PGPool-II & pg_shard

萧少聪 scott.siu@postgres.cn

DTCC

2015中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2015

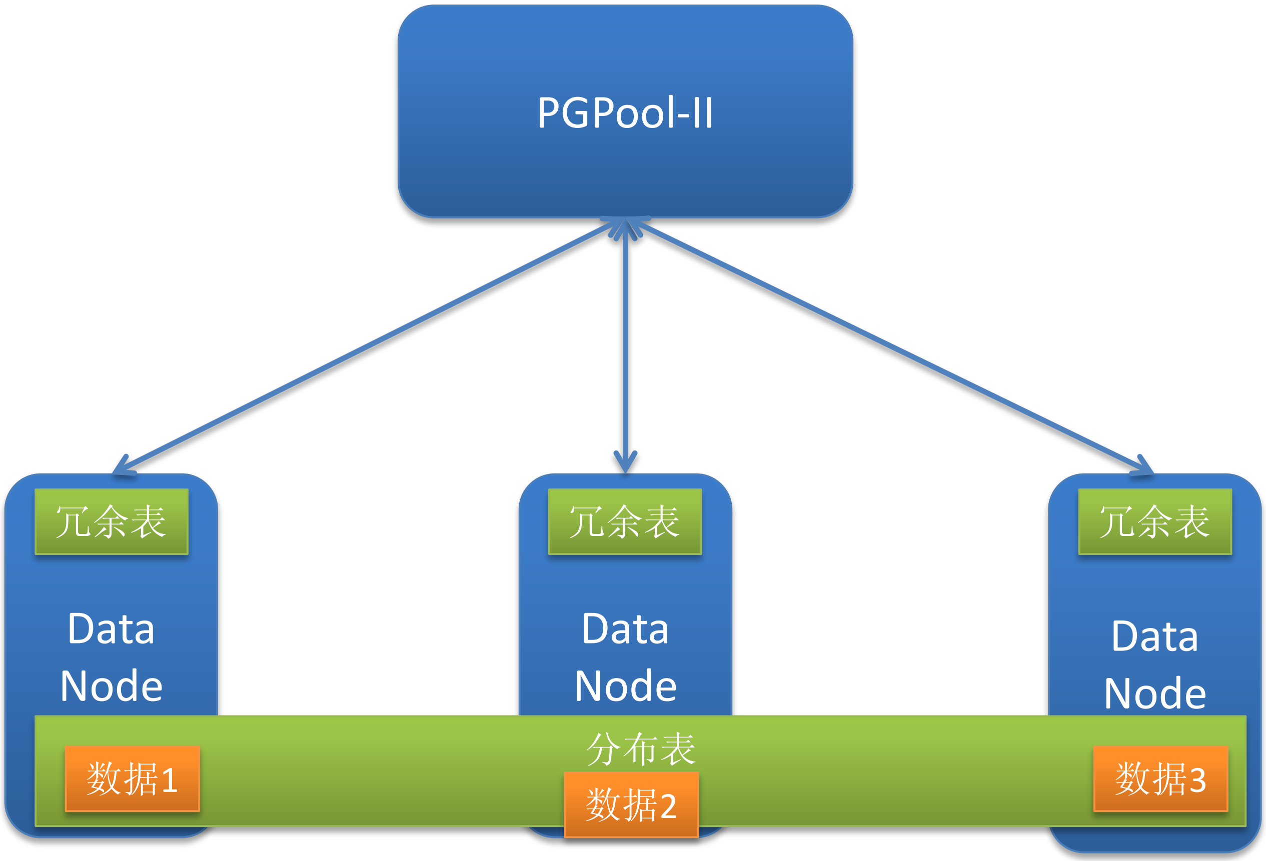
大数据技术探索和价值发现

IT168.com ChinaUnix ITPUB

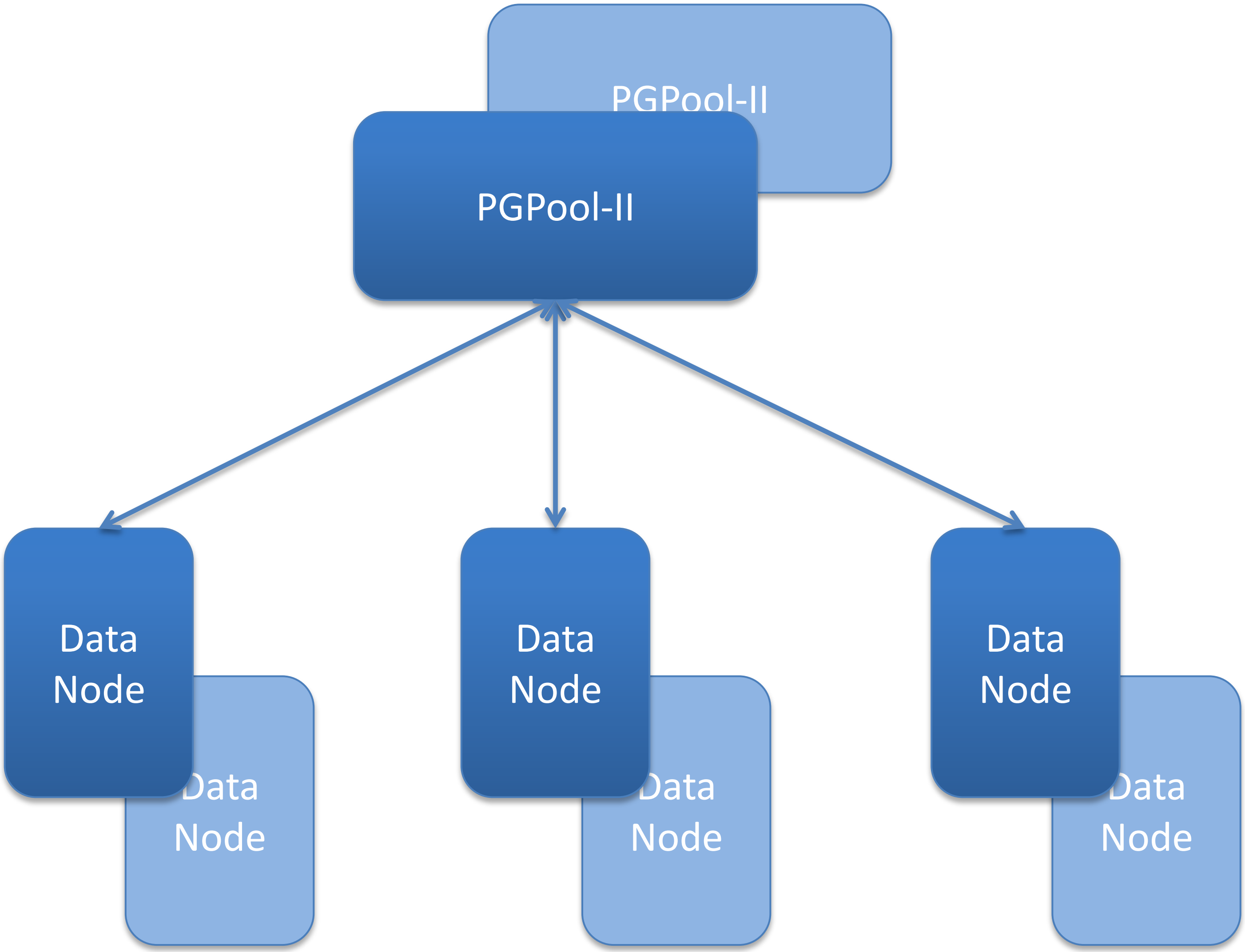
PGPool-II 主要的三种模式

- 连接池
- 水平分库
- 查询负载均衡

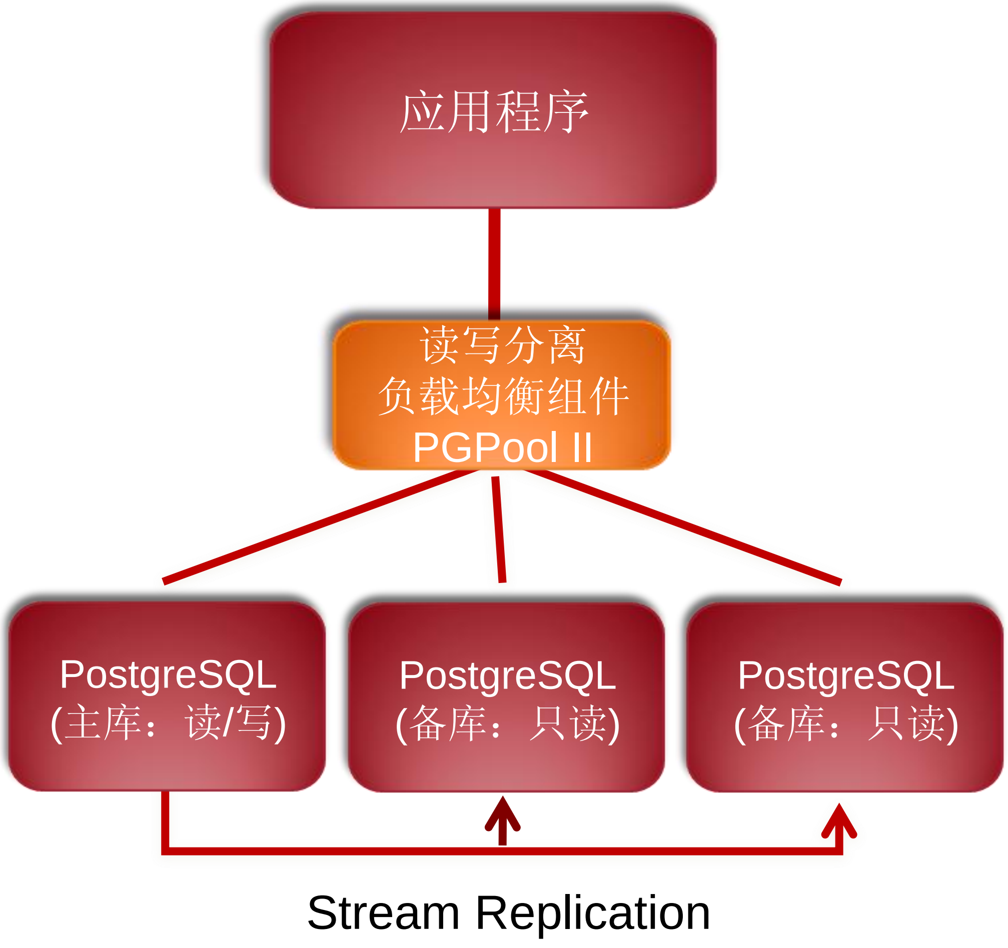
