# Optimize POLARDB on POLARSTORE

Zongzhi Chen
Alibaba Cloud

# Agenda

- Overview of POLARDB
- Compare POLARFS and ext4
- Optimize on the redo IO
- Optimize on the page IO

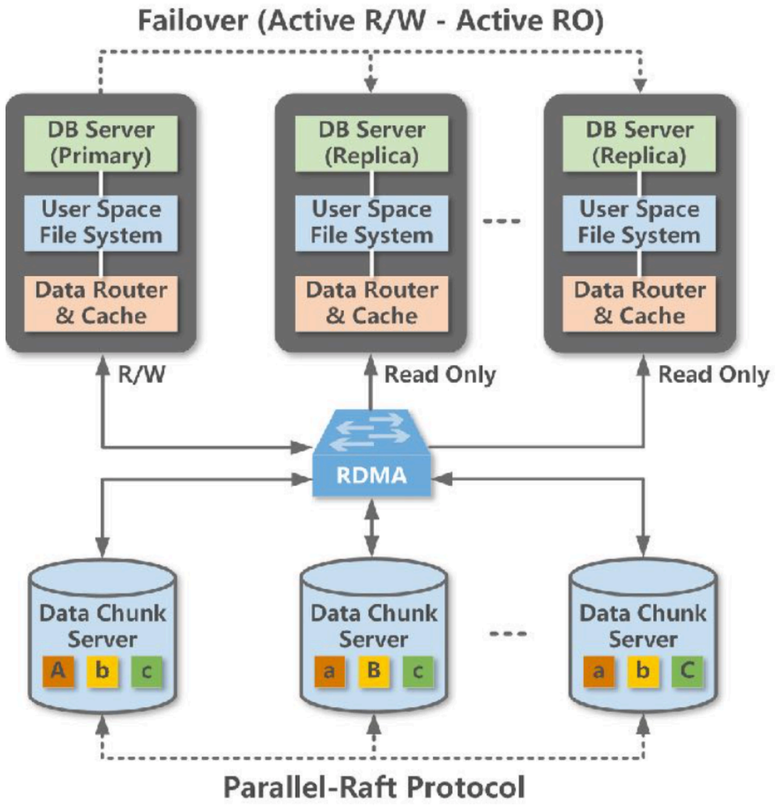
# Agenda

- Overview of POLARDB
- Compare POLARFS and ext4
- Optimize on the redo IO
- Optimize on the page IO

