ZFS The Future Of File Systems

C Sanjeev Kumar
Charly V. Joseph
Mewan Peter D'Almeida
Srinidhi K.

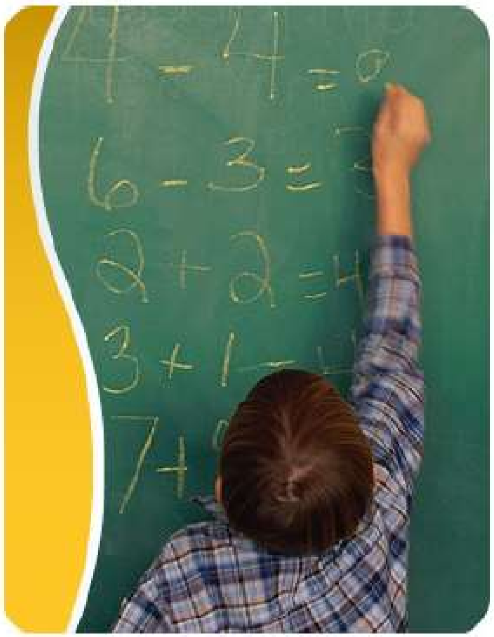
Introduction

What is a File System?

File systems are an integral part of any operating systems with the capacity for long term storage

- present logical (abstract) view of files and directories
- facilitate efficient use of storage devices
- Example-NTFS,XFS,ext2/3...etc

Why a New File System?



Data Management Costs are High

The Value of Data is Becoming Even More Critical

The Amount of Storage is Ever-Increasing

Trouble with Existing File Systems?

Good for the time they were designed, but...

No Defense Against Silent Data Corruption

Any defect in datapath can corrupt data... **undetected**

Difficult to Administer−Need a Volume Manager

Volumes, labels, partitions, provisioning and lots of limits

Older/Slower Data Management Techniques

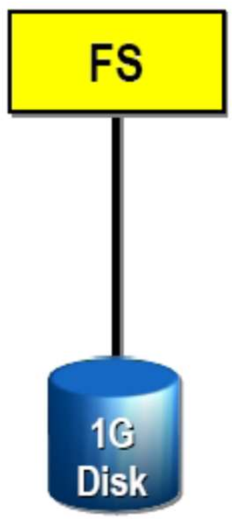
Fat locks, fixed block size, naive pre-fetch, dirty region logging

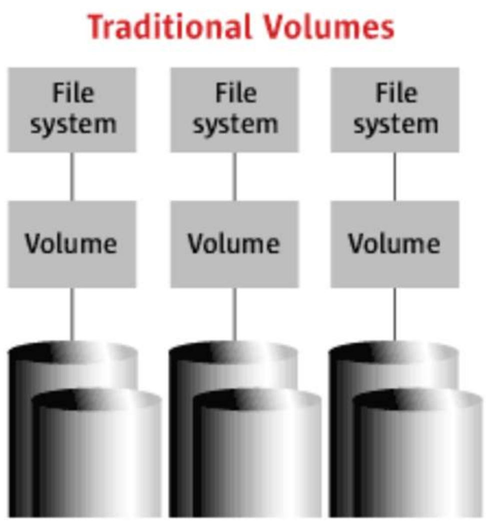
ZFS Design Principles

- Start with a new design around today's requirements
- Pooled storage
  - Eliminate the notion of volumes
  - Do for storage what virtual memory did for RAM
- End-to-end data integrity
  - Historically considered too expensive.
  - Now, data is too valuable not to protect
- Transactional operation
  - Maintain consistent on-disk format
  - Reorder transactions for performance gains – big performance win

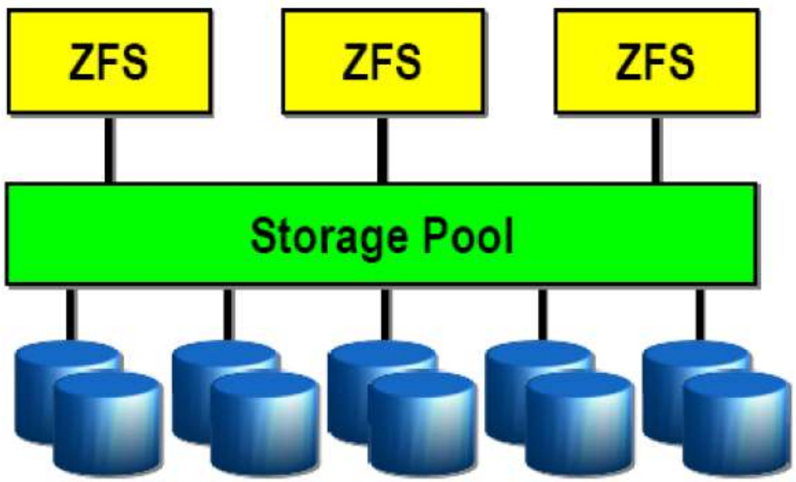
Evolution of Disks and Volumes

- Intially, we had simple disks
- Abstraction of disks into volumes to meet requirements
- Industry grew around HW / SW volume management





Zpool



- ZFS file systems are built on top of virtual storage pools called **zpools.**
- A zpool is constructed of devices, real or logical.
- They are constructed by combining block devices using either mirroring or RAID-Z.

