Integrating ZFS into your Application Environment

Brian Wong

www.opensolaris.org/os/community/zfs

Agenda

- ZFS Architecture
- How to Deploy ZFS
- So you want to help develop ZFS?
- Links/Resources

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
- Provable end-to-end data integrity
  - Detects and corrects silent data corruption
  - Historically considered “too expensive”
- Transactional design
  - Always consistent on disk
  - Removes most constraints on I/O order – huge performance wins
- Simple administration
  - Concisely express your intent

FS/Volume Model vs. ZFS

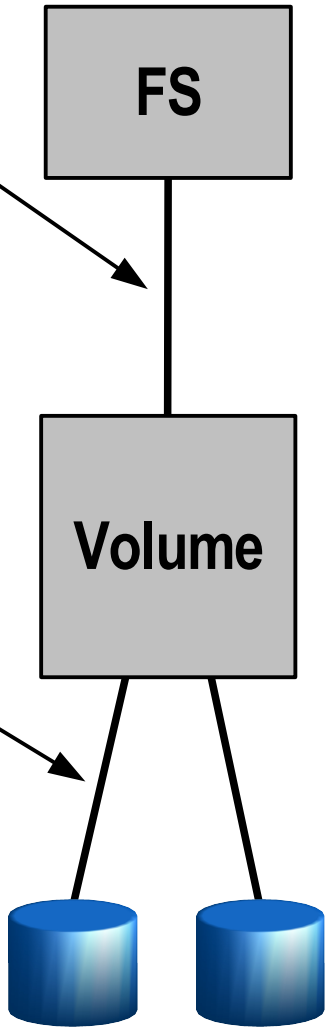
FS/Volume I/O Stack

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

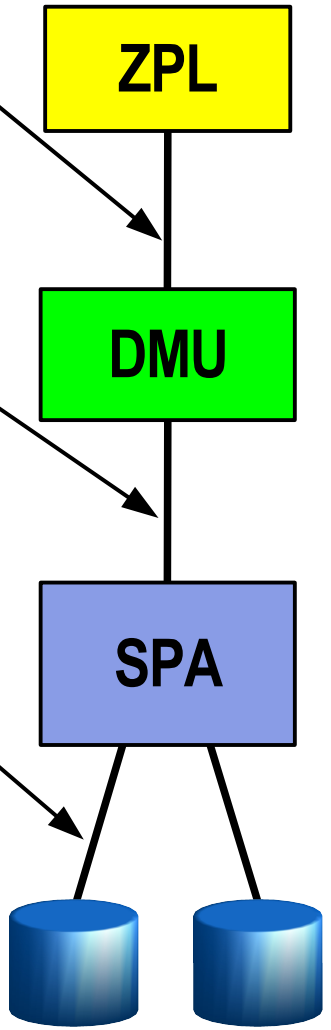
- "Make these 7 changes to these 3 objects"
- All-or-nothing

Transaction Group Commit

- Again, all-or-nothing
- Always consistent on disk
- No journal – not needed

Transaction Group Batch I/O

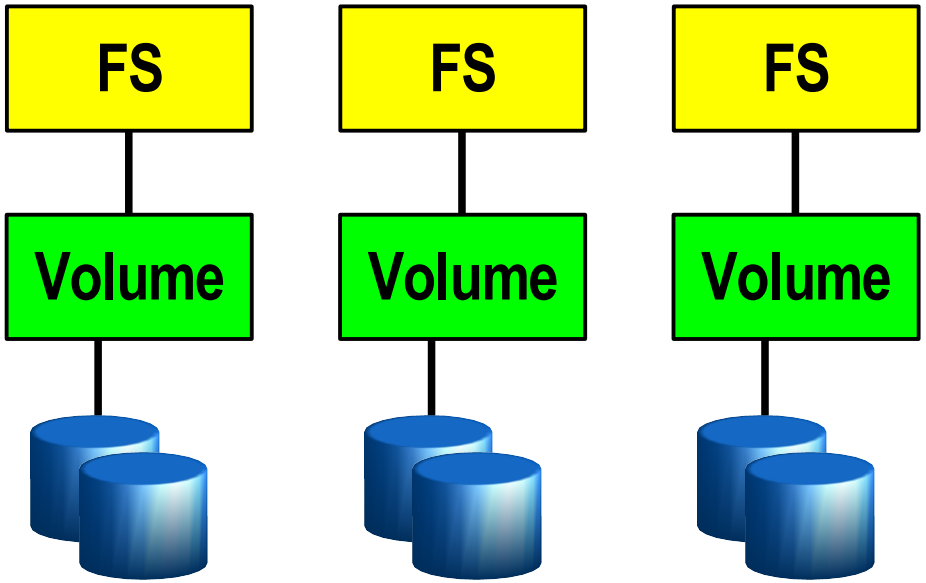
- Schedule, aggregate, and issue I/O at will
- No resync if power lost
- Runs at platter speed



