Flexible, Modular File Volume Virtualization in Loris

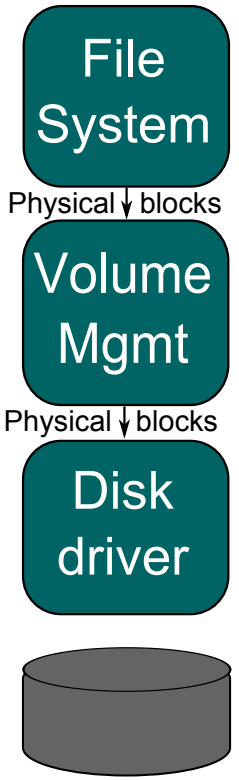
Raja Appuswamy, David C. van Moolenbroek,
Andrew S. Tanenbaum

Vrije Universiteit, Amsterdam

May 27, 2011

Traditional Storage Stack

- Originally one one file volume per block-based disk
- Administration tradeoffs
  - Efficiency vs Flexibility tradeoff
  - Root cause was the file volume per disk bond
- Volume managers virtualized file volumes
  - Backward compatible logical disk abstraction
  - One file volume per ~~physical~~ logical disk bond
- Compatibility-driven integration has fatal flaws



Traditional storage stack

Flexibility(1): Complicated Device Management

- Complicated storage model
  - Simple device operations require several error-prone steps
- "File volume per logical disk" bond is the root cause
  - Need to change data structures in both layers
  - One operation per layer (example: expand LV, expand FS)
- An ideal storage system should
  - Allow administrator to just state the intent
  - Automate implementation details

Flexibility(2): Coarse-grained File Management

- Coarse-grained, volume-level policy specification
  - Semantically unaware - no knowledge of block relationship
  - Snapshotting & encryption of individual files not possible
- But customers need more flexibility
  - Storage retention/ILM policies applied to business objects
  - Storage tiering performed on per-file basis
  - End-users associate policies with files and file types
- An ideal storage system should
  - Enable policy specification at a range of granularities
  - Have a modular policy-mechanism split

Flexibility(2): Coarse-grained File Management

- Coarse-grained, volume-level policy specification
  - Semantically unaware - no knowledge of block relationship
  - Snapshotting & encryption of individual files not possible
- But customers need more flexibility
  - Storage retention/ILM policies applied to business objects
  - Storage tiering performed on per-file basis
  - End-users associate policies with files and file types
- An ideal storage system should
  - Enable policy specification at a range of granularities
  - Have a modular policy-mechanism split

The traditional stack lacks flexibility

Heterogeneity(1): Complicated Integration of New Devices

- New devices with new interfaces need to be integrated
  - Byte-accessible or page-accessible flash devices
  - Object-based storage devices
- Building device-specific file systems
  - Not compatible with block-based volume managers
- Building a translation layer to hide device-specific interfaces
  - Widens the "Information gap"
  - Duplication of functionality

Heterogeneity(2): Inability to Exploit Device Characteristics

- SSDs with widely varying performance characteristics

SSD	Sequential Read	Sequential Write	Random Read	Random Write
Intel X25-V	170MB/s	**35 MB/s**	25,000 4KB IOPS	2500 4KB IOPS
Intel X25-M	250MB/s	**100 MB/s**	35,000 4KB IOPS	8600 4KB IOPS

- Device-specific layout is required
  - Write-optimized layout on X25-M
  - Read-optimized layout on X25-V
- Impossible to exploit heterogeneity with the traditional stack
  - Impossible to bind file systems to devices
  - Multiple file systems can share a device, rendering layout optimizations futile

Heterogeneity(2): Inability to Exploit Device Characteristics

- SSDs with widely varying performance characteristics

SSD	Sequential Read	Sequential Write	Random Read	Random Write
Intel X25-V	170MB/s	**35 MB/s**	25,000 4KB IOPS	2500 4KB IOPS
Intel X25-M	250MB/s	**100 MB/s**	35,000 4KB IOPS	8600 4KB IOPS

- Device-specific layout is required
  - Write-optimized layout on X25-M
  - Read-optimized layout on X25-V
- Impossible to exploit heterogeneity with the traditional stack
  - Impossible to bind file systems to devices
  - Multiple file systems can share a device, rendering layout optimizations futile

The traditional stack fails to support heterogeneity both within and across device families

Context - The Loris Storage Stack

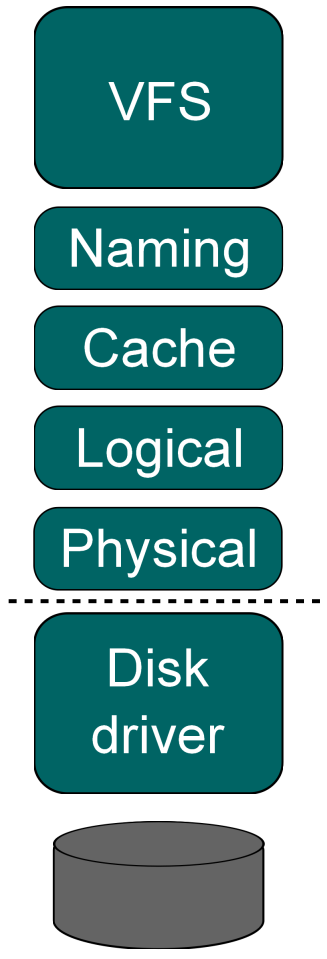
- Traditional stack also suffers from serious reliability issues
  - Silent data corruption, RAID write hole
  - Lack of support for graceful degradation
- In prior work, we presented Loris
  - A modular redesign of the traditional storage stack

