SAND2021-14558PE

Sandia National Laboratories

Exceptional service in the national interest

CONSTRUCTING CONTAINERS FOR EXASCALE COMPUTING

RED HAT @ SC21

PRESENTED BY

ANDREW J. YOUNGE

SAND2021-XXXX C

HPC AND CLOUD CONVERGENCE?

"I can't run my national security workloads in the cloud"

"The cloud is still not up to the same performance as leadership-HPC"

"They don't support MPI"

"I couldn't find the `sbatch` command"

Many emerging "HPC" workloads being developed today target cloud deployments

- Deep learning frameworks and models
- Distributed memory apps outside of HPC, using whole new models
- Most/all of these tools can & are being deployed using containers

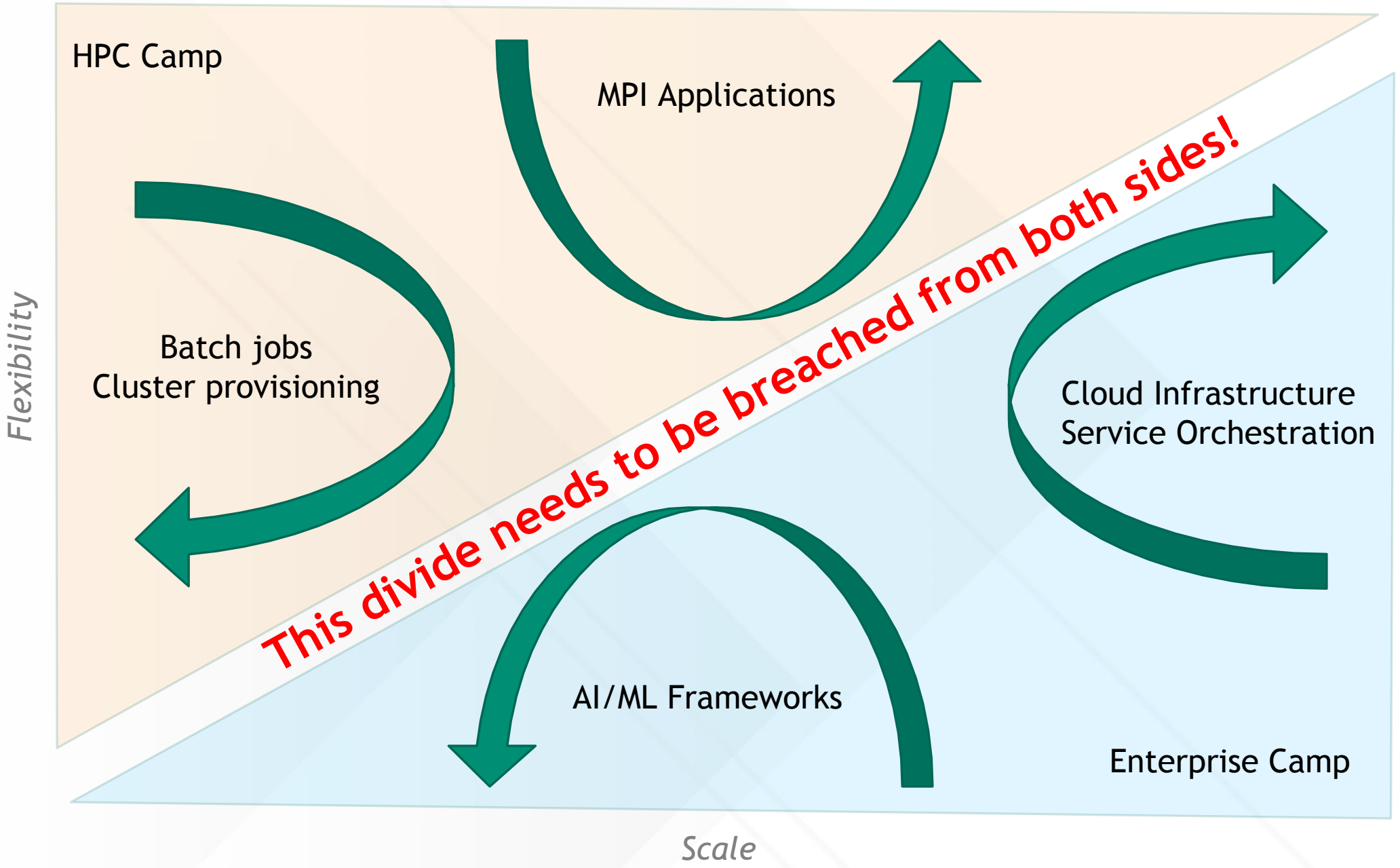
Cloud usage models are far more adaptable

- It is a service model, but you can deploy your own resource management system too
- Failover & service guarantees
- Deploy any software you want

Yet the need for leadership-class supercomputing has not decreased

Does the Exascale era provide us with an opportunity to increase HPC flexibility & interoperability?

TRENDS ARE NON-INTERSECTING



ECP SUPERCONTAINERS

Joint DOE effort - LANL, LBNL, LLNL, Sandia, U. of Oregon

Ensure container runtimes will be scalable, interoperable, and well integrated across DOE

- Enable container deployments from laptops to Exascale
- Assist Exascale applications and facilities leverage containers most efficiently

Three-fold approach

- Scalable R&D activities
- Collaboration with related ST and AD projects
- Training, Education, and Support

Activities conducted in the context of interoperability

- Portable solutions
  - Optimized E4S container images for each machine type
  - Containerized ECP that runs on Astra, A21, El-Capitan, …
- Work for multiple container implementations
  - Not picking a “winning” container runtime
- Multiple DOE facilities at multiple scales

SUPERCONTAINERS

ECP

EXASCALE COMPUTING PROJECT

DOCKER DOES NOT FIT HPC

Docker is not a good fit for HPC

- Docker privileges functionally equivalent to root-level access
  - A no-go for HPC's shared resource models
  - No good way to "fix" this within the docker runtime itself
- Based on client/daemon architecture
  - Great for the cloud/services world, but terrible for HPC
  - Daemons are unpredictable and can be source of significant noise
    - "The Case of the Missing Supercomputer Performance" @ LANL circa 2003

Building with Docker on your laptop/workstation is fine

- Assuming you own that machine
- But is your laptop's hardware anything like your supercomputer?

HPC CONTAINER RUNTIMES TODAY

- Several different container runtime options in HPC
- Shifter
- Singularity
- Charliecloud
- Sarus

SARUS (not official logo)

- All these HPC container runtimes are usable in HPC today
- Each runtime offers different designs and OS mechanisms
  - Storage & mgmt of images
  - User, PID, Mount namespaces
  - Security models
  - Image signing, validation, registries, etc

CONTAINER RUNTIMES SUPPORT ON DIFFERENT (PRE)EXASCALE SYSTEMS

ALCF

- Theta: Singularity
- Aurora: Singularity

OLCF

- Summit: Singularity (trial)
- Frontier: Singularity

NERSC

SHIFTER

- Cori: Shifter
- Perlmutter: Shifter & Podman

LLNL

- Sierra/Lassen: Singularity (trial)
- Linux clusters: Singularity
- El Capitan: Singularity & Podman

LANL

Charliecloud

- Trinity: Charliecloud
- Linux clusters: Charliecloud
- Crossroads: Charliecloud

