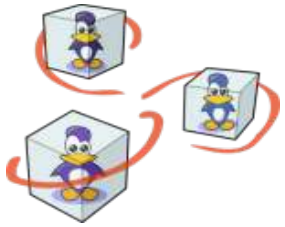
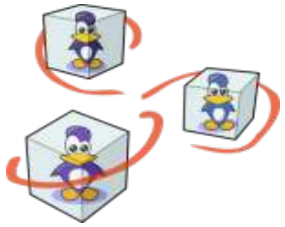


Kerrighed Internal

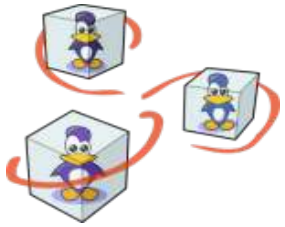
Renaud Lottiaux, Pascal Gallard
Paris Team, Irisa / Inria
July 7, 2005

Kerrighed
Linux clusters made easy



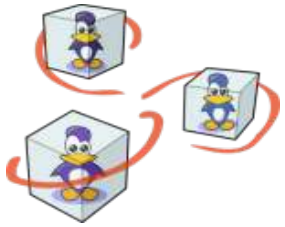
Context

- Today clusters :
  - Set of PCs
  - Fast interconnection network
  - One instance of a regular OS on each node
- View of a fully distributed machine
  - Distributed resources
    - Memory, disk, CPU, ...
  - Poor distributed services
    - NFS, RPC, ...
- No standard OS designed for clusters



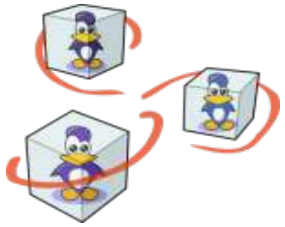
Kerrighed goals

- Goal : create an OS dedicated to clusters
  - Ease the use of clusters
  - Achieve high performance
- Target HPC Applications
  - Scientific applications
  - Message passing (MPI, PVM, sockets, ...)
  - Shared memory (Pthread, Open-MP, ...)
  - Sequential applications



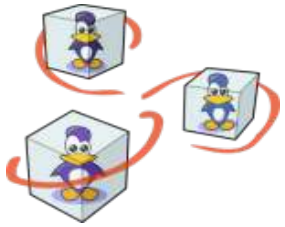
Kerrighed Design Philosophy

- 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



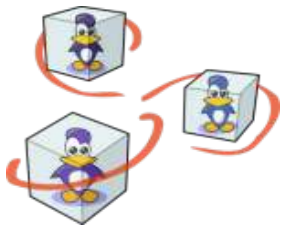
Factorization examples (1)

- 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
  - ...

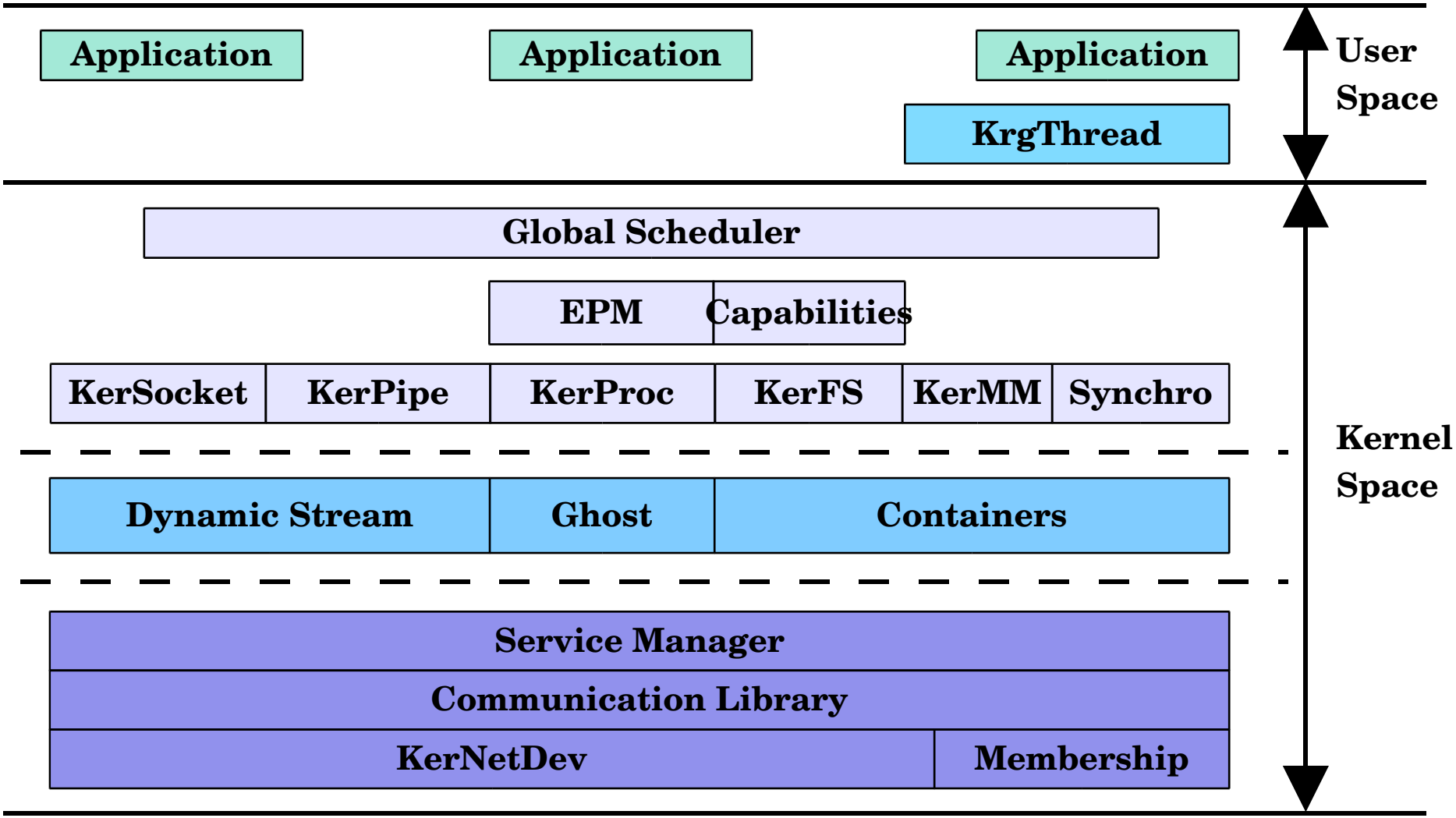


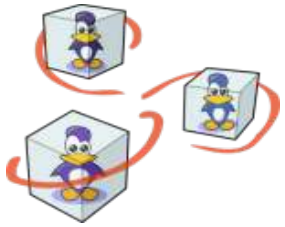
Factorization examples (2)

- 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

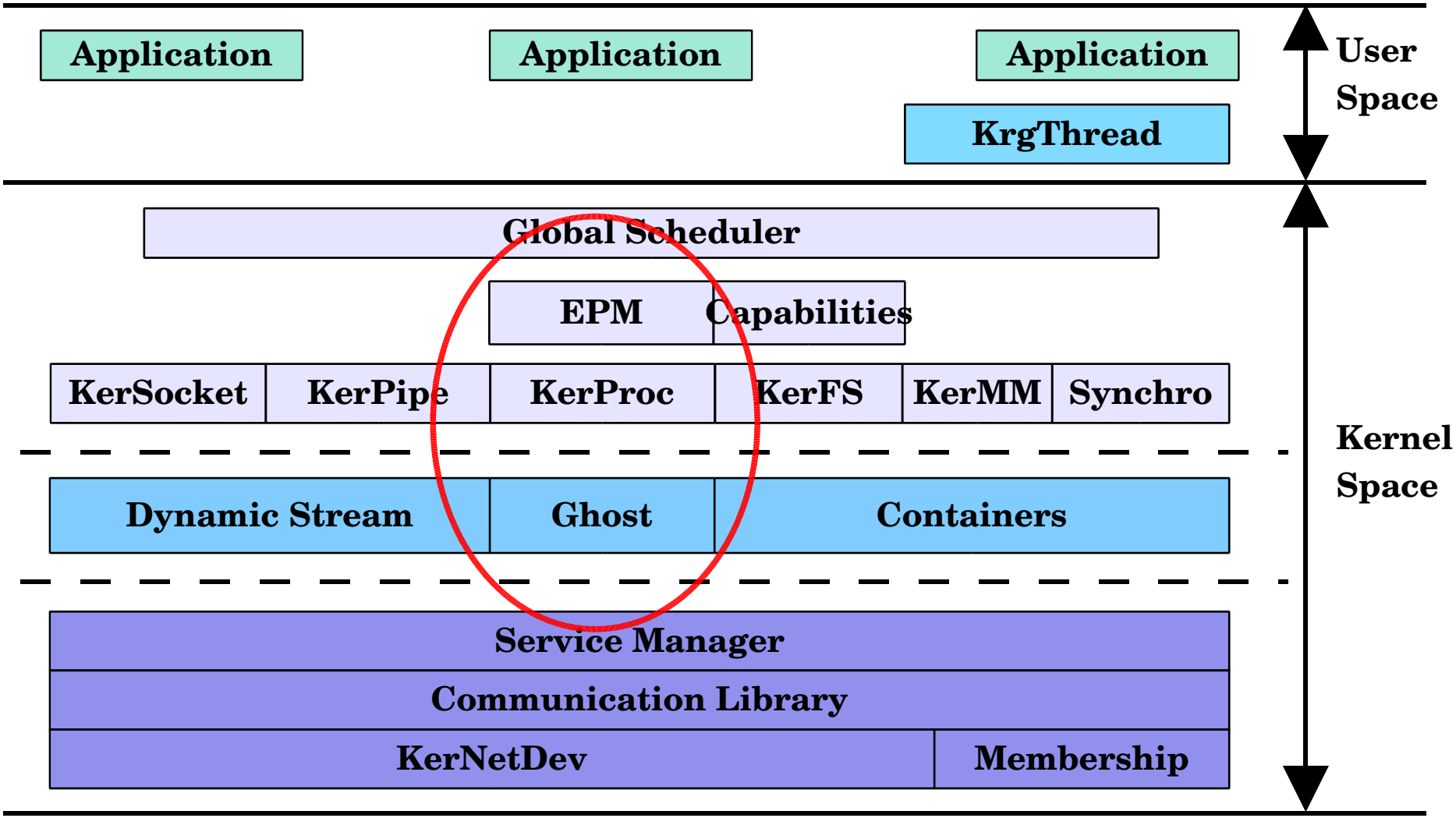


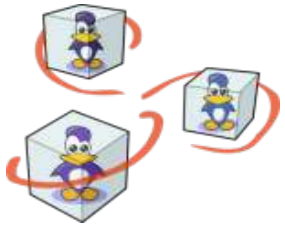


Outline

- Global Process Management
- Containers
- Global memory management
- Dynamic streams
- Capabilities
- Kerrighed in Action !

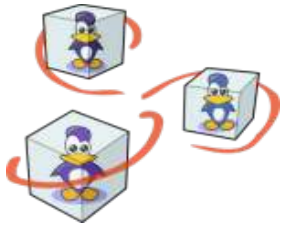
Global View of the Kerrighed Software Architecture





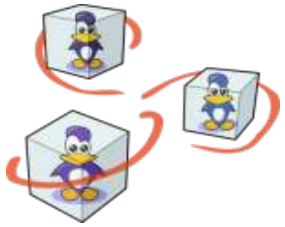
Global Process Management

- To achieve global process management we need
  - To name processes cluster wide
    - We use a cluster wide pid
      - Kill a remote task
      - Killall cluster wide
      - Global *ps*, global *top*
      - ...
  - To schedule processes cluster wide
    - We use process migration



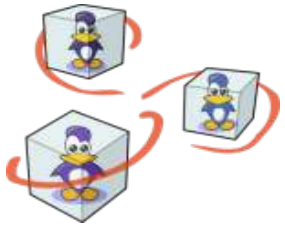
Process Migration in Kerrighed

- 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



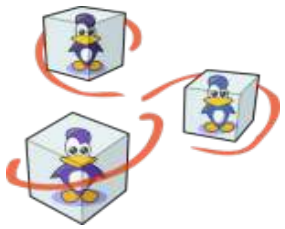
Process Migration in Kerrighed (2)