The Loris Storage Stack - Layers and Interfaces

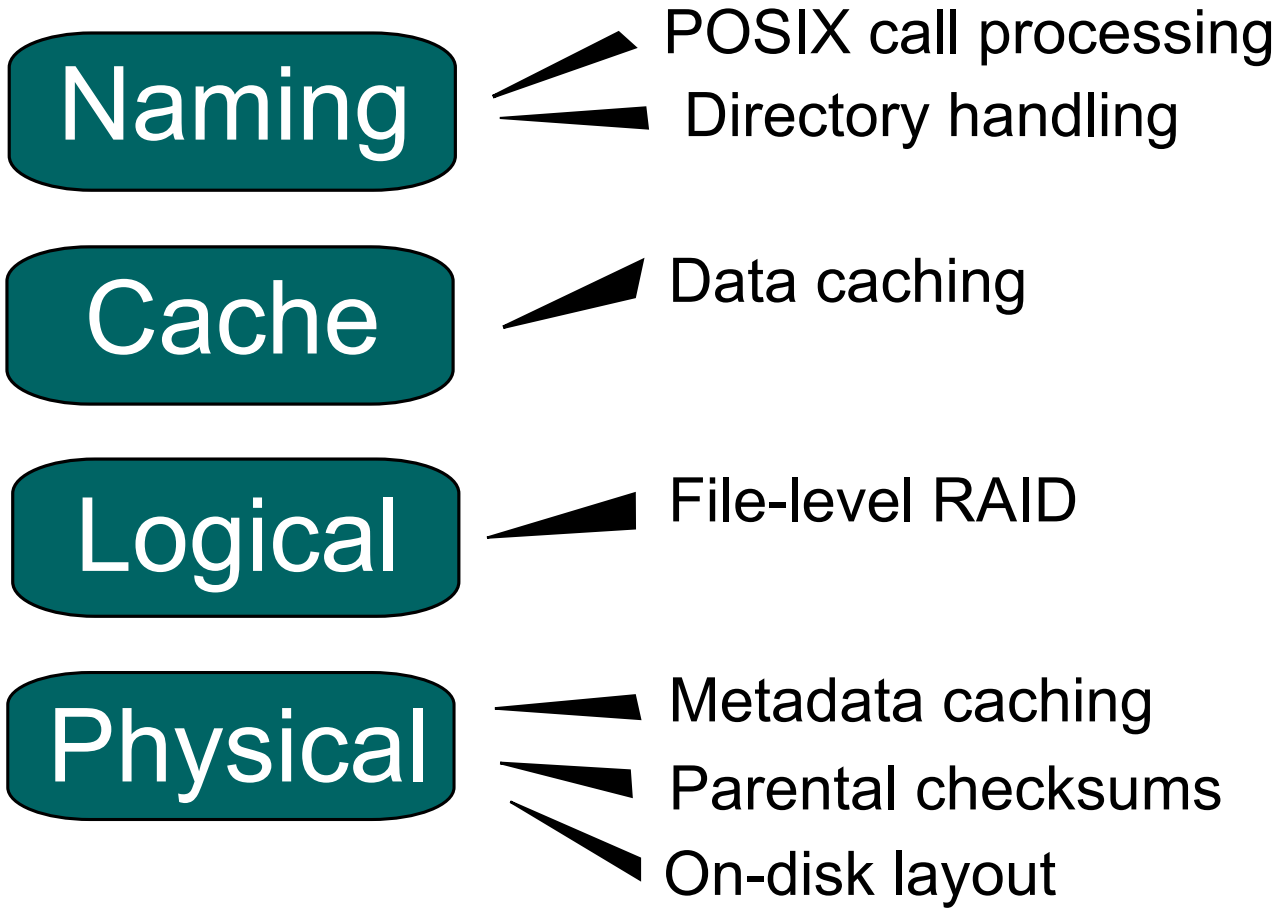
- File-based interface between layers
  - Each file has a unique file identifier
  - Each file has a set of attributes

