Diverse Workloads need Specialized System Software: An approach of Multi-kernels and Application Containers

Balazs Gerofi

bgerofi@riken.jp

Exa-scale System Software Research Group
RIKEN Advanced Institute for Computational Science, JAPAN

2017/Aug/28 -- ROME'17 Santiago De Compostela, Spain

RIKEN

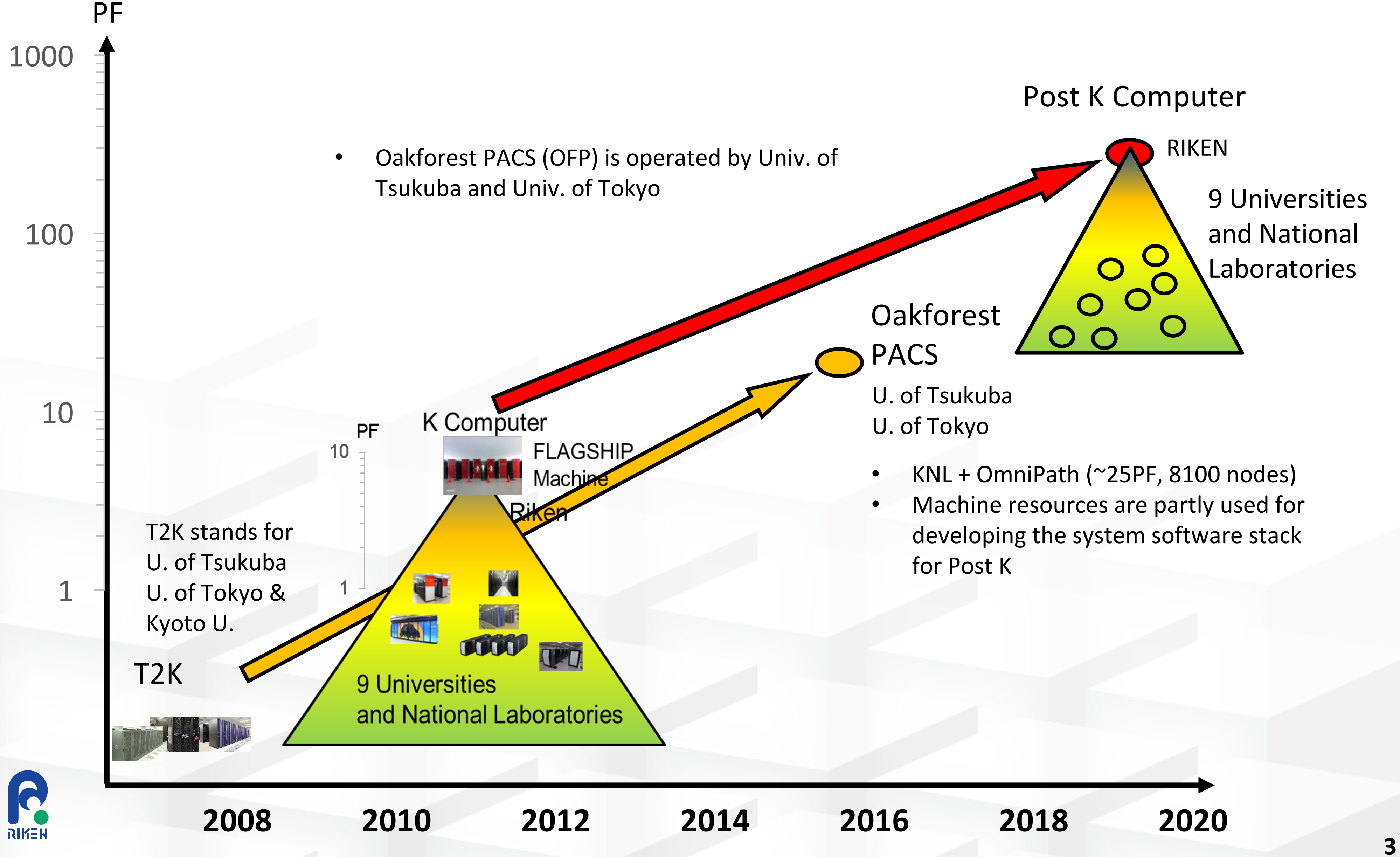
What is RIKEN?

- RIKEN is Japan's largest (government funded) research institution
- Established in 1917
- Research centers and institutes across Japan

K Computer

Advanced Institute for Computational Science (AICS)

Towards the Next Generation Flagship Japanese Supercomputer (without Tsubame series)



Agenda

- **Motivation**
- **Lightweight Multi-kernels**
  - IHK/McKernel
- **Linux container concepts**
- **conexec: integration with multi-kernels**
- **Results**
- **Conclusion**

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

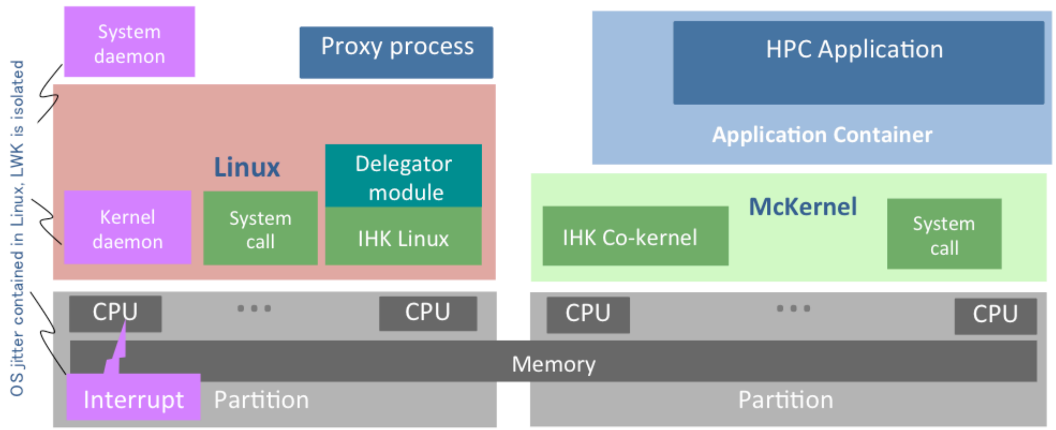
- **Node architecture: increasing complexity and heterogeneity**
  - Large number of (heterogeneous) CPU cores, deep memory hierarchy, complex cache/NUMA topology, specialized PUs
- **Applications: increasing diversity**
  - Traditional/regular HPC + in-situ data analytics + Big Data processing + Machine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

- **What do we need from the system software/OS (HPC perspective)?**
  - 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, specialized PUs
  - Performance isolation and dynamic reconfiguration
    - According to workload characteristics, support for co-location

RIKEN

Approach: embrace diversity and complexity

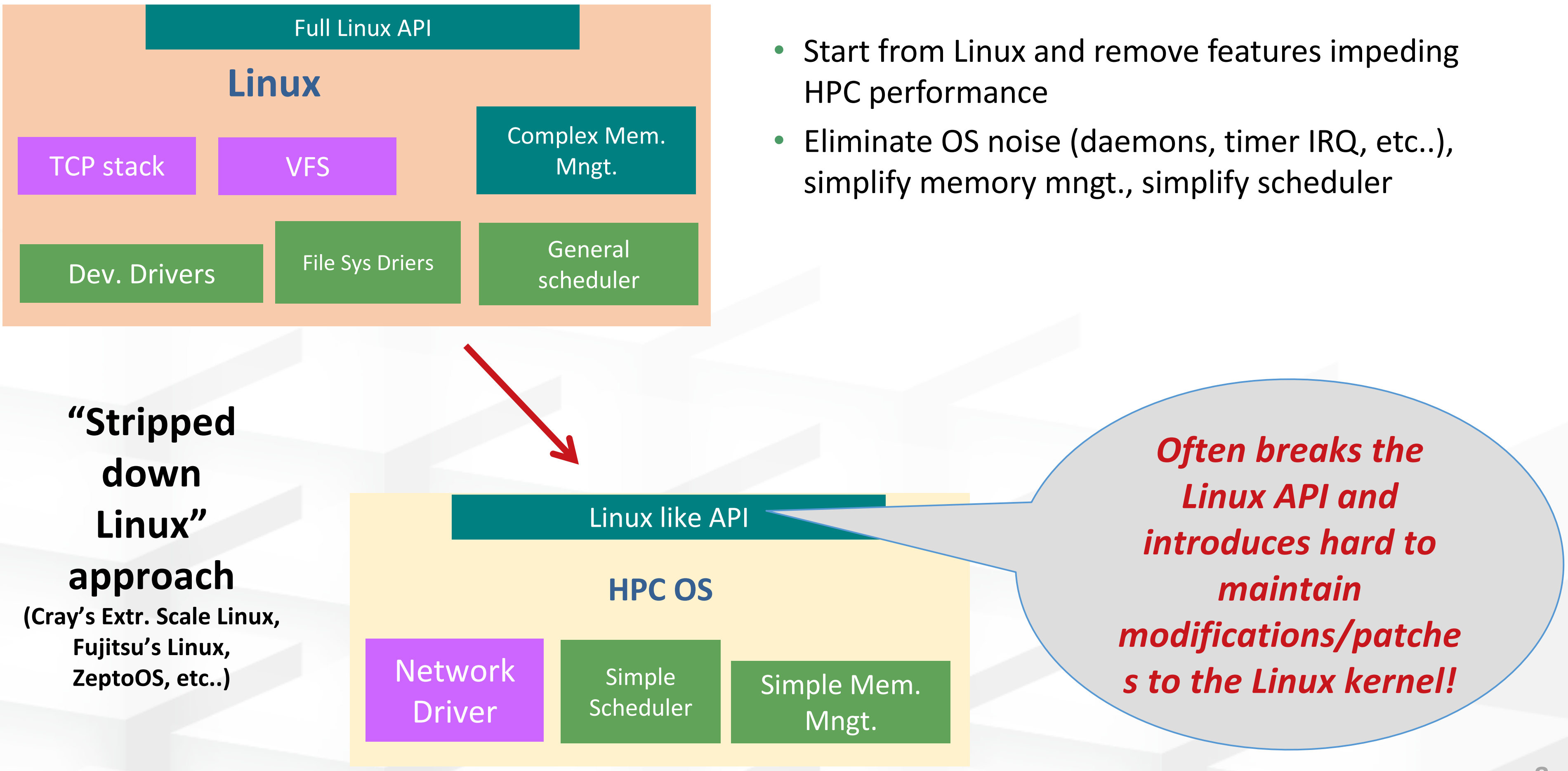
- **Enable *dynamic specialization of the system software stack* to meet application requirements**
  - User-space: Full provision of libraries/dependencies for all applications will likely not be feasible:
    - Containers (i.e., namespaces) – specialized user-space stack
  - Kernel-space: Single monolithic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



Lightweight Multi-Kernels

Background – HPC Node OS Architecture

- **Traditionally: driven by the need for scalable, consistent performance for bulk-synchronous HPC**



- Start from Linux and remove features impeding HPC performance
- Eliminate OS noise (daemons, timer IRQ, etc..), simplify memory mngt., simplify scheduler

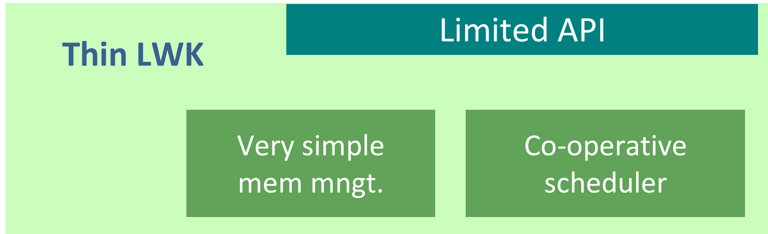
"Stripped down Linux" approach

(Cray's Extr. Scale Linux, Fujitsu's Linux, ZeptoOS, etc..)

