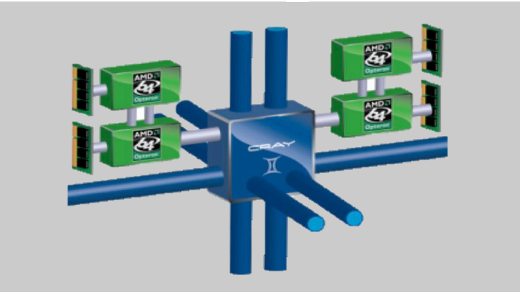
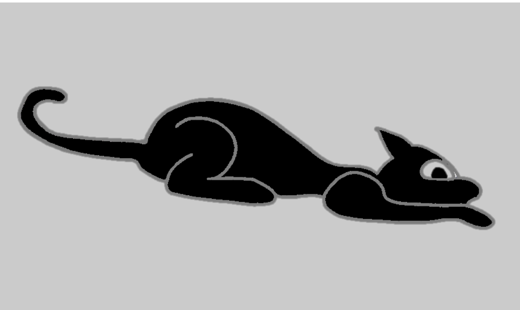
SAND2012-7203P



OS Issues for HPC: Past, Present, Future



Kevin Pedretti
Scalable System Software
Sandia National Laboratories
Albuquerque, NM
ktpedre@sandia.gov

Sandia National Laboratories

Exceptional service in the national interest

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. SAND NO. 2011-XXXXP

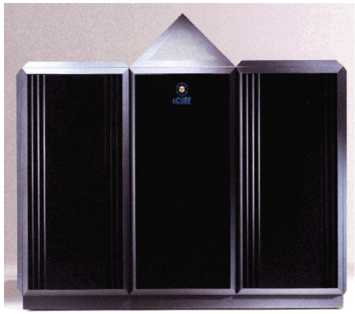
Outline

- Past / Background
  - Lightweight Kernel (LWK) as an optimization layer
- Present
  - Kitten LWK
- Future
  - LWK as tool for runtime <-> OS <-> HW co-design
- Closing thoughts

Sandia Massively Parallel Systems

Sandia National Laboratories

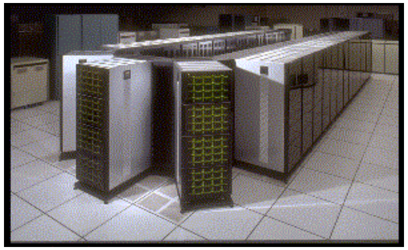
1990



nCUBE2

- Sandia's first large MPP
- Achieved Gflops performance on applications

1993



Paragon

- Tens of users
- First periods processing MPP
- World record performance
- Routine 3D simulations
- SUNMOS lightweight kernel

1997

ASCI Red

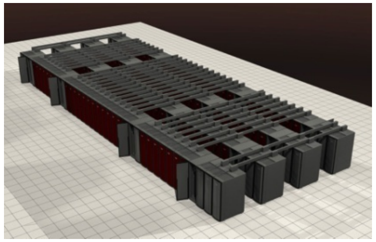
- Production MPP
- Hundreds of users
- Red & Black partitions
- Improved interconnect
- High-fidelity coupled 3-D physics
- Puma/Cougar lightweight kernel

1999

Cplant

- Commodity-based supercomputer
- Hundreds of users
- Enhanced simulation capacity
- Linux-based OS licensed for commercialization
- ~2000 nodes

2004



Red Storm

- Prototype Cray XT
- Custom interconnect
- Purpose built RAS
- Highly balanced and scalable
- Catamount lightweight kernel

Sandia Lightweight Kernel Targets

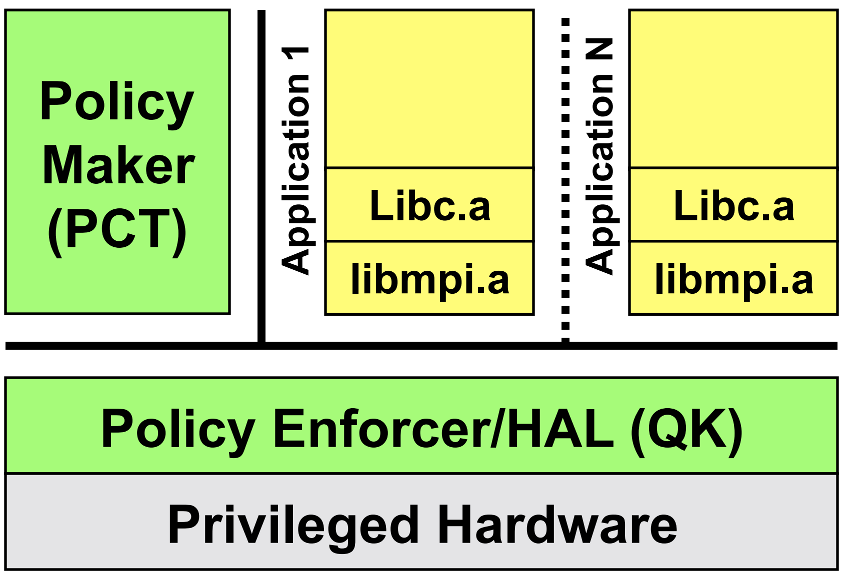
- Massively-parallel, distributed-memory machine with a tightly-coupled network
- Scientific and engineering modeling and simulation applications
- Enable fast message passing and execution
- Small memory footprint
- Deterministic performance
- Emphasize efficiency over functionality
- Maximize performance delivered to application

Reasons for a Specialized Approach

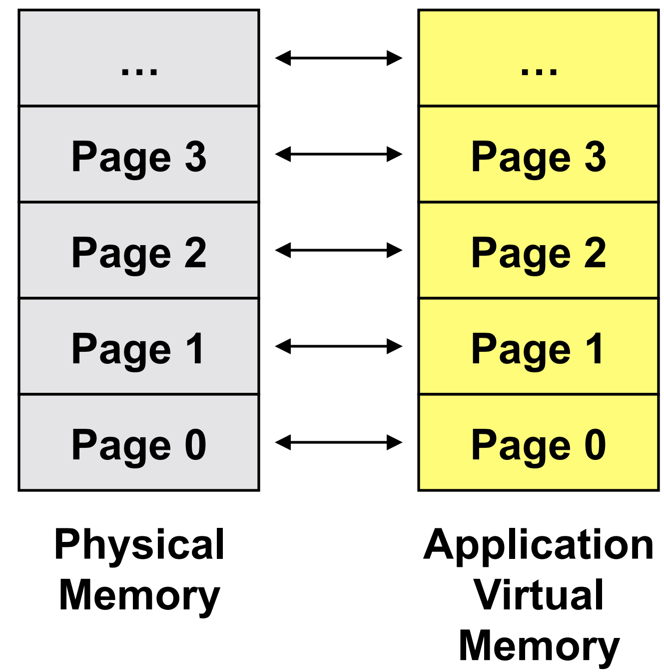
- Maximize available compute node resources
  - Maximize CPU cycles delivered to application
    - Minimize time taken away from application process
    - No daemons
    - No paging
    - Deterministic performance
  - Maximize memory given to application
    - Minimize amount of memory used for message passing
    - Static kernel size
  - Maximize memory bandwidth
    - Use large pages to avoid TLB misses, speed TLB miss handling
  - Maximize network resources
    - Physically contiguous memory layout
    - Simple address translation and validation, no pinning
- Increase reliability
  - Relatively small amount of source code
  - Reduced complexity
  - Support for small number of devices

LWK Overview

Basic Architecture

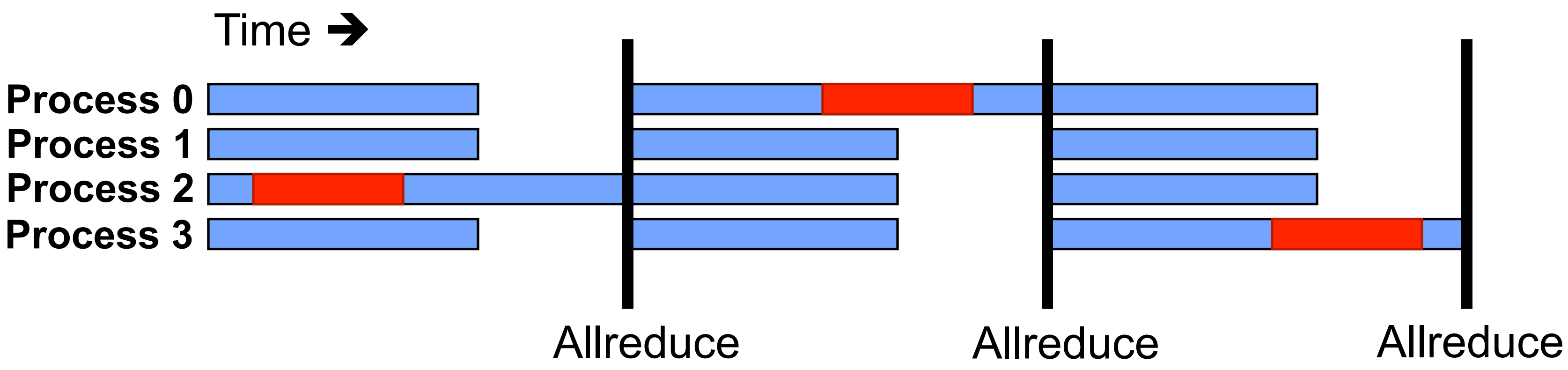


Memory Management



- POSIX-like environment
- Inverted resource management
- Low noise OS noise/jitter
- Straight-forward network stack (e.g., no pinning)
- Simplicity leads to reliability