- File-oriented requests:

create	**truncate**
delete	**getattr**
read	**setattr**
write	**sync**



Loris - Division of Labor

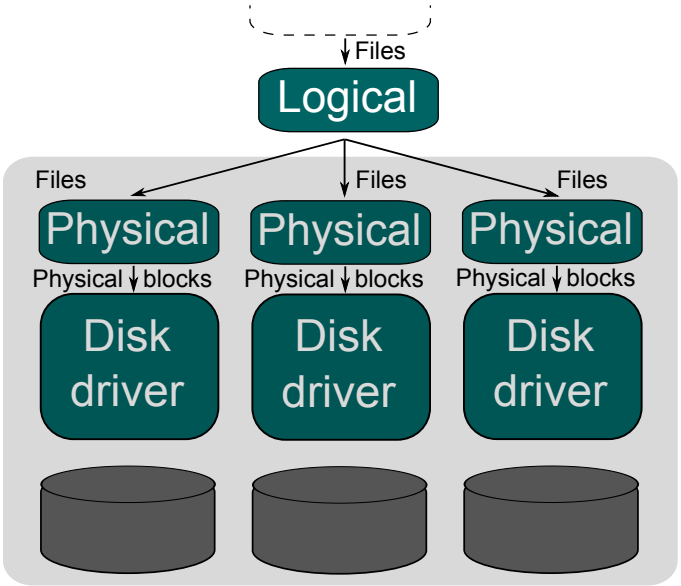


Loris V1

- Loris V1 did not support file volume virtualization
  - One file volume per set of devices bond
  - No file volume snapshotting or thin provisioning support
- Storage model similar to traditional file system days
  - Online device addition/removal not possible
- In this work, we augment the Loris stack to
  - Automated device management using File Pools
  - Provide flexible file volume virtualization

File Pools - Our New Storage Model

- The pool of files serviced by a group of physical modules

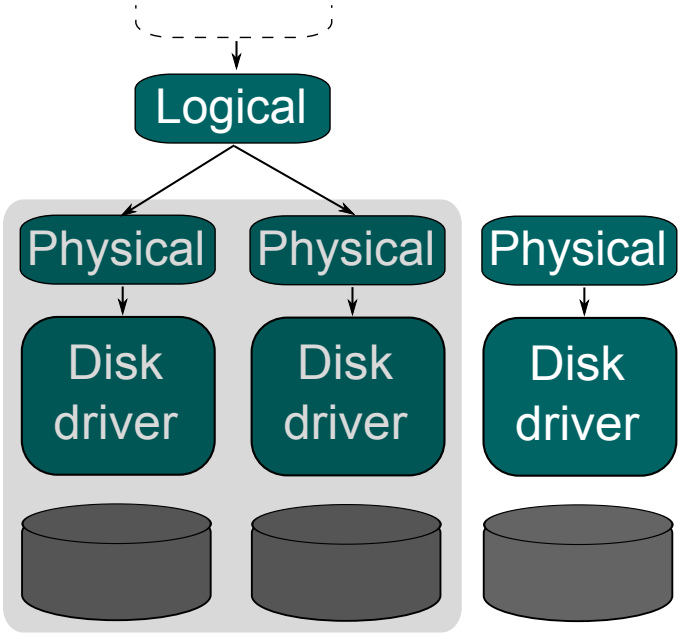


- File pools form the unit of device administration
  - Each device is a part of one file pool
  - Multiple file pools for performance isolation

Simplified Device Administration with File Pools

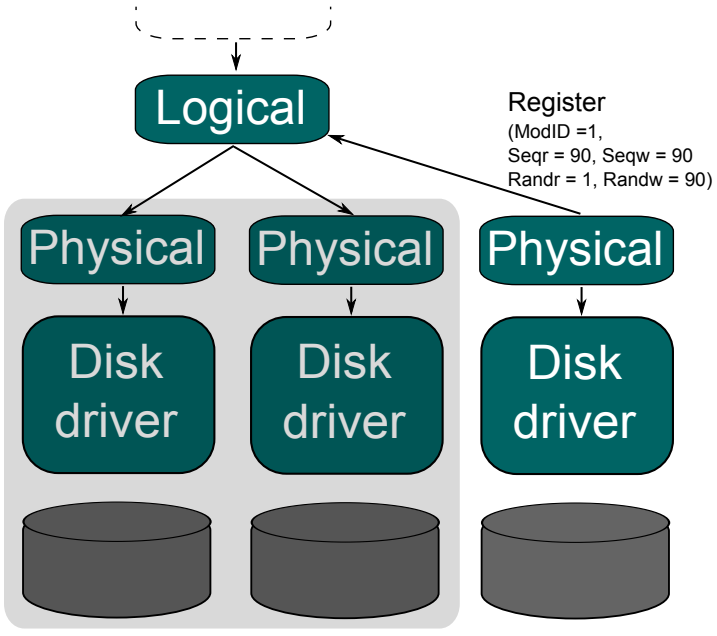
- No resizing required - one physical module per device
- Single-step device addition
- Efficient device removal
  - Performed by moving files (not blocks) between physical modules
  - File-level data movement moves only live data

