SAND2014-20457PE

Sandia National Laboratories

Exceptional service in the national interest

Hobbes Extreme Scale OS

Ron Brightwell
R&D Manager
Scalable System Software
Center for Computing Research
Sandia National Laboratories

SPPEXA Workshop on Application Interfaces for an Exascale OS
TU Dresden
December 8, 2014

U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

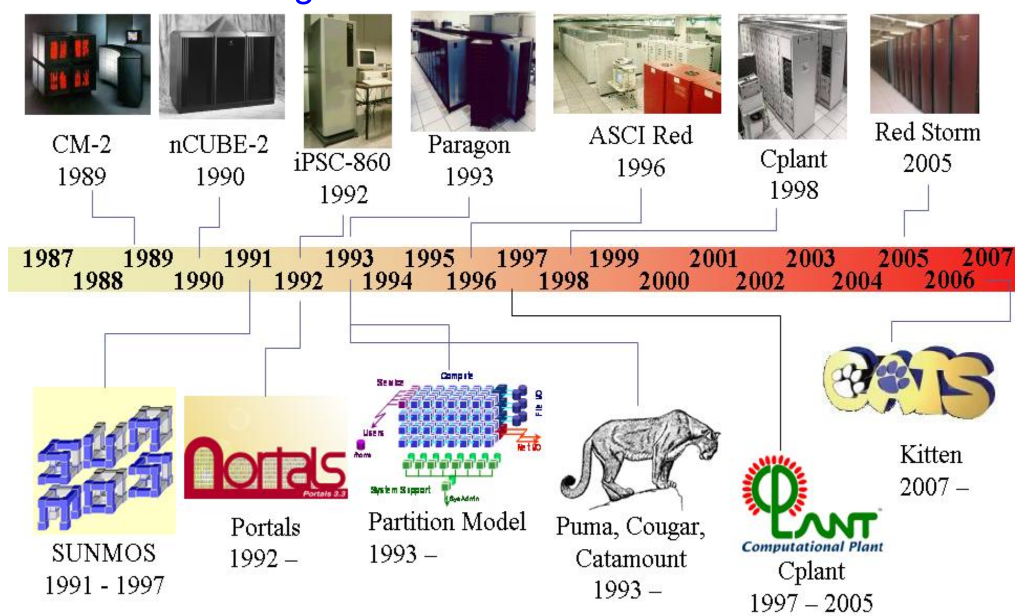
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

System Software@Sandia

- Established the functional partition model for HPC systems
  - Tailor system software to function (compute, I/O, user services, etc.)
- Pioneered the research, development, and use of lightweight kernel operating systems for HPC
  - Only DOE lab to deploy OS-level software on large-scale production machines
  - Provided blueprint for IBM BG/L,Q CNK
- Set the standard for scalable parallel runtime systems for HPC
  - Fast application launch on tens of thousands of processors
- Significant impact in the design and of scalable HPC interconnect APIs
  - Only DOE lab to deploy low-level interconnect API on large-scale production machines

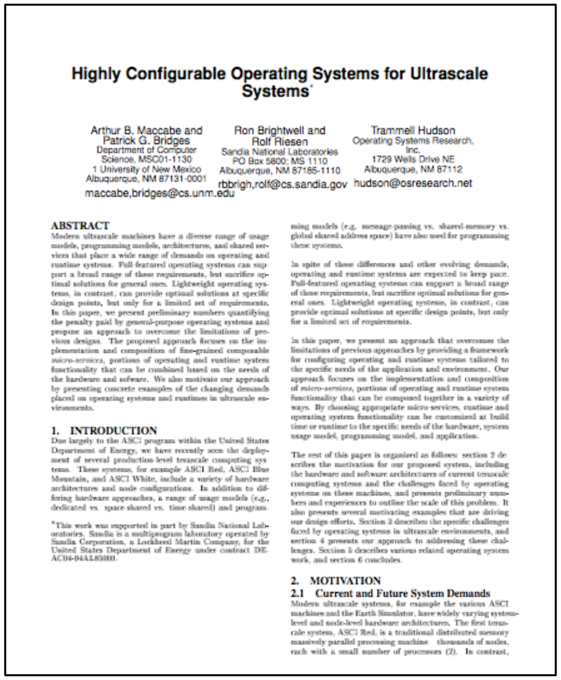
AWARDS:

- **1998** Sandia Meritorious Achievement Award, TeraFLOP Computer Installation Team
- **2006** Sandia Meritorious Achievement Award, Red Storm Design, Development and Deployment Team
- **2006** NOVA Award Red Storm Design and Development Team
- **2009** R&D 100 Award for Catamount N-Way Lightweight Kernel
- **2010** Excellence in Technology Transfer Award, Federal Laboratory Consortium for Technology Transfer
- **2010** National Nuclear Security Administration Defense Programs Award of Excellence



Configurable OS Project

- Part of the US DOE FAST-OS Program
  - Partnership between Sandia, UNM, LSU
  - 2005-7
- Lessons learned
  - HPC users want Linux environment
    - Most only care about toolchain
  - Very little middle ground between LWK and Linux
  - Difficult to concentrate on $500K research project when $75M Red Storm system needs attention



Highly Configurable Operating Systems for Ultrascale Systems*

Arthur B. Maccabe and Patrick G. Bridges
Department of Computer Science, MSC01-1130
1 University of New Mexico
Albuquerque, NM 87131-0001
maccabe,bridges@cs.unm.edu

Ron Brightwell and Rolf Riesen
Sandia National Laboratories
PO Box 5800; MS 1110
Albuquerque, NM 87185-1110
rbbrigh,rolf@cs.sandia.gov

Trammell Hudson
Operating Systems Research, Inc.
1729 Wells Drive NE
Albuquerque, NM 87112
hudson@osresearch.net

ABSTRACT

Modern ultrascale machines have a diverse range of usage models, programming models, architectures, and shared services that place a wide range of demands on operating and runtime systems. Full-featured operating systems can support a broad range of these requirements, but sacrifice optimal solutions for general ones. Lightweight operating systems, in contrast, can provide optimal solutions at specific design points, but only for a limited set of requirements. In this paper, we present preliminary numbers quantifying the penalty paid by general-purpose operating systems and propose an approach to overcome the limitations of previous designs. The proposed approach focuses on the implementation and composition of fine-grained composable *micro-services*, portions of operating and runtime system functionality that can be combined based on the needs of the hardware and sofware. We also motivate our approach by presenting concrete examples of the changing demands placed on operating systems and runtimes in ultrascale environments.

1. INTRODUCTION

Due largely to the ASCI program within the United States Department of Energy, we have recently seen the deployment of several production-level terascale computing systems. These systems, for example ASCI Red, ASCI Blue Mountain, and ASCI White, include a variety of hardware architectures and node configurations. In addition to differing hardware approaches, a range of usage models (e.g., dedicated vs. space-shared vs. time-shared) and programming models (e.g. message-passing vs. shared-memory vs. global shared address space) have also used for programming these systems.

In spite of these differences and other evolving demands, operating and runtime systems are expected to keep pace. Full-featured operating systems can support a broad range of these requirements, but sacrifice optimal solutions for general ones. Lightweight operating systems, in contrast, can provide optimal solutions at specific design points, but only for a limited set of requirements.

In this paper, we present an approach that overcomes the limitations of previous approaches by providing a framework for configuring operating and runtime systems tailored to the specific needs of the application and environment. Our approach focuses on the implementation and composition of *micro-services*, portions of operating and runtime system functionality that can be composed together in a variety of ways. By choosing appropriate micro-services, runtime and operating system functionality can be customized at build time or runtime to the specific needs of the hardware, system usage model, programming model, and application.

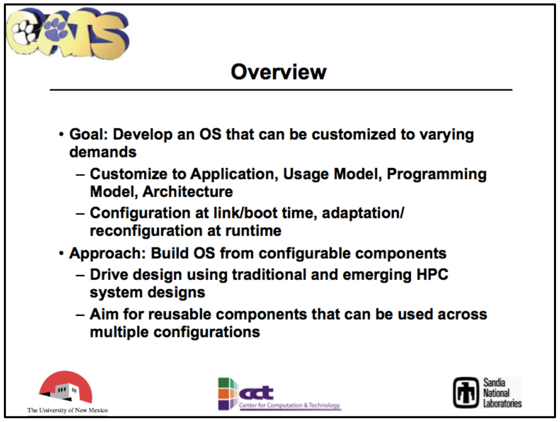
The rest of this paper is organized as follows: section 2 describes the motivation for our proposed system, including the hardware and software architectures of current terascale computing systems and the challenges faced by operating systems on these machines, and presents preliminary numbers and experiences to outline the scale of this problem. It also presents several motivating examples that are driving our design efforts. Section 3 describes the specific challenges faced by operating systems in ultrascale environments, and section 4 presents our approach to addressing these challenges. Section 5 describes various related operating system work, and section 6 concludes.

2. MOTIVATION

2.1 Current and Future System Demands

Modern ultrascale systems, for example the various ASCI machines and the Earth Simulator, have widely varying system-level and node-level hardware architectures. The first terascale system, ASCI Red, is a traditional distributed memory massively parallel processing machine – thousands of nodes, each with a small number of processors (2). In contrast,

*This work was supported in part by Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

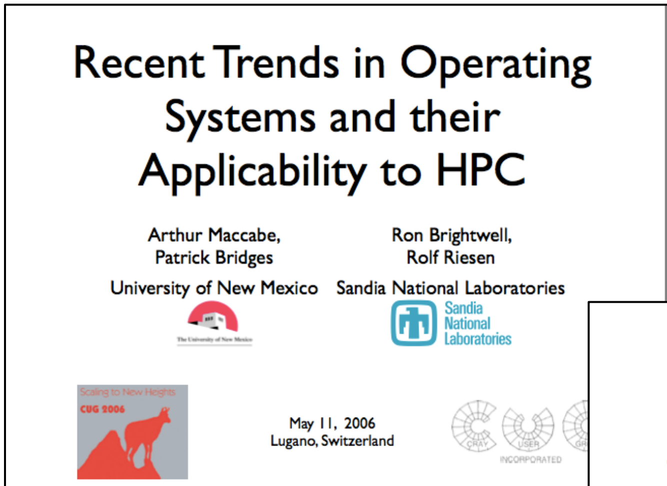


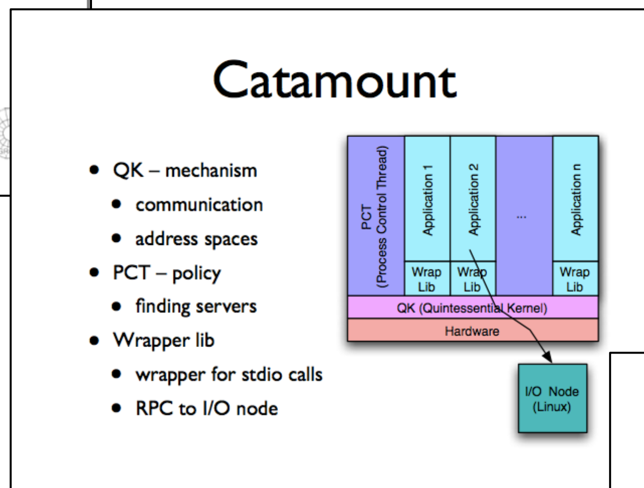
Overview

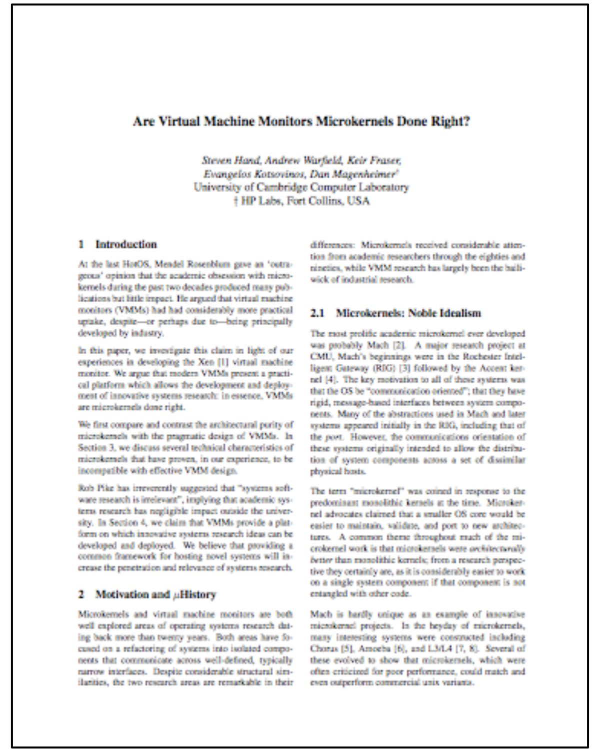
- Goal: Develop an OS that can be customized to varying demands
  - Customize to Application, Usage Model, Programming Model, Architecture
  - Configuration at link/boot time, adaptation/ reconfiguration at runtime
- Approach: Build OS from configurable components
  - Drive design using traditional and emerging HPC system designs
  - Aim for reusable components that can be used across multiple configurations

The University of New Mexico
Center for Computation & Technology
Sandia National Laboratories

Virtualization May Help







Are Virtual Machine Monitors Microkernels Done Right?

Steven Hand, Andrew Warfield, Keir Fraser, Evangelos Kotsovinos, Dan Magenheimer†
University of Cambridge Computer Laboratory
† HP Labs, Fort Collins, USA

1 Introduction

At the last HotOS, Mendel Rosenblum gave an 'outrageous' opinion that the academic obsession with microkernels during the past two decades produced many publications but little impact. He argued that virtual machine monitors (VMMs) had had considerably more practical uptake, despite—or perhaps due to—being principally developed by industry.

In this paper, we investigate this claim in light of our experiences in developing the Xen [1] virtual machine monitor. We argue that modern VMMs present a practical platform which allows the development and deployment of innovative systems research: in essence, VMMs are microkernels done right.

