Object based Storage Cluster File Systems & Parallel I/O

Peter J. Braam

Stelias Computing

and Carnegie Mellon University

braam@cs.cmu.edu

Your speaker...

- - 1991: Full time mathematician
- 1991 – clustering, storage, file systems
  - Regular faculty at Oxford, UK
  - Lead Coda project at CMU 96 – 99
  - Full time @ stelias: 99 –
- Current projects:
  - InterMezzo: similar to Coda
  - Object based storage: this talk
  - A distributed lock manager for Linux

Stelias Computing

- Small
- Open source only
- Pioneers new solutions
- File Systems, Clusters & Storage

Networked File Systems

- **Distributed file systems (InterMezzo)**
  - Single system image, location transparency
  - Disconnected operation, replication
- **Cluster file systems (Lustre)**
  - Sharing database files among systems
  - Recovery from failed nodes
- **Parallel file systems (POBIO)**
  - Support distributed computing
  - Large files, resource management

Talk overview

- Object storage
  - Components
  - Lustre: object based cluster file system
  - Parallel I/O and Object storage
- Linux clustering
- InterMezzo
- Discussion

Object Storage

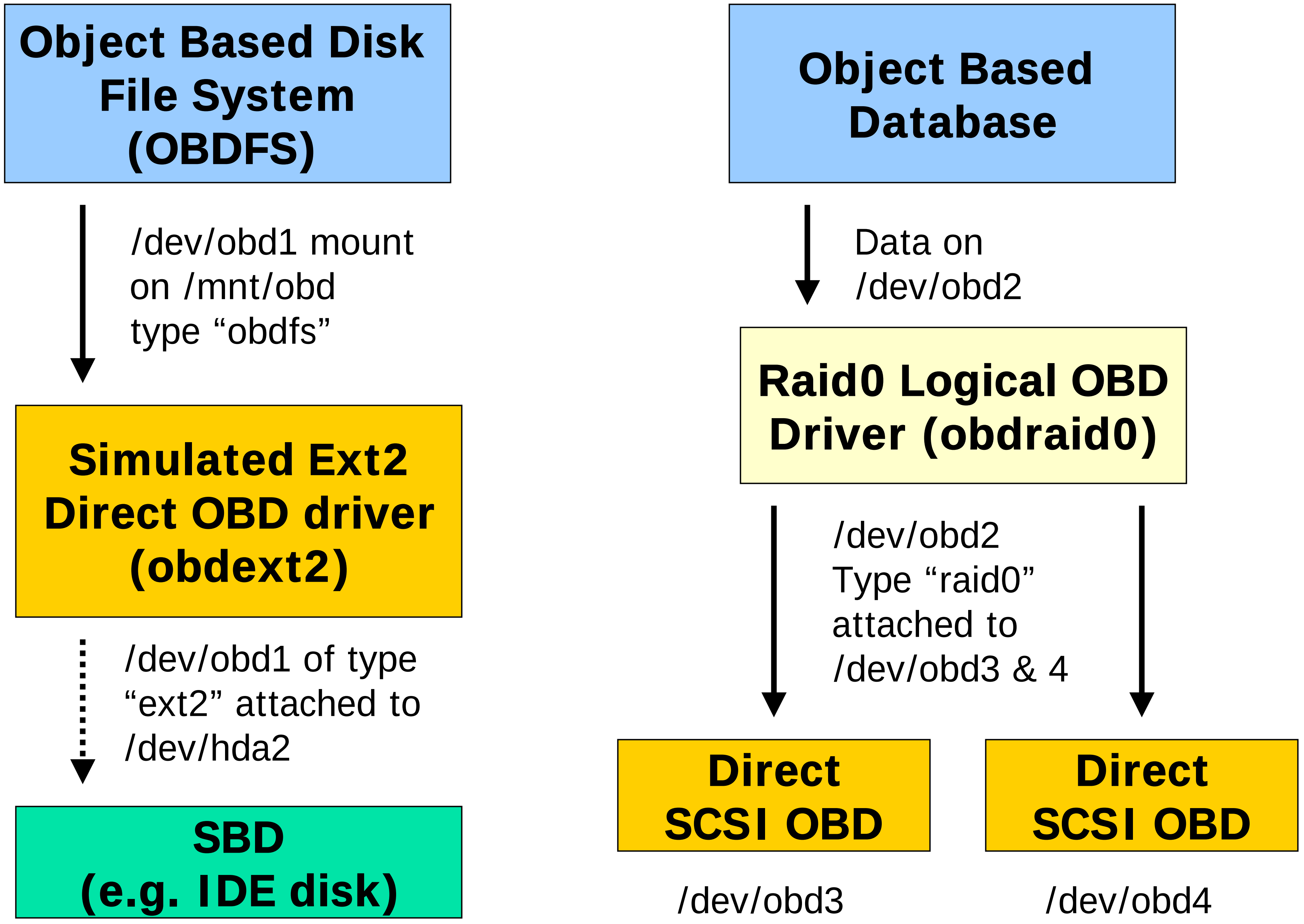
http://www.lustre.org

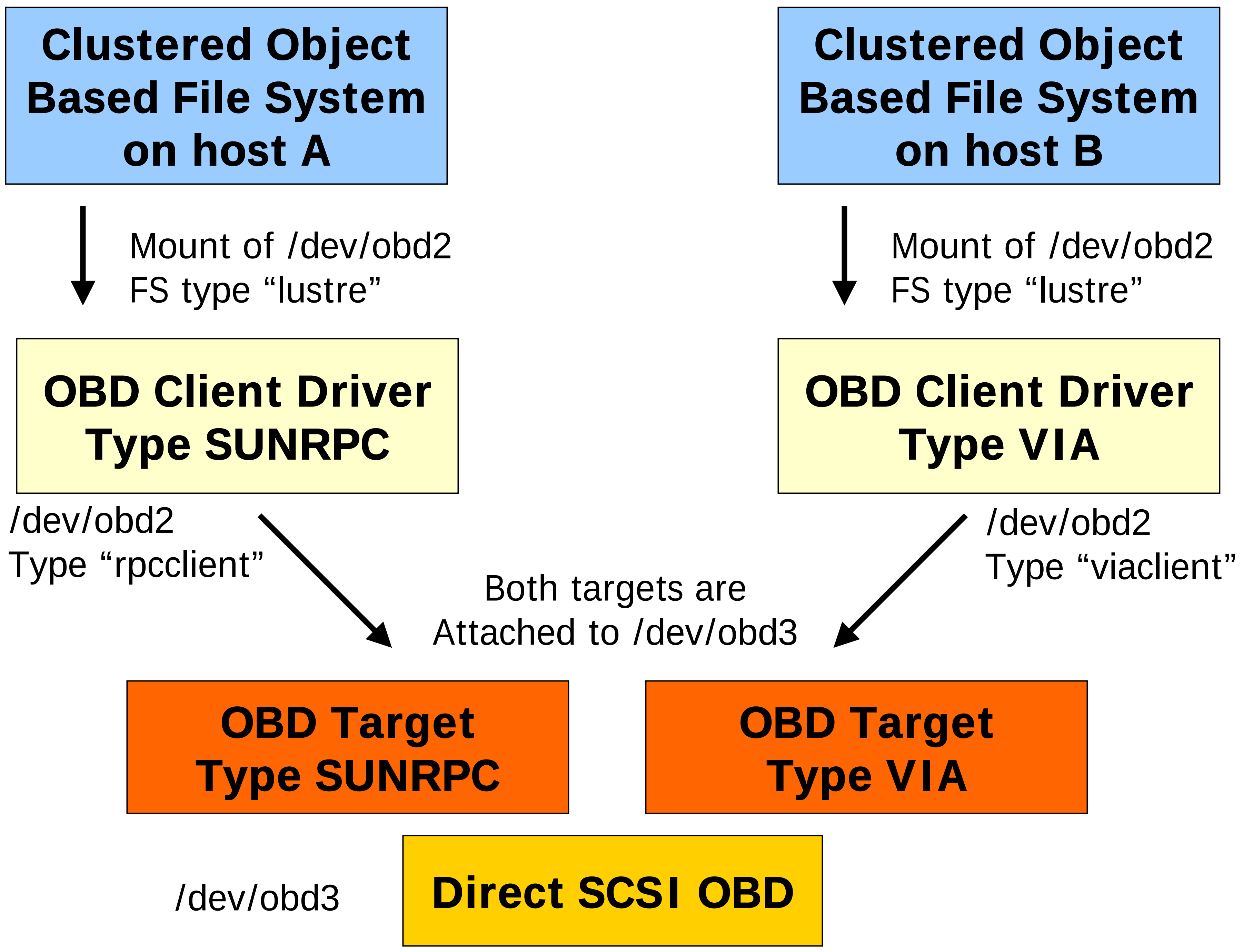
What are OBSDs ?