ZFS Disk Management

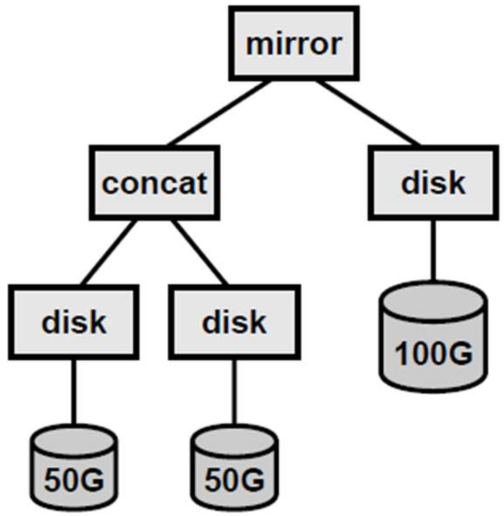


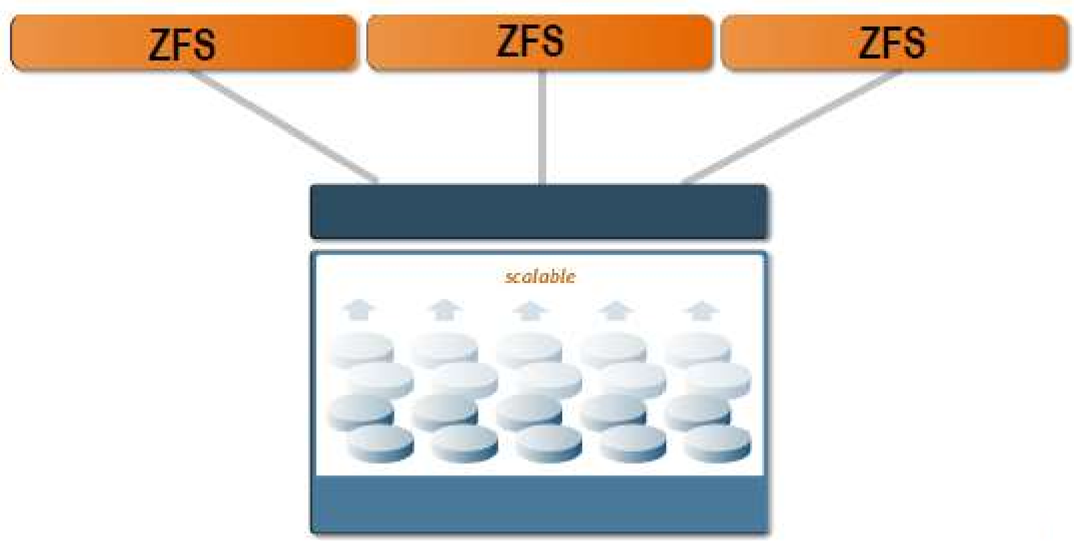
Figure 5: Example vdev with the description `mirror(concat(/dev/dsk/a,/dev/dsk/b),/dev/dsk/c)` where disks `a` and `b` are the 50 GB disks and disk `c` is the 100 GB disk.

These operations are supported in the SPA.
ZFS also implements "RAID-Z," which is RAID-5-like but designed to be resilient to failures during write of a stripe.

FS/Volume Model vs. ZFS

Traditional Volumes	ZFS Pooled Storage
Partitions/volumes exist in traditional file Systems.	With ZFS's common storage pool, there are no partitions to manage.
With traditional volumes, storage is fragmented and stranded. Hence storage utilization is poor.	File systems share all available storage in the pool, thereby leading to excellent storage utilization.
Since traditional file systems are constrained to the size of the disk, so growing file systems and volume is difficult.	Size of zpools can be increased easily by adding new devices to the pool. Moreover file systems sharing available storage in a pool, grow and shrink automatically as users add/remove data.

Traditional Volumes	ZFS Pooled Storage
Each file system has limited I/O bandwidth.	The combined I/O bandwidth of all the devices in the storage pool is always available to each file system.
Configuring a traditional file system with volumes involves extensive command line or graphical user interface interaction and takes many hours to complete.	Creation of a similarly sized Solaris ZFS file system takes a few seconds.