FS/Volume Model vs. ZFS

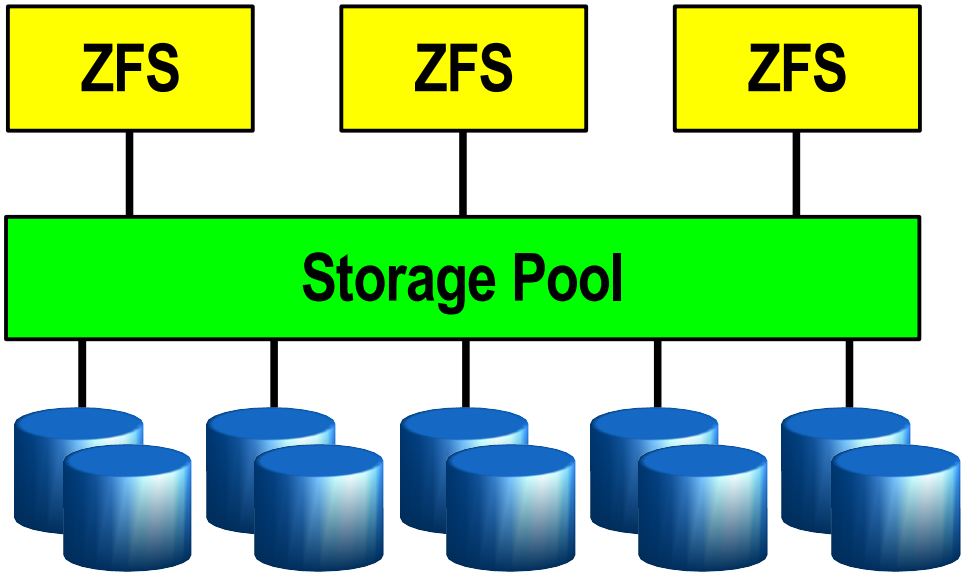
Traditional Volumes

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



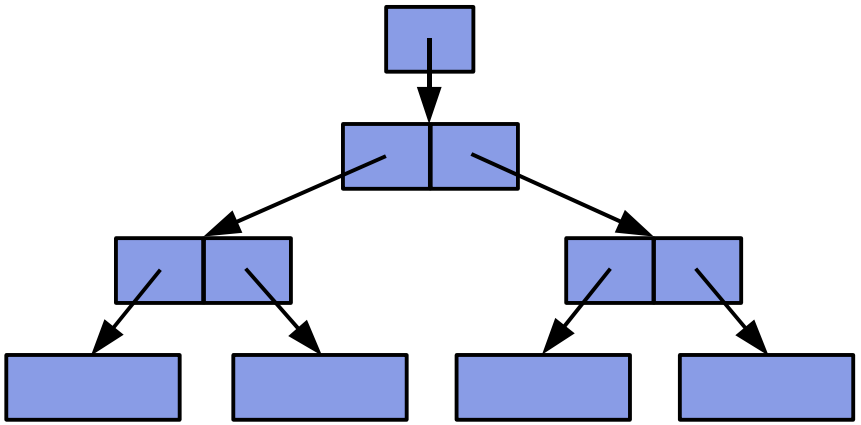
ZFS Pooled Storage

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

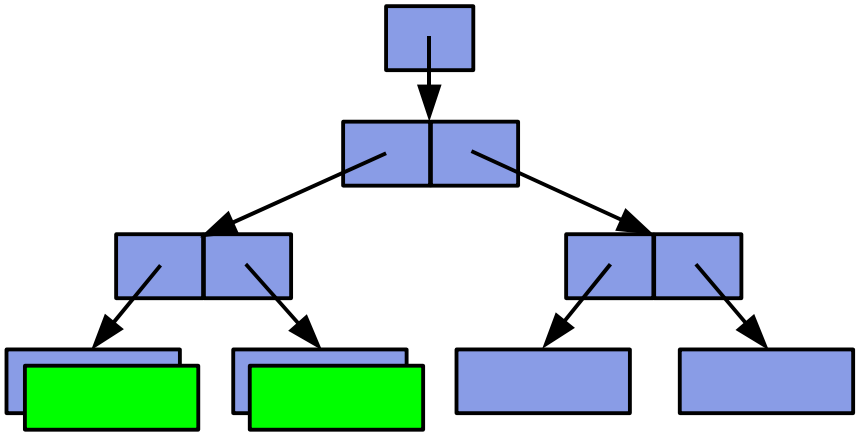


Copy-On-Write Transactions

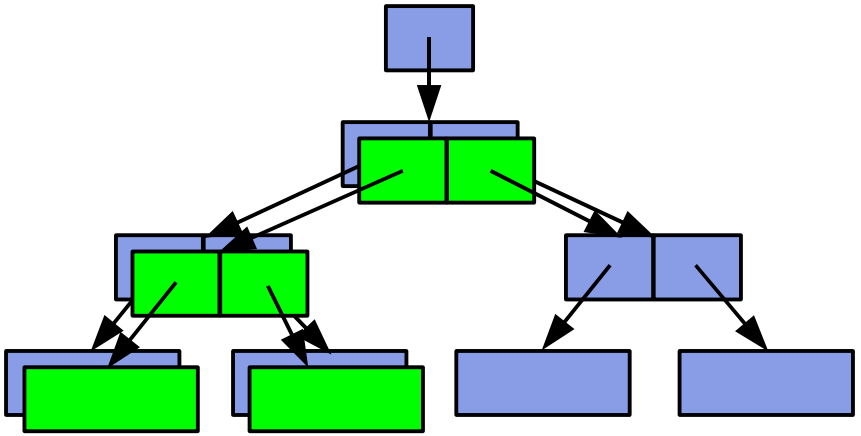
1. Initial block tree



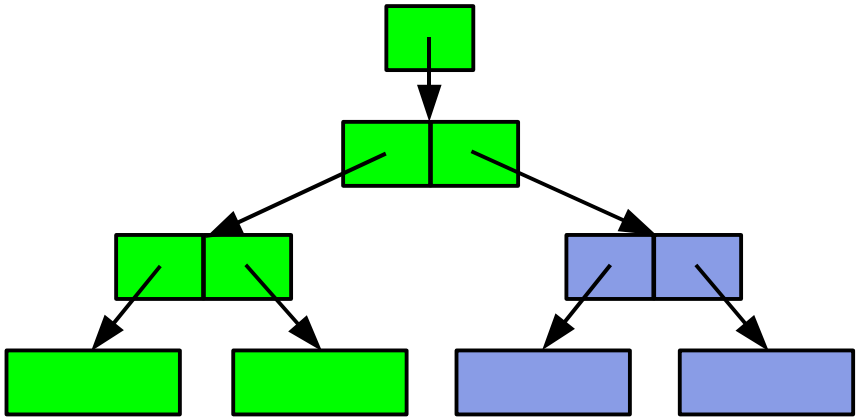
2. COW some blocks



3. COW indirect blocks

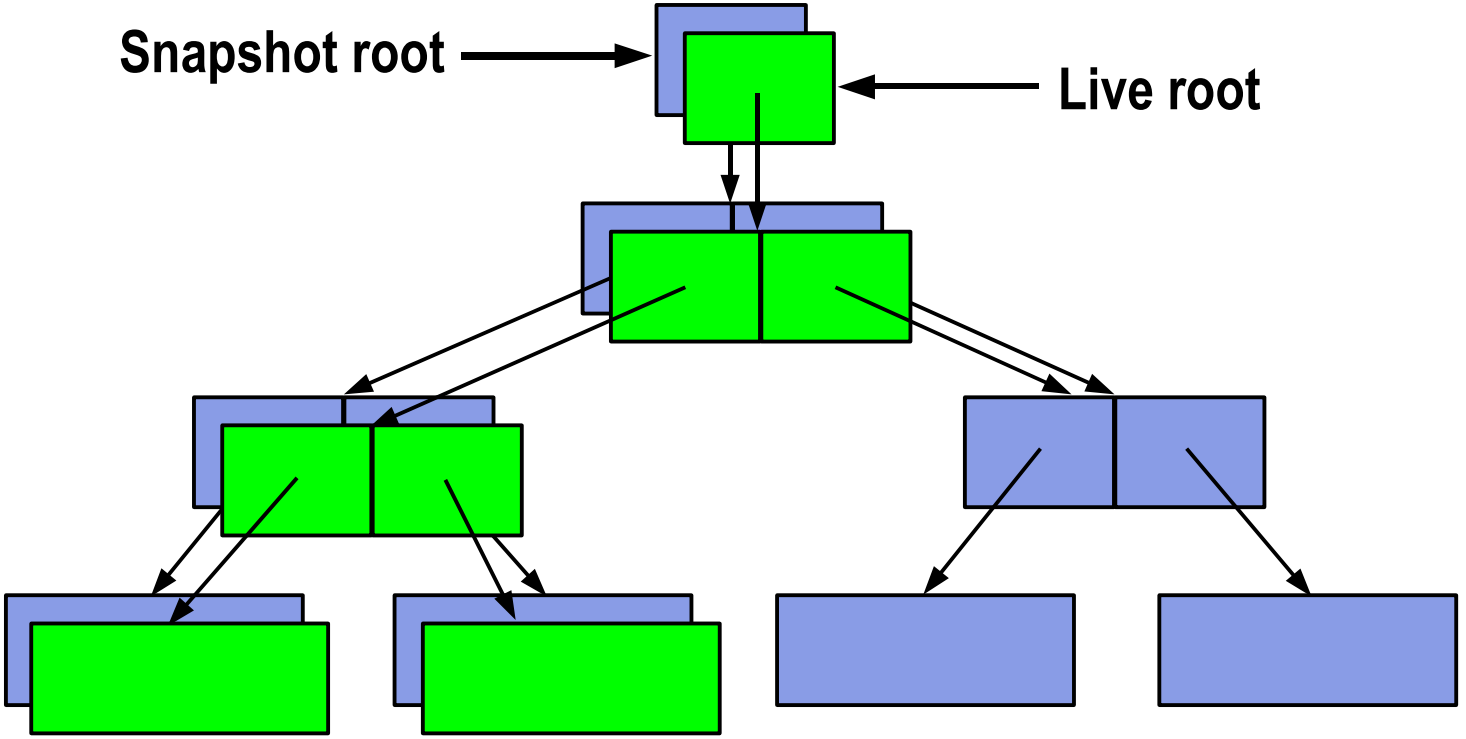


4. Rewrite uberblock (atomic)



Bonus: Constant-Time Snapshots