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

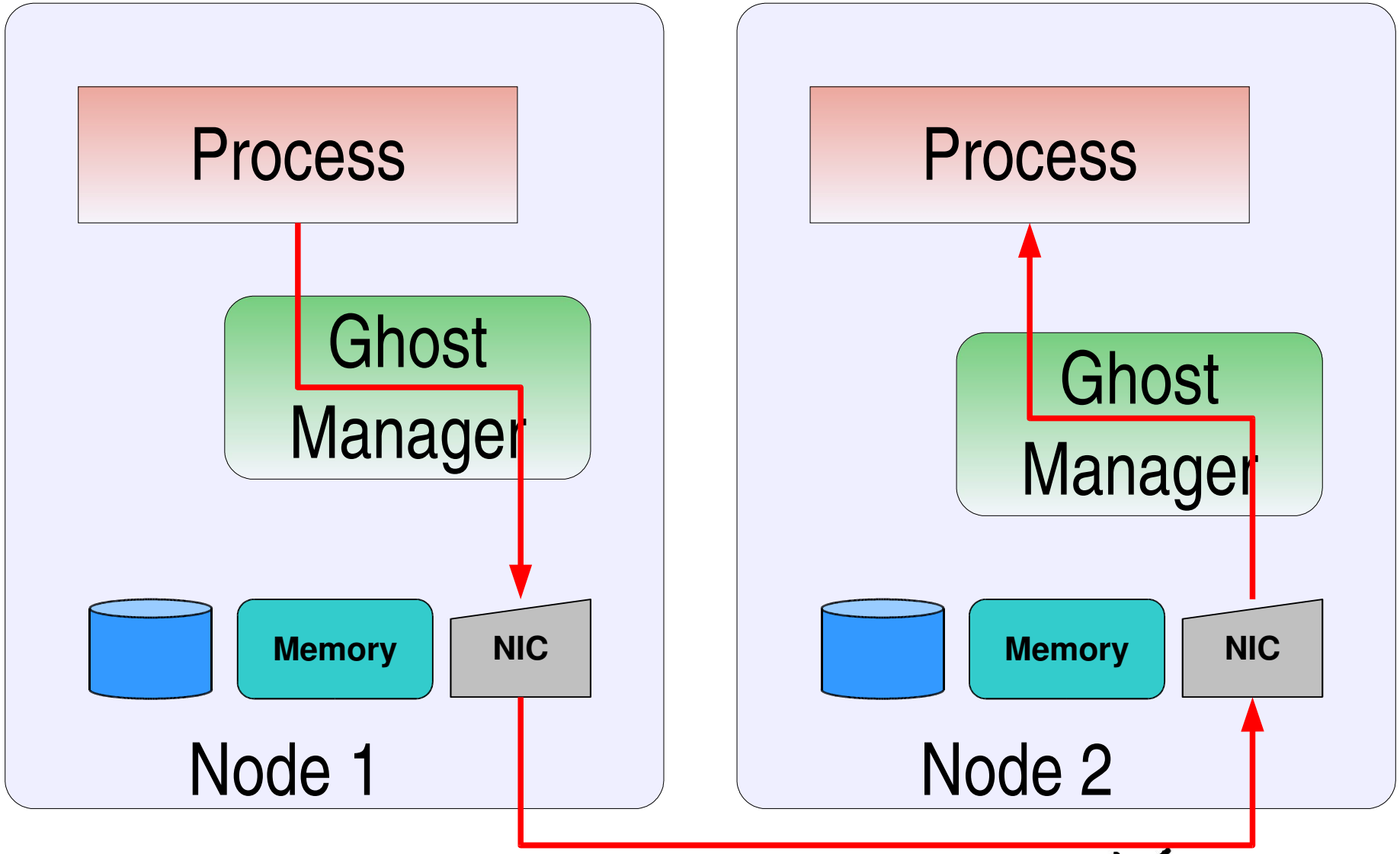


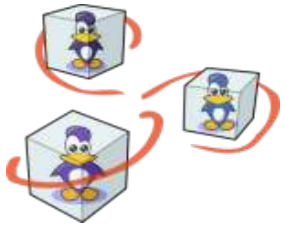
Process ghosts

- 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

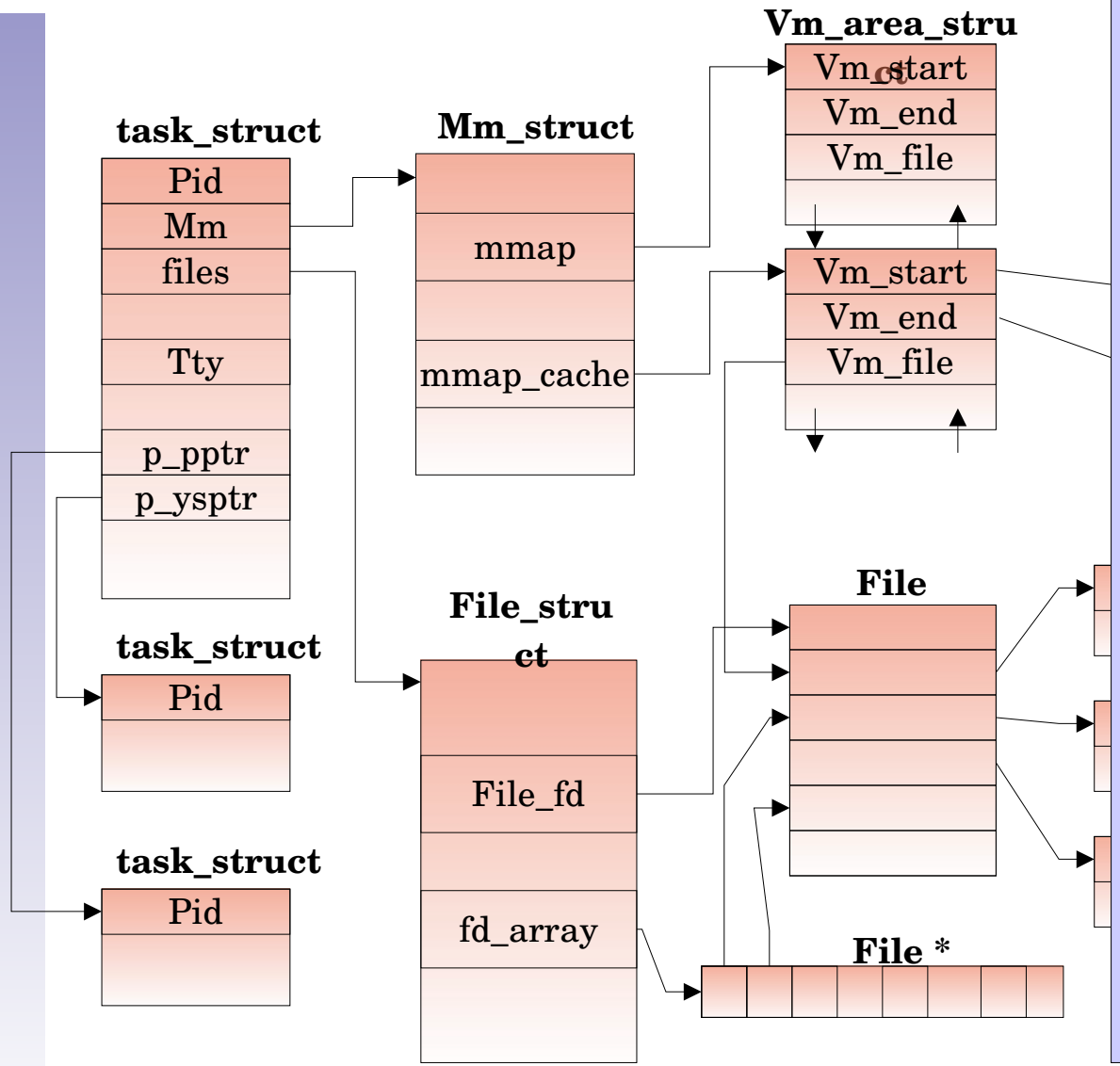




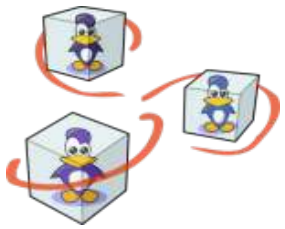
Ghost Architecture

- A ghost is defined by
  - A cluster wide ghost identifier
  - An associated device
    - Network
    - Memory
    - Disk
  - A ghosting action
    - Migration
    - Duplication
    - Checkpoint
    - Restart

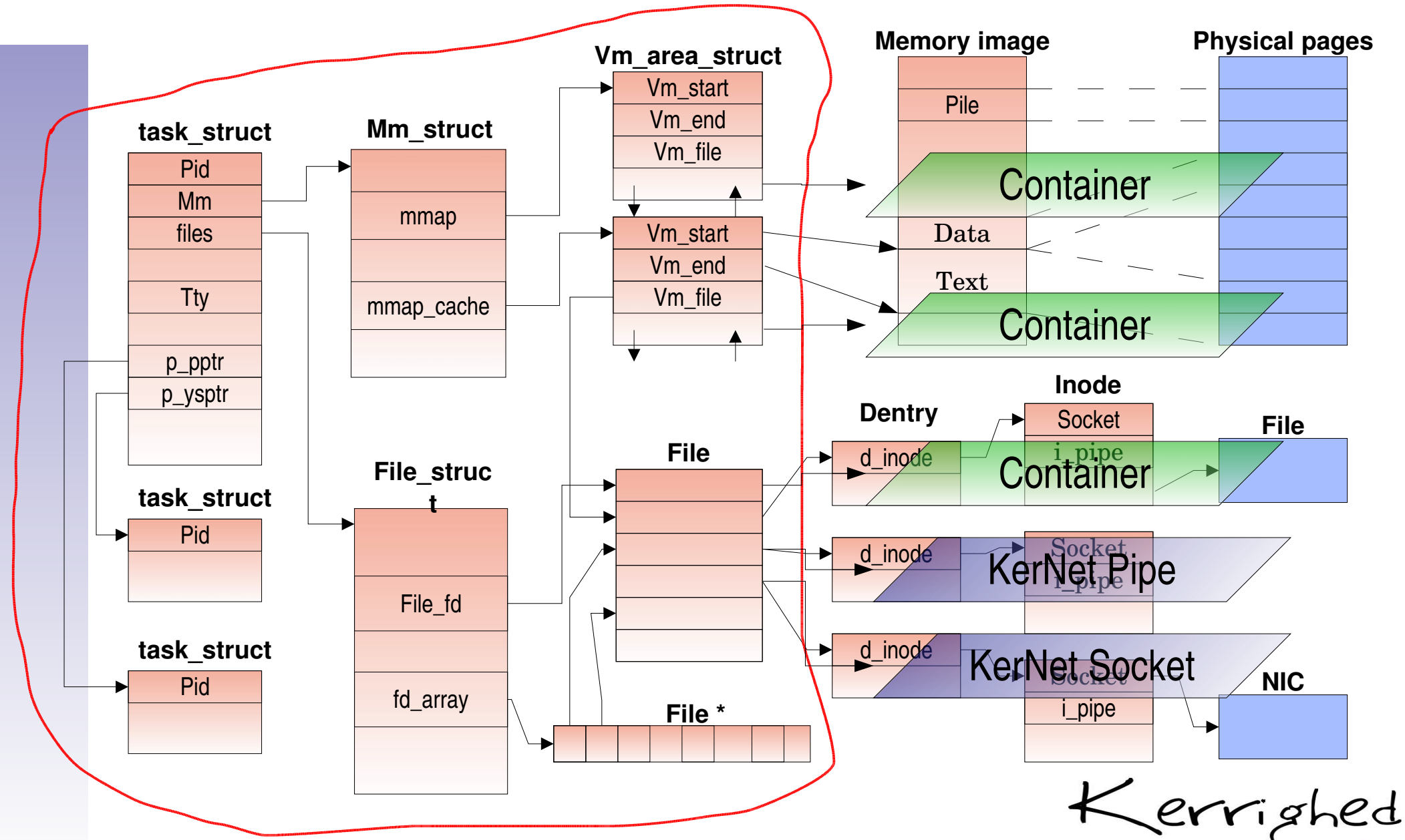
“Ghosting” a task structure



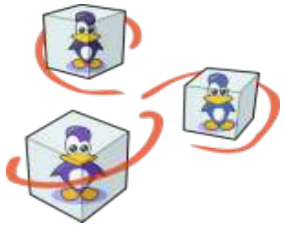

```
export_task_struct(
    ghost_t *ghost,
    struct task_struct *tsk)
{
 ghost_write (ghost, tsk,
      sizeof(task_struct));

 make_ghost_of_mm(ghost,
      tsk->mm);

 make_ghost_of_files (ghost,
      tsk->mm);

  ...
}
```



The Ghost Frontier



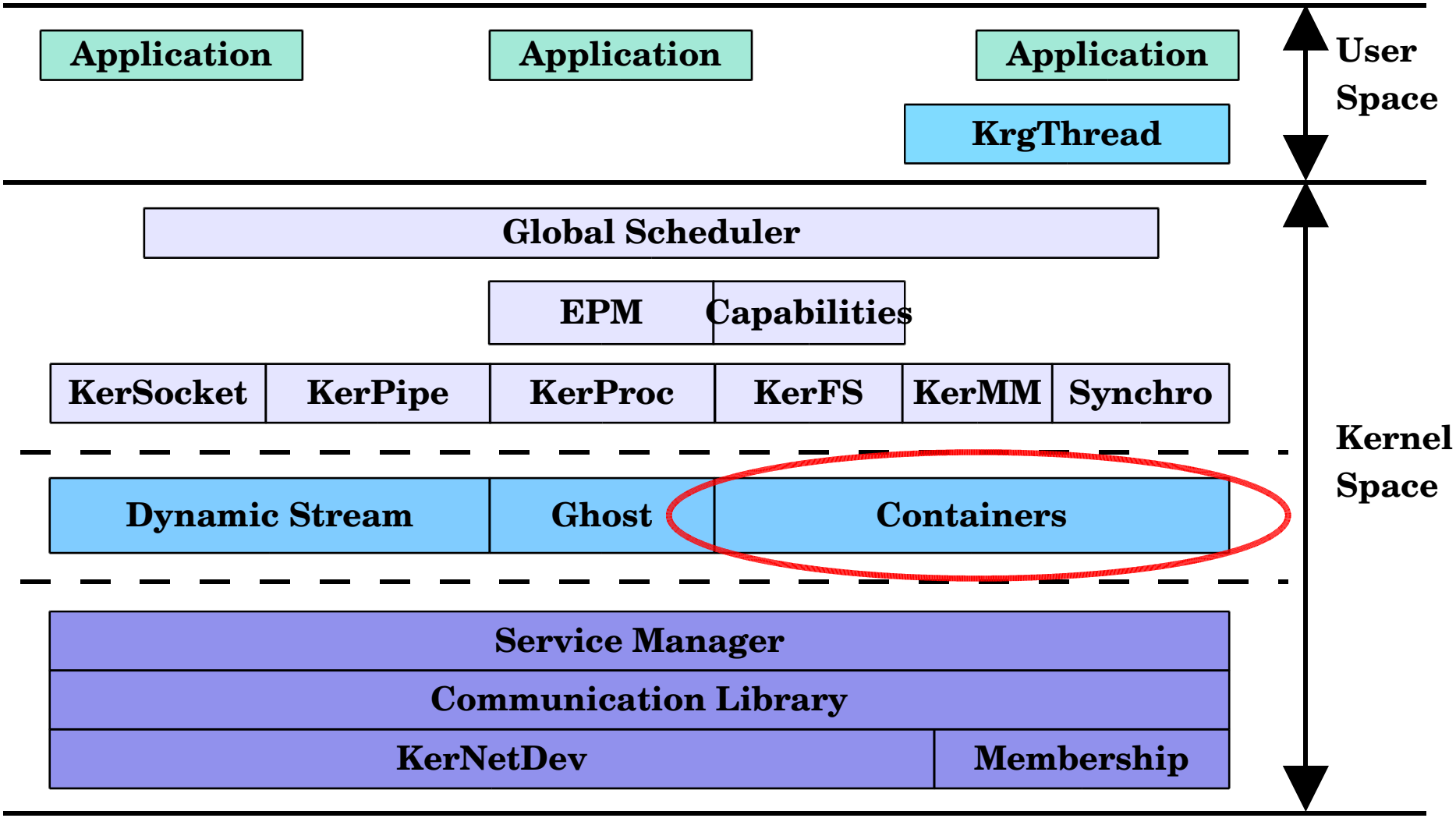
Kerrighed
Linux clusters made easy

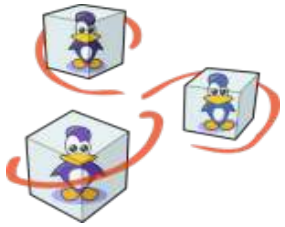


Outline

- Global Process Management
- Containers
- Global memory management
- Dynamic streams
- Capabilities
- Kerrighed in Action !

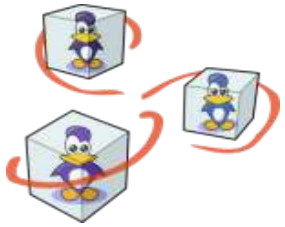
Global View of the Kerrighed Software Architecture





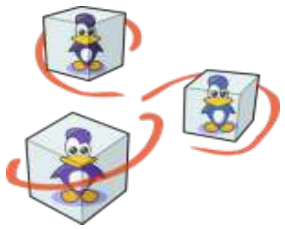
Definition of container

- 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

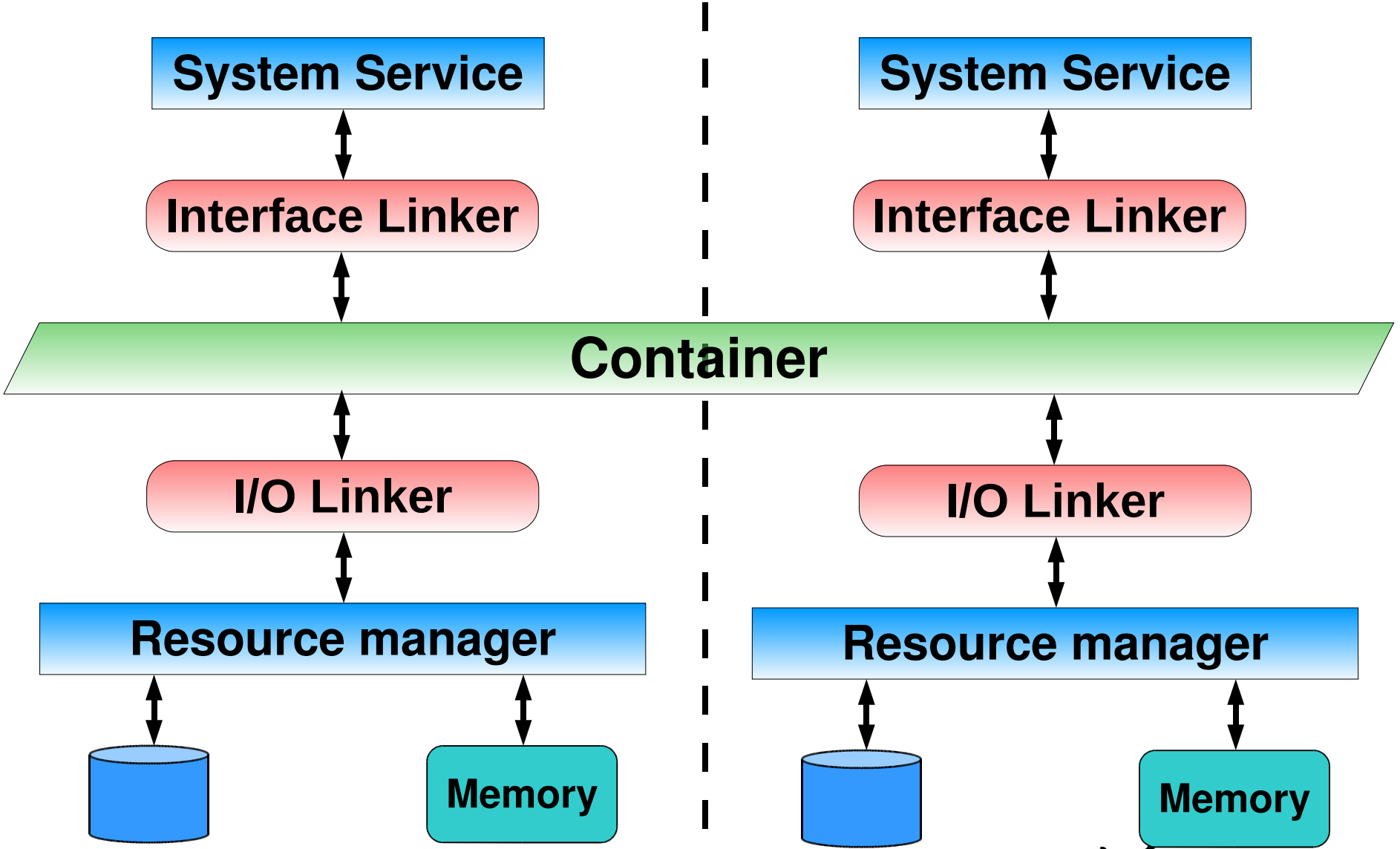


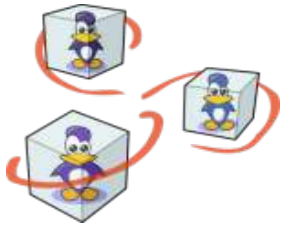
Containers : a Generic Data Sharing Mechanism