Sandia

Charliecloud

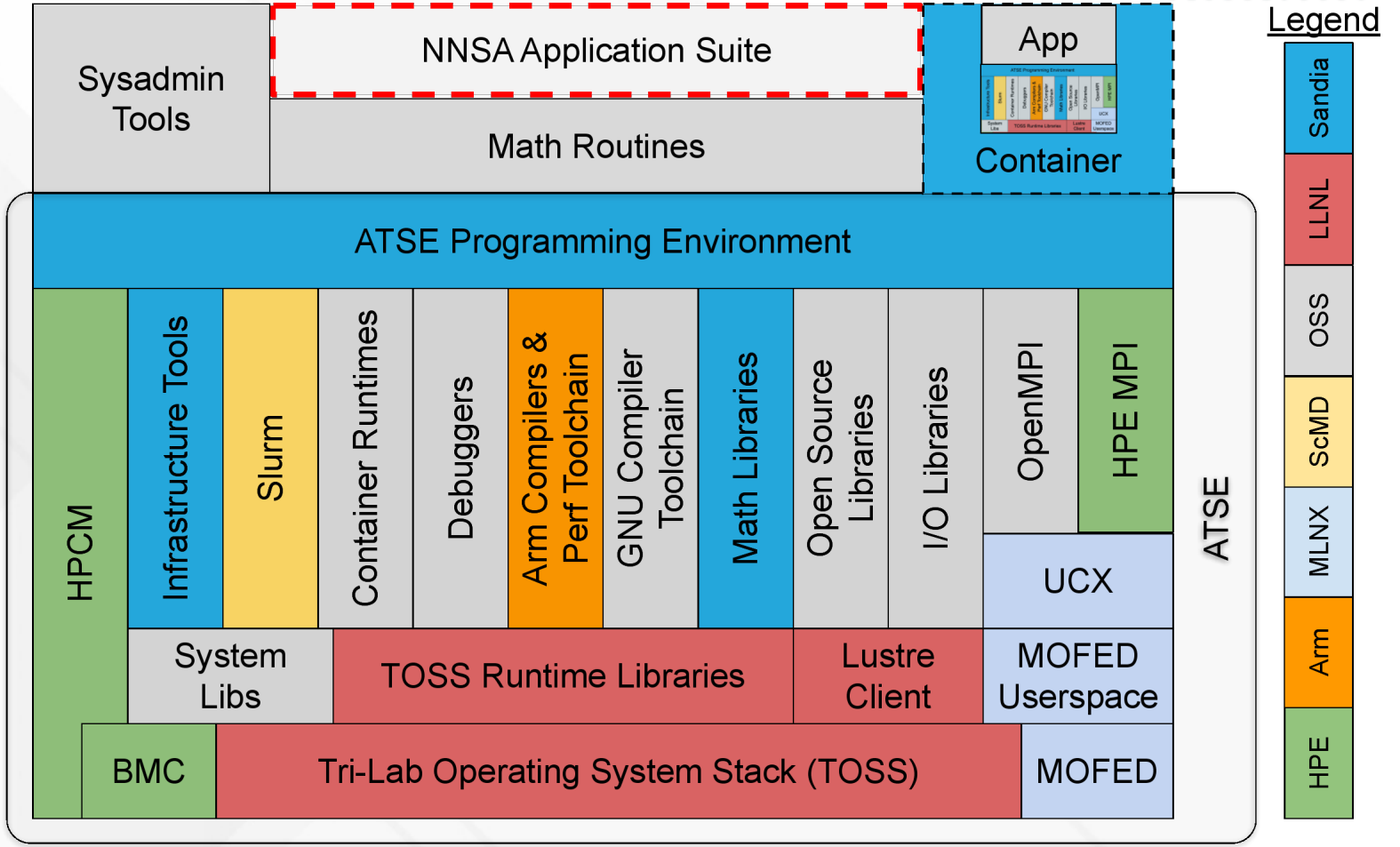
- Astra: Singularity, Charliecloud, & Podman
  Linux clusters: Shifter & Singularity

Many sites are rolling out container runtimes for users.
We are developing resources to facilitate consistent, performant deployment across sites.

But what about *building* containers for HPC??

ATSE: ADVANCED TRI-LAB SOFTWARE ENVIRONMENT

- ATSE is a collaboration with HPE, OpenHPC, and ARM
- Many pieces to the software stack puzzle
- HPE's HPC Software Stack
  - HPE Cluster Manager
  - HPE MPI (+ XPMEM)
- Arm
  - Arm HPC Compilers
  - Arm Math Libraries
  - Arm Allinea Tools
- Open source tools - OpenHPC
  - Slurm, OpenMPI, etc.
- Mellanox-OFED & HPC-X
- RedHat 7.x for aarch64 – TOSS



PODMAN FOR UN-PRIVILEGED CONTAINER BUILDS

Cannot build a container for Astra from my laptop

Need to build containers directly on the supercomputer
- Doing so requires root level privs
- root in HPC is bad, Docker is root equivalent

Leverage user namespaces for _building_ containers

Podman and Buildah provide container builds while maintaining user-level permissions
- Podman is CLI equivalent to Docker
- User namespaces
- Set uid/gid mappers
- TBD Overlay & FUSE for mount

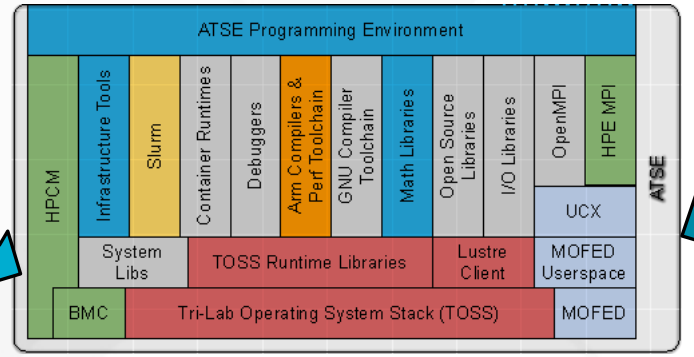
Ongoing Collaboration with

RedHat & Singularity folks

salloc -N 2048 && mpirun -np $NP singularity exec atse-astra-1.2.4.sif /app

singularity build atse-astra-1.2.4.sif docker://gitlab.doe.gov/atse/astra:1.2.4

podman push gitlab.doe.gov/atse/astra:1.2.4



podman build -t “gitlab.doe.gov/atse/astra:1.2.4” .

