# High Performance I/O

Michael Kuhn

Scientific Computing
Department of Informatics
Universität Hamburg

2017-06-08


Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

# I/O Layers

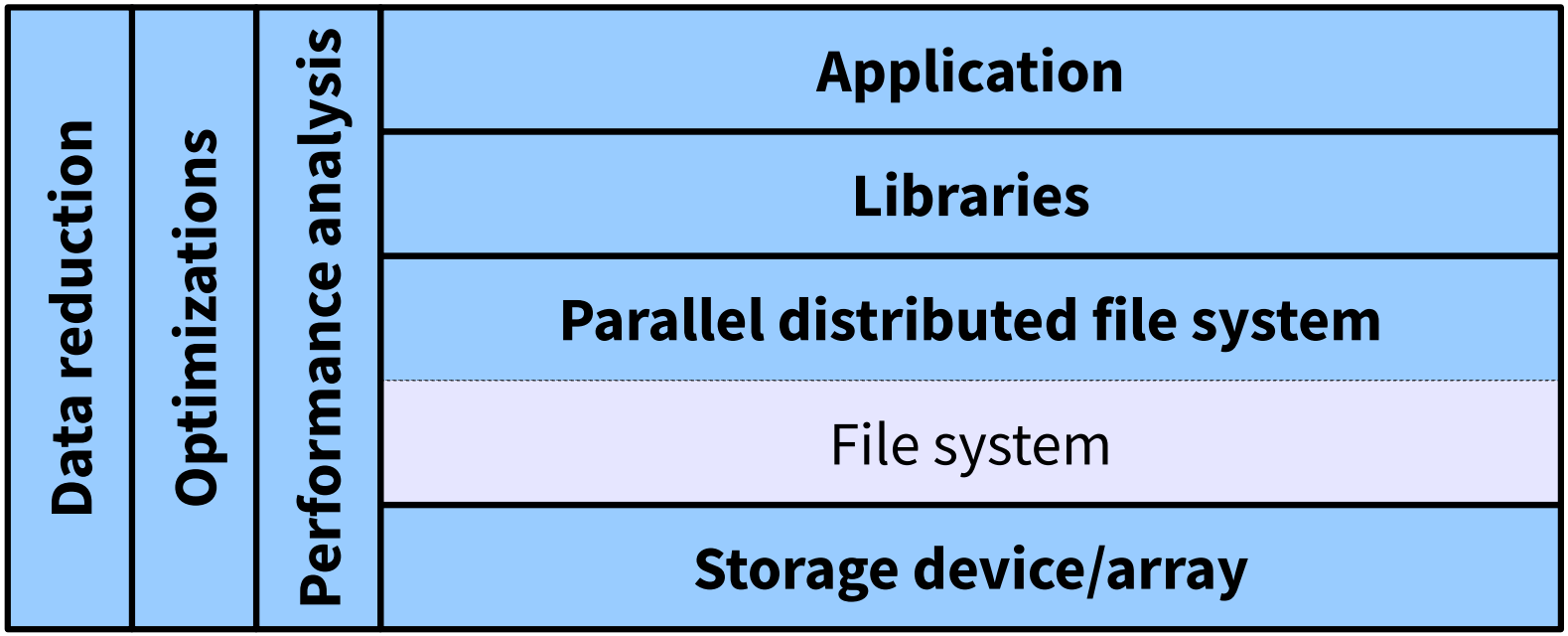



Figure: I/O layers and related topics

# Hard Disk Drives

- First HDD: 1956
  - IBM 350 RAMAC (3,75 MB, 8,8 KB/s, 1.200 RPM)
- HDD development
  - Capacity: factor of 100 every 10 years
  - Throughput: factor of 10 every 10 years

| Parameter | Started with | Developed to | Improvement |
|---|---|---|---|
| Capacity (formatted) | 3.75 megabytes[9] | eight terabytes | two-million-to-one |
| Physical volume | 68 cubic feet (1.9 $m^3$)[c][3] | 2.1 cubic inches (34 cc)[10] | 57,000-to-one |
| Weight | 2,000 pounds (910 kg)[3] | 2.2 ounces (62 g)[10] | 15,000-to-one |
| Average access time | about 600 milliseconds[3] | a few milliseconds | about 200-to-one |
| Price | US$9,200 per megabyte[11][*dubious – discuss*] | < $0.05 per gigabyte by 2013[12] | 180-million-to-one |
| Areal density | 2,000 bits per square inch[13] | 826 gigabits per square inch in 2014[14] | > 400-million-to-one |

Figure: HDD development [9]

# Solid-State Drives

- Benefits
  - Read throughput: factor of 2
  - Write throughput: factor of 3
  - Latency: factor of 100
  - Energy consumption: factor of 1–10
- Drawbacks
  - Price: factor of 10
  - Write cycles: 10.000–100.000

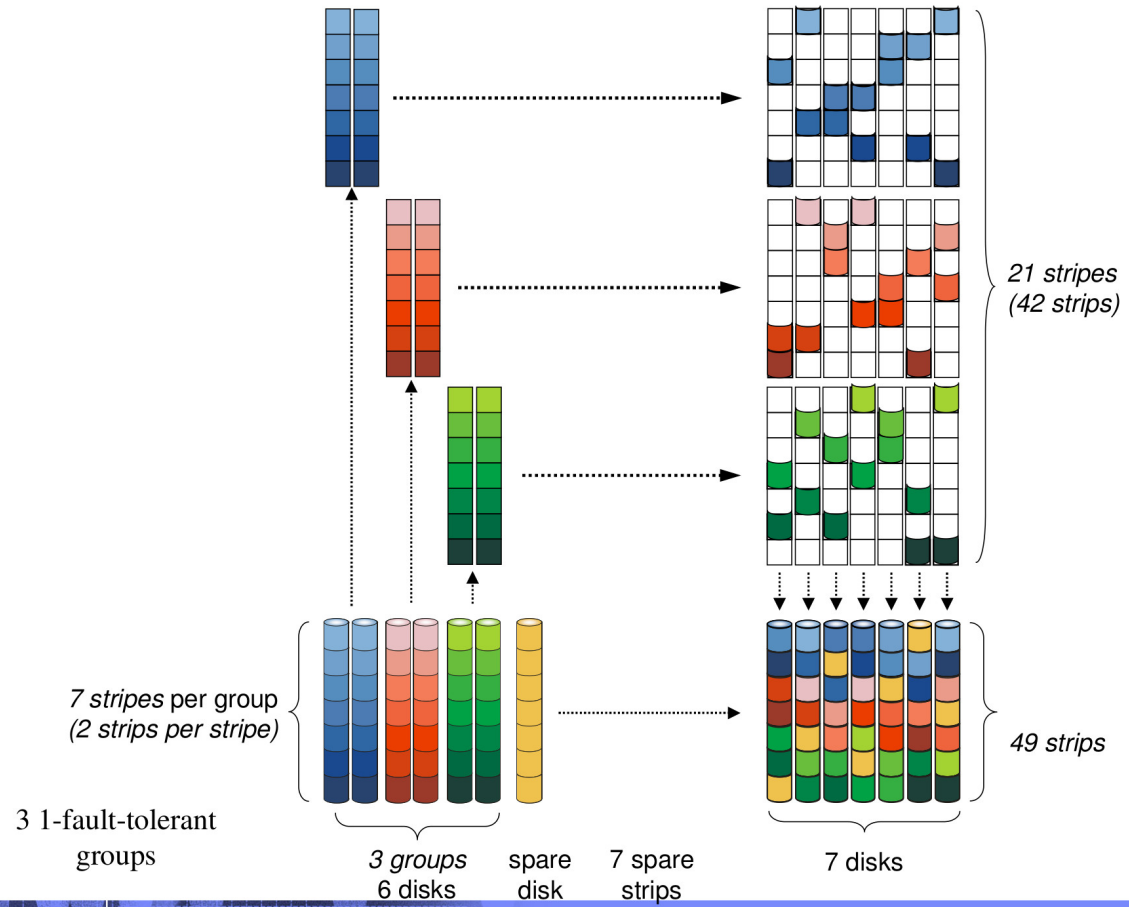
# RAID

- RAID
  - RAID 0: striping
  - RAID 1: mirroring
  - RAID 2/3: bit/byte striping
  - RAID 4: block striping
  - RAID 5/6: block striping
- Failures
  - HDDs usually have roughly the same age
  - Same batch
- Reconstruction
  - Read errors on other HDDs
  - Duration (30 min in 2004, 11 h in 2017)

# RAID... [7]



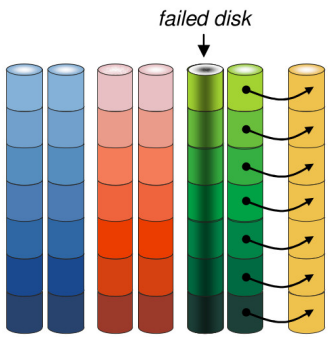
## Declustered RAID1 Example

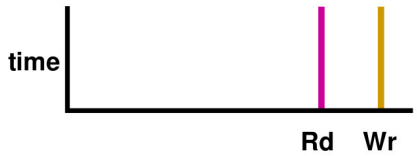


# RAID... [7]

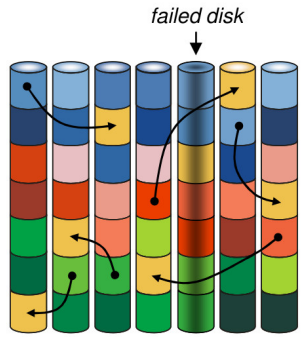


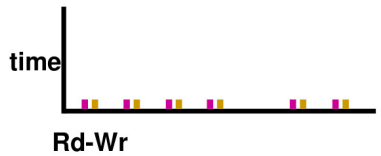
## Declustered RAID Rebuild Example – Single Fault




