携程 Redis 多数据中心双向同步实践

祝辰

祝辰

目前任职携程框架架构部门资深研发工程师

负责框架Redis团队的开发工作

目录

开篇 & 背景

Redis 在携程的规模

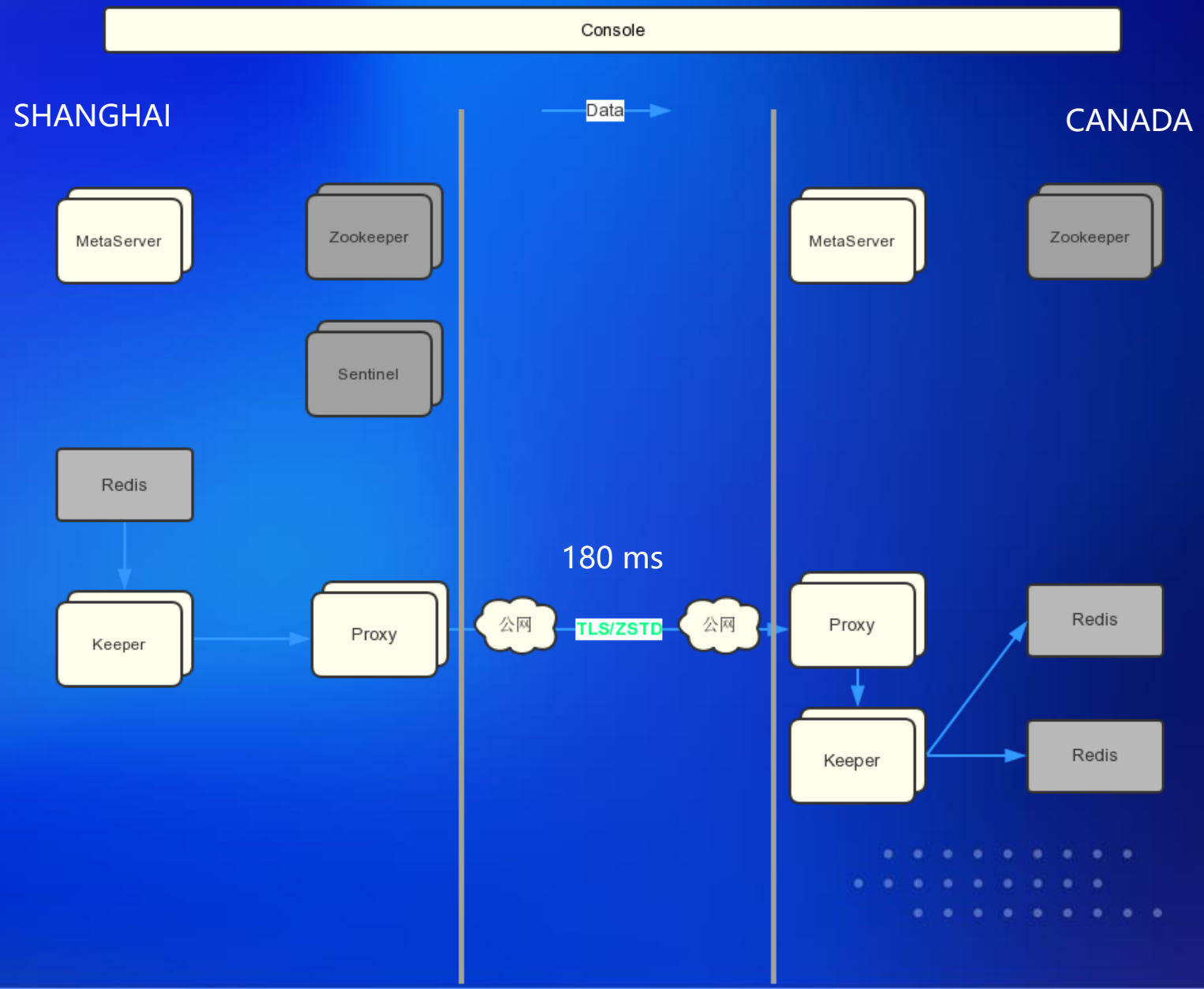
25,000,000

QPS

2000+Cluster

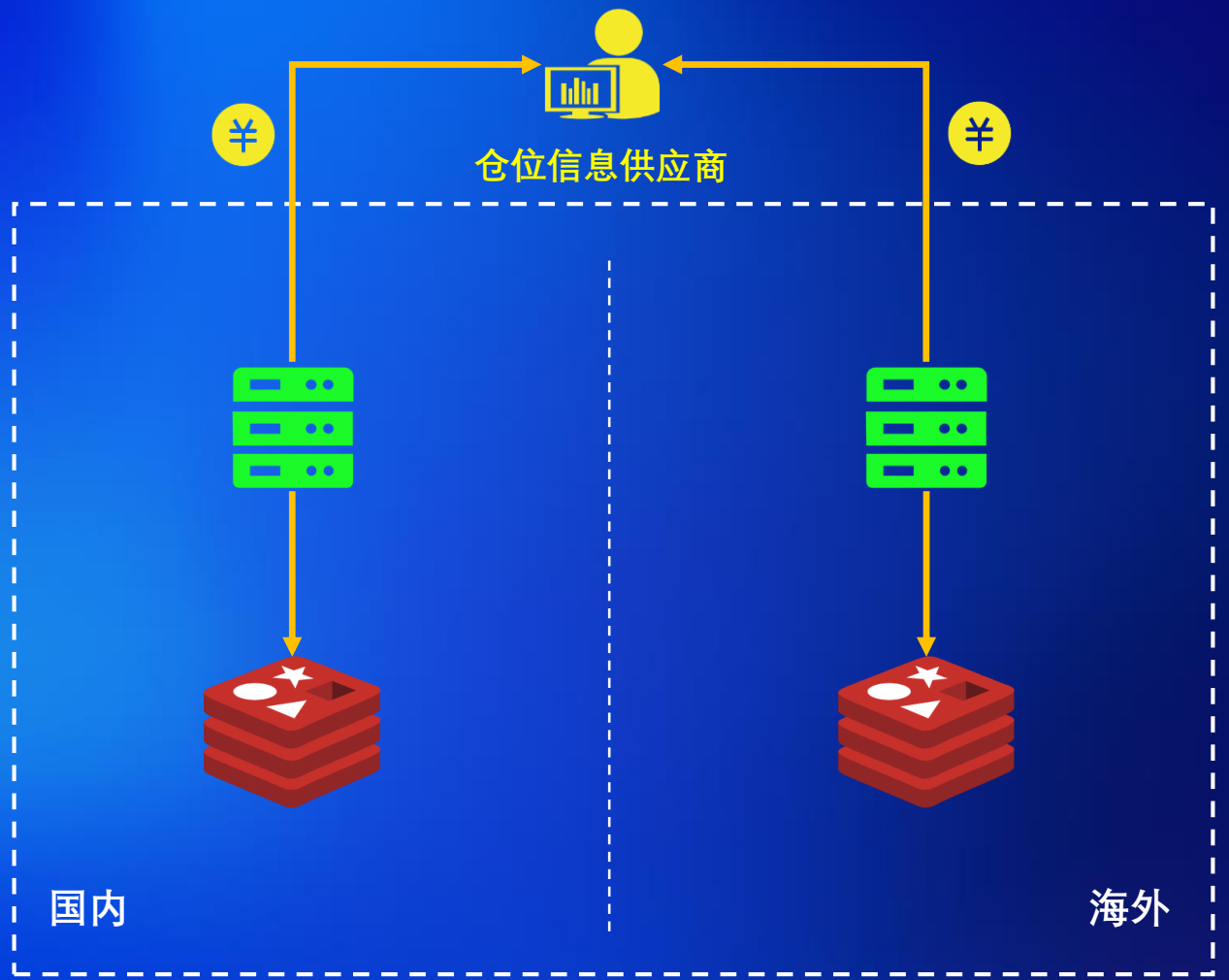
200TB+

跨公网同步


Console
SHANGHAI
Data
CANADA
MetaServer
Zookeeper
Sentinel
Redis
Keeper
Proxy
180 ms
公网
TLS/ZSTD
公网
MetaServer
Zookeeper
Proxy
Redis
Keeper
Redis