# POLARDB

# Architecture of POLARDB

**Separation of Compute & Storage**



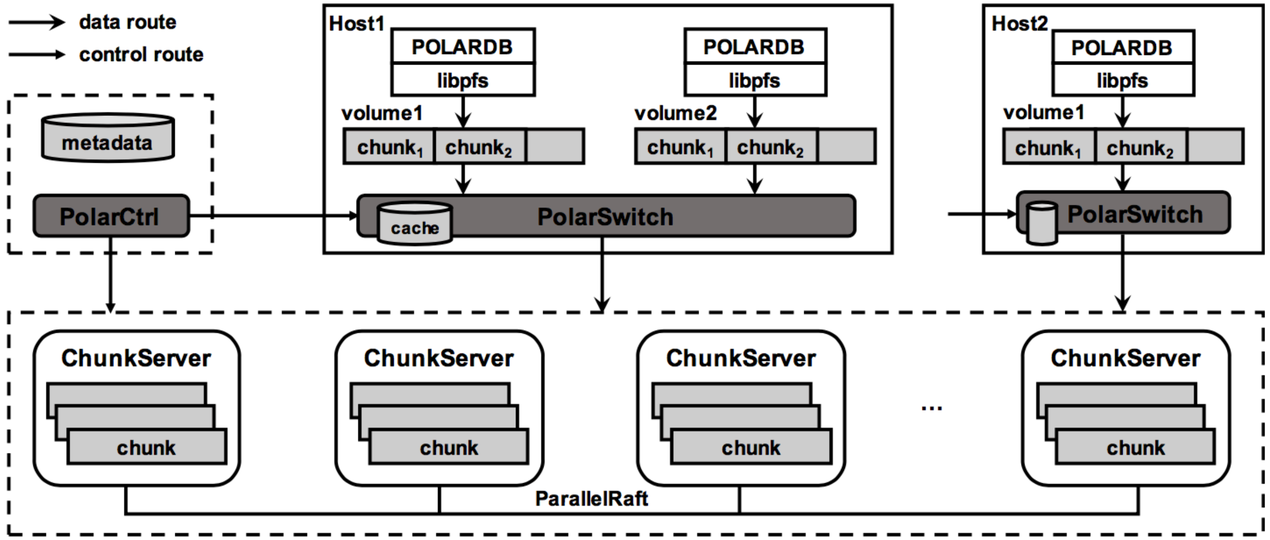

Different hardwares,  customized, optimized

Disk pool, no fragmentation, no imbalance, easy to scale-out

Easy for compute migration, improved storage for replication & HA

Easy to implement Serverless

# Architecture of POLARDB




Separation of computing & storage

All in userspace, zero copy

