Compression in Lustre

Anna Fuchs

Department of Informatics
Research Group Scientific Computing
Universität Hamburg

25.09.2019

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Outline

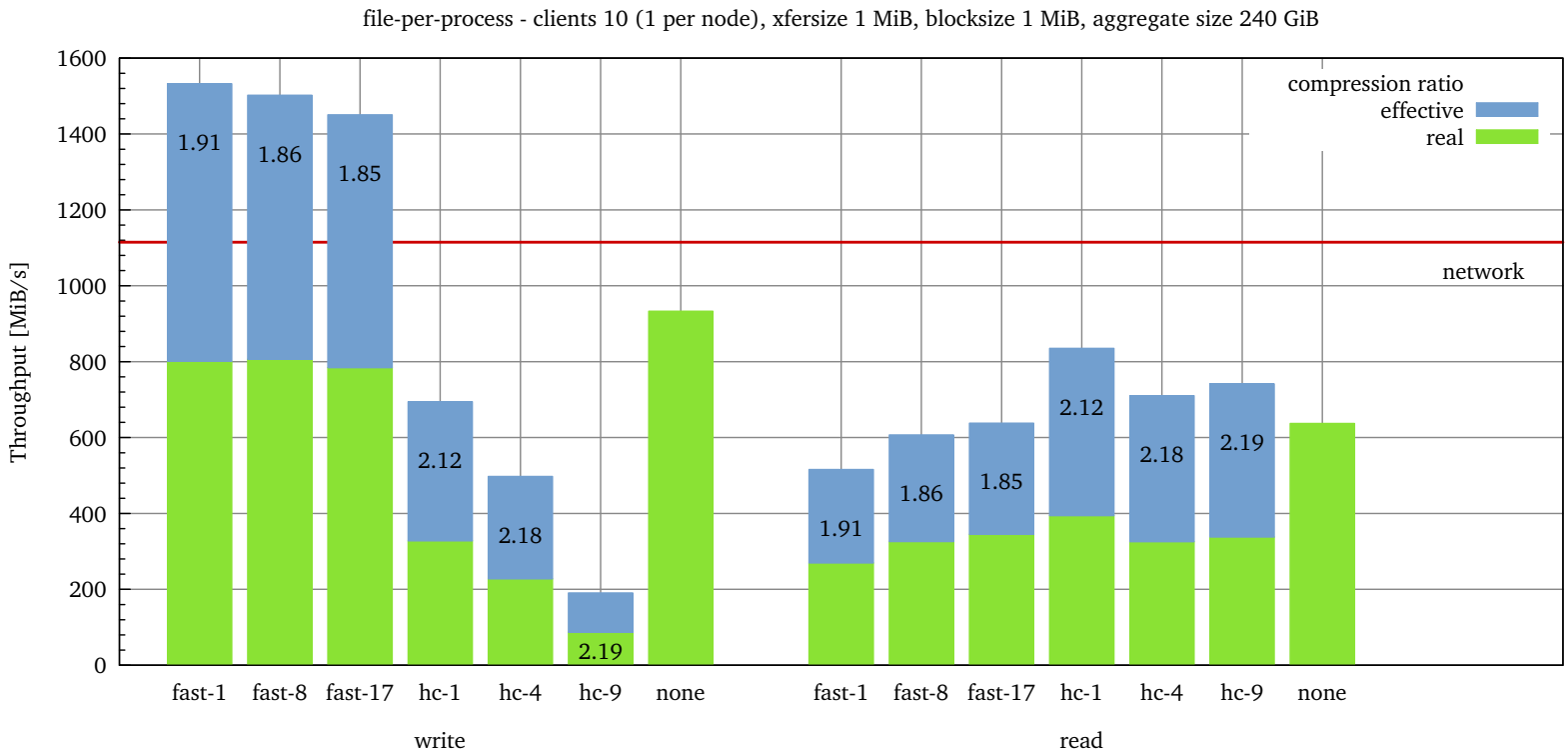
Project IPCC-L

- Intel Parallel Computing Center for Lustre (Universität Hamburg)
- Research project "Enhanced Adaptive Compression in Lustre"
- Funding - 2 years full time
- Scientific researcher, PhD-student

- Compensate I/O bottleneck with computation
- Compression can save costs
  - Higher network throughput
  - Better storage usage and more capacity
  - Lower power consumption

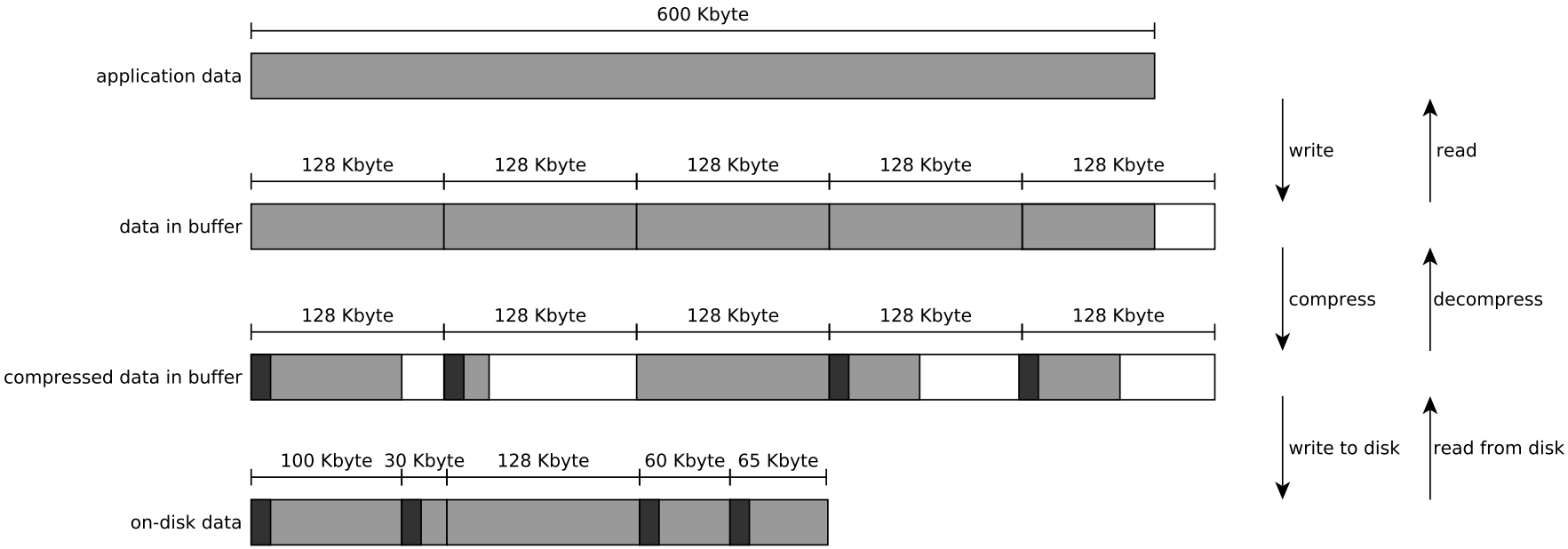
Preliminary work (2016)

