Toward Full Specialization of the HPC System Software Stack: Reconciling Application Containers and Lightweight Multi-kernels

Balazs Gerofi†, Yutaka Ishikawa†, Rolf Riesen‡, Robert W. Wisniewski‡

bgerofi@riken.jp

†RIKEN Advanced Institute for Computational Science, JAPAN

‡Intel Corporation, USA

2017/Jun/27 -- ROSS'17 Washington, D.C.

Agenda

- Motivation
- Full system software stack specialization
- Overview of container concepts
- conexec: integration with lightweight multi-kernels
- Results
- Conclusion

Motivation – system software/OS challenges for high-end HPC (and for converged BD + HPC stack?)

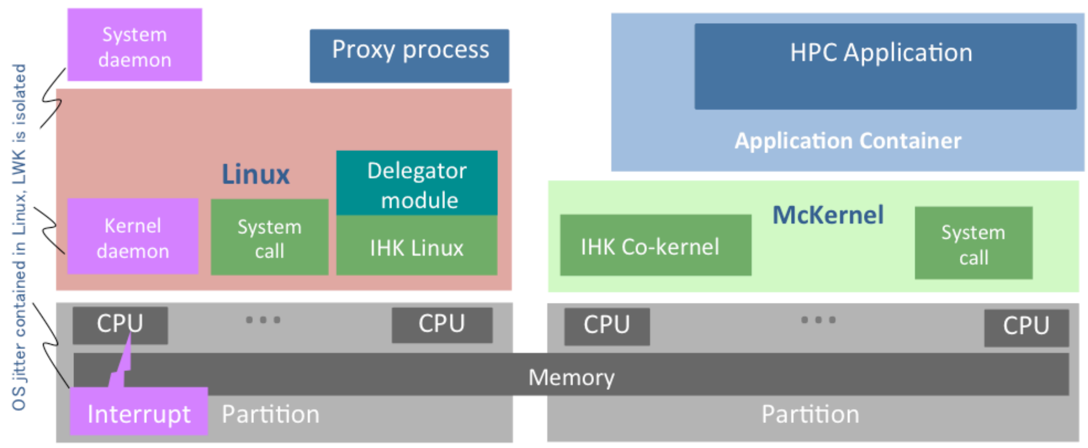
- **Node architecture: increasing complexity and heterogeneity**
  - Large number of (heterogeneous) CPU cores, deep memory hierarchy, complex cache/NUMA topology
- **Applications: increasing diversity**
  - Traditional/regular HPC + in-situ data analytics + Big Data processing + Machine Learning + Workflows, etc.

- **What do we need from the system software/OS?**
  - Performance and scalability for large scale parallel apps
  - Support for Linux APIs – tools, productivity, monitoring, etc.
  - Full control over HW resources
  - Ability to adapt to HW changes
    - Emerging memory technologies, power constrains
  - Performance isolation and dynamic reconfiguration
    - According to workload characteristics, support for co-location

RIKEN

Approach: embrace diversity and complexity

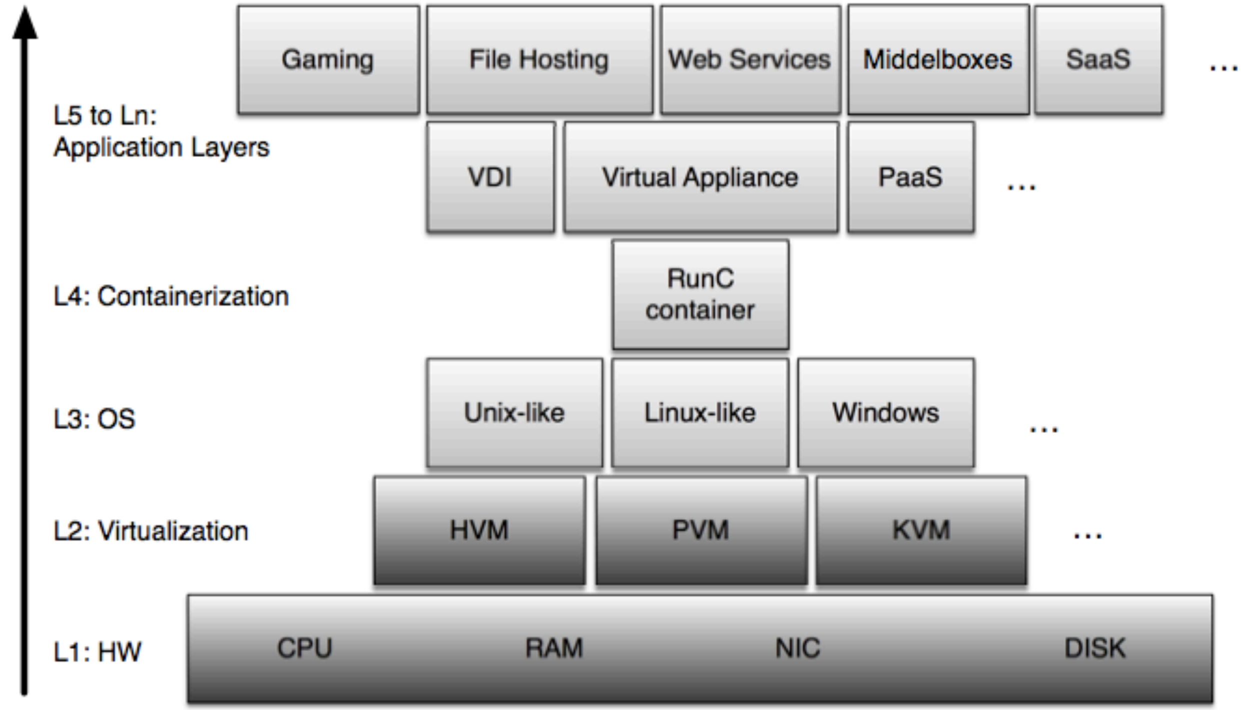
- **Enable *dynamic specialization of the system software stack* to meet application requirements**
  - User-space: Full provision of libraries/dependencies for all applications will likely not be feasible:
    - Containers (i.e., namespaces) – specialized user-space stack
  - Kernel-space: Single monolithic OS kernel that fits all workloads will likely not be feasible:
    - Specialized kernels that suit the specific workload
    - Lightweight multi-kernels for HPC



Linux Container Concepts

Are containers the new narrow waist?