- 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



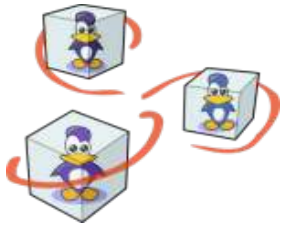
Containers Architecture





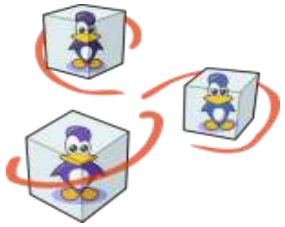
Container interface

- 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



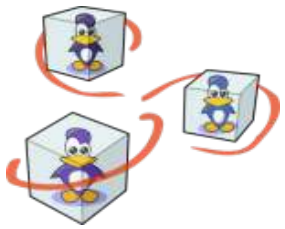
Container interface (2)

- 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

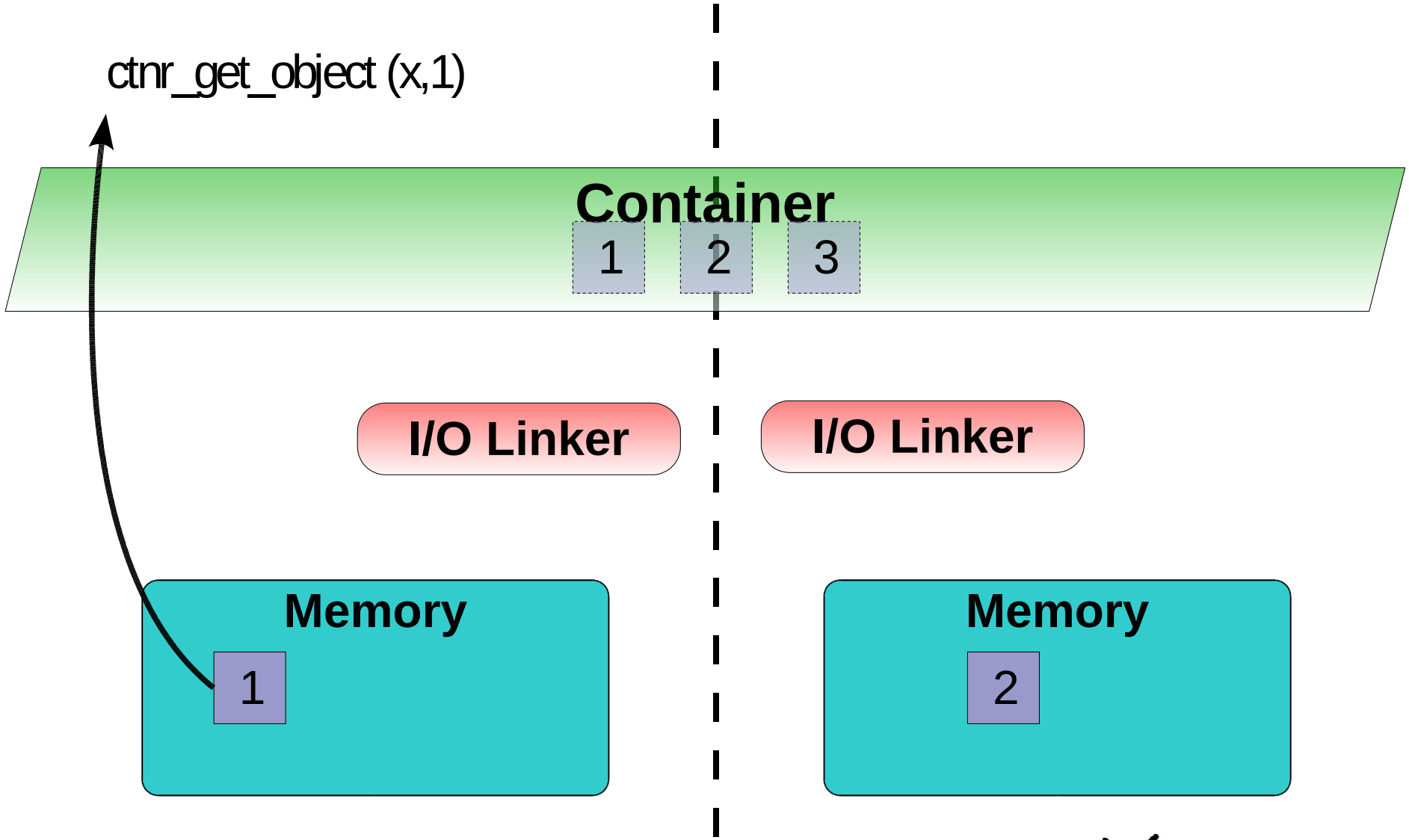


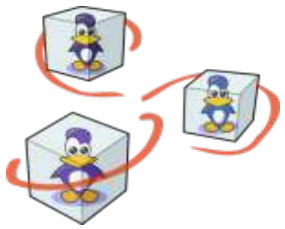
Data input/output in Containers

- 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

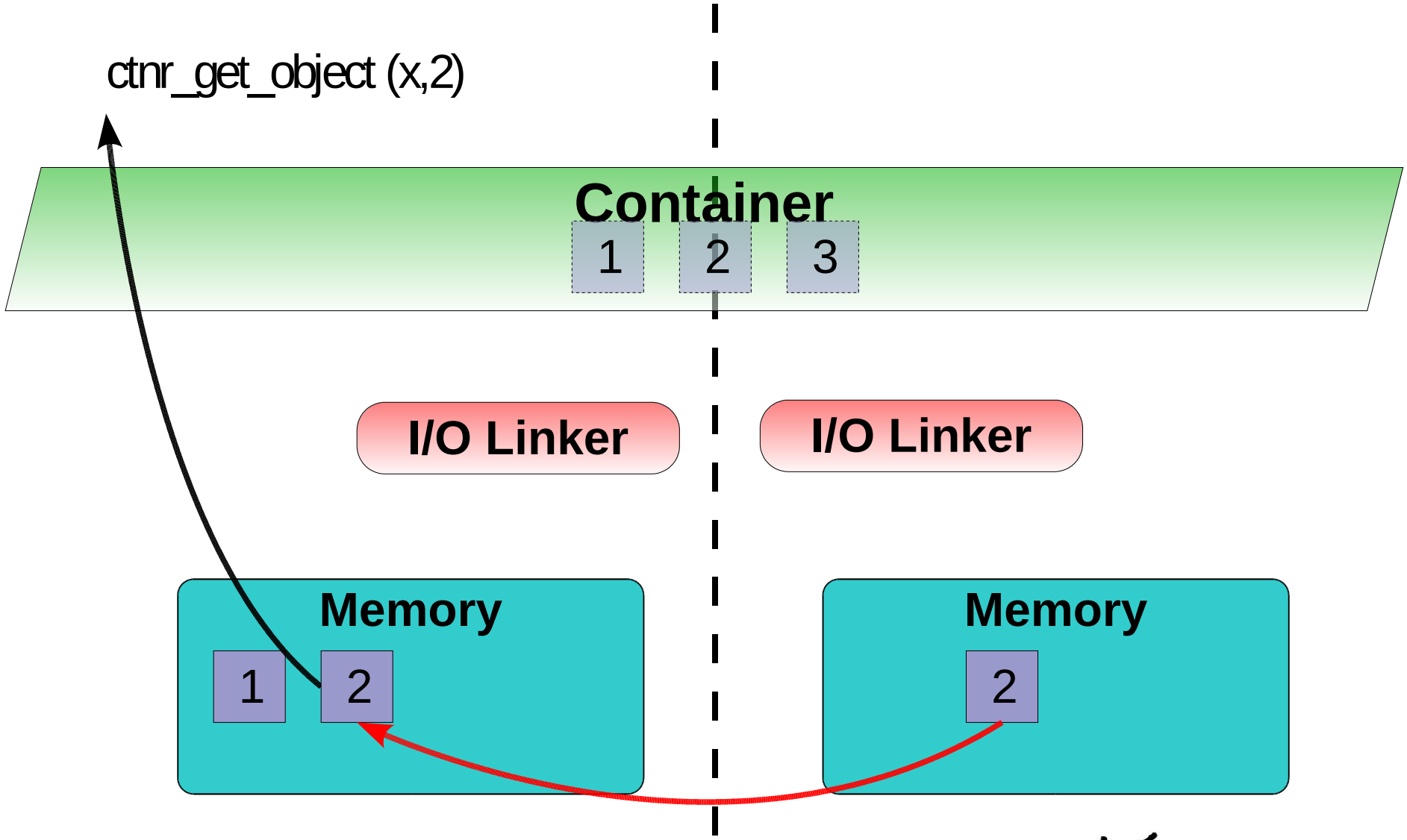


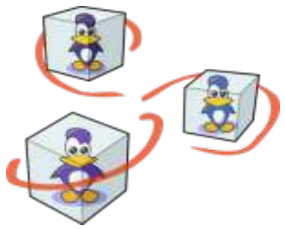
Get Page Example



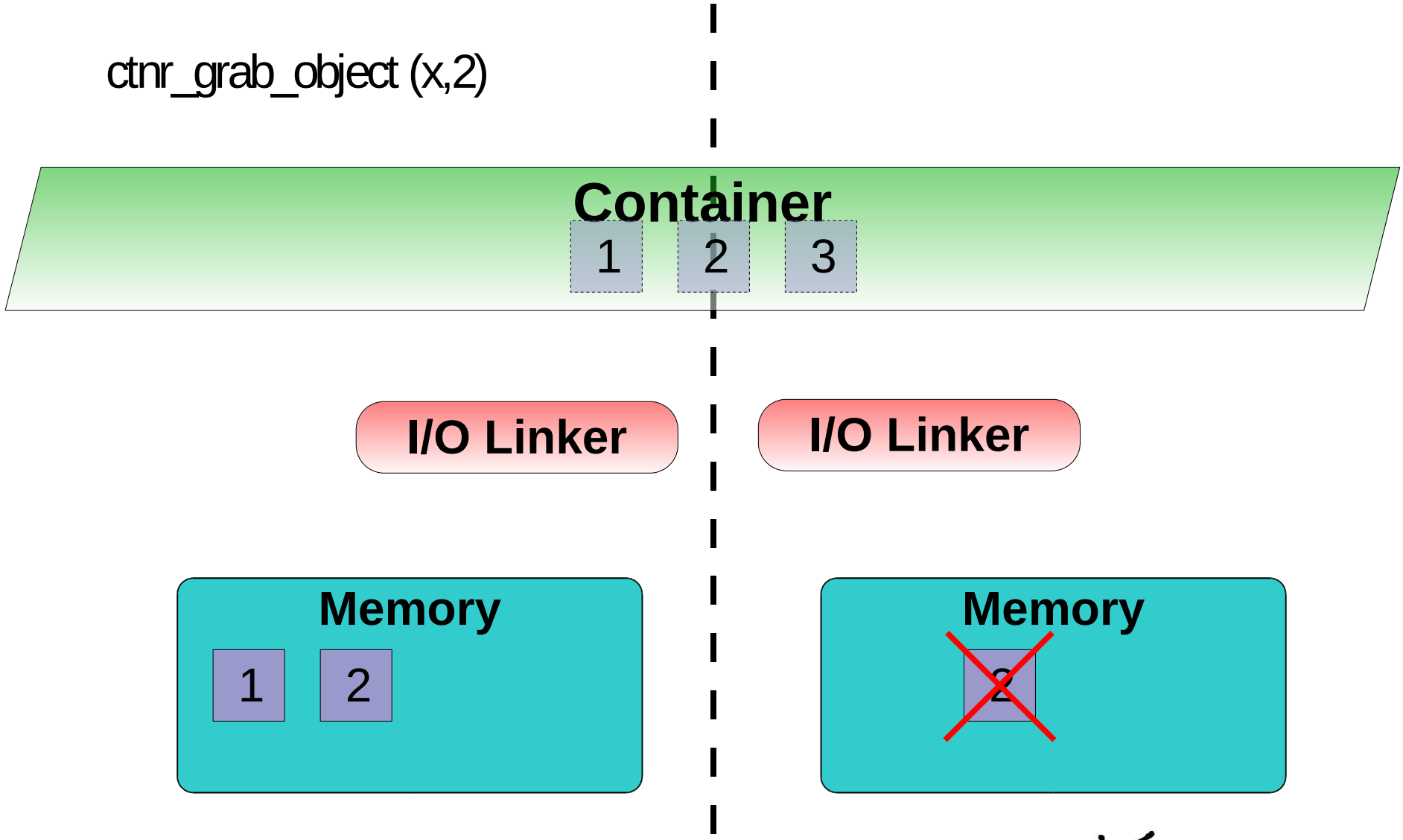


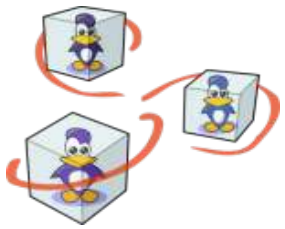
Get Page Example (2)



