A Container-Based Approach to OS Specialization for Exascale Computing

Judicael A. Zounmevo, Swann Perarnau, Kamil Iskra, Kazutomo Yoshii, Roberto Gioiosa, Brian C. Van Essen, Maya B. Gokhale, and Edgar A. Leon

Argonne National Laboratory
Pacific Northwest National Laboratory
Lawrence Livermore National Laboratory

Outline

- Towards exascale computing
- Overview of the Argo project
- Establishing the need for OS specialization
  - Lean OS
  - Provisioning for legacy applications
  - Provisioning for heterogeneous resources
  - Provisioning for different compute needs
- OS specialization via Compute Containers
- Single kernel, multiple OS personalities
- NodeOS and features
- Conclusion

Towards exascale computing

- Billions of execution threads
- Complex and composite workloads
- Highly heterogeneous sets of resources
  - Taking to another level the trend of mixing CPU cores, accelerators and various kinds of physical memories
- Variability
  - Changing job configuration and resource needs
- Resiliency

Overview of Argo

- A **node OS** at node level
- A user-level lightweight runtime for massive parallelism
- A System-wide signaling
- Global OS with global view.

OS Specialization

- An autonomous view of the OS meant for a specific use
- A specialization is characterized by:
  - Spanned resources
  - Set of exposed features, mechanisms and policies
  - Mandate:
    - e.g. Noise-sensitive computation
    - e.g. Heavy I/O
    - e.g. Tailored for heavy use of accelerator
    - etc.

Lean OS

Comparing Linux to a lightweight HPC OS kernel

Linux

- ~1.5 millions LoC (3.x kernels)

Blue Gene Q CNK

- ~60,000 LoC

Lean OS

Comparing Linux to a lightweight HPC OS kernel

Linux

- ~1.5 millions LoC (3.x kernels)
- General purpose
  - Tailored for myriads of uses

Blue Gene Q CNK

- ~60,000 LoC
- Special purpose
  - Built to allow HPC jobs to get the most out of hardware resources.

Lean OS

Comparing Linux to a lightweight HPC OS kernel

Linux

- ~1.5 millions LoC (3.x kernels)
- General purpose
  - Tailored for myriads of uses
- OS kernel can run on any core
  - Can be controlled

Blue Gene Q CNK

- ~60,000 LoC
- Special purpose
  - Built to allow HPC jobs to get the most out of hardware resources.
- OS kernel is strictly on dedicated core

Lean OS

Comparing Linux to a lightweight HPC OS kernel

Linux

- ~1.5 millions LoC (3.x kernels)
- General purpose
  - Tailored for myriads of uses
- OS kernel can run on any core
  - Can be controlled
- Has many kinds of per-CPU core kernel threads
  - All the rcuXXX
  - The watchdogs
  - The ksoftirqd
  - The kworkers
  - Other device-specific kernel threads (e.g. network)

Blue Gene Q CNK

- ~60,000 LoC
- Special purpose
  - Built to allow HPC jobs to get the most out of hardware resources.
- OS kernel is strictly on dedicated core
- No per-CPU core kernel thread

Lean OS

Comparing Linux to a lightweight HPC OS kernel

Linux

- ~1.5 millions LoC (3.x kernels)
- General purpose
  - Tailored for myriads of uses
- OS kernel can run on any core
  - Can be controlled
- Has many kinds of per-CPU core kernel threads
  - All the rcuXXX
  - The watchdogs
  - The ksoftirqd
  - The kworkers
  - Other device-specific kernel threads (e.g. network)
- Interference in HPC application at runtime … your mileage might vary

Blue Gene Q CNK

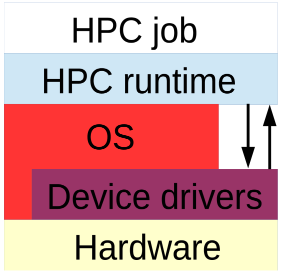
- ~60,000 LoC
- Special purpose
  - Built to allow HPC jobs to get the most out of hardware resources.
- OS kernel is strictly on dedicated core
- No per-CPU core kernel thread
- Extremely low interference in HPC application at runtime.

Lean OS

- Lean OSes like CNK reduce interference between the OS and the HPC job.
- HPC runtimes bypass lean OSes for hardware access.

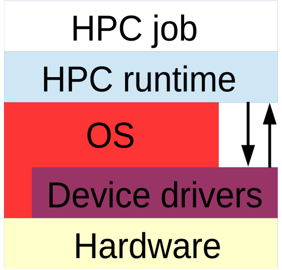
Lean OS

- Lean OSes like CNK reduce interference between the OS and the HPC job.
- HPC runtimes bypass lean OSes for hardware access.
- Traditional Linux-based HPC stacks already selectively bypass the OS for certain activities (E.g. RDMA)

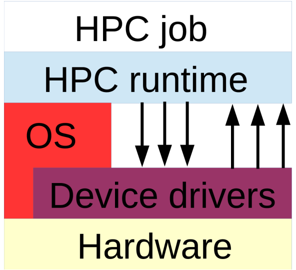


Lean OS

- Lean OSes like CNK reduce interference between the OS and the HPC job.
- HPC runtimes bypass lean OSes for hardware access.
- Traditional Linux-based HPC stacks already selectively bypass the OS for certain activities (E.g. RDMA)

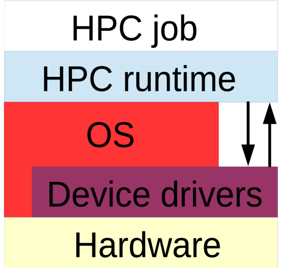


- The Argo NodeOS seeks to expose most hardware resources to the HPC runtime
  - Because the runtime knows more about the Application needs than the OS
  - HPC runtime want to make their own policies and craft their own fine-tuned optimizations
  - Right on top of the hardware

