ZFS

THE LAST WORD IN FILE SYSTEMS

Bill Moore
Sr Staff Engineer
Sun Microsystems

ZFS Overview

- Provable data integrity
  - Detects and corrects silent data corruption
- Immense capacity
  - The world's first 128-bit filesystem
- Simple administration
  - “You're going to put a lot of people out of work.”
    – Jarod Jenson, ZFS beta customer
- Smokin' performance

Trouble With Existing Filesystems

- No defense against silent data corruption
  - Any defect in disk, controller, cable, driver, or firmware can corrupt data silently; like running a server without ECC memory
- Brutal to manage
  - Labels, partitions, volumes, provisioning, grow/shrink, /etc/vfstab...
  - Lots of limits: filesystem/volume size, file size, number of files, files per directory, number of snapshots, ...
  - Not portable between platforms (e.g. x86 to/from SPARC)
- Dog slow
  - Linear-time create, fat locks, fixed block size, naïve prefetch, slow random writes, dirty region logging

ZFS Objective

End the Suffering

- Data management should be a pleasure
  - Simple
  - Powerful
  - Safe
  - Fast

Design

You Can't Get There From Here

Free Your Mind

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

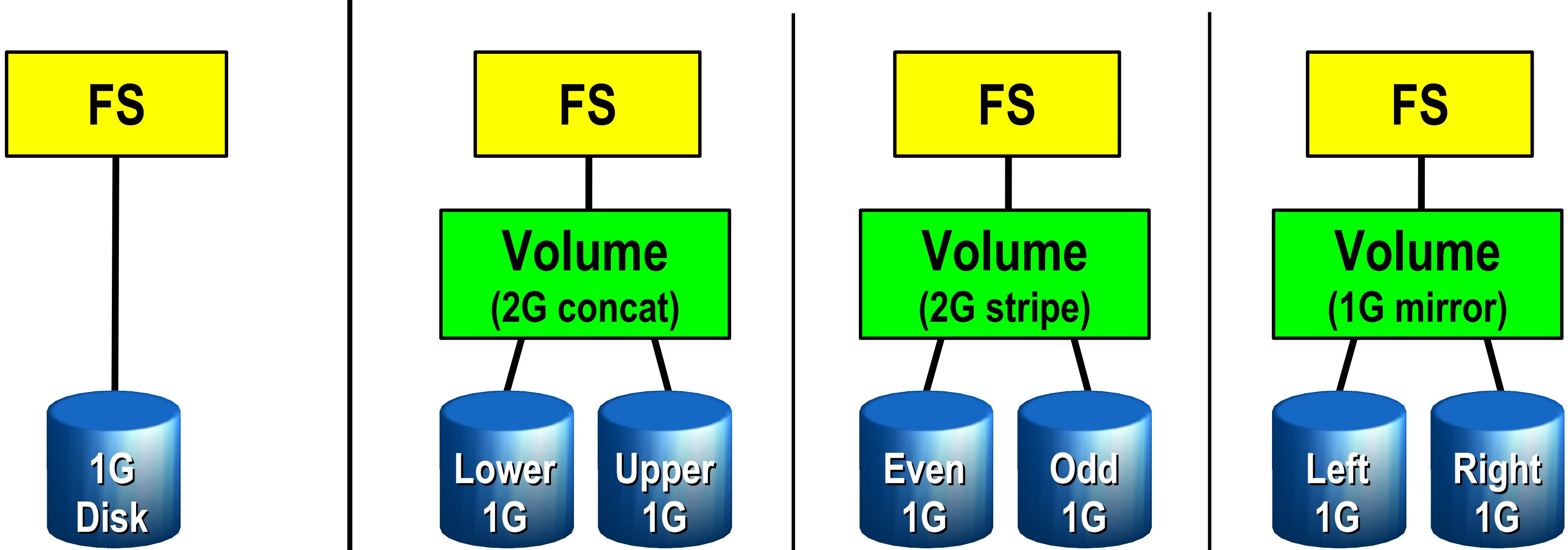
ZFS Design Principles

- Pooled storage
  - Completely eliminates the antique notion of volumes
  - Does for storage what VM did for memory
- End-to-end data integrity
  - Historically considered “too expensive”
  - Turns out, no it isn't
  - And the alternative is unacceptable
- Transactional operation
  - Keeps things always consistent on disk
  - Removes almost all constraints on I/O order
  - Allows us to get <u>huge</u> performance wins

Why Volumes Exist

In the beginning, each filesystem managed a single disk.