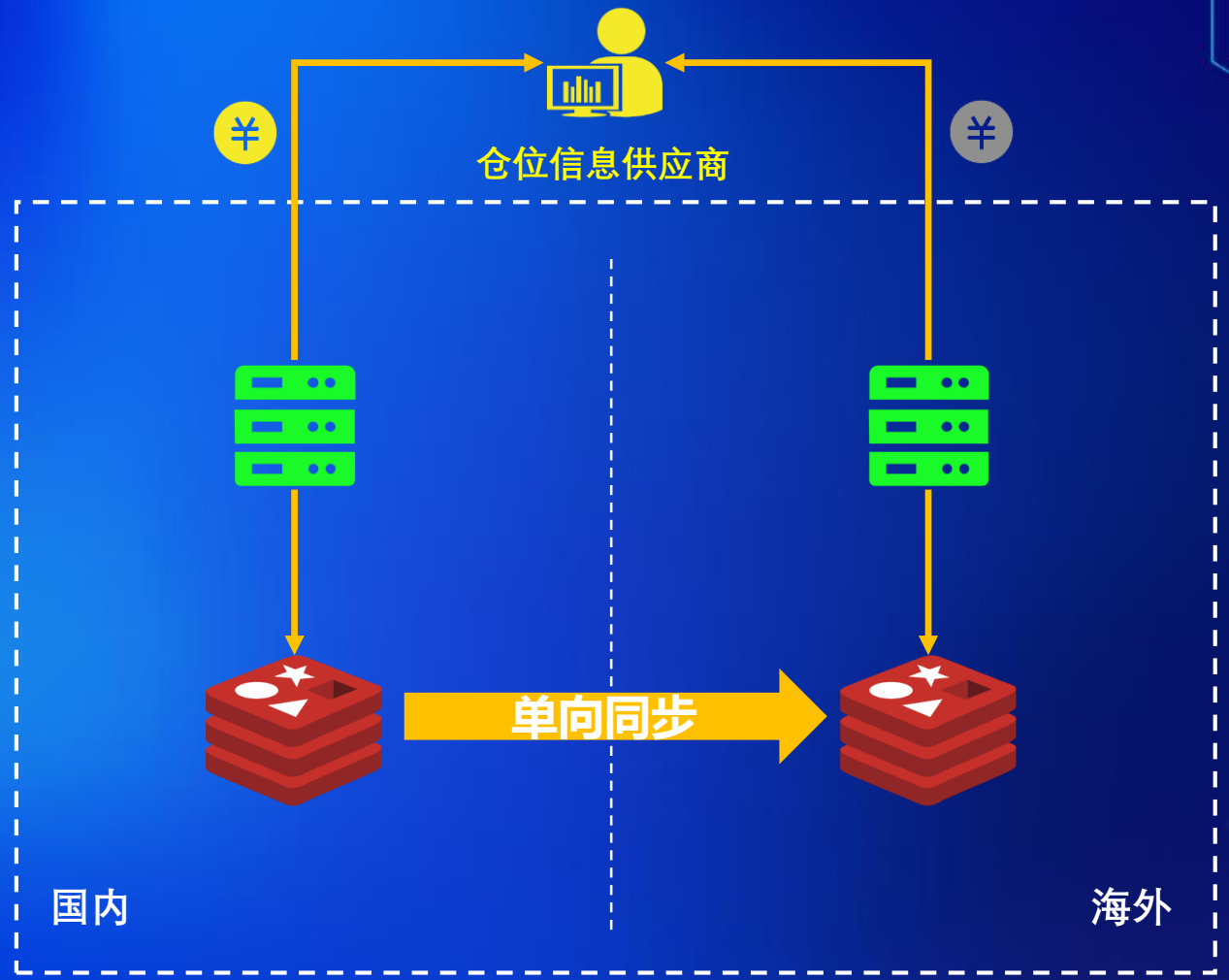
业务痛点

- 海外用户和国内用户查询同一份数据
- 需要向供应商付费2次



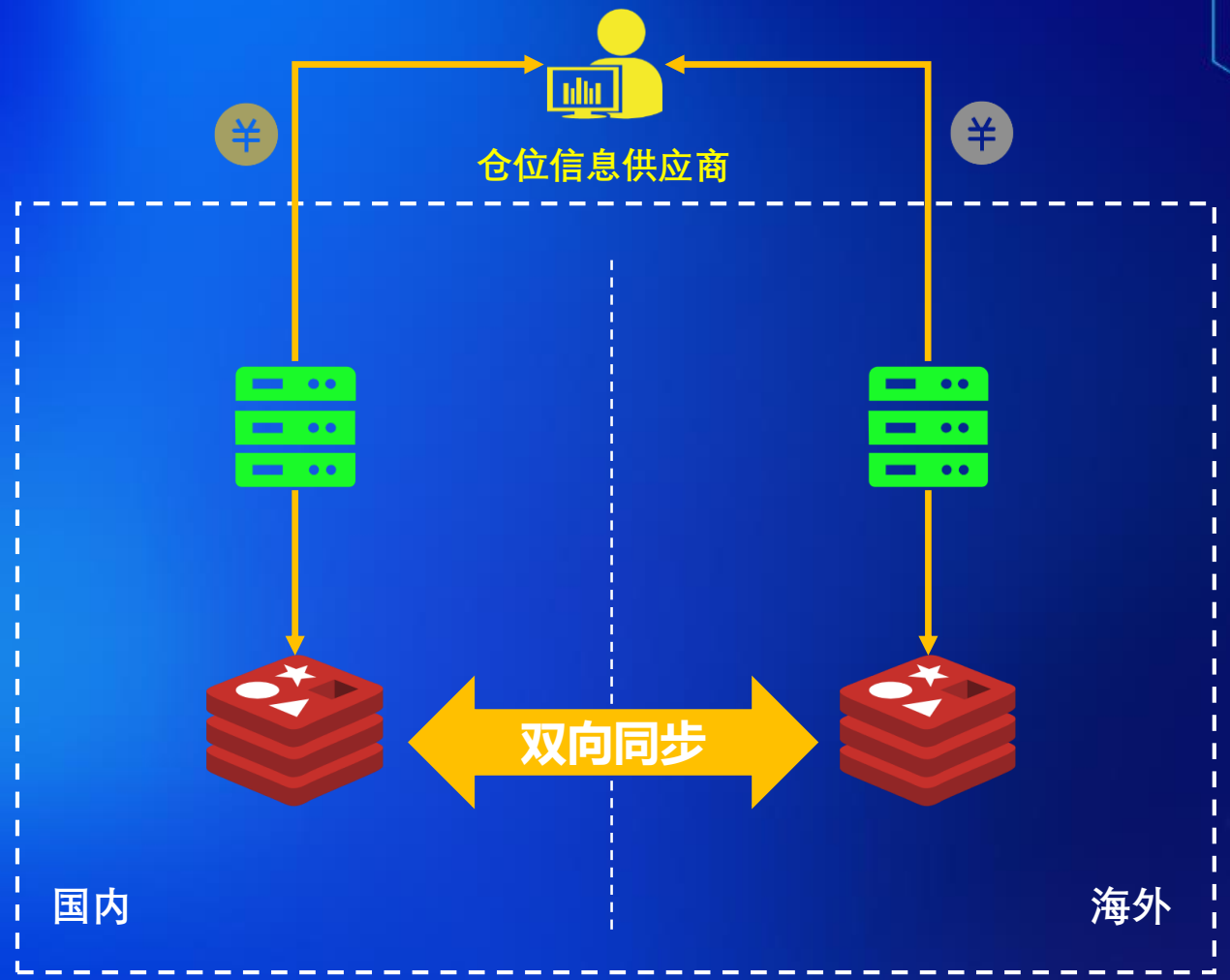
业务痛点

- 单向同步可以解决海外重复收费的问题
- 无法解决上海重复收费的问题



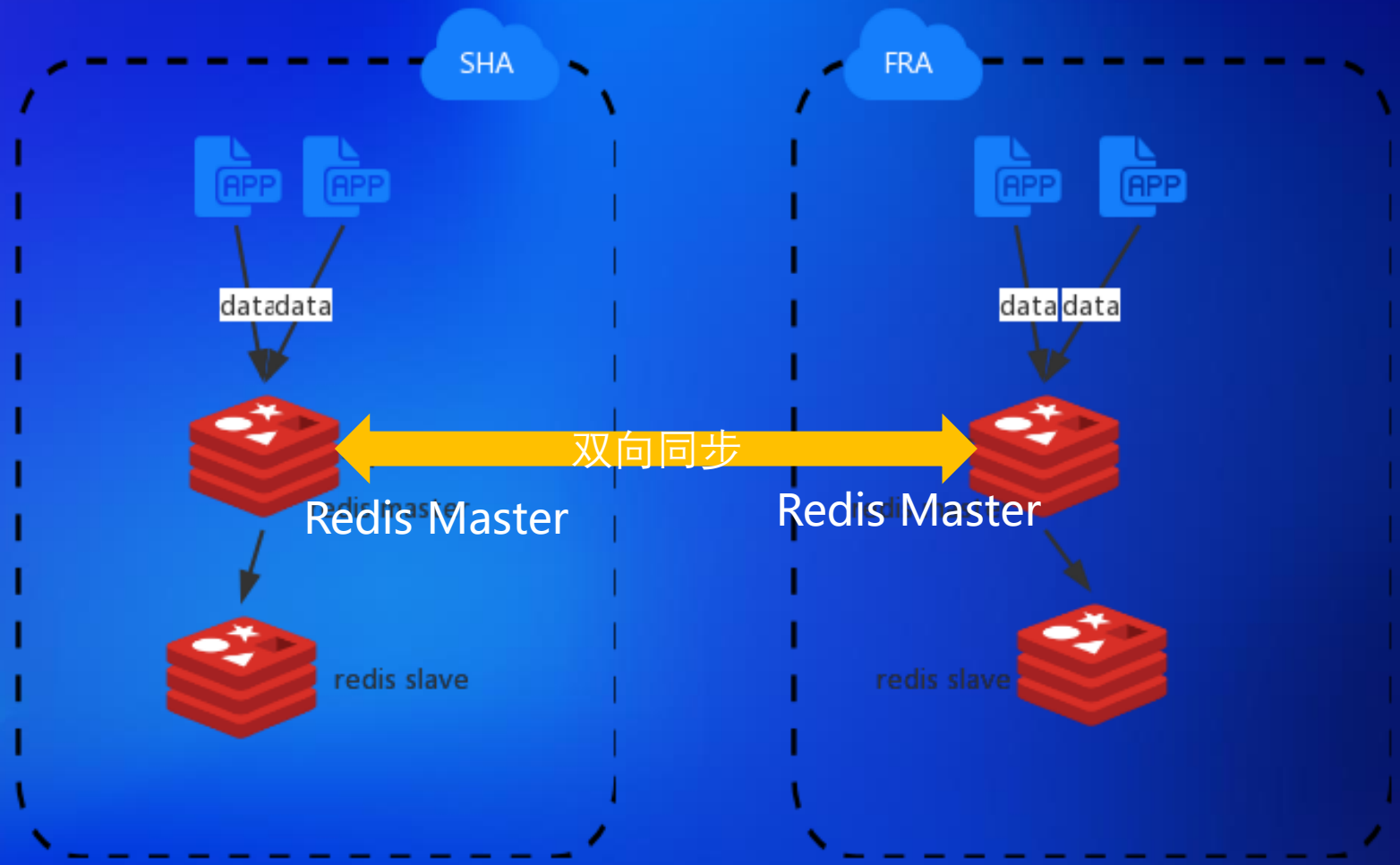