- **BDEC community's view of how the future of the system software stack may look like**
- **Based on: the hourglass model**
  - The narrow waist "used to be" the POSIX API



[1] Silvery Fu, Jiangchuan Liu, Xiaowen Chu, and Yueming Hu. **Toward a standard interface for cloud providers: The container as the narrow waist.** *IEEE Internet Computing*, 20(2):66–71, 2016.

Linux Namespaces

- **A namespace is a “scoped” view of kernel resources**

- **mnt (mount points, filesystems)**
- **pid (processes)**
- **net (network stack)**
- **ipc (System V IPC, shared mems, message queues)**
- **uts (hostname)**
- **user (UIDs)**

- **Namespaces can be created in two ways:**
  - During process creation
    - clone() syscall
  - By “unsharing” the current namespace
    - unshare() syscall

Linux Namespaces

- **The kernel identifies namespaces by special symbolic links (every process belongs to exactly one namespace for each namespace type)**
  - /proc/PID/ns/*
  - The content of the link is a string: namespace_type:[inode_nr]

- **A namespace remains alive until:**
  - There are any processes in it, *or*
  - There are any references to the NS file representing it

```
bgerofi@vm:~/containers/namespaces# ls -ls /proc/self/ns
total 0
0 lrwxrwxrwx 1 bgerofi bgerofi 0 May 27 17:52 ipc -> ipc:[4026531839]
0 lrwxrwxrwx 1 bgerofi bgerofi 0 May 27 17:52 mnt -> mnt:[4026532128]
0 lrwxrwxrwx 1 bgerofi bgerofi 0 May 27 17:52 net -> net:[4026531957]
0 lrwxrwxrwx 1 bgerofi bgerofi 0 May 27 17:52 pid -> pid:[4026531836]
0 lrwxrwxrwx 1 bgerofi bgerofi 0 May 27 17:52 user -> user:[4026531837]
0 lrwxrwxrwx 1 bgerofi bgerofi 0 May 27 17:52 uts -> uts:[4026531838]
```

Mount Namespace

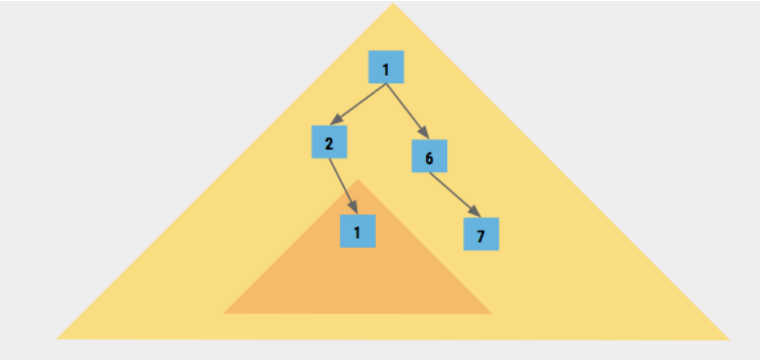
- **Provides a new scope of the mounted filesystems**
- **Note:**
  - Does not remount the /proc and accessing /proc/mounts won't reflect the current state unless remounted
    - mount proc –t proc /proc –o remount
  - /etc/mtab is only updated by the command line tool "mount" and not by the mount() system call

- **It has nothing to do with chroot() or pivot_root()**

- **There are various options on how mount points under a given namespace propagate to other namespaces**
  - Private
  - Shared
  - Slave
  - Unbindable

PID Namespace

- **Provides a new PID space with the first process assigned PID 1**
- **Note:**
  - "ps x" won't show the correct results unless /proc is remounted
    - Usually combined with mount NS




```
bgerofi@vm:~/containers/namespaces$ sudo ./mount+pid_ns /bin/bash
bgerofi@vm:~/containers/namespaces# ls -ls /proc/self
0 lrwxrwxrwx 1 bgerofi bgerofi 0 May 27  2016 /proc/self -> 3186
bgerofi@vm:~/containers/namespaces# umount /proc; mount proc -t proc /proc/
bgerofi@vm:~/containers/namespaces# ls -ls /proc/self
0 lrwxrwxrwx 1 bgerofi bgerofi 0 May 27 18:39 /proc/self -> 56
bgerofi@vm:~/containers/namespaces# ps x
  PID TTY      STAT   TIME COMMAND
    1 pts/0    S      0:00 /bin/bash
   57 pts/0    R+     0:00 ps x
```

cgroups (Control groups)

- **The cgroup (control groups) subsystem does:**
  - Resource management
    - It handles resources such as memory, cpu, network, and more
  - Resource accounting/tracking
  - Provides a generic process-grouping framework
    - Groups processes together
    - Organized in trees, applying limits to groups

- **Development was started at Google in 2006**
  - Under the name "process containers”
- **v1 was merged into mainline Linux kernel 2.6.24 (2008)**
- **cgroup v2 was merged into kernel 4.6.0 (2016)**

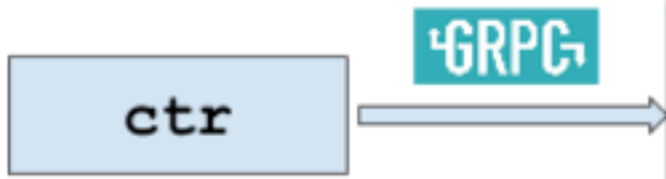
- **cgroups I/F is implemented as a filesystem (cgroupfs)**
  - e.g.: mount -t cgroup -o cpuset none /sys/fs/cgroup/cpuset

