OpenZFS

Novel algorithms of the ZFS storage system

Matt Ahrens
Principal Engineer, Delphix
ZFS co-creator
Brown CS 2001

BROWN UNIVERSITY

IN DEO SPERAMUS

Directions to Brown | Photo Tour | News & Events | Phone Book | What's New? | Search Brown

---

- **About Brown**
  Getting Here, Phone Book, History, Campus Photographs. Welcome New Students
- **Administration**
  Departments, Policies, Brown Job Listing.
- **Alumni Information**
  Alumni Events and Information, Brown Alumni Association, Alumni Monthly.
- **Computing Services**
  Computer Store; Scholarly Technology Group; Services; Training; Web;Mail, Network, Phone, Video.
- **News & Events**
  George Street Journal, Weekly Calendar, News Releases.
- **Academics**
  College, Graduate School, School of Medicine, Departments, Summer Studies, Courses, Registrar's Office, Continuing Education.
- **Admission**
  Application Information and Financial Aid
  Virtual Campus Tour
- **Athletics**
  NCAA Athletic Certification Review
- **Library**
  General Information, Electronic Resources, Beyond Brown, Library Publications.
- **Student Information**
  Academic Deans, Academic Calendar, Course Information, Handbook, Organizations, Room Schedule.

Brown in the Community | **About Rhode Island**

Established in 1764, Brown is a liberal arts University of some 5,500 undergraduates located in Providence, Rhode Island. A member of the Ivy League, Brown is known for its flexible curriculum and opportunities for independent study and collaborative work with faculty at all levels. Brown has a School of Medicine (300 students) and Graduate School (1,300 students).

Directions to Brown | Photo Tour | News & Events | Phone Book | What's New? | Search Brown

Talk overview

- History
- Overview of the ZFS storage system
- **How ZFS snapshots work**
- ZFS on-disk structures
- **How ZFS space allocation works**
- **How ZFS RAID-Z works**
- Future work

OpenZFS

Sun microsystems

ORACLE

FreeBSD

Linux

opensolaris

ubuntu

illumos

DELPHIX

iXsystems

intel Software

ubuntu

OSNEXUS

Joyent

Lawrence Livermore National Laboratory

nexenta
Enterprise class storage for everyone

datto

ZFS History

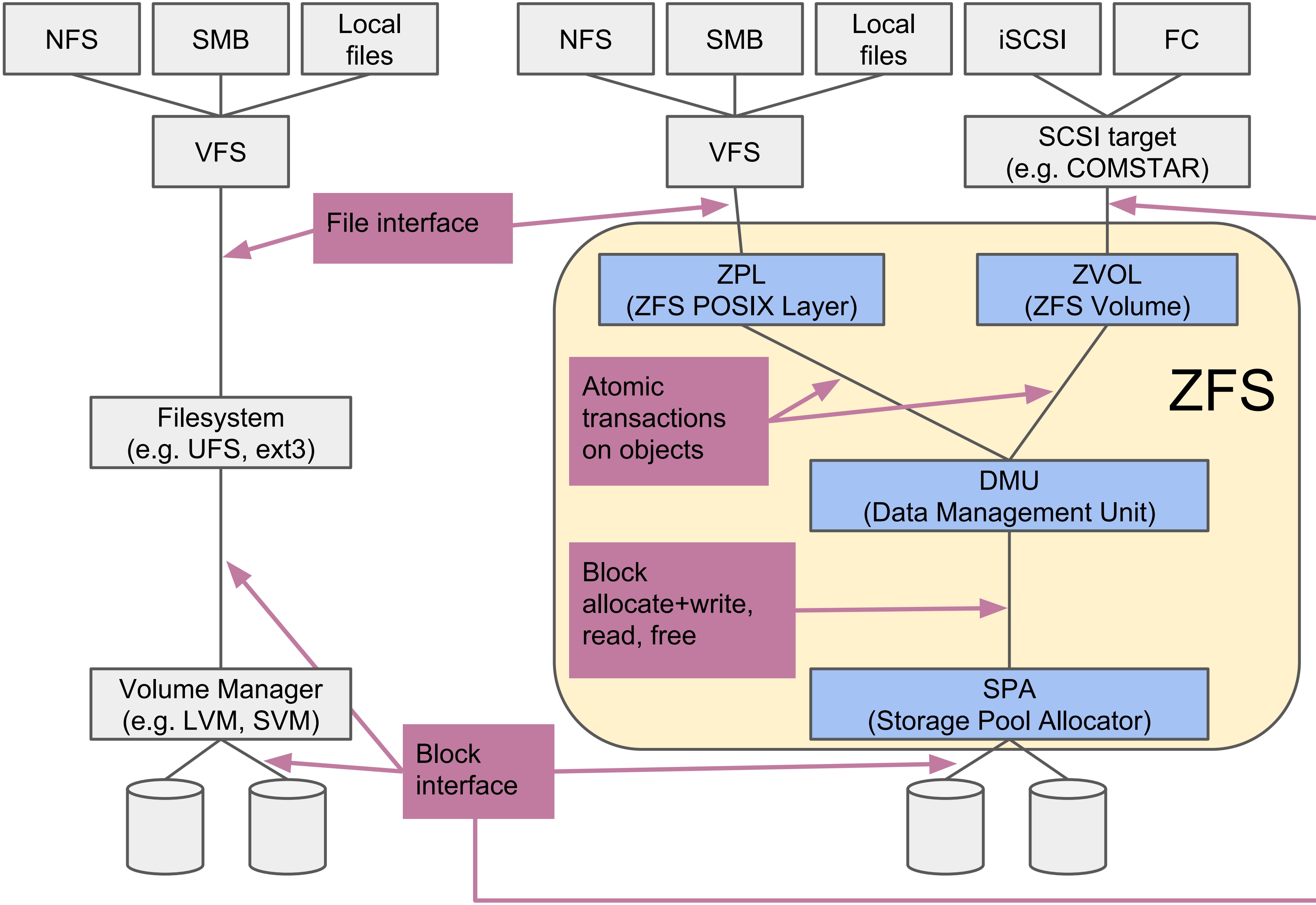
- 2001: development starts at Sun with 2 engineers
- 2005: ZFS source code released
- 2008: ZFS released in FreeBSD 7.0
- 2010: Oracle stops contributing to source code for ZFS
- 2010: illumos is founded as the truly open successor to OpenSolaris
- 2013: ZFS on (native) Linux GA
- 2013: Open-source ZFS bands together to form OpenZFS
- 2014: OpenZFS for Mac OS X launch

Talk overview

- History
- **Overview of the ZFS storage system**
- How ZFS snapshots work
- ZFS on-disk structures
- How ZFS space allocation works
- How ZFS RAID-Z works
- Future work

Overview of ZFS

- Pooled storage
  - Functionality of filesystem + volume manager in one
  - Filesystems allocate and free space from pool
- Transactional object model
  - Always consistent on disk (no FSCK, ever)
  - Universal - file, block, NFS, SMB, iSCSI, FC, …
- End-to-end data integrity
  - Detect & correct silent data corruption
- Simple administration
  - Filesystem is the administrative control point
  - Inheritable properties
  - Scalable data structures

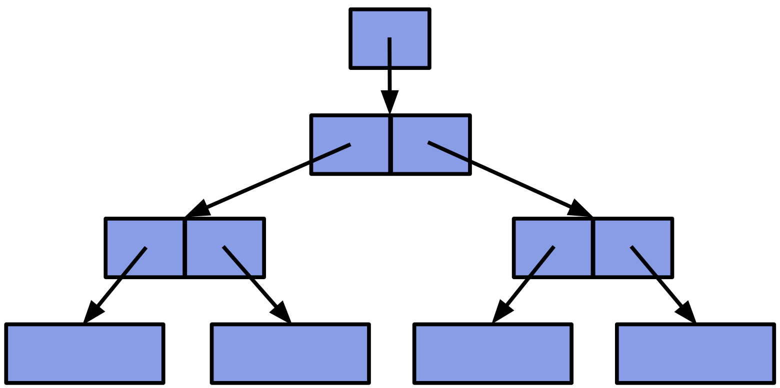


NFS
SMB
Local files
VFS
Filesystem (e.g. UFS, ext3)
Volume Manager (e.g. LVM, SVM)
File interface
Block interface
NFS
SMB
Local files
iSCSI
FC
VFS
SCSI target (e.g. COMSTAR)
ZFS
ZPL (ZFS POSIX Layer)
ZVOL (ZFS Volume)
Atomic transactions on objects
DMU (Data Management Unit)
Block allocate+write, read, free
SPA (Storage Pool Allocator)

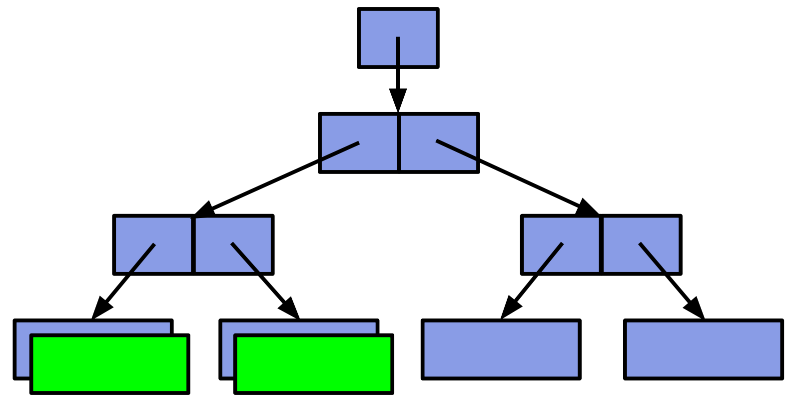
```
zpool create tank raidz2 d1 d2 d3 d4 d5 d6
zfs create tank/home
zfs set sharenfs=on tank/home
zfs create tank/home/mahrens
zfs set reservation=10T tank/home/mahrens
zfs set compression=gzip tank/home/dan
zpool add tank raidz2 d7 d8 d9 d10 d11 d12
zfs create -o recordsize=8k tank/DBs
zfs snapshot -r tank/DBs@today
zfs clone tank/DBs/prod@today tank/DBs/test
```