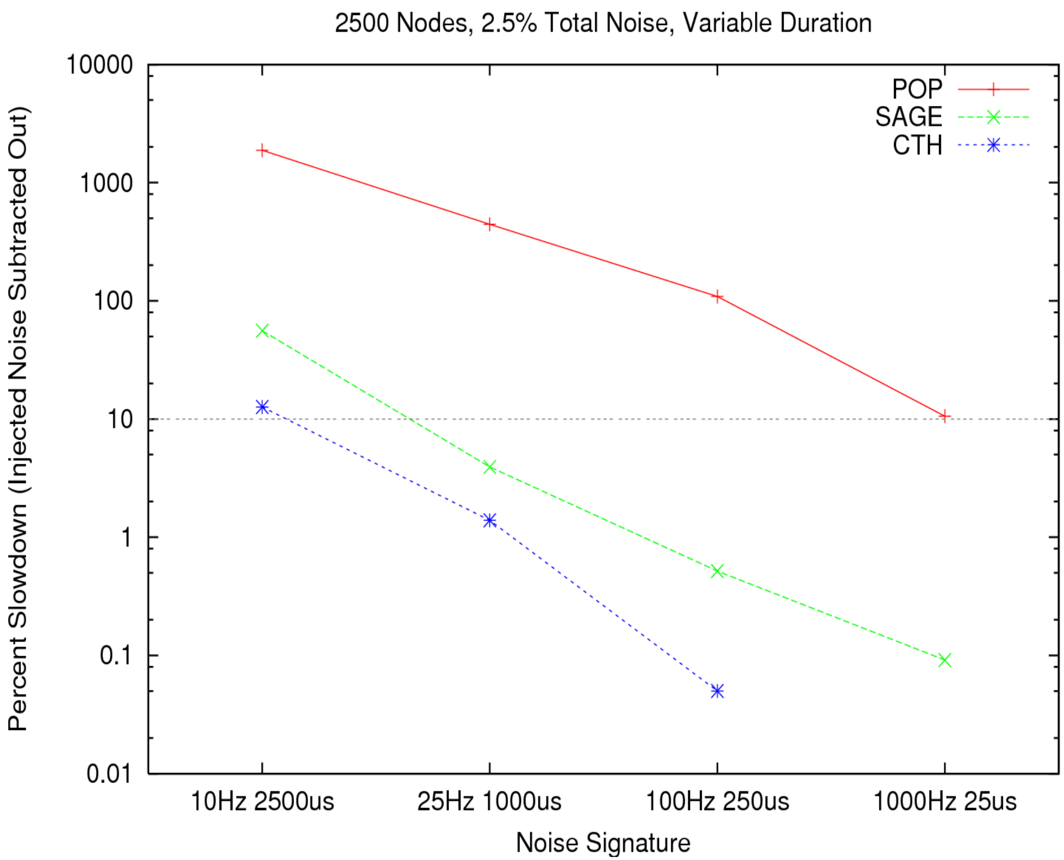
OS Noise Matters for Tightly Synchronized Applications



- Impact of noise increases with scale (basic probability)
- Idle noise measurements can distort reality
  - Not asking OS/network/mem to do anything
  - Micro-benchmark != real application

See "The Case of the Missing Supercomputer Performance", Petrini, et al.

Red Storm Catamount Noise Injection Experiments



- Result:
  Noise duration is more important than frequency
- OS should break up work into many small & short pieces
- Opposite of current efforts
  - Linux Dynaticks
- Cray CNL with 10 Hz timer had to revert back to 250 Hz due to OS noise duration issues

Noise Becomes Issue at Large Node Counts; Often Suddenly

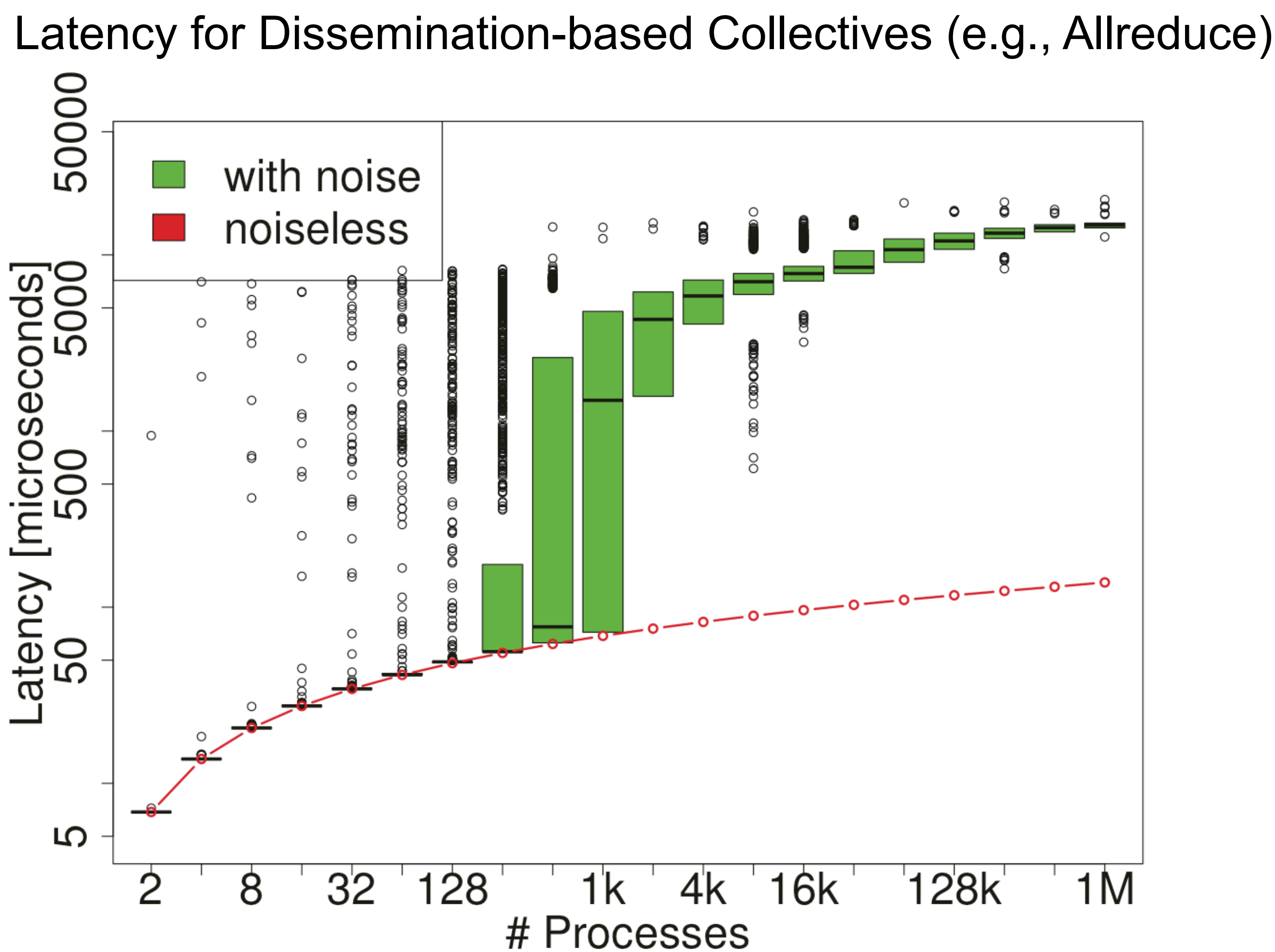
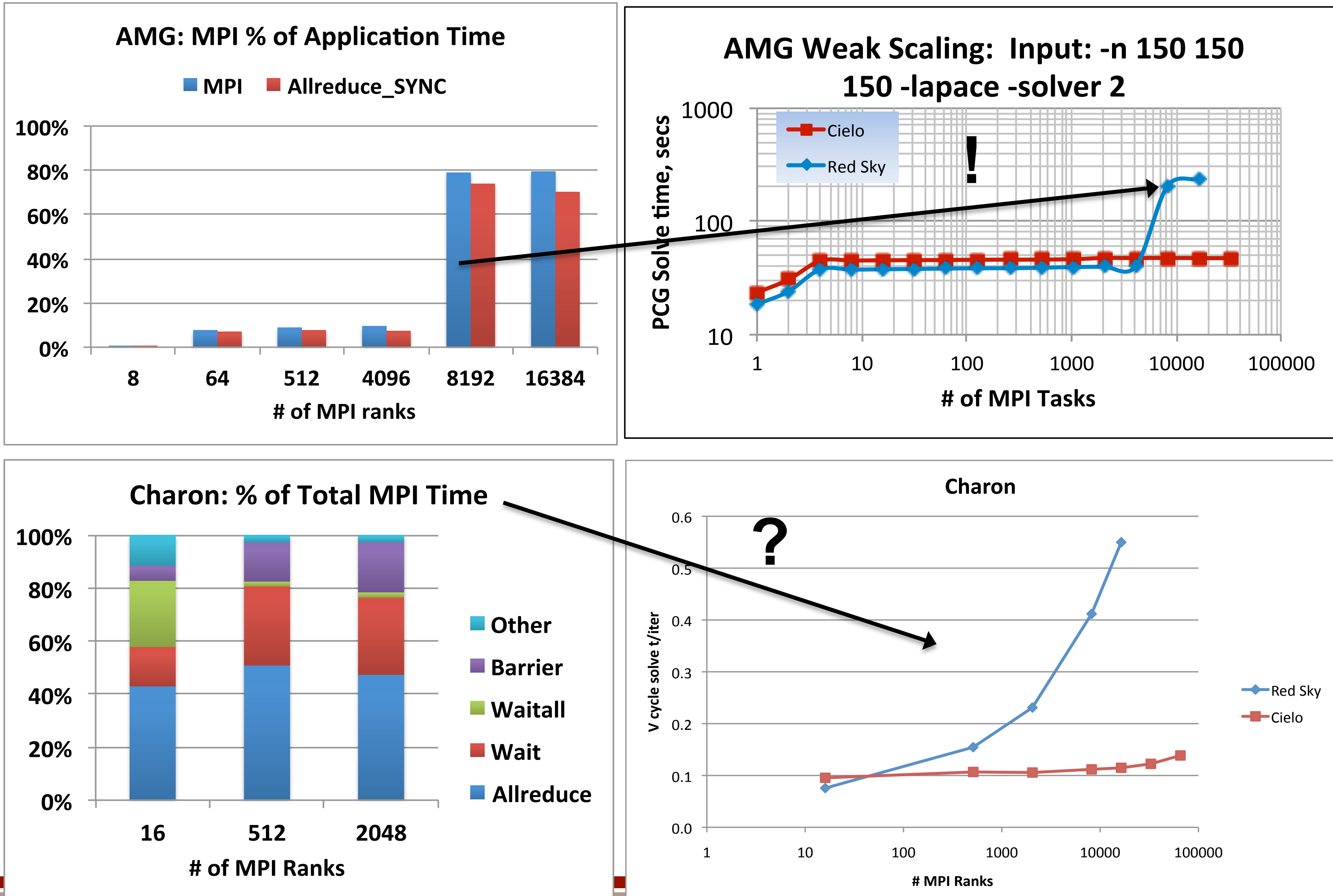


