ZFS overview

And how ZFS is used in ECE/CIS
At the University of Delaware

www.eecis.udel.edu

Ben Miller

ZFS

File system and volume manager

After five years of development first appeared in OpenSolaris, later in Solaris 10 6/06 (u2).

Has been ported to FreeBSD and MacOS X

Linux has license issues... FUSE port done

Jeff Bonwick is a UDel (Math) graduate

ZFS main features

Simple administration
 - cli in two commands: zpool and zfs

Pooled storage

Transactional semantics
 - uses copy on write (COW)
 - always consistent on disk

End-to-end data integrity
 - blocks are checksummed
 - in replicated configs data is repaired

Scalability
 - 128 bit filesystem

ZFS pools

Disk storage is managed more like RAM than traditionally

A ZFS pool is a set of disks (usually) that are pooled together

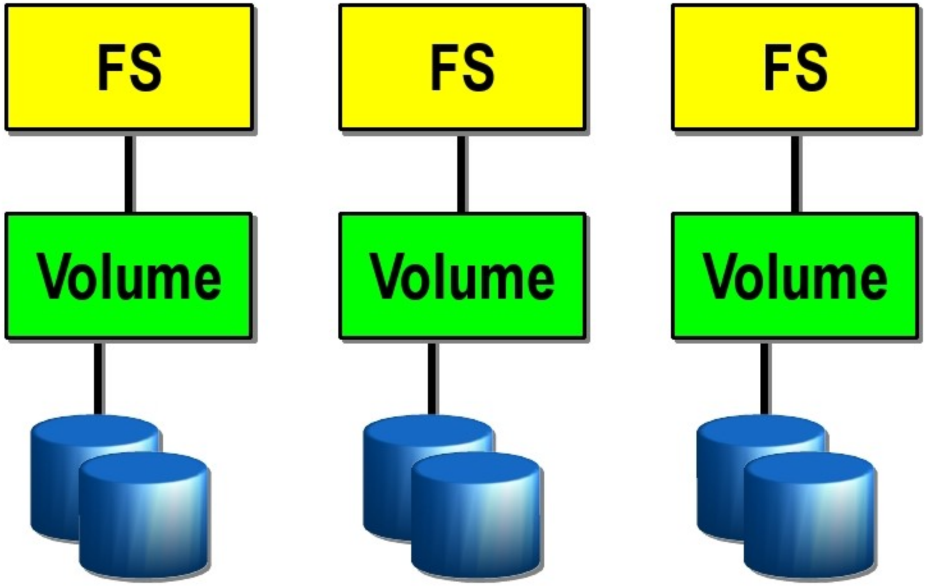
Filesystems (and zvols) can then be created on top of the zpools (dynamic)

All operations are COW which keeps the on-disk state consistent.

FS/Volume Model vs. Pooled Storage

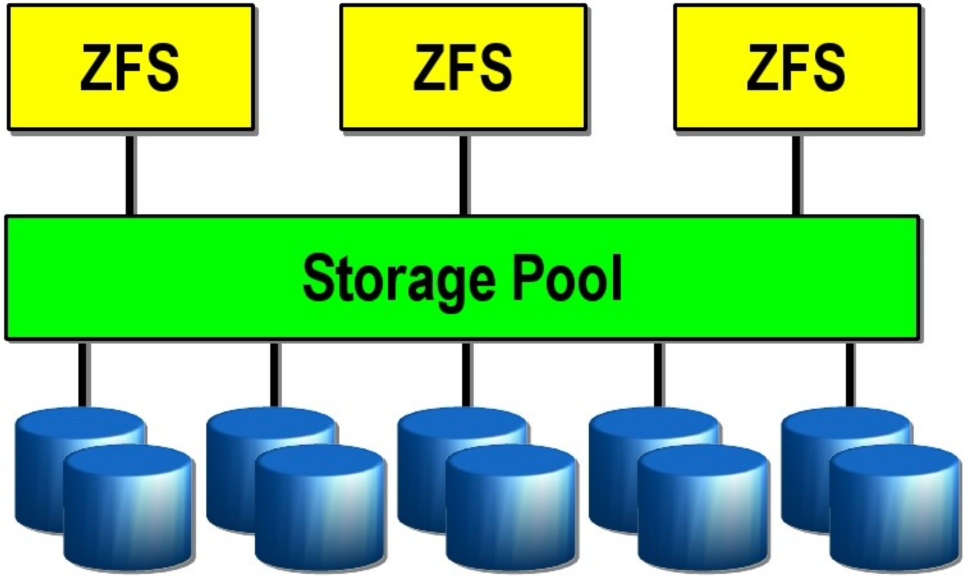
Traditional Volumes

- Abstraction: virtual disk
- Partition/volume for each FS
- Grow/shrink by hand
- Each FS has limited bandwidth
- Storage is fragmented, stranded



ZFS Pooled Storage

- Abstraction: malloc/free
- No partitions to manage
- Grow/shrink automatically
- All bandwidth always available
- All storage in the pool is shared



FS/Volume Interfaces vs. ZFS

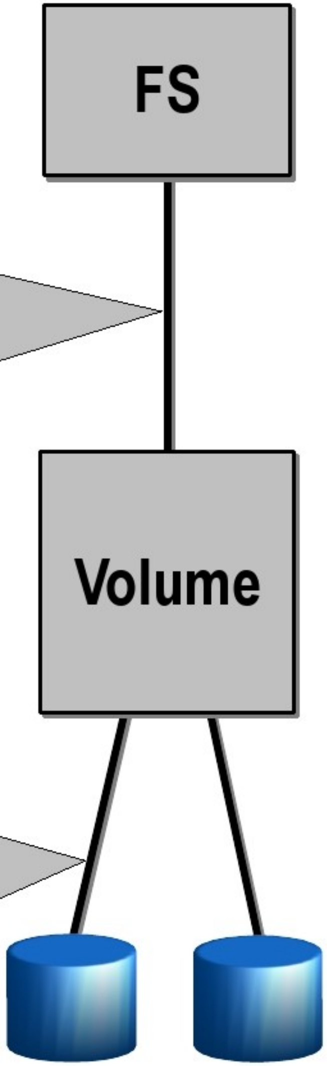
FS/Volume I/O Stack

Block Device Interface

- “Write this block, then that block, ...”
- Loss of power = loss of on-disk consistency
- Workaround: journaling, which is slow & complex

