Scientific Computing
with
Linux Containers

Infraestrutura Nacional de Computação Distribuída

LABORATÓRIO DE INSTRUMENTAÇÃO E FÍSICA EXPERIMENTAL DE PARTÍCULAS

CSIC
CONSEJO SUPERIOR DE INVESTIGACIONES CIENTÍFICAS

Jorge Gomes <jorge@lip.pt> / Isabel Campos <isabel@campos-it.es>

EOSC-hub
Cofinanciado por:
COMPETE 2020
PORTUGAL 2020
UNIÃO EUROPEIA
Fundo Europeu de Desenvolvimento Regional
Lisb@20^20
FCT Fundação para a Ciência e a Tecnologia

Scientific computing and containers

Running across systems often requires considerable effort

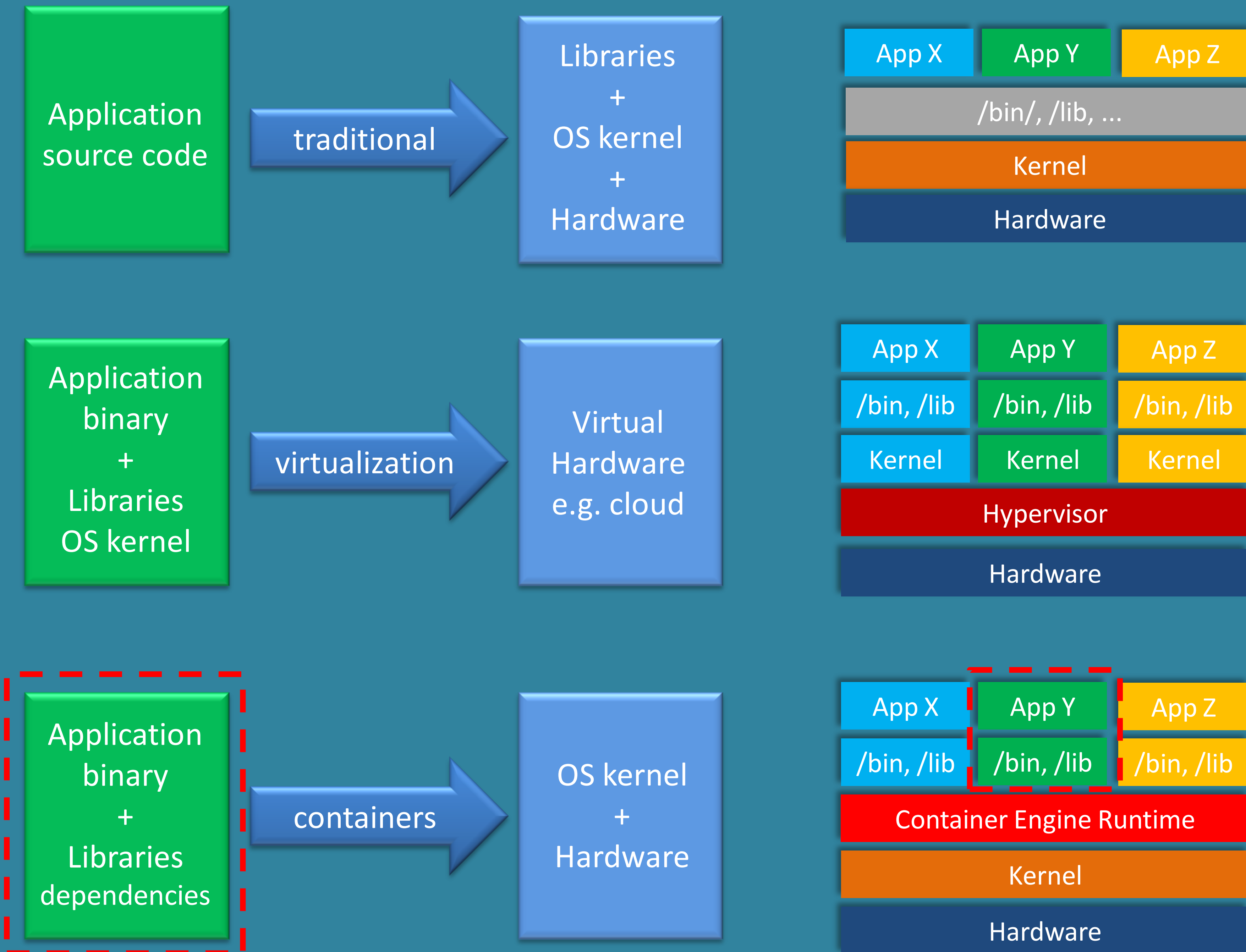
Computers
- Several computing systems
- Notebooks, Desktops, Farms, Cloud, Large HPC machines

OSes
- Several operating systems
- Linux flavors, Distributions, Versions

Environments
- Specific computing environments
- Compilers, Libraries, Customizations, Specialized Hardware

Applications
- Multiple applications
- Portability, Maintainability, Reproducibility

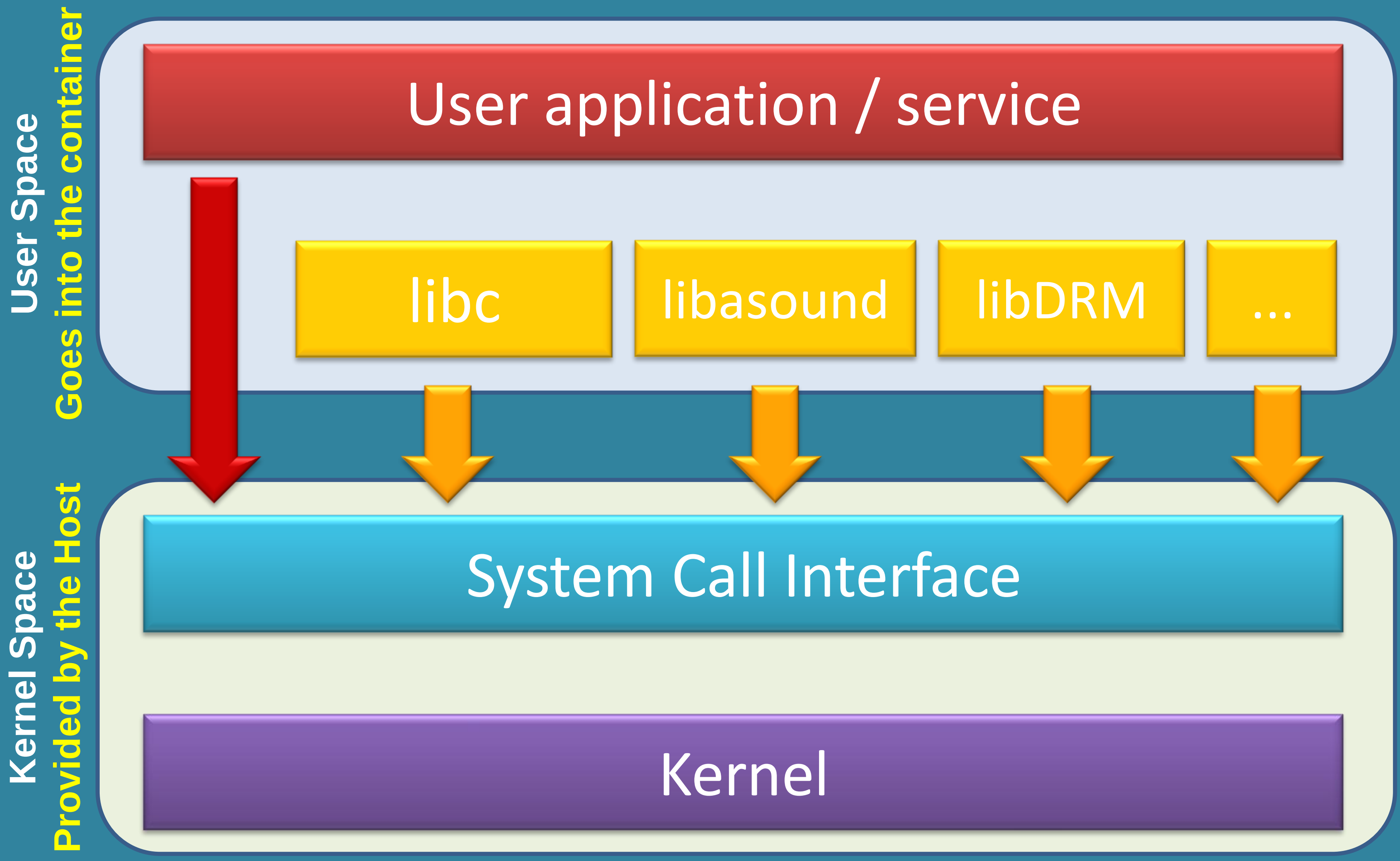
Containers can provide a consistent portable environments to execute software applications and services