- Userspace IOR, Ethernet
- 10 Lustre clients, 10 servers, 240 files each 1 GiB, 1 MiB stripes, ZFS, Lustre 2.8
- Single thread compression with LZ4

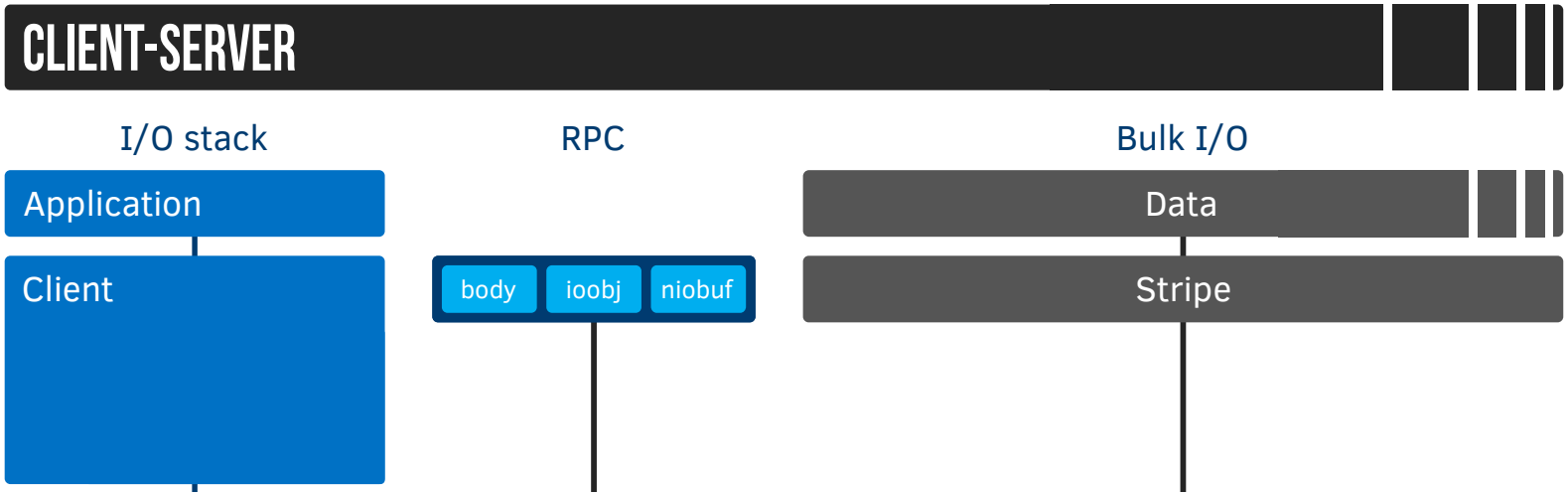


Overview

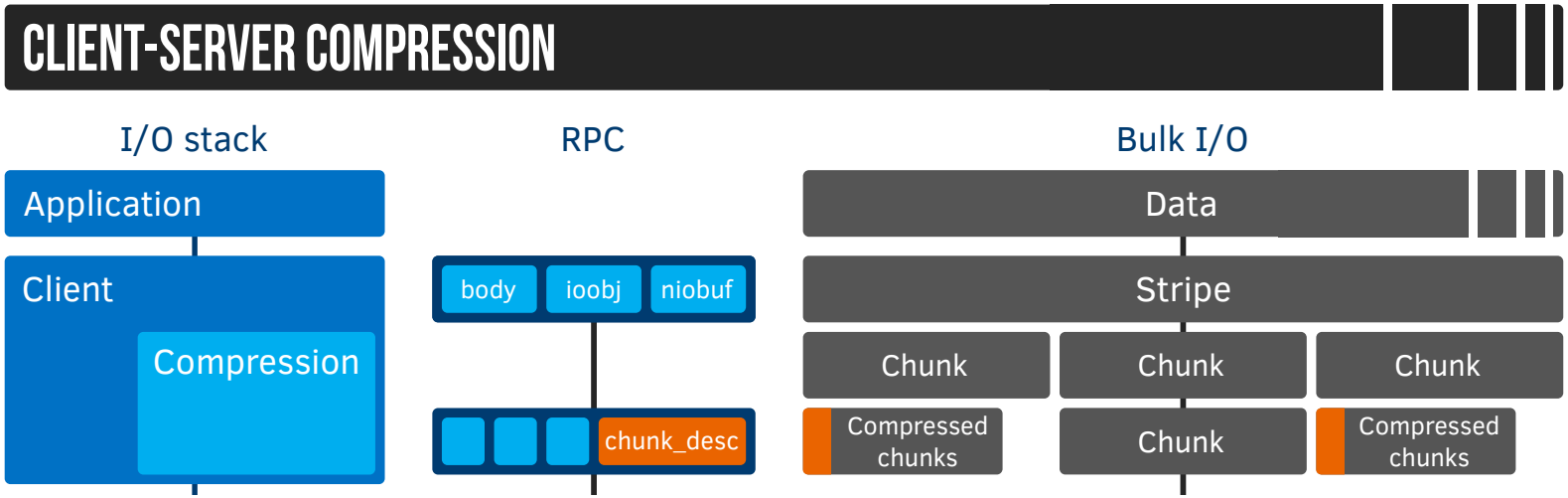
- Compression based on chunks
- Reduce RMW-issues
- Still good ratio with small blocks (down to 64KiB)
- ZFS backend



