Backpacking with Code: Software Portability for DHTC

Rachel Lombardi

Slides adapted from Christina Koch

OSG User School 2024

Goals For This Session

- Describe what it means to make software “portable.”
- Compare and contrast software portability techniques.
- Choose the best portability technique for your software.
- Build a portable software environment.
  - Follow steps to build a container
  - Compile code on a Linux computer

Introduction


Imagine that we asked you to
go home and bake a cake…

An Analogy

Photo by jschantz on flickr, CC-BY

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

On Your Computer

- You know what is there.
  - 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.

OSPool: Other People's Computers

The Challenge

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

Photo by F Deventhal on Wikimedia, CC-BY

On Someone Else's Computer

- What's already there?
  - Is R installed? Or Python? What about the packages you need?
- If the software you need is installed, do you know where it is or how to access it?
- Are you allowed to change whatever you want?

The Solution

- Imagine going camping or backpacking – what do you need to do to cook anywhere?
- Similarly: take your software with you to any computer.
- This is called making software portable.

Photo by andrew welch on Unsplash

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

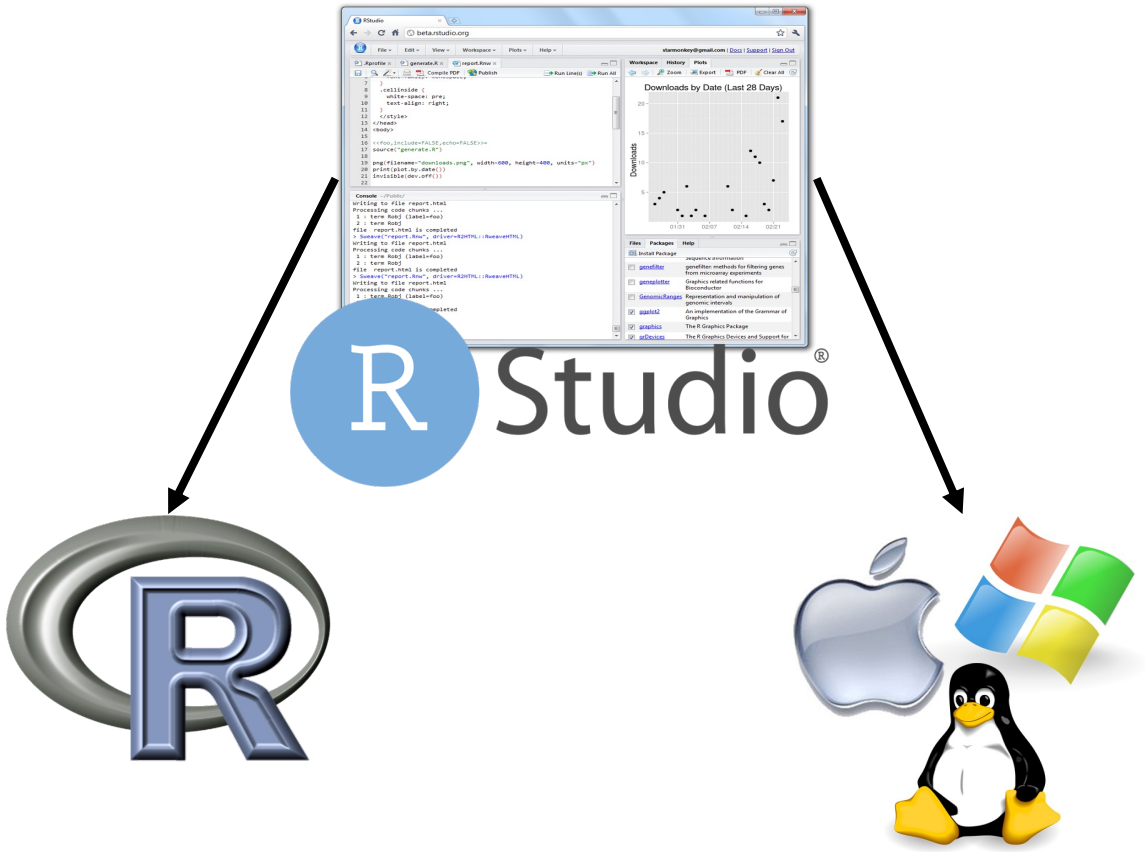
- Behind the scenes, any commands we run is referencing software files stored somewhere on the computer.

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

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

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

Many Software Files



The base software files will have dependencies:

- on other software
- on a specific operating system version

Finding Software Files

On a laptop, I can search for existing software…

On Linux, software is found by searching the “PATH”

```
$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/ada.lovelace/.local/bin:/home/ada.lovelace/bin
```

Finding Software Files

The “which” command will show you where a program lives:

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

$ which echo
/usr/bin/echo

$ ls -lh /usr/bin
```


Three Ways to Findability

- Provide a specific path to the files

