ZFS Internal Structure

Ulrich Gräf
Senior SE
Sun Microsystems

ZFS – Filesystem of a New Generation

- Integrated Volume Manager
- Transactions for every change on the Disk
- Checksums for everything
- Self Healing
- Simplified Administration
  - Also accelerated
  - Changes online
- Performance through Controll of Datapath

Everything new? No!

But new in this combination!

Another explanation why using ZFS

Current Trends in Datacenters

- Larger filesystems
- Data lives longer on disks
- Backup devices are sufficient
- Enough devices for Restore: Expensive
- Backups are complemented by copies on disk
- Copies on disks are more vulnerable to failures

ZFS and failures

ZFS can correct structural errors caused by

- Bit errors (1 sectorin 10^16 reads)
- Errors caused by mis-positioning
  - Phantom writes
  - Misdirected reads
  - Misdirected writes
- DMA parity errors
- Bugs in software and firmware
- Administration errors

ZFS Self Healing

Elements:

- Integrated Volume Manager
- (Large!) Checksums inside of Block Pointer

How does it work?

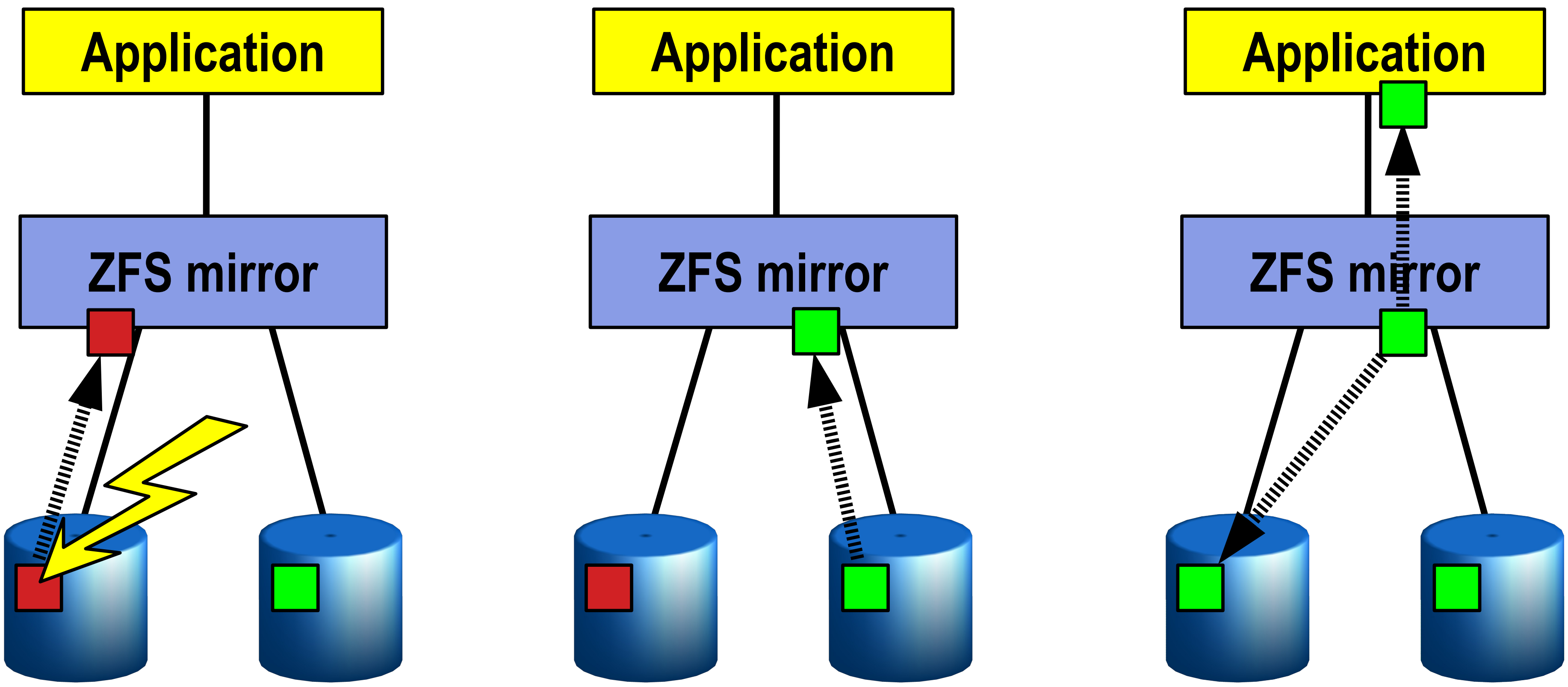
- Read a block determined by Block Pointer
- Create a checksum
- Compare it with checksum in Block Pointer
- On Error: use/compute block (redundancy)

