Advanced File Systems, ZFS

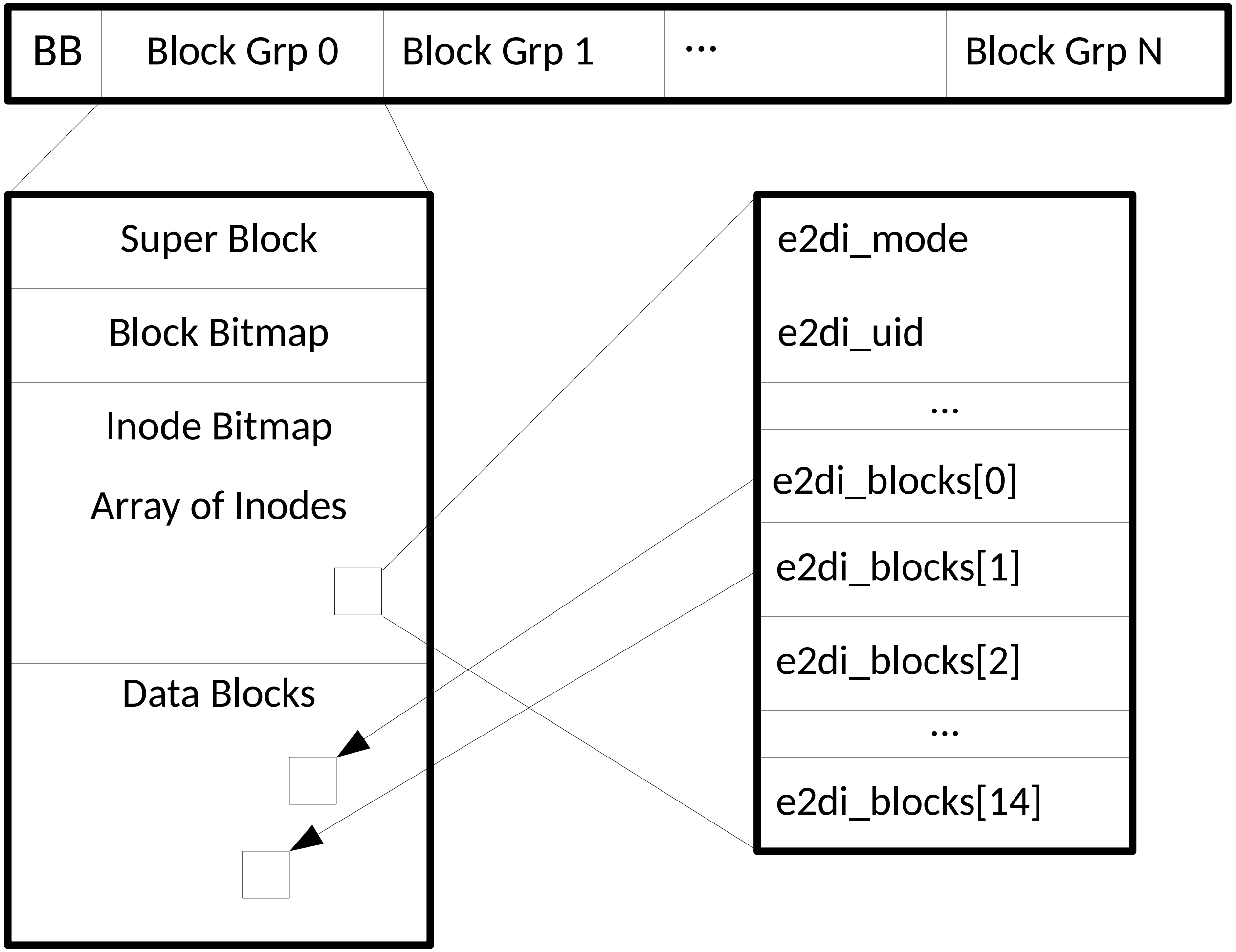
http://d3s.mff.cuni.cz/aosy

CHARLES UNIVERSITY
Faculty of Mathematics and Physics

Department of Distributed and Dependable Systems

Jan Šenolt
Jan.Senolt@Oracle.COM

Traditional UNIX File System, ext2



Crash consistency problem

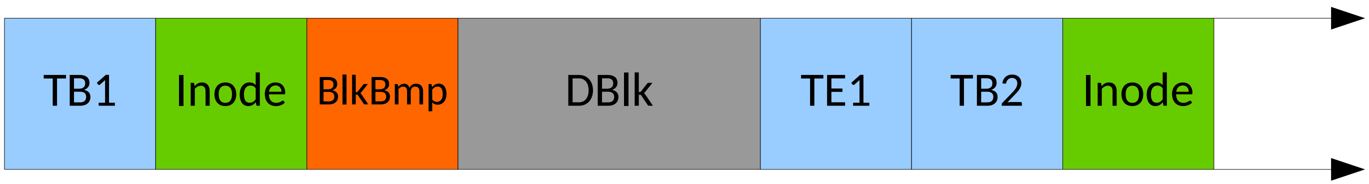
- Appending a new block to the file involves at least 3 IOs to different data structures at different locations:
  - Block bitmap – mark block as allocated
  - Inode – update e2di_blocks[], e2di_size, ...
  - Block – write the actual payload
- Cannot be performed atomically – what will happen if we fail to make some of these changes persistent?

fsck

- Lazy approach: try to detect the inconsistency and fix it
  - Does not scale well
  - Can take very long time for large file system
- Checks metadata only, unable to detect some types of inconsistencies