Rebuild activity confined to just a few disks – slow rebuild, disrupts user programs




Rebuild activity spread across many disks, faster rebuild or less disruption to user programs

# Performance Assessment

- Different performance criteria
  - Data throughput
    - Examples: photo/video editing, numerical applications
  - Request throughput
    - Examples: databases, metadata management
- Appropriate hardware
  - Data throughput
    - HDDs: 150–200 MB/s
    - SSDs: 500 MB/s
  - Request throughput
    - HDDs: 75–100 IOPS (7,200 RPM) to 175–210 IOPS (15,000 RPM)
    - SSDs: 8,600 IOPS (old) to 85,000–90,000 IOPS (current)
- Appropriate configuration
  - Small blocks for data, large blocks for requests
  - Partial block/page accesses can reduce performance

# Tasks

- Structure
  - Typically files and directories
  - Hierarchical organization
  - Other approaches: tagging
- Management of data and metadata
  - Block allocation
  - Access permissions, timestamps etc.
- File systems use underlying storage devices
  - Or a storage array
  - Logical Volume Manager (LVM) and/or mdadm

# I/O Interfaces

- Requests are realized throughput I/O interfaces
  - Forwarded to the file system
  - Different abstraction levels
- Low-level functionality: POSIX, MPI-IO, ...
- High-level functionality: HDF, NetCDF, ...

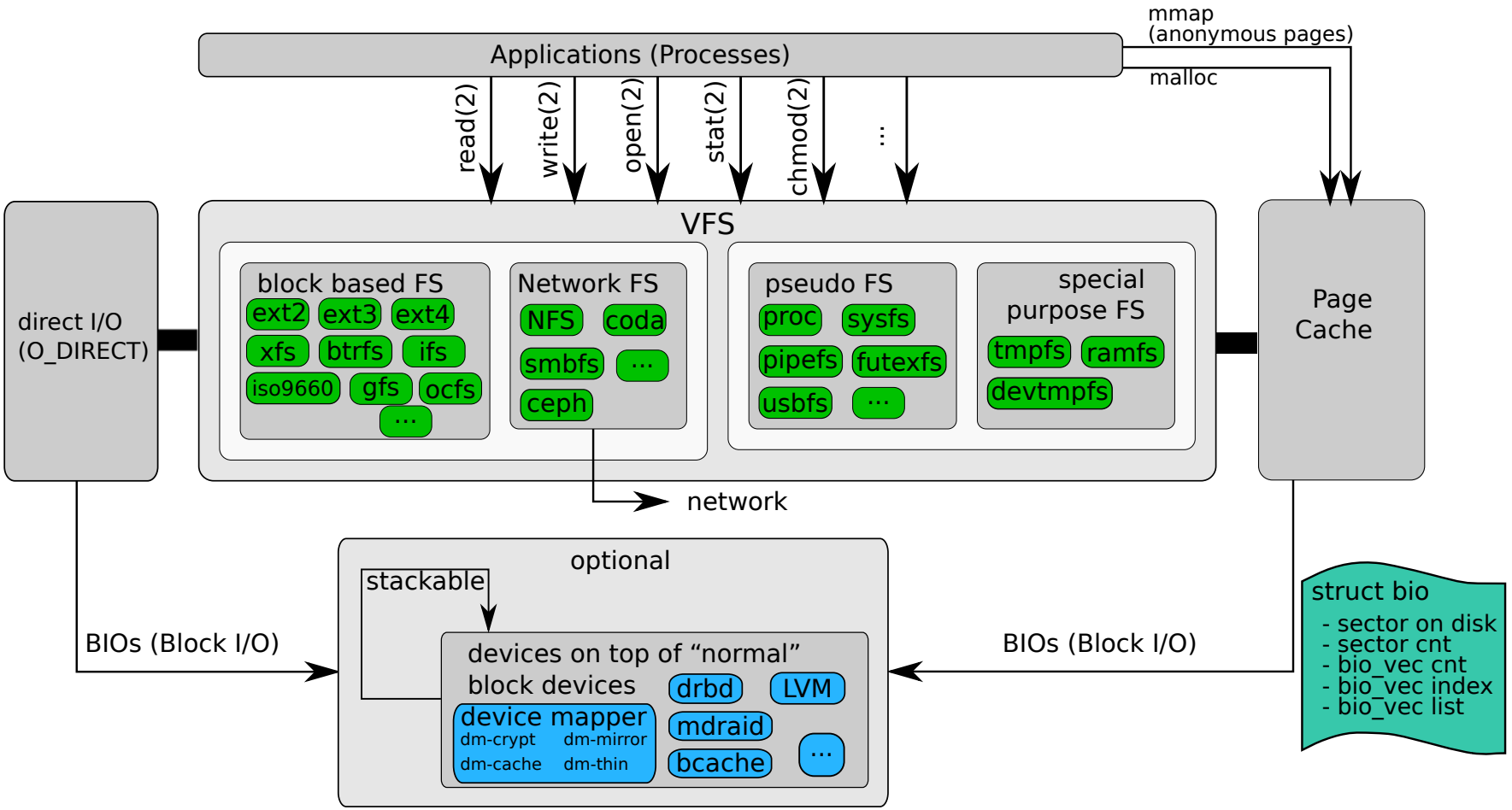
```
1 fd = open("/path/to/file", O_RDWR | O_CREAT | O_TRUNC,
      ↪ S_IRUSR | S_IWUSR);
2 nb = write(fd, data, sizeof(data));
3 rv = close(fd);
4 rv = unlink("/path/to/file");
```

- Initial access via path
  - Afterwards access via file descriptor (few exceptions)
- Functions are located in `libc`
  - Library executes system calls

# Virtual File System (Switch)

- Central file system component in the kernel
  - Sets file system structure and interface
- Forwards applications' requests based on mount point
- Enables supporting multiple different file systems
  - Applications are still portable due to POSIX
- POSIX: standardized interface for all file systems
  - Syntax
    - open, close, creat, read, write, lseek, chmod, chown, stat etc.
  - Semantics
    - `write`: *"POSIX requires that a read(2) which can be proved to occur after a write() has returned returns the new data. Note that not all filesystems are POSIX conforming."*

# Virtual File System (Switch)… [8]




The Linux Storage Stack Diagram
http://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram
Created by Werner Fischer and Georg Schönberger
License: CC-BY-SA 3.0, see http://creativecommons.org/licenses/by-sa/3.0/

# File System Objects

- User vs. system view
  - Users see files and directories
  - System manages inodes
    - Relevant for `stat` etc.
- Files
  - Contain data as byte arrays
  - Can be read and written (explicitly)
  - Can be mapped to memory (implicit)
- Directories
  - Contain files and directories
  - Structures the namespace

