ZFS
The Last Word in Filesystem

lwhsu (2019-2020, CC BY)
tzute (2018)
? (?-2018)

交大資工系資訊中心
Computer Center of Department of Computer Science, NCTU

Copyright

Besides authors listed in the cover, this deck contains the slides from following people:

- Allan Jude <allanjude@FreeBSD.org>
    - ZFS history and OpenZFS
- Benedict Reuschling <bcr@FreeBSD.org>
    - ZFS introduction and zfs/zpool command usage
- Philip Paeps <philip@FreeBSD.org>
    - ZFS introduction and zfs/zpool command usage

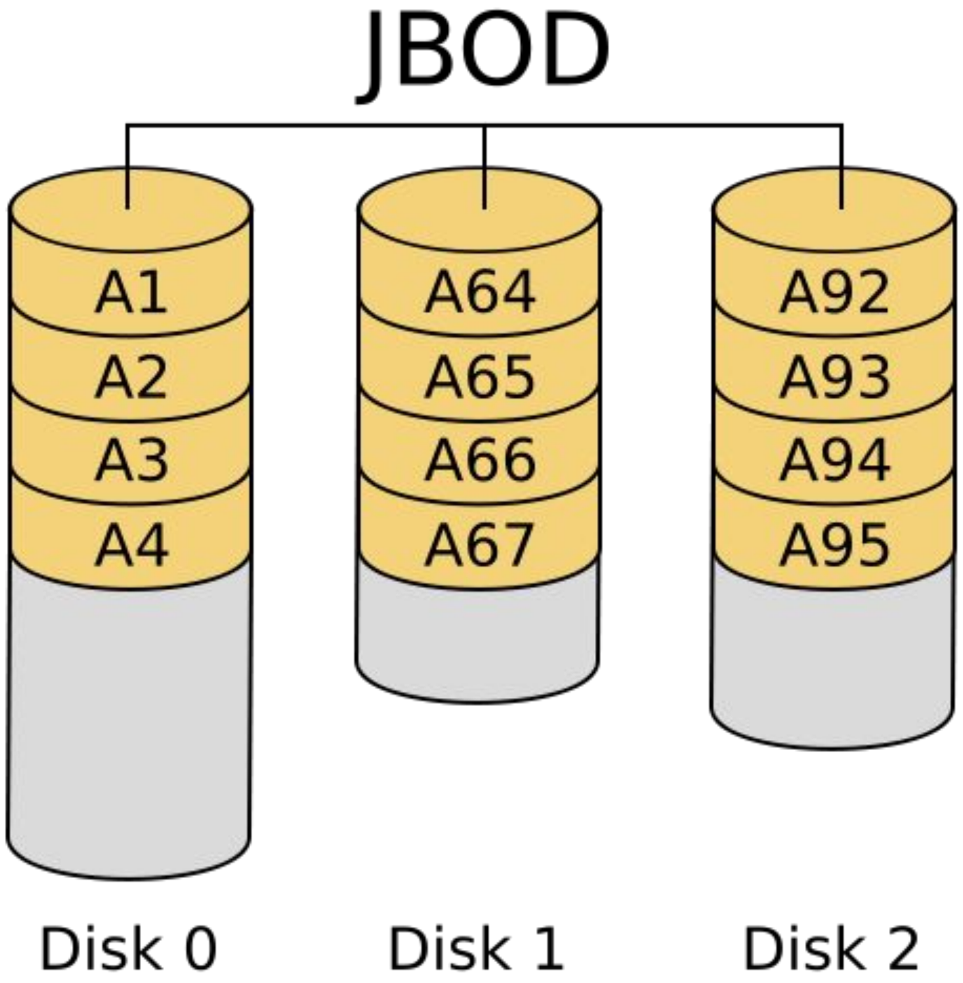
RAID

- Redundant Array of Independent Disks
    - Old name: Inexpensive
- A group of drives combined into one

Common RAID types

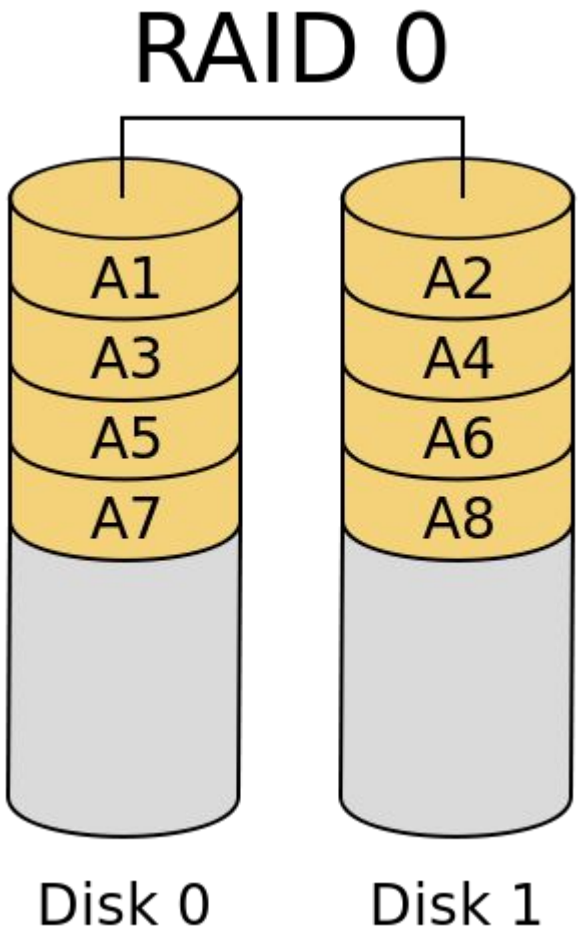
- JBOD
- RAID 0
- RAID 1
- RAID 5
- RAID 6
- RAID 10
- RAID 50
- RAID 60

JBOD (Just a Bunch Of Disks)



https://zh.wikipedia.org/zh-tw/RAID

RAID 0 (Stripe)

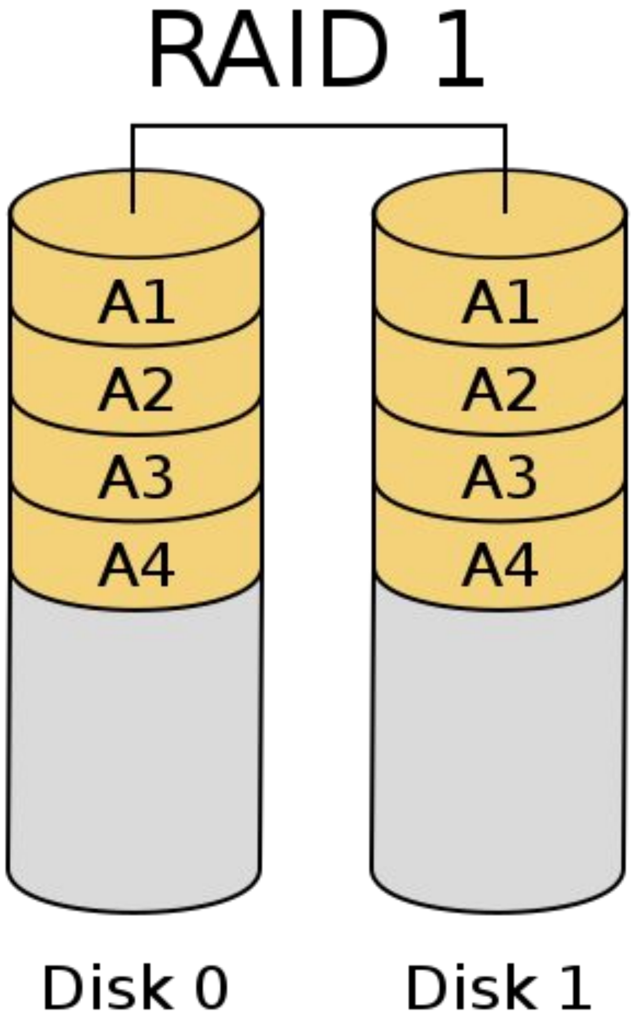


https://zh.wikipedia.org/zh-tw/RAID

RAID 0 (Stripe)

- Striping data onto multiple devices
- Increase write/read speed
- Data corrupt if ANY of the device fails

RAID 1 (Mirror)

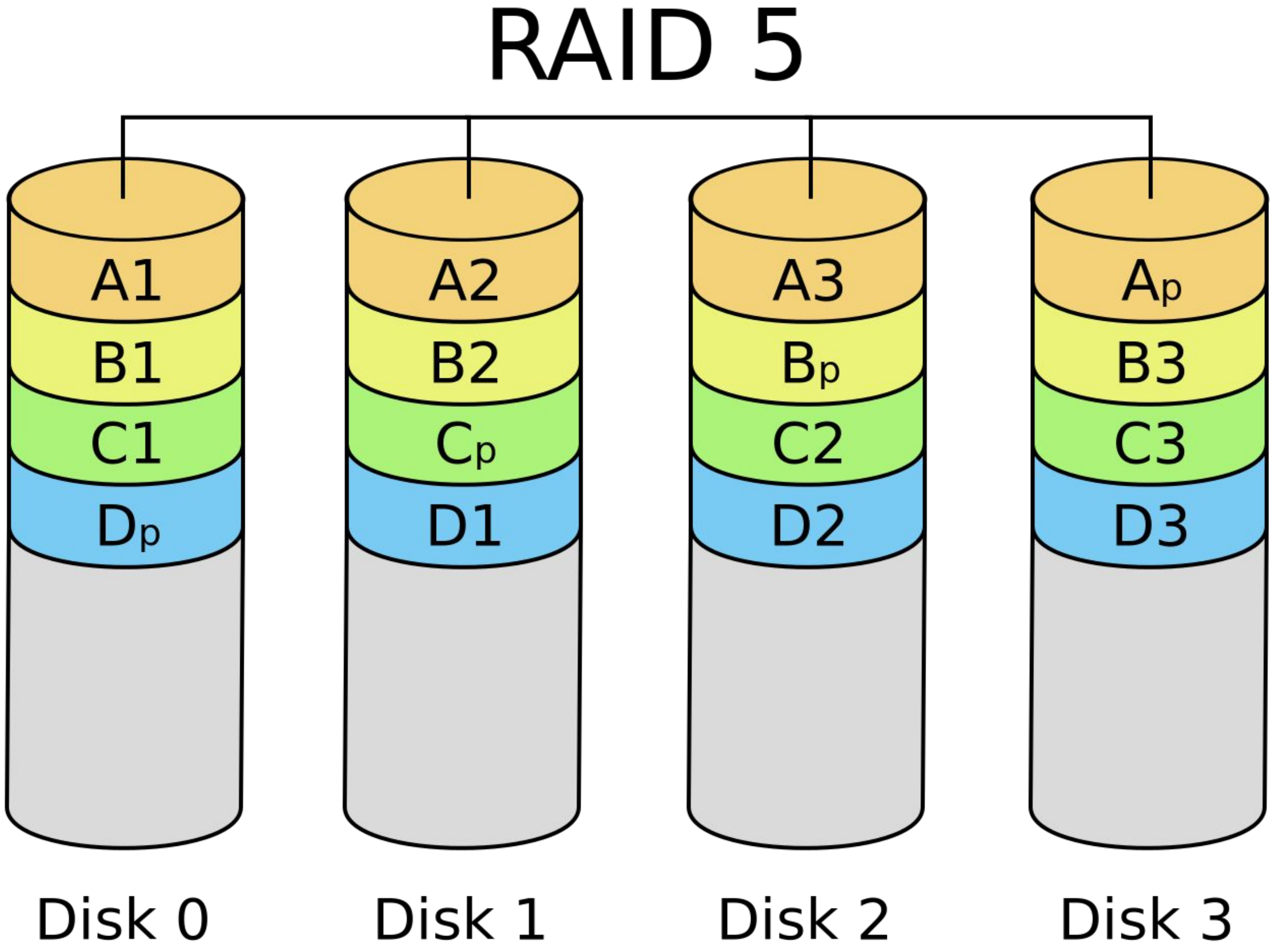


https://zh.wikipedia.org/zh-tw/RAID

RAID 1 (Mirror)