Application source code
traditional
Libraries + OS kernel + Hardware
App X
App Y
App Z
/bin/, /lib, ...
Kernel
Hardware
Application binary + Libraries OS kernel
virtualization
Virtual Hardware e.g. cloud
App X
App Y
App Z
/bin, /lib
/bin, /lib
/bin, /lib
Kernel
Kernel
Kernel
Hypervisor
Hardware
Application binary + Libraries dependencies
containers
OS kernel + Hardware
App X
App Y
App Z
/bin, /lib
/bin, /lib
/bin, /lib
Container Engine Runtime
Kernel
Hardware

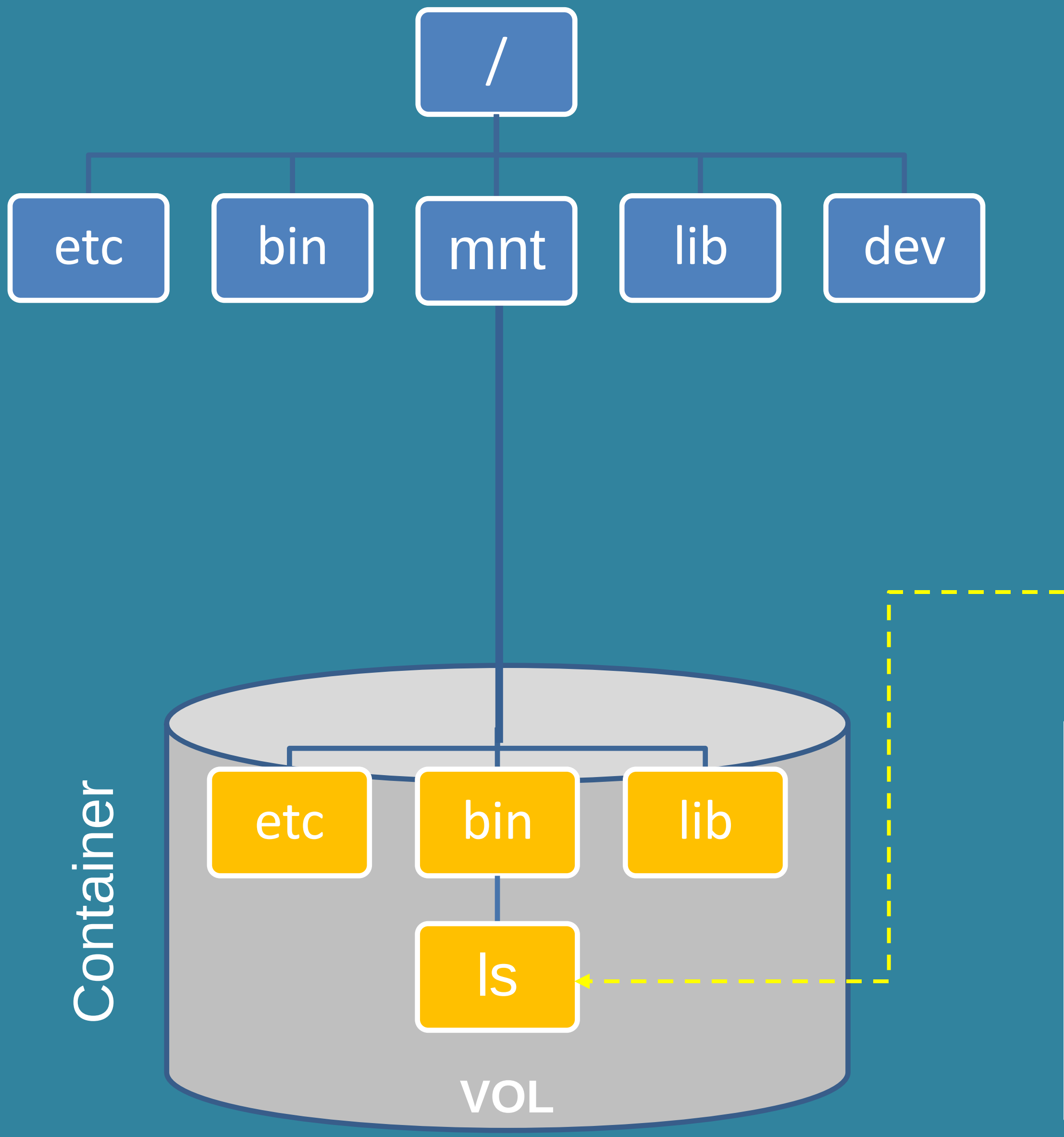
Advantages: Containers vs Traditional

- Encapsulation
  - Applications and dependencies are packed together
  - Portability across systems
  - Easier to distribute and share ready to use software
- Reproducibility
  - The whole run-time environment is in the container
  - Can be easily stored for later reuse, replay or preservation
- Isolation
  - Provides run-time environments that are independent from the host
  - May provide a limited root environment
  - May provide extra security to contain the applications
  - May provide resource usage limits for QoS
- Less effort
  - Easier maintenance and deployment

Linux System Call Interface (SCI)



Container file system with chroot




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

- 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 namespaces

Linux kernel features for isolation

- **chroot, pivot_root**: make a given directory root of the file system
- **Kernel namespaces**: isolate system resources from process
  - **Mount**: isolate mount points (cannot see host or other containers mounts)
  - **UTS**: virtualize hostname and domain
  - **IPC** : inter process communications isolation (semaphores, shmem, msgs)
  - **PID**: isolate and remap process identifiers (cannot see other processes)
  - **Network**: isolate network resources (interfaces, tables, firewall etc)
  - **cgroup**: isolate cgroup directories
  - **User**: isolate and remap user/group identifiers (user can be a limited root)
  - **Time**: virtualize boot and monotonic clocks
- **cgroups**: process grouping and resource consumption limits
- **seccomp**: system call filtering
- **POSIX capabilities**: split and drop root privileges
- **AppArmor** and **SELinux**: kernel access control

Linux user namespace

Available only on recent kernels/distributions

- Allows an unprivileged user to have a different UID/GID
- Enables an unprivileged user to become UID 0 root
- Enables executing the chroot and mount calls

