OAK RIDGE National Laboratory

Containers on Summit

Subil Abraham

OLCF January 2023 User Conference Call

2023-01-25

U.S. DEPARTMENT OF ENERGY

https://docs.olcf.ornl.gov/software/containers_on_summit.html

docs.olcf.ornl.gov -> Software -> Containers on Summit

Goals for this presentation

- Why you might want containers
- What containers are
- How to build and run containers on Summit
- Some pitfalls when building and running containers
- Where containers are not useful

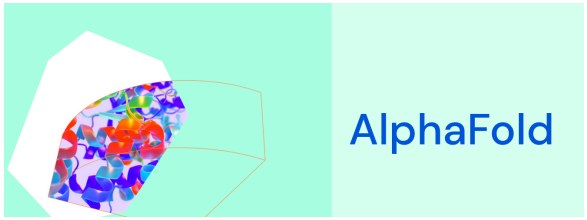
Containers in HPC: Why even bother?

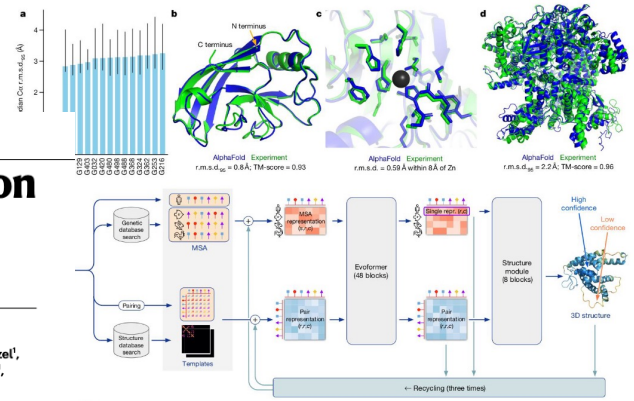
- Containers package your application along with its software environment
  - Removes dependence on host system's stack
  - Finer control over system level libraries also
- For some software, container is the only option
- Potentially easier to port to cloud or other systems*
- Just hand a container image to a new user on your project
  - Save time setting up their environment from scratch

*Provided they're the same architecture

Alphafold2

- a landmark protein folding application that predicted protein structures with DL
- Required Bazel, which hardcoded compiler paths
- Built with Podman and Singularity, scaled on Summit with Dask
- GT + ORNL studying environmentally relevant proteomes w/ Alphafold2



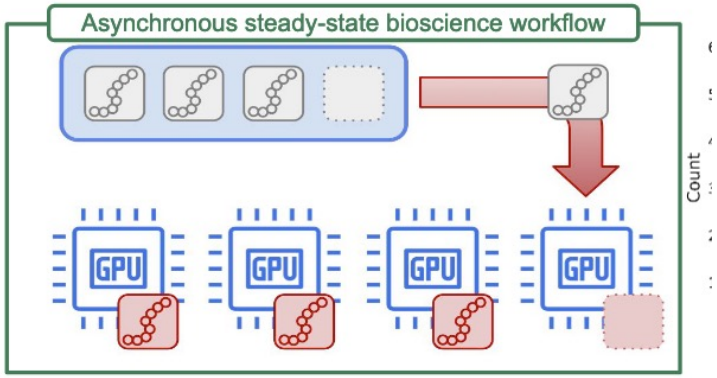


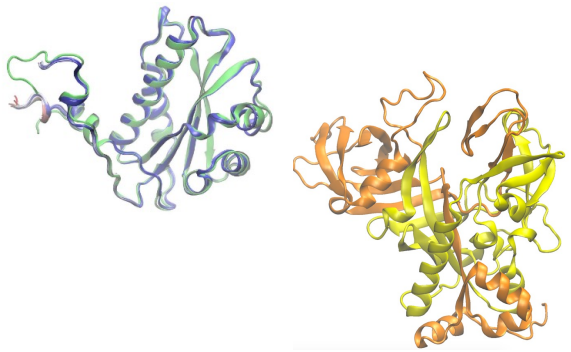
Article

Highly accurate protein structure prediction for the human proteome

https://doi.org/10.1038/s41586-021-03828-1
Received: 11 May 2021
Accepted: 16 July 2021
Published online: 22 July 2021
Open access
Check for updates

Kathryn Tunyasuvunakool[1✉], Jonas Adler[1], Zachary Wu[1], Tim Green[1], Michal Zielinski[1], Augustin Žídek[1], Alex Bridgland[1], Andrew Cowie[1], Clemens Meyer[1], Agata Laydon[1], Sameer Velankar[2], Gerard J. Kleywegt[2], Alex Bateman[2], Richard Evans[1], Alexander Pritzel[1], Michael Figurnov[1], Olaf Ronneberger[1], Russ Bates[1], Simon A. A. Kohl[1], Anna Potapenko[1], Andrew J. Ballard[1], Bernardino Romera-Paredes[1], Stanislav Nikolov[1], Rishub Jain[1], Ellen Clancy[1], David Reiman[1], Stig Petersen[1], Andrew W. Senior[1], Koray Kavukcuoglu[1], Ewan Birney[2], Pushmeet Kohli[1], John Jumper[1,3✉] & Demis Hassabis[1,3✉]





Quantitative Biology > Quantitative Methods

[Submitted on 25 Jan 2022]

Proteome-scale Deployment of Protein Structu Workflows on the Summit Supercomputer

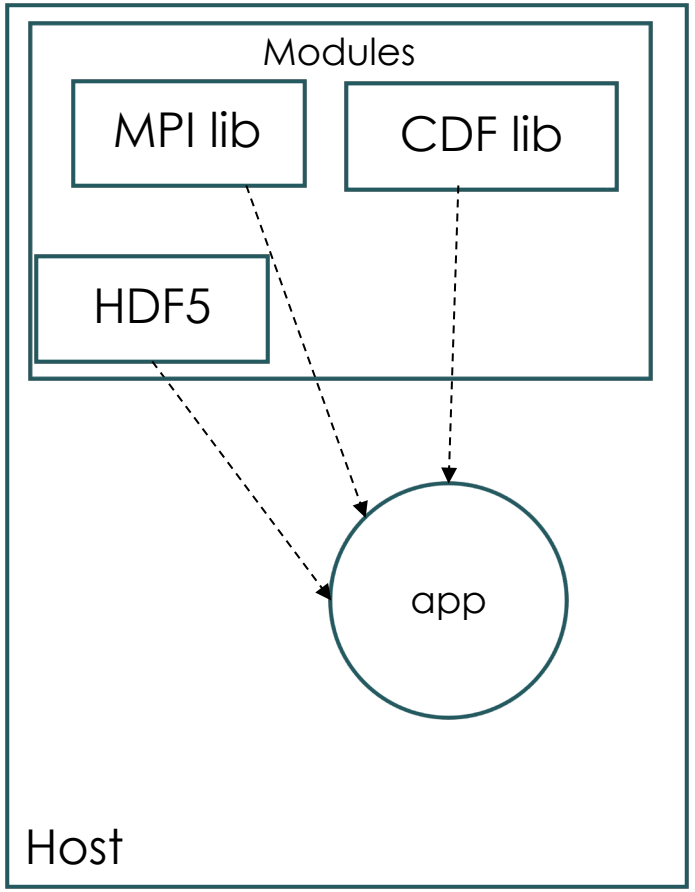
Mu Gao, Mark Coletti, Russell B. Davidson, Ryan Prout, Subil Abraham, Be Sedova

Deep learning has contributed to major advances in the prediction of protein struc fundamental problem in structural bioinformatics. With predictions now approachi crystallographic resolution in some cases, and with accelerators like GPUs and TPU large models rapid, fast genome-level structure prediction becomes an obvious ai computing resources can be used to perform genome-scale protein structure pred art deep learning models, providing a wealth of new data for systems biology applications. Here we describe our efforts to efficiently deploy the AlphaFold2 program, for full-proteome structure prediction, at scale on

