Tencent Big Data

# How Tencent uses PGXZ(PGXC) in WeChat payment system

**jason(李跃森)**
**jasonysli@tencent.com**

# About Tencent and Wechat Payment

- Tencent, one of the biggest internet companies in China.

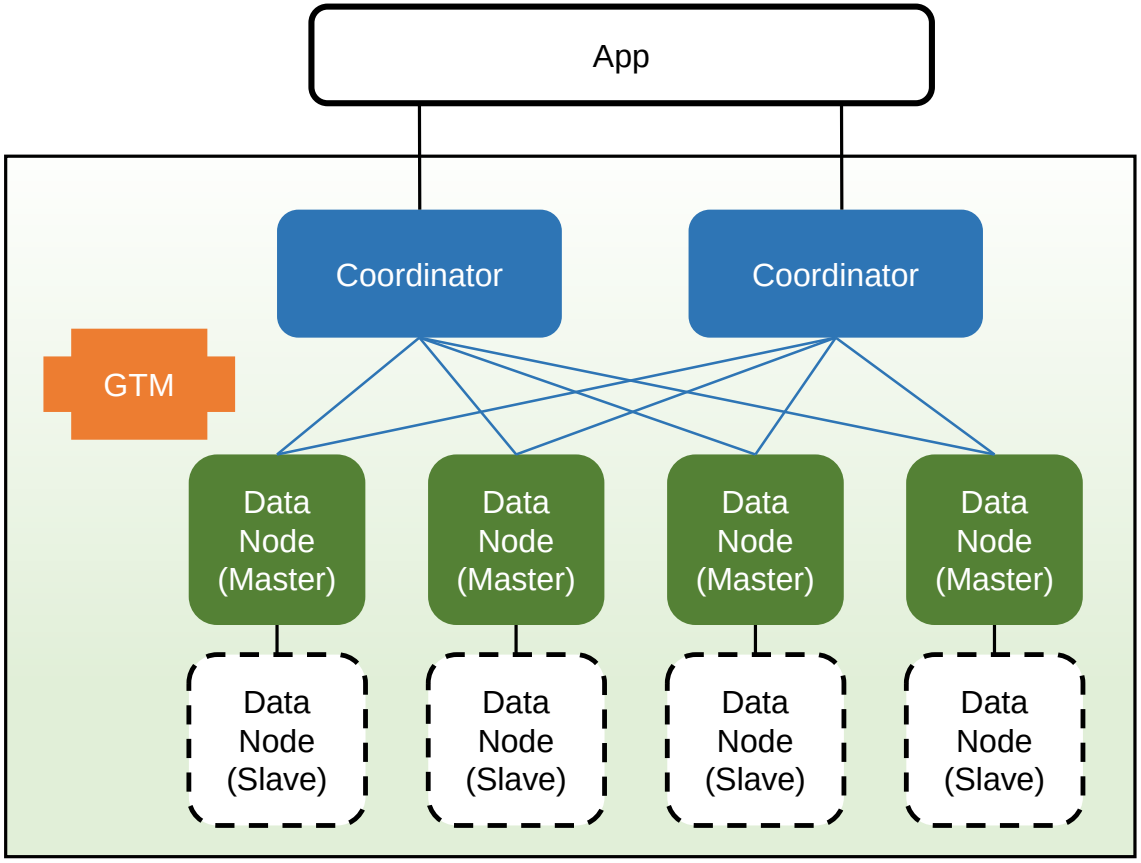

- Wechat, the most popular social network APP in China.

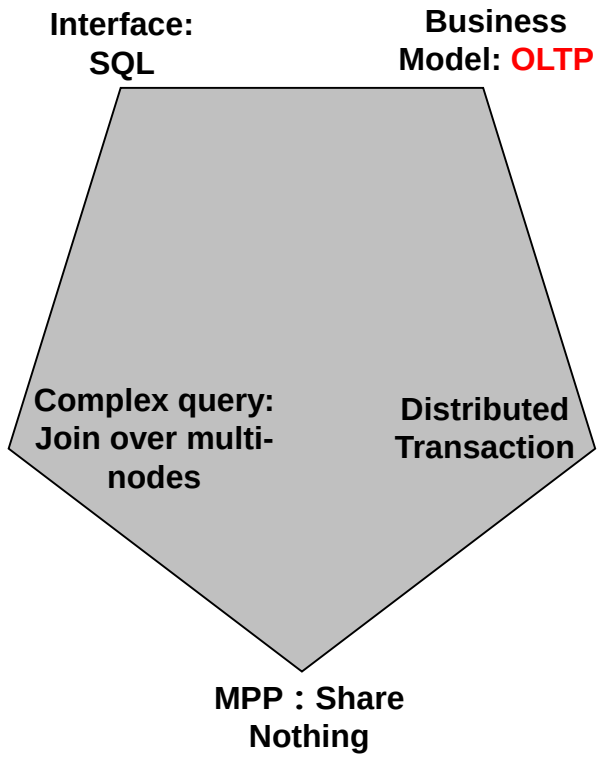

- Wechat Payment, wireless, fast , secure, efficiency payment solution provided by Wechat.

# Why PGXZ(PGXC)

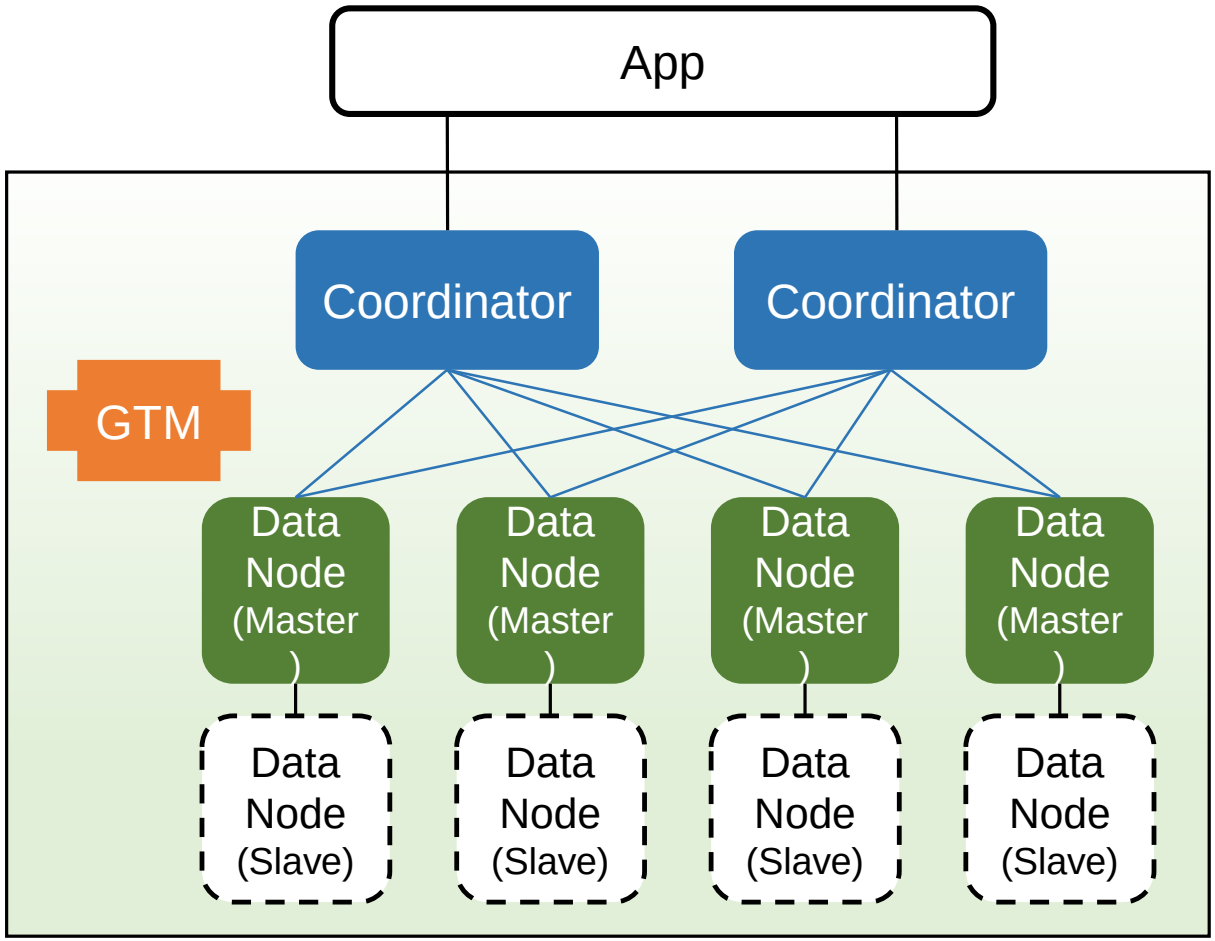
- Data size is HUGE(30TB per year and increasing): database cluster
- SQL is more flexible and popular(Compared to key value and other solution)
- Complex query(JOIN ACROSS multi-datanode)
- Distributed transaction(Easy to use)
- High transaction throughoutput(Peak 1.5 Million per min)
- Capability of  horizontal scale out(Read and Write workload)

