Lustre HPCS Design Overview

Andreas Dilger
Senior Staff Engineer, Lustre Group
Sun Microsystems

Scalability Workshop – ORNL – May 2009

Topics

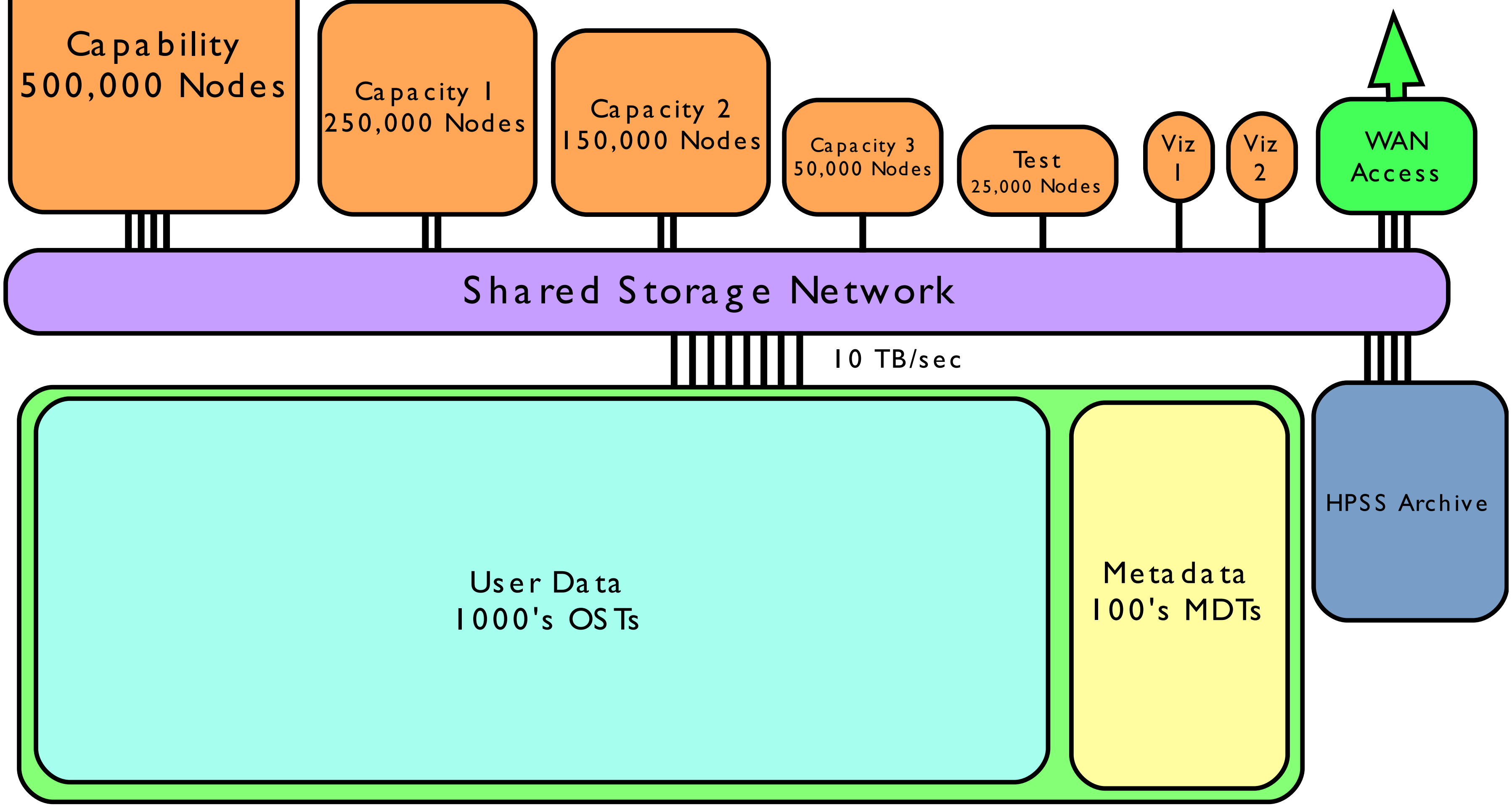
HPCS Goals

HPCS Architectural Improvements

Performance Enhancements

Conclusion

HPC Center of the Future



HPCS Goals - Capacity

Filesystem Limits

- 100 PB+ maximum file system size *(10PB)*
- **1 trillion files** (10^{12}) per file system *(4 billion files)*
- > 30k client nodes *(> 30k clients)*

Single File/Directory Limits

- **10 billion** files per directory *(15M)*
- 0 to 1 PB file size range *(360TB)*
- 1B to 1 GB I/O request size range *(1B - 1GB IO req)*
- **100,000 open** shared files per process *(10,000 files)*
- Long file names and 0-length files as data records
  (long file names and 0-length files)

HPCS Goals - Performance

Aggregate

- 10,000 metadata operations per second *(10,000/s)*
- **1500 GB/s** file per process/shared file *(180GB/s)*
- No impact RAID rebuilds on performance *(variable)*

Single Client

- **40,000 creates/s**, up to 64kB data *(5k/s, 0kB)*
- **30 GB/s** full-duplex streaming I/O *(2GB/s)*

Miscellaneous

- POSIX I/O API extensions proposed at OpenGroup* *(partial)*

* High End Computing Extensions Working Group http://www.opengroup.org/platform/hecewg/

HPCS Goals - Reliability

- End-to-end resiliency T10 DIF equivalent *(net only)*
- No impact RAID rebuild on performance *(variable)*
- Uptime of **99.99%** *(99% ?)*
  - Downtime < **1h/year**
- **100h** filesystem integrity check *(8h, partial)*
  - 1h downtime means check must be online

HPCS Architectural Improvements

Use Available ZFS Functionality

End-to-End Data Integrity

RAID Rebuild Performance Impact

Filesystem Integrity Checking

Clustered Metadata

Recovery Improvements

Performance Enhancements

Use Available ZFS Functionality

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

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)

