Containerization of applications and services in DT-GEO

Jorge Gomes and Mario David

LIP

This project has received funding from the European Union's Horizon research and innovation programme under the grant agreement No 101058129

Containerisation

What is it

- Lightweight operating system level virtualization method
  - Relying on isolation instead of virtualization or emulation
  - Isolation of processes from the host operating system with very low overhead
  - Execution across different software environments
- Enables self contained encapsulation of a given application or service
  - Including configurations
  - Including software dependencies e.g. libraries and executables
- Limitations
  - Hardware architecture must be the same
  - Operating system kernel must have the same binary interface

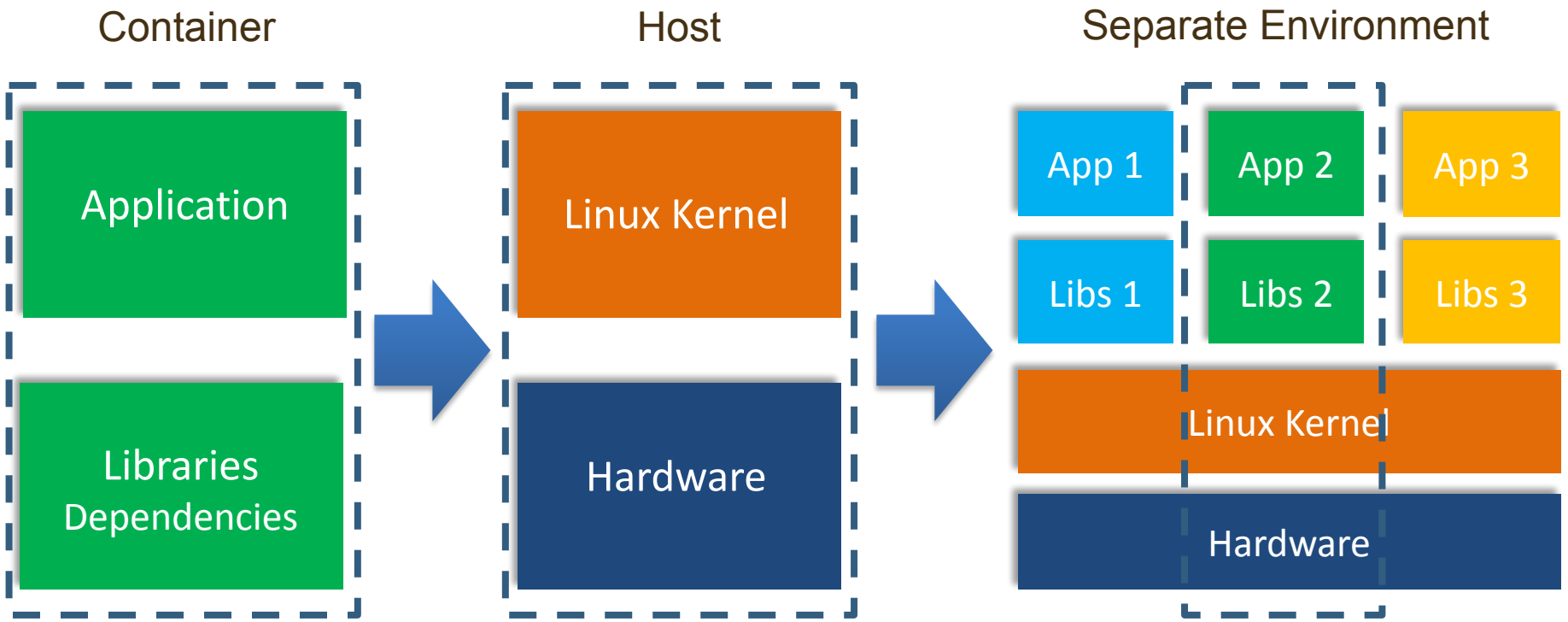
Containerisation

Other advantages

- High efficiency
  - One single operating system kernel shared by many applications
  - Avoids duplication of system processes
  - Performance and resource consumption similar to direct execution in the host
  - Can take advantage of newer more optimized libraries and compilers
- Better maintainability
  - Easier application maintenance, distribution and deployment
  - Instead of adapting the user sw to the host, it brings the user environment to the host
- Easier reproducibility and preservation
  - Having whole application or service plus its run-time environment in an image
  - Container images can be easily stored for later replay, reuse and preservation

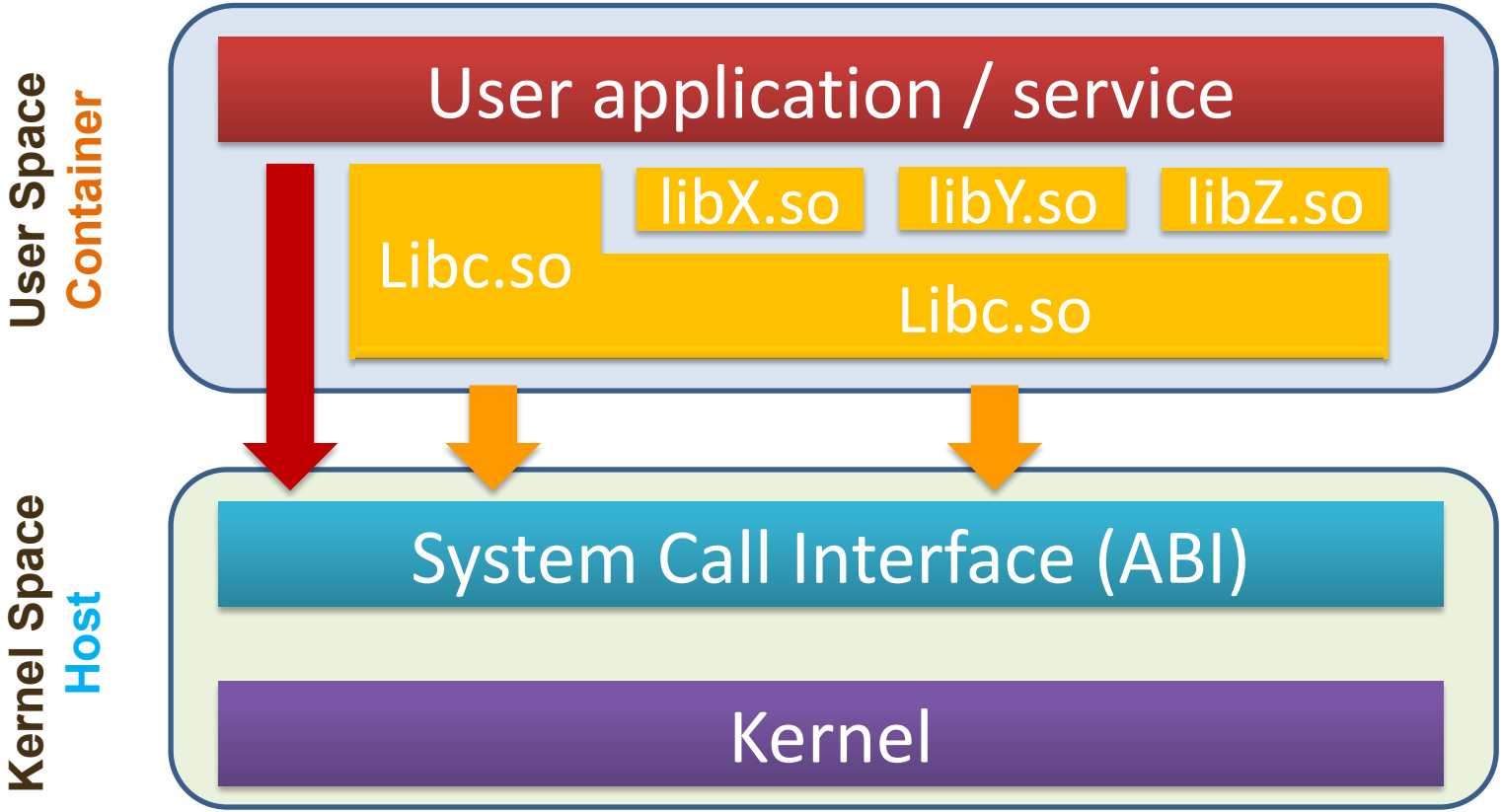
Linux containers

Bringing the user software environment to the execution host



Linux containers

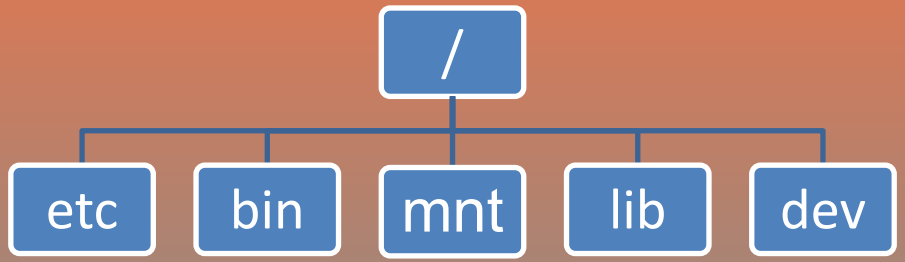
Execution across different distributions