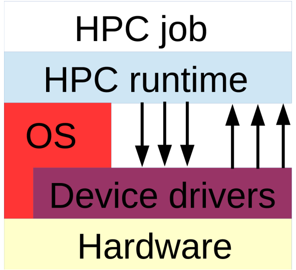


Lean OS

- Lean OSes like CNK reduce interference between the OS and the HPC job.
- HPC runtimes bypass lean OSes for hardware access.
- Traditional Linux-based HPC stacks already selectively bypass the OS for certain activities (E.g. RDMA)



- The Argo NodeOS seeks to expose most hardware resources to the HPC runtime
  - Because the runtime knows more about the Application needs than the OS
  - HPC runtime want to make their own policies and craft their own fine-tuned optimizations
  - Right on top of the hardware

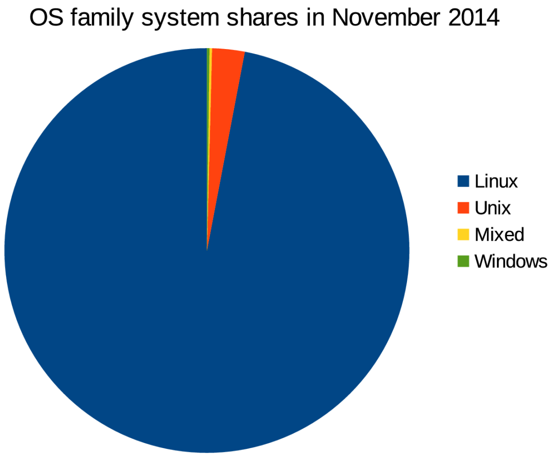


So … is the exascale OS going to be lean?

OS Specialization

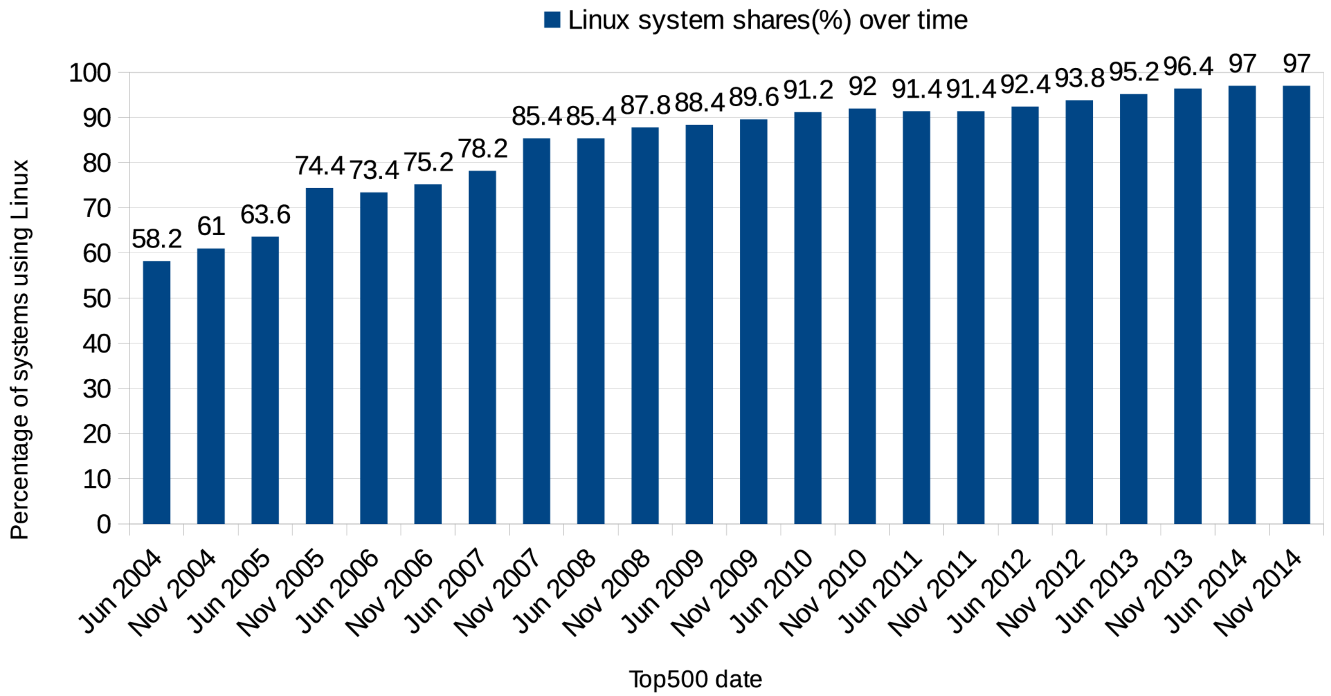
- As of November 2014, Linux-equipped systems delivered ~98.23% of the aggregated FLOPS of the 500 most powerful supercomputers (Top500).

- Linux equipped 97% of the Top500 systems



In fact, for each ranking in the past decade, more than half the Top500 systems used Linux

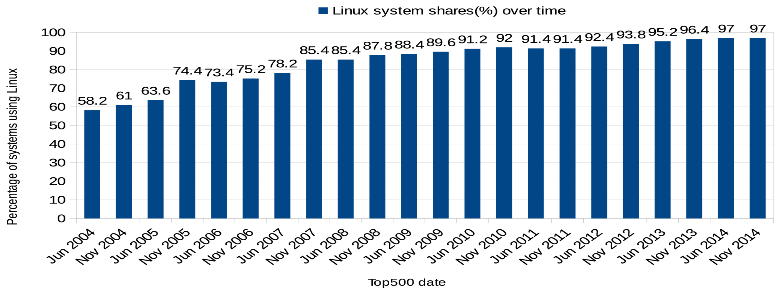
And the trend shows no change in direction!



OS Specialization

There is basically a massive amount of existing (legacy) HPC applications that assume a Linux-like environment:

- Some well-known system calls
- POSIX
- etc.



