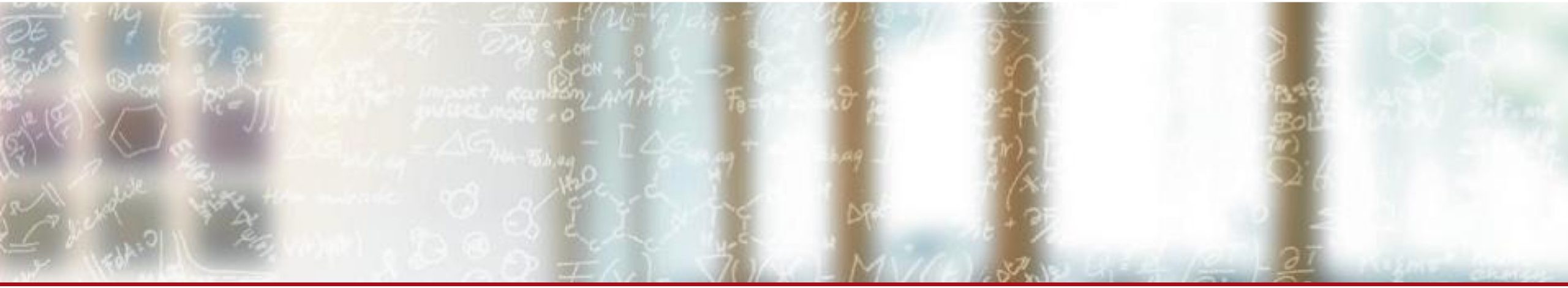
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



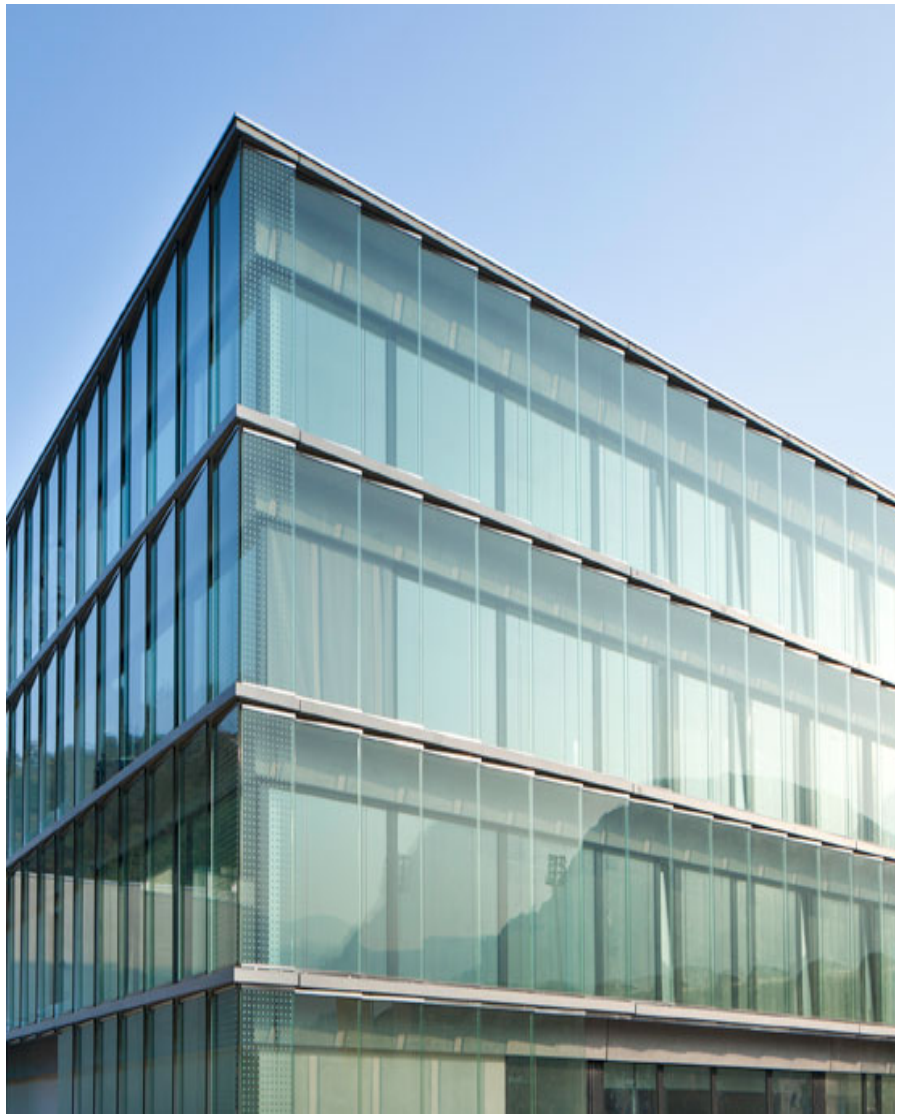
Sarus: Highly Scalable Docker Containers for HPC Systems

5th EasyBuild User Meeting
Jan 29th – 31st 2020, Barcelona

Luca Marsella and **Alberto Madonna**

Swiss National Supercomputing Center (CSCS)

Outline



- **Introduction**
  - **Linux Containers Ecosystem**
  - **Containers timeline @ CSCS**
- Sarus
  - CSCS requirements
  - Sarus features
- OCI hooks
  - NVIDIA Container Toolkit and MPI hook
- Use case
  - Intel MKL and MPI with NVIDIA Cuda
- Performance tests
  - Gromacs, TensorFlow, Cosmo, QE

Containers

- Isolated environments to run applications / services
- Container images include all software dependencies
- Prescriptive, portable, easy to build, quick to deploy

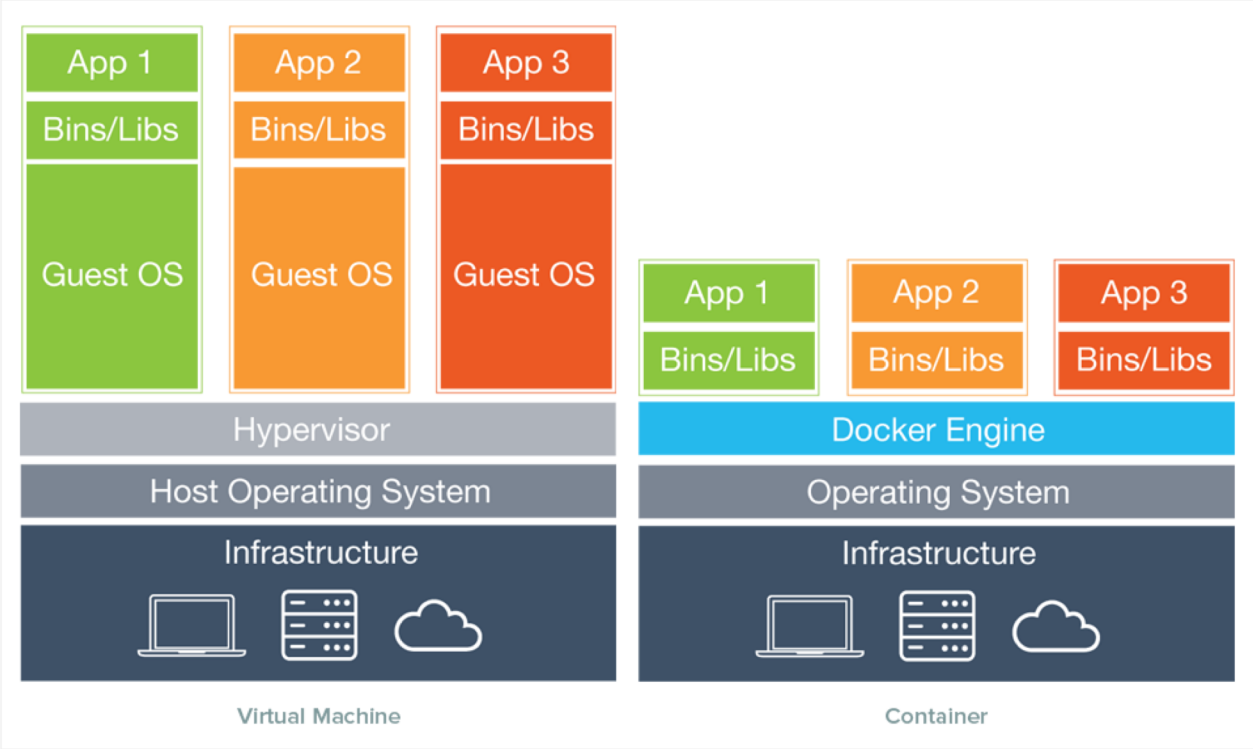
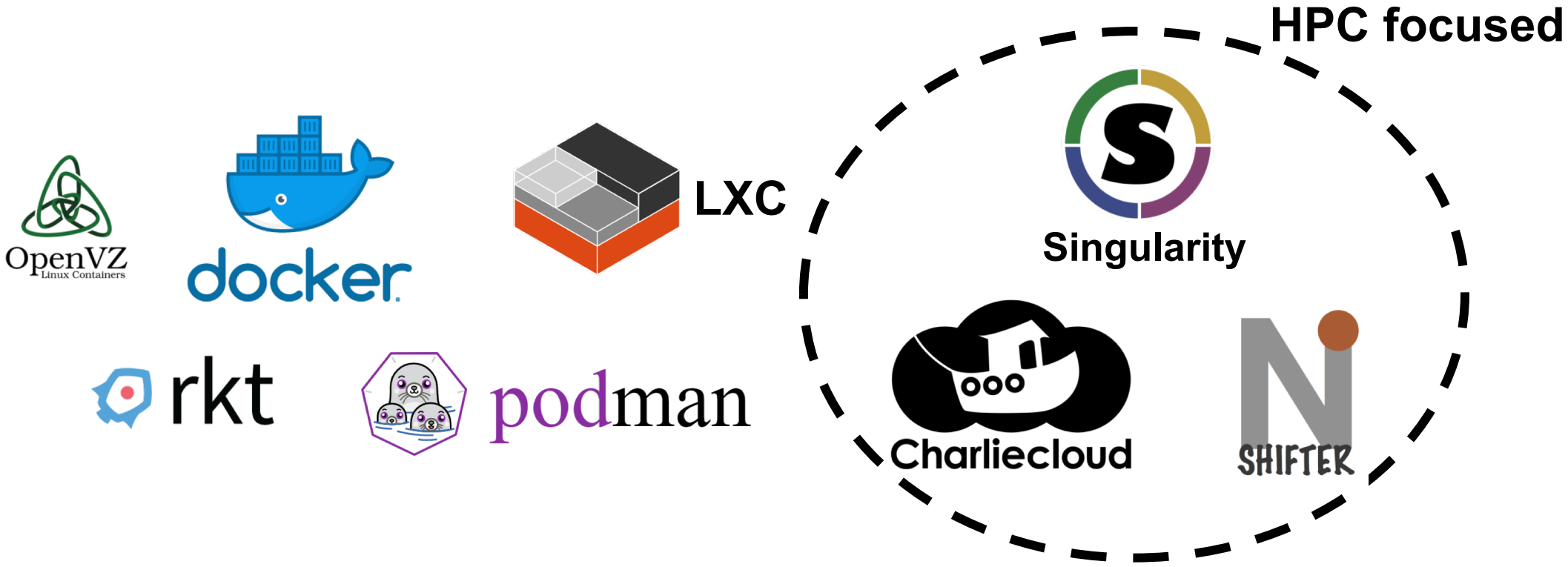


Image credit: Docker Inc.

