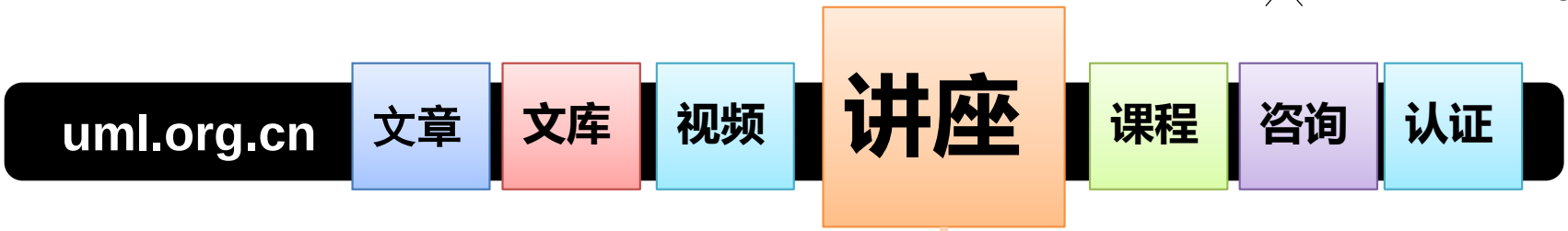


火龙果讲堂：

- **一线专家**
- **案例回顾**
- **经验分享**

快的人的“出埃及记”

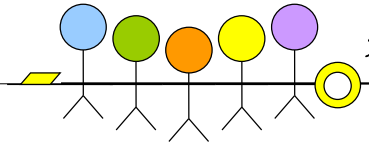
——快的打车成长历程分享

随时听讲座
每天看新文

追随技术信仰

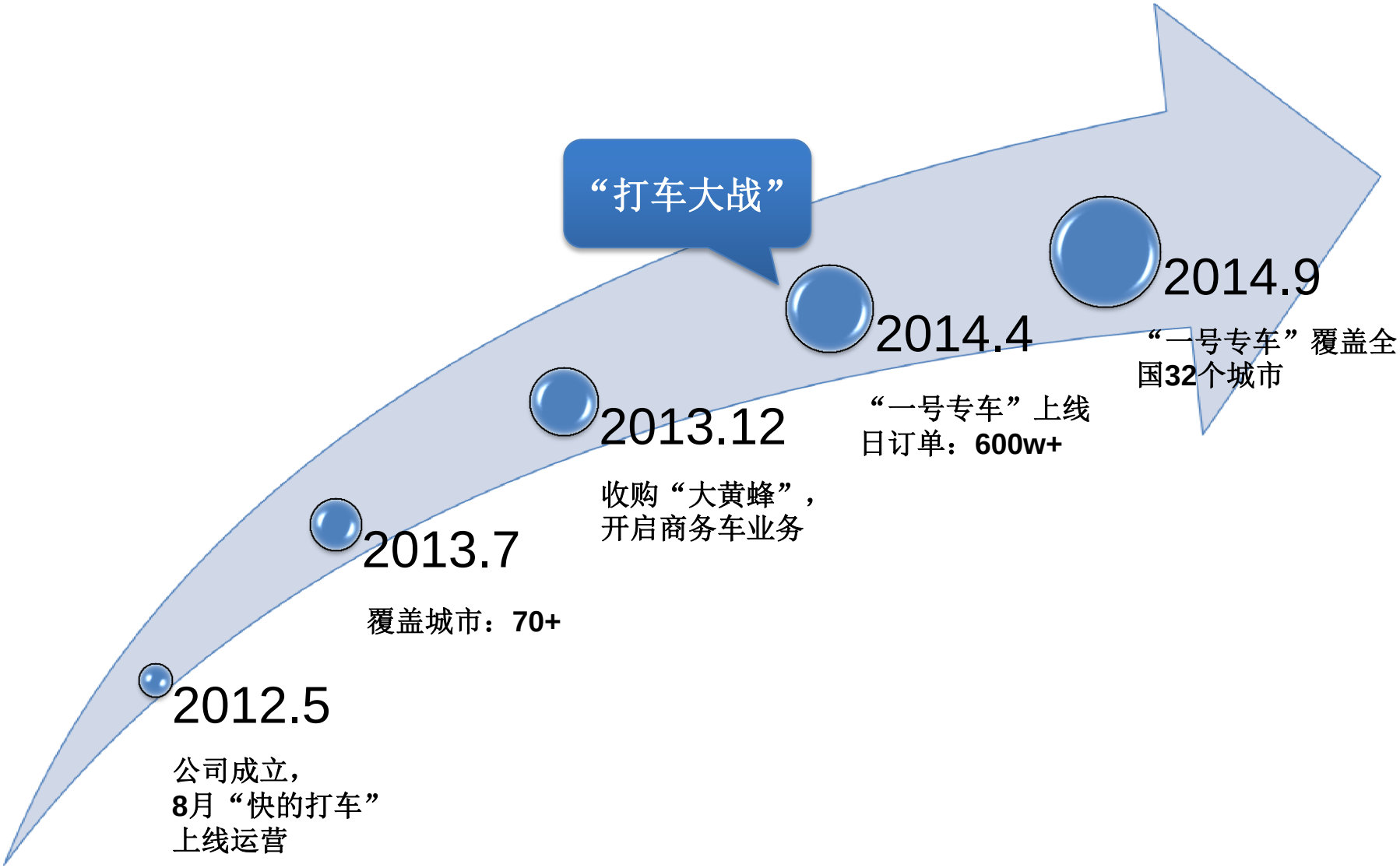
欧阳康 架构师

大纲



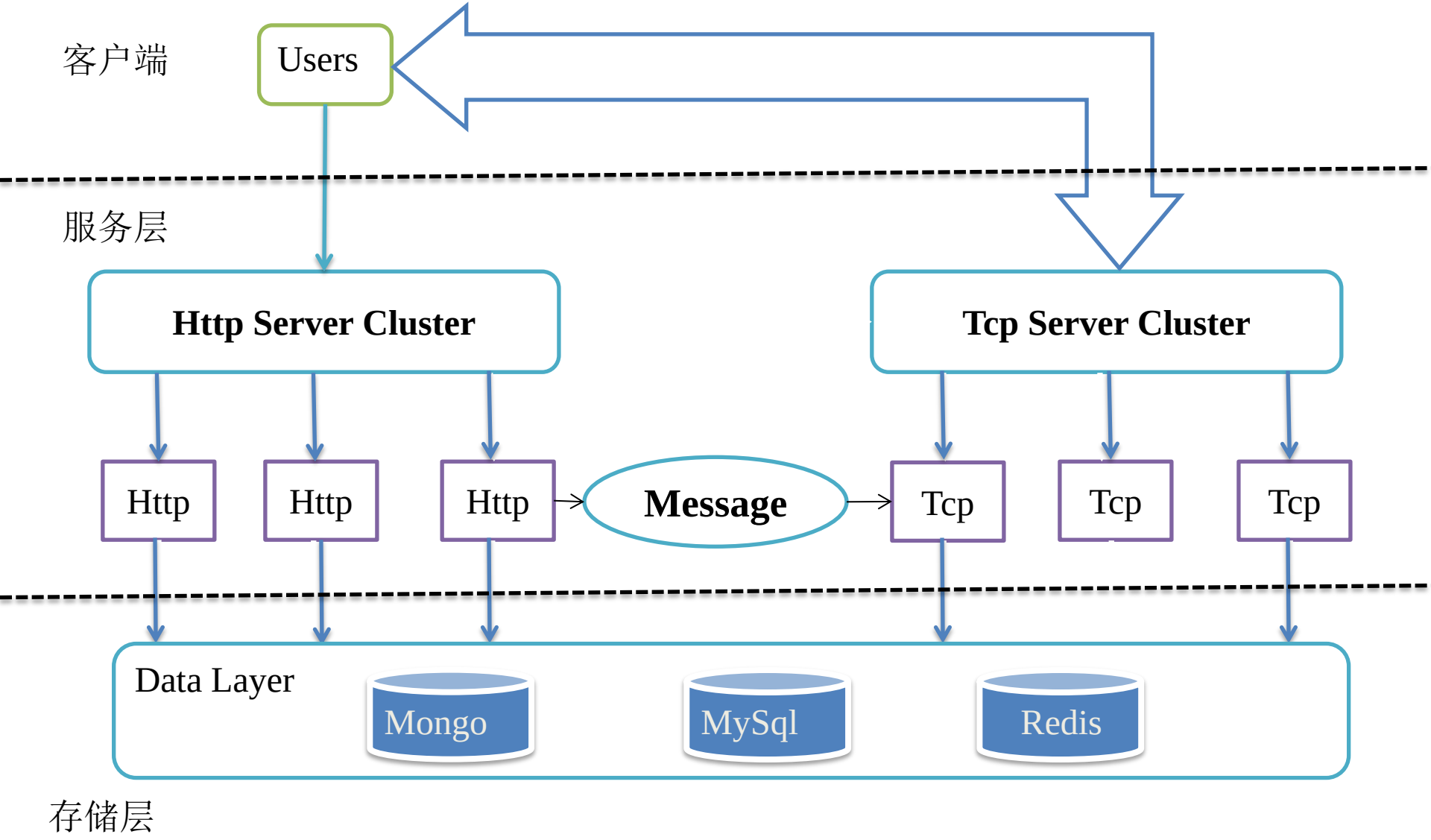
- 业务发展概况
- 业务发展过程中遇到的问题及解决方案
- 未来架构规划

业务发展概况

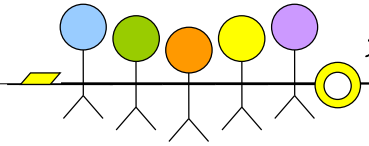

“打车大战”
2012.5
公司成立，
8月“快的打车”
上线运营
2013.7
覆盖城市：70+
2013.12
收购“大黄蜂”，
开启商务车业务
2014.4
“一号专车”上线
日订单：600w+
2014.9
“一号专车”覆盖全
国32个城市