We first compare and contrast the architectural purity of microkernels with the pragmatic design of VMMs. In Section 3, we discuss several technical characteristics of microkernels that have proven, in our experience, to be incompatible with effective VMM design.

Rob Pike has irreverently suggested that "systems software research is irrelevant", implying that academic systems research has negligible impact outside the university. In Section 4, we claim that VMMs provide a platform on which innovative systems research ideas can be developed and deployed. We believe that providing a common framework for hosting novel systems will increase the penetration and relevance of systems research.

2 Motivation and μHistory

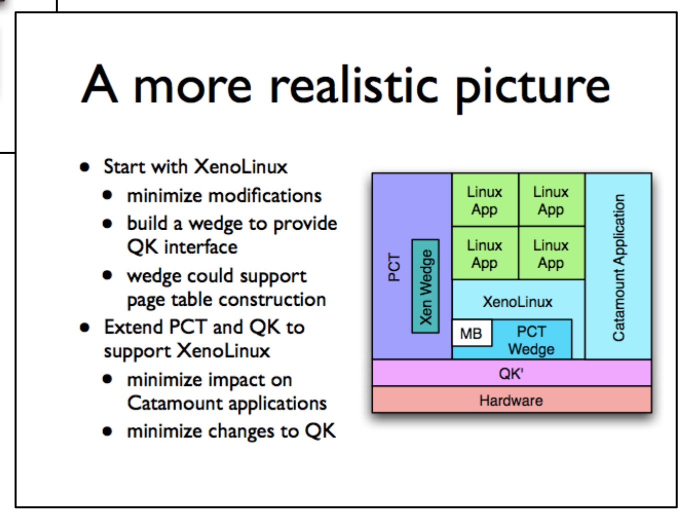
Microkernels and virtual machine monitors are both well explored areas of operating systems research dating back more than twenty years. Both areas have focused on a refactoring of systems into isolated components that communicate across well-defined, typically narrow interfaces. Despite considerable structural similarities, the two research areas are remarkable in their differences: Microkernels received considerable attention from academic researchers through the eighties and nineties, while VMM research has largely been the bailiwick of industrial research.

2.1 Microkernels: Noble Idealism

The most prolific academic microkernel ever developed was probably Mach [2]. A major research project at CMU, Mach's beginnings were in the Rochester Intelligent Gateway (RIG) [3] followed by the Accent kernel [4]. The key motivation to all of these systems was that the OS be "communication oriented"; that they have rigid, message-based interfaces between system components. Many of the abstractions used in Mach and later systems appeared initially in the RIG, including that of the *port*. However, the communications orientation of these systems originally intended to allow the distribution of system components across a set of dissimilar physical hosts.

The term "microkernel" was coined in response to the predominant monolithic kernels at the time. Microkernel advocates claimed that a smaller OS core would be easier to maintain, validate, and port to new architectures. A common theme throughout much of the microkernel work is that microkernels were *architecturally better* than monolithic kernels; from a research perspective they certainly are, as it is considerably easier to work on a single system component if that component is not entangled with other code.

Mach is hardly unique as an example of innovative microkernel projects. In the heyday of microkernels, many interesting systems were constructed including Chorus [5], Amoeba [6], and L3/L4 [7, 8]. Several of these evolved to show that microkernels, which were often criticized for poor performance, could match and even outperform commercial unix variants.



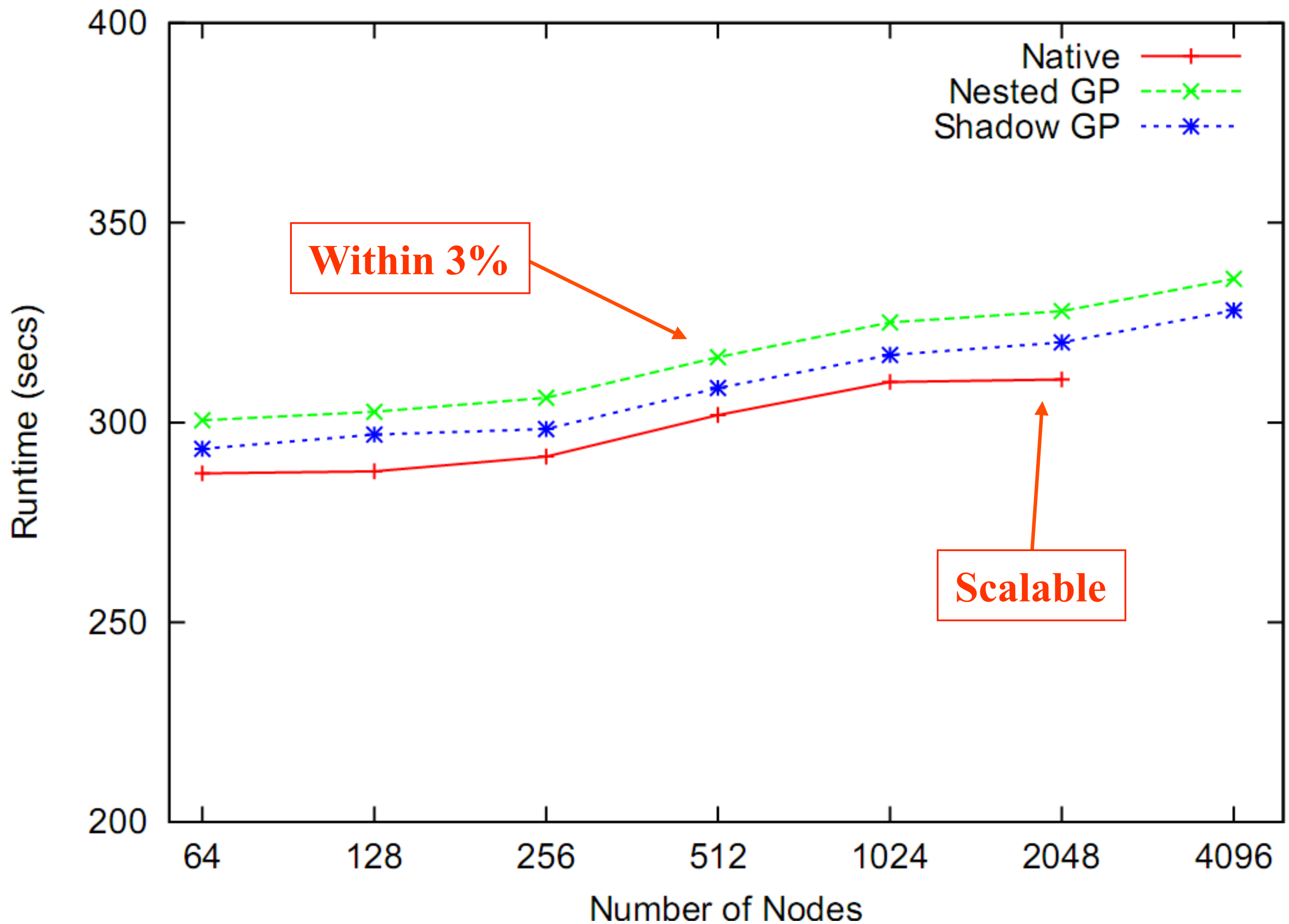
Kitten Lightweight Kernel

- Monolithic, C code, GNU toolchain, Kbuild configuration
- Supports x86-64 architecture only, porting to ARM
  - Boots on standard PC architecture, Cray XT, and in virtual machines
  - Boots identically to Linux (Kitten bzImage and init_task)
- Repurposes basic functionality from Linux
  - Hardware bootstrap
  - Basic OS kernel primitives (lists, locks, wait queues, etc.)
  - Directory structure similar to Linux, arch dependent/independent dirs
- Custom address space management and task management
  - User-level API for managing physical memory, building virtual address spaces
  - User-level API for creating tasks, which run in virtual address spaces
- Small, highly reliable code base
- Focused on scalable HPC applications
  - Low noise
  - Small memory footprint
- Open source and freely available

Palacios Virtual Machine Monitor

- OS-independent embeddable virtual machine monitor
  - Can be combined with Kitten or Linux
- Full system virtualization
  - No need to modify guest OS
- Supports running multiple guests concurrently
- Makes extensive use of virtualization extensions in modern Intel and AMD x86 processors
- Passthrough resource partitioning
- Extensive configurability
- Low noise
- Open source and freely available
- Small, highly reliable code base
- Developed at Northwestern and University of New Mexico

Low Overhead for Palacios+Kitten on Red Storm (2009)



CTH: multi-material, large deformation, strong shockwave simulation

DOE/ASCR X-Stack 2012

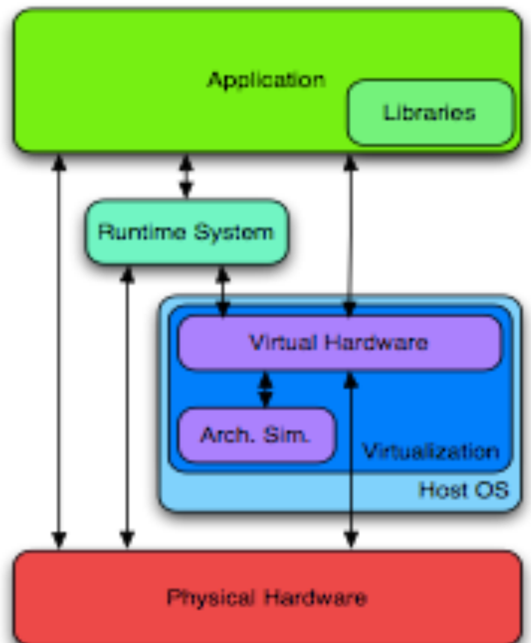
Enabling Exascale Hardware and Software Design through Scalable System Virtualization

Patrick G. Bridges, University of New Mexico; Peter Dinda, Northwestern University;
Jack Lange, University of Pittsburgh; Kevin Pedretti, Sandia National Laboratories;
Stephen Scott, Oak Ridge National Laboratory

Overview

In this project, we are investigating system software tools to accelerate the development and use of exascale systems. In particular, we are developing new system virtualization techniques to enable the development of the hardware and software innovations needed to enable exascale systems. Virtualization techniques provide traction on a wide range of exascale design and development issues, as described below.

We are using virtualization to enable the design, development, and use of exascale systems. Virtualization allows new hardware and software features to be prototyped as extensions to a virtual machine monitor (VMM), making them immediately available for experimentation and use. Furthermore, it allows new system software and runtime stacks to be launched above production host operating systems without the need for dedicated system time. The VMM



DOE LAB 13-02 FOA – January 2013

Exascale Operating and Runtime Systems Program

- $6M of funding for OS/R research at US DOE labs
- Focus areas
  - Power management
    - Adaptive power management to meet 20 MW goal
  - Support for dynamic programming environments
    - Manage billions of threads
  - Programmability and tuning support
    - Dynamic adaptation and debugging
  - Resilience
    - Predict, detect, contain, and recover from faults
  - Heterogeneity
    - Hierarchical process and memory systems
  - Memory management
    - Use of new memory technologies
  - Global optimization
    - Manage resources with a system-wide view

Exascale OS/R Focus is on Hardware

- Reliability/Resilience
- Power/Energy
- Heterogeneity
- Memory hierarchy
- Cores, cores, and more cores

- Risk
  - Hardware advancements and investments can provide orders of magnitude improvement
  - OS/R advancements can provide double-digit percentage improvement

What About Applications?

- Focus is on parallel (many-core) programming model
  - Adaptive runtime systems
  - Node-level resource allocation and management
  - Managing locality
  - Extracting parallelism
  - Introspective, adaptive capabilities
    - US DOE/ASCR XPRESS project is addressing OS support for adaptive RTS
- Risk
  - Incremental approach (OpenMP) wins
    - Advanced runtime capabilities are overkill
  - No clear on-node parallel programming model winner
    - Difficult to optimize OS/R

Application Composition Will Be Increasingly Important at Extreme-Scale

- More complex workflows are driving need for advanced OS services and capability
  - Exascale applications will continue to evolve beyond a space-shared batch scheduled approach
- HPC application developers are employing ad-hoc solutions
  - Interfaces and tools like mmap, ptrace, python for coupling codes and sharing data
- Tools stress OS functionality because of these legacy APIs and services
- More attention needed on how multiple applications are composed
- Several use cases
  - Ensemble calculations for uncertainty quantification
  - Multi-{material, physics, scale} simulations
  - In-situ analysis
  - Graph analytics
  - Performance and correctness tools
- Requirements are driven by applications
  - Not necessarily by parallel programming model
  - Somewhat insulated from hardware advancements