Linux containers

Isolation

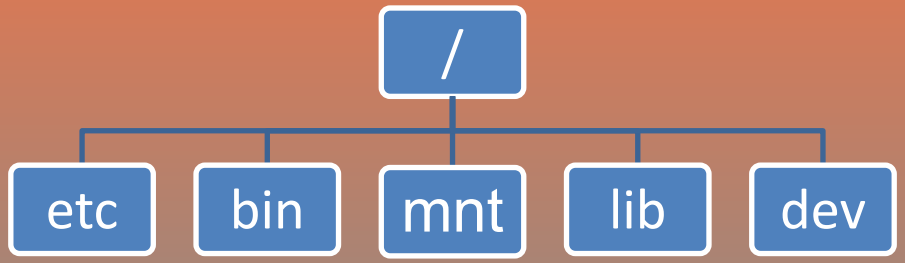
Host

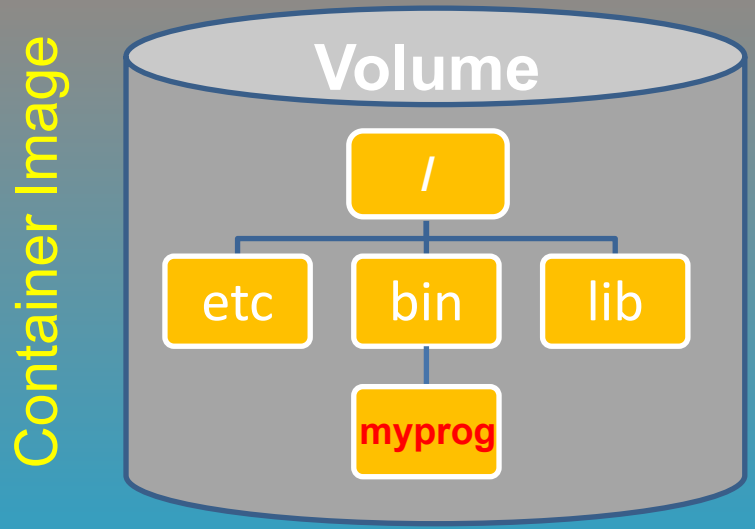


Linux containers

Isolation

Host

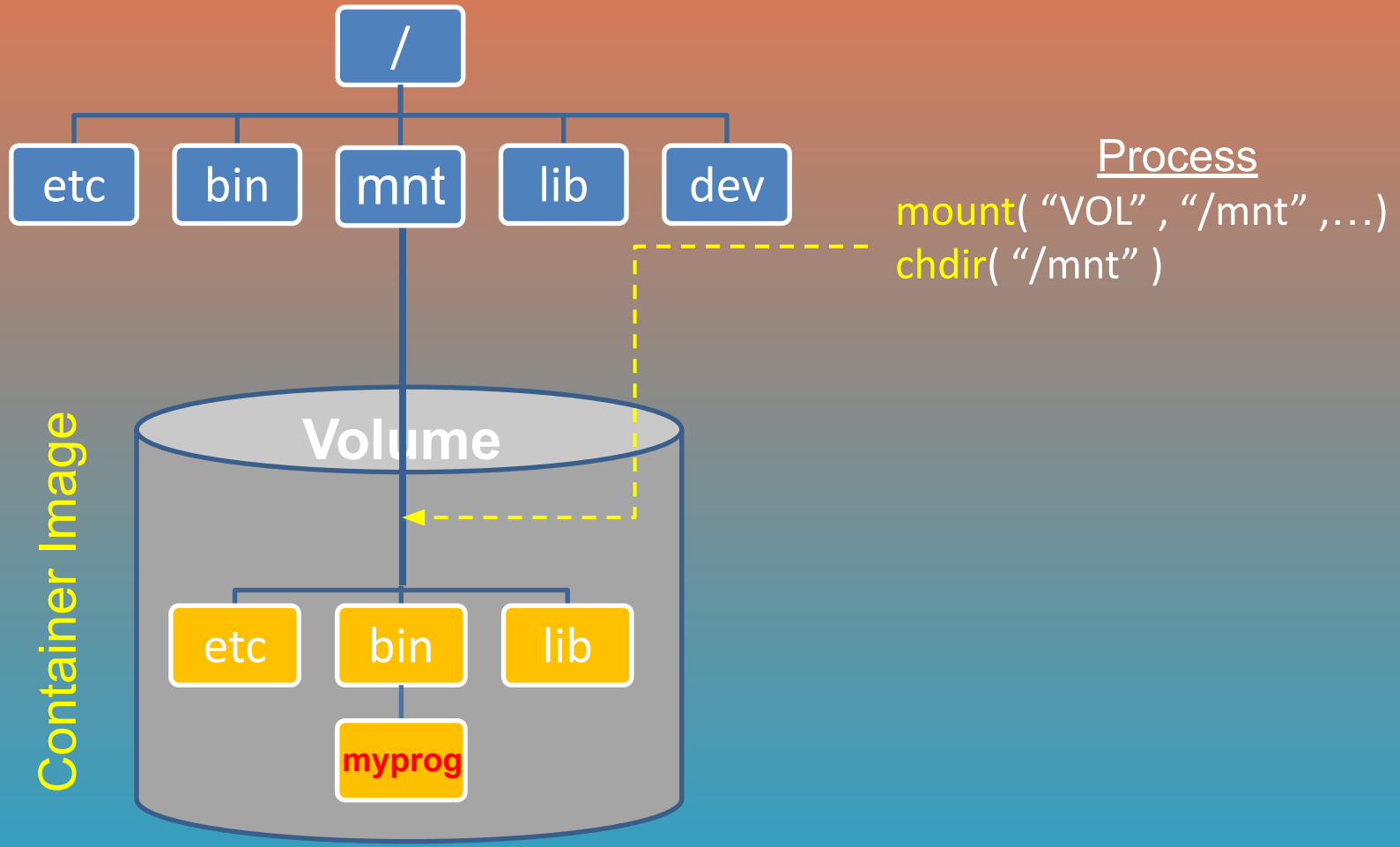




Linux containers

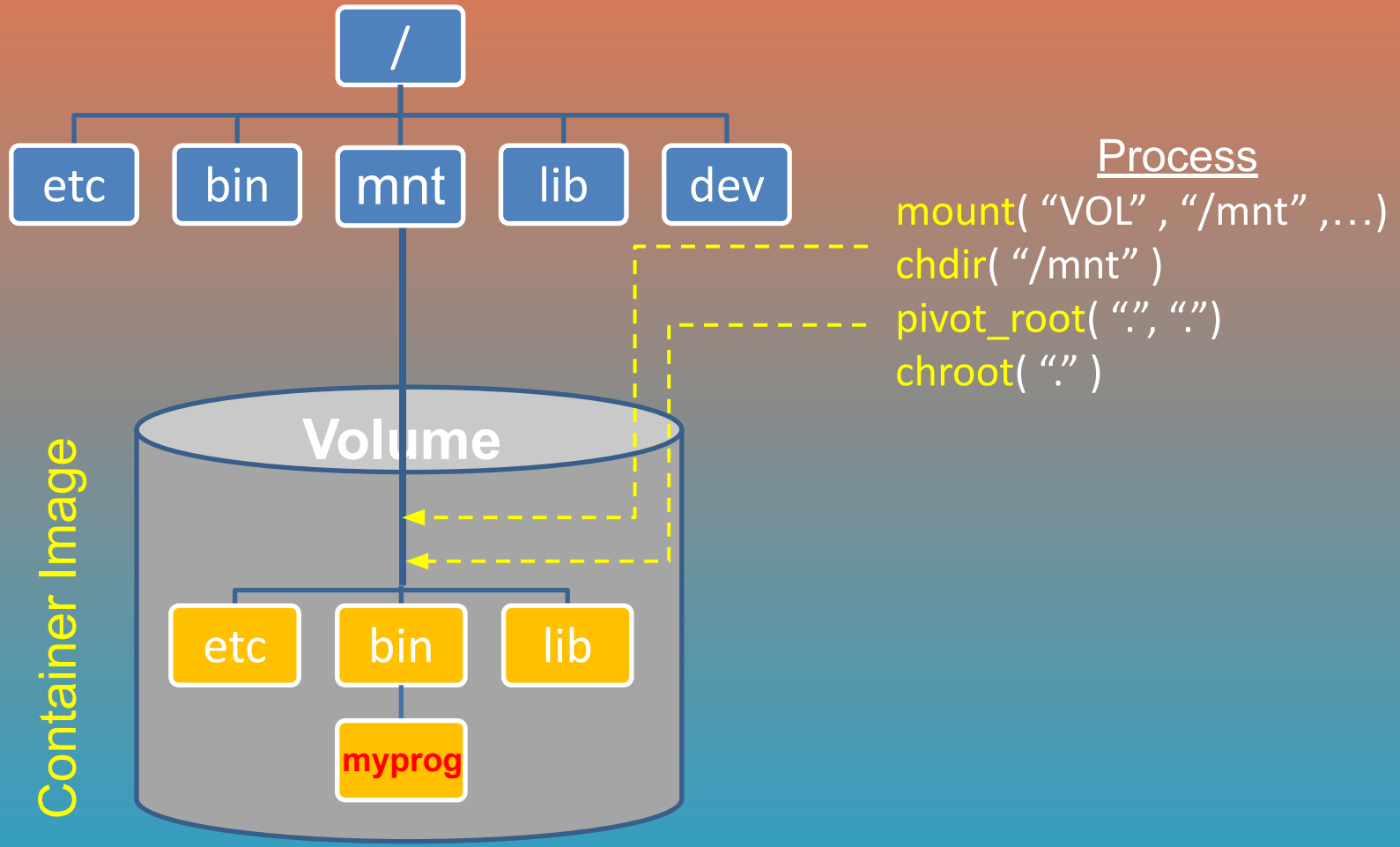
Isolation

Host



Linux containers

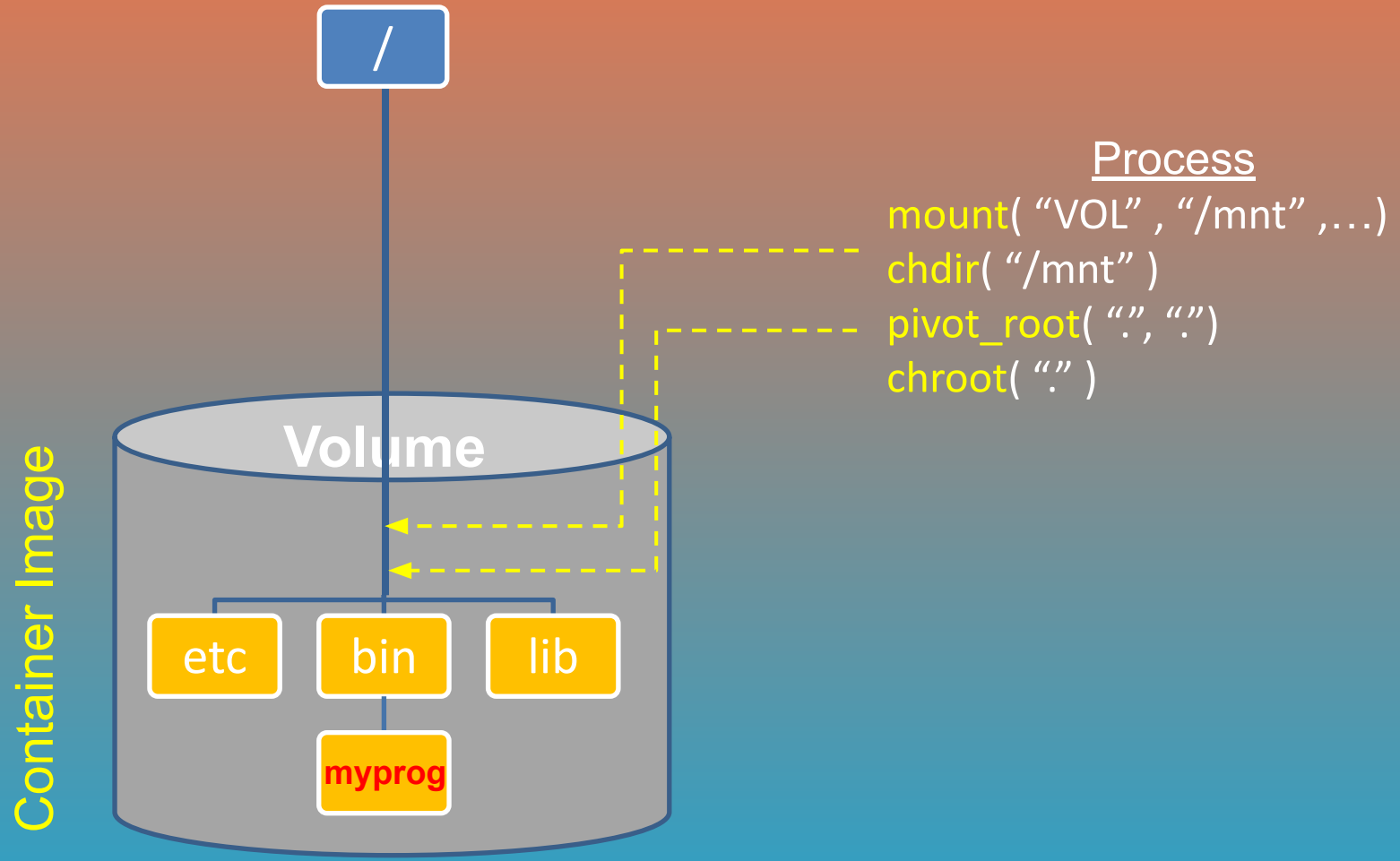
Isolation



Linux containers

Isolation

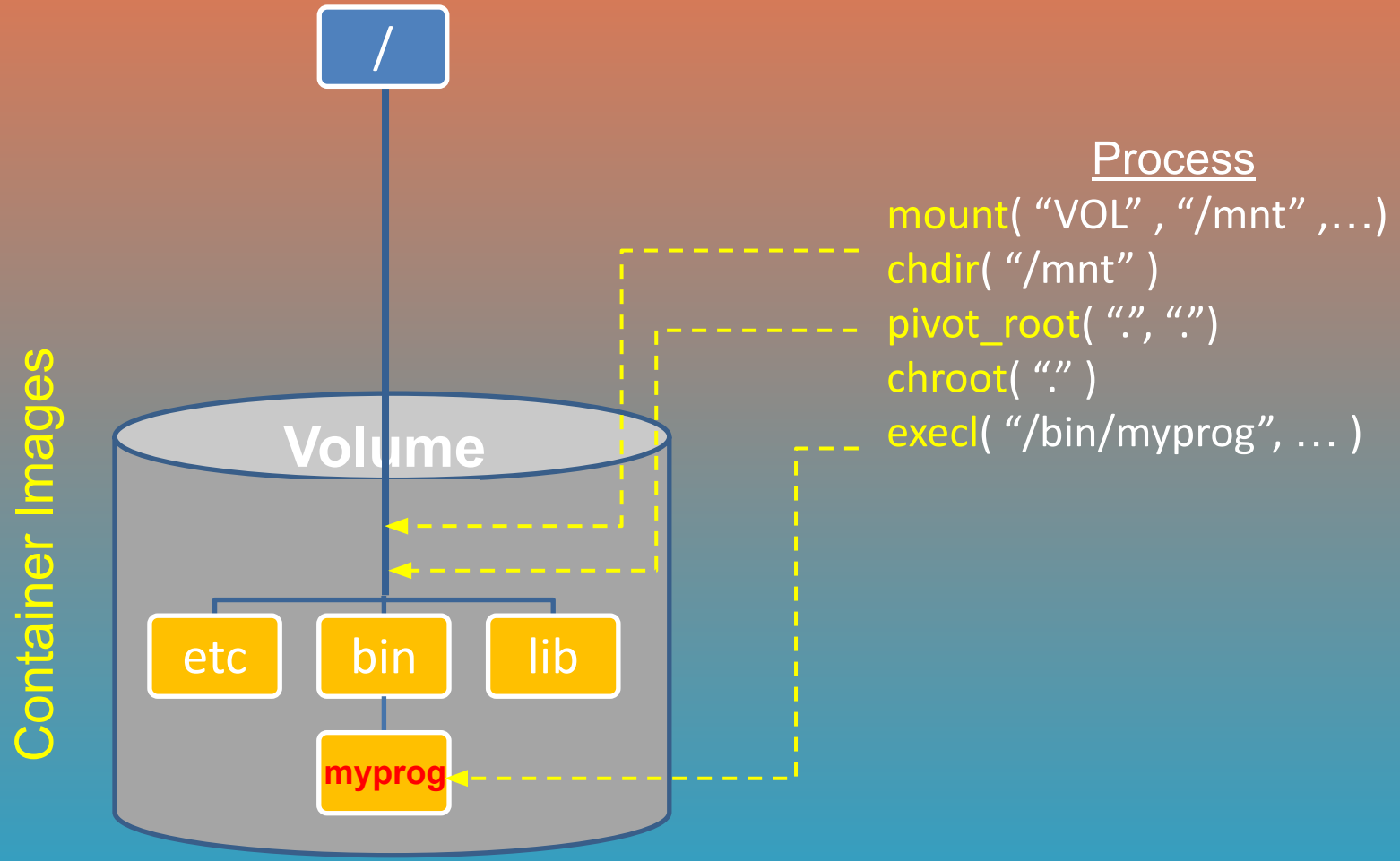
Host



Linux containers

Isolation

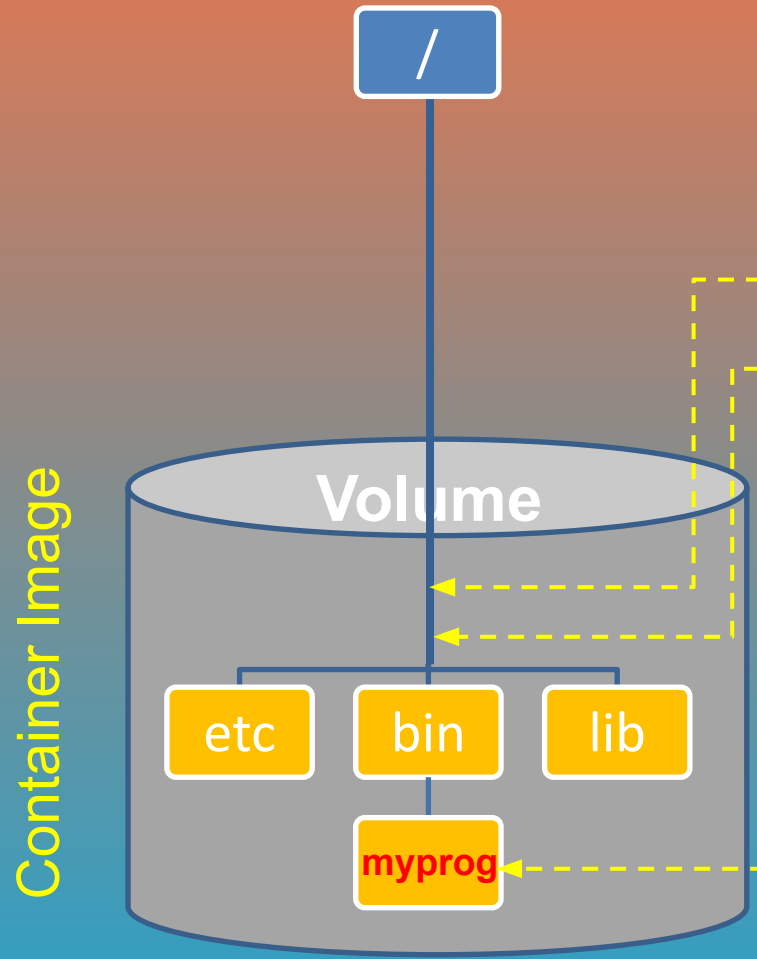
Host



Linux containers

Isolation

Host



Process

```
mount( “VOL” , “/mnt” ,…)
chdir( “/mnt” )
