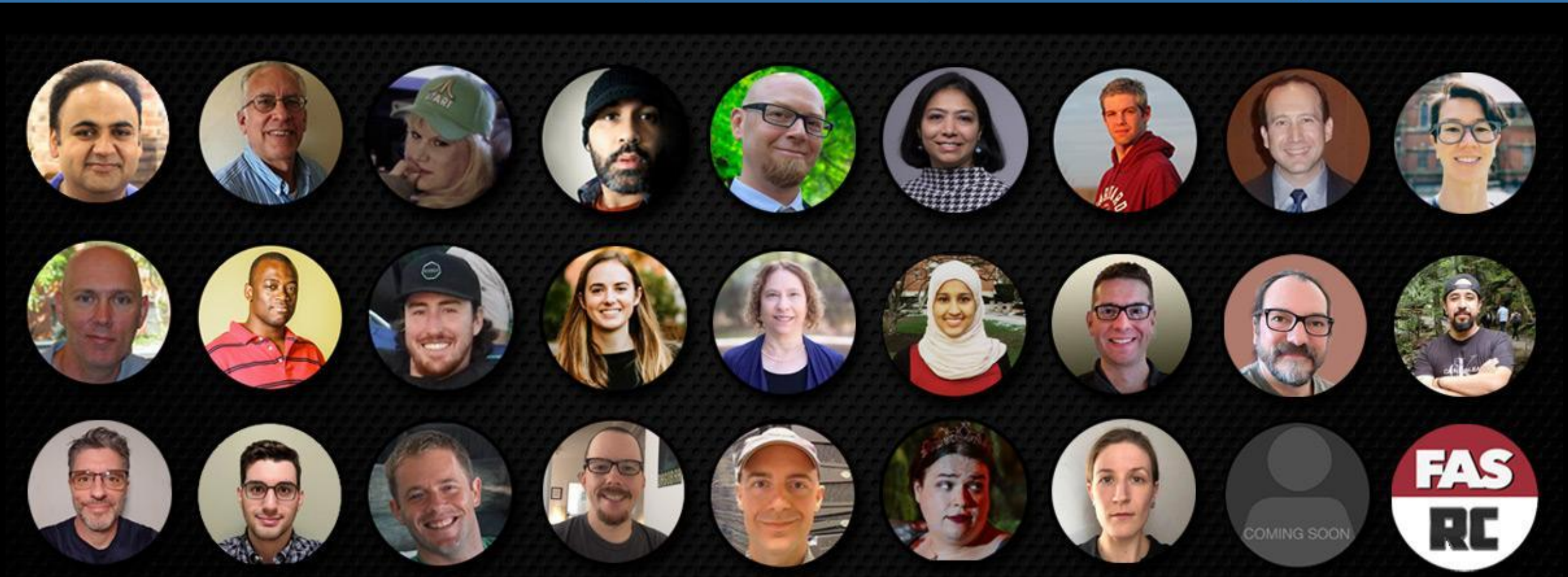


Using Singularity Containers on the FASRC clusters

Objectives

- Software difficulties on HPC systems
- Why use containers?
- Containers overview
- Singularity containers
  - How to build your own Singularity containers
  - How to run Singularity containers on Cannon/FASSE
  - Bind mounts

Software difficulties on HPC systems

- Building software is often complicated, particularly on a shared and multi-tenant system
- Some applications might need dependencies that are not readily available and/or complex to build from source on a shared system
- Applications requiring software compatible with a different OS than what is offered on the cluster, e.g., Rocky Linux vs Ubuntu
- Reproducibility:
  - Different researchers may install different versions of an application and/or dependencies
- Portability & system-agnostic
  - Hard to share workflows & pipelines, with external collaborators, on another HPC system

Why use containers?

Overcome software stack, reproducibility and portability difficulties

- To create a virtual environment that contains all the software stack needed
- They package in one single file all necessary dependencies
- You can choose a linux operating system that is different than host (e.g. Ubuntu)
- Easy to publish
- Portable
- Reproducible

Why use containers?

