ZFS

- Das endgültige Filesystem

Detlef Drewanz

Ambassador Operating Systems

Sun Microsystems GmbH

http://www.opensolaris.org

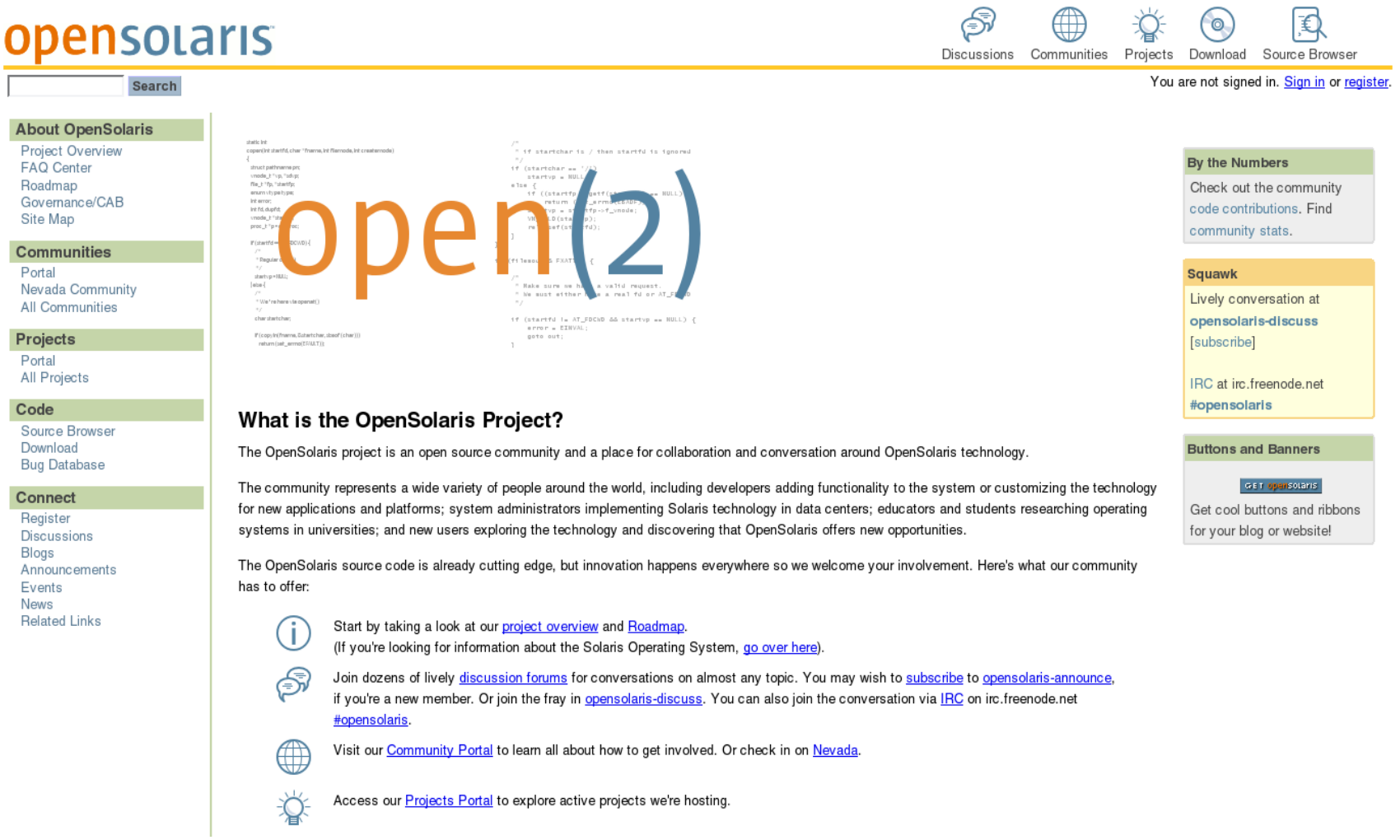
Das opensolaris™ Projekt

<innovation matters>
opensolaris™

Browse
Source code
FAQs
Communities
Projects
Discussions

Download
Source Code
Binaries
Tools

Participate
IRC #opensolaris
Discussions
Blogs
Communities
User Groups
Buttons



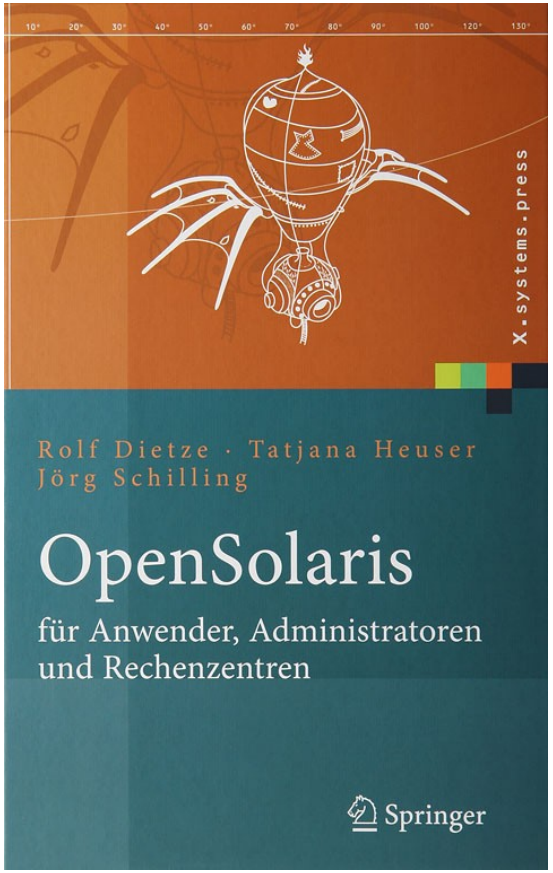
What's New in Solaris Express: http://docs.sun.com/app/docs/doc/819-2183

"How To" Guides

- Consolidating Servers and Applications Guide
- How To Use DTrace
- Eliminating Web Site Hijacking
- Quickly Installing Solaris 10
- Managing ZFS in Solaris Containers
- Coming Soon — Setting up a Solaris optimized Postgres database

Neue Bücher

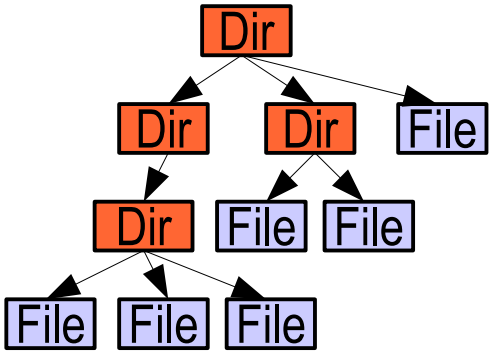
http://docs.sun.com



http://www.solarisinternals.com

Altes Modell / ZFS:

Filesystem:



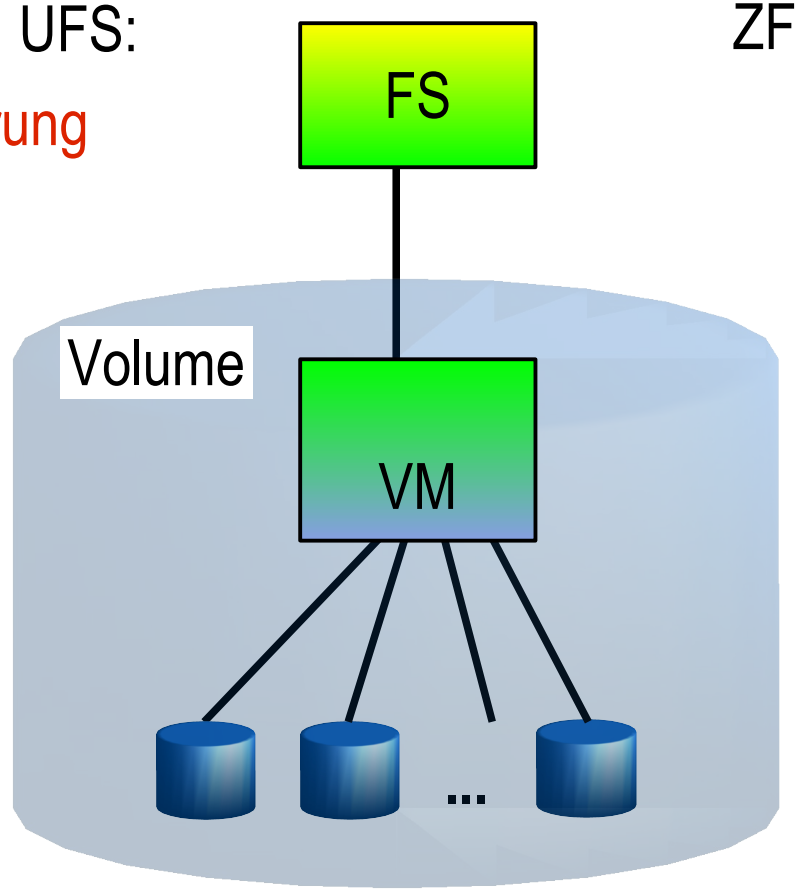
Posix-Interface:

UFS:

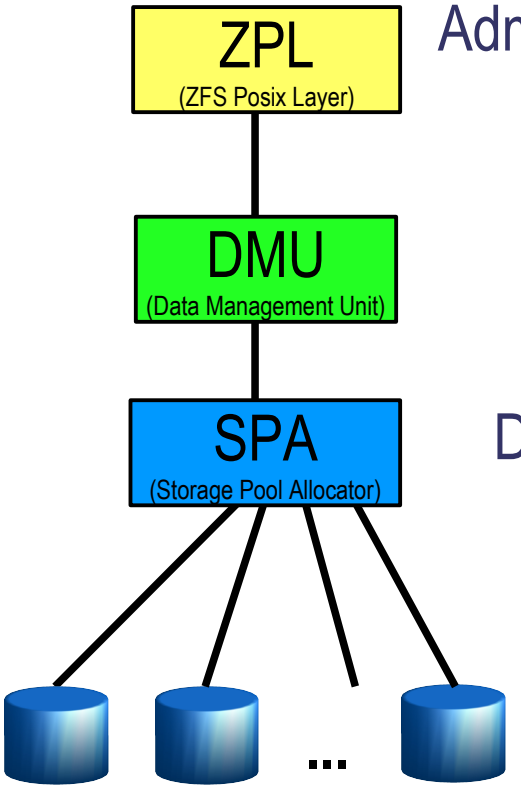
Bit-Optimierung

Abstraktion

Hiding



ZFS:



Admin-Optimierung

Prüfsummen

Transaktionen

Dynamic Striping

Kompression

Self-Healing

HW-Optimierung

ZFS Überblick

Datenmanagement der nächsten Generation

End-to End Daten Integrität

Checksummen für alle Blöcke

Volume Manager integriert

Einfache Administration

ein Filesystem mit einem Kommando

Immense Daten Kapacität

128-bit Filesystem

Dynamische Parameter

Erhebliche Performance Zuwächse

Für Geschwindigkeit optimiert

Existierende Filesysteme

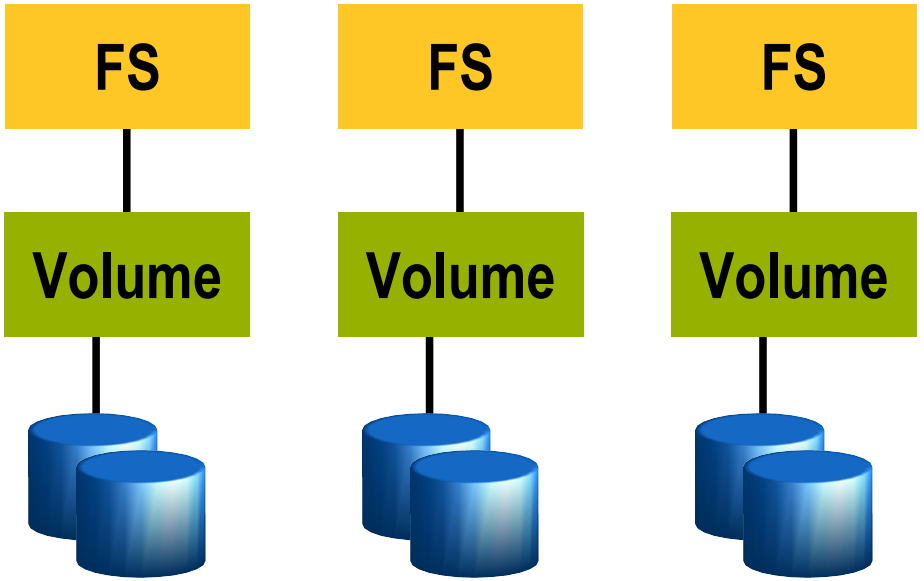
- Datenkorruption ist möglich
  - Maximal Prüflesen direkt nach Schreiben
  - Volume Manager kennt nicht die Struktur der Daten
  - Filesystem kennt nicht den Spiegel / das physikalische Layout
- Hoher Administrationsaufwand
  - Ein Filesystem erfordert (nacheinander!):
    - Partitionieren
    - Volume erzeugen (Wartezeit)
    - Filesystem erzeugen (Wartezeit)
    - vfstab Eintrag
    - dfstab Eintrag
  - grow/shrink ist zeitaufwendig

