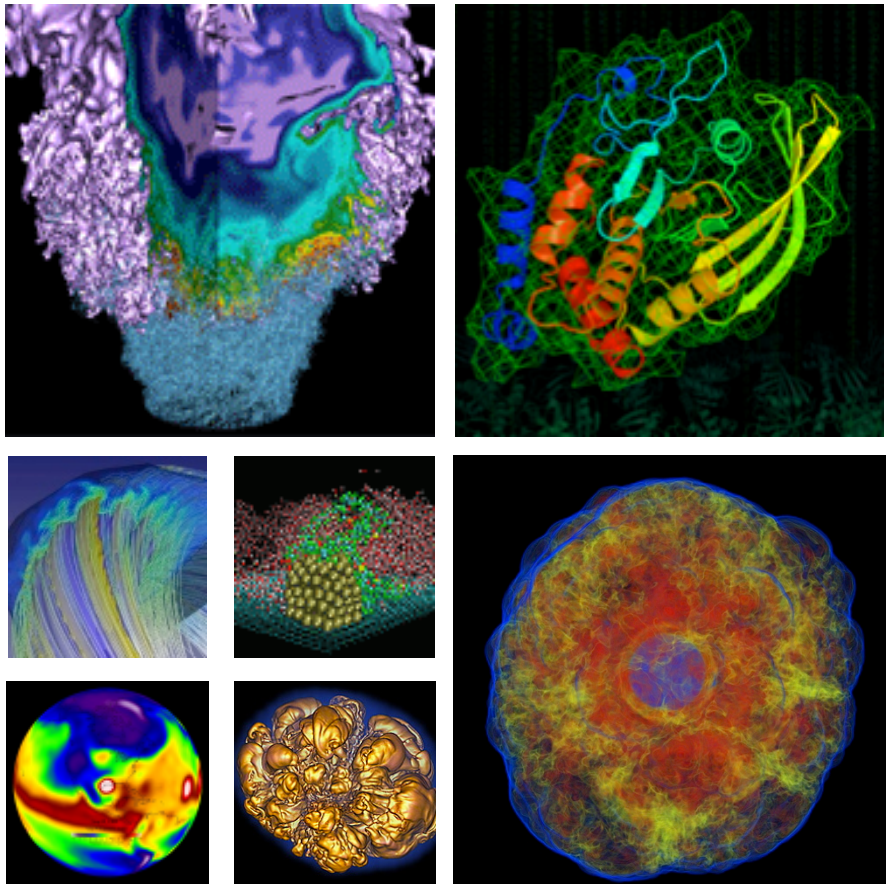
Shifter



Shane Canon

SC16 - Tutorial

Topics

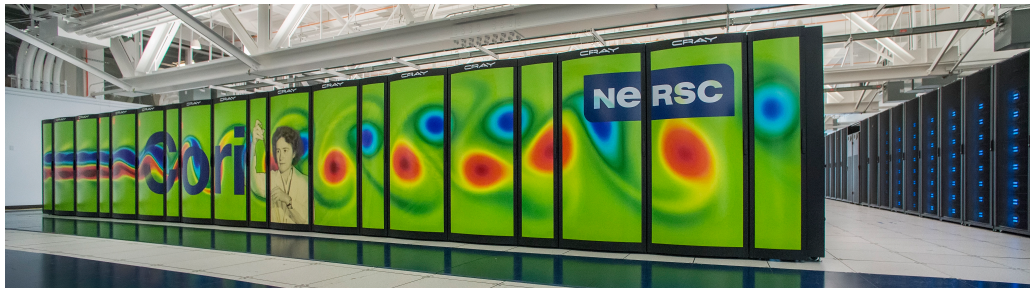
- Context - NERSC
- Shifter Architecture and Design
- Shifter in Action
- Discussion and Future Work

NERSC: DOE's Scientific Computing Center

- 6,000 active users, 750+ codes, 2000+ paper/year
- Biology, Energy, Environment
- Computing
- Materials, Chemistry, Geophysics
- Particle Physics, Cosmology
- Nuclear Physics
- Fusion Energy, Plasma Physics

HPC is Awesome

- **Cori Cray XC40**
  - Data-intensive (32-core Haswells, 128GB) partition
  - Compute-intensive (68-core KNLs, 90GB) partition
  - ~10k nodes, ~700k cores
- **Edison Cray XC30**
  - 2.5PF
  - 357TB RAM
  - ~5000 nodes, ~130k cores
- **High speed parallel file system**
  - >10 PB project file system (GPFS)
  - >38 PB scratch file system (Lustre)
  - >1.5 PB Burst Buffer (flash)
- **High Speed Aires interconnect**
  - Some speed numbers here



…But HPC Systems can be difficult

- **No local disk**
  - Breaks a lot of standard Linux work flows
- **Minimal OS**
  - Designed to accelerate parallel software
  - Many “expected” Linux tools are absent
  - Runs SUSE, and doesn’t upgrade often
- **Different file systems have different responses**
  - Sometimes unclear to users where is the best place to put their software
- **Many groups have turned to Shifter to over come these obstacles**