PGPool-II 水平分库



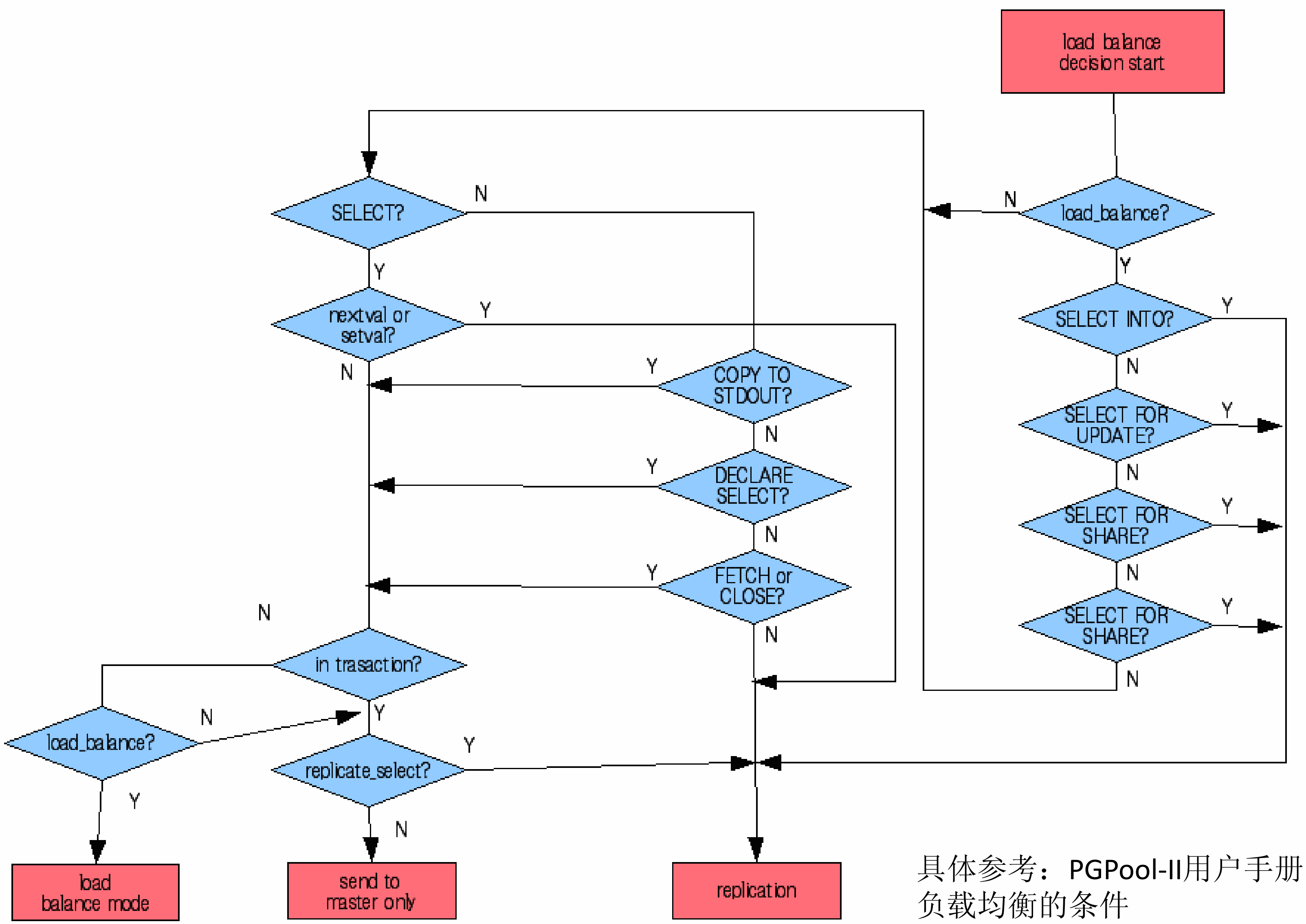
PGPool-II 水平分库



PGPool-II 查询负载均衡

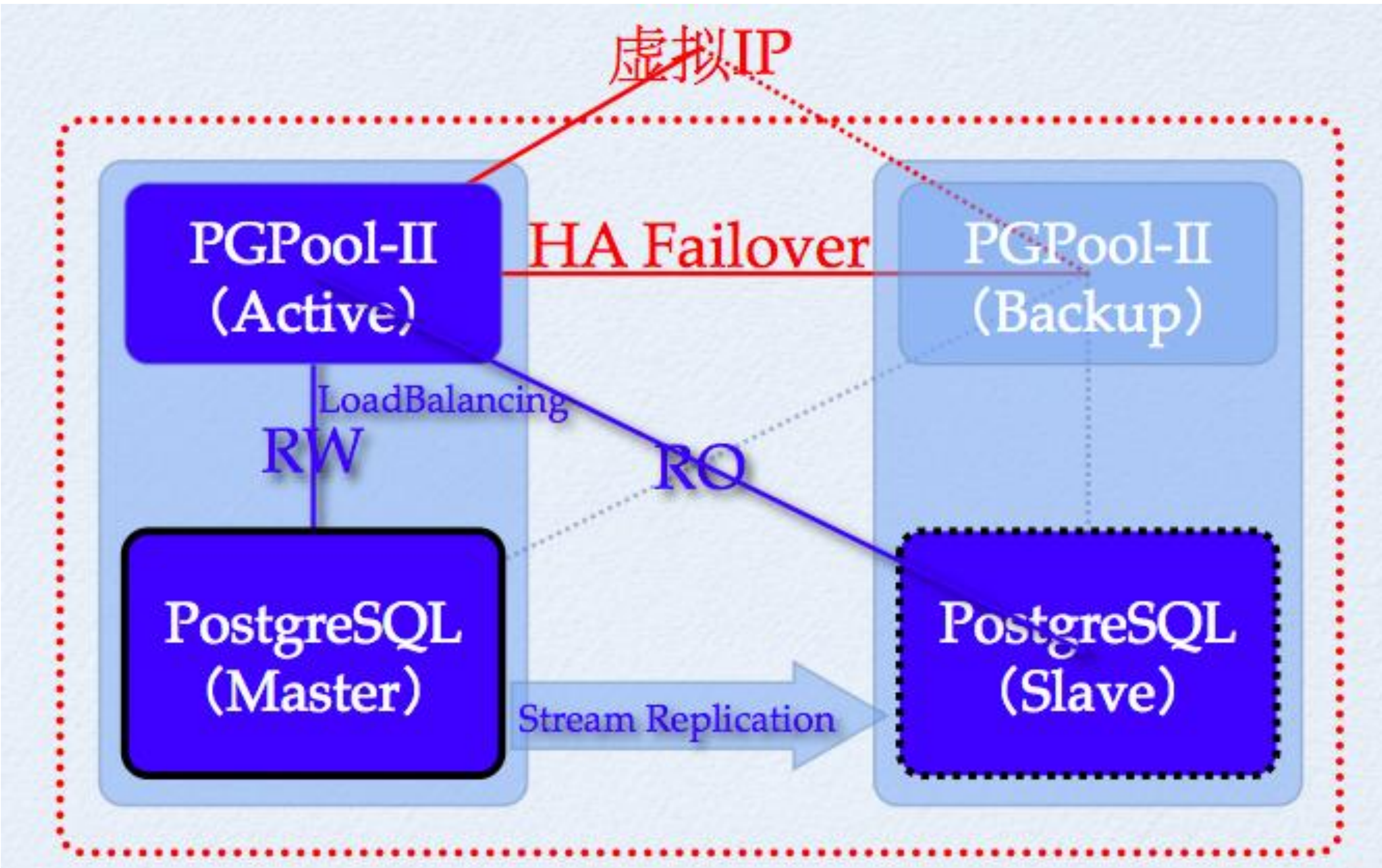


PGPool-II 查询负载均衡

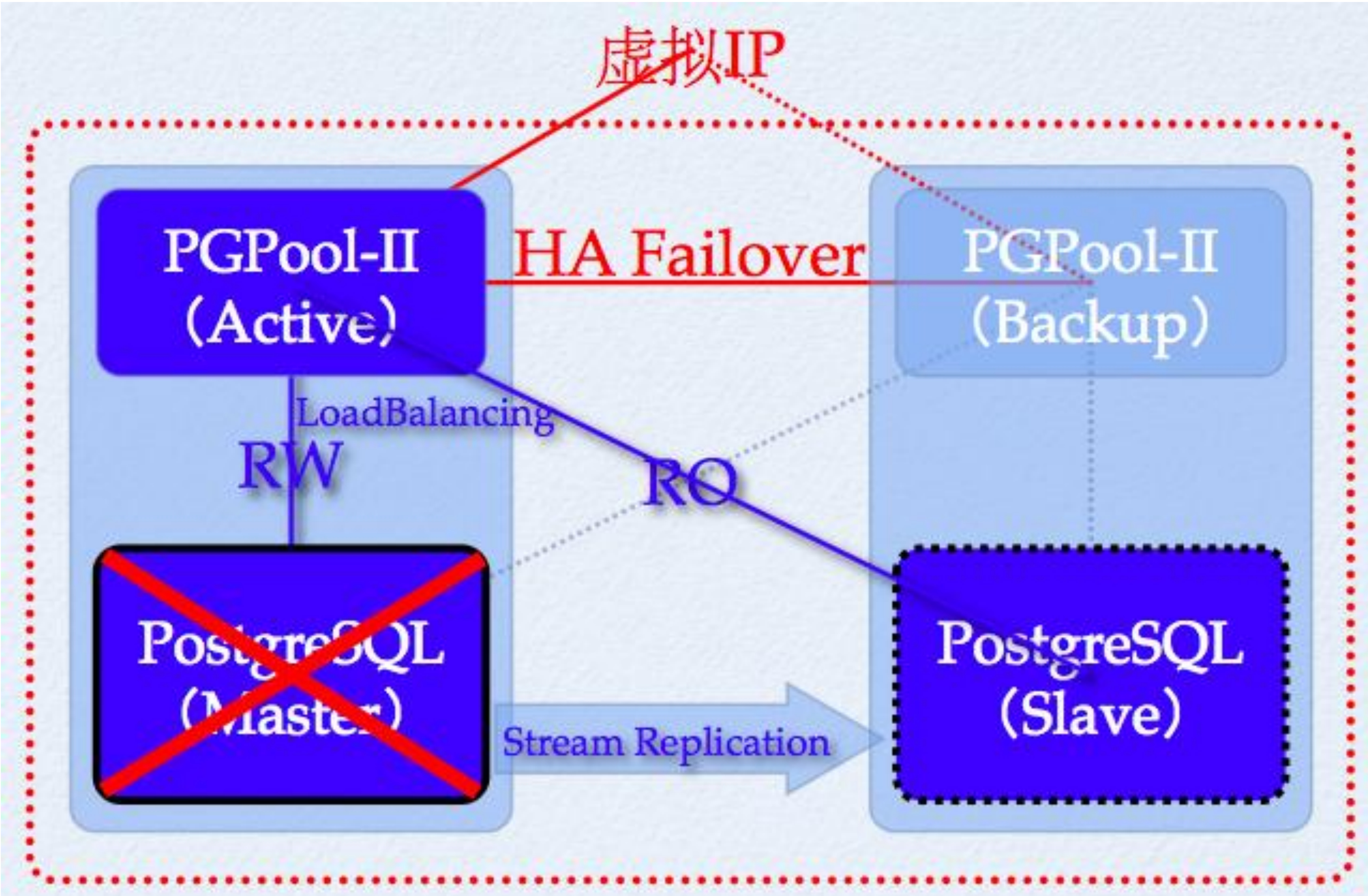


具体参考：PGPool-II用户手册中的负载均衡的条件

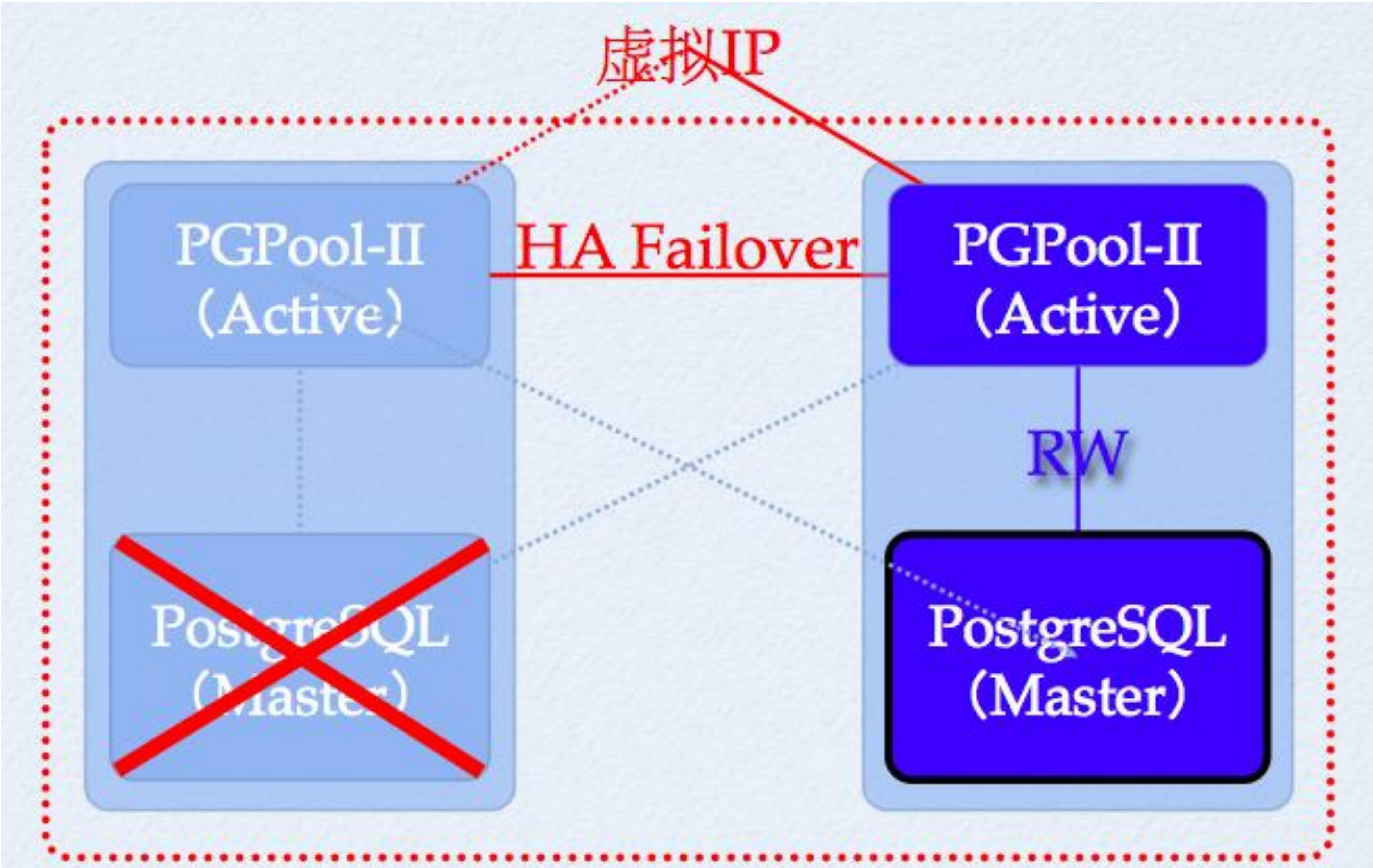
我的PGPool演变过程 – 双节点



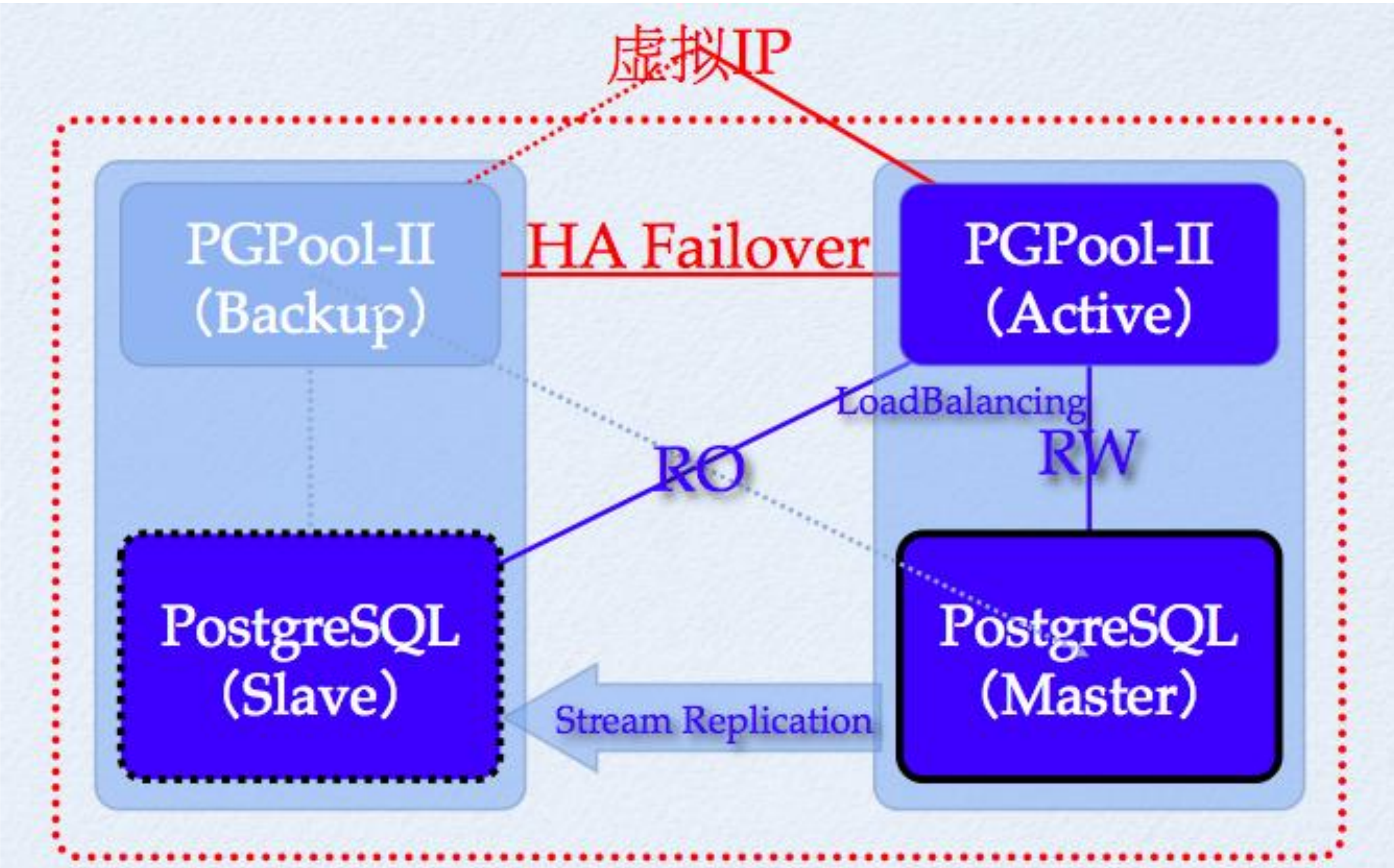