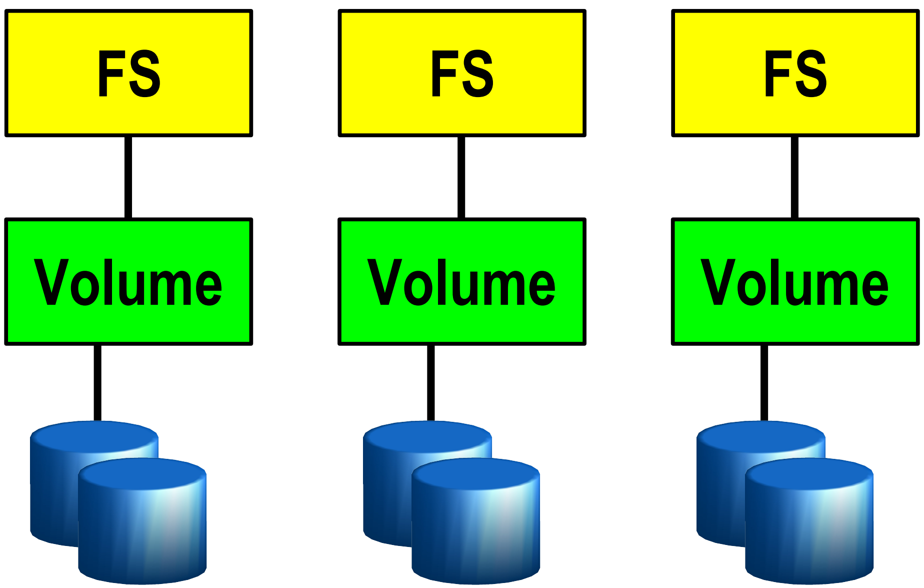
- Customers wanted more space, bandwidth, reliability
  - Rewrite filesystems to handle many disks: hard
  - Insert a little shim ("volume") to cobble disks together: easy
- An industry grew up around the FS/volume model
  - Filesystems, volume managers sold as separate products
  - Inherent problems in FS/volume <u>interface</u> can't be fixed



