



HBase Practice At XiaoMi

tianjy1990@gmail.com

openinx@apache.org

Part-1 Problems In Practice



Problems in XiaoMi

- ❑ Problem 1. How to satisfy the regular demand of scanning table without affecting other requests?

Better support for data analysis

- ❑ Scan is expensive
- ❑ Data analysis need to scan a large number of data from hbase
- ❑ They are executed by mapreduce or spark, that put a heavy burden on HBase



Scan snapshot directly

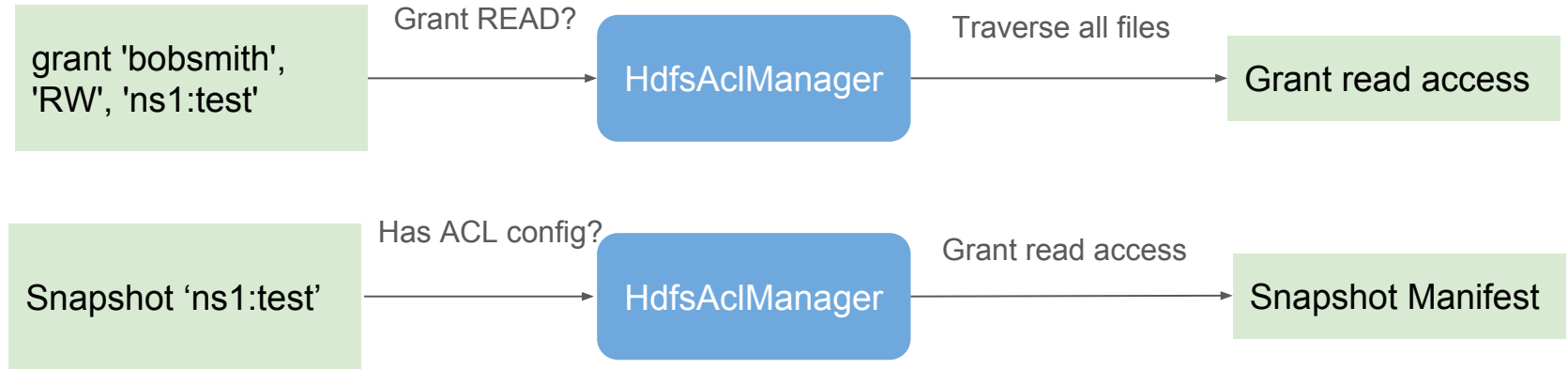
- ❑ HBase already provides this feature: **TableSnapshotInputFormat** (ClientSideRegionScanner)
 - ❑ Construct regions by snapshot files
 - ❑ Read data without any HBase RPC requests
 - ❑ **Required READ access to reference files and HFiles**

Snapshot ACL

- ❑ HDFS ACL could grant accesses to certain users besides owner and group
- ❑ Can support upto 16 users: 32 ACLs / 2 (include user, group, mask, other)

```
[tianjingyun@c4-hadoop-build01 bin]$ ./hdfs dfs -getfacl /hbase/c3prc-xiaomi98
18/05/29 19:37:55 INFO security.UserGroupInformation: Can't login from keytab, try to login from ticket cache
# file: /hbase/c3prc-xiaomi98
# owner: hbase_prc
# group: supergroup
user::rwx
user:hbase_admin:rwx    #effective:r-x
group:---
mask::r-x
other::r-x
default:user::rwx
default:user:hbase_admin:rwx
default:group:---
default:mask::rwx
default:other:---
```

HDFSACLManager



Problems in XiaoMi

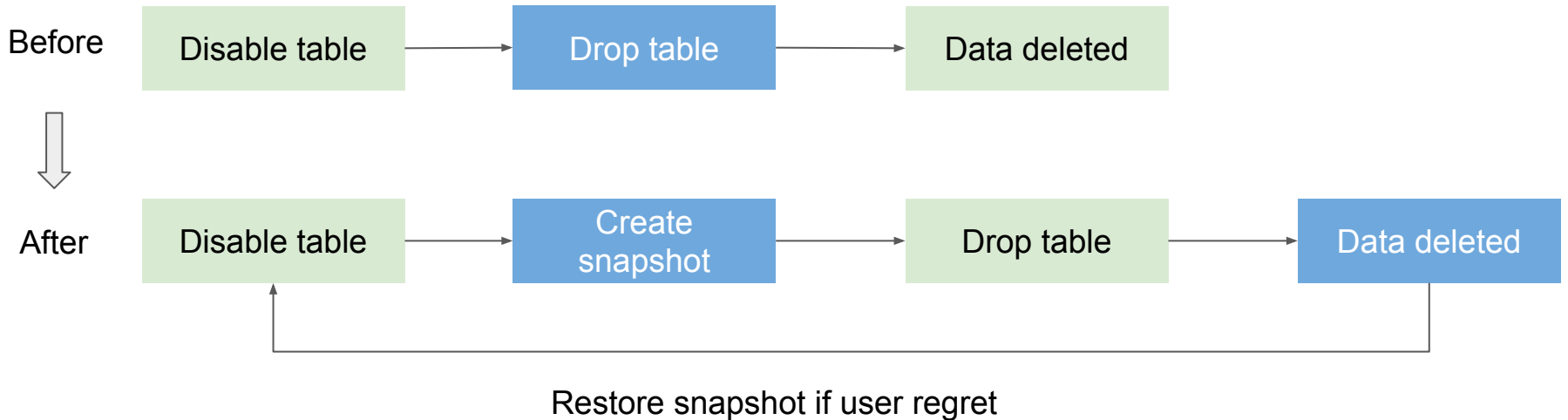
- ❑ Problem 2. How to prevent data from being contaminated or lost when user operate incorrectly?

Users may make mistakes

- ❑ User regret dropping table
- ❑ User delete data by mistake or write in wrong data

Soft deletion

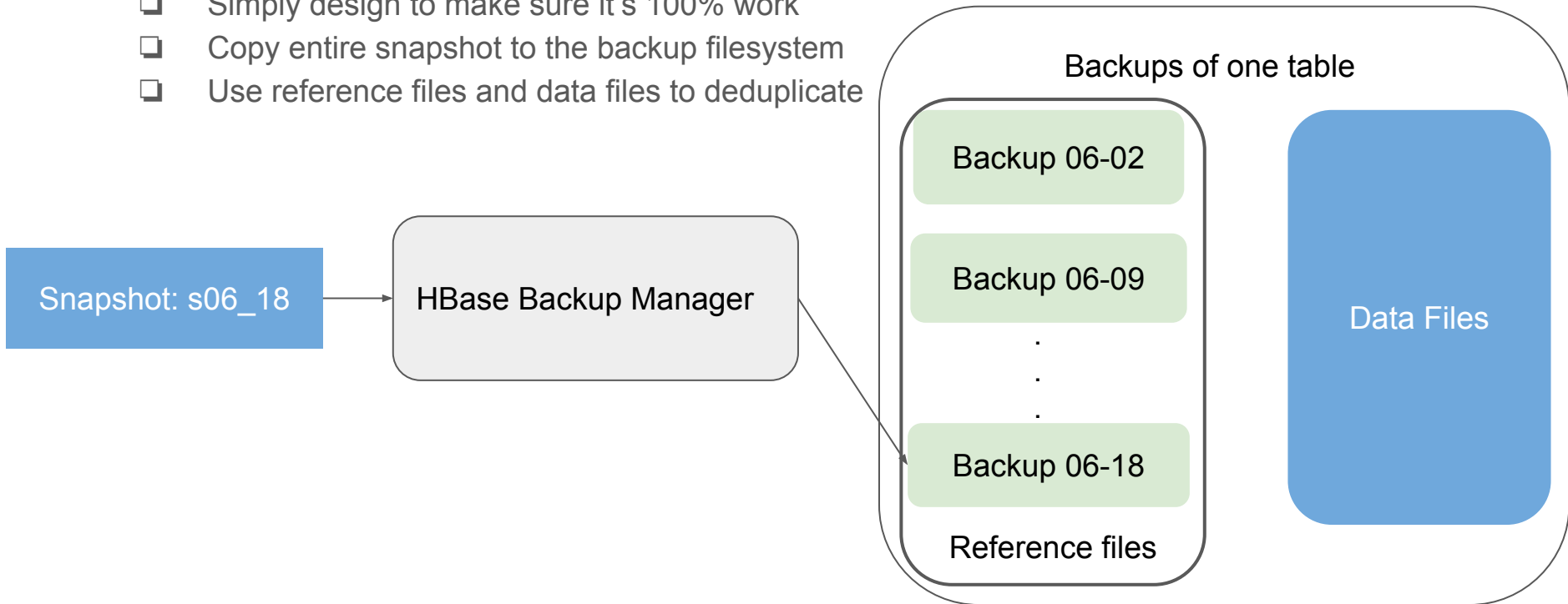
- ❑ Take a snapshot before do real table deletion
- ❑ Clean these snapshots after a certain period (e.g. one week)