Linux Containers Ecosystem

- Linux containers rely on abstraction features (*namespaces*[1]) provided by the kernel
- Different design decisions and use cases gave rise to several solutions:

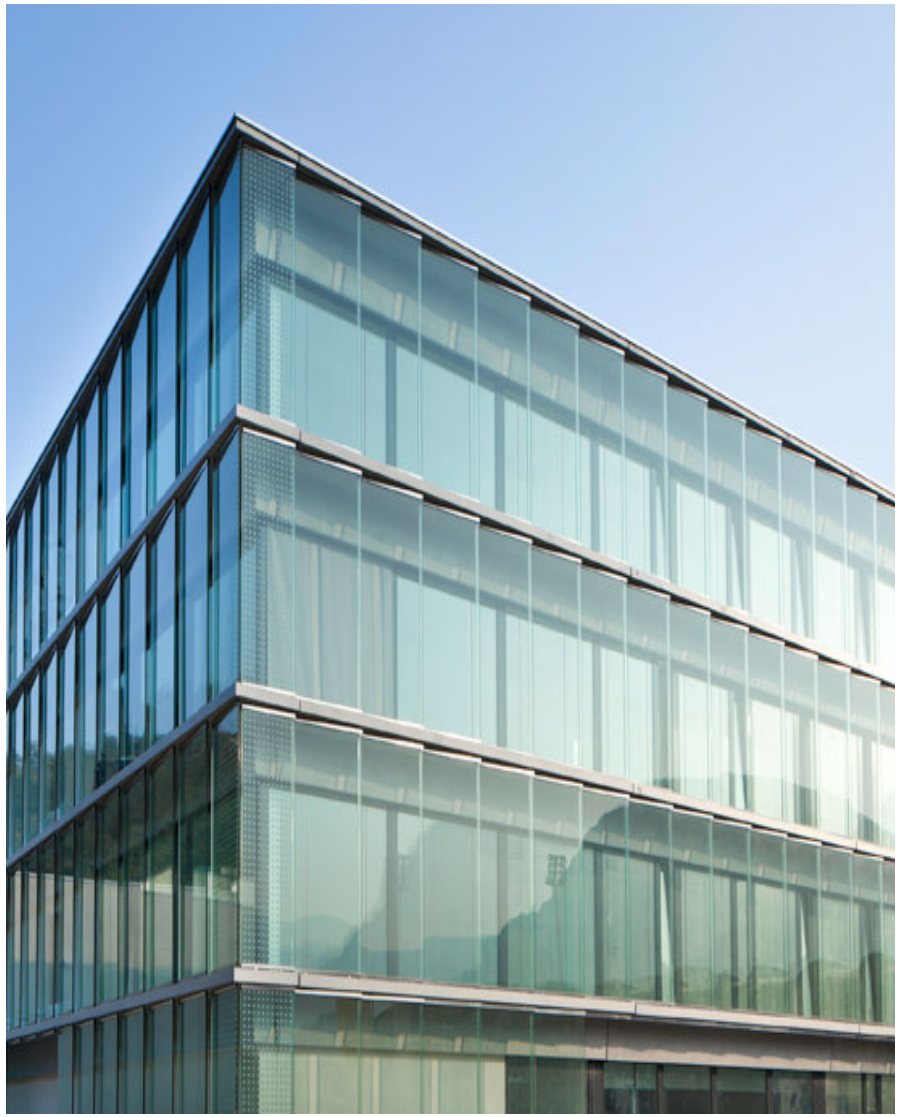


[1] **Namespaces in operation, part 1: namespaces overview** at https://lwn.net/Articles/531114

Containers timeline @ CSCS

2015	Early Docker evaluation for accessing GPUs
2016	Extended Shifter: - Native GPU support - Native MPI support
2017	Shifter in production
2018	Developed Shifter-NG (internally): - Docker-like command line interface - Image manager
2018	Shifter-NG in production
2019	Sarus in production on Piz Daint (11.19)

Outline



- Introduction
  - Linux Containers Ecosystem
  - Containers timeline @ CSCS
- **Sarus**
  - **CSCS requirements**
  - **Sarus features**
- OCI hooks
  - NVIDIA Container Toolkit and MPI hook
- Use case
  - Intel MKL and MPI with NVIDIA Cuda
- Performance tests
  - Gromacs, TensorFlow, Cosmo, QE

CSCS Requirements

- **Suitable for HPC**
  - Diskless compute nodes
  - Compatibility with workload managers
  - Parallel filesystem friendly
  - No privilege escalation (multi tenant)
- **Pluggable vendor support** (standard OCI hooks)
  - NVIDIA GPU
  - Cray MPI
  - RDMA (e.g. GPUDirect)
- **User Experience**
  - Docker-like CLI
  - Docker Hub integration
  - Writable container filesystem
- **Admin experience**
  - Easy installation (e.g. single binary)
  - Container customization (e.g. plugins)
- **Maintenance effort**
  - Small codebase (reuse 3rd party tech)
  - Leverage community efforts

CSCS Requirements

- **Suitable for HPC**
  - Diskless compute nodes
  - WLM compatibility
  - Parallel filesystem friendly
  - No privilege escalation (multi tenant)

- **Pluggable vendor support** (standard OCI hooks)
  - NVIDIA GPU
  - Cray MPI
  - RDMA (e.g. GPUDirect)

- **User Experience**
  - Docker-like CLI
  - Docker Hub integration
  - Writable container filesystem

- **Admin experience**
  - Easy installation (e.g. single binary)
  - Container customization (e.g. plugins)

- **Maintenance effort**
  - Small codebase (reuse 3rd party tech)
  - Leverage community efforts

CSCS Requirements

- **Suitable for HPC**
  - Diskless compute nodes
  - WLM compatibility
  - Parallel filesystem friendly
  - No privilege escalation (multi tenant)
- **Pluggable vendor support** (standard OCI hooks)
  - NVIDIA GPU
  - Cray MPI
  - RDMA (e.g. GPUDirect)
- **User Experience**
  - Docker-like command line interface
  - Docker Hub integration
  - Writable container filesystem
- **Admin experience**
  - Easy installation (e.g. single binary)
  - Container customization (e.g. plugins)
- **Maintenance effort**
  - Small codebase (reuse 3rd party tech)
  - Leverage community efforts

CSCS Requirements

- **Suitable for HPC**
  - Diskless compute nodes
  - WLM compatibility
  - Parallel filesystem friendly
  - No privilege escalation (multi tenant)
- **Pluggable vendor support** (standard OCI hooks)
  - NVIDIA GPU
  - Cray MPI
  - RDMA (e.g. GPUDirect)
- **User Experience**
  - Docker-like CLI
  - Docker Hub integration
  - Writable container filesystem
- **Admin experience**
  - Easy installation (e.g. single binary)
  - Container customization (e.g. OCI hooks)
- **Maintenance effort**
  - Small codebase (reuse 3rd party tech)
  - Leverage community efforts

CSCS Requirements

- **Suitable for HPC**
  - Diskless compute nodes
  - WLM compatibility
  - Parallel filesystem friendly
  - No privilege escalation (multi tenant)
- **Pluggable vendor support** (standard OCI hooks)
  - NVIDIA GPU
  - Cray MPI
  - RDMA (e.g. GPUDirect)
- **User Experience**
  - Docker-like CLI
  - Docker Hub integration
  - Writable container filesystem
- **Admin experience**
  - Easy installation (e.g. single binary)
  - Container customization (e.g. plugins)
- **Maintenance effort**
  - Small codebase (reuse 3rd party tech)
  - Leverage community efforts

Sarus features: usability

- **Suitable for HPC**
  - Single squashfs image, parallel filesystem friendly
  - Image loop mount + RAM filesystem for fast image access
  - Workload Manager compatible, native GPU and MPI support
- **Pluggable vendor support**
  - NVIDIA Container Toolkit
  - Standard OCI hooks support
- **User Experience**
  - Docker-like command line interface with Docker Hub integration
  - OverlayFS for writable container filesystem
  - Preserve identity and file permissions

Sarus features: maintainability

- **Admin experience**
  - Single executable binary easily deployable
  - OCI hooks for container customization
  - Secure container isolation through runc
- **Maintenance effort**
  - Reuse runc as the core runtime and third-party software
  - Well tested with unit test coverage ~ 84%
  - Compliant with OCI standard

