

# Memcached VS Redis

qianshi@taobao.com  
@淘宝千石

# 使用开源软件的四要素

## 应用场景

*首先搞清楚用在什么地方，解决什么问题。*

## 软件特性

*有哪些优秀的特性和自己的需求契合。*

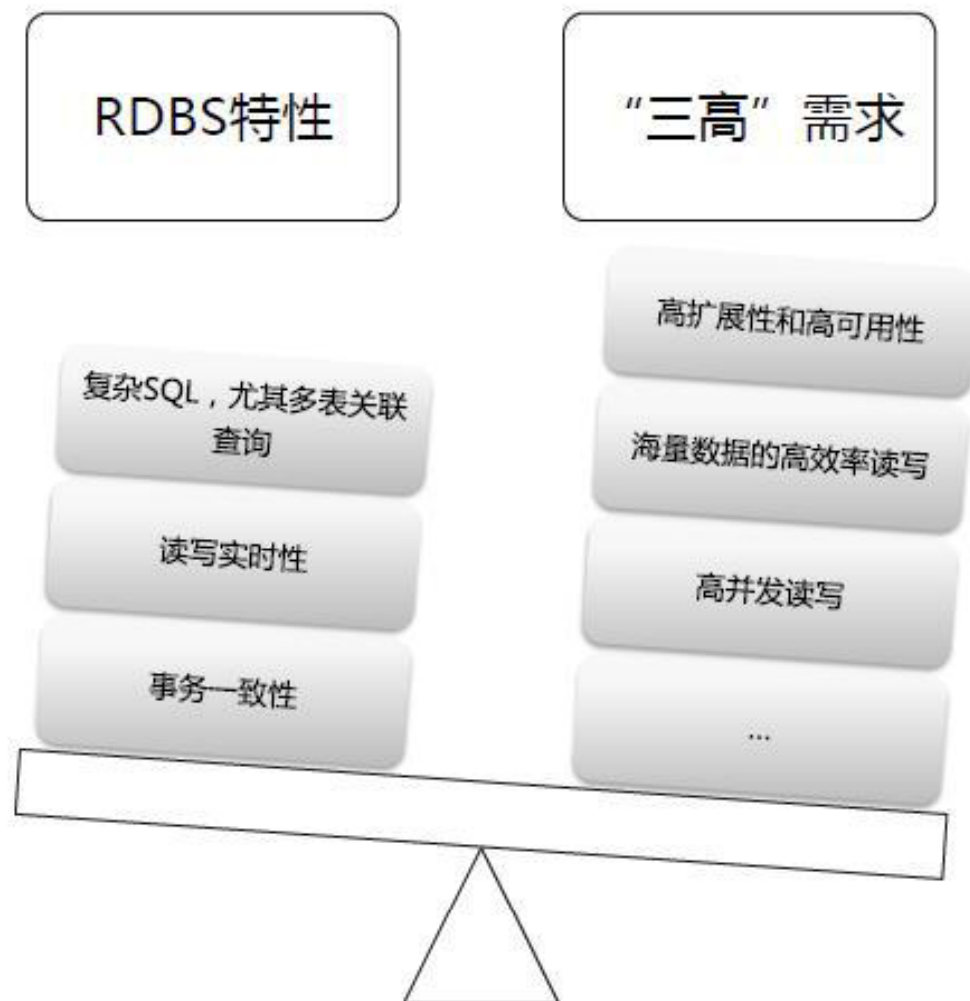
## 部署方案

*延续前人的成功，但不要重复他们的失败。*

## 应用tips

*魔鬼都在细节中，不了解她请不要说爱她。*

# 时代的主题：web 2.0 三高



**Jim Gray**为我们指明了方向

*Memory is the new Disk, Disk is the new Tape.*

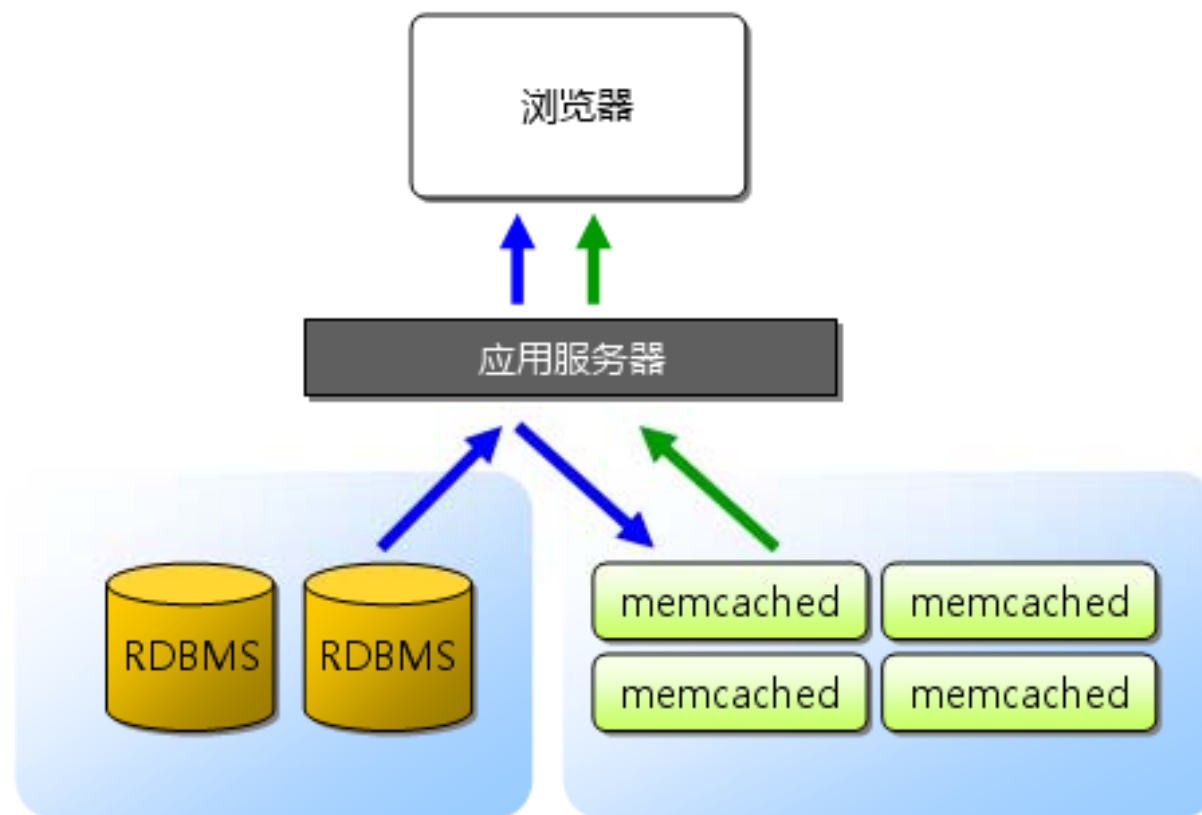


# Memcached

*A high-performance, distributed **memory object caching system.***

*Intended for use in speeding up dynamic web applications by **alleviating database load.***

# 应用场景



- ➡ 首次访问：从RDBMS中取得数据保存到memcached
- ➡ 第二次后：从memcached中取得数据显示页面

# 软件特性

协议简单

基于libevent的事件处理

内置内存存储方式

不互相通信的分布式



# 软件特性：内存模型

**Slab Class: 1**

**Chunks:**

88 bytes

88 bytes

88 bytes

... lots more!

