QCon+ 案例研习社

如何基于Apache Hudi构建企业级数据湖

T3 出行 / Apache Hudi PMC & Committer / 杨 华

1 • 为什么要构建数据湖

2 • Apache Hudi与数据湖

3 • T3出行基于Hudi构建数据湖的实践

T3出行:全国首家基于车联网的出行平台

数据湖支撑T3智慧出行

人 Driver

车 Vehicle

路 Road

云 Cloud

数据采集

- 背调数据
- 人脸数据
- 交易数据
- 行为数据
- 驾驶数据
- ……

- 车况数据
- 行驶数据
- 能耗数据
- 事故数据
- 故障数据
- ……

- 路况数据
- 环境数据
- 轨迹数据
- POI数据
- 异常数据
- ……

- 风控数据
- 运力数据
- 交易数据
- 城市数据
- 用户数据
- ……

应用场景

- 安全管理
- 司机管理
- UBI保险
- 驾驶模式研究
- ……

- 运力调度
- 主动维修
- 产品改进
- 运营定制车设计
- ……

- 地图绘制
- 实时路况
- 安全管理
- 市政管理
- ……

- 智能调度
- 智能决策
- 智能营销
- 客户体验
- ……

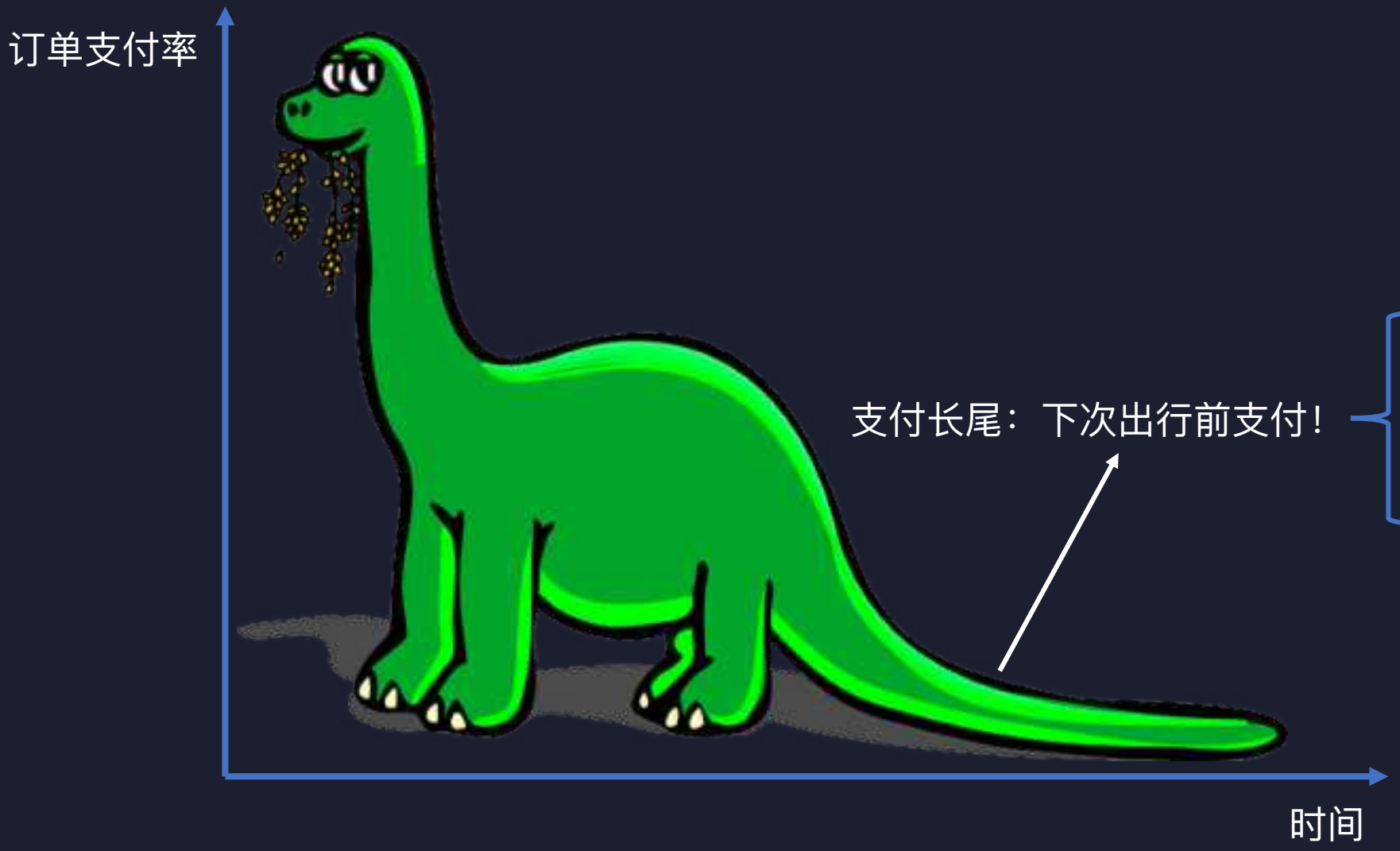
什么是数据湖?

AWS的定义：

A data lake is a centralized repository that allows you to store all your structured and unstructured data at any scale. You can store your data as-is, without having to first structure the data, and run different types of analytics—from dashboards and visualizations to big data processing, real-time analytics, and machine learning to guide better decisions.

数据湖是**一个集中式的存储库**，允许您以任意规模存储所有结构化和非结构化数据。您可以按原样存储数据（无需先对数据进行结构化处理），并运行不同类型的分析 – 从控制面板和可视化到大数据处理、实时分析和机器学习，以指导做出更好的决策。

