

ZFS

Yet another Filesystem?

Thorsten Ries
thorsten.ries@tudor.lu
Centre de Recherche Public Henri Tudor

- ▶ The thing with the filesystems
- ▶ ZFS: What is it?
- ▶ Features
- ▶ Demo

- ▶ Some while ago...
 - ... each filesystem managed a single disk
- ▶ By the time, people wanted more bandwidth, reliability, etc.
- ▶ By the time, disks grew
- ▶ By the time, volume managers were developed
- ▶ By the time this wasn't (isn't) enough...

- ▶ 1984 - UFS
- ▶ 1985 - HFS
- ▶ 1987 - HPFS
- ▶ 1990 - JFS
- ▶ 1993 - ext2
- ▶ 1993 - NTFS
- ▶ 1994 - WAFL
- ▶ 1994 - XFS
- ▶ 1998 - ReiserFS
- ▶ 1998 - HFS plus
- ▶ 2000 - JFS2
- ▶ 2001 - ext3
- ▶ 2004 - ZFS
- ▶ 2004 - Reiser4
- ▶ 2006 - ext4

▶ Why a new filesystem?

- The value of Data is becoming even more critical
 - Silent data corruption happens
- High administrative expense
 - Remember the steps of creating a FS
 - Create a RAID x
 - Portability (e.g. SPARC to/from x86)
- Current FS performance
 - Fixed block sizes
 - slow random writes
- The amount of storage is tremendously increasing

▶ Objectives of ZFS

- Simple to manage
- Powerful
- Fast
- Safe

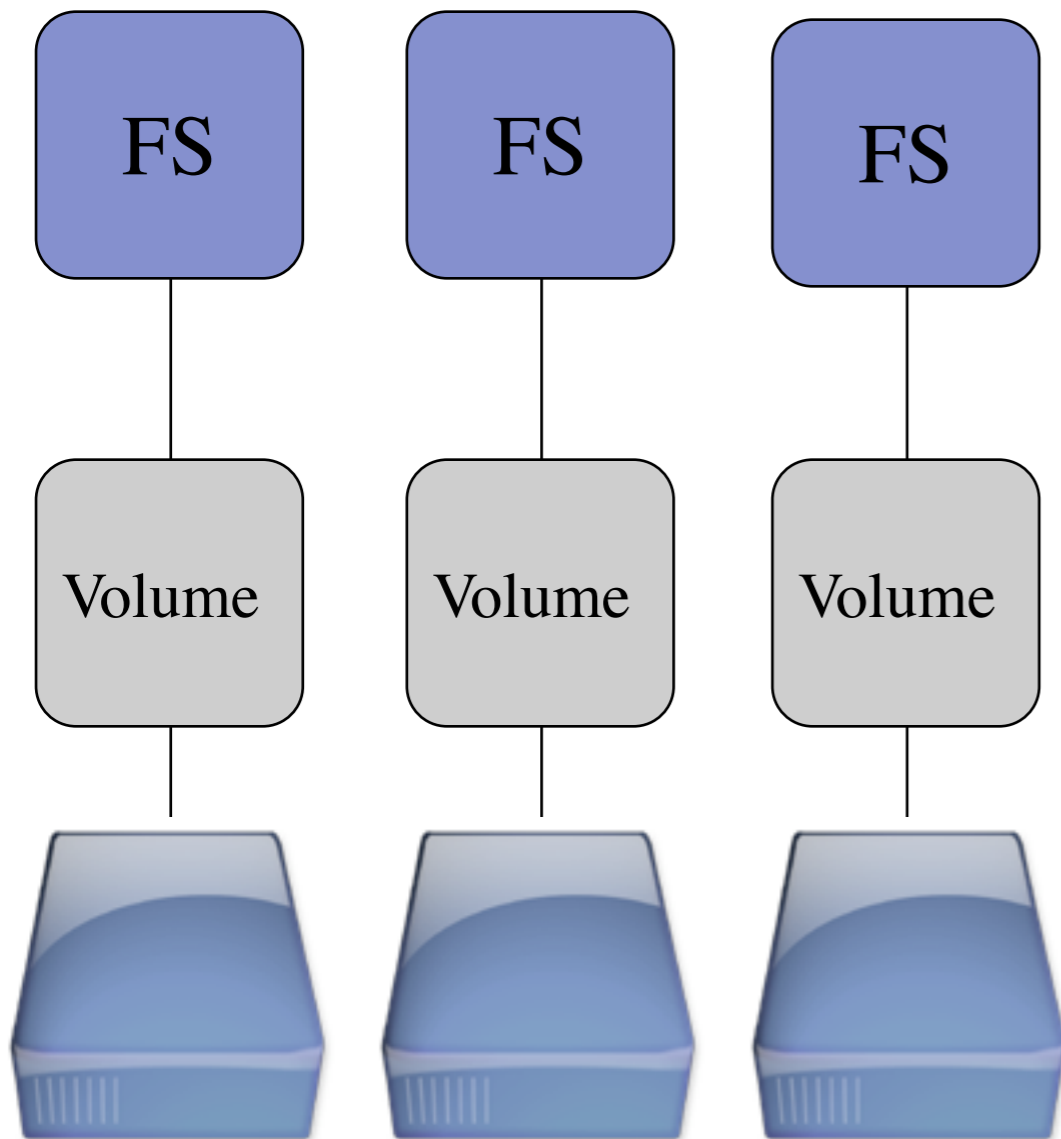
- ▶ 128 bit filesystem
 - Max filesystem size: 16 exabytes (2^{64})
 - Max filesize: 16 exabytes (2^{64})
 - Number of files in any FS: 2^{48}
 - Number of devices in any pool: 2^{48}
 - Number of zpools in a system: 2^{64}
 - Number of filesystems in a pool: 2^{64}
 - Number of files in a directory: 2^{56} (2^{48})

