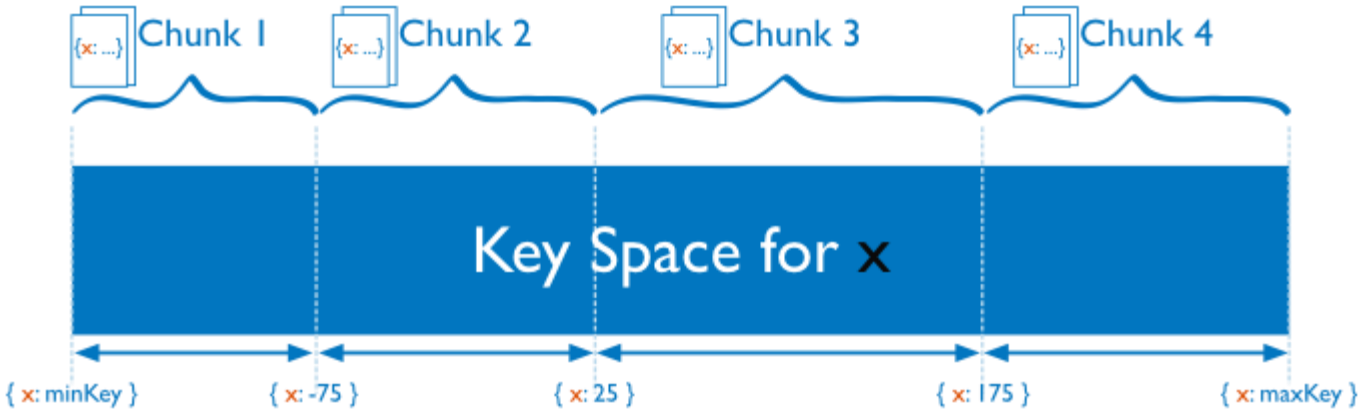
分片集群中的一个写入操作

郭一然

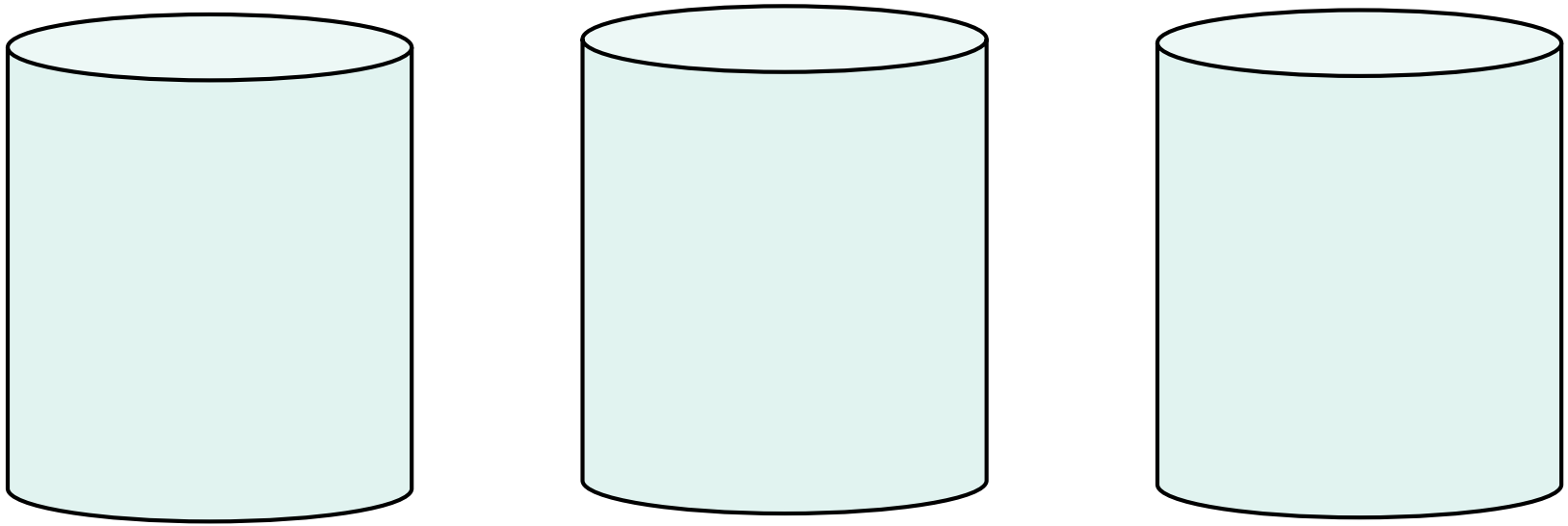
(Based on Randolph’s MDBW '16 talk)

MONGODB WORLD'16 | FOR GIANT IDEAS

分片概念

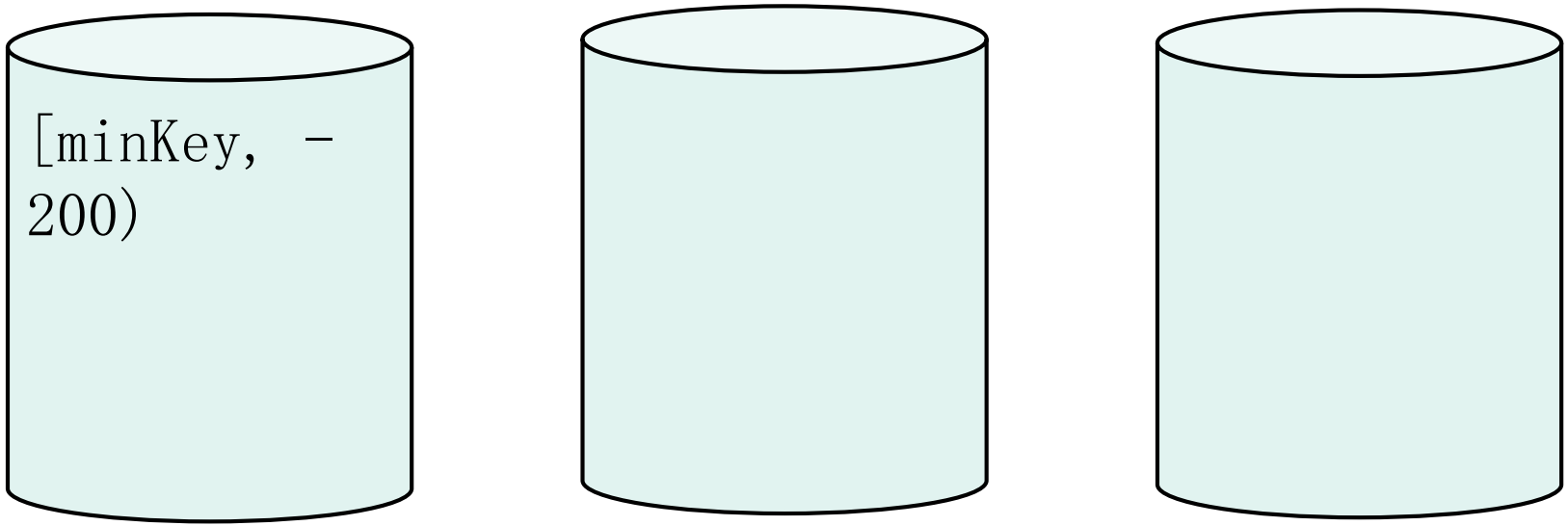

{x: ...}
Chunk 1
{x: ...}
Chunk 2
{x: ...}
Chunk 3
{x: ...}
Chunk 4
Key Space for x
{ x: minKey }
{ x: -75 }
{ x: 25 }
{ x: 175 }
{ x: maxKey }