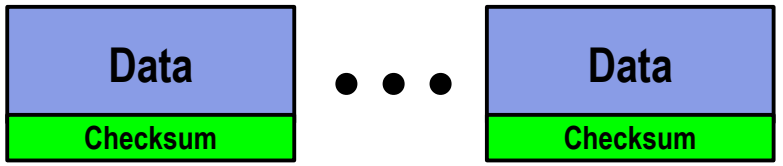
- At end of TX group, don't free COWed blocks
  - Actually cheaper to take a snapshot than not!



End-to-End Data Integrity

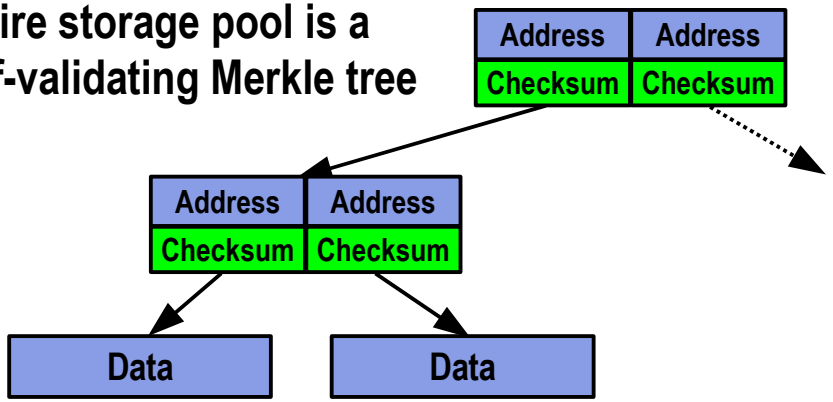
Disk Block Checksums

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



ZFS Data Authentication

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



Disk checksum only validates media

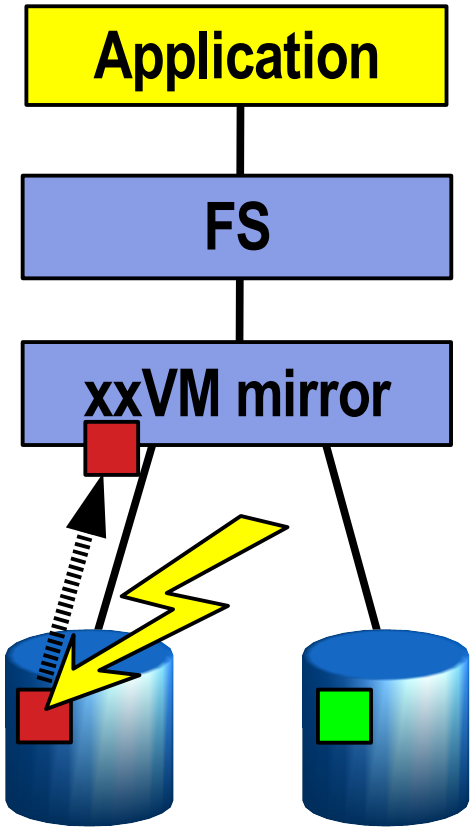
- ✔ Bit rot
- ✗ Phantom writes
- ✗ Misdirected reads and writes
- ✗ DMA parity errors
- ✗ Driver bugs
- ✗ Accidental overwrite

