XEMEM: Efficient Shared Memory for Composed Applications on Multi-OS/R Exascale Systems

Brian Kocoloski

Jack Lange

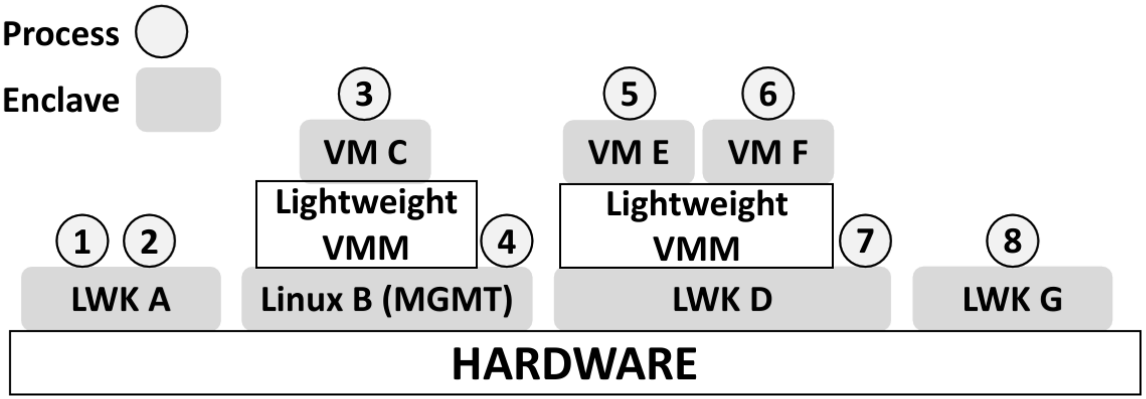
University of Pittsburgh

Multi-Enclave Exascale Systems

- Recent efforts in exascale operating systems and runtimes (OS/R):
  - **Hobbes (SNL, LBNL, LANL, ORNL, U. Pitt, various universities)**
  - Argo (ANL, LLNL, PNNL, various universities)
  - mOS (Intel), FusedOS (Intel / IBM)
  - McKernel (RIKEN AICS, University of Tokyo)
- **Common theme: no “one-size-fits-all” OS/R at exascale**
  - Significant heterogeneity in resources on exascale nodes
  - Applications specialized for specific hardware and runtime environment
- **Challenge: how can applications coordinate across multiple OS/R instances?**
  - Communication required for composed workloads, system services
- **This talk: XEMEM, efficient shared memory for a functional multi-OS/R exascale environment**

XEMEM: Cross Enclave Memory

***Enclave*: Partition of node hardware and independent system software environment (e.g., lightweight kernel (LWK), Linux, virtual machine (VM))**



- **XEMEM supports shared memory between all processes**
  - LWK processes, Linux processes, VM processes
  - Supports composed application workflows and system services
- **Unmodified applications written for single OS/R supported**
  - API backwards compatible with Cray/SGI XPMEM API
  - **Requires no user level knowledge of enclave configuration**

Talk Roadmap

- Multi-OS/R Shared Memory
- XEMEM Implementation
- Evaluation
- Conclusion/Questions

4 Tenets of Multi-OS/R Shared Memory

1. **Maintain Simplicity of Single OS Programming**
   - Multi-OS/R programming should not be more difficult than single OS

2. **Support Arbitrary Enclave Topologies**
   - System should not require a particular enclave configuration
   - Processes should not need knowledge of topology

3. Be Resource Efficient, Provide Dynamic Mappings
   - Construct memory mappings at the granularity requested by processes

4. Employ Localized Address Space Management
   - Avoid error prone manipulation of remote enclave address spaces

Maintain Simplicity of Single OS Programming

- Two key challenges: **unique naming and discoverability**
- These operations are simple in a single OS
  - Unique naming: e.g., tuple <PID, virtual address>
  - Discoverability: plethora of shared infrastructure (e.g., filesystems)
- **However, lack of shared infrastructure and global address space complicates multi-OS/R system**
  - Each enclave has a different PID space, filesystems, etc.
- **Our approach: name server enclave**
  - Naming: allocate globally unique IDs for all shared memory regions
  - Discoverability: allow enclaves to query existence of shared regions

