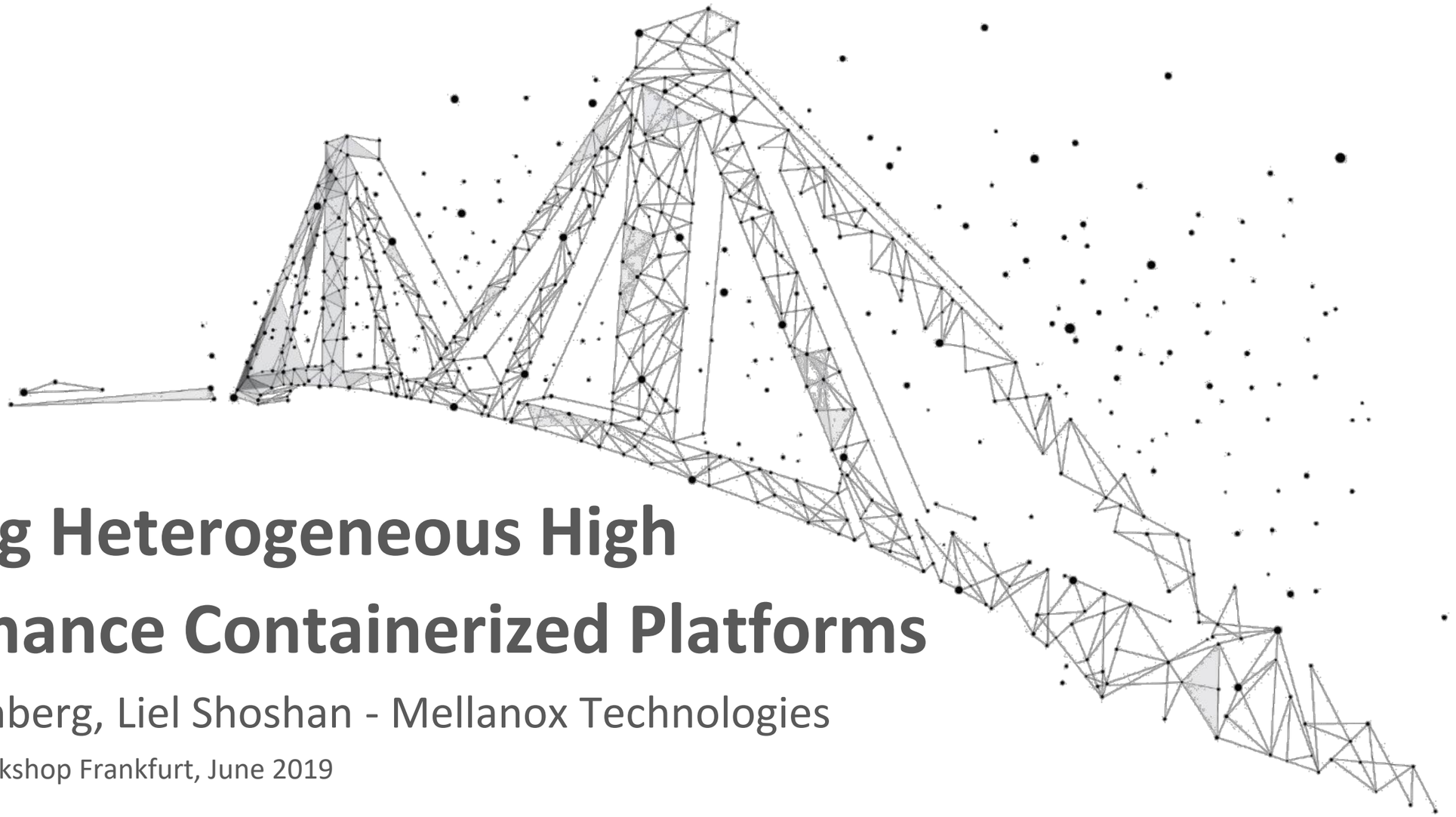


Enabling Heterogeneous High Performance Containerized Platforms

Dror Goldenberg, Liel Shoshan - Mellanox Technologies

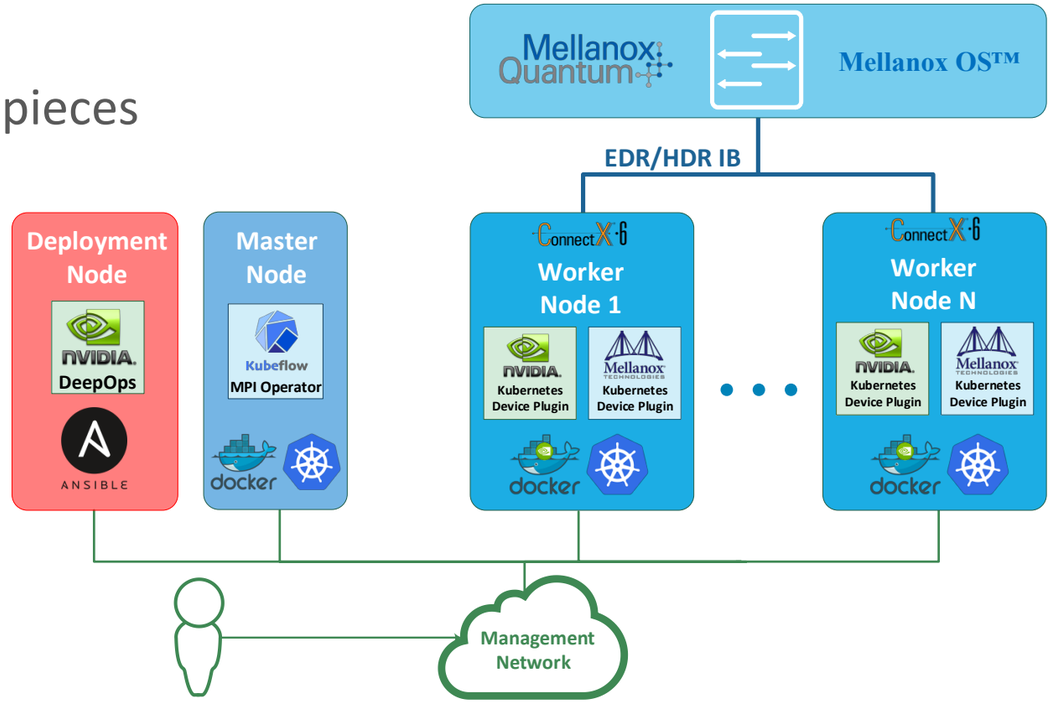
ISC Container Workshop Frankfurt, June 2019

Containerized Applications for Heterogeneous Architectures

- Heterogeneous cluster architectures allows parallel computing that relies on CPU and GPU
- Used for HPC & ML
  - Leveraging high speed, low latency, smart interconnects to speed-up data computation
- GPUDirect RDMA technology improves GPU-GPU communication and eliminates CPU involvement
- Kubernetes serves as a useful way of distributing compute-intensive work across such clusters
- Containerizing compute-intensive applications poses challenges on configuration, deployment and orchestration of the required system devices

Challenges in Building a K8s HPC/ML Cluster

- Deploying a K8s HPC cluster requires installation of various device drivers, libraries and toolkits
  - On the node level
    - Nvidia Driver, CUDA toolkit, cuDNN, MLNX_OFED, GPUDirect, Docker, K8s, etc.
  - On the orchestration level
    - Nvidia Device Plugin, RDMA Device Plugin, K8s CNI, Kubeflow, etc.
  - On the container level
    - Tensorflow, Horovod, MLNX_OFED, OpenMPI, etc.
- One of the biggest challenges is making all these code pieces up and running in an easy and consistent manner

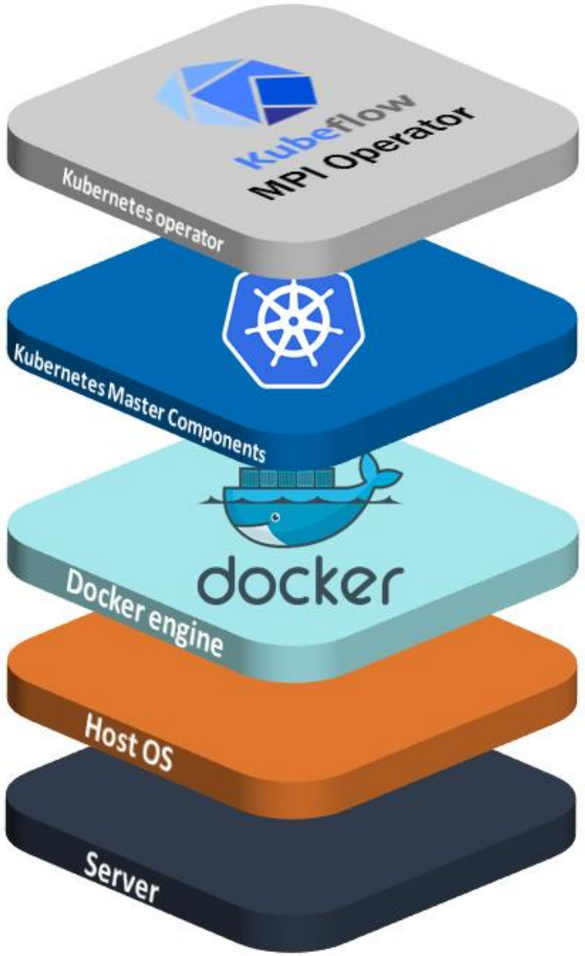


The Solution

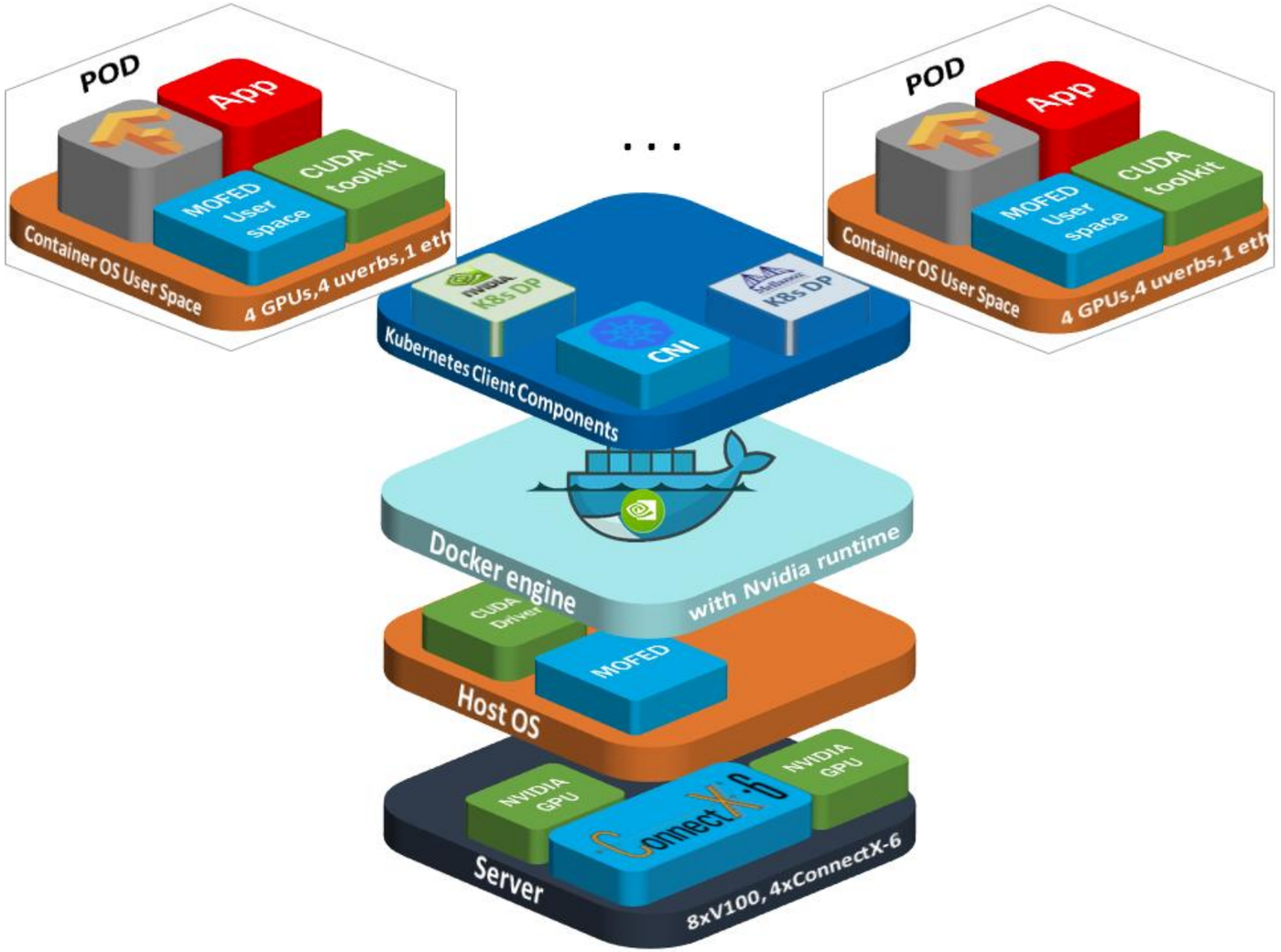
- The following projects speed up deployment time, while making cluster installation vastly simpler
  - DeepOps
    - Facilitates deployment of multi-node GPU and RDMA K8s clusters for ML and HPC environments
    - Employs best practices when setting storage and configuring authentication and user access
  - Kubeflow
    - Kubernetes-native platform for developing, orchestrating, deploying and running scalable and portable ML workloads
    - Provides a straightforward way to deploy best-of-breed open-source systems for ML to diverse infrastructures
    - Helps support reproducibility and collaboration in ML workflow lifecycles
    - MPI Operator
      - Makes it easy to run allreduce-style
  - Mellanox addons for DeepOps
    - Ansible playbook for MLNX_OFED, GPU Direct and K8s device plugin
- Reference deployment guides can be found on community.Mellanox.com and docs.Mellanox.com