**Slab Class: 2**

**Chunks:**

112 bytes

112 bytes

112 bytes

... lots more!

**Slab Class: 3**

**Chunks:**

144 bytes

144 bytes

144 bytes

... lots more!

**Slab Class: n**

**Chunks:**

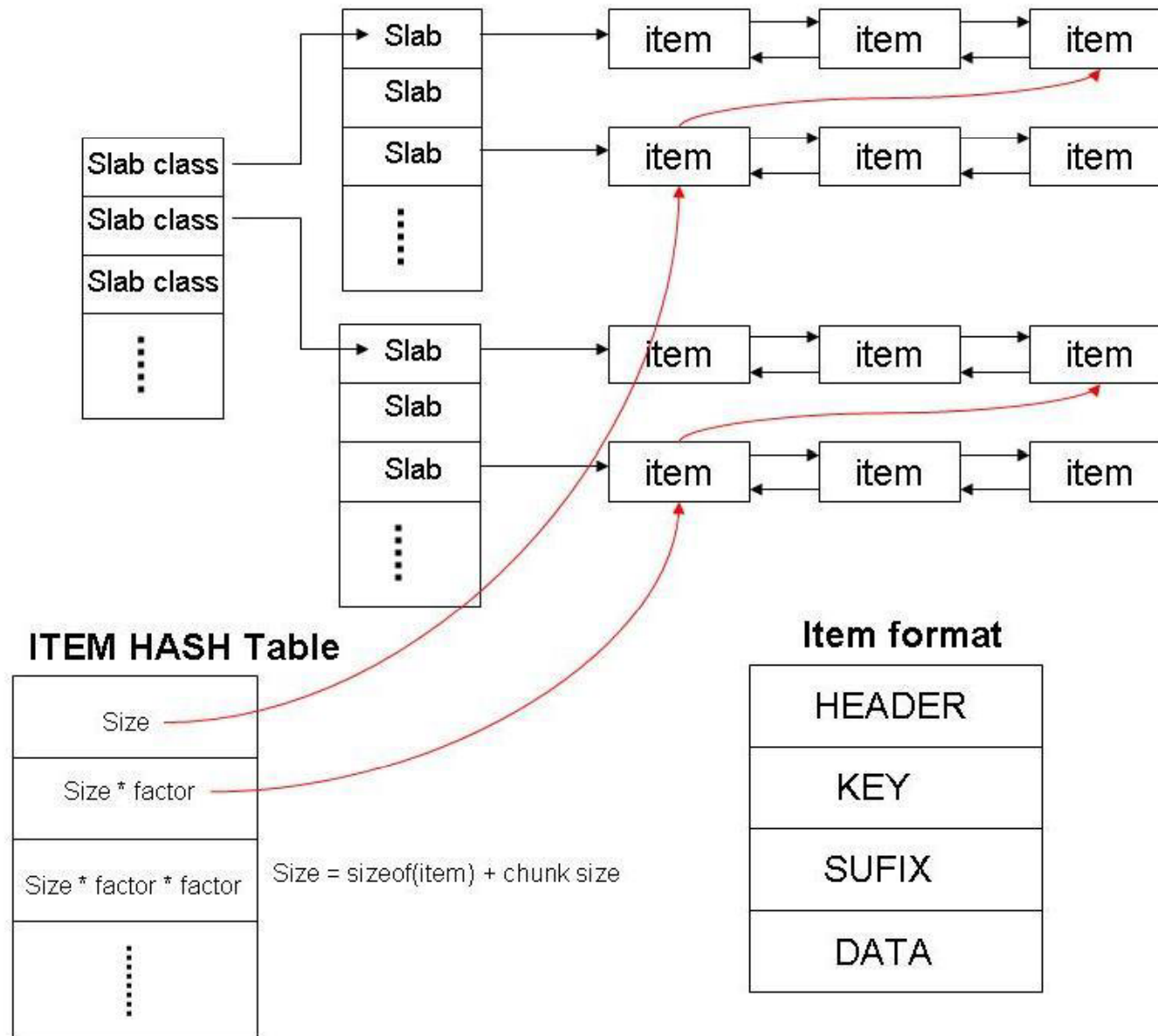
*n* bytes

*n* bytes

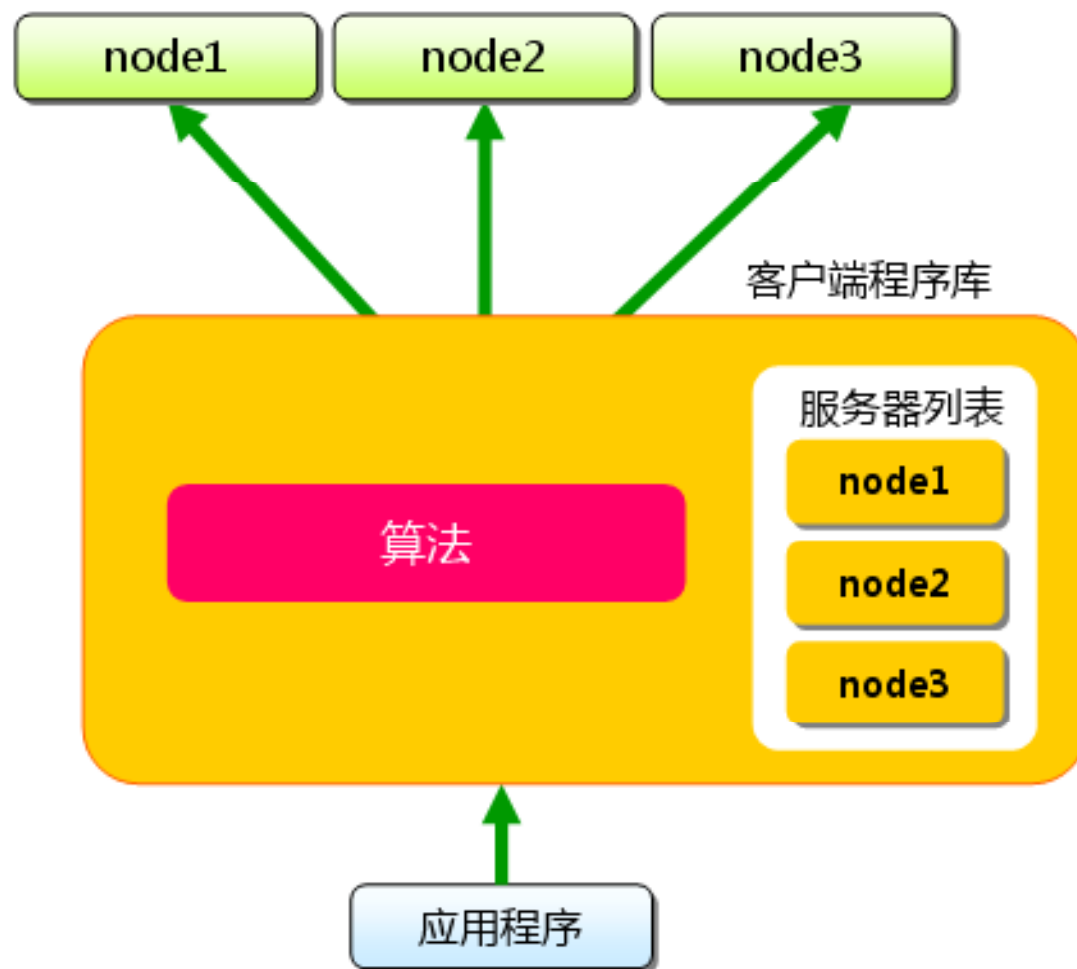
*n* bytes

... more!