ZFS Design Prinzipien

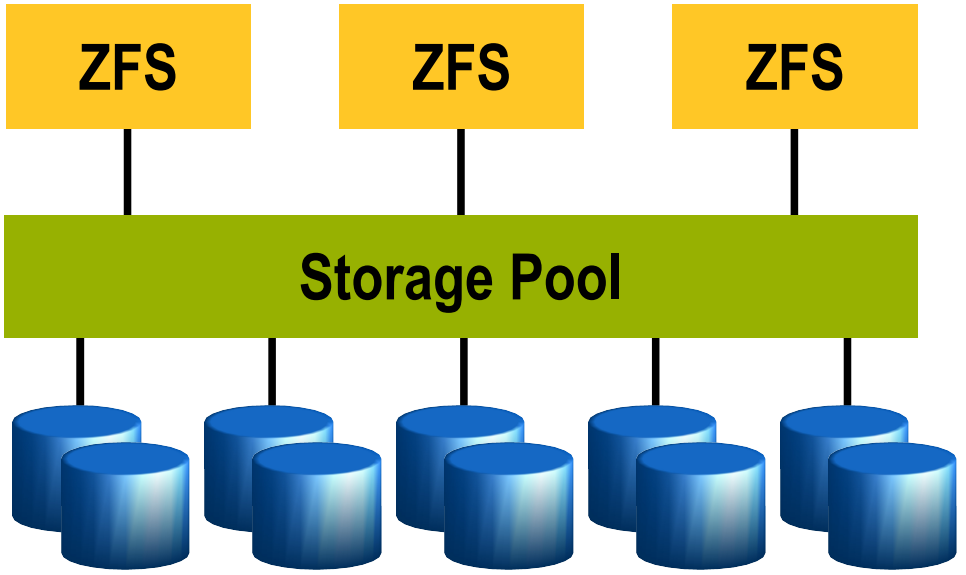
- Pooled Storage
  - Filesystem auf mehreren Disks
  - Filesystem kann hierarchisch weiter aufgeteilt werden
- End-to-end Daten-Integrität
  - Prüfsummen auf jedem Block
  - Heutige Maschinen können das (früher: zu teuer)
  - Andere Filesysteme eigentlich nicht haltbar
- Transaktionen für jede Veränderung im Filesystem
  - fsck niemals notwendig
    (Filesystem mit Volumes ist auf IO-Reihenfolge angewiesen)
  - Erhöht die Performance

FS/Volume Modell vs. ZFS

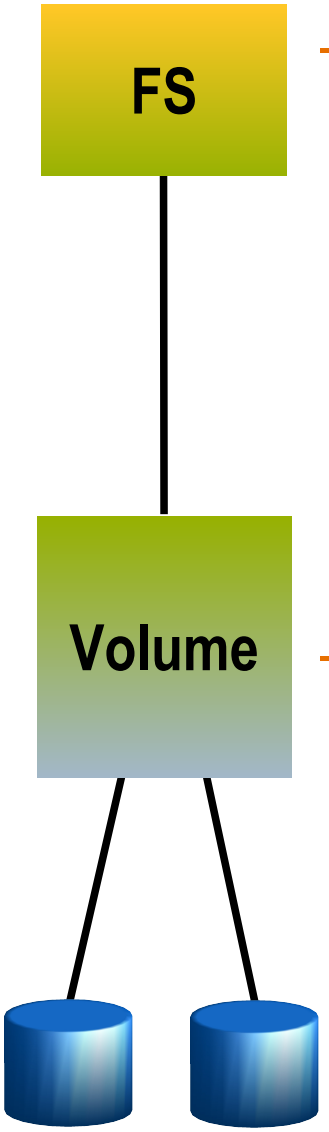
- Traditionelle Volumes
  - Abstraktion: virtuelle Disk (fest)
  - Volume für jedes Filesystem
  - Grow/shrink nur koordiniert
  - Bandbreite / IOs aufgeteilt
  - Fragmentierung des freien Platzes



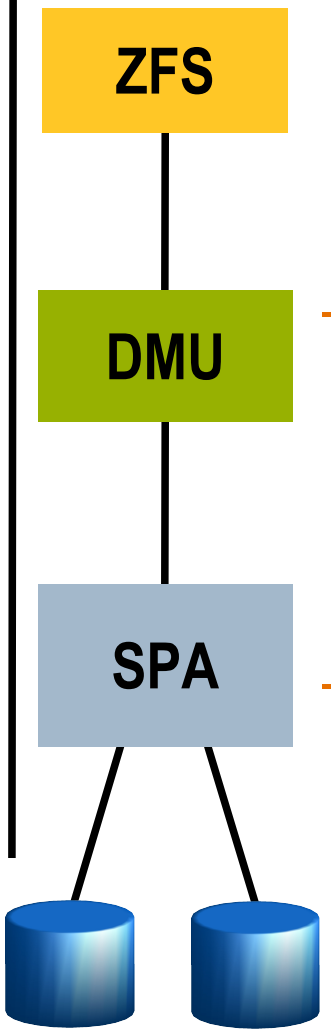
- ZFS Pooled Storage
  - Abstraktion: Datei (variabel)
  - Keine feste Platzeinteilung
  - Grow/shrink via Schreiben/Löschen
  - Volle Bandbreite / IOs verfügbar
  - Freier Platz wird geshart



FS/Volume Model vs. ZFS



- FS/Volume I/O Stack
  - Block Device Interface
  - Schreib-Reihenfolge erzeugt quasi-Integrität
  - Stromausfall: Metadaten konsistent mittels fsck
  - Journaling (nur Metadaten)
- Block Device Interface
  - Muß so schreiben wie es vom FS kommt
  - Stromausfall: ggf. Spiegel ungleich
  - Langsam, da synchron