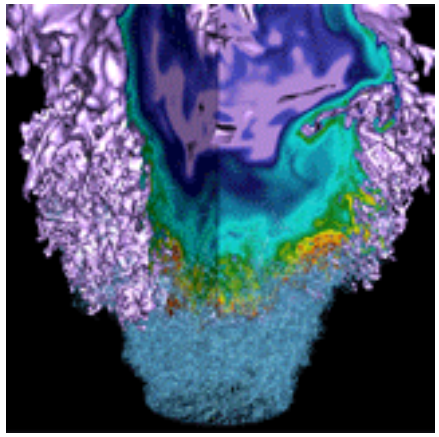
Why not just run Docker

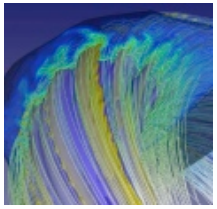
- System Architecture: Docker assumes local disk
- Security: Docker currently uses an all or nothing security model. Users would effectively have system privileges
- Integration: Docker doesn't play nice with batch systems.
- System Requirements: Docker typically requires very modern kernel
- Complexity: Running real Docker would add new layers of complexity

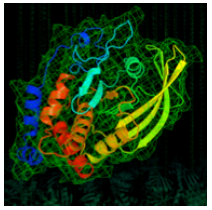
Solution: Shifter

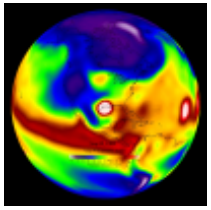
- **Partnership with Cray to design a solution to run containers on an HPC platform.**
- **Design Goals:**
  - User independence: Require no administrator assistance to launch an application inside an image
  - Shared resource availability (e.g., file systems and network interfaces)
  - Leverages or integrates with public image repos (i.e. DockerHub)
  - Seamless user experience
  - Robust and secure implementation

