Integration of HPC resources and techniques

New Challenges in Data Science: Big Data and Deep Learning on Data Clouds

Univ. Internacional Menéndez Pelayo,
18th – 21st June 2018, Santander

Jorge Gomes <jorge@lip.pt>

Cofinanciado por:

Overview

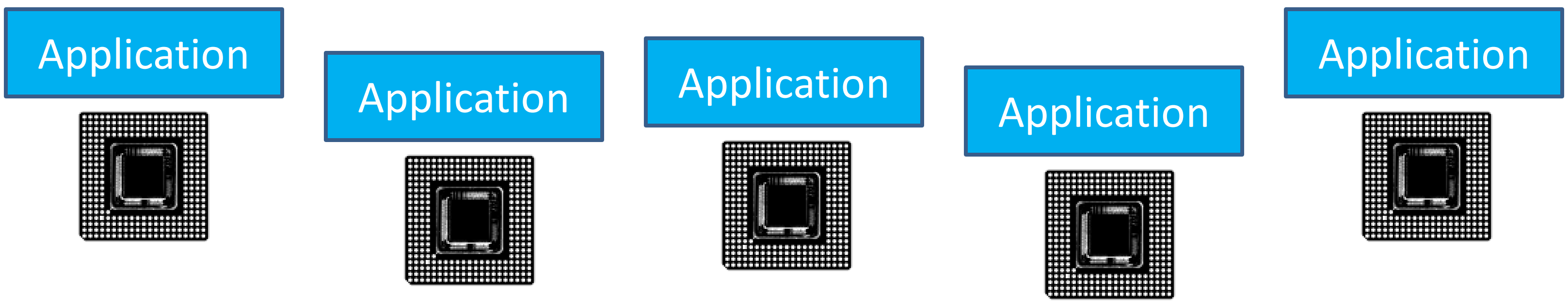
- ❖ **What is High Performance Computing**
- ❖ **How a HPC cluster looks like**
- ❖ **HPC in the Cloud**
- ❖ **Virtualization and containers in HPC**

High Performance Computing

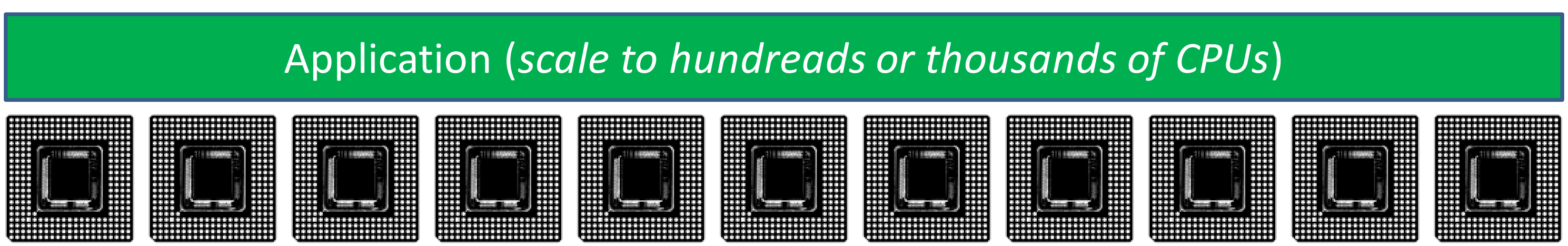
High Performance Computing generally refers to the practice of ***aggregating computing power*** *in a way that delivers much higher performance than one could get out of a typical desk computer in order* ***to solve large computational problems****.*

HTC and HPC

- ***High Throughput Computing (HTC)***
  - Efficient execution of a large number of loosely-coupled tasks (many fully independent jobs ex. 1000 jobs of 1 CPU)

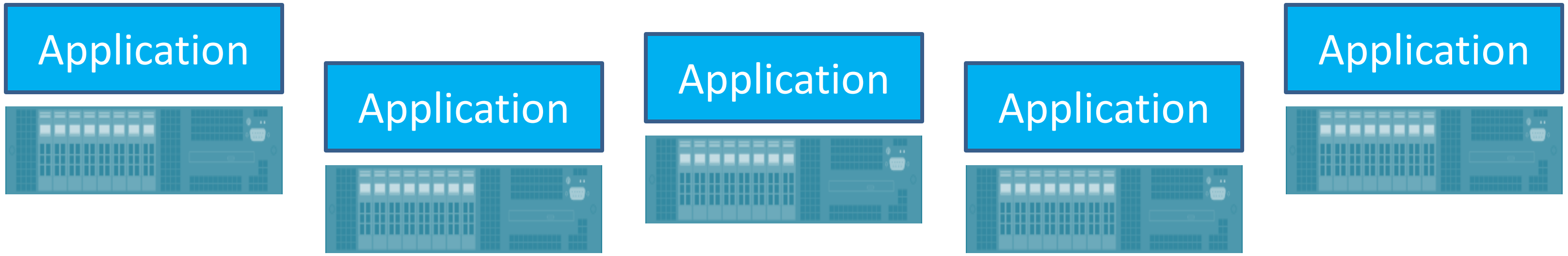


- ***High Performance Computing (HPC) → parallel processing***
  - Getting the maximum performance for a single tightly coupled task running in parallel across many CPUs (ex. 1 job of 1000 CPUs)

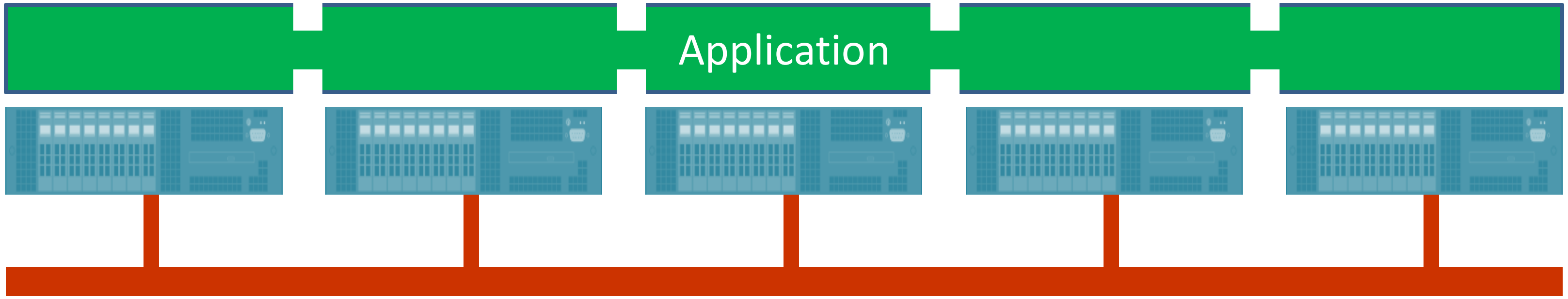


HTC and HPC

- ***High Throughput Computing (HTC)***
    - Many independent computers, no communication between application instances.



- ***High Performance Computing (HPC) ➔ parallel processing***
    - Very large computers or smaller but interconnect by very fast networks.
    - Instances of the application need to communicate between each other.



Communicate across many processes

- Shared data storage
  - write/read to/from a shared file (can be slow)
  - Shared storage
- Shared memory
  - May require a very large multi processor machine
  - Very specific expensive systems
  - Limited scalability
- Local Area Network
  - Conventional machines
  - Messages across the network
  - Easier/cheaper join multiple machines
- Low latency Network
  - Uses a very fast network interconnect (Infiniband, Omni-Path, etc)
  - High scalability

HPC clusters

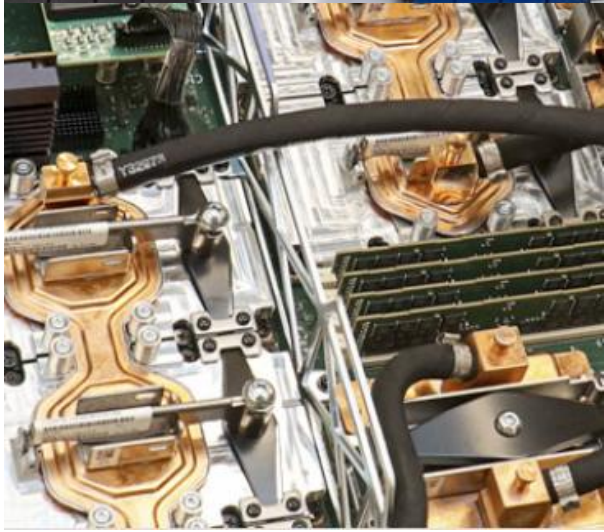
Top500 November 2017

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, **Sunway** , NRCPC National Supercomputing Center in Wuxi - China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, **TH Express-2**, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou – China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, **Aries interconnect** , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) – Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, **Infiniband EDR,** PEZY-SC2 700Mhz , ExaScaler Japan Agency for Marine-Earth Science and Technology – Japan	19,860,000	19,135.8	28,192.0	1,350
5	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray **Gemini interconnect**, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory - United States	560,640	17,590.0	27,112.5	8,209
6	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL - United States	1,572,864	17,173.2	20,132.7	7,890
7	**Trinity** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, **Aries interconnect** , Cray Inc. DOE/NNSA/LANL/SNL - United States	979,968	14,137.3	43,902.6	3,844
8	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, **Aries interconnect** , Cray Inc. DOE/SC/LBNL/NERSC - United States	622,336	14,014.7	27,880.7	3,939
9	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel **Omni-Path** , Fujitsu Joint Center for Advanced High Performance Computing – Japan	556,104	13,554.6	24,913.5	2,719
10	K computer, SPARC64 VIIIfx 2.0GHz, **Tofu interconnect** , Fujitsu RIKEN Advanced Institute for Computational Science (AICS) - Japan	705,024	10,510.0	11,280.4	12,660

Summit at Oak Ridge

Hybrid DataCloud

- 200-petaflop just became operational will put US back in #1
- 4,608 nodes each with:
  - 2x Power9 CPUs
  - 6x NVIDIA Tesla V100 GPUs
- Mellanox dual-rail EDR InfiniBand network
  - 200Gbps for each node
- GPUs alone will provide:
  - 215 peak petaflops at double precision
  - 125 teraflops of mixed precision
  - 3.3 exaflops of Tensor Core for deep learning
  - 1.88 exaflops using the Tensor Core capability already demonstrated



Source Top500

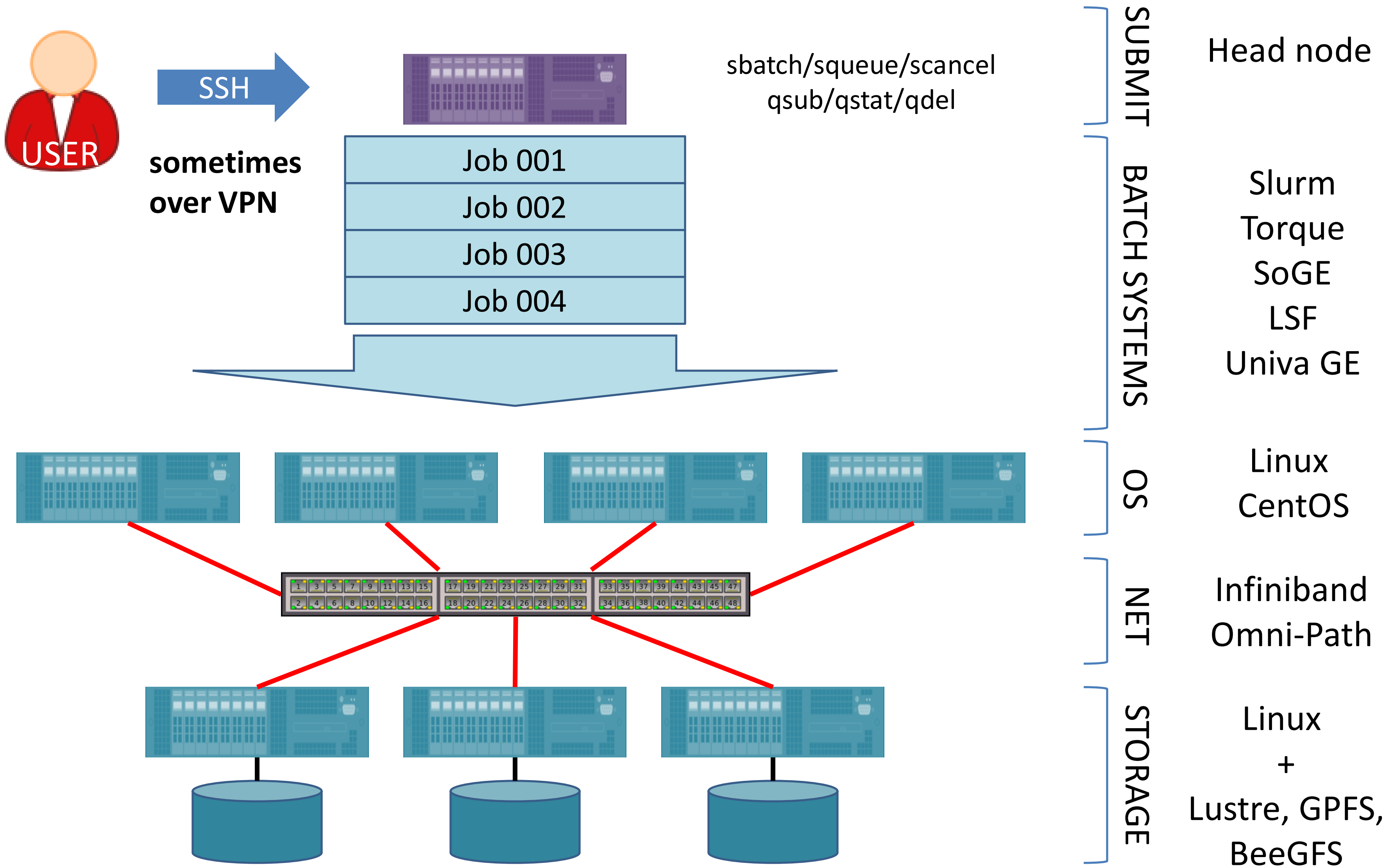
Altamira @ IFCA in Santander

- HPC cluster
  - 158 main compute nodes (2528 CPU cores)
  - 5x GPU compute nodes (2x GPU cards per node)
  - login server and several service servers
- Main compute nodes
  - 2x Intel Sandybridge E5-2670 CPUs 8 cores 2.6 GHz
  - 64 GB of RAM memory (i.e. 4 GB/core)
  - 500 GB local disk
  - Scientific Linux (currently 6.2 version)
- The internal network in Altamira includes:
  - Infiniband Network (FDR)
  - used by parallel applications and data transfer
- Shared storage system
  - GPFS (Global Parallel File System - GPFS)
  - 2 PB capacity

HPC Computing Cluster Components

- Set of computing machines
  - Multi-core servers (x86_64, POWER, SPARC, ARM, etc)
  - Machines must be homogeneous
- Interconnected by a network
  - Ethernet or low latency interconnect
- Usually running Linux
  - Uniform installation of Linux (e.g. CentOS 6)
  - Centrally managed with fixed configurations ➔ production
- Having a batch system
  - Job queuing and scheduling
  - Job execution and management
- High performance parallel file system
  - POSIX like filesystem
  - Provides shared data storage
  - Aggregates storage servers for capacity, performance & resiliency