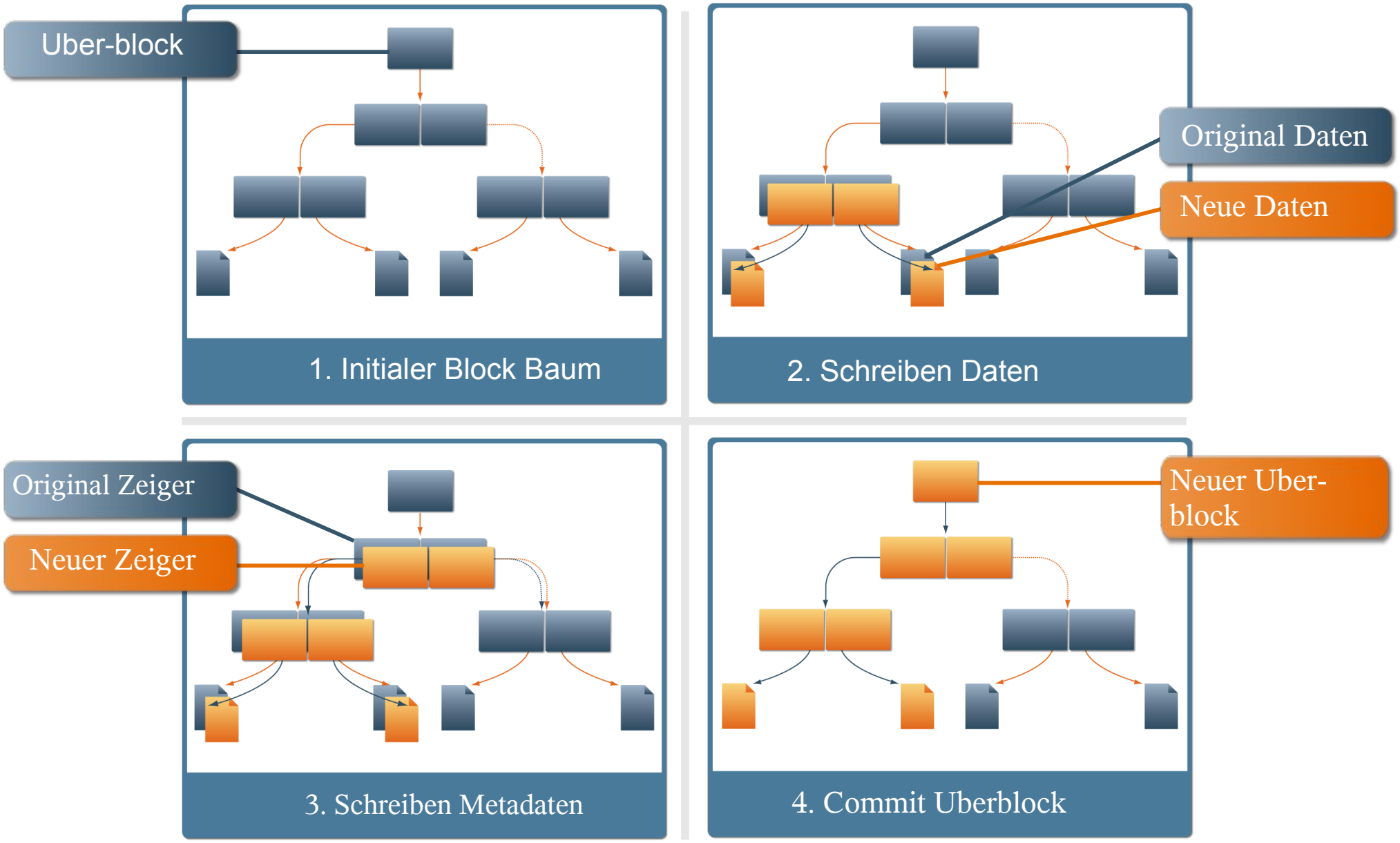


- ZFS I/O Stack
  - Object-basierte Transaktionen
  - Daten-Änderungen => Transaktionen
  - All-or-nothing!
- Transaction Group Commit
  - Daten-Änderung und zugehörige Metadaten => Transaktions-Gruppen
  - Commit mittels COW
- Transaction Group Batch I/O
  - Zusammenfassung + Optimierung
  - Stromausfall -> alter Zustand
  - Platten werden ausgenutzt

ZFS Daten Integrität

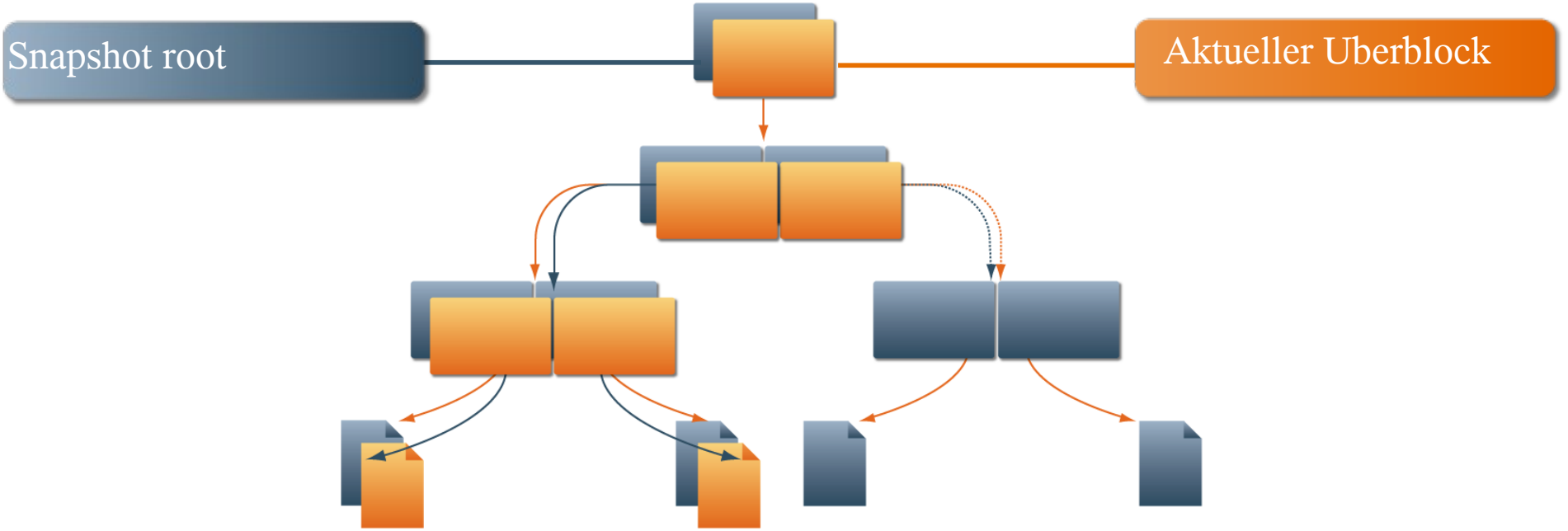
- Copy-On-Write
  - Daten werden niemals überschrieben (= Rücksetzen ist möglich)
  - Neuer Zustand wird in freien Blöcken vorbereitet
- Transaktionen (ausschließlich)
  - Neue uberblock-Version -> neuer Zustand
  - Transaktionen umfassen auch die Daten im FS
  - Journaling nicht notwendig!
- Prüfsummen auf jedem Block des Filesystems
  - Daten und Metadaten
  - Uberblock via Seriennummer und CRC
  - Filesystem Header

Copy-on-Write Transaktion



ZFS Snapshot

- read-only point-in-time Kopie eines File Systems
- Copy-on-write ist als Mechanismus bereits vorhanden
- Platzsparend - nur Änderungen werden gespeichert
- Sofort verfügbar - einfach die Kopie nicht löschen