  - For example: updated the bitmap and the inode but crashed before writing the data block content
- ... but we still need fsck to detect other issues

Journaling

- Write all changes to the journal first, make sure that all writes completed and then made the actual in-place updates
  - Can be a file within fs or a dedicated disk



- Journal replay – traverse the log, find all complete records and apply them
- Physical journaling
  - Writes actual new content of blocks
  - Requires more space but is simple to replay
- Logical journaling
  - Description of what needs to be done
  - Must be idempotent

Journaling (2)

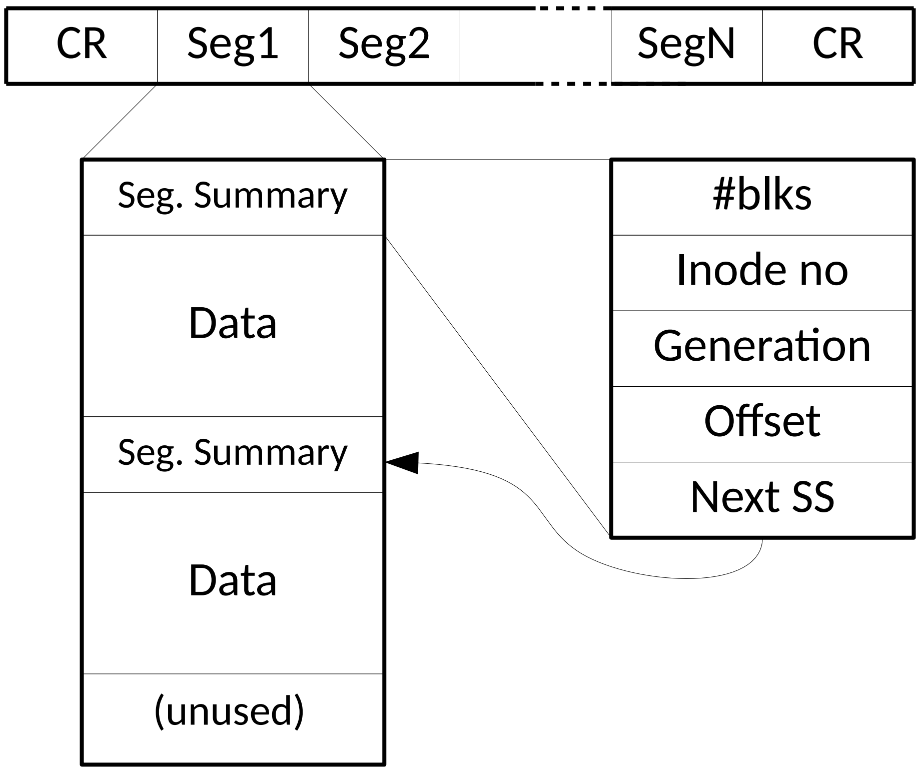
- Journal aggregation
  - Do multiple updates in memory, log them together in one transaction
  - Efficient when updating the same data multiple times
- (Ordered) metadata-only journal
  - Log only metadata → smaller write overhead
  - Write data block first, then create transaction for metadata
    - Metadata block reuse issue

