ZFS

The Last Word In File Systems

Jeff Bonwick

Bill Moore

(modifed by yupu for CS736)

Trouble with Existing Filesystems

- No defense against silent data corruption
  - Any defect in disk, controller, cable, driver, laser, or firmware can corrupt data silently; like running a server without ECC memory
- Brutal to manage
  - Labels, partitions, volumes, provisioning, grow/shrink, /etc files...
  - Lots of limits: filesystem/volume size, file size, number of files, files per directory, number of snapshots ...
  - Different tools to manage file, block, iSCSI, NFS, CIFS ...
  - Not portable between platforms (x86, SPARC, PowerPC, ARM ...)
- Dog slow
  - Linear-time create, fat locks, fixed block size, naïve prefetch, dirty region logging, painful RAID rebuilds, growing backup time

ZFS Objective

End the Suffering

- Figure out why storage has gotten so complicated
- Blow away 20 years of obsolete assumptions
- Design an integrated system from scratch

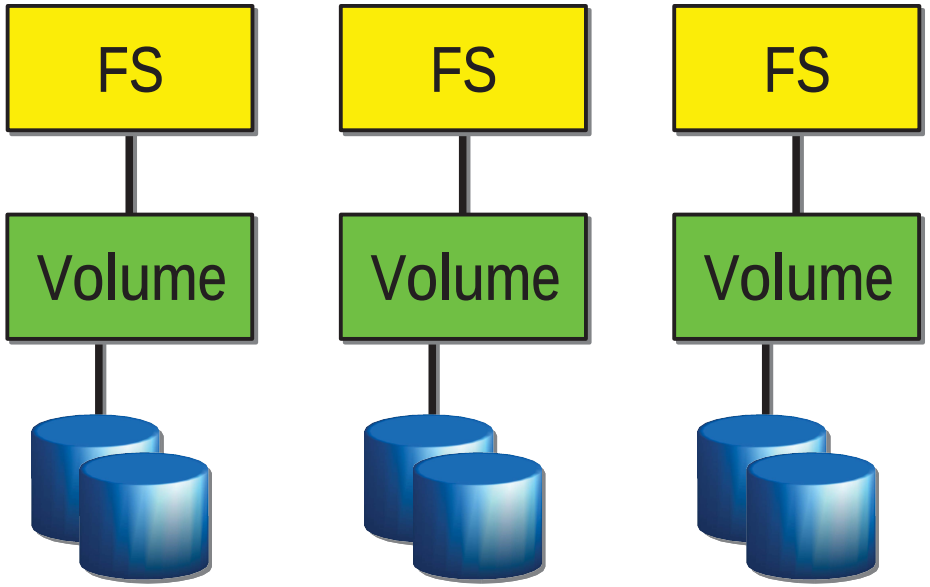
ZFS Overview

- Pooled storage
  - Completely eliminates the antique notion of volumes
  - Does for storage what VM did for memory
- Transactional object system
  - Always consistent on disk – no fsck, ever
  - Universal – file, block, iSCSI, swap ...
- Provable end-to-end data integrity
  - Detects and corrects silent data corruption
  - Historically considered “too expensive” – no longer true
- Simple administration
  - Concisely express your intent