Copy-On-Write Transaction Groups (TXG’s)

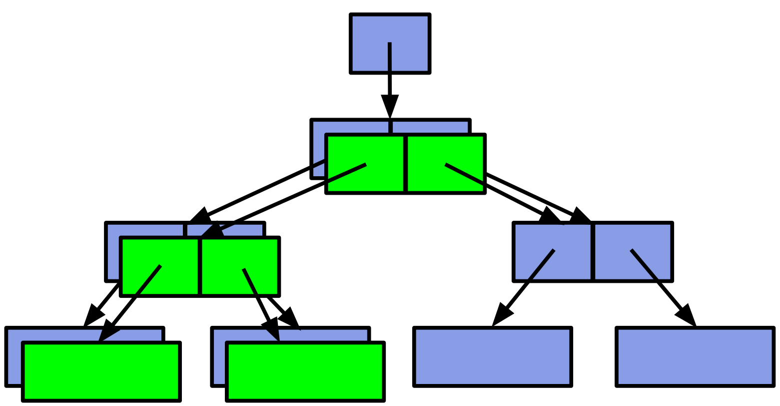
1. Initial block tree



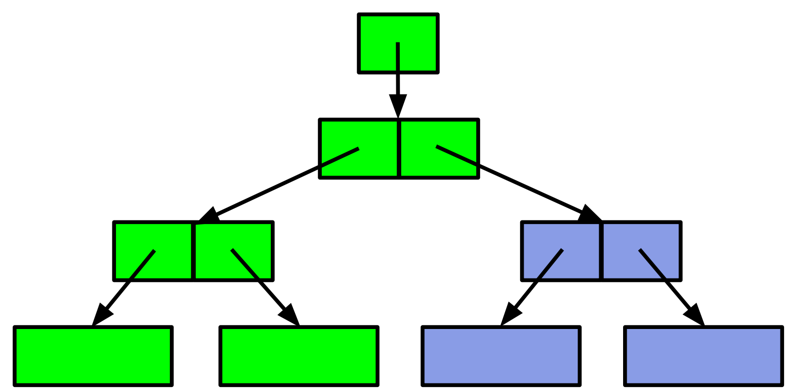
2. COW some blocks



3. COW indirect blocks

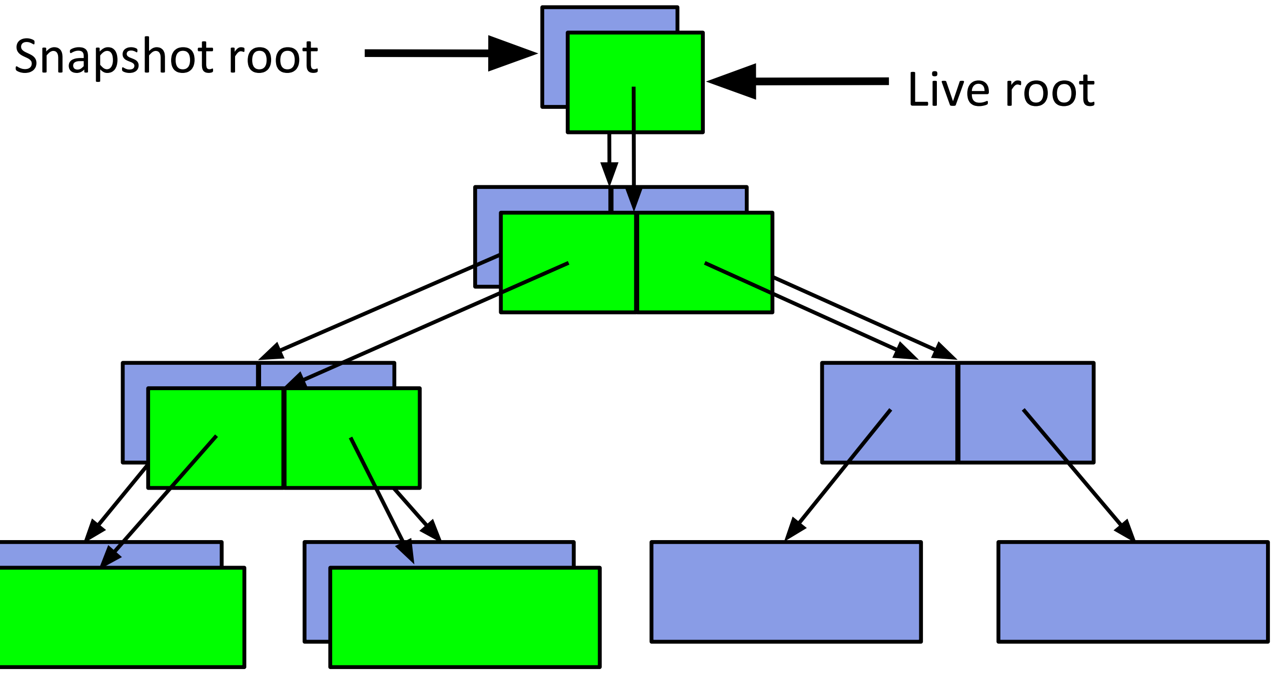


4. Rewrite uberblock (atomic)



Bonus: Constant-Time Snapshots

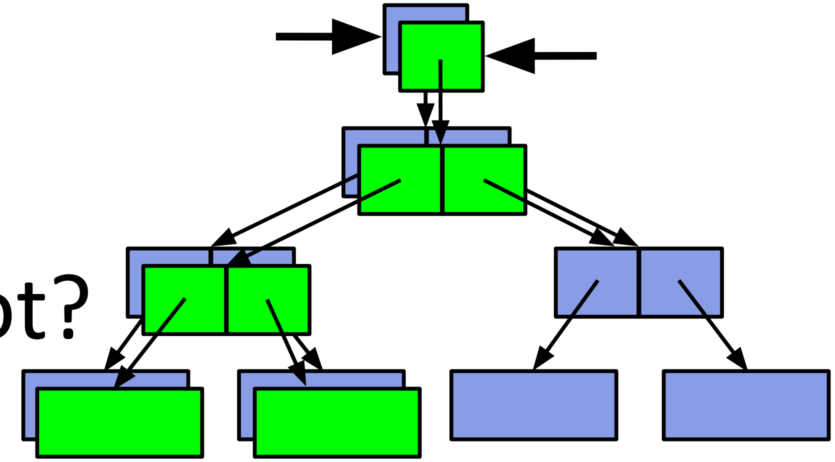
- The easy part: at end of TX group, don't free COWed blocks



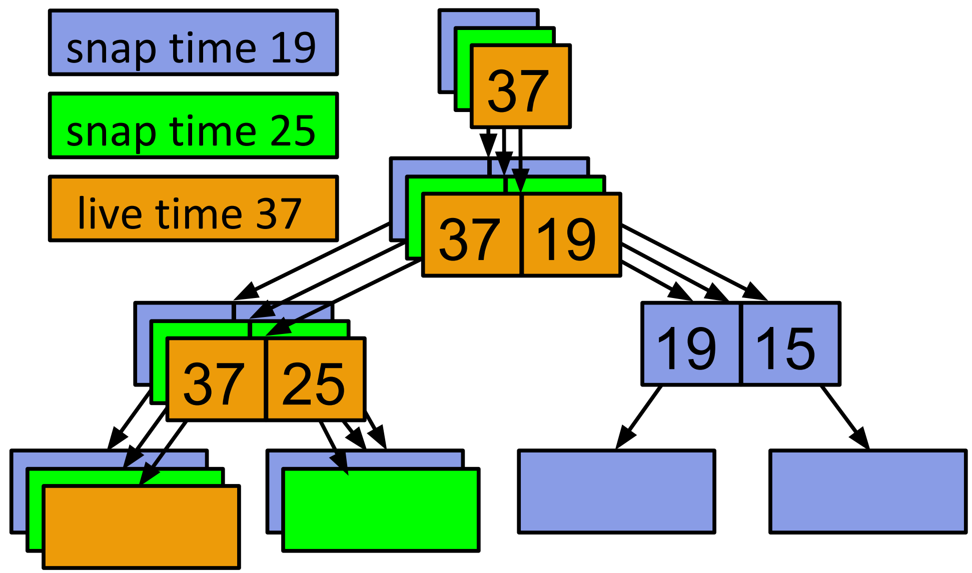
Talk overview

- History
- Overview of the ZFS storage system
- **How ZFS snapshots work**
- ZFS on-disk structures
- How ZFS space allocation works
- How ZFS RAID-Z works
- Future work

ZFS Snapshots



- How to create snapshot?
  - Save the root block
- When block is removed, can we free it?
  - Use BP's birth time
  - If birth > prevsnap
    - Free it

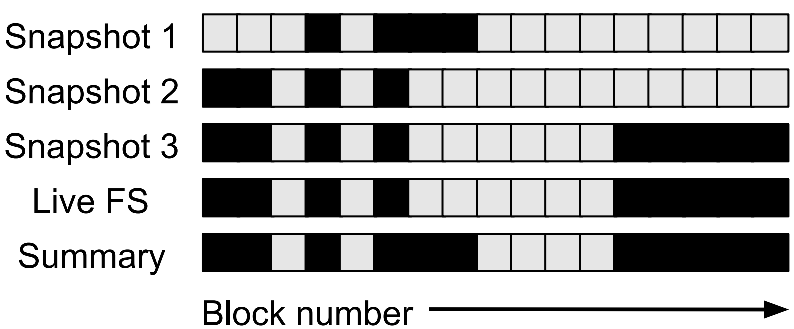


- When delete snapshot, what to free?
  - Find unique blocks - Tricky!