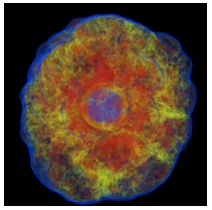
Implementation

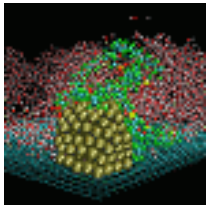






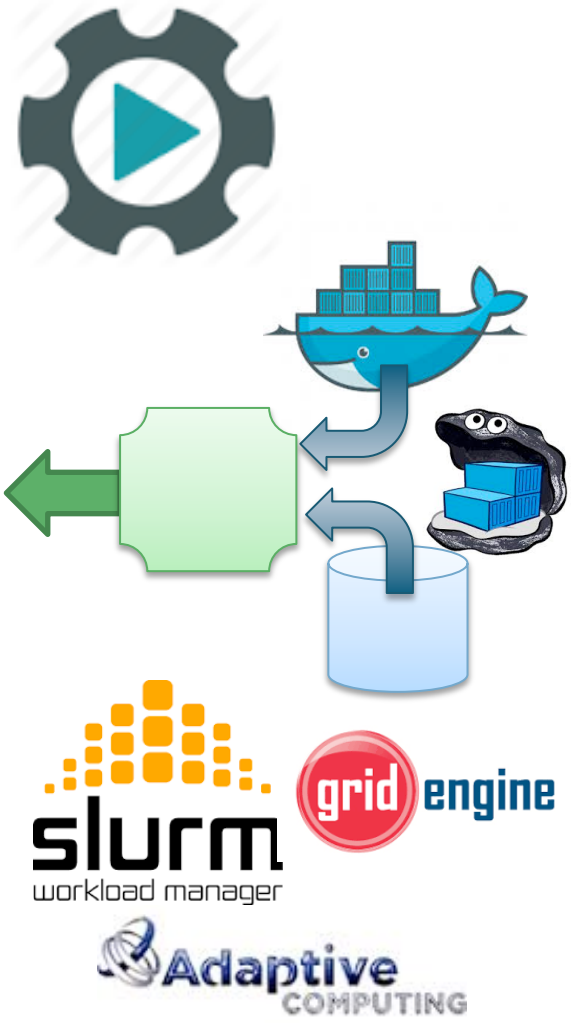




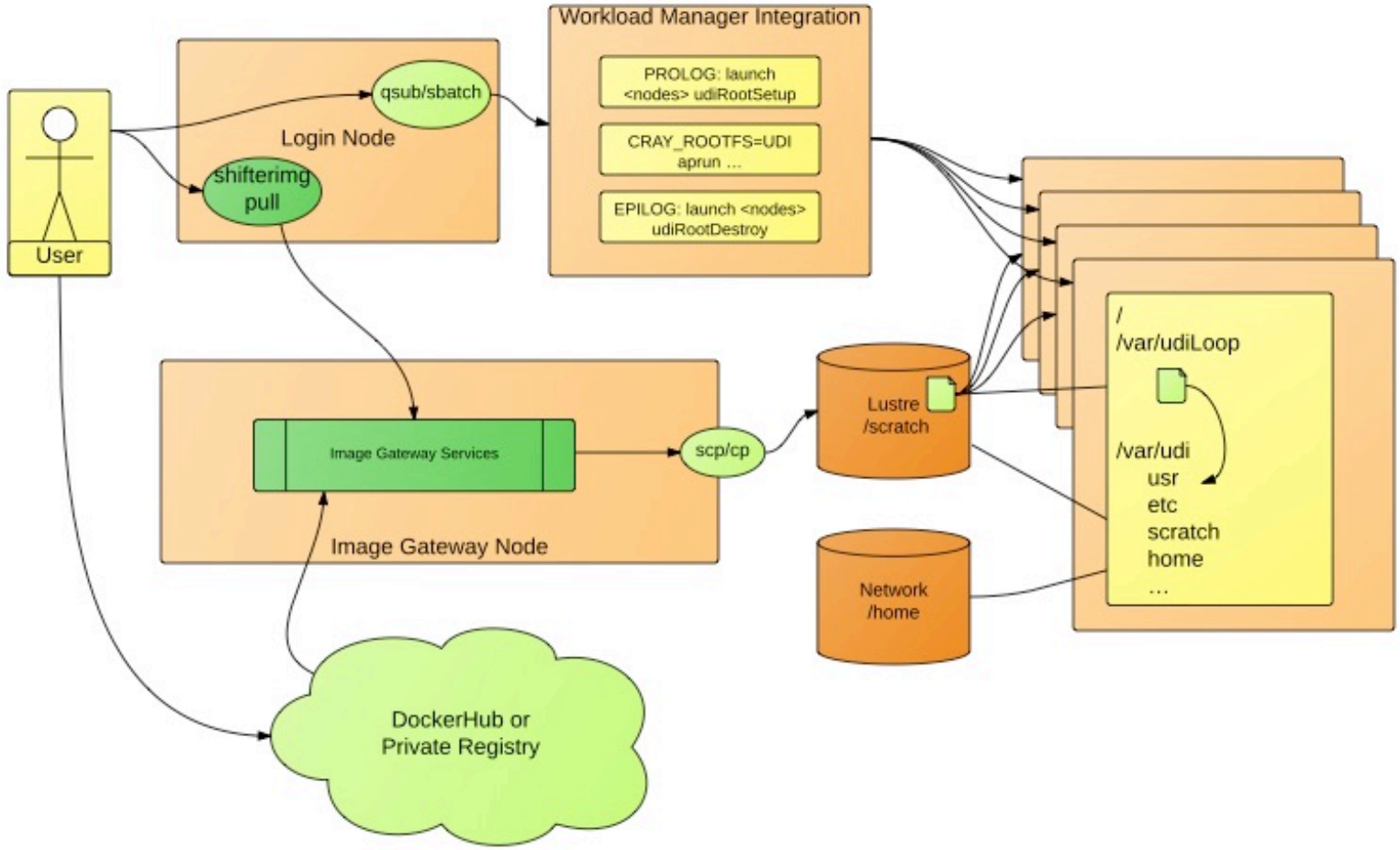


Shifter Components

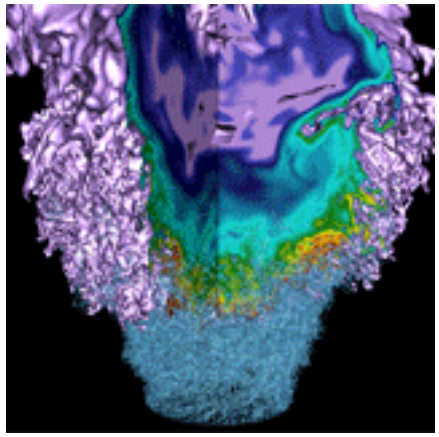
- **Shifter Image Gateway**
  - Imports and converts images from DockerHub and Private Registries
- **Shifter Runtime**
  - Instantiates images securely on compute resources
- **Work Load Manager Integration**
  - Integrates Shifter with WLM

