KEYSPACE

2025 / 北京

Valkey Dual Channel Replication

binbinzhu(朱彬彬) | 2025/12/13

腾讯云研发，Valkey TSC Member

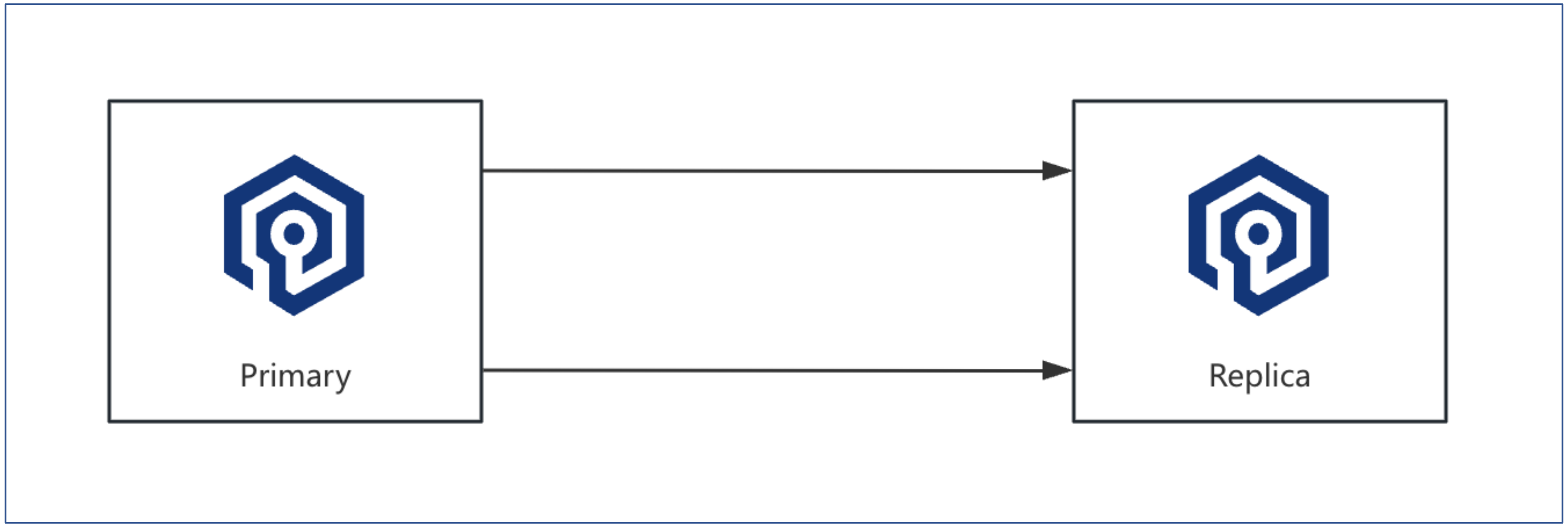
About me

- 2021 - 2024 Redis committer
- 2024 - now Valkey TSC member
- Software engineer at Tencent Cloud
- Github: github.com/enjoy-binbin
- Email: binloveplay1314@qq.com
- https://github.com/enjoy-binbin/enjoy-binbin/tree/main/2025-12-13-Valkey_Keyspace_2025_Beijing

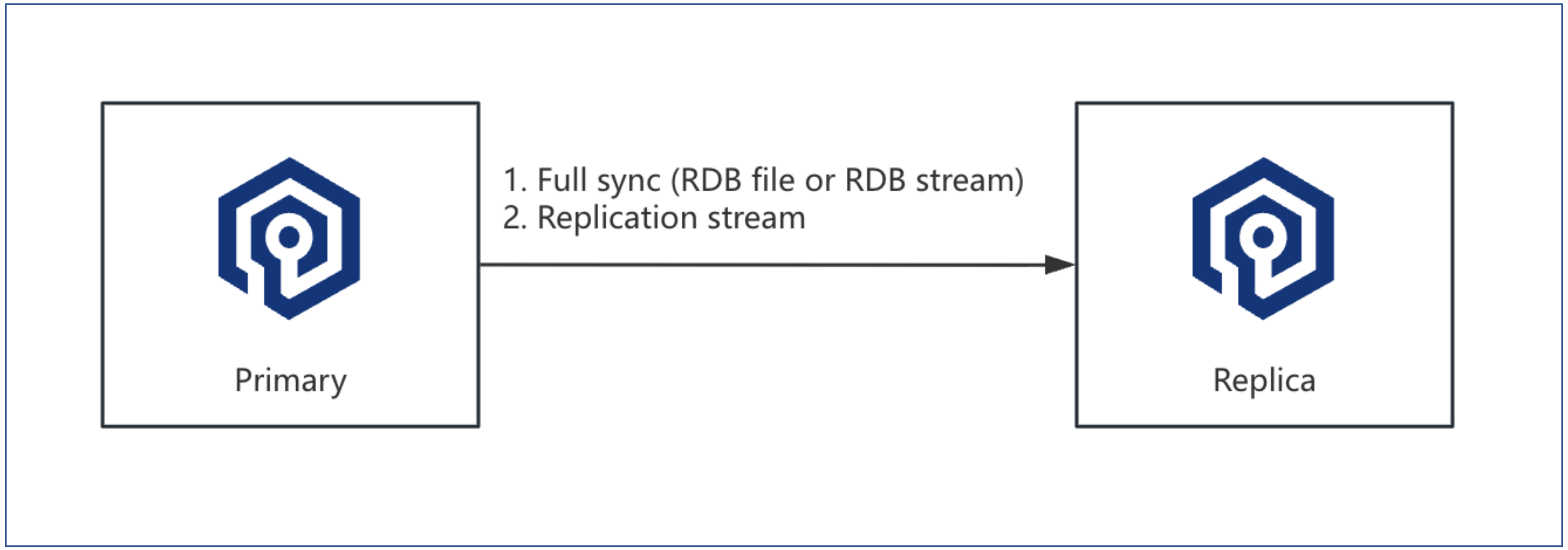
Agenda

- Introducing replication
- Full sync && Psync
- Replication backlog
- Replica client output buffer
- Replication memory usage
- Replication memory issue
- Dual channel replication
- Make valkey great together

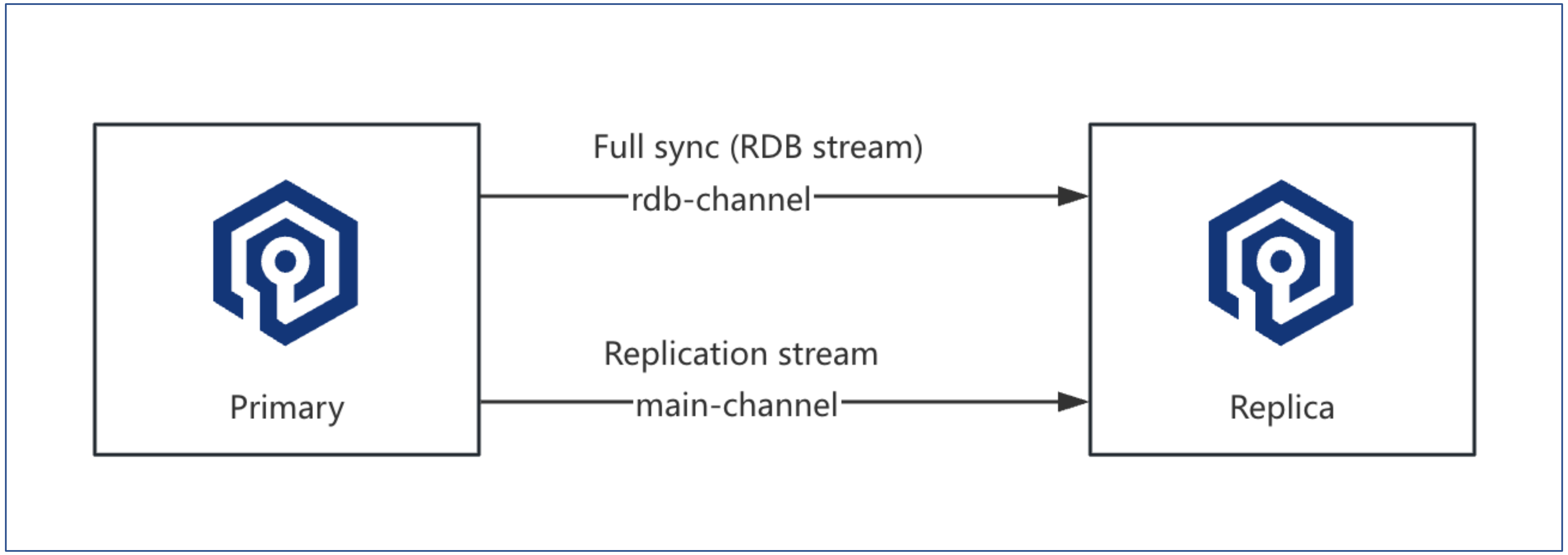
Dual channel replication overview - 1



Dual channel replication overview - 2

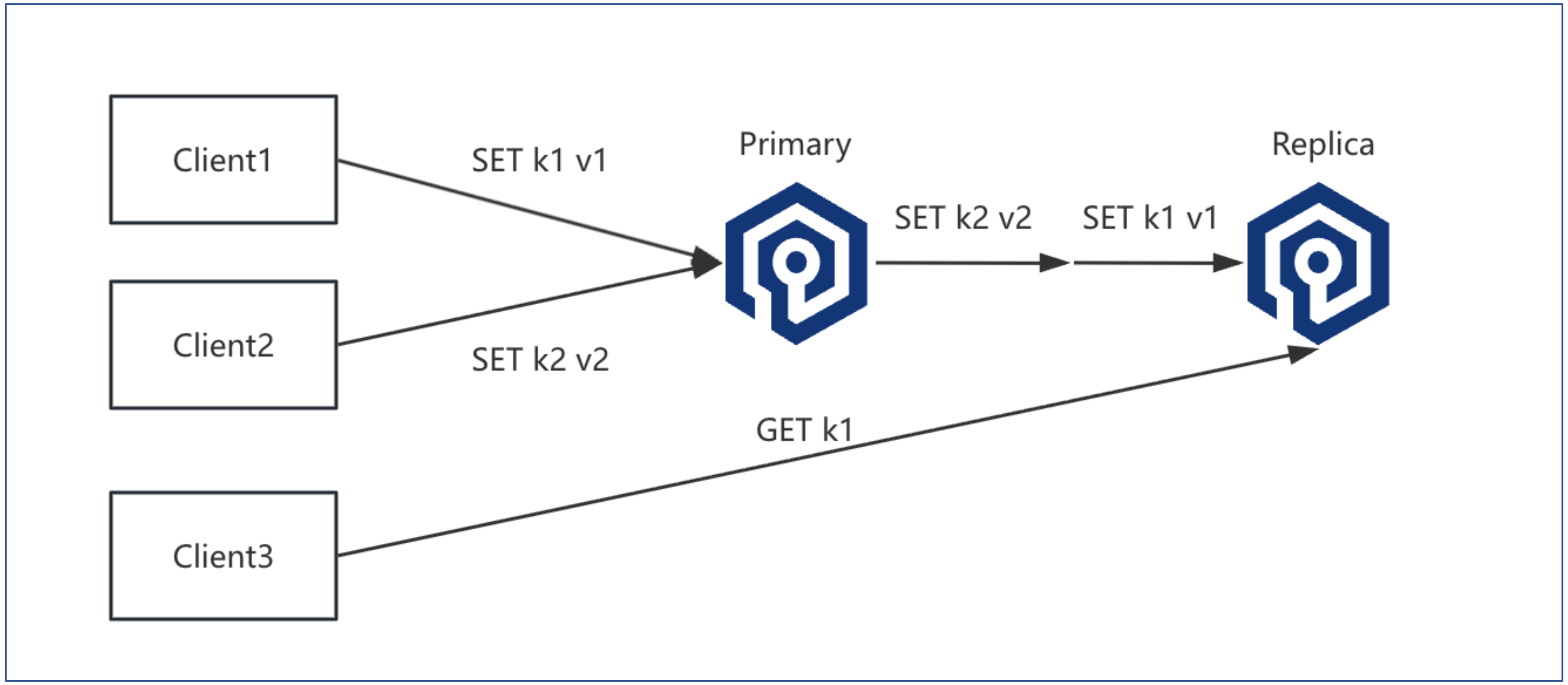


Dual channel replication overview - 3



Dual channel replication #60 - https://github.com/valkey-io/valkey/pull/60

Replication Basic



https://valkey.io/topics/replication/

Replication Key Points

- 主从复制是异步的
  - 副本与主服务器之间会异步确认复制偏移量
- 主从复制在主节点侧是非阻塞的
  - 主节点不会等待副本节点完成同步
- 主从复制在副本节点侧是可能阻塞的
  - 副本节点视角里主节点客户端和普通客户端没太多区别
- 主节点可以拥有任意数量的副本节点
  - 主节点视角里副本节点客户端和普通客户端没太多区别
- 通常副本节点是只读的
  - 标准版模式下可配置可写副本
- 在标准版模式下，副本也可以是别的副本的主节点

Need for Replication

- 高可用
  - 主节点宕机时有副本节点顶上恢复服务
- 扩展实例的读性能
  - 读写分离将读请求打到副本节点上
- 提高数据安全性
  - 主节点宕机时有副本节点顶上恢复数据
- 另一种方式的持久化
  - 主节点不配置持久化，副本节点配置持久化

Type of Replication

- full sync (full synchronization)
    - 全量同步
- psync (partial resynchronizations)
    - 部分重新同步 / 增量同步

Full sync

- full sync (full synchronization / full resynchronization)
  - 顾名思义，就是将主节点上的全部数据同步给副本节点

- full sync 是怎么在不影响客户端访问下实现的？
  - fork 子进程生成数据快照
  - replica client output buffer 传输增量变更

- 怎么传输 RDB 快照？
  - 有盘复制：子进程生成 RDB 文件，主进程发送 RDB 文件给副本节点
  - 无盘复制（< 6.0）：子进程边生成 RDB 快照边将内容发送给副本节点
  - 无盘复制（>= 6.0）：子进程边生成 RDB 快照边将内容通过匿名管道发送给主进程，主进程边接收边将内容发送给副本节点

- 什么情况下会发生 full sync？
  - 添加一个新副本
  - 现有副本重连后没办法做 psync

Psync

- psync (partial sync / partial resynchronizations)
  - 副本在断连重连时，会尝试使用复制偏移量信息以进行部分全同步，以避免全量同步
  - 主节点在副本重连后，会尝试使用复制积压缓冲区传输副本缺失的数据，以避免全量同步

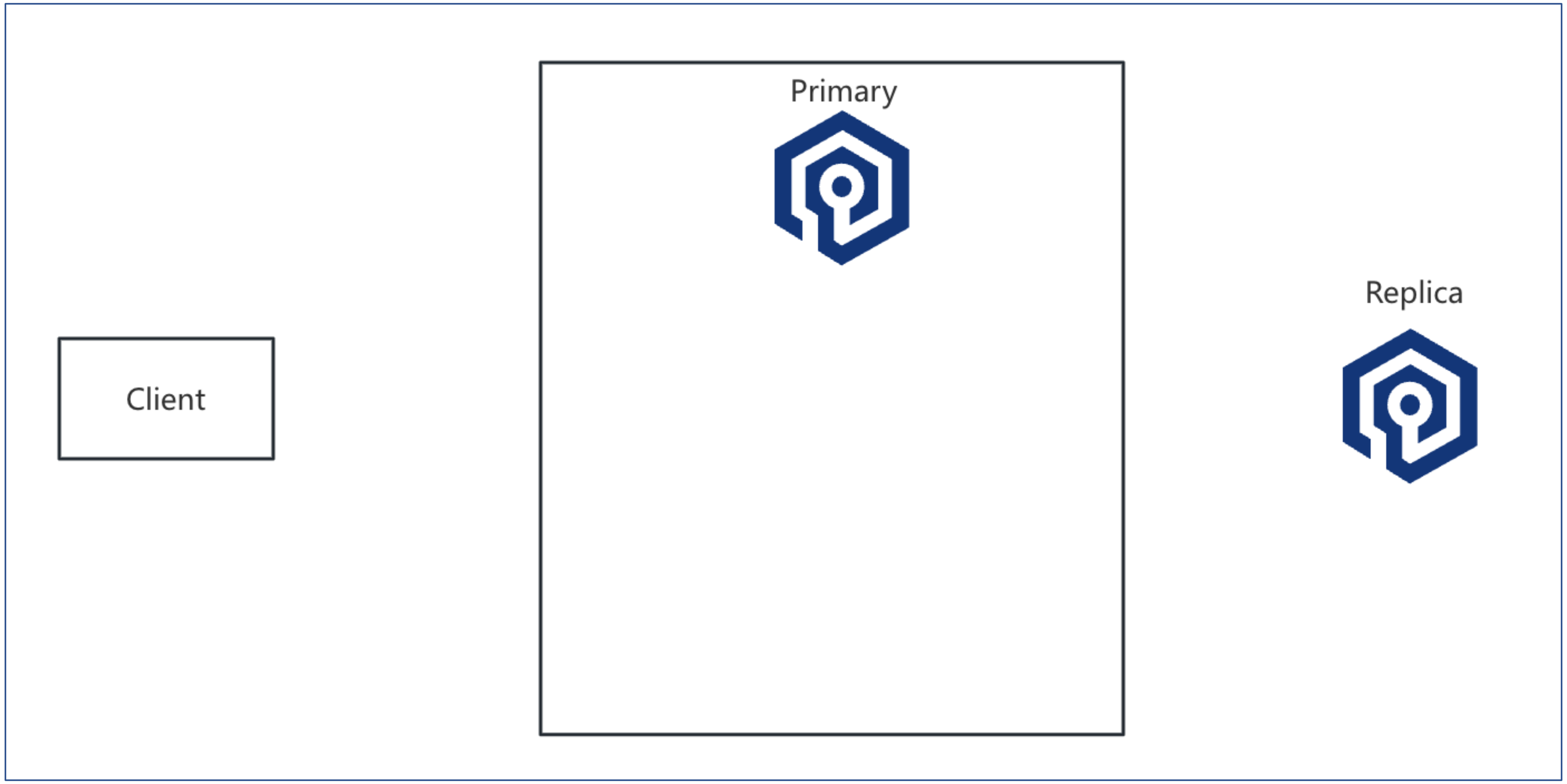
- Replication backlog
  - 复制积压缓冲区，固定大小，缓存追踪最近的增量更新