挑战与解决方案

快的1.0系统架构概览



挑战与解决方案



架构演变总体目标：高可用，高伸缩，高扩展

客户端架构：

速度，流量——SPDY协议，私有协议

服务层架构：

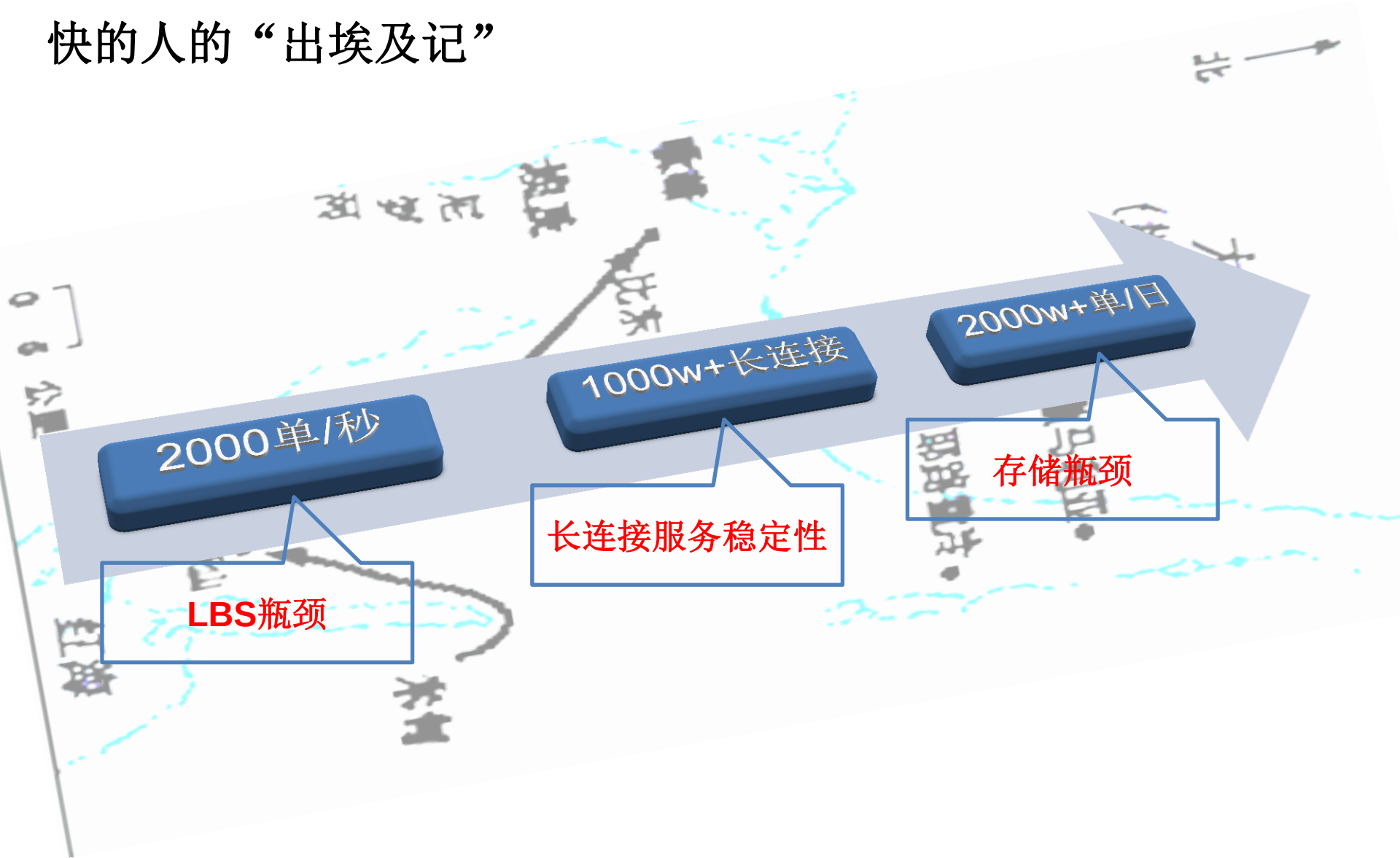
复用，解耦——服务化(SOA)

存储层架构：

水平伸缩——KV化

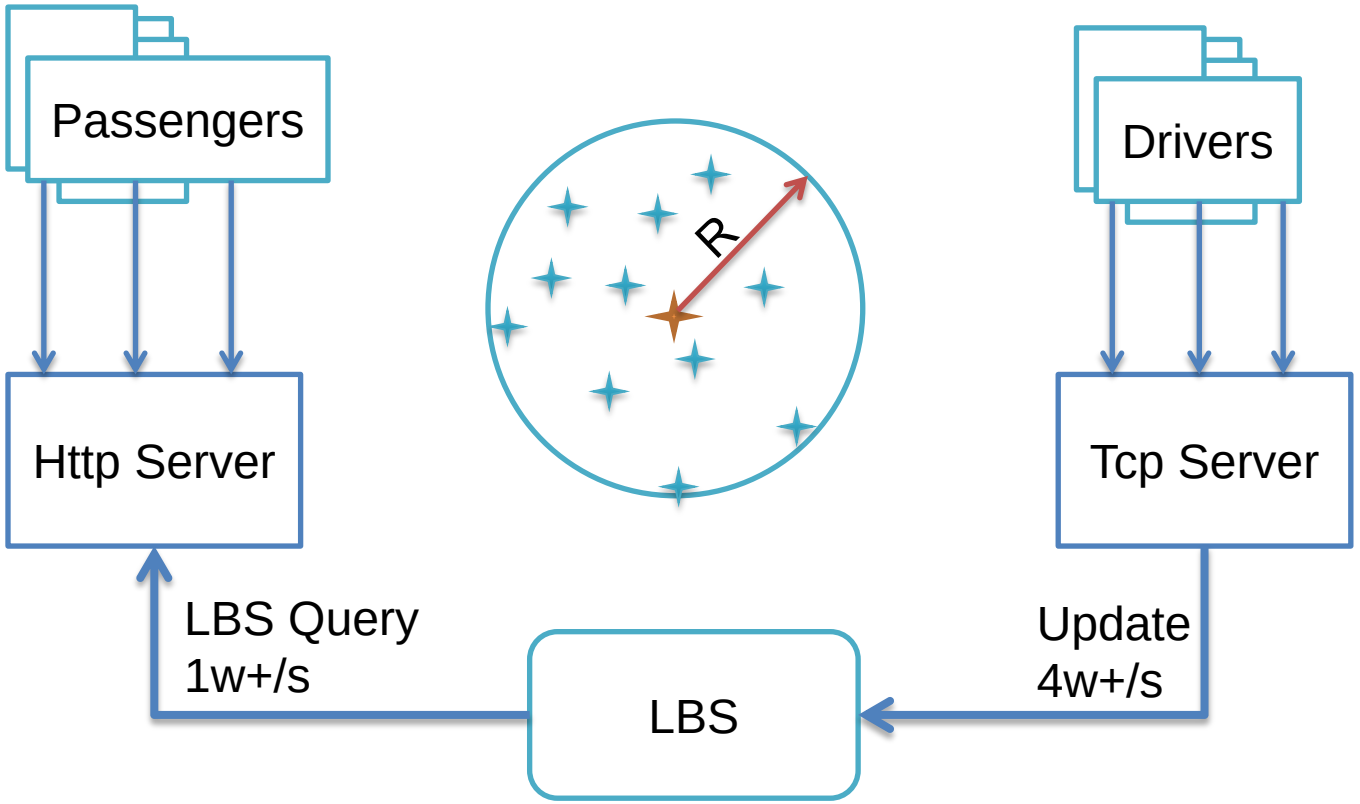
挑战与解决方案

快的人的“出埃及记”



LBS瓶颈及解决方案

快的 **LBS**主要应用场景



LBS瓶颈及解决方案

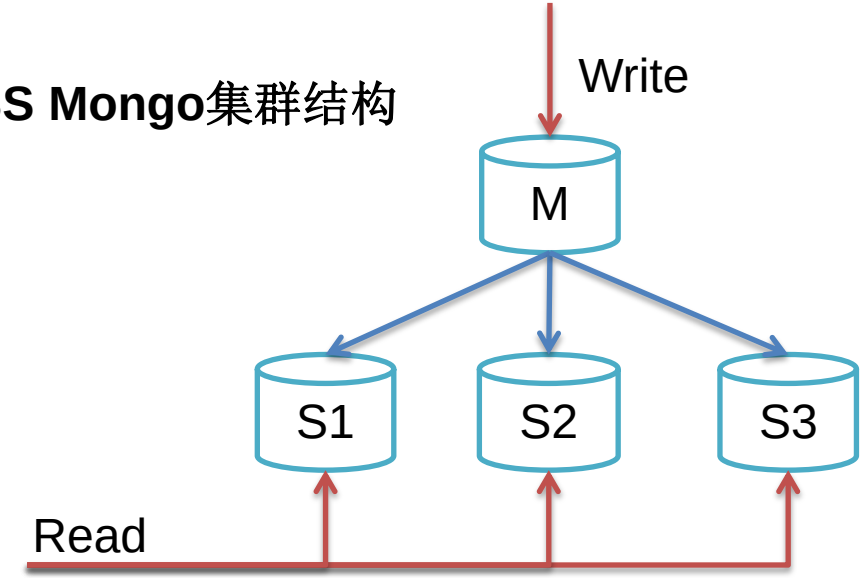
LBS可选方案

LBS瓶颈及解决方案

MongoDB as LBS Server

- ◆对实时更新支持较好
- ◆通过副本集可以很容易地实现读写分离及负载均衡
- ◆开发、部署较简单

快的LBS Mongo集群结构



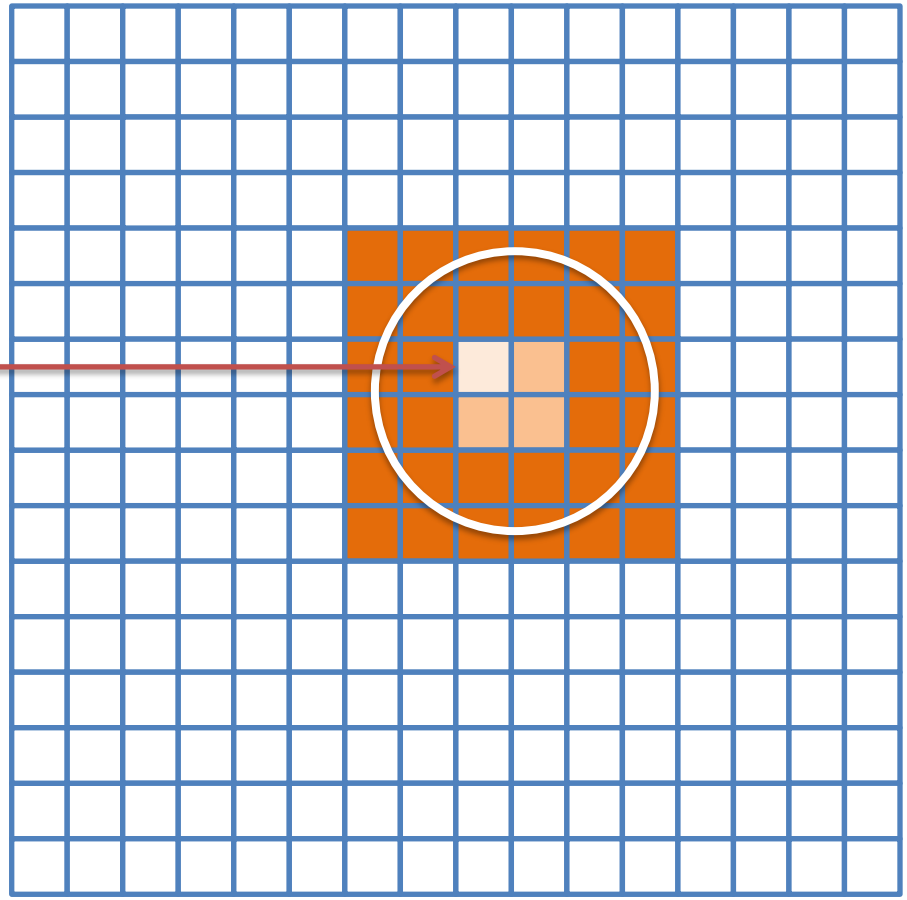
LBS瓶颈及解决方案

MongoDB 2d 查询原理(2.6版本)

```
db.runCommand( {
                geoNear: “mycollection”,
                near : [120.5842,30.6597],
                maxDistance:5
               });
```

