



2017第八届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2017

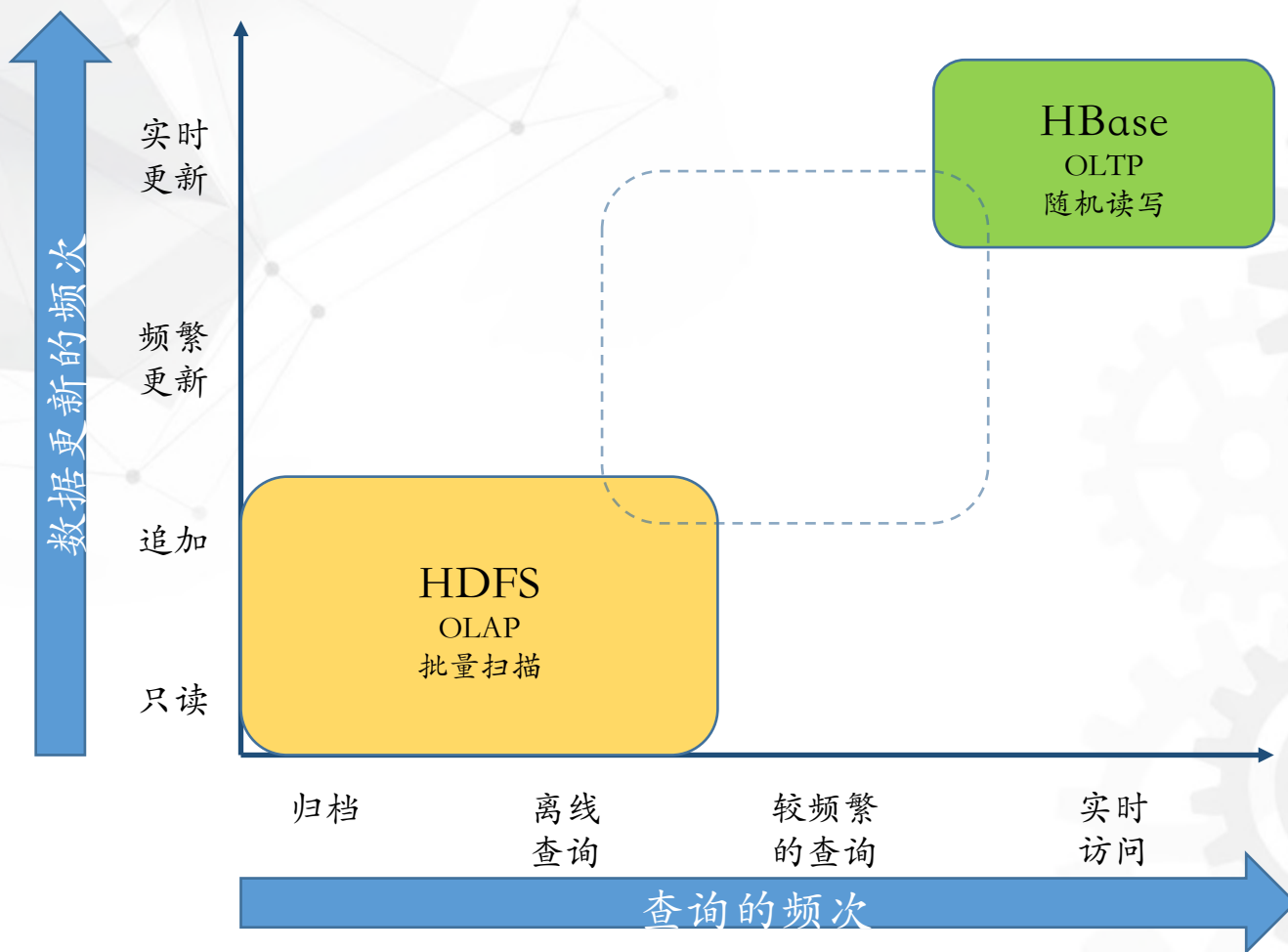
Kudu介绍及实践

小米 张震

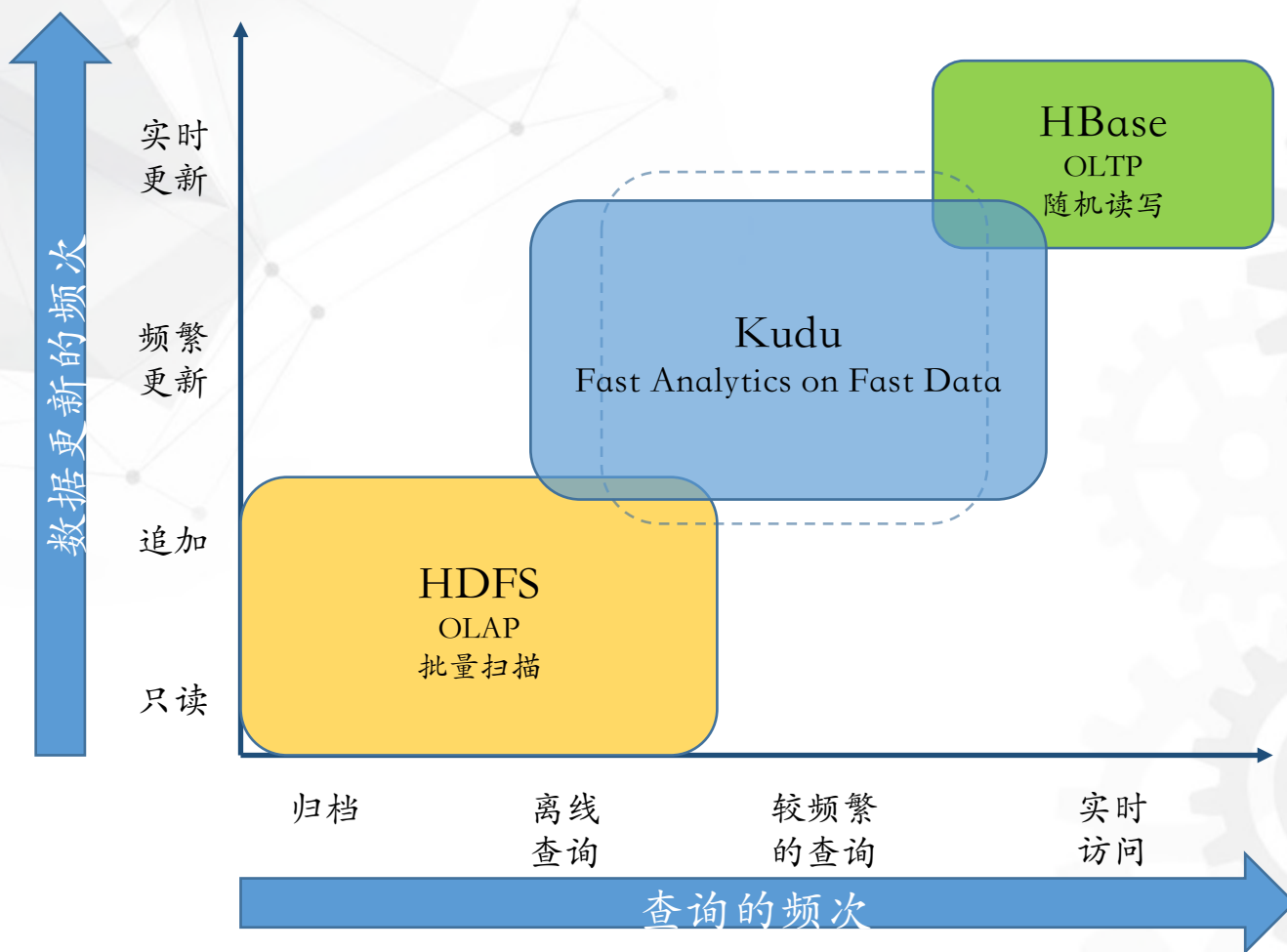
大纲

- Kudu介绍
- Kudu与Spark的整合
- 小米实践

存储系统的现状



Kudu的设计目标



Kudu的设计目标

高性能:

- 快速的批量扫描 2x Parquet
- 低延迟的随机读写 1ms on SSD

可扩展:

- 400 nodes, 1000s of nodes
- low MTTR

关系型数据模型:

- 强schema, 有限列, 不支持BLOBs
- NoSQL APIs (insert/update/delete/scan), Java/C++ Client

事务支持:

- 单行的ACID支持

与Hadoop生态的集成:

- Flume/Impala/Spark

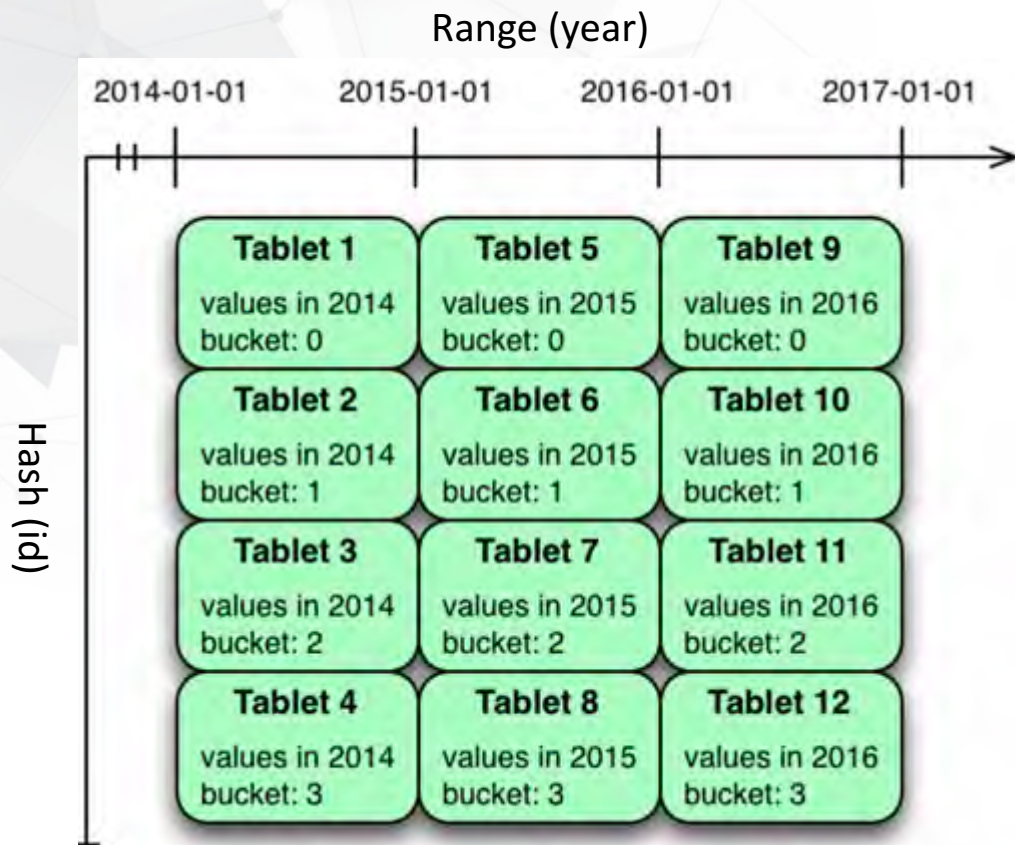
Kudu的数据模型

- 有限固定列
- 强类型

```
CREATE TABLE sales_by_year(  
  year INT,  
  sale_id INT COLPROPERTIES (encoding="bitshuffle"),  
  amount INT,  
  PRIMARY KEY (year, sale_id)  
)  
PARTITION BY HASH (sale_id) PARTITIONS 4,  
RANGE (year)  
(  
  PARTITION 2014 <= VALUES <= 2016,  
  PARTITION VALUE = 2017  
) STORED AS KUDU  
TBLPROPERTIES (replication=3);
```