- Object Based Storage Device
  - More intelligent than block device
- Speak storage at “inode level”
  - create, unlink, read, write, getattr, setattr
- OBSD implementations:
  - **Device driver: lower half of an fs**
  - PDL/NASD style OBD’s – fixed protocol
  - “Real obds” – ask disk vendors

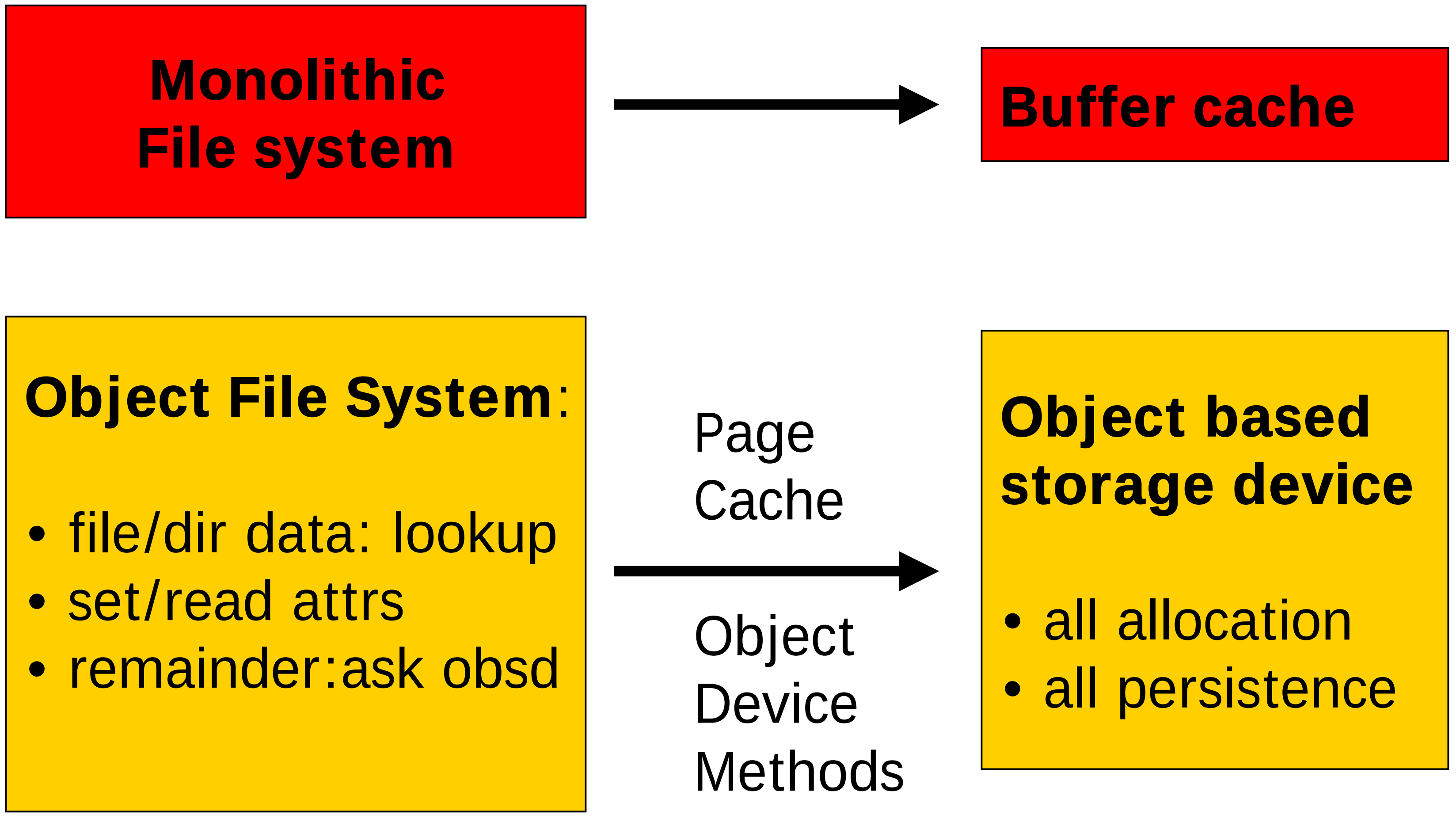
Components of OB Storage

- Storage Object Device Drivers
  - class drivers – attach driver to interface
    - **Targets, clients** – remote access
    - **Direct drivers** – to manage physical storage
    - **Logical drivers** – for storage management
- object storage applications:
  - (cluster) file systems
  - Advanced storage: parallel I/O, snapshots
  - Specialized apps: caches, db’s, filesrv


Object Based Disk File System (OBDFS)
/dev/obd1 mount on /mnt/obd type “obdfs”
Simulated Ext2 Direct OBD driver (obdext2)
/dev/obd1 of type “ext2” attached to /dev/hda2
SBD (e.g. IDE disk)
Object Based Database
Data on /dev/obd2
Raid0 Logical OBD Driver (obdraid0)
/dev/obd2 Type “raid0” attached to /dev/obd3 & 4
Direct SCSI OBD
Direct SCSI OBD
/dev/obd3
/dev/obd4


Clustered Object Based File System on host A
Clustered Object Based File System on host B
Mount of /dev/obd2 FS type “lustre”
Mount of /dev/obd2 FS type “lustre”
OBD Client Driver Type SUNRPC
OBD Client Driver Type VIA
/dev/obd2 Type “rpcclient”
/dev/obd2 Type “viaclient”
Both targets are Attached to /dev/obd3
OBD Target Type SUNRPC
OBD Target Type VIA
/dev/obd3
Direct SCSI OBD

OBDFS


Monolithic File system
Buffer cache
Object File System:
• file/dir data: lookup
• set/read attrs
• remainder:ask obsd
Page Cache
Object Device Methods
Object based storage device
• all allocation
• all persistence

Why obd's...