Department of Distributed and Dependable Systems

Log structured FS

- Copy-on-Write
  - Fast crash recovery
- Long sequential I/O instead of many small I/Os
  - Better I/O bandwidth utilization
- Aggressive caching
  - Most I/Os are actually writes
- No block/inode bitmaps
- But disk has a finite size
  - Needs garbage collector

Log structured FS (2)



- Inode Map
  - inode# to block mapping
  - stored with other data but pointed from Checkpoint Regions
- UID
  - <inode# : gen>

Log structured FS (3)

- Segment cleaner (garbage collector)
  - Creates empty segments by compacting fragmented ones:
    1) Read whole segment(s) into memory
    2) Determine live data and copy them to another segment(s)
    3) Mark original segment as empty
  - Live data = still pointed by its inode
    - Increment inode version number when file deleted

Department of Distributed and Dependable Systems

Soft Updates

- Enforce rules for data updates:
  - Never point to an initialized structure
  - Never reuse block which is still referenced
  - Never remove existing reference until the new one exits
- Keep block in memory, maintain their dependencies and write them asynchronously
- Cyclic dependencies
  - Create a file in a directory
  - Remove a different file in the same dir (both files' inodes are in the same block)

Soft Updates (2) – pro and con

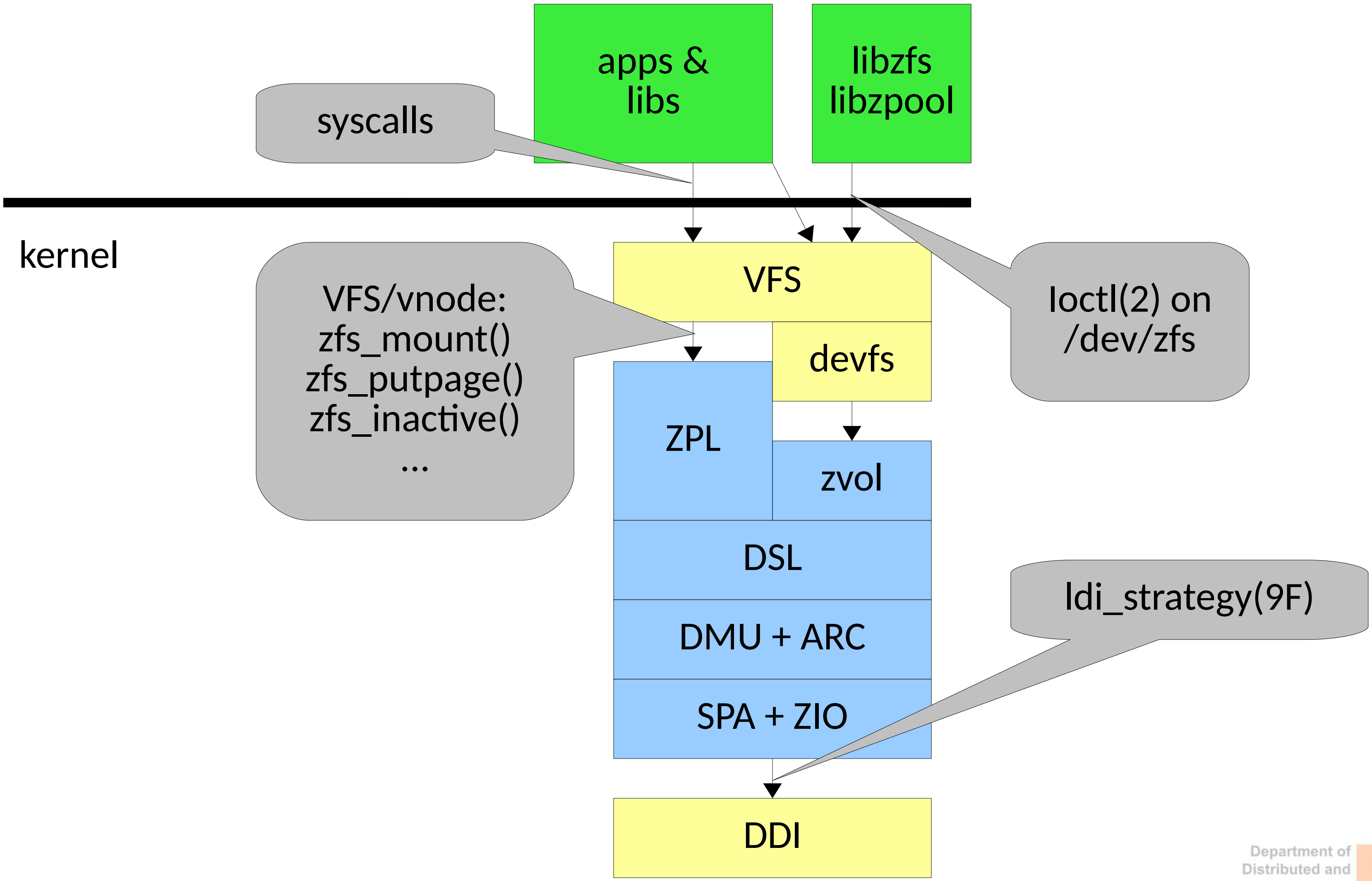
- Can mount the FS immediately after crash
  - The worst case scenario is a resource leak
  - Run fsck later or on background
    - Need snapshot
- Hard to implement properly
  - Delayed unref breaks POSIX
- fsync(2) and umount(2) slowness