- 列存储
- 每一列均可以设置 encoding 及压缩方式
- 主键索引
- 范围分区和哈希分区
- 多副本

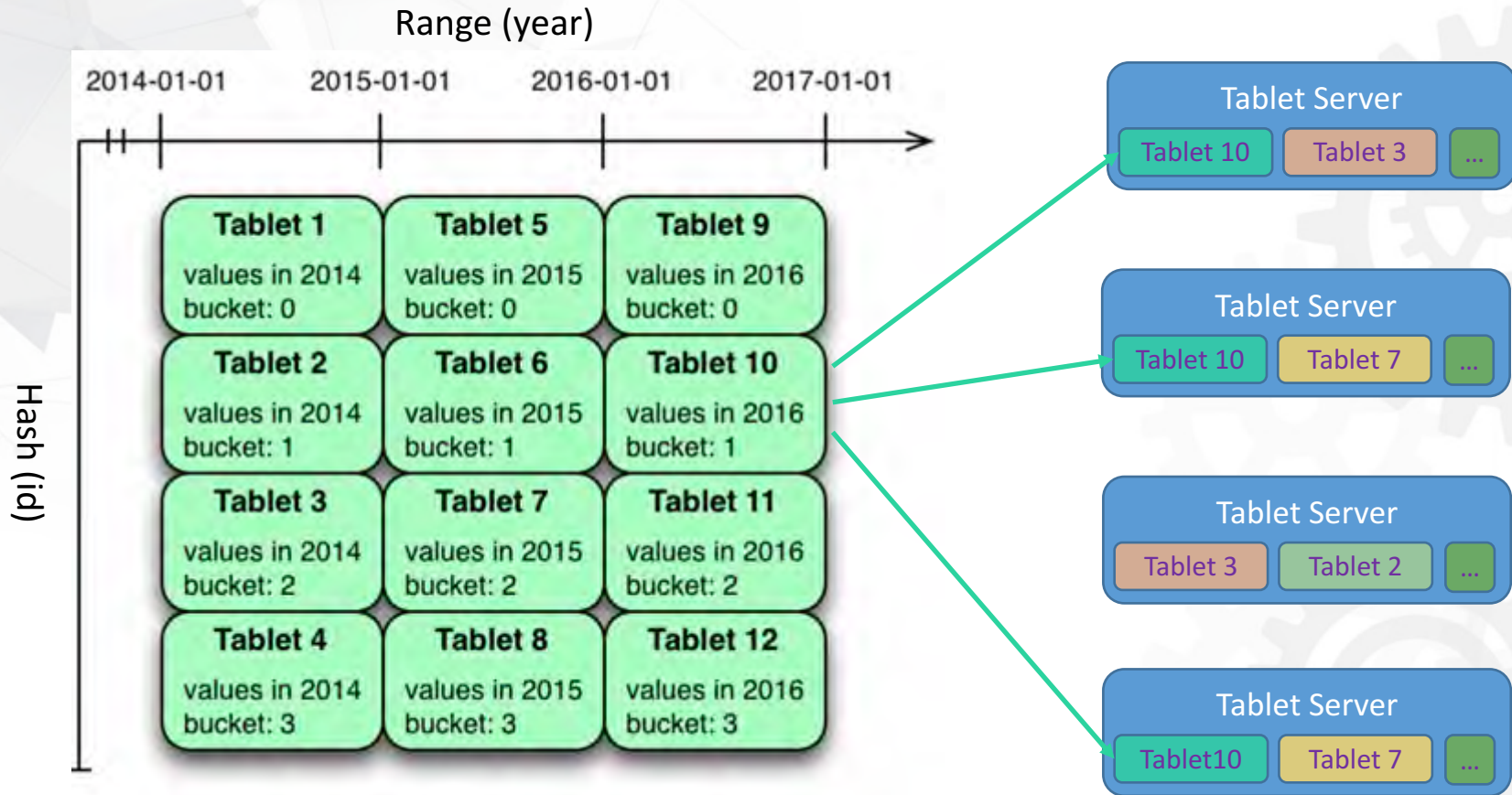
分区和Tablet



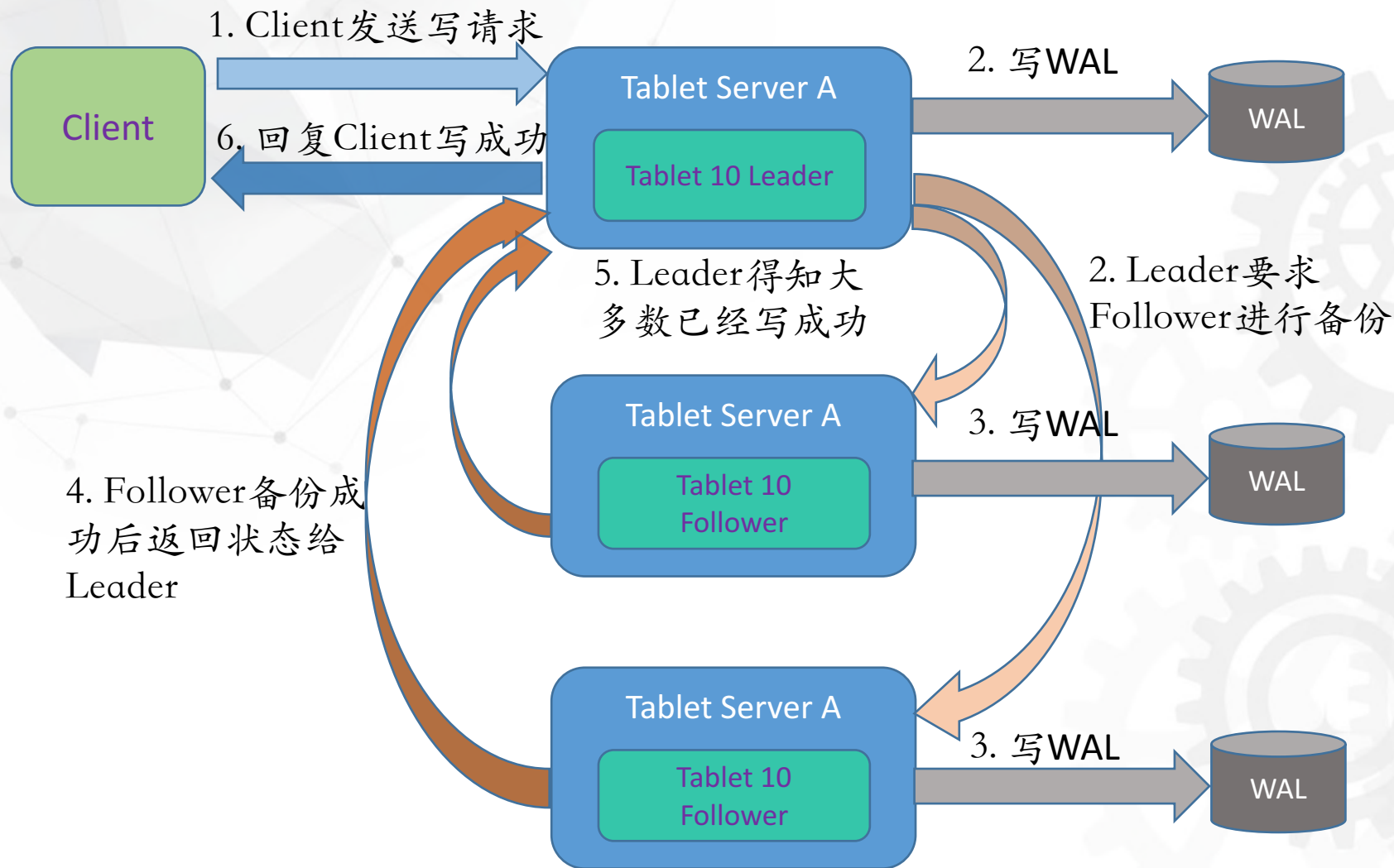
PARTITION BY
HASH (id) PARTITIONS 4,
RANGE (year)

(
PARTITION VALUE = 2014
PARTITION VALUE = 2015
PARTITION VALUE = 2016
)

多备份



多备份 & raft协议



多备份 & 容错

多备份:

- 强一致, 低延迟
- 允许从Follower进行Snapshot读

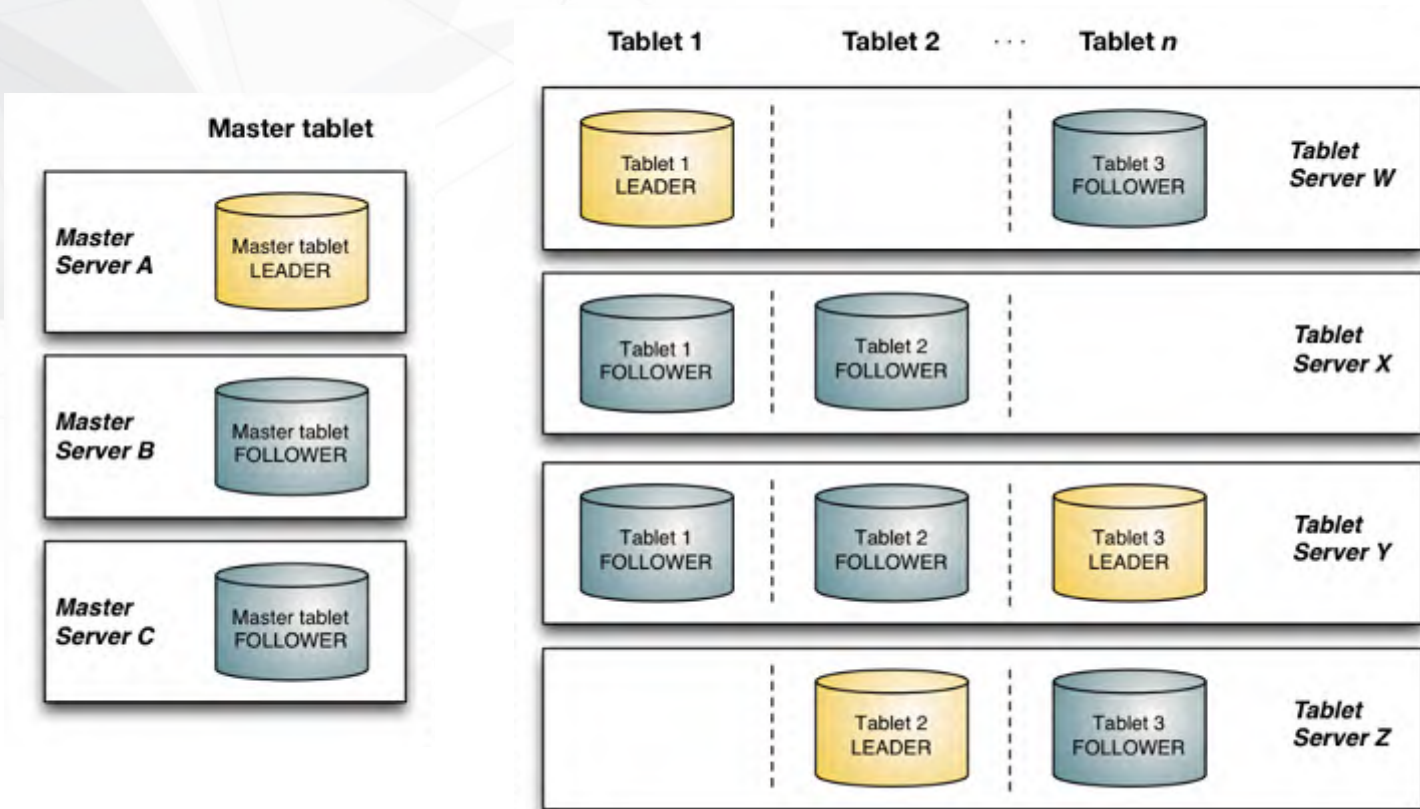
临时失败 (5分钟内):

- Leader临时失败 - 自动选主, 大概在5s内完成
- Follower临时失败 - 少数的Follower失败不会影响读写

永久失败 (超过5分钟):

- 踢出Raft Config, Master会重新选择一台TabletServer部署新的Replica

整体架构



Master

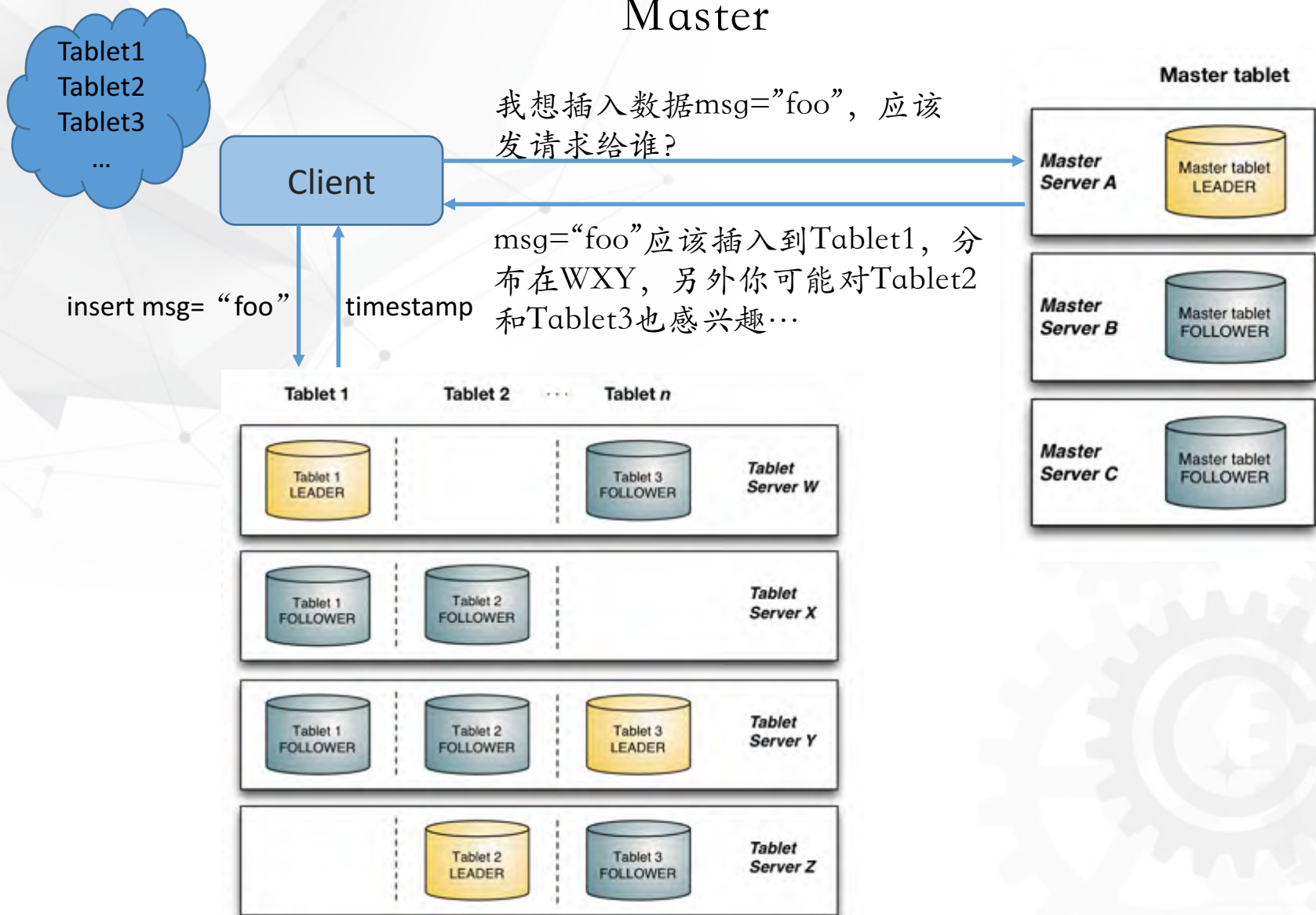
特殊的元表:

- 表结构为: type int8, id int64, meta string
 - type 为 table 或 tablet
 - id 为 table id 或 tablet id
 - meta 为 protobuf 序列化后的元数据
- table 元数据
 - schema, partition schema, 包含哪些 tablets
- tablet 元数据
 - replica locations
 - tablet state

Coordinator:

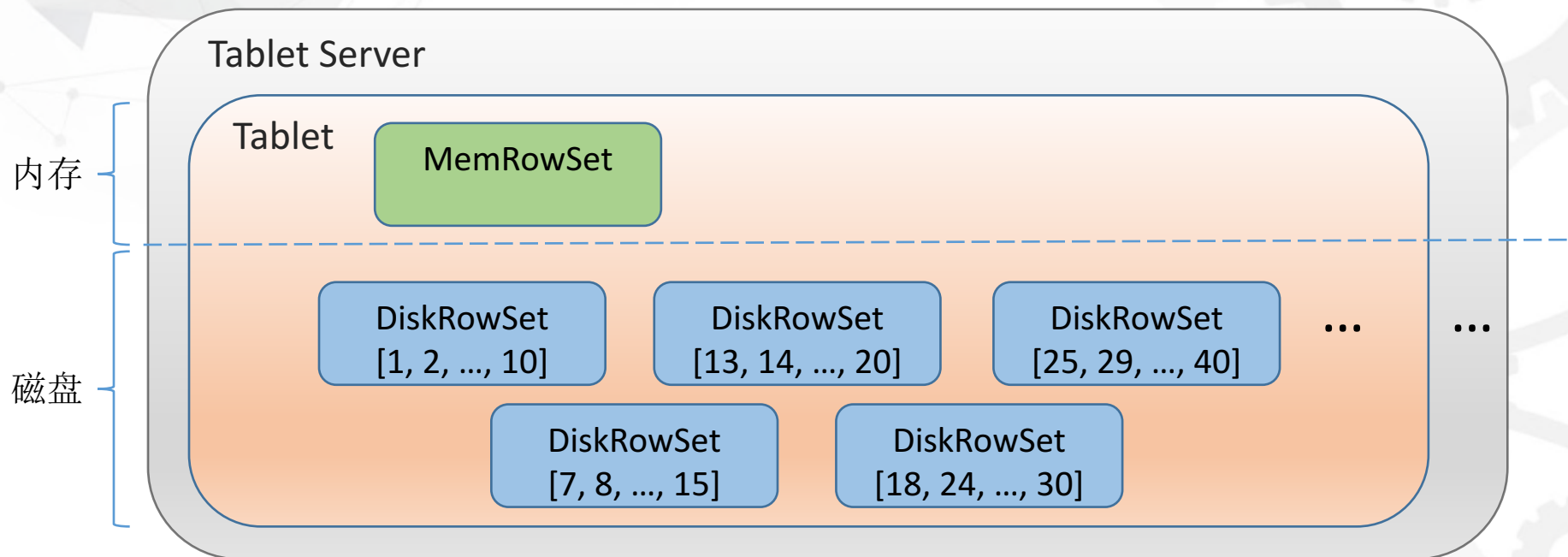
- 接收 TabletServer 的心跳, 跟踪集群状态
- create/alter/drop table
- 对 under-replicated 的 tablet 重新分配 replica

Master



Tablet

- 类LSM(HBase):
 - 每个Tablet包含一个MemRowSet和多个DiskRowSet；新插入的数据存储在MemRowSet中，定期flush成DiskRowSet
 - 不同于LSM，每一个Row只存在于一个RowSet中
- RowSet
 - RowSet内有序，RowSet间无序，不同RowSet所包含的key range允许重叠
 - MemRowSet内部使用B树行存储，DiskRowSet使用列存储



MVCC

- MVCC
 - 对每一行都保存多个版本
- MVCC所提供的能力
 - Snapshot读
 - 历史读
 - 增量备份
 - 同一个Tablet内多行写的原子性
- 一个写操作的执行步骤
 - leader收到写请求，获得要写的行锁
 - 为写操作分配timestamp
 - 通过raft协议备份写操作
 - 真正对行数据做更改
 - 更改timestamp为committed，对外可见

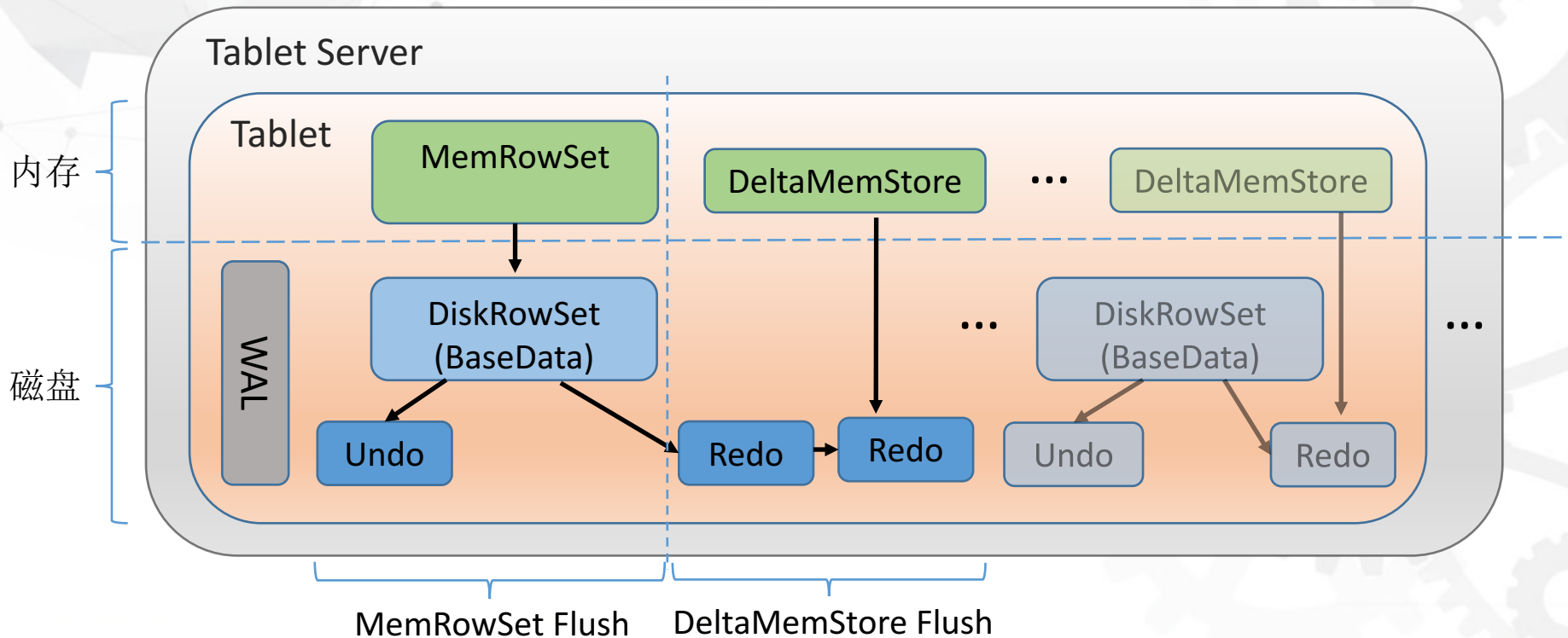
MVCC & 数据存储

Insert:

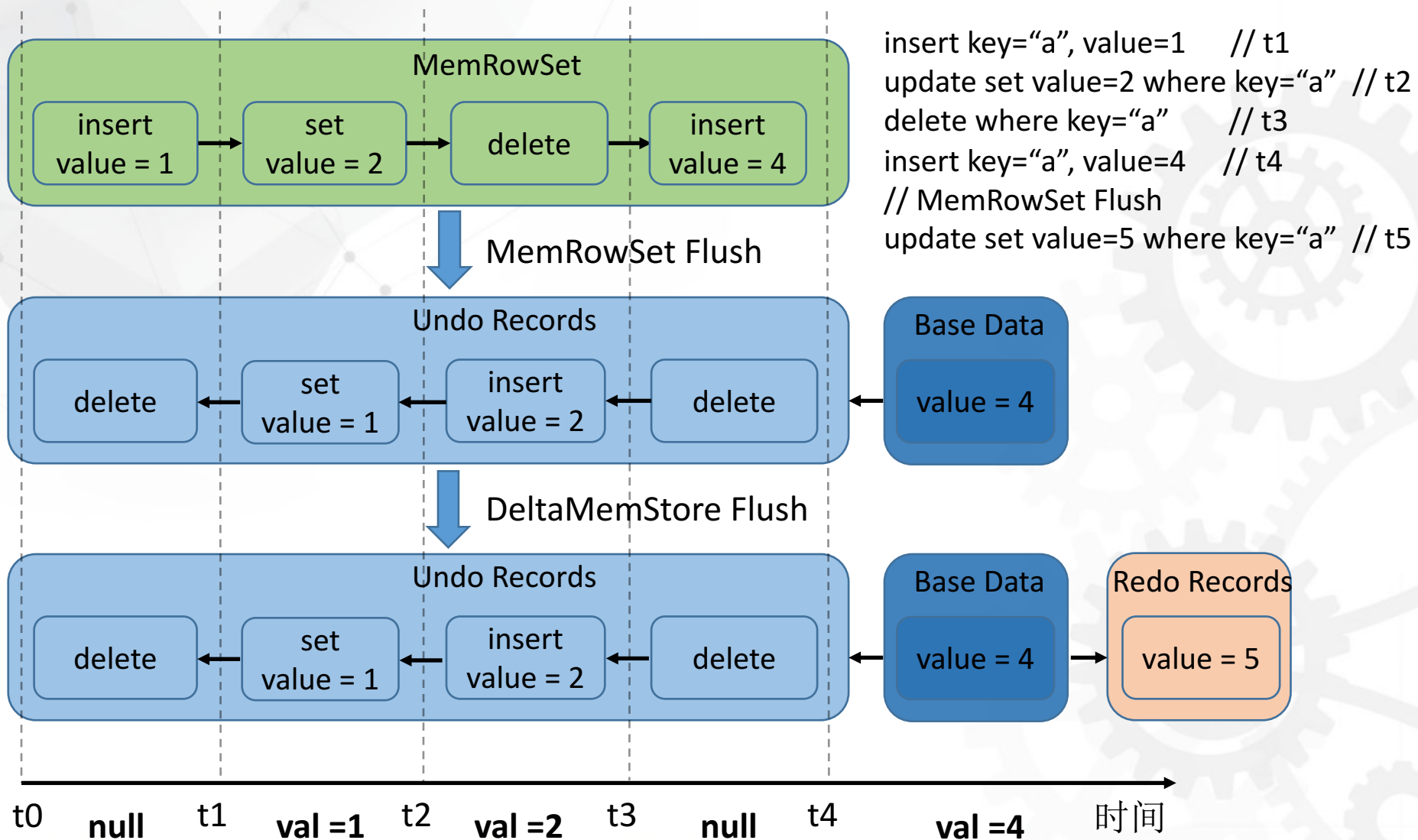
- 所有新插入的数据均存储在MemRowSet中
- 插入前使用interval tree和bloom filter检查是否已存在

Update/Delete:

- 对MemRowSet中的数据更新，Flush后形成Undo Records
- 对DiskRowSet中的数据更新，存储在DeltaMemStore中，Flush后形成Redo Records



Flush 示例



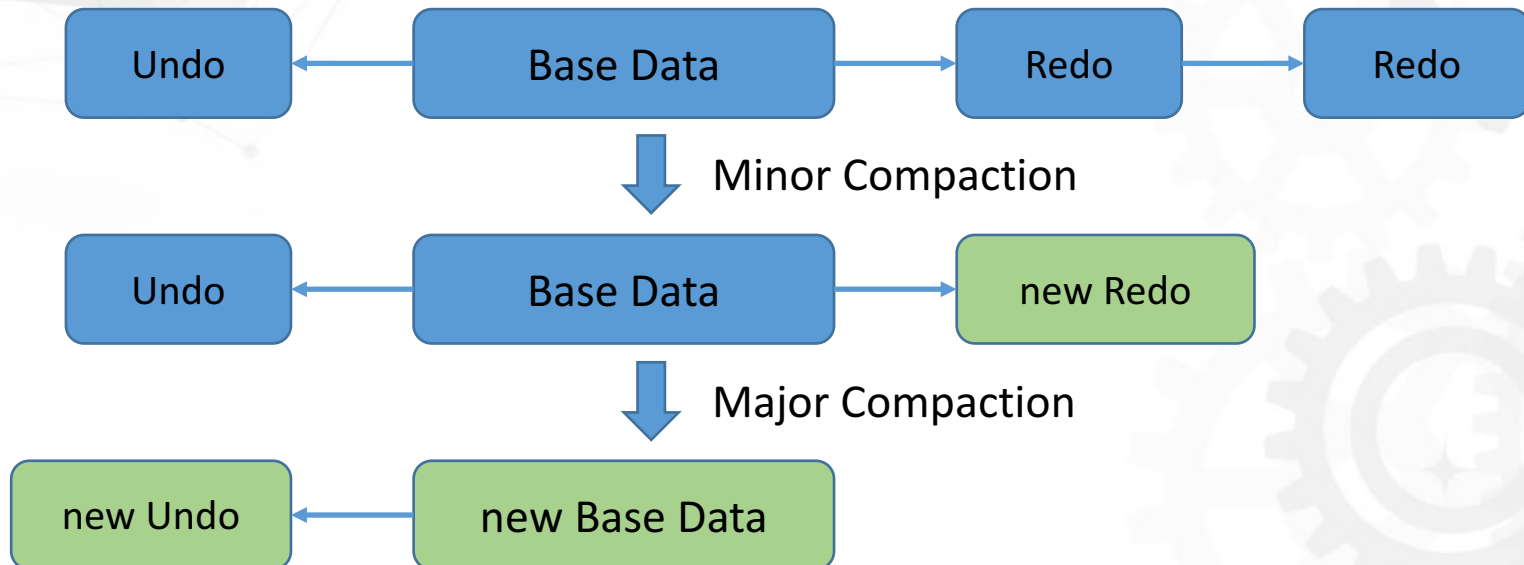
Delta Compaction

Minor Compaction:

- 减少Redo文件的数量，增加读效率

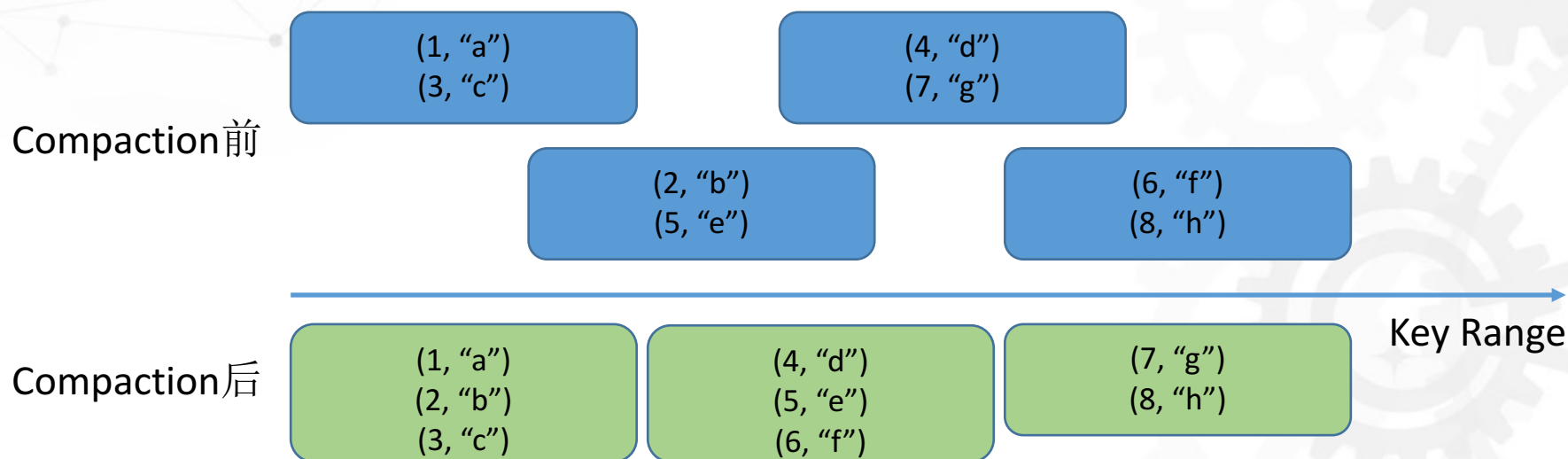
Major Compaction:

- 只读Base Data，节省apply delta的操作



DiskRowSet Compaction

- 减少DiskRowSet间的重叠情况
 - 写操作时减少需要查询的DiskRowSet
 - 读操作读取更少的文件，增加读效率
- 后台线程在空闲时做小规模Compaction
- 在设计表的schema时，需要考虑尽量进行顺序写，以减少后期Compaction的操作。不要使用id做为主键的第一列。



Integration with Spark

Kudu为Spark带来了什么？

- 实时数据的快速分析
- 谓词下推，加速查询
- 基于主键索引的快速查询
- Update和Delete的支持

Spark为Kudu带来了什么？

- 更简单的数据操作方式

NoSQL API

Write

```
KuduTable table = client.openTable("my_table");  
KuduSession session = client.newSession();  
Insert ins = table.newInsert();  
ins.getRow().addString("metric", "load-avg.1sec");  
ins.getRow().addDouble("value", 0.05);  
session.apply(ins);  
session.flush();
```

Read

```
KuduScanner scanner = client.newScannerBuilder(table)  
    .setProjectedColumnNames(Lists.of("value"))  
    .build();  
while (scanner.hasMoreRows()) {  
    RowResultIterator batch = scanner.nextRows();  
    while (batch.hasNext()) {  
        RowResult result = batch.next();  
        System.out.println(result.getDouble("value"));  
    }  
}
```

Spark DataFrame

```
// spark-shell --packages org.apache.kudu:kudu-spark_2.10:1.2.0
// Import kudu datasource
import org.kududb.spark.kudu._
val kuduDataFrame = sqlContext.read.options(
    Map("kudu.master" -> "master1,master2,master3",
        "kudu.table" -> "my_table_name")).kudu

// Then query using Spark data frame API
kuduDataFrame.select("id").filter("id" >= 5).show()
// (prints the selection to the console)

// Or register kuduDataFrame as a table and use Spark SQL
kuduDataFrame.registerTempTable("my_table")
sqlContext.sql("select id from my_table where id >= 5").show()
// (prints the sql results to the console)
```

KuduRDD的实现

- getPartitions
 - 根据下推的predicate获得需要访问的tablet list
 - 每个tablet均包装为一个Partition
- getPreferredLocations
 - 每个tablet的replica locations
- execute
 - 新建KuduScanner进行数据扫描

```
override protected def getPartitions: Array[Partition] = {  
  val builder = kuduContext.syncClient  
    .newScanTokenBuilder(table)  
    .batchSizeBytes(batchSize)  
    .setProjectedColumnNames(projectedCols.toSeq.asJava)  
    .cacheBlocks(true)  
  
  for (predicate <- predicates) {  
    builder.addPredicate(predicate)  
  }  
  
  val tokens = builder.build().asScala
```

SparkSQL on Kudu

目标:

- 和SparkSQL on HDFS提供一致的对外接口（数据库/表/SQL）
- 更好的利用Kudu提供的能力，如update/delete
- 增加内部需要的功能，例如动态分区管理，权限控制等

实现:

- 修改SparkSQL语法
- 增加KuduExternalCatalog，直接使用KuduClient获得表的元数据信息
- 新建Kudu系统表，存储内部功能需要的其他元信息

SparkSQL on Kudu

// 创建表

```
CREATE TABLE kudu_test (  
  year INT WITH (key=true),  
  id INT WITH (key=true),  
  msg STRING )  
PARTITIONED BY  
HASH (id) INTO 10 BUCKETS,  
RANGE (year) RANGE BETWEEN ((2015),(2016));
```

// 增删分区

```
ALTER TABLE kudu_test ADD PARTITION ((2016), (2017));  
ALTER TABLE kudu_test DROP PARTITION ((2015), (2016));
```

// 展示Range分区

```
SHOW PARTITIONS kudu_test;
```

// Update/Delete

```
UPDATE kudu_test SET msg="b" WHERE year=2015 AND id=1;  
DELETE FROM kudu_test WHERE year=2015 AND id=1;
```