分片概念

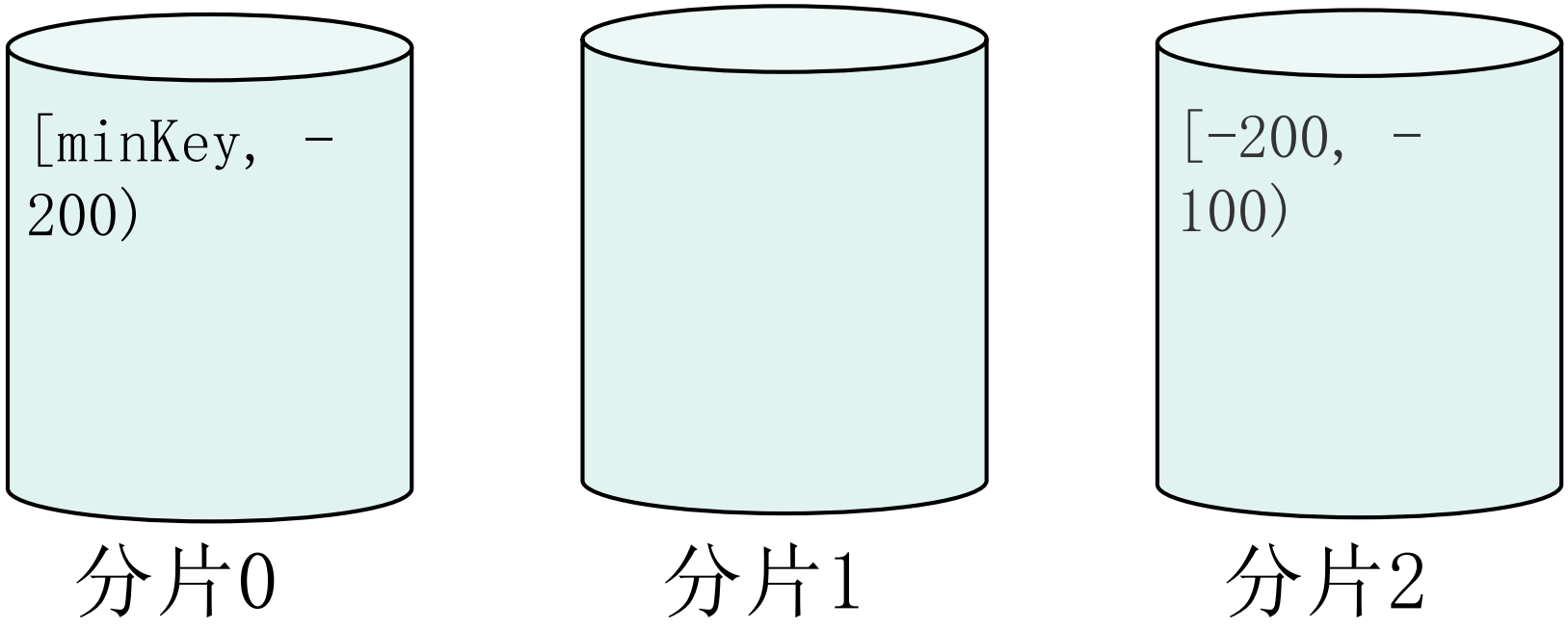


分片0　　分片1　　分片2

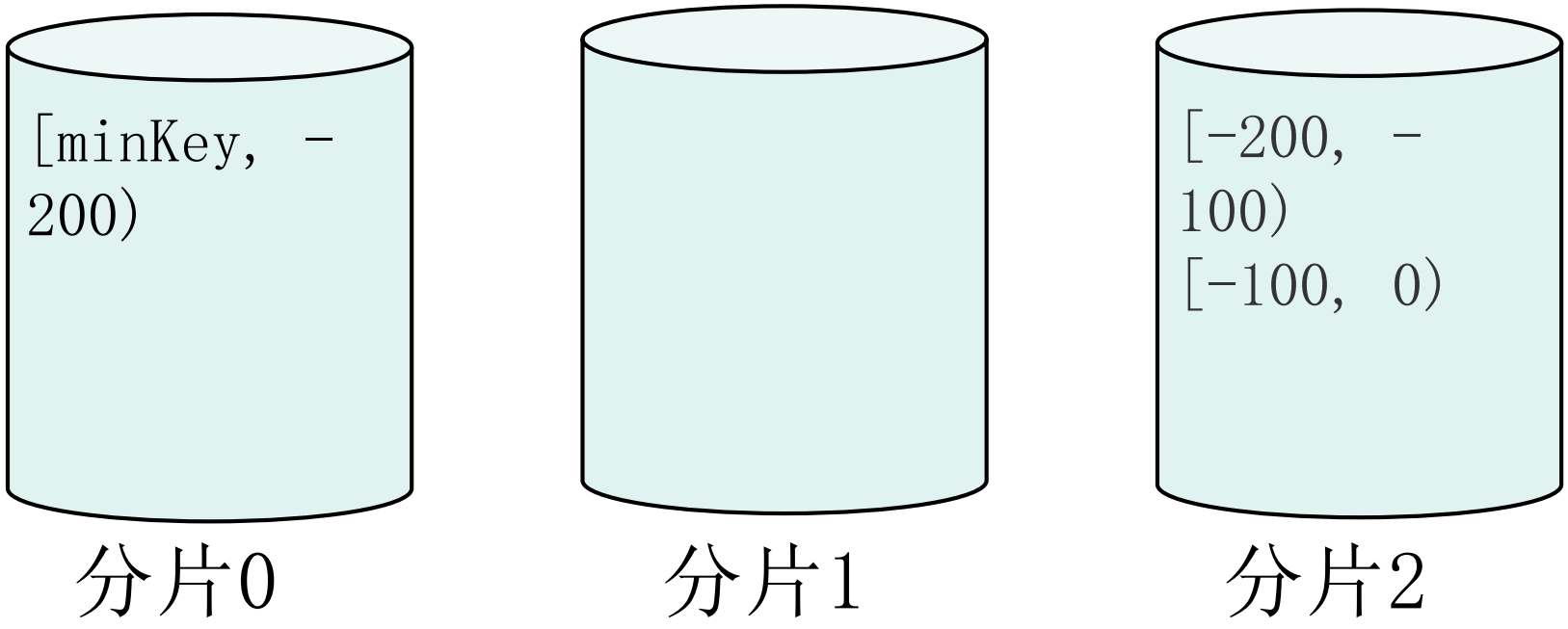
分片概念



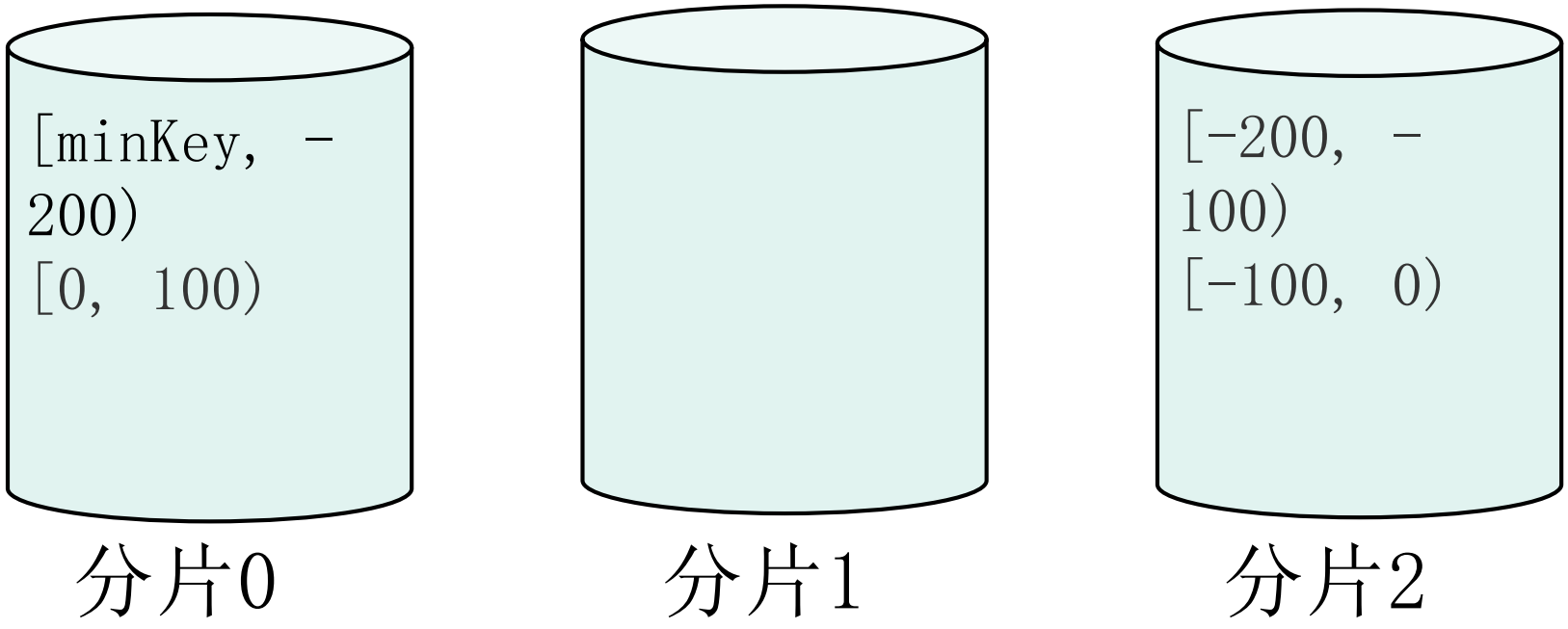
分片概念



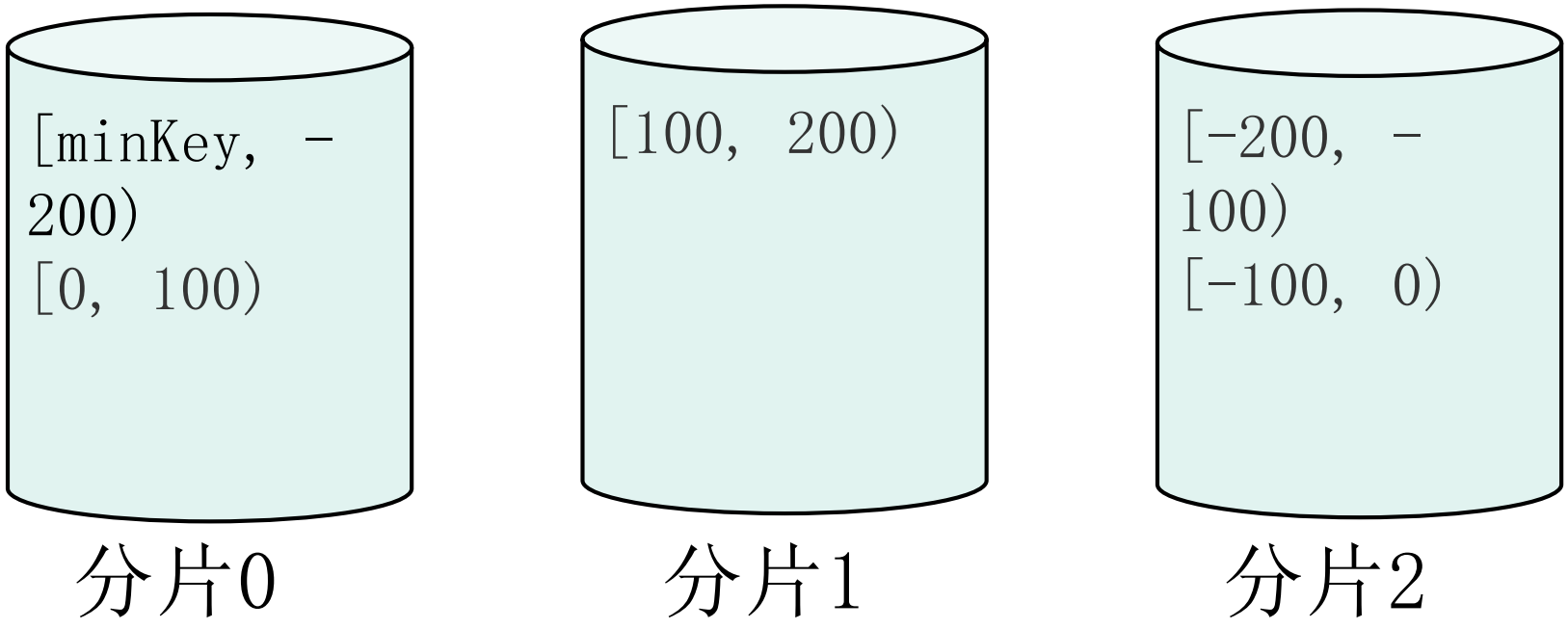
分片概念



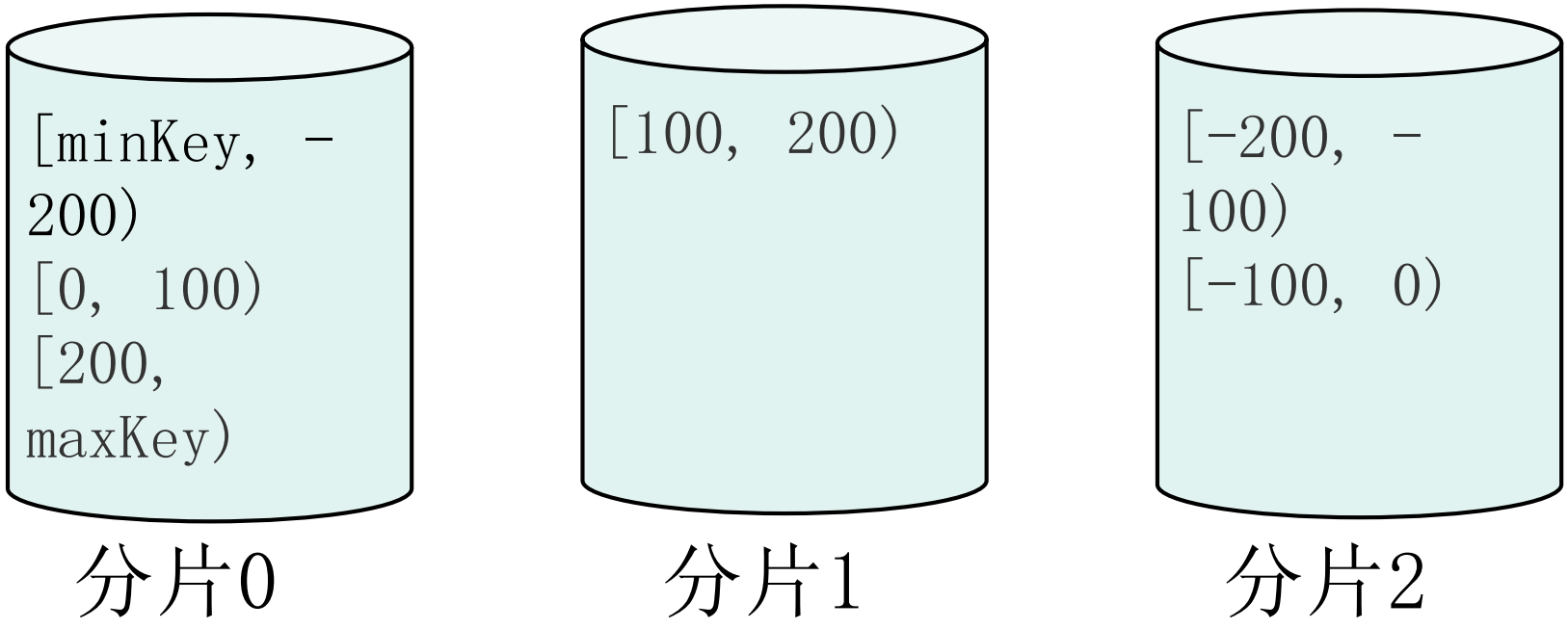
分片概念



分片概念



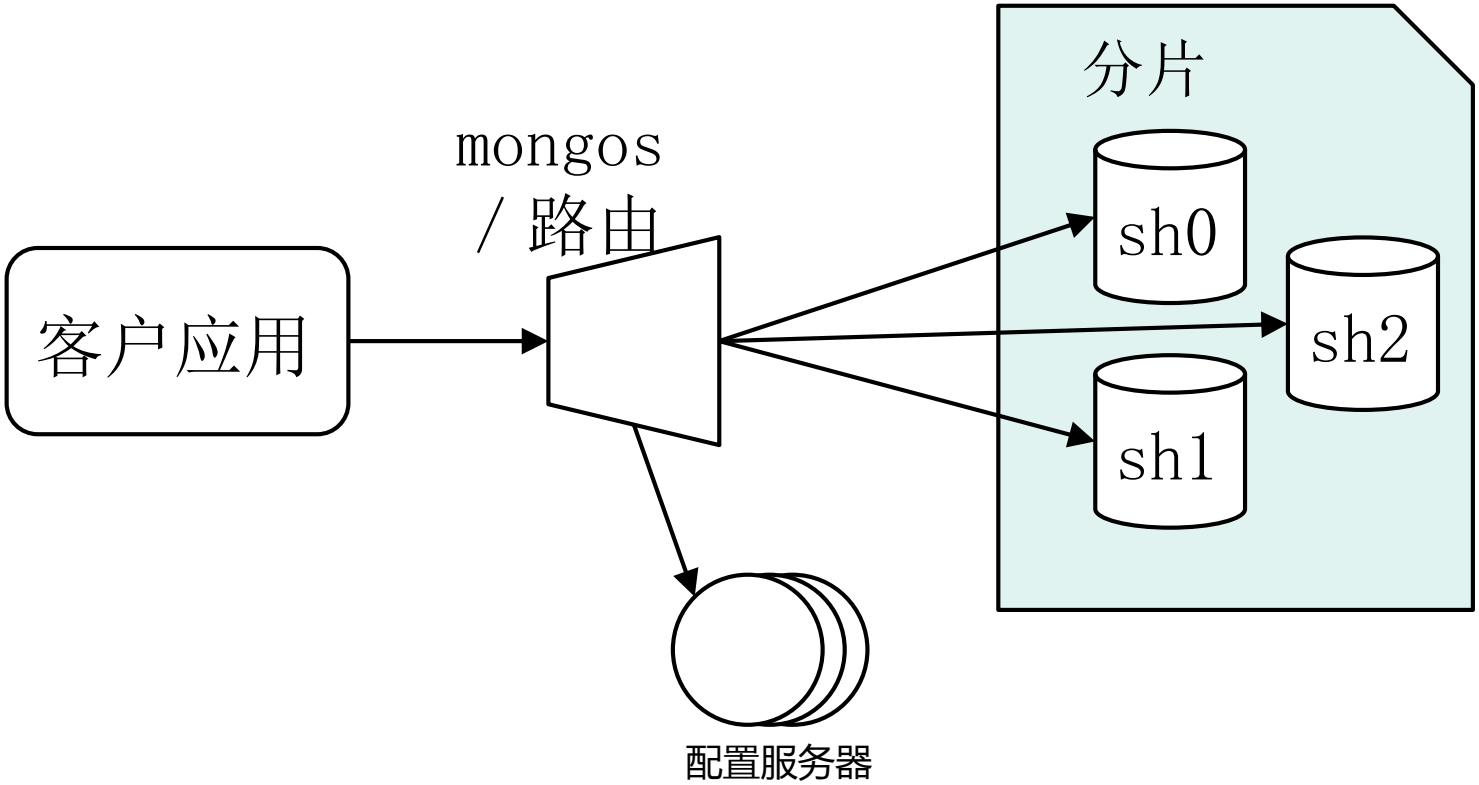
分片概念



路由表

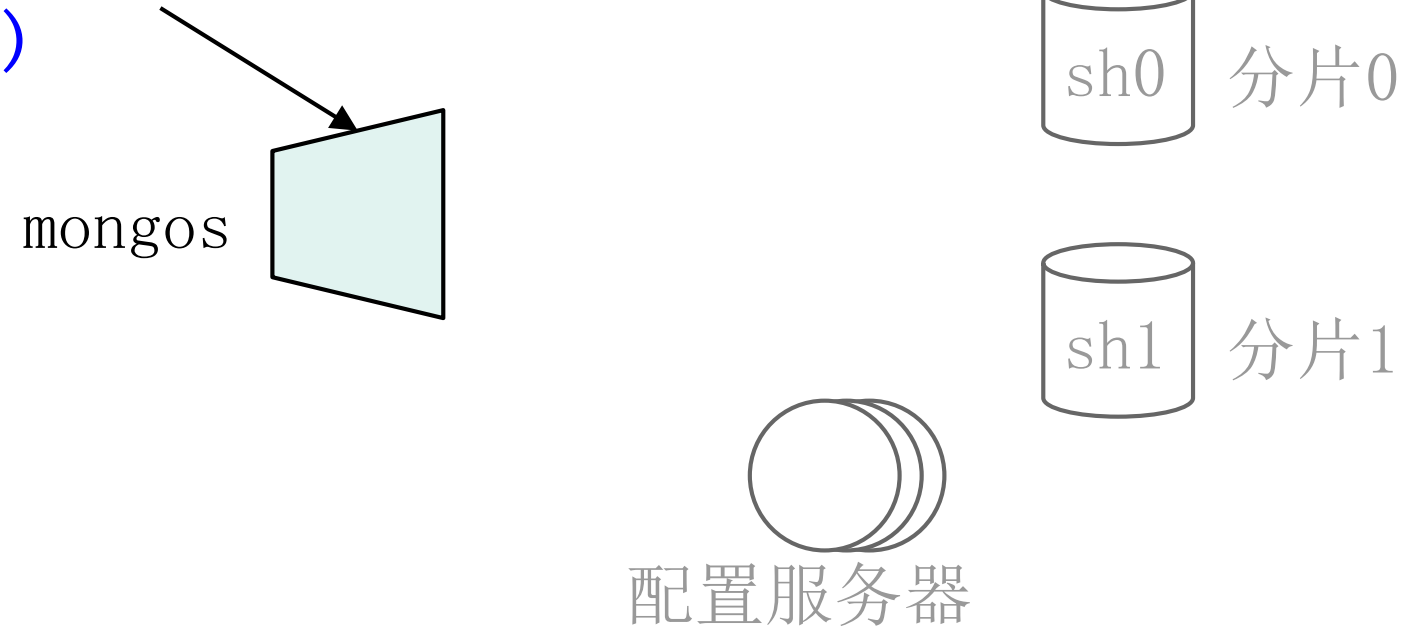
最小值	最大值	所在分片
MinKey	-200	分片0
-200	-100	分片2
-100	0	分片2
0	100	分片0
100	200	分片1
200	MaxKey	分片0

分片集群架构

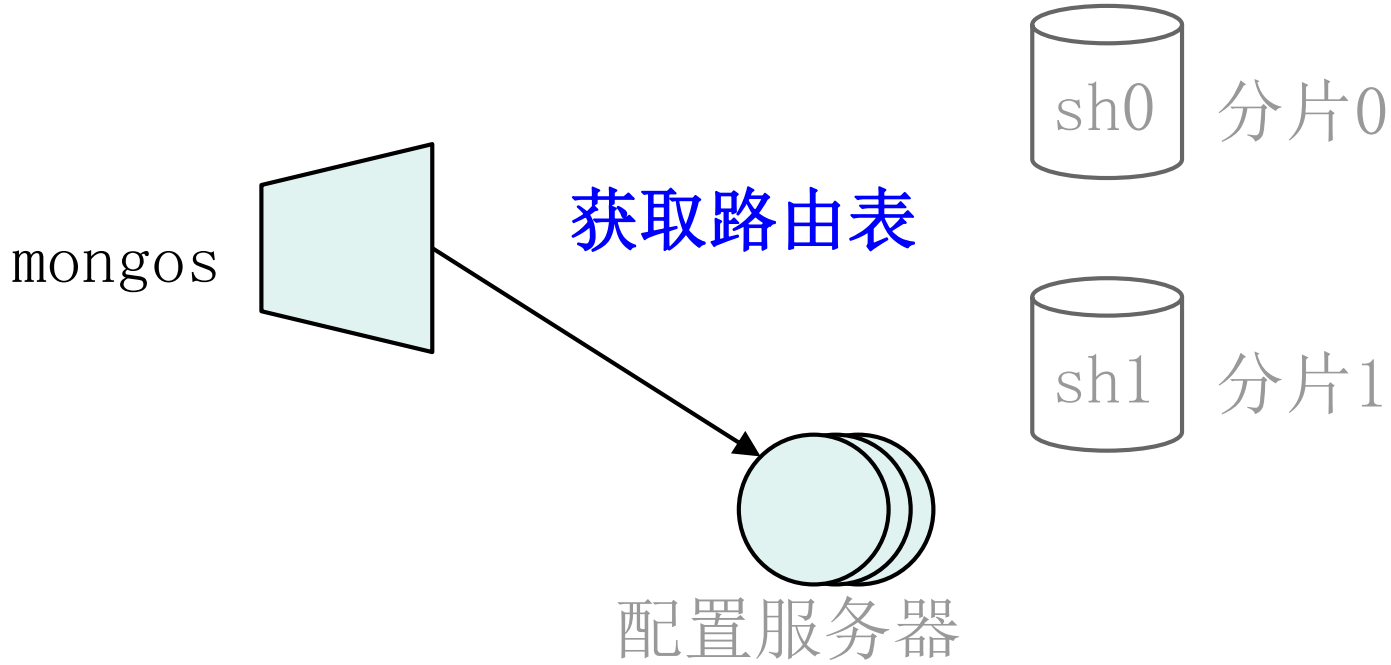


单文档写入

```
db.rio.insert({ 金牌:
8 })
```



单文档写入

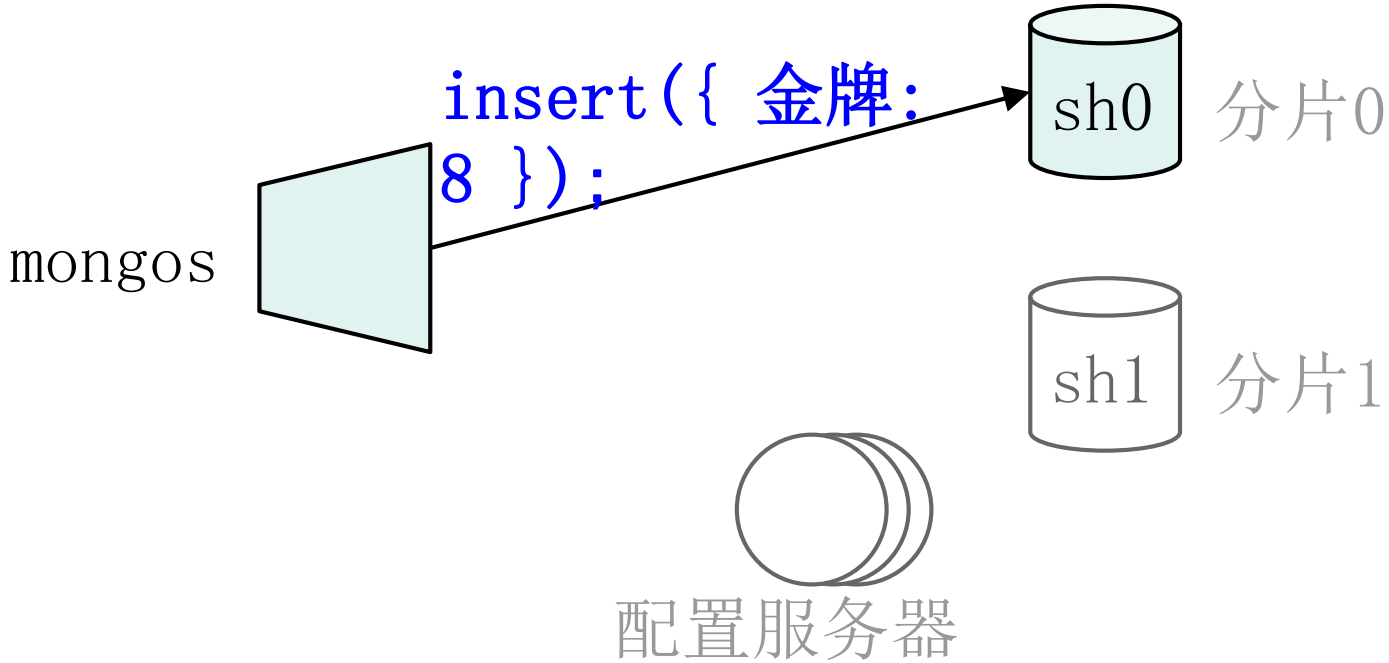


路由表

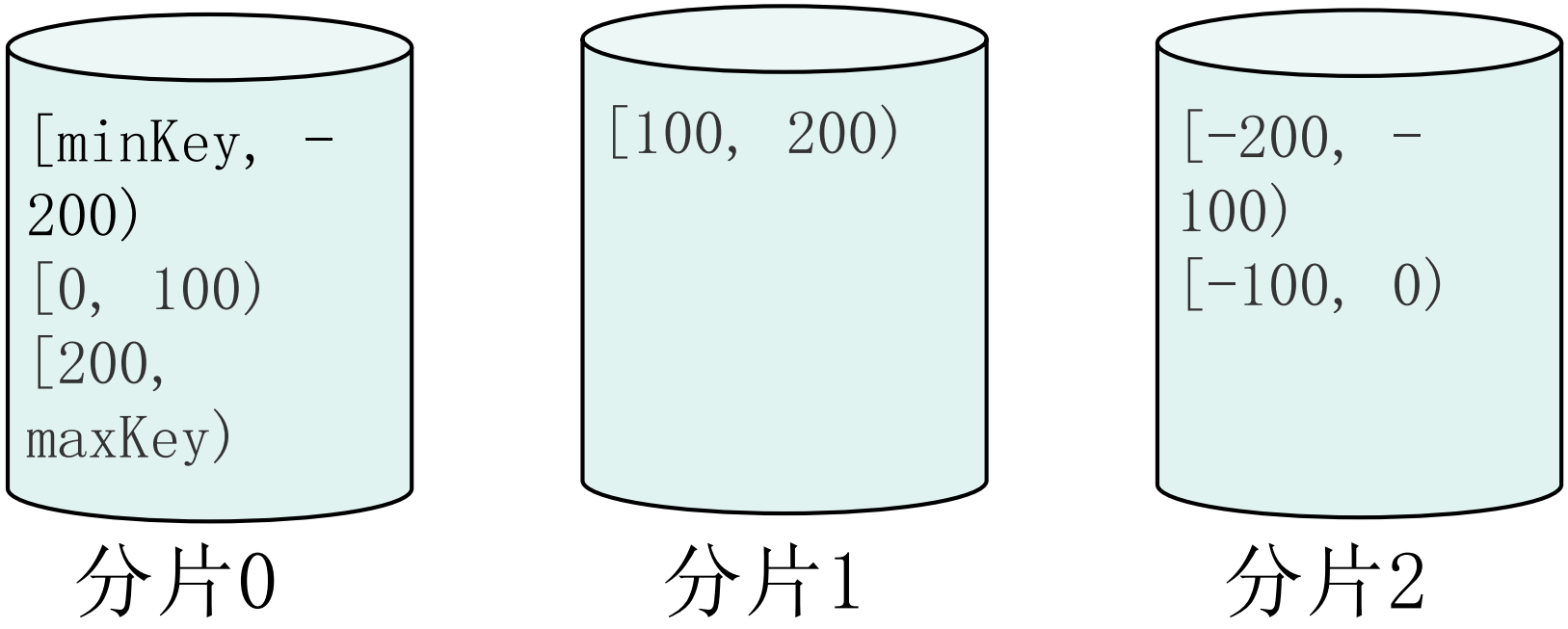
```
db.rio.insert({ 金牌: 8 });
```

最小值	最大值	所在分片
MinKey	-200	分片0
-200	-100	分片2
-100	0	分片2
0	100	分片0
100	200	分片1
200	MaxKey	分片0