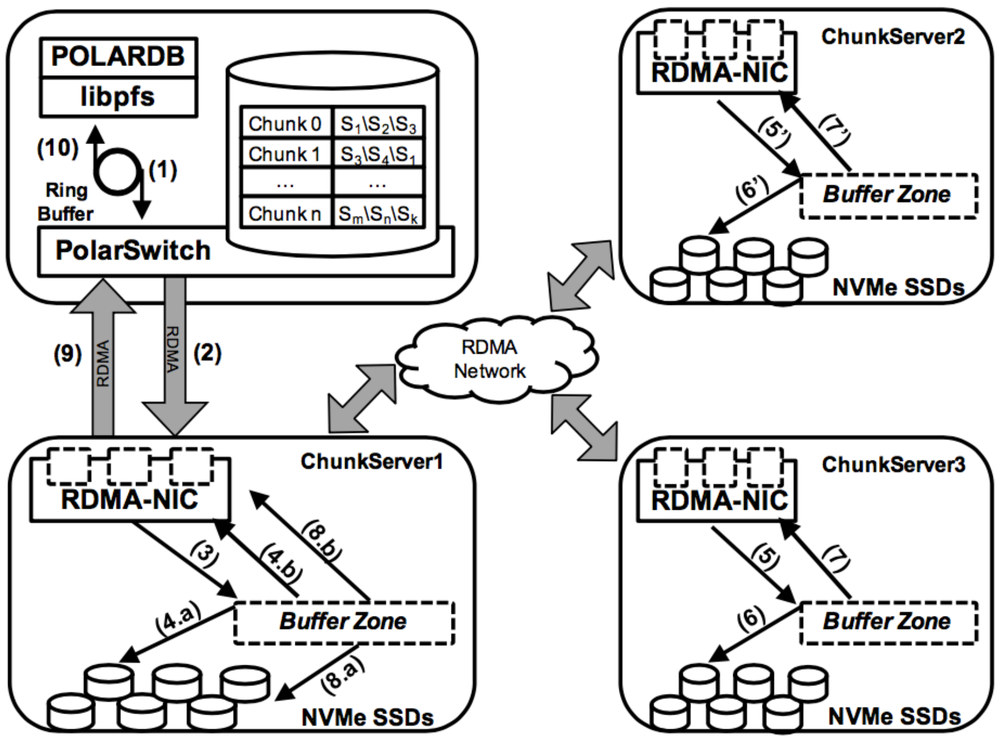
ParellelRaft, out of order Raft

Exploration of new hardwares

# POLARFS

# Architecture of POLARDB

**All Userspace**

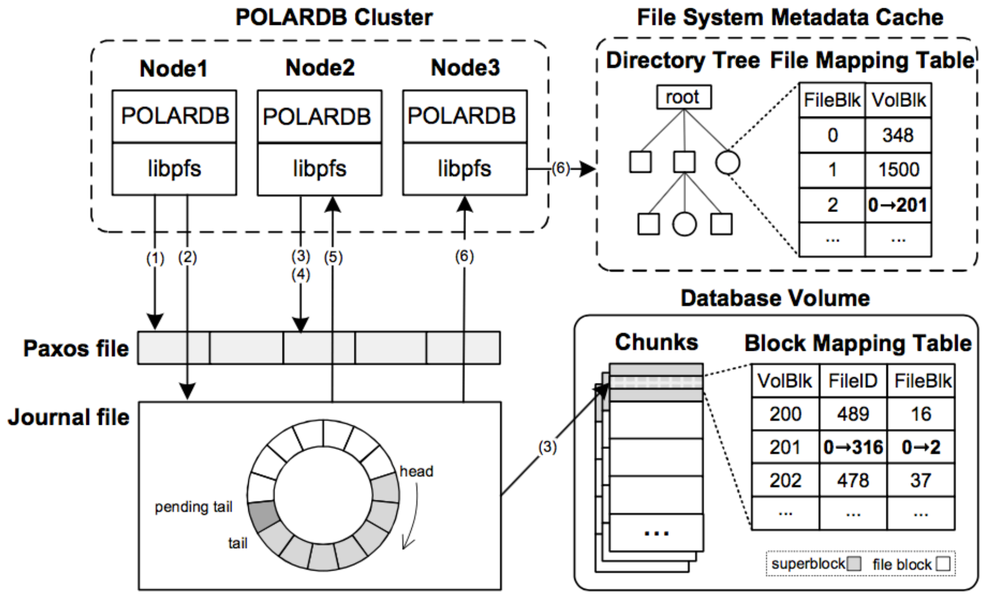



DPDK RDMA SPDK zero copy

All userspace, no context switch

# Architecture of POLARDB

**Illustration of POLARDB filesystem**



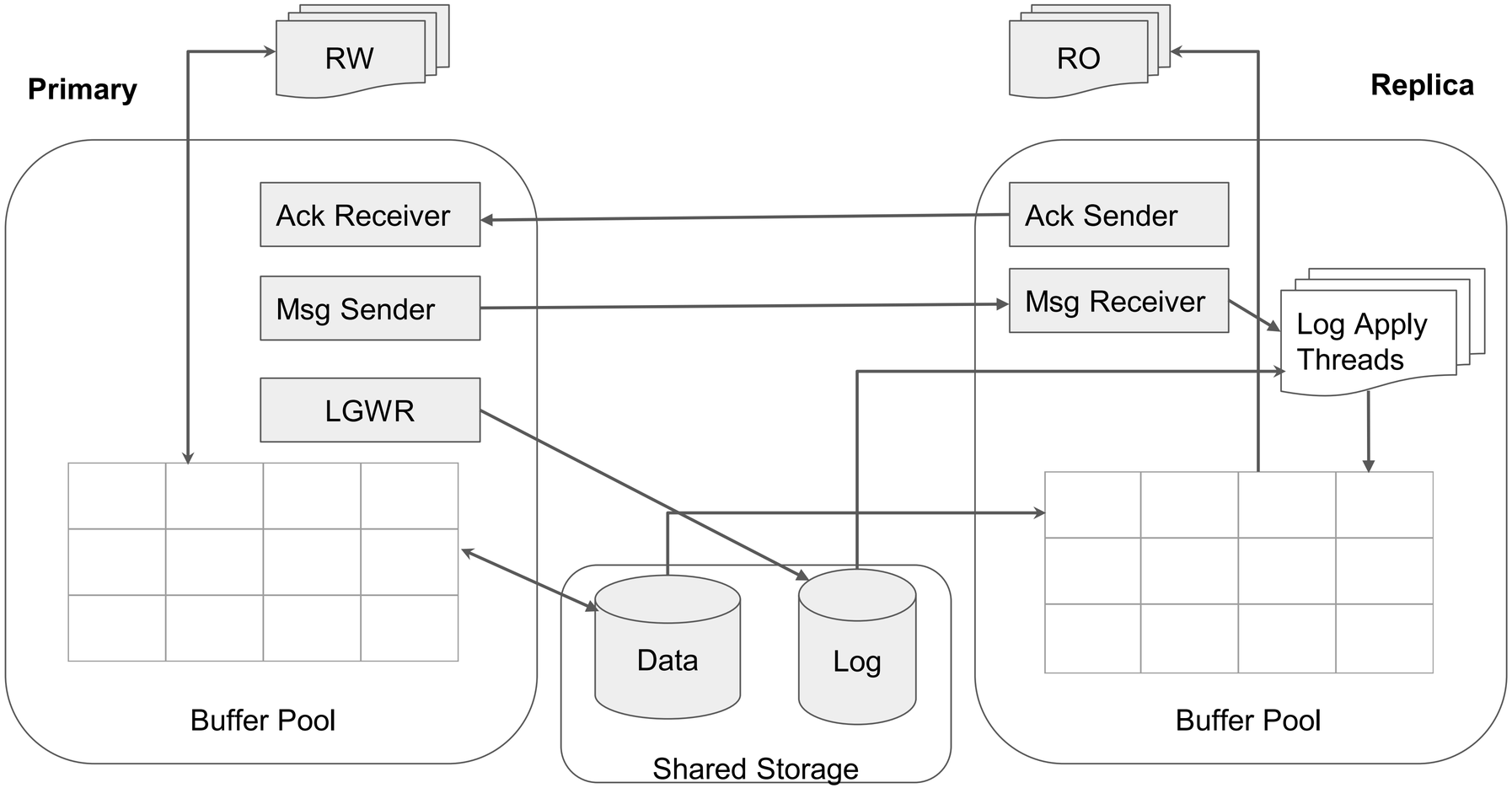

POSIX API, low invasion to DB engine

Metadata synced by PAXOS globally