FS/Volume Model vs. Pooled Storage

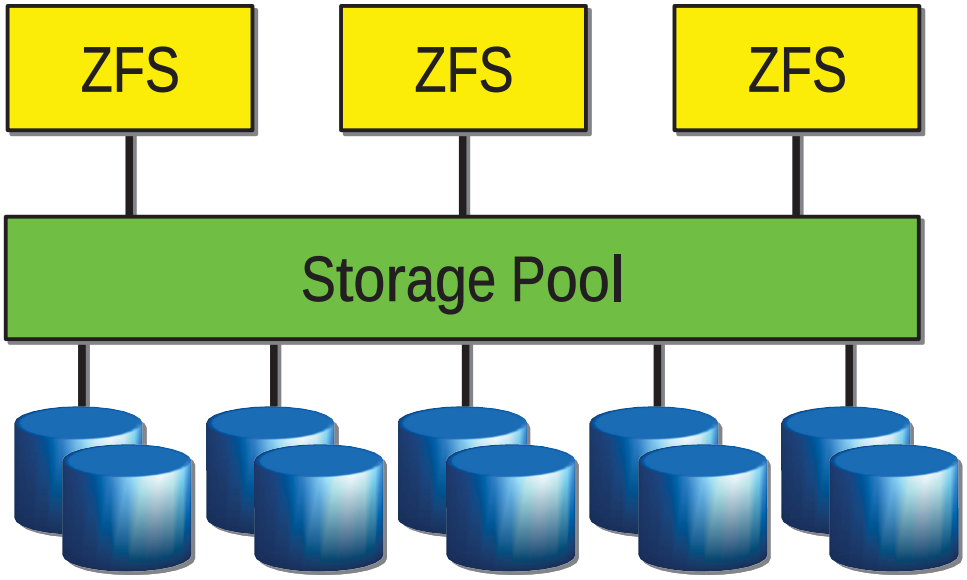
Traditional Volumes

- 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



ZFS Scalability

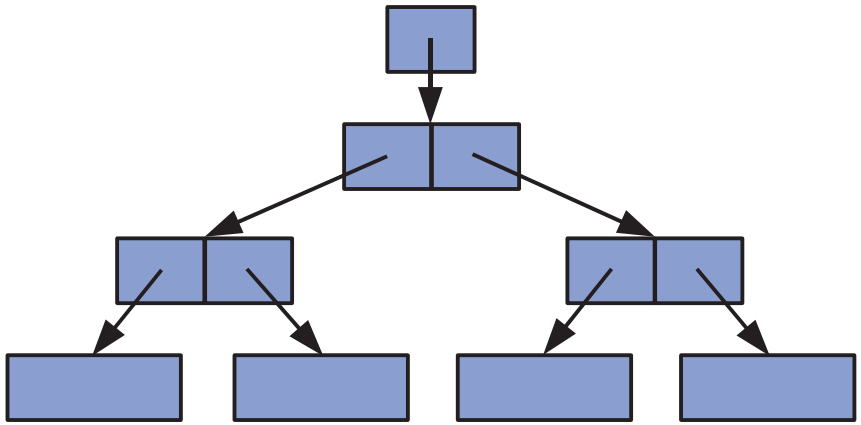
- Immense capacity (128-bit)
  - 1 zettabyte = 1 billion TB
  - ZFS Capacity = 256 quadrillion ZB
- No practical limits on number of
  - inodes
  - files
  - directories
  - snapshots

ZFS Data Integrity Model

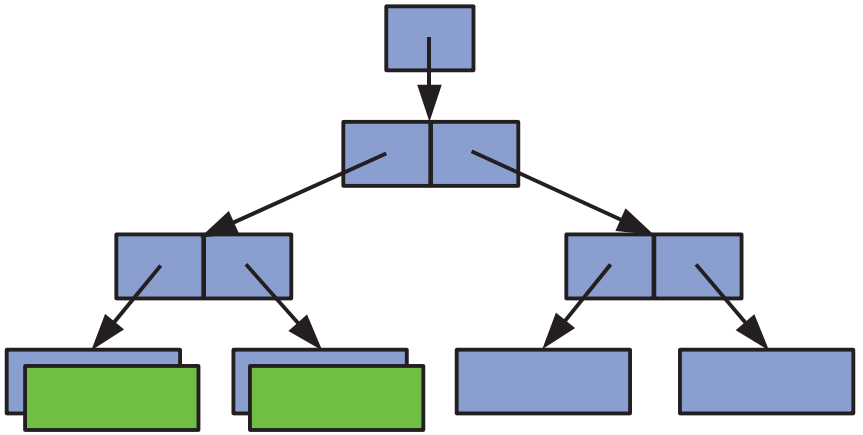
- Three Big Rules
  - All operations are copy-on-write
    - Never overwrite live data
    - On-disk state always valid – no "windows of vulnerability"
    - No need for fsck(1M)
  - All operations are transactional
    - Related changes succeed or fail as a whole
    - No need for journaling
  - All data is checksummed
    - No silent data corruption
    - No panics on bad metadata

