

ArchSummit

International Architect Summit

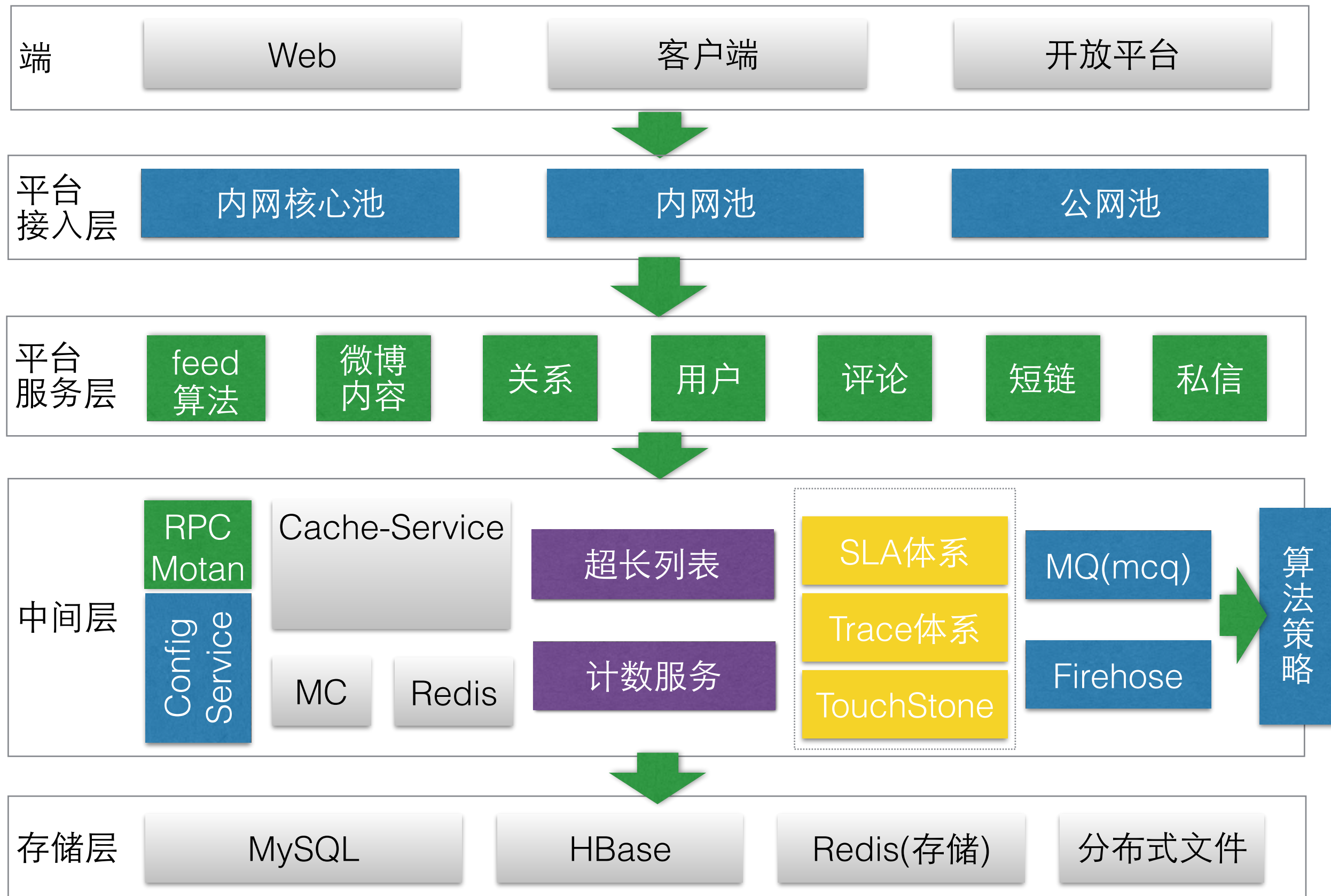
全球架构师峰会

2014年12月19-20日
北京国际会议中心

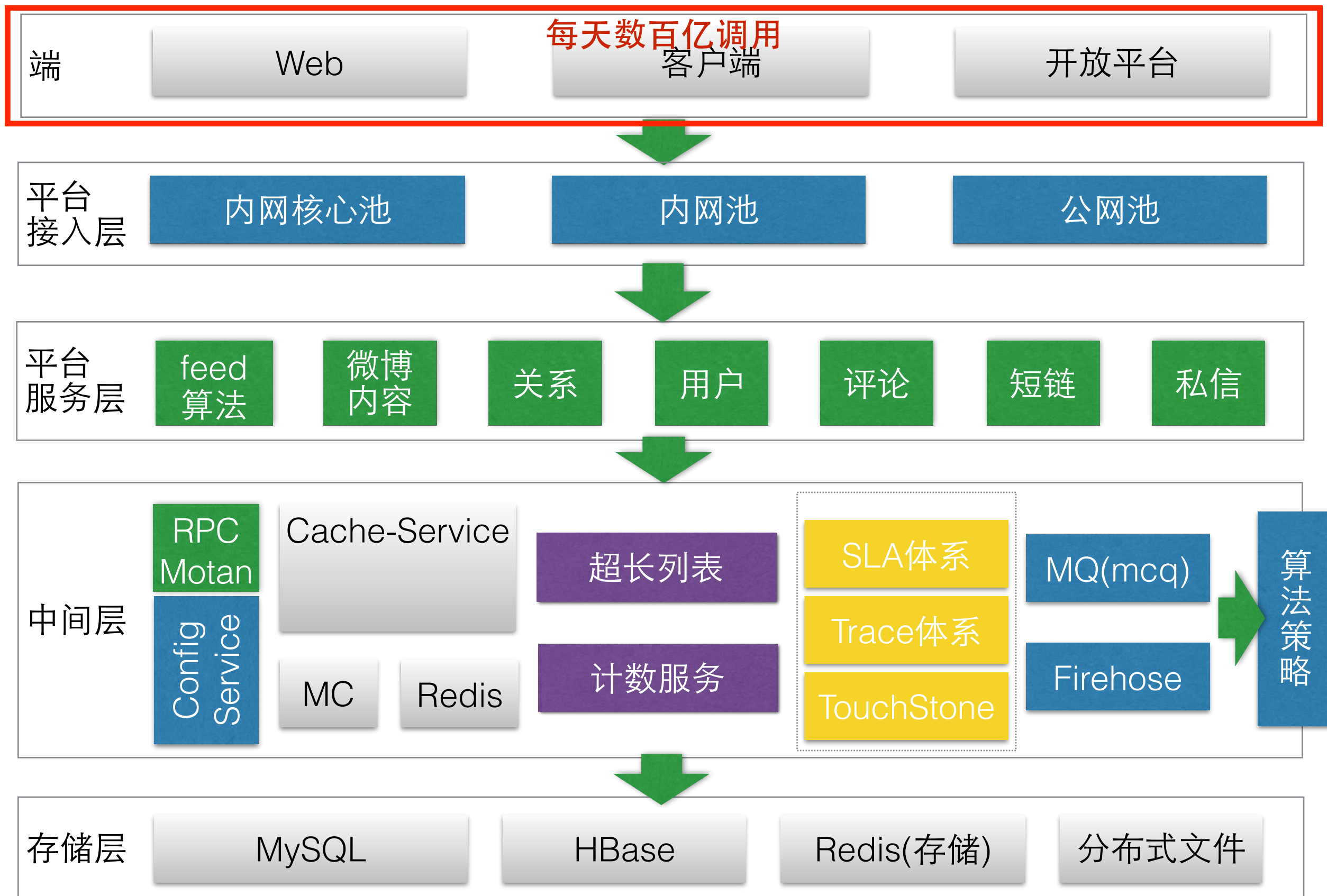
大数据时代 feed架构

新浪微博 @TimYang

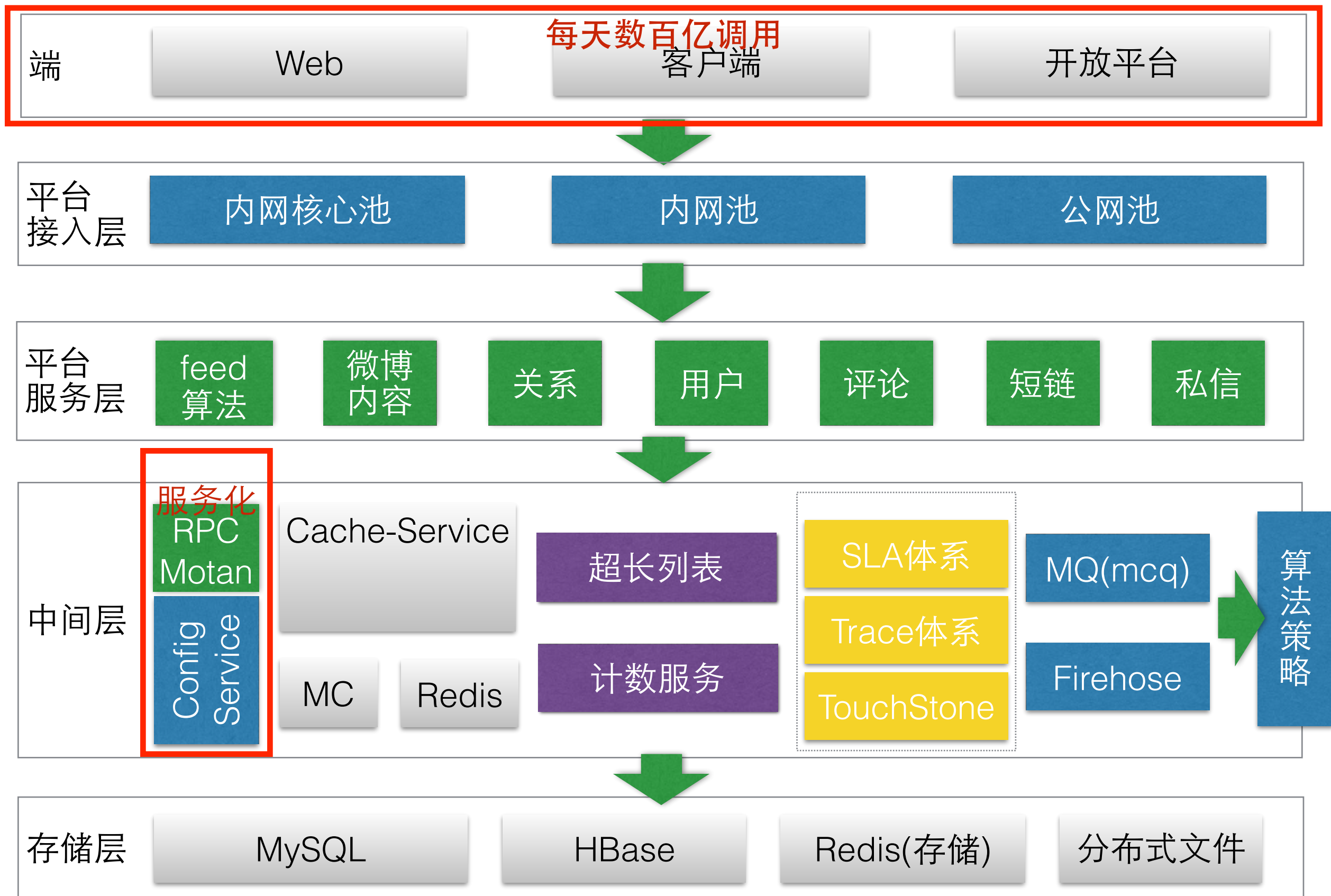
当前架构



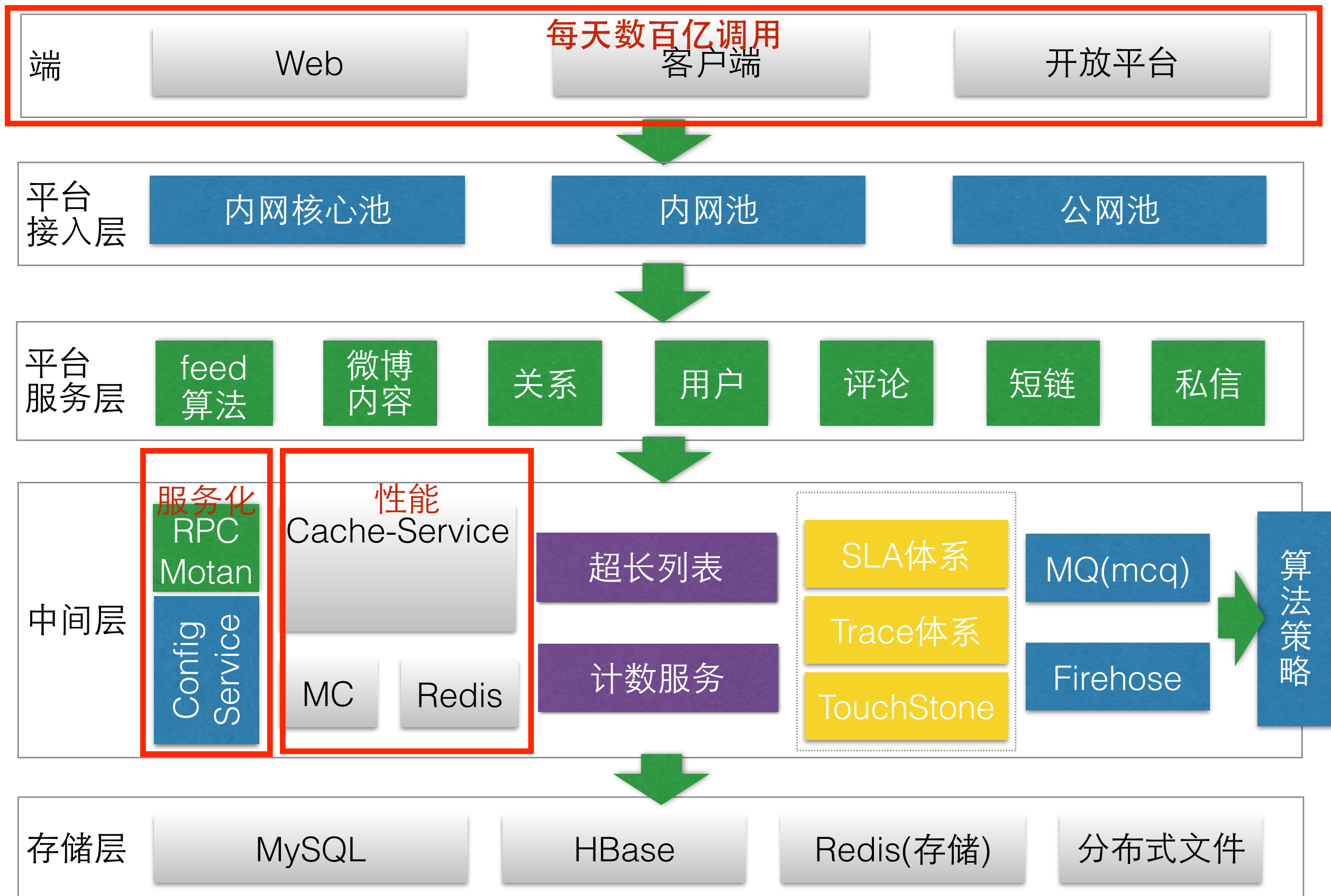
当前架构



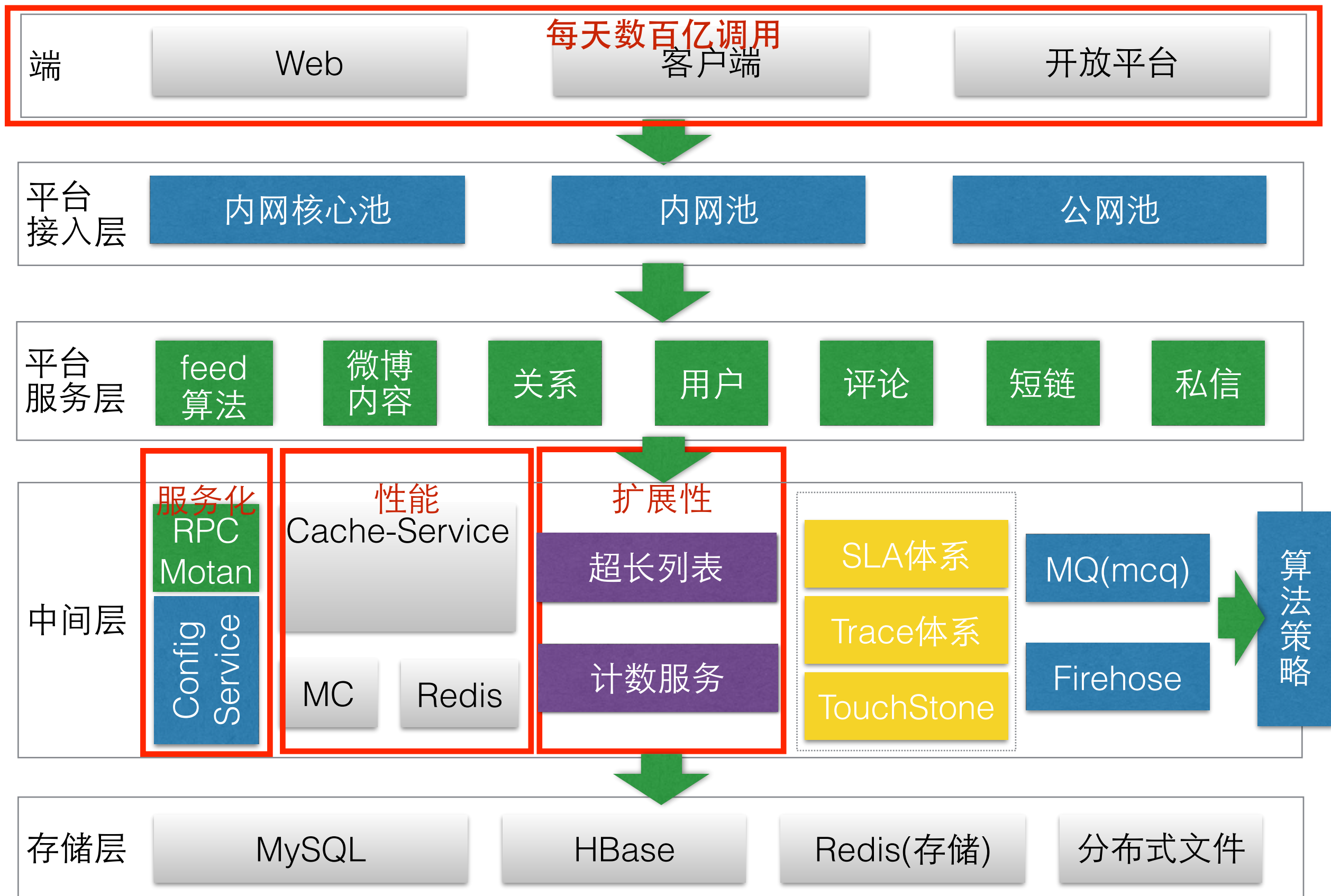
当前架构



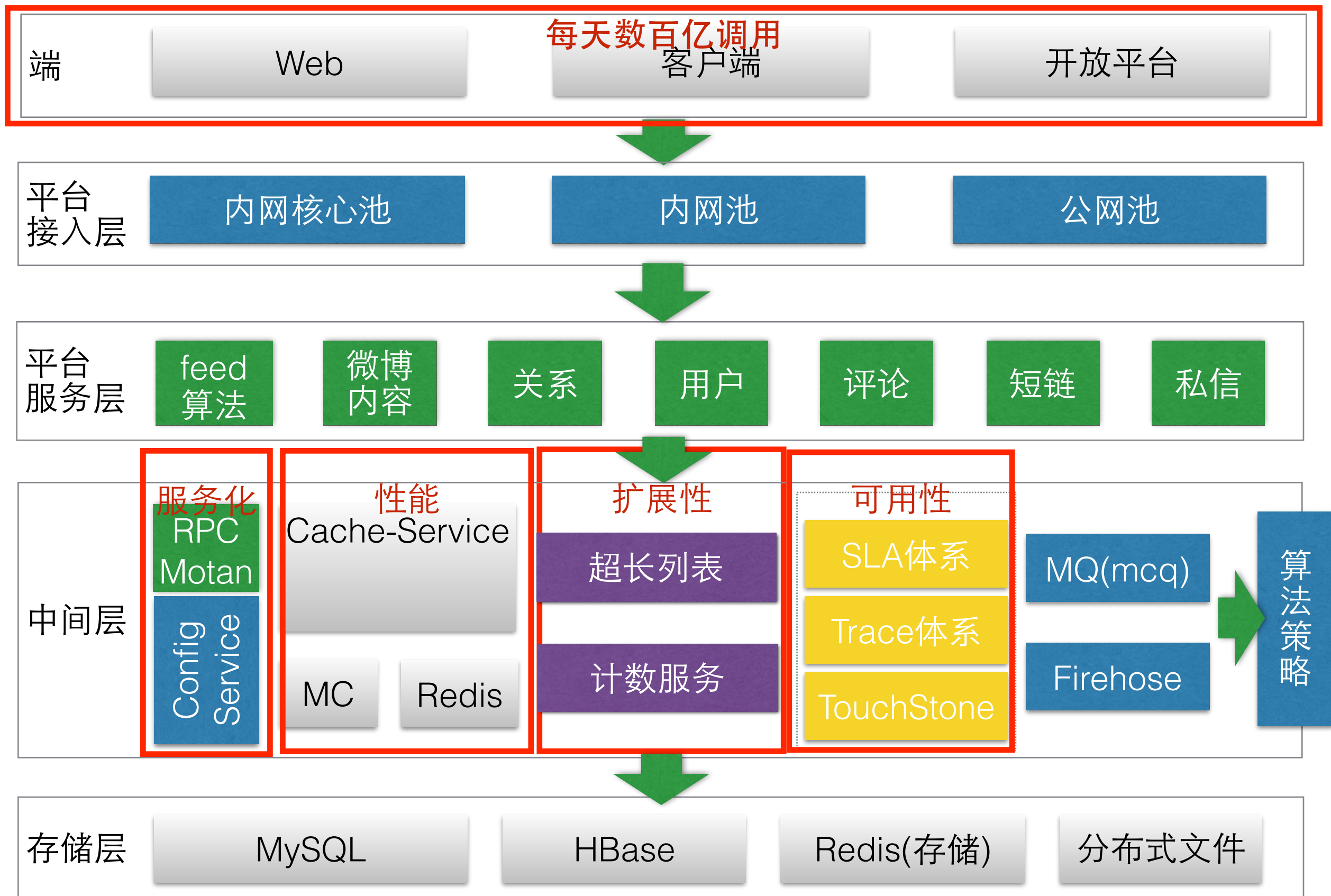
当前架构



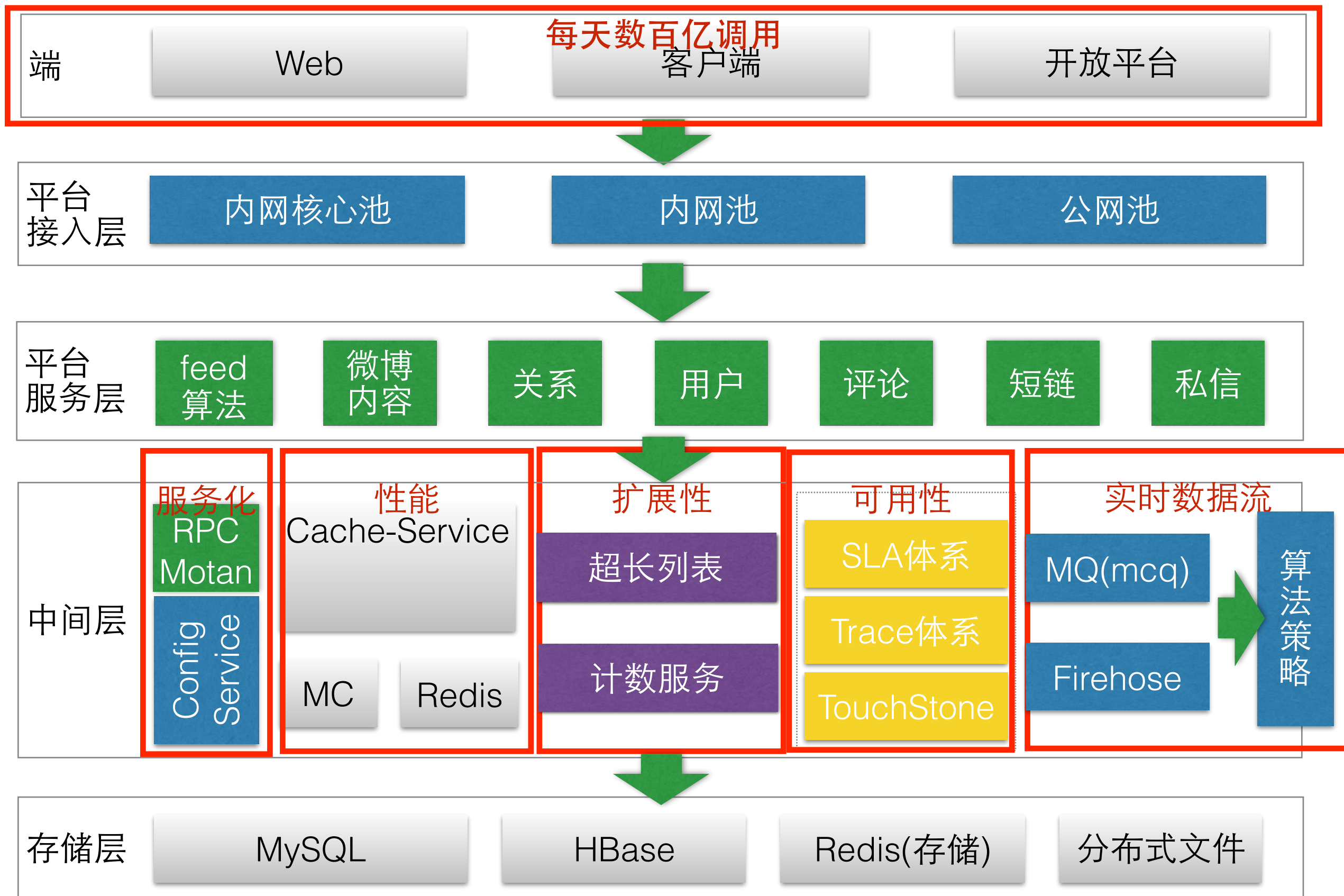
当前架构



当前架构



当前架构



架构特点