# PGXZ Architecture: 5 aspects



# PGXZ Architecture:




**Coordinator：**

1. Access entrance of the cluster
2. Sharding data
3. Routing shard
4. Process tuples of transaction over multi-nodes
5. Only catalog, no user data

**DataNode:**

1. Store application data
2. DML ops against data resided in this node
3. Replication

**GTM：**

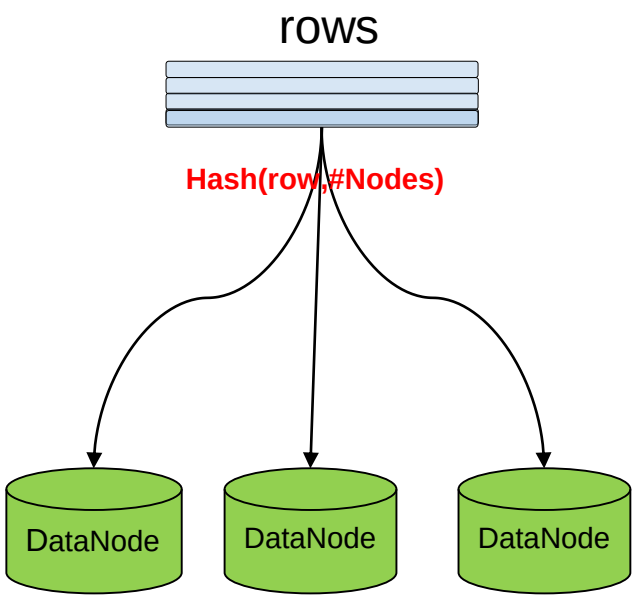
1. Manage transaction traffic

**Schema：**

1. One copy for every node
2. Every coordinator has the same global catalog info.

# Online add/remove node--Data Sharding

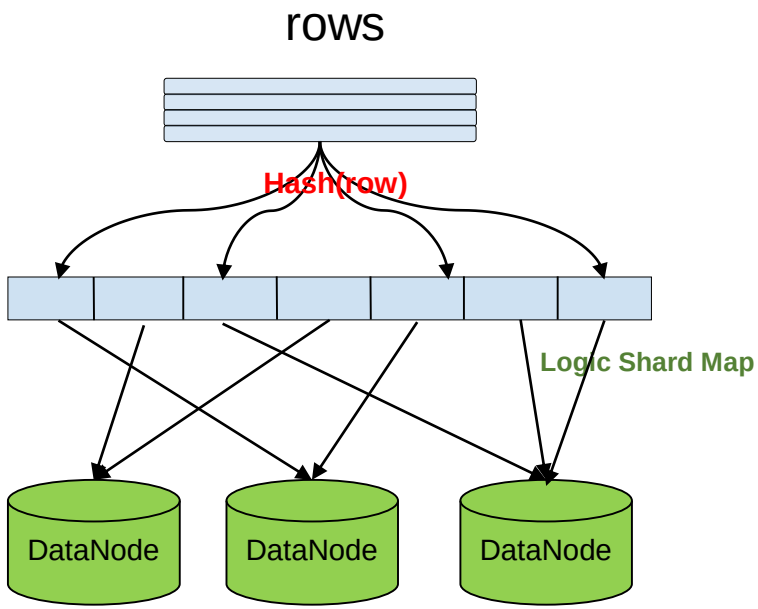
PGXC




hash_value = Hash(row)
nodeid= hash_value % #nodes

**MUST** rehash all data of whole cluster when add/remove datanode

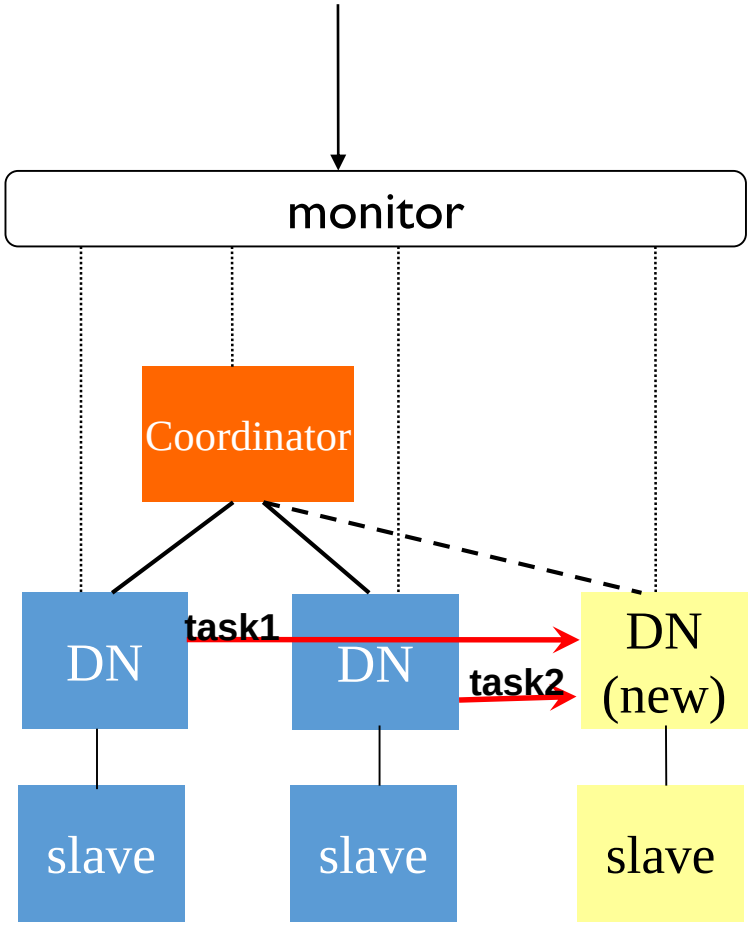
PGXZ




shardid = hash(row)
nodeid = map(shardid)

Data routing is decoupled from the number of data nodes by adding Logical Shard Map

