THE UNIVERSITY *of* EDINBURGH

Installation and Experience at Edinburgh

Marcus Ebert
marcus.ebert@ed.ac.uk

- Why not a traditional storage system
- What is ZFS
- ZFS Features and current status on Linux

THE UNIVERSITY *of* EDINBURGH

Installation and Experience at Edinburgh

Marcus Ebert
marcus.ebert@ed.ac.uk

- Why not a traditional storage system
- What is ZFS
- ZFS Features and current status on Linux

Using the example of the Edinburgh storage installation

Traditional storage installation at Edinburgh

Traditional GridPP storage system @Edinburgh

- 3x Dell R610, 12GB RAM, 8/16C E5620 2.4GHz
  PERC H800, 36x2TB NL6 7.2k in 3xPowerVault MD1200
  RAID6, 17 disks + 1 HS (2x)

- 3x Dell R510, 24GB RAM, 8/16C E5620 2.4GHz
  12+24 x 2TB NL6 7.2k internal (H700) and in 2xMD1200 (H800)
  RAID6, 11 disks + 1 HS and 23 disks + 1HS

- 5x Dell R720xd, 64GB RAM, 12/24C E5-2620v2 2.1GHz
  12+24 x 4TB NL6 7.2k internal (H700) and in 2xMD1200 (H800)
  RAID6, 11 disks + 1 HS and 23 disks + 1HS

Traditional storage server setup

1. Creating and initialization of the hardware raid
   - Over 11 to 17 disks, 22TB to 92TB
   - Takes long time, access performance reduced while ongoing
2. Use *fdisk* to partition the virtual disks
   - Fast, but additional manual step
   - Not too long ago ext4 was limited to 16TB (ext3 still is)
3. Use *mkfs* to create filesystem
   - *mkfs.ext4* on single 8TB partition takes 1:11h on H700, 0:34h on H800
   - Can take whole day just for formatting disk space on a server
4. *mkdir* for mount point
5. Edit */etc/fstab*
6. *mount*

Only 2TB and 4TB disks used so far, imagine upgrading to larger disks...

Data protection in RAID6+HS

- Writes go to all disks
- Reads come from data disks only
- When a disk is unavailable, read remaining data disks+ redundancy disk and rebuild broken disk from all remaining disks using HS
  - Needs to read all disk blocks on all disks, even if empty
    100TB raid, 10% full, with 8TB disks: read 100TB, write 8TB
  - Rebuild always takes very long time
  - Data only available once the rebuild finished
- Protect against disk failures and makes sure the application gets data
- Can not protect against silent data corruption!
  - It doesn't know if the data read is the data the application sent to disk
  - If wrong data is read, no way to correct it even if it could use the redundancy in the RAID

There is not really data protection with hardware raid systems...

Tests at CERN

- Write $\sim$ 2GB file with specific pattern in it, read it back, compare patterns
  - Deployed on 3000 nodes, run every 2h
  - After 5 weeks, 500 errors on 100 nodes
    - Single bit errors: 10%
    - Sector or page sized region errors: 10%
    - Bug in WD disk firmware: 80%
- Hardware raid verify run on 492 systems (1.5PB) over 4 weeks
  - "fix" of 300 block errors
  - However, hardware raid doesn't know if data or parity block is wrong
- Read normal files back from disk and compare to checksum stored on tape
  - 33,700 files checked (8.7TB)
  - 22 mismatches found
  - 1 out of 1500 files bad
  - (at least) 1 error every 400GB
  - Byte error rate of 3×10^{-7}

https://indico.cern.ch/event/13797/contributions/1362288/attachments/115080/163419/Data_integrity_v3.pdf

back in 2007 by Dr. Bernd Panzer-Steindel

Was done before and later with similar results, also regulary done by companies like IBM

Failure scenario

Copy 100GB file to the RAID array, reboot in the middle through iDRAC. After reboot it shows:

```
[root@pool7 gridstorage02]# ls -lh
total 1.9G
-rw-r--r--    1 root   root    1.9G Mar 18 17:11 test.tar

[root@pool7 gridstorage02]# df -h .
Filesystem            Size  Used Avail Use% Mounted on
/dev/sdb2             8.1T   214M  8.1T   1% /mnt/gridstorage02

[root@pool7 gridstorage02]# du -sch .
 1.9G    .
 1.9G    total

[root@pool7 gridstorage02]# rm test.tar
[root@pool7 gridstorage02]# du . -sch
 41M     .
 41M     total

[root@pool7 gridstorage02]# df -h .
Filesystem            Size  Used Avail Use% Mounted on
/dev/sdb2             8.1T -1.7G  8.1T   0% /mnt/gridstorage02
```

Upgrade of disks

- Purchased 34 8TB disks
- Wanted to replaced 2TB disks and get as much additional space as reasonable
- How to handle this with a hardware raid??
  - Machines with 3xMD1200 are oldest, out of warranty, have lowest RAM
  - 3 other machines with 2TB disks and 2 MD1200 each
    * All 8TB disks on one server? 2x2TB disks left on the same controller, no use
      H700 showed worst performance than H800
    * Only 12x8TB disks per server to fill internal space? 10x8TB disks left
      H700 showed worst performance than H800
    * Split between 2 servers equally? 2x7x2TB disks have to be in 2 additional raid6
      loose at least another 4 disks for parity + maybe 2 for HS, not efficient

No good way to do this upgrade efficiently. All or nothing with hardware raid.

Why not a traditional storage system

- No data awareness
  - Rebuilds need to read always all disks and write full new disk, no matter how much data is stored
  - Rebuild can take many days, and it's getting worst with larger disks in the future
    With long rebuild times, it's more likely to have more failed disks before redundancy is fully restored $\Rightarrow$ RAID5 dead for a reason, soon RAID6 will be too
- No data protection, only disk space protection
  - If a block can be read then it is delivered, no matter if what is read is correct or not
  - No way to correct silent data corruption, can't even detect it
- Long and complicated process to setup storage before usable in the OS
  - Will get worst in the future with larger disks
- File system on top can easily get confused in failure situations
- Strong connection between OS system config and storage availability
  - In case of upgrades/OS disk failures, system config needs to be redone to use storage
- Not easy to upgrade storage based on hardware raid systems

ZFS

What is ZFS

a very small selection of features...

