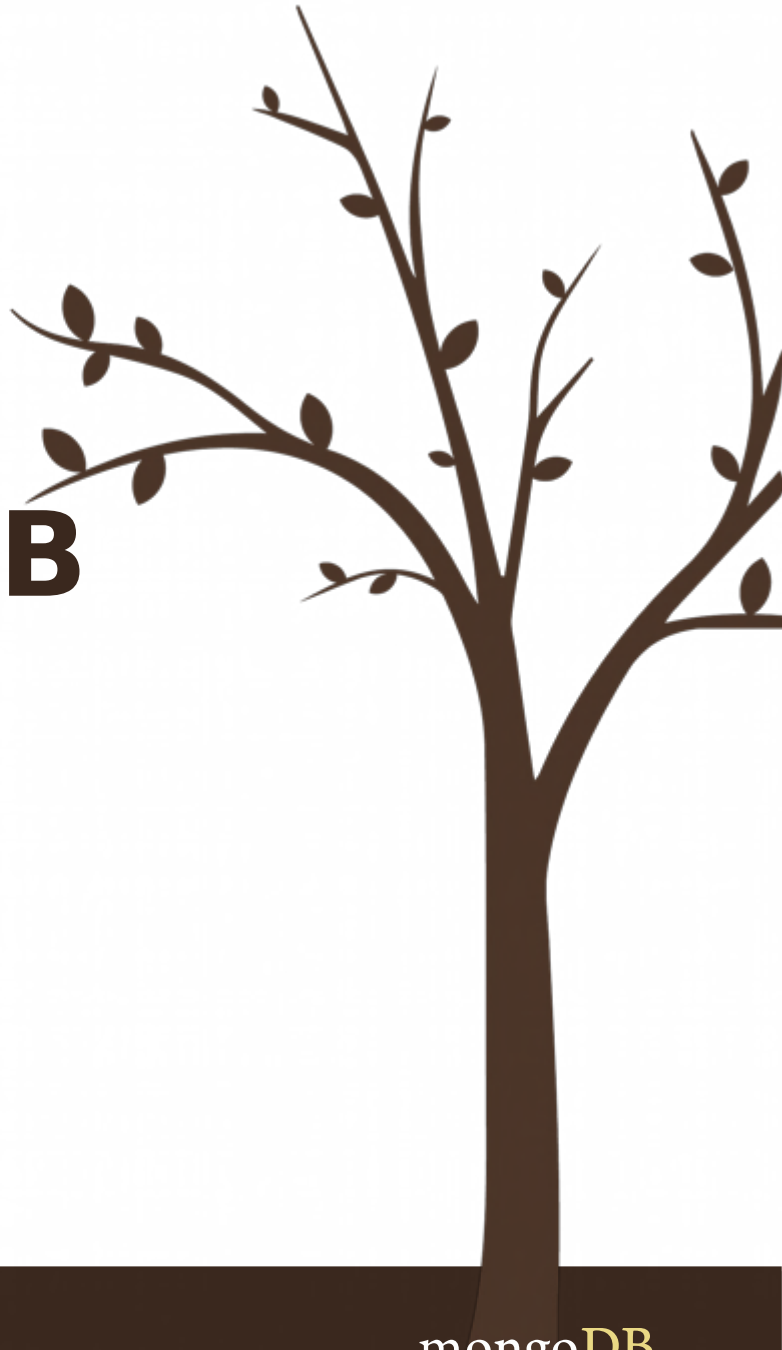
# Scaling MongoDB
## 如何进行性能扩展

TJ Tang
Senior Solutions Architect



mongoDB

# About Me

TJ - 唐建法

MongoDB 高级方案架构师

原 FedEx 首席架构师

15 年电脑键盘， 5 年乐队键盘

微博： @TJ_MongoDB
微信： tjtang826

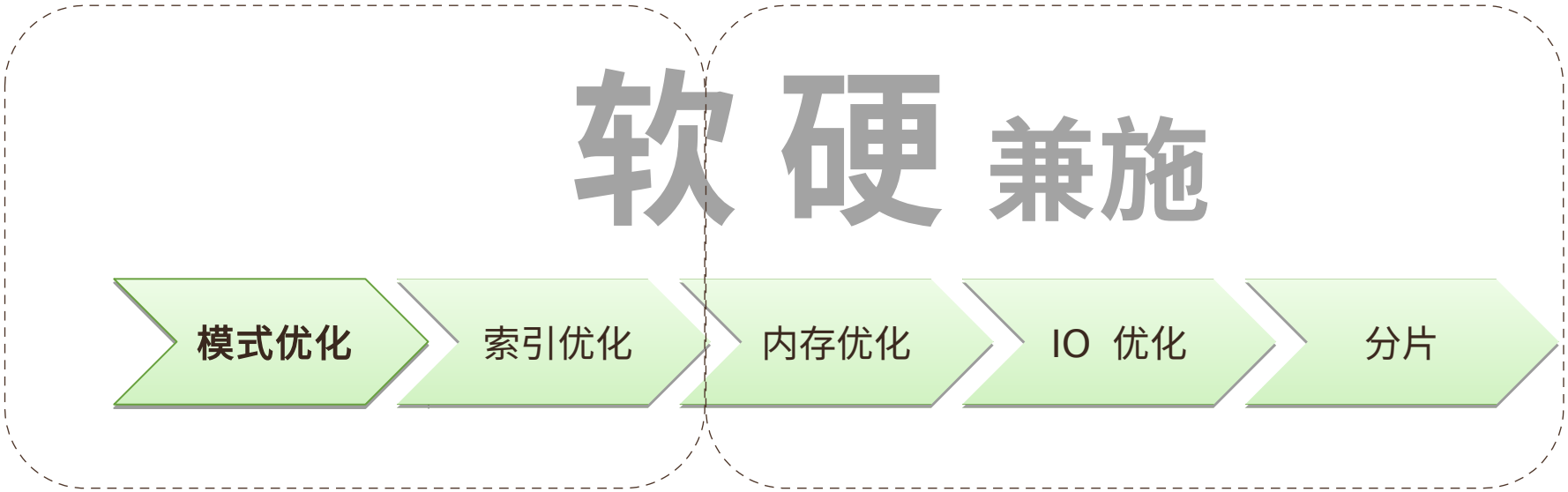
# BUILD SOMETHING BIG

# WITH MONGODB


每秒 120 万次数据库操作


500 亿条记录并在快速增长中

它们是怎样“Scale”上去的呢?