HPC Computing Cluster



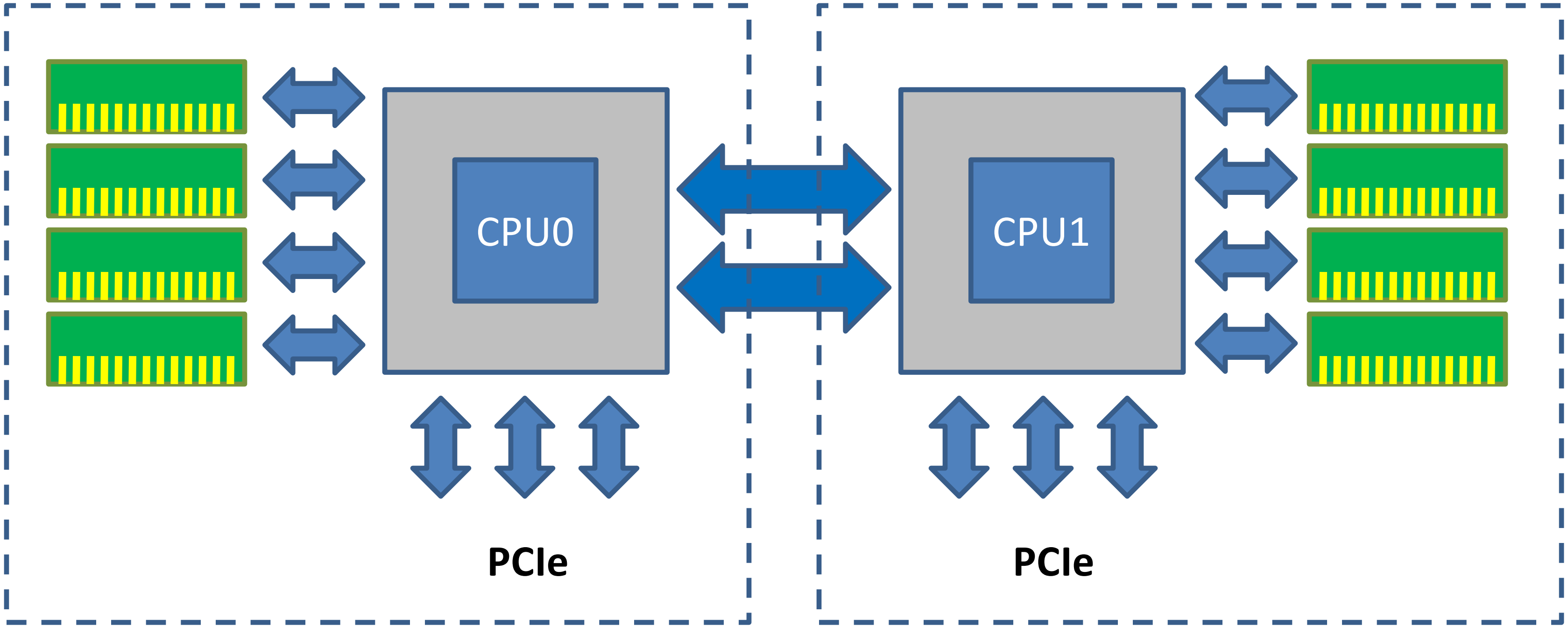
Compute Server

- 2x or more CPU sockets
- Memory
- Local disk storage

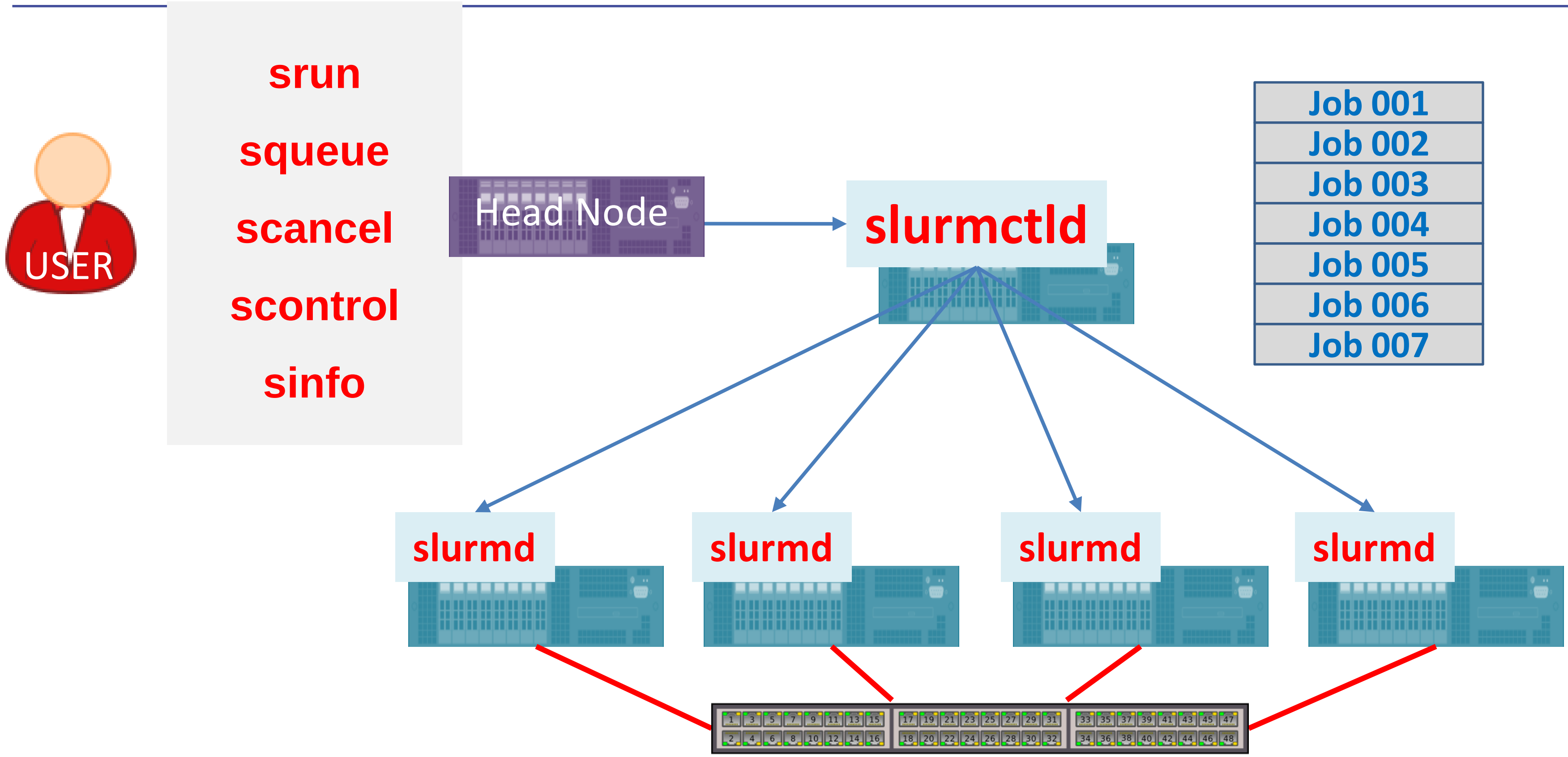
- Accelerators (optional)
- Fast network (low latency)
- Remote management processor
- Redundant PSUs

Non-uniform memory access (NUMA)

- Groups processor with its own memory (NUMA node)
- Any processor can access the whole memory
- Access to local memory faster than access to remote memory



Batch system – example Slurm

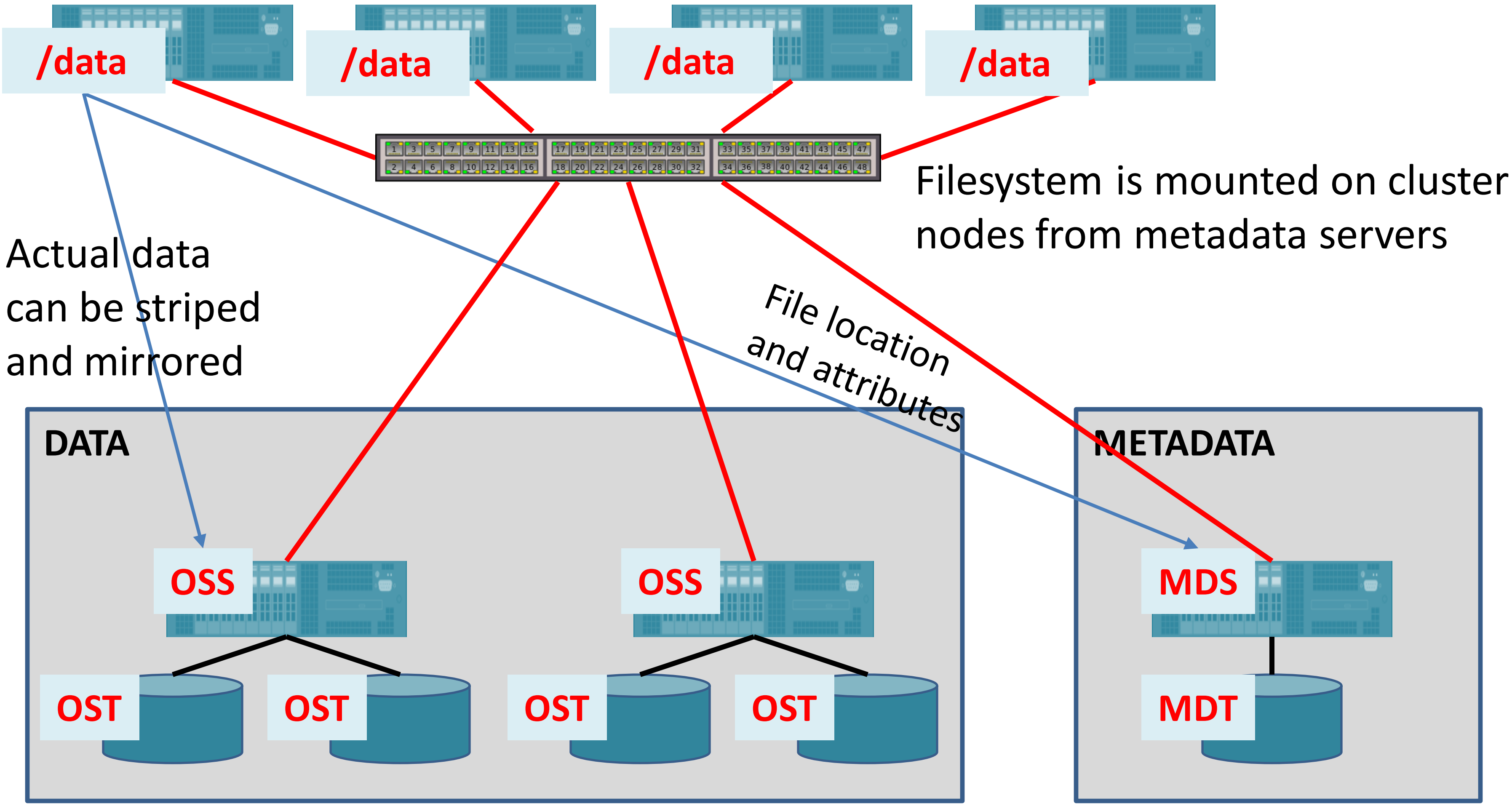


$ srun --ntasks=2 --label /bin/hostname
0: node14
1: node17

-- cpus-per-task=#
--nodes=#

https://slurm.schedmd.com/

Parallel Filesystem – Lustre example



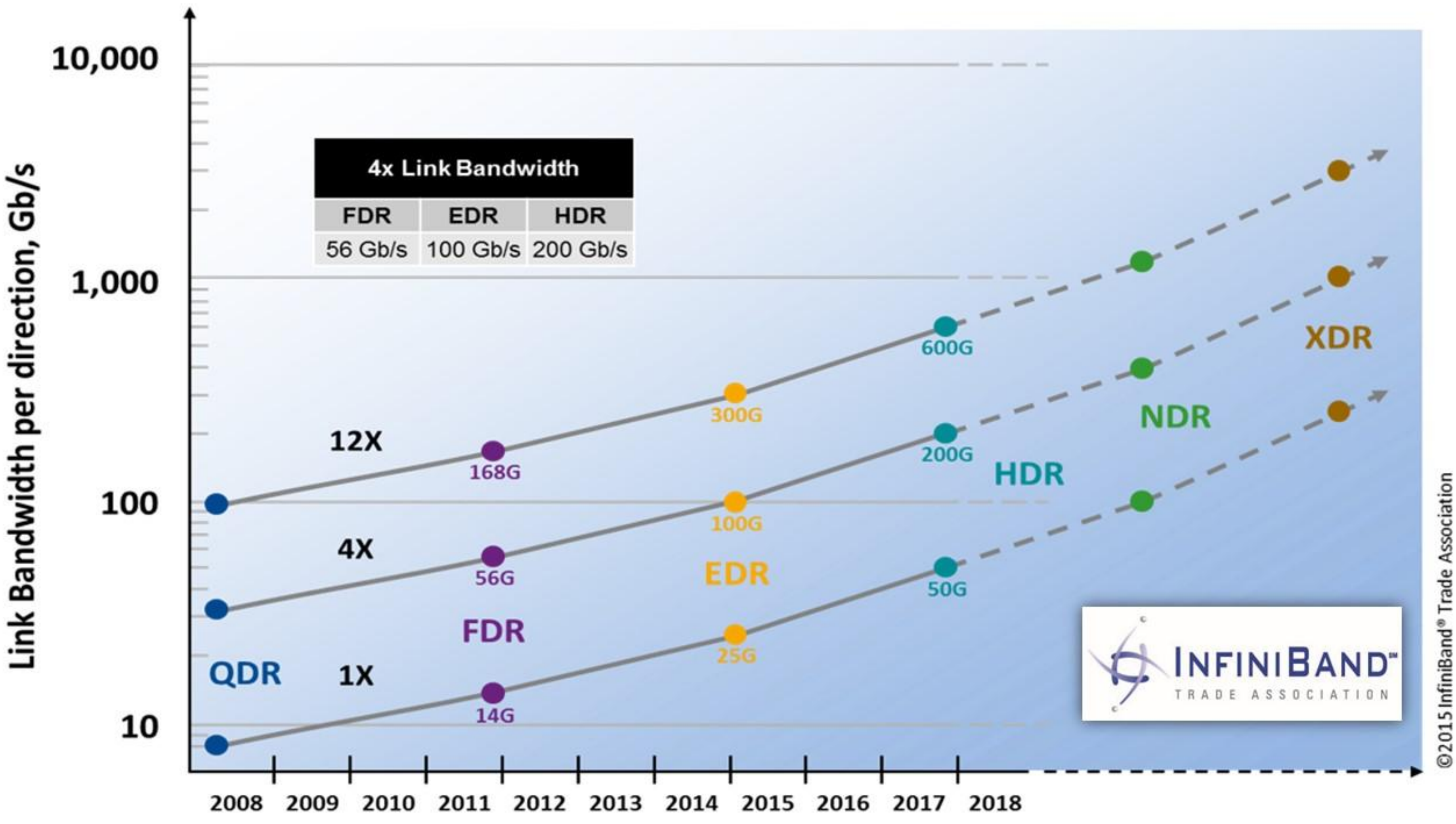
Low latency interconnects

- Very low communication delays
- High bandwidth
- Low communication overhead

- Can be used by:
  - Parallel applications (exchange of application data)
  - Access to data storage (e.g Lustre filesystem)

- They are Important to:
  - Minimize communication and I/O wait time
  - Maximize CPU compute capacity and utilization

Infiniband evolution

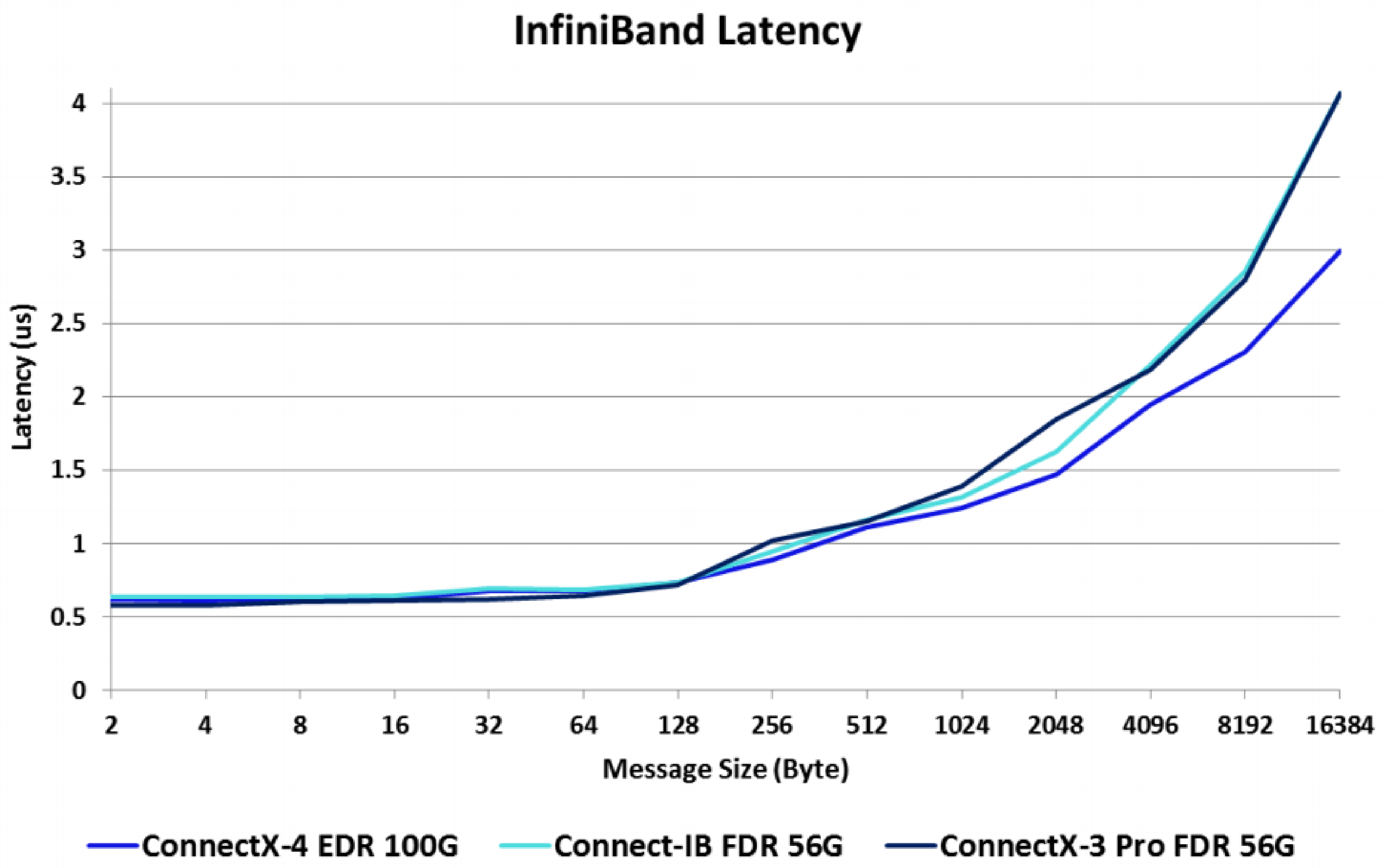


Infiniband evolution

	SDR	DDR	QDR	FDR10	FDR	EDR	HDR	NDR	XDR
Theoretical effective throughput, Gbs, per 1x link	2	4	8	10	13.64	25	50	100	250
Speeds for 4x links (Gbit/s)	8	16	32	40	54.54	100	200	400	1000
Speeds for 8x links (Gbit/s)	16	32	64	80	109.08	200	400	800	2000
Speeds for 12x links (Gbit/s)	24	48	96	120	163.64	300	600	1200	3000
Adapter latency (microseconds)	**5**	**2.5**	**1.3**	**0.7**	**0.7**	**0.5**	**less?**	**?**	**?**
Year	2001, 2003	2005	2007	2011	2011	2014	2017	after 2020	future (2023?)