我的PGPool演变过程 – 双节点



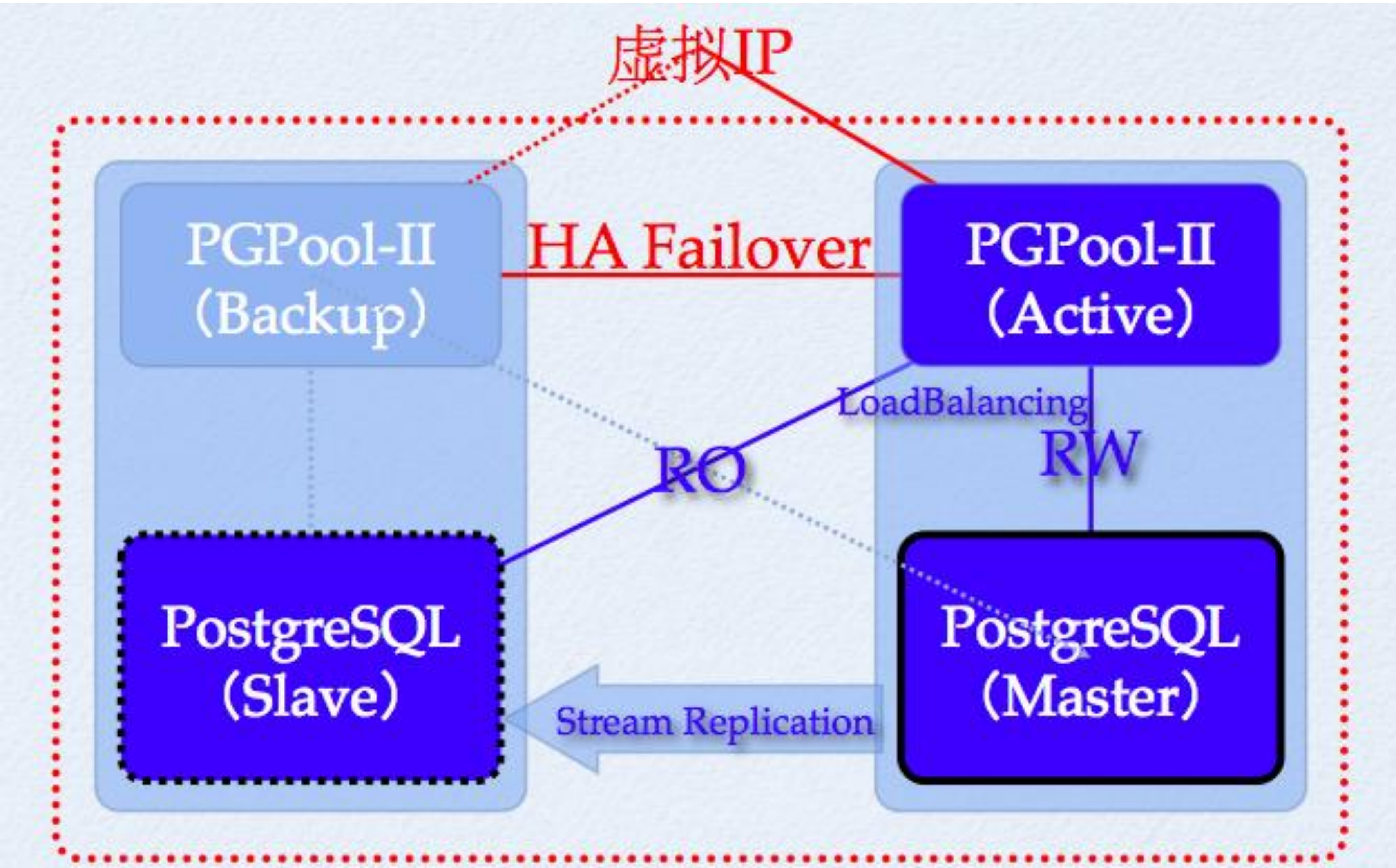
我的PGPool演变过程 – 双节点



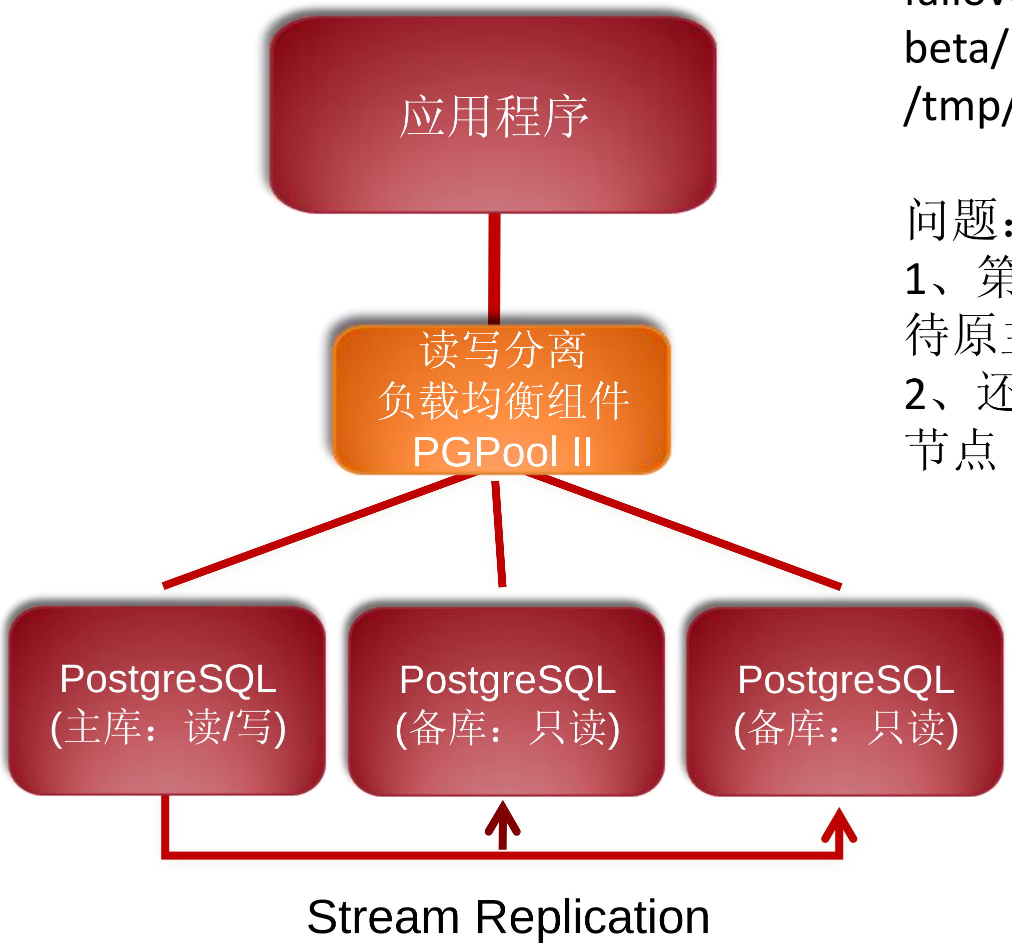
我的PGPool演变过程 – 双节点



我的PGPool演变过程 – 双节点



我的PGPool演变过程 – PGPool自动Failover



failover_command = '/usr/local/src/pgsql/9.0-beta/bin/failover_stream.sh %d %H /tmp/trigger_file0'

