架构 始于需求

基于应用场景的NoSQL选型与实践

@iammutex

HTTP://NoSQLFan.COM

yun wei

运维

kai fa

开发

我想要一个牛x的数据库：性能高，还无限扩展

这玩意不存在！

我希望开发都只做主键查询

这种事不存在！

需求分析

- 数据量：记录大小，总量，增量，热数据
- 访问特点：读写比，并发，吞吐，冷热比
- 其它需求：统计，实时，一致性
- $￥£：机器成本，人力（开发、维护）

an li

案例

图片-需求

- 单条长，不可变，最近最热
- 增量写，主键查
- 元数据：大小，尺寸，日期

图片-MongoDB

- MongoDB定时fsync，顺序IO
- 元数据和文件内容不必分两套
- 同步方案完善，Replica Sets

GridFS or Doc

- GridFS双层索引
- 速度慢
- 内存占用大
- oplog多

GridFS or Doc

```
mongod --syncdelay=0 --quiet ...
```

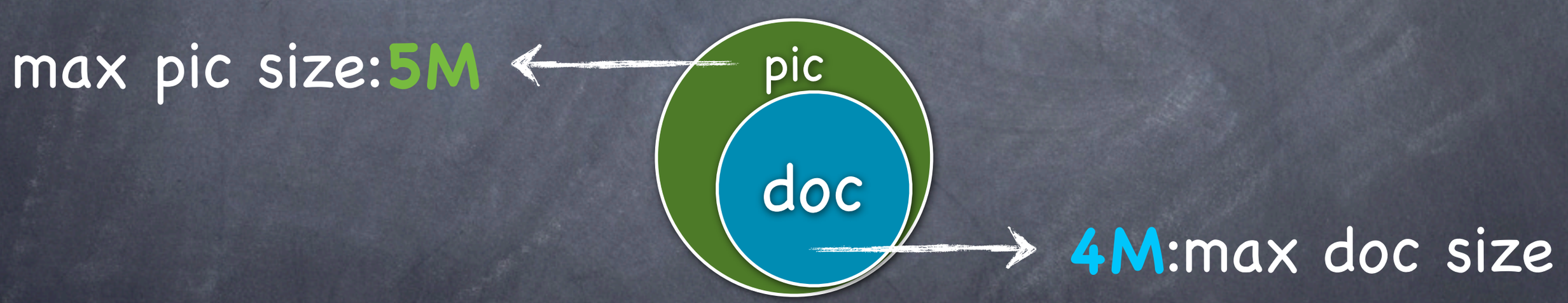
数据大小	GridFS	Doc
1k	0.24s	0.01s
10k	0.28s	0.03s
100k	0.71s	0.21s
1m	25s	15s

我们选择了GridFS

WHY?

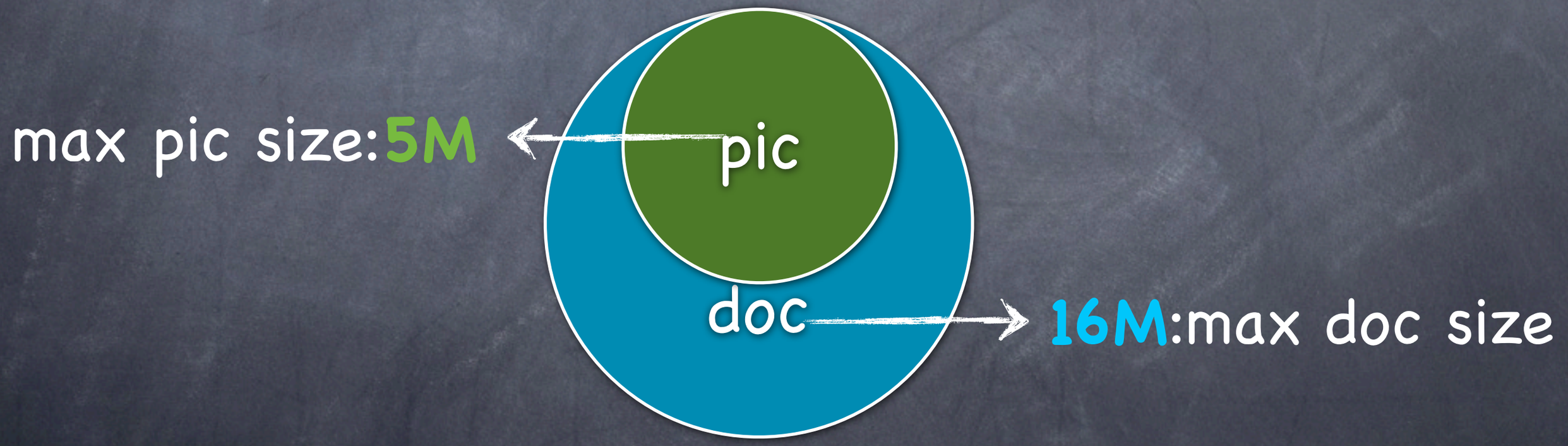
MongoDB v1.6

1.6版本，单条记录最大4M



MongoDB v1.8

1.8版本，单条记录最大16M

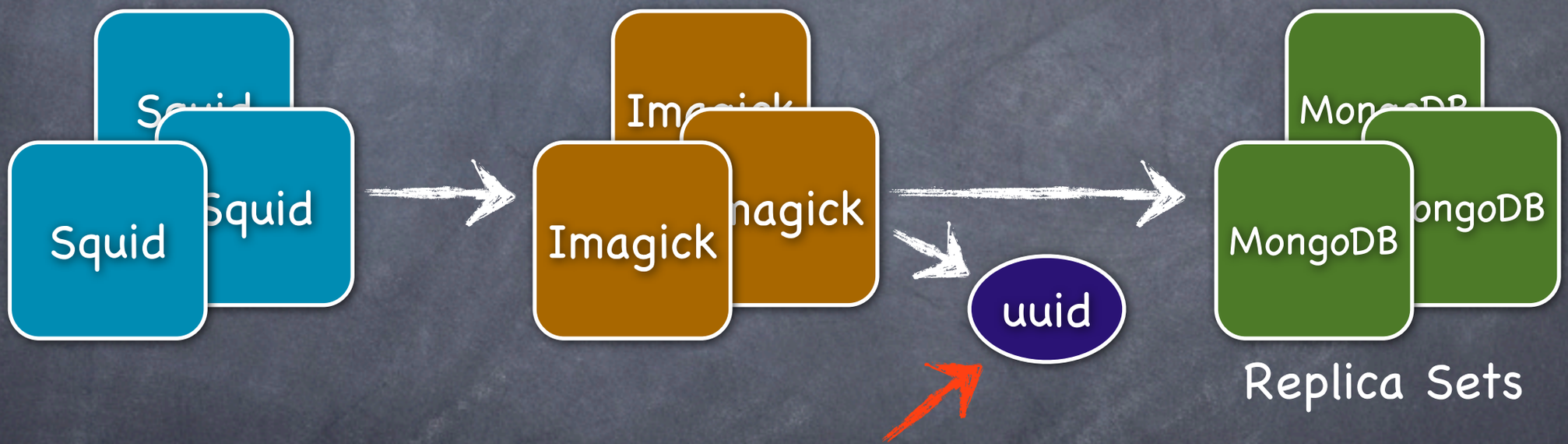


记录格式

```
{
  _id:100000321
  pic:********,
  tm:1319964521,
  uid:1023846261,
  ...
  x:'1',
}
```

```
{pic:**********,tm:1319964521,uid:1023846261,...}
{pic:*******,tm:1319964521,uid:10131231678,...}
{pic:*****,tm:1319964522,uid:1023945752,...}
{pic:*********,tm:1319964522,uid:1023811261,...,x:'1'}
{pic:******,tm:1319964525,uid:1032810261,...}
{pic:**************,tm:1319964526,uid:1084162361,...}
{pic:*****,tm:1319964527,uid:1018426261,...}
{pic:***********,tm:1319964527,uid:1023841621,...}
{pic:*******,tm:1319964527,uid:1023846261,...}
```