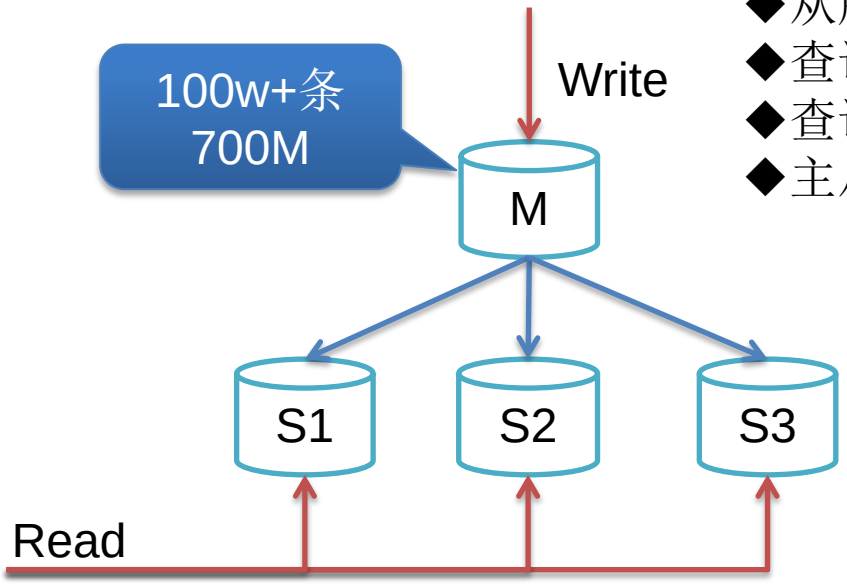
↓

```
db.mycollection.find({loc:” wxec4e02”});
db.mycollection.find({loc:” wxdf0ef3”});
……
```



LBS瓶颈及解决方案

MongoDB geo 瓶颈



读写都很密集(4w+/s写,1w+/s读)时出现的问题:

- ◆从服务器 cpu负载急剧上升
- ◆查询性能急剧降低(大量查询耗时超过800毫秒)
- ◆查询吞吐量大幅降低
- ◆主从复制出现较大的延迟

原因：锁等待

1.Mongodb 目前(2.6.4版本)锁是集合别的锁，每一次写都会锁住整个集合

2.每一次LBS查询，会分解成许多次单独的子查询，增大了整个查询的锁等待概率

解决方案：

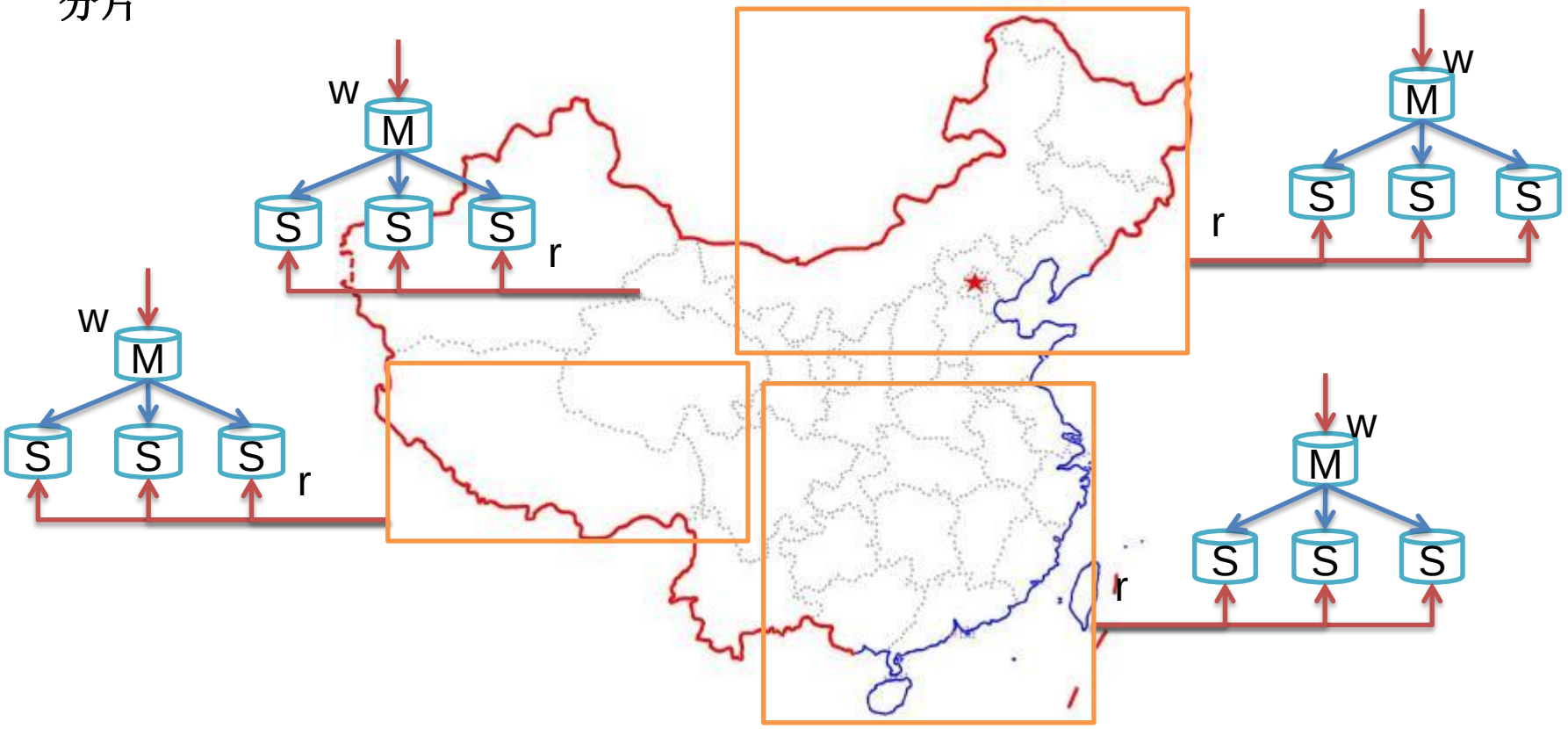
A. mongodb行锁(2.8版本，据说)