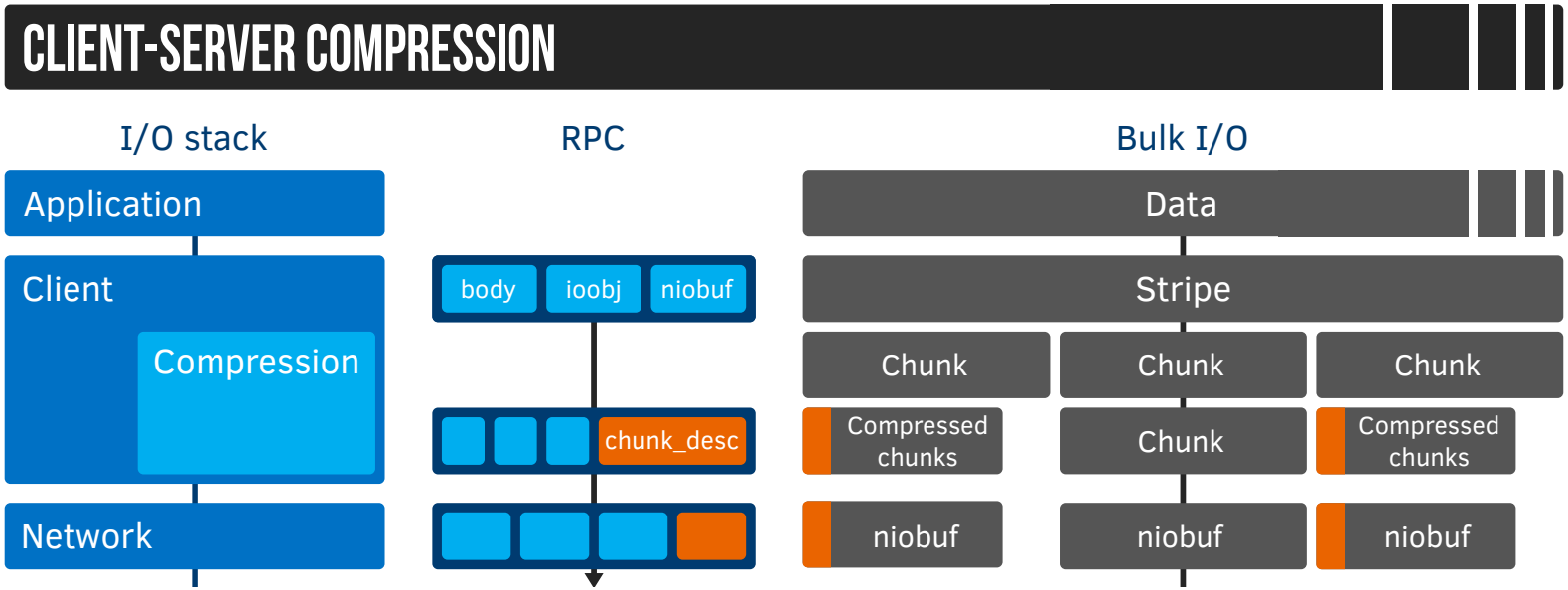
Architecture (write)



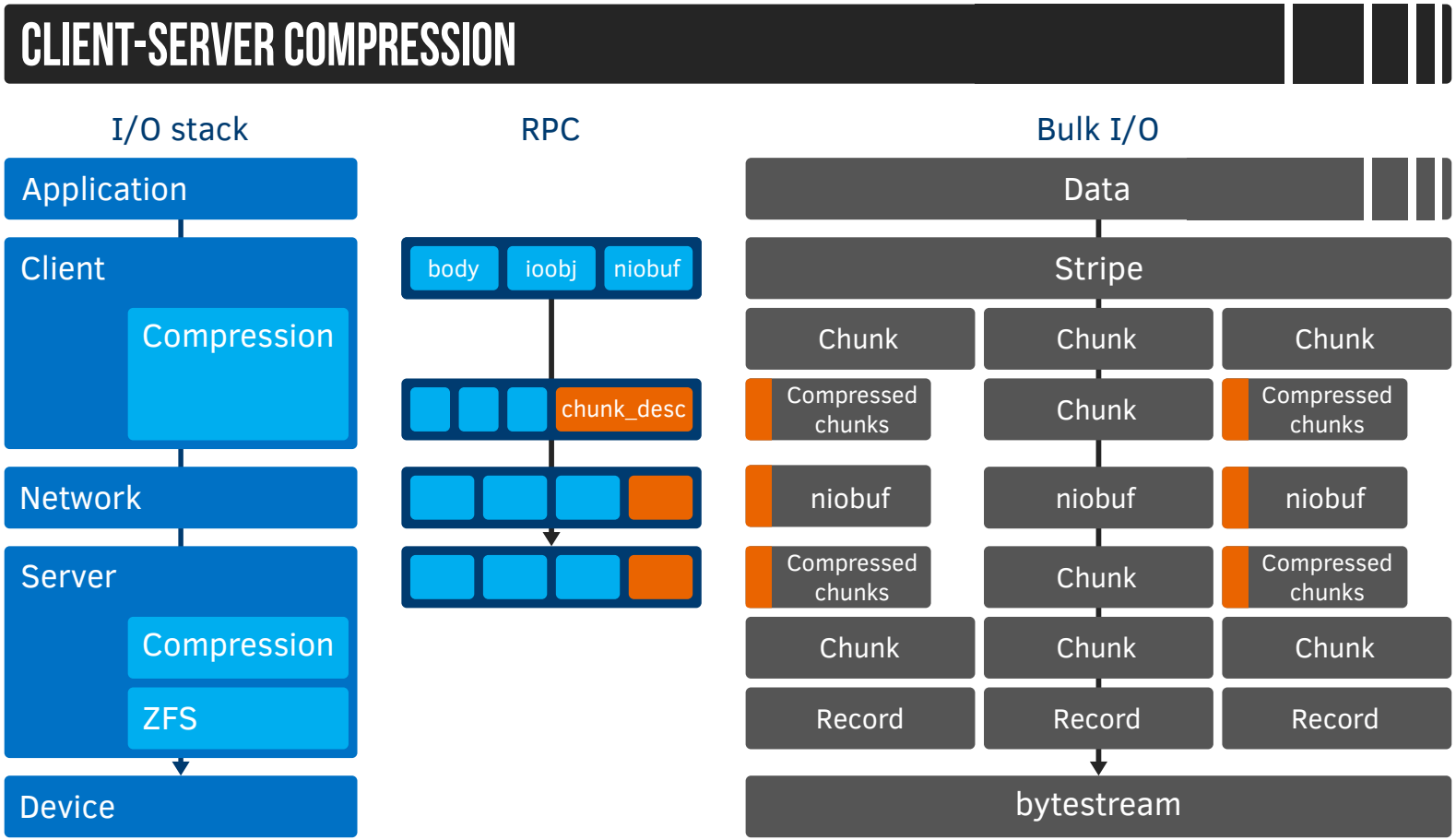
Architecture (write)