- Replication offset
  - 复制偏移量，逻辑下标对应每一个字节。复制偏移量和复制 ID 是密切相关的。

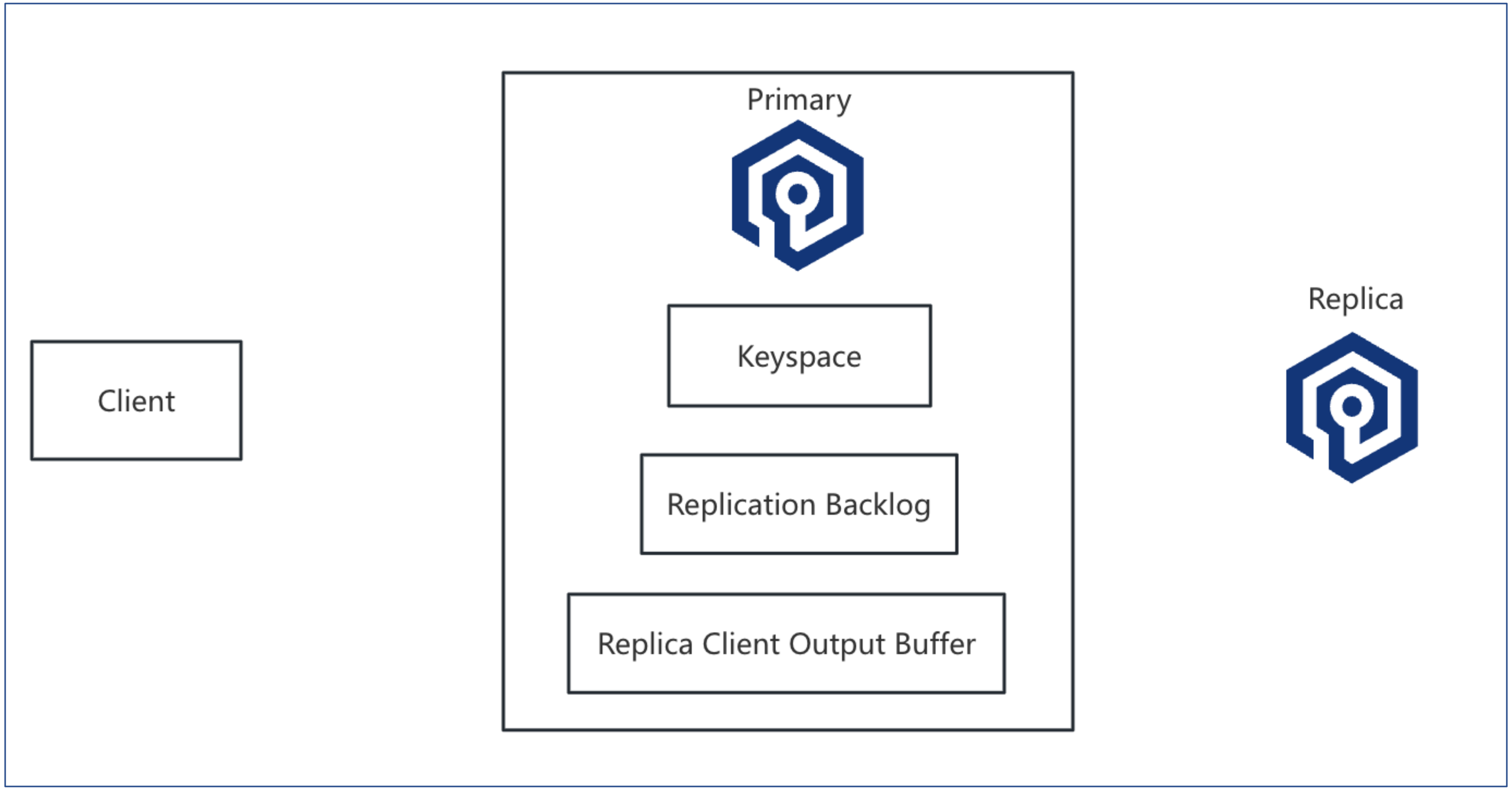
- Replication ID
  - 复制 ID，用来标识复制历史

- Replica client output buffer
  - 副本客户端输出缓冲区，传输变更

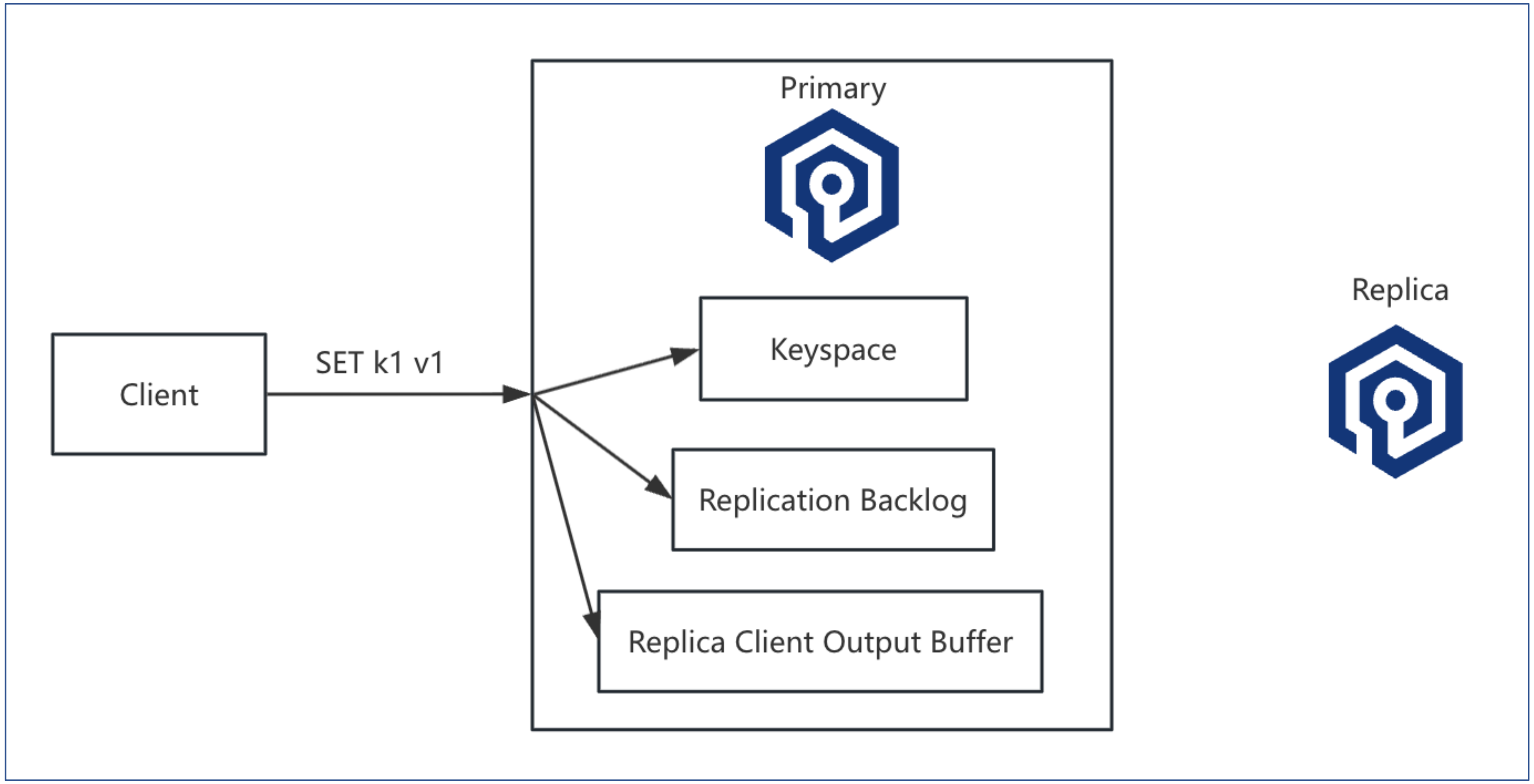
Putting it together - 1



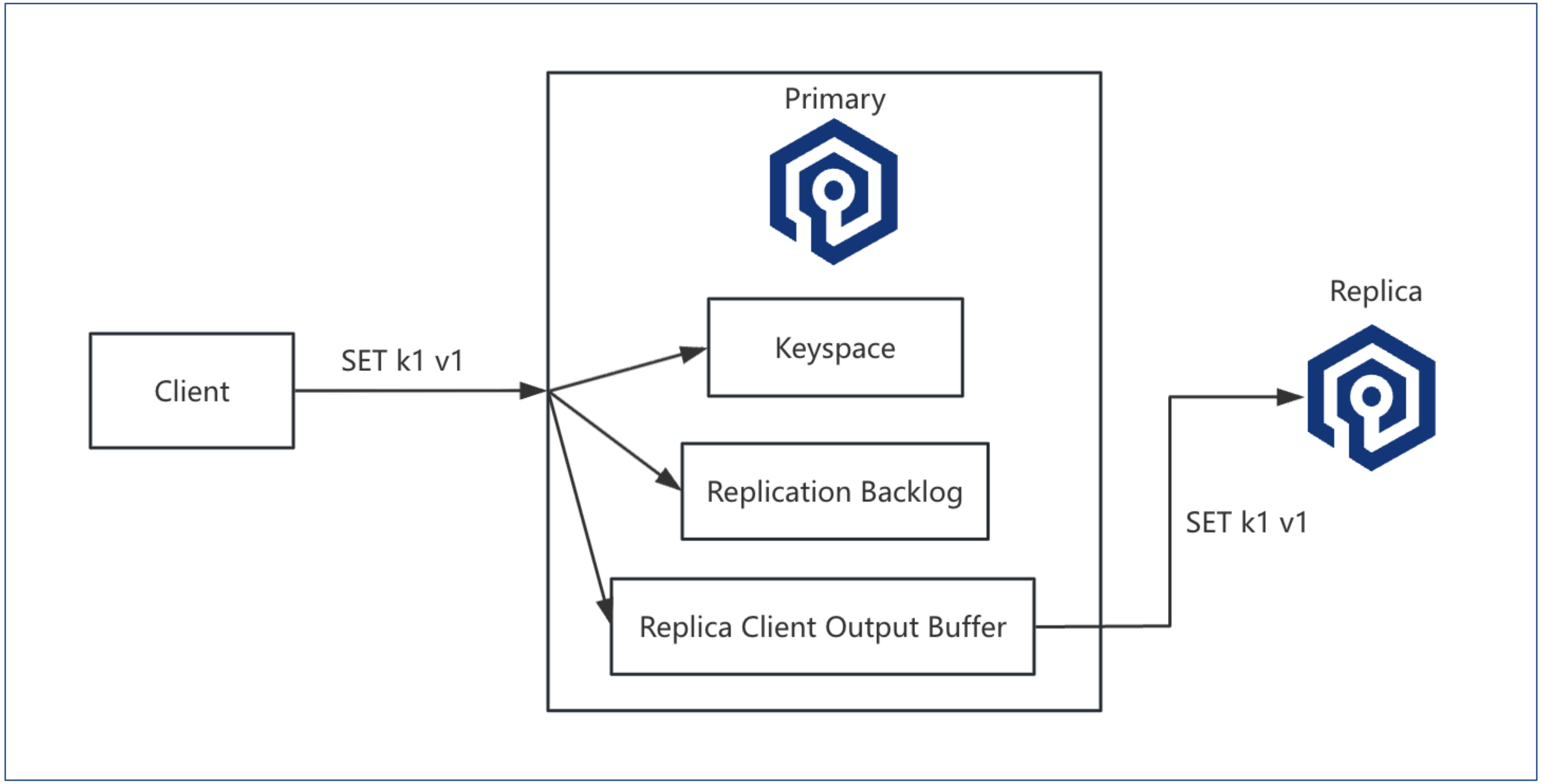
Putting it together - 2



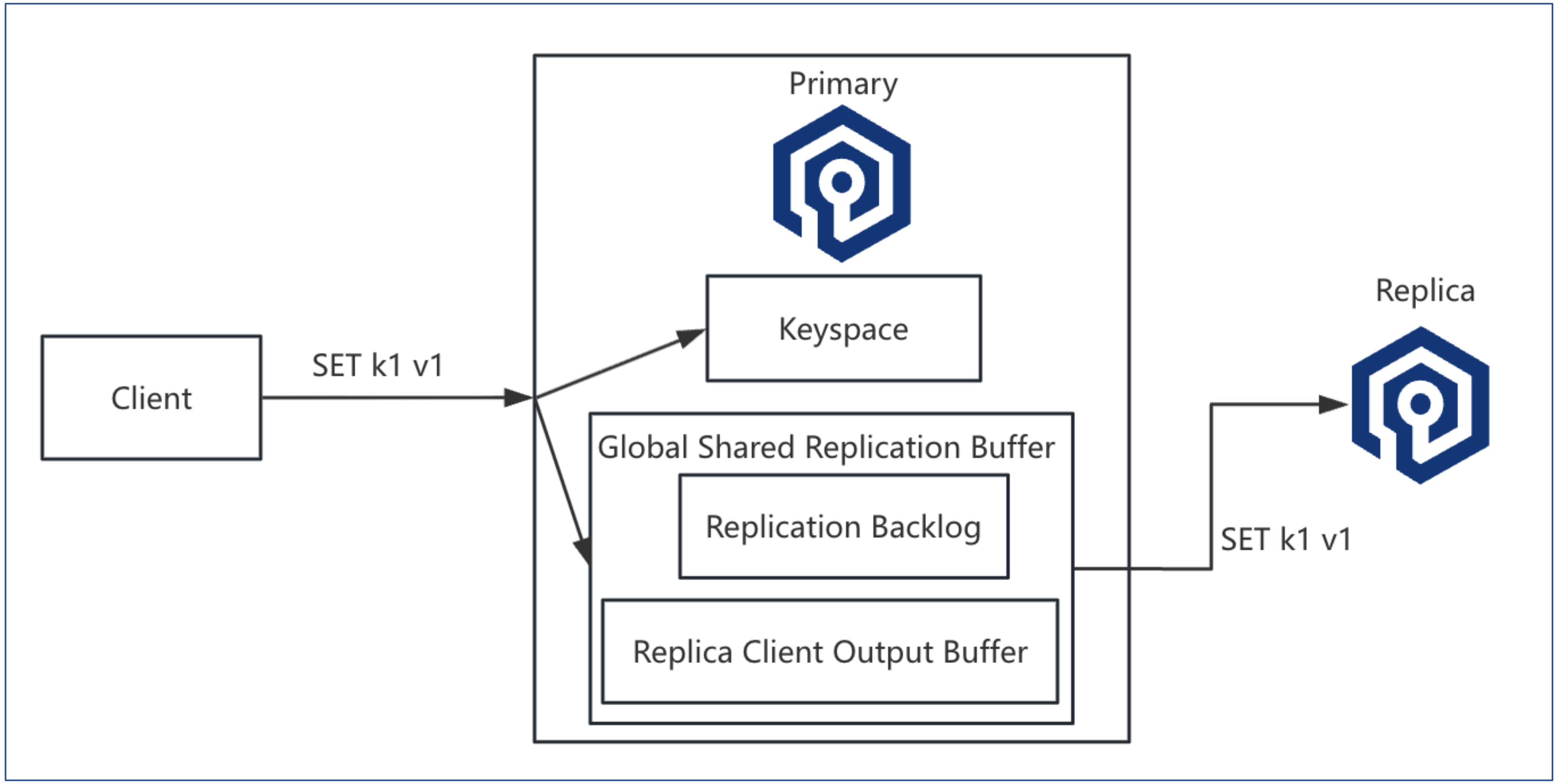
Putting it together - 3



Putting it together - 4



Putting it together - 5



Replication backlog and replicas use one global shared replication buffer #9166

https://github.com/redis/redis/pull/9166

Replica client output buffer

- Valkey 是内存数据库
    - 所有数据都是存储在内存上
    - 从内存里读数据很快
    - 往网络里写数据很慢
    - 需要能处理变长响应

- 客户端输出缓冲区：
    - 要发送给客户端的数据会先写入到客户端对应的输出缓冲区中
    - 断连慢速客户端
    - client-output-buffer-limit normal <hard limit> <soft limit> <soft seconds>

- 副本客户端输出缓冲区：
    - 副本在主节点视角里也是一个客户端，主节点通过网络向副本客户端写入数据
    - 断连慢速副本客户端
    - client-output-buffer-limit replica <hard limit> <soft limit> <soft seconds>

Replication Backlog - 1

- replication backlog
  - 复制积压缓冲区

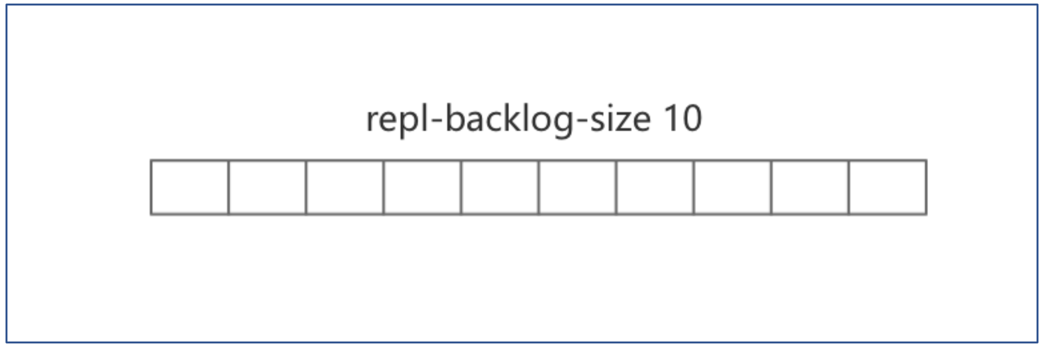
- repl_backlog_size
  - 复制积压缓冲区可缓存的字节大小，通过配置项 repl-backlog-size 设置

- master_repl_offset
  - 主节点的复制偏移量，从 0 开始，主节点写入的每一个字节，都会响应增加这个复制偏移量

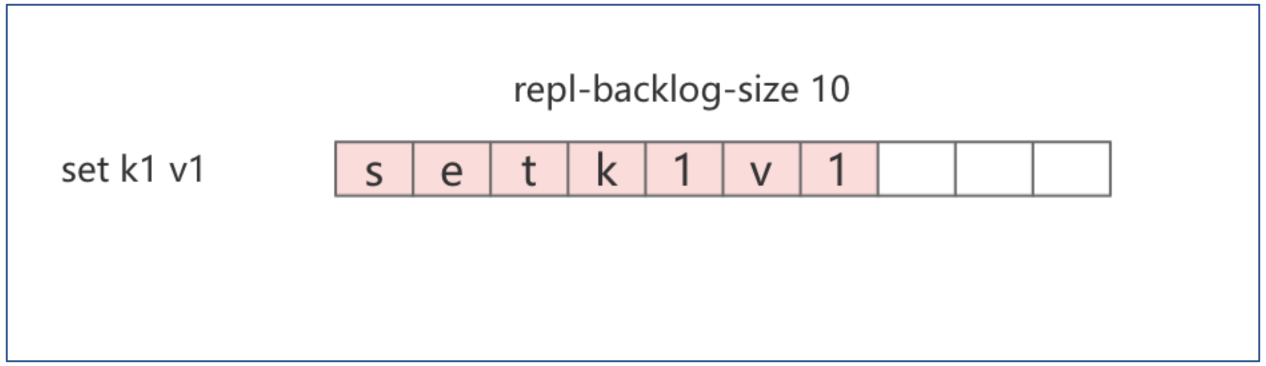
- repl_backlog_first_byte_offset
  - 复制积压缓冲区里第一个字节所对应的主节点复制偏移量。它从 1 开始，直到 backlog 被填满，当在 backlog 里产生覆盖写时，它的值才会开始增加

- repl_backlog_histlen
  - 当前复制积压缓冲区的积累的数据字节大小，或者就复制积压缓冲区里实际的/有效的数据长度
  - 有效数据始终位于第一个字节偏移量和主节点复制偏移量之间