Sarus features: command line interface

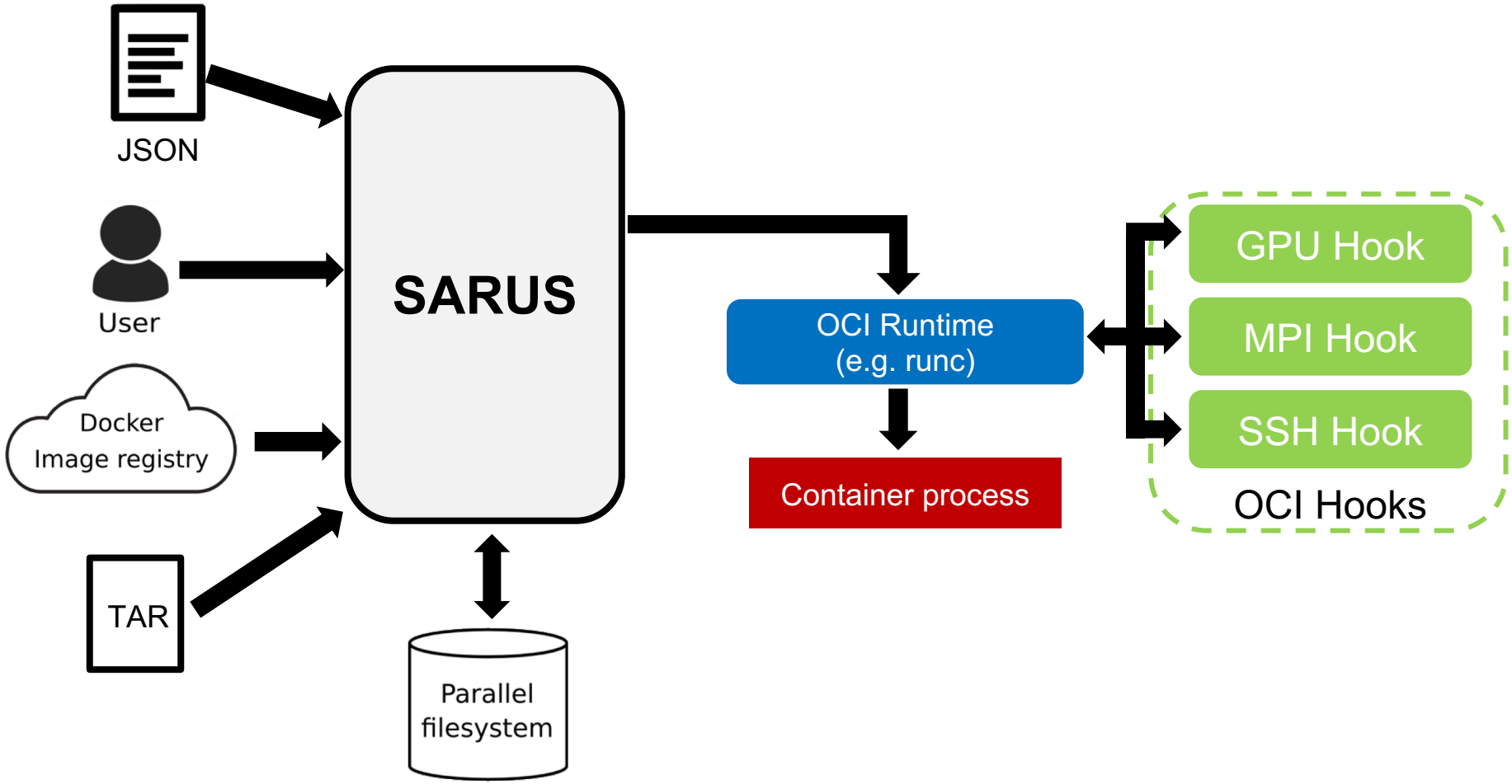
- Sarus

```
$ sarus pull [options] <image>[<:tag>]

$ sarus load [options] <file> <image>

$ sarus images

$ sarus rmi <image>[<:tag>]

$ sarus run [options] <image>[<:tag>]
<command> <args>
```

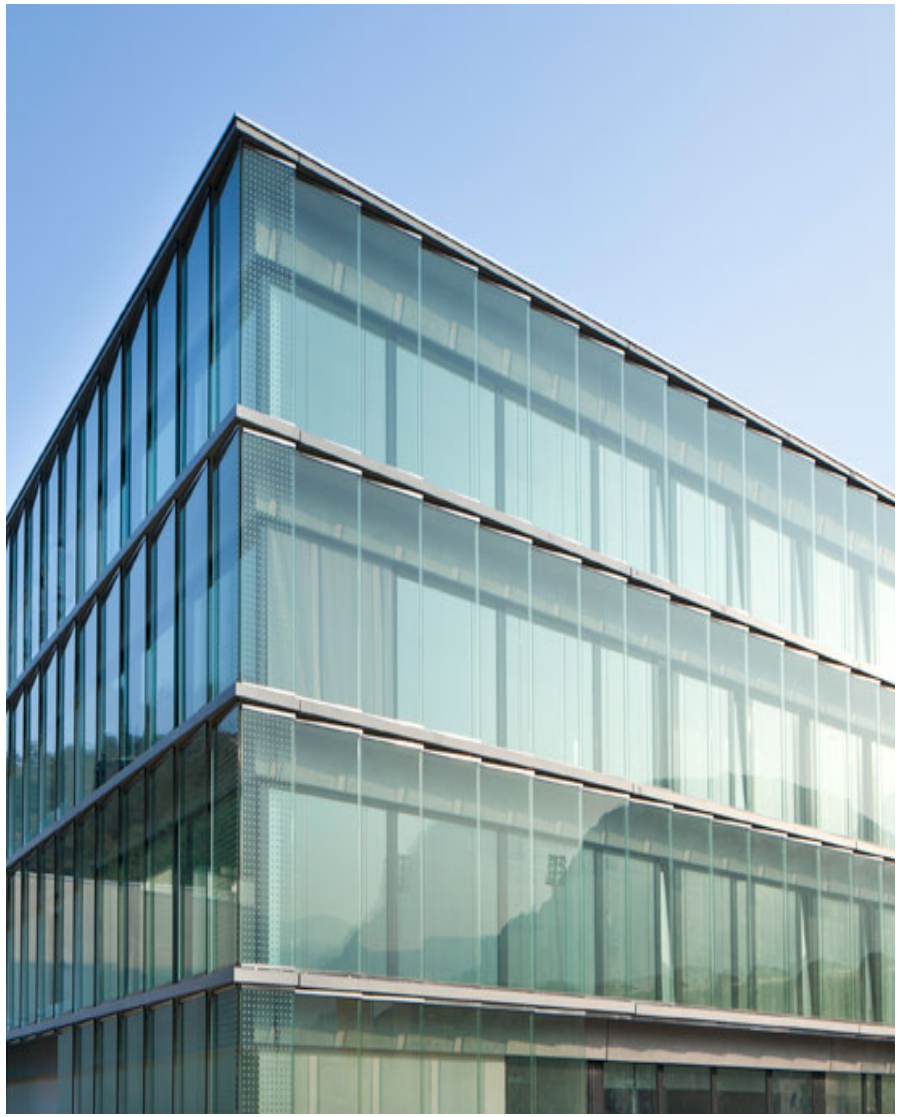
- Docker

```
$ docker pull [options] <image>[<:tag>]

$ docker load [options] -i <file>

$ docker images [options] [repo[<:tag>]]

$ docker rmi [options] <image> [image…]

$ docker run [options] <image>[<:tag>]
<command> <args>
```

Sarus architecture overview



Outline



- Introduction
  - Linux Containers Ecosystem
  - Containers timeline @ CSCS
- Sarus
  - CSCS requirements
  - Sarus features
- **OCI hooks**
  - **NVIDIA Container Toolkit and MPI hook**
- Use case
  - Intel MKL and MPI with NVIDIA Cuda
- Performance tests
  - Gromacs, TensorFlow, Cosmo, QE

The Open Container Initiative (OCI) Hooks

- An open governance structure for the express purpose of creating open industry standards around container formats and runtime at https://www.opencontainers.org

- The OCI Runtime Specification defines an interface for a plug-in (or **hook**)
  - external programs will use the interface to customize the container at runtime

- Primary HPC use cases:
  - High-performance MPI
  - GPU acceleration

CSCS OCI Hooks

- NVIDIA Container Toolkit for GPU support
- Hooks developed at CSCS and bundled with Sarus:
  - MPI hook (MPICH-based)
    - Native performance from host MPICH-based libraries
  - Glibc hook
    - Replaces container's glibc if older than host's glibc
    - Ensures that mounted host resources (e.g. MPI) will work inside the container
  - SSH hook
    - Setup ssh connections inside containers
  - SLURM sync hook
    - Waits for all processes in a SLURM job to start before executing container applications
  - Timestamp hook
    - Writes a timestamp (useful for developers to time/profile hooks)

NVIDIA Container Toolkit

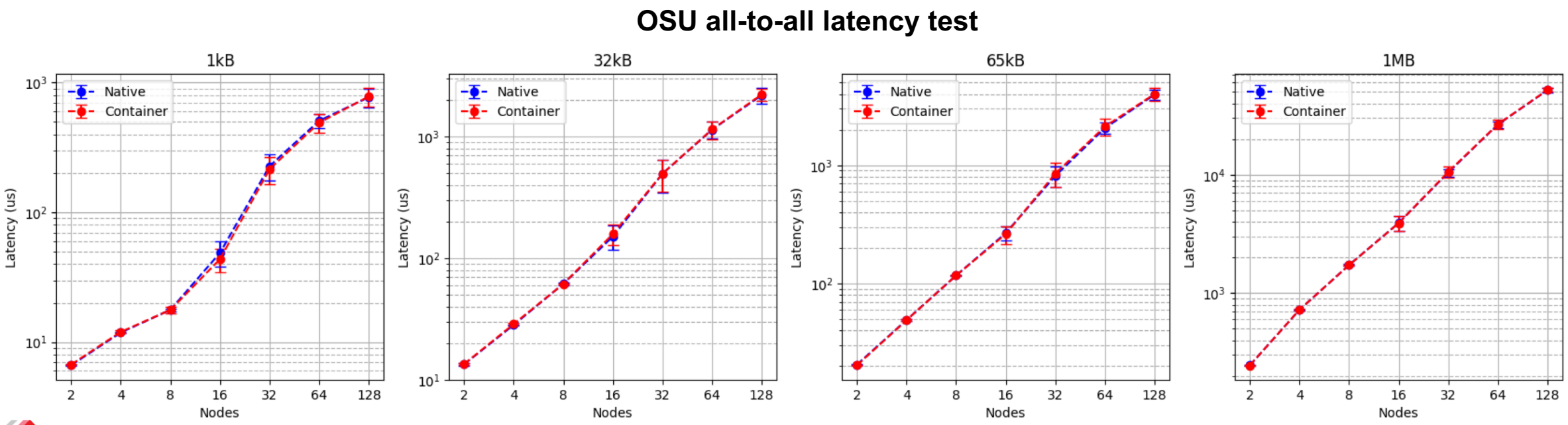
- Part of the NVIDIA Container Runtime (https://github.com/NVIDIA/nvidia-container-runtime)
- Imports the NVIDIA driver and GPU device files into the container
- Native performance, no input required from the user
- First example of **vendor** hook to be successfully tested on Piz Daint