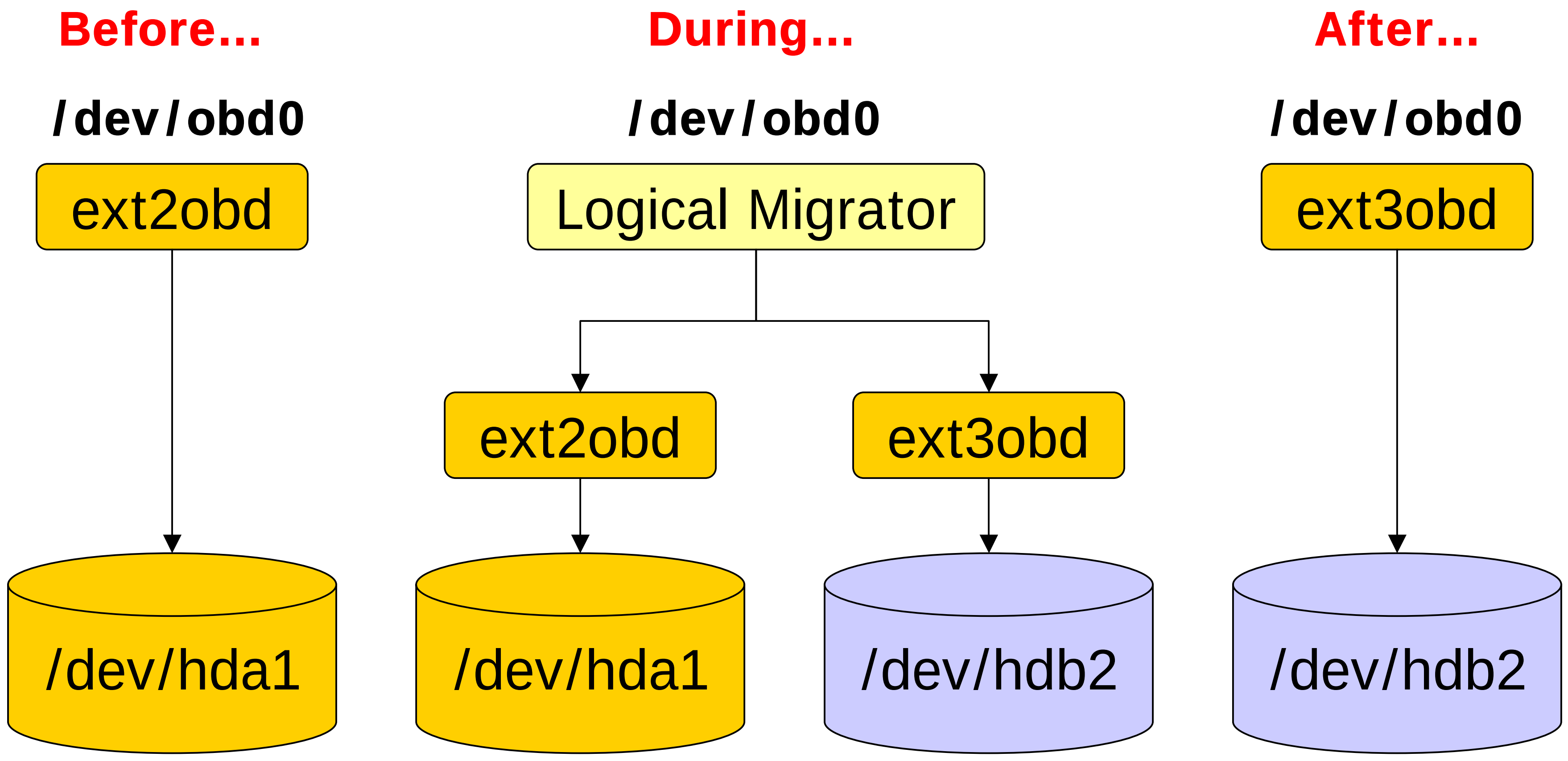
- Storage management: easier
  - File system snapshots
  - Hot file migration
  - Hot resizing
  - Raid
  - Backup
- File systems:
  - Clustering much simpler
  - Component vs monolithic
- Example: parallel I/O

Flexibility with stacking

- Object protocols can be "chained", "stacked"
  - Similar to NT/VMS device driver model
- Plug and Play storage management
- Examples...

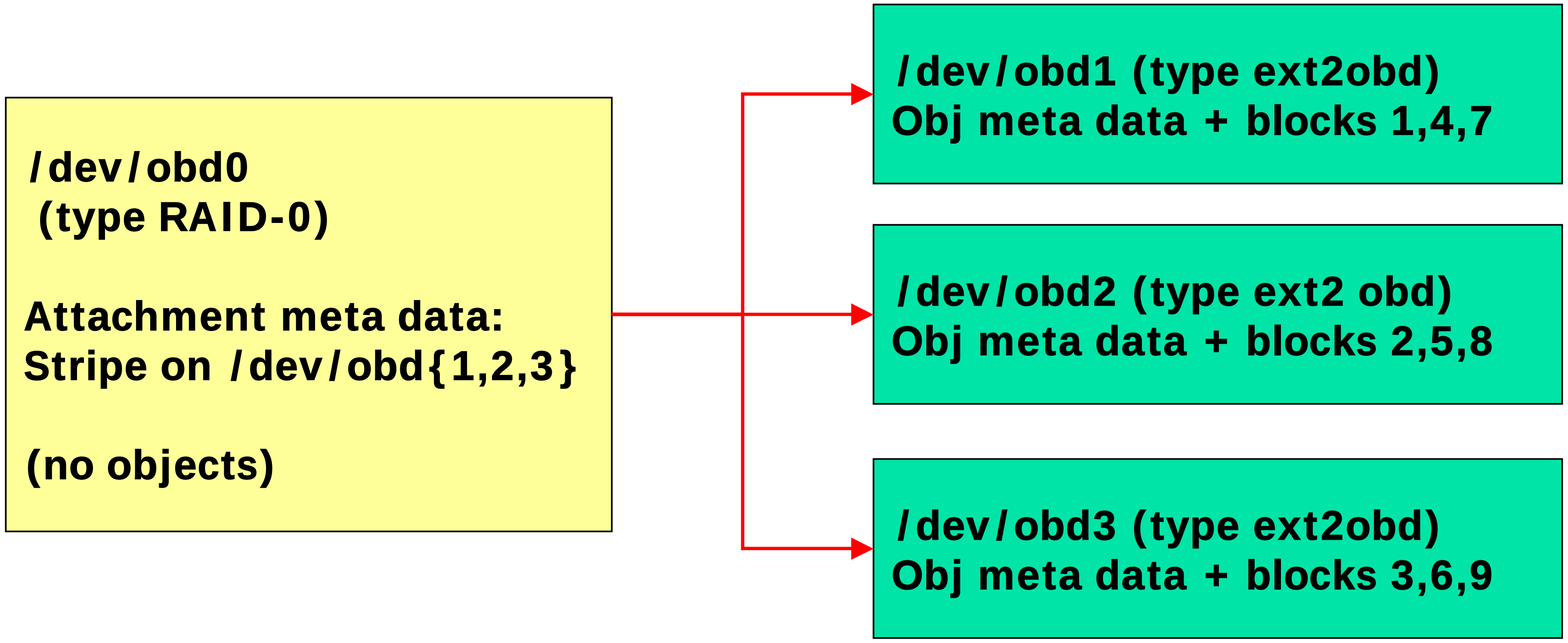
Hot data migration:

Key principle: dynamically switch object device types



LOVM: can do it all - Raid

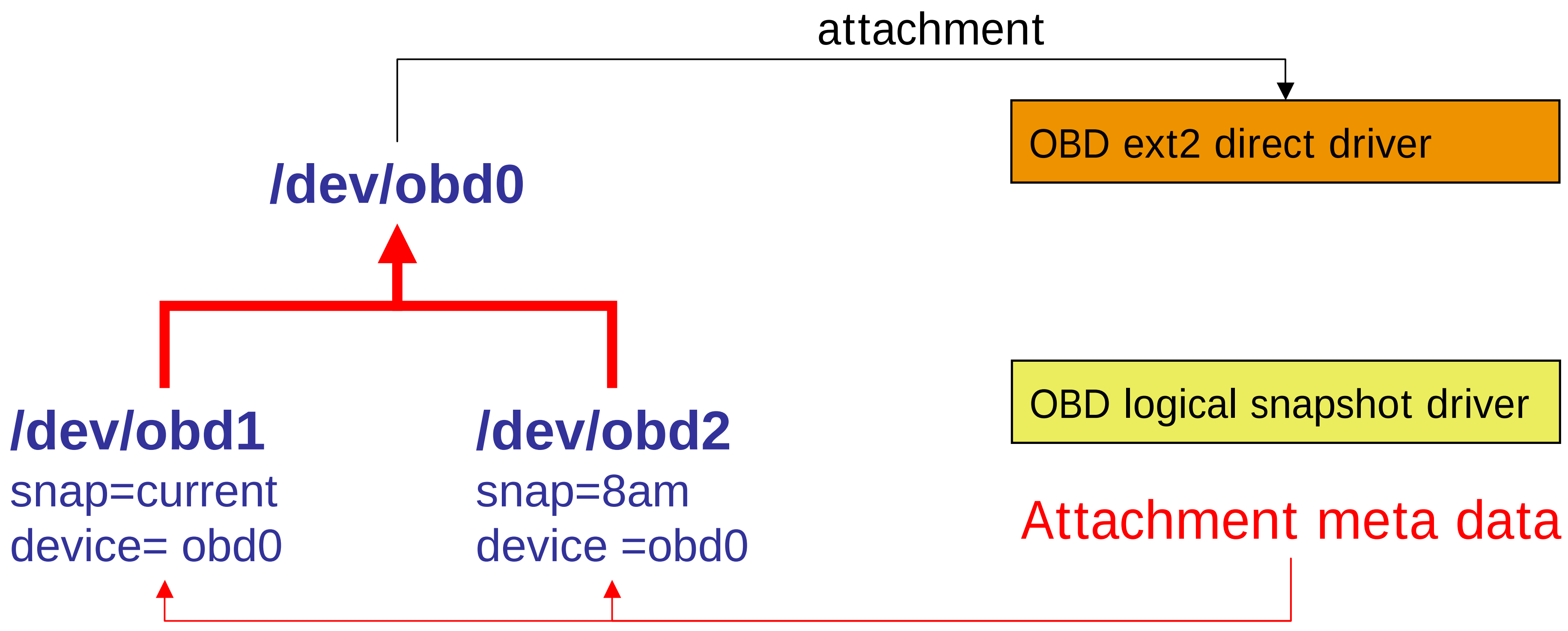
Logical Object Volume Management:


/dev/obd0
(type RAID-0)
Attachment meta data:
Stripe on /dev/obd{1,2,3}
(no objects)
/dev/obd1 (type ext2obd)
Obj meta data + blocks 1,4,7
/dev/obd2 (type ext2 obd)
Obj meta data + blocks 2,5,8
/dev/obd3 (type ext2obd)
Obj meta data + blocks 3,6,9

Objects may be files, or not...

- Common case:
  - Object, like inode, represents a file
- Object can also:
  - represent a stripe (RAID)
  - bind an (MPI) File_View
  - redirect to other objects

Snapshot setup



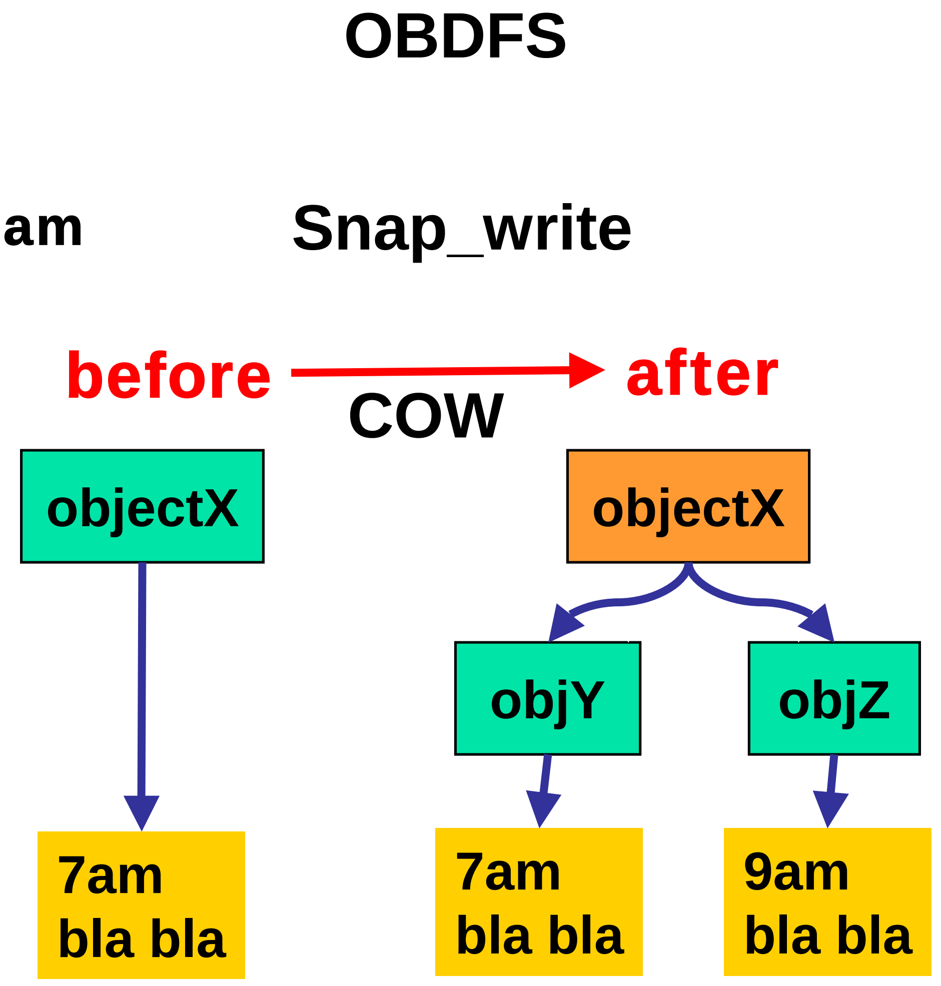
- **Result:**
  - /dev/obd2 is read only clone
  - /dev/obd1 is copy on write (COW) for 8am

Snapshots in action

- **mount /dev/obd1 /mnt/obd**
- **mount /dev/obd2 /mnt/obd/8am**

- **Modify /mnt/obd/files**

- **Result:**
  - **new copy** in /mnt/obd/files
  - **old copy** in /mnt/obd/8am



