ZFS: The Last Word in File Systems

James C. McPherson
SAN Engineering Product Development
Data Management Group
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

What is Wrong 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**
  - Disk labels, partitions, volumes, provisioning, grow/shrink, hand-editing /etc/vfstab...
  - Lots of limits: filesystem/volume size, file size, number of files, files per directory, number of snapshots, ...
  - Not portable between x86 and SPARC
- **Dog slow**
  - Linear-time create, fat locks, fixed block size, naïve prefetch, slow random writes, dirty region logging

The ZFS Objective

- End the suffering
- Design an integrated system from scratch
- Blow away 20 years of obsolete assumptions

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
- **Everything is transactional**
  - > Keeps things always consistent on disk
  - > Removes almost all constraints on I/O order
  - > Allows us to get huge performance wins

Some terminology


Standalone


Cluster


Hot swap


RAID 0


RAID 1


RAID 5

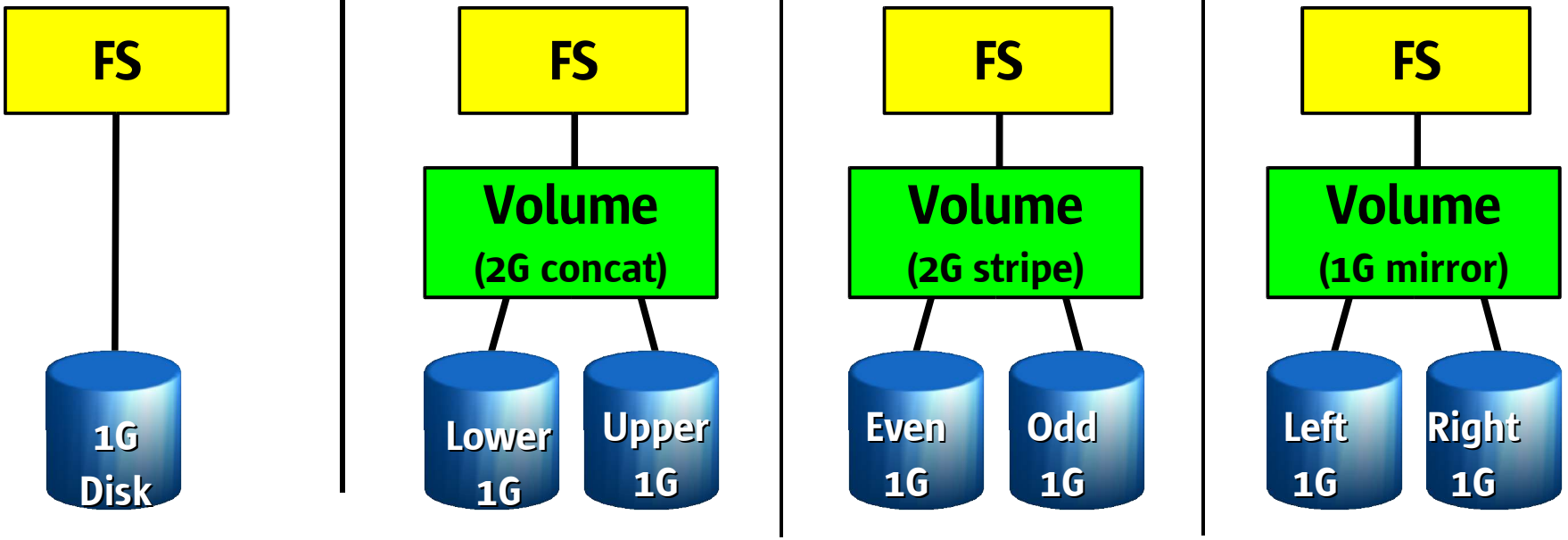

RAID 0+1

Source: http://www.epidauros.be/raid.jpg

Background: Why Volumes Exist

In the beginning, each filesystem managed a single disk.