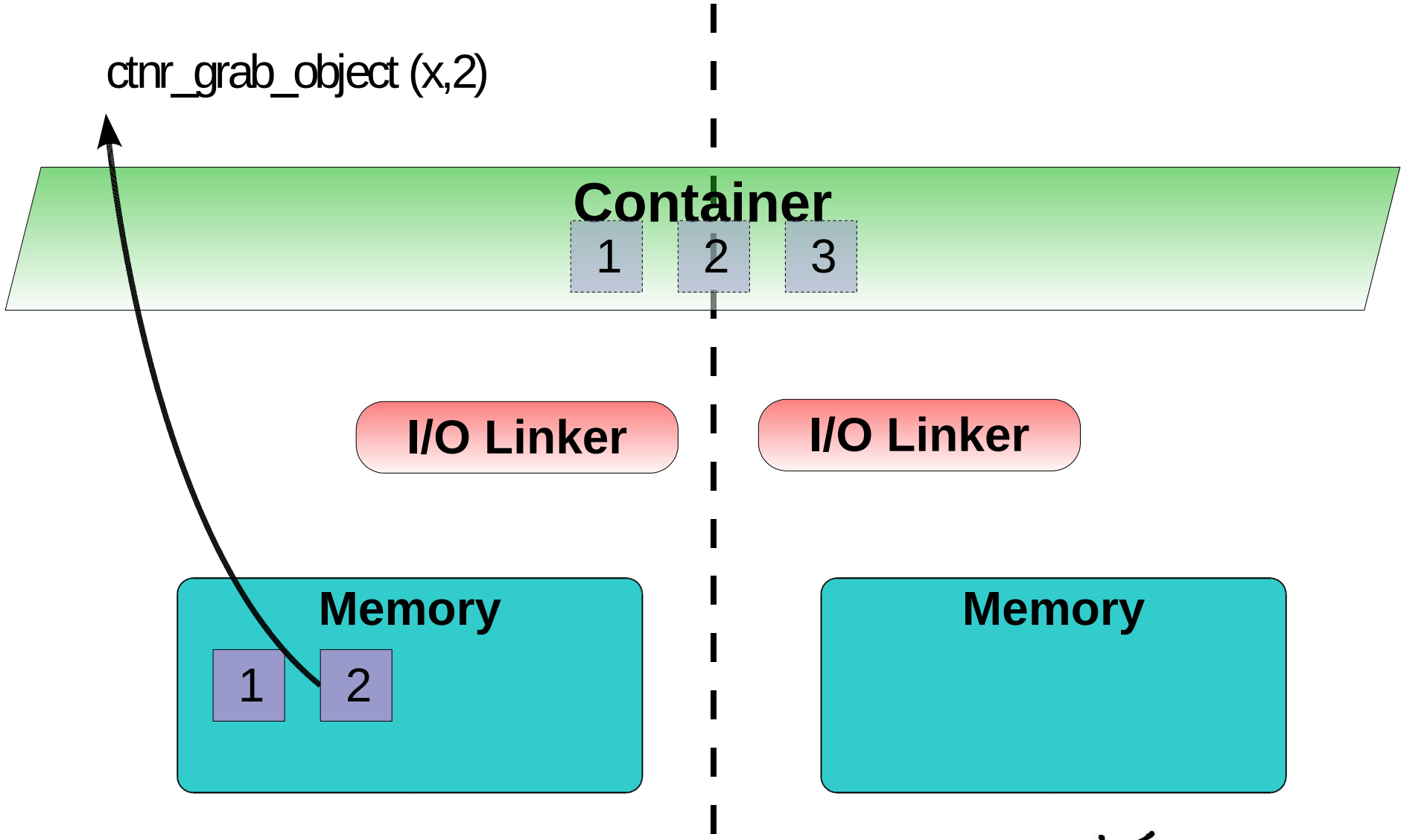


Grab Page Example

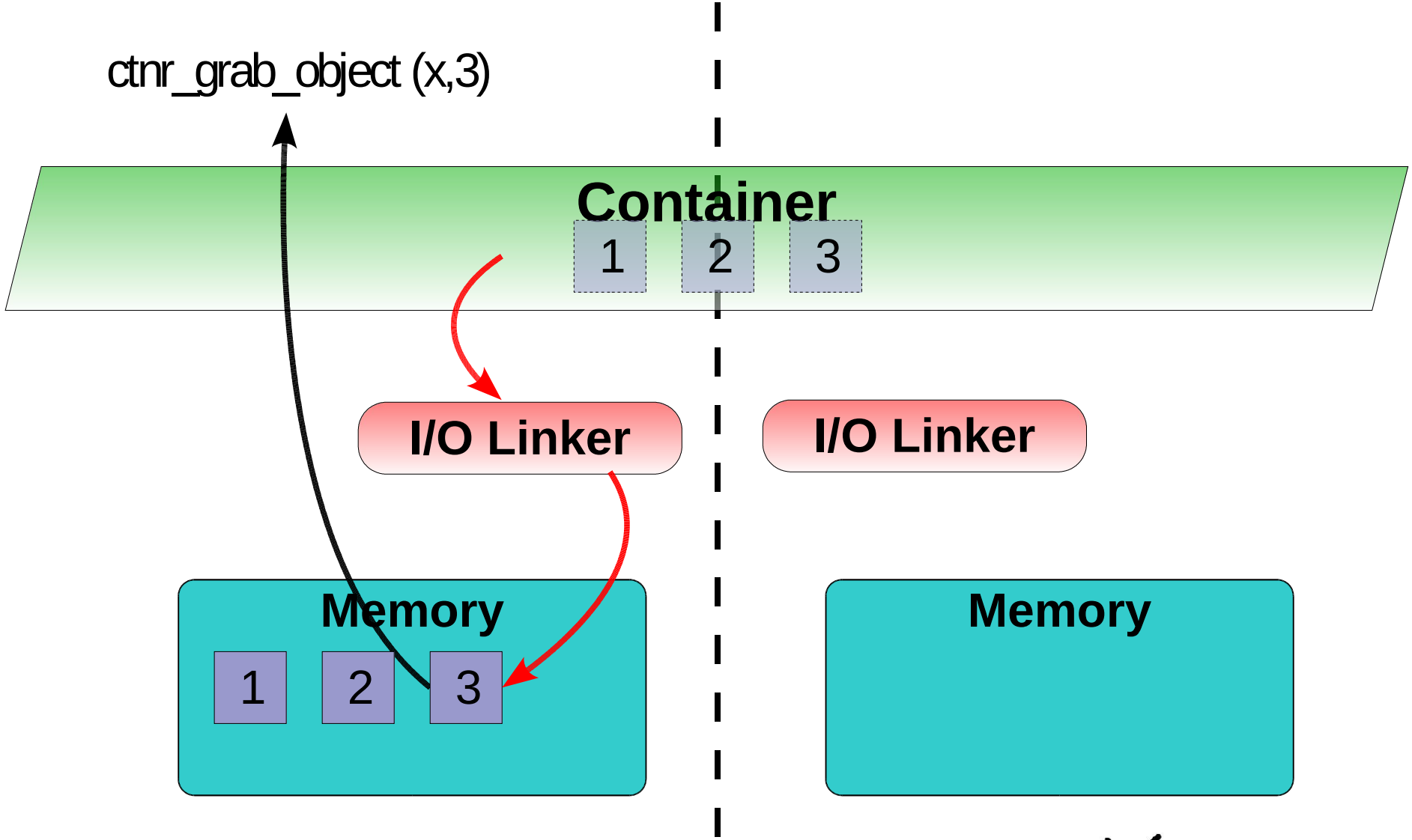


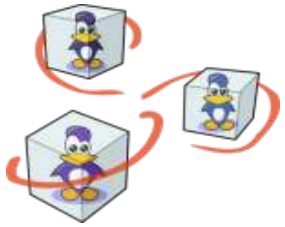


Grab Page Example



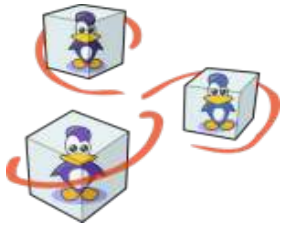
Grab Page Example (2)





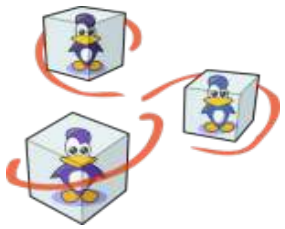
Container use in Kerrighed

- 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
  - Cluster wide locks
  - Signal sharing
  - Etc...

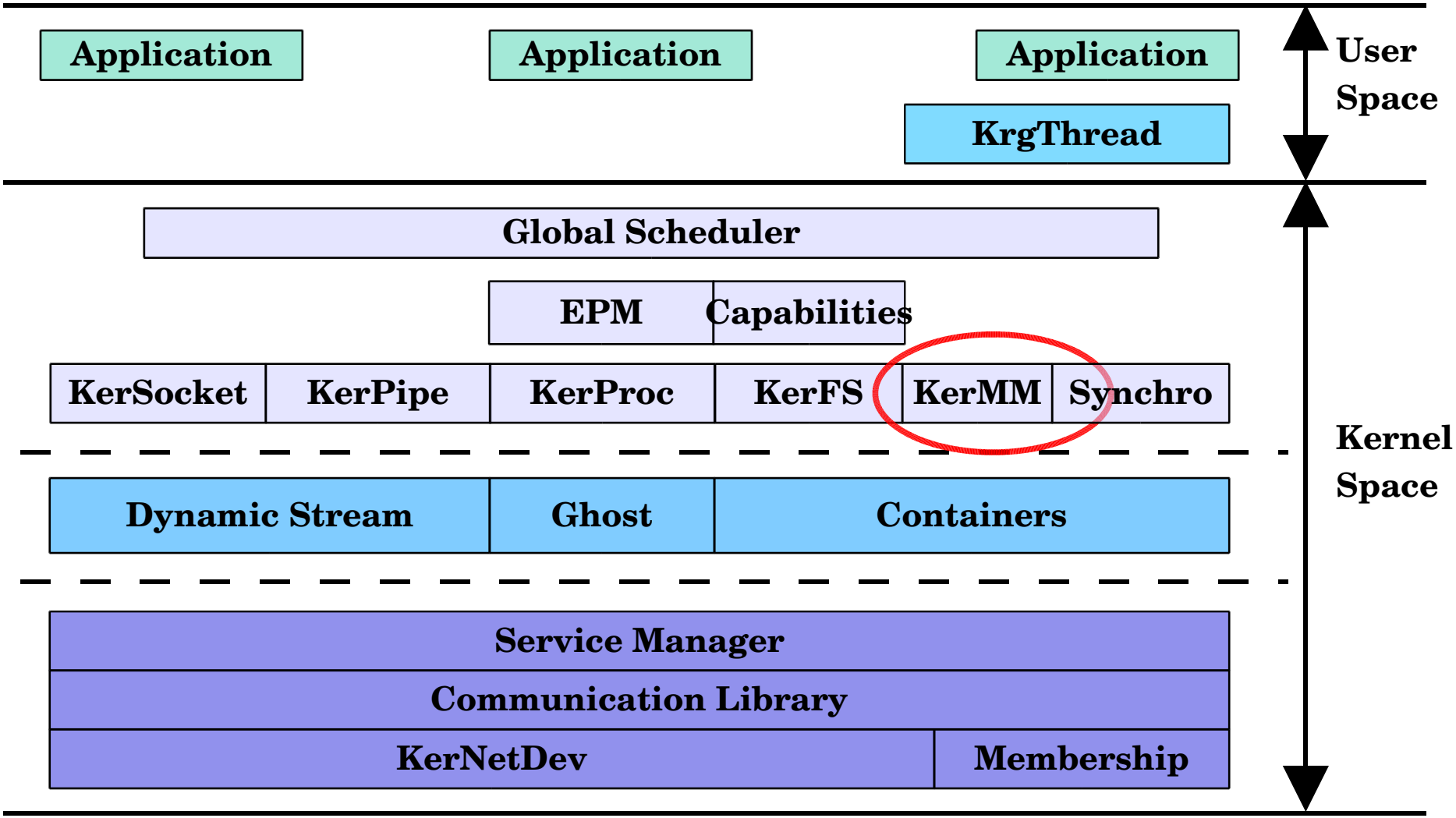


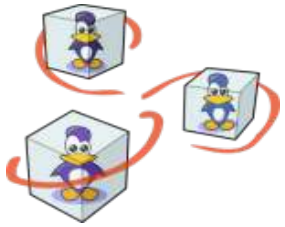
Outline

- Global Process Management
- Containers
- Global memory management
- Dynamic streams
- Capabilities
- Kerrighed in Action !