半自动化分片


Squid
Squid
Imagick
Imagick
uuid
MongoDB
MongoDB
Replica Sets

播放记录

- 写多读少，读写比 < 1：10
- 更新频繁：数据量小，oplog大
- 用户数据分散：浪费内存，多次seek

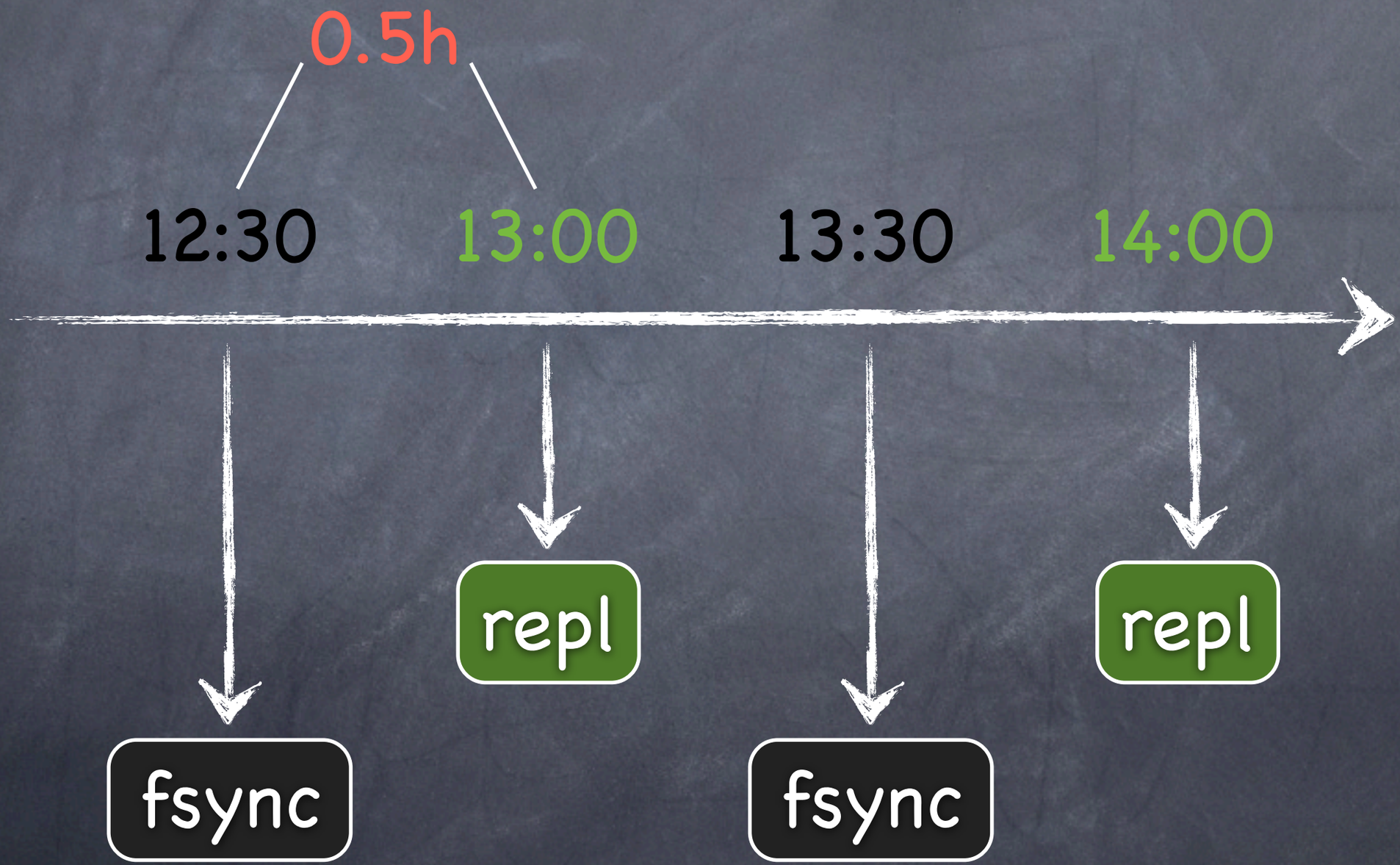
播放记录:优化

- 写多读少： syncdelay == 1h
- oplog： no oplog + ensureIndex({mt:-1})
- 紧凑存储： uid:{tvid1:time,tvid2:time,...}

播放记录:优化

- 写多读少：syncdelay == 1h
- **oplog：no oplog + ensureIndex({mt:-1})**
- 紧凑存储：uid:{tvid1:time,tvid2:time,...}

安全策略



播放记录:优化

- 写多读少：syncdelay == 1h
- oplog：no oplog + ensureIndex({mt:-1})
- 紧凑存储：**uid:{tvid1:time,tvid2:time,....}**

吃内存!!

4k

{uid:100773310,tvid:3726}
{uid:100230310,tvid:11932}
{uid:102212328,tvid:92801}
{uid:193742657,tvid:32111}
{uid:183746222,tvid:37261}
{uid:100230310,tvid:33211}
{uid:110283746,tvid:34678}
{uid:102384744,tvid:8375}
{uid:100230310,tvid:9928}
{uid:139446572,tvid:11029}

吃内存??

