Lustre, ZFS, and End-to-End Data Integrity

Andreas Dilger
Senior Staff Engineer, Lustre Group
Sun Microsystems

Regensburg HPC Workshop 2009

Topics

Overview

ZFS Data Integrity

Lustre Data Integrity

ZFS-Lustre Integration

2012 Lustre Requirements

Filesystem Limits

- 100 PB+ maximum file system size
- **1 trillion files** (10^{12}) per file system

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 Our Requirements

Capacity

- Single filesystem 100TB+ (2^{64} LUNs * 2^{64} bytes)
- Trillions of files in a single file system (2^{48} files)
- Dynamic addition of capacity/performance

Reliability and resilience

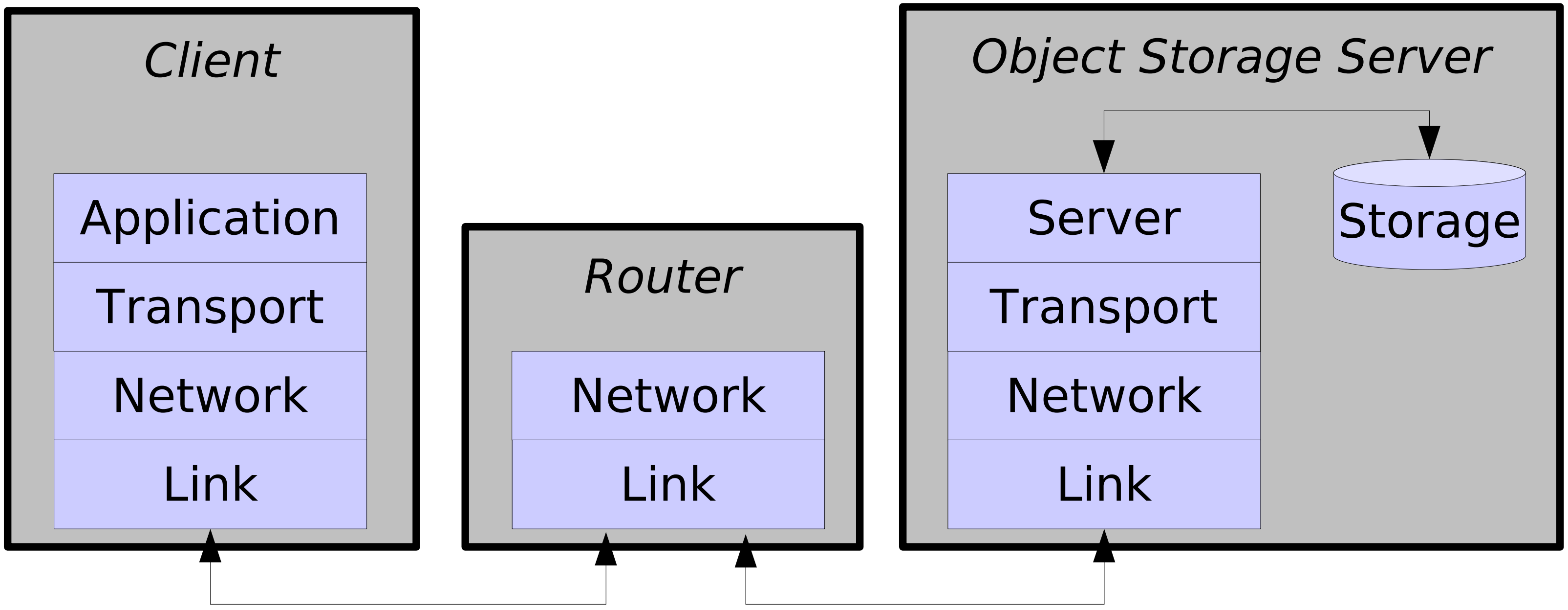
- Transaction based, copy-on-write
- Internal data redundancy (double parity, 3 copies)
- End-to-end checksum of all data/metadata
- Online integrity verification and reconstruction

Functionality

- Snapshots, filesets, compression, encryption
- Online incremental backup/restore
- Hybrid storage pools (HDD + SSD)

How Safe Is Your Data?