- ▶ Those are large numbers... ;-)
 - If 1000 files are created every second, it would take about 9000 years to reach the limit of files in a FS.
 - Project leader Jeff Bonwick said: “Populating 128-bit file systems would exceed the quantum limits of earth-based storage. You couldn’t fill a 128-bit storage pool without boiling the oceans.”[1]

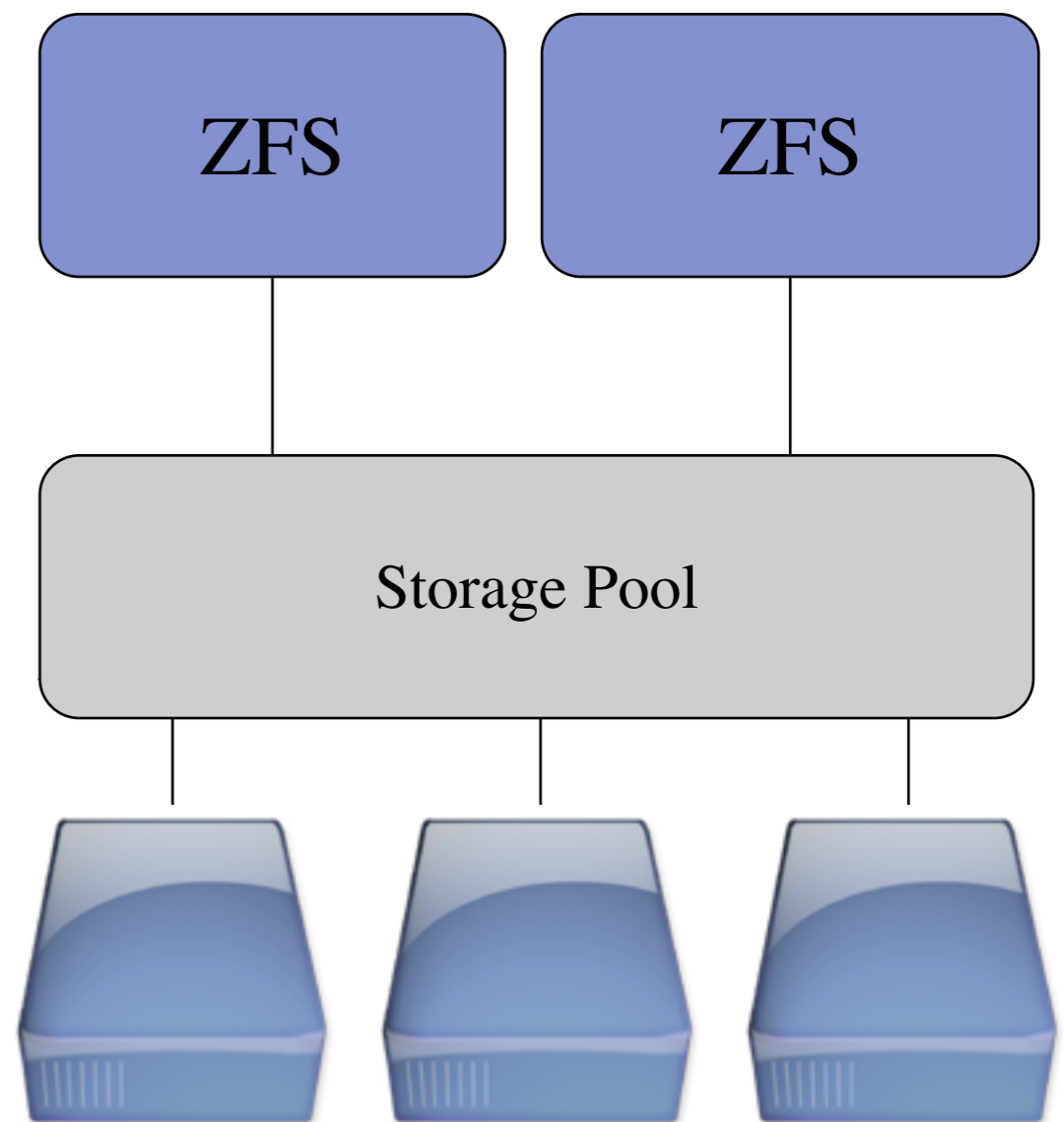
- ▶ Pooled storage
 - Elimination of antique volumes
 - Does for storage what VM did for memory
- ▶ End-to-end data integrity
 - Historically considered “too expensive”
 - ... but seems not...
- ▶ Transactional operation
 - Disk consistency
 - Even though performance wins
 - No fsck :-)