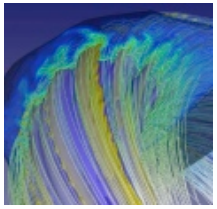


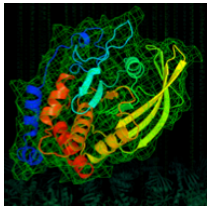
Shifter Architecture and Flow

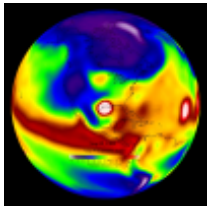


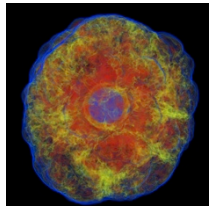
Shifter in Action

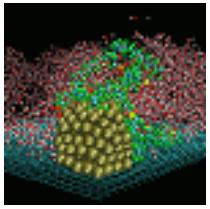




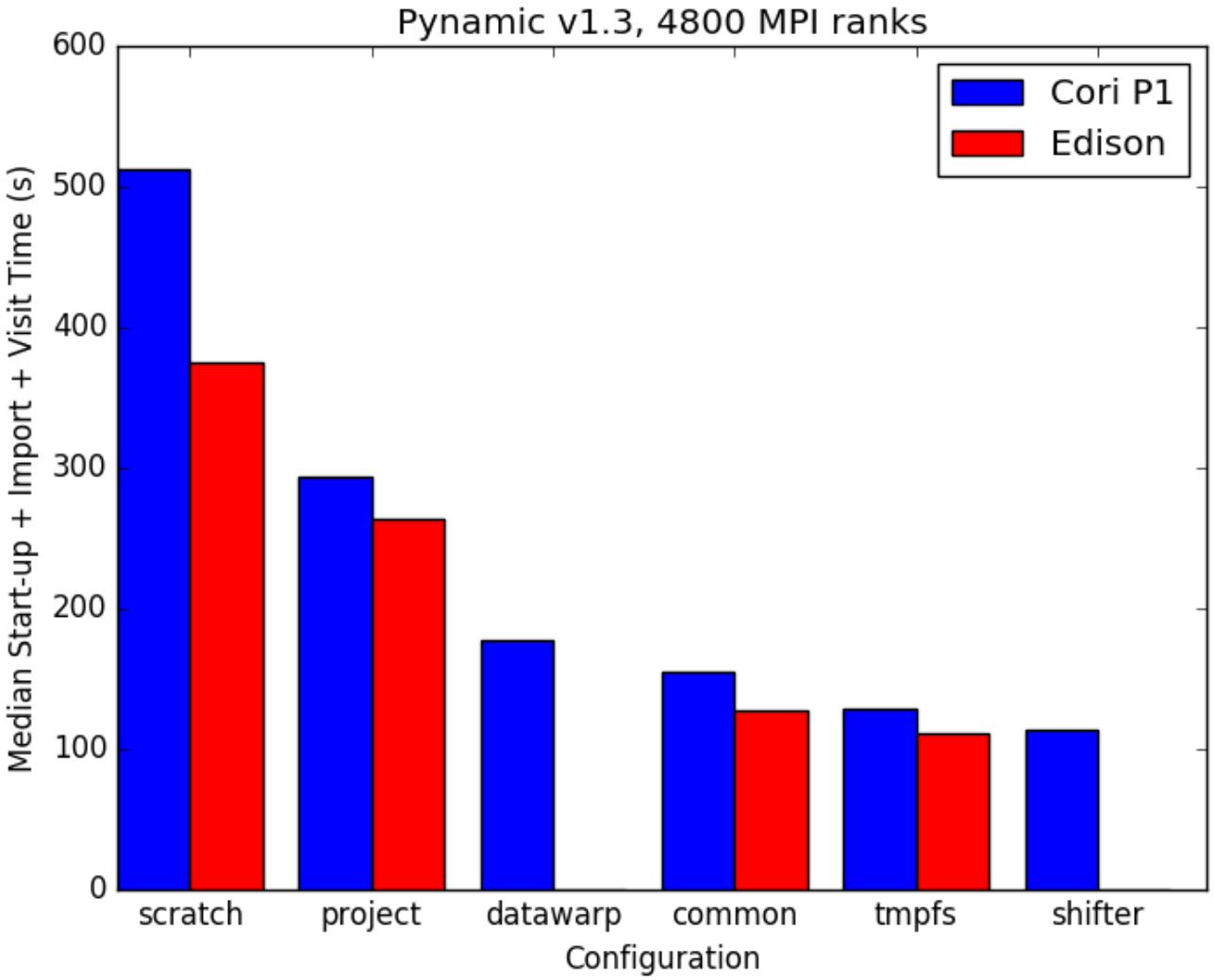








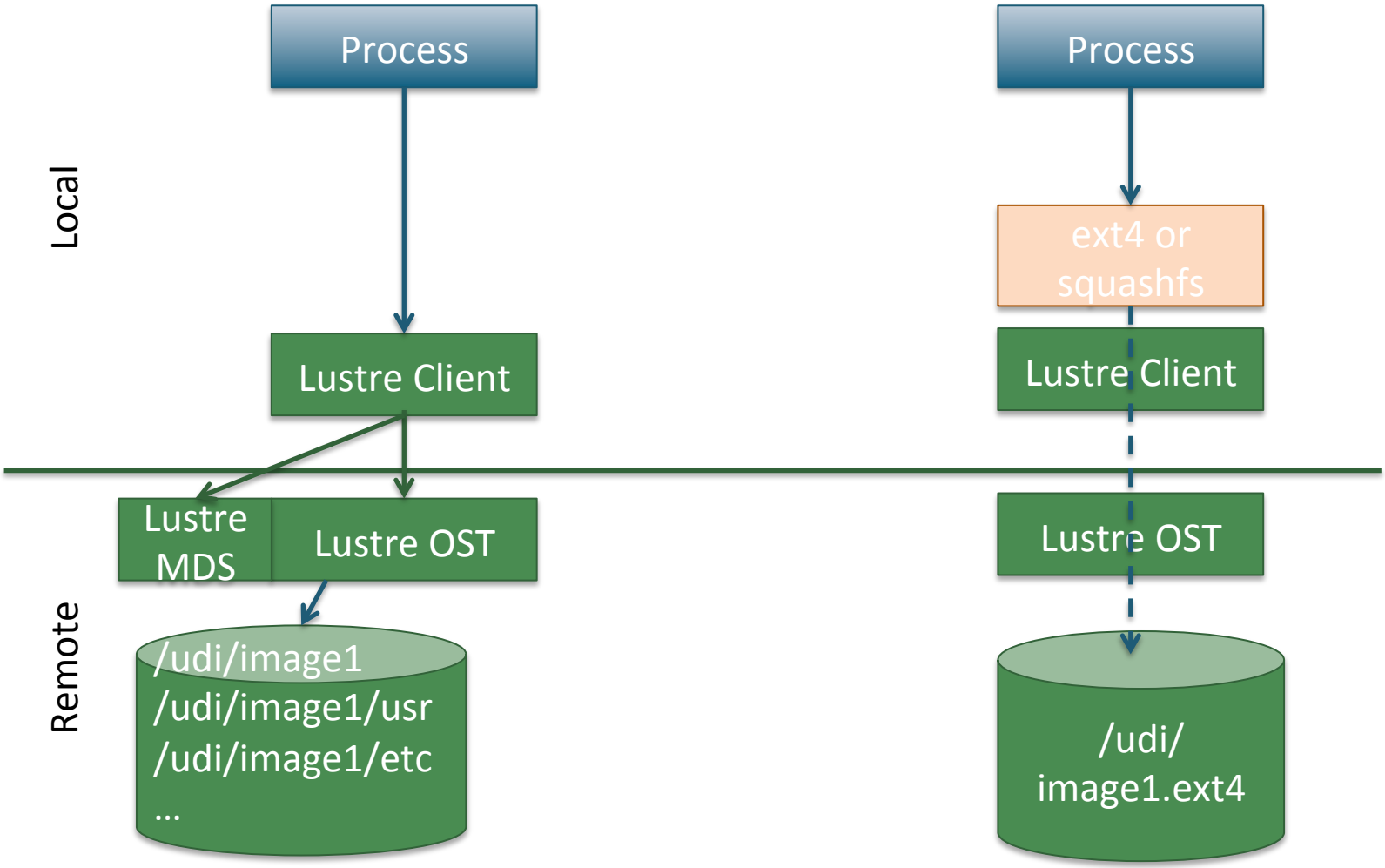
Shifter accelerates Python Apps



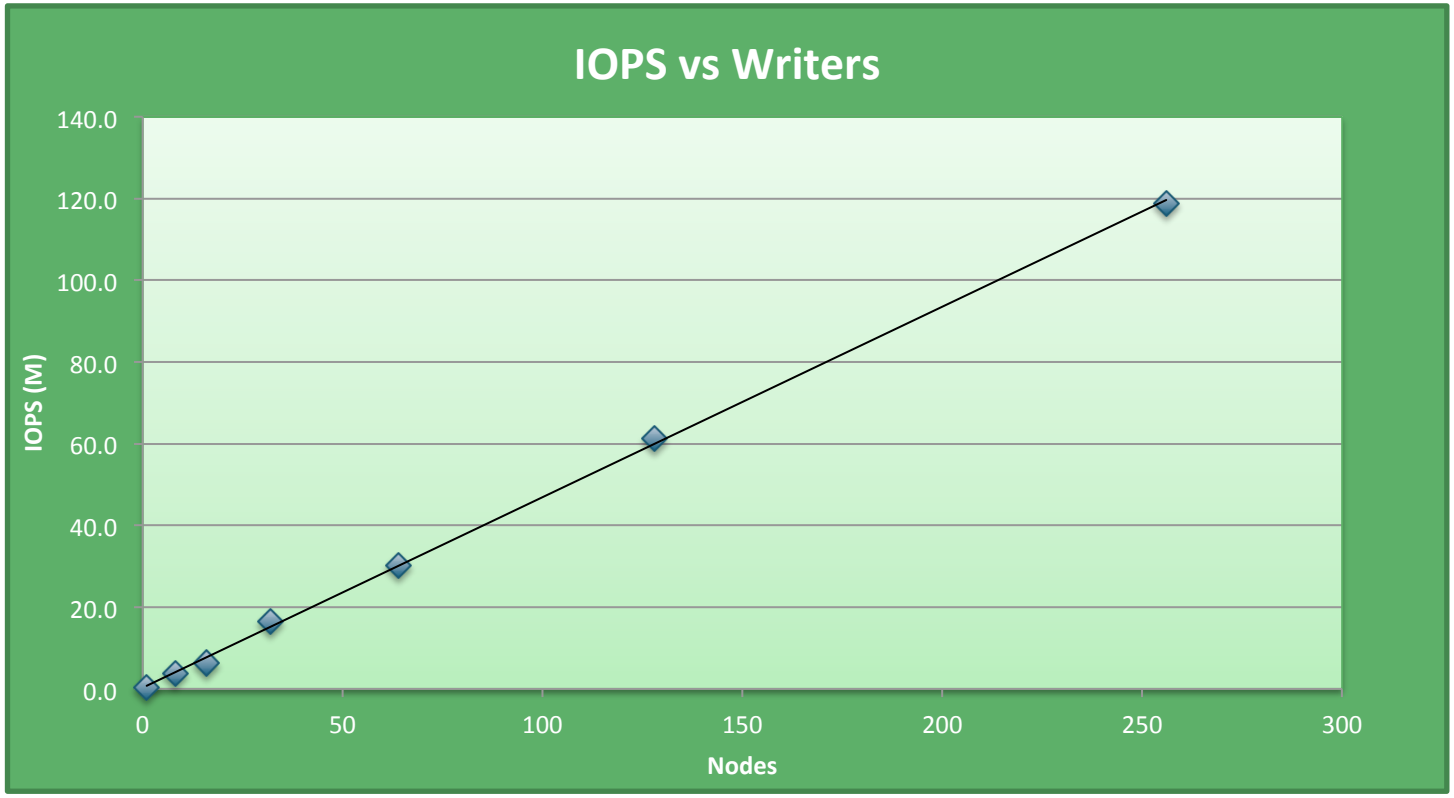
Why?