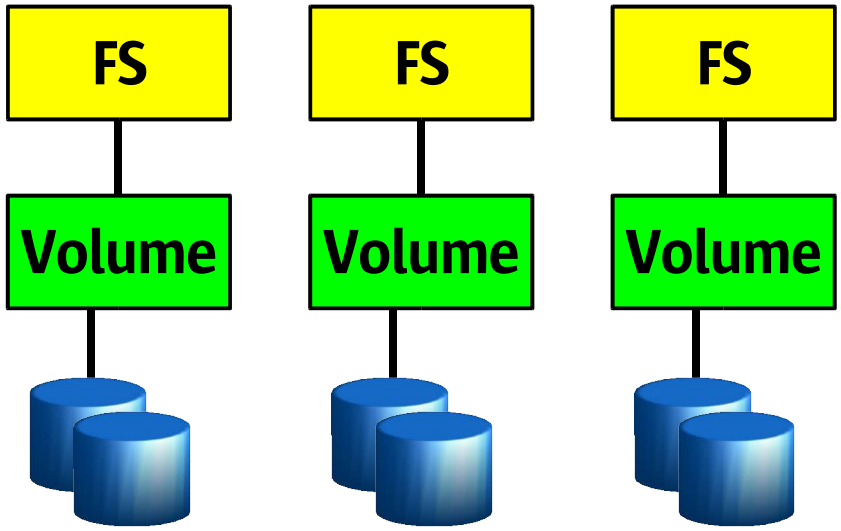
- Customers wanted more space, bandwidth, reliability
  - > Rewrite filesystems to handle many disks: hard
  - > Insert a little shim ("volume") to cobble disks together: easy
- An industry grew up around the FS/volume model
  - > Filesystems, volume managers sold as separate products
  - > Inherent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



FS/Volume Model vs. ZFS (1)

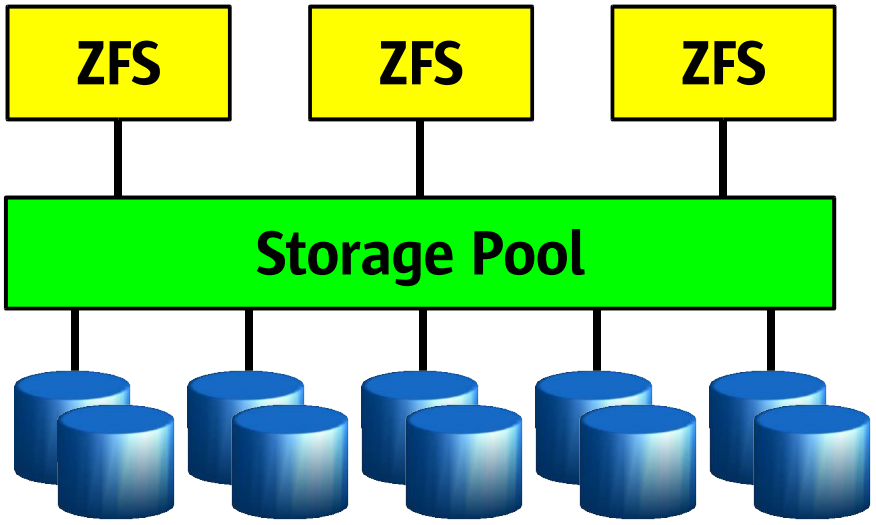
Traditional Volumes

- 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
- Pool allows space to be shared



FS/Volume Model vs. ZFS (2)

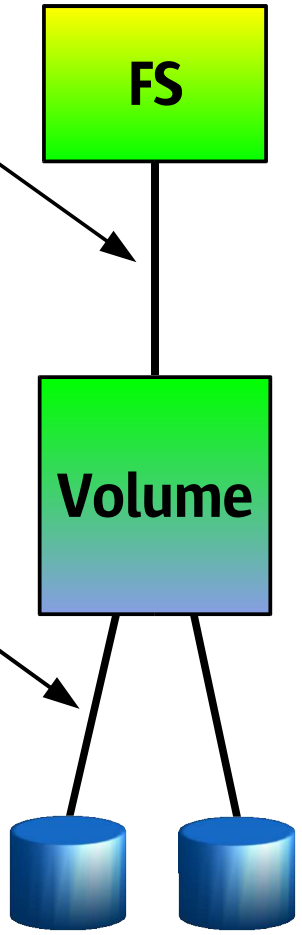
FS/Volume I/O Stack

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

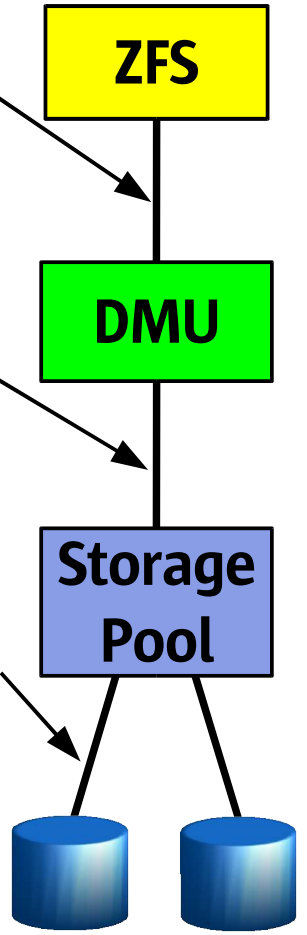
- "Make these 7 changes to these 3 objects"
- All-or-nothing

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**



Traditional Disk Storage Administration

But with ZFS....