Structural Integrity (remember: Star Trek)

ZFS Self Healing

- Is different from other filesystems
- Is a quality not available from other filesystems
- Is only possible when combining
  - Integrated Volume Manager
  - Redundant Setup
  - Large Checksums
- Is not available on Reiser*, ext3/ext4, WAFL, xfs
- Will be available on btrfs, when it is finished (but not all other ZFS features)

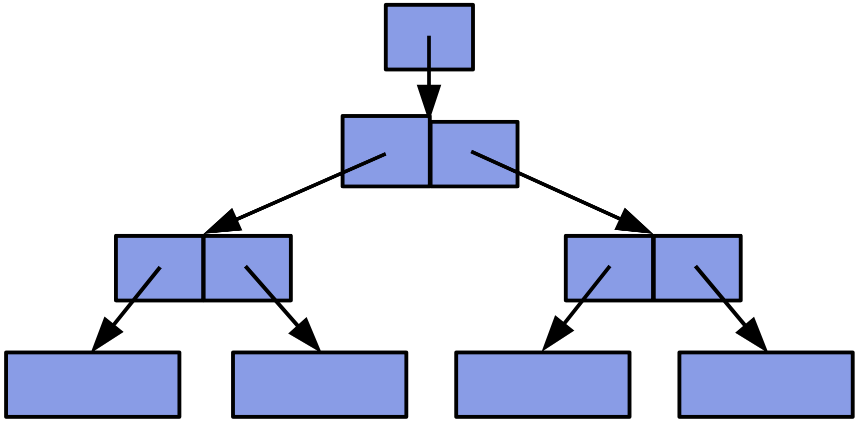
ZFS Self Healing



ZFS Structure

ZFS Structure:

- Uberblock
- Tree with Block Pointers
- Data only in leaves



ZFS Structure: *vdev*

A ZFS pool (zpool) is built from

- Whole disks
- Disk partitions
- Files

… called *physical vdev*

ZFS Structure: Configuration

Configuration can be

- Single device
- Mirrored (mirror)
- RAID-5/RAID-6 (raidz, raidz2)
- Recently: raidz3 (raidz*n* is in planning)

ZFS: *physical vdev*

Each *physical vdev* contains

- 4 *vdev labels* (256 KB each)
  - 2 labels at the beginning
  - 2 labels at the end
- A 3.5 MB hole for boot code
- 128kb blocks for data of the zpool

ZFS: *vdev label*

A *vdev label* contains 3 parts

- gap (avoid conflicts with disk labels)
- nvlist (name – value pair list) (128KB)
  - Attributes of the zpool
  - Including the configuration of the zpool
- uberblock array (128 entries, each 1KB)

Configuration also defines *logical vdevs*

- mirror or raidz, log and cache devices

ZFS: nvlist in a *vdev label* (1)

```
$ zdb -v -v data
version=4
    name='data'
    state=0
    txg=162882
    pool_guid=1442865571463645041
    hostid=13464466
    hostname='nunzio'
    vdev_tree ...
```

ZFS: nvlist in a *vdev label* (2)

```
vdev_tree
    type='root'
    Id=0
    guid=14428655714636450 41
    children[0]
            type='disk'
            id=0
            guid=15247716718277951357
            path='/dev/dsk/c1t0d0s7'
            devid='id1,sd@SATA_____SAMSUNG_HM251JJ_______S1J...
            phys_path='/pci@0,0/pci1179,1@1f,2/disk@0,0:h'
            whole_disk=0
            metaslab_array=14
            metaslab_shift=27
            ashift=9
            asize=25707413504
            is_log=0
```

ZFS: *uberblock*

Verification

- Magic number (0x00bab1oc) for endianess
- Version
- Transaction Group number
- Time-stamp
- Checksum

Content:

- Pointer to the root of the zpool tree

ZFS: *uberblock:* Example

```
$ zdb -v -v data
...
Uberblock

        magic = 0000000000bab10c
        version = 4
        txg = 262711
        guid_sum = 16690582289741596398
        timestamp = 1256864671 UTC = Fri Oct
23 12:04:31 2009
        rootbp = …
...
```