- In 1990s, one OS with one app could be deployed on a single server. For more apps or different OS, additional servers were required
- In 2000s, virtualization technology used a software, hypervisor, to split the server to host multiple OS. But still only 1 app/OS
- Decade later, containerization allowed each app to be in its own container and single OS to host multiple containers/apps
- Makes servers efficient and app deployment faster
- See container animation

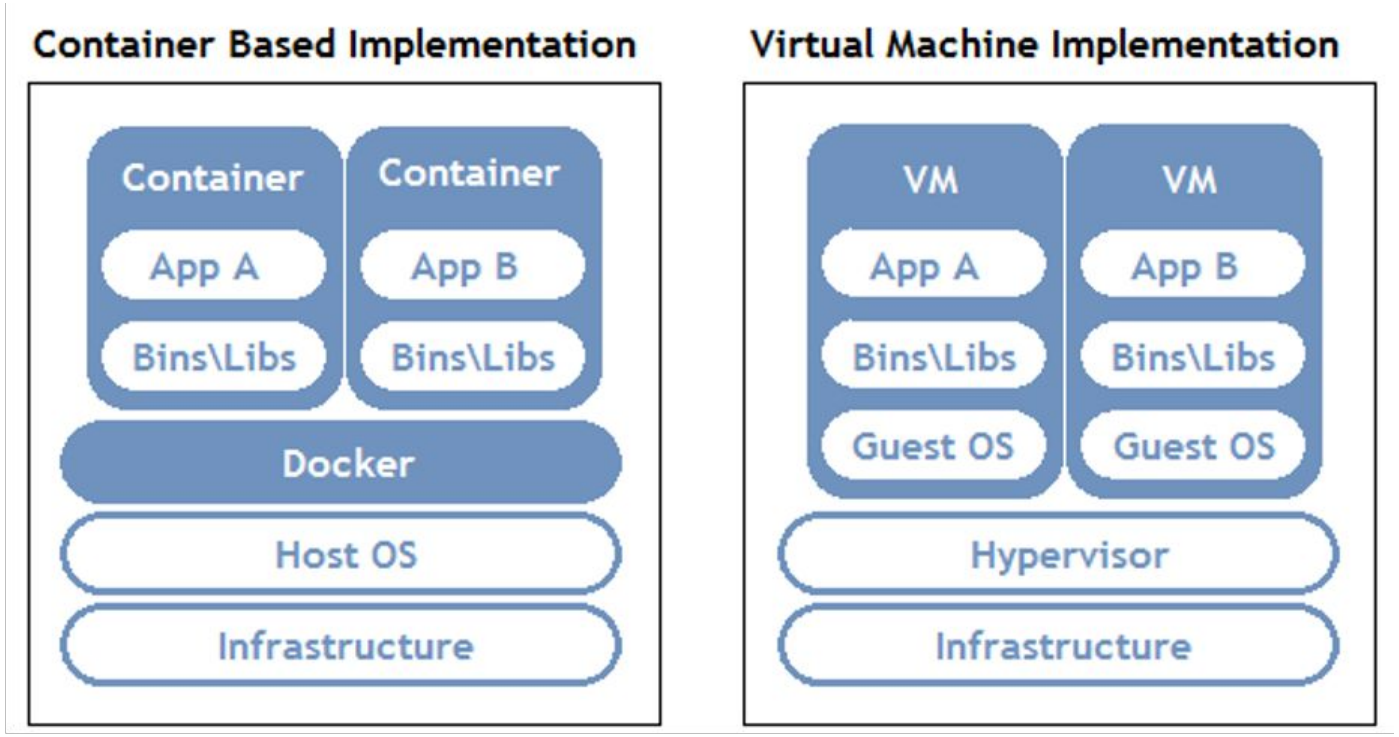
Virtual machines (VMs) vs. Containers

Virtual Machines	Containers
Very flexible -- for example, run Windows on MacOS	Less flexible Only Linux systems
Heavyweight -- need to install all files of virtual environment	Very lightweight -- uses the kernel of host OS

Adapted from LSU Singularity training slides:
http://www.hpc.lsu.edu/training/weekly-materials/2022-Fall/HPC_Singularity_Fall2022.pdf

Virtual machines (VMs) vs. Containers

- Abstraction of resources at OS instead of hardware level
- Shares host OS kernel
- Results in faster, lightweight instances with application portability
- Consists of an entire runtime environment – an application + its dependencies (libraries, binaries, configuration files, etc.)