ZFS validates the entire I/O path

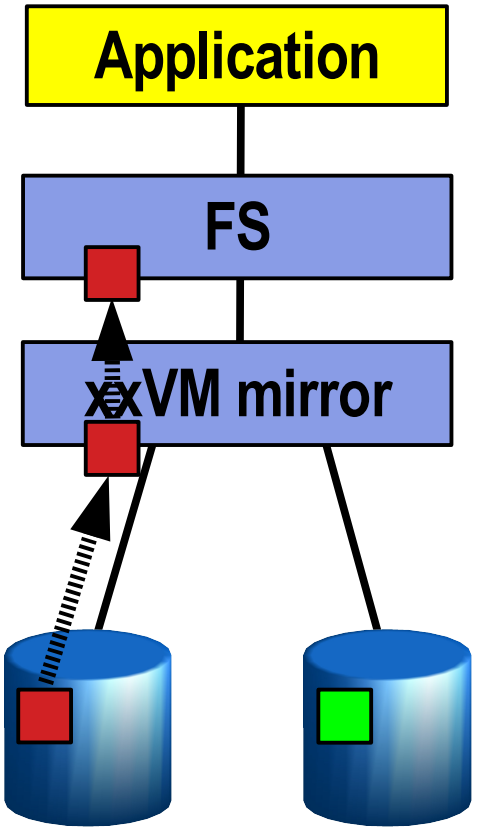
- ✔ Bit rot
- ✔ Phantom writes
- ✔ Misdirected reads and writes
- ✔ DMA parity errors
- ✔ Driver bugs
- ✔ Accidental overwrite

Traditional Mirroring

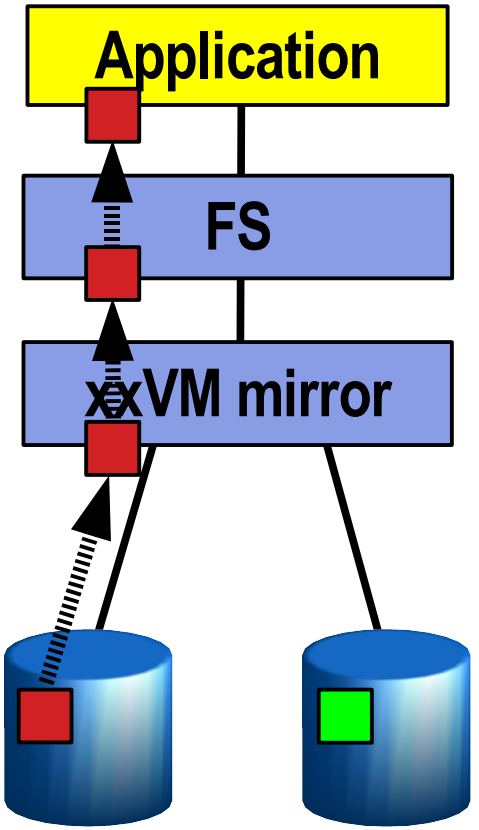
1. Application issues a read. Mirror reads the first disk, which has a corrupt block. It can't tell.



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

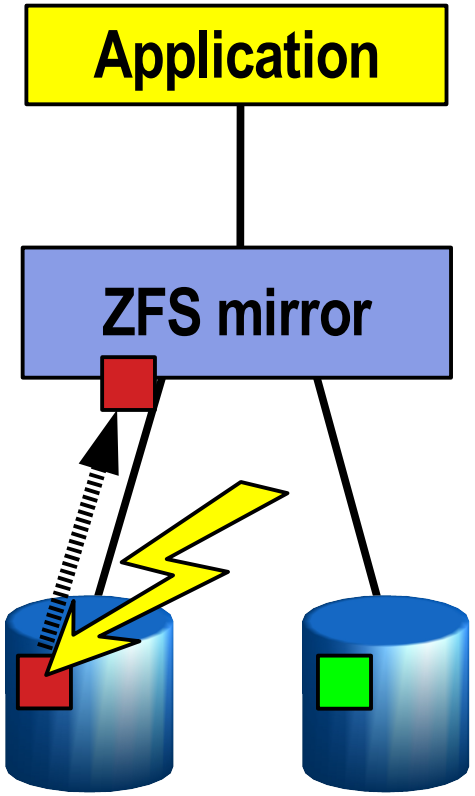


3. Filesystem returns bad data to the application.

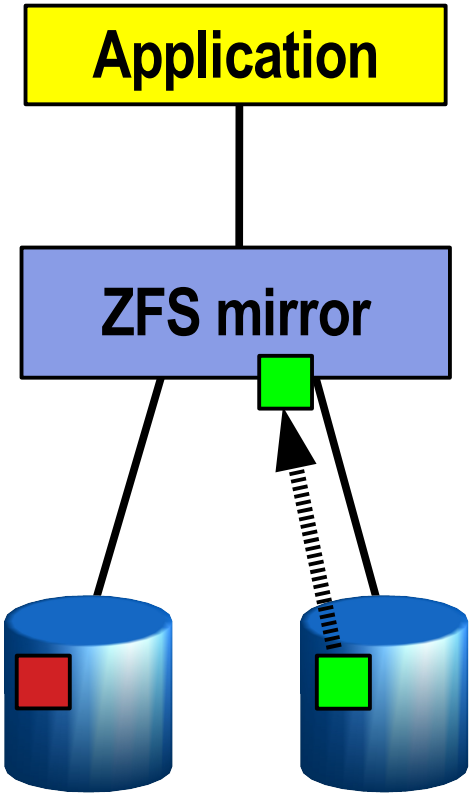


Self-Healing Data in ZFS

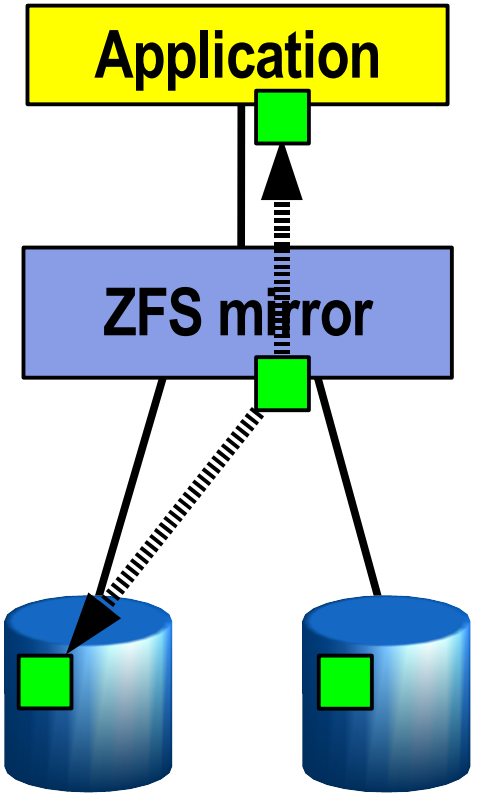
1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



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.