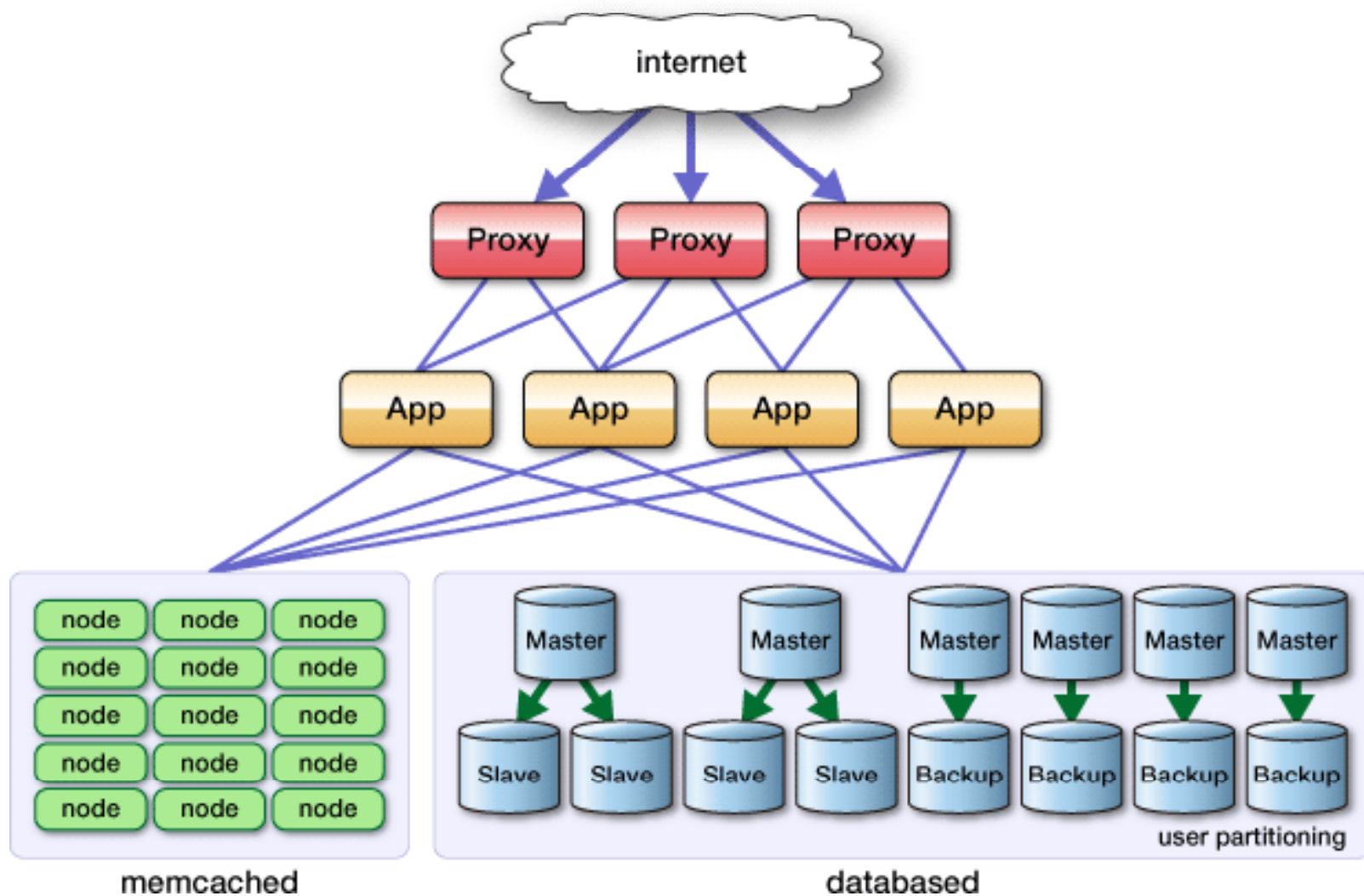
# 软件特性：内存模型（续）



## 软件特性：基于客户端的分布式



# 部署方案



## 应用tips

`/usr/bin/memcached -p 11211 -u nobody -m 3000 -c 30720`

4GB物理内存的系统最大为memcached配置3GB内存可以保证不会有swap。

memcached 服务器几乎不占用CPU.