Figure Credit: Hoefler, et al.,
“Characterizing the Influence of System Noise to Large-Scale Applications by Simulation

OS Noise Mitigation Techniques are Well Understood

- Unsynchronized systems
  - Tune OS to have balanced noise pattern, short detours
  - Isolate OS work to set of sequestered cores
  - Non-blocking collectives
  - Algorithm changes, be more asynchronous, overlap
- Systems with global clock
  - Co-schedule OS activities across entire system

Scaling Differences: Linux != Linux



Light Weight Linux Experiments

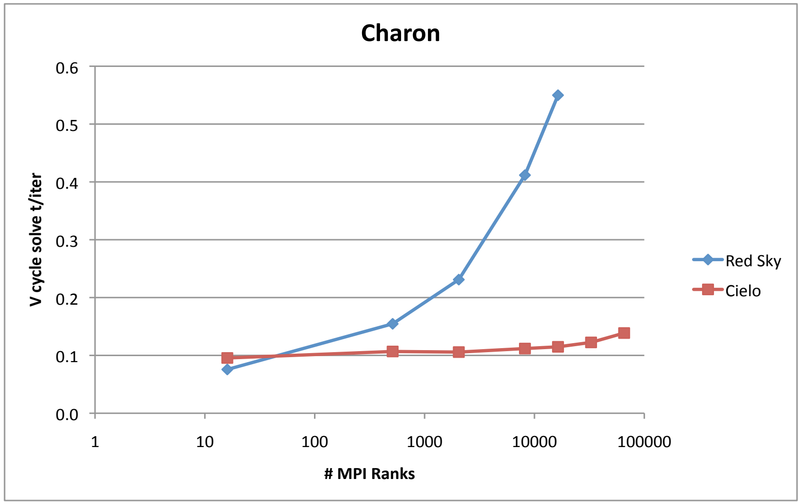
- LWOS/LWK environments are known to be a necessary, but not sufficient, condition for good scalability
- We took Red Sky's OS (heavy-weight OS) and modified it to include light-weight OS features
  - Installed on Red Sky test bed, Red Dune (single rack)
  - Lower-frequency timer interrupts (from 1000 Hz to 250 Hz)
  - Balanced timer interrupt handling, i.e. no single core taking all timer interrupts
  - Fewer system daemons
  - No periodic system health monitoring processes
- What impact does this "LWOS" have on some of the suspect application?
  - AMG MPI_ALLREDUCE issue at 8K PEs
  - Charon scaling above 512 PEs

Effect of "LWOS" on Charon

- Charon Performance at 512 PEs is significantly improved, 9.7%, when comparing HWOS and LWOS on Red Dune test bed
- LWOS performance approaches that seen on Cielo

Cielo	Red Sky	HWOS	LWOS
0.1068	0.1546	0.1231	0.1111

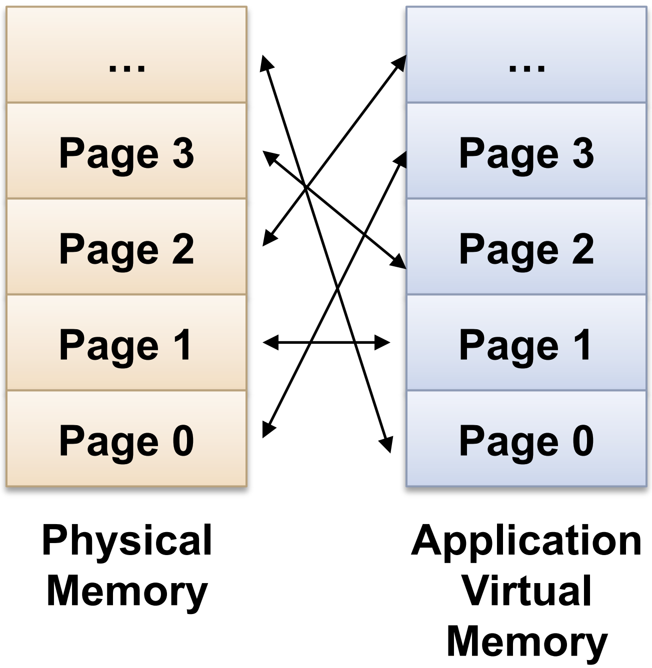
Original Charon Test Result



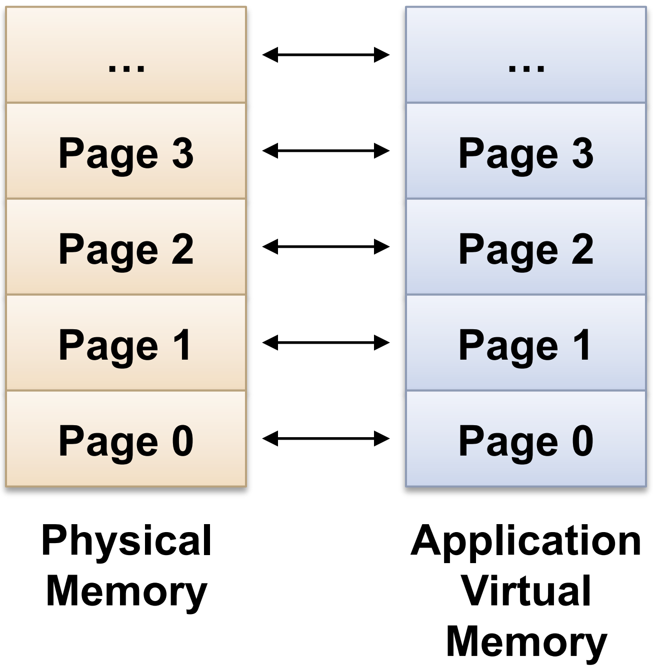
LWK Memory Management

- Simple, static virtual to physical mapping
  - Eliminates non-determinism
  - Enables straightforward use of large page sizes
  - Enables optimization in network stack
  - Physical memory managed by user-level process