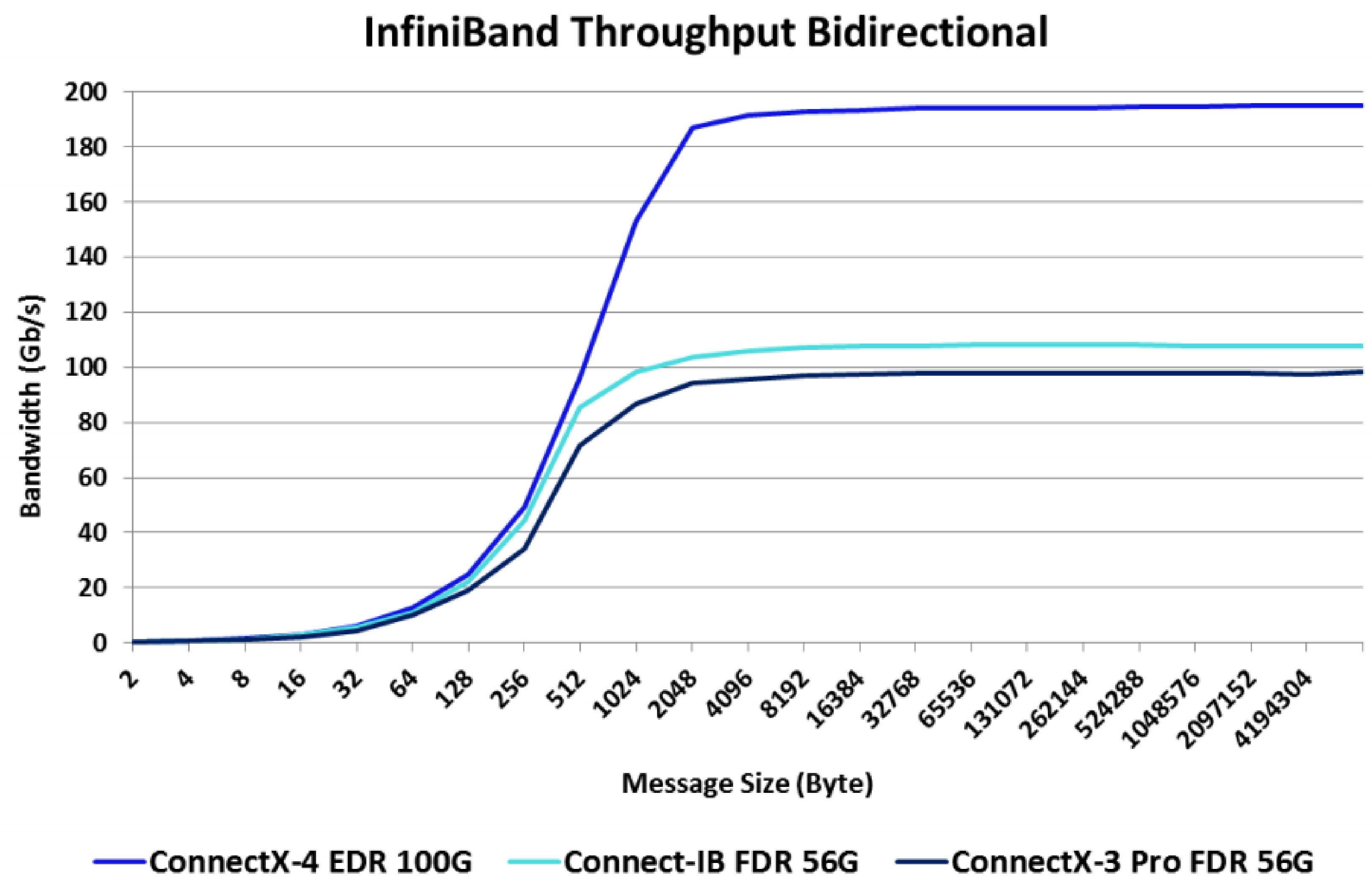
Source Wikipedia

Infiniband evolution



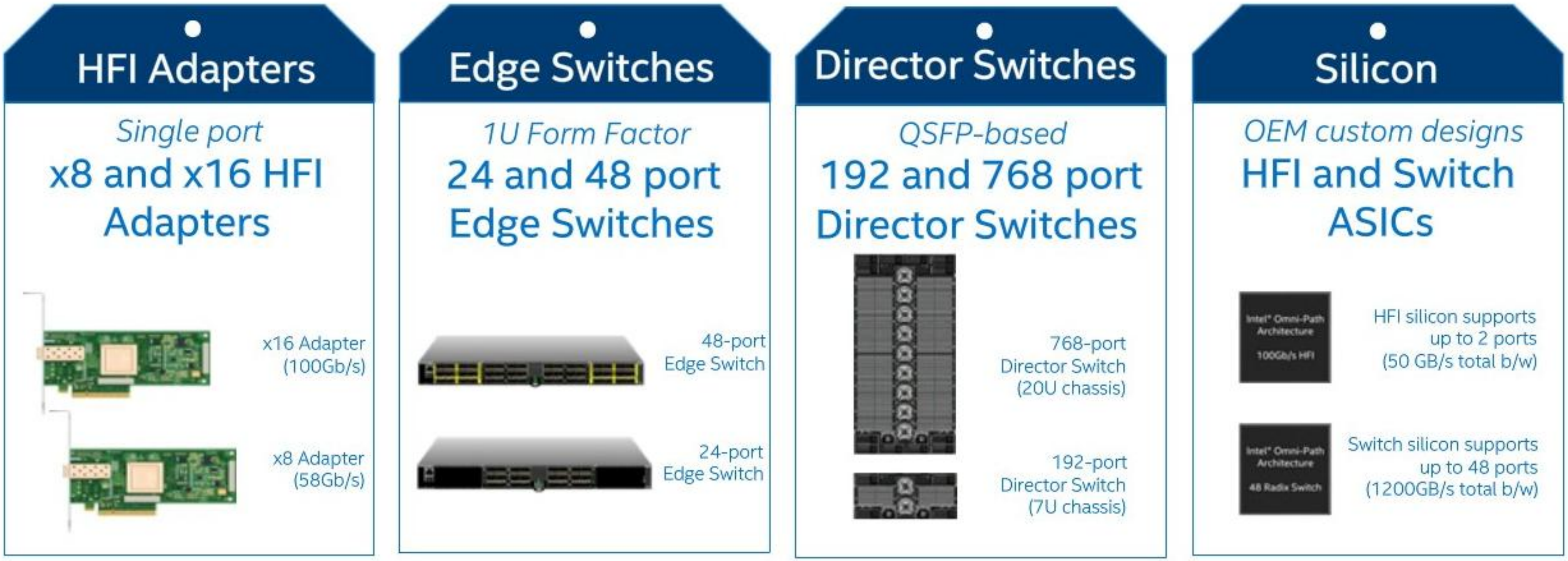
Source Mellanox

Infiniband evolution



Source Mellanox

Omni-Path



Source Intel

Message Passing Interface - MPI

- Message-passing specification for parallel computing
  - Enable communication between processes
  - Bindings for C, C++, and Fortran90, Python, R, ...

- Included in the specification:
  - Point-to-point communication
  - Communication contexts
  - Process topologies
  - The Info object
  - One-sided communication
  - External interfaces
  - Process creation and management
  - Datatypes
  - Collective operations
  - Process groups
  - Parallel file I/O

Some MPI functions

- MPI_INIT(int *argc, char **argv) /* initialize at beginning */
- MPI_FINALIZE() /* terminate processing */

- MPI_COMM_SIZE(comm, size) /* number of processes */
- MPI_COMM_RANK(comm, id) /* get this process id */

- MPI_SEND(buf, count, datatype, dest, tag, comm) /* send msg */
- MPI_RECV(buf, count, datatype, source, tag, comm, status) /* receive */

- comm: MPI_COMM_WORLD
- tag: MPI_ANY_TAG or message tag (int to identify the message content)
- source: MPI_ANY_SOURCE or process id (rank)
- dest: other id (rank)
- Type: MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG, MPI_UNSIGNED_CHAR, MPI_UNSIGNED_SHORT, MPI_UNSIGNED, MPI_UNSIGNED_LONG MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE, MPI_BYTE, MPI_PACKED

MPI example

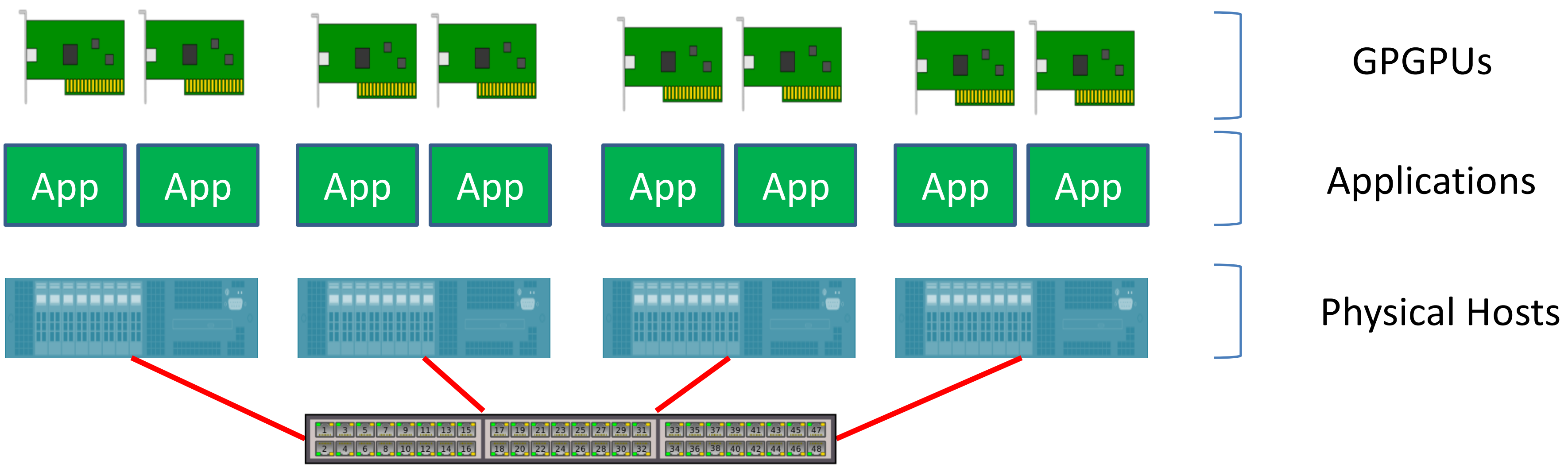
```
#include "mpi.h"

int main( int argc, char *argv[]) {
    char message[20];
    int myrank;
    MPI_Status status;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
    if (myrank == 0)  {     /* code for process zero */
        strcpy(message,"Hello, there");
        MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 99, MPI_COMM_WORLD);
    }  else if (myrank == 1) {     /* code for process one */
        MPI_Recv(message, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status);
        printf("received :%s:\n", message);
    }
    MPI_Finalize();
    return 0;
}
```

destination rank

tag

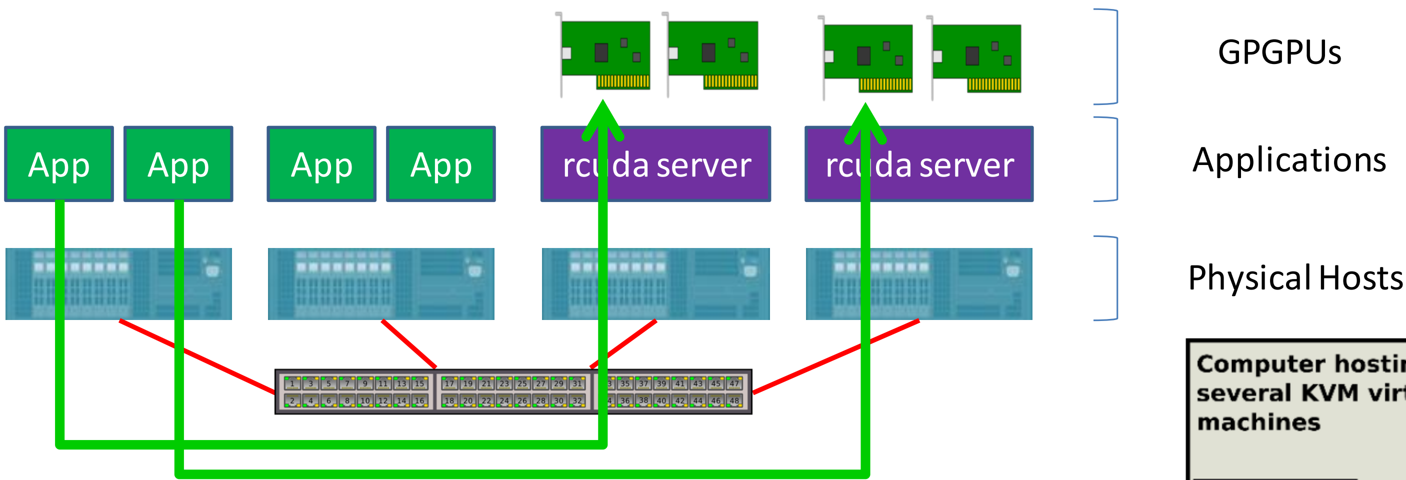
HPC clusters and accelerators

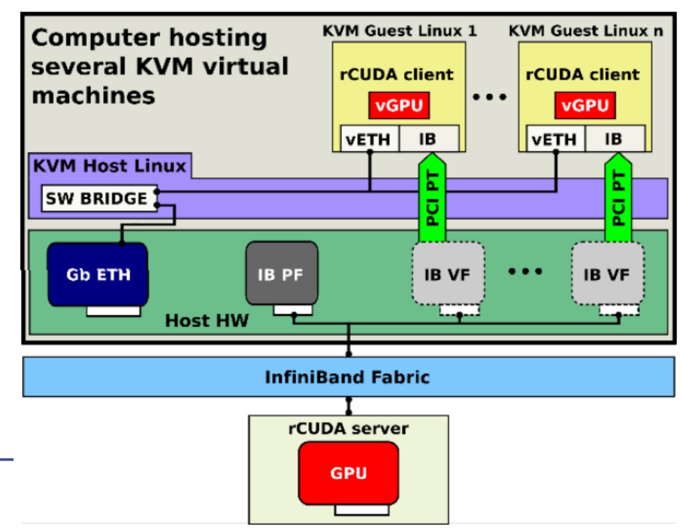
- Can improve processing speed for certain operations
- GPGPUs and Xeon-Phi are ideal for vector processing
- GPGPUs very popular for Machine Learning
- Sharing GPGPU accelerators by applications is non-trivial
  - One GPGPU per application instance



HPC clusters and accelerators

- Remote access to the GPUs with rcuda
- Latency is an issue but in HPC we have low latency networks
- rcuda enables access to remote GPUs and even sharing
- A limited set of GPUs can be made available to cluster nodes



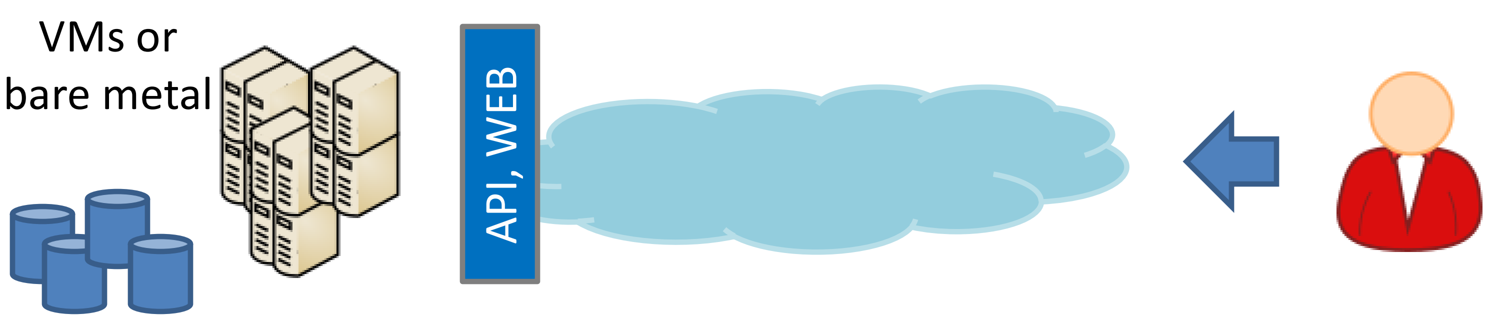


http://www.rcuda.net

HPC in the cloud

Cloud Computing

- ***Cloud Computing – Infrastructure as a Service (IaaS)***
  - access to pools of configurable system resources (and higher-level services) that can be provisioned with minimal management effort, over the network.
  - Example: Amazon (AWS), private OpenStack deployments
    - Instantiate machines on demand
    - Access remote storage
    - Flexibility
    - Elasticity
    - Pay-per-use