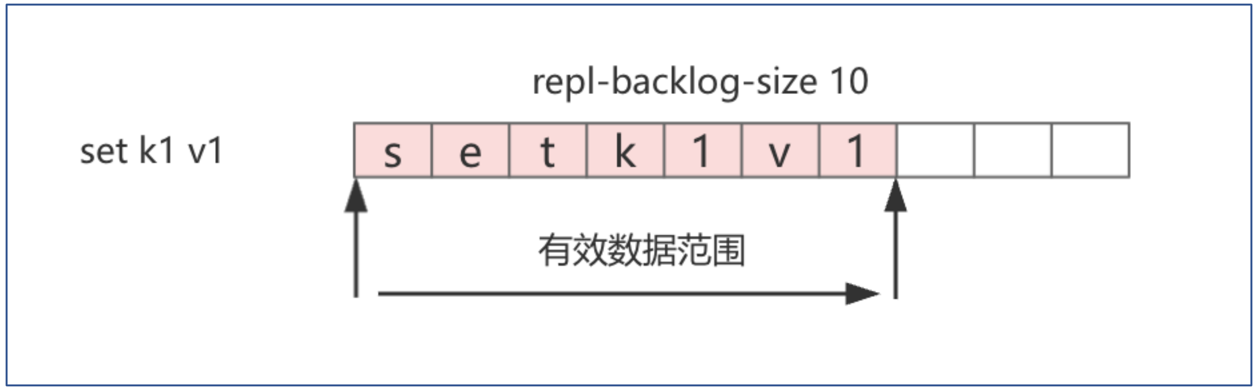
Replication Backlog - 2



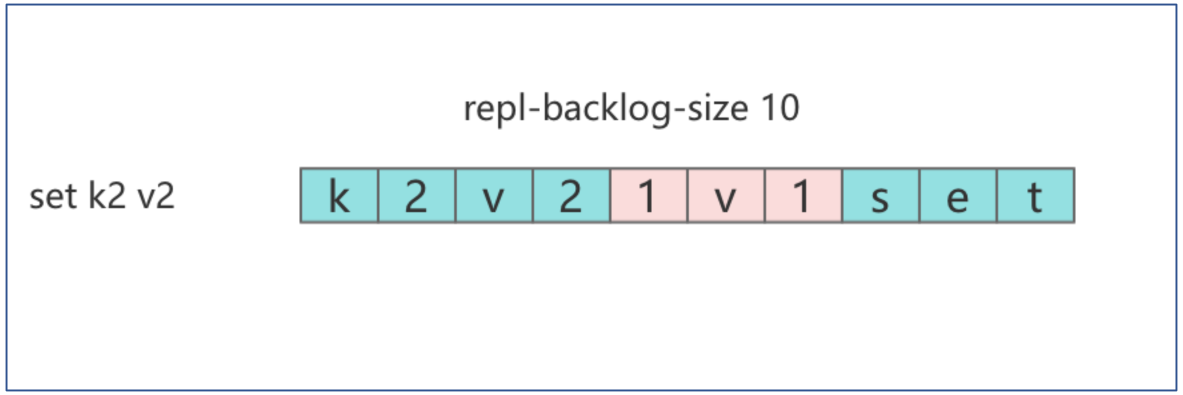
Replication Backlog - 3



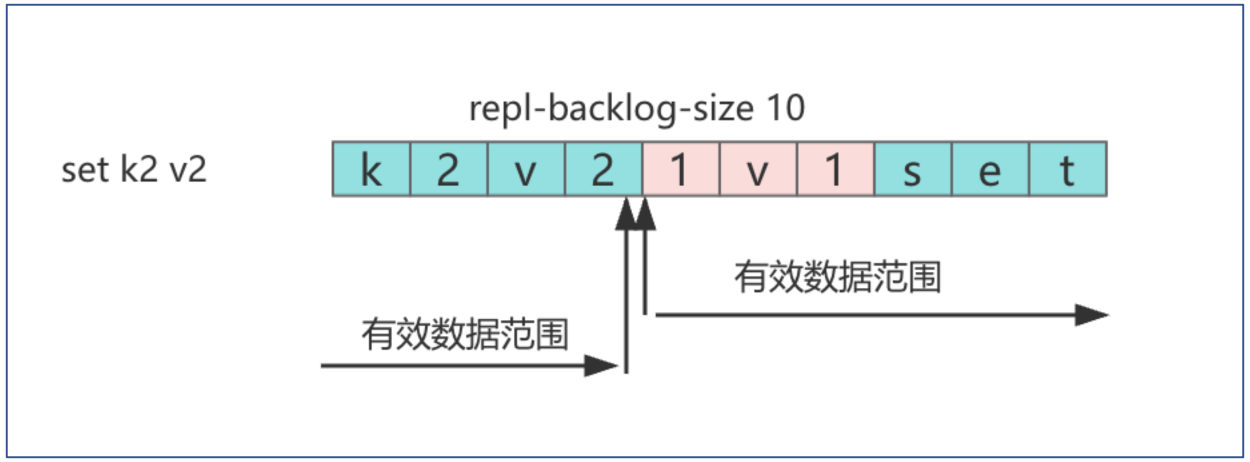
Replication Backlog - 4



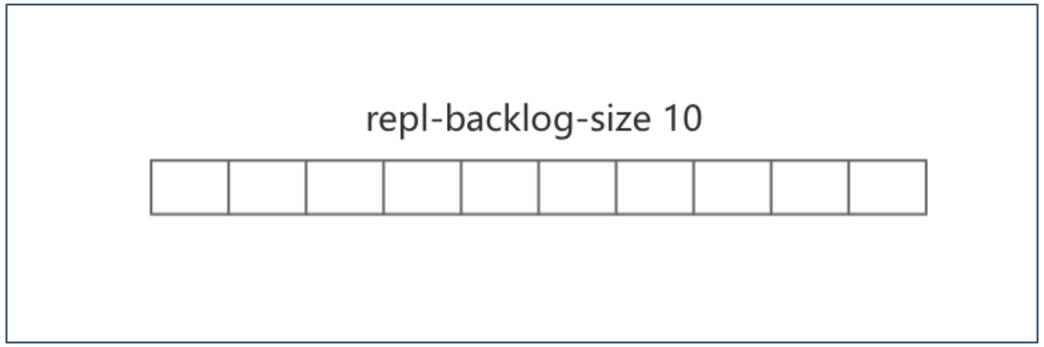
Replication Backlog - 5



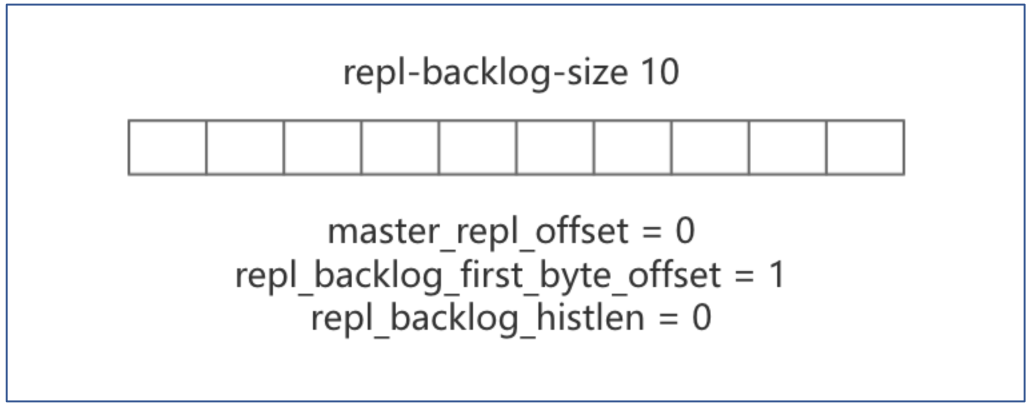
Replication Backlog - 6



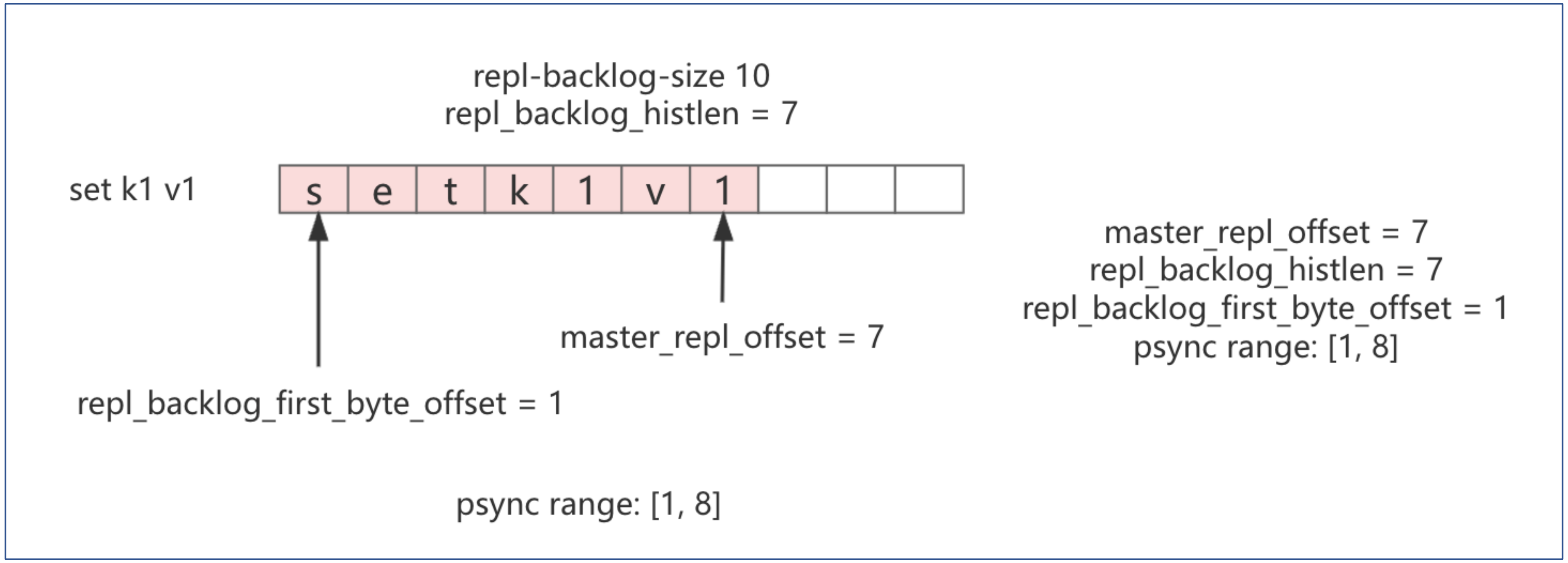
Replication Backlog - 7



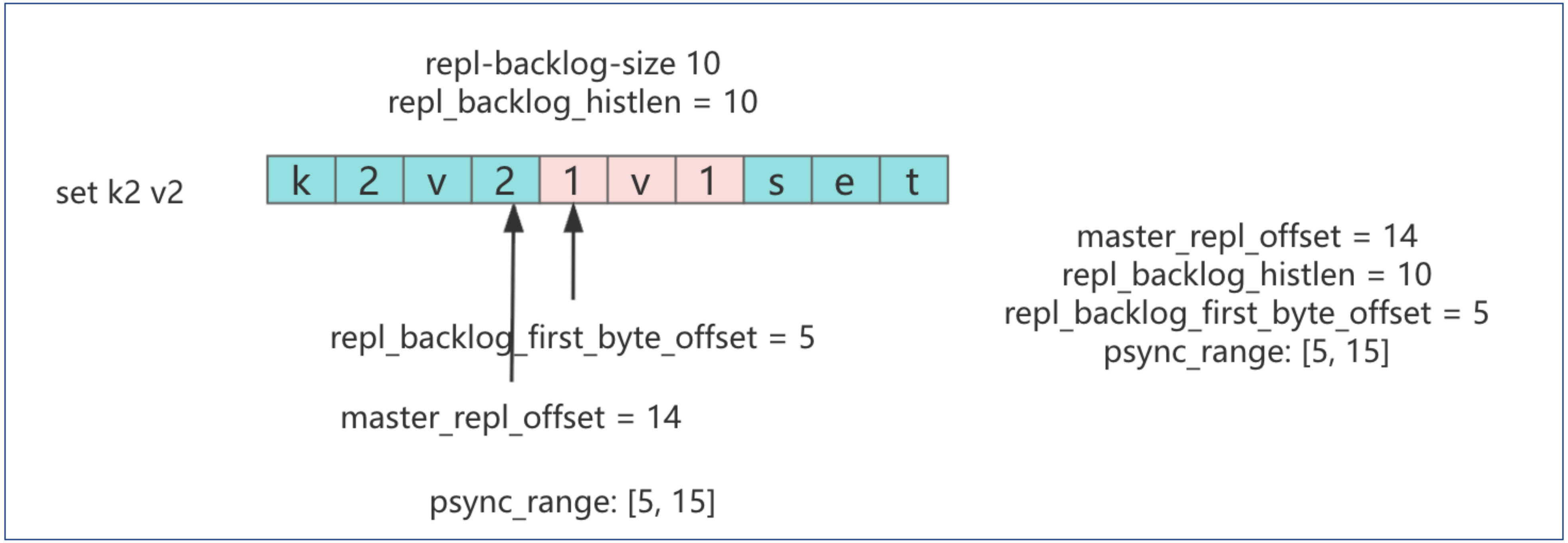
Replication Backlog - 8



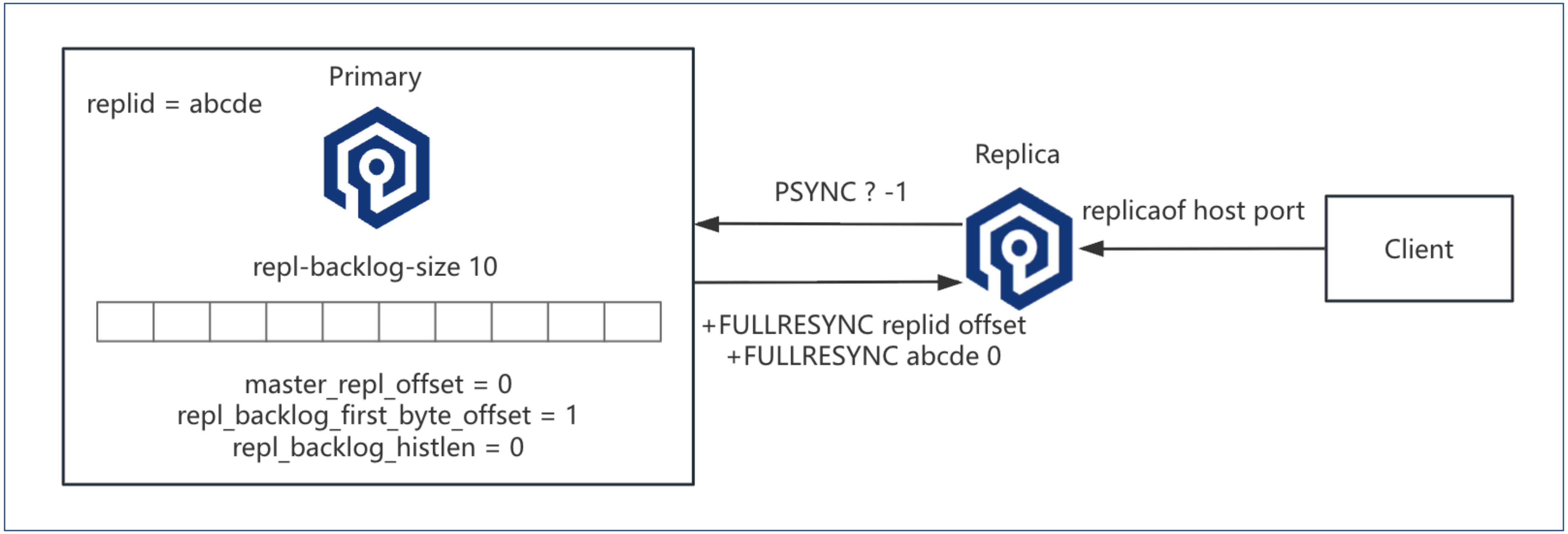
Replication Backlog - 9



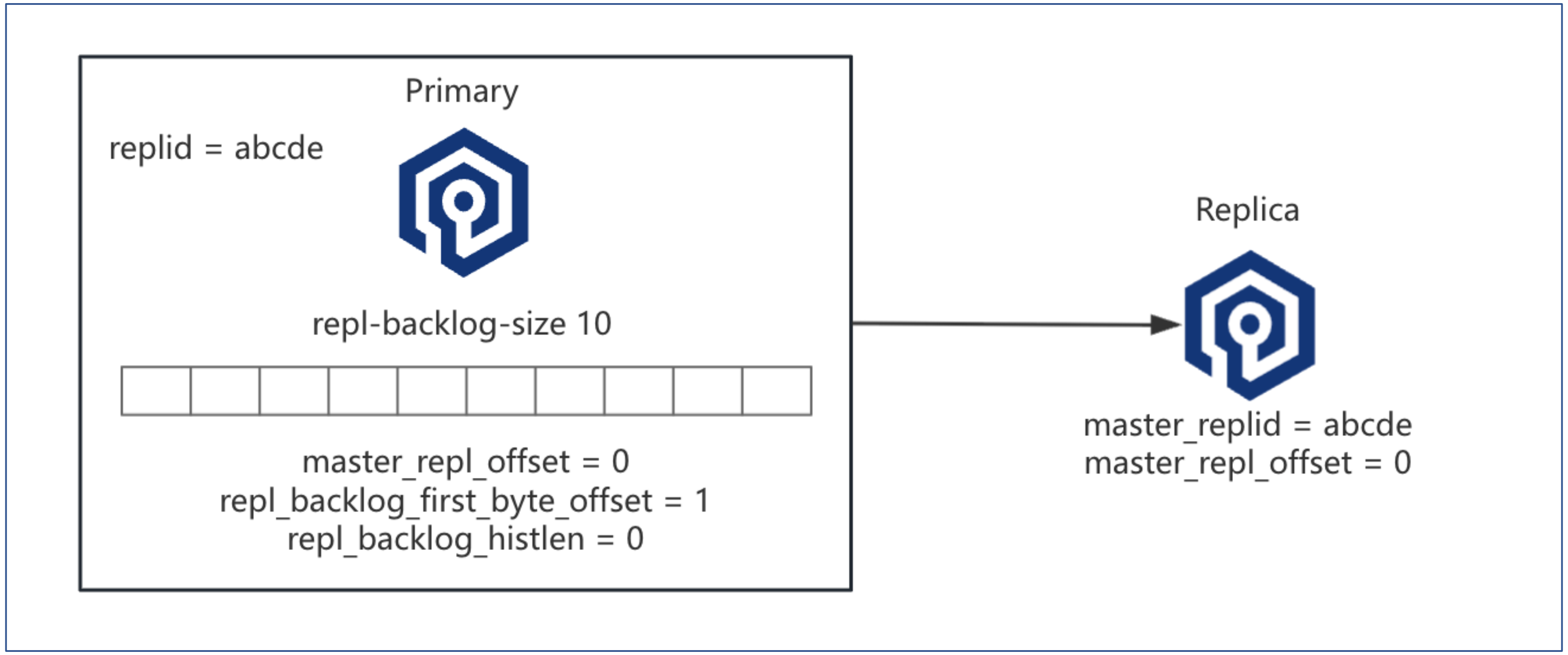
Replication Backlog - 10



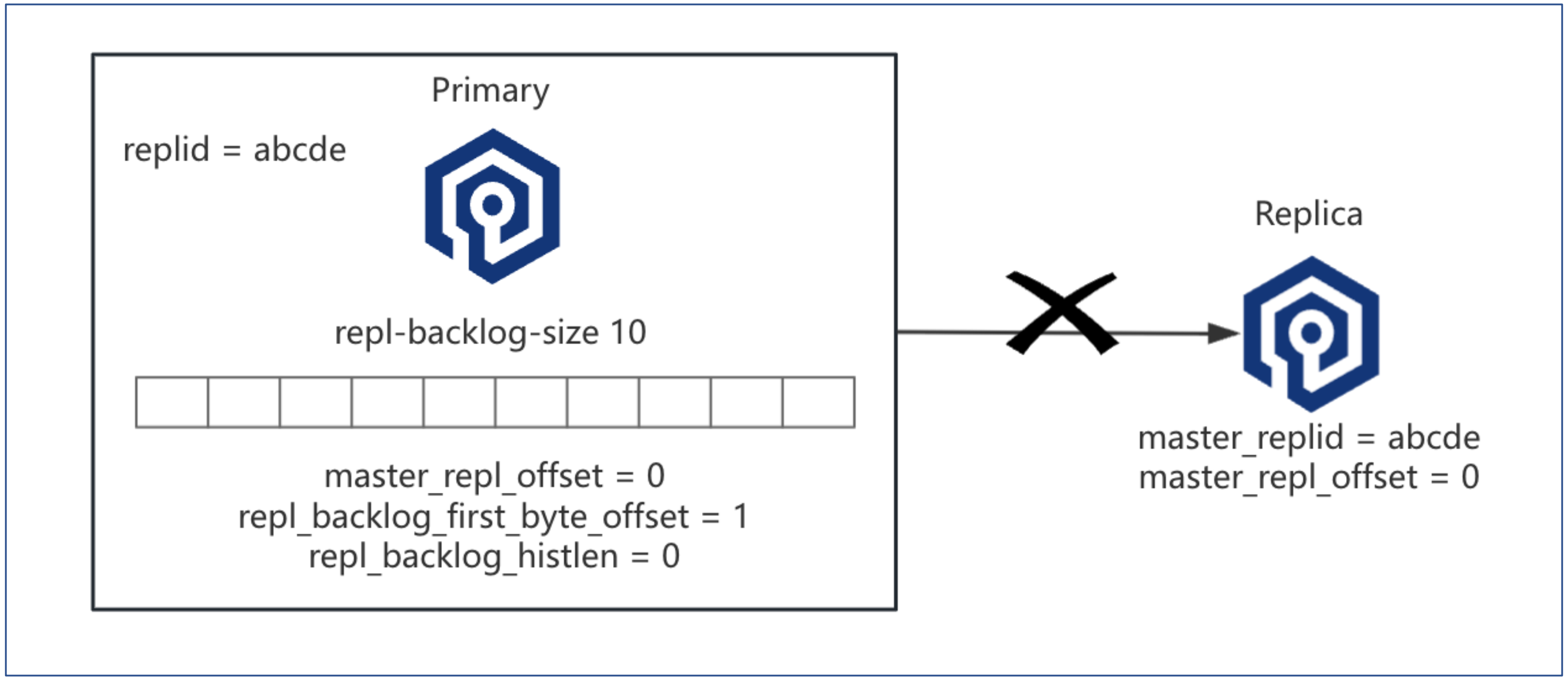
replicaof host port



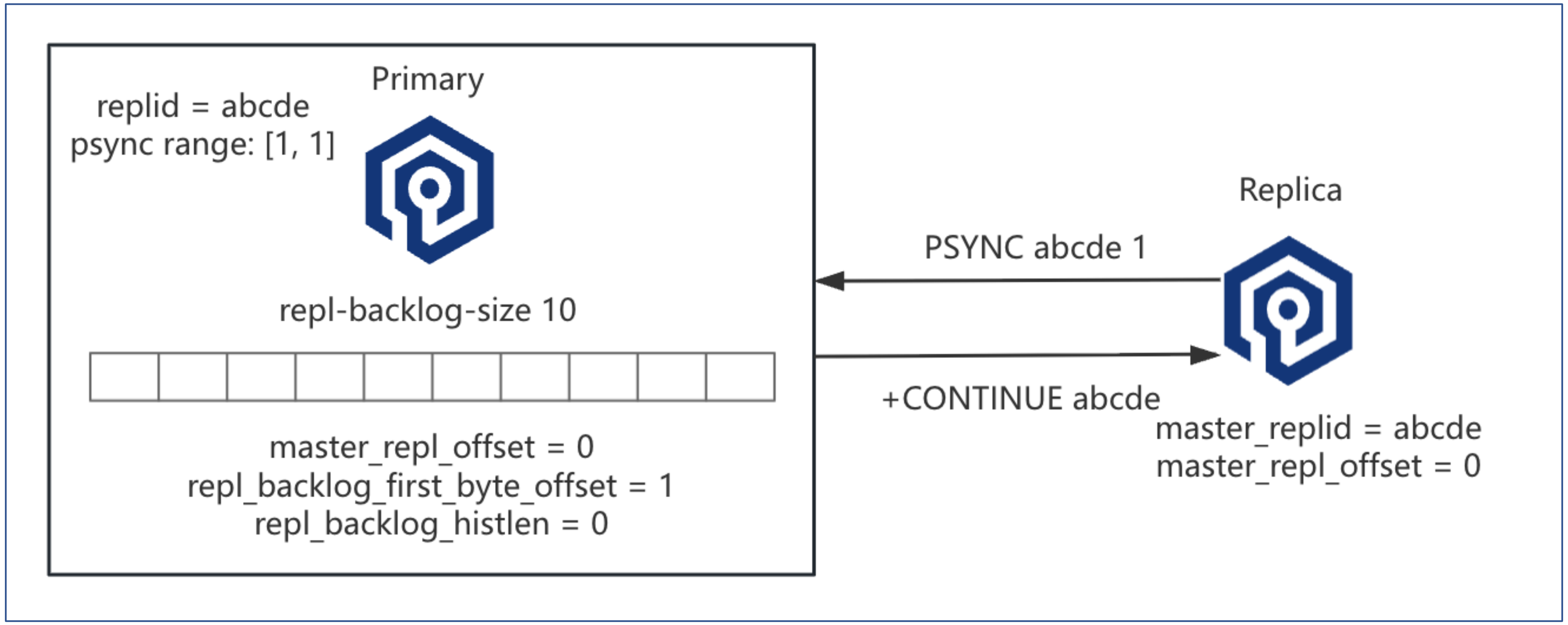
Full sync


Primary
replid = abcde
repl-backlog-size 10
master_repl_offset = 0
repl_backlog_first_byte_offset = 1
repl_backlog_histlen = 0
Replica
master_replid = abcde
master_repl_offset = 0

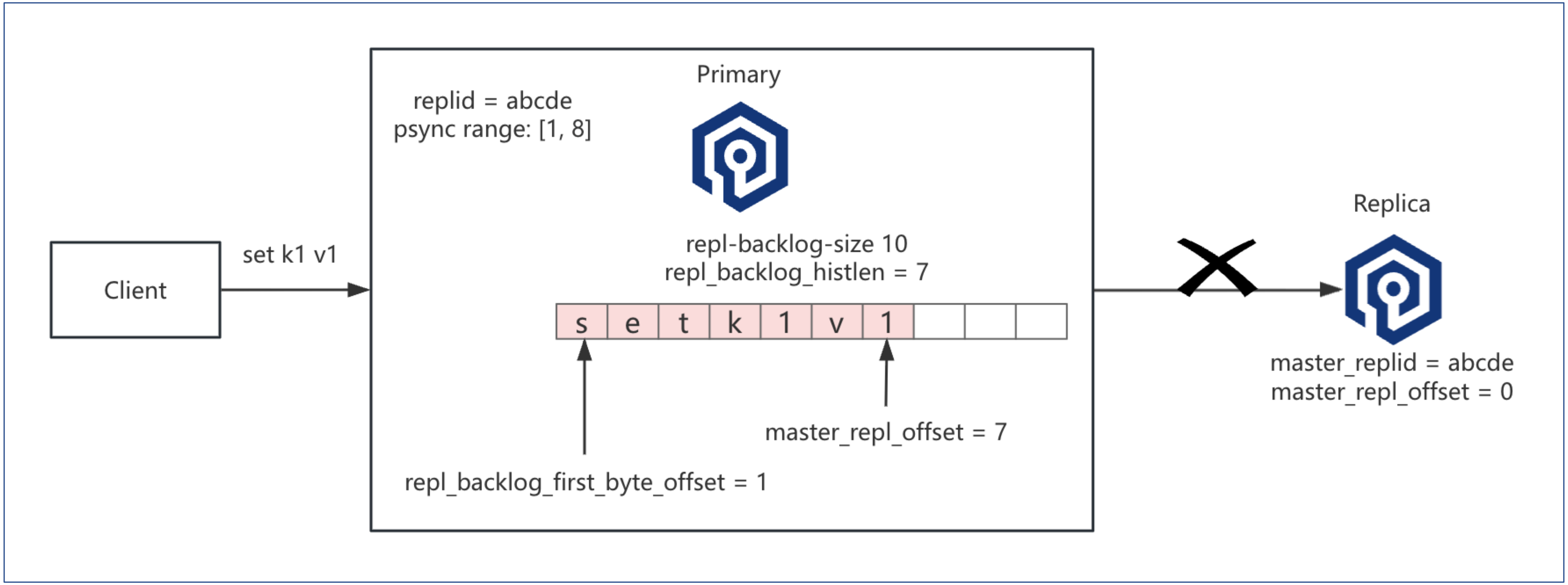