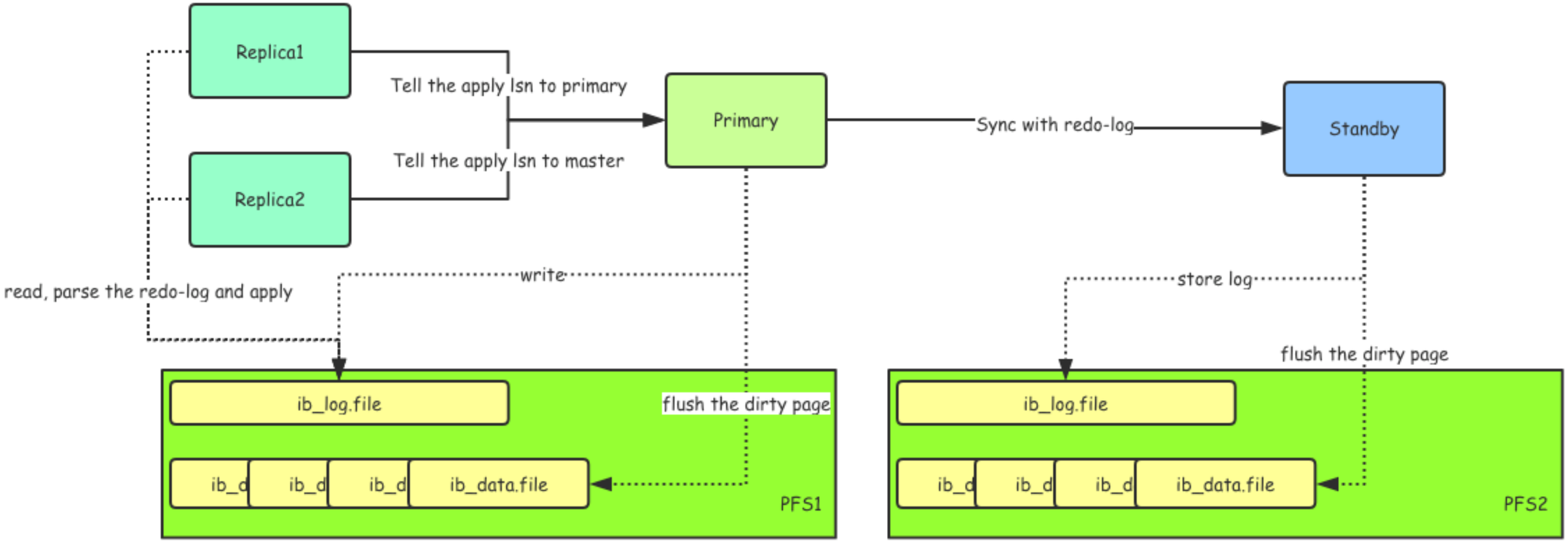
Metadata cache for read acceleration

Volume->mount point
Directory->Directory
File->File
Chunk->Block

# Physical Copy



# Architecture with POLARFS

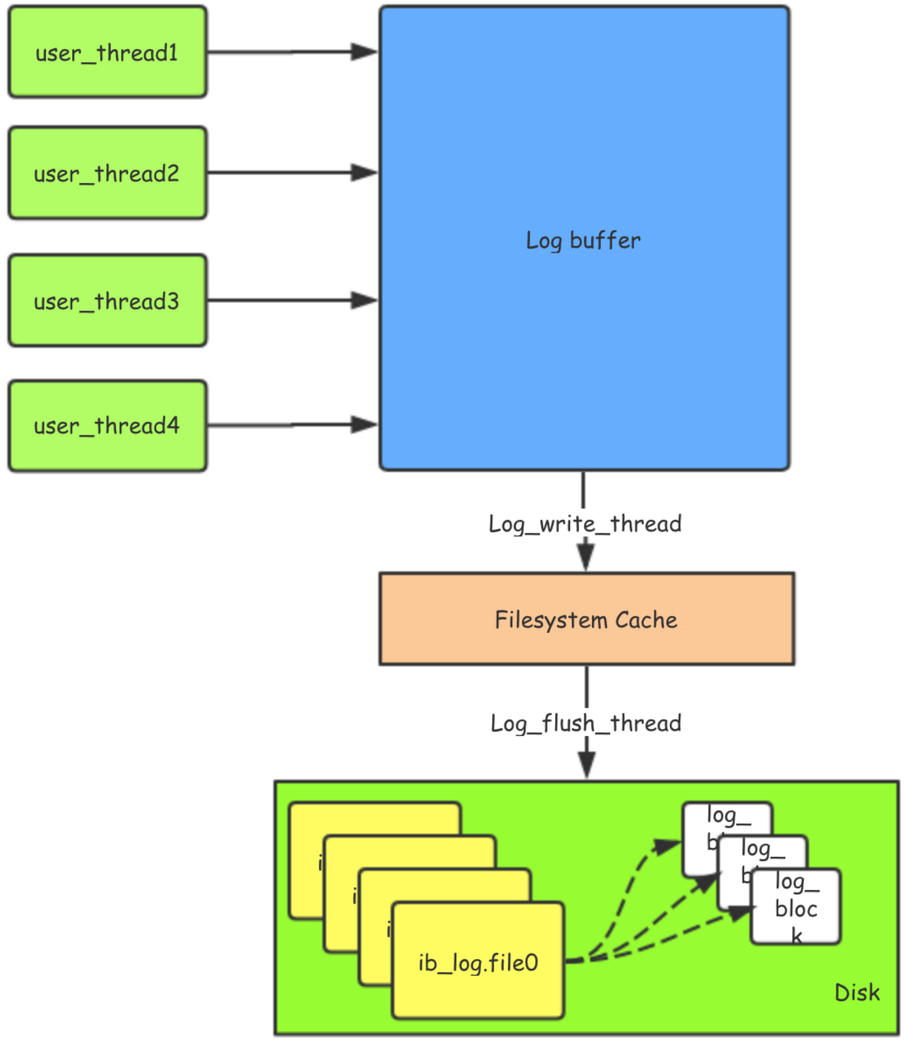


# POLARFS vs ext4

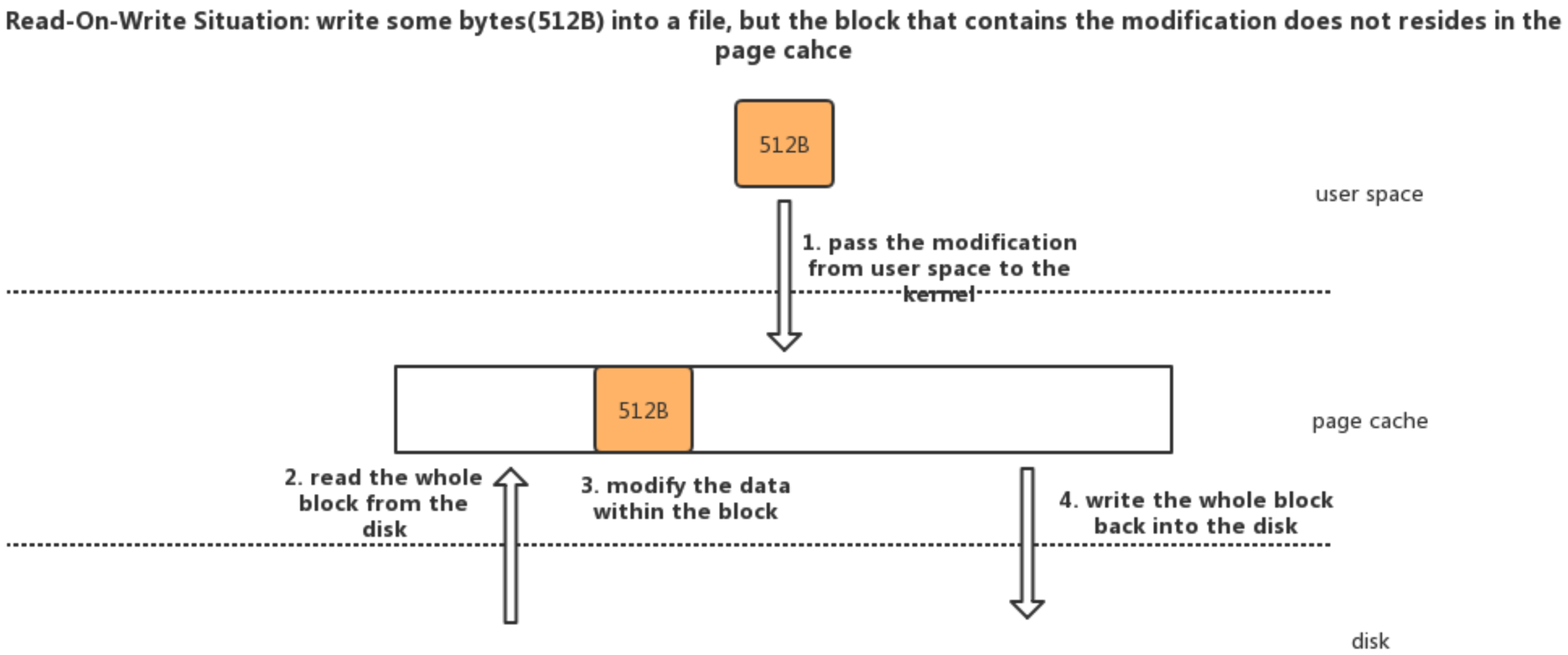
- no page cache
- support 16kb atomic write
- no asynchronous IO
- a bit higher latency,  higher bandwith
- only support 4k aligned write