业务痛点

- 我们希望可以通过Redis的双向同步解决重复收费的问题



双向同步

Redis双向同步

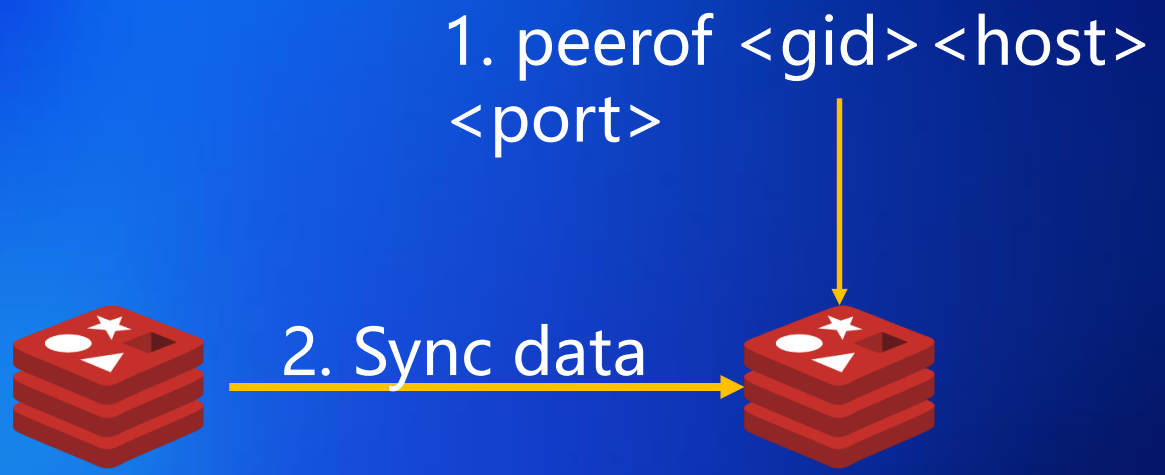

SHA
FRA
APP
APP
APP
APP
datadata
data data
双向同步
Redis Master
Redis Master
redis slave
redis slave