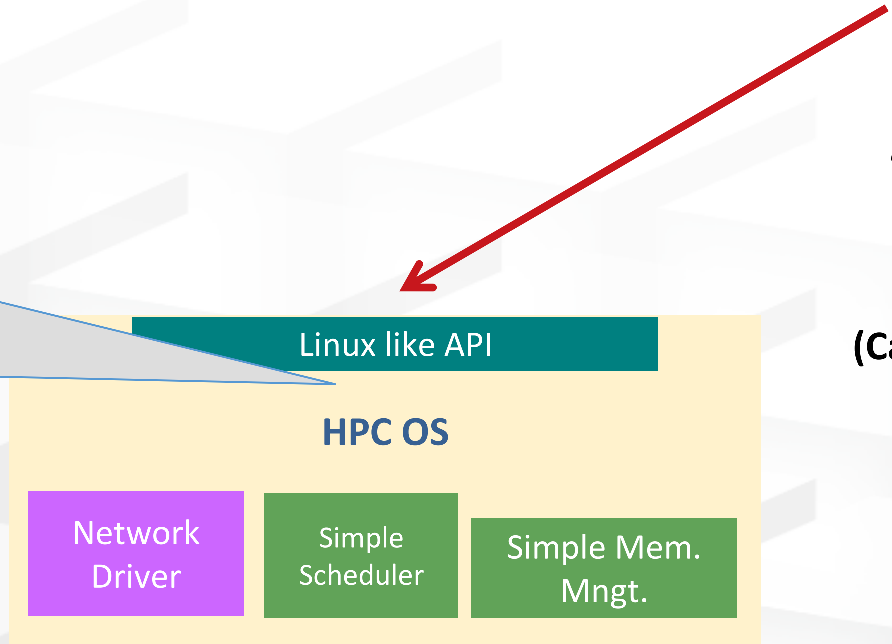
Background – HPC Node OS Architecture

- **Traditionally: driven by the need for scalable, consistent performance for bulk-synchronous HPC**

  - Start from a thin Lightweight Kernel (LWK) written from scratch and add features to provide a more Linux like I/F, but keep scalability
  - Support dynamic libraries, allow thread over-subscription, support for `/proc` filesystem, etc..



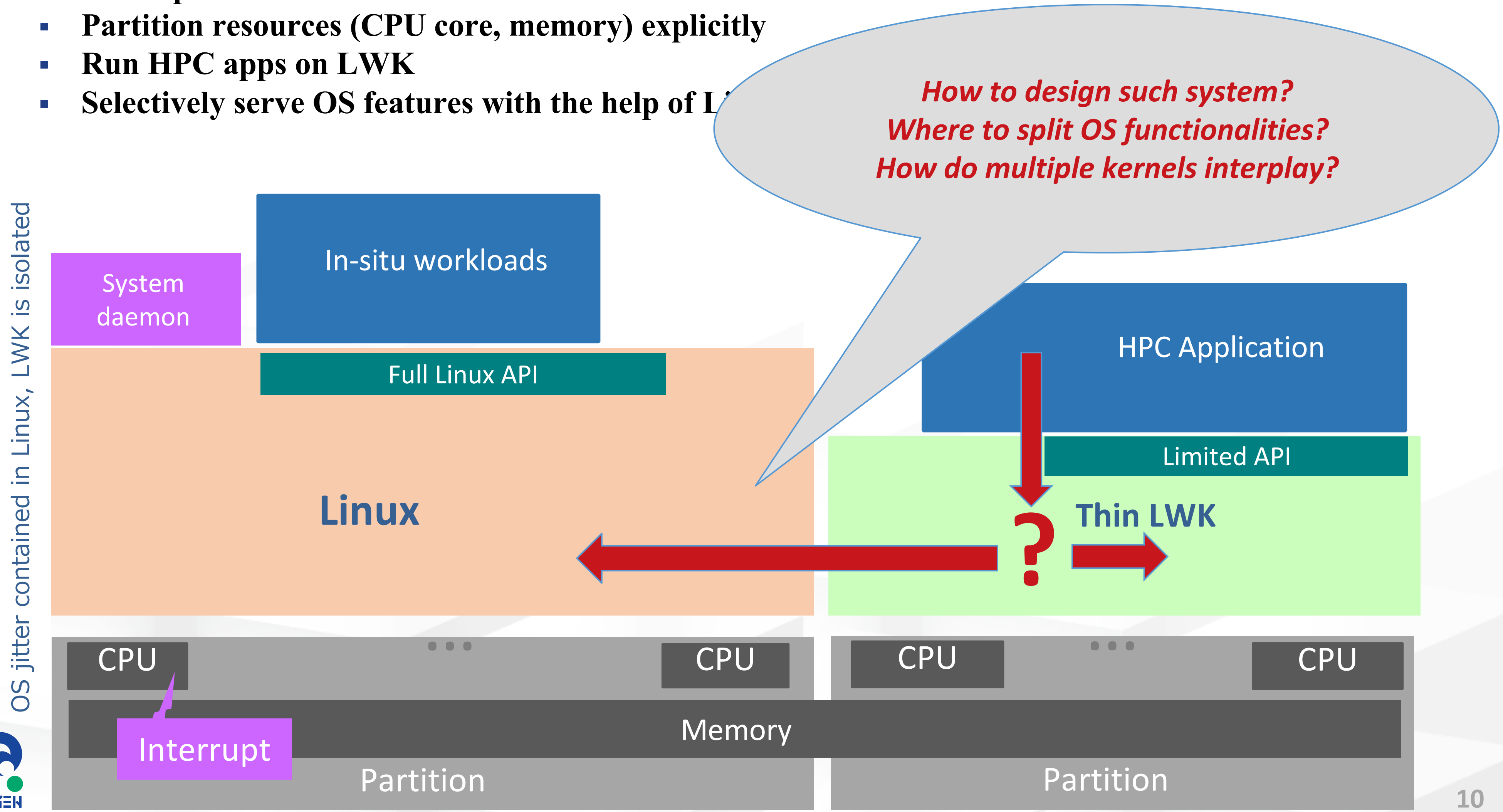
No full Linux API, lack of device drivers and support for tools!



“Enhanced LWK” approach
(Catamount, CNK, Kitten, etc..)

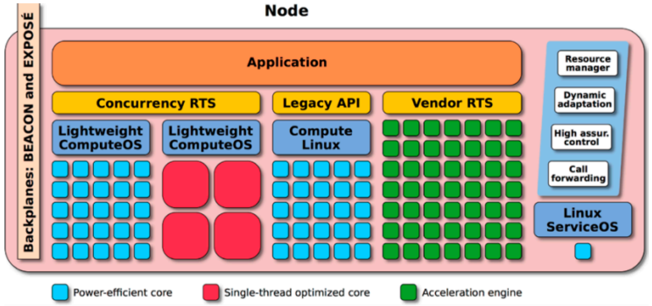
High Level Approach: Linux + Lightweight Kernel

- With the abundance of CPU cores comes the hybrid approach: run Linux and LWK side-by-side in compute nodes!
- Partition resources (CPU core, memory) explicitly
- Run HPC apps on LWK
- Selectively serve OS features with the help of L

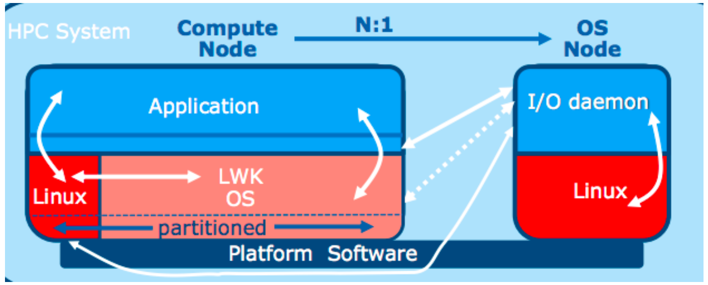


Hybrid/Specialized (co)-Kernels

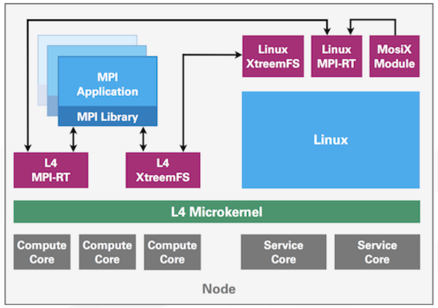
The idea of combining FWK+LWK was first proposed by FusedOS @ IBM!



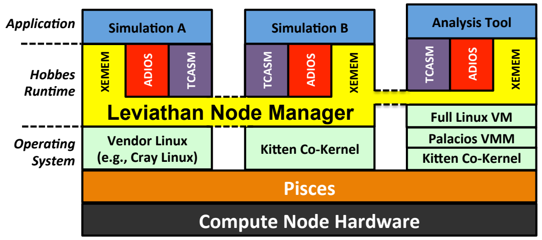
Argo (nodeOS), led by Argonne National Laboratory