XEMEM Shared Memory Protocol

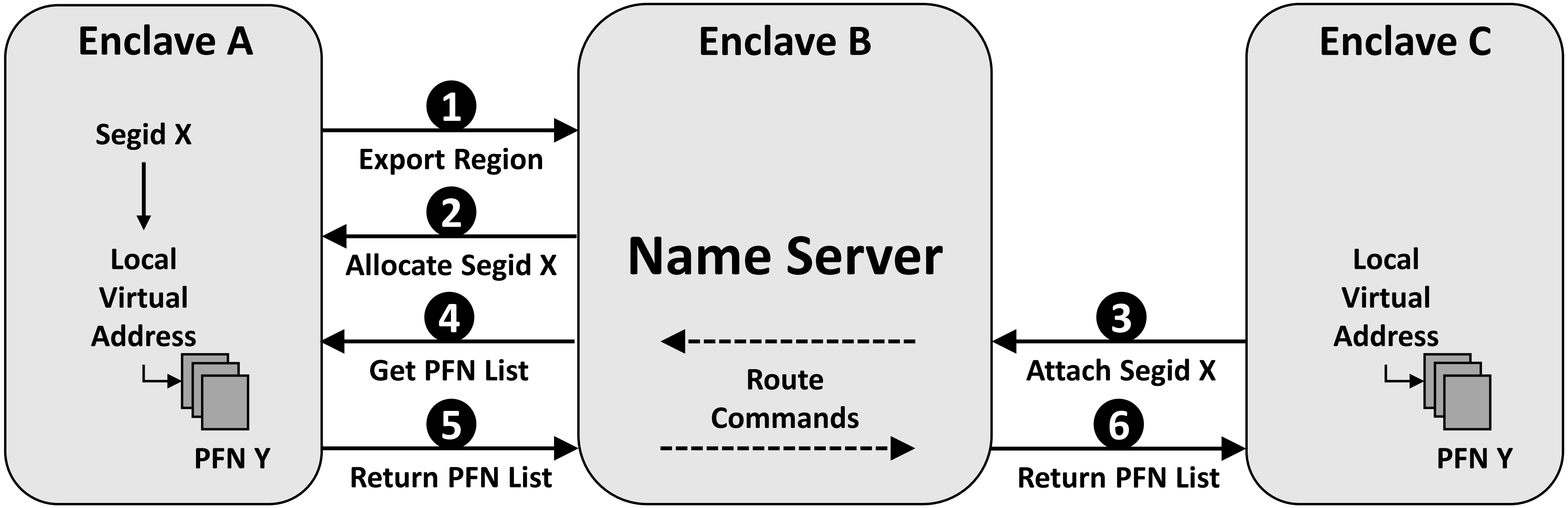
- Protocol based on the Cray/SGI XPMEM user-level API
- Allows sharing of arbitrary virtual address ranges, no explicit allocation of shared memory
- Focus on ***xpmem_make*** and ***xpmem_attach***

Function	Operation
xpmem_make	**Export address region as shared memory. Returns *segid***
xpmem_remove	Remove an exported region associated with a *segid*
xpmem_get	Request access to shared memory region associated with *segid*. Returns permission grant
xpmem_release	Release permission grant
xpmem_attach	**Map a region of shared memory associated with a *segid***
xpmem_detach	Unmap region of shared memory

- **Processes not required to have knowledge of underlying topology**
- Q: How does an enclave know which destination enclave to send to?
  - **By default, messages are sent to the name server, which is aware of enclave topological locations**

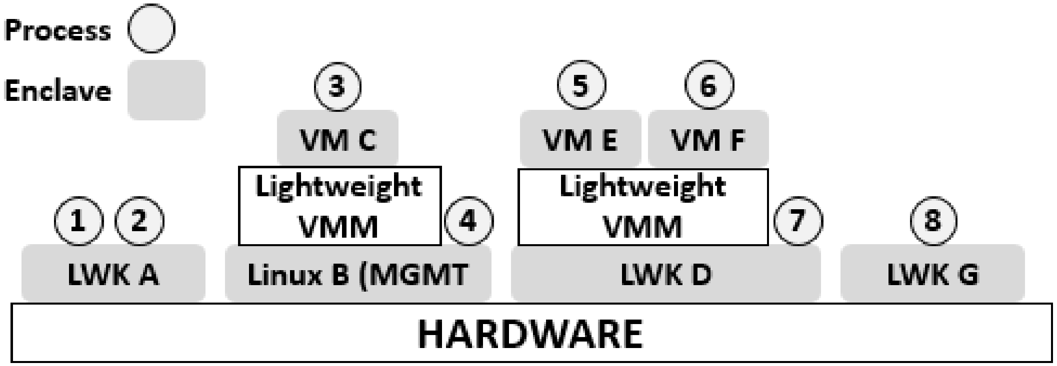
XEMEM Shared Memory Protocol

Invokes *xpmem_make* Invokes *xpmem_attach*

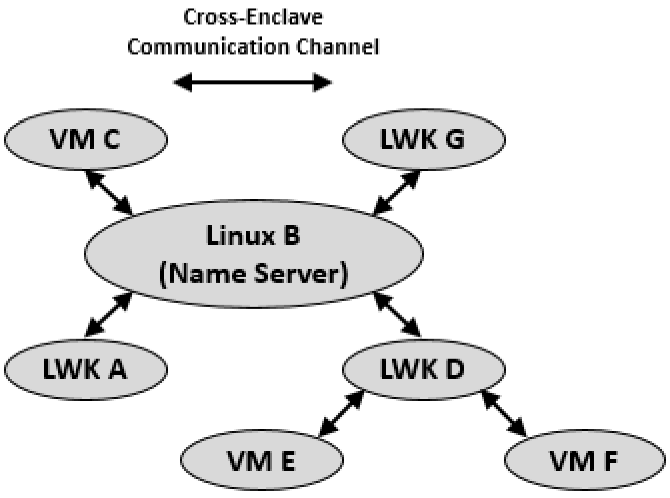


Enclave Topology

- Enclave Topology: architectural partitioning and inter-enclave communication interfaces
- **Assumption: no guarantee of point to point communication interfaces**