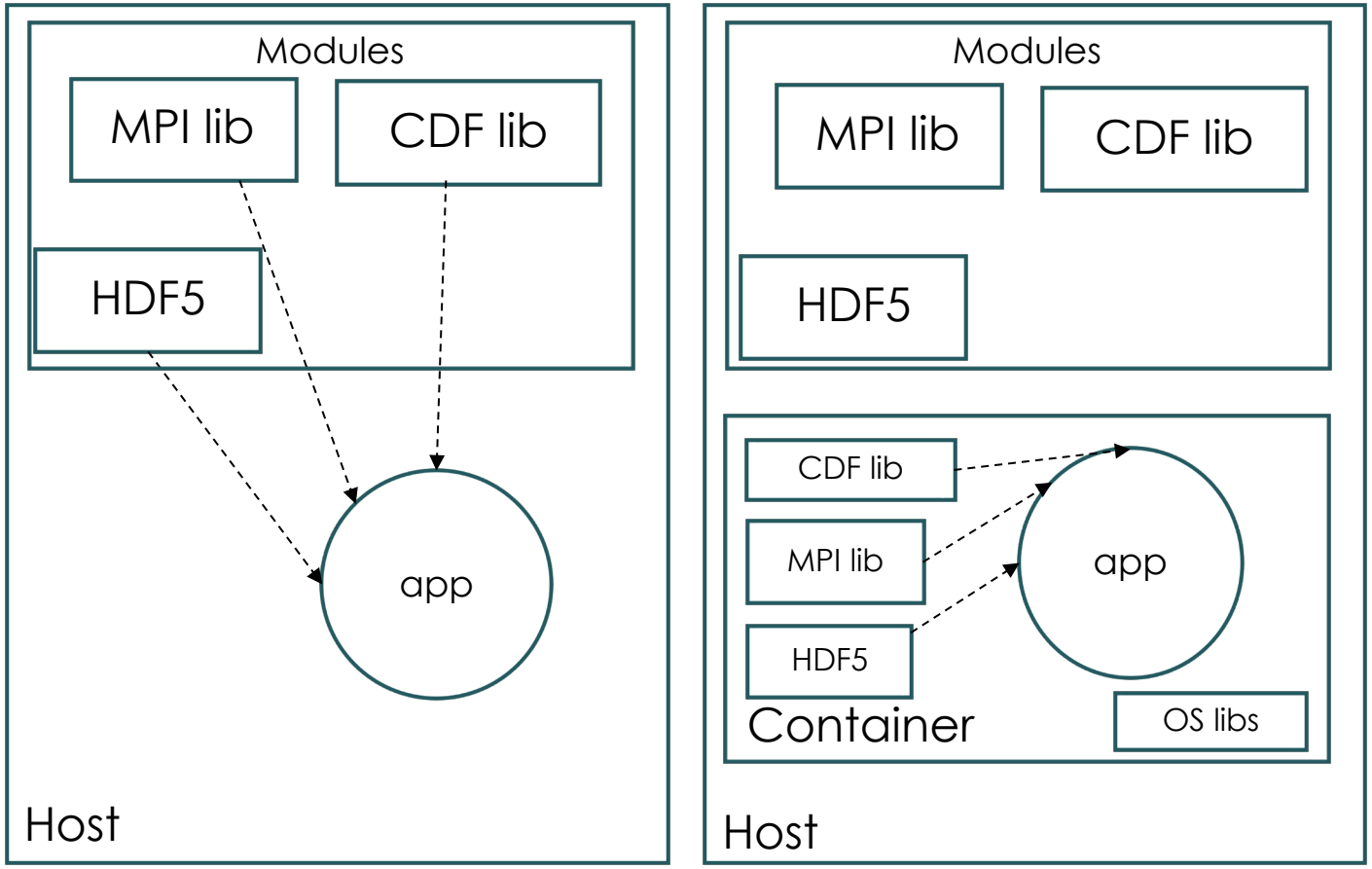
High-Performance Deep Learning Toolbox for Genome-Scale Prediction of Protein Structure and Function

Mu Gao**, Peik Lund-Andersen†, Alex Morehead§, Sajid Mahmud§, Chen Chen§, Xiao Chen§, Nabin Giri§, Raj S. Roy§, Farhan Quadir§, T. Chad Effler‡, Ryan Prout‡, Subil Abraham‡, Wael Elwasif‡, N. Quentin Haas‡, Jeffrey Skolnick*, Jianlin Cheng§*, Ada Sedova‡*
*Georgia Institute of Technology, Atlanta, GA
†University of Idaho, Moscow, ID
‡Oak Ridge National Laboratory, Oak Ridge, TN
§University of Missouri, Columbia, MO
*Corresponding email: mu.gao@gatech.edu, chengji@missouri.edu, sedovaaa@ornl.gov

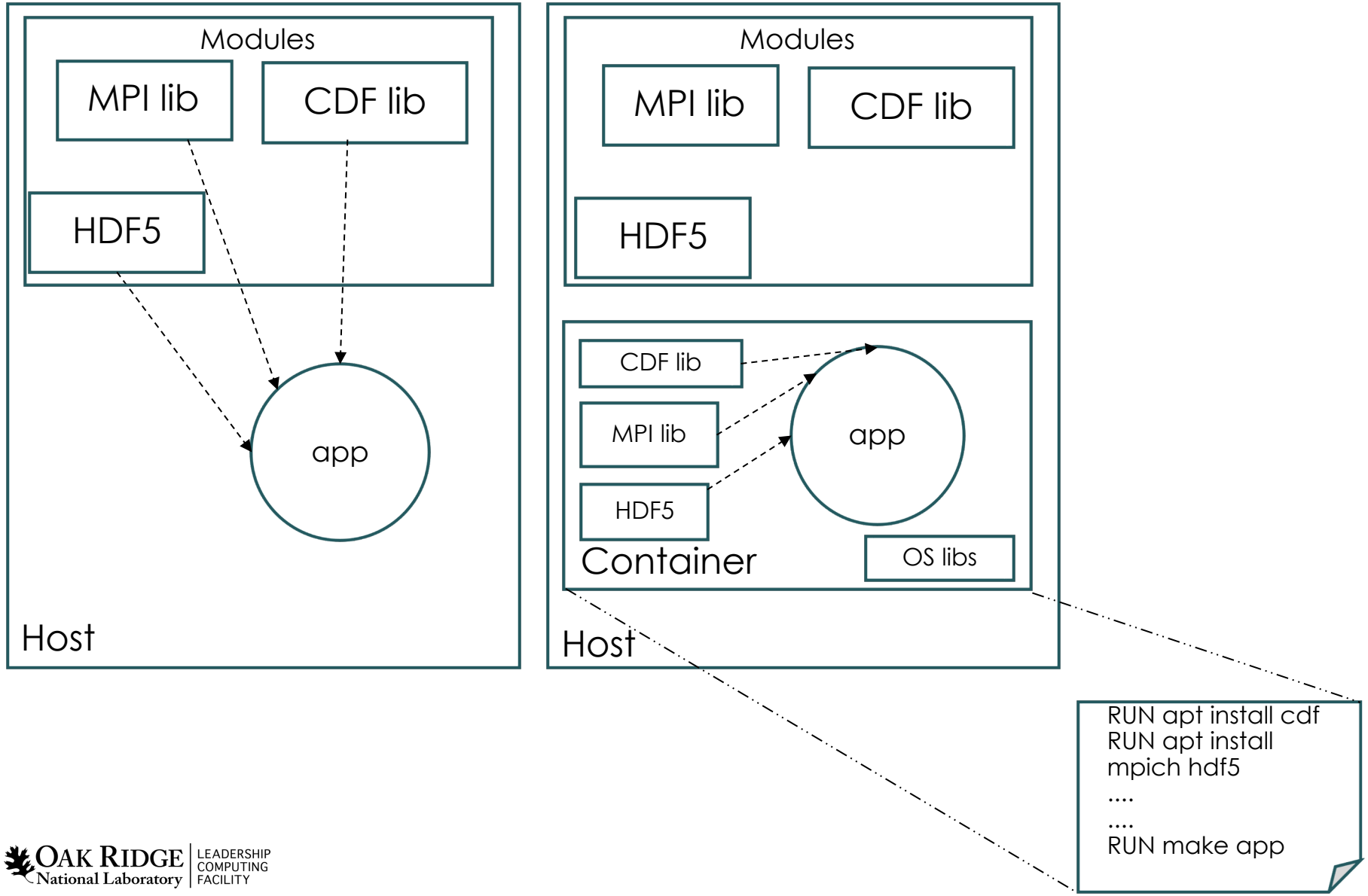
What are Containers?