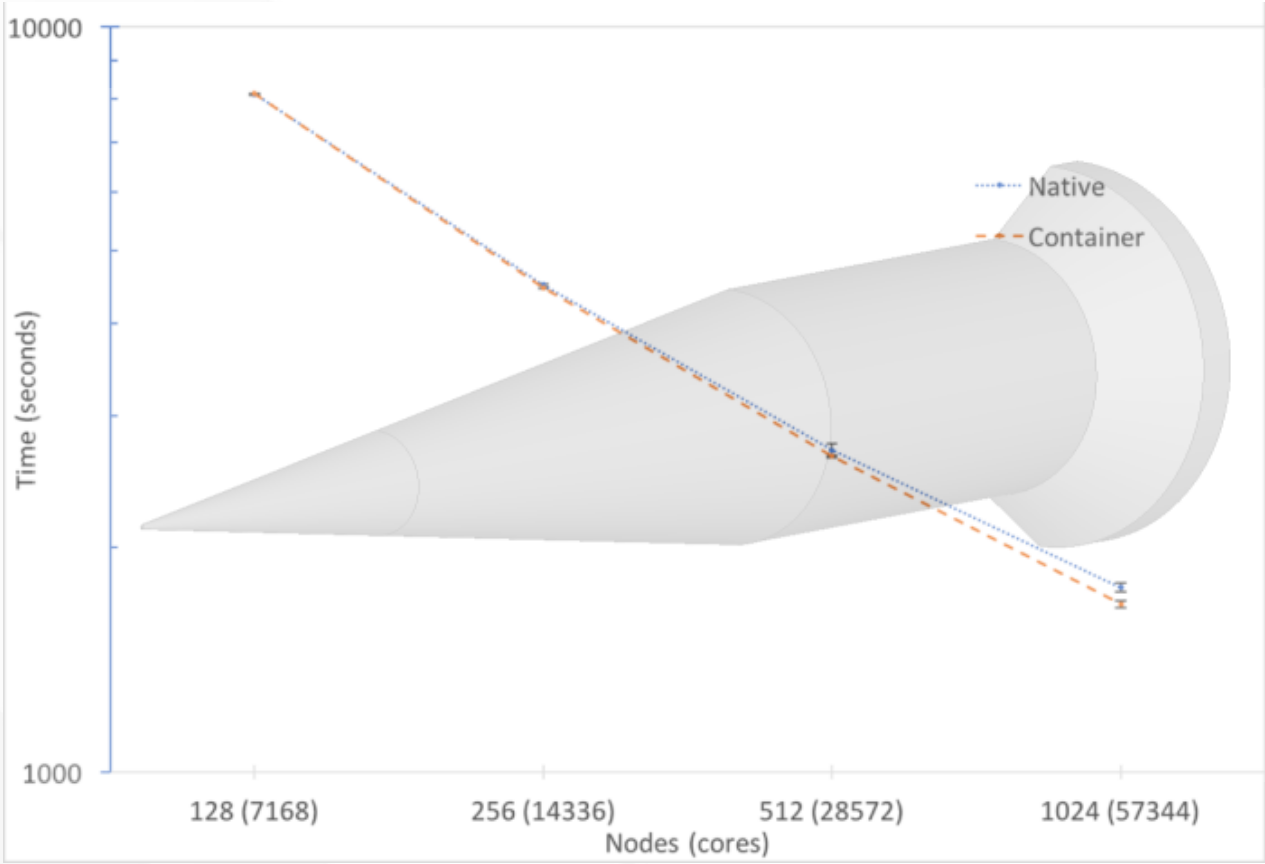
Pedretti, Younge, et. al, *Chronicles of Astra: Challenges and Lessons from the First Petascale Arm Supercomputer,* Supercomputing 2022

SPARC DEMONSTRATION WITH CONTAINERS

- SPARC containerized build & deployment
- Container image build with Podman
- Container on Astra with Singularity
  - Up to 2048 nodes
  - Largest non-x86 container deployment
  - Testing HIFiRE-1 Experiment (MacLean et al. 2008)
- Unable to determine any significant overhead by running containers
  - Confirm previous assertion from LANL (Torrez et al. 2019)
  - Can use multiple containers to compare performance in build variations

Nodes	Cores	Native (s)	Container (s)	Percent Diff
128	7168	8110	8119	+0.1%
256	14336	4501	4461	-0.9%
512	28672	2702	2651	-1.9%
1024	57344	1767	1705	-3.6%
2048	114688	1412	1429	+1.2%

Containers also useful for quickly testing PE changes

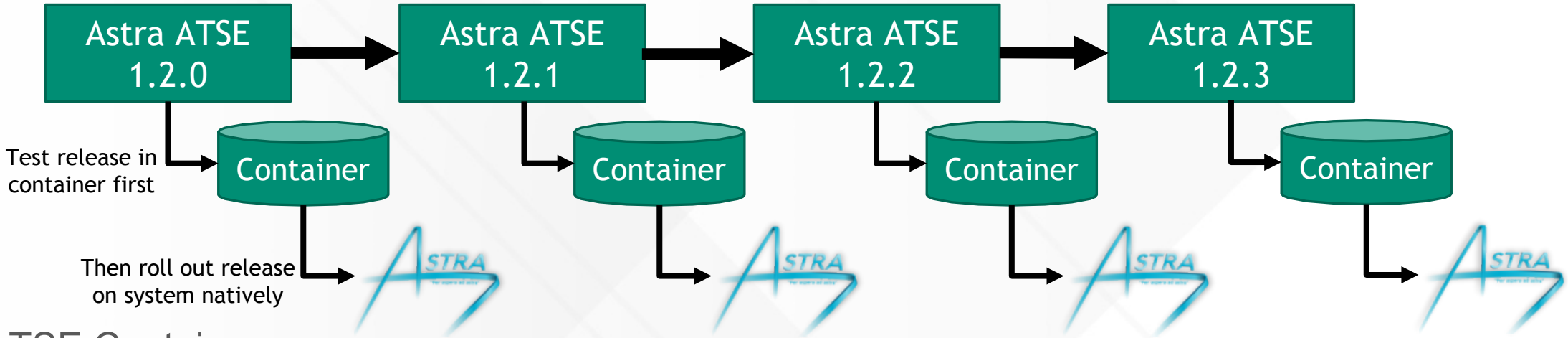


- Near-native performance using a container
- Performance difference within app variation at scale
- Above chart show container built identical to native but with new compiler optimizations
  - Container testing new optimizations for TX2 CPUs

CONTAINERS FOR SOFTWARE TESTING & DEBUGGING

Astra ATSE programming environment release consists of:

- TOSS base operating system + Mellanox InfiniBand stack
- {2 compilers} * {3 mpi implementations} * {~25 libraries} = 150 packages
- Each release packaged as a container for testing and archival purposes



ATSE Container use cases:

- **Release testing:** Enables full applications to be built and run at scale (2048+ nodes) before rolling out natively
- **Rollback debug:** If issues are identified, ability to easily go back to a prior software release and test
- **Cross-system synchronization**: Move full user-level software environments between systems. In one instance, this allowed an Astra InfiniBand library bug to be debugged off platform on another Arm cluster.

CONTAINER PERFORMANCE PORTABILITY CONTINUUM

How do we strike the right balance?