```
$ ~/mypy/bin/python --version
2.7.7
```

- Add a files location to the PATH

```
$ export PATH=/Users/alice/mypy/bin:$PATH
$ which python
/Users/alice/mypy/bin/python
```

- Install to a default location (requires administrative privileges)

Making Software Portable

- When we install and use software, we are:
  1. Downloading or making the software files
     - Need to be compatible with Linux
     - Need to include other software files that are needed
  2. Making the software files findable
     - Putting them in a default location
     - Indicating where to find them in another way
- To make software portable, we have be able to do these two steps on any computer, where we are likely not an administrator.

Two Approaches

Containers

- Create complete, custom Linux environment, with software.

Bring Along Files

- Include individual software files with job, indicate where they are.

Two Approaches

Containers

- Create complete, custom Linux environment, with software.

Bring Along Files

- Include individual software files with job, indicate where they are.

The rest of the talk will go into this in detail.

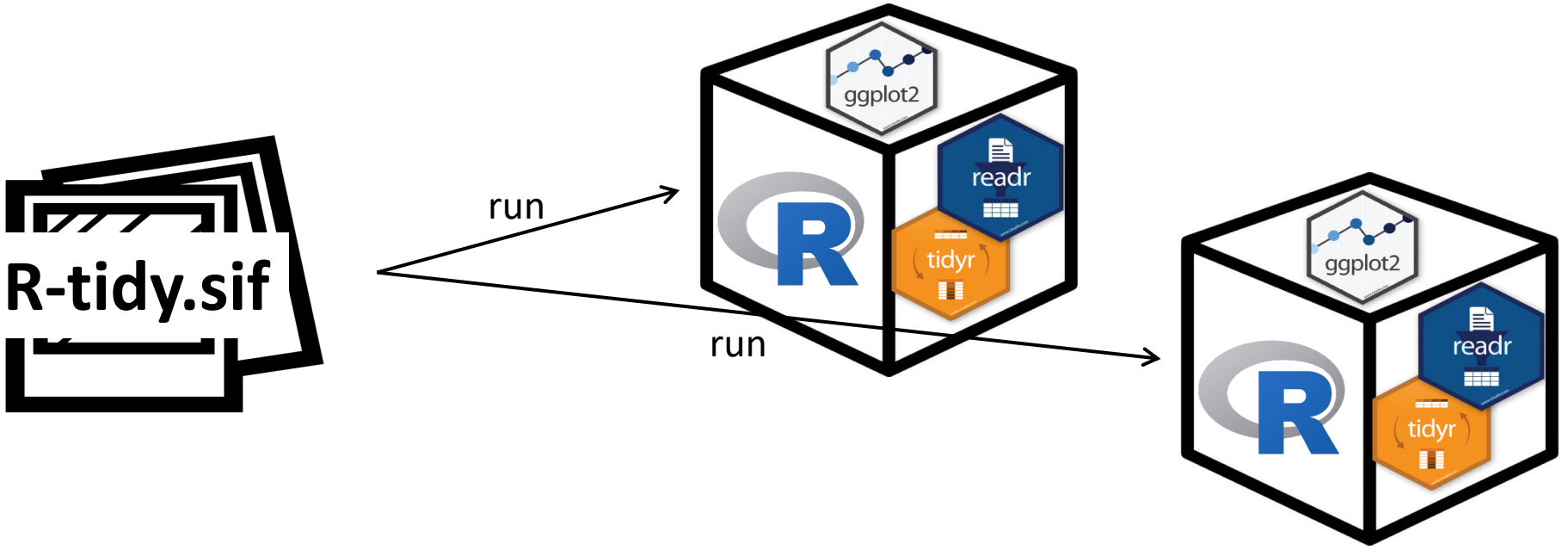
Option 1: Containers

Returning to Our Analogy…

Using a container is like bringing along a whole kitchen.

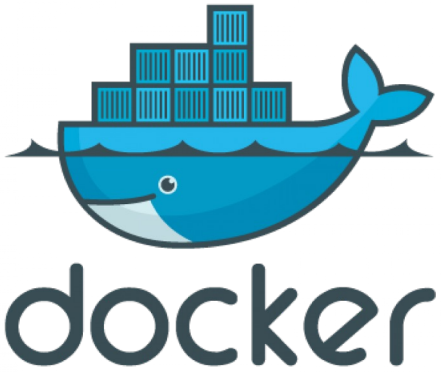
Containers

Containers are a tool for capturing an entire “environment” (software, libraries, operating system) into an “image” that can be run and used as the environment for a job



Credit: polaroid photos by Nick Bluth from the Noun Project

Container Technologies



Container system not used on most research computing systems, but has a huge catalog of existing containers.

Singularity/Apptainer containers are more commonly supported on research computing systems.
Apptainer is a fork of Singularity - we use the two names/products interchangeably on OSG services.

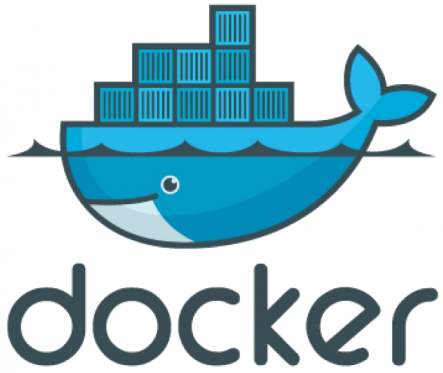
Container Technologies

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

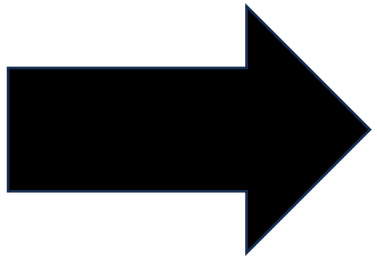
Use Existing Containers

- OSG provided: https://portal.osg-htc.org/documentation/htc_workloads/using_software/available-containers-list/
- OSG user provided (just a list, no descriptions): https://github.com/opensciencegrid/cvmfs-singularity-sync/blob/master/docker_images.txt
- Docker Hub: https://hub.docker.com/

Explore Containers



python:3.10



