Operating Systems for OpenSPARC

Karsten Guthridge
Compiler Performance Engineering

OpenSPARC
October 2008

Solaris on UltraSPARC T1

Solaris 10 (and beyond) run on UltraSPARC T1

Run on top of Hypervisor ("sun4v") layer

Fully supported by Sun and OpenSolaris

Linux Ports to date

Sun T1000 support putback to kernel.org

Bulk of support for UltraSPARC/OpenSPARC T1

putback by David Miller, approx Dec 2005

in 2.6.17 Linux kernel

runs on top of Hypervisor

Full Ubuntu distribution (announced ~Spring 2006)

Gentoo Distribution (announced August 2006)

Wind River Linux (announced October 2006)

“carrier-grade” Linux, notably for Telecom applications

*BSD on OpenSPARC T1

FreeBSD port for UltraSPARC T1 announced Nov 2006

Other *BSD ports are underway

OpenSolaris Program

OpenSPARC
October 2008

Agenda

What is OpenSolaris?

Why use OpenSolaris?

Curriculum Development Resources

Selected Features of OpenSolaris OS

Performance and Tracing Tools

Agenda

What is OpenSolaris?

Why use OpenSolaris?

Curriculum Development Resources

Selected Features of OpenSolaris OS

Performance and Tracing Tools

What is OpenSolaris?

Community Participation + Solaris Innovation

OpenSolaris as a Distribution

Solaris Innovation with a 21st Century Release Model

Core OS based on latest open source Solaris development work

Closing the “familiarity gap”

Easier to acquire, easier to install, GNU userland, package management system...

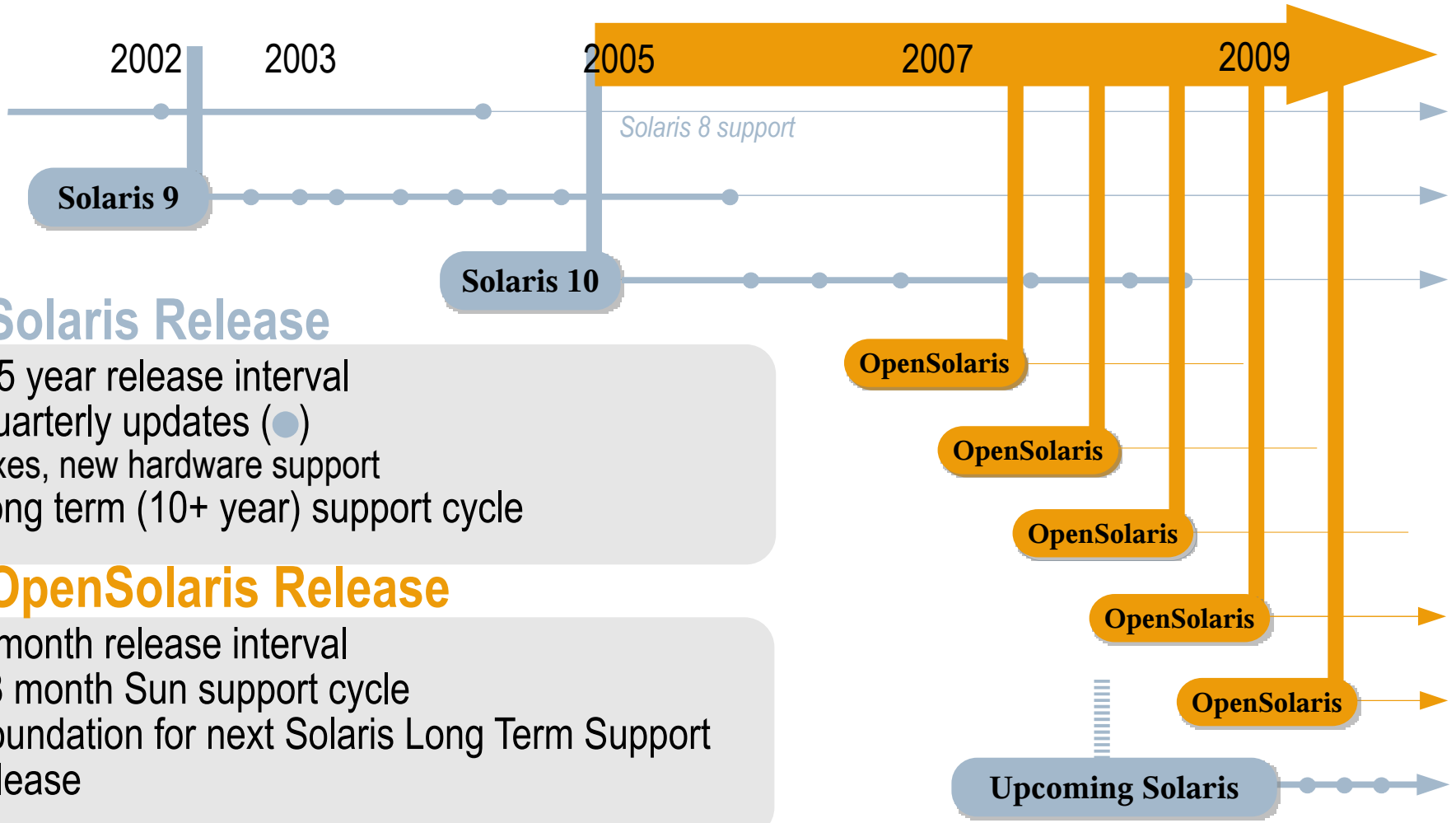
Package repositories delivering Sun and non-Sun innovation

Unique Solaris capabilities

Upgrade rollback via ZFS snapshots, AMP stack with integrated DTrace probes, binary compatibility...

Solaris/OpenSolaris Release Model

20 Years, One Development Base



Solaris Release

3-5 year release interval
Quarterly updates (●)
Fixes, new hardware support
Long term (10+ year) support cycle

OpenSolaris Release

6 month release interval
18 month Sun support cycle
Foundation for next Solaris Long Term Support release

Web Resources for OpenSolaris

Country Portals - http://opensolaris.org/os/project/portals/

Discussion Forums

Communities

Form around interest groups, technologies, support, tools, and user groups, etc.