- Pooled storage system that combines file system, volume manager, and raid system
- 128 bit transactional file system
- COW file system that transforms random writes into sequential writes
- Protection against silent data corruption and has self healing of data
- Supports on-disk compression
- Moves data and disk management out of the OS into the file system
- Simplifies storage management and very easy administration

A brief history of ZFS

- Developed by SUN Microsystems for Solaris
- Became OpenSource together with OpenSolaris in 2005
  - License: CDDL
- Oracle stopped publishing source code after SUN was acquired by Oracle
- OpenZFS formed and continued with the last open source code
  - Fully integrated into FreeBSD
  - Lustre can use ZFS
- Storage industry built on it
  - Oracle, Nexenta, iXsystems, ...
  - You can buy single servers managing up to 5PB
    (limited by rack space vs single disk size, and cost vs needs)
- Well regarded for reliability, data protection, and performance under (Open)Solaris

ZFS on Linux

- Unfortunately, CDDL and GPL are not compatible
  - Reason why it is not in kernel source and why ext4/xfs still used during the last 10 years
- Early tries to implement it through fuse
  - poor performance
- later native kernel modules became available
  - based on OpenZFS
  - Distribution of source code that one has to compile
  - DKMS usable
  - First stable version back in 2013
- Storage solutions based on it at enterprise-level
  - JovianDSS by Open-E, TrueNAS by iXsystems
- included in repositories of many Linux distributions
  - Repositories for RedHat exists
  - Fully integrated in Ubuntu 16.04, despite the license problem...
- Not all features ported, like encryption and iSCSI share
  But in active development - source on github

ZFS installation at Edinburgh

ZFS as plain storage file system

- All GridPP storage managed by ZFS on Linux

- Use 2x ZFS raidz2 + HS for machines with the new 8TB disks
  - 2 redundancy disks similar like raid6
  - No problem with mixing disks on different controllers
  - Gives even better performance when load goes to different controllers

- Use ZFS raidz3 + HS for machines with single kind of disks
  - 3 disks for redundancy

- Usable disk storage space managed by ZFS: 1,035TB
  (without compression taken into account)
  - On 11 machines and 396 disks (182x2TB disks + 180x4TB + 34x8TB disks)

- Enabled compression

ZFS storage server setup

1. Make disks available to the OS
   - Unfortunately, our hardware raid controllers purchased before do not support JBODs
   - Had to create 36 single raid0, 1 per disk
   - If possible use JBODs to have direct disk access.

2. Create the zfs raid system and mount it with a single command
   - *zpool create NAME raidz3 sda sdb ... ... sdk ... spare sdx*
   - This creates a raid system with 3 redundancy disks and 1 HS
   - Automatically mounted under /NAME, no need to edit system files!
   - Takes less than 10s

Very fast setup and independent of disk sizes

(If your controller supports JBOD, only a single step.)

Data protection in raidz2+HS

- Writes go to as many disks as needed
  - variable block size/stripe length depending on file size
- Reads come from data disks only
- When a disk is unavailable, read remaining data disks + redundancy disk and rebuild broken disk from all remaining disks using HS
  - Only needs to read blocks with live data
    100TB raid, 10% full, with 8TB disks: read only 10TB, write 0.8TB
  - Rebuild goes through the file system starting from root directory
  - With ongoing rebuild time, more and more redundancy is restored
- Protect against disk failures and makes sure the application gets data
- Can also protect against data corruption
  - Checksum calculated for each block and stored in parent block
  - Each read is compared against it's checksum to make sure the correct data is read
  - ZFS can detect wrong data is read, reconstruct correct data from redundancy and correct wrong data blocks
- ZFS never overwrites live data (COW), file system always consistent, never needs fsck
  - Random writes $\longrightarrow$ sequential writes
    Also reorders different write requests

Failure scenario

Copy 100GB file to the ZFS pool, reboot in the middle through iDRAC. After reboot it shows:

```
[root@pool6 ~]# ls -lh /tank
 total 46G
-rw-r--r--.  1 root root  16G Mar 19 19:07 test10G
-rw-r--r--.  1 root root  13G Mar 19 20:12 test.tar
-rw-------.  1 root root  18G Mar 19 20:06 .test.tar.EM379W

[root@pool6 ~]# df -h /tank
Filesystem      Size  Used Avail Use% Mounted on
tank             16T   46G   16T   1% /tank
[root@pool6 ~]# du -sch /tank
 46G     /tank
 46G     total
[root@pool6 ~]# rm /tank/*test.tar*
[root@pool6 ~]# du -sch /tank
 17G     /tank
 17G     total
[root@pool6 ~]# ls -lh /tank
total 16778239
-rw-r--r--.  1 root root  16G Mar 19 19:07 test10G
```

All really consistent?

```
[root@pool6 ~]# du -sch /tank
 17G     /tank
 17G     total
[root@pool6 ~]# ls -lh /tank
total 16778239
-rw-r--r--.  1 root root  16G Mar 19 19:07 test10G
```

- ZFS Indent Log (ZIL) saves actions that need to be done
- After the *rm* command it continues to free up space, *rm* doesn't need to wait until all is deleted
- *du* and *ls* became identical again later

ZFS properties

Selection of ZFS properties

List all properties and current status: *zfs get all ZFS-NAME*

```
[root@pool10 ~]# zfs get all tank-4TB
NAME      PROPERTY              VALUE                  SOURCE
tank-4TB  used                  80.3T                  -
tank-4TB  available             32.2T                  -
tank-4TB  referenced            54.8K                  -
tank-4TB  compressratio         1.04x                  -
tank-4TB  mounted               yes                    -
tank-4TB  quota                 none                   default
tank-4TB  reservation           none                   default
tank-4TB  recordsize            128K                   default
tank-4TB  mountpoint            /tank-4TB              default
```