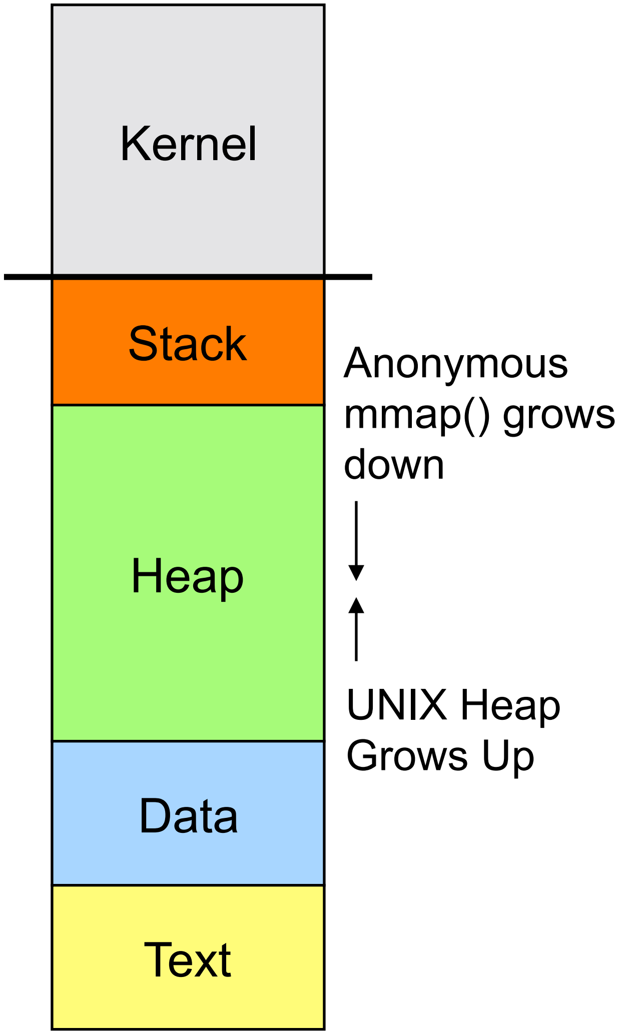
General-Purpose OS, Demand Paging



LWK Static Mapping



LWK Virtual Memory Regions

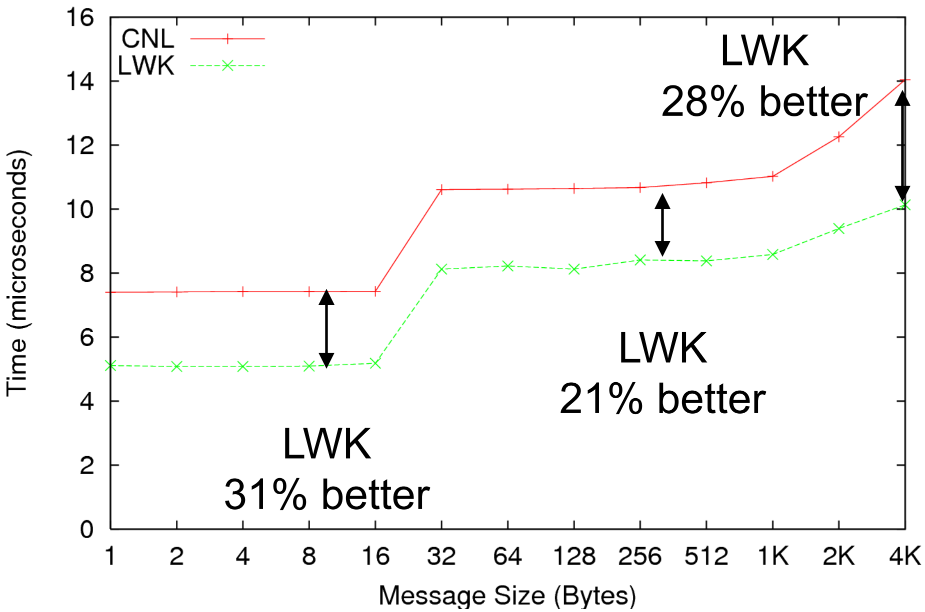


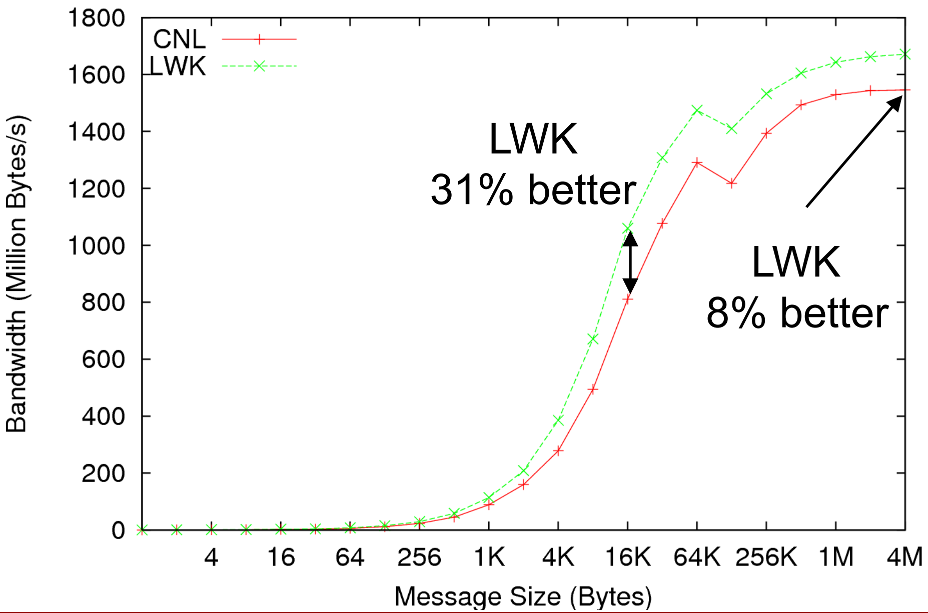
- User address space divided into virtual memory regions:
  - Text
  - Data
  - Heap
  - Stack
- Each region is mapped to a contiguous region of physical memory
  - Straightforward to use large pages
  - PCT in user-space sets up the mapping
- All virtual<->physical mapping occurs before application starts
  - No demand paging
  - No memory oversubscription

SeaStar Network Performance Benefited from LWK Memory Mgmt.

- LWK's static, contiguous memory layout simplifies network stack and HW
  - No pinning/unpinning overhead
  - Send address/length to SeaStar NIC
  - NIC does not need TLB or page table walk engine

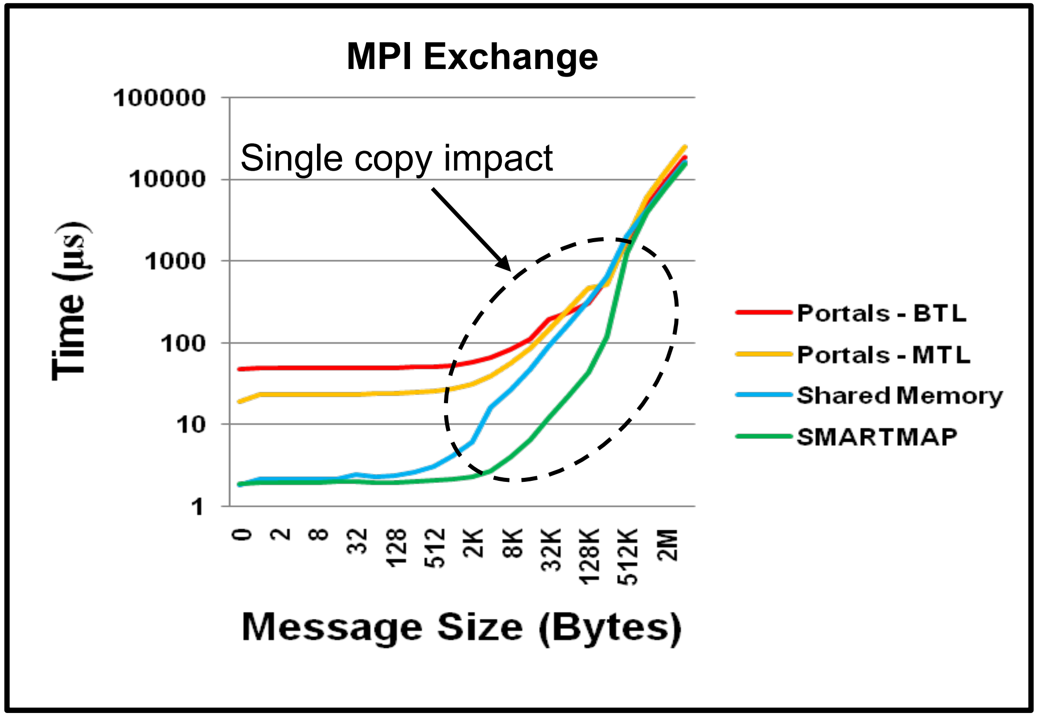
XT4/SeaStar Catamount vs. Cray Compute Node Linux
Host-based Network Stack (Generic Portals)