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 ;
```

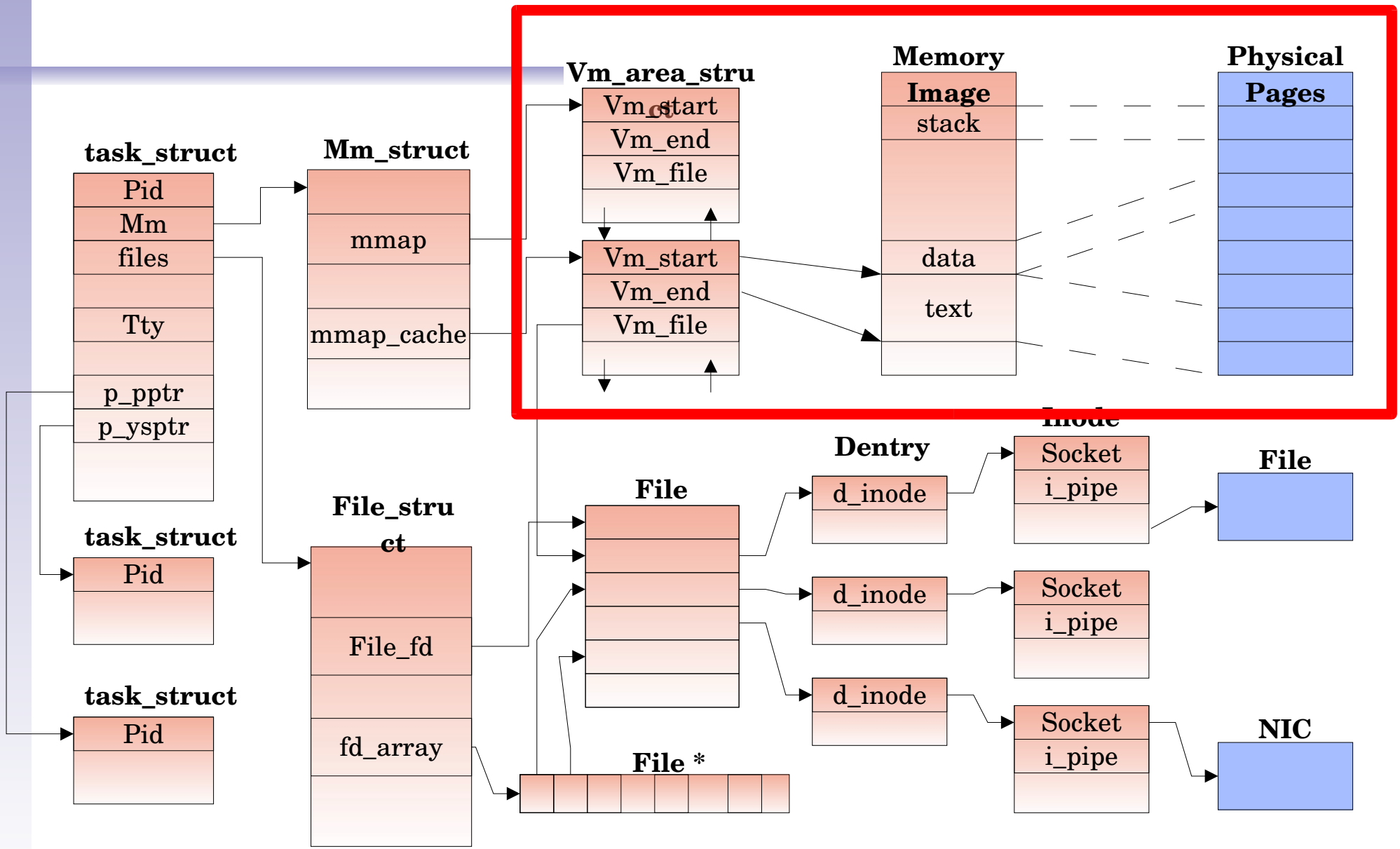
```
Invalidate_Page ( p )
  NOP ;
```

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

Memory Interface Linker

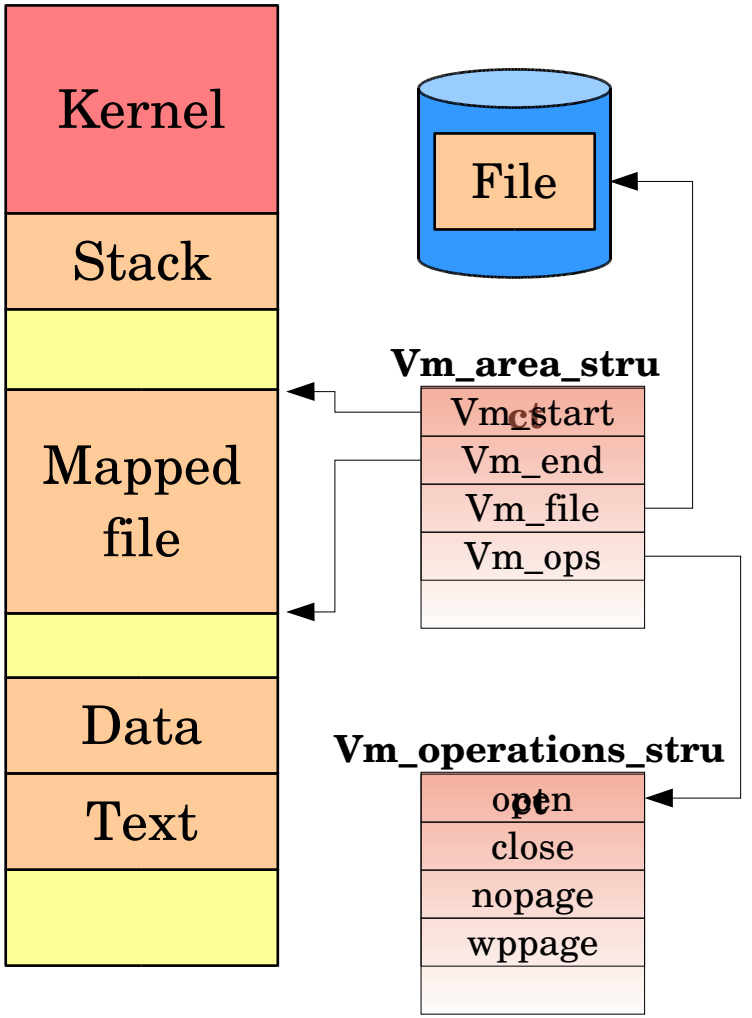
- Goals
  - Link a memory segment to a container
  - Divert page faults to the linked container
- Done at the Virtual Memory Area (VMA) level

Integration in the Linux Kernel

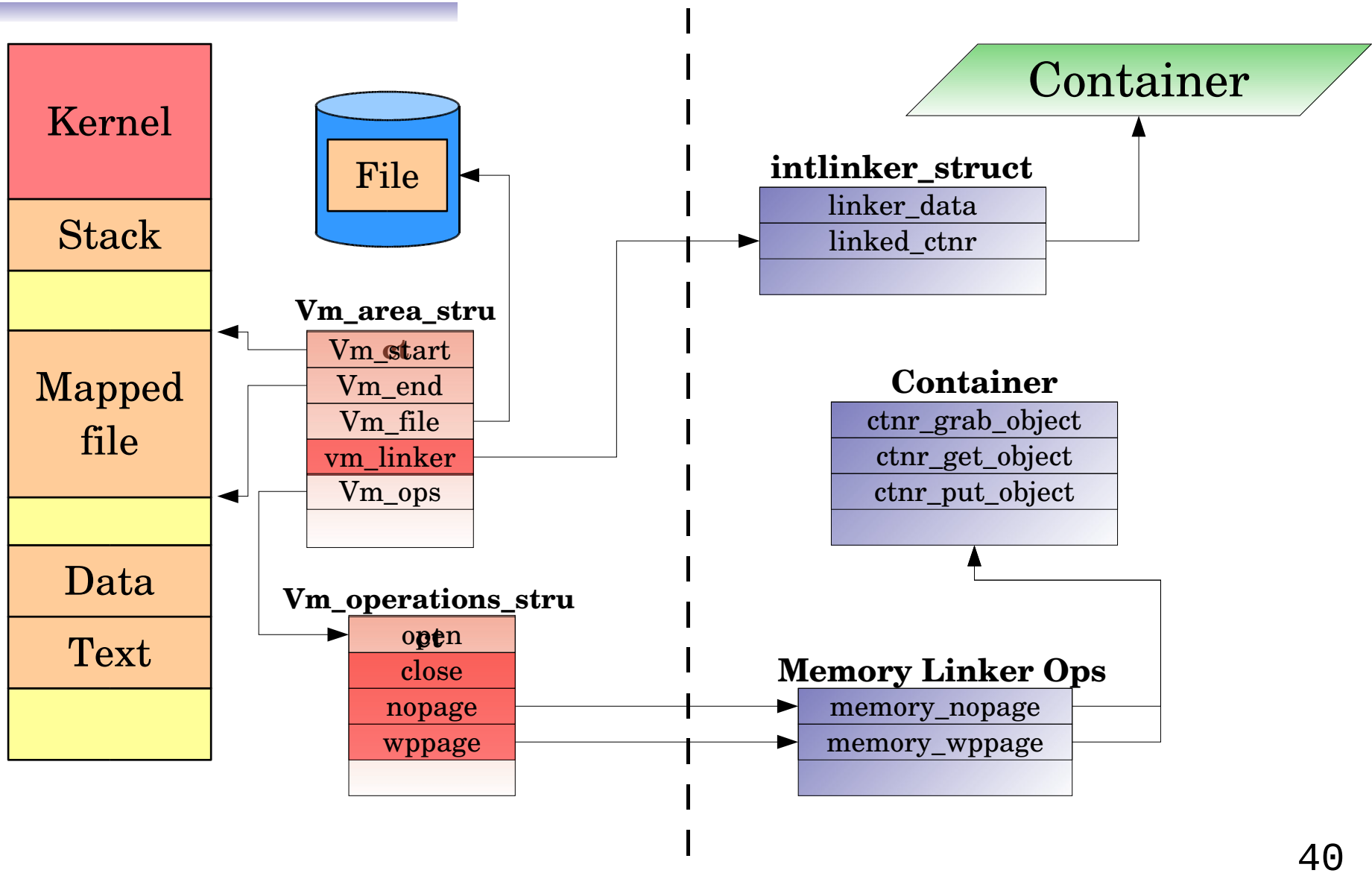


Memory Interface Linker (2)

- 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
  - ...



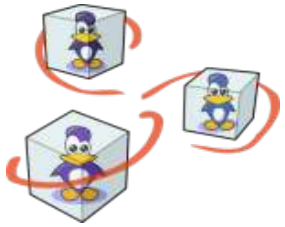
Memory Interface Linker (3)



Memory Interface Linker (4)

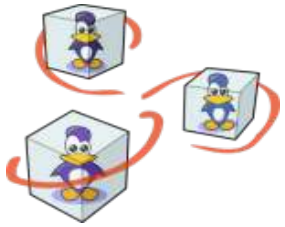
```
memory_nopage ( ctnr, vma, address )
  if (write access)
    page = ctnr_grab_object (ctnr, pageid) ;
  else
    page = ctnr_get_object (ctnr, pageid) ;
  return page ;
```

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



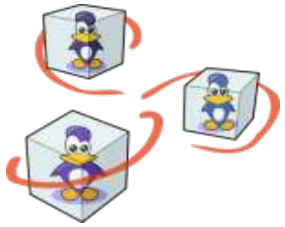
Outline

- Global Process Management
- Containers
- Global memory management
- Dynamic streams
- Capabilities
- Kerrighed in Action !



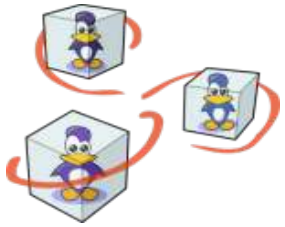
What's the problem ?

- Migrating process may communicate!
- Communications are **inside** of the cluster
- Communications by
  - system calls
  - direct memory access



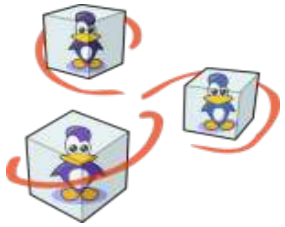
What's the problem ?

- Migrating process may communicate!
- Communications are **inside** of the cluster
- Communications by
  - system calls
    - Unix socket (share a memory page)
    - Pipe/FIFO (share a memory page)
    - INET socket (link to an IP address)



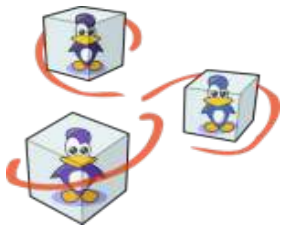
Data streams