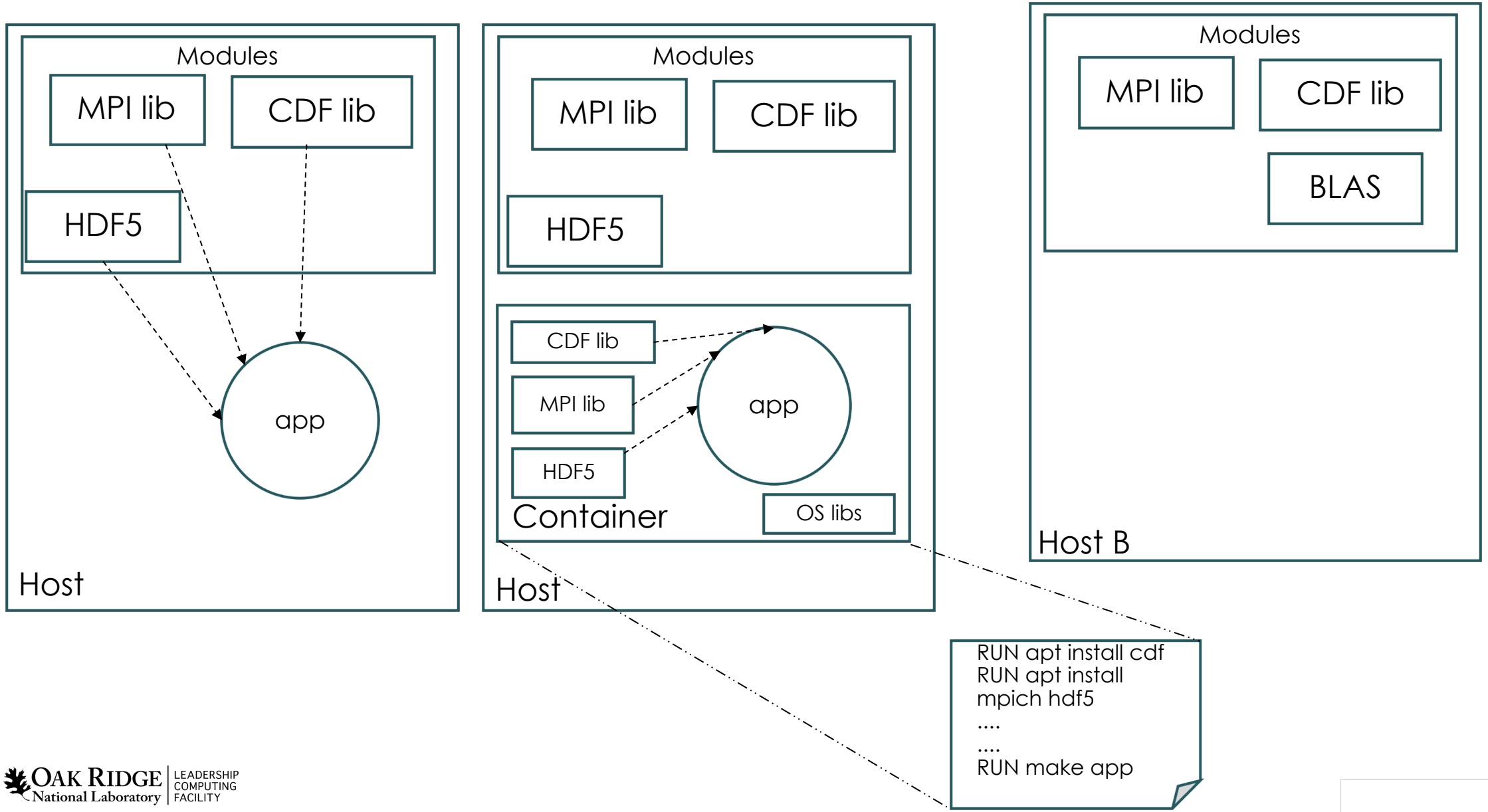
What are Containers?



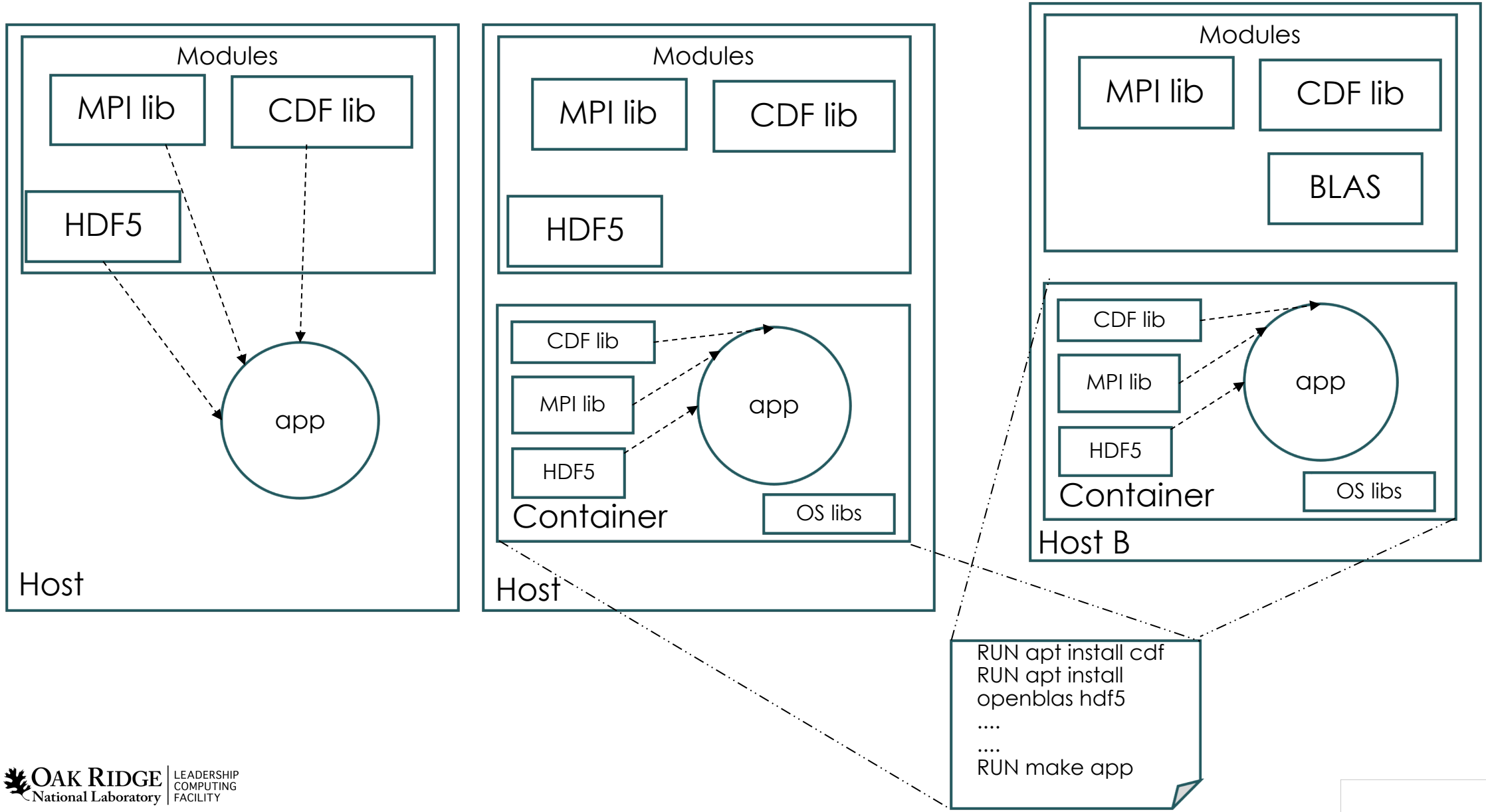
What are Containers?



What are Containers?



What are Containers?



Terminology

- Container image (or) image – The file or set of files that encapsulate your application and libraries.
  - Equivalent to your binary executable you get from compiling code
- Container – The actual running process on the machine that is doing the work
  - Equivalent to the process that is started when you run your binary executable
- Docker/Podman/Singularity – names of container software
  - These software give you the ability to build images and run containers

Summit's Container Infrastructure

- We use a combination of Podman and Singularity
  - Podman to build the image
  - Singularity to run it
- We provide a Summit base container image
  - Best practice to start with this base image for your builds
- Containers are built and run with **no root privileges**
  - If you encounter any permission related issues during your build, email help@olcf.ornl.gov

Prerequisites to building

- The storage.conf file
  - Necessary for Podman to build images correctly
- Specifies a location in the NVMe to store Podman image layers
- Podman will throw errors without it

~/.config/containers/storage.conf

```
[storage]
driver = "overlay"
graphroot = "/tmp/containers/subil"

[storage.options]
additionalimagestores = [
]

[storage.options.overlay]
ignore_chown_errors = "true"
mount_program = "/usr/bin/fuse-
overlayfs"
mountopt = "nodev,metacopy=on"

[storage.options.thinpool]
```