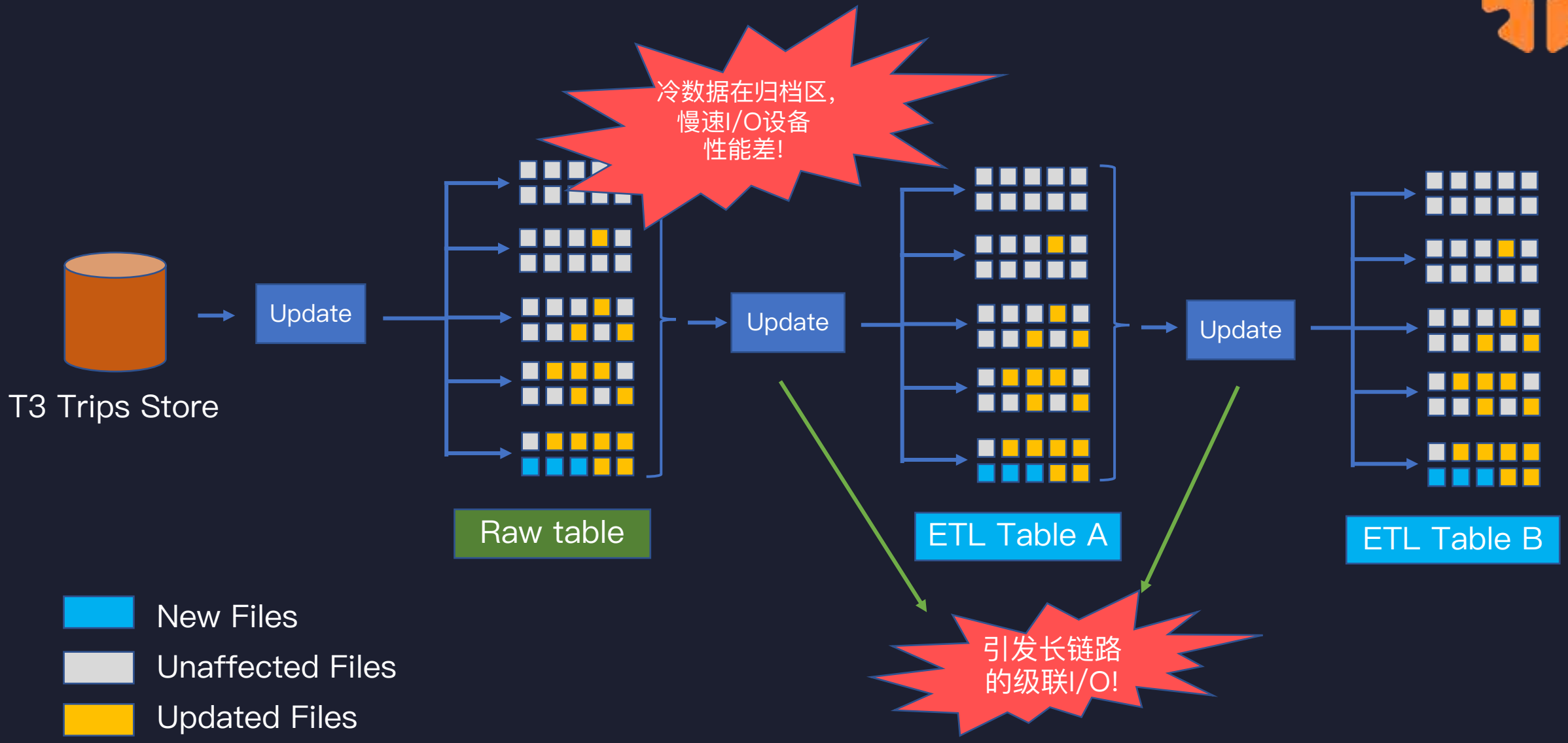
出行业务的“支付长尾”属性



支付长尾：下次出行前支付！

- 超长的业务闭环窗口
- 冷热数据随机更新，无法识别
- 多级更新，链路长，成本高

长尾更新引发冷数据频繁与级联更新



超长的业务窗口导致订单分析回溯成本高

订单　　司机　　车　　乘客　　行程

Order(快照表)

order_id	driver_id	user_id	veh_id	…	status	create_time	lastupdate_time
…	…	…		…	…	…	…
xxx	xxx	xxx	xxx	xxx	end	2020-01-01 xx:xx:xx	
…	…	…		…	…	…	…

半年前的历史快照早已无法访问!

Driver(快照表)

driver_id	

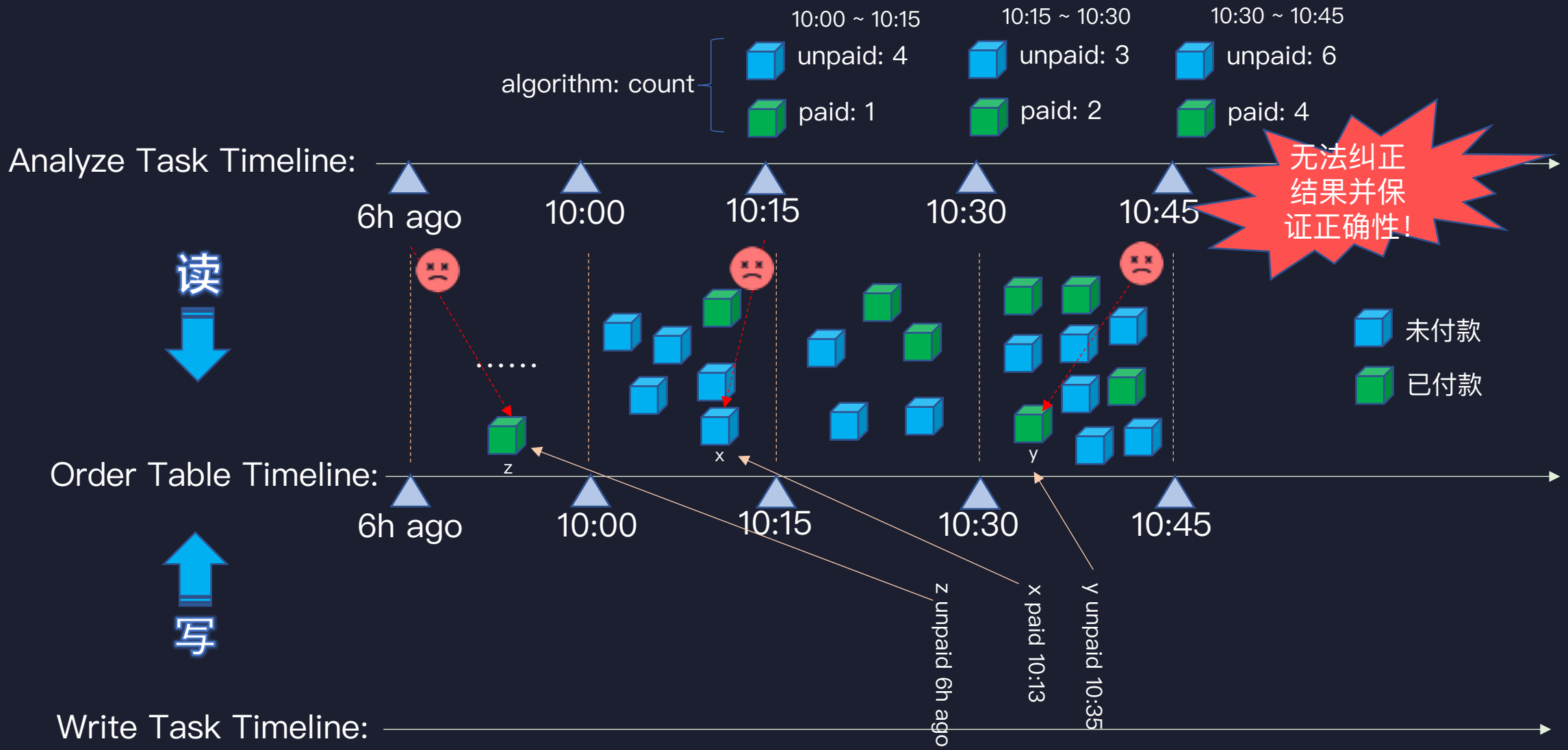
User(快照表)

user_id	

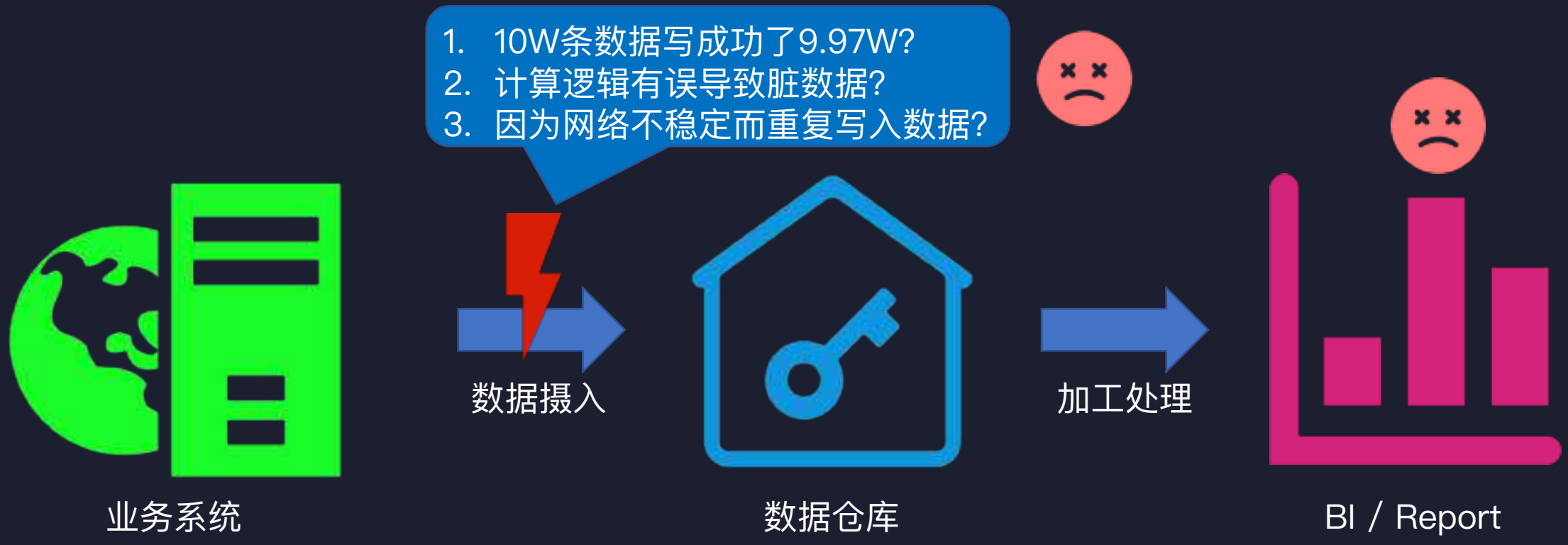
Trip(快照表)