- quota or reservations can be set directly in the file system
- Overview of used, available, and compressed space
- Mountpoint also specified within the file system, not in the OS
- Recordsize is variable, 128k used to calculate free space, could be set fixed for db applications

Selection of ZFS properties

```
tank-4TB  sharenfs              off                    default
tank-4TB  compression           lz4                    local
tank-4TB  exec                  on                     default
tank-4TB  readonly              off                    default
tank-4TB  copies                1                      default
tank-4TB  sharesmb              off                    default
tank-4TB  primarycache          all                    default
tank-4TB  secondarycache        all                    default
tank-4TB  dedup                 off                    default
tank-4TB  redundant_metadata    all                    default
```

- NFS and SMB exports directly stored in the file system, no need to edit OS files!
- Supports different modes of compression
- Filesystem can be made readonly or ban the execution of files on it
- Supports block level de-duplication
- Can define how different cache levels are used
- Metadata is stored with more than 1 copy by default
- Independent of raid level, each file can have more than 1 copy, works on block level
  - Could be interesting with larger and cheaper disks due to rebuild and performance issues
  - Use 2 (3) copies of all files instead of raid?

Administration

```
[root@pool10 ~]# zpool list
NAME       SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank-4TB   127T  87.9T  39.1T         -     4%    69%  1.00x  ONLINE  -
```

```
[root@pool10 ~]# zfs list
NAME                     USED  AVAIL  REFER  MOUNTPOINT
tank-4TB                80.3T  32.1T  54.8K  /tank-4TB
tank-4TB/gridstorage01  80.3T  32.1T  80.3T  /tank-4TB/gridstorage01
```

```
[root@pool10 ~]# zfs get compressratio tank-4TB/gridstorage01
NAME                    PROPERTY       VALUE  SOURCE
tank-4TB/gridstorage01  compressratio  1.04x  -
```

ZFS compression

- No negative impact on performance
- Average over all servers: 4%
- 4% for ATLAS files
- 16% for LSST files
- Assume 4% for total storage: about 42TB saved
  - gives 42TB extra space for free and reduces I/O
  - 42TB is nearly as much storage as on one single server with 2TB disks
  - Gives storage of such server for free, without the extra running cost for server and power

(We have mostly only ATLAS and LSST files on disk, LSST files are only some TB.)

Administration

```
[root@pool11 ~]# zpool iostat 1 10
               capacity     operations    bandwidth
pool         alloc   free   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
tank-4TB    56.5T  70.5T    342    865  40.7M  23.8M
tank-4TB    56.5T  70.5T    303      0  37.4M      0
tank-4TB    56.5T  70.5T     83    223  9.06M  22.8M
tank-4TB    56.5T  70.5T    308  6.07K  34.7M  78.2M
tank-4TB    56.5T  70.5T    276  6.38K  33.1M   140M
tank-4TB    56.5T  70.5T    243  6.67K  29.8M  80.6M
tank-4TB    56.5T  70.5T    114    558  14.2M  62.4M
tank-4TB    56.5T  70.5T    178  3.39K  20.9M  9.92M
tank-4TB    56.5T  70.5T     47      0  5.99M      0
tank-4TB    56.5T  70.5T    778      0  96.2M      0
```

- Shows stat for disk access, one output line per second, 10x
- First line shows average since zpool was imported
- In MB/s and in IOPS, separate for read and write
- Option "-v" shows that for all disks in the pool individually

Administration

```
[root@pool7 ~]# zpool status tank-8TB
  pool: tank-8TB
 state: ONLINE
  scan: scrub repaired 0 in 77h42m with 0 errors on Fri May 20 13:52:43 2016
config:

        NAME                                            STATE     READ WRITE CKSUM
        tank-8TB                                        ONLINE       0     0     0
          raidz2-0                                      ONLINE       0     0     0
            scsi-36782bcb0695e37001e99299e5fabc701      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929a3600263ae      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929a9605a701a      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929af60af386f      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929b4610790cc      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929ba61637907      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929c061bab963      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929c6621883b4      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929cc62768d77      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929d362d404da      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929d963339d48      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929df639196d3      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929e563ede75d      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929eb644fbd3d      ONLINE       0     0     0
            scsi-36782bcb0695e37001e9929f764fe9d24      ONLINE       0     0     0
            scsi-36782bcb0695e37001e992a0866055661      ONLINE       0     0     0
        spares
          scsi-36782bcb0695e37001e992a1766e9895b        AVAIL

errors: No known data errors
```

zfs scrub

- Each block has a 256bit checksum at a higher level in the file system tree
- Scrub checks every live data block, meta block, and duplicated block against its checksum
- Walks through the file system tree, top to down
- When errors are found, it repairs the data using redundancy (raid or additional copies) or tells you which files are broken
- Runs by default at low priority to not interfere with production, but can be adjusted
- Errors and progress are visible in *zpool status*
- Email can be sent out by errors or when finished with status report
- Same checks + repairs are also done during normal reading of data from disk, but regular scrub can detect problems earlier
- Setup monthly check for each pool

ZFS Event Daemon

- Configure through */etc/zfs/zed.d/zed.rc*
- Specify email to get notified about problems
- Specify when to get notified
- Specify under which conditions a hot spare is activated
  - Doesn't seem to work correctly
  - Probably due to use of raid0 instead of giving direct access to disk
  - Needs more testing
  - Email notification works fine + manual activation of spare disk

Data management

- To create a new file system under zpool tank: *zfs create tank/NewName*
  - Inherits all properties from upper file system
  - Can be used instead of directories (*df* vs *du*)
- To share a file system over NFS (similar for SMB): *zfs set sharenfs=on tank/Name*
  - That's all that is needed, now clients can mount it
  - Extra options are possible, like ro, root access, specific IPs,...
  - (Open)Solaris also supports shareiscsi
- Remove pool from OS: *zpool export POOL*
- Make pool available after OS upgrade/new install: *zpool import POOL*
  - Automatically mounts all filesystems in POOL
  - Automatically makes all zfs available over NFS/SMB if in properties
  - Automatically applies all quotas, reservations, and other properties
- History of all commands: *zpool history POOL*
- Event log: *zpool events POOL*