- Devices contain identical data
- 100% redundancy
- Faster read (but might be slower write)

RAID 5

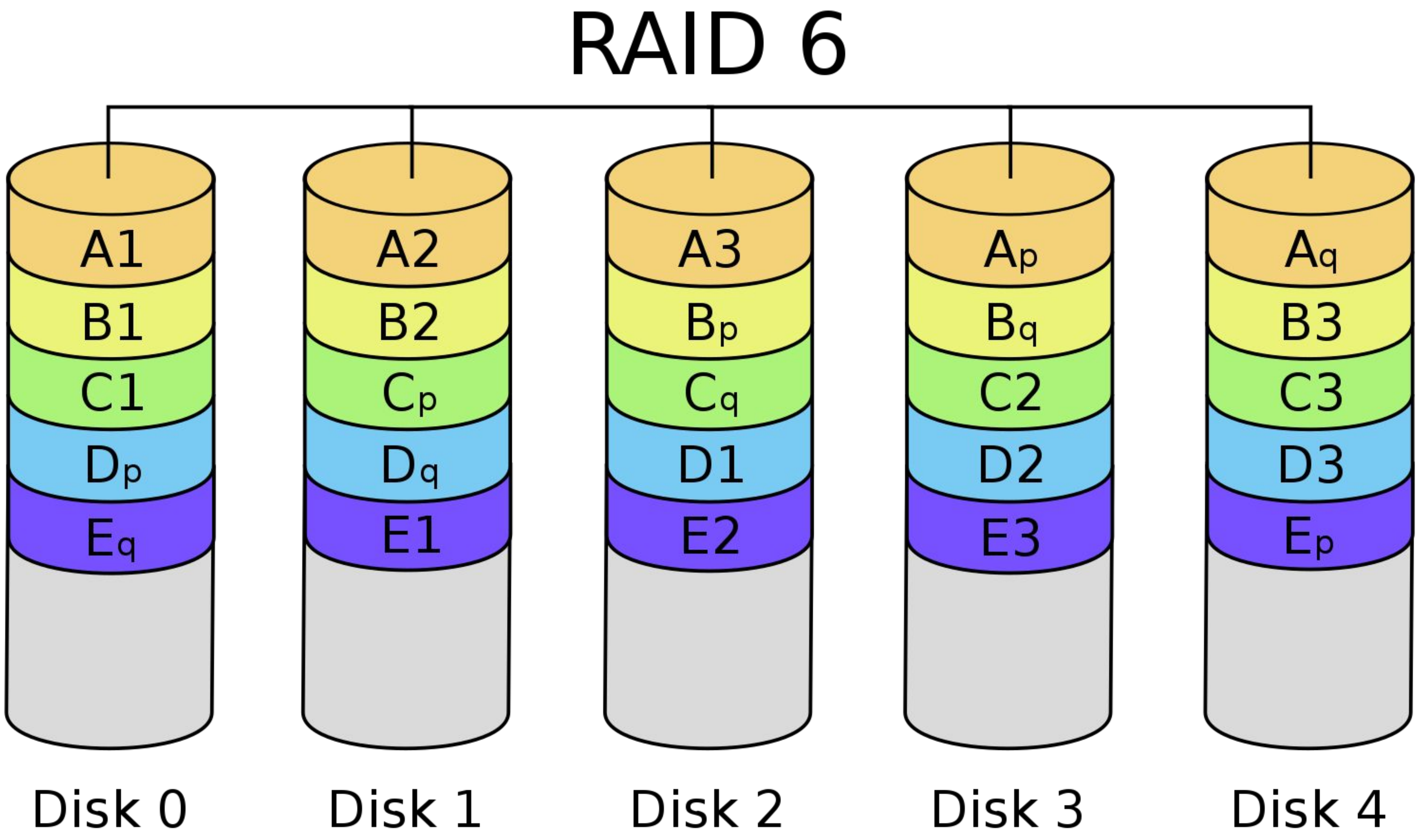


https://zh.wikipedia.org/zh-tw/RAID

RAID 5

- Slower than RAID 0 / RAID 1
- Higher CPU usage

RAID 6



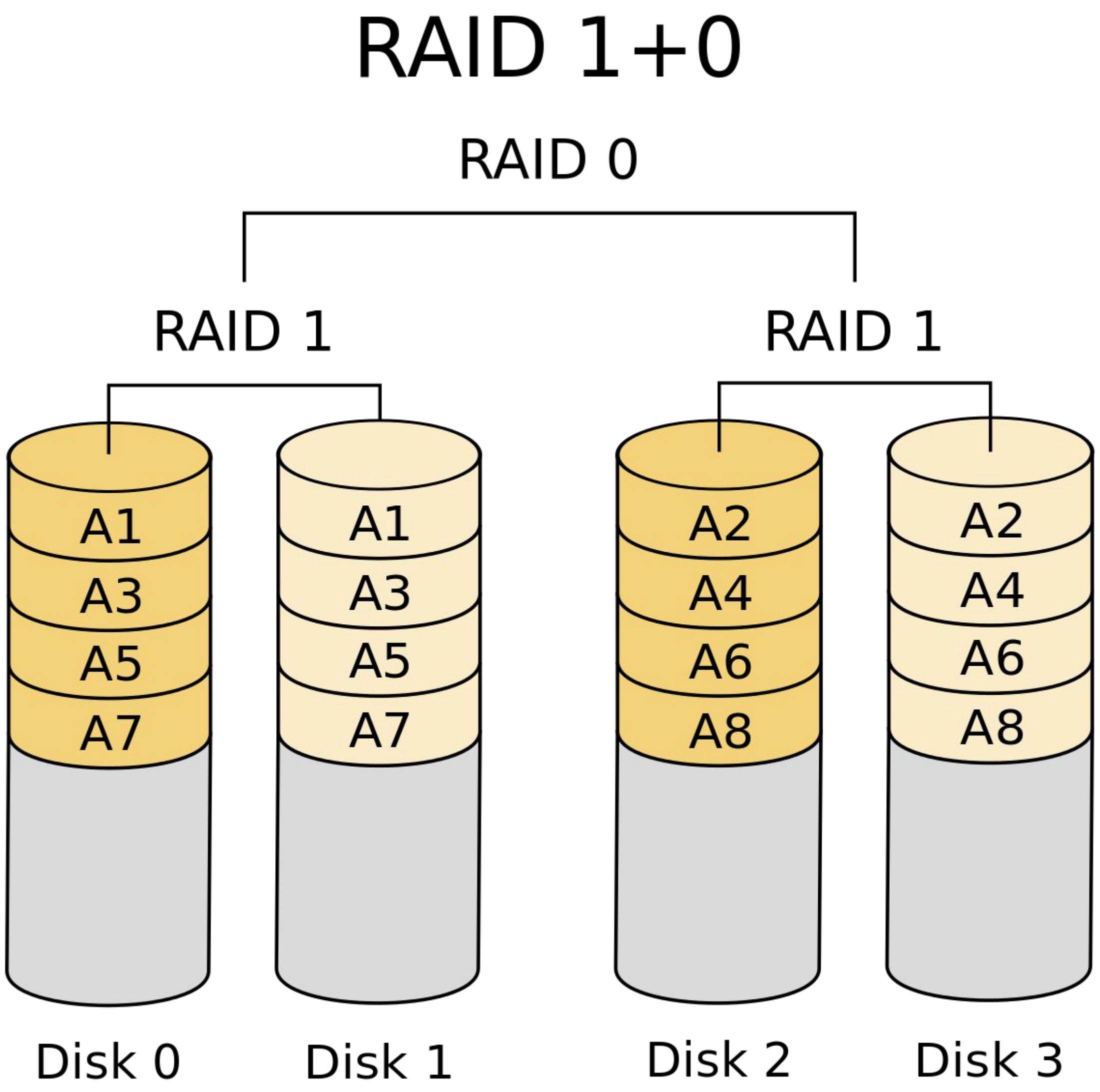
https://zh.wikipedia.org/zh-tw/RAID

RAID 6

- Slower than RAID 5
- Use two different correcting algorithms
- Usually implemented via hardware

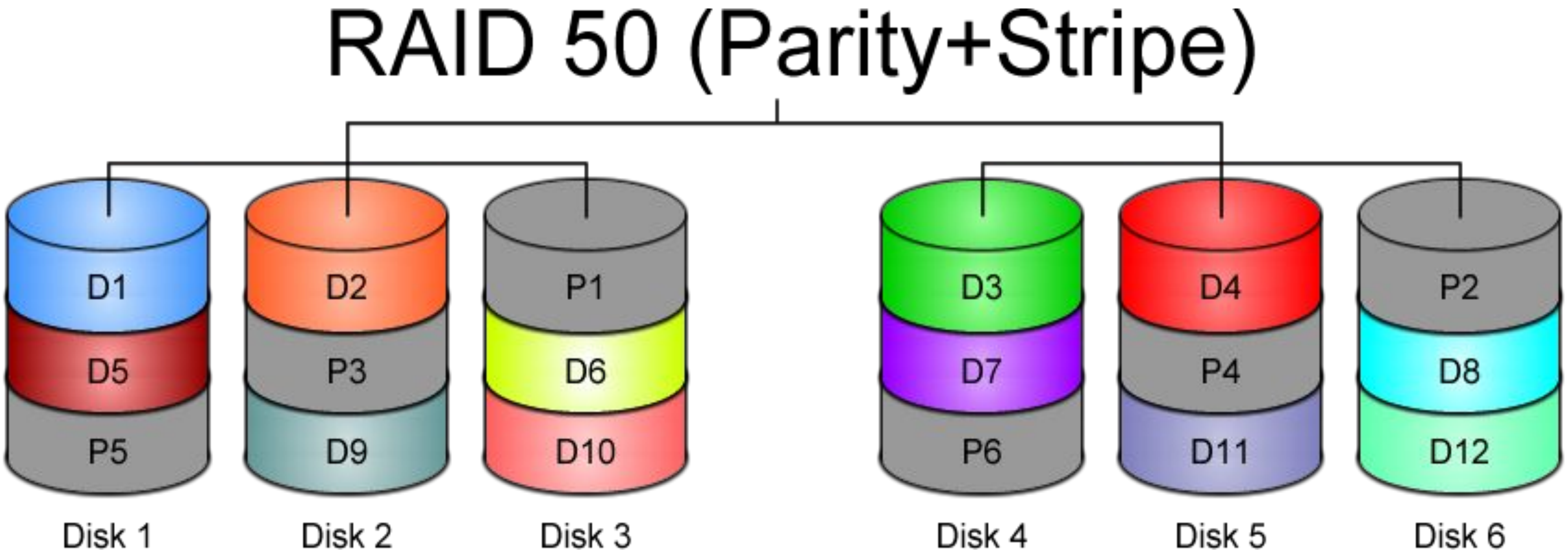
RAID 10

- RAID 1+0



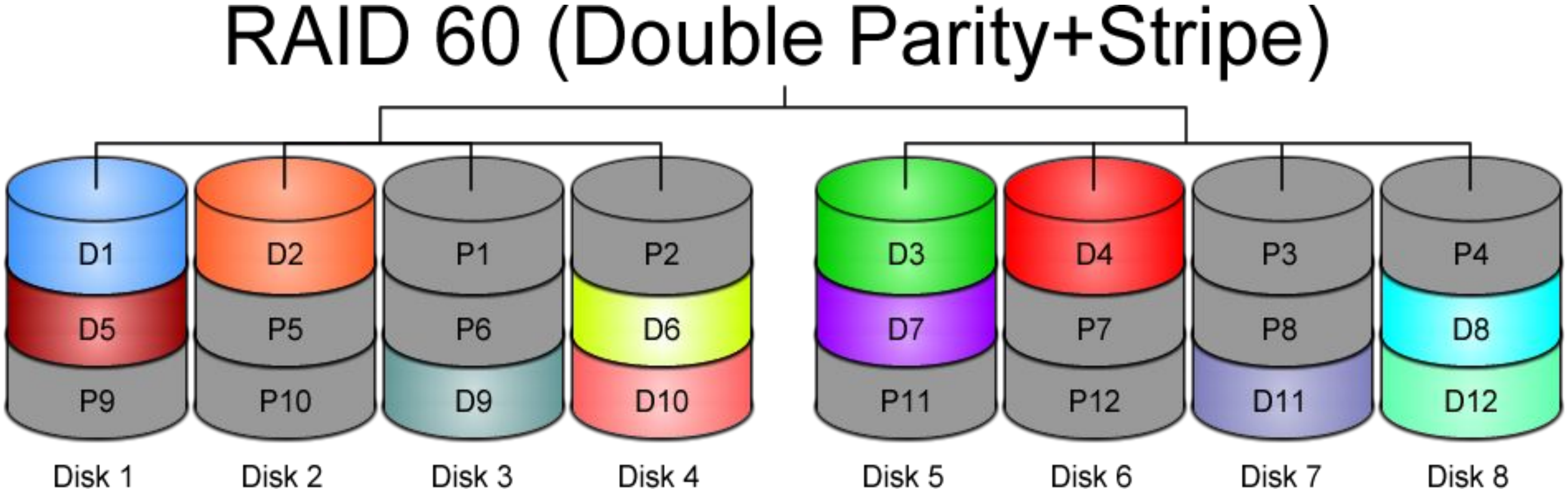
https://zh.wikipedia.org/zh-tw/RAID

RAID 50?