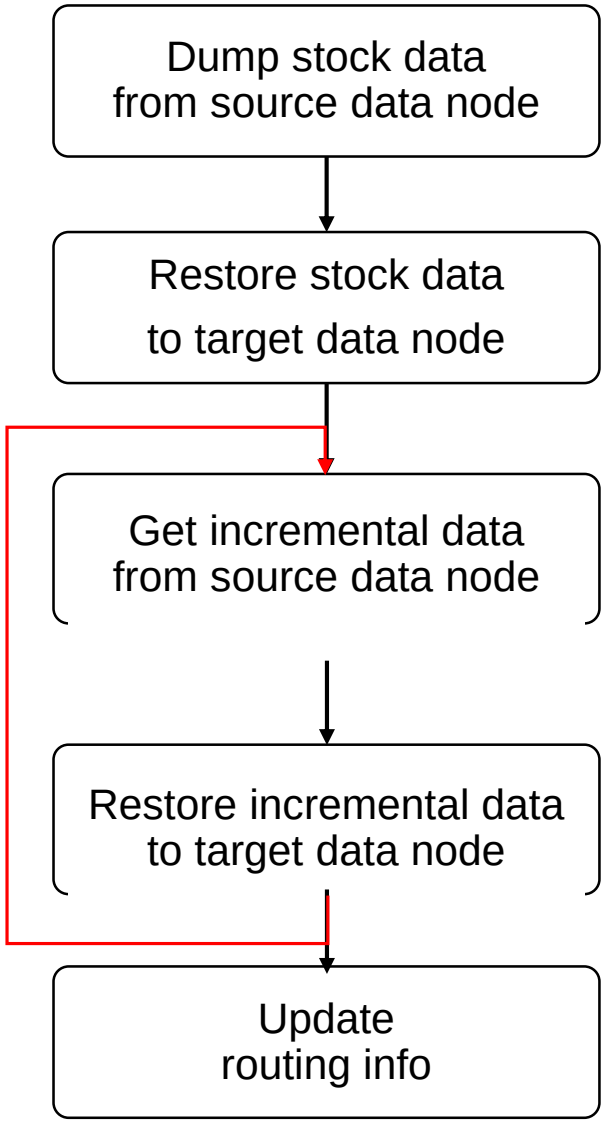
# Online Scale out--Steps




**Monitor :**

1. **Add data node to cluster**
2. **Make a PLAN of moving data to new data node**
    1. **A PLAN consists of several data moving TASKS**
    2. **One data moving TASK can move several shards**
    3. **One data moving TASK is a atomic operation**

# Online add/remove node-Data Moving




1. Start to dump stock data with a snapshot
2. Start logical decoding with same snapshot

**Ensure there is no overlap or miss between stock data and incremental data**.

1. Restore stock data to target data node

1. Get incremental data from source data node

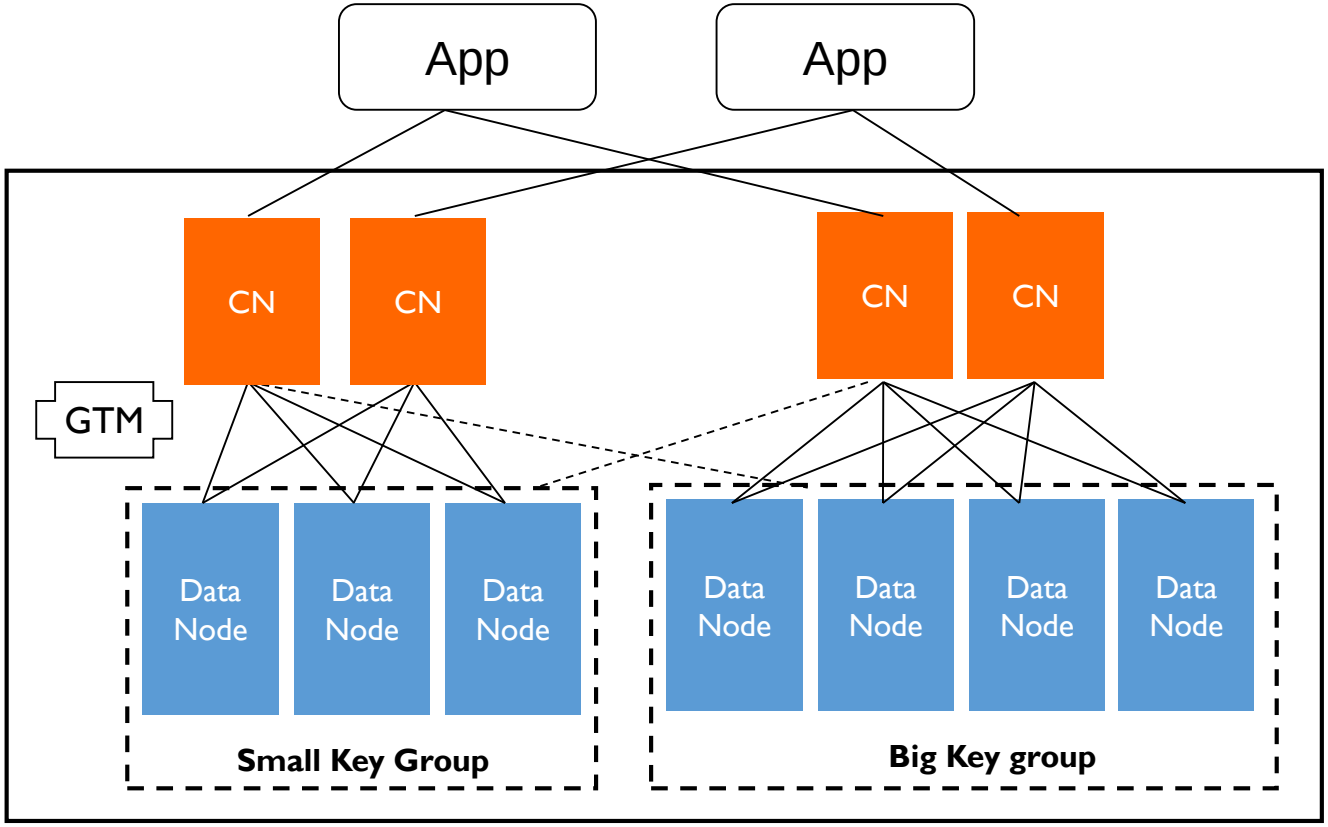
**Ensure there is no overlap or miss between two iterations of logical decoding(incremental data)**

1. Restore Incremental data to target data node
2. Go to next iteration

**If rows written at this iteration is less than a threshold, then go to update routing info**

1. Block writing at moving data(shard/partition),
2. Update routing info at all of coordinators,
3. Release blocking (20ms ~ 30ms)