OpenSolaris User Groups:
http://www.opensolaris.org/os/community/advocacy/usergroups/

Projects

Collaborative efforts with code repositories & committers

OpenGrok™

source code search and cross reference engine

Agenda

What is OpenSolaris?

Why use OpenSolaris?

Curriculum Development Resources

Selected Features of OpenSolaris OS

Performance and Tracing Tools

Why Use OpenSolaris?

Price

$0.00 for infinite right-to-use

Innovative Core Features

Solaris zones, DTrace, New IP Stack, ZFS...

Backward Compatibility

Software built for Solaris N will run correctly on Solaris N+1 and subsequent versions

Hardware Platform Neutrality

Same feature sets & API for SPARC & x86

Development Tools

Sun Studio suite, gcc, gdb, mdb...

Agenda

What is OpenSolaris?

Why use OpenSolaris?

Curriculum Development Resources

OpenSolaris Kernel Features & Architecture

Core Features of OpenSolaris OS

Performance and Tracing Tools

Curriculum Development Resources

Curriculum Development Guide

Enable CS educators to incorporate OpenSolaris technology into a CS curriculum

http://www.opensolaris.org/os/community/edu/curriculum_development/

Curriculum “Plugins Preparation”

Presentations on various topics

Curriculum “Plugins”

Specific aspects of OpenSolaris which may be “plugged into” an existing curriculum

Solaris features, architecture, processes, threads, scheduling, memory management, file systems, device management, etc.

Workshop on Service Management Facility (SMF)

Academic & Research Community

An OpenSolaris community for students, faculty & researchers

http://www.opensolaris.org/os/community/edu/

No Cost Resources

http://www.opensolaris.org/os/community/edu/nocost_resources/

Free training

Free software & development tools

etc.

Agenda

What is OpenSolaris?

Why use OpenSolaris?

Curriculum Development Resources

Selected Features of OpenSolaris OS

Performance and Tracing Tools

Selected Features of OpenSolaris

DTrace – Dynamic Tracing utility

Solaris Zones – OS-level virtualization

Solaris ZFS – Zetabyte File System

Innovate in Real Time with DTrace

- Enables dynamic modification of the system to record arbitrary data
- Promotes tracing of live systems
- Is completely safe – its use cannot induce fatal failure
- Allows tracing of both the kernel and user-level programs

Why DTrace?

DTrace has the following capabilities:

Can enable tens of thousands or tracing points called probes

When a probe fires, it can record any arbitrary kernel (or userland data).

The arbitrary data that is recorded using DTrace could include:

Any input argument to a function

Any global variable

A nanosecond timestamp

A stack trace

DTrace Abilities

DTrace facilitates:

Examining the entire software stack (user to kernel)

Determining the root cause of performance problems

Tracking down the source of aberrant behavior

DTrace Architecture

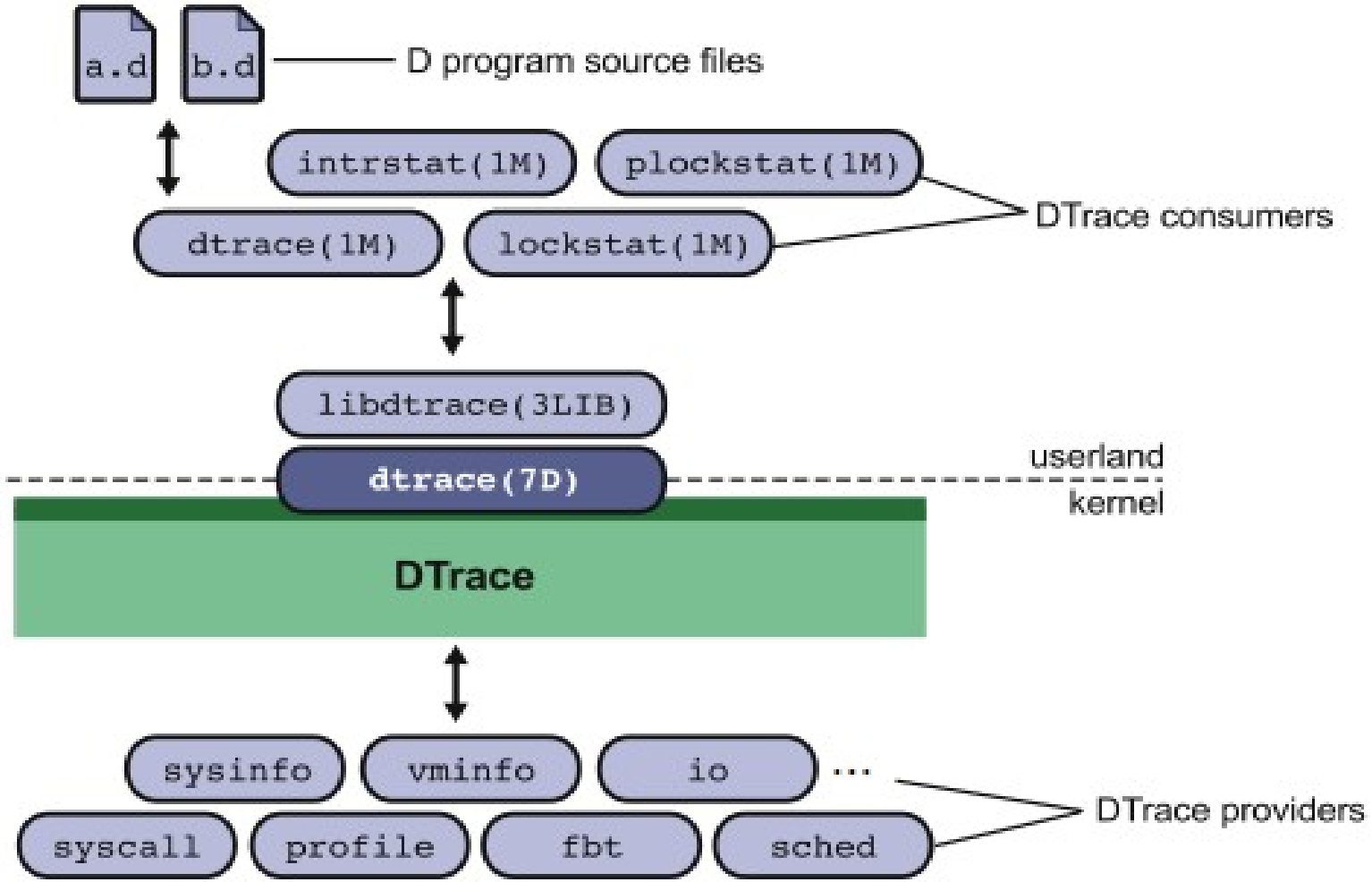
Probes are sensors placed at the points of interest in the kernel.