Hardware + system software partitioning



Enclave Topology

Arbitrary Enclave Topologies

- Topologies for different architectures may be significantly different
  - Virtualization capabilities may or may not be present
  - Application workloads may be different on different nodes and require different types of enclaves
  - **Node workload characteristics will be dynamic and may change according to application requirements**

- **At the same time, user-level should not be required to understand this topology**

- **Our approach: support arbitrary communication by routing messages hierarchically according to enclave topology**

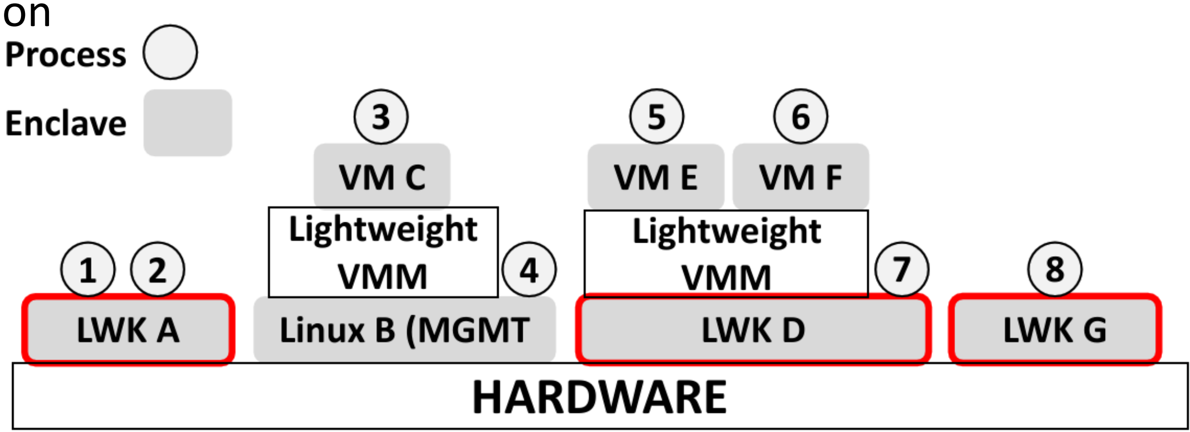
XEMEM Implementation

- Kitten Lightweight Kernel (LWK)
  - **New feature:** Dynamic heap expansion
  - **New feature:** Integration with SMARTMAP
- Palacios Virtual Machine Monitor (VMM)
  - **New feature:** Host-to-guest memory sharing
  - **New feature:** Guest-to-host memory sharing
- Pisces Co-kernel framework
  - **New feature:** Cross-enclave page frame shipping

OS/R Fundamentals: Kitten

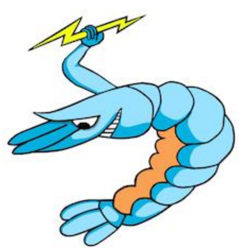
- Lightweight kernel (LWK) from Sandia National Laboratories designed to execute massively parallel HPC applications
- **Major design goal: provide more repeatable performance than general purpose OS (like Linux) for complex workloads**
- **XEMEM challenges**
  - Kitten pre-maps all VA space to physical memory at process creation
  - Kitten uses SMARTMAP for local enclave shared memory
- **XEMEM features**
  - Dynamic heap expansion
  - Integration with SMARTMAP