# Multi-Groups: Handle huge merchants(Data Skew)



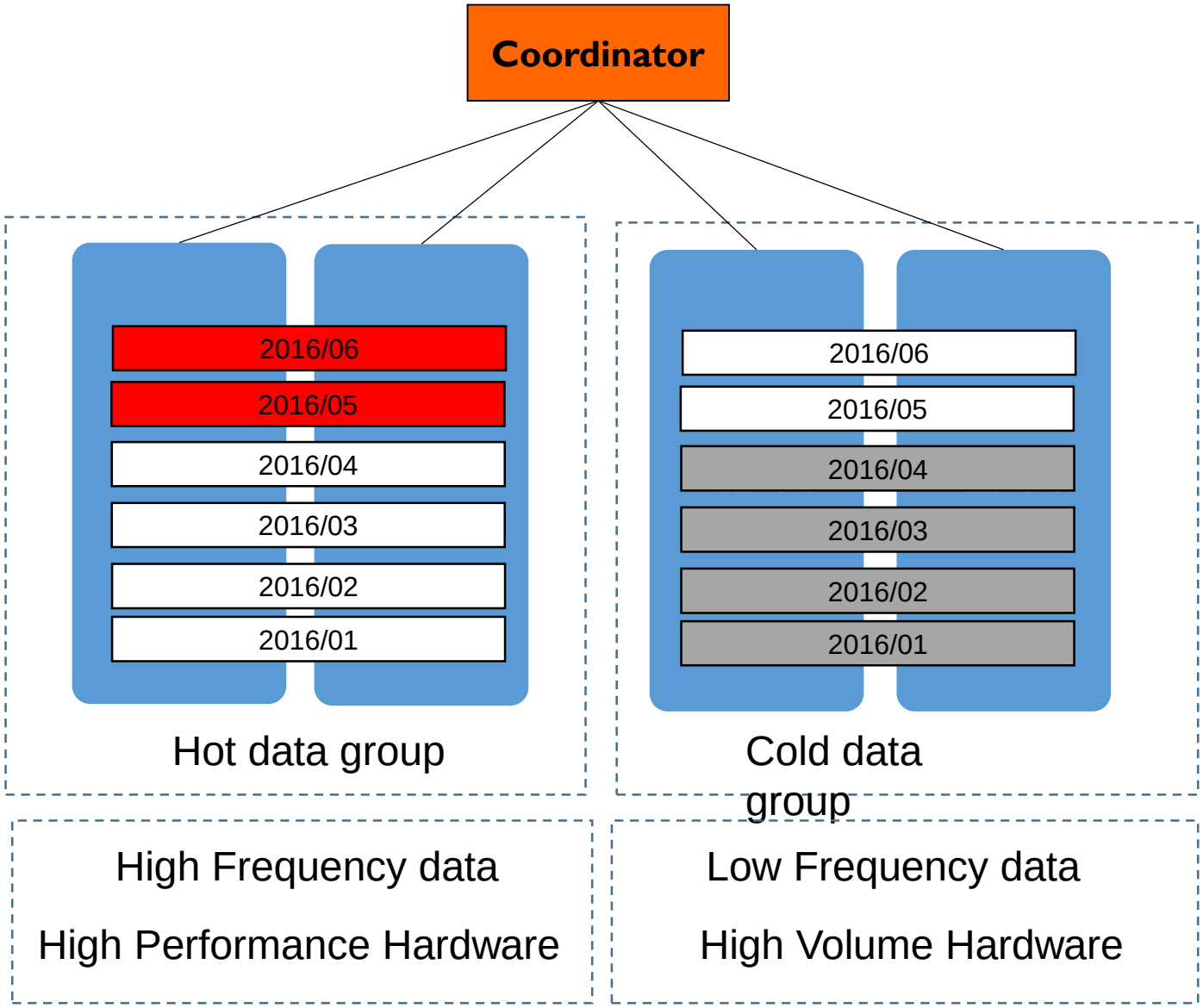

**Small Key Group:** Data belongs to one merchant resides at only one data node.

- ❑ No distribute transaction when writing one small merchant.
- ❑ No join across multiple data nodes when query data from one small merchant.

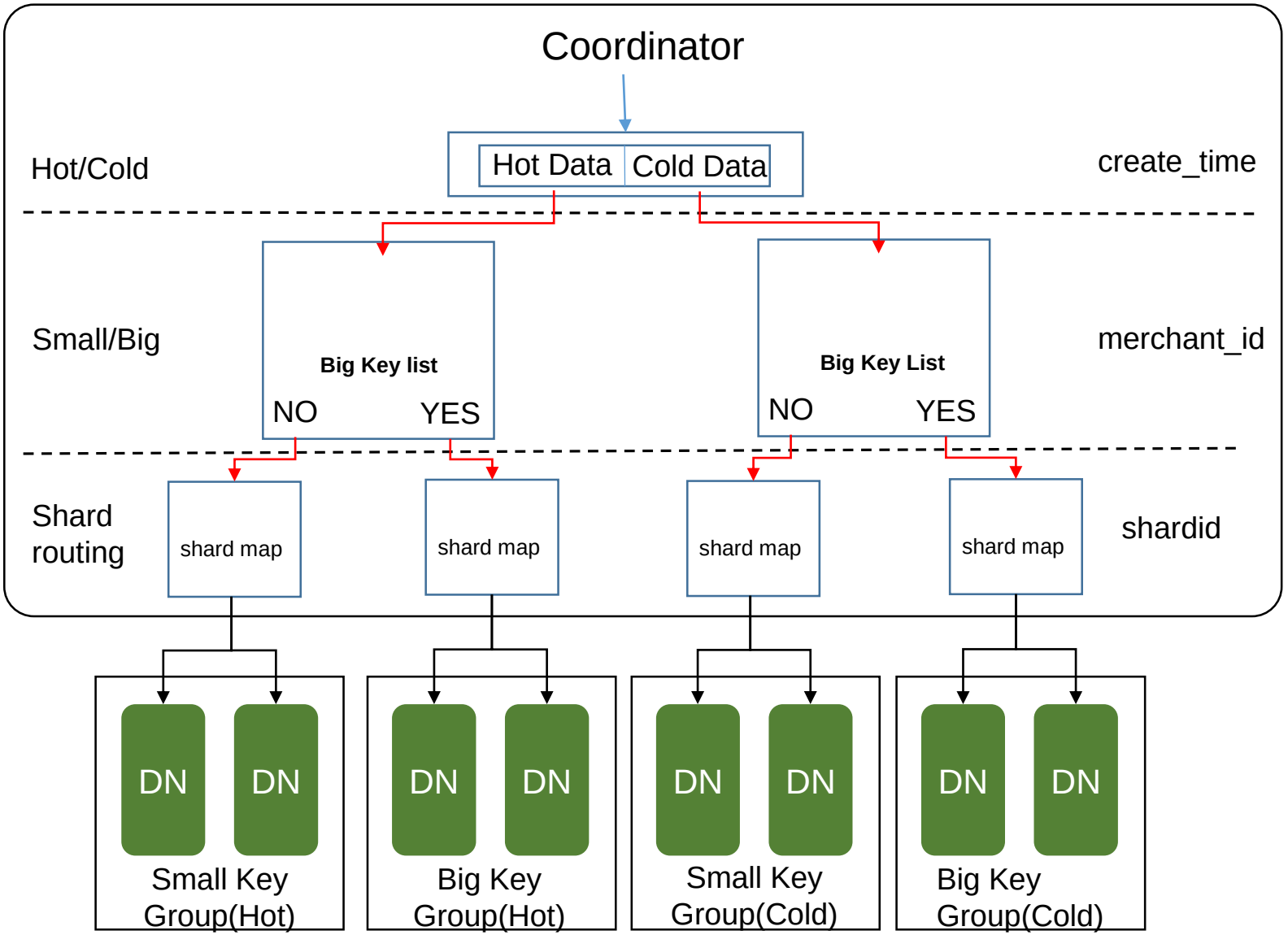
**Big Key Group:** Scatter One merchant data to all of data nodes inside the group. One merchant data generated in same day reside at same data node.

- ❑ No matter how big the merchant is, it can be stored in this cluster.