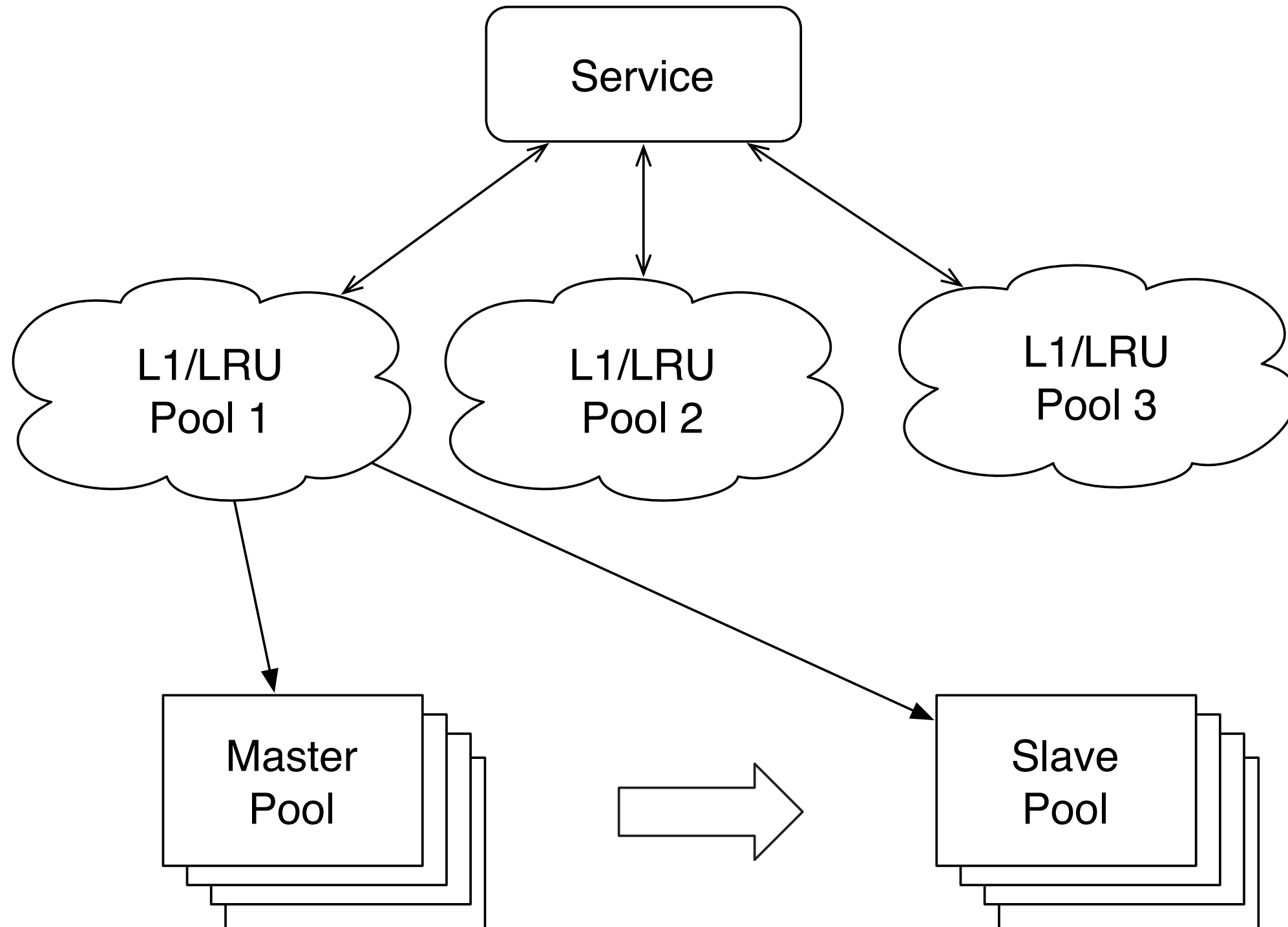
- ✓ 解决了数据规模大且超长LIST访问的问题
 - ✓ MySQL sharding by time range
- ✓ 解决了数据存储可扩展的问题
 - ✓ 2~3 years
- ✓ 解决了百万QPS访问的问题
 - ✓ Cache replication及分级
- ✓ 解决了可用性及错误隔离问题
 - ✓ by SLA体系，核心功能 99.99%+

读写比例高	10:1读写比以上
冷热数据明显	80%访问的是当天内的数据
存在热点问题	峰值写入80万/分钟 (2014元旦)
高访问量	每天超过7000万用户访问 (2014Q3数据)

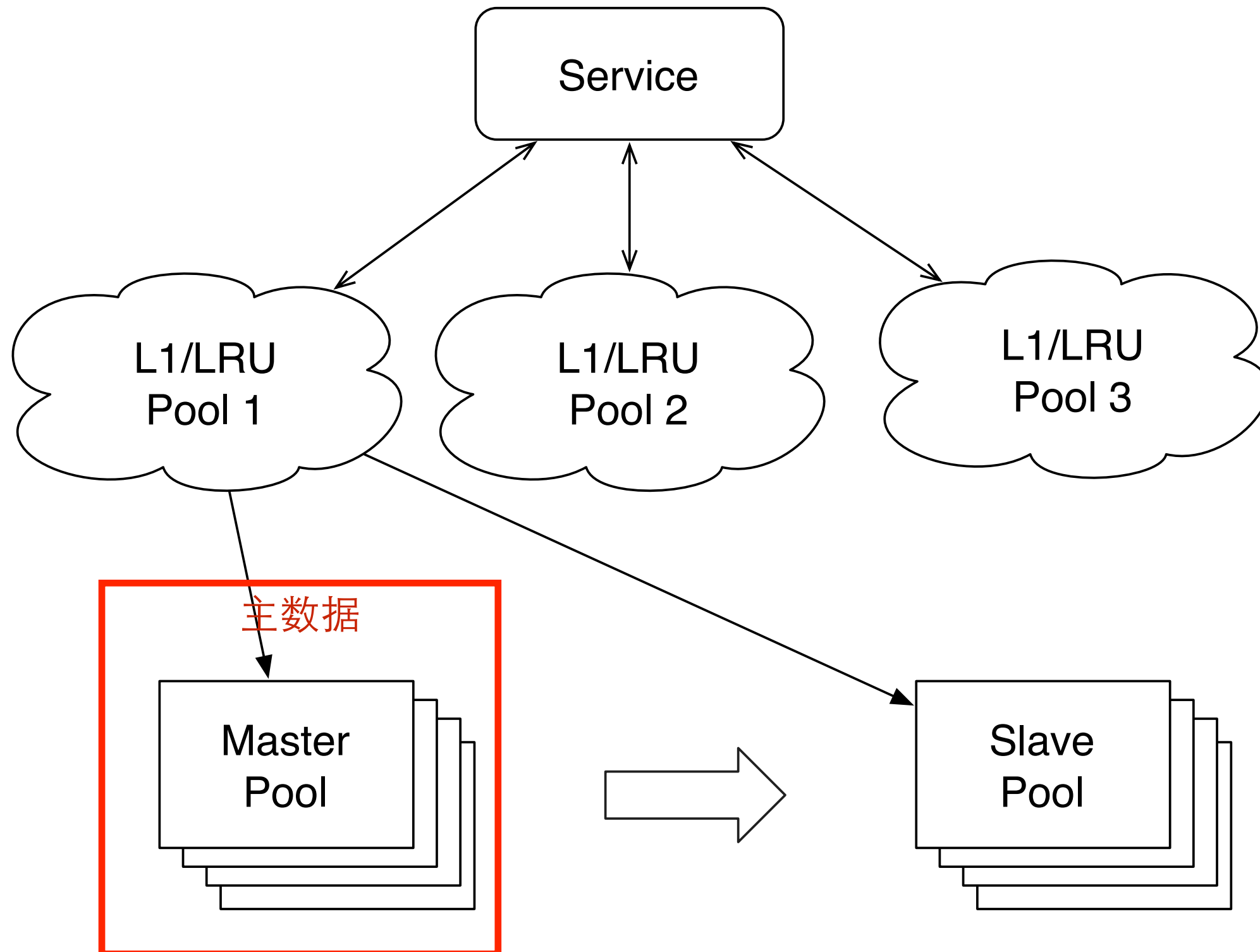
读写比例高	10:1读写比以上
冷热数据明显	80%访问的是当天内的数据
存在热点问题	峰值写入80万/分钟 (2014元旦)
高访问量	每天超过7000万用户访问 (2014Q3数据)