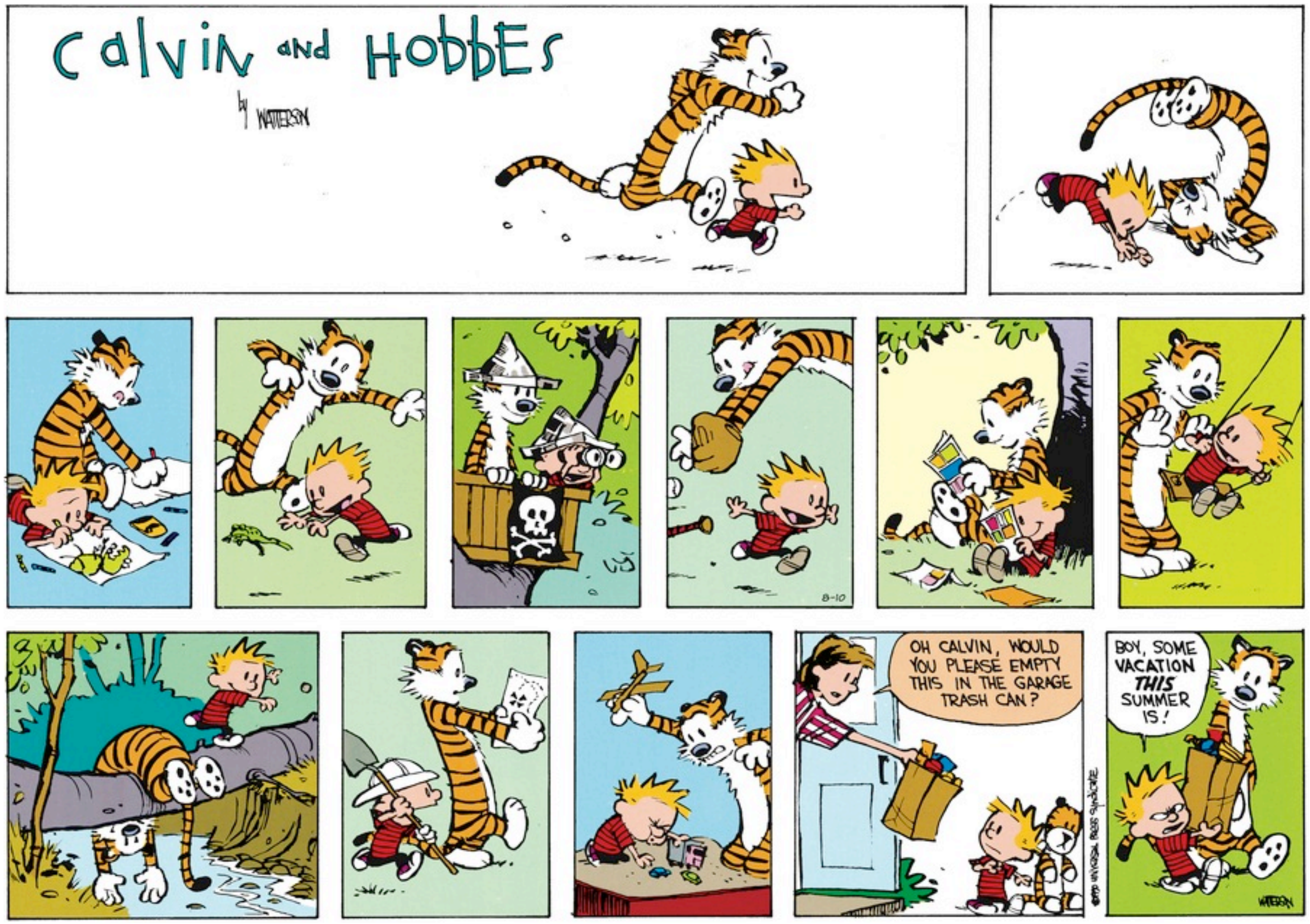
Hobbes Project Goals

- Deliver prototype OS/R environment for R&D in extreme-scale scientific computing
- Focus on application composition as a fundamental driver
  - Develop necessary OS/R interfaces and system services required to support resource isolation and sharing
  - Support complex simulation and analysis workflows
- Provide a lightweight OS/R environment with flexibility to build custom runtimes
  - Compose applications from a collection of enclaves
- Leverage Kitten lightweight kernel and Palacios lightweight virtual machine monitor
  - Enable high-risk high-impact research in virtualization, energy/power, scheduling, and resilience
- Enable High-Risk/High-Impact R&D in key areas

Fundamental Principles of Our Approach

- OS/R must be viewed as technologies that enable and support the research and development of other critical capabilities required for effective use of extreme-scale high-performance computing (HPC) systems
- OS/R support for composition of applications is a critical capability that will be the foundation of the way extreme-scale systems must be used in the future
- Addressing near-term OS/R challenges such as energy/power, scheduling, and resilience without considering application composition will lead to incomplete solutions

About the Name….



Or Possibly…

- HPC
- OS
- Building
- Blocks for
- Extreme-scale
- Systems

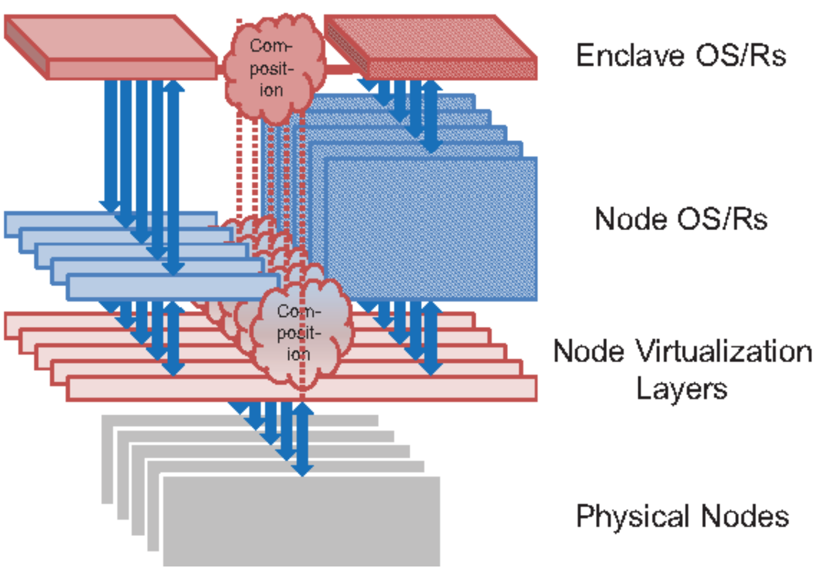
A Deeper Look at Composition

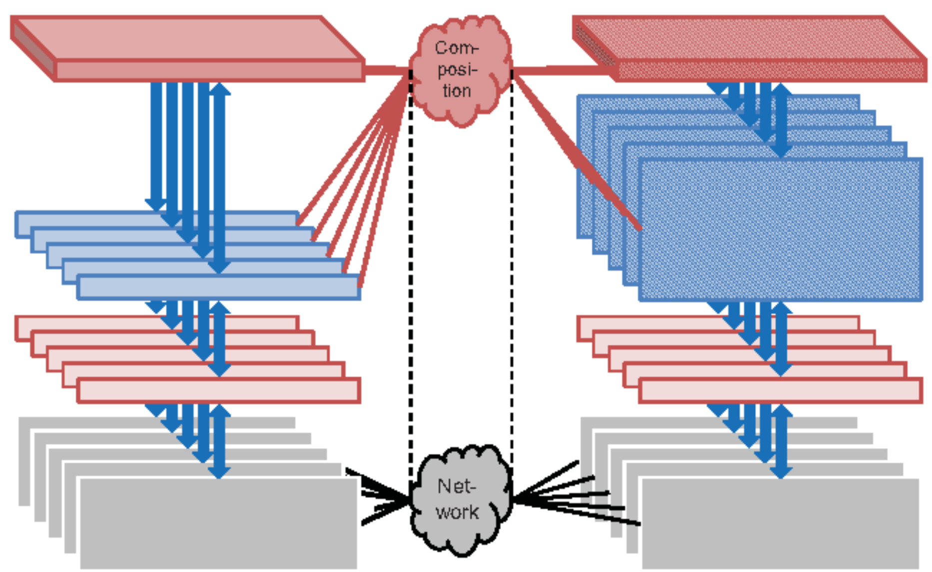
Intra-Node Composition

- Components co-located on same set of nodes
- Virtualization used to isolate NOS environments on each node
- Composition (coupling) takes place via shared memory

Inter-Node Composition

- Components deployed to separate sets of nodes
- Composition (coupling) takes place via network





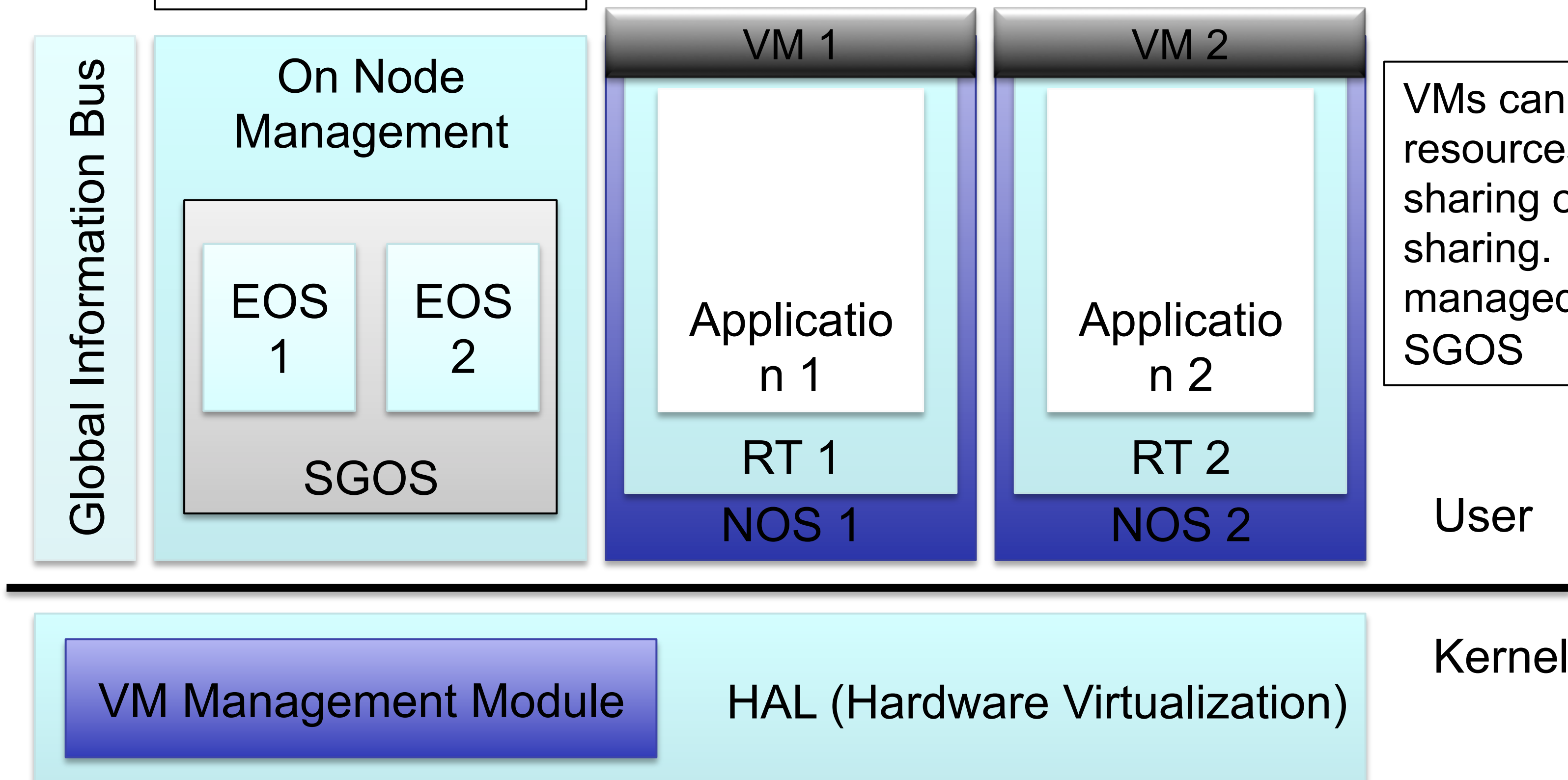
Composition of Enclaves

- An enclave provides a single OS/R environment to the application
- Hobbes approach is to provide the minimum “amount” of OS/R required by the application (do what is necessary and get out of the way!)
- Modern, complex applications are increasingly created by assembling (often substantial) software components
  - E.g., analytics connected to applications, code coupling, application frameworks, …
- Components may have distinct requirements for OS/R support
- Two options:
- Assemble an all-in-one OS/R stack that satisfies all component needs
  - Potential challenges at both OS and RTS levels
  - Requires integration work for every combination supported
- Provide each component the OS/R it needs, and provide efficient, low-level mechanisms to connect the components (and the OS/Rs)

Hobbes Node Architecture

Independent Operating and Runtime Systems

Policies to manage the VMs on a single node.



VMs can share the resources via time sharing or space sharing. This is managed by the SGOS

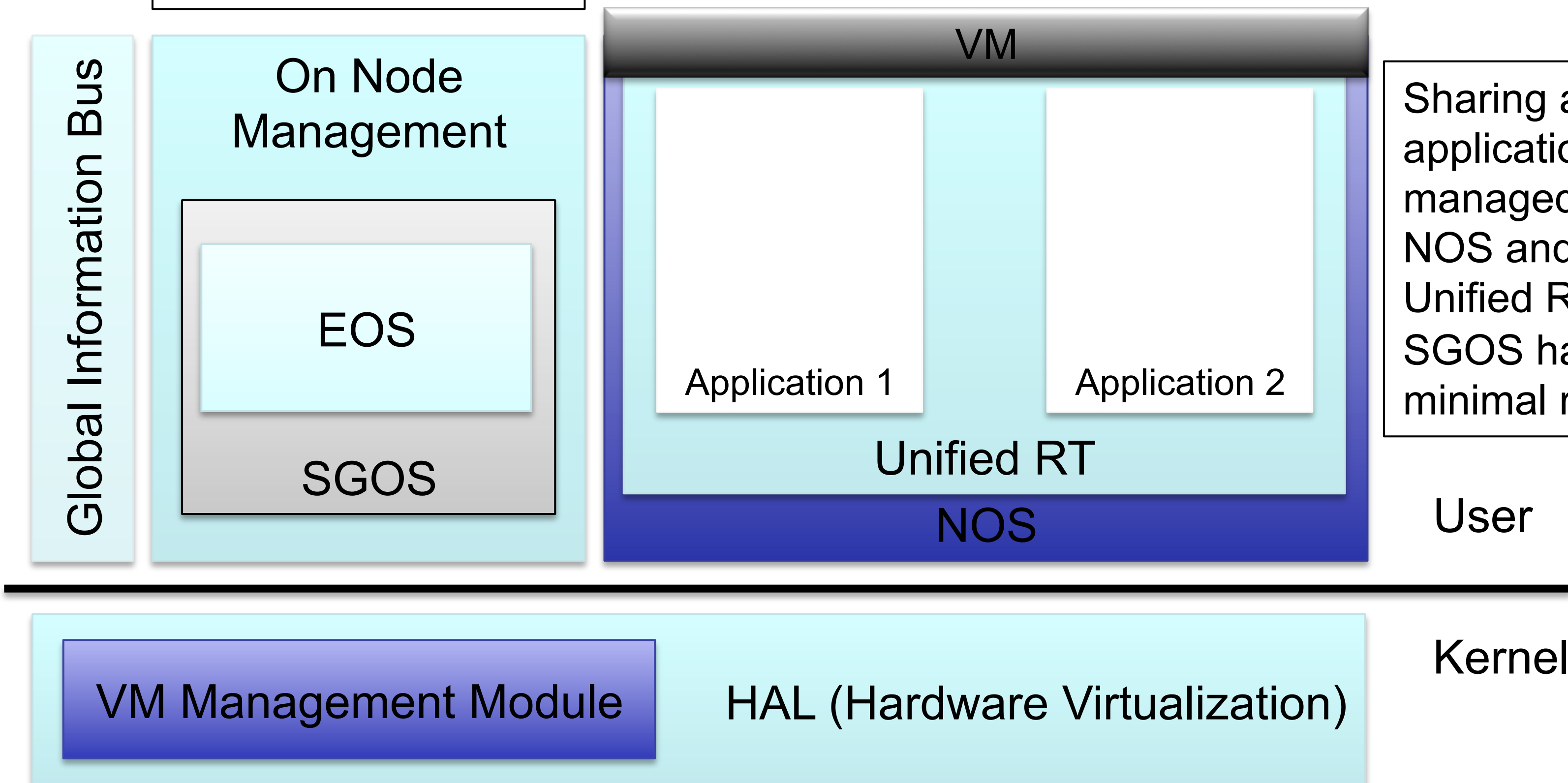
Additional mechanisms needed to manage multiple VMs. Run in kernel mode to take advantage of VM support in modern processors (Palacios).