Back to … comparing Linux to a lightweight HPC OS kernel (a few differences)

Linux

- …
-

Blue Gene Q CNK

- …
- Offers only 63 system calls.
  - E.g., no forking
- No sophisticated virtual to physical memory mapping
- No time-quantum

Provisioning for legacy applications

- The next generation HPC OS cannot ignore the massive amount of existing legacy HPC code

Provisioning for legacy applications

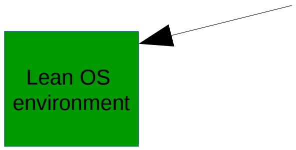
- The next generation HPC OS cannot ignore the massive amount of existing legacy HPC code
- An OS specialization is required that offers something like a fully-fledged Linux environment

Provisioning for legacy applications

- The next generation HPC OS cannot ignore the massive amount of existing legacy HPC code
- An OS specialization is required that offers something like a fully-fledged Linux environment
- We are designing the Argo exascale NodeOS as not simply lean or simply fully-fledged Linux; but both simultaneously

Provisioning for legacy applications

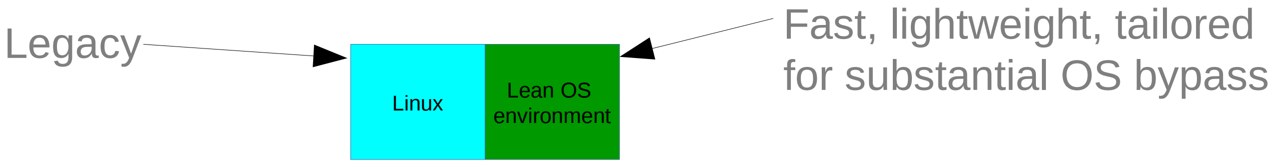
- The next generation HPC OS cannot ignore the massive amount of existing legacy HPC code
- An OS specialization is required that offers something like a fully-fledged Linux environment
- We are designing the Argo exascale NodeOS as not simply lean or simply fully-fledged Linux; but both simultaneously



Fast, lightweight, tailored for substantial OS bypass

Provisioning for legacy applications

- The next generation HPC OS cannot ignore the massive amount of existing legacy HPC code
- An OS specialization is required that offers something like a fully-fledged Linux environment
- We are designing the Argo exascale NodeOS as not simply lean or simply fully-fledged Linux; but both simultaneously



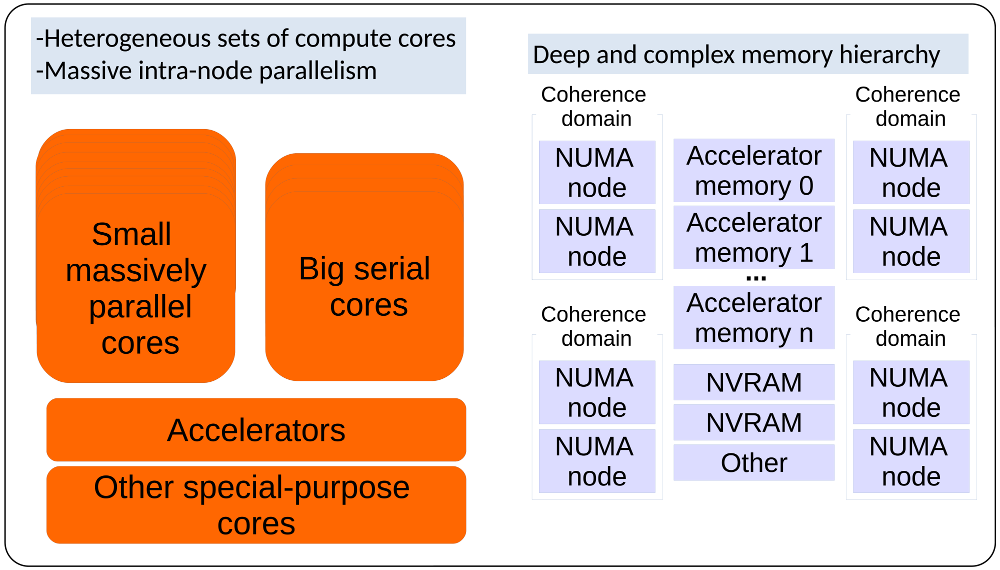
Provisioning for heterogeneous hardware resources

- Nowadays supercomputers are more and more heterogeneous
    - E.g., Tianhe-2 with Intel MIC + regular x86_64
    - E.g., Titan (Cray XK-7) with NVIDIA GPU + regular x86_64

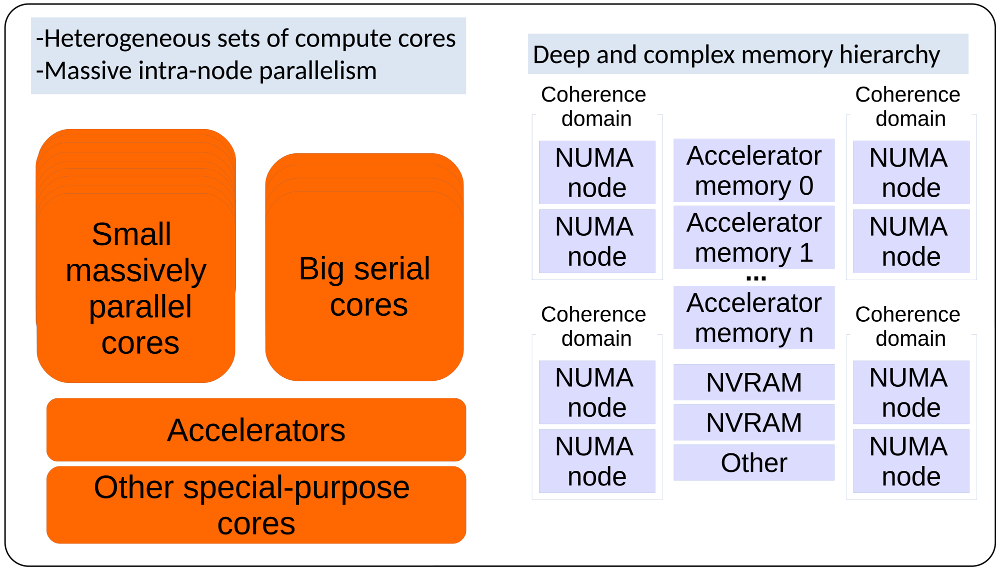
Provisioning for heterogeneous hardware resources

