ZFS: An Overview

OmniTI / Eric Sproul

What is ZFS?

- Filesystem, volume manager, and RAID controller all in one
- More properly: a storage sub-system
- Production debut in Solaris 10 6/06 ("Update 2")
- Also available on FreeBSD, Linux, MacOS X
- 128-bit
- 2^{64} snapshots, 2^{48} files/directory, 2^{64} bytes/filesystem, 2^{78} bytes/pool, 2^{64} devices/pool, 2^{64} pools/system

OmniTI

128 Bits? Are You High?

"That's enough to survive Moore's Law until I'm dead."

- Jeff Bonwick, co-author of ZFS, 2004

http://blogs.oracle.com/bonwick/en_US/entry/128_bit_storage_are_you

- Petabyte data sets are increasingly common
- 1PB = 2^{50} bytes
- 64-bit capacity limit only 14 doublings away
- Storage capacities doubling every 9-12 months
- Have about a decade before 64-bit space exhausted
- Filesystems tend to be around for several decades
- UFS, HFS: mid-1980s, ext2: 1993, XFS: 1994

OmniTI

What Does ZFS Do?

- Turns a collection of disks into a storage pool
- Provides immense storage capacity
  - 256 ZB, or 2^{78} bytes/pool
- Simplifies storage administration
  - Two commands: zpool, zfs

OmniTI

What *Else* Does ZFS Do?

- Always consistent on disk (goodbye, fsck!)
- End-to-end, provable data integrity
- Snapshots, clones
- Block-level replication
- NAS/SAN features: NFS & CIFS shares, iSCSI targets
- Transparent compression, de-duplication
- Can use SSDs seamlessly to:
  - extend traditional RAM-based read cache (L2ARC)
  - provide a low-latency sync write accelerator (SLOG)

OmniTI

Pooled Storage?

Old & Busted

- Must decide on partitioning up front
- Limited number of slices
- Leads to wasted space, unintuitive layouts
- Costly to fix if wrong

New Hotness

- Big bucket o' storage
- "Slice" becomes meaningless concept
- Data only occupies space as needed
- Organize data according to its nature

OmniTI

Useful Terms

zpool: One or more devices that provide physical storage and (optionally) data replication for ZFS datasets. Also the root of the namespace hierarchy.

vdev: A single device or collection of devices organized according to certain performance and fault-tolerance characteristics. These are the building blocks of zpools.

dataset: A unique path within the ZFS namespace, e.g. tank/users, tank/db/mysql/data

property: Read-only or configurable object that can report statistics or control some aspect of dataset behavior. Properties are inherited from the parent unless overridden by the child.

Zpool Structure

- Zpools contain top-level vdevs, which in turn may contain leaf vdevs
- vdev types: block device, file, mirror, raidz{1,2,3}, spare, log, cache
- Certain vdev types provide fault tolerance (mirror, raidzN)
- Data striped across multiple top-level vdevs
- Zpools can be expanded on-the-fly by adding more top-level vdevs, but cannot be shrunk

Zpool Examples

Single disk: `zpool create data c0t2d0`

```
NAME        STATE     READ WRITE CKSUM
data        ONLINE       0     0     0
  c0t2d0    ONLINE       0     0     0
```

Mirror: `zpool create data mirror c0t2d0 c0t3d0`

```
NAME        STATE     READ WRITE CKSUM
data        ONLINE       0     0     0
  mirror-0  ONLINE       0     0     0
    c0t2d0  ONLINE       0     0     0
    c0t3d0  ONLINE       0     0     0
```

OmniTI

Zpool Examples

Striped Mirror: `zpool create data mirror c0t2d0 c0t3d0 mirror c0t4d0 c0t5d0`

```
NAME         STATE     READ WRITE CKSUM
data         ONLINE       0     0     0
  mirror-0   ONLINE       0     0     0
    c0t2d0   ONLINE       0     0     0
    c0t3d0   ONLINE       0     0     0
  mirror-1   ONLINE       0     0     0
    c0t4d0   ONLINE       0     0     0
    c0t5d0   ONLINE       0     0     0
```

RAID-Z: `zpool create data raidz c0t2d0 c0t3d0 c0t4d0`

```
NAME         STATE     READ WRITE CKSUM
data         ONLINE       0     0     0
  raidz1-0   ONLINE       0     0     0
    c0t2d0   ONLINE       0     0     0
    c0t3d0   ONLINE       0     0     0
    c0t4d0   ONLINE       0     0     0
```

OmniTI

Datasets

- Hierarchical namespace, rooted at <poolname>
- Default type: filesystem
- Other types: volume (zvol), snapshot, clone
- Easy to create; use datasets as policy administration points
- Can be moved to another pool or backed up via zfs send/recv

```
# zfs list data
NAME                 USED  AVAIL  REFER  MOUNTPOINT
data                1.03G  6.78G    22K  /data
data/myfs             21K  6.78G    21K  /data/myfs
data/myfs@today        0      -    21K  -
data/home             21K  6.78G    21K  /export/home
data/myvol          1.03G  7.81G    16K  -
```

Dataset Properties

```
# zfs get all data/myfs
NAME        PROPERTY              VALUE                      SOURCE
data/myfs   type                  filesystem                 -
data/myfs   creation              Tue Sep  3 19:28 2013      -
data/myfs   used                  31K                        -
data/myfs   available             441G                       -
data/myfs   referenced            31K                        -
data/myfs   compressratio         1.00x                      -
data/myfs   mounted               yes                        -
data/myfs   quota                 none                       default
data/myfs   reservation           none                       default
data/myfs   recordsize            128K                       default
data/myfs   mountpoint            /data/myfs                 default
data/myfs   sharenfs              off                        default
data/myfs   checksum              on                         default
data/myfs   compression           on                         inherited from data
data/myfs   atime                 on                         default
...
```

OmniTI

Property Example: mountpoint

```
# zfs get mountpoint data/myfs
NAME        PROPERTY    VALUE       SOURCE
data/myfs   mountpoint  /data/myfs  default

# df -h
Filesystem             size   used  avail capacity  Mounted on
data                   7.8G    21K   6.8G     1%    /data
data/myfs              7.8G    21K   6.8G     1%    /data/myfs

# zfs set mountpoint=/omgcool data/myfs

# zfs get mountpoint data/myfs
NAME        PROPERTY    VALUE       SOURCE
data/myfs   mountpoint  /omgcool    local

# df -h
Filesystem             size   used  avail capacity  Mounted on
data                   7.8G    21K   6.8G     1%    /data
data/myfs              7.8G    21K   6.8G     1%    /omgcool
```