CUDA SDK N-body sample: FP64 GFLOPS		
	Average	Std. deviation
Native	3059.34	5.30
Container	3058.91	6.29

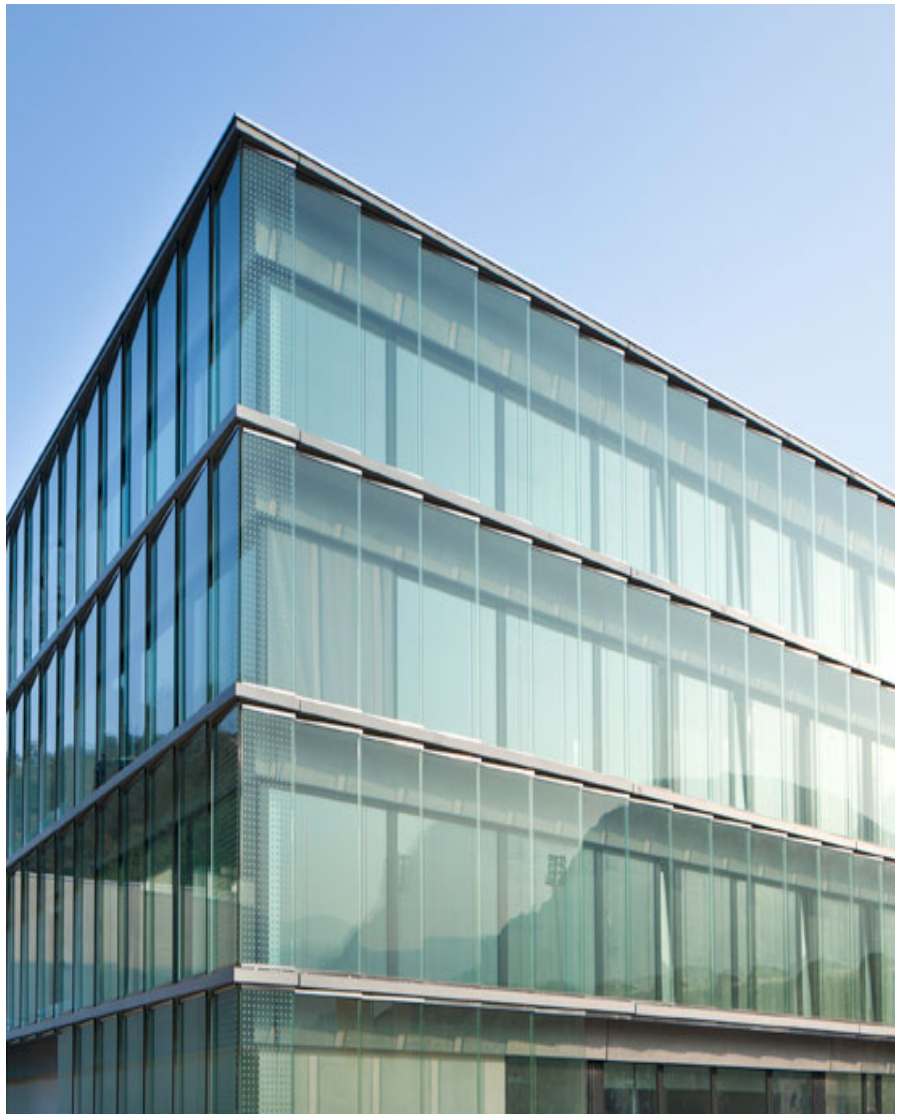
MPI Hook

- Replace the container MPI with host libraries at runtime, achieving native performance
- Relies on MPICH ABI compatibility (https://www.mpich.org/abi)
- Completely transparent to the user:

```
sarus run --mpi ethcscs/osu-mb:5.3.2-mpich3.1.4 ../collective/osu_alltoall
```

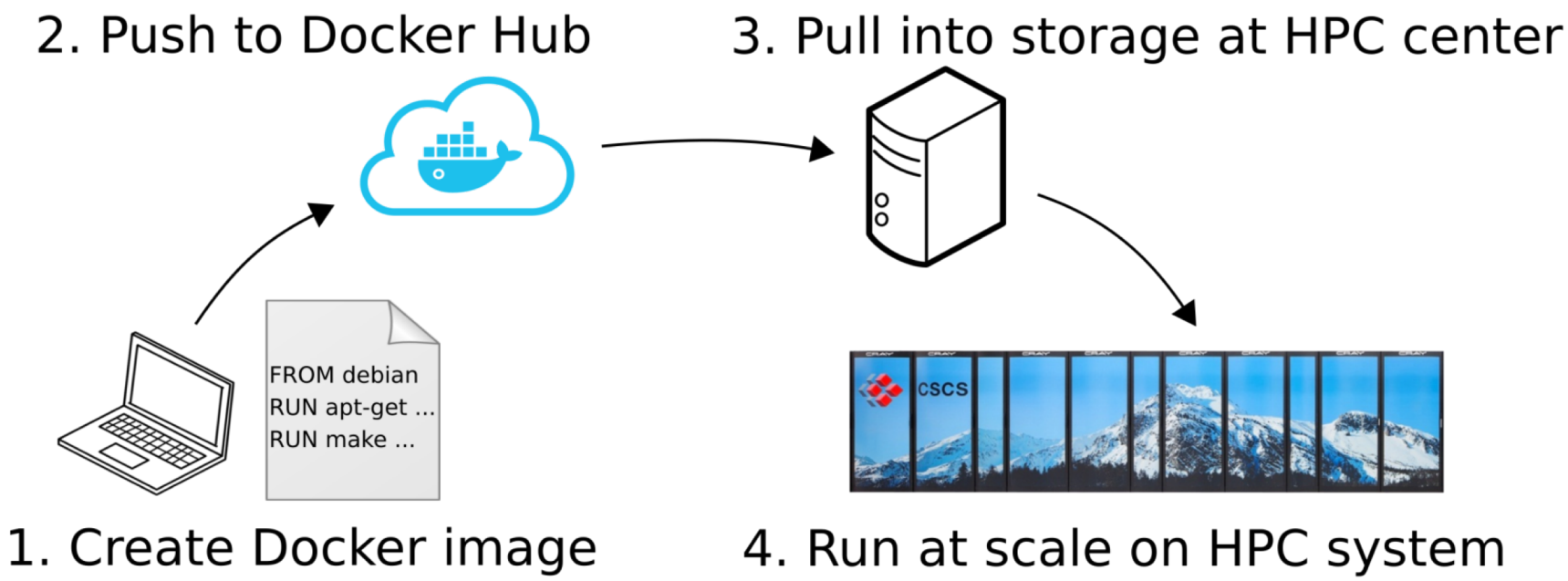


Outline



- Introduction
  - Linux Containers Ecosystem
  - Containers timeline @ CSCS
- Sarus
  - CSCS requirements
  - Sarus features
- OCI hooks
  - NVIDIA Container Toolkit and MPI hook
- **Use case**
  - **Intel MKL and MPI with NVIDIA Cuda**
- Performance tests
  - Gromacs, TensorFlow, Cosmo, QE

Typical user workflow



Create the image of Intel MKL and MPI with CUDA 10.1

Container based on Nvidia images at https://hub.docker.com/r/nvidia/cuda

- CUDA Toolkit 10.1
- Ubuntu 18.04

```
FROM nvidia/cuda:10.1-devel-ubuntu18.04
SHELL ["/bin/bash", "-c"]
WORKDIR /usr/local/src
RUN apt-get update && apt-get install cpio
COPY intel_licence_file.lic /opt/intel/licenses/USE_SERVER.lic
ADD intel19.01_silent.cfg .
ADD parallel_studio_xe_2019_update1_cluster_edition_online.tgz .
RUN cd parallel_studio_xe_2019_update1_cluster_edition_online \
    && ./install.sh --ignore-cpu –s /usr/local/src/intel19.01_silent.cfg
```

Runtime variables and library path

Final steps after installing Intel compiler and libraries:

- Intel script with runtime variables to be sourced before building apps
- ldconfig configuration file in /etc/ld.so.conf.d with custom library path

```
RUN echo -e "source /opt/intel/bin/compilervars.sh intel64 \nsource
/opt/intel/mkl/bin/mklvars.sh intel64 \nsource
/opt/intel/impi/2019.1.144/intel64/bin/mpivars.sh release" >
/etc/profile.d/intel.sh \    && echo -e "/opt/intel/lib/intel64
\n/opt/intel/mkl/lib/intel64 \n/opt/intel/impi/2019.1.144/intel64/lib
\n/opt/intel/impi/2019.1.144/intel64/lib/release
\n/opt/intel/impi/2019.1.144/intel64/libfabric/lib
\n/opt/intel/impi/2019.1.144/intel64/libfabric/lib/prov" >
/etc/ld.so.conf.d/intel.conf \     && ldconfig
```

Build the container

Create a container with your application from the Intel CUDA image:

- Use a login SHELL to source Intel scripts
- Run ldconfig to update the runtime library path
- Build your application inside the container and save it