Basic mechanisms needed to virtualize hardware resources like address spaces (Kitten).

Hobbes Node Architecture

Unified Operating and Runtime Systems

Policies to manage the VMs on a single node.

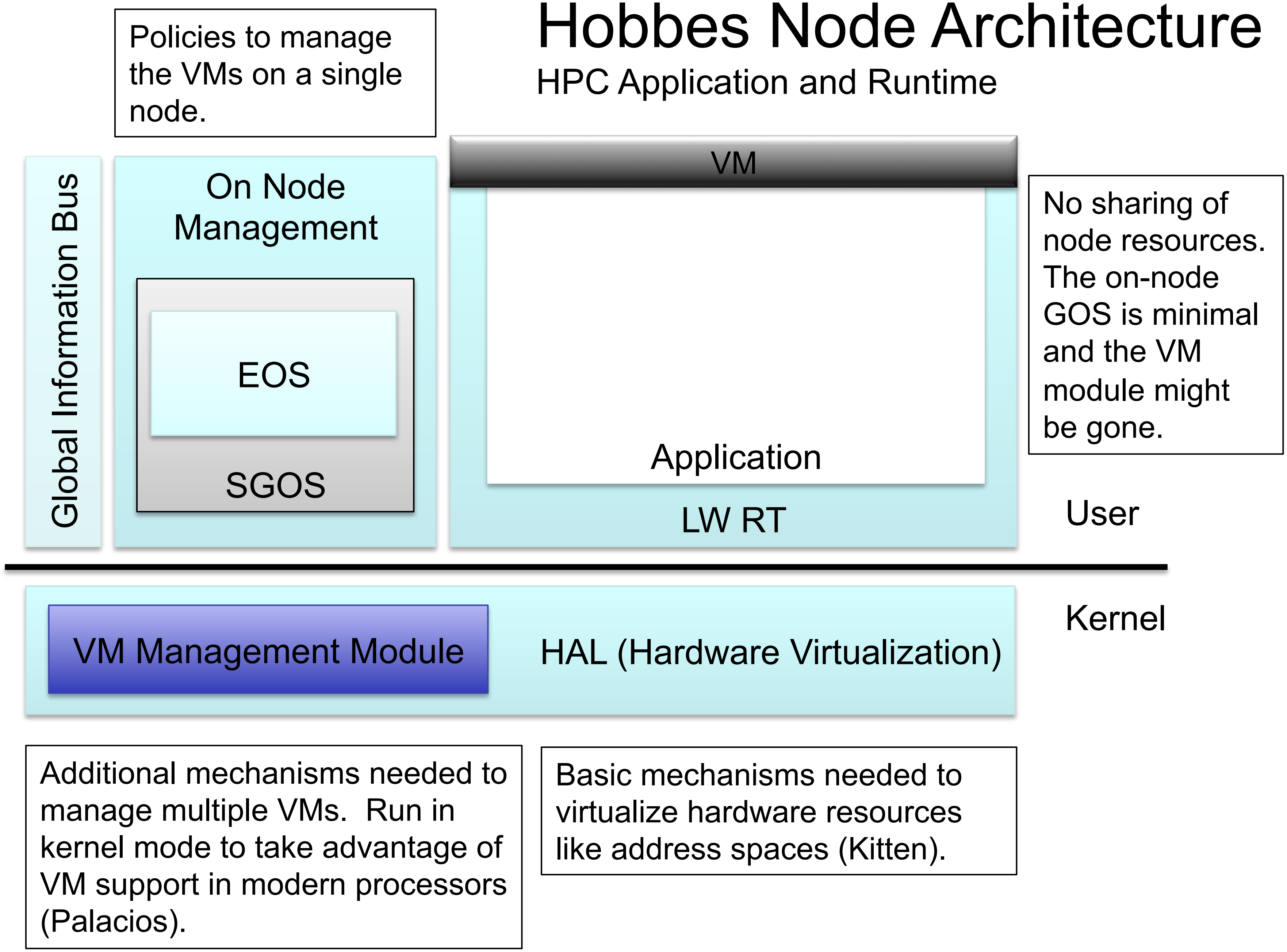


Sharing among applications is managed by the NOS and Unified RT. The SGOS has a minimal role.

Additional mechanisms needed to manage multiple VMs. Run in kernel mode to take advantage of VM support in modern processors (Palacios).

Basic mechanisms needed to virtualize hardware resources like address spaces (Kitten).

Hobbes Node Architecture



Composition Examples

- SNAP + Analytics
  - "SNAP calculates synonymous and non-synonymous substitution rates based on a set of codon-aligned nucleotide sequences." (HIV related)
  - Proxy app from LANL used for example
- GTC-P + Analytics
  - Fusion simulation testing/proxy app used to test new hardware and algorithm integration into the PIC model. (PPPL)
  - Analytics generate statistics on particles (histograms), sorts, and filters on bounding boxes
- LAMMPS + Analytics
  - Full, production molecular dynamics application from Sandia
  - Analytics look for crack formation by calculating atomic spacing in output data to change simulation from coarse to fine grained.

Hobbes Has Several Components

- Node Virtualization Layer
- Enclave OS
- Scheduling
- Programming Models
- Global Information Bus
- Resilience
- Power/Energy

Hobbes Team

Institution	Person	Role
Georgia Institute of Technology	Karsten Schwan	PI
Indiana University	Thomas Sterling	PI
Los Alamos National Lab	Mike Lang	PI
Lawrence Berkeley National Lab	Costin Iancu	PI
North Carolina State University	Frank Mueller	PI
Northwestern University	Peter Dinda	PI
Oak Ridge National Laboratory	David Bernholdt	PI
Oak Ridge National Laboratory	Arthur B. Maccabe	Chief Scientist
Sandia National Laboratories	Ron Brightwell	Coordinating PI
University of Arizona	David Lowenthal	PI
University of California – Berkeley	Eric Brewer	PI
University of New Mexico	Patrick Bridges	PI
University of Pittsburgh	Jack Lange	PI

Node Virtualization Layer (NVL)

Why a Node Virtualization Layer?

- Flexibility, support multiple OS/R stacks simultaneously
  - There is likely to be no one-size-fits-all OS/R stack, lots of exploration
  - Co-location of VMs, efficient sharing of resources between enclaves
  - Native environment freed from legacy constraints
- Low overhead
  - Our past work has shown CPU and memory overheads negligible
  - Network I/O is still an issue, but tractable
- Industry momentum
  - Virtualization has been commoditized, is everywhere
  - Academic and student mindshare, where the jobs are
- Mostly orthogonal to "FusedOS" approach others are taking
  - FusedOS could run in NVL VM or natively, in the same machine
  - NVL could be co-designed with FusedOS
- Already doing node OS R&D under XPRESS

Exploring Spectrum of Virtualization

- Virtualization doesn't have to be "big and heavy"
  - Don't have to trap everything
  - VMM can setup paths to hardware, then get out of way
- There are multiple virtualization architectures, not just one
  - Hobbes NVL team working across spectrum (Blue items, research in Light Blue)

Lightest Weight ⟷ **Heaviest Weight**

LWK
Virtual Linux Evironment
(**Kitten**, CNK)

Combo OS
Multiple-native OSes
(**Pisces**, FusedOS)

Full HW VM
Runs Unmodified Guest OSes, Passthru
(**Palacios**, KVM, …)

LWK
Custom
(Catamount, **HybridVM**)

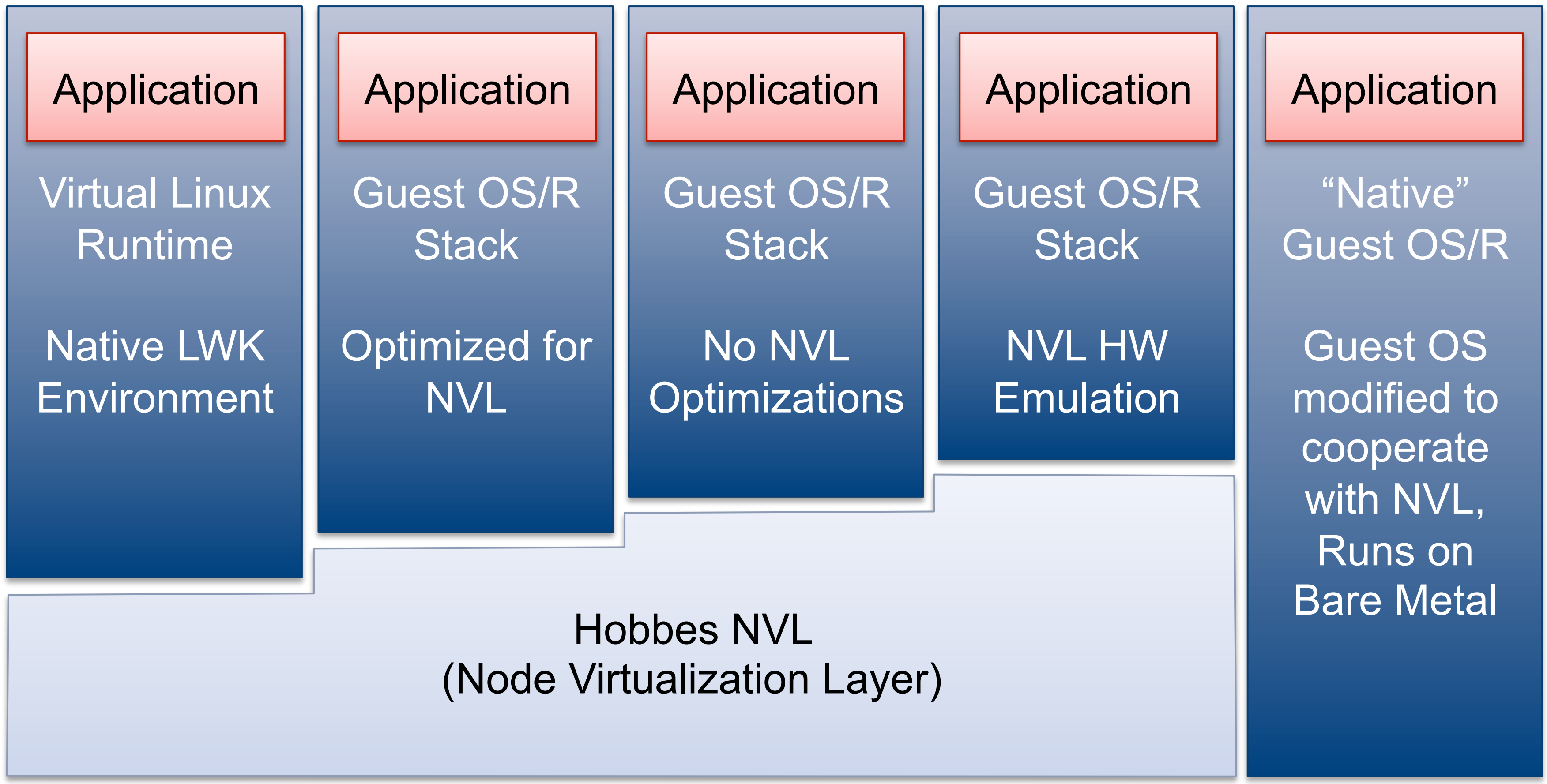
Para-virtual
Implicit, VMM Changes Guest OS
(**Gears, Guarded Modules**)

Para-virtual
Explicit, Guest OS Modified or Augmented
(Orig. Xen, **Device Drivers**)