Disconnect


Primary
replid = abcde
repl-backlog-size 10
master_repl_offset = 0
repl_backlog_first_byte_offset = 1
repl_backlog_histlen = 0
Replica
master_replid = abcde
master_repl_offset = 0

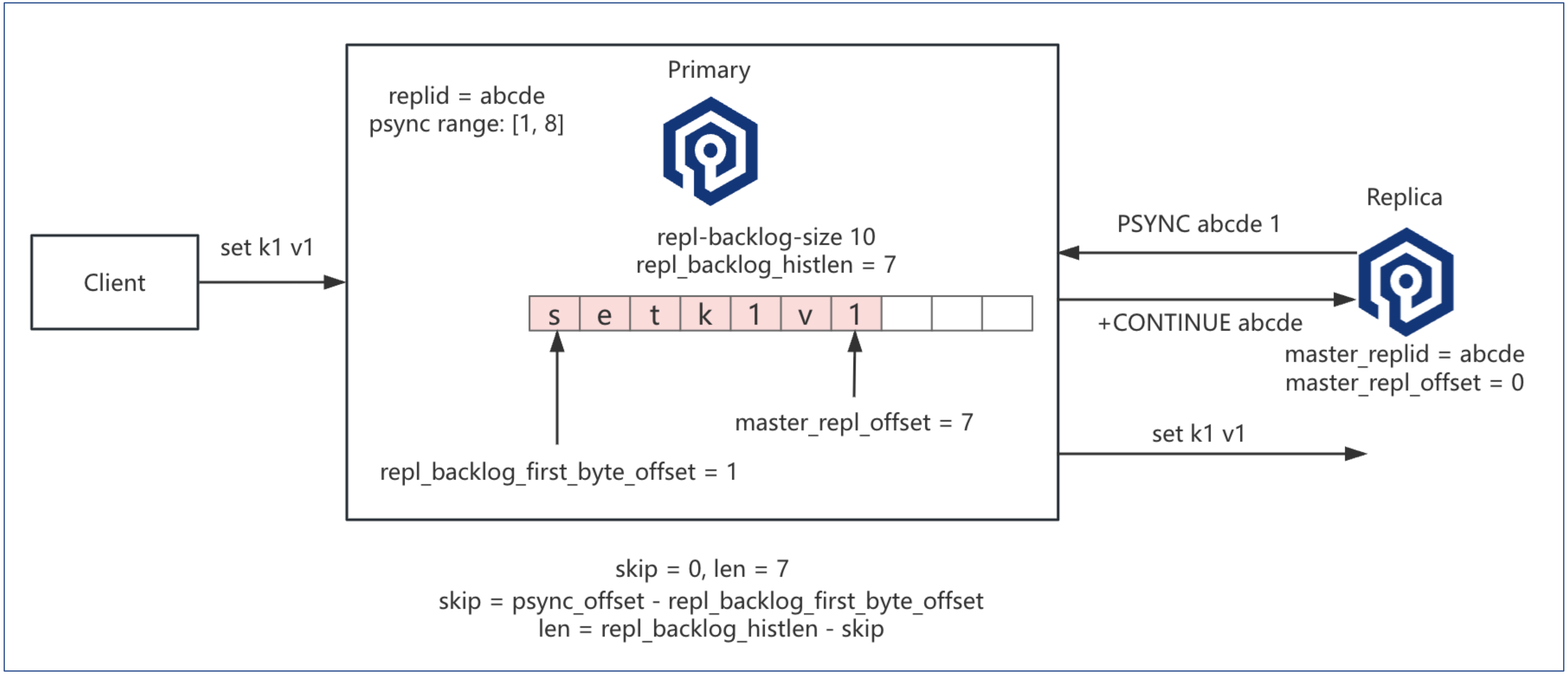
Psync


Primary
replid = abcde
psync range: [1, 1]
repl-backlog-size 10
master_repl_offset = 0
repl_backlog_first_byte_offset = 1
repl_backlog_histlen = 0
Replica
PSYNC abcde 1
+CONTINUE abcde
master_replid = abcde
master_repl_offset = 0

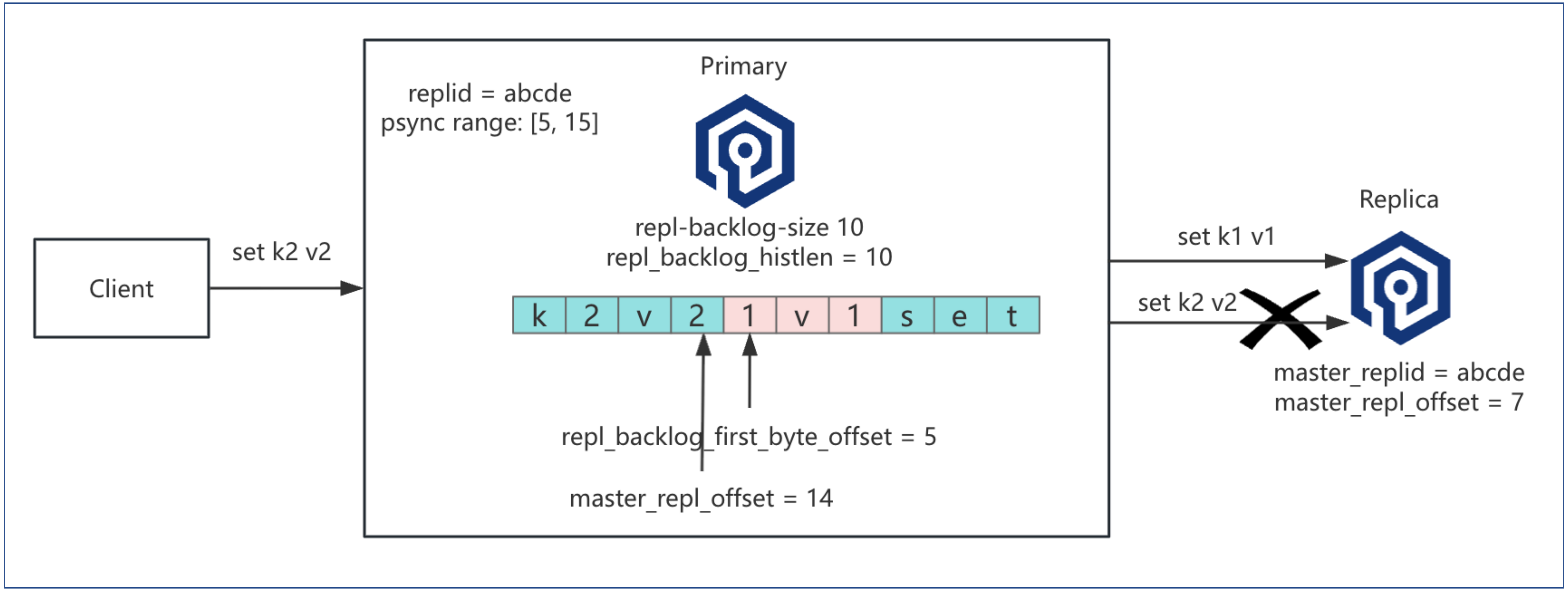
How does psync work? - 1



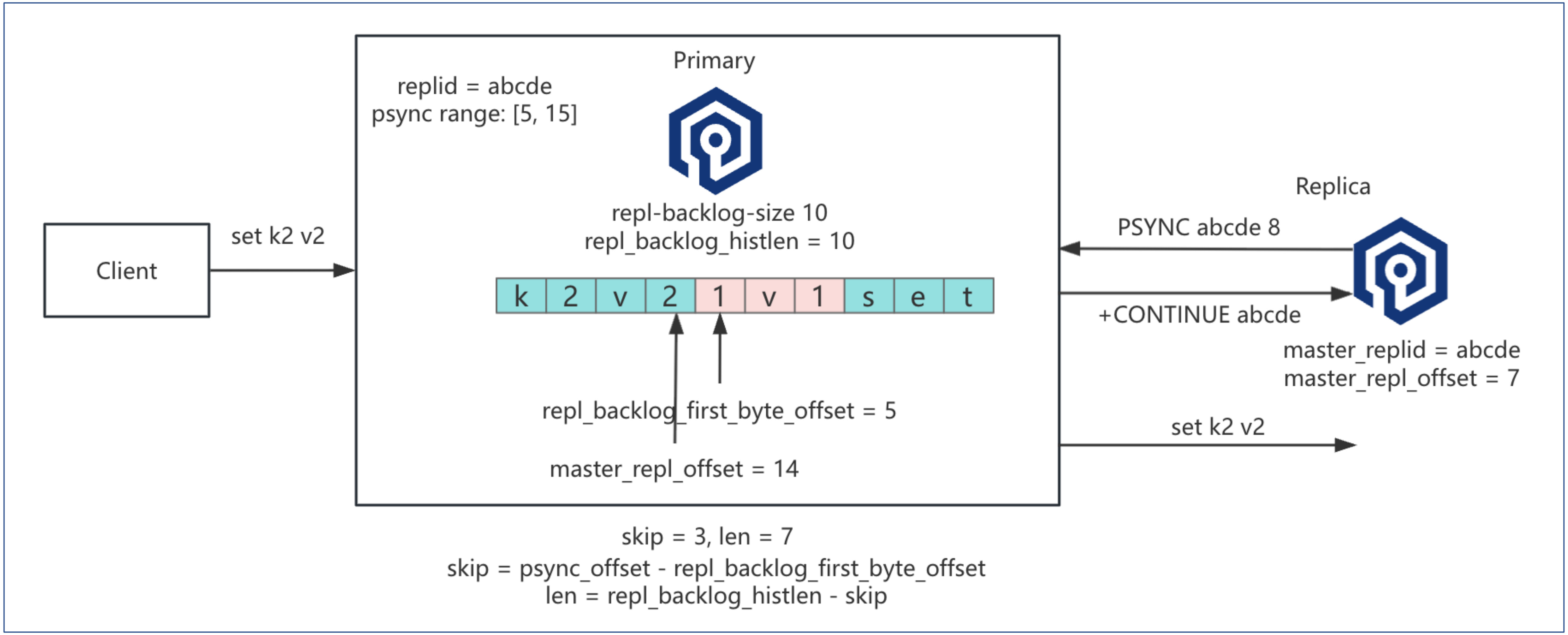
How does psync work? - 2



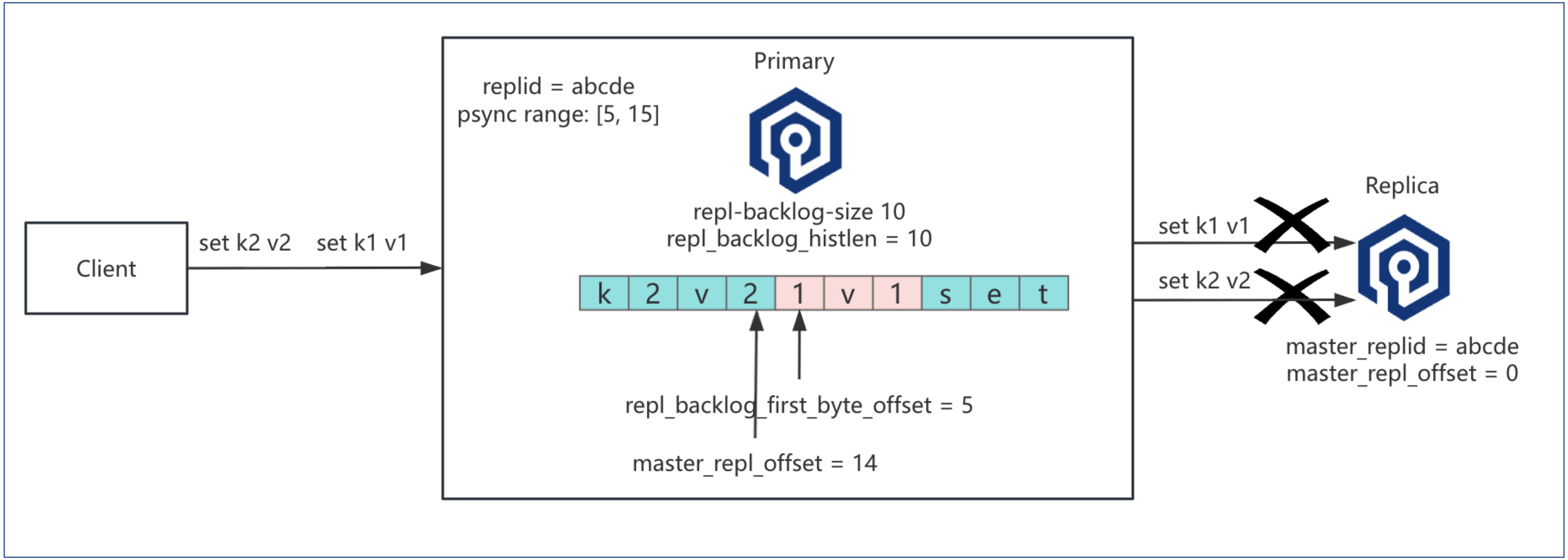
How does psync work? - 3



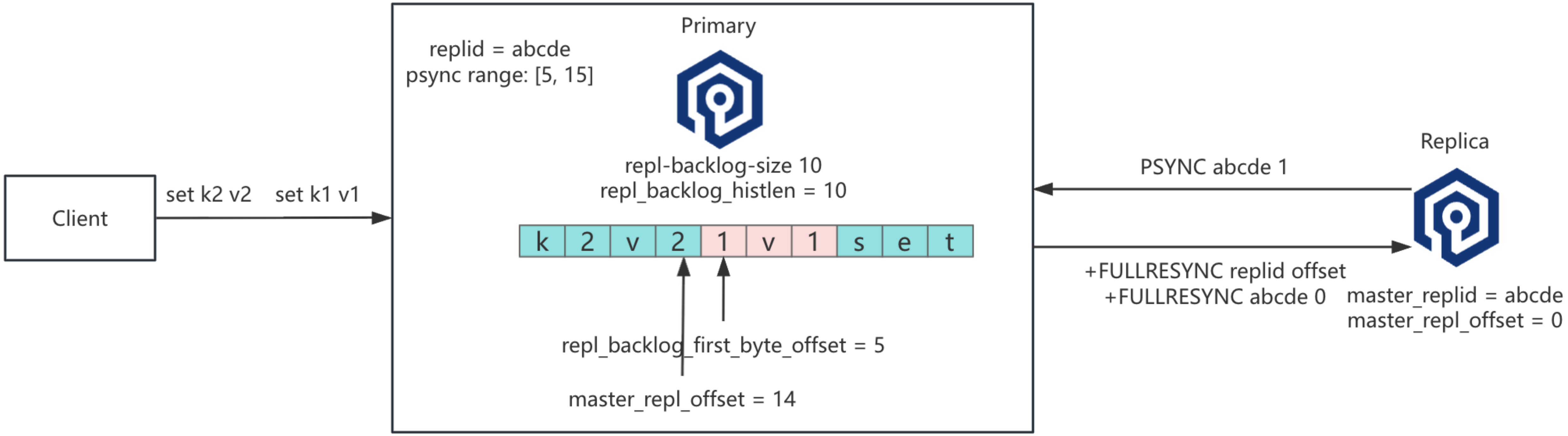
How does psync work? - 4



How does psync work? - 5



How does psync work? - 6



Configuration Items

- repl-backlog-size <size>
  - >= Valkey 8.0 default is 10mb
  - < Valkey 8.0 default is 1mb