Property Example: compression

Setting affects new data only
Default algorithm is LZJB (an LZO variant)
Also gzip(-{1-9}), zle, lz4

```
# cp /usr/dict/words /omgcool/
# du -h --apparent-size /omgcool/words
202K    /omgcool/words

# zfs set compression=on data/myfs
```

```
# cp /usr/dict/words /omgcool/words.2
# du -h --apparent-size /omgcool/words*
202K    /omgcool/words
202K    /omgcool/words.2

# du -h /omgcool/words*
259K    /omgcool/words
138K    /omgcool/words.2
```

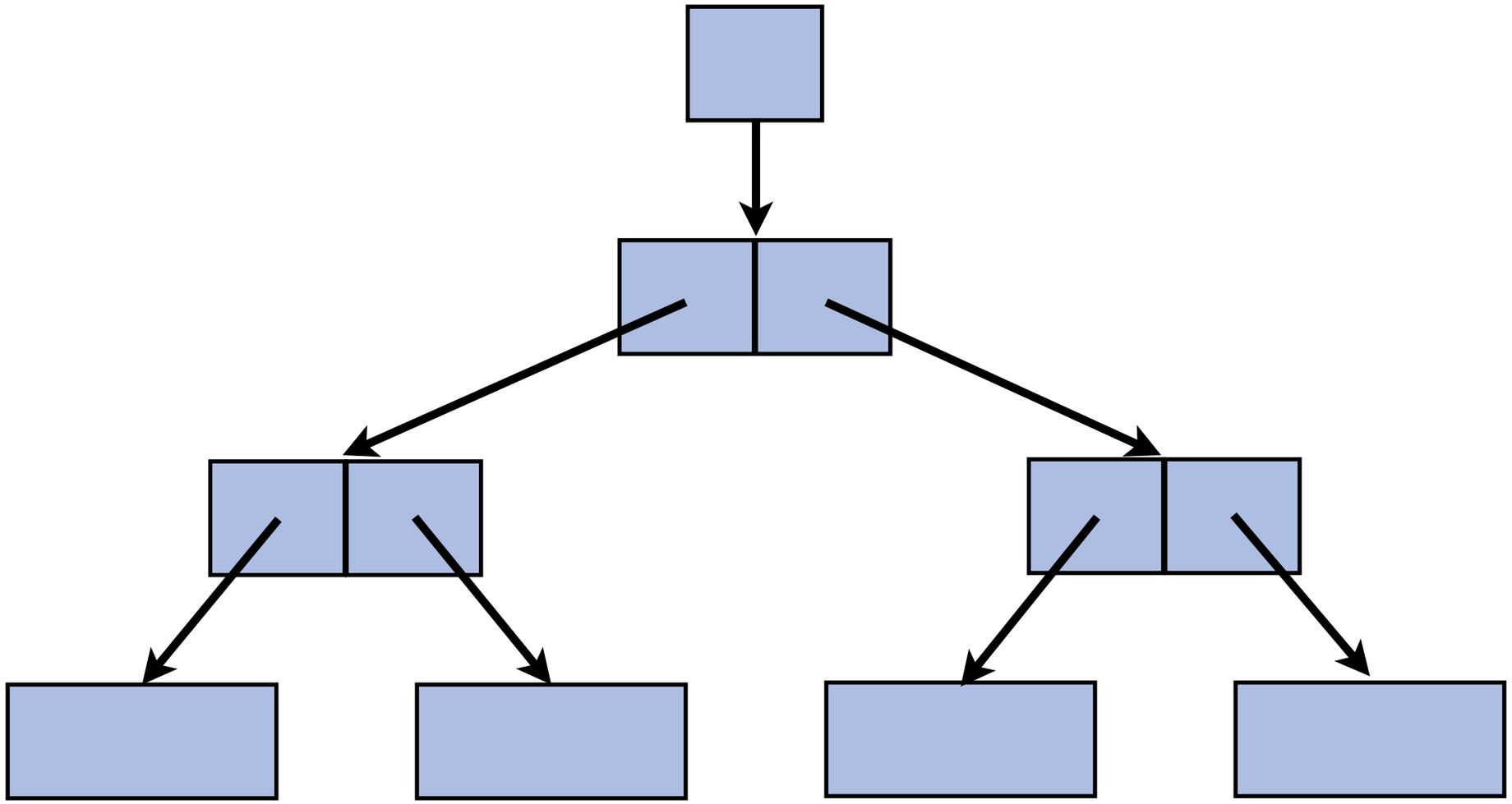
OmniTI

On-Disk Consistency

- Copy-on-write: never overwrite existing data
- Transactional, atomic updates
- In case of power failure, data is either old or new, not a mix
- *This does NOT mean you won't lose data!* Only that you stand to lose what was in flight, instead of (potentially) the entire pool.