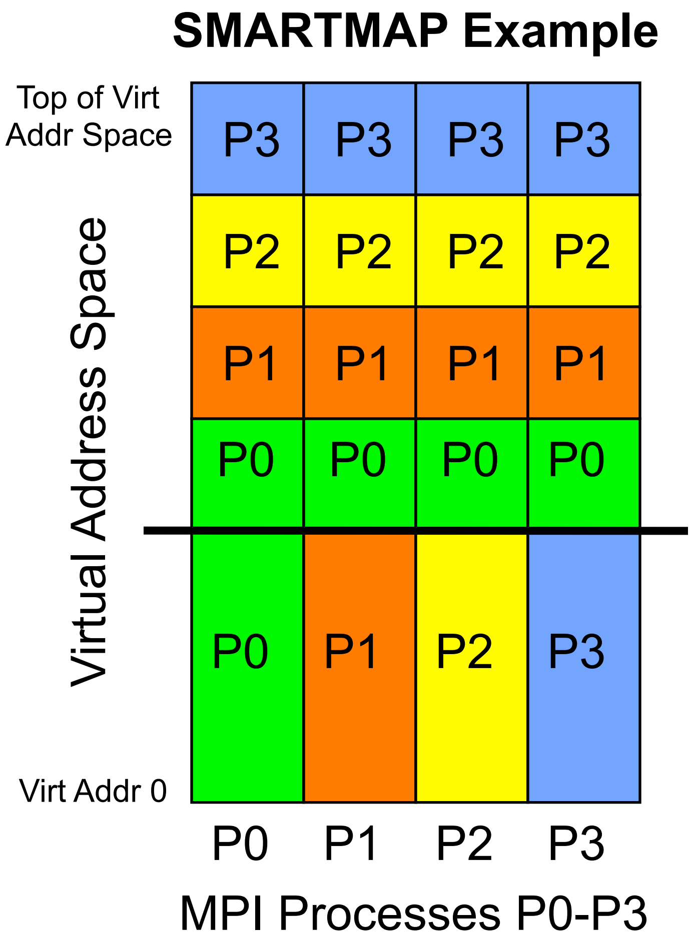


SMARTMAP Intra-node Optimization Eliminates Unnecessary Memory Copies

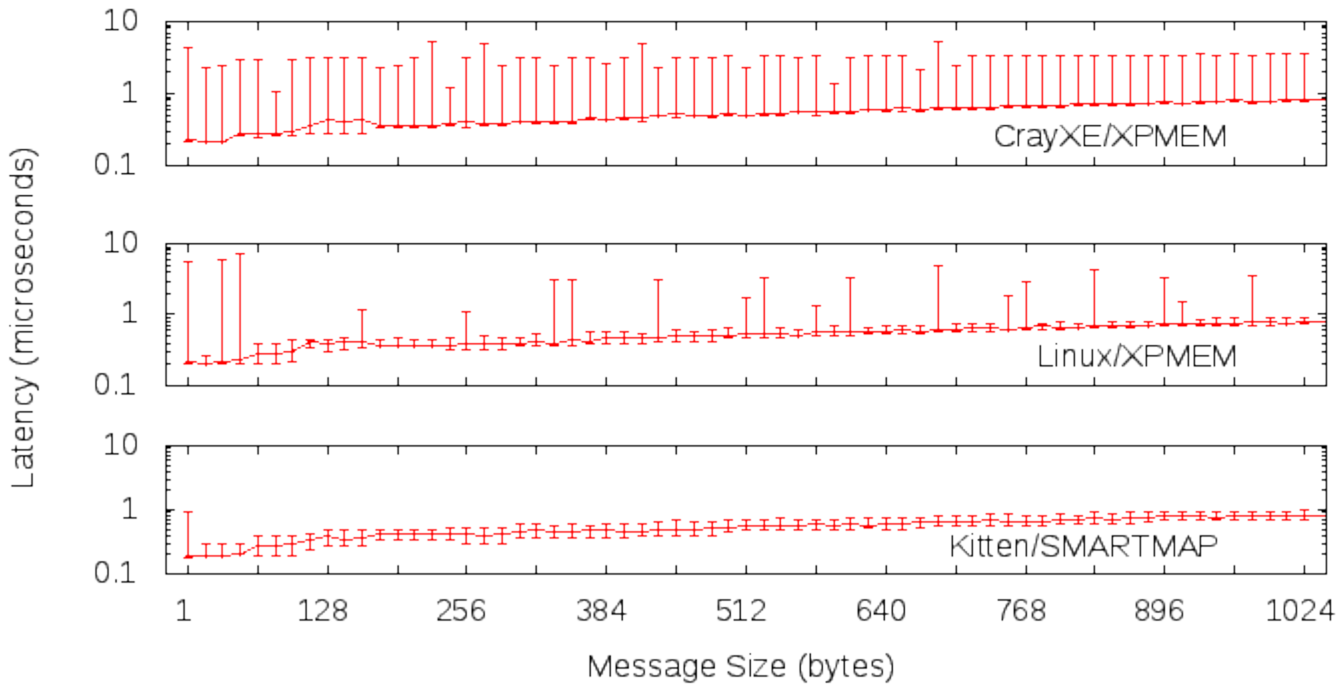
- **Basic Idea: Each process on a node maps the memory of all other processes on the same node into its virtual address space**
- **Enables single copy process to process message passing (vs. multiple copies in traditional approaches)**



See SC-08 paper



SHMEM over SMARTMAP/XPMEM



Cray XE / Linux (Stock CLE)

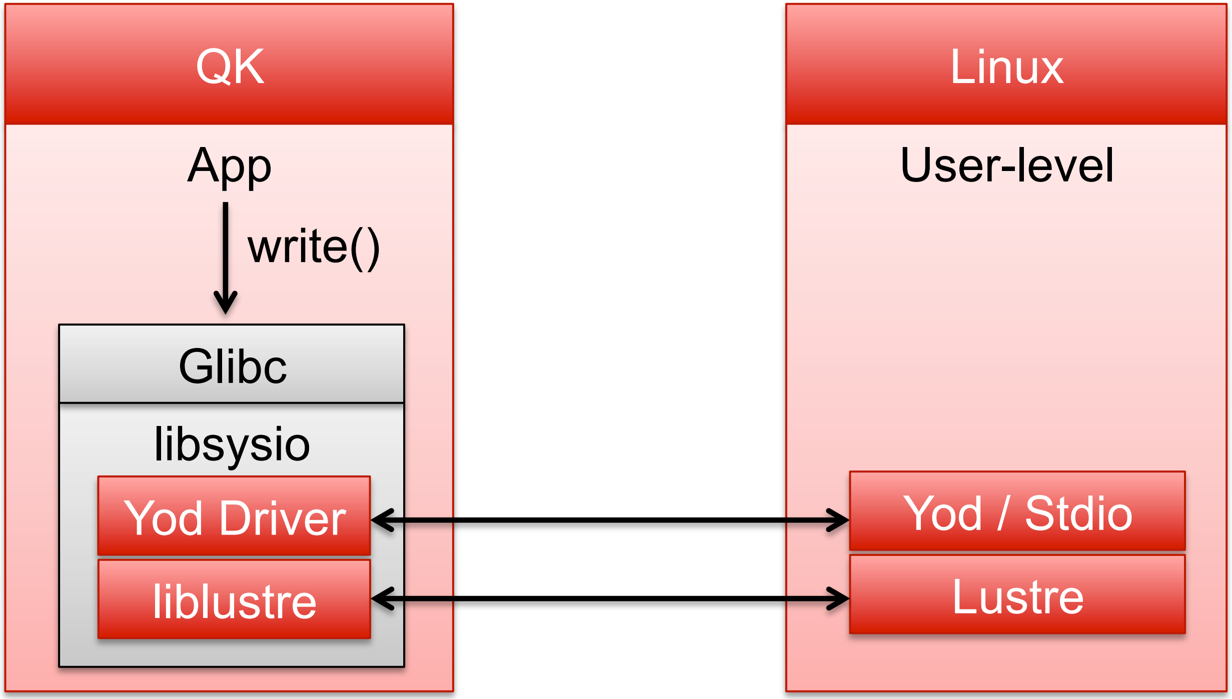
HP Blade / Linux

HP Blade / Kitten

See PGAS-11 paper

Catamount I/O Forwarding

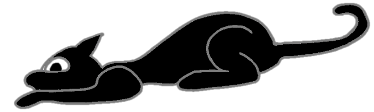
- Based on libsysio, user level VFS layer (on SourceForge)
- Stdio, liblustre, and ramfs drivers for libsysio
  - Portals used for all off-node communication
- Custom Glibc port
- Every compute node was a Lustre client