FS/Volume Model vs. ZFS

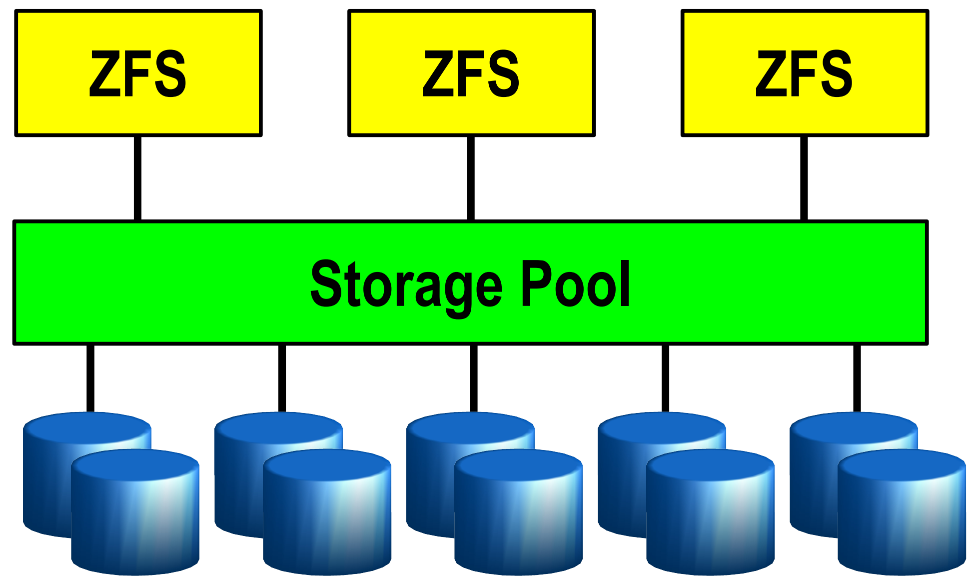
Traditional Volumes

- 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 Model vs. ZFS

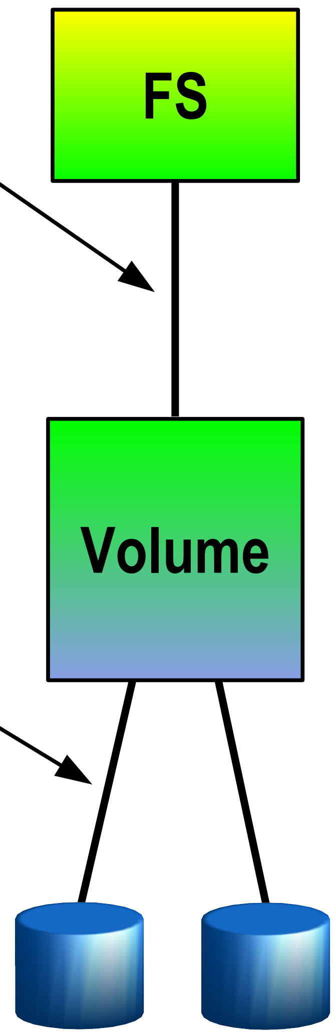
FS/Volume I/O Stack

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

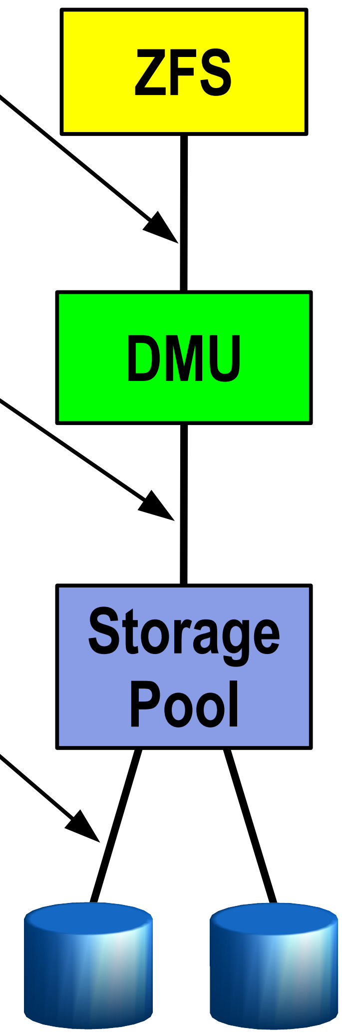
- "Make these 7 changes to these 3 objects"
- All-or-nothing

Transaction Group Commit

- Again, all-or-nothing
- 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



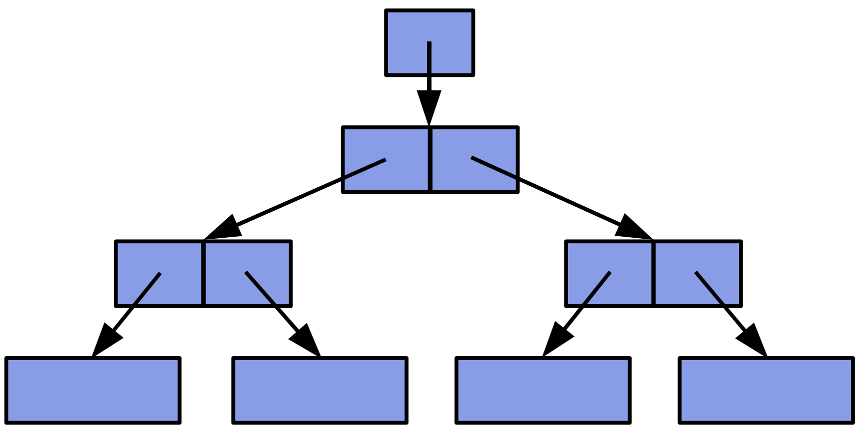
Data Integrity

ZFS Data Integrity Model

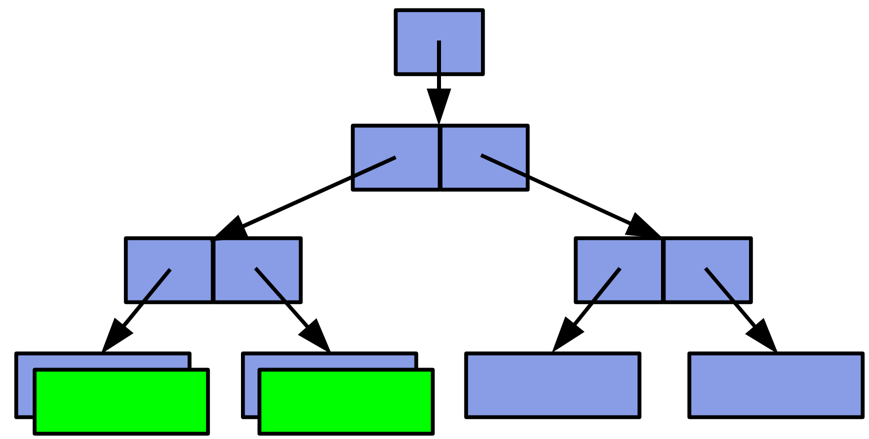
- Everything is copy-on-write
  - Never overwrite live data
  - On-disk state always valid – no “windows of vulnerability”
  - No need for fsck(1M)
- Everything is transactional
  - Related changes succeed or fail as a whole
  - No need for journaling
- Everything is checksummed
  - No silent data corruption
  - No panics due to silently corrupted metadata