RAID-Z

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

LBA \ Disk	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 Scalability

- Immense capacity (128-bit)
  - Moore's Law: need 65th bit in 10-15 years
  - ZFS capacity: 256 quadrillion ZB (1ZB = 1 billion TB)
  - 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
  - Byte-range locking: parallel read/write without violating POSIX
  - Parallel, constant-time directory operations

ZFS Performance

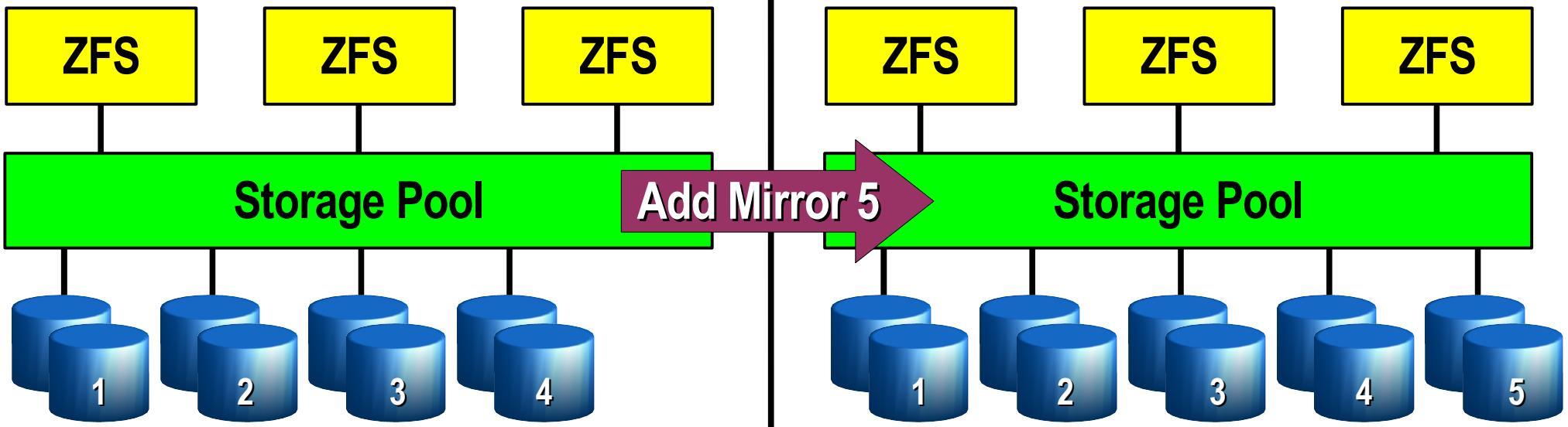
- Copy-on-write design
  - Turns random writes into sequential writes
  - Intrinsically hot-spot-free
- Multiple block sizes
  - Automatically chosen to match workload
- Pipelined I/O
  - Fully scoreboarded 24-stage pipeline with I/O dependency graphs
  - Maximum possible I/O parallelism
  - Priority, deadline scheduling, out-of-order issue, sorting, aggregation
- Dynamic striping across all devices
- 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



How to Deploy ZFS

Pool Setup and Configuration

- Top-level vdevs
- Redundancy at ZFS level
  - Yes!
  - Use spares
  - Software RAID vs. Hardware RAID
- Only 1 pool per system
  - Separate intent log (if necessary)
  - 2 pools if using ZFS root (1 for root, 1 for data)

Number of File Systems

- Have lots!
  - There is a cost per-FS when mounting/sharing (fix going through final testing)
  - There is a small per-FS memory overhead
- Think 1 per user
  - UFS/VxFS have 1 directory per user
  - ZFS has 1 file system per user
- Quotas and Reservations
  - Per file system, not per user

Properties

- Stored with the data
  - No more /etc
- Properties per file system
- Properties per pool
- Can be set dynamically
  - Some should be set at creation time
- Inheritance

Backup

- zfs send/recv
  - Very useful, but not a true backup scheme
- Veritas NetBackup
- Legato Networker
- NDMP (coming to OpenSolaris)

Replication

- asynchronous
  - zfs send/recv
- synchronous
  - AVS
  - http://www.opensolaris.org/os/project/avs

Delegated Administration

- Ability to delegate zfs(1M) administrative tasks to normal users
- 'zfs allow'
- 'zfs unallow'
- Example:
  - # zfs allow marks create,snapshot tank/marks
- http://blogs.sun.com/marks/entry/zfs_delegated_administration

Command History

- 'zpool history'
  - -l for long format (user, hostname, zone)
  - -i for “internal events” (debugging)
- Example:

```
diskmonster# zpool history
History for 'wombat':
2007-09-03.19:15:15 zpool create wombat mirror c7t7d0 c7t6d0
2007-09-03.19:15:30 zpool add wombat mirror c7t5d0 c7t4d0
2007-09-03.19:15:40 zfs create -p wombat/sub/fs/here
```

Is ZFS a Cluster File System?

- No
- At least not yet
- And no, no plans yet
  - On the radar?
    - No
      - No blip?
        - No blip

So if ZFS isn't a Cluster File System, what do I do?

- SunCluster for HA
- pNFS (coming)
- Lustre for HPC

ZFS + DB Setup

- 1 pool
  - Separate intent log
- Set recordsize to match your database I/O size
- Other resources:
  - http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide
  - http://blogs.sun.com/erickustarz/entry/vdev_cache_improvments_to_help
  - http://blogs.sun.com/realneel/entry/zfs_and_databases
  - http://blogs.sun.com/realneel/entry/zfs_and_databases_time_for

So you want to help develop ZFS? Develop using ZFS?

