



Lustre, ZFS, and Data Integrity

Lustre ZFS Team
Sun Microsystems

Andreas Dilger
Sr. Staff Engineer
Sun Microsystems

Topics

Overview

ZFS/DMU features

ZFS Data Integrity

Lustre Data Integrity

ZFS-Lustre Integration

Current Status

2012 Lustre Requirements

Filesystem Limits

- 240 PB file system size
- **1 trillion files** (10^{12}) per file system
- 1.5 TB/s aggregate bandwidth
- 100k clients

Single File/Directory Limits

- **10 billion** files in a single directory
- 0 to 1 PB file size range

Reliability

- **100h** filesystem integrity check
- End-to-end data integrity

ZFS Meets Future Requirements

Capacity

- Single filesystems 512TB+ (theoretical 2^{64} devices * 2^{64} bytes)
- Trillions of files in a single file system (theoretical 2^{48} files per fileset)
- Dynamic addition of capacity/performance

Reliability and Integrity

- Transaction based, copy-on-write
- Internal data redundancy (up to triple parity/mirror and/or 3 copies)
- End-to-end checksum of all data/metadata
- Online integrity verification and reconstruction

Functionality

- Snapshots, filesets, compression, encryption
- Online incremental backup/restore
- Hybrid storage pools (HDD + SSD)
- Portable code base (BSD, Solaris, ... Linux, OS/X?)

Lustre/ZFS Project Background

ZFS-FUSE (2006)

- The first project of ZFS porting to the FUSE framework for the Linux in the user space

Kernel DMU (2008)

- LLNL engineer ported core parts of ZFS to RedHat Linux as a kernel module

Lustre-kDMU (2008 – present)

- Modify Lustre to work on ZFS/DMU storage backend for a production environment

ZFS Overview

ZFS POSIX Layer (ZPL)

- Interface to user applications
- Directories, namespace, ACLs, xattr

Data Management Unit (DMU)

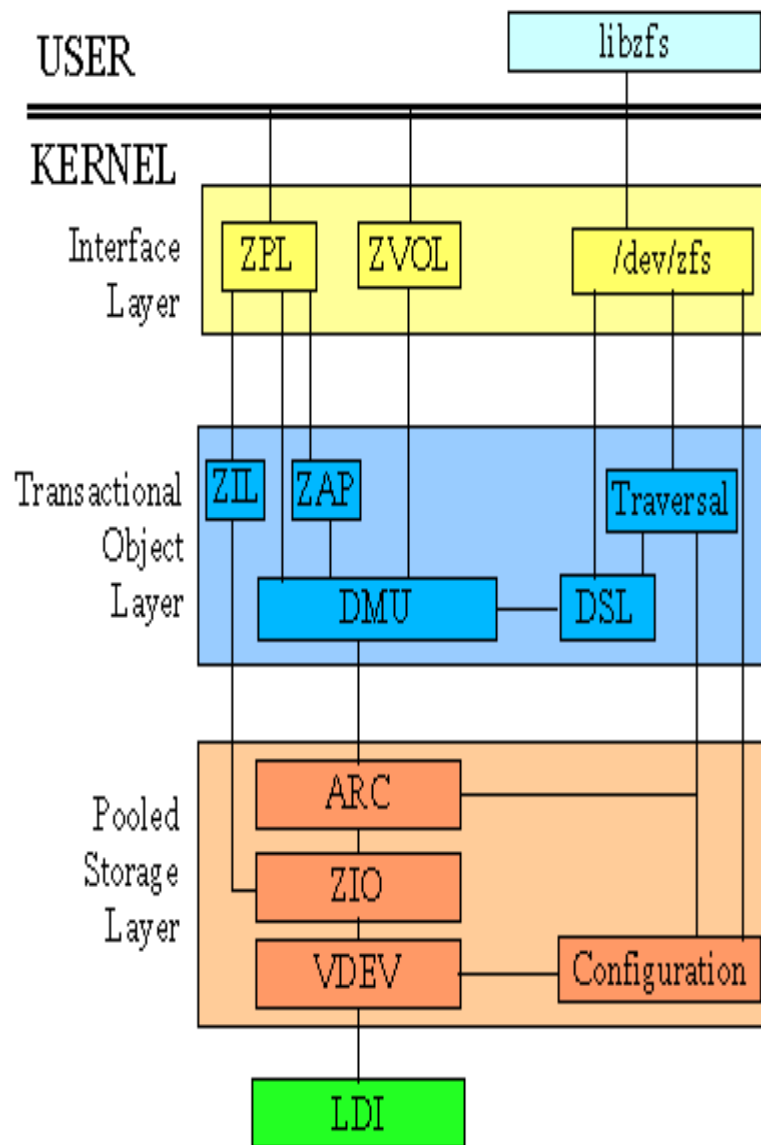
- Objects, datasets, snapshots
- Copy-on-write atomic transactions
- name->value lookup tables (ZAP)