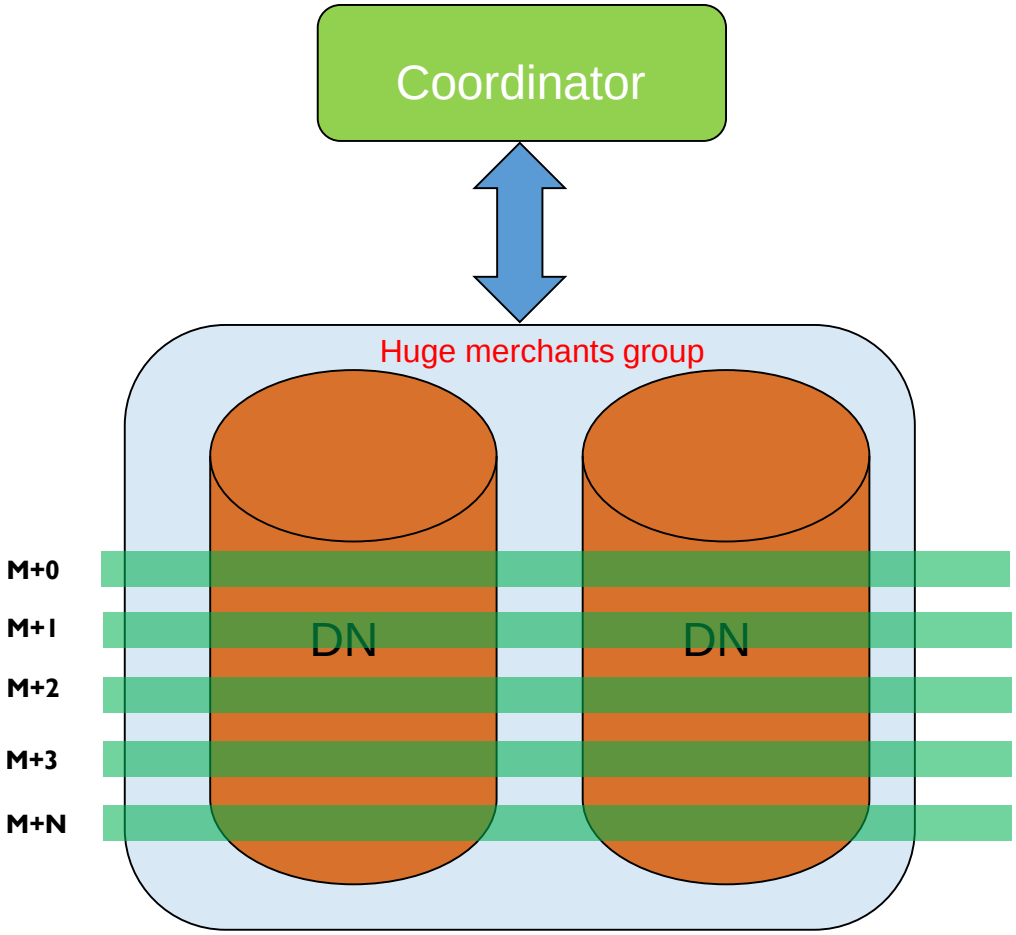
# Multi-Groups: data stored separately by access frequency ( Cost Saving )



# Multi-Groups: Routing to 4 groups

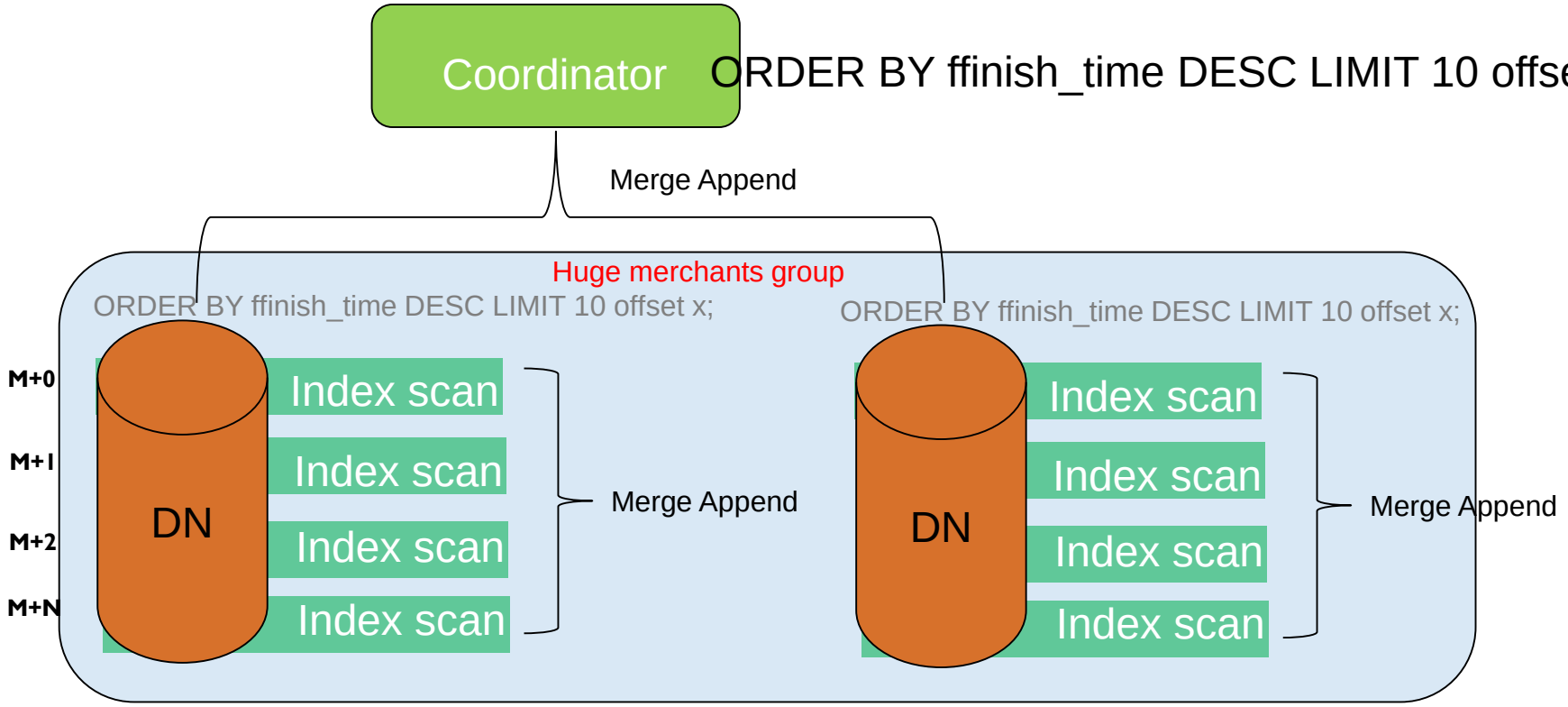


# Plan optimization--Huge Merchant order query on 90 million rows



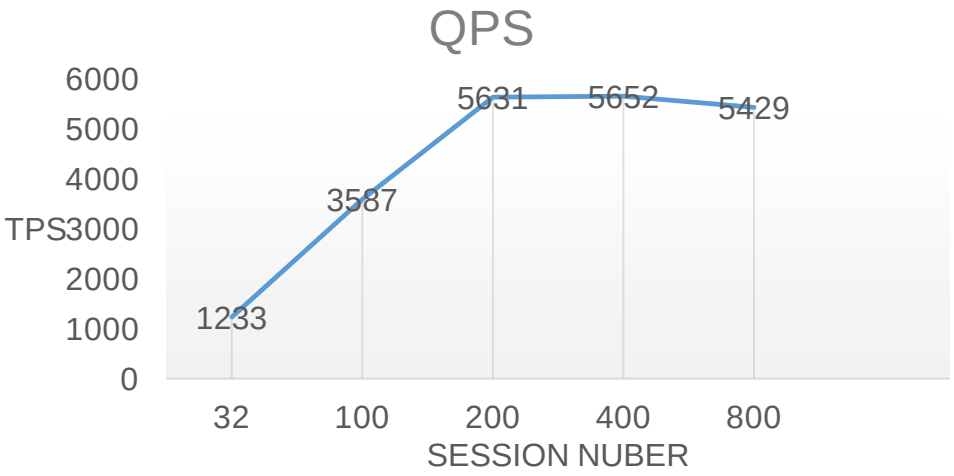

- SQL:select * from t_trade_ref where (ftotal_amount between 1 and 683771 and fcreate_time >= '2015-07-31 00:00:00' AND fcreate_time < '2015-08-30 00:00:00' and fmerchant_id = 7777777) ORDER BY ffinish_time DESC LIMIT 10 offset x;
- Table partitioned by month on the created time of row
- Create multi-column index on fmerchant_id and ffinish_time. ffinish_time is used for order by.