On-Disk Version

- SPA_VERSION
  - Older bits that don't support your pool's version can't import your pool
  - 'zpool upgrade'
  - http://www.opensolaris.org/os/community/zfs/version/[1-8]
- ZPL_VERSION
  - For incompatibilities on 'zfs recv' end
  - 'zfs upgrade'

Adding a Property

- 3 Property Types (zfs_proptype_t)
  - Number (ex: recordsize)
  - String (ex: sharenfs)
  - Index (ex: checksum)
- To add a new Property
  - zfs_prop_init()
  - register_{number,string,index,impl}
- zfs_prop.h/zfs_prop.c

Adding a Pipeline Stage

- zio_impl.h
  - zio_stage_t
  - Should it be in ZIO_ASYNC_PIPELINE_STAGES?
- zio.c
  - Add your pipeline function
- Future:
  - Encryption
  - <your idea here>

Block Allocator

- space_map_ops / metaslab_ff_ops
  - Currently only have“first-fit”
- metaslab_weight()
  - Takes into account capacity
  - Favors lower LBAs
  - Doesn't account for slower vdevs
- metaslab.h/metaslab.c

.zfs directory

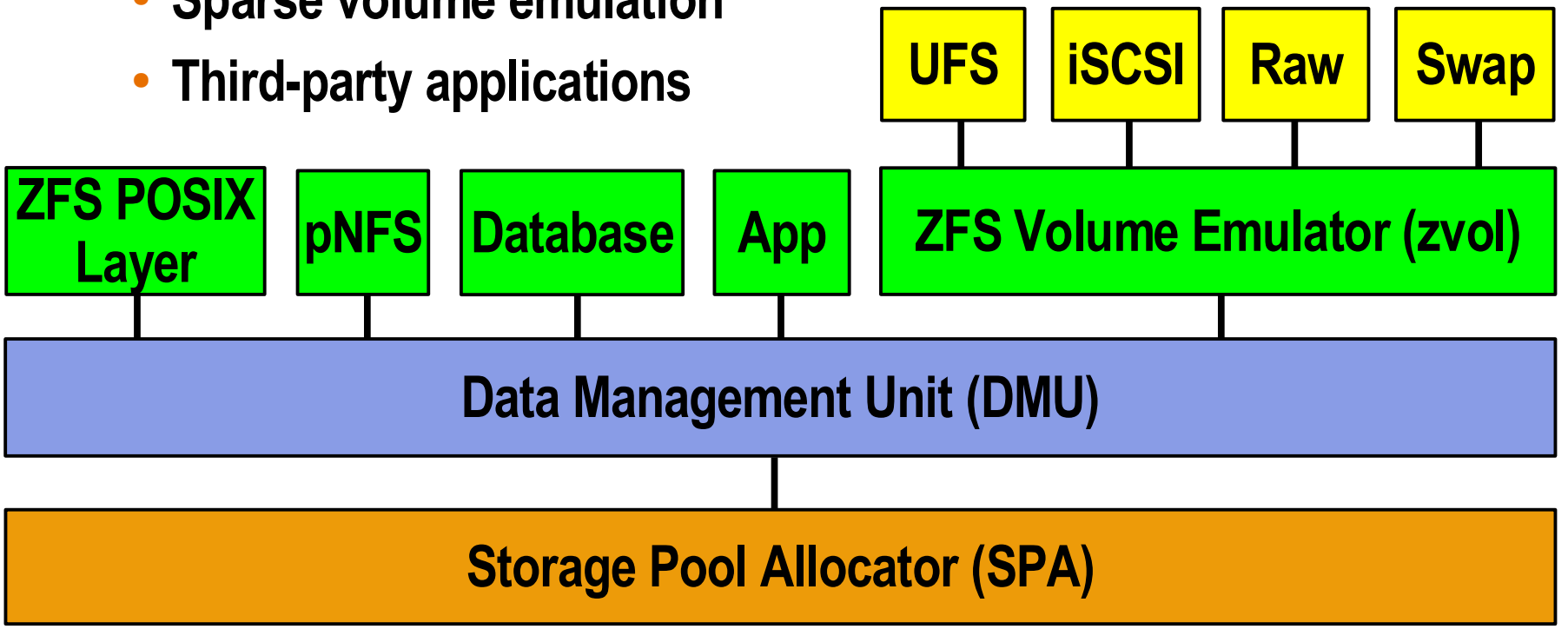
- Browse snapshots via .zfs/snapshot
- Accessible over NFS
- Future:
  - .zfs/props
    - get/set ZFS properties over NFS
  - <insert your idea here>
- zfs_ctldir.h/zfs_ctldir.c

DMU consumers

- Currently have:
  - ZPL (ZFS Posix Layer)
  - ZVOL (ZFS Volume Emulator)
  - pNFS Data Server
  - Lustre
- In the future...
  - Database(s)
  - <insert your idea here>

Object-Based Storage

- DMU is a general-purpose transactional object store
  - Filesystems
  - Databases
  - Swap space
  - Sparse volume emulation
  - Third-party applications



Using your new DMU Consumer

- New objects
  - dmu_object_type_t
  - dmu_object_type_info_t / dmu_ot[]
- dmu_object_alloc/free()
- dmu_read/write()
- dmu.h/dmu.c

Transaction Flow for DMU Consumers

- dmu_tx_create()
- dmu_hold_XXX()
- dmu_tx_assign()
- Modify your Data
  - Ex: dmu_write{_uio}()
- dmu_tx_commit()

Test Suite

- http://www.opensolaris.org/os/community/zfs/zfstestsuite
- Contribute new tests
- Contribute new features (more automatic?)
- Help port the test suite
  - OSX!
  - FreeBSD

Links/Resources

Main ZFS page

- http://opensolaris.org/os/community/zfs/
  - Source
  - Man pages
  - Admin guide
  - On-disk format guide
  - Data Structures
  - Demos
  - Links

THE mailing lists

- zfs-discuss@opensolaris.org
  - Questions
  - Feedback
  - Features you want
  - Performance Results
- zfs-code@opensolaris.org
  - Code questions
  - Code contributions

ZFS Ports

- http://opensolaris.org/os/community/zfs/porting
- OSX
- FreeBSD
- Linux FUSE
- <insert your work here>