Trickiness will be worth it!

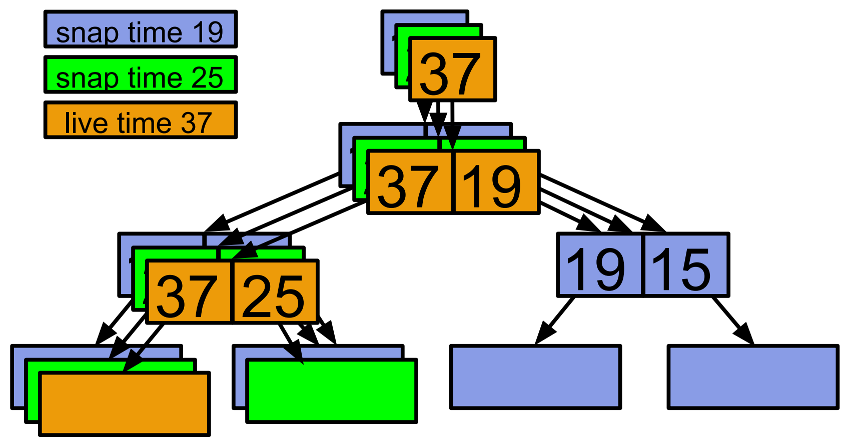
Per-Snapshot Bitmaps

- Block allocation bitmap for every snapshot
  - O(N) per-snapshot space overhead
  - Limits number of snapshots
- O(N) create, O(N) delete, O(N) incremental
  - Snapshot bitmap comparison is O(N)
  - Generates unstructured block delta
  - Requires some prior snapshot to exist



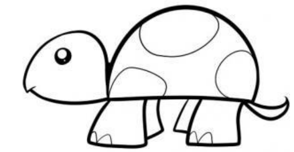
ZFS Birth Times

- Each block pointer contains child's birth time
  - O(1) per-snapshot space overhead
  - Unlimited snapshots
- O(1) create, O(Δ) delete, O(Δ) incremental
  - Birth-time-pruned tree walk is O(Δ)
  - Generates semantically rich object delta
  - Can generate delta since any point in time

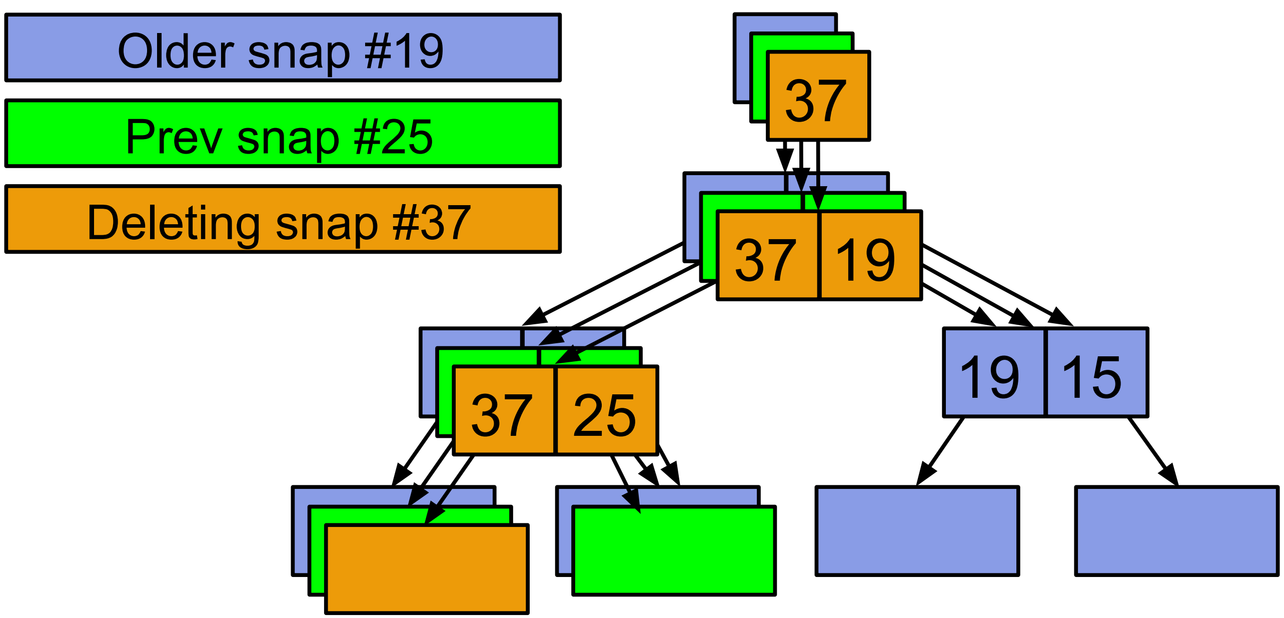


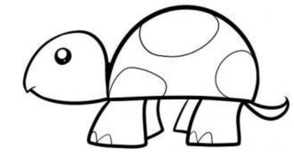
Snapshot Deletion

- Free unique blocks (ref'd only by this snap)
- Optimal algo: O(# blocks to free)
  - And # blocks to read from disk << # blocks to free
- Block lifetimes are contiguous
  - AKA "there is no afterlife"
  - Unique = not ref'd by prev or next (ignore others)

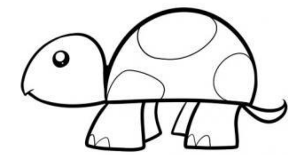
Snapshot Deletion ()

- Traverse tree of blocks
- Birth time <= prev snap?
  - Ref'd by prev snap; do not free.
  - Do not examine children; they are also <= prev



Snapshot Deletion ()

- Traverse tree of blocks
- Birth time <= prev snap?
  - Ref'd by prev snap; do not free.
  - Do not examine children; they are also <= prev
- Find BP of same file/offset in next snap
  - If same, ref'd by next snap; do not free.
- O(# blocks written since prev snap)
- How many blocks to read?
  - Could be 2x # blocks written since prev snap

Snapshot Deletion ()

- Read Up to 2x # blocks written since prev snap
- Maybe you read a million blocks and free nothing
  - (next snap is identical to this one)
- Maybe you have to read 2 blocks to free one
  - (only one block modified under each indirect)
- RANDOM READS!
  - 200 IOPS, 8K block size -> free 0.8 MB/s
  - Can write at ~200MB/s

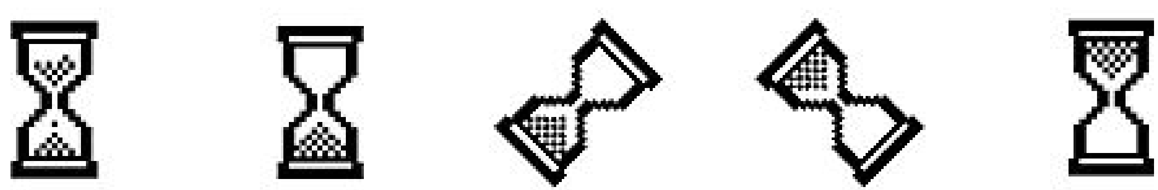
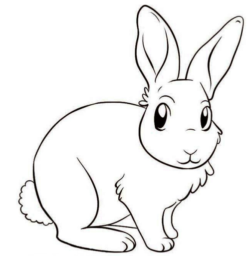
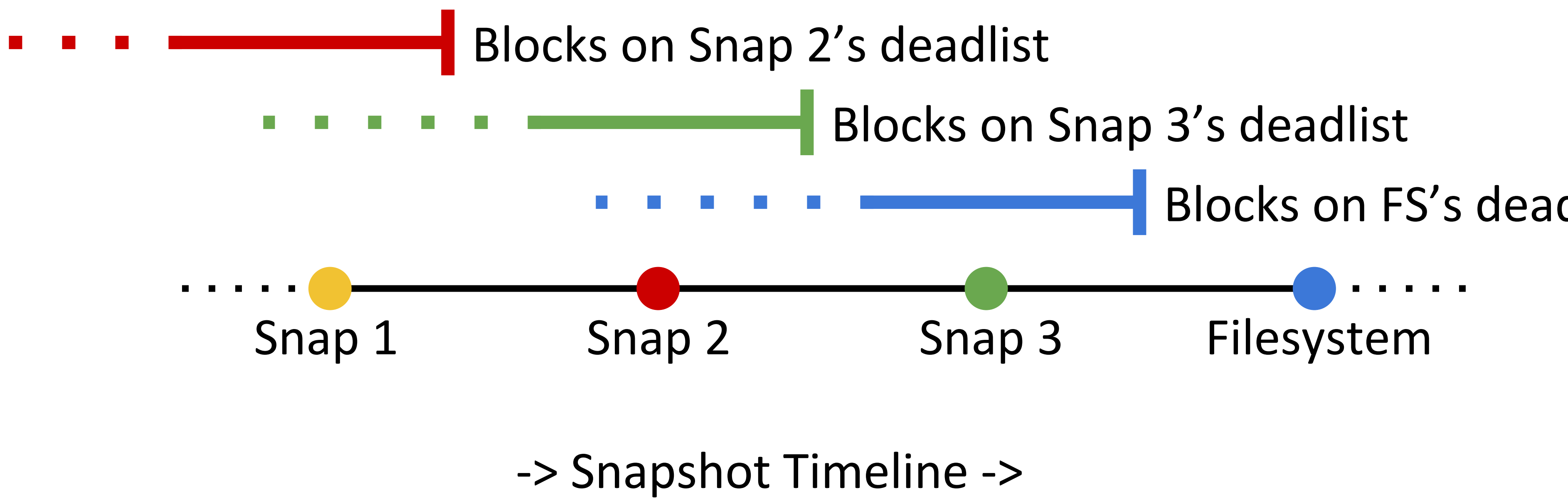
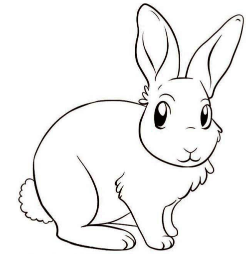