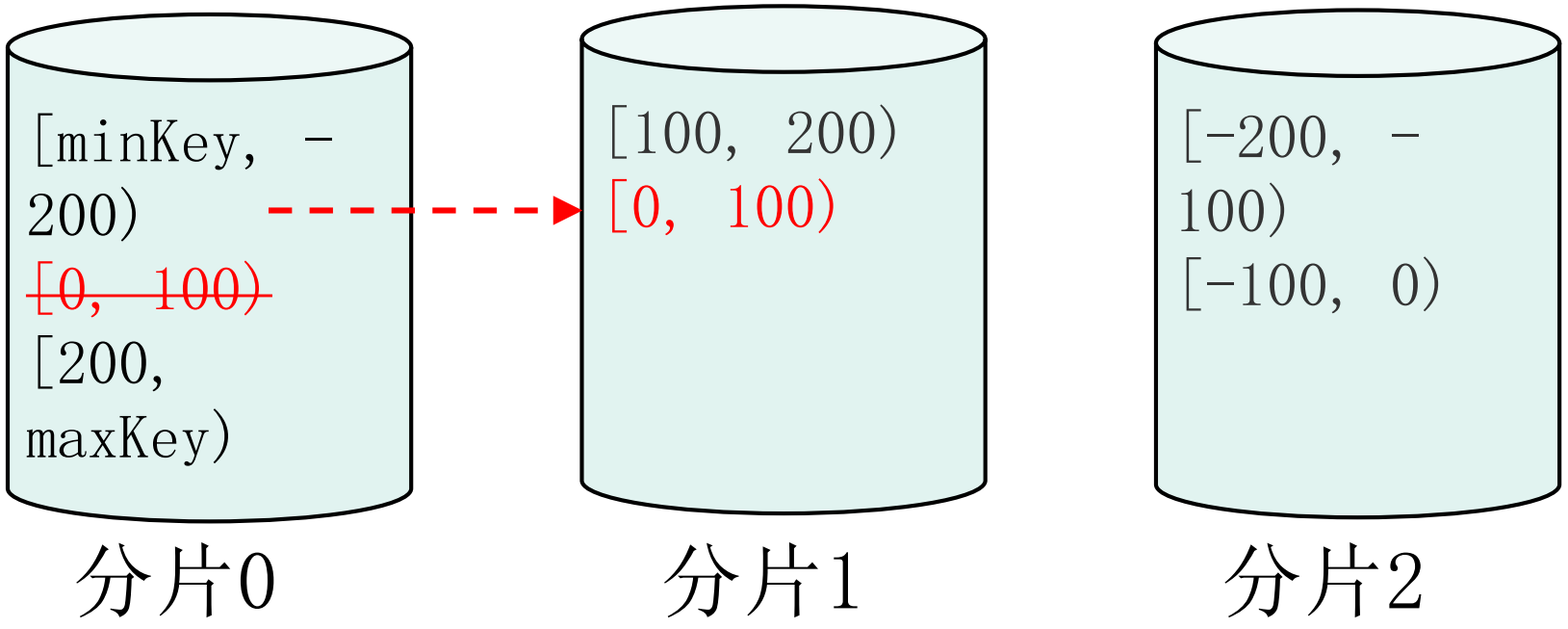
单文档写入



数据段迁徙



数据段迁徙


[minKey, -200)
[0, 100)
[200, maxKey)
[100, 200)
[0, 100)
[-200, -100)
[-100, 0)
分片0
分片1
分片2

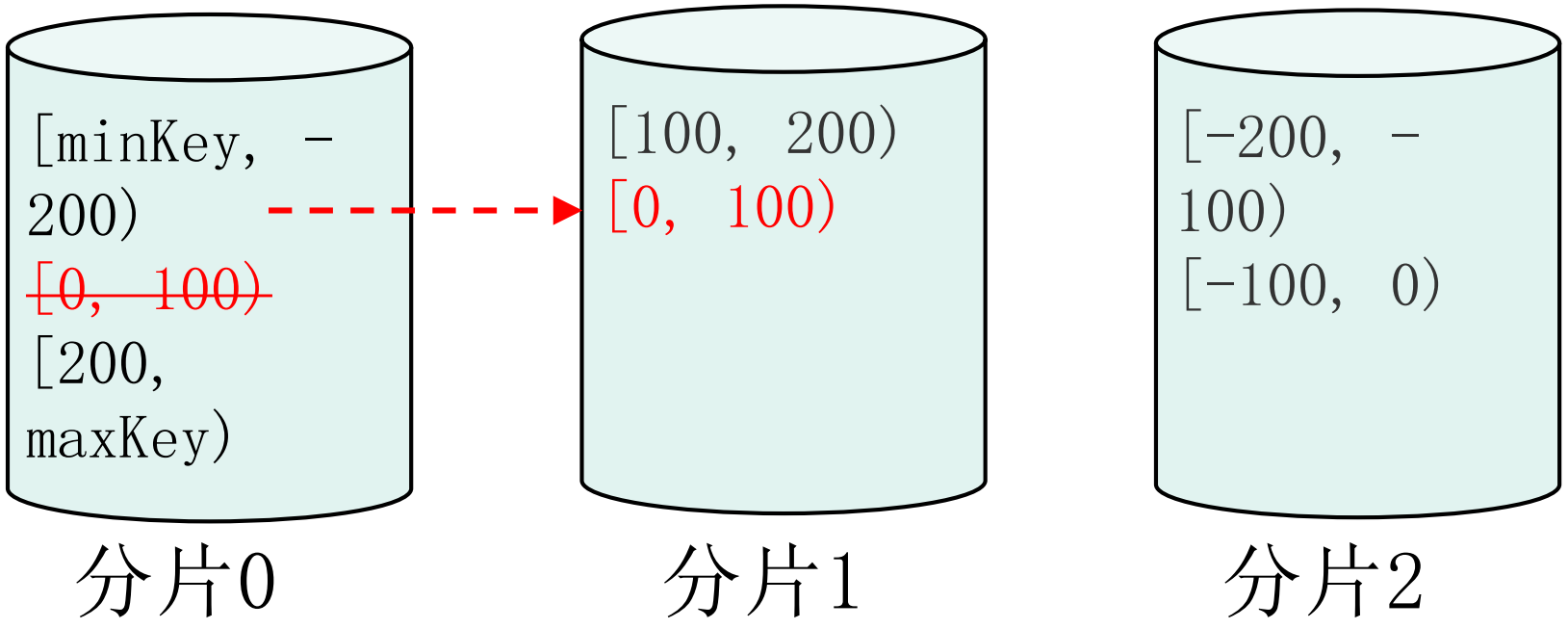
分片版本协议

版本	最小值	最大值	所在分片
1	MinKey	-200	分片0
2	-200	-100	分片2
3	-100	0	分片2
4	0	100	分片0
5	100	200	分片1
6	200	MaxKey	分片0

分片版本协议

新版本 = max(路由表版本) + 1

数据段迁徙

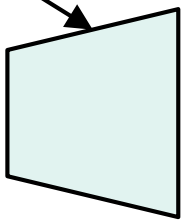

[minKey, -200)
[0, 100)
[200, maxKey)
[100, 200)
[0, 100)
[-200, -100)
[-100, 0)
分片0
分片1
分片2

分片版本协议

版本	最小值	最大值	所在分片
1	MinKey	-200	分片0
2	-200	-100	分片2
3	-100	0	分片2
4 -> 7	0	100	分片0 -> 分片1
5	100	200	分片1
6	200	MaxKey	分片0

单文档写入

```
db.rio.insert({ 金牌: 8 });
```

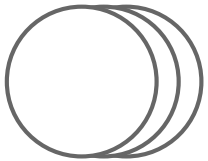


mongos

路由表v6

分片0
路由表v7

分片1
路由表 v7

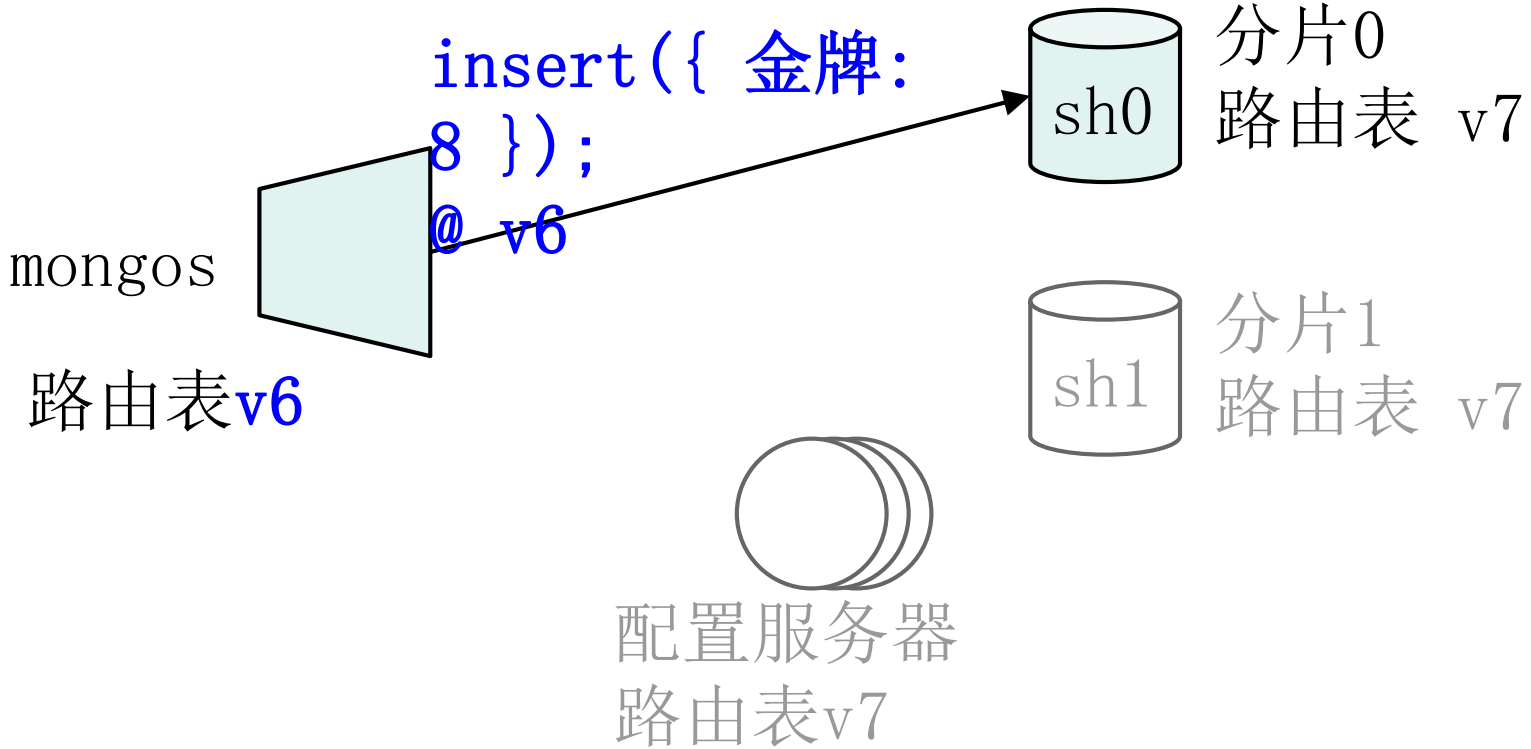


配置服务器路由表v7

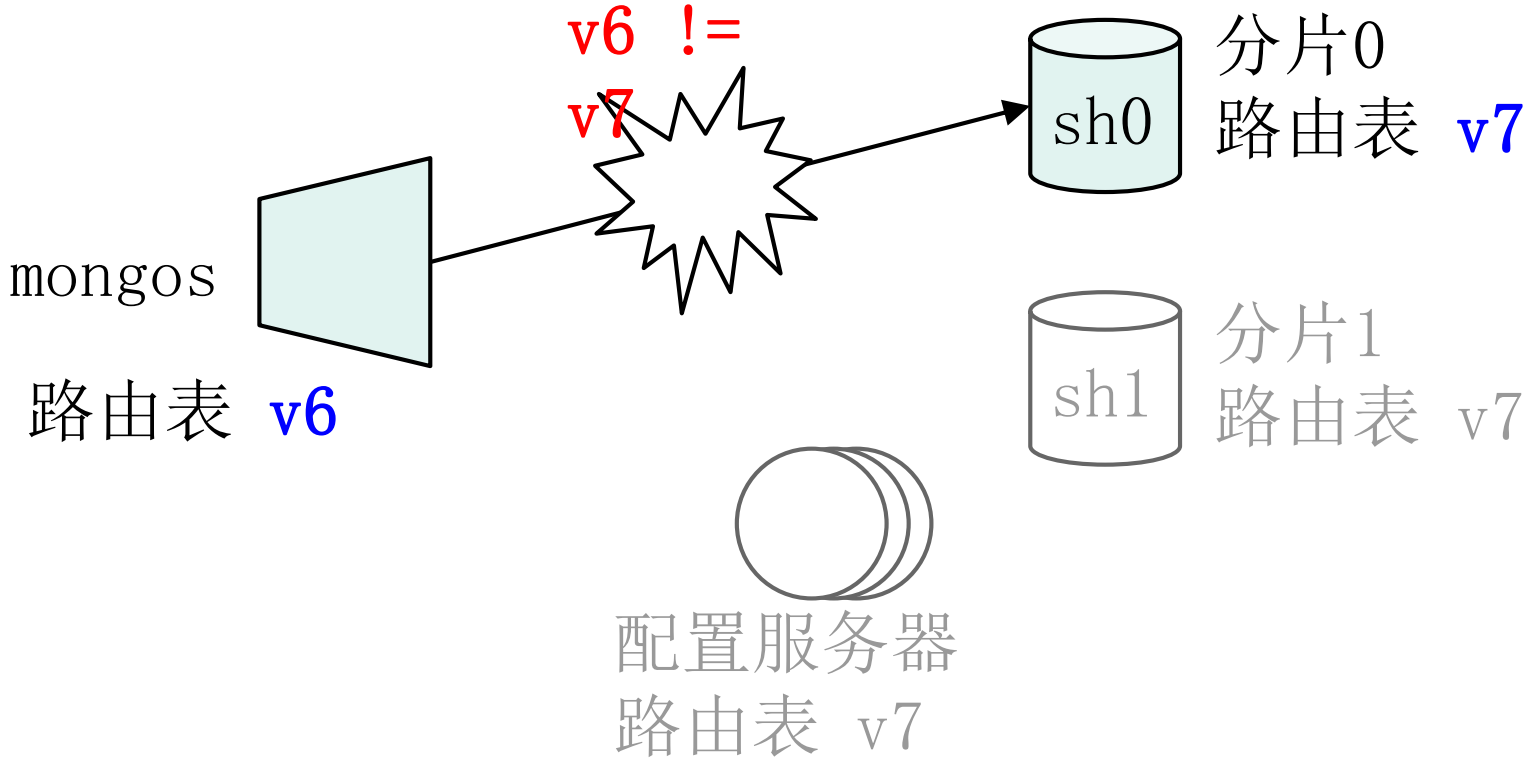
分片版本协议

版本	最小值	最大值	所在分片
1	MinKey	-200	分片0
2	-200	-100	分片2
3	-100	0	分片2
4	0	100	分片0
5	100	200	分片1
6	200	MaxKey	分片0