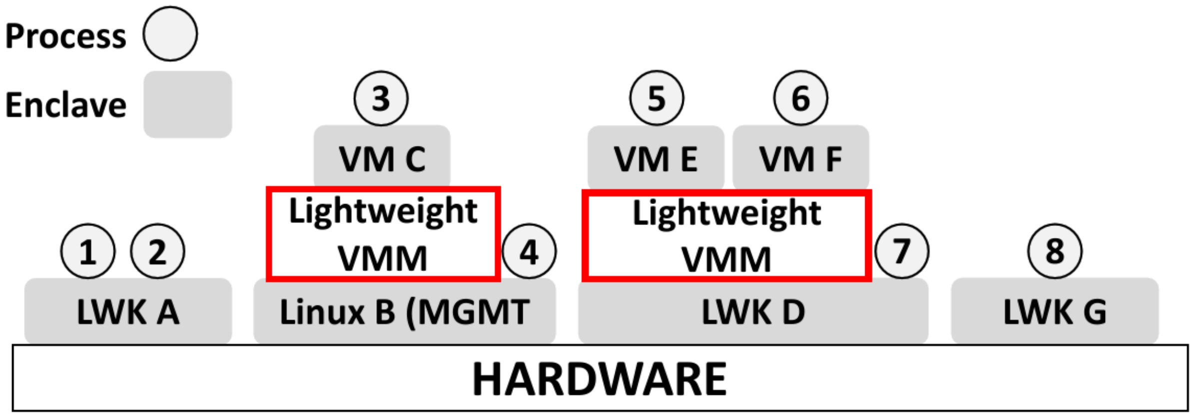
https://software.sandia.gov/trac/kitten



OS/R Fundamentals: Palacios

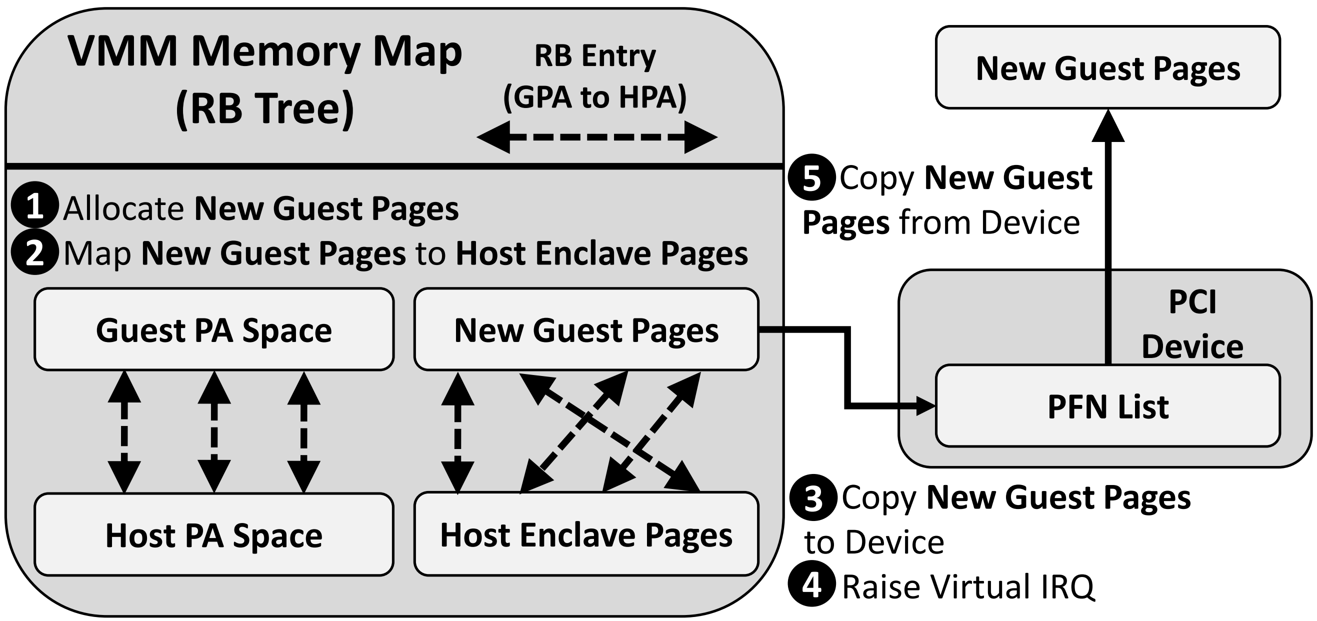
- Like Kitten, designed to execute massively parallel HPC applications
- Lightweight resource management policies
- Established history of providing virtualized environments for HPC
  - **Palacios + Kitten: Near native performance at 4096 nodes of a Cray XT3 [Lange et al., VEE '11]**
  - **Palacios + Linux: Better than native performance with Kitten as guest [Kocoloski and Lange, ROSS '12]**