Availability

- Switched the first server about half year ago from hardware raid to ZFS
- Changed remaining 10 servers over the following months
- Last server switched to ZFS at end of April
- Stable without any availability problems
  - Large load on servers/disks, got writes from drained servers
- ZFS scrub found and repaired problems on 3 different servers
- 7 of the old 2 TB disks failed during the time ZFS was already running
  - caused no problems or visible performance issues
  - disks were changed, attached to the pool using zpool command, and rebuild started

Rebuild and performance

- Rebuild is slow by default
  - Same as scrub
  - Low priority compared to ongoing production use
  - Can be configured using properties under /sys/module/zfs/parameters/
  - Time also depends on how much data is stored in the zpool
- Our setup is optimized for capacity, not for speed
  - Still fast enough for the access pattern we see
  - Also fast enough for the 10Gbps network link
  - Draining a server with 20-30 total threads in parallel averages at 7Gbps, peaks at 9.6Gbps
- Have not looked into dedup
  - From previous experience using ZFS on different systems:
    large impact on performance, especially when deleting files, IF one has not enough RAM
  - Recommendation: 2-5GB/TB, O(100GB) RAM just for the dedup table in our case, depends on block size since dedup is block based
  - Will do some tests over the summer

Conclusion

- ZFS on Linux is as reliable and stable as on Solaris
  - If you can, use controllers that allow JBODs
- DKMS works fine with ZFS on SL6
- Very simple and easy management and administration of storage
- Provides real data protection compared to hardware raid, and self-healing
- Compression works very well even for LHC files
- **No need to use hardware raids or ext4/xfs any longer**
- Interesting option of having more than 1 copy of a file
  - Could become interesting when larger disks make rebuild times even on ZFS too long compared to expected errors on the other disks
  - Needs testing

ZFS on Linux is a filesystem with easy and simple management,
it has a high reliability without any errors or problems seen,
and **it comes with real data protection!**

Traditional Disk Storage Administration

http://www.iro.umontreal.ca/~dift3830/ZFS.pdf

But with ZFS....

http://www.iro.umontreal.ca/~dift3830/ZFS.pdf

If you want to use ZFS too

- ZFS on Linux main web page: `http://zfsonlinux.org/`

- Blog of the GridPP storage group: `http://gridpp-storage.blogspot.co.uk/`
  - Comparisons of hardware raid and ZFS (3 posts)
  - ZFS compression
  - Setup of a ZFS on Linux storage server for GridPP
  - Direct links to ZFS related GridPP storage posts: `http://ebert.homelinux.org/blogposts.html`

- ZFS talk by Bill Moore and Jeff Bonwick, main ZFS architects

  `https://www.cs.utexas.edu/users/dahlin/Classes/GradOS/papers/zfs_lc_preso.pdf`

  `http://wiki.illumos.org/download/attachments/1146951/zfs_last.pdf`

- Other talks about different ZFS features by other SUN people

  `http://www.slideshare.net/mewandalmeida/zfs`

  `http://www.slideshare.net/Clogeny/zfs-the-last-word-in-filesystems`

If you want to use ZFS too

- ZFS on Linux main web page: `http://zfsonlinux.org/`

- Blog of the GridPP storage group: `http://gridpp-storage.blogspot.co.uk/`
  - Comparisons of hardware raid and ZFS (3 posts)
  - ZFS compression
  - Setup of a ZFS on Linux storage server for GridPP
  - Direct links to ZFS related GridPP storage posts: `http://ebert.homelinux.org/blogposts.html`

- ZFS talk by Bill Moore and Jeff Bonwick, main ZFS architects

  `https://www.cs.utexas.edu/users/dahlin/Classes/GradOS/papers/zfs_lc_preso.pdf`

  `http://wiki.illumos.org/download/attachments/1146951/zfs_last.pdf`

- Other talks about different ZFS features by other SUN people

  `http://www.slideshare.net/mewandalmeida/zfs`

  `http://www.slideshare.net/Clogeny/zfs-the-last-word-in-filesystems`

Thank you!

Other interesting features

- Delegate privileges to users: *zfs allow -u USER PROPERTY ZFS*

- Create snapshot: *zfs snapshot tank/Name@first*
  - Instantly, uses only space for files deleted in main file system, read only
  - Whole file system visible in snapshot as it was when snapshot was done
- Backup/Copy to another server:
  zfs send tank/Name@first| ssh host2 zfs receive otherpool/Name
  - Sends whole file system as continuously stream, incl. all properties
- Can also send only differences
  - *zfs send -i tank/Name@first tank/Name@second | ssh host2 zfs receive otherpool/Name*
  - perfect for hourly/daily/monthly backups

- Create a block device with 100GB on zpool tank
  zfs create -V 100G tank/VM-device
  creates */dev/zvol/tank-4TB/VM-device*
  Can be mounted in VM to put own file system on it (still benefits from ZFS features!)