Block Device Interface

- Write each block to each disk immediately to keep mirrors in sync
- Loss of power = resync
- Synchronous and slow



ZFS I/O Stack

Object-Based Transactions

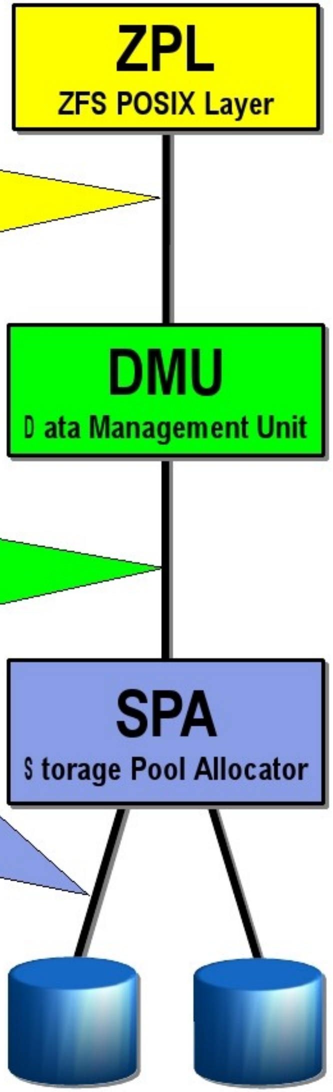
- “Make these 7 changes to these 3 objects”
- Atomic (all-or-nothing)

Transaction Group Commit

- Atomic for entire group
- Always consistent on disk
- No journal – not needed

Transaction Group Batch I/O

- Schedule, aggregate, and issue I/O at will
- No resync if power lost
- Runs at platter speed



ZFS RAID features

Mirroring is supported
 - N-way mirrors are possible

RAIDZ is similar to RAID5

RAIDZ2 is similar to RAID6 (double parity)

No redundancy is also possible
 - dynamic striping
 - usually not recommended

RAIDZ

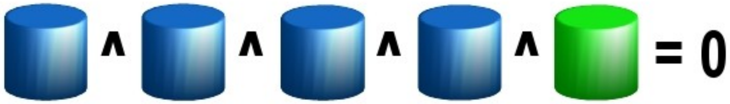
Works similar to RAID5

- have one extra disk and spread parity over all the disks
- can operate in degraded mode with one failed disk

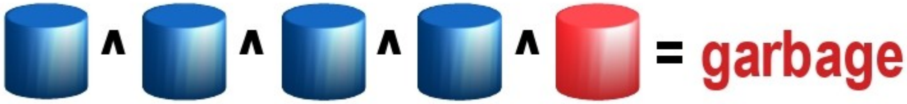
- uses variable stripe width which does away with the RAID5 write hole
- RAIDZ2 is double parity
- prefer JBOD with ZFS rather than hardware RAID

Traditional RAID-4 and RAID-5

- Several data disks plus one parity disk



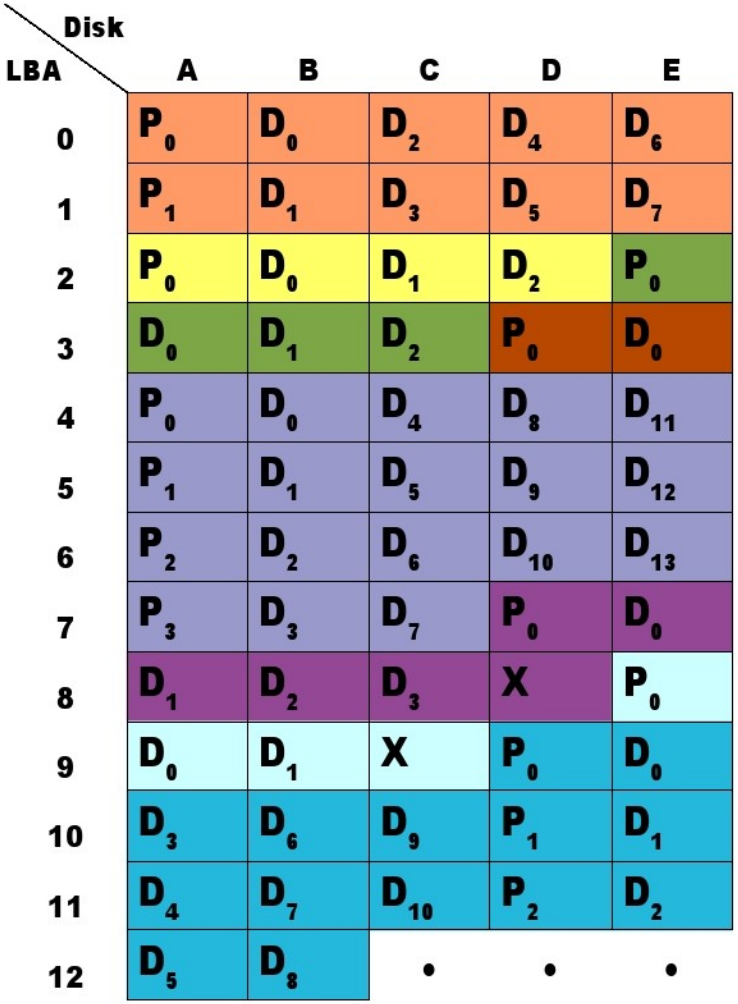
- Fatal flaw: partial stripe writes
  - Parity update requires read-modify-write (slow)
    - Read old data and old parity (two synchronous disk reads)
    - Compute new parity = new data ^ old data ^ old parity
    - Write new data and new parity
  - Suffers from *write hole:*



    - Loss of power between data and parity writes will corrupt data
    - Workaround: $$$ NVRAM in hardware (i.e., don't lose power!)
- Can't detect or correct silent data corruption

RAID-Z

- Dynamic stripe width
  - Variable block size: 512 – 128K
  - Each logical block is its own stripe
- All writes are full-stripe writes
  - Eliminates read-modify-write (it's fast)
  - Eliminates the RAID-5 write hole (no need for NVRAM)
- Both single- and double-parity
- Detects and corrects silent data corruption
  - Checksum-driven combinatorial reconstruction
- No special hardware – ZFS loves cheap disks



Disk LBA	A	B	C	D	E
0	P_0	D_0	D_2	D_4	D_6
1	P_1	D_1	D_3	D_5	D_7
2	P_0	D_0	D_1	D_2	P_0
3	D_0	D_1	D_2	P_0	D_0
4	P_0	D_0	D_4	D_8	D_{11}
5	P_1	D_1	D_5	D_9	D_{12}
6	P_2	D_2	D_6	D_{10}	D_{13}
7	P_3	D_3	D_7	P_0	D_0
8	D_1	D_2	D_3	X	P_0
9	D_0	D_1	X	P_0	D_0
10	D_3	D_6	D_9	P_1	D_1
11	D_4	D_7	D_{10}	P_2	D_2
12	D_5	D_8	•	•	•

ZFS with JBOD

No need for NVRAM in hardware or expensive RAID controllers.