- Nowadays supercomputers are more and more heterogeneous
  - E.g., Tianhe-2 with Intel MIC + regular x86_64
  - E.g., Titan (Cray XK-7) with NVIDIA GPU + regular x86_64
- The heterogeneity will be pushed further for exascale systems:
  - Massive numbers of small cores
  - Big serial cores
  - Accelerators
  - Deeper and more complex memory hierarchies
    - Multiple NUMA domains
    - Multiple coherence domains

Provisioning for heterogeneous hardware resources

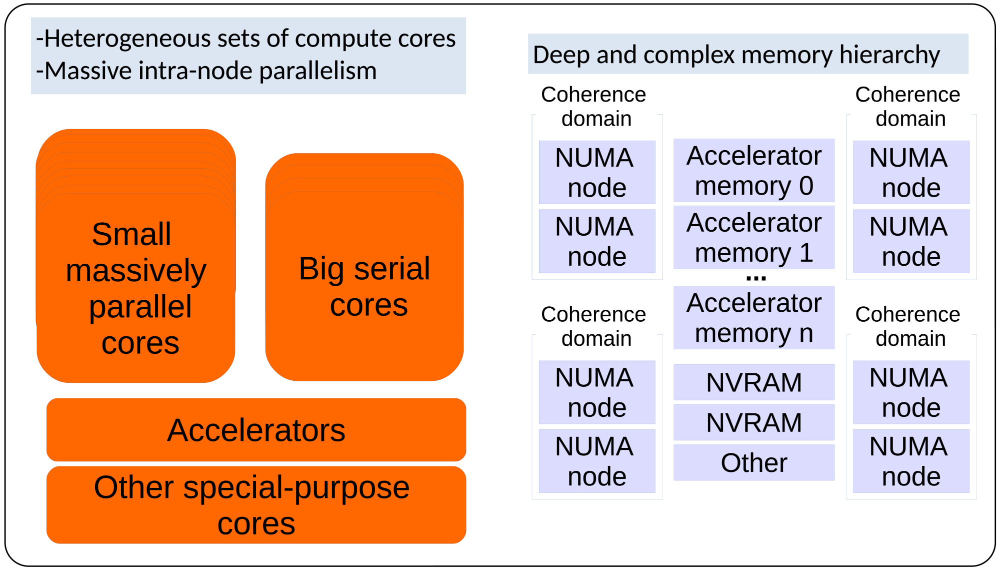


Provisioning for heterogeneous hardware resources

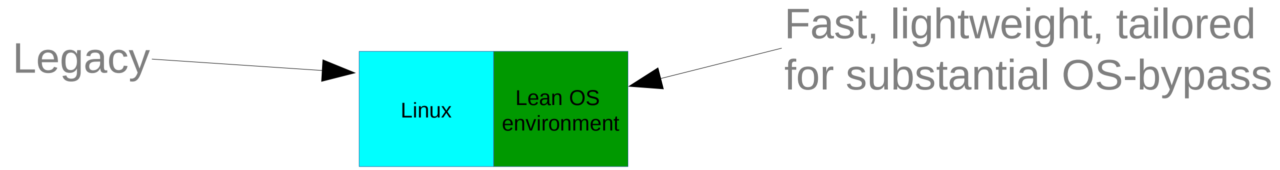


- New HPC hardware vendors do not target exotic or niche OSes.
- HPC hardware vendors target well-established OSes (e.g., Linux)

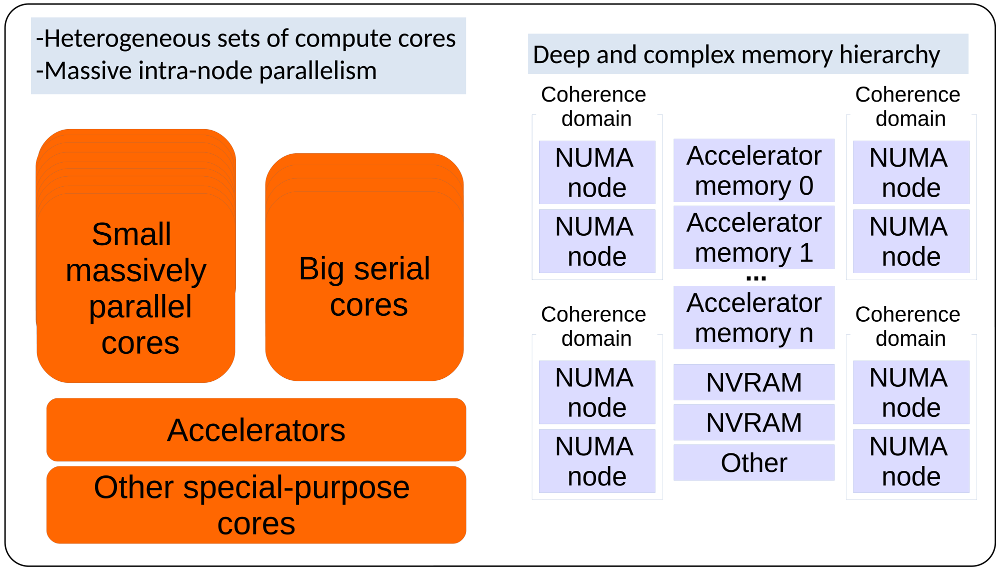
Provisioning for heterogeneous hardware resources



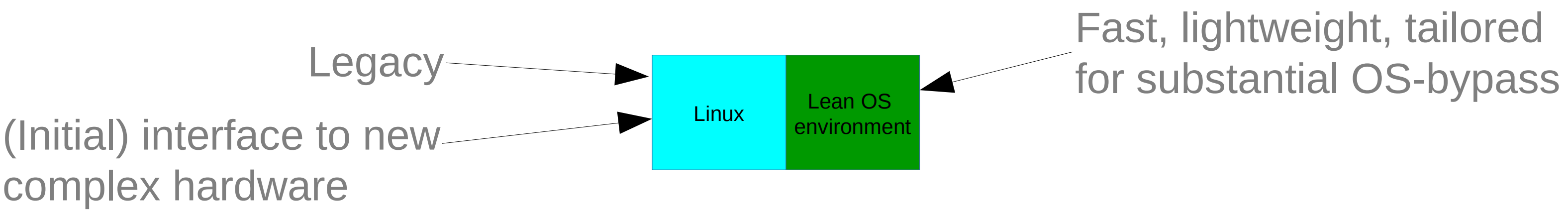
- New HPC hardware vendors do not target exotic or niche OSes.
- HPC hardware vendors target well-established OSes (e.g., Linux)



Provisioning for heterogeneous hardware resources



- New HPC hardware vendors do not target exotic or niche OSes.
- HPC hardware vendors target well-established OSes (e.g., Linux)



Provisioning for different compute needs

- What features should a lean OS environment provide for a “broadly useful” supercomputer?
  - What scheduling policies (if any at all)?
  - What system calls?
  - What memory allocation mechanism?
  - … ?