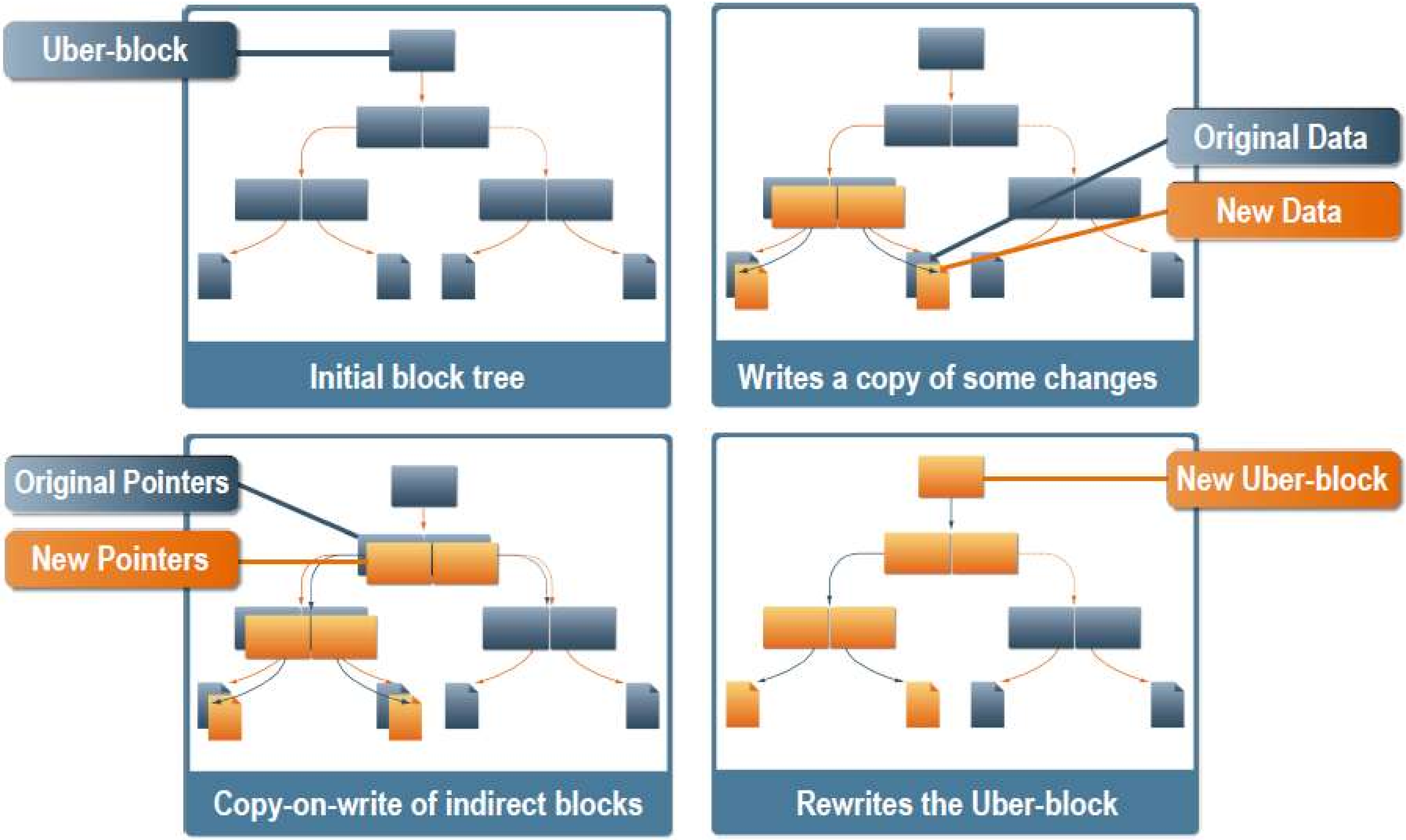


Data Integrity

ZFS Data Integrity Model

- Everything is copy-on-write
  - > Never overwrite live data
  - > On-disk state always valid – no fsck
- Everything is transactional
  - > Related changes succeed or fail as a whole
  - > No need for journaling
- Everything is checksummed
  - > No silent corruptions
  - > No panics from bad metadata
- Enhanced data protection
  - > Mirrored pools, RAID-Z, disk scrubbing

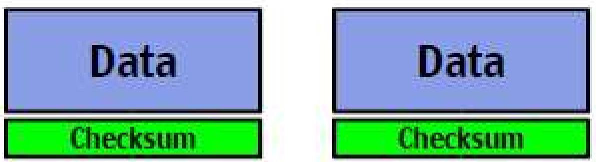
Copy-on-Write and Transactional



End-to-End Checksums

Disk Block Checksums

- Checksum stored with data block
- Any self-consistent block will pass
- Can't even detect stray writes
- Inherent FS/volume interface limitation

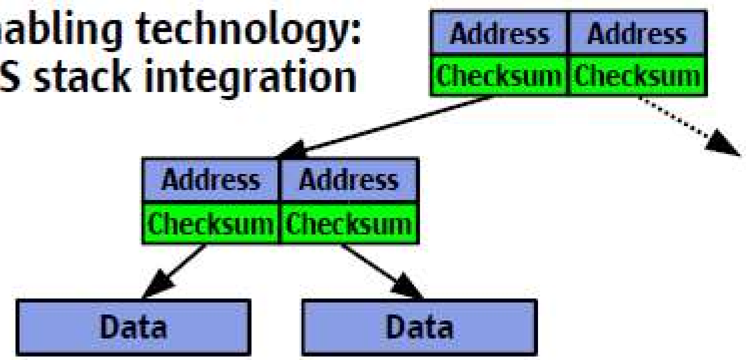


Only validates the media

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

ZFS Checksum Trees

- Checksum stored in parent block pointer
- Fault isolation between data and checksum
- Entire pool (block tree) is self-validating
- Enabling technology:
  ZFS stack integration



Validates the entire I/O path

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

ZFS Block Pointer

- Pointer can refer to up to 3 copies of the block
- Block size isn't fixed
- Blocks can be stored compressed
- PSIZE is physical size, LSIZE is logical size (ASIZE includes indexing overhead
- checkum[0-3] are copies of the block's checksum value
- Blocks have a type (e.g., to indicate whether it's a data block or an indirect block)