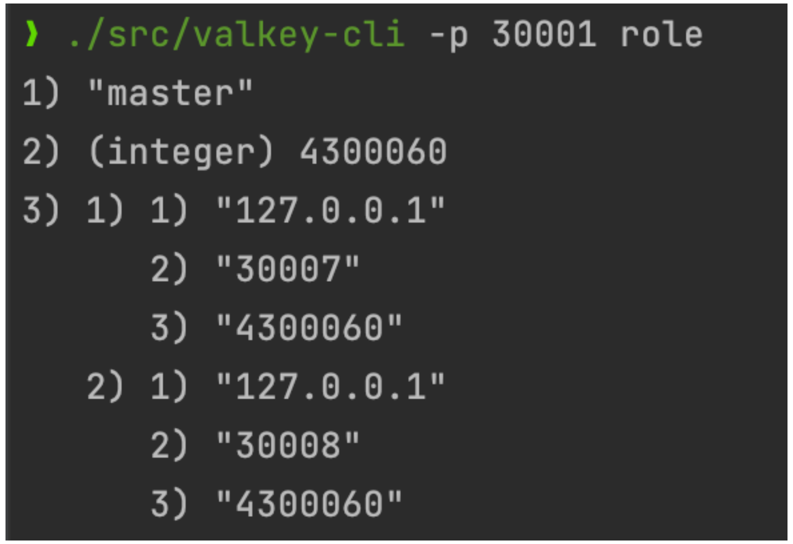
- client-output-buffer-limit <class> <hard limit> <soft limit> <soft seconds>
  - client-output-buffer-limit replica 256mb 64mb 60

Monitoring items

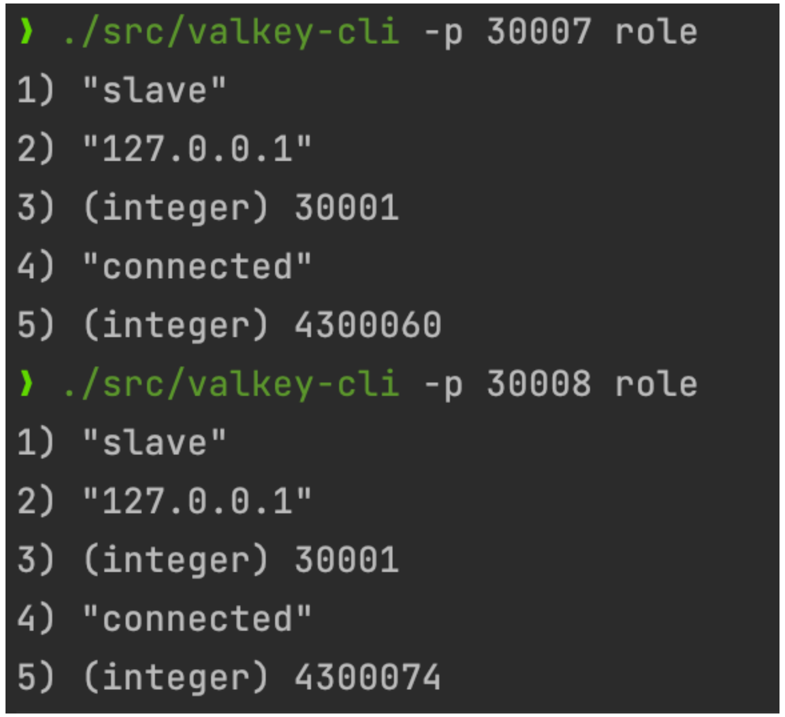
- role
  - https://valkey.io/commands/role/

- info replication
  - https://valkey.io/commands/info/

role command

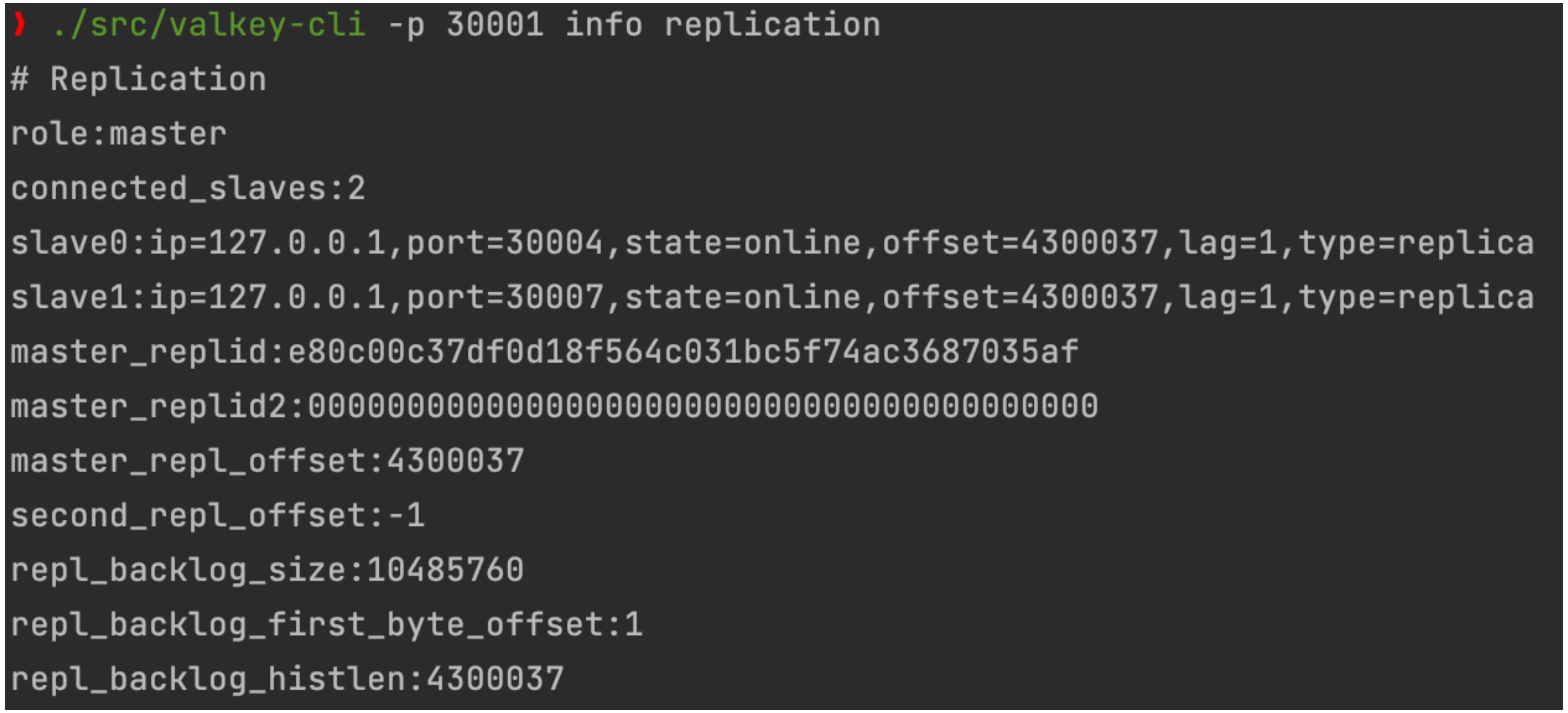


```
❱ ./src/valkey-cli -p 30001 role
1) "master"
2) (integer) 4300060
3) 1) 1) "127.0.0.1"
      2) "30007"
      3) "4300060"
   2) 1) "127.0.0.1"
      2) "30008"
      3) "4300060"
```



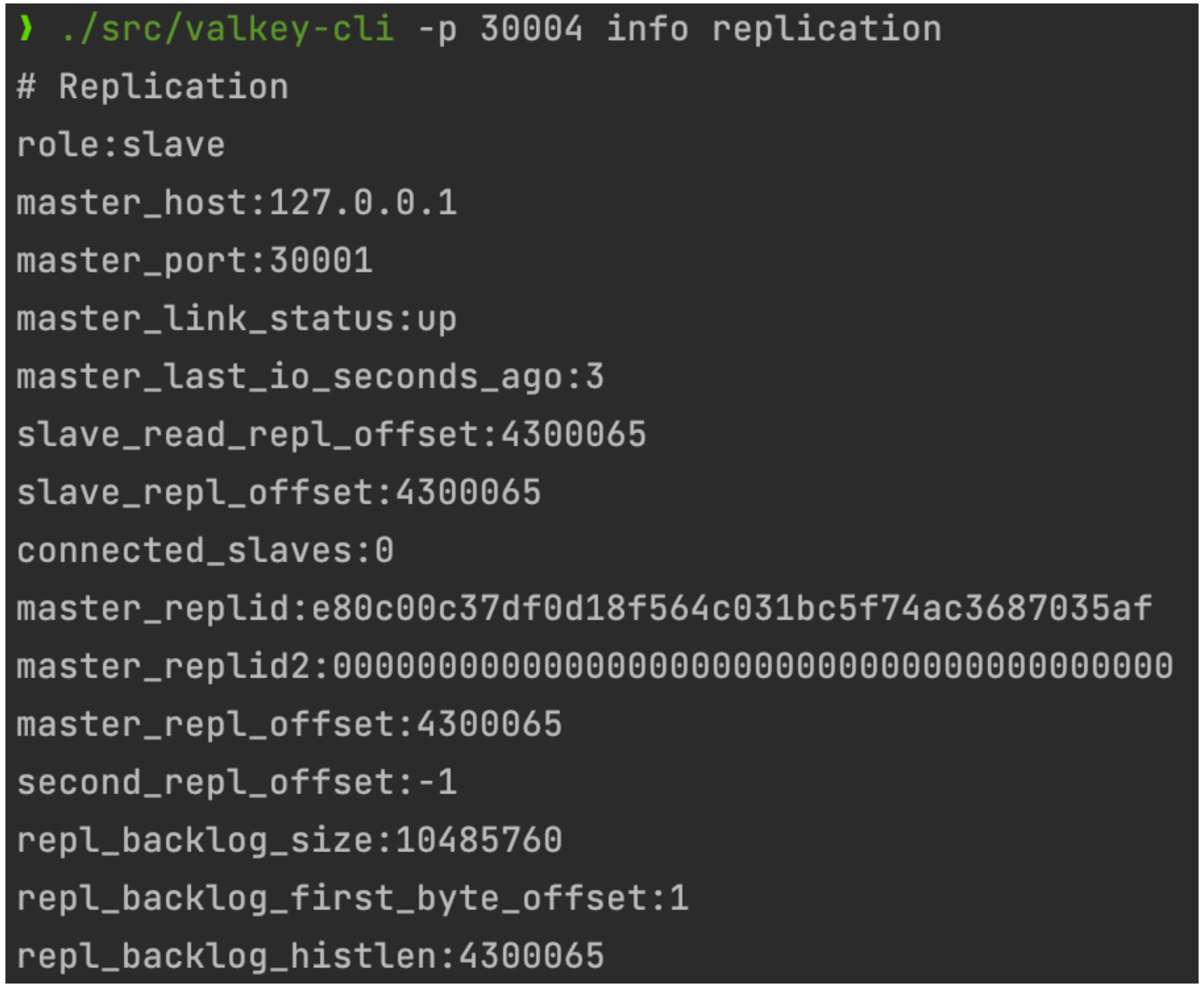
```
❱ ./src/valkey-cli -p 30007 role
1) "slave"
2) "127.0.0.1"
3) (integer) 30001
4) "connected"
5) (integer) 4300060
❱ ./src/valkey-cli -p 30008 role
1) "slave"
2) "127.0.0.1"
3) (integer) 30001
4) "connected"
5) (integer) 4300074
```

info command - primary side

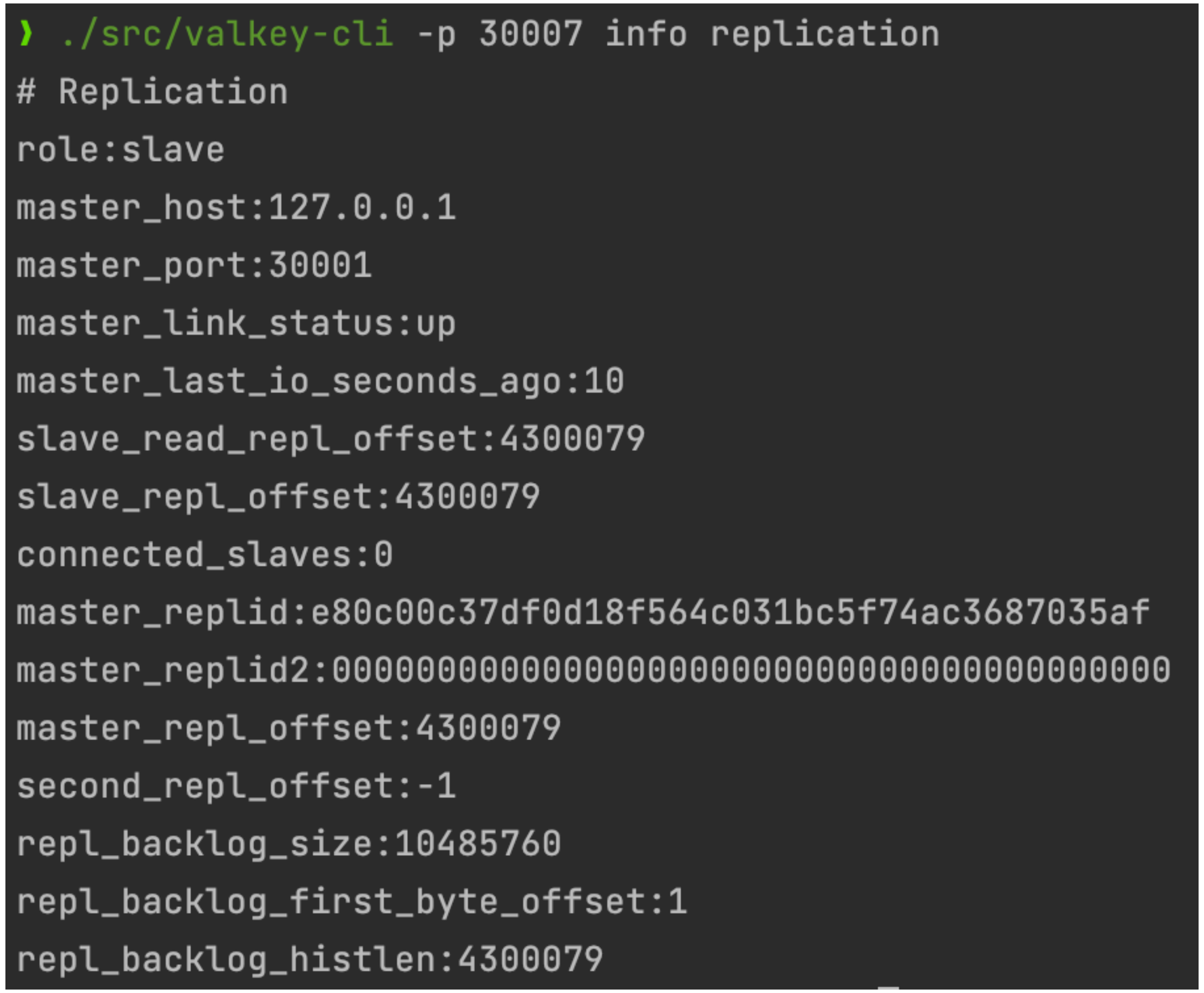


```
❯ ./src/valkey-cli -p 30001 info replication
# Replication
role:master
connected_slaves:2
slave0:ip=127.0.0.1,port=30004,state=online,offset=4300037,lag=1,type=replica
slave1:ip=127.0.0.1,port=30007,state=online,offset=4300037,lag=1,type=replica
master_replid:e80c00c37df0d18f564c031bc5f74ac3687035af
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:4300037
second_repl_offset:-1
repl_backlog_size:10485760
repl_backlog_first_byte_offset:1
repl_backlog_histlen:4300037
```

info command - replica side



```
❯ ./src/valkey-cli -p 30004 info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:30001
master_link_status:up
master_last_io_seconds_ago:3
slave_read_repl_offset:4300065
slave_repl_offset:4300065
connected_slaves:0
master_replid:e80c00c37df0d18f564c031bc5f74ac3687035af
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:4300065
second_repl_offset:-1
repl_backlog_size:10485760
repl_backlog_first_byte_offset:1
repl_backlog_histlen:4300065
```

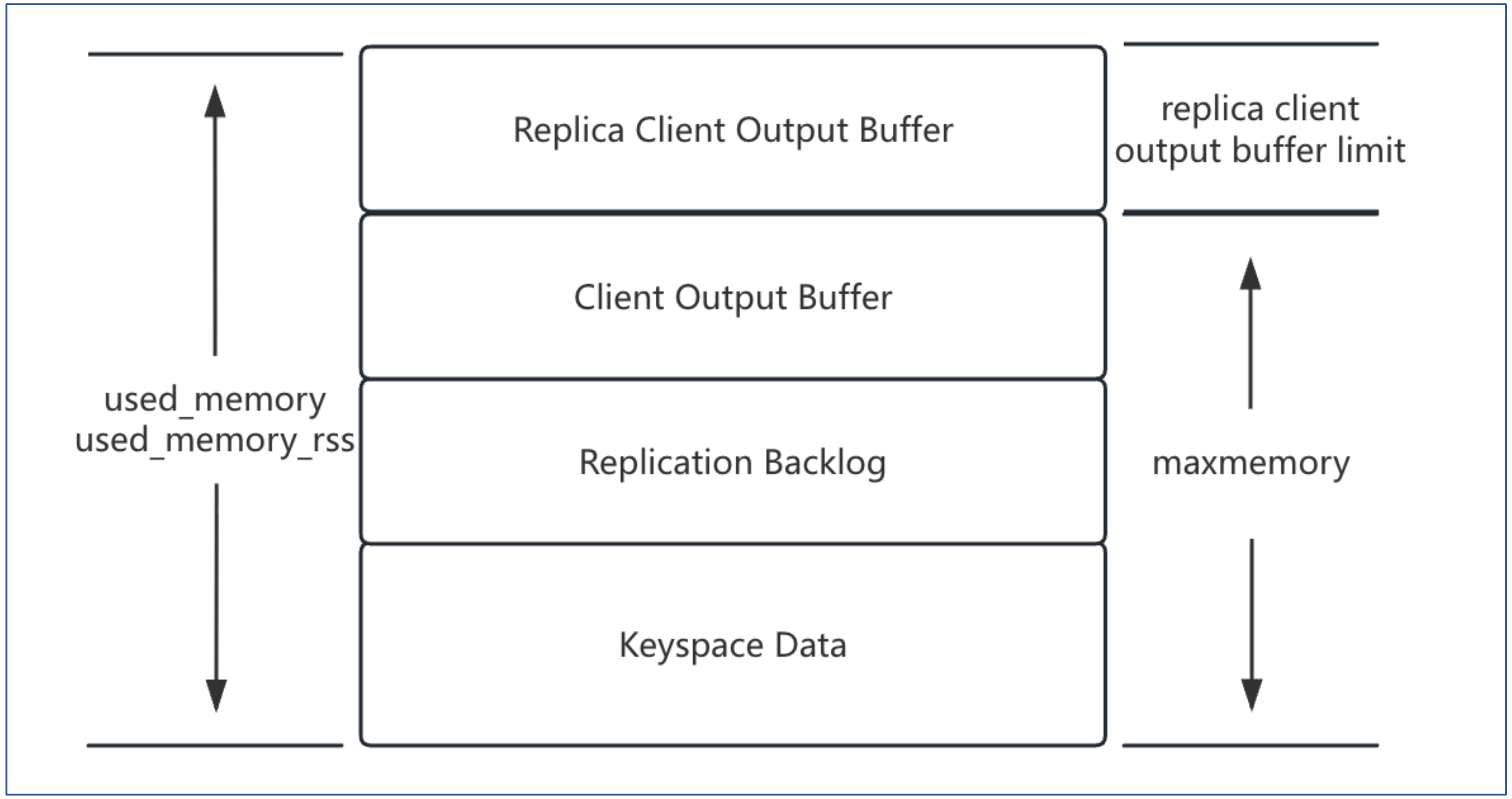


```
❯ ./src/valkey-cli -p 30007 info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:30001
master_link_status:up
master_last_io_seconds_ago:10
slave_read_repl_offset:4300079
slave_repl_offset:4300079
connected_slaves:0
master_replid:e80c00c37df0d18f564c031bc5f74ac3687035af
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:4300079
second_repl_offset:-1
repl_backlog_size:10485760
repl_backlog_first_byte_offset:1
repl_backlog_histlen:4300079
```

Replication memory usage - 1

内存是否统计进 used_memory ?