宕机重启: `daemontools`

持久化: `memcachedb`: 存储到BerkleyDB。

## 应用**tips**（续）

*Consistent Hashing*: 避免牵一发而动全身。

*Lazy Expiration*: 高效就是要少做事。

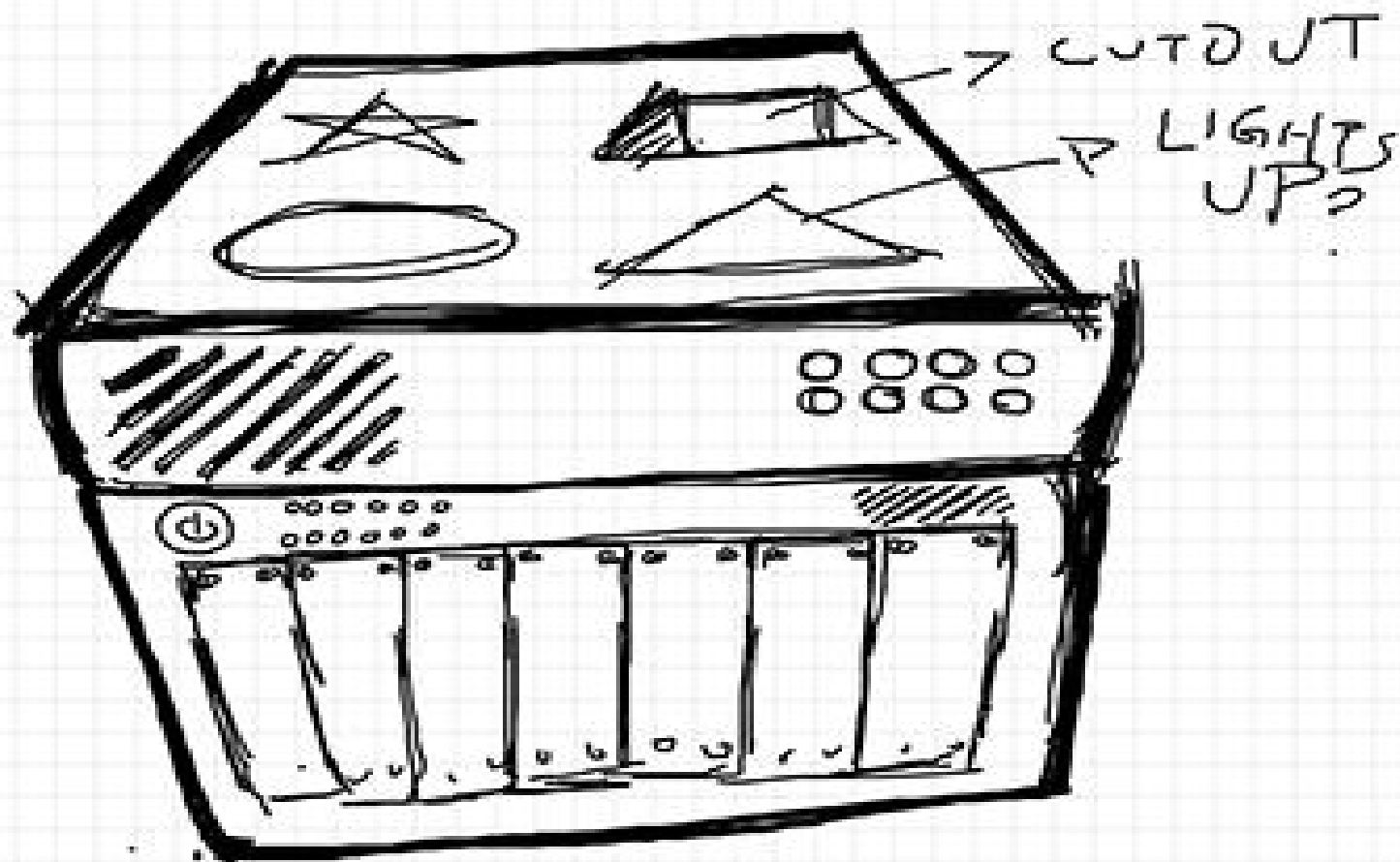
*LRU*: 从缓存中有效删除数据的原理。

*应用层的CAS*: 解决并发问题。



redis

# 设计草稿





# Redis

*Redis is an open source, advanced  
key-value store.*

*It is often referred to as a data structure  
server since keys can contain strings,  
hashes, lists, sets and sorted sets.*

*Redis is **a collection of data structures**  
exposed over the network.*

# 应用场景

取最新N个数据的操作

排行榜应用，取TOP N操作

需要精准设定过期时间的应用

计数器应用

Uniq操作，获取某段时间所有数据排重值

实时系统，反垃圾系统

Pub/Sub构建实时消息系统

构建队列系统

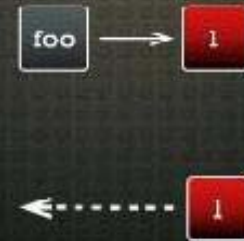
缓存

# redis-string

```
> set foo bar  
OK
```



```
> incr foo  
(integer) 1
```



```
> get foo  
"bar"
```



```
> decr foo  
(integer) 0
```



# redis-list

```
> rpush foo baz  
(integer) 1
```



```
> rpush foo qux  
(integer) 2
```



```
> lpush foo bar  
(integer) 3
```



```
> lrange foo 0 -1  
1) "bar"  
2) "baz"  
3) "qux"
```



```
> lpop foo  
"bar"
```



```
> rpop foo  
"qux"
```



# redis-hash

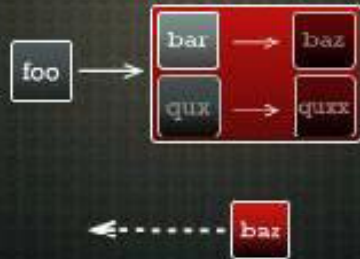
```
> hset foo bar baz  
(integer) 1
```



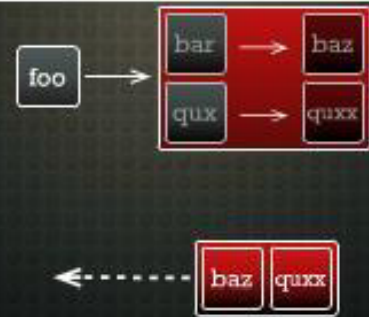
```
> hset foo qux quxx  
(integer) 1
```



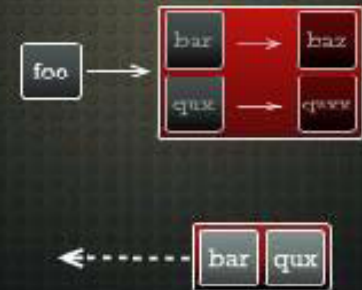
```
> hget foo bar  
"baz"
```



```
> hvals foo  
1) "baz"  
2) "quxx"
```



```
> hkeys foo  
1) "bar"  
2) "qux"
```



# redis-set

```
> sadd m1 jan  
(integer) 1
```




A diagram showing a box labeled 'm1' with an arrow pointing to a red box containing the text 'jan'.

```
> sadd m1 feb  
(integer) 1
```



A diagram showing a box labeled 'm1' with an arrow pointing to a red box containing the text 'feb' and 'jan' stacked vertically.

```
> sismember m1 jan  
(integer) 1
```



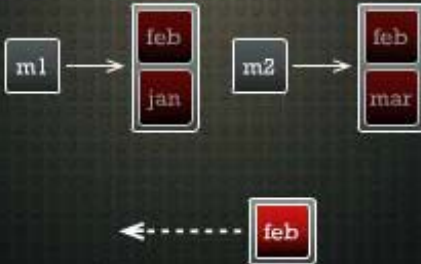
A diagram showing a box labeled 'm1' with an arrow pointing to a red box containing the text 'feb' and 'jan' stacked vertically. Below it, a dashed arrow points from the text '1' in a red box.

```
> smembers m1  
1) "feb"  
2) "jan"
```



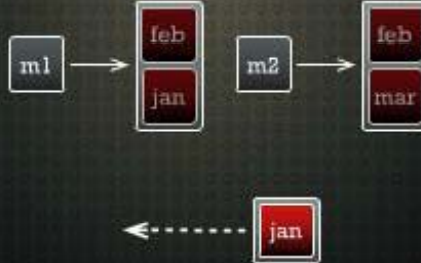
A diagram showing a box labeled 'm1' with an arrow pointing to a red box containing the text 'feb' and 'jan' stacked vertically. Below it, a dashed arrow points from a red box containing the text 'feb' and 'jan' stacked vertically.

```
> sinter m1 m2  
1) "feb"
```



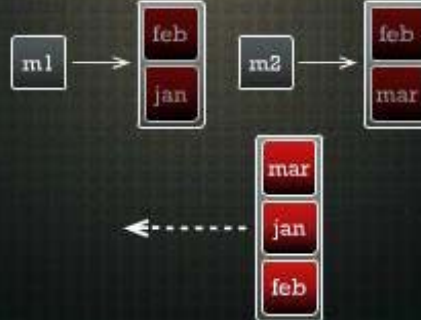
A diagram showing two boxes labeled 'm1' and 'm2'. 'm1' points to a red box with 'feb' and 'jan'. 'm2' points to a red box with 'feb' and 'mar'. Below them, a dashed arrow points from a red box containing the text 'feb'.

```
> sdiff m1 m2  
1) "jan"
```



A diagram showing two boxes labeled 'm1' and 'm2'. 'm1' points to a red box with 'feb' and 'jan'. 'm2' points to a red box with 'feb' and 'mar'. Below them, a dashed arrow points from a red box containing the text 'jan'.

```
> sunion m1 m2  
1) "mar"  
2) "jan"  
3) "feb"
```




A diagram showing two boxes labeled 'm1' and 'm2'. 'm1' points to a red box with 'feb' and 'jan'. 'm2' points to a red box with 'feb' and 'mar'. Below them, a dashed arrow points from a red box containing the text 'mar', 'jan', and 'feb' stacked vertically.