Redis双向同步

- slaveof命令
- redis变成slave，同步数据
- Slave无法写入

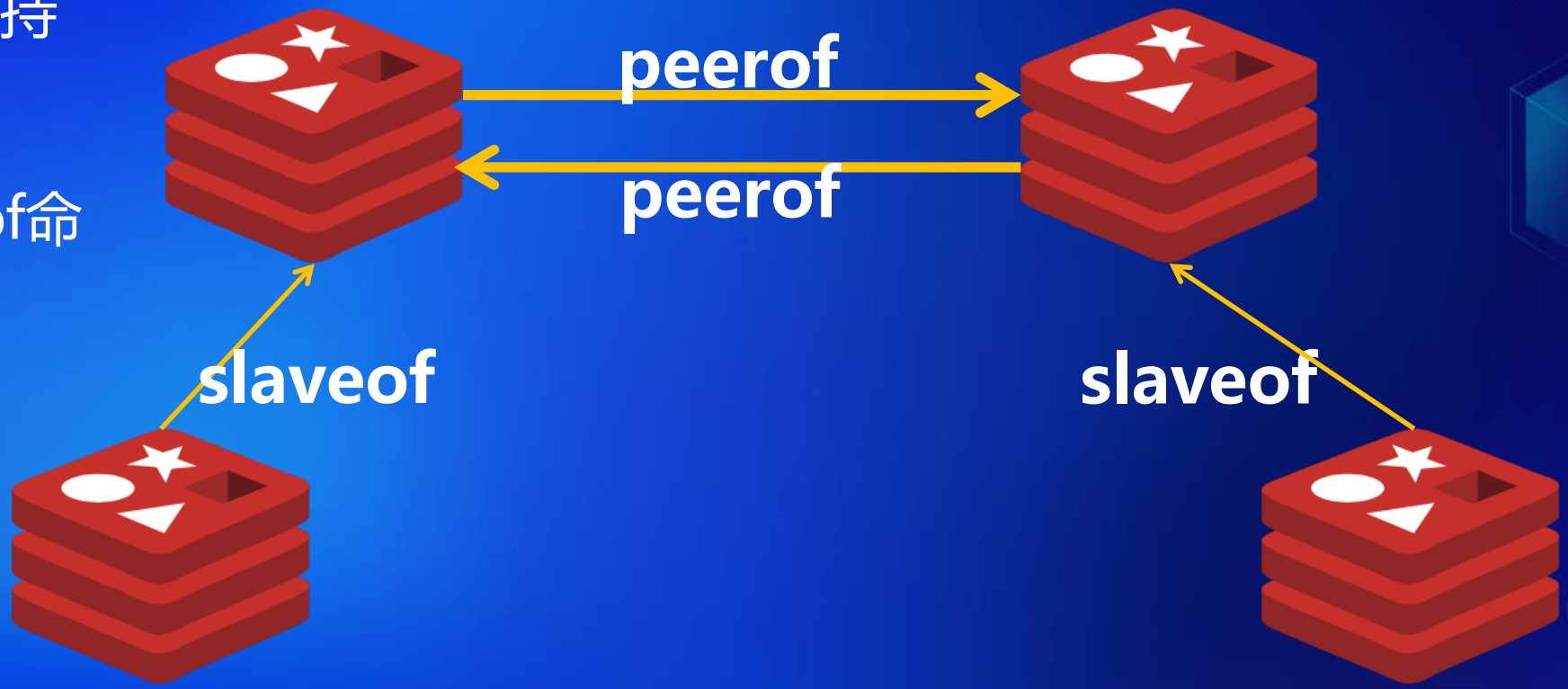
Redis双向同步

- 新的命令“peerof”
- 同步数据
- 继续保持Master的角色



如何解决

- 新的协议支持双向同步
- 兼容Slaveof命令



Peerof 命令

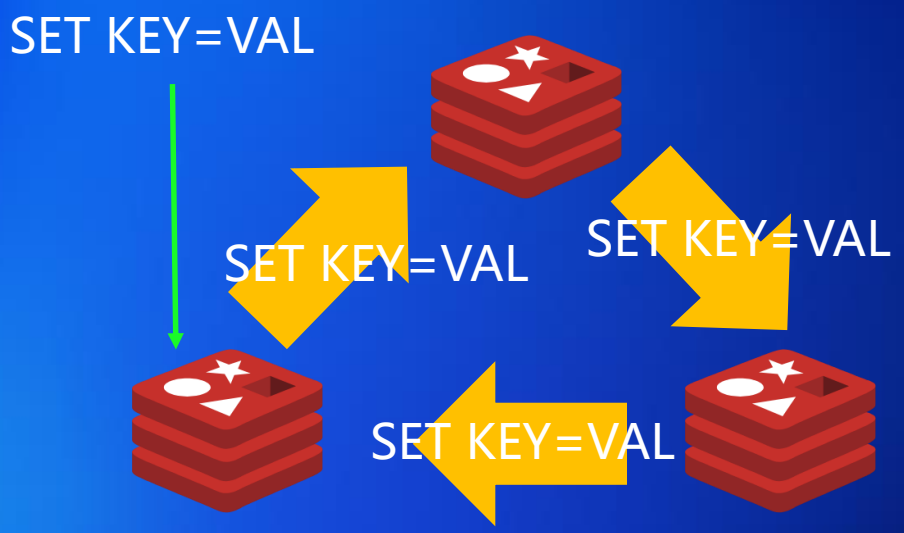
- Redis提供了方便开发的平台
- 实现一个命令的方式

```
    {"peerof",peerofCommand,4,"ast",0,NULL,0,0,0,0,0},
    {"debugCancelCrdt",debugCancelCrdt,3,"ast",0,NULL,0,0,0,0,0},
    {"tombstoneSize",tombstoneSizeCommand,1,"rF",0,NULL,0,0,0,0,0},
    {"crdt.ovc",crdtOvcCommand,3,"rF",0,NULL,0,0,0,0,0},
};
```

```
// peerof <gid> <ip> <port>
//  0       1    2    3
void peerofCommand(client *c) {
    /* PEEROF is not allowed in cluster mode as replication is automatically
    * configured using the current address of the master node. */
    if (server.cluster_enabled) {
        addReplyError(c, err: "PEEROF not allowed in cluster mode.");
        return;
    }

    long port;
    long long gid;
    if ((getLongLongFromObjectOrReply(c,  o: c->argv[1], &gid,  msg: NULL) != C_OK))
        return;

    if (!strcasecmp(c->argv[2]->ptr,"no") &&
        !strcasecmp(c->argv[3]->ptr,"one")) {

        if (getPeerMaster(gid)) {
            crdtReplicationUnsetMaster(gid);
            sds client = catClientInfoString(sdsempty(),c);
            serverLog(LL_NOTICE, fmt: "[CRDT] REMOVE MASTER %lld enabled (user request from '%s')",
                      gid, client);
            sdsfree(client);
        }

        server.dirty ++;
        addReply(c, shared.ok);
        return;
    }
```