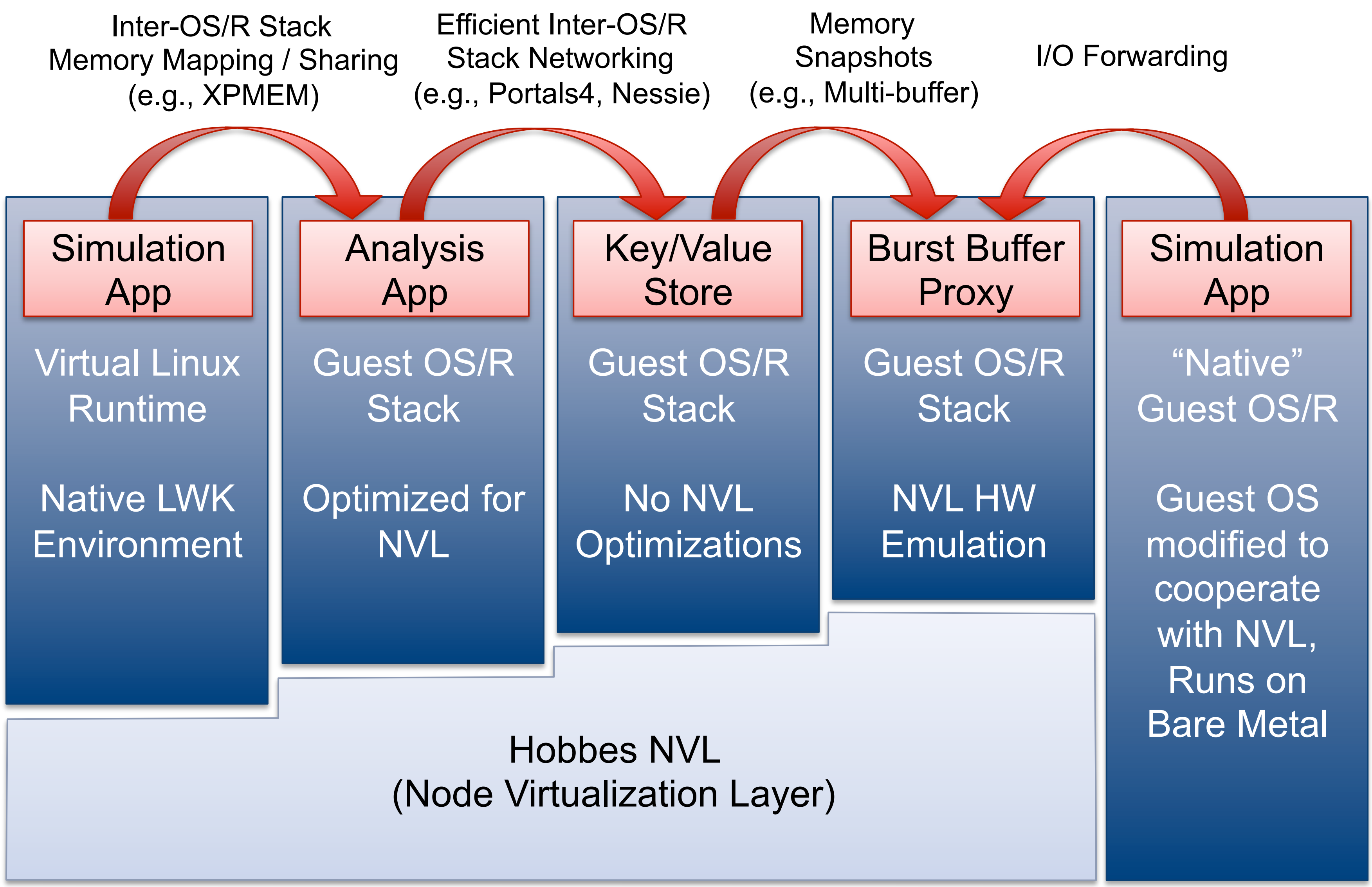
Software Virt
Emulate HW, Binary Translation, …
(Qemu, Vmware, **Emulate HW Trans Memory pre-product**)

NVL Has Multiple Levels of Virtualization

- Existing Hypervisors typically support one level
- NVL couples LWK "native" runtime with guest OS/R stacks

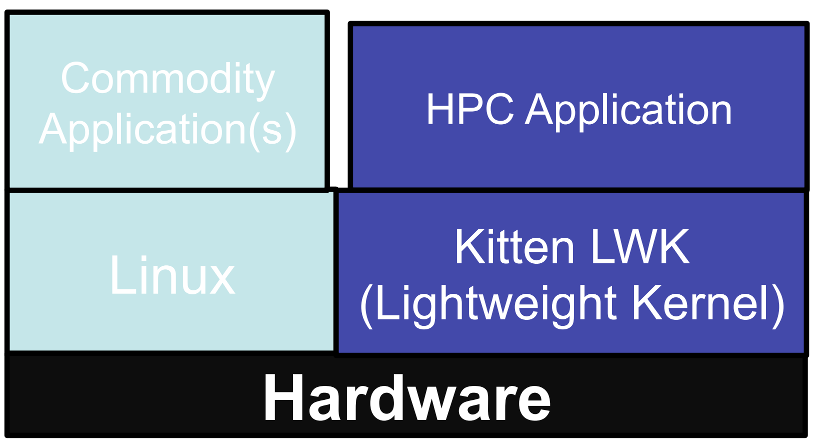


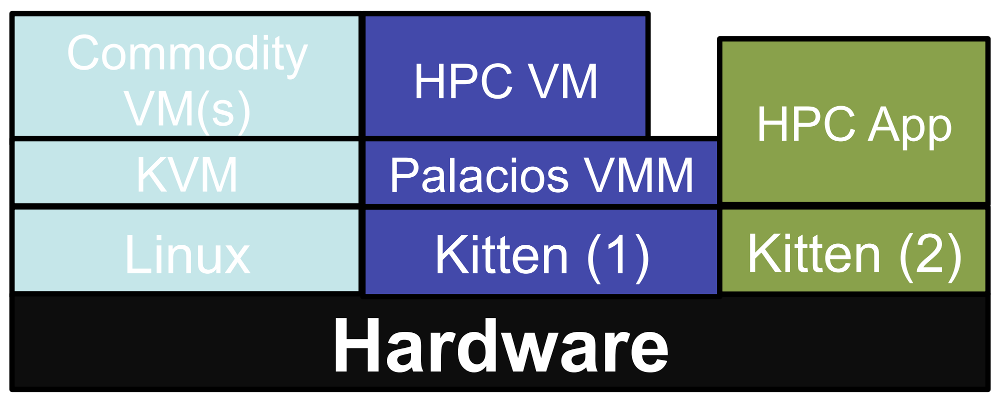
NVL Provides Composition Mechanisms

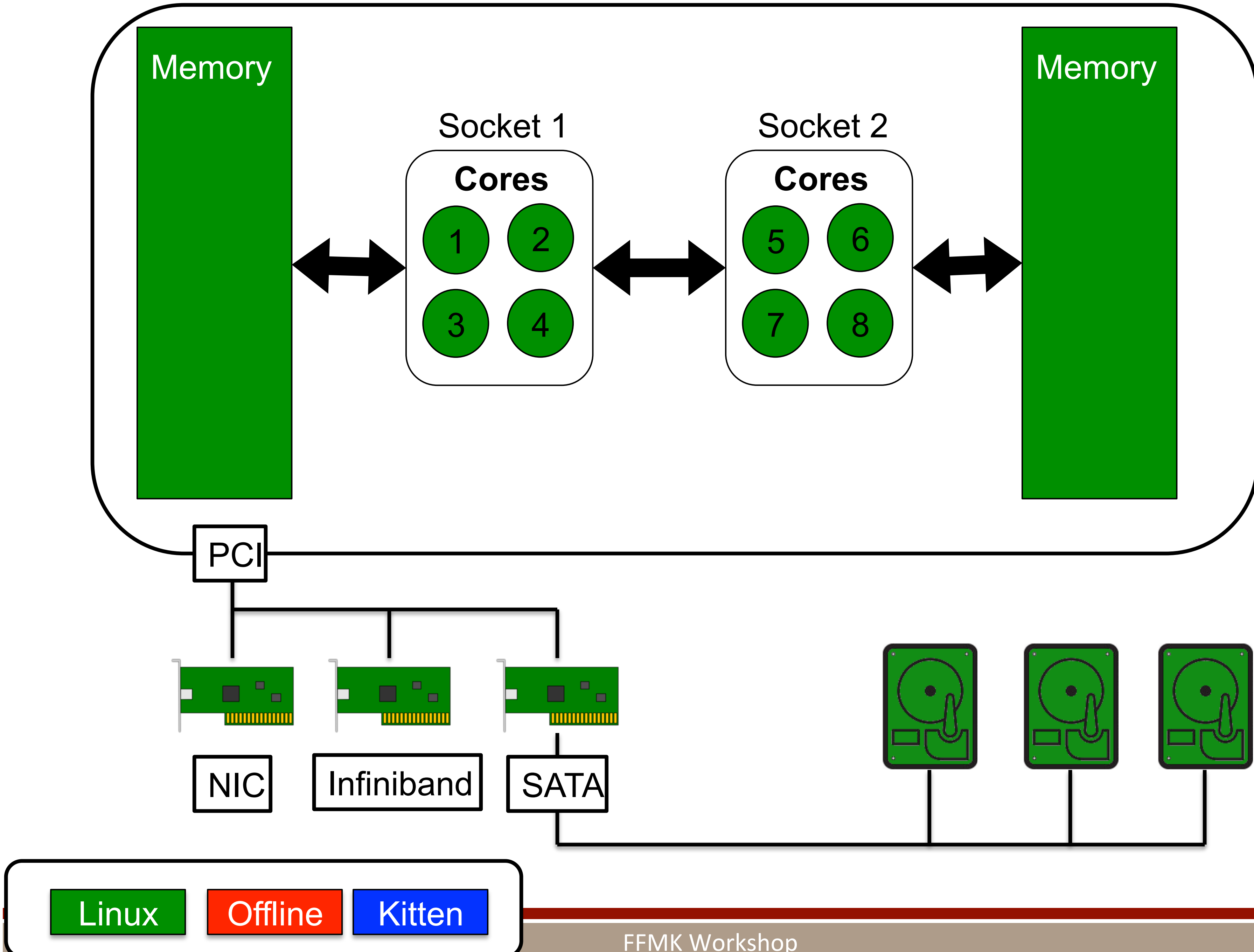


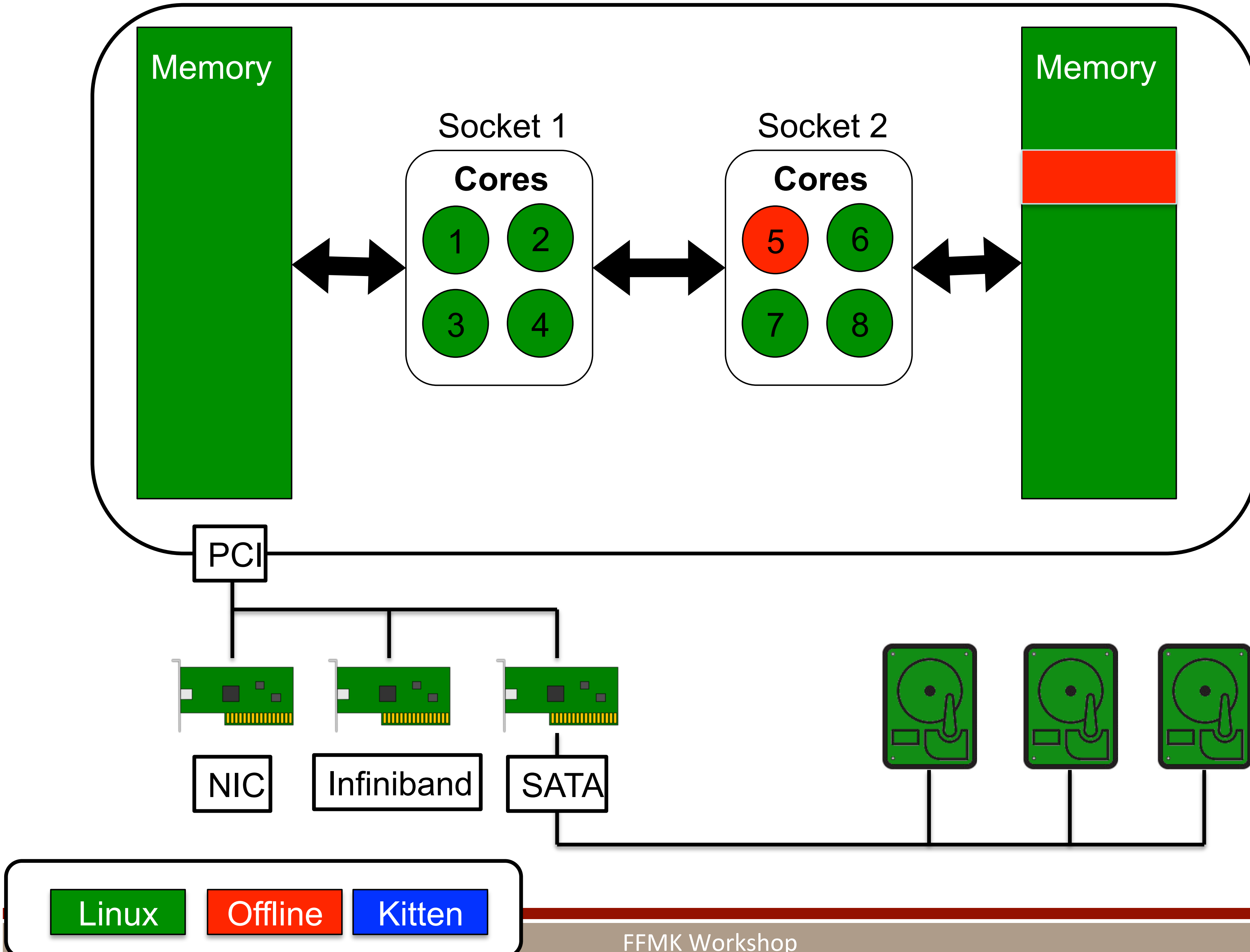
Pisces Multi-stack Architecture is Our Starting Point