在统计进 used_memory 的情况下，
是否会参与 maxmemory 计算 ?



Replication memory usage - 2

- Replication Backlog (< 7.0)
  - 是一个提前分配的环形缓冲区
  - 被所有副本共享
  - 会统计进 used_memory，会参与 maxmemory 计算
  - info field: mem_replication_backlog

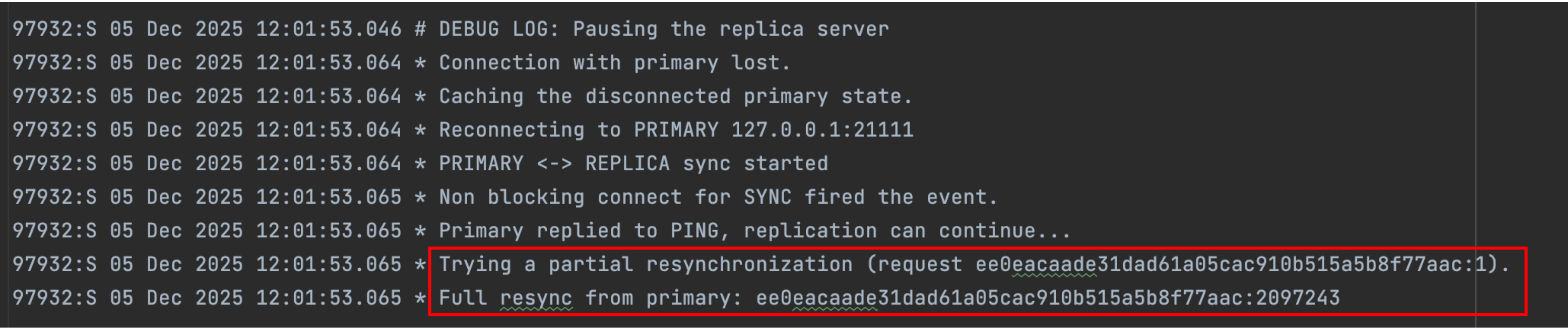
- Replica Client Output Buffer (< 7.0)
  - 每个副本都会有自己的客户端输出缓冲区
  - 有 hard limit 和 soft limit 限制慢副本连接输出缓冲区的内存占用
  - 会统计进 used_memory，但不会参与 maxmemory 计算
  - info field: mem_clients_slaves
  - info field: mem_not_counted_for_evict

- Global Replication Buffer (>= 7.0)
  - 复制积压缓冲区和所有副本客户端输出缓冲区，共享同一个全局复制缓冲区
  - info field: mem_total_replication_buffers

Replication issues

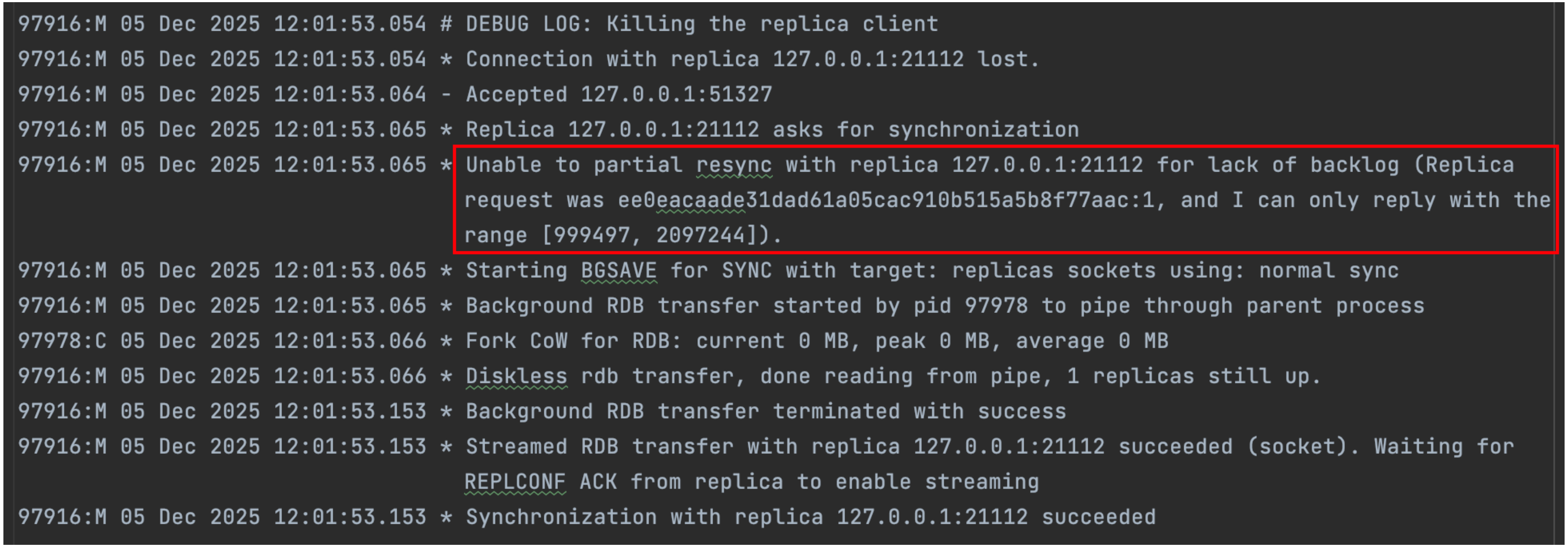
- 副本落后太多无法 psync
  - 副本节点落后太久，由于缺少复制积压数据，无法进行 psync 只能 full sync
- 慢副本因 COB 限制被断连
  - 慢副本在线期间消费太慢，因为 client output buffer limit 限制被主节点断连
- 副本主从复制反复 full sync
  - 副本 full sync 期间因为 client output buffer limit 限制失败，反复做 full sync
- 不限制 replica COB
  - 不限制 replica client output buffer limit 对主节点内存的影响

副本落后太多无法 psync - 副本节点日志



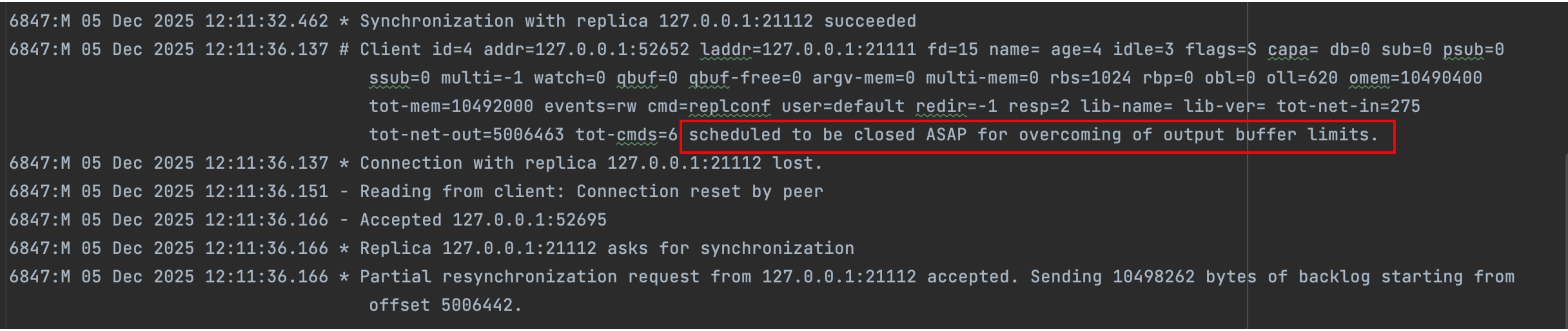
```
97932:S 05 Dec 2025 12:01:53.046 # DEBUG LOG: Pausing the replica server
97932:S 05 Dec 2025 12:01:53.064 * Connection with primary lost.
97932:S 05 Dec 2025 12:01:53.064 * Caching the disconnected primary state.
97932:S 05 Dec 2025 12:01:53.064 * Reconnecting to PRIMARY 127.0.0.1:21111
97932:S 05 Dec 2025 12:01:53.064 * PRIMARY <-> REPLICA sync started
97932:S 05 Dec 2025 12:01:53.065 * Non blocking connect for SYNC fired the event.
97932:S 05 Dec 2025 12:01:53.065 * Primary replied to PING, replication can continue...
97932:S 05 Dec 2025 12:01:53.065 * Trying a partial resynchronization (request ee0eacaade31dad61a05cac910b515a5b8f77aac:1).
97932:S 05 Dec 2025 12:01:53.065 * Full resync from primary: ee0eacaade31dad61a05cac910b515a5b8f77aac:2097243
```

副本落后太多无法 psync - 主节点日志



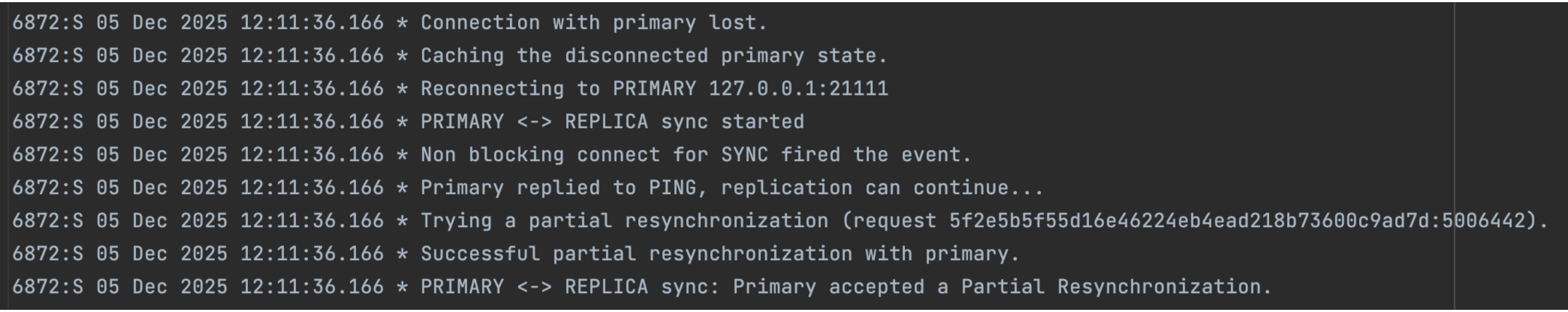
```
97916:M 05 Dec 2025 12:01:53.054 # DEBUG LOG: Killing the replica client
97916:M 05 Dec 2025 12:01:53.054 * Connection with replica 127.0.0.1:21112 lost.
97916:M 05 Dec 2025 12:01:53.064 - Accepted 127.0.0.1:51327
97916:M 05 Dec 2025 12:01:53.065 * Replica 127.0.0.1:21112 asks for synchronization
97916:M 05 Dec 2025 12:01:53.065 * Unable to partial resync with replica 127.0.0.1:21112 for lack of backlog (Replica
                                    request was ee0eacaade31dad61a05cac910b515a5b8f77aac:1, and I can only reply with the
                                    range [999497, 2097244]).
97916:M 05 Dec 2025 12:01:53.065 * Starting BGSAVE for SYNC with target: replicas sockets using: normal sync
97916:M 05 Dec 2025 12:01:53.065 * Background RDB transfer started by pid 97978 to pipe through parent process
97978:C 05 Dec 2025 12:01:53.066 * Fork CoW for RDB: current 0 MB, peak 0 MB, average 0 MB
97916:M 05 Dec 2025 12:01:53.066 * Diskless rdb transfer, done reading from pipe, 1 replicas still up.
97916:M 05 Dec 2025 12:01:53.153 * Background RDB transfer terminated with success
97916:M 05 Dec 2025 12:01:53.153 * Streamed RDB transfer with replica 127.0.0.1:21112 succeeded (socket). Waiting for
                                    REPLCONF ACK from replica to enable streaming
97916:M 05 Dec 2025 12:01:53.153 * Synchronization with replica 127.0.0.1:21112 succeeded
```

慢副本因 COB 限制被断连 - 主节点日志



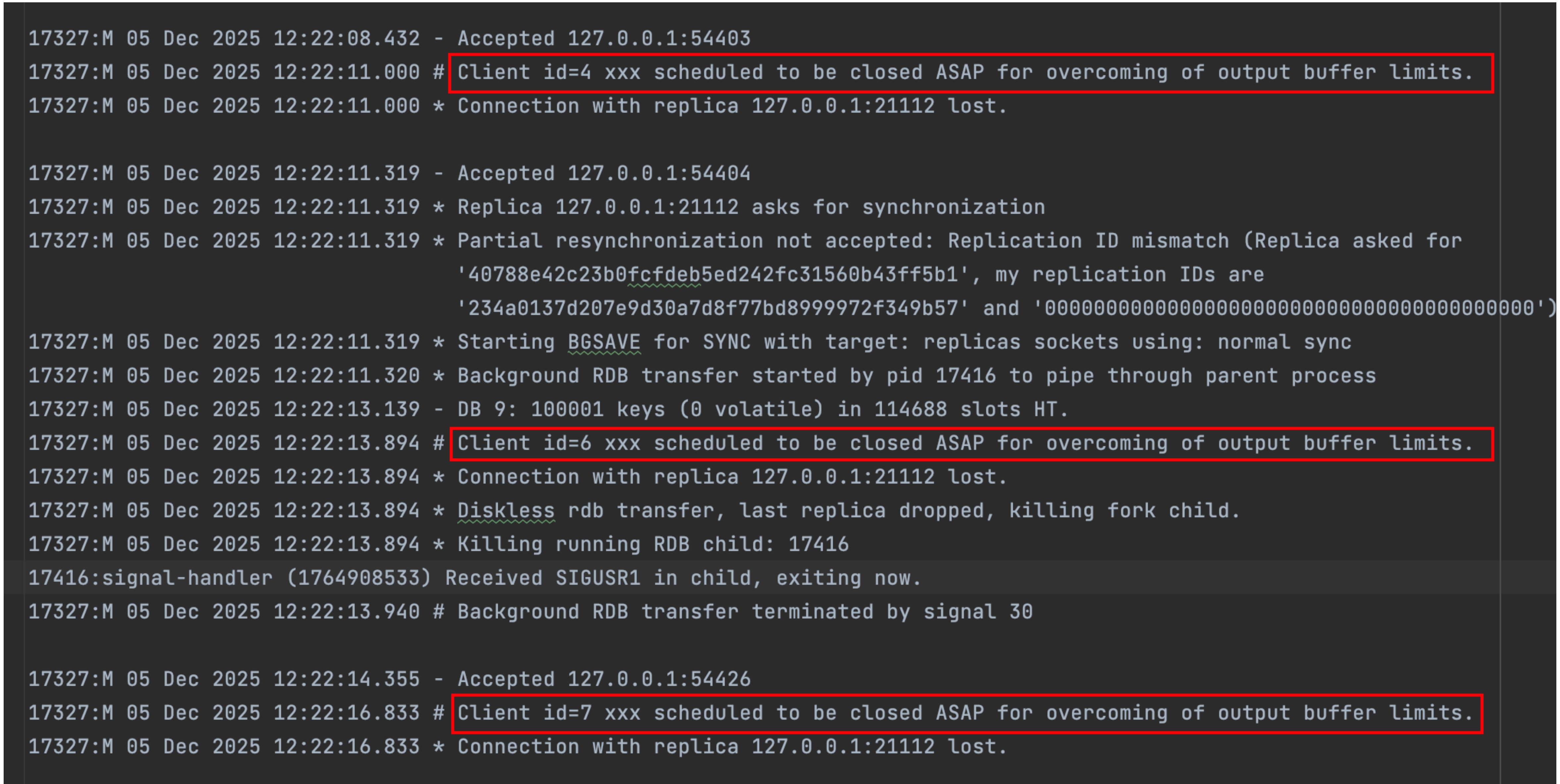
```
6847:M 05 Dec 2025 12:11:32.462 * Synchronization with replica 127.0.0.1:21112 succeeded
6847:M 05 Dec 2025 12:11:36.137 # Client id=4 addr=127.0.0.1:52652 laddr=127.0.0.1:21111 fd=15 name= age=4 idle=3 flags=S capa= db=0 sub=0 psub=0
                                   ssub=0 multi=-1 watch=0 qbuf=0 qbuf-free=0 argv-mem=0 multi-mem=0 rbs=1024 rbp=0 obl=0 oll=620 omem=10490400
                                   tot-mem=10492000 events=rw cmd=replconf user=default redir=-1 resp=2 lib-name= lib-ver= tot-net-in=275
                                   tot-net-out=5006463 tot-cmds=6 scheduled to be closed ASAP for overcoming of output buffer limits.
6847:M 05 Dec 2025 12:11:36.137 * Connection with replica 127.0.0.1:21112 lost.
6847:M 05 Dec 2025 12:11:36.151 - Reading from client: Connection reset by peer
6847:M 05 Dec 2025 12:11:36.166 - Accepted 127.0.0.1:52695
6847:M 05 Dec 2025 12:11:36.166 * Replica 127.0.0.1:21112 asks for synchronization
6847:M 05 Dec 2025 12:11:36.166 * Partial resynchronization request from 127.0.0.1:21112 accepted. Sending 10498262 bytes of backlog starting from
                                   offset 5006442.
```