mOS @ Intel Corporation



FFMK, led by TU Dresden



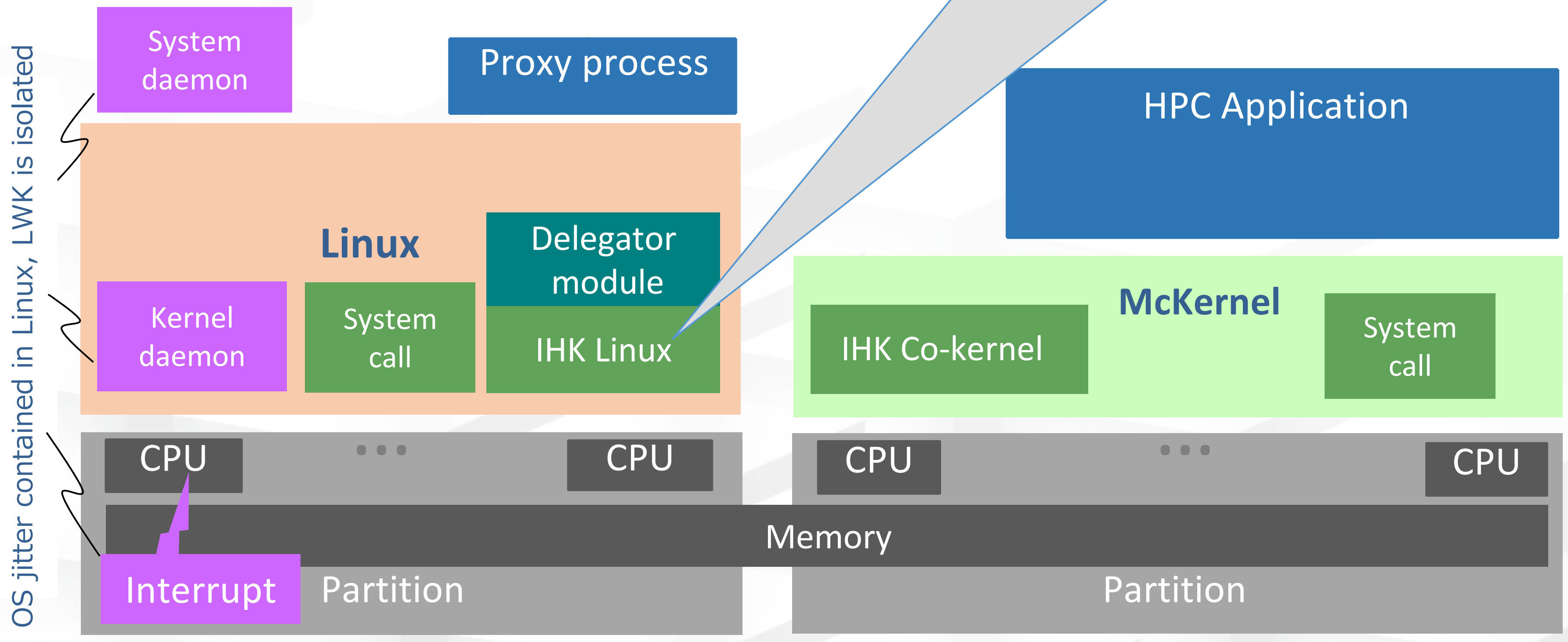
Hobbes, led by Sandia National Laboratories

Property/Project	Unmodified Linux Kernel	Device Driver Transparency in LWK	Kernel Level Workload Isolation	Full POSIX Support on LWK	Development Effort
Argo	No	Yes	No	Yes	Ideally small
mOS	No	Yes	Yes/No?	Yes	Ideally small
Hobbes (a.k.a., Pisces+Kitten)	Yes	No	Yes	No	Significant
FFMK (L4+Linux)	No	No	Yes	No	Significant
IHK/McKernel	Yes	Yes	Yes	Yes	Significant

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 notificati
- **McKernel:**
  - A lightweight kernel developed from scratch, boo
  - Designed for HPC, noiseless, simple, implements only p and memory management) and the rest are offloaded to Linux

No Linux modifications!
Dynamic reconfiguration.
No reboot of the host Linux required!



McKernel and System Calls

- McKernel is a lightweight (co-)kernel designed for HPC
- Linux ABI compatible
- Boots from IHK (no intention to boot it stand-alone)
- Noiseless, simple, with a minimal set of features implemented and the rest offloaded to Linux

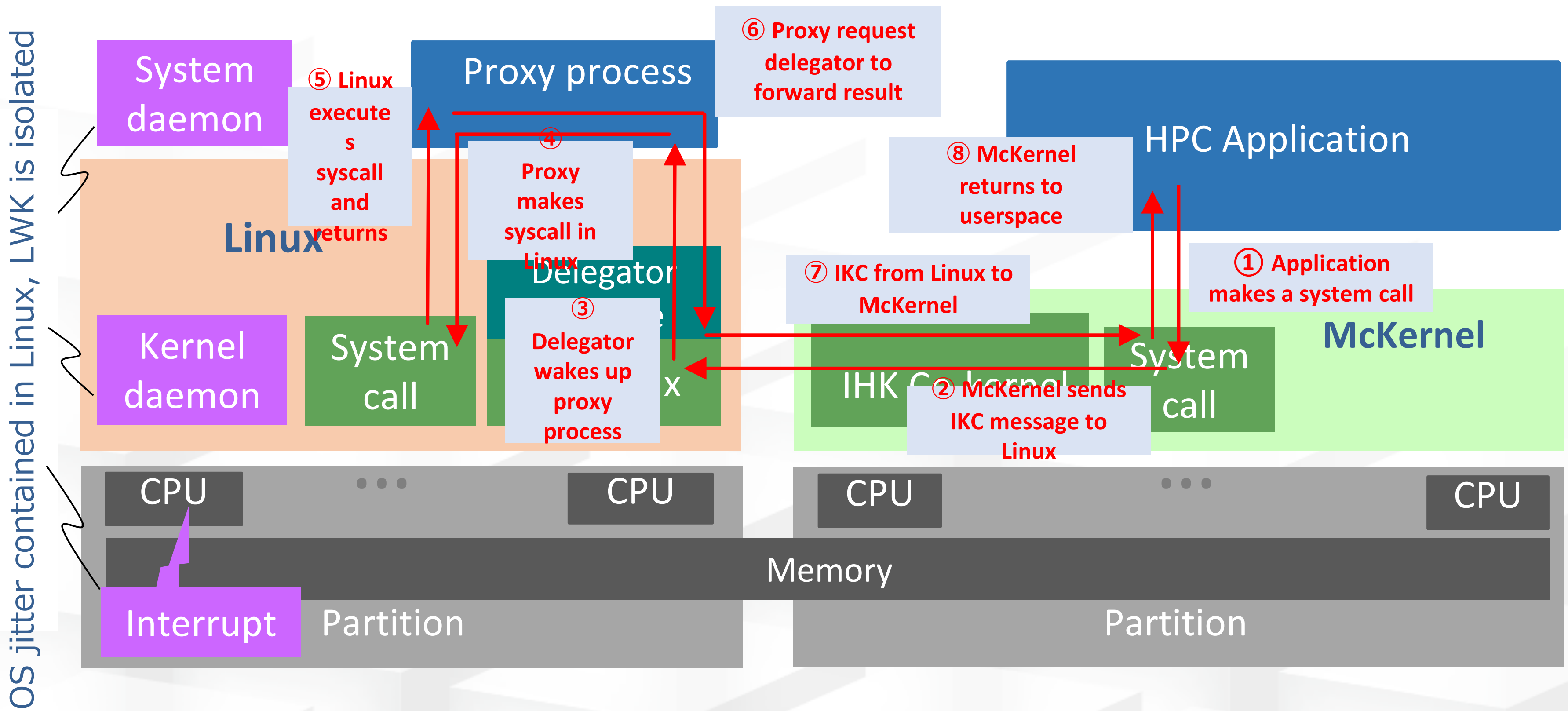
	Implemented	Planned
Process Thread	arch_prctl, clone, execve, exit, exit_group, fork, futex, getpid, getrlimit, kill, pause, ptrace, rt_sigaction, rt_sigpending, rt_sigprocmask, rt_sigqueueinfo, rt_sigreturn, rt_sigsuspend, set_tid_address, setpgid, sigaltstack, tgkill, vfork, wait4, signalfd, signalfd4, ptrace	get_thread_area, getrlimit, rt_sigtimedwait, set_thread_area, setrlimit
Memory management	brk, gettid, madvise, mlock, mmap, mprotect, mremap, munlock, munmap, remap_file_pages, shmat, shmctl, shmdt, shmget, mbind, set_mempolicy, get_mempolicy	get_robust_list, mincore, mlockall, modify_ldt, munlockall, set_robust_list
Scheduling	sched_getaffinity, sched_setaffinity, getitimer, gettimeofday, nanosleep, sched_yield, settimeofday	setitimer, time, times
Performance Counter	Direct PMC interface: pmc_init, pmc_start, pmc_stop, pmc_reset, PAPI Interface	

- System calls not listed above are *offloaded* to Linux
- POSIX compliance: *almost the entire LTP test suite passes! (2013 version: 100%, 2015: 99%)*

RIKEN

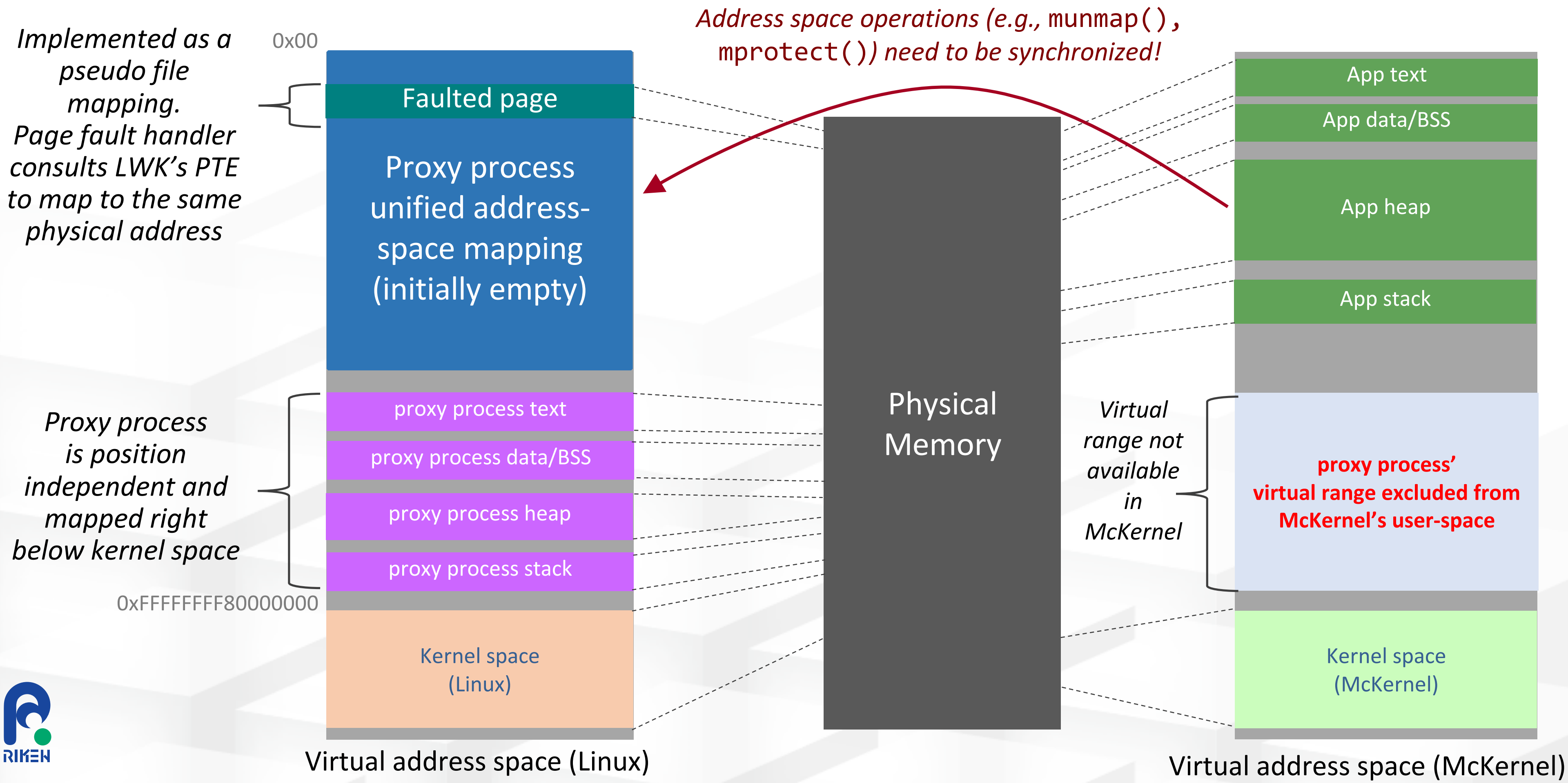
Proxy Process and System Call Offloading in IHK/McKernel