End-to-End Data Integrity

Current Lustre checksumming

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

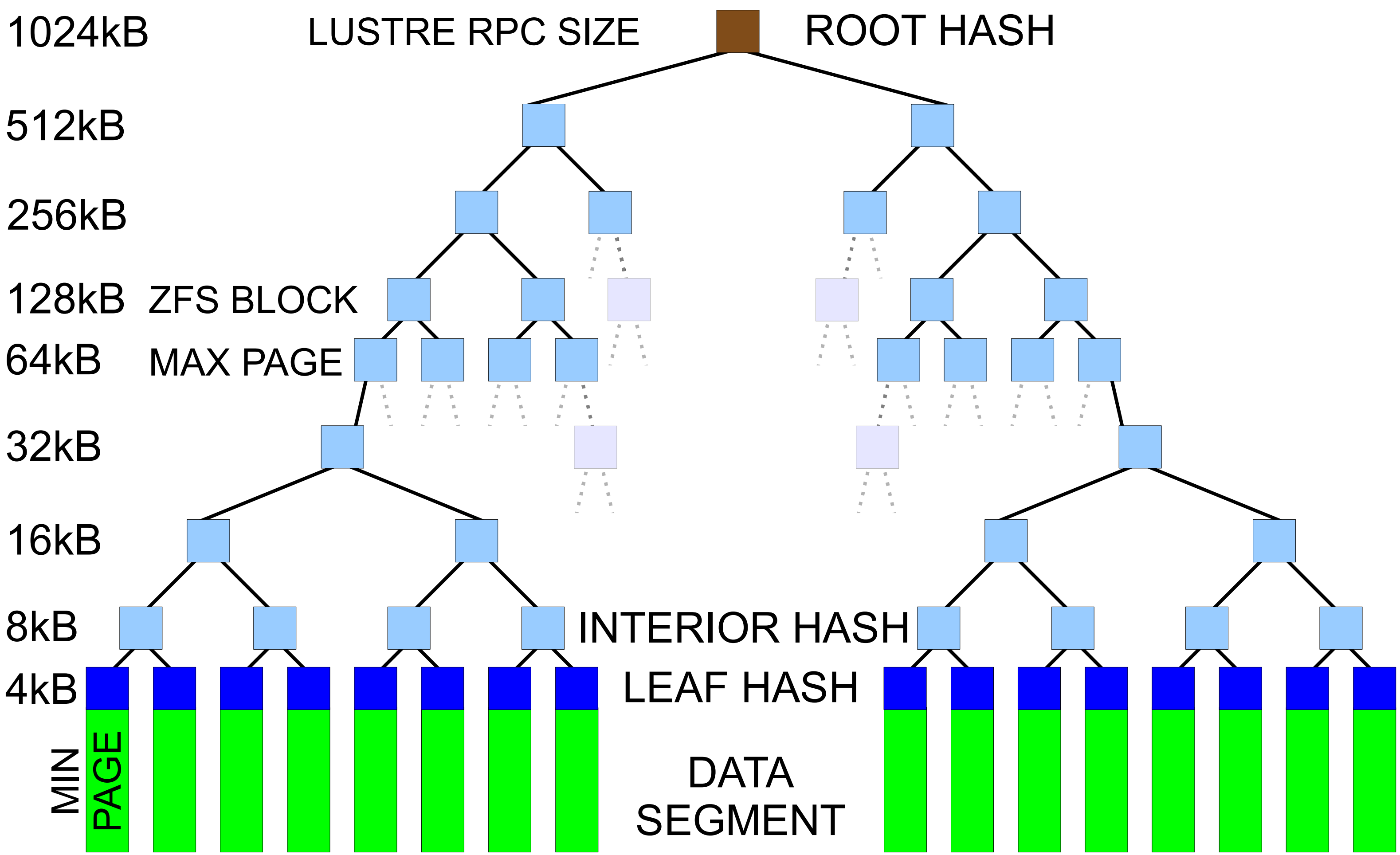
ZFS stores data/metadata checksums

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

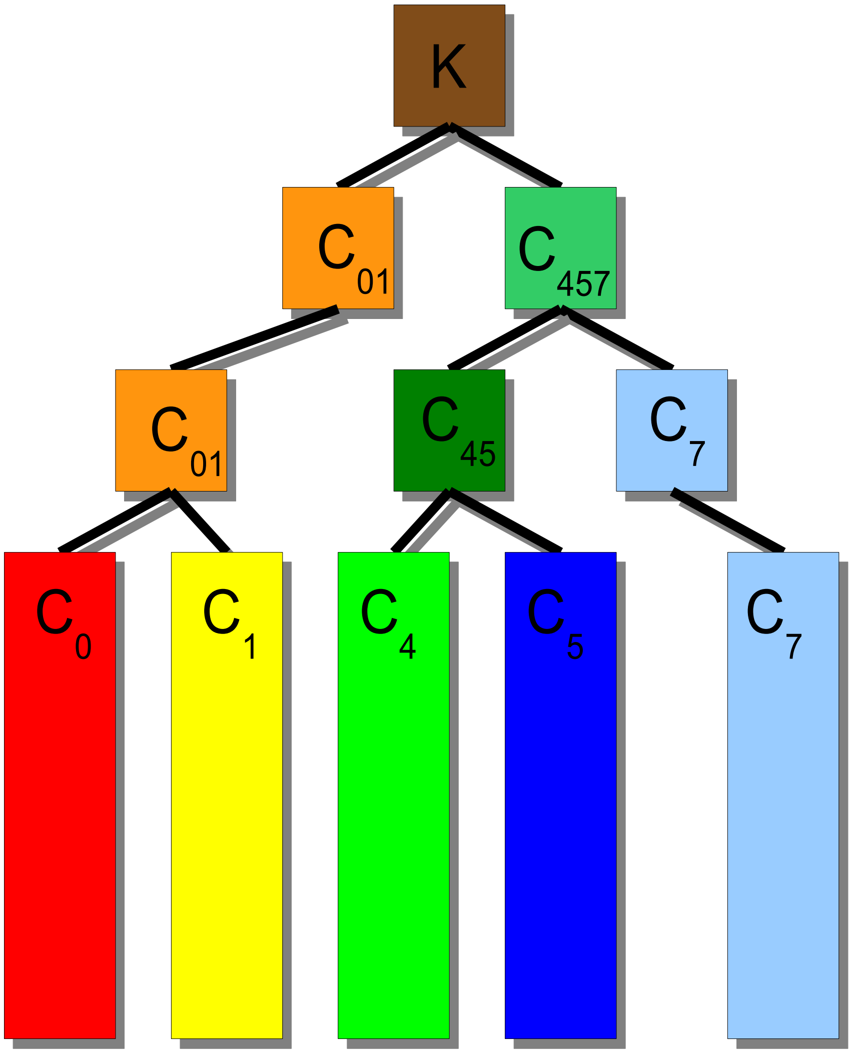
HPCS Integration

- 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

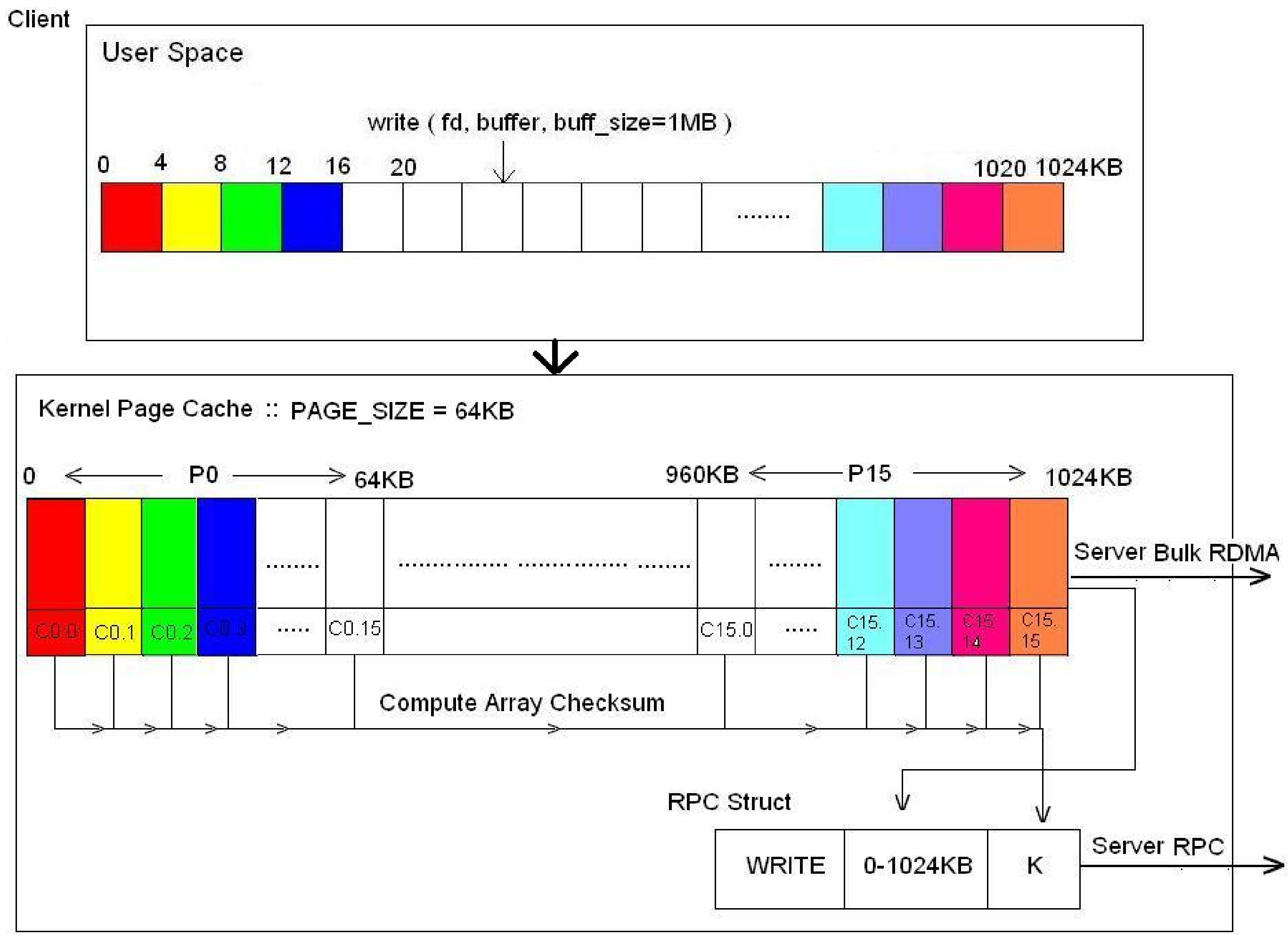
C_0 = LH(data segment 0)
C_1 = LH(data segment 1)
C_4 = LH(data segment 4)
C_5 = LH(data segment 5)
C_7 = LH(data segment 7)