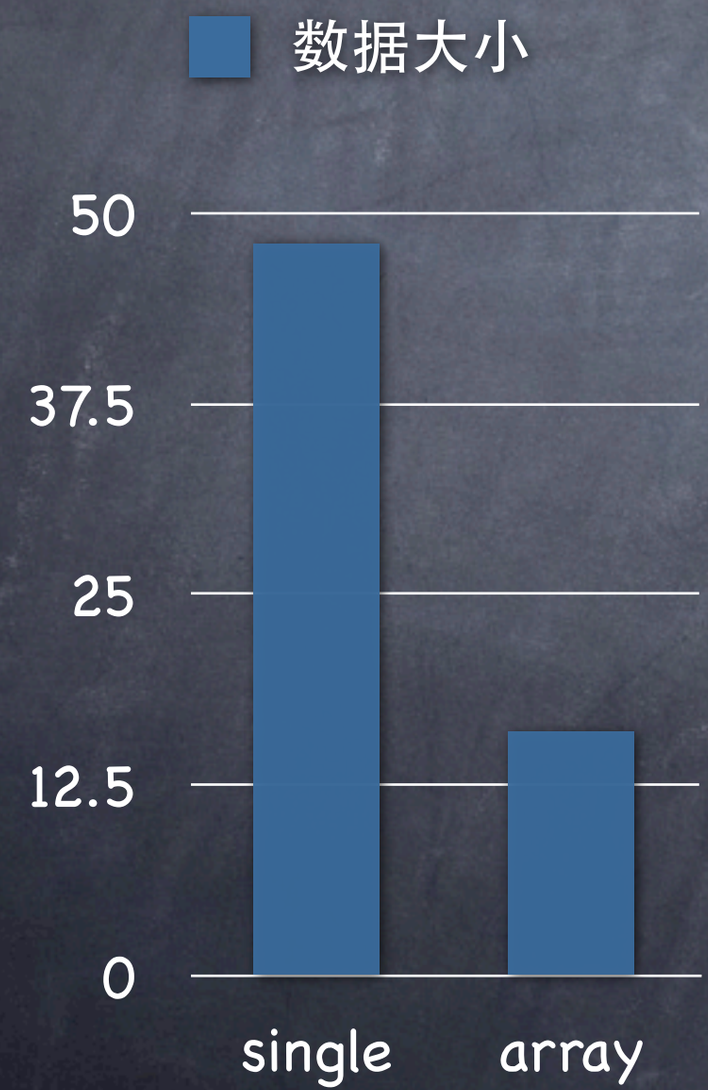
4k

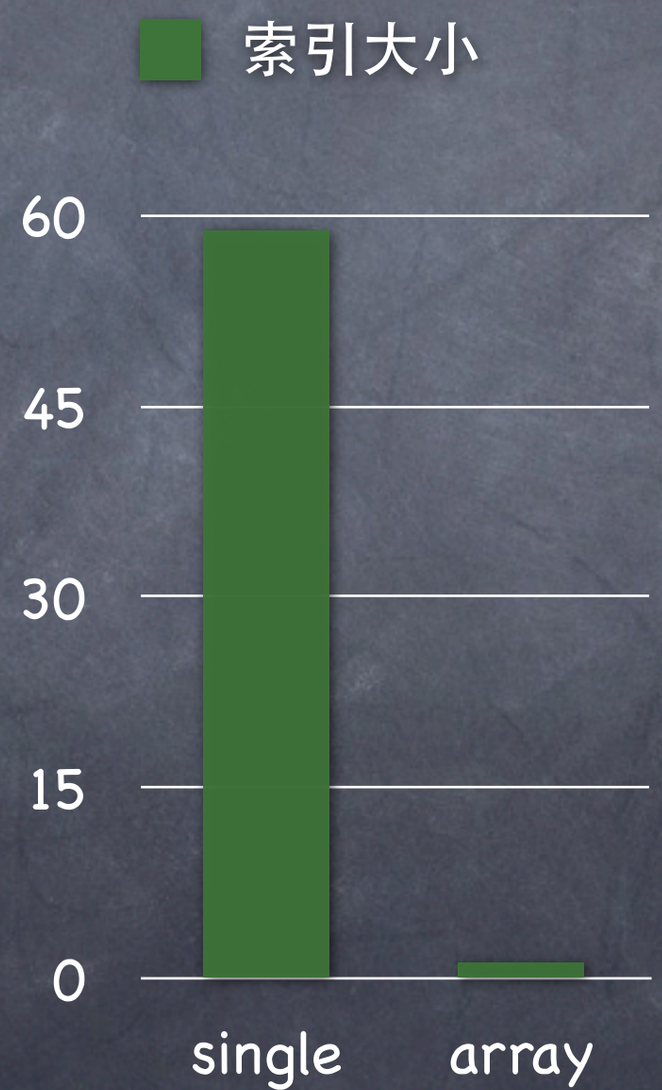
{uid:100230310,123:19394,234:11232,345:93846,.....}

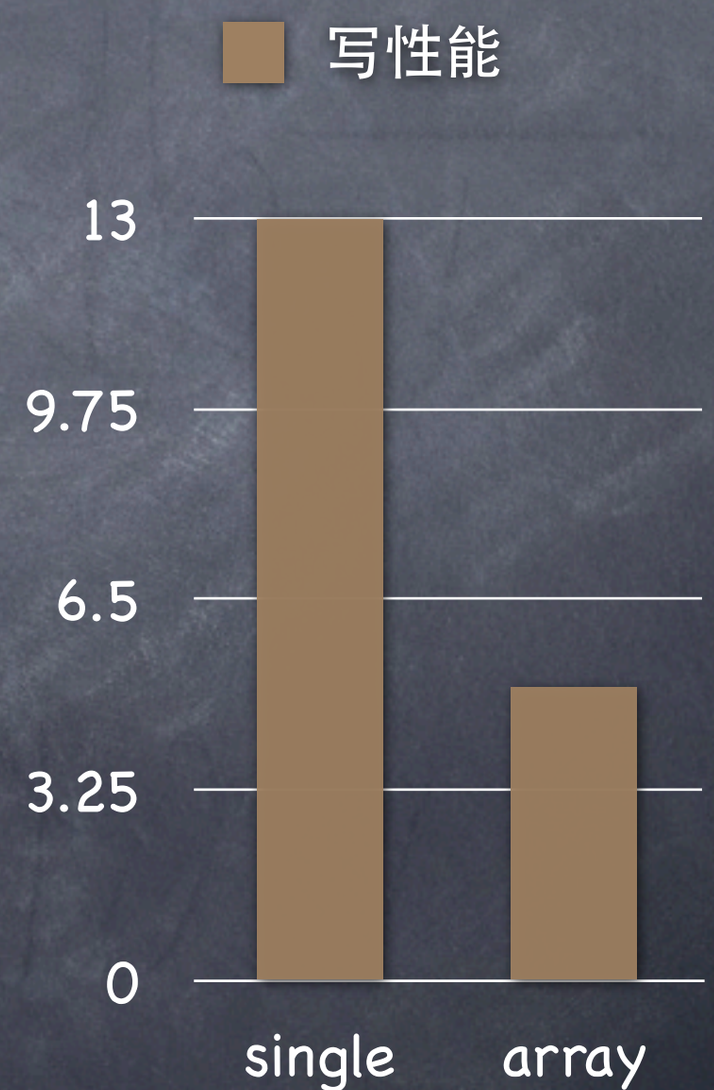
{uid:101208531,231:43242,634:7132,955:7675,.....}