- May require some setup of subuid and subgid files
- Network namespace becomes useless
- root has limitations
  - Cannot creates devices (mknod)
  - Cannot load kernel modules
  - Mount is restricted to some file system types
  - Issues on changing user ids group ids
  - Accessing files in the host (mount bind) can become problematic
- Not available/enabled in some distributions (notably RedHat/CentOS)

Containers

Run programs as processes in a standard way

No hardware emulation or vm hypervisors

Just a separate process environment

Therefore simple and efficient

docker

docker

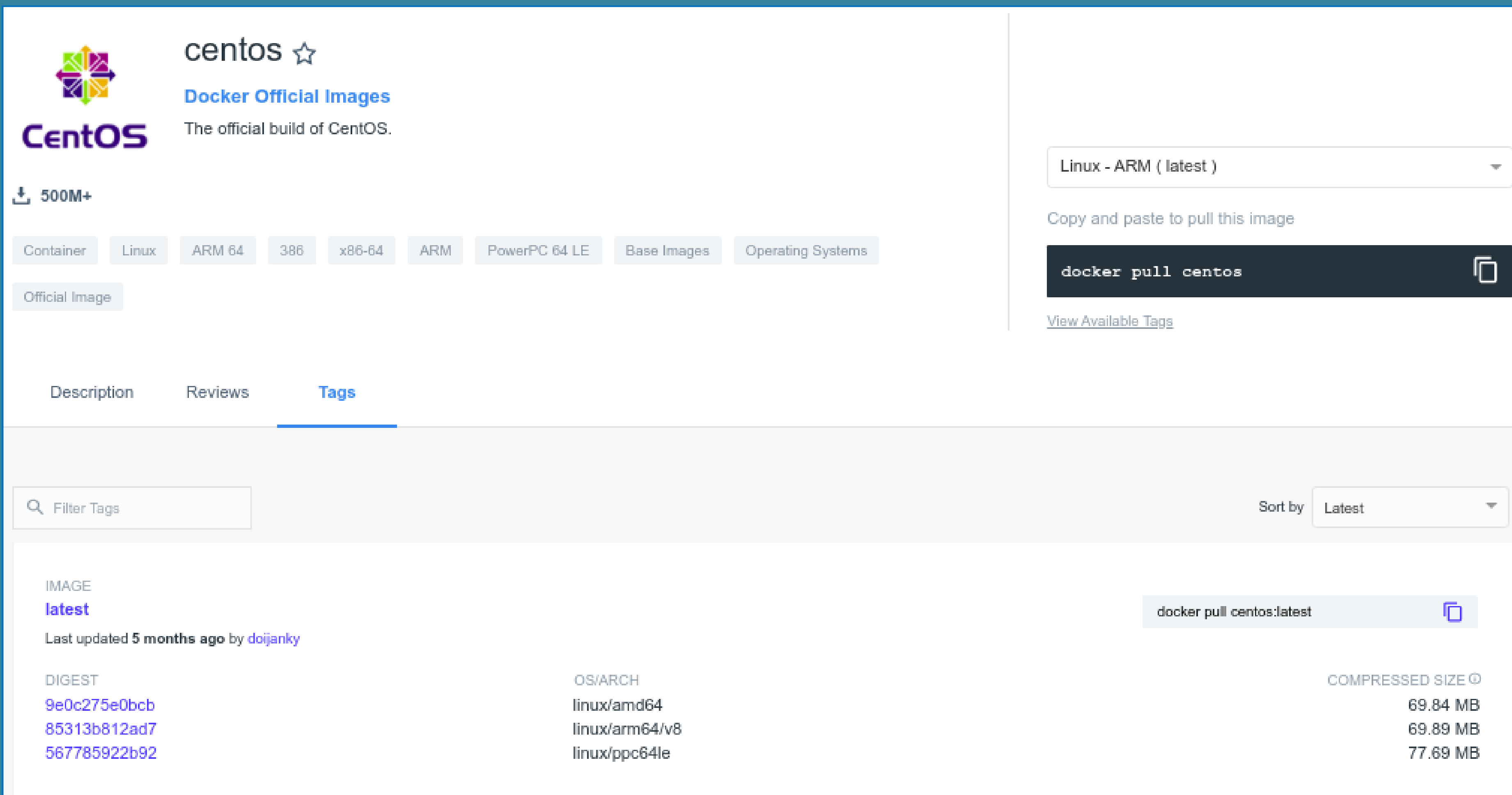
- **Docker is oriented to services and services composition:**
  - One service or application per container plus dependencies
  - Containers can be published in public or private repositories
  - Relies heavily on kernel functionalities such as namespaces
  - **Run the container everywhere (in any compatible Linux kernel)**

- **DevOps ➔ integration of IT development and operations**
  - docker has been a key technology enabling automation and DevOps
  - Developers: develop, produce containers, push them to production
  - Administrators: manage the underlying physical/virtual infrastructure

```
$ docker  run  -i  -t  centos:centos6
[root@28f89ada747e /]#  cat  /etc/redhat-release
CentOS release 6.8 (Final)
```

docker hub

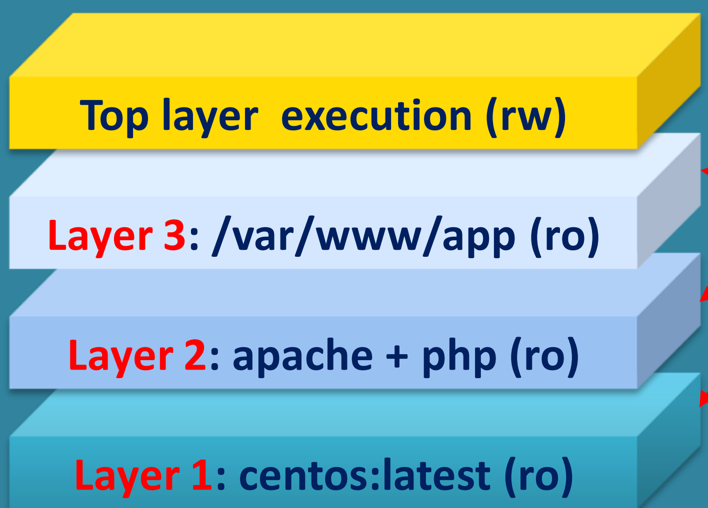
container images can be fetched from the docker hub repository



docker images

- Uses a layered file-system based
  - Implemented at host level by: AUFS
- New images can be easily created from existing ones
  - Created by using **Dockerfiles** and docker build
- Layers can be shared decreasing bandwidth and storage usage