Providers implement and enable user-defined probes

Actions can be defined to record arbitrary program data when a probe fires.

The new D programming language is used to specify probes and related actions.

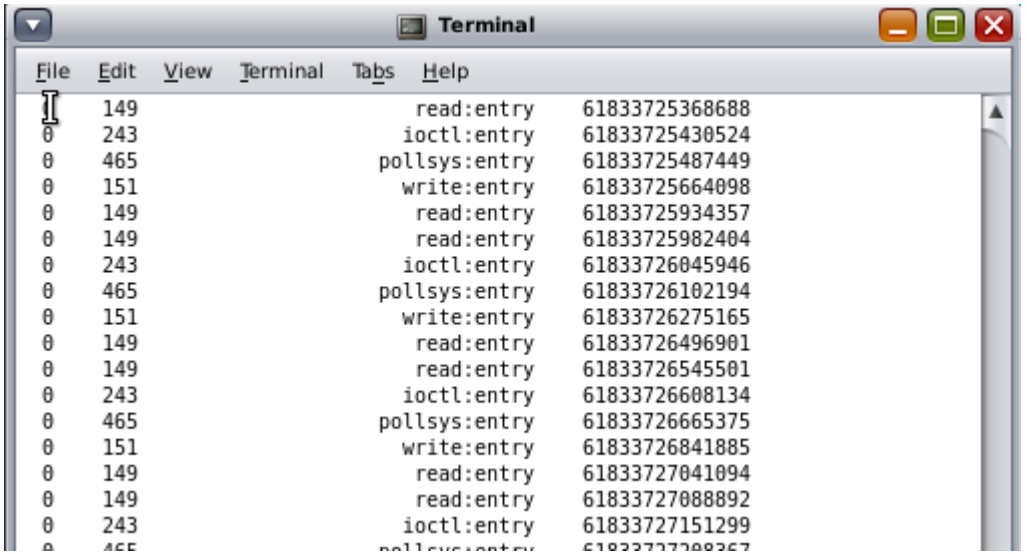
DTrace Architecture & Components



DTrace One Liners

```
# dtrace -n 'syscall:::entry {trace(timestamp)}'
```

The above example traces the time of entry to each system call.



DTrace Toolkit

The DTrace toolkit is a collection of useful DTrace scripts

http://www.opensolaris.org/os/community/dtrace/dtracetoolkit/

The toolkit contains:

Scripts

Man pages

Example documentation

Note files

Tutorials

Virtualization Technology

Trend to flexibility

Trend to isolation

Hard Partitions

Virtual Machine

OS Virtualization

Resource Management

Multiple OSes

Single OS

Dynamic System Domains

Xen
Logical Domains
VMware

Solaris Containers (Zones + SRM)
Solaris Containers for Linux Applications
Solaris Trusted Extensions

Solaris Resource Manager (SRM)

Solaris Zones

Virtualize OS layer: file system, devices, network

Secure boundary around virtualized instance

Provides:

Privacy: can't see outside zone

Security: can't affect activity outside zone

Failure isolation: application failure in one zone doesn't affect others

Minimal (if any) performance overhead

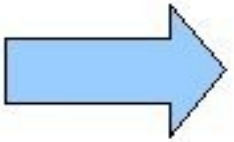
Resource controls provided by Solaris RM

Application/Service Consolidation

E-mail Application

Webserver1

Webserver2



E-mail Application
Webserver1
Webserver2

Solaris Zone: Security

- Root can't be trusted
  - Most operations requiring root disabled
  - Exceptions: file operations, set[ug]id, other “local” operations
- Processes within zone only see/control other processes within zone
- May want to allow specific additional privileges
  - zone in separate processor set can call priocntl

Solaris Zone: File Systems

Each zone allocated part of file system hierarchy

One zone can't see another zone's data

Loopback mounts allow sharing of read-only data (e.g., /usr)

Can't escape (unlike chroot)

Sparse Root Model vs. Whole Root Model

Zones References

http://opensolaris.org/os/community/zones/

http://forum.java.sun.com/forum.jspa?forumID=846

http://www.sun.com/bigadmin/features/articles/backup_zones.jsp

http://www.sun.com/bigadmin/content/submitted/zone_config_steps.jsp

http://www.sun.com/software/solaris/howtoguides/containersLowRes.jsp

ZFS Overview

Pooled storage

Completely eliminates the antique notion of volumes

Does for storage what VM did for memory

Provable end-to-end data integrity

Detects and corrects silent data corruption

Historically considered “too expensive”

Transactional design

Always consistent on disk

Removes most constraints on I/O order – huge performance wins

Simple administration

Concisely express your intent

Why Volumes Exist

In the beginning, each file system managed a single disk.