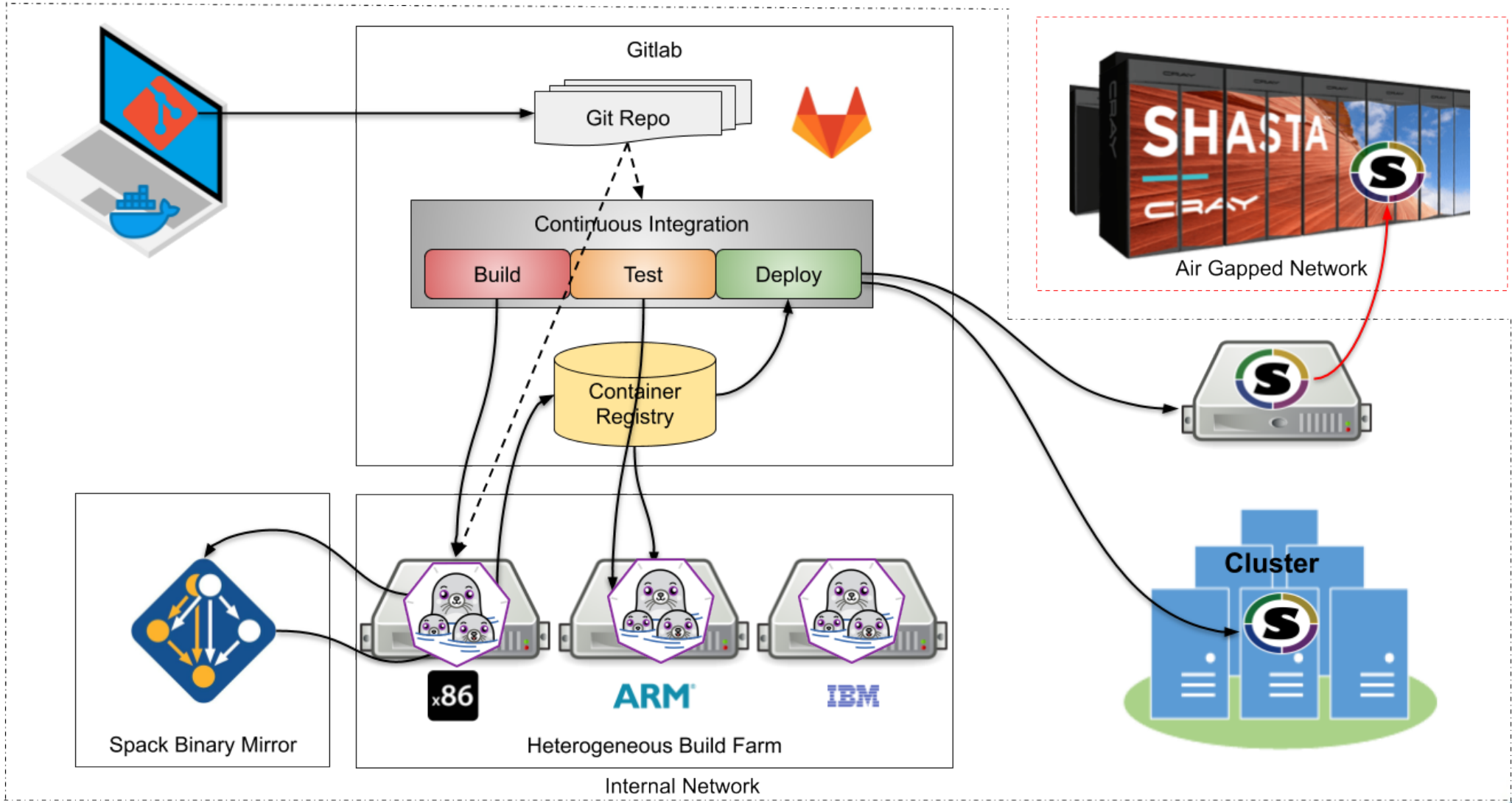
Reproducibility is possible
- Every
- Traceability is possible via build manuscripts
- No image modifications

Performance can suffer – no optimizations
- Can't build for AVX512 and run on Haswell
- Unable to leverage latest GPU drivers

- Performant container images can run at near-native performance compared to natively build applications
- Requires targeted builds for custom hardware
  - Specialized interconnect optimizations
  - Vendor-proprietary software
- Host libraries are mounted into containers
  - Load system MPI library
  - Match accelerator libs to host driver
- Not portable across multiple systems

CONTAINERIZED CONTINUOUS INTEGRATION

As a *developer* I want to *generate container builds from code pull requests* so that *containers are used to test new code on target HPC machines.*



PODMAN – IN – PODMAN

- Many DOE codes use Gitlab for developing HPC applications
- Need to leverage Continuous Development and Continuous Integration capabilities
  - Build – test – deploy HPC apps
  - Includes automatic building container images
- Gitlab CI has runners, but expect elevated privileges
  - Can't enable on HPC systems!
- Solution:
  - Setup the gitlab runner in a container
  - Run Rootless Podman to have gitlab-runner think it has root privs
  - Gitlab-runner then auto-starts a container build process within the first container!
  - Push resulting container to Gitlab container registry (or sitewide registry)
- Simplified container build & deploy infrastructure

ENABLING PODMAN TO SCALE

- Podman (and buildah) have proven tools for building HPCcontainers
- But we still use HPC container runtimes (Shifter, Singularity, …) for running at scale

- ECP Supercontianers team collaborating with Red Hat's Podman team to enable scalable launch with Podman
  - Investigate container distribution modes
  - Interoperability between other container image types
  - Integration with HPC shared filesystems (eg: Lustre)
- One production container implementation to *build*, *run*, and *scale* on Supercomputers & Clouds

OVERVIEW FOR ENABLING EXASCALE CONTAINERS

Select production container technology with wider (cloudier) market
- Help community move beyond Docker
- **Podman & Buildah** best target to date
- Standardize around a tech target and product

Podman

Focus HPC container runtimes on R&D activities
- Solve the unique problems to HPC for HPC first
- Prove out new & innovative approaches at scale

SHIFTER

Charliecloud

"HPC-ify" cloud-based production container runtime with HPC
- Enable scalable launch & integration with job schedulers
- Integrate the proven R&D as new capabilities
- Demonstrate next-gen AI workloads with new container deployments on HPC

SUPERCONTAINERS

Bring HPC developers up to date with modern DevOps
- CI pipelines which produce container images that run on leadership-class supercomputers

Re-investigate resource management solutions for HPC
- Do we still need to batch everything?

kubernetes

We are hiring! ajyoung@sandia.gov

EMERGING WORKLOADS ON HPC WITH CONTAINERS

Extreme-scale Scientific Software Stack (**E4S**)

- Container image contains everything and the kitchen-sink
  - Includes all ECP software activities
- Lightweight base images now available

Support merging AI/ML/DL frameworks on HPC

- Containers may be useful to adapt ML software to HPC
- Already supported and heavily utilized in industry

Working with DOE app teams to deploy custom ML tools in containers

Investigating scalability challenges and opportunities

Credit: Sameer Shende (U. Oregon)

SPACK ENVIRONMENTS HELP WITH BUILDING CONTAINERS

We recently started providing base images with Spack preinstalled.

Very easy to build a container with some Spack packages in it:

spack-docker-demo/
- Dockerfile
- spack.yaml

```
FROM spack/centos:7

WORKDIR /build
COPY spack.yaml .
RUN spack install
```

Base image with Spack in PATH

Copy in spack.yaml
Then run `spack install`

```
spack:
  specs:
    - hdf5 @1.8.16
    - openmpi fabrics=libfabric
    - nalu
```