pivot_root( “.”, “.”)
chroot( “.” )
execl( “/bin/myprog”, … )
```

- Using mount usually requires privileges (CAP_SYS_MOUNT)
  - Can use FUSE e.g. libguestfs
- Using chroot and pivot_ root usually requires privileges (CAP_SYS_CHROOT)
  - Can use user namespace

Linux containers

Usual process isolation and limitation

- Kernel namespaces: isolate system resources from process perspective
  - Mount namespaces: isolate mount points
  - UTS namespaces: hostname and domain isolation
  - IPC namespaces: inter process communications isolation
  - PID namespaces: isolate and remap process identifiers
  - Network namespaces: isolate network resources
  - User namespaces: isolate and remap user/group identifiers
  - Cgroup namespaces: isolate Cgroup directories
- Seccomp: system call filtering
- Cgroups: process grouping and resource consumption limits
- POSIX capabilities: split/enable/disable root privileges
- chroot and pivot_root: isolated directory trees
- AppArmor and SELinux: kernel access control

Containerisation

In short

01 Run programs as processes in a standard way

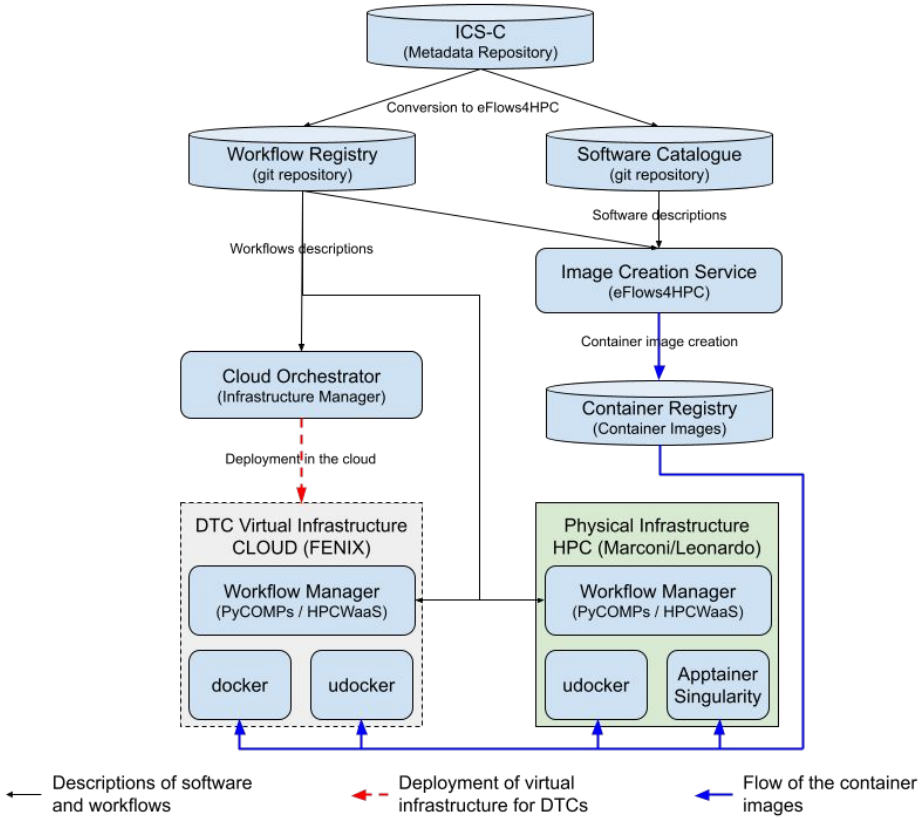
02 No hardware emulation or vm hypervisors