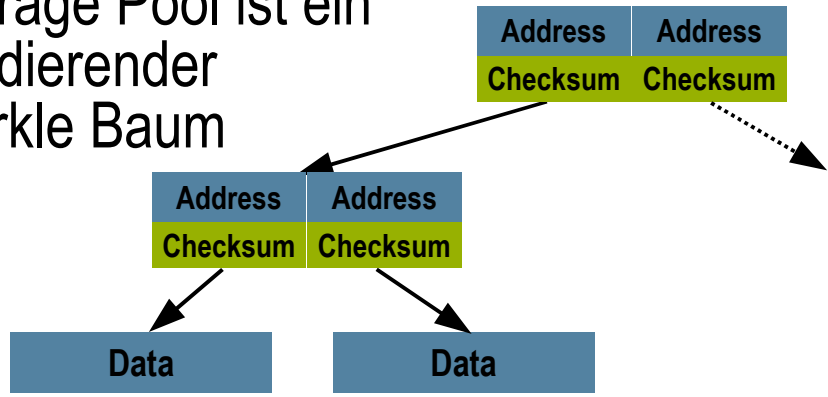
End-to-End Daten Integrität

- Disk Block Prüfsummen
  - Prüfsummen bei Datenblock
  - Auf Disks meist kurz (Fehler unentdeckt)
- Einige Disk Fehler bleiben unentdeckt

Nur Fehler auf Medium erkennbar

- ✔ Bit rot
- ✗ Phantom writes
- ✗ Misdirected reads and writes
- ✗ DMA parity errors
- ✗ Driver bugs
- ✗ Accidental overwrite

- ZFS Daten Integrität
  - Prüfsumme bei Adresse
  - Gemeinsamer Fehler: unwahrscheinlich
  - Storage Pool ist ein validierender Merkle Baum

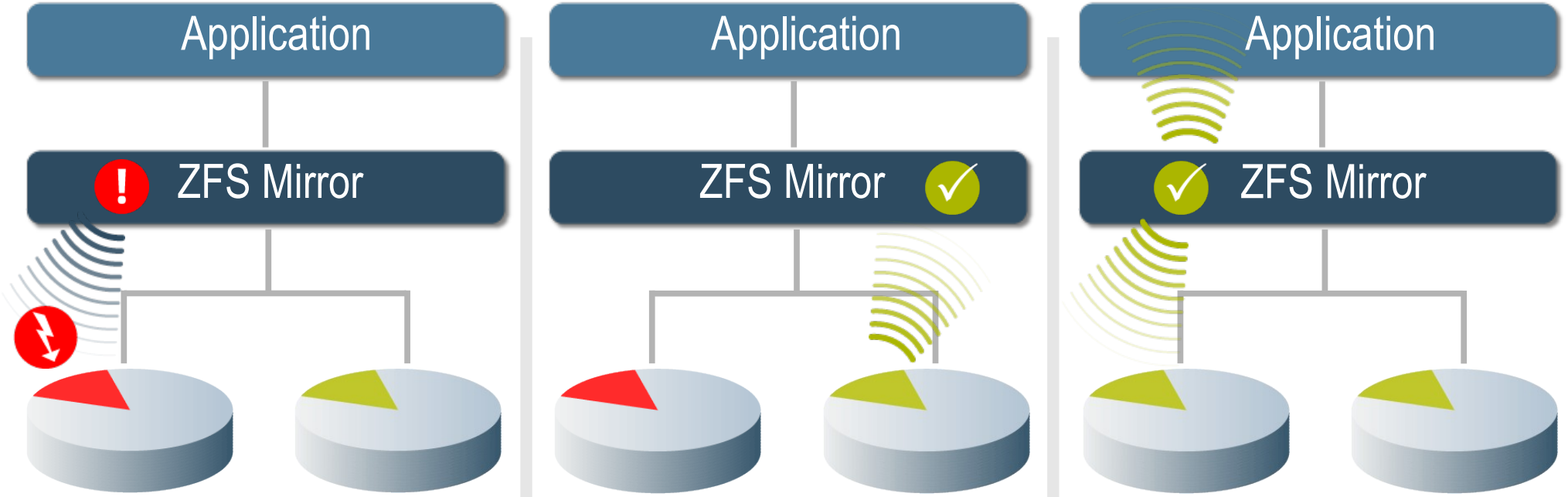


ZFS validiert alle Blöcke

- ✔ Bit rot
- ✔ Phantom writes
- ✔ Misdirected reads and writes
- ✔ DMA parity errors
- ✔ Driver bugs
- ✔ Accidental overwrite

Self-Healing in ZFS

- Erkennung kaputter Datenblöcke durch Checksummen
- Blöcke durch gespiegelte Blöcke reparieren



Detects Bad Data

Gets Good Data from Mirror

“Heals” Bad Copy

Disk Scrubbing

- Alle Daten werden gelesen und ggf. korrigiert
  - mirror: alle Spiegel-Seiten
  - raidz: alle Daten und Parity-Blöcke
- Daten Integrität erhalten für lange unbenutzte Daten
  - Wird auch für Wiederherstellung von Spiegeln benutzt

ZFS: RAID-Z

- wie RAID-5, mit
  - > Dynamische Größe des Stripe
    (4+1, wenn wenig Daten dann vielleicht 2+1)
  - > Dynamische Block-Größe
  - > Immer full-stripe write (kein read-modify-write): Performance
  - > Neuer Zustand durch COW-Transaktion (write hole entfällt)
    - – RAID-Z braucht keinen Cache mit Batterie
- Datenkorruption wird wie bei Spiegel erkannt
- Keine spezielle HW notwendig: “ZFS loves cheap disks”
- In Planung Raid-Z2 (Dual Parity)
  - > Bereits in OpenSolaris enthalten