Customers wanted more space, bandwidth, reliability

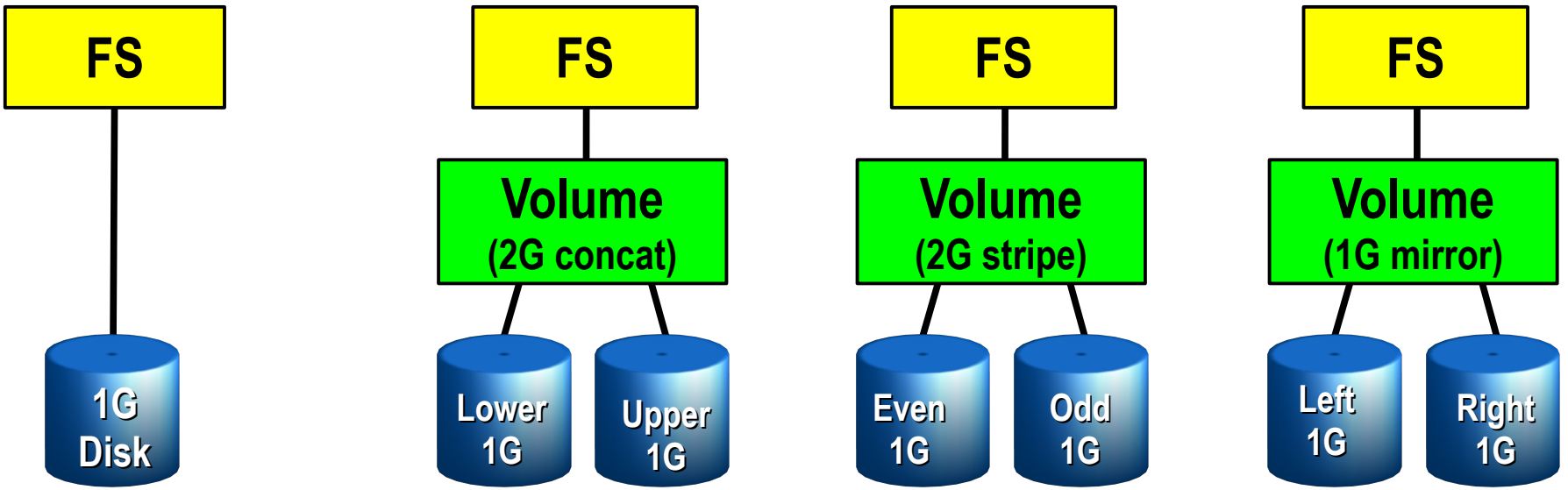
Hard: redesign file systems to solve these problems well

Easy: insert a little shim ("volume") to cobble disks together

An industry grew up around the FS/volume model

File systems, volume managers sold as separate products

Inherent problems in FS/volume interface can't be fixed



FS Volume Model vs. ZFS

Traditional Volumes

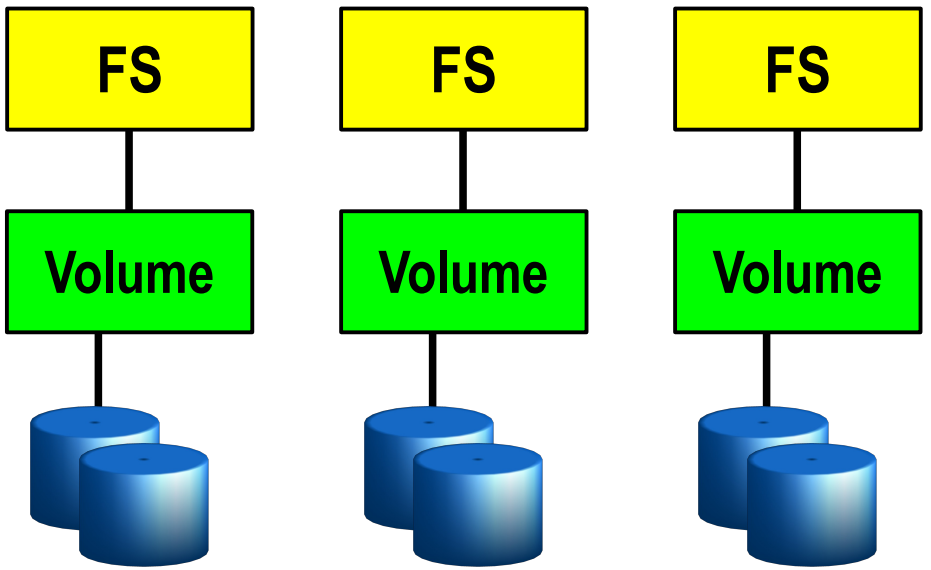
Abstraction: virtual disk

Partition/volume for each FS

Grow/shrink by hand

Each FS has limited bandwidth

Storage is fragmented, stranded



ZFS Pooled Storage

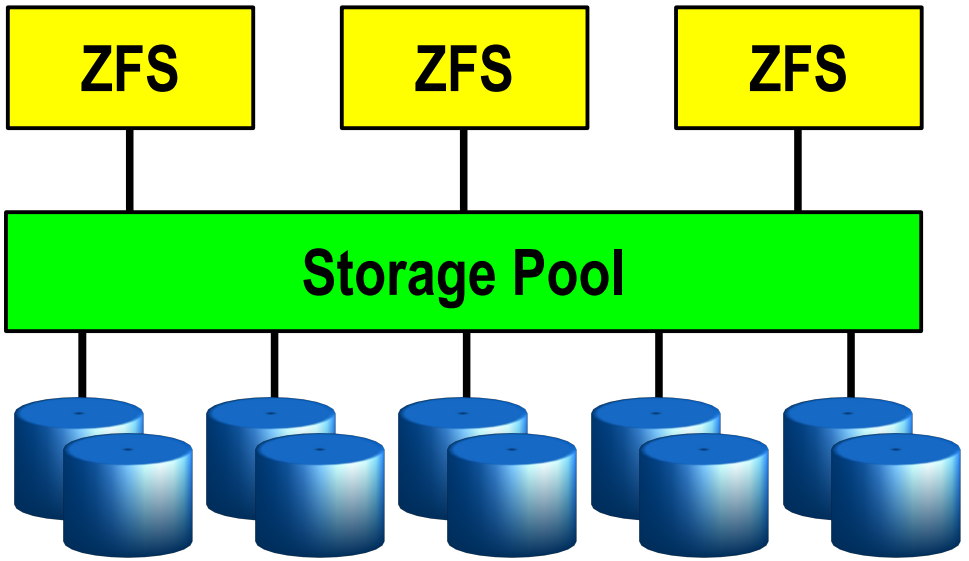
Abstraction: malloc/free

No partitions to manage

Grow/shrink automatically

All bandwidth always available

All storage in the pool is shared



FS Volume Model vs. ZFS

FS/Volume I/O Stack

Block Device Interface

"Write this block, then that block, ..."