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

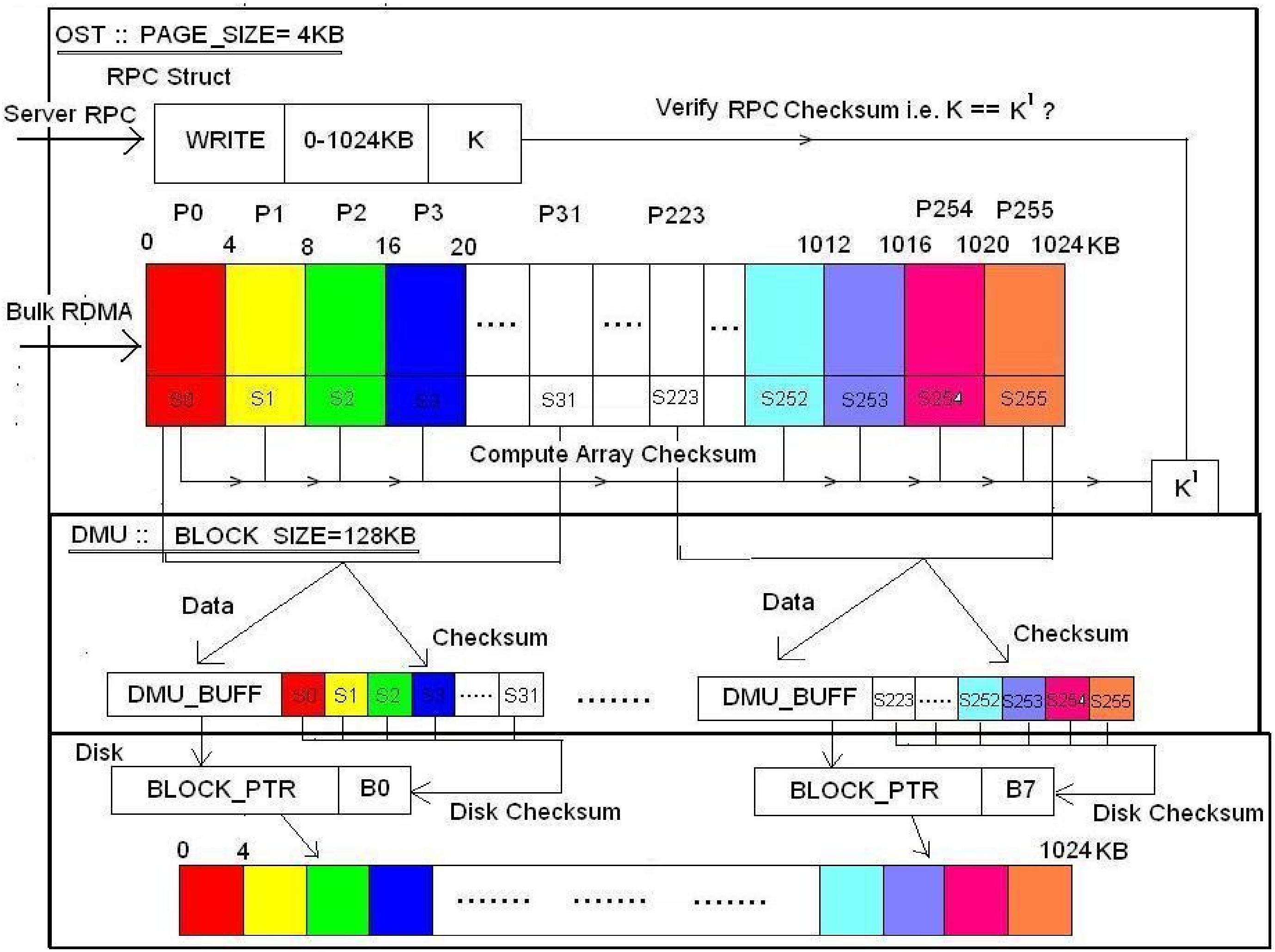
$C_{457} = IH(C_{45} + C_7)$

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

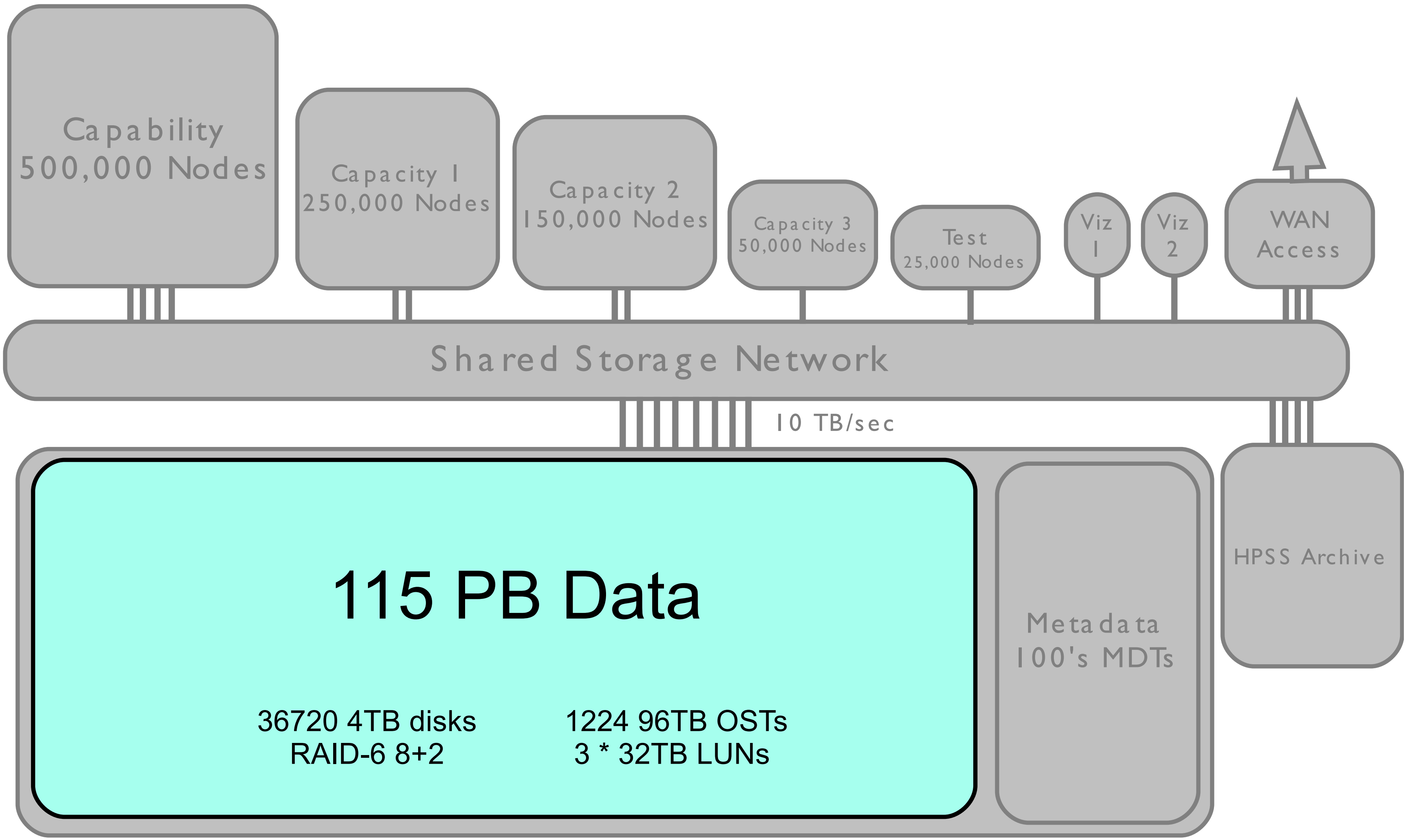
End-to-End Integrity Client Write



End-to-End Integrity Server Write



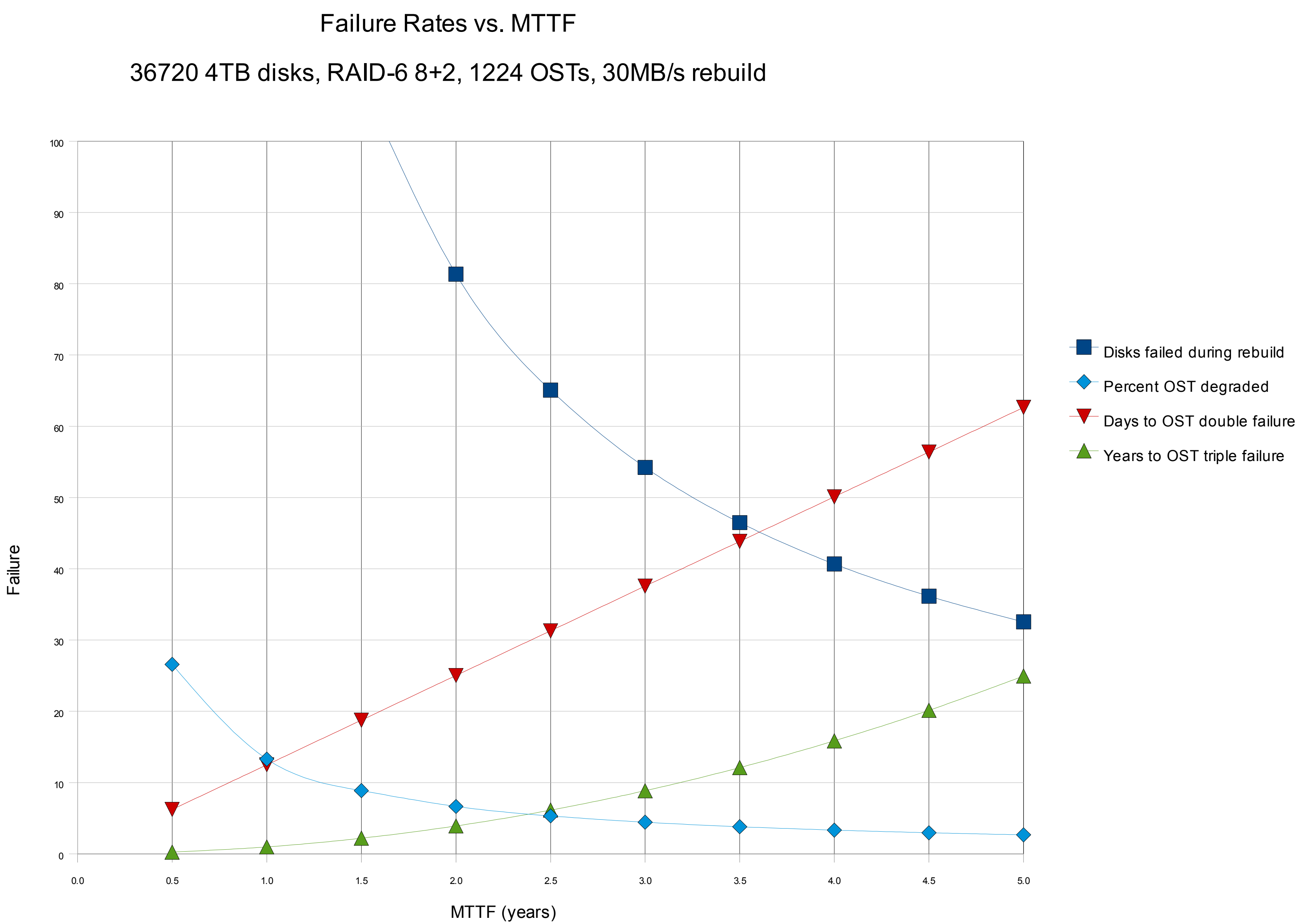
HPC Center of the Future



RAID Failure Rates

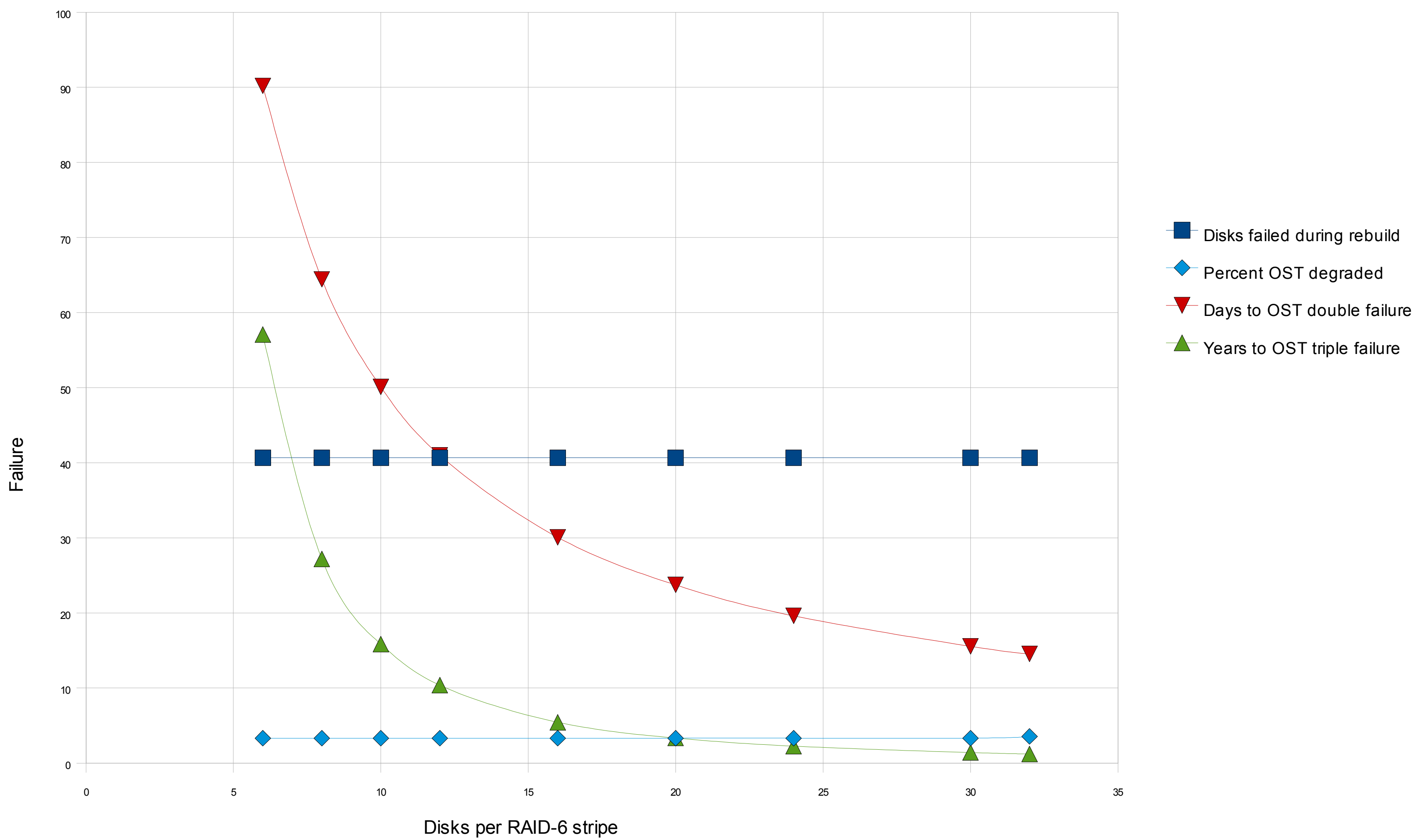
115PB filesystem, 1.5TB/s

- 36720 4TB disks in RAID 6 8+2 LUNs
- 4 year Mean Time To Failure
- 1 disk fails **every hour** on average
- 4TB disk @ 30MB/s → 38hr rebuild