Adaptive Replacement Cache (ARC)

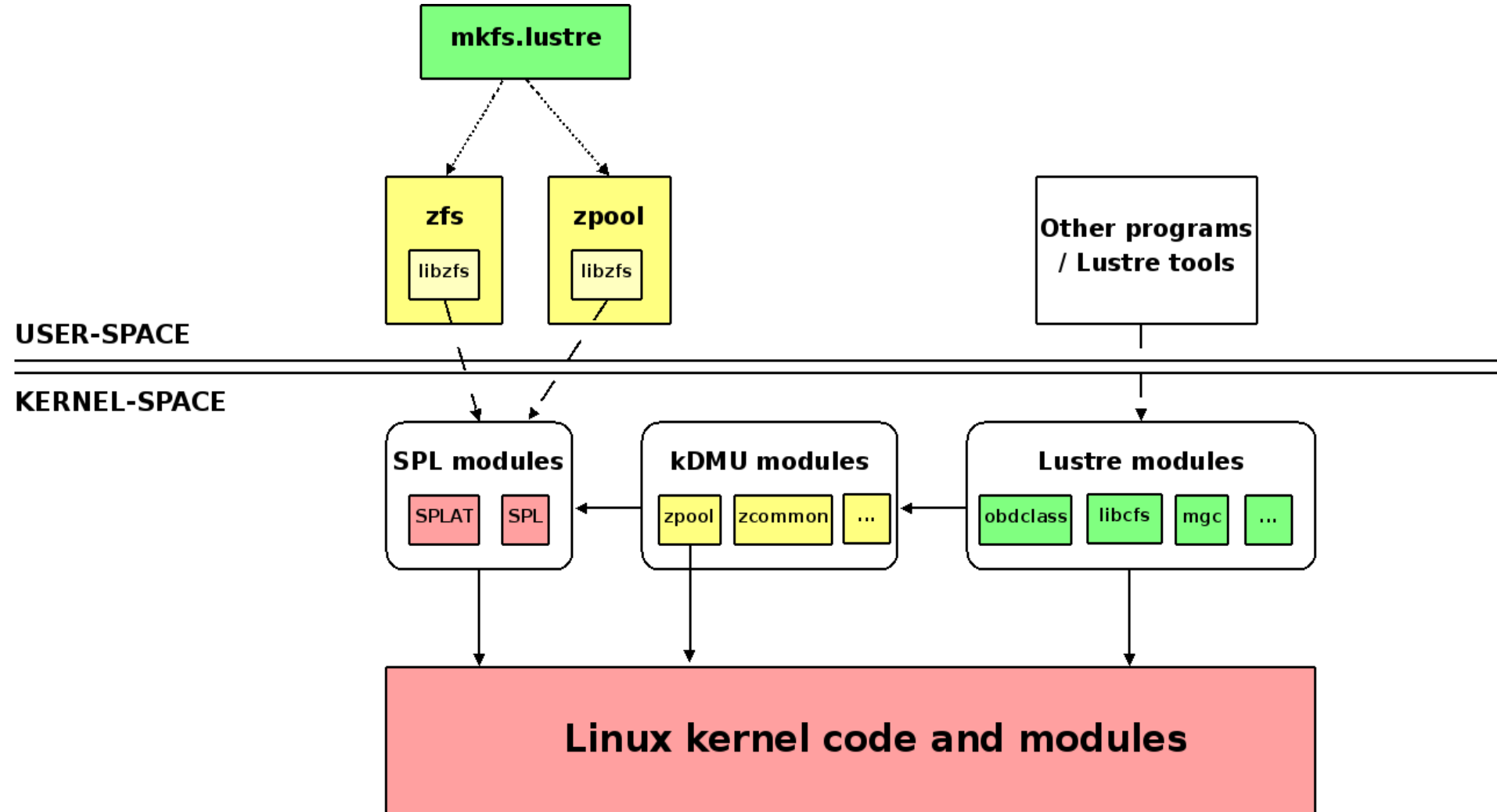
- Manage RAM, flash cache space

Storage Pool Allocator (SPA)

