FROM ZFS TO OPEN STORAGE

Danilo Poccia
Senior Systems Engineer
Sun Microsystems Southern Europe

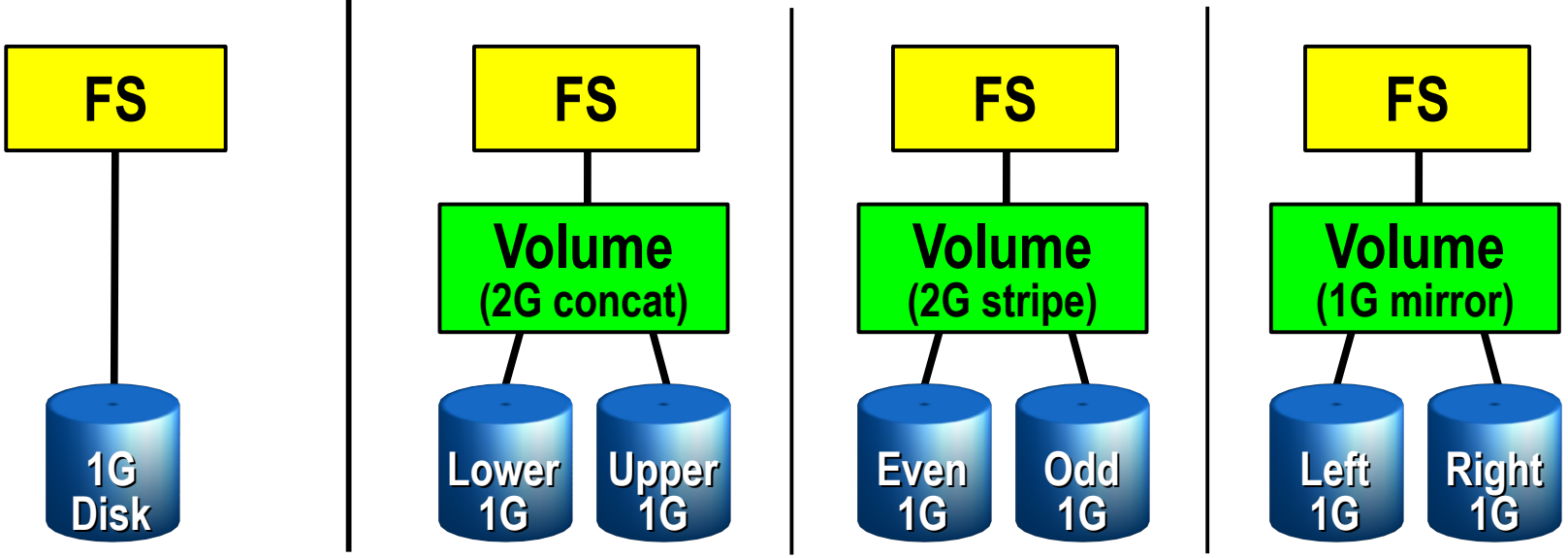
ZFS Design Principles

- 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 huge performance wins

Why Volumes Exist

In the beginning, each filesystem managed a single disk.

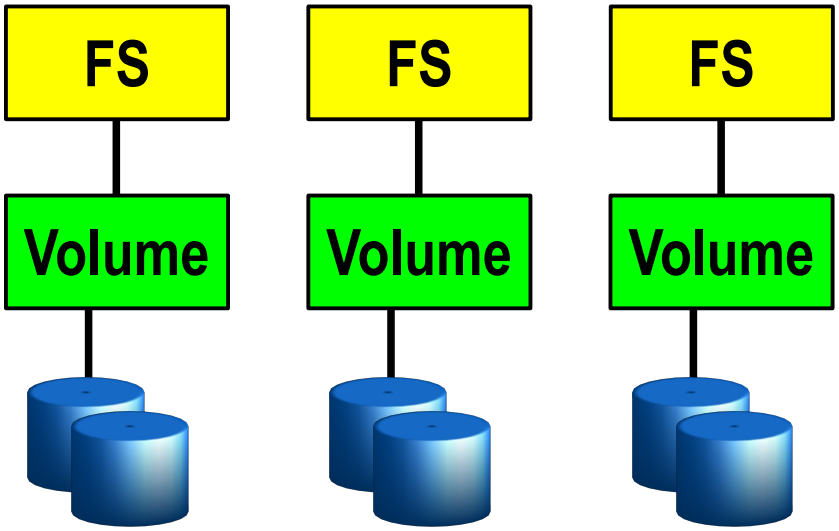
- Customers wanted more space, bandwidth, reliability
  - Hard: redesign filesystems to solve these problems well
  - Easy: insert a little shim ("volume") to cobble disks together
- An industry grew up around the FS/volume model
  - Filesystems, volume managers sold as separate products
  - Inherent problems in FS/volume <u>interface</u> can't be fixed