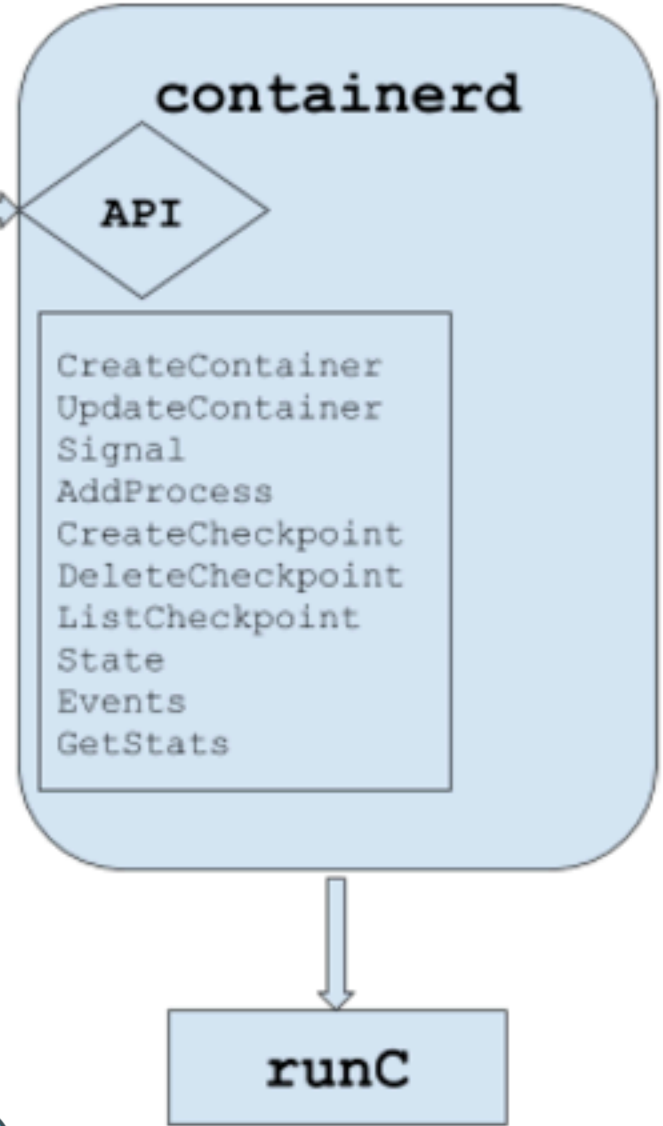
- **Configuration is done via cgroup controllers (files)**
  - 12 cgroup v1 controllers and 3 cgroup v2 controllers

Some cgroup v1 controllers

Controller/subsystem	Kernel object name	Description
blkio	io_cgrp_subsys	sets limits on input/output access to and from block devices such as physical drives (disk, solid state, USB, etc.)
cpuacct	cpuacct_cgrp_subsys	generates automatic reports on CPU resources used by tasks in a cgroup
cpu	cpu_cgrp_subsys	sets limits on the available CPU time
cpuset	cpuset_cgrp_subsys	assigns individual CPUs (on a multicore system) and memory nodes to tasks in a cgroup
devices	devices_cgrp_subsys	allows or denies access to devices by tasks in a cgroup
freezer	freezer_cgrp_subsys	suspends or resumes tasks in a cgroup
hugetlb	hugetlb_cgrp_subsys	controls access to hugeTLBfs
memory	memory_cgrp_subsys	sets limits on memory use by tasks in a cgroup and generates automatic reports on memory resources used by those tasks

Docker Architecture





- **Docker client talks to daemon (http)**
- **Docker daemon prepares root file system and creates config.json descriptor file**
- **Calls runc with the config.json**
- **runc does the following steps:**
  - Clones a new process creating new namespaces
  - Sets up cgroups and adds the new process
- **New process:**
  - Re-mounts pseudo file systems
  - pivot_root() into root file system
  - execve() container entry point

Singularity Container

- **Very simple HPC oriented container**
- **Uses primarily the mount namespace and chroot**
  - Other namespaces are optionally supported
- **No privileged daemon, but *sexec* is setuid root**

- **http://singularity.lbl.gov/**

- **Advantage:**
  - Very simple package creation
    - v1: Follows dynamic libraries and automatically packages them
    - v2: Uses bootstrap files and pulls OS distributions
      - No longer does dynamic libraries automatically

- **Example: mini applications:**
    - 59M May 20 09:04 /home/bgerofi/containers/singularity/miniapps.sapp
      - Uses Intel's OpenMP and MPI from the OpenHPC repository
    - Installing all packages needed for the miniapps requires 7GB disk space

Shifter Container Management

- **NERSC's approach to HPC with Docker**
- **https://bitbucket.org/berkeleylab/shifter/**

- **Infrastructure for using and distributing Docker images in HPC environments**
- **Converts Docker images to UDIs (user defined images)**
  - Doesn't run actual Docker container directly

- **Eliminates the Docker daemon**
- **Relies only on mount namespace and chroot**
  - Same as Singularity