https://www.icc-usa.com/wp-content/themes/icc_solutions/images/raid-calculator/raid-50.png

RAID60?



https://www.icc-usa.com/wp-content/themes/icc_solutions/images/raid-calculator/raid-60.png

Issues of RAID

- https://en.wikipedia.org/wiki/RAID#Weaknesses
  - Correlated failures
    - Use different batches of drivers!
  - Unrecoverable read errors during rebuild
  - Increasing rebuild time and failure probability
  - Atomicity: including parity inconsistency due to system crashes
  - Write-cache reliability
- Know the limitations and make decision for your scenario

Software Implementations

- Linux – mdadm
- FreeBSD – GEOM classes

Here comes ZFS

Evolution of ZFS

- Originally developed at Sun Microsystems starting in 2001
- Open source under CDDL in 2005
- Oracle bought Sun in 2010, and close sourd further work
- illumos, a fork of the last open source version of (Open)Solaris became the new upstream for work on ZFS
- ZFS was ported to many platforms
    - FreeBSD 2007
    - Linux 2008
- The OpenZFS project founded to coordinate development across platforms

OpenZFS

- https://openzfs.org
- https://openzfs.github.io/openzfs-docs/
- https://github.com/openzfs/zfs
- All platforms can get the new feature faster
- OS dependent and OS independent codes in one repository
  - The old model (OS independent only) doesn't work well
- Working on standardize the command line interface where it has diverged across platforms
- More effort into effective naming of tunables (closer to user)

OpenZFS Platforms

- OpenZFS is now available on almost every platform
    - illumos (OmniOS, OpenIndiana, SmartOS, DilOS, Tribblix)
    - FreeBSD (FreeNAS, XigmaNAS, pfSense, etc.)
    - NetBSD
    - Linux
    - macOS
    - Windows
    - OSv

Why ZFS?

- Filesystem is always consistent
  - Never overwrite an existing block (transactional Copy-on-Write)
  - State atomically advance at checkpoints
  - Metadata redundancy and data checksums
- Snapshots (ro) and clones (rw) are cheap and plentiful
- Flexible configuration
  - Stripe, mirror, single/double/triple parity RAIDZ
- Fast remote replication and backups
- Scalable (the first 128 bit filesystem)
- SSD and memory friendly
- Easy administration (2 commands: zpool & zfs)

https://www.bsdcan.org/2015/schedule/events/525.en.html

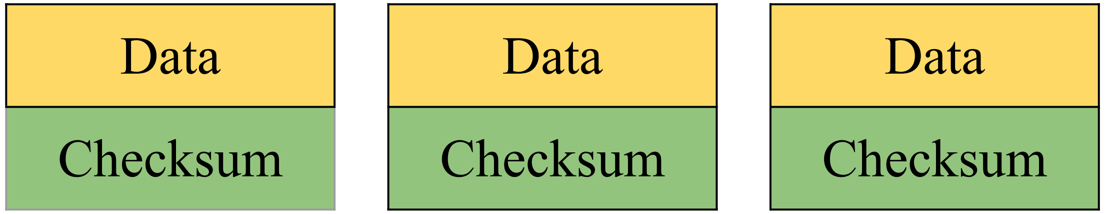
End-to-end data integrity

- Disks
- Controllers
- Cables
- Firmware
- Device drivers
- Non-ECC memory

Disk block checksums

- Checksums are stored with the data blocks
- Any self-consistent block will have a correct checksum
- Can't even detect stray writes
- Inherently limited to single file systems or volumes

Disk block checksums only validate media

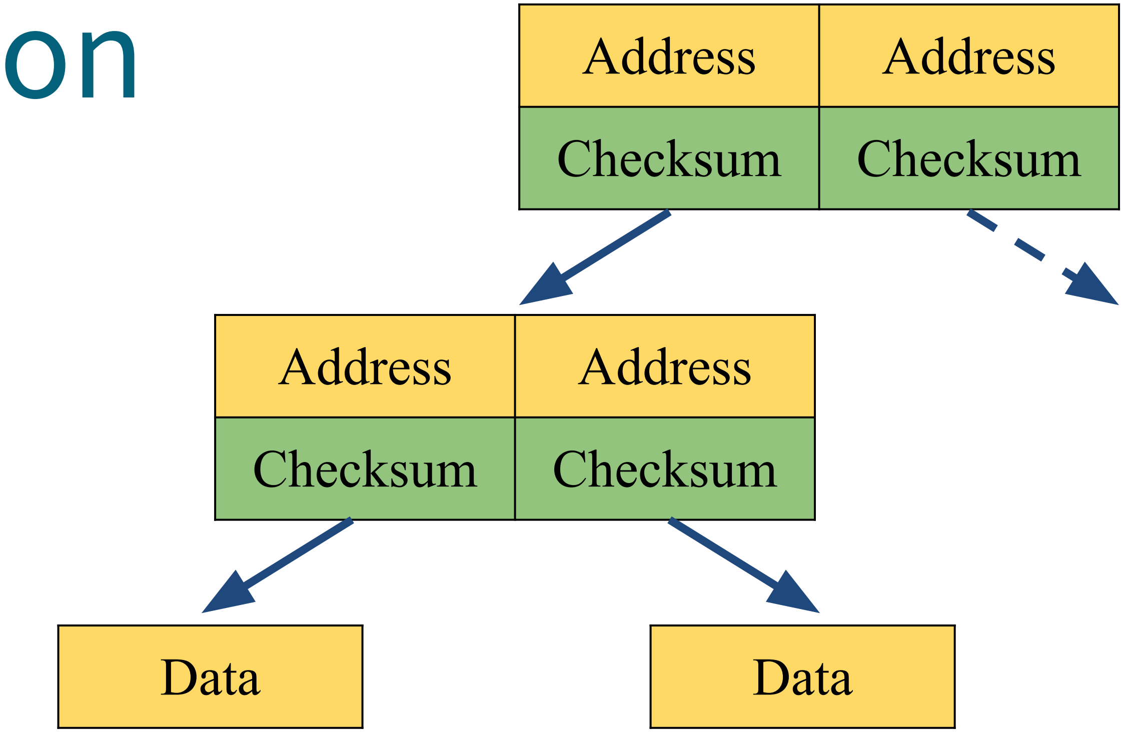


✓ Bit rot
- Phantom writes
- Misdirected reads and writes
- DMA parity errors
- Driver bugs
- Accidental overwrite

ZFS data authentication

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

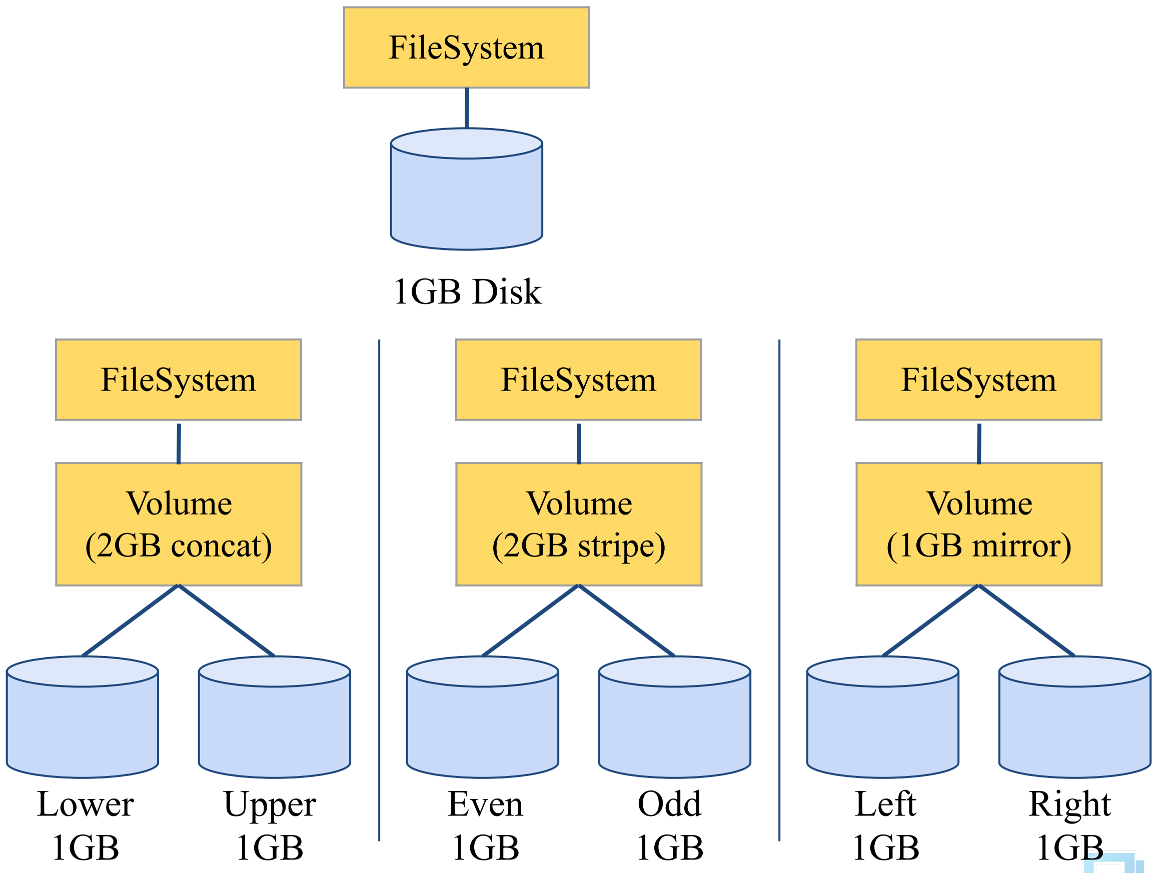
ZFS data authentication validates entire I/O path