ZFS Skalierbarkeit

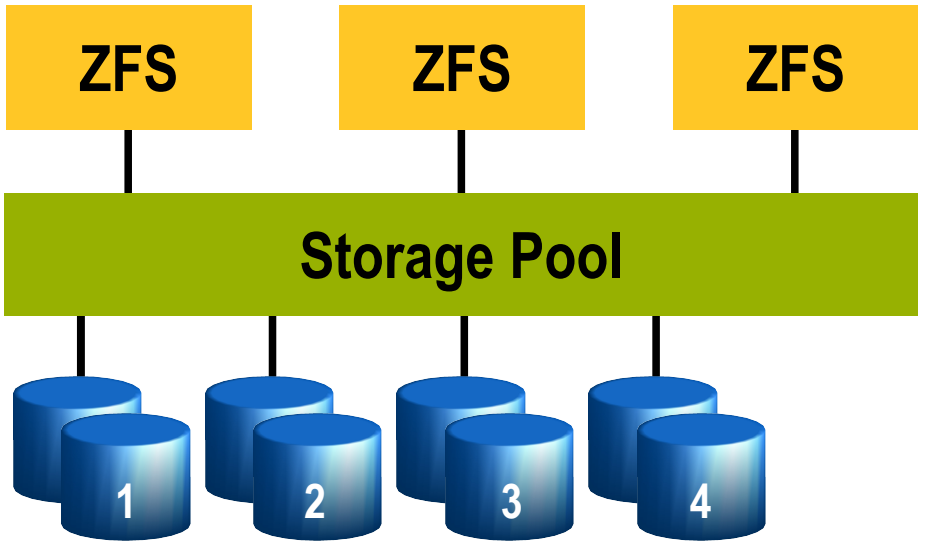
- Immense Kapazität (128-bit)
  - Moore's Law: Bit 65 bis 2020 notwendig
  - Zetabyte = 70-bit (ISO Vorsilbe für 1 Milliarde TB)
    - tera = 10^{12} ≈ 2^{40} heutige Disks
    - peta = 10^{15} ≈ 2^{50} übliche Unternehmens Datenmenge
    - exa = 10^{18} ≈ 2^{60} übliche Unternehmens Datenmenge
    - Zeta = 10^{21} ≈ 2^{70} größte ISO Vorsilbe
    - ??? = 10^{36} ≈ 2^{128} max Größe eines ZFS
  - Max Filegröße 2^{63}-1 (noch?) durch Posix API limitiert
- Metadaten: dynamisch, automatisch anpassend
- Einstellungen: online durch Attribute
- Parallele Nutzung aller Basis-Disks

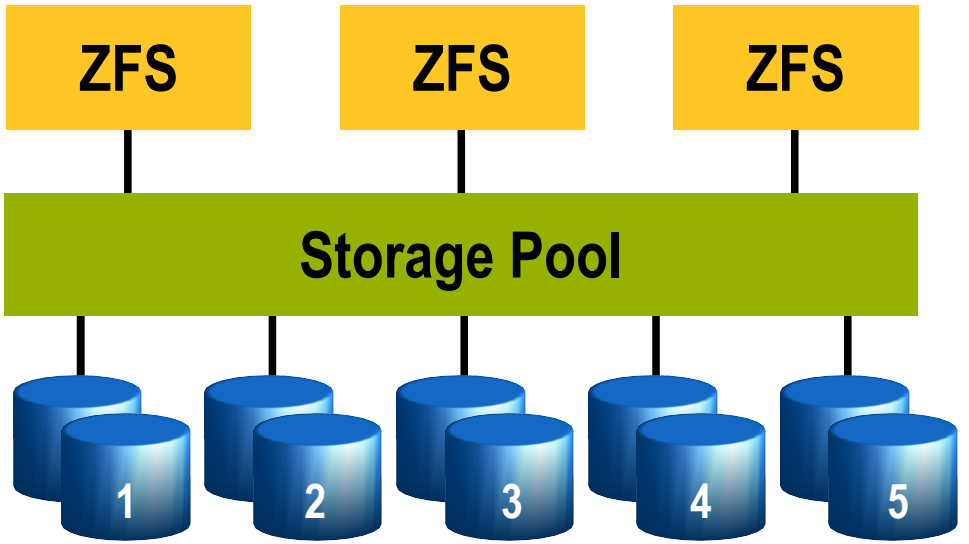
ZFS Performance

- Copy-on-write Design
  - Geschriebene Daten werden neu allokiert (sonst sind Transaktionen nicht rücksetzbar)
  - Viele kleine Random writes => wenige größere sequential writes
- Multiple Blockgrößen
  - Automatische Anpassung nach Datenaufkommen
- Pipelined I/O
  - Scoreboard Pipeline (24 Stufen) mit I/O Abhängigkeits-Analyse
  - Parallelisierung über Disks
  - Priority, deadline scheduling, out-of-order issue, sorting, aggregation
  - Dynamic striping, multi-user prefetch pro Datei

Dynamic Striping

- Automatische Migration von 4 auf 5 Spiegel
  - Hinzufügen von 5. Spiegel ->
  - write: 5-fach Stripe
  - existierende Daten bleiben bestehen