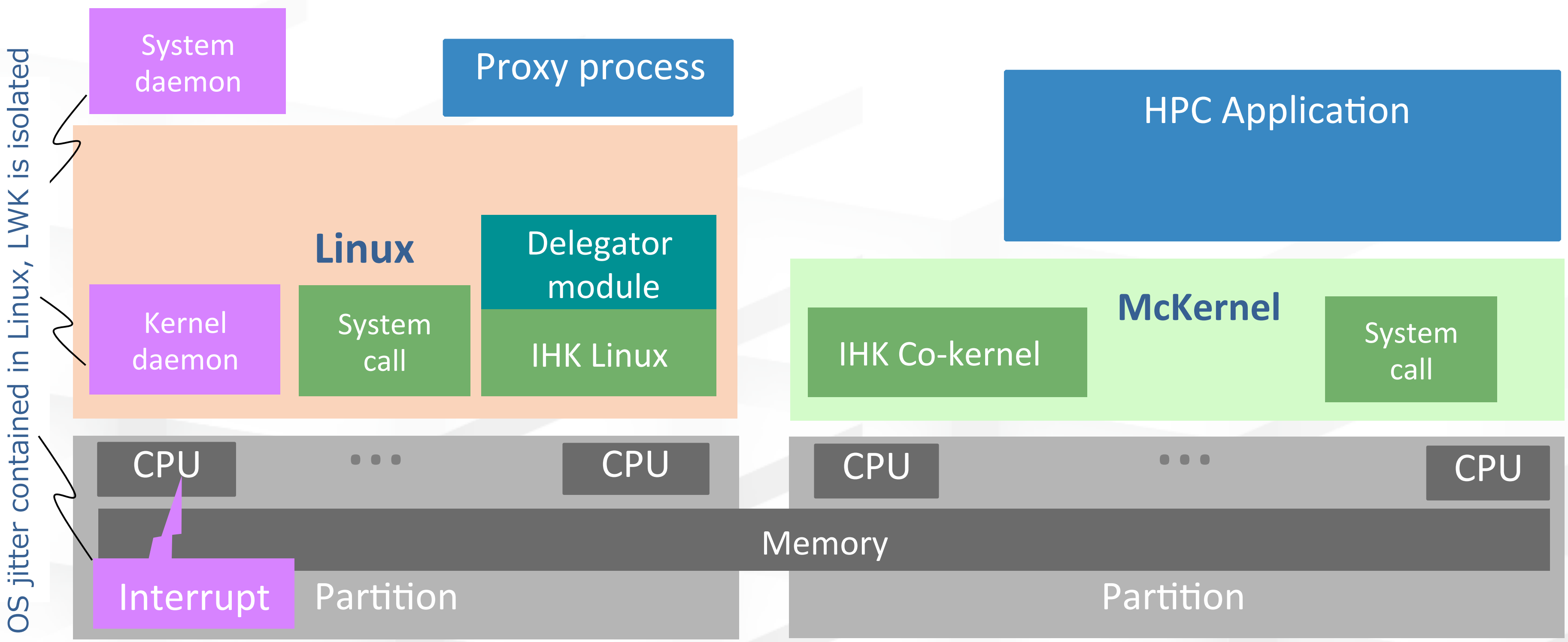
Comparison of container technologies

Project/ Attribute	Docker	rkt	Singularity	Shifter
Supports/uses namespaces	yes	yes	mainly mount (PID optionally)	only mount
Supports cgroups	yes	yes	no	no
Image format	OCI	appc	sapp (in-house)	UDI (in-house)
Industry standard image	yes	yes	yes (convertible)	no
Daemon process required	yes	no	no	no
Network isolation	yes	yes	no	no
Direct device access	yes	yes	yes	yes
Root FS	pivot_root()	chroot()	chroot()	chroot()
Implementation language	Go	Go	C, python, sh	C, sh

Integration of containers and lightweight multi-kernels

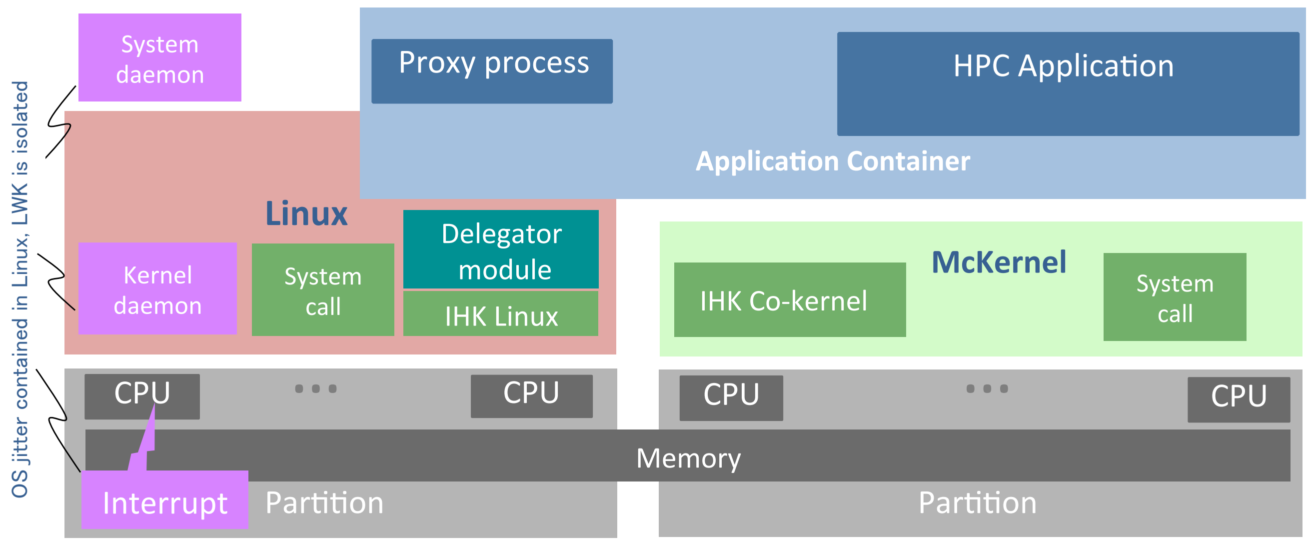
IHK/McKernel Architectural Overview

- **Interface for Heterogeneous Kernels (IHK):**
  - Allows dynamic partitioning of node resources (i.e., CPU cores, physical memory, etc.)
  - Enables management of multi-kernels (assign resources, load, boot, destroy, etc..)
  - Provides inter-kernel communication (IKC), messaging and notification
- **McKernel:**
  - A lightweight kernel developed from scratch, boots from IHK
  - Designed for HPC, noiseless, simple, implements only performance sensitive system calls (roughly process and memory management) and the rest are offloaded to Linux