{uid:103928442,938:7364,223:1192,145:8371,.....}

maxlen == 100


数据大小
50
37.5
25
12.5
0
single
array


索引大小
60
45
30
15
0
single
array


写性能
13
9.75
6.5
3.25
0
single
array

Redis计数排序

- ZSet插入排序，写排序
- ZADD weight myitem 100
- Sort命令，读排序
- SORT mylist BY weight_*

读排？ 写排？

	实时性高	实时性低
数据集大	ZSets写排	?
数据集小	?	Sort读排 +缓存

ZSet

- 结构：hashtable + skiplist
- 数据：update频繁，排序结果变化不频繁
- 问题：update操作复杂，但不必要

ZSet的成本

```
curobj = dictGetEntryKey(de);
redisAssert(zslDelete(zs->zsl,curscore,curobj));
znode = zslInsert(zs->zsl,score,curobj);
incrRefCount(curobj);
dictGetEntryVal(de) = &znode->score;
```

delete

insert

update

ZSet的成本

```
curobj = dictGetEntryKey(de);
redisAssert(zslDelete(zs->zsl,curscore,curobj));
znode = zslInsert(zs->zsl,score,curobj);
incrRefCount(curobj);
dictGetEntryVal(de) = &znode->score;
```

O(logn)

O(logn)

O(1)

改进

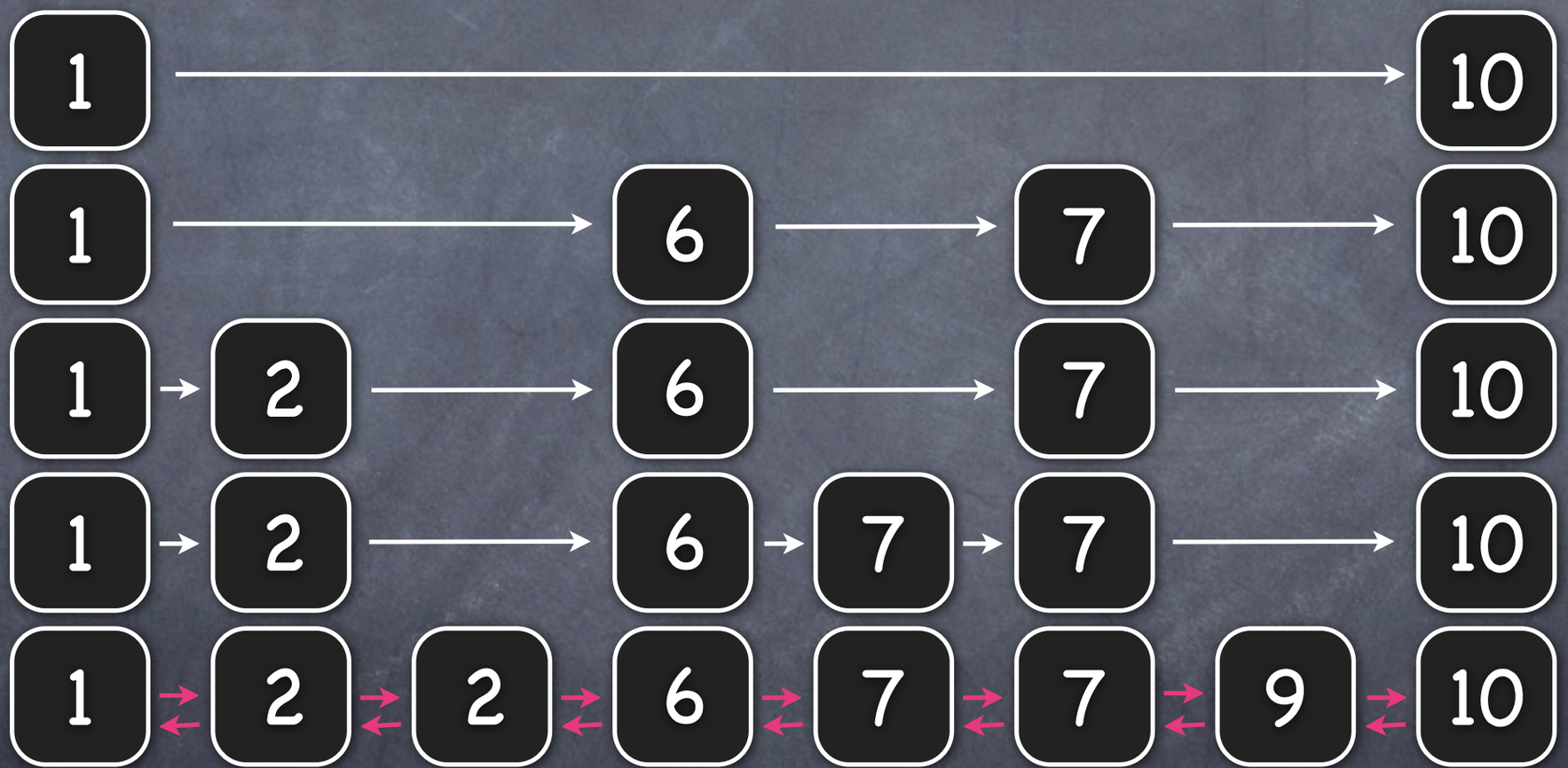
HashTable

Key:Score => Key:Node

添加判断

if no_need_move

双向链表



SkipList

增加判断

```
929             //iammutex add start
930                 if((ln->backward == NULL || score > ln->backward->score)
931                 && (ln->level[0].forward == N                    vel[0].forward->score))
932                 {
933                     //no need move
934                     ln->score = score;
935                 }else{
936                     //need move
937                     redisAssert(zslDelete(zs->zsl,curscore,curobj));
938                     znode = zslInsert(zs->zsl,score,curobj);
939                     incrRefCount(curobj); /* Re-inserted in skiplist. */
940                     dictGetEntryVal(de) = znode; /* Update score ptr. */
941                 }
942             //iammutex add end
```

O(1)

if need_move

del + add O(log(N))