Backup snapshot to heterogeneous FileSystem

- ❑ Design of backup logic
 - ❑ Simply design to make sure it's 100% work
 - ❑ Copy entire snapshot to the backup filesystem
 - ❑ Use reference files and data files to deduplicate





HBase Backup Manager

- ❑ Delete snapshot regularly
- ❑ Verify the backups on the heterogeneous fileSystem



Problems in XiaoMi

- ❑ Problem 3. Restart of a small cluster is stuck at log splitting

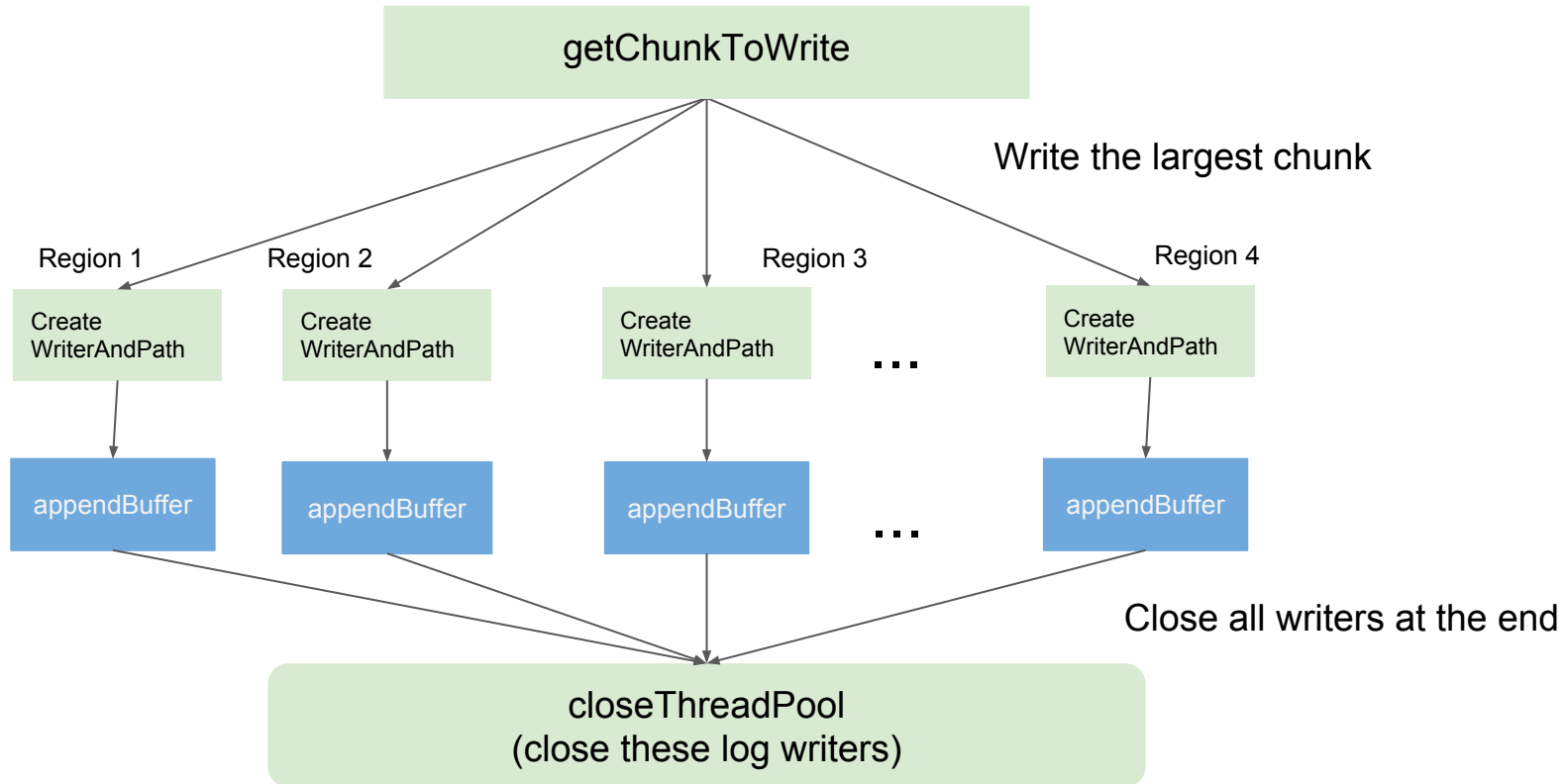


Why stuck at splitting

- ❑ HBASE-19358
- ❑ Problem: Log splitting has a bad performance or even not working when the number of region is very large, e.g 1000+ per regionserver



Old design of log splitting



Log splitting problem - 1

- ❑ Too much space need to be reserved at the same time. HDFS may not able to assign enough space to create a new block.

*Reserved space = number of Log * **number of region** * configured size of HLog block * number of replica*

- ❑ Especially for a small hdfs cluster, whose remaining space is small.

Log splitting problem - 2

- ❑ Too many HDFS streams created at the same time. Then it is prone to failure since each datanode need to handle too many streams.

*Number of streams = $\frac{\text{number of Log} * \text{number of region} * \text{number of replica} * \text{number of regionserver}}{\text{number of datanodes}}$*