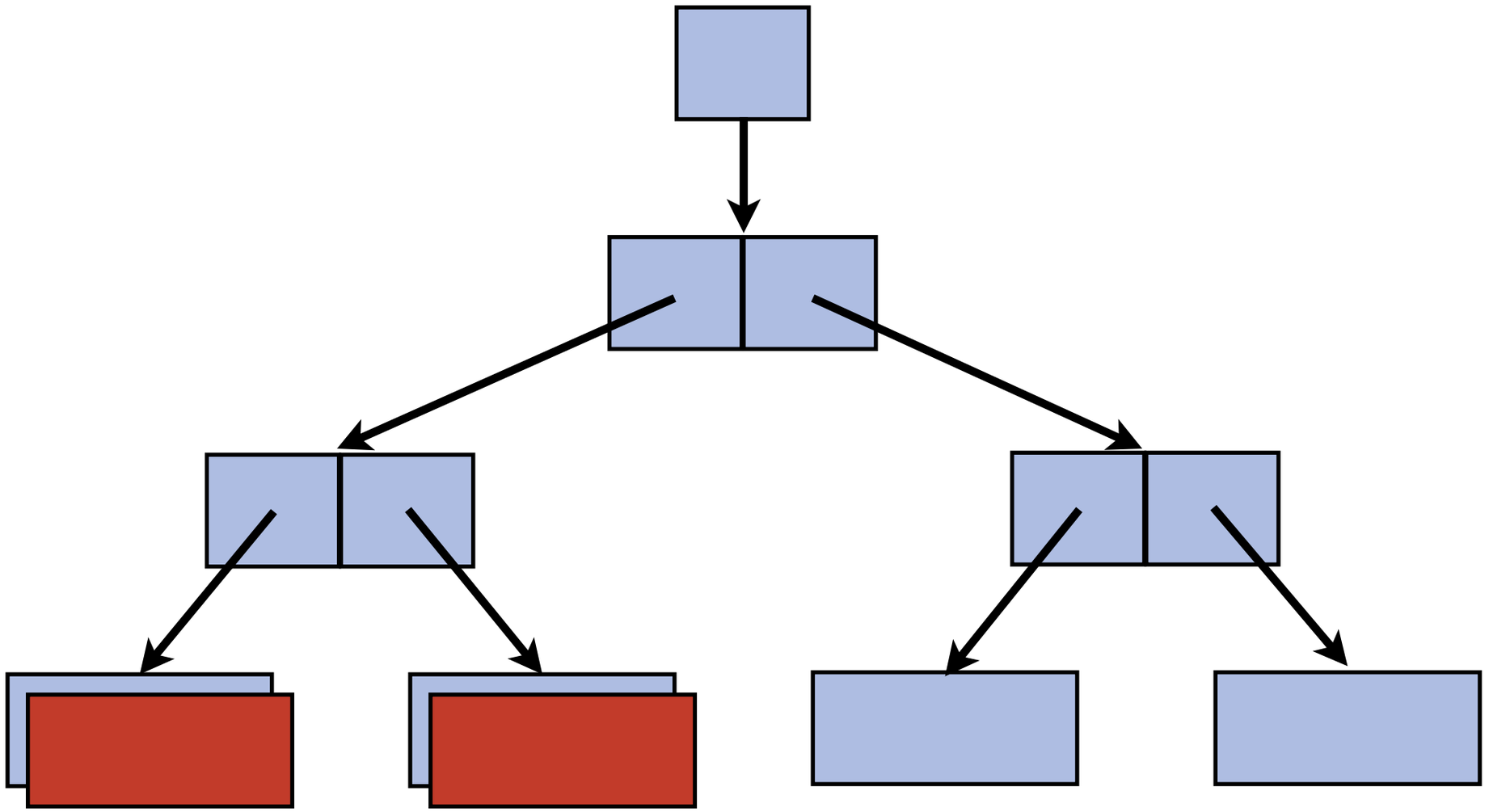
OmniTI

Copy-On-Write



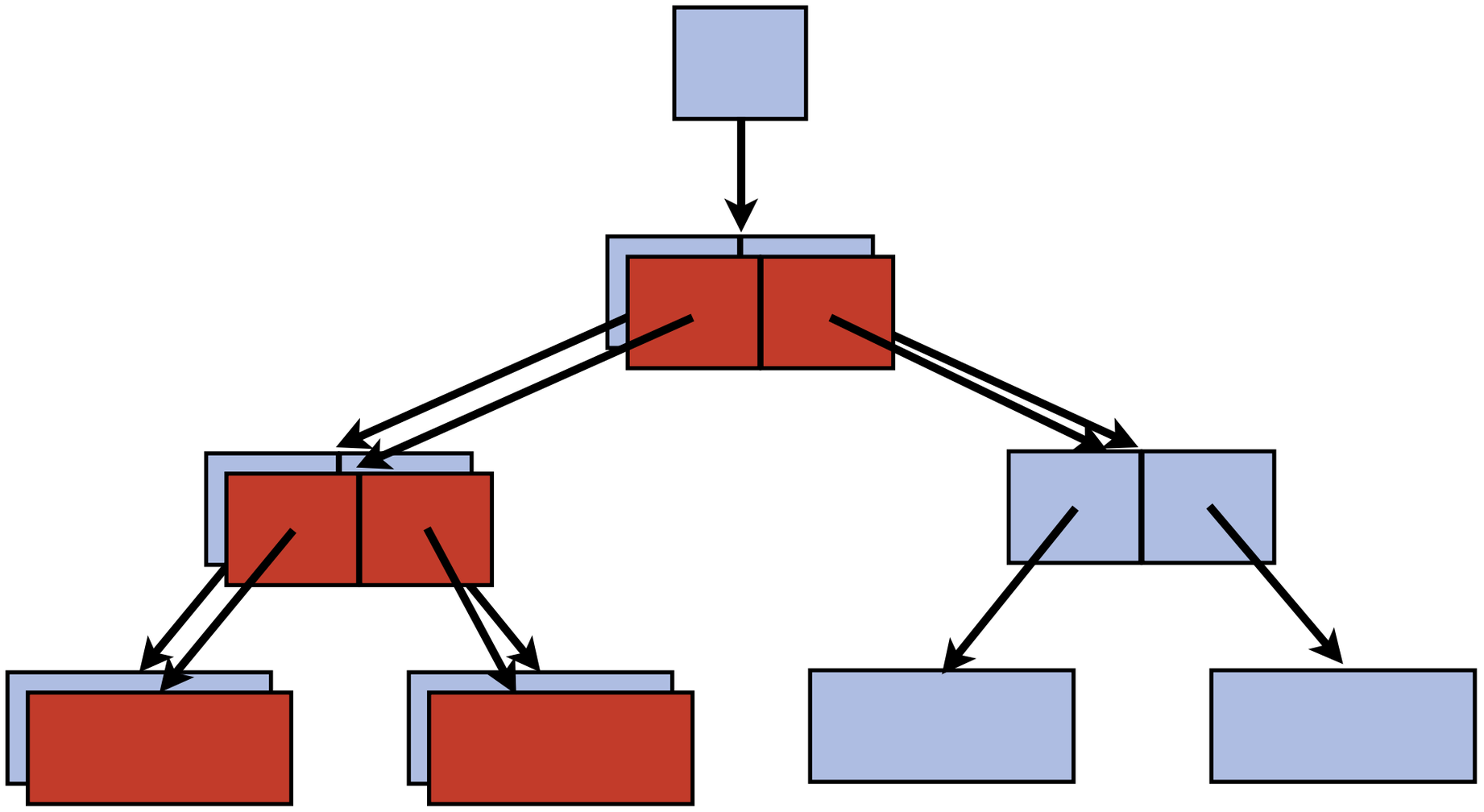
Starting block tree

Copy-On-Write



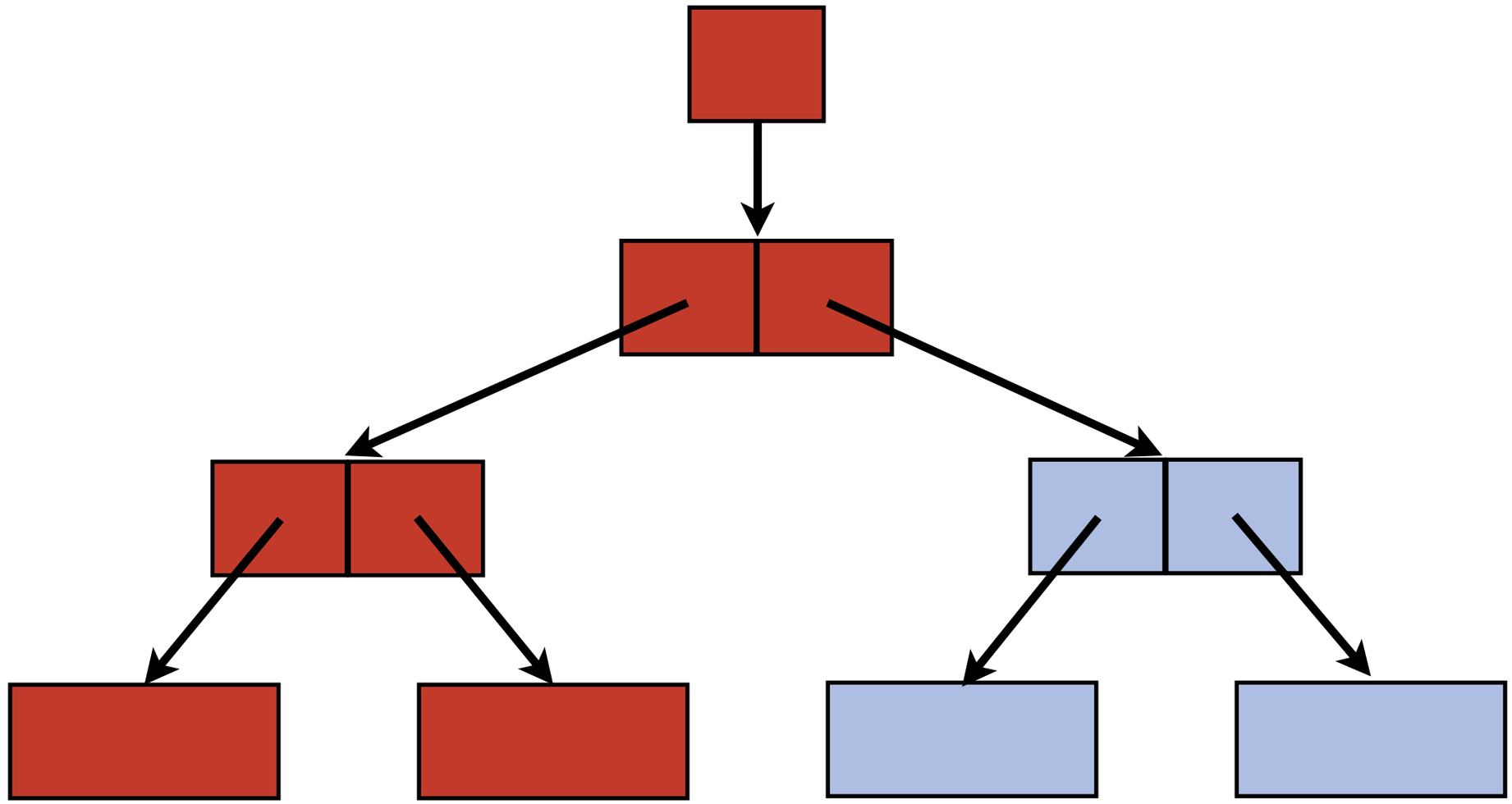
Changed data get new blocks
Never modifies existing data

Copy-On-Write



Indirect blocks also change

Copy-On-Write



Atomically update* uberblock to point at updated blocks

*The uberblock technically gets overwritten, *but:*
4 copies are stored as part of the vdev label and are updated in transactional pairs

OmniTI

Data Integrity

- Silent corruption is our mortal enemy
  - Defects can occur anywhere: disks, firmware, cables, kernel drivers
  - Main memory has ECC and periodic scrubbing; why shouldn't storage have something similar?
- "Noisy" corruption still a problem too
  - Power outages, accidental overwrite, use a disk as swap