FIGURE 131. Hourglass

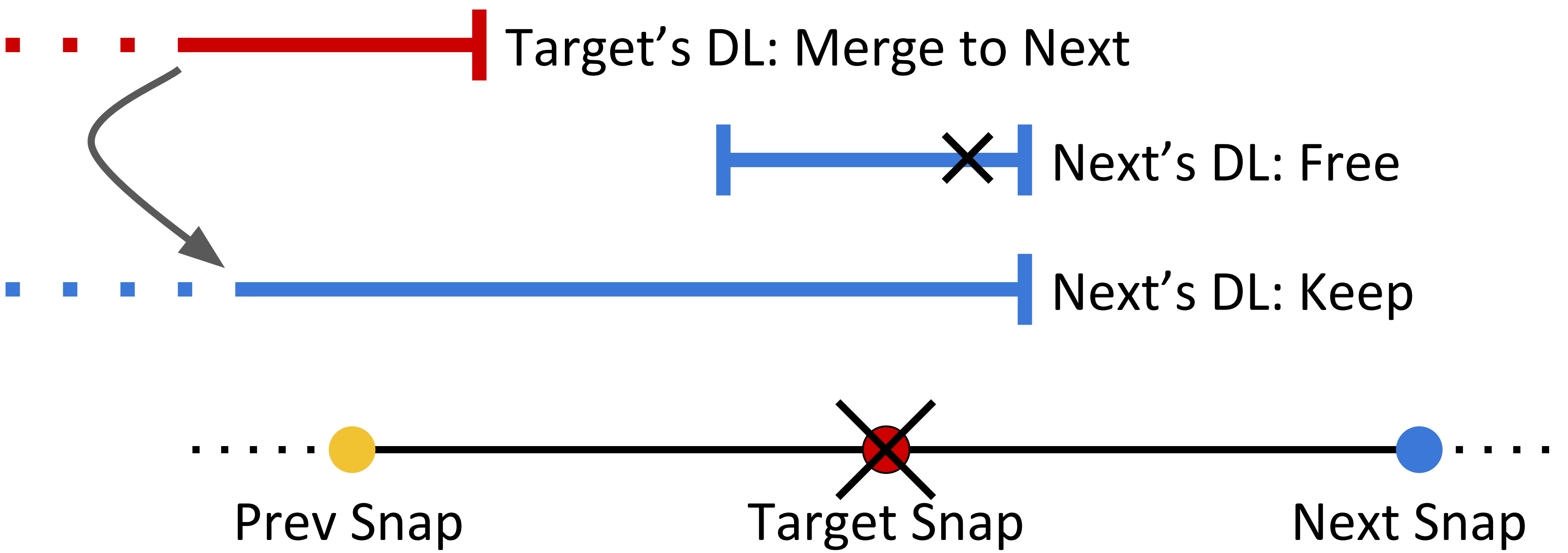
Snapshot Deletion ()

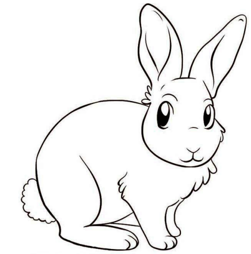
- Keep track of no-longer-referenced (“dead”) blocks
- Each dataset (snapshot & filesystem) has “dead list”
  - On-disk array of block pointers (BP’s)
  - blocks ref’d by prev snap, not ref’d by me

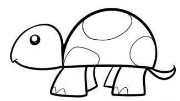


Snapshot Deletion ()

- Traverse next snap’s deadlist
- Free blocks with birth > prev snap



Snapshot Deletion ()

- O(size of next's deadlist)
  - = O(# blocks deleted before next snap)
  - Similar to  (# deleted ~= # created)
- Deadlist is compact!
  - 1 read = process 1024 BP's
  - Up to 2048x faster than Algo 1!
- Could still take a long time to free nothing

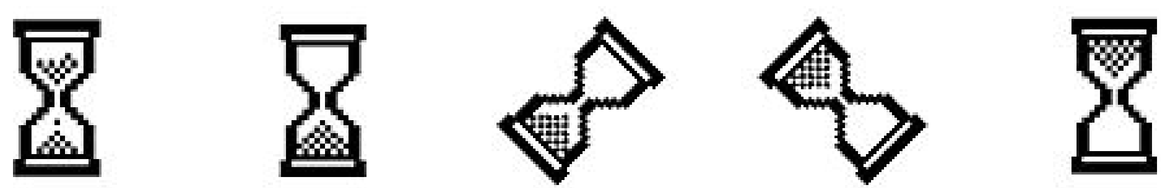
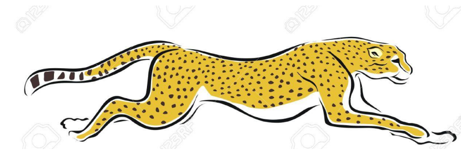
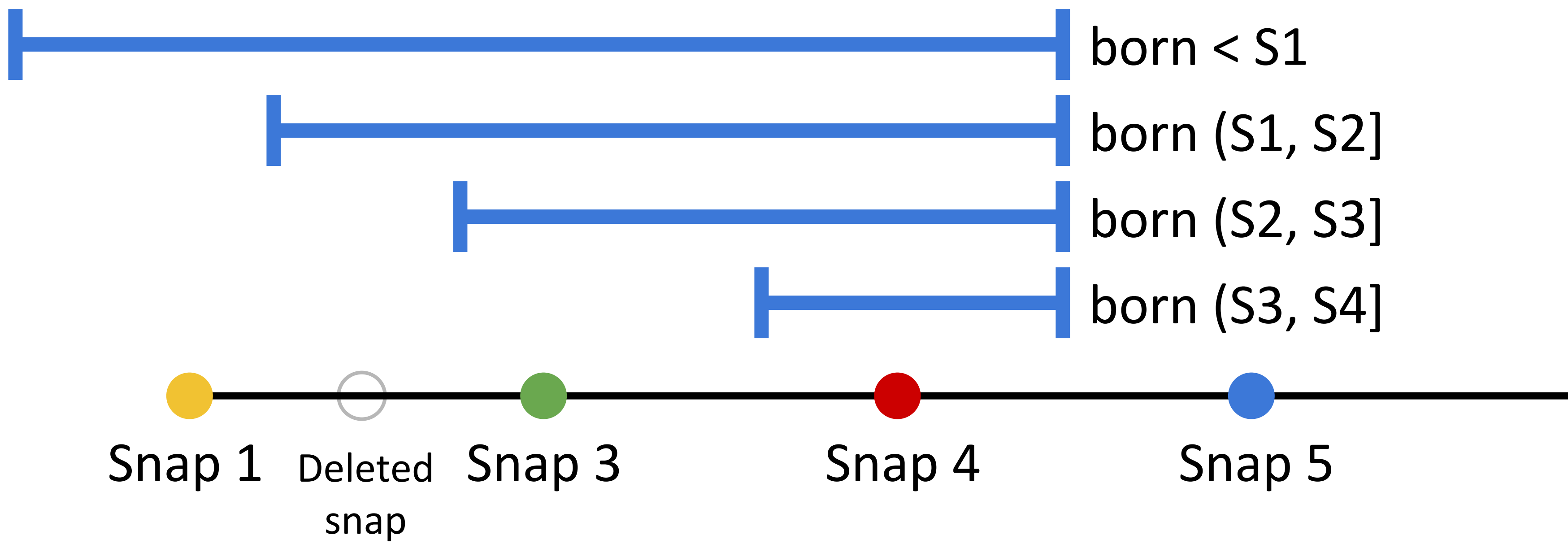
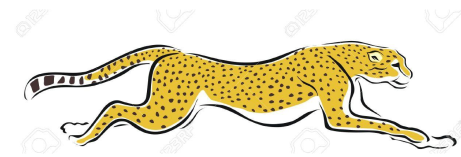