Twitter 实时数据

```
0: jdbc:hive2://10.10.10.10:16000/default> SELECT count(*) FROM twitter;
+-----+
| count(1) |
+-----+
| 13472982 |
+-----+
1 row selected (0.396 seconds)
0: jdbc:hive2://10.10.10.10:16000/default> SELECT current_timestamp();
+-----+
| current_timestamp() |
+-----+
| 2017-05-03 23:47:35.131 |
+-----+
1 row selected (0.060 seconds)
0: jdbc:hive2://10.10.10.10:16000/default> SELECT created_at, text, user_name FROM twitter ORDER BY created_ts DESC
LIMIT 1;
+-----+-----+-----+
| created_at | text | user_name |
+-----+-----+-----+
| 2017-05-03 23:47:32 | RT @minghawt: I can't believe the royal princes of Korea are coming back https://t.co/tLVk4ZtGw2 | kongblitch |
+-----+-----+-----+
1 row selected (4.986 seconds)
0: jdbc:hive2://10.10.10.10:16000/default> SELECT user_location, count(*) AS tweets_num FROM twitter WHERE created_
ts>unix_timestamp()-3600 AND instr(text, "trump")!=0 AND instr(user_location, "USA")!=0 GROUP BY user_location ORDER
BY tweets_num DESC LIMIT 5;
+-----+-----+
| user_location | tweets_num |
+-----+-----+
| USA | 34 |
| California, USA | 28 |
| Florida, USA | 24 |
| New Jersey, USA | 22 |
| Texas, USA | 18 |
+-----+-----+
5 rows selected (4.148 seconds)
0: jdbc:hive2://10.10.10.10:16000/default>
```

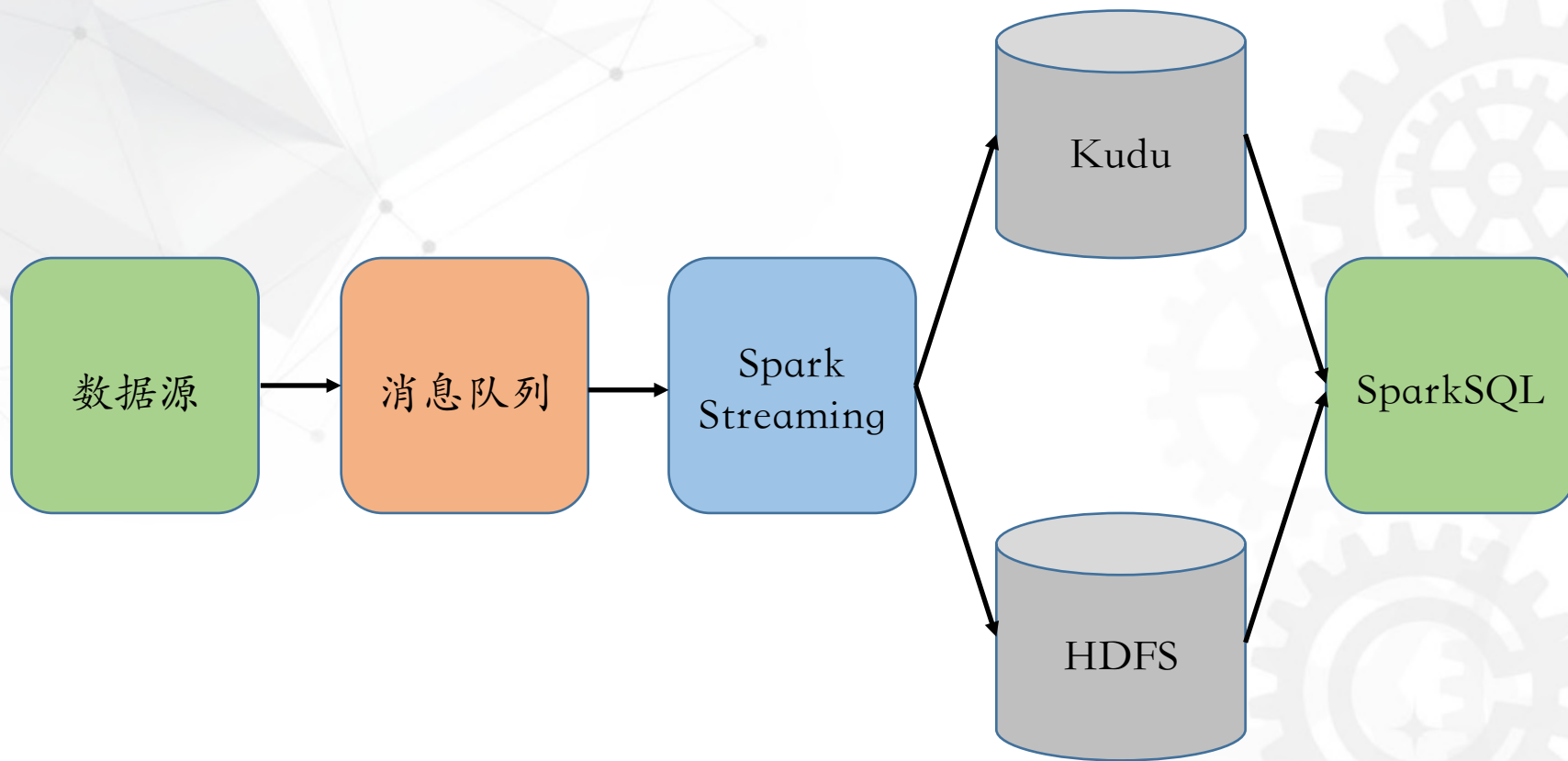
13M数据 →

当前时间 →

最新记录 →

Adhoc查询 →

小米的实时数据仓库



限制 & Roadmap

限制:

- 没有自增主键
- 不支持嵌套结构
- 监控工具不完善
- Load balance

Roadmap:

- 运维上, 增强运维工具的功能 (错误恢复, 问题诊断)
- 性能上, 和查询引擎的深度优化 (runtime filter), 优化批量导入的性能
- 功能上, 完善安全机制, 细粒度的权限管理, 和Sentry的整合

Kudu vs 流式计算

流式计算的一些局限：

- 预计算固定的指标，Ad-hoc查询能力较弱
- 复杂的查询难以支持（大表join）
- 精度要求（TopN，count(distinct)）
- 存在时间窗口
- 容错状态保存，依赖于外部存储，难以开发

Kudu vs Druid

数据精度不同:

- Druid对数据进行预聚合, 不保存原始数据

内部存储不同:

- 均是列存储, 但Druid内部使用bitmap倒排表, Kudu使用bitshuffle等encoding方式

应用场景不同:

- Druid更适合做在线指标的实时统计等工作, Kudu适合实时的分析类查询

查询方式不同:

- Druid有自己的查询引擎, 尚不支持join; Kudu依靠Impala/SparkSQL, 对SQL的支持更加完整

查询性能不同:

- Druid查询的是OLAP Cube, 速度更快; Kudu需要扫描原始数据, 可支持的查询更灵活



THANKS

SequeMedia
威拓传媒

IT168.com

ITPUB

ChinaUnix