else

update O(1) >90%

you hua

优化

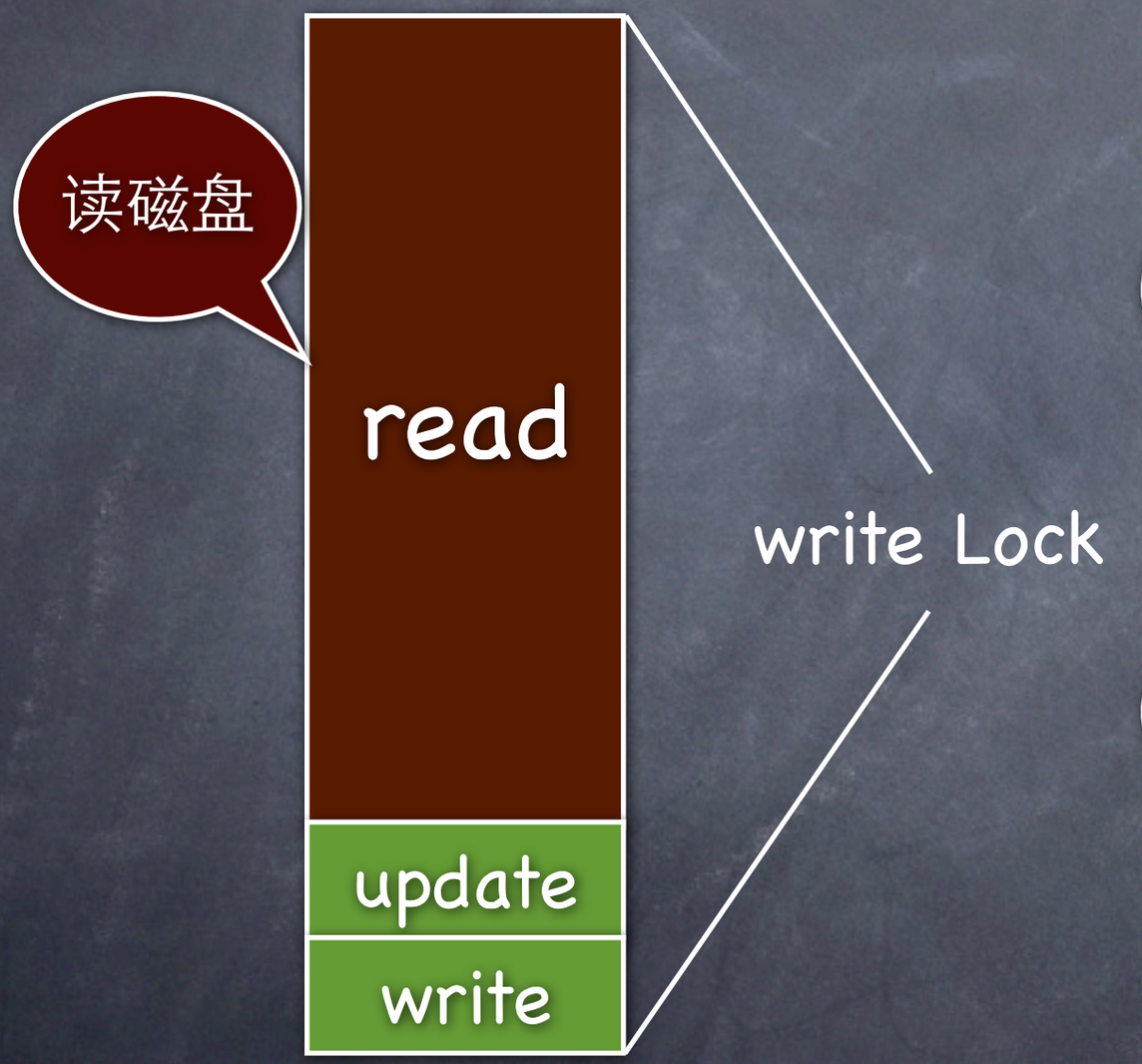
MongoDB优化

- 文件系统
  - XFS/ext4利用fallocate快速分配空间
- 容量规划
  - 内存 >= 热数据+索引+系统开销
  - fault到磁盘能承受，引入SSD?
- 连接优化：线程的stack size调整

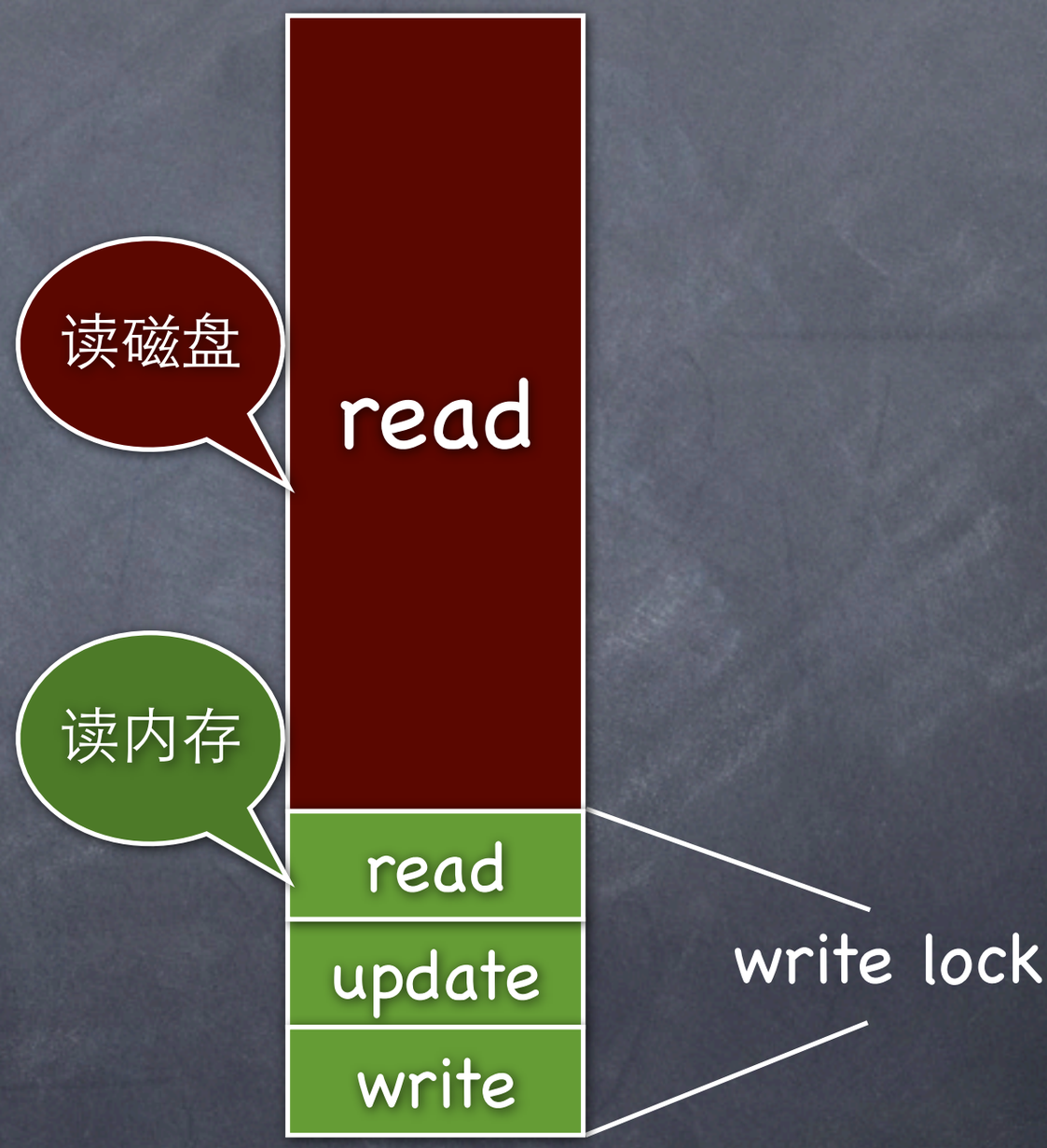
MongoDB优化

- 基于MMAP的优化
  - 先读后改，cat预热，热点集中
- 基于BSON格式的优化
  - 数据类型，Key值映射，大头定理，压缩
- 索引优化
  - 数据类型，Sparse Index，实时排序