Copy-On-Write Transactions

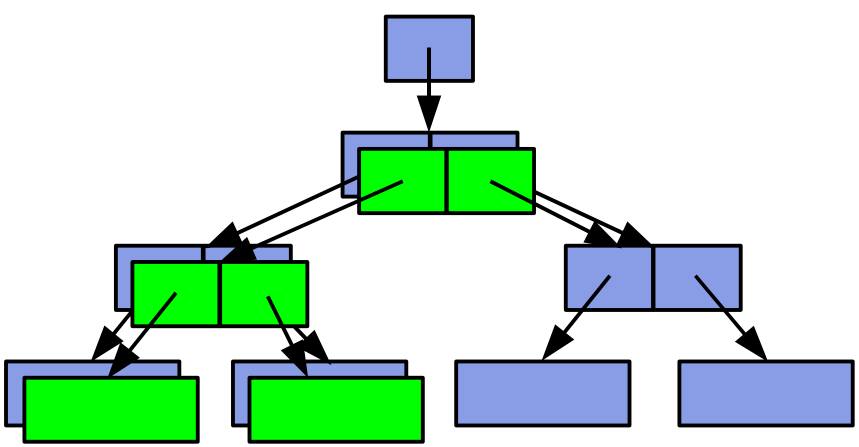
1. Initial block tree



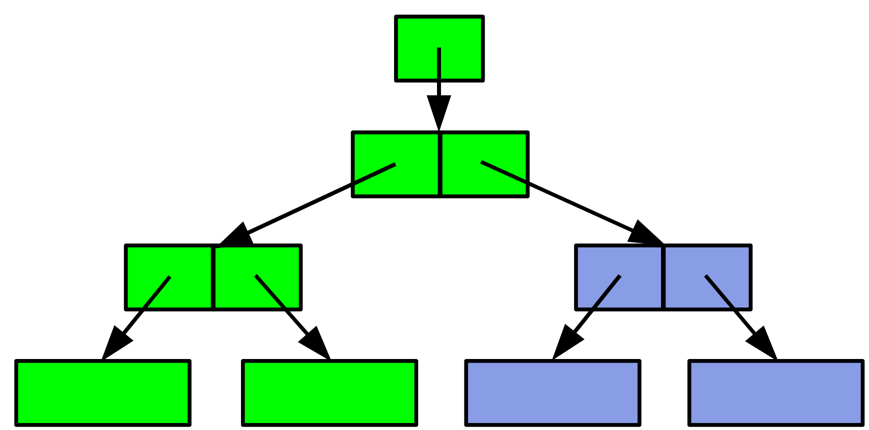
2. COW some blocks



3. COW indirect blocks

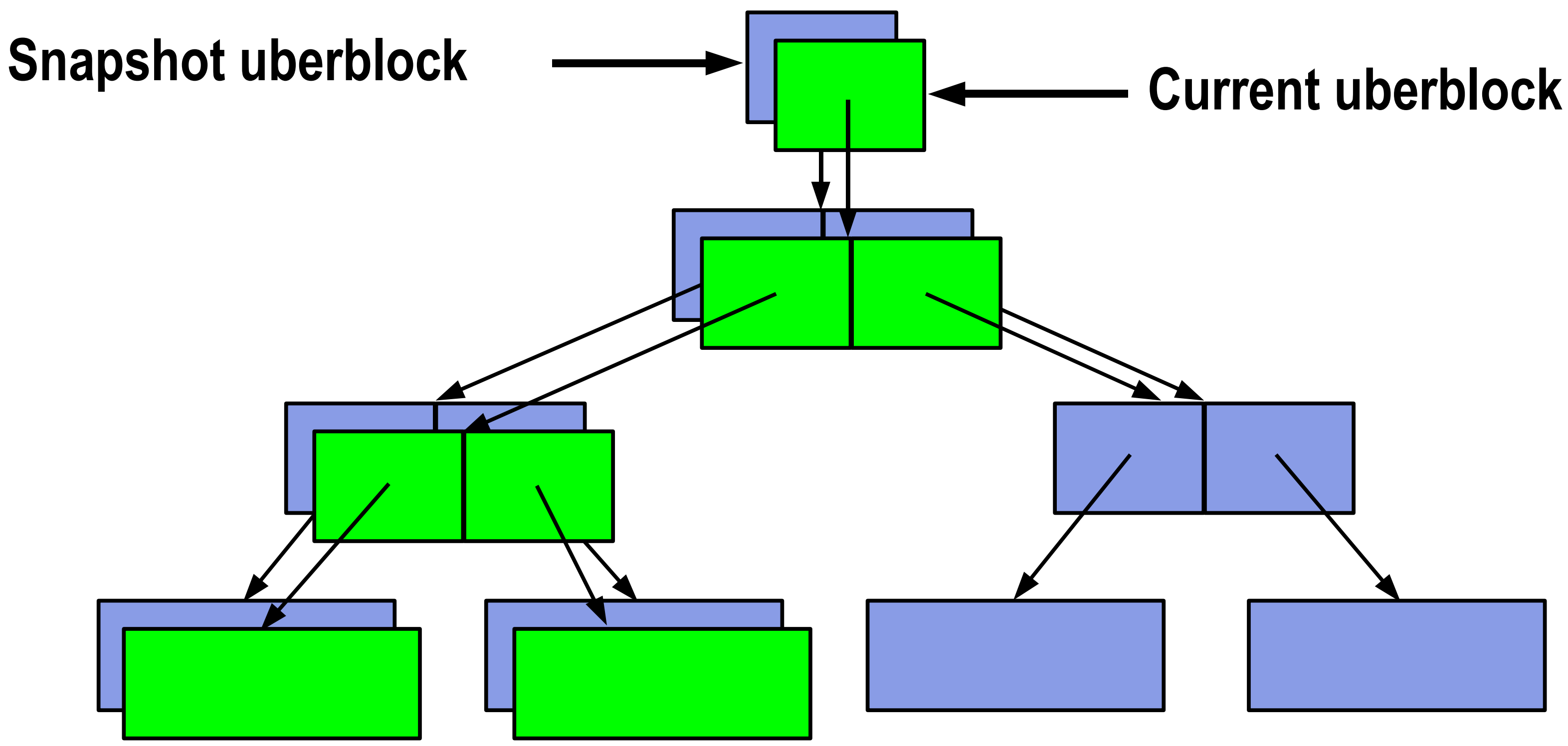


4. Rewrite uberblock (atomic)



Bonus: Constant-Time Snapshots

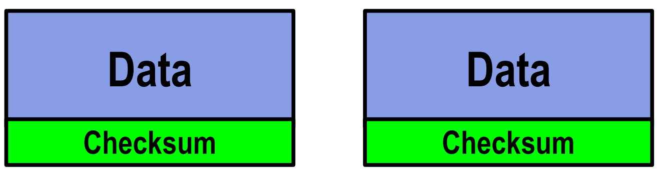
- At end of TX group, don't free COWed blocks
  - Actually cheaper to take a snapshot than not!



End-to-End Checksums

Disk Block Checksums

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