# redis-zset

Redis REPL


```
> zadd z1 1 jan  
(integer) 1
```



A diagram showing a variable 'z1' in a box with an arrow pointing to a vertical container. Inside the container, there is a single entry with a score of '1' and a member 'jan'.

Redis REPL

```
> zrange z1 0 1 withscores  
1) "jan"  
2) "1"  
3) "feb"  
4) "2"
```



A diagram showing a variable 'z1' in a box with an arrow pointing to a vertical container. The container has three entries: '1' and 'jan', '2' and 'feb', and '3' and 'mar'.

Redis REPL

```
> zscore z1 feb  
"2"
```



A diagram showing a variable 'z1' in a box with an arrow pointing to a vertical container. The container has three entries: '1' and 'jan', '2' and 'feb', and '3' and 'mar'. A dashed arrow points from the 'feb' member to a box containing the score '2'.

Redis REPL

```
> zrangebyscore z1 2 3 withscores  
1) "feb"  
2) "2"  
3) "mar"  
4) "3"
```



A diagram showing a variable 'z1' in a box with an arrow pointing to a vertical container. The container has three entries: '1' and 'jan', '2' and 'feb', and '3' and 'mar'. A dashed arrow points from the 'feb' member to a box containing the score '2'.



# 软件特性

*In-memory storage*

*Super fast*

*Persistence*

*High level data types*

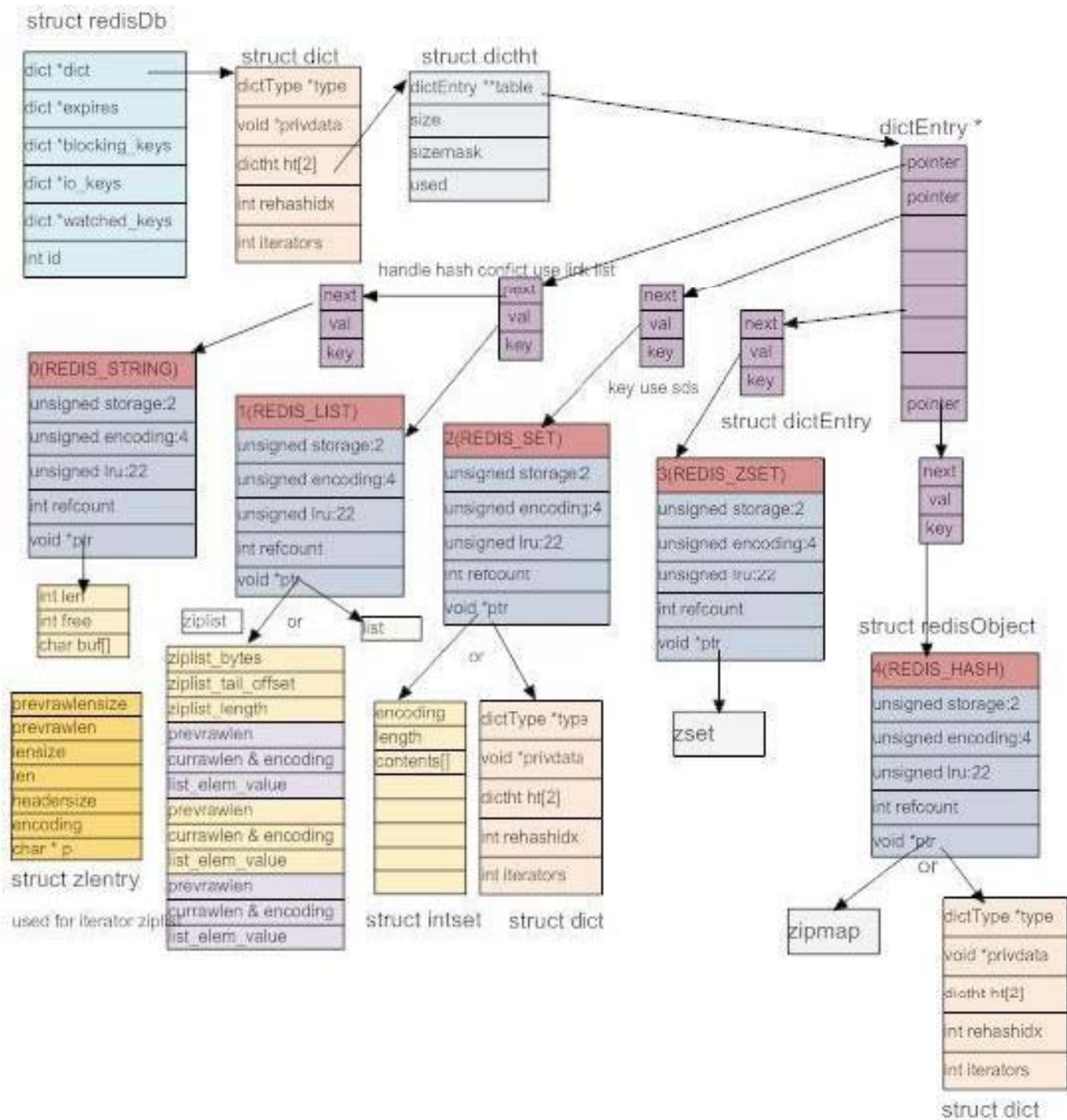
*Atomic*

*Replication, Sharding*

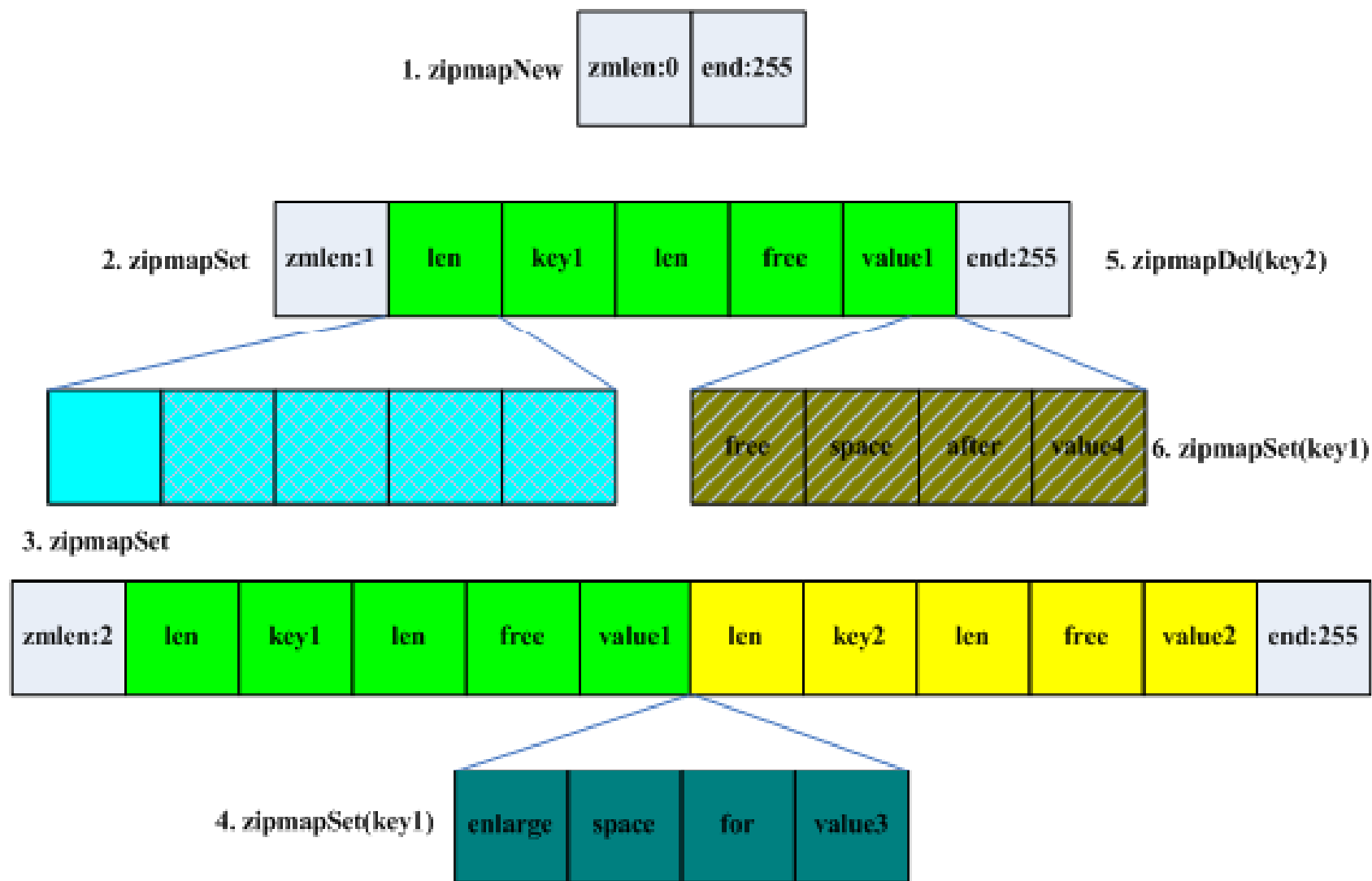
*Single-threaded*

*No dependencies*

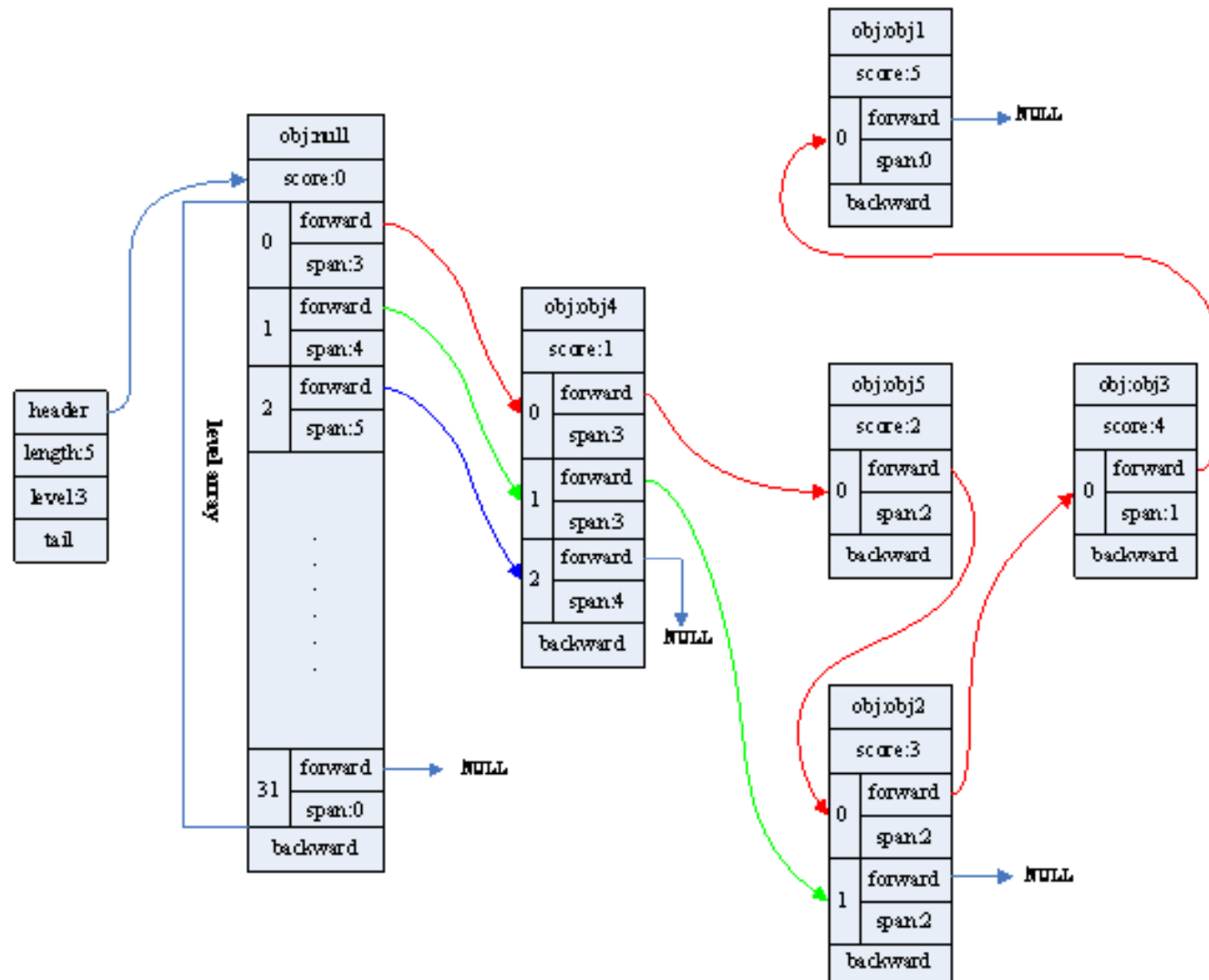
## 软件特性：内存模型



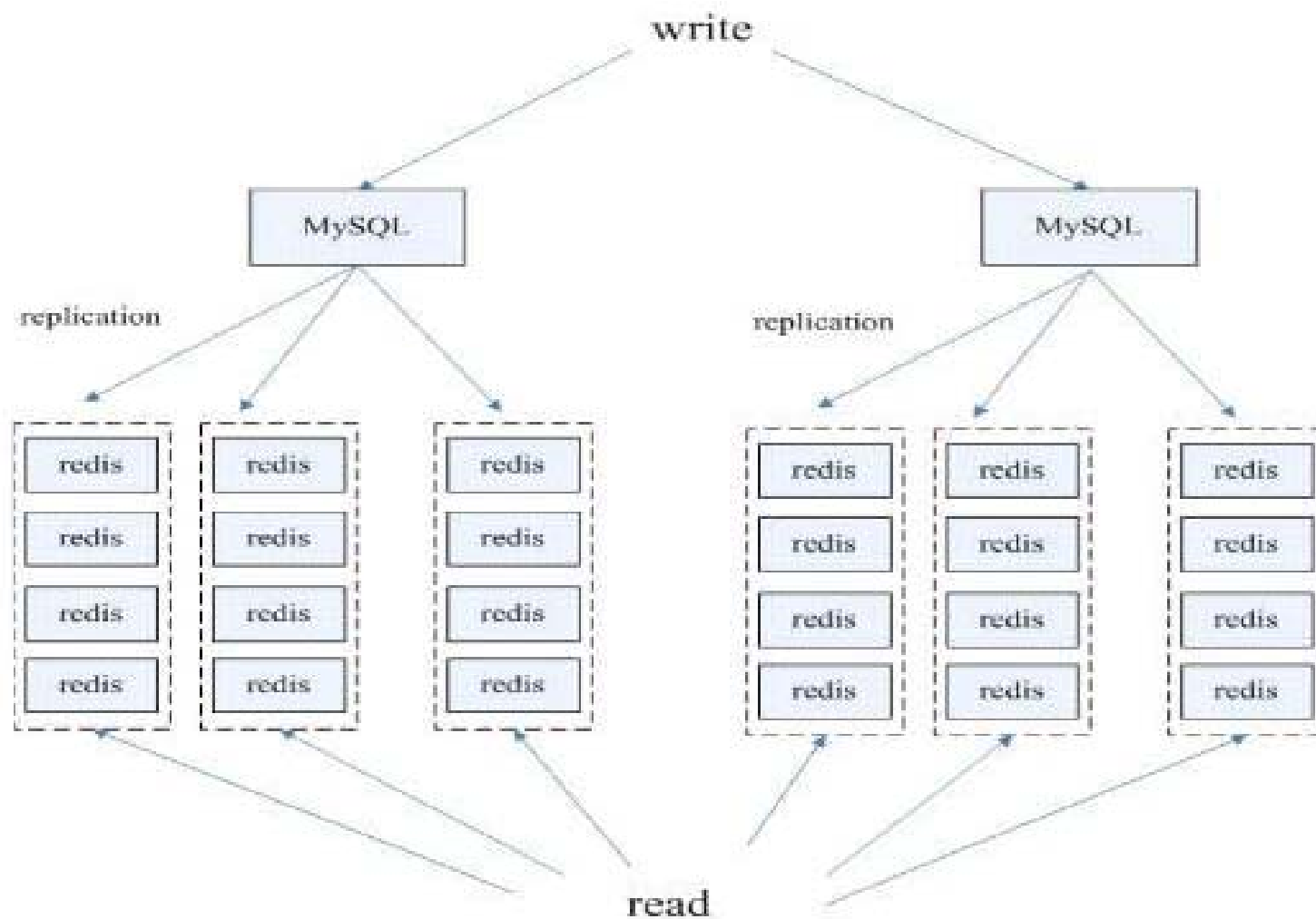
# 软件特性: zipmap结构省内存



# 软件特性: **skiplist**实时排序



## 部署方案：异构读写分离



## 应用**tips**

做好容量规划，保证In-memory。

不要过度依赖复制和持久化。

使用pipeline减小网络IO开销。

小心Redis的内存碎片。