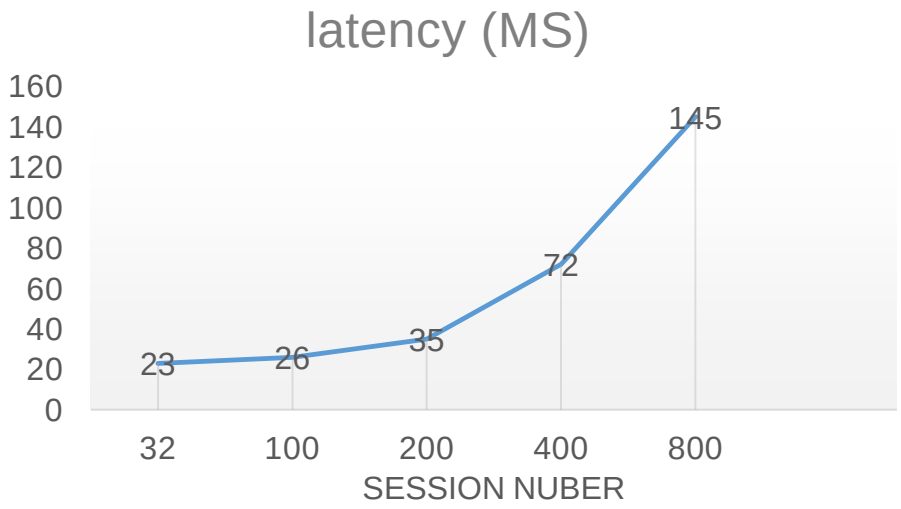
# Plan Optimization—Plan detail




- CN pushes down limit offset.
- DN uses index scan for each partition.
- DN combine multi partitions by Merge Append.
- CN combines multi datanodes by Merge append.

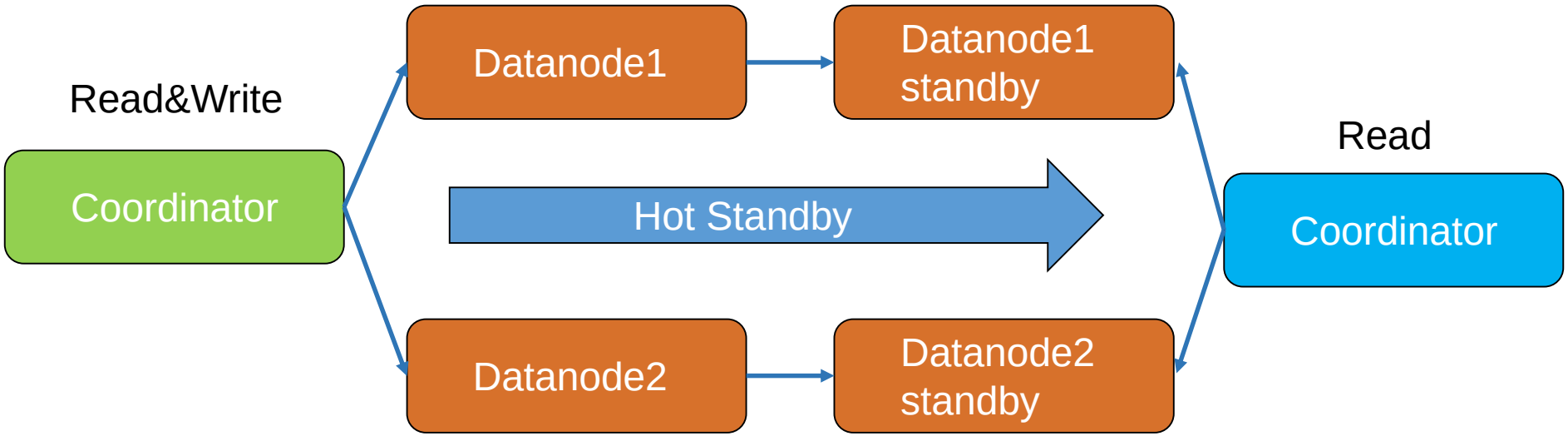
# Plan optimization-performance result



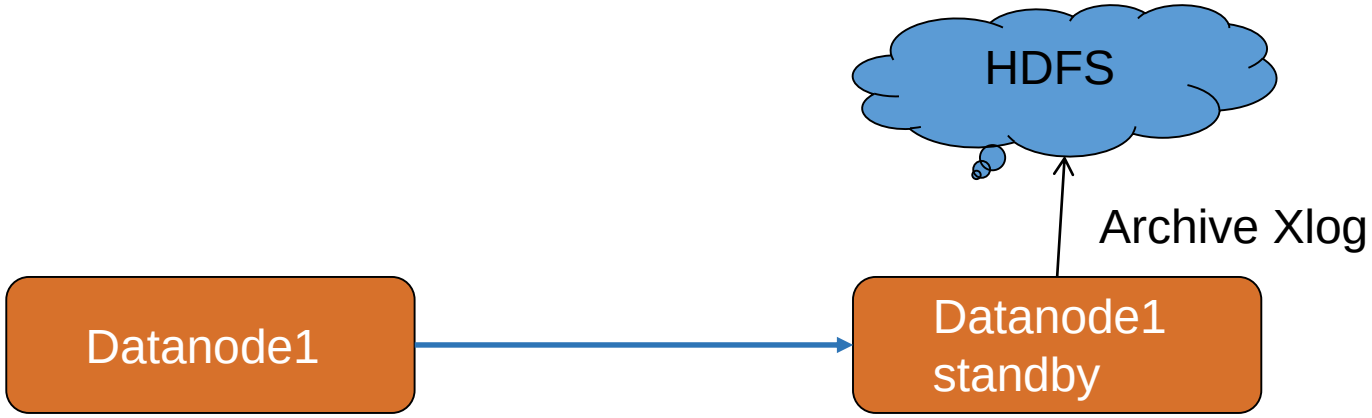

- Finish query within less than 50ms.
- QPS can reach up to 5 thousand.