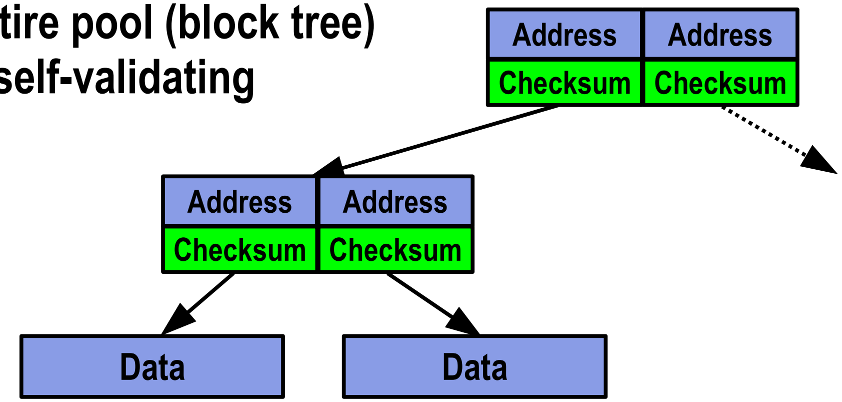


Disk checksum only validates media

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

ZFS Checksum Trees

- Checksum stored in parent block pointer
- Fault isolation between data and checksum
- Entire pool (block tree) is self-validating

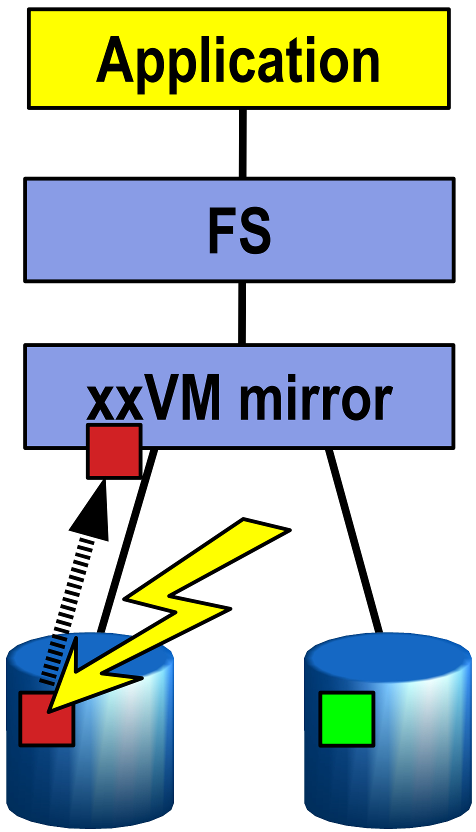


ZFS validates the entire I/O path

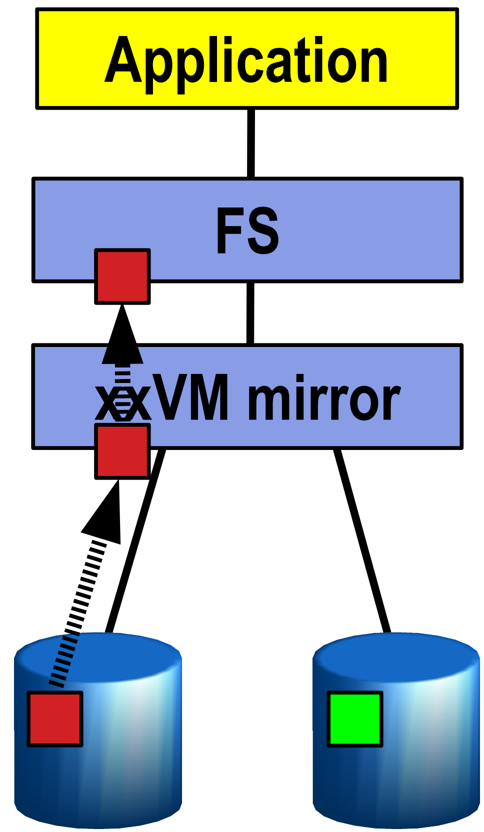
- ✔ Bit rot
- ✔ Phantom writes
- ✔ Misdirected reads and writes
- ✔ DMA parity errors
- ✔ Driver bugs
- ✔ Accidental overwrite