FS/Volume Model vs. ZFS

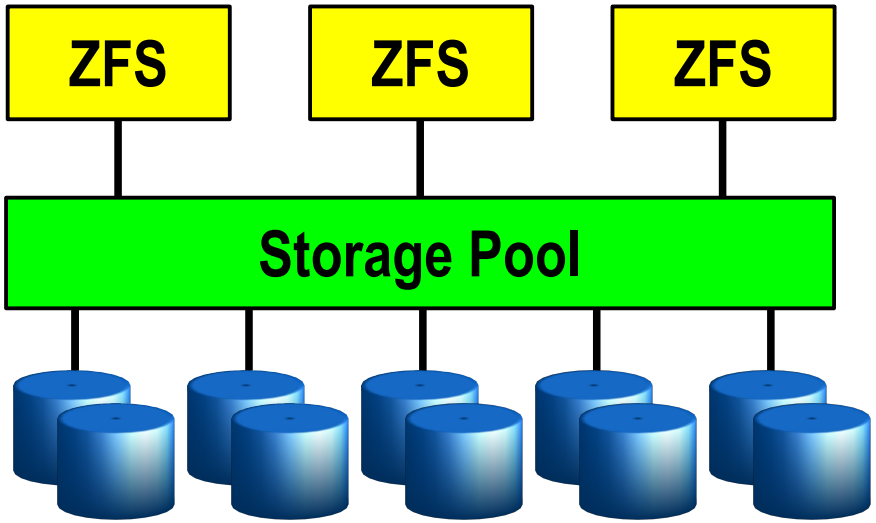
Traditional Volumes

- 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

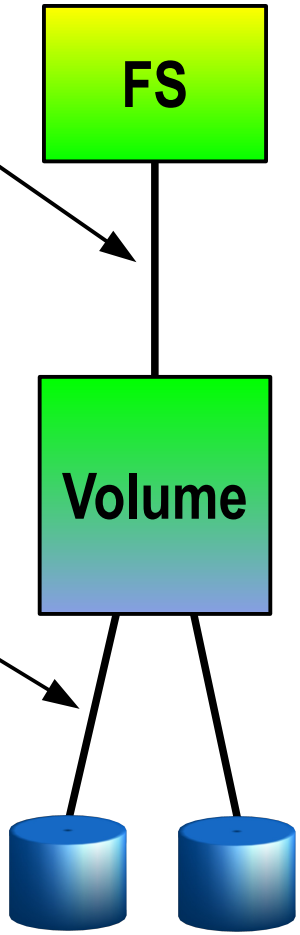
FS/Volume I/O Stack

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

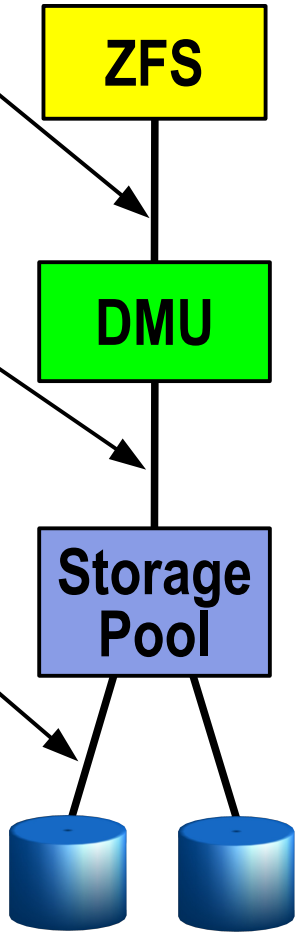
- **"Make these 7 changes to these 3 objects"**
- **All-or-nothing**

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

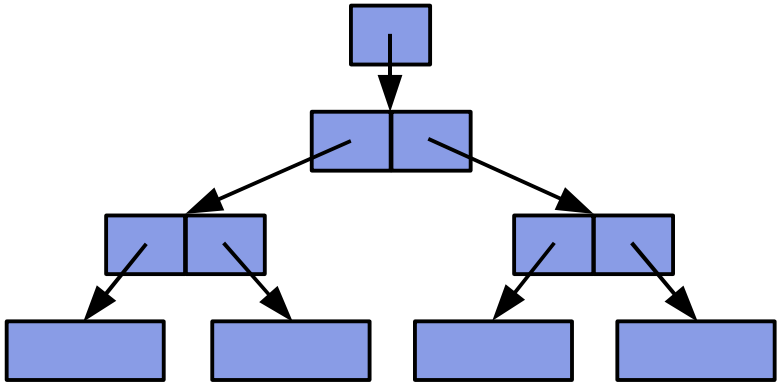


ZFS Data Integrity Model

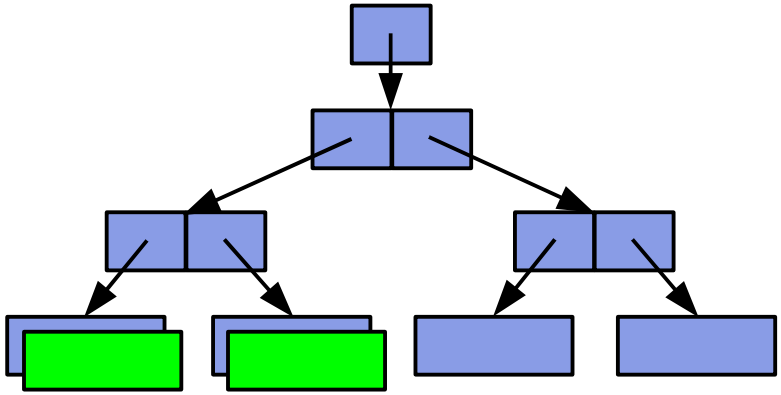
- Everything is copy-on-write
  - Never overwrite live data
  - On-disk state always valid – no “windows of vulnerability”
  - No need for fsck(1M)
- Everything is transactional
  - Related changes succeed or fail as a whole
  - No need for journaling
- Everything is checksummed
  - No silent data corruption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

