Kerrighed : An Overview

Renaud Lottiaux
Paris Team, Irisa
February 27, 2005

What is Kerrighed ?

- An **operating system** for cluster
  - Build as an **extension** to Linux
- Key idea : offering the view of a virtual SMP
  - Single System Image (SSI)
- Key features
  - Vision of an unique machine
  - High performance
  - Highly configurable
  - Transparency to users and applications

Vision of a unique machine

- One virtual CPU
  - Unique process space (pid)
  - Cluster wide scheduling
- One virtual memory
  - Support memory sharing between threads
  - Support System V memory segments
- One global file system (alpha version)
  - Unique file name space
  - Cluster wide disk mount

High Performance

- Target scientific applications
  - Sequential applications
  - Parallel applications
- High performance stream migration mechanism
  - Pipe, sockets, …
- Efficient software shared memory
- Target high performance networks
  - Gigabit Ethernet, Myrinet, Infiniband, Quadrix
- Cooperative file cache

Highly Configurable

- Customizable scheduling policy
  - New policies can be hot loaded in the kernel
- SSI features can be enable/disable on a per process basis
- Customizable data storage on disk
  - Redundant, non redundant
  - RAID 0, 1, ...

Outline

- Kerrighed Overview
  - What is Kerrighed ?
  - What about other system ?
  - Performance Evaluation
- Kerrighed Internal
  - Introduction
  - Ghosts
  - Containers
  - KerMM
- Conclusion

What about other projects ?

- Several projects exist
  - openMosix
  - OpenSSI
  - Kerrighed
  - B-Proc
  - DragonFly BSD
  - Genesis
  - Plurix
  - ...
- Who Kerrighed compares to others ?

OpenMosix

- Based on Mosix
  - Started in 1981 at University of Jerusalem
  - Port to Linux in 1999
  - OpenMosix “fork” in 2002
- Main features
  - Global process scheduling

OpenMosix Internals

- Process migration through deputy
  - A deputy remains on the process “home node”
  - Most system calls are redirected to the deputy
- Mosix FS (MFS)
  - Allow access to distant disks
- Direct File System Access (DFSA)
  - Enhance file system accesses through local accesses

OpenSSI

- Project started in 2001 at HP Labs
- Mainly aggregates existing softwares
  - NonStop Cluster for UnixWare
  - Cluster File System (CFS)
  - Cluster Infrastructure (CI Linux)
  - Distributed Lock Manager (DLM)
- Main features
  - Global process scheduling (load leveling)
  - Unique file system tree

OpenSSI Internals

- Process migration through daemon servers
  - A pool of daemons run on each nodes
  - Many system calls are redirected to deamons
- Cluster File System (CFS)
  - Unique file system tree
- Cluster Infrastructure (CI Linux)
  - Node addition and removal

Kerrighed

- Project started in 1999 (Gobelins) at IRISA (INRIA)
  - Written from scratch within Linux
- Project renamed Kerrighed in 2002
- Main features
  - Customizable global process scheduler
  - Shared memory support
  - Efficient stream migration
  - Unique file system tree

Kerrighed Internal

- Ghost Process
  - Process migration, checkpointing, remote creation
- Scheduling policy writing framework
  - Ease creation of new policies, hot-pluggable
- Container
  - Shared memory, cooperative file cache, ...
- Dynamic global stream
  - Efficient migration of sockets, pipes, ...

View of a Unique Machine

	Kerrighed	OpenMosix	OpenSSI
Unique PID space	✔	✔ / ✘	✔
Global PS	✔	✔ / ✘	✔
Global Top	✔	✔ / ✘	✘
Global /dev	✘	✘	✔
Unique FS tree	✔ / ✘	✔ / ✘	✔

Global Process Management

	Kerrighed	OpenMosix	OpenSSI
Process migration	✔	✔	✔
Individual thread migration	✔	✘	✘
Threaded application migration	✔	✘	✔
Global process scheduler	✔	✔	✔
Customizable process scheduler	✔	✘	✘

IPC Migration Support

	Kerrighed	OpenMosix	OpenSSI
System V memory segment	✔	✘	✔ / ✘
System V semaphores	✘	✔	✔
Pipe	✔	✔	✔
Unix Socket	✔	✔	✔
INET Socket	✔	✔	✔

Fault Tolerance and Checkpointing

	Kerrighed	OpenMosix	OpenSSI
Hot node addition	✘	✔	✔
Hot node removal	✘	✔	✔
Tolerance to node failure	✘	✔ / ✘	✔ / ✘
Process Checkpoint	✔	✔	✘
Threaded application checkpoint	✘	✘	✘
Message passing application checkpoint	✘	✘	✘

Outline

- Kerrighed Overview
  - What is Kerrighed ?
  - What about other system ?
  - Performance Evaluation
- Kerrighed Internal
  - Introduction
  - Ghosts
  - Containers
  - KerMM
- Conclusion

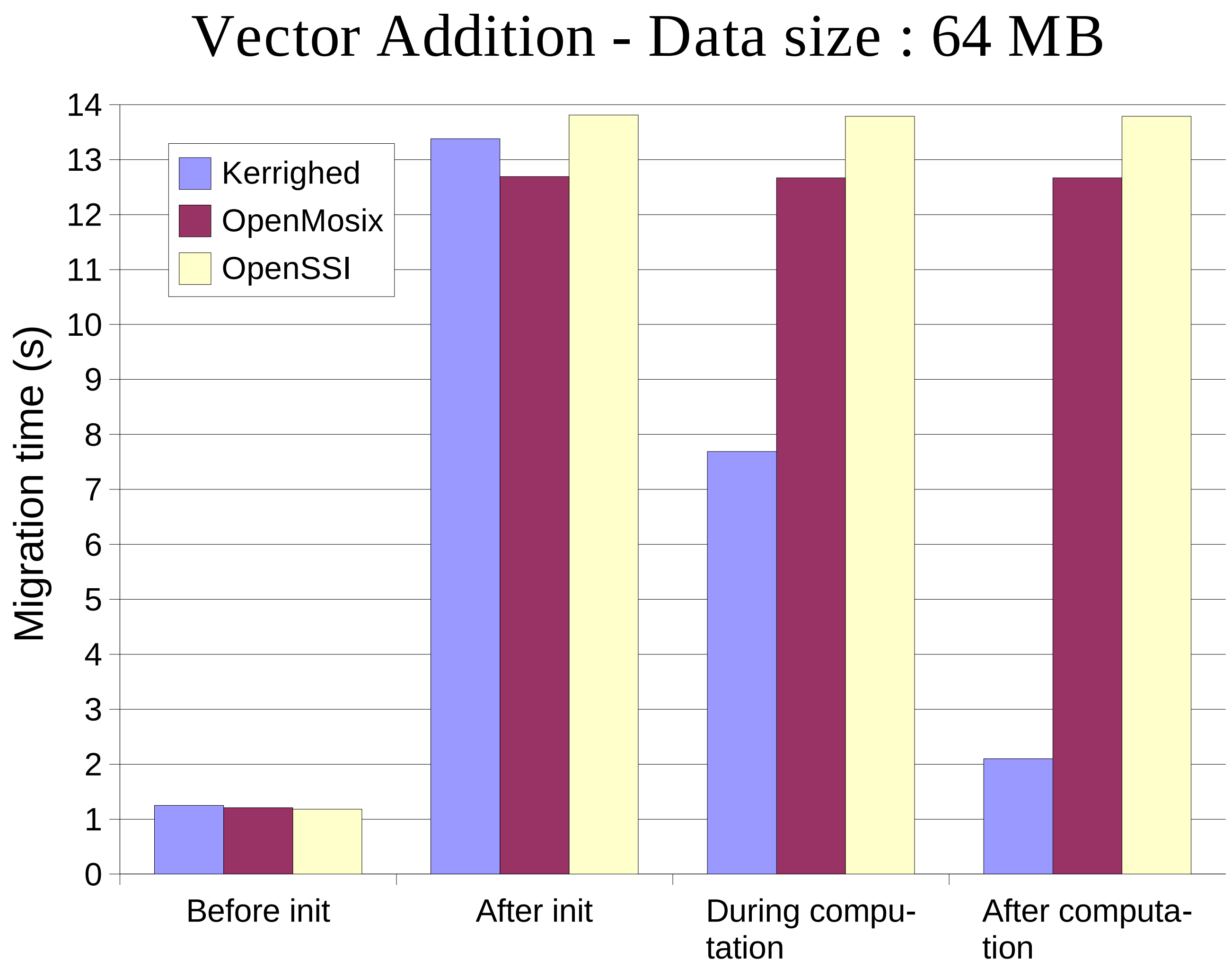
Performance evaluation

- Experimental platform
  - 4 nodes cluster
  - Intel Pentium III 1Ghz
  - 512 MB main memory
  - Fast Ethernet
- Tested systems
  - Kerrighed 1.0-rc9 (kernel 2.4.24)
  - OpenMosix 2.4.22-3 (kernel 2.4.22)
  - OpenSSI 1.0.0-rc5 (kernel 2.4.20)

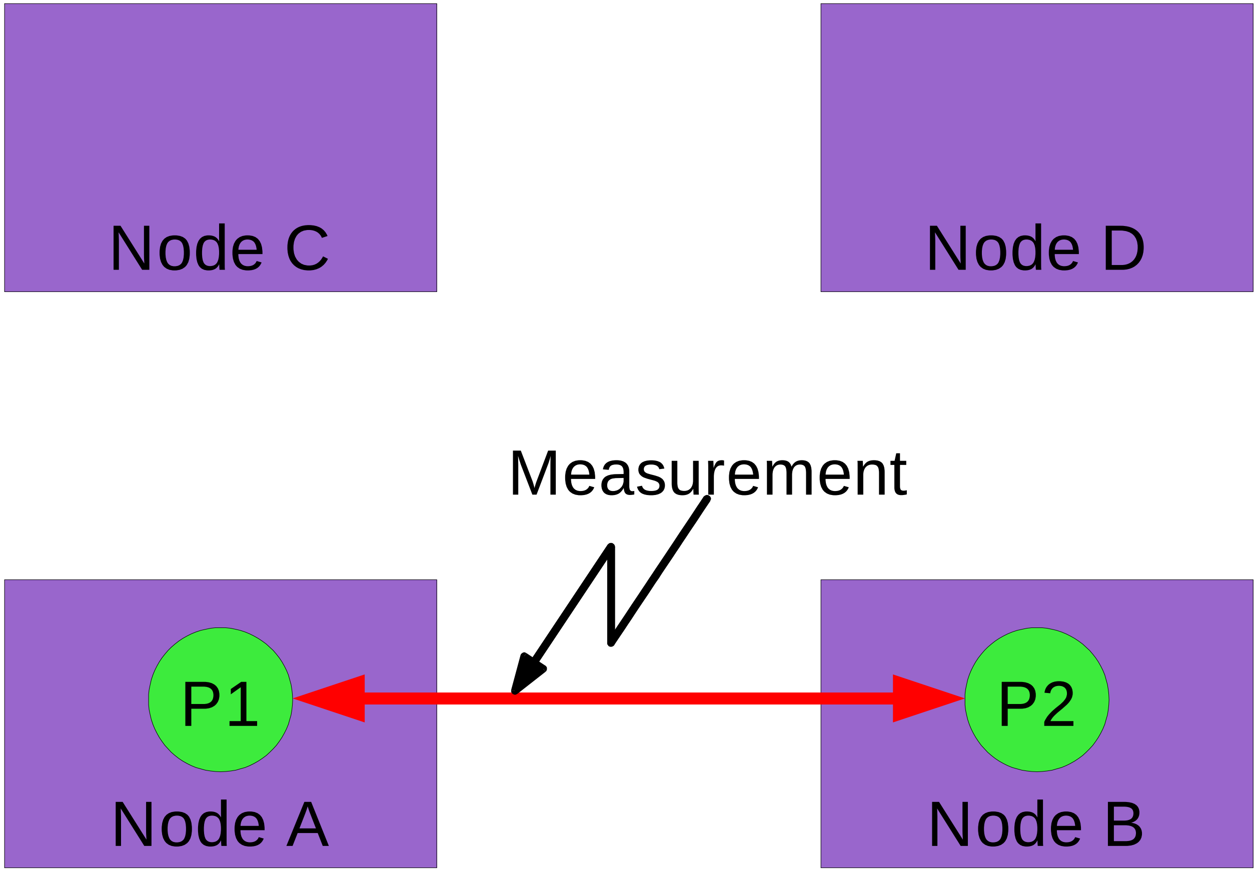
Process Migration

- Vector addition
- Data size
  - 64 MB / vector
- Migrate at different execution time
  - Before vector initialization
  - Before computation
  - During computation
  - End of computation
- Compute the overhead

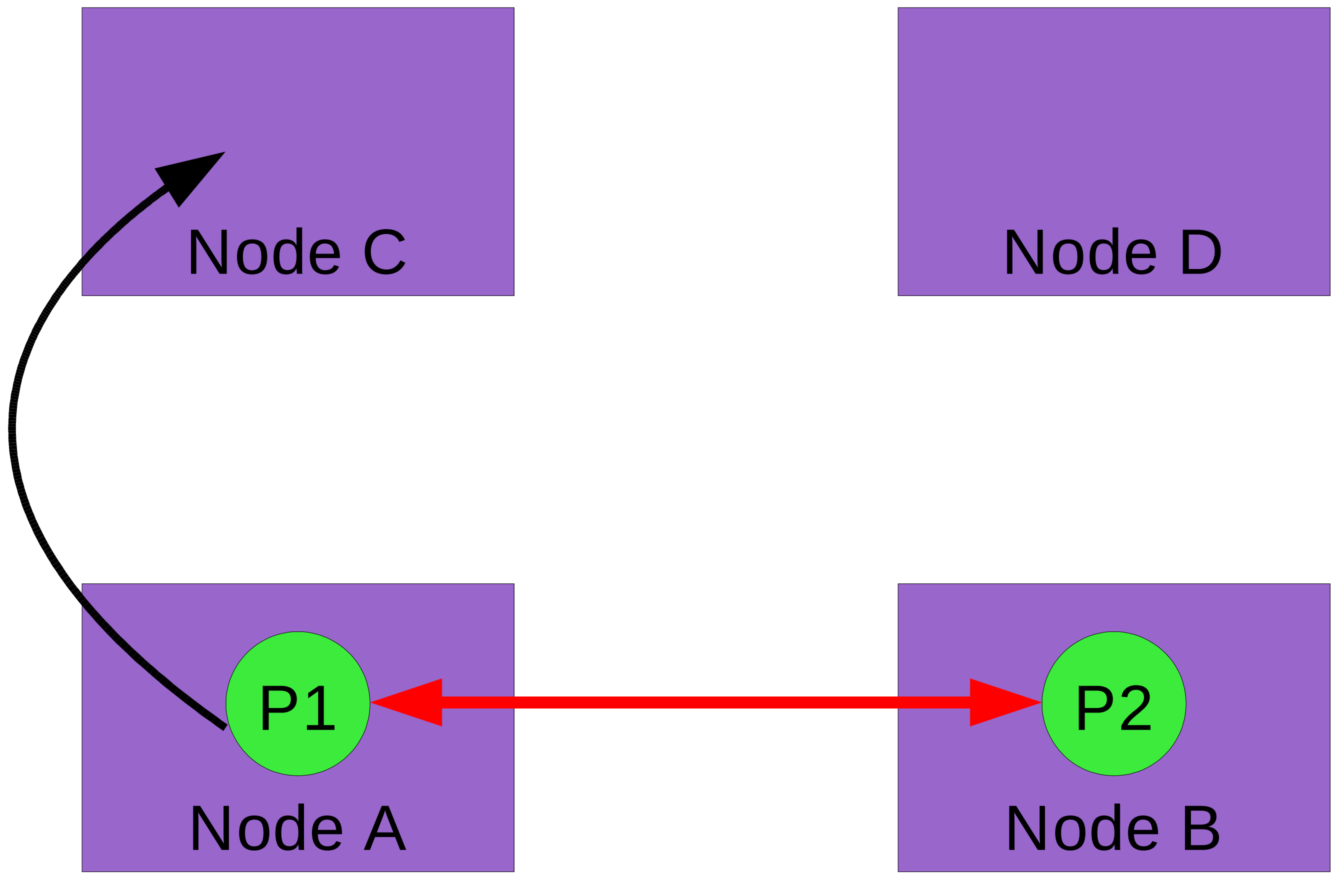
Process Migration Overhead



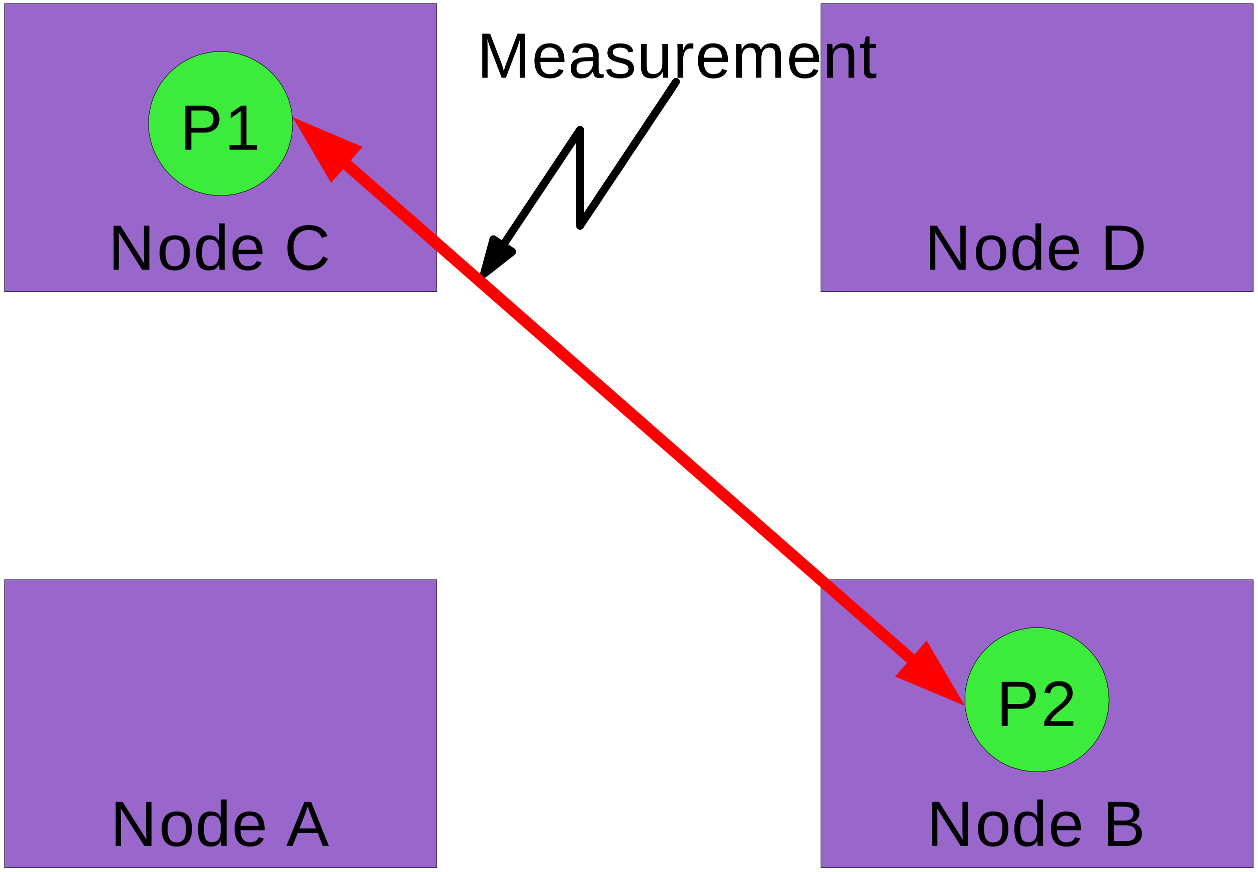
Socket Migration (Case 1)