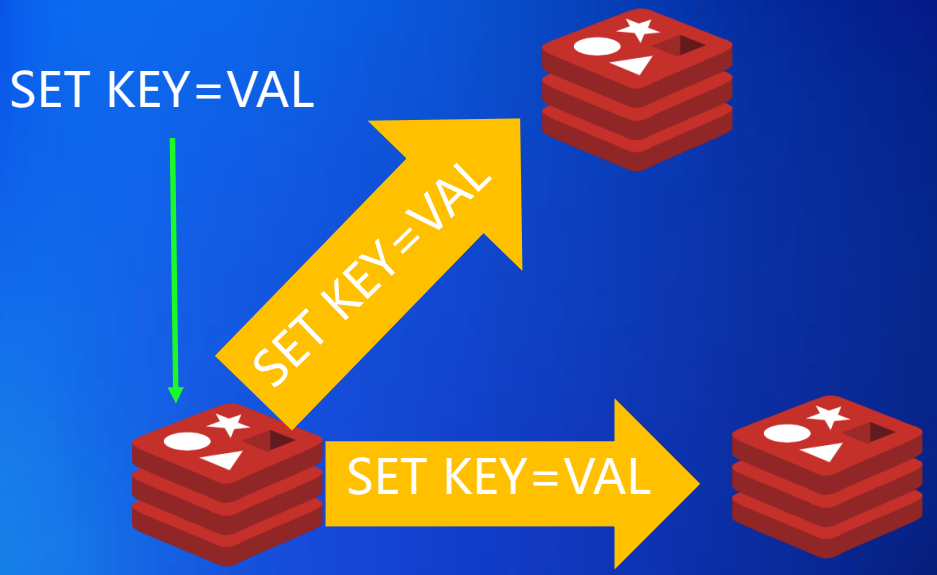
回环复制

- 网络风暴
- 数据不一致



如何解决

- 标记数据来源
- 只发送本站点数据



两份缓存

SET
KEY=VAL

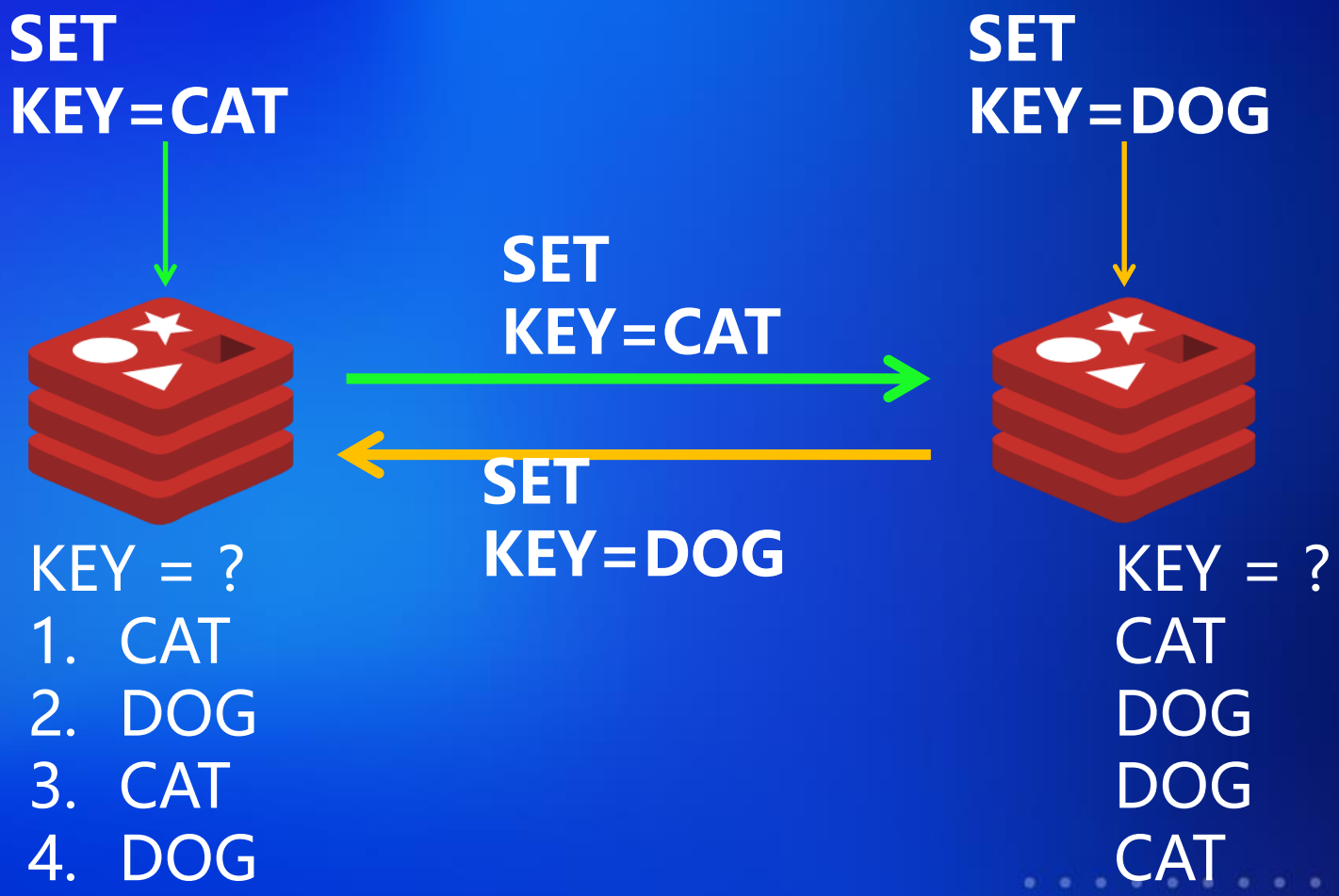
SET KEY=VAL

Master-Master
缓存区

SET KEY=VAL

Master-Slave
缓存区

写入冲突


SET
KEY=CAT
SET
KEY=DOG
SET
KEY=CAT
SET
KEY=DOG
KEY = ?
1. CAT
2. DOG
3. CAT
4. DOG
KEY = ?
CAT
DOG
DOG
CAT

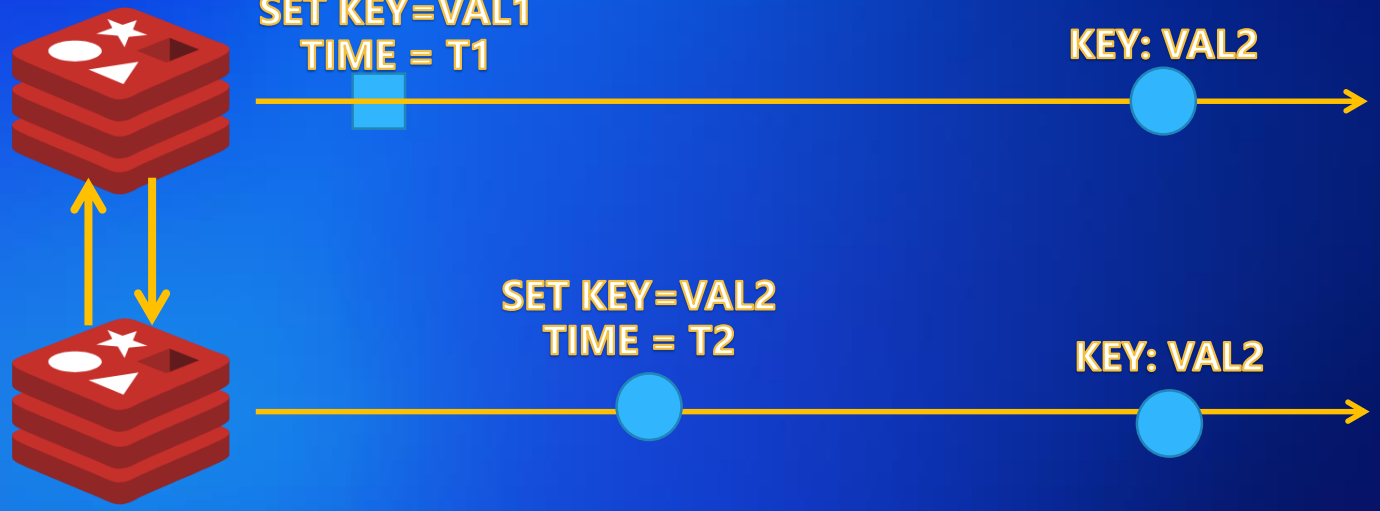