Department of Distributed and Dependable Systems

ZFS vs traditional file systems

- **New administrative model**
  - 2 commands: `zpool(1M)` and `zfs(1M)`
  - Pooled storage
    - Eliminates the notion of volume and slices (partitions)
  - dynamic inode allocation
- **Data protection**
  - Transactional object system
    - always consistent on disk, no `fsck(1M)`
  - Detects and corrects data corruption
  - Integrated RAID
    - stripes, mirror, RAID-Z
- **Advanced features**
  - snapshots, writable snapshots, transparent compression & encryption, replication, integrated NFS & CIFS sharing, deduplication, ...

ZFS in Solaris



Pooled Storage Layer, SPA

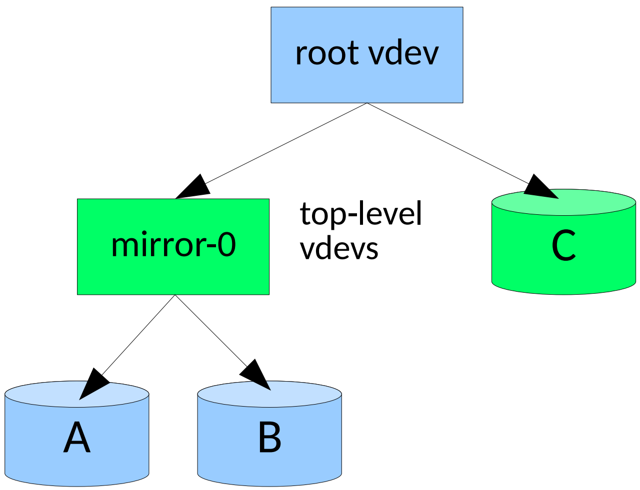
- **ZFS pool**
  - Collection of blocks allocated within a vdev hierarchy
    - top-level vdevs
    - physical x logical vdevs
    - leaf vdevs
    - special vdevs: l2arc, log, meta
  - Blocks addressed via “block pointers” - blkptr_t
- **ZIO**
  - Pipelined parallel IO subsystem
  - Performs aggregation, compression, checksumming, …
  - Calculates and verifies checksums
    - self-healing

Pooled Storage Layer, SPA (2)

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

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            /A      ONLINE       0     0     0
            /B      ONLINE       0     0     0
          /C        ONLINE       0     0     0