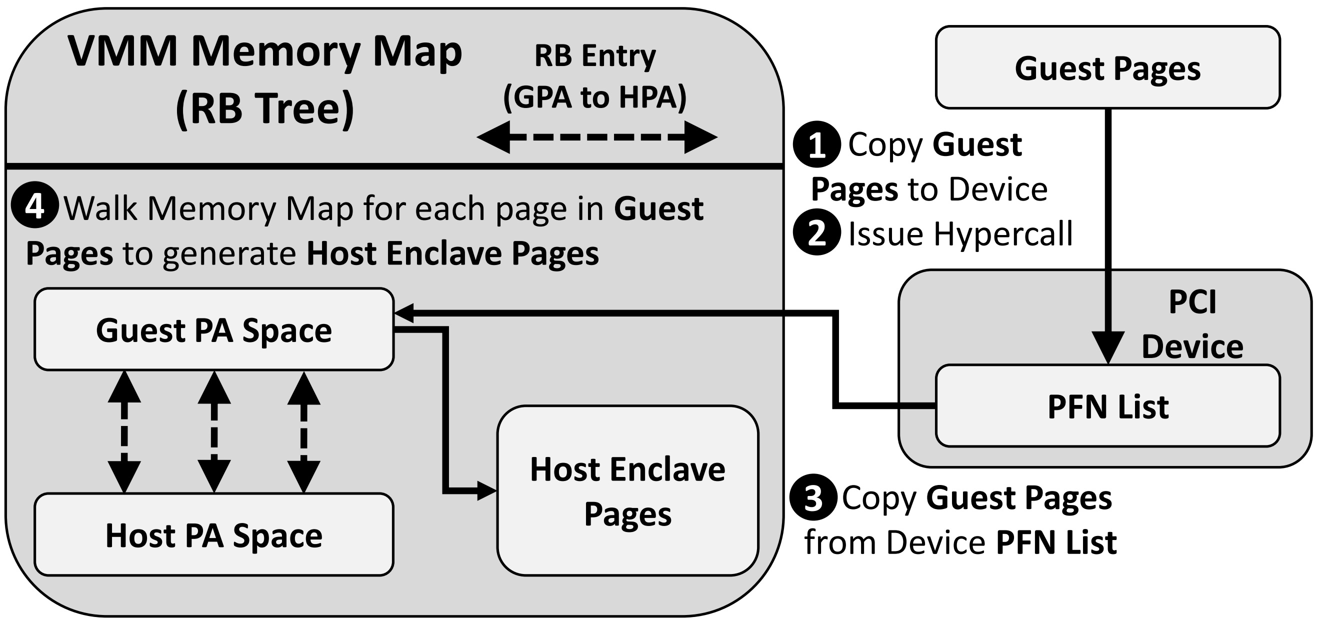


http://www.prognosticlab.org/palacios
http://v3vee.org

Palacios **Host-to-Guest** XEMEM Implementation

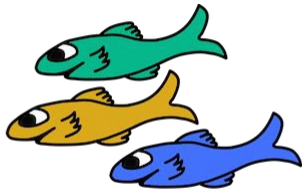


Palacios **Guest-to-Host** XEMEM Implementation

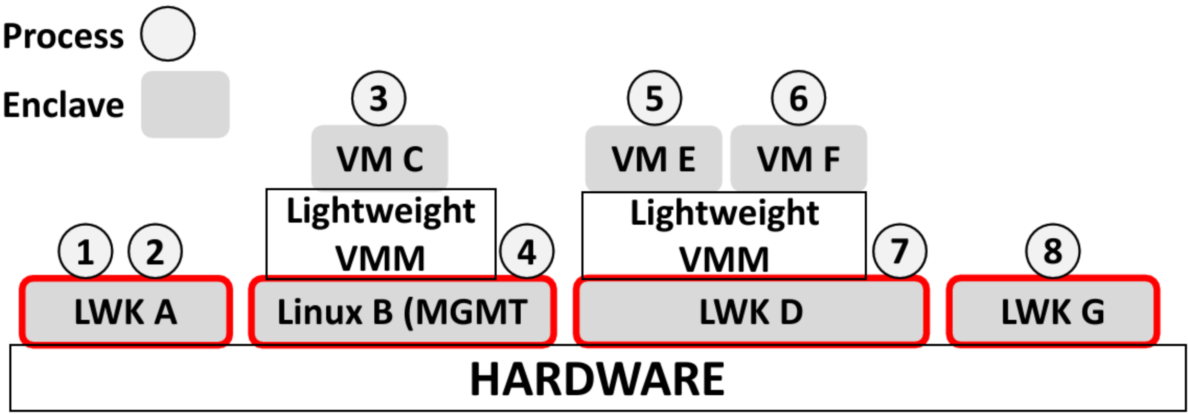


OS/R Fundamentals: Pisces

- **Upcoming HPDC talk (tomorrow 2PM): Jiannan Ouyang: Achieving Performance Isolation with Lightweight Co-Kernels**
- Lightweight "co-kernel" architecture
  - Decomposes node's hardware into partitions managed by independent system software environments ("co-kernels")
  - **Primary design goal: provide strong performance isolation between enclaves**
- **XEMEM feature**: cross-enclave shipping of page frame lists via inter-processor interrupts (IPIs)