B. 分片

C. LBS From Scratch

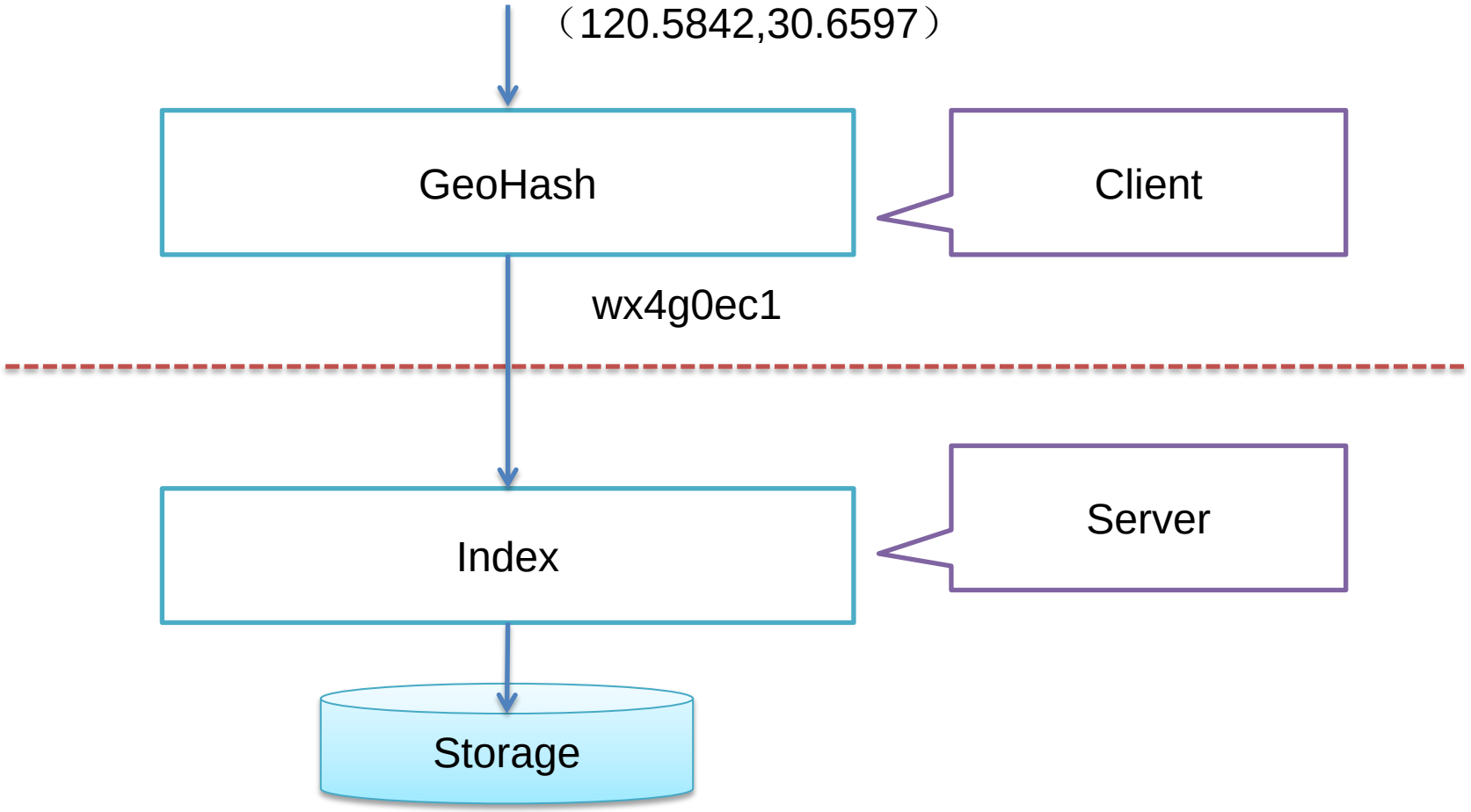
LBS瓶颈及解决方案

分片



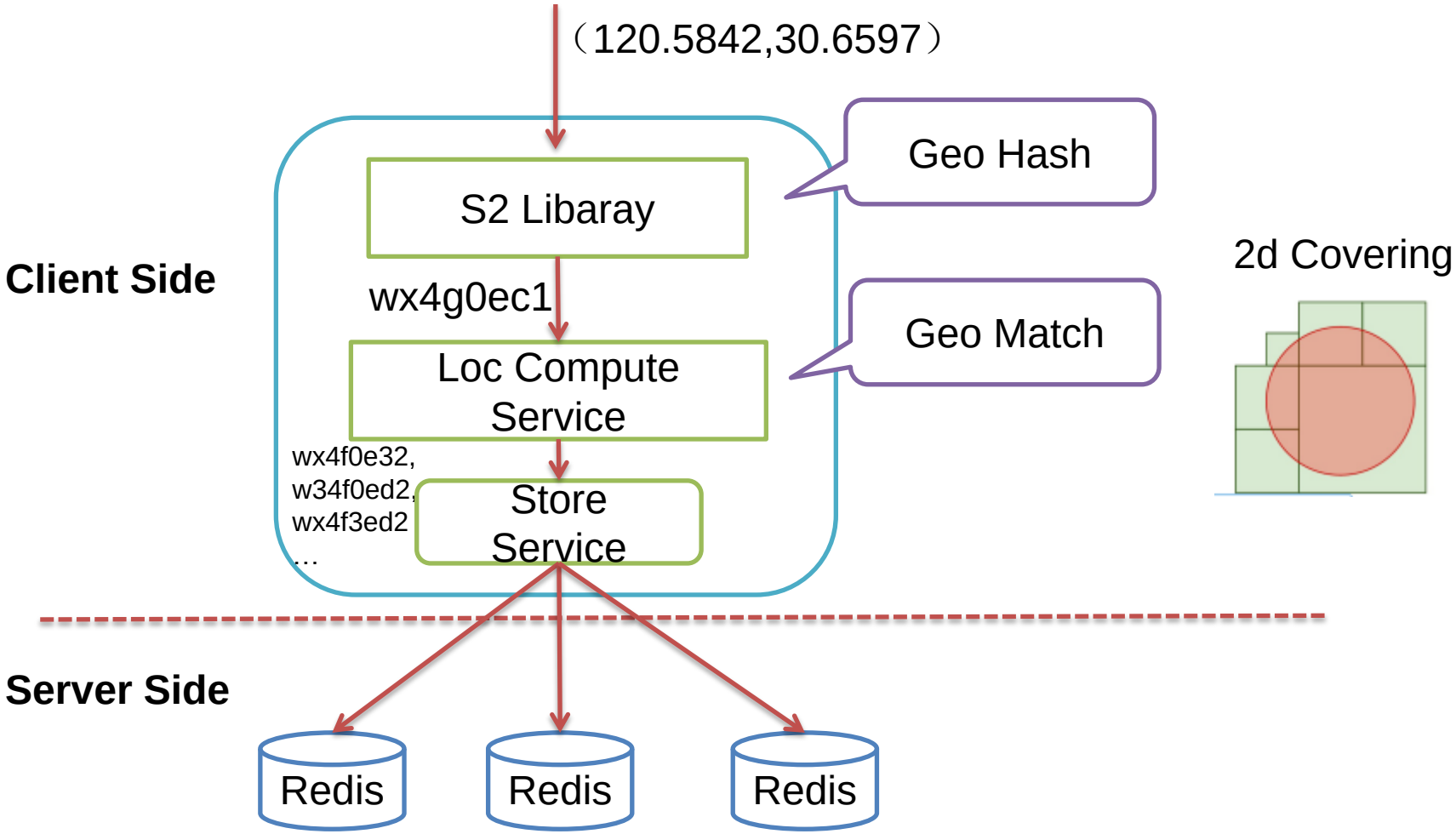
LBS瓶颈及解决方案

LBS From Scratch

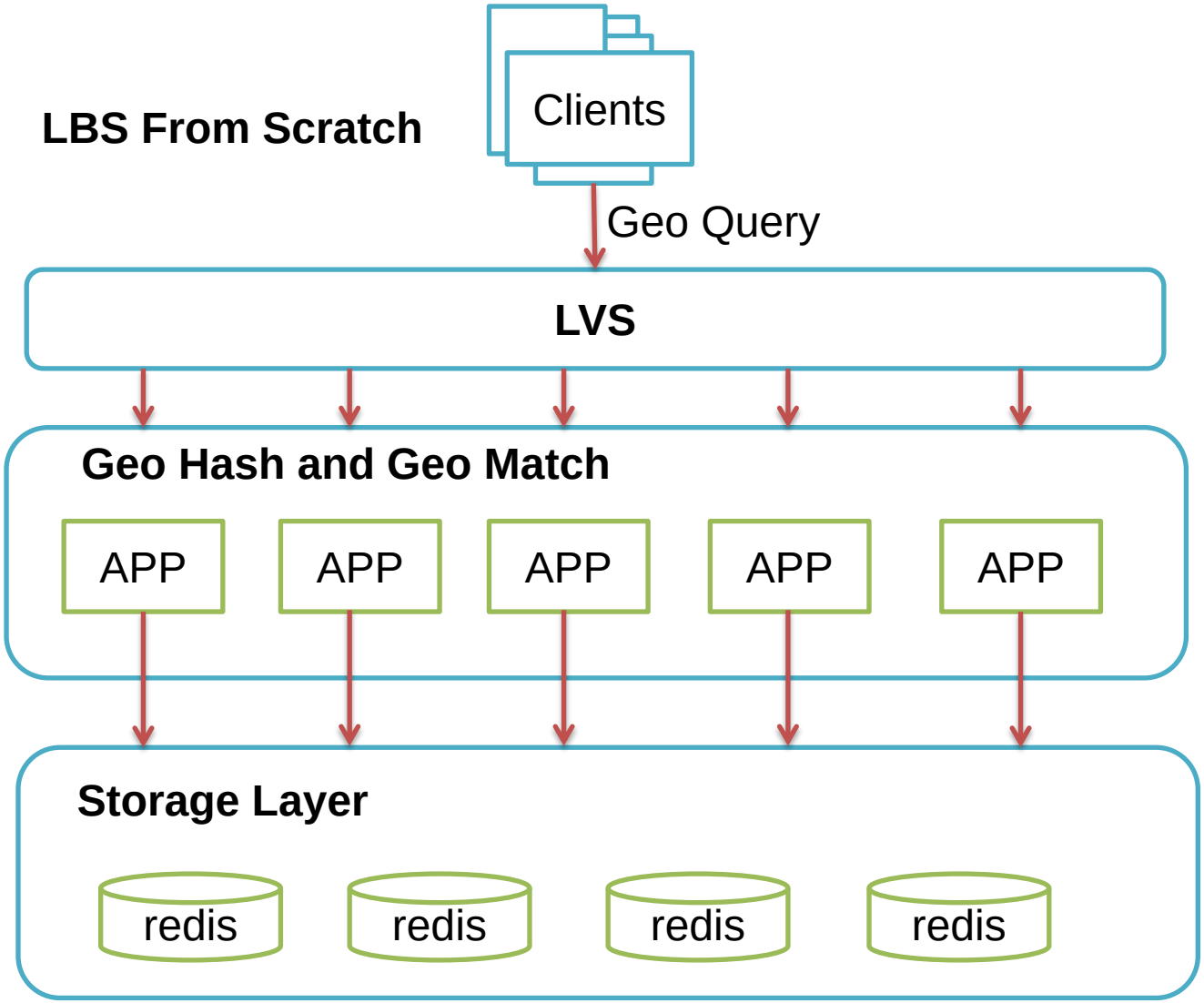


LBS瓶颈及解决方案

LBS From Scratch



LBS瓶颈及解决方案



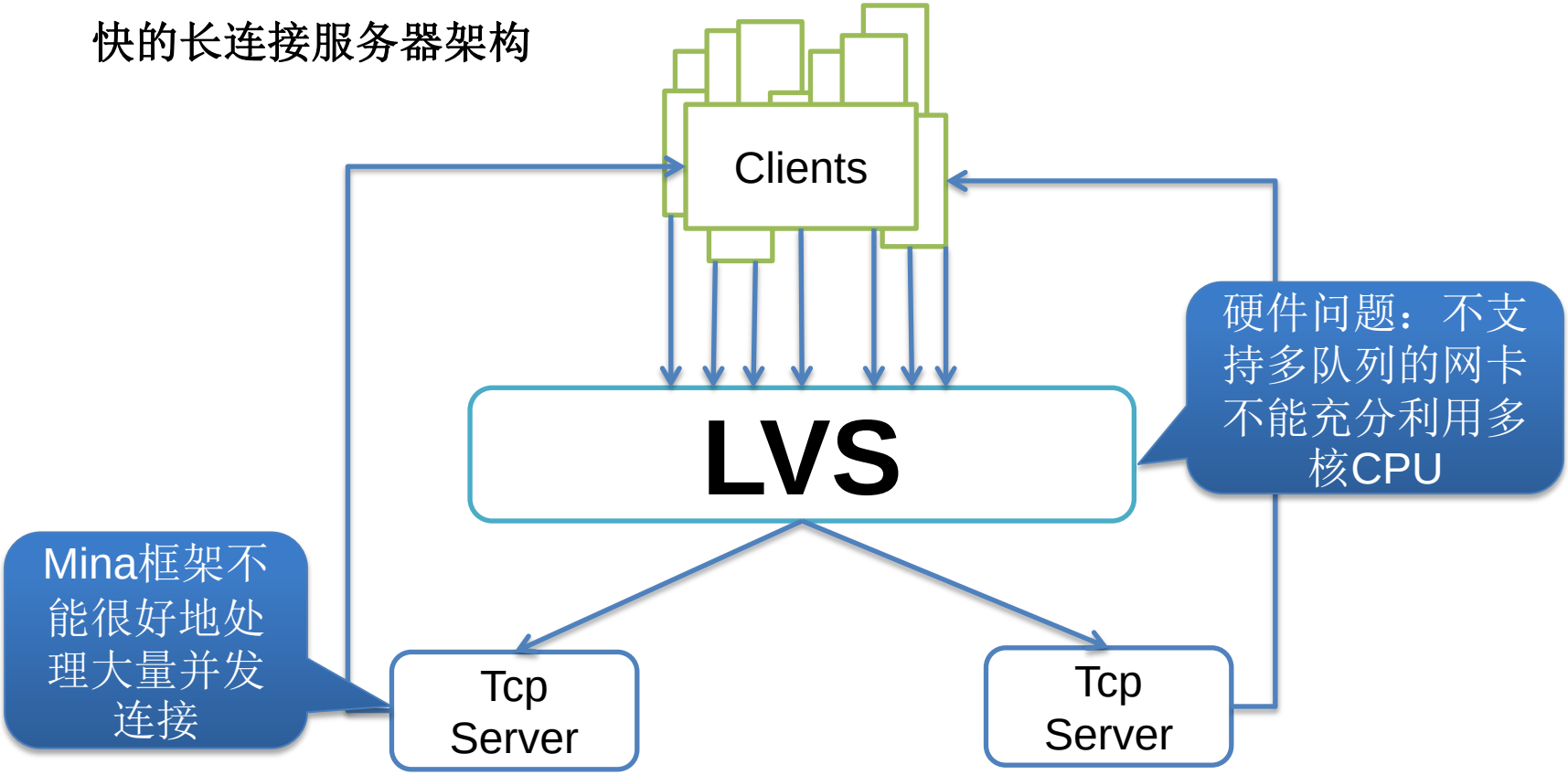
性能(3台redis)
写：15w+
读：18w+

伸缩性
GeoHash,GeoMatch放在客户端，很容易水平伸缩，服务端的存储通过一致性hash算法实现伸缩性

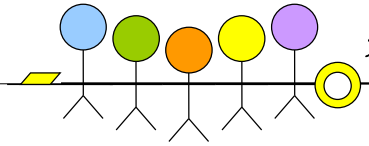
扩展性
GeoHash,GeoMatch,Storage 都设计成插件，便于替换