ZFS works very well with JBOD (preferred)

Makes enterprise class storage much less expensive

Better to use JBOD and disable hardware RAID!

ZFS error correction

When a disk fails in a replicated config the replacement will be resilvered.

ZFS provides scrubbing (like ECC mem) to detect errors and correct the data.

During a scrub the pool is traversed and the 256-bit checksum for each block is checked. Can happen while pool is in use.

ZFS snapshots

Similar to NetApps WAFL snapshots (see patent lawsuit)

Read-only image of the filesystem at the point it is taken.

Multiple snapshots can be taken
- good for online backups

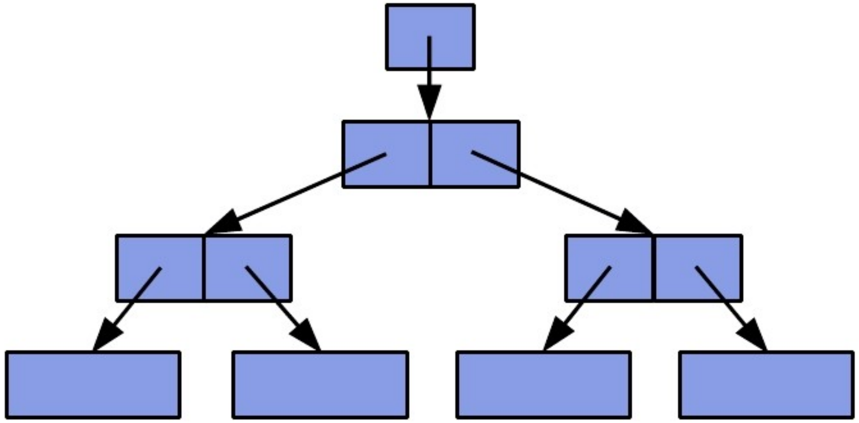
A clone is a writable snapshot and can be mounted elsewhere.

A snapshot takes no space initially

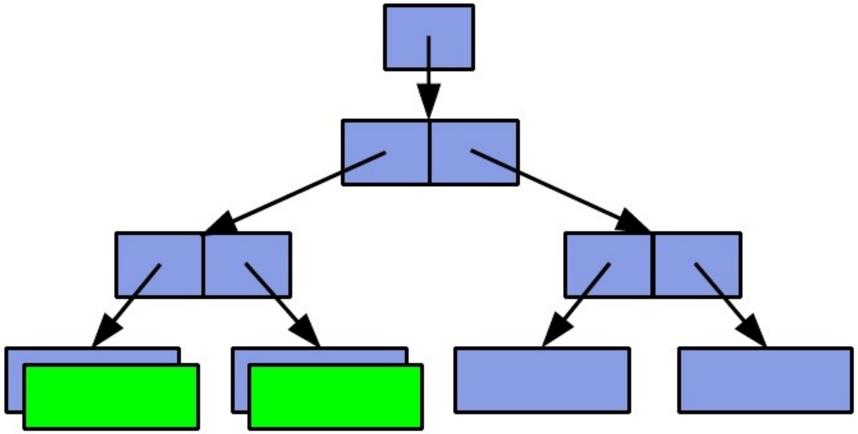
As a result of COW space is used by snapshots and clones with changes.