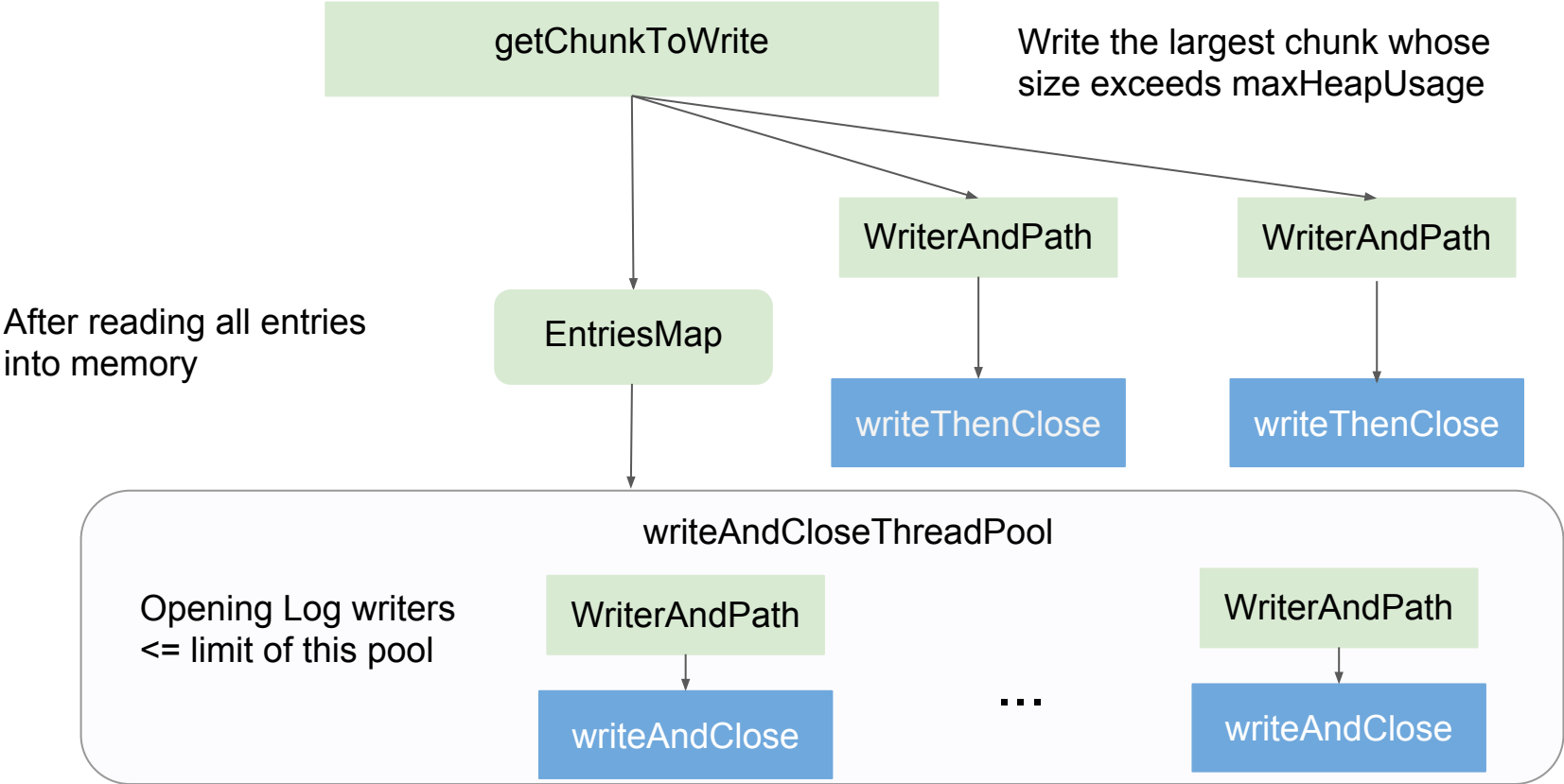
Streams each datanode need to handle = Number of streams / number of datanodes

Number of regionserver usually equals to number of datanodes.

- ❑ Clusters all facing this problem no matter they are large or small.

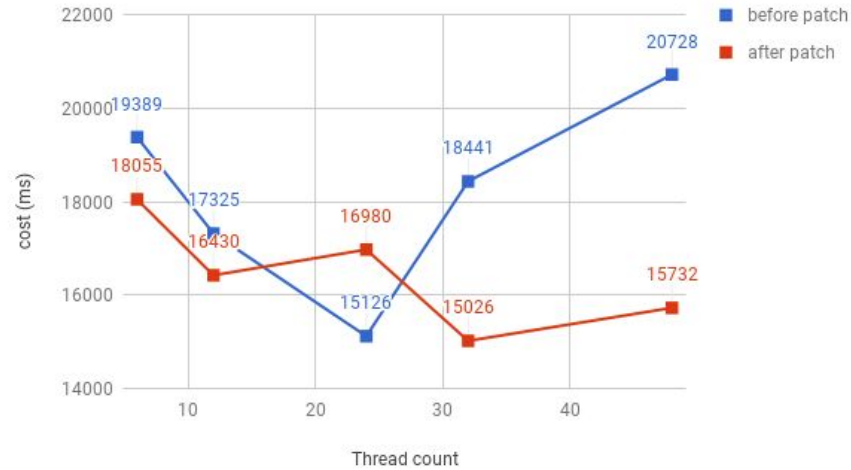


New design of log splitting



Performance

Time cost to split one 512MB HLog



Time cost to split one 512MB HLog

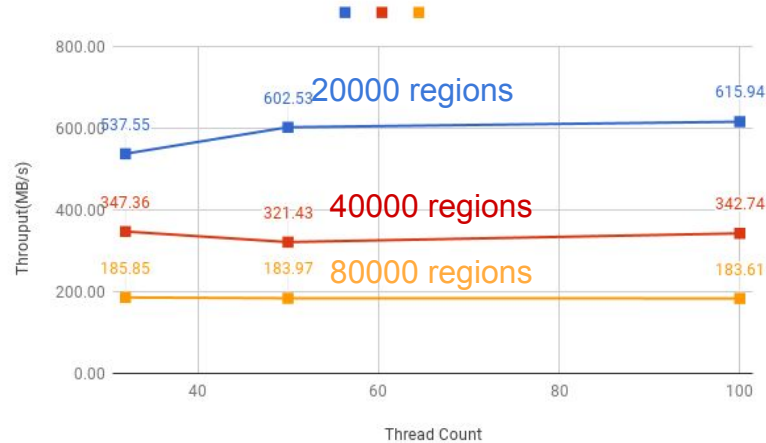


Performance

- Restart a cluster of 18 datanodes and 18 regionServers
 - Throughput = size of Hlog / cost time

total Region count	Hlog Size(GB)	thread count	cost time(seconds)	throughput(MB/s)
20000	136	32	253	537.55
20000	142.8	50	237	602.53
20000	127.5	100	207	615.94
40000	144.5	32	416	347.36
40000	135	50	420	321.43
40000	127.5	100	372	342.74
80000	161.5	32	869	185.85
80000	140	50	761	183.97
80000	144.5	100	787	183.61

Performance



- ❑ For one series, it almost has no change as the thread count increase
- ❑ More regions means more recovered log files will be generated.
- ❑ **The time of log splitting depends mainly on number of generated recovered log files.**



Recommend settings

- ❏ Enable this feature

Hbase.split.writer.creation.bounded = true

- ❏ Configurations:

hbase.regionserver.hlog.splitlog.bufferize (maxHeapUsage): set it equals or larger than the size limit of HLog.

hbase.regionserver.hlog.splitlog.writer.threads : Not having a equation to calculate. 32 is best according to our test

Part-2 Replication Improvement



Abstract

- ❑ New Replication Storage Layer
- ❑ Replication Task Flow
- ❑ Revisit Serial Replication