# MongoDB 性能扩展策略

# 真实案例 – 电商产品目录

产品检索缓慢

Page Faults 居高不下

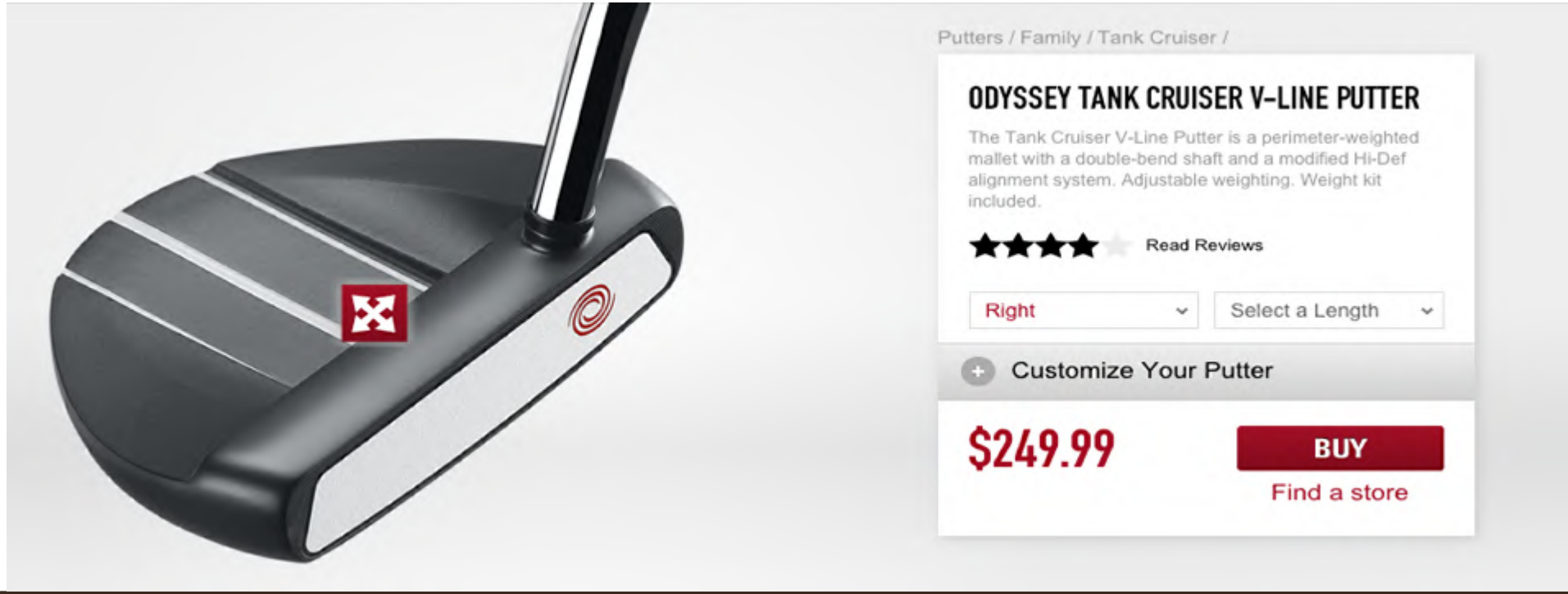
# 产品文档数据模式：

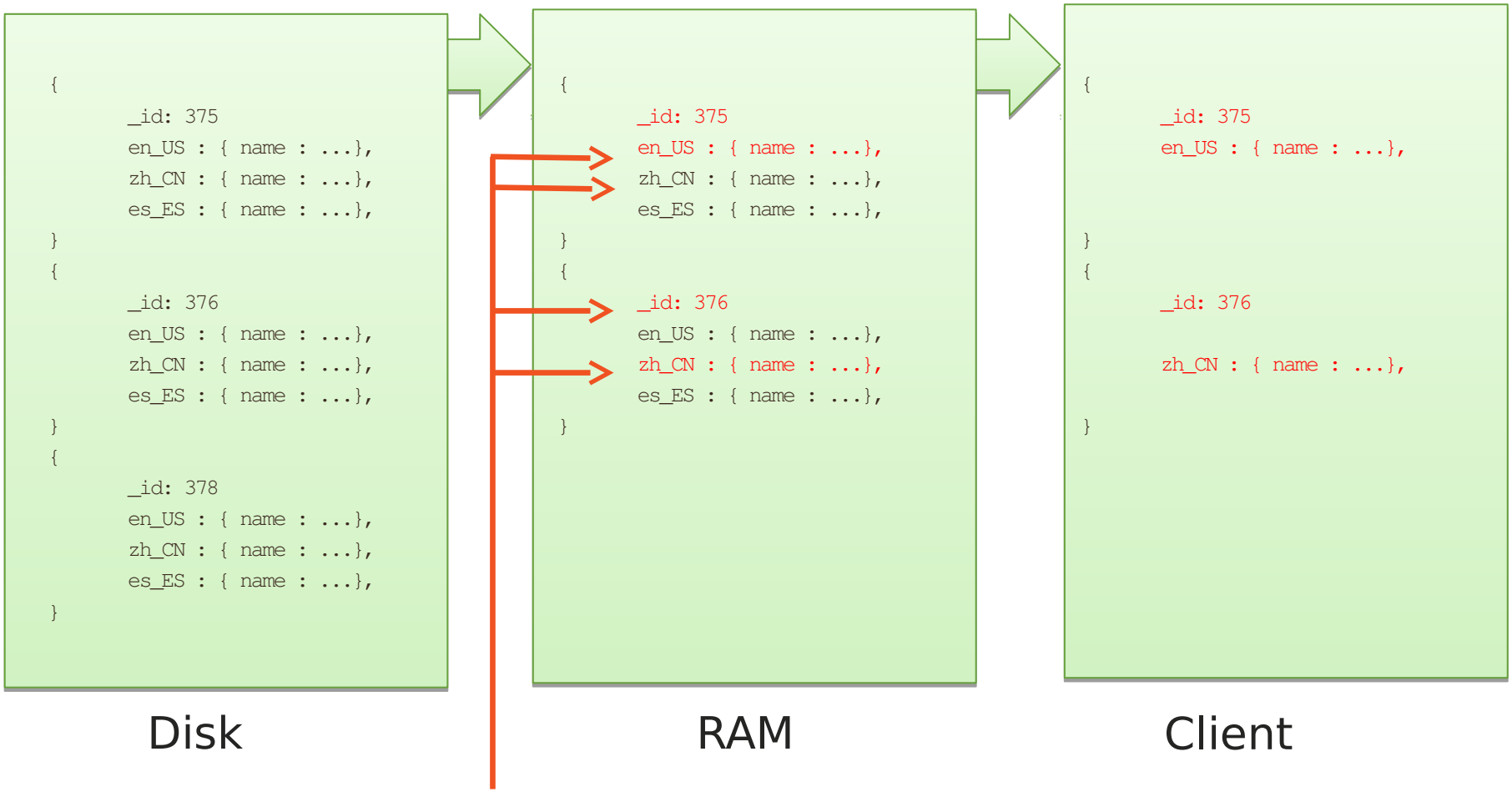
```
{
    _id: 375
    en_US : { name : "Callaway V-Line Putter", desc : ...,
<etc...> },
    zh_CN : { name : "卡拉威 V 系列推杆" ,          desc : ...,
<etc...> },
    es_ES : { name : ..., description : ..., <etc...> },
    de_CH : ...,
    <... 20 多种语言 ... >
}
```

**设计思想：**所有产品相关信息都在一个文档内

# 常用查询

```
db.catalog.find( { _id : 375 } , { en_US : true } );
db.catalog.find( { _id : 376 } , { zh_CN : true } );
```




{
_id: 375
en_US : { name : ...},
zh_CN : { name : ...},
es_ES : { name : ...},
}
{
_id: 376
en_US : { name : ...},
zh_CN : { name : ...},
es_ES : { name : ...},
}
{
_id: 378
en_US : { name : ...},
zh_CN : { name : ...},
es_ES : { name : ...},
}
{
_id: 375
en_US : { name : ...},
zh_CN : { name : ...},
es_ES : { name : ...},
}
{
_id: 376
en_US : { name : ...},
zh_CN : { name : ...},
es_ES : { name : ...},
}
{
_id: 375
en_US : { name : ...},
}
{
_id: 376
zh_CN : { name : ...},
}
Disk
RAM
Client
浪费的内存空间：
20x

# 解决方案

```
{ _id: "375-en_US", name : ...,    <etc...> }

{ _id: "375-zh_CN", name : ...,   <etc...> }

{ _id: "375-fr_FR", name : ..., <etc...> }

  ...    ...
```

- 内存可以装下 20 多倍文档
- 产品检索时间显著的缩短

# 企业社交网“朋友圈”

圈子动态显示太慢
查询需要 2 秒以上

# 数据模式

## 状态集合

```
{
    creator_id: “xxxxx”,
    status: “MongoDB 2.8 rc1 发布了”
}
{
    creator_id: “yyyyy”,
    status: “休假中。。。”
}
```

## 用户集合

```
{
    _id: “xxxxx”,
    friends: [ “yyyyy”, “zzzzz” ]
}
```

## 我圈子的动态：

```
db.status.find( { creator_id: { $in: [array-o-friends] } } ).sort({ _id: -1 } )
```

N 次检索

# 使用冗余来提升读性能

状态集合

```
{
    creator_id: "xxxxx",
    friends: [ "yyyyy", "zzzzz" ],
    status: "MongoDB 2.8 rc1 发布了"
}
```

用户集合

```
{
    _id: "xxxxx",
    friends: [ "yyyyy", "zzzzz" ]
}
```

我圈子的动态：

```
db.status.find( { friends: "xxxxx" } ).sort({ _id: -1 } )
```

耗时： 几个毫秒

# 小结：

- 根据数据的访问方式而设计模式
- 为最重要的查询做预聚合处理
  - 必要的时候可以冗余数据


索引优化
Low Hanging Fruit

# 100 万条记录， 4 万个 John Smith 记录

```
{
     first_name:   “John”,
     last_name:    “Smith”,
     phone:        “15838247283”
}
```

db.phonebook.find( { first_name: “John”, last_name: “Smith” } )

# 无索引

用 explain() 来显示查询计划

```
> db.phonebook.find({first_name:'John', last_name:'Smith'}).explain()
{
        "cursor" : "BasicCursor",
        "isMultiKey" : false,
        "n" : 40001,
        "nscannedObjects" : 1000000,
        "nscanned" : 1000000,
        "nscannedObjectsAllPlans" : 1000000,
        "nscannedAllPlans" : 1000000,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 7812,
        "nChunkSkips" : 0,
        "millis" : 760,
        "server" : "MacBook-Pro-13.local:27017",
        "filterSet" : false
}
>
```