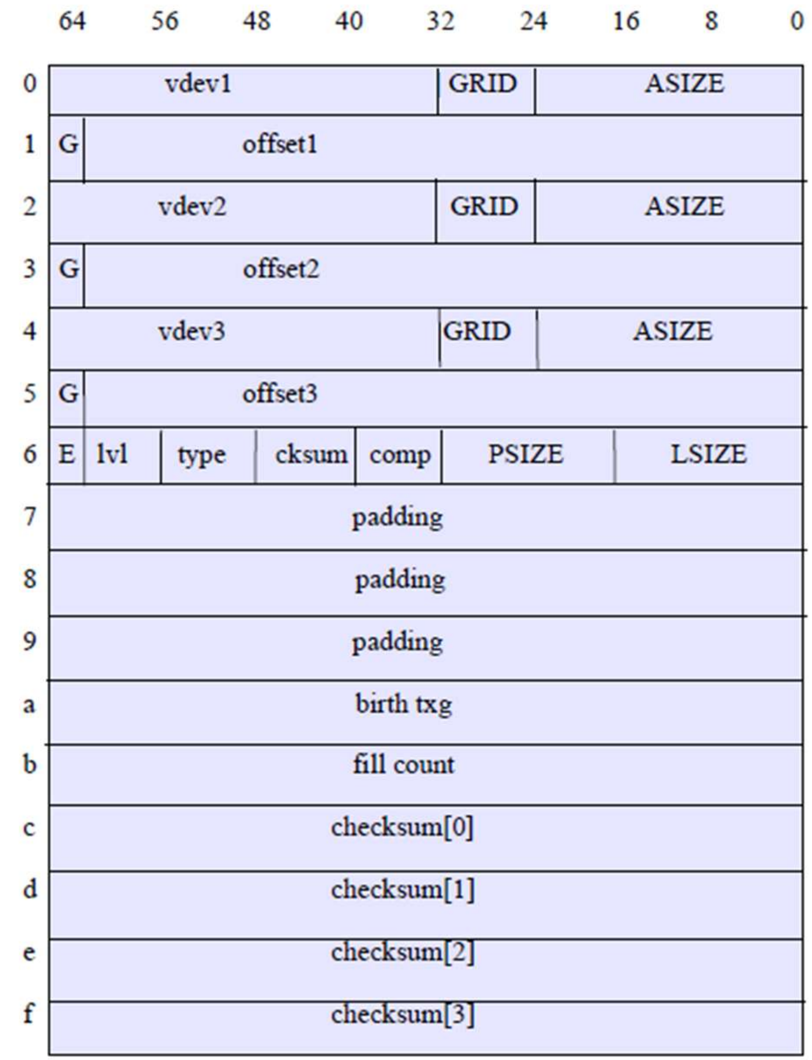
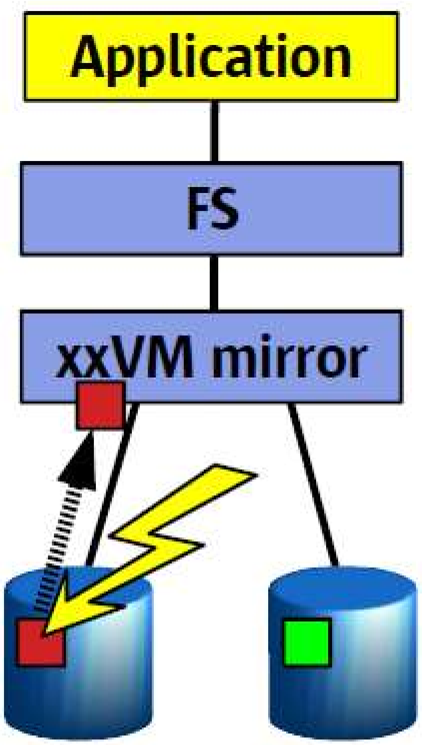


Illustration 8 Block pointer structure showing byte by byte usage.

Traditional Mirroring

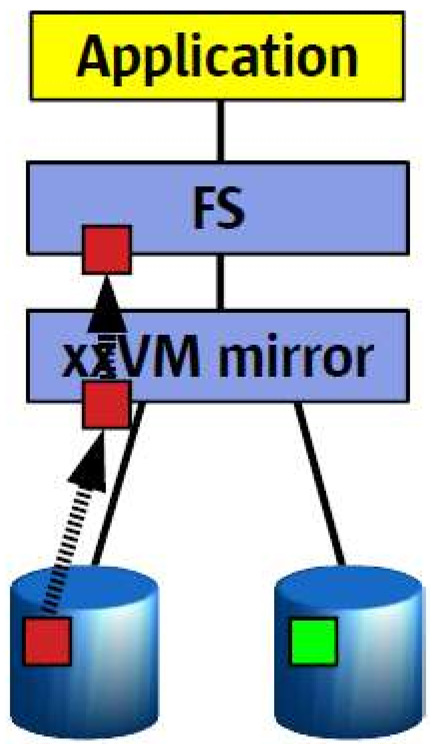
1. Application issues a read. Mirror reads the first disk, which has a corrupt block.

It can't tell.

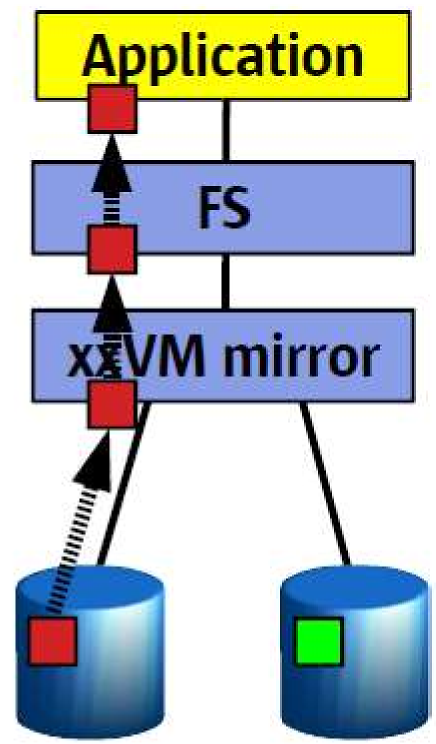


2. Volume manager passes bad block up to filesystem.

If it's a metadata block, the filesystem panics. If not...