Device Addition Example



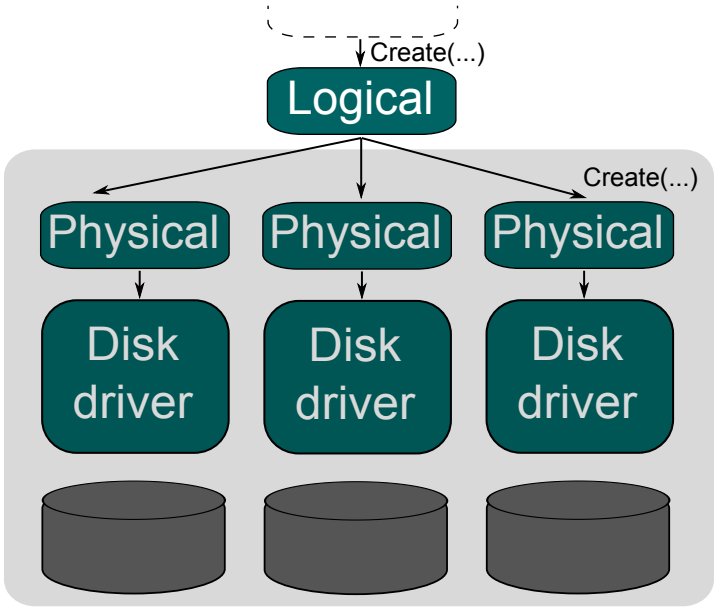
- Fully-automated device addition
  - Device-specific physical module started automatically

Device Addition Example



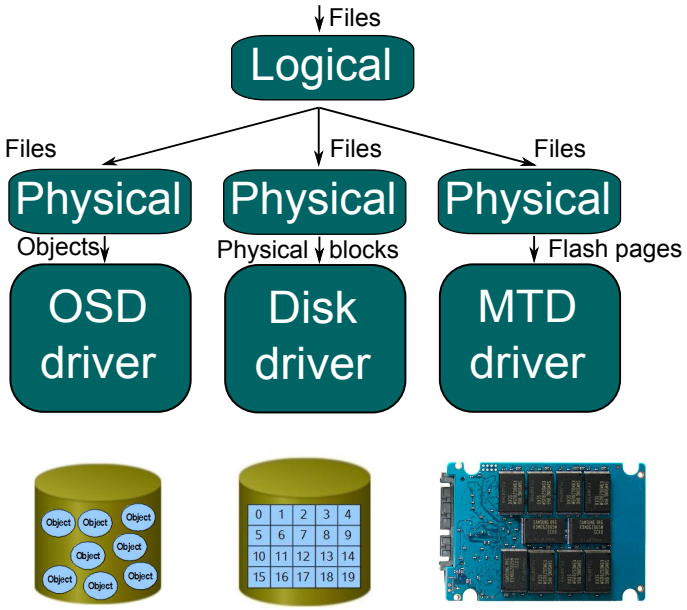
- Fully-automated device addition
  - Device-specific physical module started automatically
  - Registration and handshake with logical module

Device Addition Example



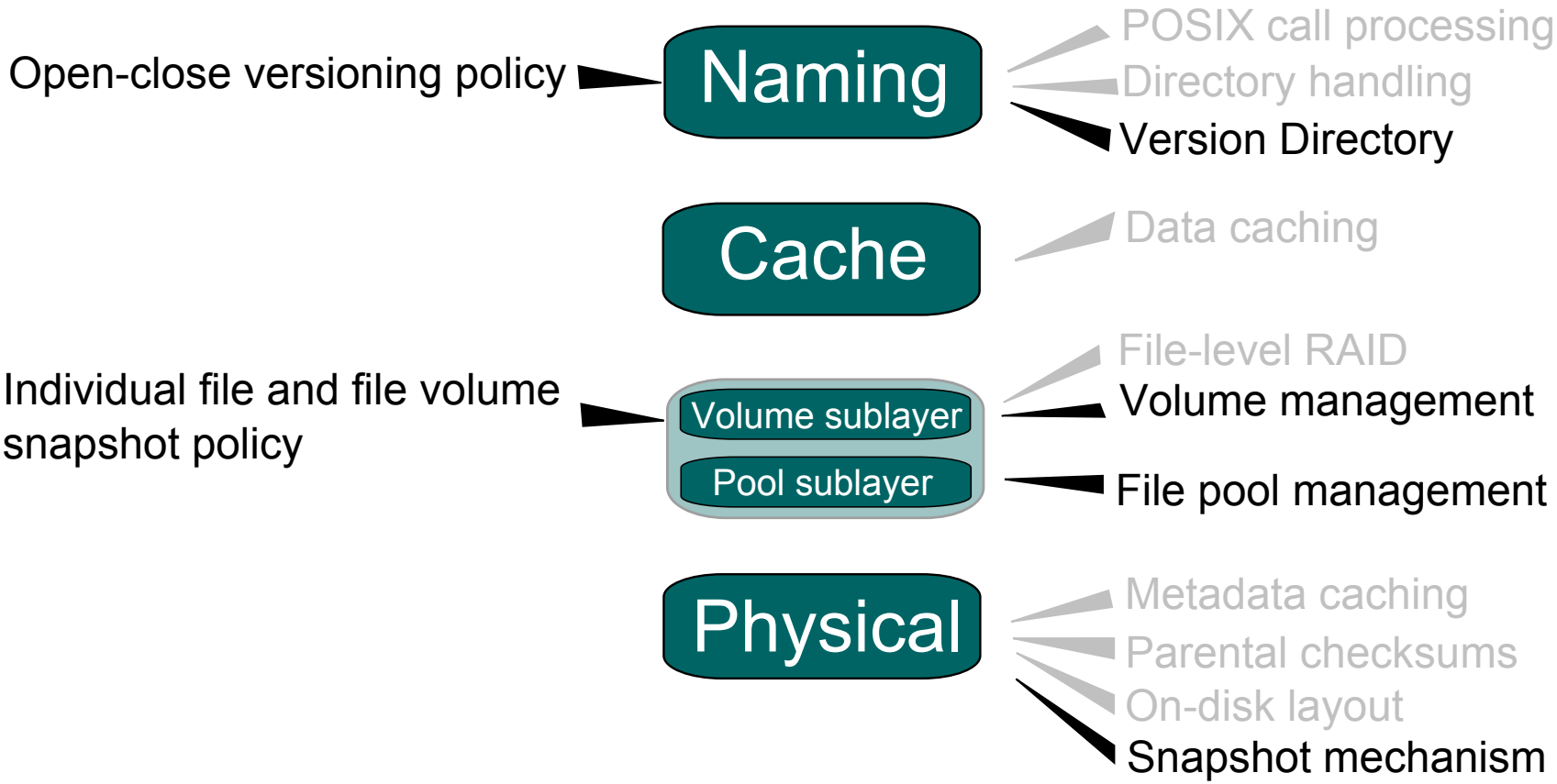
- Fully-automated device addition
  - Device-specific physical module started automatically
  - Registration and handshake with logical module
  - All files in the new physical module are available for use