Copy-On-Write Transactions

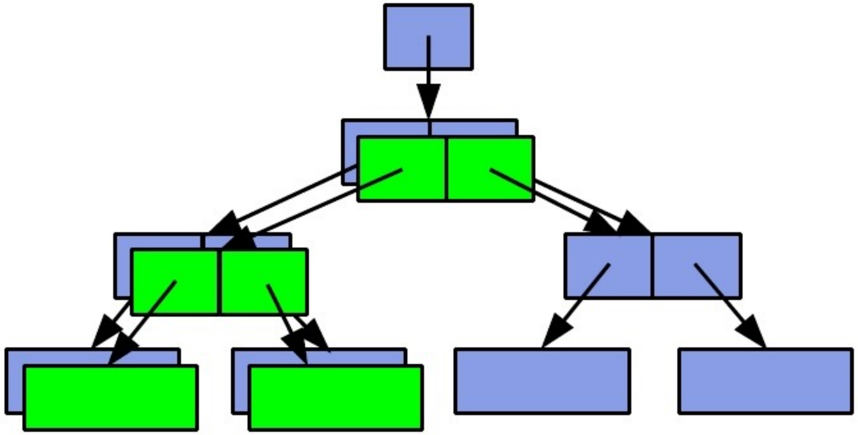
1. Initial block tree



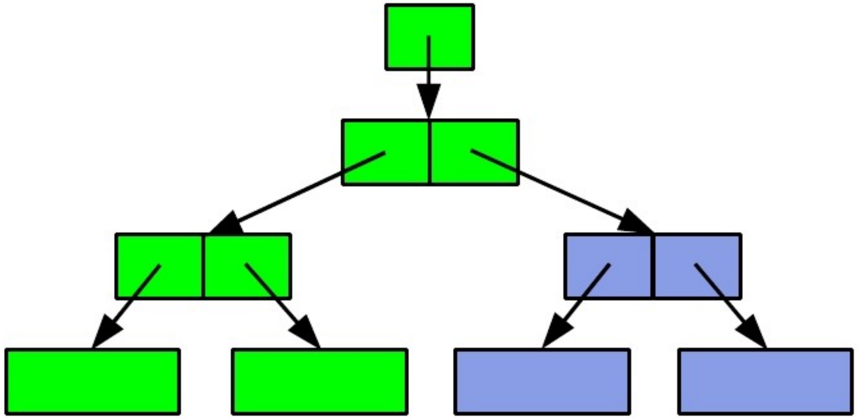
2. COW some blocks



3. COW indirect blocks

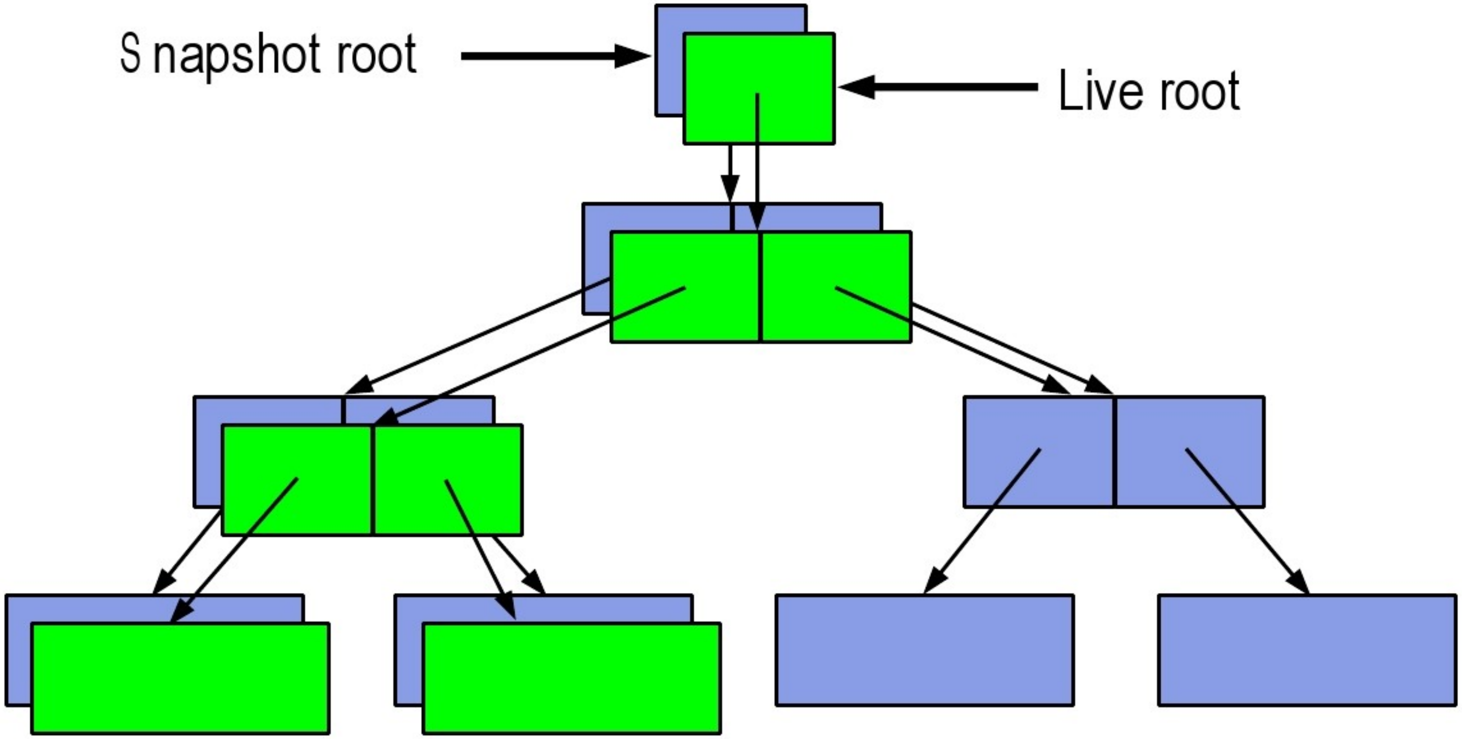


4. Rewrite uberblock (atomic)



Bonus: Constant-Time Snapshots

- At end of TX group, don't free COWed blocks
  - Actually cheaper to take a snapshot than not!

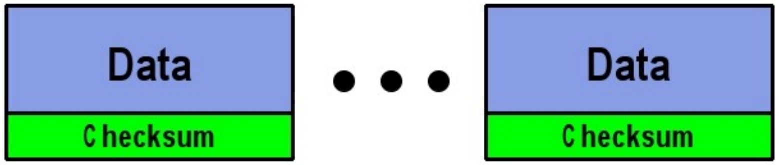


- The tricky part: how do you know when a block is free?

End-to-End Data Integrity in ZFS

Disk Block Checksums

- Checksum stored with data block
- Any self-consistent block will pass
- Can't detect stray writes
- Inherent FS/volume interface limitation

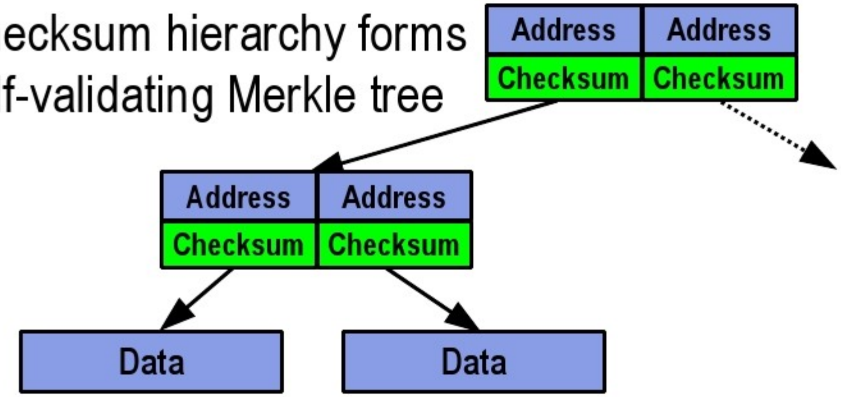


Disk checksum only validates media

- ✓ Bit rot
- ✗ Phantom writes
- ✗ Misdirected reads and writes
- ✗ DMA parity errors
- ✗ Driver bugs
- ✗ Accidental overwrite

ZFS Data Authentication

- Checksum stored in parent block pointer
- Fault isolation between data and checksum
- Checksum hierarchy forms self-validating Merkle tree