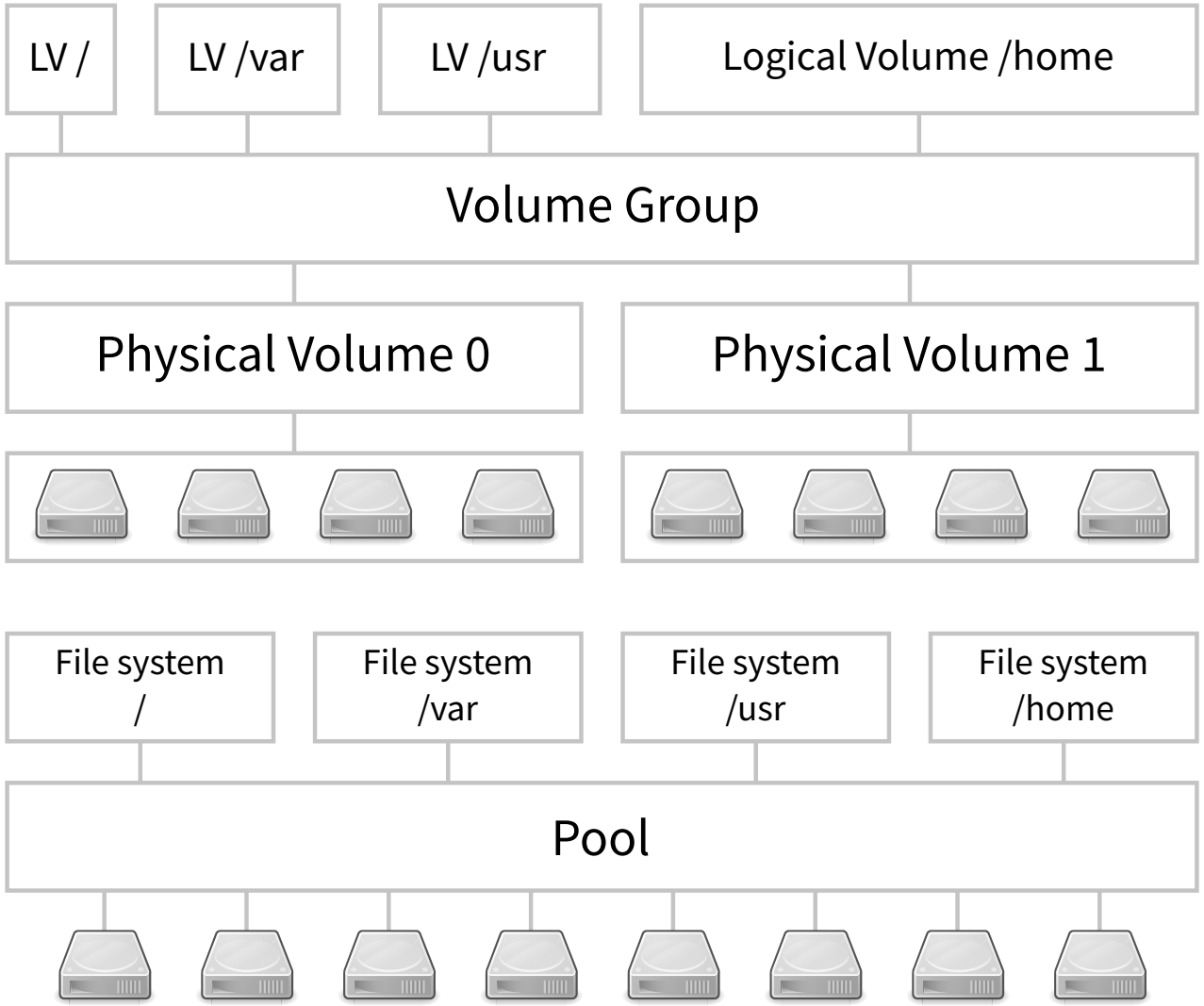
# Overview

- File system demands are growing
  - Data integrity, storage management, convenience functionality
- Error rate for SATA HDDs: 1 in $10^{14}$ to $10^{15}$ bits [6]
  - That is, one bit error per 12,5–125 TB
  - Additional bit errors in RAM, controller, cable, driver etc.
- Error rate can be problematic
  - Amount can be reached in daily use
  - Bit errors can occur in the superblock
- File system does not have knowledge about storage array
  - Knowledge is important for performance
  - For example, special options for ext4

# Pools

- Traditionally, one file system per partition
  - Volume manage enabled file system to span multiple partitions/devices
- Current file systems are static
  - Size changes are problematic
- New pool concept
  - Use hardware's full capacity and throughput
  - Keep file systems dynamic

# Pools…


LV /
LV /var
LV /usr
Logical Volume /home
Volume Group
Physical Volume 0
Physical Volume 1
File system /
File system /var
File system /usr
File system /home
Pool

# Copy on Write

- Blocks are not overwritten
  - Traditionally, blocks are modified in-place
  - If the system crashed in the meantime, data can be inconsistent
- All blocks are copied first
  - Copies are modified and written
  - Actually Redirect on Write, typically called Copy on Write
- Changes are done outside the live file system
  - If the system crashes, changes are not visible and discarded
- New blocks are integrated atomically

# Copy on Write…

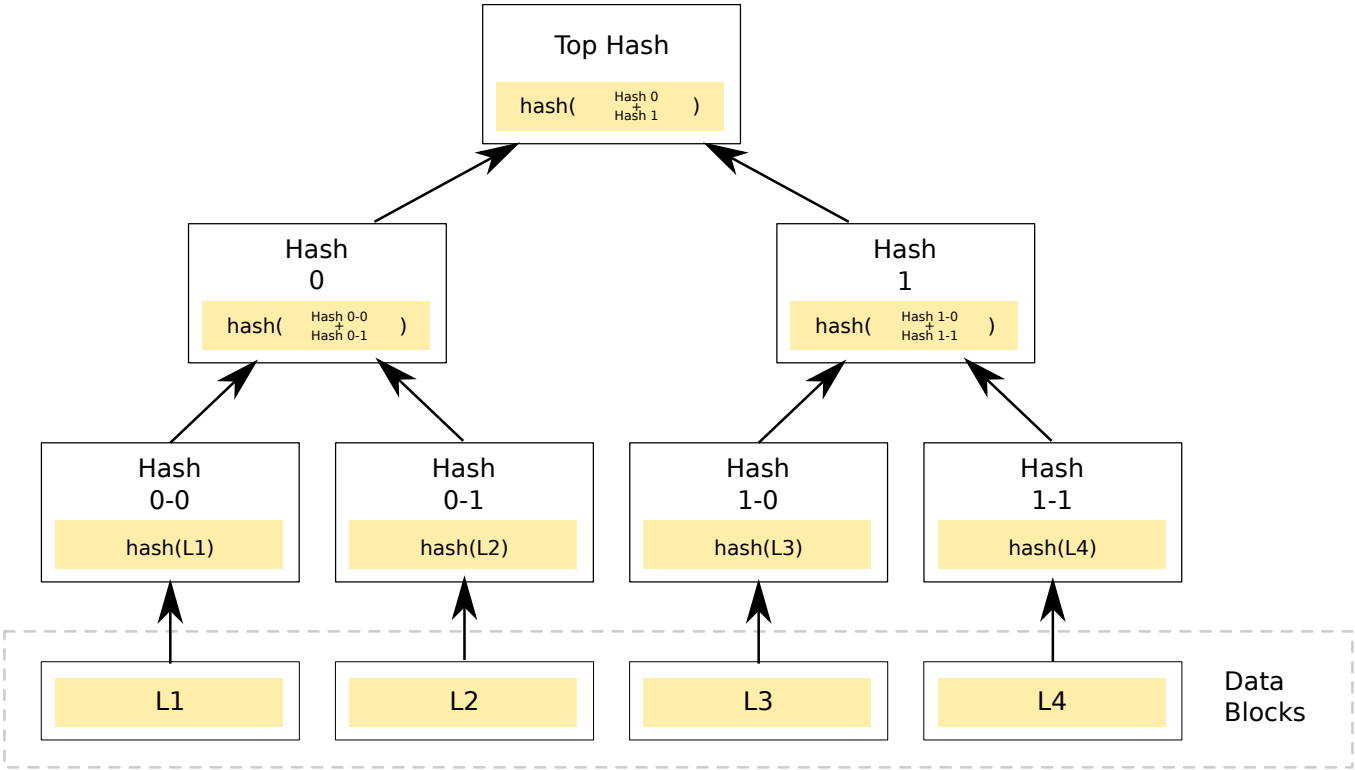



Figure: Copy on Write [10]

1. Initial state

# Copy on Write…

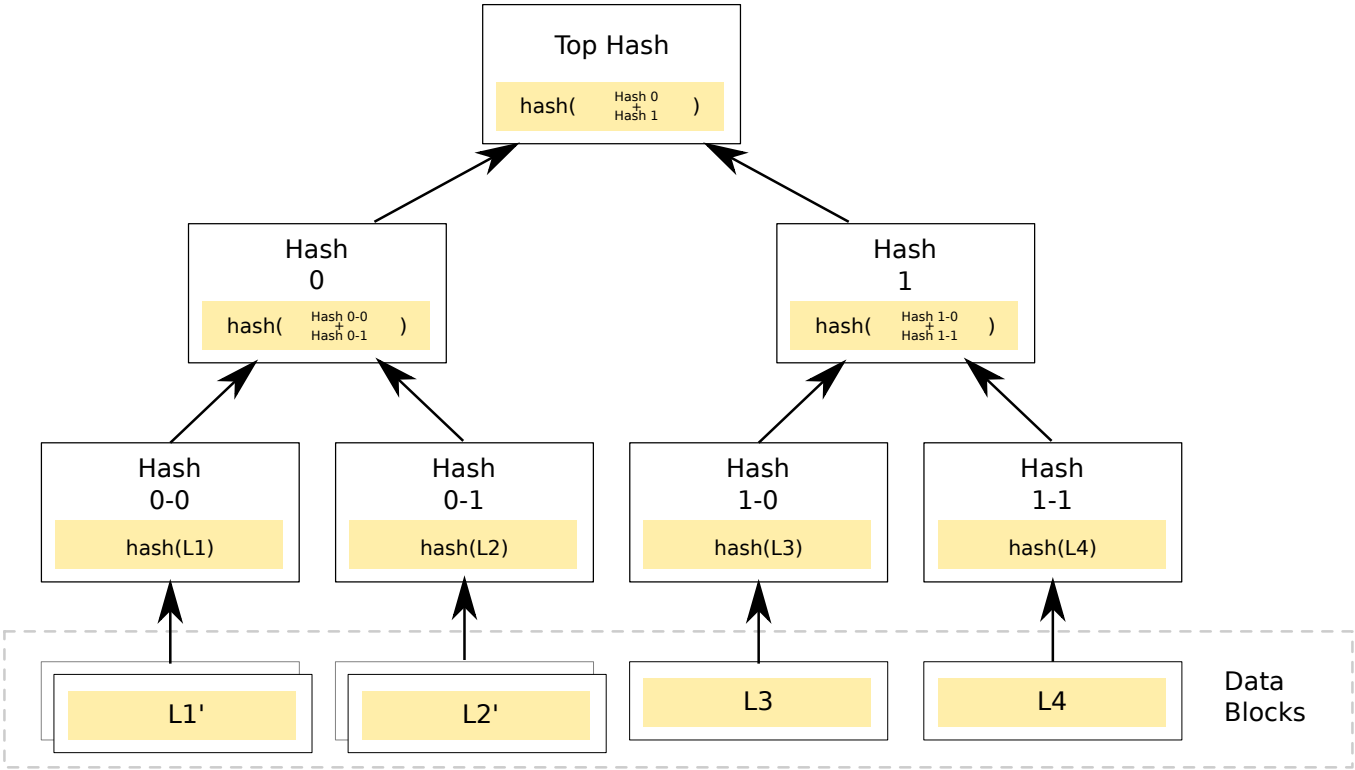



Figure: Copy on Write [10]