- Python must walk through the python libraries to construct the namespace
- Python must load up (read) any dynamic libraries that are required
- The loader must traverse the LD_LIBRARY_PATH to find the libraries to load
- Result: Lots of metadata accesses which put a load on the file system Metadata server

File System flow – Traditional vs Shifter



Per-Node Write Cache (IOPS)



Results of an IOR File per-process, 2 tasks per node, 512B transfer size, 2GB write. 100x faster than Lustre at the same scale.

Shifter and MPI

- **In Image**
  - Add required libraries directly into image.
  - Users would have to maintain libraries and rebuild images after an upgrade.
- **Managed Base Image (Golden Images)**
  - User builds off of a managed image that has required libraries.
  - Images are built or provided as part of a system upgrade.
  - Constrained OS choices and a rebuild is still required.
- **Volume Mounting**
  - Applications built using ABI compatibility.
  - Appropriate libraries are volume mounted at run time.
  - No rebuild required, but may not work for all cases.

Approach 1 – In the image

Dockerfile

```
FROM centos:6
## update packages and install dependencies
RUN yum upgrade -y && \
    yum -y install csh tar numpy scipy matplotlib gcc
WORKDIR /
## replace mpi4py with cray-tuned one
ADD optcray_cori.tar /
ADD mpi4py-1.3.1.tar.gz /usr/src
ADD mpi.cfg /usr/src/mpi4py-1.3.1/
RUN cd /usr/src/mpi4py-1.3.1 && \
    chmod -R a+rX /opt/cray && chown -R root:root /opt/cray && \
    python setup.py build && \
    export MPI4PY_LIB=$( rpm -ql $(rpm -qa | grep mpi4py | head -1) | egrep "lib$" ) && \
    export MPI4PY_DIR="${MPI4PY_LIB}/.." && \
    python setup.py install && \
    cd / && rm -rf /usr/src/mpi4py-1.3.1 && \
    echo "/opt/cray/wlm_detect/default/lib64/libwlm_detect.so.0" >>/etc/ld.so.preload && \
    (echo "/opt/cray/mpt/default/gni/mpich2-gnu/48/lib\n/opt/cray/pmi/default/lib64";\
     echo "/opt/cray/ugni/default/lib64\n/opt/cray/udreg/default/lib64";\
     echo "/opt/cray/xpmem/default/lib64\n/opt/cray/alps/default/lib64") \
     >> /etc/ld.so.conf && \
    ldconfig
```

```
> docker build –t scanon/myapp:1.1 .
> docker push scanon/myapp:1.1
```

Approach 2 - Golden Image

Dockerfile

```
FROM registry.services.nersc.gov/cori:latest

ADD . /app
RUN cd /app && \
    mpicc –o hello helloworld.c
```

```
> docker build –t scanon/myapp:1.1 .
> docker push scanon/myapp:1.1
```