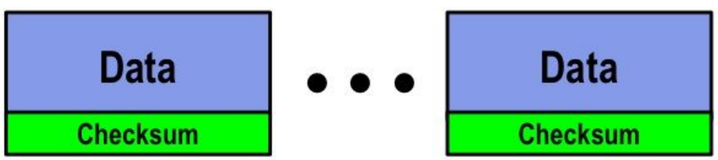
- There are many sources of data corruption
  - Software: client, NIC, router, server, HBA, disk!
  - RAM, CPU, disk cache, media
  - Client network, storage network, cables



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

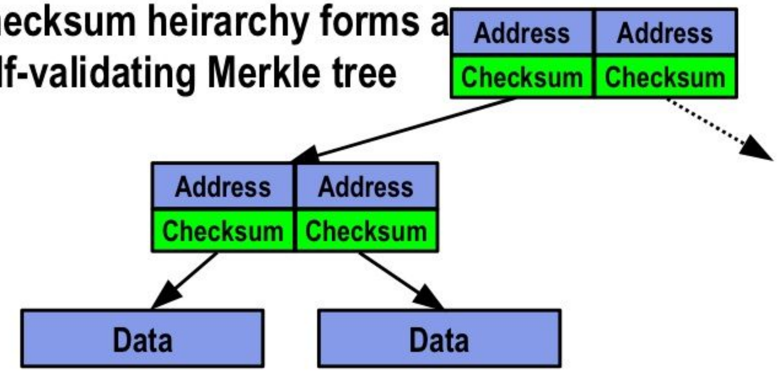