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

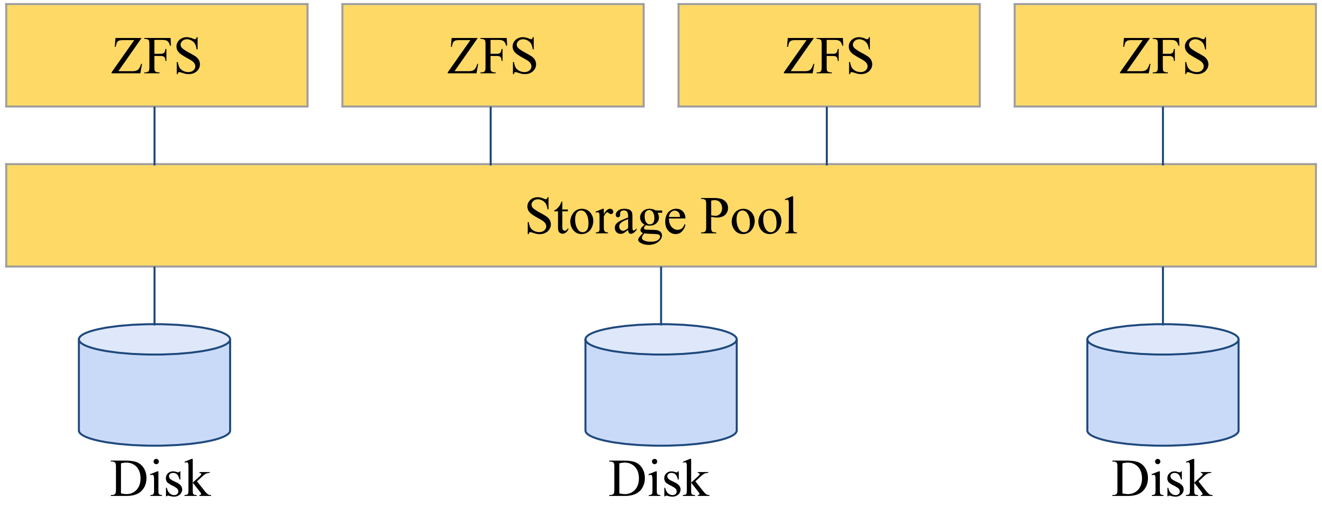
Traditional storage architecture

- Single partition or volume per filesystem
- Each filesystem has limited I/O bandwidth
- Filesystems must be manually resized
- Storage is fragmented

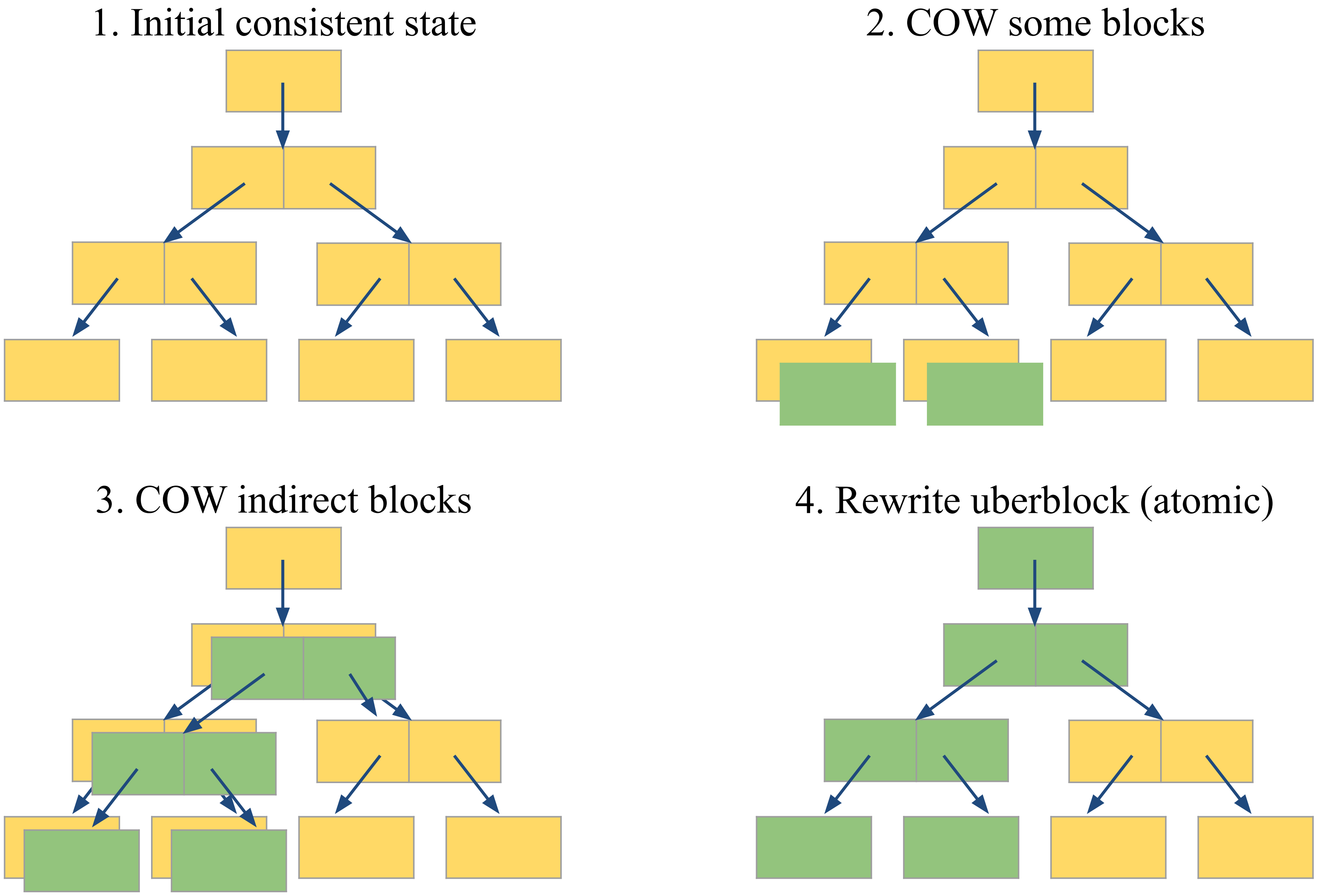


ZFS pooled storage

- No partitions required
- Storage pool grows automatically
- All I/O bandwidth is always available
- All storage in the pool is shared



Copy-on-write transactions



1. Initial consistent state

2. COW some blocks

3. COW indirect blocks

4. Rewrite uberblock (atomic)

Simple administration

- **Only two commands:**
  - Storage pools: **zpool**
    - Add and replace disks
    - Resize pools
  - Filesystems: **zfs**
    - Quotas, reservations, etc.
    - Compression and deduplication
    - Snapshots and clones
    - atime, readonly, etc.

Storage Pools

ZFS Pool

- ZFS is not just a filesystem
- ZFS = filesystem + volume manager
- Works out of the box

- "Z"uper "Z"imple to create
- Controlled with single command
    - zpool
- zpool(8)
- zpoolconcepts(8)

ZFS Pools Components

- Pool is create from “Virtual Devices” (vdevs)
- **disk**: A real disk (typically under /dev)
- **file**: A file
- **mirror**: Two or more disks mirrored together
- **raidz1/2/3**: Three or more disks in RAID5/6*
- **spare**: A spare drive
- **log**: A write log device (ZIL SLOG; typically SSD)
- **cache**: A read cache device (L2ARC; typically SSD)

RAID in ZFS

- **Dynamic Stripe**: Intelligent RAID 0
  - zfs copies=1 | 2 | 3
- **Mirror**: RAID 1
- **Raidz1**: Improved from RAID5 (parity)
- **Raidz2**: Improved from RAID6 (double parity)
- **Raidz3**: Triple parity

Storage pools

Creating storage pools (1/2)

- To create a storage pool named "tank" from a single disk:
  - zpool create tank /dev/md0
    - ZFS can use disks directly. There is no need to create partitions or volumes.
- After creating a storage pool, ZFS will automatically:
  - Create a filesystem with the same name (e.g. tank)
  - Mount the filesystem under that name (e.g. **/tank**)
- **The storage is immediately available**

Storage pools

Creating storage pools (2/2)

- All configuration is stored with the storage pool and persists across reboots.
- No need to edit /etc/fstab.

```
# mount | grep tank
# ls -al /tank
ls: /tank: No such file or directory
# zpool create tank /dev/md0
# mount | grep tank
tank on /tank (zfs, local, nfsv4acls)
# ls -al /tank
total 9
drwxr-xr-x   2 root  wheel   2 Oct 12 12:17 .
drwxr-xr-x  23 root  wheel  28 Oct 12 12:17 ..
# reboot
[...]
# mount | grep tank
tank on /tank (zfs, local, nfsv4acls)
```

Storage pools

Displaying pool status

```
# zpool list
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH
ALTROOT
tank  1016G    83K  1016G        -         -    0%     0%  1.00x  ONLINE  -

# zpool status
  pool: tank
 state: ONLINE
  scan: none requested
config:

    NAME        STATE     READ WRITE CKSUM
    tank        ONLINE       0     0     0
      md0       ONLINE       0     0     0

errors: No known data errors
```

Storage pools
Displaying I/O statistics

- ZFS contains a built-in tool to display I/O statistics.
- Given an interval in seconds, statistics will be displayed continuously until the user interrupts with **Ctrl+C**.
- Use -v (verbose) to display more detailed statistics.

```
# zpool iostat 5
               capacity     operations     bandwidth
pool         alloc   free   read  write   read  write
----------   -----  -----  -----  -----  -----  -----
tank           83K  1016G      0      0    234    841
tank           83K  1016G      0      0      0      0

# zpool iostat -v
               capacity     operations     bandwidth
pool         alloc   free   read  write   read  write
----------   -----  -----  -----  -----  -----  -----
tank           83K  1016G      0      0    206    739
  md0          83K  1016G      0      0    206    739
----------   -----  -----  -----  -----  -----  -----
```

Storage pools

Destroying storage pools

- Destroying storage pools is a constant time operation. If you want to get rid of your data, ZFS will help you do it very quickly!
- All data on a destroyed pool will be **irretrievably lost**.

```
# time zpool create tank /dev/md0
    0.06 real   0.00 user   0.02 sys

# time zpool destroy tank
    0.09 real   0.00 user   0.00 sys
```

Storage pools

Creating stripes

- A pool with just one disk does not provide any redundancy, capacity or even adequate performance.
- Stripes offer higher capacity and better performance (reading will be parallelised) but they provide **no redundancy**.

```
# zpool create tank /dev/md0 /dev/md1
# zpool status
  pool: tank
 state: ONLINE
  scan: none requested
config:

    NAME        STATE     READ WRITE CKSUM
    tank        ONLINE       0     0     0
      md0       ONLINE       0     0     0
      md1       ONLINE       0     0     0

errors: No known data errors

# zpool list
NAME    SIZE  ALLOC   FREE  CAP  DEDUP  HEALTH
tank   1.98T    86K  1.98T   0%  1.00x  ONLINE
```

Storage pools
Creating mirrors (RAID-1)

- Mirrored storage pools provide **redundancy** against disk failures and better read performance than single-disk pools.
- However, mirrors only have **50% of the capacity** of the underlying disks.

```
# zpool create tank mirror /dev/md0 /dev/md1
# zpool status
  pool: tank
 state: ONLINE
  scan: none requested
config:

       NAME        STATE     READ WRITE CKSUM
       tank        ONLINE       0     0     0
         mirror-0  ONLINE       0     0     0
           md0     ONLINE       0     0     0
           md1     ONLINE       0     0     0

errors: No known data errors
# zpool list
NAME   SIZE  ALLOC   FREE CAP  DEDUP  HEALTH
tank  1016G    93K 1016G  0%  1.00x  ONLINE
```

Storage pools

Creating raidz groups

- raidz is a variation on RAID-5 with single-, double-, or triple parity.
- A raidz group with N disks of size X with P parity disks can hold approximately $(N - P) * X$ bytes and can withstand P device(s) failing before data integrity is compromised.

```
# zpool create tank \
> raidz1 /dev/md0 /dev/md1 /dev/md2 /dev/md3
# zpool status
 pool: tank
 state: ONLINE
  scan: none requested
config:

    NAME        STATE     READ WRITE CKSUM
    tank        ONLINE       0     0     0
      raidz1-0  ONLINE       0     0     0
        md0     ONLINE       0     0     0
        md1     ONLINE       0     0     0
        md2     ONLINE       0     0     0
        md3     ONLINE       0     0     0

errors: No known data errors
```