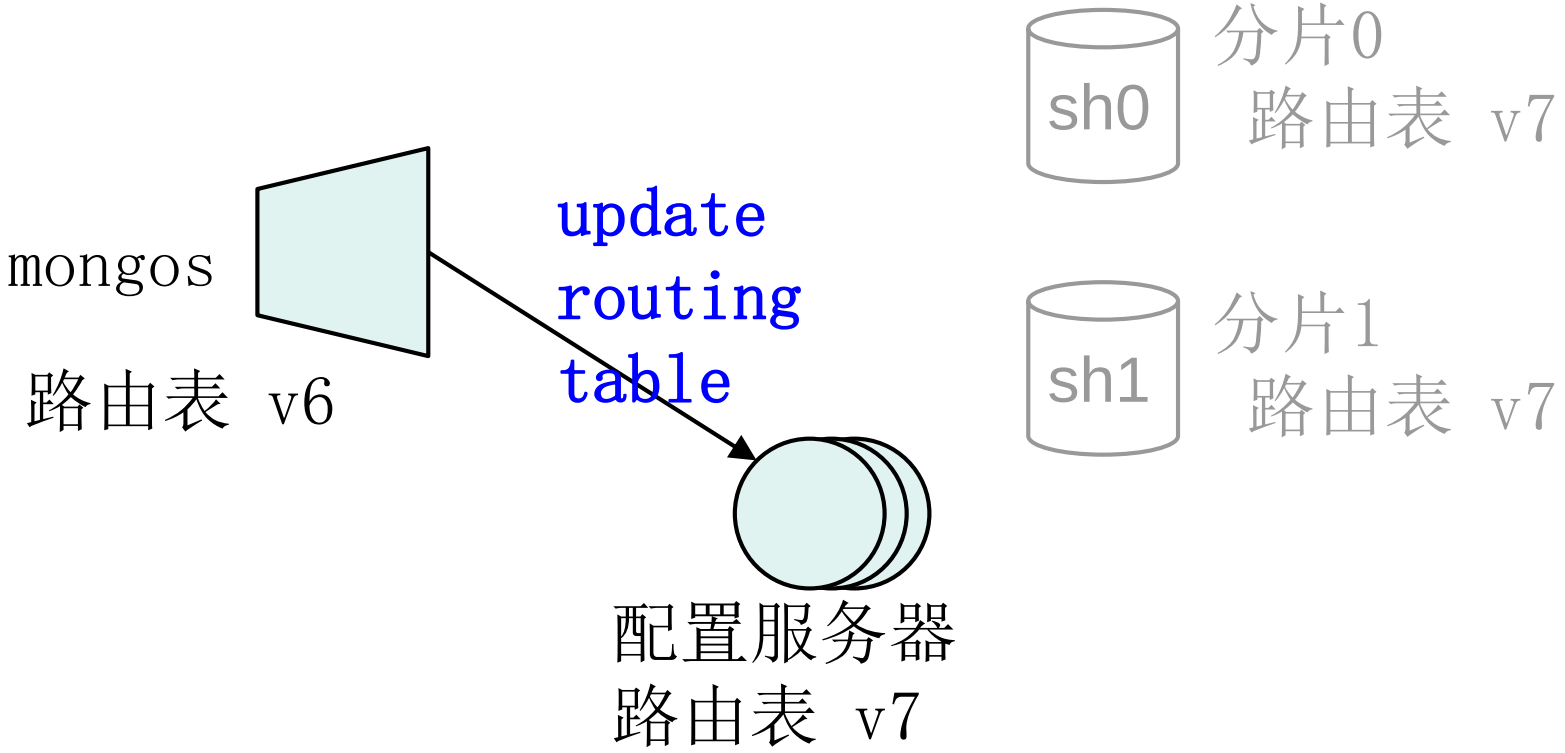
单文档写入


insert({ 金牌:
8 });
@ v6
mongos
路由表v6
sh0
分片0
路由表 v7
sh1
分片1
路由表 v7
配置服务器
路由表v7

单文档写入



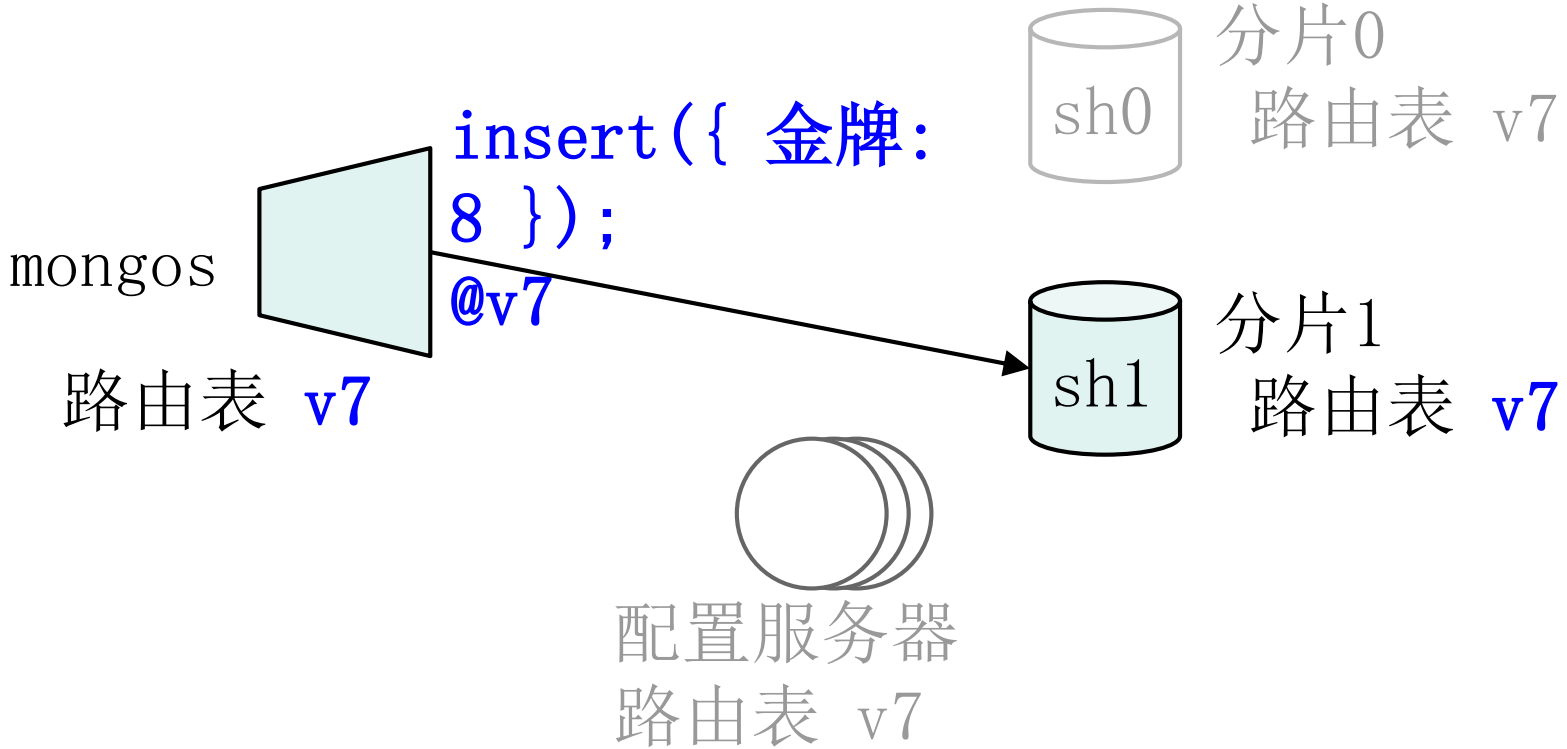
单文档写入


sh0
分片0
路由表 v7
sh1
分片1
路由表 v7
mongos
路由表 v6
update
routing
table
配置服务器
路由表 v7

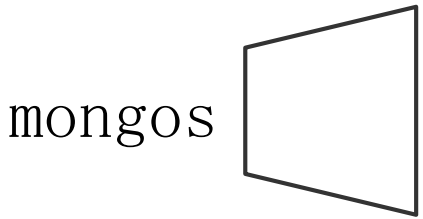
分片版本协议

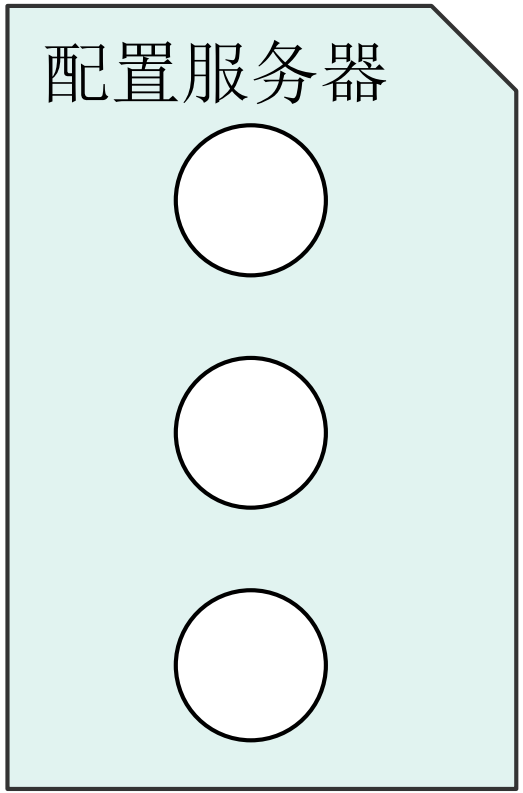
版本	最小值	最大值	所在分片
1	MinKey	-200	分片0
2	-200	-100	分片2
3	-100	0	分片2
7	0	100	分片1
5	100	200	分片1
6	200	MaxKey	分片0