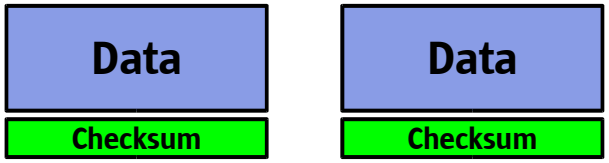
ZFS Data Integrity Model

- 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

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

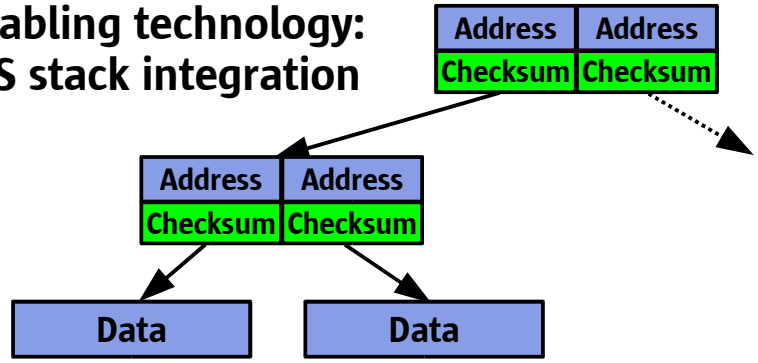


Only validates the media

- ✔ 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
- Enabling technology:
  ZFS stack integration



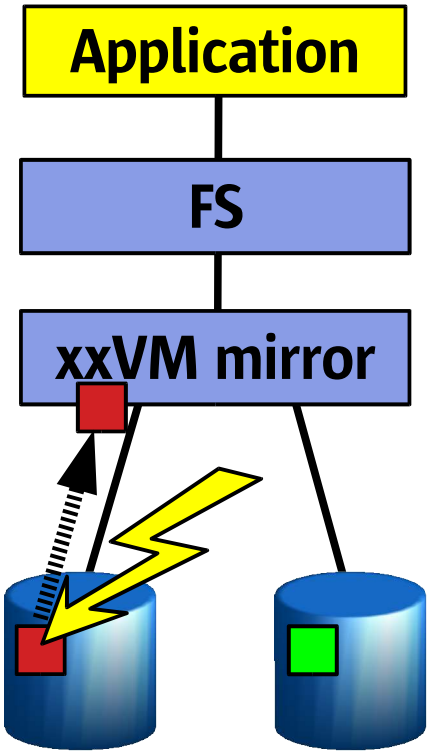
Validates the entire I/O path

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

Traditional Mirroring

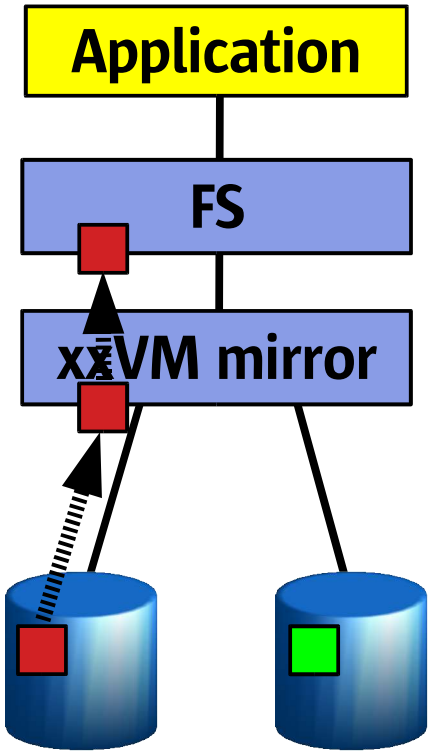
1. Application issues a read. Mirror reads the first disk, which has a corrupt block.

It can't tell.