- Allocate blocks at IO time
- Manages multiple disks directly
- Mirroring, parity, compression



Lustre/Kernel DMU Modules



Lustre Software Stack Updates

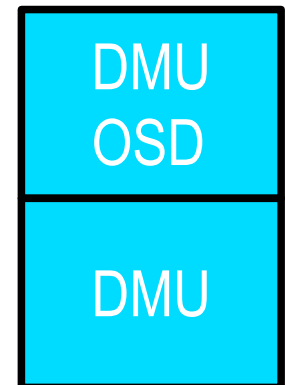
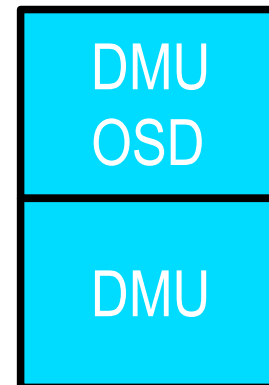
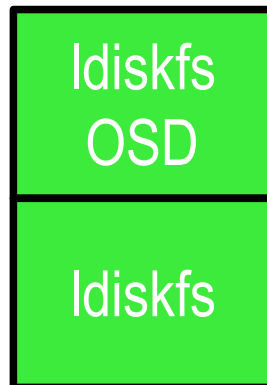
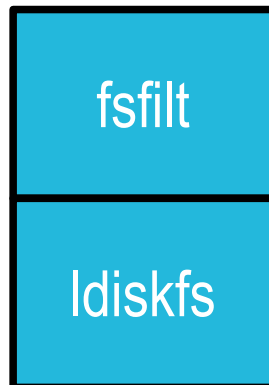
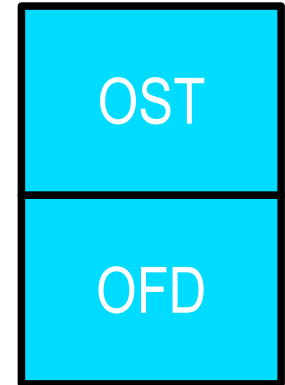
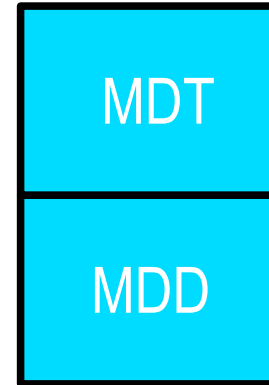
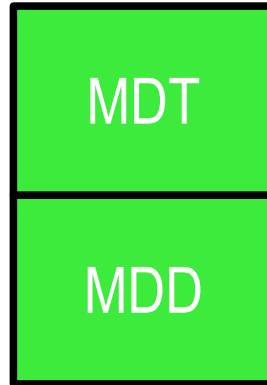
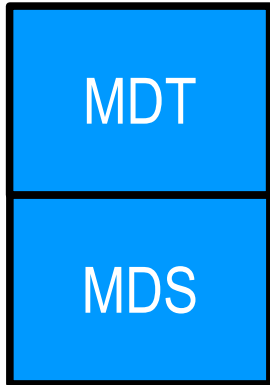
1.8 MDS