Storage pools

Combining vdev types

- Single disks, stripes, mirrors and raidz groups can be combined in a single storage pool
- ZFS will complain when adding devices would make the pool less redundant
- zpool add log/cache/spare

```
# zpool create tank mirror /dev/md0 /dev/md1
# zpool add tank /dev/md2
invalid vdev specification
use '-f' to override the following errors:
mismatched replication level:
pool uses mirror and new vdev is disk

# zpool create tank \
> raidz2 /dev/md0 /dev/md1 /dev/md2 /dev/md3

# zpool add tank \
> raidz /dev/md4 /dev/md5 /dev/md6
invalid vdev specification
use '-f' to override the following errors:
mismatched replication level:
pool uses 2 device parity and new vdev uses 1
```

Storage pools

Increasing storage pool capacity

- More devices can be added to a storage pool to increase capacity without downtime.
- Data will be striped across the disks, increasing performance, but there will be **no redundancy**.
- If any disk fails, **all data is lost!**

```
# zpool create tank /dev/md0
# zpool add tank /dev/md1
# zpool list
NAME    SIZE  ALLOC   FREE CAP  DEDUP  HEALTH
tank   1.98T   233K  1.98T  0%  1.00x  ONLINE
# zpool status
  pool: tank
 state: ONLINE
  scan: none requested
config:

	NAME        STATE     READ WRITE CKSUM
	tank        ONLINE       0     0     0
	  md0       ONLINE       0     0     0
	  md1       ONLINE       0     0     0

errors: No known data errors
```

Storage pools

Creating a mirror from a single-disk pool (1/4)

- A storage pool consisting of only one device can be converted to a mirror.
- In order for the new device to mirror the data of the already existing device, the pool needs to be “resilvered”.
- This means that the pool synchronises both devices to contain the same data at the end of the resilver operation.
- During resilvering, access to the pool will be slower, but there will be no downtime.

Storage pools

Creating a mirror from a single-disk pool (2/4)

```
# zpool create tank /dev/md0
# zpool status
  pool: tank
 state: ONLINE
  scan: none requested
config:

    NAME        STATE     READ WRITE CKSUM
    tank        ONLINE       0     0     0
      md0       ONLINE       0     0     0

errors: No known data errors

# zpool list
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank  1016G    93K  1016G        -         -     0%     0%  1.00x  ONLINE  -
```

Storage pools

Creating a mirror from a single-disk pool (3/4)

- zpool attach

```
# zpool create tank /dev/md0
# zpool status
  pool: tank
 state: ONLINE
  scan: none requested
config:

    NAME        STATE     READ WRITE CKSUM
    tank        ONLINE       0     0     0
      md0       ONLINE       0     0     0

errors: No known data errors

# zpool list
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank  1016G    93K  1016G        -         -     0%     0%  1.00x  ONLINE  -
```

Storage pools

Creating a mirror from a single-disk pool (4/4)

```
# zpool status
  pool: tank
 state: ONLINE
  scan: resilvered 44.2M in 0h1m with 0 errors on Fri Oct 12 13:56:29 2018
config:

    NAME        STATE     READ WRITE CKSUM
    tank        ONLINE       0     0     0
      mirror-0  ONLINE       0     0     0
        md0     ONLINE       0     0     0
        md1     ONLINE       0     0     0

errors: No known data errors

# zpool list
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank  1016G  99.5K  1016G        -         -     0%     0%  1.00x  ONLINE  -
```

Zpool command

- zpool(8)
    - zpool list
        - list all the zpool
    - zpool status [pool name]
        - show status of zpool
    - zpool export/import [pool name]
        - export or import given pool
    - zpool set/get <properties/all>
        - set or show zpool properties
    - zpool online/offline <pool name> <vdev>
        - set an device in zpool to online/offline state
    - zpool attach/detach <pool name> <device> <new device>
        - attach a new device to an zpool/detach a device from zpool
    - zpool replace <pool name> <old device> <new device>
        - replace old device with new device
    - zpool scrub
        - try to discover silent error or hardware failure
    - zpool history [pool name]
        - show all the history of zpool
    - zpool add <pool name> <vdev>
        - add additional capacity into pool
    - zpool create/destroy
        - create/destory zpool

Zpool properties

```
# zpool get all zroot
NAME   PROPERTY                       VALUE                          SOURCE
zroot  size                           460G                           -
zroot  capacity                       4%                             -
zroot  altroot                        -                              default
zroot  health                         ONLINE                         -
zroot  guid                           1306392864376526758 5           default
zroot  version                        -                              default
zroot  bootfs                         zroot/ROOT/default             local
zroot  delegation                     on                             default
zroot  autoreplace                    off                            default
zroot  cachefile                      -                              default
zroot  failmode                       wait                           default
zroot  listsnapshots                  off                            default
zroot  feature@async_destroy          enabled                        local
zroot  feature@device_removal         enabled                        local
```

Zpool Sizing

- ZFS reserve 1/64 of pool capacity for safe-guard to protect CoW

- RAIDZ1 Space = Total Drive Capacity -1 Drive
- RAIDZ2 Space = Total Drive Capacity -2 Drives
- RAIDZ3 Space = Total Drive Capacity -3 Drives
- Dynamic Stripe of 4* 100GB= 400 / 1.016= ~390GB
- RAIDZ1 of 4* 100GB = 300GB - 1/64th= ~295GB
- RAIDZ2 of 4* 100GB = 200GB - 1/64th= ~195GB
- RAIDZ2 of 10* 100GB = 800GB - 1/64th= ~780GB

http://cuddletech.com/blog/pivot/entry.php?id=1013

ZFS Dataset

ZFS Datasets

- Three forms:
  - filesystem: just like traditional filesystem
  - volume: block device
  - snapshot: read-only version of a file system or volume at a given point of time.
- Nested
- Each dataset has associated properties that can be inherited by sub-filesystems
- Controlled with single command:
  - zfs(8)

Filesystem Datasets

- Create new dataset with
  - zfs create <pool name>/<dataset name>(/<dataset name>/…)
- New dataset inherits properties of parent dataset

Volume Datasets (ZVols)

- Block storage
- Located at /dev/zvol/<pool name>/<dataset>
- Useful for
  - iSCSI
  - Other non-zfs local filesystem
  - Virtual Machine image
- Support "thin provisioning" ("sparse volume")

Dataset properties

```
$ zfs get all zroot
NAME   PROPERTY               VALUE                    SOURCE
zroot  type                   filesystem                  -
zroot  creation               Mon Jul 21 23:13 2014       -
zroot  used                   22.6G                       -
zroot  available              423G                        -
zroot  referenced             144K                        -
zroot  compressratio          1.07x                       -
zroot  mounted                no                          -
zroot  quota                  none                     default
zroot  reservation            none                     default
zroot  recordsize             128K                     default
zroot  mountpoint             none                     local
zroot  sharenfs               off                      default
```

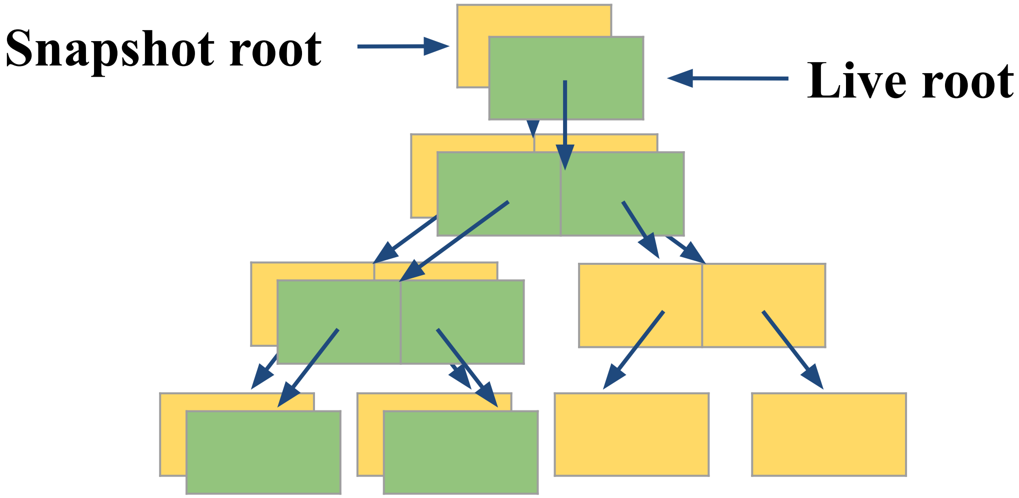
zfs command

- zfs(8)
  - zfs set/get <prop. / all> <dataset>
    - set properties of datasets
  - zfs create <dataset>
    - create new dataset
  - zfs destroy
    - destroy datasets/snapshots/clones..
  - zfs snapshot
    - create snapshots
  - zfs rollback
    - rollback to given snapshot
  - zfs promote
    - promote clone to the orgin of the filesystem
  - zfs send/receive
    - send/receive data stream of the snapshot

Snapshots

Snapshot

- Read-only copy of a dataset or volume
- Useful for file recovery or full dataset rollback
- Denoted by @ symbol
- Snapshots are extremely fast (-er than deleting data!)
- Snapshots occupy (almost) no space until the original data start to diverge
- How ZFS snapshots really work (Matt Ahrens)
  - https://www.bsdcan.org/2019/schedule/events/1073.en.html



Snapshots

Creating and listing snapshots (1/2)

- A snapshot only needs an identifier
  - Can be anything you like!
  - A timestamp is traditional
  - But you can use more memorable identifiers too…

```
# zfs snapshot tank/users/alice@myfirstbackup
# zfs list -t snapshot
NAME                              USED  AVAIL  REFER  MOUNTPOINT
tank/users/alice@myfirstbackup       0      -    23K  -

# zfs list -rt all tank/users/alice
NAME                              USED  AVAIL  REFER  MOUNTPOINT
tank/users/alice                   23K   984G    23K  /tank/users/alice
tank/users/alice@myfirstbackup       0      -    23K  -
```

Snapshots

Creating and listing snapshots (2/2)

- Snapshots save only the changes between the time they were created and the previous (if any) snapshot
- If data doesn’t change, snapshots occupy zero space

```
# echo hello world > /tank/users/alice/important_data.txt
# zfs snapshot tank/users/alice@mysecondbackup
# zfs list -rt all tank/users/alice
NAME                              USED  AVAIL  REFER  MOUNTPOINT
tank/users/alice                 36.5K   984G  23.5K  /tank/users/alice
tank/users/alice@myfirstbackup     13K      -    23K  -
tank/users/alice@mysecondbackup      0      -  23.5K  -
```

Snapshots

Differences between snapshots

- ZFS can display the differences between snapshots

```
# touch /tank/users/alice/empty
# rm /tank/users/alice/important_data.txt
# zfs diff tank/users/alice@mysecondbackup
M   /tank/users/alice/
-   /tank/users/alice/important_data.txt
+   /tank/users/alice/empty
```

Character	Type of change
+	File was added
-	File was deleted
M	File was modified
R	File was renamed

Snapshots

Rolling back snapshots (1/2)

- Snapshots can be rolled back to undo changes
- All files changed since the snapshot was created will be discarded

```
# echo hello_world > important_file.txt
# echo goodbye_cruel_world > also_important.txt
# zfs snapshot tank/users/alice@myfirstbackup

# rm *
# ls
# zfs rollback tank/users/alice@myfirstbackup
# ls
also_important.txt  important_file.txt
```

Snapshots

Rolling back snapshots (2/2)

- By default, the latest snapshot is rolled back. To roll back an older snapshot, use -r
- Note that intermediate snapshots will be destroyed
- ZFS will warn about this

```
# touch not_very_important.txt
# touch also_not_important.txt
# ls
also_important.txt       important_file.txt
also_not_important.txt  not_very_important.txt

# zfs snapshot tank/users/alice@mysecondbackup
# zfs diff tank/users/alice@myfirstbackup \
> tank/users/alice@mysecondbackup
M    /tank/users/alice/
+    /tank/users/alice/not_very_important.txt
+    /tank/users/alice/also_not_important.txt

# zfs rollback tank/users/alice@myfirstbackup
# zfs rollback -r tank/users/alice@myfirstbackup
# ls
also_important.txt  important_file.txt
```