# Load Balance:Use standby datanodes to provide read only service



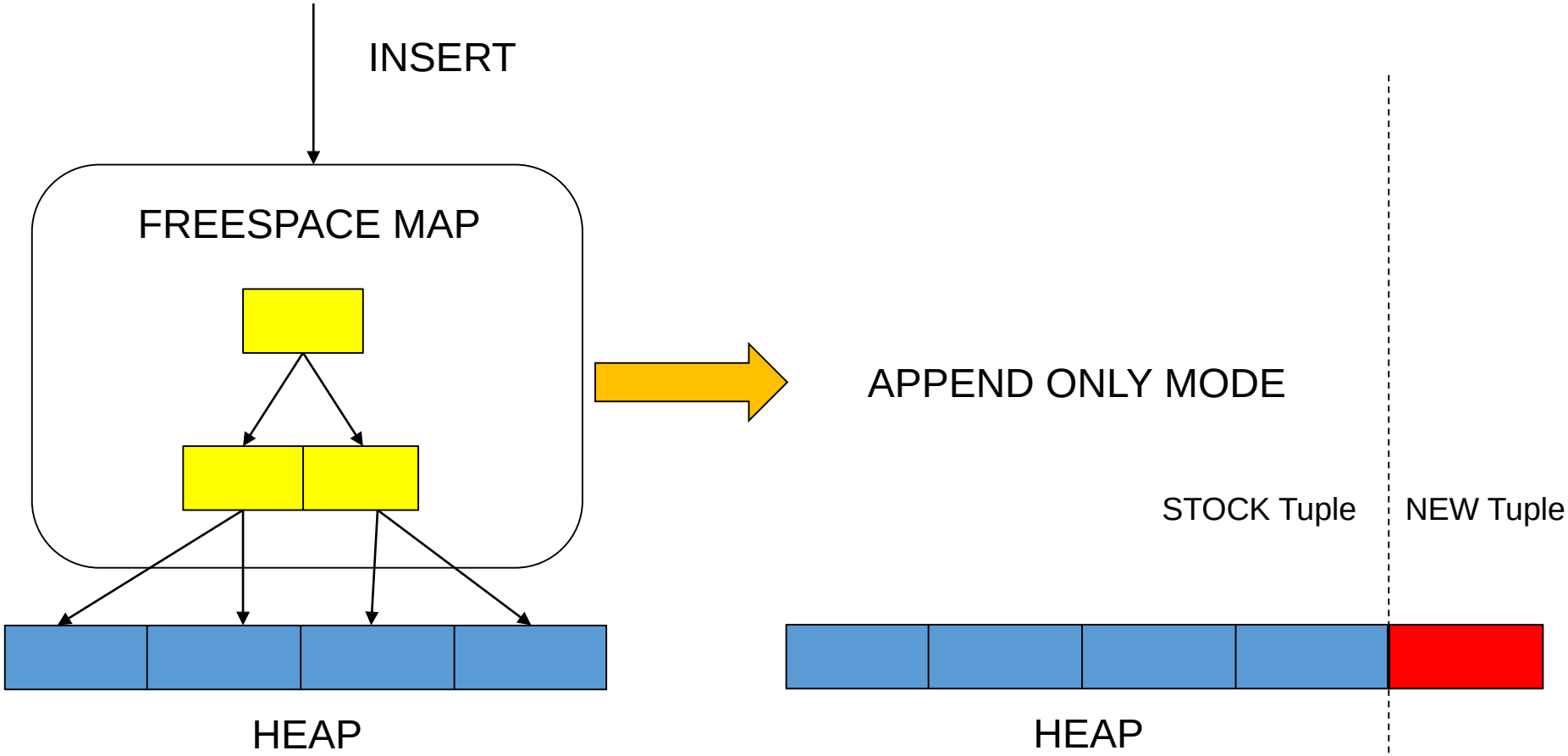

- Separated kind of coordinators
- Read only coordinators' catalog have entries of hotstandby datanodes
- Master datanode ships log  to standby in hot standby mode