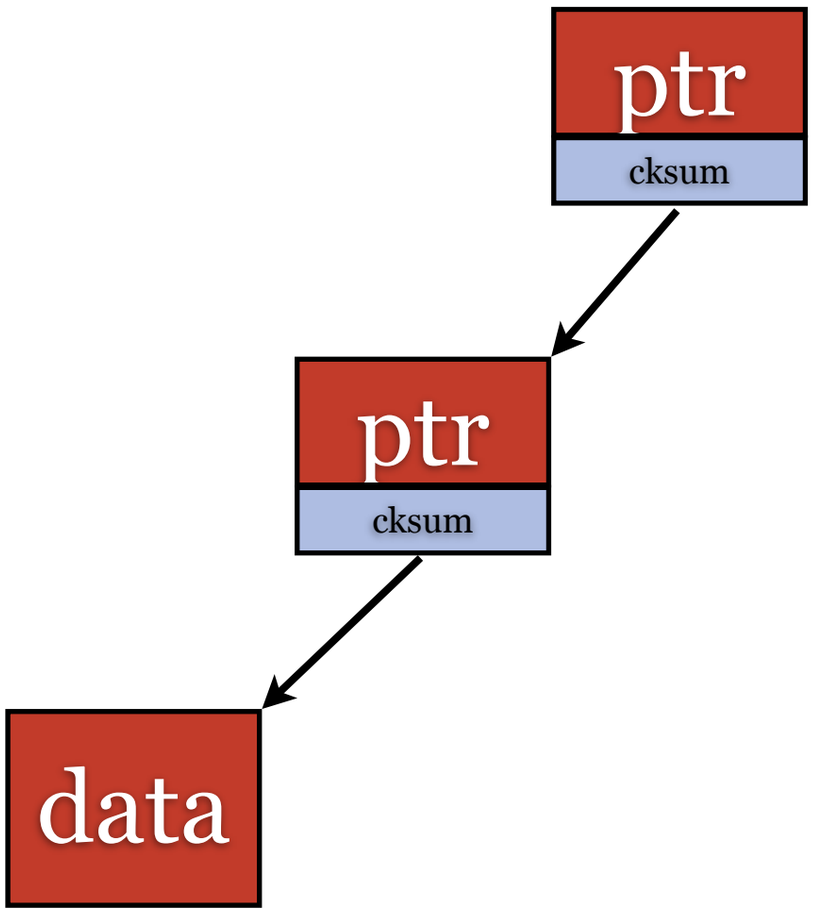
Traditional Method: Disk Block Checksum

Only detects problems after data is successfully written (“bit rot”)

Won’t catch silent corruption caused by
issues in the I/O path between host and disk, e.g.
HBA/array firmware bugs, bad cabling

Data Integrity

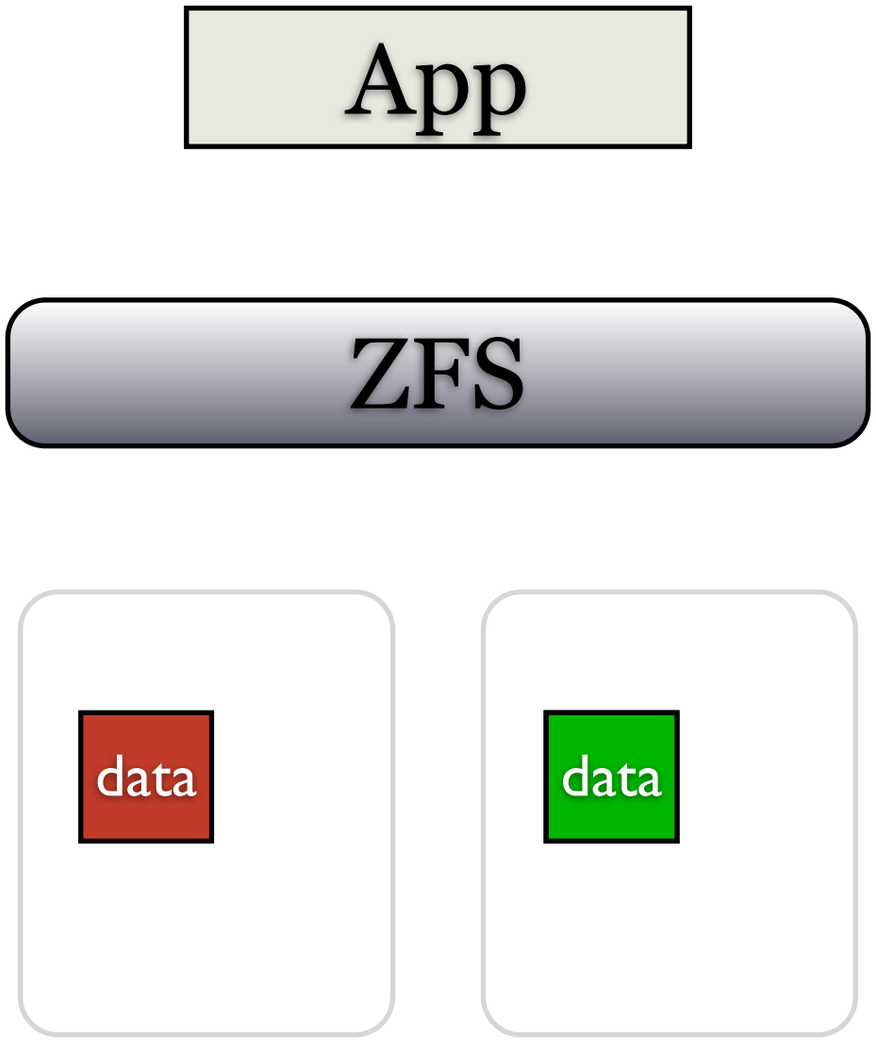
The ZFS Way



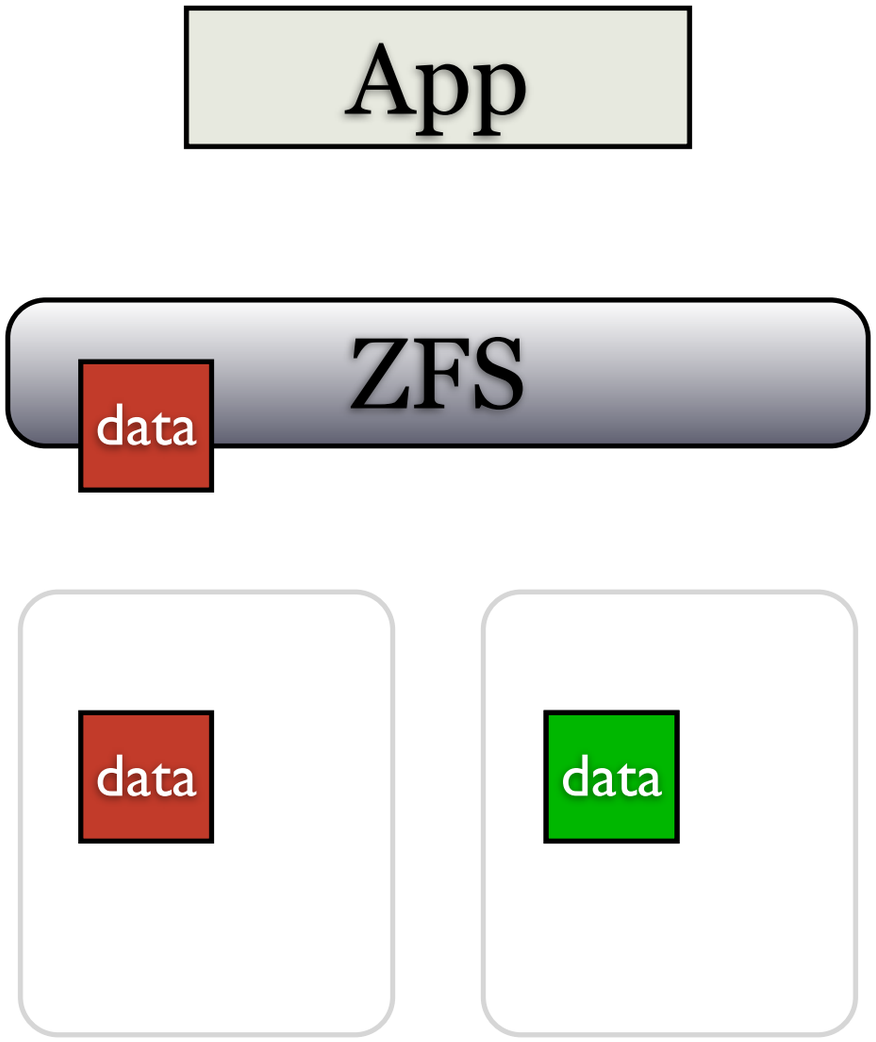
- Store checksum in block pointer
- Isolates faults between checksum and data
- Forms a hash tree, enabling validation of the entire pool
- 256-bit checksums
- fletcher4 (default; simple and fast) or SHA-256 (slower, more secure)
- Checked every time block is read
- ‘zpool scrub’: validate entire pool on demand