```
Apptainer> python3 --version
Python 3.10.14
```

```
$ apptainer shell docker://python:3.10
```

Demo

Print Python version on Access Point

```
ap40$ python3 --version
Python 3.9.19
```

Build and run an apptainer container running Python 3.10

```
$ apptainer shell docker://python:3.10
```

Print Python version inside Apptainer container

```
Apptainer> python3 --version
Python 3.10.14
```


Build Your Own Container

Definition File (cowsay.def)

```
Bootstrap: docker
From: python:3.10

%post
  pip install cowsay
```

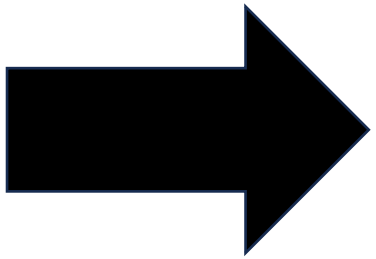
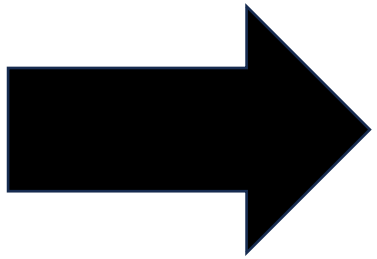


Image File (cowsay.sif)


```
$ apptainer build cowsay.sif cowsay.def
```

Explore Containers, Part 2



```
Singularity :~> cowsay -t "hello"

  _____
| Hello |
  =====
         \
          \
            ^__^
            (oo)\_______
            (__)\       )\/\
                ||----w |
                ||      ||
```

```
$ apptainer shell cowsay.sif
```

Definition File Details

```
Bootstrap: docker
From: ubuntu:20.04

%files
  install.R

%post
  apt install r-base
  Rscript install.R

%environment
  RHOME=/opt/R
```

What type of container to start with, and what is its name/where is it?

Copy in any files that are needed for installing software or running the container

Use installation tools or shell commands to install desired software or packages

Add environment variables to be used when the container is run.

Using Containers in Jobs