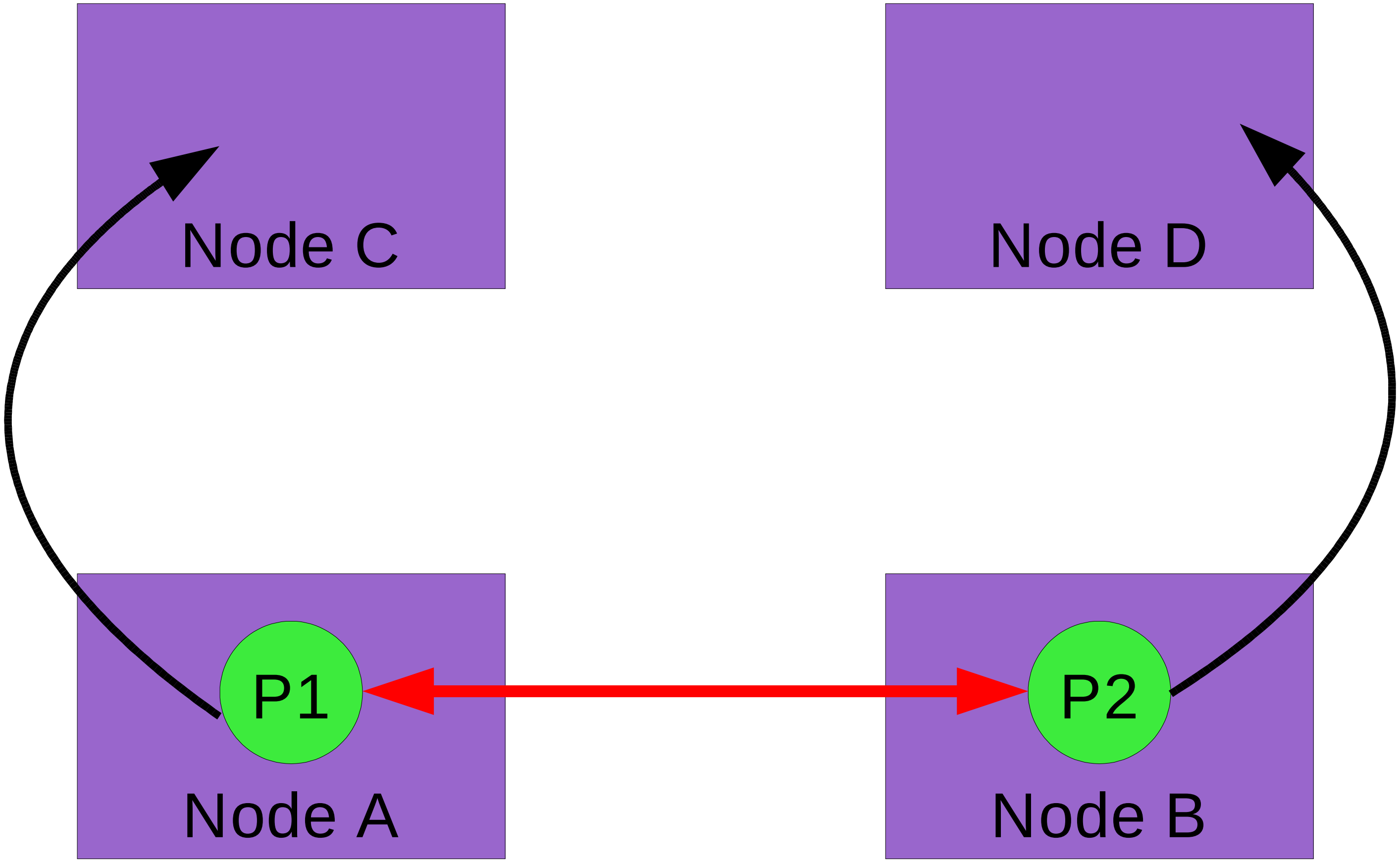
Socket Migration (Case 2)



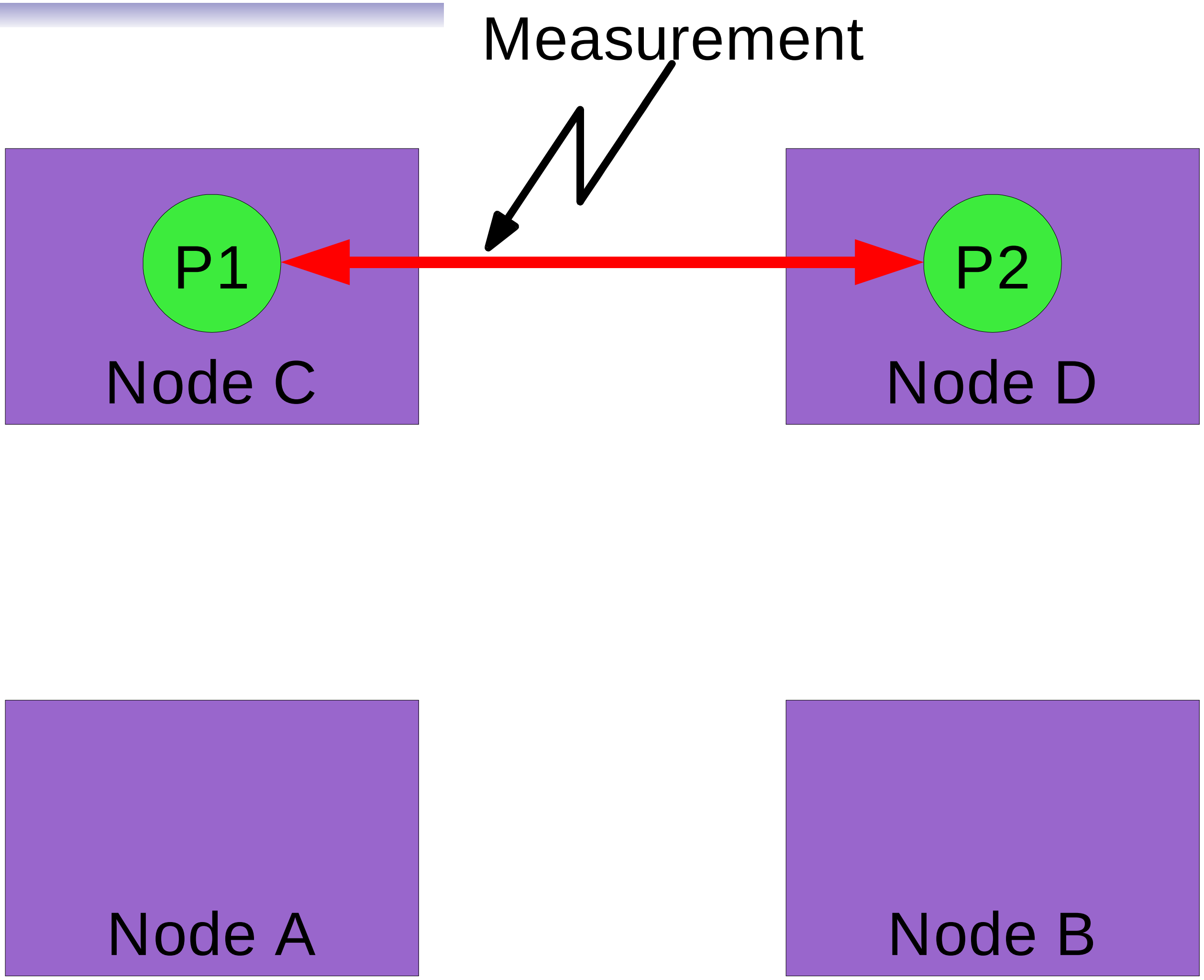
Socket Migration (Case 2)



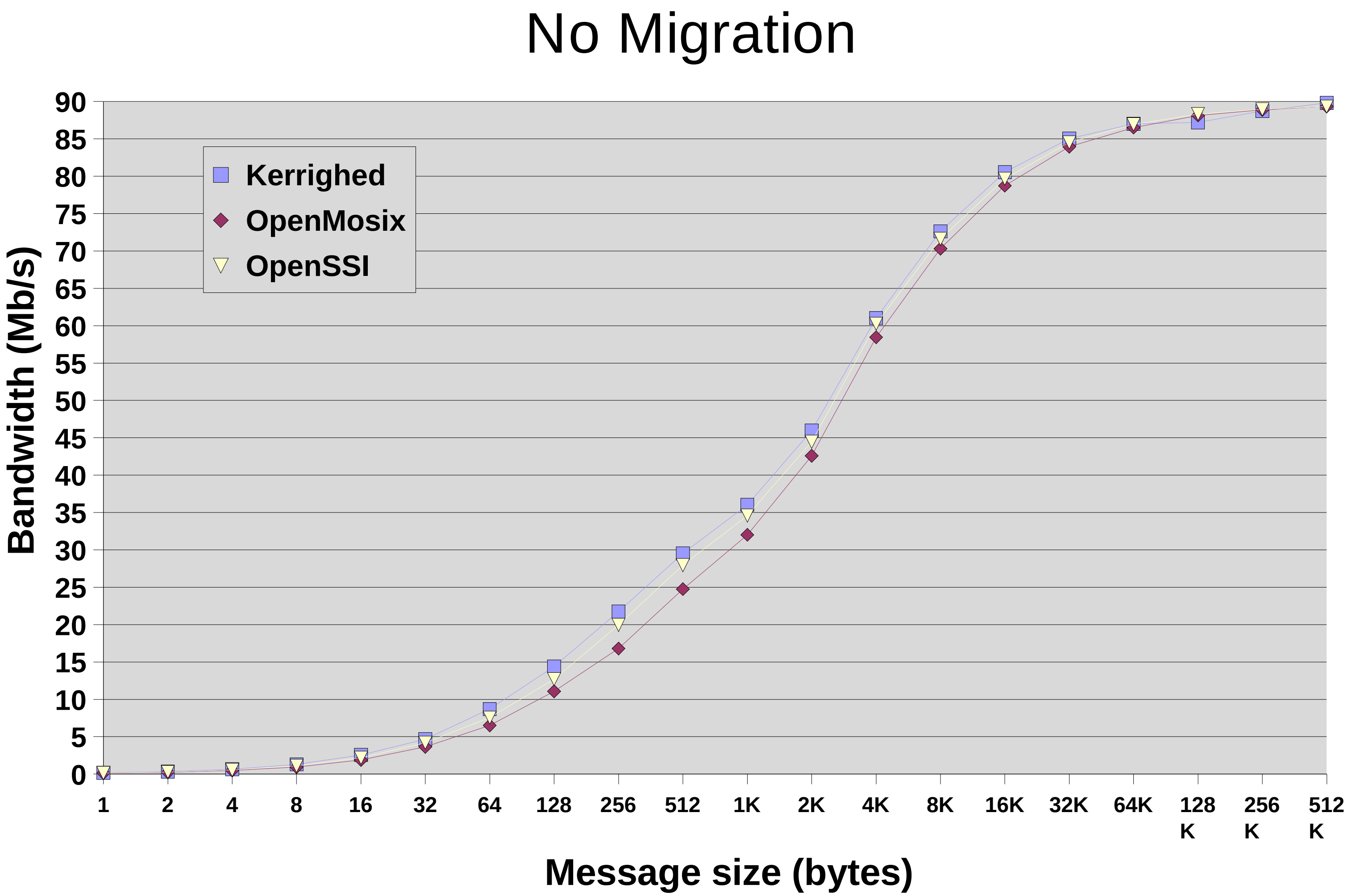
Socket Migration (Case 3)



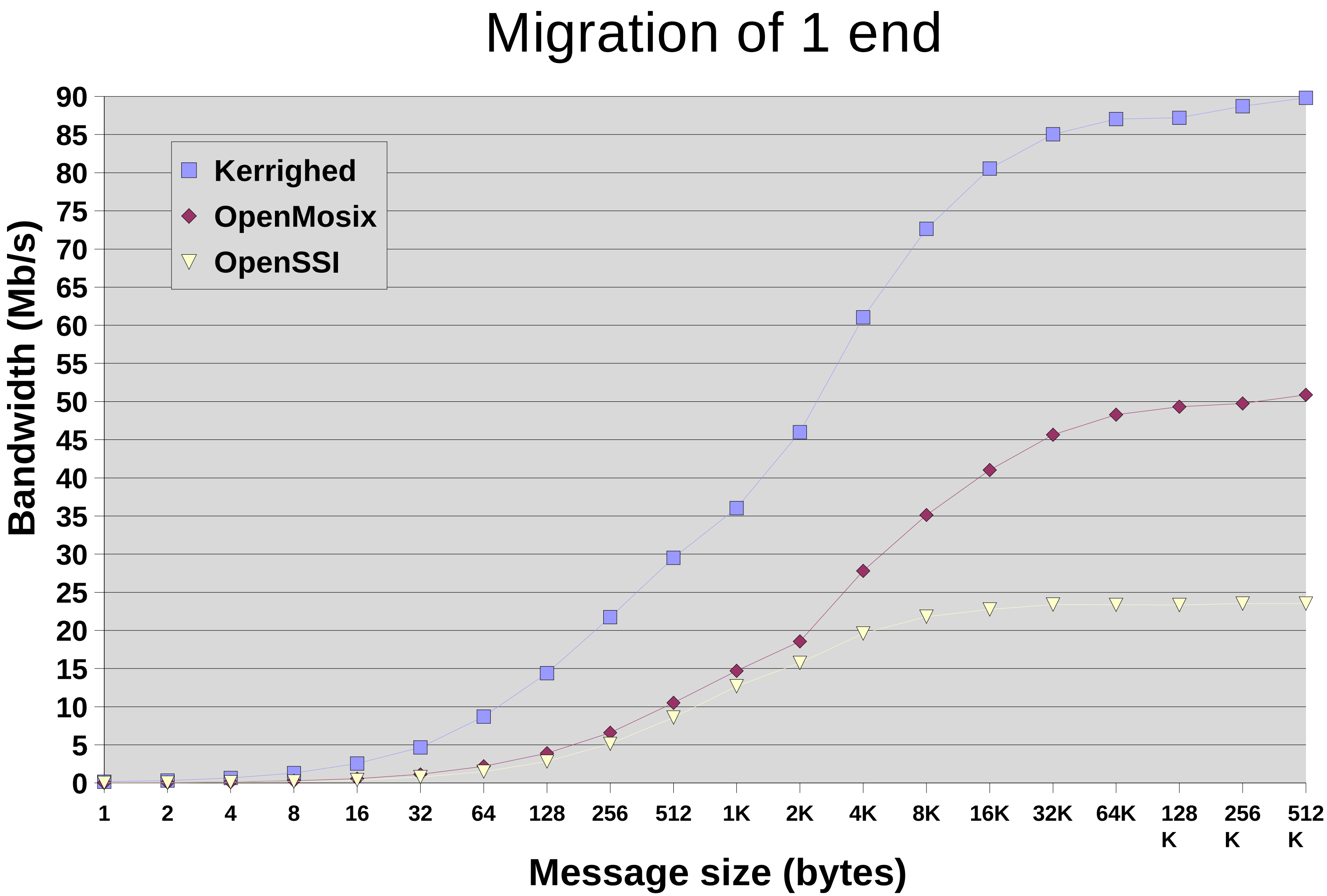
Socket Migration (Case 3)



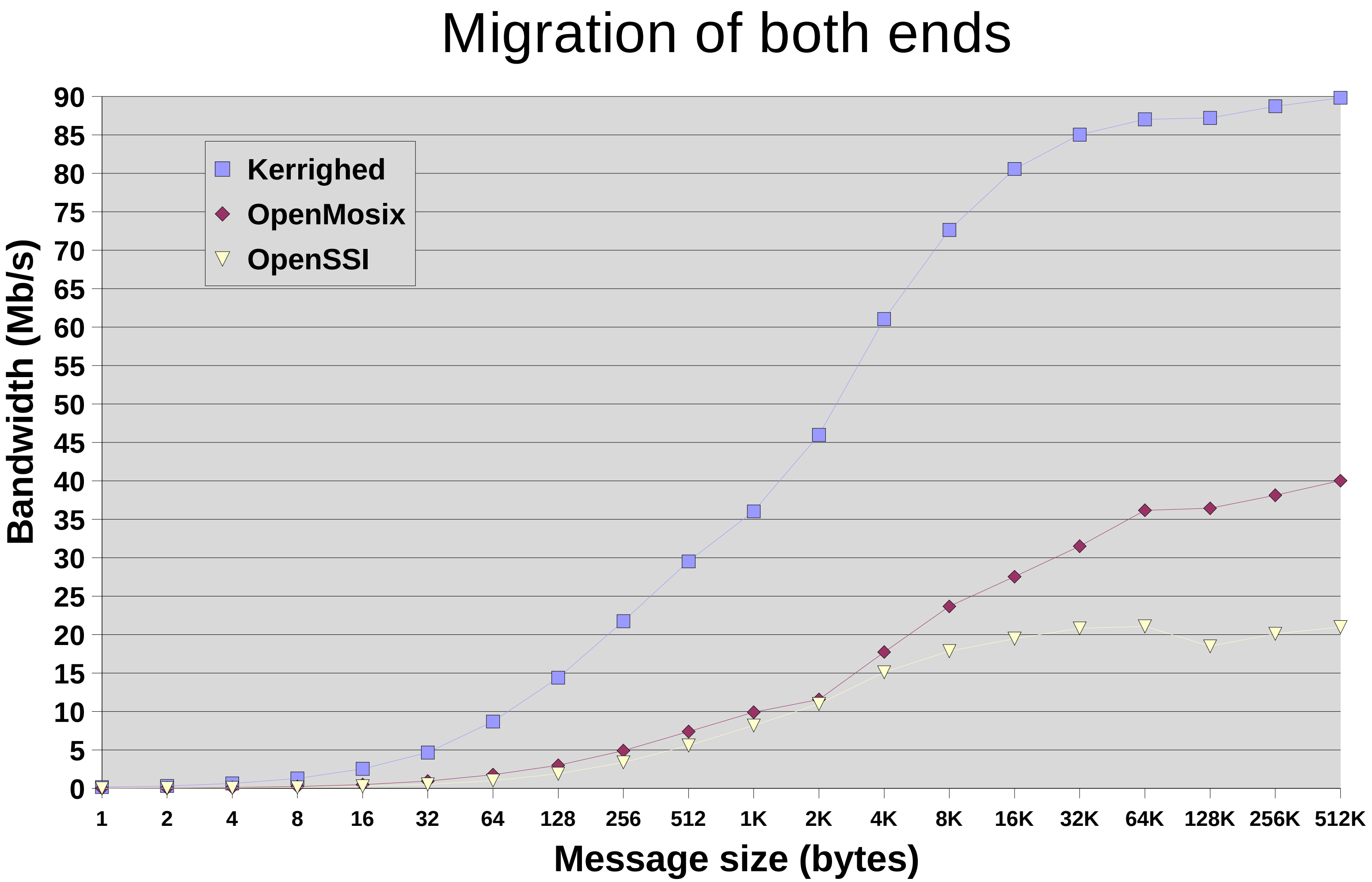
TCP Socket Bandwidth (Case 1)



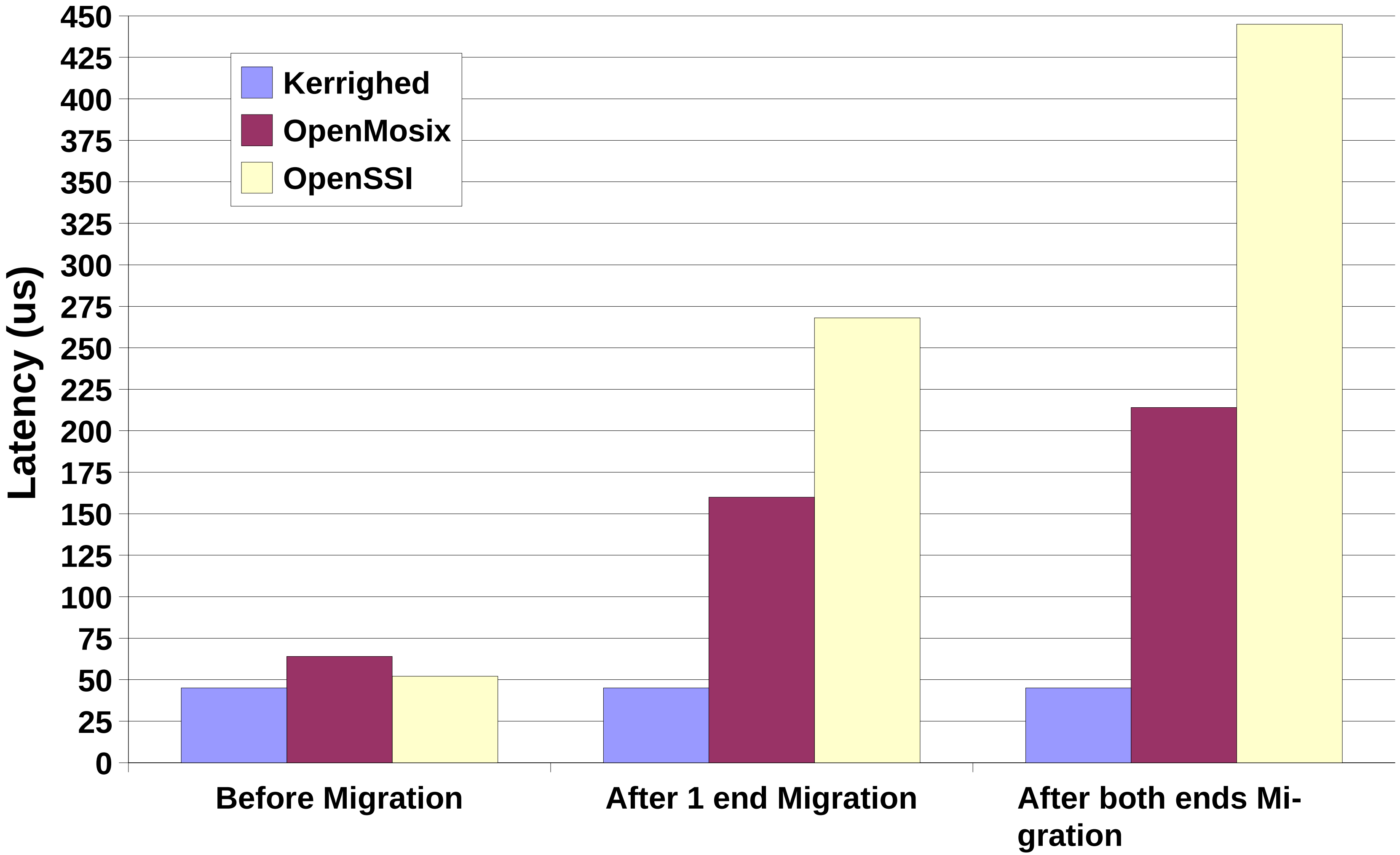
TCP Socket Bandwidth (Case 2)