FIGURE 131. Hourglass

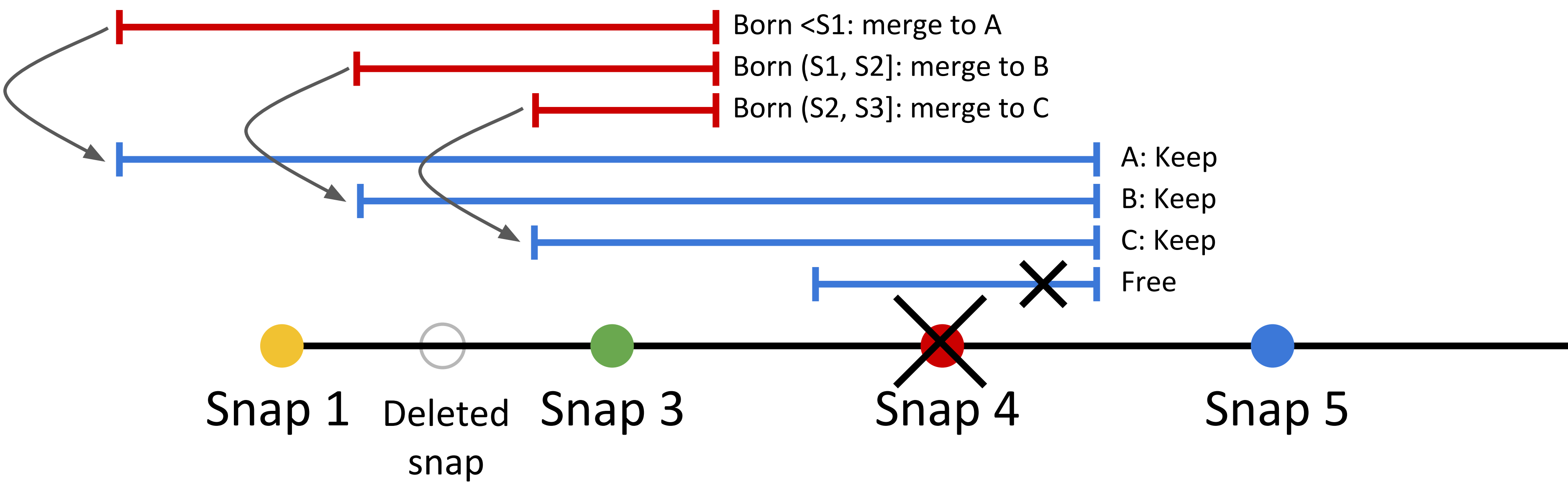
Snapshot Deletion ()

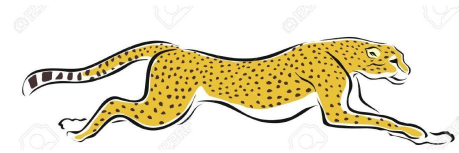
- Divide deadlist into sub-lists based on birth time
- One sub-list per earlier snapshot
  - Delete snapshot: merge FS’s sublists



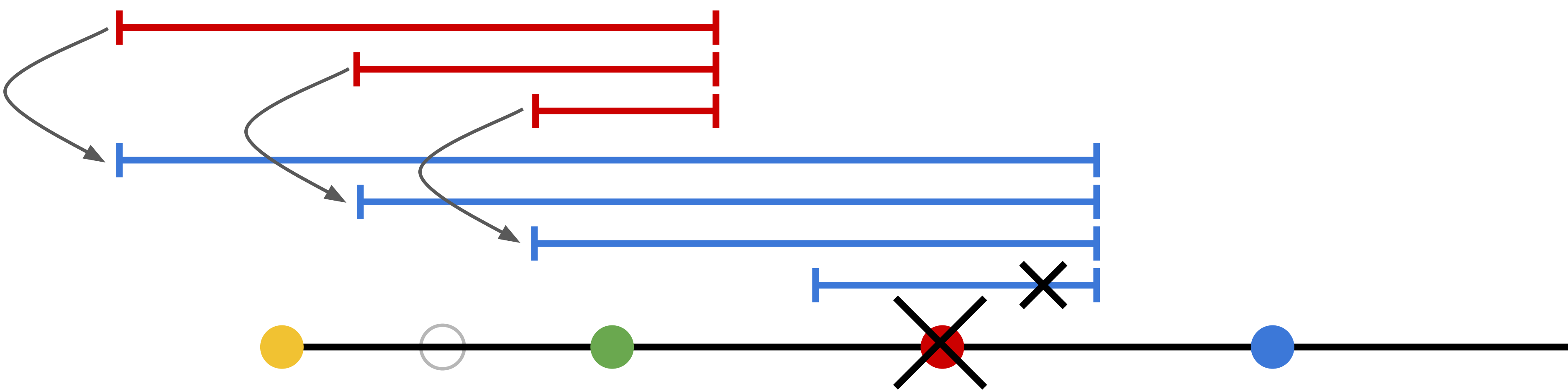
Snapshot Deletion ()

- Iterate over sublists
- If mintxg > prev, free all BP’s in sublist
- Merge target’s deadlist into next’s
  - Append sublist by reference -> O(1)



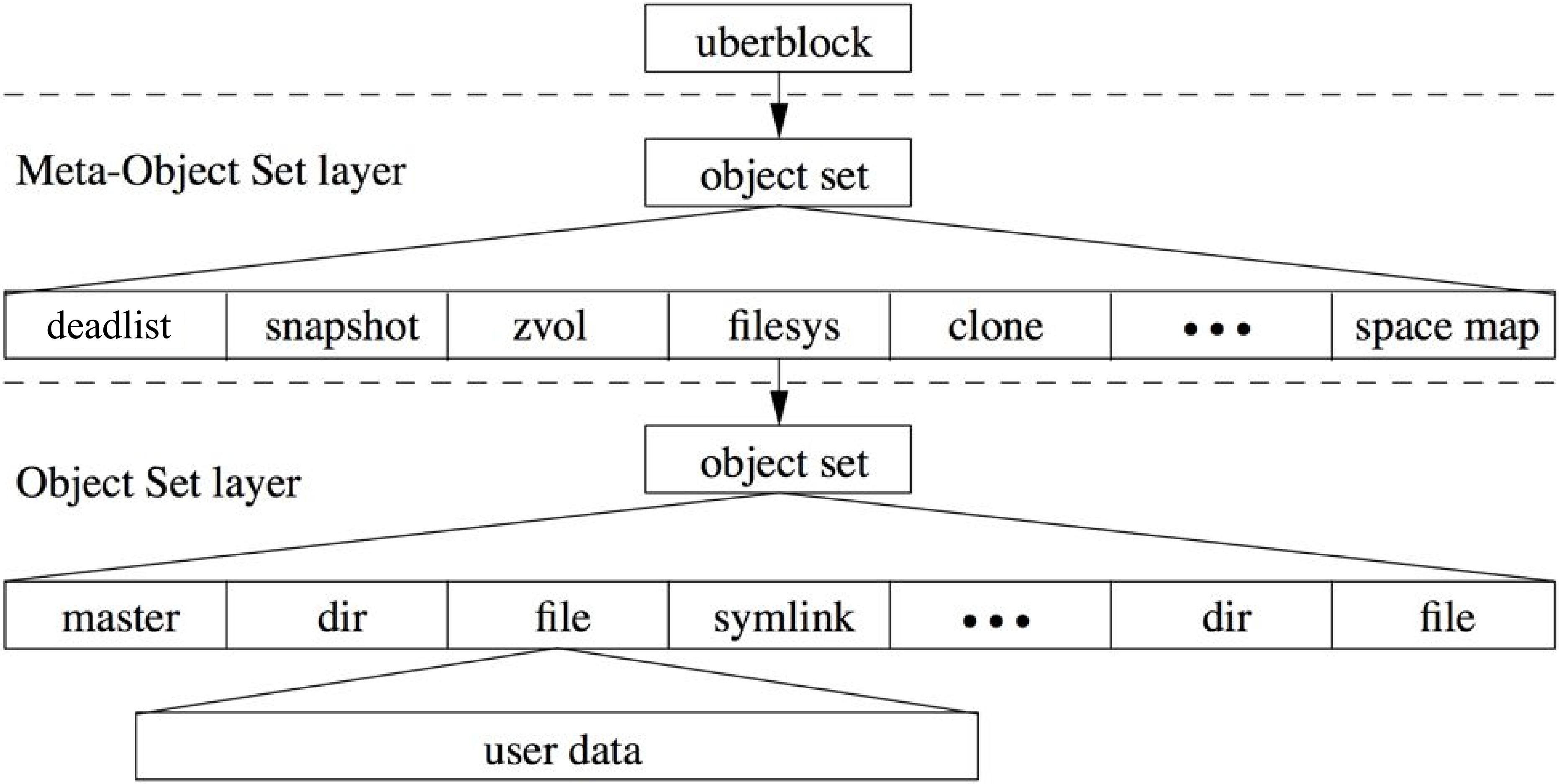
Snapshot Deletion ()