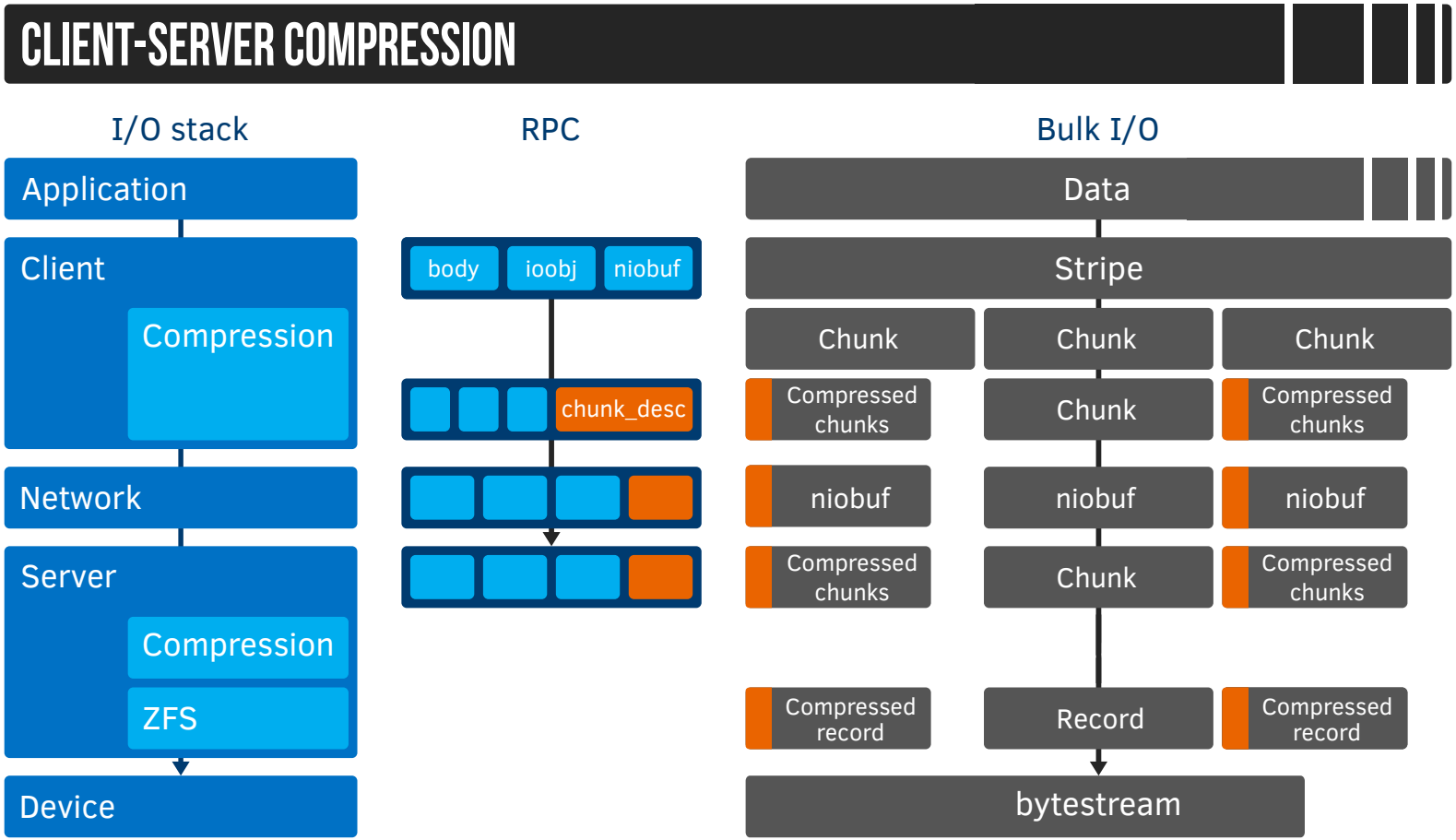
Architecture (write)



Architecture (write)

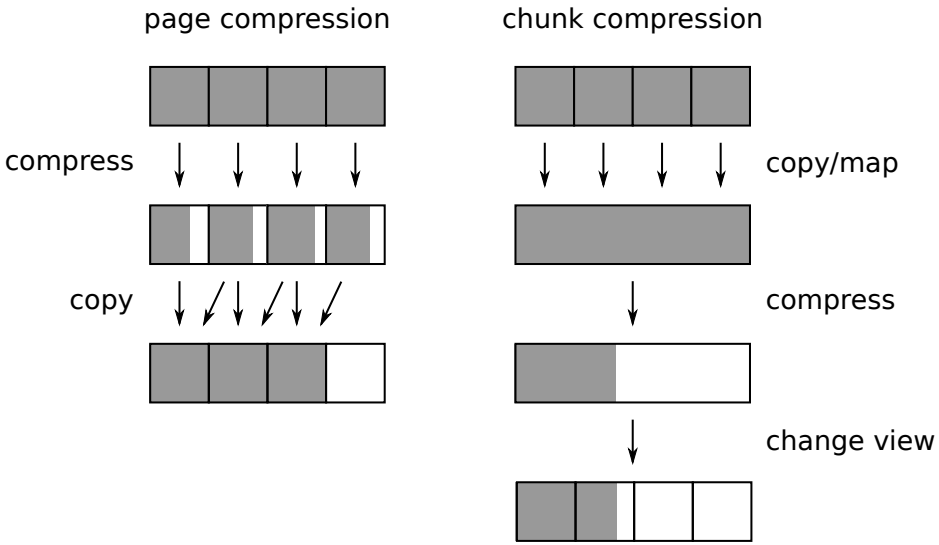


Architecture (write)



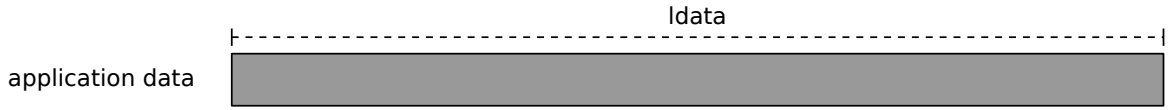
Data structure

- Compression input - page array (`brw_page` array)
- Current LZ4 requires contiguous memory
- Page-wise compression - too low ratios, still many copies
- Continuous compression requires additional memory
  - Original pages stay unaffected since seen by application
  - Additional memory held until request is finished