- 30MB/s is 50%+ disk bandwidth (seeks)
- May reduce aggregate throughput by 50%+
- 1 disk failure may cost 750GB/s aggregate
- 38hr * 1 disk/hr = 38 disks/OSTs degraded


Failure Rates vs. MTTF
36720 4TB disks, RAID-6 8+2, 1224 OSTs, 30MB/s rebuild
Failure
100
90
80
70
60
50
40
30
20
10
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
MTTF (years)
Disks failed during rebuild
Percent OST degraded
Days to OST double failure
Years to OST triple failure

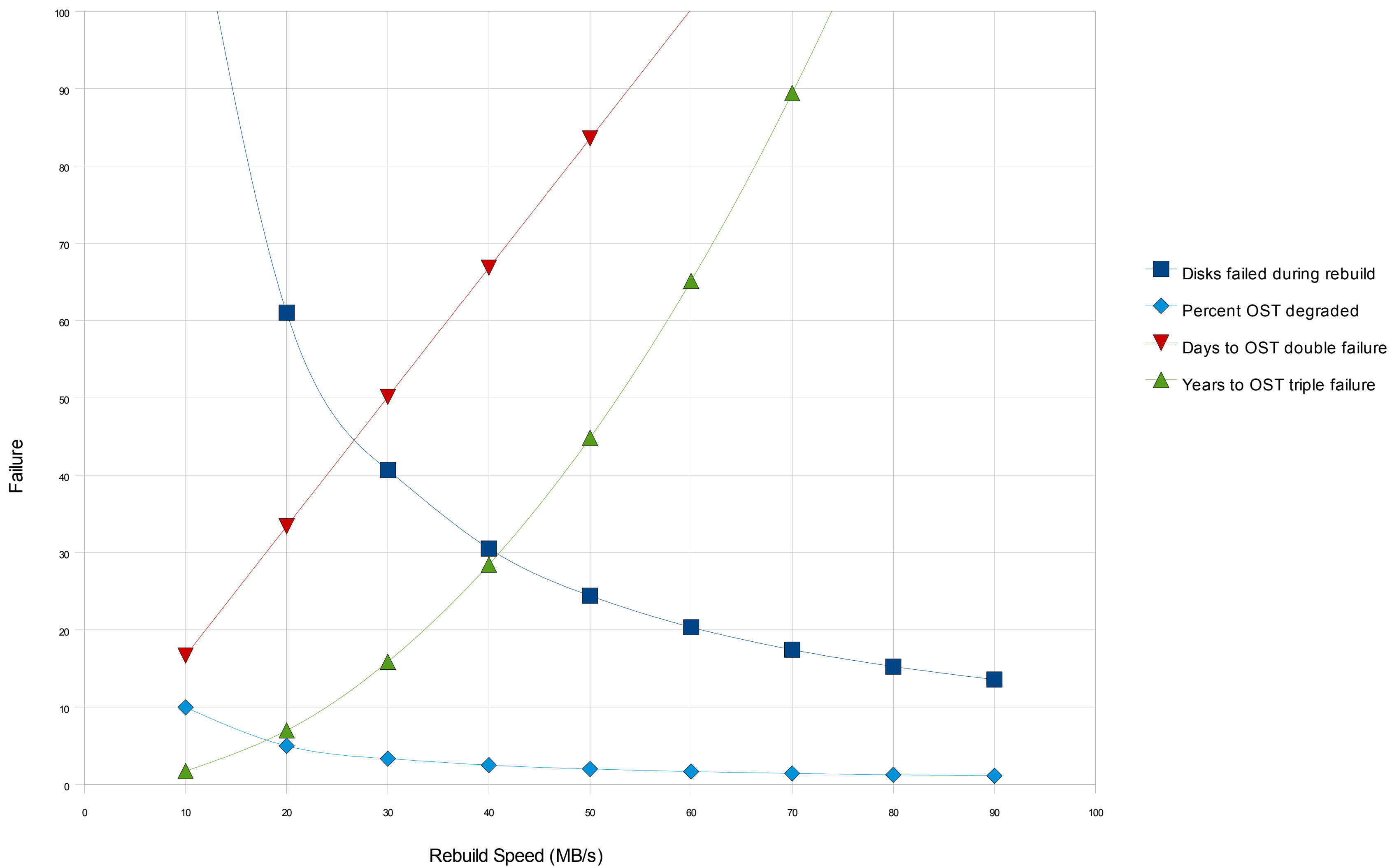
Failure Rates vs. RAID disks

36720 4TB disks, RAID-6 N+2, 1224 OSTs, MTTF 4 years



Failure Rates vs. Rebuild Speed

36720 4TB disks, RAID-6 8+2, 1224 OSTs, MTTF 4 years



Lustre-level Rebuild Mitigation

Several related problems

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

Avoids degraded OSTs for new files

- 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

ZFS-level RAID-Z Rebuild

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

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

ZFS metadata traversal for RAID rebuild

- Good: only rebuild used storage (<80%)
- Good: verify checksum of rebuilt data
- Bad: may cause random disk access