长连接服务稳定性

快的长连接服务器架构



长连接服务稳定性



长连接服务器遇到的主要问题

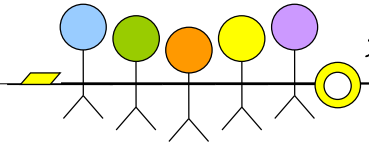
硬件：

不支持多队列的网卡，IO中断都被分配到了一个cpu核上，大量数据包到来的情况下，单个cpu核无法全部处理，导致LVS不断丢包，客户端长连接经常断线。

解决方案：

更换支持硬件多队列的网卡(Intel 82575、82576，Boardcom的57711等， linux 内核版本需要在2.6.21以上)

长连接服务稳定性



长连接服务器遇到的主要问题

软件：

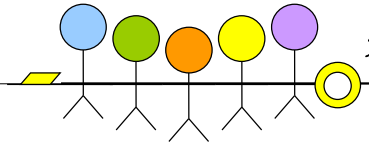
Mina框架的问题

- 内存使用控制不够细粒度，垃圾回收难以有效控制
- 空闲连接检查效率不高，在大量连接的情况下会出现周期性CPU 使用率飙高
- 编解码组件在高并发下会出现消息被截断的情况

快的长连接业务的主要特点

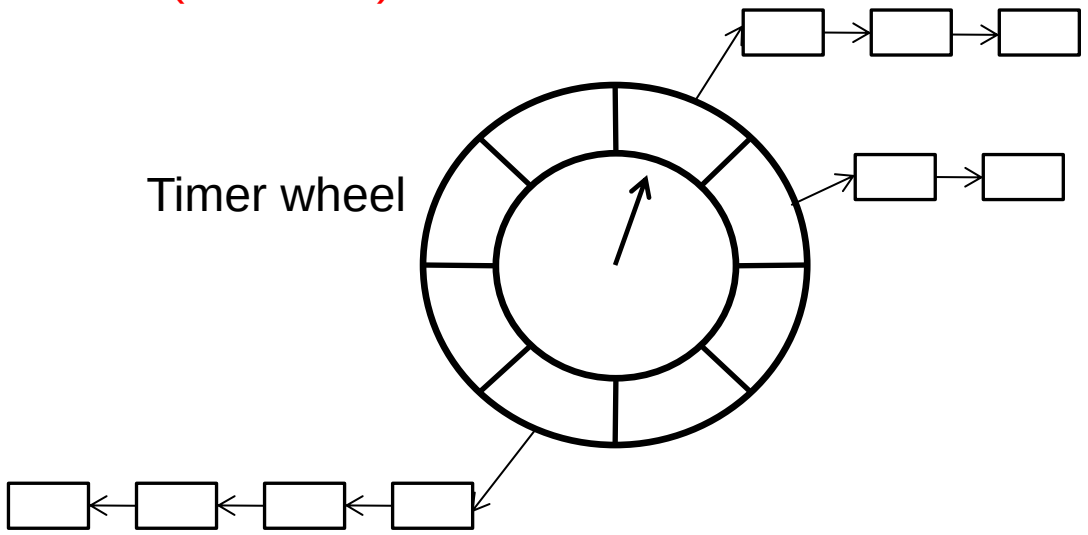
- 大量的广播
- 消息推送具有不同的优先级

长连接服务稳定性



快的自己的AIO框架（基于java 7实现）

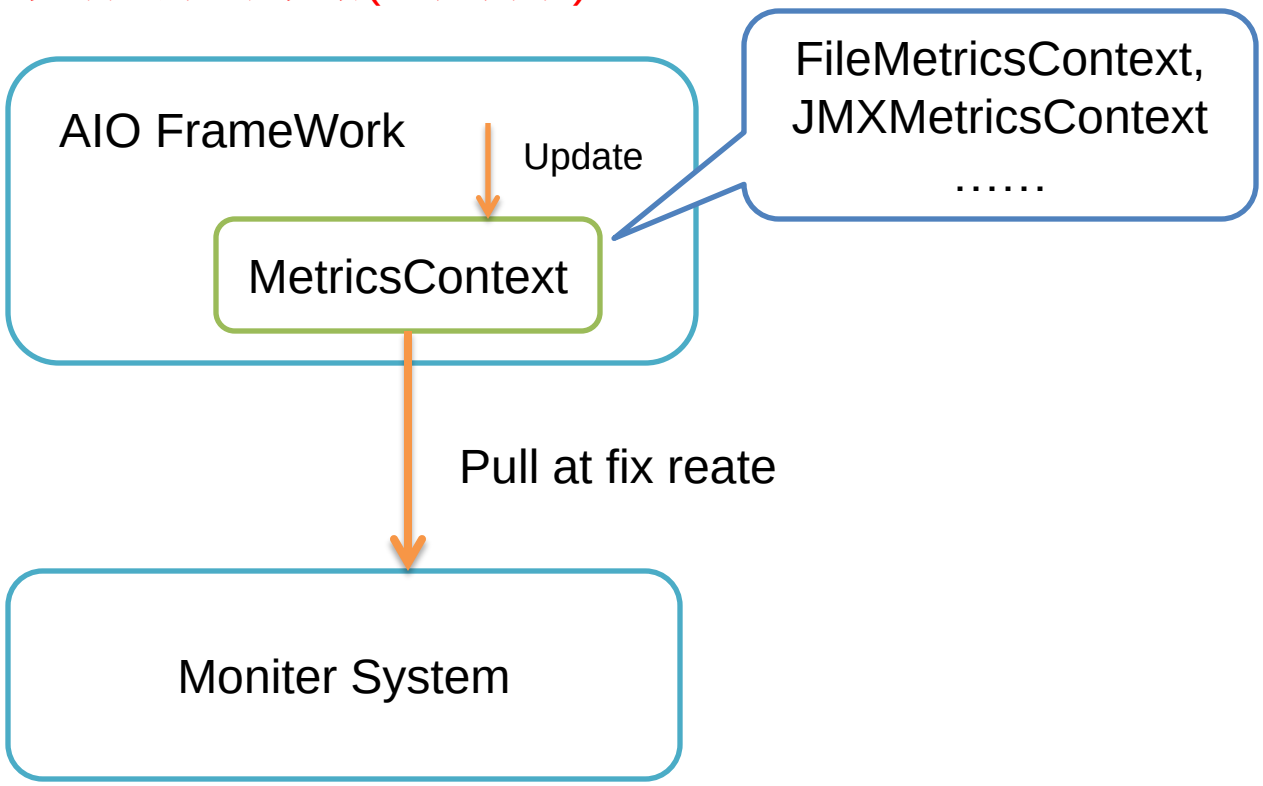
- 资源(主要是ByteBuffer)池化,减少GC造成的影响
- 广播时，一份ByteBuffer复用到多个通道，减少内存拷贝
- 使用TimerWheel检测空闲连接，消除空闲连接检测造成的 CPU尖峰
- 支持按优先级发送数据
- 完善的统计数据(监控需求)



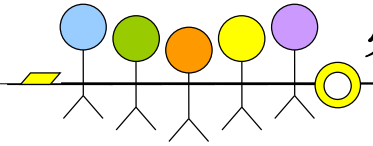
长连接服务稳定性

快的自己的AIO框架（基于java 7实现）

◆完善的统计数据(监控需求)