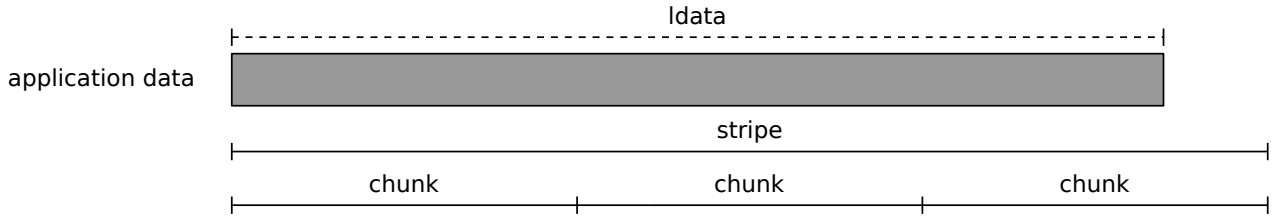
Chunks

- Logical unit: page $<$ chunk $\leq$ stripe $\leq$ RPC



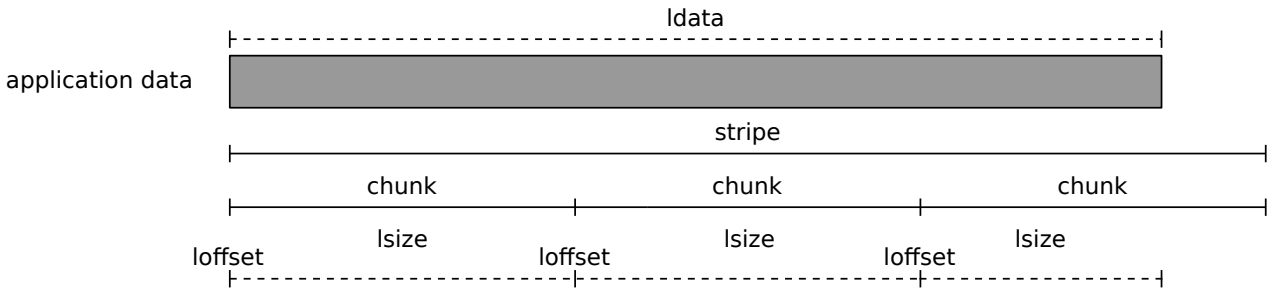
Chunks

- Logical unit: page $<$ chunk $\leq$ stripe $\leq$ RPC



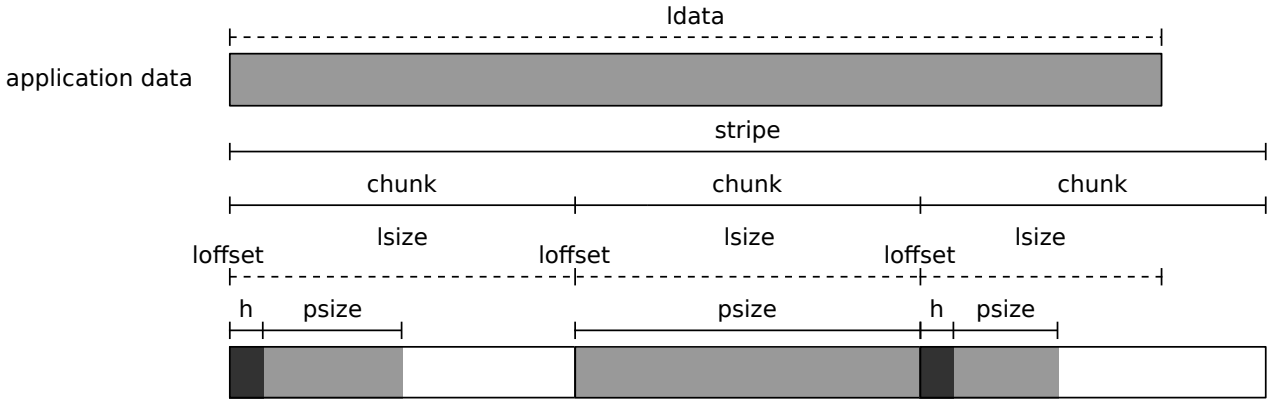
Chunks

- Logical unit: page $<$ chunk $\leq$ stripe $\leq$ RPC



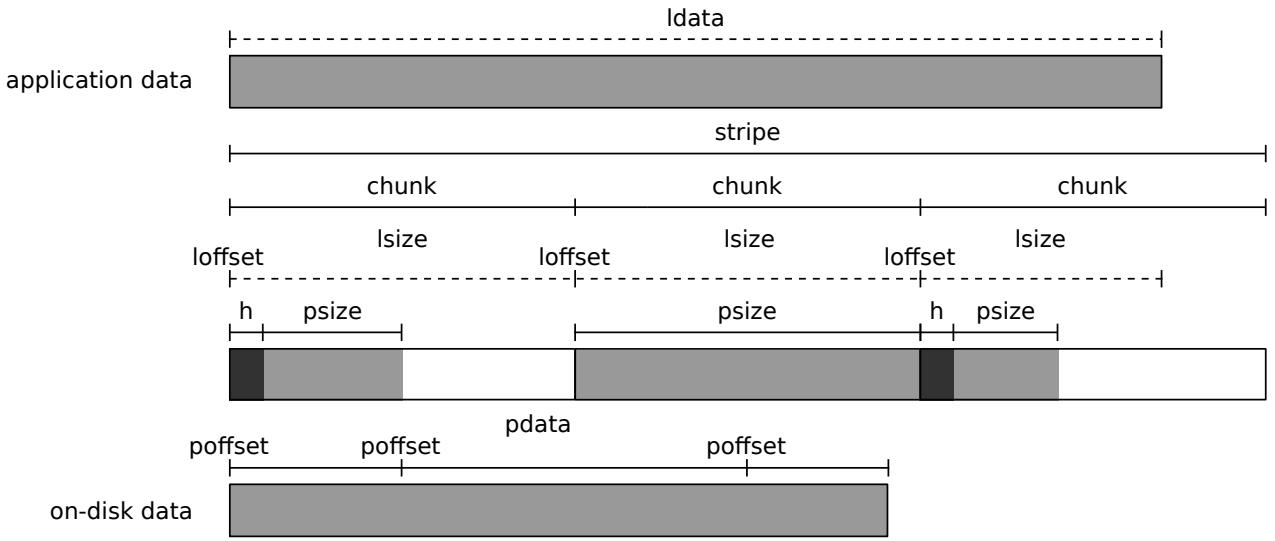
Chunks

- Logical unit: page $<$ chunk $\leq$ stripe $\leq$ RPC



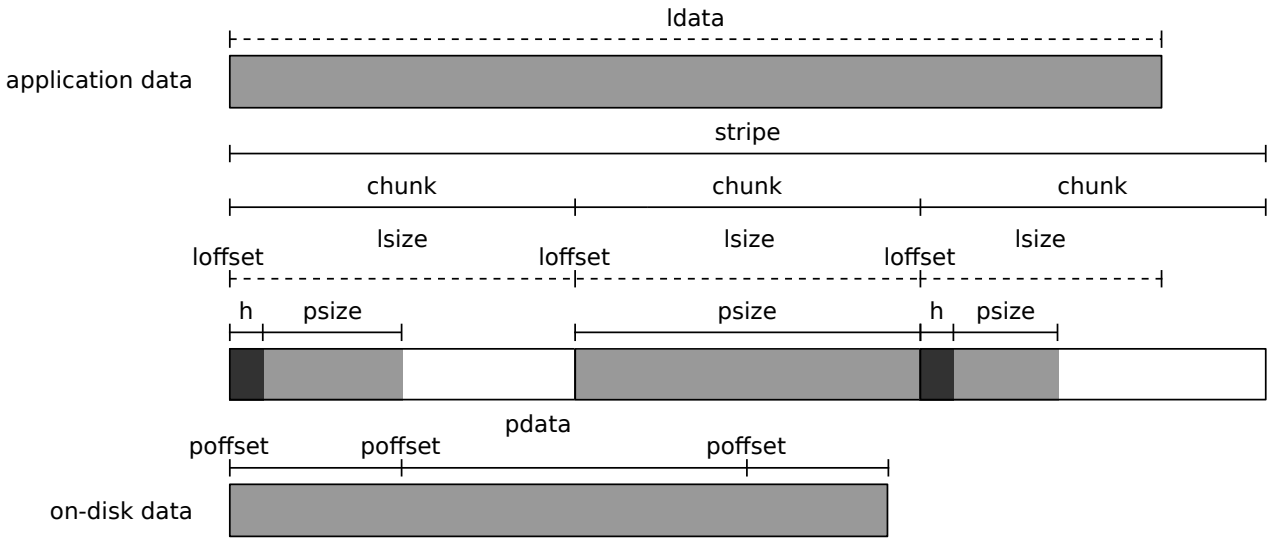
Chunks

- Logical unit: page $<$ chunk $\leq$ stripe $\leq$ RPC



Chunks

- Logical unit: page $<$ chunk $\leq$ stripe $\leq$ RPC



- 1 chunk = 1 `niobuf`
- `chunksize` to be `recordsize`
- Header with `psize`-value (4B) in each compressed chunk
  - ZFS's LZ4 compatible
- Chunk descriptor per chunk
  - `l/psize`, `l/poffset`, `l/ppages`, `flags`, `algo`