先读后改



直接Update



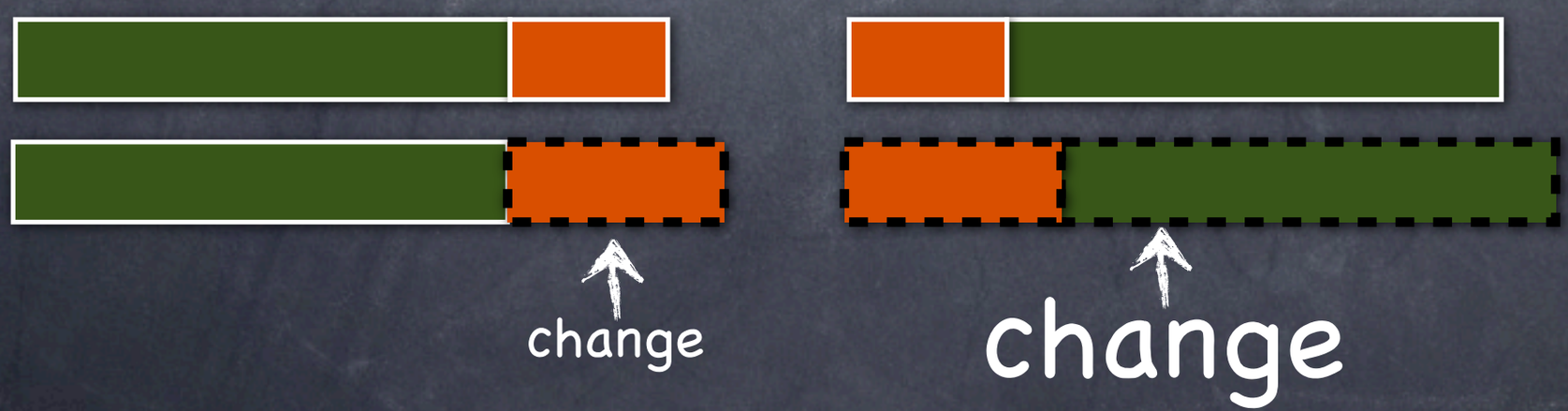
Read + Update

Cat预热

cat dbname.6 dbname.7 > /dev/null

BSON

- 压缩
- addtime or at
- Key字母顺序
- {p:**********,n:**} or {n:**,p:**********}

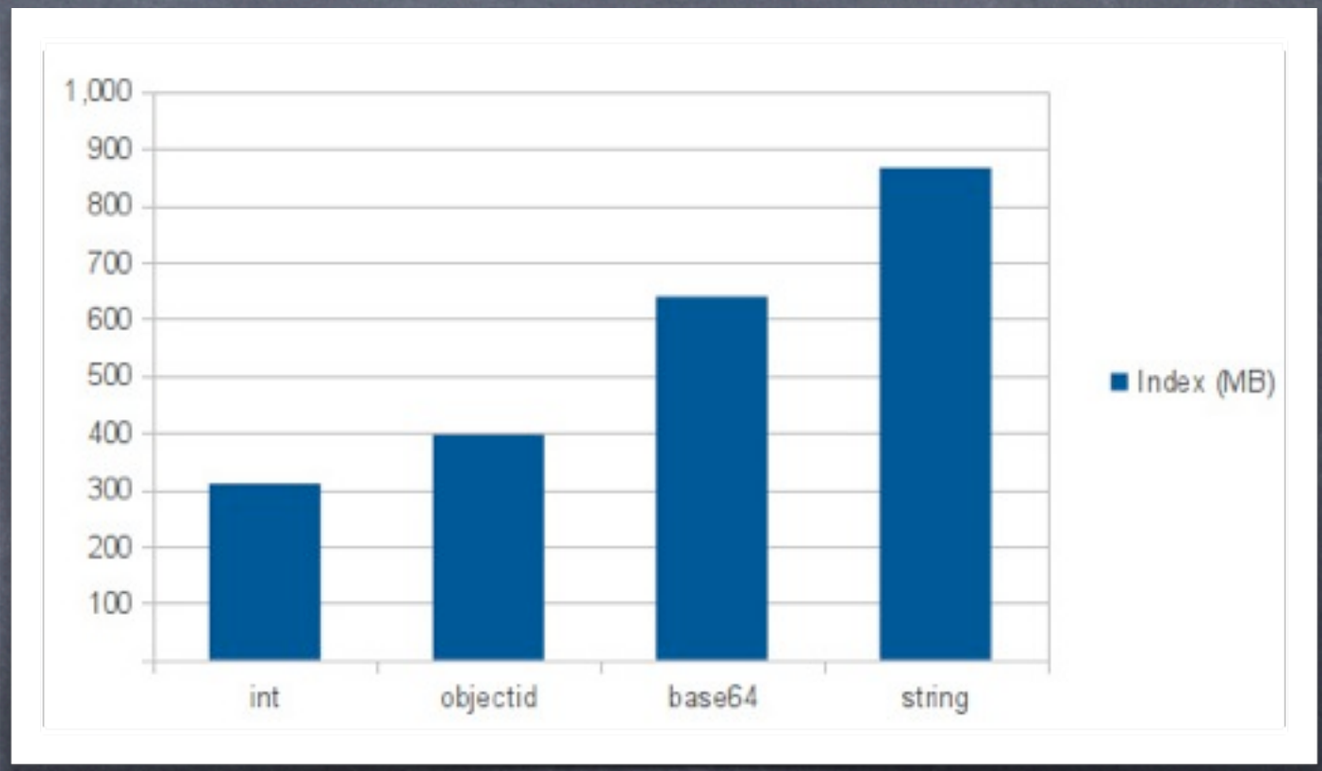


Index - _id

- 12byte的浪费
- 联合索引: _id => {a:1,b:1}
  - $gt:{a:n,b:0}