http://www.prognosticlab.org/pisces

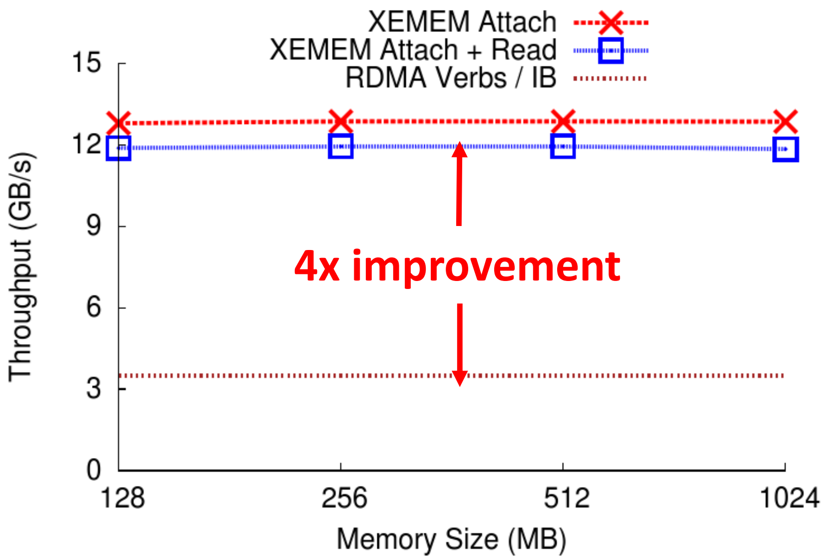


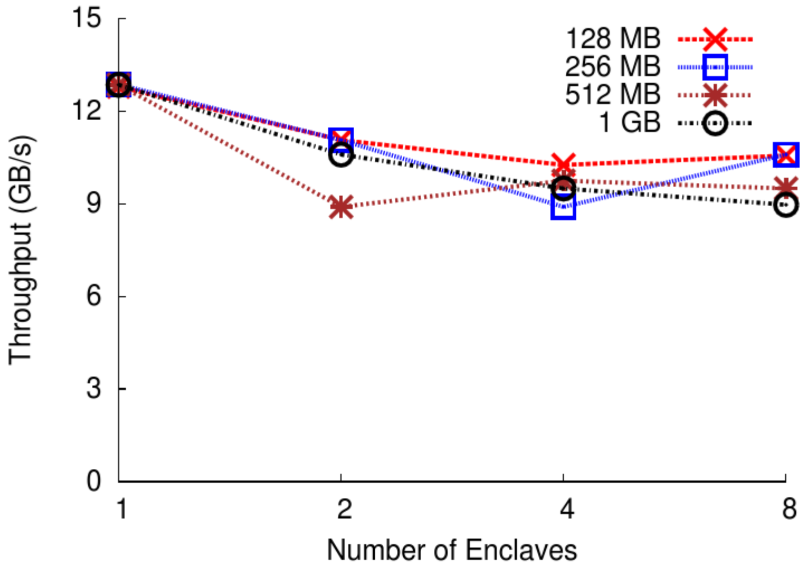
Evaluation

- 2 Part Evaluation
  - Analysis of shared memory performance and XEMEM overheads
  - Analysis of a sample *in situ* workload
- Synthetic benchmarks
  - Measure "time to availability" (TTA): **time from attachment request to attachment completion**
- Sample *In situ* workload
  - Measure runtime in an application composed in 2 separate enclaves
  - Demonstrate benefits of **performance isolation** that multi-enclave systems provide

Shared Memory Performance

- Kitten enclave exports memory region
- Process in Linux enclave attaches





- **4x improvement compared to RDMA over SR/IOV**

- **1,2,4,8 Kitten enclaves, 1,2,4,8 attaching processes in Linux**
- **Good scalability as memory size increases**

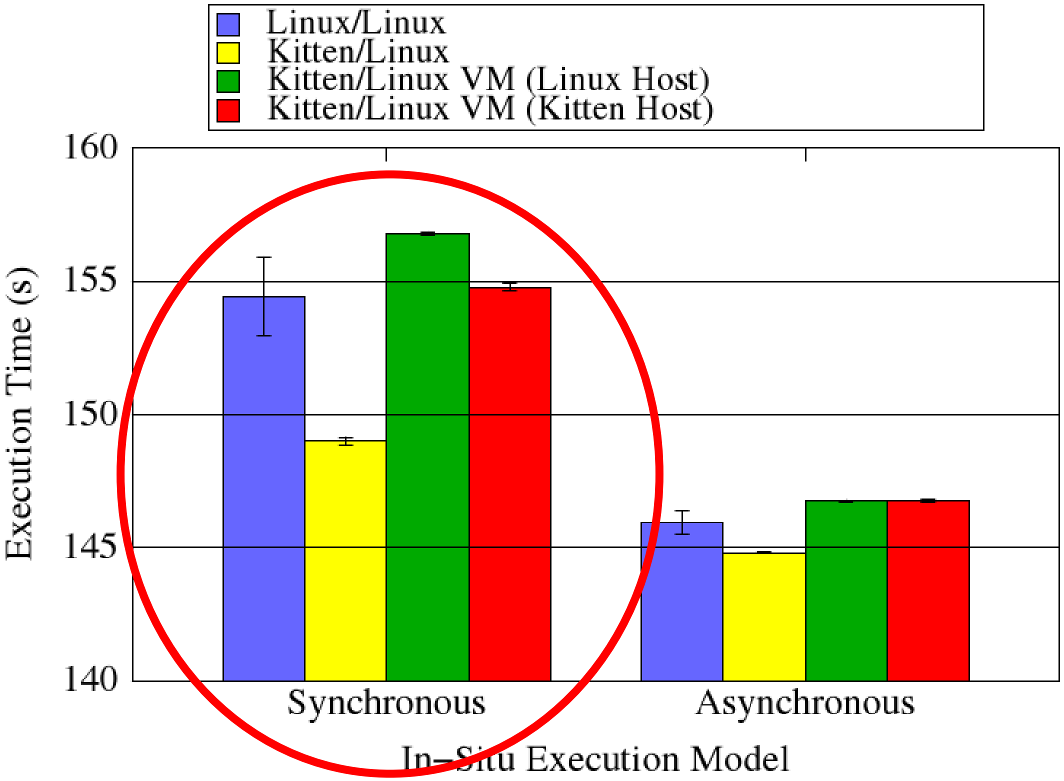
In Situ Workload Evaluation

- Sample *in situ* workload
  - HPCCG from Mantevo (simulation)
  - STREAM (analytics)
  - Components communicate via “signals” (polling on variables in shared memory)

- HPCCG performs iterative conjugate gradient solver
  - Configured to share 512MB with STREAM after certain periodic iterations
  - Send “signals” to begin STREAM execution