OmniTI

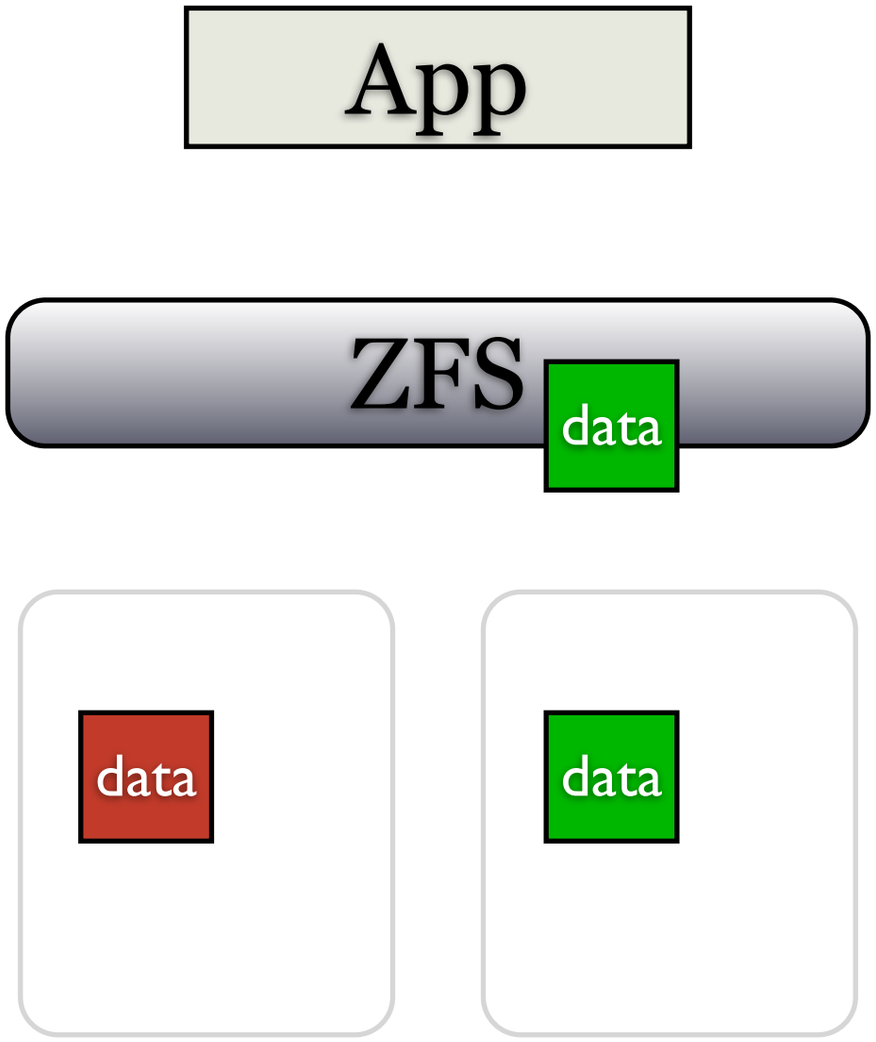
Data Integrity



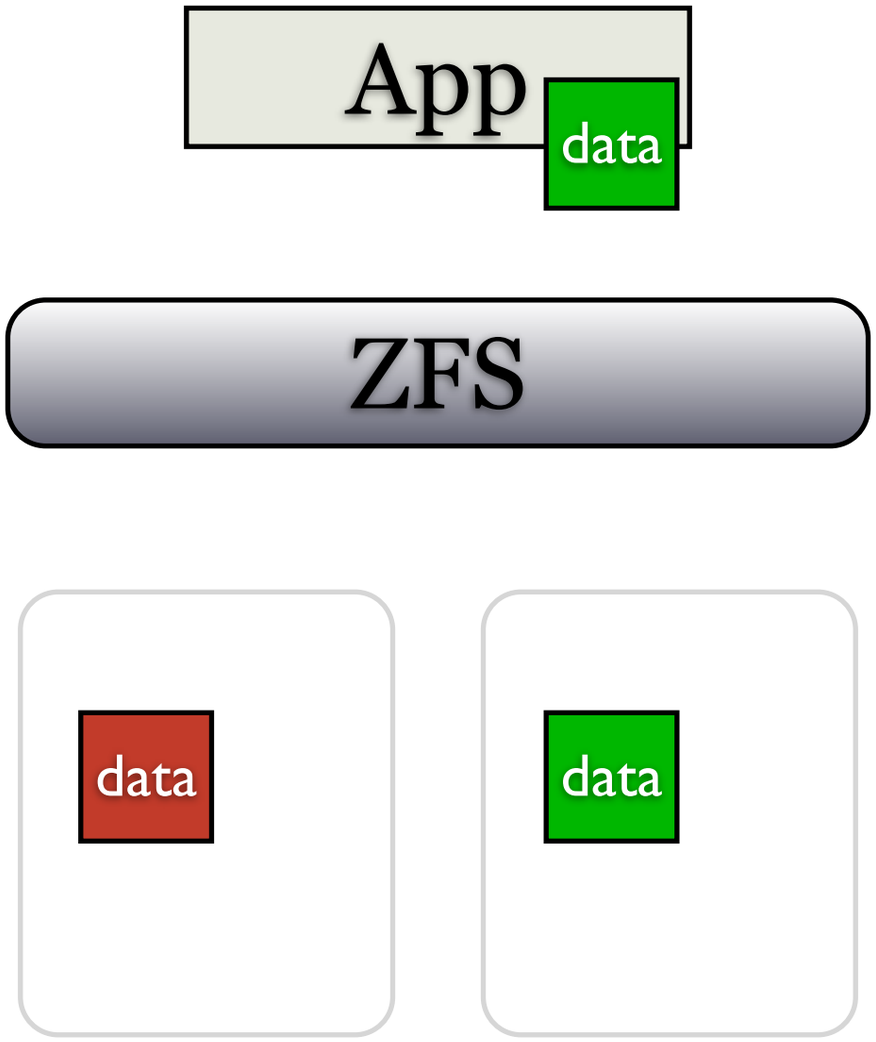
Data Integrity



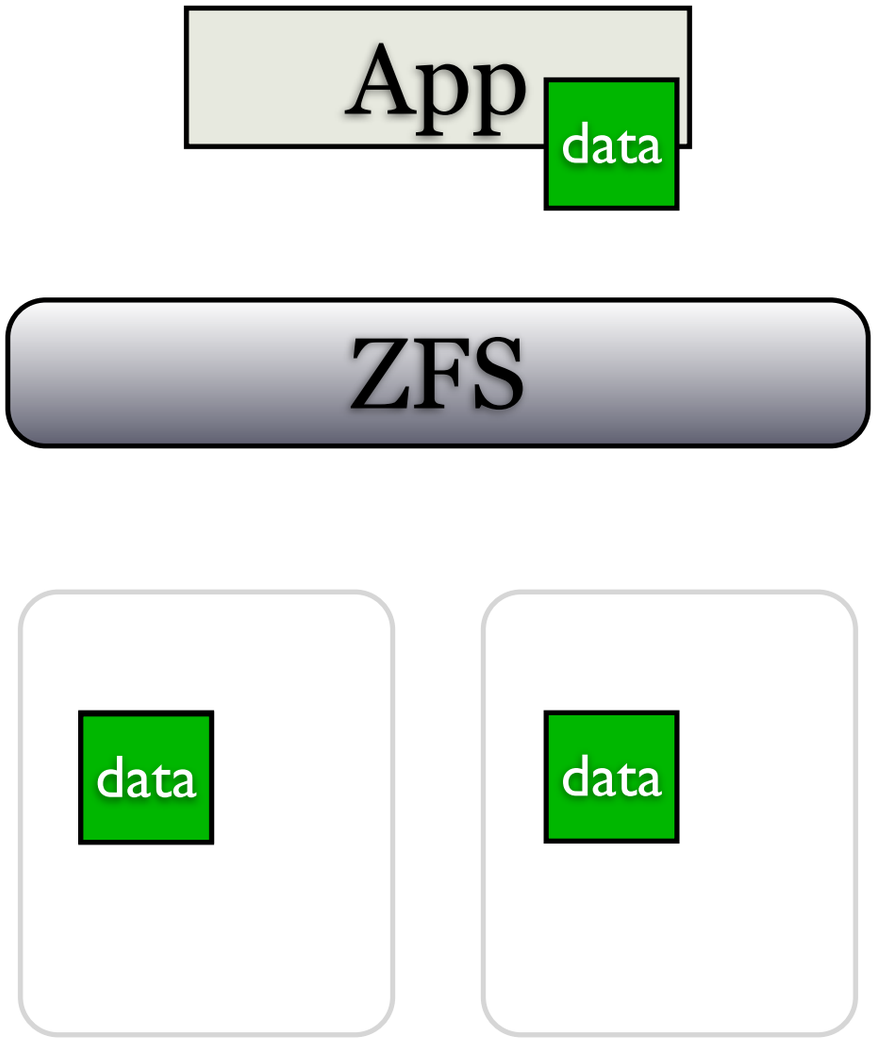
Data Integrity



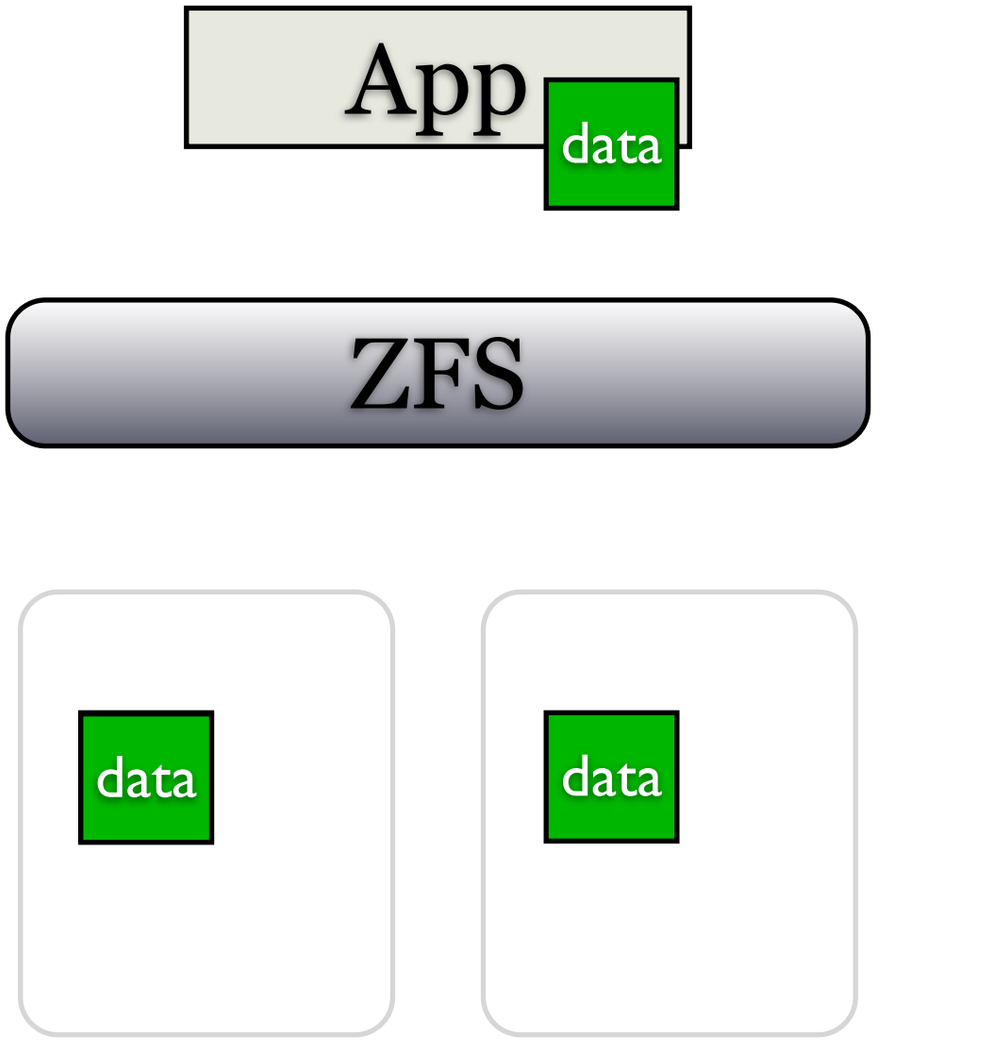
Data Integrity



Data Integrity



Data Integrity



Self-healing mirror!

Snapshots

- Read-only copy of a filesystem or volume
- Denoted by ‘@’ in the dataset name
- Constant time, consume almost no space at first
- Can have arbitrary names
- Filesystem snapshots can be browsed via a hidden directory
  - .zfs/snapshot/<snapname>
  - visibility controlled by *snapdir* property

OmniTI

Clones

- Read-write snapshot
- Uses snapshot as origin
- Changes accumulate to clone
- Unmodified data references origin snapshot
- Saves space when making many copies of similar data

Block-Level Replication

- zfs 'send' and 'receive' sub-commands
- Source is a snapshot
- 'zfs send' results in a stream of bytes to standard output
- 'zfs receive' creates a new dataset at the destination
- Send incremental between any two snapshots of the same dataset
- Pipe output to ssh or nc for remote replication

```
# zfs snapshot data/myfs@snap1
# zfs send data/myfs@snap1 | ssh host2 "zfs receive tank/myfs"
# zfs snapshot data/myfs@snap2
# zfs send -i data/myfs@snap1 data/myfs@snap2 | \
    ssh host2 "zfs receive tank/myfs"
```

OmniTI

NAS/SAN Features

- NAS: sharenfs, sharesmb
  - Activate by setting property to "on"