Traditional Mirroring

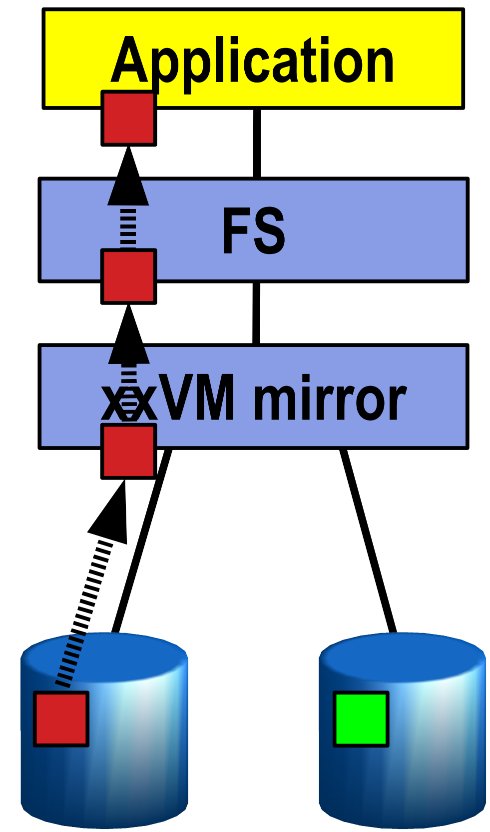
1. Application issues a read. Mirror reads the first disk, which has a corrupt block. It can't tell.