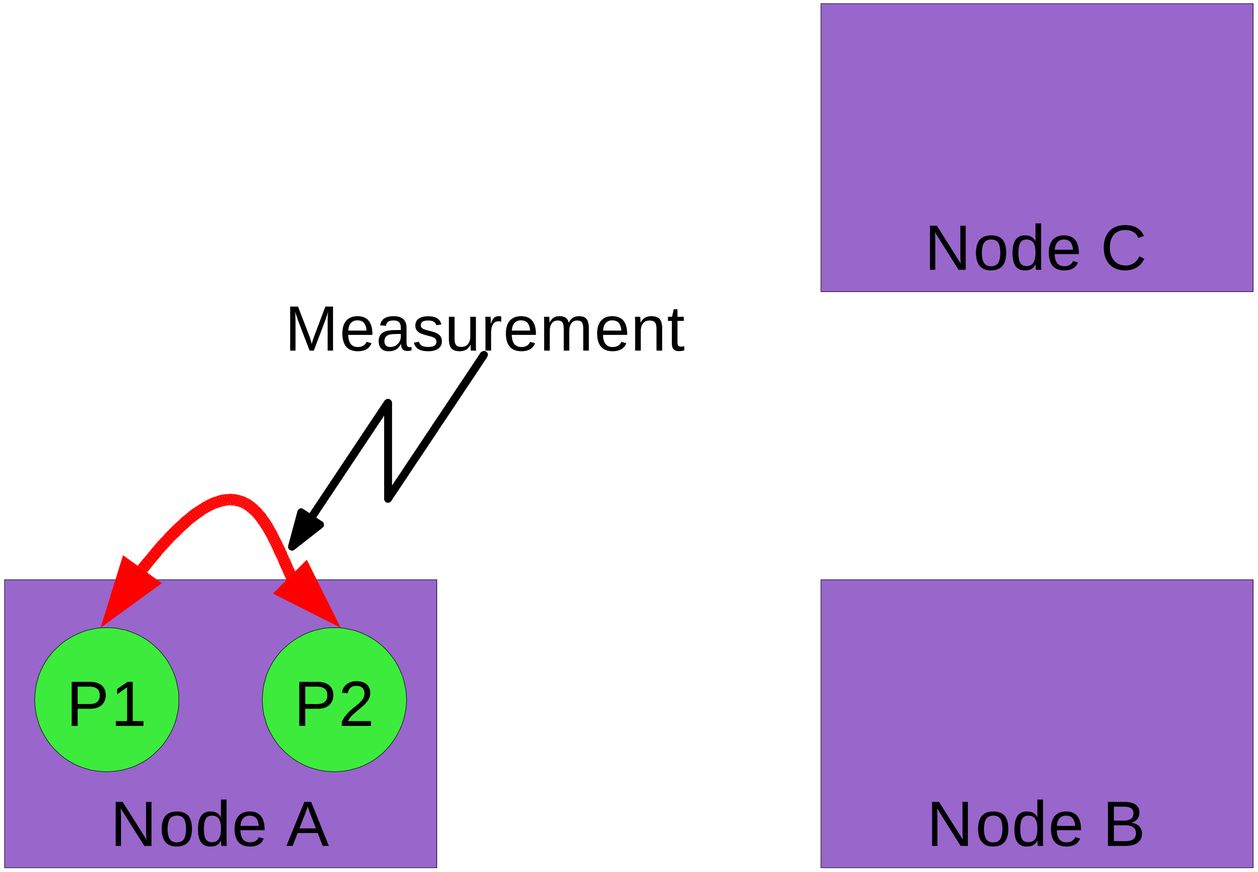
TCP Socket Bandwidth (Case 3)



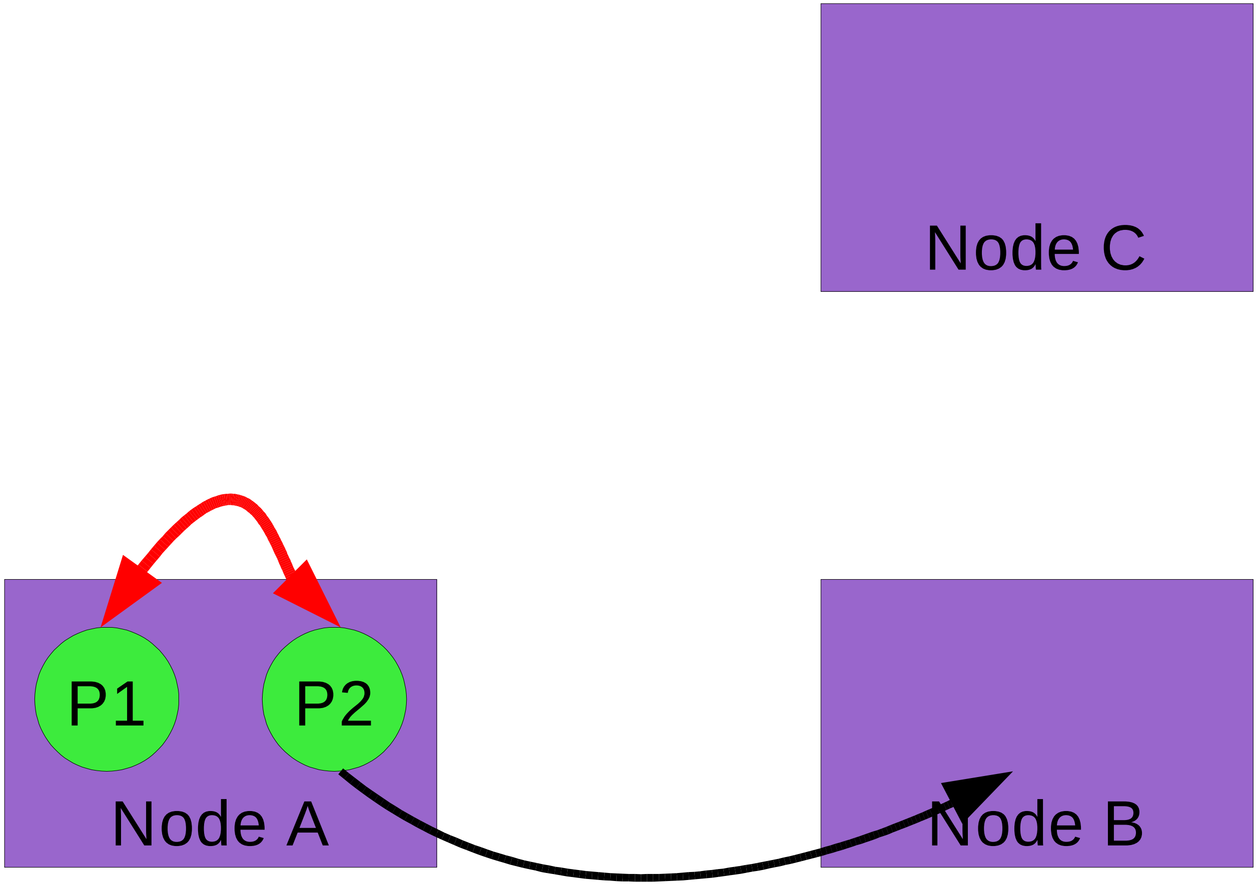
TCP Socket Latency



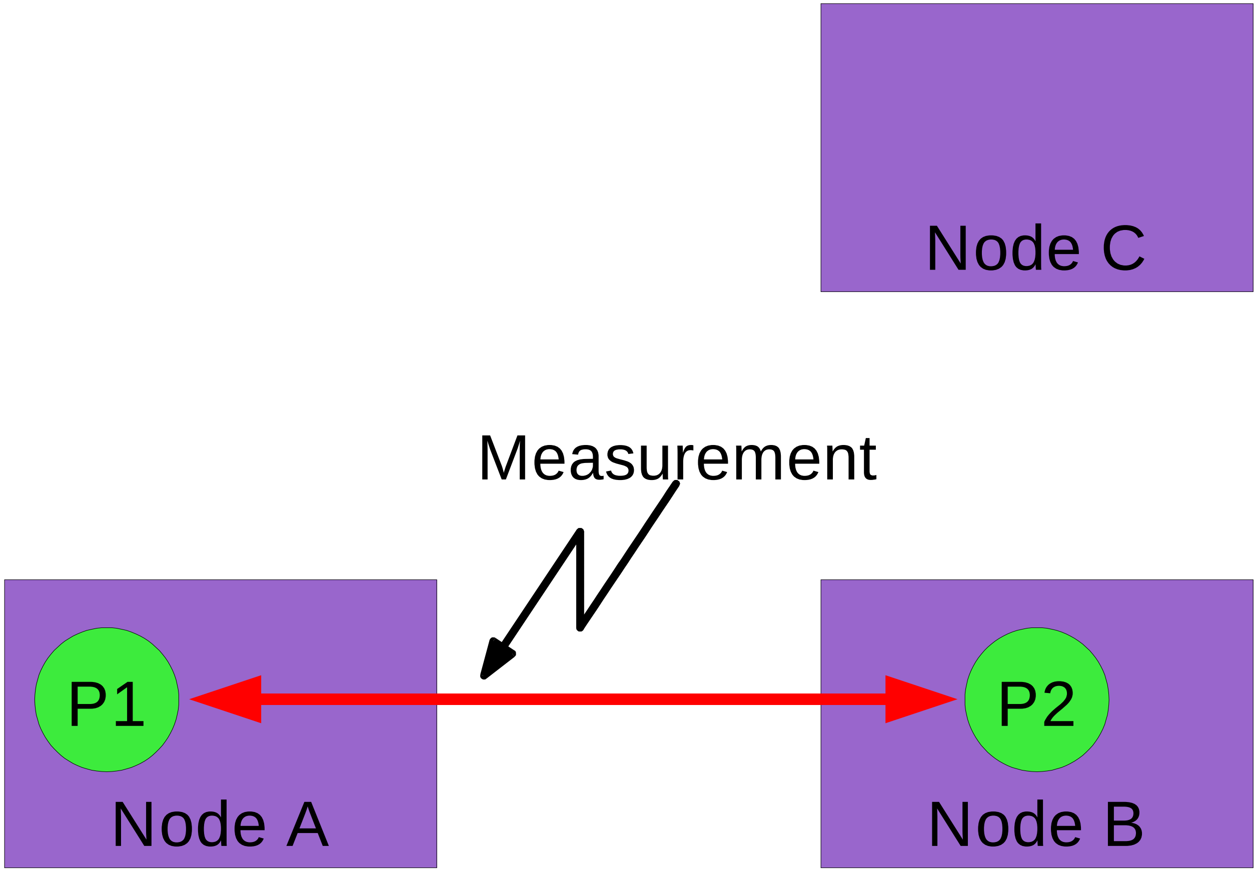
Pipe Migration (Case 1)



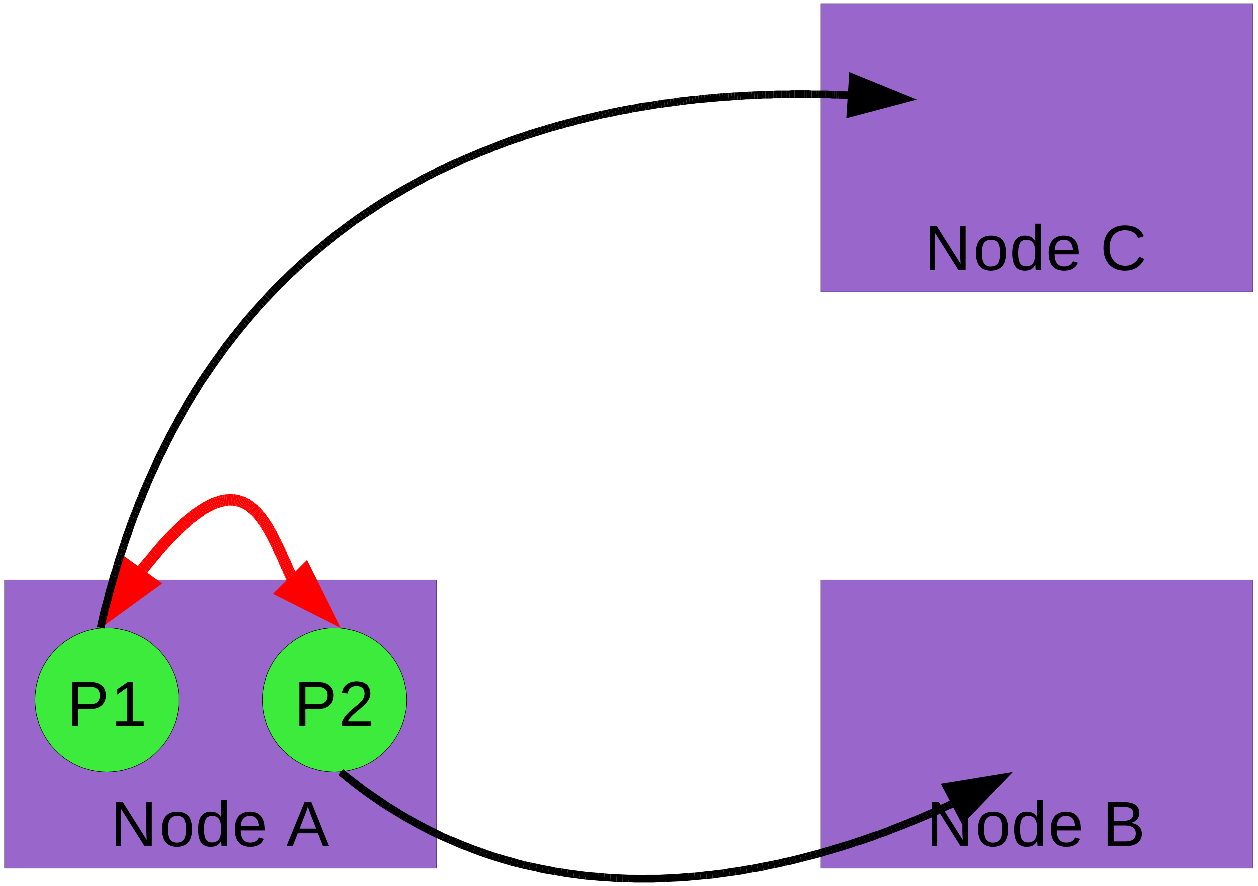
Pipe Migration (Case 2)



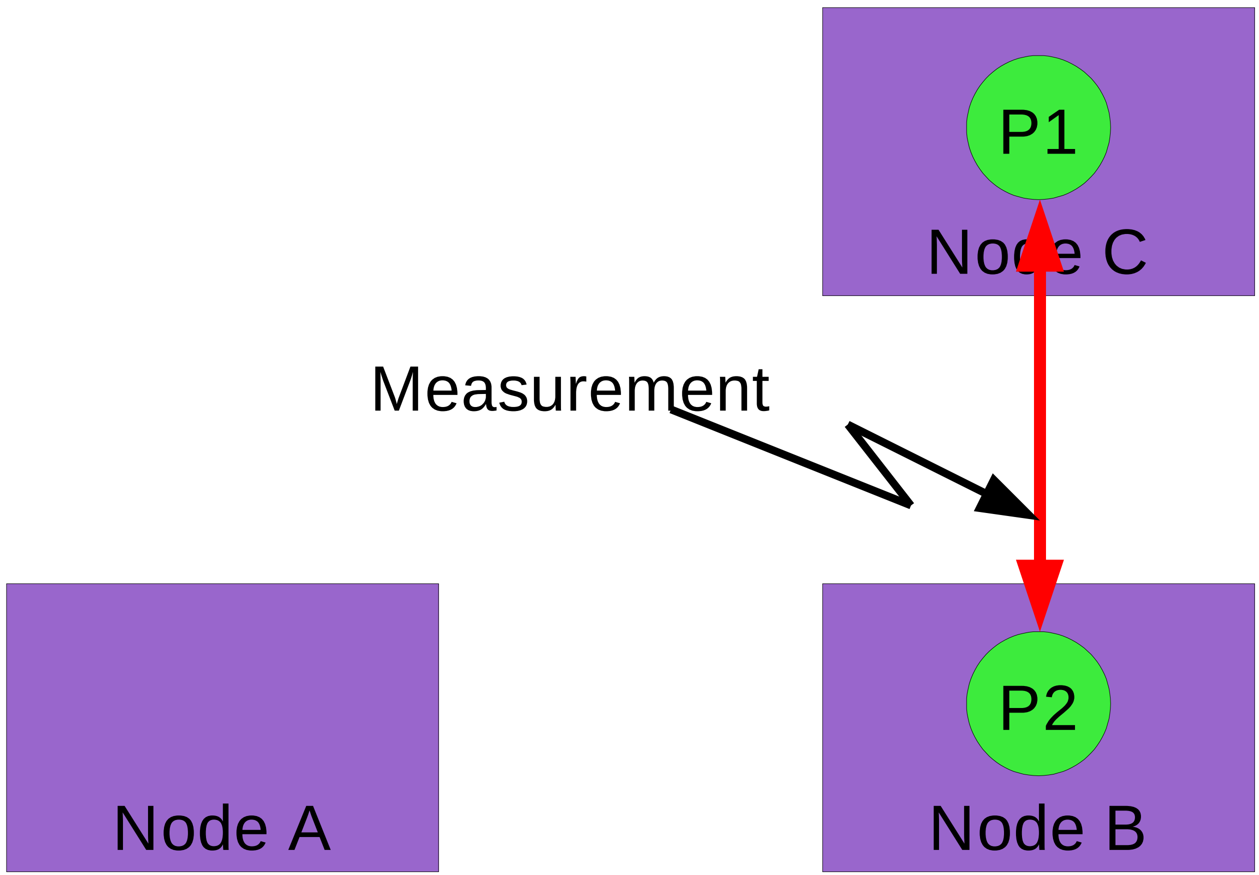
Pipe Migration (Case 2)



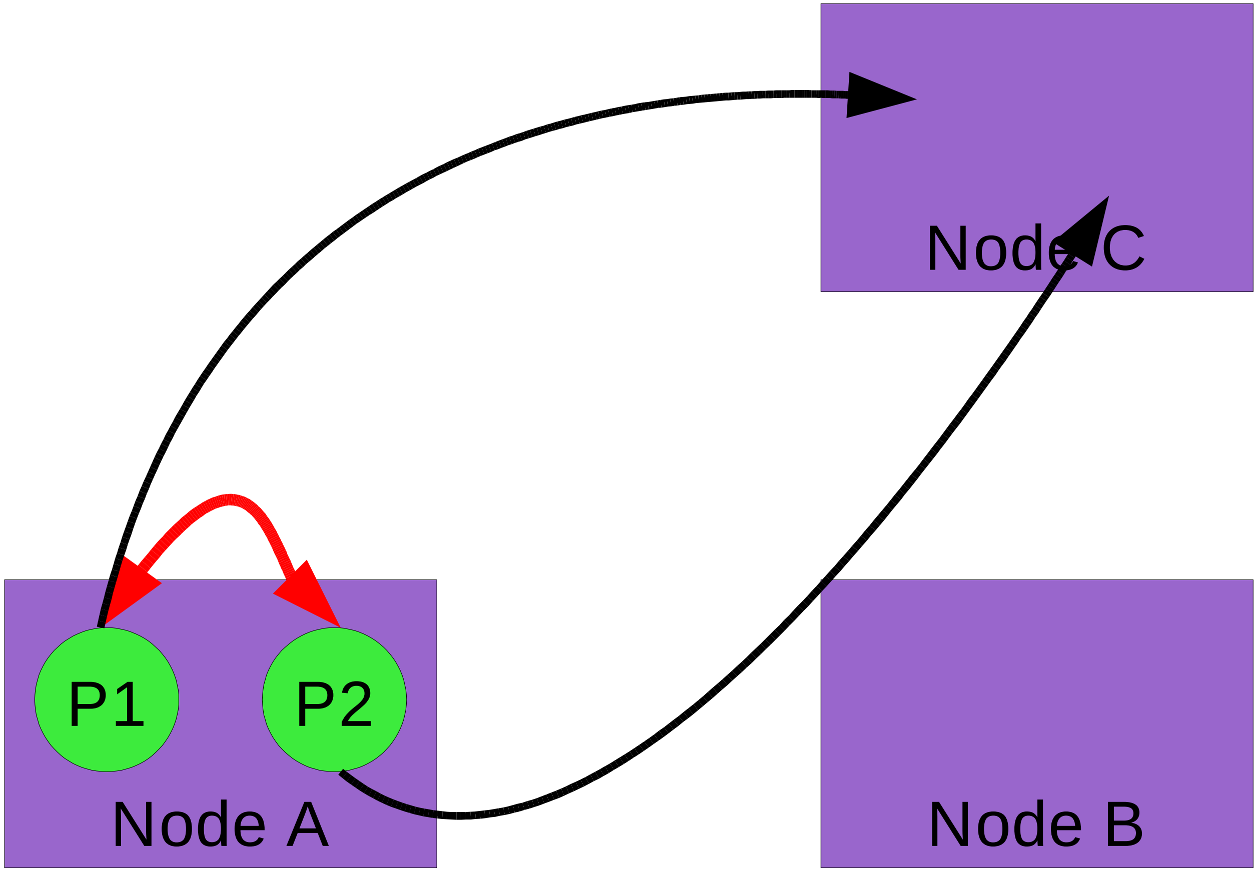
Pipe Migration (Case 3)