Outline

- Past / Background
  - Lightweight Kernel (LWK) as an optimization layer
- Present
  - Kitten LWK
- Future
  - LWK as tool for runtime <-> OS <-> HW co-design
- Closing thoughts

Kitten Lightweight Kernel



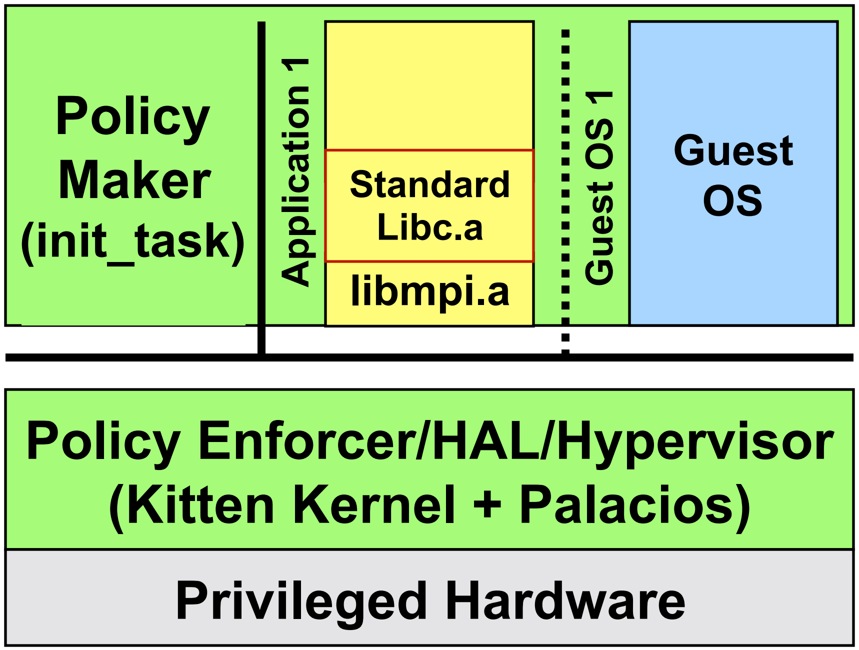
- Initial development funded by Sandia LDRD FY08-FY10
- Evolution of Sandia's line of LWKs
  - Better meet user, vendor, and researcher expectations for native LWK
  - Leverage virtualization when full-featured OS functionality needed
- Guiding Principles
  - The application/runtime knows best
  - Be deterministic whenever possible
  - Repurpose rather than reimplement
  - Fit into Linux ecosystem (use Linux API+ABI where possible)

Kitten Targets

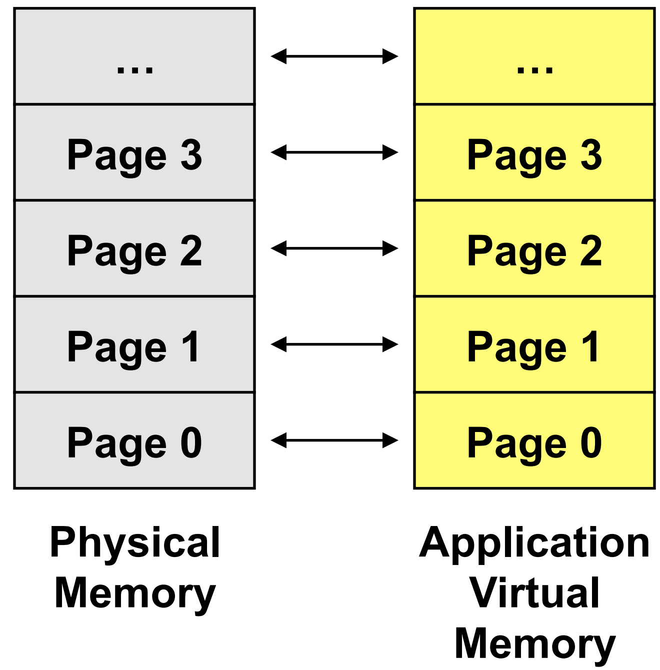
- Target 1) DOE's existing scientific computing application workloads running on extreme-scale, distributed-memory supercomputers with a tightly-coupled interconnect

- Target 2) Build a good platform for HPC OS research
  - Easy to work with codebase, relatively easy to understand
  - Allow more effort to be directed at research issue being explored, rather than working around Linux issues (e.g., memory pinning)
  - Give HPC-focused optimizations a reasonable shot at being deployed (me being optimistic)

Kitten Basic Architecture



Memory Management



- POSIX-like environment
- Inverted resource management
- Low noise OS noise/jitter
- Straight-forward network stack (e.g., no pinning)
- Less to go wrong, easier to harden

Kitten Kernel Implementation

- Monolithic, C code, GNU toolchain, Kbuild configuration
- Supports x86-64 architecture only, considering port 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.)
  - PCI, NUMA, ACPI, IOMMU, …
  - 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
  - User-level API for migrating tasks between cores

Kitten Thread Support

- Kitten user-applications link with standard GNU C library (Glibc) and other system libraries installed on the Linux build host
- Functionality added to Kitten to support Glibc NPTL POSIX threads implementation
  - Futex() system call (fast user-level locking)
  - Basic support for signals
  - Match Linux implementation of thread local storage
  - Support for multiple threads per CPU core, preemptively scheduled
- Kitten supports runtimes that work on top of POSIX threads
  - GOMP OpenMP implementation
  - Qthreads
  - Probably others with a little effort

Kitten Network Stack

- Based on Linux Open Fabrics Alliance (OFA) Infiniband stack
  - Added "Linux Compatibility Layer" to support Linux drivers
  - Supports user-level IB verbs host-to-host communication
  - Uses RDMACM, small hacks to avoid need for IP
  - OpenMPI Point-to-Point performance
    - 2.8 Gbytes/s for large messages (native 2.8 Gbytes/s)
    - Latency needs tuning: 2.9 us one-way latency (native ~1.3 us)
- Runs on Gato IB cluster at Sandia
  - 16 nodes, each with QDR ConnectX
  - Each node 2 socket Intel X5570 (Nehalem-class, 2.93 GHz), 24 GB mem