大数据环境的性能解决之道 — 缓存

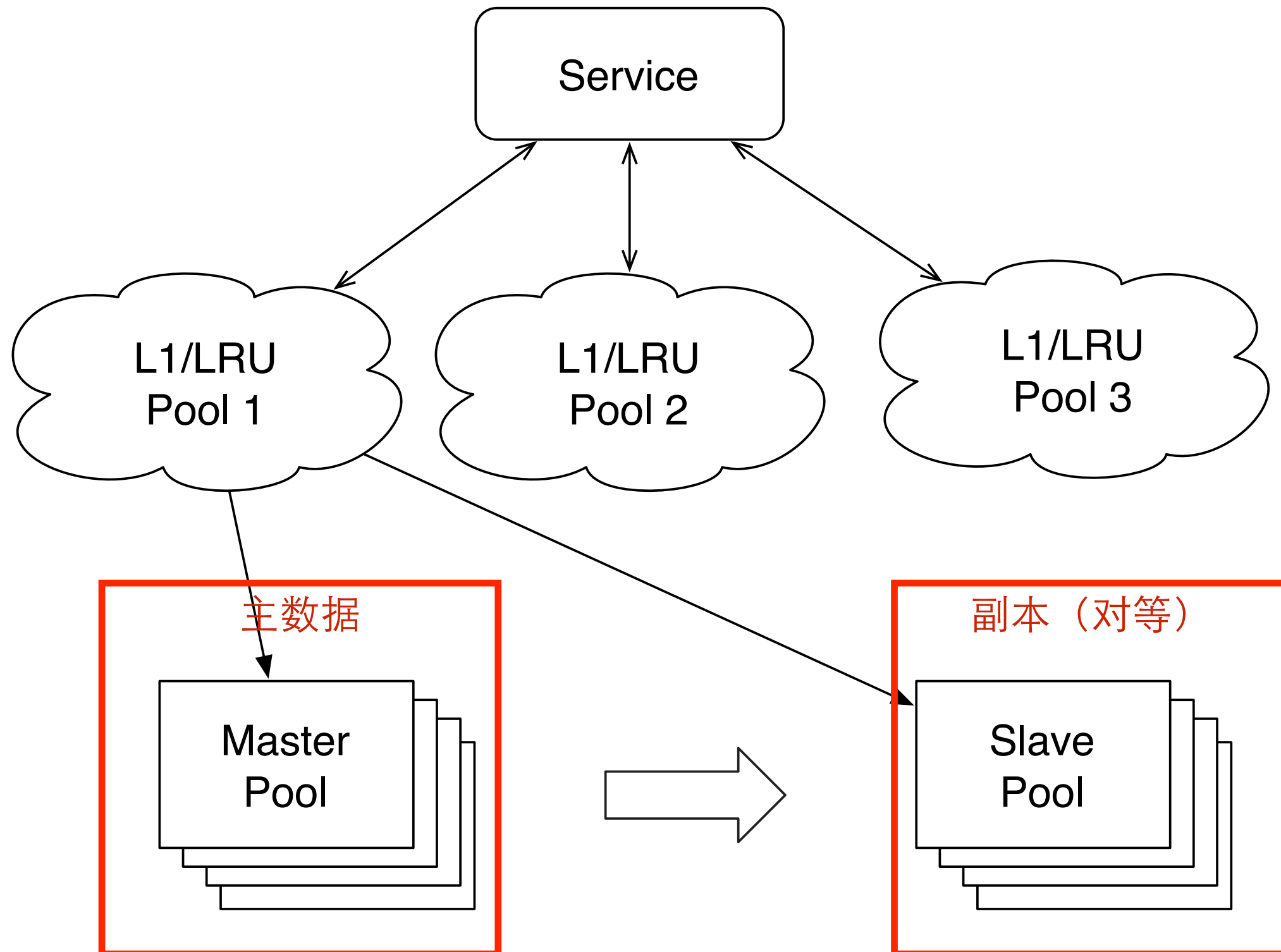
大数据环境的 性能解决之道



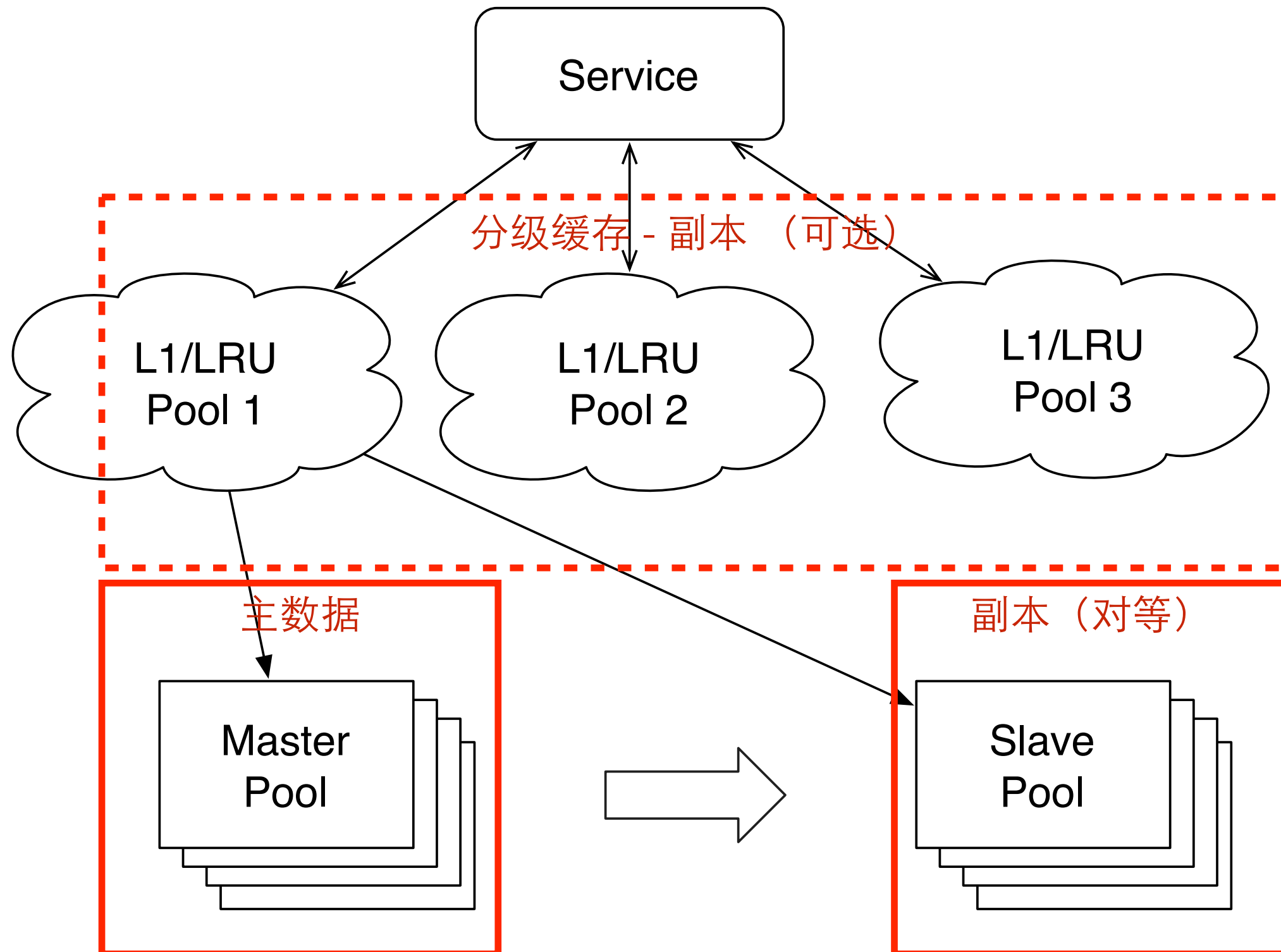
大数据环境的 性能解决之道



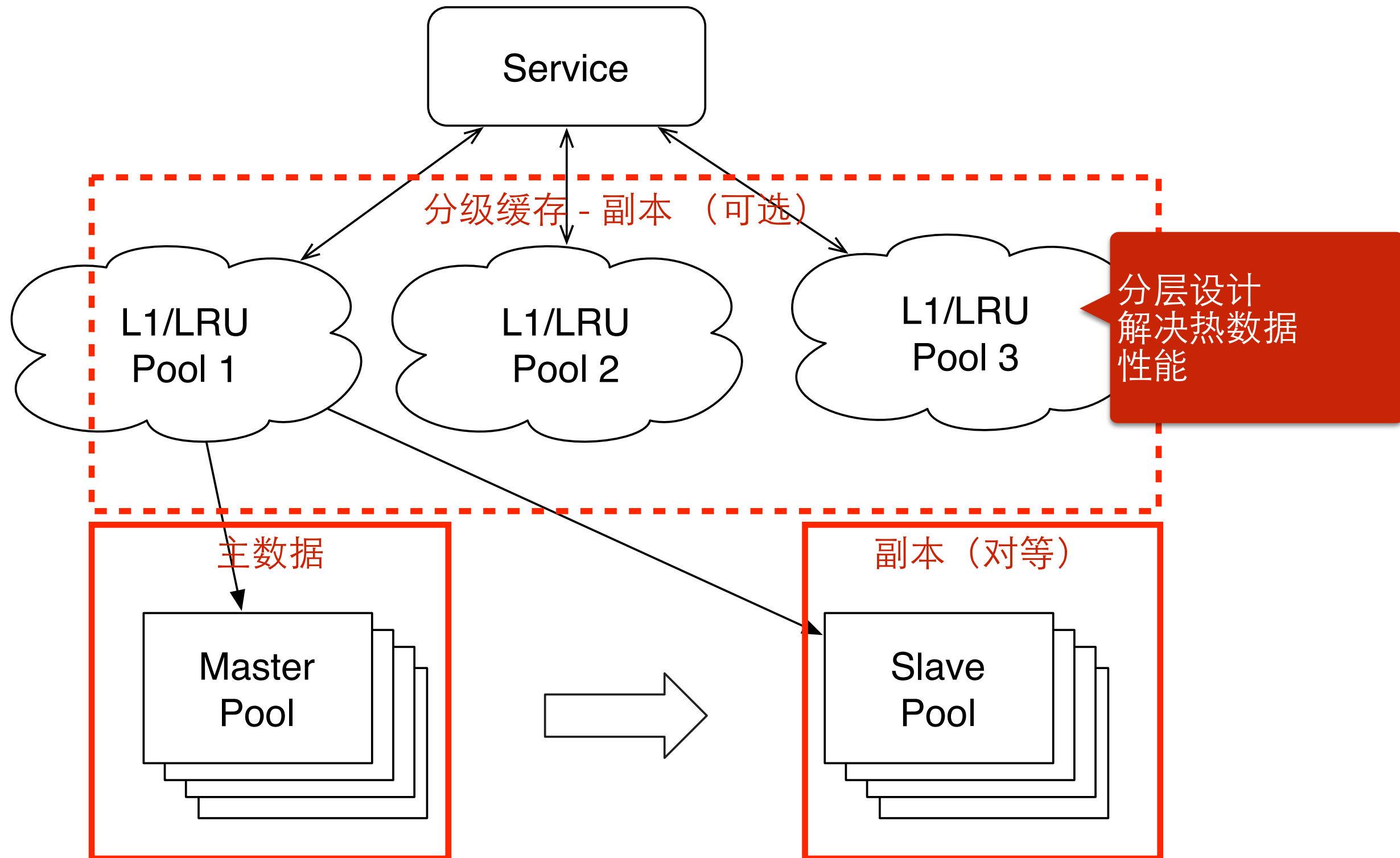
大数据环境的 性能解决之道



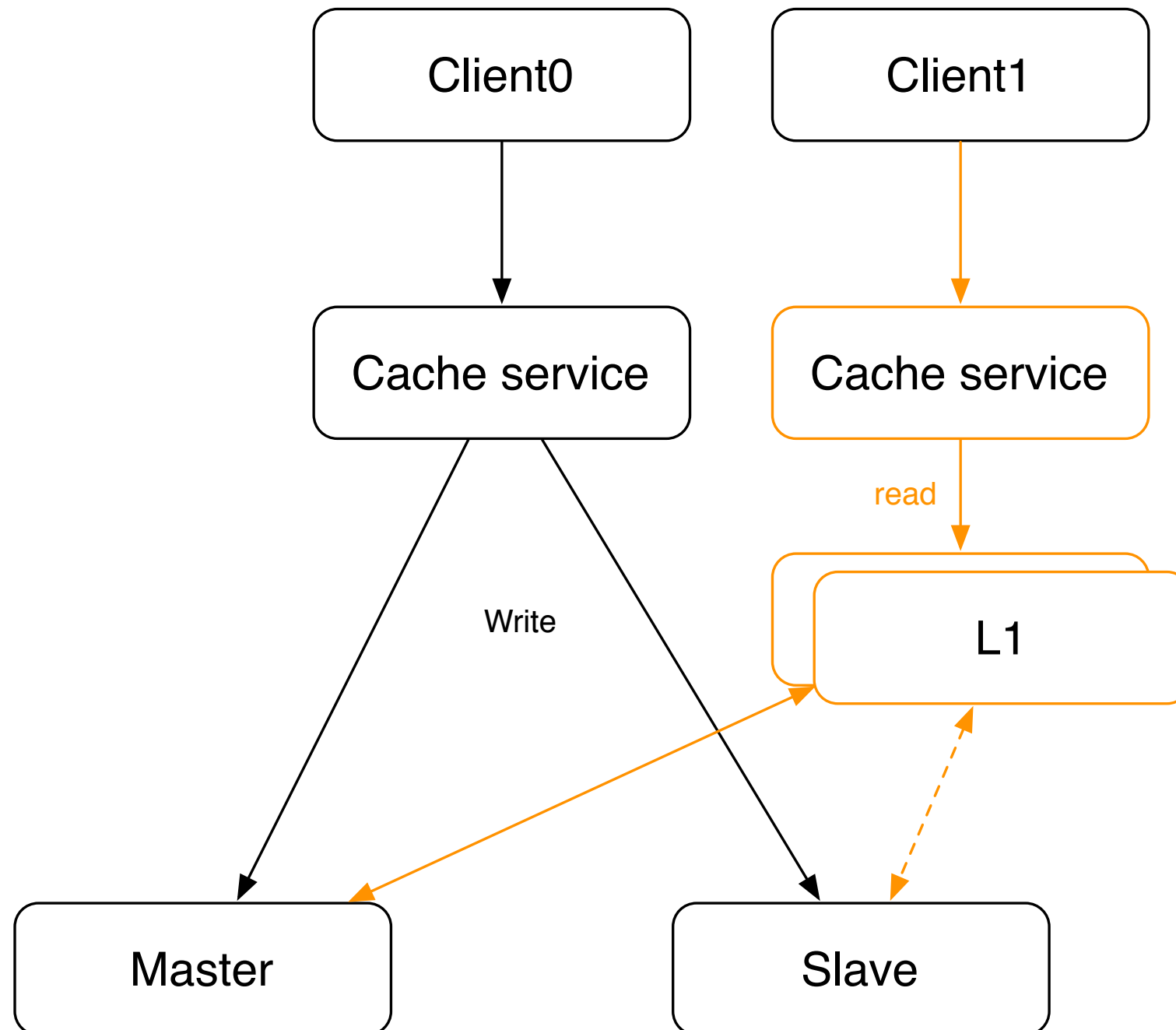
大数据环境的性能解决之道



大数据环境的性能解决之道



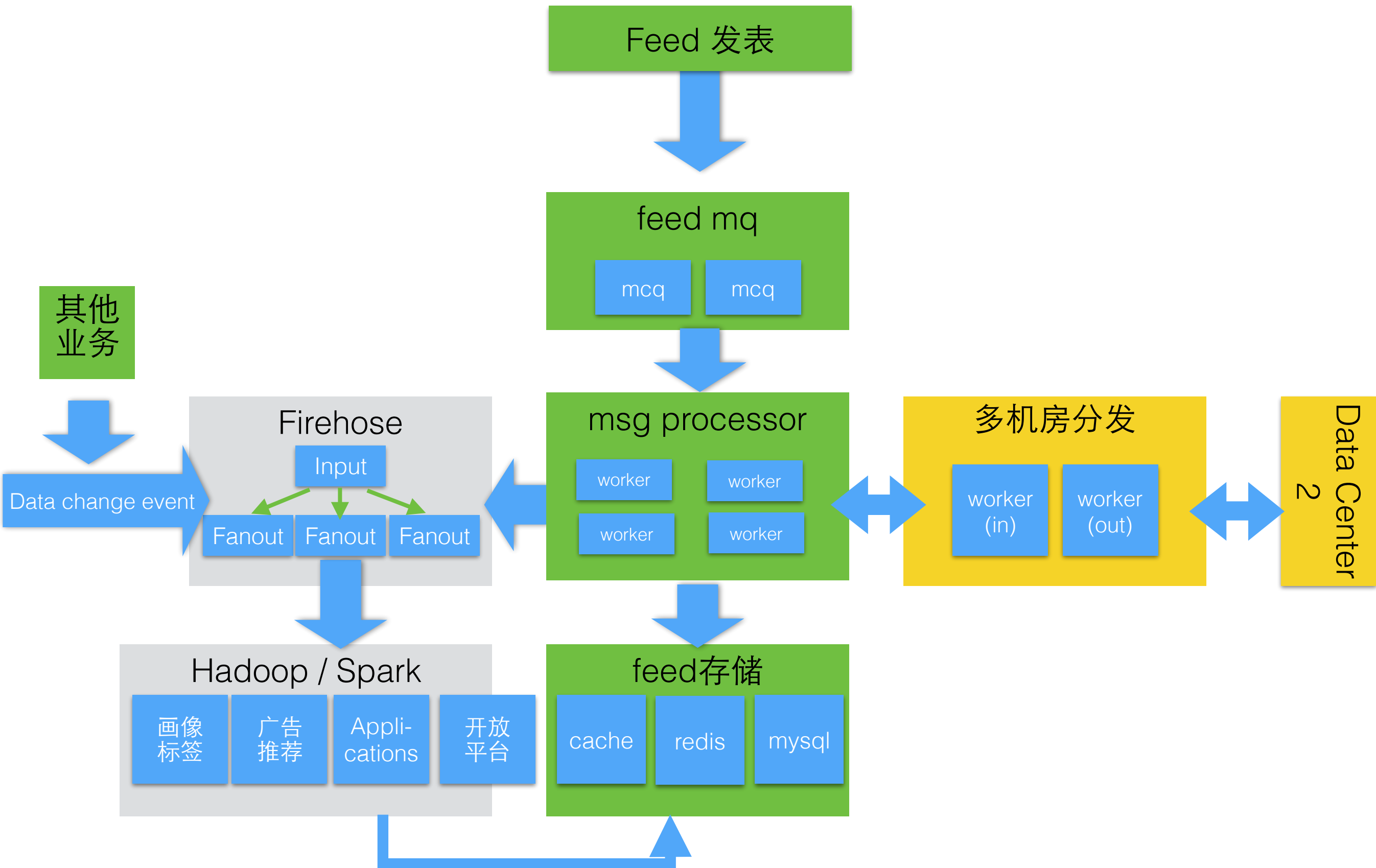
- 如何使用缓存模型来解决可用性及性能问题
- 二级缓存的运作机制详解



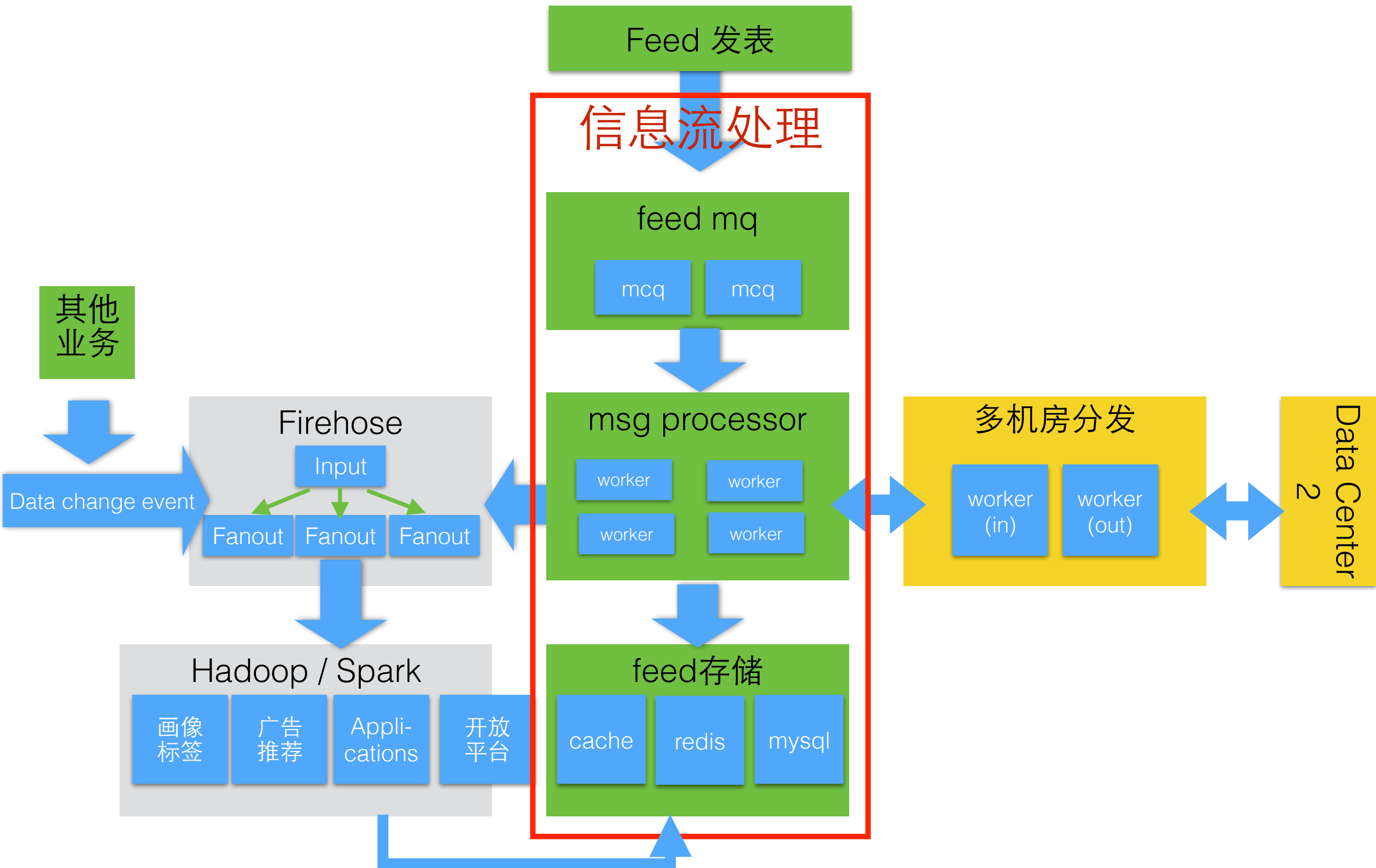
feed性能 - 总结

- ▶ Local cache?
 - ▶ 删除实现复杂
 - ▶ 可以通过短超时方式
- ▶ 极热数据的带宽问题需要提前考虑