Request

- E.g. 8 RPCs in parallel
- 1 RPC up to 16 MiB
- Continual memory allocating and freeing expensive
- Reuse memory for next buffers

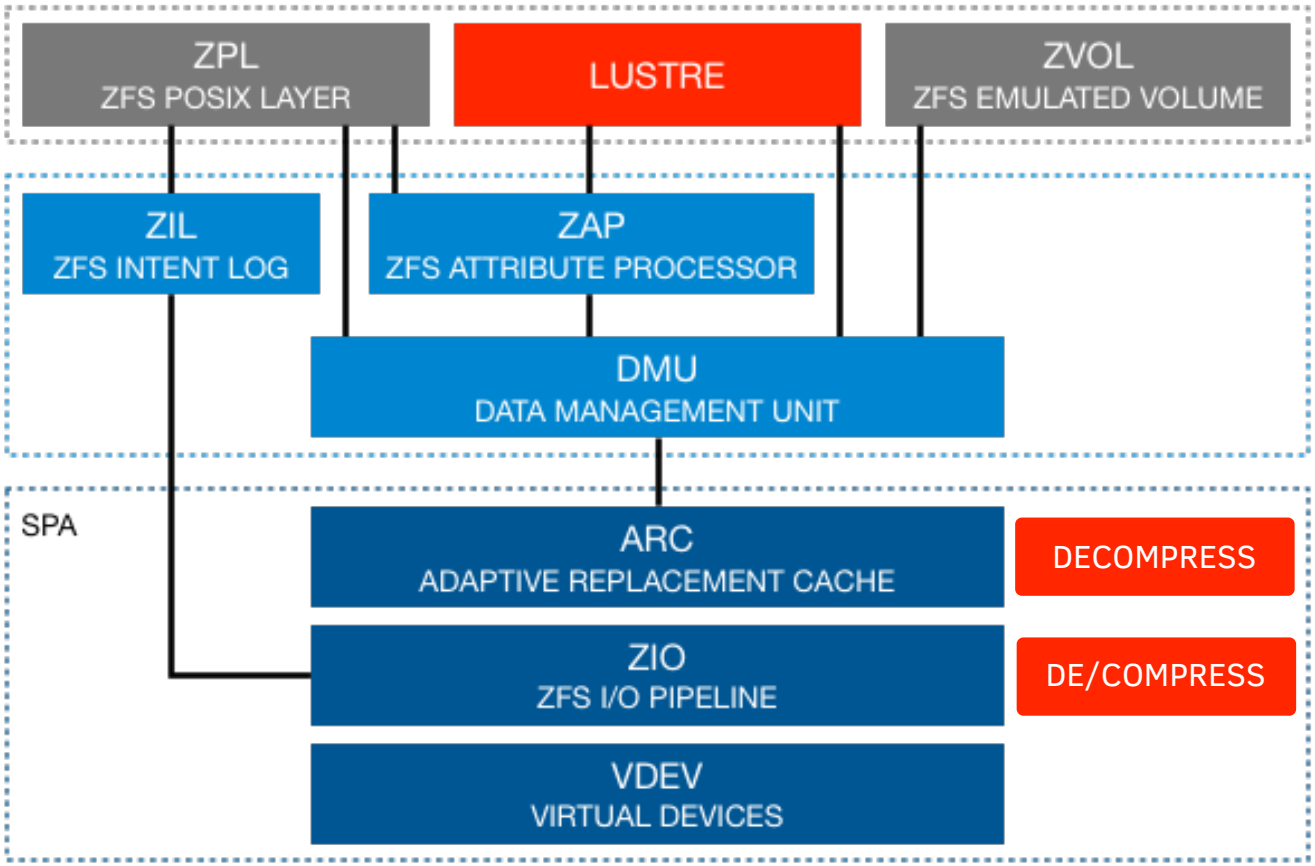
Request

- E.g. 8 RPCs in parallel
- 1 RPC up to 16 MiB
- Continual memory allocating and freeing expensive
- Reuse memory for next buffers
- Use a memory pool
- Limited memory
  - Once failed for a chunk - proceed with original uncompressed chunk
  - Once failed for a request - proceed with uncompressed request
- Reuse compressed page arrays for redo after potential network failure
- Release when request successfully finished

Memory Pool

- Currently luxury with `kmalloc`
  - Physically contiguous
  - Fast allocation, fast usage
  - Very limited amount, fragmentation problem
- 1 pool per client and server
  - Allocates x% of RAM before system start
  - `MAX(32, #cpus) * default_RPC_size`
  - Loans y blocks of fixed size (recordsize)
    - Atomically for 1 request
    - No priority
    - Put back as soon as possible
  - Enforce `--disable-server` on client to avoid double pool

ZFS


ZPL
ZFS POSIX LAYER
LUSTRE
ZVOL
ZFS EMULATED VOLUME
ZIL
ZFS INTENT LOG
ZAP
ZFS ATTRIBUTE PROCESSOR
DMU
DATA MANAGEMENT UNIT
SPA
ARC
ADAPTIVE REPLACEMENT CACHE
DECOMPRESS
ZIO
ZFS I/O PIPELINE
DE/COMPRESS
VDEV
VIRTUAL DEVICES

ZFS compression

- *Native*:

ZFS compression

- *Native*:
- Data compressed when **written** out to **disk**
  - Stored compressed on disk
  - Compression in ZIO-layer

ZFS compression

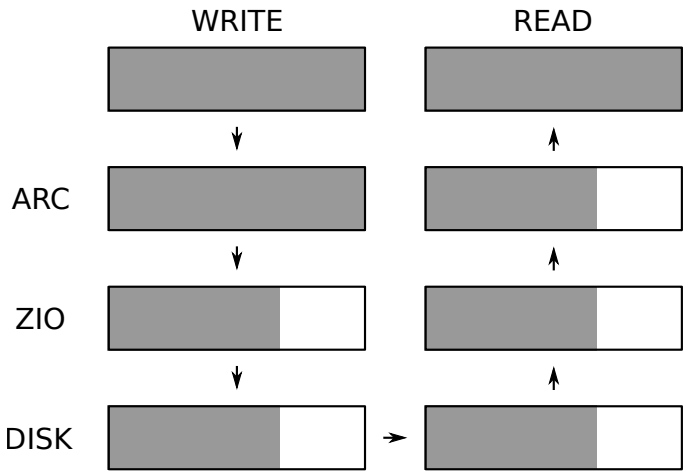
- *Native*:
- Data compressed when **written** out to **disk**
  - Stored compressed on disk
  - Compression in ZIO-layer
- Data decompressed when **read** from **disk**
  - Read less bytes
  - Decompression up to 30% of memcopy speed
  - Decompression in ZIO-layer