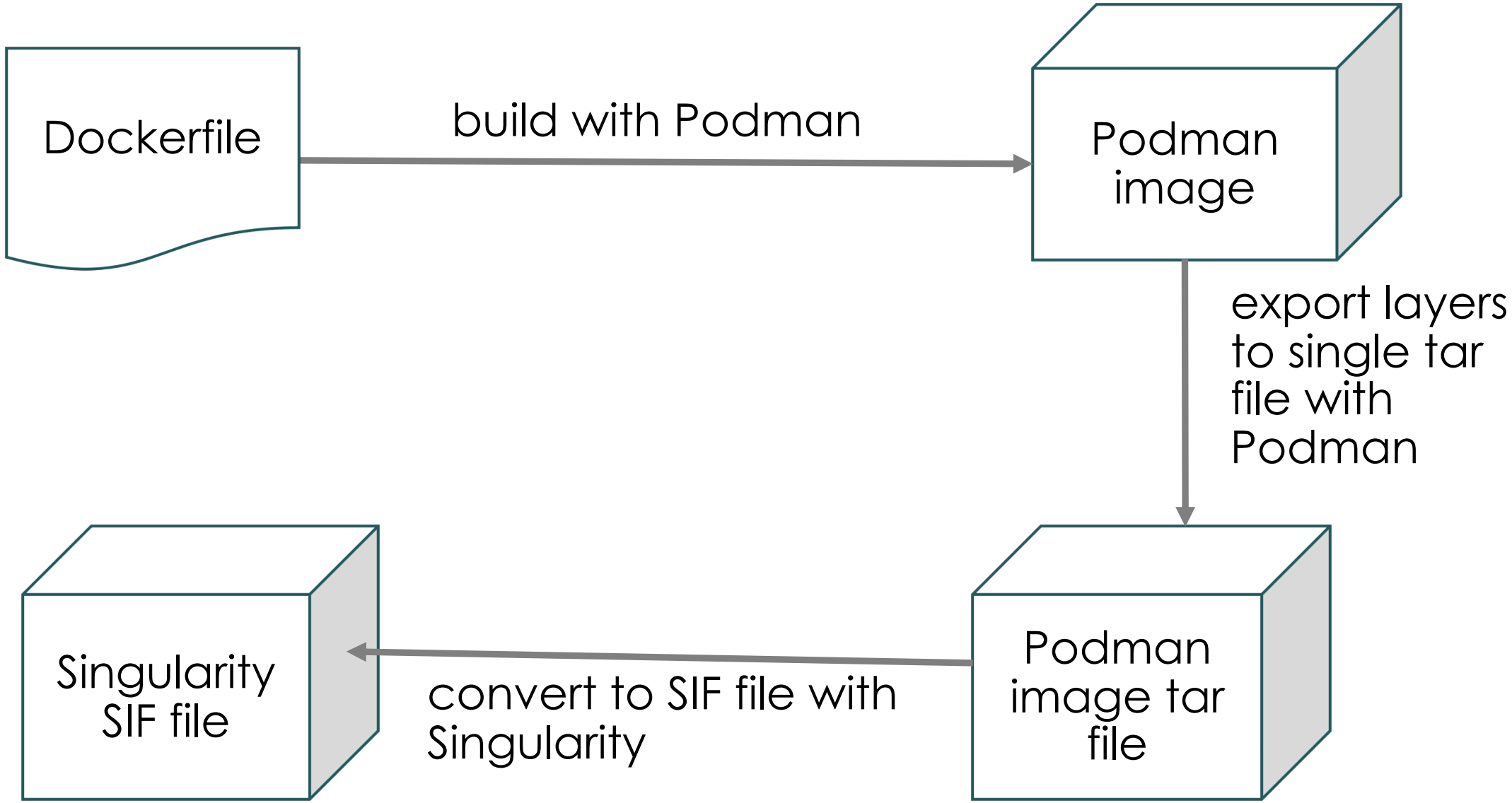
Prerequisites to building

- The storage.conf file
  - Necessary for Podman to build images correctly
- Specifies a location in the NVMe to store Podman image layers
- Podman will throw errors without it

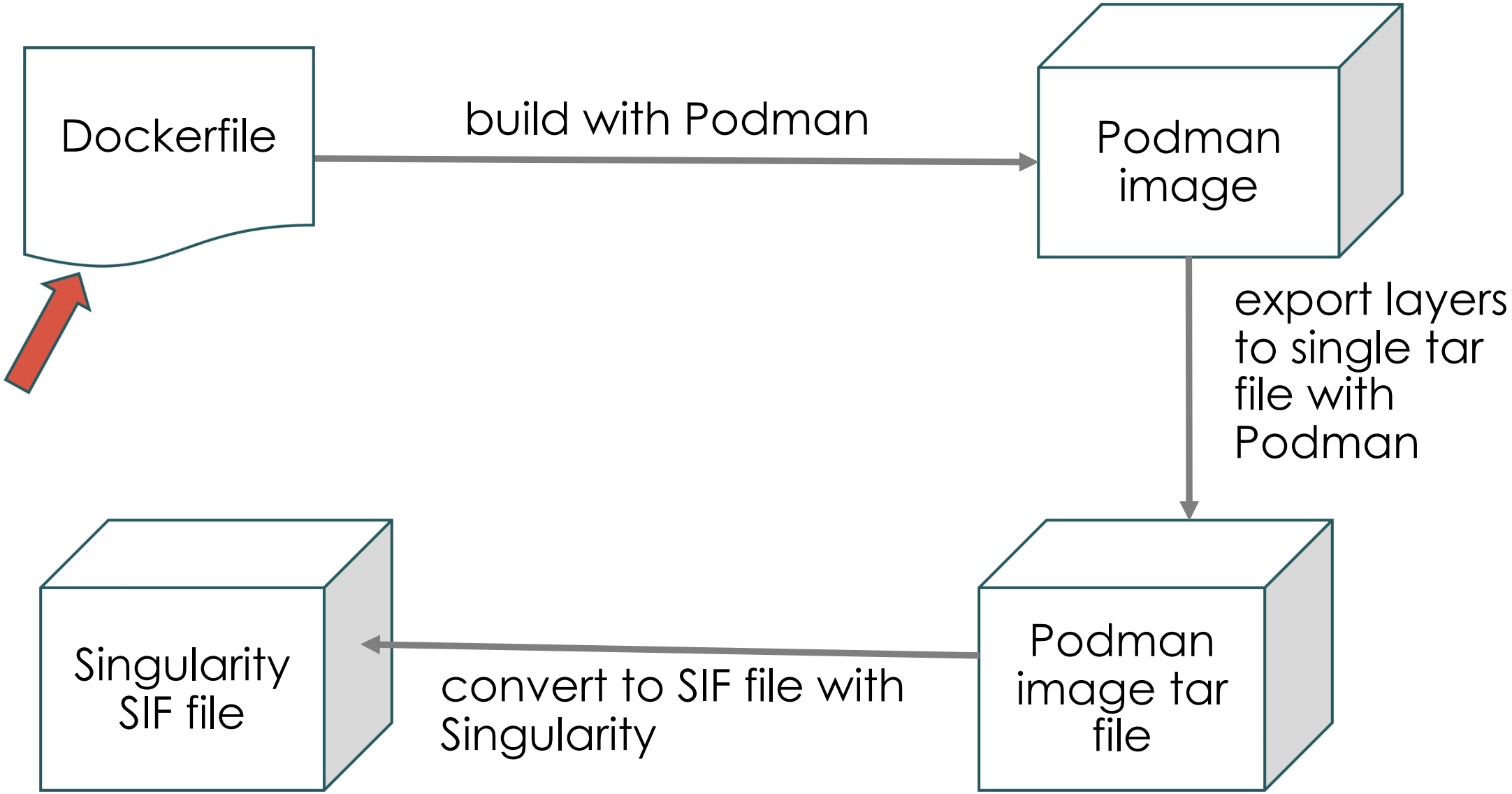
~/.config/containers/storage.conf

```
[storage]
driver = "overlay"
graphroot = "/tmp/containers/subil"

[storage.options]
additionalimagestores = [
]

[storage.options.overlay]
ignore_chown_errors = "true"
mount_program = "/usr/bin/fuse-
overlayfs"
mountopt = "nodev,metacopy=on"

[storage.options.thinpool]
```

Steps to build your container image



Steps to build your container image



Step 1: Writing your Dockerfile

- Defines your container image that you want to build.
- Specifies a base image, and a list of commands that execute to install packages, compile code, etc

```
FROM quay.io/centos/centos:stream8
RUN dnf -y install epel-release && \
      dnf -y install fakeroot
RUN fakeroot dnf upgrade -y && \
      fakeroot dnf update -y
RUN fakeroot dnf install -y wget hostname
ENTRYPOINT ["/bin/bash"]
```

Writing your Dockerfile

```
FROM quay.io/centos/centos:stream8
RUN dnf -y install epel-release && \
      dnf -y install fakeroot
RUN fakeroot dnf upgrade -y && \
      fakeroot dnf update -y
RUN fakeroot dnf install -y wget hostname
ENTRYPOINT ["/bin/bash"]
```

Writing your Dockerfile

Best suited when starting from scratch

```
FROM quay.io/centos/centos:stream8
RUN dnf -y install epel-release && \
      dnf -y install fakeroot
RUN fakeroot dnf upgrade -y && \
      fakeroot dnf update -y
RUN fakeroot dnf install -y wget hostname
ENTRYPOINT ["/bin/bash"]
```

Writing your Dockerfile

```
FROM quay.io/centos/centos:stream8
RUN dnf -y install epel-release && \
      dnf -y install fakeroot
RUN fakeroot dnf upgrade -y && \
      fakeroot dnf update -y
RUN fakeroot dnf install -y wget hostname
ENTRYPOINT ["/bin/bash"]
```