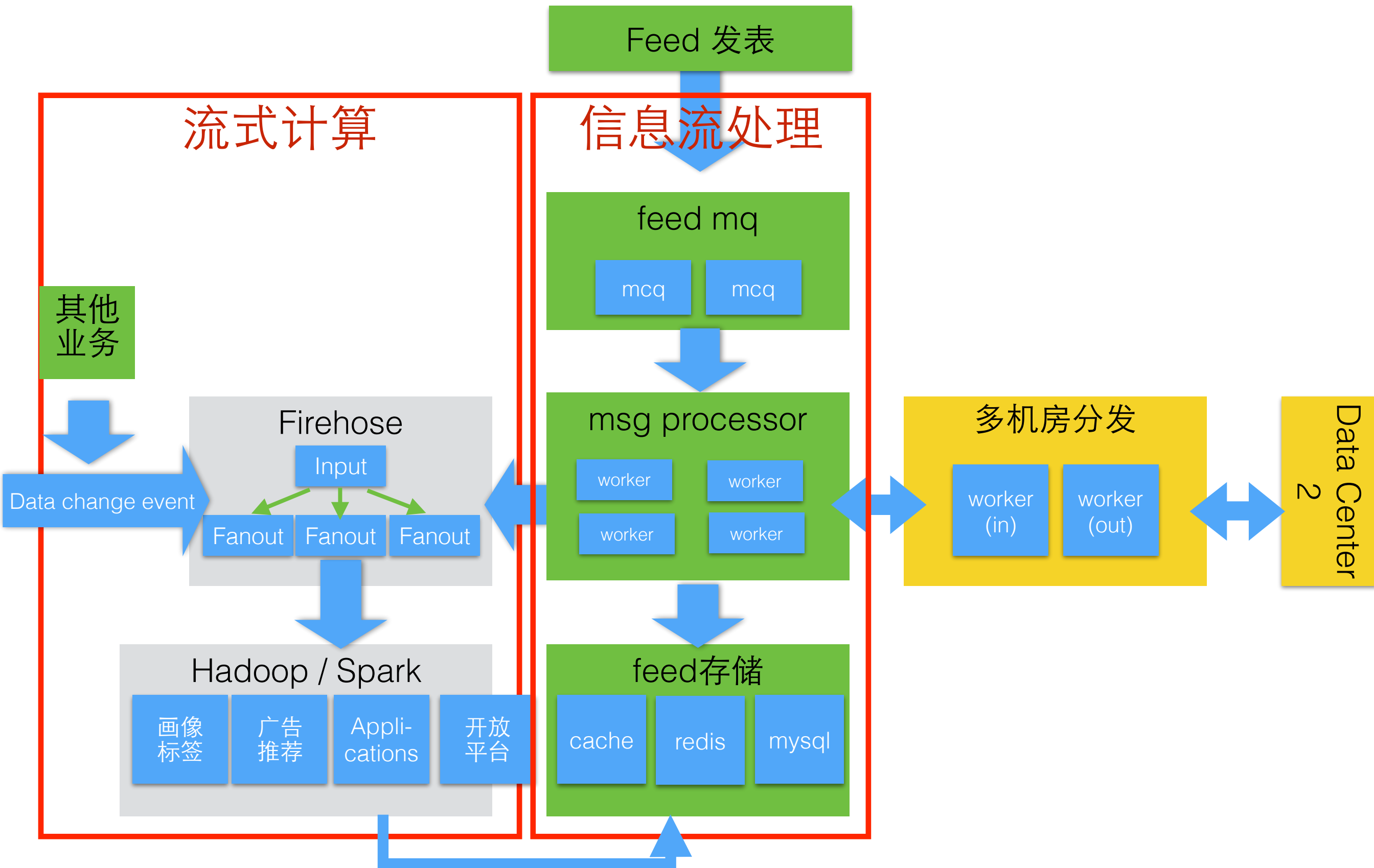
feed消息队列



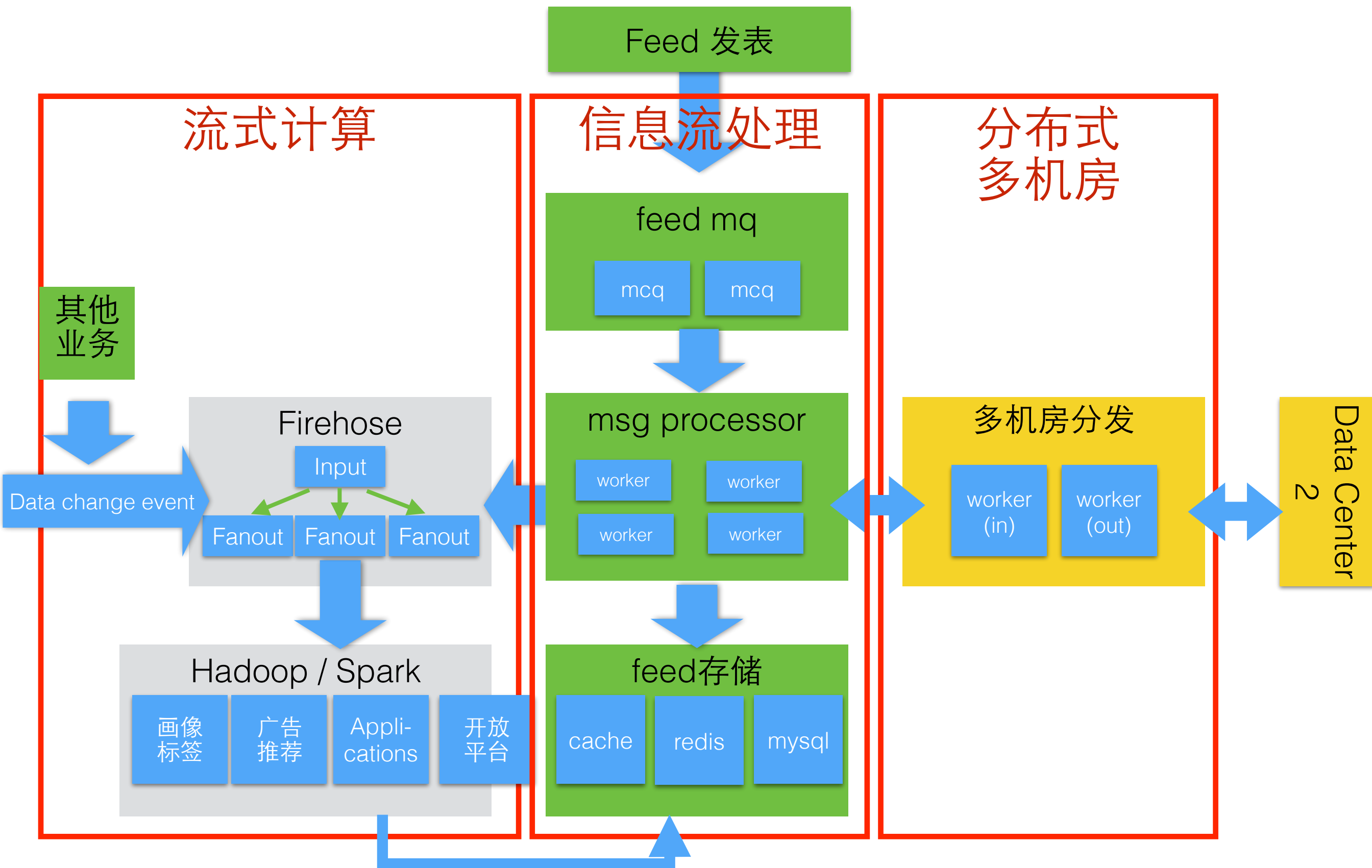
feed消息队列




feed消息队列



feed消息队列




架构特点(I)



性能

- ✓ 实时：处理时间 100ms 以内
- ✓ 可扩展：无状态设计，简单增加节点扩容
- ✓ 可用性：99.99%+，自动failover，无单点

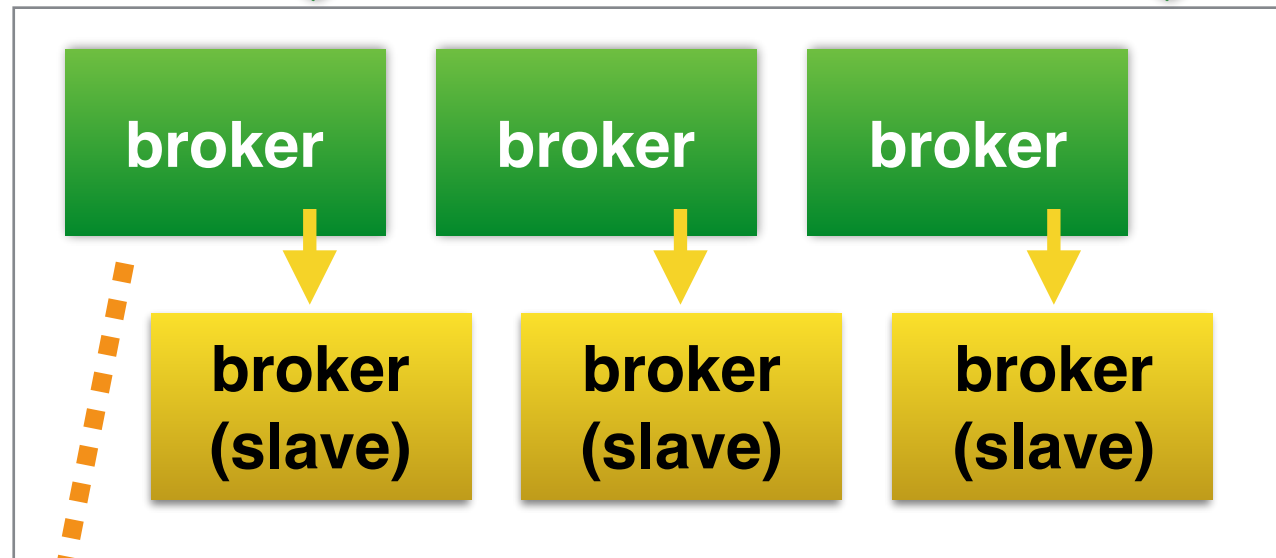
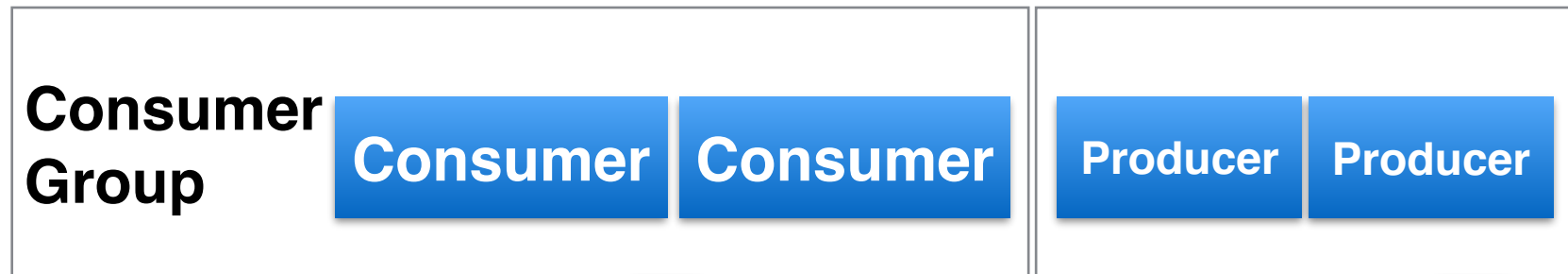
架构特点(2)



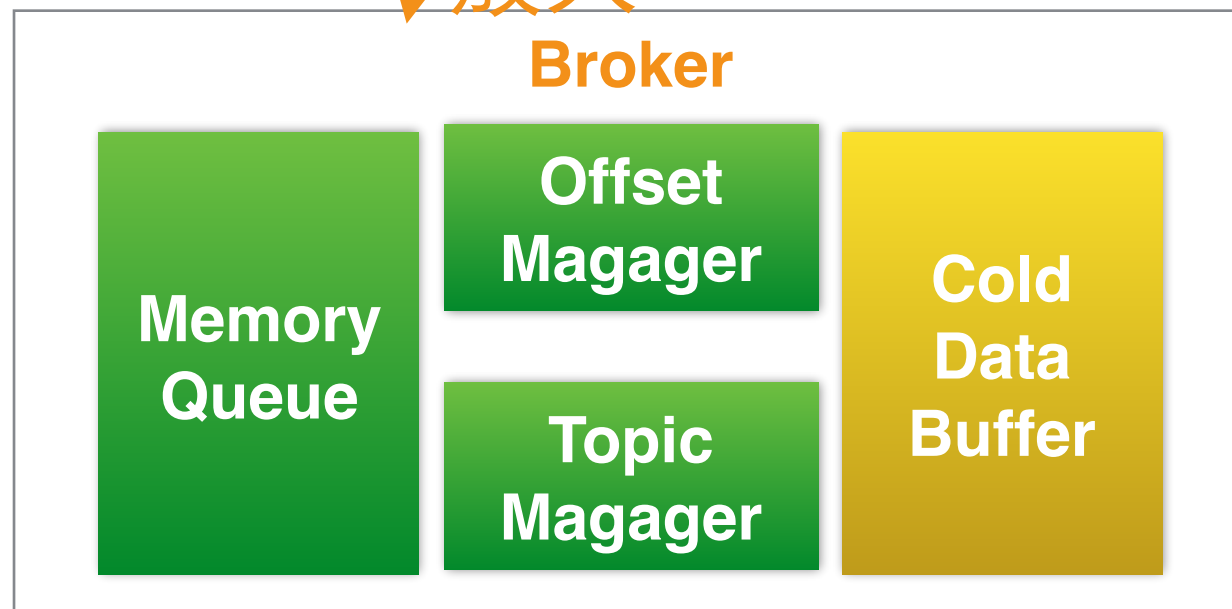
数据流

- ✓ 统一实时推送通道 — mcq & firehose
- ✓ 统一数据流，职责分明，解决三大需求
- ✓ 标准化格式，internal protobuf 格式

firehose - 实时的业务数据流



放大



✓ 一对多(pub-sub)