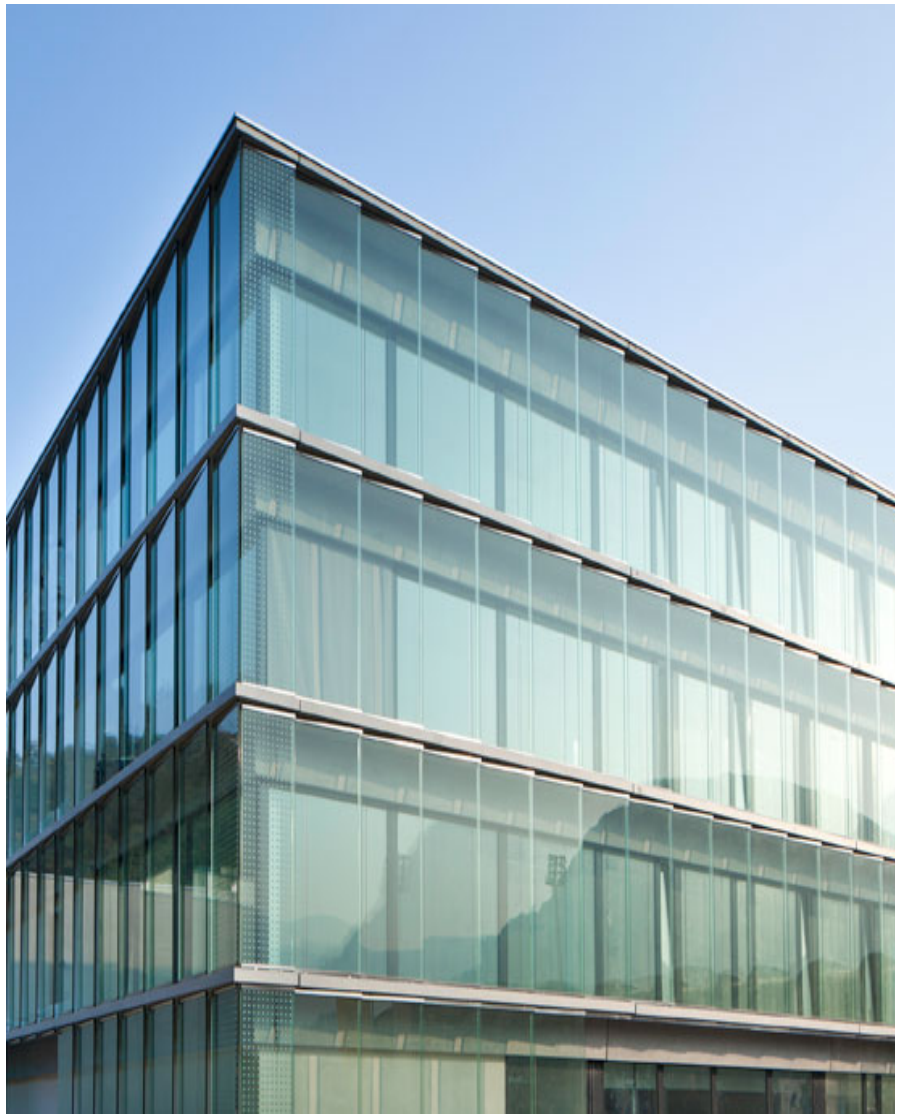
```
FROM intel:19.01-cuda10.1
SHELL ["/bin/bash", "--login", "-c"]
RUN ldconfig

RUN wget --no-check-certificate -q https://gitlab.com/QEF/q-e-gpu/-
/archive/qe-gpu-6.5a1/q-e-gpu-qe-gpu-6.5a1.tar.gz && tar xfq-e-gpu-qe-
gpu-6.5a1.tar.gz

[…]
```

Run the container

- Load the sarus modulefile available on the HPC system
- Create a tag for the image loaded within sarus
- Run the container on compute nodes using the workload manager

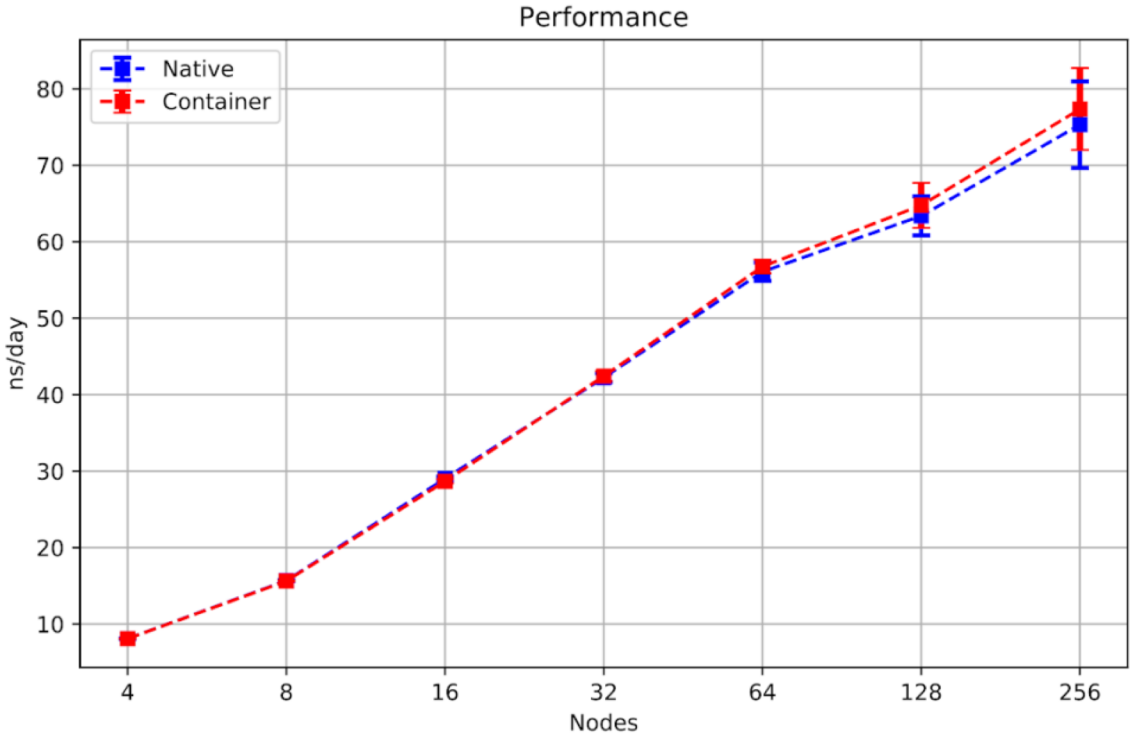
```
$ module load sarus
$ sarus load qe65a1-intel19.01-cuda10.1.tar qe65a1:intel19.01-cuda10.1

$ srun –nodes=16 --ntasks-per-node=1 sarus run --mpi  \
--mount=type=bind,source=$SCRATCH/qe/benchmarks,      \
destination=/usr/local/benchmarks                     \
load/library/qe65a1:intel19.01-cuda10.1               \
bash -c ‘cd /usr/local/benchmarks && pw.x --in input’
```

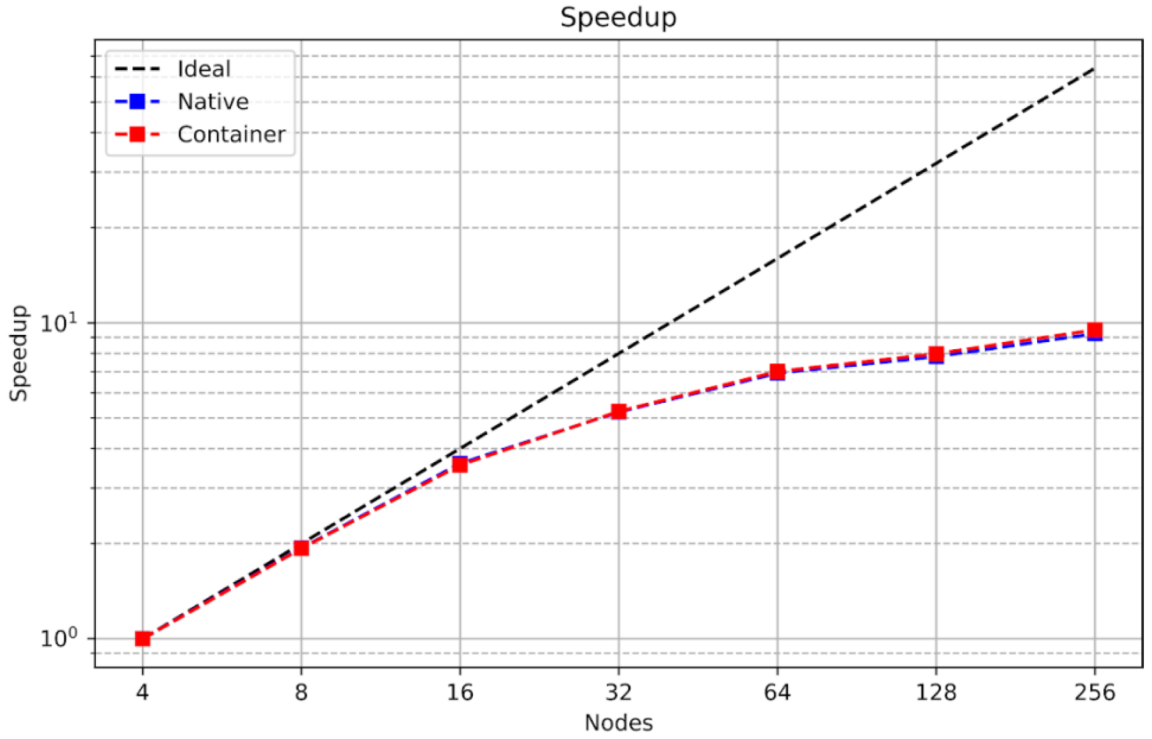
Outline



- Introduction
  - Linux Containers Ecosystem
  - Containers timeline @ CSCS
- Sarus
  - CSCS requirements
  - Sarus features
- OCI hooks
  - NVIDIA Container Toolkit and MPI hook
- Use case
  - Intel MKL and MPI with NVIDIA Cuda
- **Performance tests**
  - **Gromacs, TensorFlow, Cosmo, QE**