K8s HPC Cluster

Master



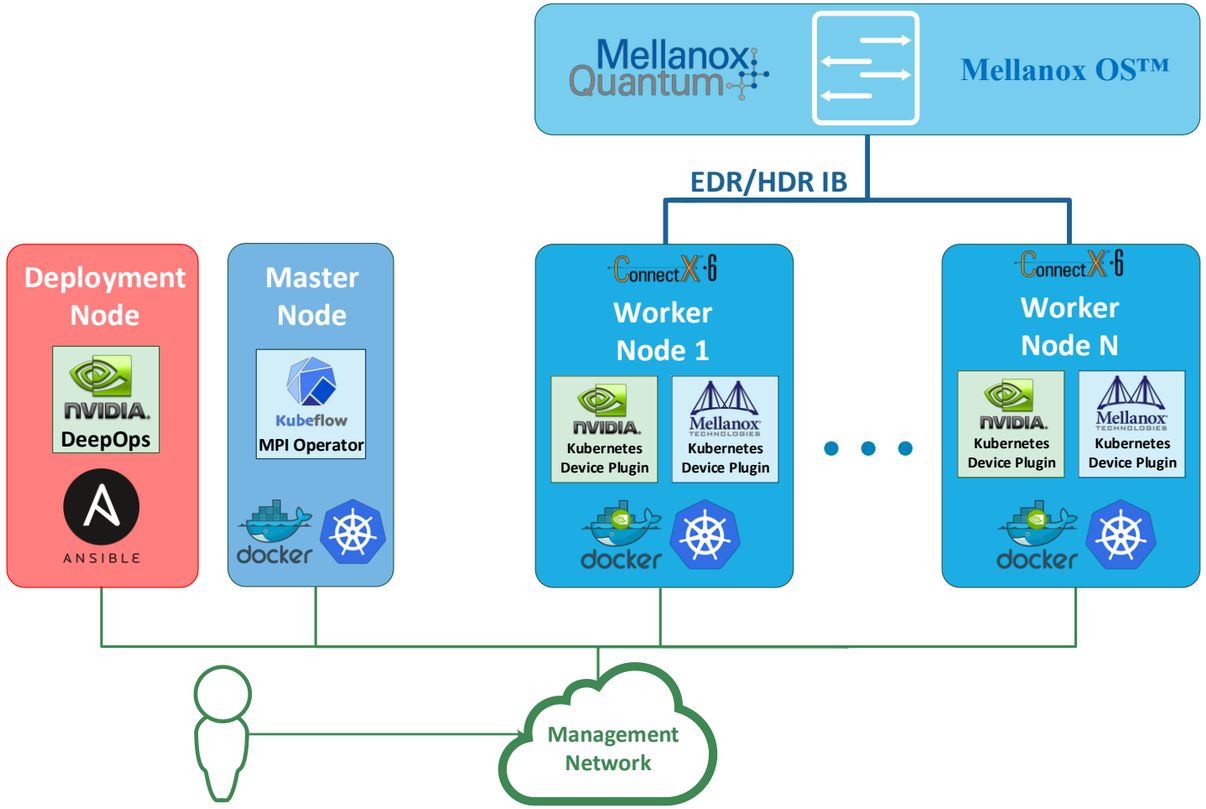
Worker node



Performance Tests

Testing Environment

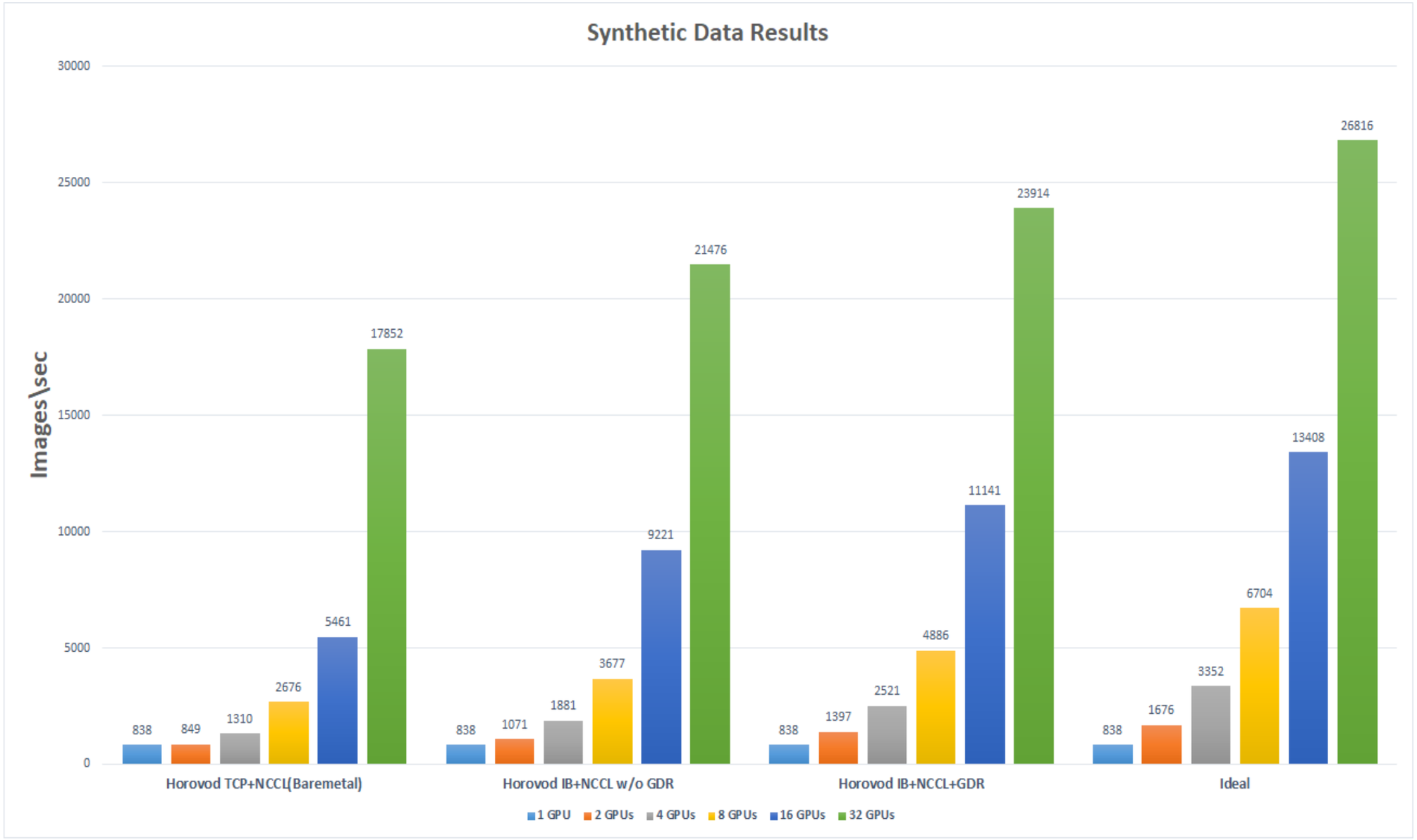
- Topology
  - Nodes
    - Deployment node
    - Master node
    - 4 x Worker nodes
      - Each node has 8 NVidia Tesla GPU cards and 4 Mellanox ConnectX-6 adapters
  - Containers
    - Each worker node runs 1 Pod
- Benchmark
  - TensorFlow v1.12.0
  - Type: Synthetic
  - Batch size: 32
  - Resnet50 model