RAID-Z Rebuild Improvements

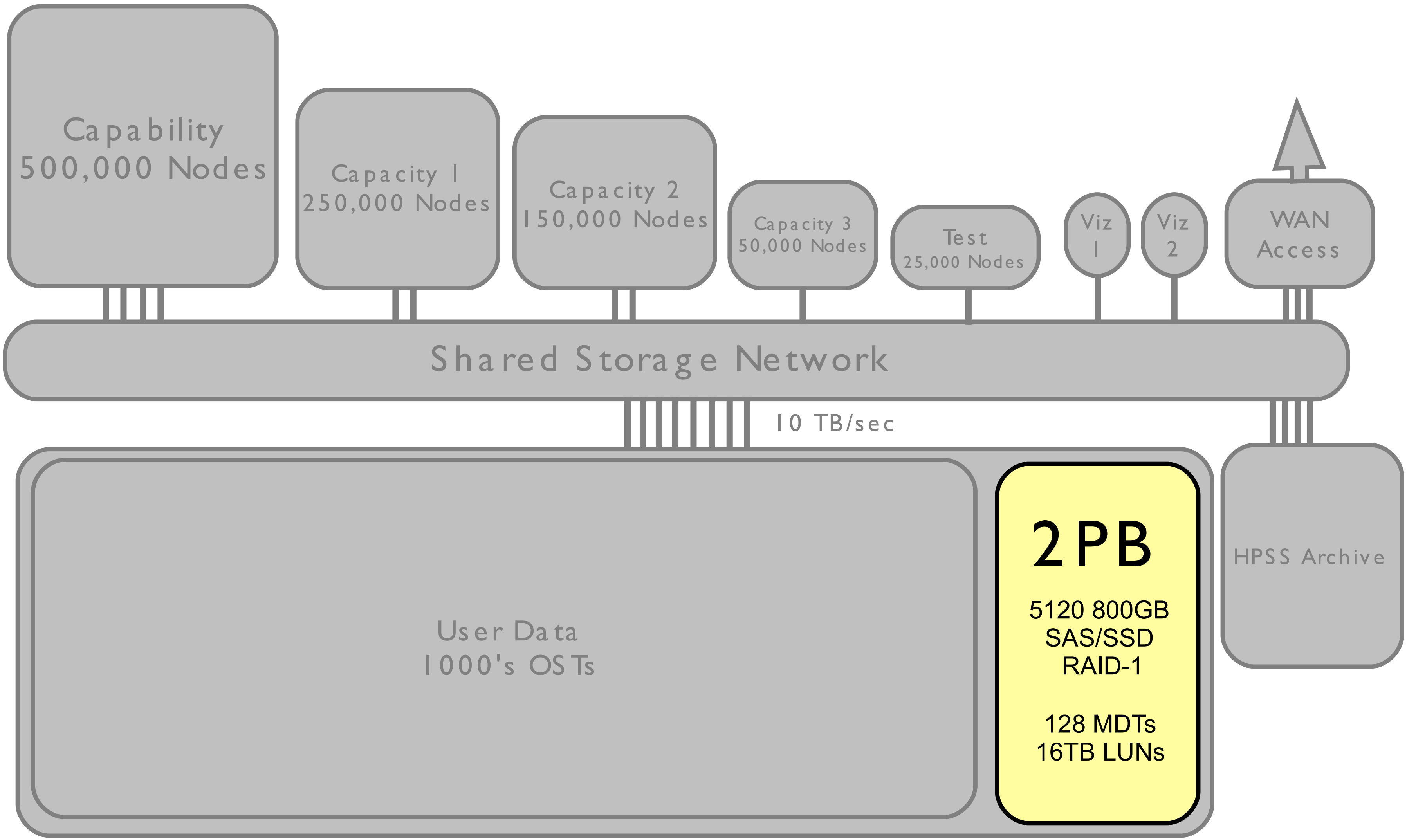
RAID-Z optimized rebuild

- ~3% of storage is metadata
- Scan metadata first, build ordered list
- Data reads mostly linear
- Bookmark to restart rebuild
- ZFS itself is not tied to RAID-Z

Distributed hot space

- Spread hot-spare rebuild space over all disks
- All disks' bandwidth/IOPS for normal IO
- All disks' bandwidth/IOPS for rebuilding

HPC Center of the Future



Clustered Metadata

100s of metadata servers

Distributed inodes

- Files normally local to parent directory
- Subdirectories often non-local

Split directories

- Split dir::name hash ⇔ Striped file::offset

Distributed Operation Recovery

- Cross directory mkdir, rename, link, unlink
- Strictly ordered distributed updates
- Ensures namespace coherency, recovery
- At worst inode refcount too high, leaked

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

Filesystem Integrity Checking

Integrate Lustre with ZFS scrub/rebuild

- ZFS callback to check Lustre references
- Event-driven checks means fewer re-reads
- ldiskfs can use an inode table iteration

Back-pointers to allow direct verification

- Pointer from OST object to MDT inode
- Pointer list from inode to {parent dir, name}
- No saved state needed for coherency check
- About 1 bit/block to detect leaks/orphans
    - Or, a second pass in reverse direction

Recovery Improvements

Version Based Recovery

- Independent recovery stream per file
- Isolate recovery domain to dependent ops

Commit on Share

- Avoid client getting any dependent state
- Avoid sync for single client operations
- Avoid sync for independent operations

Imperative Recovery

Server driven notification of failover

- Server notifies client of failover completed
- Client replies immediately to server
- Avoid client waiting on RPC timeouts
- Avoid server waiting for dead clients

Can tell between slow/dead server

- No waiting for RPC timeout start recovery
- Can use external or internal notification

Performance Enhancements

SMP Scalability

Network Request Scheduler

Channel Bonding

SMP Scalability

Future nodes will have 100s of cores

- Need excellent SMP scaling on client/server
- Need to handle NUMA imbalances

Remove contention on servers

- Per-CPU resources (queues, locks)
- Fine-grained locking
- Avoid cross-node memory access
    - Bind requests to a specific CPU deterministically
        - Client NID, object ID, parent directory

Remove contention on clients

- Parallel copy_{to,from}_user, checksums

Network Request Scheduler

Much larger working set than disk elevator

- Higher level information
  - Client NID, File/Offset

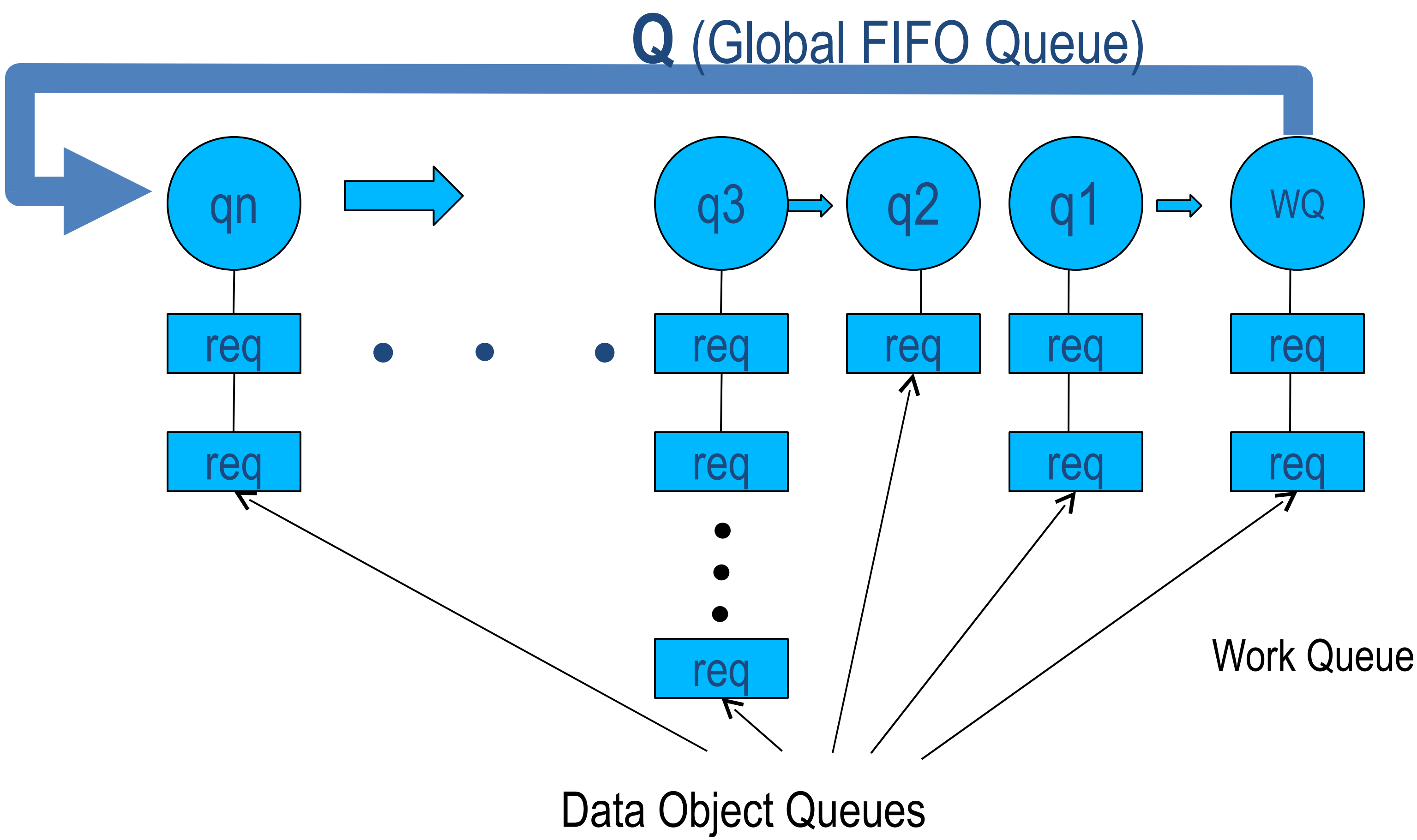
Read & write queue for each object on server