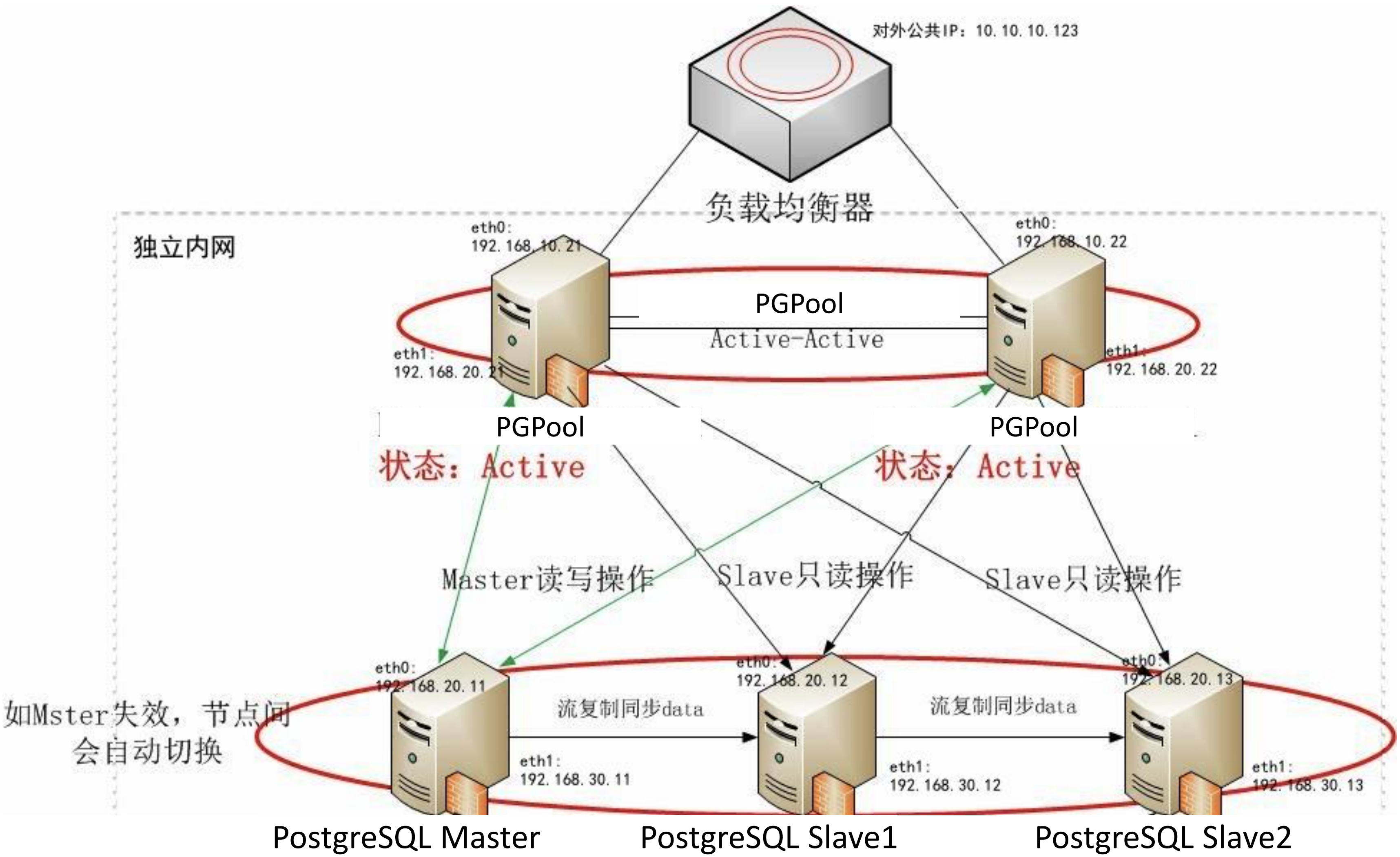
问题：

1、第1个备库promote后，第2个备库依然等待原主库的信息

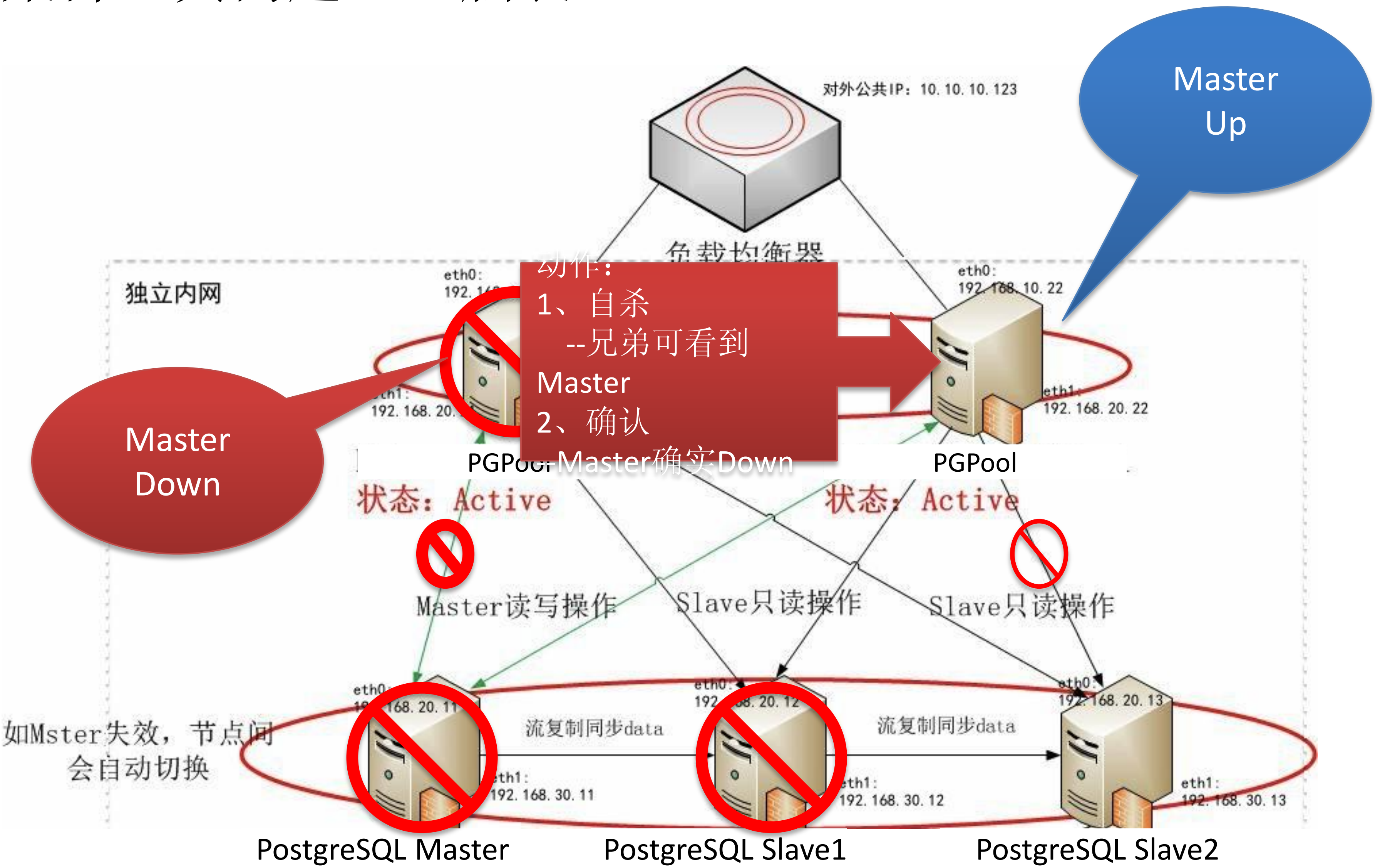
2、还要浪费1台服务器作为PGPool-II的备用节点

具体参考：PGPool-II用户手册中的流复制下的故障切换

2 PGPool +
3 PostgreSQL Node



集群经典问题：“脑裂”



Master
Up
对外公共IP：10. 10. 10. 123
独立内网
Master
Down
1、自杀
--兄弟可看到
Master
2、确认
Master确实Down
eth0:
192. 168. 10. 22
eth1:
192. 168. 20. 22
PGPool
PGPool
状态：Active
状态：Active
Master读写操作
Slave只读操作
Slave只读操作
流复制同步data
流复制同步data
eth1:
192. 168. 30. 11
eth1:
192. 168. 30. 12
eth1:
192. 168. 30. 13
192. 168. 20. 13
如Mster失效，节点间会自动切换
PostgreSQL Master
PostgreSQL Slave1
PostgreSQL Slave2

2 PGPool + 3 PostgreSQL Node
性能表现

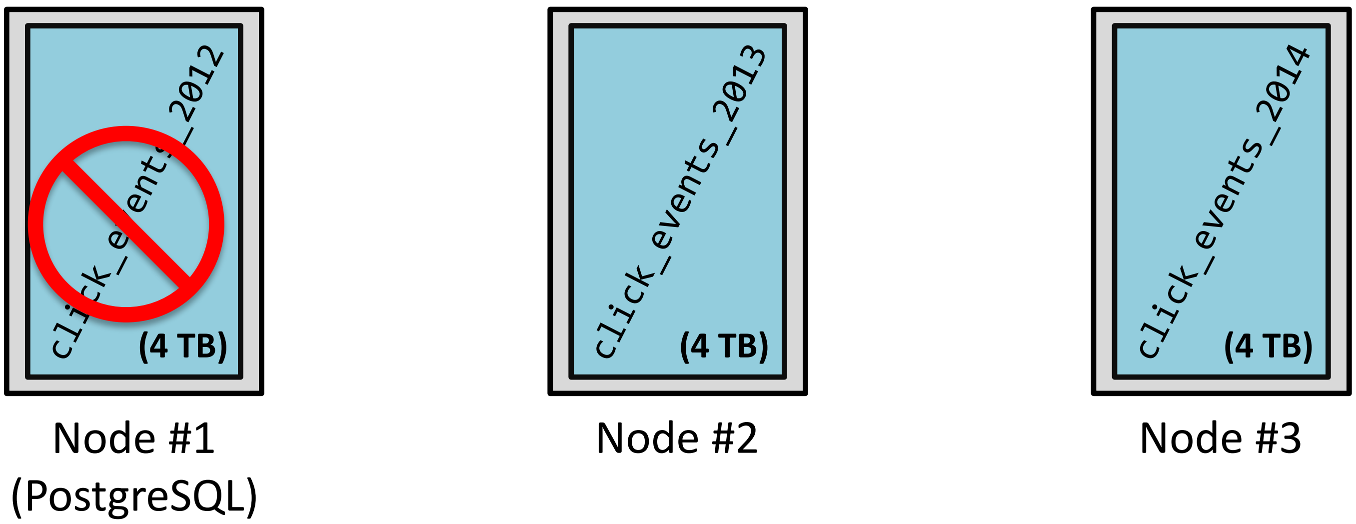
使用pgbench设计数据量级分别为：1.6GB、16GB、22GB、75GB（75GB大于当前数据库服务器的64GB内存容量）