1.8/2.0 OSS

2.0 MDS

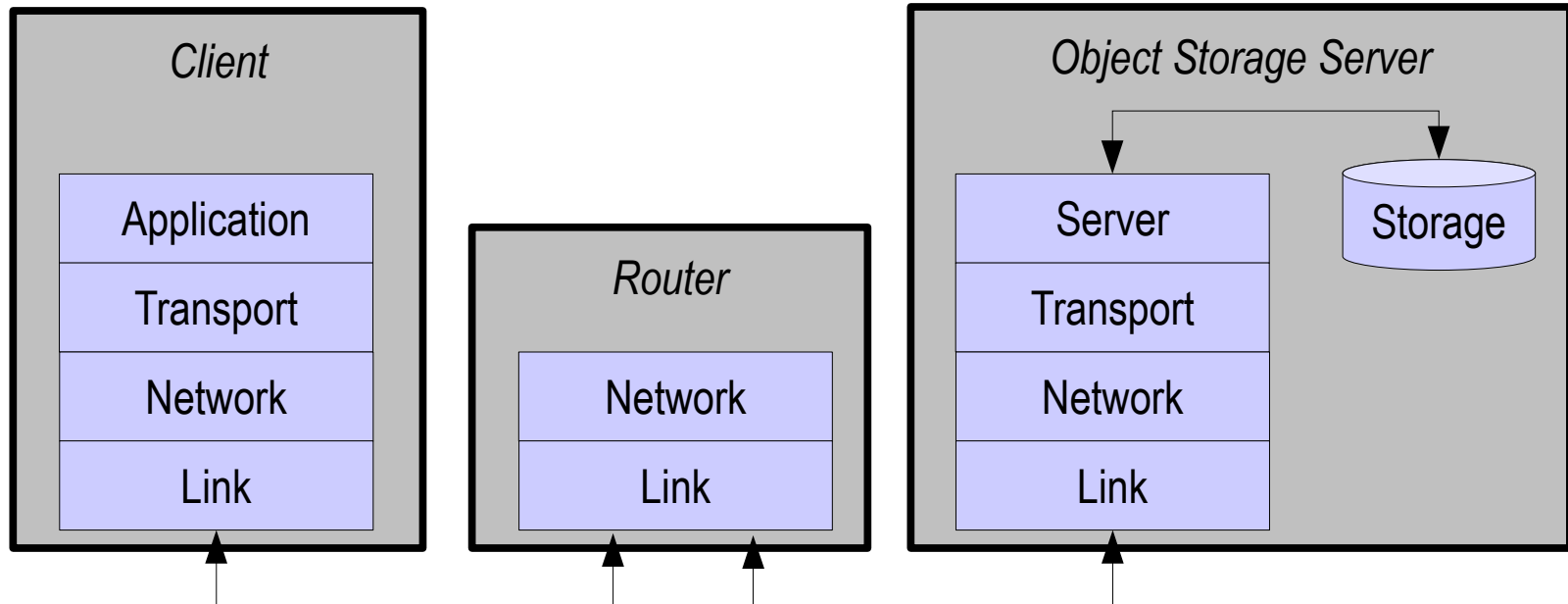
2.1 MDS

2.1 OSS



How Safe Is Your Data?

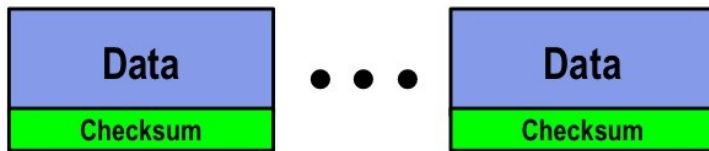
- There are many sources of data corruption
 - Software/firmware: client, NIC, router, server, HBA, disk
 - RAM, CPU, disk cache, media
 - Client network, storage network, cables



ZFS Industry Leading Data Integrity

Disk Block Checksums

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

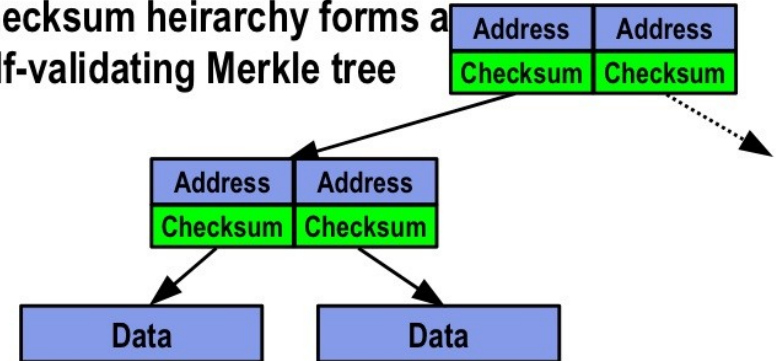


Disk checksum only validates media

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

Data Authentication

- Checksum stored separate from data
- Fault isolation between data and checksum
- Checksum heirarchy forms a self-validating Merkle tree