Supporting Heterogeneity with File Pools

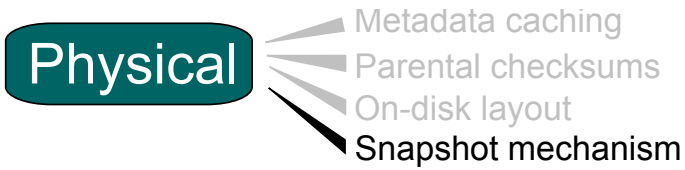


- Single logical layer implementation across all device types
- Integrating new device types requires only a new physical layer
- Device-specific layout schemes to exploit heterogeneity

Flexible File Volume Virtualization - Policy Mechanism Split

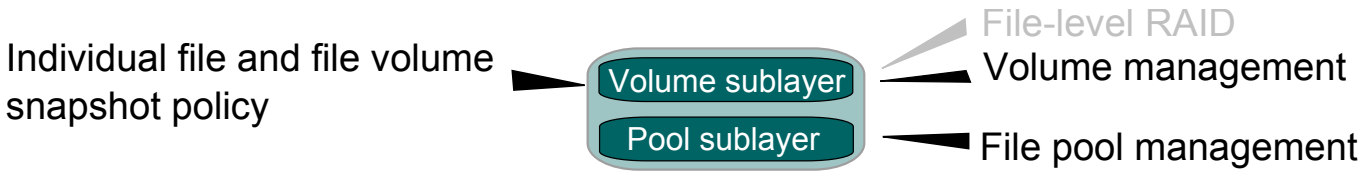


Physical Layer



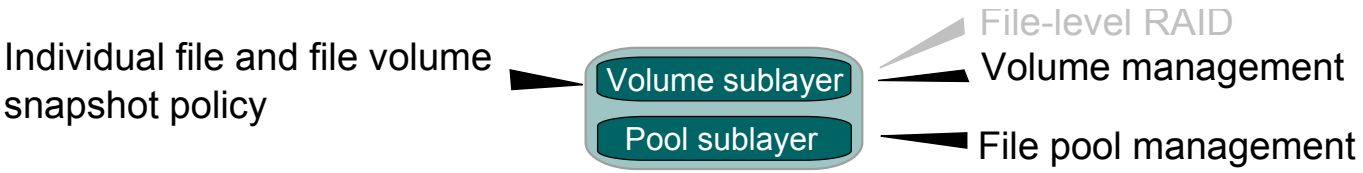
- Must provide some form of physical file snapshotting
  - Schemes like copy or COW-based snapshotting etc.
  - Prototype uses an MFS-style COW-based physical layer
- No space efficiency-performance tradeoffs
  - Block-granular data sharing between versions
  - Integrated with on-disk layout to maximize performance
- A new call to snapshot inodes exposed to logical layer

Logical Layer - Mechanism



- Consists of File Pool and File Volume sublayers
- File pool sublayer implements the storage model
- File volume sublayer provides volume management
  - Supports volume administration (create/delete vol)
  - Maintains <file-file volume> and <logical file-physical file> relationships
- All file volumes share a single pool of files (Thin Provisioning)

Logical Layer - Policy



- Volume sublayer also acts as a policy enforcer
  - Policies for snapshotting files/file volumes
  - Builds on physical layer's inode snapshotting
- Logical layer provides a new snapshot call
  - Snapshot file/file volume using fileID/VolumeID