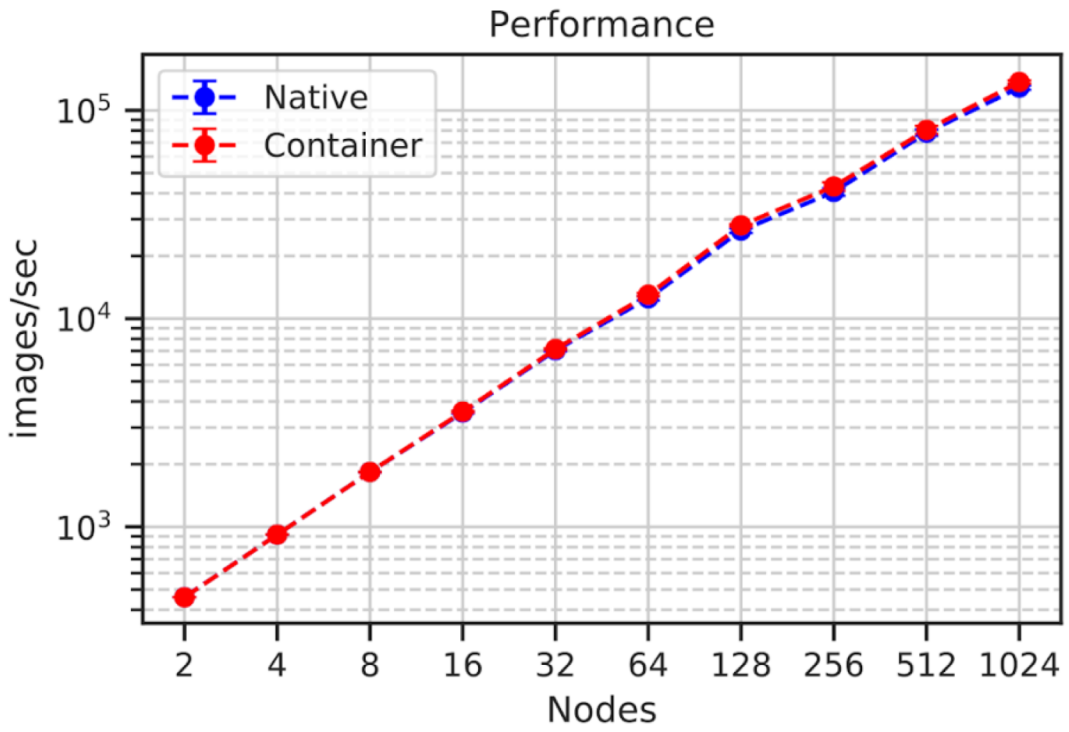
GROMACS: Classical Molecular Dynamics

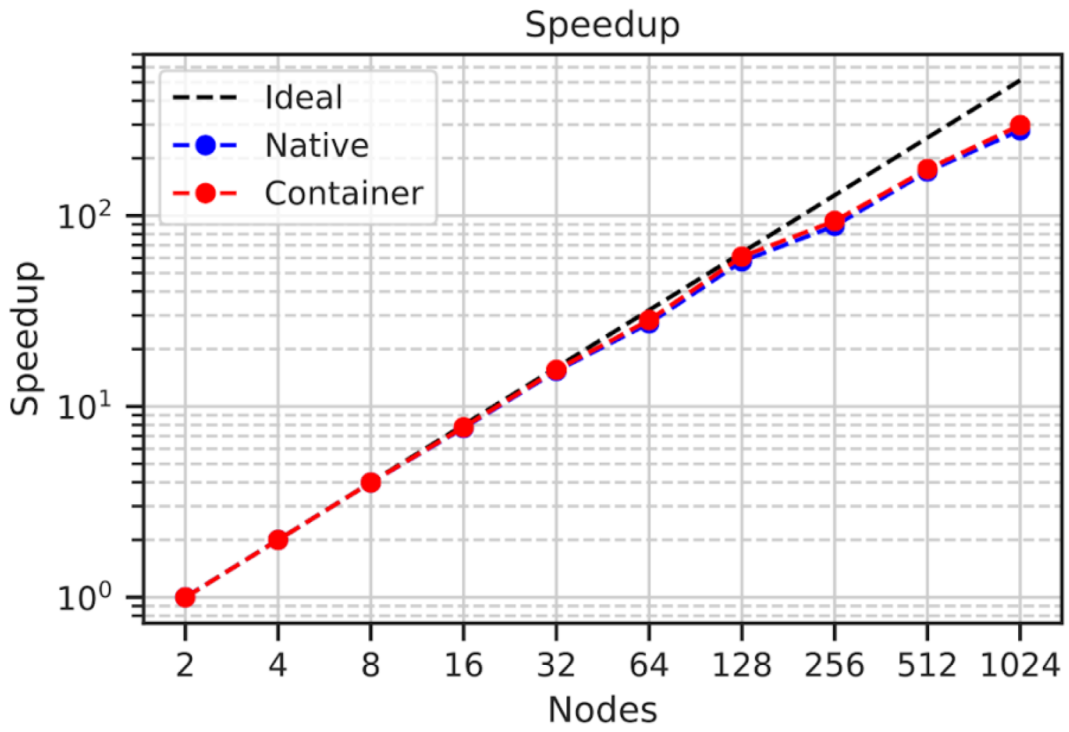




Software: GROMACS 2018.3, CUDA 9.1
Test case: PRACE Unified European Applications Benchmark Suite, GROMACS Test Case B
System: Piz Daint hybrid partition (Intel Xeon E5-2690 v3, NVIDIA Tesla P100, Cray Aries Interconnect)