Copy-On-Write Transactions

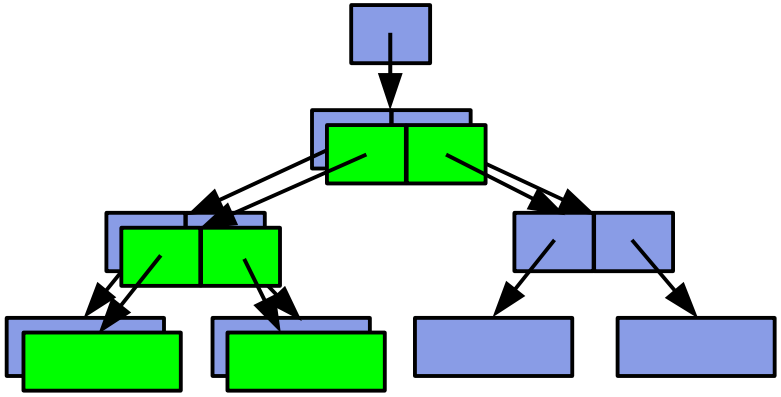
1. Initial block tree



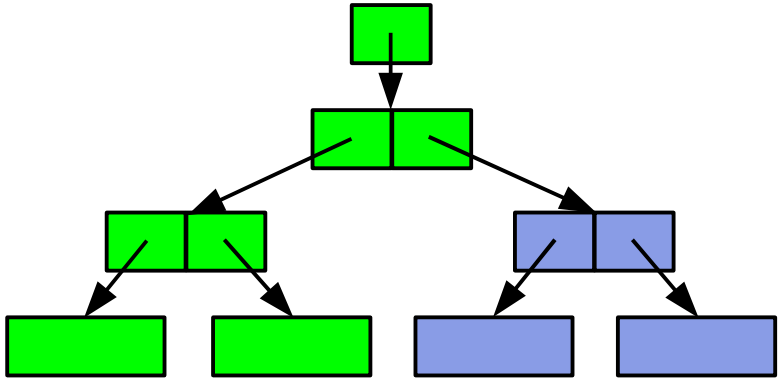
2. COW some blocks



3. COW indirect blocks

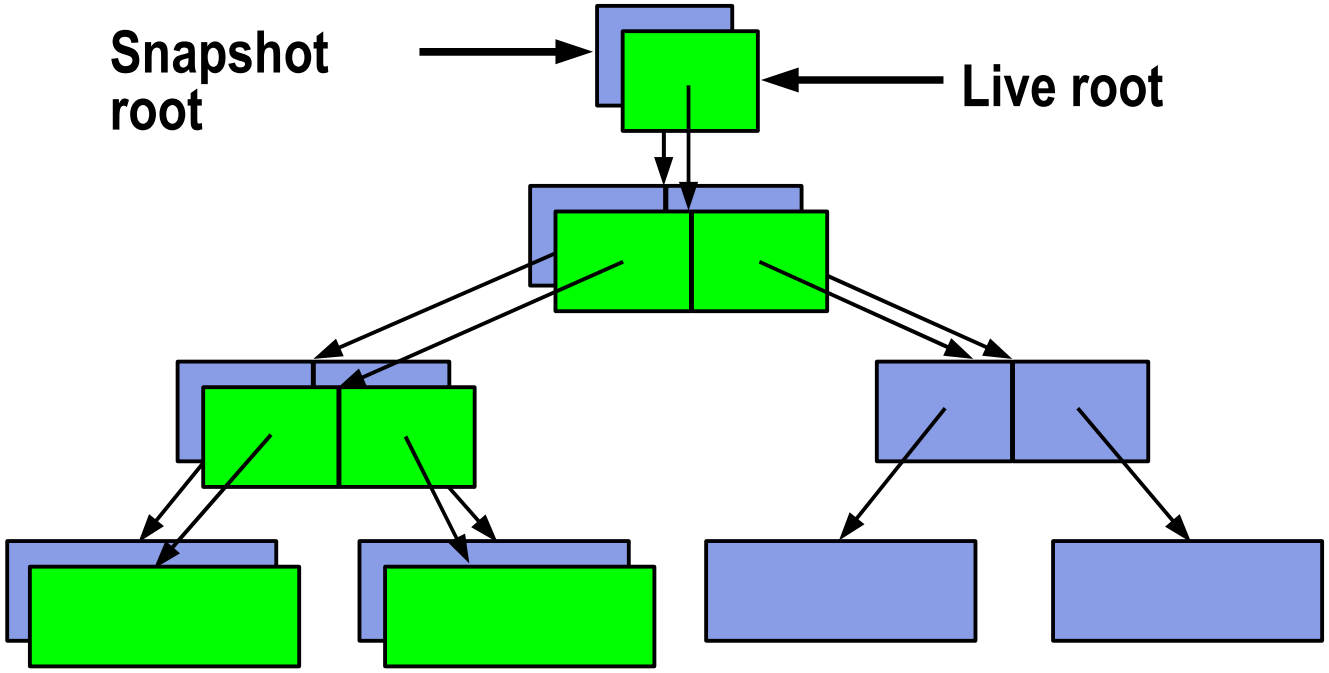


4. Rewrite uberblock (atomic)



Bonus: Constant-Time Snapshots

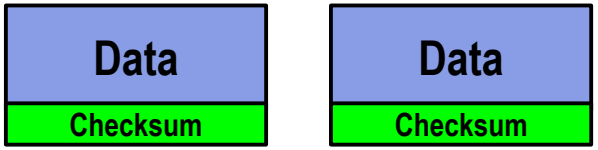
- At end of TX group, don't free COWed blocks
  - Actually cheaper to take a snapshot than not!