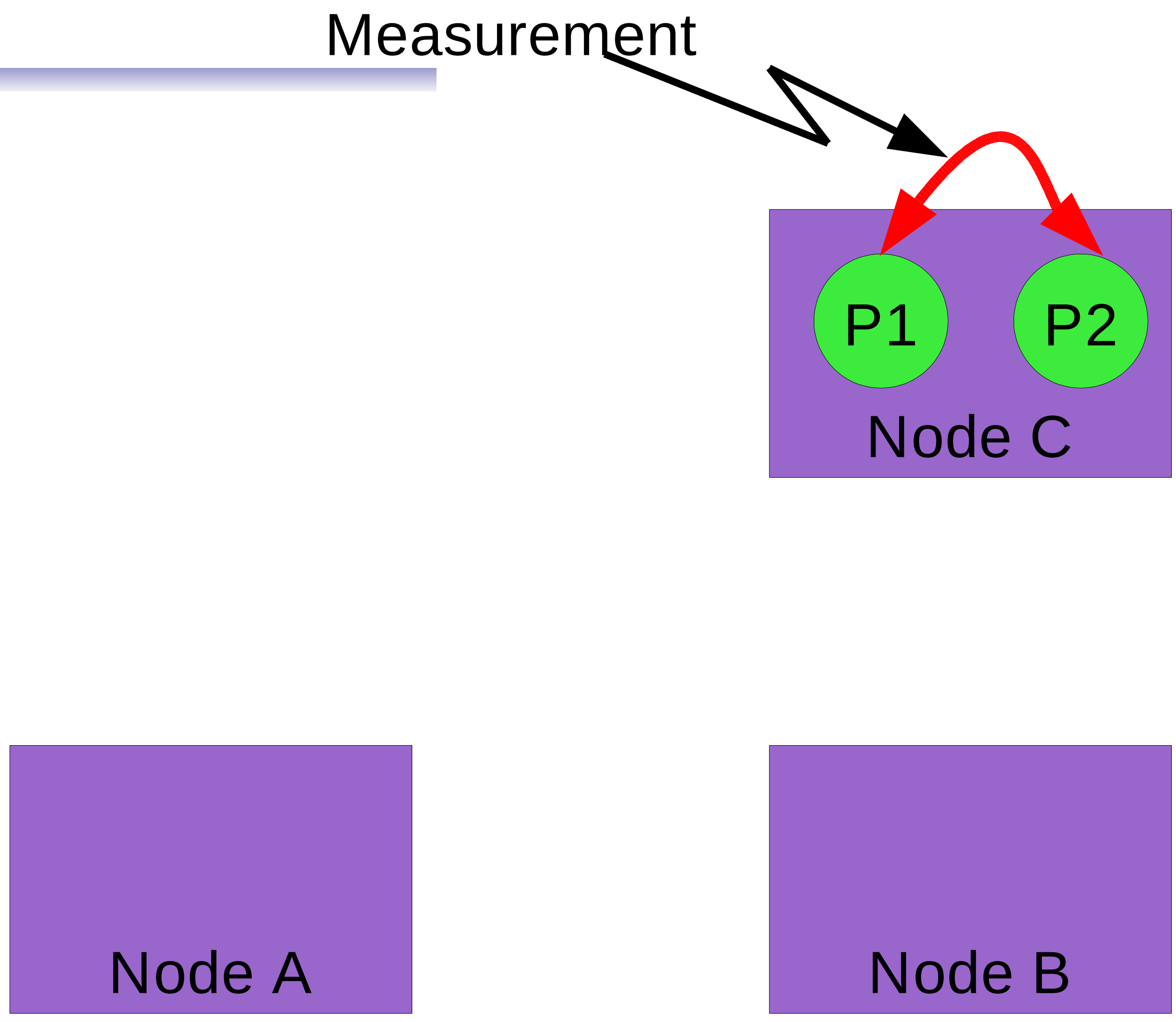
Pipe Migration (Case 3)



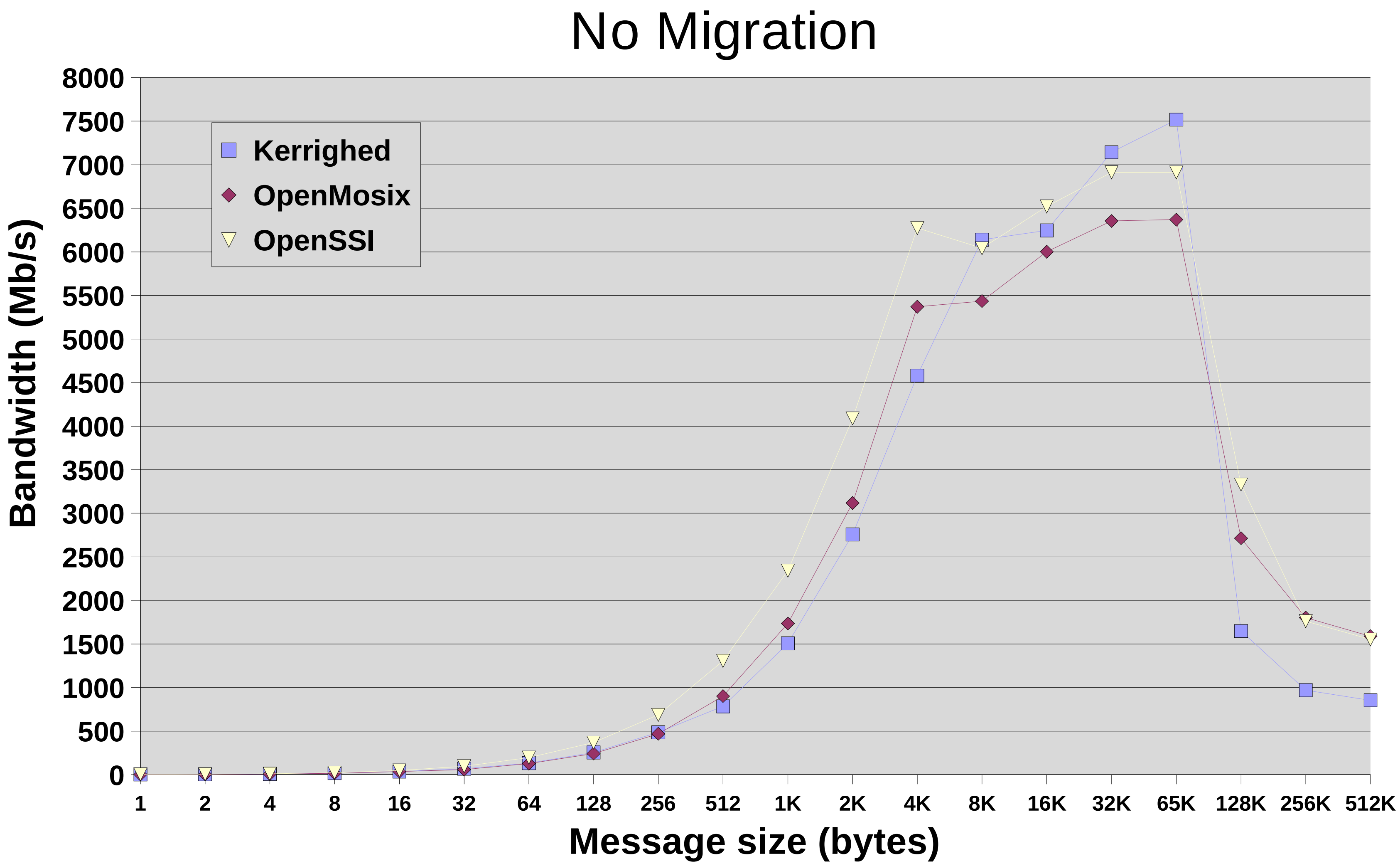
Pipe Migration (Case 4)



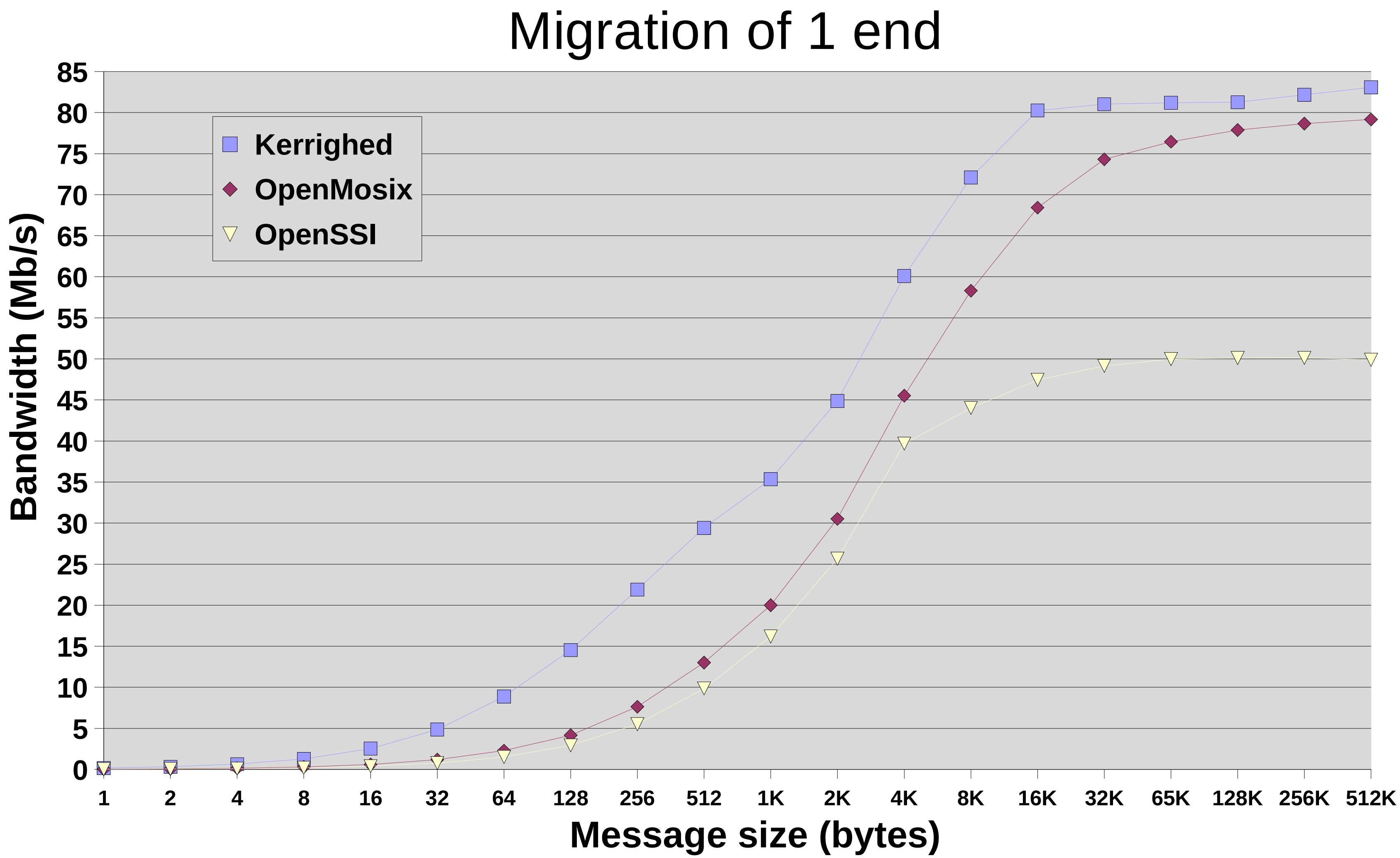
Pipe Migration (Case 4)



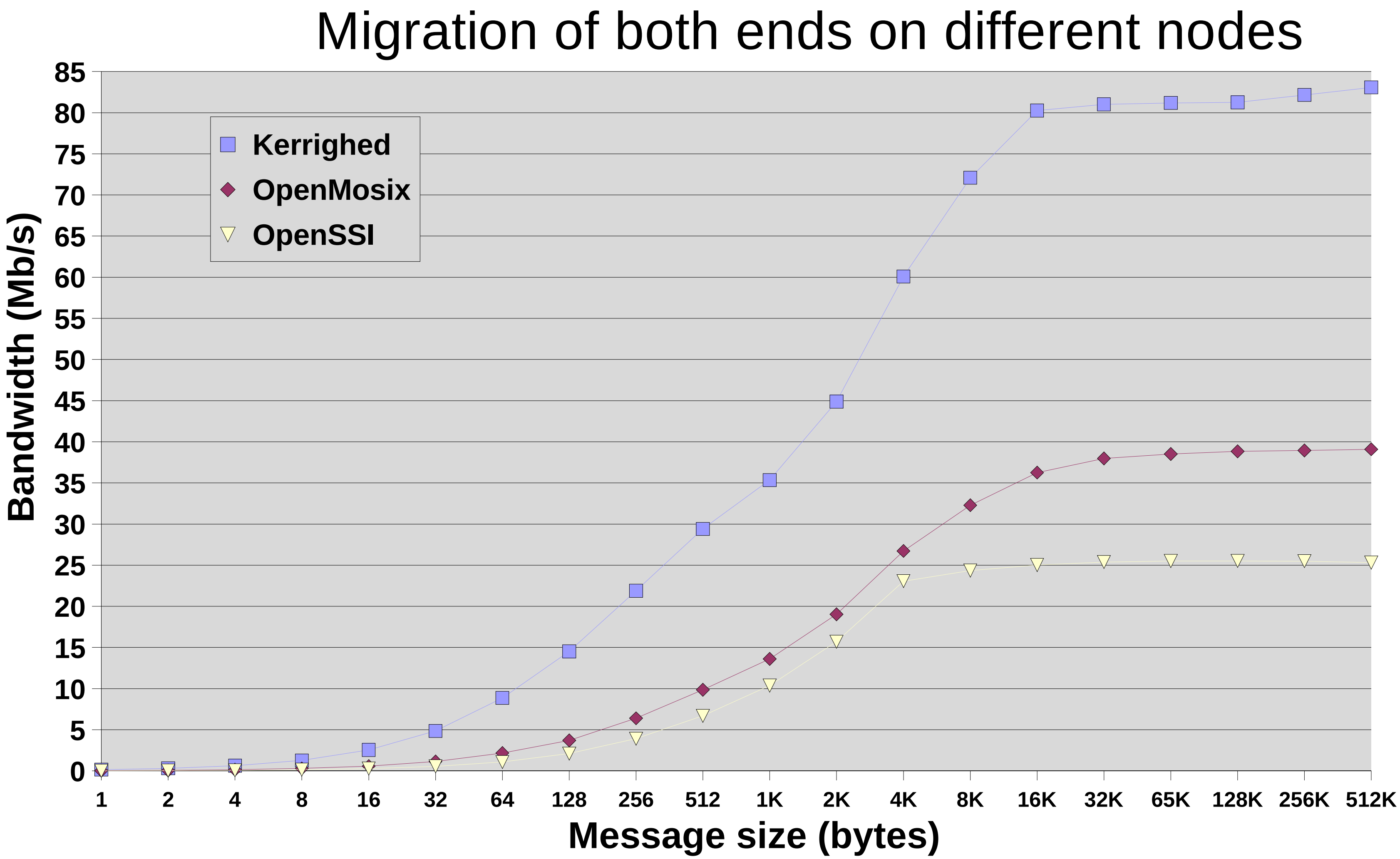
Pipe Bandwidth (Case 1)



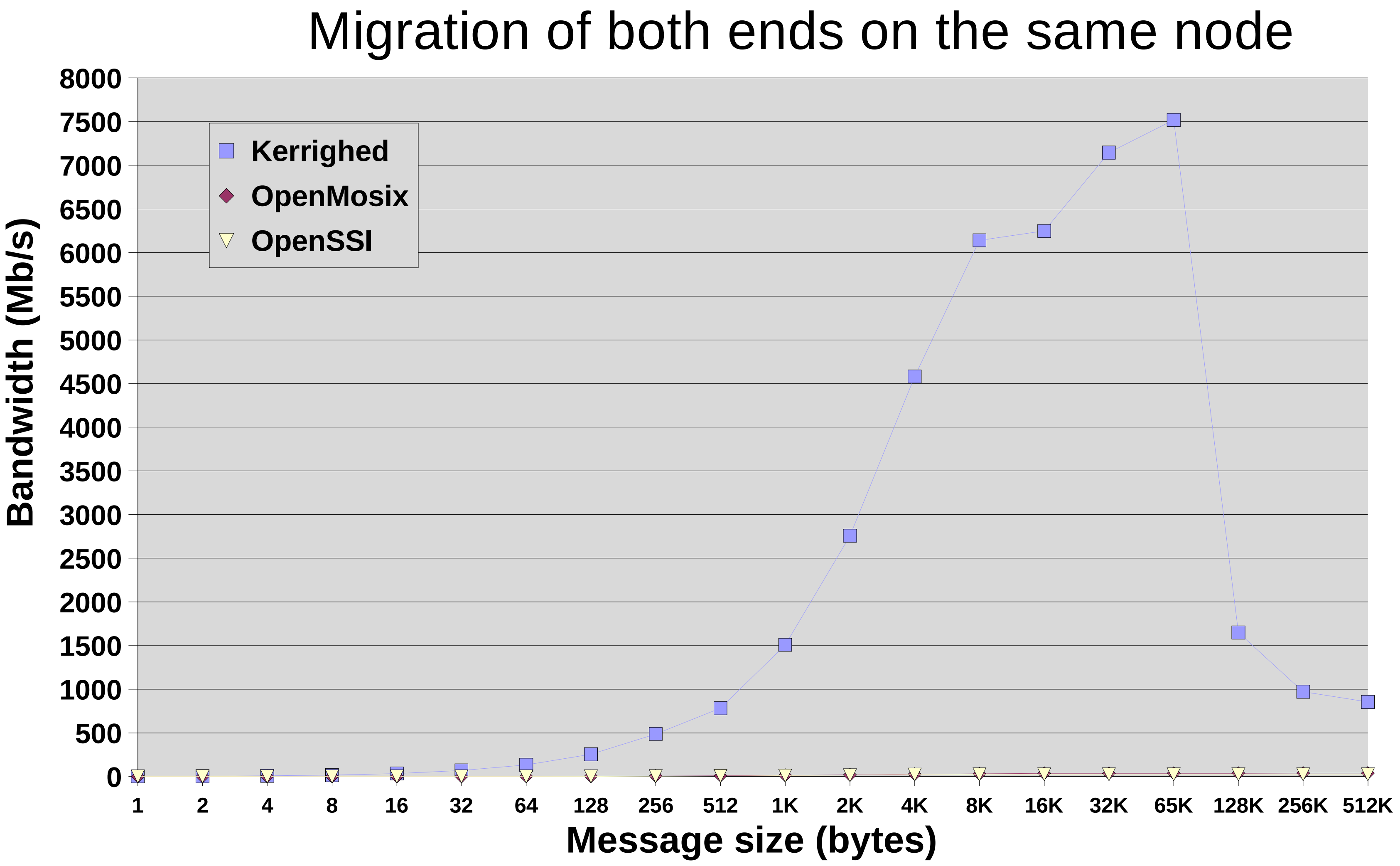
Pipe Bandwidth (Case 2)