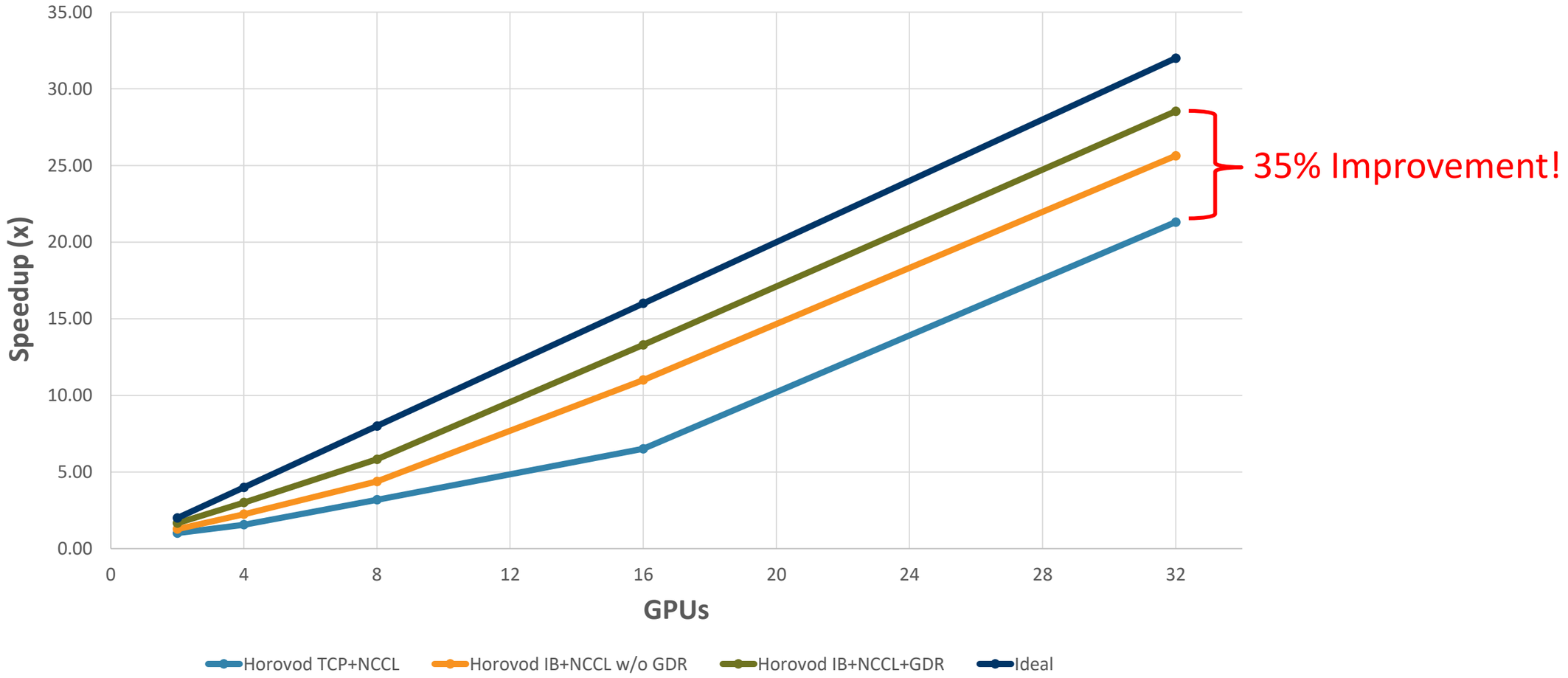
kubernetes

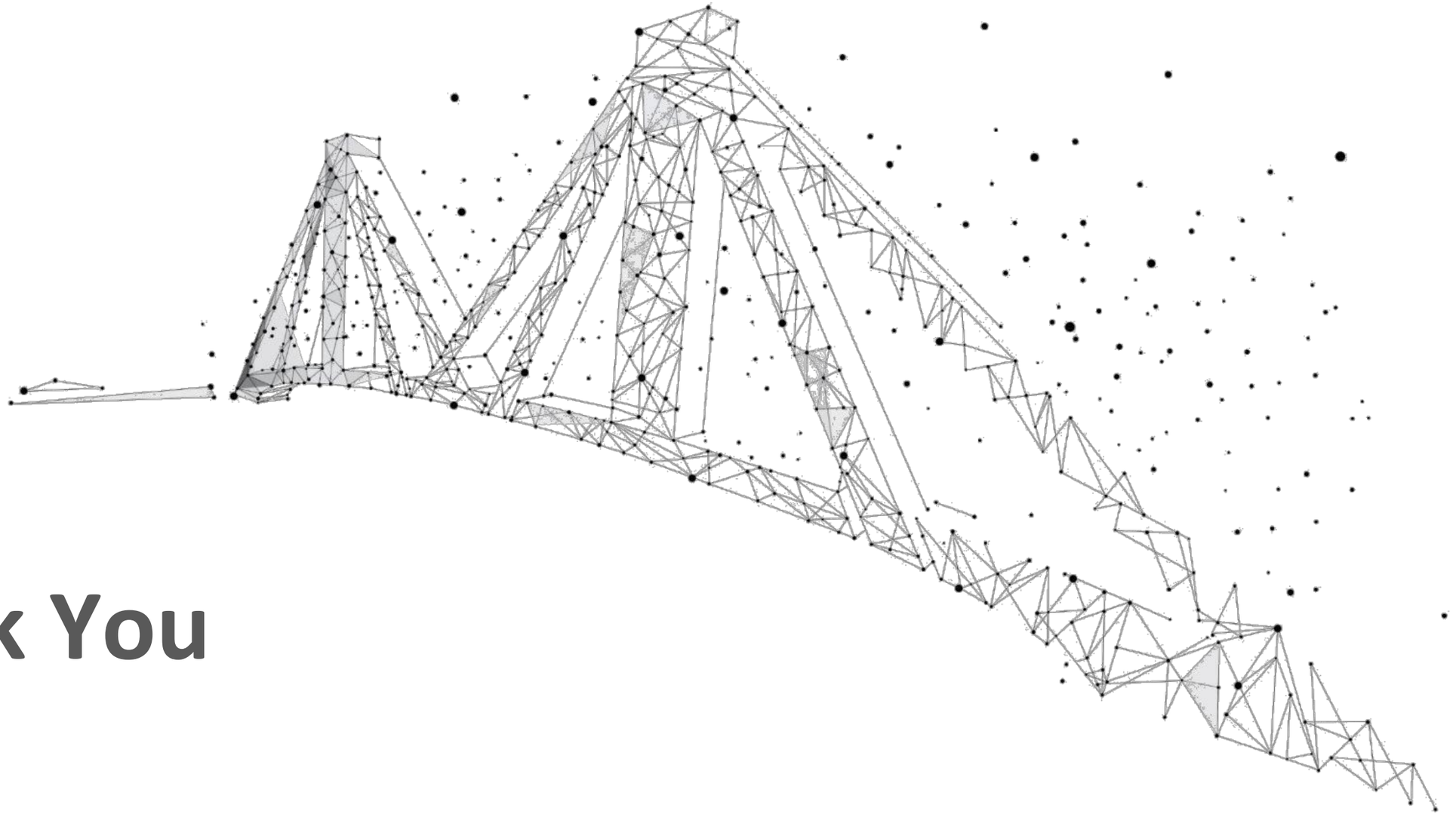
INFINIBAND℠ TRADE ASSOCIATION

Resnet50 Performance Results



Resnet50 Container Performance Results





Thank You

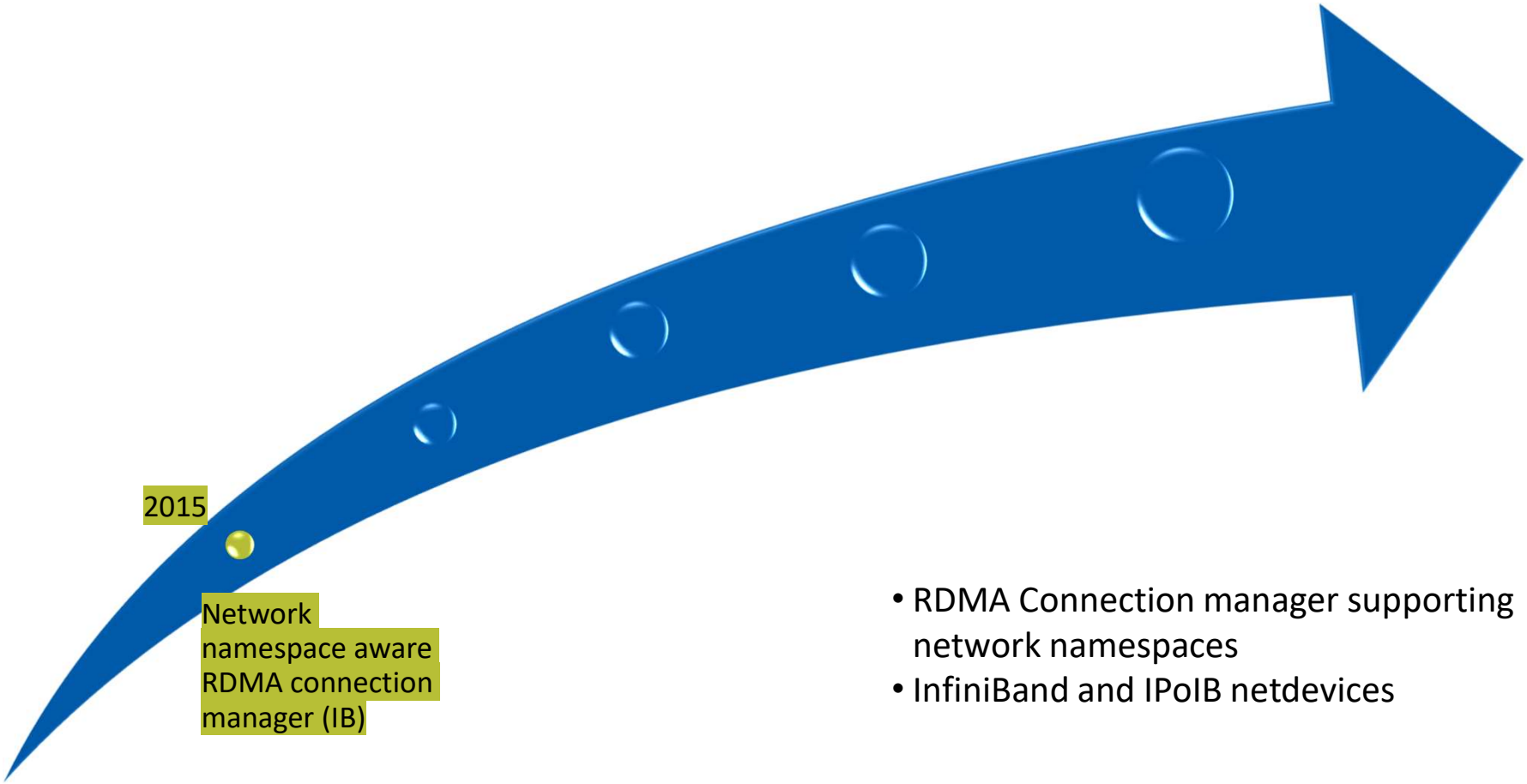
Mellanox Container Journey

Dror Goldenberg, Parav Pandit

Mellanox Technologies

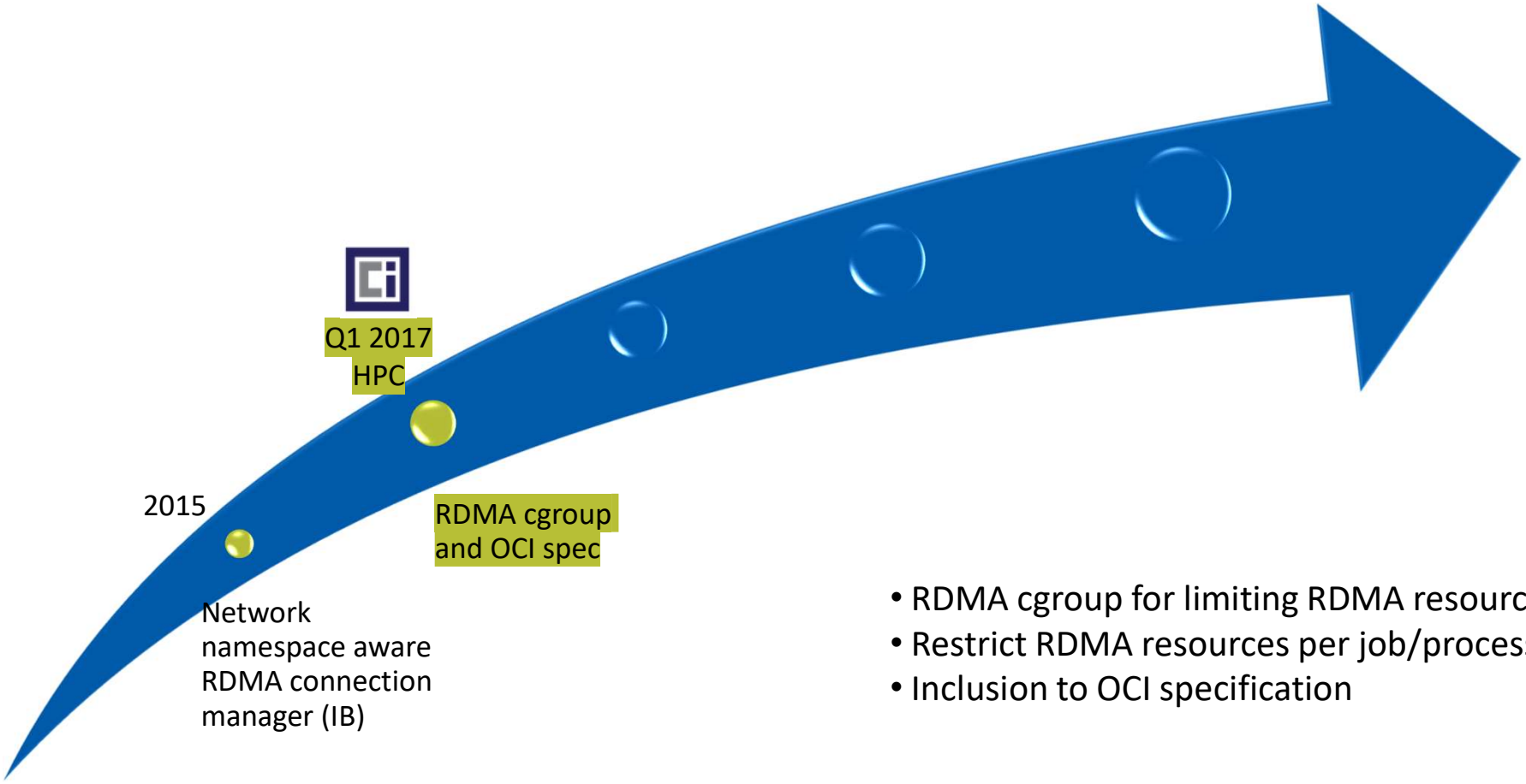
June 2019

Start of RDMA container support



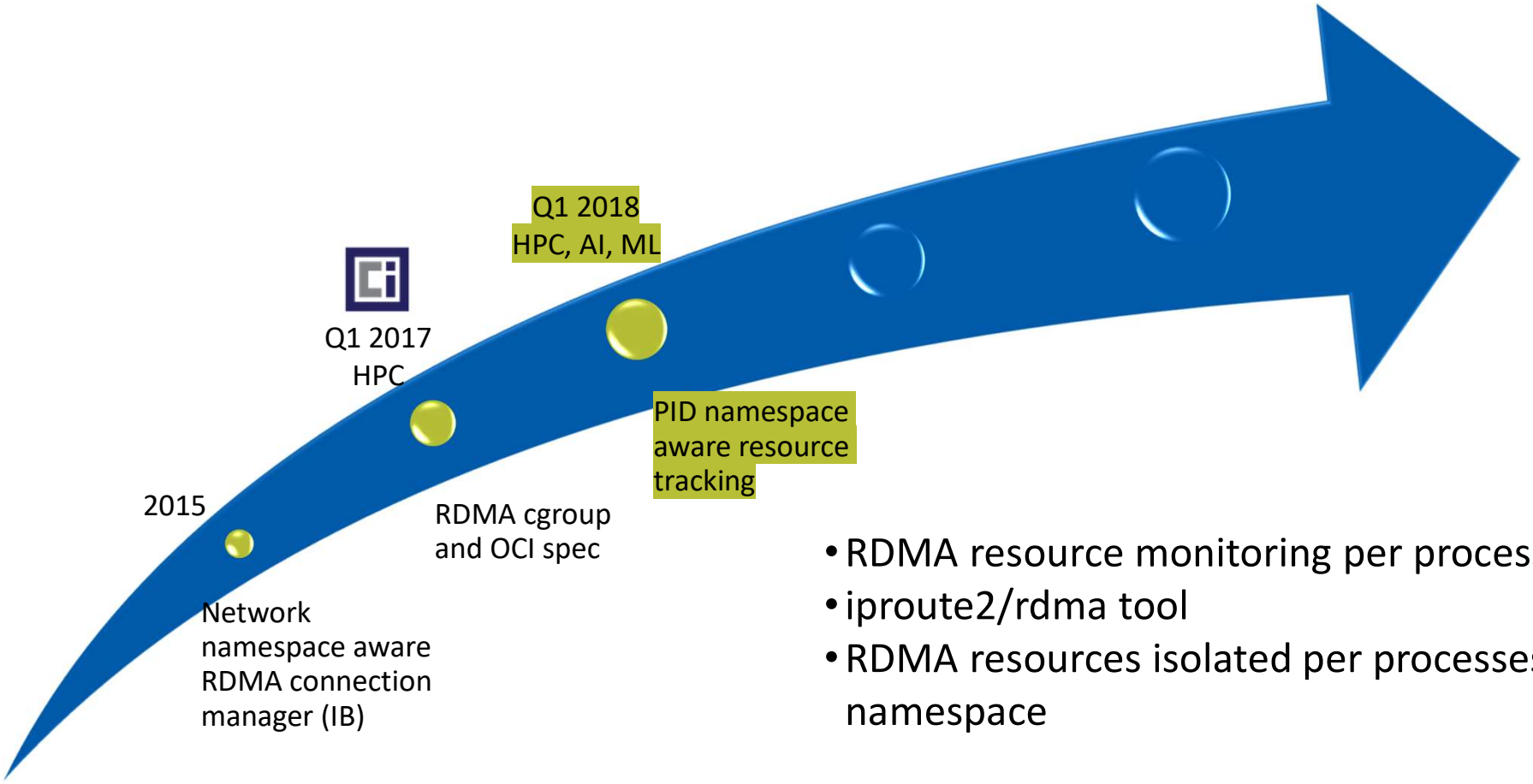
- RDMA Connection manager supporting network namespaces
- InfiniBand and IPoIB netdevices