List of packages to install, with constraints

Build with `podman build .`

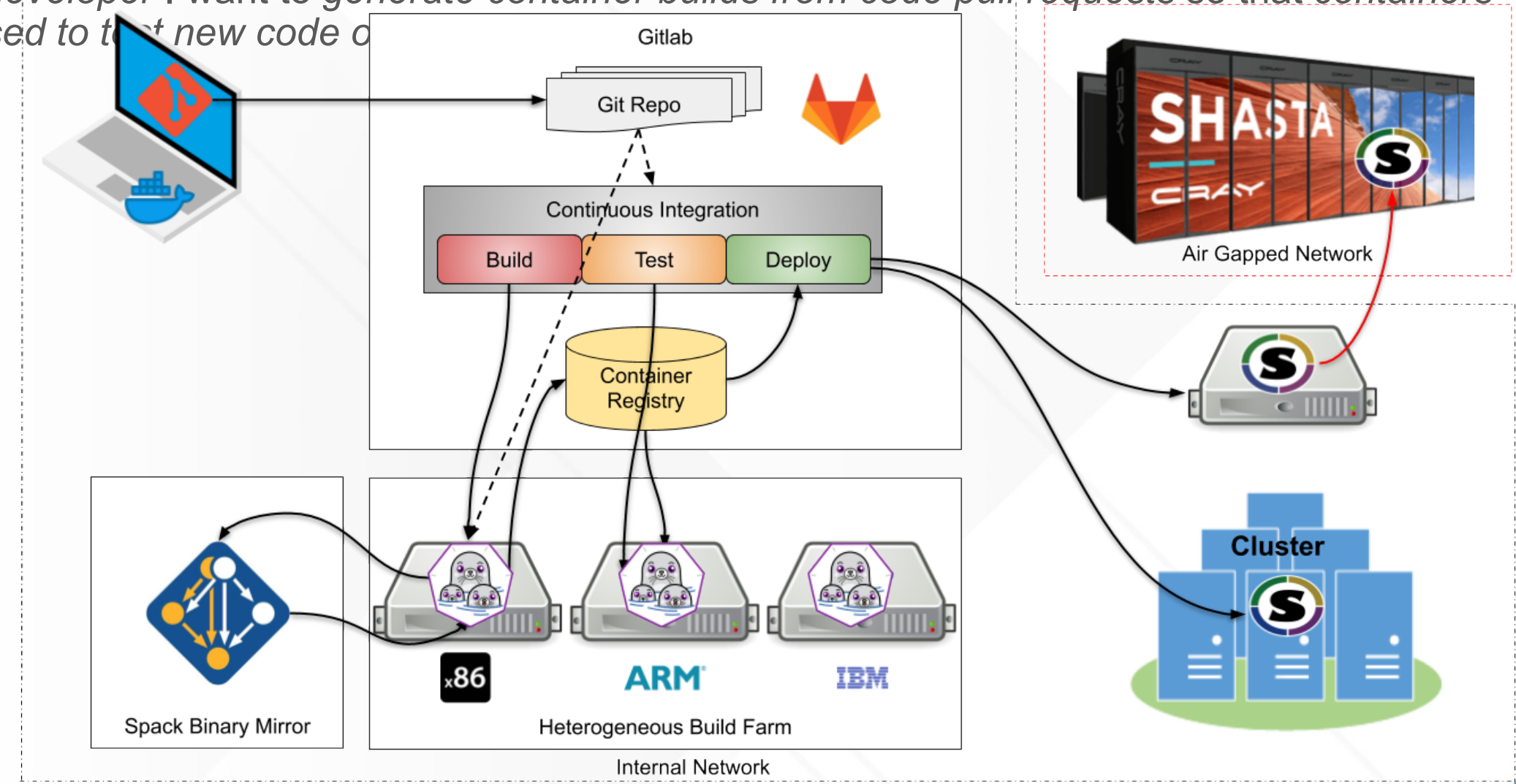
Run with Singularity
(or some other tool)

Credit: Todd Gamblin (LLNL)

Currently under development

CONTAINERIZED CI PIPELINE

As a *developer* I want to *generate container builds from code pull requests* so that *containers are used to test new code c*



CONTAINER TAKEAWAYS (TUPPERWARE?)

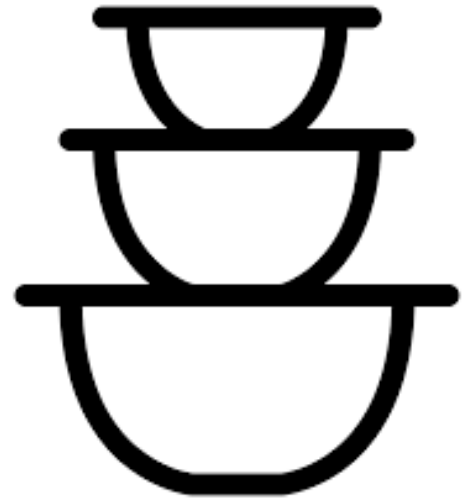
Use ~~Docker~~ Podman to build manifests of full apps
- Developers specify base OS, configuration, TPLs, compiler installs, etc
- Use base or intermediate container images (eg: TOSS RPMs in a container)
- Leverage container registry services for storing images
- Import/flatten OCI images into Singularity & run on HPC resources
  - Also works for Charliecloud and Shifter
  - Can Podman also be used for scalable launch in the future?

Containers have demonstrated minimal overhead for HPC apps

Enabling On-prem unprivileged containers builds
- More to come with Podman & Buildah for HPC

HPC Container Advantages
- Simplify deployment to analysts (just run this container image)
- Simplify new developer uptake (just develop FROM my base container image)
- Decouple development from software release cycle issues
- Reproducibility has a new hope?

Supercontainers for Exascale
- Preparing to enable containers at Exascale
- Simplify HPC application deployment via modern DevOps
- Support next generation AI & ML apps



THANKS!

ajyoung@sandia.gov

ATSE PROVIDED A FOCAL POINT FOR DEVELOPMENT

Curated HPC software stack

- Provides base set of compilers and MPI implementations known to work well together
- Didn't want users to have to build entire third-party library (TPL) stack themselves for Arm

Labor intensive to assemble and test

- Leveraged OpenHPC recipes to speed development, extremely helpful
- Complicated by specific version requirements, μarch optimizations, static library support

ATSE continuing to evolve

- Moving to Spack-based build process to improve development speed + flexibility
- Improving container packaging and deployment
- Shifting towards Vanguard-II

Effort to standardize programming environment components

ATSE 1.2.5 Recipes Available @ https://doi.org/10.5281/zenodo.4006668

CONCLUSION

Astra the first Petascale supercomputer based on 64-bit Arm processors
- First Vanguard Advanced Prototype platform for DOE/NNSA
- Now running production applications for NNSA mission

Several technology gaps were identified and addressed
- Software stack enablement, Linux bugs at scale, thermal management considerations, power management capabilities, parallel filesystem support, and enabling advanced container support.
- Pushed fixes back into community
- Focus on Arm compiler maturity

Several lessons learned applicable to any first-of-a-kind Supercomputer
- First DOE Exascale platforms
- Future Vanguard II+ platforms
- RISC-V and other custom designs

Acknowledgements:
Computer Systems & Technology Integration group (9320)
Extreme Scale Computing group (1420)
IT Infrastructure Services group (10770)
NNSA Tri-lab collaborators (LANL and LLNL)
Industry Partners: HPE, Mellanox, Red Hat, Arm, and Marvell

Arm is now a viable production HPC technology for the largest-scale supercomputers

See Fugaku - #1 Top500

POSITION 2: … AND SO IS THE CLOUD

The hyperscalers are finally paying attention to HPC

- *"The physical network topology does affect performance; particularly important is the performance of MPI Allreduce, grouped by splitting the mesh by a subset of the dimensions, which can be very efficient [5] [6] if each such group is physically connected."* – Shazeer et al Google Brain, Mesh-TensorFlow: Deep Learning for Supercomputers.
- As learning techniques grow in scale, HPC becomes more important.

HPC cannot compete with the hyperscalers

- Let's stop trying and start *integrating*
  - *That doesn't mean adopting Cloud as-is*
  - *That doesn't dissolving HPC either*
- The closer HPC and cloud paradigms get, the better we all are
  - Encourage open source infrastructure
  - Collaborative partnerships
- Avoid boutique solutions without sacrificing performance

REPRODUCIBILITY IN HPC

- Reproducibility is a cornerstone of quality science!
  - Consistent results across studies aimed at answering the same **scientific** question
  - Critically important in conducting computational science today
- DOE/NNSA must extend the lifecycle without underground testing
- Rely on modeling and simulation apps to perform this task
  - incorporate a multitude of physics and engineering models
  - Executed on leadership-class supercomputers
- Long-term studies take years
  - Any particular simulation/model may not seem important at the time
  - Later analysis may prove to demonstrate value in an old simulation
  - Need to reproduce & reevaluate runs many months or years later!
- Can containers help or hinder?