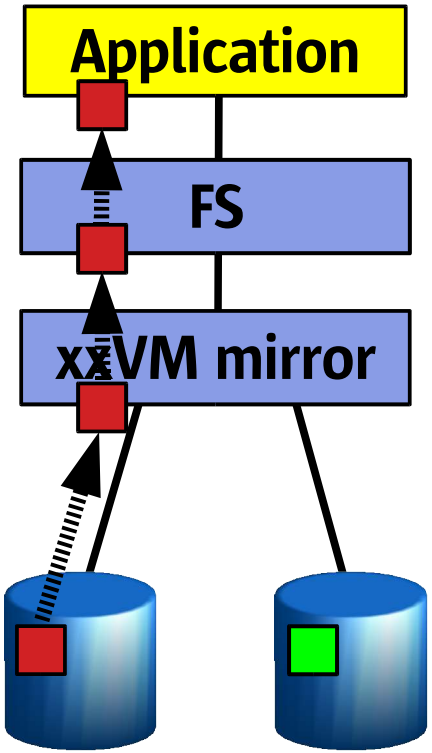


2. Volume manager passes bad block up to filesystem.

If it's a metadata block, the filesystem panics. If not...

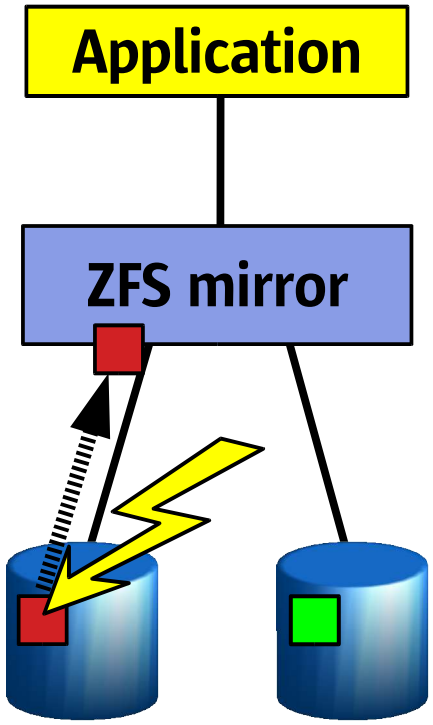


3. Filesystem returns bad data to the application.

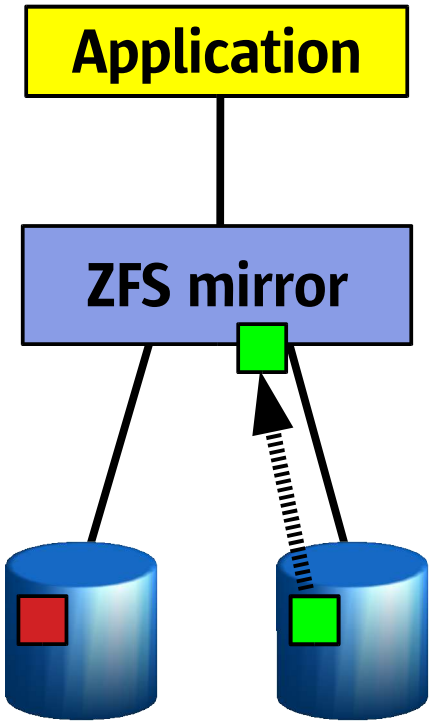


Self-Healing Data in ZFS

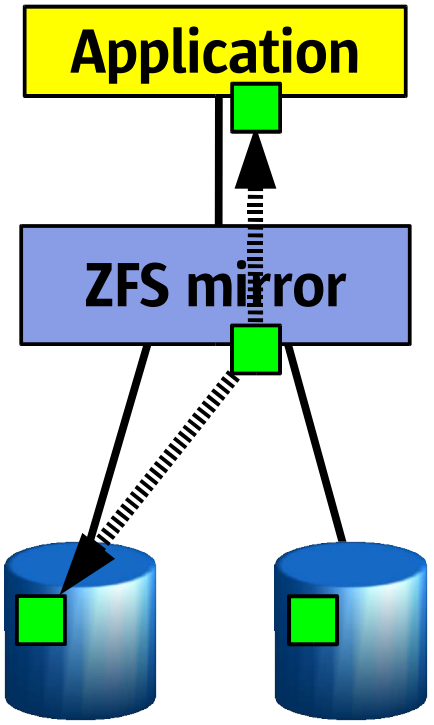
1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



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.