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

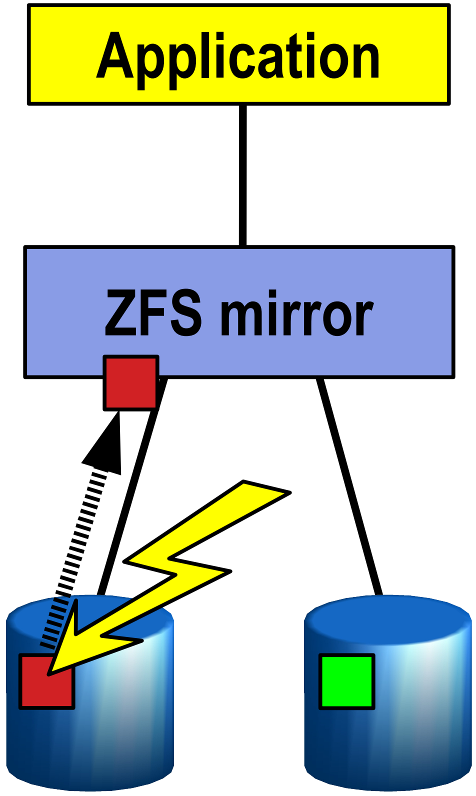


3. Filesystem returns bad data to the application.

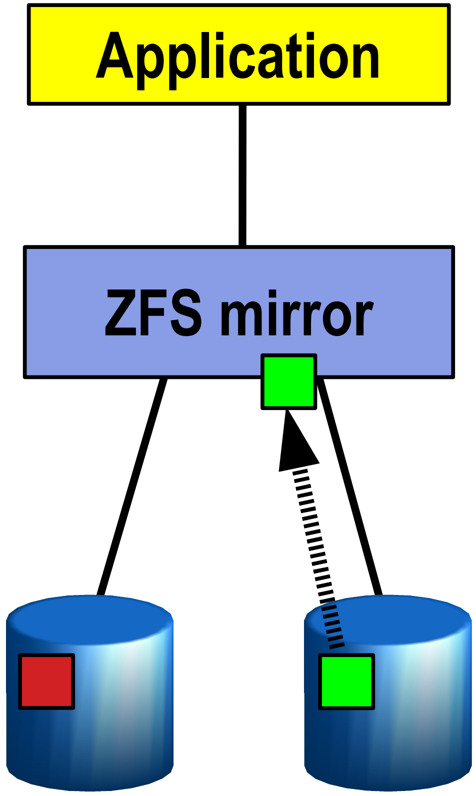


Self-Healing Data in ZFS

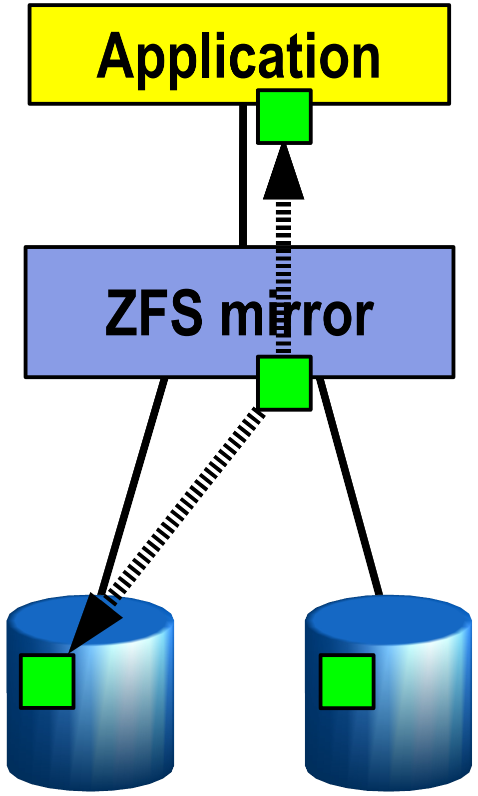
1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



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

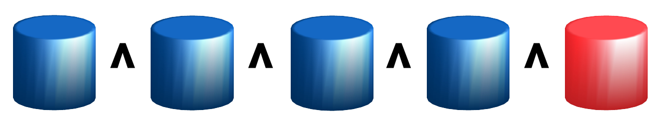


3. ZFS returns good data to the application and repairs the damaged block.



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:*** 

= garbage
    - **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
  - Each logical block is its own stripe
    - 3 sectors (logical) = 3 data blocks + 1 parity block, etc.
    - Integrated stack is key: metadata drives reconstruction
    - Currently single-parity; double-parity version in the works
- All writes are full-stripe writes
  - Eliminates read-modify-write (it's fast)
  - Eliminates the RAID-5 write hole (you don't need NVRAM)
- Detects and corrects silent data corruption
  - Checksum-driven combinatorial reconstruction
- No special hardware – ZFS loves cheap disks

Disk Scrubbing

- Finds latent errors while they're still correctable
  - ECC memory scrubbing for disks
- Verifies the integrity of all data
  - Traverses pool metadata to read every copy of every block
  - Verifies each copy against its 256-bit checksum
  - Self-healing as it goes
- Provides fast and reliable resilvering
  - Traditional resilver: whole-disk copy, no validity check
  - ZFS resilver: live-data copy, everything checksummed
  - All data-repair code uses the same reliable mechanism
    - Mirror resilver, RAID-Z resilver, attach, replace, scrub

Scalability & Performance

ZFS Scalability

- Immense capacity (128-bit)
  - Moore's Law: need 65th bit in 10-15 years
  - Zettabyte = 70-bit (a billion TB)
  - ZFS capacity: 256 quadrillion ZB
  - Exceeds quantum limit of Earth-based storage
    - Seth Lloyd, "Ultimate physical limits to computation." Nature 406, 1047-1054 (2000)
- 100% dynamic metadata
  - No limits on files, directory entries, etc.
  - No wacky knobs (e.g. inodes/cg)
- Concurrent everything
  - Parallel read/write, parallel constant-time directory operations, etc.

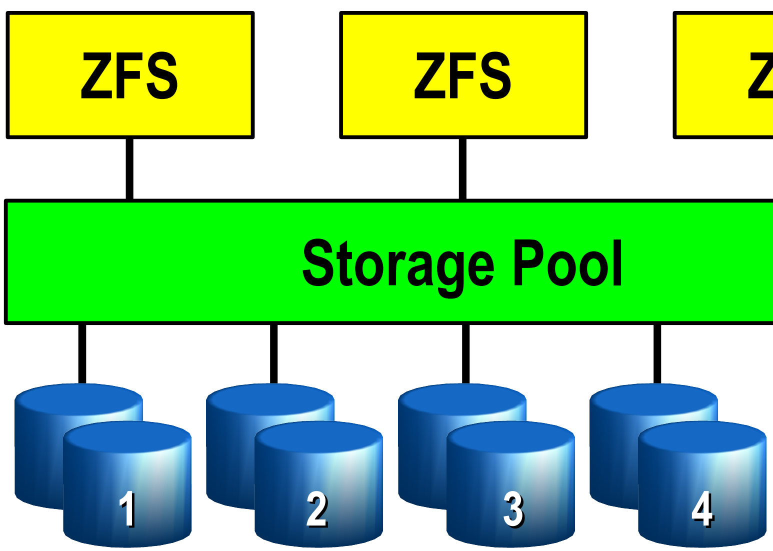
ZFS Performance

- Copy-on-write design
  - Turns random writes into sequential writes
- Dynamic striping across all devices
  - Maximizes throughput
- Multiple block sizes
  - Automatically chosen to match workload
- Pipelined I/O
  - Scoreboarding, priority, deadline scheduling, sorting, aggregation
- Intelligent prefetch