CRDT

Conflict-free Replicated Data Types

CRDT
– Last Write Wins

T2 > T1

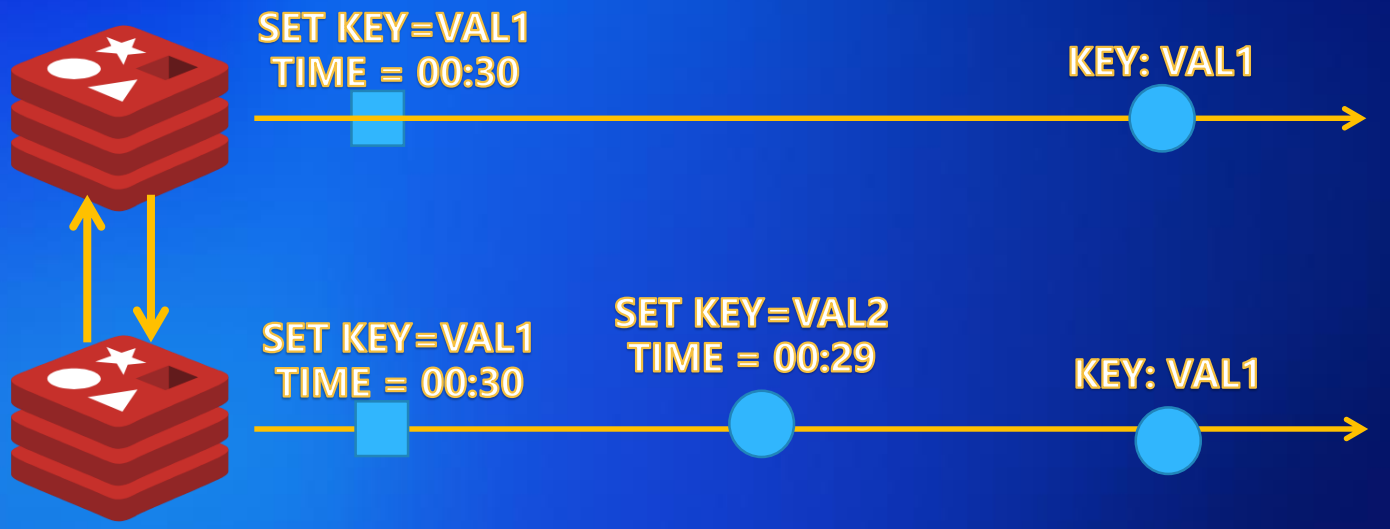
T2 WINS



时间不一致

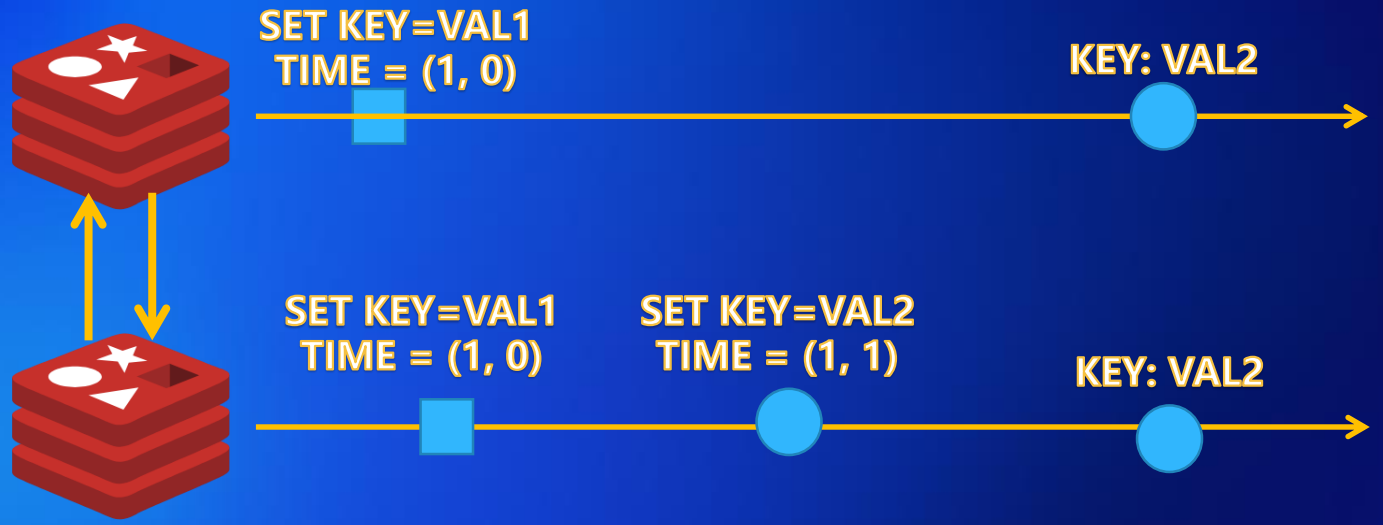
两个Redis的系统时钟不一致

最终保留了第一次的结果



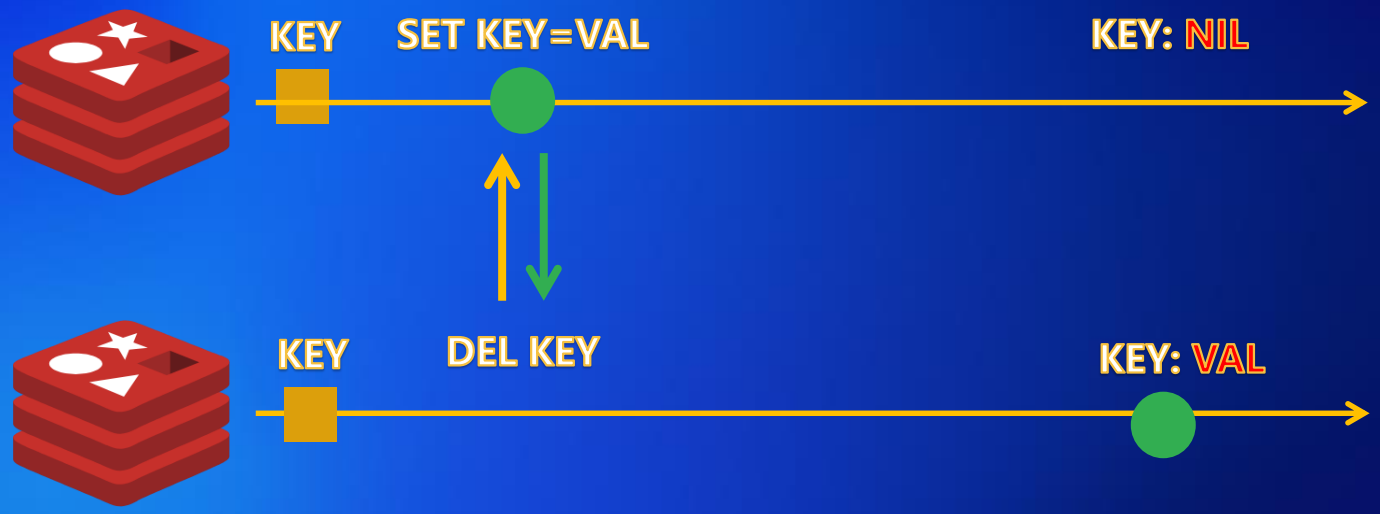
Vector Clock

- 向量表示不同节点的操作数
- SET KEY=VAL1
  (0,0)-> (1,0)
- SET KEY=VAL2
  (1,0) -> (1,1)