		1.6GB	16GB	22GB	75GB
只读	Master	7658.746290	6261.945201	5686.477821	1578.743890
读写	Master	1397.658002	1405.554289	1324.288750	1153.064096

		1.6GB	16GB	22GB	75GB
只读	PGPool1	6742.551840	6112.995046	5993.060036	1403.986160
	PGPool2	5208.373120	5174.249760	5215.216332	1445.030742
读写	PGPool1	1134.719497	1137.917411	1087.843114	1064.235234
	PGPool2	1009.877069	1076.546164	943.548856	1078.342543

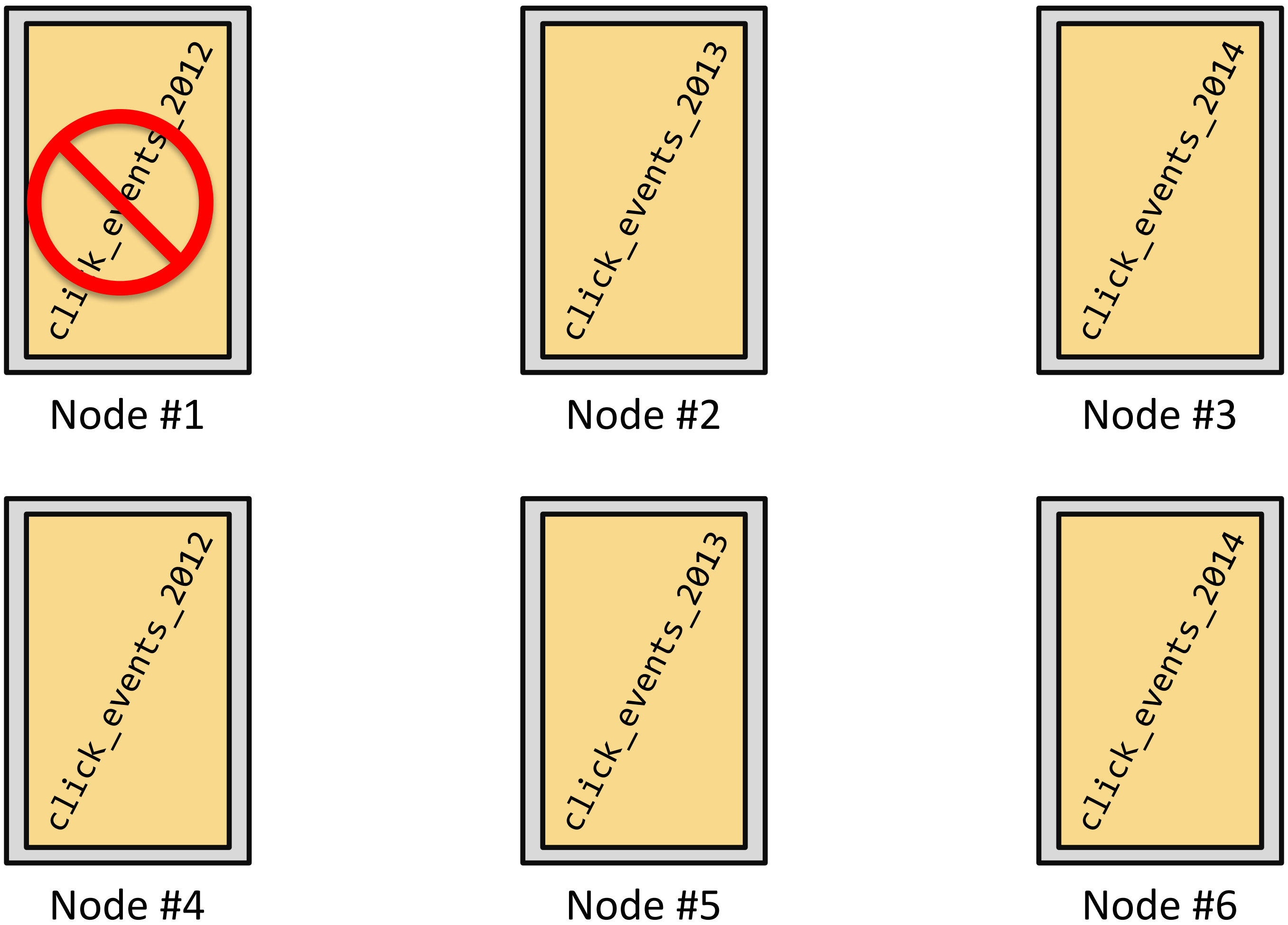
pg_shard

传统的水平分库模式



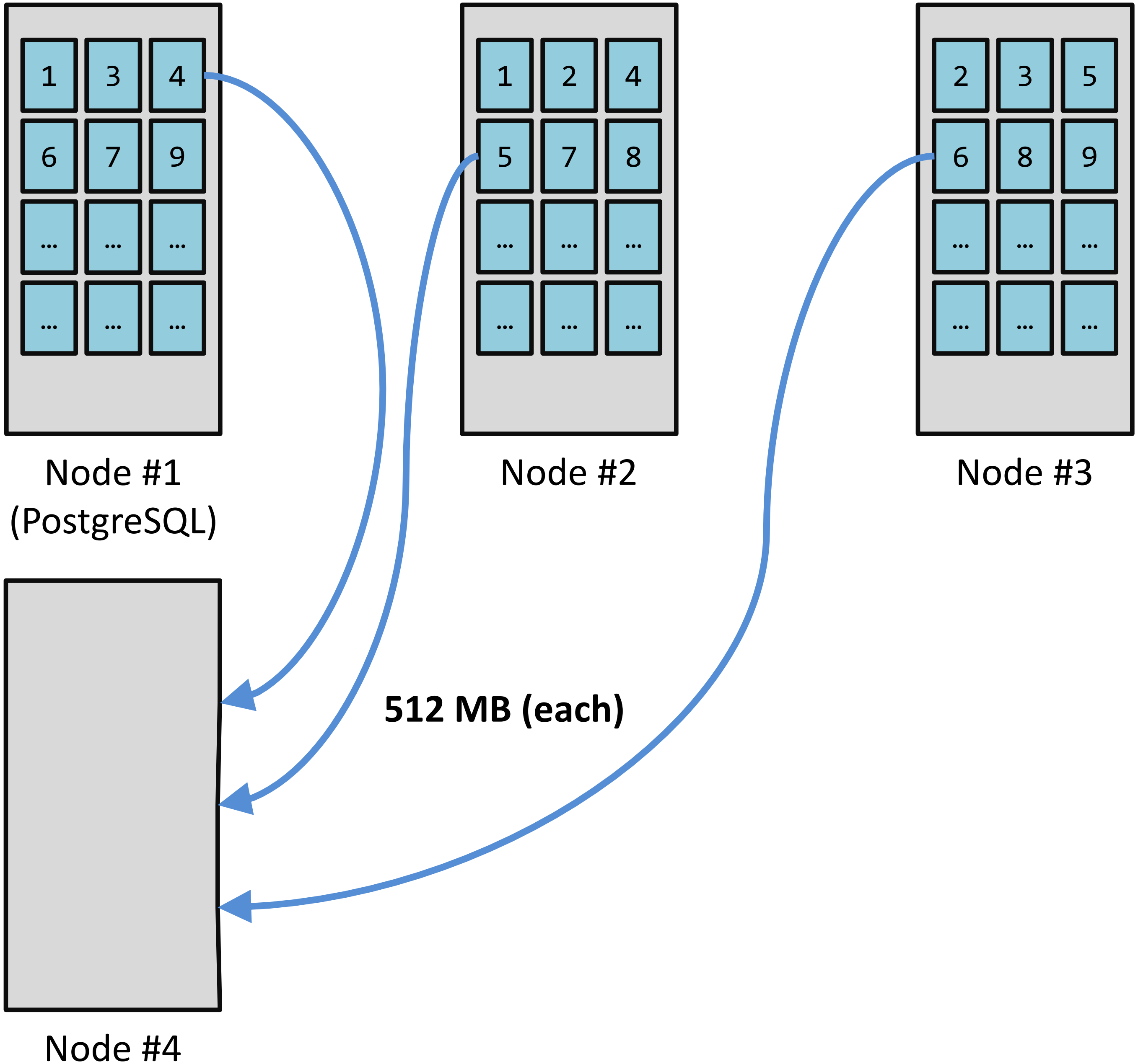
click_events_2012
(4 TB)
click_events_2013
(4 TB)
click_events_2014
(4 TB)
Node #1
(PostgreSQL)
Node #2
Node #3

传统的水平分库模式

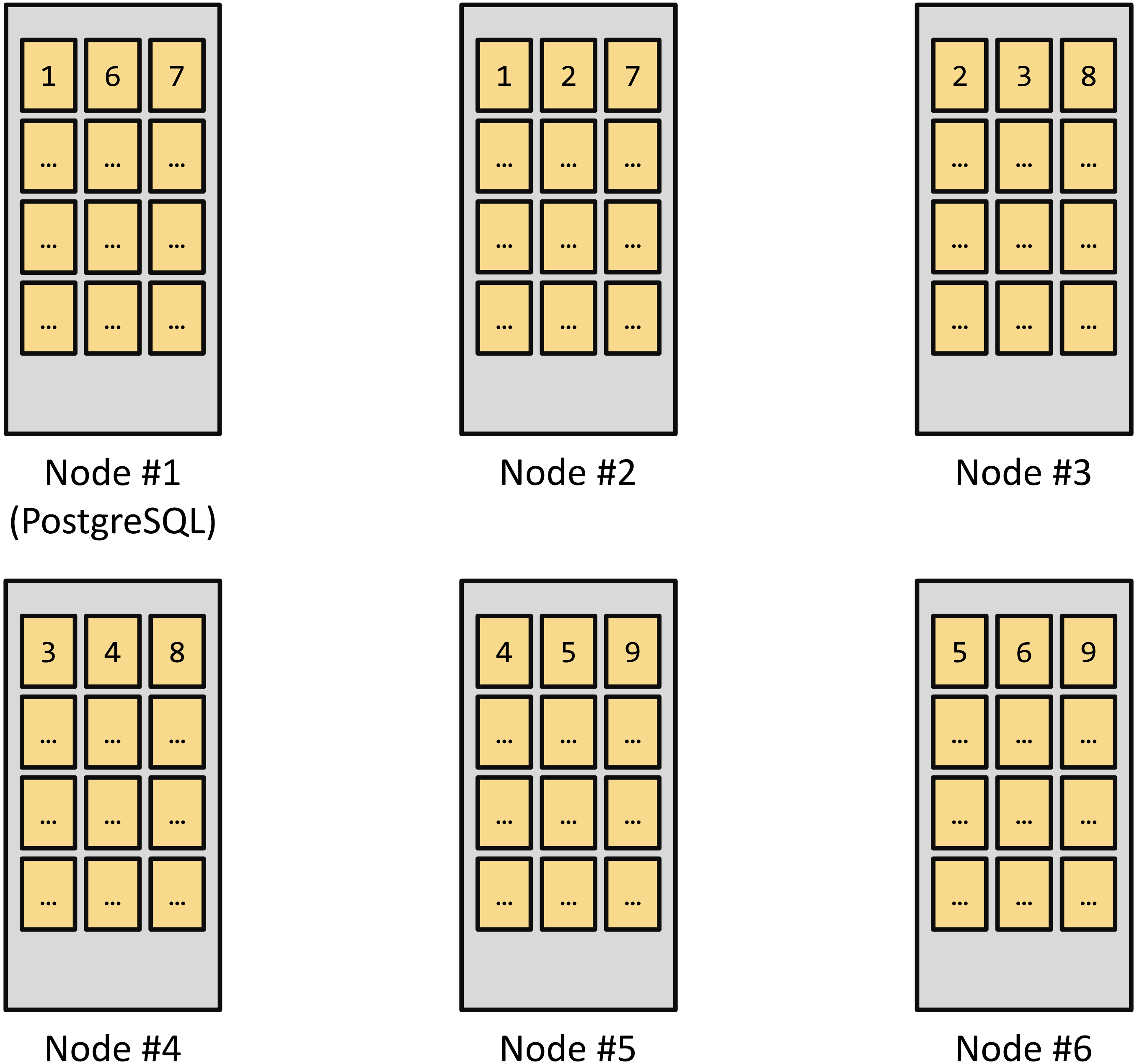

click_events_2012
Node #1
click_events_2013
Node #2
click_events_2014
Node #3
click_events_2012
Node #4
click_events_2013
Node #5
click_events_2014
Node #6