单文档写入

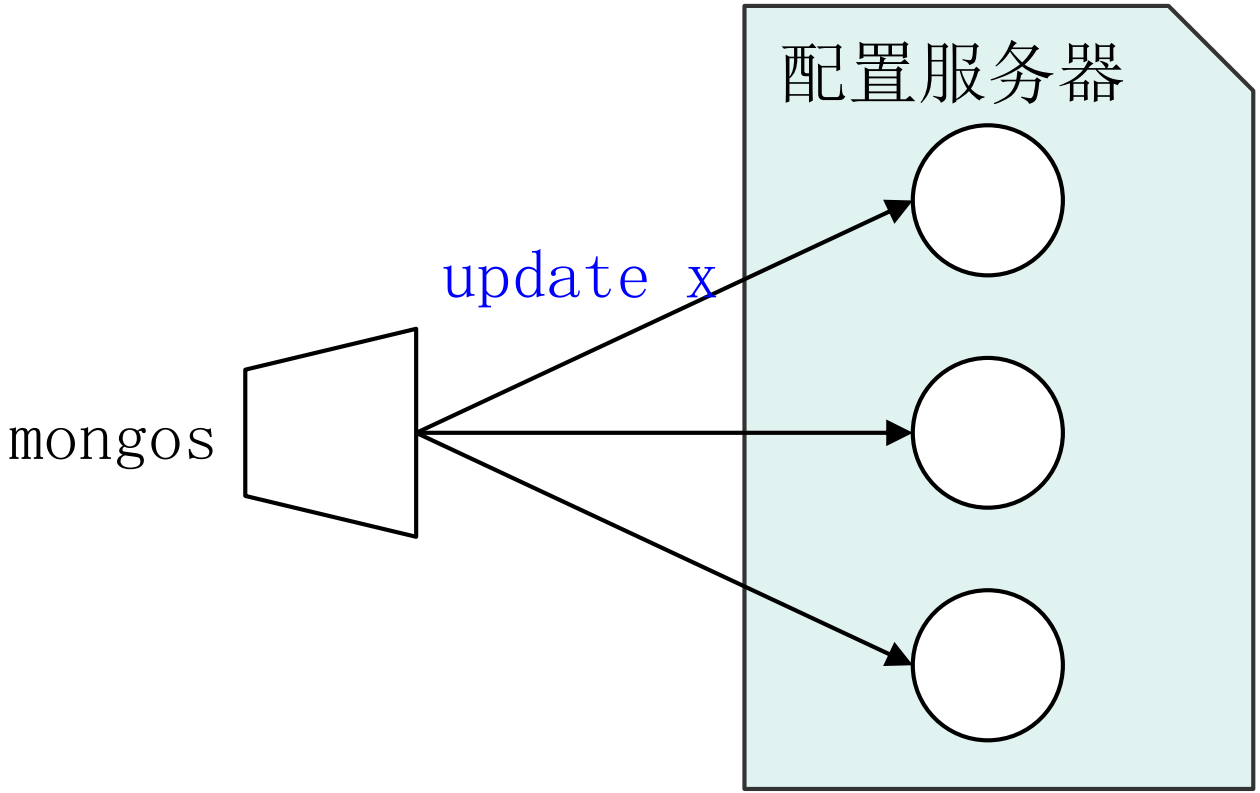

insert({ 金牌: 8 });
@v7
mongos
路由表 v7
sh0
分片0
路由表 v7
sh1
分片1
路由表 v7
配置服务器
路由表 v7

镜像配置服务器

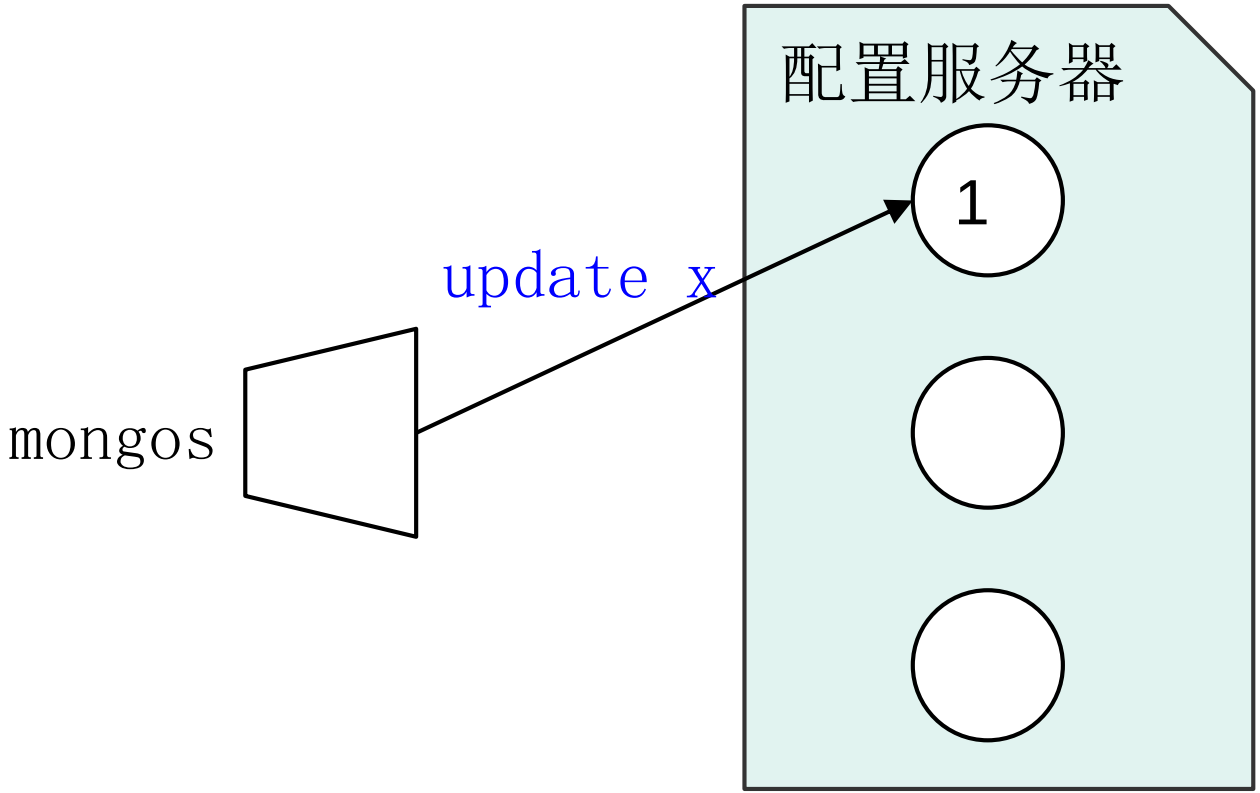




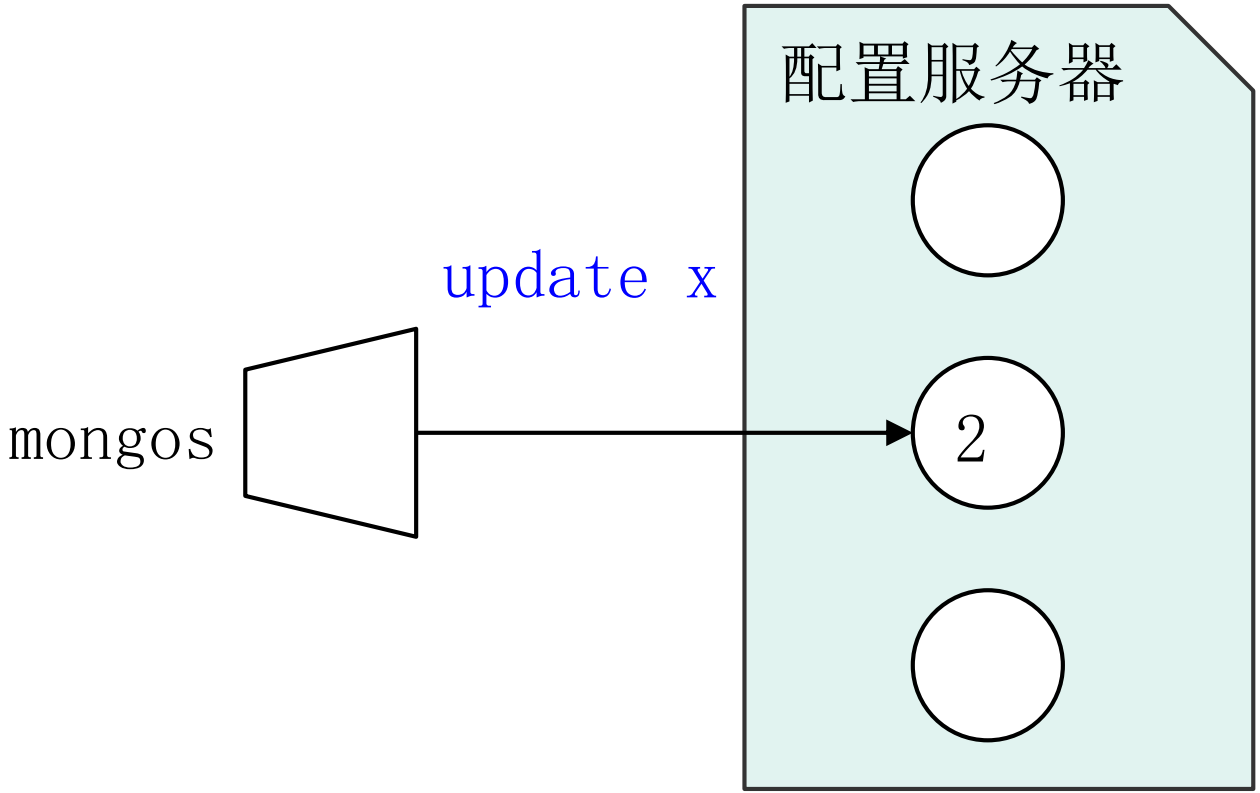
镜像配置服务器



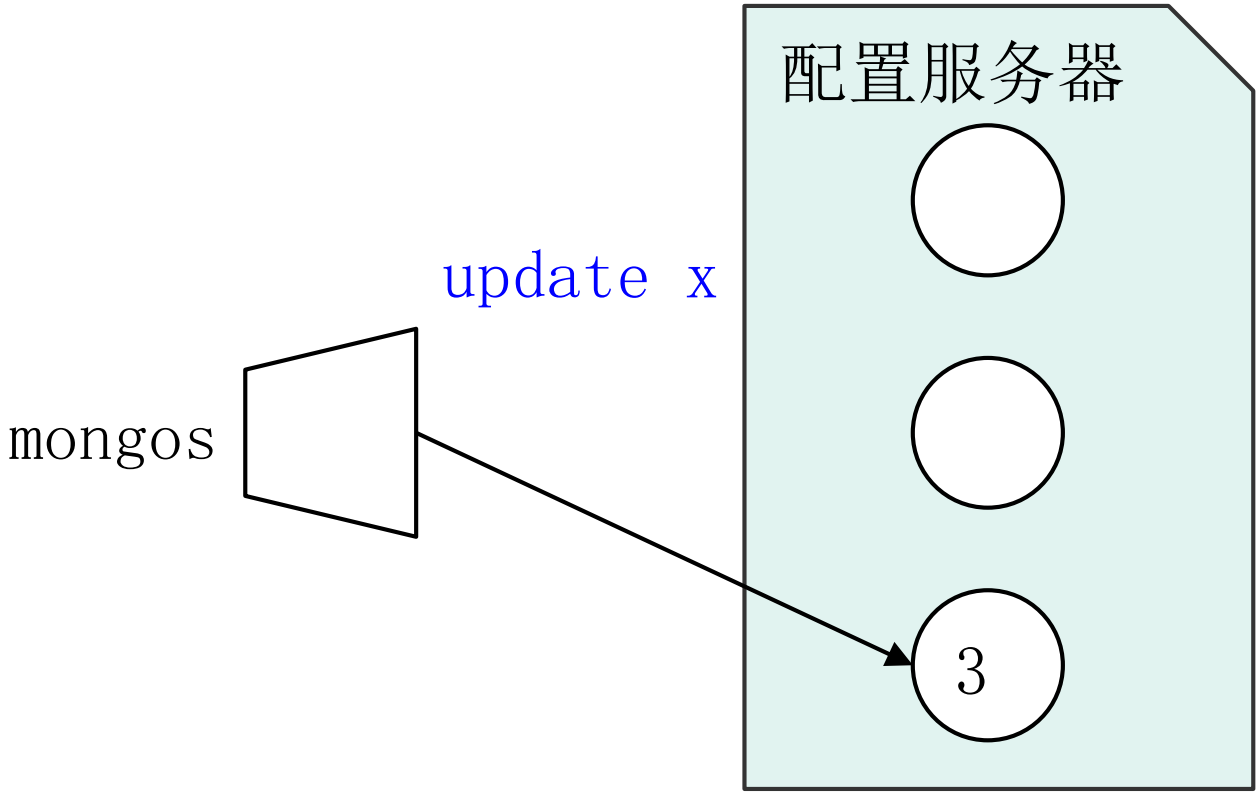
镜像配置服务器



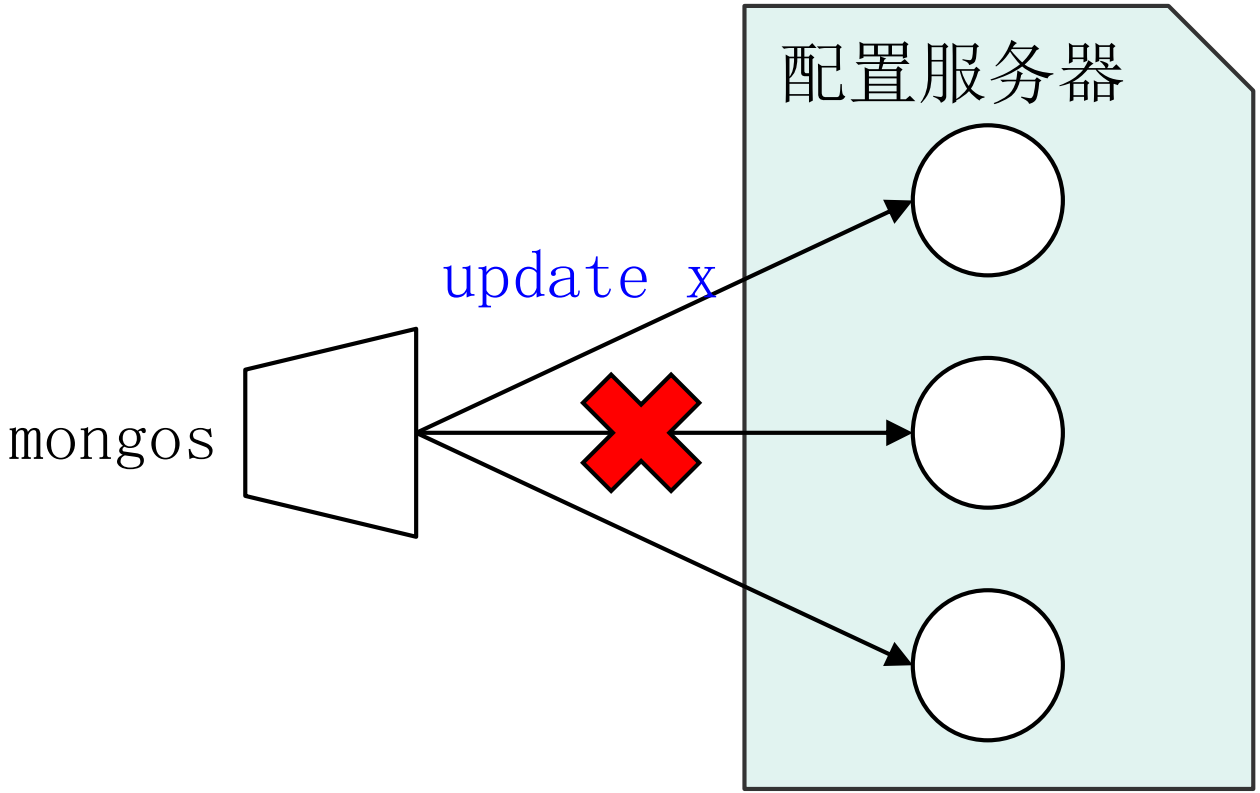
镜像配置服务器



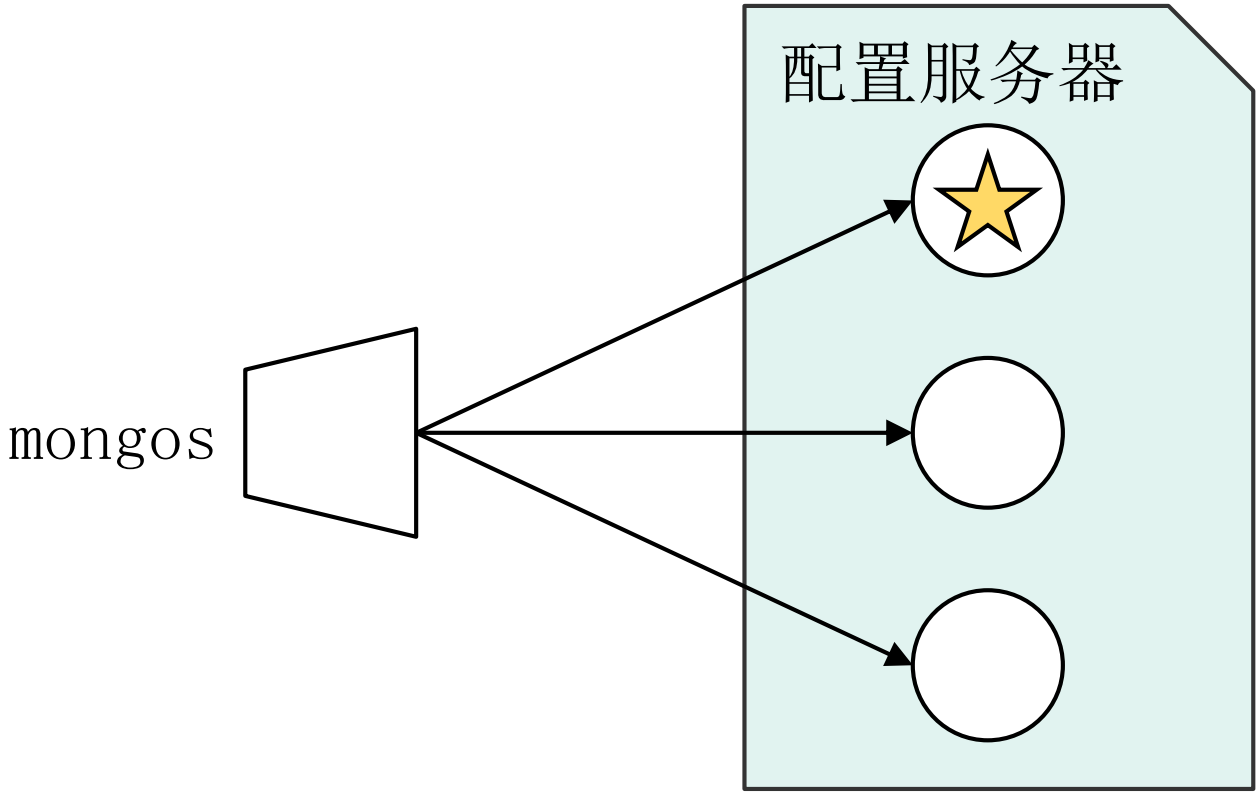
镜像配置服务器



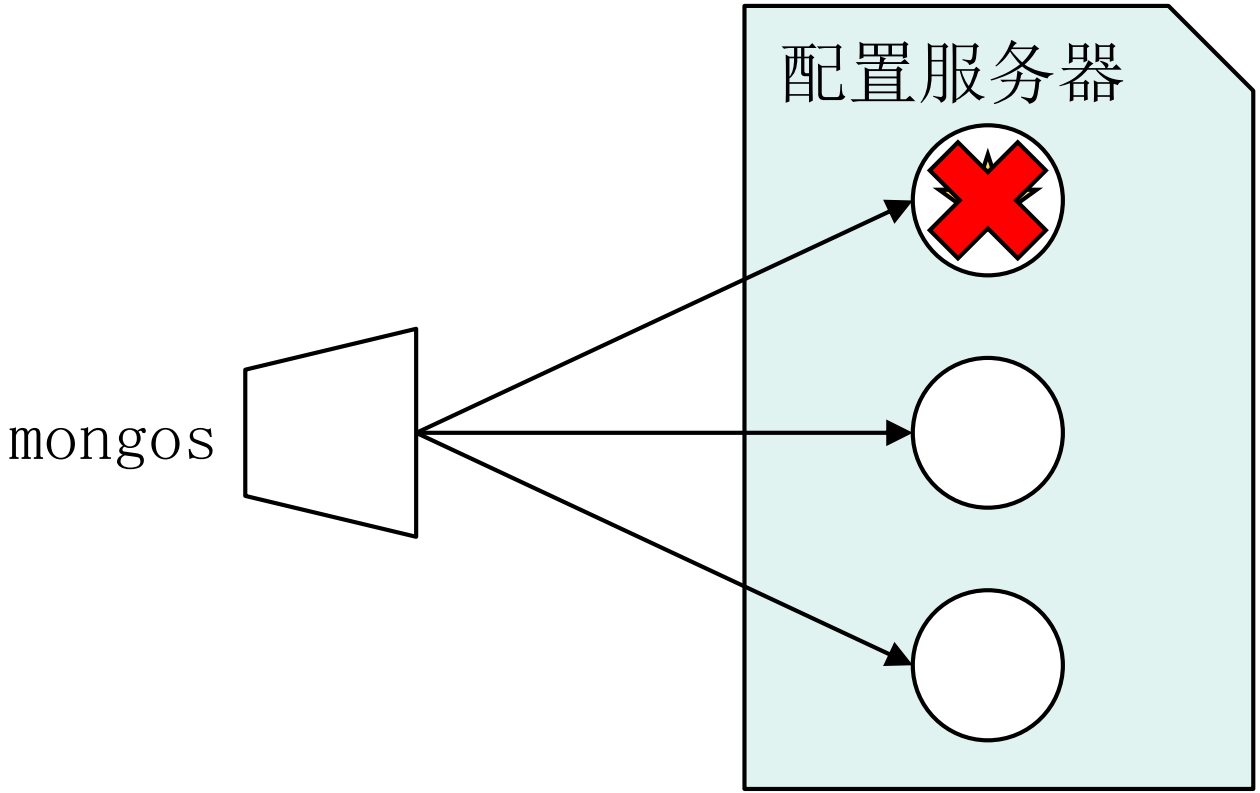
镜像配置服务器



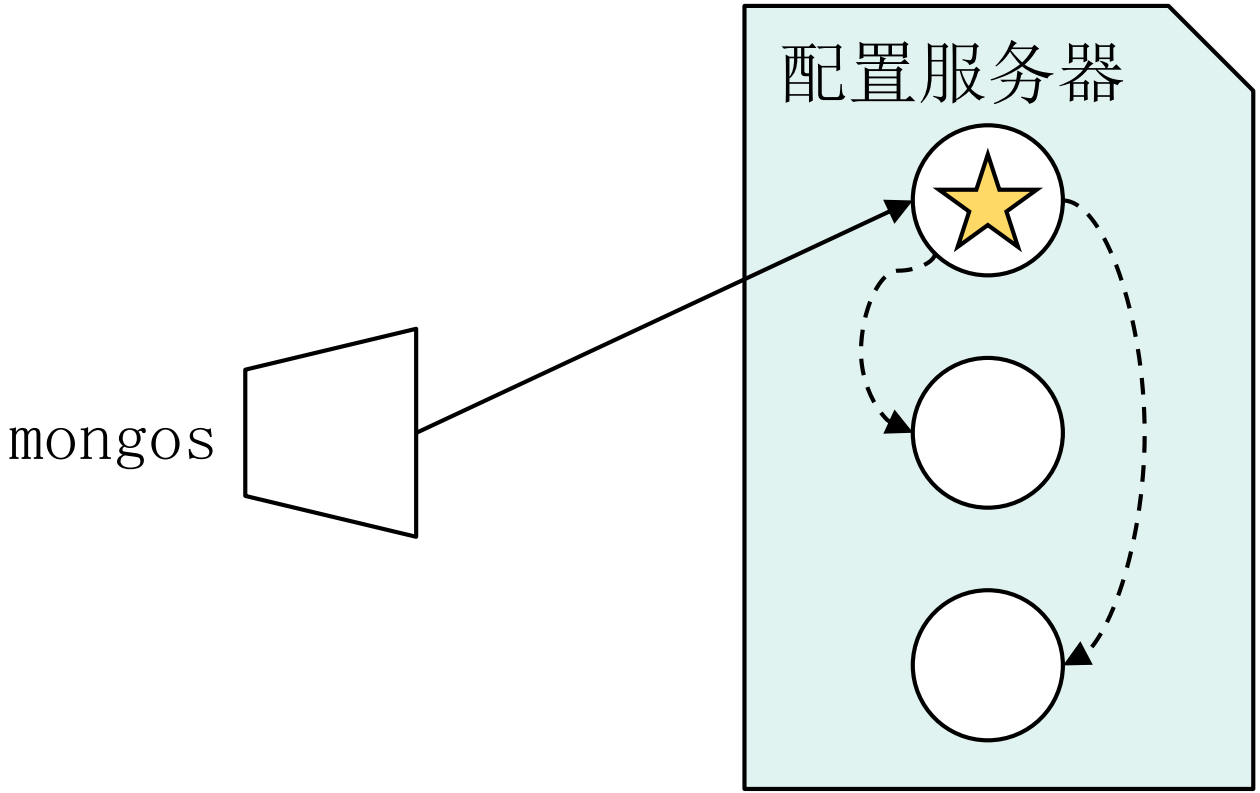
镜像配置服务器



镜像配置服务器



复制集配置服务器



镜像 VS 复制集配置服务器

镜像

失去数据一致性

复制集

镜像 VS 复制集

镜像

失去数据一致性

复制集

单一数据来源

镜像 VS 复制集

镜像

失去数据一致性

失去成员后变成只读

复制集

单一数据来源

镜像 VS 复制集

镜像	复制集
失去数据一致性	单一数据来源
失去成员后变成只读	竞选新主节点

镜像 VS 复制集

镜像

失去数据一致性

失去成员后变成只读

蛋疼的分布式锁

复制集

单一数据来源

竞选新主节点

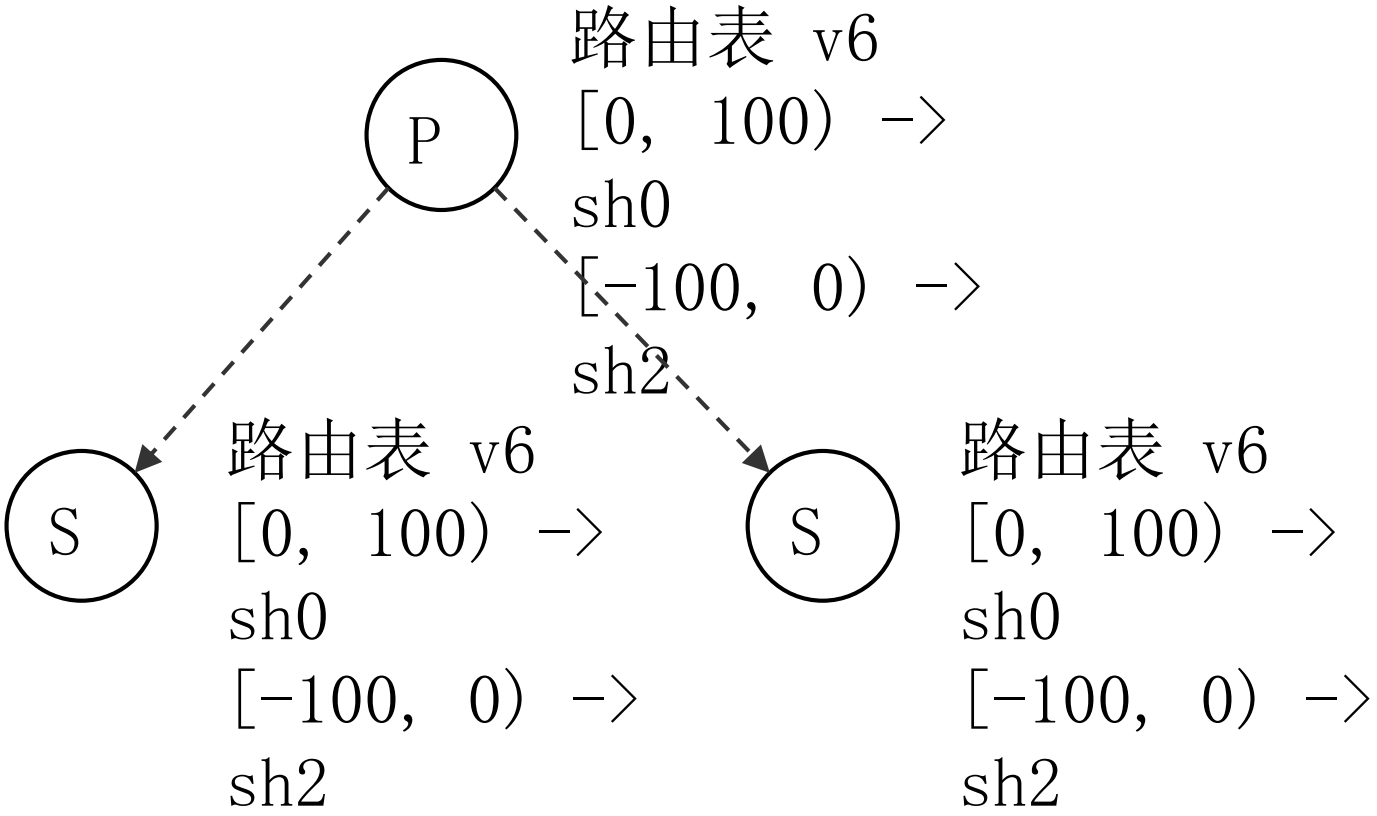
镜像 VS 复制集

镜像	复制集
失去数据一致性	单一数据来源
失去成员后变成只读	竞选新主节点
蛋疼的分布式锁	只需锁住主节点

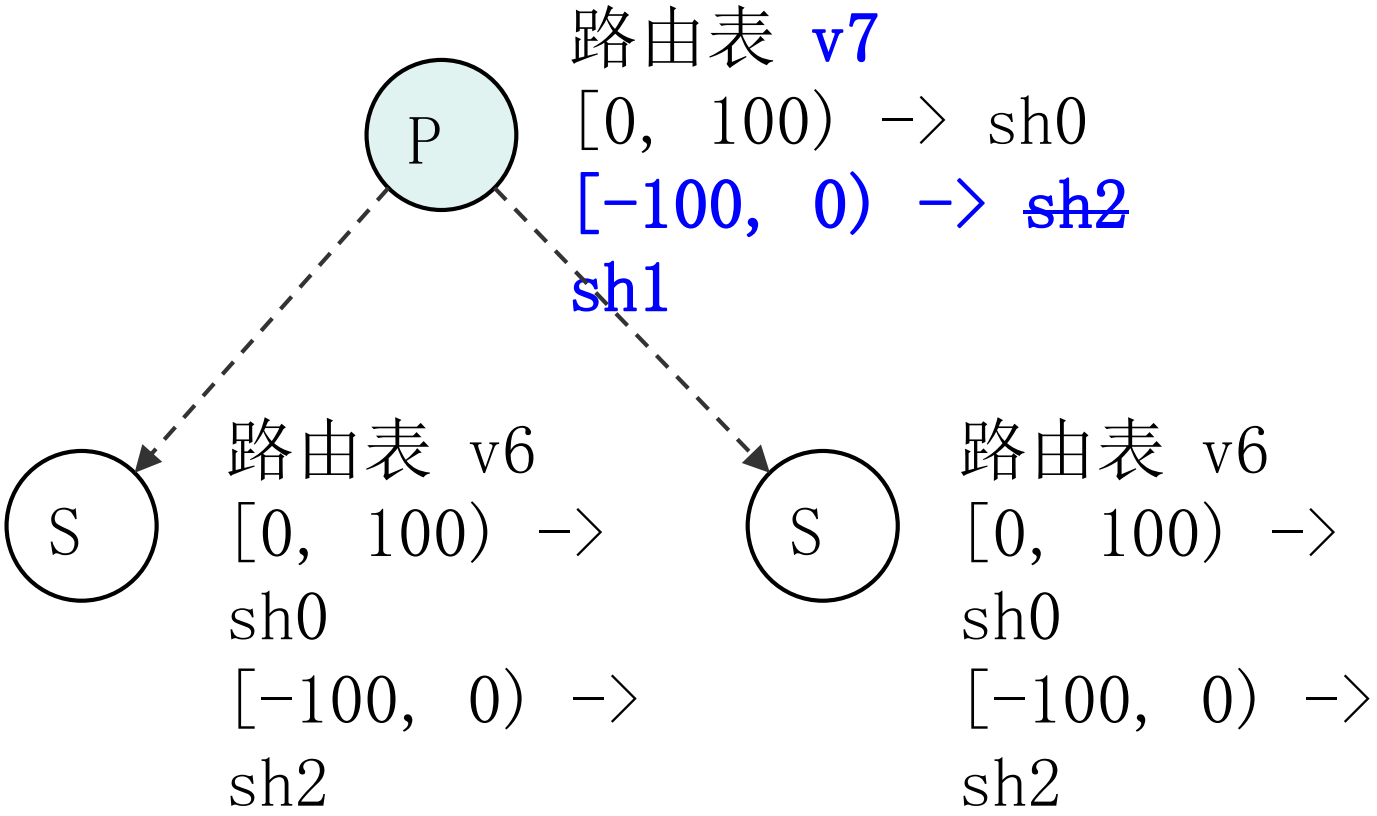
复制集配置服务器面临的挑战

- 已读取的数据可能回滚
- 从节点上的数据会过于陈旧

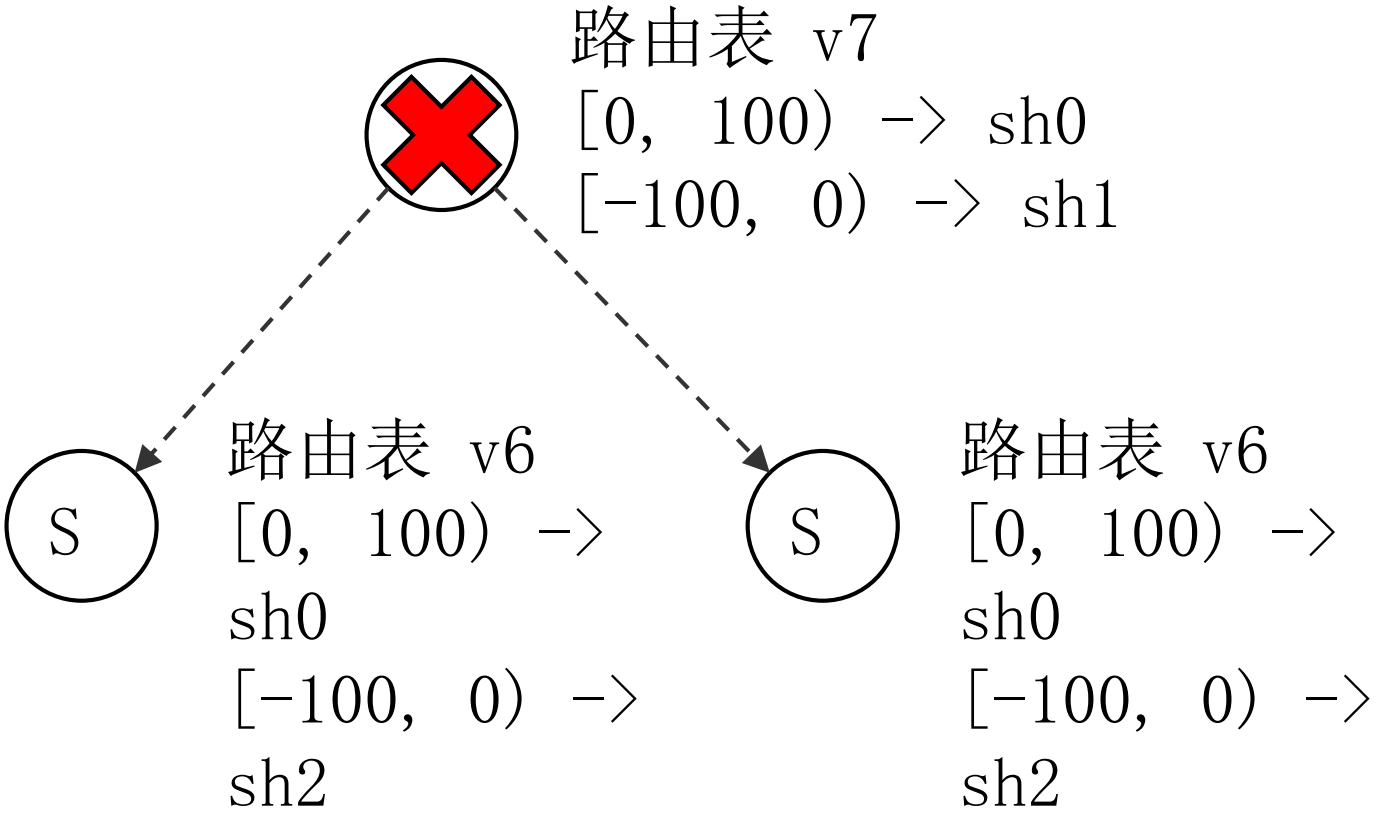
回滚案例


P
路由表 v6
[0, 100) ->
sh0
[-100, 0) ->
sh2
S
路由表 v6
[0, 100) ->
sh0
[-100, 0) ->
sh2
S
路由表 v6
[0, 100) ->
sh0
[-100, 0) ->