删除导致数据不一致

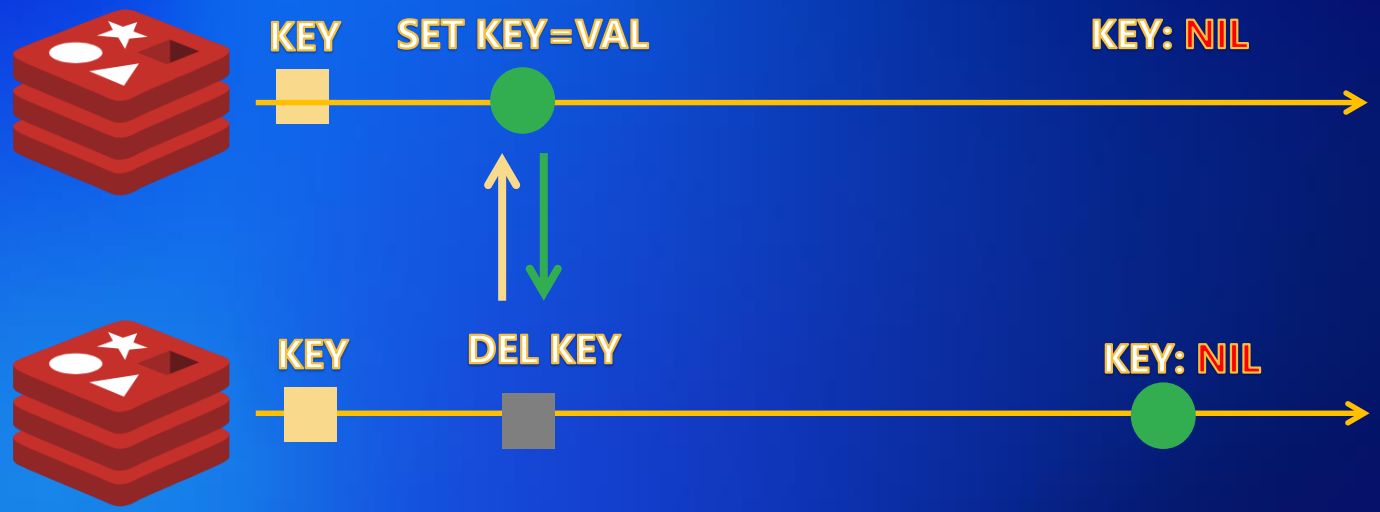
- 假设已经存在一个KEY
- Redis-A做更新操作
- Redis-B做删除操作