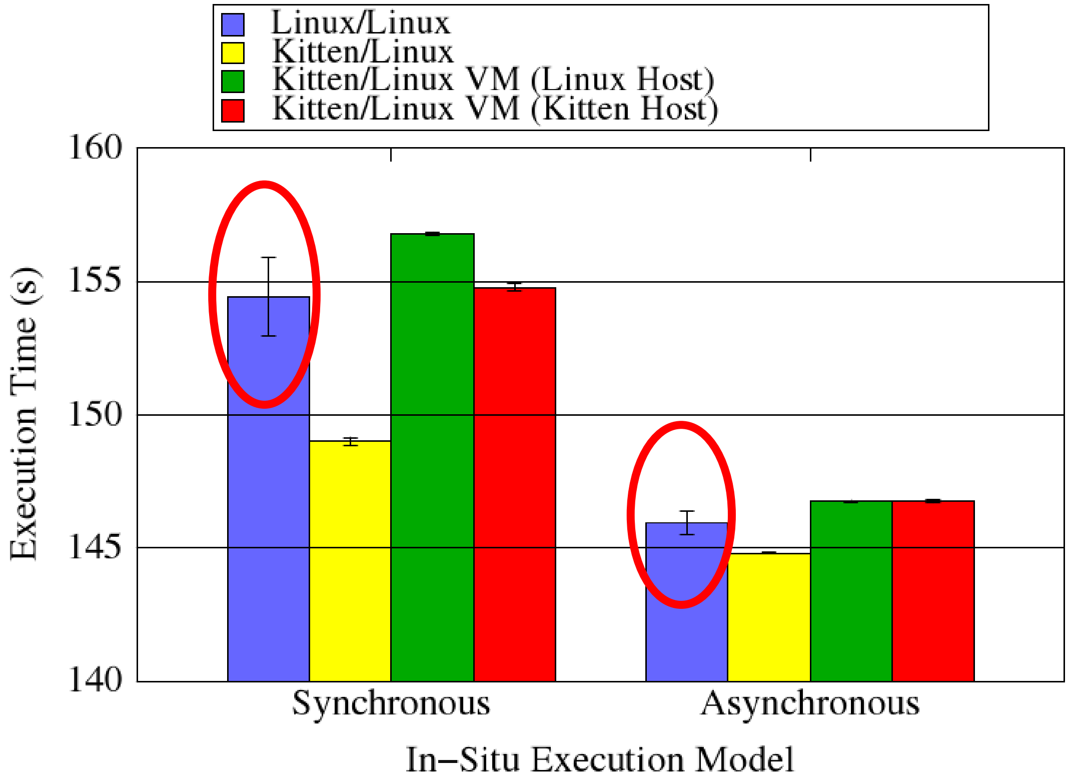
- Synchronous vs asynchronous execution
  - Synchronous: single program executes at a time
  - Asynchronous: programs execute simultaneously

In Situ Workload: Single Node



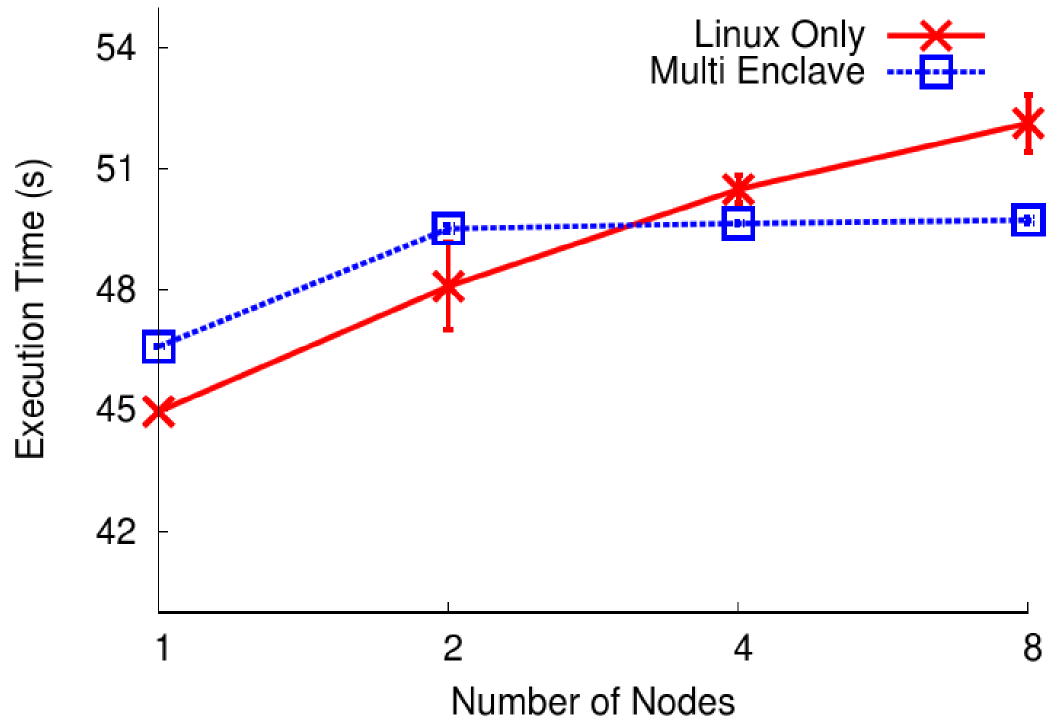
- **Best performance: HPCCG in Kitten co-kernel, STREAM in Linux**
- Synchronous: shared memory overhead in critical path