Volume vs. Pool

Volume



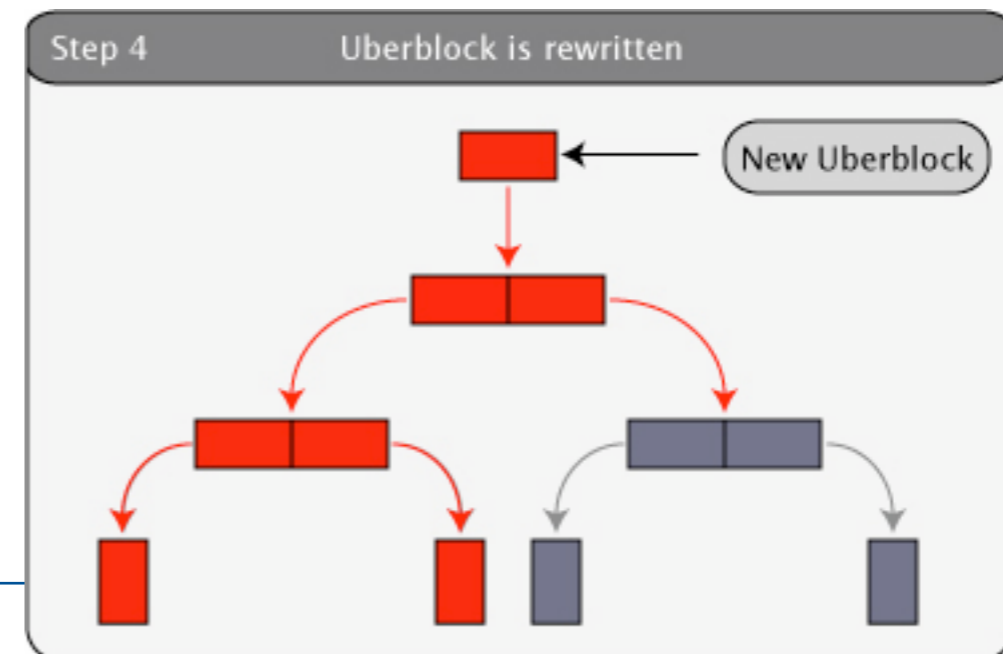
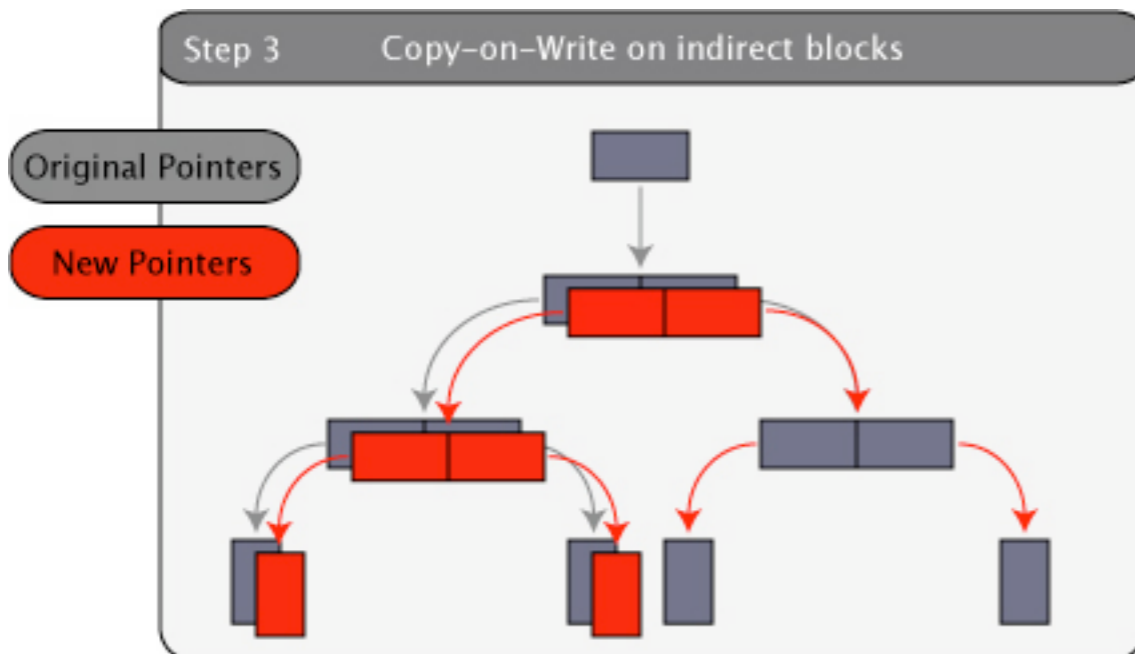
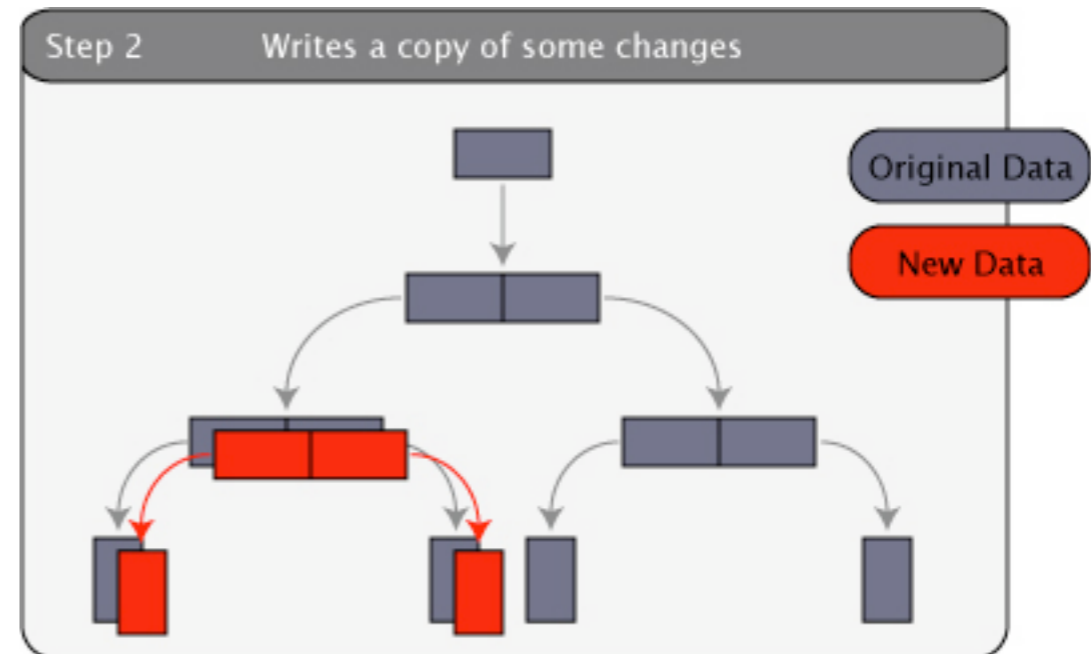
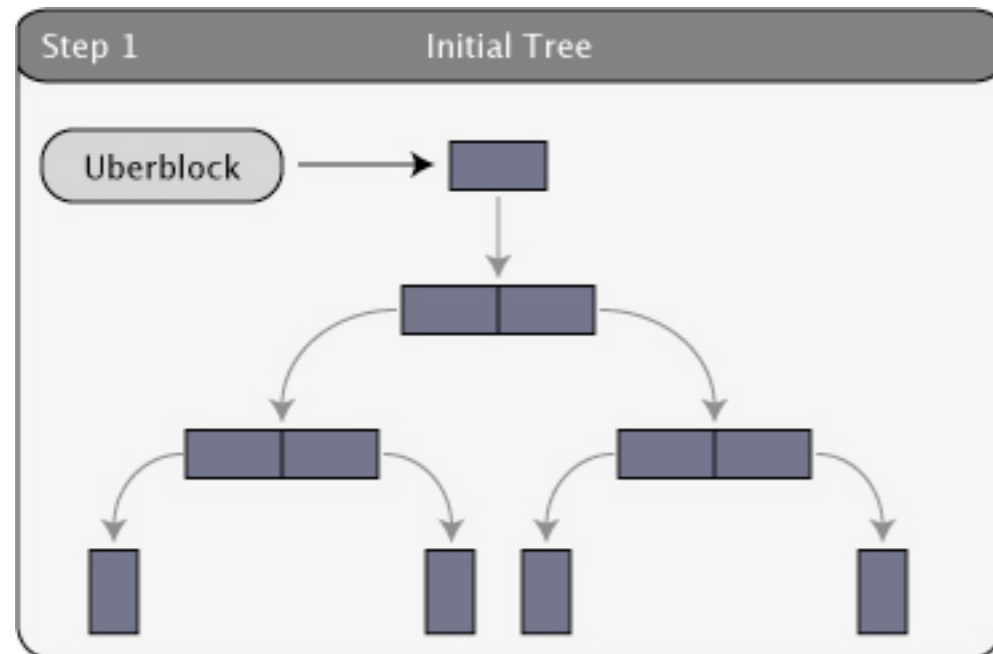
Pool