errors: No known data errors
```



Pooled Storage Layer, blkptr_t

- DVA – disk virtual address
  - VDEV – top-level vdev number
  - ASIZE – allocated size
- LSIZE
  - logical size – without compression, RAID-Z or gang overhead
- PSIZE
  - compressed size
- LVL – block level
  - 0 – data block
  - >0 – indirect block
- BDE
  - little vs big-endian
  - dedup
  - Encryption
- FILL - number of blkptrs in block

	64	56	48	40	32	24	16	8
0	VDEV 1				ncpy|L4T	ASIZE		
1	G| OFFSET 1							
2	VDEV 2				ncpy|L4T	ASIZE		
3	G| OFFSET 2							
4	VDEV 3				ncpy|L4T	ASIZE		
5	G| OFFSET 3							
6	BDE| LVL	TYPE	CKSUM	COMP	PSIZE		LSIZE	
7	PADDING							
8	PADDING							
9	PHYSICAL BIRTH TXG							
A	BIRTH TXG							
B	FILL COUNT							
C	CHECKSUM[0]							
D	CHECKSUM[1]							
E	CHECKSUM[2]							
F	CHECKSUM[3]							

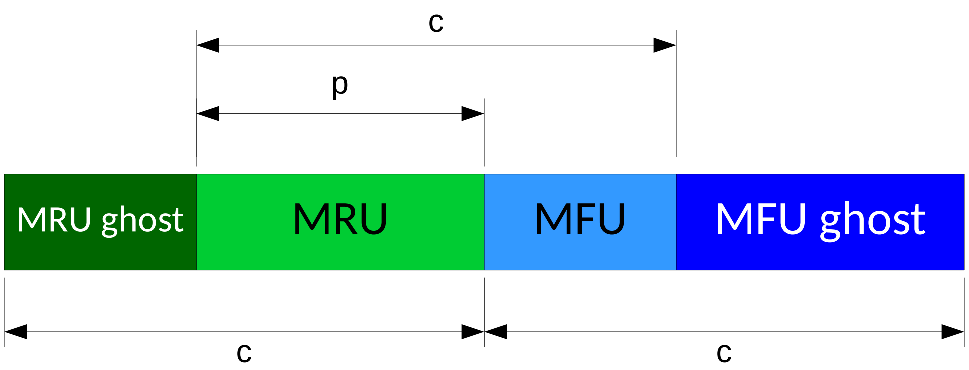
Department of Distributed and Dependable Systems

Data Management Unit, DMU

- dbuf (dmu_buf_t)
  - in-core block, stored in ARC
  - size 512B – 1MB
- object (dnode_t, dnode_phys_t)
  - array of dbufs
  - types: DMU_OT_PLAIN_FILE_CONTENTS, DMU_OT_DNODE, …
  - dn_dbufs – list of dbufs
  - dn_dirty_records – list of modified dbufs
- objset (objset_t, objset_phys_t)
  - set of objects
  - os_dirty_dnodes – list of modified dnodes

Adaptive Replacement Cache, ARC

- Megiddo, Modha: ARC: A Self-Tuning, Low Overhead Replacement Cache [1]



- p – increase if found in MRU-Ghost, decrease if found in MFU-Ghost
- p – increase to fill unused memory, arc_shrink()
- Evict list –list of unreferenced dbufs
  - can be moved to L2ARC: l2arc_feed_thread()
- Hash table
  - hash(SPA, DVA, TXG)
  - arc_hash_find(), arc_hash_insert()

Dataset and Snapshot Layer, DSL

- dsl_dir_t, dsl_dataset_t
- Adds names to objsets, creates parent – child relation
  - implements snapshots and clones
- Maintains properties
- DSL dead list
  - set of blkptrs which were referenced in the previous snapshot, but not in this dataset
  - when a block is no longer referenced:
    - free it if was born after most recent snapshot
    - otherwise put it on datasets dead list
- DSL scan
  - traverse the pool, corrupted data triggers self-healing
  - scrub – scan all txgs vs resilver – scan only txg when vdev was missing
- ZFS stream
  - serialized dataset(s)

ZFS Posix Layer, ZPL & ZFS Volume

- ZPL
  - creates a POSIX-like file system within dataset
  - znode_t, zfsvfs_t
  - System Attributes
    - portion of znode with variable layouts to accommodate type specific attributes
- ZVOL
  - block devices in /dev/zvol
  - SCSI targets (via COMSTAR)
    - direct access to DMU & ARC, RDMA

Write to file (1)

```
zfs_putapage(vnode, page, off, len, ...):
    dmu_tx_t *tx = dmu_tx_create(vnode→zfsvfs→z_os);
    dmu_tx_hold_write(tx, vnode->zp->z_id, off, len);
    err = dmu_tx_assign(tx, TXG_NOWAIT);
    if (err)
        dmu_tx_abort(tx);
        return;
    dmu_buf_hold_array(z_os, z_id, off, len, ..., &dbp);
    bcopy(page, dbp[]->db_db_data);
    dmu_buf_rele_array(dbp,...)
    dmu_tx_commit(tx);
```

dmu_buf_t **dbp

Write to file (2), dmu_tx_hold*

- What and how we are going to modify?