- *zfs* and *zpool* have very good man pages

zpool history

```
[root@pool7 ~]# zpool history tank-8TB
History for 'tank-8TB':
2016-05-06.08:13:20 zpool create tank-8TB raidz2 sdn sdo sdp sdq sdr sds sdt sdu sdv sdw \
  sdx sdy sdz sdaa sdac sdaf spare sdai
2016-05-06.08:13:57 zfs set compression=lz4 tank-8TB
2016-05-06.08:13:57 zpool set autoreplace=on tank-8TB
2016-05-06.08:13:57 zpool set autoexpand=on tank-8TB
2016-05-06.08:13:57 zfs set relatime=on tank-8TB
2016-05-06.08:13:57 zfs set xattr=sa tank-8TB
2016-05-06.08:13:57 zpool export tank-8TB
2016-05-06.08:14:03 zpool import -d /dev/disk/by-id tank-8TB
2016-05-06.08:14:08 zfs create tank-8TB/gridstorage01
2016-05-10.08:10:10 zpool scrub tank-8TB
2016-05-17.08:10:09 zpool scrub tank-8TB
```

zpool status after resilver

```
[root@pool4 ~]# zpool status
  pool: tank-2TB
 state: ONLINE
  scan: resilvered 1.60T in 44h58m with 0 errors on Mon May 30 05:42:31 2016
config:

        NAME                                            STATE     READ WRITE CKSUM
        tank-2TB                                        ONLINE       0     0     0
          raidz3-0                                      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f11477f7a3f      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f1547bf3010      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f1a480023ae      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f1e4842e23b      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f224885bba3      ONLINE       0     0     0
            scsi-36a4badb0450e83001ed869212db39a73      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f2b490fb750      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f3049566fb2      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f35499d9348      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f3949e71da5      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f3f4a341857      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f444a8023ee      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f6b4cd64c06      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f704d252627      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f754d72c8c9      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f7a4dc2cb69      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f804e15ad2c      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f844e5b495b      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f894ea169ad      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f8e4eea7c74      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f924f335552      ONLINE       0     0     0
            scsi-36a4badb0450e83001eb08f974f7d6671      ONLINE       0     0     0
            ...
```

zpool status during scrub

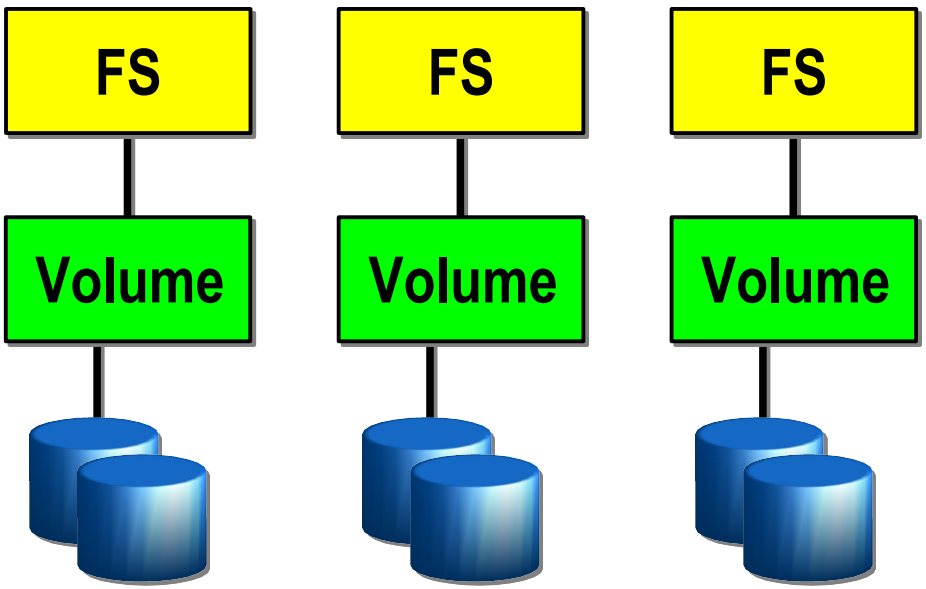
```
[root@pool10 ~]# zpool status
  pool: tank-4TB
 state: ONLINE
  scan: scrub in progress since Fri Jun 17 07:15:03 2016
    36.0T scanned out of 88.1T at 188M/s, 80h39m to go
    0 repaired, 40.84% done
config:

        NAME                                            STATE     READ WRITE CKSUM
        tank-4TB                                        ONLINE       0     0     0
          raidz3-0                                      ONLINE       0     0     0
            scsi-36c81f660dad421001ec135d3276cbd81      ONLINE       0     0     0
            scsi-36c81f660dad421001ec135d6279f13d9      ONLINE       0     0     0
            scsi-36c81f660dad421001ec135d927d1a684      ONLINE       0     0     0
            scsi-36c81f660dad421001ec135dd28066193      ONLINE       0     0     0
            scsi-36c81f660dad421001ec135e0283ad4c7      ONLINE       0     0     0
            scsi-36c81f660dad421001ec135e3286f0565      ONLINE       0     0     0
            scsi-36c81f660dad421001ec135e728a432ed      ONLINE       0     0     0
            scsi-36c81f660dad421001ec135eb28dad9ff      ONLINE       0     0     0
            scsi-36c81f660dad421001ec135ee2911b9c5      ONLINE       0     0     0
            ...
```

Following pages are from
http://wiki.illumos.org/download/attachments/1146951/zfs_last.pdf