Abstract

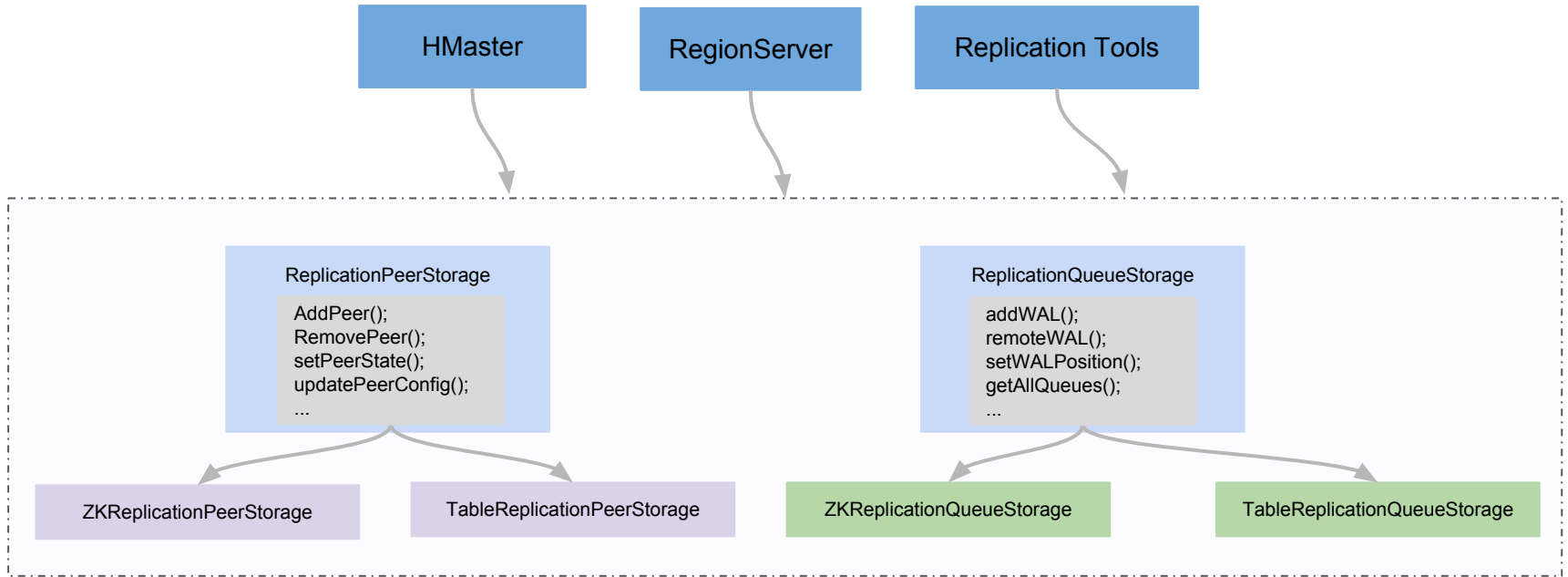
- ❑ **New Replication Storage Layer**
- ❑ Replication Task Flow
- ❑ Revisit Serial Replication

Problems of original replication storage

- Fuzzy interface definitions
 - Some classes for client, but others for region server, can integrate together.
 - ...
- No Consideration for the abstraction for table based replication storage.
 - Some methods throw KeeperException instead of ReplicationException
 -



New replication storage layer (HBASE-19397)

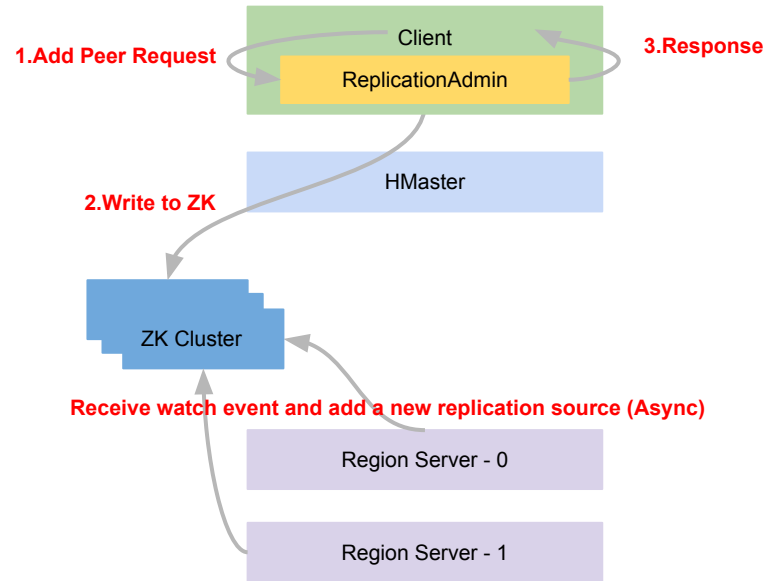




Abstract

- ❑ New Replication Storage Layer
- ❑ **Replication Task Flow**
- ❑ Revisit Serial Replication

Replication task flow - version #0

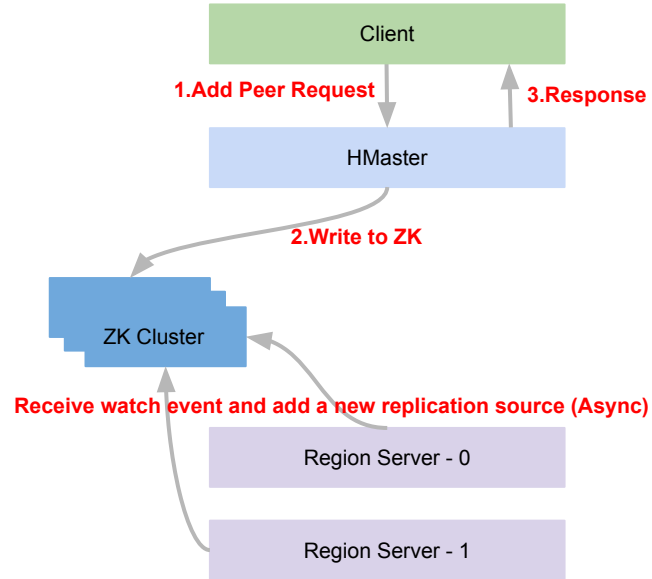


Release-1.4.x

Problems - version #0





- Expose the replication implementation to HBase client
 - The clients are allowed to read/write the replication znode(Security Problem).
 - Hard to authentication the request.
 - No coprocessor for interfaces.
- Async zookeeper notification from client to region server.
 - client won't know whether the task is success or not
 - Notification lost when RegionServer is not alive. (HBASE-12769)
 - Hard to implement the more complex task flow for Serial Replication & Sync Replication)

Replication task flow - version #1

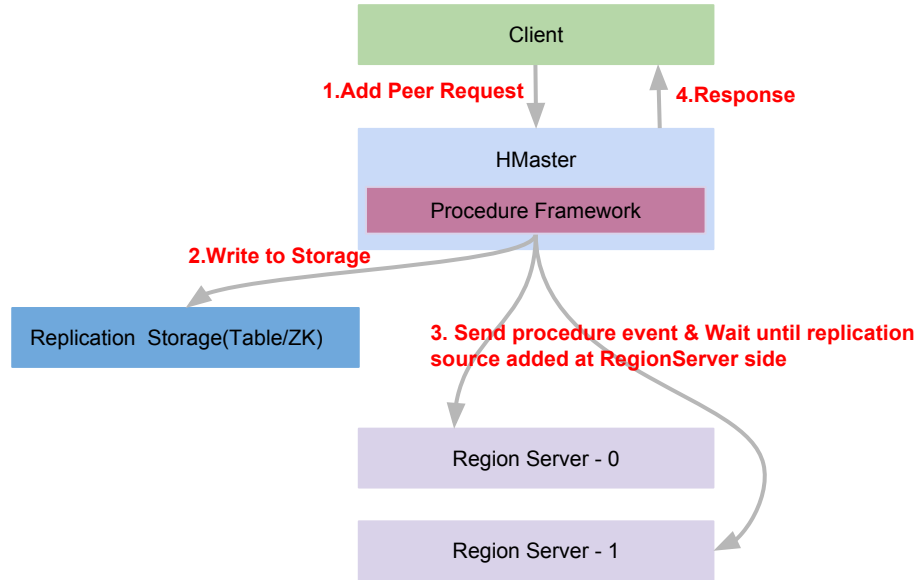


After HBASE-11392

Problems - version #1

- Expose the replication implementation to client 
 - The clients are allowed to read/write the replication znode 
 - Hard to implement request authentication at hbase level 
 - No coprocessor for interfaces in ReplicationAdmin. 
- Async zookeeper notification from client to region server.
 - Client does not know whether the peer is added success or failure at server side.
 - Notification lost when RegionServer is not alive. (HBASE-12769)
 - Hard to implement the more complex task flow (for Serial Replication/Sync Replication)