```
dmu_tx {
    list_t tx_holds;
    objset_t
    *tx_objset;
    int tx_txg;
    ...
}
```

```
dmu_tx_hold {
    dnode_t txh_dnode;
    int txh_space_towrite;
    int txh_space_tofree;
    ...
}
```

- dmu_tx_hold_free()
- dmu_tx_hold_bonus()

Department of Distributed and Dependable Systems

Write to file (3), dmu_tx_assign()

- Assign tx to an open TXG

```
dmu_tx_try_assign(tx):
    for txh in tx->tx_holds:
        towrite += txh->txh_space_towrite;
        tofree += txh->txh_space_tofree;
```

we throttle writes in order to write all changes in 5 seconds

```
dsl_pool_tempreserve_space():
    if (towrite + used > quota)
        return (ENOSPC);
    if (towrite > arc->avail)
        return (ENOMEM);
    if (towrite > write_limit)
        return (ERESTART);
    ...
```

Write to file (6), Txg Life Cycle

- Each txg goes through 3-stage DMU pipeline:
  - Open
    - accepts new dmu_tx_assign()
  - Quiescing
    - waiting for every tx to call dmu_tx_commit()
    - txg_quiesce_thread()
  - Syncing
    - writing changes to disks
    - txg_sync_thread()

Write to file (5), dmu_buf_hold_array()

- Prepare array of dbufs in ARC
  - dbuf exists
    - dbuf is active → allocate anonymous copy
    - dbuf is not active → anonymize dbuf
  - dbuf does not exist → allocate anonymous copy
- Anonymous dbuf does not know its DVA
- Link dbuf on dnode's list of dirty dbufs for this txg
  - dn_dirty_records

Department of Distributed and Dependable Systems

Write to file (6), sync

- Sync thread traverse dirty records and sync changes to disks