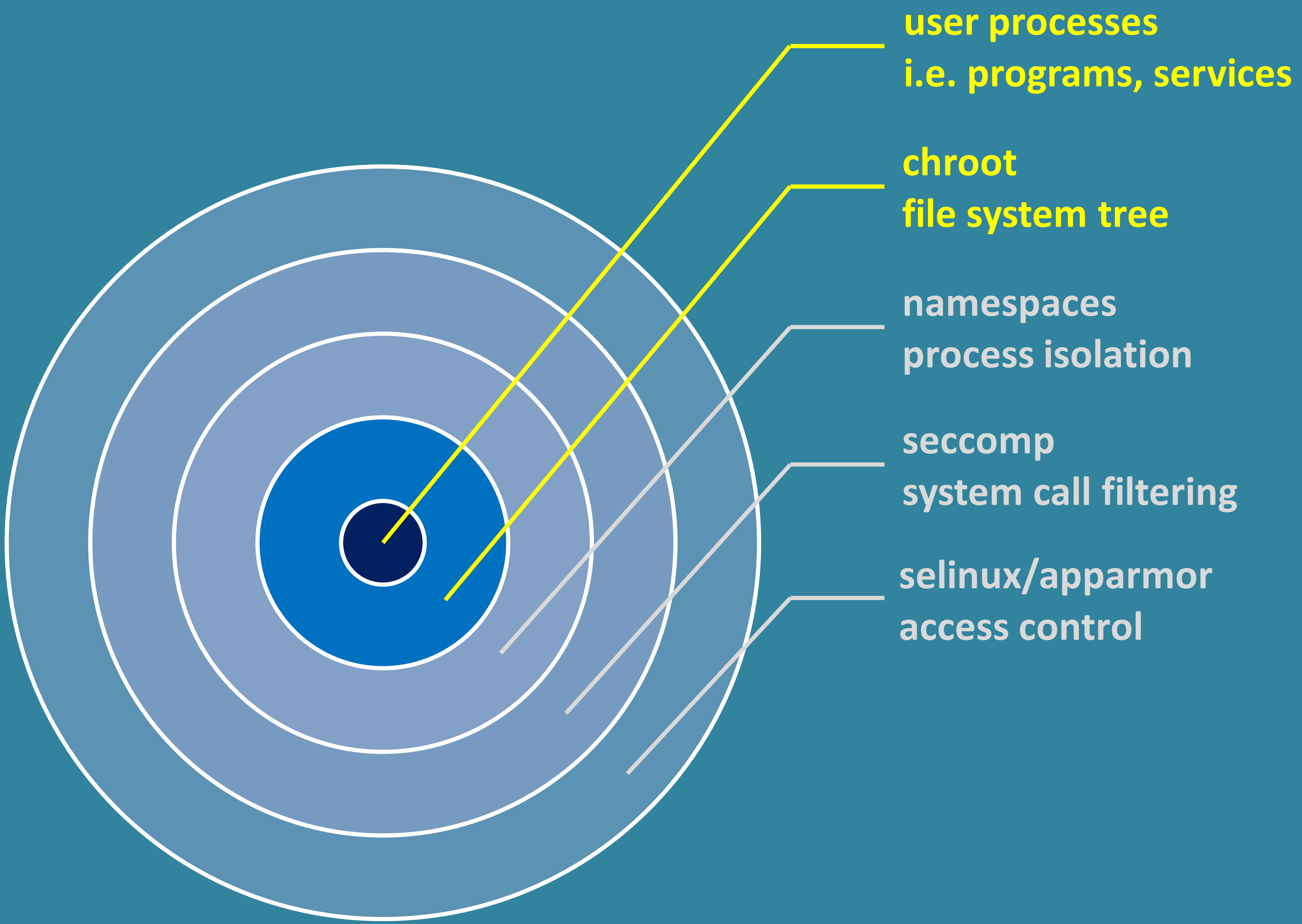
Layers



Dockerfile

1. FROM centos:centos6
2. RUN yum install –y httpd php
3. COPY /my/app /var/www/app
4. EXPOSE 80
5. ENTRYPOINT /usr/sbin/httpd
6. CMD [“-D”, “FOREGROUND”]

docker execution



docker limitations

Require root privileges to install, setup and run

- Raises security concerns especially in multi-user environments

docker API does not limit privileged actions

- Users with direct access to the API can do anything
- e.g: through the API users can mount local file systems, make devices accessible, etc.

Not oriented to end users

- docker is designed to be used as an hypervisor by DevOps & admins
- Client server model, processes run under the docker daemon
- Not suitable to batch systems because of process control, accounting and security
- Inside the container the user is usually root
- Requires separate network namespace, NAT and virtual networking

Other solutions

runC

Container engine originated by docker now developed by the Open Containers Initiative (OCI)

- Key aspects:
  - Is the runtime used by docker and other tools to execute containers
  - Contrary to docker has a fork and execute model (no daemon processes)
  - Focused on running images in OCI format
  - Requires privileges for full functionality
  - Can run without privileges using user namespaces

- Limitations:
  - Is mostly an execution runtime to be used by other tools
  - Downloading containers etc must be performed with other tools
  - Requires a description of the container environment OCI bundle spec
  - Running without privileges has limited functionality

Singularity

Container engine oriented to computing clusters

- Key aspects:
  - Has its own image format and repository
  - Can also pull images from docker
  - Fork and execute model (no daemon processes)
  - Meant to be used by the end-users
  - Requires installation by administrator and setuid privileges for full functionality
  - If setuid is unavailable can run without privileges using user namespaces

- Limitations:
  - History of security vulnerabilities
  - Running without privileges has limited functionality

Podman

Container engine for developing, managing, and running OCI Containers

- Key aspects:
  - Alternative drop-in replacement for docker
  - Has a fork and execute model
  - Uses the OCI images format, but also supports docker images
  - Requires privileges for full functionality
  - Can run without privileges using user namespaces

- Limitations:
  - Running without privileges has limited functionality
  - Not suitable for user execution with privileges via setuid

Charliecloud

Charliecloud

Engine oriented to run using user namespaces in computing clusters

- Key aspects:
  - Has a fork and execute model
  - Only runs without privileges using user namespaces
  - Executes a file-system tree already extracted to some directory

- Limitations:
  - Does not support pulling or extracting container images
  - Requires docker and/or other tools for most operations except running the container
  - Same limitations that apply to user namespaces

UDOCKER

udocker motivations

UDOCKER

Run applications encapsulated in docker containers:

- without using docker
- without using privileges
- without system administrators intervention
- without additional system software

and run:

- as a normal user from the command line
- fork and execute model
- normal process controls and accounting apply
- suitable for <u>interactive</u> or <u>batch</u> systems

Empowers end-users to run applications in containers

Search or jump to... | Pull requests | Issues | Marketplace | Explore

indigo-dc / **udocker**

Unwatch 31 | Star 695 | Fork 77

Code | Issues 41 | Pull requests 5 | Actions | Projects 0 | Wiki | Security 0 | Insights | Settings

Branch: master | udocker / **README.md** | Find file | Copy path

jorge-lip Update README.md — 3a3cce7 on Jan 23

4 contributors

347 lines (277 sloc) | 13.3 KB | Raw | Blame | History

build passing

UDOCKER

https://github.com/indigo-dc/udocker

- https://github.com/indigo-dc/udocker/tree/**master**
- https://github.com/indigo-dc/udocker/tree/**devel**

Python 2 & 3

- https://github.com/indigo-dc/udocker/tree/**devel3**

udocker is a basic user tool to execute simple docker containers in user space without requiring root privileges. Enables download and execution of docker containers by non-privileged users in Linux systems where docker is not available. It can be used to pull and execute docker containers in Linux batch systems and interactive clusters that are managed by other entities such as grid infrastructures or externally managed batch or interactive systems.

udocker other advantages

UDOCKER

Install:

- Just get the udocker python script and execute
- No need to install or compile additional software
- No need of system administrator intervention