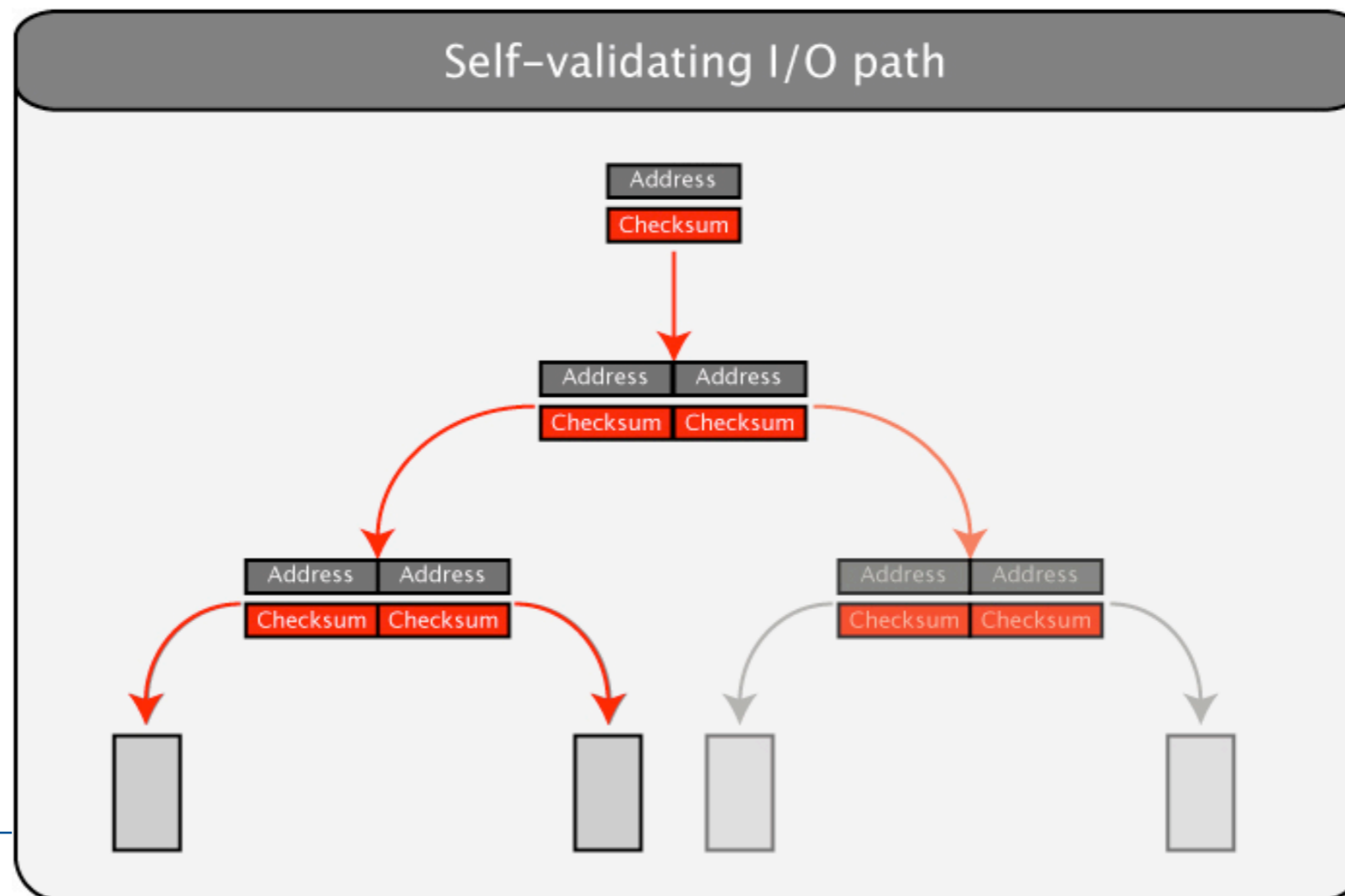
Volumes	Pool
Partion/Volume for each FS	No partitions to manage
LVM needed to have similar functionality	No extra LVM
Grow/shrink by hand	Grow/shrink by automatically
Less flexibility	Flexible pool management