Naming Layer

- Implements version directories - unified interface to browse snapshot history
  - Version directory is a virtual directory
  - Each snapshot, irrespective of origin, is a file entry
  - Browsing history done by appending any file name with @
  - Entire subtrees can be also scoped to an older snapshot
- Naming layer also provides open-close versioning policy
  - Naming layer invokes a snapshot call after each close operation

Version directories - An Example

```
#cd /usr/bar
#echo "Each file is a version directory" > foo; snapshot foo
#ls foo@
REGVOL_0 FILESNAP_1
```

Version directories - An Example

```
#cd /usr/bar
#echo "Each file is a version directory" > foo; snapshot foo
#ls foo@
REGVOL_0 FILESNAP_1

#echo "Each snapshot is a dir entry" > foo; snapshot /usr
#ls foo@
REGVOL_0 VOLSNAP_1 FILESNAP_2
```

Version directories - An Example

```
#cd /usr/bar
#echo "Each file is a version directory" > foo; snapshot foo
#ls foo@
REGVOL_0 FILESNAP_1

#echo "Each snapshot is a dir entry" > foo; snapshot /usr
#ls foo@
REGVOL_0 VOLSNAP_1 FILESNAP_2

#diff /usr/bar@2/foo /usr/bar/foo@1
< Each file is a version directory
---
> Each snapshot is a dir entry
```

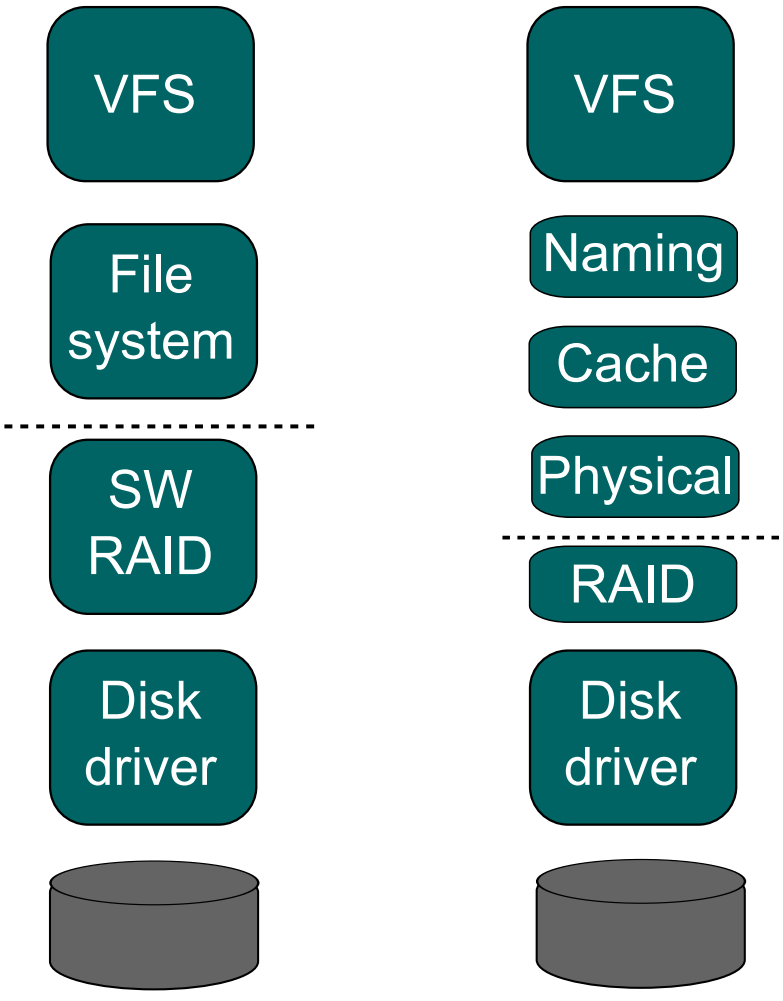
Evaluation

- File volume virtualization adds negligible overhead
  - Macro and micro-benchmarks showed less than 8% overhead
  - Absence of heavy metadata footprint unlike block-level systems
- All types of snapshotting added less than 5% overhead
  - Efficient block-granular snapshotting avoids copying data