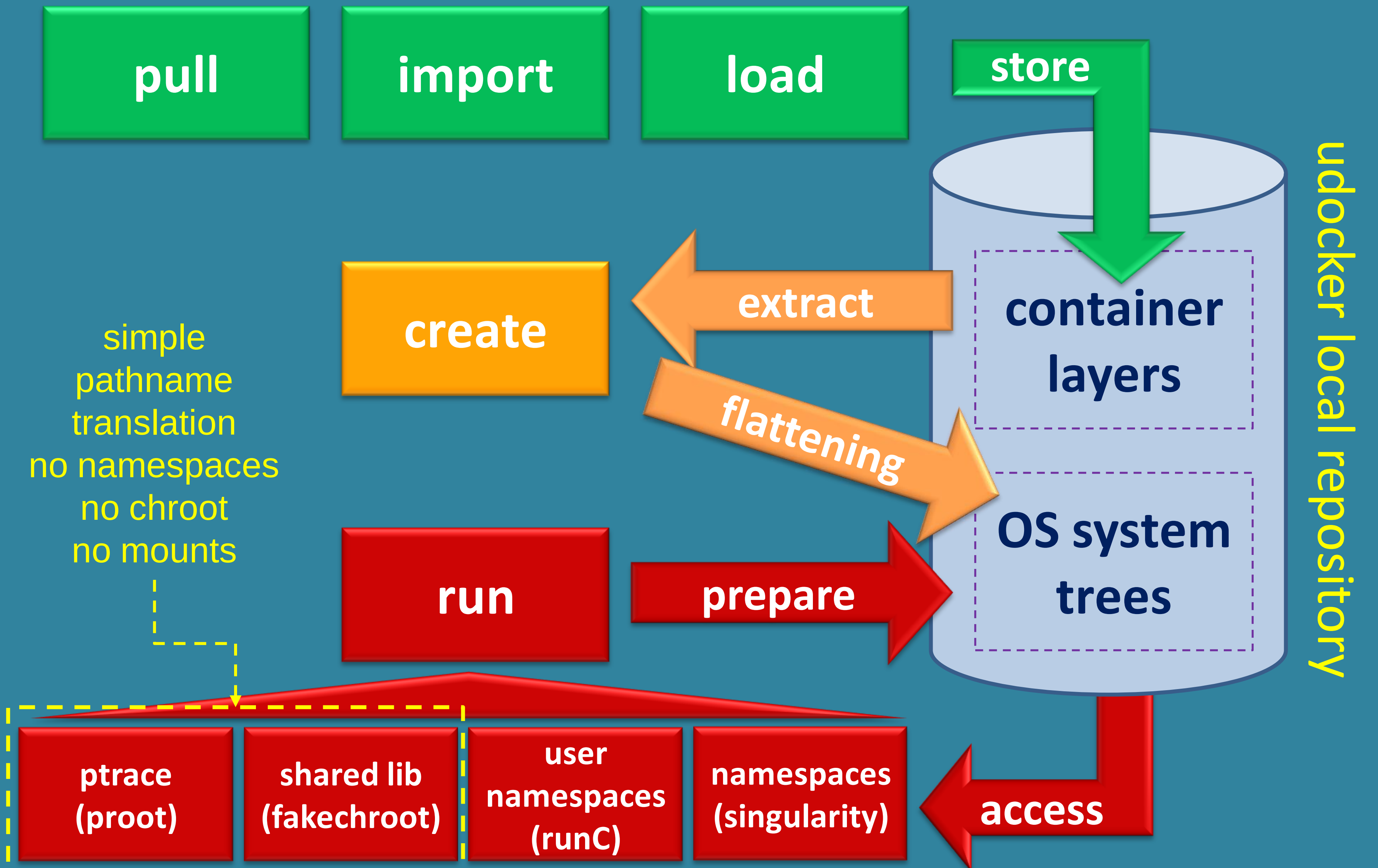
Get images:

- Pull containers from docker compatible repositories
- Load and save docker and OCI formats
- Import and export tarballs
- Extract images to file system

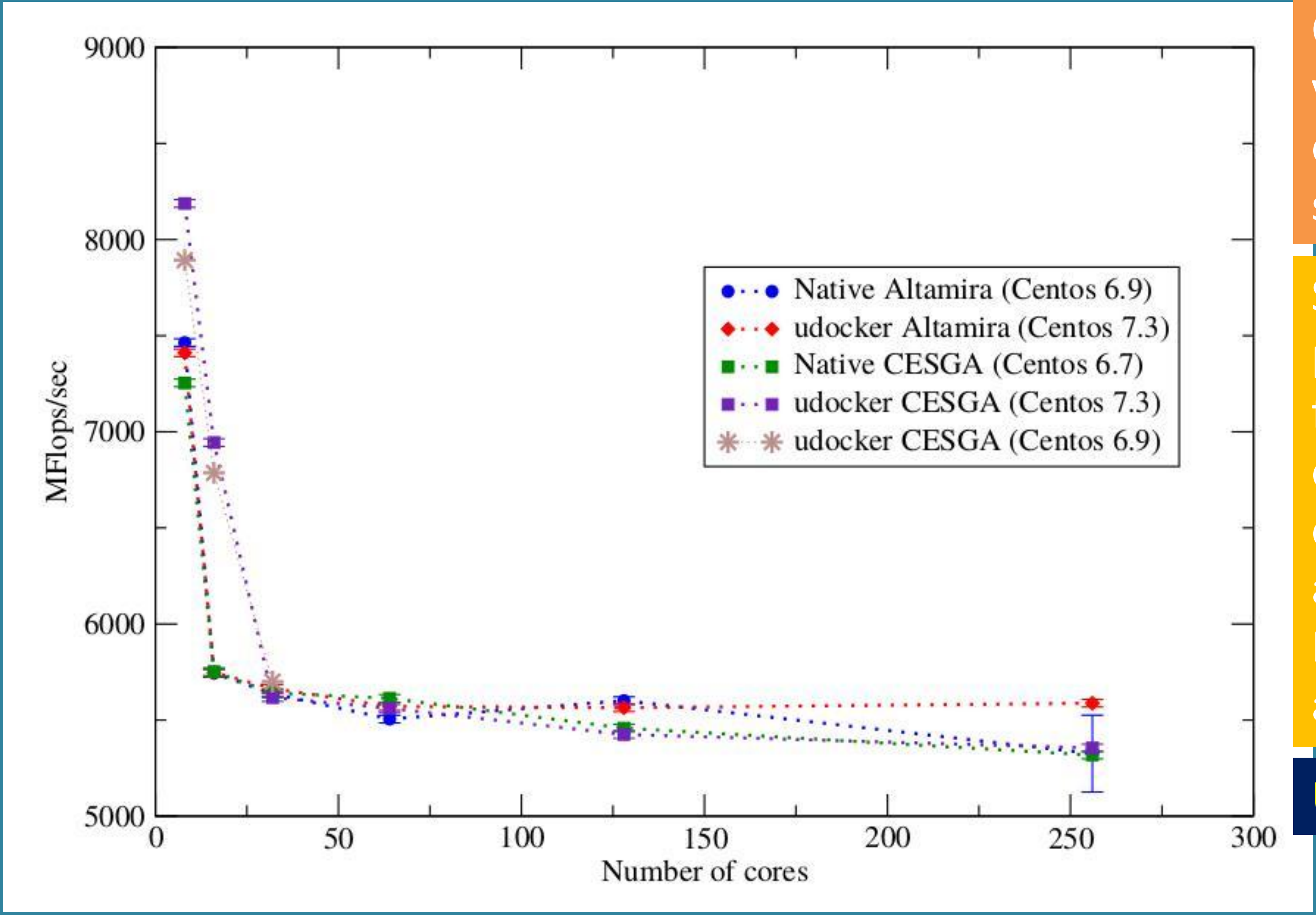
Run:

- Integrates and provides several execution engines

udocker is an integration tool



Lattice QCD



OpenQCD is a very advanced code to run lattice simulations

Scaling performance as a function of the cores for the computation of application of the Dirac operator to a spinor field.