veh_id	

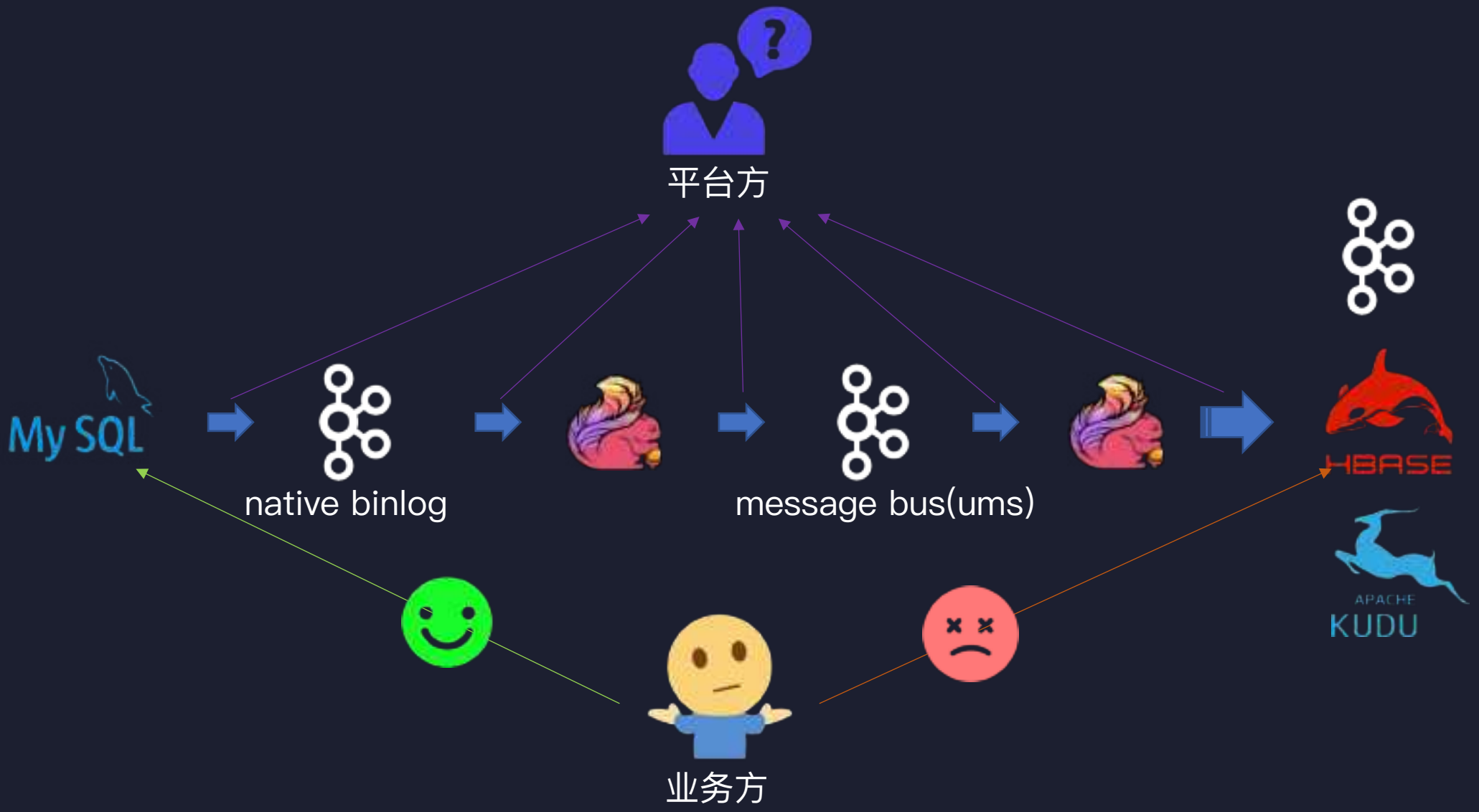
随机更新及迟到数据无法预判



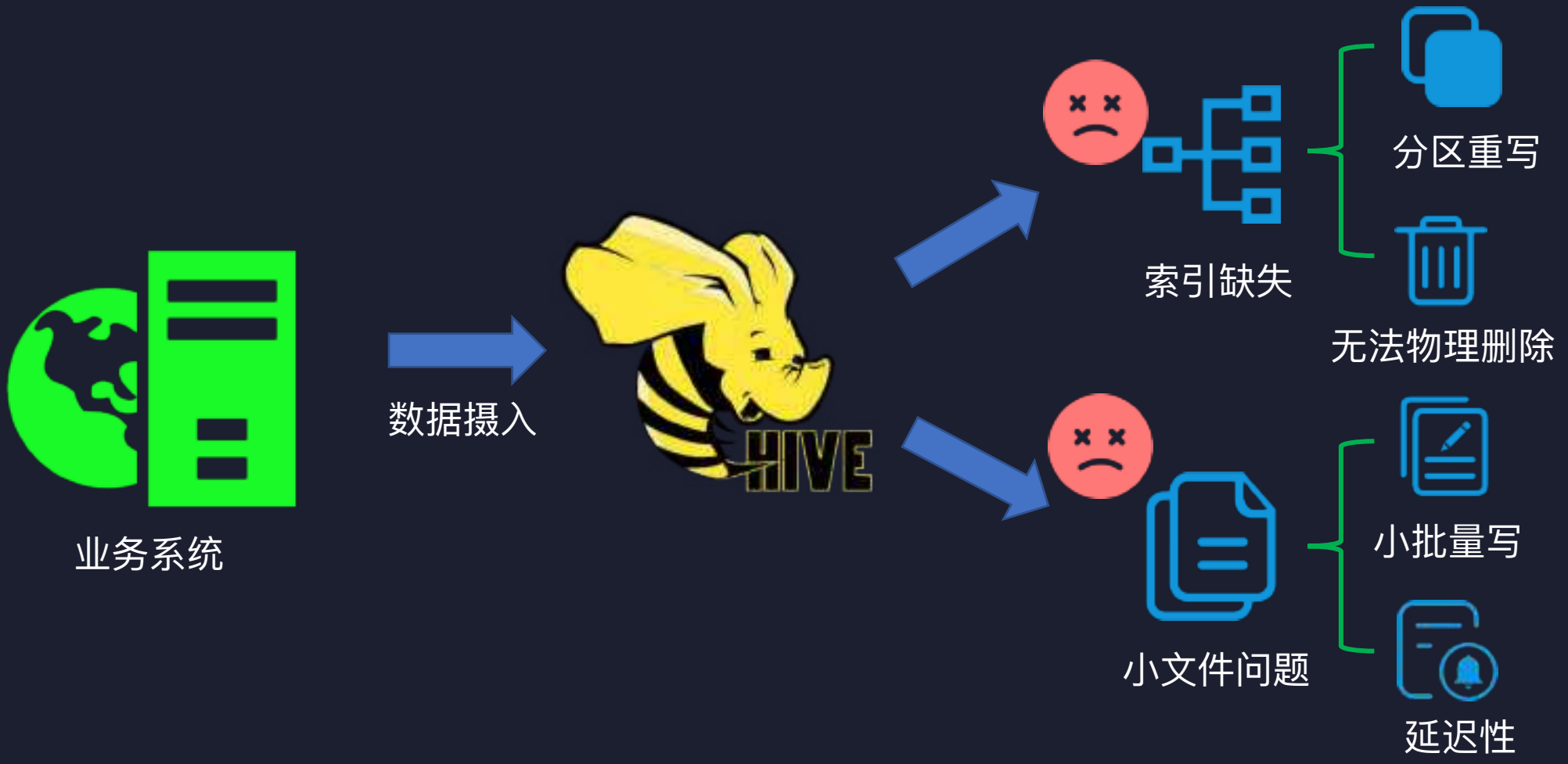
数据摄取Pipeline无法保证可靠性



分布式数据Pipeline中丢数据无法对账



数仓数据摄取的延迟性很高



总结：Hadoop 数仓体系的痛点

可靠性低

高延时

小文件问题

数据版本缺失

不支持增量处理

1 • 为什么要构建数据湖

2 • Apache Hudi与数据湖

3 • T3出行基于Hudi构建数据湖的实践

Apache Hudi框架简介

- Hadoop Upserts Deletes and Incrementals
- 管理DFS/云上超大规模(上百PB)分析数据集
- 支持插入、更新、删除的增量数据湖处理框架
- 2019年1月加入Apache孵化器，2020年5月毕业为TLP
- 对所有云服务(AWS/Tencent Cloud/Aliyun)都开箱即用
- 已在Uber线上稳定运行近4年