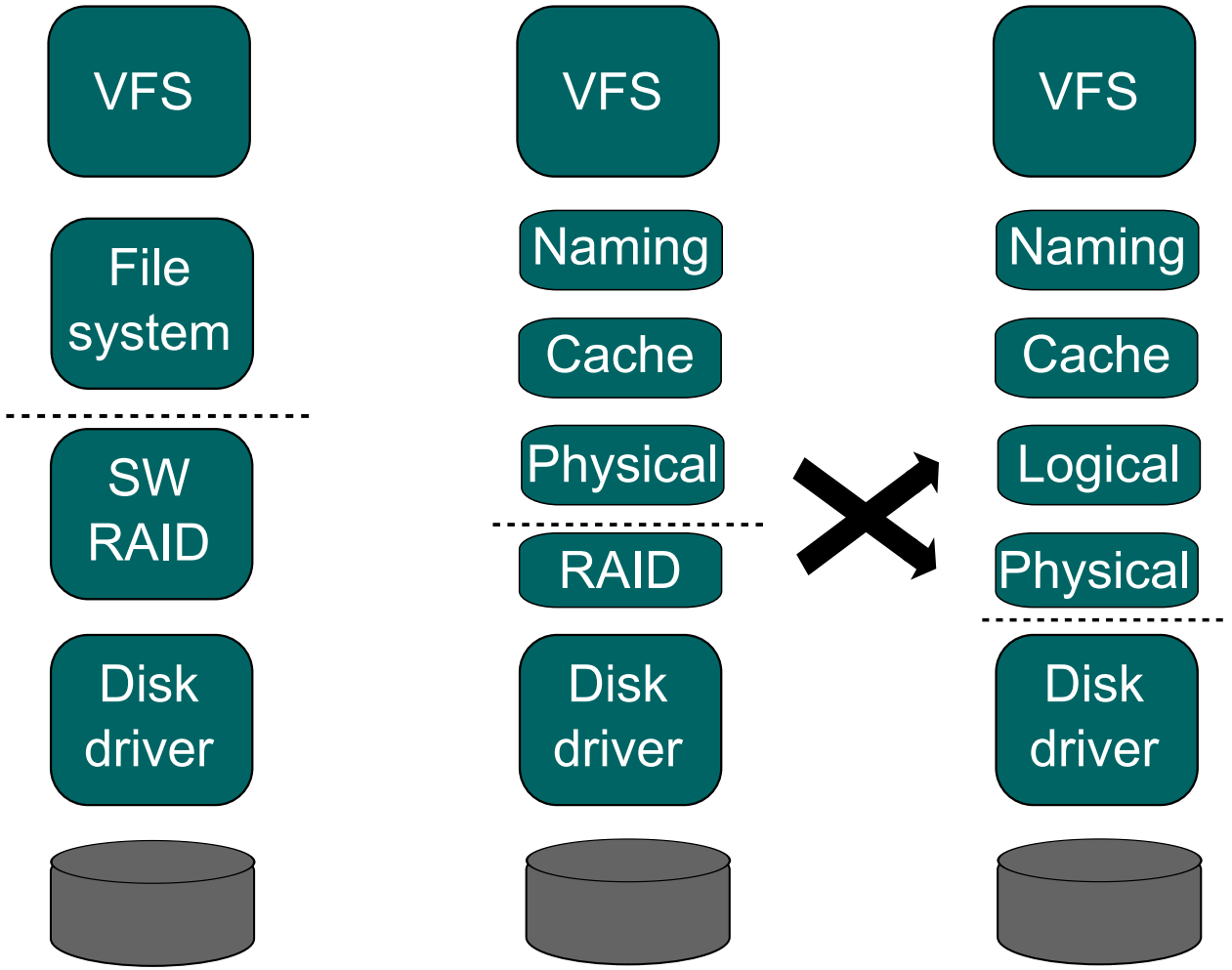
Conclusion

- We highlighted several flexibility and heterogeneity issues with the traditional stack
- We showed how Loris simplifies device management using File Pools - our new storage model
- We showed how Loris supports flexible, modular file volume virtualization and snapshotting

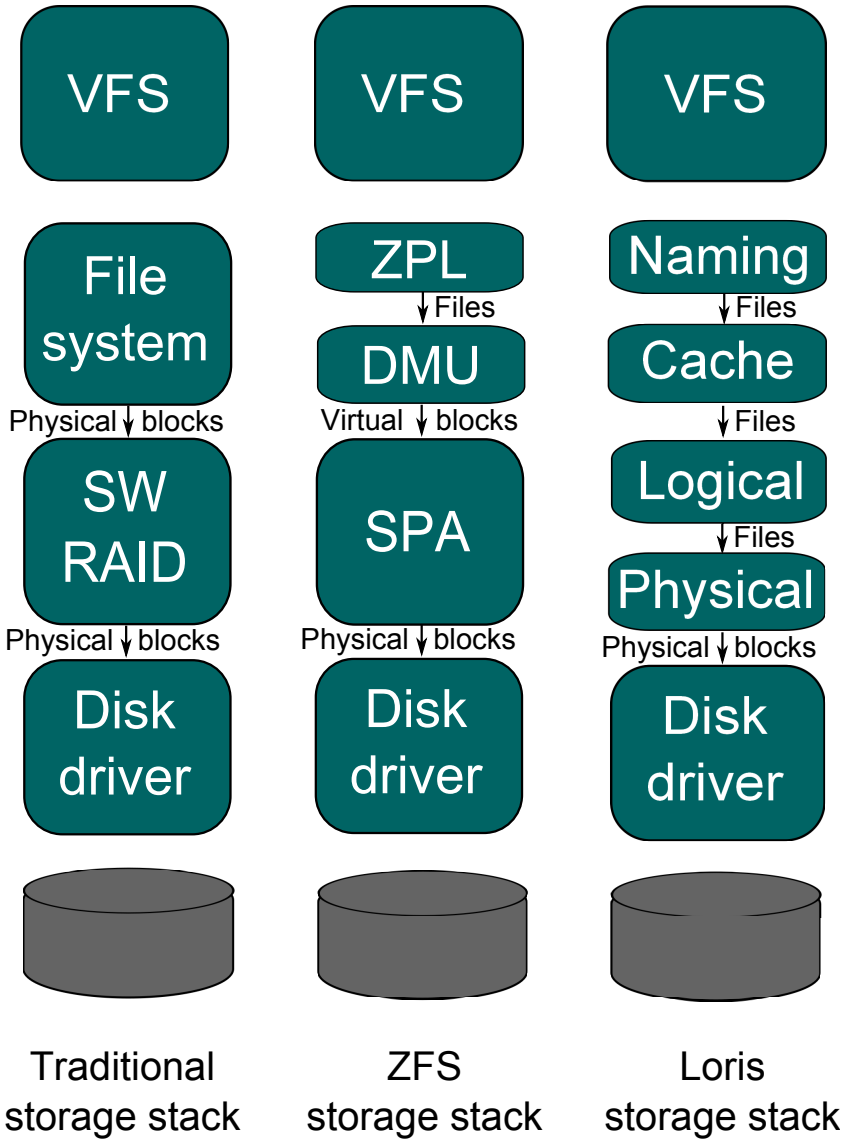
Conceptual Comparison - Modular Split (1)