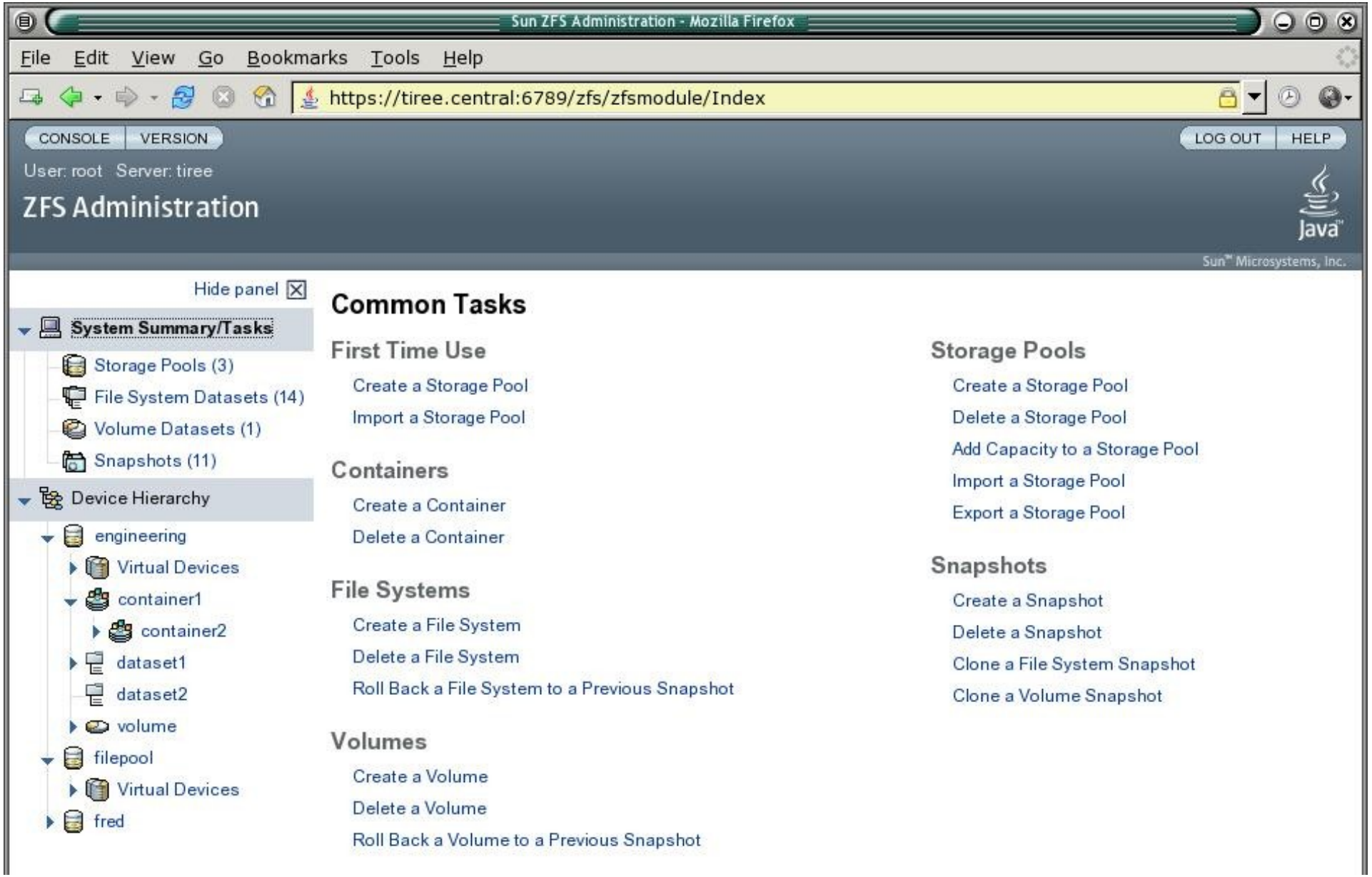


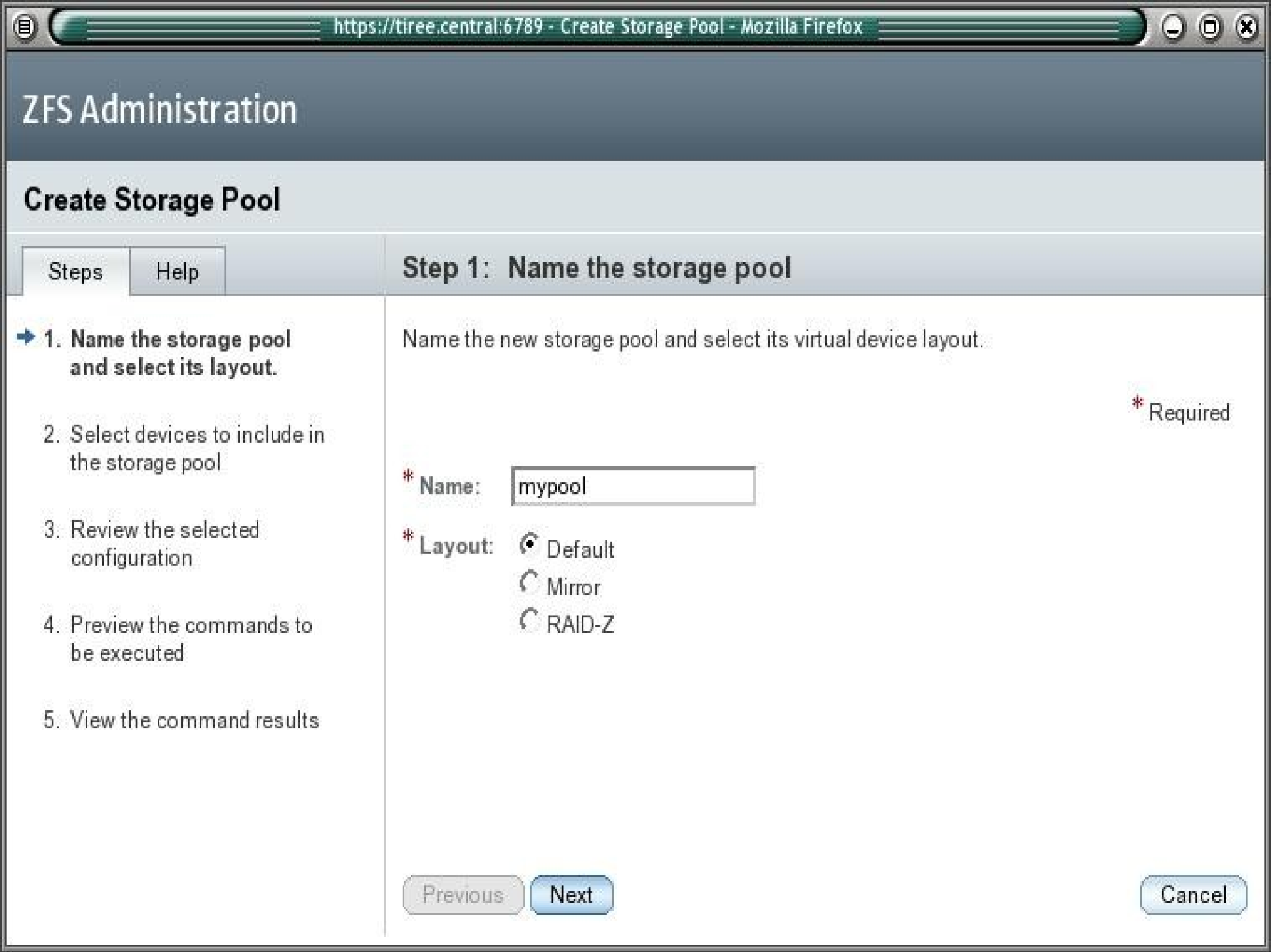


ZFS Administration per Web

- Graphisches User Interface via Web mitgeliefert.
- Einschalten Webserver
  - > /usr/sbin/smcwebserver enable
  - > /usr/sbin/smcwebserver start
- Webserver ist erreichbar unter
  - > https://localhost:6789/zfs

ZFS GUI


Sun ZFS Administration - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://tiree.central:6789/zfs/zfsmodule/Index
CONSOLE
VERSION
LOG OUT
HELP
User: root Server: tiree
ZFS Administration
Java
Sun™ Microsystems, Inc.
Hide panel
System Summary/Tasks
Storage Pools (3)
File System Datasets (14)
Volume Datasets (1)
Snapshots (11)
Device Hierarchy
engineering
Virtual Devices
container1
container2
dataset1
dataset2
volume
filepool
Virtual Devices
fred
Common Tasks
First Time Use
Create a Storage Pool
Import a Storage Pool
Containers
Create a Container
Delete a Container
File Systems
Create a File System
Delete a File System
Roll Back a File System to a Previous Snapshot
Volumes
Create a Volume
Delete a Volume
Roll Back a Volume to a Previous Snapshot
Storage Pools
Create a Storage Pool
Delete a Storage Pool
Add Capacity to a Storage Pool
Import a Storage Pool
Export a Storage Pool
Snapshots
Create a Snapshot
Delete a Snapshot
Clone a File System Snapshot
Clone a Volume Snapshot


https://tiree.central:6789 - Create Storage Pool - Mozilla Firefox
ZFS Administration
Create Storage Pool
Steps
Help
1. Name the storage pool and select its layout.
2. Select devices to include in the storage pool
3. Review the selected configuration
4. Preview the commands to be executed
5. View the command results
Step 1: Name the storage pool
Name the new storage pool and select its virtual device layout.
* Required
* Name:
mypool
* Layout:
Default
Mirror
RAID-Z
Previous
Next
Cancel