Snapshots
Restoring individual files

- Sometimes, we only want to restore a single file, rather than rolling back an entire snapshot
- ZFS keeps snapshots in a very hidden .zfs/snapshots directory
  - It's like magic :-)
  - Set snapdir=visible to unhide it
- Remember: snapshots are read-only. Copying data to the magic directory won't work!

```
# ls
also_important.txt
important_file.txt

# rm *
# ls

# ls .zfs/snapshot/myfirstbackup
also_important.txt
important_file.txt

# cp .zfs/snapshot/myfirstbackup/* .

# ls
also_important.txt
important_file.txt
```

Snapshots

Cloning snapshots

- Clones represent a writeable copy of a read-only snapshot
- Like snapshots, they occupy no space until they start to diverge

```
# zfs list -rt all tank/users/alice
NAME                              USED  AVAIL  REFER  MOUNTPOINT
tank/users/alice                  189M   984G   105M  /tank/users/alice
tank/users/alice@mysecondbackup      0      -   105M  -

# zfs clone tank/users/alice@mysecondbackup tank/users/eve

# zfs list tank/users/eve
NAME               USED  AVAIL  REFER  MOUNTPOINT
tank/users/eve        0   984G   105M  /tank/users/eve
```

Snapshots

Promoting clones

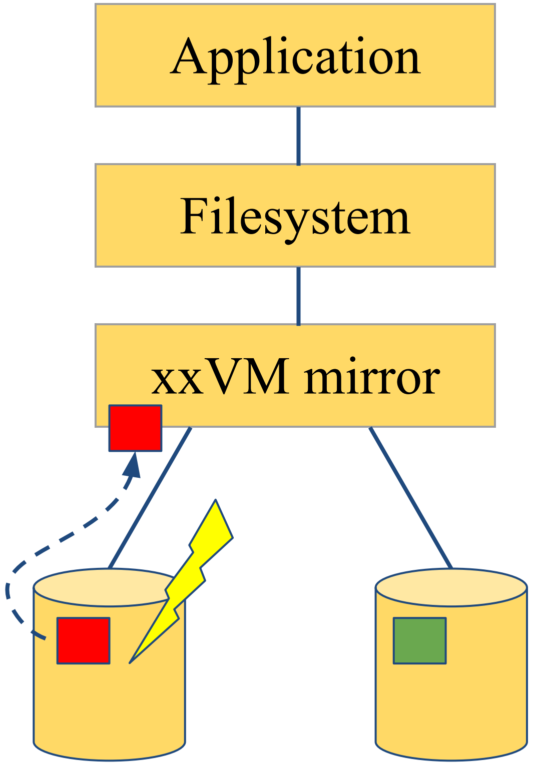
- Snapshots cannot be deleted while clones exist
- To remove this dependency, clones can be promoted to ”ordinary” datasets
- Note that by promoting the clone, it immediately starts occupying space

```
# zfs destroy tank/users/alice@mysecondbackup
cannot destroy 'tank/users/alice@mysecondbackup’:
snapshot has dependent clones
use '-R' to destroy the following datasets:
tank/users/eve

# zfs list tank/users/eve
NAME                 USED  AVAIL  REFER  MOUNTPOINT
tank/users/eve          0   984G   105M  /tank/users/eve

# zfs promote tank/users/eve

# zfs list tank/users/eve
NAME                 USED  AVAIL  REFER  MOUNTPOINT
tank/users/eve       189M   984G   105M  /tank/users/eve
```

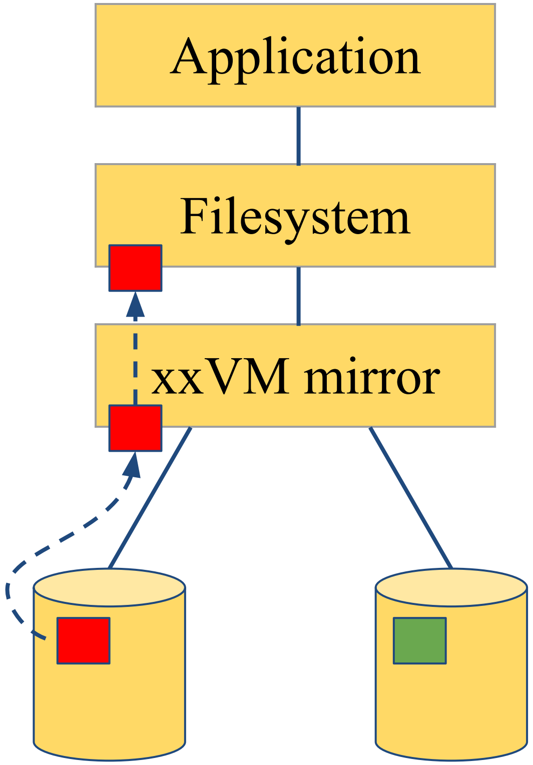
Self-healing data

Traditional mirroring

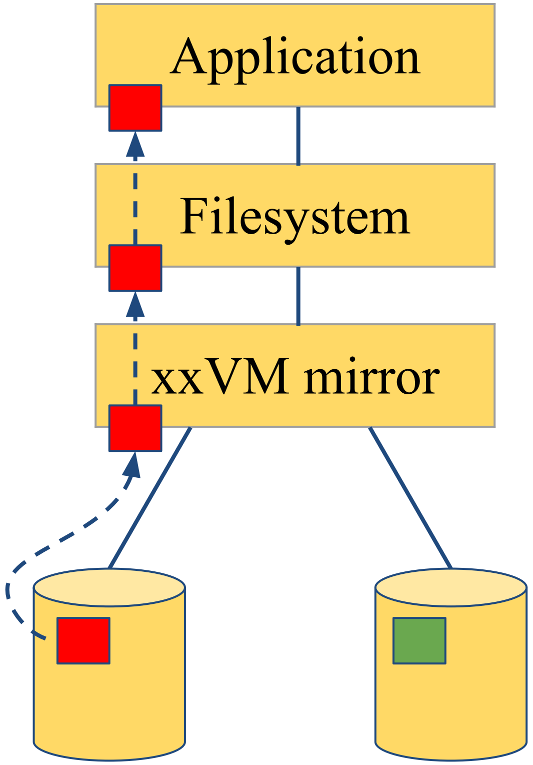
1. Application issue a read. Mirror reads the first disk, which has a corrupt block. It can’t tell



2. Volume manager passed bas block up to filesystem. If it’s a metadata block, the filesystem panics. If not...

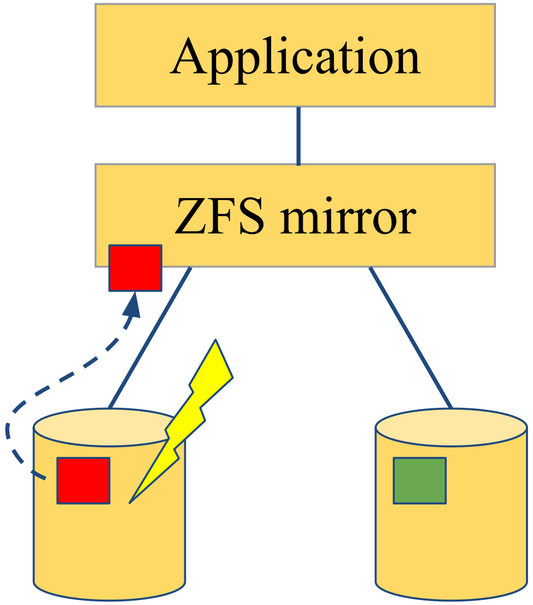


3. Filesystem returns bad data to the application

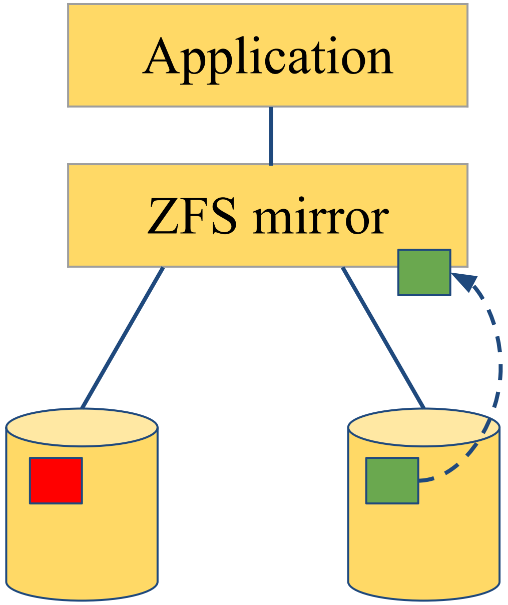


Self-healing data in ZFS

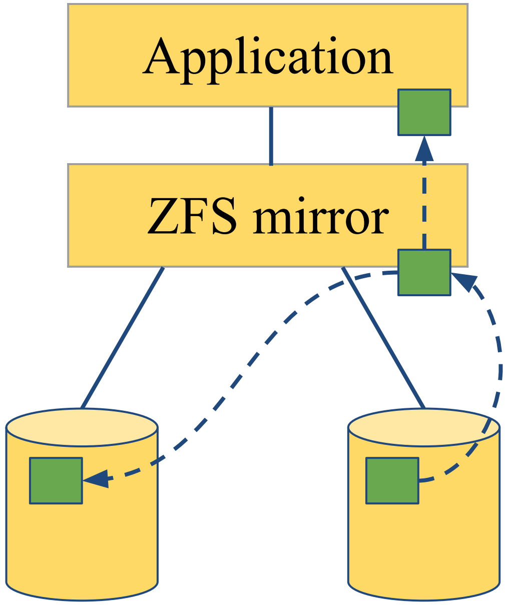
1. Application issue a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



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** on the first disk.



Self-healing data demo
Store some important data (1/2)

- We have created a redundant pool with two mirrored disks and stored some important data on it
- We will be very sad if the data gets lost! :-(

```
# zfs list tank
NAME    USED  AVAIL  REFER  MOUNTPOINT
tank     74K   984G    23K  /tank

# cp -a /some/important/data/ /tank/

# zfs list tank
NAME    USED  AVAIL  REFER  MOUNTPOINT
tank   3.23G   981G  3.23G  /tank
```

Self-healing data demo

Store some important data (2/2)

```
# zpool status tank
  pool: tank
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            md0     ONLINE       0     0     0
            md1     ONLINE       0     0     0

errors: No known data errors

# zpool list tank
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank  1016G  3.51G  1012G        -         -     0%     0%  1.00x  ONLINE  -
```

Self-healing data demo
Destroy one of the disks (1/2)

Caution!

This example can destroy data when used on the wrong device or a non-ZFS filesystem!