ECP SUPERCONTAINERS

Joint DOE effort - Sandia, LANL, LBNL, LLNL, U. of Oregon

Ensure container runtimes will be scalable, interoperable, and well integrated across DOE
- Enable container deployments from laptops to Exascale
- Assist Exascale applications and facilities leverage containers most efficiently

Three-fold approach
- Scalable R&D activities
- Collaboration with related ST and AD projects
- Training, Education, and Support

Activities conducted in the context of interoperability
- Portable solutions
  - Optimized E4S container images for each machine type
  - Containerized ECP that runs on Astra, A21, El-Capitan, …
- Work for multiple container implementations
  - Not picking a “winning” container runtime
- Multiple DOE facilities at multiple scales

SUPERCONTAINERS

MOTIVATION FOR CONTAINERS IN HPC

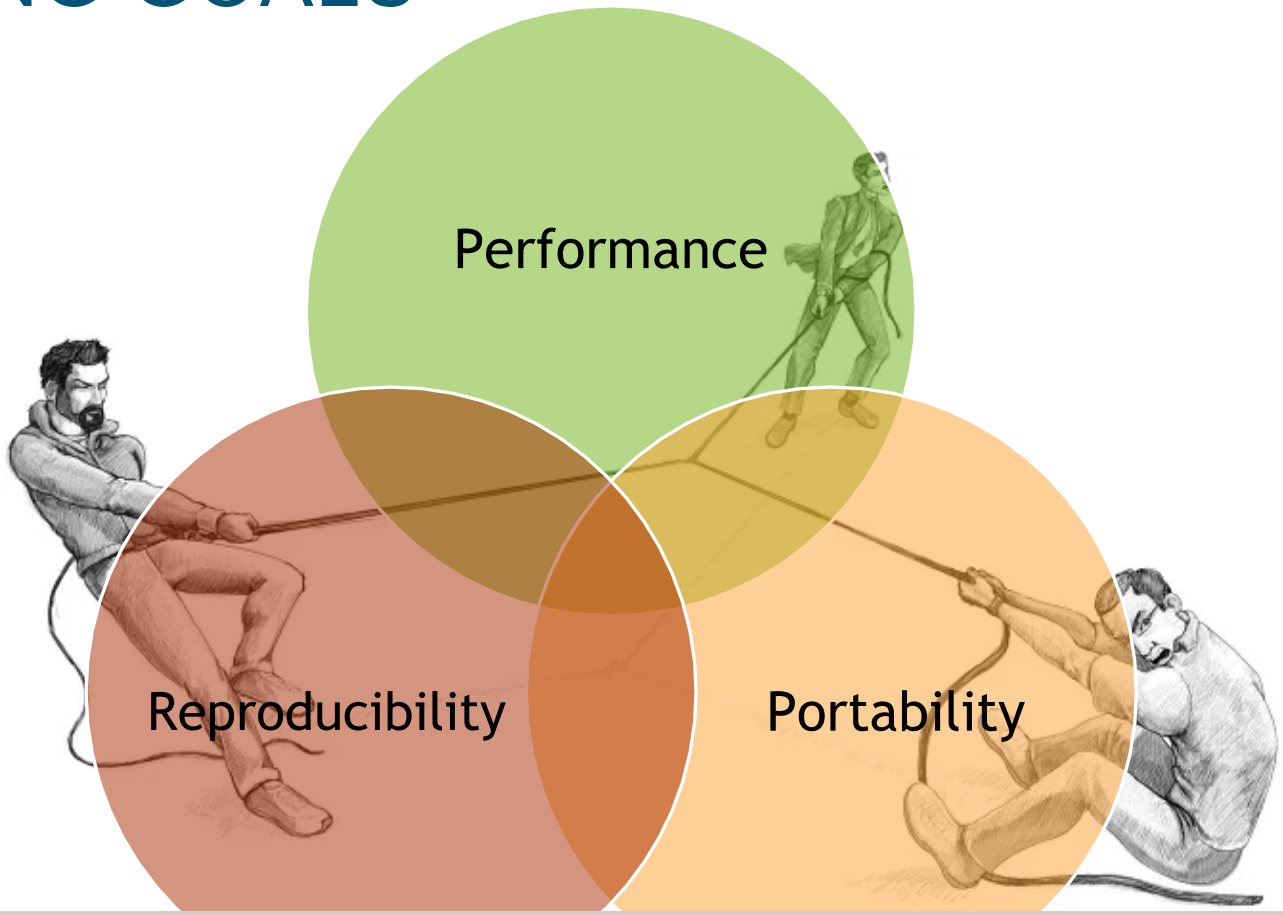
- Containerized computing is being adopted across HPC landscape
- Many potential benefits
  - Prescriptive deployment
  - Modern DevOps
  - Portability of containers
  - Reproduce containerized workloads later *
  - Flexible software ecosystem
- Several potential tools and container runtimes available today
  - Scale from your workstation to a supercomputer
- Eases barrier of entry for complex or emerging software ecosystems

SUPERCONTAINERS

PORTABILITY & REPRODUCIBILITY PROBLEMS

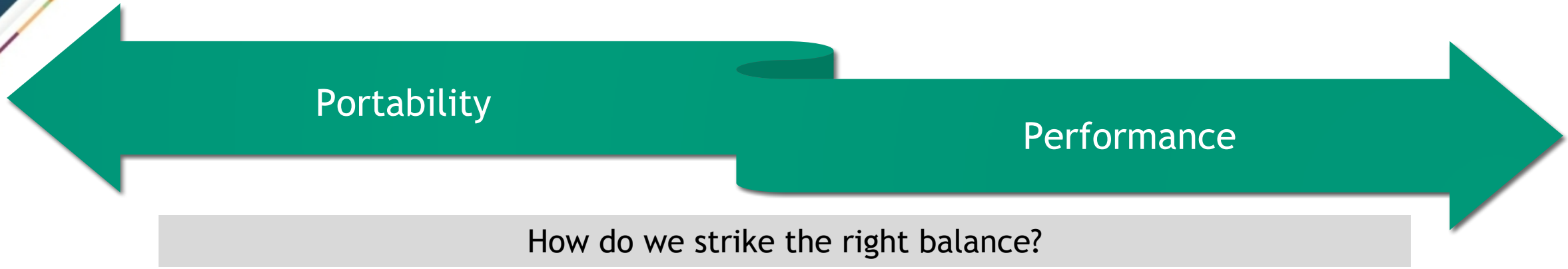
- Containers promise the potential to improve flexibility for developers
  - Support of user-defined software stacks
  - Potential impact in portability and reproducibility
- Current implementations fall short of delivering on promises
  - System must still match the host micro-arch
  - System must be capable of exploiting specialized hardware in HPC
    - High speed, low latency interconnects
    - Specialized instructions & extensions
    - Advanced GPUs and accelerators
  - Require runtimes to leverage host libraries in containers for performance

COMPETING GOALS



Achieving “Ideal” Reproducibility may impact performance and portability and vice versa

CONTAINER PERFORMANCE PORTABILITY CONTINUUM



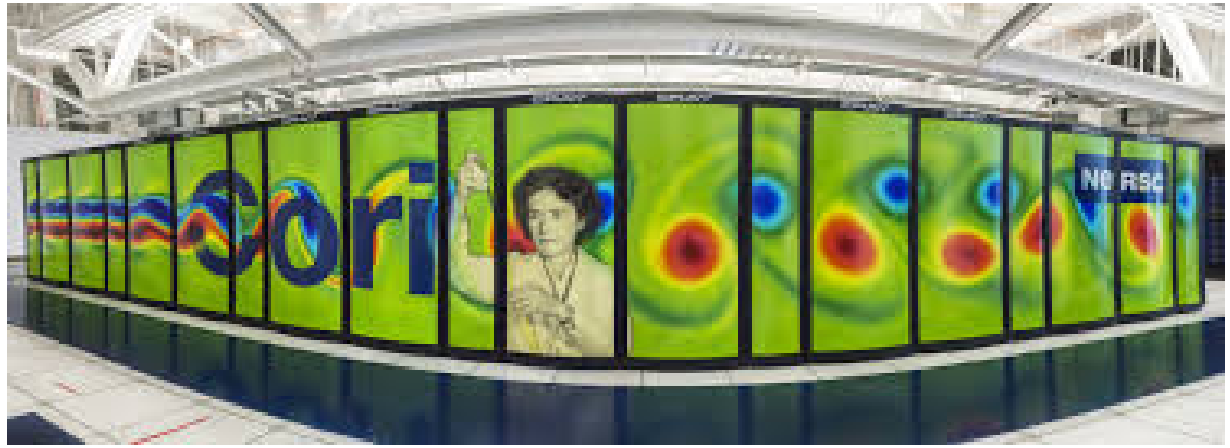
How do we strike the right balance?