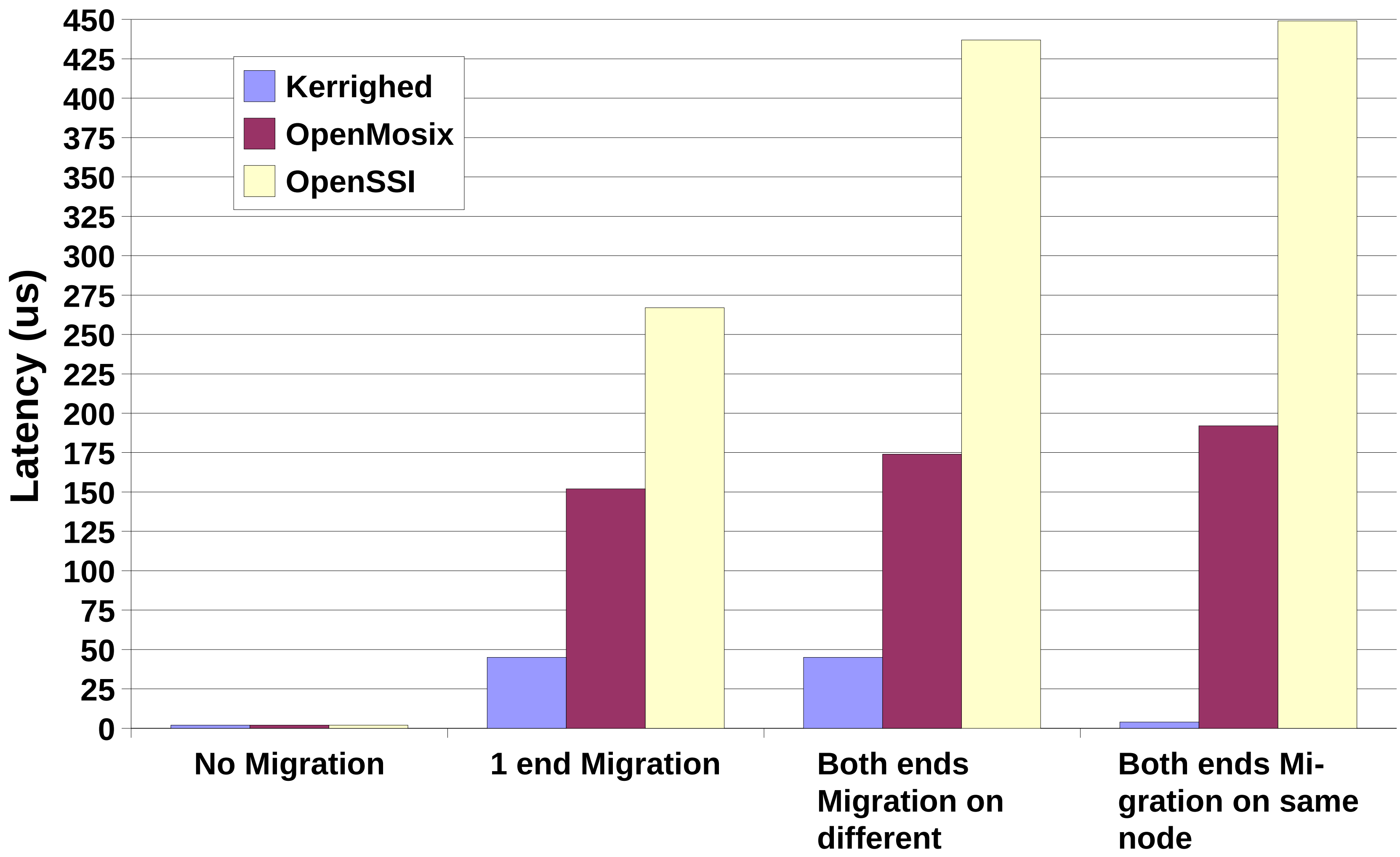
Pipe Bandwidth (Case 3)



Pipe Bandwidth (Case 4)



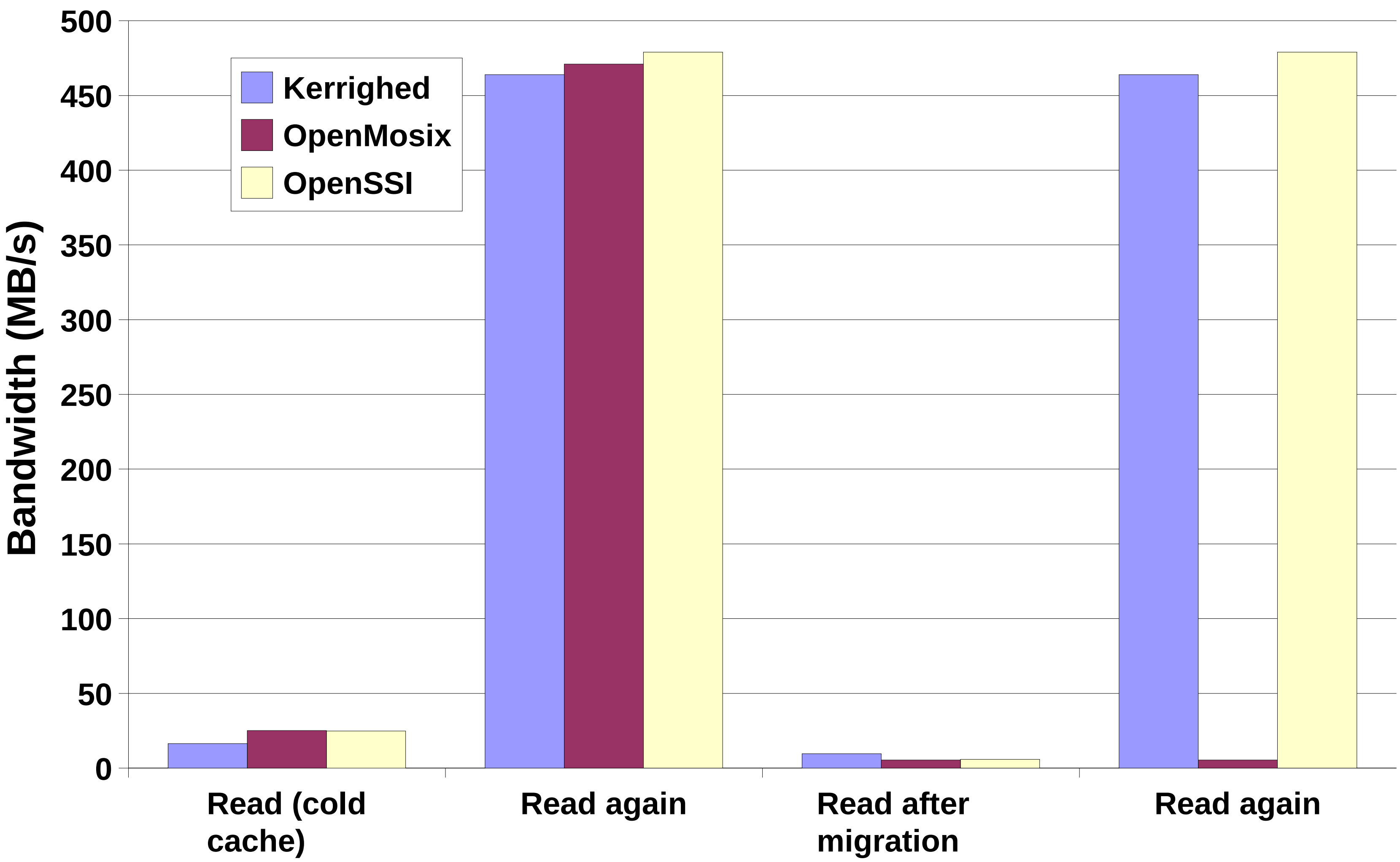
Pipe Latency



File Read

- 10 MB file
- Sequential read
- On node A
  - Read the file (cold cache)
  - Read again the file
- Migration to node B
- On node B
  - Read the file (cold cache)
  - Read again the file

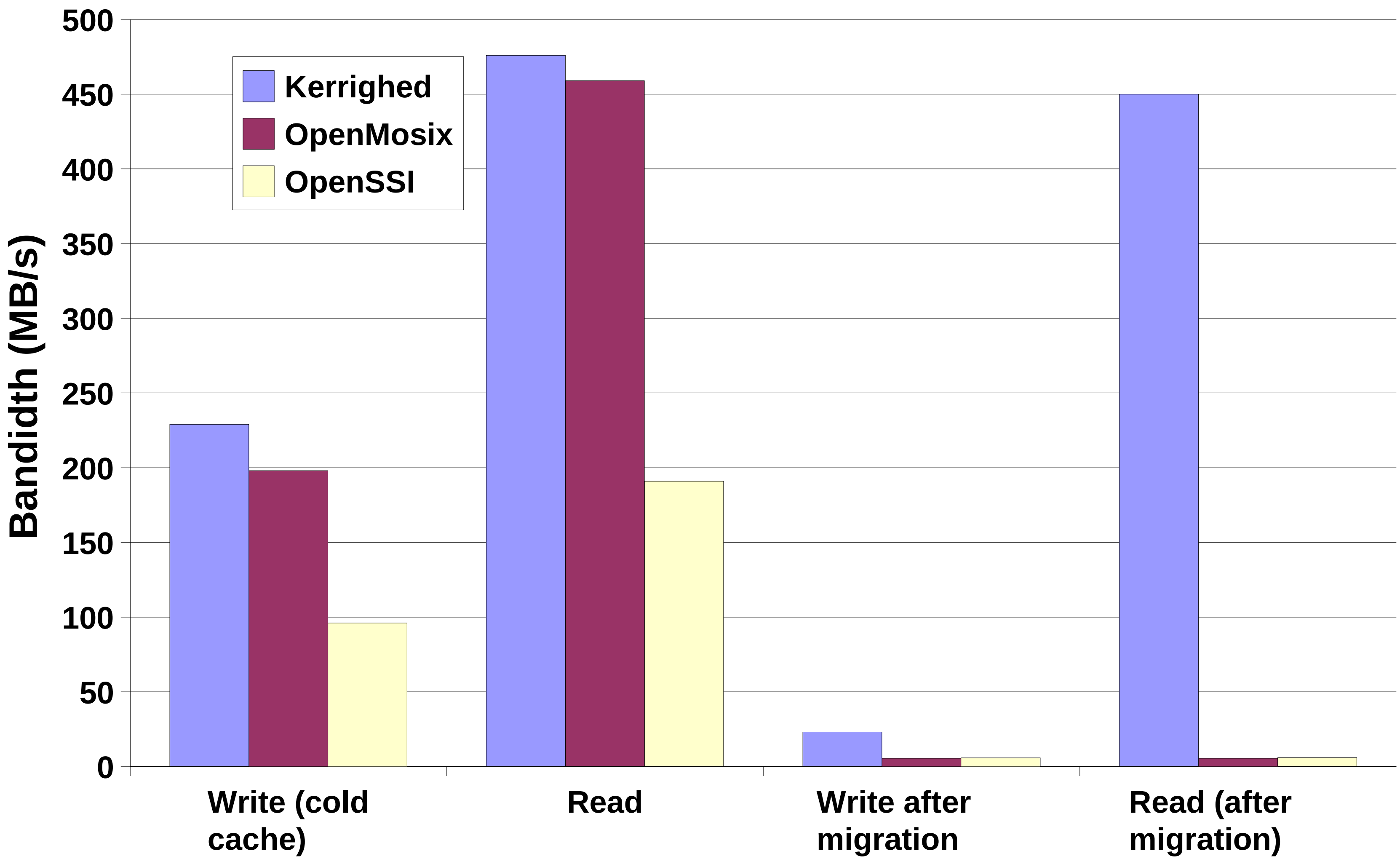
File Read: Results



File Write

- 10 MB file
- Sequential write : read
- On node A
  - Write to the file (cold cache)
  - Read the written data
- Migration to node B
- On node B
  - Over-write to the file (cold cache)
  - Read the written data

File Write : Results



Summary

- Kerrighed
  - ✔ Performance
  - ✔ Real view of a unique SMP machine
  - ✔ Highly customizable
  - ✔ Good internal software architecture
  - ✗ Stability
  - ✗ No support for node addition/removal/failure
  - ✗ No support for SMP / 64 bits architectures
  - ✗ Very small community

Summary (2)

- OpenSSI
  - ✔ Real view of a unique SMP machine
  - ✔ Support from HP
  - ✔ Robustness
  - ✗ Performance
  - ✗ Poor internal software architecture
  - ✗ Small community

Summary (3)

- OpenMosix
  - ✔ Very good global process scheduler
  - ✔ Large user community
  - ✔ Relative robustness
  - ✔ Scale very well
  - ✗ Performance
  - ✗ Incomplete view of an SMP machine

Kerrighed Future Works

- Node addition/removal (april 2005)
- SMP / 64 bits (july 2005)
- Distributed/Parallel file system (july 2005)
- 2.6 port (july 2005)
- Parallel checkpoint (? 2005)
- High availability (2006)
- Grid extensions (?)

Outline

- Kerrighed Overview
  - What is Kerrighed ?
  - What about other system ?
  - Performance Evaluation
- Kerrighed Internal
  - Introduction
  - Ghosts
  - Containers
  - KerMM
- Conclusion

Previous works

- Many works have already been carried out
  - Global process management
    - Condor, Sprite, Mosix, Bproc, ...
  - Global memory management
    - IVY, TreadMarks, GMS, ...
  - Global disk management
    - XFS, PFVS, Lustre, GFS, ...
  - Communications
    - RPC, Active messages, Madelaine, ...

Kerrighed Design Philosophy

- Avoid mechanisms and code redundancy
- Build a strong software architecture
- Integrate most previous works ideas in the same OS
  - Analyze existing and previous works
  - Factorize similar ideas within the same abstraction
  - Instantiate abstractions in distributed services
- Introduce new works

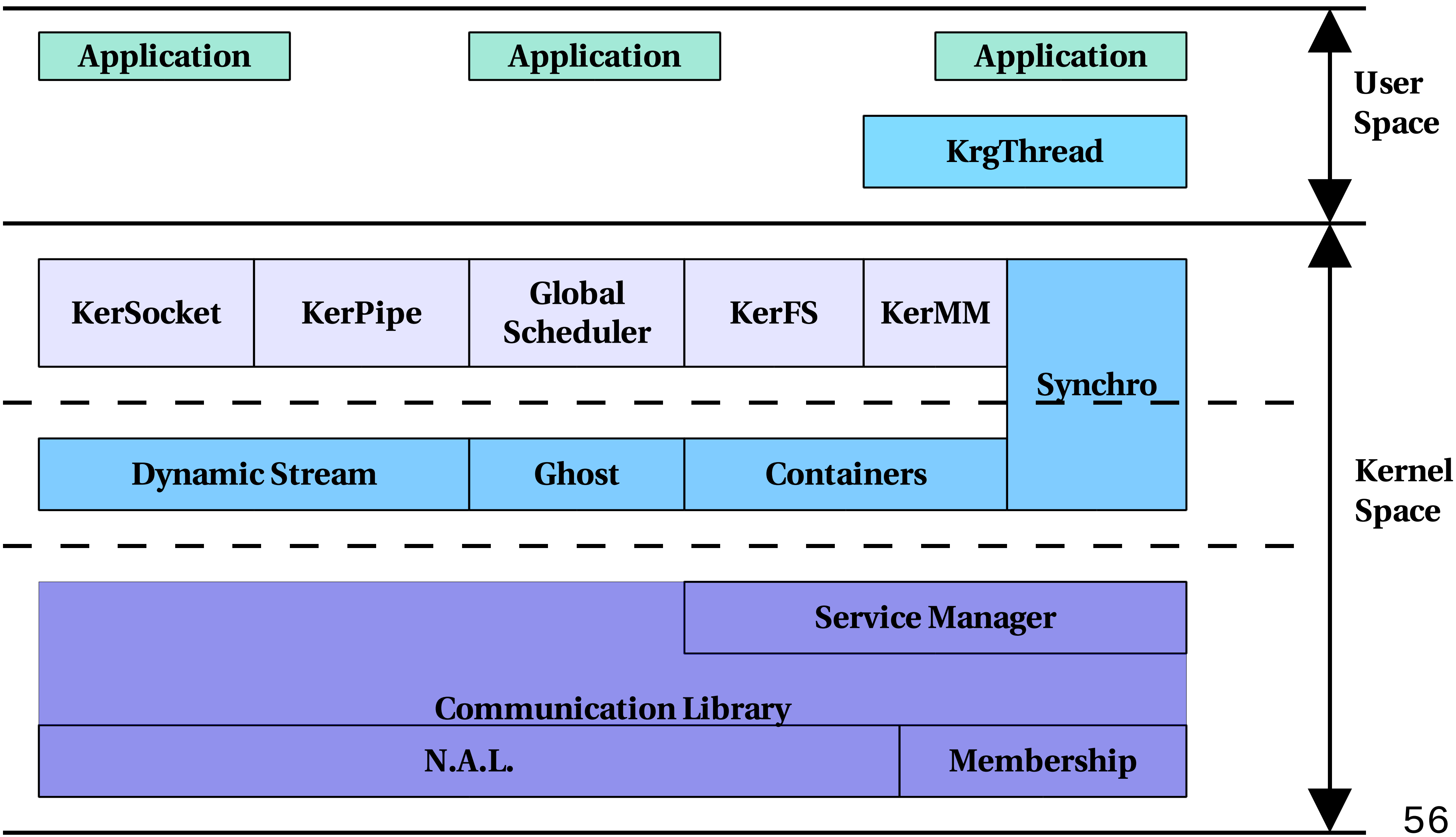
Factorization examples (1)

- Previous works (memory management)
  - Shared virtual memory (TreadMarks, ...)
  - Cooperative file cache (XFS, ...)
  - Memory injection (GMS, ...)
- Factorization : Containers
- Instantiation
  - Thread memory sharing cluster wide
  - Cluster wide IPC system V
  - Global file cache
  - ...