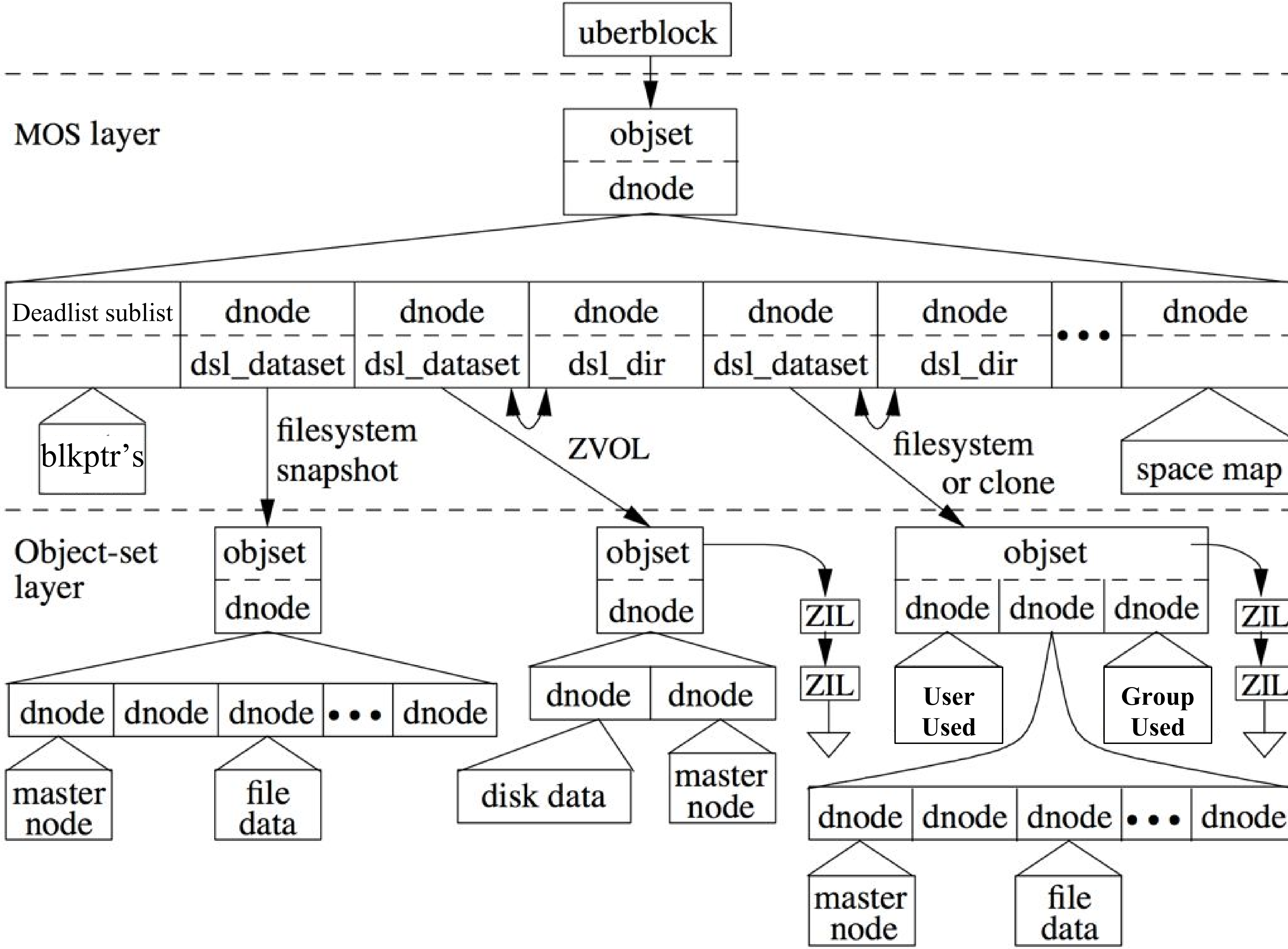
- Deletion: O(# sublists + # blocks to free)
    - 200 IOPS, 8K block size -> free 1500MB/sec
- Optimal: O(# blocks to free)
- # sublists = # snapshots present when snap created
- # sublists << # blocks to free



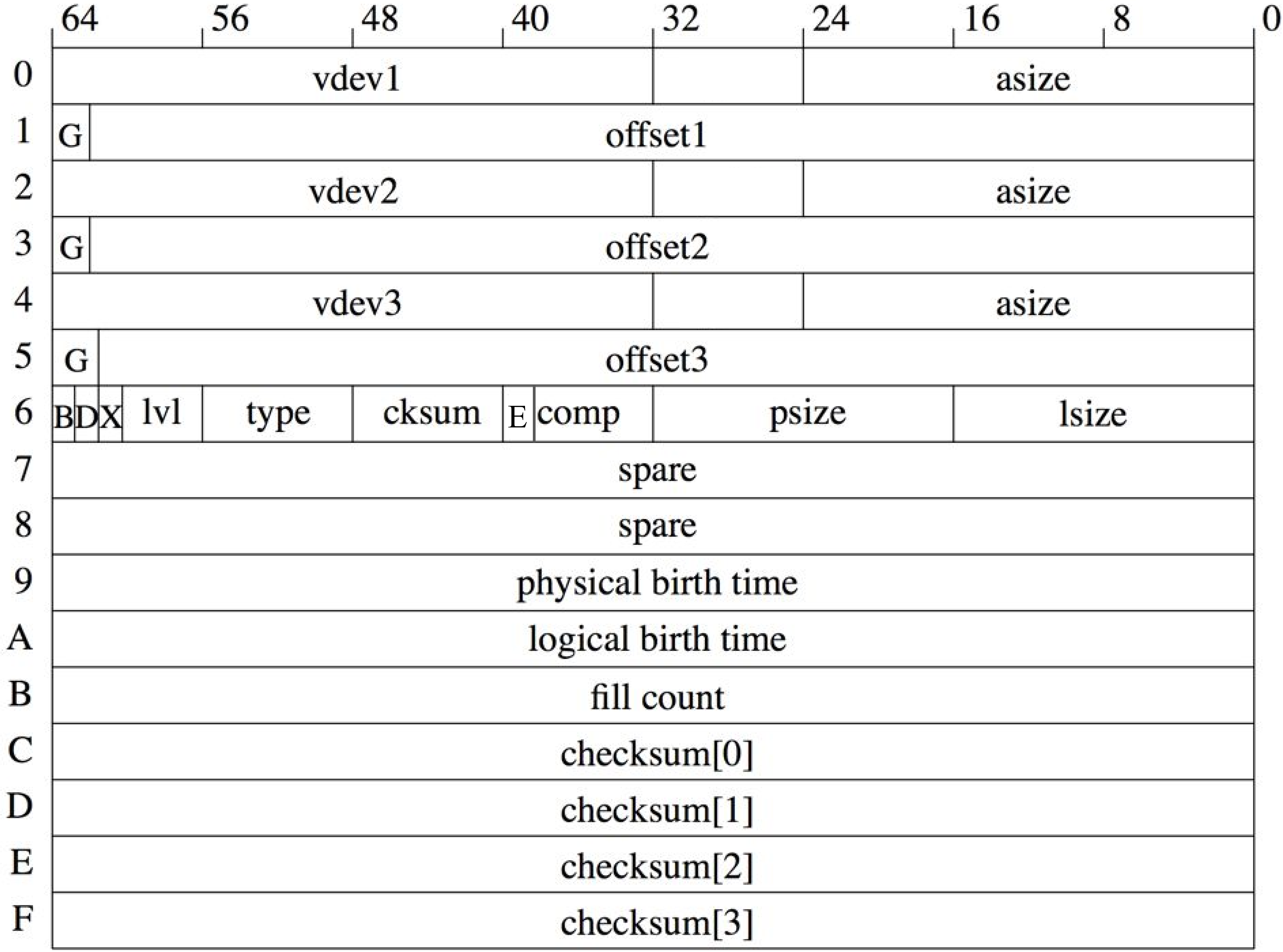
Talk overview

- History
- Overview of the ZFS storage system
- How ZFS snapshots work
- **ZFS on-disk structures**
- How ZFS space allocation works
- How ZFS RAID-Z works
- Future work

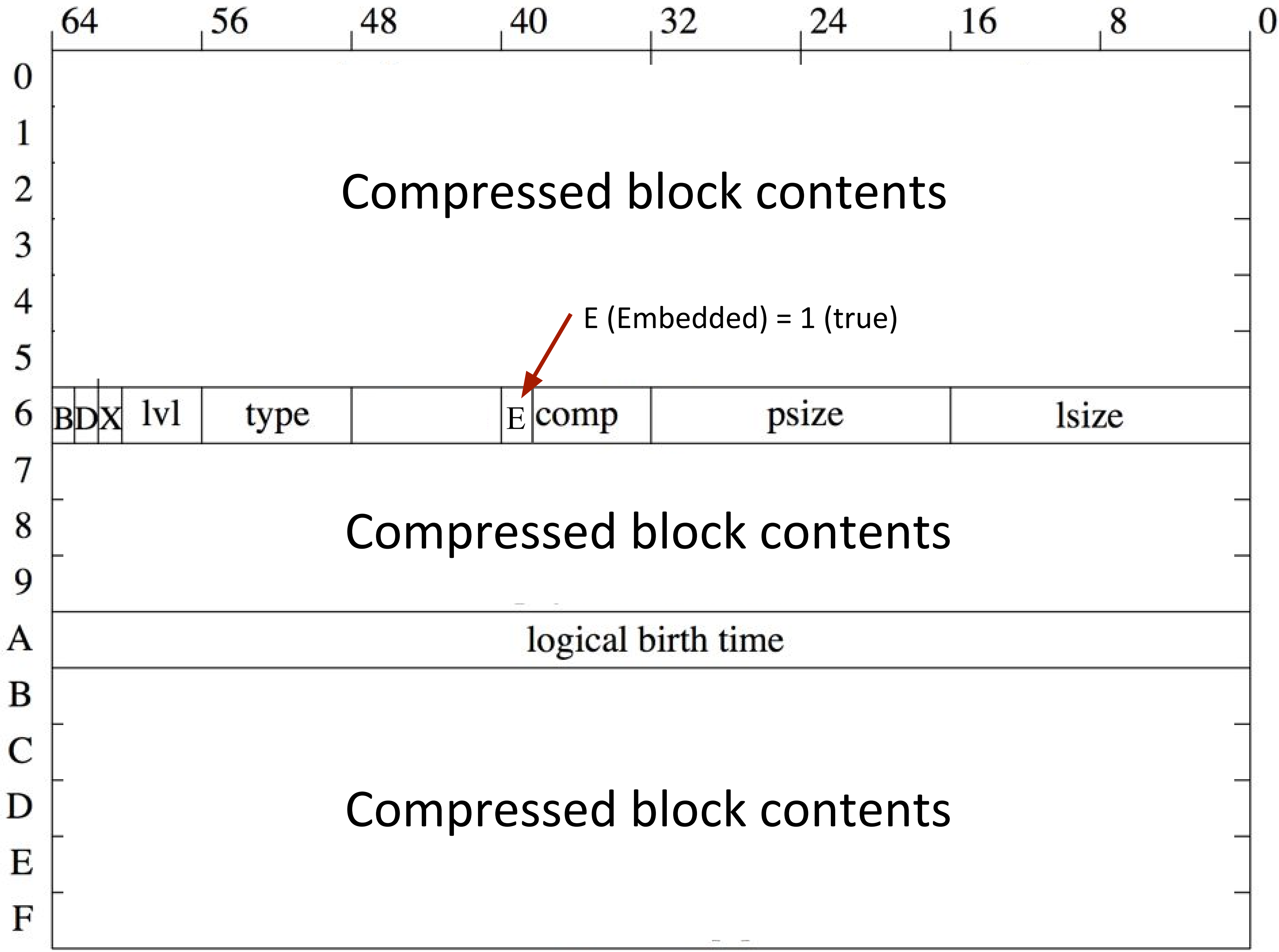

uberblock
Meta-Object Set layer
object set
deadlist
snapshot
zvol
filesys
clone
• • •
space map
Object Set layer
object set
master
dir
file
symlink
• • •
dir
file
user data