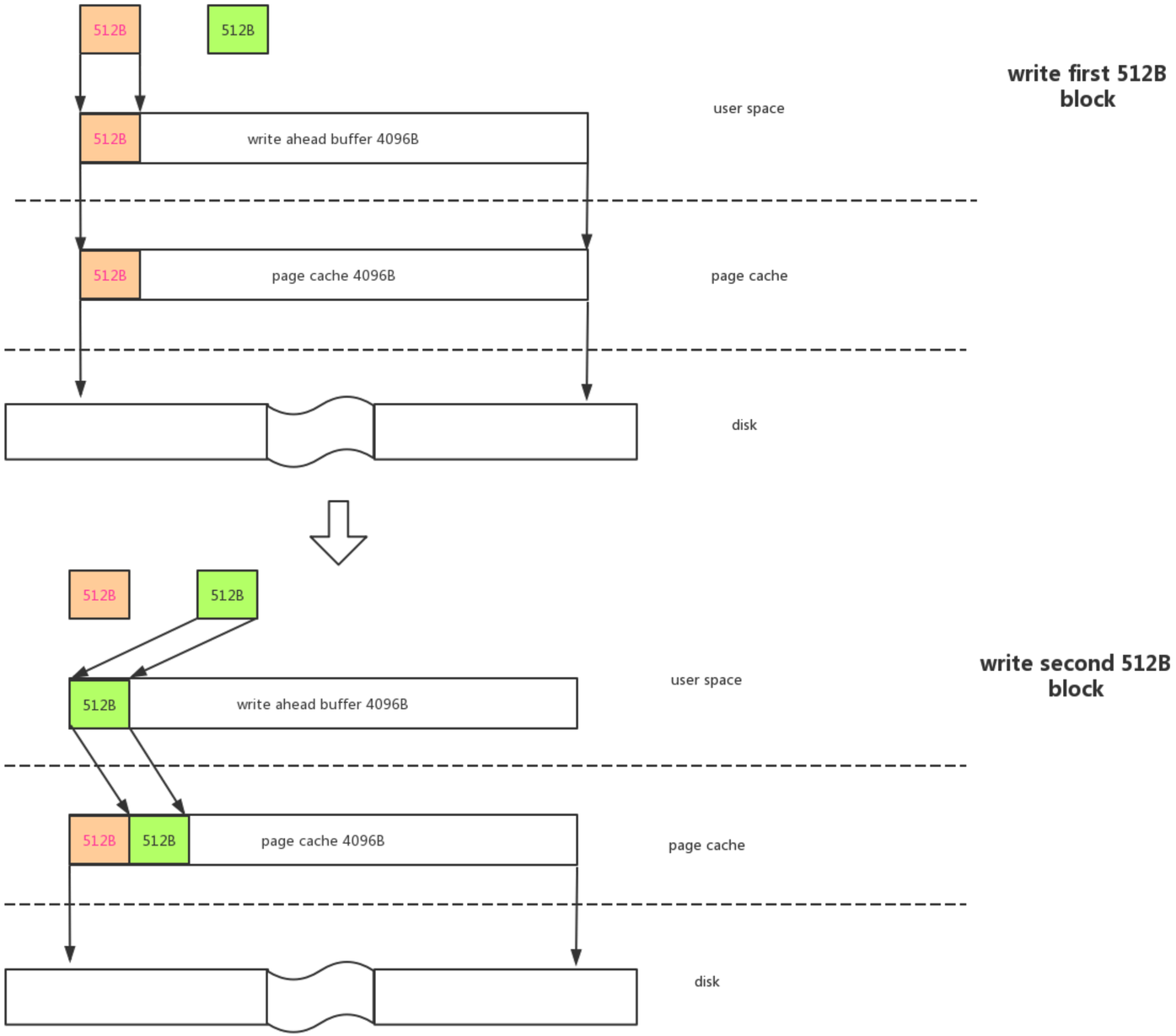
# Optimize the redo IO

# Redo log in Mysql8.0



# “read-on-write”




512B
512B
write first 512B block
user space
512B
write ahead buffer 4096B
512B
page cache 4096B
page cache
disk
512B
512B
write second 512B block
user space
512B
write ahead buffer 4096B
512B
512B
page cache 4096B
page cache
disk

# Things change in POLARDB

- redo log from rotate to increasing
- POLARFS won’t support page cache

# “read-on-write” Example

- squential write 1G file
- write 512 byte every time

# Simple Way

*// The blotrace information by the simple write way*
*// we can find that there is read IO in this way, and the read IO: write IO = 1:8*
*// the 1:8 is that 4k/512byte = 8:1*
*// It mean that when we doing 8 times write IO, then there will have a read IO to read*
*// the data from the file system*

**# an IO start here**

```
259,6    0      184     0.001777196 58995  A   R 55314456 + 8 <- (259,9) 55312408
259,9    0      185     0.001777463 58995  Q   R 55314456 + 8 [a.out]
259,9    0      186     0.001777594 58995  G   R 55314456 + 8 [a.out]
259,9    0      187     0.001777863 58995  D  RS 55314456 + 8 [a.out]
259,9    0      188     0.002418822     0  C  RS 55314456 + 8 [0]
```

**# end of an IO**

```
259,6    0      189     0.002423915 58995  A  WS 55314456 + 8 <- (259,9) 55312408
259,9    0      190     0.002424192 58995  Q  WS 55314456 + 8 [a.out]
259,9    0      191     0.002424434 58995  G  WS 55314456 + 8 [a.out]
259,9    0      192     0.002424816 58995  U   N [a.out] 1
259,9    0      193     0.002424992 58995  I  WS 55314456 + 8 [a.out]
259,9    0      194     0.002425247 58995  D  WS 55314456 + 8 [a.out]
259,9    0      195     0.002432434     0  C  WS 55314456 + 8 [0]
```

# InnoDB Way

*// the blktrace information by the void "read-on-write" way*
*// we can find that there won't be read IO in this way*

```
259,9    2      357     0.001166883     0  C  WS 75242264 + 8 [0]
## IO start
259,6    2      358     0.001173249 113640  A  WS 75242264 + 8 <- (259,9) 75240216
259,9    2      359     0.001173558 113640  Q  WS 75242264 + 8 [a.out]
259,9    2      360     0.001173664 113640  G  WS 75242264 + 8 [a.out]
259,9    2      361     0.001173939 113640  U   N [a.out] 1
259,9    2      362     0.001174017 113640  I  WS 75242264 + 8 [a.out]
259,9    2      363     0.001174249 113640  D  WS 75242264 + 8 [a.out]
259,9    2      364     0.001180838     0  C  WS 75242264 + 8 [0]
## IO end
259,6    2      365     0.001187163 113640  A  WS 75242264 + 8 <- (259,9) 75240216
259,9    2      366     0.001187367 113640  Q  WS 75242264 + 8 [a.out]
259,9    2      367     0.001187477 113640  G  WS 75242264 + 8 [a.out]
259,9    2      368     0.001187755 113640  U   N [a.out] 1
259,9    2      369     0.001187835 113640  I  WS 75242264 + 8 [a.out]
259,9    2      370     0.001188072 113640  D  WS 75242264 + 8 [a.out]
259,9    2      371     0.001194495     0  C  WS 75242264 + 8 [0]
```