# db.phonebook.ensureIndex({first_name:1})

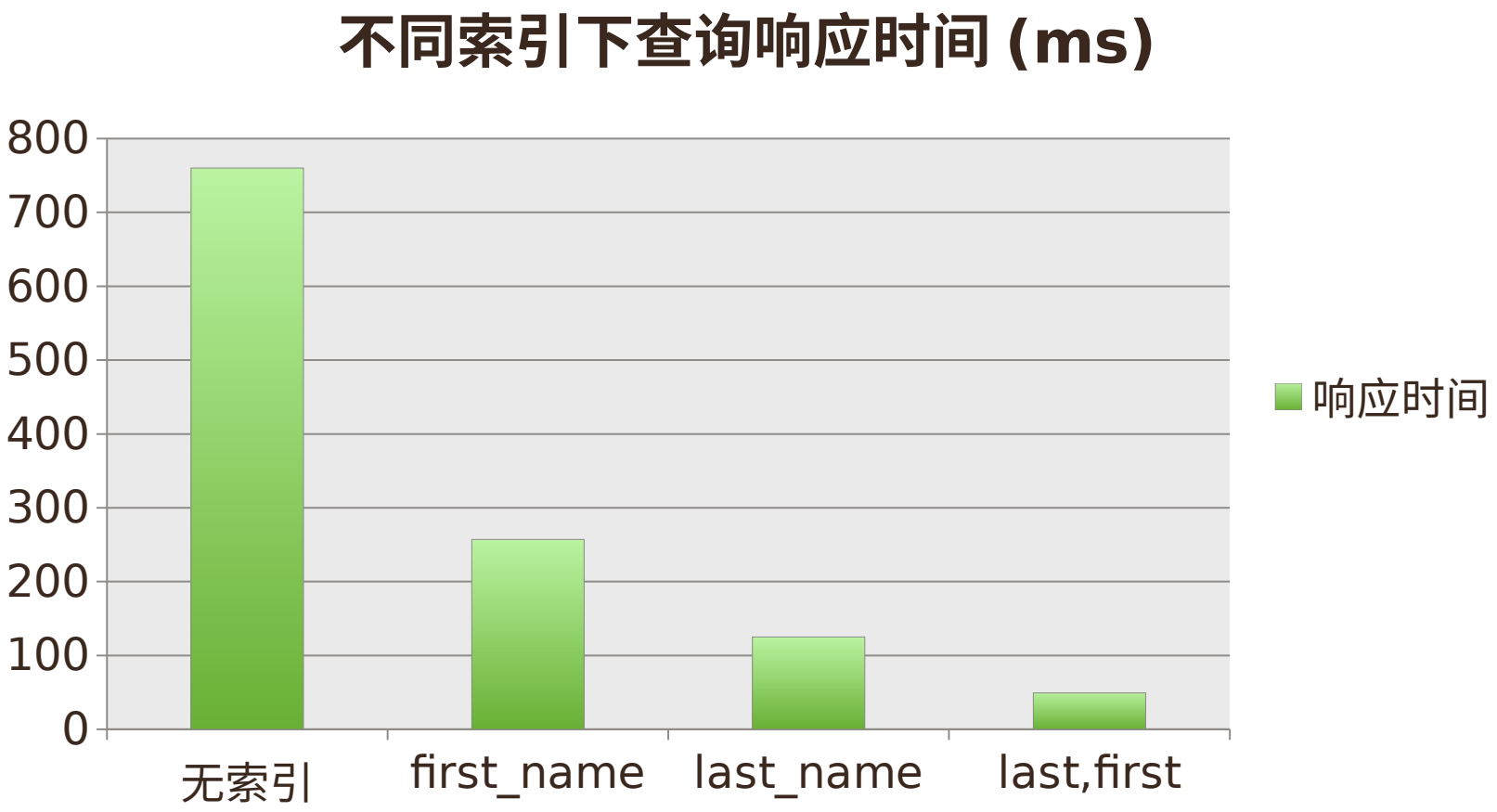
```
> db.phonebook.find({first_name:'John', last_name:'Smith'}).explain()
{
        "cursor" : "BtreeCursor first_name_1",
        "isMultiKey" : false,
        "n" : 40001,
        "nscannedObjects" : 200360,
        "nscanned" : 200360,
        "nscannedObjectsAllPlans" : 200360,
        "nscannedAllPlans" : 200360,
        "indexOnly" : false,
        "nYields" : 1565,
        "millis" : 247,
        "indexBounds" : {
                "first_name" : [
                        [
                                "John",
                                "John"
                        ]
                ]
        },
        "server" : "MacBook-Pro-13.local:27017",
}
```

# db.phonebook.ensureIndex({last_name:1})

```
> db.phonebook.find({first_name:'John', last_name:'Smith'}).explain()
{
        "cursor" : "BtreeCursor last_name_1",
        "isMultiKey" : false,
        "n" : 40001,
        "nscannedObjects" : 89925,
        "nscanned" : 89925,
        "nscannedObjectsAllPlans" : 89925,
        "nscannedAllPlans" : 89925,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 702,
        "nChunkSkips" : 0,
        "millis" : 125,
        "indexBounds" : {
                "last_name" : [
                        [
                                "Smith",
                                "Smith"
                        ]
                ]
        },
        "server" : "MacBook-Pro-13.local:27017",
        "filterSet" : false
}
```

# db.phonebook.ensureIndex({last_name:1, first_name:1})

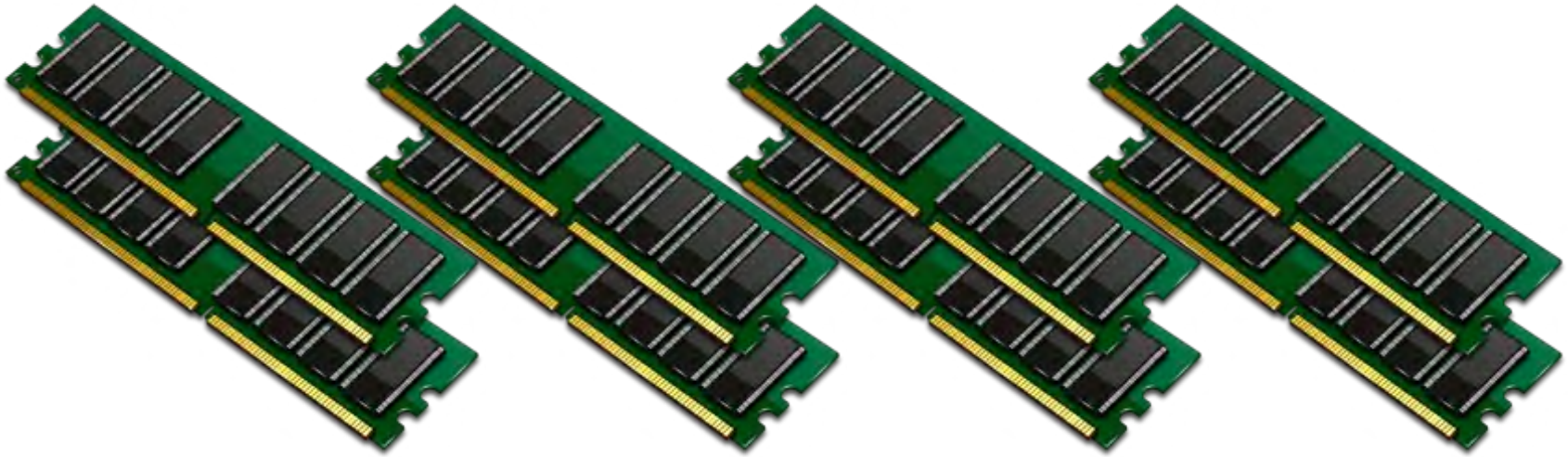
```
> db.phonebook.find({first_name:'John', last_name:'Smith'}).explain()
{
        "cursor" : "BtreeCursor last_name_1_first_name_1",
        "isMultiKey" : false,
        "n" : 40001,
        "nscannedObjects" : 40001,
        "nscanned" : 40001,
        "nscannedObjectsAllPlans" : 40102,
        "nscannedAllPlans" : 40103,
        "nYields" : 312,
        "millis" : 49,
        "indexBounds" : {
                "last_name" : [
                        [
                                "Smith",
                                "Smith"
                        ]
                ],
                "first_name" : [
                        [
                                "John",
                                "John"
                        ]
                ]
        },
}
```


不同索引下查询响应时间(ms)
800
700
600
500
400
300
200
100
0
无索引
first_name
last_name
last,first
响应时间