End-to-End Data Integrity

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

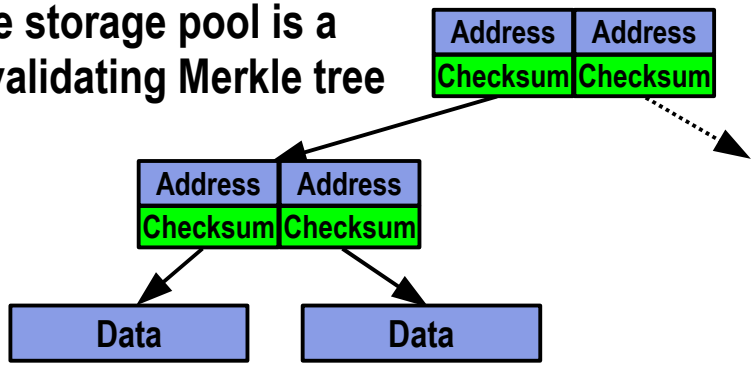


Disk checksum only validates media

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

ZFS Data Authentication

- Checksum stored in parent block pointer
- Fault isolation between data and checksum
- Entire storage pool is a 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

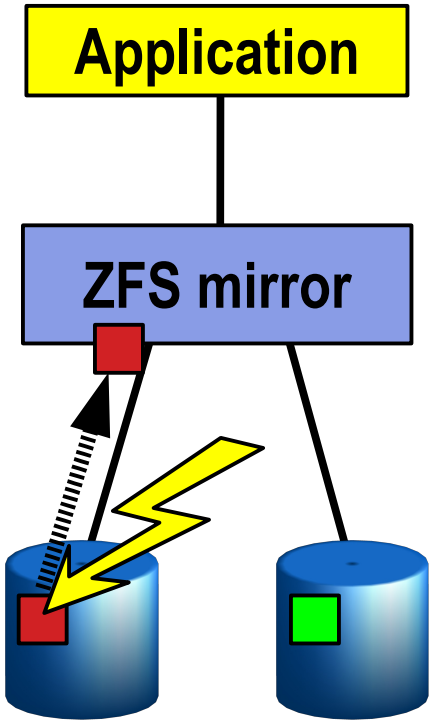
RAID-Z Protection

RAID-5 and More

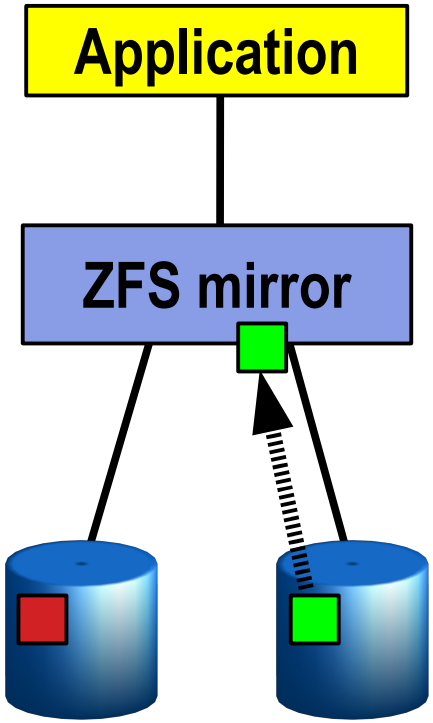
- ZFS provides better than RAID-5/6 availability
- Striping uses dynamic widths
  - Each logical block is its own stripe
- All writes are full-stripe writes
  - Eliminates read-modify-write (So it's fast!)
- Eliminates RAID-5 “write hole”
  - No need for NVRAM
- Single, Double and Triple Parity RAID
  - RAID-Z(1), RAID-Z2, RAID-Z3

Self-Healing Data

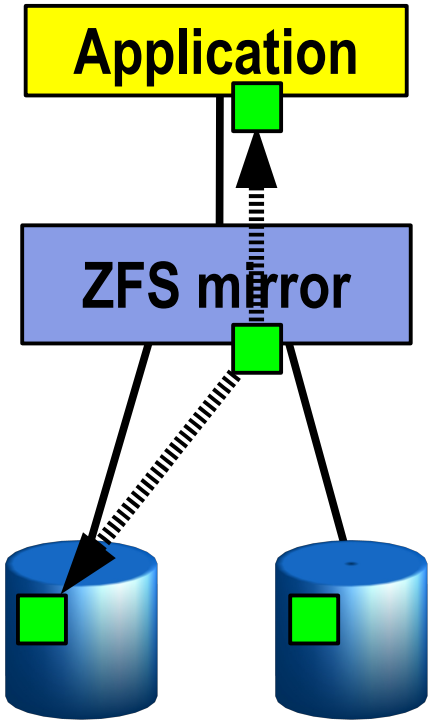
1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



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.

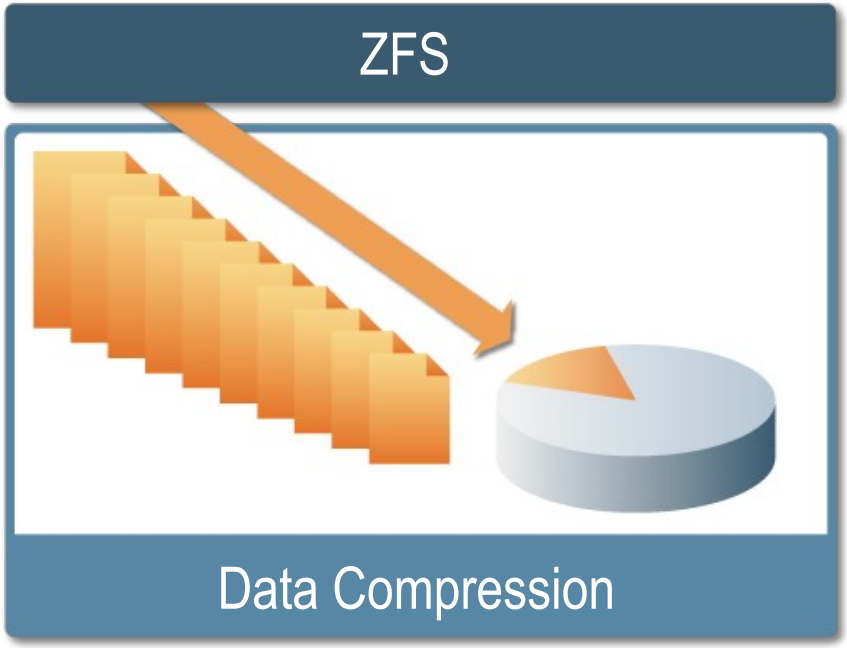


ZFS Administration

- 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? du(1) takes forever, but df(1M) is instant!
  - 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