# append-write vs overwriting

# Test Example

- buffer write
- fallocate + buffer write
- fallocate + filling zero + buffer write

# Buffer Write (1)

*# jbd2 modification metadata operation*

```
259,6   33      200     0.000755218  1392  A  WS 1875247968 + 8 <- (259,9) 1875245920
259,9   33      201     0.000755544  1392  Q  WS 1875247968 + 8 [jbd2/nvme8n1p1-]
259,9   33      202     0.000755687  1392  G  WS 1875247968 + 8 [jbd2/nvme8n1p1-]
259,6   33      203     0.000756124  1392  A  WS 1875247976 + 8 <- (259,9) 1875245928
259,9   33      204     0.000756372  1392  Q  WS 1875247976 + 8 [jbd2/nvme8n1p1-]
259,9   33      205     0.000756607  1392  M  WS 1875247976 + 8 [jbd2/nvme8n1p1-]
259,6   33      206     0.000756920  1392  A  WS 1875247984 + 8 <- (259,9) 1875245936
259,9   33      207     0.000757191  1392  Q  WS 1875247984 + 8 [jbd2/nvme8n1p1-]
259,9   33      208     0.000757293  1392  M  WS 1875247984 + 8 [jbd2/nvme8n1p1-]
259,6   33      209     0.000757580  1392  A  WS 1875247992 + 8 <- (259,9) 1875245944
259,9   33      210     0.000757834  1392  Q  WS 1875247992 + 8 [jbd2/nvme8n1p1-]
259,9   33      211     0.000758032  1392  M  WS 1875247992 + 8 [jbd2/nvme8n1p1-]
259,9   33      212     0.000758333  1392  U   N [jbd2/nvme8n1p1-] 1
259,9   33      213     0.000758425  1392  I  WS 1875247968 + 32 [jbd2/nvme8n1p1-]
259,9   33      214     0.000759065  1392  D  WS 1875247968 + 32 [jbd2/nvme8n1p1-]
259,9   33      342     0.001614981     0  C  WS 1875122848 + 16 [0]
```

*# summit the jbd2 IO, here we will commit 16 * 512 = 16kb data*

# Buffer Write(2)

259,6 33 216 0.000775814 1392 A FWFS 1875248000 + 8 <- (259,9) 1875245952
259,9 33 217 0.000776110 1392 Q WS 1875248000 + 8 [jbd2/nvme8n1p1-]
259,9 33 218 0.000776207 1392 G WS 1875248000 + 8 [jbd2/nvme8n1p1-]
259,9 33 219 0.000776609 1392 D WS 1875248000 + 8 [jbd2/nvme8n1p1-]
259,9 33 220 0.000783089 0 C WS 1875248000 + 8 [0]
*# another operation summit jbd2 IO, this time will summit 8 * 512 = 4k data size*

# user IO start
259,6 2 64 0.000800621 121336 A WS 297152 + 8 <- (259,9) 295104
259,9 2 65 0.000801007 121336 Q WS 297152 + 8 [a.out]
259,9 2 66 0.000801523 121336 G WS 297152 + 8 [a.out]
259,9 2 67 0.000802355 121336 U N [a.out] 1
259,9 2 68 0.000802469 121336 I WS 297152 + 8 [a.out]
259,9 2 69 0.000802911 121336 D WS 297152 + 8 [a.out]
259,9 2 70 0.000810247 0 C WS 297152 + 8 [0]
# user IO end

# Fallocate + Buffer Write

*# jbd2 modify meta data operation*

```
259,6   33      333     0.001604577  1392  A  WS 1875122848 + 8 <- (259,9) 1875120800
259,9   33      334     0.001604926  1392  Q  WS 1875122848 + 8 [jbd2/nvme8n1p1-]
259,9   33      335     0.001605169  1392  G  WS 1875122848 + 8 [jbd2/nvme8n1p1-]
259,6   33      336     0.001605627  1392  A  WS 1875122856 + 8 <- (259,9) 1875120808
259,9   33      337     0.001605896  1392  Q  WS 1875122856 + 8 [jbd2/nvme8n1p1-]
259,9   33      338     0.001606108  1392  M  WS 1875122856 + 8 [jbd2/nvme8n1p1-]
259,9   33      339     0.001606465  1392  U   N [jbd2/nvme8n1p1-] 1
259,9   33      340     0.001606622  1392  I  WS 1875122848 + 16 [jbd2/nvme8n1p1-]
259,9   33      341     0.001607091  1392  D  WS 1875122848 + 16 [jbd2/nvme8n1p1-]
259,9   33      342     0.001614981     0  C  WS 1875122848 + 16 [0]
```

*# summit the jdb2 IO operations, compare with buffer write, this time we only write 16 * 512 = 8K data*

# Fallocate + Buffer Write

```
259,6   33      343     0.001619920  1392  A FWFS 1875122864 + 8 <- (259,9) 1875120816
259,9   33      344     0.001620237  1392  Q  WS 1875122864 + 8 [jbd2/nvme8n1p1-]
259,9   33      345     0.001620443  1392  G  WS 1875122864 + 8 [jbd2/nvme8n1p1-]
259,9   33      346     0.001620694  1392  D  WS 1875122864 + 8 [jbd2/nvme8n1p1-]
259,9   33      347     0.001627171     0  C  WS 1875122864 + 8 [0]
# another operation summit jbd2 IO, this time will summit 8 * 512 = 4k data size
# user IO start
259,6   49      146     0.001641484 119984  A  WS 119802016 + 8 <- (259,9) 119799968
259,9   49      147     0.001641825 119984  Q  WS 119802016 + 8 [a.out]
259,9   49      148     0.001642057 119984  G  WS 119802016 + 8 [a.out]
259,9   49      149     0.001642770 119984  U   N [a.out] 1
259,9   49      150     0.001642946 119984  I  WS 119802016 + 8 [a.out]
259,9   49      151     0.001643426 119984  D  WS 119802016 + 8 [a.out]
259,9   49      152     0.001649782     0  C  WS 119802016 + 8 [0]
# end of user IO
```