POBIO

Parallel Object Based I/O

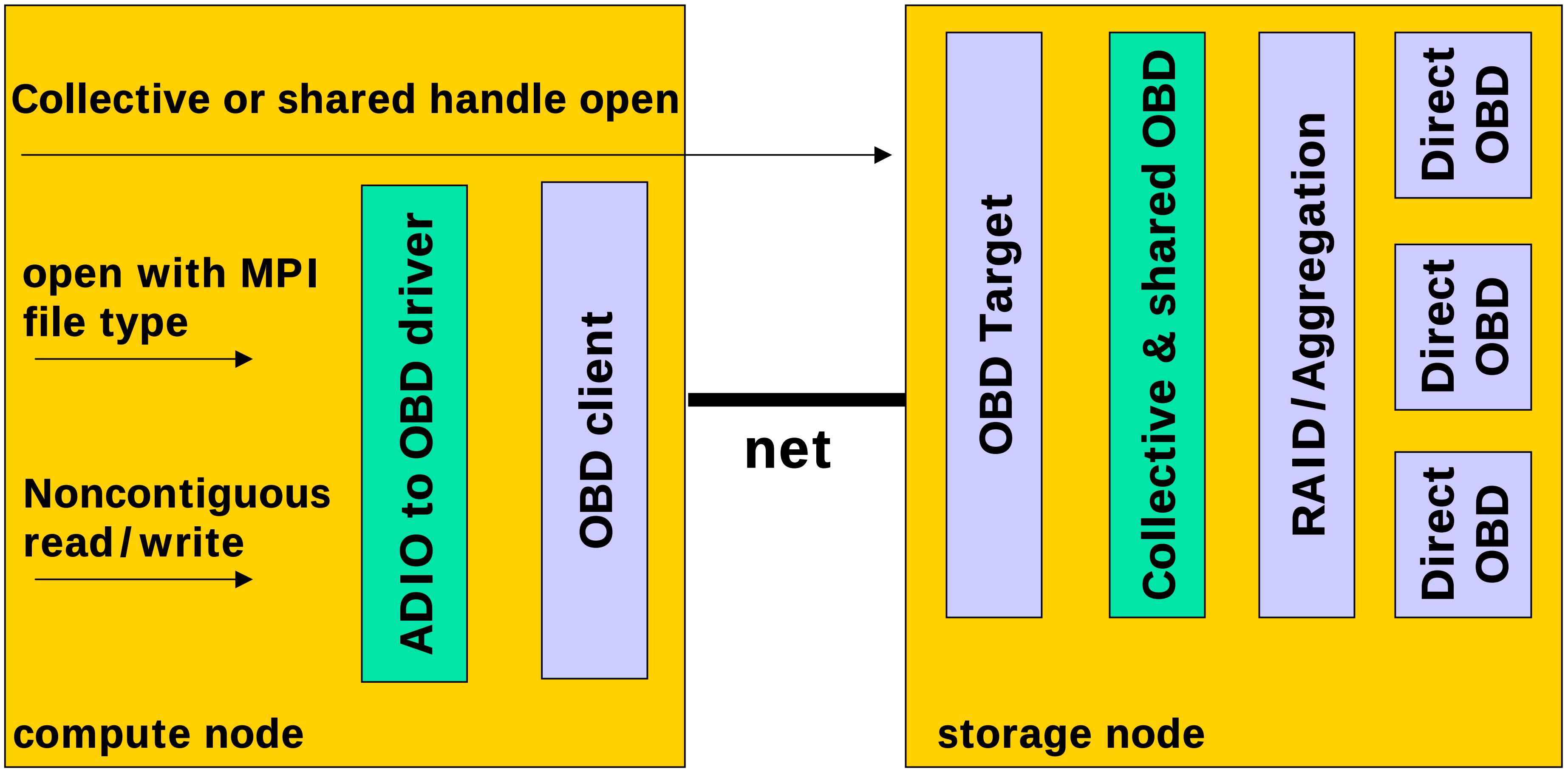
- Object Read/Write primitives
  - Send multiple buffers
  - To multiple disk destinations
  - “true scatter/gather”, not just VM
- Needed **ADIO logical object driver**
  - Abstract device I/O
  - Lower level interface to implement MPI-IO
- filetypes:
  - MPI_Data & File type support in logical OBD layer

Collective, shared, async I/O

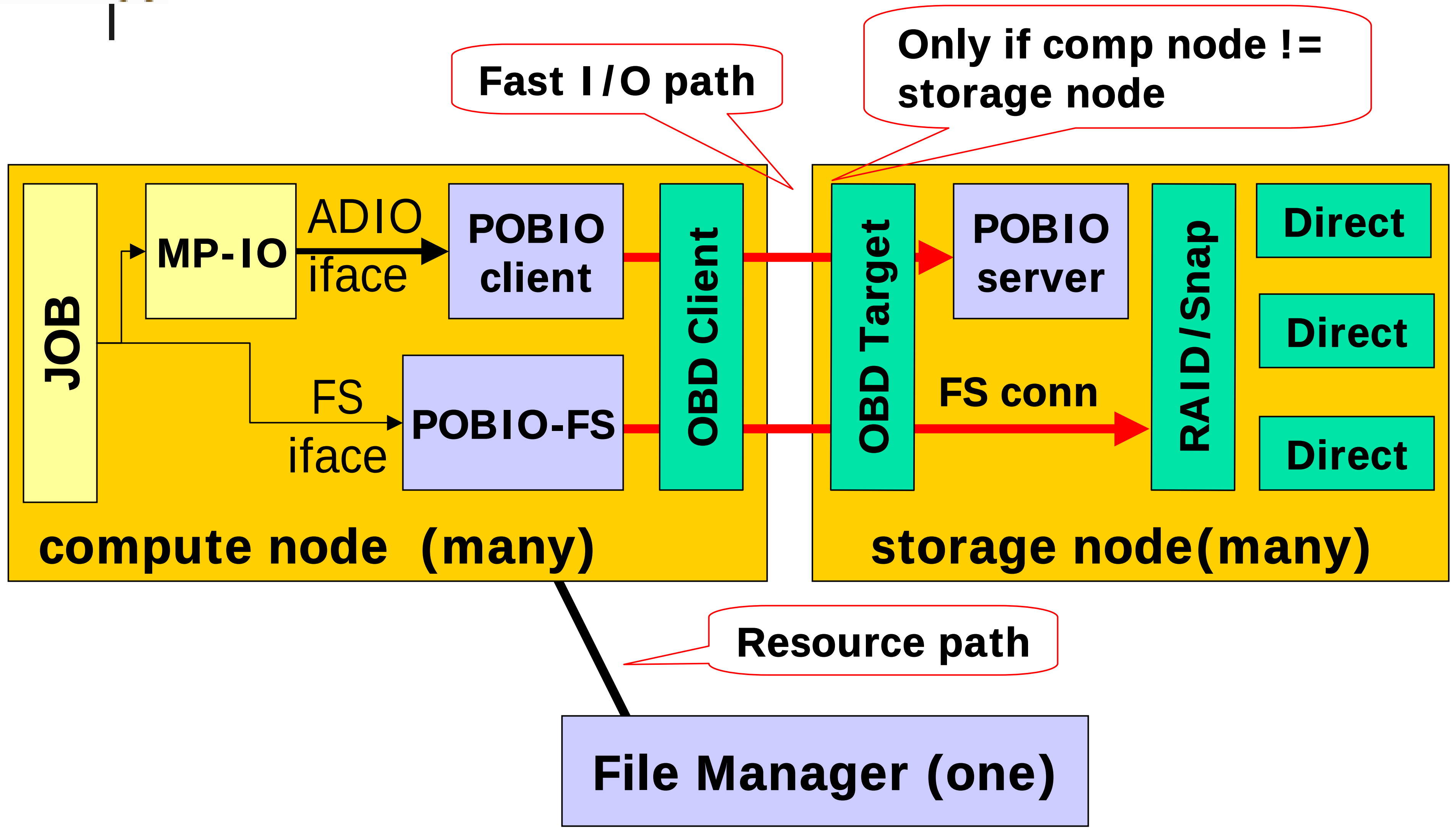
- need an object open:
  - that takes MPI_Comm
- waiting primitives for I/O completion
  - Easy to do with DLM
- shared file pointers