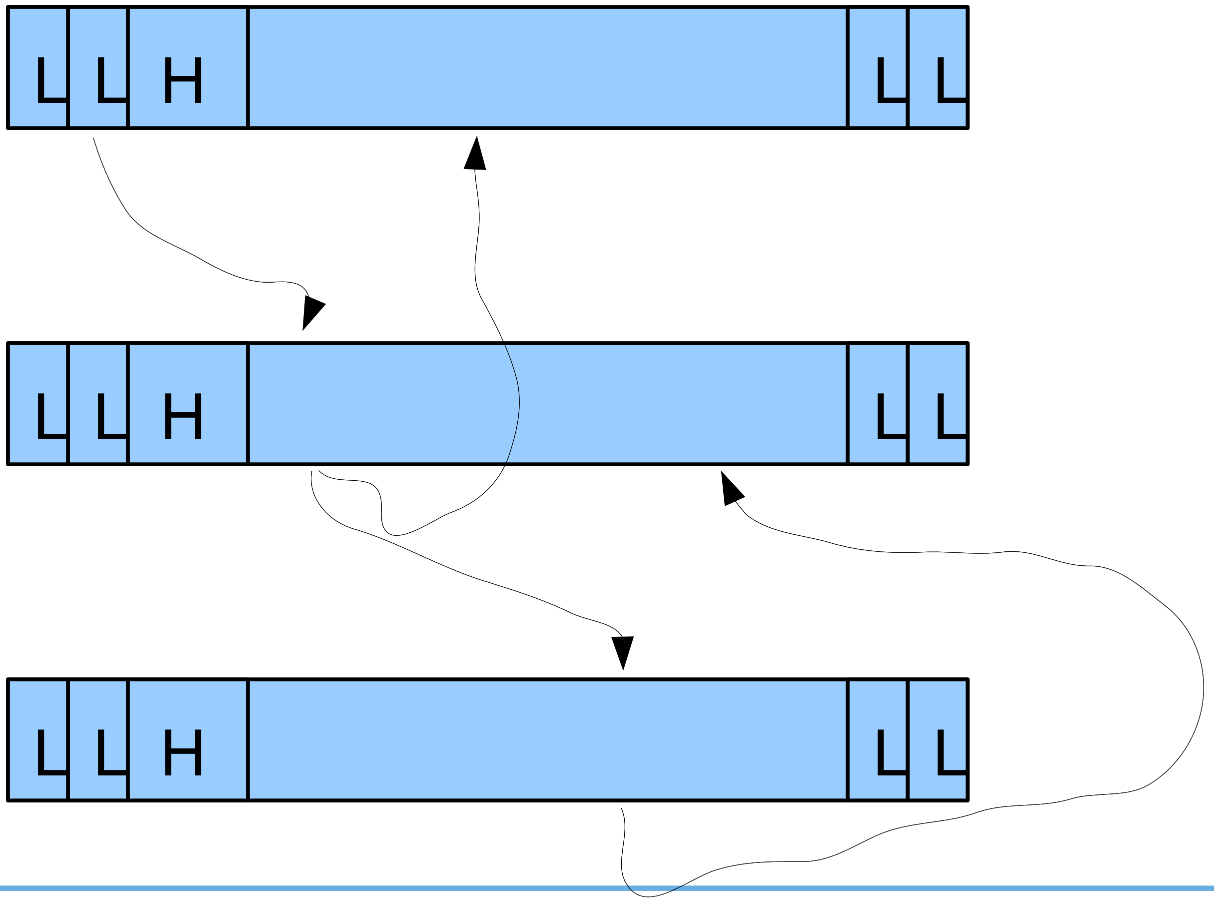
ZFS: *block pointer*

- *Data virtual address* (1, 2 or 3 dva)
  - Points to other block
  - References a *vdev* number defined in configuration
  - Contains number of block in *vdev*
  - Grid information (for raidz)
  - Gang bit (“gang chaining” of smaller blocks)
- Type and size of block (logical, allocated)
- Compression information (type, size)
- Transaction group numer
- Checksum of block (dva points to this block)