# Fallocate + Filling Zero + Buffer Write

```
# user IO start
259,6    0      184     0.001777196 58995  A   R 55314456 + 8 <- (259,9) 55312408
259,9    0      185     0.001777463 58995  Q   R 55314456 + 8 [a.out]
259,9    0      186     0.001777594 58995  G   R 55314456 + 8 [a.out]
259,9    0      187     0.001777863 58995  D  RS 55314456 + 8 [a.out]
259,9    0      188     0.002418822     0  C  RS 55314456 + 8 [0]
# end of user IO
259,6    0      189     0.002423915 58995  A  WS 55314456 + 8 <- (259,9) 55312408
259,9    0      190     0.002424192 58995  Q  WS 55314456 + 8 [a.out]
259,9    0      191     0.002424434 58995  G  WS 55314456 + 8 [a.out]
259,9    0      192     0.002424816 58995  U   N [a.out] 1
259,9    0      193     0.002424992 58995  I  WS 55314456 + 8 [a.out]
259,9    0      194     0.002425247 58995  D  WS 55314456 + 8 [a.out]
259,9    0      195     0.002432434     0  C  WS 55314456 + 8 [0]
```

# Result

- buffer write < fallocate + buffer write < fallocate + filling zero + buffer write
- fallocate + filling zero + buffer write : buffer write = 1:4

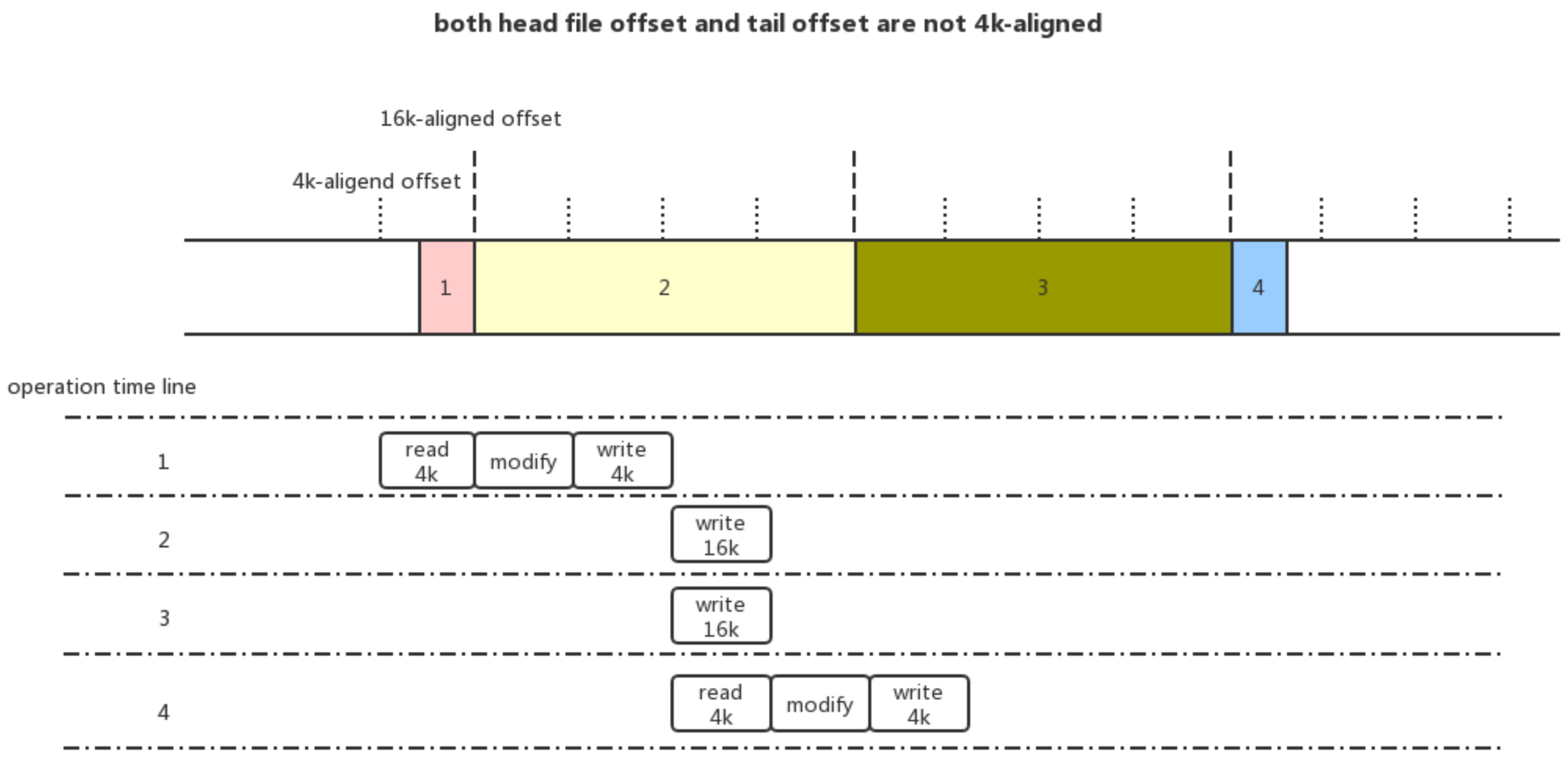
# POLARDB on POLARFS

- background thread allocate new file before not free redo log
- rename old purged redo file
- fallocate file and fill zero
- disable double write buffer

# Aligned Write

- no page cache
  - avoid smaller write
  - avoid un-aligned write
- parallel write size 128K