✓ 实时数据流

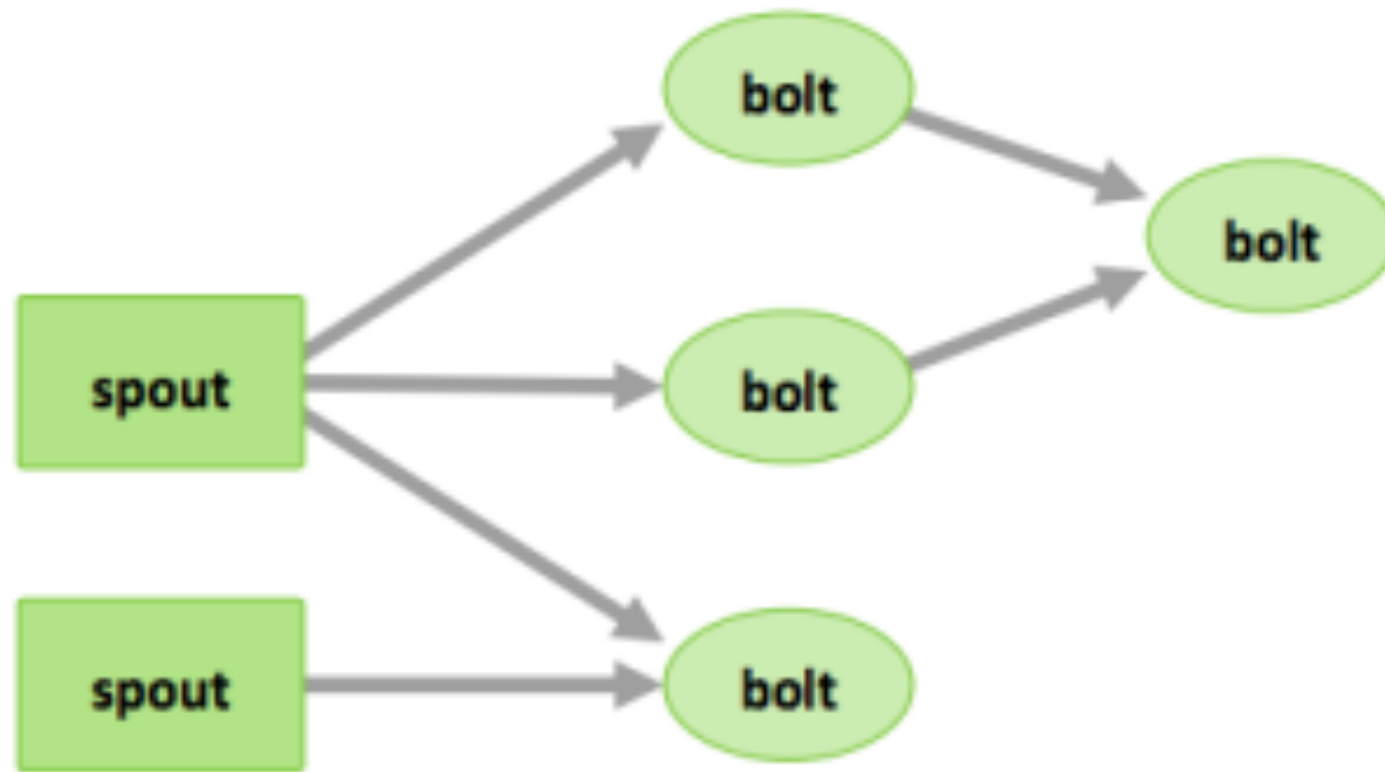
✓ 补推能力

✓ 数据量大，每秒数万条

✓ 可靠性

✓ **Fan-out**

Storm比较



- ▶ msg processor相对于storm没有调度(nimbus)功能;
- ▶ 没有bolt的streaming串联功能, 但可以通过在任务中重写对应业务的mq消息间接实现

Databus比较



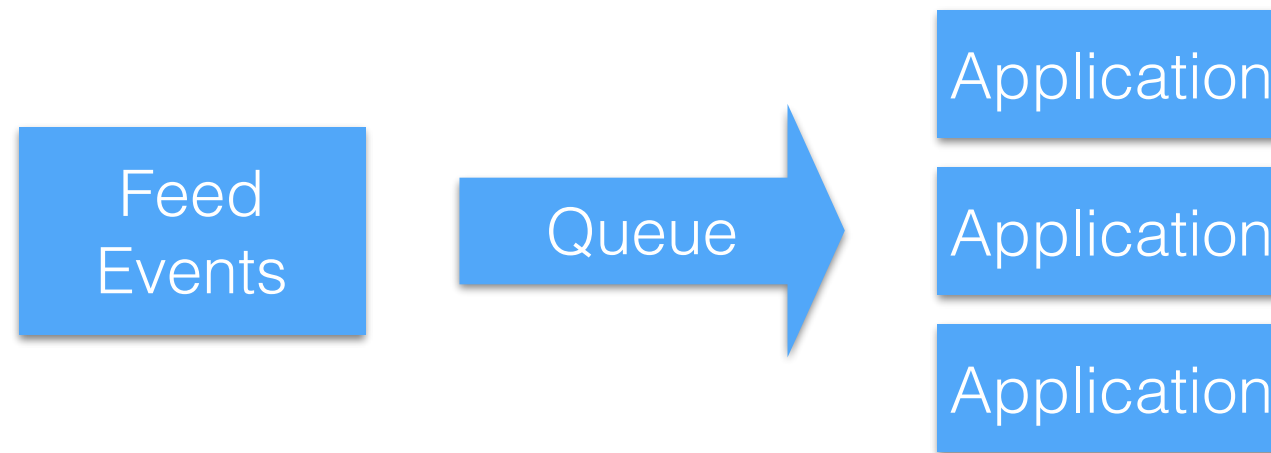
- ▶ Databus基于数据库事件触发消息到总线；
- ▶ 我们使用自行写入消息到firehose的方式

Kafka比较



- ▶ feature基本类似，firehose更偏业务
 - ▶ pub-sub/offset/at least once
- ▶ 都不支持 timeline consistency，不保证时序
 - ▶ 社交产品大多数场景适合

实时数据流 - 总结



- ▶ 罗马不是一天建成的
 - ▶ 自定义队列满天飞的时代的痛苦
 - ▶ 尝试过databus trigger方式
 - ▶ 需要具有抽象共性的意识
- ▶ Lambda architecture

多元化存储

数据类型	特点	存储解决方案	存储产品
微博内容	类型简单 海量访问	关系型数据库 分布式Key - Value	MySQL
微博列表	结构化列表数据 多维度查询	关系型数据库 NoSQL	HBase MySQL
关系	类型简单 高速访问	内存式 key-value key-list结构	MySQL Redis
长微博 图片/短视频	块数据 小文件	分布式文件系统	
计数 (微博数 阅读数...)	结构简单 数据及访问量大	内存紧凑型 NoSQL	Redis

多元化存储

数据类型	特点	存储解决方案	存储产品
------	----	--------	------

微博内容	类型简单 海量访问	关系型数据库 分布式Key - Value	MySQL
------	--------------	--------------------------	-------

微博列表	结构化列表数据 多维度查询	关系型数据库 NoSQL	HBase MySQL
------	------------------	-----------------	----------------

关系	类型简单 高速访问	内存式 key-value key-list结构	MySQL Redis
----	--------------	-----------------------------	----------------

长微博 图片/短视频	块数据 小文件	分布式文件系统	
---------------	------------	---------	--

计数 (微博数 阅读数...)	结构简单 数据及访问量大	内存紧凑型 NoSQL	Redis
-----------------------	-----------------	----------------	-------

列表型数据

数据类型	结构	单个List长度	规模
关注	{“uid”: “follow_uid1”, “follow_uid2”... “follow_uidn”}	1-3000	千亿级
粉丝	{“uid”: “fan_uid1”, “fan_uid2”... “fan_uidn”}	1-8000万	千亿级
发表微博列表	{“uid”: “feed_id1”, “feed_id2”... “feed_idn”}	1-100万+	千亿级
转发微博列表	{“weibo_id”: “repost_id1”, “repost_id2”... “repost_idn”}	1-500万+	千亿级
评论列表	{“weibo_id”: “cmt_id1”, “cmt_id2”... “cmt_idn”}	1-500万+	千亿级

列表型数据

类型多

数据类型	结构	单个List长度	规模
关注	{“uid”: “follow_uid1”, “follow_uid2”... “follow_uidn”}	1-3000	千亿级
粉丝	{“uid”: “fan_uid1”, “fan_uid2”... “fan_uidn”}	1-8000万	千亿级
发表微博列表	{“uid”: “feed_id1”, “feed_id2”... “feed_idn”}	1-100万+	千亿级
转发微博列表	{“weibo_id”: “repost_id1”, “repost_id2”... “repost_idn”}	1-500万+	千亿级
评论列表	{“weibo_id”: “cmt_id1”, “cmt_id2”... “cmt_idn”}	1-500万+	千亿级

列表型数据

类型多

数据类型

结构

变长/超长

单个List
长度

规模

关注

```
{“uid”: “follow_uid1”,  
“follow_uid2”... “follow_uidn”}
```

1-3000

千亿级

粉丝

```
{“uid”: “fan_uid1”, “fan_uid2”...  
“fan_uidn”}
```

1-8000万

千亿级

发表微博列表

```
{“uid”: “feed_id1”, “feed_id2”...  
“feed_idn”}
```

1-100万+

千亿级

转发微博列表

```
{“weibo_id”: “repost_id1”,  
“repost_id2”... “repost_idn”}
```

1-500万+

千亿级

评论列表

```
{“weibo_id”: “cmt_id1”,  
“cmt_id2”... “cmt_idn”}
```

1-500万+

千亿级

列表型数据

类型多

数据类型

结构

变长/超长

单个List
长度

大数据

规模

关注

```
{"uid": "follow_uid1",  
  "follow_uid2"... "follow_uidn"}
```

1-3000

千亿级

粉丝

```
{"uid": "fan_uid1", "fan_uid2"...  
  "fan_uidn"}
```

1-8000万

千亿级

发表微博列表

```
{"uid": "feed_id1", "feed_id2"...  
  "feed_idn"}
```

1-100万+

千亿级

转发微博列表

```
{"weibo_id": "repost_id1",  
  "repost_id2"... "repost_idn"}
```

1-500万+

千亿级

评论列表

```
{"weibo_id": "cmt_id1",  
  "cmt_id2"... "cmt_idn"}
```

1-500万+

千亿级

最近

2014

● 12月

11月

10月

9月

8月

7月

6月

5月

4月

3月

2月

1月

2013

2012

2011

● 第一条微博

列表访问效率

最近

2014

12月

11月

10月

9月

8月

7月

6月

5月

4月

3月

2月

1月

2013

2012

2011

第一条微博

列表访问效率

Newer Posts	Home	Older Posts
-------------	------	-------------

最近

2014

12月

11月

10月

9月

8月

7月

6月

5月

4月

3月

2月

1月

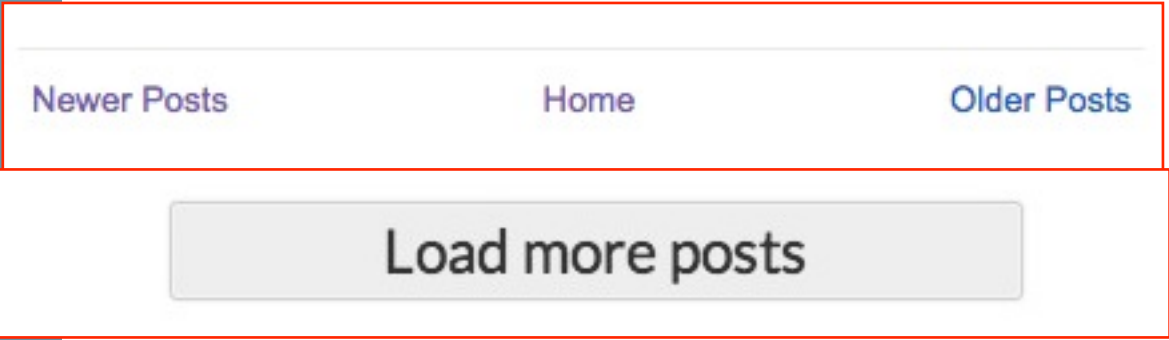
2013

2012

2011

第一条微博

列表访问效率



最近

2014

12月

11月

10月

9月

8月

7月

6月

5月

4月

3月

2月

1月

2013

2012

2011

第一条微博

列表访问效率

Newer Posts

Home

Older Posts

Page 1 of 40 1 2 3 4 5 » 10 20 30 ... Last »

Load more posts

列表访问效率

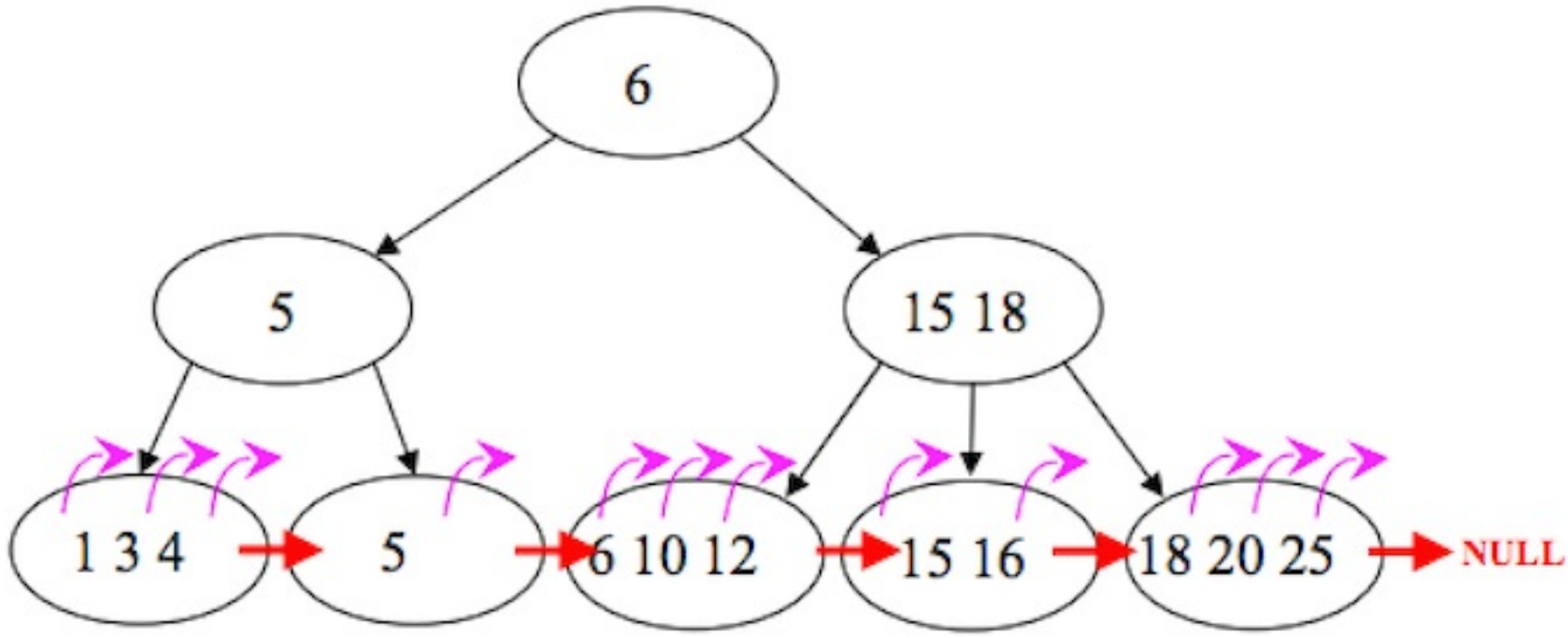
[Newer Posts](#)[Home](#)[Older Posts](#)

Load more posts

Page 1 of 40

12345»102030...Last »

B⁺-tree of order 4



列表访问效率

Newer Posts

Home

Older Posts

Load more posts

Page 1 of 40

1

2

3

4

5

»

10

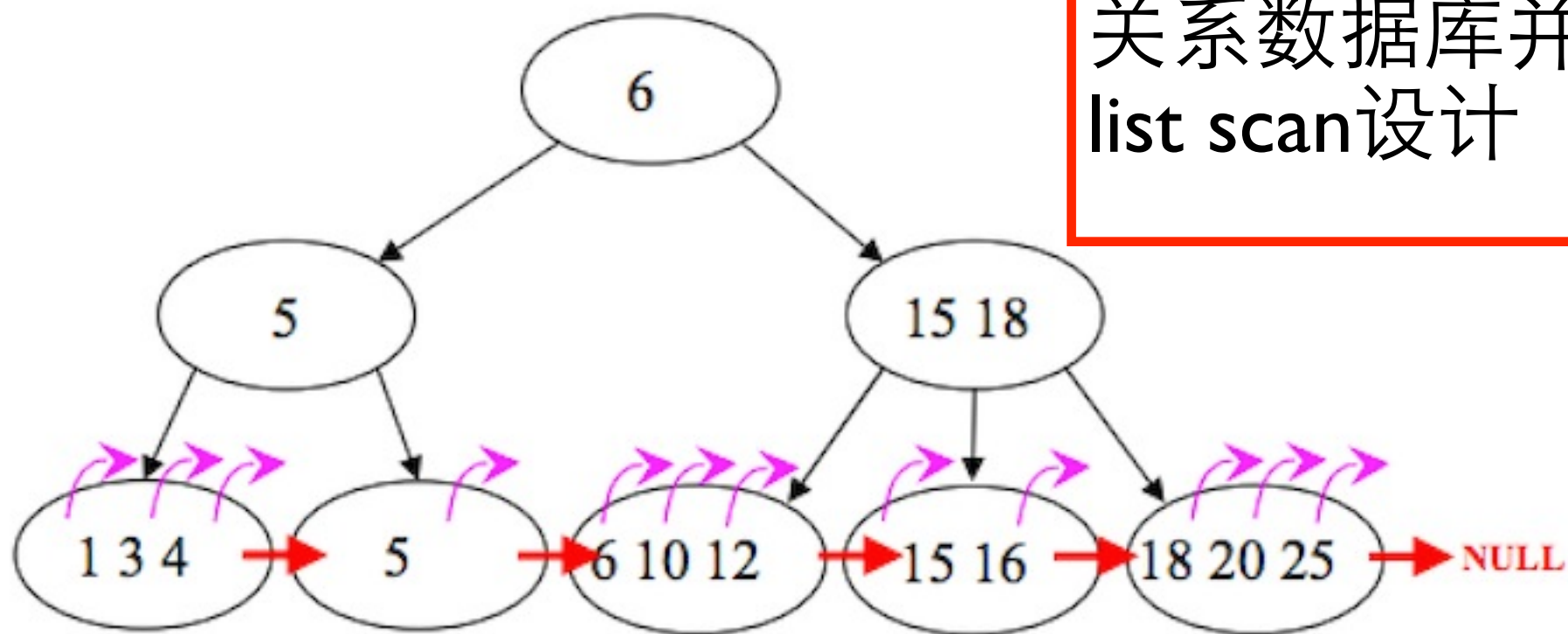
20

30

...