Integrating ZFS into your Application Environment

Brian Wong

www.opensolaris.org/os/community/zfs

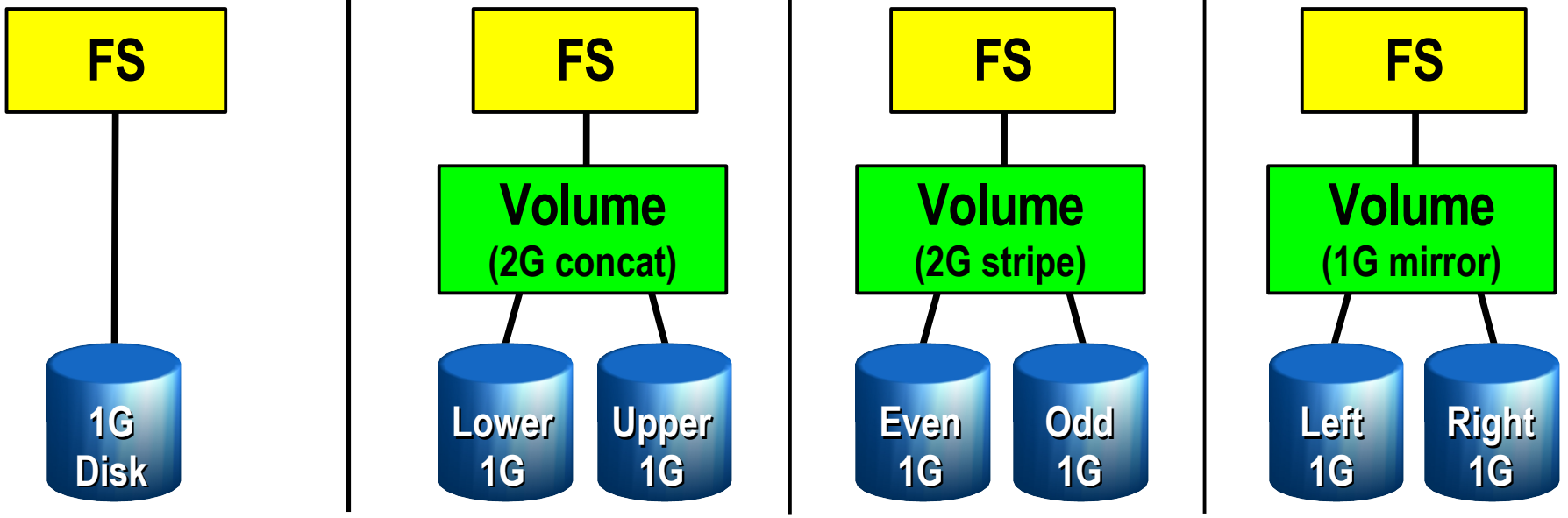
BACKUP SLIDES TO FOLLOW

-
-

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 interface can't be fixed



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

Traditional Resilvering

- Creating a new mirror (or RAID stripe):
  - Copy one disk to the other (or XOR them together) so all copies are self-consistent – even though they're all random garbage!
- Replacing a failed device:
  - Whole-disk copy – even if the volume is nearly empty
  - No checksums or validity checks along the way
  - No assurance of progress until 100% complete – your root directory may be the last block copied
- Recovering from a transient outage:
  - Dirty region logging – slow, and easily defeated by random writes

Smokin' Mirrors

- Top-down resilvering
  - ZFS resilvers the storage pool's block tree from the root down
  - Most important blocks first
  - Every single block copy increases the amount of discoverable data
- Only copy live blocks
  - No time wasted copying free space
  - Zero time to initialize a new mirror or RAID-Z group
- Dirty time logging (for transient outages)
  - ZFS records the transaction group window that the device missed
  - To resilver, ZFS walks the tree and prunes by DTL
  - A five-second outage takes five seconds to repair

Ditto Blocks

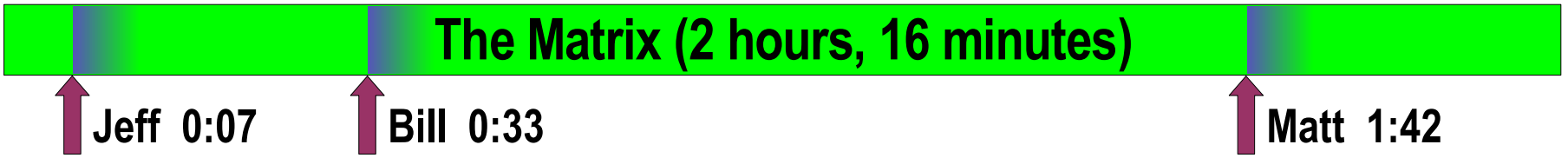
- Data replication above and beyond mirror/RAID-Z
  - Each logical block can have up to three physical blocks
    - Different devices whenever possible
    - Different places on the same device otherwise (e.g. laptop drive)
  - All ZFS metadata 2+ copies
  - Settable on a per-file basis for precious user data (snv_61)
- Detects and corrects silent data corruption
  - If the first copy is missing or damaged, try the ditto blocks
  - 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

Disk Scrubbing

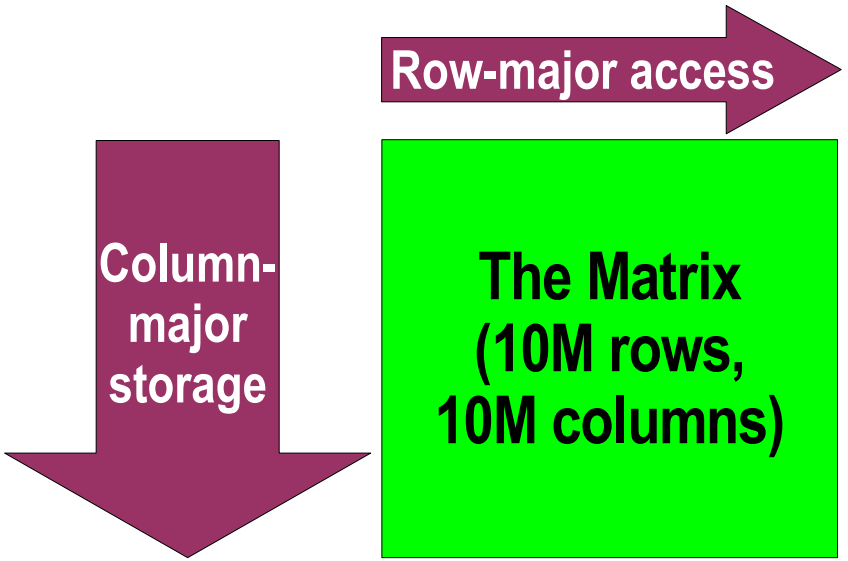
- Finds latent errors while they're still correctable
  - ECC memory scrubbing for disks
- Verifies the integrity of all data
  - Traverses pool metadata to read every copy of every block
    - All mirror copies, all RAID-Z parity, and all ditto blocks
  - Verifies each copy against its 256-bit checksum
  - Self-healing as it goes
- Minimally invasive
  - Low I/O priority ensures that scrubbing doesn't get in the way
  - User-defined scrub rates coming soon
    - Gradually scrub the pool over the course of a month, a quarter, etc.