sh2

回滚案例

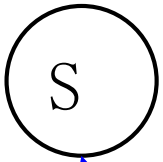

P
路由表 v7
[0, 100) -> sh0
[-100, 0) -> sh2
sh1
S
路由表 v6
[0, 100) ->
sh0
[-100, 0) ->
sh2
S
路由表 v6
[0, 100) ->
sh0
[-100, 0) ->
sh2

回滚案例

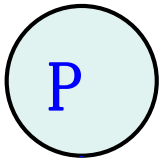

路由表 v7
[0, 100) -> sh0
[-100, 0) -> sh1
S
路由表 v6
[0, 100) -> sh0
[-100, 0) -> sh2
S
路由表 v6
[0, 100) -> sh0
[-100, 0) -> sh2

回滚案例

路由表 v7
[0, 100) -> sh0
[-100, 0) -> sh1



路由表 v6
[0, 100) -> sh0
[-100, 0) -> sh2



路由表 v6
[0, 100) -> sh0
[-100, 0) -> sh2

回滚案例

路由表 v7
[0, 100) -> sh0
[-100, 0) -> sh1

S

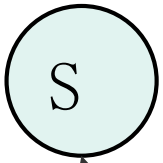
路由表 v6
[0, 100) -> sh0
[-100, 0) -> sh2

P

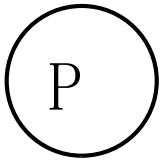
路由表 **v7**
[0, 100) -> ~~sh0~~ sh1
[-100, 0) -> sh2

回滚案例

路由表 v7
[0, 100) -> sh0
[-100, 0) -> sh1

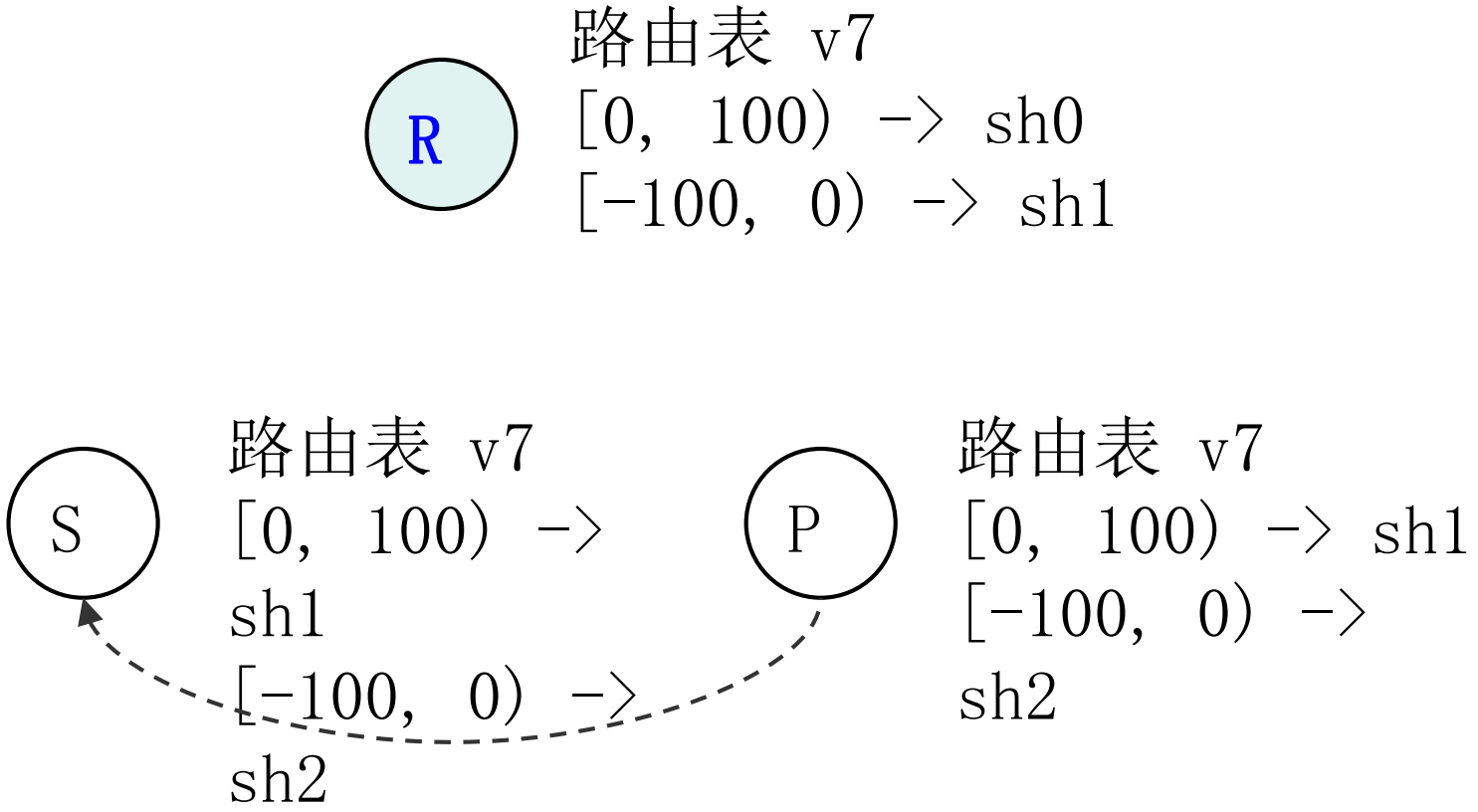


路由表 **v7**
[0, 100) ->
sh1
[-100, 0) ->
sh2

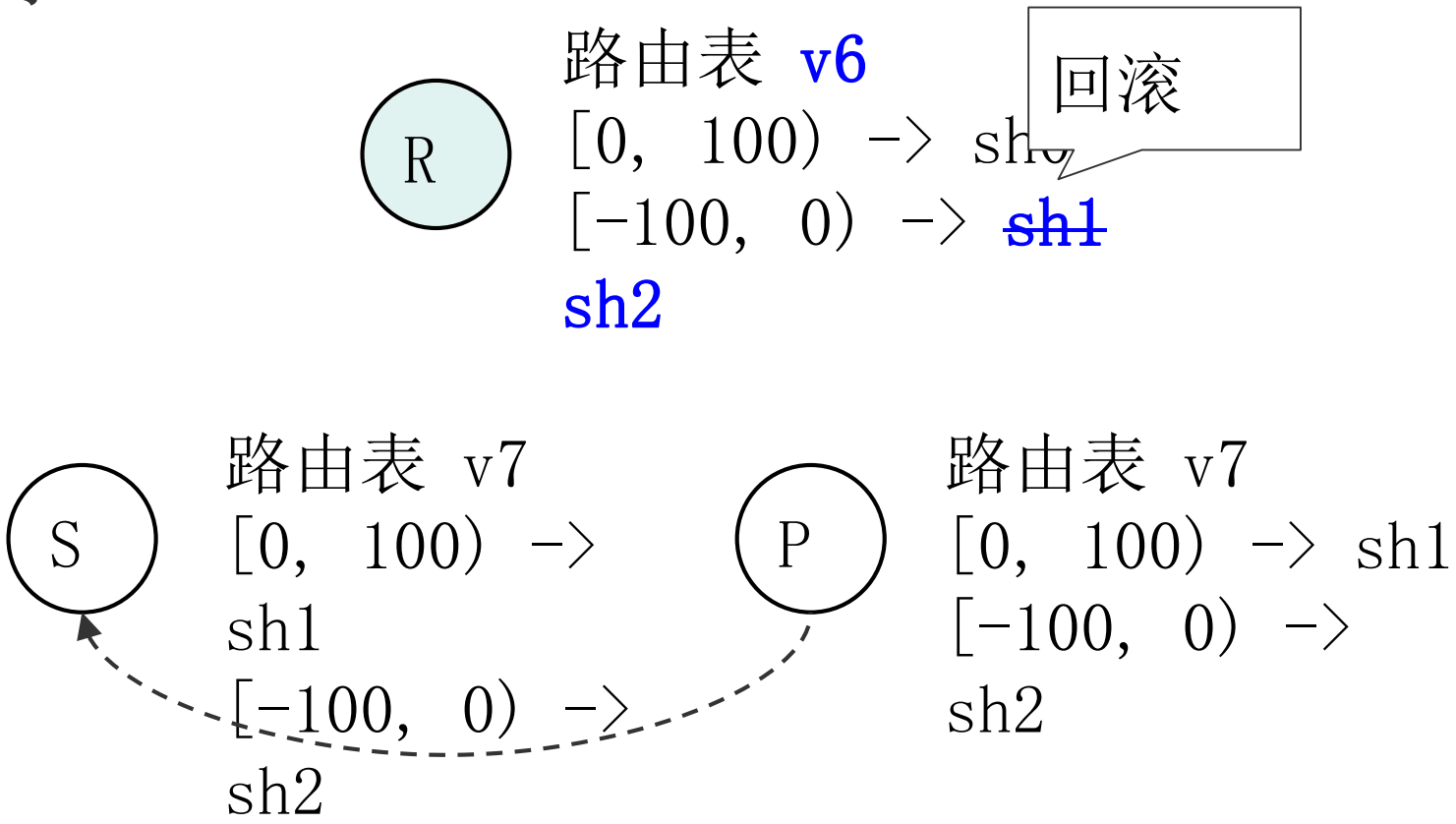


路由表 v7
[0, 100) -> sh1
[-100, 0) ->
sh2

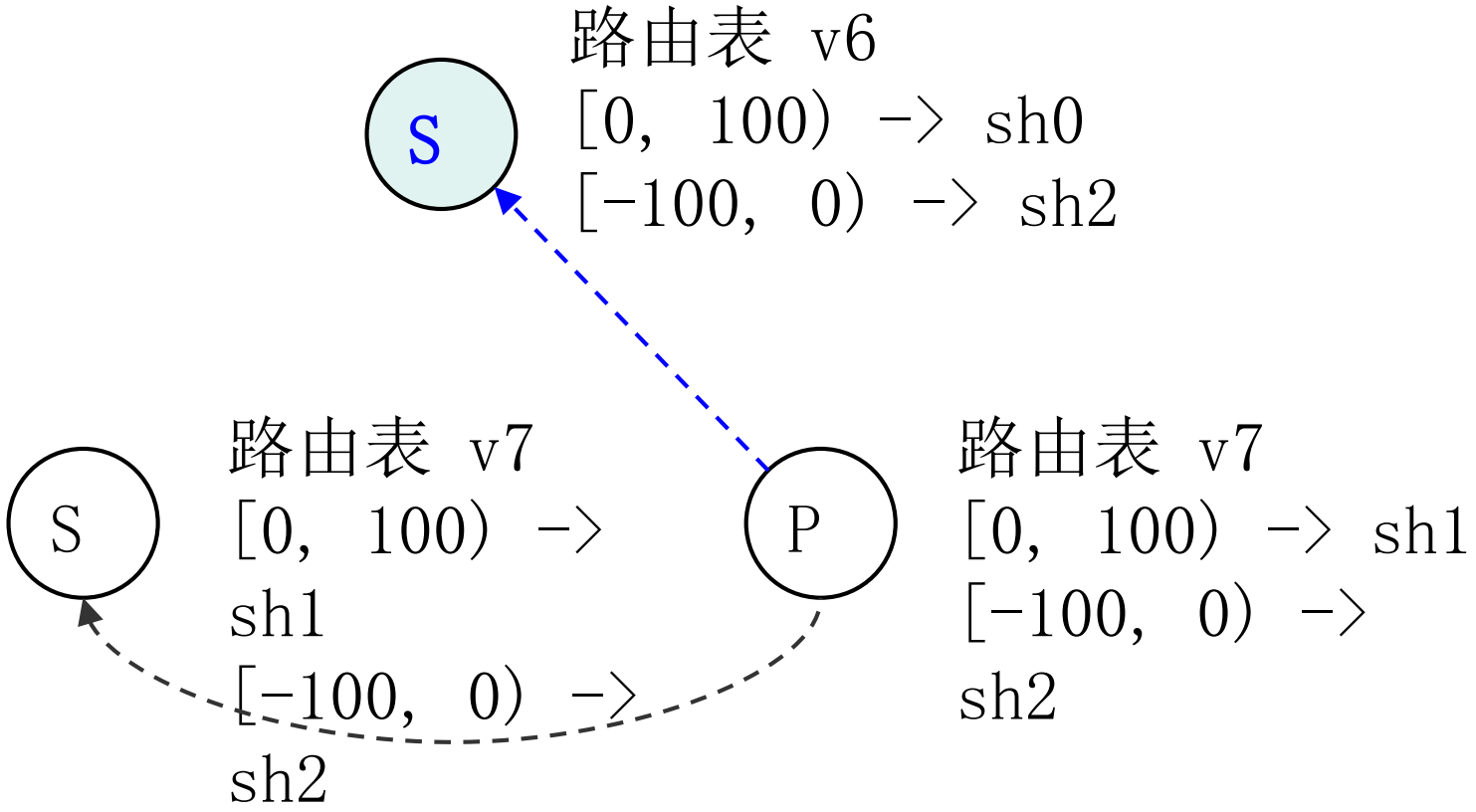
回滚案例


R
路由表 v7
[0, 100) -> sh0
[-100, 0) -> sh1
S
路由表 v7
[0, 100) -> sh1
[-100, 0) -> sh2
P
路由表 v7
[0, 100) -> sh1
[-100, 0) -> sh2