```
# For right now/testing
container_image = cowsay.sif
```

```
# Later this week
+SingularityImage = "osdf:///ospool/<apNN>/data/<USER>/my-container-v1.sif"
```


Why Use Containers

- **Consistent** and **complete**
  - Always the same software environment, with everything included
  - Good for sharing software among groups!
- Handles **complexity**, is **customizable**
  - As an “administrator” can control exactly what goes into the container and use built-in Linux installation tools
- **Cross-platform**
  - Can run (Docker) containers on Linux, Windows, Mac
- Easy to re-**create** (if you use Dockerfiles/definition files)

Learn How to Build/ Use Containers

- OSG User Documentation: https://portal.osg-htc.org/documentation/

- Videos:
  - https://www.youtube.com/watch?v=8fi7uSYlOdc
  - https://portal.osg-htc.org/documentation/support_and_training/training/osgusertraining/

Option 2: Bring Along Software Files

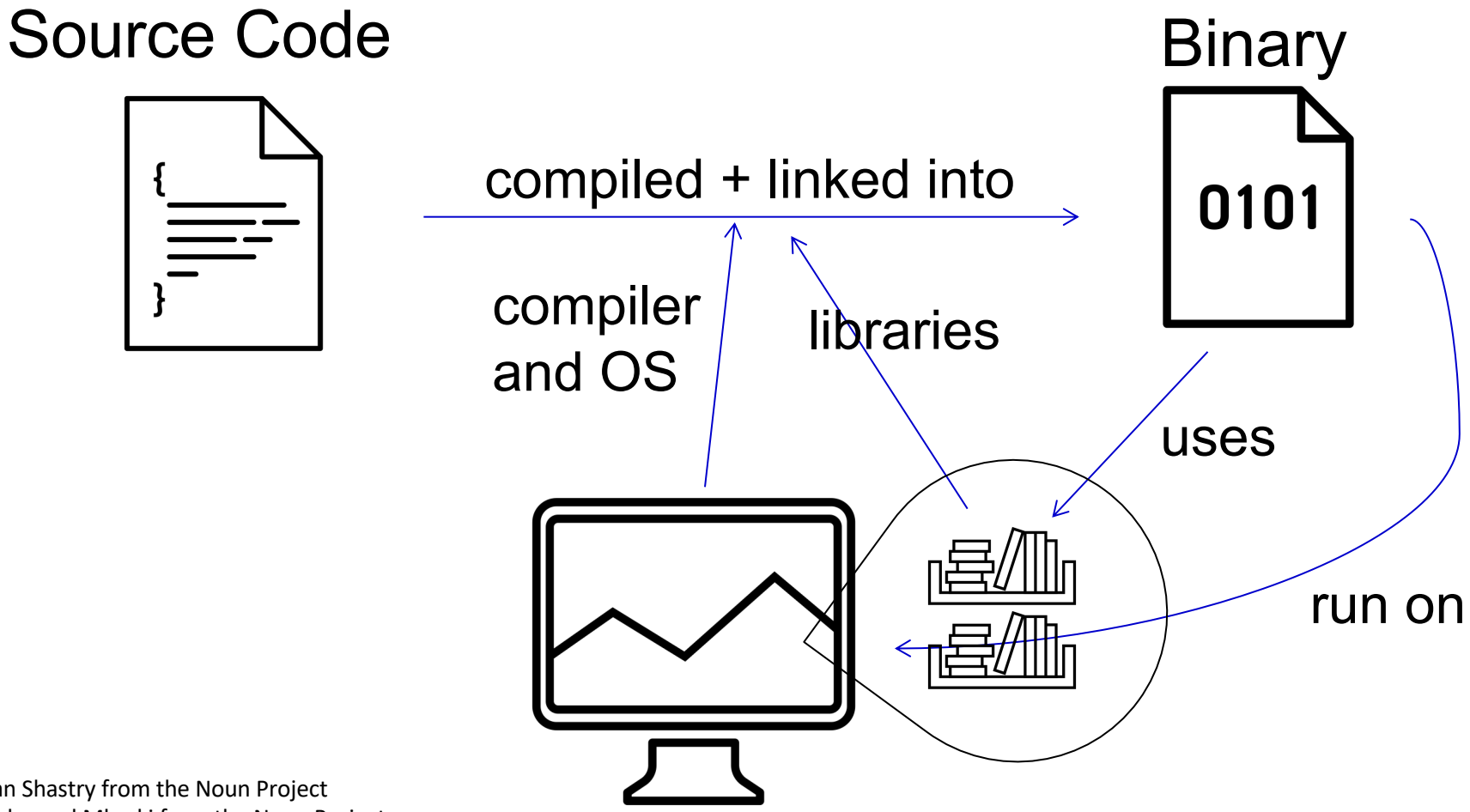
Back to the Kitchen Analogy…

A more flexible, but sometimes more challenging approach to software portability is to bring along a set of software files. This is more like taking a backpacking approach to a portable kitchen – just bringing the essentials in a bag.