Loss of power = loss of on-disk consistency

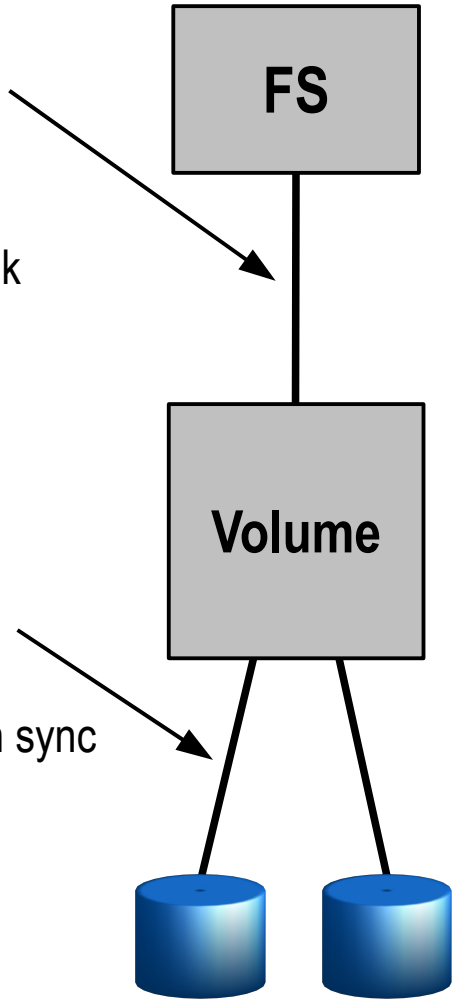
Workaround: journaling, which is slow & complex

Block Device Interface

Write each block to each disk immediately to keep mirrors in sync

Loss of power = resync

Synchronous and slow



ZFS I/O Stack

Object-Based Transactions

"Make these 7 changes to these 3 objects"