- **For each application process a "proxy-process" resides on Linux**
- **Proxy process:**
  - Provides execution context on behalf of the application so that offloaded calls can be directly invoked in Linux
  - Enables Linux to maintain certain state information that would have to be otherwise kept track of in the LWK
    - (e.g., file descriptor table is maintained by Linux)



Unified Address Space on x86

- **Issue: how to handle memory addresses in system call arguments?**
  - Consider the target buffer of a `read()` system call
- **There is a need for the proxy process to access the application's memory (running on McKernel)**
- **Unified address space ensures proxy process can transparently see applications memory contents and reflect virtual memory operations (e.g., `mmap()`, `munmap()`, etc..)**



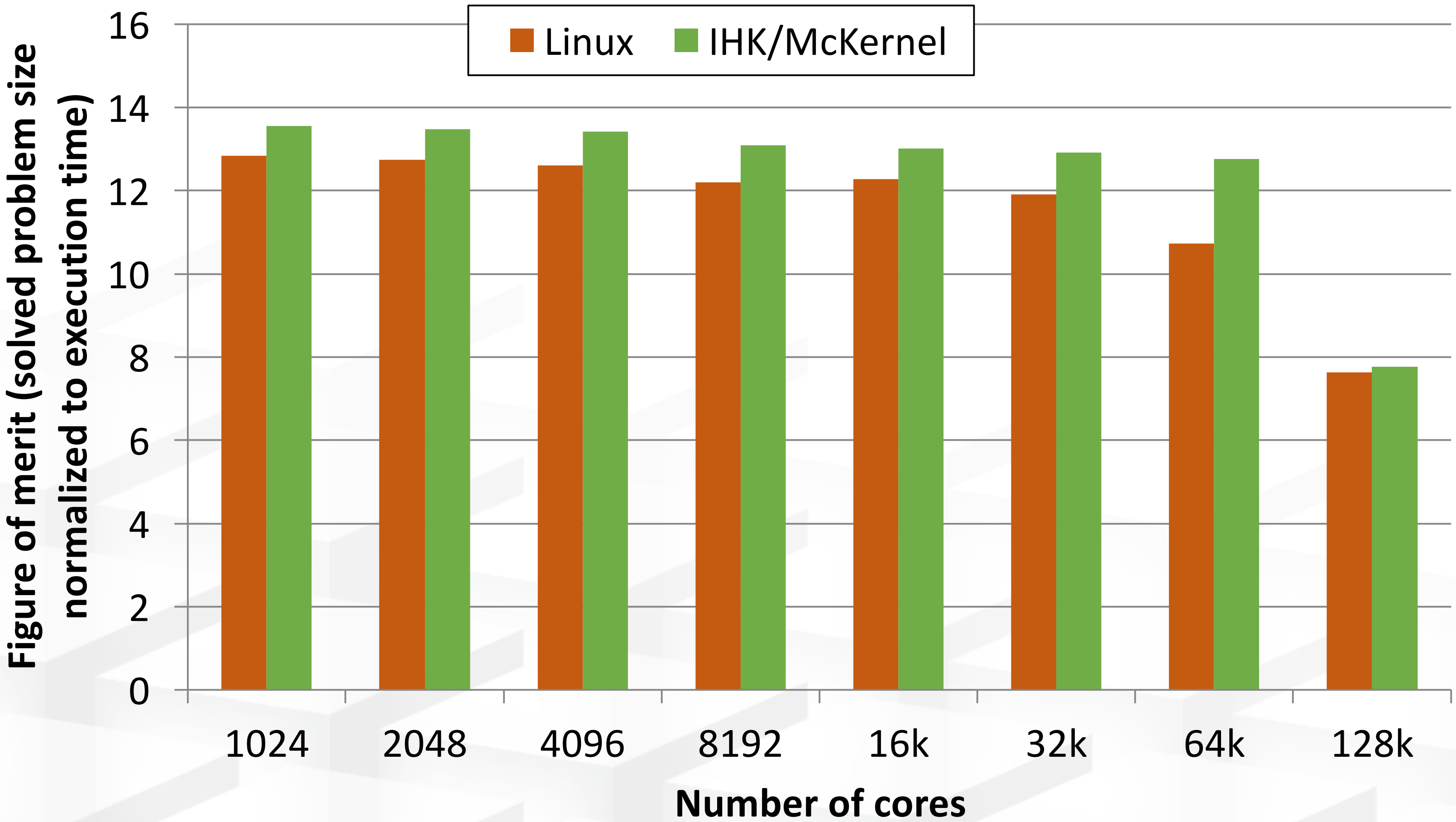
Preliminary Evaluation

- **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

- **Acknowledgements for machine access:**
  - Taisuke Boku @ The University of Tsukuba
  - Kengo Nakajima @ The University of Tokyo

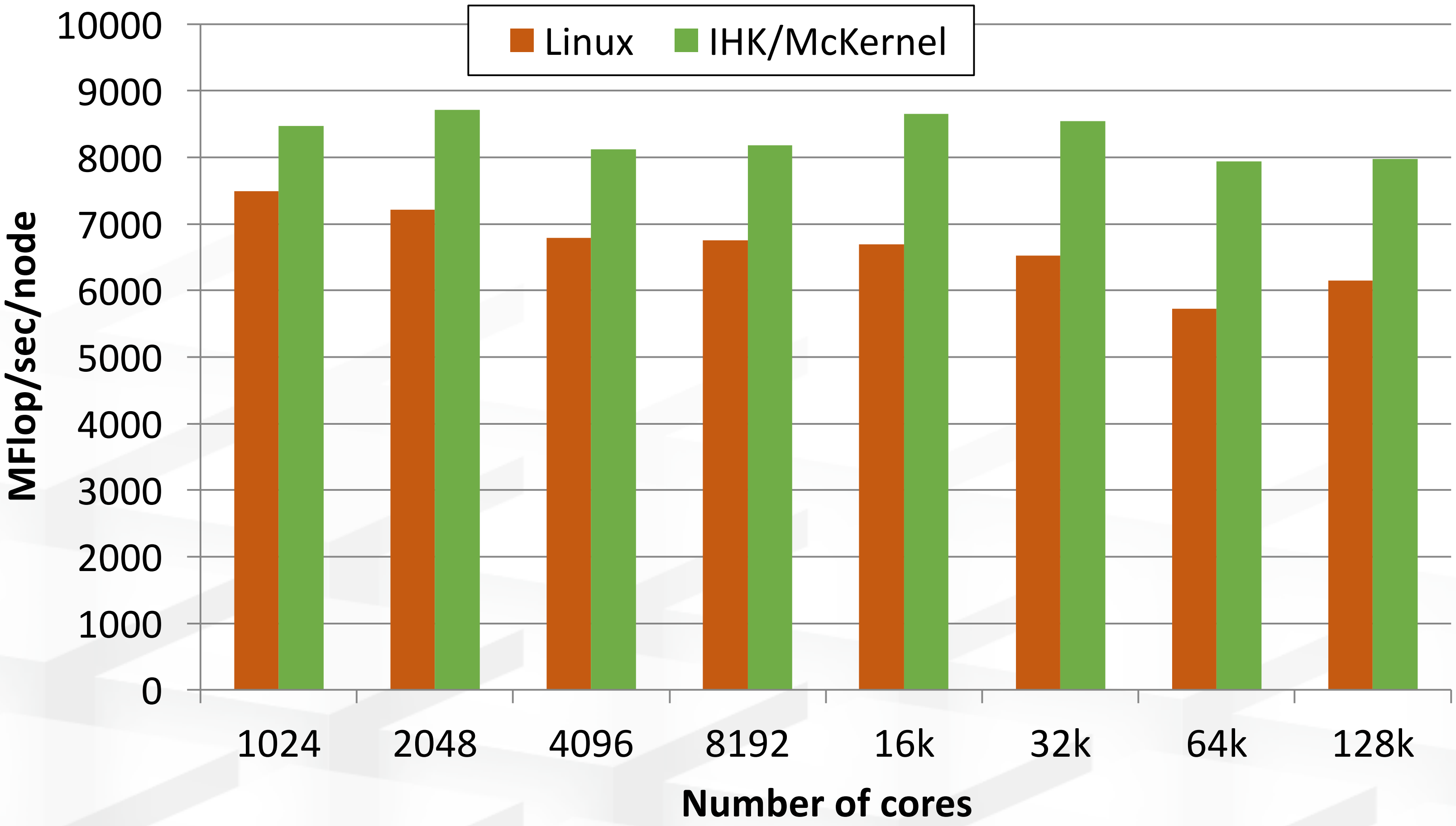
GeoFEM (University of Tokyo)