Using OpenMPI

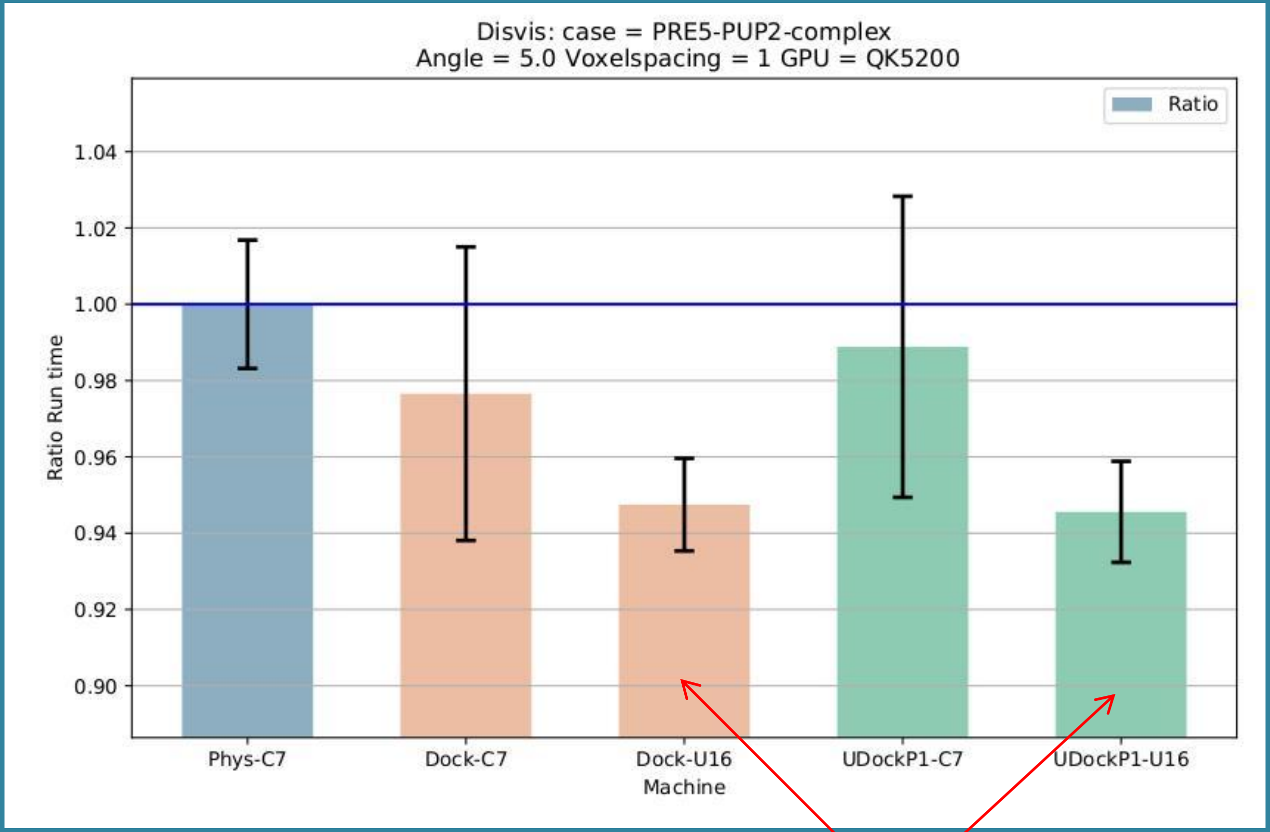
udocker in P1 mode

Running with udocker

UDOCKER

```
$ mpiexec -np 128 udocker run \
      -e LD_LIBRARY_PATH=/usr/lib \
      --hostenv \
      --hostauth \
      --user=cscdiica \
      -v /tmp \
      --workdir=/opt/projects/openQCD-1.6/main \
      openqcd \
      /opt/projects/openQCD-1.6/main/ym1 \
      -i ym1.in -noloc
```

Biomolecular complexes



DisVis is being used in production with udocker

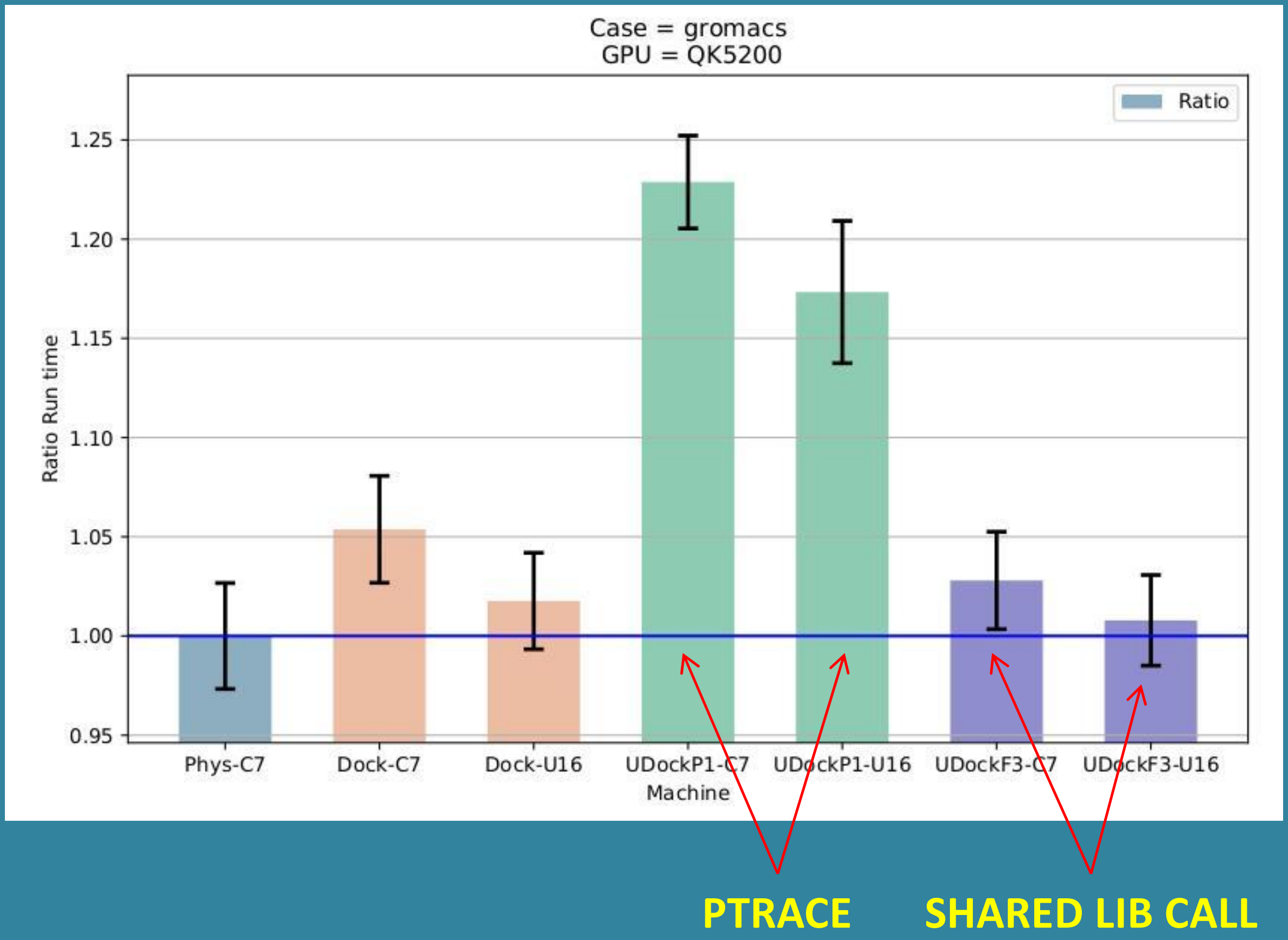
Performance with docker and udocker are the same and very similar to the host.