Noncontiguous I/O

- **OBD protocol has scatter/gather non contiguous RW**



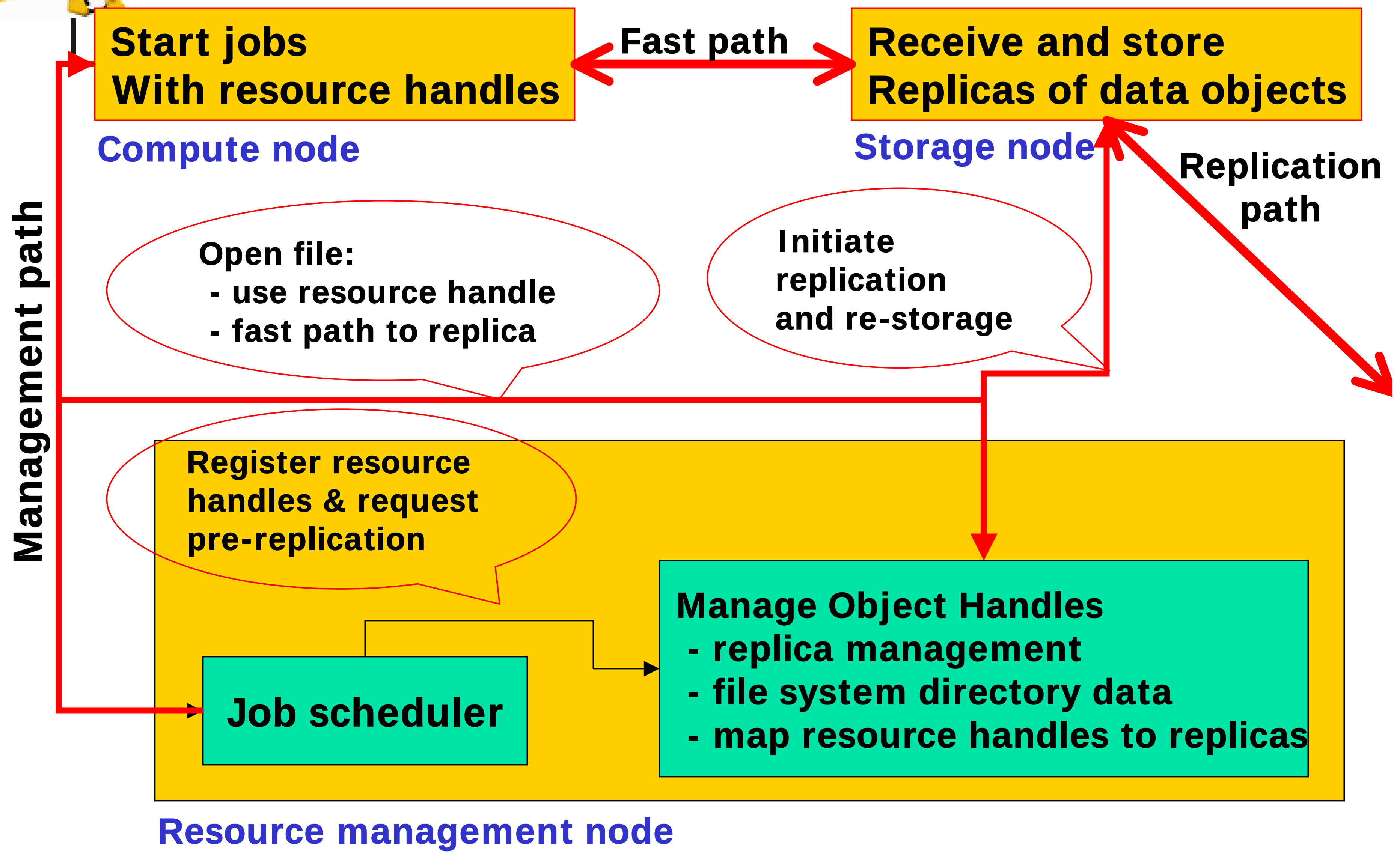
POBIO with File System



Resource management

- Make explicit provisions for
  - Scheduler resource records
  - (Pre-)Replication of (segments of) data
- Use file manager to get handles
  - Manages directory information
  - Returns “fast path” file handles to replica

POBIO – resource mgmt



POBIO – further comments

- Many components already exist
  - We have object based file system
  - Aggregation & snapshot drivers
  - Infrastructure for stacking objects
- Not monolithic:
  - Can build separate components
- Would love to build a prototype

Linux clusters

Clusters - purpose

- Assume:
  - Have a limited number of systems
  - On a secure System Area Network
- Require:
  - A scalable almost single system image
  - Fail-over capability
  - Load-balanced redundant services
  - Smooth administration

Ultimate Goal

- provide generic components
- OPEN SOURCE
- Inspiration: VMS VAX Clusters
- New:
  - Scalable (100,000’s nodes)
  - Modular
- Need distributed, cluster & parallel FS’s
  - InterMezzo, GFS/Lustre, POBIO-FS

The Linux "Cluster Cabal":

- **Peter J. Braam –** CMU, Stelias Computing, Red Hat
- **Stephen Tweedie –** Red Hat

Who is doing what?

- Tweedie
  - Project leader
  - Core cluster services
- Braam
  - DLM
  - InterMezzo FS
  - Lustre Cluster FS
- **Many others**

- McVoy
  - Cluster computing
  - SMP clusters
- Red Hat
  - Cluster apps & admin
- UMN
  - GFS: Shared block FS

Technology Overview

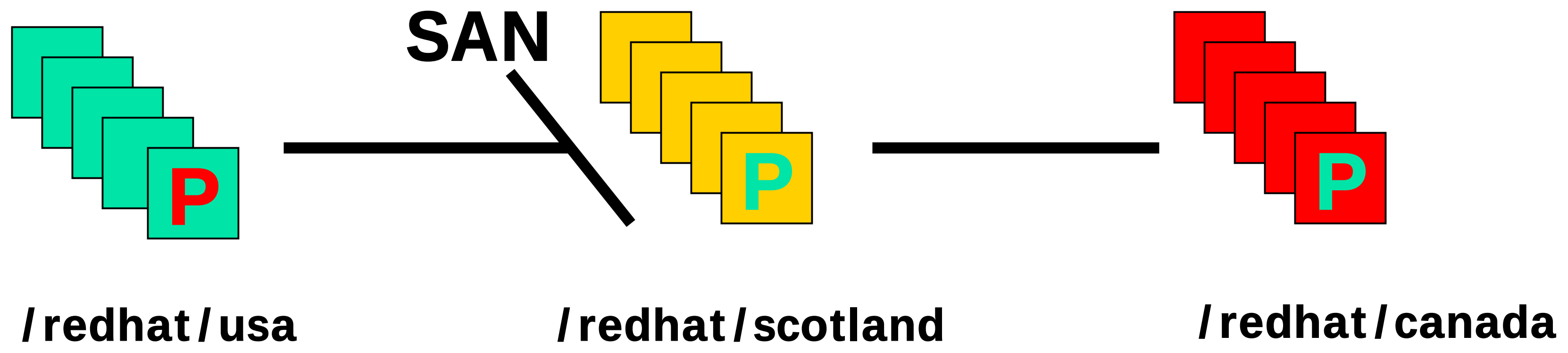
Modularized VAX cluster architecture (Tweedie)