## 大部分查询要使用索引

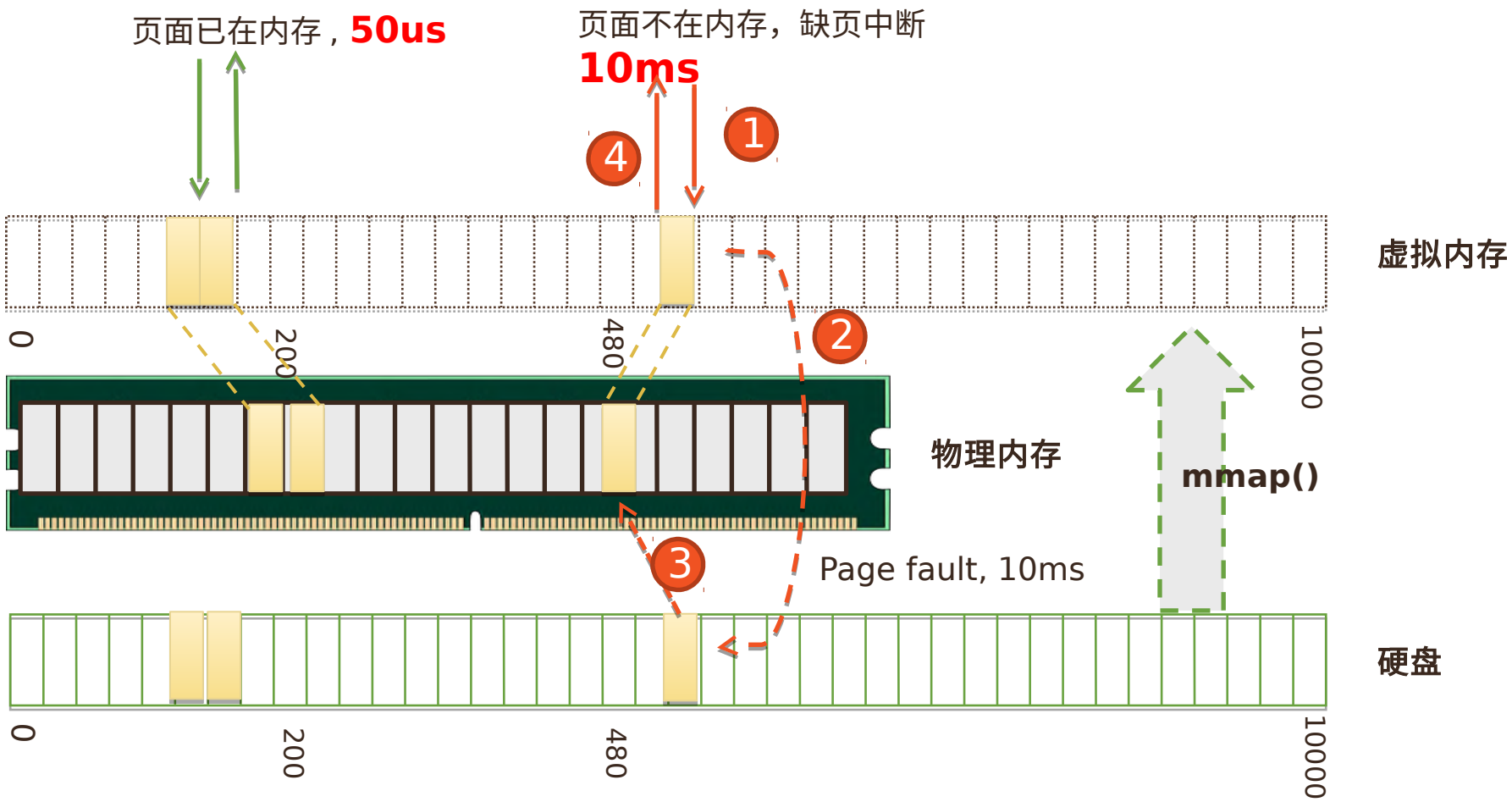
（索引会增加写操作的代价）

## 测试并选择最优索引

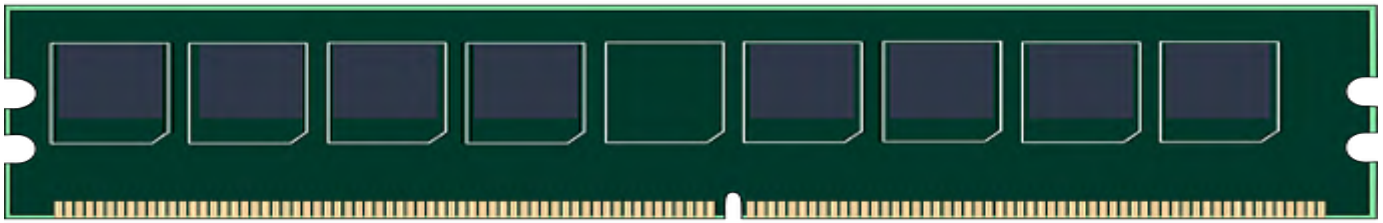


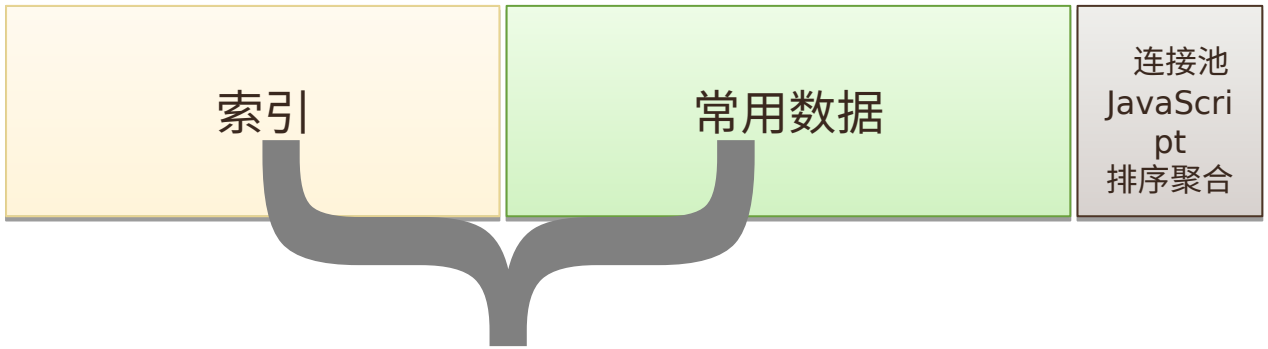
# 内存优化

# MongoDB 与内存映射




# 优化目标：内存容量 > 工作集大小






# 工作集


# 如何计算工作集大小?

# 索引

```
> db.stats()
{
        "db" : "test",
         ...
        "indexes" : 5,
        "indexSize" :
113196720,
        . . . ”
        ok" : 1
}
```

# 常用数据

## 方法一：估计

**日志分析：**

**4KB/doc x 1000000 docs/day x 30days = 120GB**

**产品目录：**

**100KB/doc x 100,000 docs = 10GB**

常用数据

## 方法二：监测

```
> db.serverStatus({workingSet:1})
{
        "workingSet" : {
                "note" :
"thisIsAnEstimate",
                "pagesInMemory" : 40707,

"computationTimeMicros" : 7897,
                "overSeconds" : 9082
        },
}
```

**pagesInMemory x 4096**

# 小结：

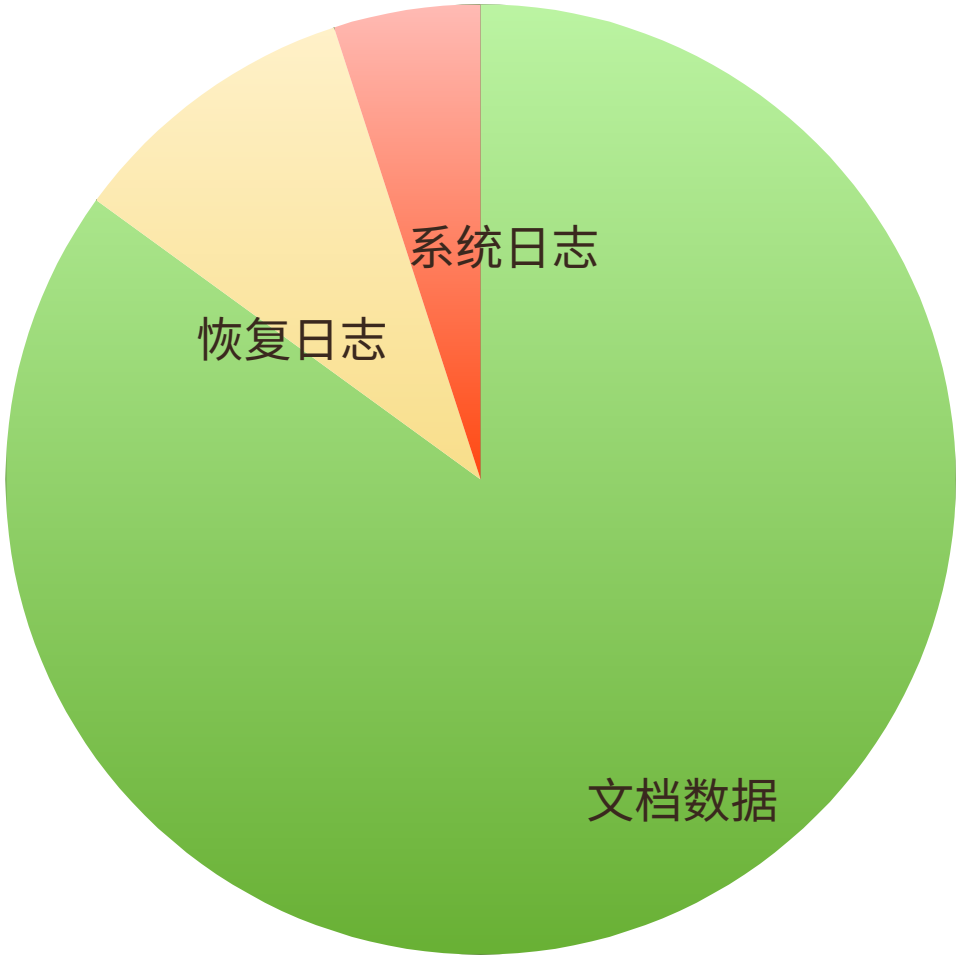
- 保证内存容量 > 工作集大小
- 工作集 = 索引大小 + 估计的常用数据大小