uberblock
MOS layer
objset
dnode
Deadlist sublist
dnode
dsl_dataset
dnode
dsl_dataset
dnode
dsl_dir
dnode
dsl_dataset
dnode
dsl_dir
• • •
dnode
blkptr's
filesystem snapshot
ZVOL
filesystem or clone
space map
Object-set layer
objset
dnode
objset
dnode
ZIL
ZIL
objset
dnode
dnode
dnode
ZIL
ZIL
dnode
dnode
dnode
• • •
dnode
master node
file data
dnode
dnode
disk data
master node
User Used
Group Used
dnode
dnode
dnode
• • •
dnode
master node
file data



64
56
48
40
32
24
16
8
0
0
vdev1
asize
1
G
offset1
2
vdev2
asize
3
G
offset2
4
vdev3
asize
5
G
offset3
6
B
D
X
lvl
type
cksum
E
comp
psize
lsize
7
spare
8
spare
9
physical birth time
A
logical birth time
B
fill count
C
checksum[0]
D
checksum[1]
E
checksum[2]
F
checksum[3]



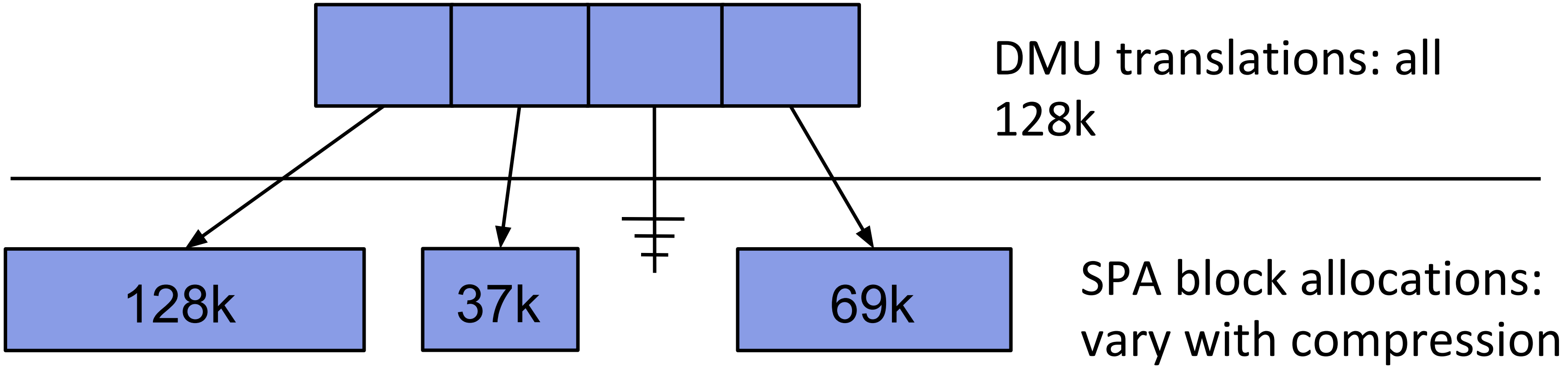
64
56
48
40
32
24
16
8
0
0
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F
Compressed block contents
E (Embedded) = 1 (true)
B
D
X
lvl
type
E
comp
psize
lsize
Compressed block contents
logical birth time
Compressed block contents

Talk overview

- History
- Overview of the ZFS storage system
- How ZFS snapshots work
- ZFS on-disk structures
- **How ZFS space allocation works**
- How ZFS RAID-Z works
- Future work

Built-in Compression

- Block-level compression in SPA
  - Transparent to other layers
  - Each block compressed independently
  - All-zero blocks converted into file holes



- Choose between LZ4, gzip, and specialty algorithms

Space Allocation

- Variable block size
  - Pro: transparent compression
  - Pro: match database block size
  - Pro: efficient metadata regardless of file size
  - Con: variable allocation size
- Can’t fit all allocation data in memory at once
  - Up to ~3GB RAM per 1TB disk
- Want to allocate as contiguously as possible

On-disk Structures

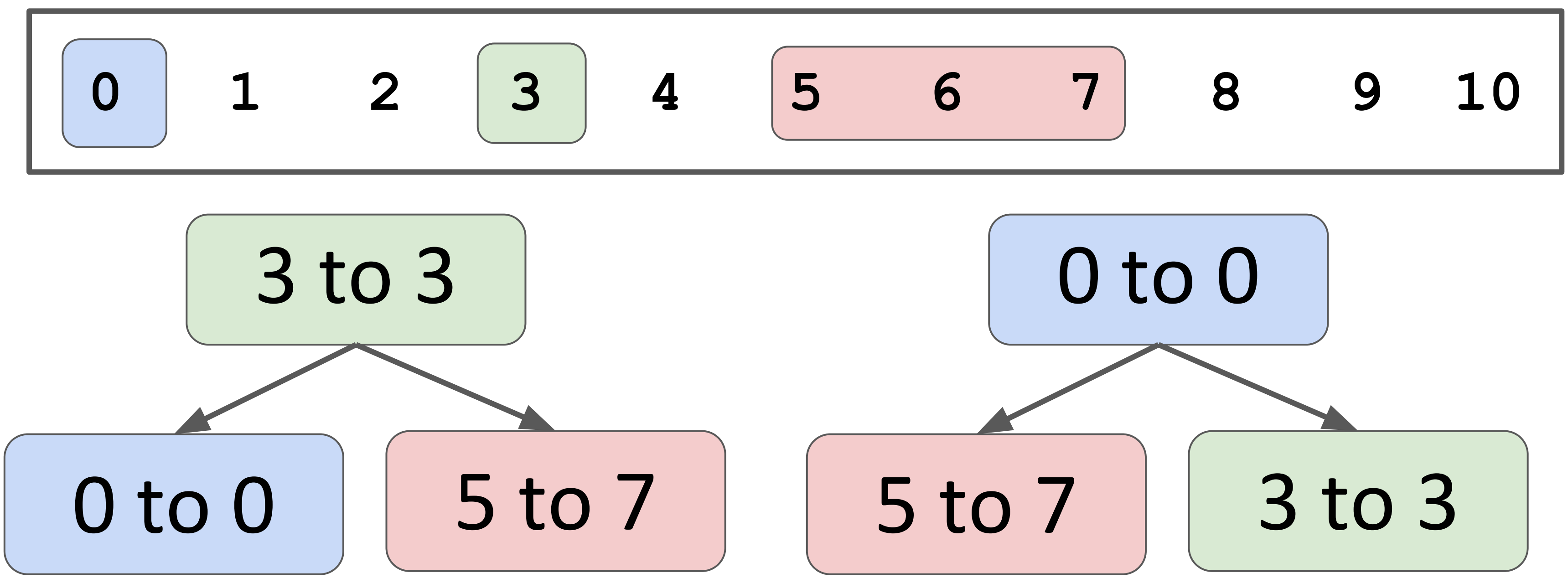
- Each disk divided into ~200 “metaslabs”
    - Each metaslab tracks free space in on-disk spacemap
- Spacemap is on-disk **log** of allocations & frees

Alloc 0 to 10	Free 0 to 10	Alloc 4 to 7	Alloc 2 to 2	Free 5 to 7	Alloc 8 to 10	Alloc 1 to 1

- Each spacemap stored in object in MOS
- Grows until rewrite (by “condensing”)

Allocation

- Load spacemap into `allocatable` range tree
- range tree is in-memory structure
  - balanced binary tree of free segments, sorted by offset
    - So we can consolidate adjacent segments
  - 2nd tree sorted by length
    - So we can allocate from largest free segment