  - Additional config options may be passed in lieu of "on"
  - illumos, Solaris, Linux have both; FreeBSD has only sharenfs
- SAN: shareiscsi
  - Only works on ZFS volumes (zvols)
  - Linux still uses this; illumos/Solaris have COMSTAR; FreeBSD does not have an iSCSI target daemon

OmniTI

Block Transforms

- Compression
  - lzjb, gzip, zle, lz4
  - lzjb, zle, lz4 are fast; basically "free" on modern CPUs
  - Can improve performance due to fewer IOPS
- De-duplication
  - Not a general-purpose solution
  - Make sure you have lots of RAM available

OmniTI

Solid-State Disks

- Used for extra read cache and to accelerate sync writes
- Middle ground of latency, cost/GB between RAM & spinning platter
- Read: L2ARC (vdev type "cache")
  - Extends ARC (RAM cache)
  - Large MLC devices
- Write: SLOG (vdev type "log")
  - Accelerates the ZFS Intent Log (ZIL), which tracks sync writes to be replayed in case of failure
  - Small SLC devices
- Increasingly, SSDs are supplanting spinning disks as primary storage

OmniTI

ZFS Case Study: Staging Database

- Developing web app fronting large PgSQL BI database
- Need a writable copy for application testing
- Requirements:
  - Quick to create
  - Repeatable
  - Must not threaten availability or redundancy

OmniTI

ZFS Case Study: Staging Database

Starting state

```
bi01tank/pgsql/data
bi01tank/pgsql/wal_archive
```

ZFS Case Study: Staging Database

Take snapshots

```
zfs snapshot -r bi01tank/pgsql@stage