- ▶ Copy-on-Write (COW), transactional design
- ▶ Checksums
- ▶ RaidZ protection
- ▶ Self-healing
- ▶ Disc scrubbing

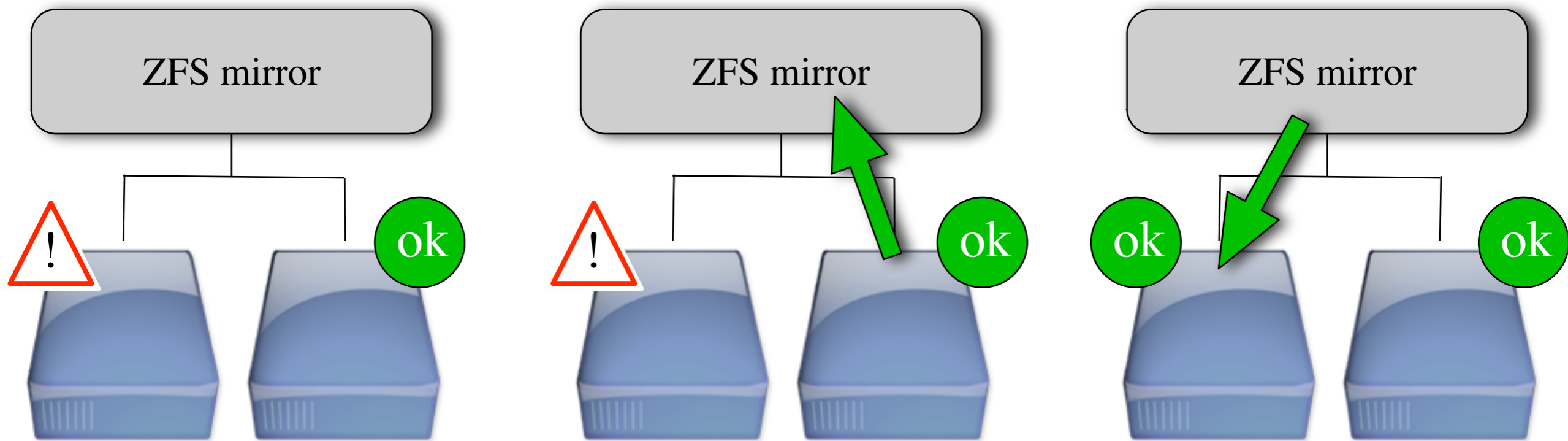
- ▶ Live data is never overwritten
- ▶ Write data to new block before changing data pointers and committing write



- ▶ All data and metadata is checksummed
- ▶ Traditional FS: checksums per block basis
 - Problem: Silent data corruption
- ▶ ZFS: checksums are stored in the parent node
 - fault isolation between data and checksum



- ▶ Bad checksums are detected and data is “healed” by the mirrored copy



- ▶ In storage systems, data can be lost due to failures of both the device level or the block level
- ▶ Technique to detect 'issues' before they are critical
- ▶ Redundancy is used to repair e.g. affected blocks

In ZFS:

- ▶ Checksums used to verify the integrity of all data
- ▶ Latent errors can be found that can be corrected
- ▶ Like ECC memory scrubbing - but for disks

- ▶ Similar to Raid5, BUT:
 - additional protection against “write-hole” problems without additional hardware
 - latent errors can be found and corrected
 - Variable-width Raid stripes
 - all writes are full stripe writes

 better than Raid5 availability

▶ Snapshot:

- Read-only copy of a filesystem or volume
- Creation instantly
- Initially no additional disk space used
- Only changes are tracked
- Rollback possibility

▶ Clone

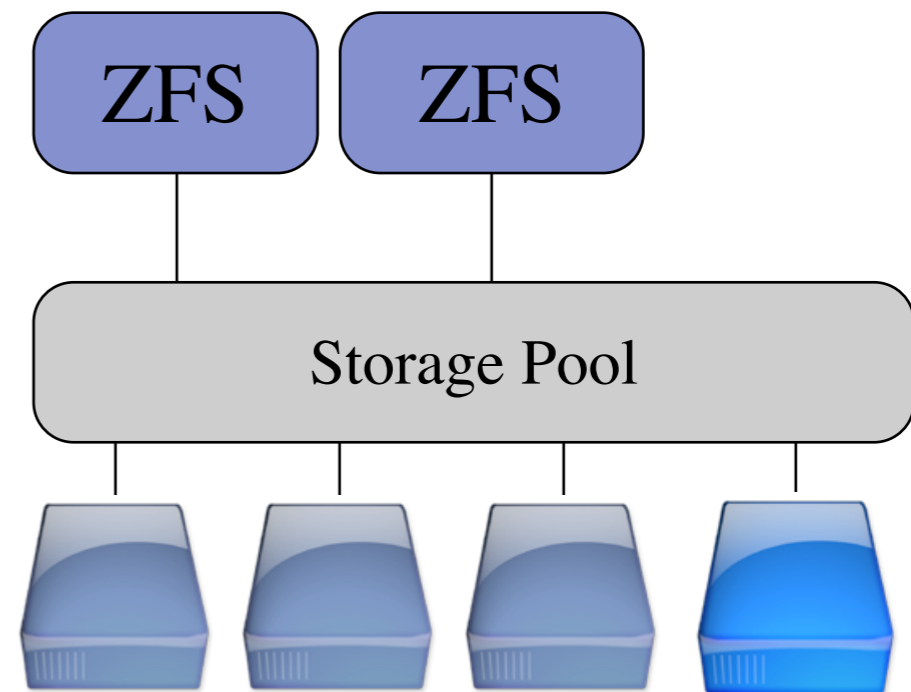
- Writable volume or filesystem
- Creation instantly
- Initially no additional disk space used
- Created from a snapshot
 - The original snapshot can not be deleted as long as the clone exist