Checksum tree validates the entire I/O path

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

End-to-End Data Integrity

Lustre Network Checksum

- Detects data corruption over network
- Ext3/4 does not checksum data on disk

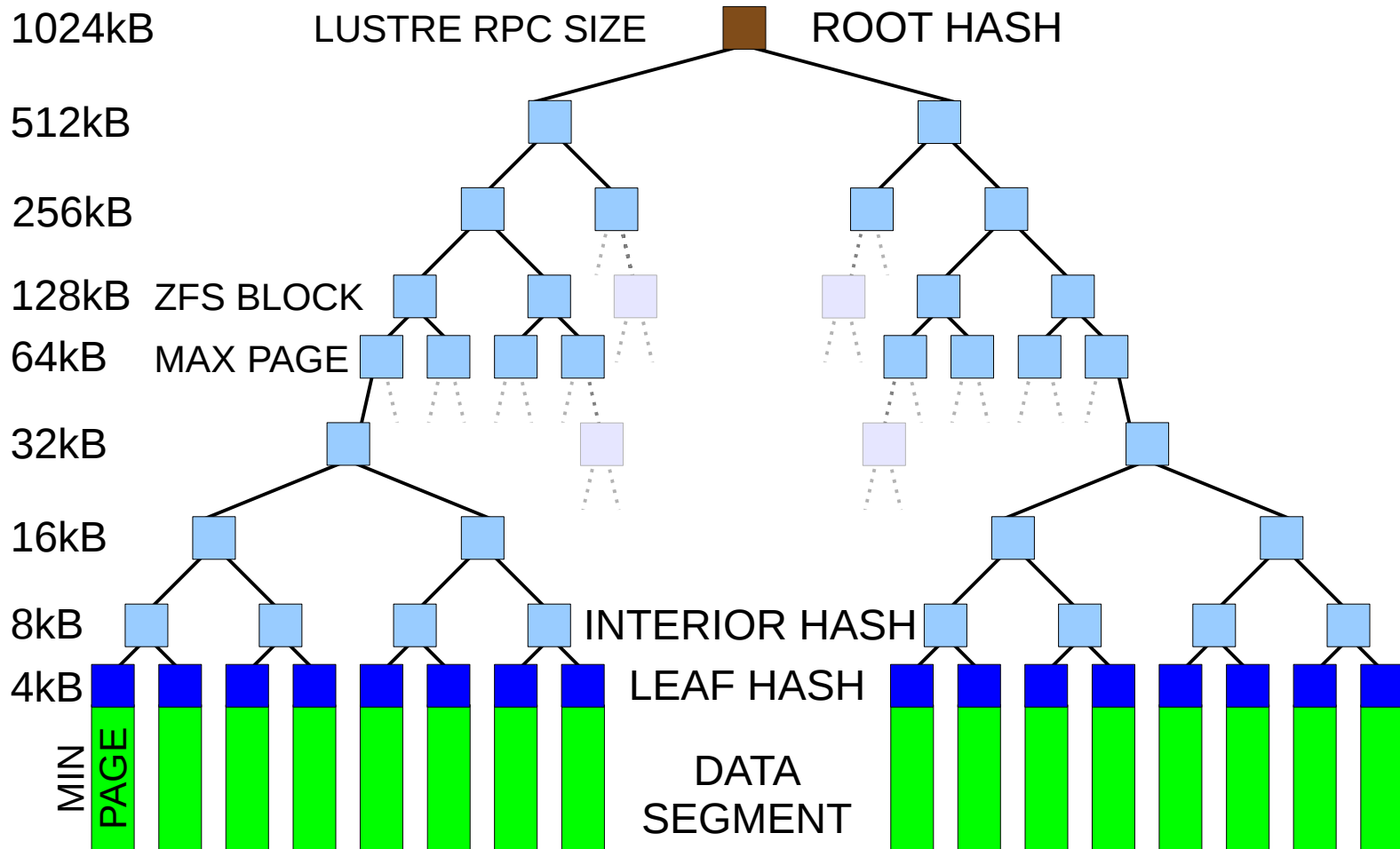
ZFS stores data/metadata checksums

- Fast (Fletcher-4 default, or none)
- Strong (SHA-256)