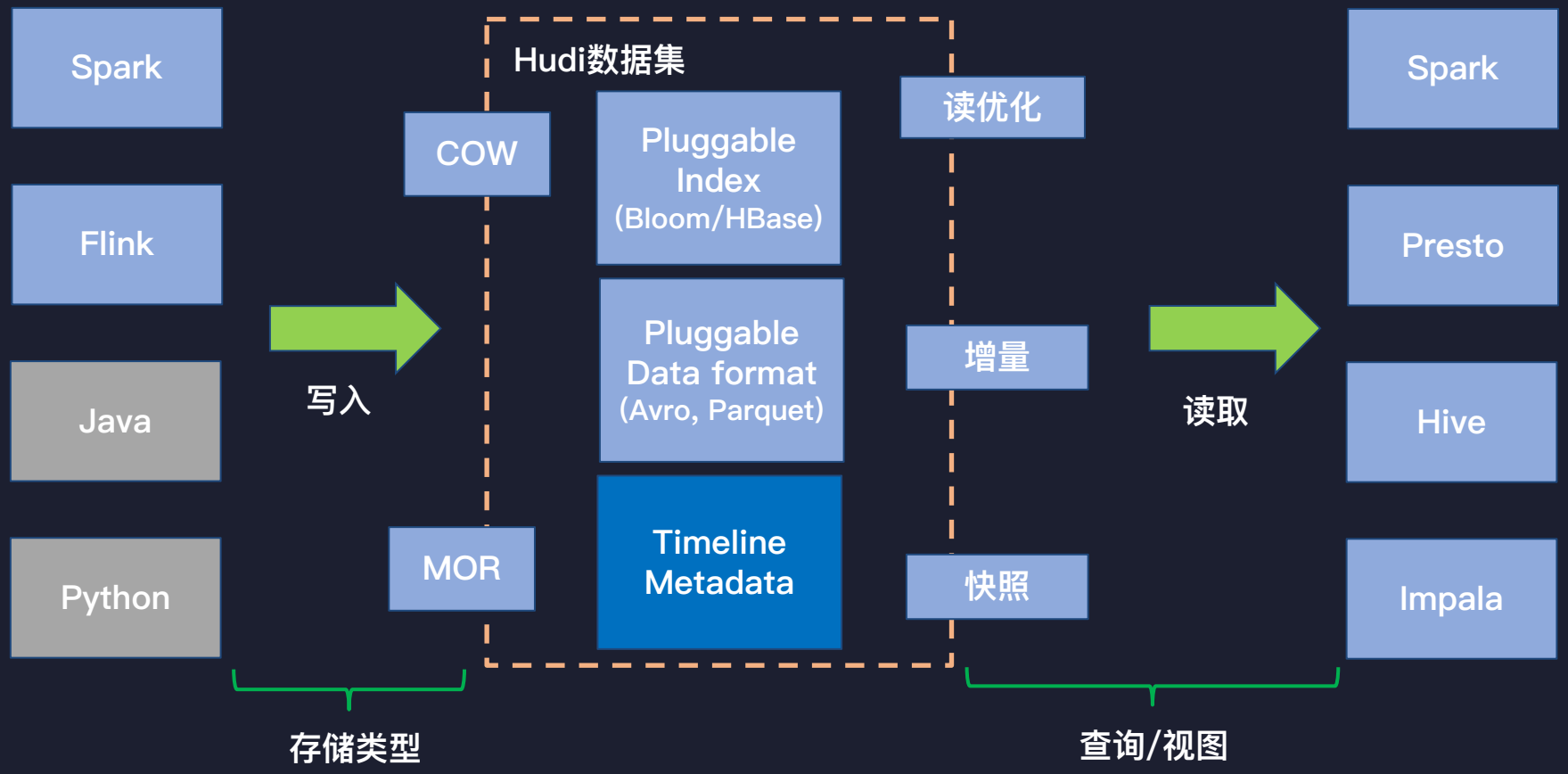
事务性(ACID)

增量处理

存储管理

时间旅行

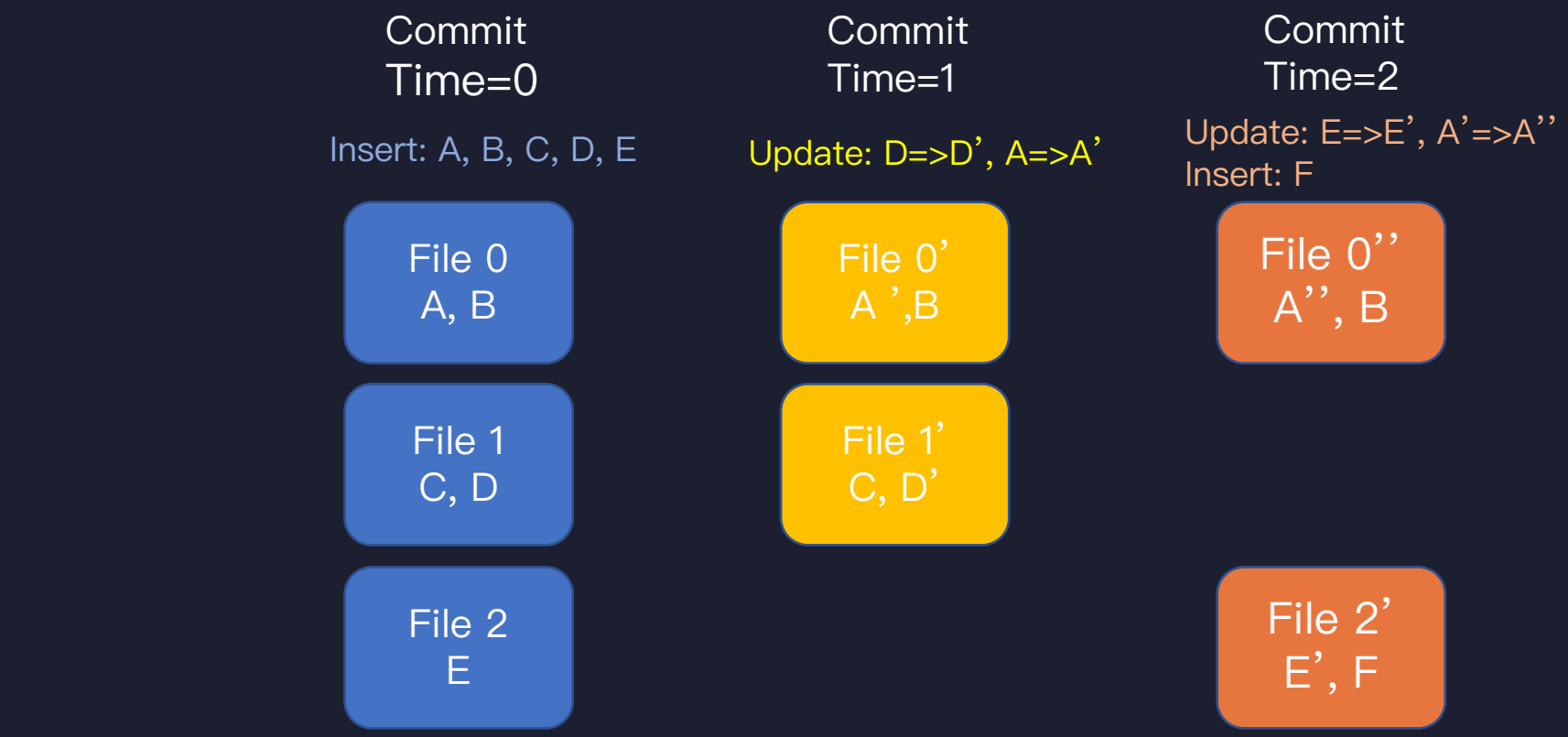
Hudi 插件化的架构


Spark
Flink
Java
Python
写入
Hudi数据集
COW
MOR
Pluggable Index (Bloom/HBase)
Pluggable Data format (Avro, Parquet)
Timeline Metadata
读优化
增量
快照
读取
Spark
Presto
Hive
Impala
存储类型
查询/视图
Pluggable Storage(HDFS, OSS, S3)