Extending with RDMA cgroup



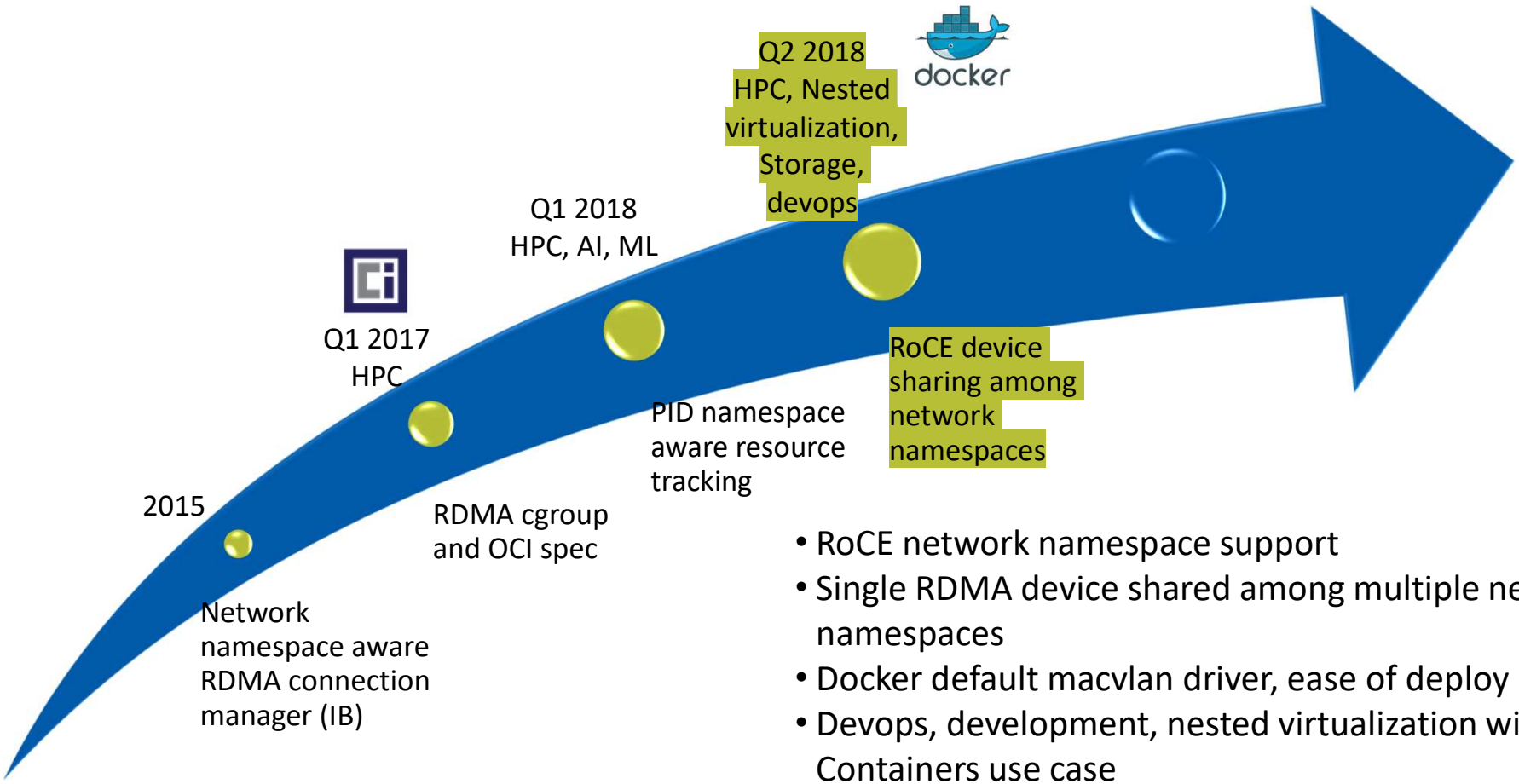
- RDMA cgroup for limiting RDMA resources
- Restrict RDMA resources per job/process
- Inclusion to OCI specification

PID namespace isolation



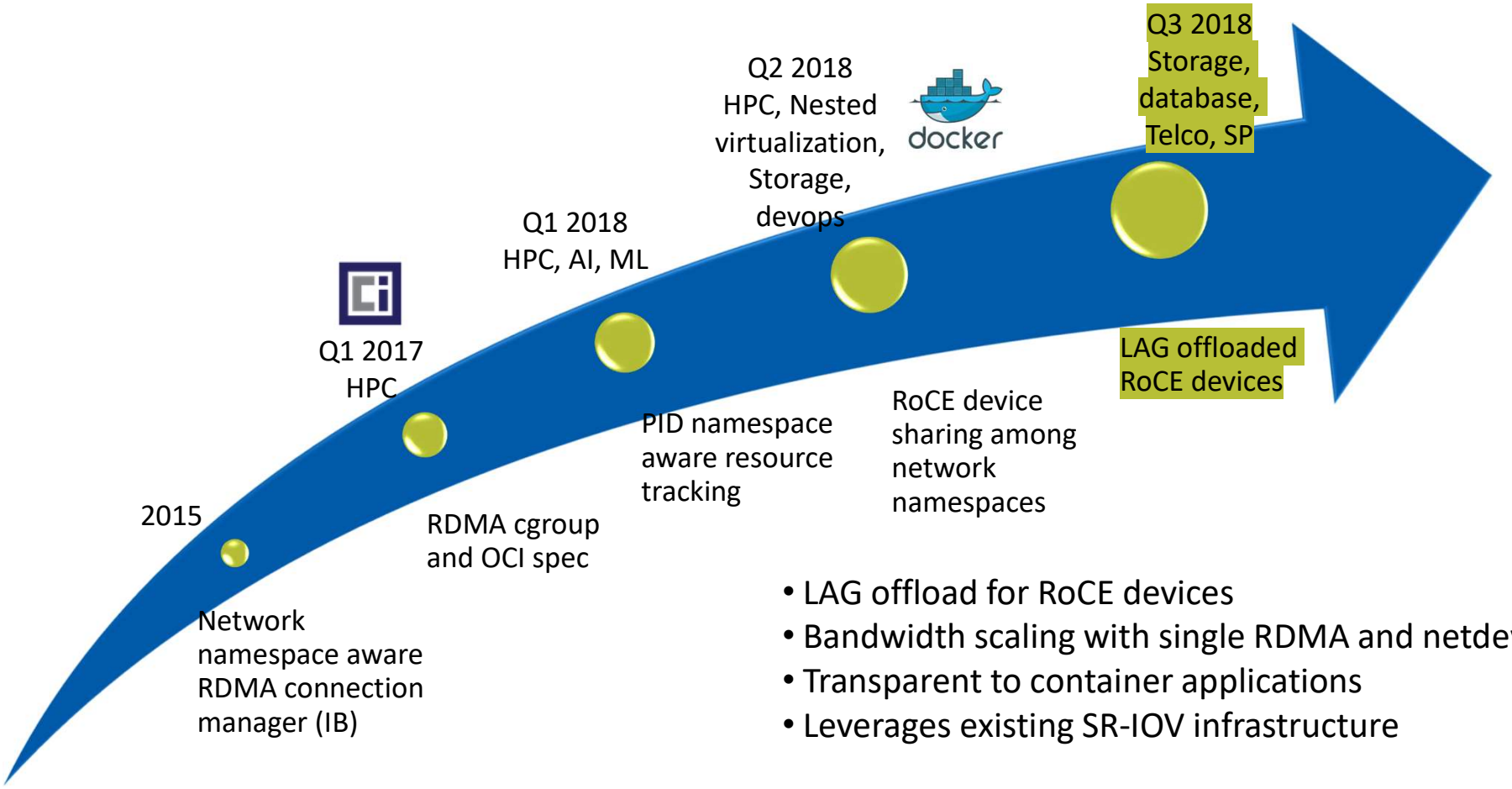
- RDMA resource monitoring per process
- iproute2/rdma tool
- RDMA resources isolated per processes of a PID namespace

RoCE network namespace support



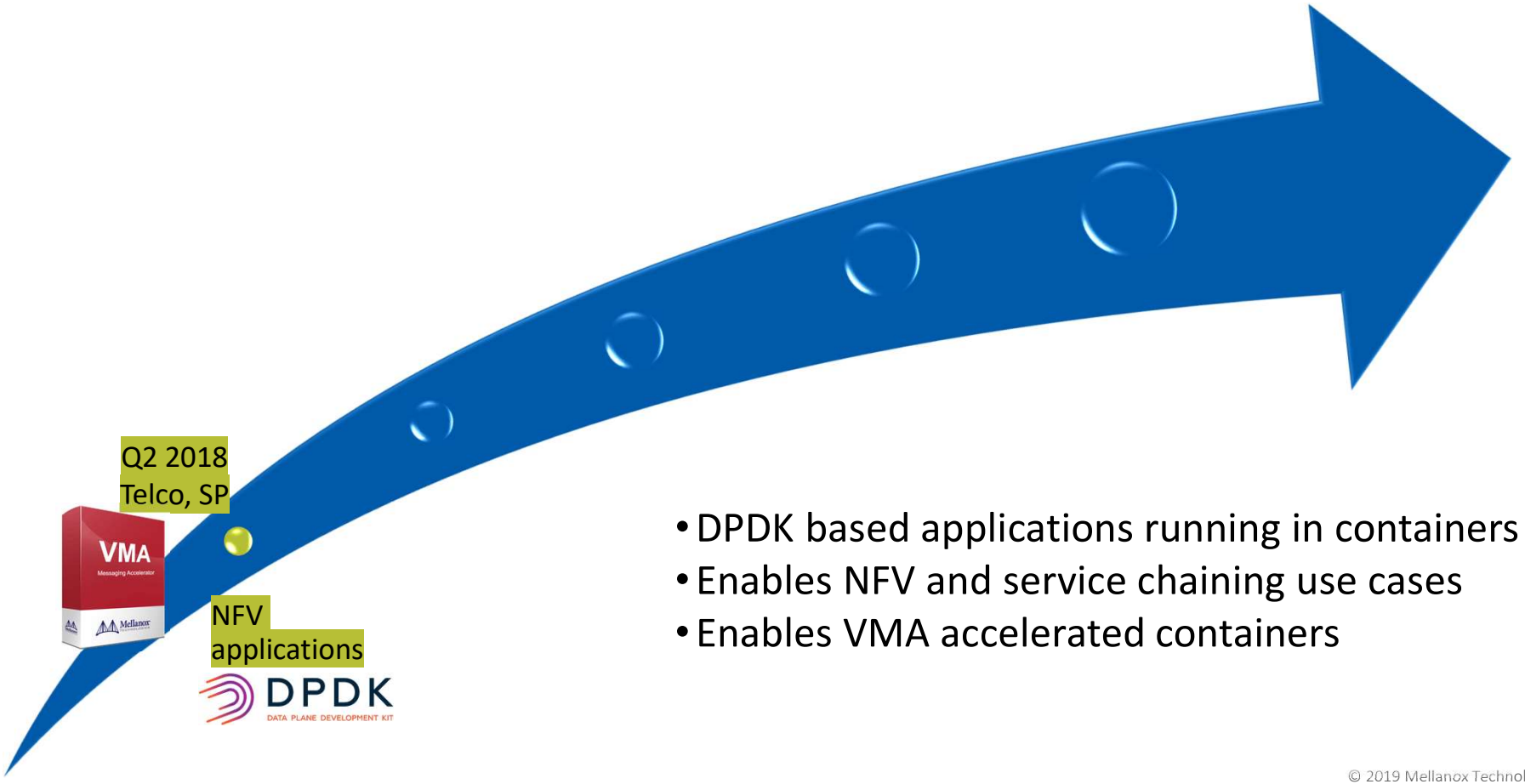
- RoCE network namespace support
- Single RDMA device shared among multiple net namespaces
- Docker default macvlan driver, ease of deploy
- Devops, development, nested virtualization with VM, Containers use case
- Docker SR-IOV network plugin

Link/Network Resilient RoCE LAG devices



- LAG offload for RoCE devices
- Bandwidth scaling with single RDMA and netdevice
- Transparent to container applications
- Leverages existing SR-IOV infrastructure

Enable DPDK based applications in container

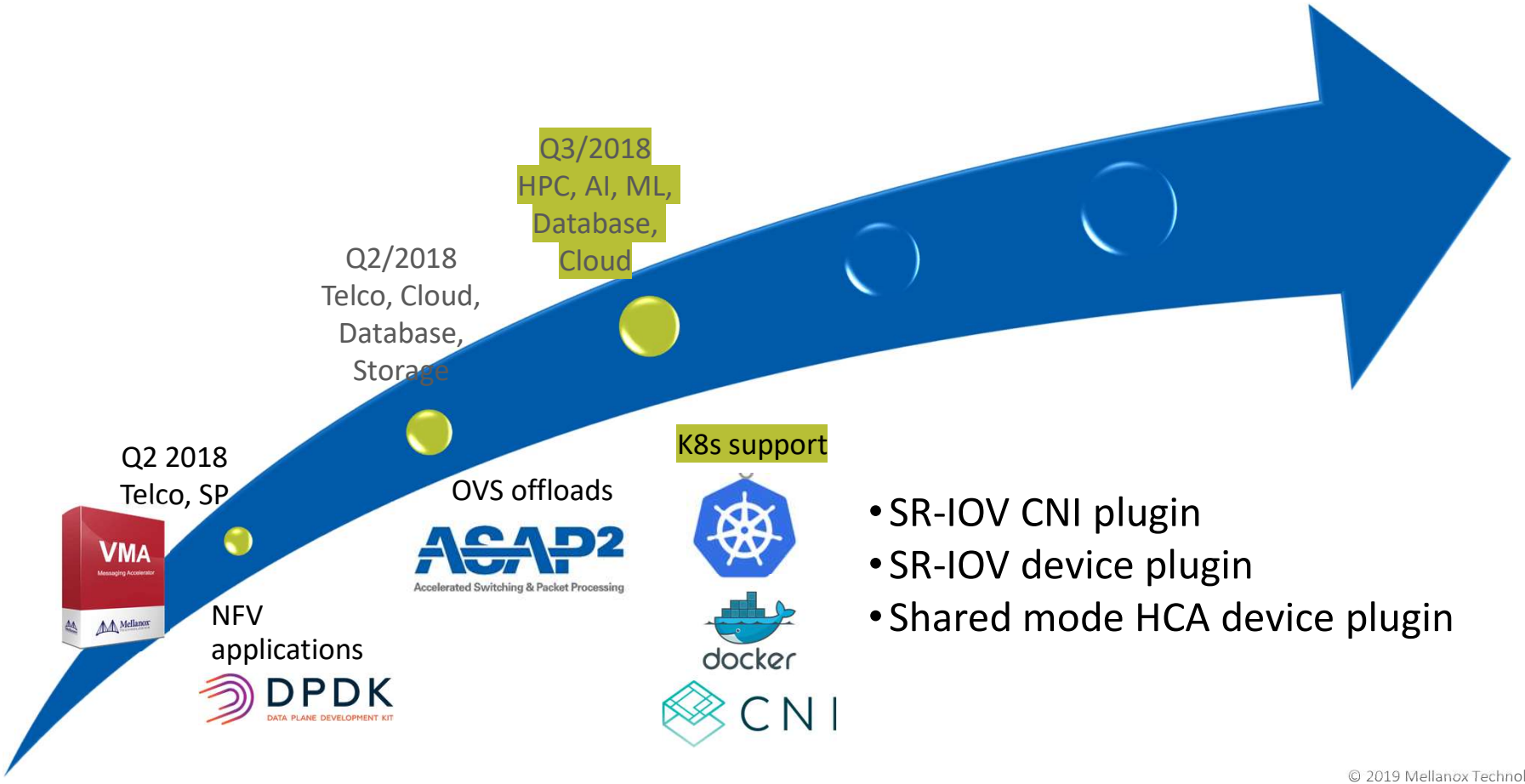


- DPDK based applications running in containers
- Enables NFV and service chaining use cases
- Enables VMA accelerated containers