Ways to Prepare Software Files

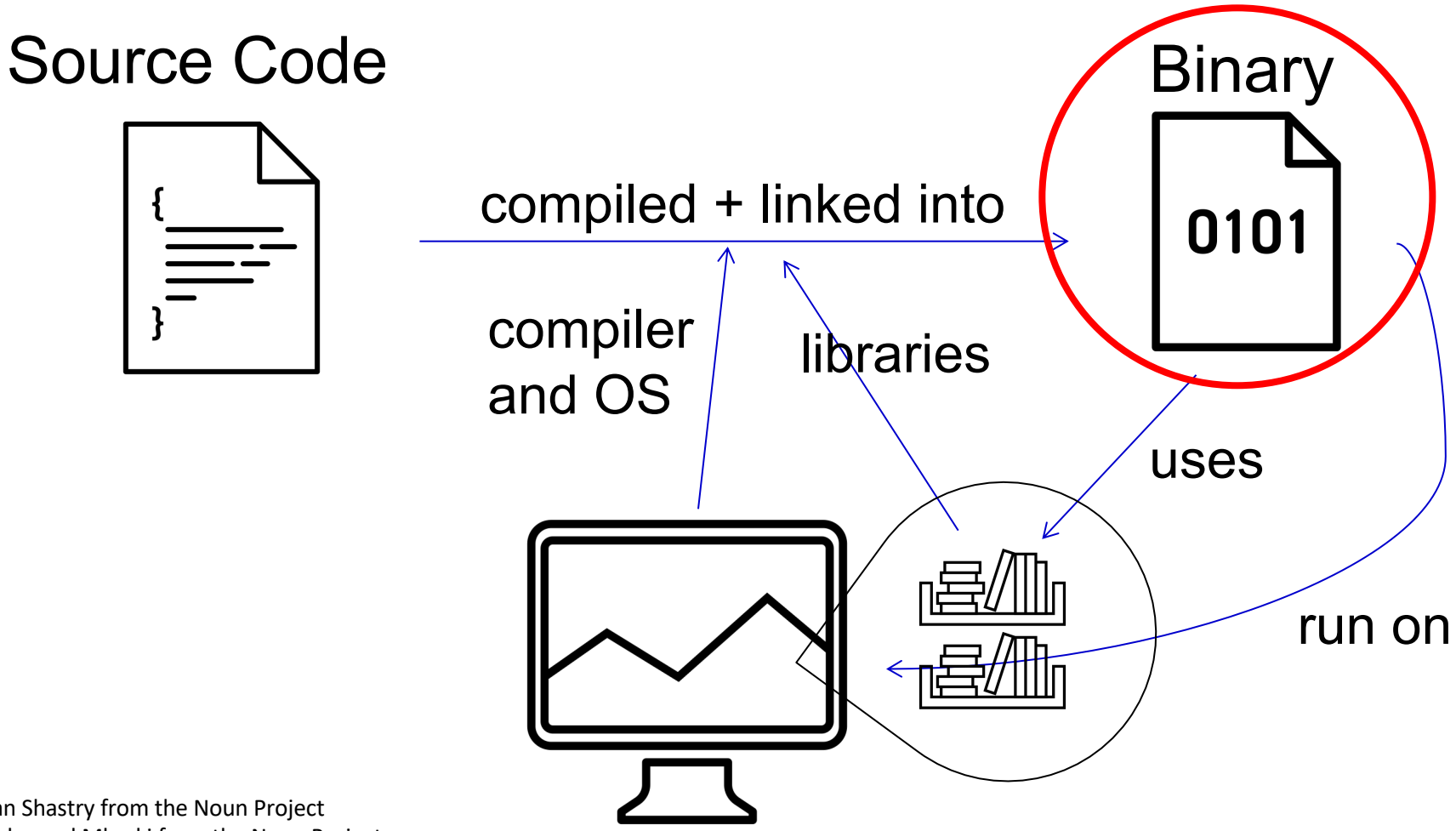
- Download pre-compiled software files
- Compile software yourself
  - Generate a single binary file
  - Create an installation with multiple binary files contained in a single folder
- We always need a “compiled” file of some kind, that is compatible with the version of Linux that is most common on the OSPool (Red Hat aka CentOS, Rocky, Alma)

What is Compilation?



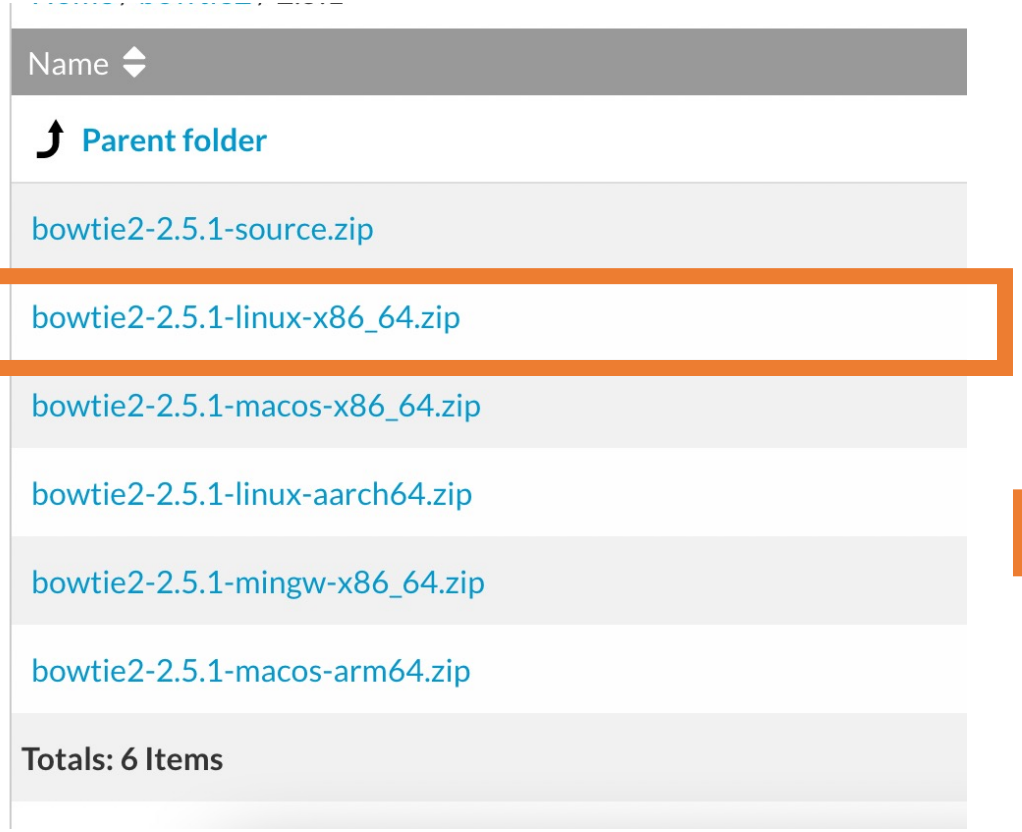
binary code by Kiran Shastry from the Noun Project
Source Code by Mohamed Mbarki from the Noun Project
Computer by rahmat from the Noun Project
books by Viral faisalovers from the Noun Project

What is Compilation?

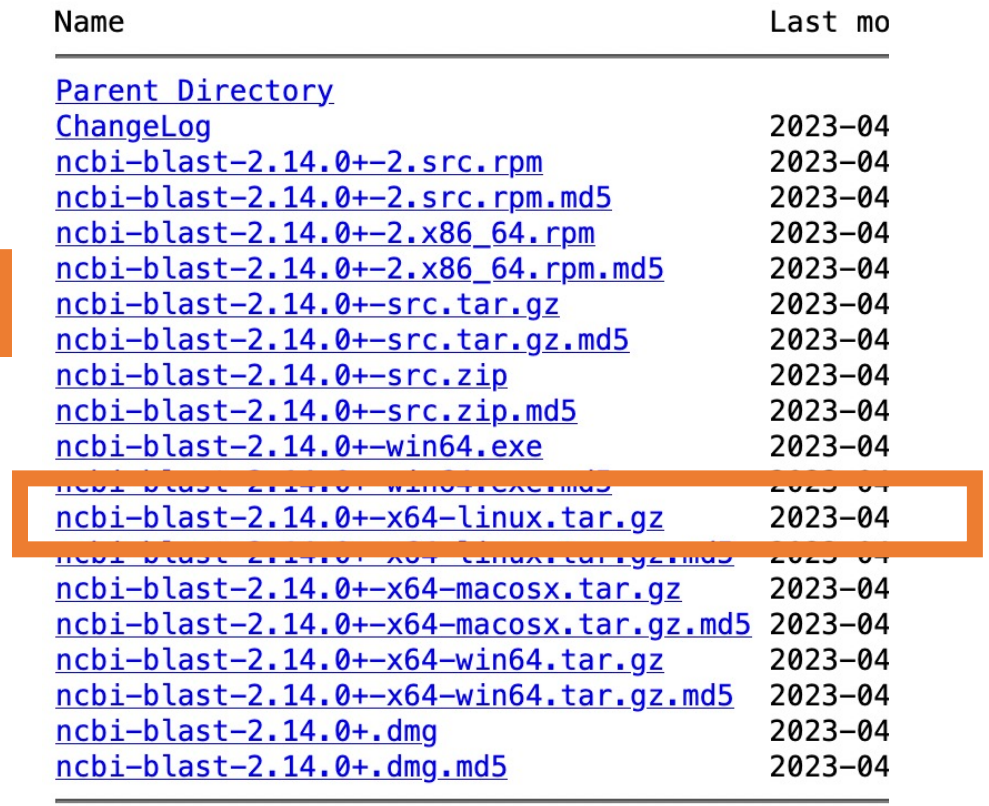


binary code by Kiran Shastry from the Noun Project
Source Code by Mohamed Mbarki from the Noun Project
Computer by rahmat from the Noun Project
books by Viral faisalovers from the Noun Project

Find Existing Software Files

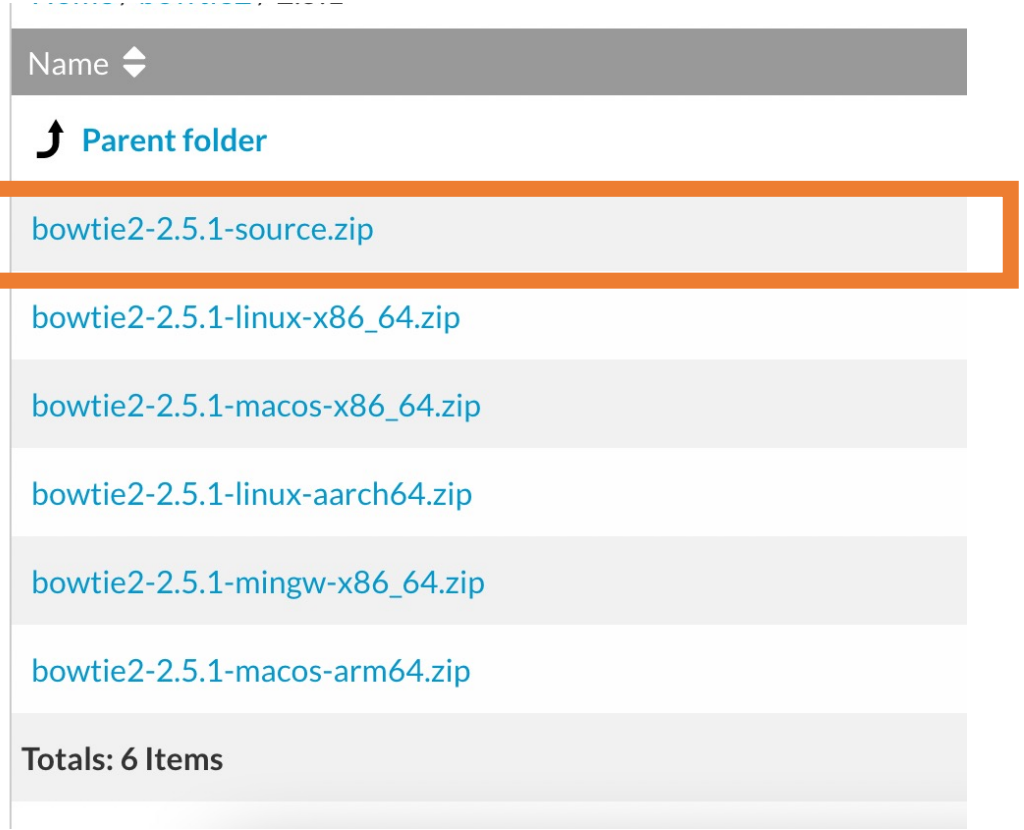


Name
Parent folder
bowtie2-2.5.1-source.zip
bowtie2-2.5.1-linux-x86_64.zip
bowtie2-2.5.1-macos-x86_64.zip
bowtie2-2.5.1-linux-aarch64.zip