# Load Balance:Use standby datanode to archive xlog



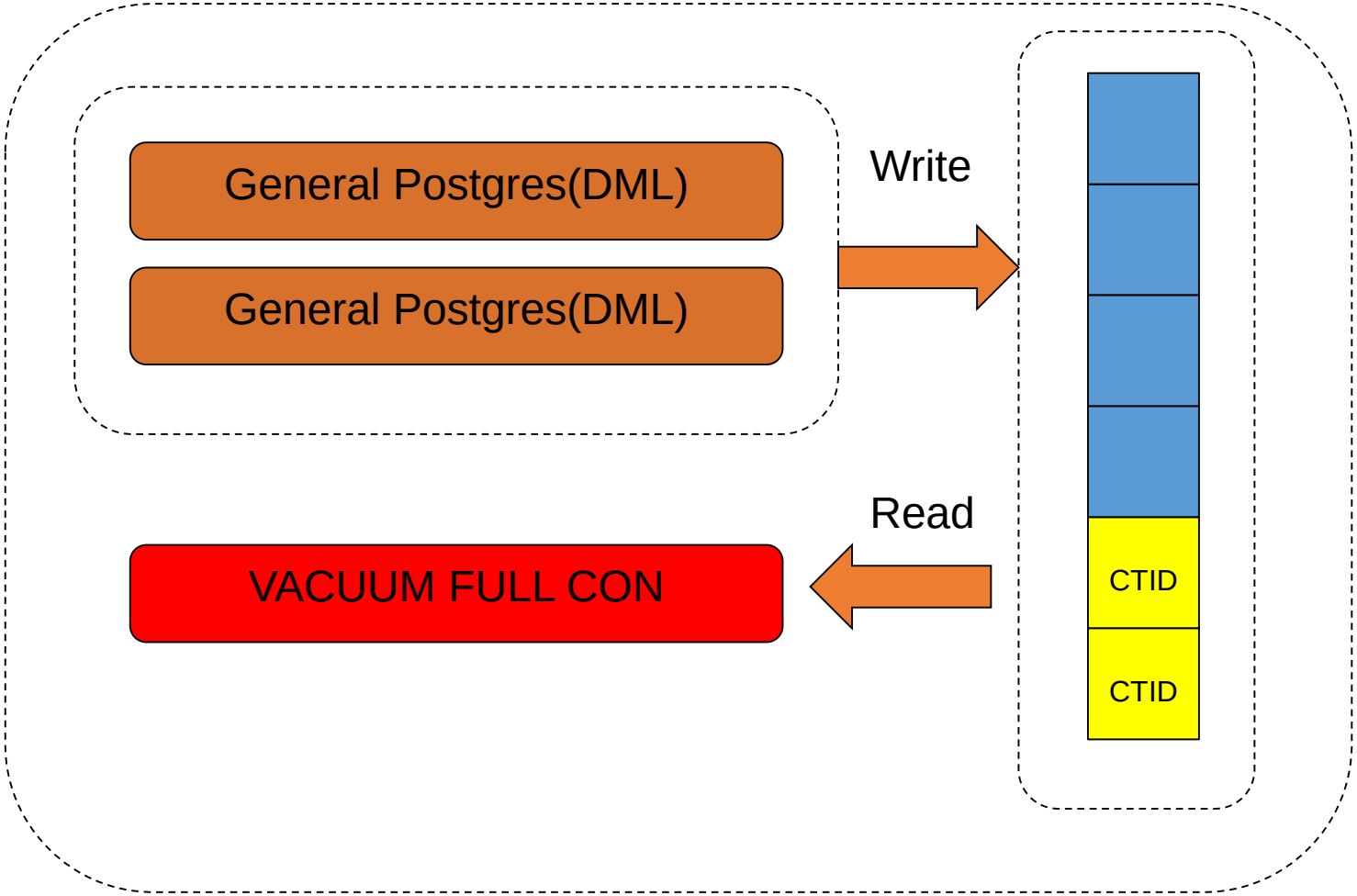

- High work load on master datanode
- Use standby datanode to do the archive job

# Online Table reorginize:Apend only heap mode



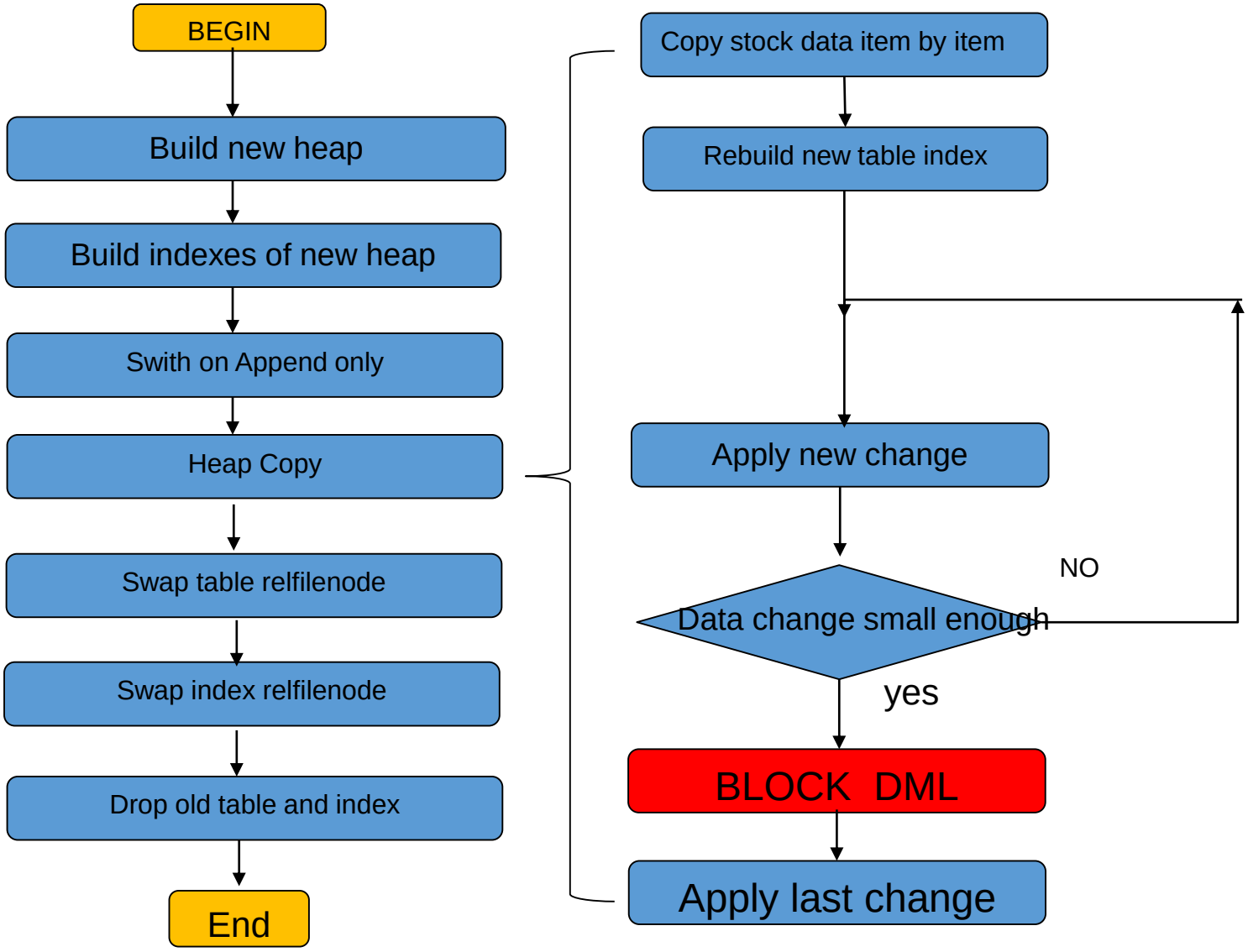

- Append tuple to the end of Heap, so we can know the new coming tuple
- Append only mode can switch on/off

# Online Table reorginize:Shared CTID pipeline

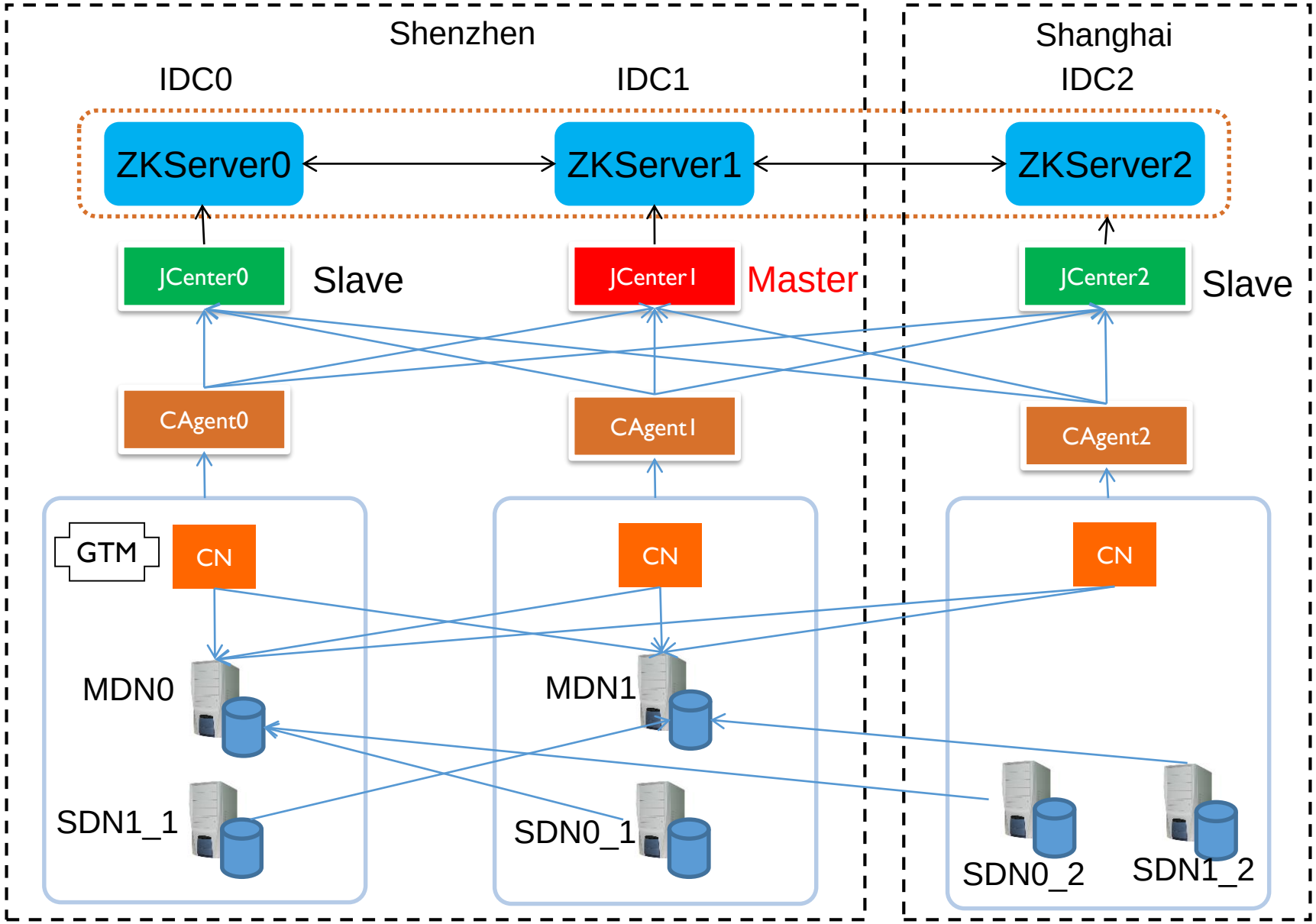



- Use shared pipeline to transfer tuple modification to VACUUM FULL process
- General postgres write to the shared pipeline
- Vacuum full process read citd info from the pipeline

# Online Table reorginize:detail steps



# High Availability--Disaster redundancy across cities and IDCS



# PGXZ @WeChat Payment

N***odes:***

***2 GTMs***（master-slave）

***7 Coordinators***

31 pairs of D***ata Nodes***

- ***17 pairs for small key group(hot)***
- ***8 pairs for huge key group(hot)***
- ***3 pairs for small key group(cold)***
- ***3 pairs for huge key group(cold)***

# Next step job

- Upgrade PGXZ(PG 9.3.5) to 9.6.X(Or may rebase on PGXL)
- Try to contribute Vacuum full concurrently

# My team


Thank
you!