OVS offloads using ASAP^2 technology

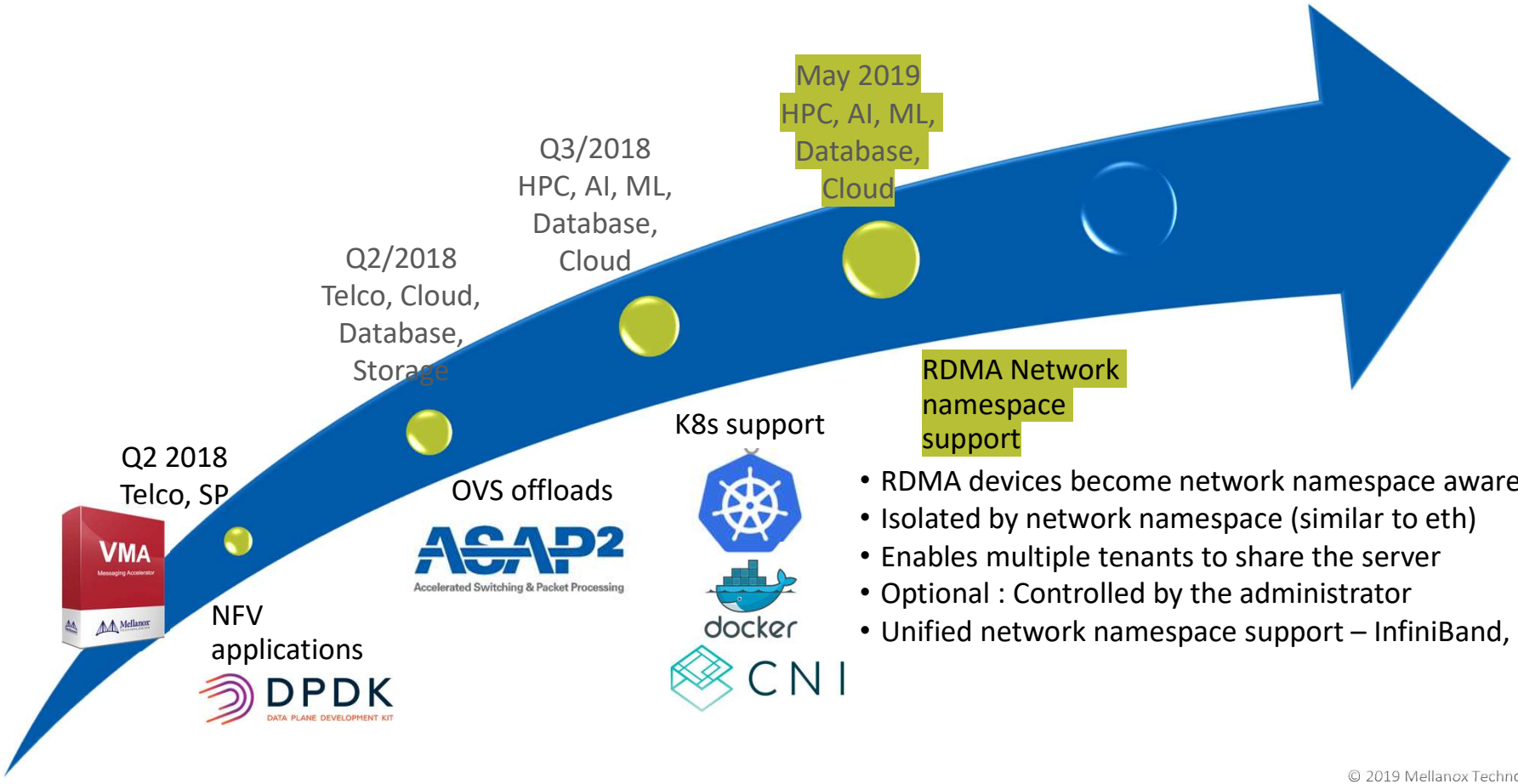
- OVS offloads with ASAP^2
  - Multi tenancy using vxlan encap/decap
  - Improves performance with offloads
  - Rich OVS policy, steering support (without plugin/CNI integration)

Kubernetes support for RDMA



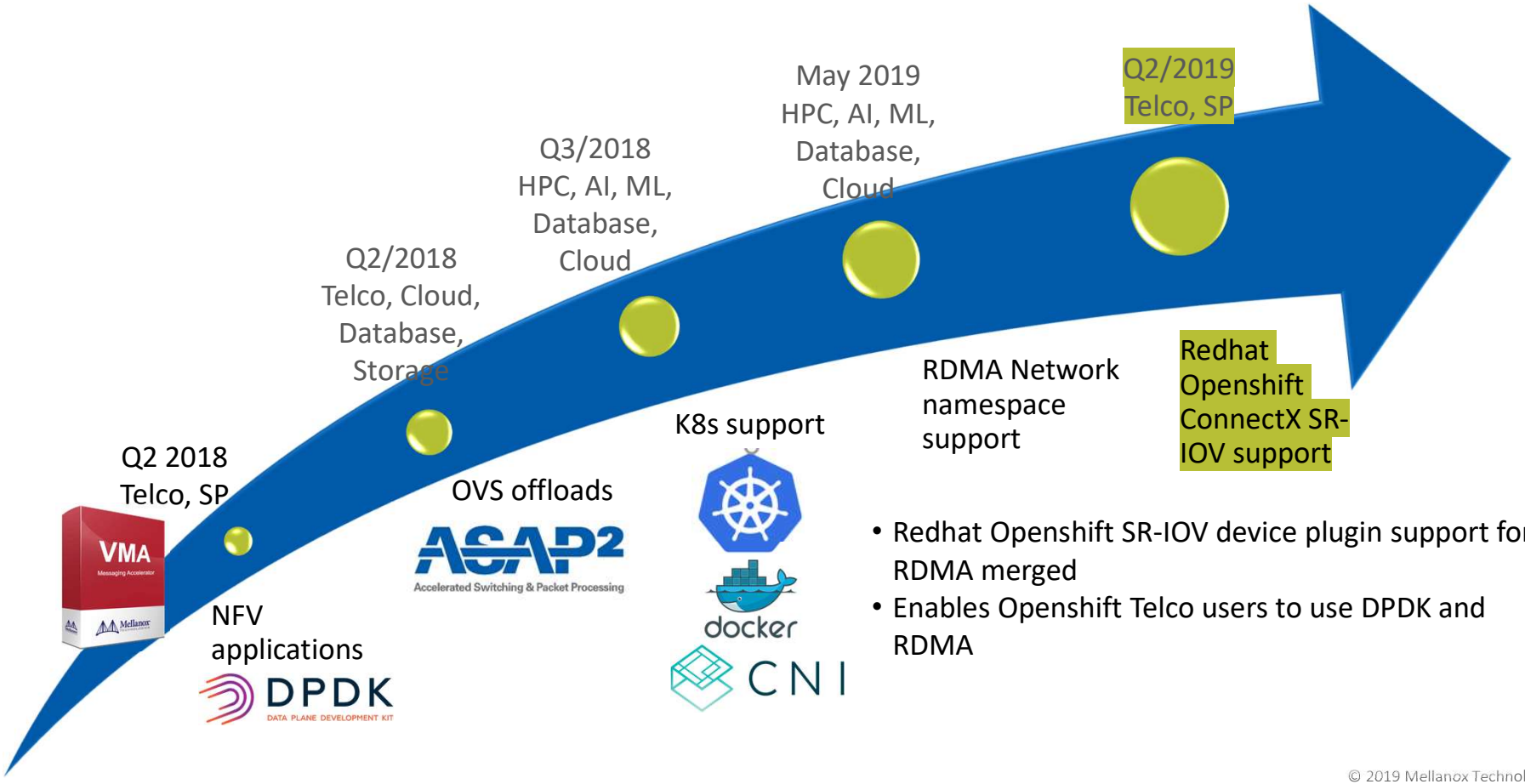
- SR-IOV CNI plugin
- SR-IOV device plugin
- Shared mode HCA device plugin

RDMA device isolation in network namespace

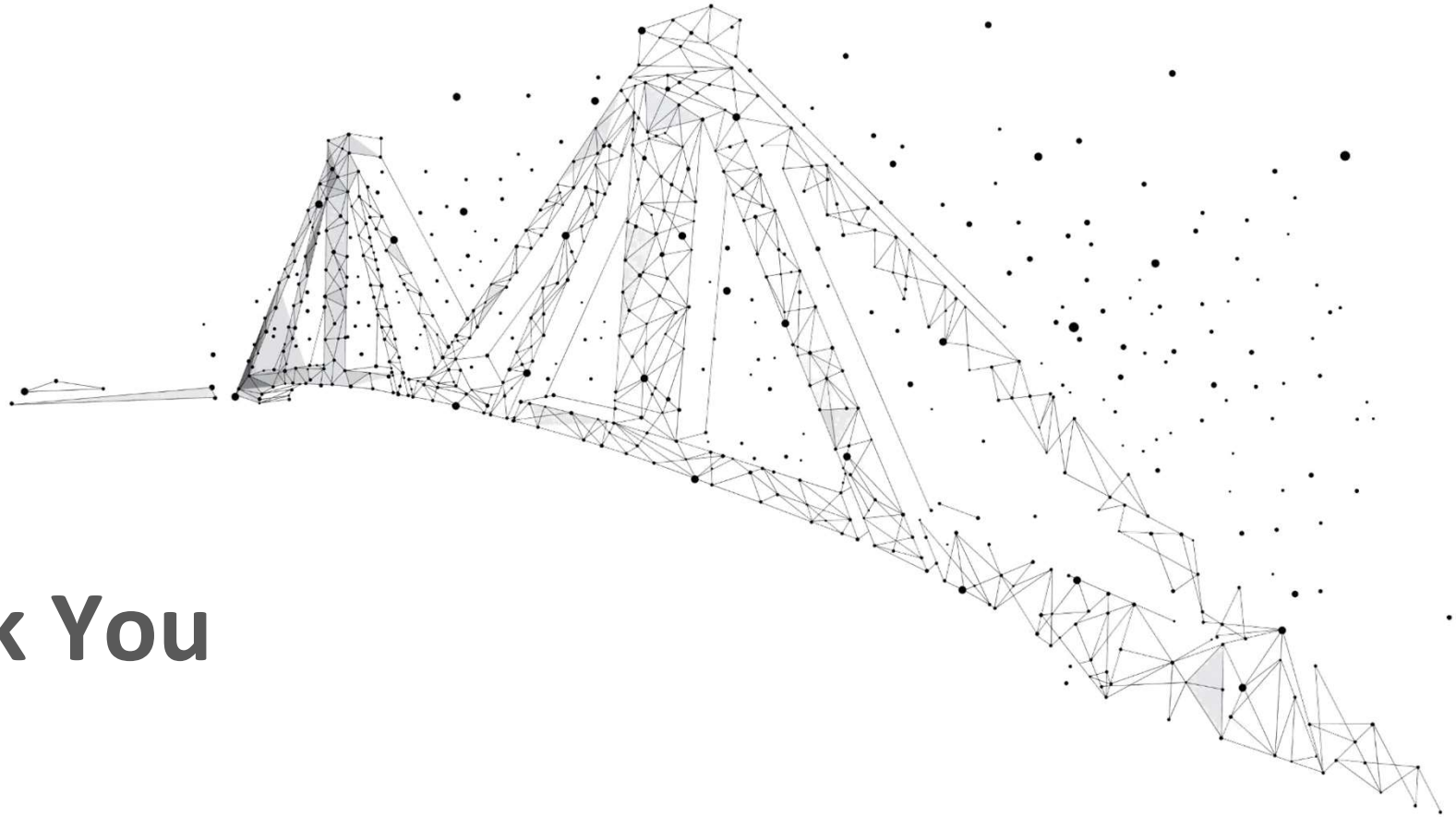


- RDMA devices become network namespace aware
- Isolated by network namespace (similar to eth)
- Enables multiple tenants to share the server
- Optional : Controlled by the administrator
- Unified network namespace support – InfiniBand, RoCE

Openshift device plugin and CNI integration



- Redhat Openshift SR-IOV device plugin support for RDMA merged
- Enables Openshift Telco users to use DPDK and RDMA



Thank You

Looking Back on the Last 5 Years of Containers

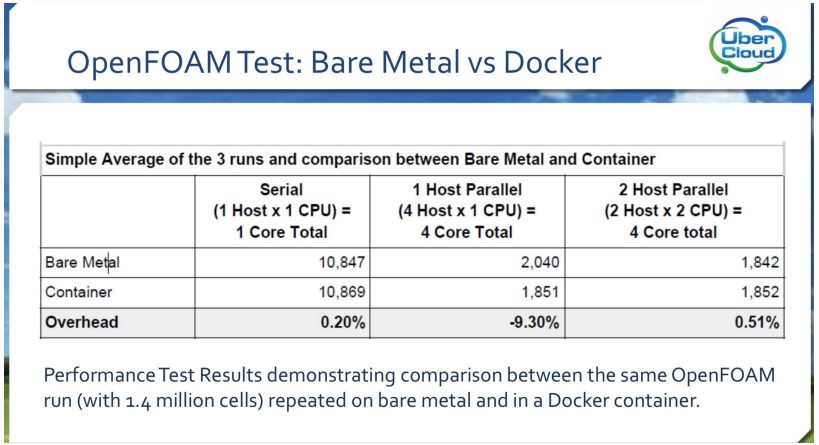
Burak Yenier
CEO, UberCloud

Who here heard about Linux Containers?