Disk checksum only validates media

- ✔ 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

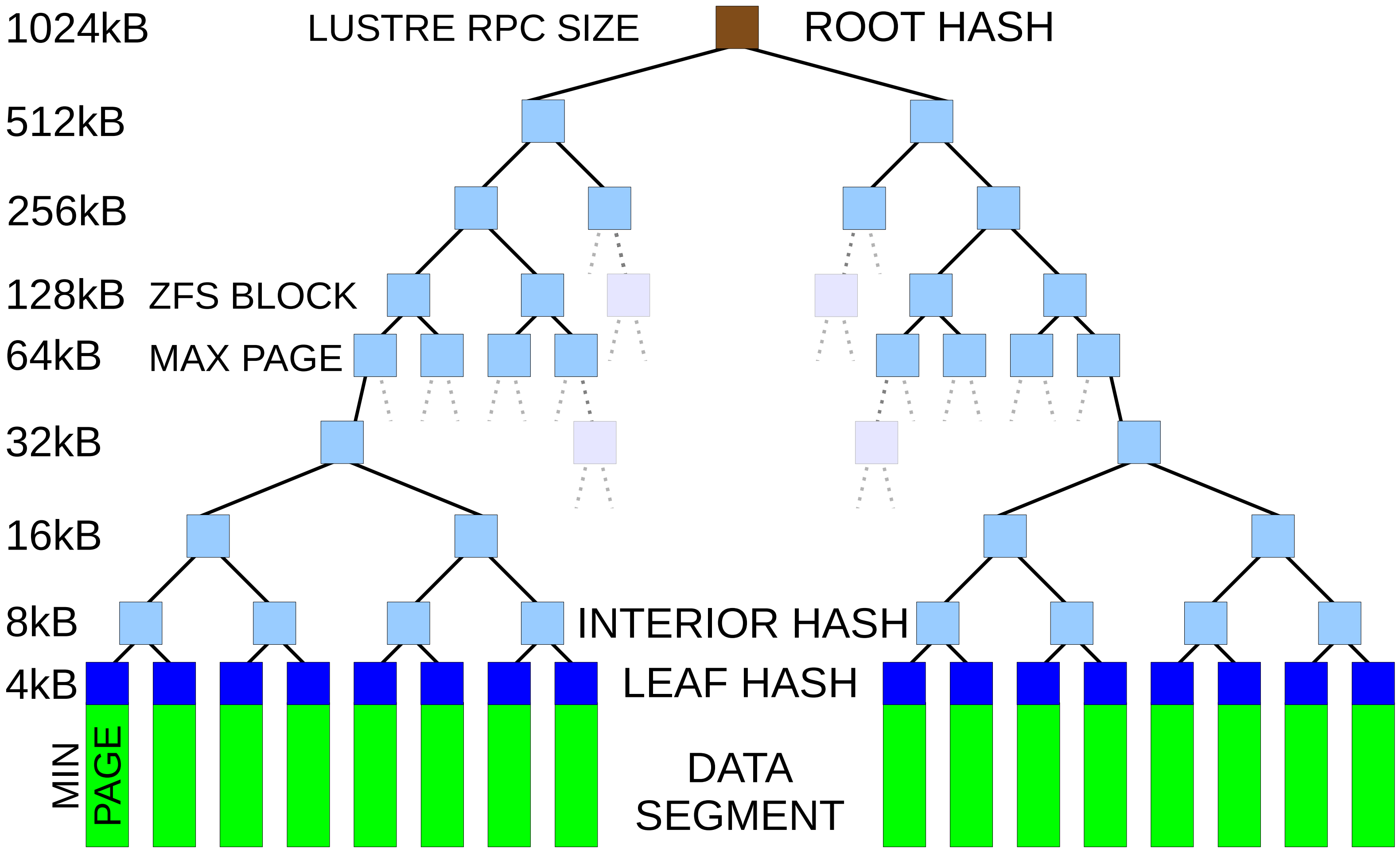
ZFS stores data/metadata checksums

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

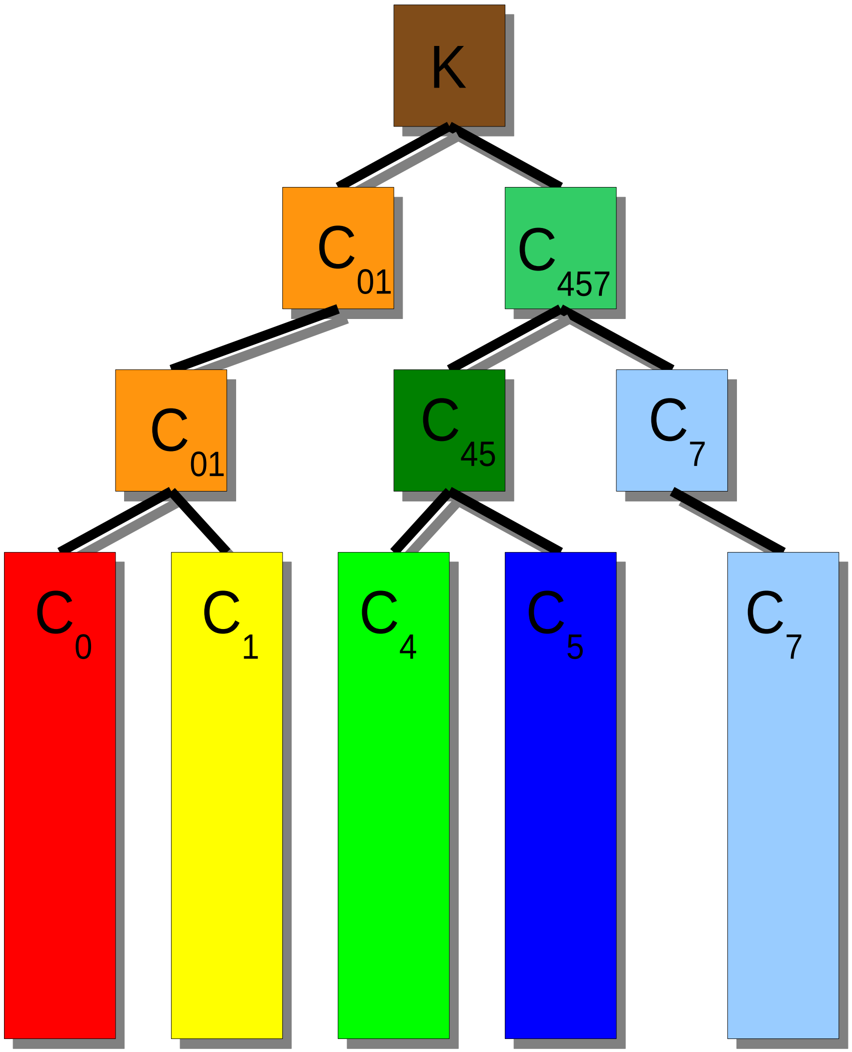