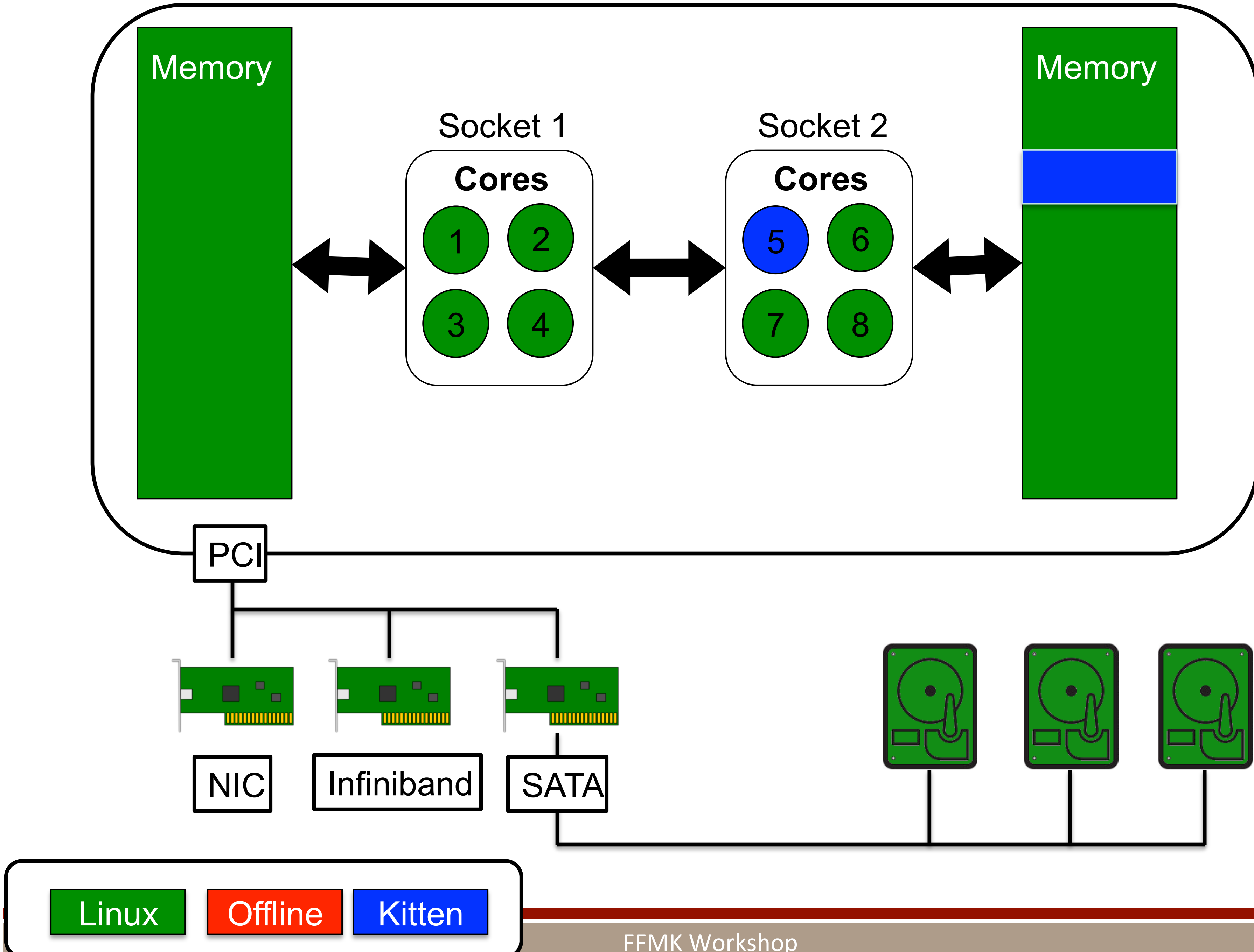
- Goals
  - Fully isolated and independent operation
  - OS Bypass communication
  - No cross-kernel dependencies
  - Leverage cloud platforms
- Uses Linux hot un/plug to bring cores, memory, and devices offline
- Recent modifications to Kitten
  - Boot process that initializes only subset of offline resources
  - Dynamic resource (re)assignment to Kitten
  - Cross stack shared memory communication (XPMEM)
  - Block Driver Interface

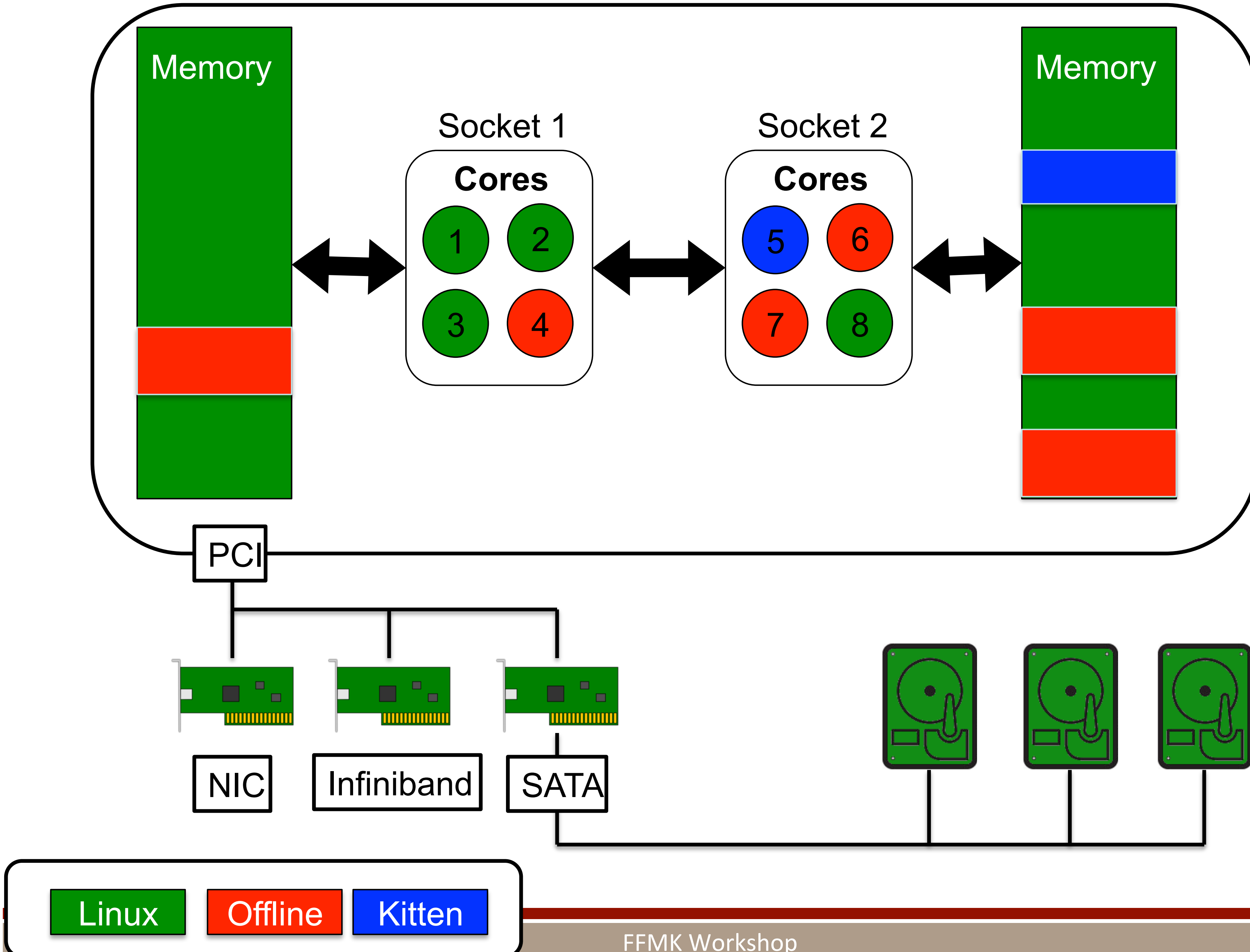


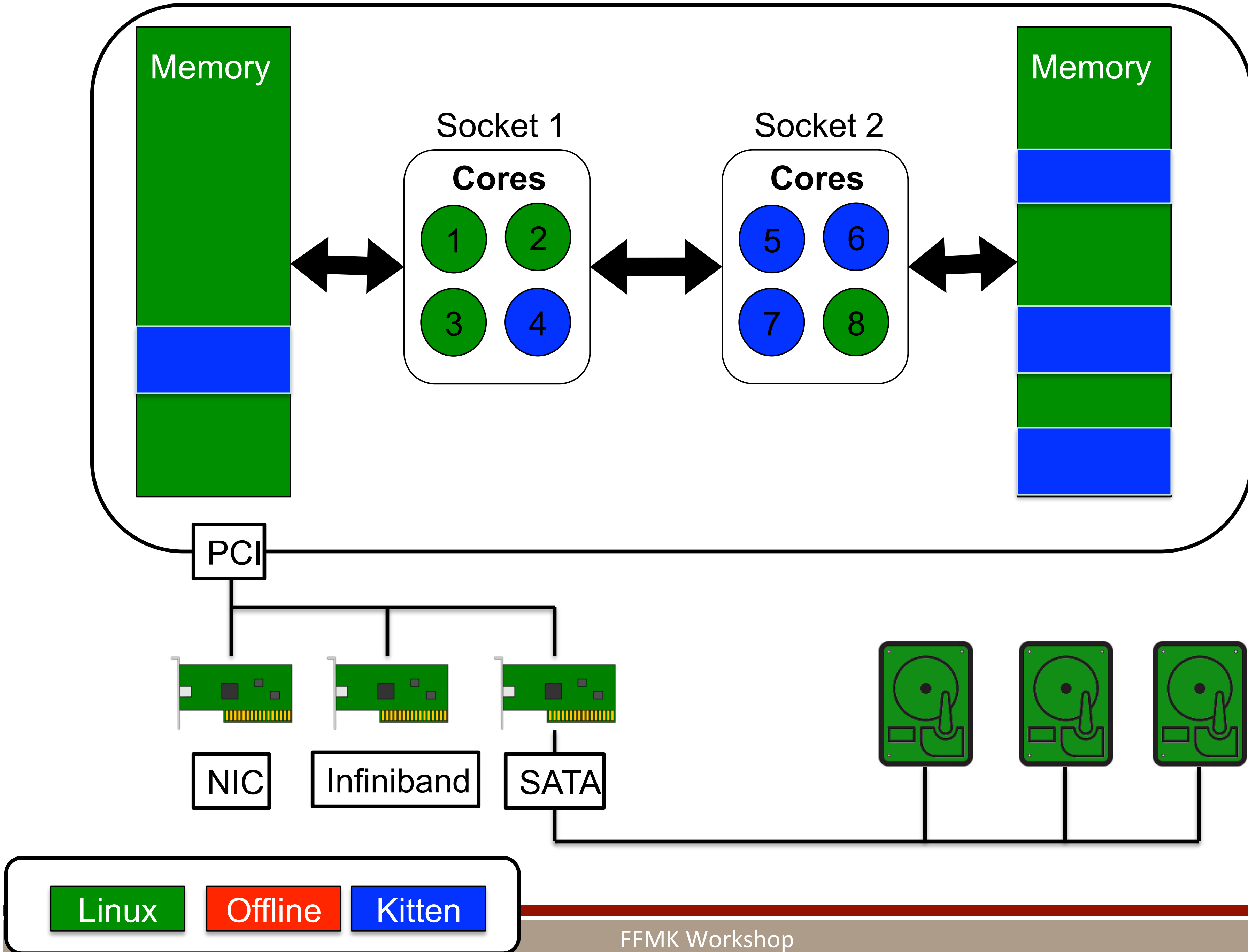


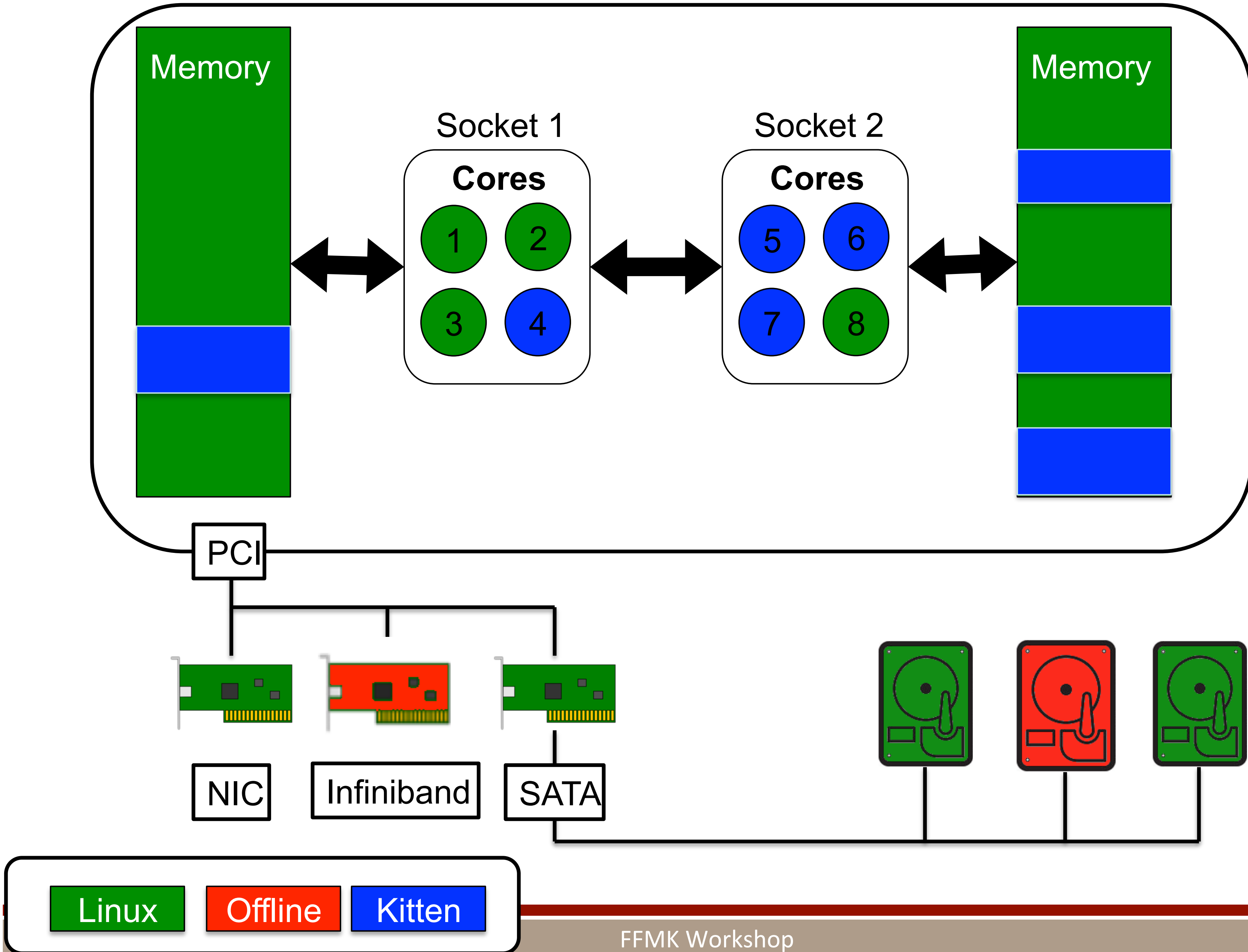

Memory
Socket 1
Cores
1
2
3
4
Socket 2
Cores
5
6
7
8
Memory
PCI
NIC
Infiniband
SATA
Linux
Offline
Kitten