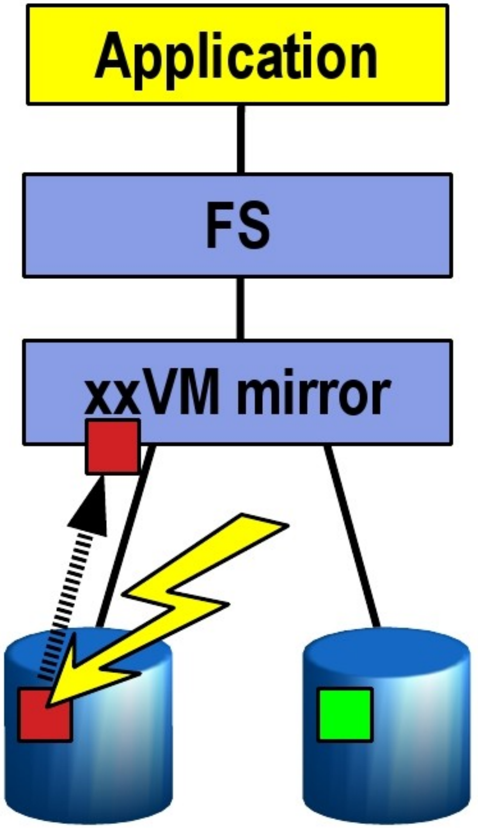


ZFS validates the entire I/O path

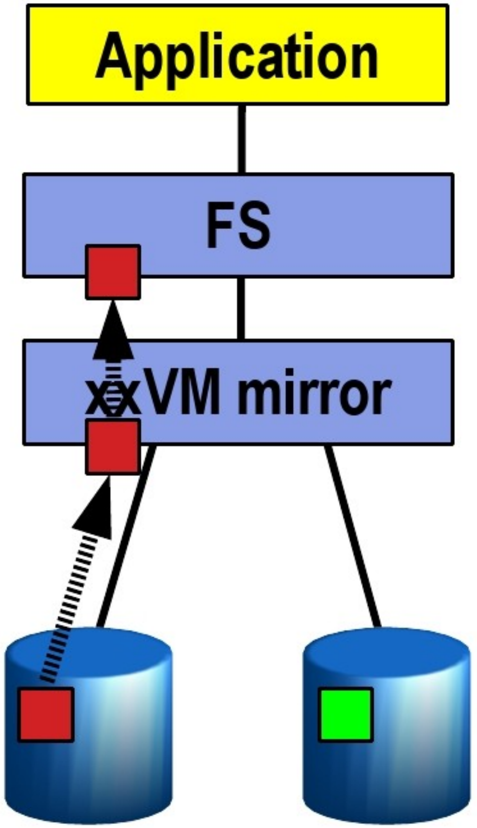
- ✓ Bit rot
- ✓ Phantom writes
- ✓ Misdirected reads and writes
- ✓ DMA parity errors
- ✓ Driver bugs
- ✓ Accidental overwrite

Traditional Mirroring

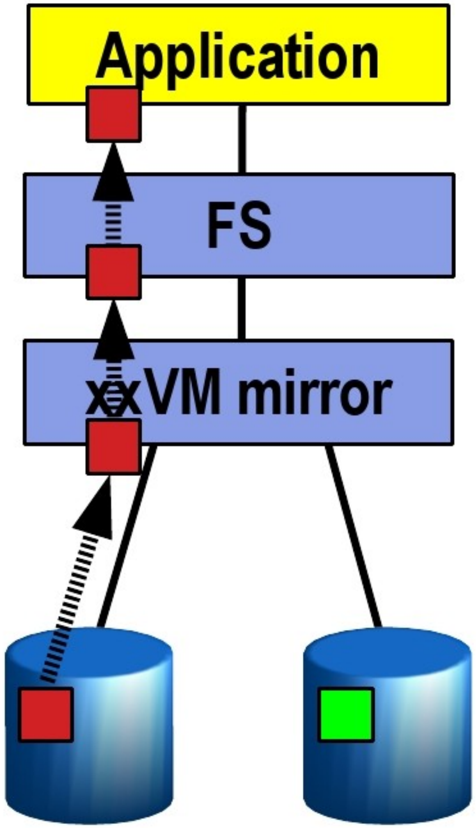
1. Application issues a read. Mirror reads the first disk, which has a corrupt block. It can't tell.