Replication task flow - version #2



After HBASE-19397

Problems - version #2

- Expose the replication implementation to client ✓
 - The clients are allowed to read/write the replication znode ✓
 - Hard to implement request authentication at hbase level ✓
 - No coprocessor for interfaces in ReplicationAdmin. ✓
- Async zookeeper notification from client to region server. ✓
 - Client does not know whether the peer is added success or failure at server side. ✓
 - Notification lost when RegionServer is not alive. (HBASE-12769) ✓
 - Hard to implement the more complex task flow (for Serial Replication/Sync Replication) ✓

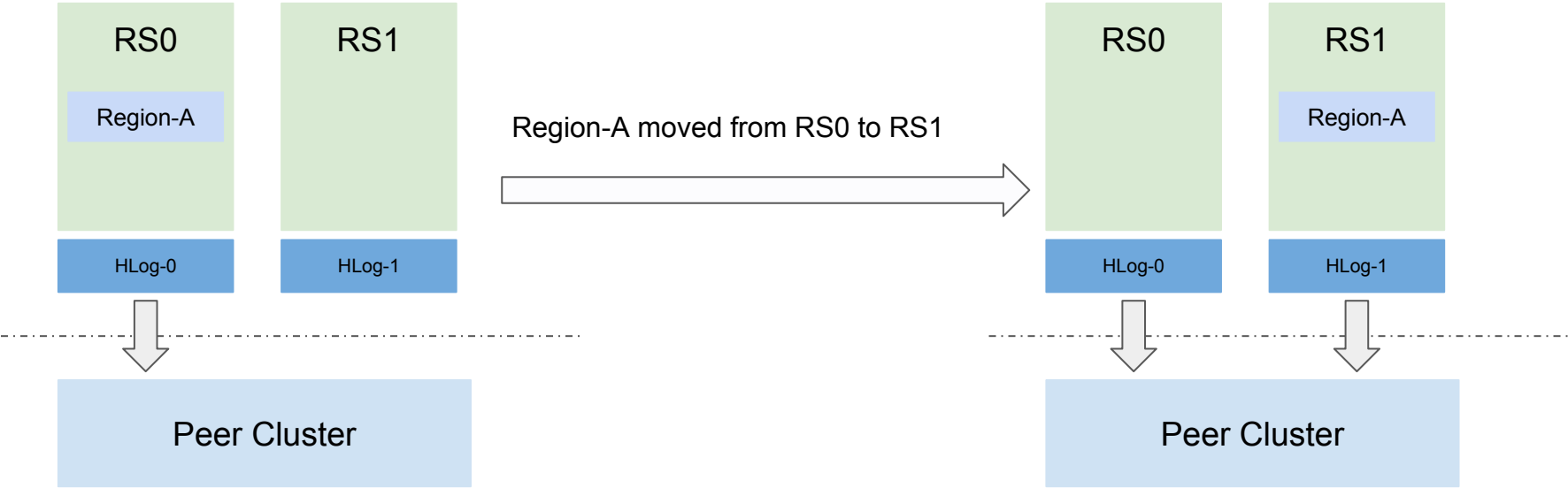


Abstract

- ❑ New Replication Storage Layer
- ❑ Replication Task Flow
- ❑ **Revisit Serial Replication**



Problem of normal replication



For region A, **Only RS0's HLog** will be replicated to peer cluster.

For region A, **Both RS0's HLog & RS1's HLog** will be replicated to peer cluster **in parallel**.

Normal replication - problem #1

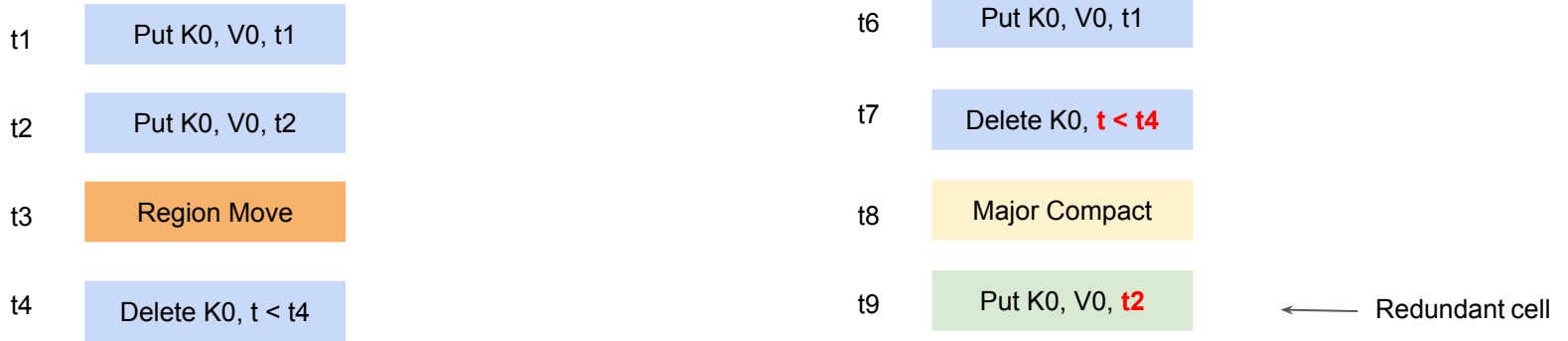


Sequence of mutations in source cluster.

A possible sequence of mutations in peer cluster

- Mutations are **out-of-order** in peer clusters. Disaster for pub/sub, message system etc.

Normal replication - problem #2



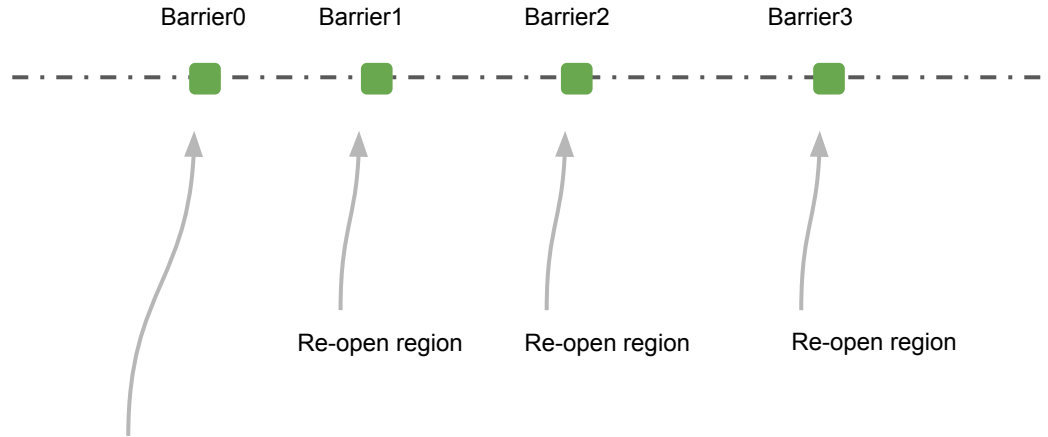
Mutation sequence in source cluster.

A possible mutation sequence in peer cluster

- The source cluster has no data, but the peer cluster has a cell (Put K0, V0, t2) in the end.
- **Data inconsistent happen** between source cluster and peer clusters.