2 New blocks are allocated and modified

# Copy on Write…

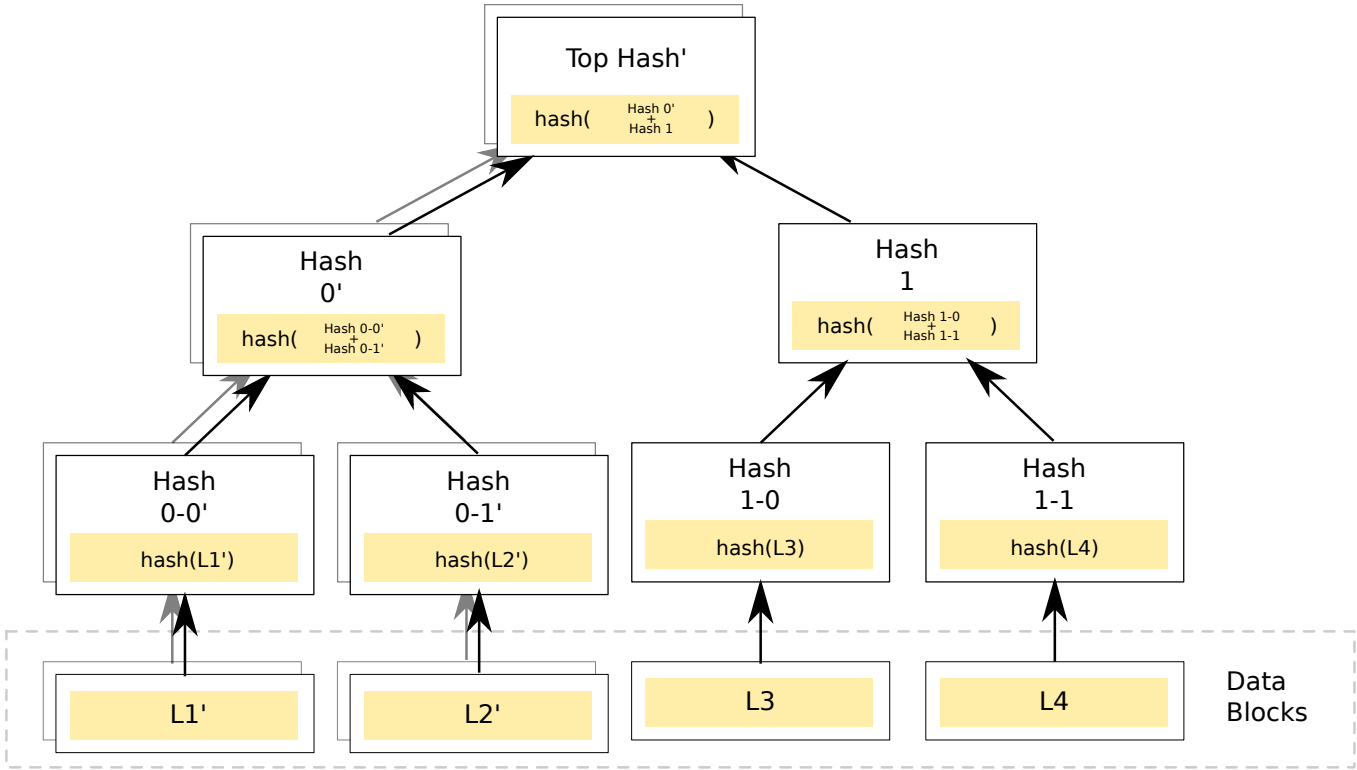



Figure: Copy on Write [10]

3 New block pointers are allocated and set

# Copy on Write…

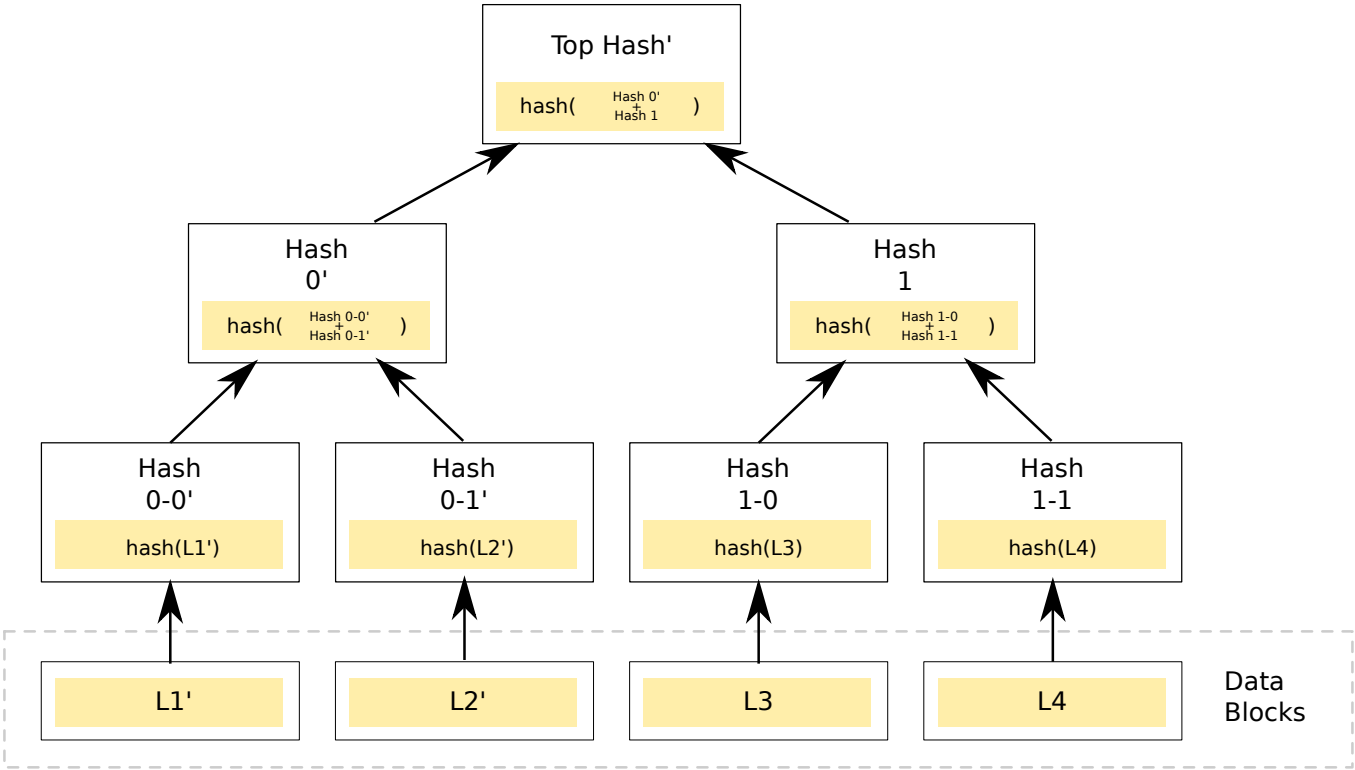
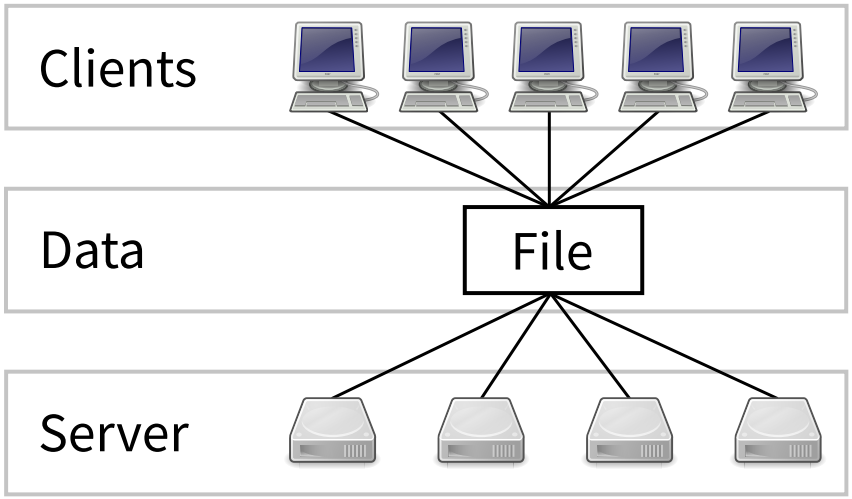



Figure: Copy on Write [10]