  - $lt:{a:n,b:999999999}

Index:类型

- Int
- ObjectID
- Base64
- String



Redis优化

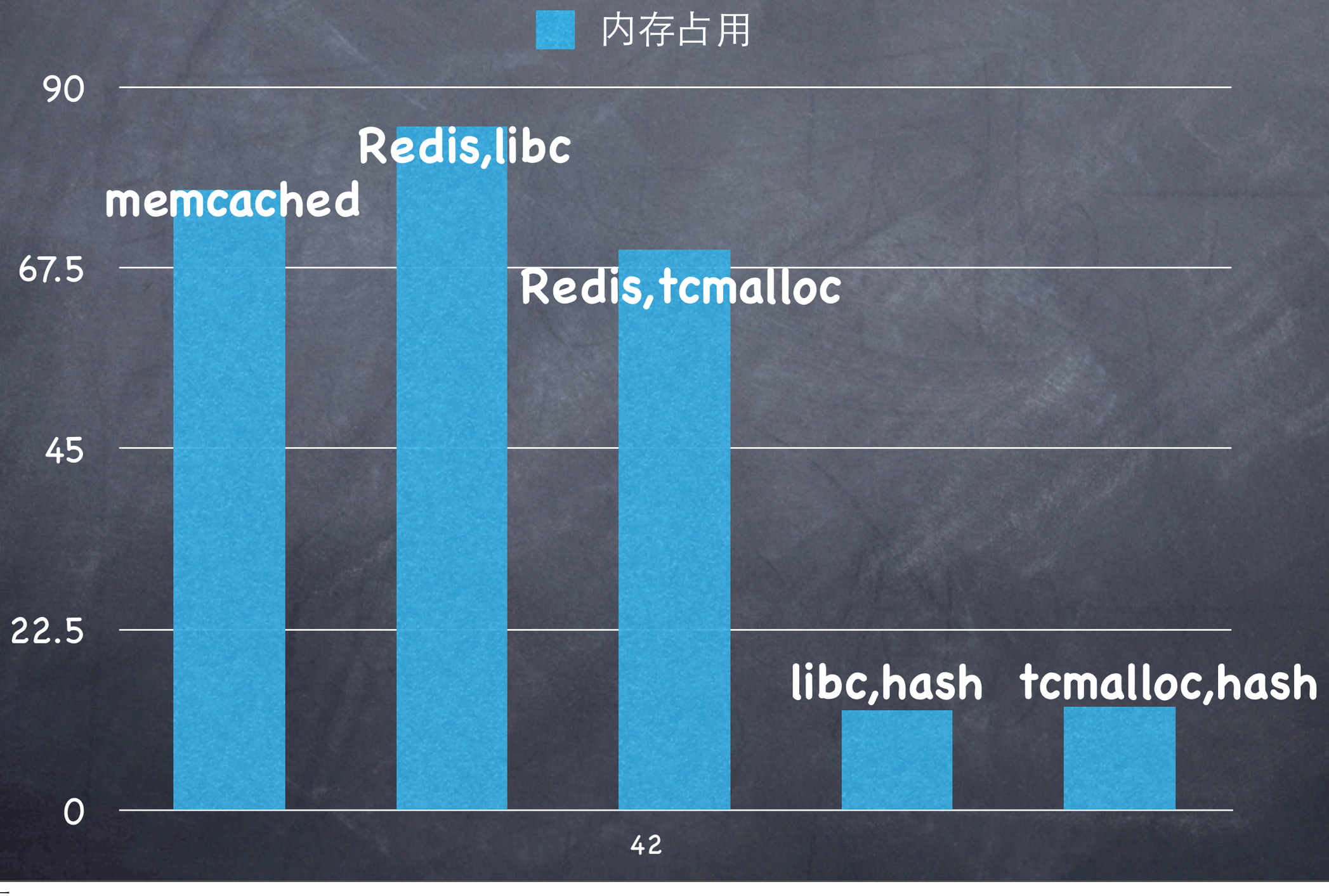
- 容量规划：all in memory，allocator选择
- 压缩：zipmap，ziplist
- 连接的代价：PHP引入Proxy层
- 一夫当关问题：快慢操作分离

all in memory

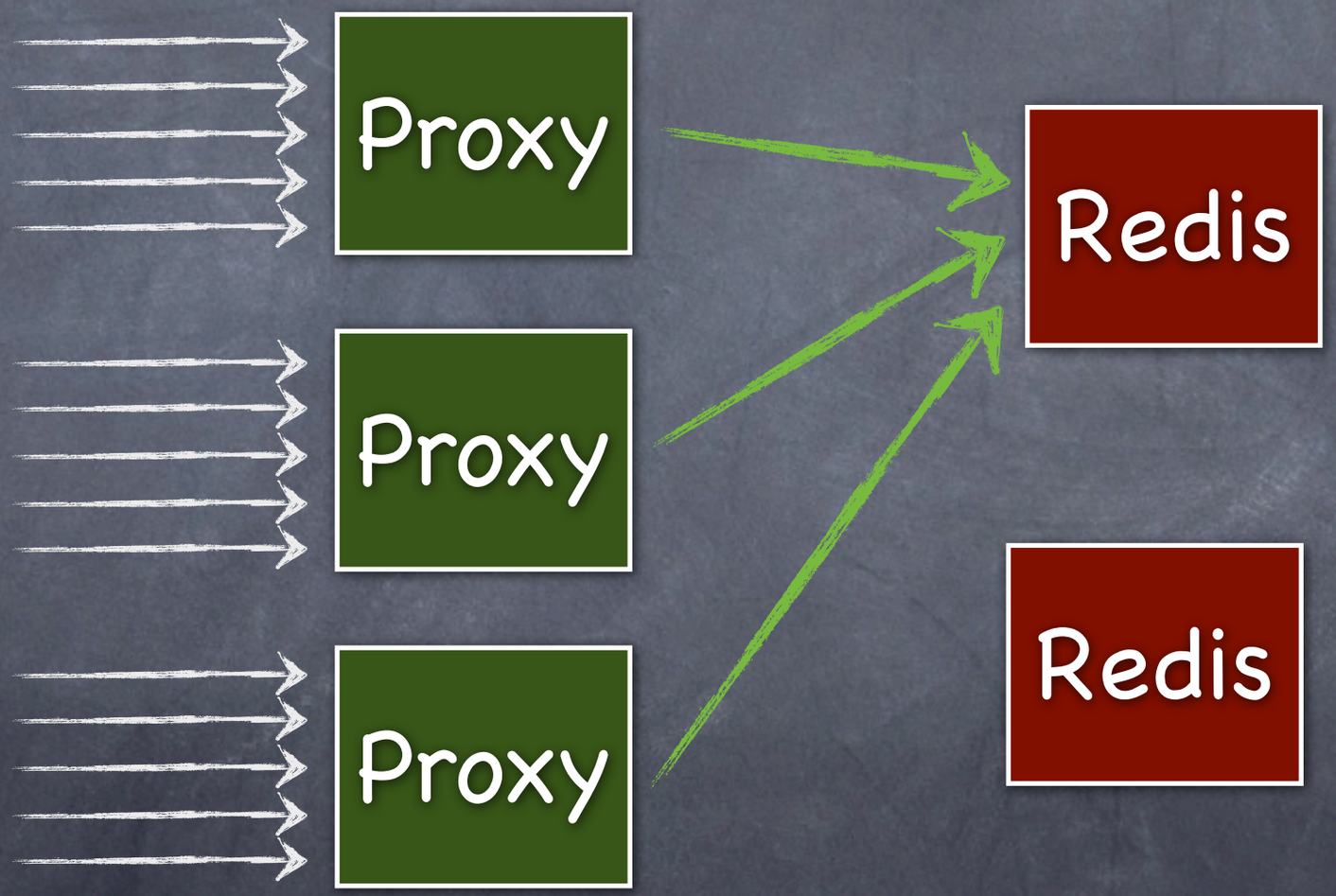
maxmemory = 21474836480 (20GB)

NUMA?