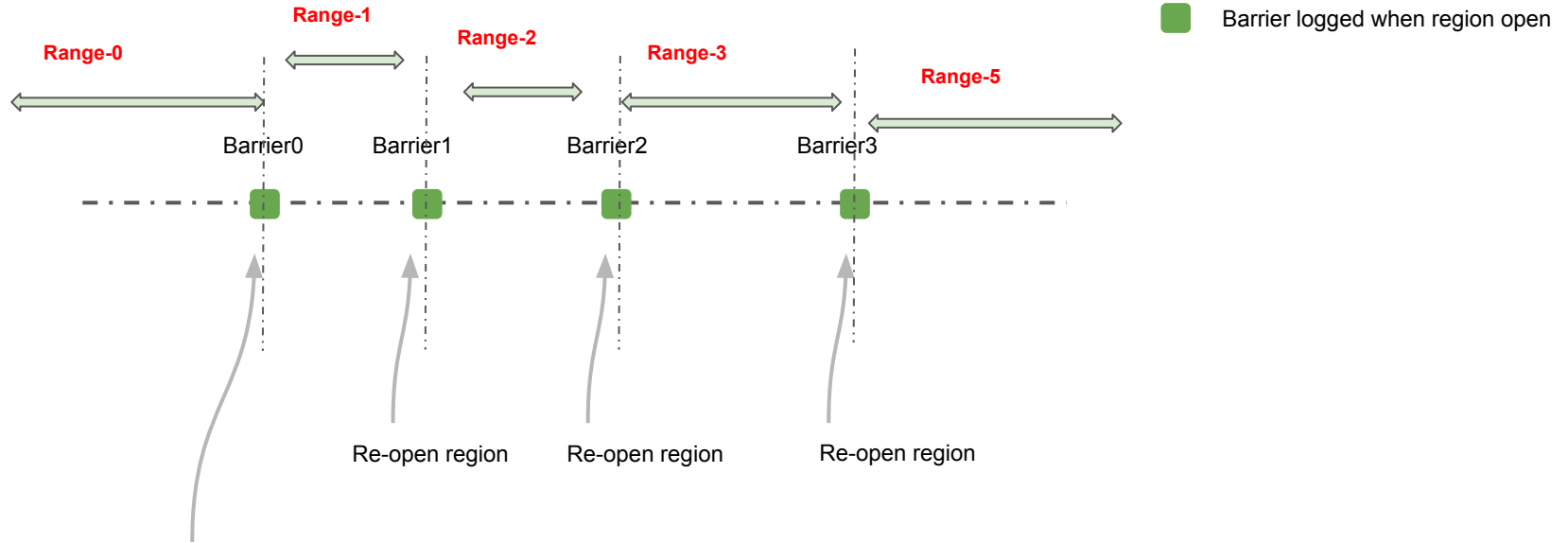
Core idea of serial replication

■ Barrier logged when region open






The initialized barrier when set peer to be serial

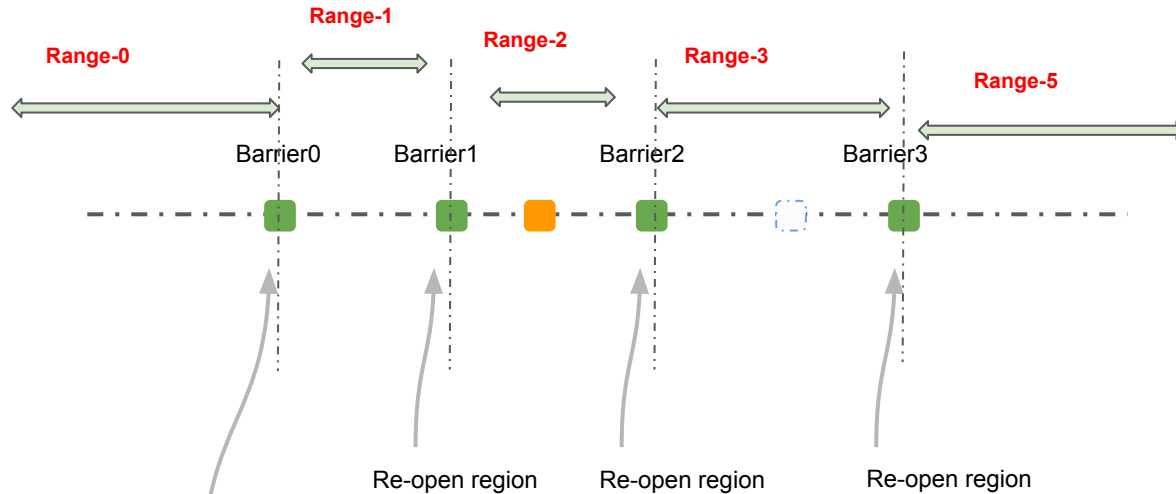
Core idea of serial replication



The initialized barrier when set peer to be serial

Core idea of serial replication

-  Last pushed sequence ID
-  Barrier logged when region open
-  Pending seq id to check whether we can push now.



The initialized barrier when set peer to be serial

Core idea of serial replication

- ❑ For each region, we save two sequence id(s):
 - ❑ **Last Pushed Sequence ID**: Update this id for the specific region after replicated a WAL entry to peer cluster, it means the progress of the serial peer.
 - ❑ **Barrier**: Will store the region's openNum as its barrier for each region when RS open it.

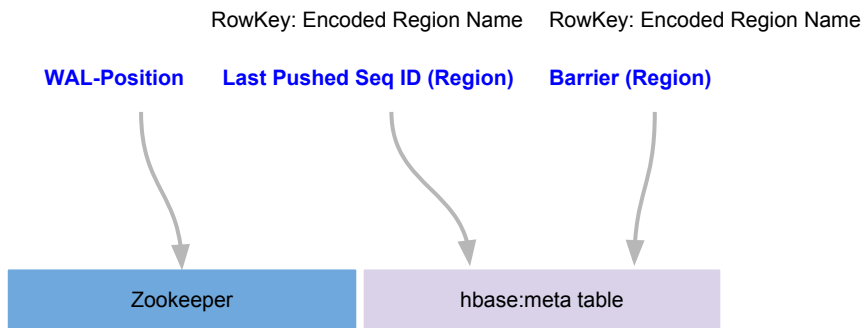
- ❑ **Last Pushed Sequence ID VS WAL Position**:
 - ❑ Last Pushed Sequence ID is a sequence id for a given region.
 - ❑ WAL Position is the latest replicated offset for current W-A-L file.

Why to reimplement the serial replication #1

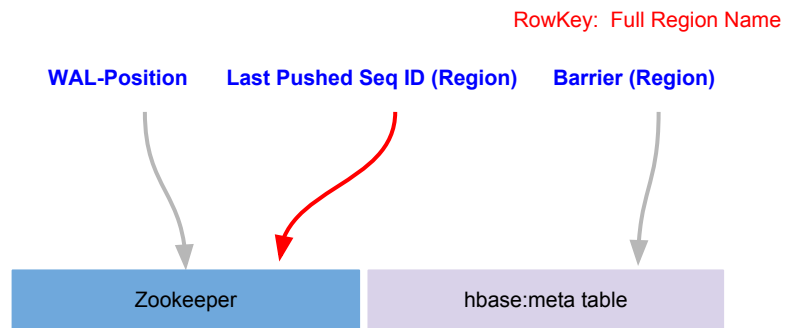
- We stored the Last Pushed Seq ID in hbase:meta
 - WAL Entry only has an **encoded region name**, but rowkey in hbase:meta is a **full region name**, the two different rowkey format messed up the hbase:meta.
 - Inconsistent between **WAL position**(ZK) and region's **Last Pushed Seq ID**(meta table)
- After HBASE-19397, we introduced the New Replication Storage Layer, Need to integrate the serial replication with it.