2. Volume manager passes bad block up to filesystem. If it's a metadata block, the filesystem panics. If not...

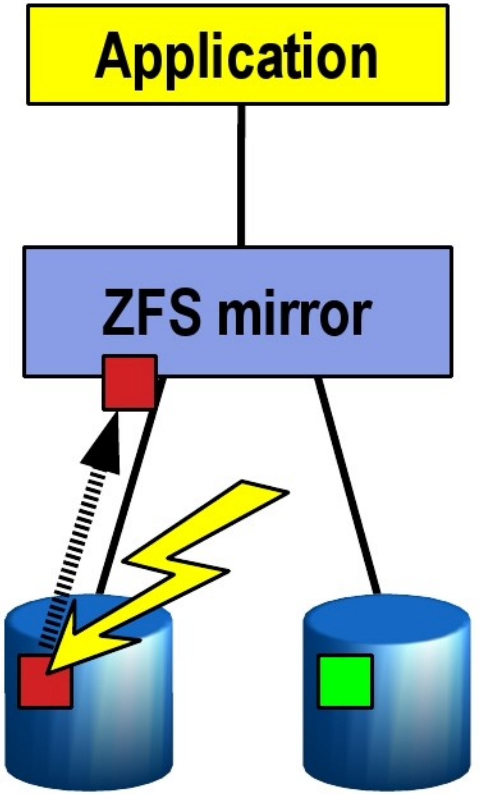


3. Filesystem returns bad data to the application.

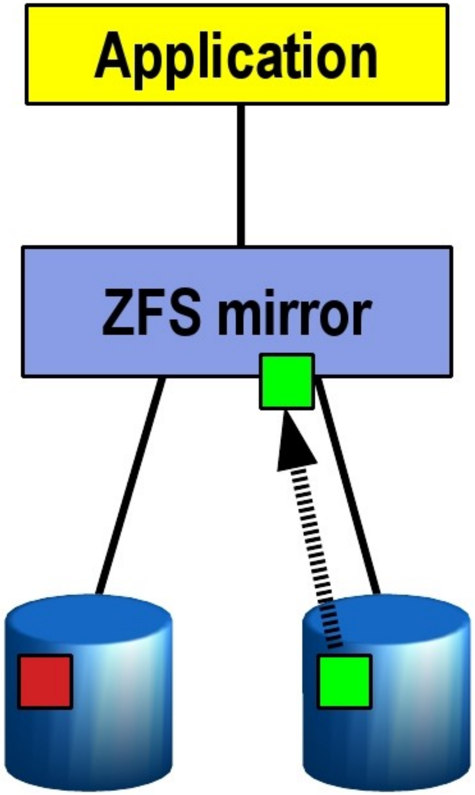


Self-Healing Data in ZFS

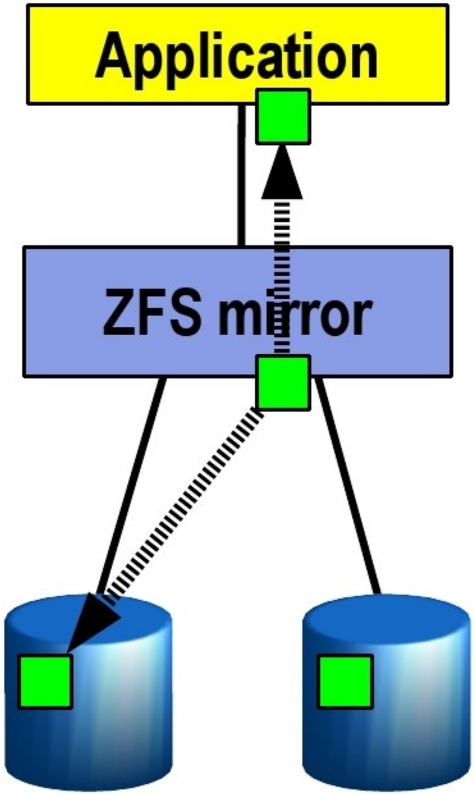
1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



2. ZFS tries the second disk. Checksum indicates that the block is good.



3. ZFS returns known good data to the application and repairs the damaged block.



ZFS properties

Another feature is that compression can be turned on at the fs level.

- saves space and I/O
- as a result may be faster than not doing compression depending on data.
- an all zero block takes no space

Encryption is being worked on (avail soon)

Many properties are integrated

- NFS sharing, mount options
- quotas, reservations

Zpool and ZFS versions

Actively developed and zpool version gets updated with new features

- zpool version up to 14 now
- ECE/CIS is using version 10 currently
- version 3 - raidz2 and hot spares
- version 6 - bootfs propety
- zpool upgrade -v
- details available at www.opensolaris.org

ZFS also has version numbers

- currently up to 3

ZFS use in ECE/CIS

ZFS first appeared in Nevada build 27a in November 2005 (OpenSolaris source).

Had to compile OpenSolaris sources at first to get ZFS support (kernel and O/N).

First test server a 32bit Dell dual Xeon
- had some 32 bit issues (ZFS likes 64bit)
- two pools, one on hardware RAID (ick) (ZFS likes JBODs much better)

ECE/CIS ZFS first production roll out

After getting some things improved/fixed that first test server was put into production.

Served two raidz pools (one from internal h/w RAID, one from a JBOD, both scsi).

Used as a samba server.

Worked until replaced in 2008 with an Amd64 system (SATA JBOD).

ECE/CIS Second big ZFS server

A Sun Fire X4200 used as the eecis mail server (postfix) in 2006.

Mail stored on a scsi JBOD (raidz) with 73GB disks.

Stored in Maildir format (one filesystem per user).

/var/postfix is a mirror on internal SAS

Later spamassassin put on dedicated SAS

Now uses ZFS boot also...

Other ECE/CIS uses

All home directories converted to either raidz or raidz2 (when made available).

Web servers

Zones – makes having lots of zones on a system easy! (zfs clones can be used).

Tape backup server

Backup replication server (we built half a thumper... 24 750GB SATA drives).

Everything using ZFS boot now, no more UFS.

More ECE/CIS uses

Clones used for diskless installation

Snapshots – done at noon, 6pm and 11pm
- noon and 6pm replace previous day
- 11pm replaces a week ago
- online backups for deleted/corrupted files.
- Also used in tape backups

NFS sharing can be much more finer grained than with UFS.

Replication server

AKA virtual tape system for laptop backups
  - use rsync from Windows/Mac
  - snapshots as well