Combine for End-to-End Integrity

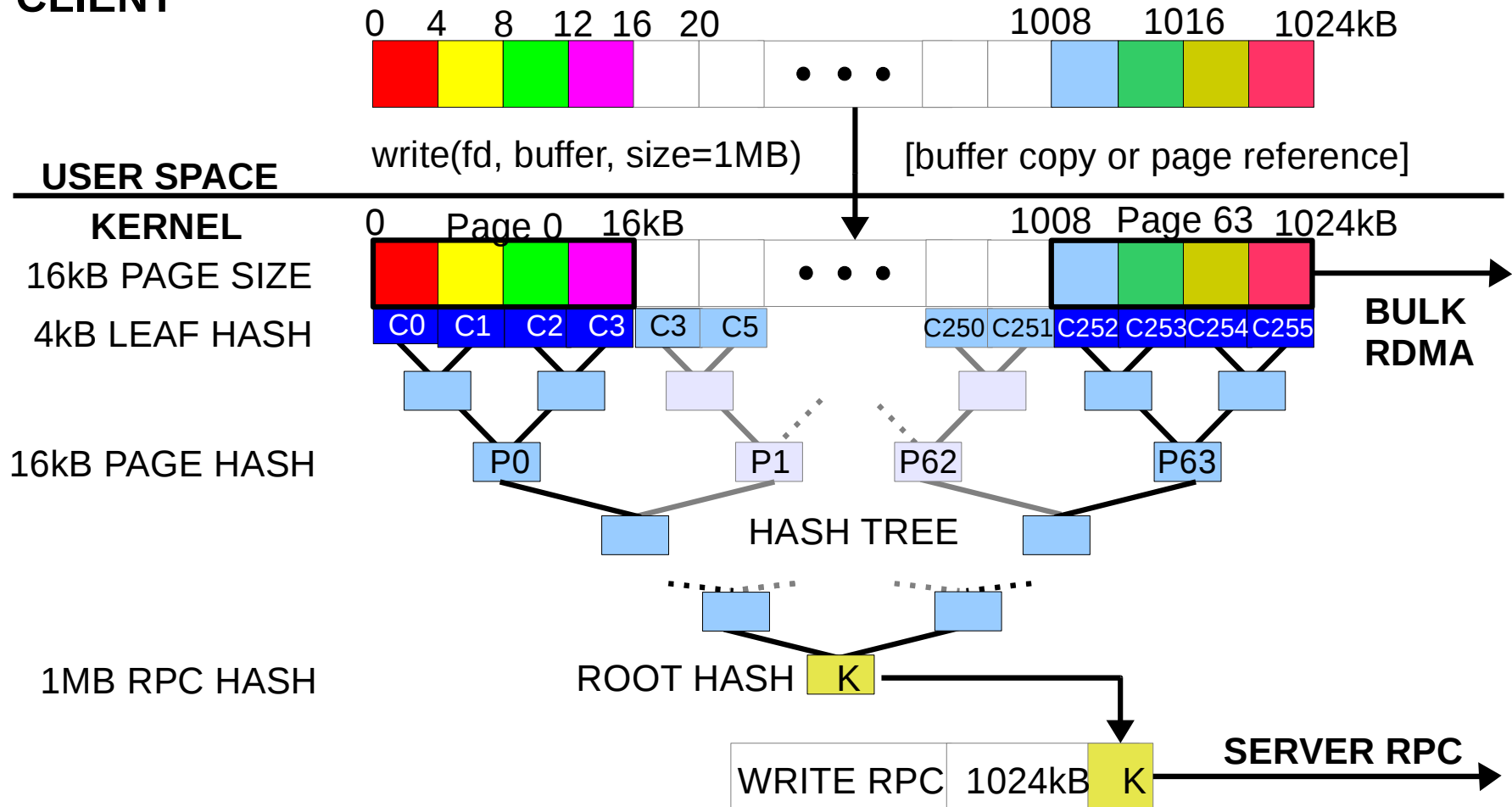
- Integrate Lustre and ZFS checksums
- Avoid recompute full checksum on data
- Always overlap checksum coverage
- Use scalable tree hash method