Factorization examples (2)

- Previous works
  - Process migration (Mosix, ...)
  - Process placement (Bproc, ...)
  - Process checkpointing (Condor, ...)
- Factorization : Process ghosts
- Instantiation
  - Distant process/thread creation
  - Process/thread migration
  - Process/thread checkpoint/restart

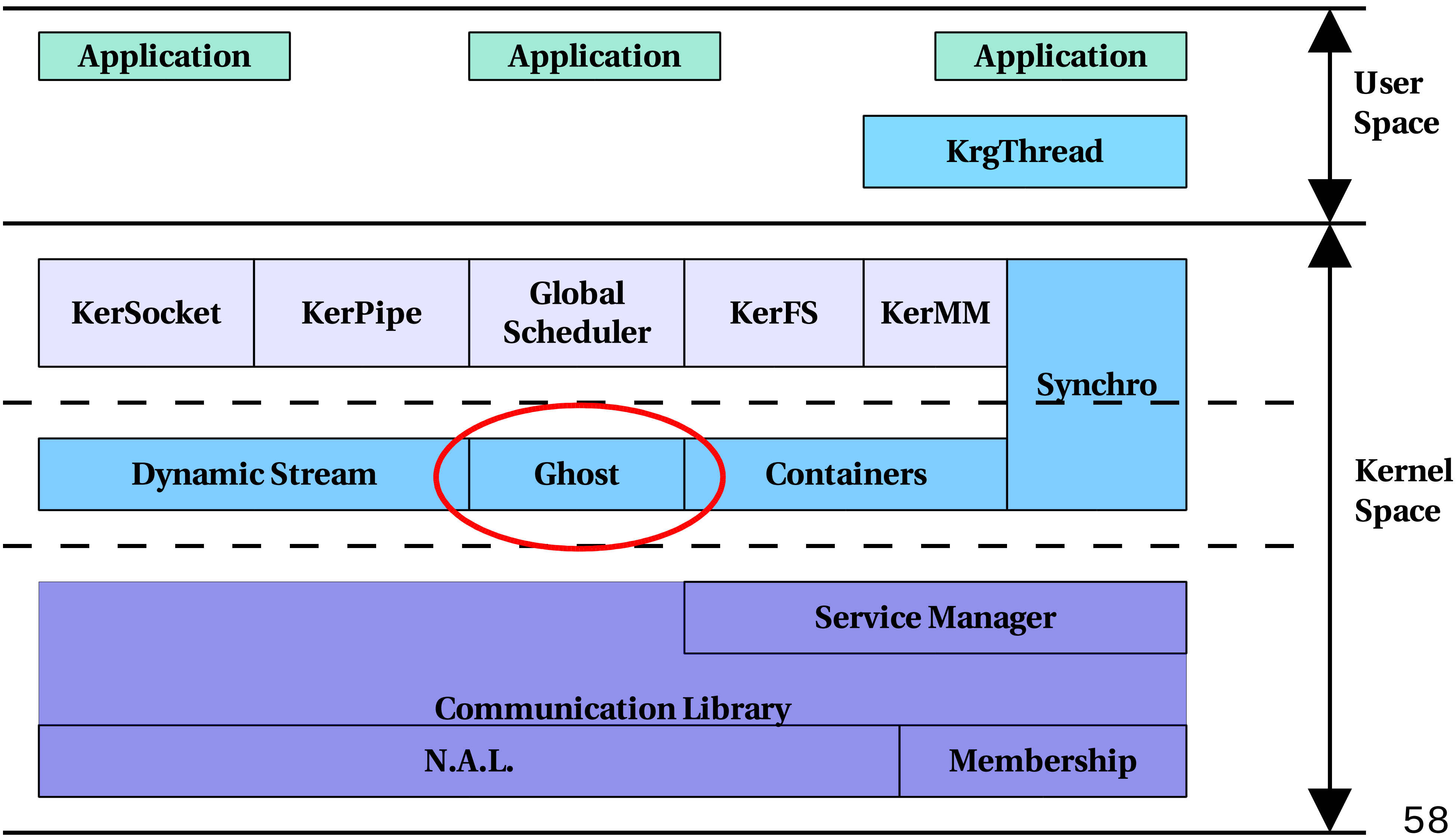
Global View of the Kerrighed Software Architecture



Outline

- Kerrighed Overview
  - What is Kerrighed ?
  - What about other system ?
  - Performance Evaluation
- Kerrighed Internal
  - Introduction
  - Ghosts
  - Containers
  - KerMM
- Conclusion

Global View of the Kerrighed Software Architecture



Process Migration in Kerrighed

- Migrate a process means migrating
  - Process context
  - Process memory
  - Open files
  - Active open streams (pipe, socket, ...)
  - ...
- Everything is fully distributed
  - No deputation
  - No home node

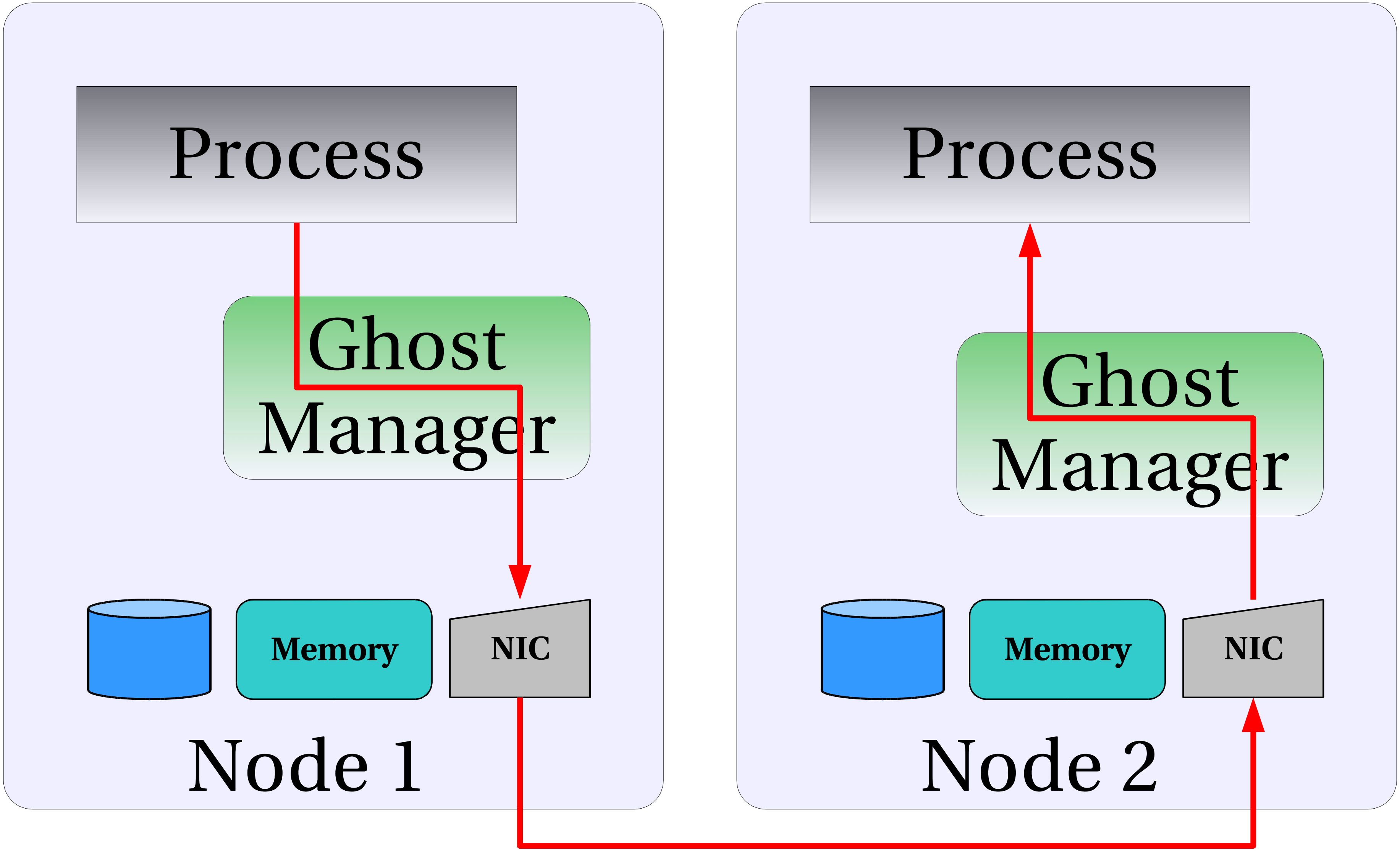
Process Migration in Kerrighed (2)

- 4 Main mechanisms
  - **Ghost**
    - Migrate process context
  - **Containers**
    - Migrate memory space
  - **Dynamic streams**
    - Migrate open streams
  - **KerFS**
    - Migrate open files

Process ghosts

- Generic mechanism to handle kernel data structures
  - Duplication
  - Migration
  - Checkpoint / restart
- Create an image of data structures and send it :
  - On disk
  - Over the network
  - In memory
- Ghost creation is independent of destination support

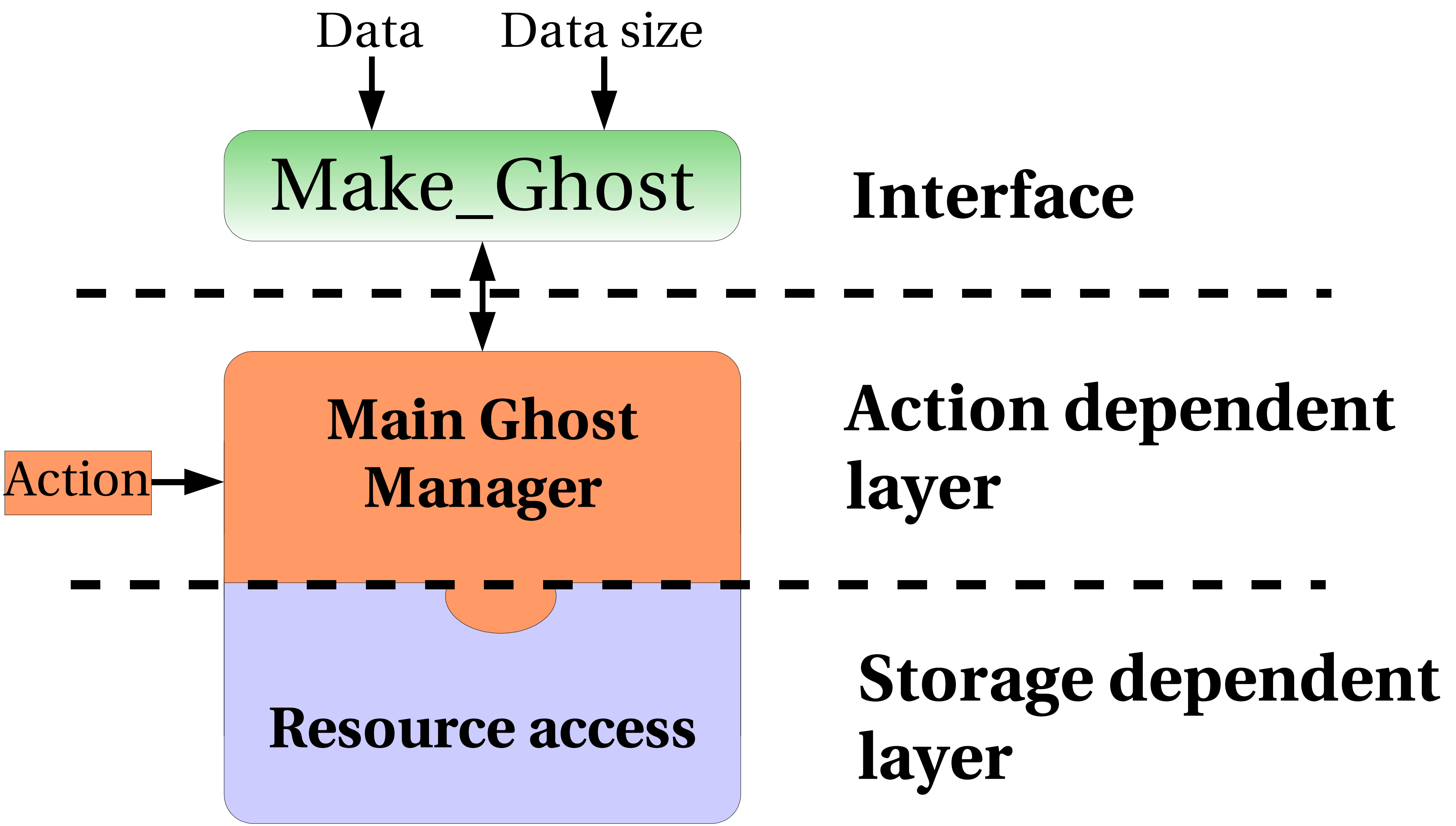
Migration / Duplication



Ghost Architecture

- A ghost is defined by
  - A cluster wide ghost identifier
  - An associated device
    - Network
    - Memory
    - Disk
- A ghost is controlled by an action
    - Load data
    - Store data
- One unique interface
  - Same code for load/restore data

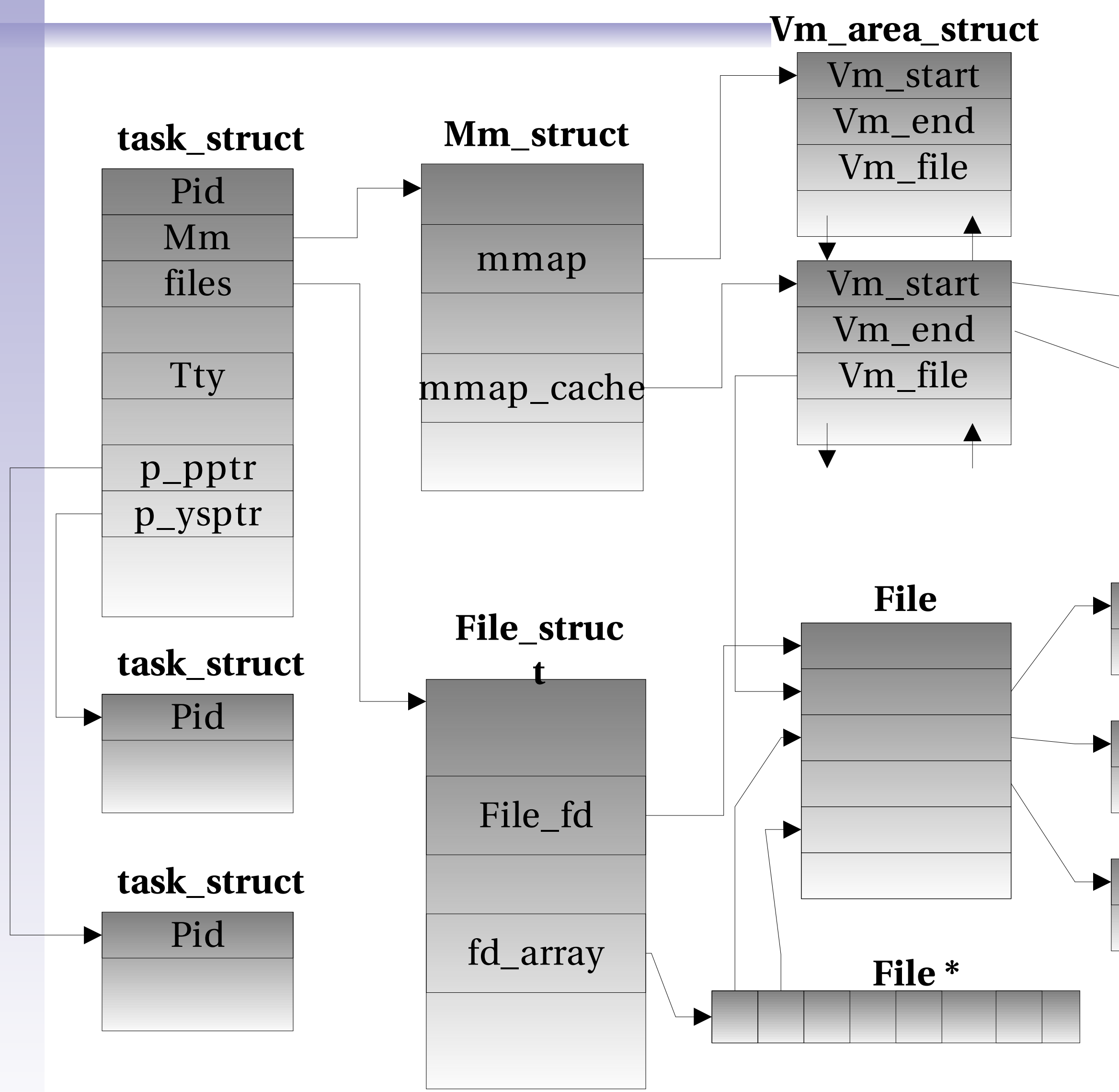
Ghost Architecture (2)



Ghost Use

- Ghost are used by functions parsing kernel data structures
  - Structure data pre-processing
  - Make ghost
  - Structure data post-processing
  - Call sub functions for sub data-structures

“Ghosting” a task structure

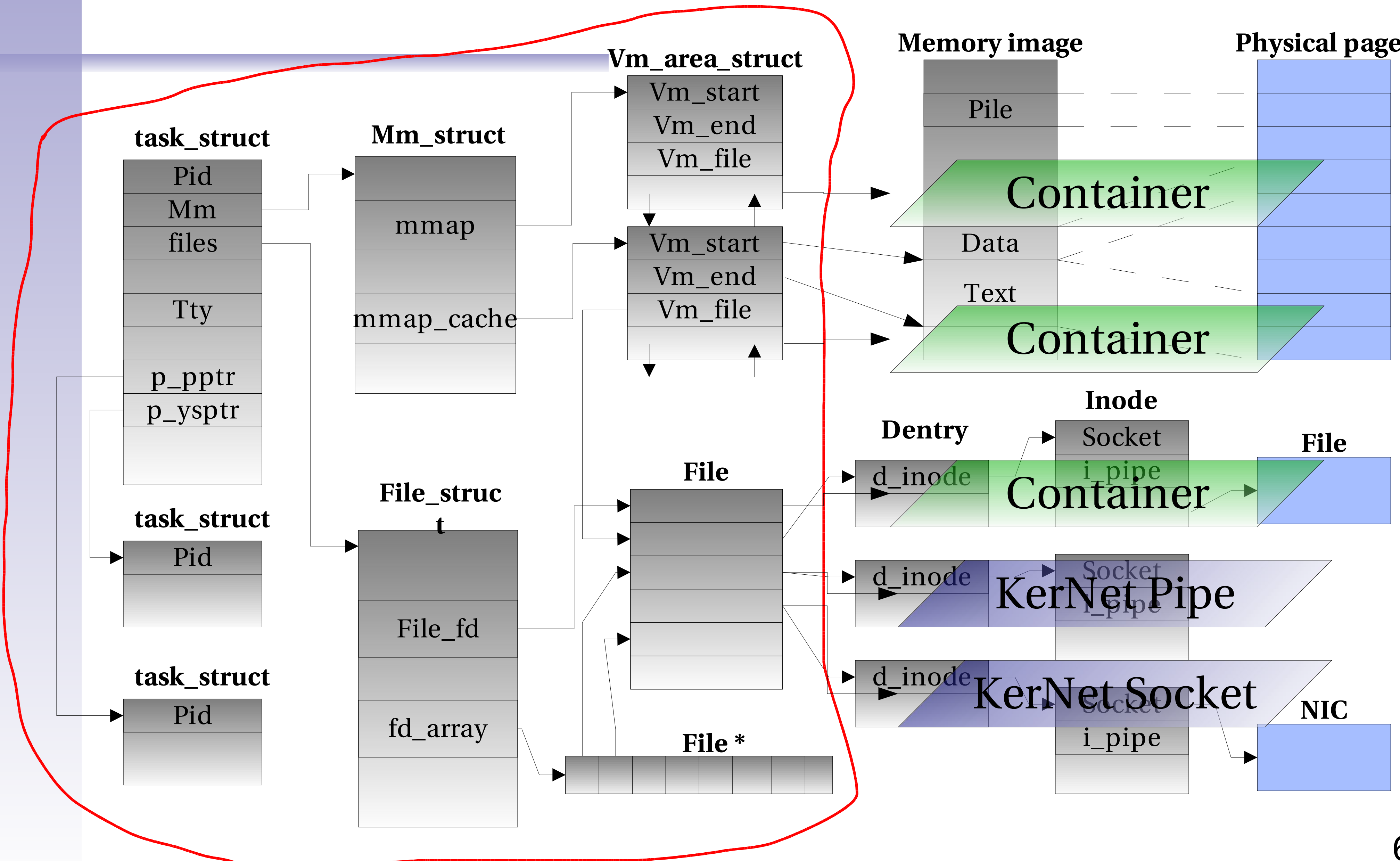



```
make_ghost_of_task_struct(
    ghost_t *ghost,
    struct task_struct *tsk)
{
 make_ghost (ghost, tsk,
      sizeof(task_struct));

 make_ghost_of_mm(ghost,
      tsk->mm);

 make_ghost_of_files (ghost,
      tsk->mm);

  ...
}
```

Migrating a task

- Ghost mechanism side effect :
  - We can migrate a pure computational sequential task !
    - Ghost device target = network
  - We can checkpoint a pure computational sequential task
    - Ghost device target = disk