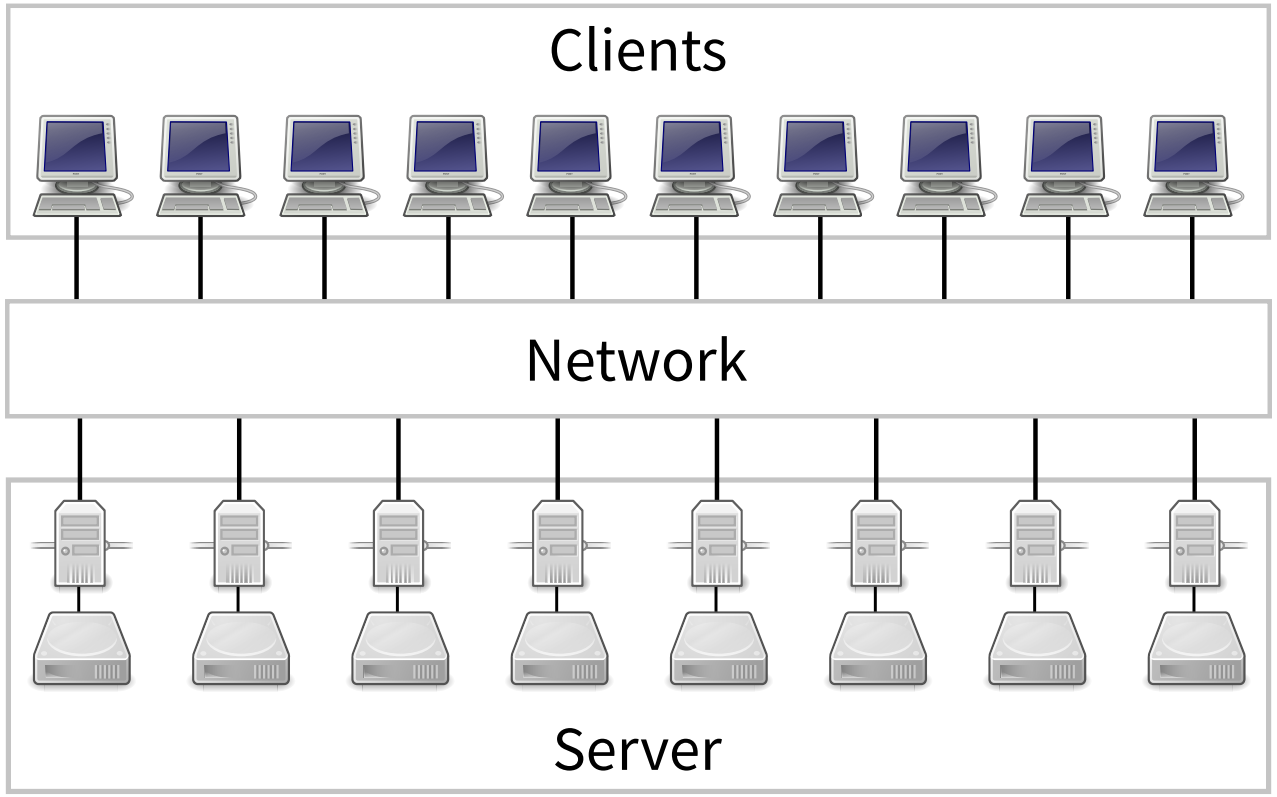
4 Uberlock is updated

# Definition

- Parallel file systems
  - Allow parallel access to shared resources
  - Access should be as efficient as possible
- Distributed file systems
  - Data and metadata is distributed across multiple servers
  - Single servers do not have a complete view
- Naming is inconsistent
  - Often just “parallel file system” or “cluster file system”



# Architecture

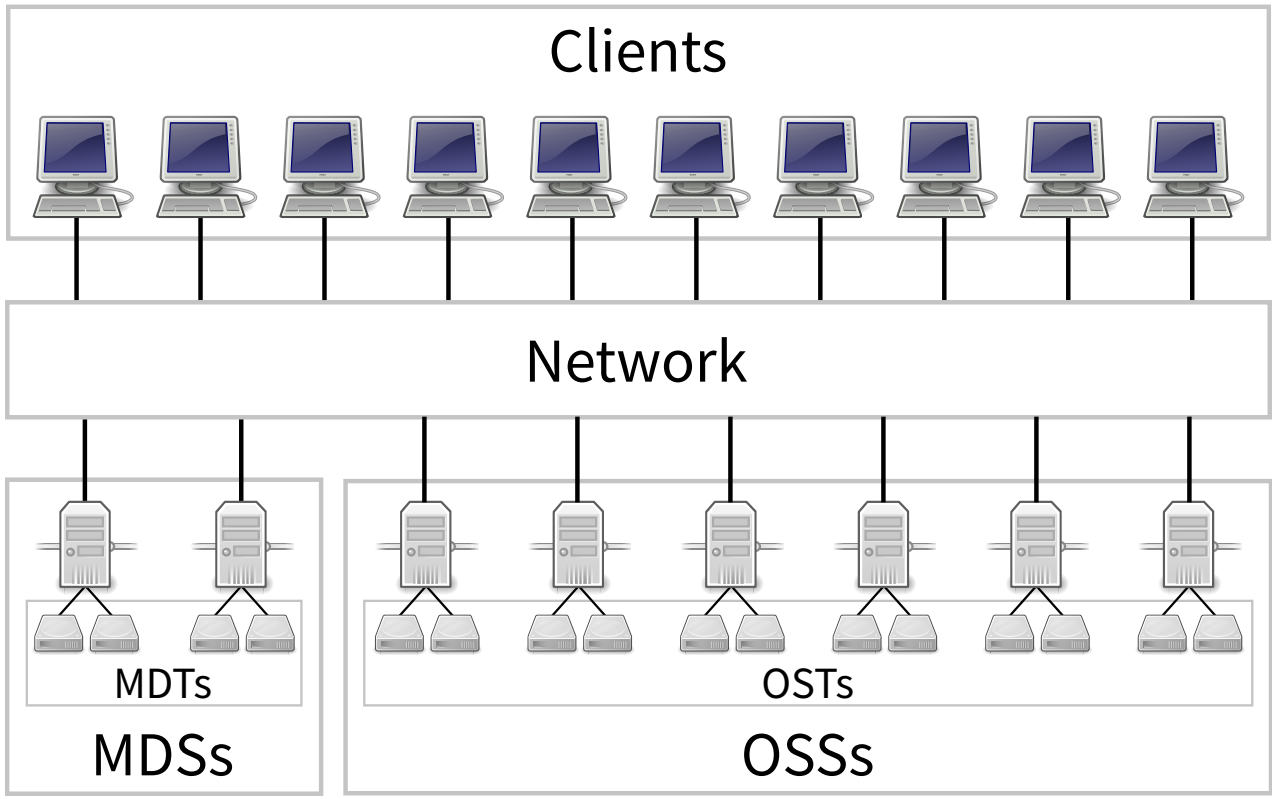



- Access to file system via I/O interface
  - Typically standardized, frequently POSIX
- Interface consists of syntax and semantics
  - Syntax defines operations, semantics defines behavior

# Semantics

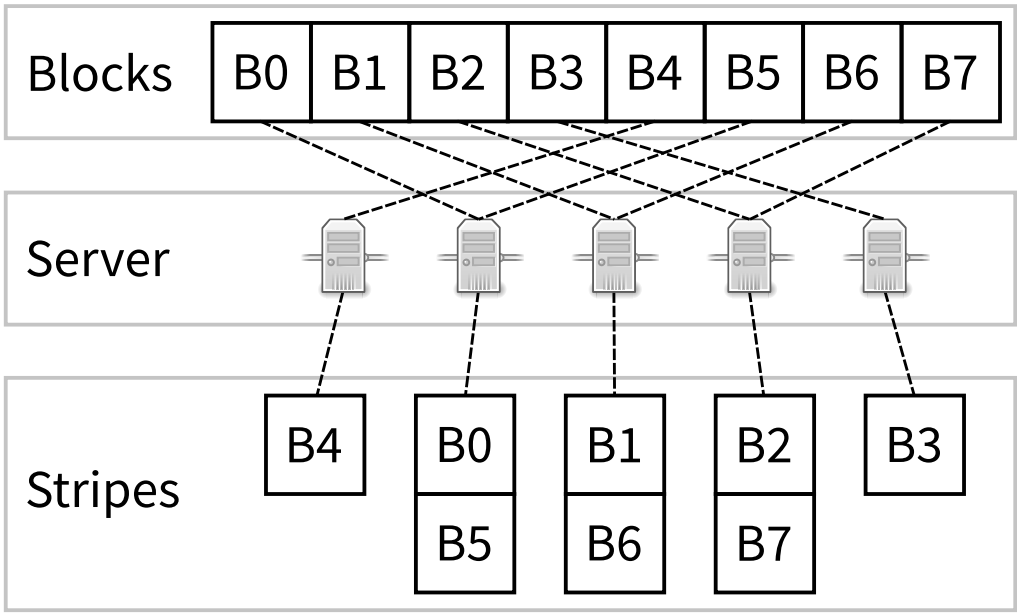
- POSIX has strong consistency/coherence requirements
  - Changes have to be visible globally after `write`
  - I/O should be atomic
- POSIX for local file systems
  - Requirements easy to support due to VFS
- Contrast: Network File System (NFS)
  - Same syntax, different semantics
- Session semantics
  - Changes only visible to other clients after session end
  - `close` writes changes and returns potential errors
- Later: MPI-IO
  - Less strict for higher scalability