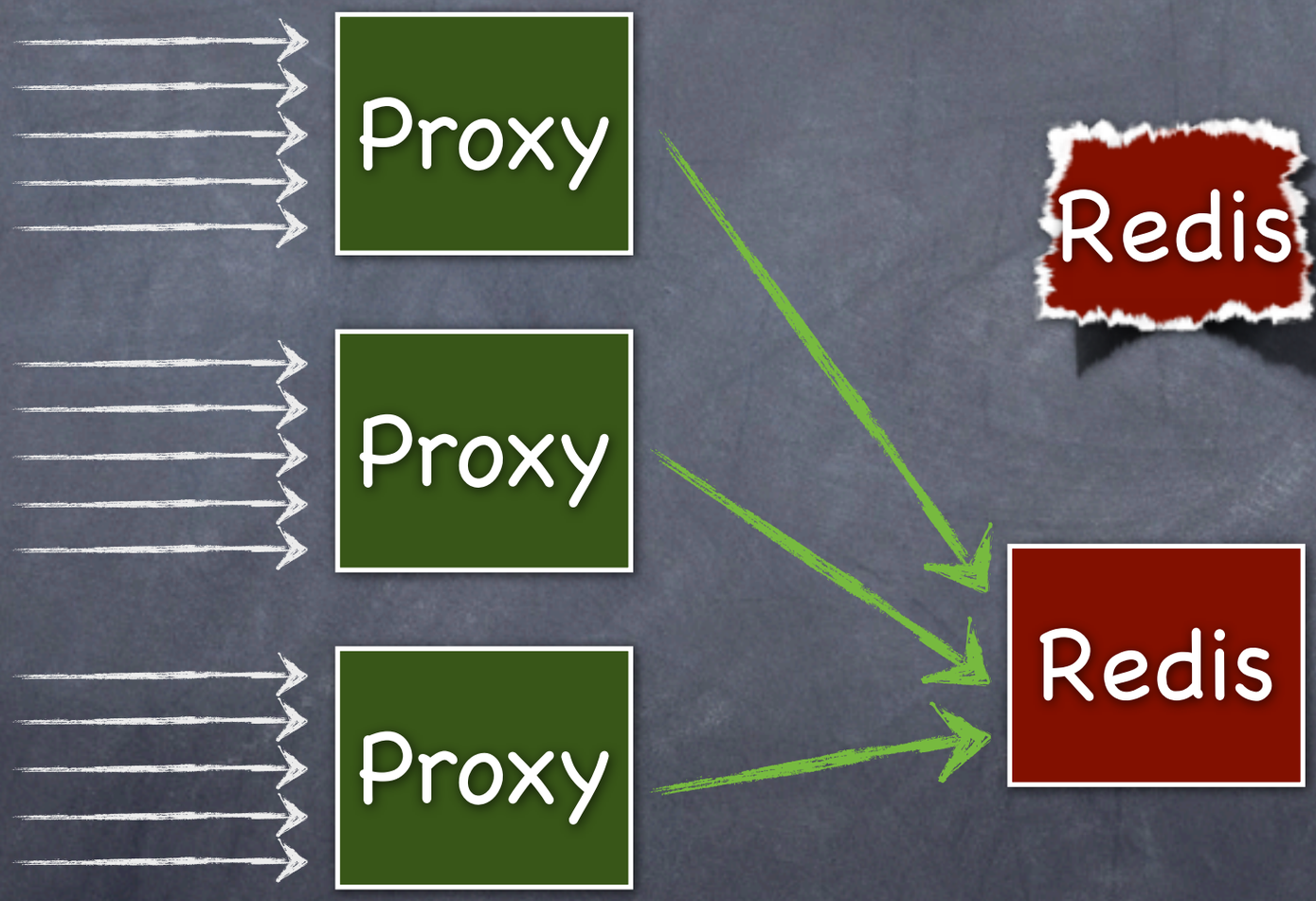
Redis耗内存?


内存占用
90
67.5
45
22.5
0
memcached
Redis,libc
Redis,tcmalloc
libc,hash
tcmalloc,hash
42

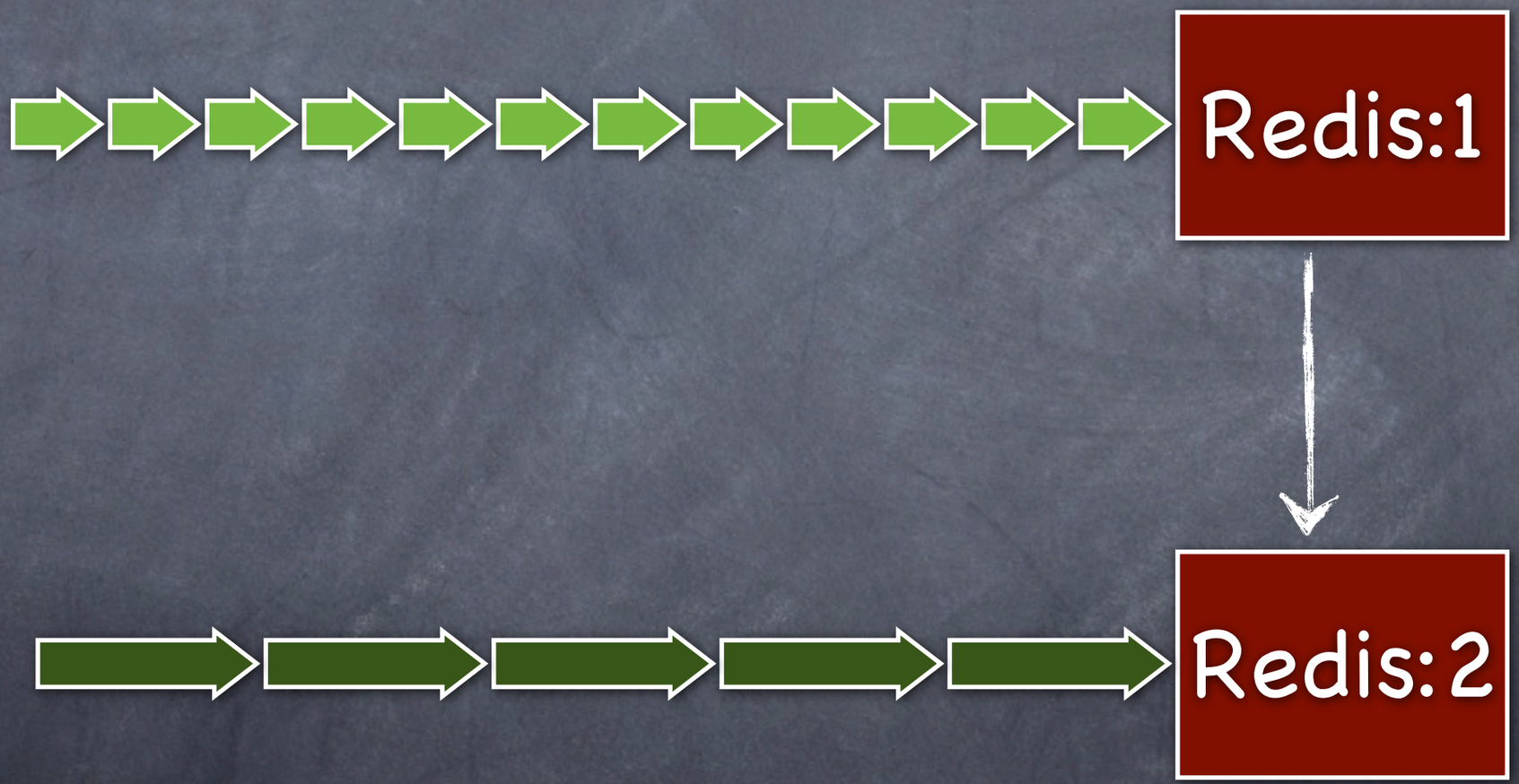
Proxy:连接池



Proxy:热切换



操作分离



xie xie

谢谢