CRDT -- Tombstone

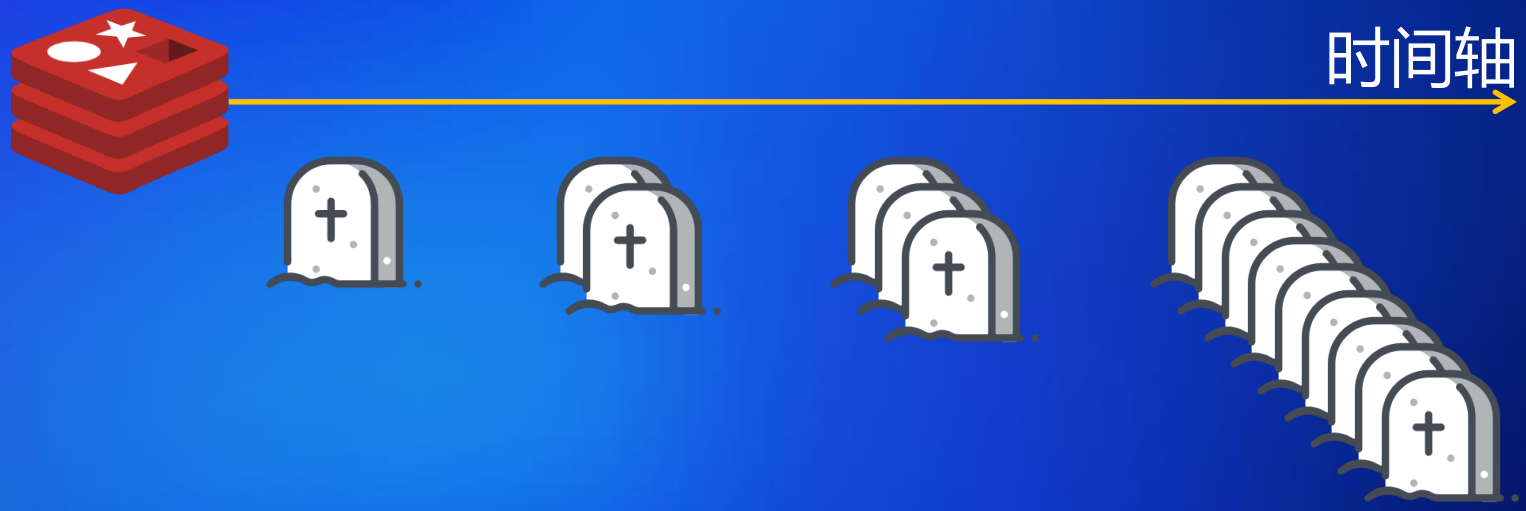
删除操作时，只做逻辑删除

保留被删除的记录



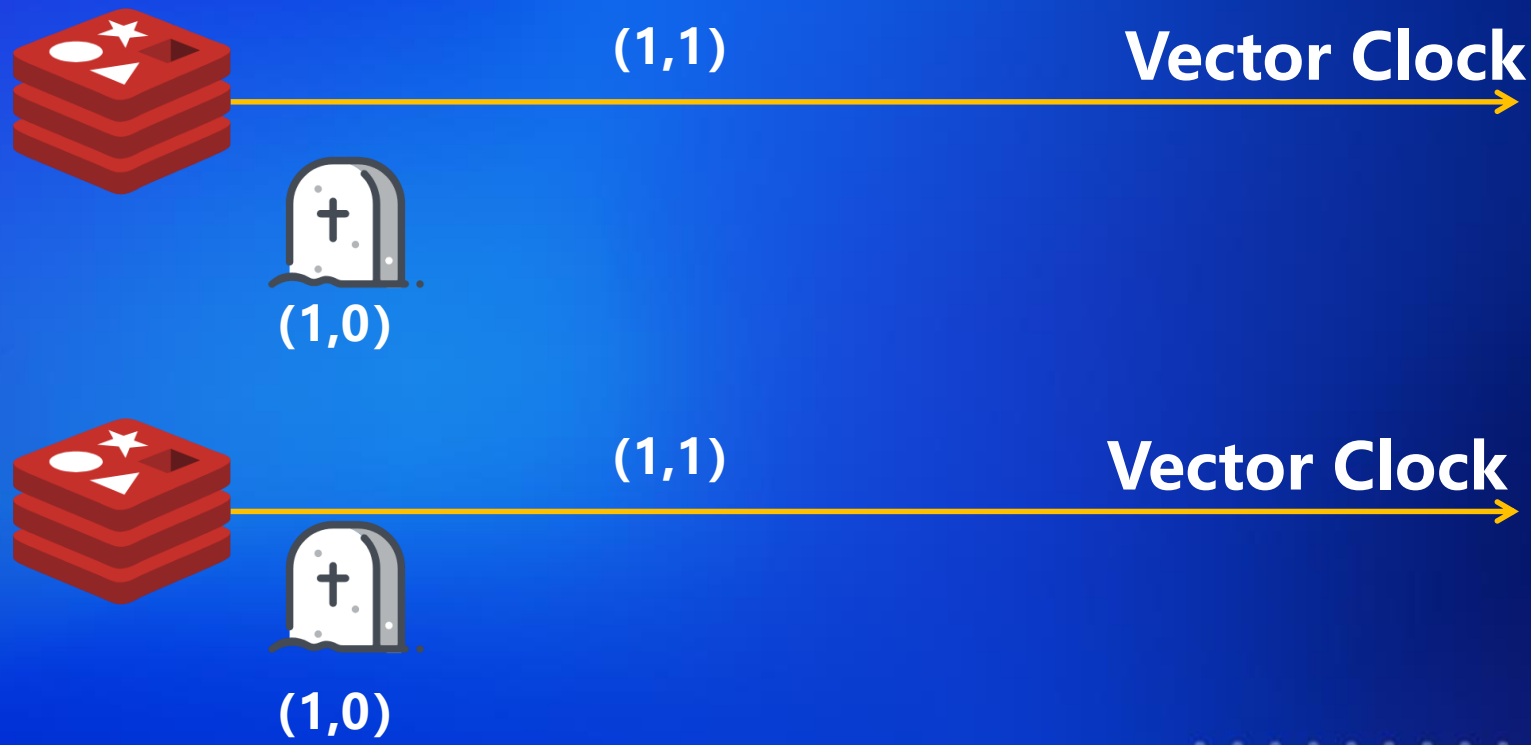
内存爆满

随着时间的推移，大量的失效KEY驻留



CRDT -- GC

基于节点之间的vector clock的通讯，删除不必要的失效KEY



什么是 CRDT

State-based Replication

- 交换律
- 结合律
- 幂等性

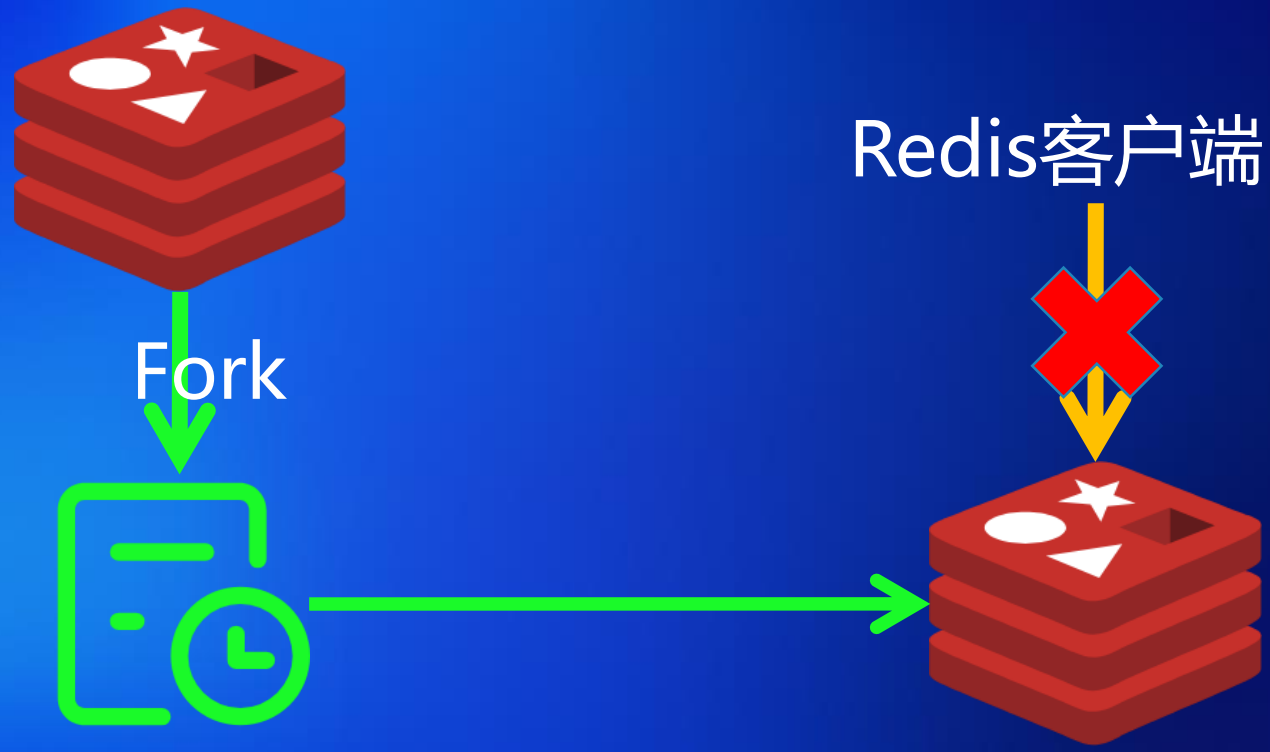
Op-based Replication

- 交换律
- 结合律

高可用

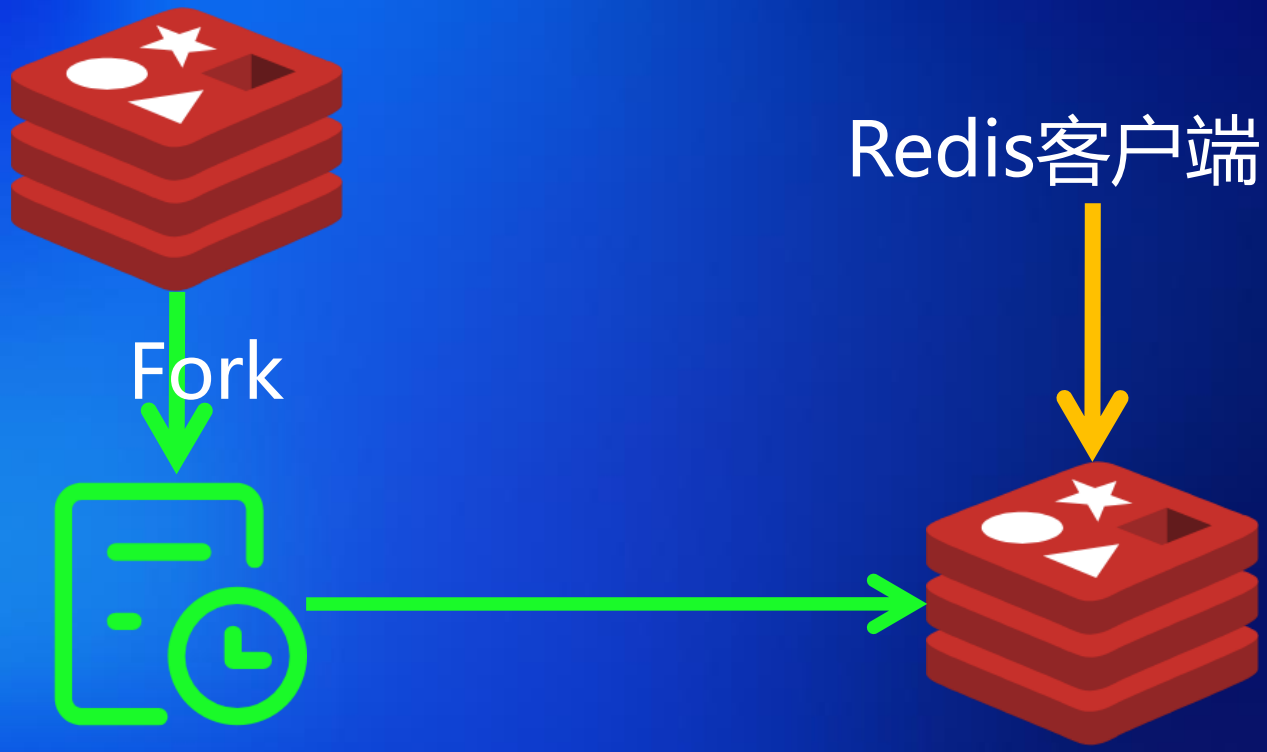
全量同步

- Redis生成内存快照
- 发送给下游Redis同步
- 期间，下游Redis不可用



CRDT的优势

- Redis生成内存快照
- 使用OP-LOG的形式发送
- 期间，下游Redis可用



- CRDT入门
- A CRDT Primer Part I: Defanging Order Theory
- A CRDT Primer Part II: Convergent CRDTs
-
- CRDT相关论文
- 重点推荐: A comprehensive study of Convergent and Commutative Replicated Data Types
- Conflict-free replicated data types
- Delta State Replicated Data Types
- CRDTs: Making δ-CRDTs Delta-Based
- Key-CRDT Stores
- A Conflict-Free Replicated JSON Datatype
- OpSets: Sequential Specifications for Replicated Datatypes
-

Thanks For Watching

本PPT来自2019携程技术峰会

更多信息请关注“携程技术中心”微信公众号~