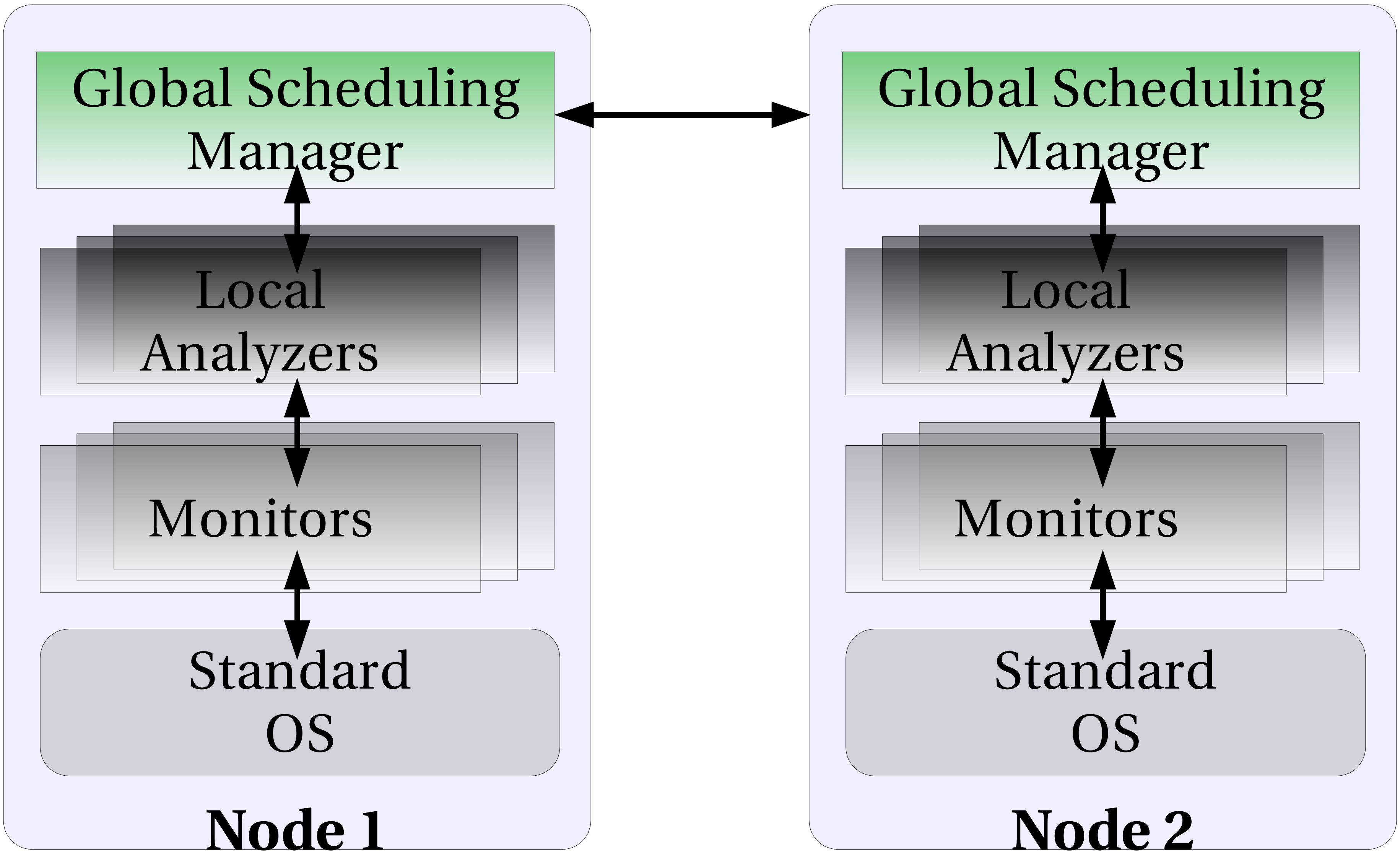
The Ghost Frontier



Configurable Global Scheduler : Design Goals

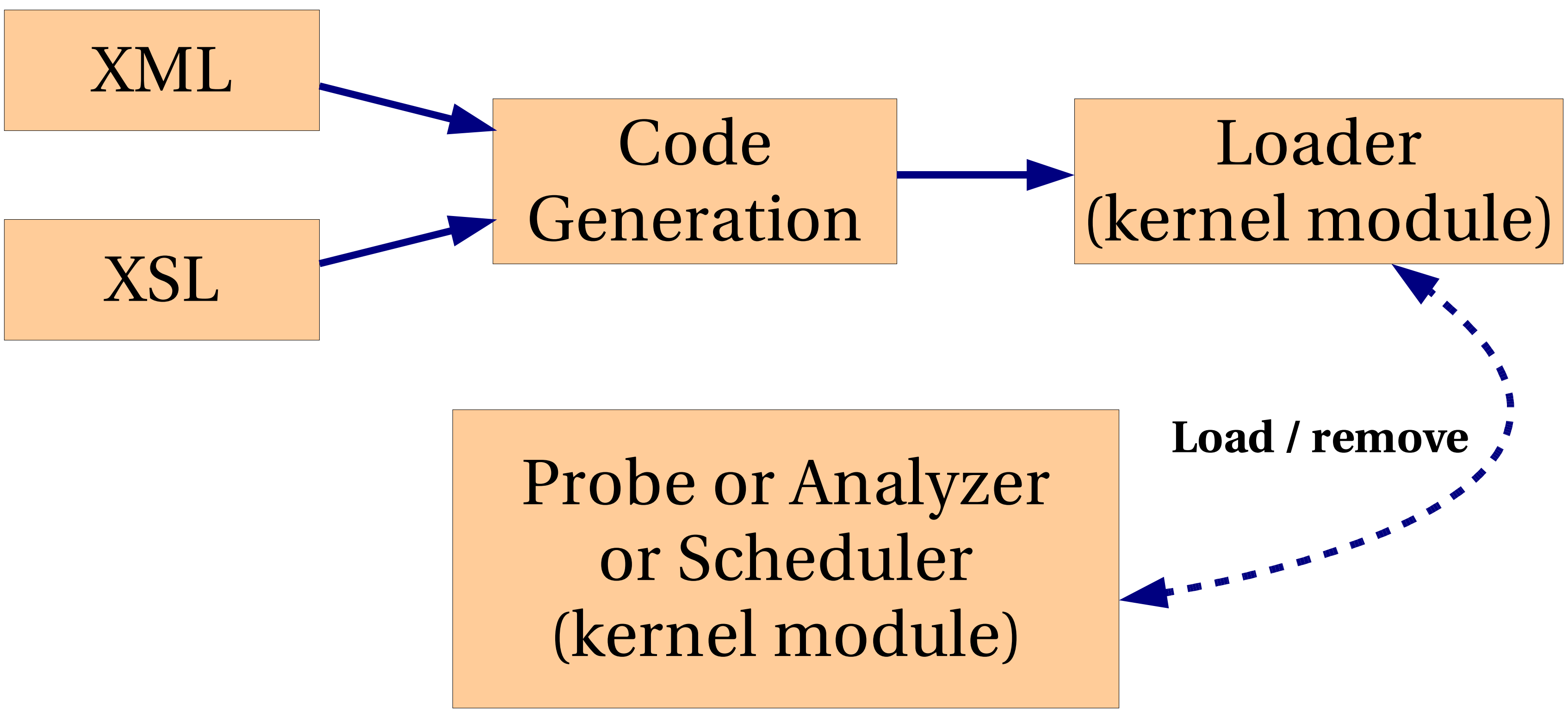
- It should be possible to implement any traditional placement or load balancing policy
  - Development and integration of global policies should be easy
    - Development environment
    - Modular architecture
  - Dynamic configuration of the global scheduler
  - Adaptive global scheduler
- Efficient process management mechanism
  - Minimal modification to the OS kernel
    - No modifications to the local OS scheduler

Modular Global Scheduler

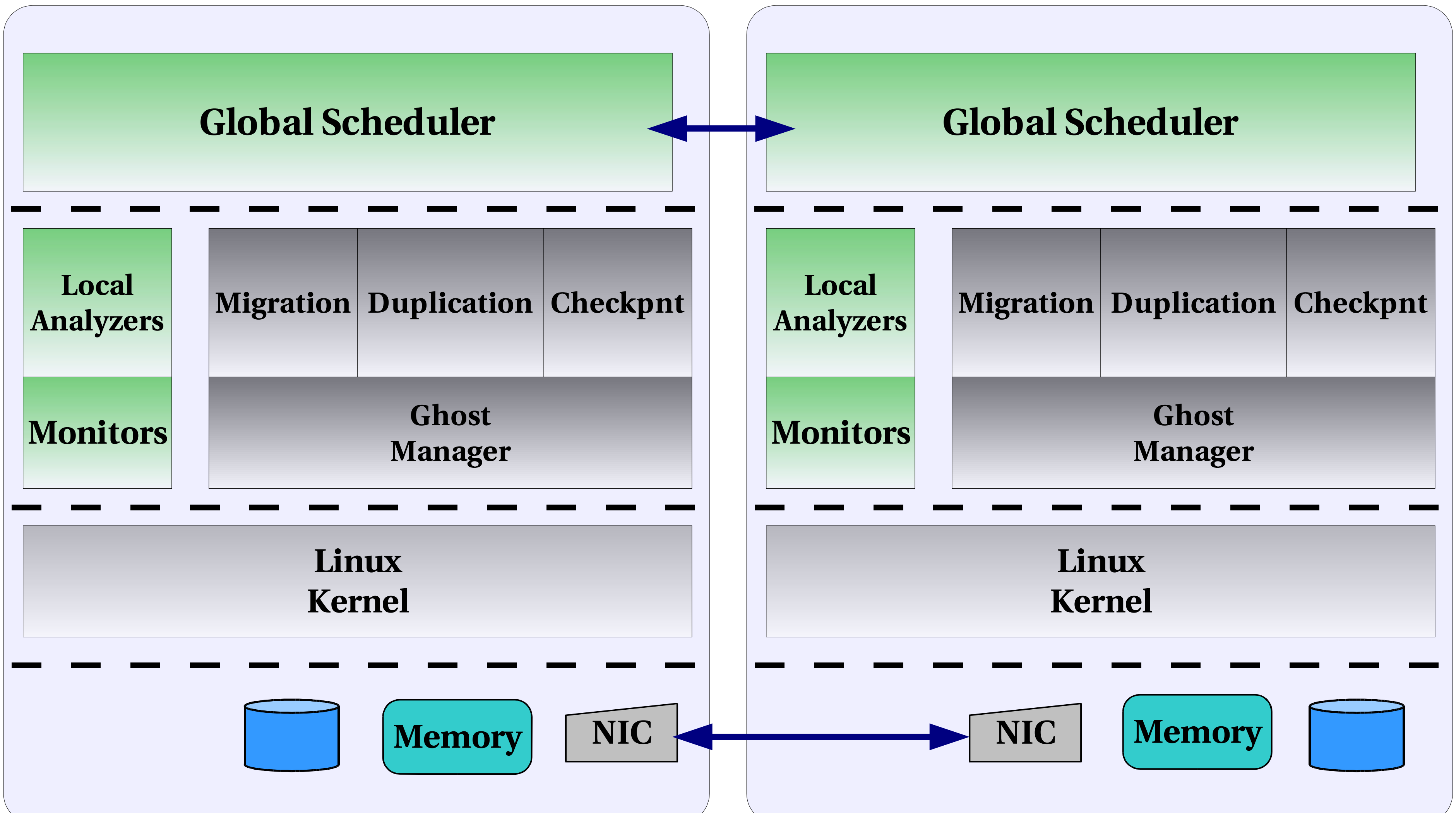


Configuration

- All components are configured with XML files
- All components can be hot-loaded and hot-removed



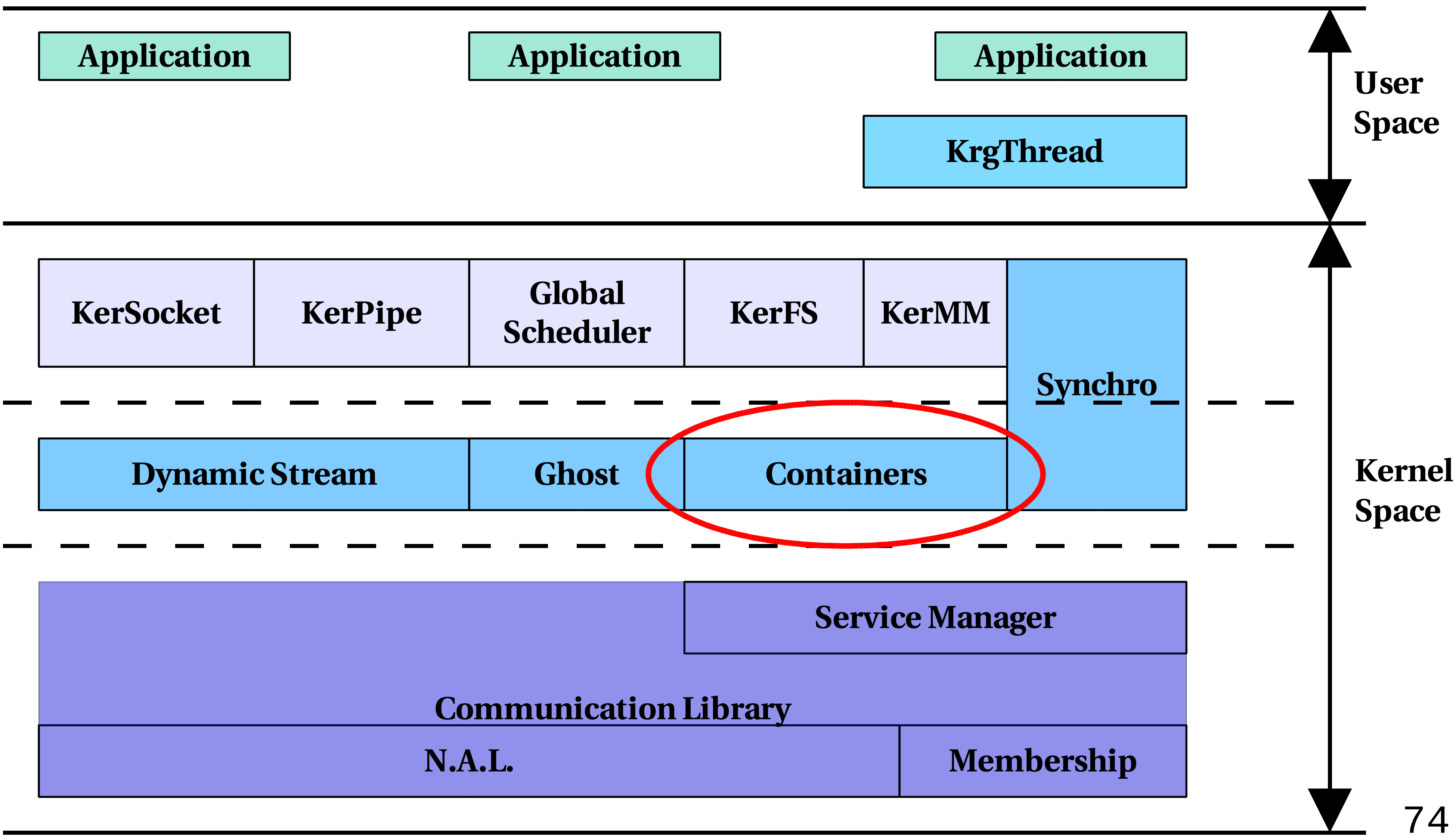
Process Management Big Picture



Outline

- Kerrighed Overview
  - What is Kerrighed ?
  - What about other system ?
  - Performance Evaluation
- Kerrighed Internal
  - Introduction
  - Ghosts
  - Containers
  - KerMM
- Conclusion

Global View of the Kerrighed Software Architecture



Definition of container

- Containers
  - Generic mechanism to share data cluster wide
  - Share data between cluster nodes at OS level
  - Transparent access to remote data
  - Ensure coherency of shared data
  - Efficient access to data
- Linkers
  - Interface between containers and host OS

Containers : a Generic Data Sharing Mechanism

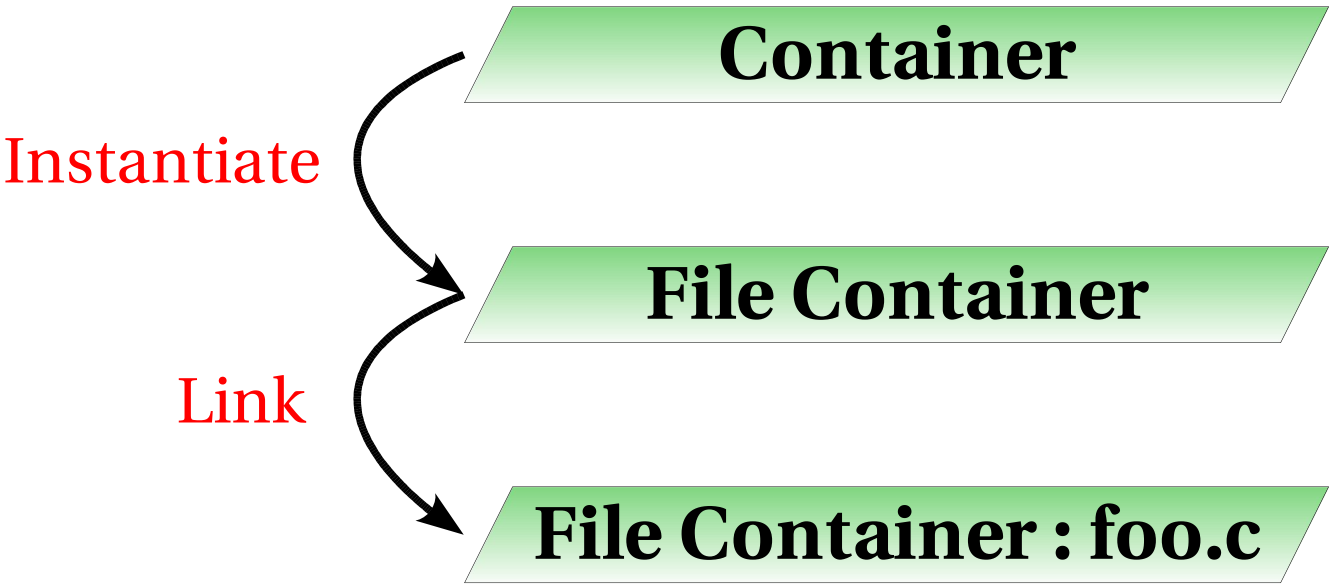
- Data hosting and sharing
  - A container hosts a set of **objects**
  - Objects : memory pages, data structure
  - Unit of sharing : 1 object
  - Node local memory = **cache** of container objects
- Object coherence management
  - **MESI**-like coherence algorithm
    - Single writer / multiple reader
    - Invalidation on write

Container Instantiation : IO Linkers

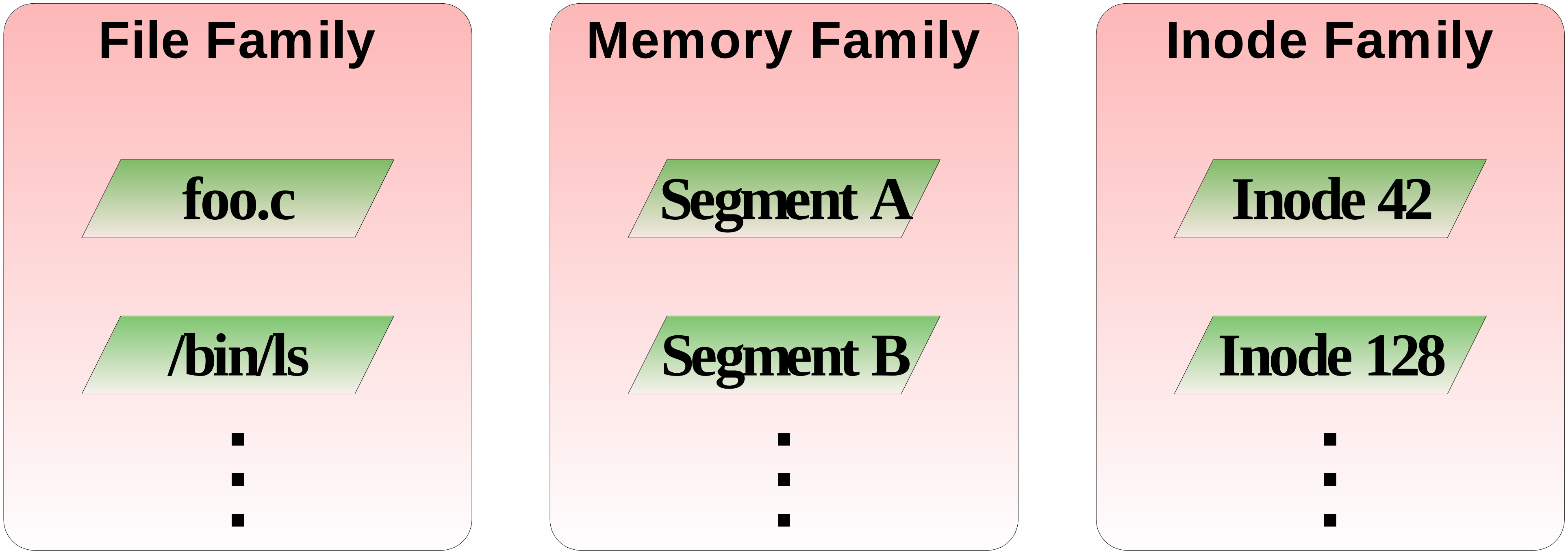
- Container : **generic** mechanism
  - Can host any kind of object
- Containers instantiated by **IO linkers**
  - Determine the nature (**family**) of hosted object
  - Define object input/output functions
  - One kind of IO linker per kind of object to share
    - Memory pages
    - File cache pages
    - Inodes
    - ...