FS/Volume Model vs. Pooled Storage

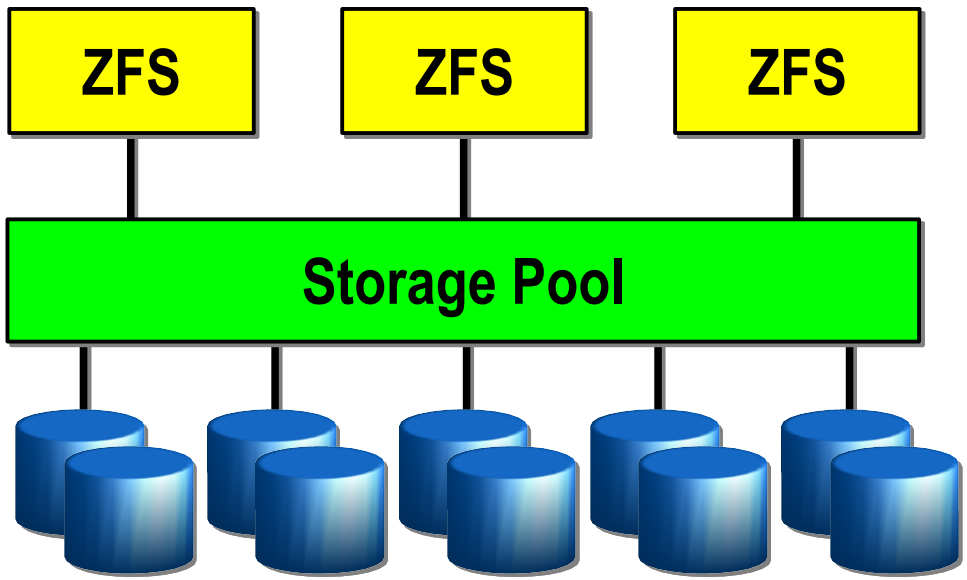
Traditional Volumes

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



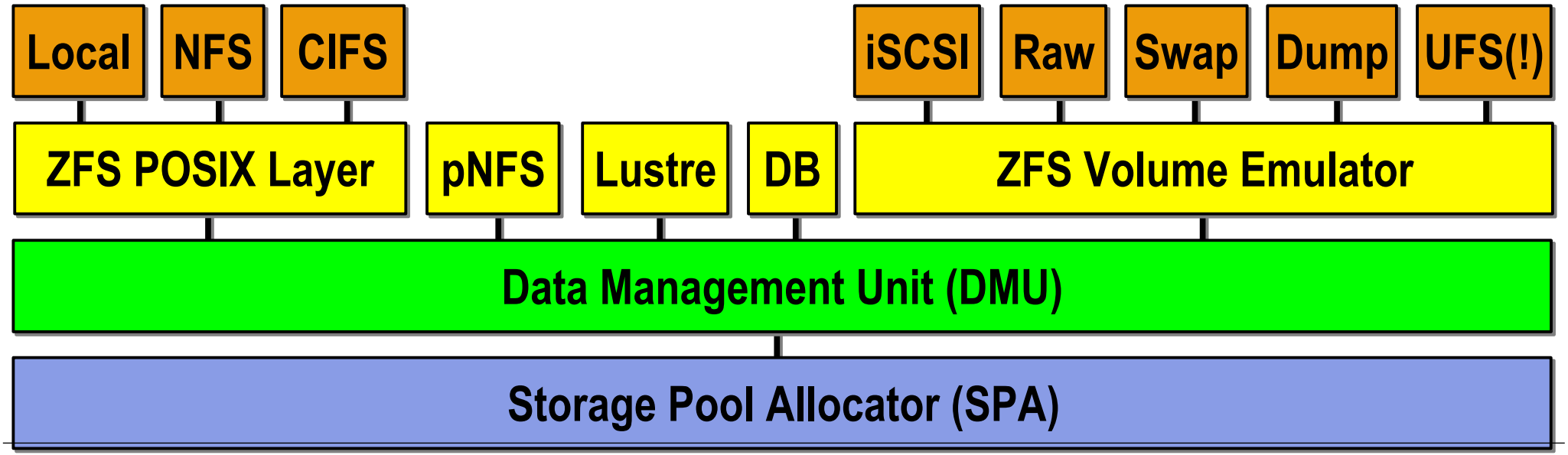
ZFS Pooled Storage

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



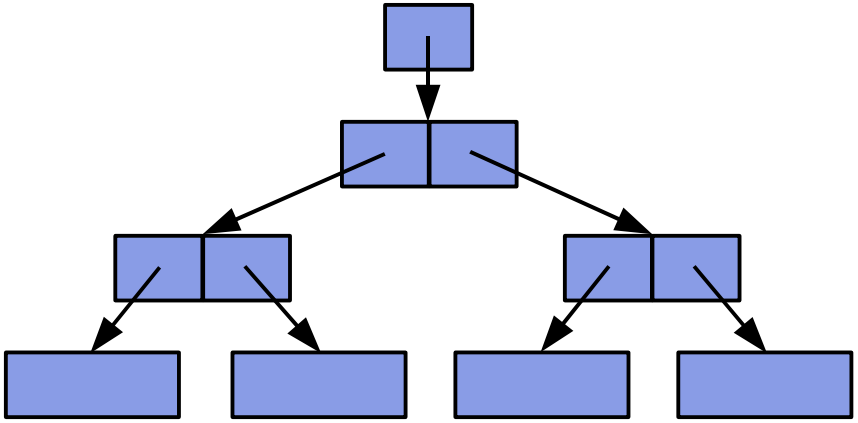
Universal Storage

- 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

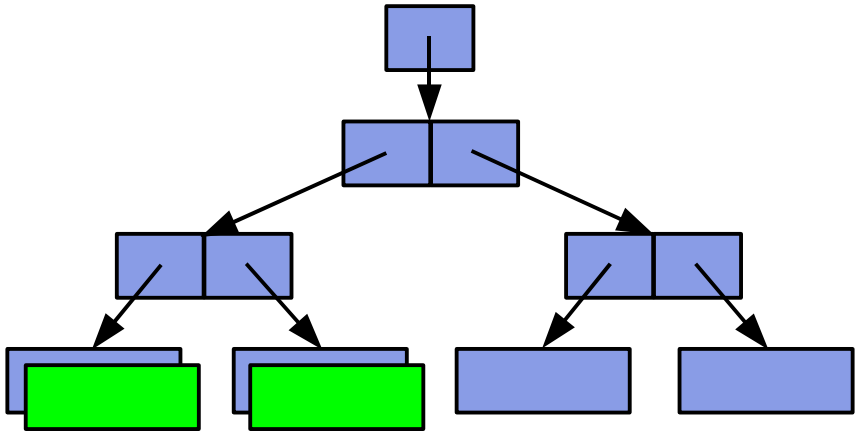


Copy-On-Write Transactions

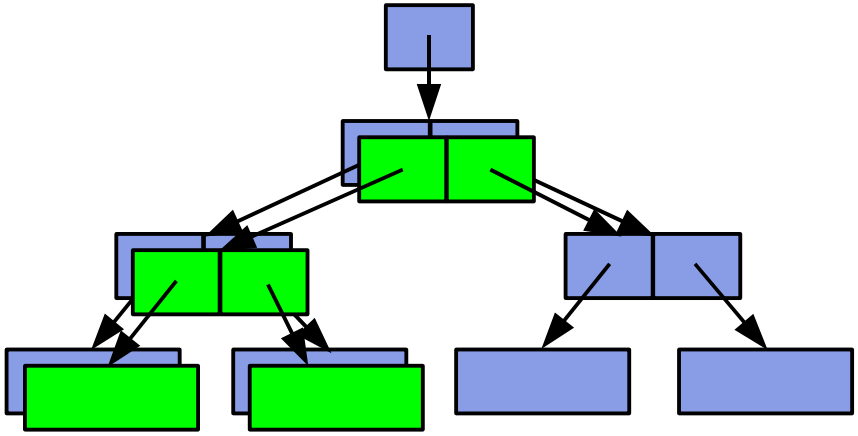
1. Initial block tree



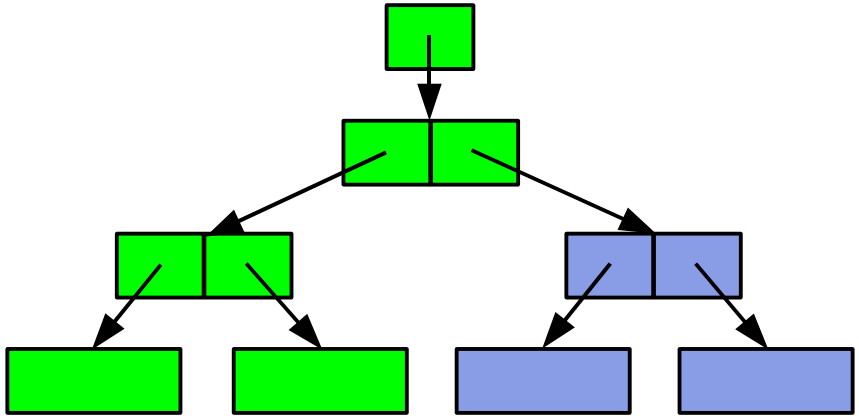