Core	Support	Clients
Transition	Cluster db	Distr. Computing
Integrity	Quorum	Cluster Admin/Apps
Link Layer	Barrier Svc	Cluster FS & LVM
Channel Layer	Event system	DLM

Events

- Cluster transition:
  - Whenever connectivity changes
  - Start by electing “cluster controller”
- Only merge fully connected sub-clusters
- Cluster id: counts “incarnations”
- Barriers:
  - Distributed synchronization points

Scalability – e.g. Red Hat cluster



- **P = peer**
  - Proxy for remote core cluster
  - Involved in recovery
- **Communication**
  - Point to point within core clusters
  - Routable within cluster
  - Hierarchical flood fill
- **File Service**
  - Cluster FS within cluster
  - Clustered Samba/Coda etc
- **Other stuff**
  - Membership / recovery
  - DLM / barrier service
  - Cluster admin tools

Lustre File System

- Lustre ~ Linux Cluster
- Object Based Cluster File System
  - Based on OBSD’s
- Symmetric - no file manager
- Cluster wide Unix semantics: DLM
- Journal recovery
- Suitable for e.g. clustered database files

Benefits of Lustre design

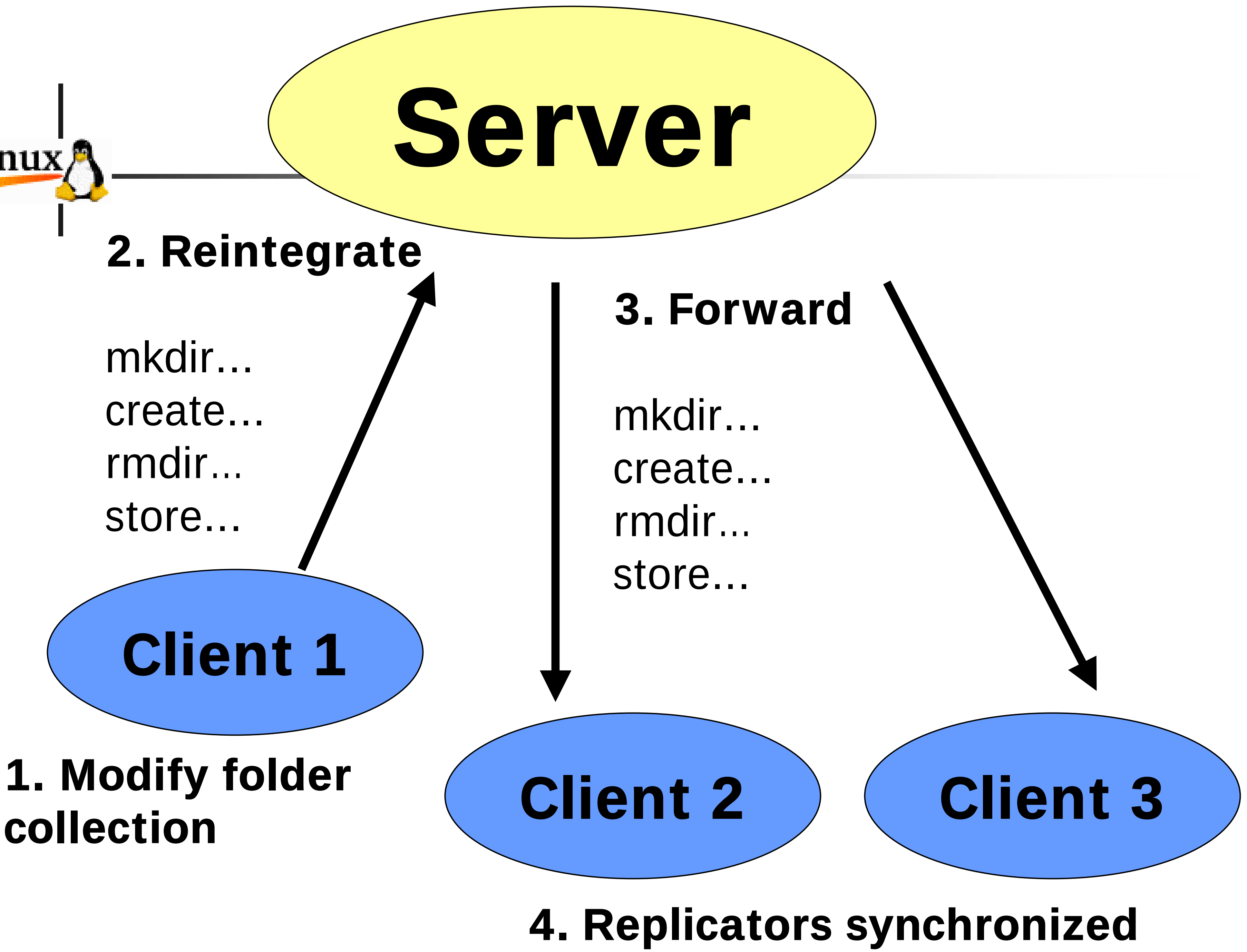
- Space & object allocation
  - Managed where it is needed
  - Eliminate sharing bitmaps etc
- Consequences
  - Somewhat similar to Calypso (IBM)
  - IBM (Devarakonda etc): less traffic
  - **Much** simpler locking