长连接服务稳定性



火龙果讲堂

uml.org.cn

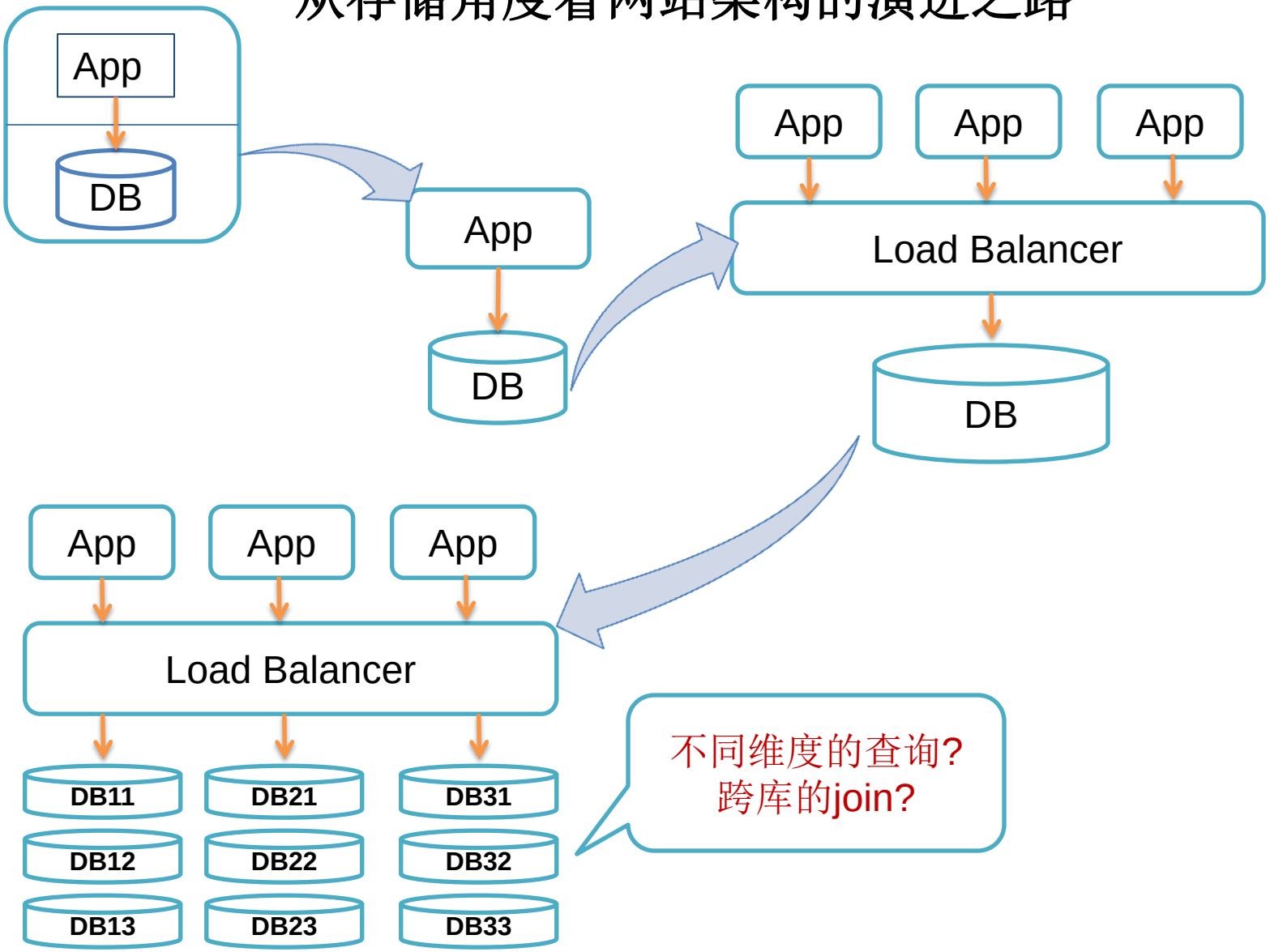
快的自己的AIO框架（基于java 7实现）

最近分钟报表

时间	在线司机数	在线乘客数	连接数	读发生的次数	写发生的次数	写超时次数	写超时次数环比	写最大耗时	写平均耗时	写最大字节数	写平均字节数	读最大字节数	读平均字节数
14:42	241551	166556	357289	1270431	679920	36	32%↓	810	0	22621	202	1024	112
14:41	241476	166669	357345	1264137	676192	53	15%↑	810	0	22621	200	1024	112
14:40	241445	166801	357432	1277928	679820	46	10%↑	810	0	22621	196	1024	114
14:39	241579	166211	356935	1298952	685049	42	19%↓	810	0	22621	203	1024	118
14:38	241482	166006	356640	1301380	688634	52	5%↓	810	0	22621	205	1024	119
14:37	241423	166455	357006	1284058	696560	55	7%↓	810	0	22621	213	1024	110
14:36	241392	166588	357152	1289045	699065	59	2%↓	810	0	22621	208	1024	111
14:35	241332	166854	357341	1276167	687871	60	2%↓	810	0	22621	208	1024	111
14:34	241392	166242	356810	1283040	686538	61	53%↑	810	0	22621	205	1024	113
14:33	241396	166254	356866	1294601	686853	40	9%↓	810	0	22621	205	1024	116
14:32	241306	166624	357154	1308168	690895	44	0%↑	810	0	22621	202	1024	118
14:31	241377	166787	357418	1307065	700170	44	13%↑	810	0	22621	210	1024	115
14:30	241273	166861	357397	1299144	691567	39	0%↑	810	0	22621	209	1024	116
14:29	241239	166495	357013	1303471	691801	39	13%↓	810	0	22621	211	1024	115
14:28	241131	166417	356848	1299919	678364	45	18%↑	810	0	22621	199	1024	119
14:27	241099	166684	357062	1300961	685325	38	3%↑	810	0	22621	200	1024	117
14:26	241216	166608	357099	1297001	679577	37	12%↑	810	0	22621	200	1024	118
14:25	241323	166807	357424	1308471	685581	33	31%↓	810	0	22621	200	1024	118
14:24	241283	166696	357277	1316356	692745	48	13%↓	810	0	22621	205	1024	119
14:23	241217	166851	357391	1306436	689543	55	96%↑	810	0	22621	206	1024	118

存储瓶颈

从存储角度看网站架构的演进之路



存储瓶颈

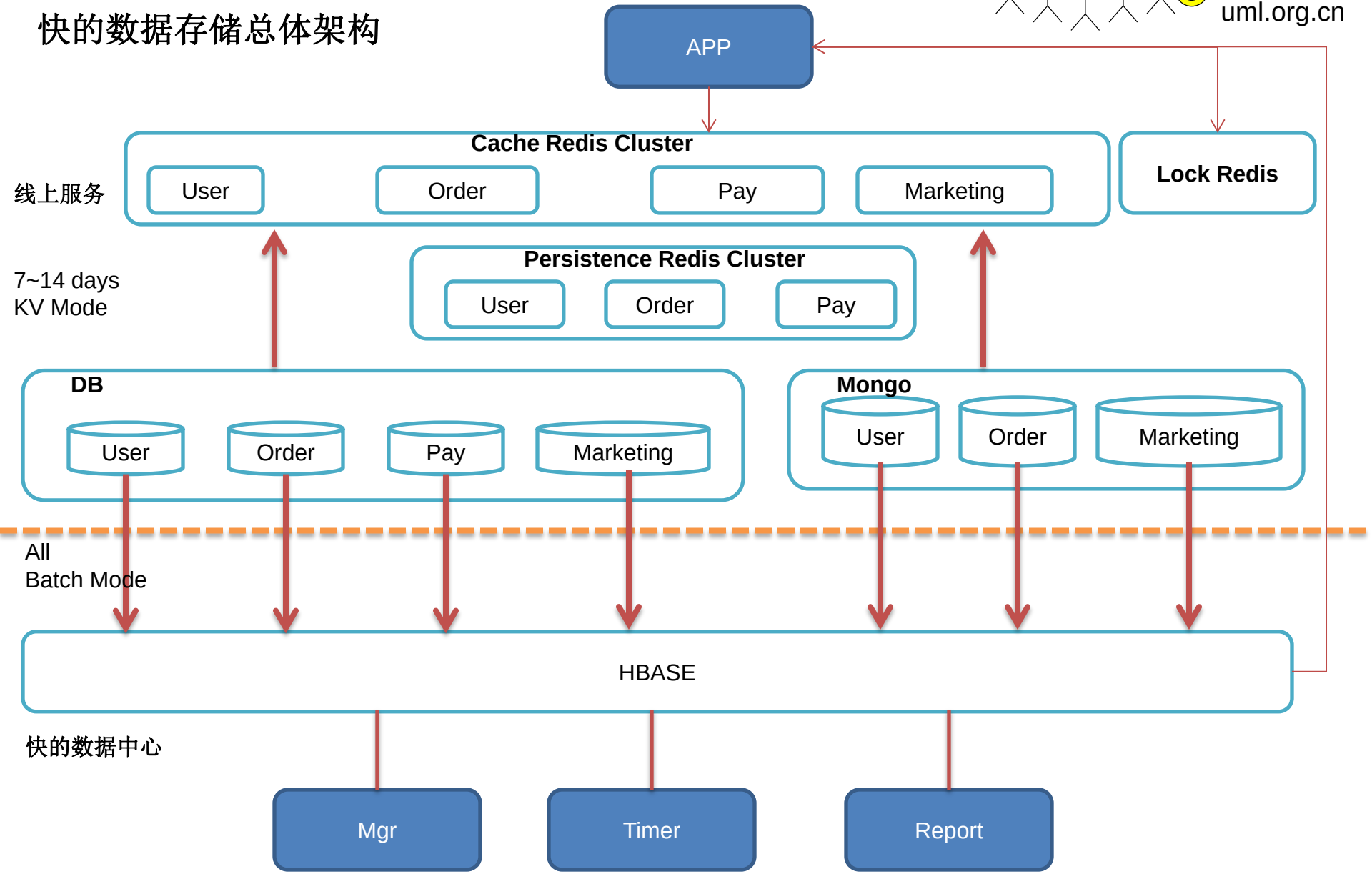
快的数据库瓶颈：
10亿+条数据的表，1500w+日增量
业务数据特点：
热点数据集中在2-3天内访问，一周以前的数据很少访问
方案：
“KV化”

什么是KV化？

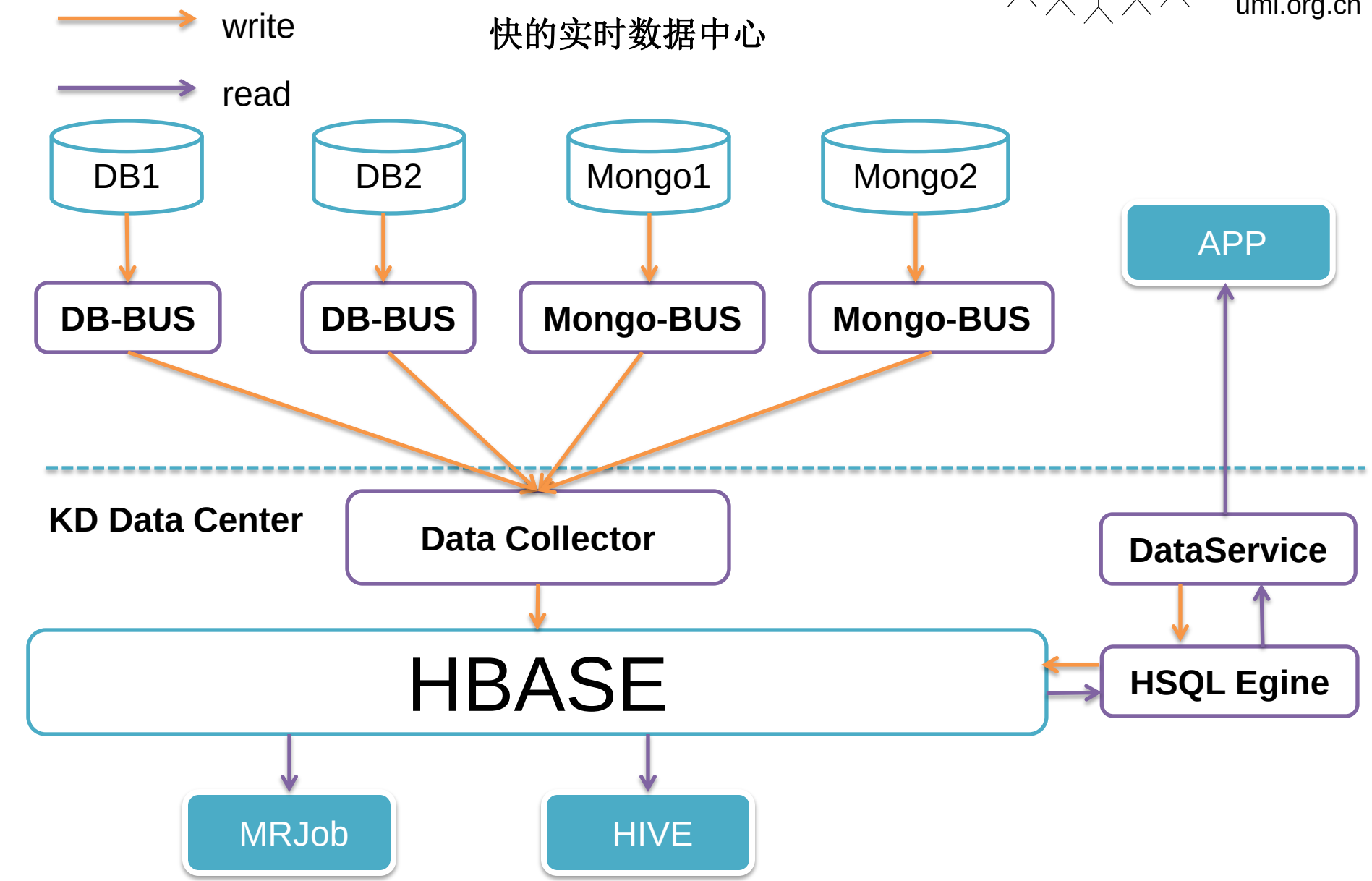
KV化是指**线上数据**的访问方式按照key/value的方式进行，数据的存取都是根据单个的key获取单个的值(key可以是redis存储的键，也可以是mysql的主键或摸个二级索引，值可以是redis中存储的值，也可以是mysql单表中的一条记录).