▶ Dynamic striping

- Segmentation of sequential data (e.g. a single file) so that segments can be written to multiple physical devices
- Across all devices
- All disks in pool are used (copy-on-write used)
- Balance load across all devices

▶ Variable Block size

- up to 128kB
- Administrator can tune block size
- If compression is used, block size is adapted too
 - Advantage: improved I/O throughput



- ▶ Two possibilities:
 - Web interface (<https://localhost:6789/>)



The screenshot shows a web browser window titled "https://localhost:6789 - Erstellen von Speicherpools - Mozilla Firefox". The page content is "ZFS Administration" and "Erstellen von Speicherpools". It features a progress bar with five steps: 1. Benennen des Speicherpools, 2. Auswahl einzubindender Geräte, 3. Überprüfen der gewählten Konfiguration (highlighted with a blue arrow), 4. Vorschau der auszuführenden Befehle, and 5. Anzeige der Befehlsergebnisse. The main area displays the configuration for "Speicherpools pool1" as a "Spiegel (1,95 GB)" consisting of two "Festplatte" (c1d0 and c1d1), each 1,95 GB. Below this, it says "Einhängepunkt:". At the bottom, there are buttons for "Zurück", "Weiter" (highlighted with a dotted border), and "Abbrechen". The status bar at the bottom left says "Fertig" and the bottom right shows "localhost:6789".

▶ CLI

- Only two commands [zpool|zfs]
- Very quick FS creation
- Simple mount and NFS management

▶ The most basic FS

```
# zpool create pool c2t0d0
```

- This command creates a zpool with the name 'pool' on disk c2t0d0
- Creates a filesystem with the same name
- Mounts the filesystem (/pool)
- Automatically mounted after reboot

- ▶ Creating a mirrored pool

```
# zpool create pool mirror c2t2d0 c4t1d0
```

- ▶ Creating a 'sub-FS'

```
# zfs create pool/home
```

- ▶ Setting the mountpoint

```
# zfs set mountpoint=/export/home/ries pool/home
```

- ▶ built-in compression

```
# zfs set compression=on pool/home
```

- ▶ Set quota

```
# zfs set quota=2M pool/home/dummy
```

- ▶ Reserve 100GB

```
# zfs set reservation=100G pool/home/ries
```

- ▶ Replace a disk

```
# zpool replace pool c2t2d0 c4t1d0
```

- ▶ Add more space

```
# zpool add pool c4t0d0
```

- ▶ Scrub all disks and verify integrity

```
# zpool scrub pool
```

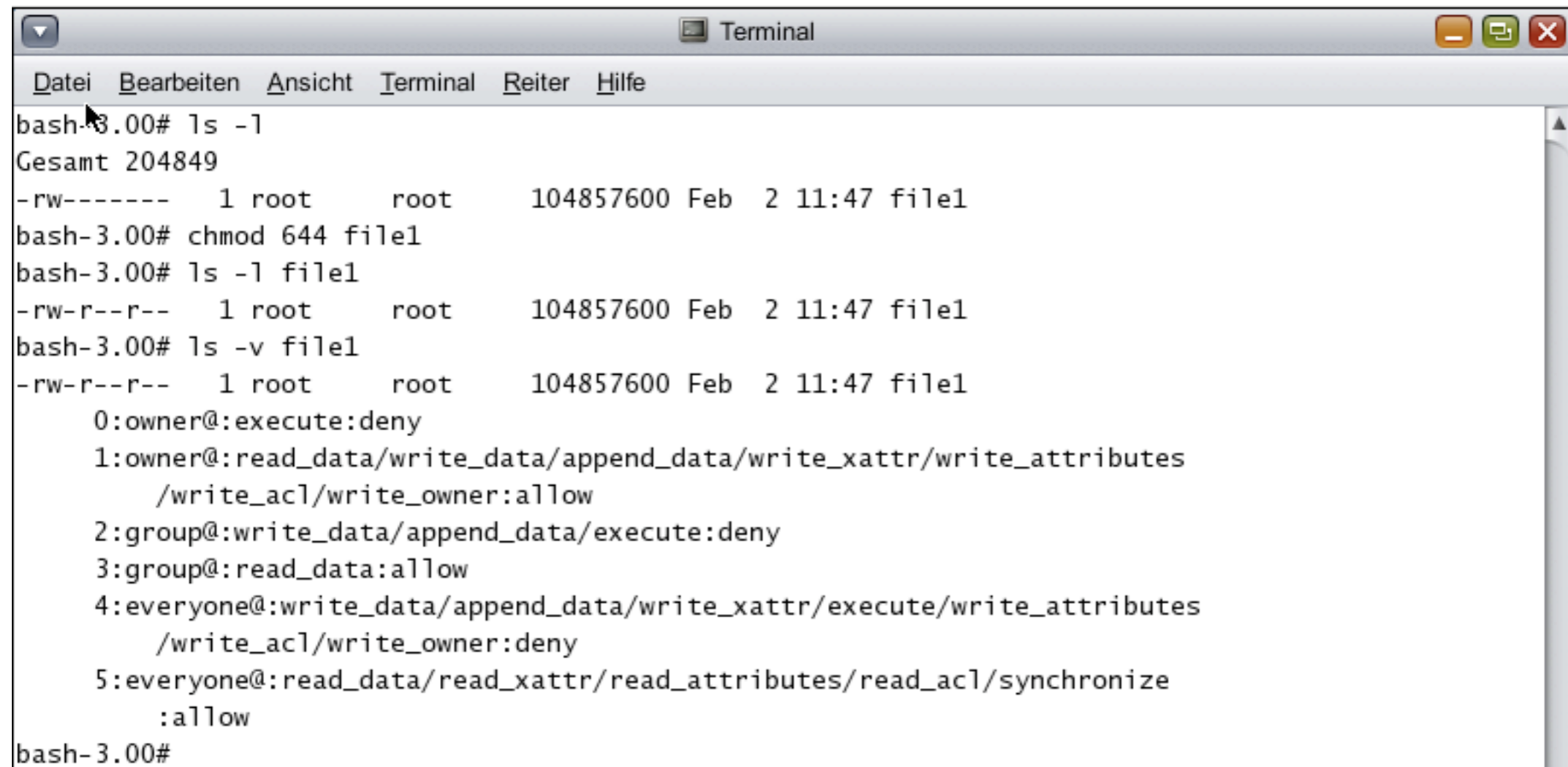
- ▶ Move a pool from a SPARC machine to a Intel machine

```
[on SPARC]  
# zpool export tank
```