# Architecture




- Separate servers for data and metadata
  - Produce different access patterns

# Architecture…




- File is split into blocks, distributed across servers
  - In this case, with a round-robin distribution
- Distribution does not have to start at first server

# Performance

- Parallel distributed file systems enable massive high performance storage systems
- Blizzard (DKRZ, GPFS)
  - Capacity: 7 PB
  - Throughput: 30 GB/s
- Mistral (DKRZ, Lustre)
  - Capacity: 54 PiB
  - Throughput: 450 GB/s (5.9 GB/s per node)
  - IOPS: 80,000 operations/s (phase 1 only)
- Titan (ORNL, Lustre)
  - Capacity: 40 PB
  - Throughput: 1.4 TB/s

# Overview

- POSIX and MPI-IO can be used for parallel I/O
  - Both interfaces are not very convenient for developers
- Problems
  - Exchangeability of data, complex programming, performance
- Libraries offer additional functionality
  - Self-describing data, internal structuring, abstract I/O
- Alleviating existing problems
  - SIONlib (performance)
  - NetCDF, HDF (exchangeability)
  - ADIOS (abstract I/O)

# SIONlib

- Mainly exists to circumvent deficiencies in existing file systems
  - On the one hand, problems with many files
    - Low metadata performance but high data performance
  - On the other hand, shared file access also problematic
    - POSIX requires locks, access pattern very important
- Offers efficient access to process-local files
  - Accesses are mapped to one or a few physical files
  - Aligned to file system blocks/stripes
- Backwards-compatible and convenient to use
  - Wrappers for `fread` and `fwrite`
  - Opening and closing via special functions

# NetCDF

- Developed by Unidata Program Center
  - University Corporation for Atmospheric Research
- Mainly used for scientific applications
  - Especially in climate science, meteorology and oceanography
- Consists of libraries and data formats
  1. Classic format (CDF-1)
  2. Classic format with 64 bit offsets (CDF-2)
  3. NetCDF-4 format
- Data formats are open standards
  - Classic formats are international standards of the Open Geospatial Consortium

# HDF

- Consists of libraries and data formats
  - Allows managing self-describing data
- Current version is HDF5
  - HDF4 is still actively supported
- Problems with previous versions
  - Complex API, limitations regarding 32 bit addressing
- Used as base for many higher-level libraries
  - NetCDF-4 (climate science)
  - NeXus (neutron, x-ray and muon science)

# ADIOS

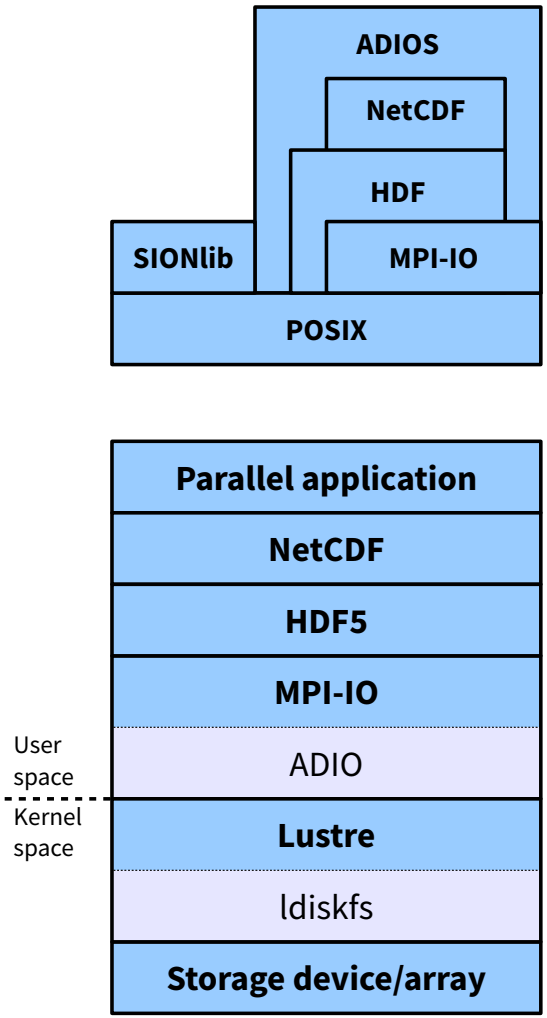
- ADIOS is heavily abstracted
  - No byte- or element-based access
  - Direct support for application data structures
- Designed for high performance
  - Mainly for scientific applications
  - Caching, aggregation, transformation etc.
- I/O configuration is specified via an XML file
  - Describes relevant data structures
  - Can be used to generate code automatically
- Developers specify I/O on a high abstraction level
  - No contact to middleware or file system

# ADIOS...

```
1 <adios-config host-language="C">
2   <adios-group name="checkpoint">
3     <var name="rows" type="integer"/>
4     <var name="columns" type="integer"/>
5     <var name="matrix" type="double"
          ↪ dimensions="rows,columns"/>
6   </adios-group>
7   <method group="checkpoint" method="MPI"/>
8   <buffer size-MB="100" allocate-time="now"/>
9 </adios-config>
```

- Data is combined in groups
- I/O methods can be specified per group
- Buffer sizes etc. can be configured

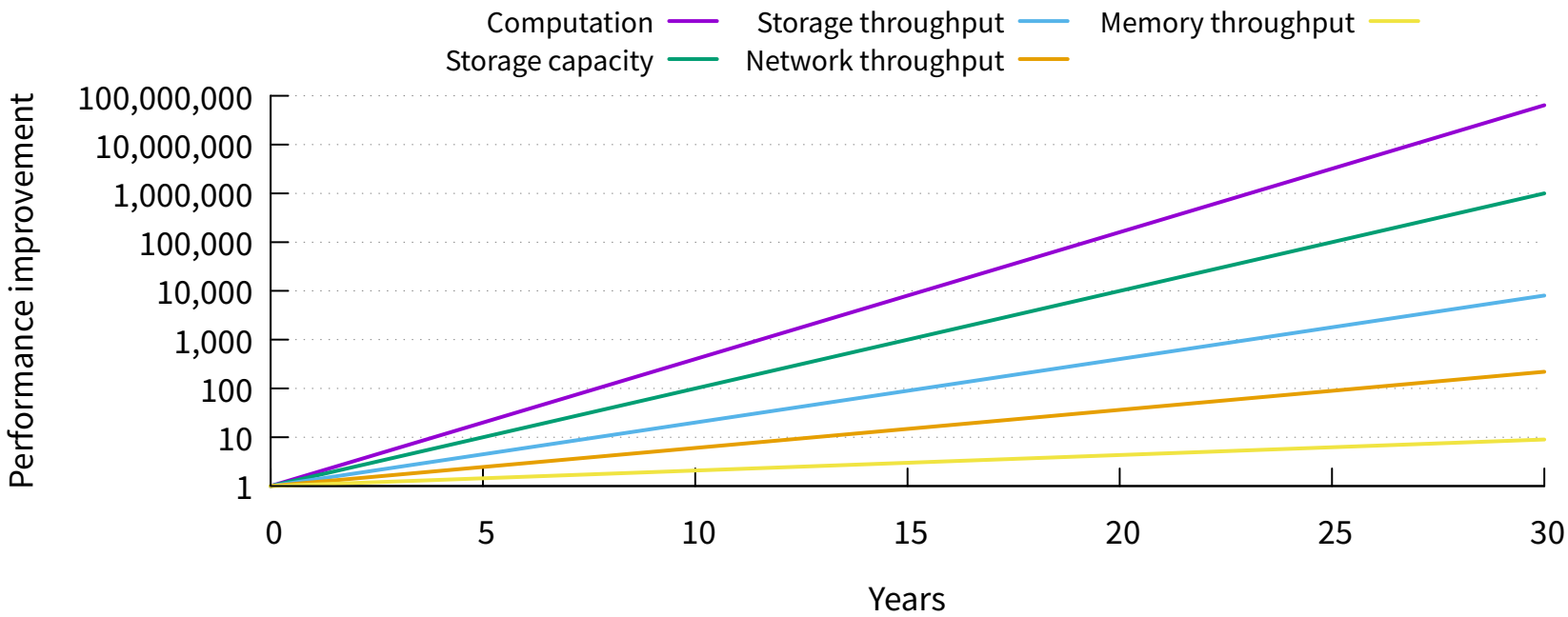
# Interaction…




- Data transformation
  - Transport through all layers
  - Loss of information
- Complex interaction
  - Optimizations and workarounds on all layers
  - Information about other layers
  - Analysis is complex
- Convenience vs. performance
  - Structured data in application
  - Byte stream in POSIX

# Development of Computation and Storage

- Capacity and performance continue to improve exponentially
  - Components improve at different rates
- I/O is becoming increasingly problematic
  - Data can be produced faster than it can be stored
- Storage hardware is a significant portion of TCO
  - DKRZ: approximately 20 % of overall costs (€ 6,000,000)