HPC in the Cloud

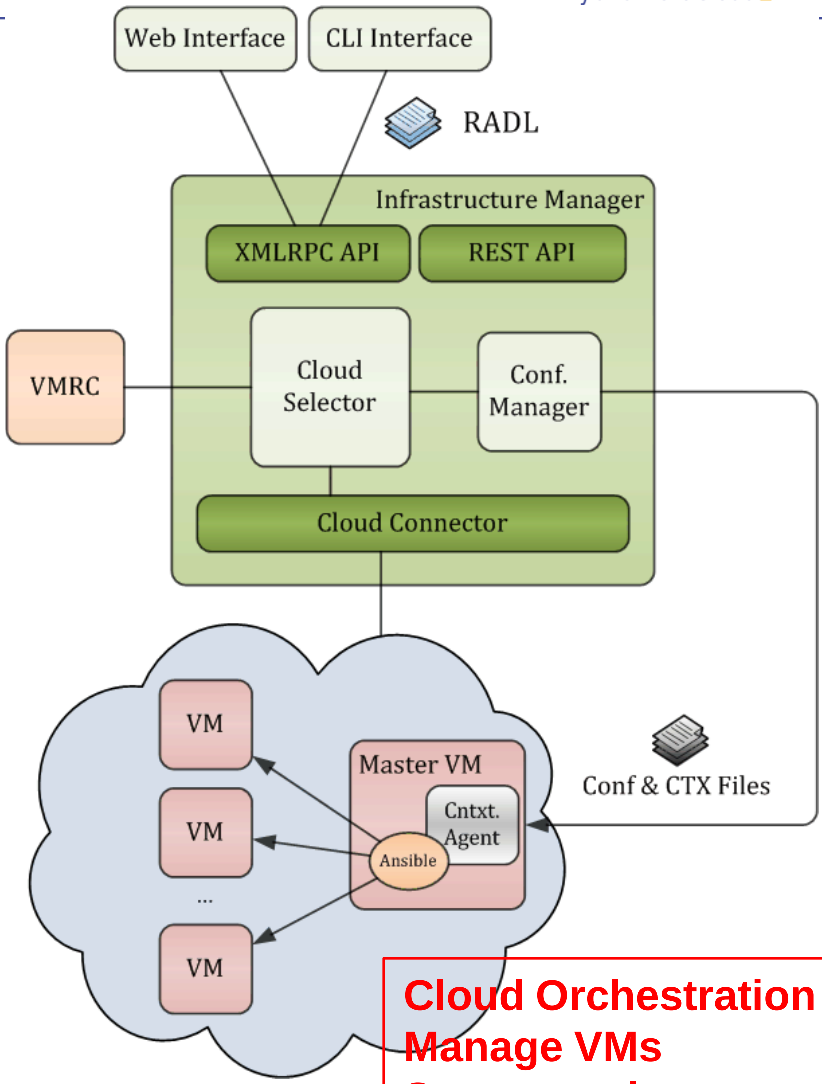
- **Instantiate HPC machines in a cloud infrastructure**
  - Machines can be physical (bare-metal) or virtual
  - Entire cluster
    - ✓ Submission machine (head-node)
    - ✓ Compute nodes (elasticity as you go)
    - ✓ Batch scheduler
    - ✓ Low latency interconnect or fast network
  - Individual machines
    - ✓ Accelerators (GPGPUs etc)
- **Providers**
  - public/commercial
    - Amazon, MS Azure, Google cloud platform, IBM cloud, Oracle cloud, Alibaba cloud, …
  - private/organizational
    - Scientific, academic, and companies

Cloud Computing Orchestration

Machine Learning

Infrastructure Manager

- IM is a Cloud Orchestrator:
  - Supports TOSCA YAML with INDIGO-DataCloud custom types.
    - Input a description of the intended machines and their setup and configuration
    - Deploys the whole set of machines
  - Support a wide range of Cloud back-ends:
    - **OCCI, OpenNebula and OpenStack** (using native APIs).
    - Public providers: **AWS, Azure, GCE**.
    - To perform **hybrid deployments across multiple Cloud sites**.

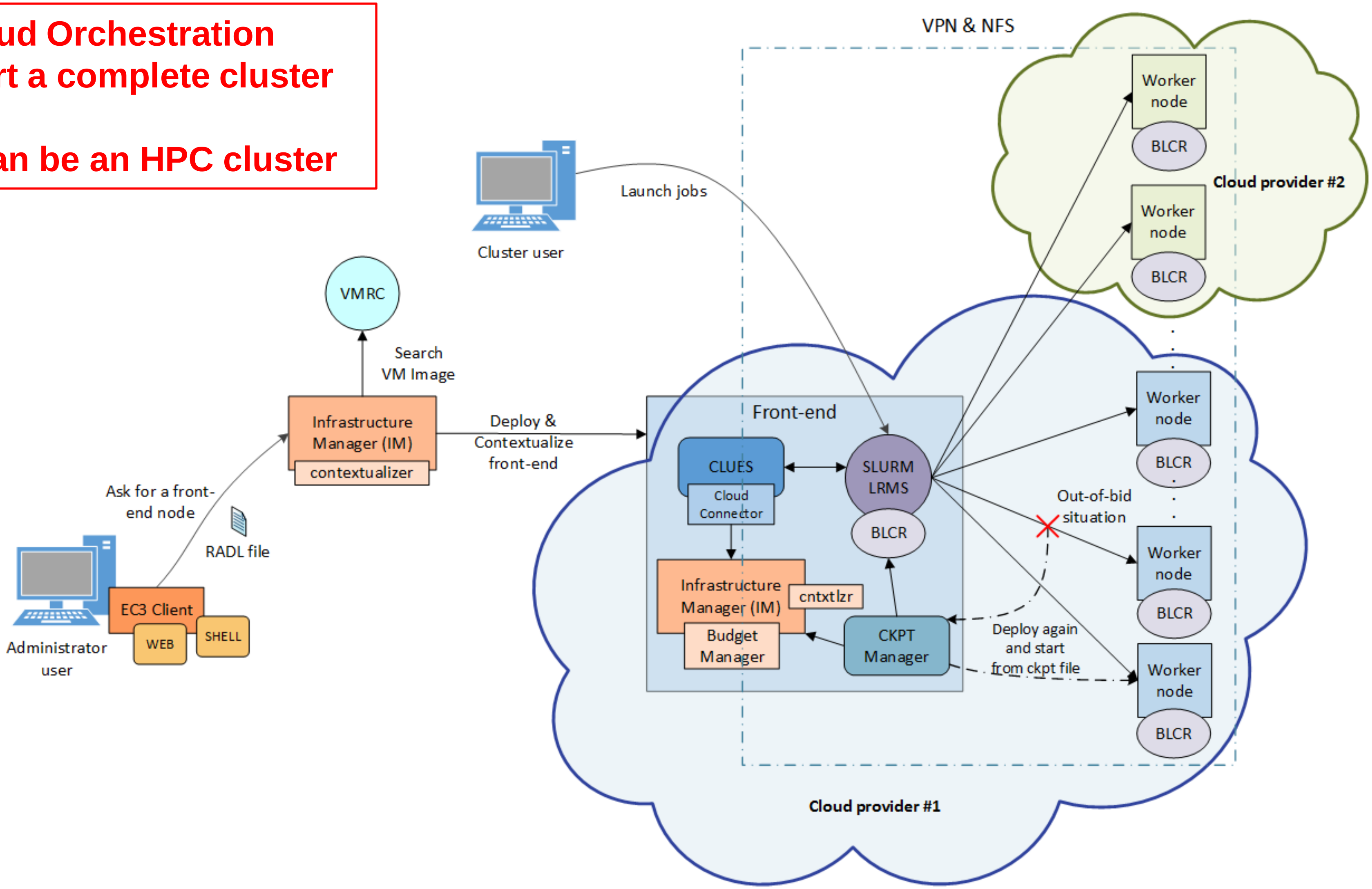


Cloud Orchestration
Manage VMs
Start complex setups

Elastic Cloud Computing Cluster (EC3)

Cloud Orchestration
Start a complete cluster

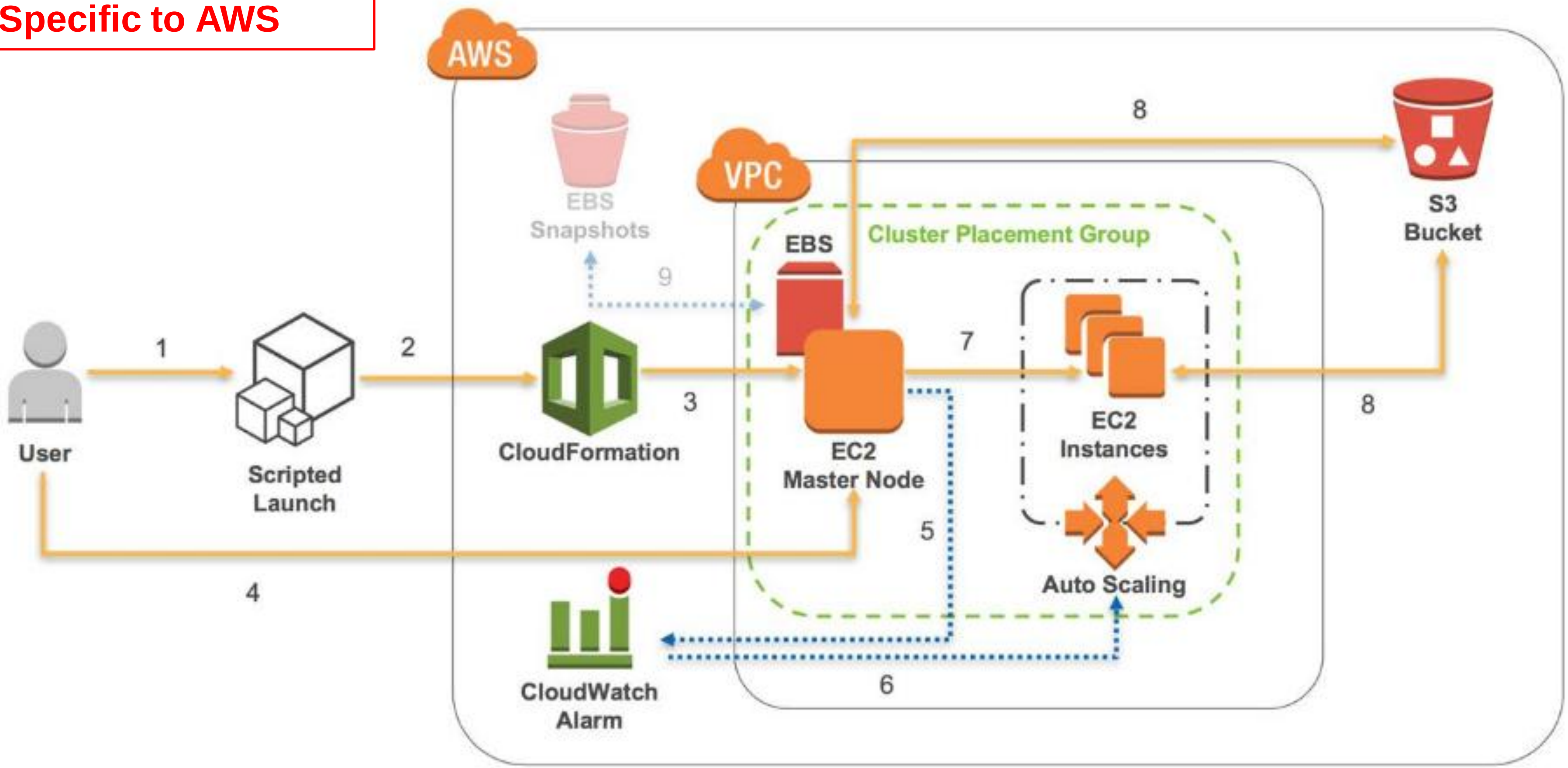
It can be an HPC cluster



http://www.grycap.upv.es/ec3

AWS cluster

Proprietary solution
Specific to AWS



http://www.grycap.upv.es/ec3

GPUs at Google

Google Cloud GPU Type			VM Configuration Options		
NVIDIA GPU	GPU Mem	GPU Hourly Price**	GPUs	vCPUs*	System Memory*
V100	16GB	$2.48 *Standard* $1.24 *Preemptible*	1,8 (2,4) coming in beta	1-96	1-624 GB
P100	16GB	$1.46 *Standard* $0.73 *Preemptible*	1,2,4	1-96	1-624 GB
K80	12GB	$0.45 *Standard* $0.22 *Preemptible*	1,2,4,8	1-64	1-416 GB

Source: NVIDIA

125 ML teraflops

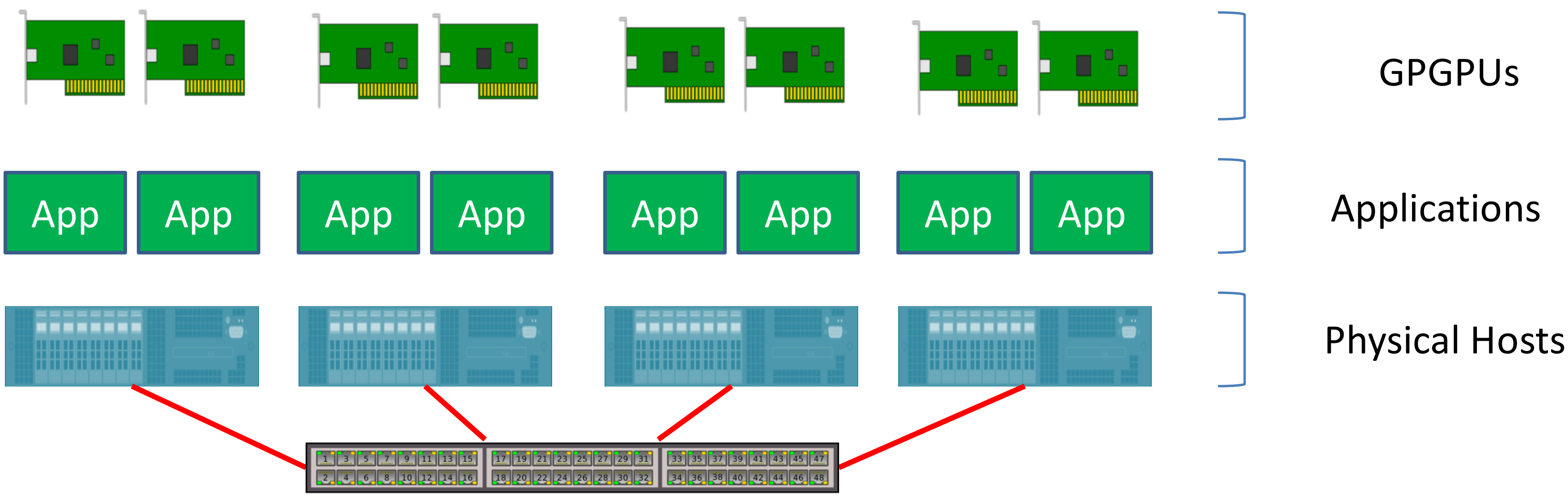
Previously:
Google renting of TPU board $6.50 per hour.
180 ML teraflops and a minimal software stack based on the TensorFlow framework.