# 内存无法装下工作集?

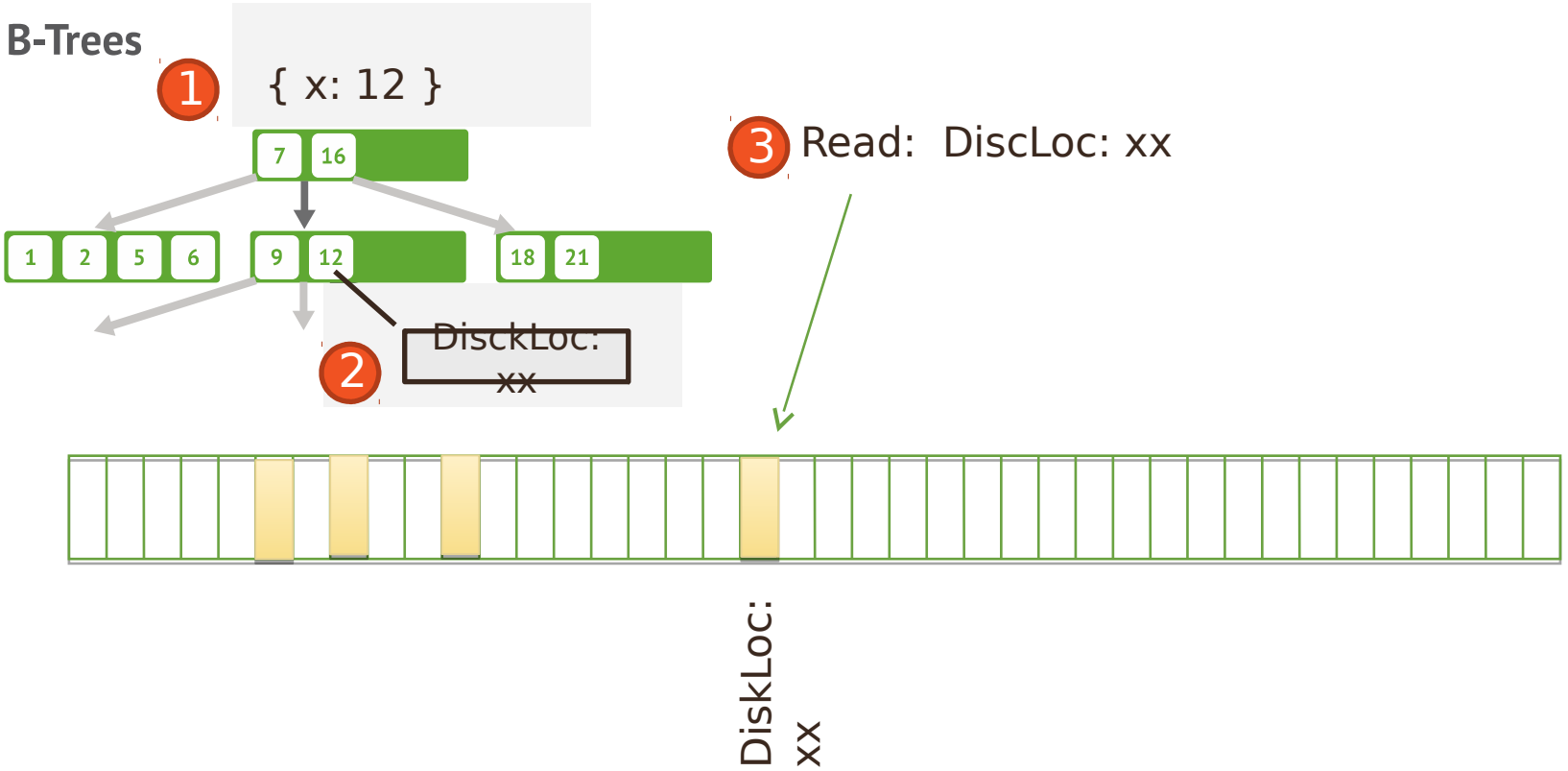
# 优化 IO

# MongoDB 的 IO 操作

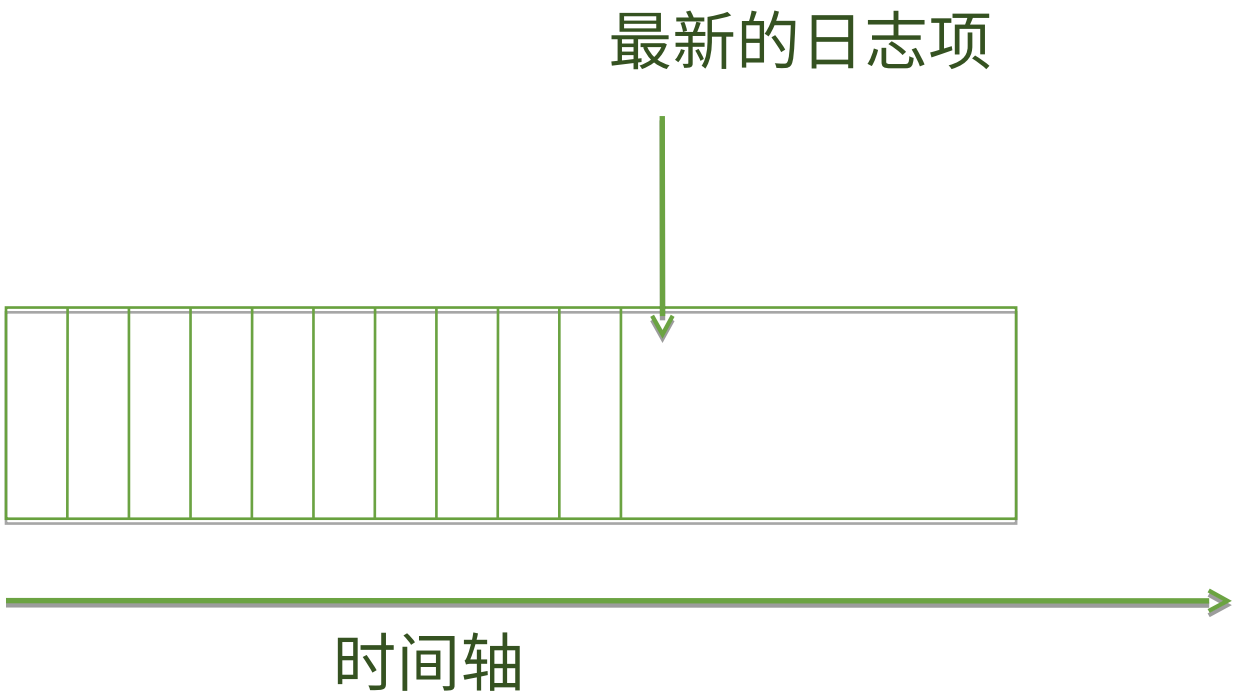
优化 IO



# 数据 IO： 随机访问为主


B-Trees
1
{ x: 12 }
7
16
1
2
5
6
9
12
18
21
2
DiskLoc:
xx
3
Read: DiscLoc: xx
DiskLoc:
xx

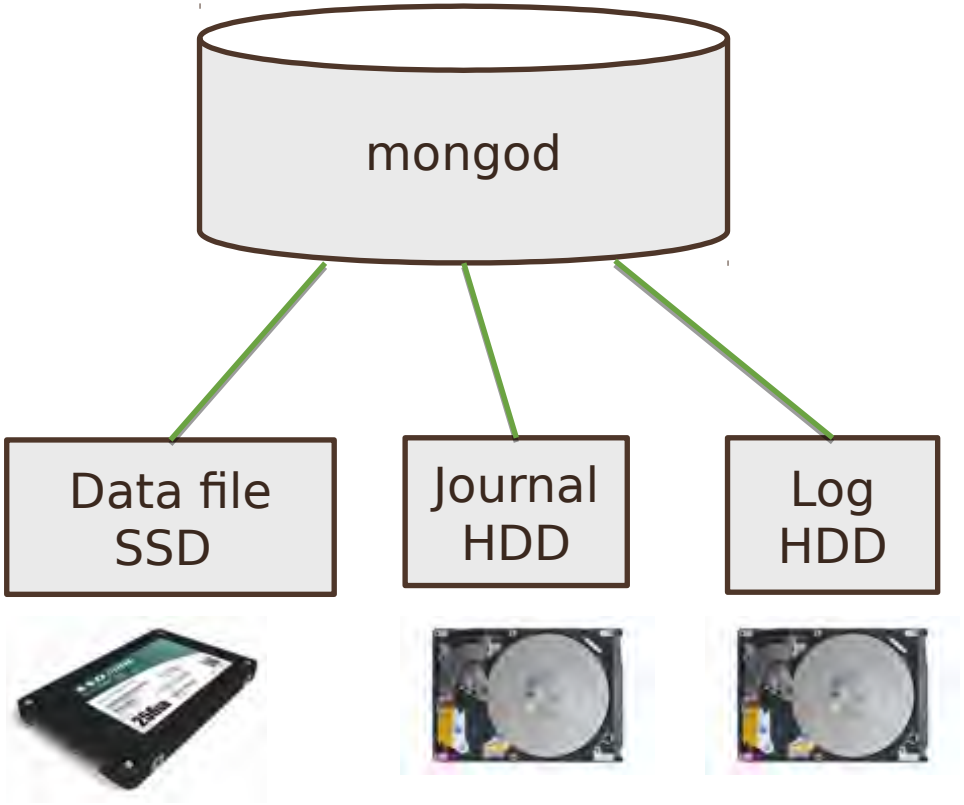
# 恢复日志和系统日志：顺序写



VS.