Provisioning for different compute needs

- What features should a lean OS environment provide for a “broadly useful” supercomputer?
  - What scheduling policies (if any at all)?
  - What system calls?
  - What memory allocation mechanism?
  - … ?
- Distinct HPC jobs have distinct needs

Provisioning for different compute needs

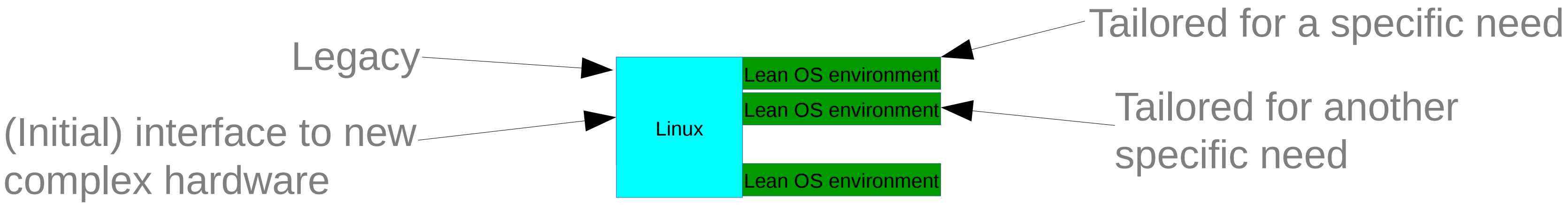
- What features should a lean OS environment provide for a “broadly useful” supercomputer?
  - What scheduling policies (if any at all)?
  - What system calls?
  - What memory allocation mechanism?
  - … ?
- Distinct HPC jobs have distinct needs
- The same unique HPC job can be multi-aspect (e.g. compute + co-visualization) with different aspects having different needs.

Provisioning for different compute needs

- What features should a lean OS environment provide for a “broadly useful” supercomputer?
  - What scheduling policies (if any at all)?
  - What system calls?
  - What memory allocation mechanism?
  - … ?
- Distinct HPC jobs have distinct needs
- The same unique HPC job can be multi-aspect (e.g. compute + co-visualization) with different aspects having different needs.
- No single set of lean OS characteristics can fit all the various compute needs … without voiding the leanness.

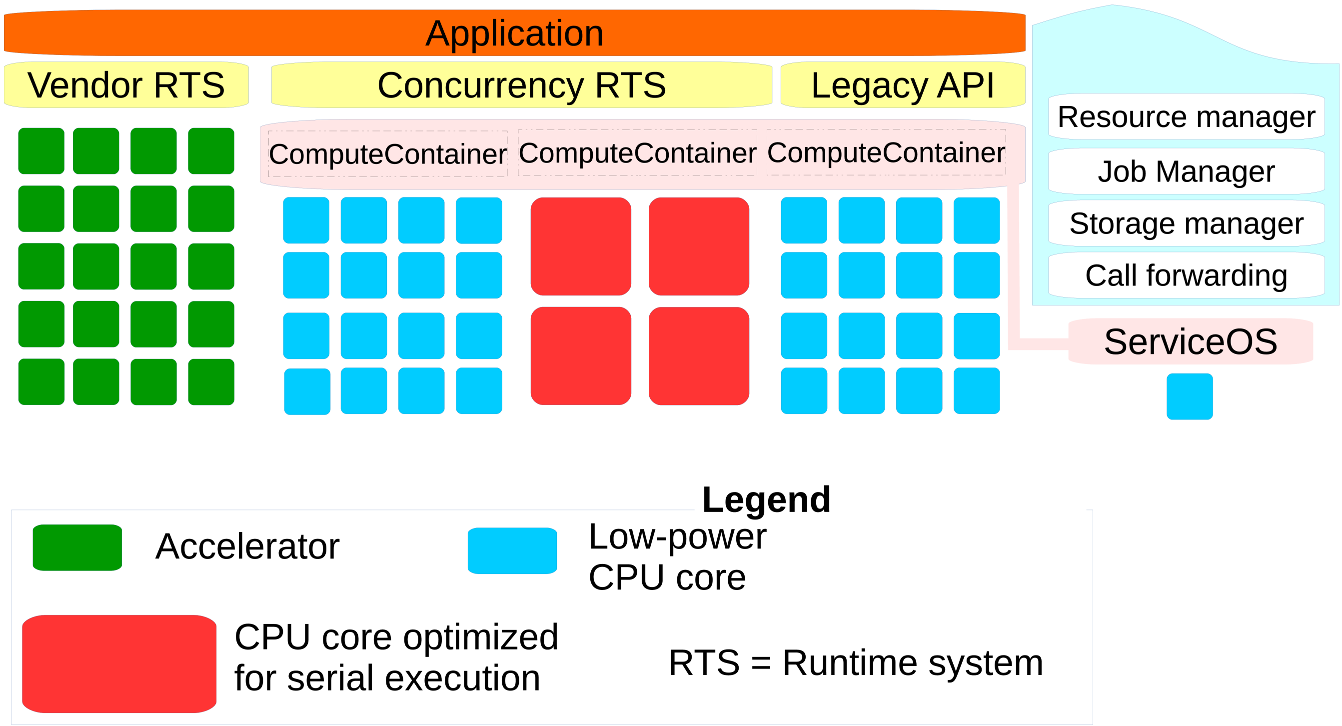
Provisioning for different compute needs

- What features should a lean OS environment provide for a “broadly useful” supercomputer?
  - What scheduling policies (if any at all)?
  - What system calls?
  - What memory allocation mechanism?
  - ... ?
- Distinct HPC jobs have distinct needs
- The same unique HPC job can be multi-aspect (e.g. compute + co-visualization) with different aspects having different needs.
- No single set of lean OS characteristics can fit all the various compute needs ... without voiding the leanness.