ZFS compression

- *Compressed ARC* (since ZoL 0.7)
- Data compressed when **written** out to **disk**
  - Stored compressed on disk
  - Compression in ZIO-layer
- Data decompressed when **read** from **ARC**
  - Compressed block from disk lands in ARC
  - Decompression up to 30% of memcopy speed
  - Decompression in ARC-layer
  - Block in ARC remains compressed

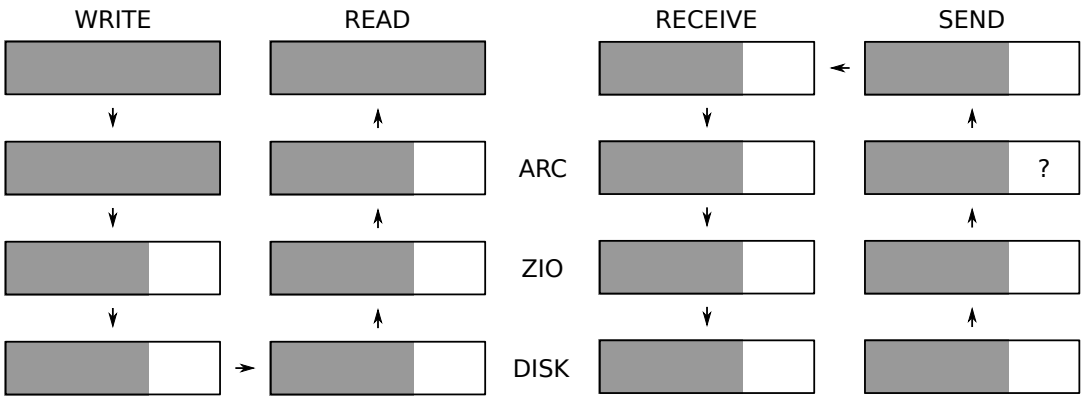
ZFS compression

- *Compressed ARC* (since ZoL 0.7)
- Data compressed when **written** out to **disk**
  - Stored compressed on disk
  - Compression in ZIO-layer
- Data decompressed when **read** from **ARC**
  - Compressed block from disk lands in ARC
  - Decompression up to 30% of memcopy speed
  - Decompression in ARC-layer
  - Block in ARC remains compressed



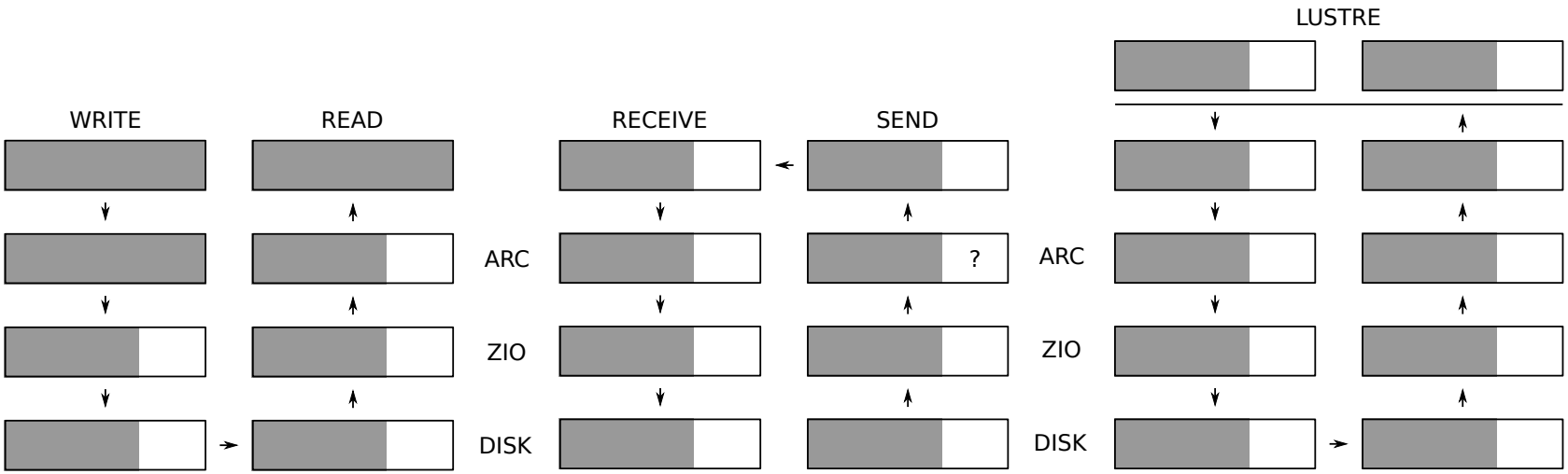
ZFS compressed send/receive

- Stream data for migration/backup/snapshots/...
- Send = read, receive = write
- Interface for incoming and outgoing compressed data



ZFS compressed send/receive

- Stream data for migration/backup/snapshots/...
- Send = read, receive = write
- Interface for incoming and outgoing compressed data
- Our proposed interface extensions
  - Combine all paths, extend interface, be compatible
  - Gain compressed ARC on write path!



OSD-zfs write

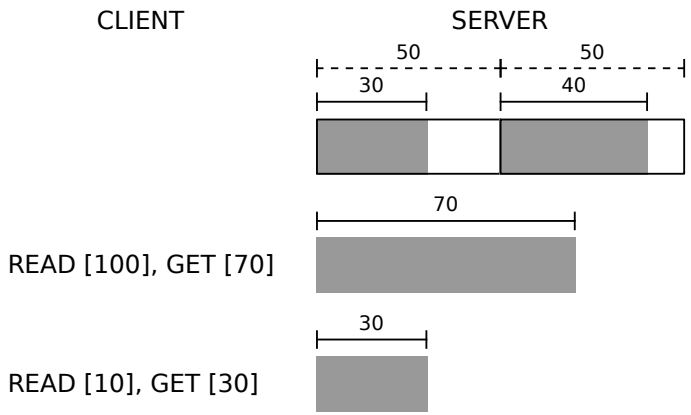
- Full block
  - Zero copy write
  - ARC-buffer loaned to Lustre
  - `dmu_request_arcbuf (db, dn->dn_datablksz)`
  - `dmu_request_compressed_arcbuf (..., psize, cmp)`
    - `dn->dn_datablksz` to be specified by client, not ZFS
    - LU-8342 - set dnodesize and recordsize at file system creation
    - Currently overwritten outside of transaction
    - Problem - recordsize only upper bound