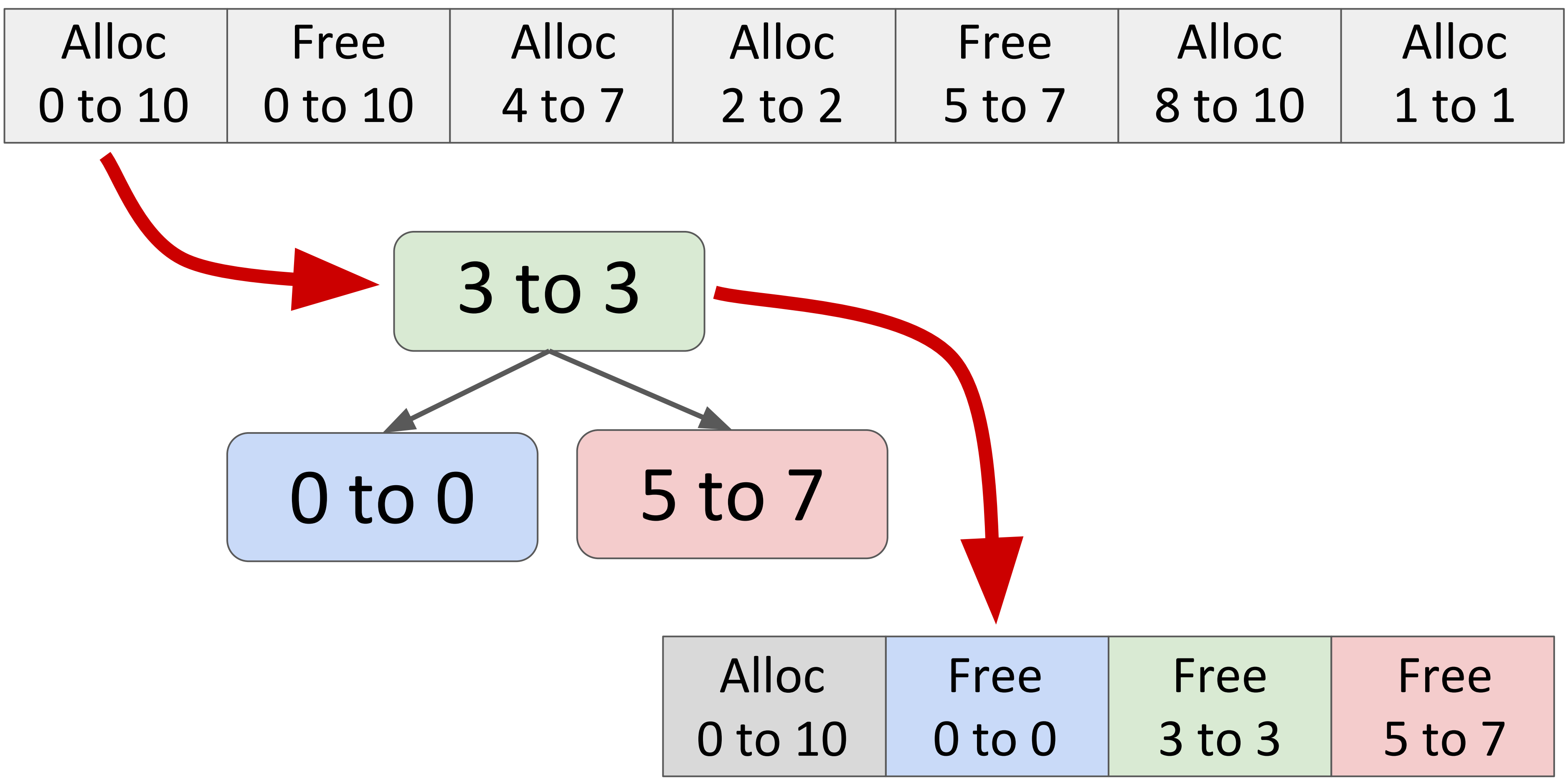


Writing Spacemaps

- While syncing TXG, each metaslab tracks
  - allocations (in the `allocating` range tree)
  - frees (in the `freeing` range tree)
- At end of TXG
  - append alloc & free range trees to space_map
  - clear range trees
- Can free from metaslab when not loaded
- Spacemaps stored in MOS
  - Sync to convergence

Condensing

- Condense when it will halve the # entries
    - Write `allocatable` range tree to new SM



Talk overview

- History
- Overview of the ZFS storage system
- How ZFS snapshots work
- ZFS on-disk structures
- How ZFS space allocation works
- **How ZFS RAID-Z works**
- Future work

Traditional RAID (4/5/6)

- Stripe is physically defined
- Partial-stripe writes are awful
  - 1 write -> 4 i/o's (read & write of data & parity)
  - Not crash-consistent
    - "RAID-5 write hole"
    - Entire stripe left unprotected
      - (including unmodified blocks)
    - Fix: expensive NVRAM + complicated logic

RAID-Z

- Single, double, or triple parity
- Eliminates “RAID-5 write hole”
- No special hardware required for best perf
- How? No partial-stripe writes.

RAID-Z: no partial-stripe writes

- Always consistent!
- Each block has its own parity
- Odd-size blocks use slightly more space
- Single-block reads access all 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	•	•	•

Talk overview

- History
- Overview of the ZFS storage system
- How ZFS snapshots work
- ZFS on-disk structures
- How ZFS space allocation works
- How ZFS RAID-Z works
- **Future work**

DEVELOPER SUMMIT

OpenZFS

2016

Future work

- Easy to manage on-disk encryption
- Channel programs
  - Compound administrative operations
- Vdev spacemap log
  - Performance of large/fragmented pools
- Device removal
  - Copy allocated space to other disks

Further reading

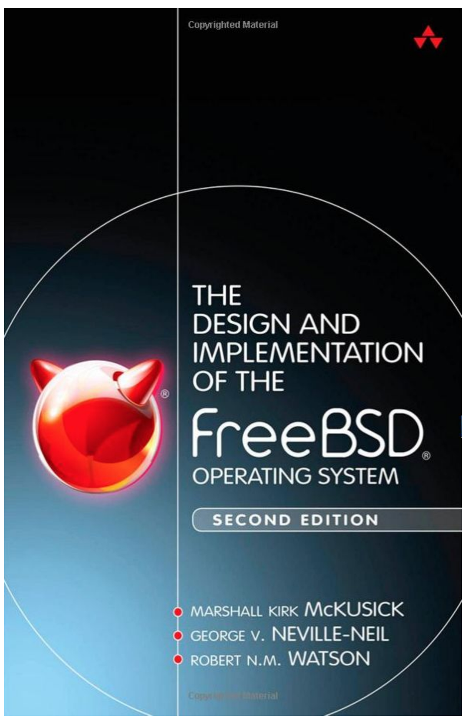
http://www.open-zfs.org/wiki/Developer_resources

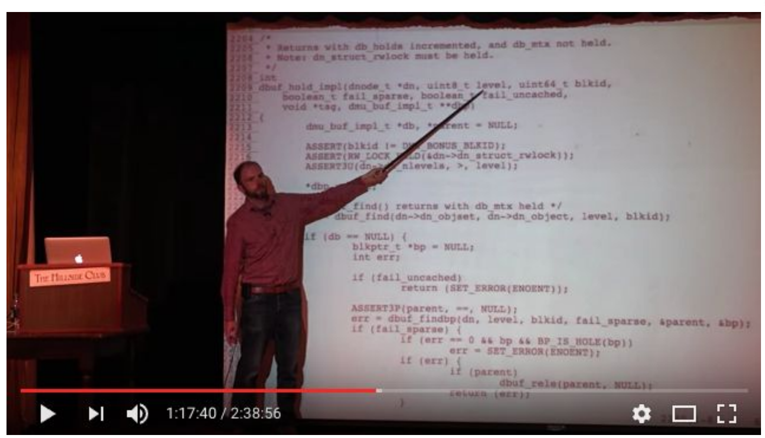
Specific Features

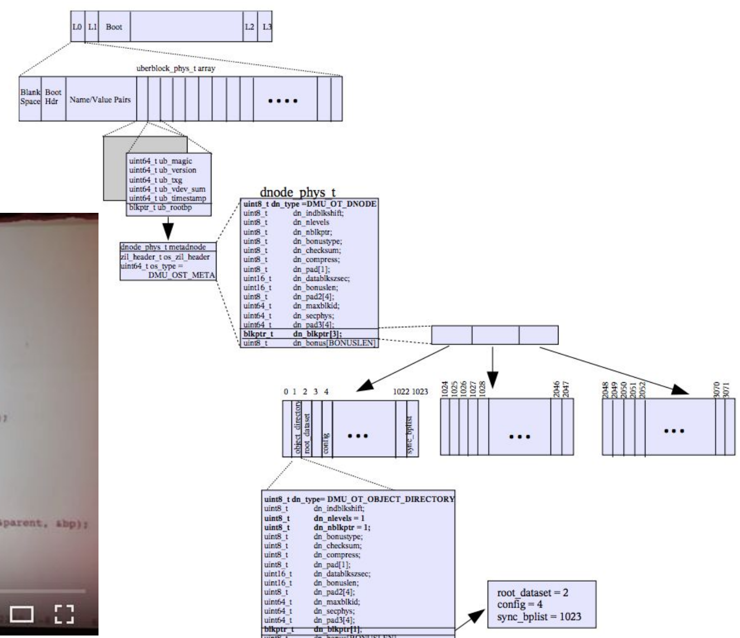
- Space allocation video (slides) - Matt Ahrens ‘01
- Replication w/ send/receive video (slides)
  - Dan Kimmel ‘12 & Paul Dagnelie
- Caching with compressed ARC video (slides) - George Wilson
- Write throttle blog 1 2 3 - Adam Leventhal ‘01
- Channel programs video (slides)
  - Sara Hartse ‘17 & Chris Williamson
- Encryption video (slides) - Tom Caputi
- Device initialization video (slides) - Joe Stein ‘17
- Device removal video (slides) - Alex Reece & Matt Ahrens

Further reading: overview

- Design of FreeBSD book - Kirk McKusick
- Read/Write code tour video - Matt Ahrens
- Overview video (slides) - Kirk McKusick
- ZFS On-disk format pdf - Tabriz Leman / Sun Micro







Community / Development

- History of ZFS features video - Matt Ahrens
- Birth of ZFS video - Jeff Bonwick
- OpenZFS founding paper - Matt Ahrens

OpenZFS

http://openzfs.org

Matt Ahrens
Principal Engineer, Delphix
ZFS co-creator
Brown CS 2001