HPC in the cloud challenges

- Virtualization performance impact
  - Full virtualization as used by many providers has performance overheads
  - Sharing of host resources by other VMs
  - May require bare metal or dedicated hosts
- Accessing accelerators
  - If virtualization is used some performance may be lost when using multiple GPUs in the same host
- Availability of low latency interconnects
  - Many providers do not provide low latency interconnects
  - Who else is sharing the low latency network
- Accessing low latency interconnects
  - Need to use SR-IOV

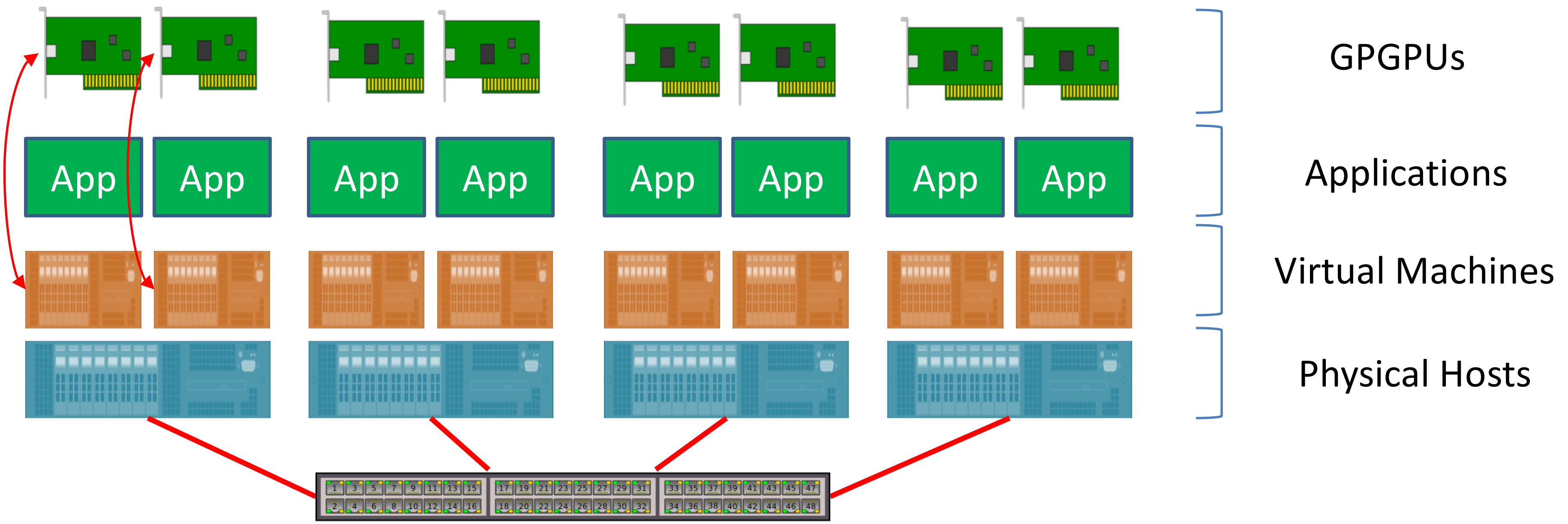
GPGPUs in the Cloud – Bare metal

- Running on hardware without virtualization – Bare metal
- Physical machines are configured and delivered by the cloud management framework
- Similar to the conventional HPC cluster
- The physical machines are installed on demand



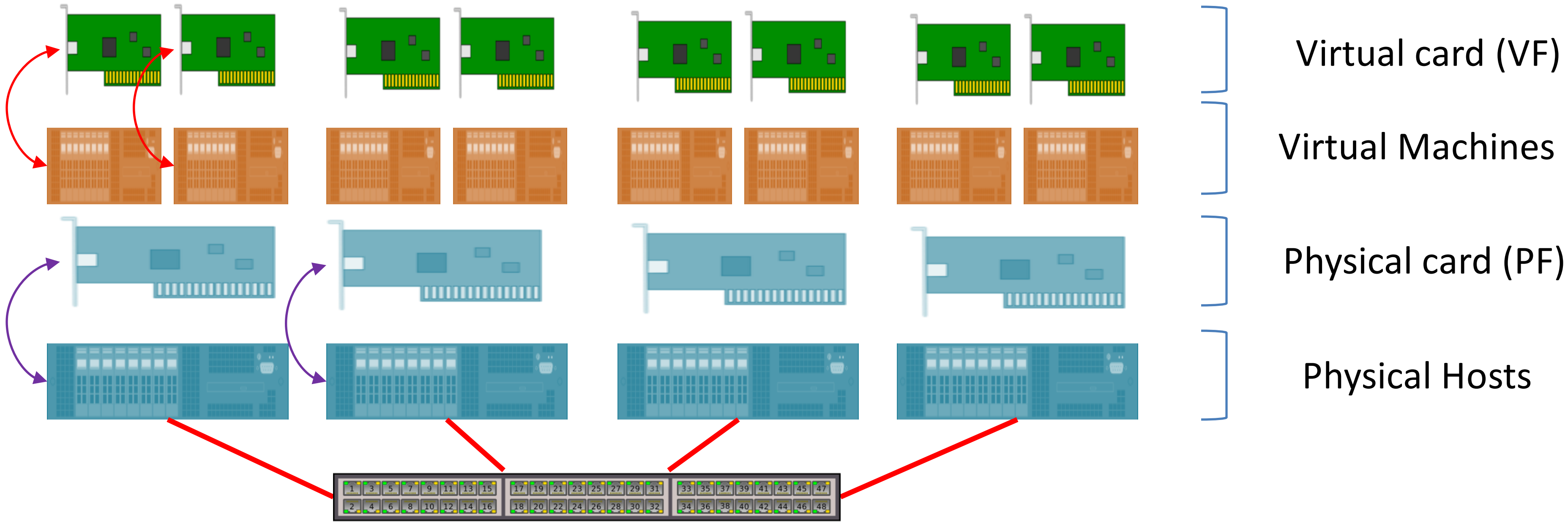
GPGPUs in the Cloud - Virtualization

- Running on virtual machines
- VMs are delivered by the cloud management framework
- Need to pass the GPGPU to the VM
- GPGPU appears as a PCI device in the VM – PCI passthrough



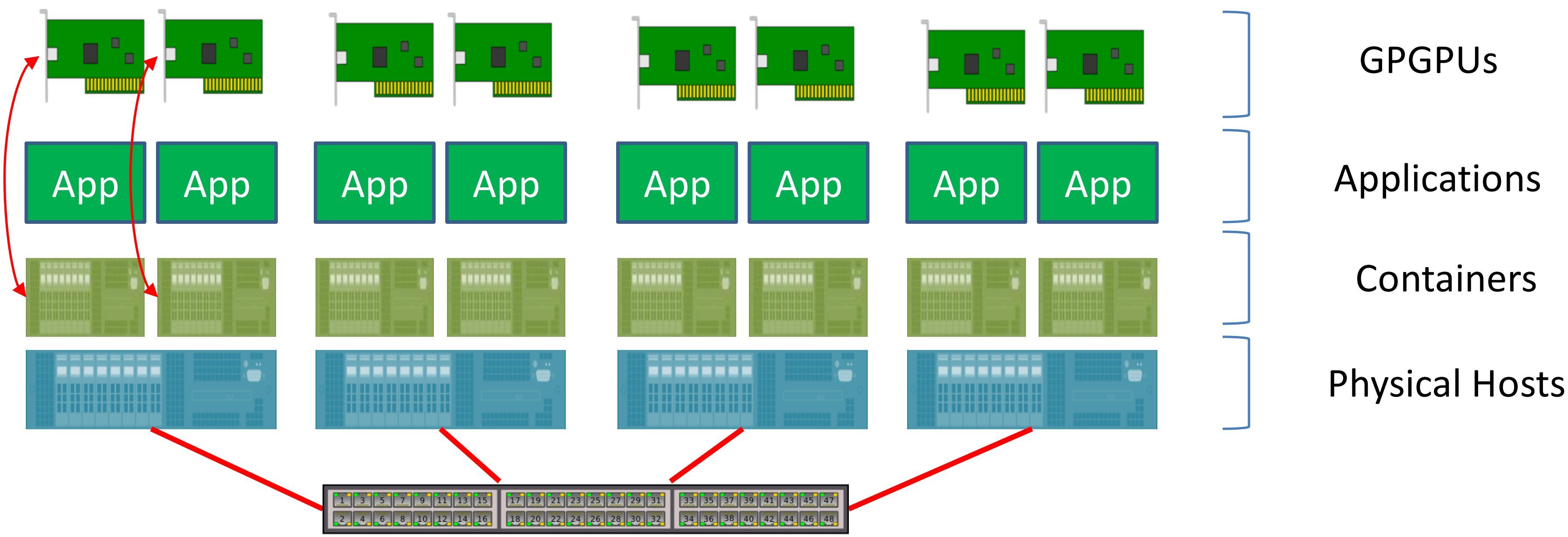
Network in the cloud - Virtualization

- Network virtualization will defeat performance
- Use **SR-IOV** to:
  - Create virtual instances of the PCIe network cards (VF)
  - Map them directly into the VMs
  - Make sure the driver in the VM is the correct one for the VF



HPC in the Cloud - Containers

- Running on containers in the physical machines
- Need to pass the GPGPU Linux device to the container
- GPGPU appears as a device in the container
- Host network can be shared with the container
- Containers are much more lightweight than virtual machines



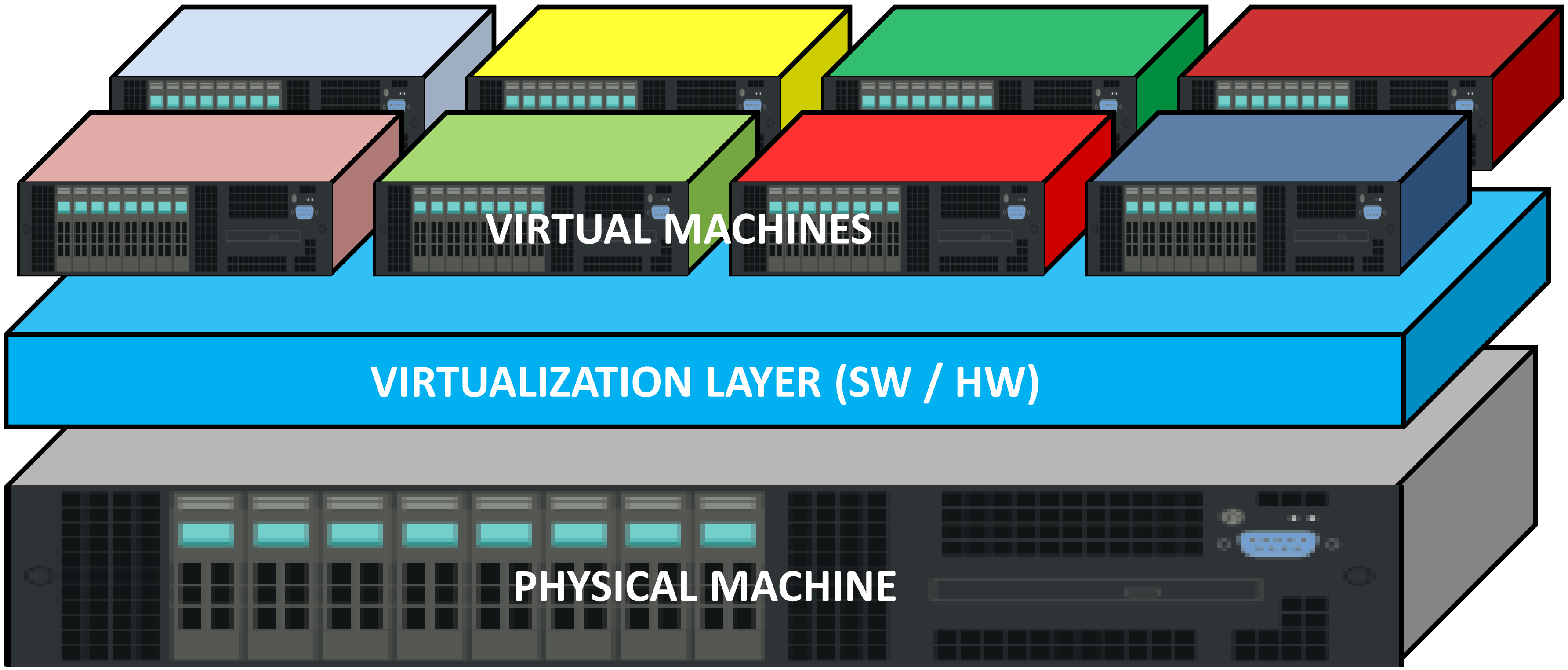
Containers

Applications vs computing resources

	Desktop Fedora 26	Portable Ubuntu 18	HTC cluster SL 6	HPC cluster CentOS 6	Cloud ***
Application 1 + libs + Ubuntu 14					
Application 2 + libs + CentOS 6					
Application 3 + Keras + Ubuntu16					
Application 4 + Theano + CentOS 7					

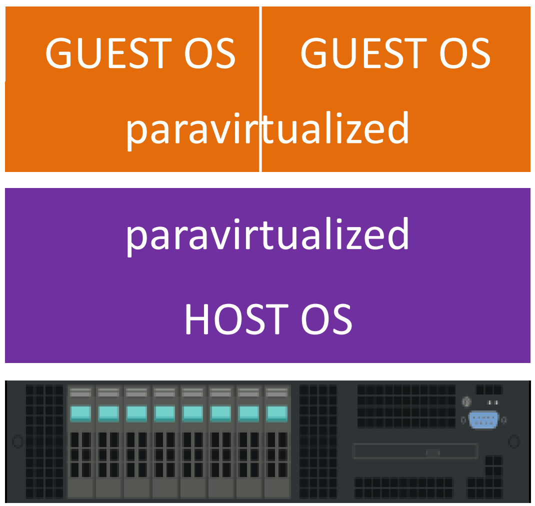
Virtualization

- Valid approach in the cloud
- Not usually available in conventional HPC clusters



Common types of virtualization

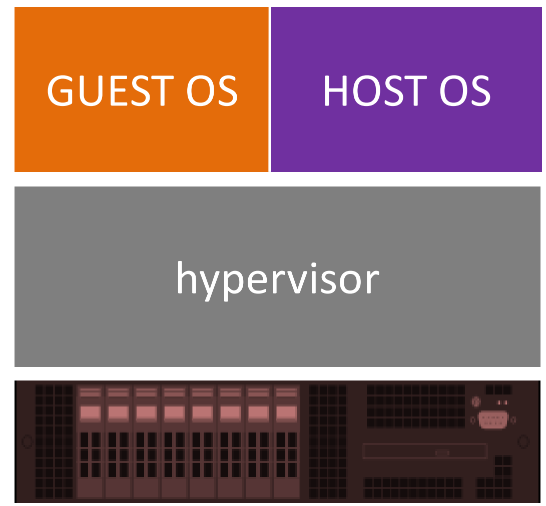
Paravirtualization



- Both kernels changed
- Emulation replaced by hypercalls to the host

- Ex. Xen

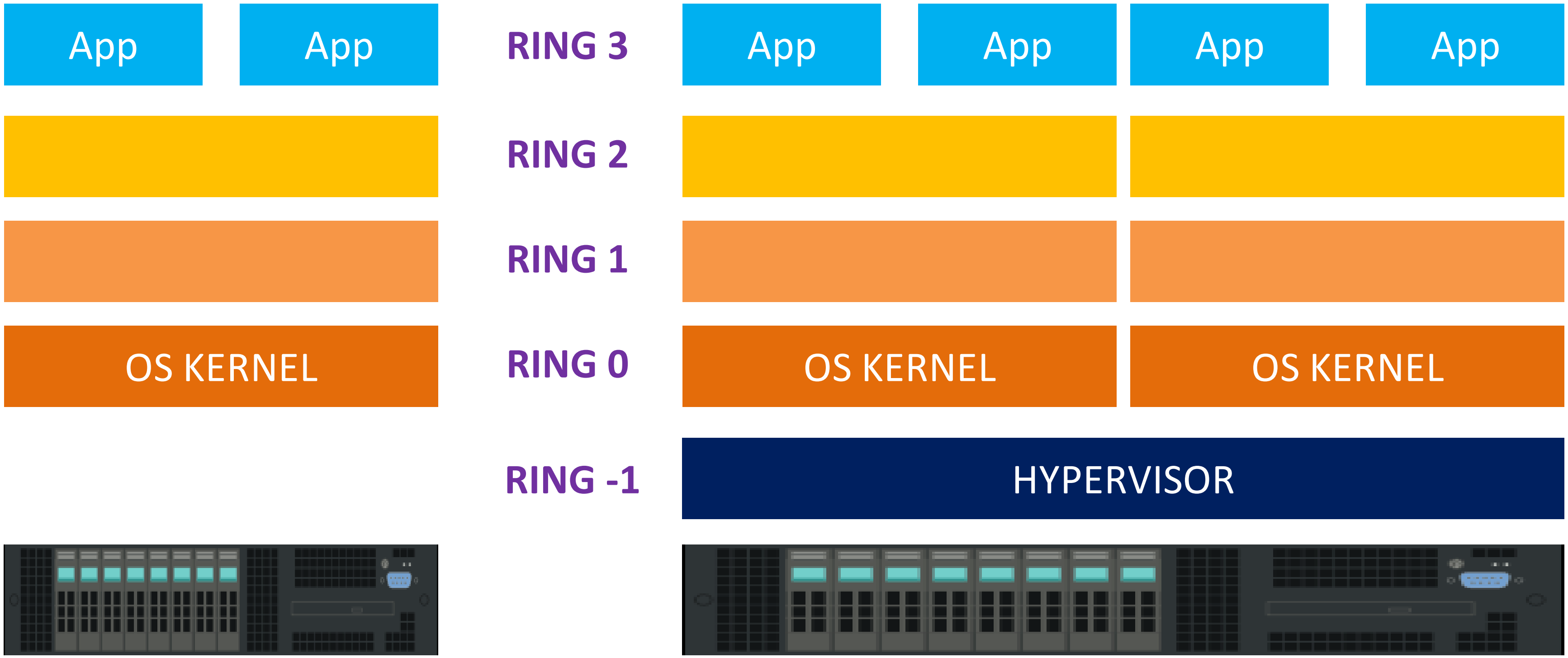
Hardware assisted virtualization



- Both kernels unchanged
- Hardware assisted requires CPU support

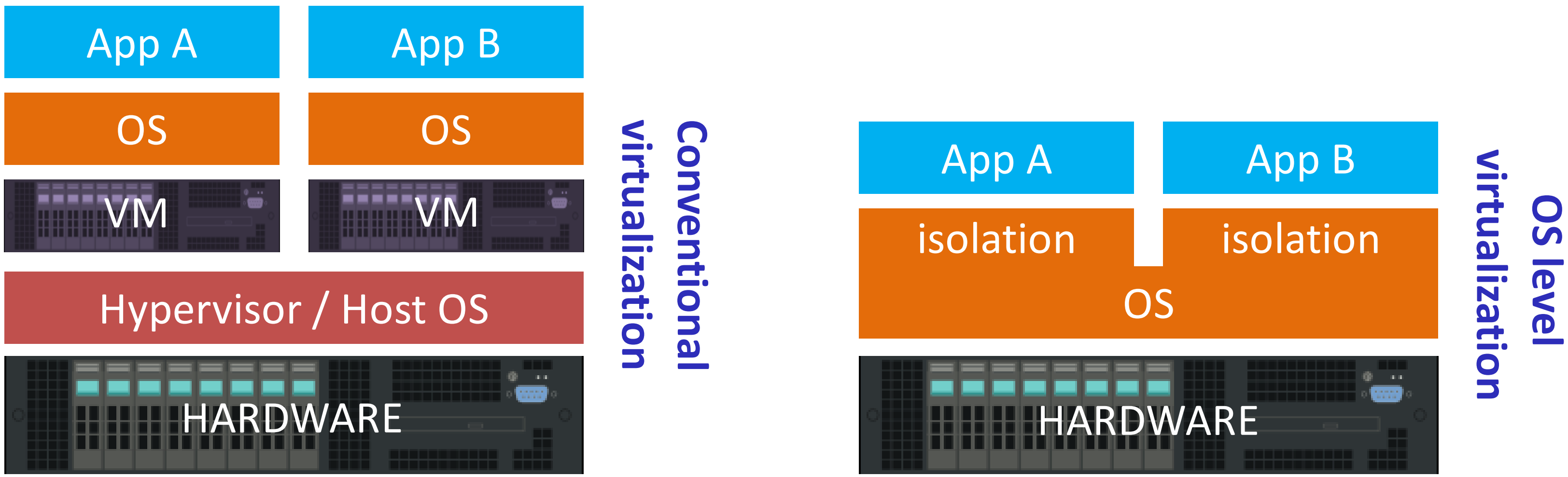
- Ex. KVM

Rings and hardware virtualization



- Rings are hierarchical protection domains within the CPU
- Lower rings have higher privileges in the processor
- Intel VT-x and AMD-V add a ring -1 for hypervisors