2. COW some blocks



3. COW indirect blocks



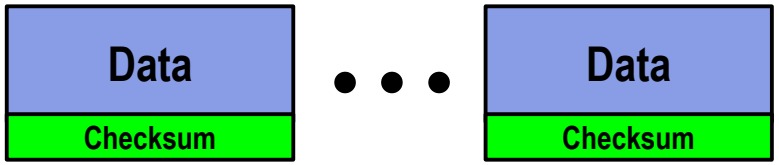
4. Rewrite uberblock (atomic)



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

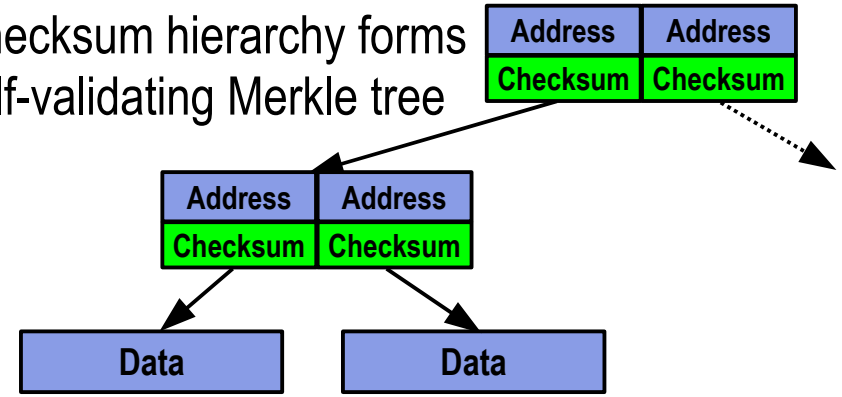


Disk checksum only validates media

✔	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

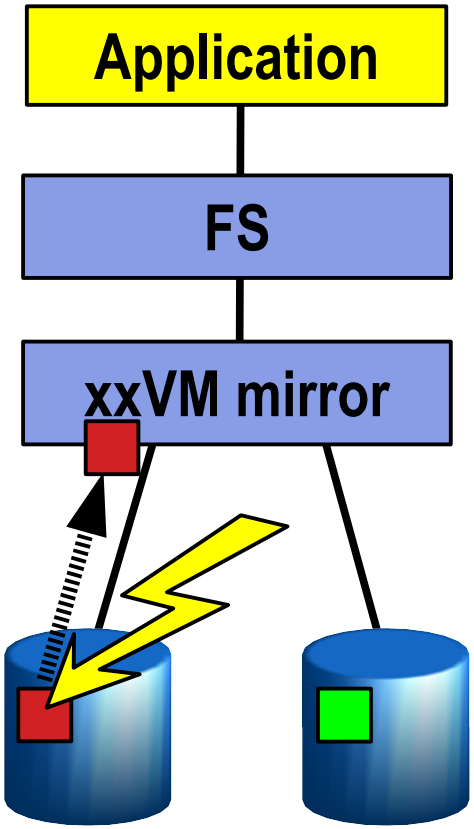


ZFS validates the entire I/O path

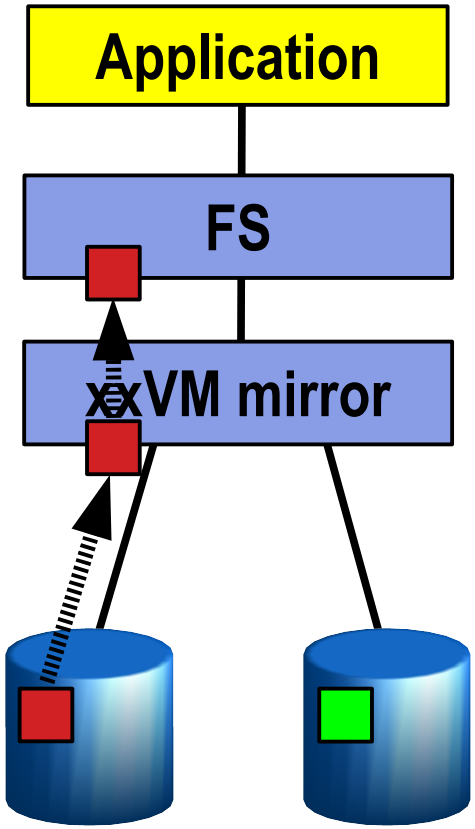
✔	Bit rot
✔	Phantom writes
✔	Misdirected reads and writes
✔	DMA parity errors
✔	Driver bugs
✔	Accidental overwrite