- Data streams:
  - data: just bytes running from one node to one node
  - meta-data: any useful informations needed to handle the stream (usually manages by the protocol)
- Meta-data provide:
  - protocol dependent details
  - location of the extremities of the stream



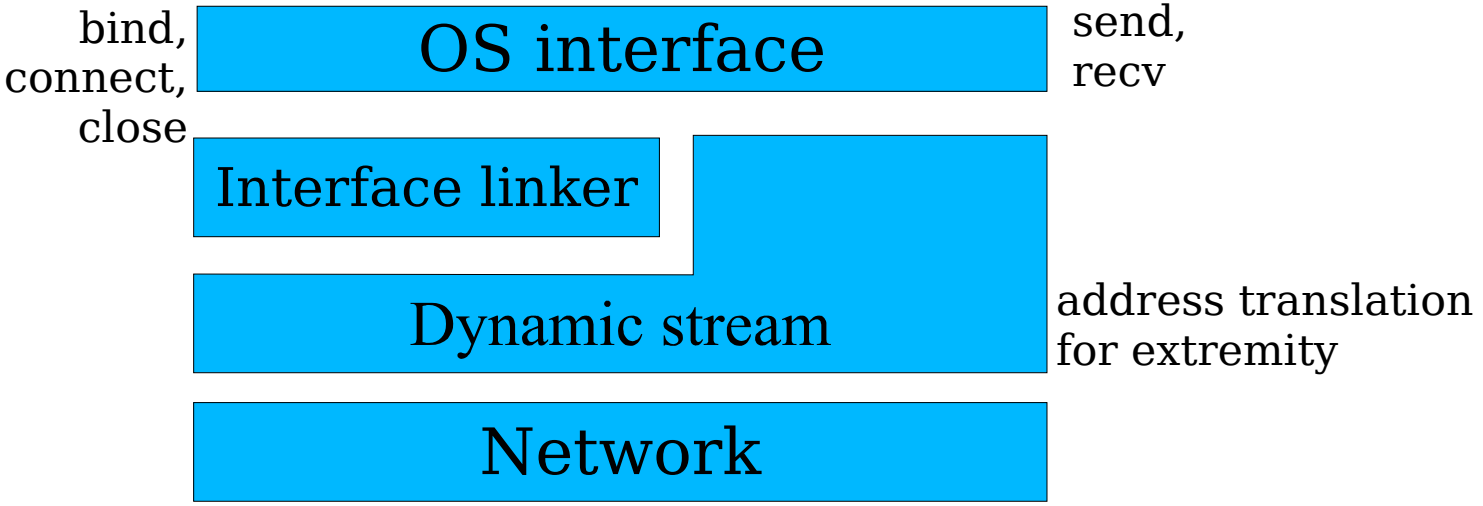
Data streams in SSI

- How to locate an extremity ?
- How to detect the migration of an extremity ?
- How to be efficient ?

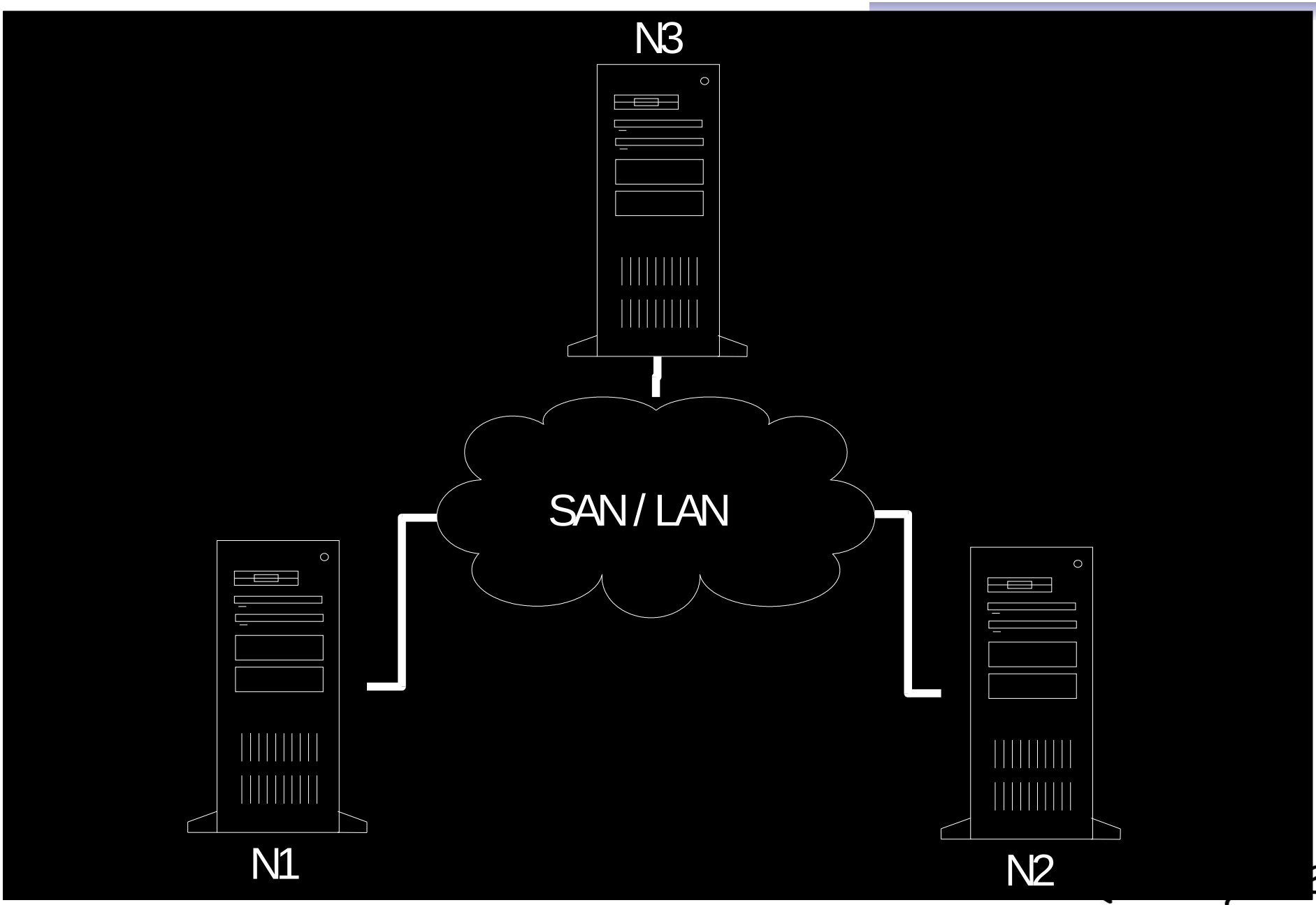


Here is the answer!

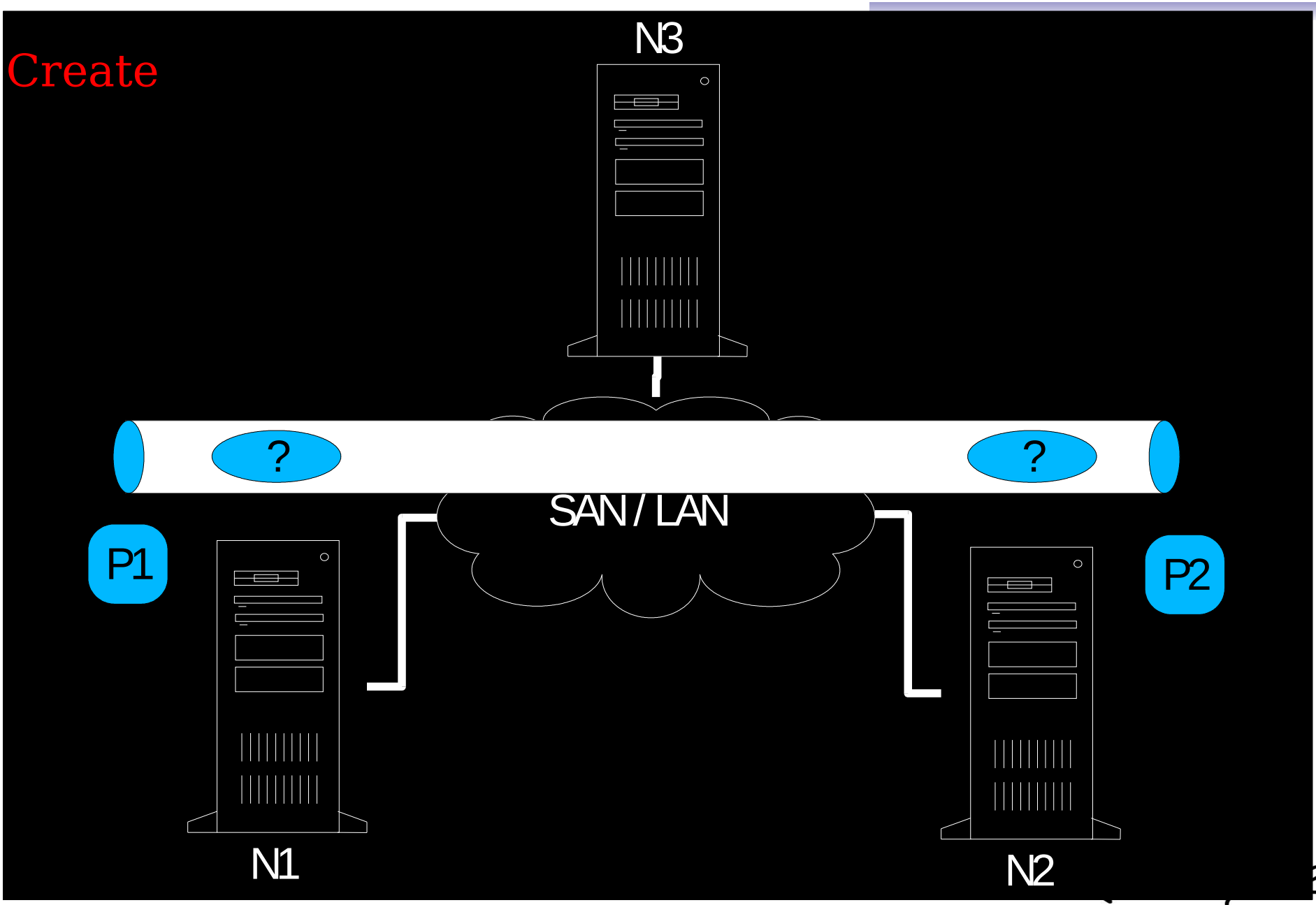
- Split the communication layer:
  - dynamic stream: define how to follow the extremities of a stream
  - Protocol linker: link Kerrighed protocol to the regular Linux OS interface



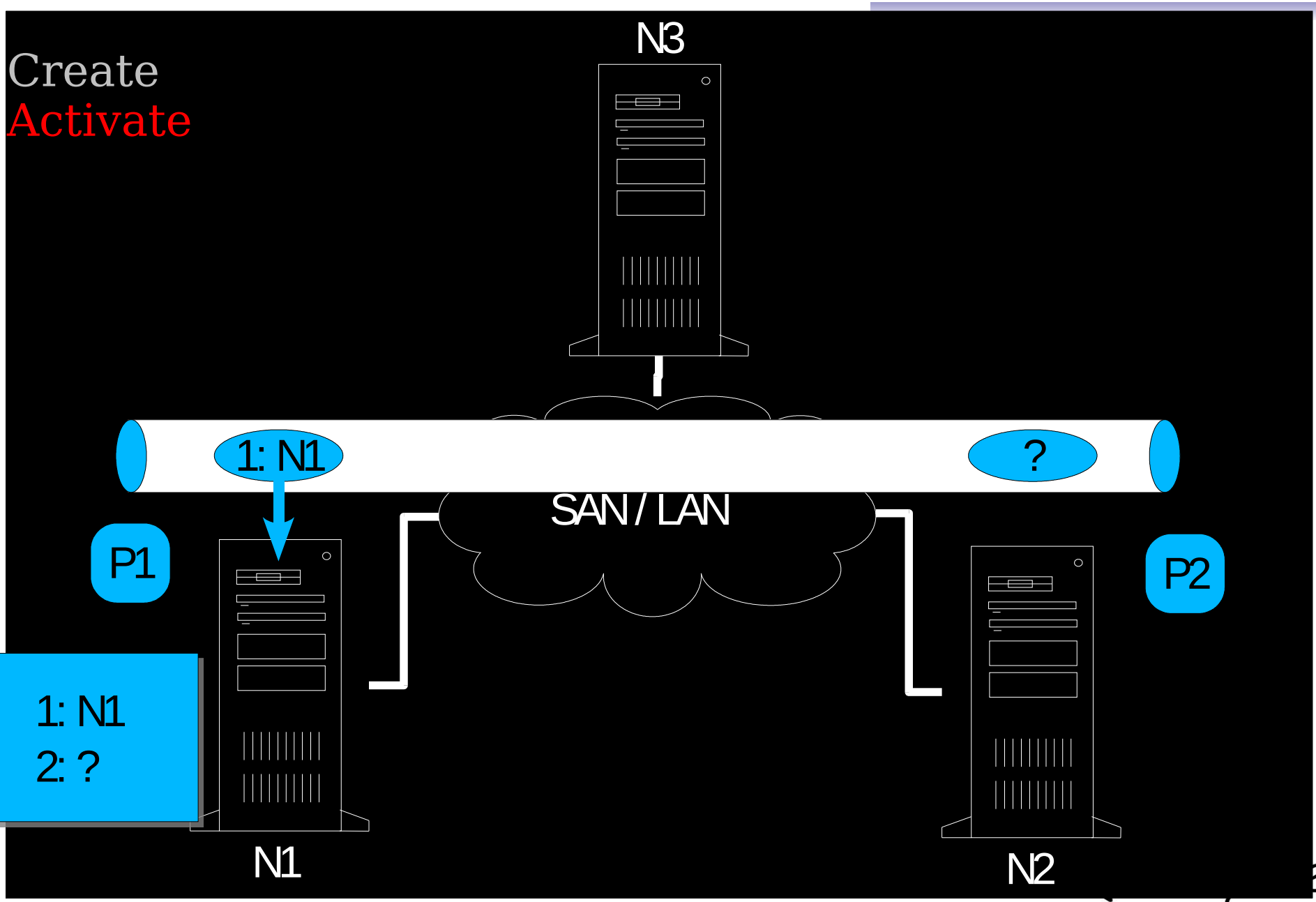
Communicating process migration



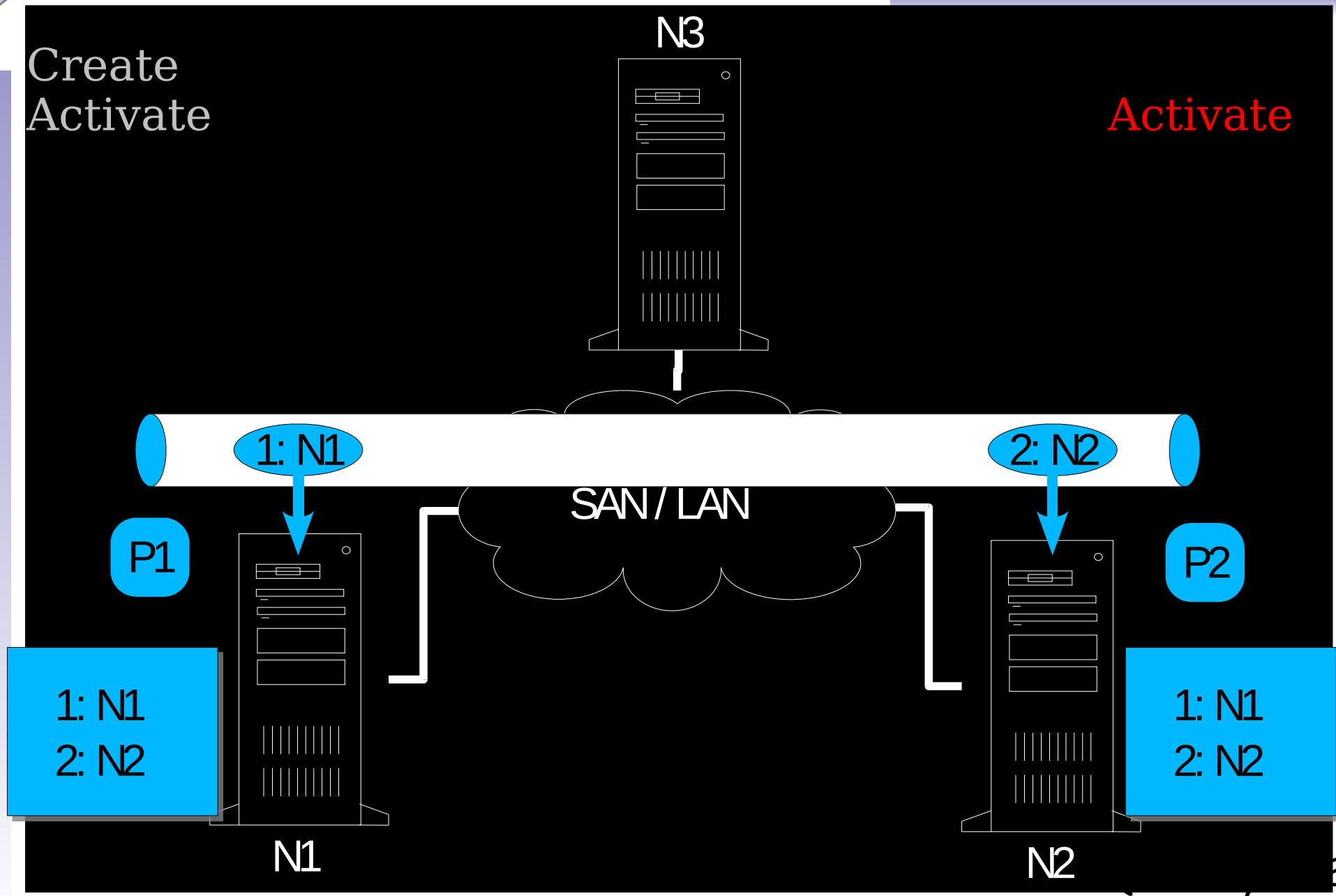
Communicating process migration



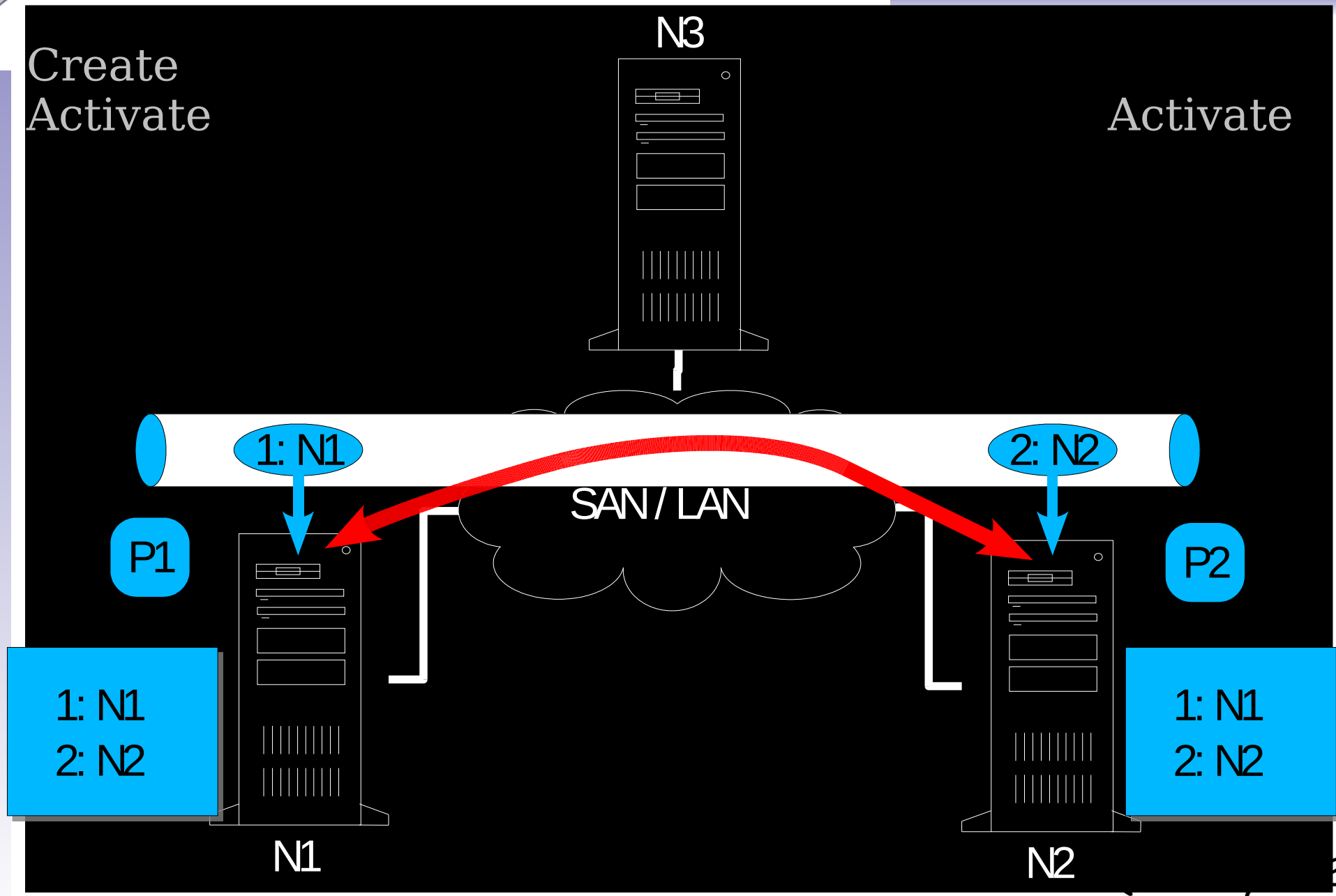
Communicating process migration



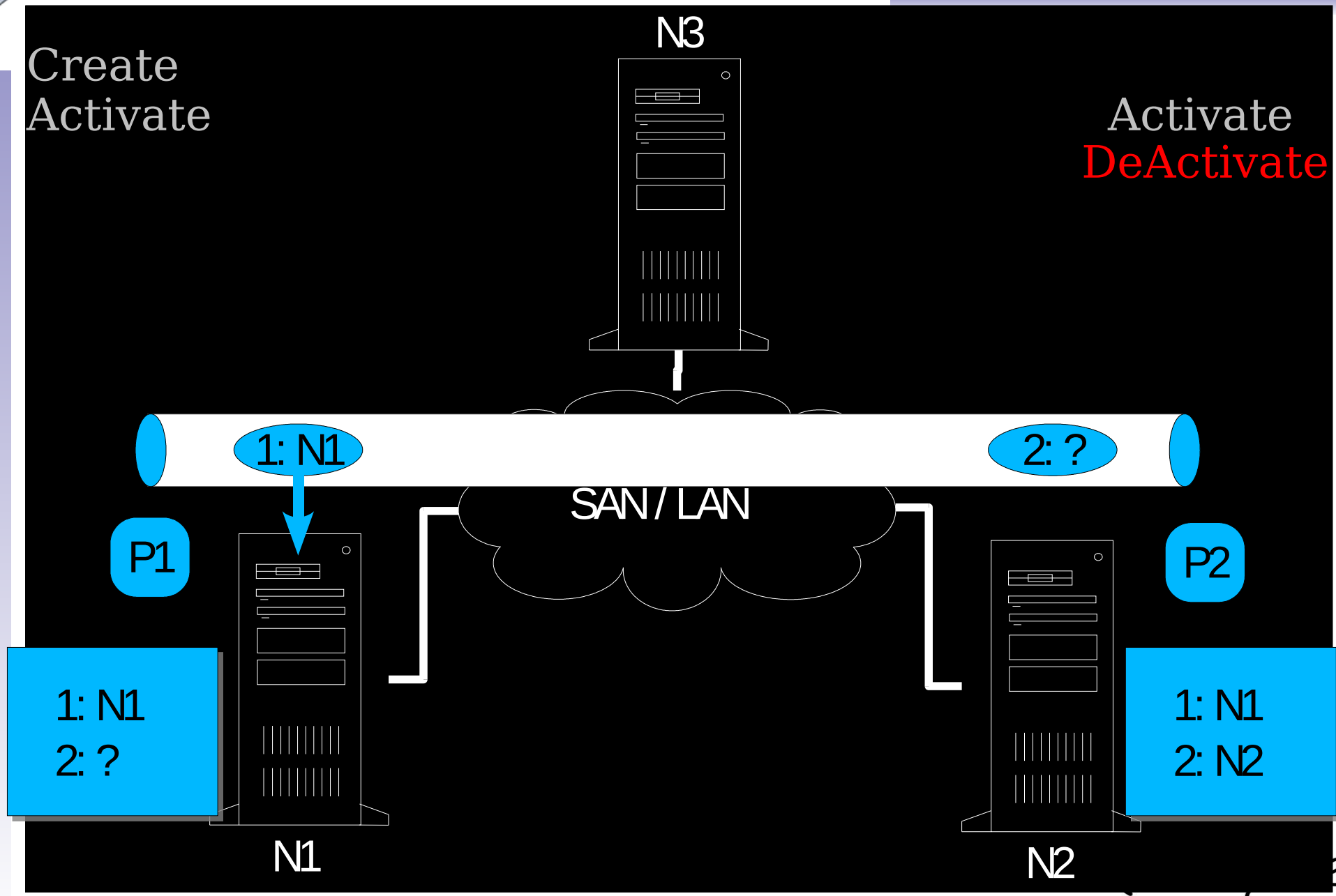
Communicating process migration



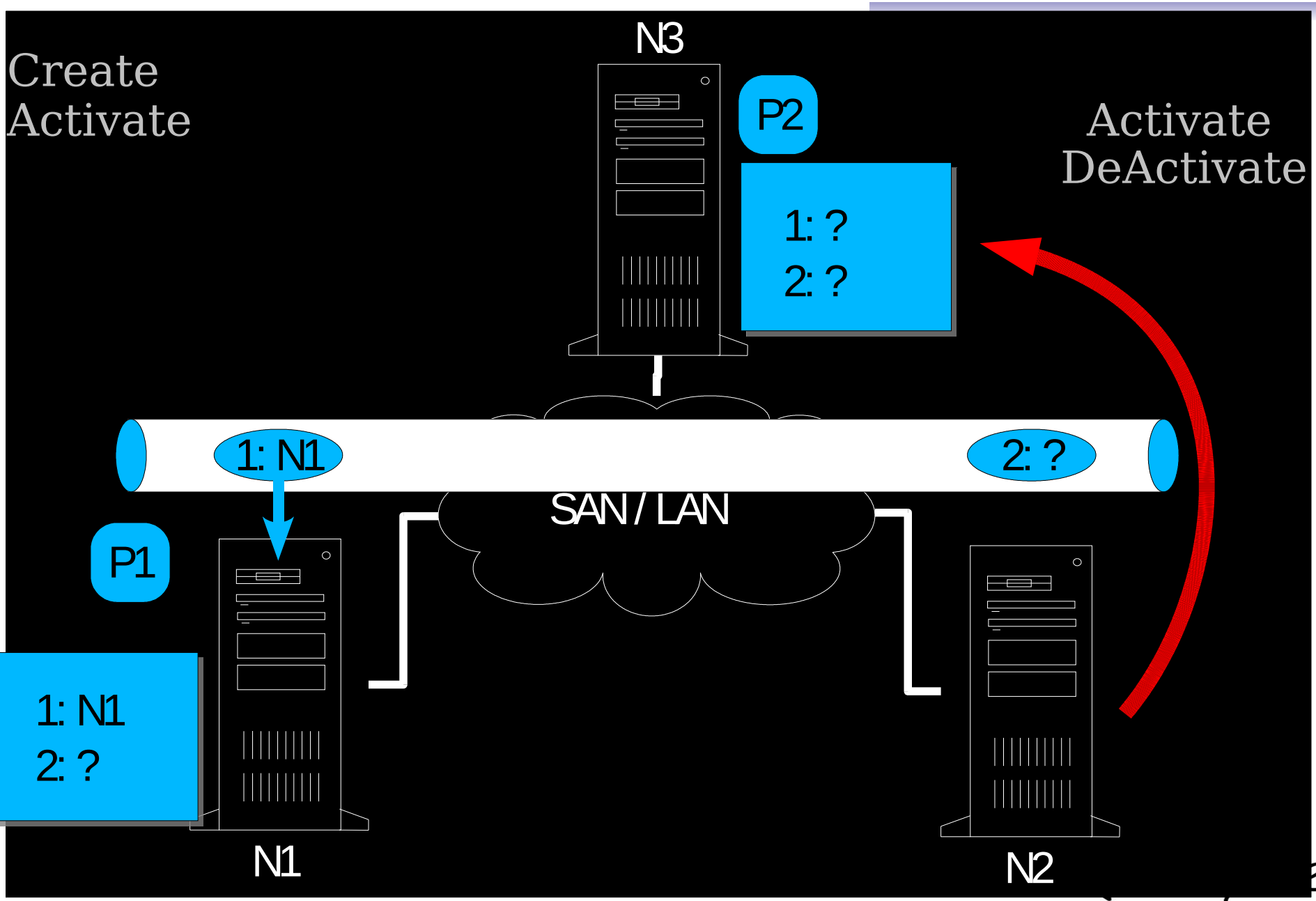
Communicating process migration



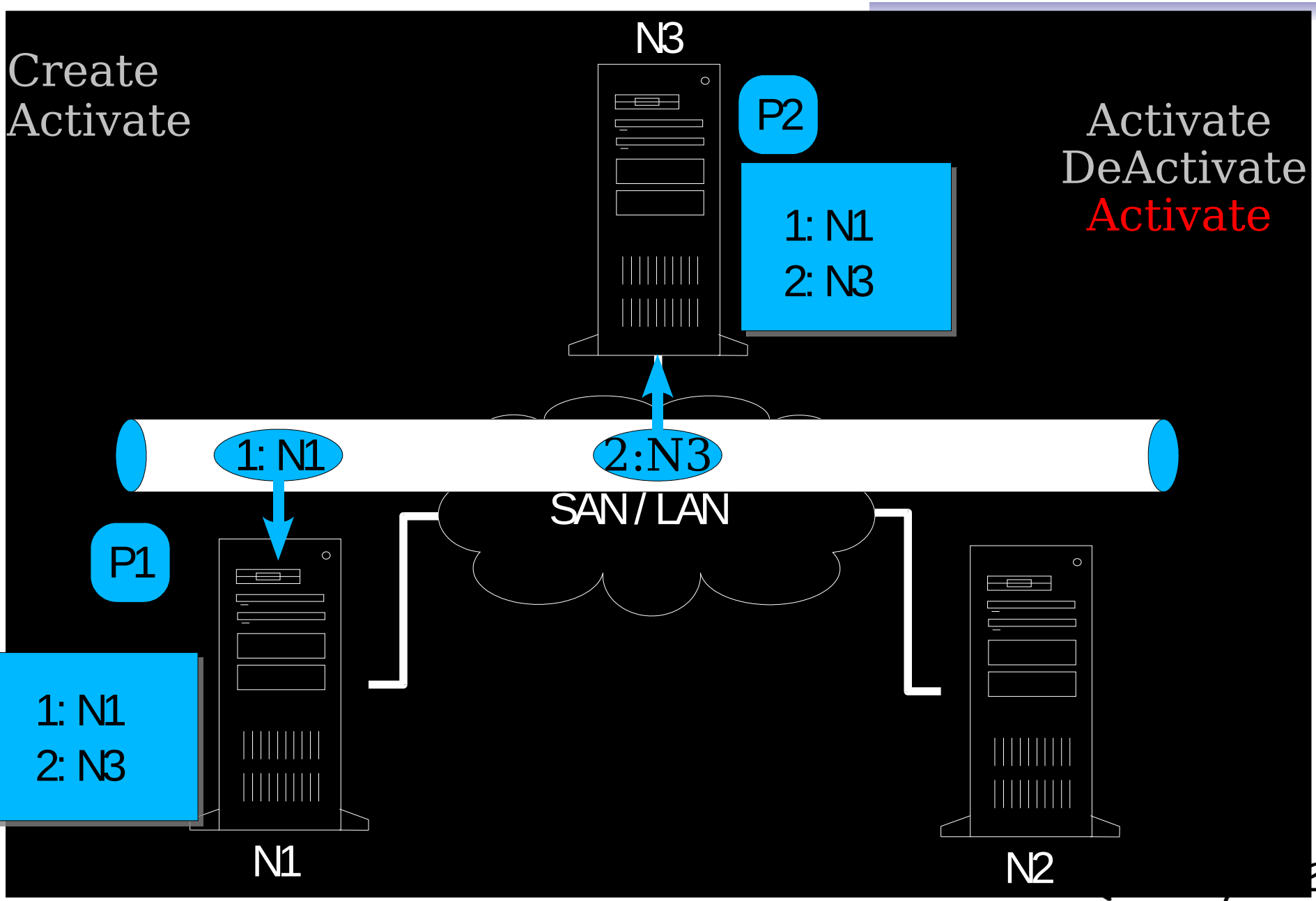
Communicating process migration



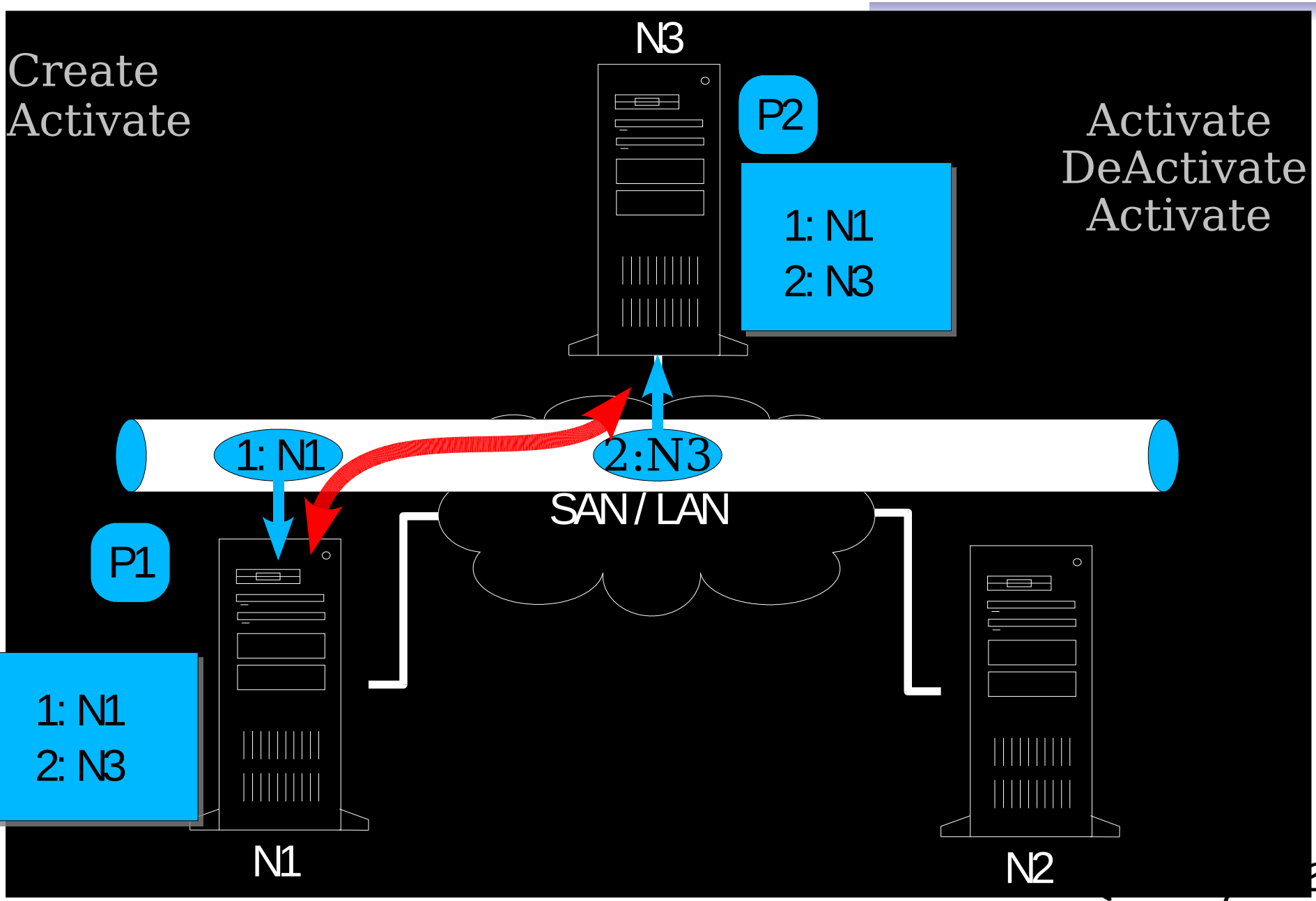
Communicating process migration

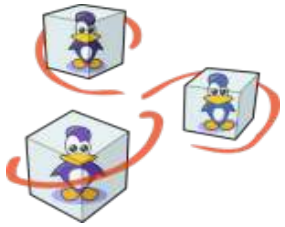


Communicating process migration



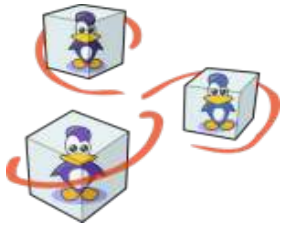
Communicating process migration





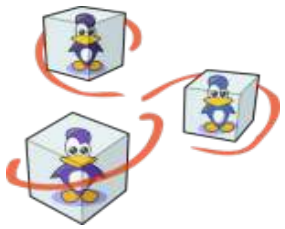
Outline

- Global Process Management
- Containers
- Global memory management
- Dynamic streams
- Capabilities
- Kerrighed in Action !



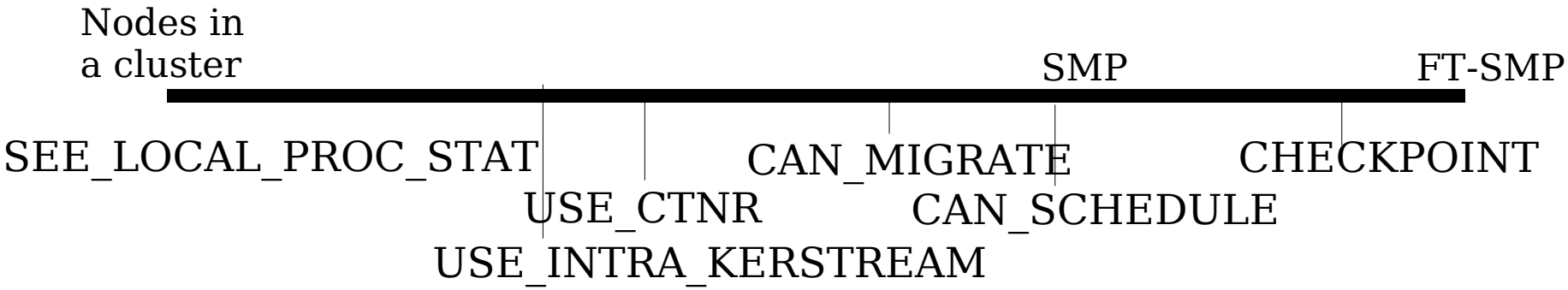
Capabilities: SSI fit your needs

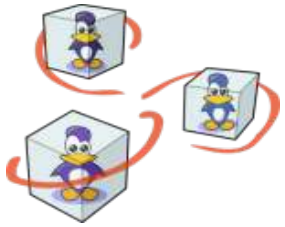