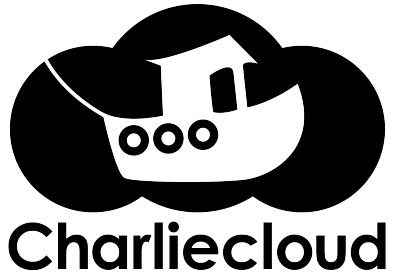


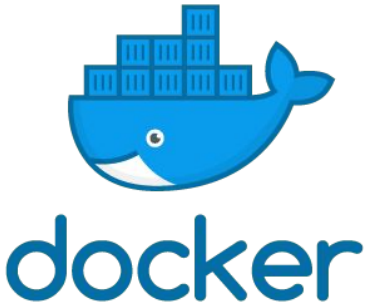
https://www.aquasec.com/cloud-native-academy/docker-container/docker-architecture/

Container vocabulary

- SingularityCE, Apptainer, Docker – the software that creates the container
  - As in “SingularityCE” or “Apptainer” or “Docker”
- Image
  - a compressed, usually read-only file that contains an OS and specific software stack
  - provides a template for a container
  - Examples: “Build a Matlab2021a image”, “Build an Alphafold image”, “Build an OpenFOAM image”
- Container
  - The technology: “containers vs. virtual machines”; is a running application
  - An instance of an image
    - Example: “process my data in a Singularity container of Matlab”; build an image & run a container using that
- Host – computer/supercomputer/laptop where the container is run

Containers





Apptainer

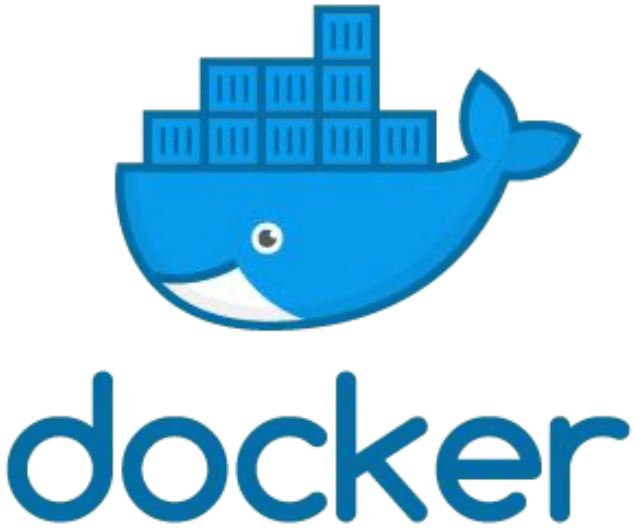
SingularityCE

<u>HPC Oriented:</u>

SingularityCE,
Charliecloud, Shifter

- WLM compatible
- No privilege escalation

Docker vs. SingularityCE



- Assumes user has root (admin) privileges on the host system
- Not designed for HPC and multi-tenant systems

- Assumes user **does not** have root (admin) privileges on the host system
- Designed for HPC and multi-tenant systems

SingularityCE (Community Edition)

- Open-source container software: [SingularityCE | Sylabs]
- Specifically designed for HPC systems (i.e. multi-tenant systems)
  - No root (admin) privileges
- Package applications with their dependencies and workflow into a single file
- Singularity, SingularityCE, Apptainer
  - Singularity: deprecated since 2021
  - SingularityCE and Apptainer: branches/children of Singularity since 2021
  - SingularityCE: maintained by Sylabs since May 2021
  - Apptainer: Singularity open source project & maintained by the Linux Foundation since Nov 2021

How to build SingularityCE images

- SingularityCE is best on compute nodes!!!
  - Cannon: request interactive job using the `salloc` command
  - FASSE: does not allow `salloc` – request a Remote Desktop job on FASSE Open OnDemand and launch a terminal
  - For details, see SingularityCE on the clusters
- Follow docs: https://github.com/fasrc/User_Codes/blob/master/Singularity_Containers/README.md#build-your-own-singularityce-container