OSD-zfs write

- Full block
  - Zero copy write
  - ARC-buffer loaned to Lustre
  - `dmu_request_arcbuf (db, dn->dn_datablksz)`
  - `dmu_request_compressed_arcbuf (..., psize, cmp)`
    - `dn->dn_datablksz` to be specified by client, not ZFS
    - LU-8342 - set dnodesize and recordsize at file system creation
    - Currently overwritten outside of transaction
    - Problem - recordsize only upper bound
- Partial block
  - Part of a block to be written
  - No buffer loaning, Lustre allocates additional pages
  - Copy of corresponding data
  - Loop over `dmu_write (page)`
  - Problem - completely different path
  - No interface to pass through any compression info

OSD-zfs read

- `dmu_buf_hold_array_by_bonus`
  - Finds/creates appropriate ARC buffers
  - Buffers filled with actual data
- `dmu_buf_hold_array_by_bonus_compressed`
  - Returns additional compression "metadata"
  - Asking for lsizes, getting psizes in lbuffers



- Number of niobufs depending on logical chunks
  - 1 chunk descriptor per chunk/niobuf

OSD-zfs read

- ARC-match
  - Data is compressed in ARC
  - ZFS disk format to be compatible
  - Able to decompress, but preventing here

OSD-zfs read

- ARC-match
  - Data is compressed in ARC
  - ZFS disk format to be compatible
  - Able to decompress, but preventing here
- ZIO-read
  - Data is not in ARC, start disk I/O
  - Prevent decompression in ZIO- and ARC-layer

OSD-zfs read

- ARC-match
  - Data is compressed in ARC
  - ZFS disk format to be compatible
  - Able to decompress, but preventing here
- ZIO-read
  - Data is not in ARC, start disk I/O
  - Prevent decompression in ZIO- and ARC-layer
- Getting and setting correct flags
- Due to compatibility, data accessible without Lustre

Close future

- Next commit
  - Remove macros `#ifdef COMPRESSION`
  - Build always, testable without `--enable-compression`
  - Fix minor bugs

Close future

- Next commit
  - Remove macros `#ifdef COMPRESSION`
  - Build always, testable without `--enable-compression`
  - Fix minor bugs
- Update ZFS pull request #8143
  - Revert on-disk-format changes
  - Compatible de-/compression interface

Close future

- Next commit
  - Remove macros `#ifdef COMPRESSION`
  - Build always, testable without `--enable-compression`
  - Fix minor bugs
- Update ZFS pull request #8143
  - Revert on-disk-format changes
  - Compatible de-/compression interface
- Dynamic enable/disable feature
- Write path (store compressed)
- Read path

Close future

- Next commit
  - Remove macros `#ifdef COMPRESSION`
  - Build always, testable without `--enable-compression`
  - Fix minor bugs
- Update ZFS pull request #8143
  - Revert on-disk-format changes
  - Compatible de-/compression interface
- Dynamic enable/disable feature
- Write path (store compressed)
- Read path
- Performance measurements (write)
  - Paper for Eurosys 2020 in progress
  - Deadline 4th November

Performance

- Algorithms
  - LZ4 block format too memory expensive → LZ4 frame
  - zstd
  - Page-array based algorithms
    - Experimental algorithm based on LZ4, developed by us
    - ZFS compatibility?
  - Hardware compression

Performance

- Algorithms
  - LZ4 block format too memory expensive → LZ4 frame
  - zstd
  - Page-array based algorithms
    - Experimental algorithm based on LZ4, developed by us
    - ZFS compatibility?
  - Hardware compression
- Memory pool
  - Switch to `kvmalloc`
  - Add statistics, tuning

Performance

- Algorithms
  - LZ4 block format too memory expensive → LZ4 frame
  - zstd
  - Page-array based algorithms
    - Experimental algorithm based on LZ4, developed by us
    - ZFS compatibility?
  - Hardware compression
- Memory pool
  - Switch to `kvmalloc`
  - Add statistics, tuning
- RMW
  - High flexibility for de-/compression
    - Lustre/ZFS
    - Client/server
    - Write/read
  - Chunking, random I/O
  - Challenge when first chunks grow

LZ4 frame

- Landed in kernel 4.11 with our update and LZ4 fast
- Allows to stream input (and output?)
- Creates dictionary using the entropy of previous blocks
- Iterate over pages as separate blocks
- Output still one buffer
  - May be page array
  - May not be contiguous?
- Performance to be evaluated

LZ4 frame

- Landed in kernel 4.11 with our update and LZ4 fast
- Allows to stream input (and output?)
- Creates dictionary using the entropy of previous blocks
- Iterate over pages as separate blocks
- Output still one buffer
  - May be page array
  - May not be contiguous?
- Performance to be evaluated
- No partial module build for kernel $3.11 < x < 4.11$
  - Probably not a problem for RHEL
  - Always build an own `lustre_lz4` module?
  - Use ZFS's LZ4 module? Client?

Enhanced adaptive ...

- Flexible decisions
  - De-/compress what/where/when
  - Per chunk/stripe/file

Enhanced adaptive ...

- Flexible decisions
  - De-/compress what/where/when
  - Per chunk/stripe/file
- Different focus
  - Overall performance
  - Network throughput
  - Storage space
  - Energy ...

Enhanced adaptive ...

- Flexible decisions
  - De-/compress what/where/when
  - Per chunk/stripe/file
- Different focus
  - Overall performance
  - Network throughput
  - Storage space
  - Energy ...
- Consider all metrics
  - Static stats
    - Clients, servers, disks
    - RAM, network, CPU
  - Dynamic stats
    - Current system load, bottlenecks
  - User load
    - High/low I/O, access patterns

Encryption

- Similarities
  - Rebuild request
  - Memory copy
  - Data transformation - RMW
  - Chunking may be worthwhile

Encryption

- Similarities
  - Rebuild request
  - Memory copy
  - Data transformation - RMW
  - Chunking may be worthwhile
- Differences
  - Data grows
  - No hard requirement for contiguous memory
  - ZFS-compatibility?

Encryption

- Similarities
  - Rebuild request
  - Memory copy
  - Data transformation - RMW
  - Chunking may be worthwhile
- Differences
  - Data grows
  - No hard requirement for contiguous memory
  - ZFS-compatibility?
- Combine
  - What first?
  - Any additional challenges?
- What else?

Open topics.. not addressed yet

- Short I/O
  - Introduce threshold
  - $\texttt{sIO} < \texttt{th} < \texttt{comp}$
  - What happens when a block grows? MDT $\rightarrow$ OST

Open topics.. not addressed yet

- Short I/O
  - Introduce threshold
  - `sIO` $<$ `th` $<$ `comp`
  - What happens when a block grows? MDT $\rightarrow$ OST
- Compatibility flags
  - Client/Server versions
  - MDS with ldiskfs?

Open topics.. not addressed yet

- Short I/O
  - Introduce threshold
  - `sIO` $<$ `th` $<$ `comp`
  - What happens when a block grows? MDT $\rightarrow$ OST
- Compatibility flags
  - Client/Server versions
  - MDS with ldiskfs?
- Dynamic layout
  - Stripe $<$ chunk ?