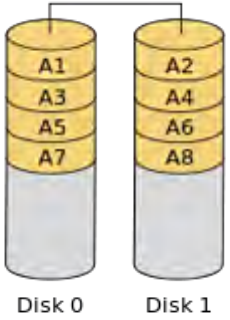
| | HDD | SSD |
|---|---|---|
| 随机访问 (IOPS) | 125 | 12000 (100x) |
| 顺序读写 (Throughput) | ~100MB/s | ~200MB/s (2x) |

# 优化的 IO 部署方案



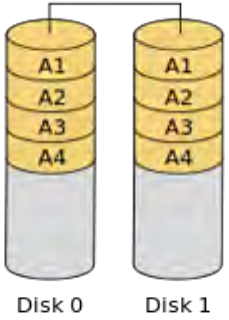
# 建议的 RAID Level


RAID 0



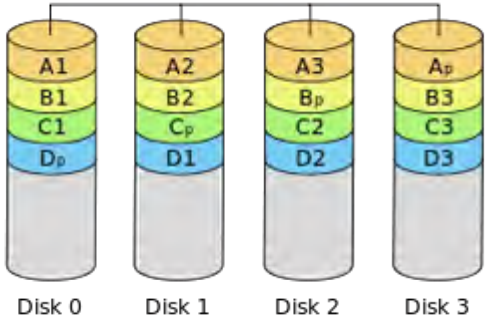

无冗余

RAID 1



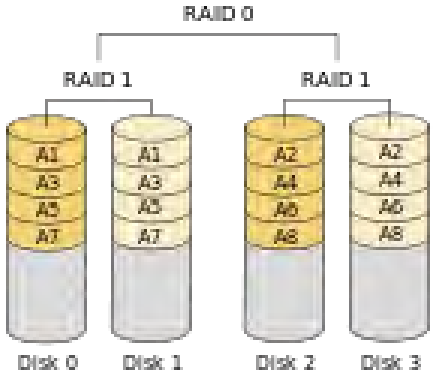

写性能一般

RAID 5




写入速度慢

RAID 1+0




条带化 + 镜像

性能好 + 数据冗余

# 其他的 IO 优化

- 预读（Read Ahead） 设置： 16 或者 32 扇区
- 文件系统： ext4 or xfs
- 禁止 noatime

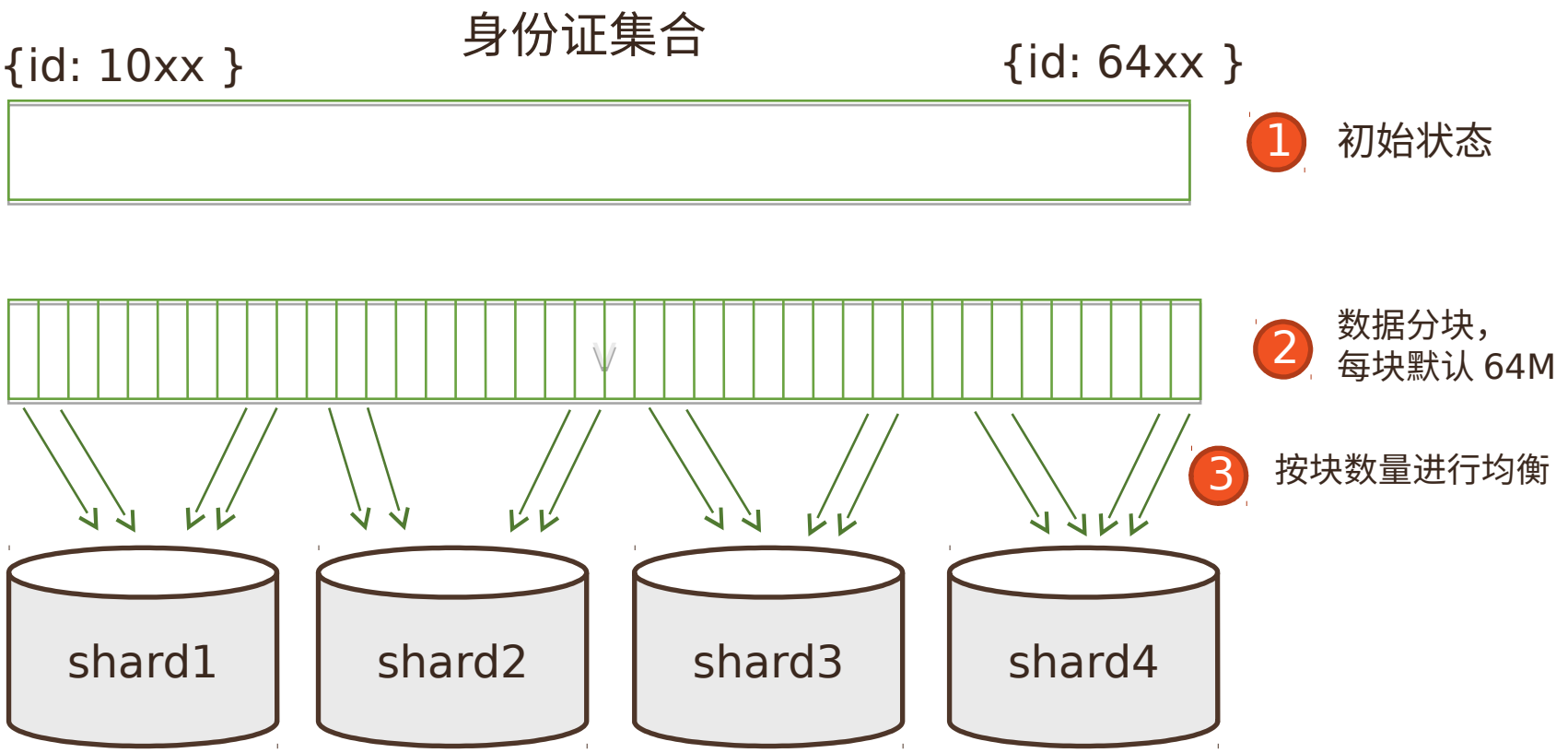

Collection1
1TB
Shard A
Collection1
256 GB
Shard B
256 GB
Shard C
256 GB
Shard D
256 GB