SingularityCE workflow

Once: Build Singularity image with one of the following methods

1. Pull (i.e. download) existing container from SingularityCE Container Library
2. Pull existing Docker container from DockerHub (downloads as Singularity container)
3. Build a SingularityCE container from a Singularity definition file directly on Cannon/FASSE – unprivileged build with `proot`
4. Build a SingularityCE container from a local Singularity definition file using option `--remote`. This will build an image on Sylabs cloud which is automatically downloaded to Cannon/FASSE

Many times: Use image

1. Pull image from SingularityCE Container Library

SingularityCE library (https://cloud.sylabs.io/library)

```
# request interactive job
[jharvard@boslogin06 ~]$ salloc --partition test --time 01:30:00 -c 4 --mem 16G
[jharvard@holy8a24302 ~]$ mkdir -p ~/singularity
[jharvard@holy8a24302 ~]$ cd singularity/

# pull laughing cow container container
[jharvard@holy8a24302 singularity]$ singularity pull library://library/default/ubuntu
INFO:    Downloading library image
28.4MiB / 28.4MiB [=============================================] 100 % 3.5 MiB/s 0s

# pull ubuntu container
[jharvard@holy8a24302 singularity]$ singularity pull library://library/default/ubuntu
INFO:    Downloading library image
28.4MiB / 28.4MiB [=============================================] 100 % 3.5 MiB/s 0s
```

2. Pull image from DockerHub (Example 1)

DockerHub (https://hub.docker.com/)

you may choose the image name

```
# pull laughing cow container container
[jharvard@holy8a24302 singularity]$ singularity pull lolcow_from_docker.sif
docker://sylabsio/lolcow
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
INFO:    Fetching OCI image...
27.2MiB / 27.2MiB [=============================================] 100 % 28.6 MiB/s 0s
45.8MiB / 45.8MiB [=============================================] 100 % 28.6 MiB/s 0s
INFO:    Extracting OCI image...
INFO:    Inserting Singularity configuration...
INFO:    Creating SIF file...
```

To save space while pulling a container:
export SINGULARITY_CACHEDIR=/scratch/$USER/SINGULARITY_CACHE
OR, --disable cache in singularity pull command

export SINGULARITY_TMPDIR=/tmp

2. Pull image from DockerHub (Example 2, part 1/2)

1. Go to DockerHub (https://hub.docker.com/) and search for a container
2. For example, alphafold
3. Click on tacc/alphafold
4. Click on the “Tags” tab
5. Select the version that you need. You will see something like

   ```
   docker pull tacc/alphafold:2.3.2
   ```
6. Singularity syntax to pull the image:

   ```
   singularity pull docker://<organization>/<repository>:<version>
   ```

   For tacc/alphafold, this becomes

   ```
   singularity pull docker://tacc/alphafold:2.3.2
   ```

2. Pull image from DockerHub (Example 2, part 2/2)

```
[jharvard@holy8a24302 singularity]$ singularity pull docker://tacc/alphafold:2.3.2
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
INFO:    Fetching OCI image...
25.5MiB / 25.5MiB [=============================================] 100 % 28.6 MiB/s 0s
838.4MiB / 838.4MiB [===========================================] 100 % 28.6 MiB/s 0s
6.9MiB / 6.9MiB [===============================================] 100 % 28.6 MiB/s 0s
10.3MiB / 10.3MiB [=============================================] 100 % 28.6 MiB/s 0s
1.4GiB / 1.4GiB [===============================================] 100 % 28.6 MiB/s 0s
430.3KiB / 430.3KiB [===========================================] 100 % 28.6 MiB/s 0s
1.4GiB / 1.4GiB [===============================================] 100 % 28.6 MiB/s 0s
210.8MiB / 210.8MiB [===========================================] 100 % 28.6 MiB/s 0s
33.0MiB / 33.0MiB [=============================================] 100 % 28.6 MiB/s 0s
31.9MiB / 31.9MiB [=============================================] 100 % 28.6 MiB/s 0s
31.9MiB / 31.9MiB [=============================================] 100 % 28.6 MiB/s 0s
930.1MiB / 930.1MiB [===========================================] 100 % 28.6 MiB/s 0s
221.5MiB / 221.5MiB [===========================================] 100 % 28.6 MiB/s 0s
INFO:    Extracting OCI image...
INFO:    Inserting Singularity configuration...
INFO:    Creating SIF file...
```