Hash Tree and Multiple Block Sizes



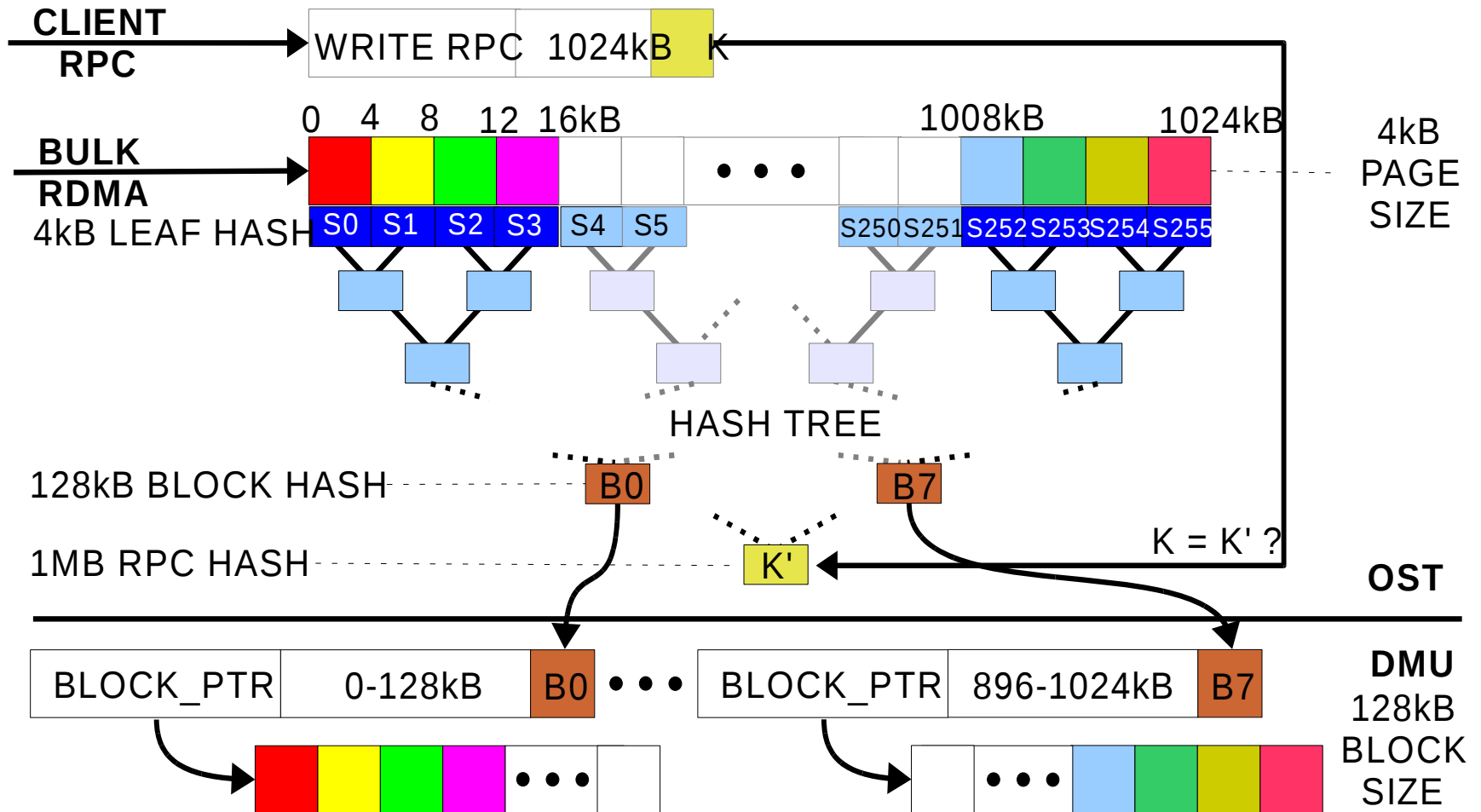
End-to-End Integrity Client Write

CLIENT



End-to-End Integrity Server Write

SERVER



Integrity Checking Targets

Scale of proposed filesystem is enormous

- 1 trillion files checked in 100h
- **2-4PB** of MDT filesystem metadata
- ~3 million files/sec, 3GB/s+ for one pass

Need to handle CMD coherency as well

- Distributed link count on files, directories
- Directory parent/child relationship
- Filename to FID to inode mapping

Distributed Coherency Checking

Traverse MDT filesystem(s) only once

- Piggyback on ZFS resilver/scrub to avoid extra IO
- Lustre processes blocks after checksum validation

Validate OST/remote MDT data opportunistically

- Track which objects have not been validated (1 bit/object)
- Each object has a back-reference to validate user
- Back-reference avoids need to keep global reference table

Lazily scan OST/MDT in the background

- Load sensitive scanning
- Continuous online verification/fix of distributed state