```

```
bi01tank/pgsql/data@stage
bi01tank/pgsql/wal_archive@stage
```

ZFS Case Study: Staging Database

Create clones

```
zfs clone <snapshot> <new_dataset>
```

```
bi01tank/pgsql/data@stage
bi01tank/pgsql/wal_archive@stage
```

```
bi01tank/stage/data
bi01tank/stage/wal_archive
```

Cloned datasets are dependent on their origin
Unchanged data is referenced, new data accumulates to clone

OmniTI

ZFS Case Study: Staging Database

Stage zone

```
set zonepath=/zones/bistage
set autoboot=true
set
limitpriv=default,dtrace_proc,dtrace_user
set ip-type=shared
add net
set address=10.11.12.13
set physical=bnx0
end
add dataset
set name=bi01tank/stage
end
```

OmniTI

ZFS Case Study: Staging Database

- Inside "bistage" zone we now have a writable copy of the DB
- Can now bring up Postgres, use it, discard data when done
- Only changed data occupies additional space
- Unmodified data references origin snapshot

OmniTI

Where Can I Get ZFS?

- illumos (SmartOS, OmniOS, OpenIndiana, etc.)
- Oracle Solaris 10, 11
- FreeBSD >= 7
- Linux: http://zfsonlinux.org/
  - supports kernels 2.6.26 - 3.11
  - packages for most distros
- MacOS X
  - MacZFS: https://code.google.com/p/maczfs/
  - supports 10.5-10.8

OmniTI

Questions?

- http://open-zfs.org/
- http://wiki.illumos.org/display/illumos/ZFS
- http://zfsonlinux.org/faq.html
- http://www.freebsd.org/doc/handbook/filesystems-zfs.html
- https://code.google.com/p/maczfs/wiki/FAQ

OmniTI