Containers - 2012 through 2014

- DevOps teams fall in love with containers as pressure to organize the from-source-to-production pipeline. Separation of concerns, growth of microservices and cloud fuel the fire.
- Containers <> VM's
- March 2013, Docker open sourced by dotCloud
- October, 2013 R.I.P dotCloud, long live Docker, Inc.
- November, 2013 UberCloud defined its HPC Container Roadmap
- October, 2014 Docker Version 1

Containers - ISC 2015 and 2016



OpenFOAM Test: Bare Metal vs Docker

Simple Average of the 3 runs and comparison between Bare Metal and Container			
	Serial (1 Host x 1 CPU) = 1 Core Total	1 Host Parallel (4 Host x 1 CPU) = 4 Core Total	2 Host Parallel (2 Host x 2 CPU) = 4 Core total
Bare Metal	10,847	2,040	1,842
Container	10,869	1,851	1,852
Overhead	**0.20%**	**-9.30%**	**0.51%**

Performance Test Results demonstrating comparison between the same OpenFOAM run (with 1.4 million cells) repeated on bare metal and in a Docker container.

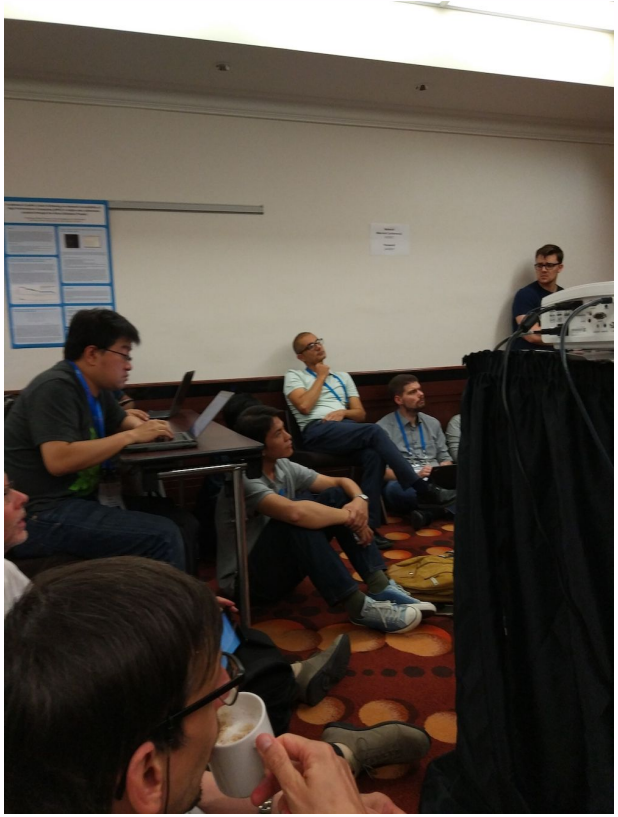
Docker containers for HPC take shape:

- Multi-node parallelism
- MPI functional, stable
- Few HPC workloads containerized
- Performance characteristics understood

- Singularity announced by LLNL
- Shifter announced by NERSC

Container workshops at ISC 2015 & 2016:
Docker: User-Friendly Application & Service Containers for HPC Environments.

Containers - ISC 2017



- Containers! Growing popularity in HPC community. Standing room only at the ISC Workshop.
- Toolset grows with Kubernetes, Singularity, Shifter, Docker, Charliecloud, others.
- Customer success stories emerge
- Orchestration: Integrations with HPC Schedulers

Containers - ISC 2018

- March 2018, Au revoir Solomon
- Singularity popularity among HPC Centers increasing.
- Containers used widely in HPC. Singularity, Shifter, Docker success stories and best practices emerge.
- Orchestration is well understood, multiple HPC schedulers support containers
- Advanced topics are well understood (GPU’s, Infiniband)
- Shortcomings, roadmaps are documented in a clear way

Containers - ISC 2019

- Containers are being used for HPC workloads on HPC Centers, Clouds, on-premise clusters.
- Orchestrators widely support containers.
- Community is aware of containers, benefits, best-practices.
- Gaps and roadmaps are clear.
- Your comments:

_____

_____

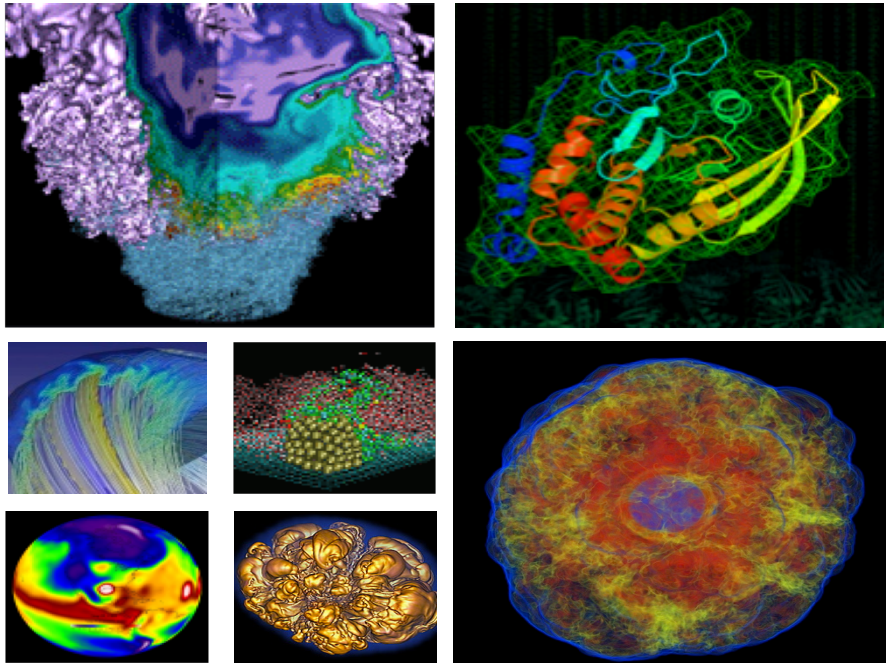
_____

_____

Thank you

burak@TheUberCloud.com

Containers at NERSC



Shane Canon
Lawrence Berkeley National Lab

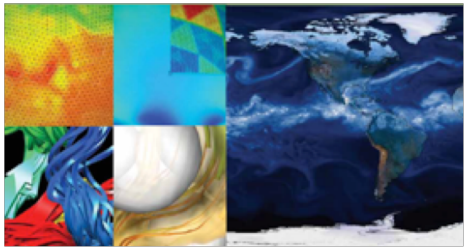
June 20, 2019

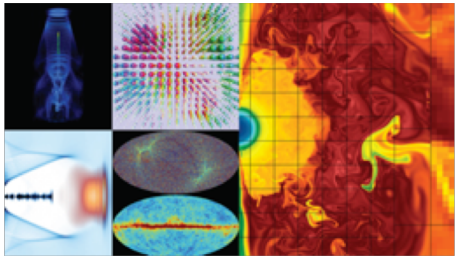
U.S. DEPARTMENT OF ENERGY | Office of Science

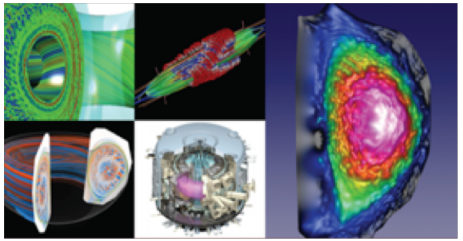
BERKELEY LAB

NERSC is the mission HPC computing center for the DOE-SC

- NERSC deploys advanced HPC and data systems for the broad Office of Science community
- NERSC staff provide advanced application and system performance expertise to users
- Approximately 7000 users and 750 projects
- Over 2000 publication resulting in NERSC resources per year
- Data Initiative: Pioneer new capabilities to enable scientists to make large-scale data-intensive science discoveries.







Supporting Data Intensive Systems and HPC

Data users and other emerging communities need the scale and unique capabilities from our HPC systems

But migrating to HPC systems can kill productivity

HPC can be Awkward

- **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 and data
- **Many groups have turned to Shifter to over come these obstacles**

Long History with Custom Environments

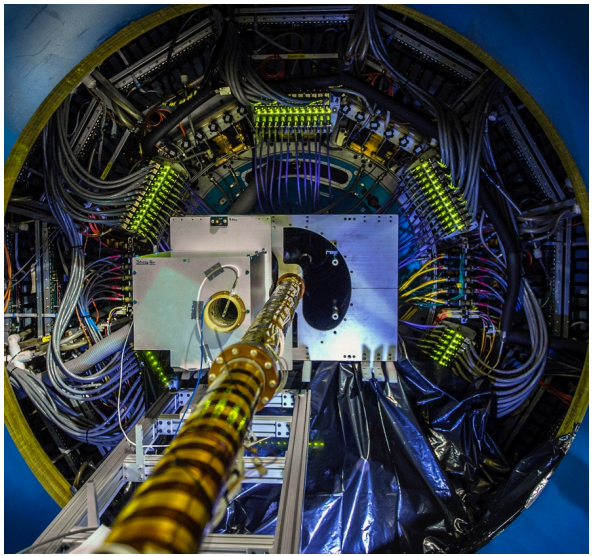
- **~2003: ChrootOS(CHOS) – System to enable projects to have customized environments on a shared cluster (PDSF). Integrated with login and batch and used a custom kernel module to provide a seamless experience.**
- **~2014: MyDock – Thin wrapper around Docker to allow users to securely run Docker on a shared cluster (Jesup/Carver).**
- **~2015: Shifter: "User Defined Environments"**

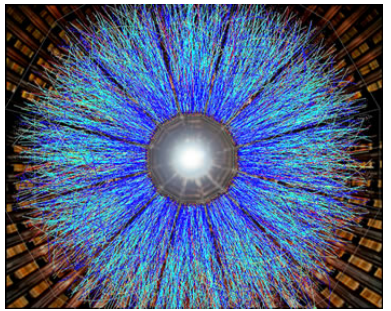
Containers and Science

- **Productivity**
  - Pick the OS that works best for your app and use the system package manager to install dependencies.
- **Reusability and Collaboration**
  - Share images across a project to avoid rebuilds and avoid mistakes
- **Reproducibility**
  - Everything you need to redo a scientific analysis can be in the image (apps, libraries, environment setup, scripts)
- **Portability**
  - Can easily run on different resources (of the same architecture)

Probing The Nucleus

- **STAR at Brookhaven, NY**
  - smashing nuclei into each other to understand their component parts
- **Data analysis and simulation**
- **Why Shifter?**
  - Difficult software dependencies (32-bit libraries)





How'd They Do It?

```
# Build STAR environment image from tarballs
FROM ringo/scientific:6.4
MAINTAINER Mustafa Mustafa <mmustafa@lbl.gov>

# RPMs
RUN yum -y install libxml2 tcsh libXpm.i686 libc.i686 libXext.i686 libXrender.i
686 libstdc++.i686 fontconfig.i686 zlib.i686 libgfortran.i686 libSM.i686 mysql-
libs.i686 gcc-c++ gcc-gfortran glibc-devel.i686 xorg-x11-xauth wget make libxml
2.so.2 gdb libXtst.{i686,x86_64} libXt.{i686,x86_64} glibc glibc-devel gcc-c++

# Dev Tools
RUN wget -O /etc/yum.repos.d/slc6-devtoolset.repo http://linuxsoft.cern.ch/cern
/devtoolset/slc6-devtoolset.repo && \
    yum -y install devtoolset-2-toolchain
COPY enable_scl /usr/local/star/group/templates/

# untar STAR OPT
COPY optstar.sl64_gcc482.tar.gz /opt/star/
COPY installstar /
RUN python installstar SL16c && \
     rm -f installstar && \
     rm -f optstar.sl64_gcc482.tar.gz

# untar ROOT
COPY rootdeb-5.34.30.sl64_gcc482.tar.gz /usr/local/star/
COPY installstar /
RUN python installstar SL16c && \
     rm -f installstar && \
     rm -f rootdeb-5.34.30.sl64_gcc482.tar.gz

# DB load balancer
COPY dbLoadBalancerLocalConfig_generic.xml /usr/local/star/packages/SL16d/StDb/
servers/

# production pipeline utility macros
COPY Hadd.C /usr/local/star/packages/SL16d/StRoot/macros/
COPY lMuDst.C /usr/local/star/packages/SL16d/StRoot/macros/
COPY checkProduction.C /usr/local/star//packages/SL16d/StRoot/macros/
```