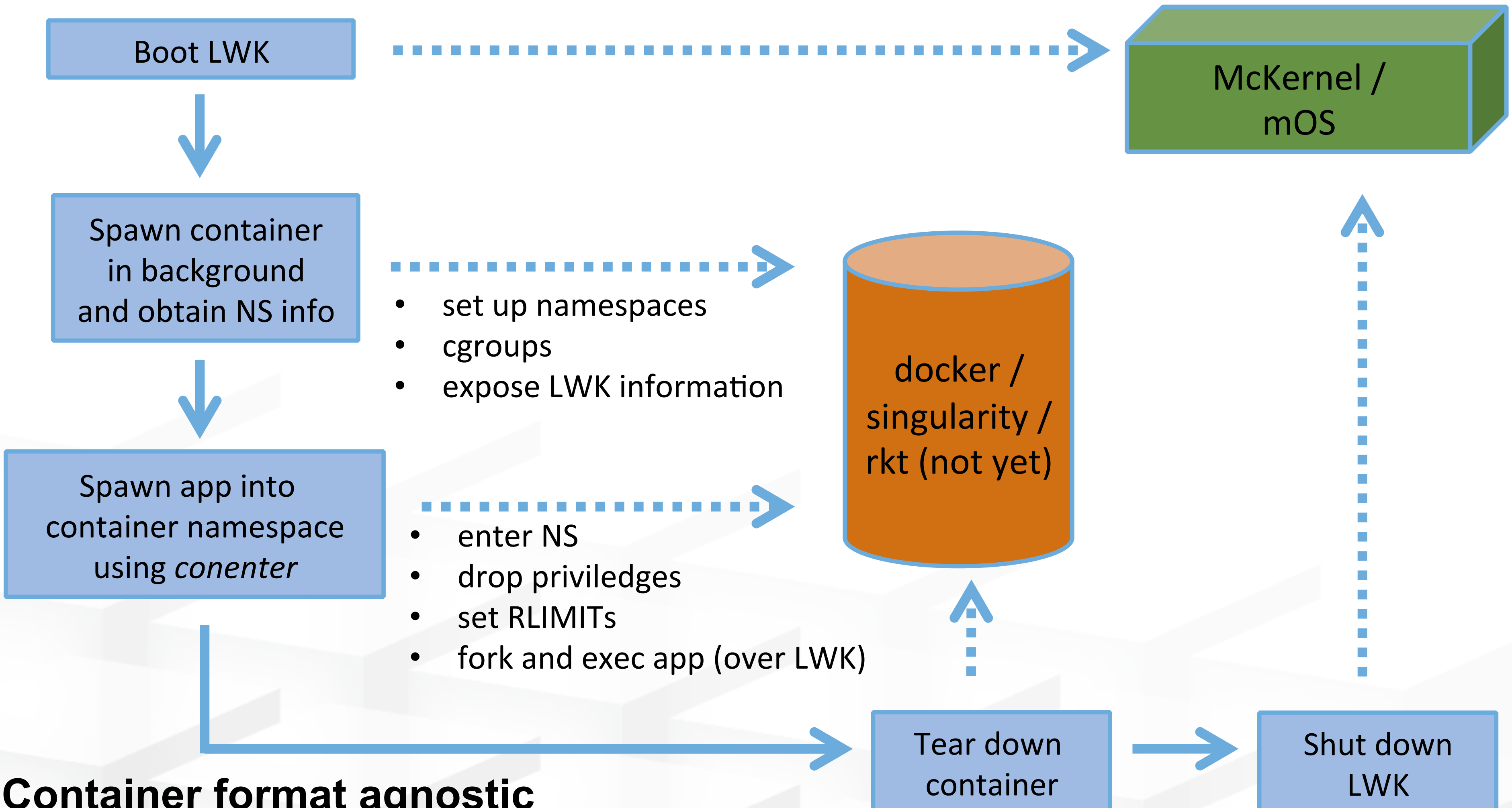


IHK/McKernel with Containers -- Architecture

- **Proxy runs in Linux container's namespace(s)**
  - Some modifications were necessary to IHK to properly handle namespace scoping inside the Linux kernel
- **IHK device files need to be exposed in the container**
  - Bind mounting /dev/mcdX and /dev/mcosX
- **McKernel specific tools (e.g., mcexec) also need to be accessible in the container**
  - Similar to IB driver, GPU driver issues (more on this later)



conexec/conenter: a tool based on setns() syscall



- **Container format agnostic**
- **Naturally works with mpirun**
- **User needs no privileged operations (almost)**
  - McKernel booting currently requires insmod

conexec/conenter: a tool based on setns() syscall

- **conexec (options) [container] [command] (arguments)**

- **options:**
  - --lwk: LWK type (mckernel|mos)
  - --lwk-cores: LWK CPU list
  - --lwk-mem: LWK memory (e.g.: 2G@0,2G@1)
  - --lwk-syscall-cores: System call CPUs
- **container: protocol://container_id**
  - e.g.:
    - docker://ubuntu:tag
    - singularity:///path/to/file.img