Combine for End-to-End Integrity

- 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



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

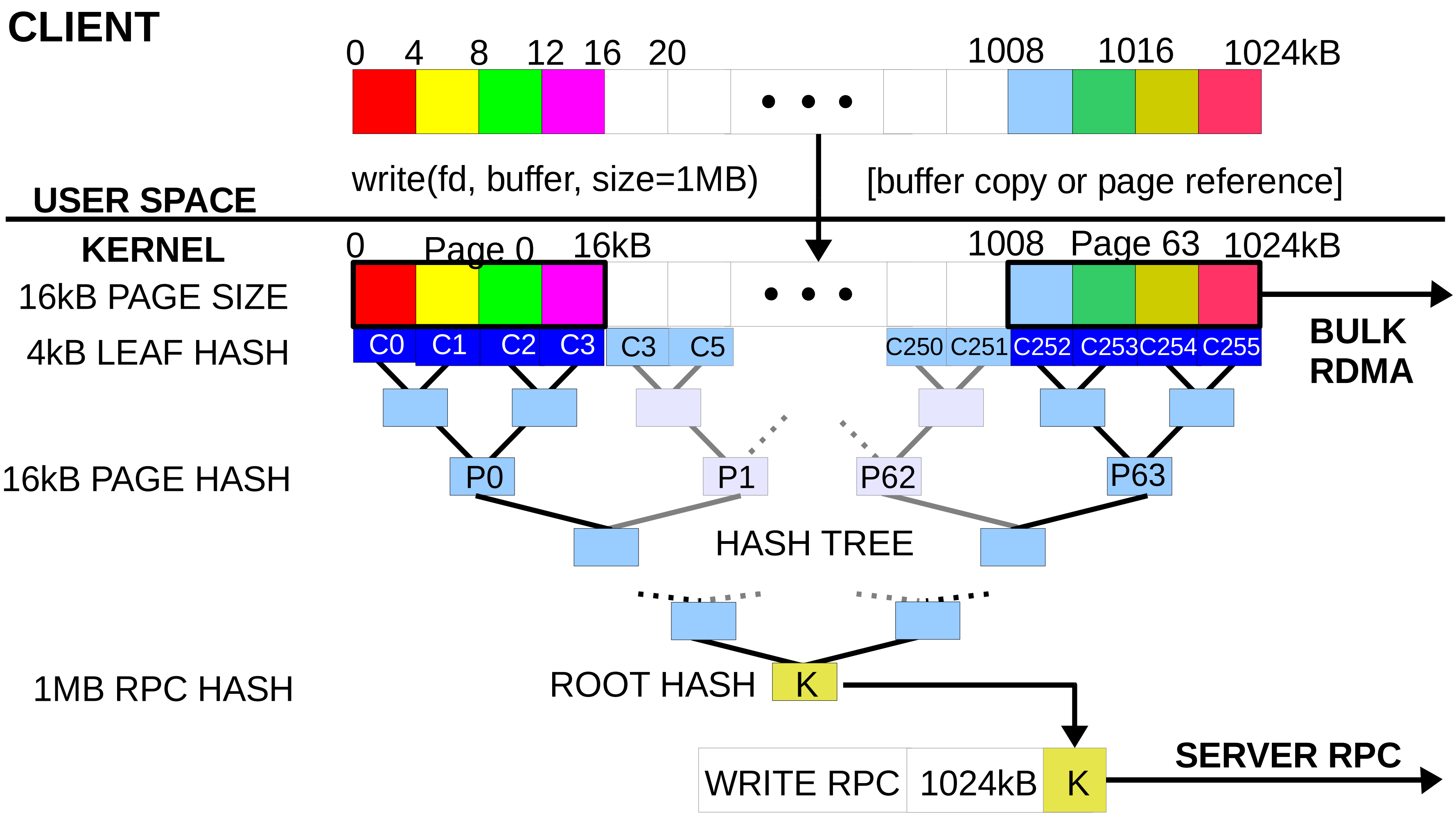
C_0 = LH(data segment 0)
C_1 = LH(data segment 1)
C_4 = LH(data segment 4)
C_5 = LH(data segment 5)
C_7 = LH(data segment 7)