Publically available SL6.4 image

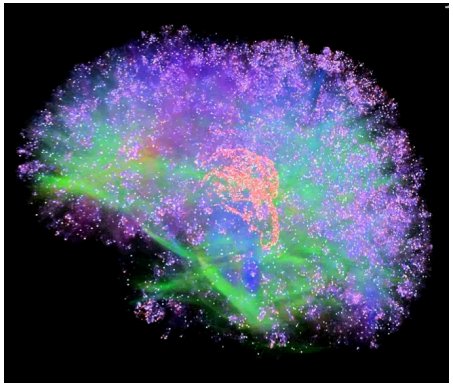
Custom STAR software:
Compiled on **another** system

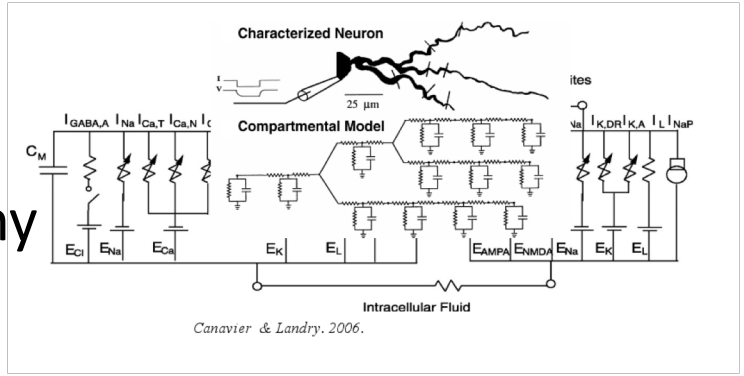
Leveraging Shifter for Easy Scalability

- **Shifter has capability to loop mount an xfs file**
  - Backed by Lustre, but all metadata actions are limited to a single node, so access is very fast
- **STAR needs to read from a ~100 MB MySQL database**
  - Running 32 individual jobs / node
- **DB on Lustre, query timed out after 30 minutes**
- **Copied DB to Shifter's xfs**
  - Initial copy ~5 minutes
  - DB Query was instantaneous
- **Used this functionality to quickly scale up without re-engineering their workflow**

Modeling the Mind

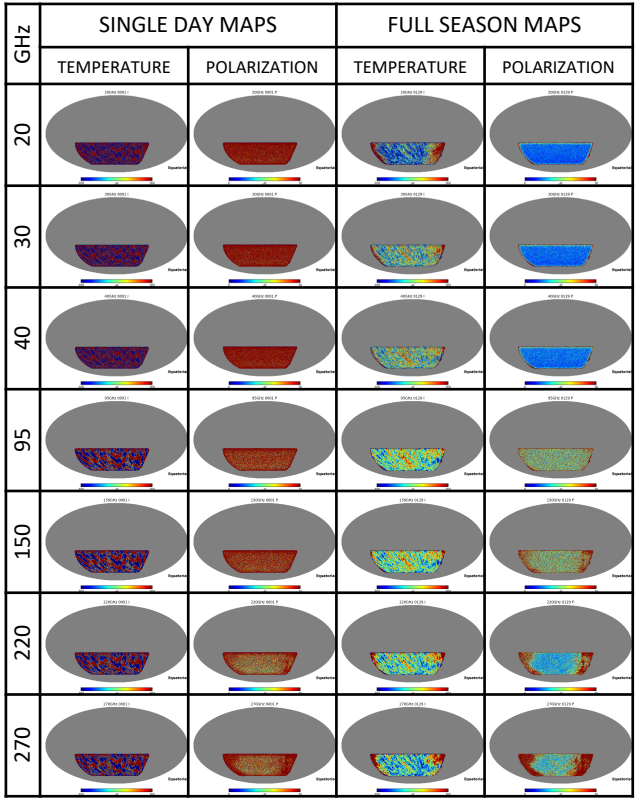
- **Neuron**
  - Simulation program to model neuron response to stimuli
- **Simulation:** 3000 points by 4000 time steps for 400 neurons
- **Why Shifter?**
  - Software developed in 1985, many older dependencies





CMB Simulation At Scale With TOAST (& Shifter)

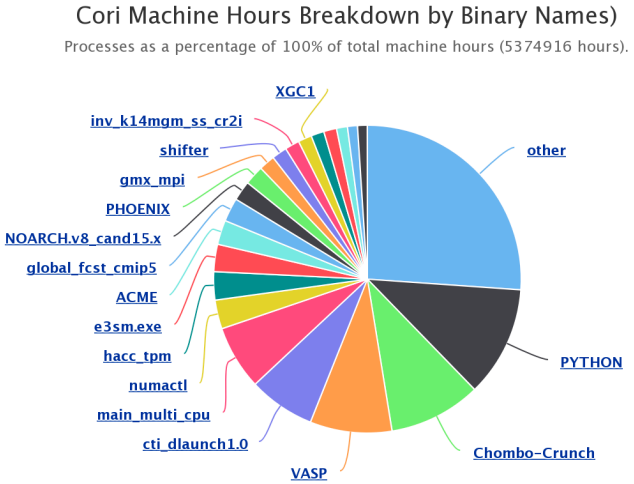
- **Simulate & map 50,000 detectors gathering 30X Planck mission data**
- **Using all of NERSC's Cori-2 supercomputer**
  - KNL optimization including Cray/Intel "dungeon session"
  - **Docker/Shifter to distribute python-wrapped code to 600,000 cores.**
- **Including realistic atmosphere simulation, instrument noise & sky signal.**
- **Re-using the expensive simulation in situ to allow multiple reduction pipelines.**
- **Meets Y1 stretch goals for LBNL LDRD project: Simulation & Analysis of CMB-S4 on Xeon Phi Systems.**
- **Animations at http://crd.lbl.gov/cmb-sims**

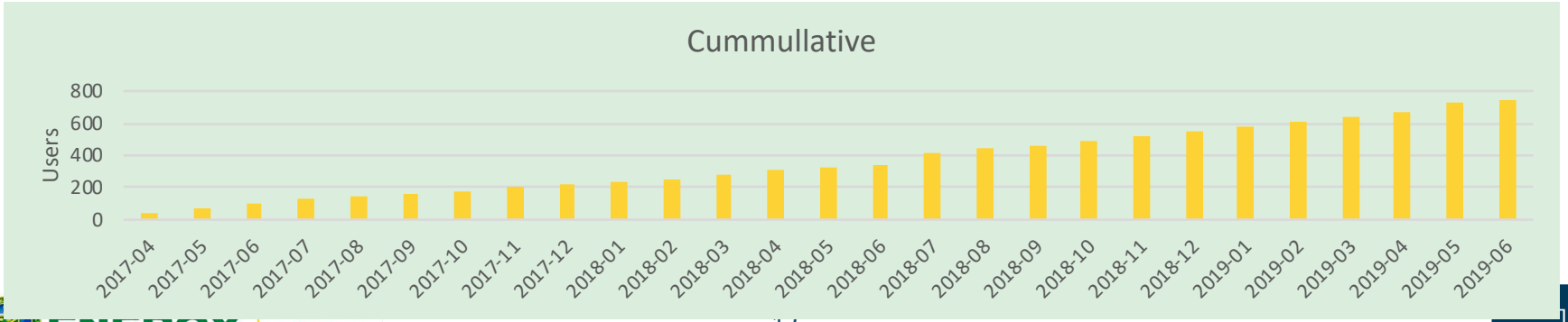


J. Borrill, R. Keskitalo & T. Kisner

Stats on Shifter Use at NERSC

- 4M Image Lookups
- 728 Unique Image tags
- 348 Unique Users
- Still a small fraction of NERSC overall use





Supporting "Edge Services" for Science

Spin* is a system to **build and deploy science gateways, workflow managers, databases, and other "edge services"** using Docker containers.

It is designed to be **flexible, scalable, and integrated tightly** with NERSC resources:

- Develop on your laptop; **deploy in minutes**.
- **Scale out** for performance.
- Access **HPC networks and file systems**.
- NERSC manages everything under the hood.

* Scalable Platform Infrastructure at NERSC

ECP SuperContainers Project

- **Led by Andrew Younge (Sandia)**
- **Focus on:**
  - Establishing Best Practices
  - Enabling Portability across Centers and Container Runtimes
  - Integrating with CI and Deployment Models
  - Training Scientists

CANOPIE-HPC Workshop

Home | Introduction | Call for Papers | Topics | Organizers | Program

CALL FOR PAPERS

Call for Papers

1st Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE HPC)

Held in conjunction with SC19: The International Conference on High Performance Computing, Networking, Storage, and Analysis, Nov 17th-22nd, 2019

Website: http://canopie-hpc.org

Submission Deadline: September 2nd, 2019

Acceptance Notice: October 2nd, 2019

Final papers due: October 7th, 2019

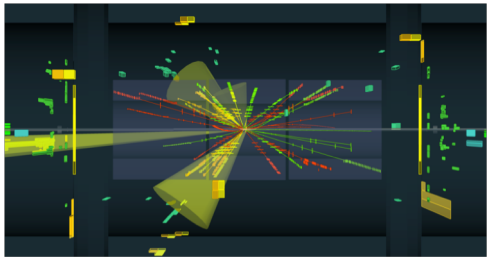
Questions

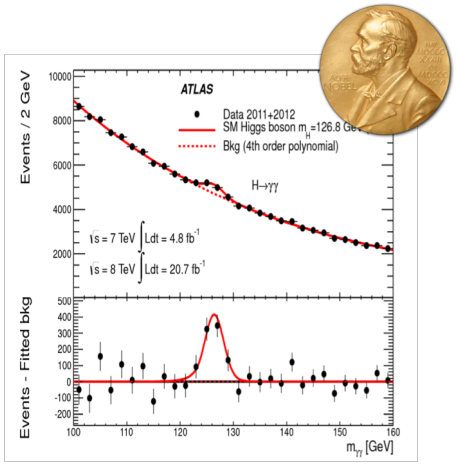
NERSC Resources

- **Cori Cray XC40**
  - Data-intensive (32-core Haswells, 128GB) partition
  - Compute-intensive (68-core KNLs, 90GB) partition
  - ~10k nodes, ~700k cores
  - High Speed Aires interconnect8 GB/s MPI bandwidth
- **High speed parallel file systems**
  - >10 PB project file system (GPFS)
  - >28 PB scratch file system (Lustre)
  - >1.5 PB Burst Buffer (flash)
- **Archive Storage**
  - 200 PB Tape Archive
- **Perlmutter – Cray Shasta (2020)**

Probing the Fundamentals of Matter

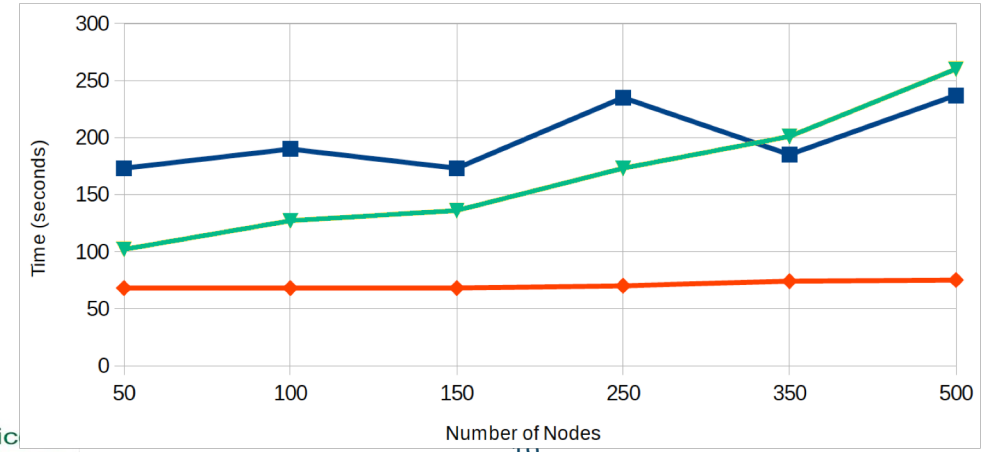
- **Large Hadron Collider (LHC)**
  - 300 trillion proton-proton collisions and 30 PBs of data per year.
- **Data analysis, simulation, multi-site data and computing pool**
- **Why Shifter?**
  - Complicated software stack: Needs FUSE and elevated permissions to run
  - Integrated framework for running with images at all computing sites