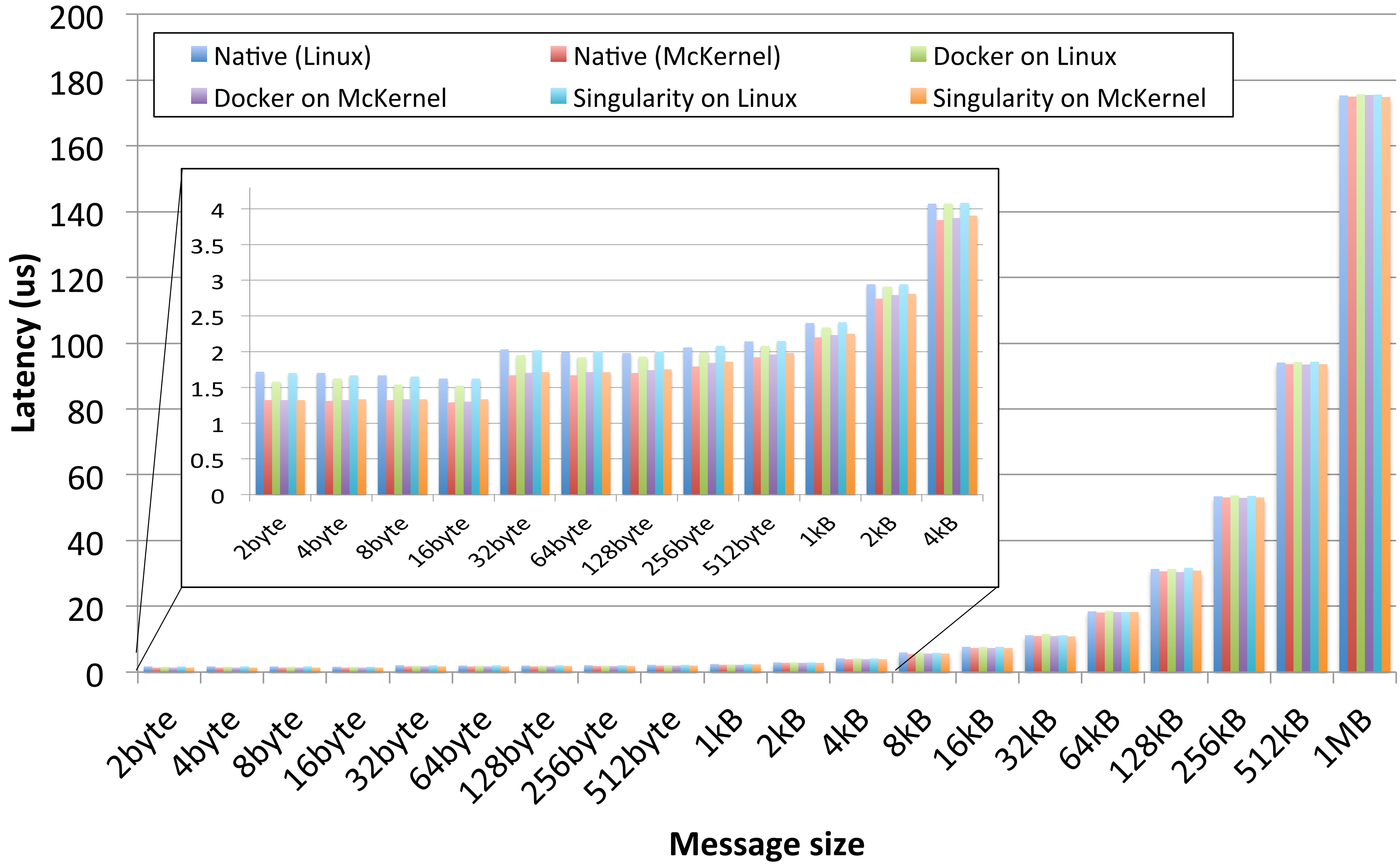
- **Running with MPI:**
  - mpirun -genv I_MPI_FABRICS=dapl -f hostfile -n 16 -ppn 1 /home/bgerofi/Code/conexec/conexec --lwk mckernel --lwk-cores 10-19 --lwk-mem 2G@0 singularity:///home/bgerofi/containers/singularity2/miniapps.img /opt/IMB_4.1/IMB-MPI1 Allreduce

Preliminary Evaluation

- **Platform1: Xeon cluster with Mellanox IB ConnectX2**
  - 32 nodes, 2 NUMA / node, 10 cores / NUMA
- **Platform2: Oakforest PACS**
  - 8k Intel KNL nodes
  - Intel OmniPath interconnect
  - ~25 PF (6th on 2016 Nov Top500 list)
- **Intel Xeon Phi CPU 7250 model:**
  - 68 CPU cores @ 1.40GHz
  - 4 HW thread / core
    - 272 logical OS CPUs altogether
  - 64 CPU cores used for McKernel, 4 for Linux
  - 16 GB MCDRAM high-bandwidth memory
  - 96 GB DRAM
  - SNC-4 flat mode:
    - 8 NUMA nodes (4 DRAM and 4 MCDRAM)
- **Linux 3.10 XPPSL**
  - nohz_full on all application CPU cores

- **Containers**
  - Ubuntu 14.04 in Docker and Singularity
  - Infiniband and OmniPath drivers contained