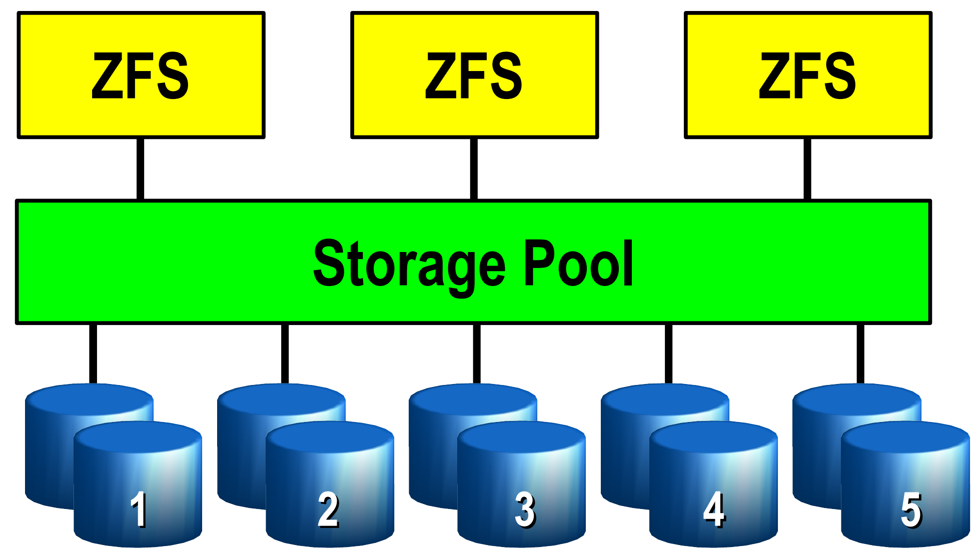
Dynamic Striping

- **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)

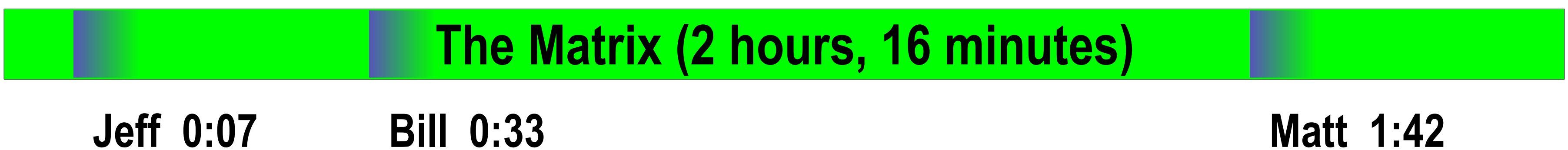


- Writes: striped across all five mirrors
- Reads: wherever the data was written
- No need to migrate existing data
  - Old data striped across 1-4
  - New data striped across 1-5
  - COW gently reallocates old data



Intelligent Prefetch

- **Multiple independent prefetch streams**
  - Crucial for any streaming service provider



- **Automatic length and stride detection**
  - Great for HPC applications
  - ZFS understands the matrix multiply problem
    - Detects any linear access pattern
    - Forward or backward

The Matrix
(10K rows,
10K columns)

ZFS Administration

ZFS Administration

- Pooled storage – no more volumes!
  - All storage is shared – no wasted space, no wasted bandwidth
- Hierarchical filesystems with inherited properties
  - Filesystems become administrative control points
    - Per-dataset policy: snapshots, compression, backups, privileges, etc.
    - Who's using all the space? df(1M) is cheap, du(1) takes forever!
  - Manage logically related filesystems as a group
  - Control compression, checksums, quotas, reservations, and more
  - Mount and share filesystems without /etc/vfstab or /etc/dfs/dfstab
  - Inheritance makes large-scale administration a snap
- Online everything

Creating Pools and Filesystems

- Create a mirrored pool named “tank”

```
# zpool create tank mirror c0t0d0 c1t0d0
```

- 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 c2t0d0 c3t0d0
```

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
```

ZFS Clones

- **Writable copy of a snapshot**
  - Instantaneous creation, unlimited number
  - Ideal for storing many private copies of mostly-shared data
    - Software installations
    - Workspaces
    - Diskless clients
- Create a clone of your OpenSolaris source code

```
# zfs clone tank/solaris@monday tank/ws/lori/fix
```

ZFS Data Migration

- Host-neutral on-disk format
  - Change server from x86 to SPARC, it just works
  - Adaptive endianness: neither platform pays a tax
    - Writes always use native endianness, set bit in block pointer
    - Reads byteswap only if host endianness != block endianness
- ZFS takes care of everything
  - Forget about device paths, config files, /etc/vfstab, etc.
  - ZFS will share/unshare, mount/unmount, etc. as necessary
- Export pool from the old server

```
old# zpool export tank
```

- Physically move disks and import pool to the new server

```
new# zpool import tank
```

ZFS Data Security

- **NFSv4/NT-style ACLs**
  - Allow/deny with inheritance
- **Authentication via cryptographic checksums**
  - User-selectable 256-bit checksum algorithms, including SHA-256
  - Data can't be forged – checksums detect it
  - Uberblock checksum provides digital signature for entire pool
- **Encryption (coming soon)**
  - Protects against spying, SAN snooping, physical device theft
- **Secure deletion (coming soon)**
  - Thoroughly erases freed blocks