Writing your Dockerfile

```
FROM quay.io/centos/centos:stream8
RUN dnf -y install epel-release && \
      dnf -y install fakeroot
RUN fakeroot dnf upgrade -y && \
       fakeroot dnf update -y
RUN fakeroot dnf install -y wget hostname
ENTRYPOINT ["/bin/bash"]
```

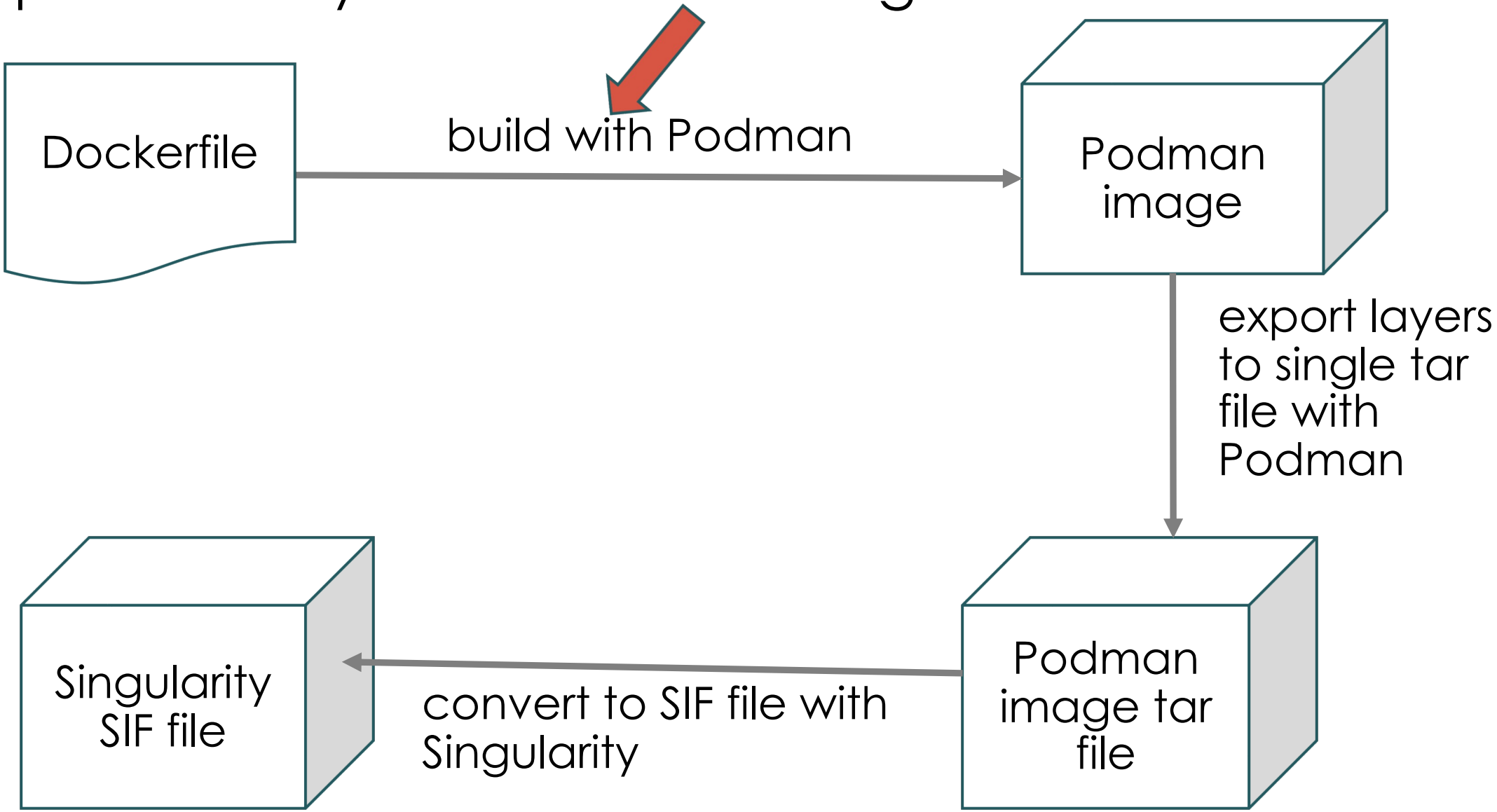
Writing your Dockerfile (if you're using MPI and/or GPUs)

```
FROM code.ornl.gov:4567/olcfcontainers/olcfbaseimages/mpiimage-
centos-cuda:latest
RUN mkdir /app
COPY mpiexample.c /app
RUN cd /app && mpicc -o mpiexample mpiexample.c
```

Writing your Dockerfile (if you're using MPI and/or GPUs)

```
FROM code.ornl.gov:4567/olcfcontainers/olcfbaseimages/mpiimage-centos-cuda:latest
RUN mkdir /app
COPY mpiexample.c /app
RUN cd /app && mpicc -o mpiexample mpiexample.c
```

Steps to build your container image



Building your container image with Podman

```
$ module load gcc/9.1.0

$ podman build -v $MPI_ROOT:$MPI_ROOT -f \
    mpiexample.dockerfile -t mpiexample:latest .
```

Building your container image with Podman

required for MPI support

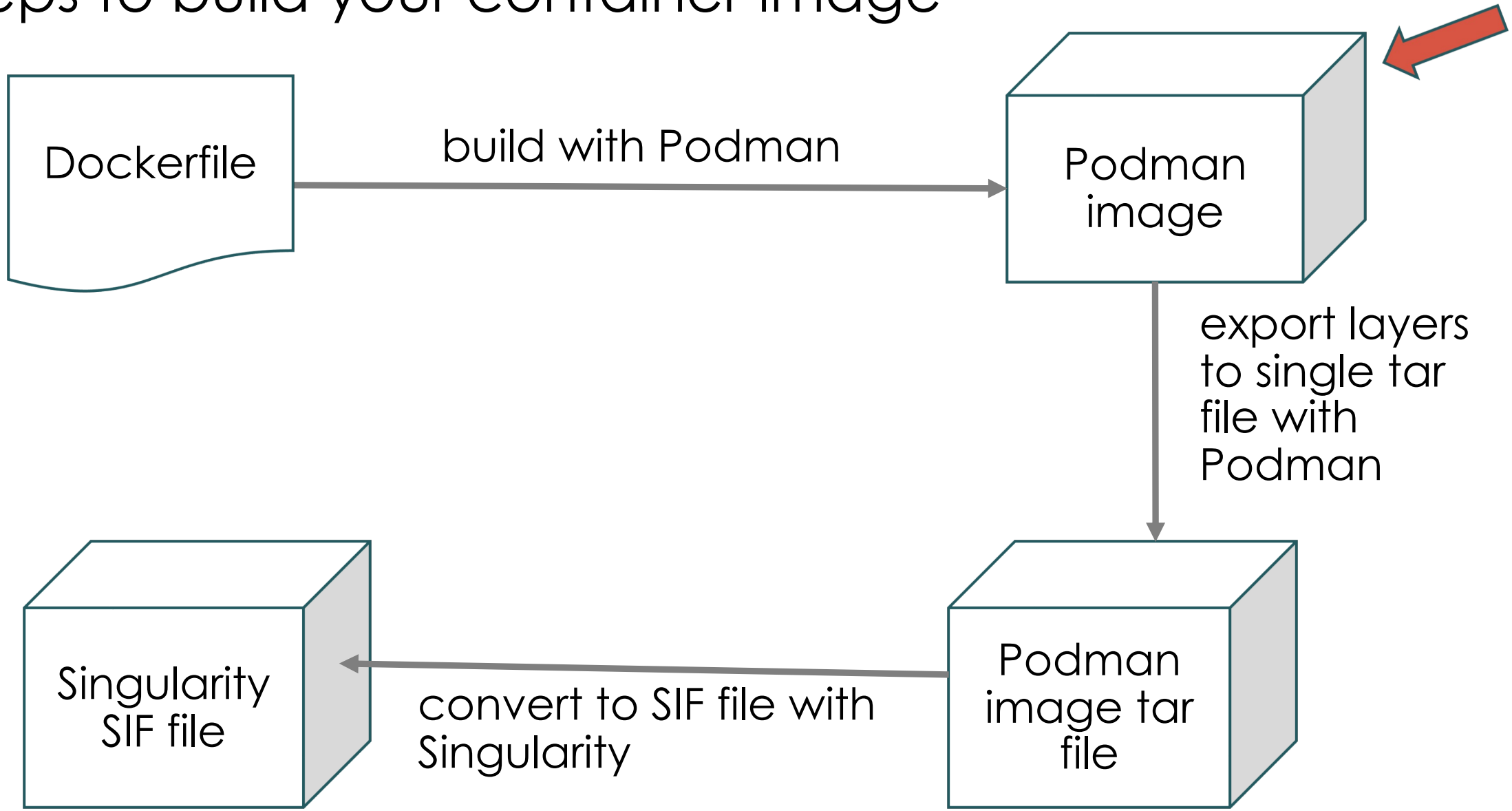
```
$ module load gcc/9.1.0

$ podman build -v $MPI_ROOT:$MPI_ROOT -f \
    mpiexample.dockerfile -t mpiexample:latest .
```