03 Just separation of process environments

04 Therefore simple and efficient

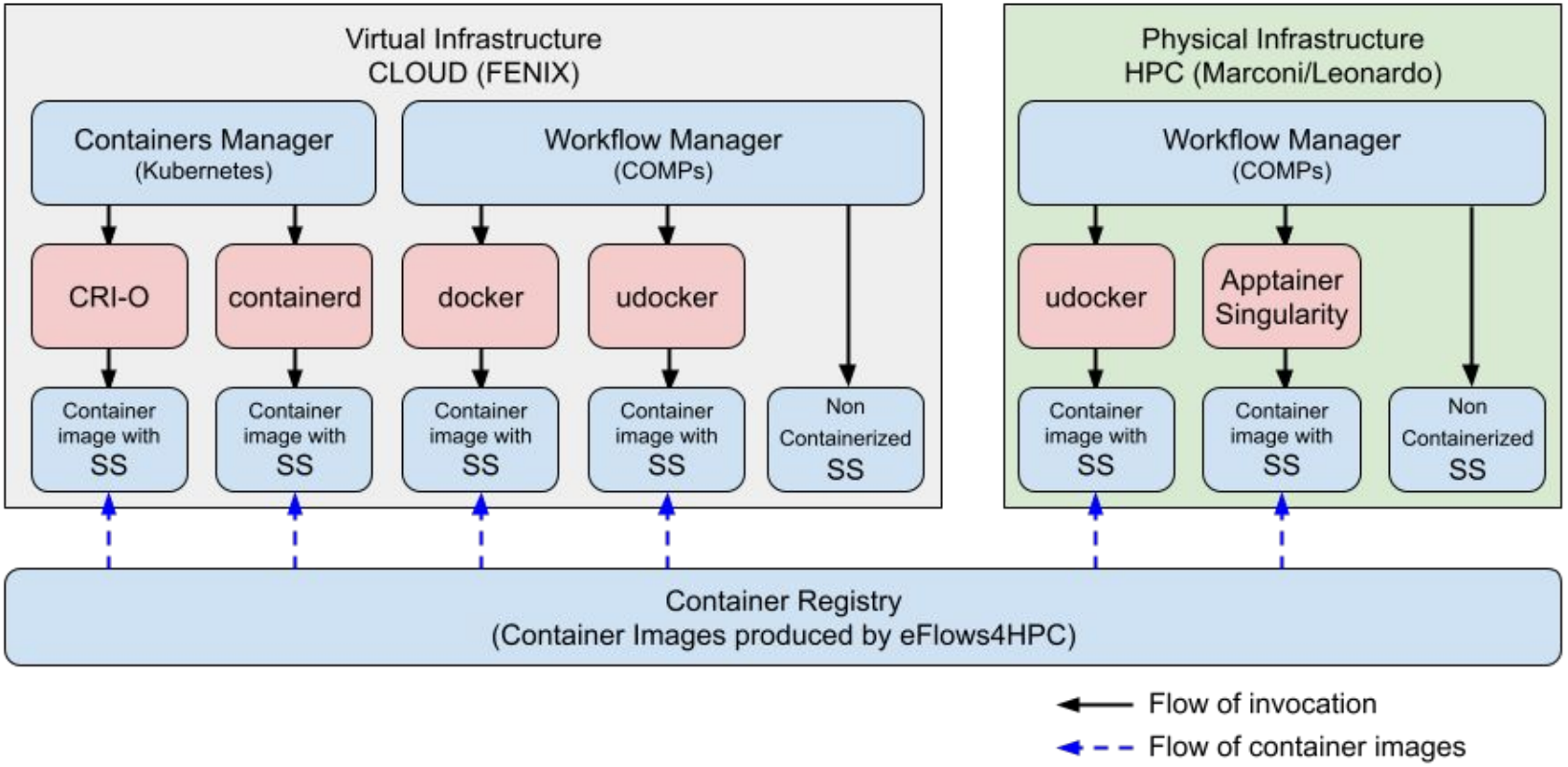
DT-GEO

From metadata to the execution of containerised applications and services



- Metadata repository
  - software, workflows, etc
- DevOps inspired approach
  - git repositories
- Container image creation
- Container image registry
- Quality assurance
- Infrastructure
  - Cloud based
  - HPC based
- PyCOMPs to implement workflows
- **Container execution engines**
  - **Singularity/Apptainer**
  - **docker**
  - **udocker**

Container execution engines



Workflows with COMPSs and containers

```
from pycompss.api.container import container
from pycompss.api.task import task
from pycompss.api.binary import binary
from pycompss.api.parameter import *

@container(engine="DOCKER", image="container-image-name")
@binary(binary="grep", working_dir=".")
@task(infile={Type:FILE_IN_STDIN}, result={Type:FILE_OUT_STDOUT})
def grepper():
     pass

if __name__=='__main__':
    infile = "infile.txt"
    outfile = "outfile.txt"
    grepper("THIS EXPRESSION", infile, outfile)
```

Container engine

Container image

Executable in image

Input and Output files

Container image types

- Docker and OCI images
  - Widely used and supported formats, OCI is a standard
    - docker, udocker, Kubernetes, podman, Singularity, Apptainer …

- Singularity images
  - Specific format of Singularity
    - Singularity, Apptainer …

Containers and infrastructure environments

- Cloud
  - docker: simple container execution or execution via COMPSs
  - Kubernetes: execution of containerised services with scalability and HA

- HPC
  - udocker: execution everywhere, execution across heterogeneous hosts, execution without namespaces, privileges or other dependencies
  - Singularity or Apptainer: execution in HPC environments, singularity image format may yield faster file access within the container

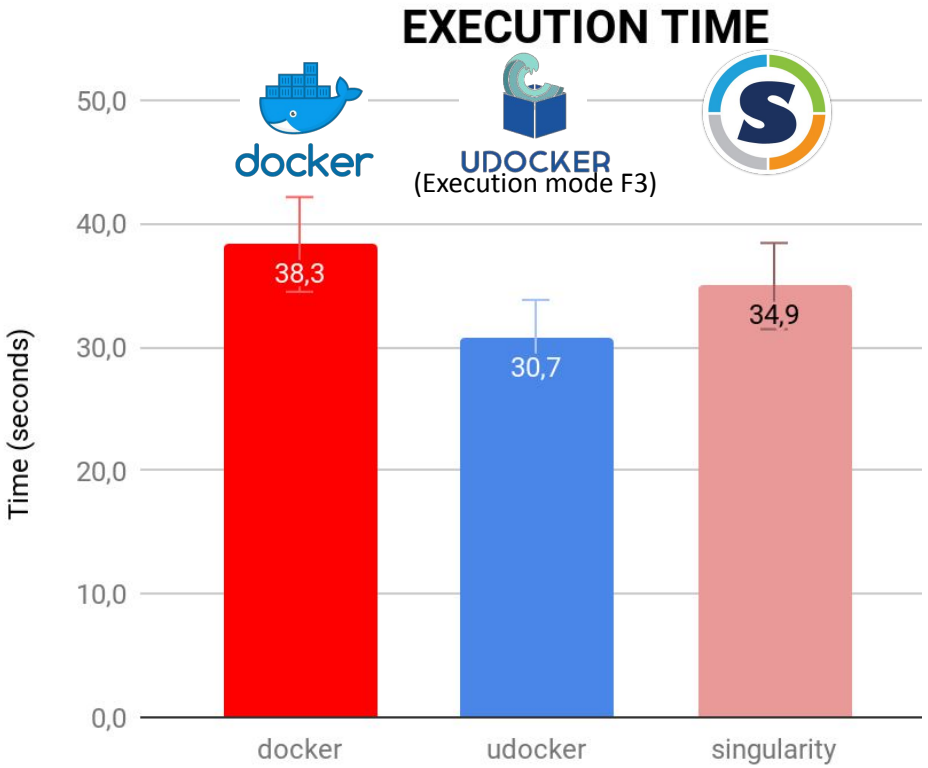
Container execution engines

Comparison

	Open source	User deploy and execute	Image Types			Isolation Method			Infrastructure	
			OCI images	docker images	Singularity images	namespaces	system call intercept	shared lib call intercept	HPC and batch	CLOUD VM
docker	✓	✗	✓	✓	✗	✓	✗	✗	✗	✓
singularityCE	✓	✗	✓	✓	✓	✓	✗	✗	✓	
singularityPRO	✗	✗	✓	✓	✓	✓	✗	✗	✓	
apptainer	✓	✗	✓	✓	✓	✓	✗	✗	✓	
podman	✓	✗	✓	✓	✗	✓	✗	✗		✓
kubernetes	✓	✗	✓	✓	✗	✓	✗	✗	✗	✓
udocker	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓

Container execution engines

Performance



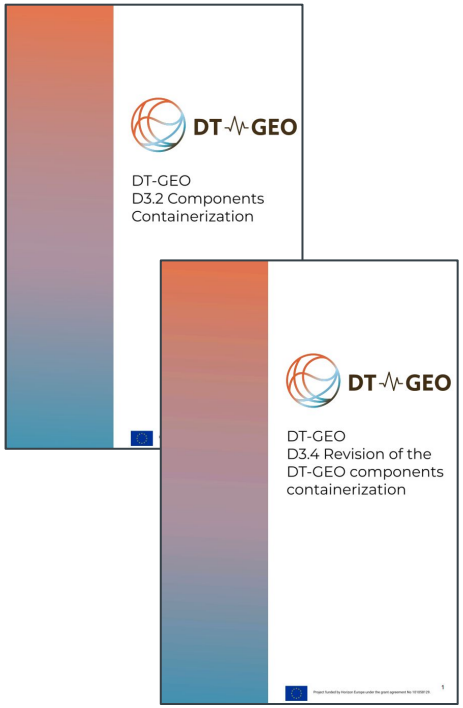
Train a model to recognize handwritten digits (the MNIST data set).