Copy-On-Write

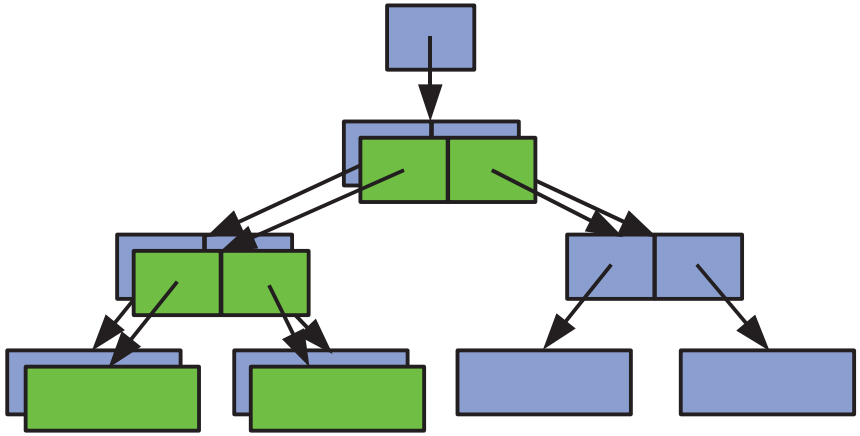
1. Initial block tree



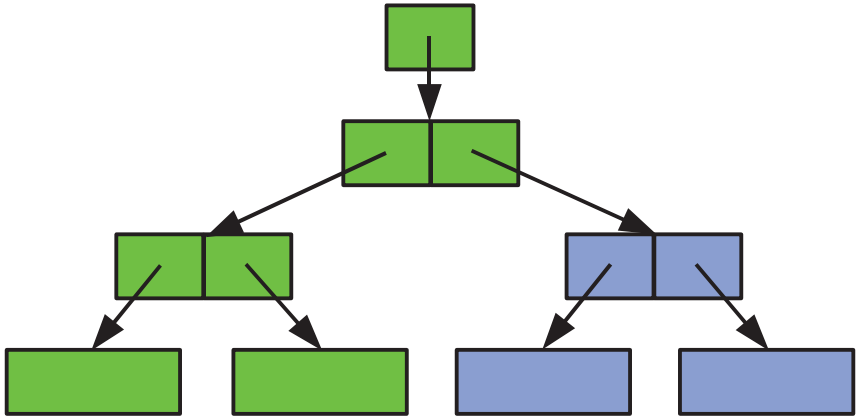
2. COW some blocks



3. COW indirect blocks



4. Rewrite uberblock (atomic)



Transactional Object System

- Everything is an object in ZFS
  - e.g., files, directories
  - System calls => modifications on objects
- Transaction
  - Each high-level opeartion is a transaction
- Transaction Group
  - Each transaction is added to a transaction group
  - A transaction group is commited to disk periodically as a whole
    - Either succeed or fail
    - Always consistent on disk

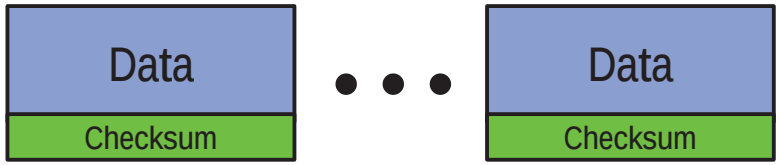
Trends in Storage Integrity

- Uncorrectable bit error rates have stayed roughly constant
  - 1 in 10^{14} bits (~12TB) for desktop-class drives
  - 1 in 10^{15} bits (~120TB) for enterprise-class drives (allegedly)
  - Bad sector every 8-20TB in practice (desktop and enterprise)
- Drive capacities doubling every 12-18 months
- Number of drives per deployment increasing
- → Rapid increase in error rates
- Both silent and “noisy” data corruption becoming more common
- Cheap flash storage will only accelerate this trend