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



ZFS Administration

- Pooled storage – no more volumes!
  - All storage is shared – no wasted space, no wasted bandwidth
- Filesystems become administrative control points
  - Hierarchical, with inherited properties
    - Per-dataset policy: snapshots, compression, backups, privileges, etc.
    - Who's using all the space? du(1) takes forever, but df(1M) is instant
  - Control compression, checksums, quotas, reservations, and more
  - Delegate administrative privileges to ordinary users
  - Policy follows the data (mounts, shares, properties, etc.)
  - Manage logically related filesystems as a group
    - 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 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 >/backup/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
```

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

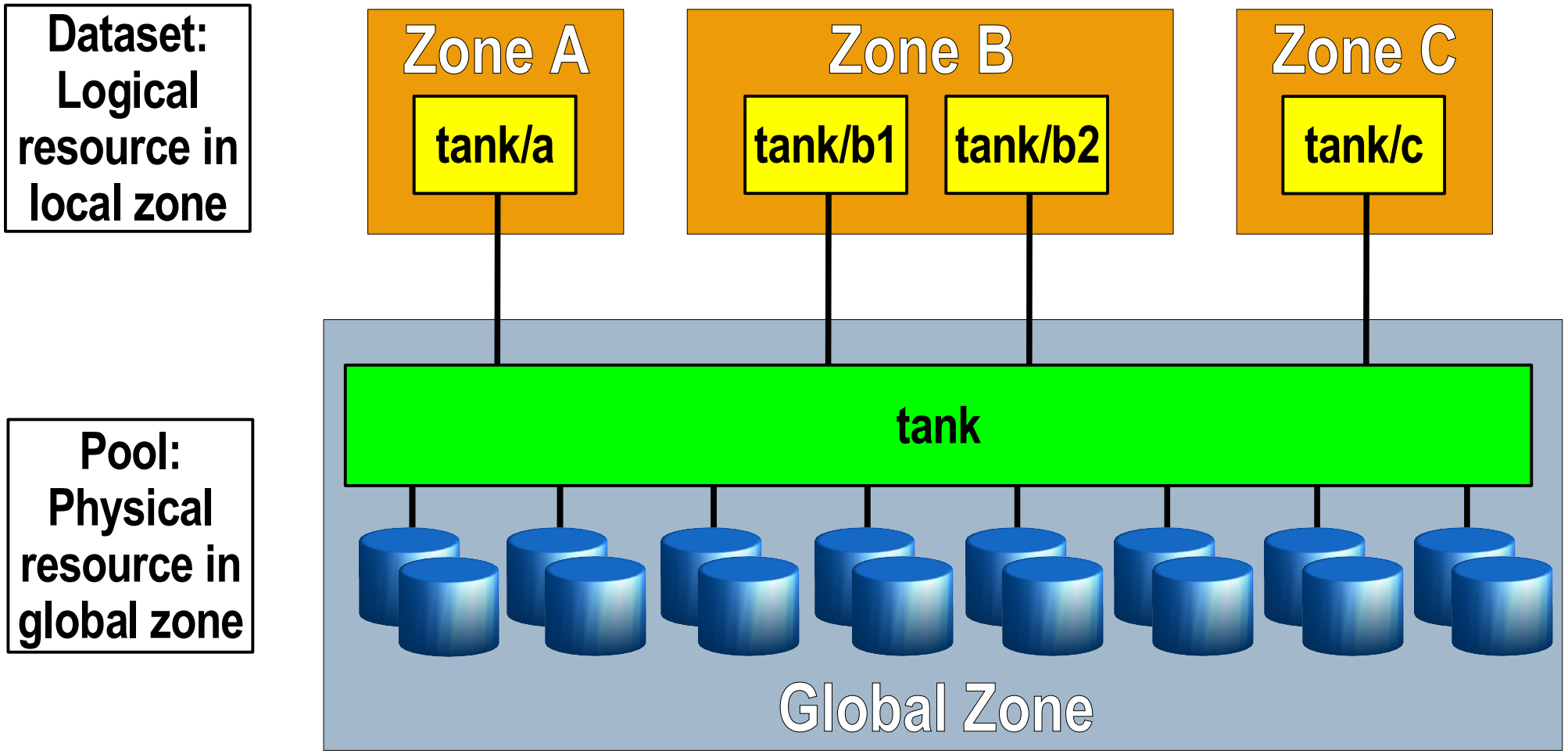
ZFS Root (snv_62)

- Brings all the ZFS goodness to /
  - Checksums, compression, replication, snapshots and clones
  - Boot from any dataset
- Patching becomes safe
  - Take snapshot, apply patch... rollback if you don't like it
- Live upgrade becomes fast
  - Create clone (instant), upgrade, boot from clone
  - No “extra partition”
- Based on new Solaris boot architecture
  - ZFS can easily create multiple boot environments
  - GRUB can easily manage them

ZFS and Zones (Virtualization)

Strong security model

Local zones cannot even see physical devices



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 losing data integrity or leaking a single block**

ZFS Summary

End the Suffering • Free Your Mind

- **Simple**
  - Concisely expresses the user's intent
- **Powerful**
  - Pooled storage, snapshots, clones, compression, scrubbing, RAID-Z
- **Safe**
  - Detects and corrects silent data corruption
- **Fast**
  - Dynamic striping, intelligent prefetch, pipelined I/O
- **Open**
  - http://www.opensolaris.org/os/community/zfs
- **Free**