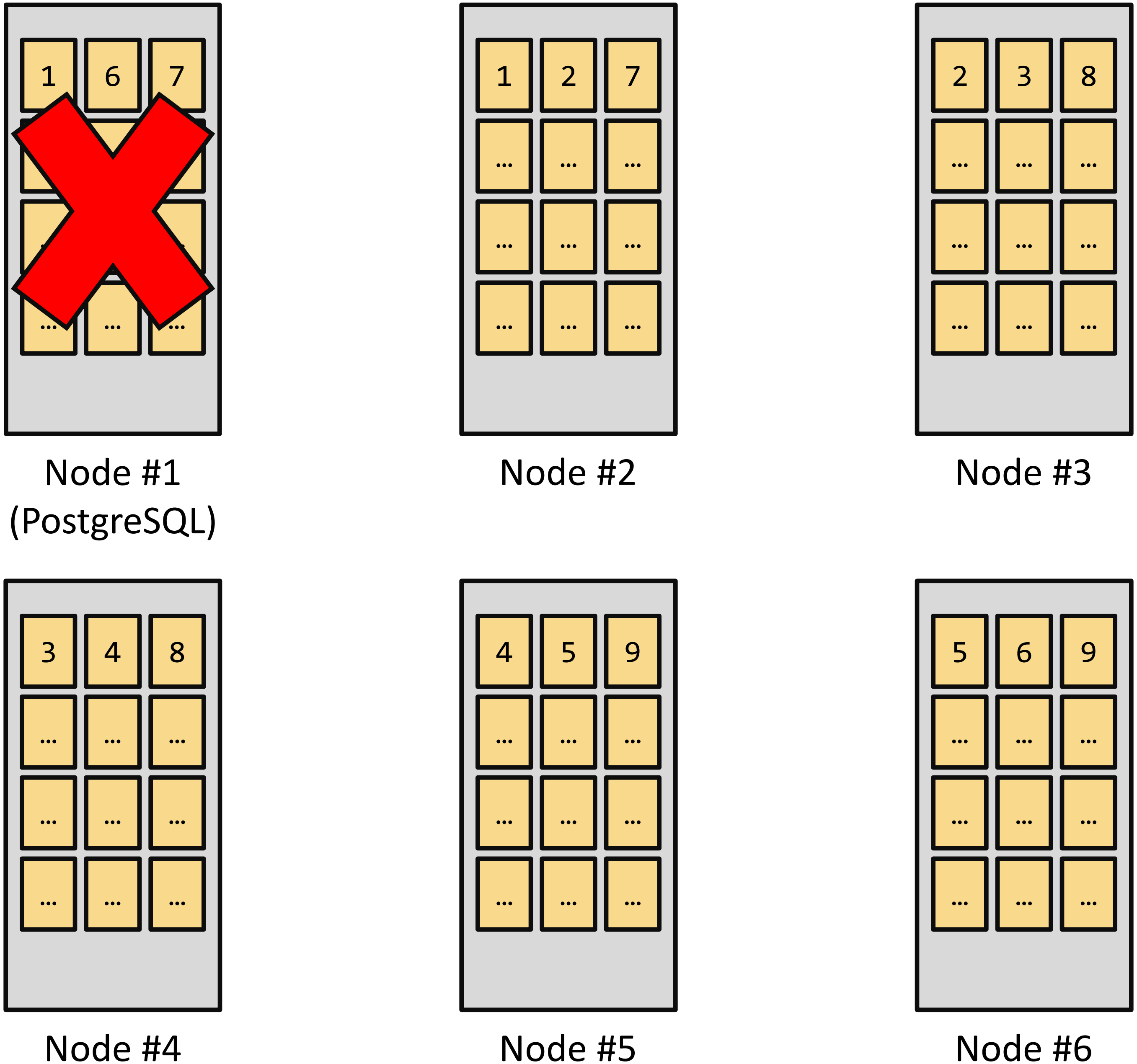
pg_shard

like Hadoop



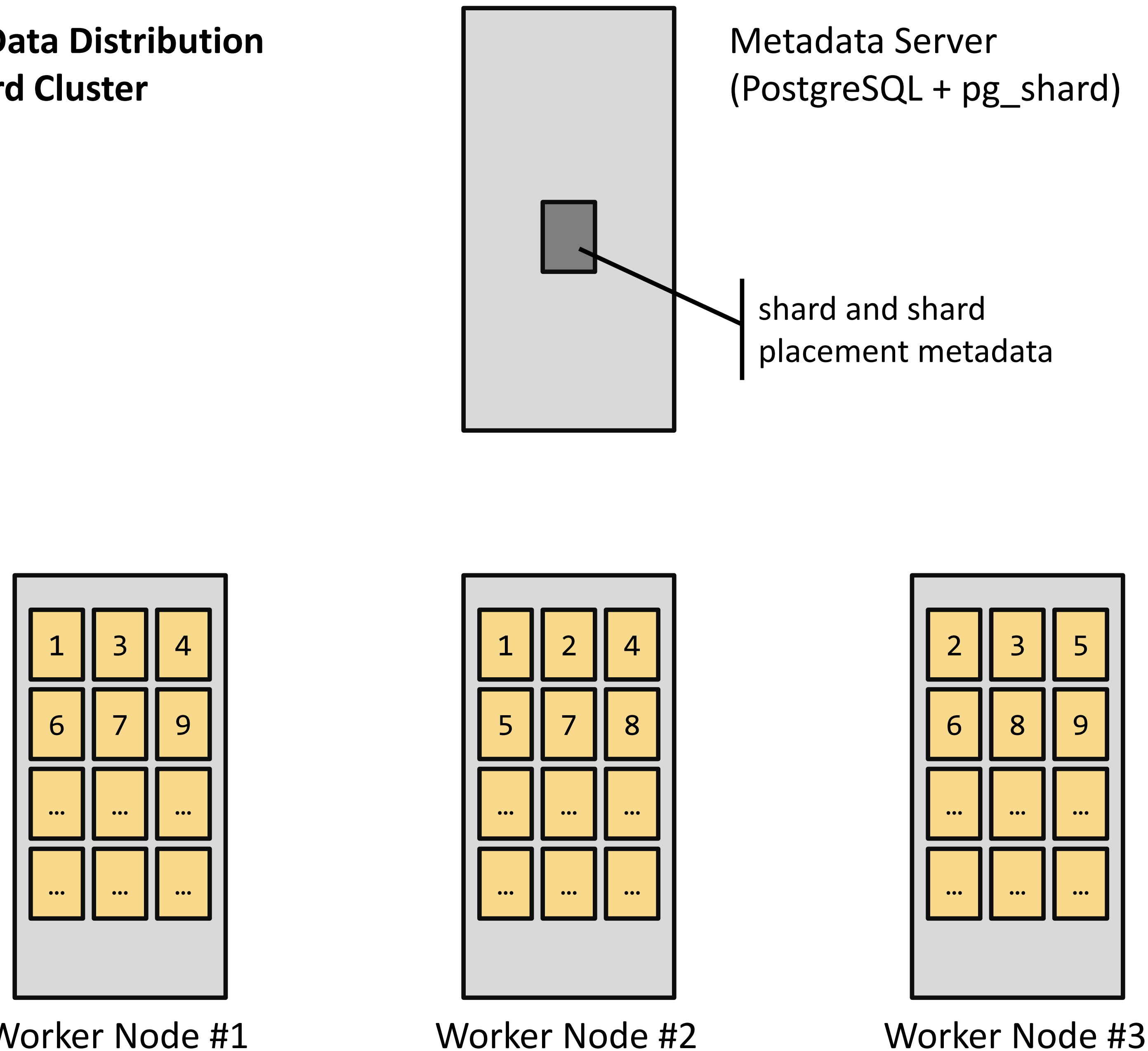
1
3
4
6
7
9
...
Node #1
(PostgreSQL)
1
2
4
5
7
8
...
Node #2
2
3
5
6
8
9
...
Node #3
512 MB (each)
Node #4


1 6 7
Node #1
(PostgreSQL)
1 2 7
Node #2
2 3 8
Node #3
3 4 8
Node #4
4 5 9
Node #5
5 6 9
Node #6



1
6
7
Node #1
(PostgreSQL)
1
2
7
Node #2
2
3
8
Node #3
3
4
8
Node #4
4
5
9
Node #5
5
6
9
Node #6

Example Data Distribution in pg_shard Cluster



Users: Making Scaling SQL Easy

```
CREATE EXTENSION pg_shard;   -- create a regular
PostgreSQL table:  CREATE TABLE customer_reviews
(customer_id TEXT NOT NULL,
                                review_date DATE, ...);
  -- distribute the table on the given partition key:
 SELECT
master_create_distributed_table('customer_reviews',
                                        'customer_id');
```