All-or-nothing

Transaction Group Commit

Again, all-or-nothing

Always consistent on disk

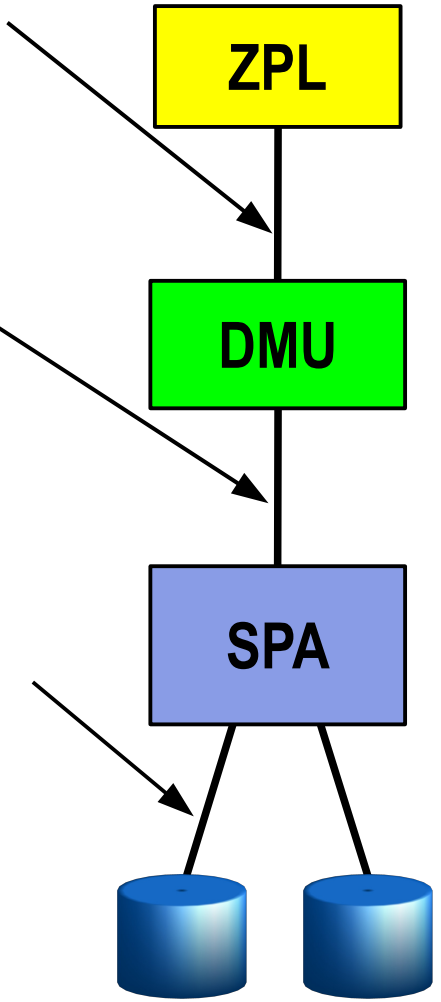
No journal – not needed

Transaction Group Batch I/O

Schedule, aggregate, and issue I/O at will

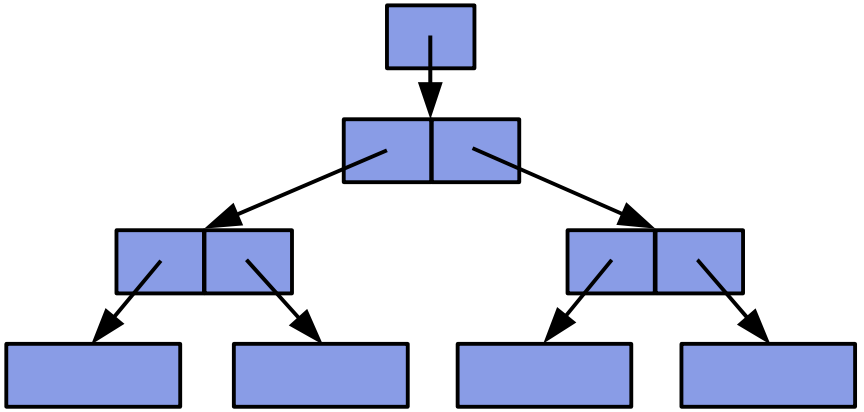
No resync if power lost

Runs at platter speed

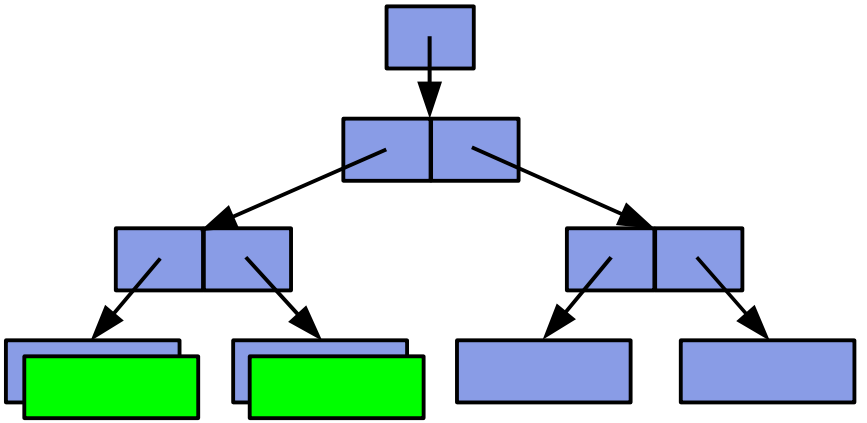


Copy-on-write Transactions

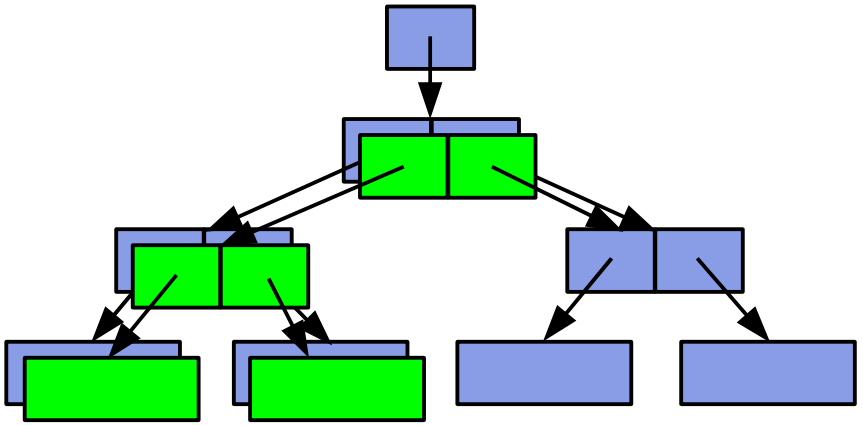
1. Initial block tree



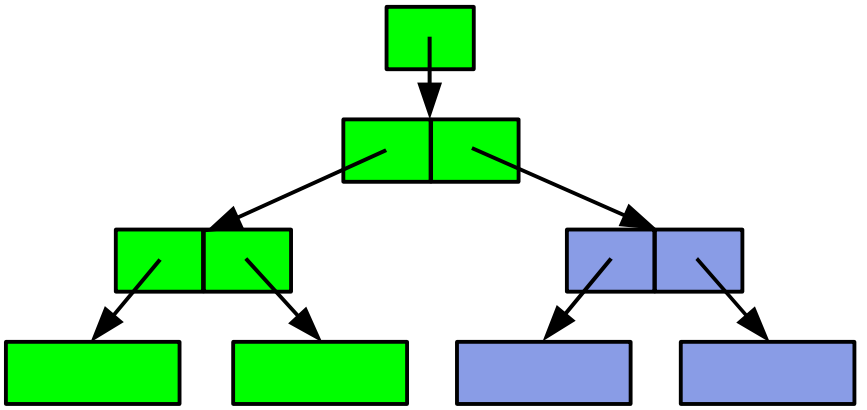
2. COW some blocks



3. COW indirect blocks



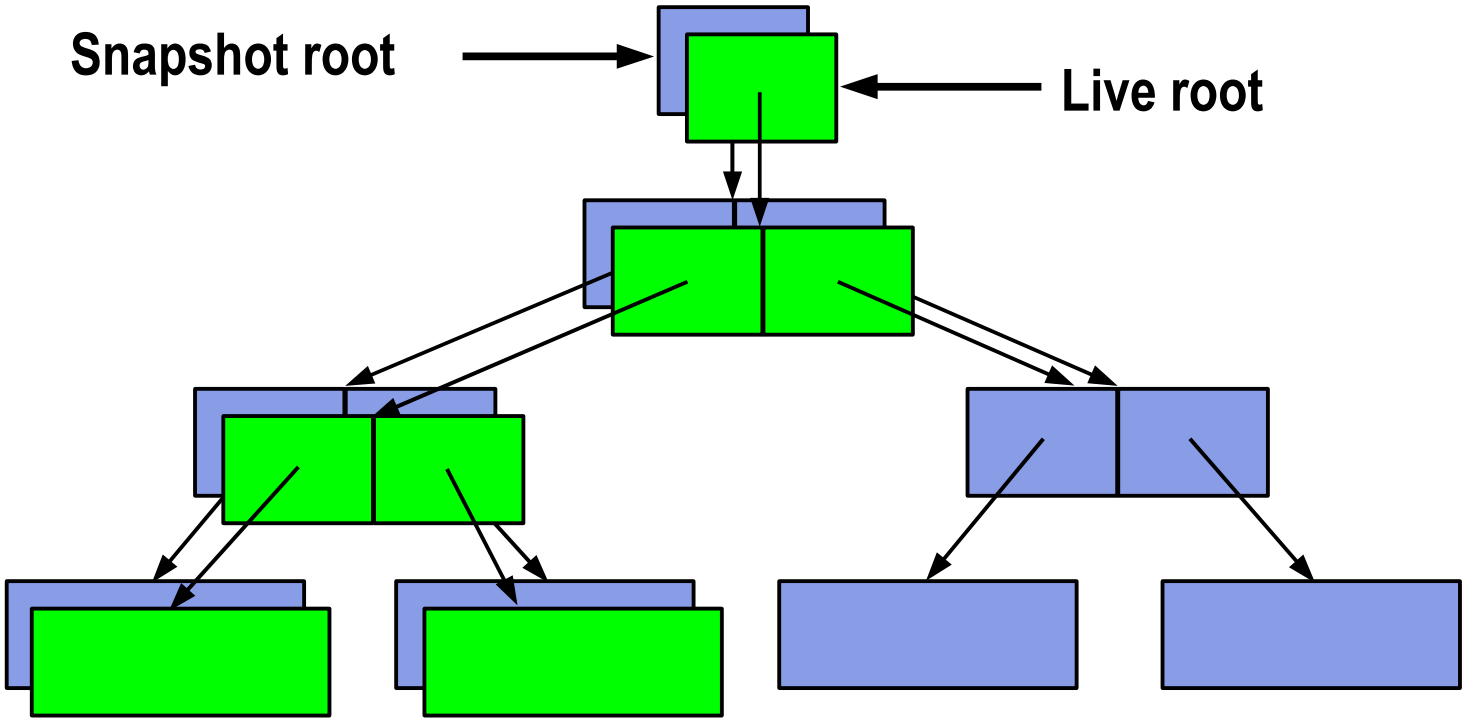
4. Rewrite uberblock (atomic)



Constant-time Snapshots

At end of TX group, don't free COWed blocks

Actually cheaper to take snapshots than not!