Move the last pushed seq id to zookeeper



Original Design

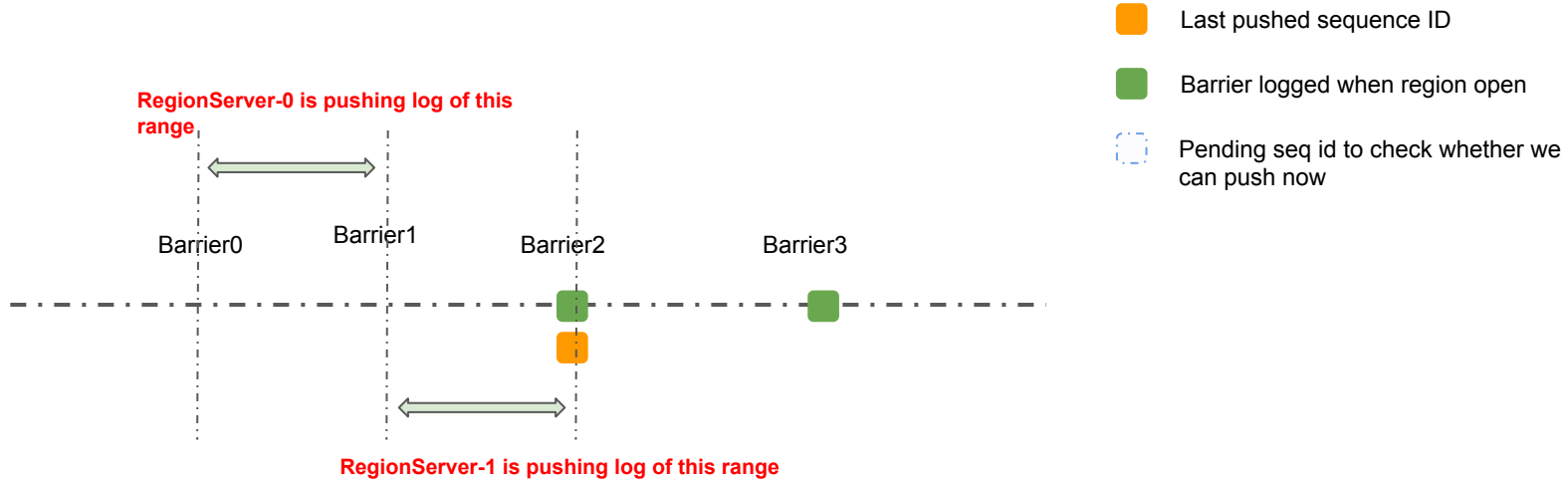


New Design

Why to reimplement the serial replication #1

- We stored the Last Pushed Seq ID in hbase:meta
 - WAL Entry only has an **encoded region name**, but rowkey in hbase:meta is a **full region name**, the two different rowkey format messed up the hbase:meta. ✓
 - Inconsistent between **WAL position**(ZK) and region's **Last Pushed Seq ID**(meta table)
- After HBASE-19397, we introduced the New Replication Storage Layer, Need to integrate the serial replication with it. ✓

Why need to update WAL-pos/seq-id in CAS



- Both RS-0 and RS-1 are updating their last pushed seq id
- The latest last pushed seq id written by one region server **may be overwritten by other RS**
- The serial peer will be stuck **if last pushed seq id is not strictly increasing**

Solution to update WAL-pos/seq-id in CAS

- Read **current znode version** & last pushed seq id ;
- If the new last pushed seq id \leq last pushed seq id ; then skip to update;
- Else persist the new last pushed seq id by **setData with the current znode version**.
- If no version conflict, then the CAS is success.
- Else just retry the Step.1 again.

Why to reimplement the serial replication #1

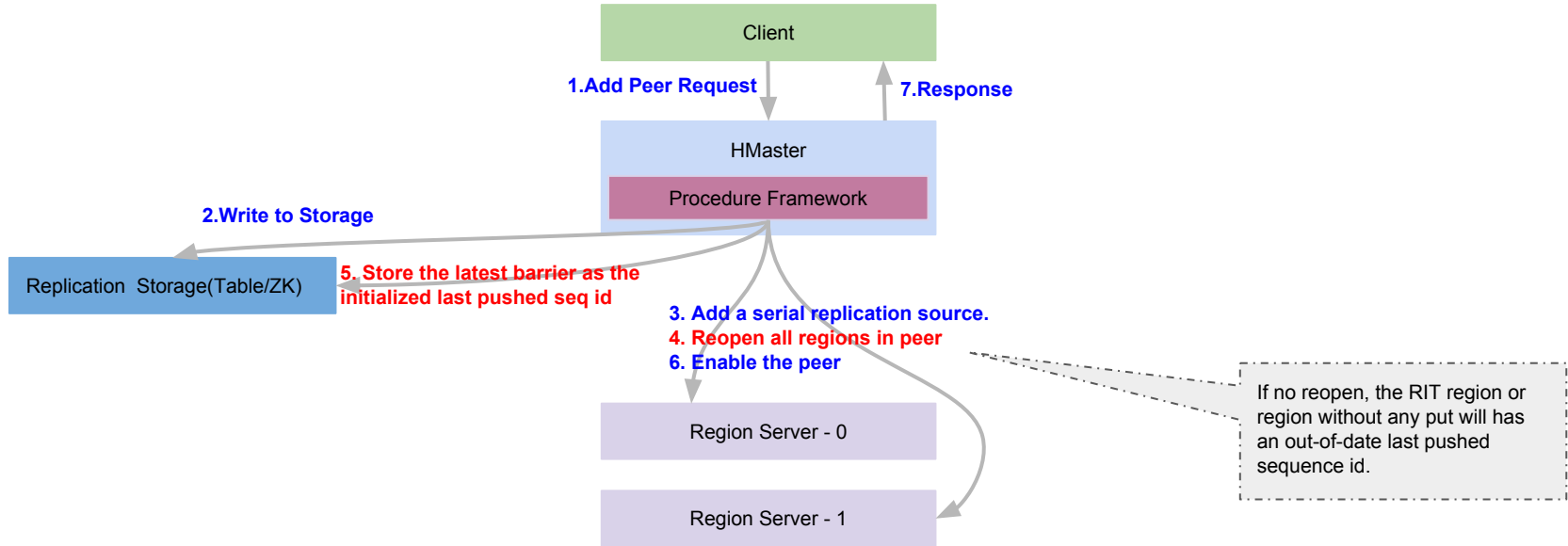
- We stored the Last Pushed Seq ID in hbase:meta
 - WAL Entry only has an **encoded region name**, but rowkey in hbase:meta is a **full region name**, the two different rowkey format messed up the hbase:meta. ✓
 - Inconsistent between **WAL position**(ZK) and region's **Last Pushed Seq ID**(meta table) ✓
- After HBASE-19397, we introduced the New Replication Storage Layer, Need to integrate the serial replication with it. ✓



Why to reimplement the serial replication #2

- No consideration for the initialization of last pushed seq id, which may block the serial replication. (HBASE-20147)
 - BTW, we moved the serial attribution from table to peer.