Metadata and Hash Partitioning

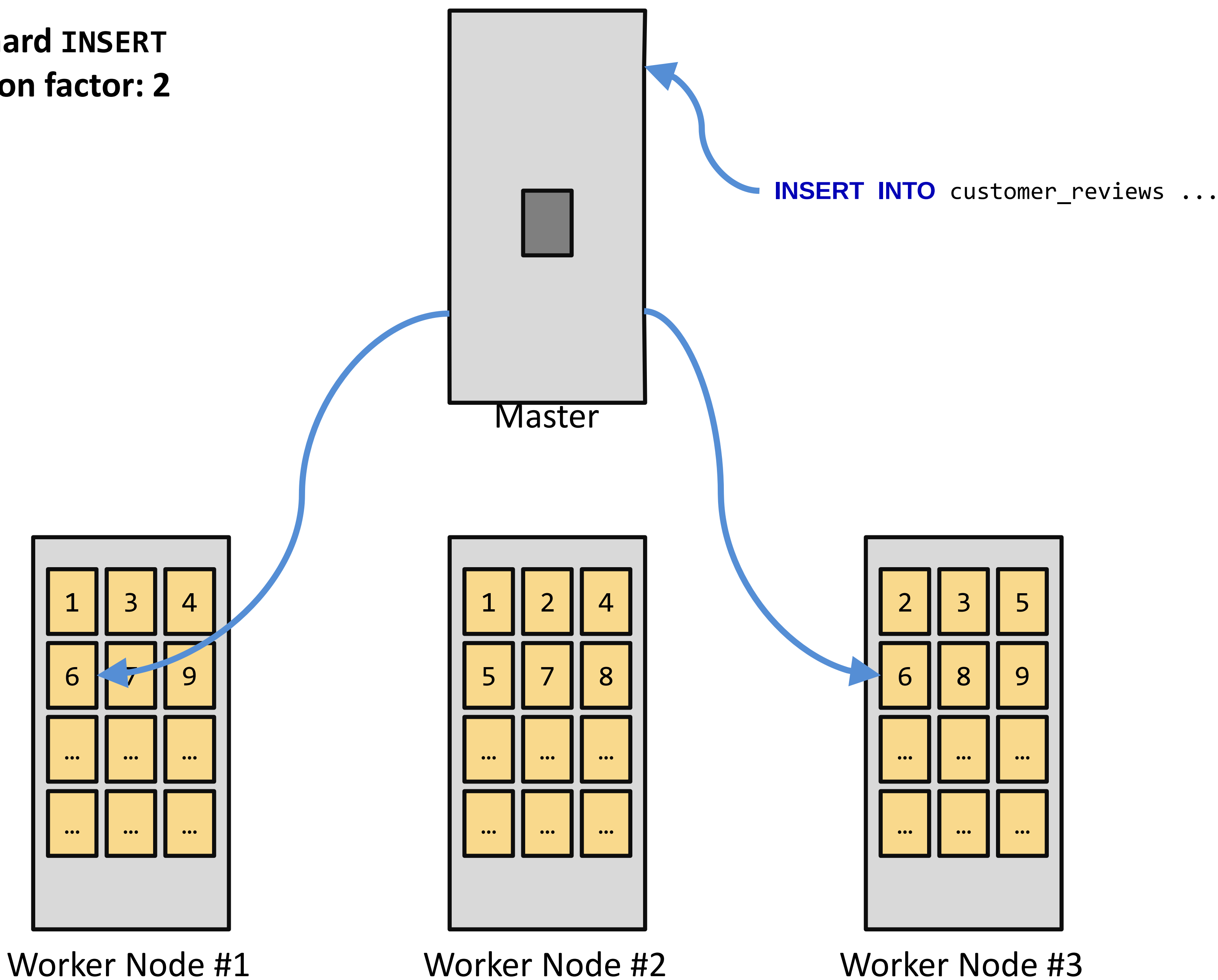
```
postgres=# SELECT * FROM pgs_distribution_metadata.shard;
  id   | relation_id | storage |  min_value   |  max_value
-------+-------------+---------+-------------+-------------
 10004 |      177880 | t       | -2147483648 | -1879048194
 10005 |      177880 | t       | -1879048193 | -1610612739
 10006 |      177880 | t       | -1610612738 | -1342177284
 10007 |      177880 | t       | -1342177283 | -1073741829
 10008 |      177880 | t       | -1073741828 | -805306374
 10009 |      177880 | t       | -805306373  | -536870919
   ... |         ... | ...     |  ...        |  ...
```

Planning Example

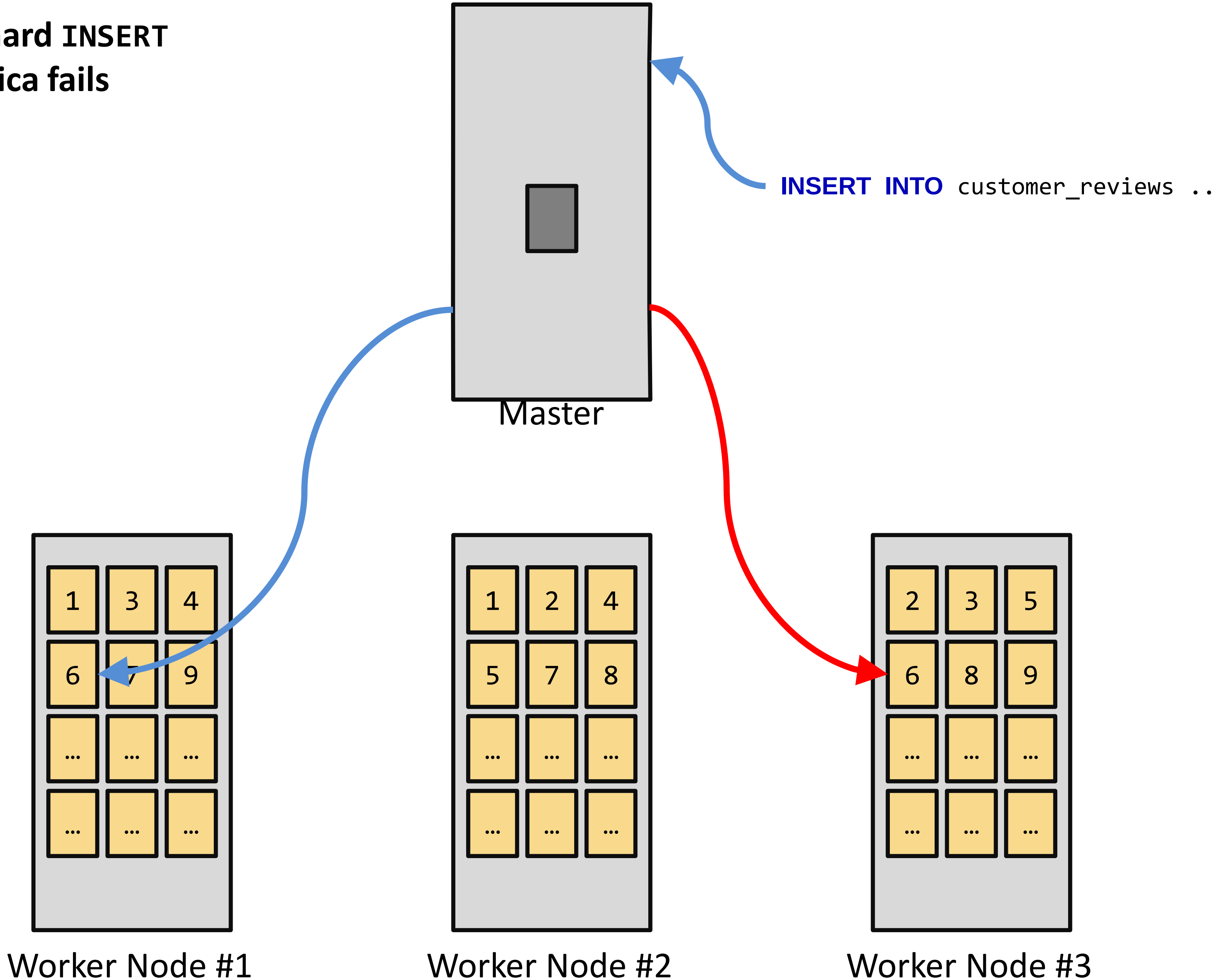
```
INSERT INTO customer_reviews (customer_id,
rating)
                                 VALUES ('HN892', 5);
```

1. Determine partition key clauses:
   `customer_id = 'HN892'`
2. Find shards from metadata tables:
   `hashtext('HN892') BETWEEN min_value AND max_value`
3. Produce shard-specific SQL
   ```
   INSERT INTO customer_reviews_16
     (customer_id, rating) VALUES ('HN892', 5);
   ```

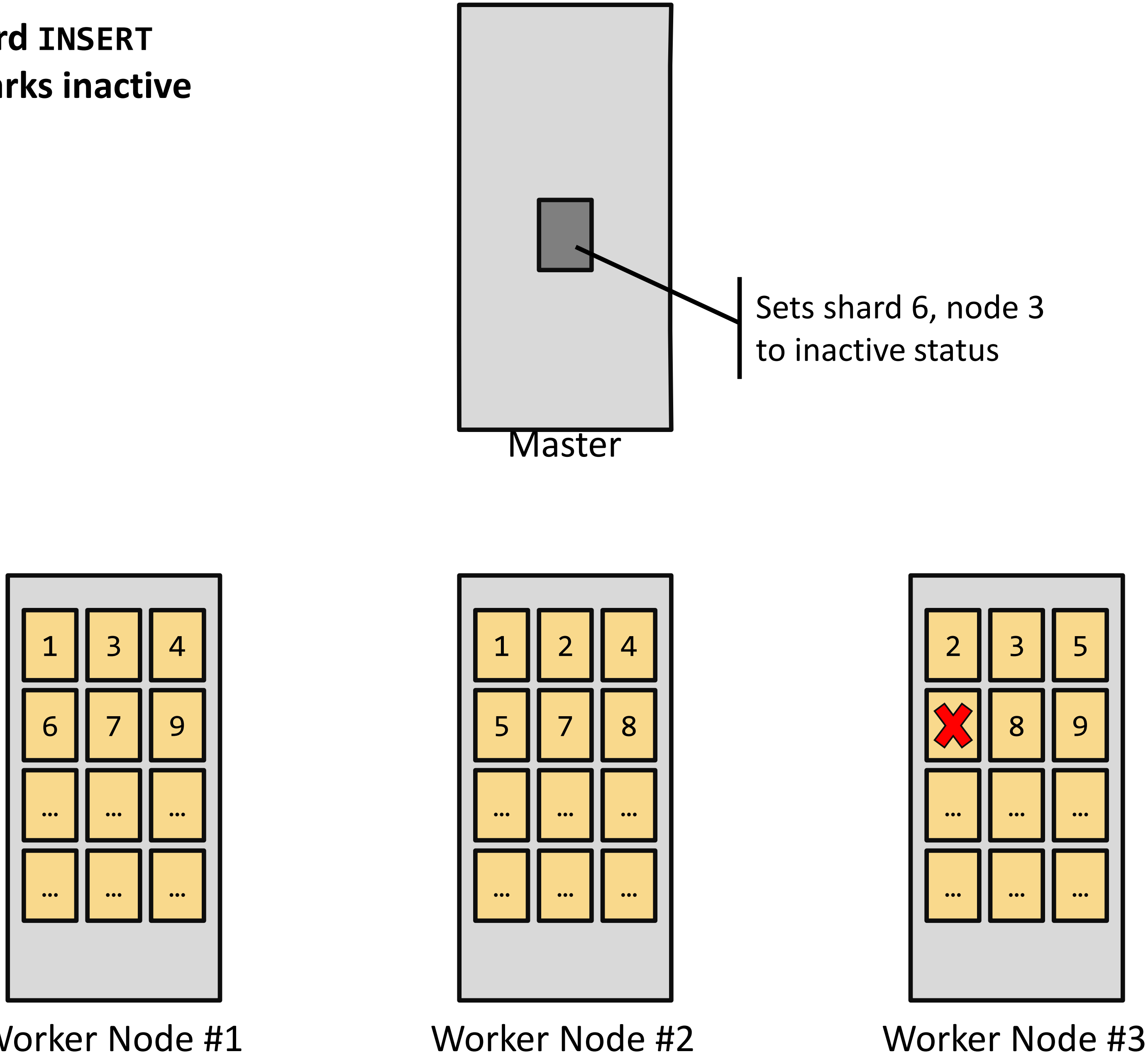
Single-shard `INSERT`
Replication factor: 2