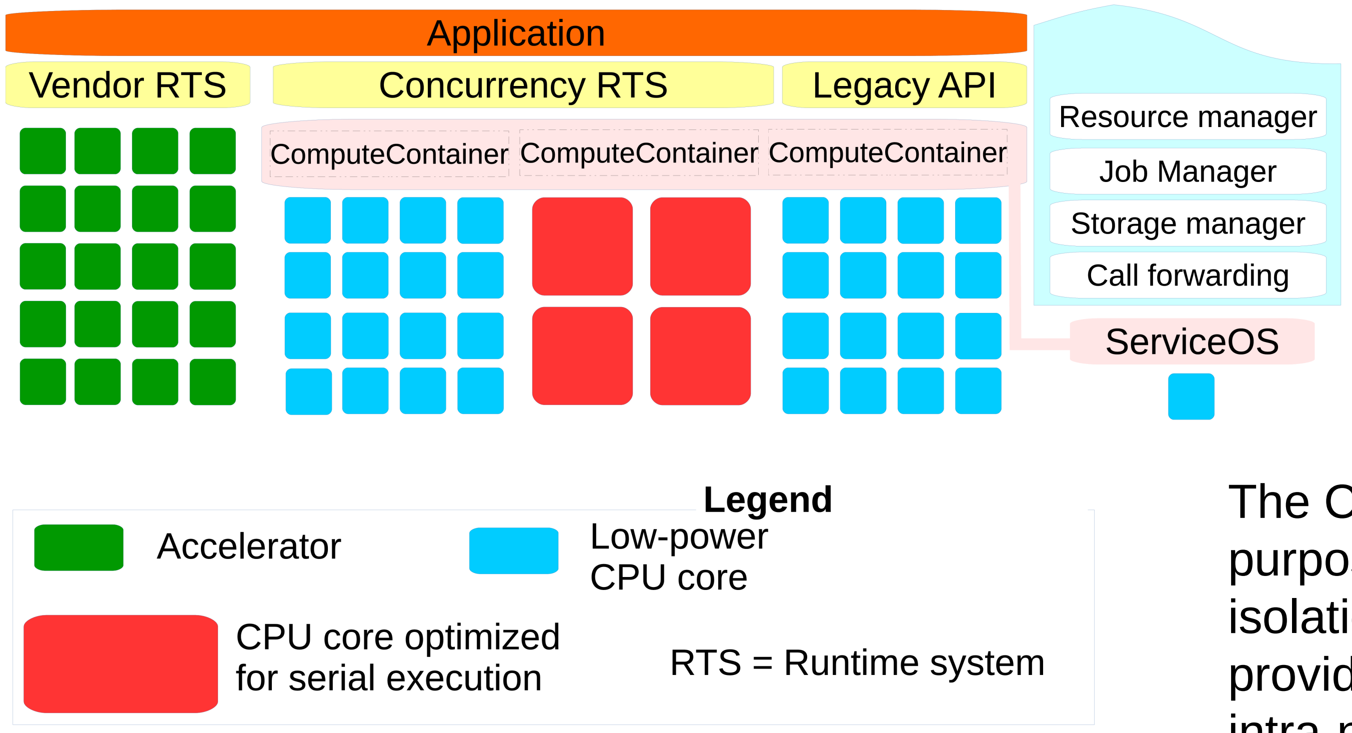
NodeOS: OS Specialization via Compute Containers

The Argo NodeOS is <u>specialized</u> into a single *ServiceOS* and one or multiple *Compute Containers*



NodeOS: OS Specialization via Compute Containers

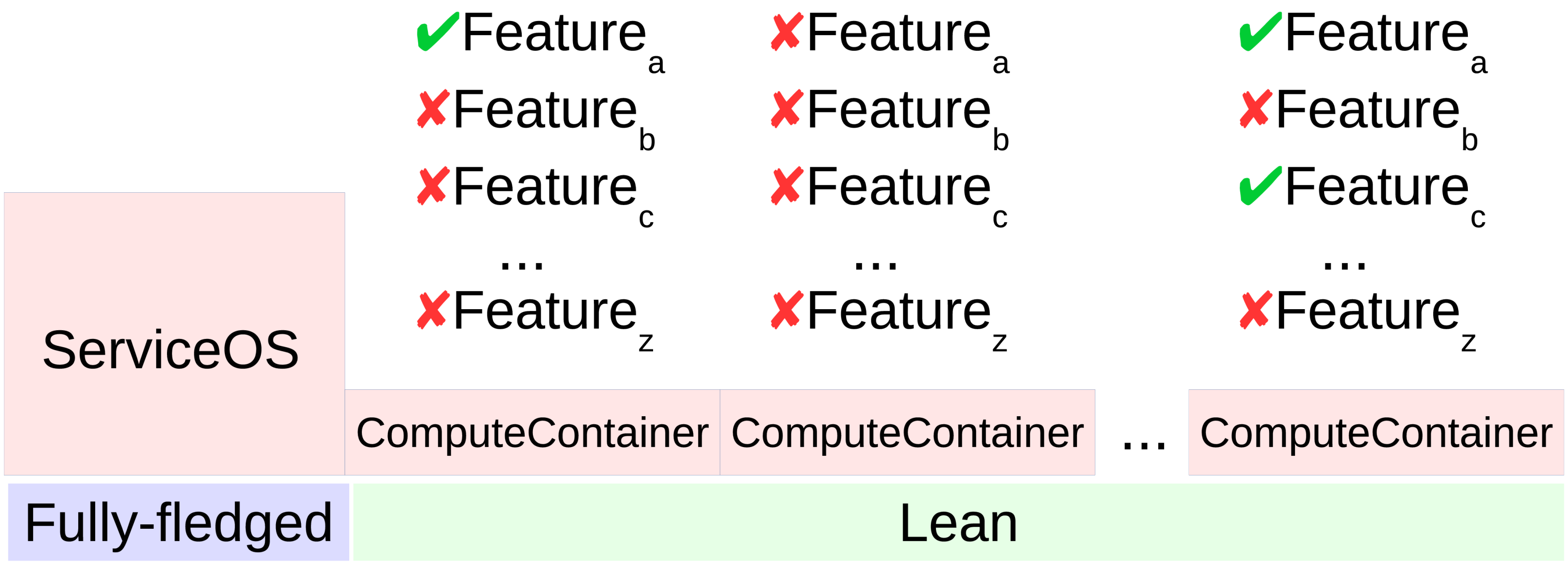
The Argo NodeOS is <u>specialized</u> into a single *ServiceOS* and one or multiple *Compute Containers*