慢副本因 COB 限制被断连 - 副本节点日志



```
6872:S 05 Dec 2025 12:11:36.166 * Connection with primary lost.
6872:S 05 Dec 2025 12:11:36.166 * Caching the disconnected primary state.
6872:S 05 Dec 2025 12:11:36.166 * Reconnecting to PRIMARY 127.0.0.1:21111
6872:S 05 Dec 2025 12:11:36.166 * PRIMARY <-> REPLICA sync started
6872:S 05 Dec 2025 12:11:36.166 * Non blocking connect for SYNC fired the event.
6872:S 05 Dec 2025 12:11:36.166 * Primary replied to PING, replication can continue...
6872:S 05 Dec 2025 12:11:36.166 * Trying a partial resynchronization (request 5f2e5b5f55d16e46224eb4ead218b73600c9ad7d:5006442).
6872:S 05 Dec 2025 12:11:36.166 * Successful partial resynchronization with primary.
6872:S 05 Dec 2025 12:11:36.166 * PRIMARY <-> REPLICA sync: Primary accepted a Partial Resynchronization.
```

副本主从复制反复 full sync - 主节点视角

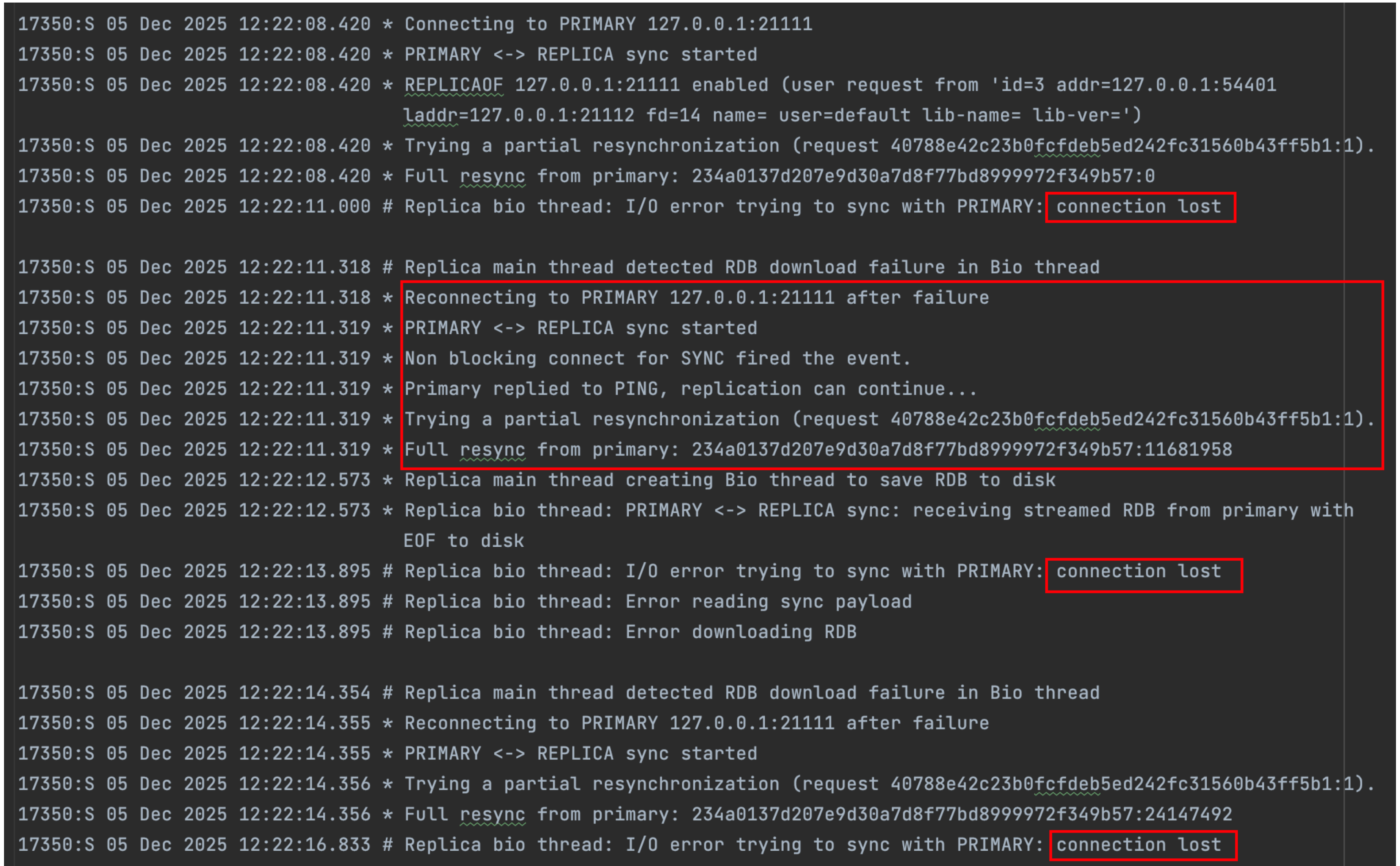


```
17327:M 05 Dec 2025 12:22:08.432 - Accepted 127.0.0.1:54403
17327:M 05 Dec 2025 12:22:11.000 # Client id=4 xxx scheduled to be closed ASAP for overcoming of output buffer limits.
17327:M 05 Dec 2025 12:22:11.000 * Connection with replica 127.0.0.1:21112 lost.

17327:M 05 Dec 2025 12:22:11.319 - Accepted 127.0.0.1:54404
17327:M 05 Dec 2025 12:22:11.319 * Replica 127.0.0.1:21112 asks for synchronization
17327:M 05 Dec 2025 12:22:11.319 * Partial resynchronization not accepted: Replication ID mismatch (Replica asked for
                                    '40788e42c23b0fcfdeb5ed242fc31560b43ff5b1', my replication IDs are
                                    '234a0137d207e9d30a7d8f77bd8999972f349b57' and '0000000000000000000000000000000000000000')
17327:M 05 Dec 2025 12:22:11.319 * Starting BGSAVE for SYNC with target: replicas sockets using: normal sync
17327:M 05 Dec 2025 12:22:11.320 * Background RDB transfer started by pid 17416 to pipe through parent process
17327:M 05 Dec 2025 12:22:13.139 - DB 9: 100001 keys (0 volatile) in 114688 slots HT.
17327:M 05 Dec 2025 12:22:13.894 # Client id=6 xxx scheduled to be closed ASAP for overcoming of output buffer limits.
17327:M 05 Dec 2025 12:22:13.894 * Connection with replica 127.0.0.1:21112 lost.
17327:M 05 Dec 2025 12:22:13.894 * Diskless rdb transfer, last replica dropped, killing fork child.
17327:M 05 Dec 2025 12:22:13.894 * Killing running RDB child: 17416
17416:signal-handler (1764908533) Received SIGUSR1 in child, exiting now.
17327:M 05 Dec 2025 12:22:13.940 # Background RDB transfer terminated by signal 30

17327:M 05 Dec 2025 12:22:14.355 - Accepted 127.0.0.1:54426
17327:M 05 Dec 2025 12:22:16.833 # Client id=7 xxx scheduled to be closed ASAP for overcoming of output buffer limits.
17327:M 05 Dec 2025 12:22:16.833 * Connection with replica 127.0.0.1:21112 lost.
```

副本主从复制反复 full sync - 副本节点视角

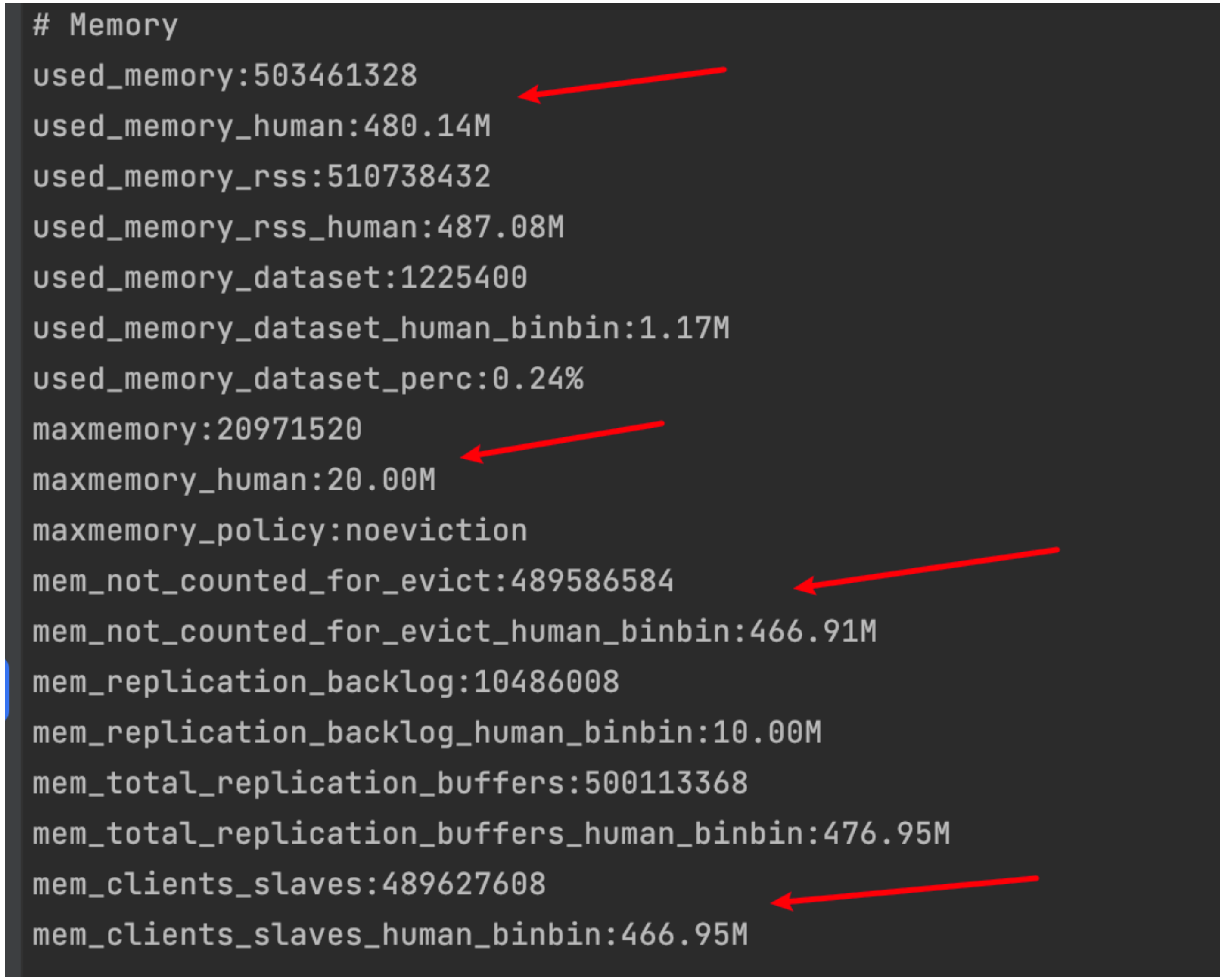


```
17350:S 05 Dec 2025 12:22:08.420 * Connecting to PRIMARY 127.0.0.1:21111
17350:S 05 Dec 2025 12:22:08.420 * PRIMARY <-> REPLICA sync started
17350:S 05 Dec 2025 12:22:08.420 * REPLICAOF 127.0.0.1:21111 enabled (user request from 'id=3 addr=127.0.0.1:54401
                                   laddr=127.0.0.1:21112 fd=14 name= user=default lib-name= lib-ver=')
17350:S 05 Dec 2025 12:22:08.420 * Trying a partial resynchronization (request 40788e42c23b0fcfdeb5ed242fc31560b43ff5b1:1).
17350:S 05 Dec 2025 12:22:08.420 * Full resync from primary: 234a0137d207e9d30a7d8f77bd8999972f349b57:0
17350:S 05 Dec 2025 12:22:11.000 # Replica bio thread: I/O error trying to sync with PRIMARY: connection lost

17350:S 05 Dec 2025 12:22:11.318 # Replica main thread detected RDB download failure in Bio thread
17350:S 05 Dec 2025 12:22:11.318 * Reconnecting to PRIMARY 127.0.0.1:21111 after failure
17350:S 05 Dec 2025 12:22:11.319 * PRIMARY <-> REPLICA sync started
17350:S 05 Dec 2025 12:22:11.319 * Non blocking connect for SYNC fired the event.
17350:S 05 Dec 2025 12:22:11.319 * Primary replied to PING, replication can continue...
17350:S 05 Dec 2025 12:22:11.319 * Trying a partial resynchronization (request 40788e42c23b0fcfdeb5ed242fc31560b43ff5b1:1).
17350:S 05 Dec 2025 12:22:11.319 * Full resync from primary: 234a0137d207e9d30a7d8f77bd8999972f349b57:11681958
17350:S 05 Dec 2025 12:22:12.573 * Replica main thread creating Bio thread to save RDB to disk
17350:S 05 Dec 2025 12:22:12.573 * Replica bio thread: PRIMARY <-> REPLICA sync: receiving streamed RDB from primary with
                                   EOF to disk
17350:S 05 Dec 2025 12:22:13.895 # Replica bio thread: I/O error trying to sync with PRIMARY: connection lost
17350:S 05 Dec 2025 12:22:13.895 # Replica bio thread: Error reading sync payload
17350:S 05 Dec 2025 12:22:13.895 # Replica bio thread: Error downloading RDB

17350:S 05 Dec 2025 12:22:14.354 # Replica main thread detected RDB download failure in Bio thread
17350:S 05 Dec 2025 12:22:14.355 * Reconnecting to PRIMARY 127.0.0.1:21111 after failure
17350:S 05 Dec 2025 12:22:14.355 * PRIMARY <-> REPLICA sync started
17350:S 05 Dec 2025 12:22:14.356 * Trying a partial resynchronization (request 40788e42c23b0fcfdeb5ed242fc31560b43ff5b1:1).
17350:S 05 Dec 2025 12:22:14.356 * Full resync from primary: 234a0137d207e9d30a7d8f77bd8999972f349b57:24147492
17350:S 05 Dec 2025 12:22:16.833 # Replica bio thread: I/O error trying to sync with PRIMARY: connection lost
```

不限制 replica COB - 主节点内存占用



```
# Memory
used_memory:503461328
used_memory_human:480.14M
used_memory_rss:510738432
used_memory_rss_human:487.08M
used_memory_dataset:1225400
used_memory_dataset_human_binbin:1.17M
used_memory_dataset_perc:0.24%
maxmemory:20971520
maxmemory_human:20.00M
maxmemory_policy:noeviction
mem_not_counted_for_evict:489586584
mem_not_counted_for_evict_human_binbin:466.91M
mem_replication_backlog:10486008
mem_replication_backlog_human_binbin:10.00M
mem_total_replication_buffers:500113368
mem_total_replication_buffers_human_binbin:476.95M
mem_clients_slaves:489627608
mem_clients_slaves_human_binbin:466.95M
```

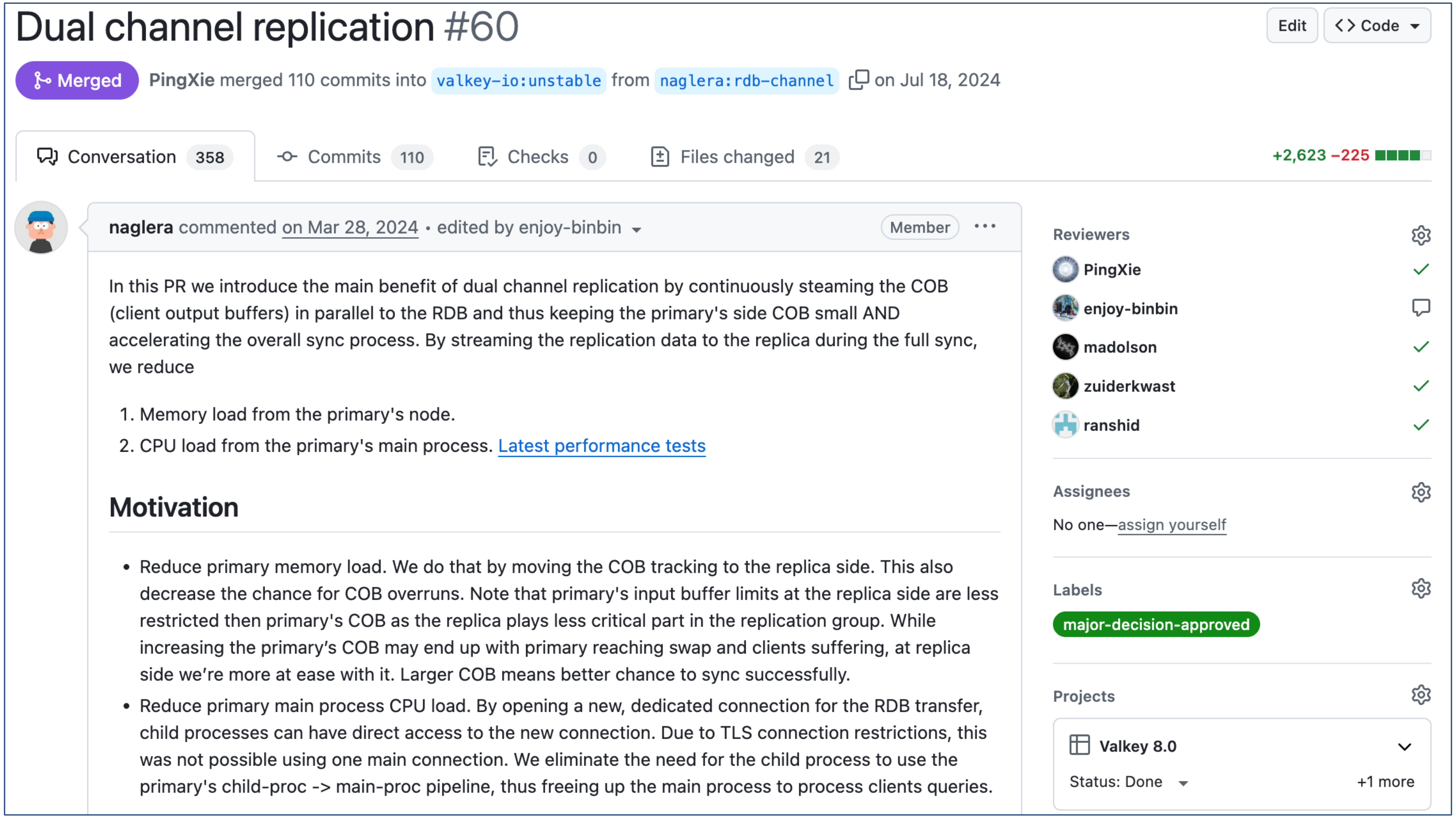
Dual channel replication - 1

- 问题：
  - 主节点全量同步期间副本客户端输出缓冲区内存占用多
  - 副本客户端输出缓冲区不容易计算出准确的限制值
  - 如果一直调大限制，则节点机器内存存在 OOM 风险
  - 如果一直不调整，则节点主从同步全量同步将一直无法完成，也可能存在单主风险
  - 反复的全量同步对主节点影响很大，fork / copy on write / 占用主线程 CPU 等

- 需要提高全量同步的成功率 / 可靠性

Dual channel replication #60 - https://github.com/valkey-io/valkey/pull/60

Dual channel replication - 2



Dual channel replication #60

Merged PingXie merged 110 commits into `valkey-io:unstable` from `naglera:rdb-channel` on Jul 18, 2024

naglera commented on Mar 28, 2024 • edited by enjoy-binbin Member

In this PR we introduce the main benefit of dual channel replication by continuously steaming the COB (client output buffers) in parallel to the RDB and thus keeping the primary's side COB small AND accelerating the overall sync process. By streaming the replication data to the replica during the full sync, we reduce

1. Memory load from the primary's node.
2. CPU load from the primary's main process. Latest performance tests

Motivation

- Reduce primary memory load. We do that by moving the COB tracking to the replica side. This also decrease the chance for COB overruns. Note that primary's input buffer limits at the replica side are less restricted then primary's COB as the replica plays less critical part in the replication group. While increasing the primary's COB may end up with primary reaching swap and clients suffering, at replica side we're more at ease with it. Larger COB means better chance to sync successfully.
- Reduce primary main process CPU load. By opening a new, dedicated connection for the RDB transfer, child processes can have direct access to the new connection. Due to TLS connection restrictions, this was not possible using one main connection. We eliminate the need for the child process to use the primary's child-proc -> main-proc pipeline, thus freeing up the main process to process clients queries.