3. Filesystem returns bad data to the application.

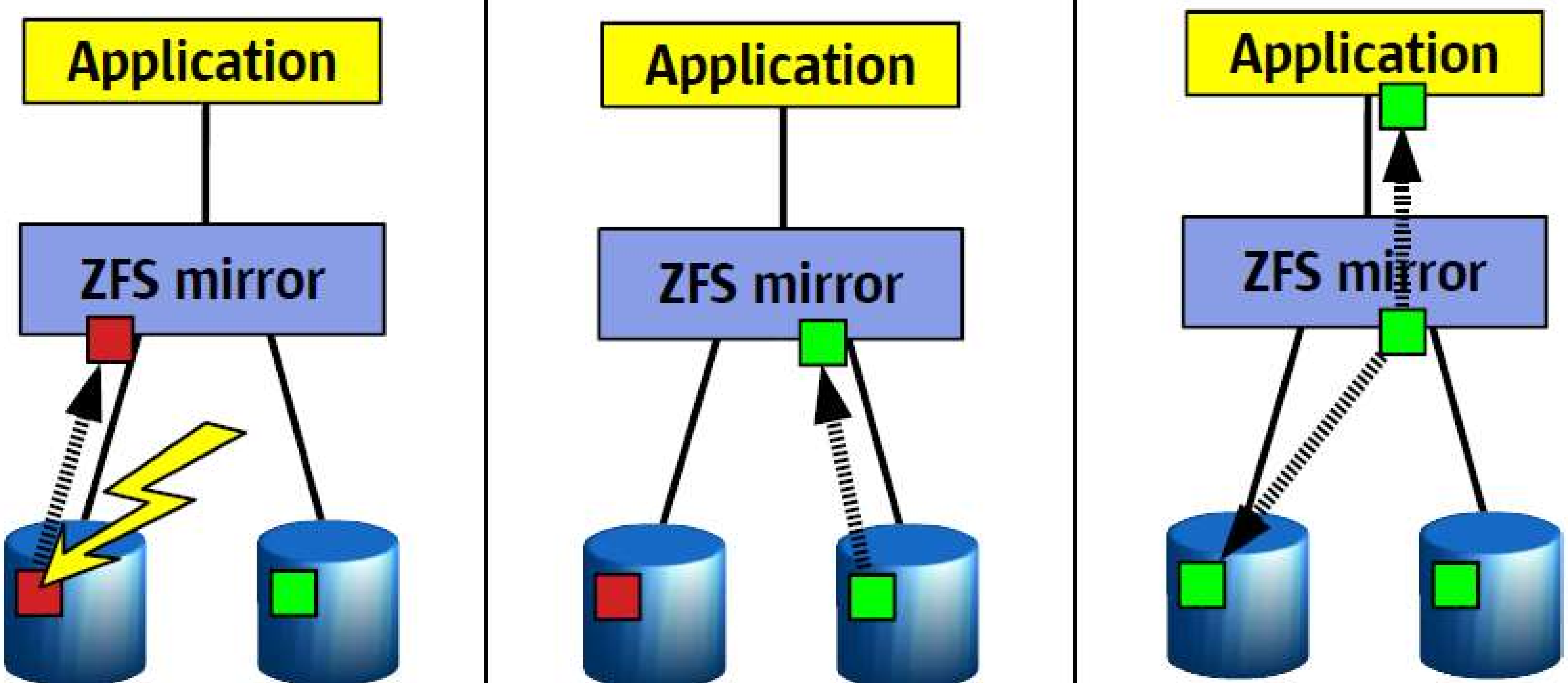


Self-Healing data in ZFS

1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.

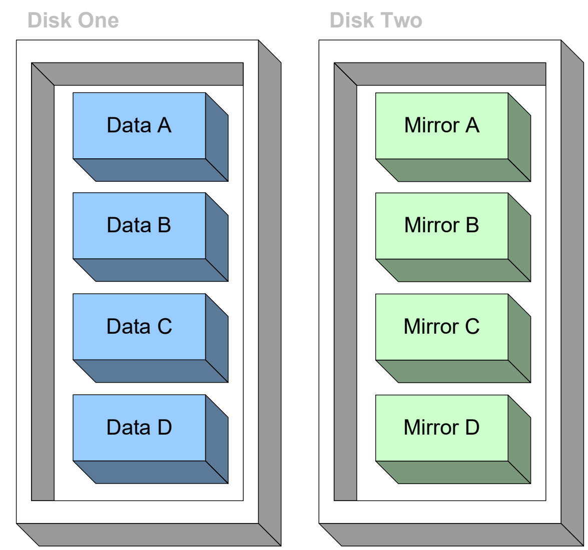
2. ZFS tries the second disk. Checksum indicates that the block is good.

3. ZFS returns good data to the application and repairs the damaged block.



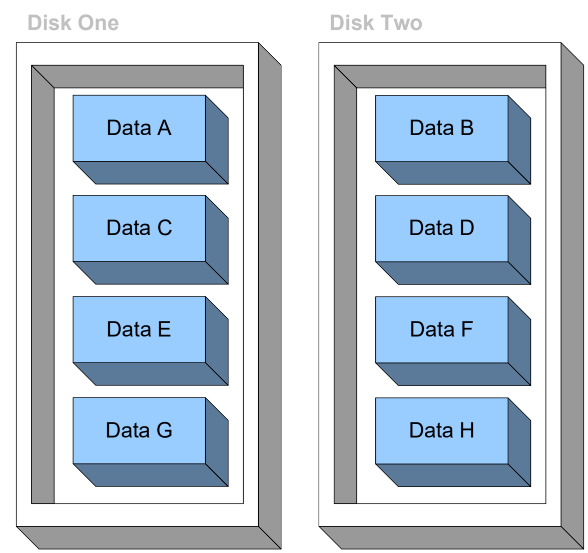
Mirroring

- The easiest way to get high availability
- Half the size
- Higher read performance



Striping

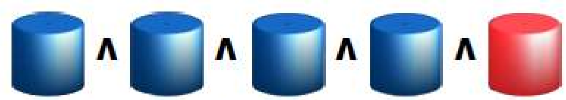
- Higher performance
- Distributed across disks
- Work in parallel



Traditional RAID-4 and RAID-5

- **Several data disks plus one parity disk**

= 0

- **Fatal flaw: partial stripe writes**
  - Parity update requires read-modify-write (slow)
    - Read old data and old parity (two synchronous disk reads)
    - Compute new parity = new data ^ old data ^ old parity
    - Write new data and new parity
  - Suffers from *write hole:* 

= garbage
    - Loss of power between data and parity writes will corrupt data
    - Workaround: $$$ NVRAM in hardware (i.e., don't lose power!)
- **Can't detect or correct silent data corruption**

RAID-Z

RAID-Z

RAID-Z Protection

RAID-5 and More

- ZFS provides better than RAID-5 availability
  - > Copy-on-write approach solves historical problems
- Striping uses dynamic widths
  - > Each logical block is its own stripe
- All writes are full-stripe writes
  - > Eliminates read-modify-write (So it's fast!)
- Eliminates RAID-5 “write hole”
  - > No need for NVRAM

Easier Administration

Disk Scrubbing

- Uses checksums to verify the integrity of all the data
- Traverses metadata to read every copy of every block
- Finds latent errors while they're still correctable
- It's like ECC memory scrubbing – but for disks
- Provides fast and reliable re-silvering of mirrors

Resilvering of mirrors

- Resilvering (AKA resyncing, rebuilding, or reconstructing) is the process of repairing a damaged device using the contents of healthy devices.