- Kerrighed provides lot of features:
  - global process ID
  - remote process creation
  - process migration
  - global scheduling
  - checkpoint/restart of process
  - cluster-wide stream layer
  - ...
- Not all of them are useful at one time



Capabilities: SSI fit your needs

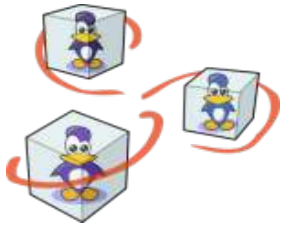
- Kerrighed provides lot of features
- Not all of them are useful at one time
- Kerrighed capabilities: enable SSI features on a per-process way.





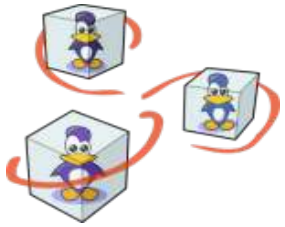
Capabilities design

- Per process architecture
- Each process manage several sets of capabilities:
  - own enabled capabilities
  - own authorized capabilities
  - default enabled capabilities for its sons
  - default authorized capabilities for its sons



Outline

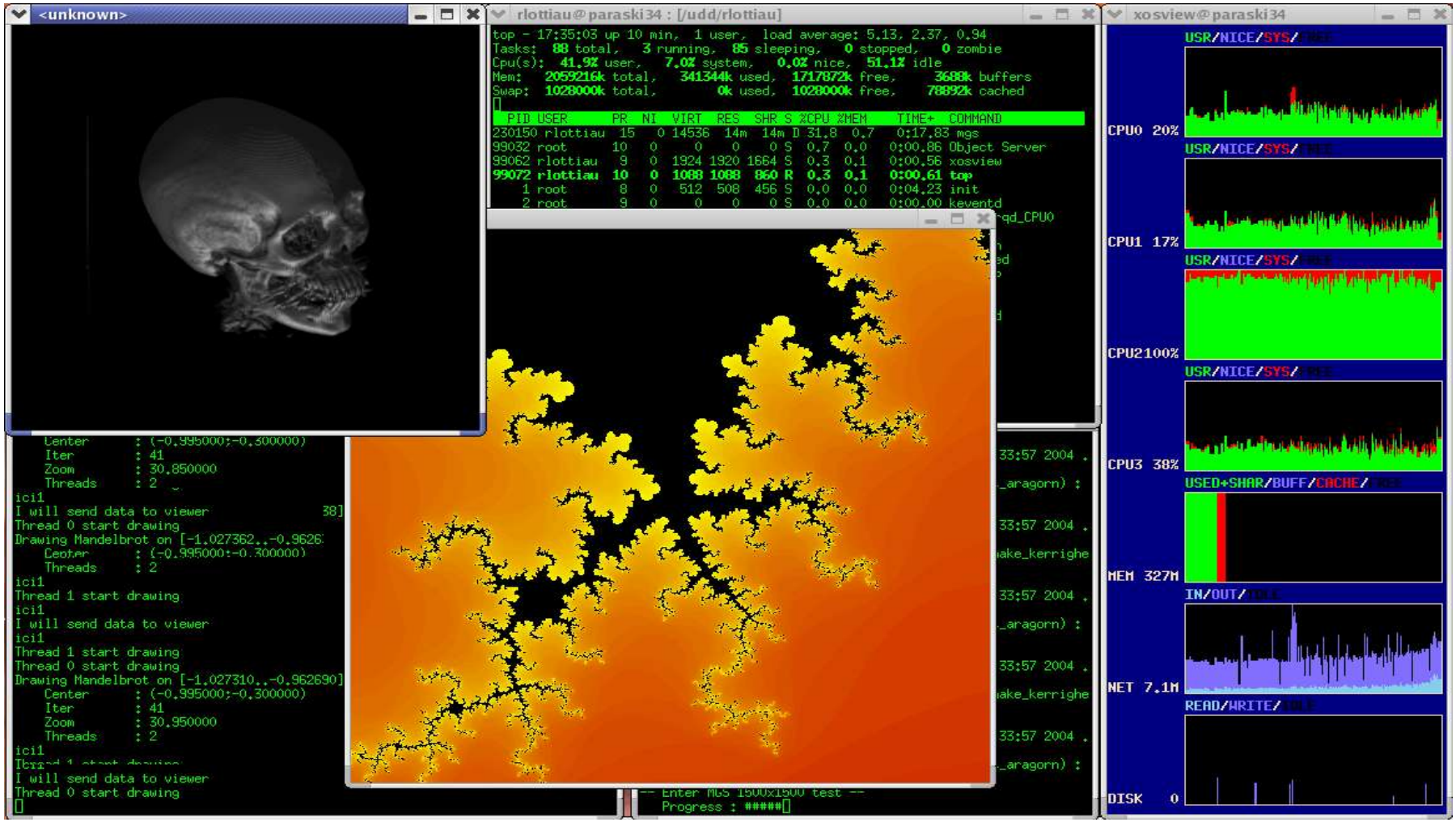
- Global Process Management
- Containers
- Global memory management
- Dynamic streams
- Capabilities
- Kerrighed in Action !



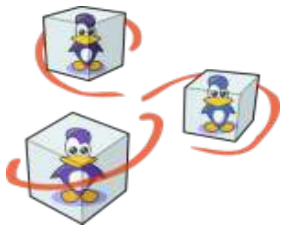
What is Kerrighed Good for ?

- Balancing CPU load
  - Large number of jobs to run
  - Parallel compilation
- Message passing parallel applications
  - Transparent use of Kerrighed high speed comms
  - Configure MPICH just like an SMP machine
- Shared memory parallel applications
  - Spread threads over cluster nodes
- Ease your life !
  - One big SMP machine

Kerrighed in Action !



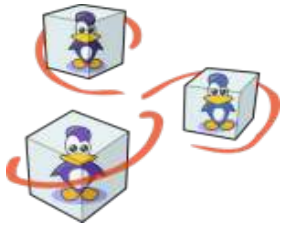
Kerrighed
Linux clusters made easy



Capabilities: an example

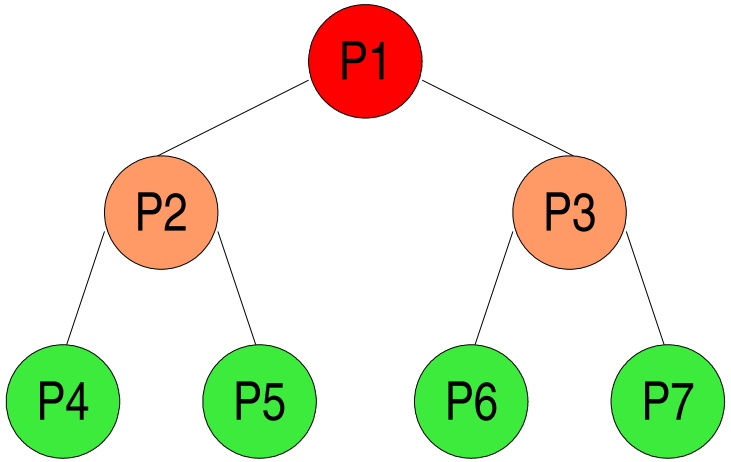
```
paraski33% krg_capset -l
  Effective Capabilities: 043
    CHANGE_KERRIGHED_CAP, USE_CONTAINERS
  Effective Capabilities: 043
    CHANGE_KERRIGHED_CAP, USE_CONTAINERS

