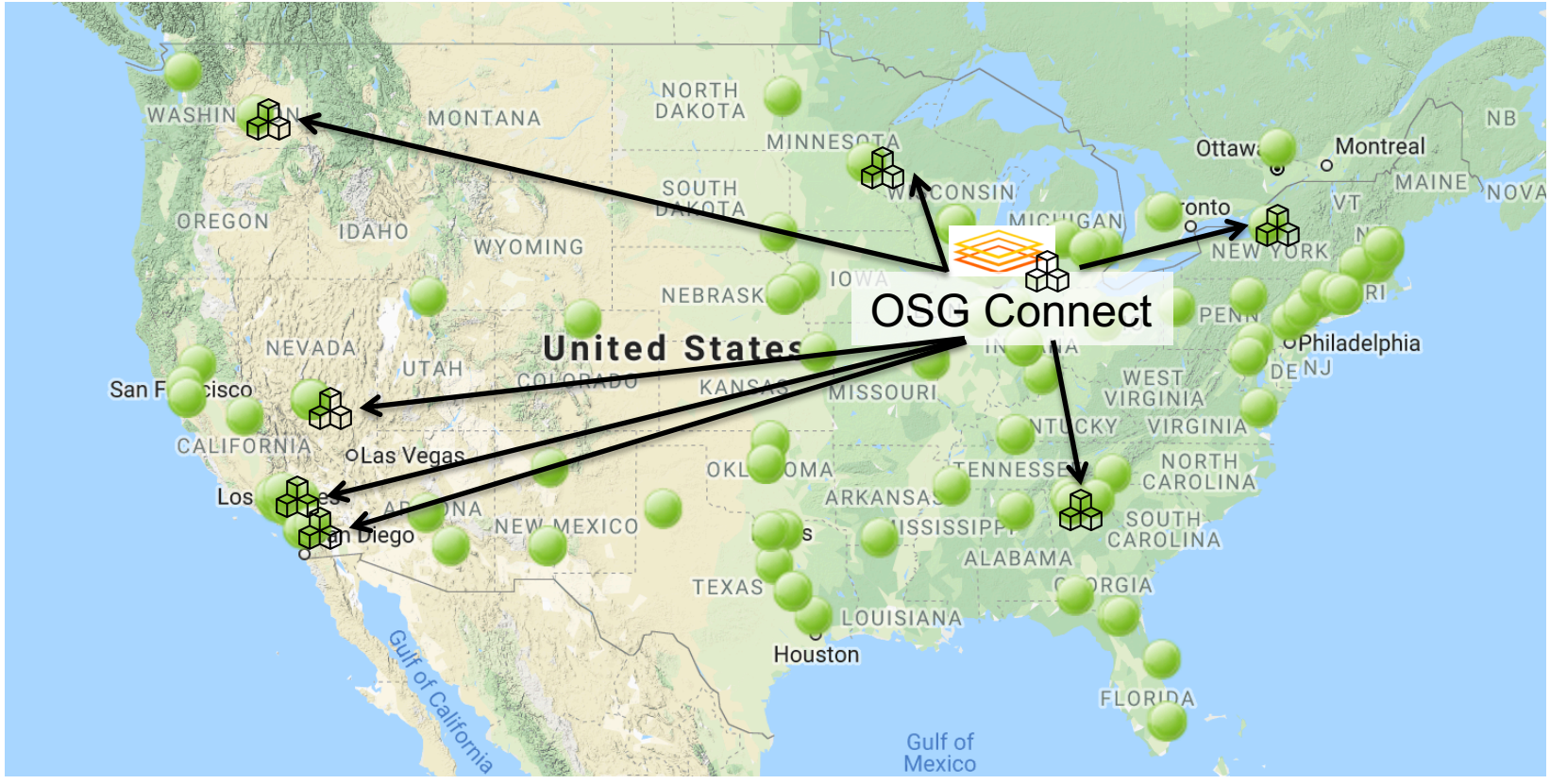
# Pre-existing Software

- The ideal for DHTC is to package and bring along your own software, but…
- You can use pre-existing software installations **if** the computers you're running on have your software installed (or access to a repository with the software).

# Pre-existing Software via OSG Connect

- On the Open Science Grid, jobs submitted from OSG Connect have access to a software repository maintained by OSG Connect staff.
- The software repository is available across the OSG.
- Software is accessed using “modules”.

# Software Across the OSG



# Module Commands

- See what modules are available

```
[~]$ module avail

[~]$ module spider lammps
```

- Load a module

```
[~]$ module load lammps/20180822
```

- See loaded modules

```
[~]$ module list
```

# Module Workflow

1. Find a module for your software
2. Write a wrapper script that loads the module and runs your code
3. Include requirements to ensure that your job has access to modules

```
requirements = (HAS_MODULES =?= true) &&
(OSGVO_OS_STRING == "RHEL7") && (OpSys == "LINUX")
```

# Module Workflow