ZFS: *block pointer*: Example

```
rootbp = [L0 DMU objset]
    400L/200P
    DVA[0]=<0:5c8087800:200>
    DVA[1]=<0:4c81a2a00:200>
    DVA[2]=<0:3d002ca00:200>
    fletcher4 lzjb LE
    Contiguous  birth=262711
    Fill=324
    cksum=914be711d:3ab1cae4571
    :c07d93434c9b:1ab1618a08eccd
```

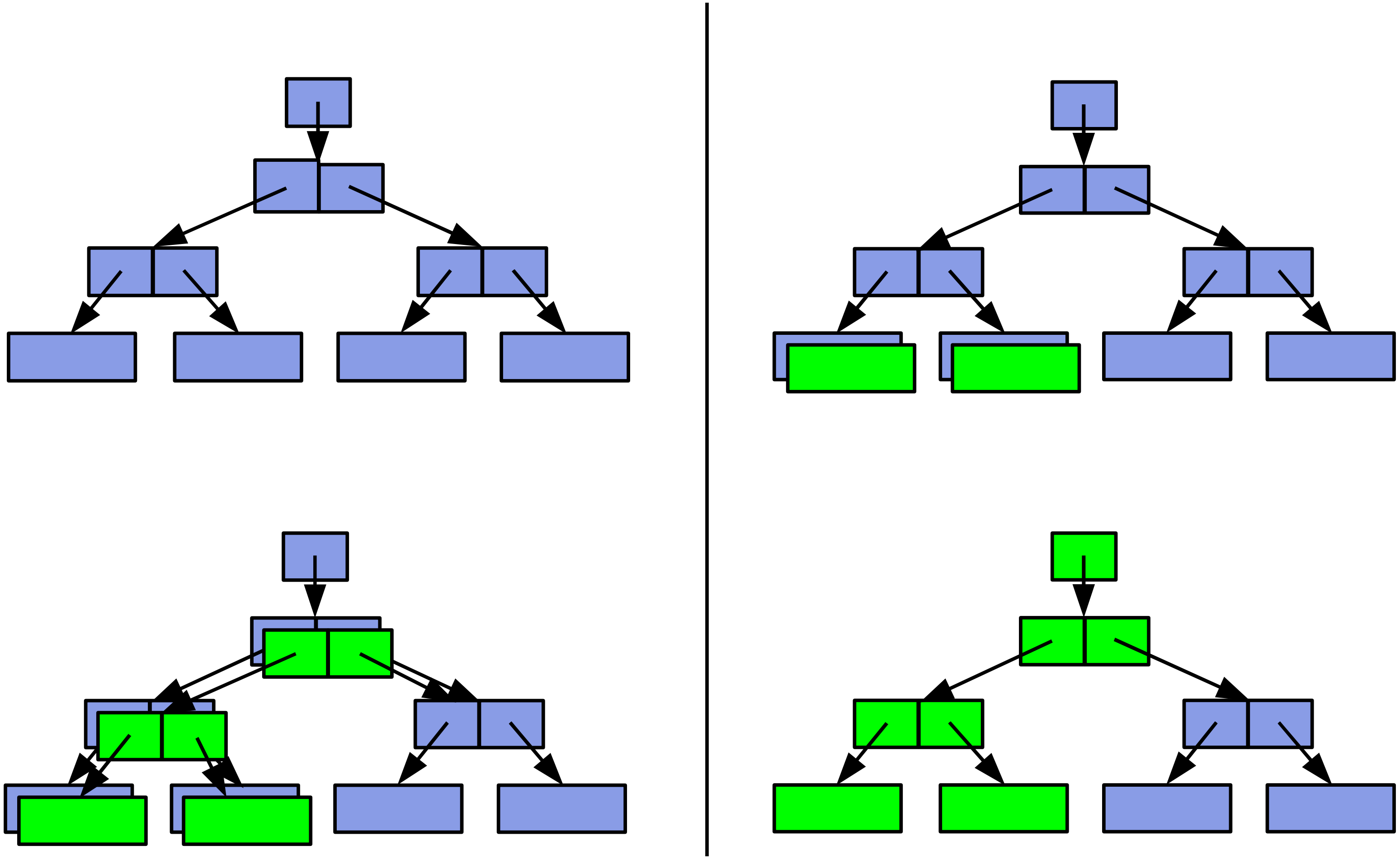
ZFS: some *block pointers* in a zpool



ZFS: Transactions

1. Starting at a consistent structure
2. Blocks may be changed by programs
   - Only prepared in main memory
   - Blocks are never overwritten on disk
3. Transaction is prepared
   - Structure is completed up to the root block
   - Blocks are written to *vdevs*
   - Only free blocks are used
4. Transaction is committed
   - The next uberblock slot is written