- For a mirror, resilvering can be as simple as a whole-disk copy. For RAID-5 it's only slightly more complicated: instead of copying one disk to another, all of the other disks in the RAID-5 stripe must be XORed together.

Resilvering of mirrors

The main advantages of this feature are as follows:

- ZFS only resilvers the minimum amount of necessary data.
- The entire disk can be resilvered in a matter of minutes or seconds,
- Resilvering is interruptible and safe. If the system loses power or is rebooted, the resilvering process resumes exactly where it left off, without any need for manual intervention.
- **Transactional pruning.** If a disk suffers a transient outage, it's not necessary to resilver the entire disk -- only the parts that have changed.
- **Live blocks only.** ZFS doesn't waste time and I/O bandwidth copying free disk blocks because they're not part of the storage pool's block tree.

Resilvering of mirrors

Types of resilvering:

- **Top-down resilvering-**the very first thing ZFS resilvers is the uberblock and the disk labels. Then it resilvers the pool-wide metadata; then each file system's metadata; and so on down the tree.

- **Priority-based resilvering-**Not yet implemented in ZFS.

Create ZFS Pools

- Create a ZFS pool

  # zpool create tank c0d1 c1d0 c1d1

  # zpool list

NAME	SIZE	USED	AVAIL	CAP	HEALTH	ALTROOT
tank	23.8G	91K	23.8G	0%	ONLINE	-

- Destroy a pool

  # zpool destroy tank

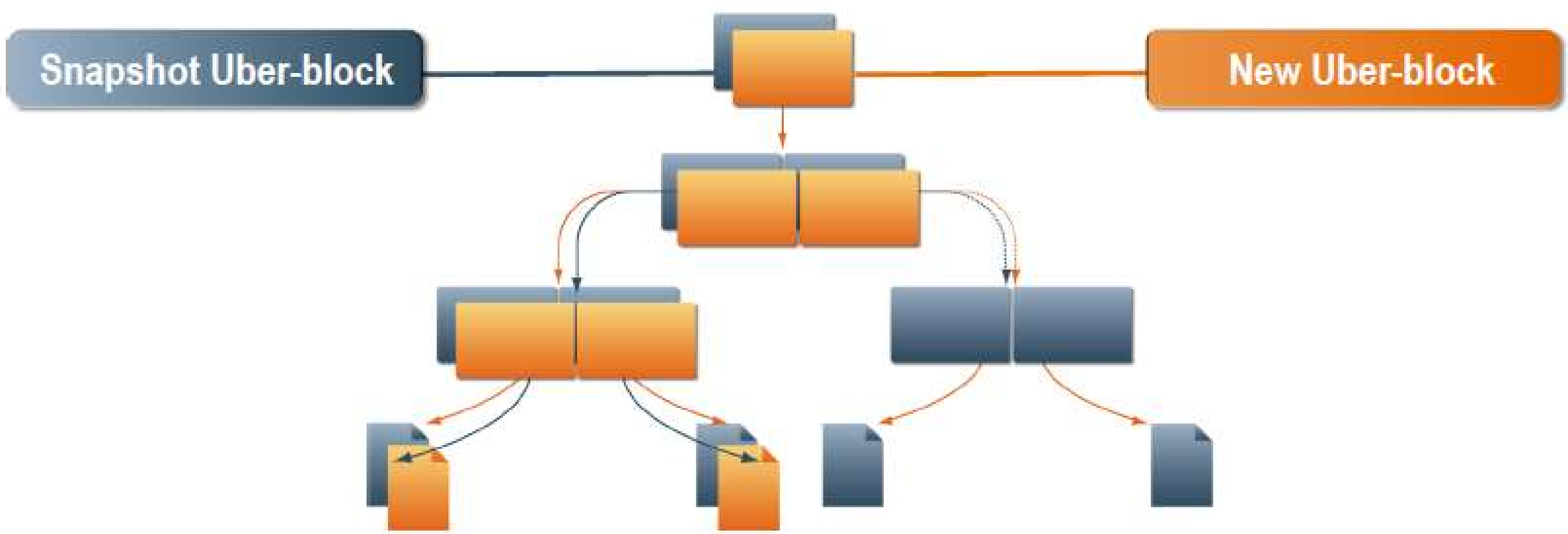
- Create a mirrored pool

  # zpool create mirror c1d0 c1d1

  - *Mirror between disk c1d0 and disk c1d1*
  - *Available storage is the same as if you used only one of these disks*
  - *If disk sizes differ, the smaller size will be your storage size*