bowtie2-2.5.1-mingw-x86_64.zip
bowtie2-2.5.1-macos-arm64.zip
Totals: 6 Items



Name	Last mo
Parent Directory	
ChangeLog	2023-04
ncbi-blast-2.14.0+-2.src.rpm	2023-04
ncbi-blast-2.14.0+-2.src.rpm.md5	2023-04
ncbi-blast-2.14.0+-2.x86_64.rpm	2023-04
ncbi-blast-2.14.0+-2.x86_64.rpm.md5	2023-04
ncbi-blast-2.14.0+-src.tar.gz	2023-04
ncbi-blast-2.14.0+-src.tar.gz.md5	2023-04
ncbi-blast-2.14.0+-src.zip	2023-04
ncbi-blast-2.14.0+-src.zip.md5	2023-04
ncbi-blast-2.14.0+-win64.exe	2023-04
ncbi-blast-2.14.0+-x64-linux.tar.gz	2023-04
ncbi-blast-2.14.0+-x64-macosx.tar.gz	2023-04
ncbi-blast-2.14.0+-x64-macosx.tar.gz.md5	2023-04
ncbi-blast-2.14.0+-x64-win64.tar.gz	2023-04
ncbi-blast-2.14.0+-x64-win64.tar.gz.md5	2023-04
ncbi-blast-2.14.0+.dmg	2023-04
ncbi-blast-2.14.0+.dmg.md5	2023-04

Download Source and Compile



Name
Parent folder
bowtie2-2.5.1-source.zip
bowtie2-2.5.1-linux-x86_64.zip
bowtie2-2.5.1-macos-x86_64.zip
bowtie2-2.5.1-linux-aarch64.zip
bowtie2-2.5.1-mingw-x86_64.zip
bowtie2-2.5.1-macos-arm64.zip
Totals: 6 Items

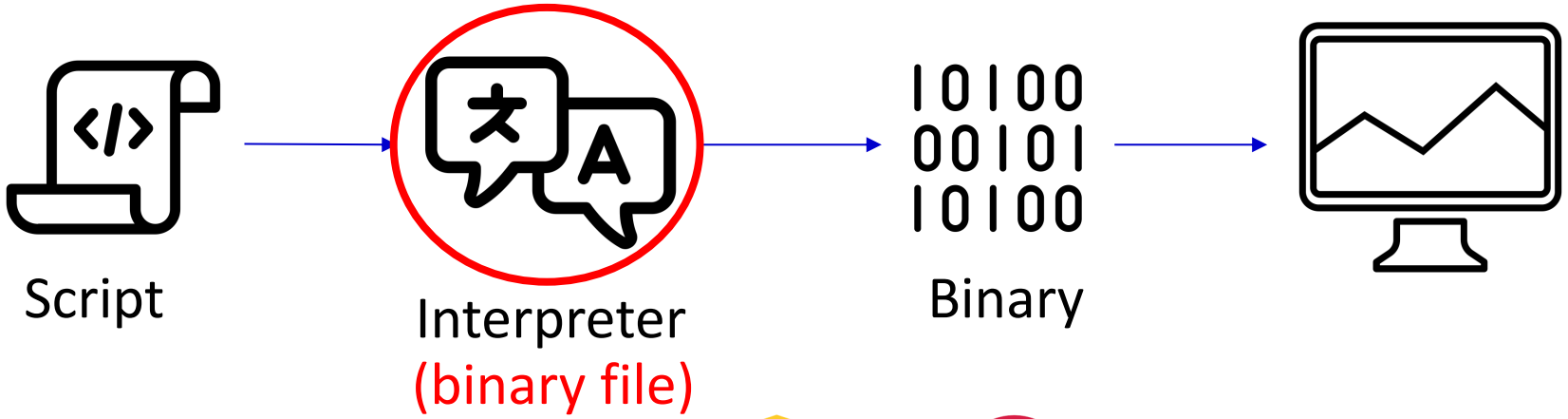
Name	Last mo
Parent Directory	
ChangeLog	2023-04
ncbi-blast-2.14.0+-2.src.rpm	2023-04
ncbi-blast-2.14.0+-2.src.rpm.md5	2023-04