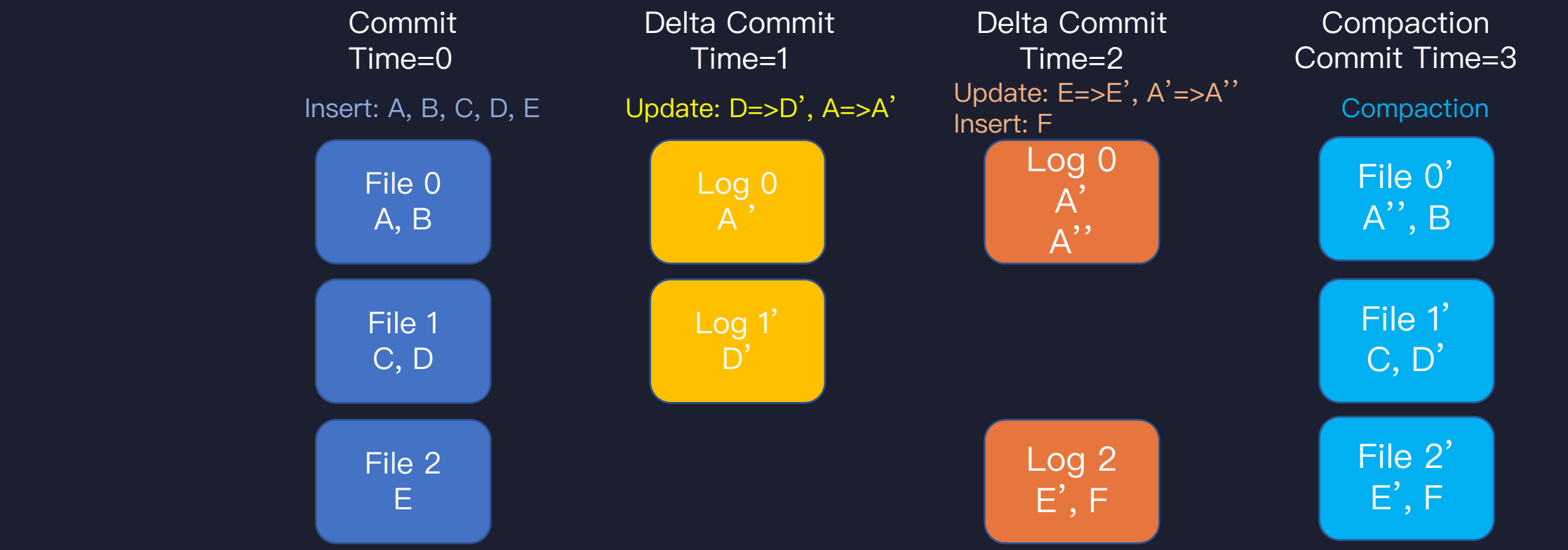
Hudi 存储模式与视图

存储模式	支持的查询	特点	查询引擎	快照查询	增量查询	读优化查询
Copy On Write	快照查询，增量查询	• Read Heavy • 侧重于低延迟的查询 • 列式Parquet数据文件	Hive	Y	Y	–
			Spark SQL	Y	Y	–
			Spark Datasource	Y	Y	–
			Presto	Y	N	–
			Impala	Y	N	–
Merge On Read	读优化查询，增量查询，快照查询	• Write Heavy • 侧重于快速的数据摄入 • 列式Parquet数据文件 • 行式Avro增量文件	Hive	Y	Y	Y
			Spark SQL	Y	Y	Y
			Spark Datasource	N	N	Y
			Presto	N	N	Y
			Impala	N	N	Y

增量查询：Copy On Write

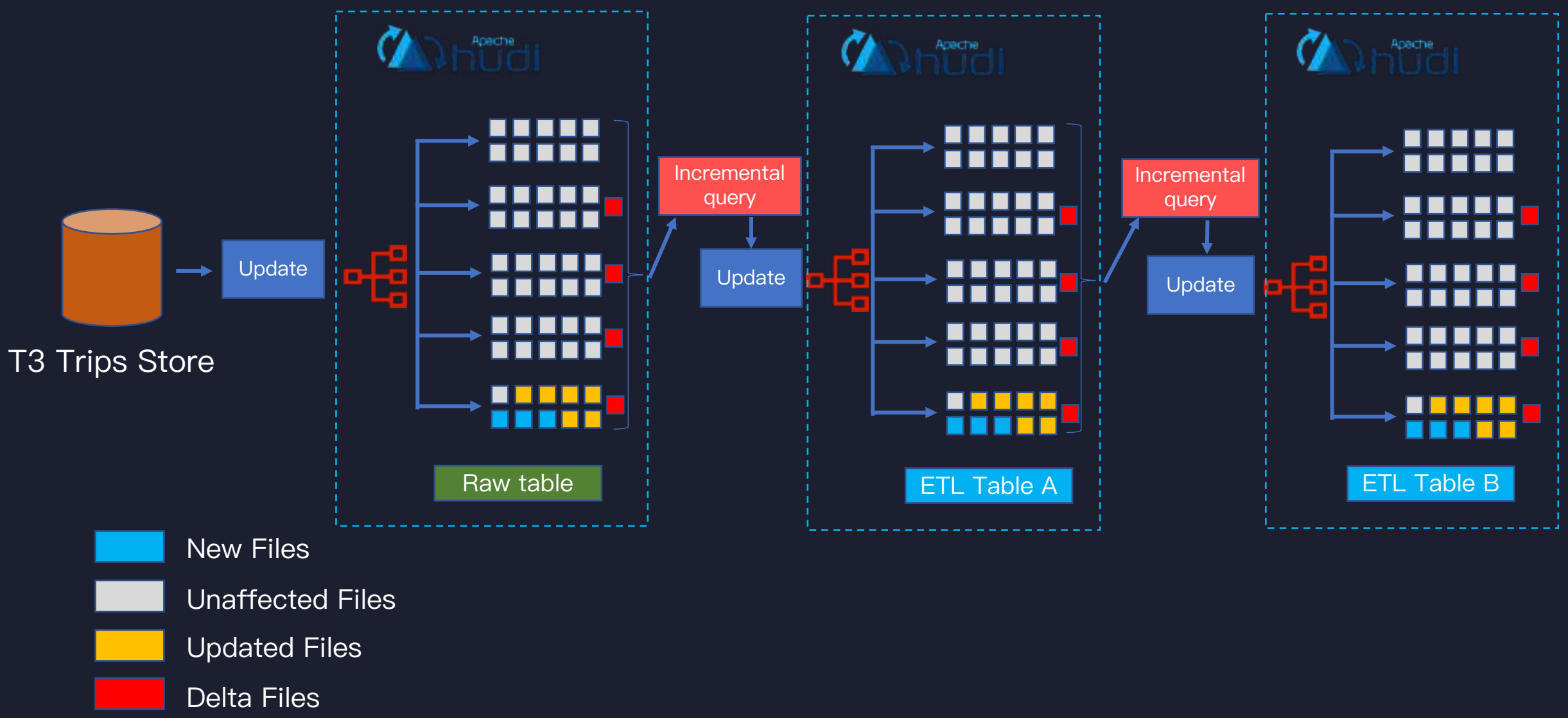


增量查询：Merge On Read



Snapshot :	A,B,C,D,E	A',B,C,D',E	A'',B,C,D',E',F	A'',B,C,D',E',F
Incremental :	A,B,C,D,E	A',D'	A'',E',F	
Read Optimized :	A,B,C,D,E	A,B,C,D,E	A,B,C,D,E	A'',B,C,D',E',F

索引+增量查询解决冷数据频繁与级联更新



时间旅行的查询让“时光倒流”

订单　　司机　　车　　乘客　　行程

Order(快照表)

order_id	driver_id	user_id	veh_id	…	status	create_time	lastupdate_time
…	…	…		…	…	…	…
xxx	xxx	xxx	xxx	xxx	end	2020-01-01 xx:xx:xx	
…	…	…		…	…	…	…