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



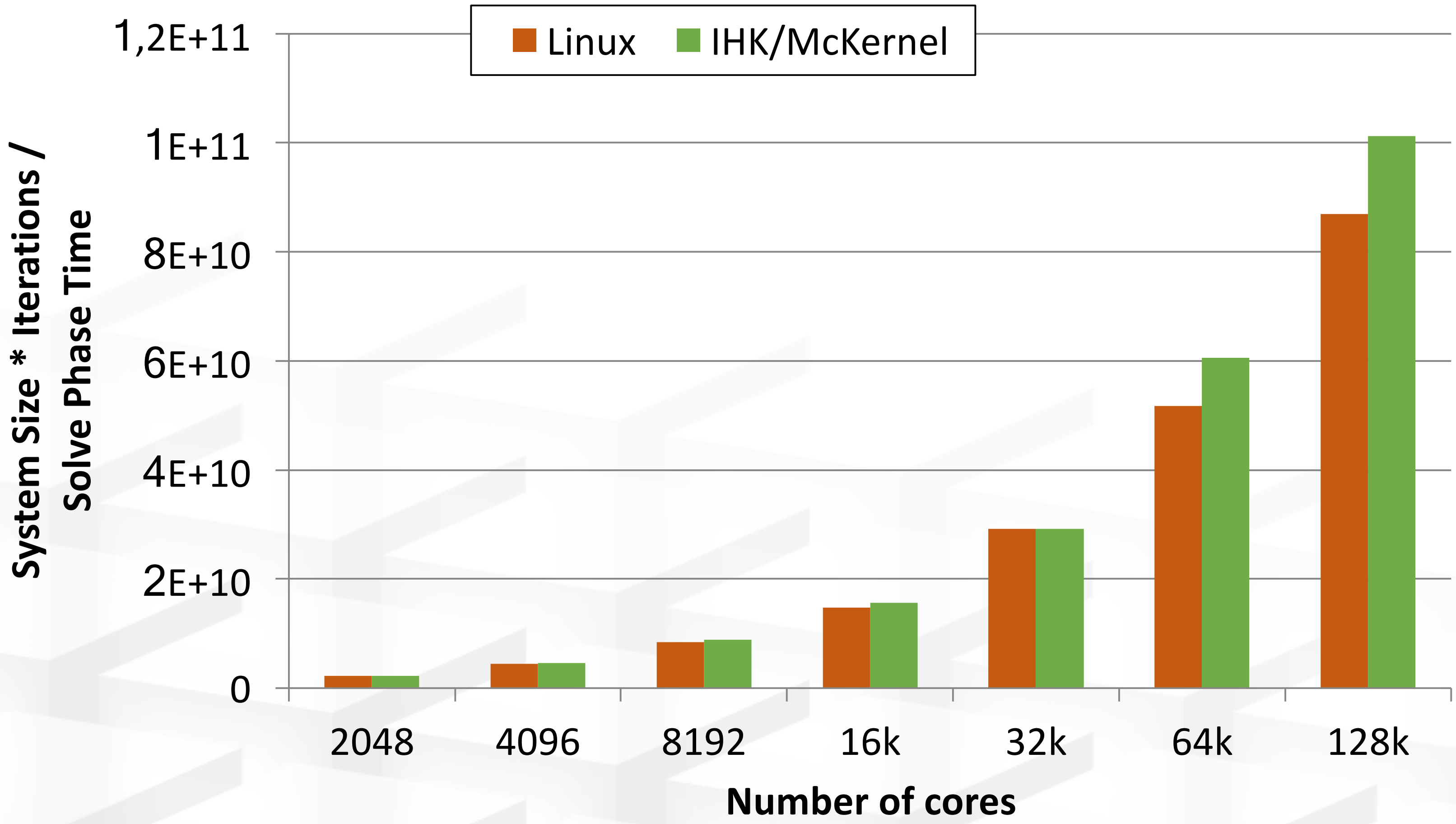
CCS-QCD (Hiroshima University)

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



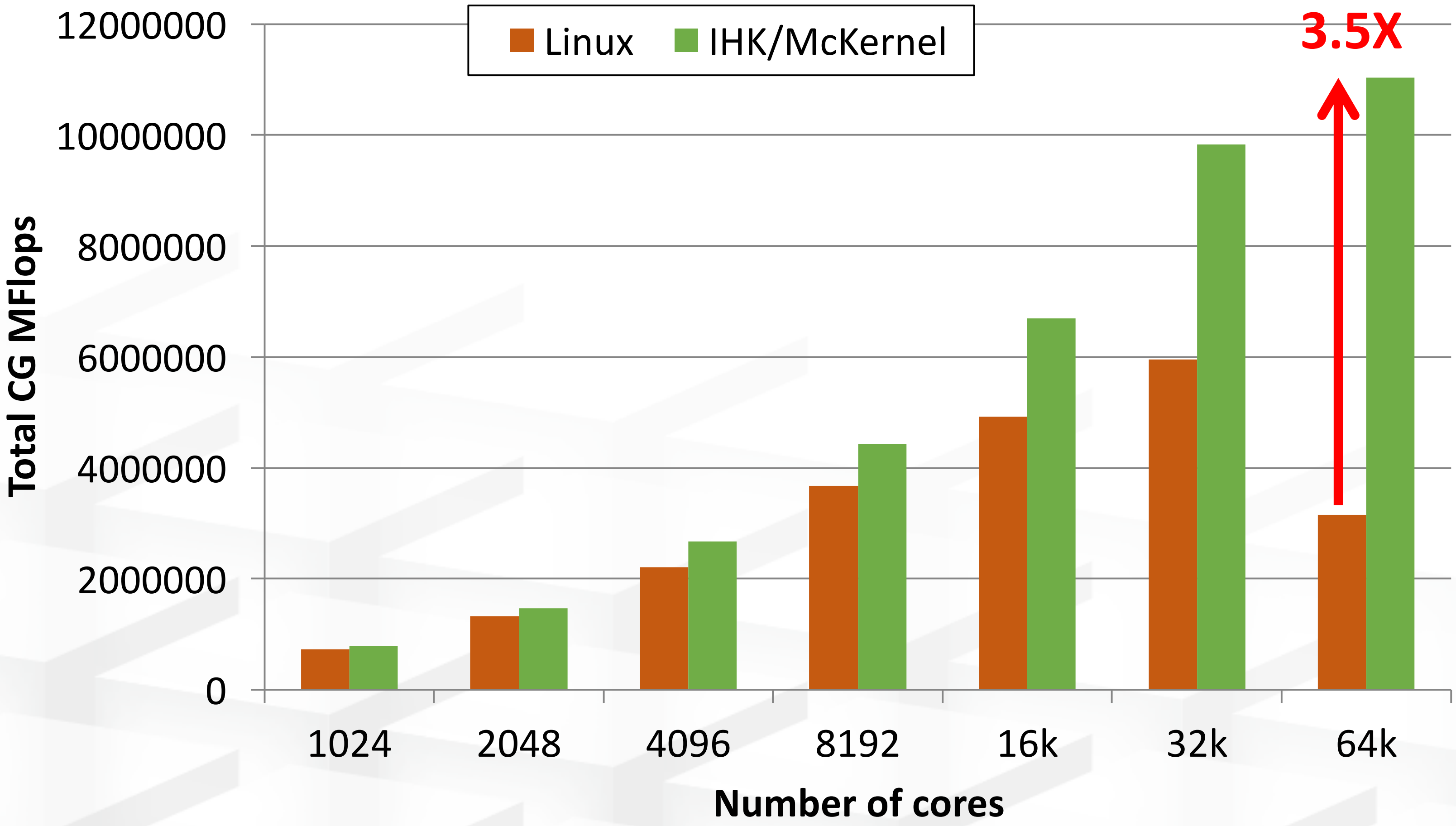
AMG2013 (CORAL benchmark suite)

- Parallel algebraic multigrid solver – weak scaling
- Up to 12% improvement and growing ☺



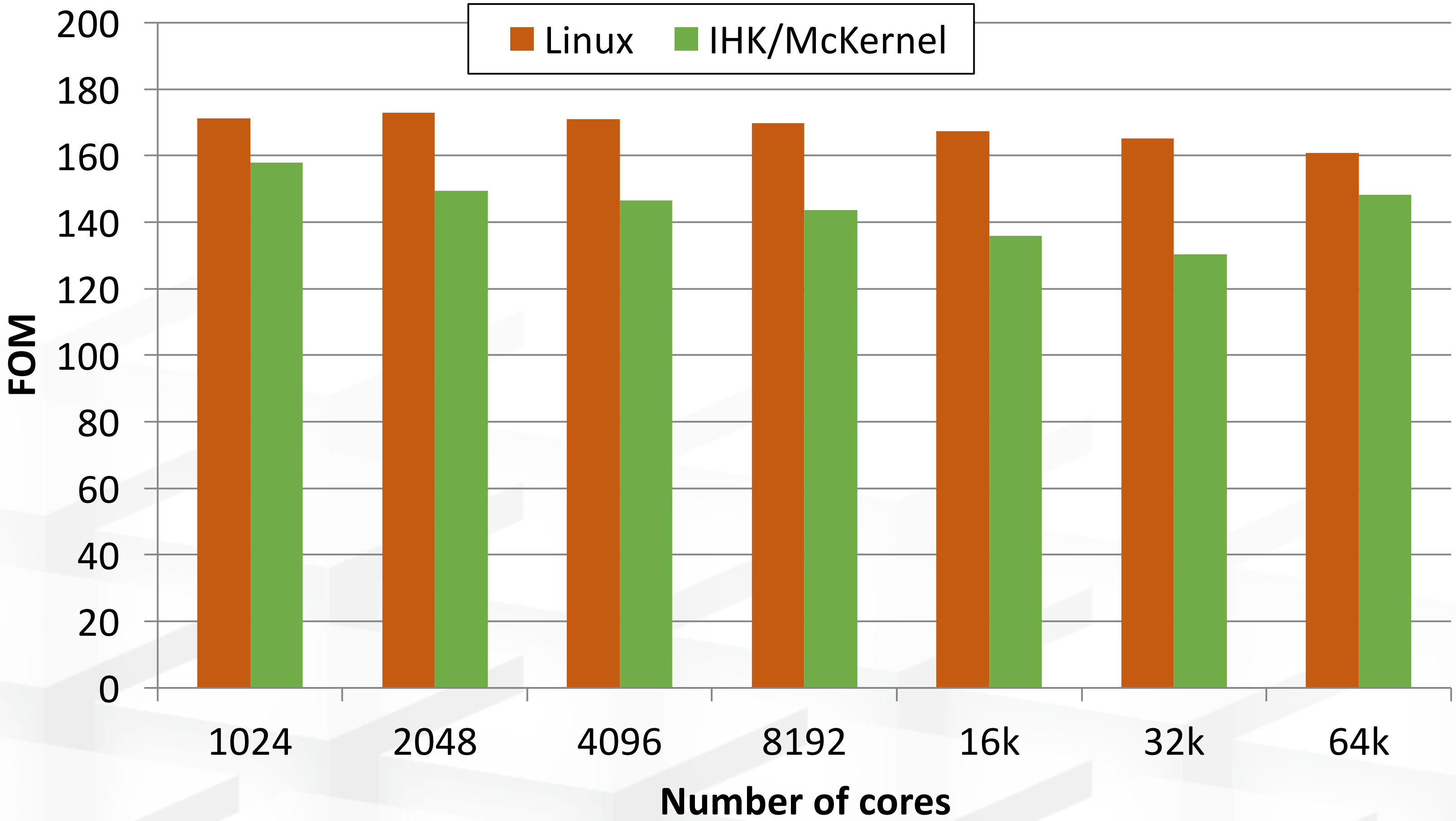
miniFE (CORAL benchmark suite)

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



lammps (CORAL benchmark suite)