```
spa_sync():
dsl_pool_sync()
    dsl_dataset_sync()
        dmu_objset_sync()
            dmu_objset_sync_dnodes()
                dnode_sync() - also changes block size, ind. level etc
                    dbuf_sync_list()
                        dbuf_sync_indirect()
                            dbuf_sync_leaf()
                                dbuf_write()
                                    zio_write() - sends dbuf to ZIO
```

- Iterate to convergence
  - usually < ~5 passes

Department of Distributed and Dependable Systems

Write to file (6), ZIO

- Depending on IO type, dbuf properties etc ZIO goes through different stages of ZIO pipeline:
  - ZIO_STAGE_WRITE_BP_INIT – data compression
  - ZIO_STAGE_ISSUE_ASYNC – moves ZIO processing to taskq(9F)
  - ZIO_STAGE_CHECKSUM_GENERATE – checksum calculation
  - ZIO_STAGE_DVA_ALLOCATE – block allocation
  - ZIO_STAGE_READY – synchronization
  - ZIO_STAGE_VDEV_IO_START – start the IO by calling vdev_op_io_start method
  - ZIO_STAGE_VDEV_IO_DONE
  - ZIO_STAGE_VDEV_IO_ASSESS – handle eventual IO error
  - ZIO_STAGE_DONE – undo aggregation

Free space tracking methods

- bitmaps (UFS, extN)
  - each allocation unit represented by a bit
    - WAFL uses 32bit per allocation unit (4K)
  - bitmap can be huge, it needs to be initialized
  - slow to search for empty block
- B+ tree (XFS, JFS)
  - tree of extents
  - each extent usually tracked twice: by offset and by size

Free Space Tracking in ZFS (1)

- Each top-level vdev is split into 200 metaslabs
  - don’t need to keep inactive metaslab’s info in RAM
- Each metaslab has associated space map
  - AVL tree of extents *in core*
    - by offset – easy to coalesce extents
    - by size – for searching by extent size
  - time ordered log of allocations and frees *on disk*
    - only append new entries
    - destroy and recreate from the tree when log is too big

Department of Distributed and Dependable Systems

Free Space Tracking in ZFS (2)

- Top-level vdev selection
  - biased round robin, change every 512KB * #children
- Choose metaslab with highest weight
  - low LBA, metaslab already in core
  - when allocating ditto copy, select metaslab which is 1/8 of vdev size away
- Choose extent
  - cursor – end of the last allocated extent
  - metaslab_ff_alloc
    - first sufficient extent after cursor
  - metaslab_df_alloc
    - FF for metaslabs up to 70% free, best-fit then

Q&A

References

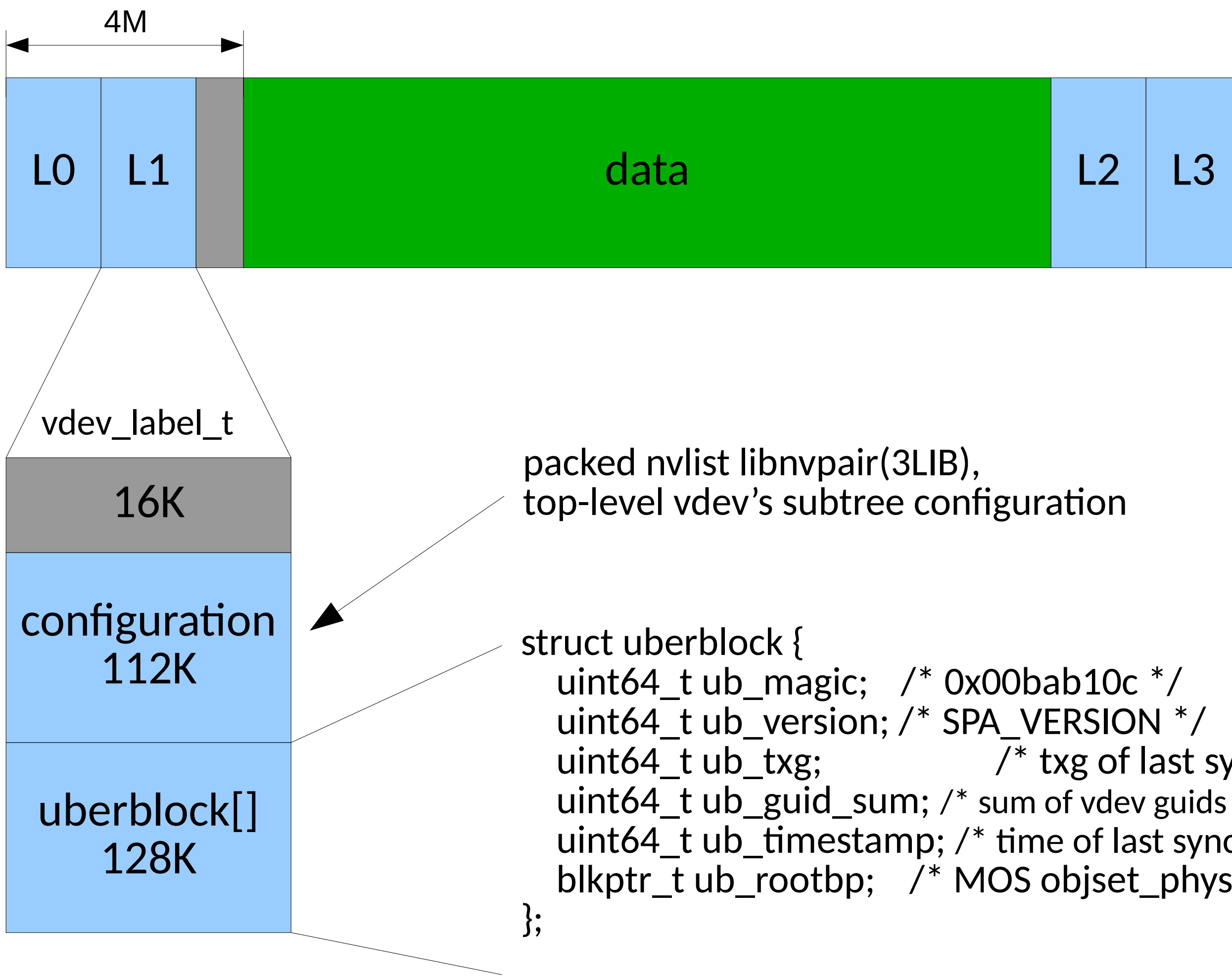
- McKusick M.: *Fsck – The UNIX File System Check Program*, Revised in 1996
- Tweedie S. C.: *Journaling the Linux ext2fs Filesystem*, The Fourth Annual Linux Expo, May 1998
- Rosenblum M., Ousterhout J.: *The Design and Implementation of a Log-Structured File System*, SOSP '91, Pacific Grove, CA, October 1991
- Ganger G., McKusick M.: Soft updates: a technique for eliminating most synchronous writes in the fast filesystem, ATEC '99 Proceedings of the annual conference on USENIX Annual Technical Conference, 1999
- Aurora V.: Soft updates, hard problems, LWM.net, 2009
- Megiddo N., Modha D.: *ARC: A Self-Tuning, Low Overhead Replacement Cache*, Proceedings of the 2003 Conference on File and Storage Technologies (FAST), 2003
- Sun Microsystem Inc: *ZFS On-Disk Specification*, Draft, 2006
- Bonwick J.: *ZFS – The last word in file systems*, 2008

Department of Distributed and Dependable Systems

Appendix

ZFS on-disk format

Pooled Storage Layer, Physical Vdev



packed nvlist libnvpair(3LIB),
top-level vdev's subtree configuration