Building your container image with Podman

Mounts the MPI root directory into the container during the build process

```
$ module load gcc/9.1.0

$ podman build -v $MPI_ROOT:$MPI_ROOT -f \
    mpiexample.dockerfile -t mpiexample:latest .
```

Steps to build your container image



Podman images

- Check that the Podman image has been created

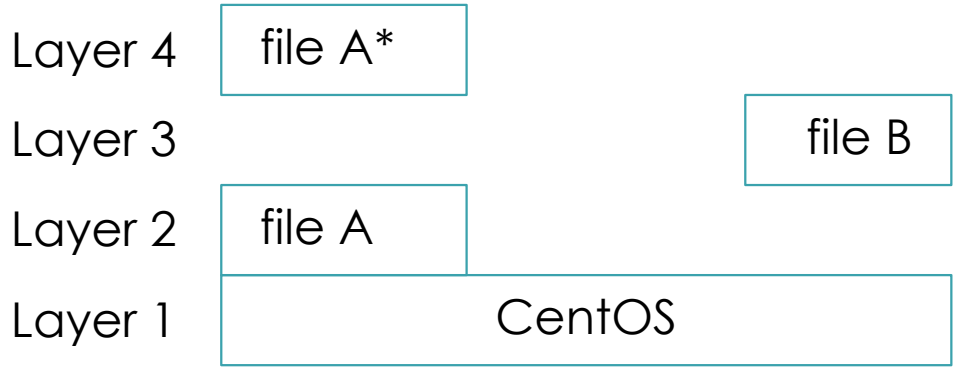
```
$ podman image ls
REPOSITORY                                TAG     IMAGE ID      CREATED        SIZE
localhost/mpiexample                      latest  d00b64738858  3 hours ago    14.3 GB
code.ornl.gov:4567/olcfcontainers...      latest  24fd3853c2f8  9 months ago   14.3 GB
```

Podman images

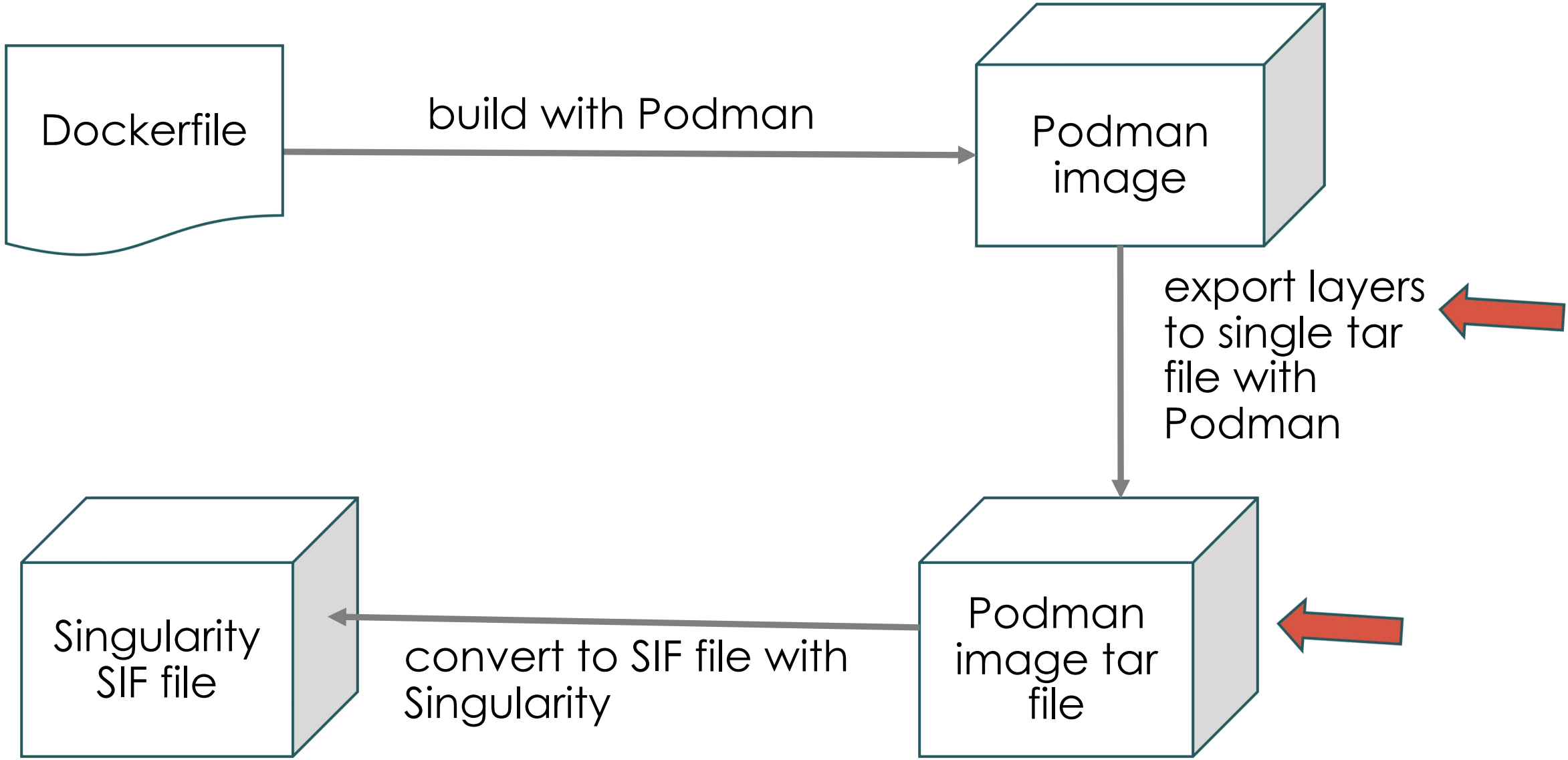
- Check that the Podman image has been created

```
$ podman image ls
REPOSITORY                                TAG     IMAGE ID      CREATED        SIZE
localhost/mpiexample                      latest  d00b64738858  3 hours ago    14.3 GB
code.ornl.gov:4567/olcfcontainers...      latest  24fd3853c2f8  9 months ago   14.3 GB
```

- Podman images are layered
- Stored in /tmp/containers
- Not accessible from compute nodes
- Layers not supported on NFS or GPFS



Steps to build your container image



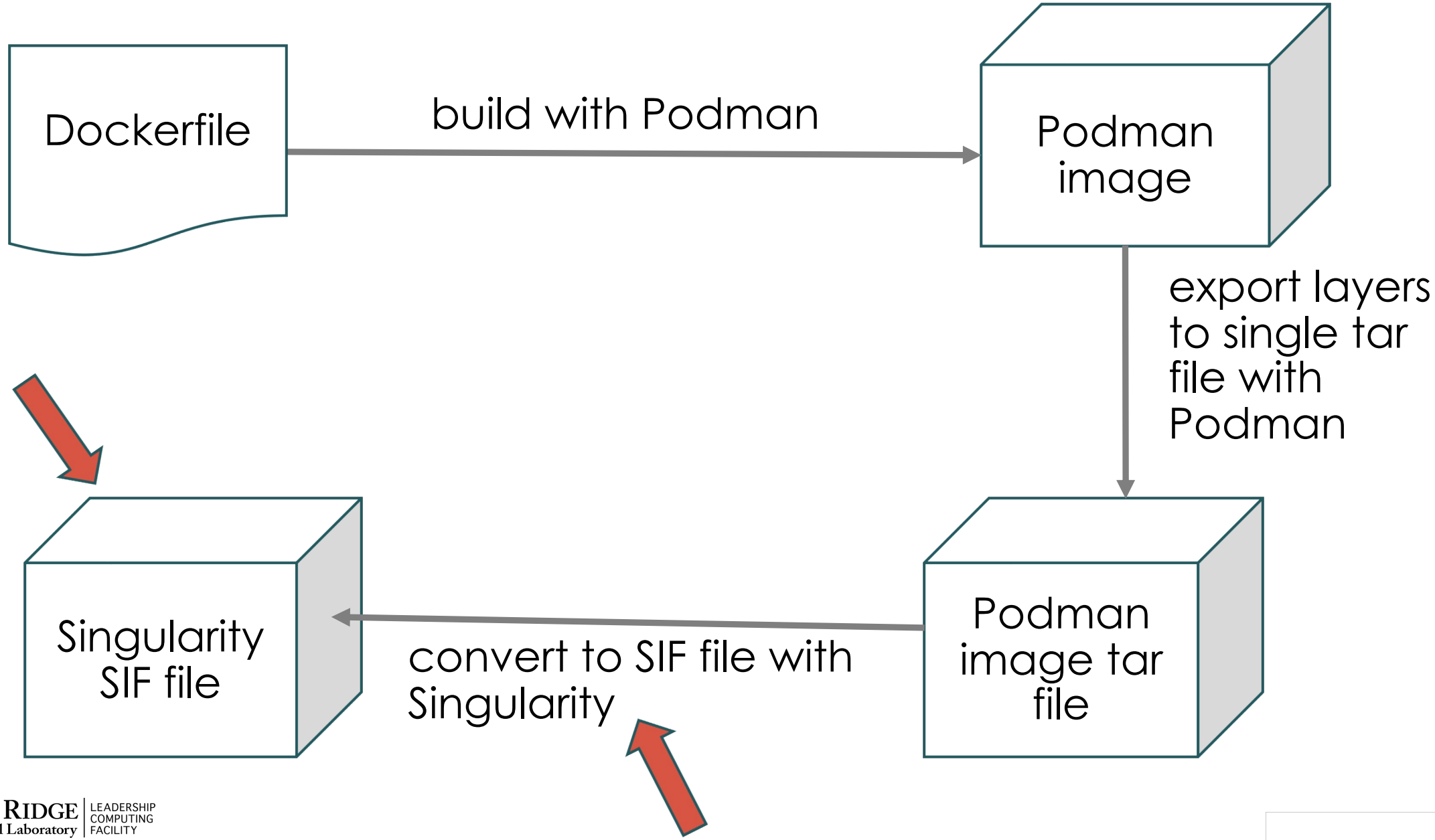
Convert your Podman image to a tar file

```
$ podman save -o mpiexampleimage.tar localhost/mpiexample:latest

$ ls

mpiexample.c
mpiexample.dockerfile
mpiexampleimage.tar
```