copy file...

```
[on Intel]  
# zpool import tank
```

- ▶ New implementation based on NFSv4 specification
- ▶ Much more granular set of access privileges
- ▶ Example:



```
Terminal
Datei Bearbeiten Ansicht Terminal Reiter Hilfe
bash-3.00# ls -l
Gesamt 204849
-rw----- 1 root root 104857600 Feb 2 11:47 file1
bash-3.00# chmod 644 file1
bash-3.00# ls -l file1
-rw-r--r-- 1 root root 104857600 Feb 2 11:47 file1
bash-3.00# ls -v file1
-rw-r--r-- 1 root root 104857600 Feb 2 11:47 file1
0:owner@:execute:deny
1:owner@:read_data/write_data/append_data/write_xattr/write_attributes
/write_acl/write_owner:allow
2:group@:write_data/append_data/execute:deny
3:group@:read_data:allow
4:everyone@:write_data/append_data/write_xattr/execute/write_attributes
/write_acl/write_owner:deny
5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow
bash-3.00#
```

▶ Scripting queries

```
# zpool list -H -o name, size  
pool1    2.0GB  
pool2    3.0GB
```

-H suppress the column headings

-o comma separated list of the columns to display

▶ Recursively destroy a ZFS

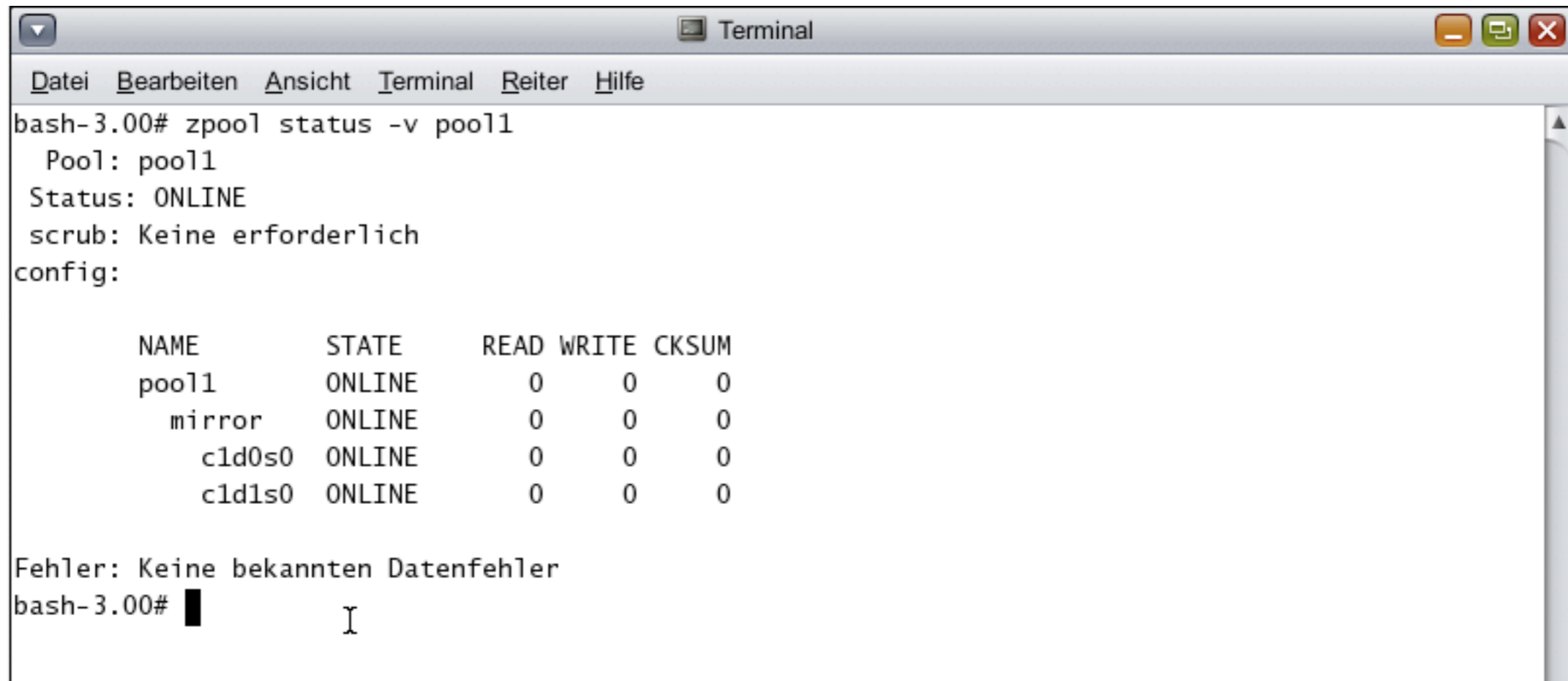


```
# zfs destroy -f pool1
```

► Get all dataset properties

```
Fenstermenü | eiten | Ansicht | Terminal | Reiter | Hilfe
bash-3.00# zfs get all pool1
NAME      PROPERTY          VALUE                SOURCE
pool1     type              filesystem           -
pool1     creation          Fr Feb  2 11:18 2007 -
pool1     used             200M                -
pool1     available         1,71G               -
pool1     referenced        100M                -
pool1     compressratio    1.00x               -
pool1     mounted          yes                  -
pool1     quota            none                 default
pool1     reservation      none                 default
pool1     recordsize       128K                 default
pool1     mountpoint       /pool1               default
pool1     sharenfs         off                  default
pool1     shareiscsi       off                  default
pool1     checksum         on                   default
pool1     compression      off                  default
pool1     atime            on                   default
pool1     devices          on                   default
pool1     exec             on                   default
pool1     setuid           on                   default
pool1     readonly         off                  default
pool1     zoned            off                  default
pool1     snapdir         hidden               default
pool1     aclmode          groupmask            default
pool1     aclinherit       secure               default
pool1     canmount         on                   default
```

► Detailed health status



```
bash-3.00# zpool status -v pool1
Pool: pool1
Status: ONLINE
scrub: Keine erforderlich
config:

    NAME      STATE    READ WRITE CKSUM