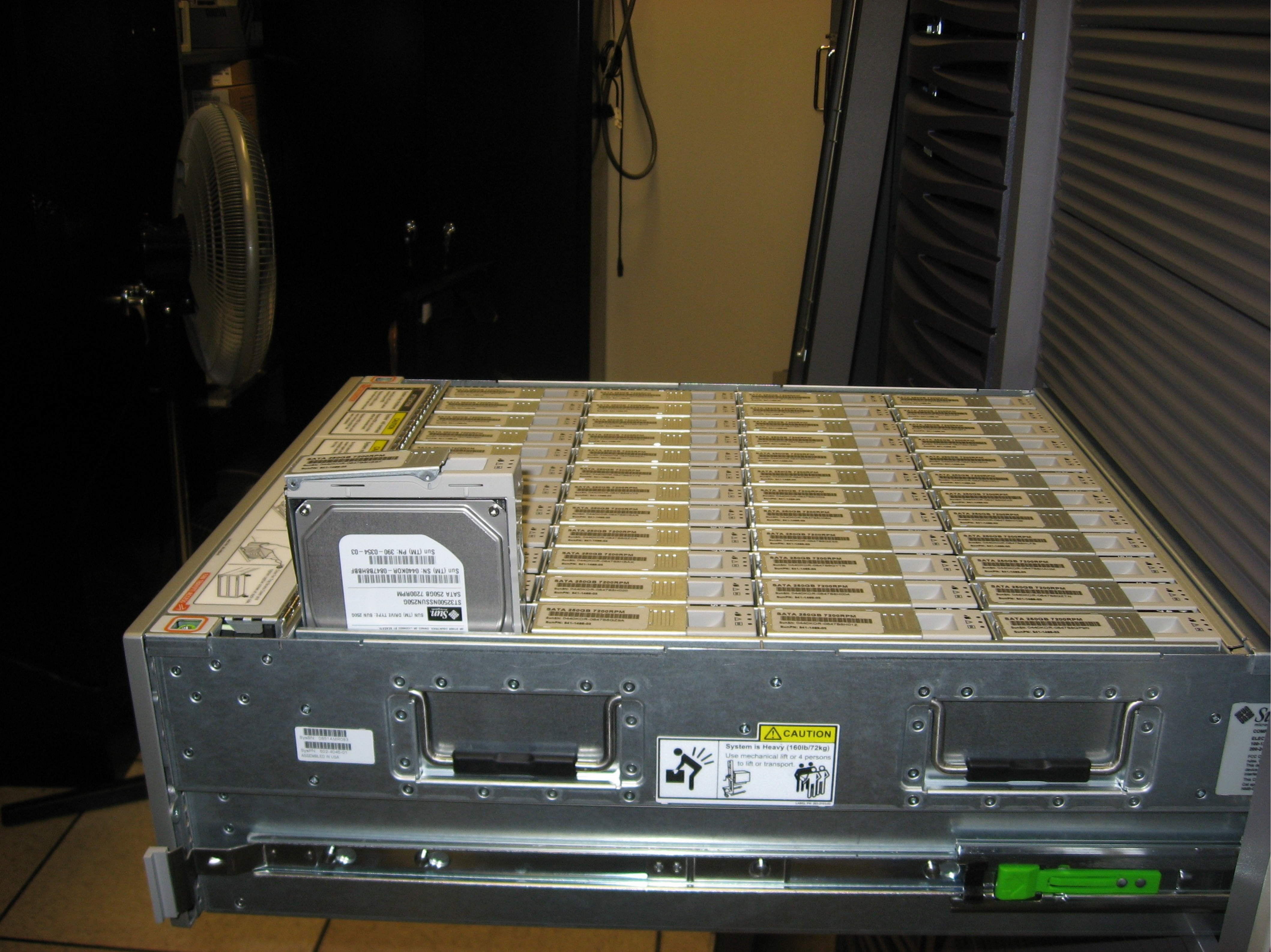
Extended for online replication of servers
  - rsync and then take a snapshot
  - also put to tape

Also some experimentation done with iscsi and TimeMachine.

Latest server – X4540 (Thor)

Follow-up to the X4500 (Thumper)
48 SATA drives, 32GB memory
2 quad AMD Opteron, CF slot
250GB drives to 1TB drives

Will be used by ECE/CIS to replace current nfs and samba servers
Will serve NFS, SMB, AFP and iSCSI
2 mirrored boot disks + CF for emergency
4 x (8+2) raidz2 = 40 disk pool
2 reserved for ZIL (SSD), 4 hot spare


SATA 250GB 7200RPM
CAUTION
System is Heavy (160lb/72kg)
Use mechanical lift or 4 persons to lift or transport.
ASSEMBLED IN USA

Quotas – one of the bigger problems

ZFS doesn't do user based quotas (will soon)

Quotas are set on filesystem.
- filesystems are cheap (*)
- give each user a filesystem

Because of COW trouble when 0 bytes left

The refquota option added to not count snapshots (has other bugs though).

Other Features used

Some filesystems are compressed
  - can actually be faster.

User delegation to destroy snapshots

Servers used mirrored boot drives
 - most other data is now raidz2

zpool status -x cron job to find problems

Creating filesystems a snap...

Creating Pools and Filesystems

- Create a mirrored pool named "tank"

```
# zpool create tank mirror c2d0 c3d0
```

- Create home directory filesystem, mounted at /export/home

```
# zfs create tank/home
# zfs set mountpoint=/export/home tank/home
```

- Create home directories for several users
  Note: automatically mounted at /export/home/{ahrens,bonwick,billm} thanks to inheritance

```
# zfs create tank/home/ahrens
# zfs create tank/home/bonwick
# zfs create tank/home/billm
```

- Add more space to the pool

```
# zpool add tank mirror c4d0 c5d0
```

Setting Properties

- Automatically NFS-export all home directories

```
# zfs set sharenfs=rw tank/home
```

- Turn on compression for everything in the pool

```
# zfs set compression=on tank
```

- Limit Eric to a quota of 10g

```
# zfs set quota=10g tank/home/eschrock
```

- Guarantee Tabriz a reservation of 20g

```
# zfs set reservation=20g tank/home/tabriz
```

ZFS Snapshots

- Read-only point-in-time copy of a filesystem
  - Instantaneous creation, unlimited number
  - No additional space used – blocks copied only when they change
  - Accessible through .zfs/snapshot in root of each filesystem
    - Allows users to recover files without sysadmin intervention
- Take a snapshot of Mark's home directory

```
# zfs snapshot tank/home/marks@tuesday
```

- Roll back to a previous snapshot

```
# zfs rollback tank/home/perrin@monday
```

- Take a look at Wednesday's version of foo.c

```
$ cat ~maybee/.zfs/snapshot/wednesday/foo.c
```

Other ZFS feature examples

```
# zfs clone tank/home@monday tank/user1
```

A full backup

```
# zfs send tank/fs@A | ssh ....
```

An incremental backup

```
# zfs send -i tank/fs@A tank/fs@B | ssh ...
```

Can be used for remote replication

The right side of the pipe would be like

```
  ssh host zfs receive -d /tank/fs
```