Always check your backups!

```
# zpool export tank
# dd if=/dev/random of=/dev/md1 bs=1m count=200
# zpool import tank
```

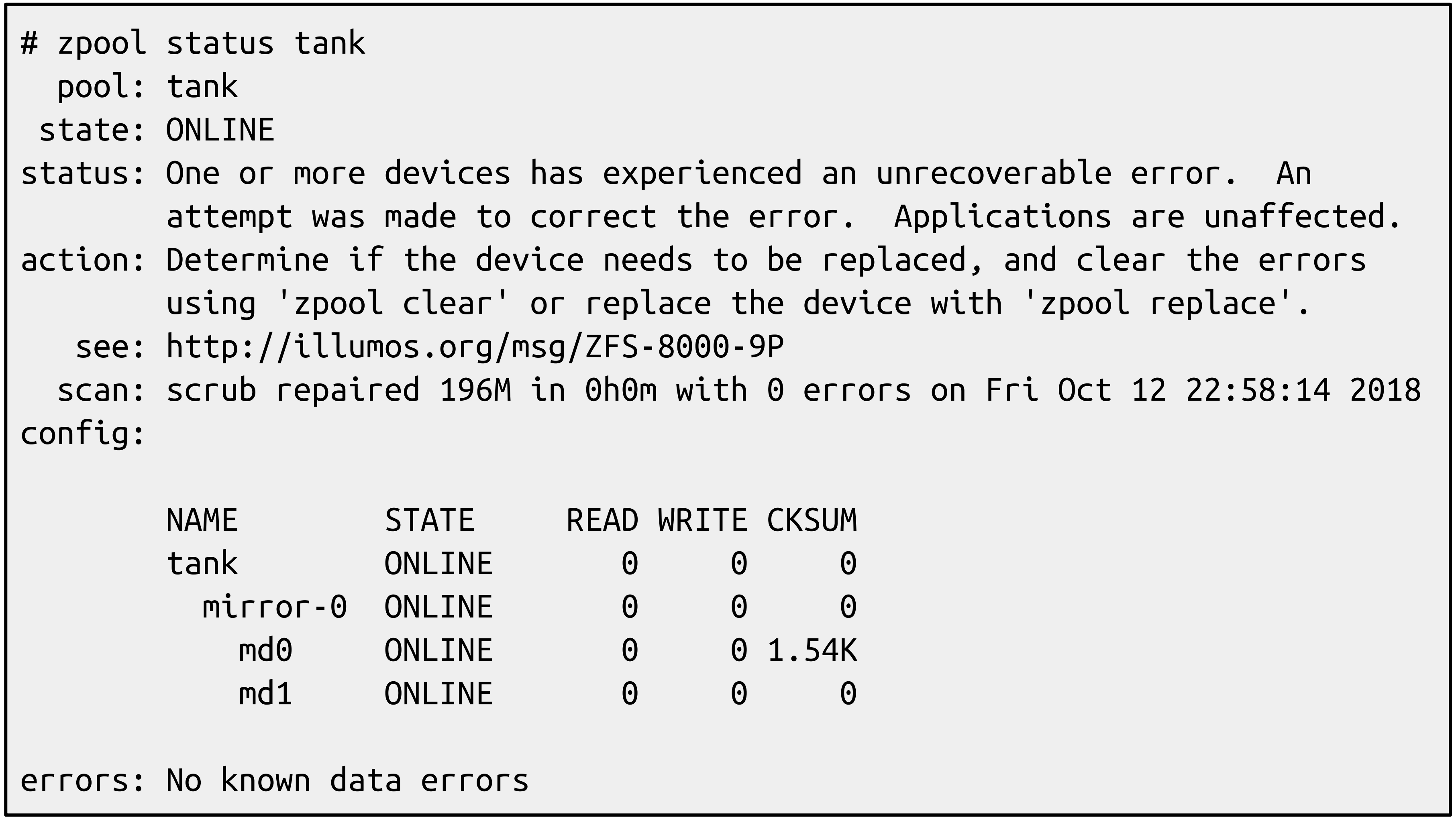
Self-healing data demo
Destroy one of the disks (2/2)

```
# zpool status tank
  pool: tank
 state: ONLINE
status: One or more devices has experienced an unrecoverable error.  An
        attempt was made to correct the error.  Applications are unaffected.
action: Determine if the device needs to be replaced, and clear the errors
        using 'zpool clear' or replace the device with 'zpool replace'.
   see: http://illumos.org/msg/ZFS-8000-9P
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            md0     ONLINE       0     0     5
            md1     ONLINE       0     0     0

errors: No known data errors
```

Self-healing data demo
Make sure everything is okay (1/3)

```
# zpool scrub tank
# zpool status tank
  pool: tank
 state: ONLINE
status: One or more devices has experienced an unrecoverable error.  An
        attempt was made to correct the error.  Applications are unaffected.
action: Determine if the device needs to be replaced, and clear the errors
        using 'zpool clear' or replace the device with 'zpool replace'.
   see: http://illumos.org/msg/ZFS-8000-9P
  scan: scrub in progress since Fri Oct 12 22:57:36 2018
        191M scanned out of 3.51G at 23.9M/s, 0h2m to go
        186M repaired, 5.32% done
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            md0     ONLINE       0     0 1.49K  (repairing)
            md1     ONLINE       0     0     0

errors: No known data errors
```

Self-healing data demo
Make sure everything is okay (2/3)



```
# zpool status tank
  pool: tank
 state: ONLINE
status: One or more devices has experienced an unrecoverable error.  An
        attempt was made to correct the error.  Applications are unaffected.
action: Determine if the device needs to be replaced, and clear the errors
        using 'zpool clear' or replace the device with 'zpool replace'.
   see: http://illumos.org/msg/ZFS-8000-9P
  scan: scrub repaired 196M in 0h0m with 0 errors on Fri Oct 12 22:58:14 2018
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            md0     ONLINE       0     0 1.54K
            md1     ONLINE       0     0     0

errors: No known data errors
```

Self-healing data demo
Make sure everything is okay (3/3)

```
# zpool clear tank

# zpool status tank
  pool: tank
 state: ONLINE
  scan: scrub repaired 196M in 0h0m with 0 errors on Fri Oct 12 22:58:14 2018
config:

        NAME          STATE     READ WRITE CKSUM
        tank          ONLINE       0     0     0
          mirror-0    ONLINE       0     0     0
            md0       ONLINE       0     0     0
            md1       ONLINE       0     0     0

errors: No known data errors
```

Self-healing data demo
But what if it goes very wrong? (1/2)

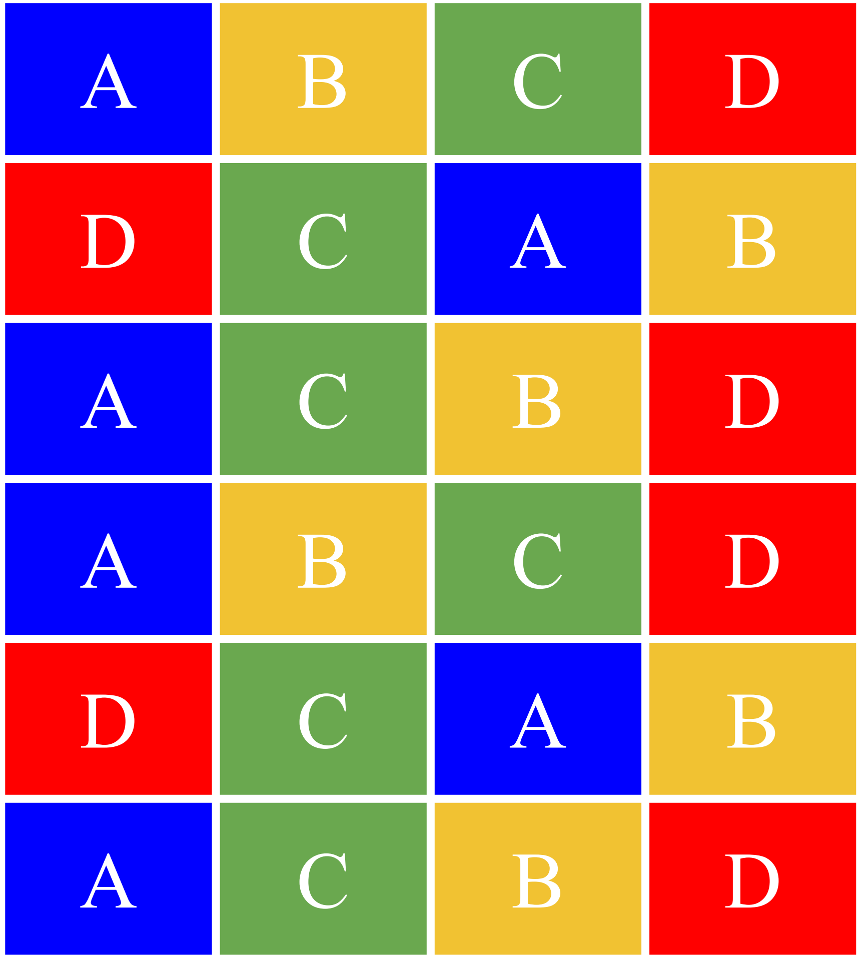
```
# zpool status
  pool: tank
 state: ONLINE
status: One or more devices has experienced an error resulting in data
        corruption.  Applications may be affected.
action: Restore the file in question if possible.  Otherwise restore the
        entire pool from backup.
   see: http://illumos.org/msg/ZFS-8000-8A
  scan: scrub in progress since Fri Oct 12 22:46:01 2018
        498M scanned out of 3.51G at 99.6M/s, 0h0m to go
        19K repaired, 13.87% done
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0 1.48K
          mirror-0  ONLINE       0     0 2.97K
            md0     ONLINE       0     0 2.97K
            md1     ONLINE       0     0 2.97K

errors: 1515 data errors, use '-v' for a list
```

Self-healing data demo
But what if it goes very wrong? (2/2)

```
# zpool status -v
  pool: tank
 state: ONLINE
status: One or more devices has experienced an error resulting in data
        corruption.  Applications may be affected.
action: Restore the file in question if possible.  Otherwise restore the
        entire pool from backup.
   see: http://illumos.org/msg/ZFS-8000-8A
  scan: scrub repaired 19K in 0h0m with 1568 errors on Fri Oct 12 22:46:25 2018
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0 1.53K
          mirror-0  ONLINE       0     0 3.07K
            md0     ONLINE       0     0 3.07K
            md1     ONLINE       0     0 3.07K

errors: Permanent errors have been detected in the following files:

        /tank/FreeBSD-11.2-RELEASE-amd64.vhd.xz
        /tank/base-amd64.txz
        /tank/FreeBSD-11.2-RELEASE-amd64-disc1.iso.xz
        /tank/intro_slides.pdf
```

Deduplication

Duplication

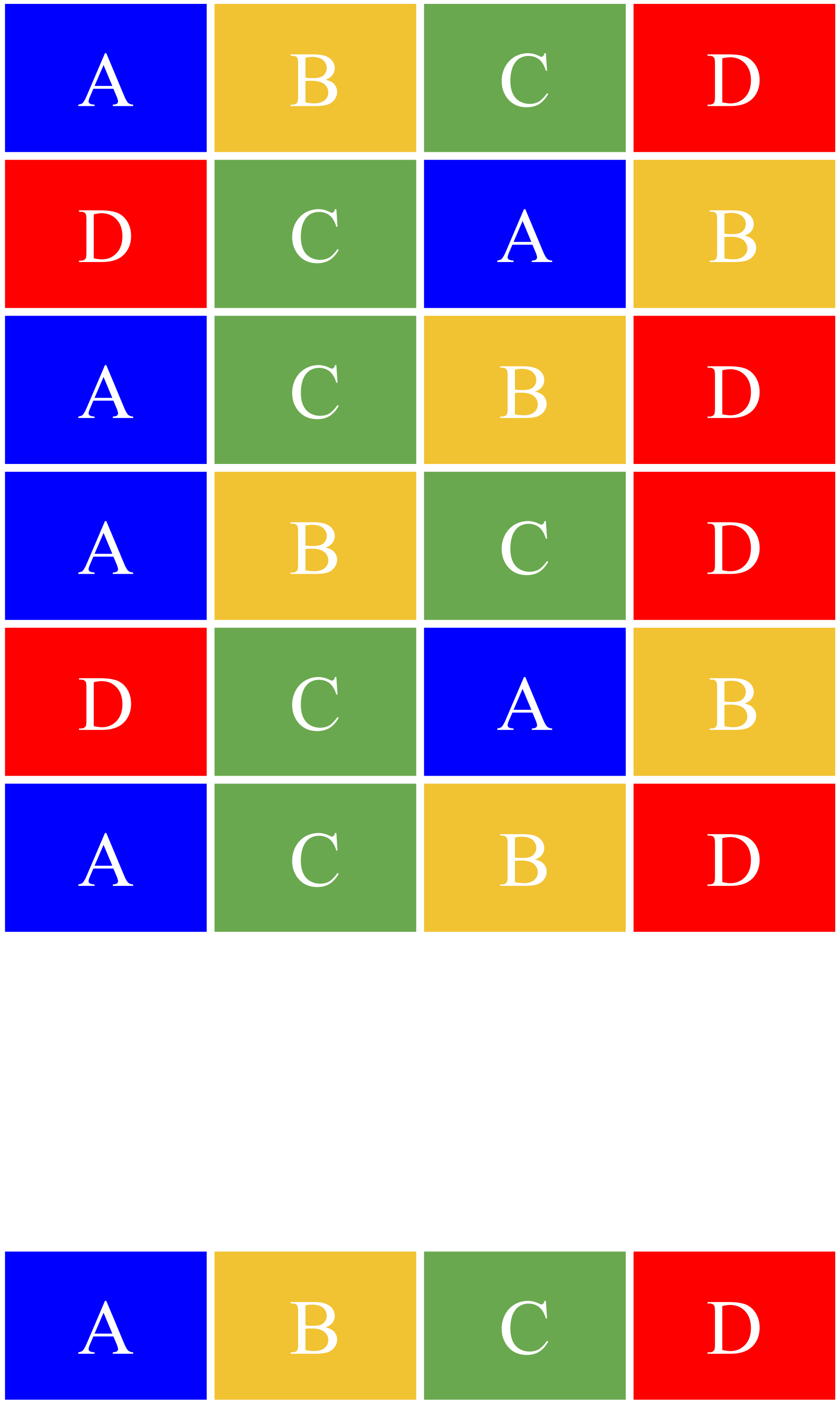


A	B	C	D
D	C	A	B
A	C	B	D
A	B	C	D
D	C	A	B
A	C	B	D

- Intentional duplication
  - ○ Backups, redundancy
- Unintentional duplication
  - ○ Application caches
  - ○ Temporary files
  - ○ Node.js (Grrr!)