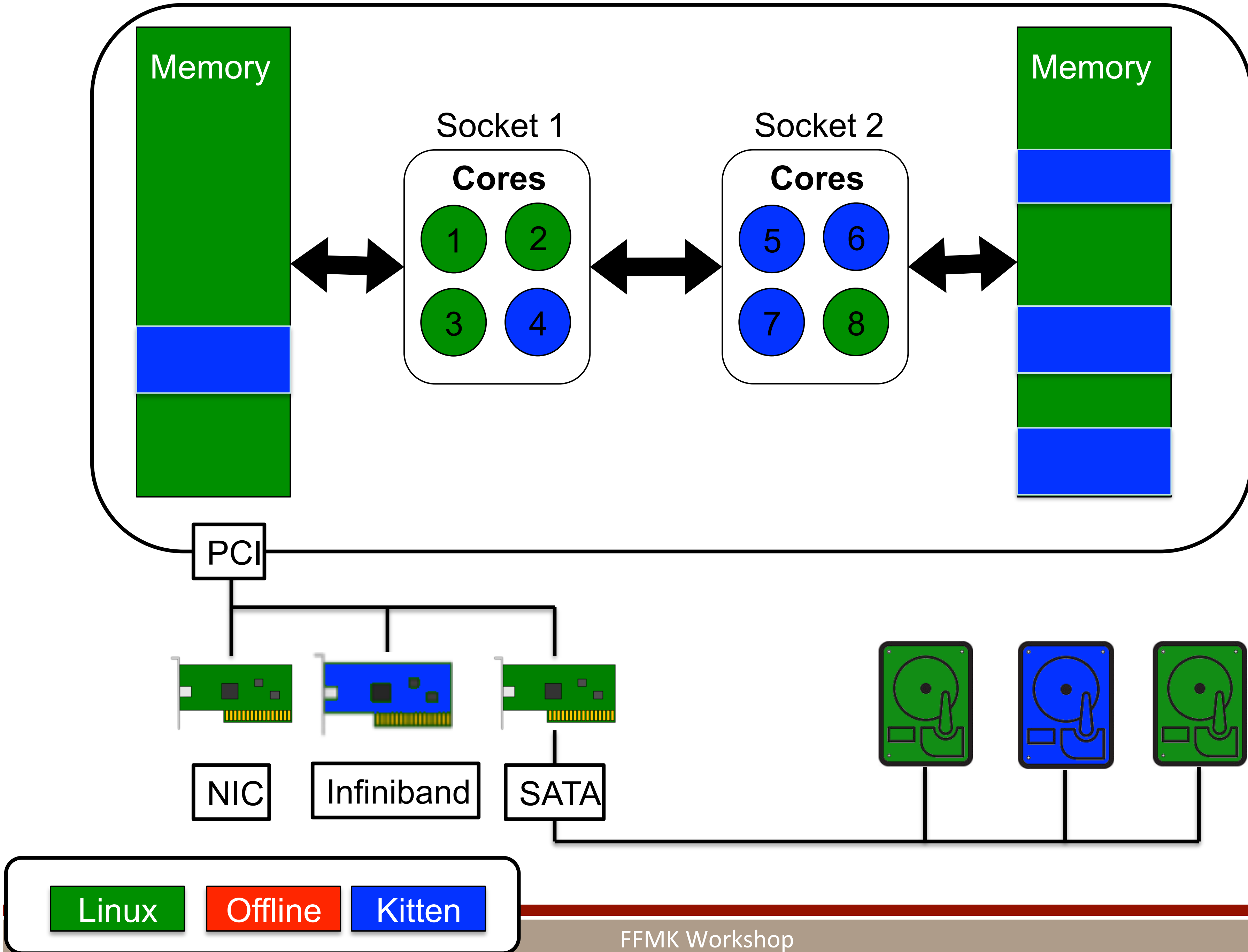

Memory
Socket 1
Cores
1
2
3
4
Socket 2
Cores
5
6
7
8
Memory
PCI
NIC
Infiniband
SATA
Linux
Offline
Kitten

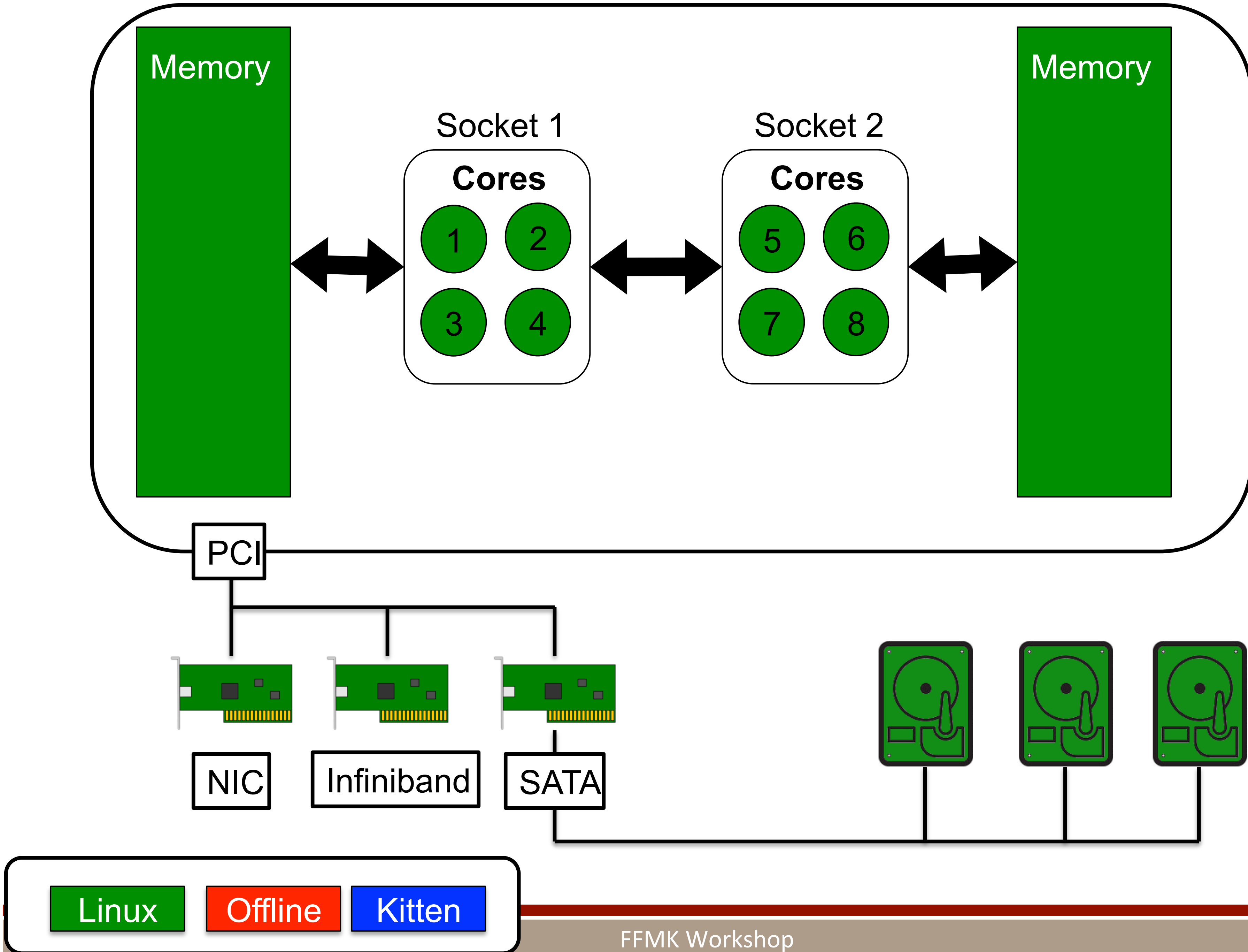

Memory
Socket 1
Cores
1
2
3
4
Socket 2
Cores
5
6
7
8
Memory
PCI
NIC
Infiniband
SATA
Linux
Offline
Kitten


Memory
Socket 1
Cores
1
2
3
4
Socket 2
Cores
5
6
7
8
Memory
PCI
NIC
Infiniband
SATA
Linux
Offline
Kitten


Memory
Socket 1
Cores
1
2
3
4
Socket 2
Cores
5
6
7
8
Memory
PCI
NIC
Infiniband
SATA
Linux
Offline
Kitten

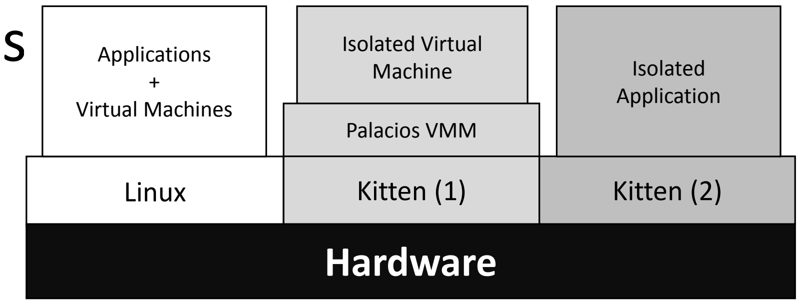

Memory
Socket 1
Cores
1
2
3
4
Socket 2
Cores
5
6
7
8
Memory
PCI
NIC
Infiniband
SATA
Linux
Offline
Kitten


Memory
Socket 1
Cores
1
2
3
4
Socket 2
Cores
5
6
7
8
Memory
PCI
NIC
Infiniband
SATA
Linux
Offline
Kitten


Memory
Socket 1
Cores
1
2
3
4
Socket 2
Cores
5
6
7
8
Memory
PCI
NIC
Infiniband
SATA
Linux
Offline
Kitten

Node Virtualization Layer Status

- Pisces multi-stack architecture tools implemented and functional
  - Host boots Linux
  - Cores and memory can be taken from Linux, forming one or more containers
  - Kitten can be launched in each container
  - Each Kitten instance operates cooperatively with Linux as a co-kernel
  - Each co-kernel can run a different application
    - Or guest OS via Palacios
  - Containers can be dynamically resized without rebooting
    - Number of cores and size of memory can grow and shrink
- Ported to Cray Linux Environment
- Multi-enclave launch working on Cray XK7 testbed at Sandia



Co-Kernel Architecture

NVL Status (continued)

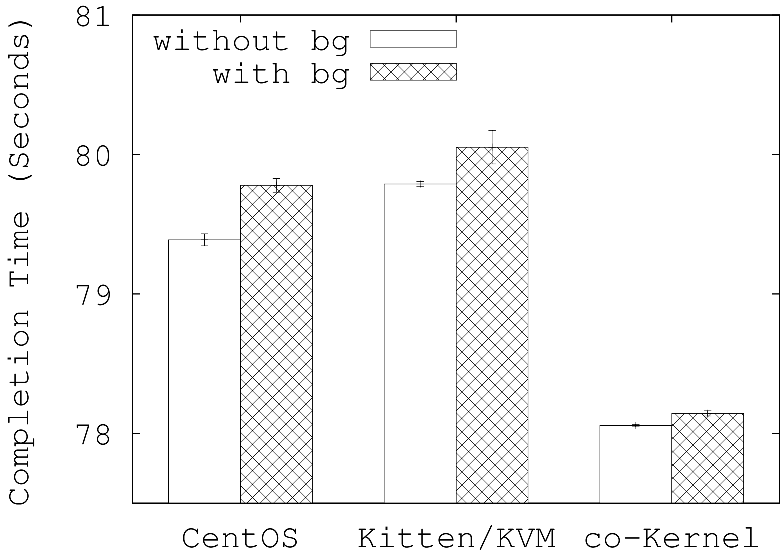
- Using XPMEM for inter-OS shared memory
  - XPMEM allows address space sharing between distinct processes
  - The OS running on an NVL instance can export and attach to memory regions from co-kernels running on the same NVL
  - All combinations working: Linux <->Linux, Linux<->Kitten, Kitten<->Kitten (where Linux is either Host Linux or Guest Linux)
- Transparently Consistent Asynchronous Shared Memory (TCASM)
  - Allows for asynchronously exporting a snapshot of a memory region to many observers
  - Completed initial port from Linux to Kitten