ncbi-blast-2.14.0+-2.x86_64.rpm	2023-04
ncbi-blast-2.14.0+-src.tar.gz	2023-04
ncbi-blast-2.14.0+-src.zip	2023-04
ncbi-blast-2.14.0+-src.zip.md5	2023-04
ncbi-blast-2.14.0+-win64.exe	2023-04
ncbi-blast-2.14.0+-win64.exe.md5	2023-04
ncbi-blast-2.14.0+-x64-linux.tar.gz	2023-04
ncbi-blast-2.14.0+-x64-linux.tar.gz.md5	2023-04
ncbi-blast-2.14.0+-x64-macosx.tar.gz	2023-04
ncbi-blast-2.14.0+-x64-macosx.tar.gz.md5	2023-04
ncbi-blast-2.14.0+-x64-win64.tar.gz	2023-04
ncbi-blast-2.14.0+-x64-win64.tar.gz.md5	2023-04
ncbi-blast-2.14.0+.dmg	2023-04
ncbi-blast-2.14.0+.dmg.md5	2023-04

Compiling Code

- Use a compiler (like gcc) directly
  - Can use options to control compilation process
- More common – a three-step build process:
    1. ./configure # or cmake # configures the build process
    2. make # does the compilation and linking
    3. make install # moves compiled files to specific location(s)
  - Installation options (like where to install) are usually set at the configure/cmake step

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.



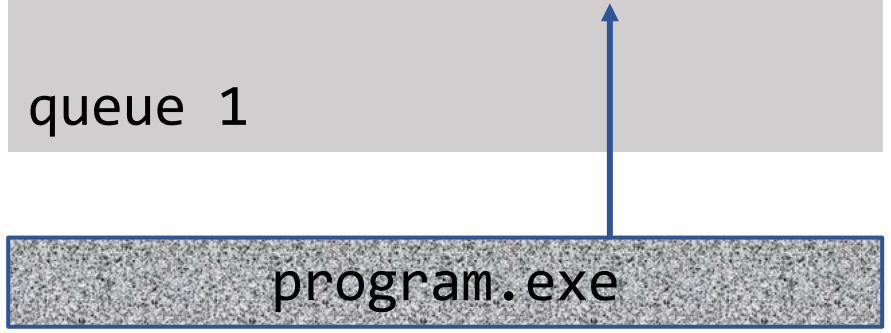
Using Software Files in Jobs

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
```