# pfs-write-timeline

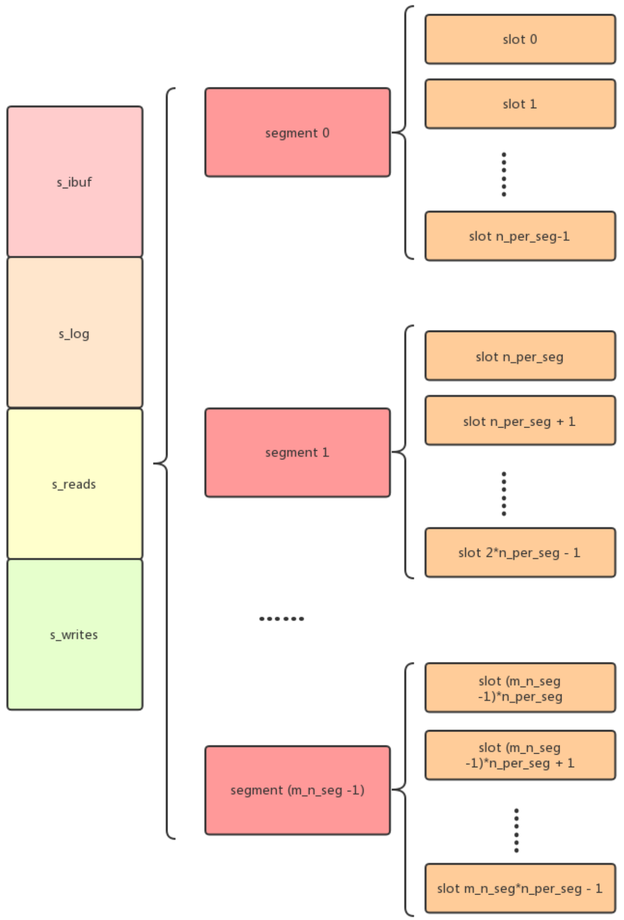


# POLARDB on POLARFS

- INNODB_LOG_WRITE_MAX_SIZE_DEFAULT 4k => 128k
- recent write buffer  1M => 4M
- padding write in log_writer_write_buffer

# Optimize the page IO

# Simulator AIO



# Difference

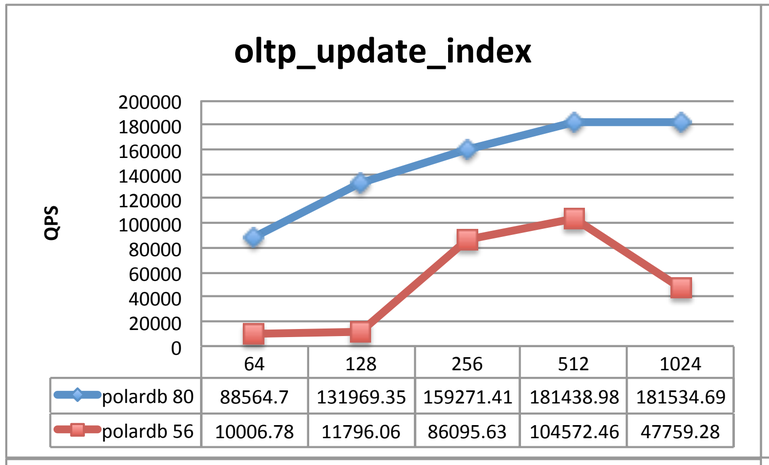
- a bit larger latency, however larger bandwidth
- parallel write size 128k

# POLARDB on POLARFS

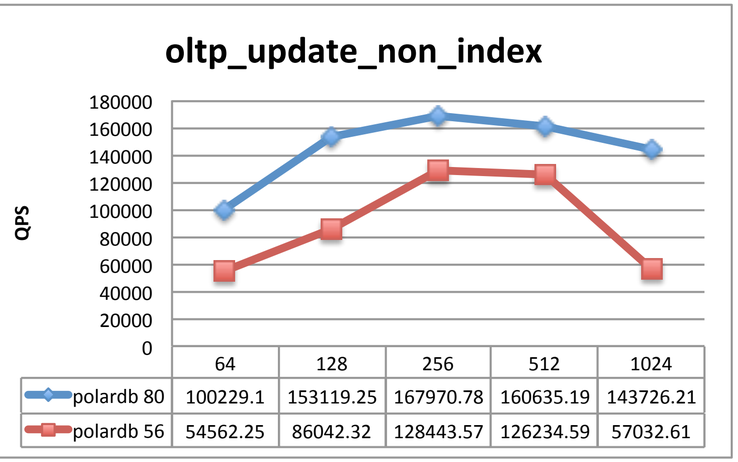
- increase slot size in IO thread
- increase background IO threads
- combine IO to support larger buffer IO

# POLARDB 8.0 VS POLARDB 5.6

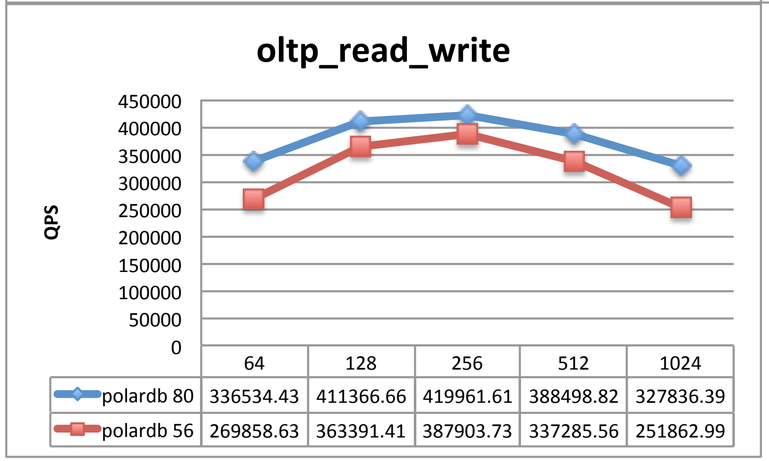
# POLARDB 8.0 VS POLARDB 5.6




| | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|
| polardb 80 | 88564.7 | 131969.35 | 159271.41 | 181438.98 | 181534.69 |
| polardb 56 | 10006.78 | 11796.06 | 86095.63 | 104572.46 | 47759.28 |




| | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|
| polardb 80 | 100229.1 | 153119.25 | 167970.78 | 160635.19 | 143726.21 |
| polardb 56 | 54562.25 | 86042.32 | 128443.57 | 126234.59 | 57032.61 |




| | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|
| polardb 80 | 336534.43 | 411366.66 | 419961.61 | 388498.82 | 327836.39 |
| polardb 56 | 269858.63 | 363391.41 | 387903.73 | 337285.56 | 251862.99 |

# Thank you