Using OpenCL and NVIDIA GPGPUs

udocker in P1 mode

Better performance with Ubuntu 16 container

Molecular dynamics



Gromacs is widely used both in biochemical and non-biochemical systems.

udocker P mode have lower performance
udocker F mode same as Docker.

Using OpenCL and OpenMP

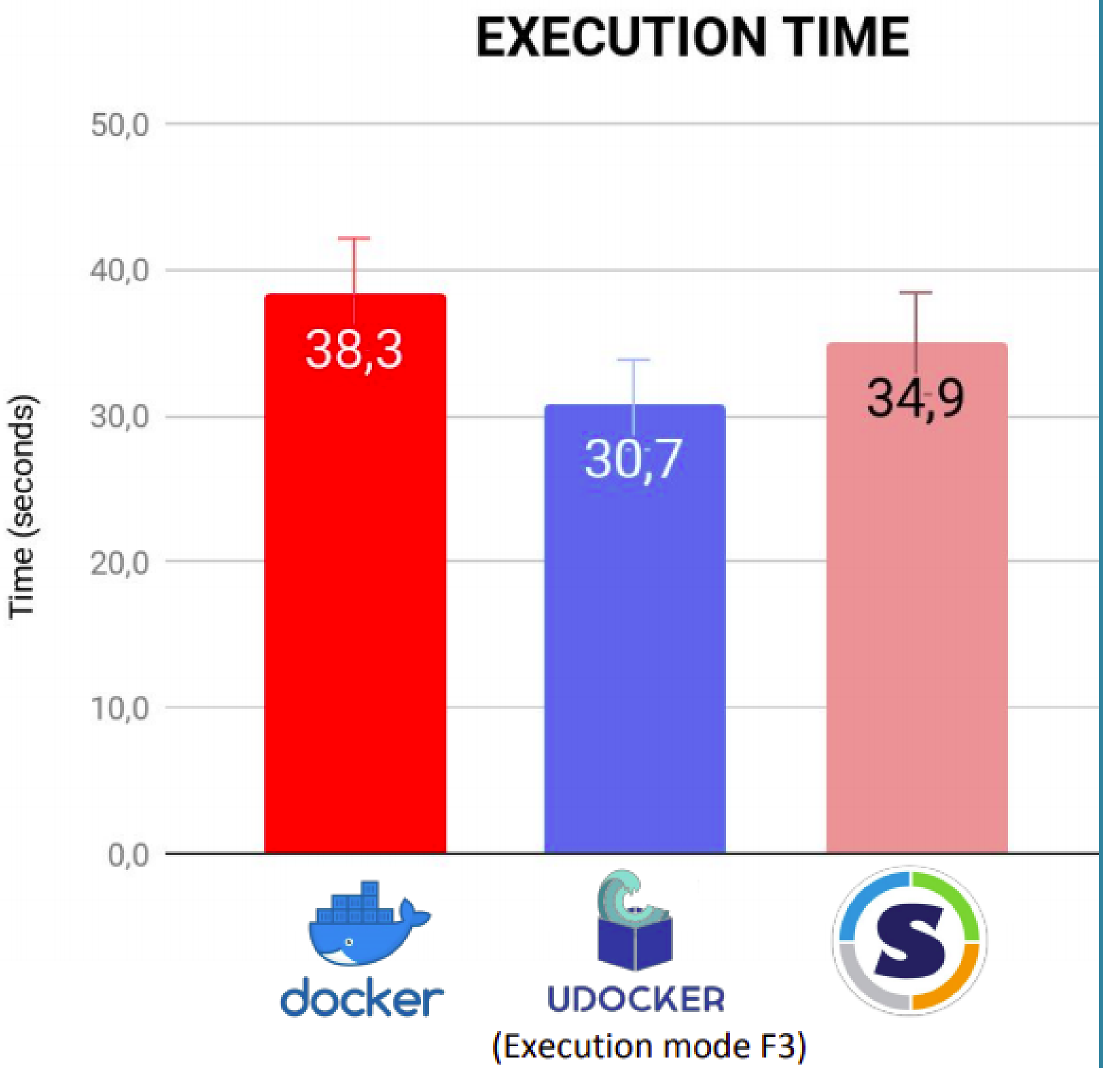
udocker in P1 mode
udocker in F3 mode

TensorFlow

Container:

- Latest GPU version of Tensorflow (from Docker Hub).
- Train a model to recognize handwritten digits (the MNIST data set).

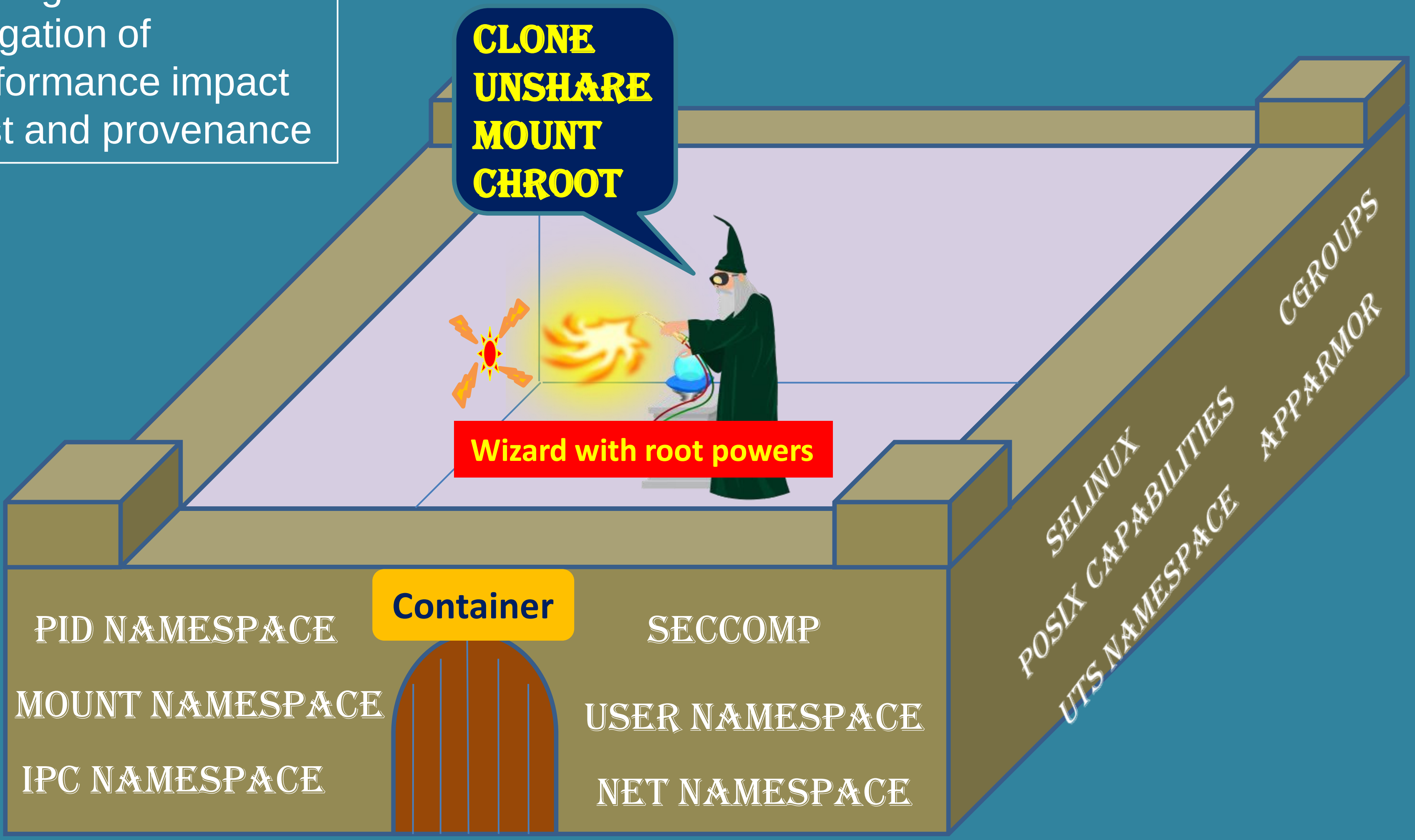
https://github.com/tensorflow/models.git