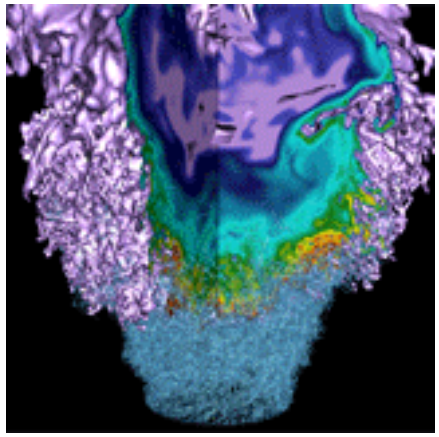
Approach 3 – Volume Mount

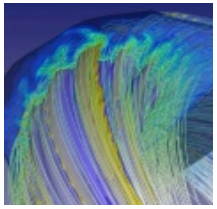
```
FROM nersc/mpi-ubuntu:14.04

ADD . /app
RUN cd /app && \
    mpicc –o hello helloworld.c
```

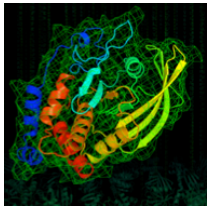
Dockerfile

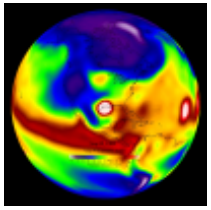
```
> docker build –t scanon/hello-vm:latest .
> docker push scanon/hello-vm:latest
```

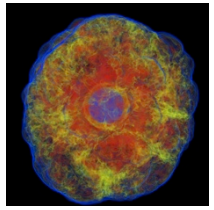
Discussion and Future Work

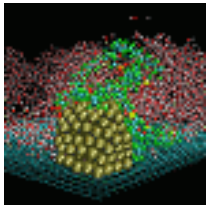












Why Users will like Shifter

Enables regular users to take advantage of Docker on HPC systems at scale.

This enables users to:

- **Develop an application on the desktop or laptop and easily run it on a cluster or Supercomputer**
- **Solve their dependency problems themselves**
- **Run the (Linux) OS of their choice and the software versions they need**

And…

- **Improves application performance in many cases**
- **Improves reproducibility**
- **Improves sharing (through sites like Dockerhub)**

How does Shifter differ from Docker?

Most Noticeable

- **Image read-only on the Computational Platform**
- **User runs as the user in the container – not root**
- **Image modified at container construction time (e.g. additional mounts)**

Less Noticeable:

- **Shifter only uses mount namespaces, not network or process namespaces**
- **Shifter does not use cgroups directly (integrated with the Workload Manager)**
- **Shifter uses individual compressed filesystem files to store images, not the Docker graph (slows down iterative updates)**
- **Shifter starts some additional services (e.g. sshd in container space)**

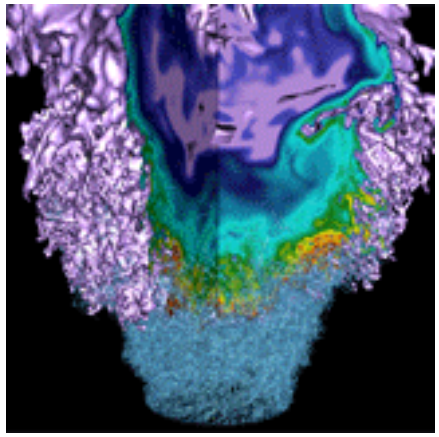
Roadmap

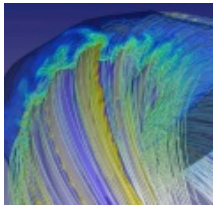
- **16.08 Release:**
  - Support for RHEL 6/7, SLES 11/12, Rhine/Redwood
  - RPM builds
  - Improved scaling
  - UI Improvements
  - Per-node write cache
  - Security and stability improvements
  - Image expiry and removal
- **17.01 Release (Goals)**
  - ACL support (private and authenticated images)
  - Support for multiple MPI stacks
  - Image usage statistics and metrics
  - Overlayfs support (stretch)
  - Debian packages for Ubuntu LTS
  - Better Docker compability (e.g. WORKDIR, ENTRYPOINT)

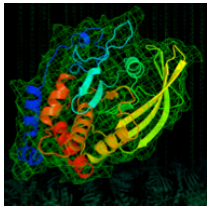
Conclusions

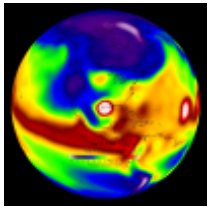
- **Shifter enables HPC systems to run Containers easily and at large scale**
- **Shifter provides the flexibility of Docker without sacrificing security, scalability or performance.**
- **Shifter opens the door to the many benefits of Docker including easy sharing of images, reproducibility, etc.**

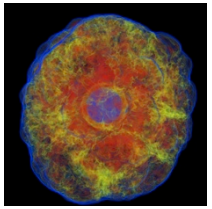
Hands On

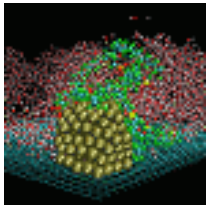












Getting Started

- **Log into NERSC using your training account**

```
laptop# ssh <trainXX>@cori.nersc.gov
```