DT-GEO containerisation

Deliverables

- D3.2 Components Containerization
  - Containerization why and how
  - Relevance for digital twins
  - Plans for containerization from DTCs

- D3.4 Revision of the DT-GEO components containerization
  - Update on containerization
  - Revised plans and status from DTCs



DT-GEO Containerisation

Status

Containerisation plans

Software components	82	Steps in workflows or services
To be containerised	64	Services and applications for HPC and cloud
No plans for containerisation	18	Most correspond to simple portable codes

Facility

HPC	22	4 require MPI, 9 require GPU
Cloud, FENIX	36	2 may require MPI, 4 require GPU
Local	19	3 require GPU
Not yet known	19	

DT-GEO Containerisation

Status

Image type

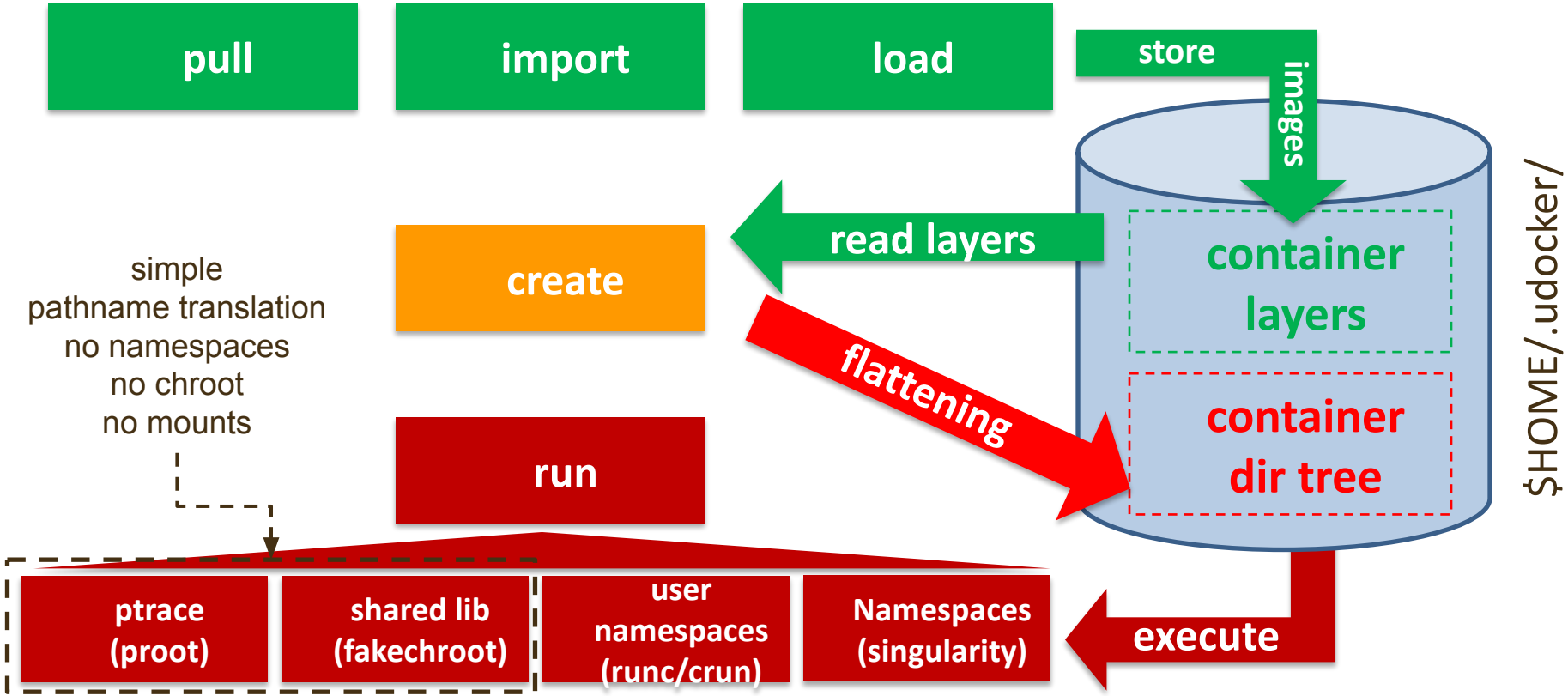
Docker	63	Can be executed with docker or udocker
Singularity	22	Must be executed with singularity

Container engine

docker	51	In cloud or local execution
udocker	42	In batch systems
singularity	30	In batch systems
kubernetes	5	In cloud