Challenges

Security in containers

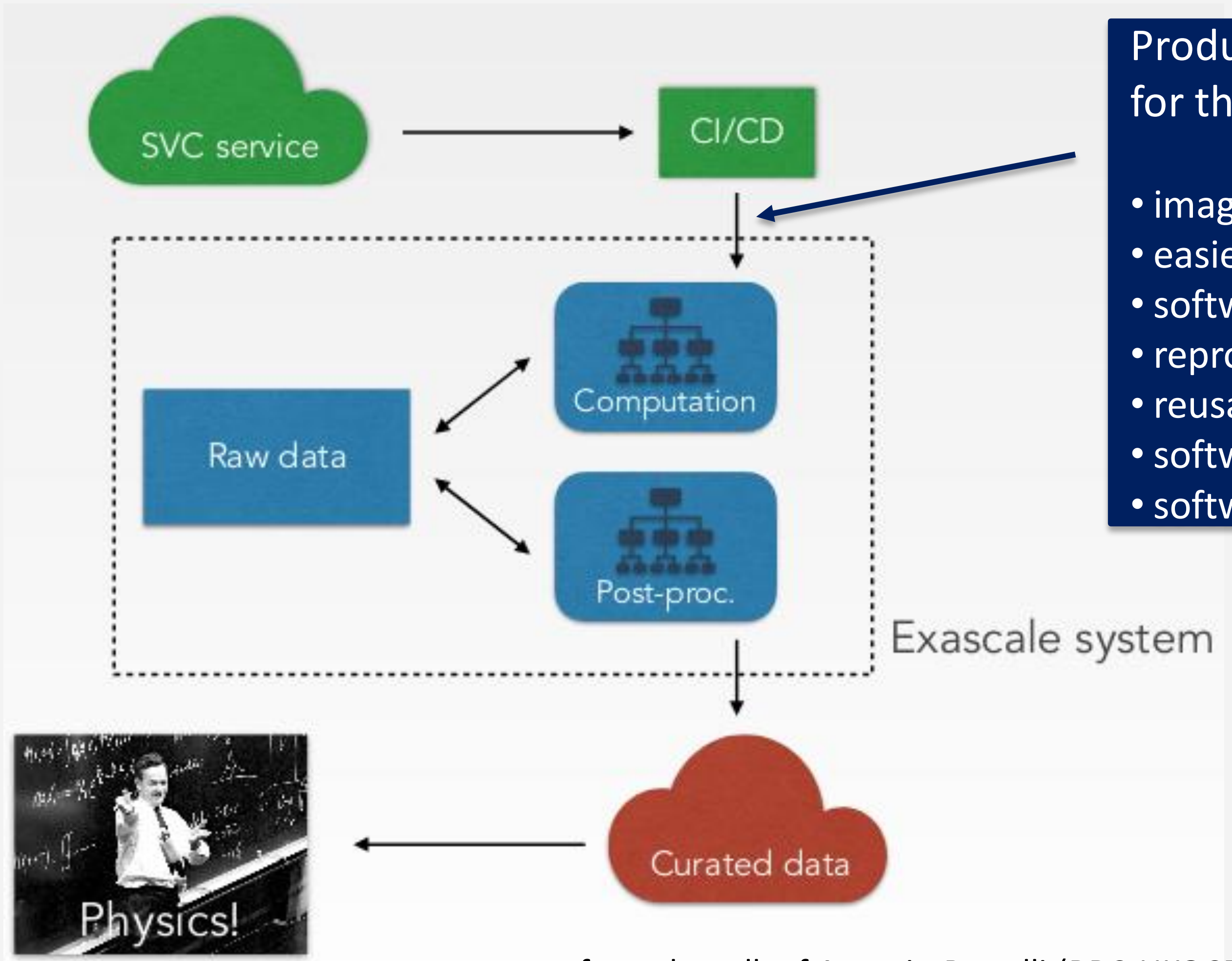
- issues related to privilege escalation
- mitigation of performance impact
- trust and provenance



Other challenging aspects

- Simplify usage of the software and hardware environment
  - Access and interoperate - host drivers, MPI , tight integration
- Productization of software
  - Automate production of application containers for the targets (CI/CD)
- Scalability
  - Processing, communications and I/O – benchmarking and optimization
- Sharing of large machines by heterogeneous workloads
  - Resource usage control - Quality of Service
- Heterogeneous hardware
  - Running in different architectures X86_64, ARM, RISC-V
- Going beyond conventional batch systems
  - Mesos, Kubernetes – containers as the execution unit
- Standardization
  - Creating, accessing and running - OCI

Productization with DevOps



from the talk of Antonin Portelli (RBC-UKQCD)

Produce container images for the targeted machines

- images prepared for targets
- easier sharing of binaries
- software provenance
- reproducibility
- reusability
- software preservation
- software quality assurance

Thank you !

Advantages: Containers vs Virtualization

- Low memory consumption
  - No need of duplicated kernels and OS related processes
  - No duplication of buffering and memory from multiple kernels
  - Less memory split across execution domains

- Very close to native performance
  - Direct execution on top of the host kernel
  - No emulation, No hypercalls, No buffer copies

- Don’t need to run OS services in each isolated environment
  - No need of duplicated NTP, SNMP, CRON, DHCP, SYSLOG, SMART, etc

- Much faster start–up times
  - No OS boot, smaller images to transfer and store

- Less effort
  - Most management effort shifted to the host system

UDOCKER

LXD

runC

Docker Compose

docker swarm

buildah

skopeo

Charliecloud

containerd

katacontainers

podman

cri-o

Istio

MESOSPHERE

Rocket

umoci

Kubernetes

RANCHER

HELM

portainer.io

CloudSlang

Apache MESOS

Nutanix Karbon

MARATHON

SUPERGIANT

docker

Execution methods

- udocker is an integration tool:
  - Supports several techniques and engines to execute containers
  - They are selected per container id via execution modes

Mode	Base	Description
P1	PRoot	PTRACE accelerated (with SECCOMP filtering) 🡐 DEFAULT
P2	PRoot	PTRACE non-accelerated (without SECCOMP filtering)
R1	runC	rootless unprivileged using user namespaces
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
F4	Fakechroot	modified loader and ELF headers dynamically changed
S1	Singularity	where locally installed using chroot or user namespaces

Container benefits