# 分片 – 水平扩展

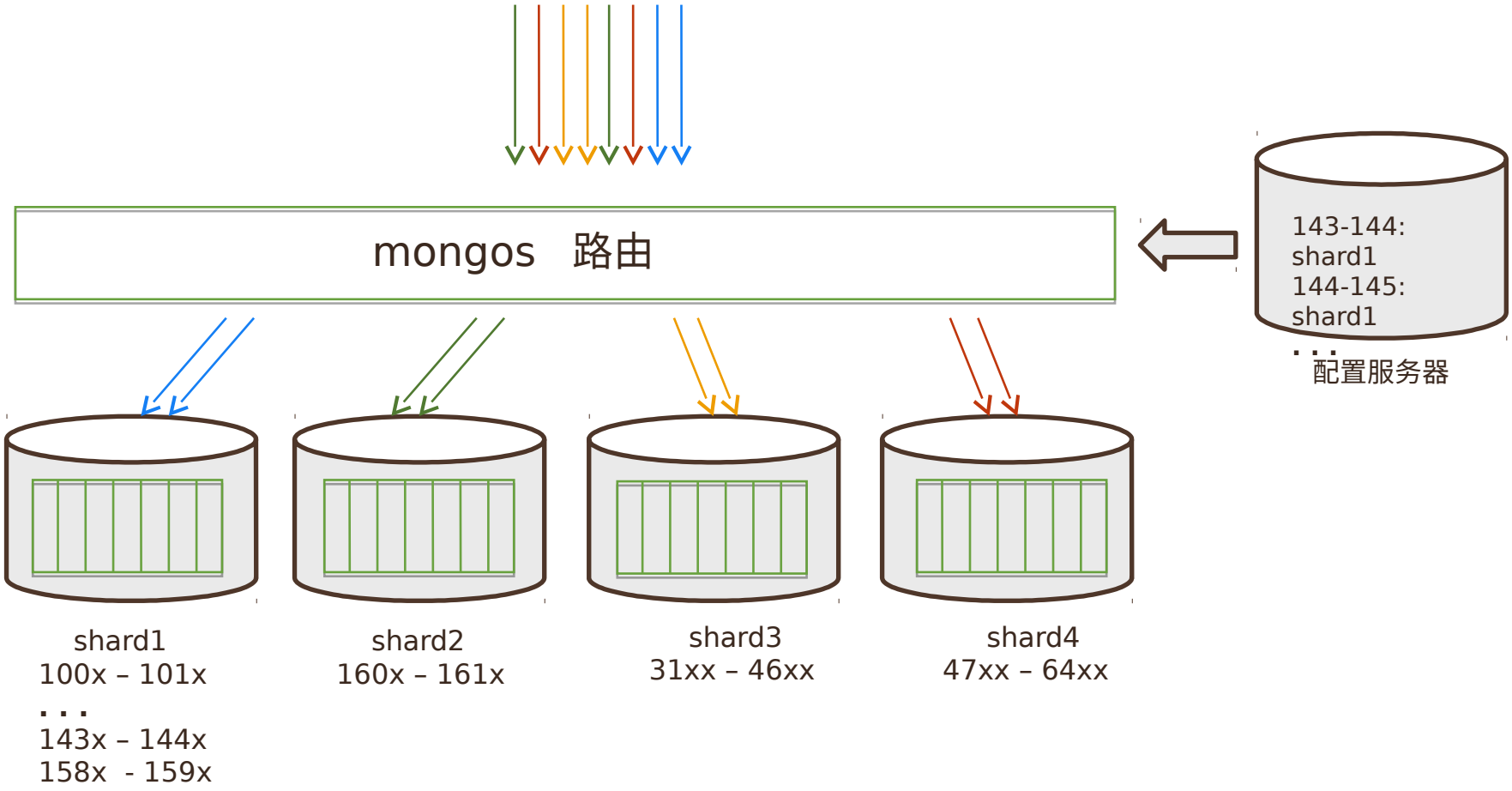
# 分片用来干什么?

- 写性能扩展
- 读性能扩展!
- 地理分布数据
- 备份的快速恢复

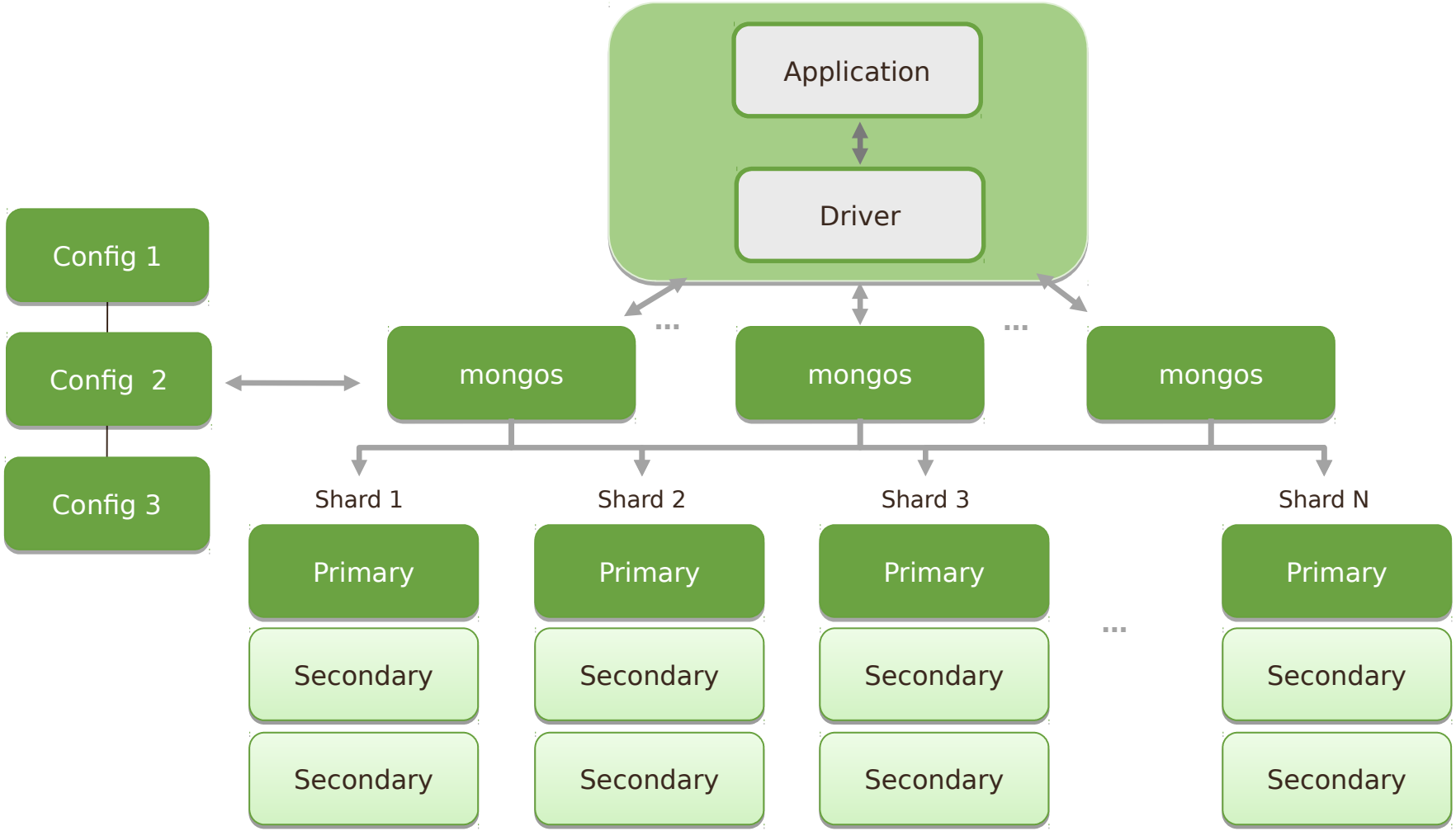
# MongoDB 分片技术原理



# MongoDB 分片技术原理



# 分片集群架构


Application
Driver
Config 1
Config 2
Config 3
mongos
mongos
mongos
Shard 1
Shard 2
Shard 3
Shard N
Primary
Secondary
Secondary