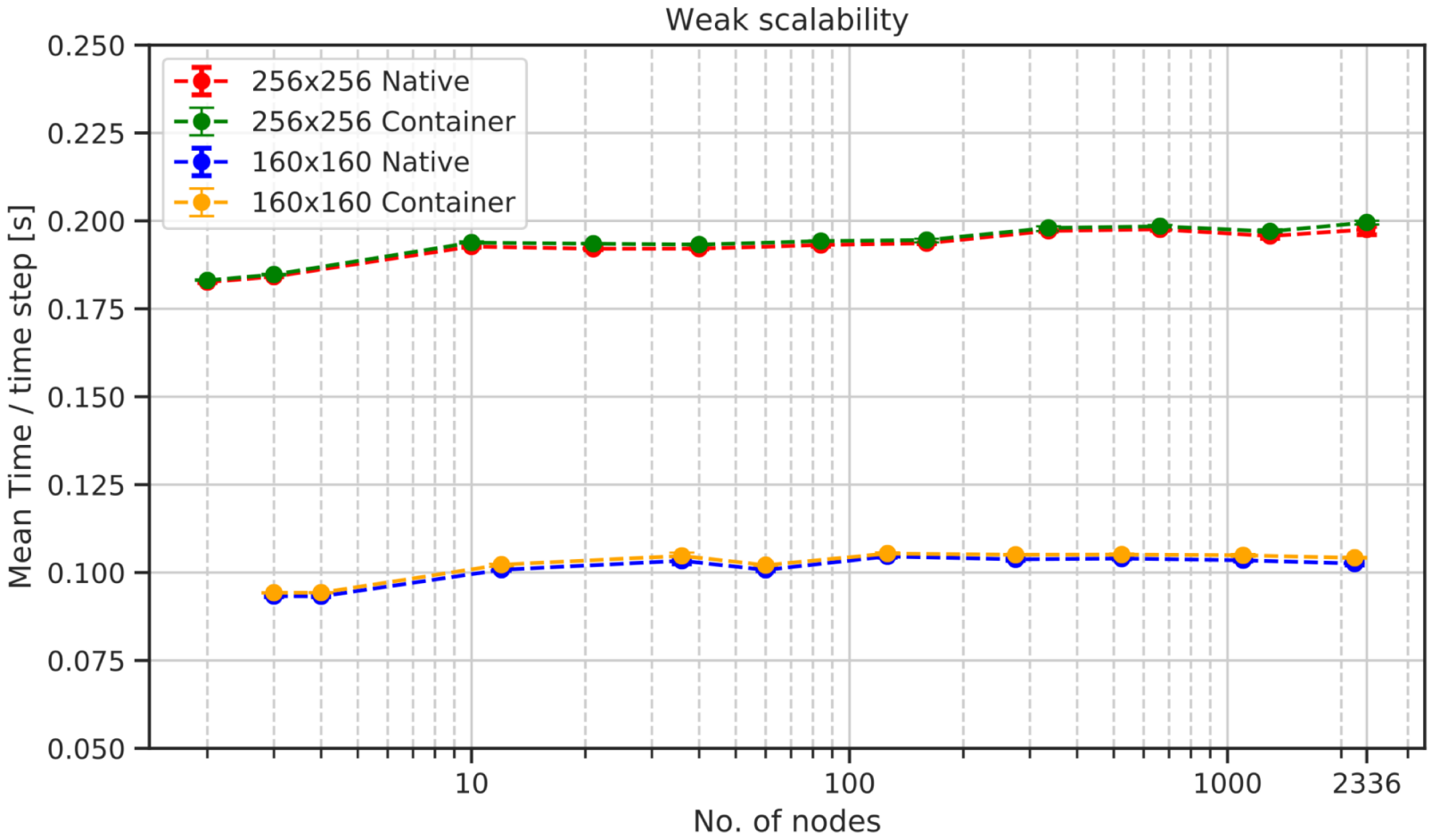
TensorFlow and Horovod: Deep Learning





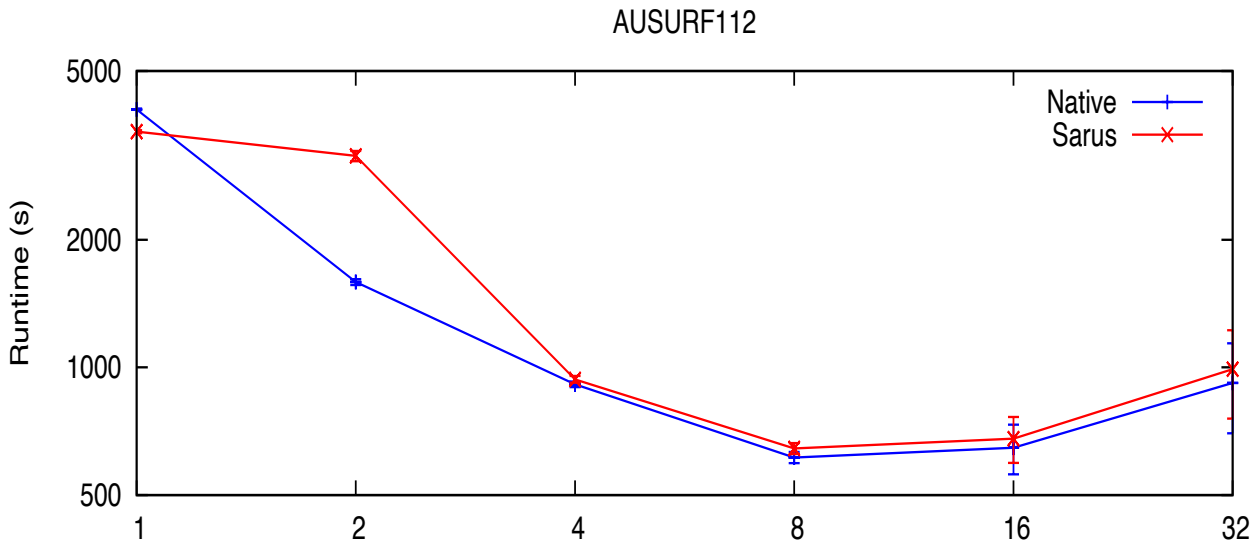
Software: TensorFlow 1.7.0, Horovod 0.15.1, CUDA 9.0
Test case: TF CNN Benchmark scripts, ResNet-50, synthetic ImageNet data
System: Piz Daint hybrid partition (Intel Xeon E5-2690 v3, NVIDIA Tesla P100, Cray Aries Interconnect)

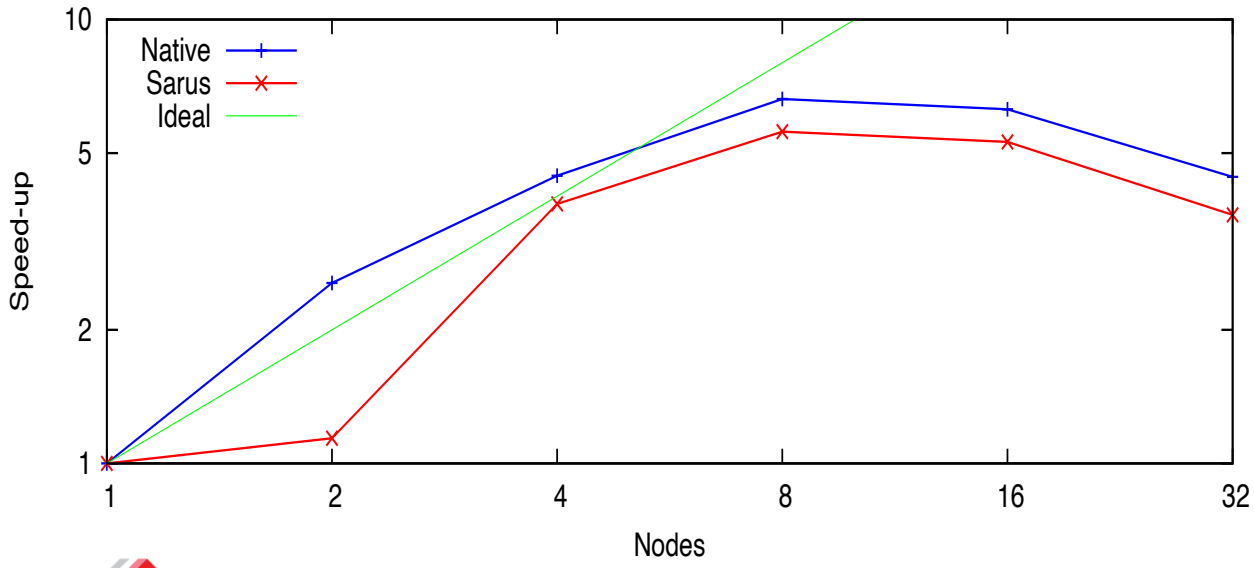
COSMO: Numerical Weather Forecast

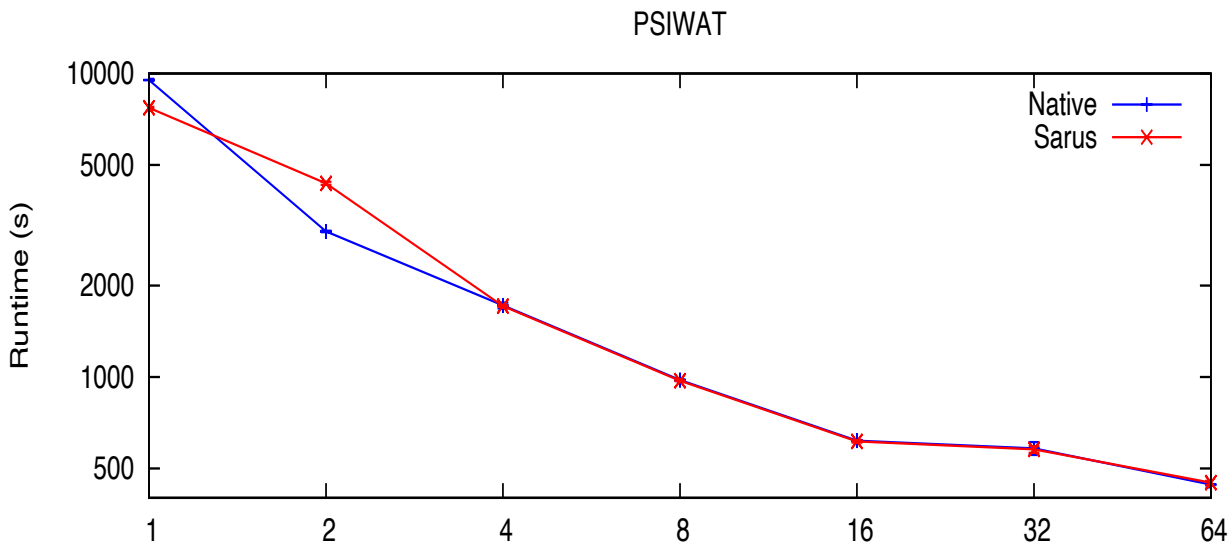


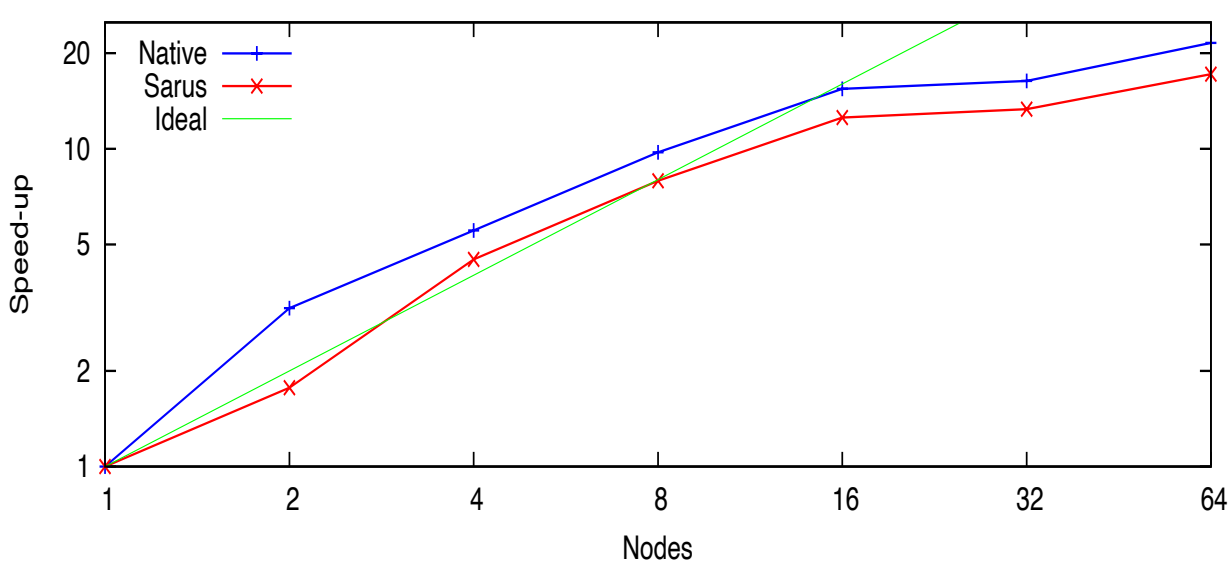
Software: COSMO 5.0, CUDA 9.1
Test case: Near-global idealized baroclinic wave
System: Piz Daint hybrid partition (Intel Xeon E5-2690 v3, NVIDIA Tesla P100, Cray Aries Interconnect)