- Requests sorted in object queues by offset
  - Queues serviced round-robin, operation count (variable)
  - Deadline for request service time
- Scheduling input: opcount, offset, fairness, delay

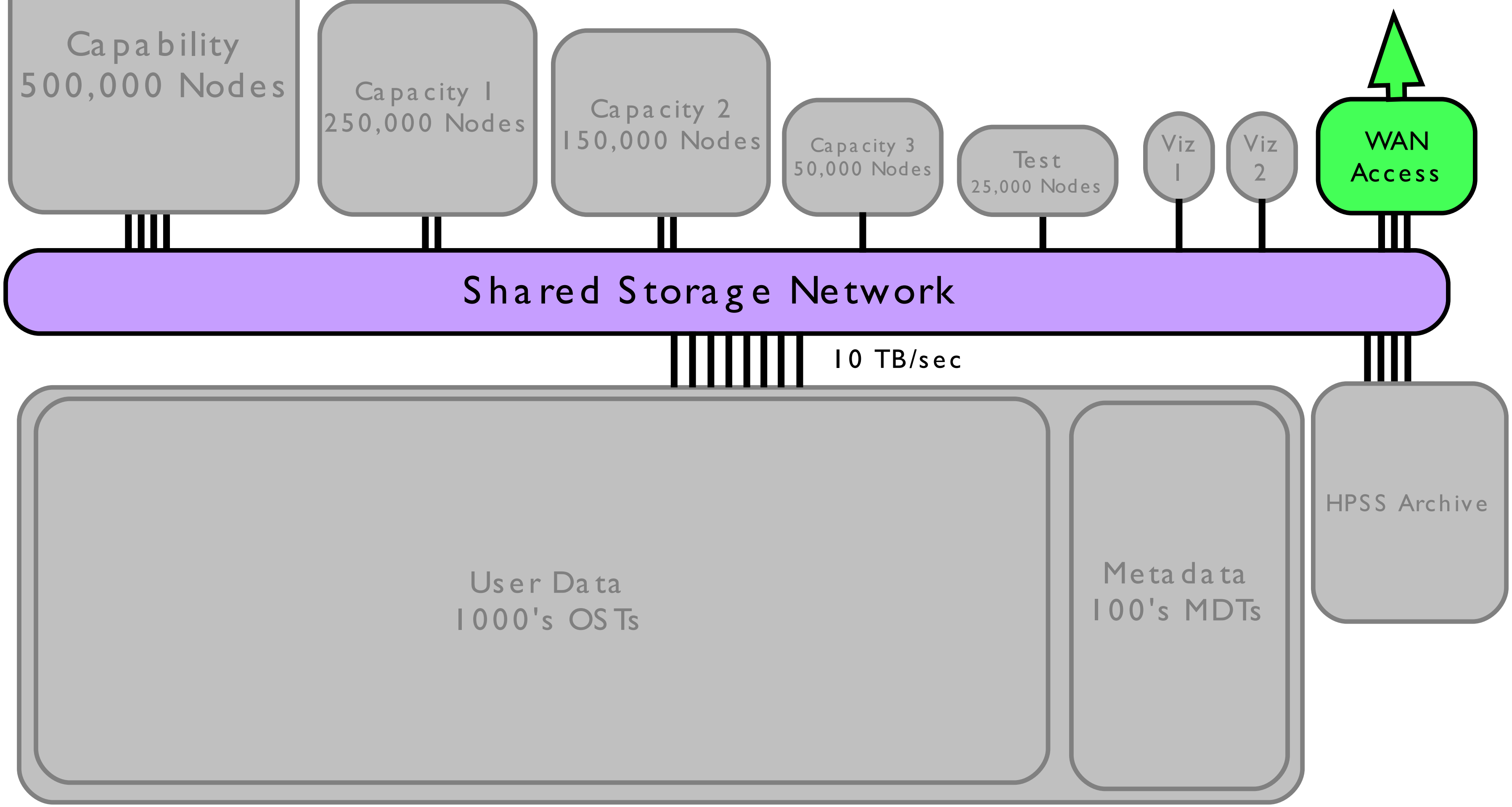
Future enhancements

- Job ID, process rank
- Gang scheduling across servers
- Quality of service
  - Per UID/GID, cluster: min/max bandwidth

Network Request Scheduler



HPC Center of the Future



Channel Bonding

Combine multiple Network Interfaces

- Increased performance
  - Shared: balance load across all interfaces
- Improved reliability
  - Failover: use backup links if primary down
- Flexible configuration
  - Network interfaces of different types/speeds
  - Peers do not need to share all networks
  - Configuration server per network

Conclusion

Lustre can meet HPCS filesystem goals

- Scalability roadmap is reasonable
- Incremental orthogonal improvements

HPCS provides impetus to grow Lustre

- Accelerate development roadmap
- Ensures that Lustre will meet future needs

Questions?

HPCS Filesystem Overview online at:

http://wiki.lustre.org/index.php/Learn:Lustre_Publications

THANK YOU

Andreas Dilger
Senior Staff Engineer, Lustre Group
Sun Microsystems

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

Metadata Improvements

Metadata Writeback Cache

- Avoids unnecessary server communication
  - Operations logged/cached locally
  - Performance of local file system when uncontended
- Aggregated distributed operations
  - Server updates batched and tranferred using bulk protocols (RDMA)
  - Reduced network and service overhead