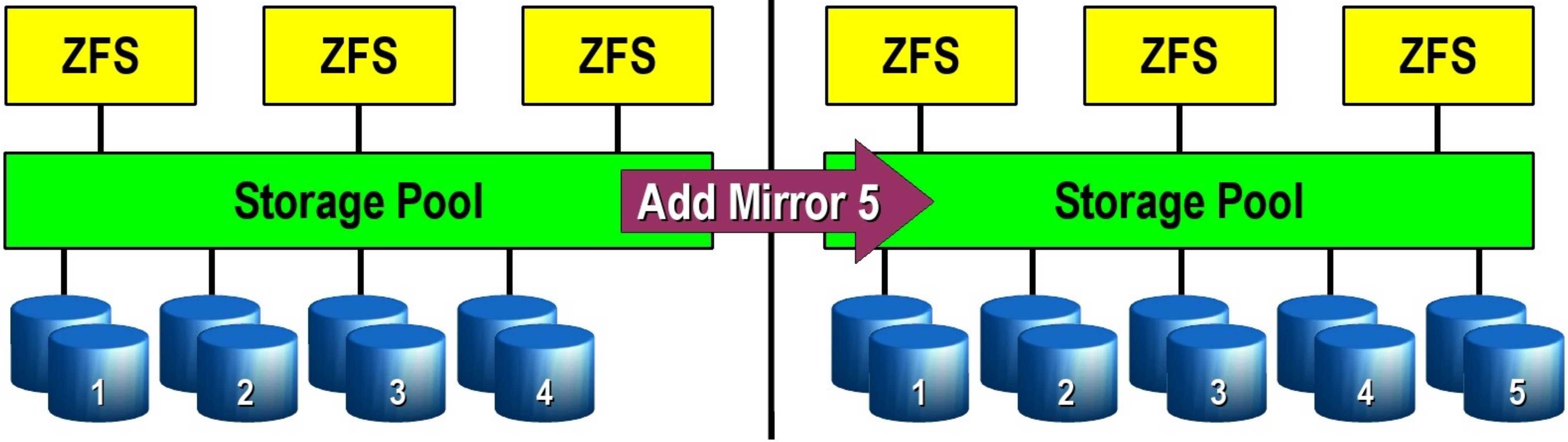
zpool export/import

Dynamic Striping

- Automatically distributes load across all devices

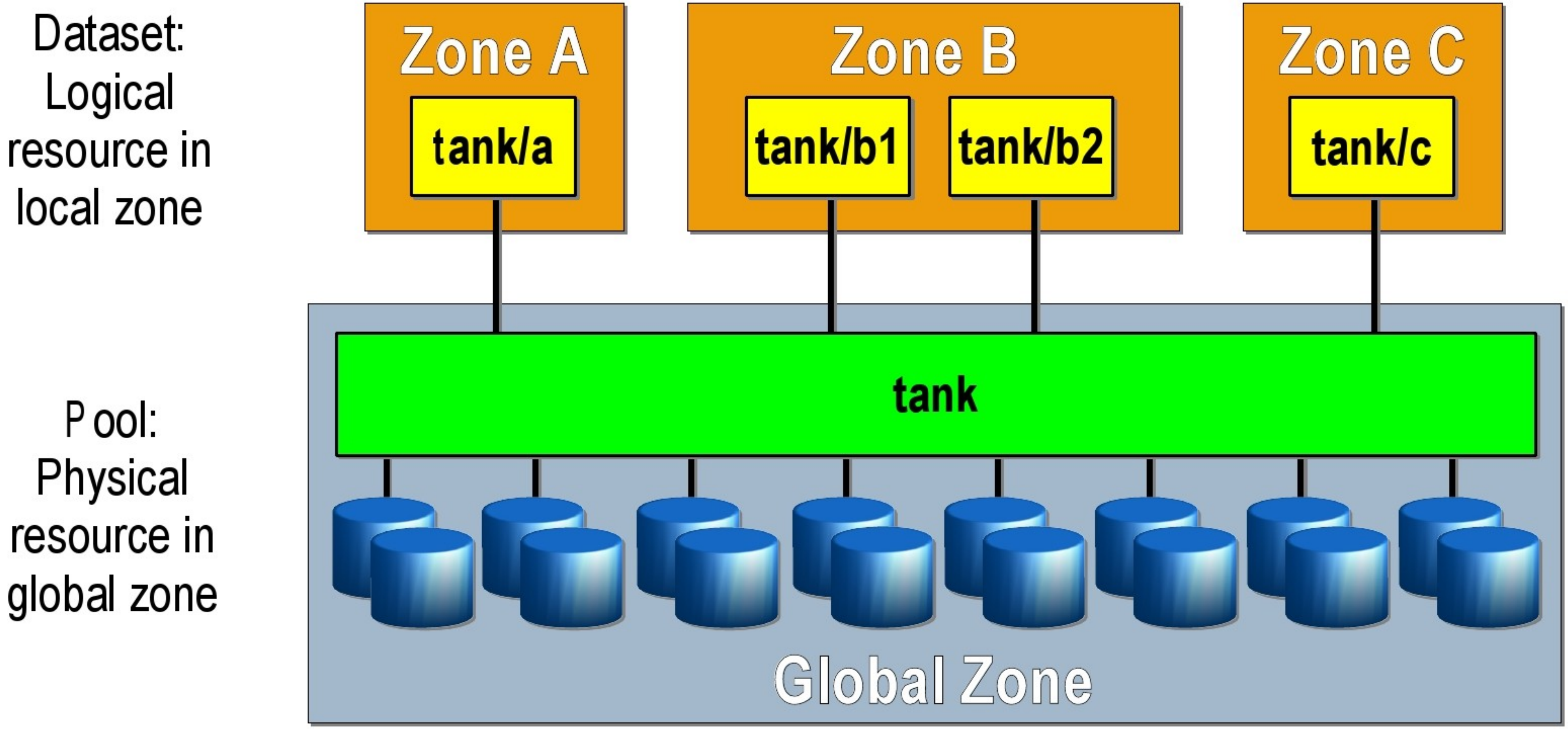
- Writes: striped across all four mirrors
- Reads: wherever the data was written
- Block allocation policy considers:
    - Capacity
    - Performance (latency, BW)
    - Health (degraded mirrors)

- Writes: striped across all five mirrors
- Reads: wherever the data was written
- No need to migrate existing data
    - Old data striped across 1-4
    - New data striped across 1-5
    - COW gently reallocates old data



ZFS and Zones (Virtualization)

- Secure – Local zones cannot even <u>see</u> physical devices
- Fast – snapshots and clones make zone creation instant



Dataset: Logical resource in local zone

Pool: Physical resource in global zone

Ditto Blocks

- Data replication above and beyond mirror/RAID-Z
  - Each logical block can have up to three physical blocks
    - Different devices whenever possible
    - Different places on the same device otherwise (e.g. laptop drive)
  - All ZFS metadata 2+ copies
    - Small cost in latency and bandwidth (metadata ≈ 1% of data)
  - Explicitly settable for precious user data
- Detects and corrects silent data corruption
  - In a multi-disk pool, ZFS survives any non-consecutive disk failures
  - In a single-disk pool, ZFS survives loss of up to 1/8 of the platter
- ZFS survives failures that send other filesystems to tape