Deduplication

- Implemented at the block layer
- ZFS detects when it needs to store an exact copy of a block
- Only a reference is written rather than the entire block
- Can save a lot of disk space



Deduplication

Memory cost

- ZFS must keep a table of the checksums of every block it stores
- Depending on the blocksize, this table can grow very quickly
- Deduplication table must be fast to access or writes slow down
- Ideally, the deduplication table should fit in RAM
- Keeping a L2ARC on fast SSDs can reduce the cost somewhat

Rule of thumb:

5GB of RAM for each TB of data stored

Deduplication
Is it worth it? (1/2)

- The ZFS debugger (`zdb`) can be used to evaluate if turning on deduplication will save space in a pool
- In most workloads, compression will provide much more significant savings than deduplication
- Consider whether the cost of RAM is worth it
- Also keep in mind that it is a lot easier and cheaper to add disks to a system than it is to add memory

Deduplication Demo

Is it worth it? (2/2)

```
# zdb -S tank
Simulated DDT histogram:

bucket            allocated                    referenced

______   ______________________________   ______________________________
refcnt   blocks   LSIZE   PSIZE   DSIZE   blocks   LSIZE   PSIZE   DSIZE
------   ------   -----   -----   -----   ------   -----   -----   -----
    1  25.1K   3.13G   3.13G   3.13G    25.1K   3.13G   3.13G   3.13G
    2  1.48K  189M   189M    189M   2.96K  378M   378M   378M
 Total 26.5K   3.32G   3.32G   3.32G    28.0K   3.50G   3.50G   3.50G

dedup = 1.06, compress = 1.00, copies = 1.00, dedup * compress / copies = 1.06
```

Deduplication demo
Control experiment (1/2)

```
# zpool list tank
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank  7.50G  79.5K  7.50G        -         -     0%     0%  1.00x  ONLINE  -

# zfs get compression,dedup tank
NAME  PROPERTY     VALUE          SOURCE
tank  compression  off            default
tank  dedup        off            default

# for p in `seq 0 4`; do
> zfs create tank/ports/$p
> portsnap -d /tmp/portsnap -p /tank/ports/$p extract &
> done

# zpool list tank
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank  7.50G  2.14G  5.36G        -         -     3%    28%  1.00x  ONLINE  -
```

Deduplication demo
Control experiment (2/2)

```
# zdb -S tank
Simulated DDT histogram:

bucket              allocated                       referenced
______   ______________________________   ______________________________
refcnt   blocks   LSIZE   PSIZE   DSIZE   blocks   LSIZE   PSIZE   DSIZE
------   ------   -----   -----   -----   ------   -----   -----   -----
     4     131K    374M    374M    374M     656K   1.82G   1.82G   1.82G
     8    2.28K   4.60M   4.60M   4.60M    23.9K   48.0M   48.0M   48.0M
    16      144    526K    526K    526K    3.12K   10.5M   10.5M   10.5M
    32       22   23.5K   23.5K   23.5K      920    978K    978K    978K
    64        2   1.50K   1.50K   1.50K      135    100K    100K    100K
   256        1     512     512     512      265    132K    132K    132K
 Total     134K    379M    379M    379M     685K   1.88G   1.88G   1.88G

dedup = 5.09, compress = 1.00, copies = 1.00, dedup * compress / copies = 5.09
```

Deduplication demo

Enabling deduplication

```
# zpool list tank
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank  7.50G  79.5K  7.50G        -         -     0%     0%  1.00x  ONLINE  -

# zfs get compression,dedup tank
NAME  PROPERTY     VALUE          SOURCE
tank  compression  off            default
tank  dedup        on             default

# for p in `seq 0 4`; do
> zfs create tank/ports/$p
> portsnap -d /tmp/portsnap -p /tank/ports/$p extract &
> done

# zpool list tank
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank  7.50G   670M  6.85G        -         -     6%     8%  5.08x  ONLINE  -
```

Deduplication demo

Compare with compression

```
# zpool list tank
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank  7.50G  79.5K  7.50G        -         -     0%     0%  1.00x  ONLINE  -

# zfs get compression,dedup tank
NAME  PROPERTY     VALUE          SOURCE
tank  compression  gzip-9         local
tank  dedup        off            default

# for p in `seq 0 4`; do
> zfs create tank/ports/$p
> portsnap -d /tmp/portsnap -p /tank/ports/$p extract &
> done

# zpool list tank
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
tank  7.50G   752M  6.77G        -         -     3%     9%  1.00x  ONLINE  -
```

Deduplication

Summary

- ZFS deduplication can save a lot of space under some workloads but at the expense of a lot of memory
- Often, compression will give similar or better results
- Always check with **zdb -S** whether deduplication would be worth it

Control experiment	2.14G
Deduplication	670M
Compression	752M

Performance Tuning

General tuning tips

- System memory
- Access time
- Dataset compression
- Deduplication
- ZFS send and receive

Random Access Memory

- ZFS performance depends on the amount of system
    - recommended minimum: 1GB
    - 4GB is ok
    - 8GB and more is good

Dataset Compression

- Save space
- Increase CPU usage
- Increase data throughput (density)

Deduplication

- Requires even more memory
- Increases CPU usage

ZFS send/recv

- Using buffer for large streams
  - misc/buffer
  - misc/mbuffer (network capable)

Database tuning

- For PostgreSQL and MySQL users recommend using a different recordsize than default 128k.

- PostgreSQL: 8k
- MySQL MyISAM storage: 8k
- MySQL InnoDB storage: 16k

File Servers

- Disable access time
- Keep number of snapshots low
- Dedup only if you have lots of RAM
- For heavy write workloads move ZIL to separate SSD drives
- Optionally disable ZIL for datasets (beware consequences)

Webservers

- Disable redundant data caching
  - Apache
    - EnableMMAP Off
    - EnableSendfile Off
  - Nginx
    - Sendfile off
  - Lighttpd
    - server.network-backend="writev"

Cache and Prefetch

ARC

- Adaptive Replacement Cache
  - Resides in system RAM
  - Major speedup to ZFS the size is auto-tuned
  - Default
    - arc max: memory size - 1GB
    - metadata limit: ¼ of arc_max
    - arc min: ½ of arc_meta_limit (but at least 16MB)

Tuning ARC

- Disable ARC on per-dataset level
- Maximum can be limited if you also run other things

```
# sysctl vfs.zfs.arc_max
# sysctl vfs.zfs.arc_free_target
```

- Increasing arc_meta_limit may help if working with (too) many files

```
# sysctl kstat.zfs.misc.arcstats.size
# sysctl kstat.zfs.misc.arcstats.arc_meta_used
# sysctl kstat.zfs.misc.arcstats.arc_meta_limit
```

- http://www.krausam.de/?p=70

L2ARC

- L2 Adaptive Replacement Cache
  - is designed to run on fast block devices (SSD)
  - helps primarily read-intensive workloads
  - each device can be attached to only one ZFS pool

```
# zpool add <pool name> cache <vdevs>
# zpool add remove <pool name> <vdevs>
```

Tuning L2ARC

- Enable prefetch for streaming or serving of large files
- Configurable on per-dataset basis
- Turbo warm-up phase may require tuning (e.g. set to 16MB)

```
vfs.zfs.l2arc_noprefetch
vfs.zfs.l2arc_write_max
vfs.zfs.l2arc_write_boost
```

```
# new names in openzfs
vfs.zfs.l2arc.noprefetch
vfs.zfs.l2arc.write_max
vfs.zfs.l2arc.write_boost
```

ZIL

- ZFS Intent Log
  - guarantees data consistency on fsync() calls
  - replays transaction in case of a panic or power failure
  - use small storage space on each pool by default
- To speed up writes, deploy zil on a separate log device(SSD)
- Per-dataset synchonocity behavior can be configured
  - # zfs set sync=[standard|always|disabled] dataset

File-level Prefetch (zfetch)

- Analyses read patterns of files
- Tries to predict next reads
- Loader tunable to enable/disable zfetch
  - vfs.zfs.prefetch_disable
  - vfs.zfs.prefetch.disable (openzfs)

Device-level Prefetch (vdev prefetch)

- reads data after small reads from pool devices
- useful for drives with higher latency
- consumes constant RAM per vdev
- is disabled by default
- Loader tunable to enable/disable vdev prefetch
  - vfs.zfs.vdev.cache.size=[bytes]

ZFS Statistics Tools

- # sysctl vfs.zfs
- # sysctl kstat.zfs
- using tools:
    - zfs-stats: analyzes settings and counters since boot
    - zfsf-mon: real-time statistics with averages
- Both tools are available in ports under sysutils/zfs-stats

References

- ZFS: The last word in filesystems (Jeff Bonwick & Bill Moore)
- ZFS tuning in FreeBSD (Martin Matuˇska):
  - Slide
    - http://blog.vx.sk/uploads/conferences/EuroBSDcon2012/zfs-tuning-handout.pdf
  - Video
    - https://www.youtube.com/watch?v=PIpI7Ub6yjo
- Becoming a ZFS Ninja (Ben Rockwood):
  - http://www.cuddletech.com/blog/pivot/entry.php?id=1075
- ZFS Administration:
  - https://pthree.org/2012/12/14/zfs-administration-part-ix-copy-on-write

References (c.)

- https://www.freebsd.org/doc/zh_TW/books/handbook/zfs-zfs.html
- "ZFS Mastery" books (Michael W. Lucas & Allan Jude)
  - FreeBSD Mastery: ZFS
  - FreeBSD Mastery: Advanced ZFS
- ZFS for Newbies (Dan Langille)
  - https://www.youtube.com/watch?v=3oG-1U5AI9A&list=PLskKNopggjc6NssLc8GEGSiFYJLYdlTQx&index=20
- The future of OpenZFS and FreeBSD (Allan Jude)
  - https://www.youtube.com/watch?v=gmaHZBwDKho&list=PLskKNopggjc6NssLc8GEGSiFYJLYdlTQx&index=23
- How ZFS snapshots really work (Matt Ahrens)
  - https://www.bsdcan.org/2019/schedule/events/1073.en.html
- An Introduction to the Implementation of ZFS (Kirk McKusick)
  - https://www.bsdcan.org/2015/schedule/events/525.en.html
- https://open-zfs.org
- Boot environments: bectl(8)