# Example: DKRZ

| | 2009 | 2015 | Factor |
|---|---|---|---|
| Performance | 150 TF/s | 3 PF/s | 20x |
| Node count | 264 | 2,500 | 9.5x |
| Node performance | 0.6 TF/s | 1.2 TF/s | 2x |
| Main memory | 20 TB | 170 TB | 8.5x |
| Storage capacity | 5.6 PB | 45 PB | **8x** |
| Storage throughput | 30 GB/s | 400 GB/s | 13.3x |
| HDD count | 7,200 | 8,500 | 1.2x |
| Archive capacity | 53 PB | 335 PB | **6.3x** |
| Archive throughput | 9.6 GB/s | 21 GB/s | **2.2x** |
| Power consumption | 1.6 MW | 1.4 MW | 0.9x |
| Acquisition costs | € 30 M | € 30 M | 1x |

# Data Reduction Techniques

- Recomputation of results
  - Instead of storing all results, recompute them on demand
  - Data has to be analyzed in situ, reproducibility is problematic
- Deduplication
  - Data is split into blocks, redundant blocks reference original
  - Requires additional main memory for tables
  - Studies show 20–30 % savings for HPC data
- Compression
  - Data can be compressed by the application or file system
  - Requires additional CPU
  - Studies show 30+ % savings for HPC data

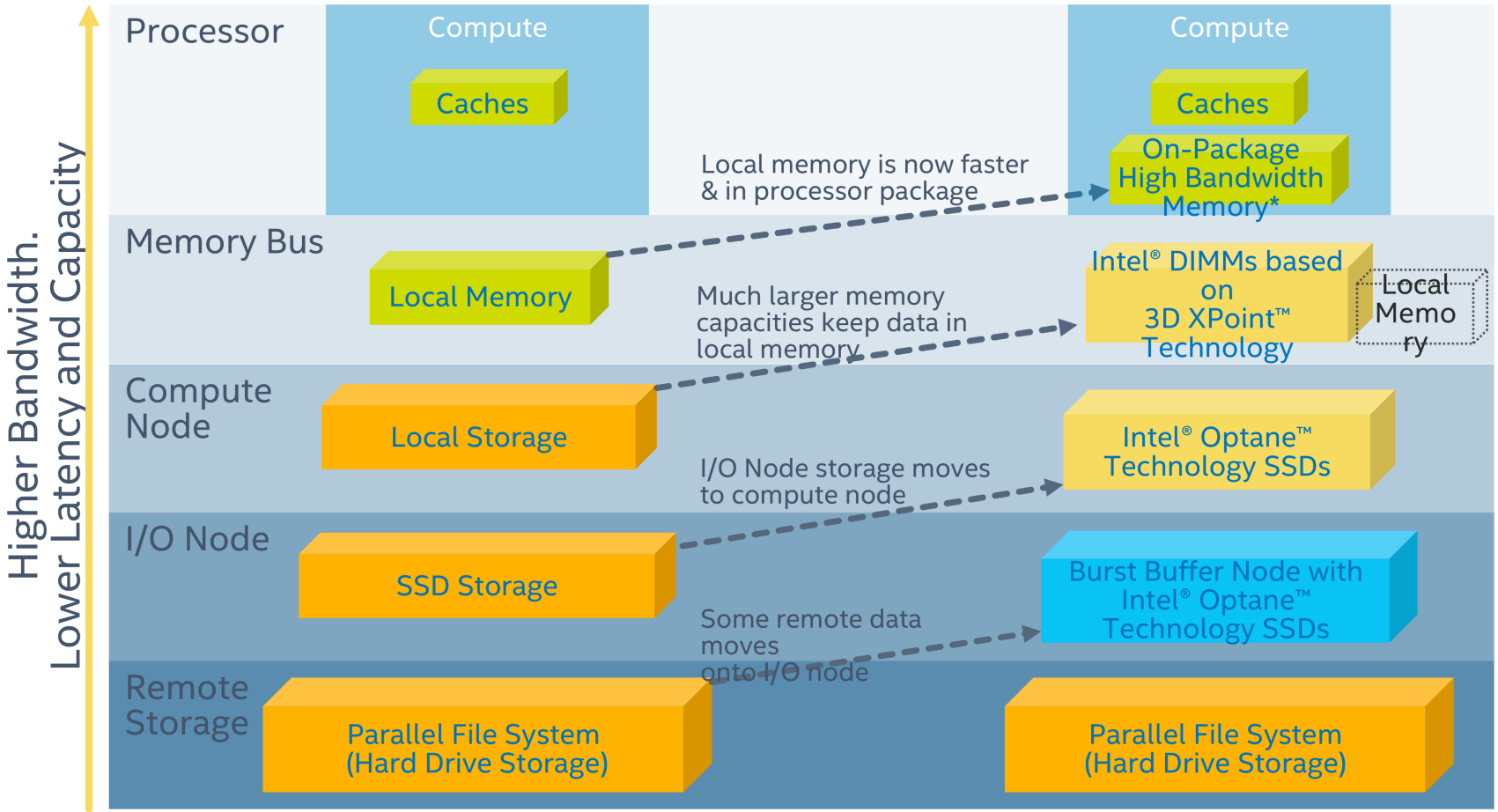
# Storage Hierarchy

- Current state
  - L1, L2, L3 cache, RAM, SSD, HDD, tape
- Latency gap from RAM to SSD
  - Huge performance loss if data is not in RAM
- Performance gap is worse on supercomputers
  - RAM is node-local, data is in parallel distributed file system
- New technologies to close gap
  - NVRAM, 3D XPoint etc.

| Level | Latency |
|---|---|
| L1 cache | ≈ 1 ns |
| L2 cache | ≈ 5 ns |
| L3 cache | ≈ 10 ns |
| RAM | ≈ 100 ns |
| NVRAM | ≈ 1.000 ns |
| 3D XPoint | ≈ 10.000 ns |
| SSD | ≈ 100.000 ns |
| HDD | ≈ 10.000.000 ns |
| Tape | ≈ 50.000.000.000 ns |

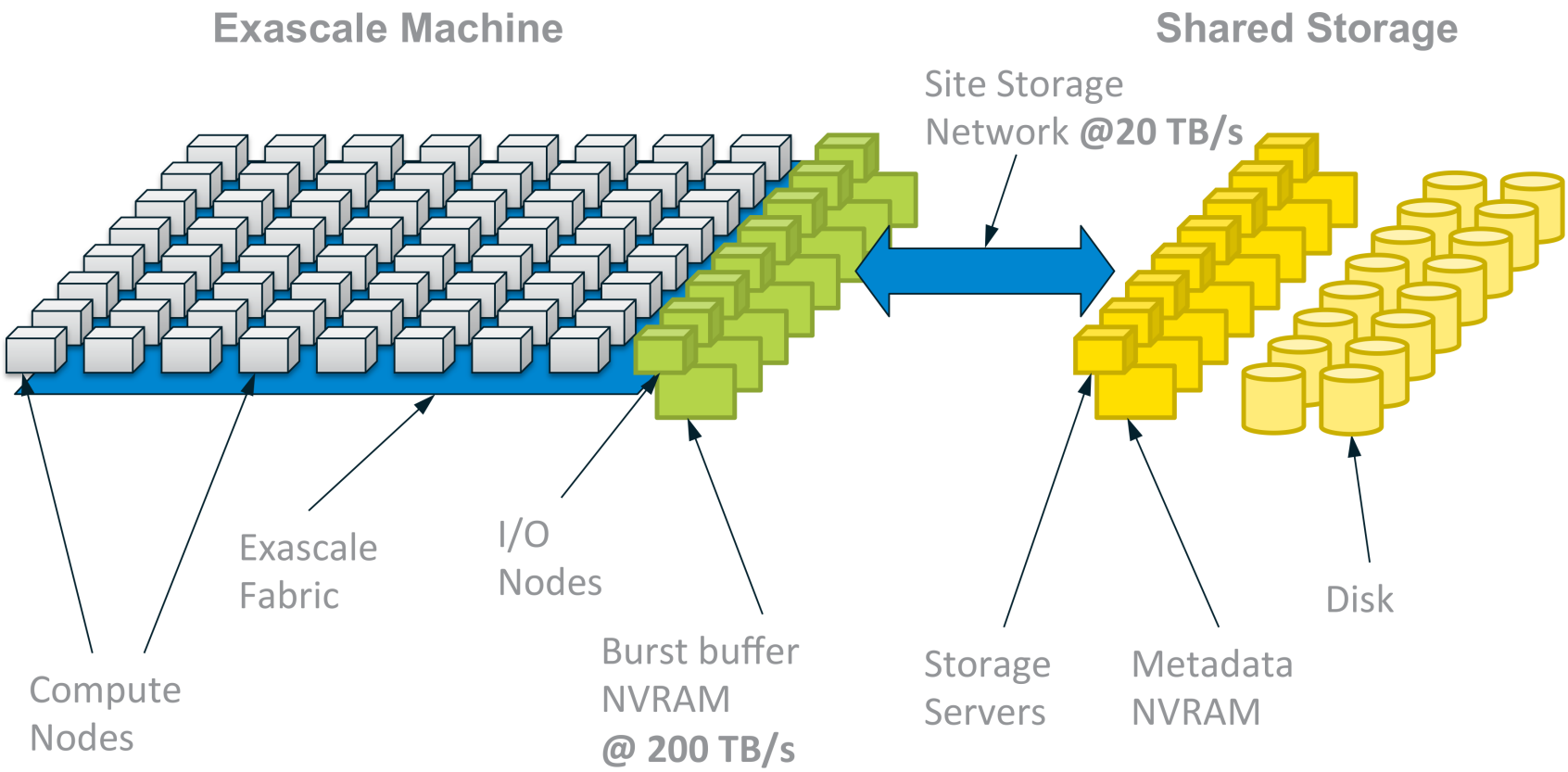
Table: Latencies [4, 3]