Last »

B⁺-tree of order 4



关系数据库并非为
list scan设计

最近

2014

12月

11月

10月

9月

8月

7月

6月

5月

4月

3月

2月

1月

2013

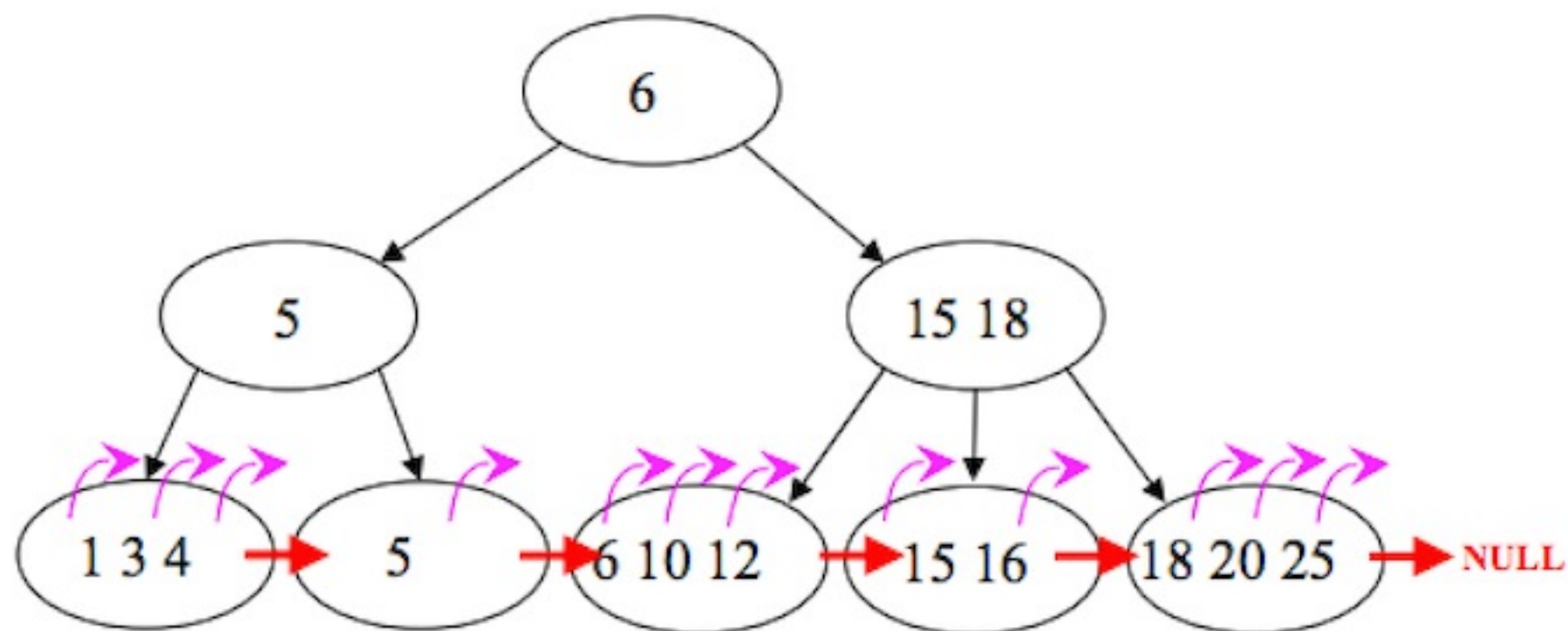
2012

2011

第一条微博

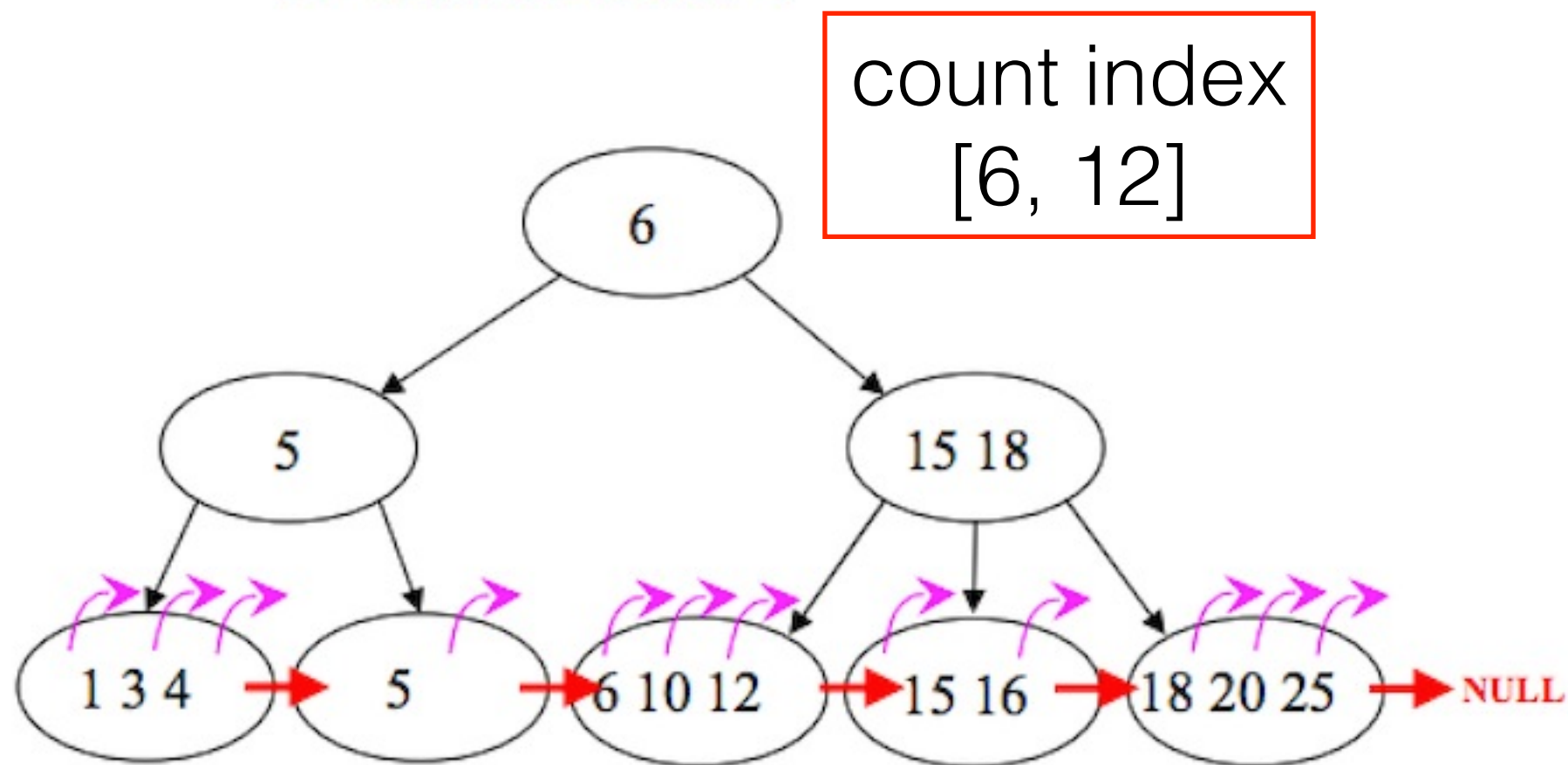
通用的二级索引

B⁺-tree of order 4



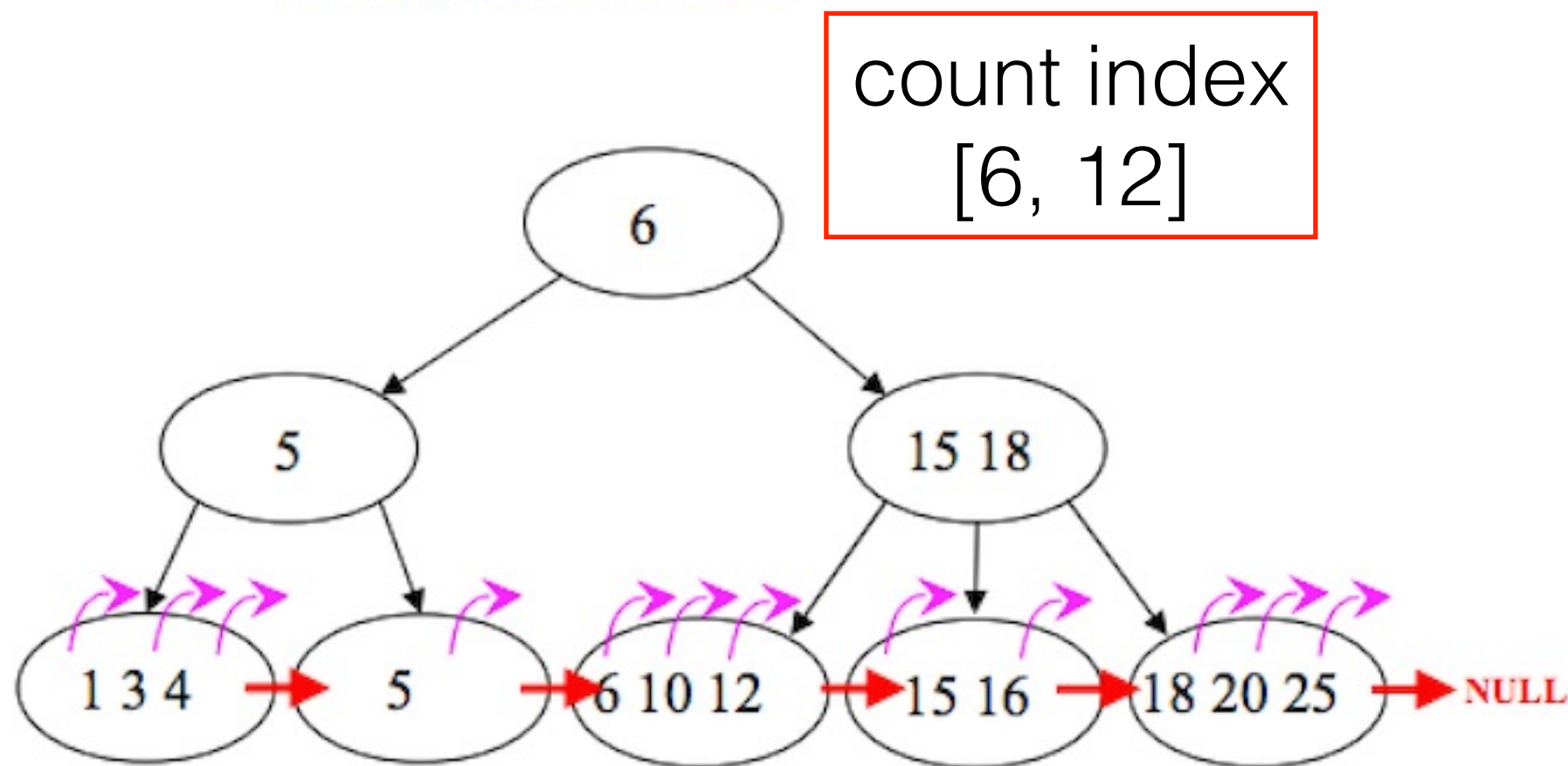
通用的二级索引

B⁺-tree of order 4



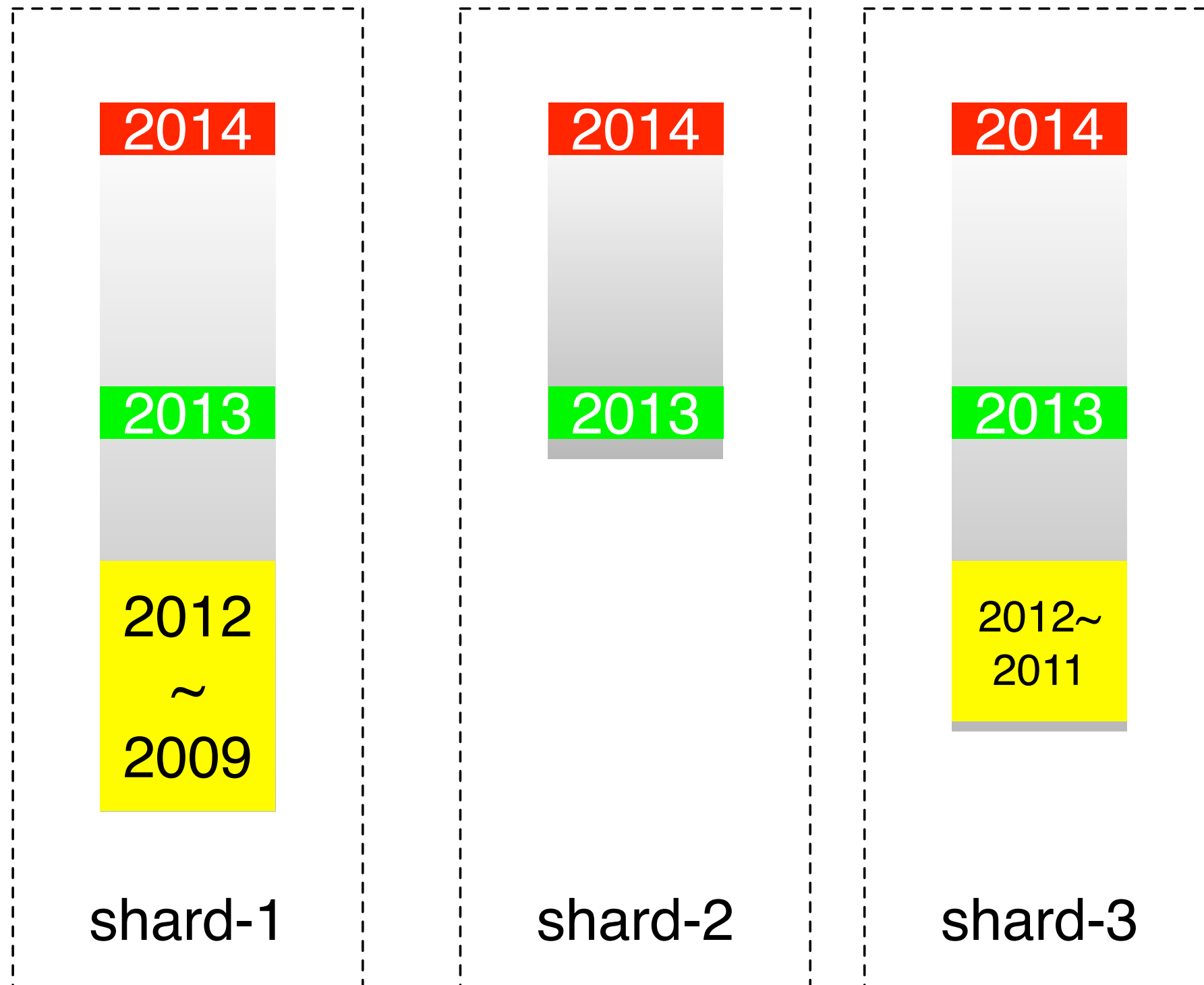
通用的二级索引

B⁺-tree of order 4

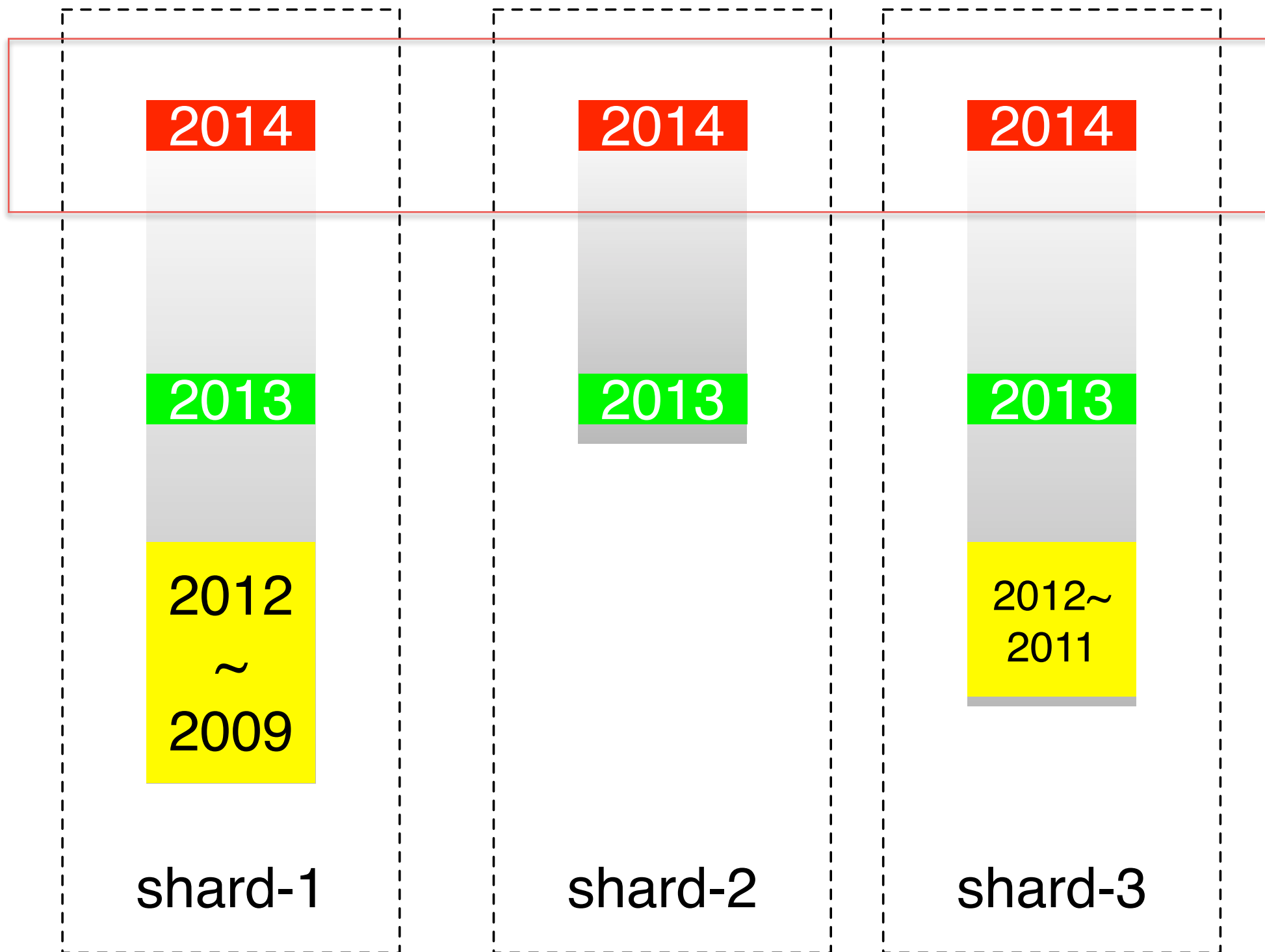


Offset index
[15, 8]