3. Create container with `proot` (part 1/4)

Documentation:
https://github.com/fasrc/User_Codes/blob/master/Singularity_Containers/README.md#build-a-singularityce-container-from-a-singularity-definition-file

1. Download `proot` in the directory `~/bin`
2. Ensure `~/bin` (e.g. `/n/home01/jharvard/bin`) is included in your `PATH`. If not, add it
3. Write/obtain a definition file
4. Build SingularityCE image

3. Create container with `proot` (part 2/4)

```
# make ~/bin directory
[jharvard@holy2c02302 ~]$ mkdir -p ~/bin

# change to ~/bin directory, download proot, and change permissions to make it executable
[jharvard@holy2c02302 ~]$ cd ~/bin
[jharvard@holy2c02302 bin]$ curl -LO https://proot.gitlab.io/proot/bin/proot
[jharvard@holy2c02302 bin]$ chmod +x ./proot

# print PATH
[jharvard@holy2c02302 ~]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/puppetlabs/bin:/n/home01/jharvard/.loc
al/bin

# since /n/home01/jharvard/bin is not part of PATH, add it
[jharvard@holy2c02302 ~]$ export PATH=$PATH:~/bin
```

3. Create container with `proot` (part 3/4)

Singularity definition file `lolcow.def`

```
Bootstrap: docker
From: ubuntu:22.04
```

Header: base container image

```
%labels
    Author: J. Harvard
```

Label: container metadata

```
%post
    apt-get -y update
    apt-get -y install cowsay lolcat
```

Post: section where you add your own packages

```
%environment
    export LC_ALL=C
    export PATH=/usr/games:$PATH
```

Environment: set environmental variables

```
%runscript
    date | cowsay | lolcat
```

Runscript: commands run when you use "`singularity run`"

3. Create container with `proot` (part 4/4)

```
# build singularity image
[jharvard@holy2c02302 ~]$ singularity build lolcow.sif lolcow.def
INFO:    Using proot to build unprivileged. Not all builds are supported. If build fails, use
--remote or --fakeroot.
INFO:    Starting build...
Getting image source signatures
Copying blob 76769433fd8a done

... omitted output ...

Running hooks in /etc/ca-certificates/update.d...
done.
INFO:    Adding environment to container
INFO:    Adding runscript
INFO:    Creating SIF file...
INFO:    Build complete: lolcow.sif
```

Limitations of builds with `proot`

`proot`'s emulation of the `root` user is not complete. Limitations include:

- Header
  - Do not support `arch` / `debootstrap` / `yum` / `zypper` bootstraps
  - Use `localimage`, `library`, `oras`, or one of the `docker`/`oci` sources.
- Do not support `%pre` and `%setup` sections of definition files
- Run the `%post` sections of a build in the container as an emulated `root` user
- Are subject to any restrictions imposed in `singularity.conf`
- Incur a performance penalty due to the``ptrace``-based interception of syscalls used by `proot`
- May fail if the `%post` script requires privileged operations that `proot` cannot emulate.

From https://docs.sylabs.io/guides/latest/user-guide/build_a_container.html#unprivilged-proot-builds

How to run Singularity images

Documentation:
https://github.com/fasrc/User_Codes/blob/master/Singularity_Containers/working_with_images.md