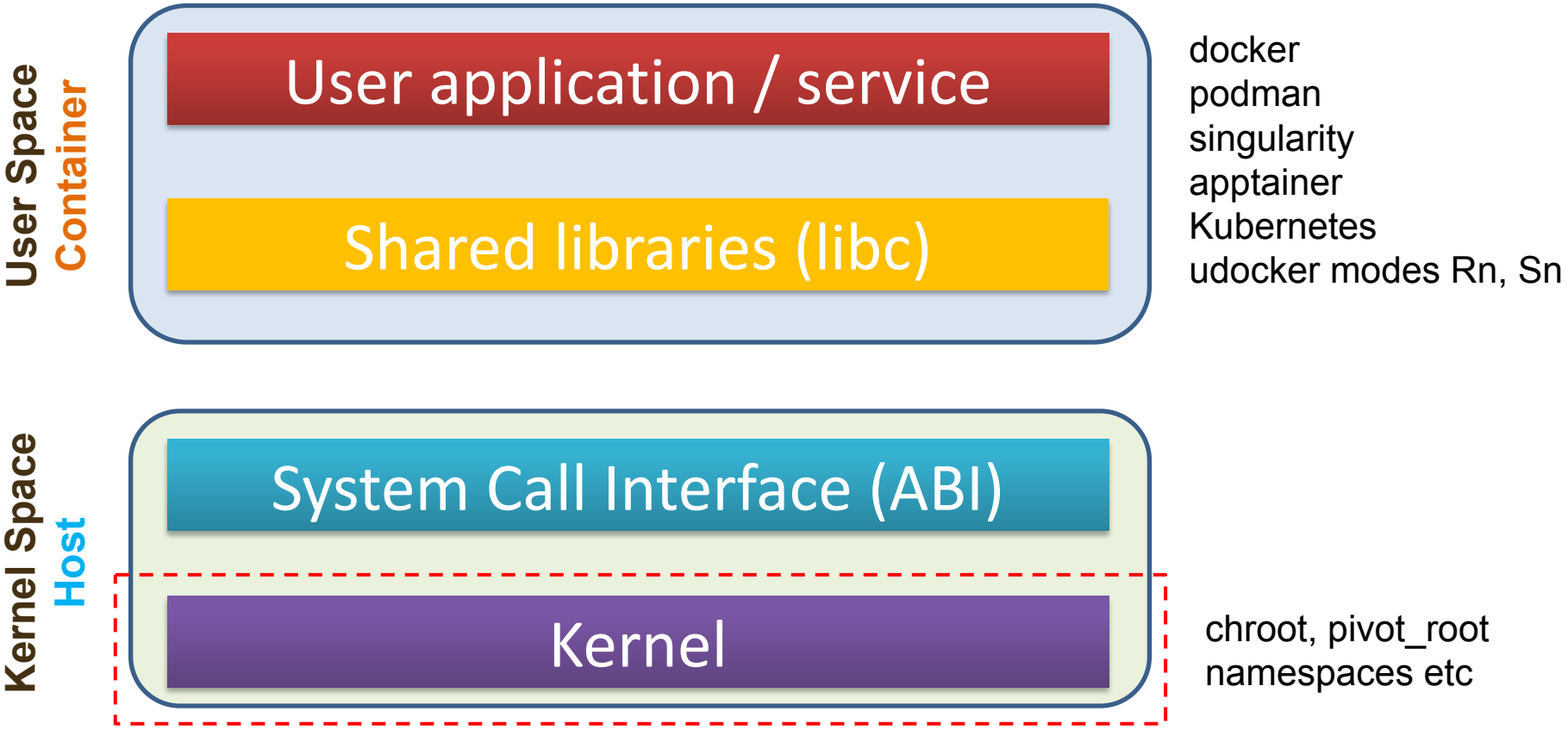
udocker

Architecture



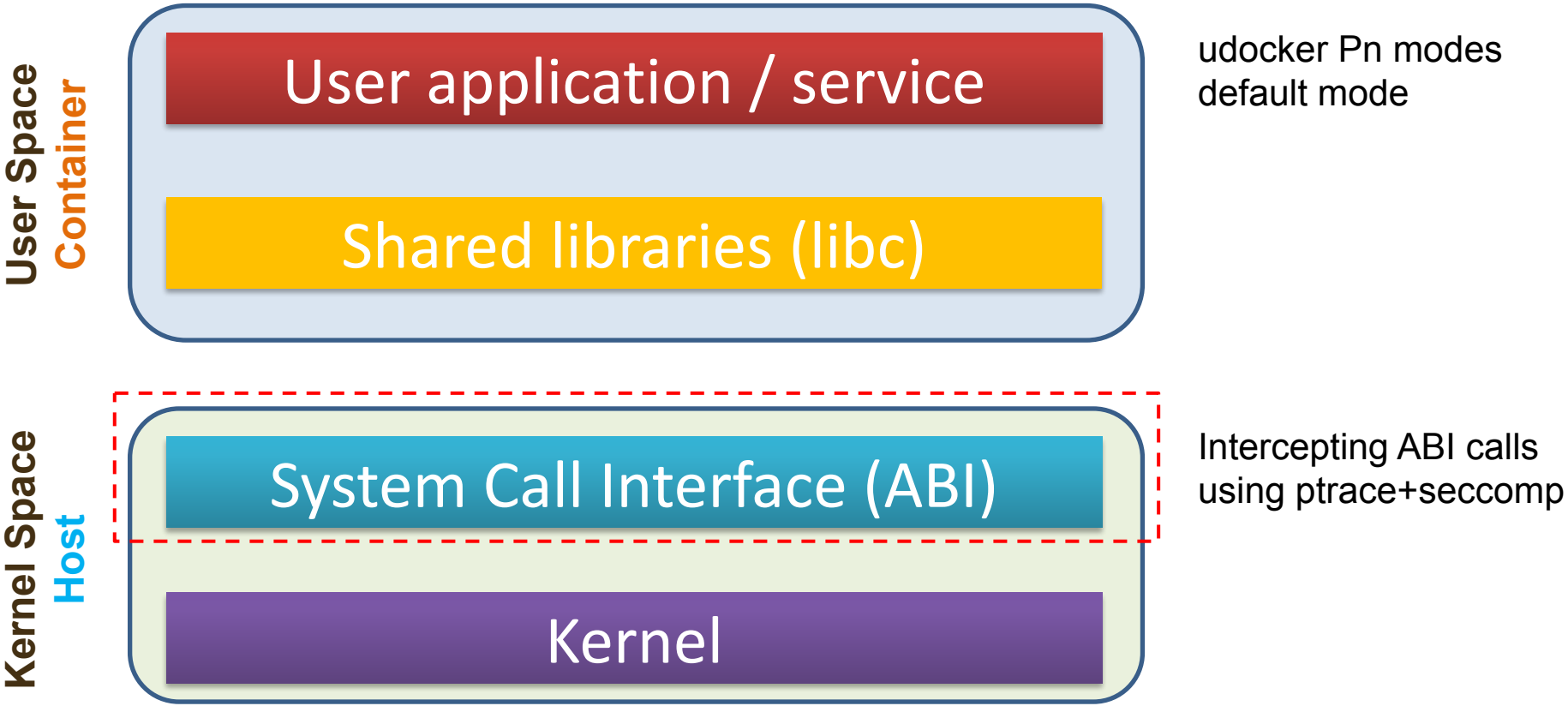
Isolation

Using Kernel functionalities



Isolation

Using system call interception

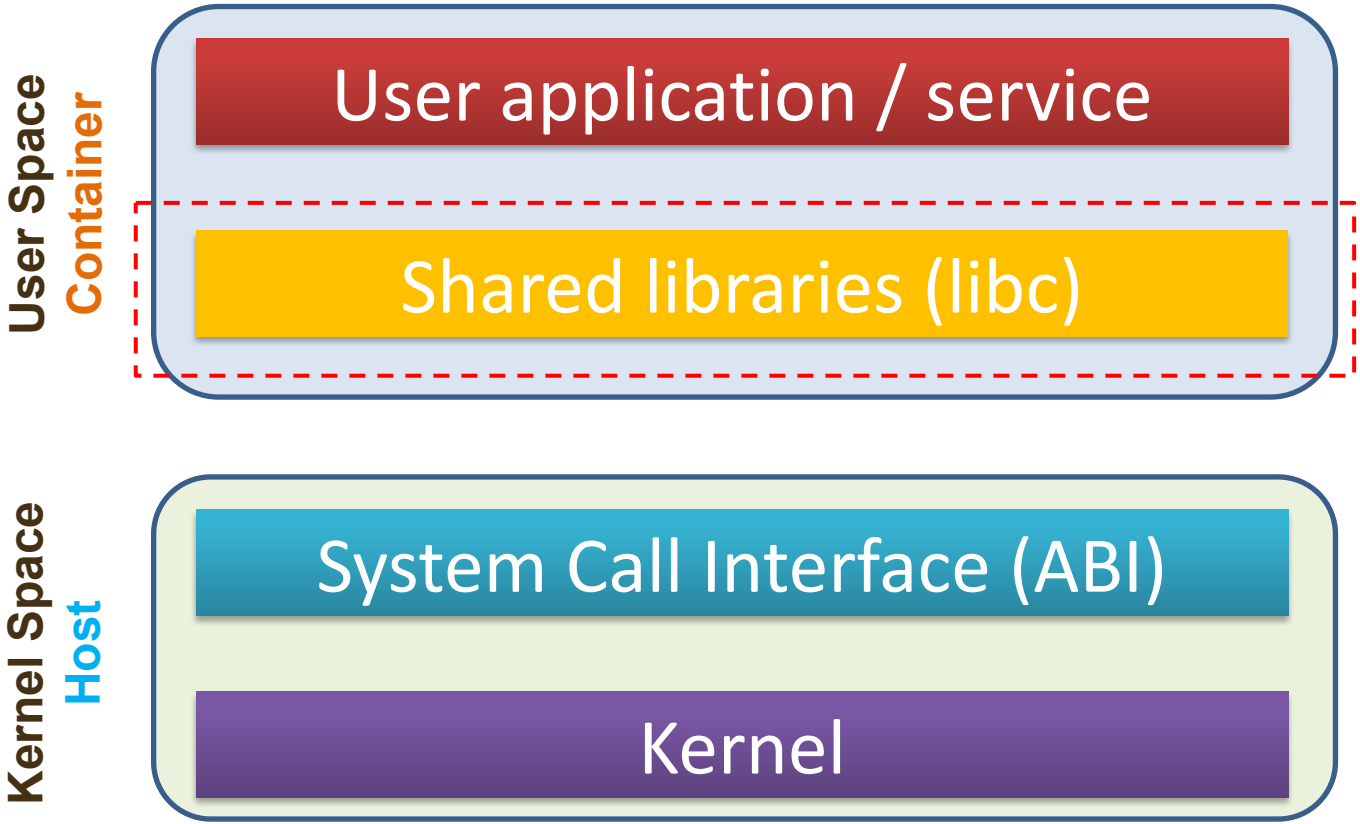


udocker Pn modes
default mode

Intercepting ABI calls
using ptrace+seccomp

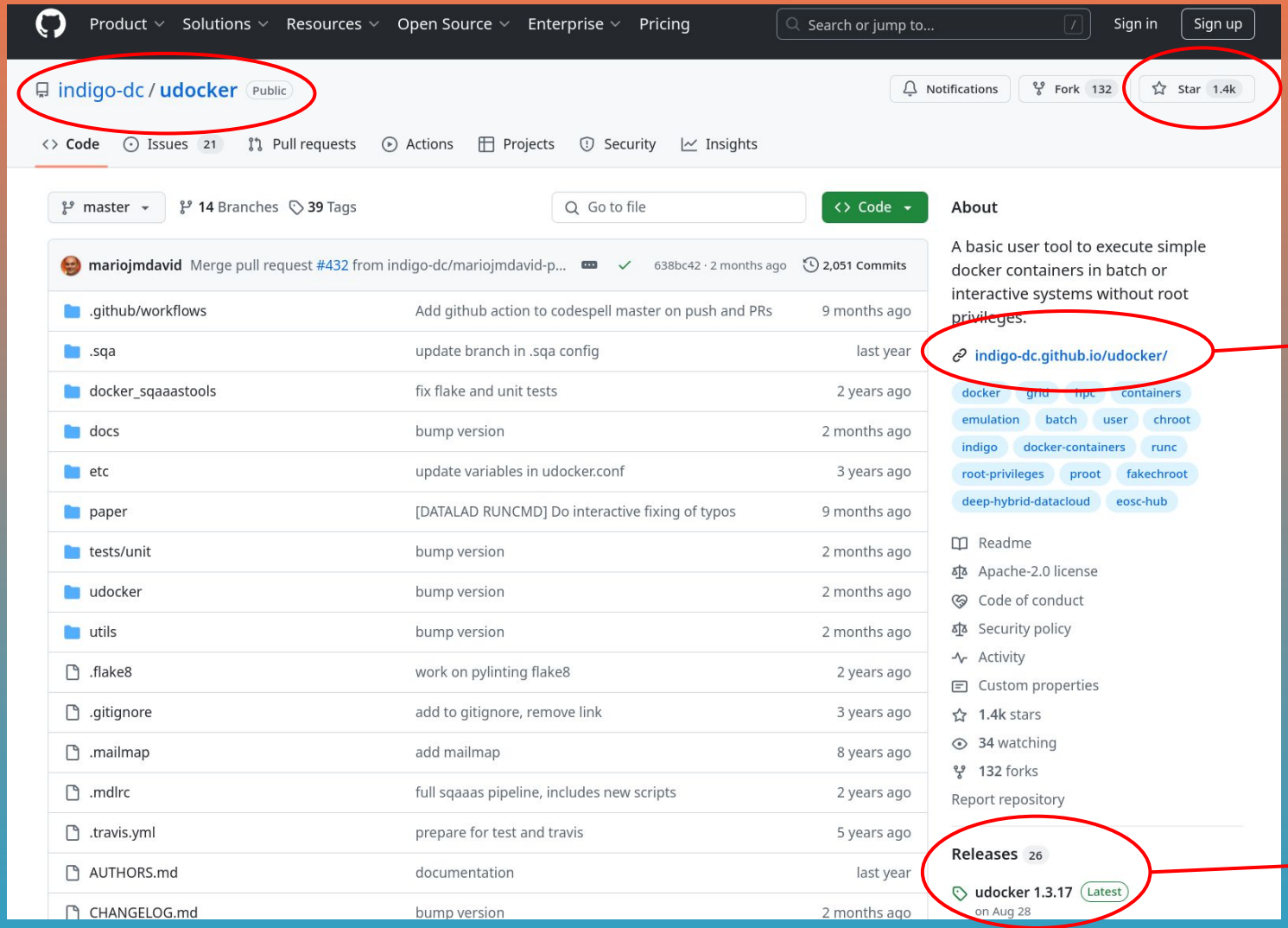
Isolation

Using shared library call interception



udocker Fn modes

Intercepting shared library calls using the loader LD_PRELOAD



GitHub: indigo-dc/udocker

Developed by LIP in the Indigo-dataCloud

Documentation

Releases

udocker

Execution modes

Mode	Base	Description
P1	PRoot	PTRACE accelerated (with SECCOMP filtering) DEFAULT
P2	PRoot	PTRACE non-accelerated (without SECCOMP filtering)
R1	runC / Crun	rootless unprivileged using user namespaces
R2	runC / Crun	rootless unprivileged using user namespaces + P1
R3	runC / Crun	rootless unprivileged using user namespaces + P2
F1	Fakechroot	with loader as argument and LD_LIBRARY_PATH
F2	Fakechroot	with modified loader, loader as argument and LD_LIBRARY_PATH
F3	Fakechroot	modified loader and ELF headers of binaries + libs changed FASTER
F4	Fakechroot	modified loader and ELF headers dynamically changed
S1	Singularity	where locally installed using chroot or user namespaces

udocker

Installation

```
$   curl  -L  \
    https://github.com/indigo-dc/udocker/releases/download/1.3.17/udocker-1.3.17.tar.gz \
      >  udocker-1.3.17.tar.gz

    ##  untar the Python code. It is extracted to a directory called udocker

$   tar  zxvf  udocker-1.3.17.tar.gz

    ##  optionally add the just created udocker directory to the PATH

$   export  PATH=$(pwd)/udocker-1.3.17/udocker:$PATH

    ##  install the binaries required to execute containers under $HOME/.udocker

$   udocker  install

$   udocker  version
```

udocker

Improvements

- Improved MPI support
  - Improved PMI interface to the batch system
- Handling of different platforms
  - Support for multiple architectures within udocker internals
  - Support for multiplatform image manifests version 2 schema 2
  - Pull or load images for architectures different from the host system
  - Emulation using QEMU user space in Pn modes
- Execution in different platforms
  - Pn: x86_64, x86, arm, arm64, armel, armhf
  - Rn: x86_64, arm64, ppc64le
  - Fn: x86_64, arm64, ppc64le

Containerization

Conclusions

- DT-GEO digital twin components
  - Include different applications and services
  - They will run on different environments both cloud and HPC
  - Workflows will be executed via PyCOMPSs that will invoke the containerized codes
- Different container execution tools are supported depending on
  - Application or service requirements
  - Hosting infrastructure (cloud, HPC etc)
- Next steps until the end of the project
  - Containerize applications and services as they became available

THANK YOU

udocker@lip.pt

@dtgeo_eu

linkedin.com/company/dt-geo/

DT-GEO Containerisation

WP5 Volcanoes

DTC-V1 Volcanic unrest	GALES code is available at https://gitlab.com/dgmaths9/gales Most SS are very simple and should not require containerization Some use machine learning and these should be containerized
DTC-V2 Forecast of volcanic ash clouds and fallout	Plan docker images for use with docker or udocker MET-get-GFS docker image in dockerhub MET-get-ERA5 docker image in dockerhub Singularity also being considered for HPC environments
DTC-V3 Lava flows	Plan to use docker and Singularity for SS5301 Plan to use docker or udocker for the others
DTC-V4 Volcanic gases	Plan to use udocker, much of the code has git repositories

DT-GEO Containerisation

WP6 Tsunamis