Containers (OS level virtualization)



- Multiple environments via OS isolation features
- Isolation limits what processes can do and see
- Same OS kernel is shared and directly used by all containers
- More efficient than VMs (avoids virtualization and guest OS)

OS level virtualization advantages

- Less memory consumption
  - No need of duplicated kernels and related processes
  - No duplication of buffering and shared memory
  - Less memory split across execution domains
- Faster I/O and execution and less latency
  - Direct execution on top of one single 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 management effort
  - Only the host machine needs to be managed (many-core is great)

OS level virtualization also not new

		Year	File system isolation	I/O limits	Memory limits	CPU quotas	Network isolation	Root priv isolation
chroot	Most unix systems	1982	X					
Jail	FreeBSD	1998	X	X	X	X	X	X
Linux-VServer	Linux	2001	X	X	X	X	X	X
Virtuozzo Containers	Linux Windows	2001	X	X	X	X	X	X
Zones	Solaris	2004	X	X	X	X	X	X
OpenVZ	Linux	2005	X	X	X	X	X	X
HP Containers	HP/UX	2007	X	X	X	X	X	
LXC	Linux	2008	X	X	X	X	X	X
Docker	Linux	2013	X	X	X	X	X	X

Wikipedia, The Free Encyclopedia. Wikimedia Foundation

Linux kernel features

- **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**: isolated directory trees
- **AppArmor** and **SELinux**: kernel access control

Namespaces

```
$ ls  -l  /proc/$$/ns
total 0
lrwxrwxrwx 1 jorge jorge 0 Dez  5 21:02 cgroup -> cgroup:[4026531835]
lrwxrwxrwx 1 jorge jorge 0 Dez  5 21:02 ipc -> ipc:[4026531839]
lrwxrwxrwx 1 jorge jorge 0 Dez  5 21:02 mnt -> mnt:[4026531840]
lrwxrwxrwx 1 jorge jorge 0 Dez  5 21:02 net -> net:[4026531993]
lrwxrwxrwx 1 jorge jorge 0 Dez  5 21:02 pid -> pid:[4026531836]
lrwxrwxrwx 1 jorge jorge 0 Dez  5 21:02 pid_for_children -> pid:[4026531836]
lrwxrwxrwx 1 jorge jorge 0 Dez  5 21:02 user -> user:[4026531837]
lrwxrwxrwx 1 jorge jorge 0 Dez  5 21:02 uts -> uts:[4026531838]
```

You are already using them !

Container

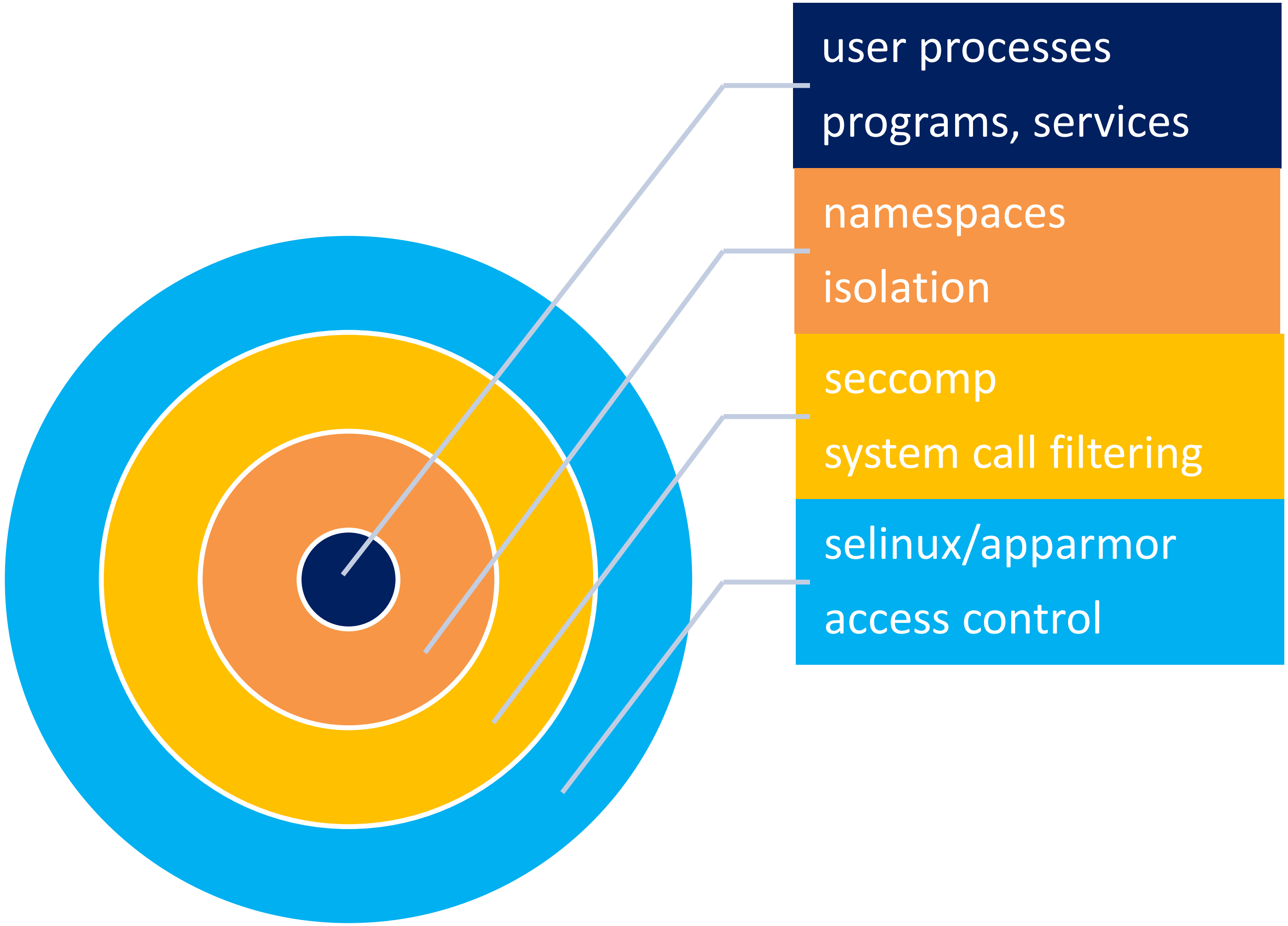
Runs programs as processes in a standard way

No emulation or hypervisors

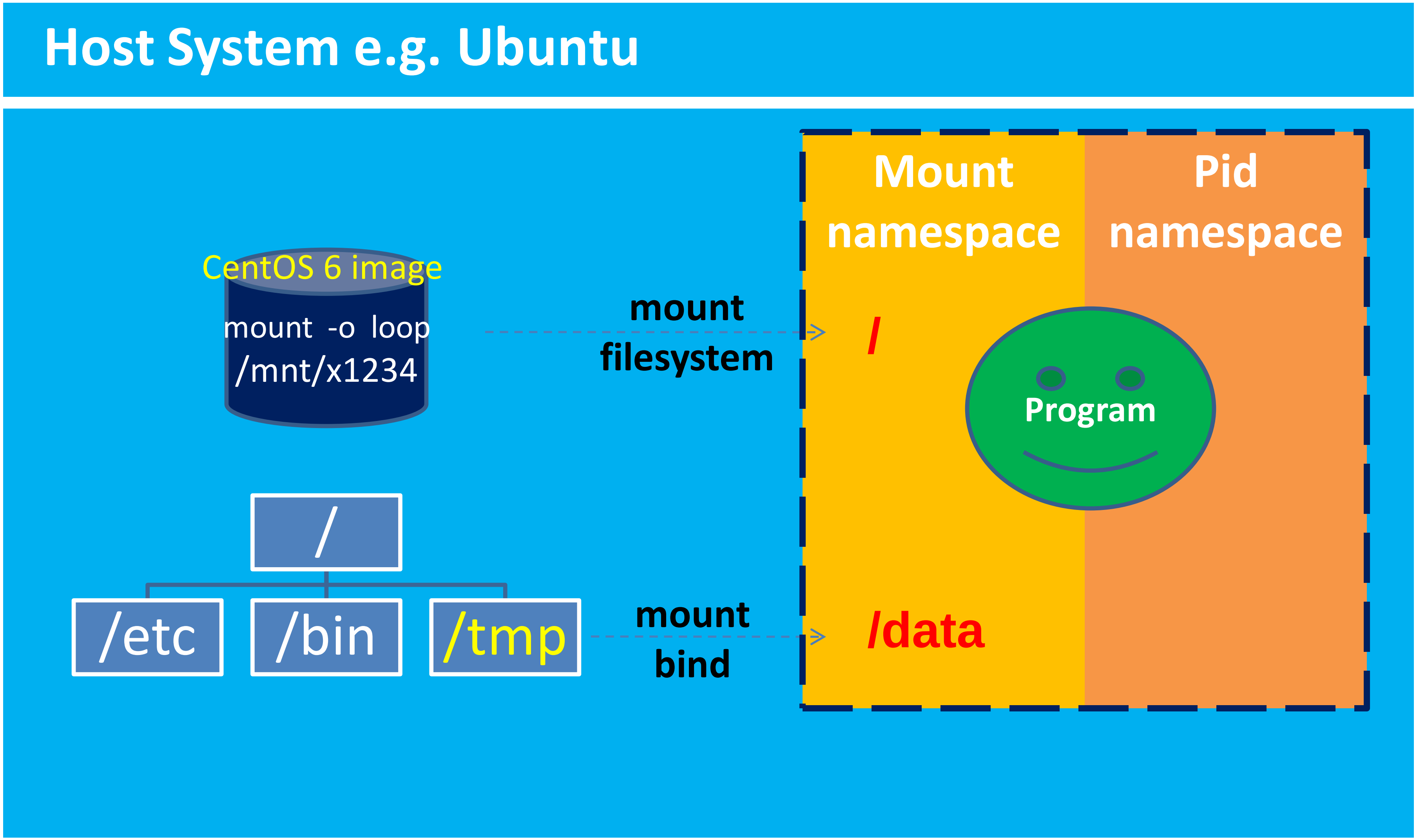
Just process isolation

Therefore much more efficient

Containers


user processes
programs, services
namespaces
isolation
seccomp
system call filtering
selinux/apparmor
access control

Container putting it together



Container putting it together

To create a container image:

- Add the required libraries, programs and data to the file-system
- Add the required programs to the container file-system

Can I run another Linux distribution using containers ?

- **Yes sure**
- **The Linux kernel ABI remains largely unchanged across versions**

Containers are usually started by the root user:

- Some operations require privileges
- Can be root user inside a container without affecting the host or the other containers (with POSIX capabilities, seccomp and namespaces)

LXC/LXD

Linux Containers project (LXC)

- First open source project to provide a toolset for containers

- Create and manage containers using the Linux Kernel features:
  - liblxc library
  - Bindings for several languages (python, ruby, lua, Go)
  - Templates
  - Tools to create/manage containers
- Tools:
  - lxc-create, lxc-destroy, lxc-start, lxc-stop, lxc-execute, lxc-console,
  - lxc-monitor, lxc-wait, lxc-cgroup, lxc-ls, lxc-ps, lxc-info, lxc-freeze,
  - lxc-unfreeze

- Limitations:
  - Requires considerable knowledge and effort

LXD

- Development from the original Linux Containers project
- Pushed and supported by Canonical (Ubuntu)

- Objective:
  - Provide an environment to run complete Linux OS distributions
  - Using Linux container support in the kernel
  - More similar to an hypervisor
  - **Start the complete OS distribution**
  - Images are tarballs

- Limitations:
  - Limited support and adoption beyond Ubuntu

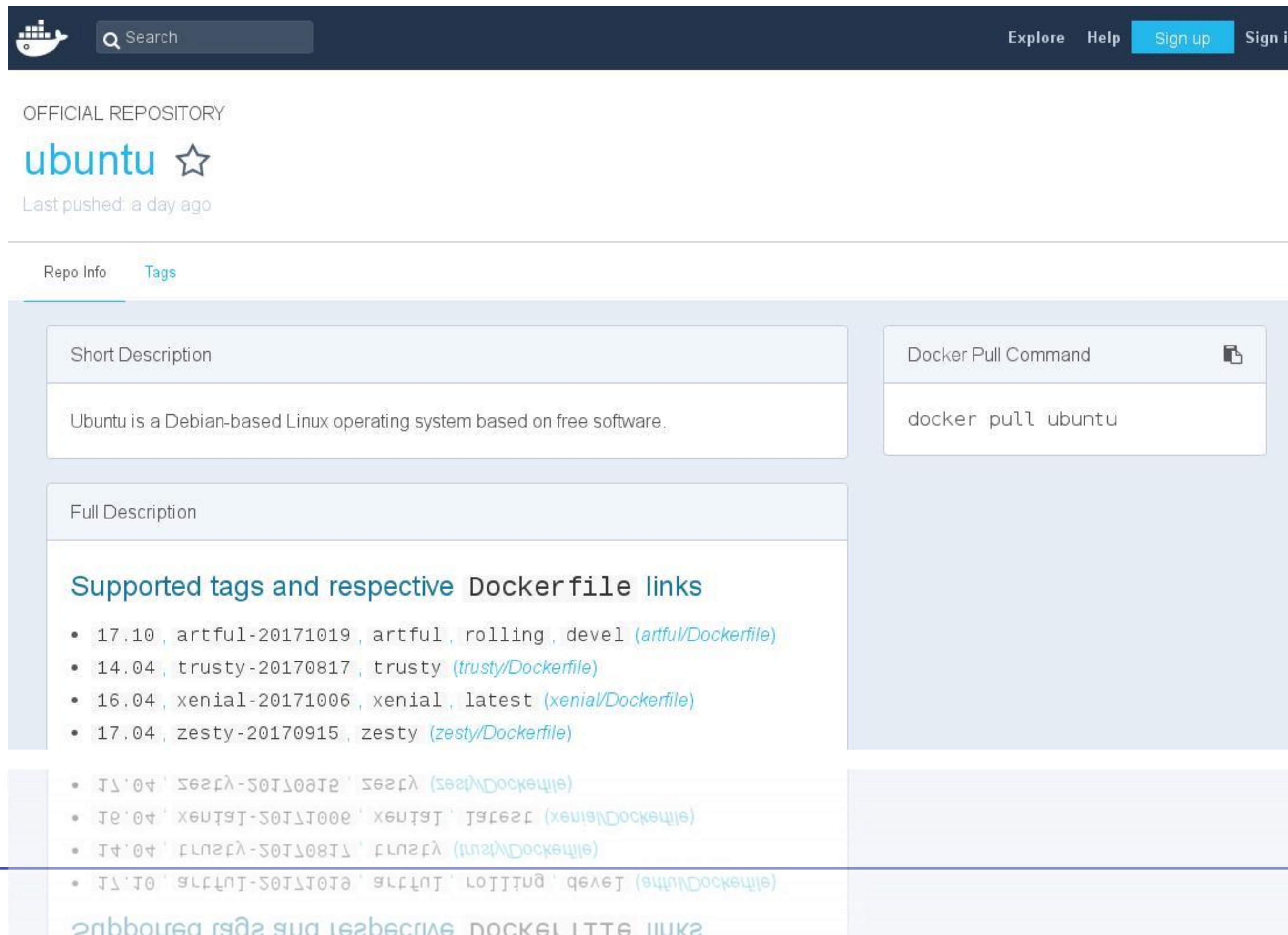
docker

- **Docker containers are oriented to services composition:**
  - (Services or Applications) + (runtime environment)
  - Self-contained and lightweight
  - **Run it everywhere** (Linux)

- **DevOps → integration of IT development and operations**
  - DevOps requires strong automation
  - Developers: focus on what's inside the container
  - Operations: may focus in the underlying infrastructure