ZFS: Transaction



ZFS DMU Objects

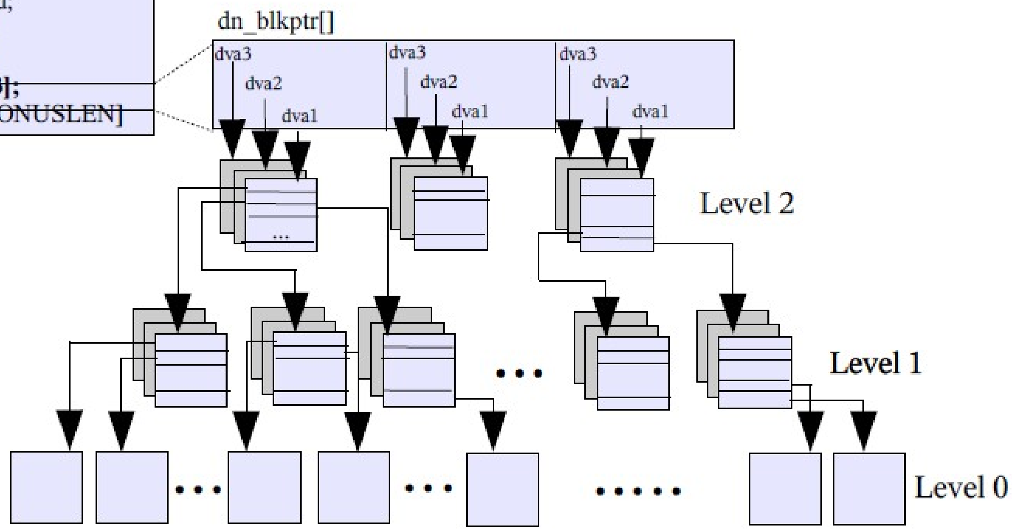
All data in a zpool is structured in objects

- *dnode* defines an object
  - Type and size, indirection depth
  - List of *block pointers*
  - Bonus buffer (f.e. for standard file attributes)
- DMU object set
  - Object that contains an array of *dnodes*
  - Uberblock: points to the *Meta Object Set*

ZFS: Object Structure

dnode_phys_t

```
uint8_t    dn_type;
uint8_t    dn_indblkshift;
uint8_t    dn_nlevels = 3
uint8_t    dn_nblkptr = 3
uint8_t    dn_bonustype;
uint8_t    dn_checksum;
uint8_t    dn_compress;
uint8_t    dn_pad[1];
uint16_t   dn_datablkszsec;
uint16_t   dn_bonuslen;
uint8_t    dn_pad2[4];
uint64_t   dn_maxblkid;
uint64_t   dn_secphys;
uint64_t   dn_pad3[4];
blkptr_t   dn_blkptr[3];
uint8_t    dn_bonus[BONUSLEN]
```



ZFS: Intent Log

- Stores all synchronously written data
- Uses unallocated blocks
- Is rooted in the *Object Set*

ZFS: Dataset and Snapshot Layer

DSL – Dataset and Snapshot Layer

- Filesystems
- Snapshots, clones
- ZFS volumes

Meta Object Set contains Object Set and

- Number of DSL directory (ZAP object)
- Copy of the vdev configuration
- Blockpointers to be freed