ZFS Data Security

- NFSv4/NT-style ACLs
  - > Allow/deny with inheritance
- Encryption
  - > Protects against spying, SAN snooping, physical device theft
- Authentication via cryptographic checksums
  - > User-selectable 256-bit checksum algorithms include SHA-256
  - > Data can't be forged – checksums detect it
  - > Uberblock checksum provides digital signature for entire pool
- Secure deletion
  - > Freed blocks thoroughly erased
- Real-time remote replication
  - > Allows data to survive disasters of geographic scope

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)
- **Scalable algorithms, 100% dynamic metadata**
  - > No limits on files, directory entries, etc.
  - > No wacky knobs (e.g. inodes/cg)
- **Built-in compression**
  - > Saves space (2-3x typical), reduces I/O – sometimes by so much that compressed filesystems are actually faster!

ZFS Performance

- Key architectural wins:
  - Copy-on-write design makes most disk writes sequential
  - Dynamic striping across all devices maximizes throughput
  - Multiple block sizes, automatically chosen to match workload
  - Explicit I/O priority with deadline scheduling
  - Globally optimal I/O sorting and aggregation
  - Multiple independent prefetch streams with automatic length and stride detection
  - Unlimited, instantaneous read/write snapshots
  - Parallel, constant-time directory operations

ZFS Administation

- Pooled storage – no more volumes!
  - All storage is shared – no wasted space
  - Filesystems are cheap: like directories with mount options
  - Grow and shrink are automatic
  - No more fsck(1M), format(1M), /etc/vfstab, /etc/dfs/dfstab...
- Unlimited, instantaneous snapshots and clones
  - Snapshot: read-only point-in-time copy of the data
  - Clone: writable copy of a snapshot
  - Lets users recover lost data without sysadmin intervention
- Host-neutral on-disk format
  - Change server from x86 to SPARC, it just works
  - Adaptive endianness ensures neither platform pays a tax
- 100% online administration

The Challenge: UFS/SVM vs ZFS

- **Claim:**
  - > ZFS pooled storage simplifies administration dramatically.
  - > So... let's consider a simple example.
- **Our Task:**
  - > Given two disks, create mirrored filesystems for Ann, Bob, and Sue.
  - > Later, add more space.

UFS/SVM Administration Summary

```
# format
... (long interactive session omitted)

# metadb -a -f disk1:slice0 disk2:slice0
# metainit d10 1 1 disk1:slice1
d10: Concat/Stripe is setup
# metainit d11 1 1 disk2:slice1
d11: Concat/Stripe is setup
# metainit d20 -m d10
d20: Mirror is setup
# metattach d20 d11
d20: submirror d11 is attached
# newfs /dev/md/rdsk/d20
newfs: construct a new file system /
dev/md/rdsk/d20: (y/n)? y
... (many pages of 'superblock backup'
output omitted)
# mount /dev/md/dsk/d20 /export/home/ann
# vi /etc/vfstab  ...
```

Repeat as needed for each filesystem

Total of 6 commands to repeat for each filesystem, as well as setting up the partition table, creating metadbs and editing /etc/vfstab.

If we want to create 3 mirrored filesystems that is a total of 20 commands to run....

This is tedious at best!

ZFS Administration

- Create a storage pool named “home”

```
# zpool create home mirror disk1 disk2
```

- Create filesystems “ann”, “bob”, “sue”

```
# zfs create home/ann /export/home/ann
# zfs create home/bob /export/home/bob
# zfs create home/sue /export/home/sue
```