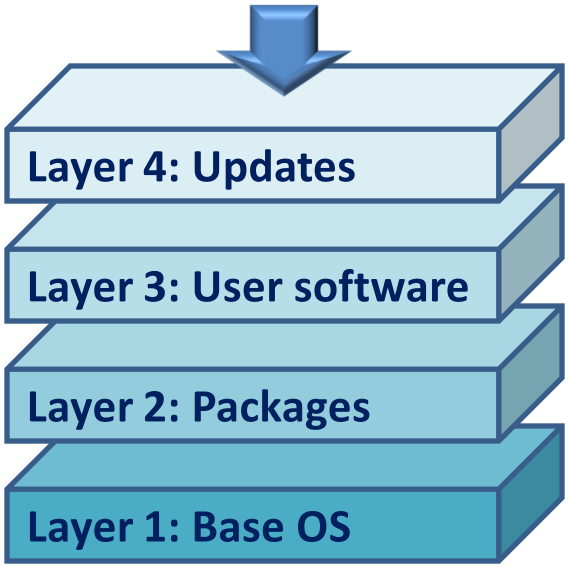
```
$ docker run -i -t centos:centos6
[root@28f89ada747e /]# cat /etc/redhat-release
CentOS release 6.8 (Final)
```

- Docker images can be fetched from the Docker Hub repository
  - There are other Docker container repositories besides Docker Hub
  - Very convenient to transfer and share containers pull/push



docker

- Docker container image is composed of:
  - I. Multiple file-system layers each one:
    - a. metadata
    - b. tarball with the files for the layer
  - II. Manifesto
  - III. Ancestry

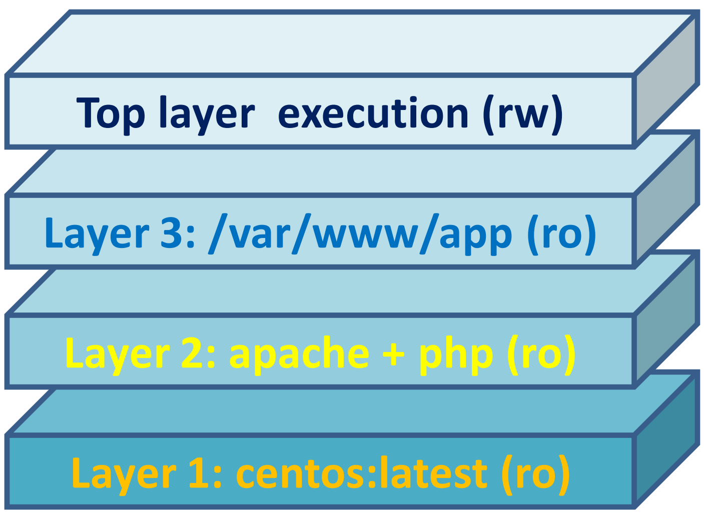


- Layers have unique ids and can be shared by multiple images
- Layers decrease storage space and transfer time
  - e.g. the same OS layer can be shared by many services and applications, avoiding duplication and downloading

docker

- **Common format to distribute and manage images:**
  - Layered file-system based
    - At the host level implemented by AUFS, device-mapper thin snapshots
  - New images can be easily created from existing ones
    - Created by using **Dockerfiles** and docker build

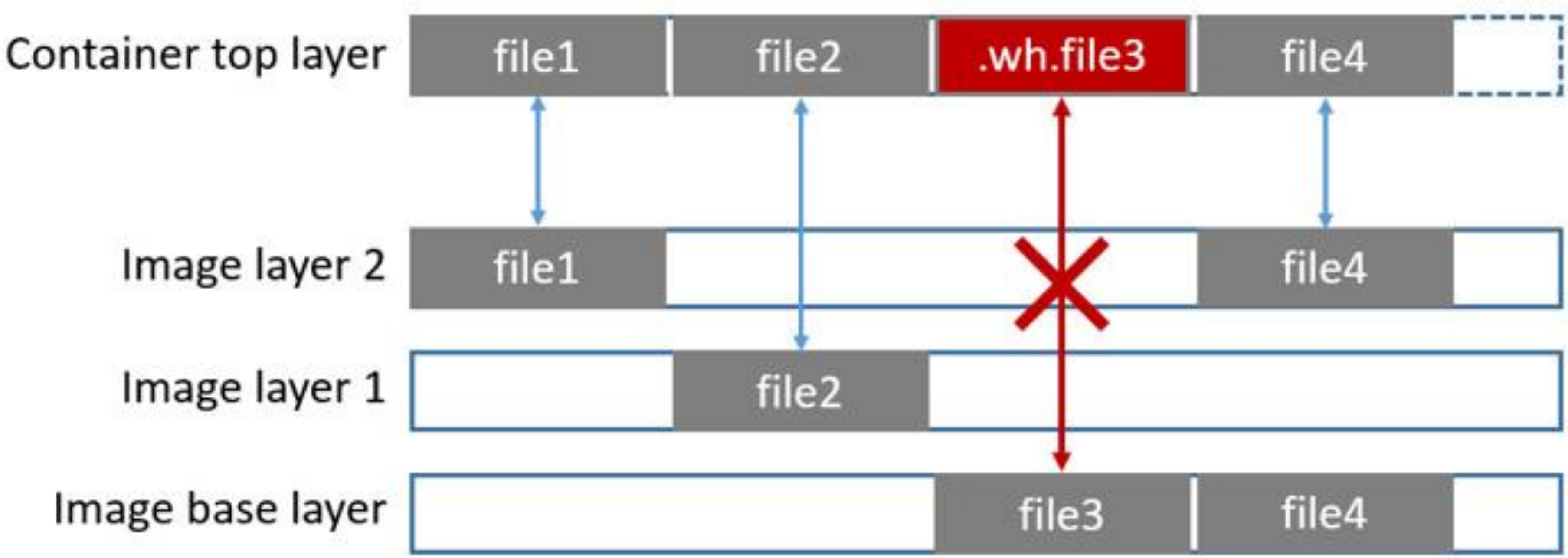
Layers



Dockerfile

```
FROM  centos:centos6
RUN  yum  install  –y  httpd php
COPY  /my/app  /var/www/app
EXPOSE   80
ENTRYPOINT  /usr/sbin/httpd
CMD [“-D”,  “FOREGROUND”]
```

- Layered file-system



Docker container
(AUFS storage-driver demonstrating whiteout file)

Limitations

Require root privileges to install, setup and run

- Security concerns especially in multi-user environments

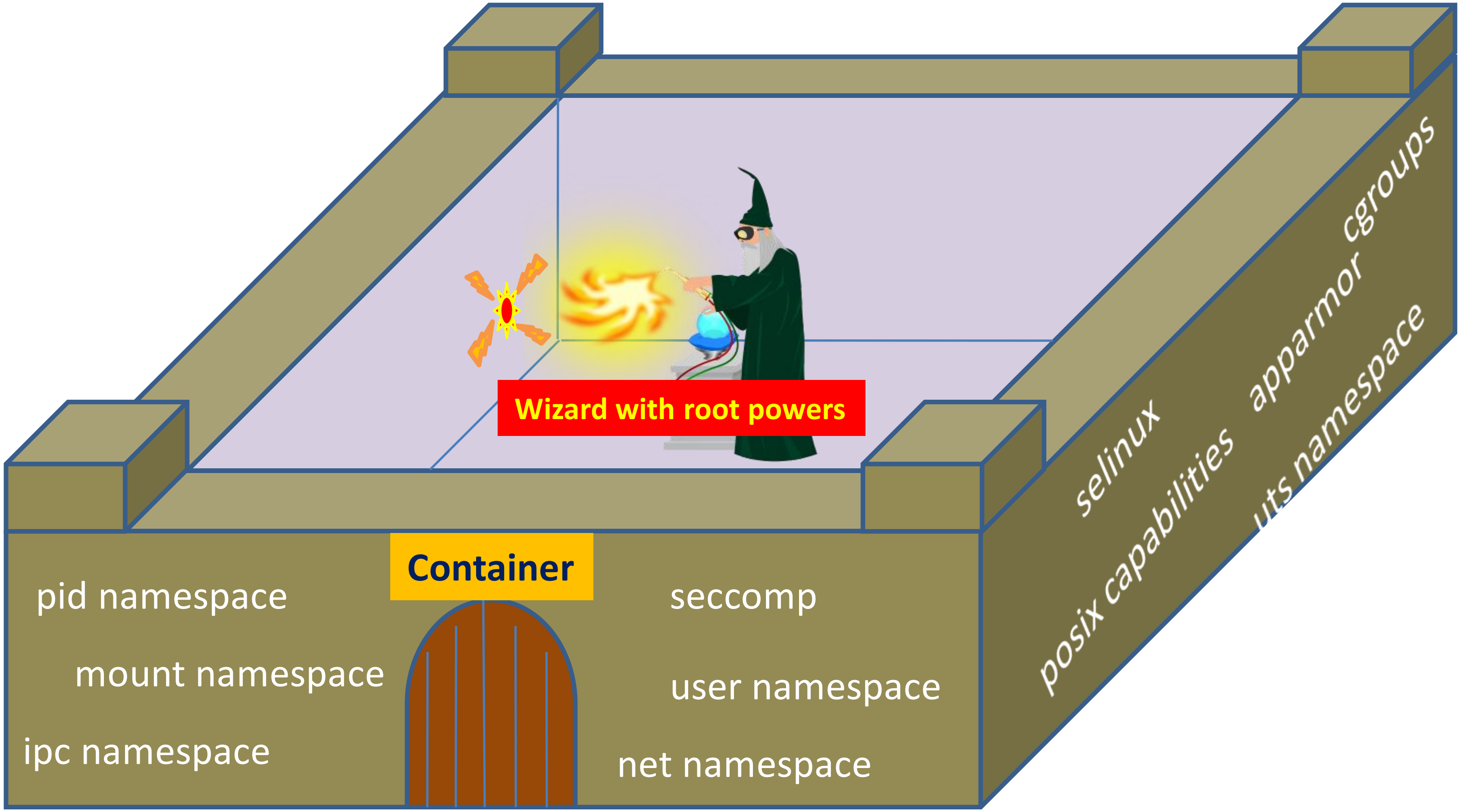
Docker API does not limit privileged actions

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

Limiting design decisions for end users

- Docker is designed to be used as an hypervisor by operators
- Difficult to use on batch systems because of process control and security (not suitable)

Containers in general ...



udocker

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
  - with the normal process controls and accounting
  - in interactive or batch systems

INDIGO-DataCloud **udocker**

udocker in open source

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

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

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

udocker: install from github

```
$ curl  https://raw.githubusercontent.com/indigo-dc/udocker/master/udocker.py > udocker

$ chmod  u+rx  udocker

$ ./udocker  install
```

or **devel**

Does not require compilation or system installation
Tools are delivered statically compiled

udocker: pull images from repository

```
$ udocker pull ubuntu:14.04
```

Search for names and tags at: https://hub.docker.com/

```
Downloading layer: sha256:bae382666908fd87a3a3646d7eb7176fa42226027d3256cac38ee0b79bdb0491
Downloading layer: sha256:f1ddd5e846a849fff877e4d61dc1002ca5d51de8521cced522e9503312b4c4e7
Downloading layer: sha256:90d12f864ab9d4cfe6475fc7ba508327c26d3d624344d6584a1fd860c3f0efa
Downloading layer: sha256:a57ea72e31769e58f0c36db12d25683eba8fa14aaab0518729f28b3766b01112
Downloading layer: sha256:783a14252520746e3f7fee937b5f14ac1a84ef248ea0b1343d8b58b96df3fa9f
Downloading layer: sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
```

udocker: list local images

```
$ udocker images
```

```
REPOSITORY
msoffice:lastest                                   .
iscampos/openqcd:latest                            .
fedora:25                                          .
docker.io/susymastercode/mastercode:latest         .
ubuntu:14.04                                       .
ubuntu:16.10                                       .
ubuntu:latest                                      .
indigodatacloud/disvis:latest                      .
jorge/private:latest                               .
busybox:latest                                     .
jorge_fedora22_32bit:latest                        .
debian:oldstable                                   .
```

udocker: create container from image

```
$ udocker create --name=ub14 ubuntu:14.04
```

container-alias

```
9fe2f9e7-ce37-3be5-b12d-829a3236d2a6
```

container-id

udocker: list containers

```
$ udocker  ps
```

container-id alias image

```
CONTAINER ID                         P M NAMES                  IMAGE
9fe2f9e7-ce37-3be5-b12d-829a3236d2a6 . W ['ub14']               ubuntu:14.04
5c7bd29b-7ab3-3d73-95f9-4438443aa6d6 . W ['myoffice']           msoffice:lastest
676eb77d-335e-3e9a-bf62-54ad08330b99 . W ['fedora_25']          fedora:25
c64afe05-adfa-39de-bf15-dcd45f284249 . W ['debianold']          debian:oldstable
7e76a4d7-d27e-3f09-a836-abb4ded0df34 . W ['ubuntu16', 'S']      ubuntu:16.10
9d12f52d-f0eb-34ae-9f0e-412b1f8f2639 . W ['f25']                fedora:25
```

udocker: run container

```
$ udocker run ub14
```

udocker respects container metadata, if the container has a default cmd to run it will be run otherwise starts a shell

```
 ******************************************************************************
 *                                                                            *
 *               STARTING 9fe2f9e7-ce37-3be5-b12d-829a3236d2a6                *
 *                                                                            *
 ******************************************************************************
 executing: bash
root@nbjorge:/# cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04.5 LTS"
root@nbjorge:/# apt-get install firefox
```

root emulation

udocker: run container as yourself

```
$ udocker run --user=jorge -v /home/jorge \
  -e HOME=/jorge/home --workdir=/home/jorge ub14
```

```
Warning: non-existing user will be created

 ##############################################################################
 *                                                                            *
 *               STARTING 9fe2f9e7-ce37-3be5-b12d-829a3236d2a6                *
 *                                                                            *
 ##############################################################################
 executing: bash
jorge@nbjorge:~$ id
uid=1000(jorge) gid=1000(jorge) groups=1000(jorge),10(uucp)
jorge@nbjorge:~$ pwd
/home/jorge
jorge@nbjorge:~$
```

udocker: run commands in the prompt

```
$ udocker run --user=jorge --bindhome \
  --hostauth ub14 /bin/bash <<EOF
id; pwd
EOF
```

```
 ******************************************************************************
 *                                                                            *
 *               STARTING 9fe2f9e7-ce37-3be5-b12d-829a3236d2a6                *
 *                                                                            *
 ******************************************************************************
 executing: bash
uid=1000(jorge) gid=1000(jorge) groups=1000(jorge),10(uucp)
/home/jorge
```

udocker: duplicate a container

```
$ udocker  clone  --name=yy  ub14
```

cloned container-id

```
9fe2f9e7-ce37-3be5-b12d-829a3236d2a6
```

udocker: export and import as image

export to
tarball

```
$ udocker export -o ub14.tar ub14

$ udocker import ub14.tar myub14:latest
```

import from
tarball

new image
name

- Only the container files are exported, metadata is lost
- This is interoperable with docker

udocker: export and import as container

export to
tarball

```
$ udocker  export  -o ub14.tar  ub14

$ udocker  import  --tocontainer  --name=xx  ub14.tar
```

import from tarball
to container

new container
alias

- Only the container files are exported, metadata is lost
- Export is interoperable with docker

udocker: export and import as container

export clone

```
$ udocker  export  --clone  -o ub14.tar  ub14

$ udocker  import  --clone  --name=xx  ub14.tar
```

import clone

- Is imported as a container saving space and time
- Container metadata and execution mode are preserved
- This is NOT interoperable with docker

udocker: save and load images

save image with all layers and metadata

```
$ docker save centos:centos6 | udocker load
```

load image with all layers and metadata

- Save from docker and load with udocker
- Piping stdout to stdin

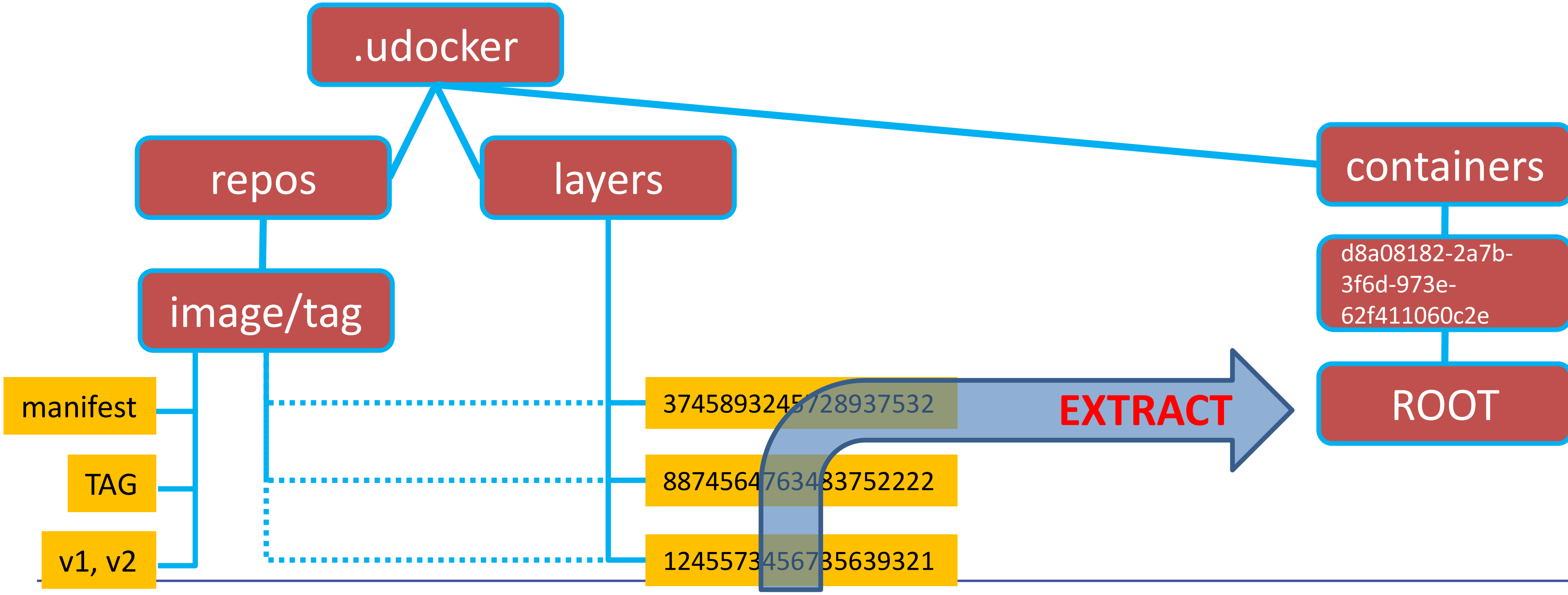
udocker
How does it work ...