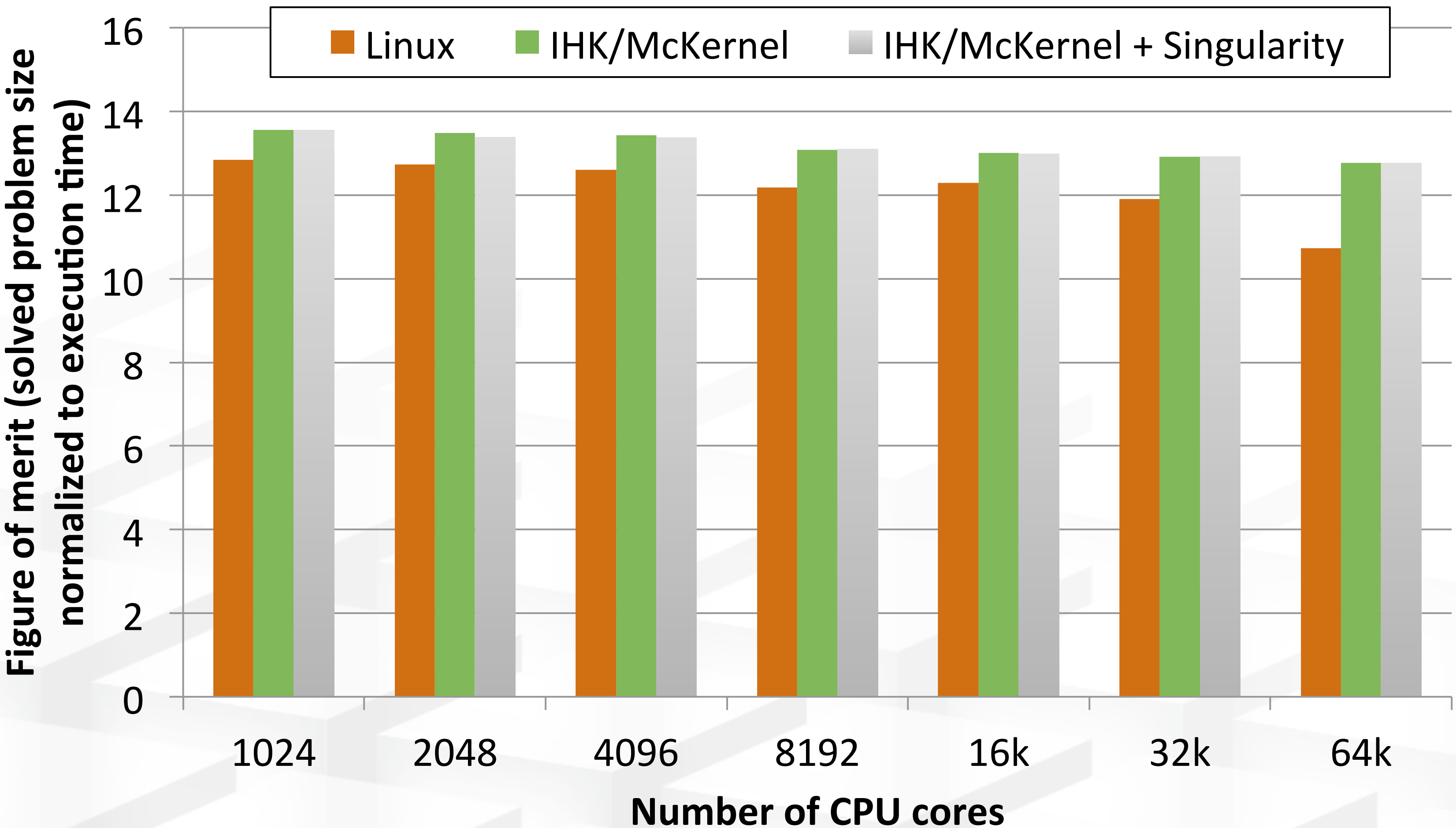
IMB PingPong – Containers impose ~zero overhead



- **Xeon E5-2670 v2 @ 2.50GHz + MLNX Infiniband MT27600 [Connect-IB] + CentOS 7.2**
- **Intel Compiler 2016.2.181, Intel MPI 5.1.3.181**
- **Note: IB communication entirely in user-space!**

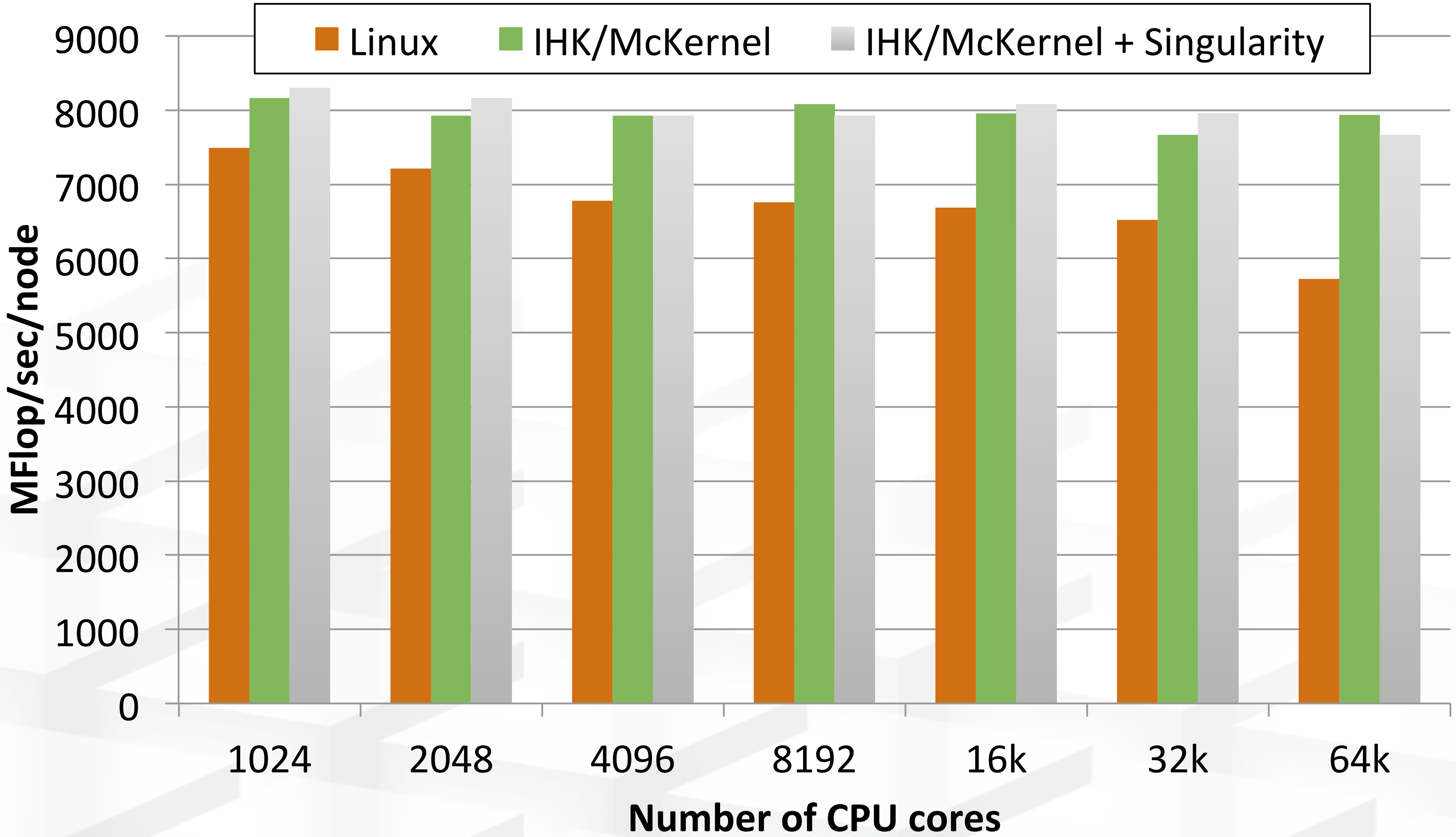
GeoFEM (University of Tokyo) in container