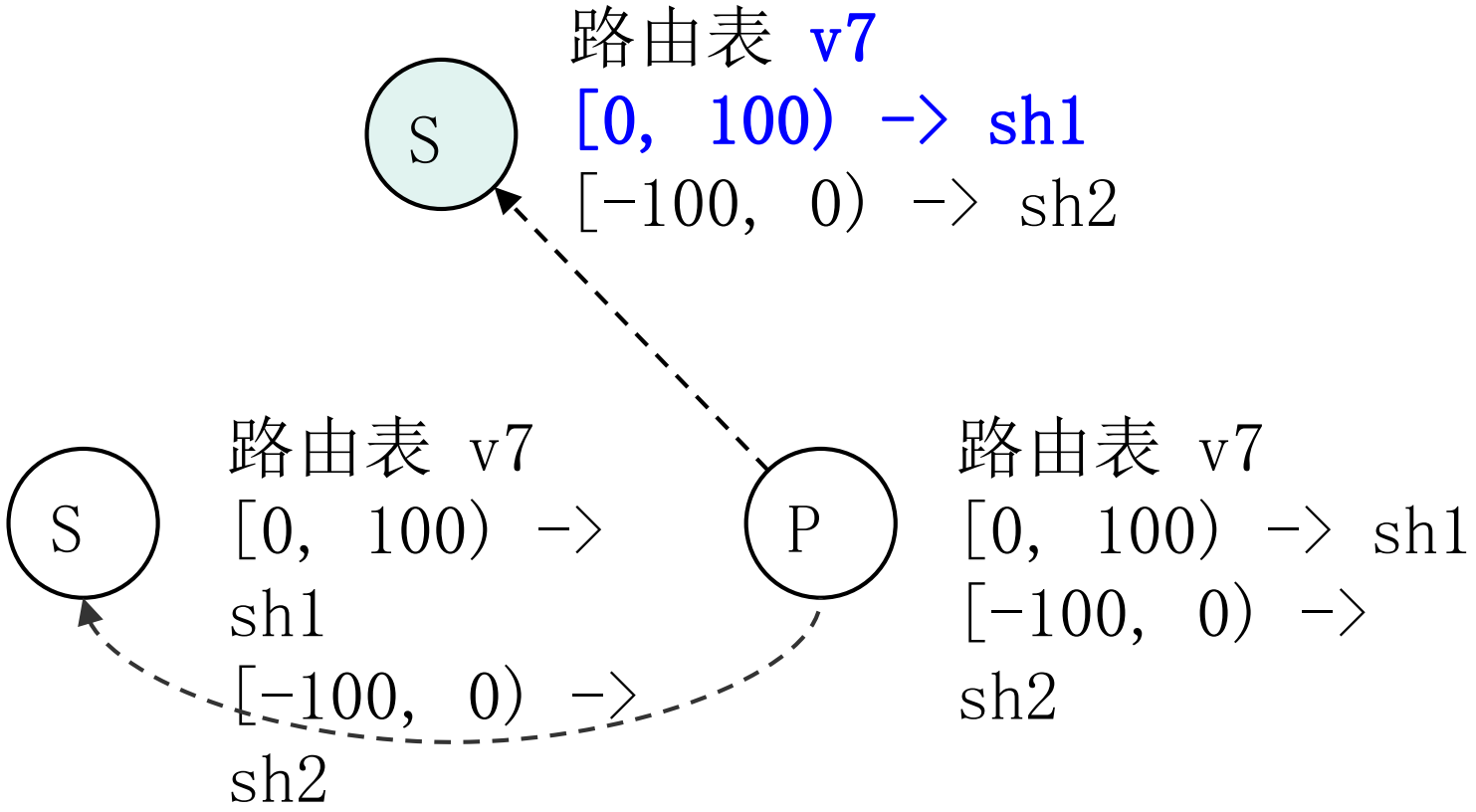
回滚案例


R
路由表 v6
[0, 100) -> sh
[-100, 0) -> sh1
sh2
回滚
S
路由表 v7
[0, 100) ->
sh1
[-100, 0) ->
sh2
P
路由表 v7
[0, 100) -> sh1
[-100, 0) ->
sh2

回滚案例


S
路由表 v6
[0, 100) -> sh0
[-100, 0) -> sh2
S
路由表 v7
[0, 100) -> sh1
[-100, 0) -> sh2
P
路由表 v7
[0, 100) -> sh1
[-100, 0) -> sh2

回滚案例


S
路由表 v7
[0, 100) -> sh1
[-100, 0) -> sh2
S
路由表 v7
[0, 100) -> sh1
[-100, 0) -> sh2
P
路由表 v7
[0, 100) -> sh1
[-100, 0) -> sh2

回滚案例

路由表 v7
[0, 100) -> sh0
[-100, 0) -> sh1

S

路由表 v7
[0, 100) -> sh1
[-100, 0) -> sh2

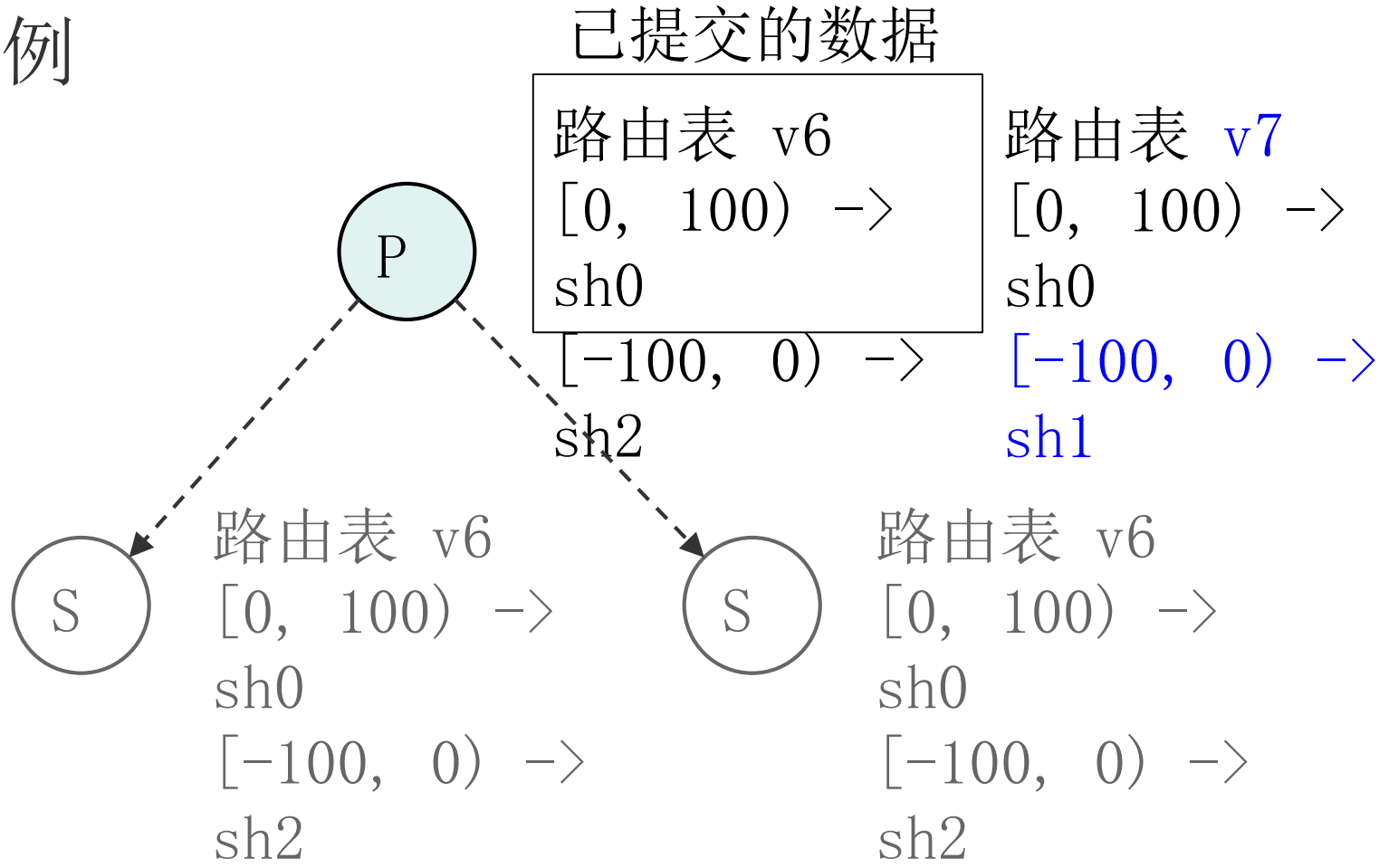
P

路由表 v7
[0, 100) -> sh1
[-100, 0) -> sh2

安全读级别：多数 （新功能）

```
db.runCommand({
    find: ‘foo’,
    filter: { 金牌: 1 },
    readConcern: { level: ‘majority’ }
});
```

回滚案例


已提交的数据
路由表 v6
[0, 100) ->
sh0
[-100, 0) ->
sh2
路由表 v7
[0, 100) ->
sh0
[-100, 0) ->
sh1
P
S
路由表 v6
[0, 100) ->
sh0
[-100, 0) ->
sh2
S
路由表 v6
[0, 100) ->
sh0
[-100, 0) ->
sh2