The Compute Containers purposely do not provide isolation; for the sake of providing seamless intra-node communication

NodeOS: Single kernel, multiple OS personalities

- The specialization occurs over a single kernel
- The kernel is fully-fledged for the ServiceOS
- The kernel is made selectively lean for the Compute Containers



NodeOS: HPC-specific features for container specialization

- Behaviors and features exposed by a Compute Container are decided (or requested) by its “clients”

NodeOS: HPC-specific features for container specialization

- Behaviors and features exposed by a Compute Container are decided (or requested) by its "clients"
- Examples of clients are the HPC runtimes or the Global OS

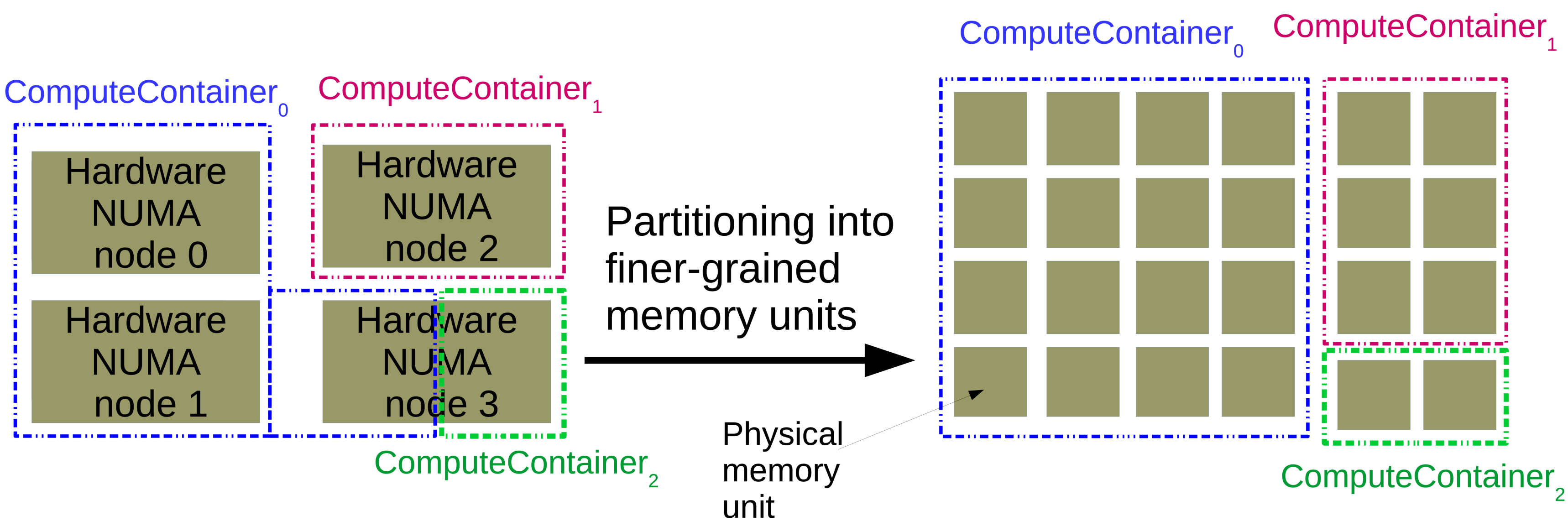
NodeOS: HPC-specific features for container specialization

- Behaviors and features exposed by a Compute Container are decided (or requested) by its “clients”
- Examples of clients are the HPC runtimes or the Global OS
- The NodeOS interface to its clients is made of:
  - Configuration daemons, scripts or binary executables
  - New API for functionalities that were not natively exposed by the host kernel
  - Wrapped or substituted implementations for existing API functions that are expected to behave differently inside Compute Containers (e.g., making certain system calls non-blocking)