In Situ Workload: Single Node



- **Best performance: HPCCG in Kitten co-kernel, STREAM in Linux**
- Synchronous: shared memory overhead in critical path
- **Single OS/R lacks performance isolation (e.g,., demand page faulting)**

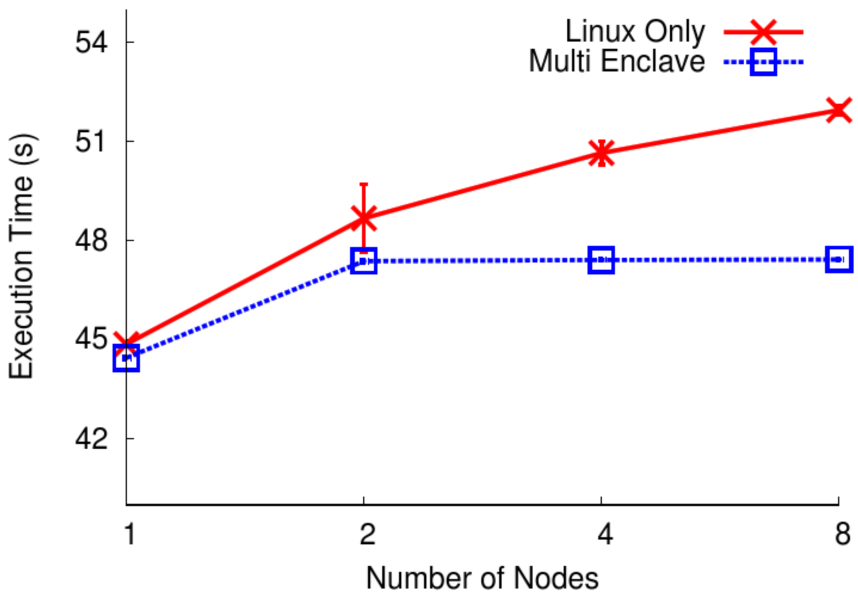
In Situ Workload: Multiple Nodes



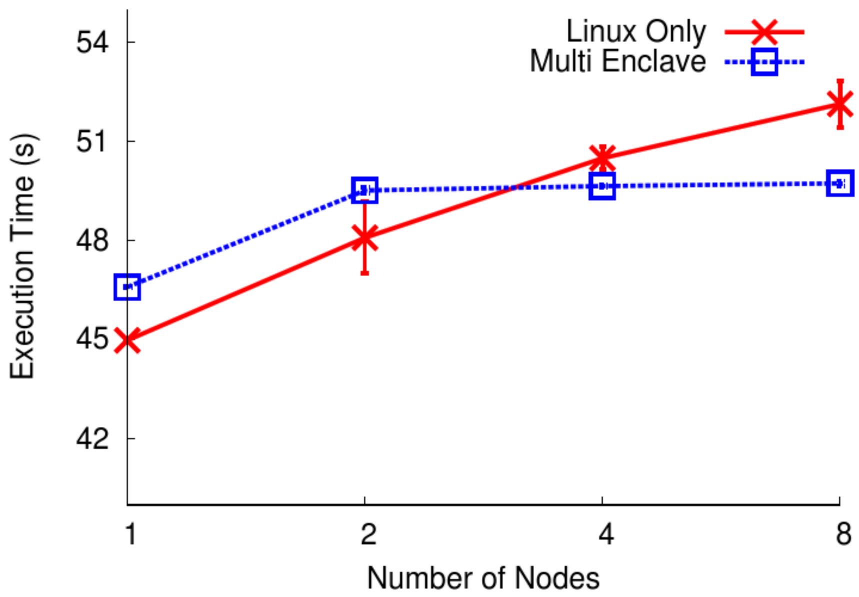
- **Multi Enclave: STREAM in native Linux, HPCCG in VM hosted by Kitten co-kernel**
- **Single node performance isolation leads to better scaling behavior**

Virtualization: Better than Native due to Performance Isolation

Single Persistent Attachment



Multiple Attachments



- **Multi Enclave: STREAM in native Linux, HPCCG in VM hosted by Kitten co-kernel**
- **Performance isolation leads to better than native performance**

Conclusion

- Multi-enclave approaches to exascale OS/Rs are gaining traction
- Composed applications and system services will require the ability to communicate across enclave boundaries
- **XEMEM: efficient shared memory for multi-OS/R systems**
  - Maintains simplicity of single OS programming
  - Supports arbitrary enclave topologies
- XEMEM implemented in a functional exascale multi-OS/R prototype
  - Benefits of performance isolation lead to more consistent performance compared to single OS
  - **Multi-OS/R approach can lead to better than native performance**

Thank You

- **Pisces Co-kernel Talk Tomorrow (2PM):**
  - **Jiannan Ouyang: Achieving Performance Isolation with Lightweight Co-kernels**

- Brian Kocoloski
  - briankoco@cs.pitt.edu
  - http://people.cs.pitt.edu/~briankoco
- Pointers to source
  - The Prognostic Lab @ U. Pittsburgh
  - http://www.prognosticlab.org