# Storage Hierarchy… [1]



# Storage Hierarchy… [2]

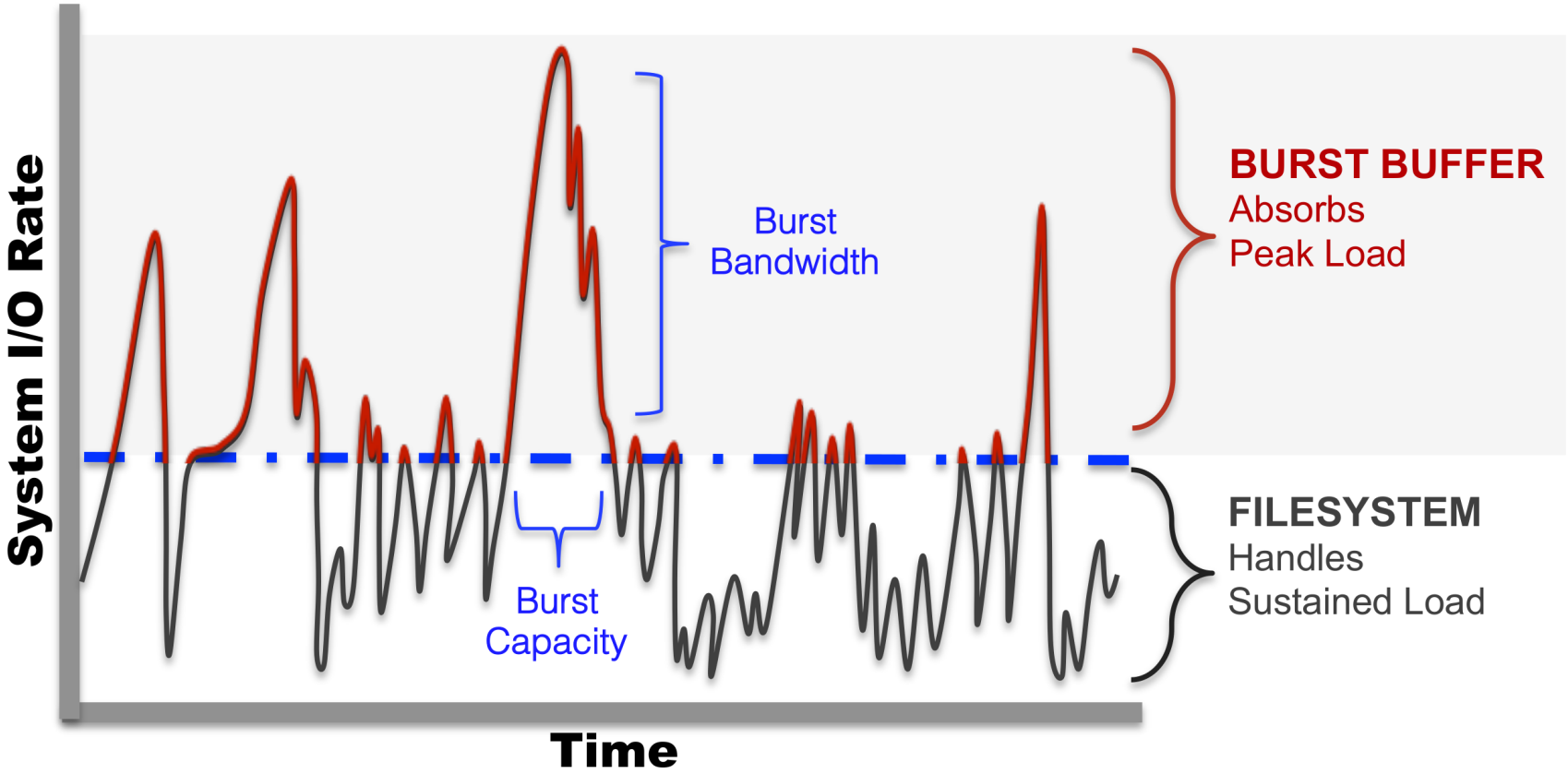
- I/O nodes with burst buffers close to compute nodes
- Slower storage network to file system servers



# Burst Buffers [5]

*Analysis of a major HPC production storage system*

- 99% of the time, storage BW utilization < 33% of max
- 70% of the time, storage BW utilization < 5% of max

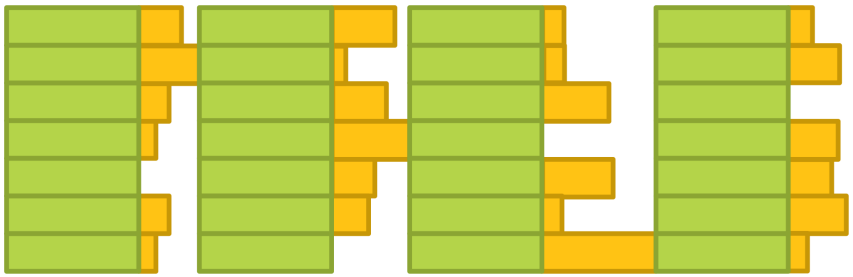


# DAOS

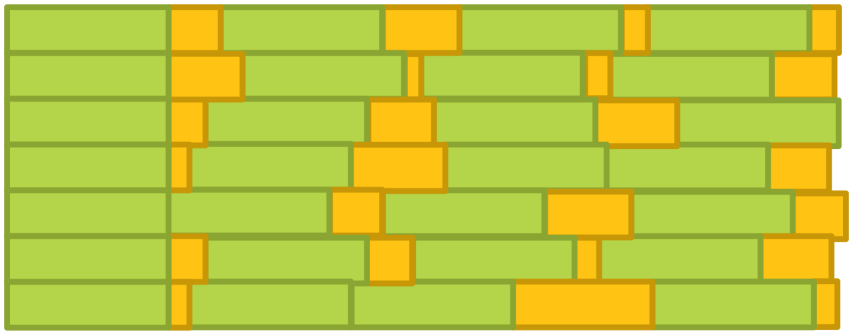
- New holistic approach for I/O
  - Distributed Application Object Storage (DAOS)
- Supports multiple storage models
  - Container: a set of shards
  - Shard: a set of objects
  - Object: key-value store
- Support for versioning
  - Operations are executed in transactions
  - Transactions are persisted as epochs
- Make use of modern storage technologies
- Native support for HDF5
  - HDF data structures are mapped to DAOS objects

# DAOS… [2]

- I/O is typically performed synchronously
  - Applications have to wait for slowest process
  - Variations of I/O performance are normal



- I/O should be completely asynchronous
  - Eliminates waiting times
  - Difficult to define consistency



# Summary

- Achieving high performance I/O is a complex task
  - Many layers: storage devices, file systems, libraries etc.
- File systems organize data and metadata
  - Modern file systems provide additional functionality
- Parallel distributed file systems allow efficient access
  - Data is distributed across multiple servers
- I/O libraries facilitate ease of use
  - Exchangeability of data is an important factor
- New hardware technologies will alter the I/O stack
  - Systems will become even more complex

# References I

[1] Brent Gorda. HPC Storage Futures – A 5-Year Outlook. http://lustre.ornl.gov/ecosystem-2016/documents/keynotes/Gorda-Intel-keynote.pdf.

[2] Brent Gorda. HPC Technologies for Big Data. http://www.hpcadvisorycouncil.com/events/2013/Switzerland-Workshop/Presentations/Day_2/3_Intel.pdf.

[3] Jian Huang, Karsten Schwan, and Moinuddin K. Qureshi. Nvram-aware logging in transaction systems. *Proc. VLDB Endow.*, 8(4):389–400, December 2014.

[4] Jonas Bonér. Latency Numbers Every Programmer Should Know. https://gist.github.com/jboner/2841832.

# References II

[5] Mike Vildibill. Advanced IO Architectures. http://storageconference.us/2015/Presentations/Vildibill.pdf.

[6] Seagate. Desktop HDD. http://www.seagate.com/www-content/datasheets/pdfs/desktop-hdd-8tbDS1770-9-1603DE-de_DE.pdf.

[7] Veera Deenadhayalan. General Parallel File System (GPFS) Native RAID. https://www.usenix.org/legacy/events/lisa11/tech/slides/deenadhayalan.pdf.

[8] Werner Fischer and Georg Schönberger. Linux Storage Stack Diagramm. https://www.thomas-krenn.com/de/wiki/Linux_Storage_Stack_Diagramm.

# References III

[9] Wikipedia. Hard disk drive.
http://en.wikipedia.org/wiki/Hard_disk_drive.

[10] Wikipedia. Merkle tree.
http://en.wikipedia.org/wiki/Merkle_tree.