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

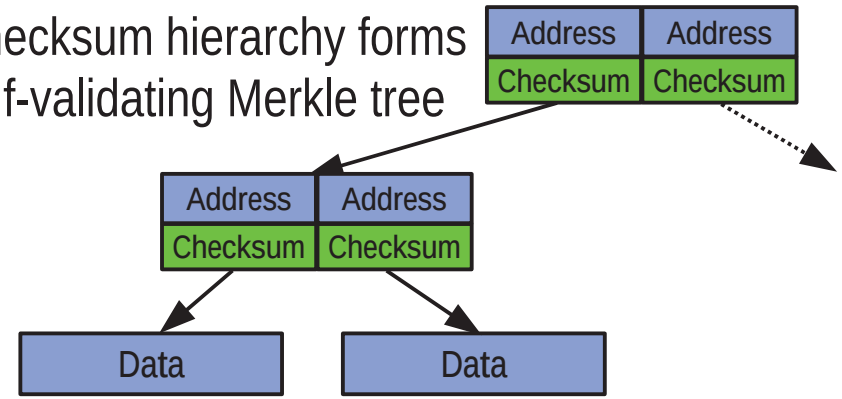


Disk checksum only validates media

✔	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

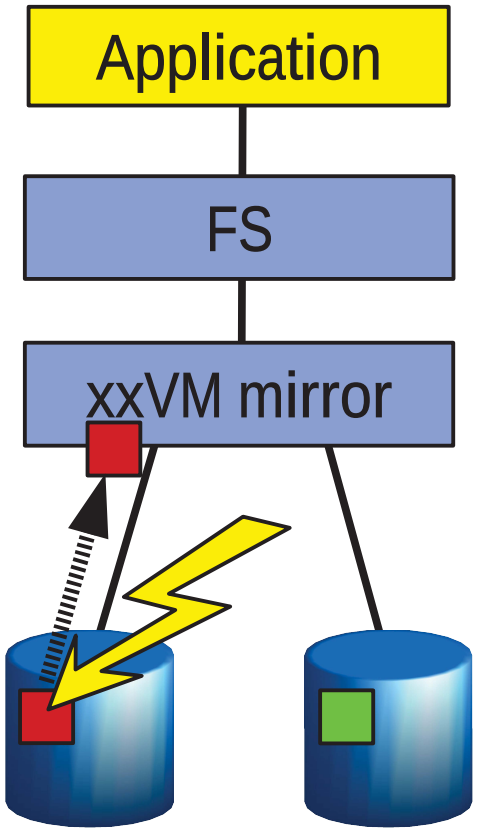


ZFS validates the entire I/O path

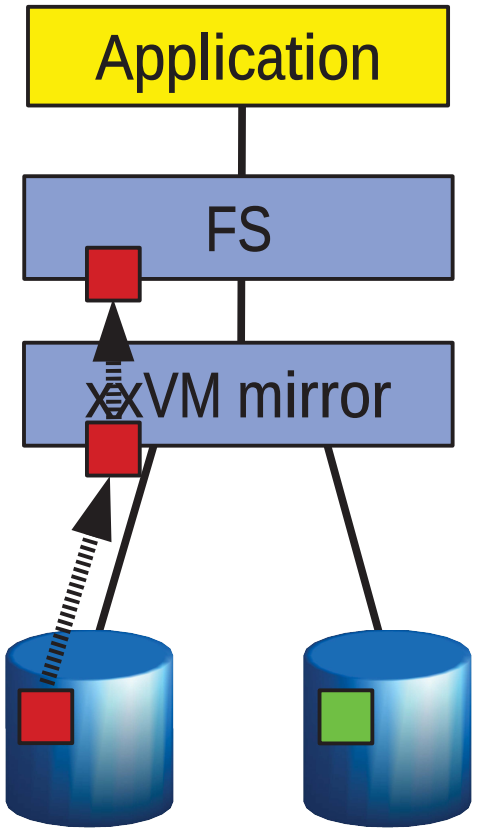
✔	Bit rot
✔	Phantom writes
✔	Misdirected reads and writes
✔	DMA parity errors
✔	Driver bugs
✔	Accidental overwrite

Traditional Mirroring

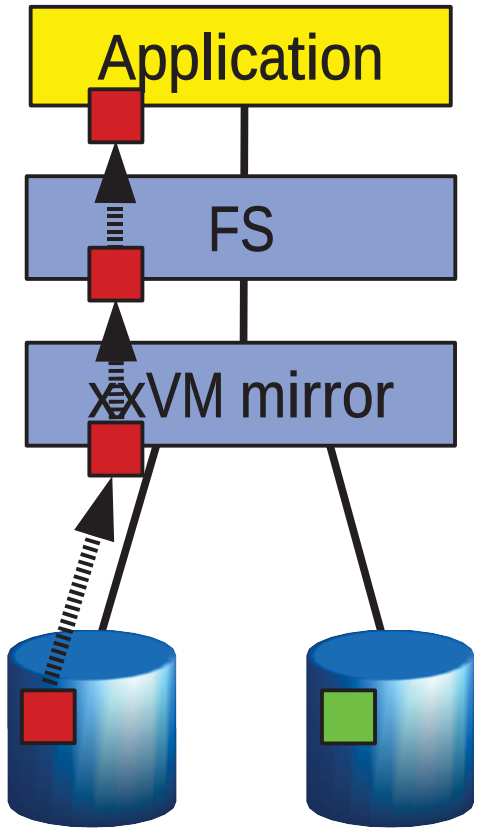
1. Application issues a read. Mirror reads the first disk, which has a corrupt block. It can't tell.