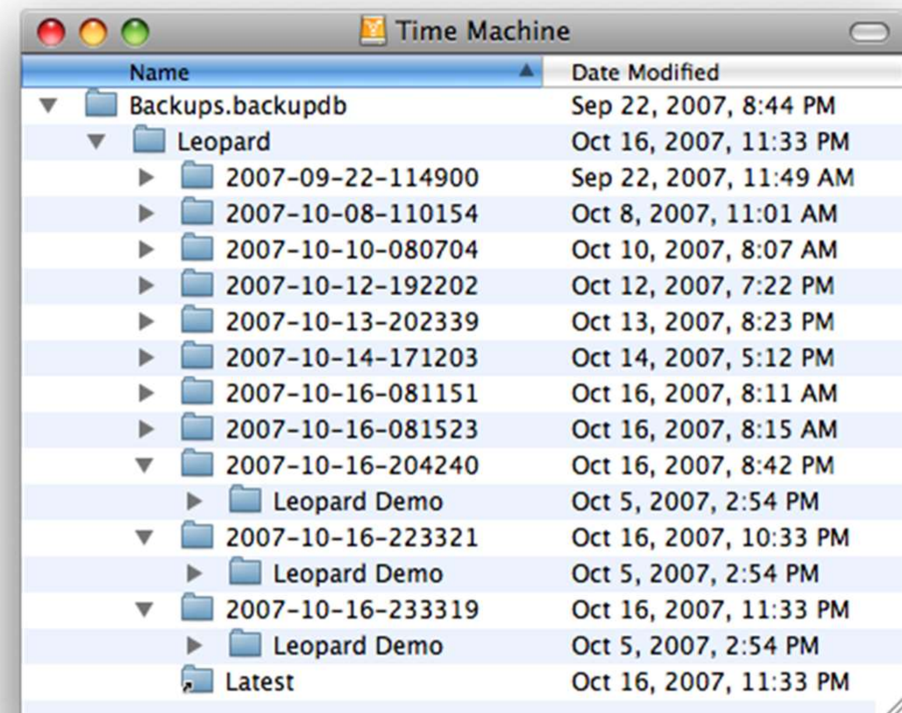
ZFS Snapshots

- View of a file system as it was at a particular point in time.
- A snapshot initially consumes no disk space, but it starts to consume disk space as the files it references get modified or deleted.
- Constant time operation.



ZFS Snapshots

- Independent of the size of the file system that it references to.
- Presence of snapshots doesn't slow down any operation.
- Snapshots allow us to take a full back-up of all files/directories referenced by the snapshot.

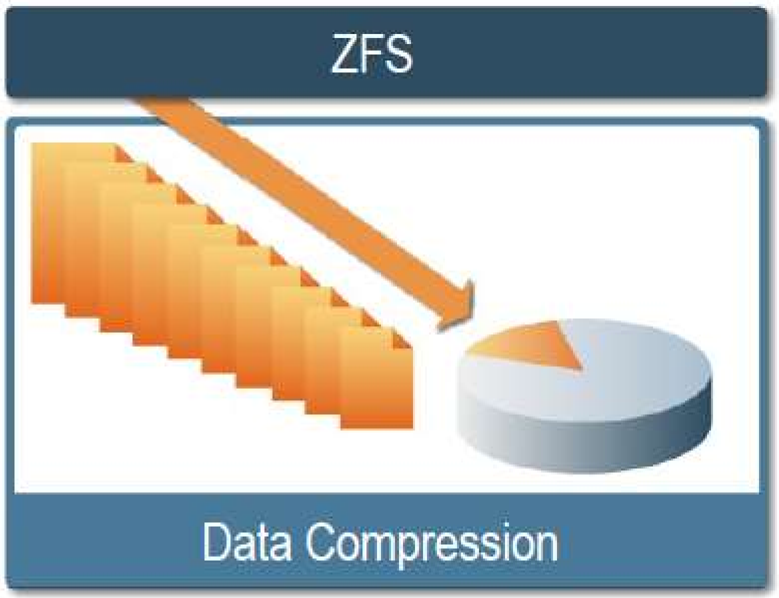


ZFS Clones

- A clone is a writable volume or file system whose initial contents are the same as the dataset from which it was created.
- Constant time operation.
- ZFS clones do not occupy additional disk space when they are created.
- Clones can only be created from a snapshot.
- An implicit dependency is created between the clone and the snapshot.

Data Compression

- Reduces the amount of disk space used
- Reduces the amount of data transferred to disk – increasing data throughput



Unparalleled Scalability

The limitations of ZFS are designed to be so large that they will not be encountered in practice for some time. Some theoretical limitations in ZFS are:

- Number of snapshots of any file system - 2^{64}
- Number of entries in any individual directory - 2^{48}
- Maximum size of a file system - 2^{64} bytes
- Maximum size of a single file - 2^{64} bytes
- Maximum size of any attribute - 2^{64} bytes
- Maximum size of any zpool - 2^{78} bytes
- Number of attributes of a file - 2^{56}
- Number of files in a directory - 2^{56}
- Number of devices in any zpool - 2^{64}
- Number of zpools in a system - 2^{64}
- Number of file systems in a zpool - 2^{64}

Traditional Disk Storage Administration

But with ZFS....

Copy-on-Write Design

Multiple Block Sizes

Pipelined I/O

Dynamic Striping

Architected for Speed

Multiple Block Size

- No single value works well with all types of files
- Large blocks increase bandwidth but reduce metadata and can lead to wasted space
- Small blocks save space for smaller files, but increase I/O operations on larger ones
- FSBs are the basic unit of ZFS datasets, of which checksums are maintained
- Files that are less than the record size are written as a single file system block (FSB) of variable size in multiples of disk sectors (512B)
- Files that are larger than the record size are stored in multiple FSBs equal to record size