- Add more space to the “home” pool

```
# zpool add home mirror disk3 disk4
```

ZFS Admin: Cool Features

- Turn on compression for Ann's data

```
# zfs compression=on home/ann
```

- Limit Bob to a quota of 10G

```
# zfs quota=10g home/bob
```

- Guarantee Sue a reservation of 20G

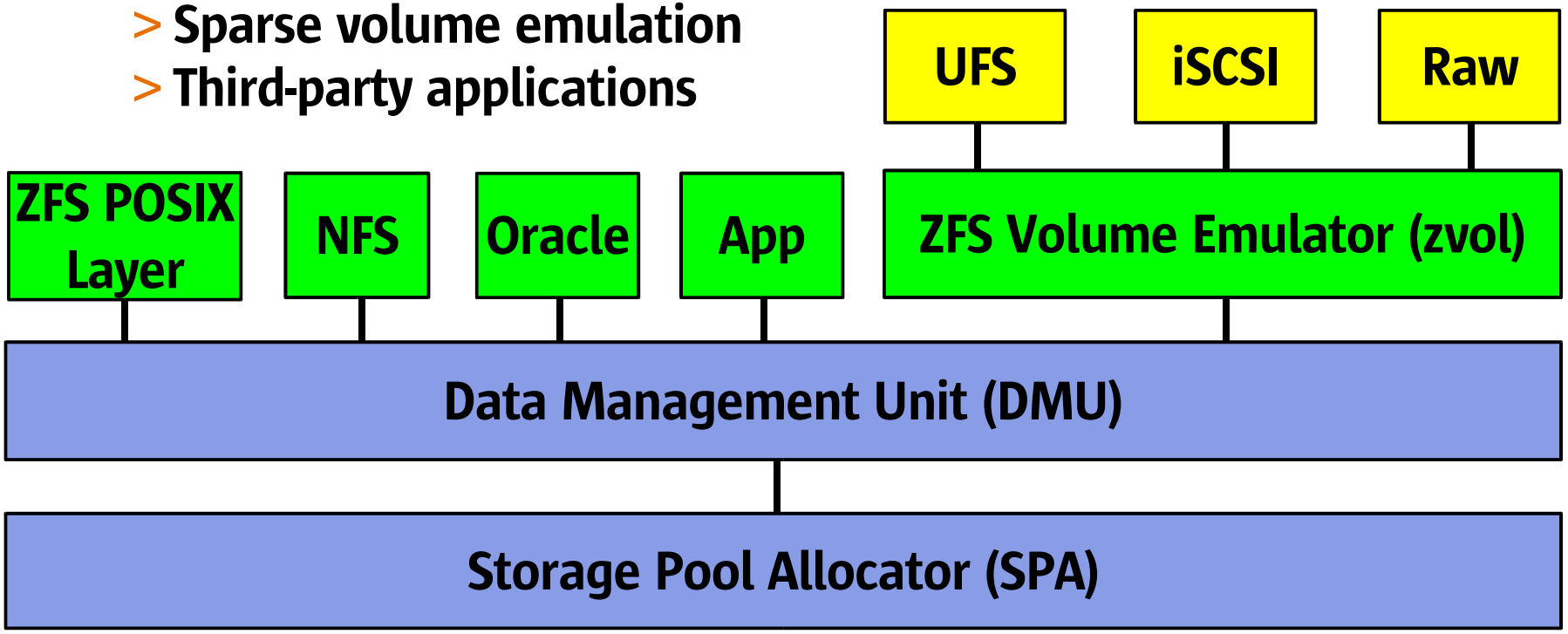
```
# zfs reservation=20g home/sue
```

- Take a snapshot of Ann's filesystem

```
# zfs create home/ann@tuesday
```

Object-Based Storage

- POSIX isn't the only game in town
  - DMU is a general-purpose transactional object store
    - Filesystems
    - Databases
    - Sparse volume emulation
    - Third-party applications



ZFS Test Methodology

- A product is only as good as its test suite
  - ZFS was designed to run in either user or kernel context
  - Nightly "ztest" program does all of the following in parallel:
    - Read, write, create, and delete files and directories
    - Create and destroy entire filesystems and storage pools
    - Turn compression on and off (while filesystem is active)
    - Change checksum algorithm (while filesystem is active)
    - Add and remove devices (while pool is active)
    - Change I/O caching and scheduling policies (while pool is active)
    - Scribble random garbage on one side of live mirror to test self-healing data
    - Force violent crashes to simulate power loss, then verify pool integrity
  - Probably more abuse in 20 seconds than you'd see in a lifetime
  - ZFS has been subjected to over a million forced, violent crashes without ever losing data integrity or leaking a single block

ZFS: Summary

- **Provable data integrity**

  Detects and corrects silent data corruption
- **Immense capacity**

  The world's first 128-bit filesystem
- **Simple administration**

  Concisely expresses the user's intent
- **Smokin' performance**

  Already #1 on several benchmarks, sometimes by multiples

Where can I learn more?

http://blogs.sun.com

Jeff Bonwick, Eric Schrock, Matt Ahrens, Bill Moore

http://www.sun.com/bigadmin

http://www.opensolaris.org

got checksums?

ZFS: The Last Word in File Systems

James C. McPherson
James.McPherson@Sun.COM