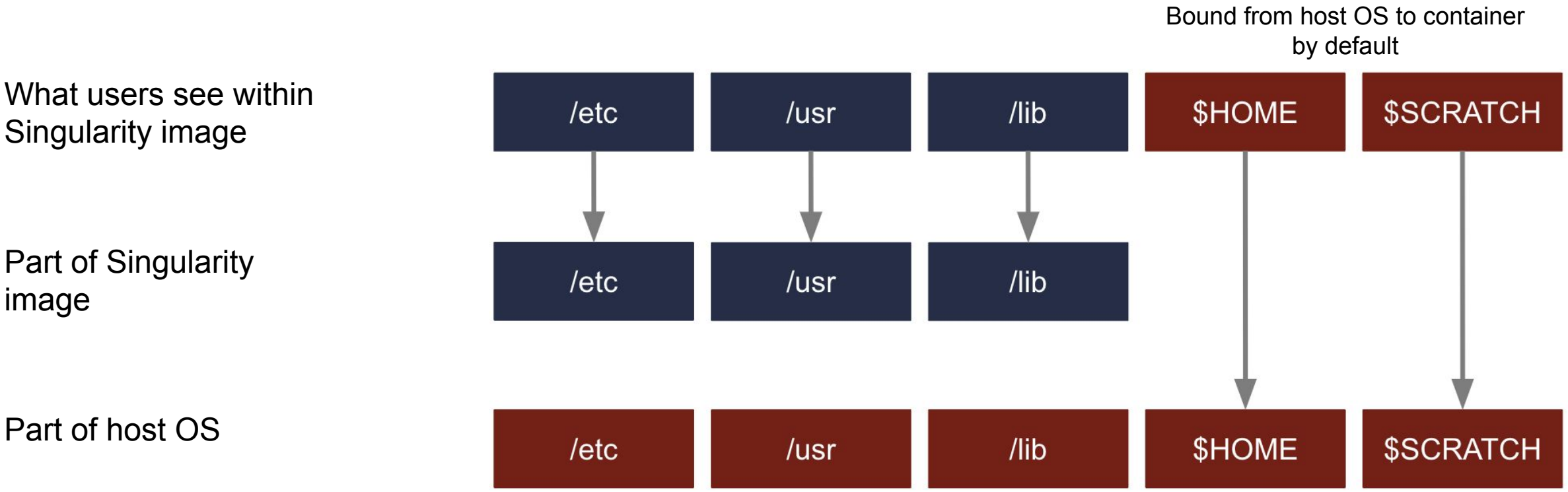
Singularity syntax

```
singularity <command> [options] <container_image.sif>
```

Commands

- `shell`: run an interactive bash shell inside the container
- `exec`: execute a command
- `run`: launch the runscript (from definition file)

Singularity and host file system



To allow other filesystems to be accessible from container, use `--bind` option: `--bind src:dest`

- See Accessing files from a container
- https://docs.sylabs.io/guides/3.7/user-guide/bind_paths_and_mounts.html

Singularity with GPU

Documentation:

https://github.com/fasrc/User_Codes/blob/master/Singularity_Containers/working_with_images.md#gpu-example

Parallel computing and Singularity

- OpenMP examples:
  https://github.com/fasrc/User_Codes/tree/master/Singularity_Containers/OMP_Apps

- MPI examples:
  https://github.com/fasrc/User_Codes/tree/master/Singularity_Containers/MPI_Apps

FASRC Upcoming Trainings

Training calendar: https://www.rc.fas.harvard.edu/upcoming-training/

VSCode on the FASRC cluster

Training is focused on connecting to Cannon via VSCode (Visual Studio Code) from your local machine.

Audience: Users who are familiar with command line, HPC systems, vscode and would like to connect to Cannon using vscode.

Note: All topics below are a brief overview to get connected to Cannon using VSCode

Objectives:

1. Run VSCode on a login node
2. Run VSCode on a compute node

Resources and help

- Documentation
  - https://docs.rc.fas.harvard.edu/
  - Singularity docs: https://github.com/fasrc/User_Codes/tree/master/Singularity_Containers
- Portal
  - http://portal.rc.fas.harvard.edu/rcrt/submit_ticket
- Email
  - rchelp@rc.fas.harvard.edu
- Office Hours
  - Wednesday noon-3pm https://harvard.zoom.us/j/255102481
- Consulting Calendar
  - https://www.rc.fas.harvard.edu/consulting-calendar/
- Training
  - https://www.rc.fas.harvard.edu/upcoming-training/

Survey

Please, fill out our course survey. Your feedback is essential for us to improve our trainings!!

http://tinyurl.com/FASRCsurvey

Thank you!