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

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

Why Bring Along Software Files

- **No Installation Required** (sometimes)
  - Software releases that are pre-compiled for Linux don't need any compiling or installation!
- **No Docker/Apptainer Required**
  - Not all computers in the OSPool support containers
- **Use Familiar Environments**
  - This approach can work with conda environments

Next Steps

Using Software in a DHTC System

- Create/find software files:
  - Put them in a container (or find a container that has them already)
  - Download them in a tar.gz or .zip file
  - Make a tar.gz file with code you have built
- Account for all dependencies, files, and requirements in the submit file.
- If needed, write a wrapper script to set up the environment when the job runs.

Two Approaches

Containers

- Files
  - Choose a base Linux version
  - Use built-in installation tools
  - Compile software files
- Findability
  - Files can be in default location
  - Can reference custom location or use the PATH variable

Bring Along Files

- Files
  - Download a tar.gz file with Linux-compatible files
  - Compile software files on Linux system + zip them up
- Findability
  - Reference custom location or use the PATH variable

Which Approach to Use?

Containers

- Container already exists with software
- Installation is complex, requires many dependencies
- Special hardware (GPUs)
- Want to share installation
- Good general option

Bring along files

- Software already exists as a tar.gz download
- Software that produces a single binary file, with few dependencies
- Easy to zip installation folder

Work Time

- Go through the introductory exercises
- Then, choose an approach for *your* software and try to find or make a portable version for OSPool jobs.

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.

Appendix: Container and Compiling Tips

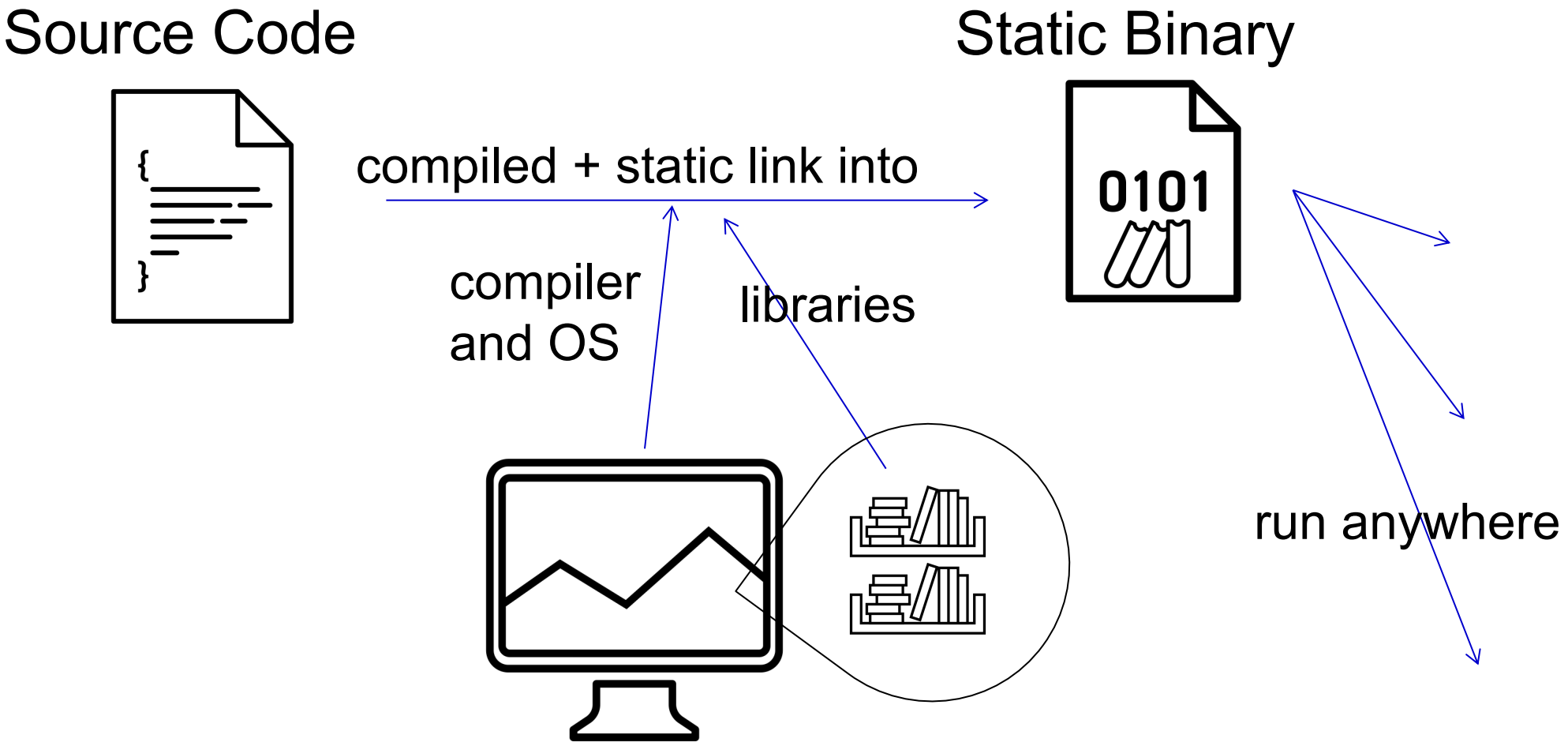
Best Practices in Using Containers on OSG

- Don’t use the **latest** tag in images
- Use **version number**/specific names in the images
- Test images with **apptainer shell**
- **Unique** image name eliminates the risk of running a job using previous versions due to stashing.

Where to install software?

- Do not use $HOME, /root or /srv
  - Container will run as some user we do not know yet, so $HOME is not known and will be mounted over
  - /root is not available to unprivileged users
  - /srv is used a job cwd in many cases
- /opt or /usr/local are good choices

Static Linking



Book by Aleksandr Vector from the Noun Project