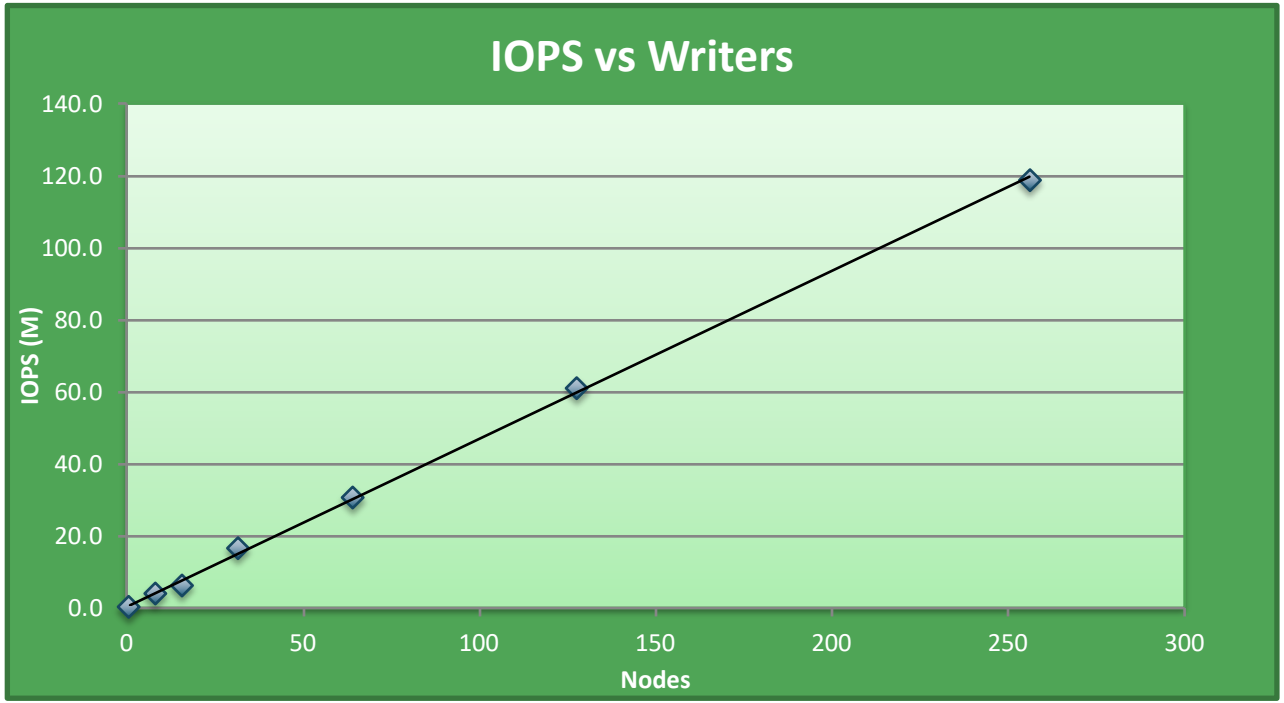


Creating a Monster

- The software stack is very big: 3.5 TB and 20M inodes
- Compress and deduplicate with squashfs: 350 GB
- Start up time shows excellent scaling out to 500 nodes (16,000 cores)



Per-Node Write Cache (IOPS)



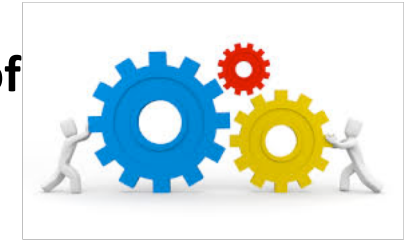
Results of an IOR File per-process, 2 tasks per node, 512B transfer size, 2GB write. 100x faster than Lustre at the same scale.

Why not just run Docker

- **Security: Docker currently uses an all or nothing security model. Users would effectively have system privileges**

```
> docker run -it -v /:/mnt --rm busybox
```

- **System Architecture: Docker assumes local disk**
- **Integration: Docker doesn't play nice with batch systems.**
- **System Requirements: Docker typically requires a very modern kernel**
- **Complexity: Running real Docker would add new layers of complexity**
- **Scale: Stock Docker would create a several scaling issues.**



Our Solution: Shifter

Design Goals and Principles:

- **Compatibility with Docker Images and Docker Registries**
- **Security – Enable site to configure and enforce policies**
- **Ability to run at scale on NERSC systems**
- **User independence: Require no administrator assistance to pull and use an image**
- **Full access to shared resource availability (e.g., file systems and network interfaces)**
- **Seamless user experience**
- **Open Source (https://github.com/nersc/shifter)**

Design Choices

- **Convert Docker images to a squashed format and loop mount the image**
  - Avoids adding additional metadata load on the PFS. Scales much better.
  - Conversion only needs to be done once and image is immediately visible across the system.
- **Centralized Image Management**
  - Prevents image corruption after mounting.
  - Prevents malicious user from crafting an image that could be used to attack the kernel.
- **Containers run as the user and images are mounted read-only**
- **Site based configuration**
  - Enable the site to limit what mount points the user can volume mount in the container.
  - Provides added security and further limits exposure.

Basic Idea

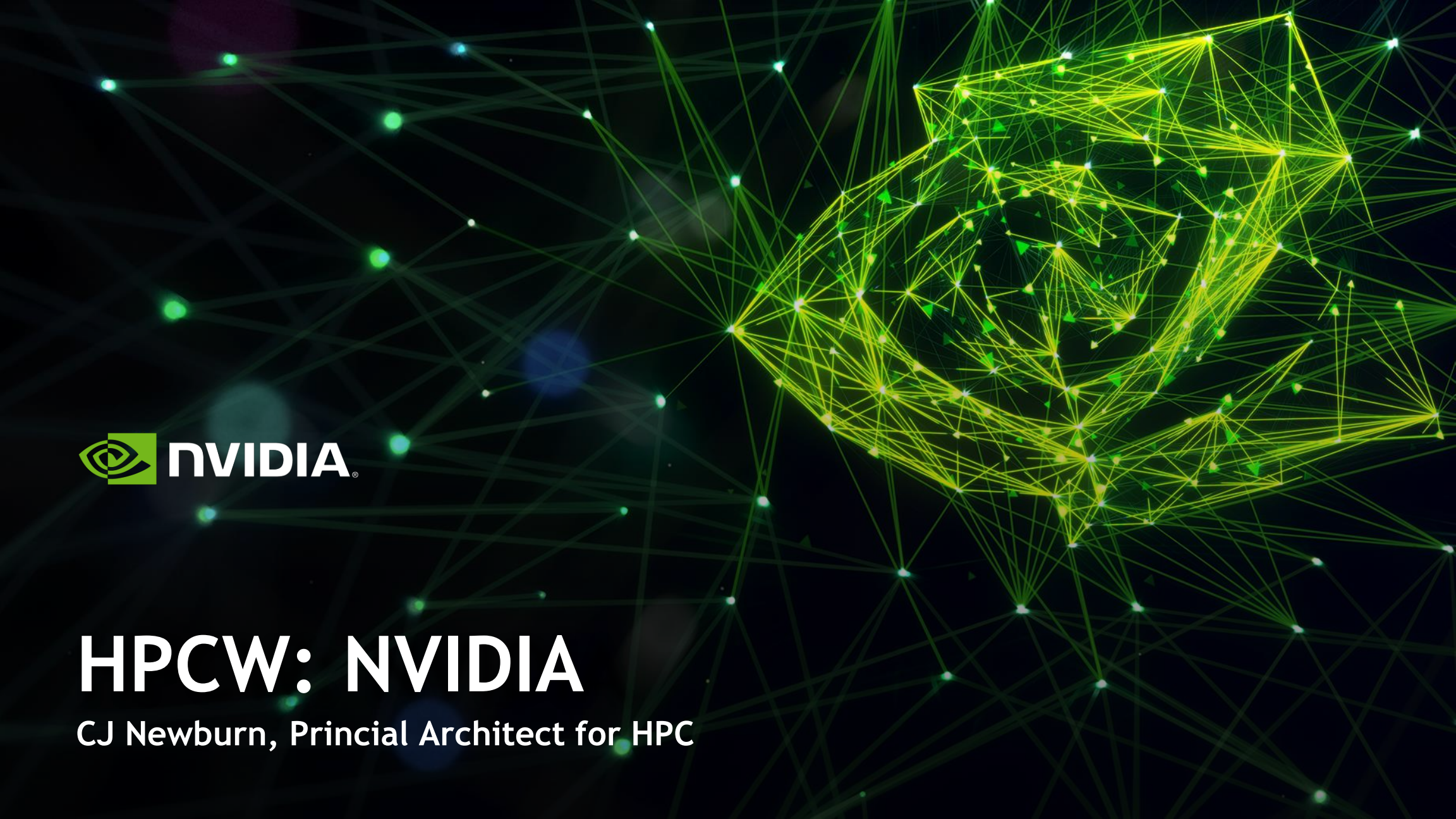
Users use Docker to create and build images

Once for each image:

1. Download the contents of an image
2. Unpack the contents
3. “Flatten” the image into a squashfs file
4. Copy that squash file to a global/cluster file system

At Runtime:

1. Mount the image with a loop back mount at run-time



HPCW: NVIDIA

CJ Newburn, Princial Architect for HPC

GOALS

- Broad container technology support
- Multiple optimized GPU architectures within a single image
- Optimized multi-node cluster support
- Smallest image size possible

CONTAINER TECHNOLOGY SUPPORT

- HPCCM outputs Docker files, Singularity recipe files, shell scripts
- NGC images are in Docker format, convertible to Singularity
  - Docker format enables multi-stage builds more easily
  - Singularity supports pulling from NGC natively
    - $ singularity pull docker://nvcr.io/hpc/lammps:24Oct2018
- A custom entrypoint handles setup in a way that's usable by Docker and Singularity
  - Must assume image FS is read-only vs. writing to /usr/lib vs. bind-mounted dir (mofed)
  - User is whoever starts the container, no sudo, no apt get install
- Documentation provides application specific best practices for both runtimes
  - Examples of running canonical problems/benchmarks provided

CONTAINER TECHNOLOGY ENABLING

- Desires:
  - Common plugin across various container runtimes
- Challenges:
  - Plugin may have limited or only predefined capabilities
  - Want to copy in files from host system (libcuda.so, nvidia-smi.so), bind mount, query inside container for compatibility (e.g. CUDA 9.x/driver 384), set LD_LIBRARY_PATH to use files in container that are compatible with a specific set of kernel mode drivers
- Creative ideas
  - Make plugin parameterizable, e.g. suppress some actions if scheduler pleasant
  - Container technology has different levels of trust for plugins

MULTI-TARGET SUPPORT

- Single image optimized for Pascal/Volta/Turing when possible
- nvcc can create multi-arch GPU binary targeting all desired architectures
- If build system doesn't support multi-arch compilation, use multiple bins
- entrypoint validates and selects correct binary based on host GPU

MULTINODE: OPENMPI

- Plugin/component-based architecture makes it very flexible
  - Many NGC images use OpenMPI which supports Slurm, PMI2, PMIx, UCX
  - Whereas MPICH seems to require static compile-time config
  - Most decisions made at runtime, ideal for portable containers
- Provides robust GPU-aware MPI support

- Use of .la metadata files inhibits our flexibility via rpath mechanism

MULTINODE: UCX

- Alternate choices
  - IB component is default starting with OpenMPI/4.0
  - Or could use legacy OpenIB byte transfer layer
  - Can compare perf between these without recompilation
- UCX features
  - IB, GDRcopy, CUDA IPC, xpmem, knem, cma optimized transports
  - Picks optimized transport at runtime based upon host capabilities
    - Compile-time decisions based upon detection of MOFED on host
    - Only enables GDRcopy if GDRcopy kernel modules available
    - Requires shipping multiple versions in the container

MULTINODE: INFINIBAND

- Support for Mellanox InfiniBand through MOFED/RDMA-Core
  - User/Kernel driver components not cross version compatible until 4.4+
- Support GPU extensions(nv_peer_mem, gdrcopy)
- Passing in host driver libs can be problematic due to varying transitive dependencies
  - rhel libnl.so <≠> ubuntu libnl-3.so

MULTINODE: INFINIBAND

- Package multiple MOFED/RDMA-Core releases within container
- Selection handled by entrypoint application

- Relocate libibverbs, libdapl, librdmacm, libmlx4, libmlx5
- Read host kernel driver version from /sys/module/mlx5_core/version
- Set LD_LIBRARY_PATH to best matching libibverbs, libdapl, librdmacm libraries
- Set IBV_DRIVERS to point to best matching libmlx4, libmlx5 driver libraries

MULTINODE: PMI

- Glue between resource mgrs, process managers, and processes
- Three common APIs(PMI, PMI2, PMIx)
- Implementations not ABI compatible, even within same API
- PMIx/3.x has robust backwards compatibility and solves many container issues
- PMIx/3.x supported by OpenMPI and Slurm
- PMI2 support useful for legacy Slurm integration

MULTINODE: LAUNCH WITH HOST MPIRUN

- $ mpirun cmd
- $ mpirun singularity run --nv nvidia.simg cmd

- Pros
  - Familiar interface: prefix cmd with Singularity
  - Maintains integration with host resource manager
- Cons
  - Requires compatible host OpenMPI/PMI installation
  - OpenMPI/4.x with PMIx provide good cross version compatibility
  - External mpi may default to using components not in container build

MULTINODE: LAUNCH WITH HOST SRUN

- $ srun cmd
- $ srun --mpi=pmix singularity run --nv nvidia.simg cmd
- $ srun --mpi=pmi2 singularity run --nv nvidia.simg cmd

- Pros
  - Familiar interface: prefix cmd with Singularity and set PMI
  - Maintains integration with host resource manager
- Cons
  - Requires compatible PMI installation; Slurm PMI2 available on most systems

MULTINODE: LAUNCH W/ CONTAINER MPIRUN

- $ mpirun cmd

- HOSTFILE=".hostfile.${SLURM_JOB_ID}"
  for host in $(scontrol show hostnames); do
    echo "${host}" >> ${HOSTFILE}
  done
- OMPI_MCA_plm=rsh
  OMPI_MCA_plm_rsh_args='-o PubkeyAcceptedKeyTypes=+ssh-dss -o ...'
  OMPI_MCA_orte_launch_agent="singularity run --nv nvidia.simg orted"
- singularity run nvidia.simg mpirun --hostfile $HOSTFILE cmd

MULTINODE: LAUNCH W/ CONTAINER MPIRUN

- Pros
  - Works on most systems without external compatibility issues
  - Workload is better contained, better reproducibility
- Cons
  - No integration with host resource manager
  - Exact SSH arguments depend on host specifics

IMAGE SIZE

- Image size important to users and administrators alike
- Heavy use of Docker multi-stage builds to ensure smallest image possible
- Use tools such as dive to audit image size
- LAMMPS container ~100MB, single "baremetal" binary ~70MB