```
struct uberblock {
    uint64_t ub_magic;    /* 0x00bab10c */
    uint64_t ub_version; /* SPA_VERSION */
    uint64_t ub_txg;               /* txg of last sync */
    uint64_t ub_guid_sum; /* sum of vdev guids */
    uint64_t ub_timestamp; /* time of last sync */
    blkptr_t ub_rootbp;    /* MOS objset_phys_t */
};
```

Department of Distributed and Dependable Systems

Pooled Storage Layer, Label

```
# zdb -luuu /dev/dsk/c1t1d0s0
LABEL 0:
    timestamp: 1489412157 UTC: Mon Mar 13 13:35:57 2017
    version: 43
    name: 'tank'
    state: 0
    txg: 4
    pool_guid: 15329707826800509494
    hostid: 613234
    hostname: 'va64-x4100e-prg06'
    top_guid: 6425423019115642578
    guid: 6425423019115642578
    vdev_children: 2
    vdev_tree:
        type: 'disk'
        id: 0
        guid: 6425423019115642578
        path: '/dev/dsk/c1t1d0s0'
        devid: 'id1,sd@SSEAGATE_ST973401LSUN72G_0411EZXT____________3LB1EZXT/a'
        phys_path: '/pci@0,0/pci1022,7450@2/pci1000,3060@3/sd@1,0:a'
        whole_disk: 1
        metaslab_array: 29
        metaslab_shift: 29
        ashift: 9
        asize: 73394552832
        is_log: 0
        is_meta: 0
        create_txg: 4
```

Department of Distributed and Dependable Systems

Pooled Storage Layer, Uberblock

```
Uberblock[0]
	magic = 0x0000000000bab10c
	version = 43
	txg = 132
	guid_sum = 5921737069822600244
	pool_guid = 15329707826800509494
	hostid = 0x95b72
	timestamp = 1489412593 date = Mon Mar 13 14:43:13 CET 2017
	rootbp = DVA[0]=<1:58001f400:800:STD:1> DVA[1]=<0:540cbc600:800:STD:1>
DVA[2]=<1:80002ac00:800:STD:1> [L0 DMU objset] fletcher4 uncompressed LE contiguous
unique unencrypted 3-copy size=800L/800P birth=132L/132P fill=7c
cksum=2e47b25da:5540247ccc2:4eb3db21abd63:308d529e5d9b7f9
```

Pooled Storage Layer, On-disk

objset_phys_t

```
dnode_phys_t os_meta_dnode;
zil_header_t os_zil_header;
uint64_t os_type = DMU_OST_META
```

dnode_phys_t

```
uint8_t dn_type = DMU_OT_DNODE
uint8_t dn_indblkshift;
uint8_t dn_nlevels;
uint8_t dn_nblkptr;
...
blkptr_t dn_blkptr[];
```

dnode_phys_t []

obj 0: DMU dnode
obj 1: Object dir
obj 2: Master obj
obj 3: ...

dnode_phys_t

```
dn_type = DMU_OT_OBJECT_DIRECTORY
dn_indblkshift = 14;
dn_nlevels = 1;
dn_nblkptr = 1;
...
blkptr_t dn_blkptr[0];
```

mzap_phys_t

```
uint64_t mz_block_type;
uint64_t mz_salt;
uint64_t mz_normflags;
uint64_t mz_pad[5];
mzap_ent_phys mz_chunk[]
```

mzap_ent_phys_t

```
uint64_t mze_value = 2
uint32_t mze_cd = ...
uint32_t mze_pad;
char mze_name[] = 'root_dataset'
```

64B