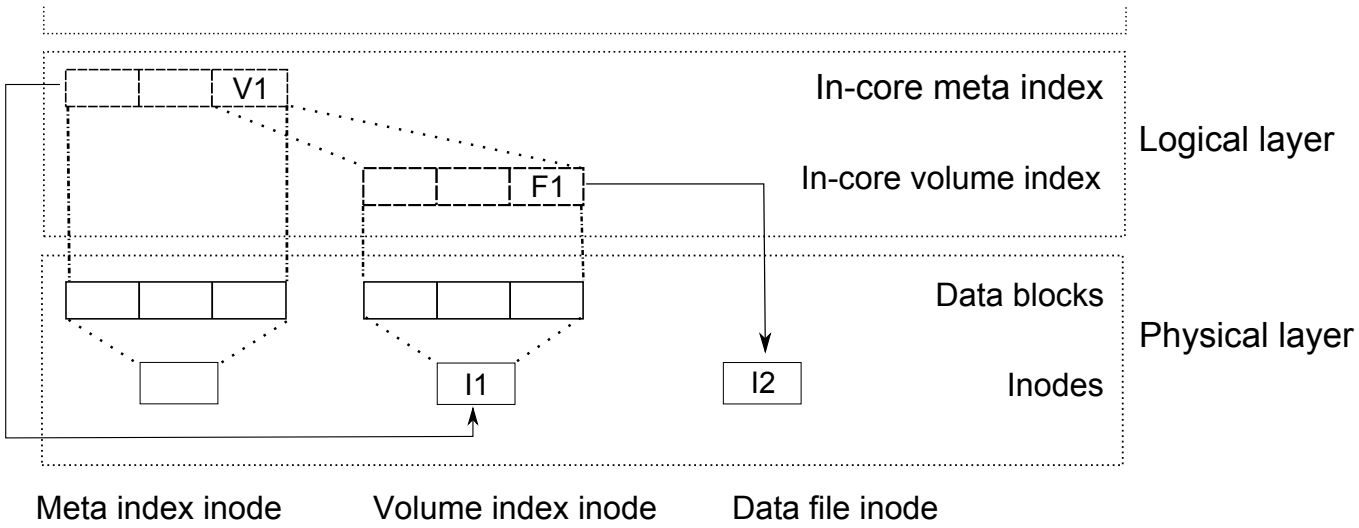
Conceptual Comparison - Reliable Flip (2)



ZFS Comparison



File Volume Virtualization in Loris - Data Structures



- Each file volume has a volume index file
  - One entry per logical file in that volume
  - F1 = <RAID=1, PFILE=<P1:I2>>
- File volumes themselves tracked using meta index
  - < *V*1,REGULAR_VOL, VOLIDX=<RAID=1, PFILE=<P1:I1>>>

Fully Automated Storage Tiering

Perf metric	Preferred tier	Secondary Tier
SEQ READ	HDD	SSD
SEQ WRITE	HDD	SSD
RANDOM READ	SSD	HDD
RANDOM WRITE	HDD	SSD

File type	Tier assigned
size = SMALL, rw = R	SSD
size = SMALL, rw = W	HDD
size = SMALL, rw = RW	SSD
size = LARGE, rw = R, atype = SEQ	HDD
size = LARGE, rw = R, atype = RAND	SSD
size = LARGE, rw = W, atype = SEQ	HDD
size = LARGE, rw = W, atype = RAND	HDD
size = LARGE, rw = RW, atype = SEQ	HDD
size = LARGE, rw = RW, atype = RAND	SSD