ZFS: Erzeugen

- Das einfachste ZFS Filesystem

```
# zpool create tank c1t0d0
```

  - Erzeugt einen zpool namens tank auf der Platte c1t0d0 (Inklusive Erzeugen eines EFI-Labels auf der Platte)
  - Erzeugt ein Filesystem des gleichen Namens
  - Mounted das Filesystem unter /tank
  - Merkt sich die Einstellung für Mount nach Reboot

ZFS: Erzeugen (2)

- Gespiegelter zpool mit Filesystem

```
# zpool create tank mirror c0t0d0 c1t0d0
```

- Hierarchisch untergeordnetes Filesystem mit Mountpoint

```
# zfs create tank/home
# zfs set mountpoint=/export/home tank/home
```

- Home Directories (für das ZFS Team)
  Automatisch unter /export/home/{ahrens,bonwick,billm}
  durch Vererbung

```
# zfs create tank/home/ahrens
# zfs create tank/home/bonwick
# zfs create tank/home/billm
```

- Vergrößern des zpool

```
# zpool add tank mirror c2t0d0 c3t0d0
```

ZFS Properties

- NFS-export aller Directories

```
# zfs set sharenfs=rw tank/home
```

- Kompression (gesamter zpool)

```
# zfs set compression=on tank
```

- Quota (Limit nach oben)

```
# zfs set quota=10g tank/home/eschrock
```

- Reservation (garantierter Platz)

```
# zfs set reservation=20g tank/home/tabriz
```

ZFS Snapshots

- Snapshot ist read/only (sofort und beliebig viele)
  - quasi: Kopie des uberblock
    - Platz wird nur für die Unterschiede gebraucht
    - Snapshot ist sichtbar in .zfs/snapshot im Root des Filesystems
      - Benutzer können selbst Zugriff auf alten Stand nehmen
- Snapshot erzeugen

```
# zfs snapshot tank/home/joeuser@tuesday
```

- Rollback: Filesystem zurück auf alten Stand

```
# zfs rollback tank/home/joeuser@monday
```

- Zugriff auf alte Version via Filesystem

```
$ cat ~joeuser/.zfs/snapshot/wednesday/foo.c
```

ZFS Clones

- Clone ist eine schreibbare Verision des Snapshot
  - > sofort und beliebig viele
  - > Mittel für Varianten eines FS

- Erzeugen eines Clone

```
# zfs clone tank/solaris@monday tank/ws/lori/fix
```

ZFS Serialisierung

- Serielle Repräsentation eines ZFS Snapshots
  - > Nutzung für Backup und Restore (Snapshot: fester Zustand)
  - > Bewegen eines Filesystems zwischen Systemen oder Disks
  - > Inkrementelle Serialisierung (Unterschiede zwischen Snapshots)

- Full backup

```
# zfs send tank/fs@A >/backup/A
```

- Incremental backup

```
# zfs send -i tank/fs@A tank/fs@B >/backup/B-A
```

- Remote replication: send incremental once per minute

```
# zfs send -i tank/fs@11:31 tank/fs@11:32 |
  ssh host zfs receive -d /tank/fs
```

ZFS Daten Migration

- Endian neutrales on-disk format
  - Strukturen enthalten endianness, ZFS interpretiert beide
  - ZFS schreibt in der endianess des Hosts (kein Verlust)
- ZFS enthält alle Informationen des Data Management
  - Label, Device Pfade, Striping, Mount-Point, NFS-Share
  - Ein integrierter Punkt zur Administration
  - Platten werden an anderen Stellen erkannt (auch als Kopien)
- Export zpool

```
old# zpool export tank
```

- Import nach Umbau der Platten

```
new# zpool import tank
```

ZFS vereinfacht die Administration

- Pooled storage - kein Volume Manager mehr erforderlich
  - > Platz wird geshart benutzt, keine Fragmentierung mehr
- Nur zwei Kommandos: **zpool, zfs**
- Schnelle und einfache Erzeugung von Filesystemen
- Autom. Management von Mountpunkten und nfs-shares
- Vererbung von Properties in hierarchischen Filesystemen
  - > Policies per Filesystem oder für Hierarchie-Ebenen setzen
  - > Mount-Point, NFS-Share, Quota, Reservierung, Compression, ...
- Properties dynamisch ändern

ZFS vereinfacht typische Aufgaben

- Keine Wartezeit beim Erzeugen von zpools und zfs
- Kostenloser Schutz von irrtümlichem Löschen
  - Snapshots zum Aufheben von Versionen nutzen
- Einfache Filesystemreplikation
  - send/receive (auch incrementell) für lokale oder remote ZFS
- Vereinfacht system life cycle management (*)
  - Upgrade, Live Upgrade, Patching
- Vereinfacht Datenmanagement für lokale Zonen
  - Einfache Delegation von Datasets in Zonen
  - Quotas für Platzlimitierung nutzen

* work in progress

ZFS Security Features

- ACLs wie bei NFSv4 und Windows NT-style
  - Allow/deny mit Vererbung
- Daten-Authentifizierung mittels Prüfsummen
  - Benutzer-definierbarer Algorithmus (auch SHA-256)
  - Daten können nur schwer verfälscht werden
  - uberblock Prüfsumme ist die Signatur für den gesamten Pool
- Encryption (future)
  - Wichtig für mobile Geräte und zugängliche Platten
  - Auswählbarer Algorithmus aus Solaris Security Framework
- Sicheres Löschen (future)
  - Freigegebene Blöcke werden gelöscht

ZFS: Zukünftige Updates

- Entfernen von Devices aus Zpool (**zpool remove**)
- ZFS root Filesystem
  - Boot von einem ZFS Filesystem (/ ist in ZFS)
  - Live Upgrade mit Clones
- Verschlüsselung mit Solaris Security Framework
- Sicheres Löschen (Überschreiben mit Random Daten)
  - Bei Löschen von Dateien
  - Alte Seiten bei Copy-On-Write
- Verfügbarkeit
  - Fault Management Architecture
  - Hot Spares / Hot Space
  - Dtrace für RZ Einsatz

ZFS - Zusammenfassung

- Einfach
  - Macht einfach das, was der Administrator vor hat.
- Mächtig
  - Pooled storage, Snapshots, Clones, Compression, Scrubbing, RAID-Z
- Sicher
  - Entdeckt und Korrigiert (!!!) stille Daten-Korruption.
- Schnell
  - Dynamisches Striping, Intelligentes Prefetch, Pipelined I/O
- Offen
  - Source und Diskussion unter http://www.opensolaris.org/os/community/zfs
- Kostenfrei

Vielen Dank

Detlef Drewanz
Detlef.Drewanz@sun.com