End-to-End Data Integrity

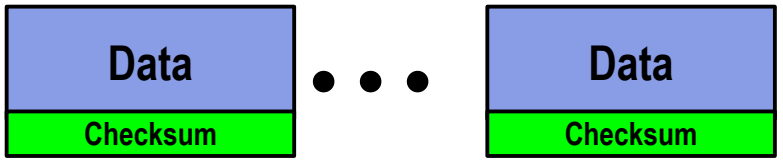
Disk Block Checksums

Checksum stored with data block

Any self-consistent block will pass

Can't even detect stray writes

Inherent FS/volume interface limitation



Disk checksum only validates media

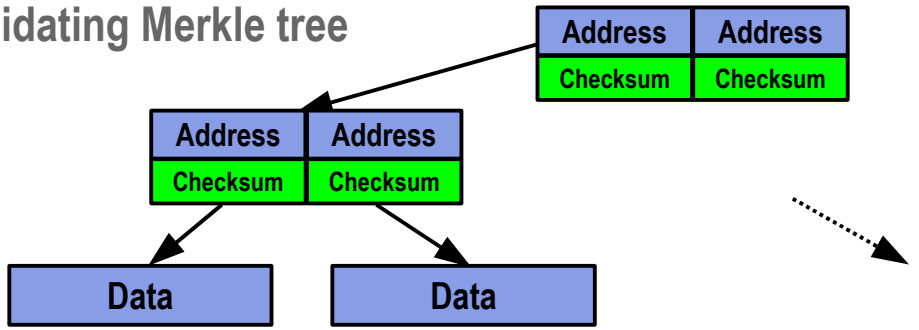
Bit rot
Phantom writes
Misdirected reads and writes
DMA parity errors
Driver bugs
Accidental overwrite

ZFS Data Authentication

Checksum stored in parent block pointer

Fault isolation between data and checksum

Entire storage pool is a self-validating Merkle tree

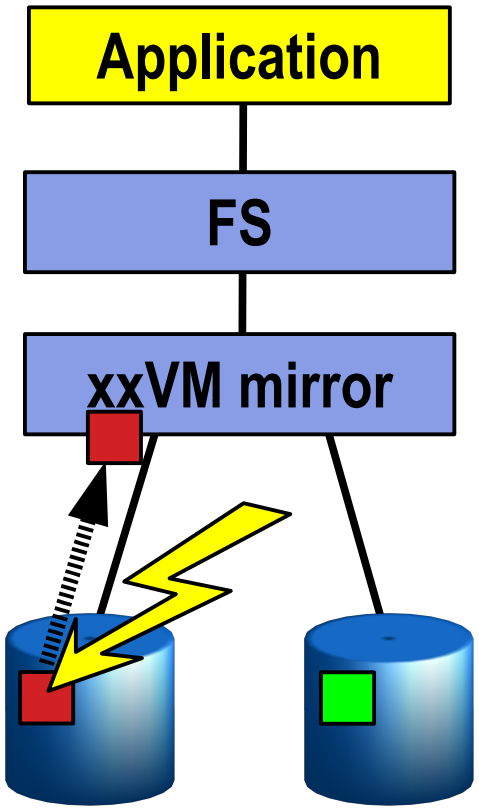


ZFS validates the entire I/O path

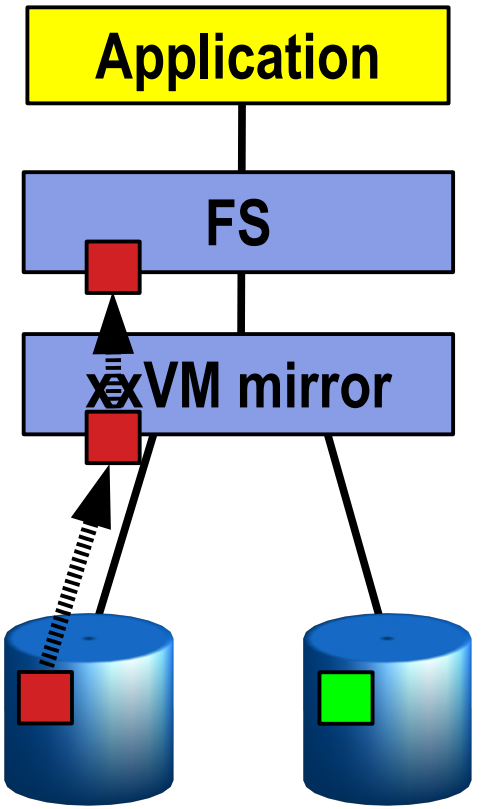
Bit rot
Phantom writes
Misdirected reads and writes
DMA parity errors
Driver bugs
Accidental overwrite

Traditional Mirroring

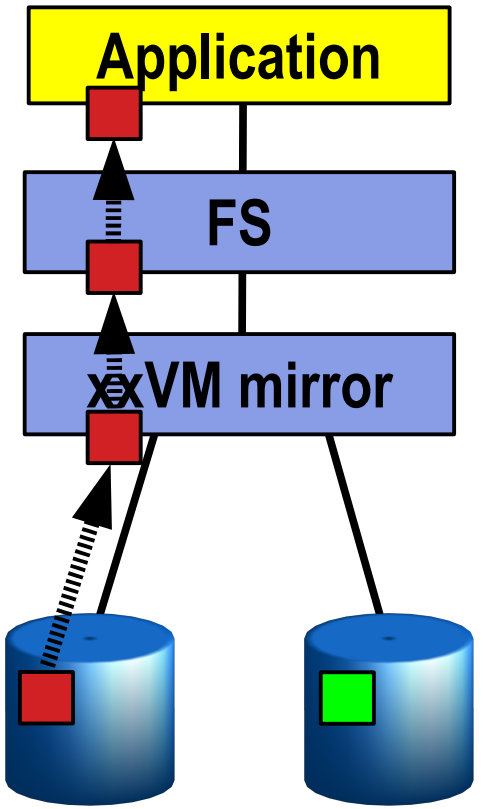
1. Application issues a read. Mirror reads the first disk, which has a corrupt block. It can't tell.