# memcached VS redis

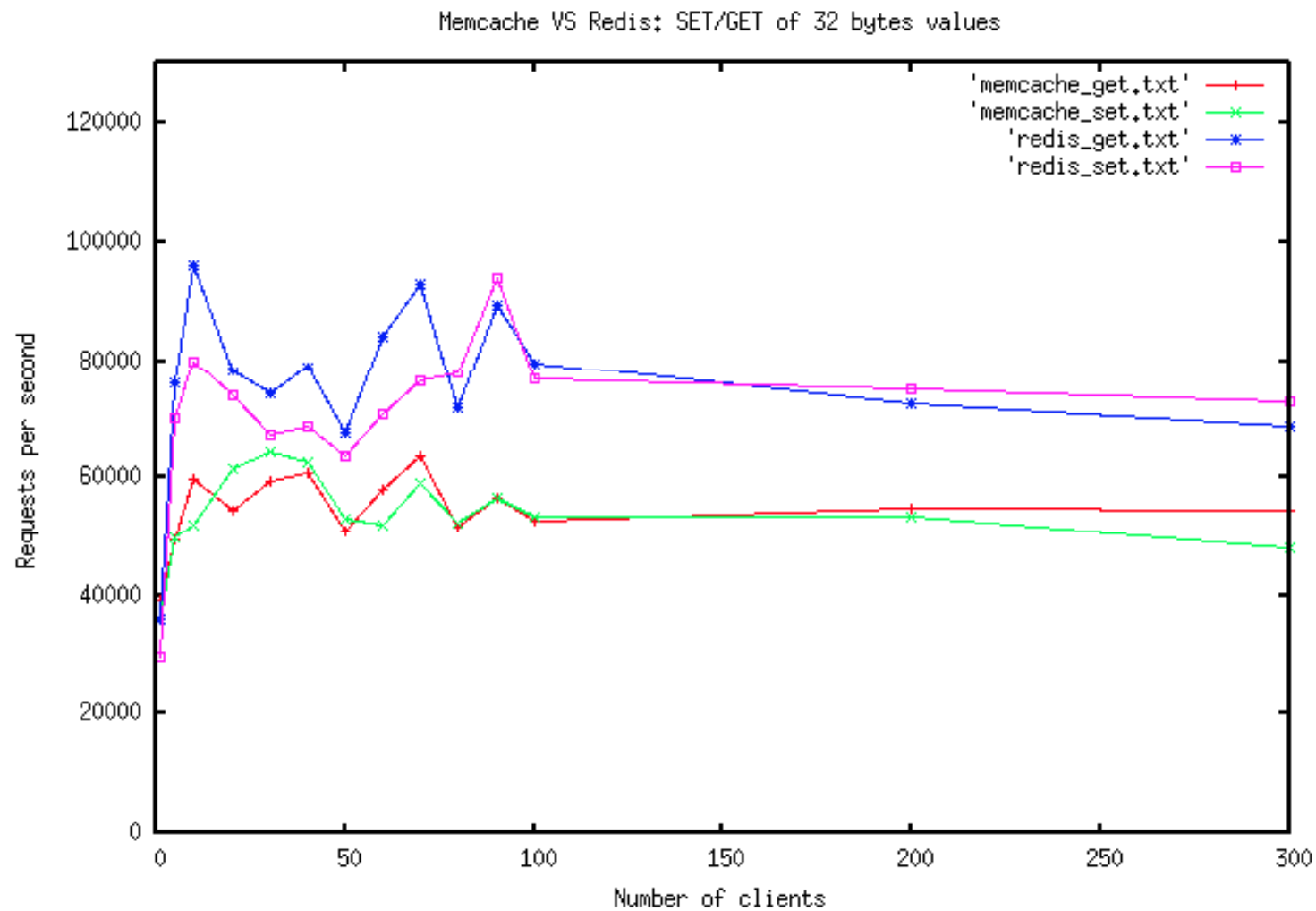
*memcached多线程，redis单线程，但处理qps都不会是瓶颈。*

*Redis在存储小数据时比Memcached性能更高。而在100k以上的数据中，Memcached性能要高于Redis。*

*redis支持数据持久化和数据同步。*

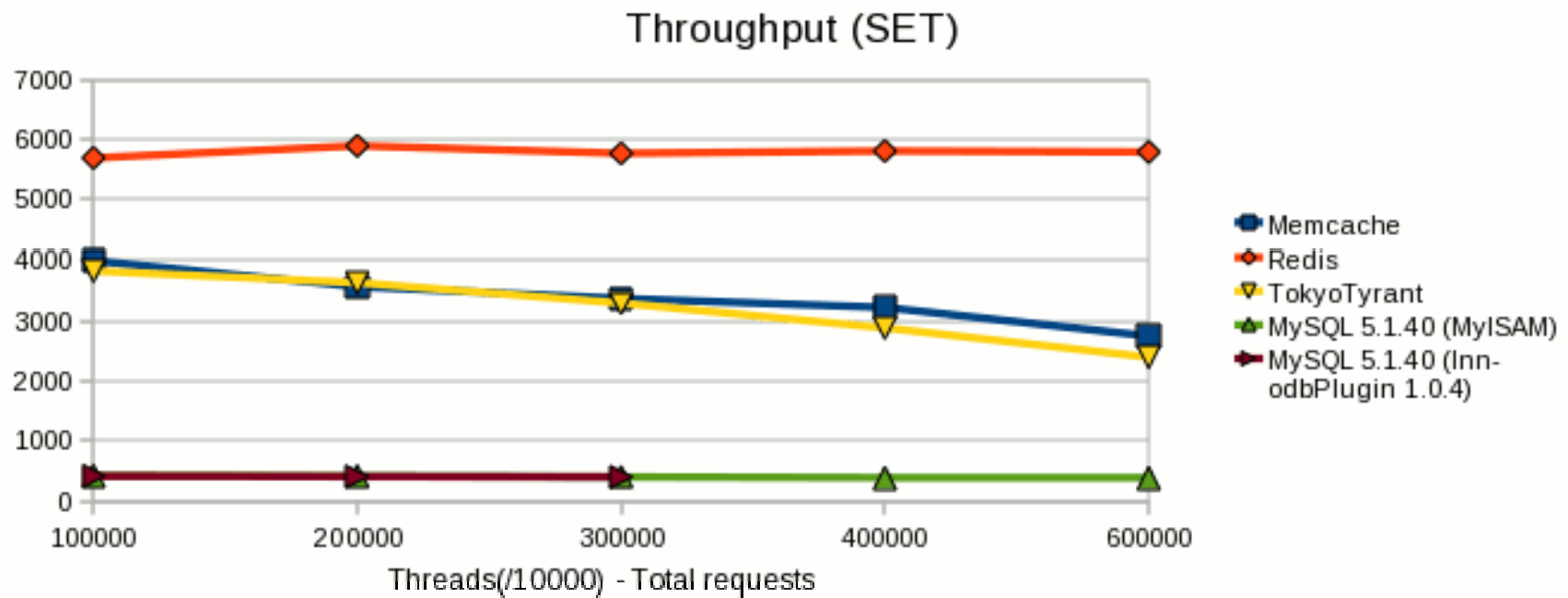
*redis拥有更多的数据结构和并支持更丰富的数据操作。*

# memcached vs redis

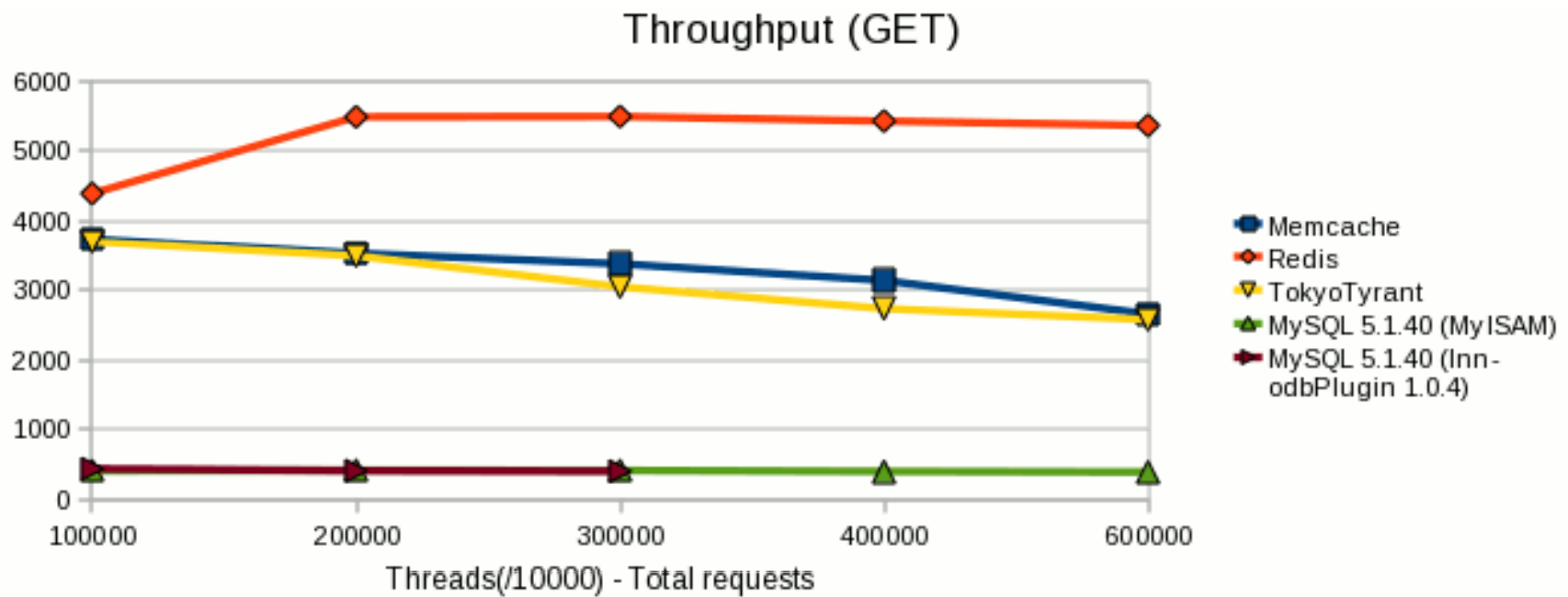




# memcached vs redis



# memcached vs redis



# The end

*尽量少的让计算机干重复的事情, cache everywhere.*

*再高效的系统也经不起滥用, to be internal.*

*No one-size-fit-all product, understanding the trade offs.*

*RTFC first, then using.*

# **nosql**资料合集

<http://blog.nosqlfan.com>