Traditional Mirroring

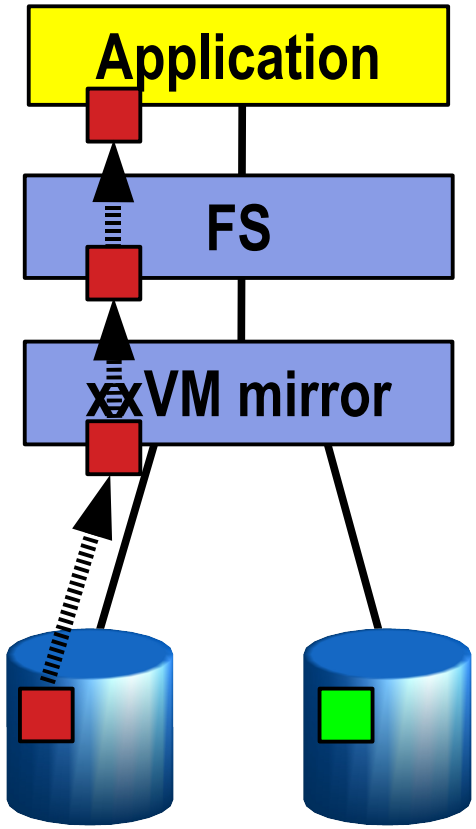
1. Application issues a read. Mirror reads the first disk, which has a corrupt block. It can't tell.



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

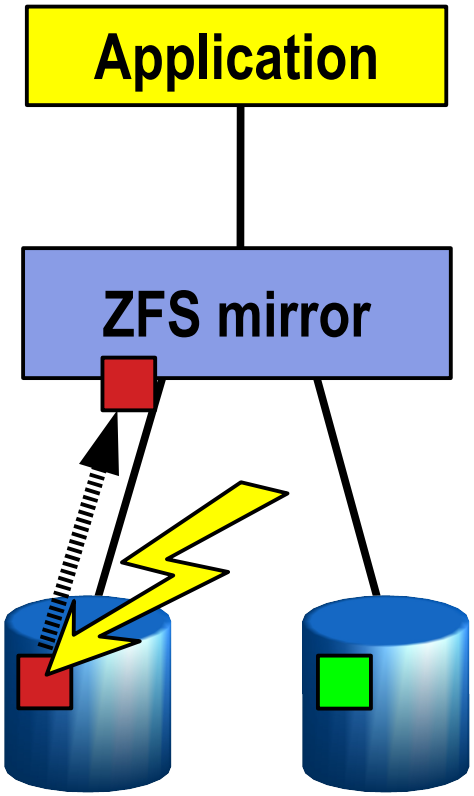


3. Filesystem returns bad data to the application.

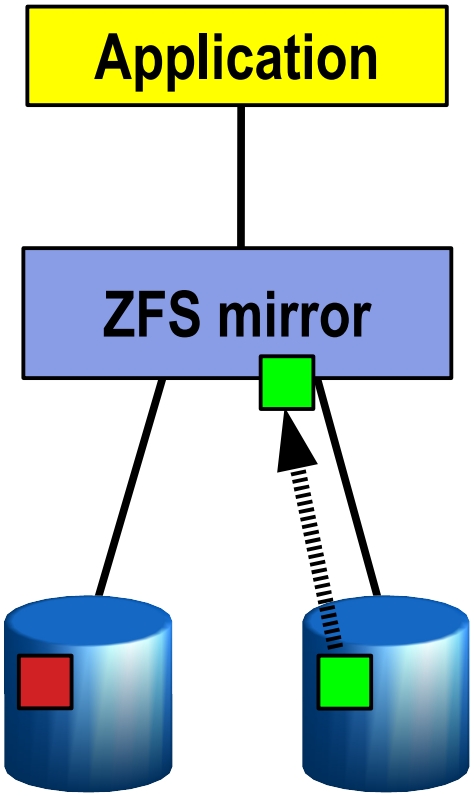


Self-Healing Data in ZFS

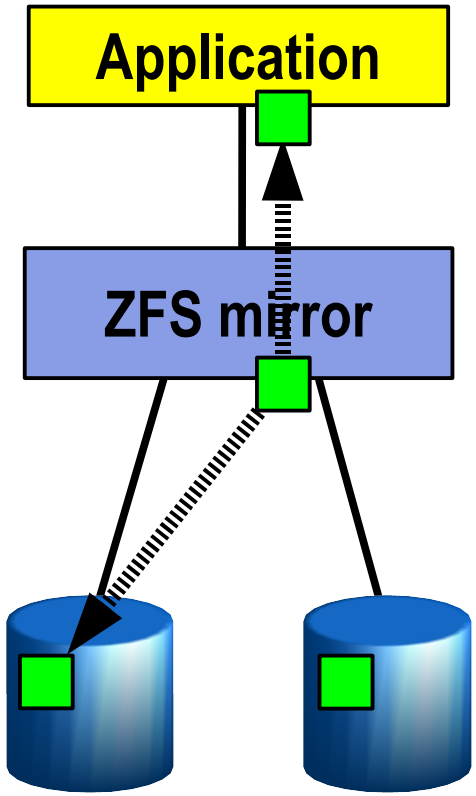
1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



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.



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

RAID-Z

- Dynamic stripe width
  - Variable block size: 512 – 128K
  - Each logical block is its own stripe
- All writes are full-stripe writes
  - Eliminates read-modify-write (it's fast)
  - Eliminates the RAID-5 write hole (no need for NVRAM)
- Both single- and double-parity
- 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