Sub-Tree Locking

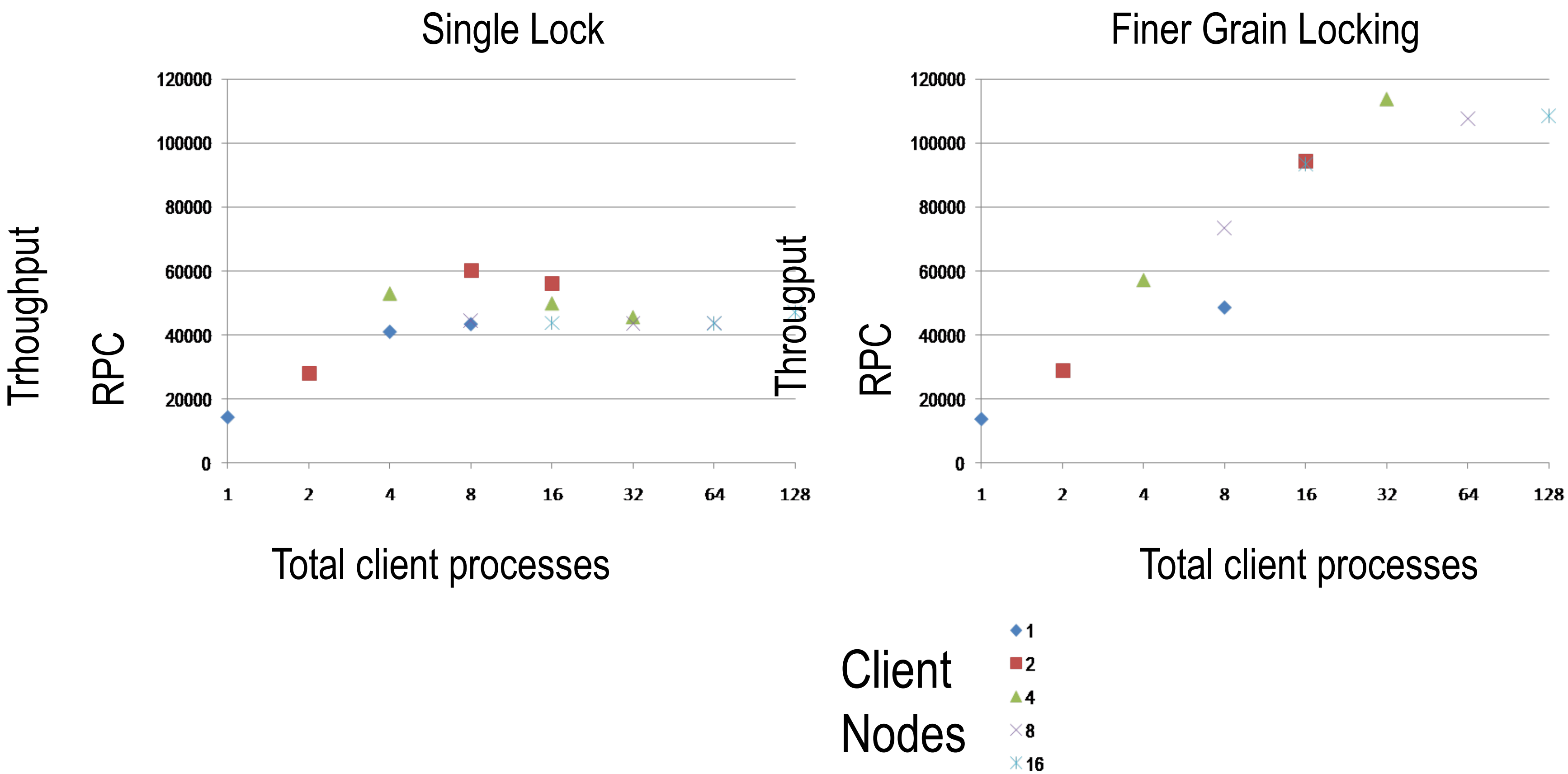
- Lock aggregation – a single lock protects a whole subtree
- Reduce lock traffic and server load

Metadata Improvements

Metadata Protocol

- Size on MDT (SOM)
  - Avoid multiple RPCs for attributes derived from OSTs
  - OSTs remain definitive while file open
  - Compute on close and cache on MDT
- Readdir+
  - Aggregation
    - Directory I/O
    - Getattrs
    - Locking

LNET SMP Server Scaling



Communication Improvements

Flat Communications model

- Stateful client/server connection required for coherence and performance
- Every client connects to every server
- O(n) lock conflict resolution

Hierarchical Communications Model

- Aggregate connections, locking, I/O, metadata ops
- Caching clients
  - Lustre ⇔ System Calls
  - Aggregate local processes (cores)
  - I/O Forwarders scale another 32x or more
- Caching Proxies
  - Lustre ⇔ Lustre
  - Aggregate whole clusters
  - Implicit Broadcast - scalable conflict resolution

Fault Detection Today

RPC timeout

- Timeout cannot distinguish death / congestion

Pinger

- No aggregation across clients or servers
- O(n) ping overhead

Routed Networks

- Router failure confused with peer failure

Fully automatic failover scales with slowest time constant

- 10s of minutes on large clusters ☹
- Finer failover control could be <u>much</u> faster ☺

Architectural Improvements

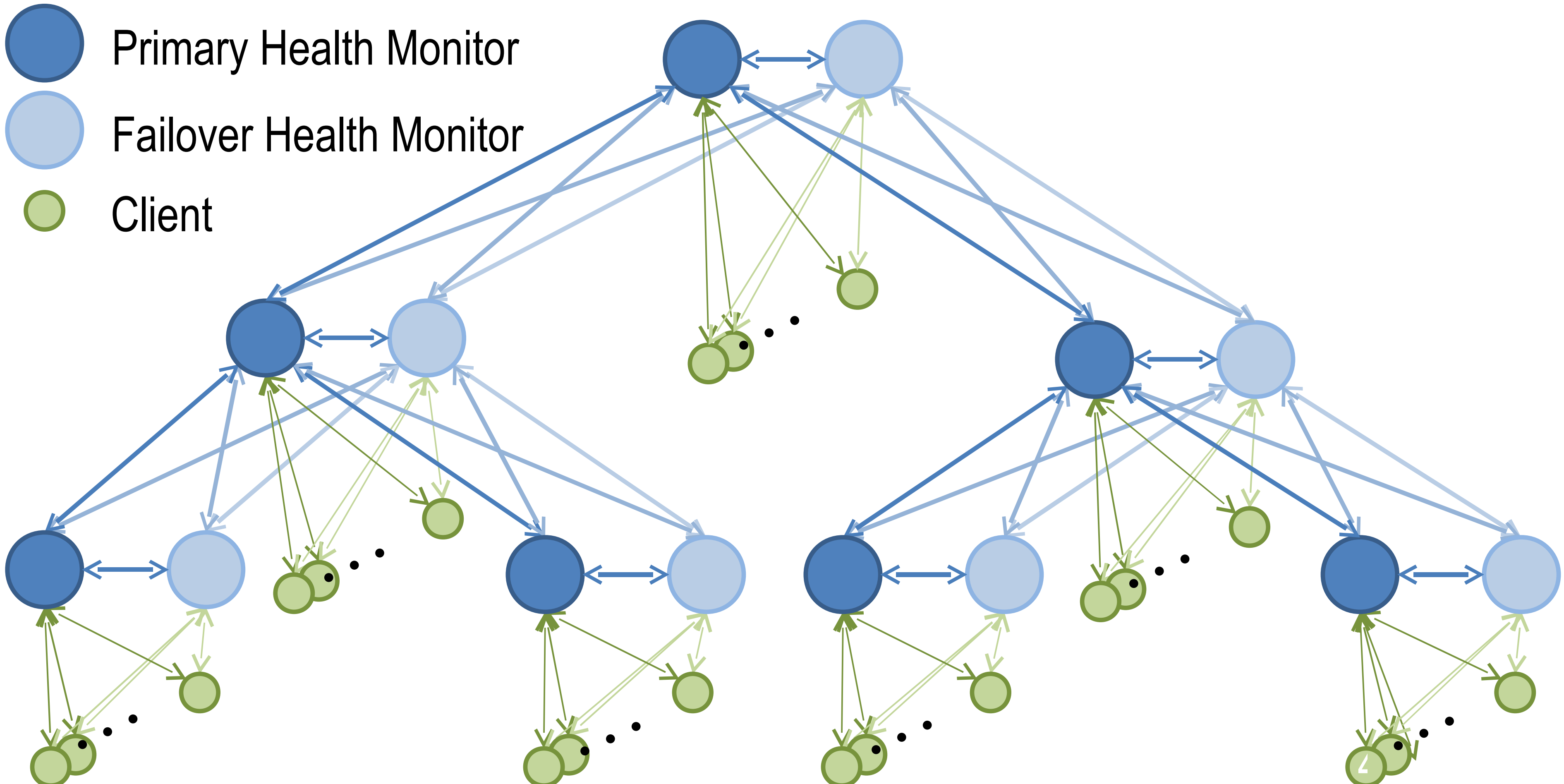
Scalable Health Network

- Burden of monitoring clients distributed – not replicated
- Fault-tolerant status reduction/broadcast network
  - Servers and LNET routers
- LNET high-priority small message support
  - Health network stays responsive
- Prompt, reliable detection
  - Time constants in seconds
  - Failed servers, clients and routers
  - Recovering servers and routers

Interface with existing RAS infrastructure

- Receive and deliver status notification

Health Monitoring Network



Operations support

Lustre HSM

- Interface from Lustre to hierarchical storage
  - > Initially HPSS
  - > SAM/QFS soon afterward

Tiered storage

- Combine HSM support with ZFS's SSD support and a policy manager to provide tiered storage management