存储瓶颈

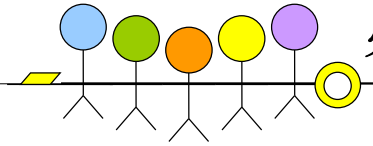
快的数据存储总体架构



存储瓶颈


write
read
快的实时数据中心
DB1
DB2
Mongo1
Mongo2
APP
DB-BUS
DB-BUS
Mongo-BUS
Mongo-BUS
KD Data Center
Data Collector
DataService
HBASE
HSQL Egine
MRJob
HIVE

存储瓶颈



快的实时数据中心

HBASE

数据存储核心组件，需要添加二级索引支持

DB-BUS,Mongo-BUS

数据同步工具，用于完成线上数据到HBASE的同步。DB-BUS采用binlog解析(mysql)方式实现，Mongo-BUS采用oplog解析方式实现

Data Collector

数据归集器，用于实现多表归一(Join)

DataService

数据服务接口，实时数据中心的数据存取都通过此接口进行(认证，访问权限控制)

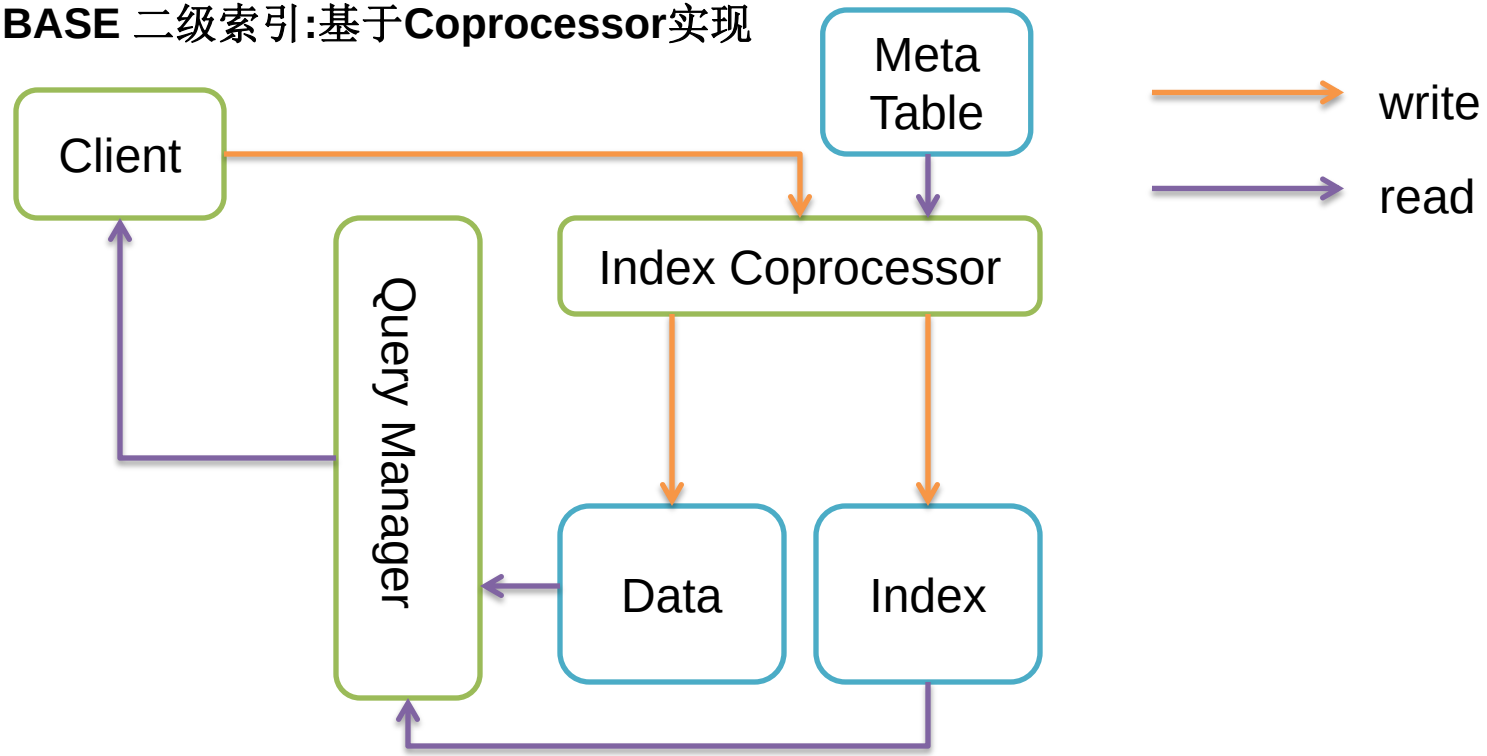
HSQL Egine

一套用于将sql转换为HBASE API的引擎。HBSE是一个无类型的系统，对HBASE而言一切皆为字节数组，因此需要一套引擎用于管理元数据(各个字段的类型)。同时，由于大多数开发人员对HBASE很陌生，但对sql很熟悉，因此需要一个引擎完成sql语句到HBASE API的转换。