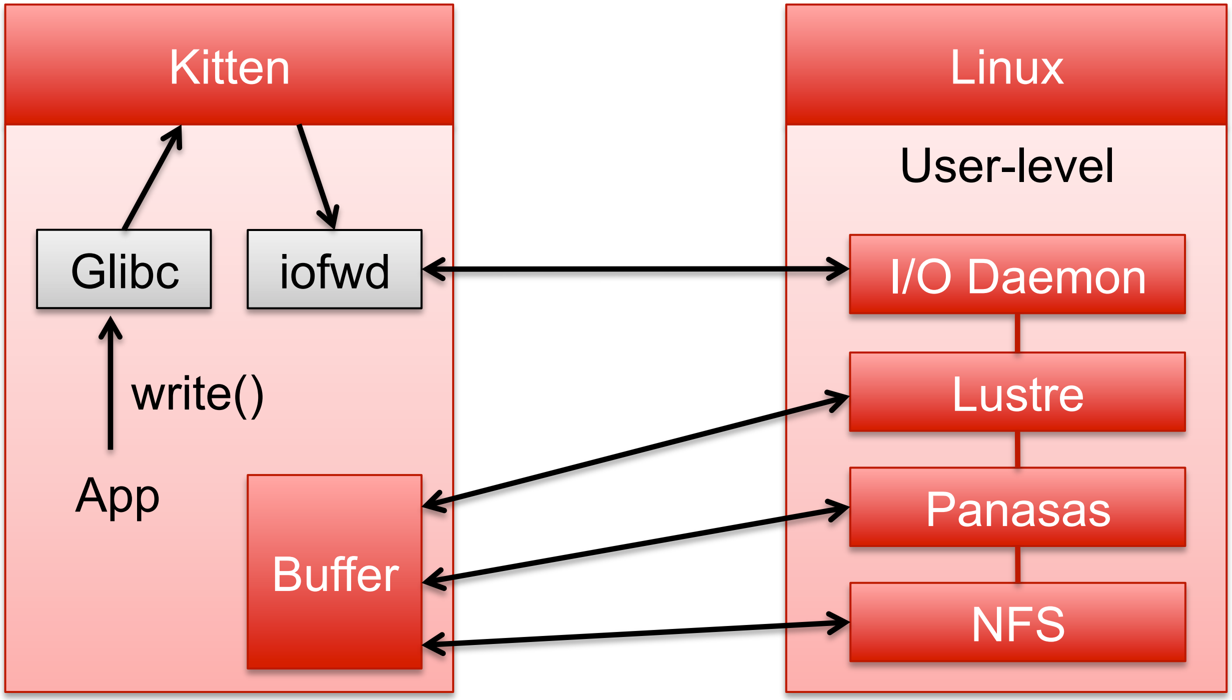
Kitten Job Launch

- Simplistic runtime over IB verbs
- Parallel application launcher (PAL) runs on Linux service node
- PAL pushes application to PCT running on compute node
- PCT sets up address space and starts application
- stdout redirected to PAL console
- Usage Example:

```
gato> pal -cpu 8 -nl 0xa000001..0xa00000f ./test_HPCCG 100 100 100
<8>(user-100) Total Time/FLOPS/MFLOPS                   = 83.5877/1.14432e+12/13690.
<8>(user-100) DDOT  Time/FLOPS/MFLOPS                   = 66.5237/7.152e+10/1075.1.
<8>(user-100)      Minimum DDOT MPI_Allreduce time (over all processors) = 0.410764
<8>(user-100)      Maximum DDOT MPI_Allreduce time (over all processors) = 65.6042
<8>(user-100)      Average DDOT MPI_Allreduce time (over all processors) = 17.4922
<8>(user-100) WAXPBY Time/FLOPS/MFLOPS                  = 2.78471/1.0728e+11/38524.7.
<8>(user-100) SPARSEMV Time/FLOPS/MFLOPS                = 14.2008/9.6552e+11/67990.5.
<8>(user-100) SPARSEMV MFLOPS W OVRHEAD                 = 3831.21.
<8>(user-100) SPARSEMV PARALLEL OVERHEAD Time           = 237.814 ( 94.3651 % ).
```

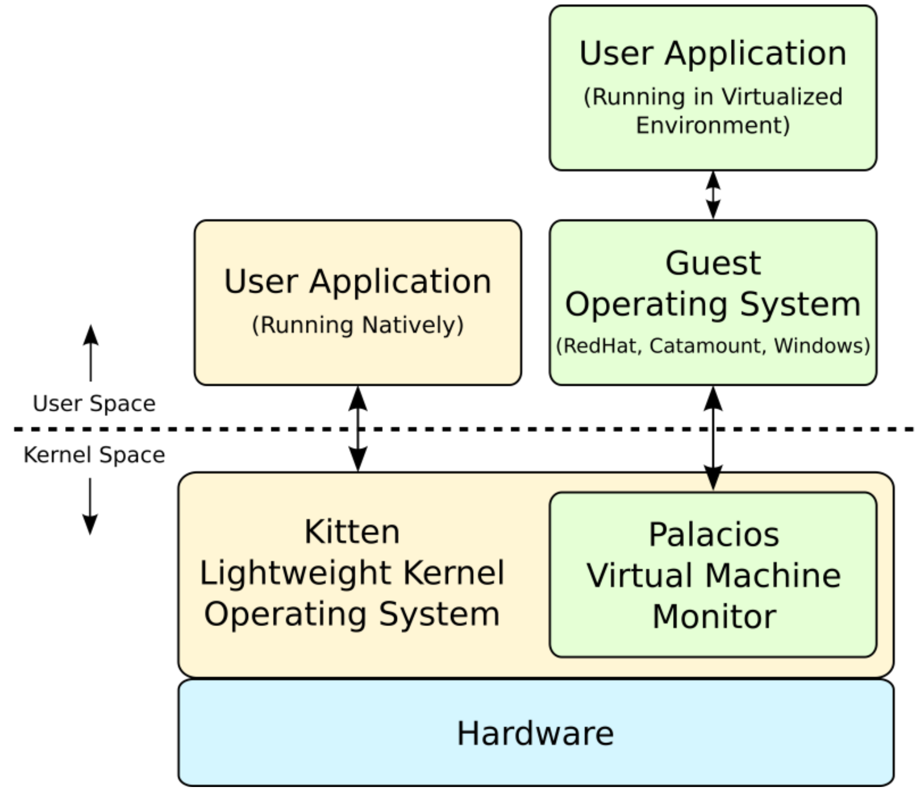
Kitten I/O Forwarding

- Paper design, no implementation yet
- Kitten reflects off-node I/O calls to user-space
  - Avoids need for custom Glibc port
  - Only control reflected, no extra buffer copies



Kitten Virtual Machine Support

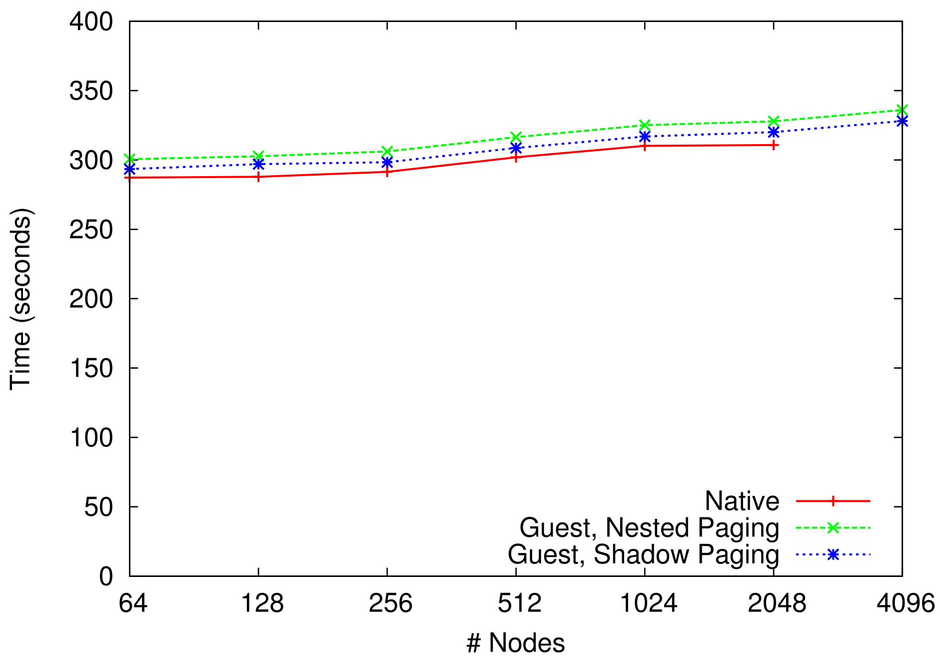
- Lightweight Kernels (LWK) traditionally have limited, fixed functionality
- Kitten LWK addresses this limitation by embedding a virtual machine monitor (collaboration with Northwestern Univ. and Univ. of New Mexico)
- Allows users to “boot” full-featured guest operating systems on-demand
- System architected for low virtualization overhead; takes advantage of Kitten’s simple memory management
- Conducted large scale experiments on Red Storm using micro-benchmarks and two full applications, CTH and Sage



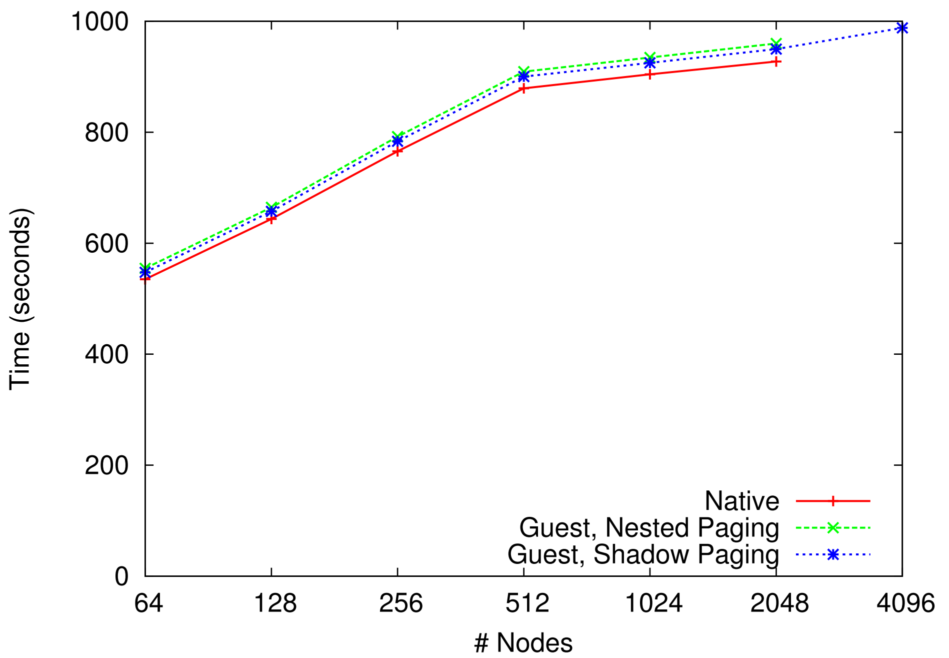
Kitten/Palacios Scalable Virtualization Experiments on Red Storm XT4