将时间拉回订单发生的时刻！

Driver(v_2020-01-01)

driver_id	

User(v_2020-01-01)

user_id	

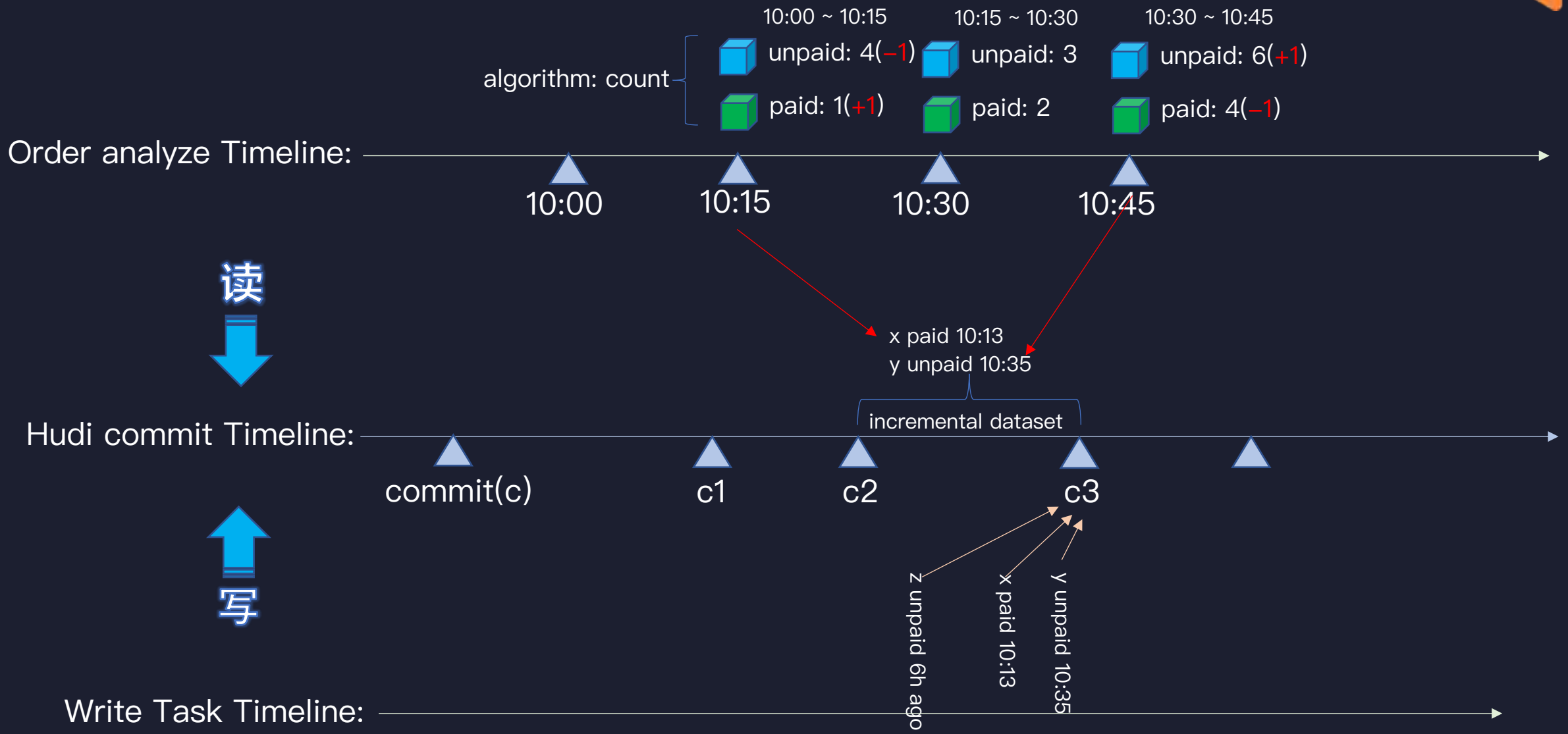
Trip(v_2020-01-01)