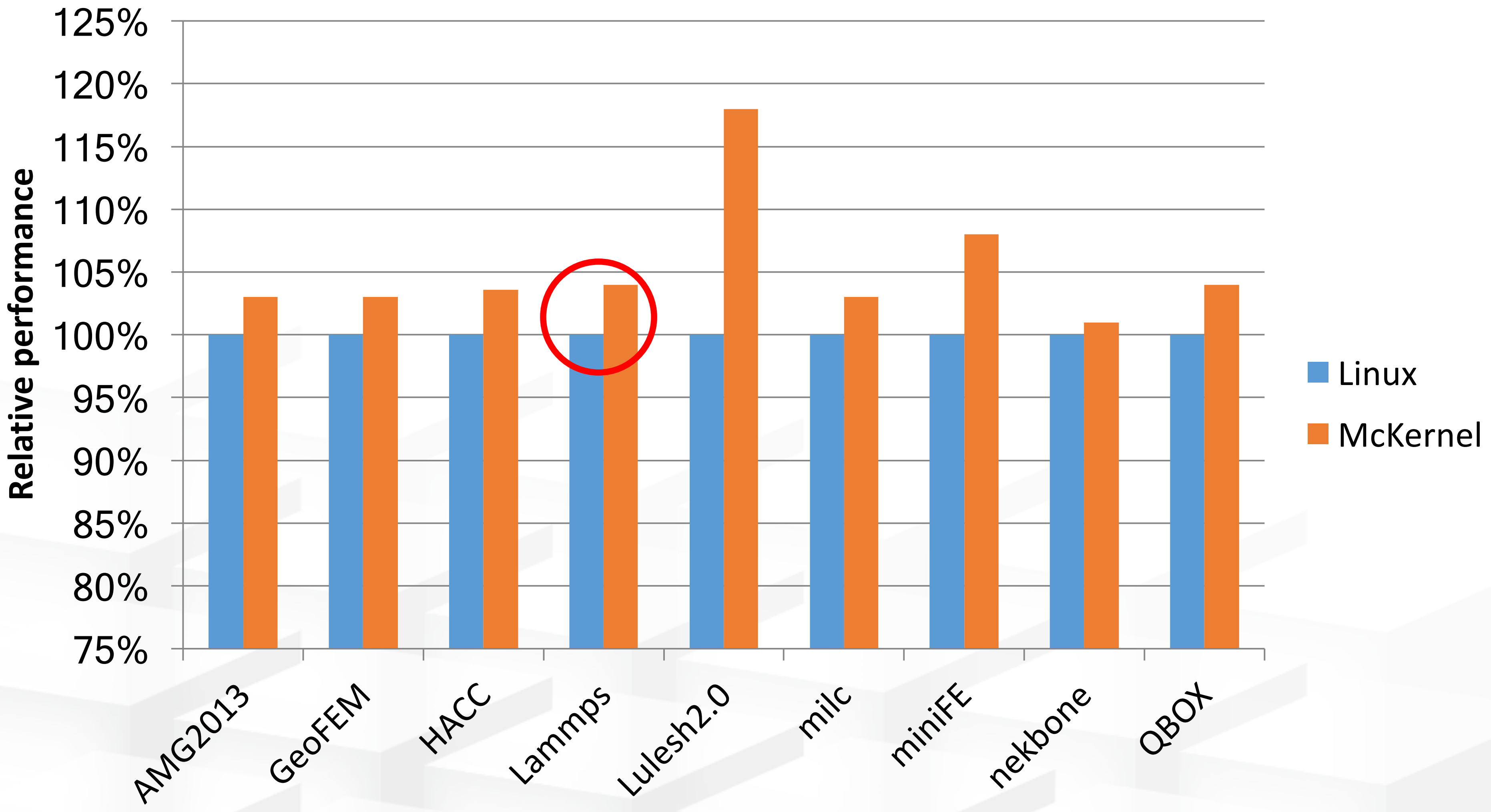
- Not all benchmarks benefit
- Up to 24% slowdown ☹



- Heavy use of `writev()` syscalls of OmniPath network driver which get offloaded to Linux
- According to Intel, next generation OP will fix this problem

Single node: McKernel outperforms Linux across the board

→ **multi-node Lammps suffers from network offloading..**

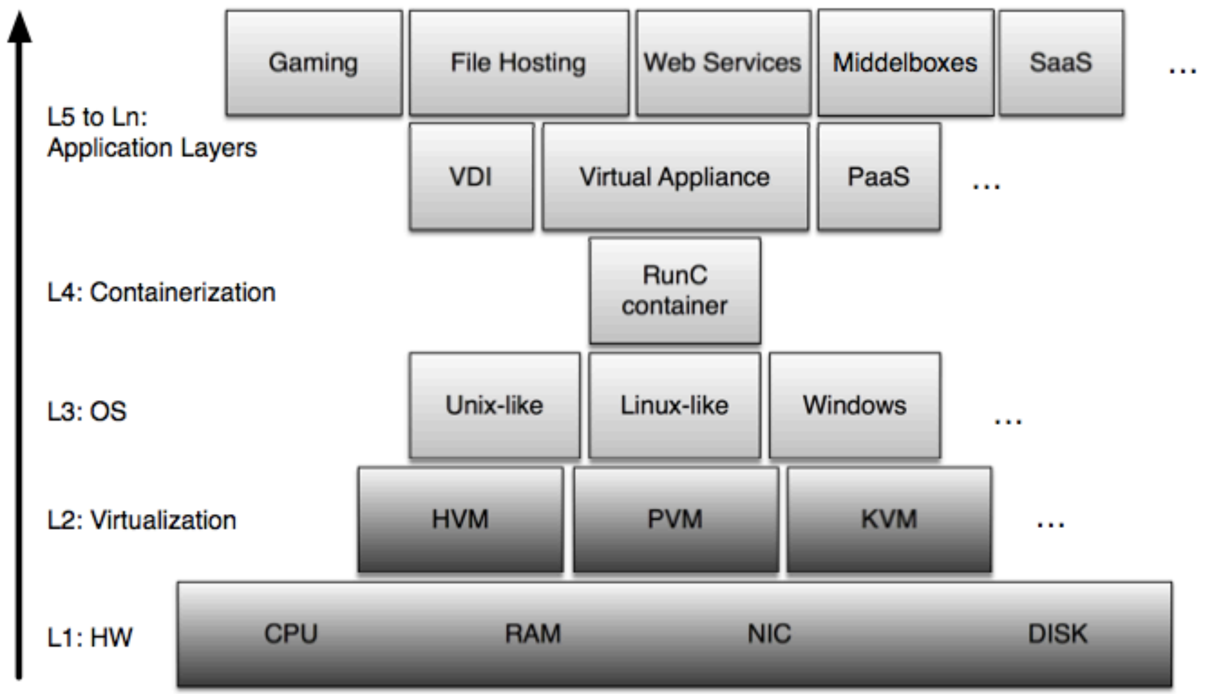


- lammps, HACC, QBOX ~4% better, as opposed to being slower than Linux on 8 nodes
- OmniPath offload overhead??