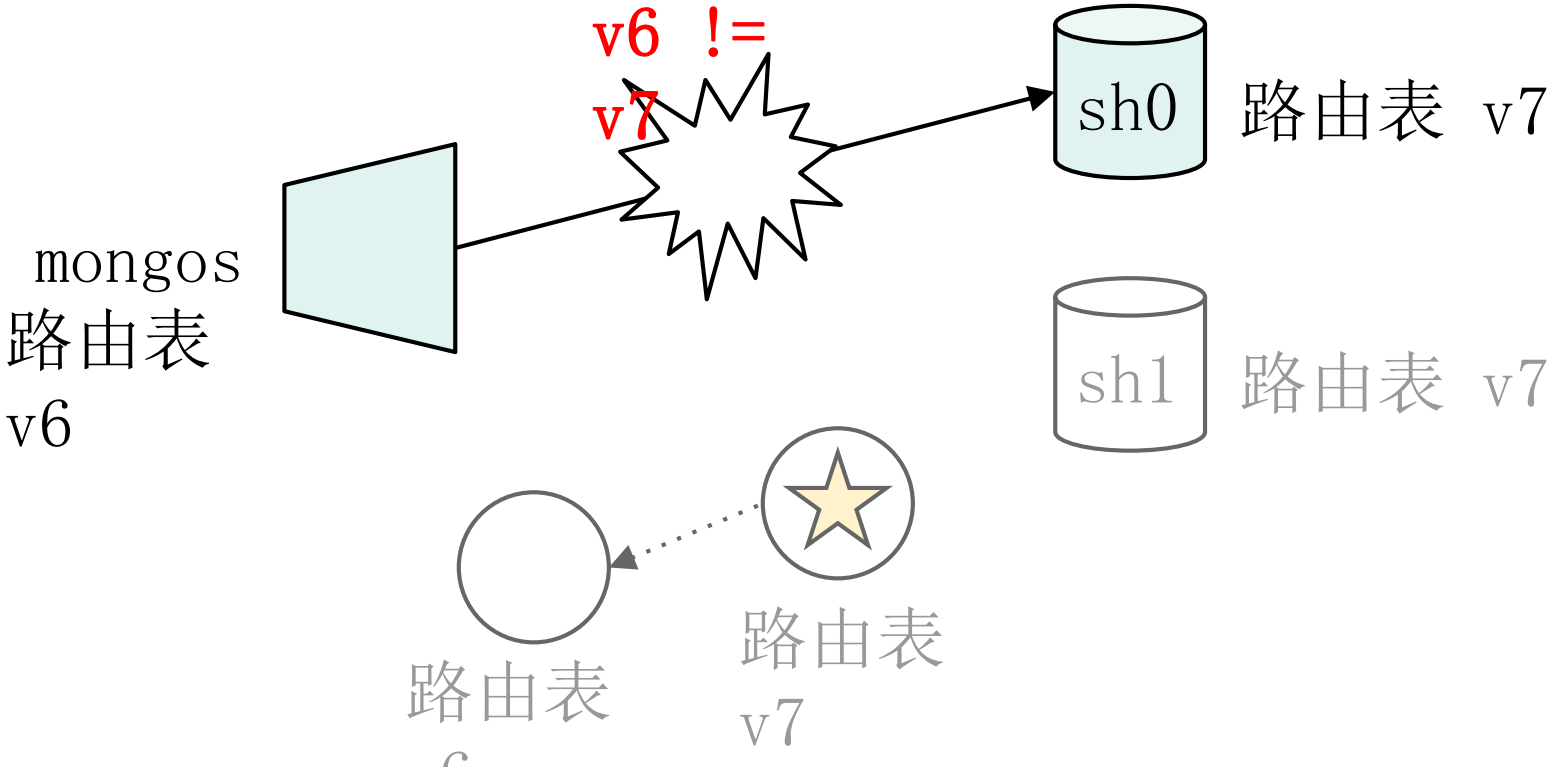
复制集配置服务器面临的挑战

- ~~已读取的数据可能回滚~~

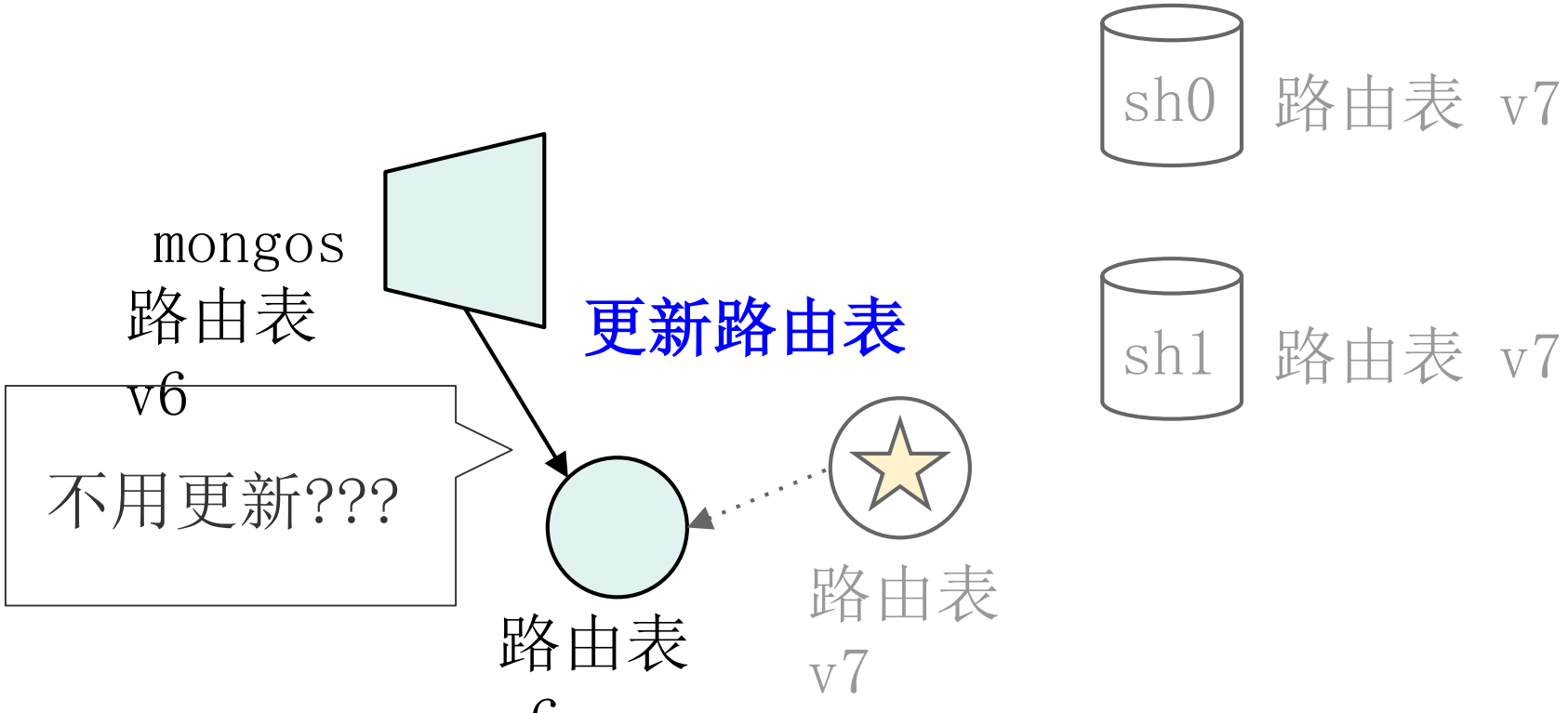
安全读级别：多数

- 从节点上的数据会过于陈旧

读取陈旧数据案例



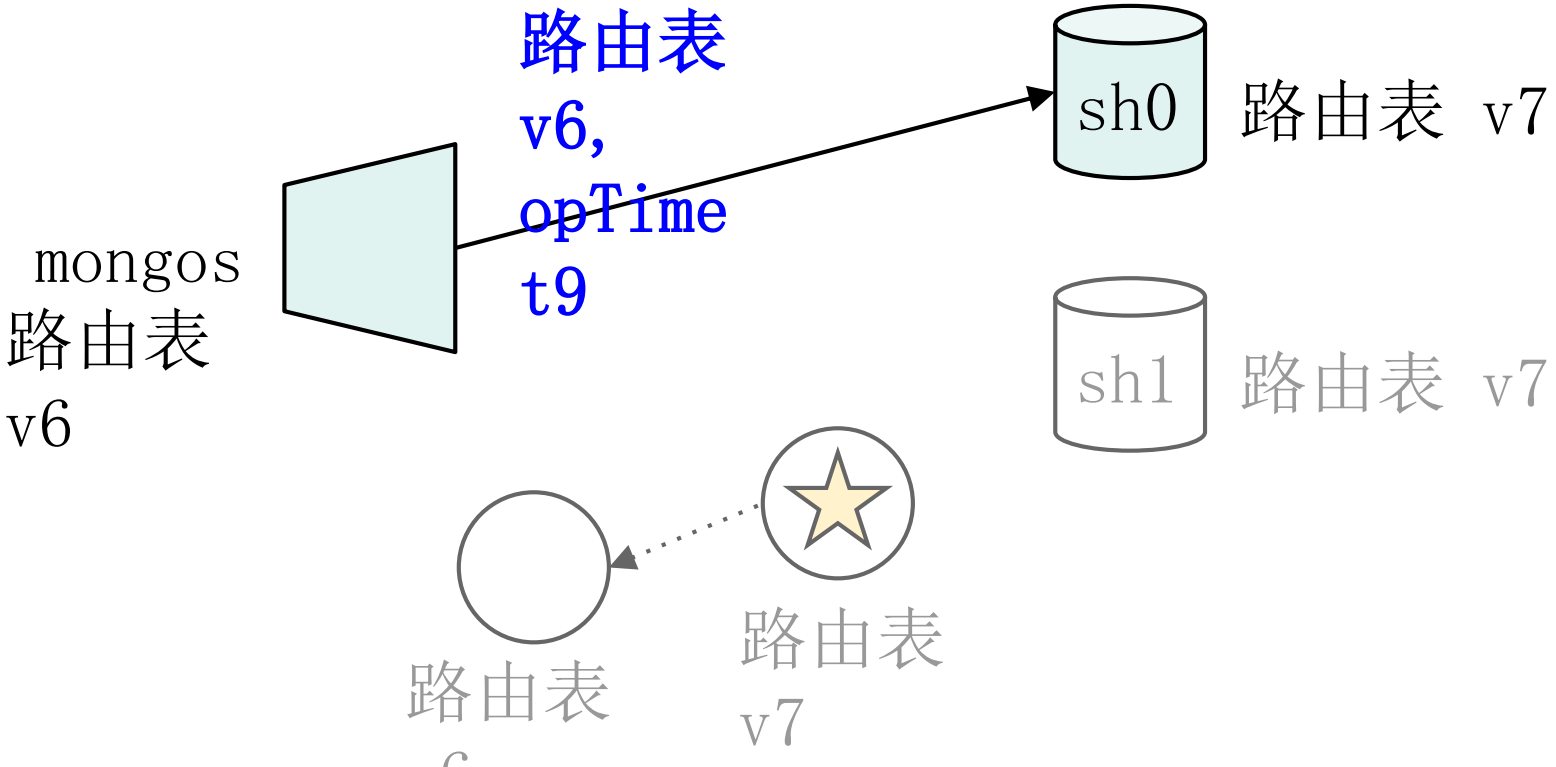
读取陈旧数据案例



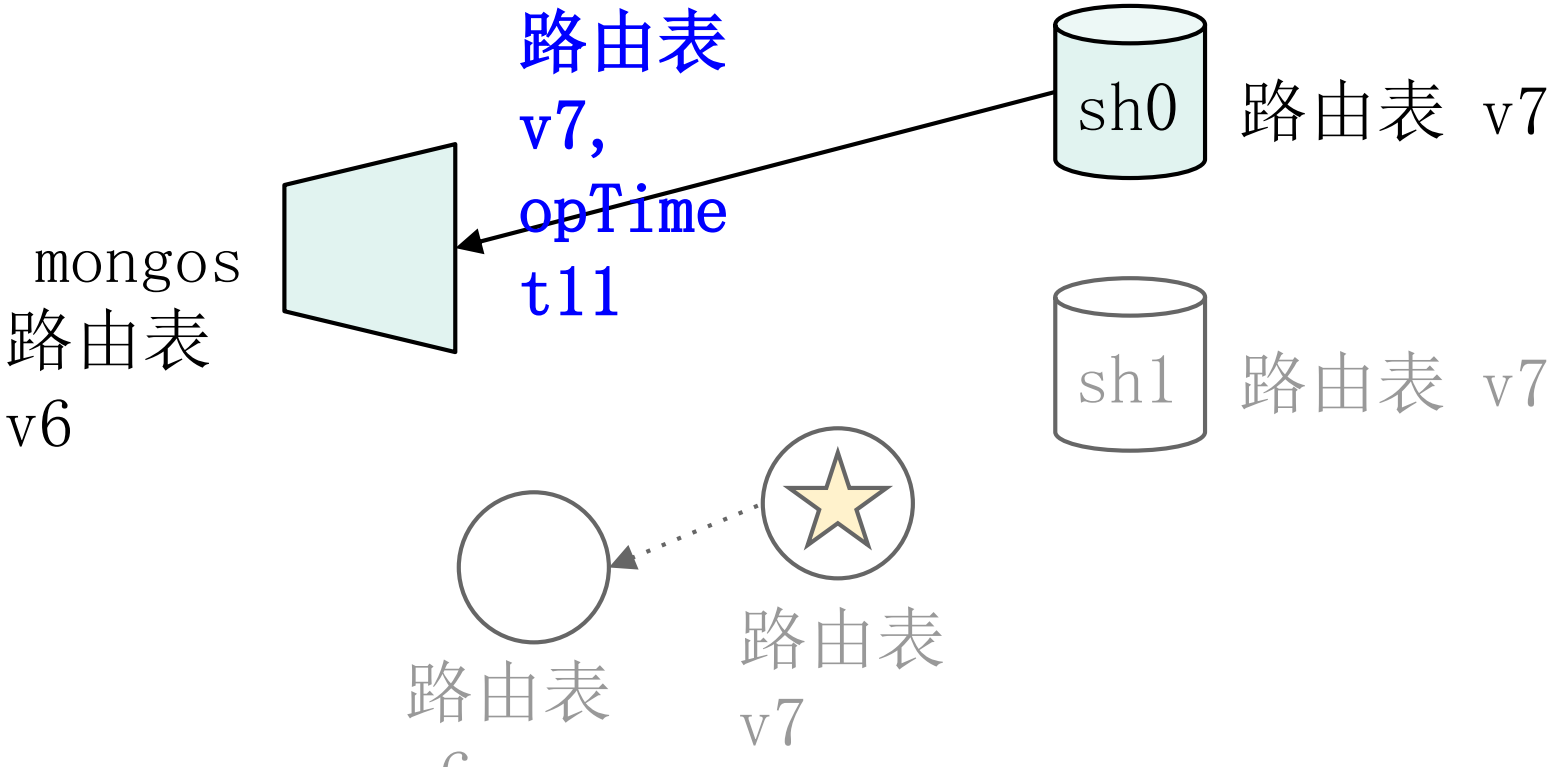
新功能（仅限内部使用）

```
db.runCommand({
    find: ‘foo’,
    filter: { 金牌: 1 },
    readConcern: {
        level: ‘majority’,
        afterOpTime: <opTime>
    }});
```

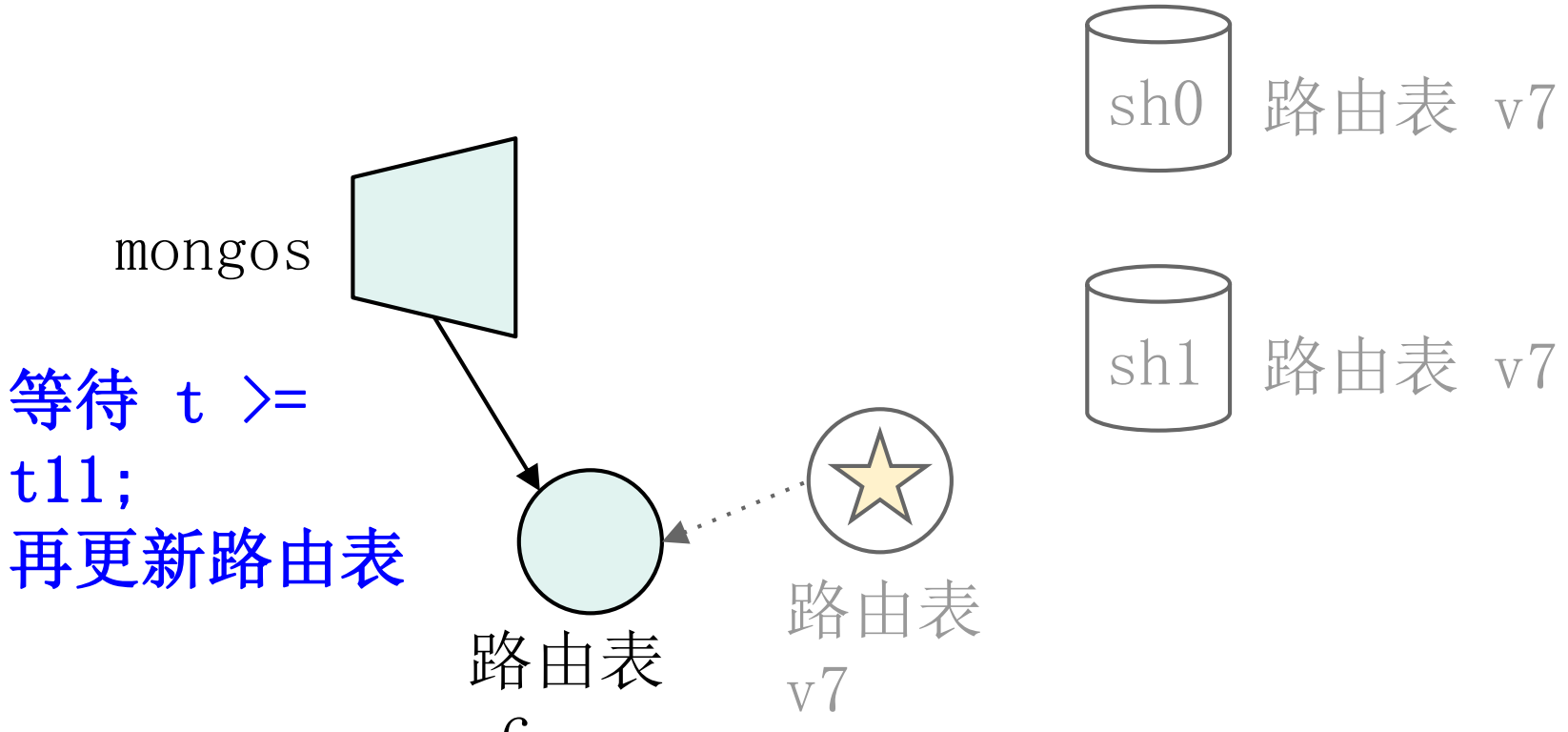
读取晚于OpTime



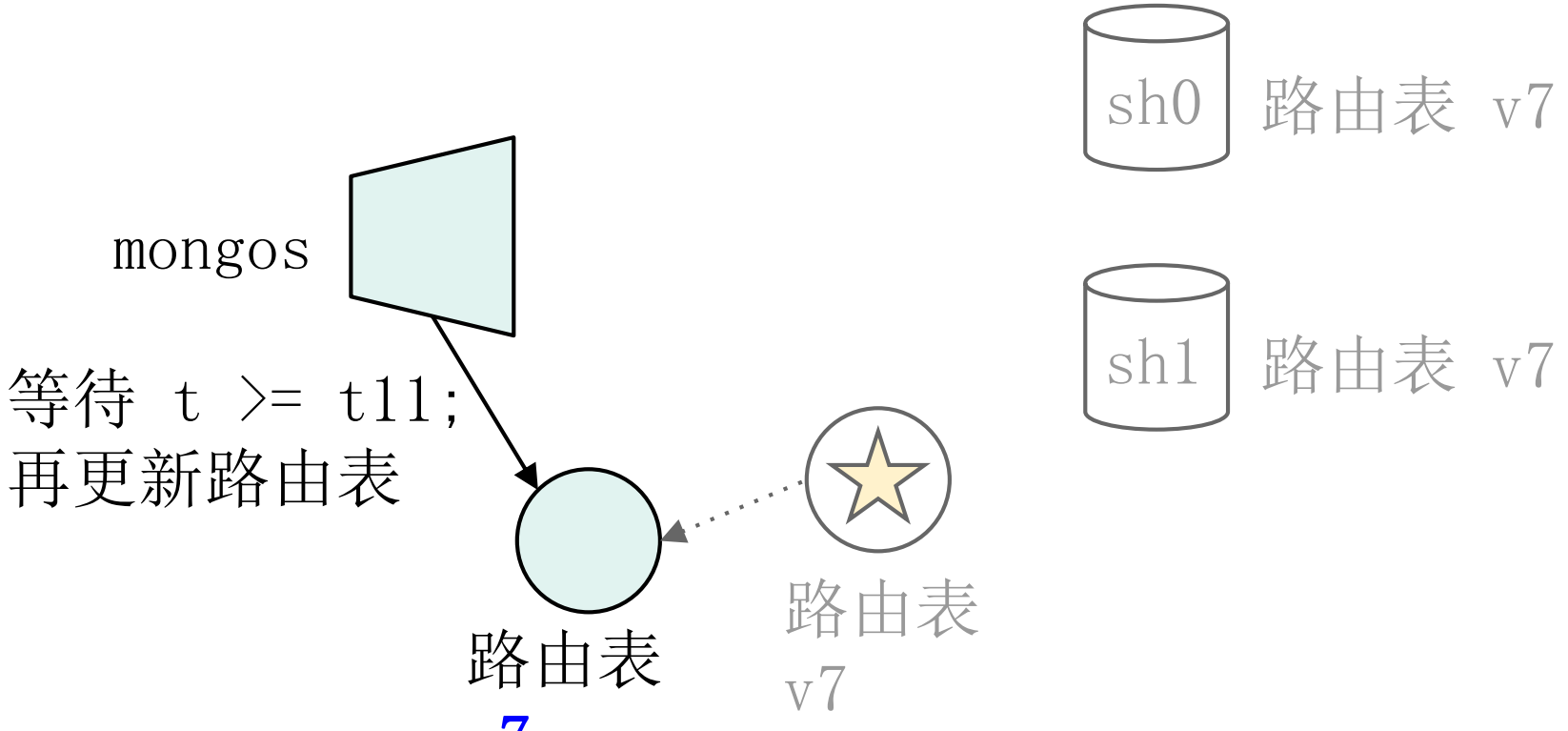
读取晚于OpTime



读取晚于OpTime



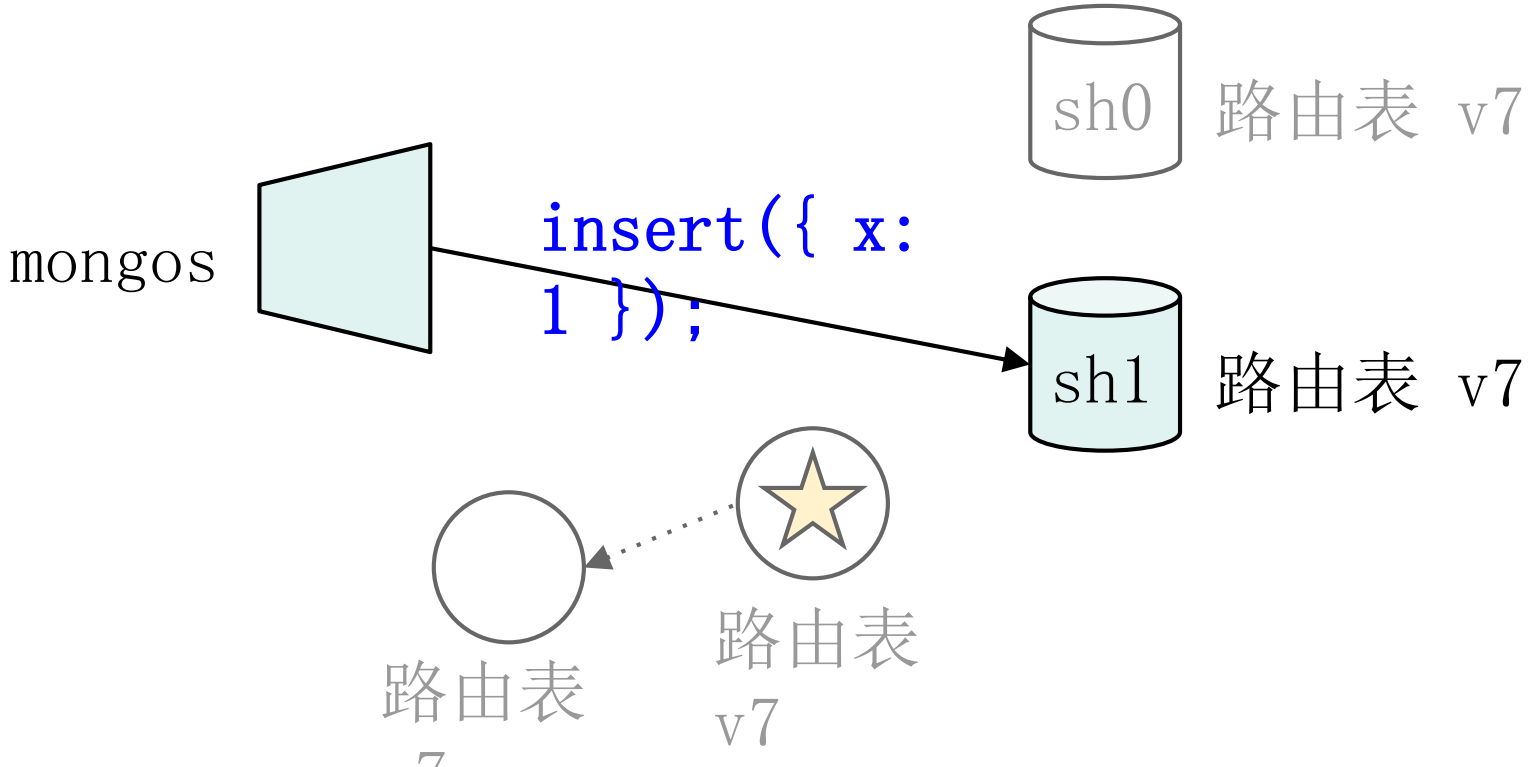
读取晚于OpTime



分片版本协议

版本	最小值	最大值	所在分片
1	MinKey	-200	分片0
2	-200	-100	分片2
3	-100	0	分片2
7	0	100	分片1
5	100	200	分片1
6	200	MaxKey	分片0

读取晚于OpTime



复制集配置服务器面临的挑战

- ~~已读取的数据可能回滚~~

安全读级别：多数

- 从节点上的数据会过于陈旧

读取晚于optime

MongoDB 3.2 中新功能

- 配置服务器可以使用复制集
  - V3.4中将强制使用复制集配置服务器
- 新功能：读取安全级别

展望未来

- 开放读取晚于optime

举手问题环节