ZFS: DSL Structure

ZFS hierarchical names

- Child Dataset Entries in the DSL Directory
- Each Child has own DSL Directory

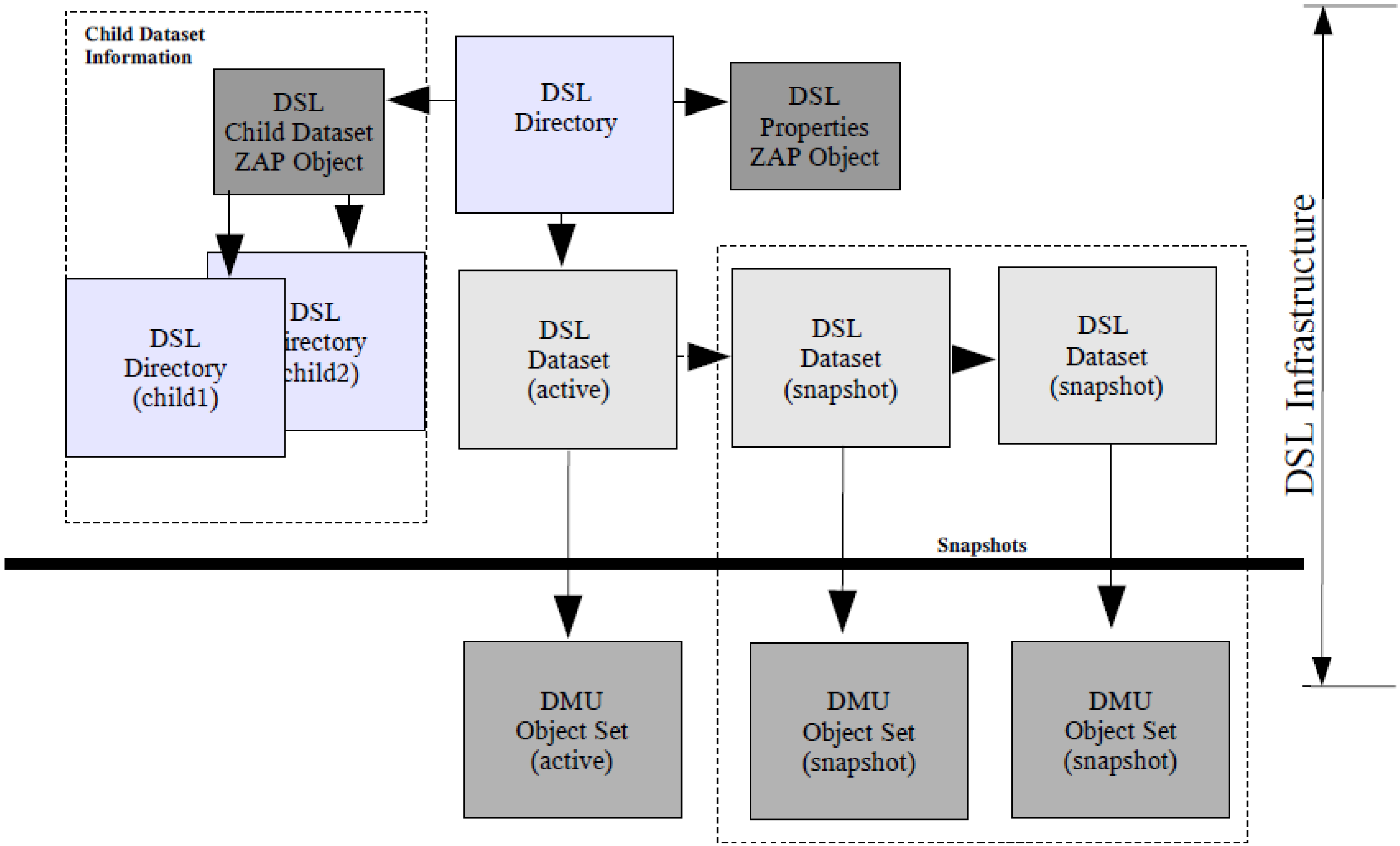
DSL Dataset

- Implemented by a DMU dnode

Snapshots and Clones

- Linked List rooted at the DSL Dataset

ZFS: DSL Structure



ZFS Attribute Processor

ZAP – ZFS Attribute Processor

- Name / value pairs
- Hash table with overflow lists
- Used for
  - Directories
  - ZFS hierarchical names
  - ZFS attributes

ZFS microZAP / FatZAP

microZAP

- One block (up to 128k)
- Simple Attributes (64 bit number)
- Name length limited (50 bytes)

FatZAP

- Object
- Hash into Pointer Table
- Pointers go to Name/Value storage

ZFS Posix Layer / Volume

ZFS Posix Layer

- Implements a Posix filesystem with objects
- Directories are ZAP objects
- Files are DMU objects
- Additional: Delete Queue

ZFS Volume

- Only one object in DSL Object set the Volume

ZFS: Misc

- Data is compressed when specified
- Metadata is compressed by default
  - All internal nodes
  - ZAP
  - DSL Directories, DSL Datasets

- Copies are implemented with DVA in BP
  - Zpool data is stored in 3 copies
  - ZFS data is stored in 2 copies
  - Data can be stored in up to 3 copies

ZFS Internal Structure

Questions?