Data Compression

- Reduces the amount of disk space used
- Reduces the amount of data transferred to disk – increasing data throughput
  - > LZW
  - > GZIP (1-9)



Copy-on-Write Design

Multiple Block Sizes

Pipelined I/O

Dynamic Striping

Intelligent Pre-Fetch

Architected for Speed

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 Send/Receive (Backup/Restore)

- Powered by snapshots
  - Full backup: any snapshot
  - Incremental backup: any snapshot delta
    - Very fast – cost proportional to data changed
- So efficient it can drive remote replication
- Generate a full backup

```
# zfs send tank/fs@A >/backup/A
```

- Generate an incremental backup

```
# zfs send -i tank/fs@A tank/fs@B > /bck/B-A
```

- Remote replication: send incremental once per minute

```
# zfs send -i tank/fs@11:31 tank/fs@11:32 |
  ssh host zfs receive -d /tank/fs
```

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
```

Cost Effective Performance

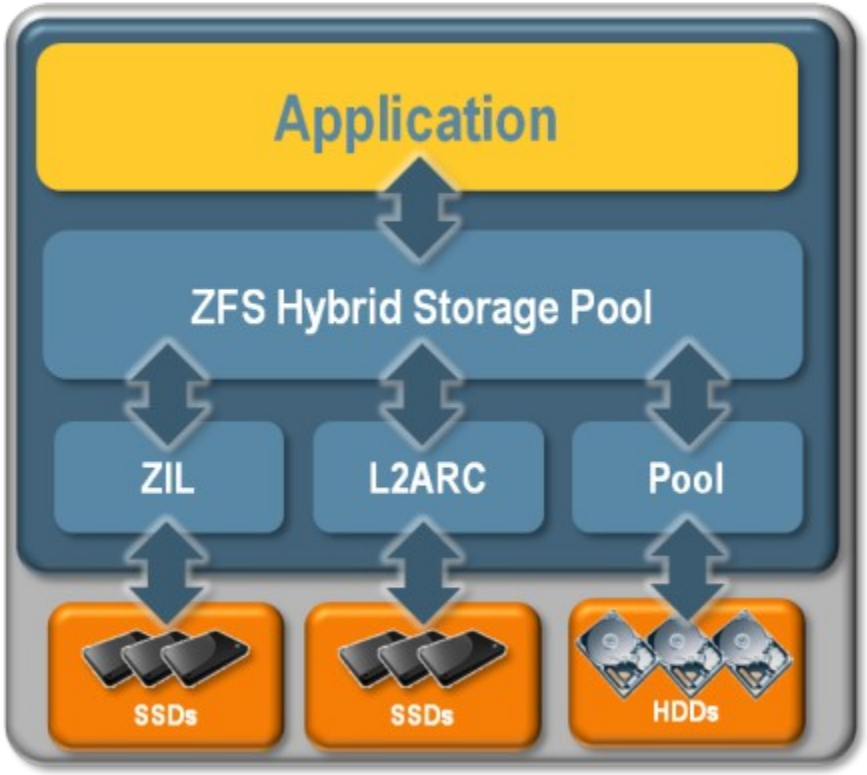
SSDs are 120X more cost effective

- Enterprise HDD
  - 180 Write IOPS
  - 320 Read IOPS
  - 300 GB
- $ per IOPS: 2.43

- Enterprise SSD
  - 7,000 Write IOPS
  - 50,000 Read IOPS
  - 32GB
- **$ per IOPS: 0.02**

ZFS Turbo Charges Applications

Hybrid Storage Pools

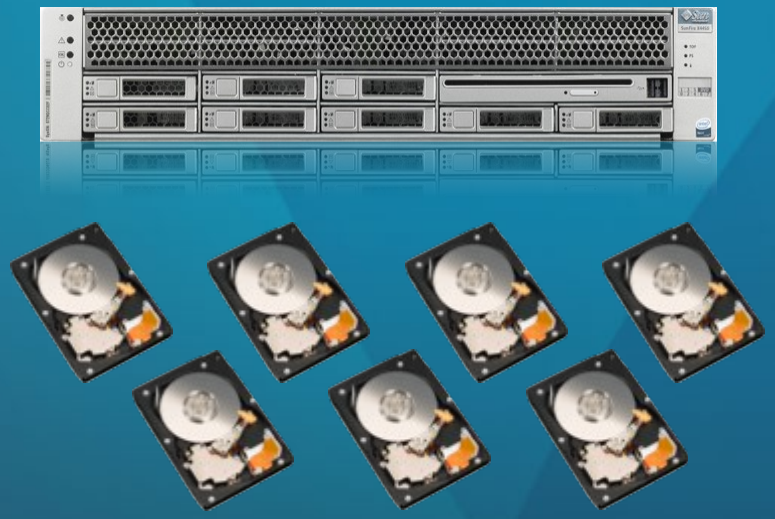


- ZFS automatically:
  - > Determines data access patterns and stores frequently accessed data in a read cache called L2ARC
  - > Bundles IO into sequential staged writes for more efficient use of low cost mechanical disks
  - > Very fast synchronous writes occur to a very fast SSD pool (ZIL) accelerating applications, such as databases & NFS

ZFS Hybrid Storage Pool

Sun X4250 Storage Server Example

Configuration A



(7) 146GB 10,000 RPM SAS Drives

Configuration B

- 4 Xeon 7350 Processors
- 32GB FB DDR2 ECC DRAM
- OpenSolaris with ZFS

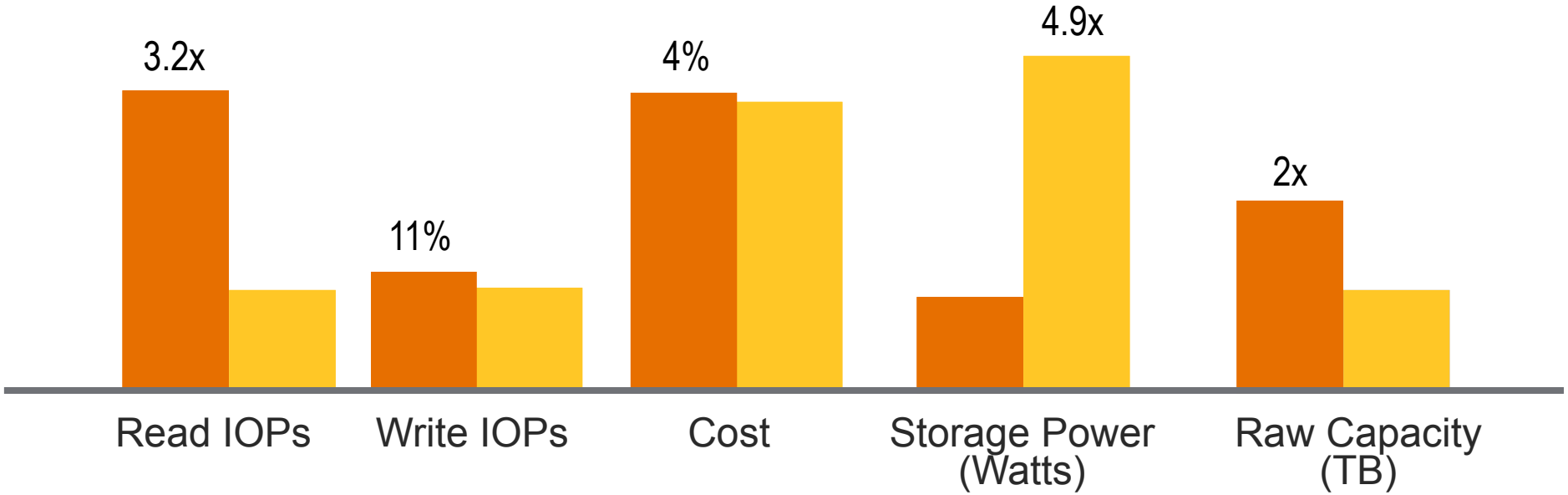
(1) 80G SSD Cache Device

(1) 32G SSD ZIL Device

(5) 400GB 4200 RPM SATA Drives

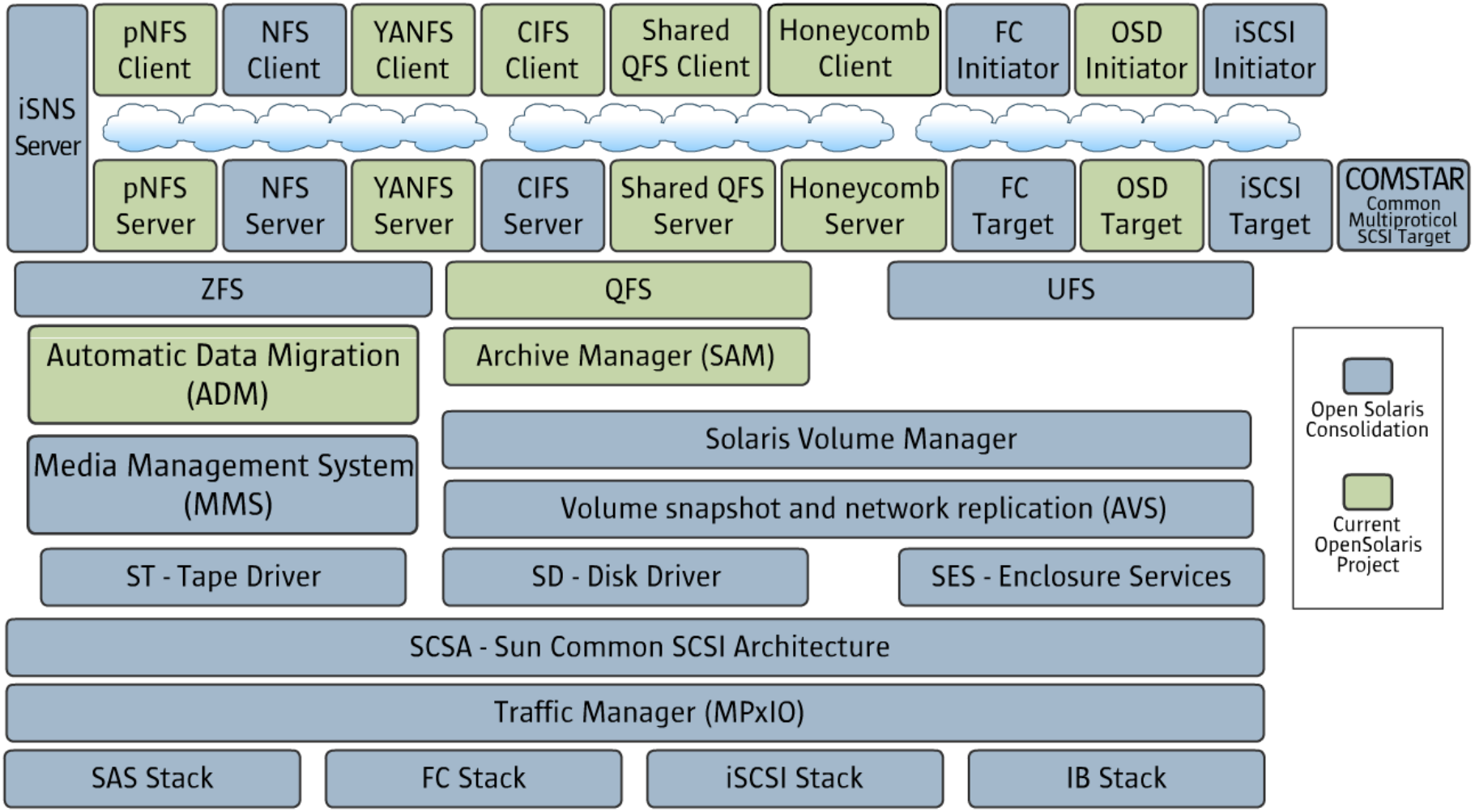
ZFS Hybrid Pool Example

Based on Actual Benchmark Results



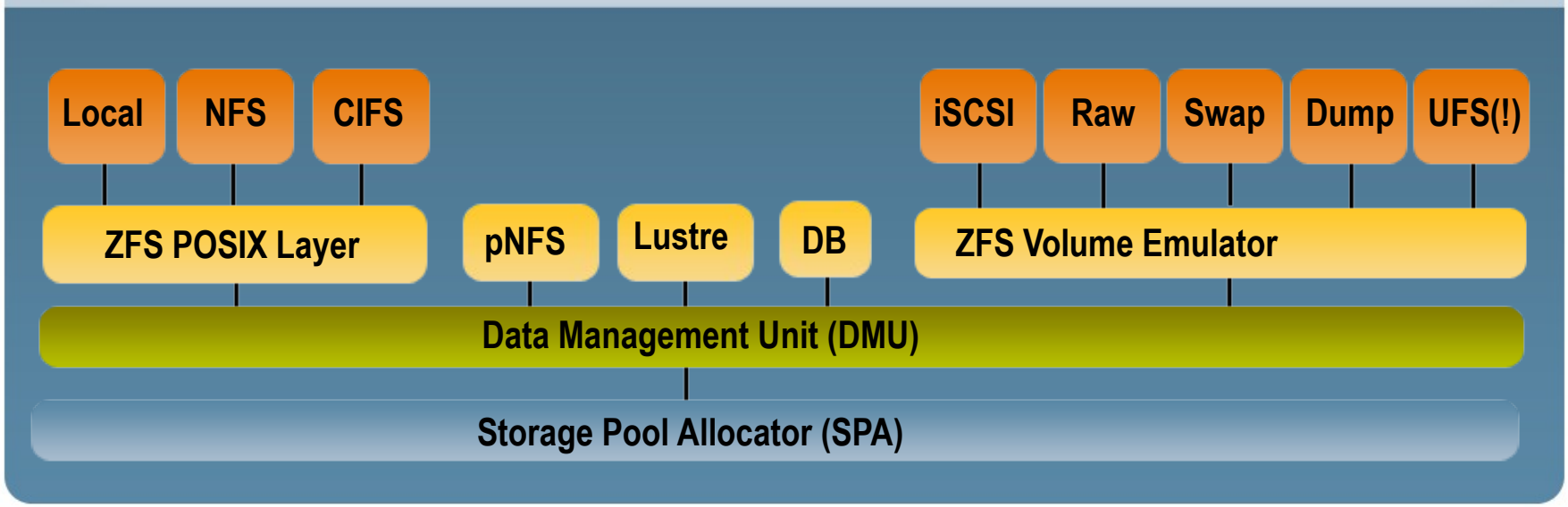
- Hybrid Storage Pool (DRAM + Read SSD + Write SSD + 5x 4200 RPM SATA)
- Traditional Storage Pool (DRAM + 7x 10K RPM 2.5”)

OpenSolaris Storage Projects



Universal Storage

- DMU is a general-purpose transactional object store
- ZFS dataset = up to 2^{48} objects, each up to 2^{64} bytes
- Key features common to all datasets
- Snapshots, compression, encryption, end-to-end data integrity
- Any flavor you want: file, block, object, network



Sun Storage 7000 Unified Storage Systems

Bringing Simplicity to Enterprise Storage

- **Sun Storage 7110**
  - 8 GB RAM