Lustre DMU development Status

Data Alpha

- OST basic functionality

MD Alpha

- MDS basic functionality, beginning recovery support

Beta

- Layouts, performance improvement, full recovery

GA

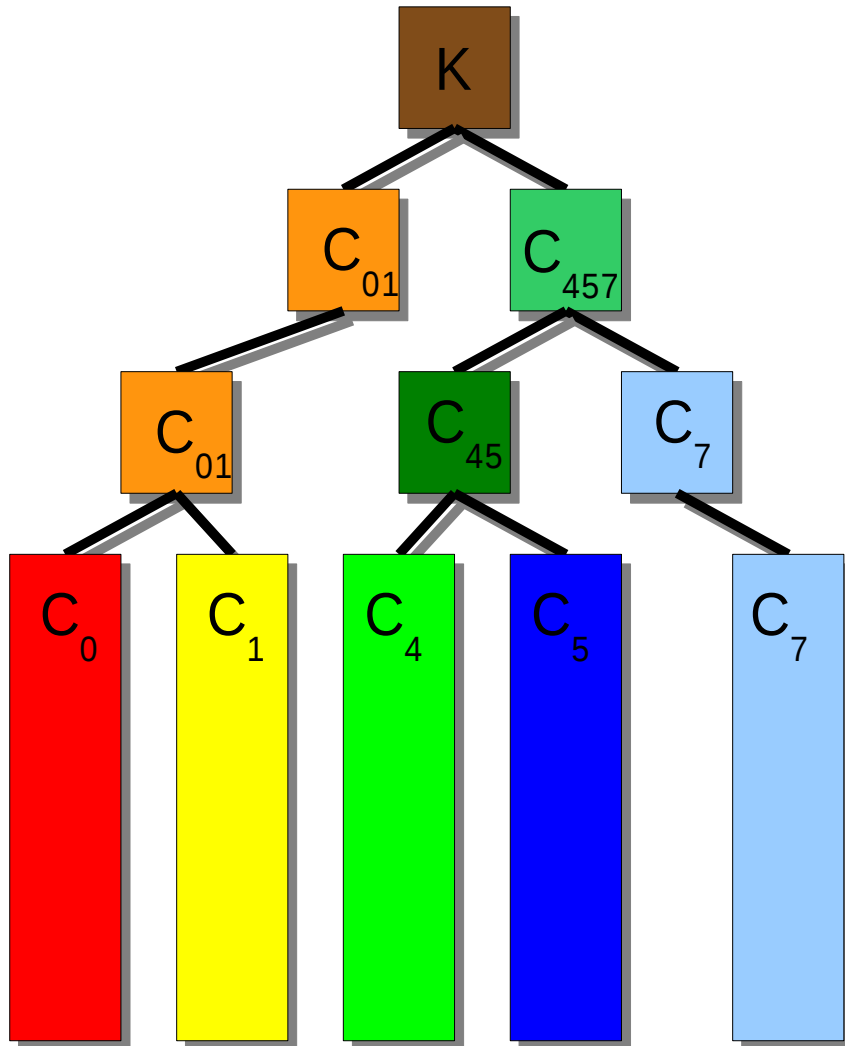
- Full performance in production release

A low-angle photograph of several rows of blue solar panels extending towards the horizon under a bright blue sky with scattered white clouds. A thick yellow curved line separates this image from the blue background on the right side of the slide.

Thank You

Questions?

Hash Tree For Non-contiguous Data



LH(x) = hash of data x (leaf)
 IH(x) = hash of data x (interior)
 x+y = concatenation x and y

$$C_0 = \text{LH}(\text{data segment 0})$$

$$C_1 = \text{LH}(\text{data segment 1})$$

$$C_4 = \text{LH}(\text{data segment 4})$$

$$C_5 = \text{LH}(\text{data segment 5})$$

$$C_7 = \text{LH}(\text{data segment 7})$$

$$C_{01} = \text{IH}(C_0 + C_1)$$

$$C_{45} = \text{IH}(C_4 + C_5)$$

$$C_{457} = \text{IH}(C_{45} + C_7)$$

$$K = \text{IH}(C_{01} + C_{457}) = \text{ROOT hash}$$

T10-DIF

vs.

Hash Tree

CRC-16 Guard Word

- All 1-bit errors
- All adjacent 2-bit errors
- Single 16-bit burst error
- 10^{-5} bit error rate

32-bit Reference Tag

- Misplaced write \neq 2nTB
- Misplaced read \neq 2nTB

Fletcher-4 Checksum

- All 1- 2- 3- 4-bit errors
- All errors affecting 4 or fewer 32-bit words
- Single 128-bit burst error
- 10^{-13} bit error rate

Hash Tree

- Misplaced read
- Misplaced write
- Phantom write
- Bad RAID reconstruction

ZFS Hybrid Pool Example



4 Xeon 7350 Processors (16 cores)
 32GB FB DDR2 ECC DRAM
 OpenSolaris™ with ZFS

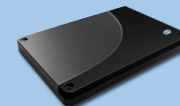
Configuration A

Seven 146GB 10,000 RPM SAS Drives

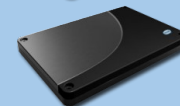


Configuration B

Five 400GB 4200 RPM SATA Drives



32GB SSD ZIL Device



80GB SSD Cache Device

ZFS Hybrid Pool Example