HPC runtimes want exclusive ownership of the resources that they use

e.g. physical memory

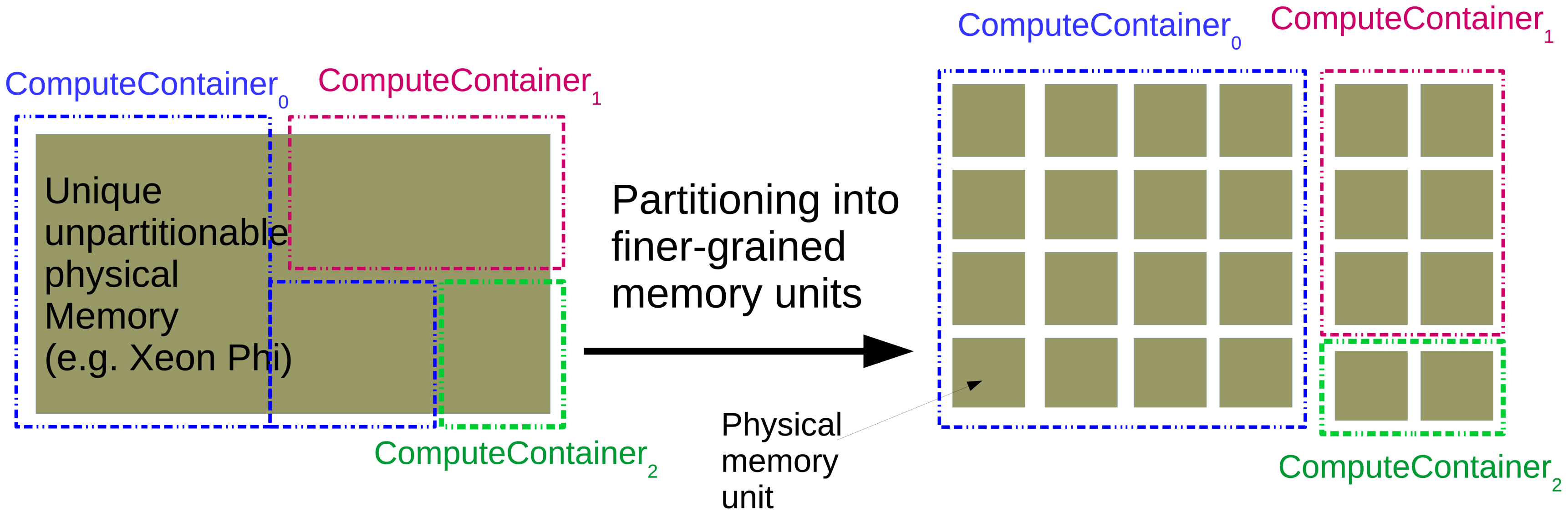
Finer-grained memory units (NUMA)



ComputeContainers can have ***guaranteed*** page frames

Static virtual-physical mapping => Free memory pinning for faster RDMA

Finer-grained memory units (UMA)



ComputeContainers can have ***guaranteed*** page frames

Static virtual-physical mapping => Free memory pinning for faster RDMA

HPC runtimes are multiple and disparate; and some (mostly legacy) are not necessarily well-equipped for their own needs

e.g. scheduling behavior

New HPC scheduling class

- Optimized for Compute Containers with guarantees of absence of oversubscribing.
- Disables load balancing and preemption
- Reduces kernel bookkeeping
- Provides predictable performance (as much as possible) for the same workload.

HPC runtimes want some of the same functionalities provided by vanilla Linux ... without giving up their freedom.

e.g. system calls … without ever blocking

Completely wait-free system API

- Cooperative scheduling is “the thing” some next generation HPC concurrency runtimes are built around.

Completely wait-free system API

- Cooperative scheduling is “the thing” some next generation HPC concurrency runtimes are built around.
- The user-level threads should not block; they should always yield instead

Completely wait-free system API

- Cooperative scheduling is “the thing” some next generation HPC concurrency runtimes are built around.

- The user-level threads should not block; they should always yield instead

What guarantee does the cooperative scheduling concurrency runtime provide if user-level threads can make system calls?

Completely wait-free system API

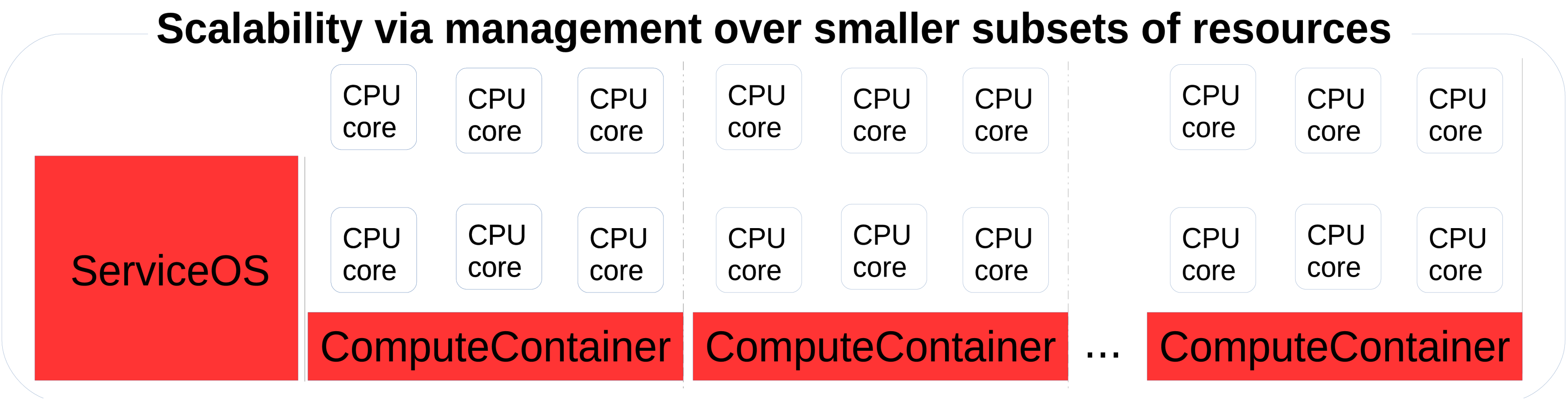
- System calls behaviors can be container-specific; that is, same API, different behaviors depending on the Compute Container hosting the calling process.

Completely wait-free system API

- System calls behaviors can be container-specific; that is, same API, different behaviors depending on the Compute Container hosting the calling process.
- Provision for all non-blocking system calls with EWOULDBLOCK or E_AGAIN returned for calling threads that have wait-freedom requirements:
  - To fulfill the concern of completely wait-free execution if desired.
  - To fulfill the need for predictability in blocking behaviors.

NodeOS: Compute Container specialization aspect

Scalability through divide and conquer



- Trade kernel-wide management of certain internal data structures with per-Compute Container approaches
- Only the subset of resources spanned by a Compute Container is considered

  e.g., RCU grace periods

Conclusion

- The Argo node operating system *specializes* a single kernel into multiple aspects that provide:
  - Lean OS environments for various OS-bypass needs and next generation HPC runtime support
  - Fully-fledged Linux environment:
    - Node booting
    - complex resource management
    - Bulk resource allocation
    - Legacy application execution

- The specialization is fulfilled over the Linux kernel with cgroups, resource controllers and new kernel additions
- Prototype sources to be made public

Questions?