# 我什么时候该使用分片?

# 如果仅仅是性能扩展，不要太早分片

- 部署和维护成本高
- 额外开销

# 单机的性能指标：

- 磁盘
- 内存和工作集
- IOPS
- 并发


# 我需要多少个分片?

# 分片数量计算

A = 所需存储总量 / 单服务器可挂载容量

40TB / 2TB = 20

B = 工作集大小 / 单服务器内存容量

2TB / 64G = 32

C = 并发量总数 / （单服务器并发量 * 0.7）

20000 / (2500*0.7) = 11

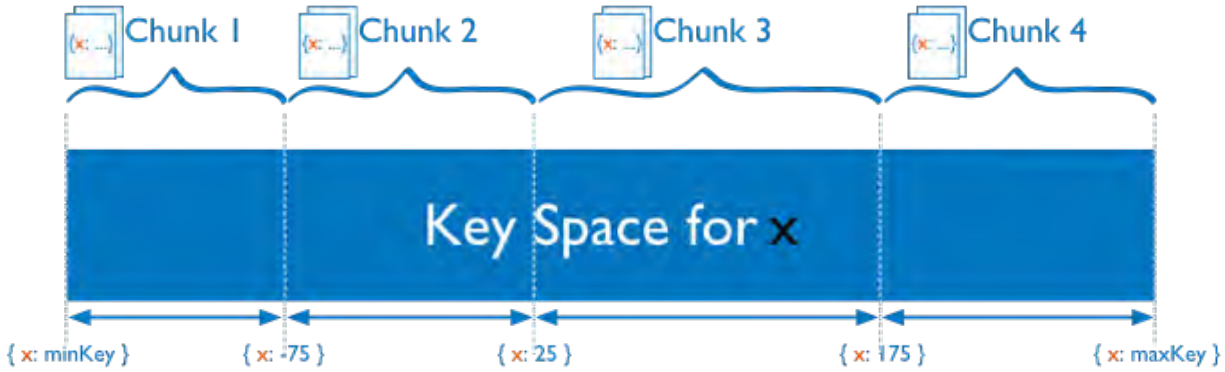
额外开销

分片数量 = max(A, B, C) = 32

# 我要用哪种方式分片?

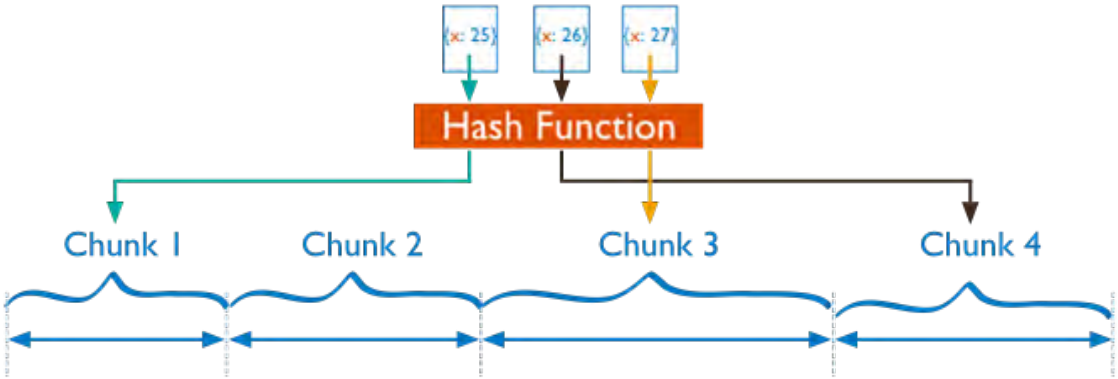
- 基于范围分片
- 基于哈希值
- 标签分片

# 基于值范围分片




| Pros | Cons |
| --- | --- |
| 片键范围查询性能好 | 数据分布可能不均匀 |
| 优化读 | 容易有热点 |

# 基于哈希分片




| Pros | Cons |
| --- | --- |
| 数据分布均匀，写优化 | 范围查询效率低 |
| 适用：日志，物联网等<br>高并发场景 | |

# 标签分片 - 定制数据分布

# 如何选择片键

# 片键的选择

- 基数要大
- 写操作分布均匀
- 查询定向性好

# Email 集合

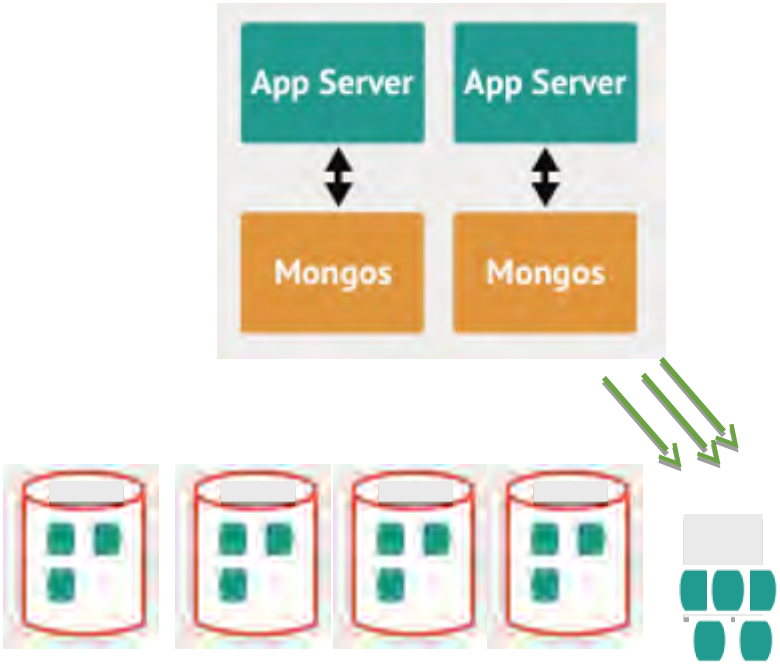
```
{
        _id: ObjectId(),
        user: 123,
        time: Date(),
        subject: “...”,
        recipients: [],
        body: “...”,
        attachments: []
}
```

# 片键： { _id: 1}

基数 ✓

写分布 ✗

定向查询 ✗

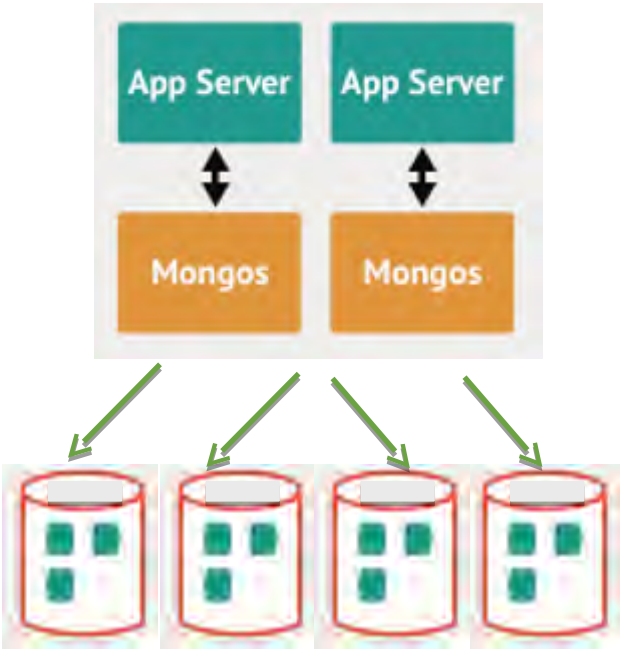


# 片键： { _id: “hashed”}

基数 ✓

写分布 ✓

定向查询 ✗

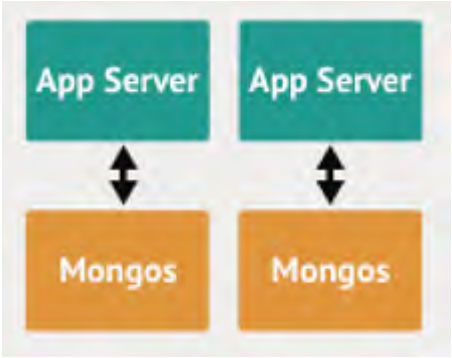


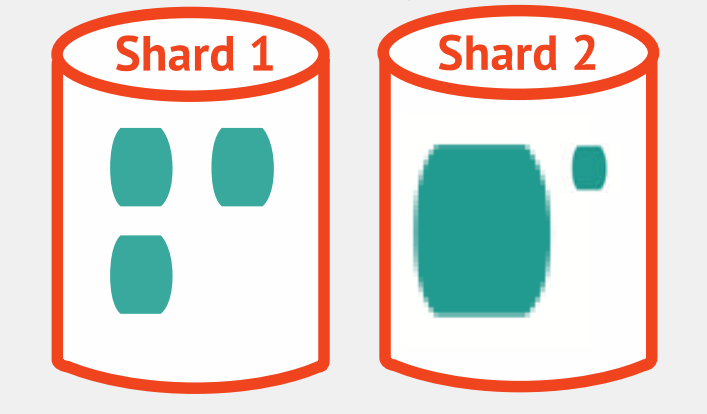
# 片键： { userid: 1}

基数 ✗

写分布 ✓

定向查询 ✓



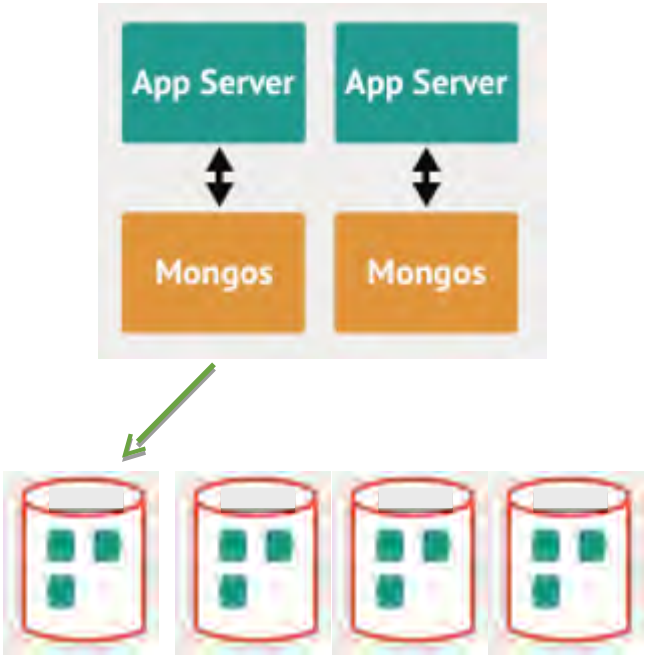



# 片键： { userid: 1, time:1}

**基数** ✓

**写分布** ✓

**定向查询** ✓



# 分片案例 – 为读性能扩展


- 130 亿条记录
- 12 个分片
- 9 成员复制集
- 3 个数据中心

# 分片案例 – 为海量数据扩展


- ~500 亿记录
- 20 分片

# 分片案例 – 为高并发扩展


- 30 万次 / 秒并发
- ~100 分片

# 软硬兼施

| | |
|---|---|
| 模式优化 | • 按数据访问方式优化模式<br>• 使用冗余优化查询 |
| 索引优化 | • 为重要的查询建索引<br>• 测试并选择最优索引 |
| 内存优化 | • 内存大于工作集大小 |
| IO 优化 | • SSD， RAID 10 |
| 分片 | • 万级操作和 TB/PB 级数据<br>• 选择好的片键 |

中文社区：www.mongoing.com

英文官网：www.mongodb.com