Convert your Podman image to a tar file

```
$ podman save -o mpiexampleimage.tar localhost/mpiexample:latest

$ ls

mpiexample.c
mpiexample.dockerfile
mpiexampleimage.tar
```

Steps to build your container image



Convert tar file to Singularity sif file

```
$ singularity build --disable-cache mpiexampleimage.sif \
    docker-archive://mpiexampleimage.tar

$ ls
mpiexample.c
mpiexample.dockerfile
mpiexampleimage.sif
mpiexampleimage.tar
```

Convert tar file to Singularity sif file

- If you get a message saying ‘Killed’, you probably hit the cgroup memory limit
- To avoid this, start an interactive job and run:

```
$ jsrun –n1 –c42 –brs singularity build ––disable–cache \
      /gpfs/path/mpiexampleimage.sif \
      docker–archive://mpiexampleimage.tar

$ ls /gpfs/path
mpiexampleimage.sif
```

Convert tar file to Singularity sif file

- If you get a message saying ‘Killed’, you probably hit the cgroup memory limit
- To avoid this, start an interactive job and run:

```
$ jsrun -n1 -c42 -brs singularity build --disable-cache \
      /gpfs/path/mpiexampleimage.sif \
      docker-archive://mpiexampleimage.tar

$ ls /gpfs/path
mpiexampleimage.sif
```

Running your containers (With MPI + GPUs)

```
#BSUB -P STF007
#BSUB -W 0:30
#BSUB -nnodes 2
#BSUB -J singularity
#BSUB -o singularity.%J
#BSUB -e singularity.%J

module purge
module load DefApps
module load gcc/9.1.0

source /gpfs/alpine/stf007/world-shared/containers/utils/requiredmpilibs.source

jsrun -n 8 -r4 -g1 singularity exec \
    --nv \
    --bind /gpfs/alpine/stf007/proj-shared/data:/data,$MPI_ROOT:$MPI_ROOT \
    /path/to/mpiexampleimage.sif \
    /app/mpiexample
```

Running your containers (With MPI + GPUs)

```
#BSUB -P STF007
#BSUB -W 0:30
#BSUB -nnodes 2
#BSUB -J singularity
#BSUB -o singularity.%J
#BSUB -e singularity.%J

module purge
module load DefApps
module load gcc/9.1.0

source /gpfs/alpine/stf007/world-shared/containers/utils/requiredmpilibs.source

jsrun -n 8 -r4 -g1 singularity exec \
    --nv \
    --bind /gpfs/alpine/stf007/proj-shared/data:/data,$MPI_ROOT:$MPI_ROOT \
    /path/to/mpiexampleimage.sif
    /app/mpiexample
```

Running your containers (With MPI + GPUs)

```
#BSUB -P STF007
#BSUB -W 0:30
#BSUB -nnodes 2
#BSUB -J singularity
#BSUB -o singularity.%J
#BSUB -e singularity.%J

module purge
module load DefApps
module load gcc/9.1.0

source /gpfs/alpine/stf007/world-shared/containers/utils/requiredmpilibs.source

jsrun -n 8 -r4 -g1 singularity exec \
    --nv \
    --bind /gpfs/alpine/stf007/proj-shared/data:/data,$MPI_ROOT:$MPI_ROOT \
    /path/to/mpiexampleimage.sif
    /app/mpiexample
```

Running your containers (With MPI + GPUs)

```
#BSUB -P STF007
#BSUB -W 0:30
#BSUB -nnodes 2
#BSUB -J singularity
#BSUB -o singularity.%J
#BSUB -e singularity.%J

module purge
module load DefApps
module load gcc/9.1.0

source /gpfs/alpine/stf007/world-shared/containers/utils/requiredmpilibs.source

jsrun -n 8 -r4 -g1 singularity exec \
    --nv \
    --bind /gpfs/alpine/stf007/proj-shared/data:/data,$MPI_ROOT:$MPI_ROOT \
    /path/to/mpiexampleimage.sif
    /app/mpiexample
```

Running your containers (With MPI + GPUs)

```
#BSUB -P STF007
#BSUB -W 0:30
#BSUB -nnodes 2
#BSUB -J singularity
#BSUB -o singularity.%J
#BSUB -e singularity.%J

module purge
module load DefApps
module load gcc/9.1.0

source /gpfs/alpine/stf007/world-shared/containers/utils/requiredmpilibs.source

jsrun -n 8 -r4 -g1 singularity exec \
    --nv \
    --bind /gpfs/alpine/stf007/proj-shared/data:/data,$MPI_ROOT:$MPI_ROOT \
    /path/to/mpiexampleimage.sif
    /app/mpiexample
```

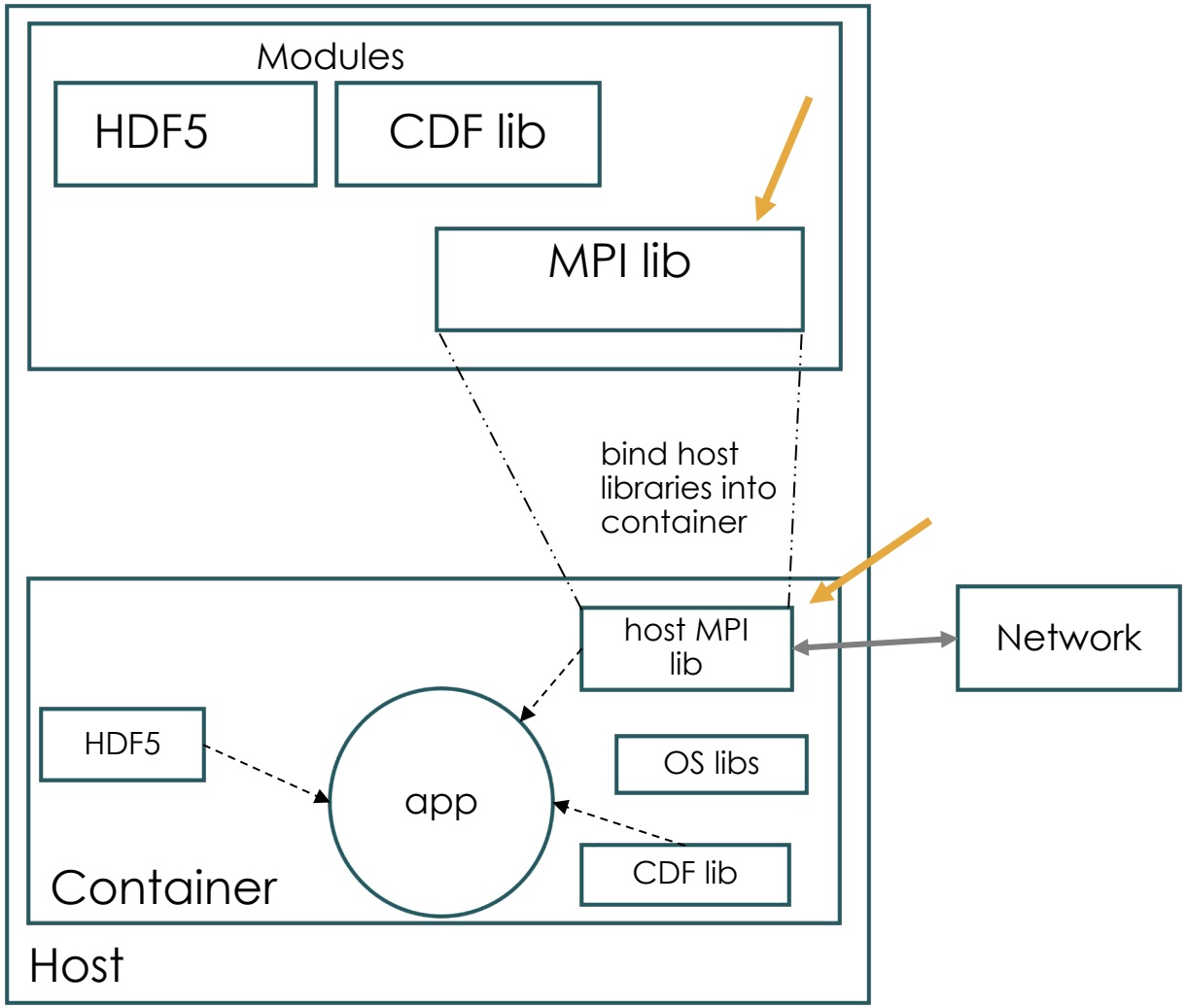
Running your containers (With MPI + GPUs)

```
#BSUB -P STF007
#BSUB -W 0:30
#BSUB -nnodes 2
#BSUB -J singularity
#BSUB -o singularity.%J
#BSUB -e singularity.%J

module purge
module load DefApps
module load gcc/9.1.0

source /gpfs/alpine/stf007/world-shared/containers/utils/requiredmpilibs.source

jsrun -n 8 -r4 -g1 singularity exec \
    --nv \
    --bind /gpfs/alpine/stf007/proj-shared/data:/data,$MPI_ROOT:$MPI_ROOT \
    /path/to/mpiexampleimage.sif
    /app/mpiexample
```