Native is Catamount running on 'bare metal', Guest is Catamount running as a guest operating system managed by Kitten/Palacios

CTH Hydrocode (SNL App)



Sage Hydrocode (LANL App)



Measured < 5% virtualization overhead for both applications

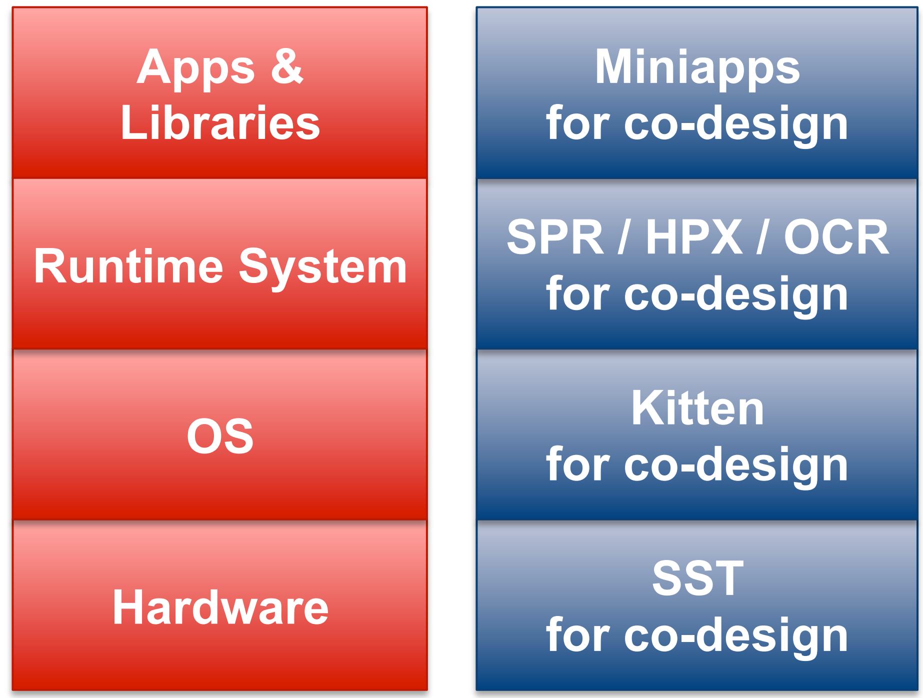
See VEE-11 paper

Outline

- Past / Background
  - Lightweight Kernel (LWK) as an optimization layer
- Present
  - Kitten LWK
- Future
  - LWK as tool for runtime <-> OS <-> HW co-design
- Closing thoughts

The OS is in the Middle

- Architectures are changing underneath OS
- Runtime systems and applications are changing above OS
- LWK can no longer be just an optimization layer
  - Too much changing!
- Need OS capability for design space exploration
  - Explore interfaces
  - Use novel HW capabilities
- Linux generally gets in way



Kitten is a Tool for Co-design

- Focusing on interfaces between Runtime <-> OS <-> HW
  - Kitten is a good starting point – a deconstructed OS ☺
  - Expect two way interaction between layers necessary
  - Persistent vs. Ephemeral; Global vs. Local

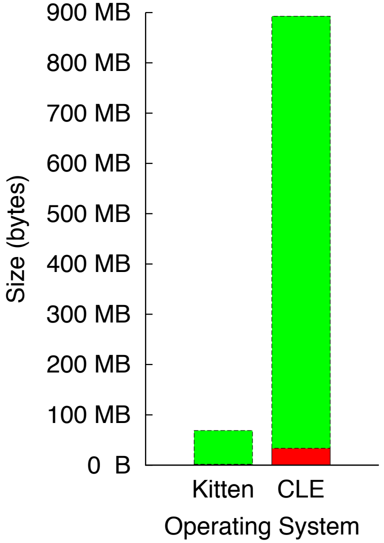
- New ASCR X-Stack 2 XPRESS project starting up
  - Involves Indiana, LSU, Houston, Oregon, RENCI, ORNL
  - Sandia is lead, major contribution is LXK OS, derived from Kitten
  - Runtime target of project is HPX-4, but also targeting other runtimes (SPR, OCR, …)

Virtualization is another Tool

- Virtualization uses in exascale timeframe
  - Backwards compatibility for legacy applications
  - As a development environment (e.g. emulate exa-system on laptop)
  - Portable containers for application environments
  - “Virtual Future Machine” (VFM) concept

- Recent virtualization activities
  - Integrated Palacios with SST to accelerate simulation
  - Developing high-performance virtual networking

Results



Kitten > 10x smaller memory footprint

OS needs to be hardened against faults, keep running for app-level resilience

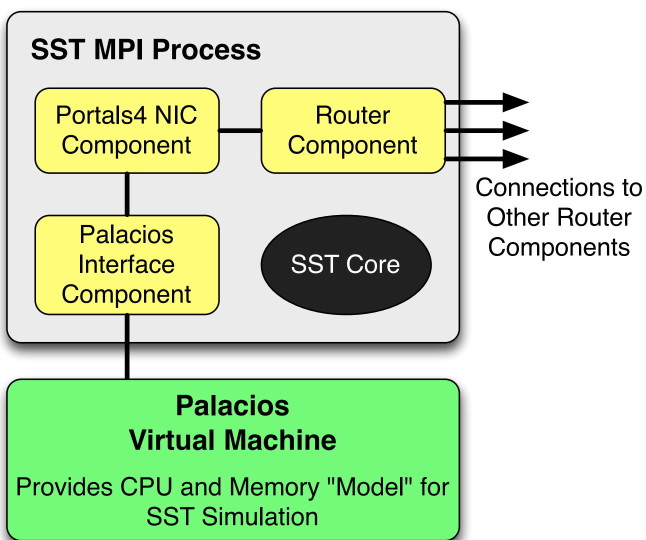
ROSS Workshop Paper

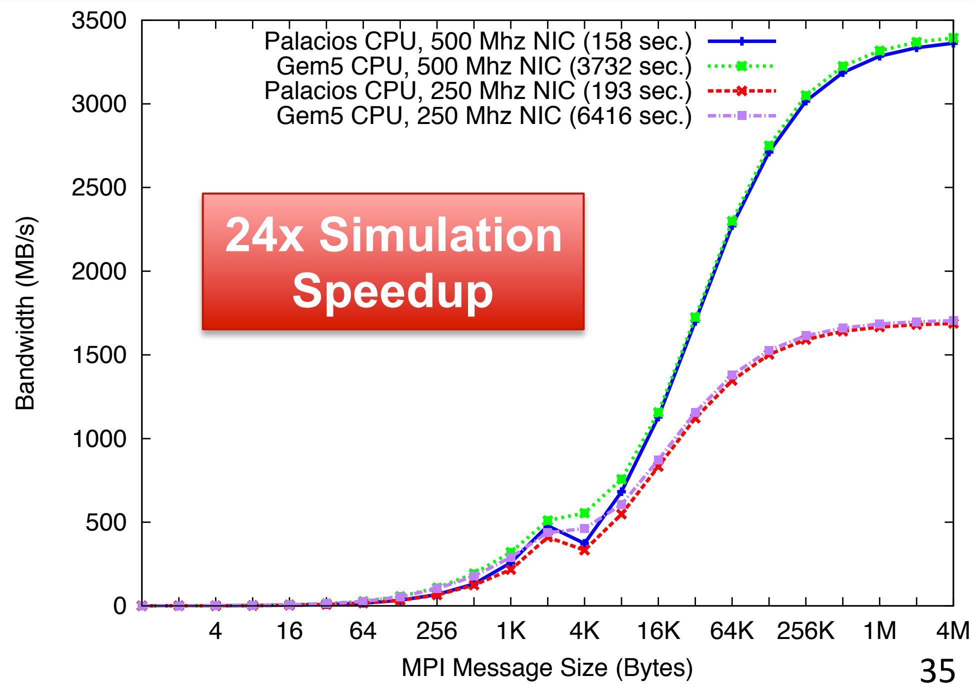
Operating System	Round-trip Task Migration Time (task migrates from core A to B and back)
Linux 2.6.35.7	4435 ns
Kitten 1.3	2630 ns

Kitten integrated SST/gem5 to enable rapid prototyping and reproducibility

SimuTools'12 Paper

SST CPU and Memory Model Implemented by Palacios VM





Conclusion

- Past: LWK as an optimization layer
- Present: Kitten is a modern LWK foundation
- Future: LWK enabling co-design in X-stack R&D

- Happy to discuss collaboration ideas
  - Improving lightweight Linux software stack
  - Incorporating virtualization layer in Appro's software stack
  - Bringup LXK/HPX on large-scale Appro systems

Acknowledgments

- Ron Brightwell (SNL)
- Doug Doerfler (SNL)
- Kurt Ferreira (SNL)

Questions and Discussion