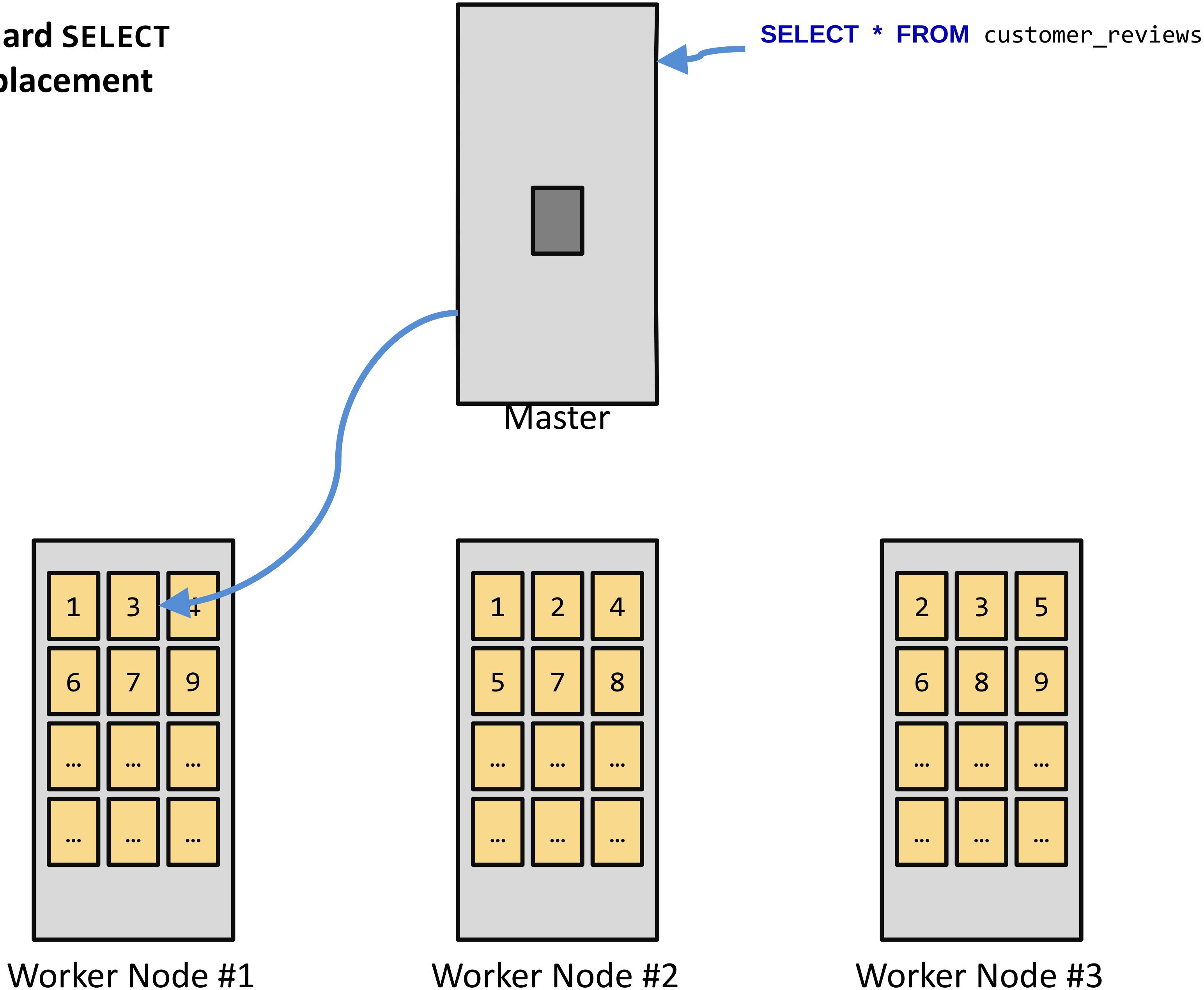
Single-shard INSERT
One replica fails



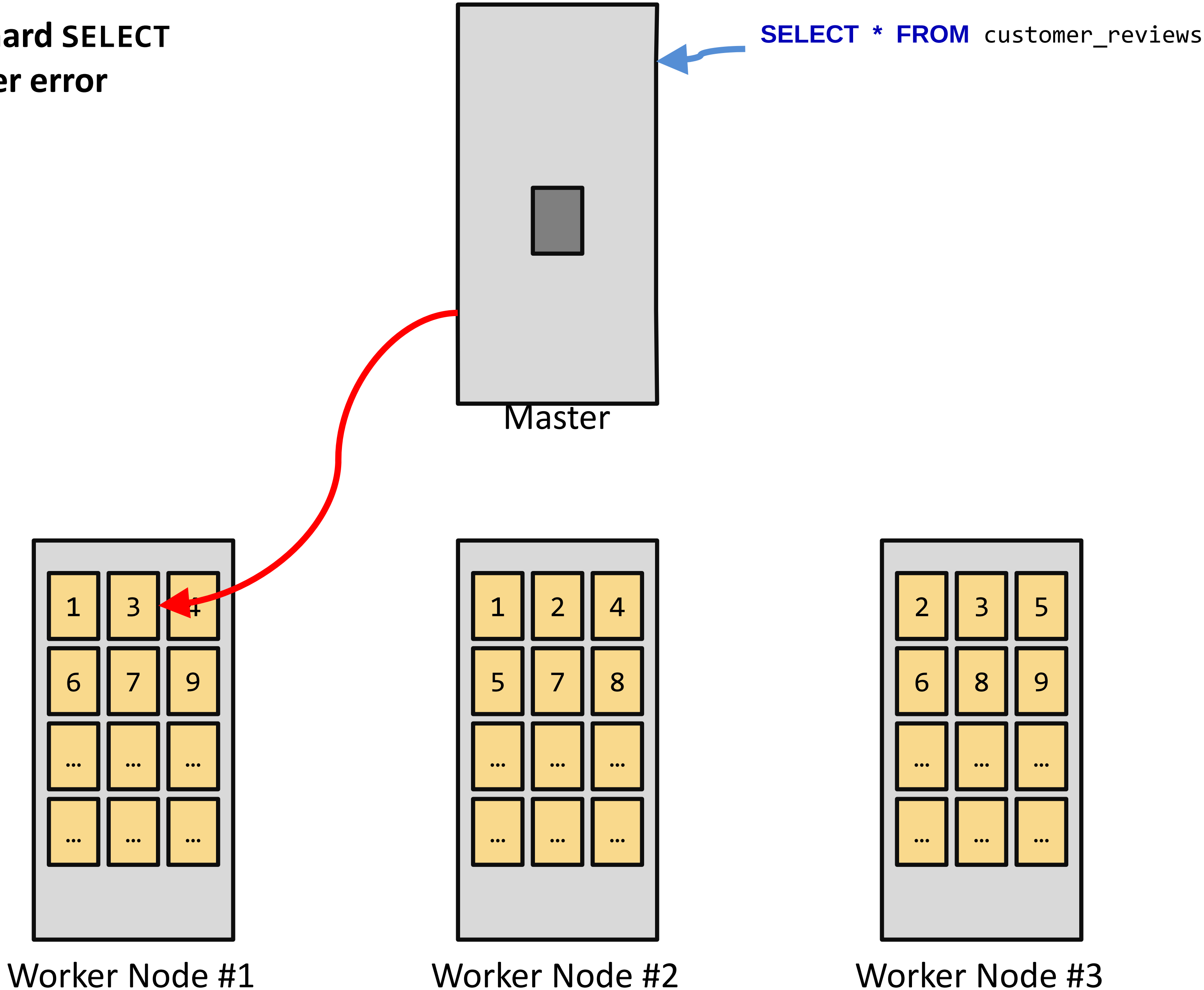
Single-shard `INSERT`
Master marks inactive



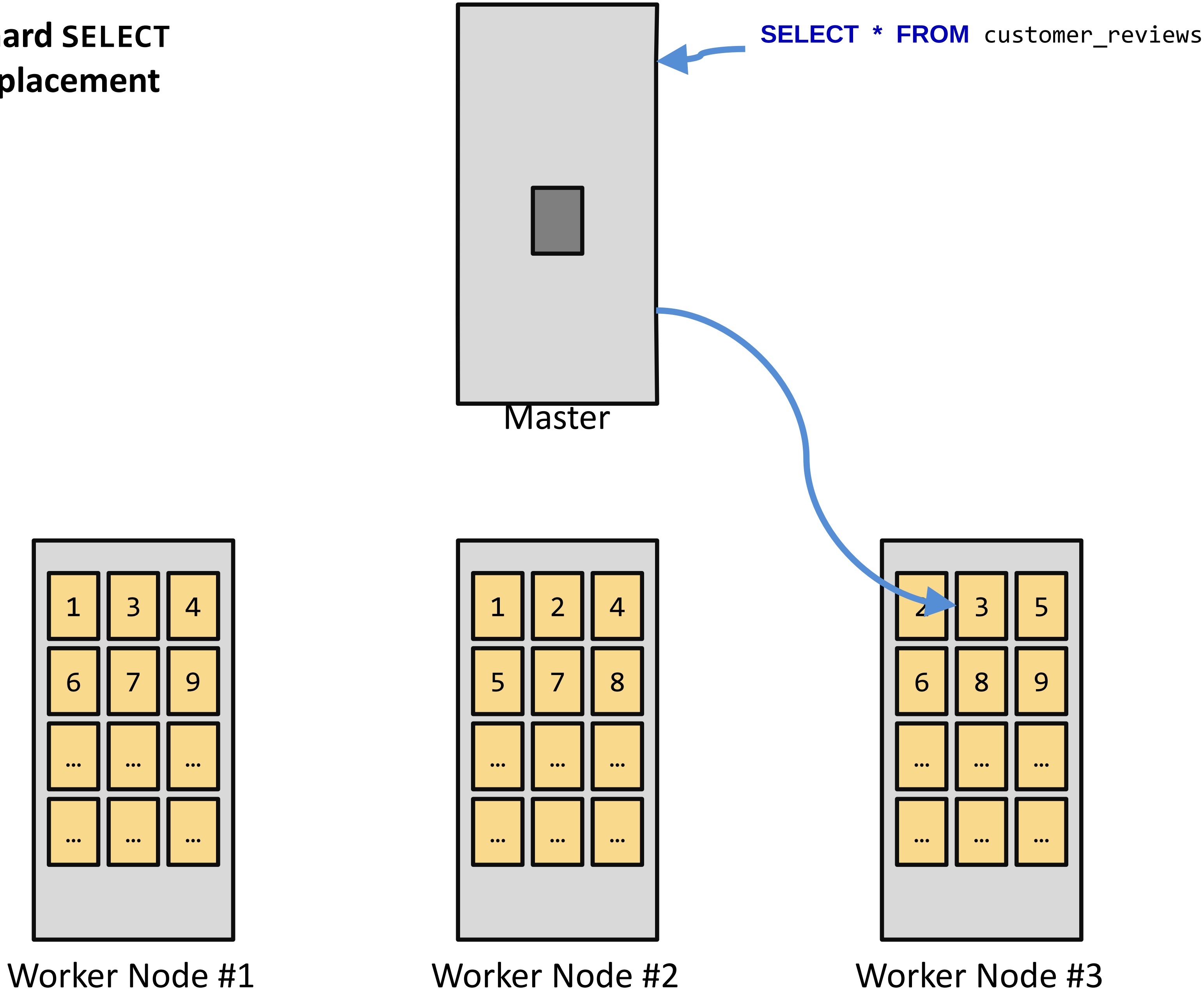
Single-shard `SELECT`
Try first placement



Single-shard `SELECT`
Encounter error



Single-shard `SELECT`
Try next placement



pg_shard性能及瓶颈

If your data set size is large and your workload is bottlenecked by disk, pg_shard will effectively scale out your workload to disks on the worker nodes. In this case, the master node is unlikely to be a scaling bottleneck.

If your workload is bottlenecked on CPU, then the master node could become a bottleneck. Our initial benchmarks show that the master node can process 5K inserts/sec per CPU core. We're currently looking at performance improvements on the master node to make query routing an even lighter operation.

> 64 shards and 256 concurrent processes, pg_shard 1.0 executed 14,532 inserts per second (ips) in our internal benchmark. In pg_shard 1.1, we had the throughput improve to 56,982 ips.

PS：少聪 没有钱搭环境，求“土豪”！！！

pg_shard
还有一些问题

- 无法支持事务完整性
- 不支持JSON操作
- 无法实现跨shard的约束
- 架构很新只有2年左右

- 试想一下，在一个RDBMS的数据库中，有一个表挂着数十台服务器还可以无限扩展（⊙o⊙）

- https://github.com/citusdata/pg_shard

分布式的“舍”与“得”

- 强一致性 ｜ 分布处理
- 存储空间 ｜ 数据冗余
- 响应速度 ｜ 整体效率
- 投入成本 ｜ 性能、可靠

THANKS

Postgres中国用户会

萧少聪 scott.siu@postgres.cn