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

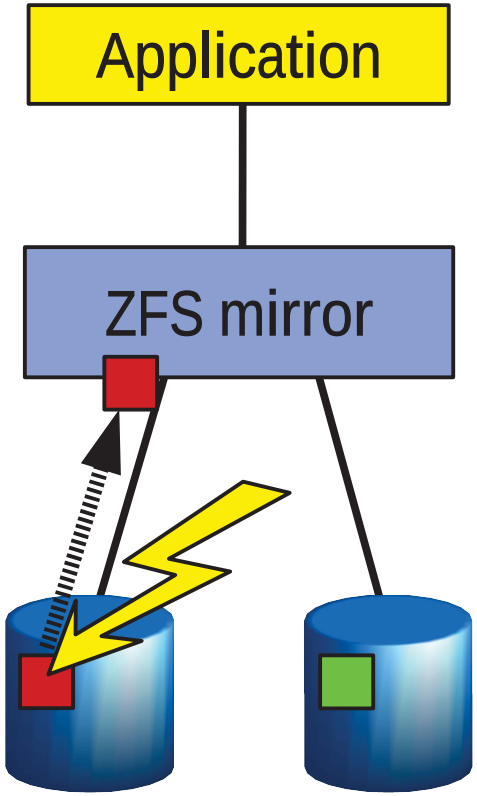


3. Filesystem returns bad data to the application.

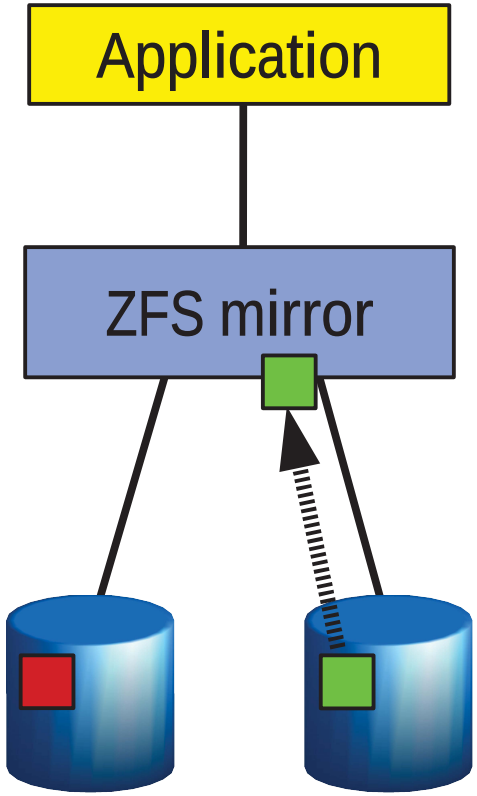


Self-Healing Data in ZFS

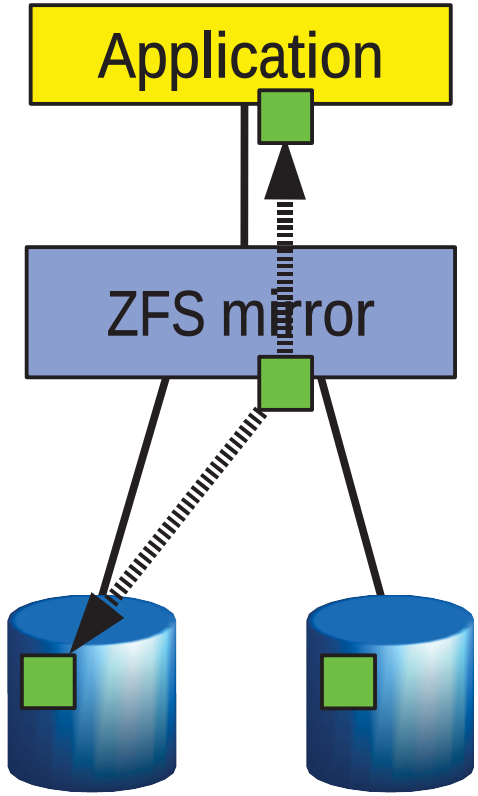
1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



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.



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

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
```

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
```