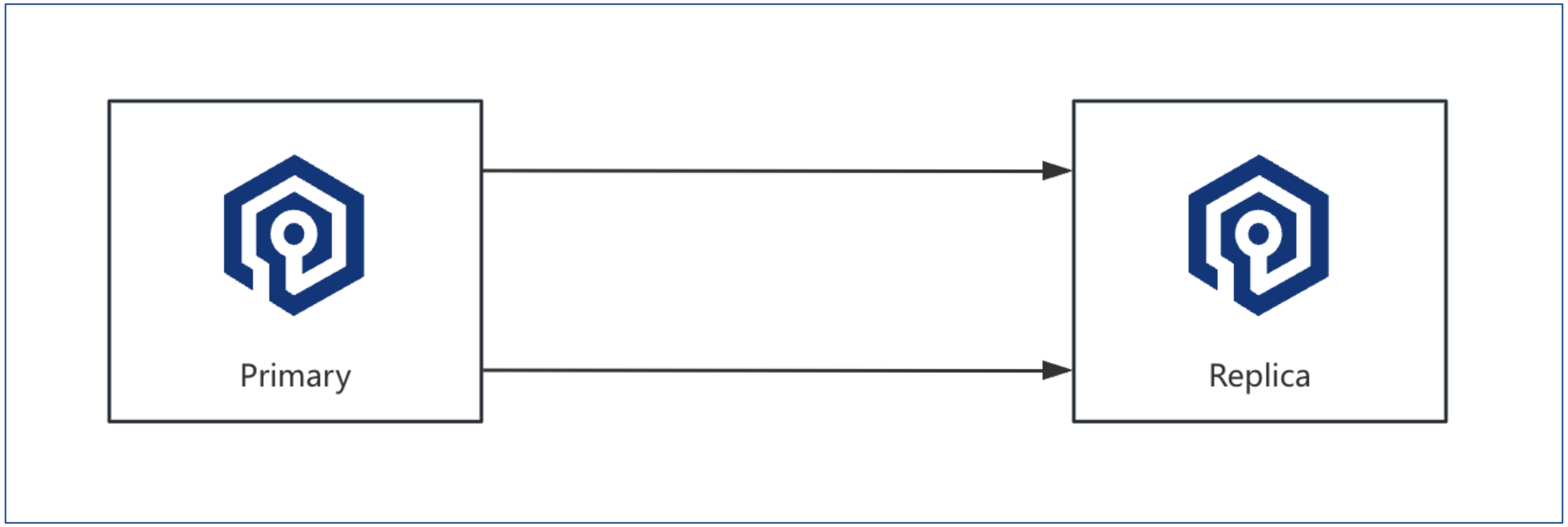
Reviewers
PingXie ✓
enjoy-binbin
madolson ✓
zuiderkwast ✓
ranshid ✓

Assignees
No one—assign yourself

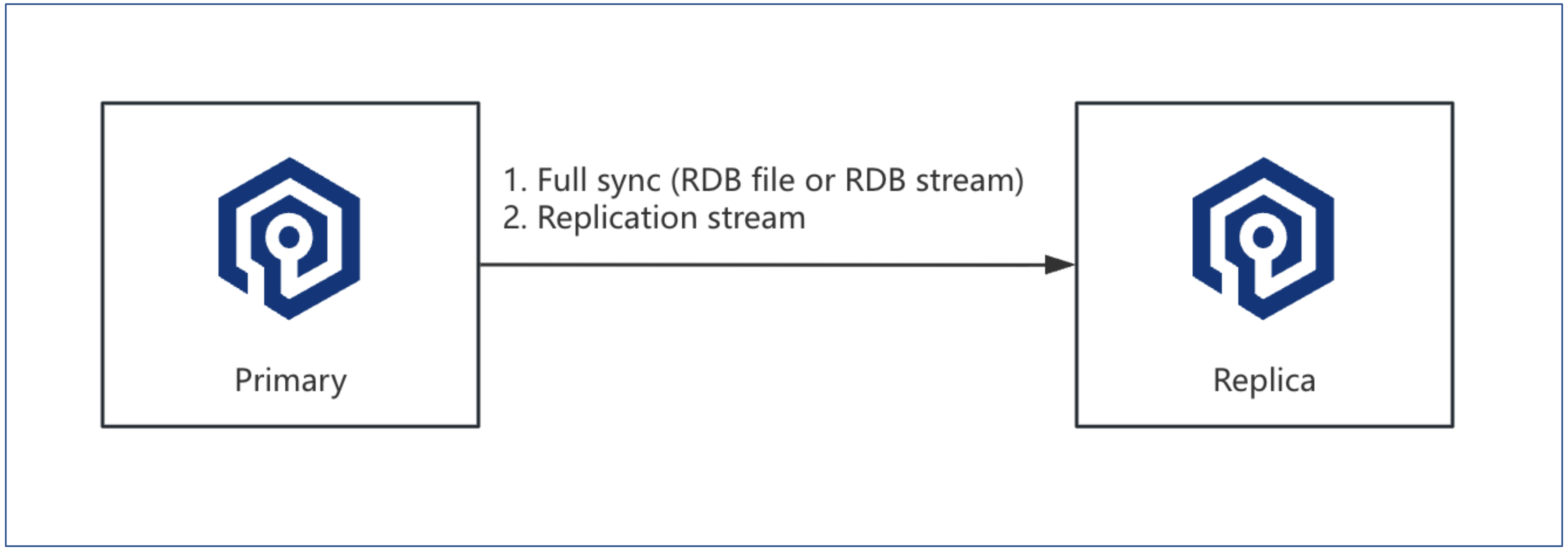
Labels
major-decision-approved

Projects
Valkey 8.0
Status: Done +1 more

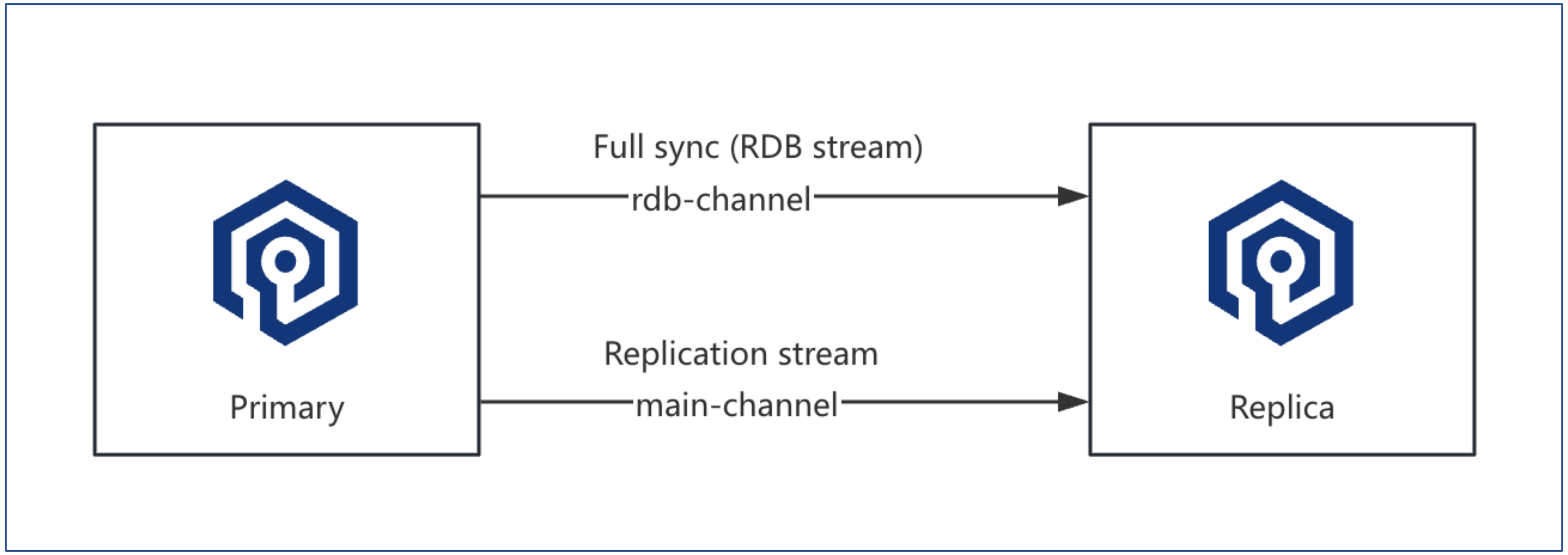
Dual channel replication - 3



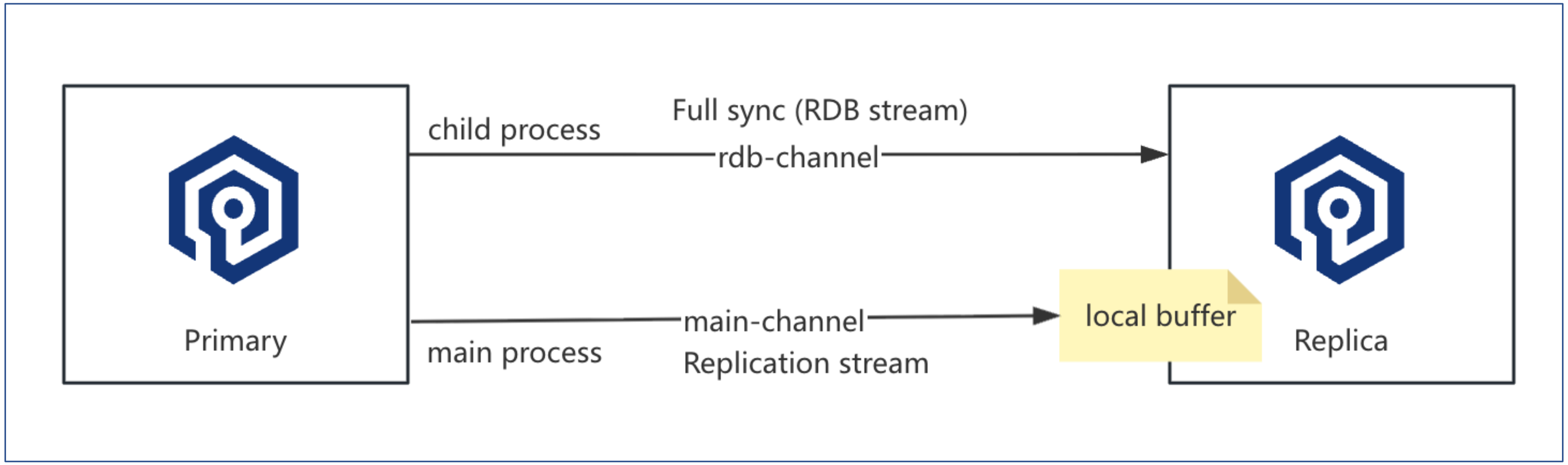
Dual channel replication - 4



Dual channel replication - 5



Dual channel replication - 6



Dual channel replication - 7

- 无盘主从复制中针对全量同步的优化

- 核心思路：
  - 在全量同步期间，副本节点会和主节点再建立一条 RDB 连接，主节点会在两条连接上并行发送 RDB 快照和副本客户端输出缓冲区里的命令流数据，副本节点会将命令流数据缓存在本地内存，等 RDB 快照加载完后回放命令流。

Dual channel replication - full sync + psync

- 全量同步 full sync for rdb channel：
    - 全量也就是 RDB 快照部分，当副本节点主从握手完收到主节点 +DUALCHANNELSYNC 响应后，会创建一条新的连接，叫做 rdb channel。
    - 副本节点会利用 rdb channel 向主节点发送 sync 命令请求全量同步。
    - 主节点在收到 rdb channel 的 sync 后，在 fork 之前会将当前的复制偏移量通过 rdb channel 发送给副本节点，即会将 snapshot end-offset 发送给副本节点。

- 增量同步 psync for main channel：
    - 副本节点收到 snapshot end-offset 后，在 main channel 里利用 end-offset 发送 psync 命令请求增量同步，请求对应 offset 之后的增量数据。
    - 副本节点会将 psync 后的增量命令流数据缓存到本地内存中。
    - 当副本节点 rdb channel 完成 RDB 快照的加载后，就会将本地内存中的增量命令加载到 DB 中。

Dual channel replication - benefits - 1

- 减少主节点内存压力：
  - 通过将 COB 从主节点转移到副本节点，这会减轻主节点的内存压力。
  - 在 full sync 期间，因为 fork 的调用，主节点同时也在承受 COW 带来的内存上涨压力，所以如果能够减少主节点内存使用，对主节点也会有好处，不然类似原本主节点需要同时承受 COW 和 COB 的开销，两个加一起内存占用非常不好限制和运维，现在主节点只需要承受 COW 的开销，而尽量少的承受 COB 的开销。
  - 副本节点本身在 full sync 期间不怎么会被 COW 影响，所以对比主节点能有较多的内存资源用于承受 COB。

- 减少主节点主线程的 CPU 负载：
  - 通过为 RDB 快照传输建立专有连接，子进程可以直接通过 RDB 连接进行数据传输，从而省去了主进程中转数据的过程，让主线程可以专注处理客户端请求。
  - 在 6.0 版本中为了支持 TLS，在无盘传输中，子进程实际上会先将 RDB 内容通过管道写给父进程，父进程读出来后发送给副本节点。父子进程需要频繁通过管道进行通信。

Dual channel replication - benefits - 2

- 提高 full sync 成功率：
  - 原本为了避免 replica COB 限制导致 full sync 失败，我们通常需要调整主节点的 COB 限制，但是这个值没有很好的计算方式去设置，并且主节点侧提高 COB 限制可能会耗尽主节点侧内存，通过 dual channel，实际上可以理解为是将 COB 限制调大了，管控会更加灵活，相当于副本节点先承担一部分 COB，然后主节点再承担后面部分的 COB，更大的 COB 意味着更高的全量同步成功率。

- 提高 full sync 的效率：
  - 在之前需要等副本节点加载完 RDB 快照后才能发送 replication buffer，是串行的。积累的 replication buffer 可能会需要一段时间才能传输完成。
  - 现在是并行发送，副本节点加载完 RDB 快照后可直接重放本地内存，全量同步期间的增量命令传输时间将会缩短，进而缩短整体同步时间。

Dual channel replication - benefits - 3

- 增强副本节点数据一致性
  - 因为有提升 full sync 的效率，某种程度上能减少副本节点提供陈旧数据的窗口时间。副本在上线后，从开始消费 replication buffer 到真正消费完之前，这中间会有一段时间窗口，主节点侧可能积累了很多数据，这些数据变更需要经过一定时间才能真正传播到副本节点。

- 快照传输和主线程流量负载不关联
  - 当有了 rdb channel 后，子进程可以直接发数据，即使父进程主线程遇到一些慢查询命令，或者命令复杂压力大，RDB 的流式传输也不会受到影响。子进程无需和负载过重的主进程共享 CPU 资源即可往副本传输数据，不仅可以提升客户端响应速度，还可以缩短同步时间。

Dual channel replication - Configurations

- client-output-buffer-limit replica 256mb 64mb 60
- repl-diskless-sync yes
- dual-channel-replication-enabled no

Dual channel replication - More? - 1

- 副本节点将 local buffer 落盘
    - 我们可以看到，dual channel 只是将内存从主节点侧转移到了副本节点侧，实际上实例使用的总内存是没有变化的，并没有完全消除内存，只是将内存开销从重要的主节点，转移到了不那么重要的副本节点上。
    - 这种可以做的优化是类似在副本节点那边，将 replication stream 的内容给落盘，之后加载完 RDB 后再从硬盘加载 replication stream，从而可以继续扩大 COB。

- 多副本同时全量同步，总内存增加
    - 7.0 引入了全局复制共享缓冲区，无论有多少个副本，主节点只需要存储一份数据到共享缓冲区中（我们允许多个副本同时全量同步），如果使用 dual channel，这意味着共享缓冲区作用失效了部分，复制流数据在多个副本上都进行了缓存，实例总内存使用情况是在增加的。
    - 这种可以做的优化可能是类似，当主节点判断出有多个副本需要同时做全量同步的时候，主节点那边不走 dual channel 而是走回原本的 single channel replication。

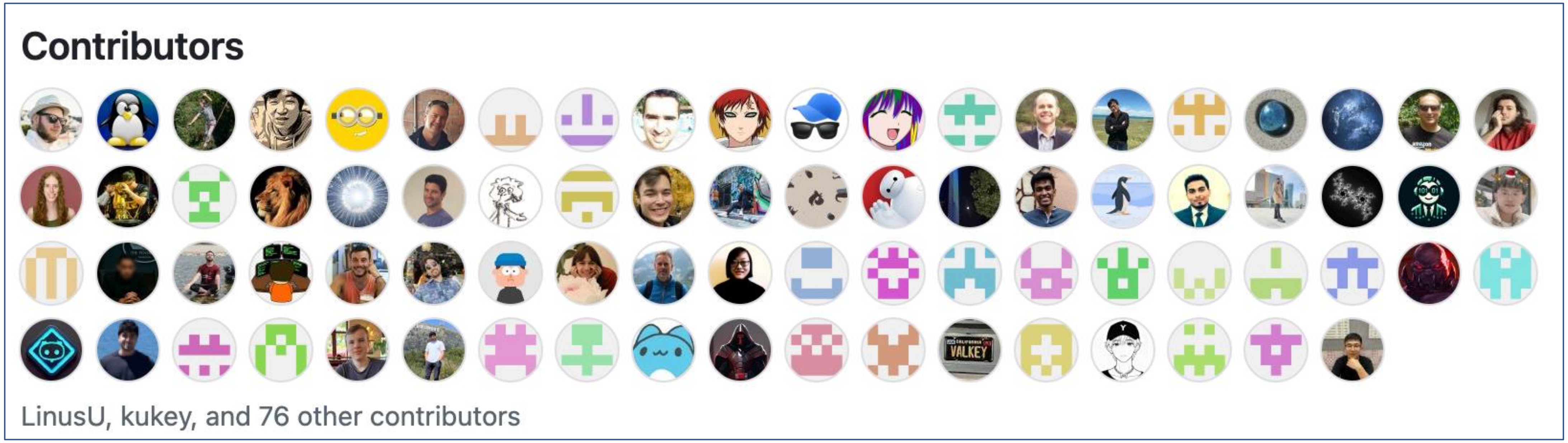
Dual channel replication - More? - 2

- 副本边加载 local buffer 边继续缓存
  - 目前 valkey 副本节点在消费 local buffer 的时候，它会将 read handler 给置空，停止继续从主节点那边读取数据，实际上我们其实可以继续读取，这样能更好的将主节点那边的内容转移过来。因为在大流量场景下，local buffer 可能会堆积的很大，要完全消费完可能需要一段时间，在这段时间主节点那边如果 COB 达到限制一样会断连，所以我们可以尽量继续转移，即边加载 local buffer 边继续缓存。

- 单独的副本 local buffer 配置项
  - 前面我们提到了副本 local buffer 的大小是继承自 replica client-output-buffer-limit 配置项的，在某些运维场景，为副本单独设置一个新的 local buffer 配置项可能会更灵活。因为副本节点可能会在任意时刻被提升为主节点，和 client-output-buffer-limit 共用一个配置项可能会没那么灵活。

Make Valkey Great Together

- No CLA, only have DCO
- https://valkey.io/
- https://github.com/valkey-io/valkey
- https://github.com/valkey-io/valkey-doc
- https://github.com/valkey-io/valkey/blob/unstable/CONTRIBUTING.md



Contributors

LinusU, kukey, and 76 other contributors

https://github.com/valkey-io/valkey/releases/tag/9.0.0-rc1

KEYSPACE
2025 / 北京

Thank you! Enjoy Valkey Keyspace!

binbinzhu(朱彬彬) | 2025/12/13

腾讯云研发，Valkey TSC Member