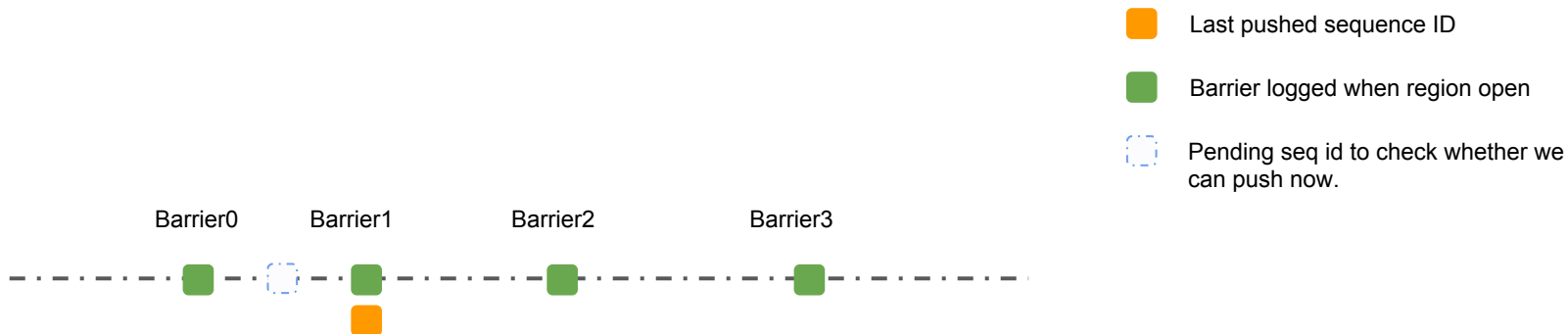
Initialize the last pushed seq id



Procedure for adding a serial peer

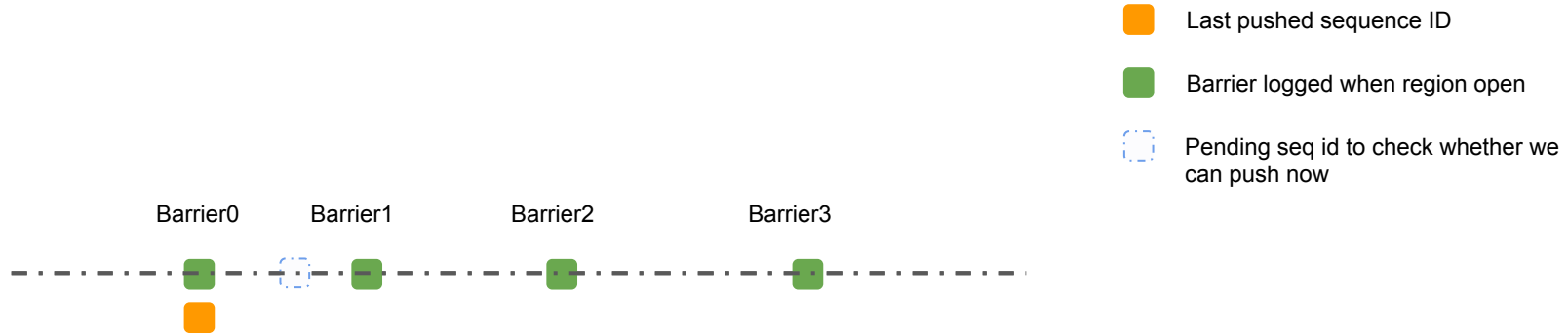


Serial replication checker #1



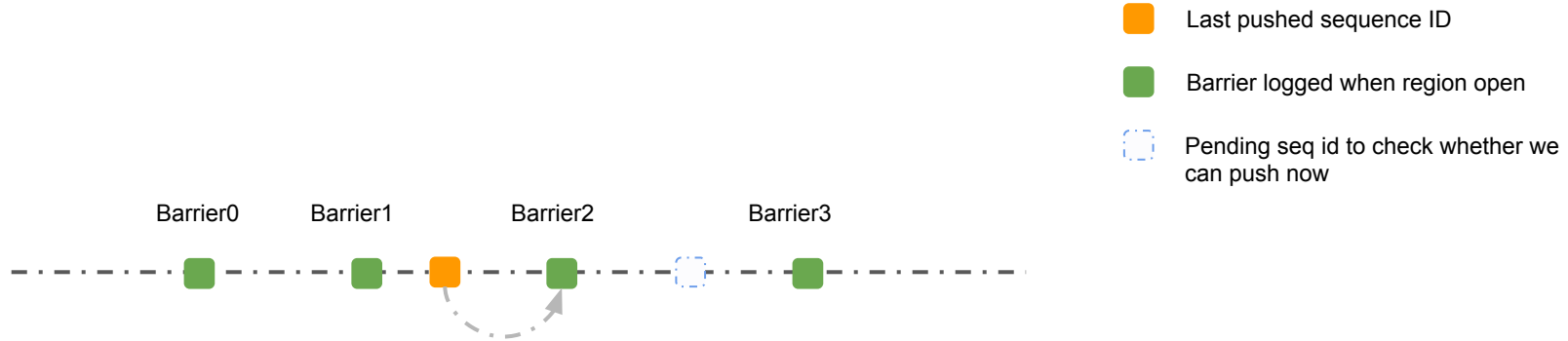
- We **won't guarantee the serialization**, for whose seq id is less than the initial last pushed sequence id.
- Now, the pending sequence id < the last pushed sequence id(Barrier1). So, just replicate the current WAL entry.

Serial replication checker #2



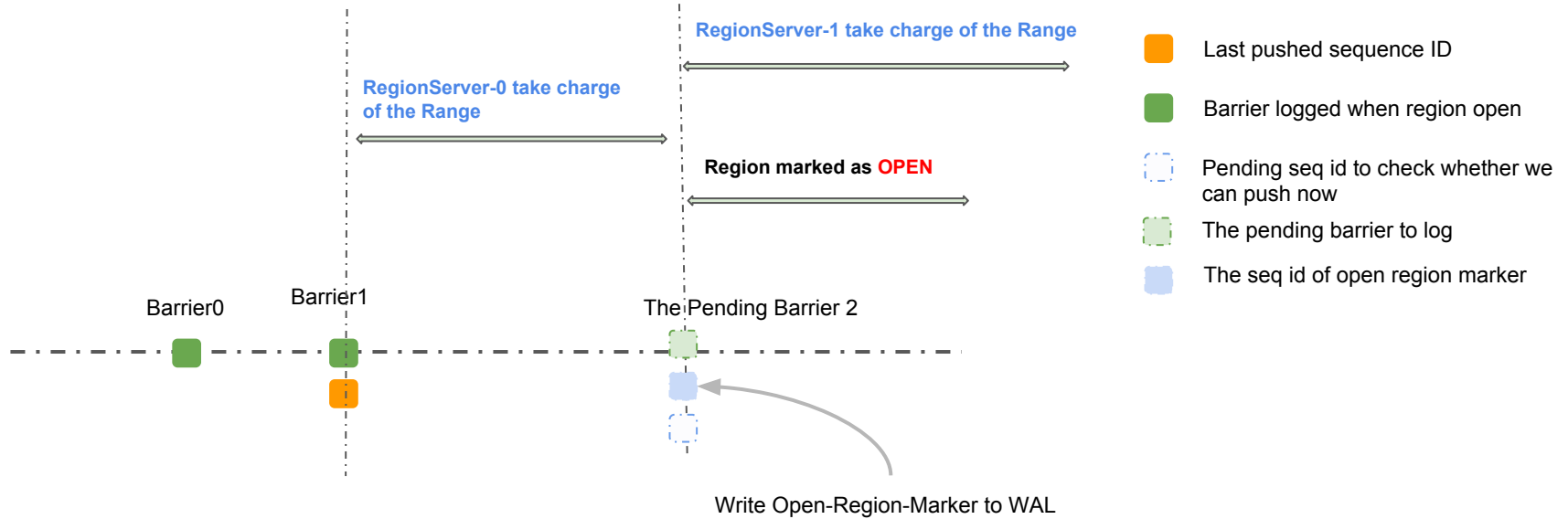
- The pending seq id **is in the first range of barriers**.
- It's possible that the region is the split/merge result of other region(s). So wait until its parent to be fully replicated (if parent exists), otherwise, just replicate the entry to peer cluster.

Serial replication checker #3



- The pending seq id is in the range of barriers, **but the last pushed seq id haven't reached now.**
- So wait until the last pushed seq id \geq Barrier2 - 1

Serial replication checker #4



Serial replication checker #4

- The pending sequence id to check is located **in the last interval** [barrier1, +oo), if the region is **OPENING** and last pushed sequence id has reached the latest barrier1, we still need to wait until the region to be **OPEN** state.
- **Step.1** Master mark the region to be **OPENING** state firstly, then request the RS1 to open region.
- **Step.2** RS1 open the region and **write an open-region-marker to WAL**.
- **Step.3** Master mark the region to be **OPEN** state, and then write the pending barrier2.
- So if replicate the open-region-marker while region is opening, the RS1 will replicate its entries even if entries in previous interval haven't been fully replicated, which break the serial replication.

Thank You !