QuantumESPRESSO: Materials Science

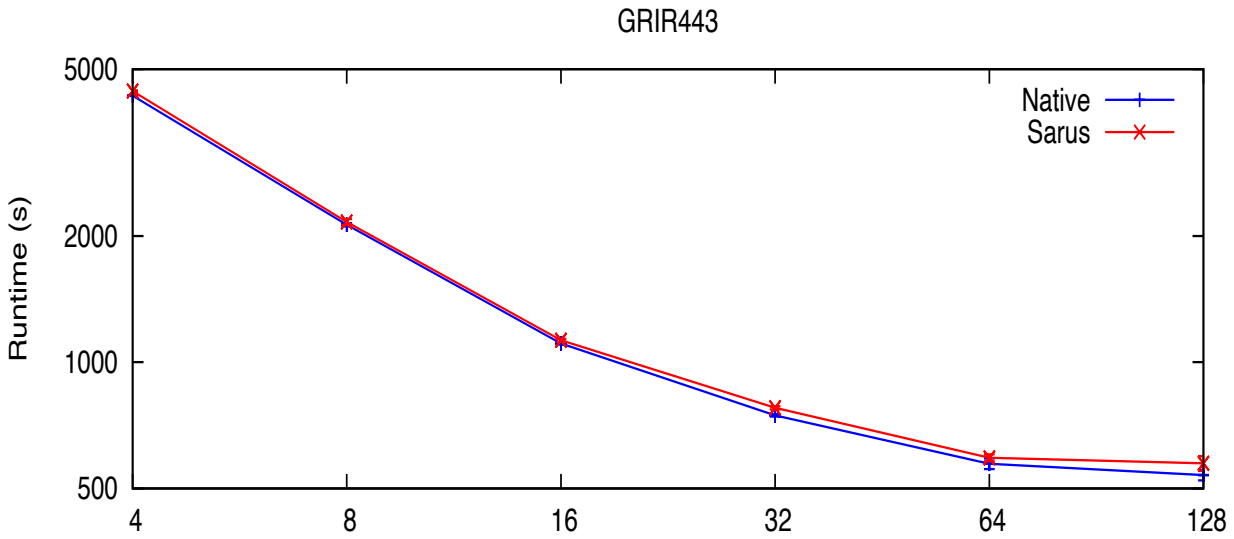


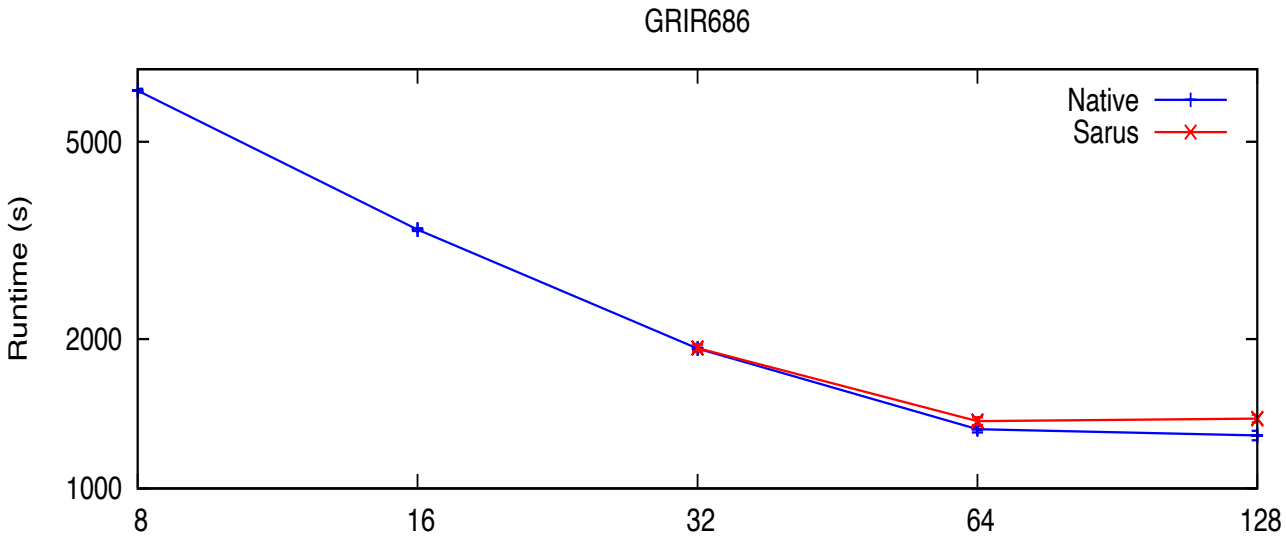


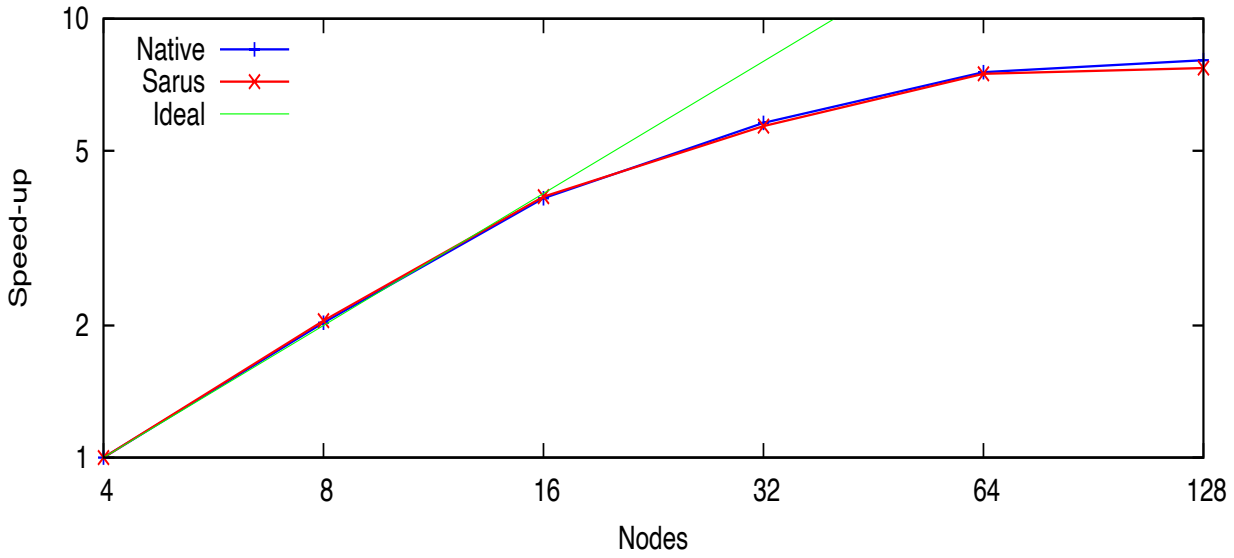


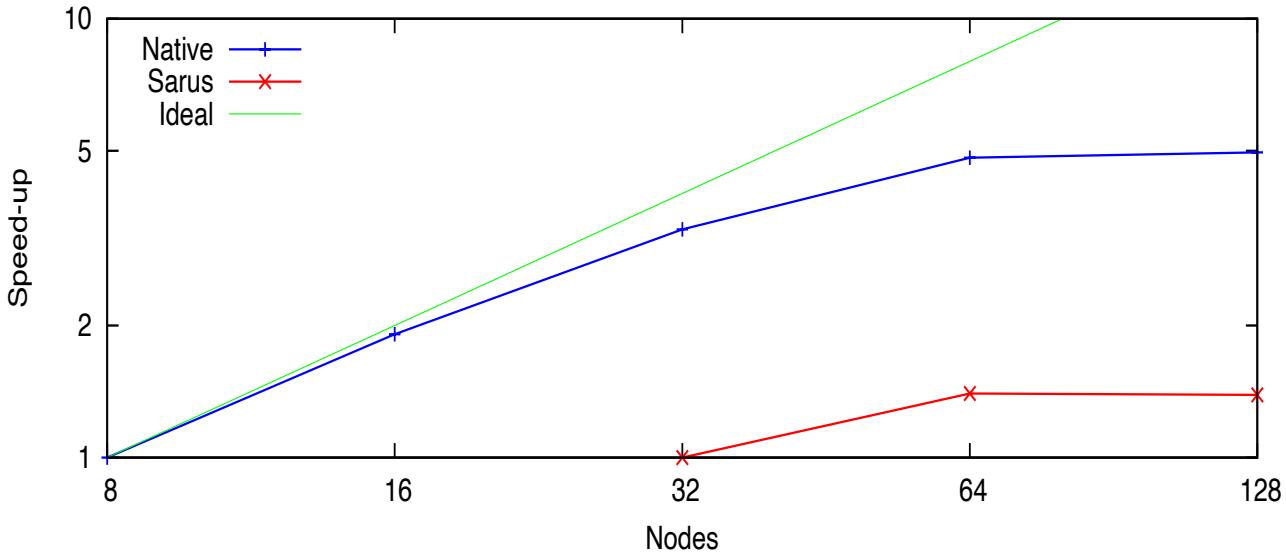


QuantumESPRESSO: Materials Science









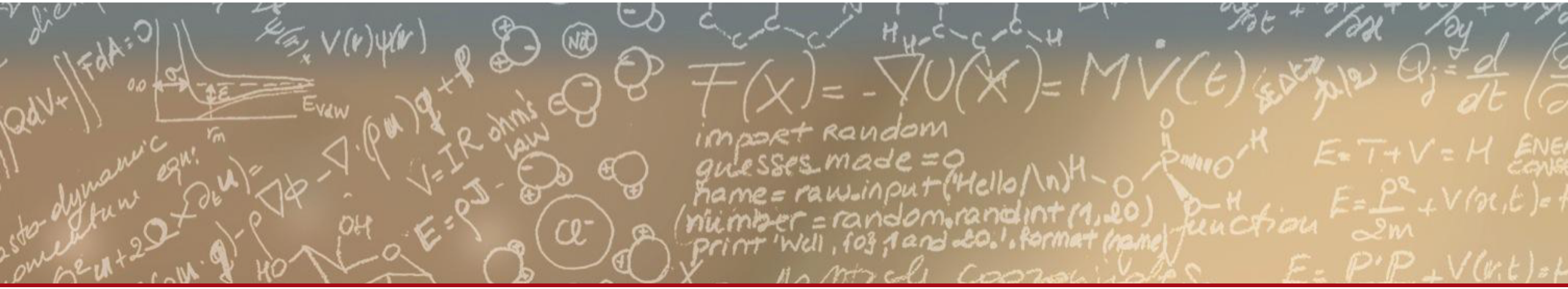
Conclusion

Sarus is a container engine for HPC system, compliant with open standards:

- Transparent native performance through hooks
- Consistent user experience with Docker: small learning curve
- Enables use of standard, open, upstream components on HPC systems
- Extensible architecture:
  - encourages vendor engagement
  - improves maintainability

Useful links

- Sarus source code on GitHub
  - https://github.com/eth-cscs/sarus
- Sarus User Documentation at CSCS
  - https://user.cscs.ch/tools/containers/sarus
- Sarus Quick Start install and user guide:
  - https://sarus.readthedocs.io/en/latest/quickstart/quickstart.html
  - Licensing: Sarus source code is freely distributed under a BSD 3-Clause license
- Sarus Official User Guide
  - https://sarus.readthedocs.io/en/latest/user/index.html
- Best practices for writing dockerfiles
  - https://docs.docker.com/develop/develop-images/dockerfile_best-practices



Thank you for your kind attention