Running your containers (With MPI + GPUs)

```
#BSUB -P STF007
#BSUB -W 0:30
#BSUB -nnodes 2
#BSUB -J singularity
#BSUB -o singularity.%J
#BSUB -e singularity.%J

module purge
module load DefApps
module load gcc/9.1.0

source /gpfs/alpine/stf007/world-shared/containers/utils/requiredmpilibs.source

jsrun -n 8 -r4 -g1 singularity exec \
    --nv \
    --bind /gpfs/alpine/stf007/proj-shared/data:/data,$MPI_ROOT:$MPI_ROOT \
    /path/to/mpiexampleimage.sif
    /app/mpiexample
```

Interlude: Binding MPI libraries



Running your containers (With MPI + GPUs)

```
#BSUB -P STF007
#BSUB -W 0:30
#BSUB -nnodes 2
#BSUB -J singularity
#BSUB -o singularity.%J
#BSUB -e singularity.%J

module purge
module load DefApps
module load gcc/9.1.0

source /gpfs/alpine/stf007/world-shared/containers/utils/requiredmpilibs.source

jsrun -n 8 -r4 -g1 singularity exec \
    --nv \
    --bind /gpfs/alpine/stf007/proj-shared/data:/data,$MPI_ROOT:$MPI_ROOT \
    /path/to/mpiexampleimage.sif
    /app/mpiexample
```

Running your containers (With MPI + GPUs)

```
#BSUB -P STF007
#BSUB -W 0:30
#BSUB -nnodes 2
#BSUB -J singularity
#BSUB -o singularity.%J
#BSUB -e singularity.%J

module purge
module load DefApps
module load gcc/9.1.0

source /gpfs/alpine/stf007/world-shared/containers/utils/requiredmpilibs.source

jsrun -n 8 -r4 -g1 singularity exec \
    --nv \
    --bind /gpfs/alpine/stf007/proj-shared/data:/data,$MPI_ROOT:$MPI_ROOT \
    /path/to/mpiexampleimage.sif
    /app/mpiexample
```

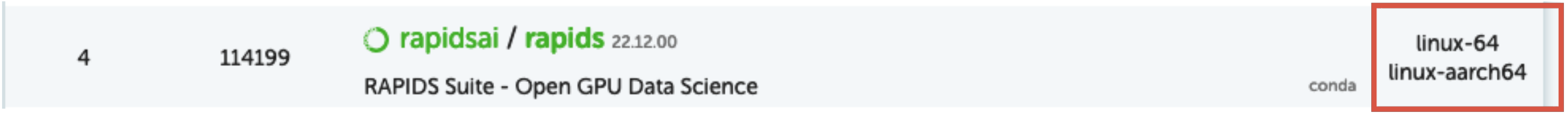
Gotchas: PowerPC Limitations

- You will still face limitations because of Summits PowerPC architecture
- You can't just take any old base image from a registry
  - many images don't support PowerPC.
- A package install that works on your x86 machine may not work on your Summit container
  - Many conda packages only have x86 arch support
  - You may have to build something you need from source within the container with the necessary libraries

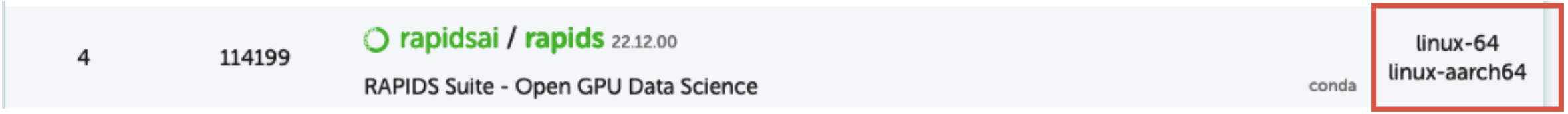
Example: Trying to install RAPIDS

4	114199	rapidsai / rapids 22.12.00 RAPIDS Suite - Open GPU Data Science	conda	linux-64 linux-aarch64

Example: Trying to install RAPIDS

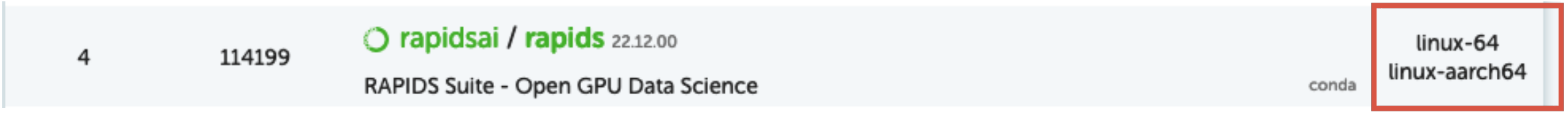


Example: Trying to install RAPIDS




```
STEP 5: RUN source /conda/etc/profile.d/conda.sh && conda activate && conda install -y -c rapidsai
rapids
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retrying with flexible solve.

PackagesNotFoundError: The following packages are not available from current channels:

- rapids

<truncated text>

Error: error building at STEP "RUN source /conda/etc/profile.d/conda.sh && conda activate && conda
install -y -c rapidsai rapids": error while running runtime: exit status 1
```

Example: Trying to install RAPIDS




```
STEP 5: RUN source /conda/etc/profile.d/conda.sh && conda activate && conda install -y -c rapidsai
rapids
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retrying with flexible solve.

PackagesNotFoundError: The following packages are not available from current channels:

- rapids

<truncated text>

Error: error building at STEP "RUN source /conda/etc/profile.d/conda.sh && conda activate && conda
install -y -c rapidsai rapids": error while running runtime: exit status 1
```

Gotchas: Building Your Image

- Your Podman image layers are stored on the login node NVMe
  - This NVMe location isn't shared between login nodes
  - Can get purged on reboots or when filled up, no storage guarantees
- If build in progress – log back in to the same login node to reuse Podman layers
- Once build is complete – immediately convert it to the singularity sif file so you can save it in on /ccs or /gpfs

When are containers on Summit not useful

- You already have a well understood workflow for building and running on Summit
  - Don't fix what isn't broken!
- You need to run software that doesn't support PowerPC
  - Containers won't magically fix that problem
- You need to run the same application in different centers with no changes
  - Can help somewhat, but you might find yourself doing center specific changes anyway
  - e.g. A Summit container image won't run as is on Perlmutter

In Summary

- Containers are useful for packaging up applications and their dependencies
- Useful when building natively is difficult or application requires using containers
- We use
  - Dockerfiles to define our container images
  - Podman to build container images
  - Singularity to run the containers
- Detailed documentation on docs.olcf.ornl.gov -> Software -> Containers on Summit
- Write to help@olcf.ornl.gov for additional assistance