2. Volume manager passes bad block up to file system. If it's a metadata block, the file system panics. If not...

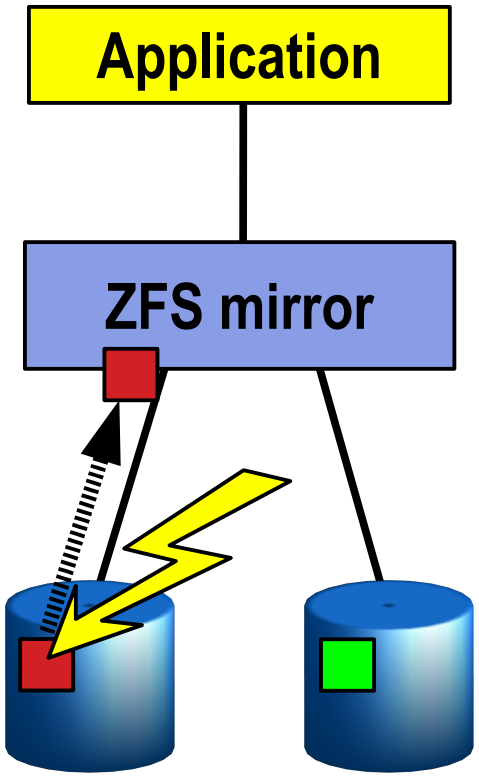


3. File system returns bad data to the application.

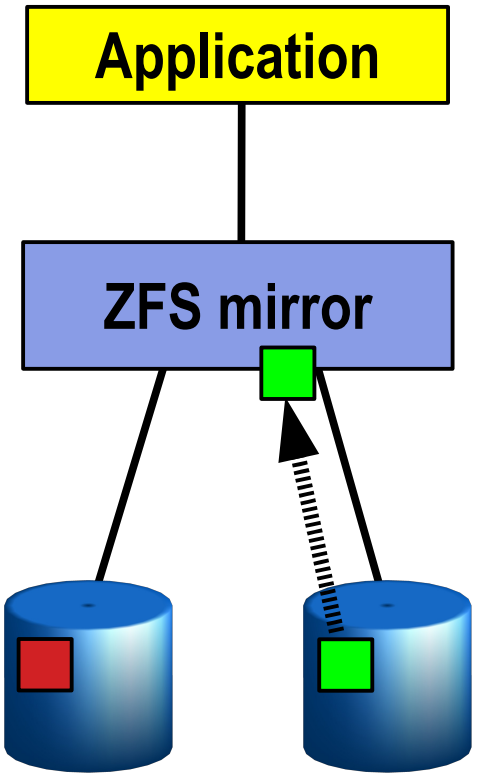


Self-Healing Data in ZFS

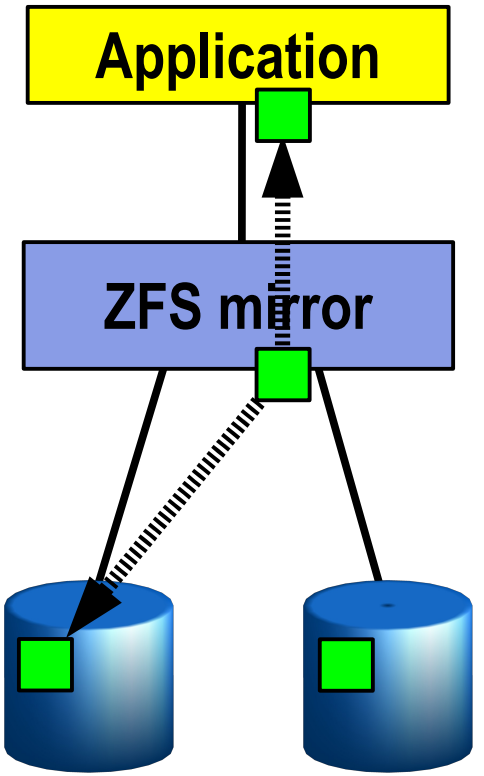
1. Application issues a read. ZFS mirror tries the first disk. Checksum reveals that the block is corrupt on disk.



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.



ZFS Summary

End the Suffering ● Free Your Mind

Simple

Concisely expresses the user's intent

Powerful

Pooled storage, snapshots, clones, compression, scrubbing, RAID-Z

Safe

Detects and corrects silent data corruption

Fast

Dynamic striping, intelligent prefetch, pipelined I/O

Open

http://www.opensolaris.org/os/community/zfs

Free

ZFS References

http://docs.sun.com/app/docs/doc/819-5461

http://blogs.sun.com/bonwick/category/zfs

http://www.opensolaris.org/os/community/zfs

http://www.opensolaris.org/os/community/zfs/docs/zfsadmin.pdf

Agenda

What is OpenSolaris?

Why use OpenSolaris?

Curriculum Development Resources

Core Features of OpenSolaris OS

Performance and Tracing Tools

Performance and Tracing Tools

Process Stats

cputrack: per-processor hw counter
pargs: process arguments
pflags: process flags
pcred: process credentials
pldd: process' library dependencies
psig: process signal disposition
pstack: process stack dump
pmap: process memory map
pfiles: open files and names
prstat: process statistics
ptree: process tree
ptime: process microstate times
pwdx: process working directory

Process Control

pgrep: grep for processes
pkill: kill process list
pstop: stop processes
prun: start processes
prctl: view/set process resources
pwait: wait for a process
preap: reap a zombie process

Performance and Tracing Tools (Cont'd)

Process tracing/debugging

abitrace: trace ABI interface

dtrace: trace the “world”

mdb: debug/control processes

truss: trace functions and system calls

Kernel tracing/debugging

dtrace: trace and monitor kernel

lockstat: monitor locking statistics

lockstat -k: profile kernel

mdb: debug live kernel cores

Performance and Tracing Tools (Cont'd)

System stats

acctcom: process accounting

busstat: bus hardware counters

cpustat: CPU hardware counters

iostat: I/O & NFS statistics

kstat: display kernel statistics

mpstat: processor statistics

netstat: network statistics

nfsstat: NFS server stats

sar: system activity reporting utility

vmstat: virtual memory stats