veh_id	

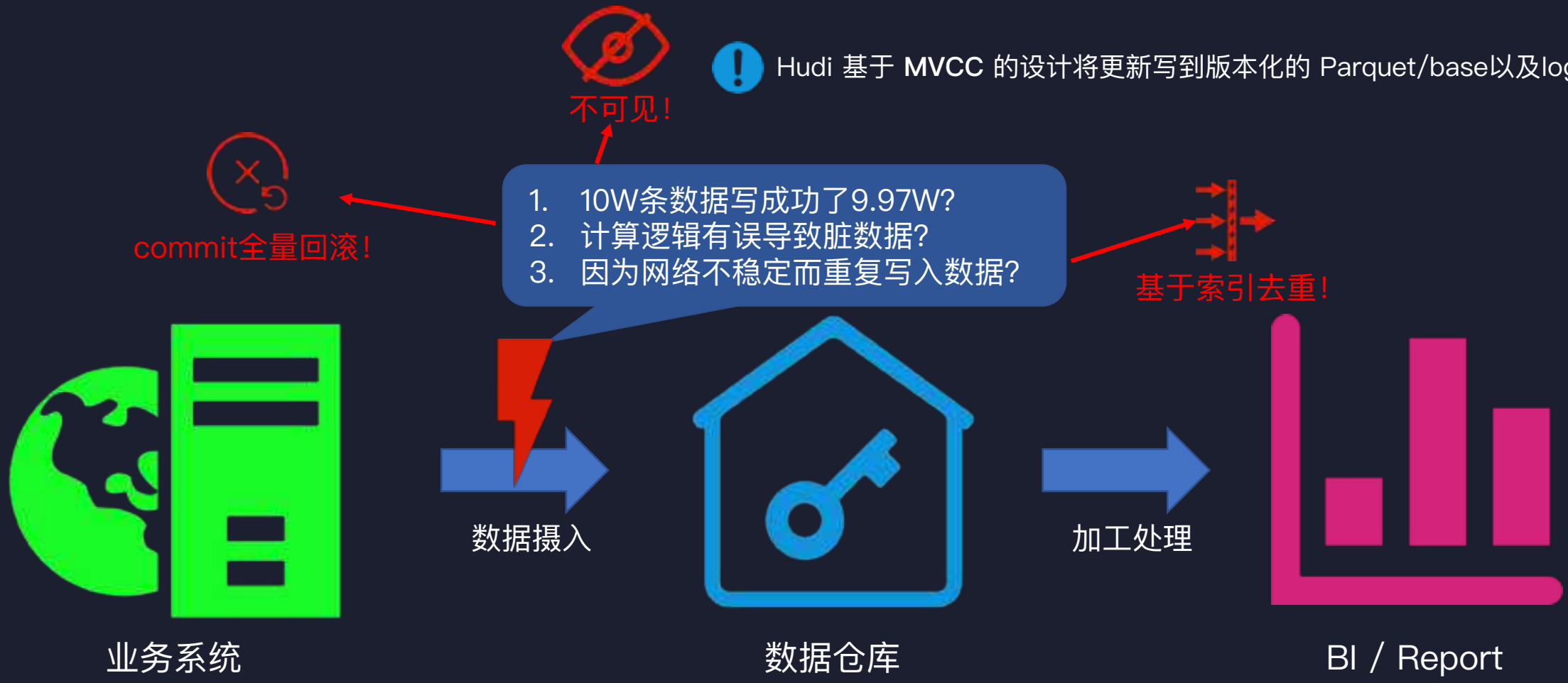
Hudi 特性：

- 时间旅行
- 数据版本

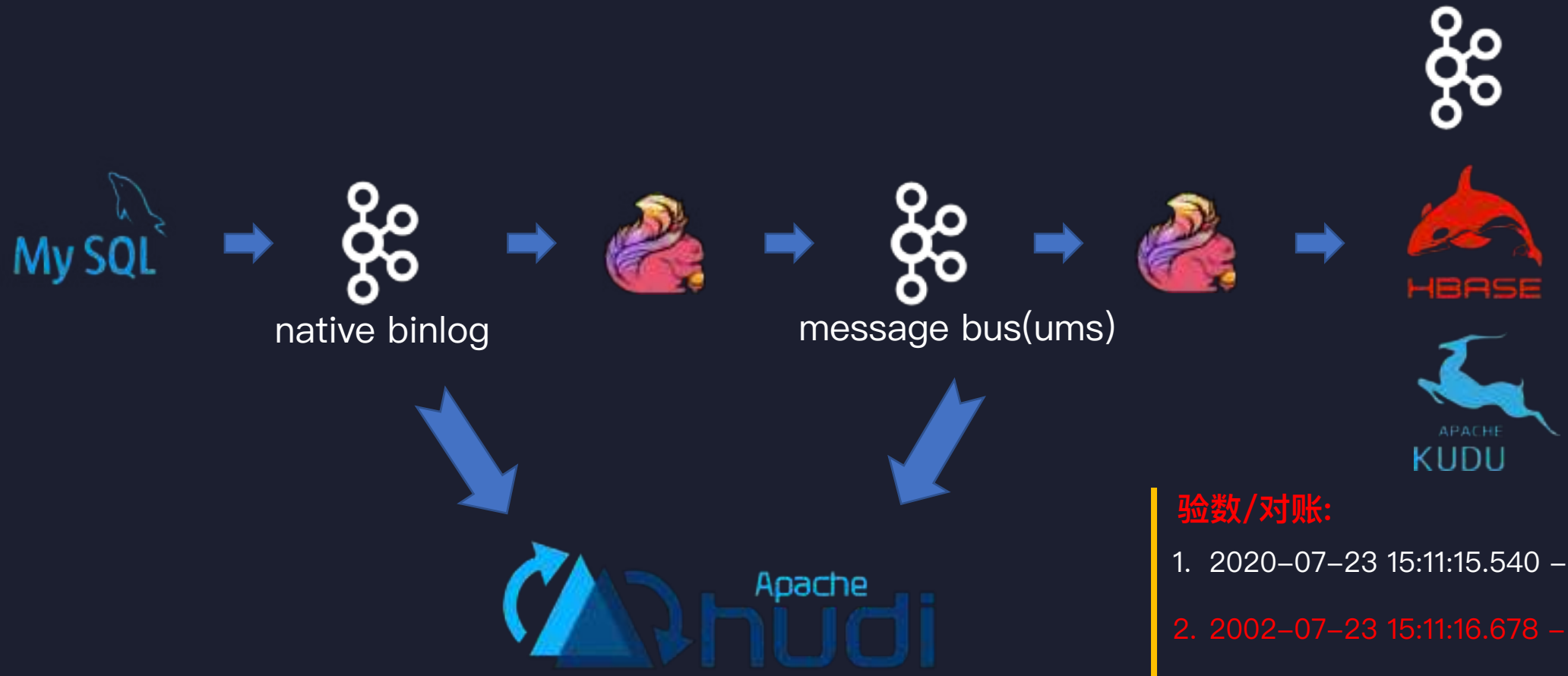
增量查询支持“智能重算”


10:00 ~ 10:15
10:15 ~ 10:30
10:30 ~ 10:45
algorithm: count
unpaid: 4(–1)
unpaid: 3
unpaid: 6(+1)
paid: 1(+1)
paid: 2
paid: 4(–1)
Order analyze Timeline:
10:00
10:15
10:30
10:45
读
x paid 10:13
y unpaid 10:35
incremental dataset
Hudi commit Timeline:
commit(c)
c1
c2
c3
写
z unpaid 6h ago
x paid 10:13
y unpaid 10:35
Write Task Timeline:

Hudi保证数据摄取Pipeline的可靠性



Hudi数据湖支撑实时链路E2E的对账



验数/对账:

1. 2020-07-23 15:11:15.540 – MySQL
2. 2002-07-23 15:11:16.678 – Hudi
3. 2020-07-23 15:11:16.970 – Hudi
4. 2020-07-23 15:11:17.143 – HBase