- **Stencil code – weak scaling**
- **Up to 18% improvement**



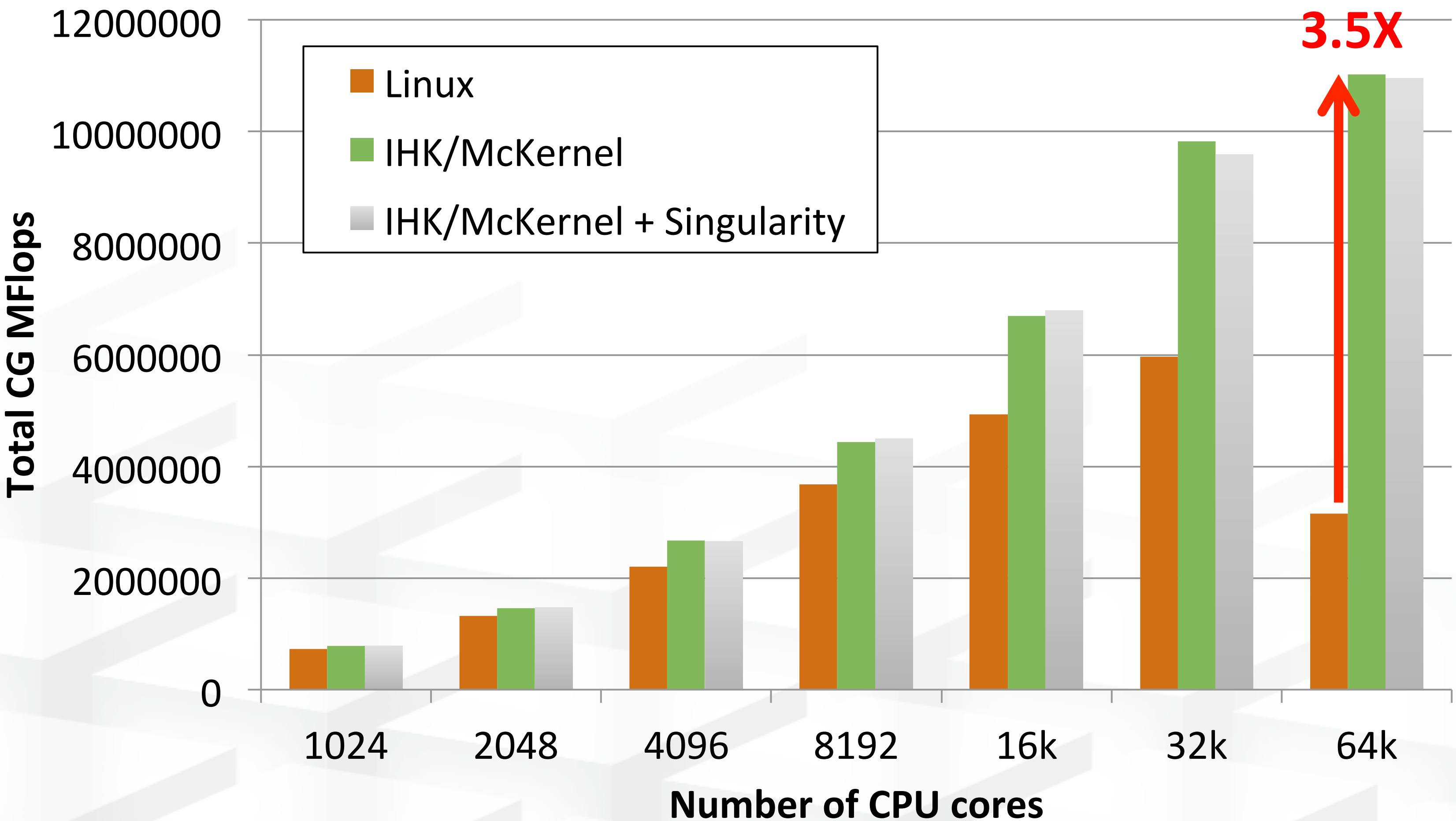
CCS-QCD (Hiroshima University) in container

- Lattice quantum chromodynamics code - weak scaling
- Up to 38% improvement



miniFE (CORAL benchmark suite) in container

- **Conjugate gradient - strong scaling**
- **Up to 3.5X improvement (Linux falls over..)**



RIKEN

Containers' limitations (or challenges) in HPC

- **User-space components need to match kernel driver's version**
  - E.g.: libmlx5-rdmav2.so needs to match IB kernel module
  - Workaround: dynamically inject libraries into container.. ?
    - Intel MPI and OpenMPI do dlopen() based on the driver env. variable
    - MPICH links directly to the shared library
    - *Is it still a "container" if it accesses host specific files? Reproducibility?*
  - E.g.: NVIDIA GPU drivers, same story..

- **mpirun on the spawning host needs to match MPI libraries in the container**
  - Workaround: spawn job from a container?
  - MPI ABI standard/compatibility with PMI implementations?

- **Application binary needs to match CPU architecture**

- **Not exactly "create once, run everywhere" …**

RIKEN

Conclusions

- **Increasingly diverse workloads will benefit from the full specialization of the system software stack**
- **Containers in HPC are promising for software packaging**
  - Specialized user-space
- **Lightweight multi-kernels are beneficial for HPC workloads**
  - Specialized kernel-space
- **Combining the two brings both of the benefits**

Thank you for your attention!
Questions?