InterMezzo

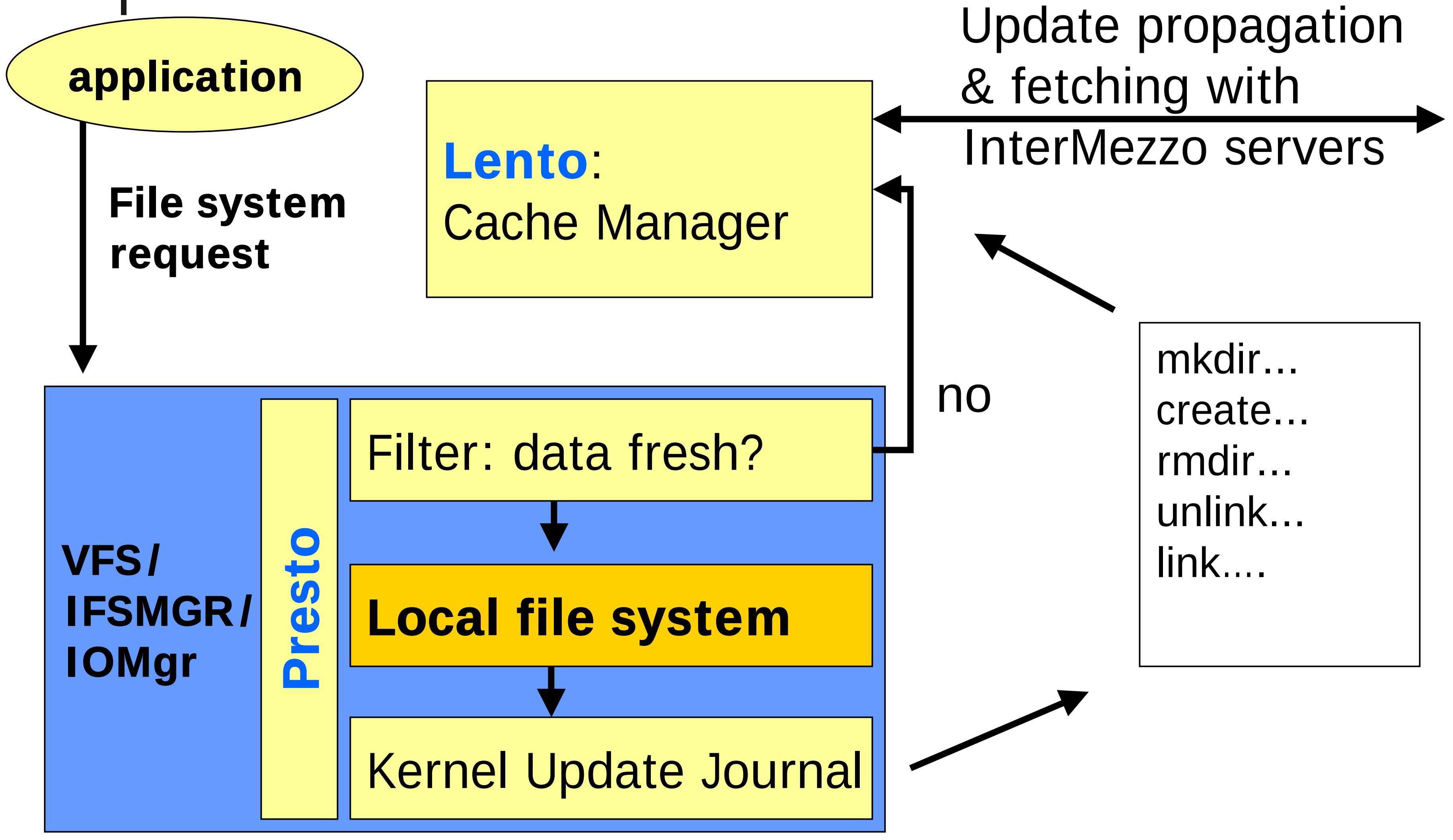
http://www.inter-mezzo.org

Target

- Replicate or cache directories
  - Automatic synchronization
  - Disconnected operation
  - Proxy servers
  - Scalable
- Purpose
  - Entire System Binaries
  - Home directories: laptop/desktop
- Very simple
  - Coda style protocols
  - Wrap around local file systems as cache


Server
2. Reintegrate
mkdir...
create...
rmdir...
store...
3. Forward
mkdir...
create...
rmdir...
store...
Client 1
1. Modify folder collection
Client 2
Client 3
4. Replicators synchronized

Basic InterMezzo



Conclusion

- Lots of interesting projects
- Object Based Storage
  - Promising: needs exploration
  - Modular structure
  - Requires only commodity hardware
- InterMezzo
  - Finding wide acceptance
  - Lots of work needed

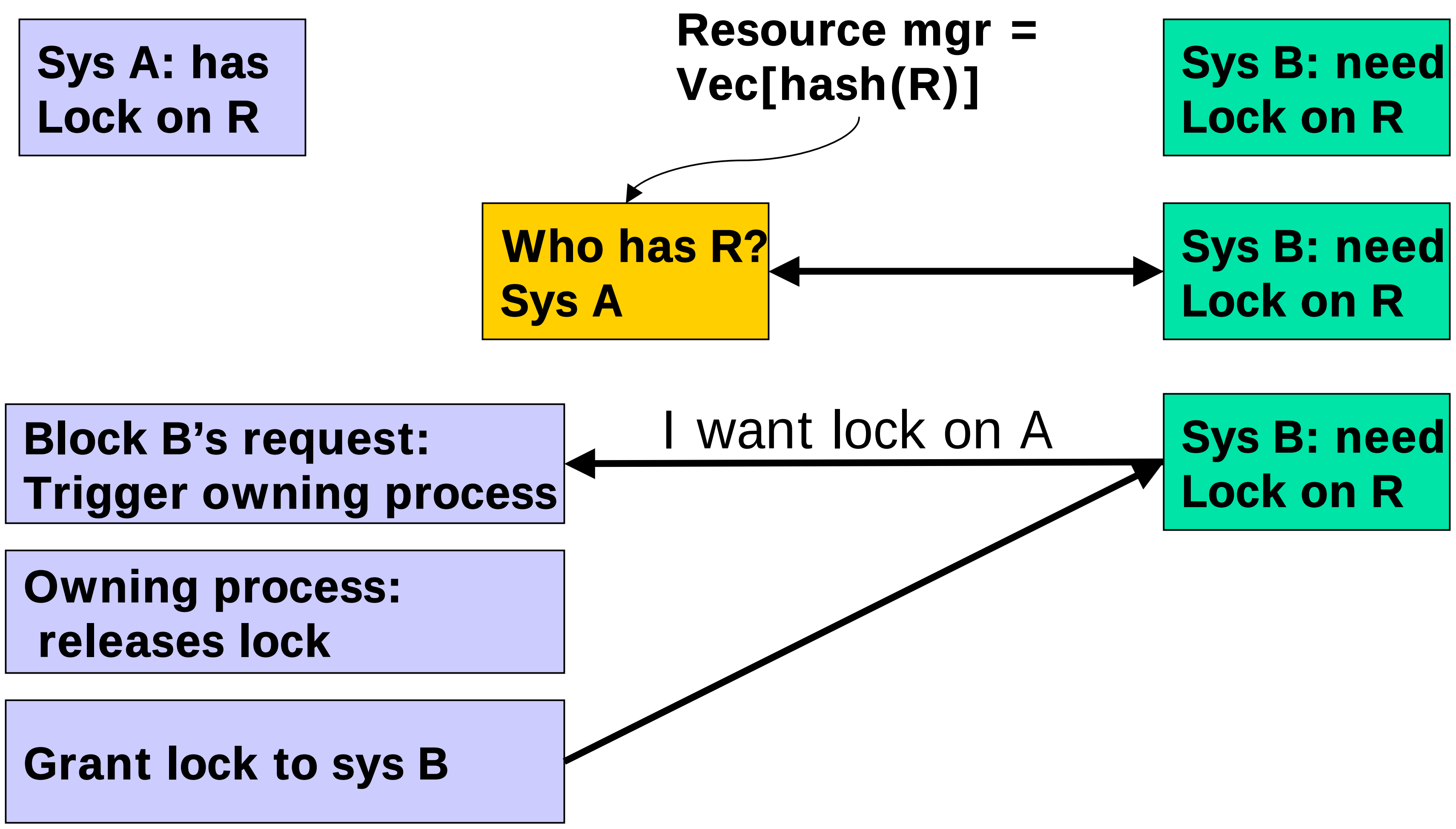
Distributed Lock Manager

Locks & resources

- Purpose: generic, rich lock service
- Will subsume “callbacks”, “leases” etc.

- Lock resources: resource database
  - Organize resources in trees
  - Most lock traffic is local
- High performance
  - node that acquires resource manages tree

Typical simple lock sequence



A few details...

- Six lock modes
  - Acquisition of locks
  - Promotion of locks
  - Compatibility of locks
- First lock acquisition
  - Holder will manage resource tree
- Remotely managed
  - Keep copy at owner
- Callbacks:
  - On blocking requests
  - On release, acquisition
- Recovery (simplified):
  - Dead node was:
    - Mastering resources
    - Owning locks
  - Re-master rsrc
  - Drop zombie locks