DTC-T1 Tsunami forecast Seissol	Docker container images in Docker Hub: https://hub.docker.com/u/seissol Singularity container built specifically for the Leonardo HPC system
DTC-T1 Tsunami forecast Tsunami-HySEA	Singularity container built with eFlows4HPC image creation service for Leonardo HPC
DTC-T1 Tsunami forecast Landslide-HySea	Singularity container image to be built
DTC-T1 Tsunami forecast BingClaw	Docker and singularity container images available
DTC-T1 Tsunami forecast SHALTOP	Docker image available, plan to create a singularity image

DT-GEO Containerisation

WP7 Earthquakes

DTC-E1 Probabilistic Seismic Hazard and Risk Assessment	Docker image available in Docker Hub
DTC-E2 Earthquake short-term forecasting	Docker container image to be build for use with Docker and Kubernetes Quakeflow includes Kubernetes helm charts and dockerfiles
DTC-E3 Ground Motion Models	Docker container images to be built
DTC-E4 Fault rupture forecasting	Docker or Singularity images, to be used with docker, udocker or Singularity
DTC-E5 Tomography and shaking simulation	Docker images, to be used with docker, udocker or Singularity
DTC-E6 Rapid event and shaking characterisation	Docker images, to be used with docker, udocker or Singularity

DT-GEO Containerisation

WP8 Anthropogenic Geophysical Extremes

DTC-AGEF1 Forecasting of long range responses of georeservoirs	Docker or Singularity image planned
DTC-AGEF2 Forecasting of late responses of georeservoirs	This DTC has the same SSs as DTC-AGEF1.
DTC-AGEF3 Modeling of maximum magnitudes	Docker or Singularity image planned
DTC-AGEF4 Induced seismic hazard map estimation	Docker or Singularity image planned Docker image planned

Installation

from a release

```
$   curl  -L  \
    https://github.com/indigo-dc/udocker/releases/download/1.3.17/udocker-1.3.17.tar.gz \
      >  udocker-1.3.17.tar.gz

    ##  untar the Python code. It is extracted to a directory called udocker

$   tar  zxvf  udocker-1.3.17.tar.gz

    ##  optionally add the just created udocker directory to the PATH

$   export  PATH=$(pwd)/udocker-1.3.17/udocker:$PATH

    ##  install the binaries required to execute containers under $HOME/.udocker

$   udocker  install

$   udocker  version
```

Installation

from the source

```
$   git  clone  https://github.com/indigo-dc/udocker.git

$   cd  udocker/udocker

    ## create a logical link

$   ln  -s  maincmd.py  udocker

    ##  optionally add the just created udocker directory to the PATH

$   export  PATH=$(pwd):$PATH

    ##  install the binaries required to execute containers under $HOME/.udocker

$   udocker  install

$   udocker  version
```

Installation

using PyPI

```
    ## Create Python 3 virtual env
$   python3 -m venv udockervenv
    ## activate the virtual env
$   source udockervenv/bin/activate

$   ## install udocker from PyPI
$   pip install udocker
    ## install the binaries required to execute containers
$   udocker install
$   udocker version
```

Installation

without outbound connectivity

```
$   wget \
      https://github.com/indigo-dc/udocker/releases/download/1.3.17/udocker-1.3.17.tar.gz

    ##  Get the additional tools (executables, libraries, etc)

$   wget \
      https://raw.githubusercontent.com/jorge-lip/udocker-builds/master/tarballs/udocker-englib-1.2.11.tar.gz

    ##  TRANSFER BOTH TARBALLS TO THE REMOTE SYSTEM and once transferred do:

$   tar zxvf udocker-1.3.17.tar.gz
$   export PATH=$(pwd)/udocker-1.3.17/udocker:$PATH

    ##  then install the binaries FROM THE TARBALL

$   export UDOCKER_TARBALL="udocker-englib-1.2.11.tar.gz"
$   udocker install
```

THANK YOU


udocker@lip.pt


@dtgeo_eu


linkedin.com/company/dt-geo/