- Portable container images can be moved form one resource deployment to another with ease
- Reproducibility is possible
  - Everything (minus kernel) is self-contained
  - Traceability is possible via build manuscripts
  - No image modifications
- Performance can suffer – no optimizations
  - Can't build for AVX512 and run on Haswell
  - Unable to leverage latest GPU drivers

- Performant container images can run at near-native performance compared to natively build applications
- Requires targeted builds for custom hardware
  - Specialized interconnect optimizations
  - Vendor-proprietary software
- Host libraries are mounted into containers
  - Load system MPI library
  - Match accelerator libs to host driver
- Not portable across multiple systems

STATE OF THE PRACTICE IN CONTAINERIZED HPC

- System-specific libraries are needed within a given HPC container image
- Combine bind mounts and dynamic linking to inject optimized libraries
  - From the host into the container runtime environment
  - Assert libraries are optimally configured to drive HPC hardware
  - ”Container Bypass” mechanism
- Example: MPICH-based implementations
  - Rely on ABI compatibility
    - Build from generic MPICH on container
    - Swap in CrayMPI at runtime
  - Force MPI apps to use optimal MPI
  - 2 methods
    - Replace libmpi.so directly
    - Overlay/bind mount and change LD_LIBRARY_PATH
  - Demonstrated with Shifter at full system scale on Cori



OPENMPI USAGE IN CONTAINERS

- Container developer has 2 options
  - custom-build OpenHPC to fit HPC spec
    - IB versions, MOFED userspace drivers, PMIX, etc
    - Requires detailed knowledge of target system
    - Not portable
  - Generic OpenMPI build and use container bypass for OpenMPI
    - OpenMPI ABI incompatibilities across sub-versions!
    - OpenMPI uses RPATH -> many library dependencies have to be handled
      - Custom bind-mounts can be unweildy
    - Userspace container drivers must match host drivers (MOFED)
- Current container usage model on Astra
  - Works & scales >2000 nodes
  - Cumbersome for custom containers

STATE OF PRACTICE: PODMAN ON THE ASTRA SUPERCOMPUTER

Astra ideal test environment for Podman container build experiments

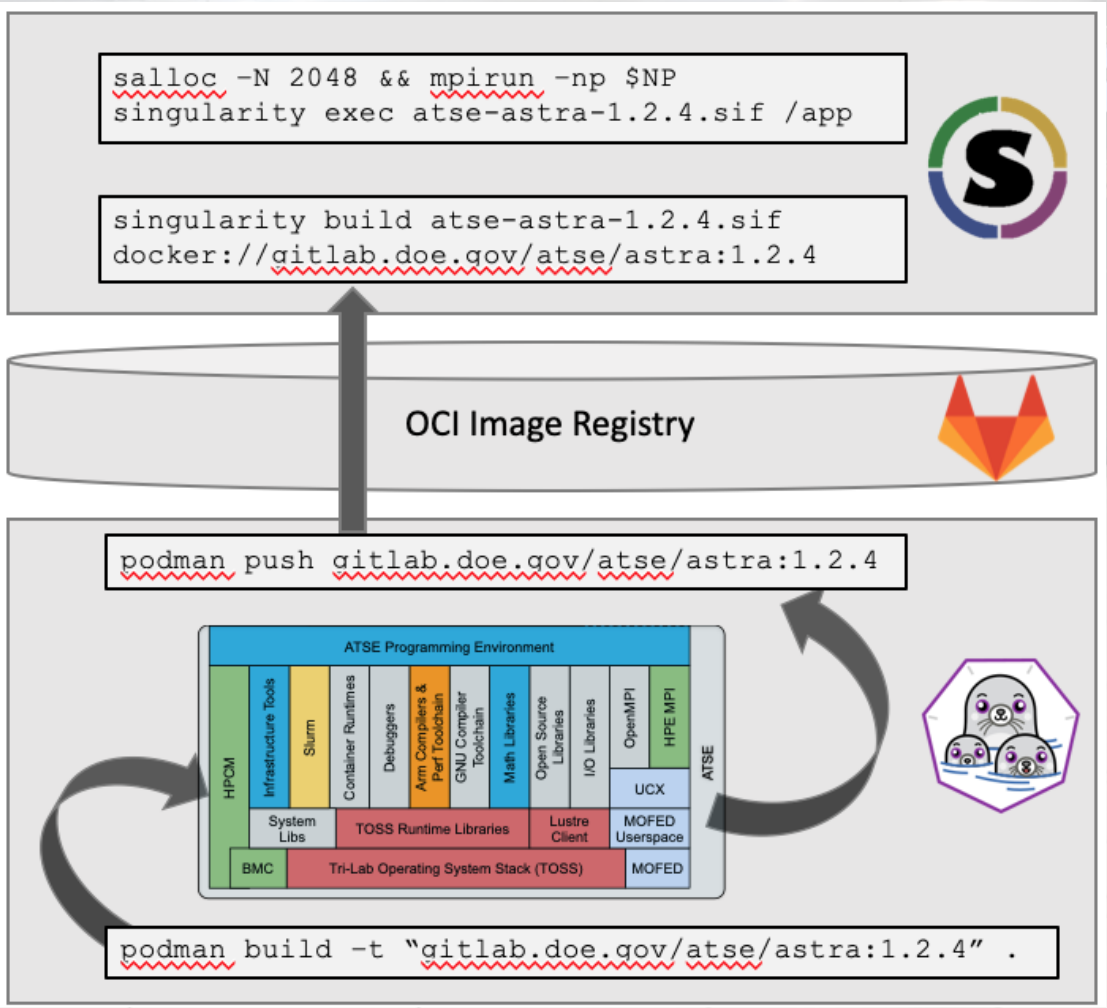
- 1st Arm-based Supercomputer on Top500 (Nov17)
  - 5000+ Marvel ThunderX2 processors (aarch64 armv8)
  - Significant need for container builds on Arm architecture
- Astra a prototype system => increased flexibility for R&D

Building Advanced Tri-lab Software Environment (ATSE) and HPC apps => in a container

- Built directly on Astra login nodes with Podman
- Now using Spack
- Pushed OCI images to site Gitlab container registry
- Singularity to run at scale => can use any HPC container runtime (with some relativity)

Demonstrated first on-platform container build, but prototype has limitations

- RHEL7 missing several features
  - Overlay, FUSE, NFS client features
- Collaborating with Red Hat to productionize on RHEL8



CONSIDERATIONS

- HPC applications need to use specialized interconnects & libraries not found or optimized for in base OS packages.
- Typical container solution requires mapping in libraries which can cause host-to-container incompatibilities.
- There are differing methodologies in container runtimes for incorporating GPUs and accelerators.
- Users may not know if a given container image can be ported to a different HPC system.
- If portability is possible, users may not know what performance implications exist when running a container on a different HPC system.