    pool1     ONLINE   0     0     0
      mirror  ONLINE   0     0     0
        c1d0s0 ONLINE   0     0     0
        c1d1s0 ONLINE   0     0     0

Fehler: Keine bekannten Datenfehler
bash-3.00# █
```

▶ IO stats

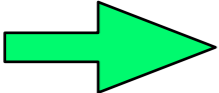
```
Datei Bearbeiten Ansicht Terminal Reiter Hilfe
bash-3.00# zpool iostat pool1 2
          capacity      operations      bandwidth
pool      used  avail  read  write  read  write
-----
pool1     200M  1,74G    0     5    58   513K
pool1     200M  1,74G    0     0     0     0
pool1     200M  1,74G    0     0     0     0
pool1     200M  1,74G    0     0     0     0
pool1     200M  1,74G    0     0     0     0
pool1     200M  1,74G    0     0     0     0
pool1     200M  1,74G    0     0     0     0
pool1     200M  1,74G    0     0     0     0
pool1     300M  1,64G    0    395     0  44,3M
pool1     300M  1,64G    0     0     0     0
^C
bash-3.00#
```

- Usage statistics for the pool tank, taken every 2 seconds until typing CTRL-C
- here: created a file of 100Mbytes

- ▶ SPARC or x86 system running (Open)Solaris
- ▶ Minimum 128Mbytes disk (A storage pool requires min. 64Mbytes)
- ▶ Solaris needs 512Mbytes RAM, 1Gbyte or more is recommended for good ZFS performance

- ▶ GNU/Linux, Fuse
- ▶ FreeBSD
- ▶ Mac OS X Leopard

- ▶ GNU/Linux, Fuse
 - v. 0.4.0alpha1
 - First version with write support
 - Successfully installed on Ubuntu/Kubuntu 6.10
 - First distribution included ZFS: Parted Magic
- ▶ Mac OS X 10.5
- ▶ FreeBSD (Kernel)

- ▶ 1 process
- ▶ 1200 leaf directories
- ▶ 64 files per leaf
- ▶  ca. 700MB of data (plus metadata)

	UFS	ZFS	ext3	reiserfs
time (starting empty)	3:24	0:35	7:28	4:43
time (consecutive run)	11:01	0:38	1:10	2:34

- ▶ Pooled storage
- ▶ Data consistency
- ▶ Transactions
- ▶ Self-Healing
- ▶ Scalability
- ▶ Snapshots and clones
- ▶ Built in compression
- ▶ Performance
- ▶ Dynamic
- ▶ Simple administration

- ▶ Even Solaris can not boot from ZFS - yet...
 - only a “data filesystem”
- ▶ No transparent encryption
- ▶ No secure deletion of single files (overwriting with random files)
- ▶ No support of per user or per group quota, a own FS needs to be created for every user/group
 - Managing thousands of FS might be complicated and time consuming
- ▶ CDDL license (GPL soon?)

- ▶ While reading through (SUN)-ZFS documentation, I got the impression that ZFS is better than sex!
 - “Free your Mind”
 - “End the Suffering”
 - “Blow away 20 years of obsolete assumptions”
 - “... I hear the disks go eeee, errr, bzzz.”

;-)))

- ▶ Collected good features from other FS
- ▶ Easy to manage  There will be many happy administrators ;-)
- ▶ Very powerful
 - Pooled storage, snapshots, clones, compression, scrubbing, RaidZ
- ▶ Secure
 - Recognizes and heals data corruption
- ▶ Fast
- ▶ OpenSource
 - Source and discussion on <http://www.opensolaris.org/os/community/zfs>

Demo