udocker:

- Implemented
  - python, C, C++, go
- Can run:
  - CentOS 6, CentOS 7, Fedora >= 23
  - Ubuntu 14.04, Ubuntu 16.04
  - Any distro that supports python 2.7
- Components:
  - Command line interface docker like
  - Pull of containers from Docker Hub
  - Local repository of images and containers
  - Execution of containers with modular engines

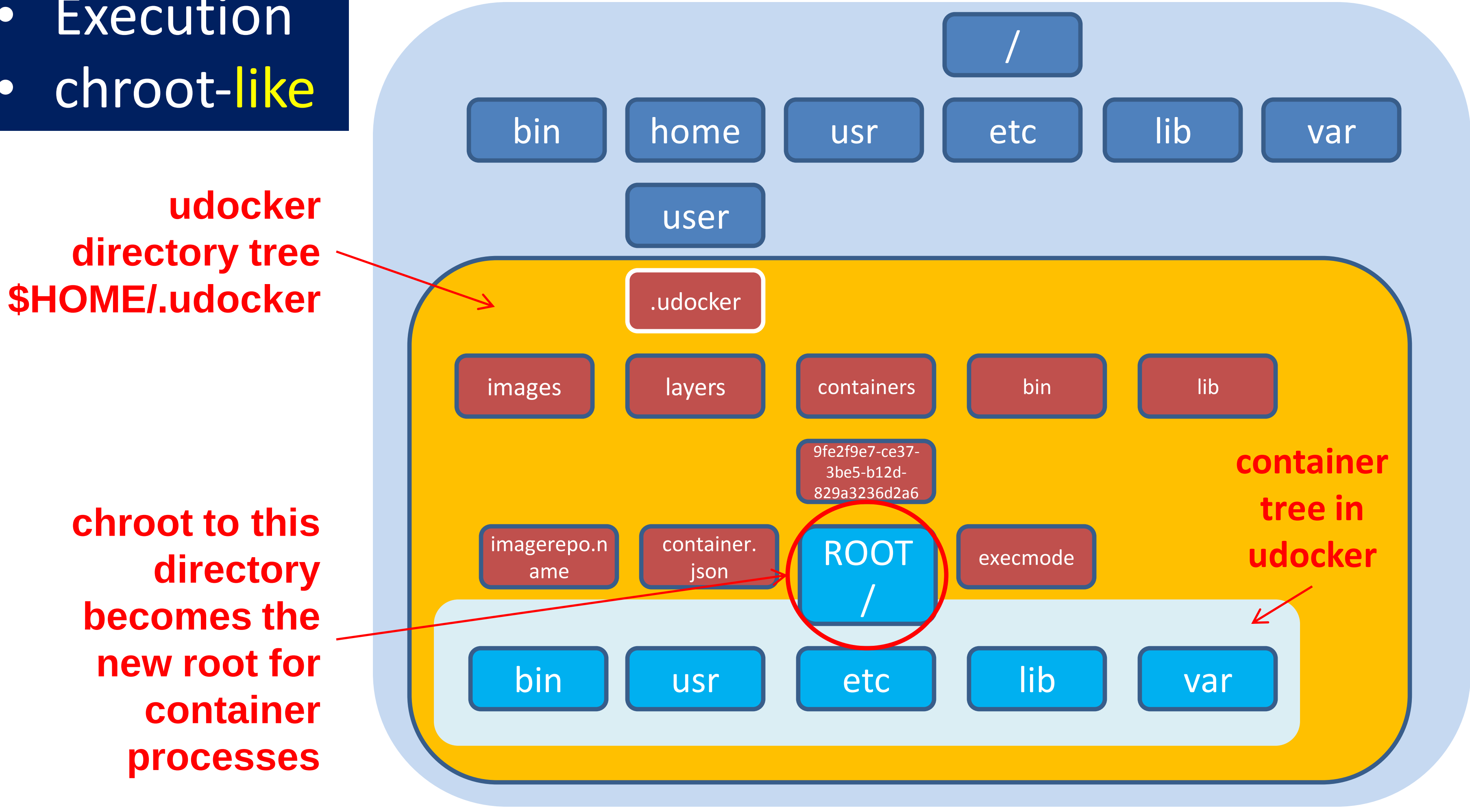
udocker:

- Containers
  - Are produced from the layers by flattening them
  - Each layer is extracted on top of the previous
  - Whiteouts are respected, protections are changed
  - The obtained directory trees are stored under ~/.udocker/containers in the user home directory



udocker: directories and execution

- Execution
- chroot-like

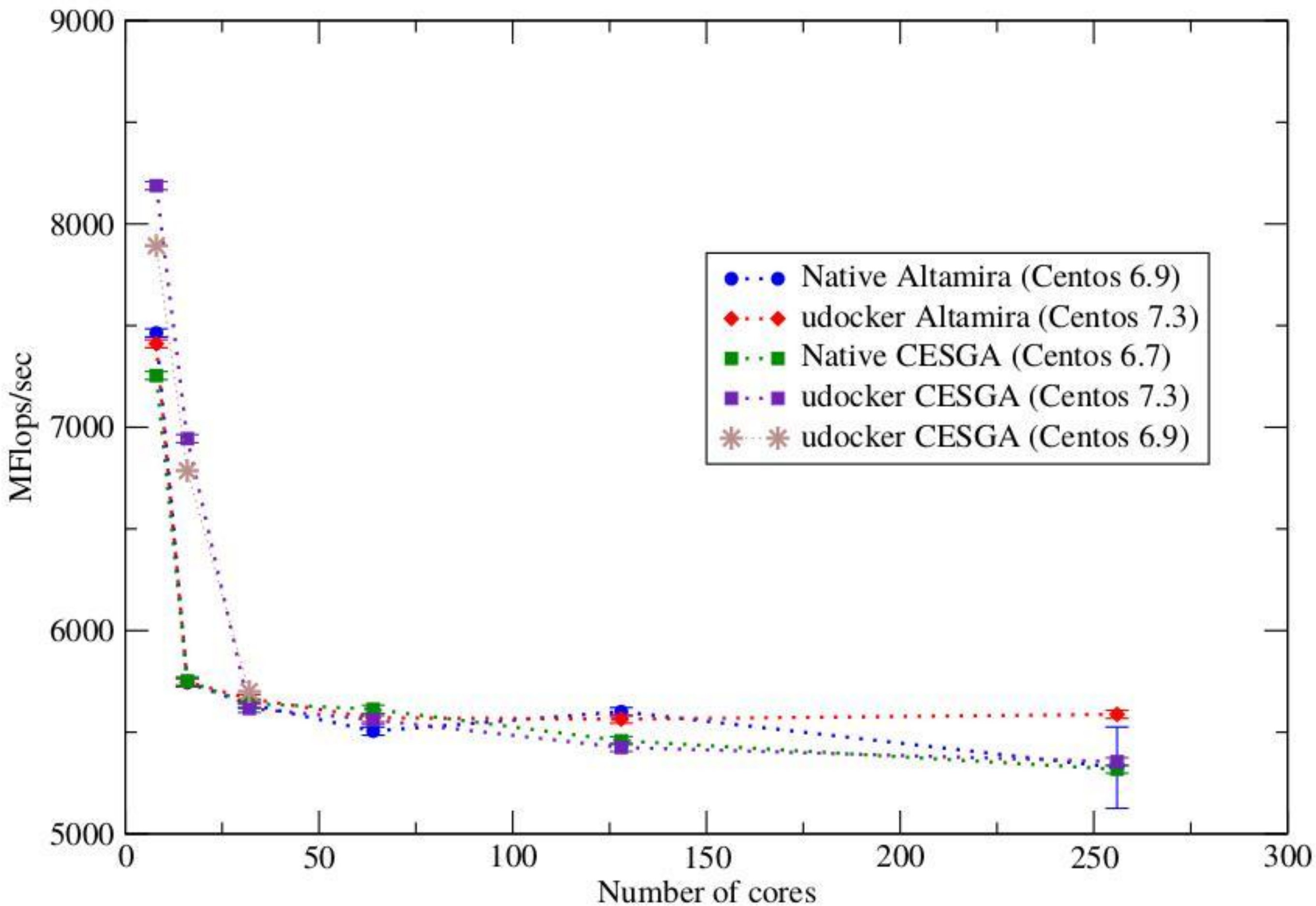


udocker: Execution methods

- udocker supports several techniques to achieve the equivalent to a chroot without using privileges
  - 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

udocker & Lattice QCD



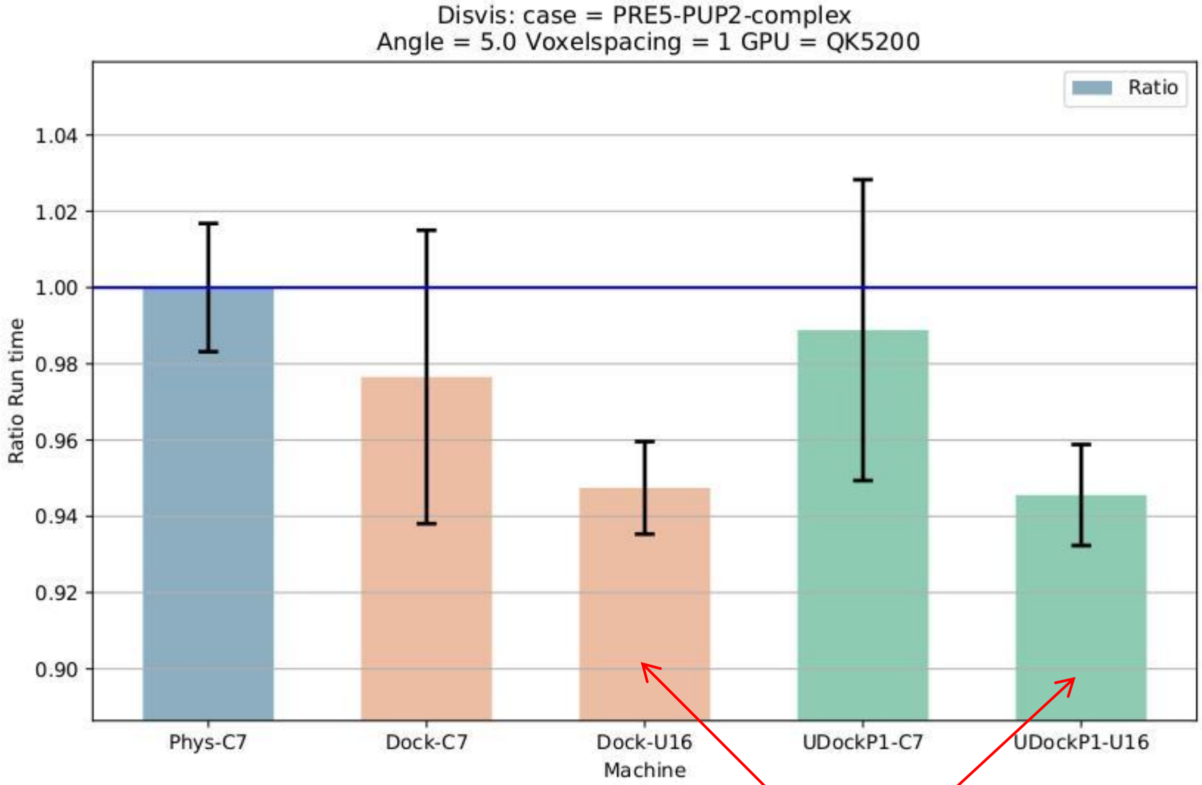
OpenQCD is a very advanced code to run lattice simulations

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

udocker & Biomolecular complexes



Better performance with Ubuntu 16 container

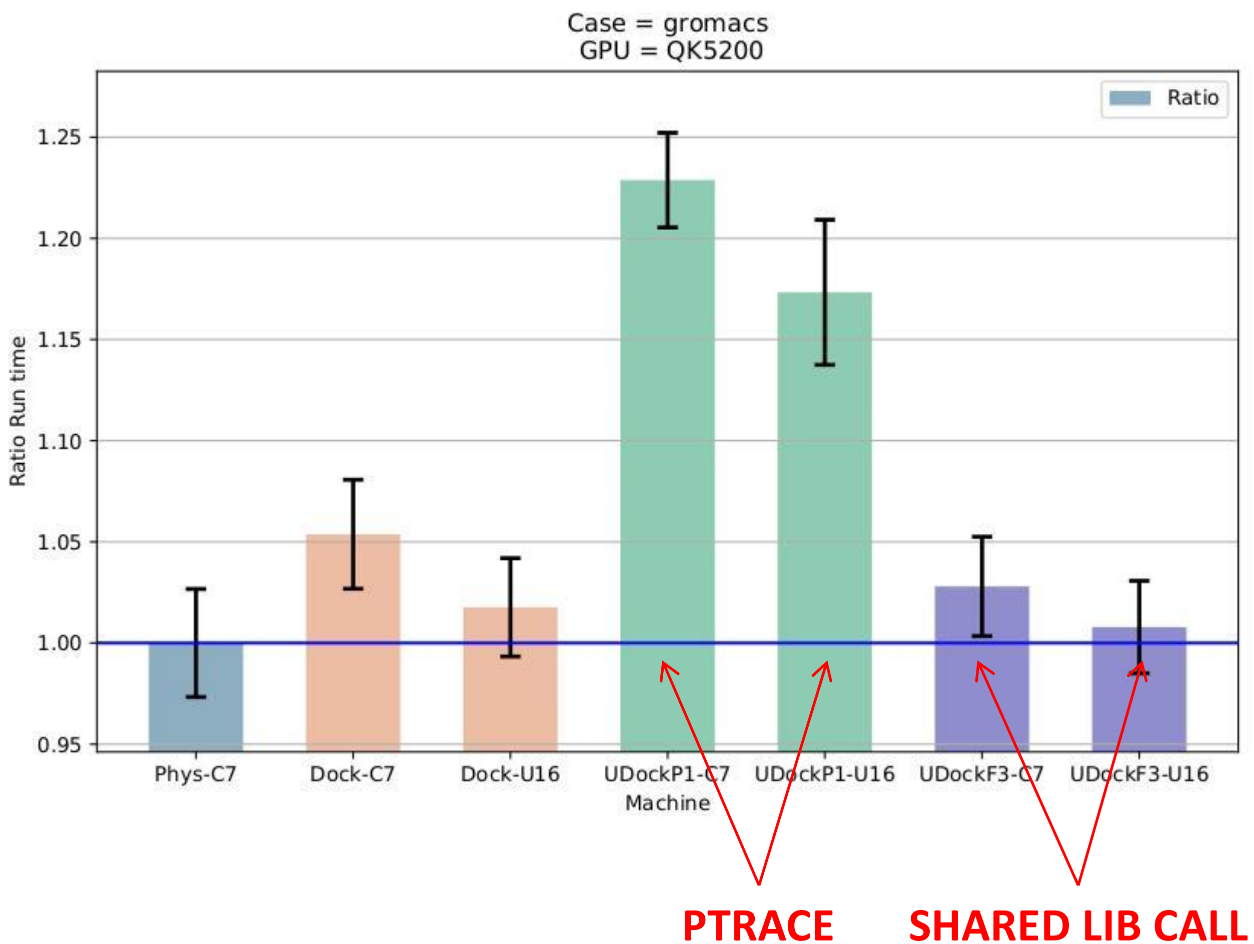
DisVis is being used in production with udocker

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

Using OpenCL and NVIDIA GPGPUs

udocker in P1 mode

udocker & 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

udocker & Phenomenology

Performance Degradation

	Compiling	Running
HOST	0%	0%
DOCKER	10%	1.0%
udocker	7%	1.3%
VirtualBox	15%	1.6%
KVM	5%	2.6%

udocker in P1 mode

MasterCode connects several complex codes. Hard to deploy.

Scanning through large parameter spaces. High Throughput Computing

C++, Fortran, many authors, legacy code

udocker & Phenomenology

```
export MASTERDIR=/gpfs/csic_users/userabc/mastercode
export UDOCKER_DIR=$MASTERDIR/.udocker

 udocker.py run --hostauth \
          -v  /home/csic/cdi/ica/mcpp-master \
          -v  /home/csic/cdi/ica \
          -user=user001 \
          -w  /home/csic/cdi/ica/mcpp-master  mastercode \
          /bin/bash -c "pwd; ./udocker-mastercode.sh"
```

Thank you

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