$C_{01} = IH(C_0 + C_1)$
$C_{45} = IH(C_4 + C_5)$

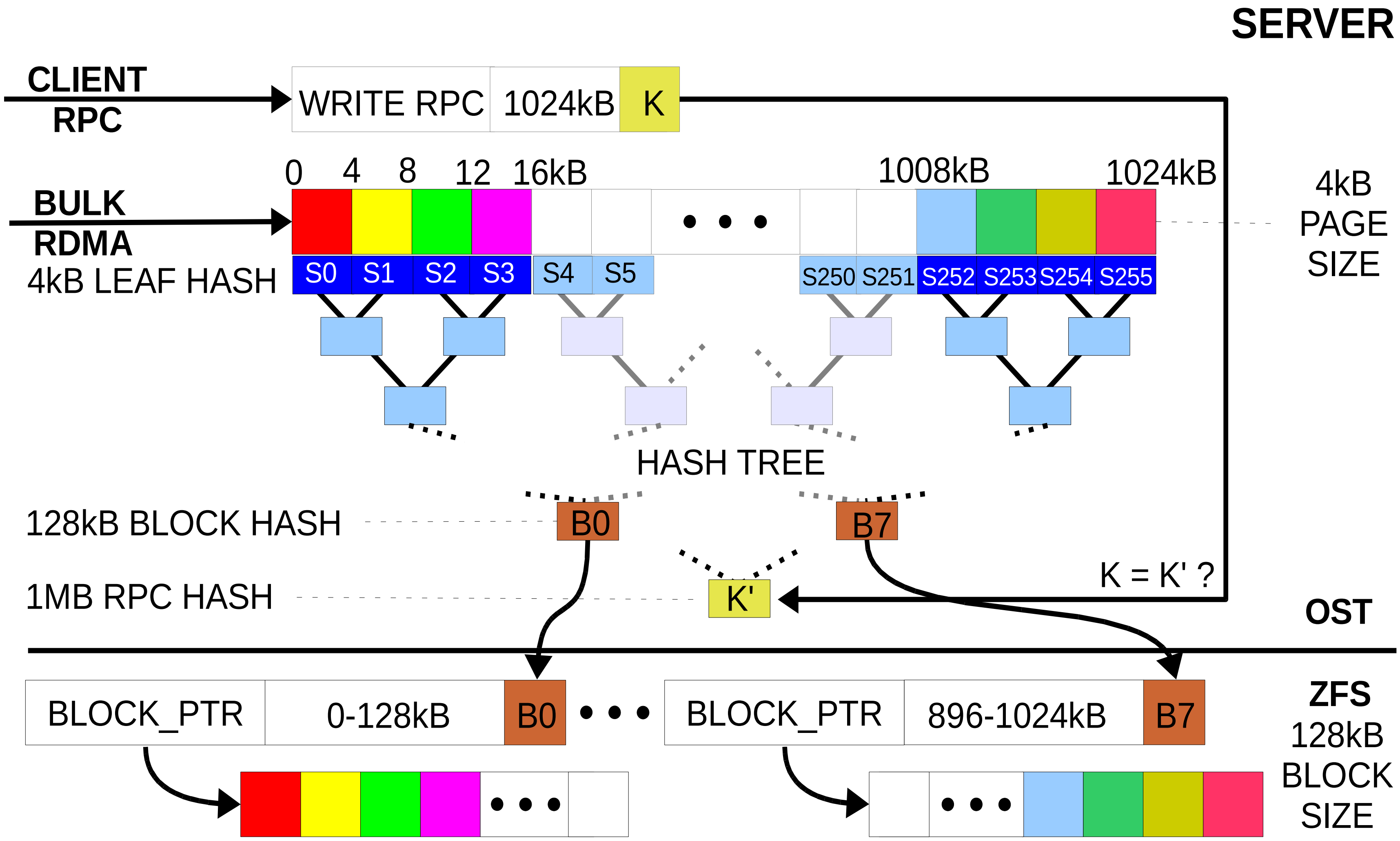
$C_{457} = IH(C_{45} + C_7)$

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

End-to-End Integrity Client Write



End-to-End Integrity Server Write



Filesystem Integrity Checking

Problem is among hardest to solve

- 1 trillion files in 100h
- **2-4PB** of MDT filesystem metadata
- ~3 million files/sec, 3GB/s+ for one pass
- 3M*stripes checks/sec from MDSes to OSSes
- 860*stripes random metadata IOPS on OSTs

Need to handle CMD coherency as well

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

ZFS-level RAID-Z Resilver

RAID-Z/Z2 is not the same as RAID-5/6

- NEVER does read-modify-write
- Allow arbitrary block size/alignment
- RAID layout is stored in block pointer

ZFS metadata traversal for RAID rebuild

- Rebuild used storage only (<80%)
- Verify checksum of rebuilt data
- Callout to Lustre handler after check

Lustre-level Rebuild Mitigation

Two related problems

- Avoid global impact *from* degraded RAID
- Avoid load *on* rebuilding RAID set

Solution: avoid degraded OSTs

- Little or no load on degraded RAID set
- Maximize rebuild performance
- Minimal global performance impact
- 30 disks (3 LUNs) per OST, 1224 OSTs
- 38 of 1224 OSTS = 3% aggregate cost
- OSTs available for existing files

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 != 2nTB
- Misplaced read != 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