存储瓶颈

快的实时数据中心

HBASE 二级索引:基于Coprocessor实现



存储瓶颈

快的实时数据中心

Hbase二级索引实现细节

Index 与Data放到不同的表，查询时，先查索引表，根据索引表返回的结果得到主表 rowkey，再查主表

优点:适合region数量巨大的表
缺点:增加一次网络通信开销

Index Table

Row Key	
a1-r1	
a2-r2	

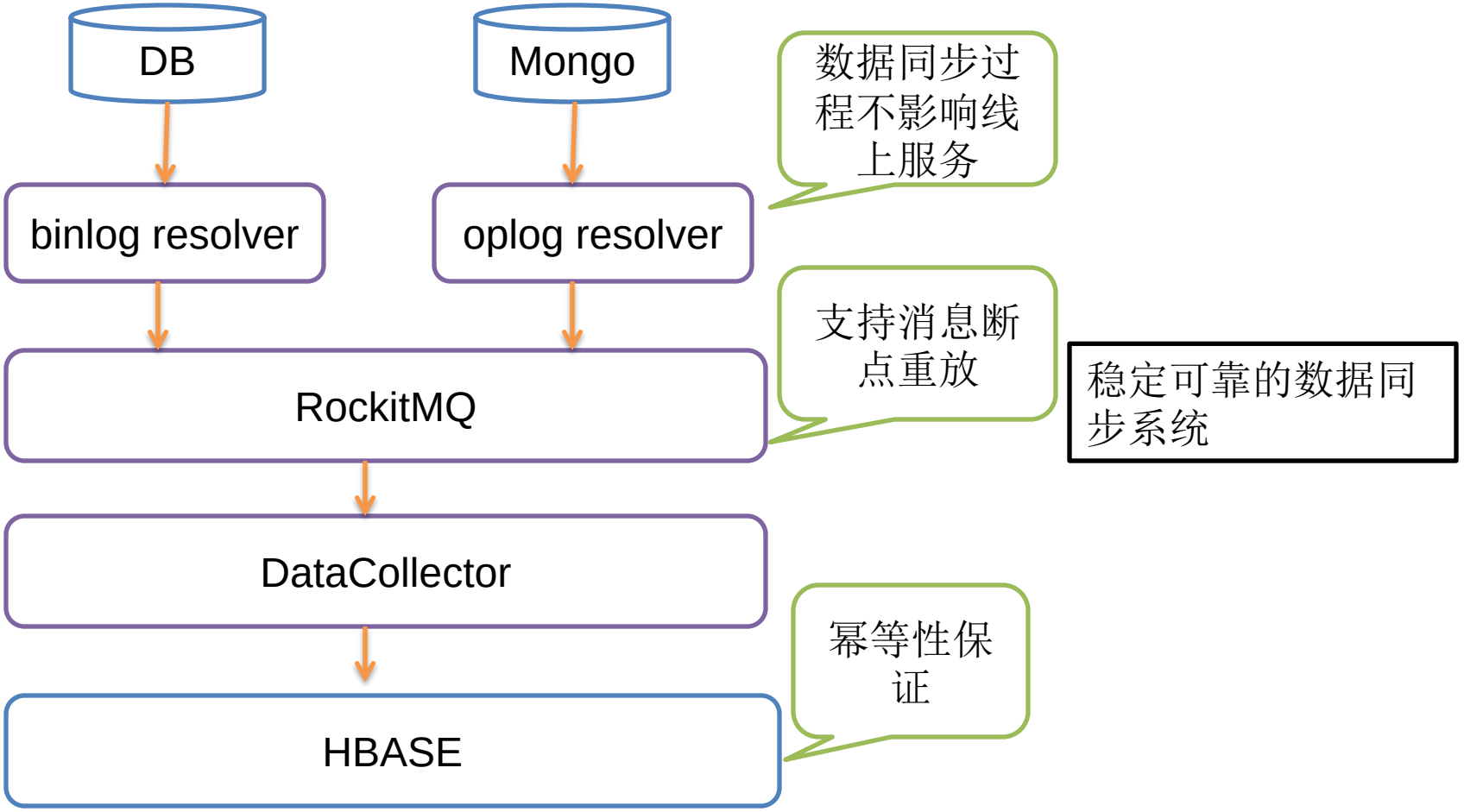
Main Table

Row Key		
r1	a1	d11
r2	a2	d21

存储瓶颈

快的实时数据中心

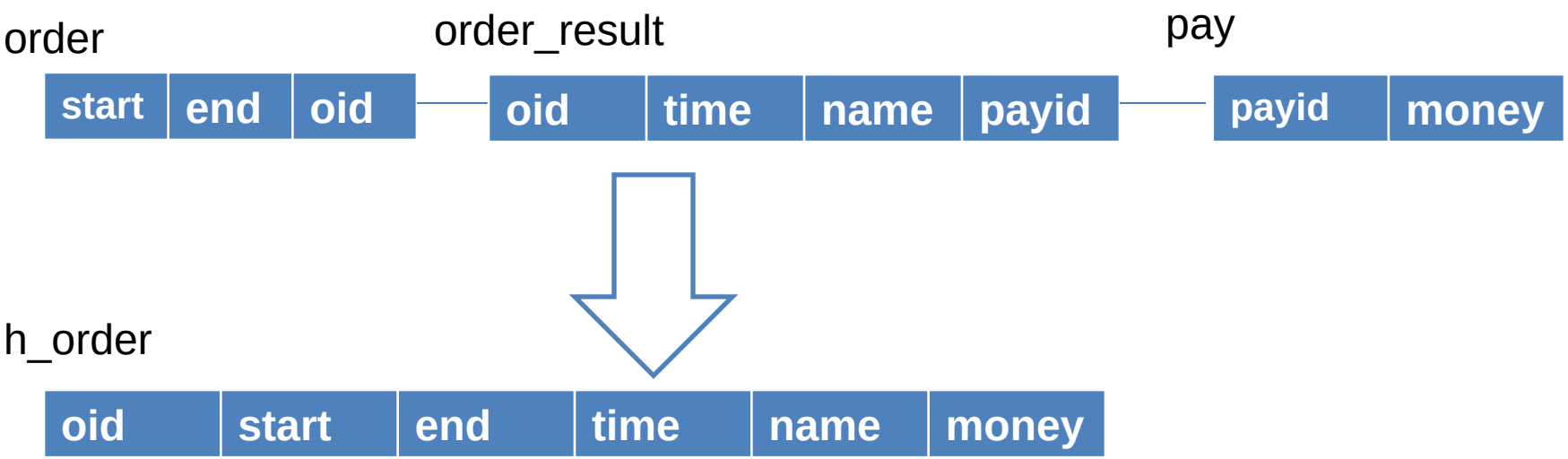
数据同步组件实现细节



存储瓶颈

快的实时数据中心

Data Collector实现细节:多表串联归一

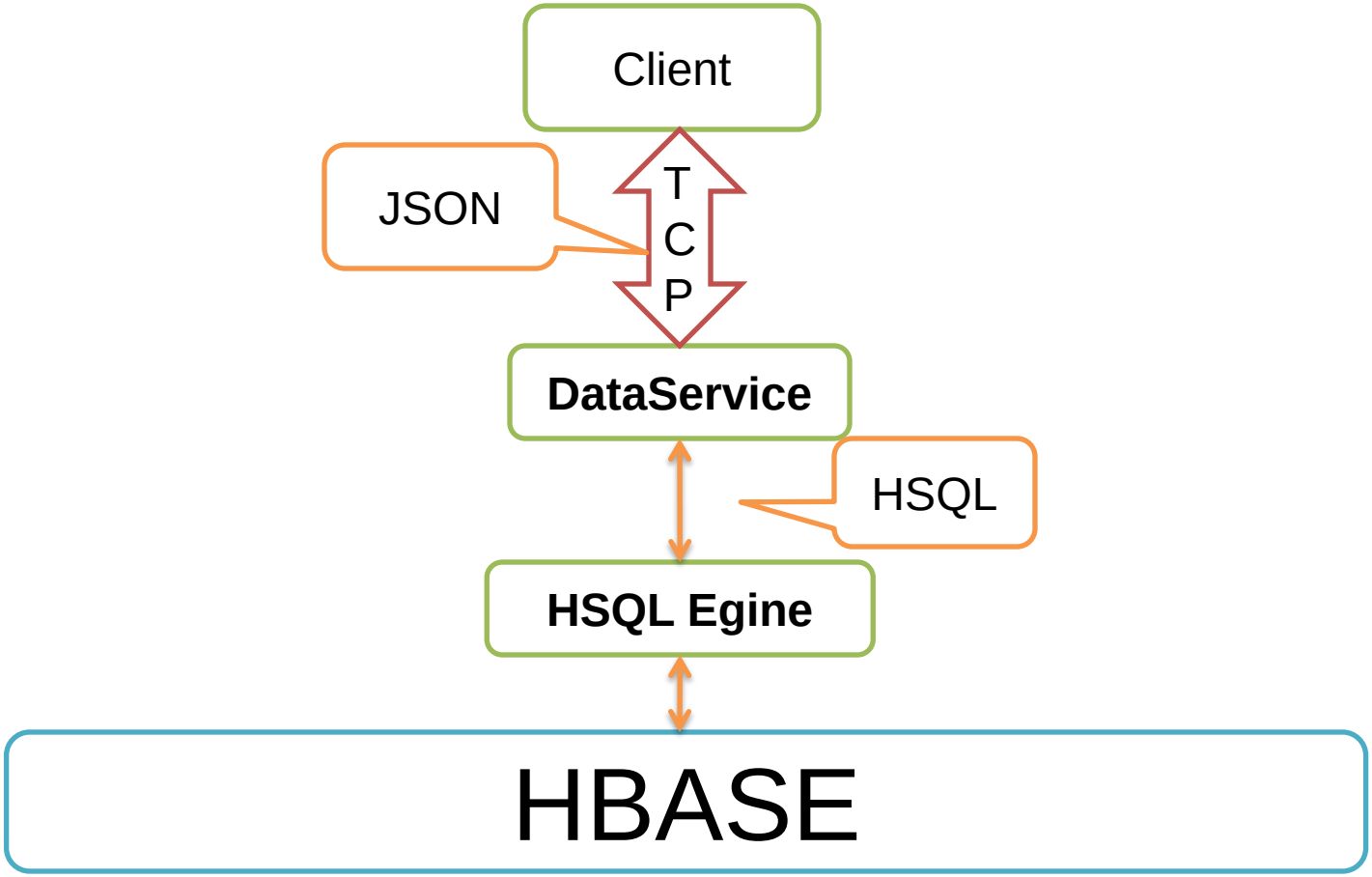


归并流程：
1.先插入主表记录
2.从表记录插入前，先从hbase中查找从表记录对应的rowkey(通过二级索引)，再插入从表记录

存储瓶颈

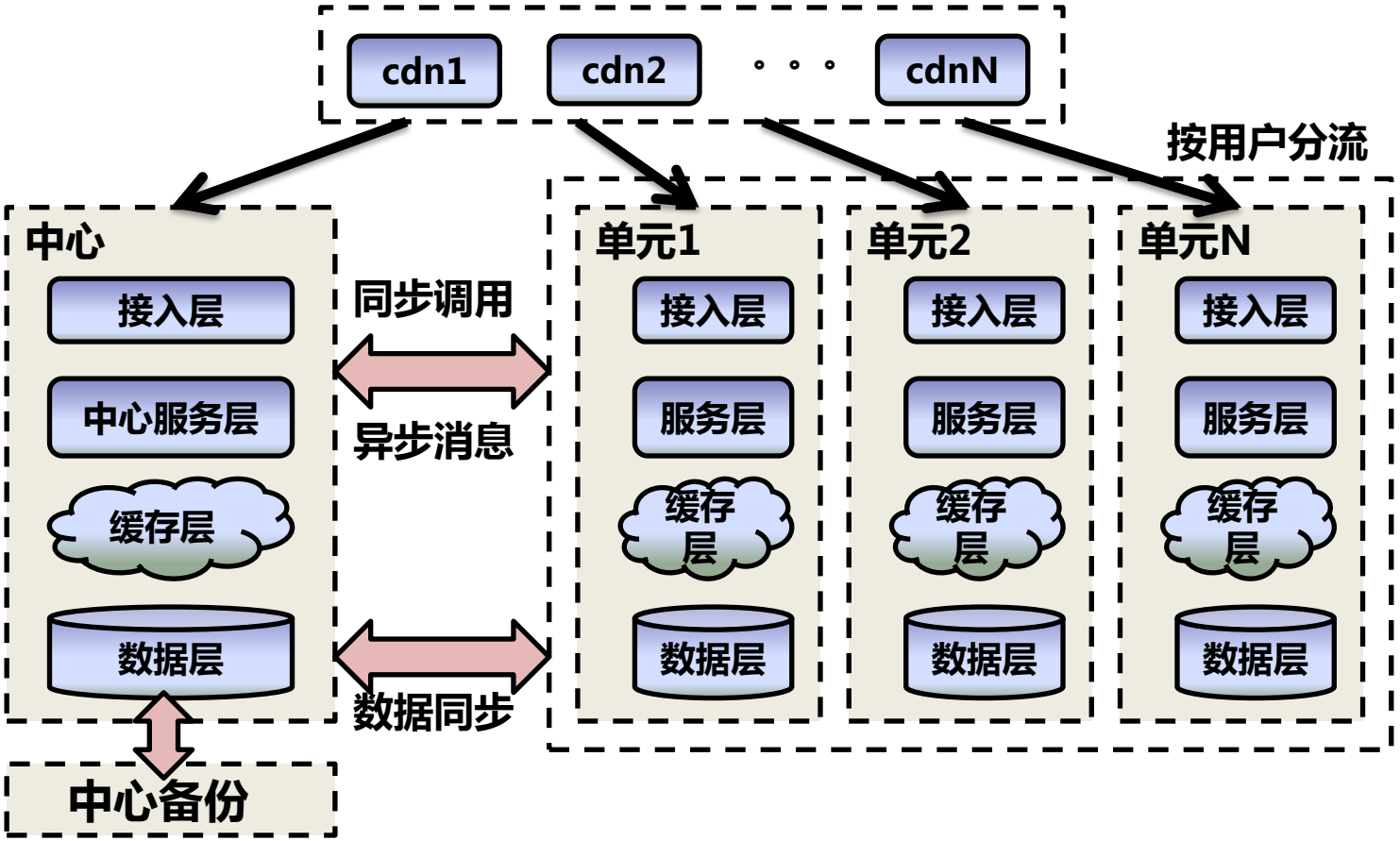
快的实时数据中心

Data Service实现：基于TCP长连接

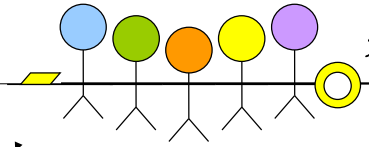


架构规划

多机房容灾


cdn1
cdn2
。。。
cdnN
按用户分流
中心
接入层
中心服务层
缓存层
数据层
同步调用
异步消息
数据同步
单元1
接入层
服务层
缓存层
数据层
单元2
接入层
服务层
缓存层
数据层
单元N
接入层
服务层
缓存层
数据层
中心备份

听众参与



- 如何优化客户端协议以提升性能？

1.http,可以考虑SPDY协议(需要经过充分测试论证)

2.tcp，使用私有协议(快的使用自己设计的TLV-Tag,Length,Value协议)

3.适当采用UDP协议(数据埋点，消息推送等场景)

- 大数据平台架构要点？

1.区分“随机查询”和“顺序查询”两种访问模式并做好隔离

2.数据同步系统必须稳定可靠，且不能影响线上服务

- 如何用好SOA架构？

1.服务治理(平滑降级，依赖关系梳理)非常重要

2.控制接口数量，面向业务设计SOA接口

交流时间

讲座 2015年4月25日 大型分布式网站架构初探

随时听讲座

每天看新文

追随技术信仰