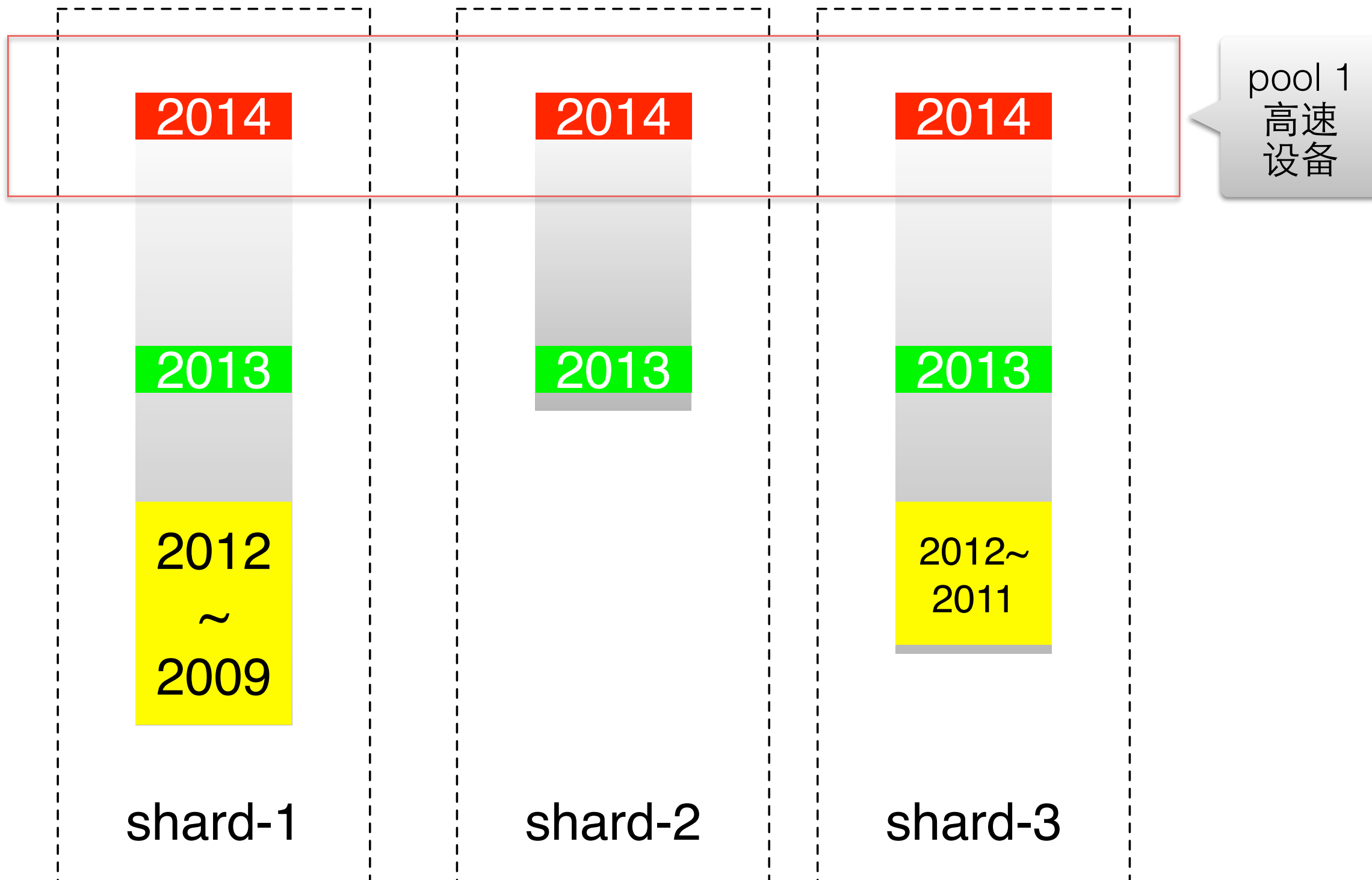
列表性能及成本



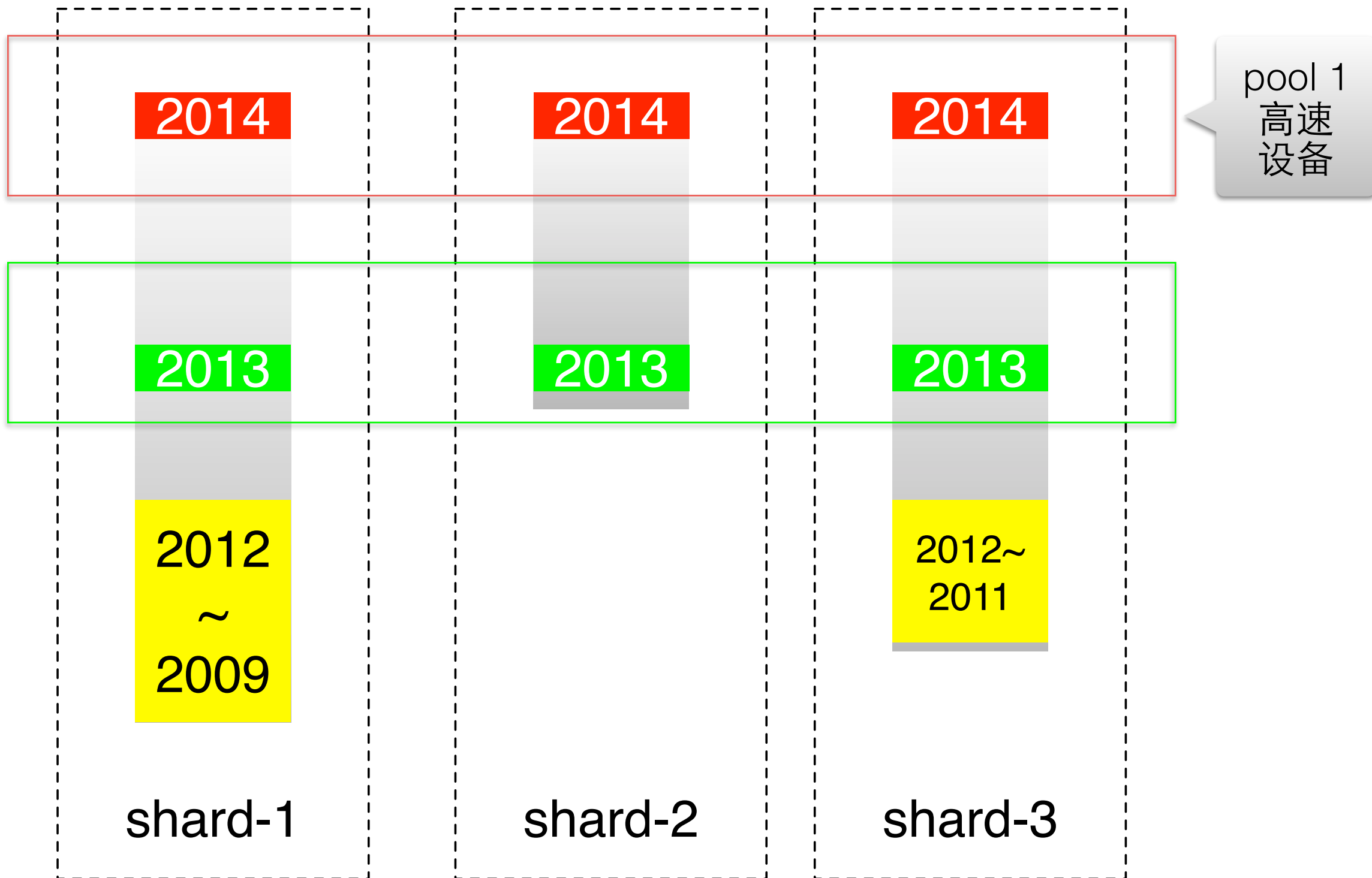
列表性能及成本



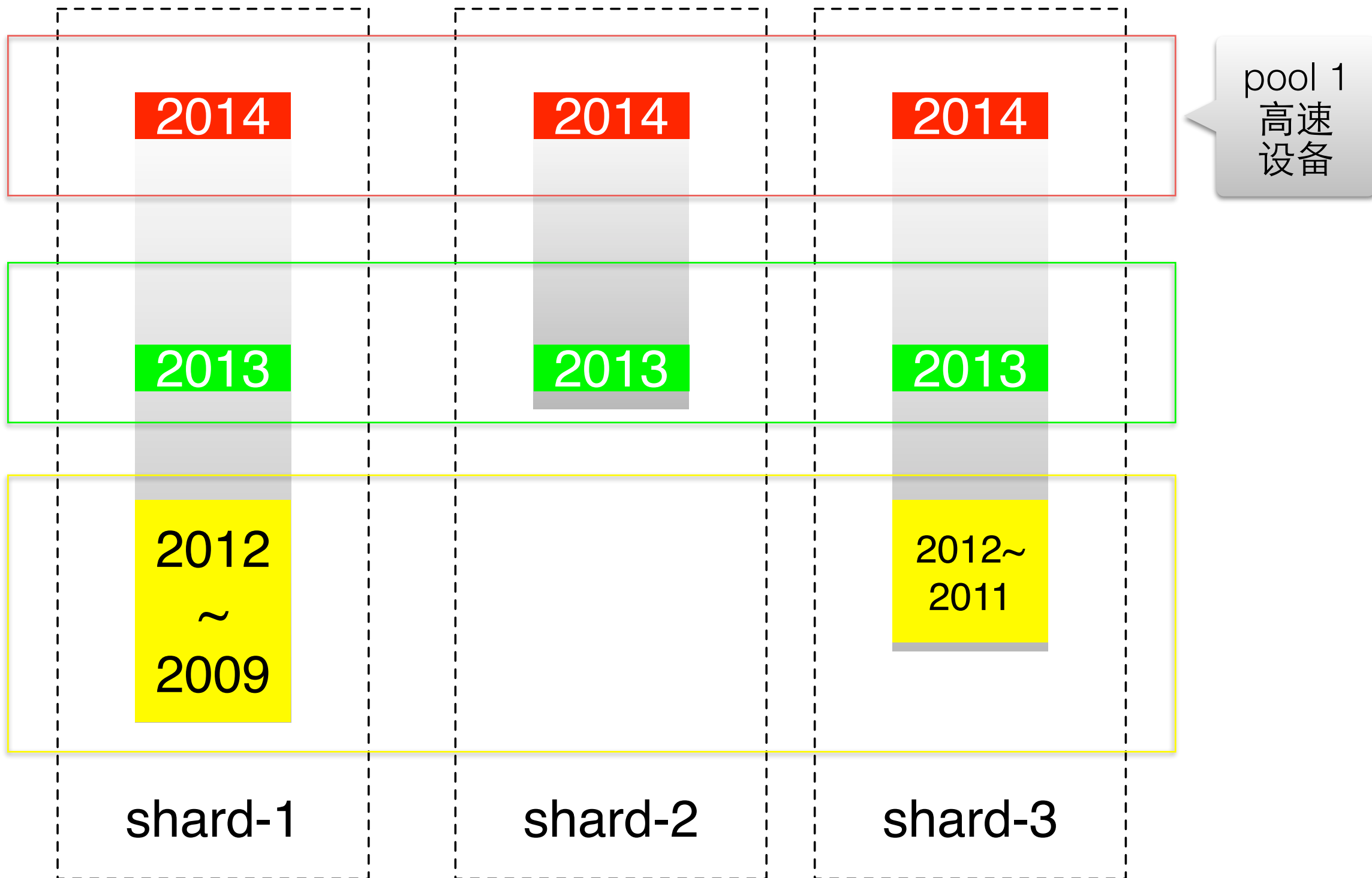
列表性能及成本



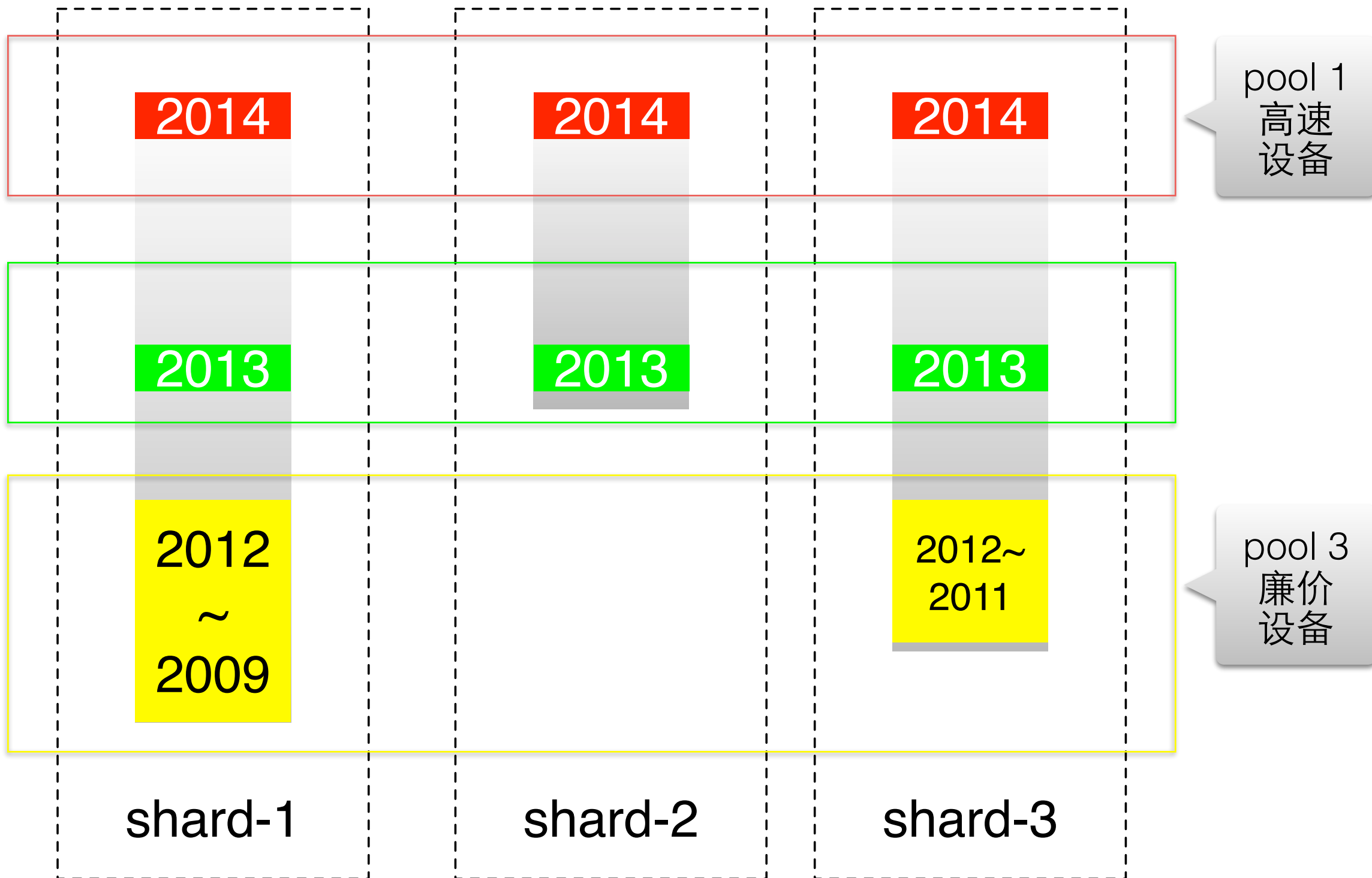
列表性能及成本



列表性能及成本



列表性能及成本



列表存储服务

接口

saveList(id, offset, size)

loadList(id, offset, size)

存储
策略层

一致性

超长列表

Sharding

SLA(QoS)