Container Linked Object

- 1 container + 1 file linker = 1 file container
- One container per object to share
  - The object to share has to be **linked** to the container



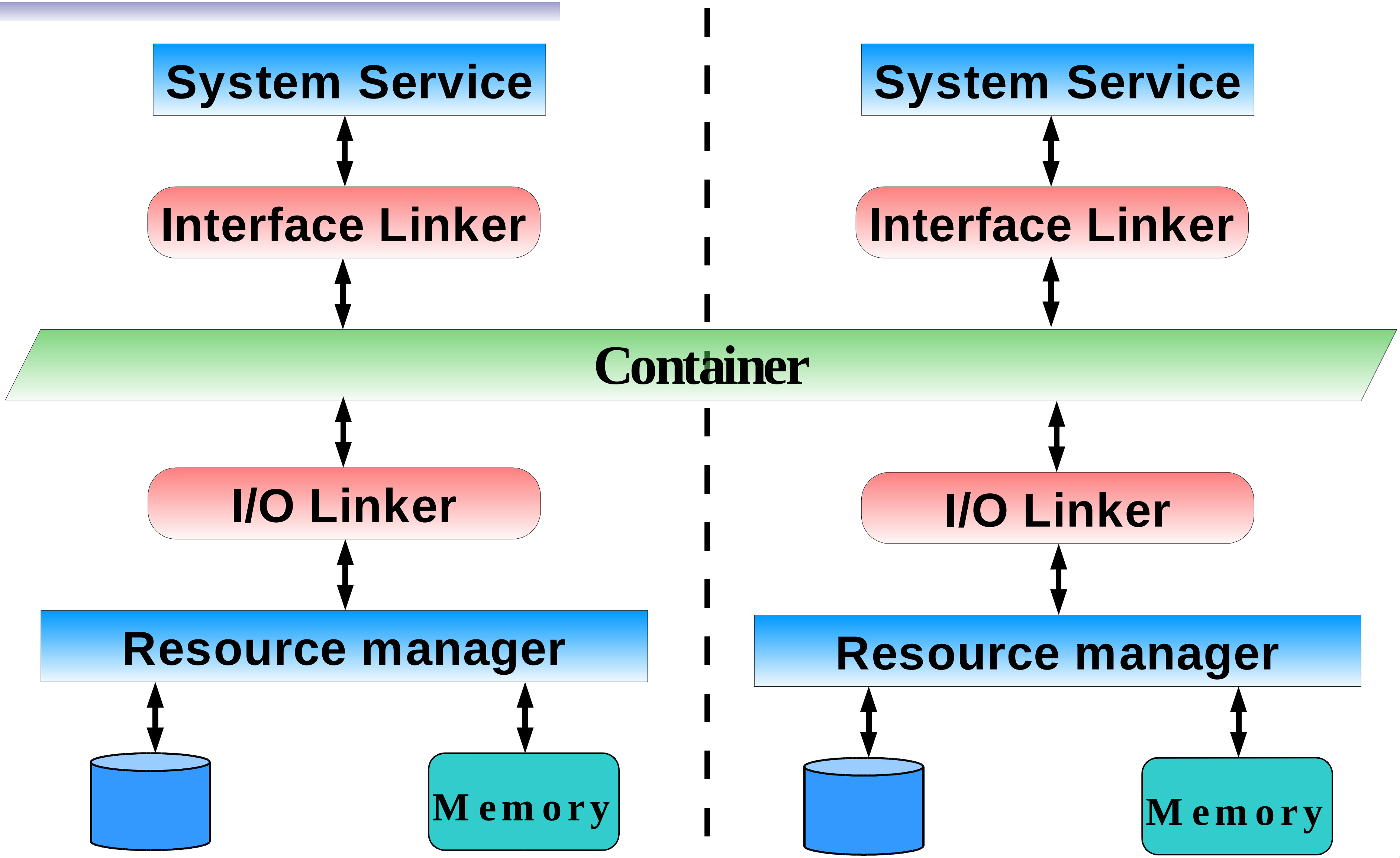
Container Families



Container Linked Node

- A container can be linked to a given device
  - Disks
- Linked containers
  - Container data are stored on device(s) located on specific node(s)
  - First access to a data is sent to the linked node(s)
- Not linked container
  - Data are not linked to a specific node
  - First access to a data can be done locally

Containers Architecture



Container interface

- High level interface (used by interface linkers)
  - **ctnr_find_object** (Container id, object id)
    - Check if an object is present in local memory
  - **ctnr_get_object** (Container id, object id)
    - Place an object copy in local memory
  - **ctnr_grab_object** (Container id, object id)
    - Place an unique object copy in local memory
  - **ctnr_put_object** (Container id, object id)
    - Release an object

Container interface (2)

- **ctnr_remove_object** (Container id, object id)
  - Remove an object from a container, cluster wide
- **ctnr_sync_object** (Container id, object id)
  - Synchronize an object with its physical device

Data input/output in Containers

- Interface offered by I/O linkers
  - **First_touch**(…)
    - Allocate and initialize data
  - **Invalidate_object(…)**
    - Evict a data from local memory (used by the coherence protocol)
  - **Remove_object**(…)
    - Remove a data from a container (container destroy or data removal)
  - **Free_object**(…)
    - Free a page from local memory (used to free a container)
- One kind of container per object type to share

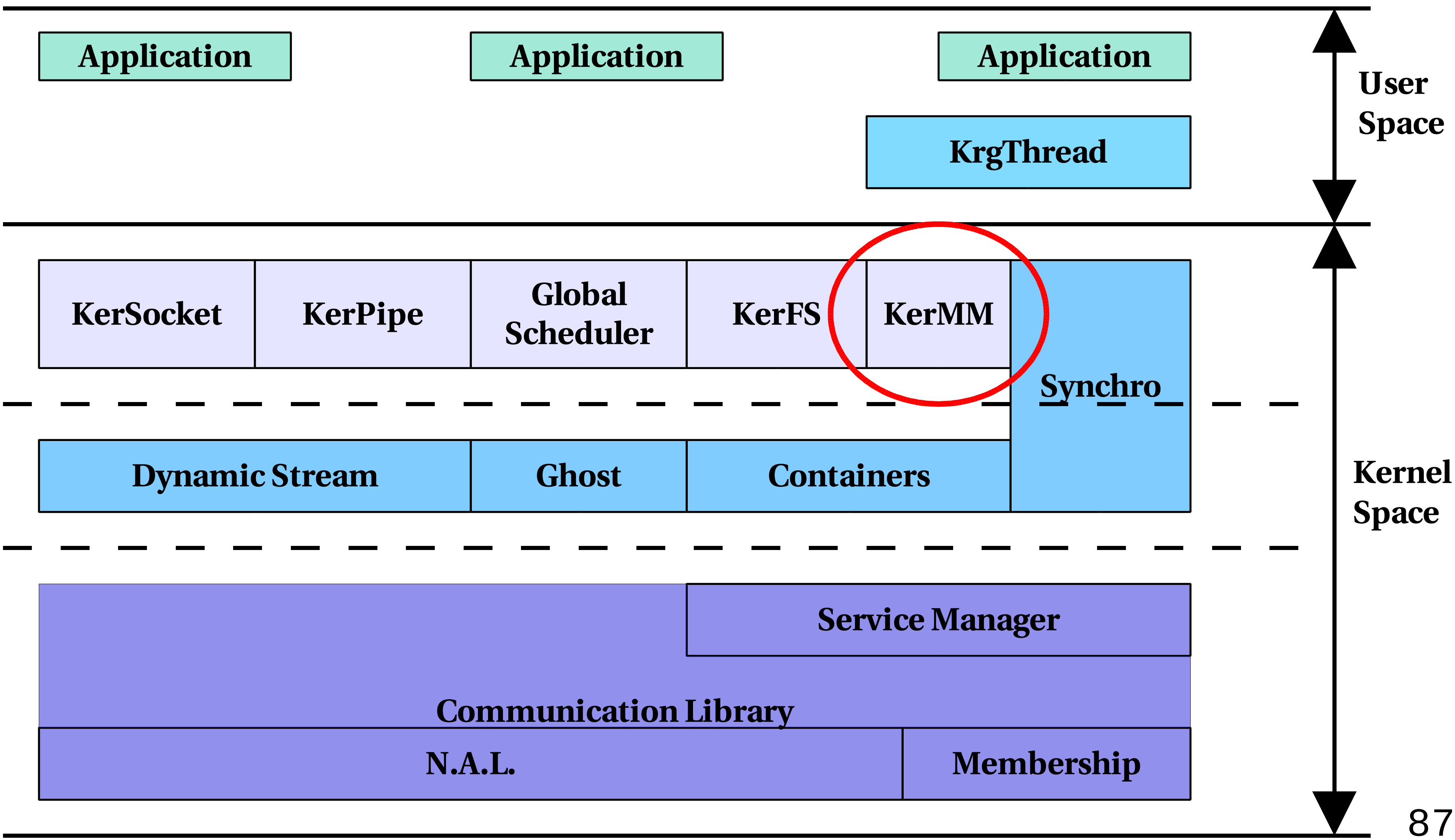
Container use in Kerrighed

- Used as a basic bloc to implement
  - Process memory migration
  - Memory sharing cluster wide
    - Thread memory
    - IPC system V segments
  - File cache sharing cluster wide
  - Inodes sharing cluster wide
  - Open files pointer sharing

Outline

- Kerrighed Overview
  - What is Kerrighed ?
  - What about other system ?
  - Performance Evaluation
- Kerrighed Internal
  - Introduction
  - Ghosts
  - Containers
  - KerMM
- Conclusion

Global View of the Kerrighed Software Architecture



Global Memory Management

- Enable memory sharing cluster wide
  - Intra-application **virtual** memory sharing
    - Threads memory
    - System V memory segments
  - Inter-nodes **physical** memory sharing
    - Remote memory paging
- Manage distributed address space
  - Address space **migration**
  - **Threads** address space management
    - Mmap, munmap, ...
    - Stack, heap

Global Memory Sharing

- Rely on **containers** for data sharing
- KerMM defines :
  - A memory IO linker
  - A memory interface linker
- Thread memory and System V memory segment
  - Same memory sharing mechanism
  - Dedicated interface link/unlink mechanisms

Memory IO Linker

```
IO_Link ( ctnr, vma )
  For each physical page in VMA
  {
    obj = alloc_ctnr_object (cntr, objid, page);
    ctnr_insert_object (ctnr, obj) ;
  }
```

```
First_Touch ( p )
  page := Alloc_Page () ;
  Return page ;
```

```
Invalidate_Page ( p )
  NOP ;
```

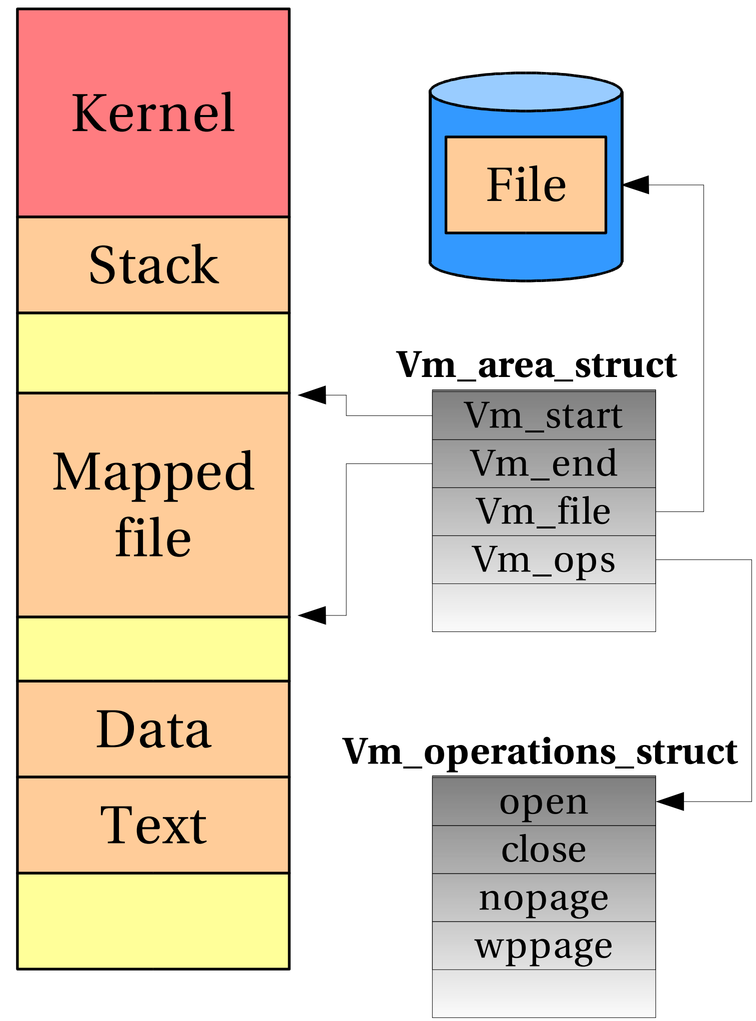
```
Flush_Page ( p )
  NOP ;
```

Memory Interface Linker

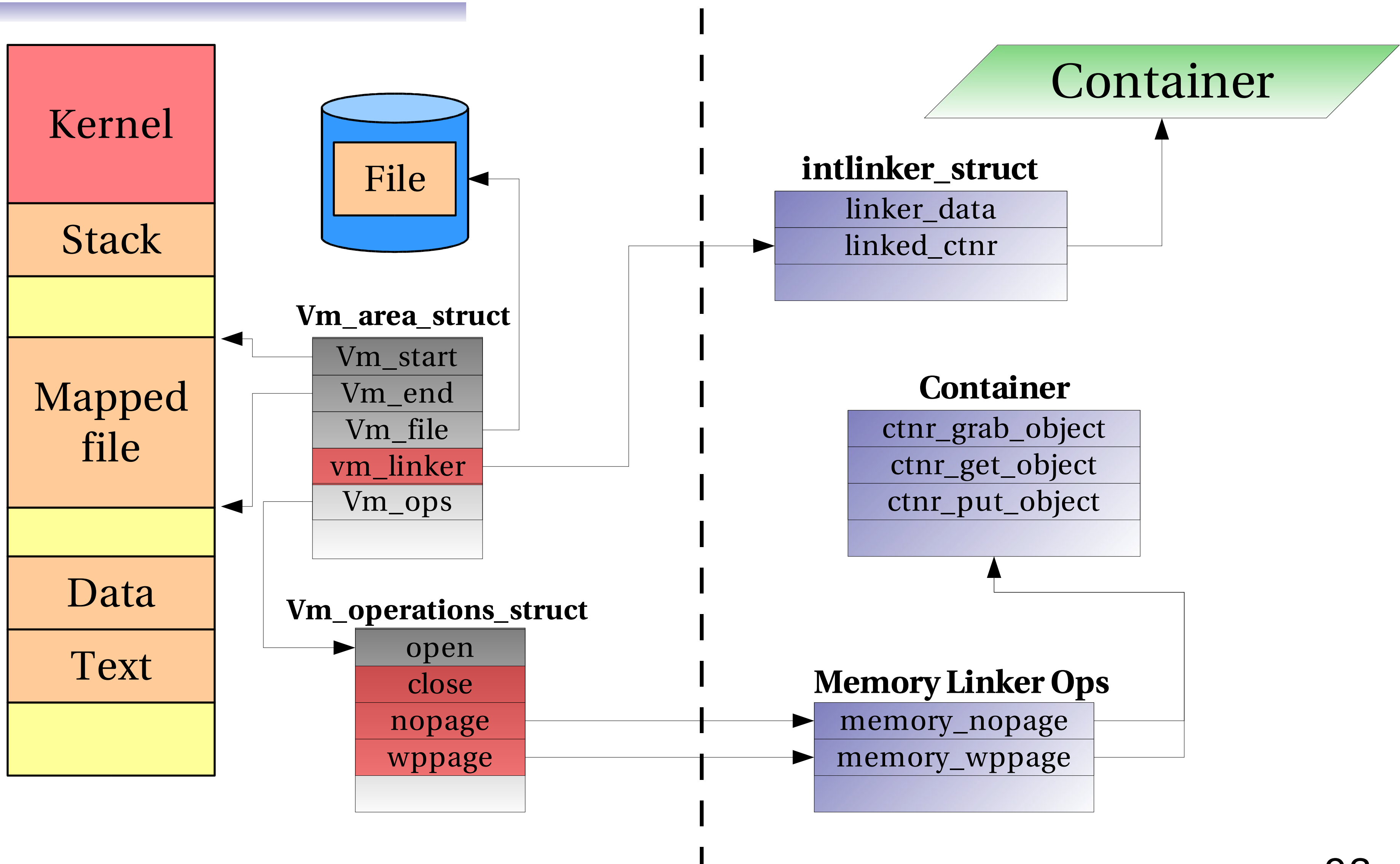
- Goals
  - Link a memory segment to a container
  - Divert page faults to the linked container
- Done at the Virtual Memory Area (VMA) level

Memory Interface Linker (2)

- A VMA is defined by
  - Begin/end address
  - Access wrights
  - Linked file
  - Memory operations
- Memory operations :
  - VMA closing
  - First access to a page
  - Copy on write
  - ...



Memory Interface Linker (3)

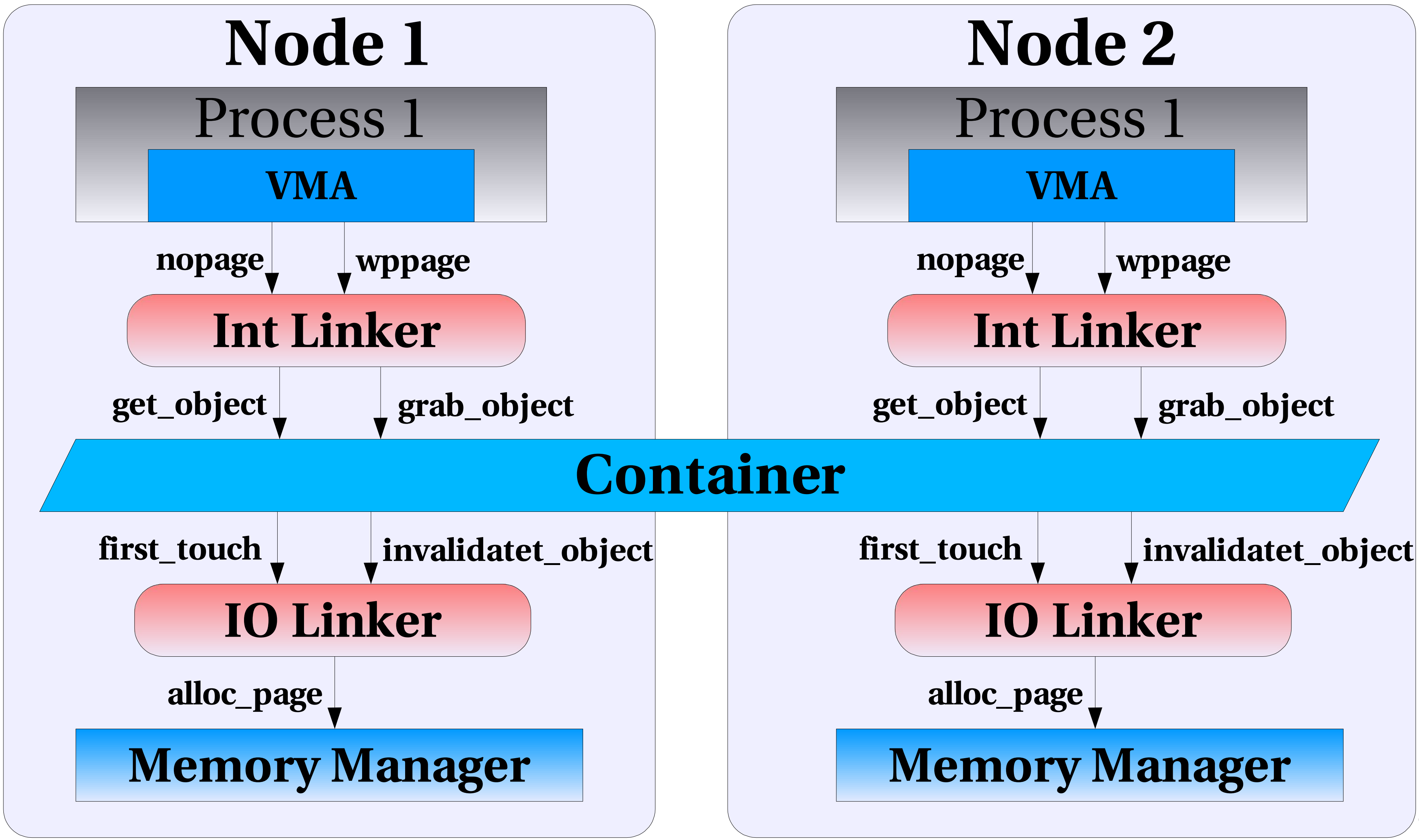


Memory Interface Linker (4)

```
memory_nopage ( ctnr, vma, address )
  if (write access)
    page = ctnr_grab_object (ctnr, pageid) ;
  else
    page = ctnr_get_object (ctnr, pageid) ;
  return page ;
```

```
memory_wppage ( ctnr, vma, address )
  page = ctnr_grab_object (ctnr, pageid) ;
  return page ;
```

Memory Linkers Summary



Outline

- Kerrighed Overview
  - What is Kerrighed ?
  - What about other system ?
  - Performance Evaluation
- Kerrighed Internal
  - Introduction
  - Ghosts
  - Containers
  - KerMM
- Conclusion

Kerrighed Code Size

Process Management	20 000
KerFS	9 500
KerMM	3 000
Container	10 000
Synchro	11 000
KerPipe / KerSocket	8 000
Dynamic Streams	3 500
Comm lib	9 000
N.A.L.	4 000
Service Manager / Ghost	7 000
Total	**85 000**

Kerrighed People

- **Project head**
  - Christine Morin
- **Research Engineers**
  - Pascal Gallard
  - Renaud Lottiaux
- **Post-Doc**
  - Geoffroy Vallée
- **Faculty**
  - Thierry Garcia
- **Ph.D. Students**
  - Emmanuel Jeanvoine
  - Louis Rilling
- **Interns**
  - Boris Daix
  - Matthieu Fertré

www.kerrighed.org