- **Run a basic Shifter command**

```
cori# shifterimg images
```

Create an MPI Docker image

- See the helloworld examples in the github repo

```
laptop# editor hello.c
```

- Build the image and push it to DockerHub

```
laptop# docker push \
    <dockername>/hellompi:latest
```

- Pull the image to Cori with shifterimg

```
cori# shifterimg pull \
    <dockername>/hellompi:latest
```

Run the image interactively with shifter

- **Use salloc to get an interactive batch job**

```
cori# salloc -N 1 -C haswell
```

- **Once a prompt has appeared run the shifter command**

```
nid# shifter --image=<earlier image>
```

- **Confirm you are in the container**

```
$ lsb_release -a
```

Run a batch job

- **Create a batch script from the github repo example**

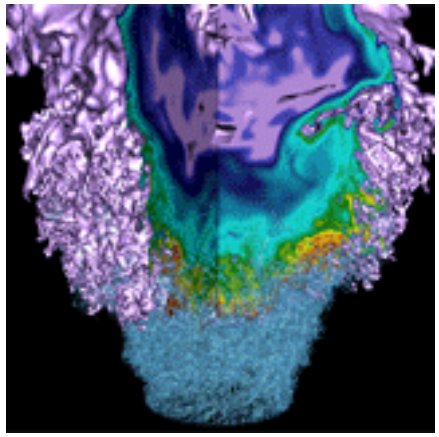
- **Submit the job using sbatch**

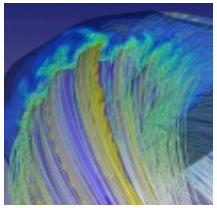
```
cori# sbatch ./batch.sl
```

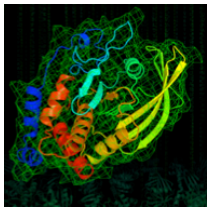
Hello World

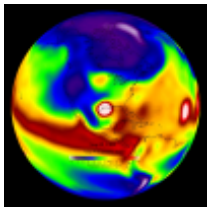
```
DOE6903508:shifter canon$ cat helloworld.c
// Hello World MPI app
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int size, rank;
    char buffer[1024];

    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    gethostname(buffer, 1024);

    printf("hello from %d of %d on %s\n", rank, size, buffer);

    MPI_Barrier(MPI_COMM_WORLD);

    MPI_Finalize();
    return 0;
}
```

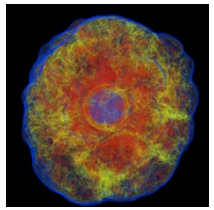
Screen Shots for Backup

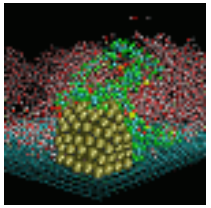












Dockerfile

```
DOE6903508:shifter canon$ cat Dockerfile
# This example makes use of an Ubuntu-based NERSC base image
# that already has MPI built and installed.
#
# This means the you just need to add your app code in and compile it.
#
# To build this example do:
# docker build -t <dockerhubid>/hellompi:latest .
#
# And to test:
# docker run -it --rm <dockerhubid>/hellompi:latest /app/hello

FROM nersc/ubuntu-mpi:14.04

ADD helloworld.c /app/

RUN cd /app && mpicc helloworld.c -o /app/hello

ENV PATH=/usr/bin:/bin:/app:/usr/local/bin
```

Building and Pushing

```
DOE6903508:shifter canon$ docker build -t scanon/hellompi .
Sending build context to Docker daemon 4.096 kB
Step 1 : FROM nersc/ubuntu-mpi:14.04
 ---> a71176e6ce4e
Step 2 : ADD helloworld.c /app/
 ---> 3ebe2000ff1b
Removing intermediate container c88799e39fa2
Step 3 : RUN cd /app && mpicc helloworld.c -o /app/hello
 ---> Running in f3ec8a1c55f3
 ---> 68491ca5b944
Removing intermediate container f3ec8a1c55f3
Step 4 : ENV PATH /usr/bin:/bin:/app:/usr/local/bin
 ---> Running in eef831fe8c61
 ---> 132ba04bff6d
Removing intermediate container eef831fe8c61
Successfully built 132ba04bff6d
DOE6903508:shifter canon$ docker push scanon/hellompi
The push refers to a repository [docker.io/scanon/hellompi]
2c96e3a94e72: Pushed
b0da9b74490f: Pushed
2a291a08dc8c: Mounted from scanon/hello
04cb0f3619cf: Mounted from scanon/hello
c410e650f359: Mounted from scanon/hello
bab10a362750: Mounted from scanon/hello
787a9151f9ae: Mounted from scanon/hello
470641744213: Mounted from scanon/hello
latest: digest: sha256:9225f1ce3fc3f6a644306899019df55e7632bc9a2e2f5a0a46e933b810a32805 size:
1990
```