  - Up to 14x300GB SAS 10K-rpm Drives
  - Up to 4 TB
  - 2RU package

- **Sun Storage 7210**
  - 32 GB and 64 GB options
  - Up to 142 TB
  - Hybrid Storage Pool I/O Acceleration
  - Write Flash/SSD options for higher performance

- **Sun Storage 7310 Cluster***
  - 16 GB (up to 64 GB) per controller
  - Up to 96 TB
  - Hybrid Storage Pool I/O Acceleration
  - Read/Write Flash/SSD options for higher performance

- **Sun Storage 7410 Cluster***
  - 64 GB, 128 GB, and 256 GB options per controller
  - Up to 288 TB
  - Includes cluster software
  - Hybrid Storage Pool I/O Acceleration
  - Read/Write Flash/SSD options for higher performance

Provides support for unified file and block data protocols, including a rich set of data services

*Cluster Optional

Full Complement of Storage Software

Included with the System at No Additional Cost

Data Protocols

- NFS v2, v3 and v4
- CIFS
- ISCSI
- HTTP
- WebDAV
- FTP/SFTP/FTPS
- NDMP v3/v4
- NFS over RDMA (Infiniband)

Data Services

- Flash Hybrid Storage Pool
- Triple-Parity RAID
- Triple Mirroring
- RAID-Z (5)
- RAID-Z DP (6)
- Mirroring
- Striping
- Active-active Clustering
- Remote Replication
- Antivirus via ICAP Protocol
- Snapshots
- Clones
- Compression
- Thin Provisioning
- End-to-End Data Integrity
- Multi-Path I/O
- Fault Management
- Microsoft VSS/Shared Folders