THE CONTAINER BYPASS PROBLEM

- The Open Container Initiative (OCI) spec can help standardize some aspects
  - But not well fit for HPC as-is.
- HPC container runtimes cannot easily determine what libraries are needed
- Problem: host libraries can have conflicting dependencies with what's in the container

```
$ srun -n 1 shifter /app/hello
...
/app/hello: /lib/x86_64-linux-gnu/
libm.so.6: version 'GLIBC_2.23' not found
(required by /opt/udiImage/modules/mpich/
lib64/dep/libquadmath.so.0)
...
```

  - Glibc mismatch error found on upgraded Cray system
    - Host: CLE7, SLES15, glibc 2.23 vs Container: Centos6, glibc 2.17
  - Host library mounted in the container not forward compatible with container's glibc
- Problem is not theoretical, it's real. And likely to occur again with interconnects, system updates, driver changes, …

PROPOSED SOLUTIONS

What do we do to avoid mismatches between host and container libraries?

Some potential options:

1. Custom Image Labels
2. Backwards Compatible Libraries
3. A container compatibility layer
4. System-level Virtualization

1. CUSTOM IMAGE LABELS

SHIFTER

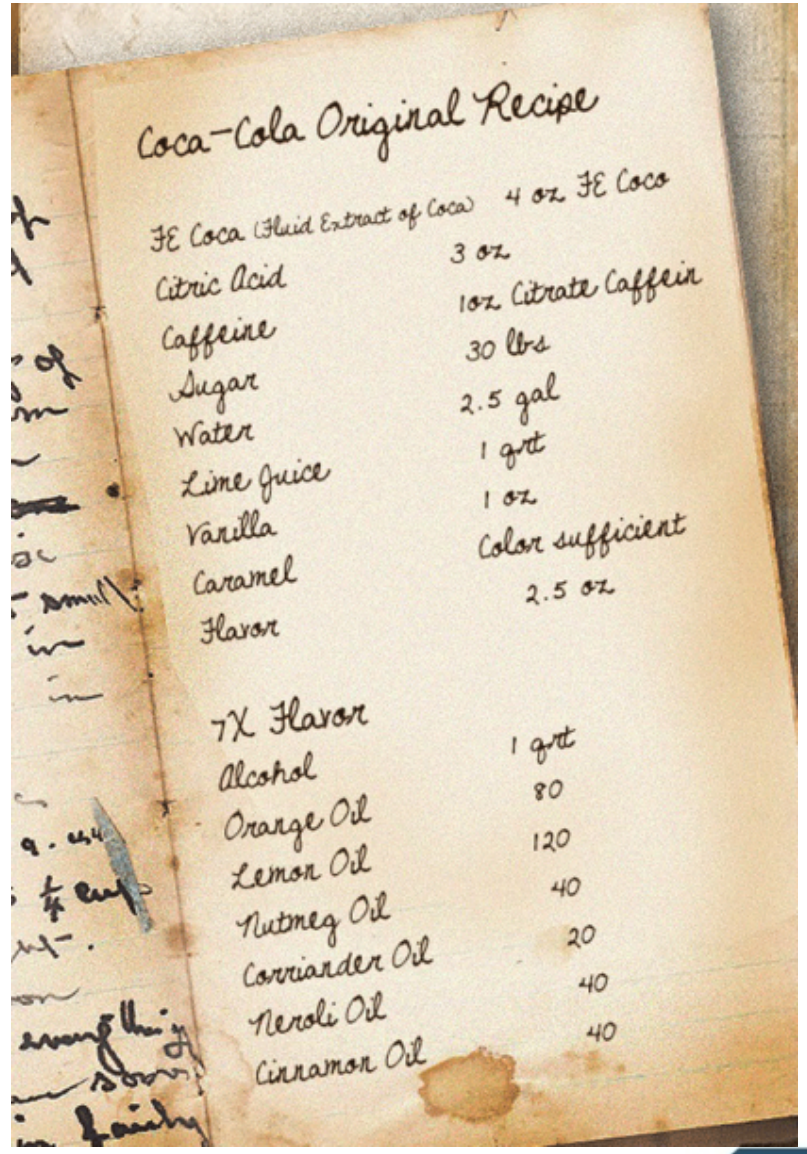
- Leverage OCI-compatible image LABELs
  - Insert directly in Dockerfile
  - Could reproduce in Singularity defs
  - Essentially embedding metadata into spec
- Labels specify expectations from the host
  - HPC container runtime intercepts labels, makes appropriate library insertion
  - Specify MPI version, Glibc expectation, etc
- Implemented prototype solution in Shifter
- Potential pitfalls:
  - Still requires container bypass
  - Requires extra work from developer

```
FROM centos:7

LABEL org.supercontainers.mpi=mpich
LABEL org.supercontainers.glibc=2.17

RUN yum -y update && \
    yum -y install gcc make gcc-gfortran \
        gcc-c++ wget curl

RUN B=mpich.org/static/downloads && V=3.2 && \
    wget $B/$V/mpich-$V.tar.gz && \
    tar xf mpich-$V.tar.gz && \
    cd mpich-$V && \
    ./configure && \
    make && \
    make install

ADD helloworld.c /src/helloworld.c

RUN mpicc -o /bin/hello /src/helloworld.c
```

Label	Values	Comment
org.supercontainers.mpi	{mpich,openmpi}	Required MPI support, ABI compatibility
org.supercontainers.gpu	{cuda,opencl,rocm, etc}	Required GPU library support
org.supercontainers.glibc	Semantic version: XX.YY.Z	Specific version of GLIBC

2. BACKWARDS COMPATIBLE LIBRARIES & SOURCE CODE

- Why can't we just build everything in a container in the first place?
  - Target hardware may not be known in advance
  - May not be possible to create base container image with key software
    - Vendor proprietary, closed source, export control restrictions, etc…
- Require vendors to provide backwards compatible libraries
  - Build MPIX multiple times, with multiple versions
  - Additional build complexity for vendors
- Require vendors to provide direct source code
  - Intractable – ask Cray for CrayMPI source ;)



3. A CONTAINER COMPATIBILITY LAYER

- Construct a lower-level library for HPC container runtimes
  - Could leverage custom labels like #1
  - Make directed decisions on how to bind-mount and `LD_PRELOAD`, resolve glibc issues
- Create a common place for vendors to integrate solutions
  - Instead of building many different solutions for each device/interconnect/accelerator, HW providers implement generic solution.
- Similar approaches exist
  - `libnvidia-container` takes similar approach, but vendor-specific.
  - Container Network Interface (CNI)
    - But not general enough for HPC
- Acceptable solution requires community-wide collaboration
  - From hardware vendors, container runtime developers, system integrators, etc etc

4. SYSTEM-LEVEL VIRTUALIZATION

- If reproducibility is paramount, we should leverage ISA virtualization
  - Virtual machines & hypervisors
  - Similar to cloud implementations, multiple levels of abstraction
  - Containers atop virtual machines atop HPC hardware
- Linux kernel & host environment to match container reqs with HPC hardware
- Control kernel config, drivers, to run container images only
  - Disable arbitrary VM models, no user control
  - Users only control container images, run in userspace
  - No root access
- Concerns
  - Significant infrastructure investment?
  - Can domain scientists, used to HPC tuning, handle this much abstraction?
  - Acceptable performance in HPC with hypervisors?

OPEN DISCUSSION

Containers are able to enable new mechanisms in portability and reproducibility
- Reproducibility is of critical importance for conducting quality science
- Performance is paramount in HPC

Containerization in HPC has to fix container bypass issues to deliver

We've outlined and started investigating several potential solutions
- Will more metadata _really_ fix things?

What do **you** think is the right path forward for the HPC community?

SUPERCONTAINERS

We are hiring! ajyoung@sandia.gov