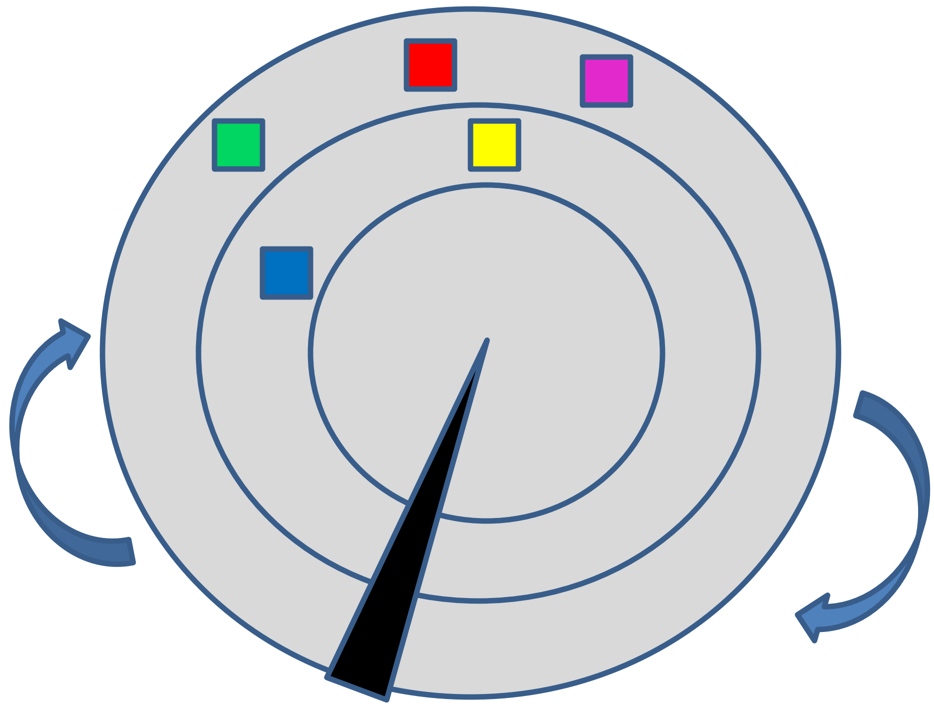
Pipelined I/O

Reorders writes to be as sequential as possible

App #1 writes:

App #2 writes:

If left in original order, we waste a lot of time waiting for head and platter positioning:



Move Head

Spin Head

Move Head

Move Head

Move Head

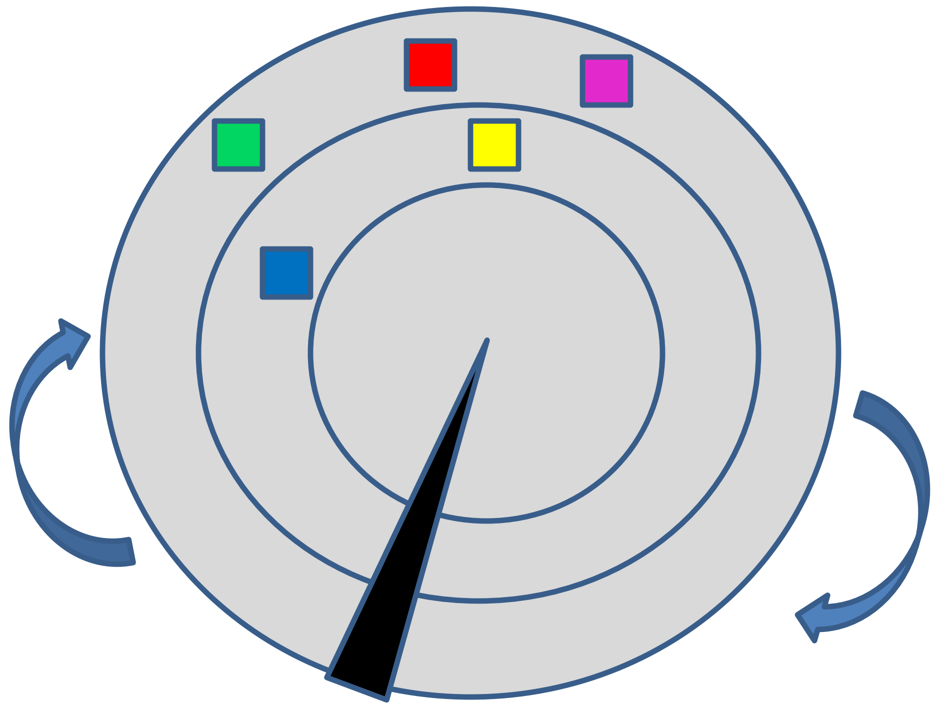
Pipelined I/O

Reorders writes to be as sequential as possible

App #1 writes:

App #2 writes:

Pipelining lets us examine writes as a group and optimize order:



Move Head

Move Head

Dynamic Striping

- Load distribution across devices
- Factors determining block allocation include:
  - Capacity
  - Latency & bandwidth
  - Device health

Dynamic Striping

Writes striped across both mirrors.
Reads occur wherever data was written.

*# zpool create tank *
*mirror c1t0d0 c1t1d0 *
mirror c2t0d0 c2t1d0

New data striped across three mirrors.
No migration of existing data.
Copy-on-write reallocates data over time, gradually spreading it across all three mirrors.

*# zpool add tank *
mirror c3t0d0 c3t1d0

Sun microsystems

Cost and Source Code

ZFS is FREE*

*Free
$ USD0
€ EUR0
£ GBP0
kr SEK0
¥ YEN0
元 YUAN0

opensolaris

- ZFS source code is included in Open Solaris
  - > 47 ZFS patents added to CDDL patent commons

Disadvantages

- ZFS is still not widely used yet.
- RAIDZ2 has a high IO overhead- ZFS is slow when it comes to external USB drives
- Higher power consumption
- No encryption support
- ZFS lacks a bad sector relocation plan.
- High CPU usage

And for the Future

More Flexible

- Pool resize and device removal
- Booting / root file system
- Integration with Solaris Containers

More Secure

- Encryption
- Secure delete — overwriting for “absolute” deletion

More Reliable

- Fault Management Architecture Integration
- Hot spares
- DTrace providers

Thank You!