Pisces Single-Node Performance Evaluation

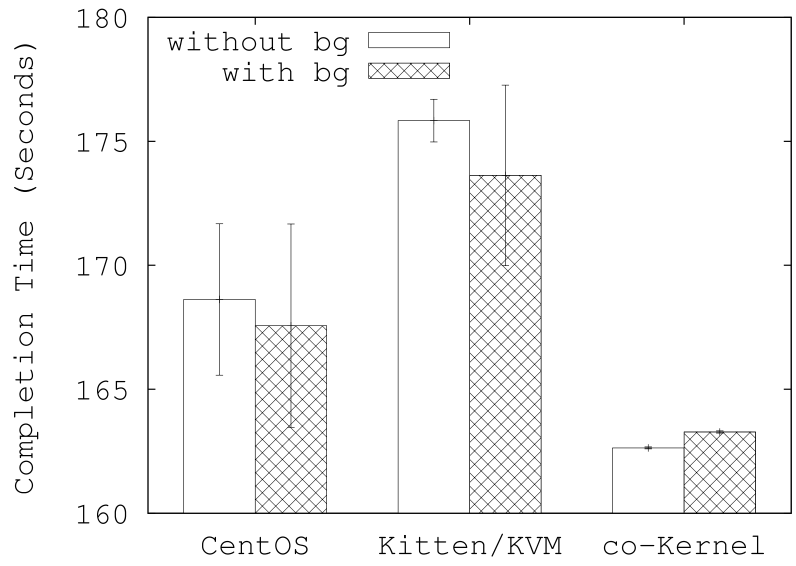
- Dell R450 server
  - Two six-core Intel IvyBridge Xeons (12 cores total)
  - 24 GB RMA in two NUMA domains
- CentOS 7 Linux Distribution
  - Linux kernel v3.16
- Benchmarks
  - miniFE from Mantevo mini-app suite
  - HPC Challenge RandomAccess Benchmark
- OS environments
  - CentOS – CentOS Linux on all 12 cores
  - Kitten/KVM – CentOS Linux on 6 cores, Kitten KVM guest on 6 cores
  - Co-Kernel – CentOS Linux on 6 cores, Kitten co-kernel on 6 cores
- Execution environment
  - Without background noise
    - Benchmark executed alone in one NUMA domain
  - With background noise
    - Benchmark executed in one NUMA domain, kernel compile in one NUMA domain

Co-Kernel Has Better Performance and Performance Isolation

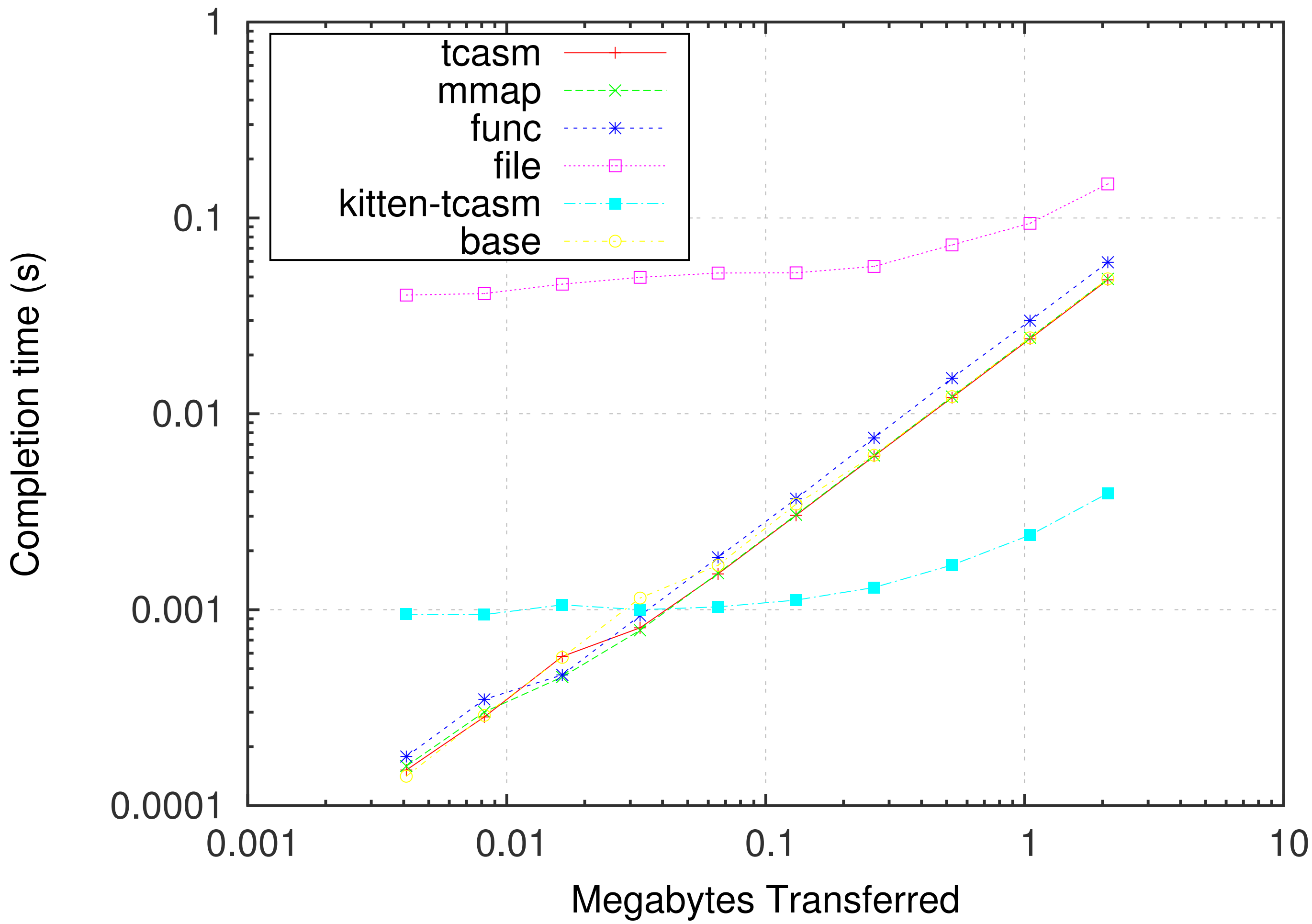
miniFE



RandomAccess



TCASM Performance



Kitten TCASM implementation showing good performance and scalability

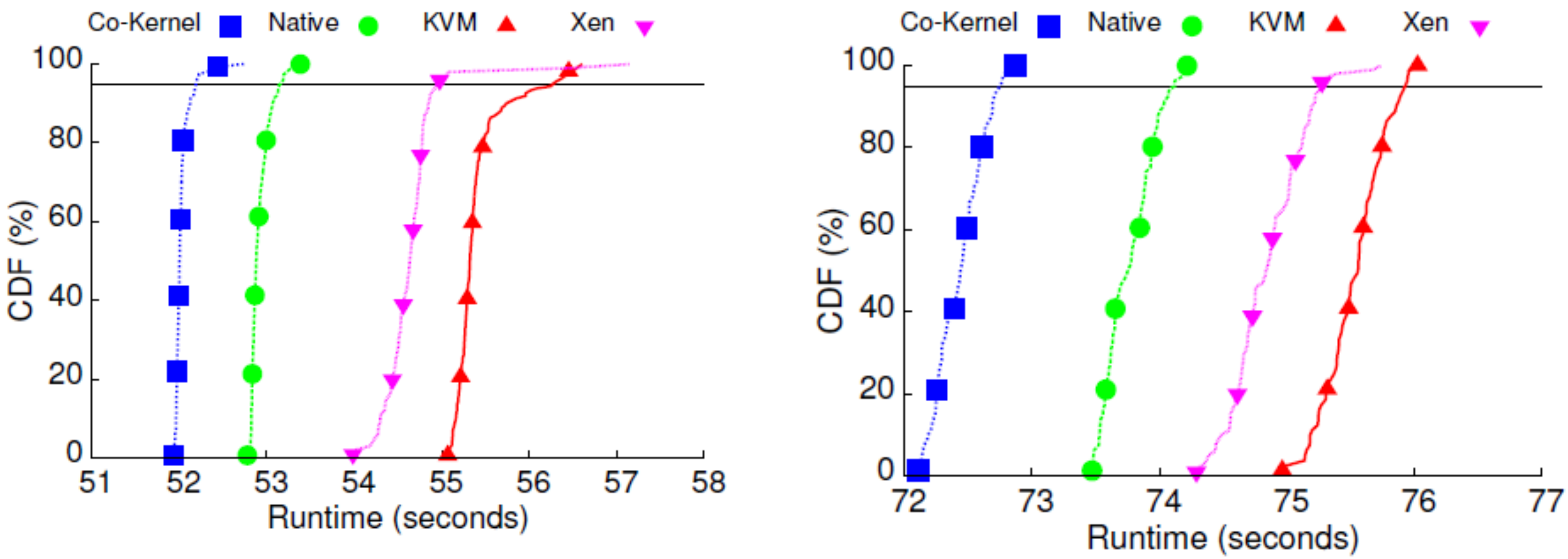
Results are for producer/consumer microbenchmark, both processes running in Kitten

SNAP running in Kitten coupled with analytics running in Linux is under test

Pisces Multi-Node Performance Evaluation

- 8 node InfiniBand cluster
- Each node has 16 cores, 24 GB of memory, and two HDDs
- Hot unplugged 8 cores, 12 GB (one socket), and one HDD from Linux on each node
- Used Pisces to launch Kitten/Palacios on 8 nodes
  - Kitten SATA driver manages one HDD on each node for local disk I/O for Palacios VMs
- Each node runs
  - A cloud-serving Ubuntu guest on a KVM host
    - Runs Hadoop data nodes with a machine learning benchmark (Mahout)
    - Bridged to a 1 GigE NIC connect to GigE switch
  - An HPC-serving Fedora 19 guest running on either same KVM host or on Palacios/ Kitten
    - HPCCG and Cloverleaf mini-apps
    - Connected to passthrough IB network
- Measured the cumulative distribution function on a few hours (300+ runs) of HPCCG running on both KVM and Palacios/Kitten while simultaneously running Hadoop on the same node

Co-Kernel Has Better Performance and Performance Isolation



(a) HPCCG

(b) CloverLeaf

NVL Plan

- ADIOS working over XPMEM
  - ADIOS provides higher-level interface for coupling than using XPMEM directly
  - Application composition use cases initially targeting ADIOS
- Develop NVL name service
  - Will allow NVL OS instances running on the same node to post and query available resources that can be shared for composition
  - Eventually may become part of the Global Information Bus
- TCASM
  - Need to better understand initial performance evaluation
  - Implement inter-OS TCASM, integrate with XPMEM
  - TCASM will be a new type of XPMEM memory region
  - Will compare, contrast, and evaluate application composition scenarios with ADIOS, XPMEM, and TCASM

Source Code Availability

- Git Source Code Repository:
  - git clone https://software.sandia.gov/git/nvl
  - See README for build instructions
- Build Appliance
  - Includes software dependencies needed for NVL development
  - Includes NVL checkout and pre-built NVL images
  - http://software-login.sandia.gov/~ktpedre/hobbes/hobbes_build_appliance.vmwarevm.tar.gz

XPRESS: LXK/RIOS Research Goals

- XPRESS aims to increase synergy of compute node OS kernel and user-level runtime systems
  - Today: Runtime must work around host OS, assume worst case
  - Vision: Runtime cooperates with host OS, delegated more control
- Key RIOS drivers (Runtime Interface to the OS)
  - Runtime needs guarantees about resource ownership and behavior
  - OS needs way to shift resources between multiple runtimes
  - Two-way interfaces needed for key resources
    - Runtime tells OS what it needs, OS tells runtime what it gets
    - OS remembers original request, notifies runtime if more resources become available. Notifies runtime of resources need to be reclaimed.
  - Event-based protocol to notify of dynamic events (e.g., power state change, transient error)
- LXK = Kitten + RIOS

XPRESS: Areas Covered by RIOS

- Legacy support services
- Job management
- Memory management
- Thread management
- Network interface
- System topology and locality
- Introspection
- File I/O
- Power management

SANDIA REPORT

SAND2013-XXXX
Unlimited Release
Printed September 2013

Runtime Interface to the Operating System (RIOS) Specification

Brian Barrett, Ron Brightwell, Stephen Olivier, Kevin Pedretti, Dylan Stark, and Thomas Sterling

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Bull eXascale Interconnect (BXI)

- New, next-generation interconnect fabric
- Hardware-based network interface offload
- Several virtual networks for QoS
- Adaptive routing
- End-to-End and link-level retry for reliability
- Based on Portals 4 communication library

Getting rid of the communications overhead

A new generation of interconnect is required

Exascale entails an explosion of performance, data volumes, power consumption and data movement. So it will be increasingly critical to ensure that CPUs are fully dedicated to computation. Today, with current interconnects, CPUs are also responsible for communications, at the expense of performance. Getting rid of this overhead would immediately and significantly free up CPU performance.

As a result, one of the cornerstones of Bull's exascale program is the development of a new-generation interconnect.

The Bull exascale Interconnect unleashes CPU performance

Bull is developing a new-generation interconnect, the Bull exascale Interconnect: code-named BXI. BXI represents a paradigm shift when it comes to boosting performance, scalability, efficiency, reliability and quality of service for extreme workloads.

The core feature of BXI is a full hardware-encoded communication management system, which enables CPUs to be dedicated to computational tasks while communications are independently managed by BXI.

BXI hardware primitives map directly to communication libraries such as MPI (Message Passing Interface) and PGAS (Partitioned Global Address Space). Thanks to this hardware acceleration, BXI will deliver the highest level of communication performance for HPC applications, at full scale, characterized by high bandwidth, low latency and high message rates.

The BXI architecture is based on the Portals 4 communication library. This enables full optimization for all MPI communication types, including the latest MPI-2 and MPI-3 extensions and PGAS. The Portals 4 non-connected protocol guarantees a minimum constant memory footprint, irrespective of system size.

BXI quality of service (QoS) enables the definition of several virtual networks and will ensure, for example, that bulky I/O messages do not impede small data message flow. In addition, BXI adaptive routing capabilities will dynamically avoid communication bottlenecks.

End-to-end error checking and link level retry have been implemented to enhance communication reliability and resilience without jeopardizing communication performance.

8 | Bull exascale program – Statement of Direction

http://xstack.sandia.gov/hobbes
http://xstack.sandia.gov/xpress