Metrics

Trace

加速层
二级索引

offset index

count index

存储
引擎

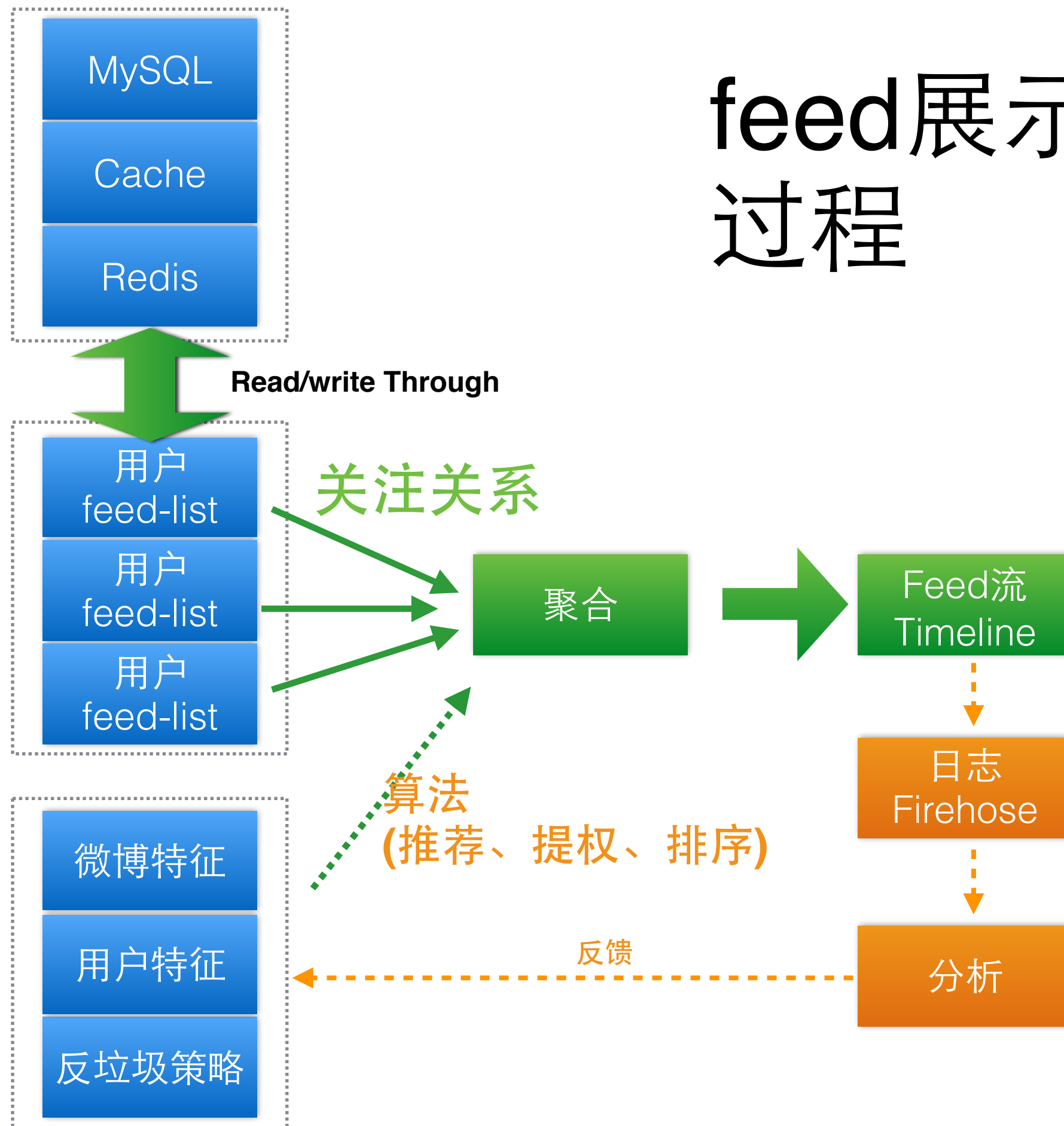
MySQL

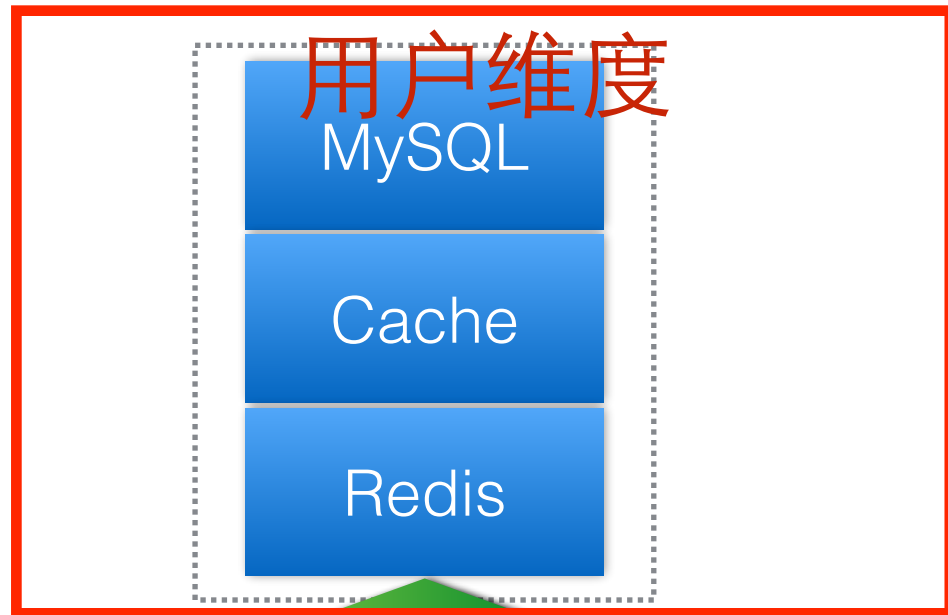
HBase

feed存储 - 总结

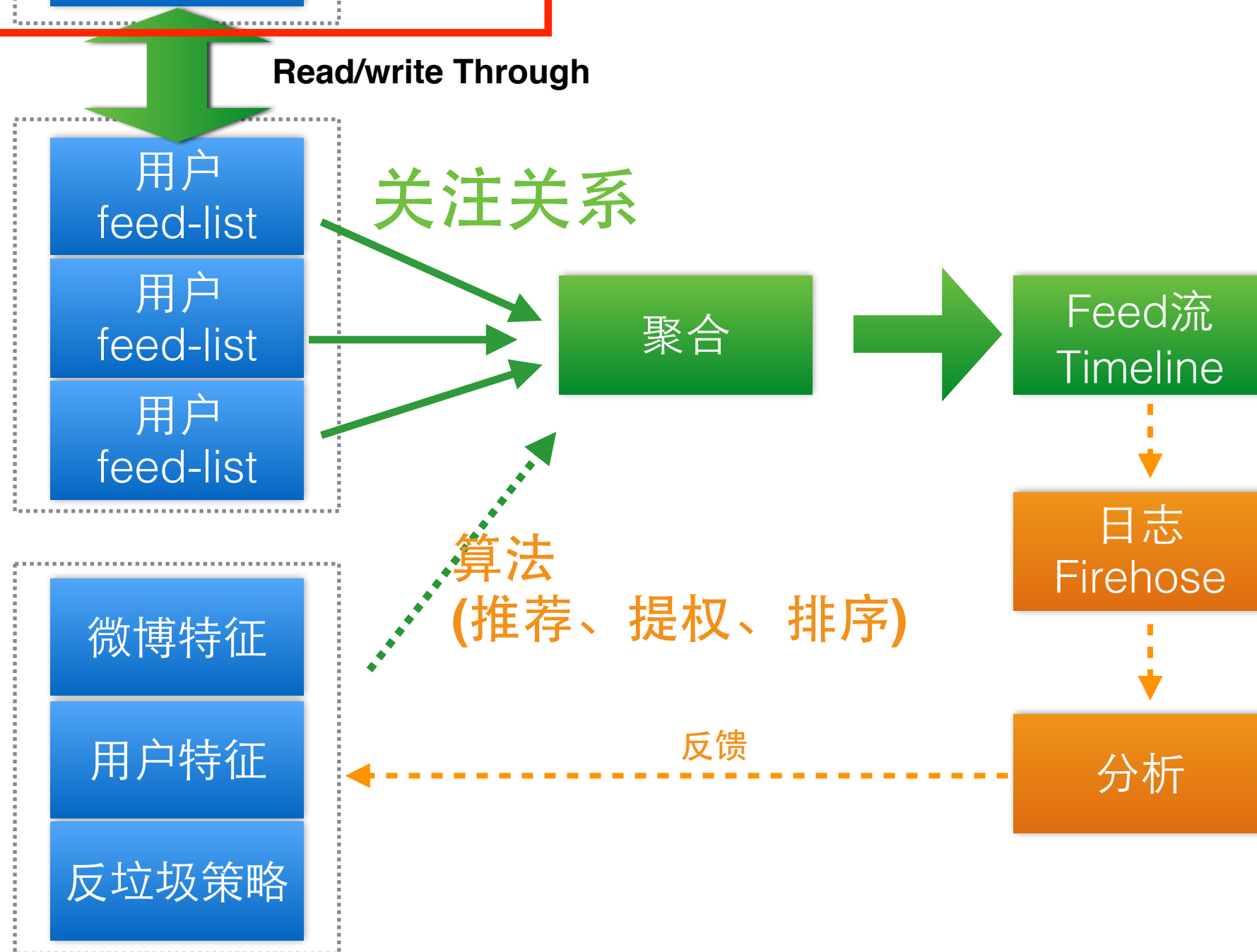
- ▶ 从各自建设到可复用的方向发展
 - ▶ 曾尝试mysql-proxy方向，但业务方需求不强烈
- ▶ 类似超长列表的服务得到了较好支持
 - ▶ 抽象共性问题并解决，而不是增加熵

feed展示过程

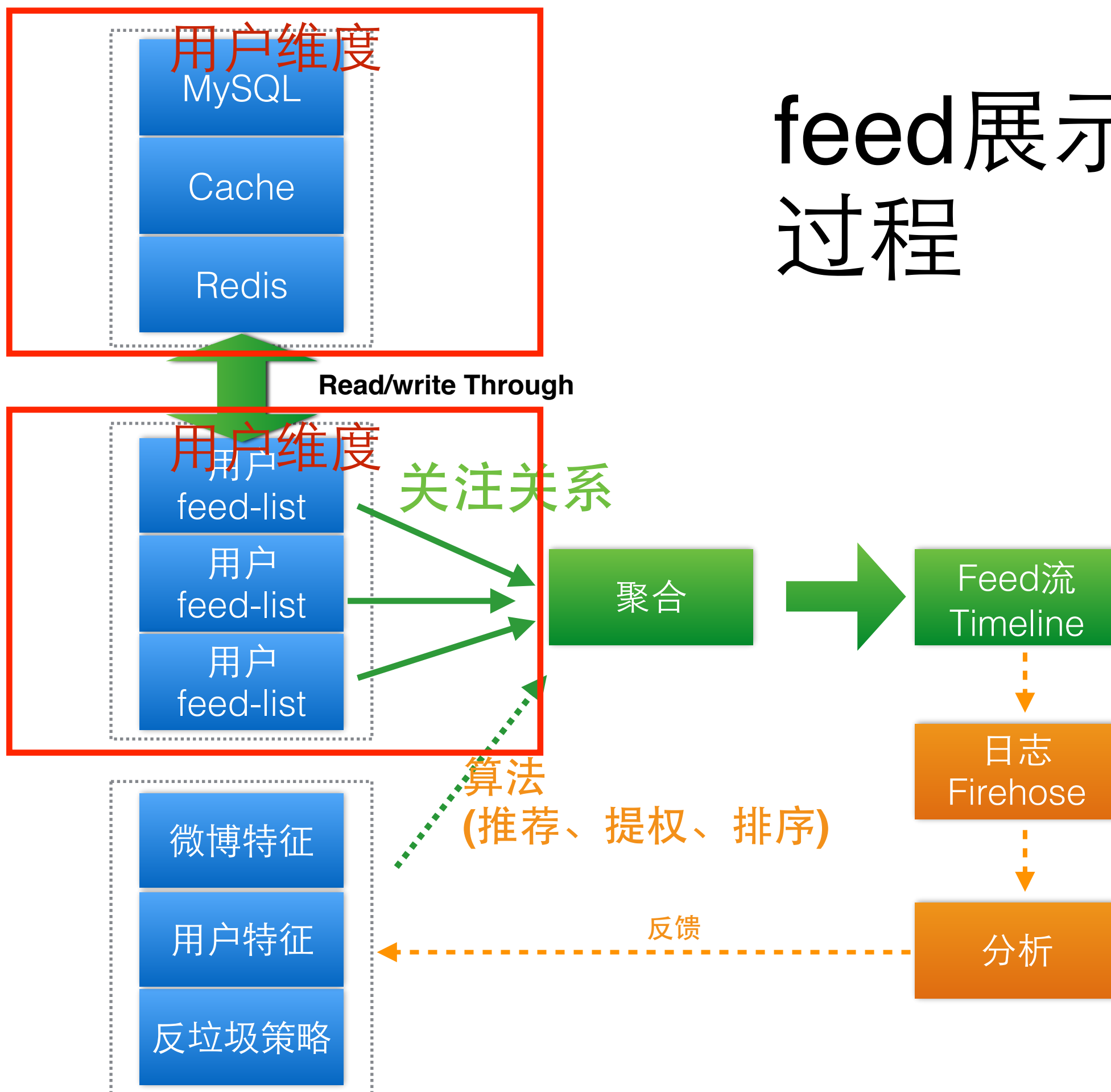




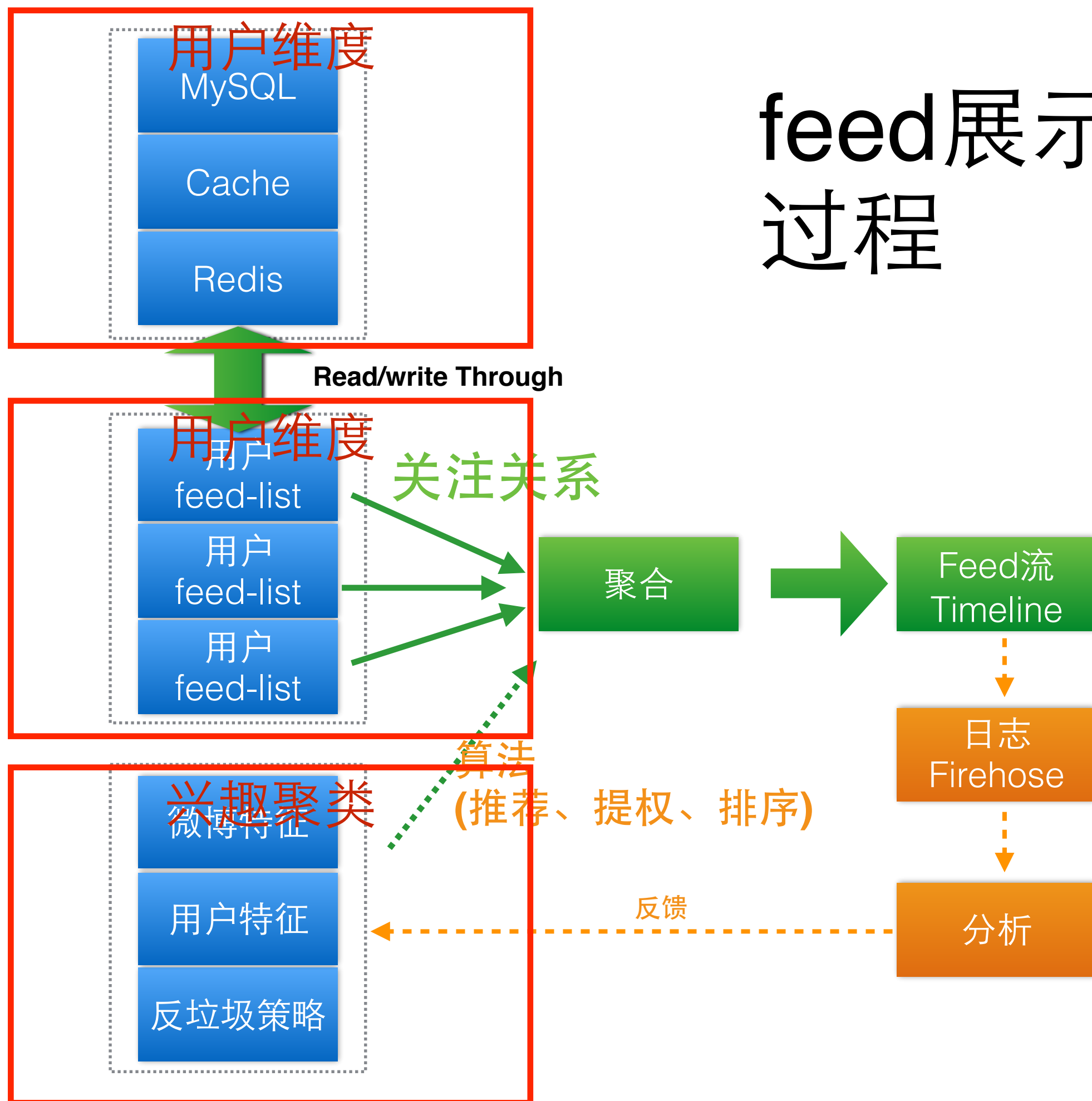
feed展示过程



feed展示过程



feed展示过程



feed展示- 总结

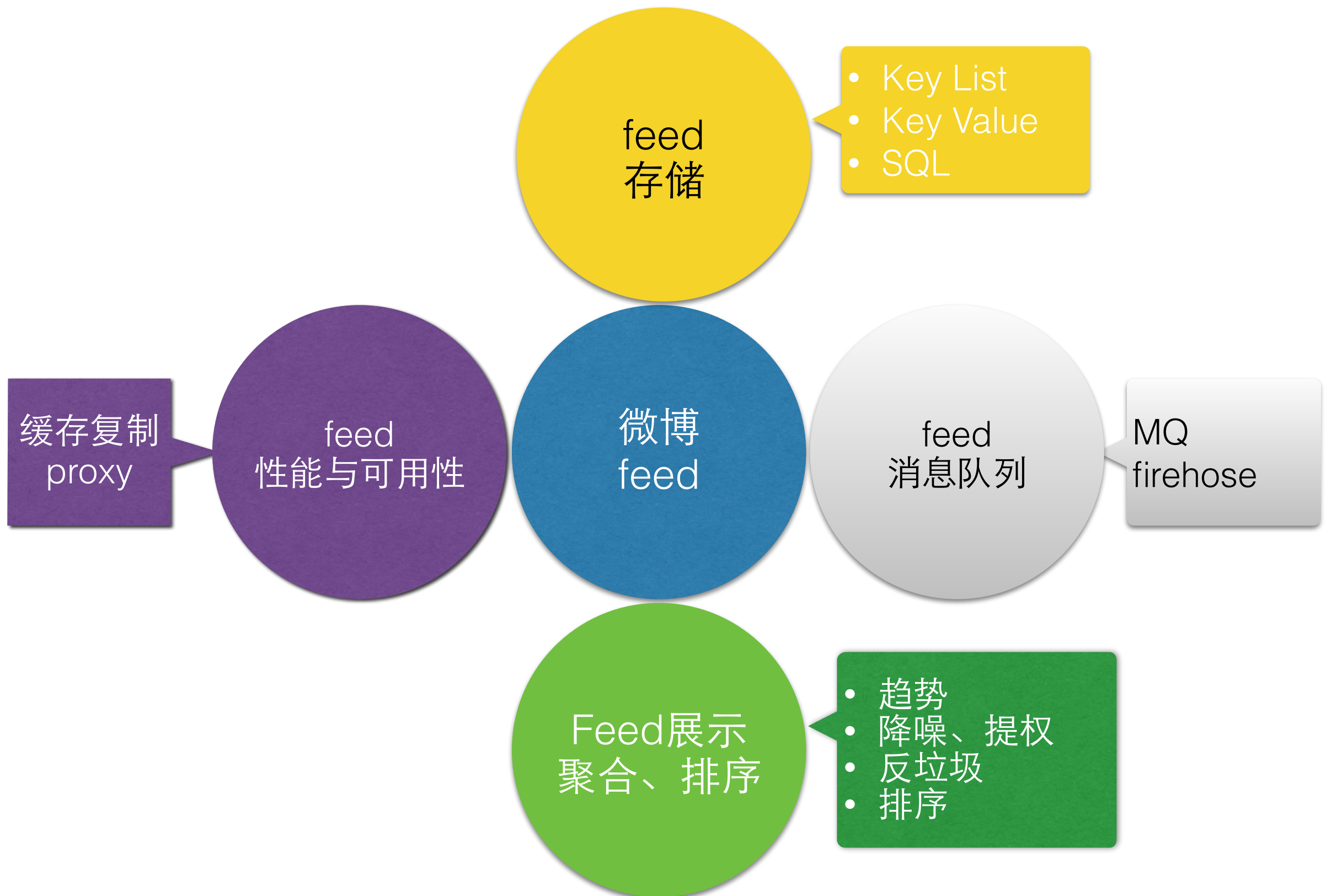
做对的

- ✓ 成熟的feed推拉聚合模型
- ✓ 成熟的用户数据组织方式

不足的

- ⊙ 基于用户维度组织内容高效满足兴趣阅读的难度
- ⊙ 信息识别及低质内容鉴定的技术挑战
- ⊙ 反垃圾算法的难度

总结与展望



Q&A



TimYang

扫一扫二维码图案，关注我吧



TimYang
微信公众号

<http://timyang.net>