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]
```

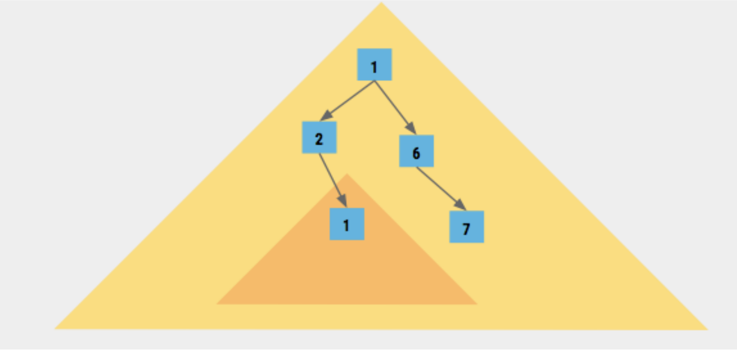
Mount Namespace

- **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

RIKEN

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

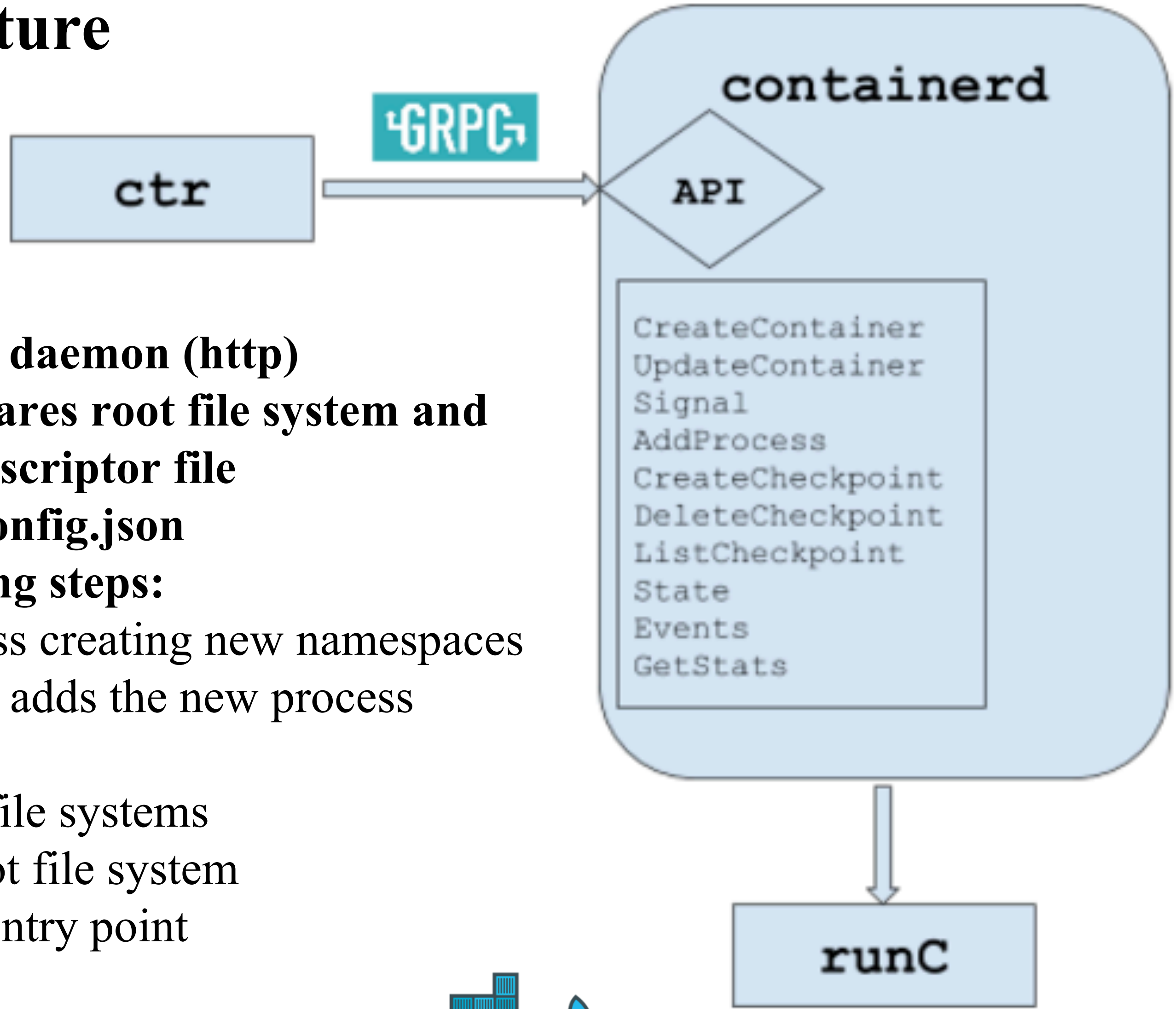
RIKEN

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

RIKEN

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

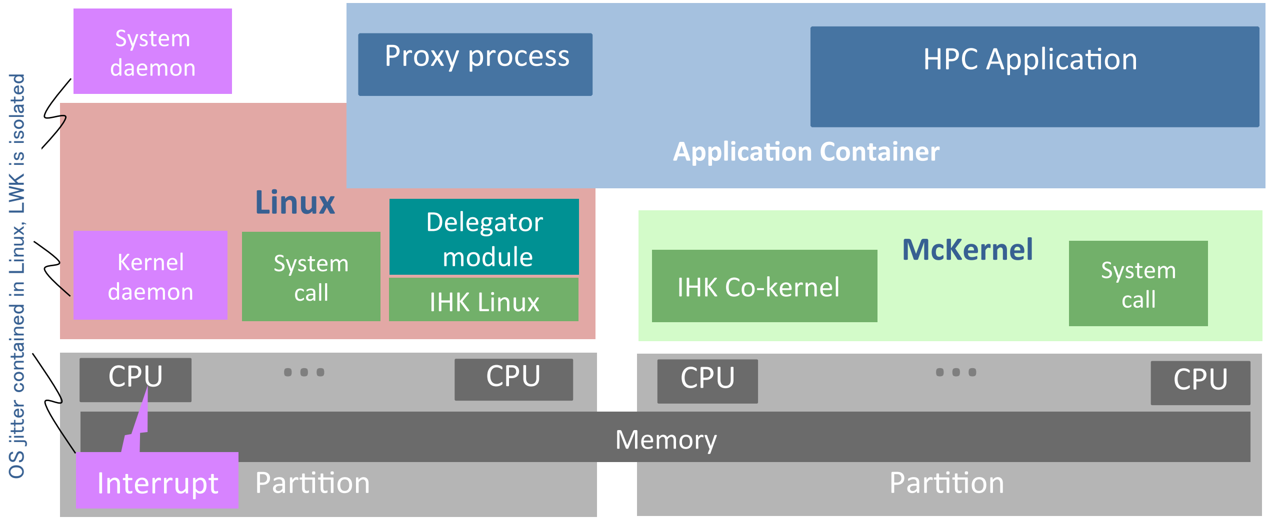
Comparison of container technologies

Project/ Attribute	Docker	rkt	Singularity	Shifter
Supports/uses namespaces	yes	yes	mainly mount (others 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/no? (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

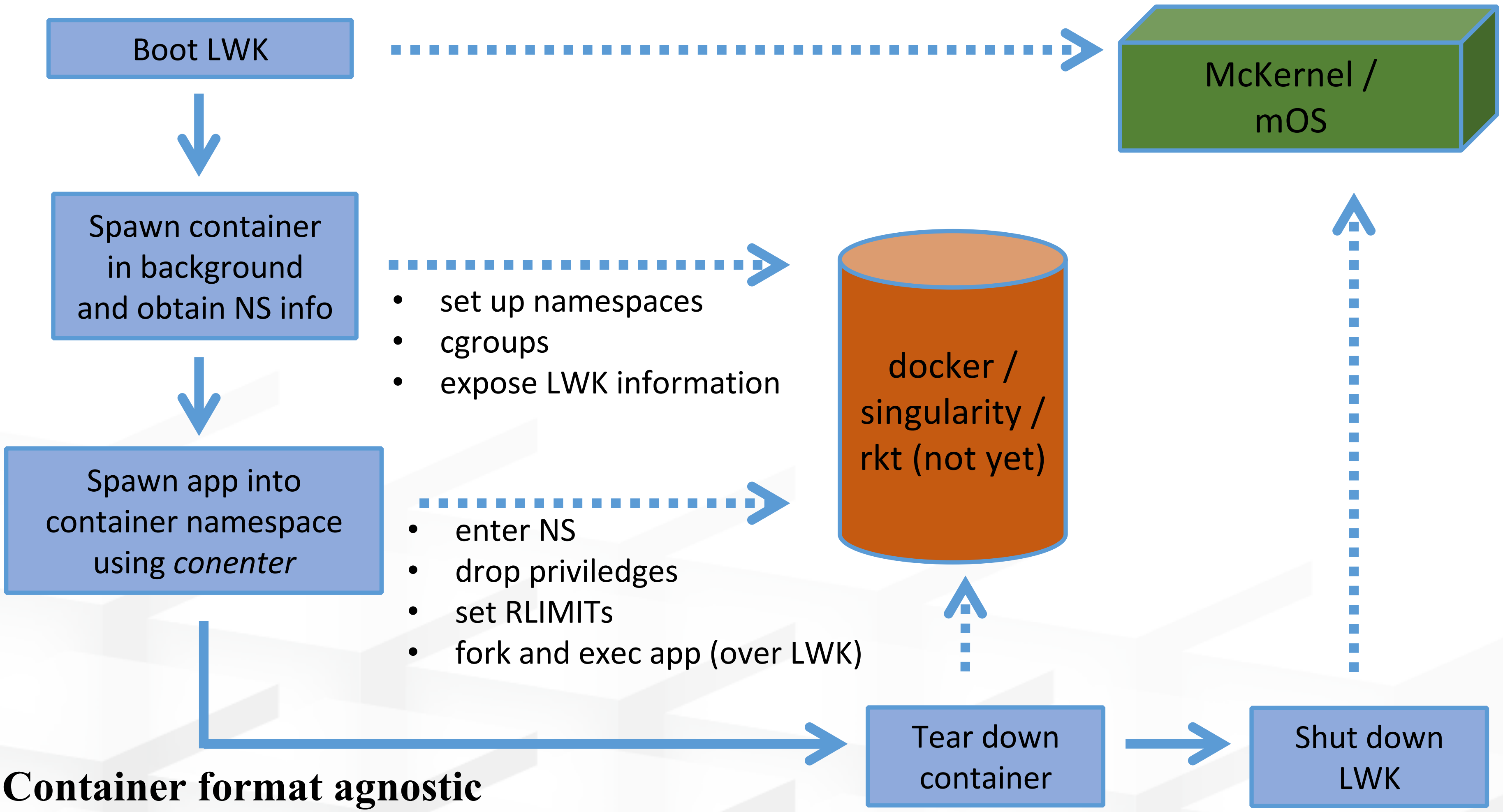
Integration of containers and lightweight multi-kernels

IHK/McKernel 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

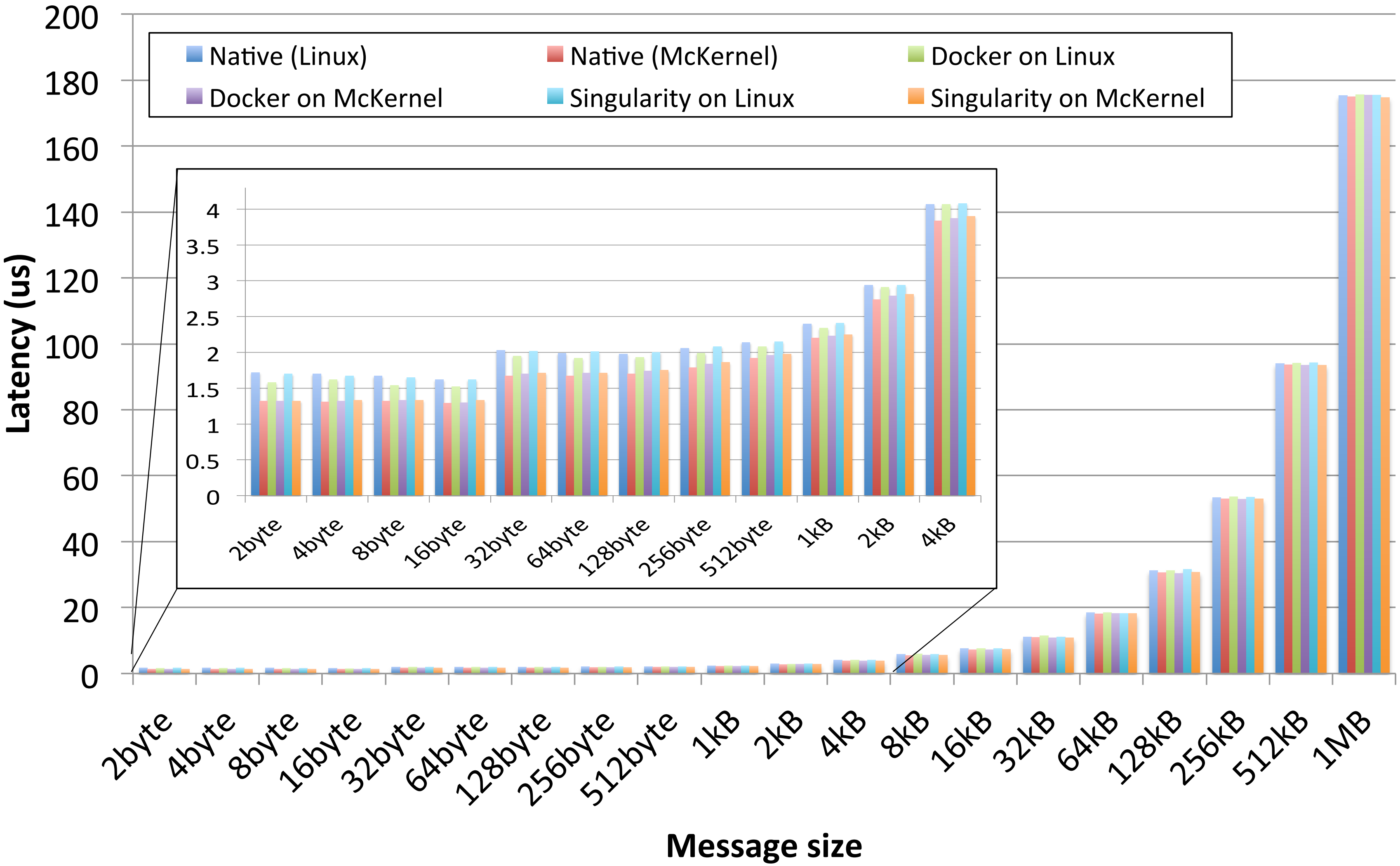
- **Running with MPI:**