paraski33% cat /proc/cpuinfo
  processor       : 0
  vendor_id       : GenuineIntel
  cpu family      : 15
  ...
  processor       : 1
  vendor_id       : GenuineIntel
  cpu family      : 15
  ...

paraski33% cat /proc/meminfo
  MemTotal:      1032644 kB
  MemFree:        831052 kB
  ...

paraski33% cat /proc/meminfo
  MemTotal:      1032644 kB
  MemFree:        831052 kB
  ...
```

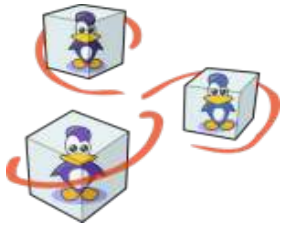
```
paraski33% krg_capset -d +SEE_LOCAL_PROC_STAT
paraski33% cat /proc/cpuinfo
  processor       : 0
  vendor_id       : GenuineIntel
  cpu family      : 15
  ...

paraski33% cat /proc/meminfo
  MemTotal:       498944 kB
  MemFree:        457936 kB
  ...
```



Capabilities: another example

```
void main(void)
{
    int nb_sons = 0;
    int depth = 0;

    while (nb_sons != 2 && depth < 2) {
        if (fork())
            nb_sons++;
        else {
            depth++;
            nb_sons = 0;
        }
    }
}
```

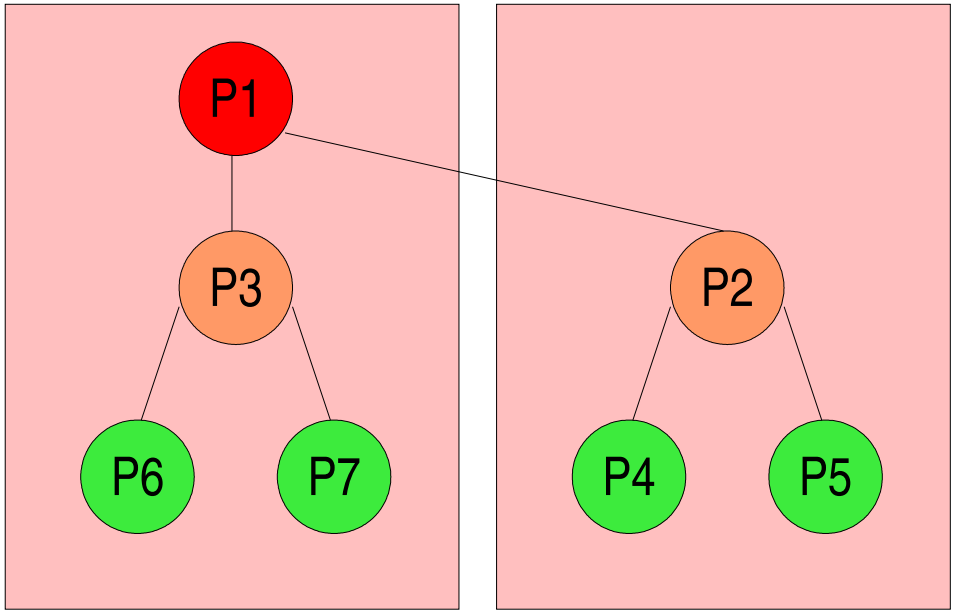


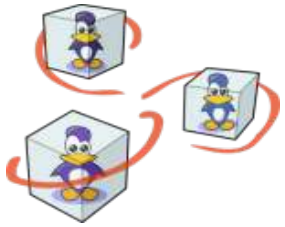


Capabilities: another example

```
void main(void)
{
     int nb_sons = 0;
     int depth = 0;

     while (nb_sons != 2 && depth < 2) {
          if (fork())
               nb_sons++;
          else {
               depth++;
               nb_sons = 0;
          }
     }
}
```

```
paraski33% krg_capset -e +DISTANT_FORK
paraski33% ./fork-test
```

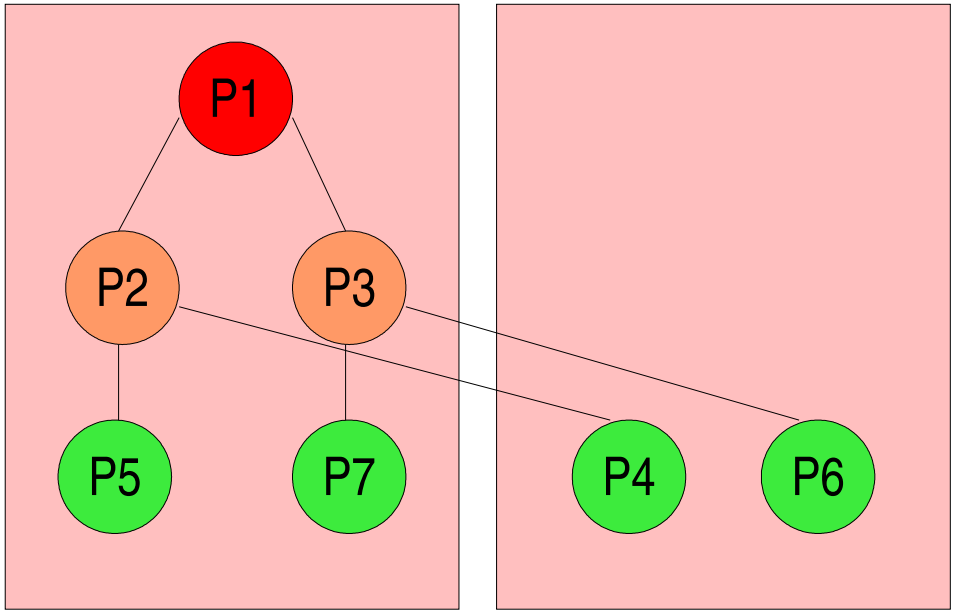


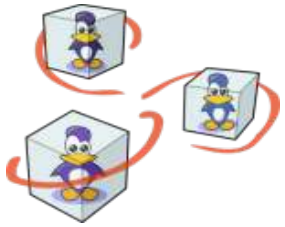


Capabilities: another example

```
void main(void)
{
     int nb_sons = 0;
     int depth = 0;

     while (nb_sons != 2 && depth < 2) {
          if (fork())
               nb_sons++;
          else {
               depth++;
               nb_sons = 0;
          }
     }
}
```

```
paraski33% krg_capset -d +DISTANT_FORK
paraski33% ./fork-test
```

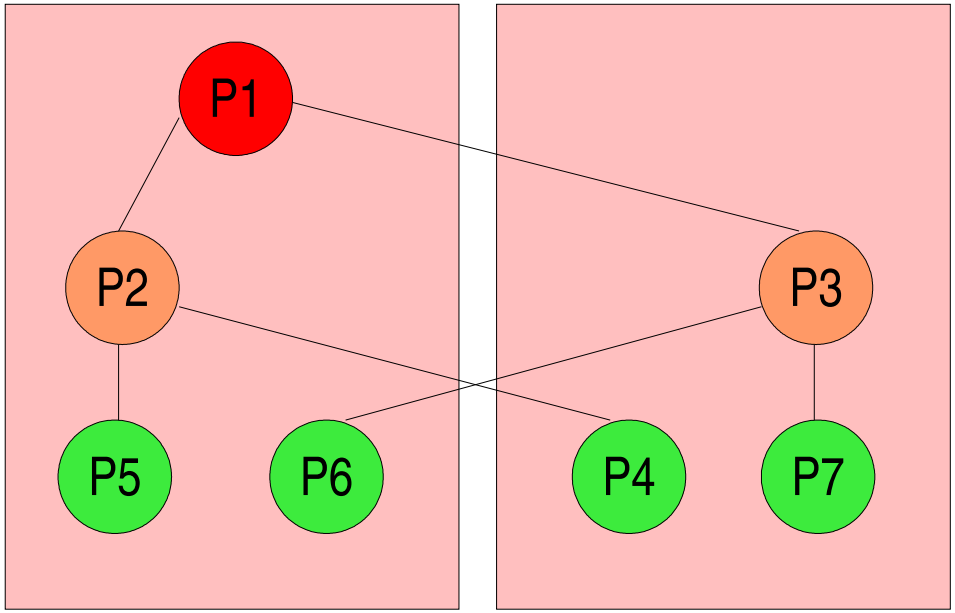


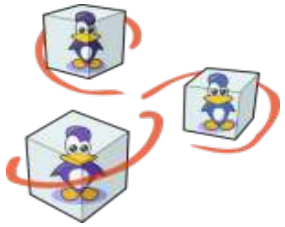


Capabilities: another example

```
void main(void)
{
    int nb_sons = 0;
    int depth = 0;

    while (nb_sons != 2 && depth < 2) {
        if (fork())
            nb_sons++;
        else {
            depth++;
            nb_sons = 0;
        }
    }
}
```

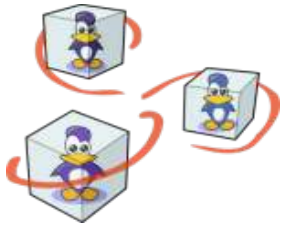
```
paraski33% krg_capset -e +DISTANT_FORK
paraski33% krg_capset -d +DISTANT_FORK
paraski33% ./fork-test
```





Conclusion

- Kerrighed provides an SMP-like system on top of a distributed architecture
- Main features:
  - High-performance architecture
  - Easily tunable
  - Full binary compatibility
- Software available under GPL licence
  - http://www.kerrighed.org



On going works...

- Kerrighed needs some works on:
  - Linux 2.6 port
  - SMP support
  - Checkpoint/restart of parallel applications
  - Add/Remove nodes operations