- Online Data Migration
- NFS Nested Mountpoints

Management

- DTrace Analytics
- Dashboard
- Role-Based Access Control
- NIS LDAP & AD Alerts
- Phone Home
- SNMP
- Scripting
- Upgrade
- Hardware View
- Advanced Networking
- Workflow Automation
- User-level Quotas
- Support for Microsoft Management Console
- System Configuration Backup/ Restore
- System Configuration Import/ Export

Providing Unprecedented Storage Analytics

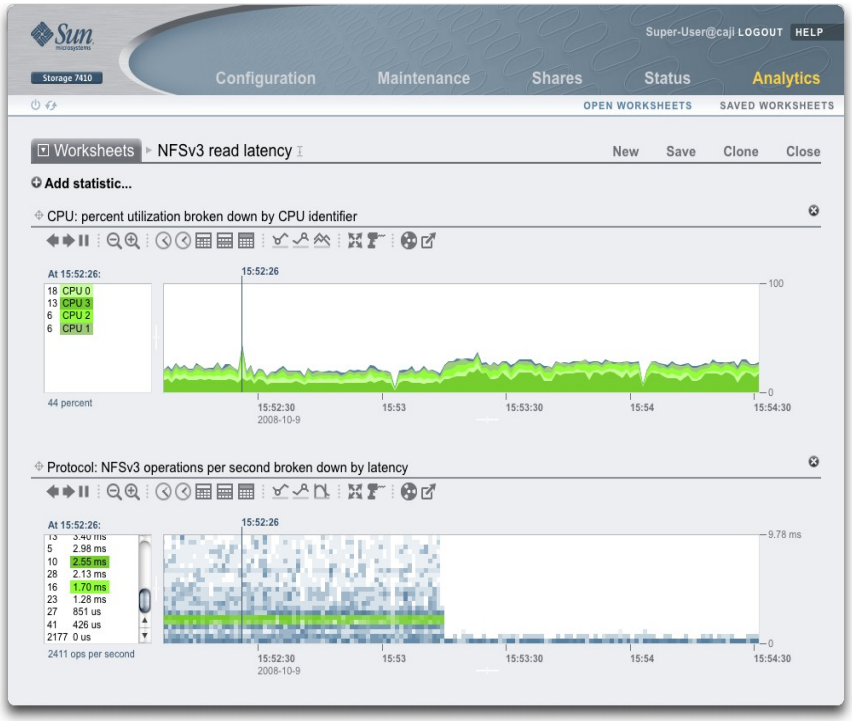
- Automatic real-time visualization of application and storage related workloads
- Simple yet sophisticated instrumentation provides real-time comprehensive analysis
- Supports multiple simultaneous application and workload analysis in real- time
- Analysis can be saved, exported and replayed for further analysis.
- Built on DTrace instrumentation

>NFSv3, NFSv4, CIFS, iSCSI, NFS over RDMA (Infiniband)

>ZFS and the Solaris I/O path

>CPU and Memory Utilization

>Networking (TCP, UDP, IP)



Instant Visibility into Key Questions

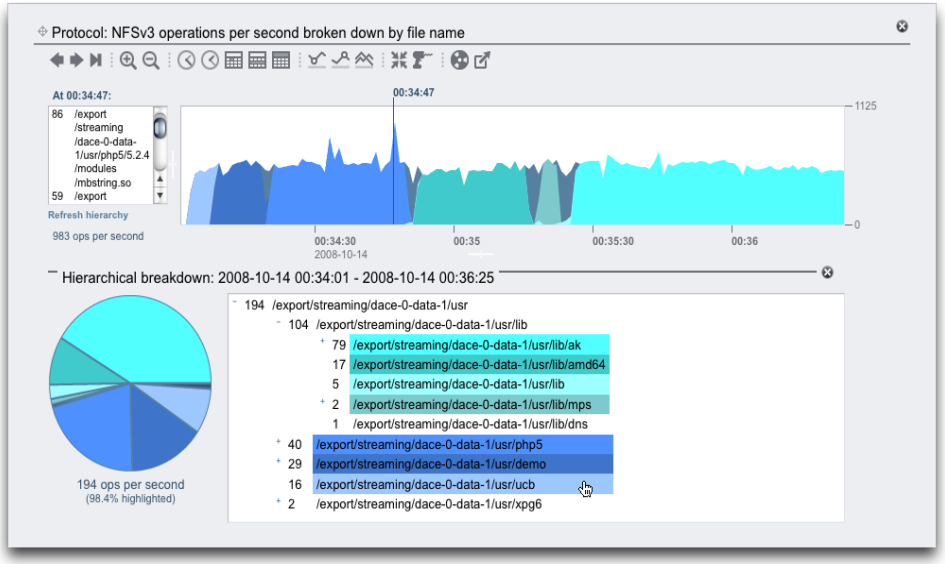
"What is CPU and Memory Utilization?"

"How much storage is being utilized?"

"How is disk performing? How many Ops/Sec?"

"What Services are active?"

"Which applications/users are causing performance issues?"



FROM ZFS
TO OPEN STORAGE

Danilo Poccia
danilo.poccia@sun.com
blogs.sun.com/danilop