  - mpirun -genv I_MPI_FABRICS=dapl -f hostfile -n 16 -ppn 1 /home/bgerofi/Code/conexec/conexec --lwk mckernel --lwk-cores 10-19 --lwk-mem 2G@0 singularity:///home/bgerofi/containers/singularity2/miniapps.img /opt/IMB_4.1/IMB-MPI1 Allreduce

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

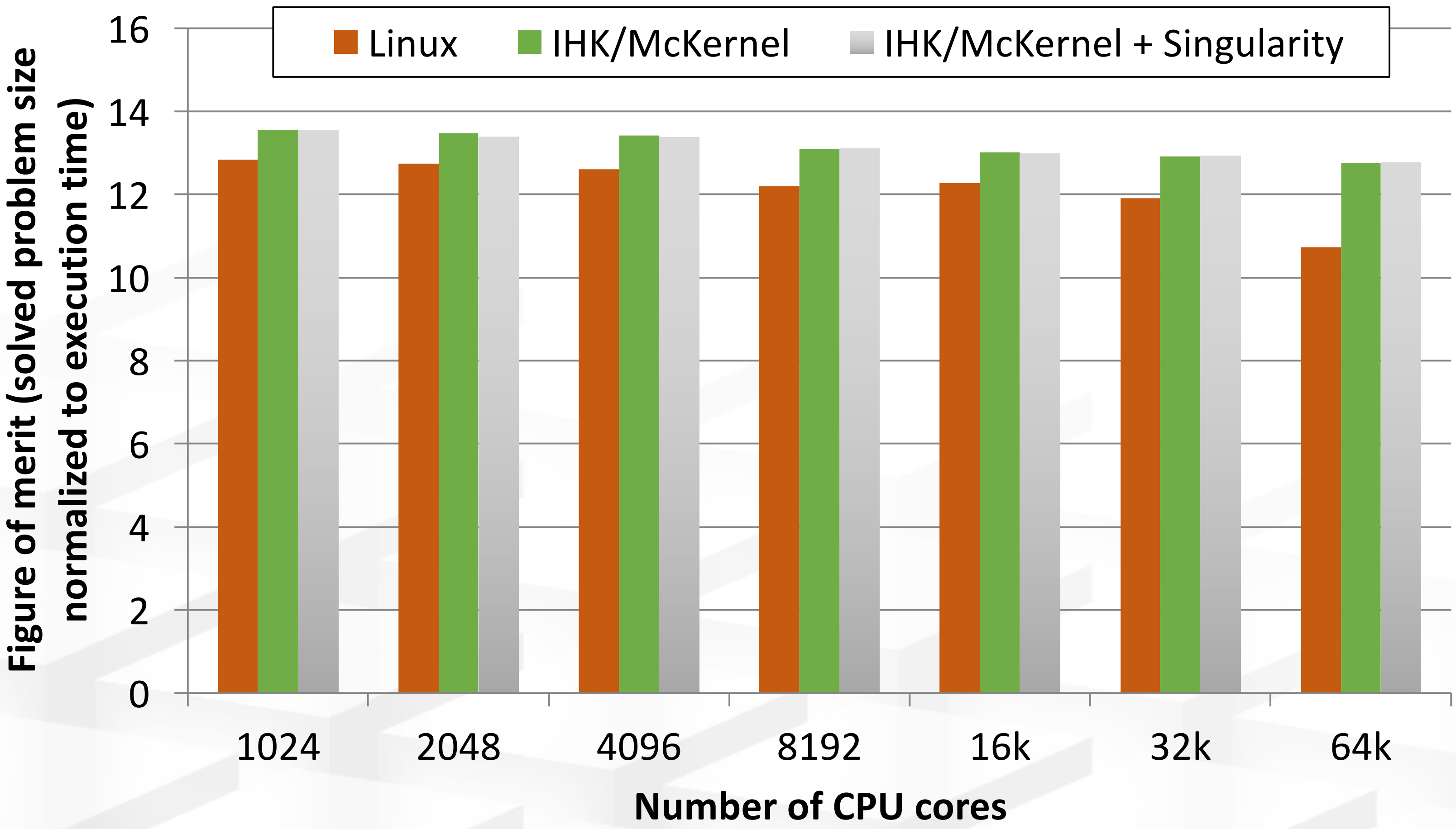
IMB PingPong – Containers impose ~zero overhead



- 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!

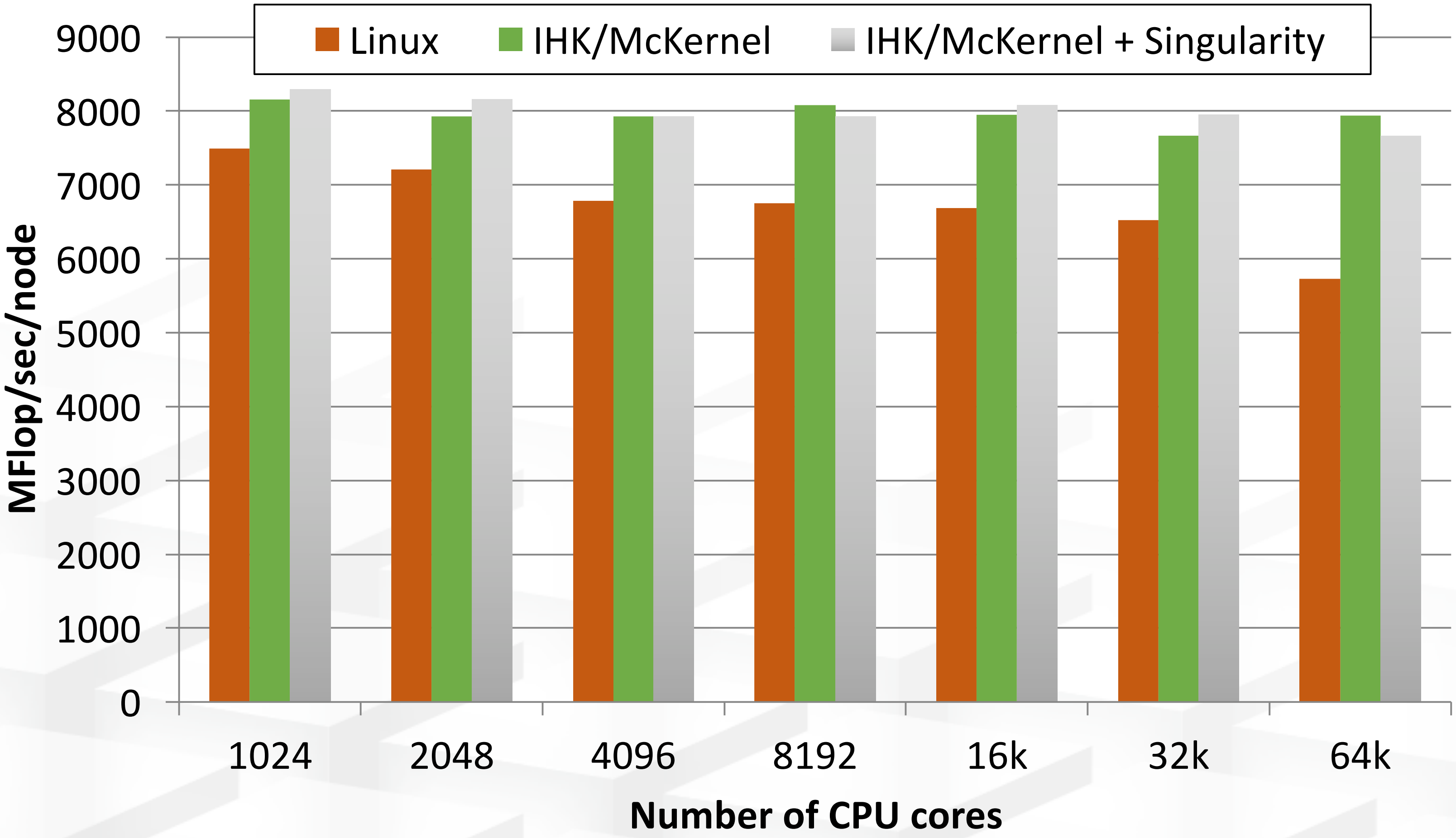
GeoFEM (University of Tokyo) in container

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



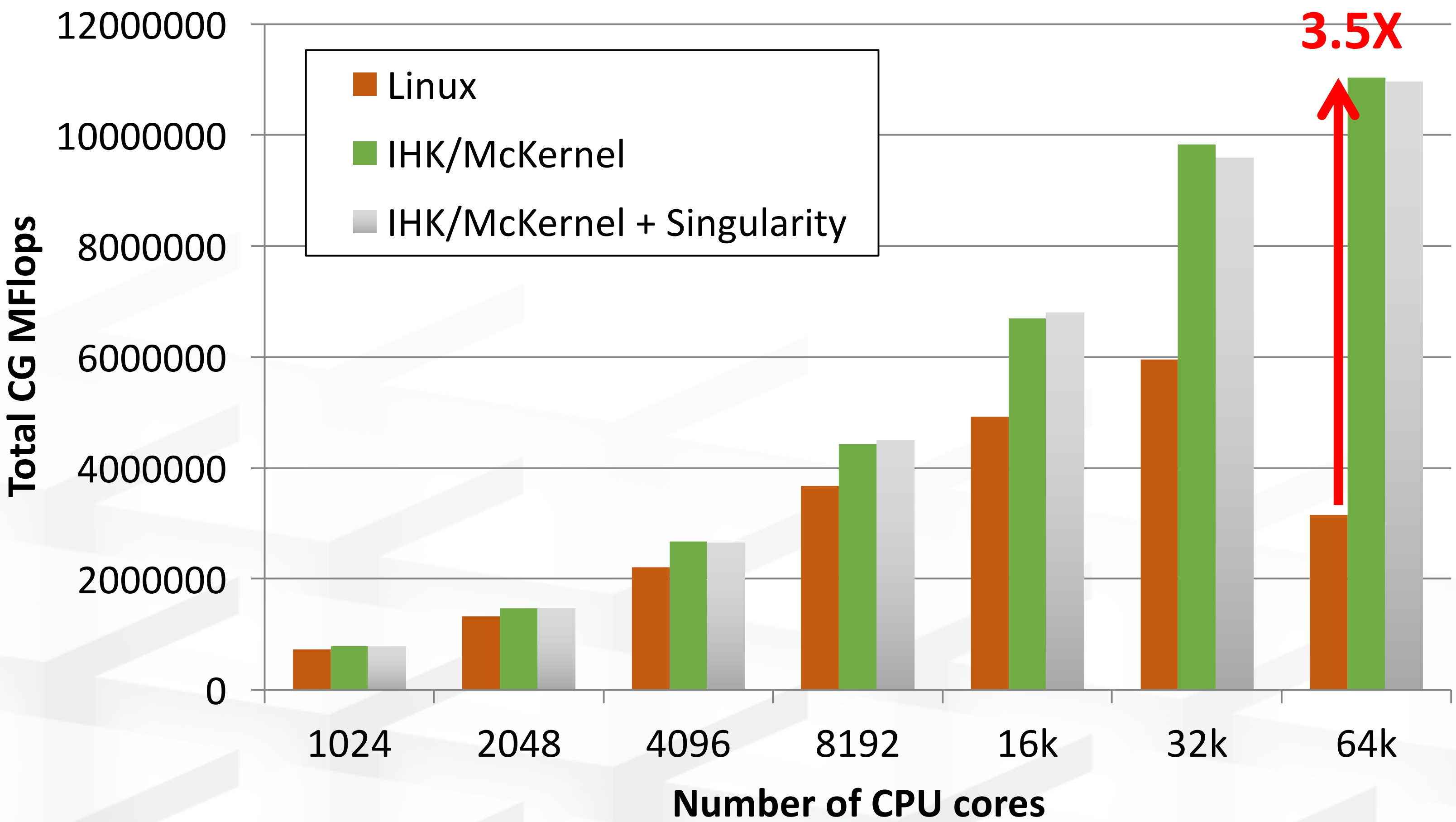
CCS-QCD (Hiroshima University) in container

- **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..)**



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**
- **Vision: a CoreOS like minimalistic Linux with workload specific multi-kernels running containers**

RIKEN

Thank you for your attention!
Questions?