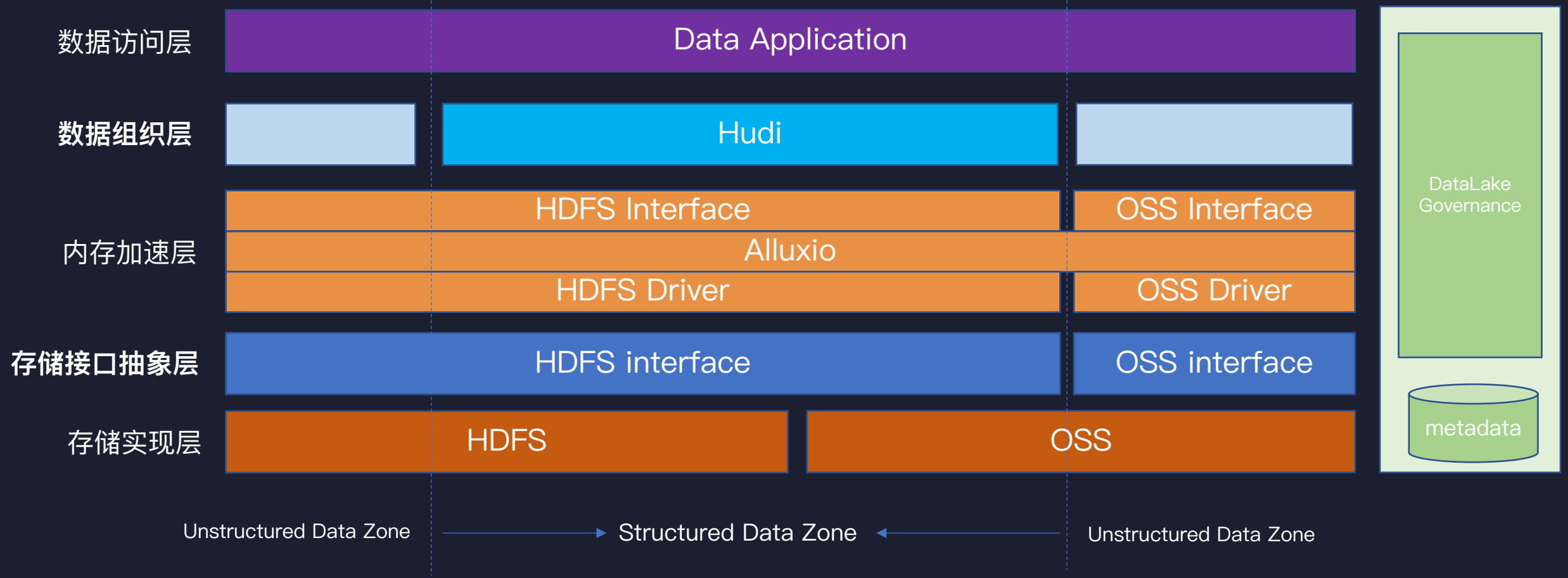
行级索引+文件管理支撑低延迟摄取

1 • 为什么要构建数据湖

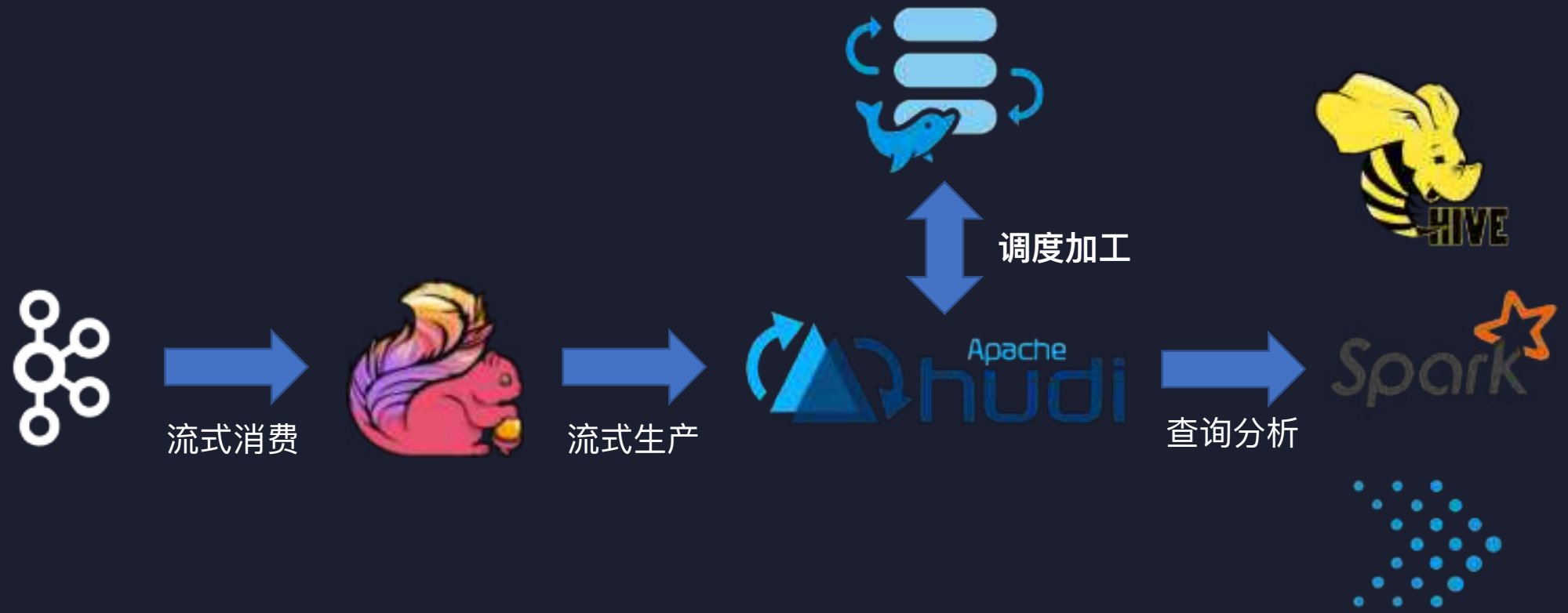
2 • Apache Hudi与数据湖

3 • T3出行基于Hudi构建数据湖的实践

T3出行的数据湖架构


数据访问层
Data Application
数据组织层
Hudi
内存加速层
HDFS Interface
OSS Interface
Alluxio
HDFS Driver
OSS Driver
存储接口抽象层
HDFS interface
OSS interface
存储实现层
HDFS
OSS
DataLake Governance
metadata
Unstructured Data Zone
Structured Data Zone
Unstructured Data Zone

T3出行如何在数据湖上进行准实时分析

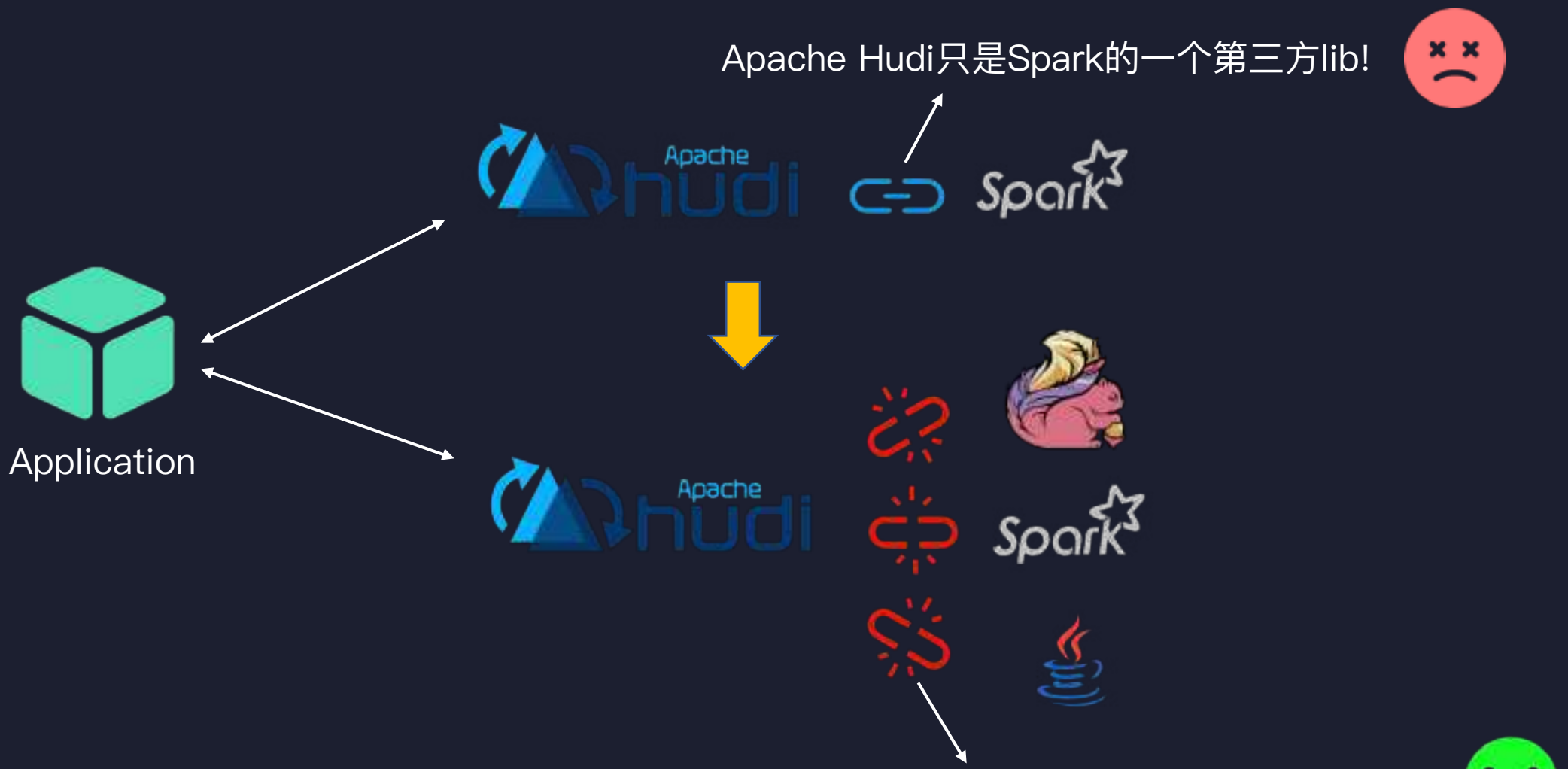


低延迟的数据摄入
- Hudi与Spark解耦
- 支持Flink引擎流式写入

高效快速的数据加工
- 写commit通知
- 调度集成

低延迟的交互式查询分析
- Zeppelin集成
- Alluxio加速

Hudi Write Client与Spark解耦(1/2)



Hudi Write Client与Spark解耦(2/2)

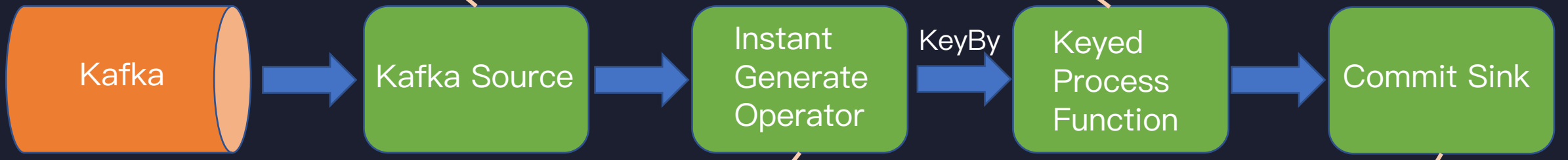
```java
public class HoodieWriteClient<T extends HoodieRecordPayload> extends AbstractHoodieWriteClient<T> {
  public JavaRDD<WriteStatus> upsert(JavaRDD<HoodieRecord<T>> records, final String instantTime) {}
  public JavaRDD<WriteStatus> insert(JavaRDD<HoodieRecord<T>> records, final String instantTime) {}
  public JavaRDD<WriteStatus> delete(JavaRDD<HoodieKey> keys, final String instantTime) {}
}
```

```java
public abstract class BaseHoodieWriteClient<T extends HoodieRecordPayload, I, K, O, P> extends BaseHoodieClient {
  public abstract O upsert(I records, final String instantTime);
  public abstract O insert(I records, final String instantTime);
  public abstract O delete(K keys, final String instantTime);
  public abstract BaseHoodieTable getTableAndInitCtx(WriteOperationType operationType, String instantTime);
  public abstract HoodieIndex<T, I, K, O, P> getIndex();
}
```

设计点解读

- 抽象出引擎无关的write client
- 抽象出引擎无关的hoodie table
- 将所有涉及到API的入参/出参都泛型化
- 引擎特定的table/index对象下推到具体类

Flink Write Client的实现逻辑



Hudi 原生的 DeltaStreamer 是循环+微批!

1、消费Kafka中的数据；
2、周期性检查点维护消费偏移状态；

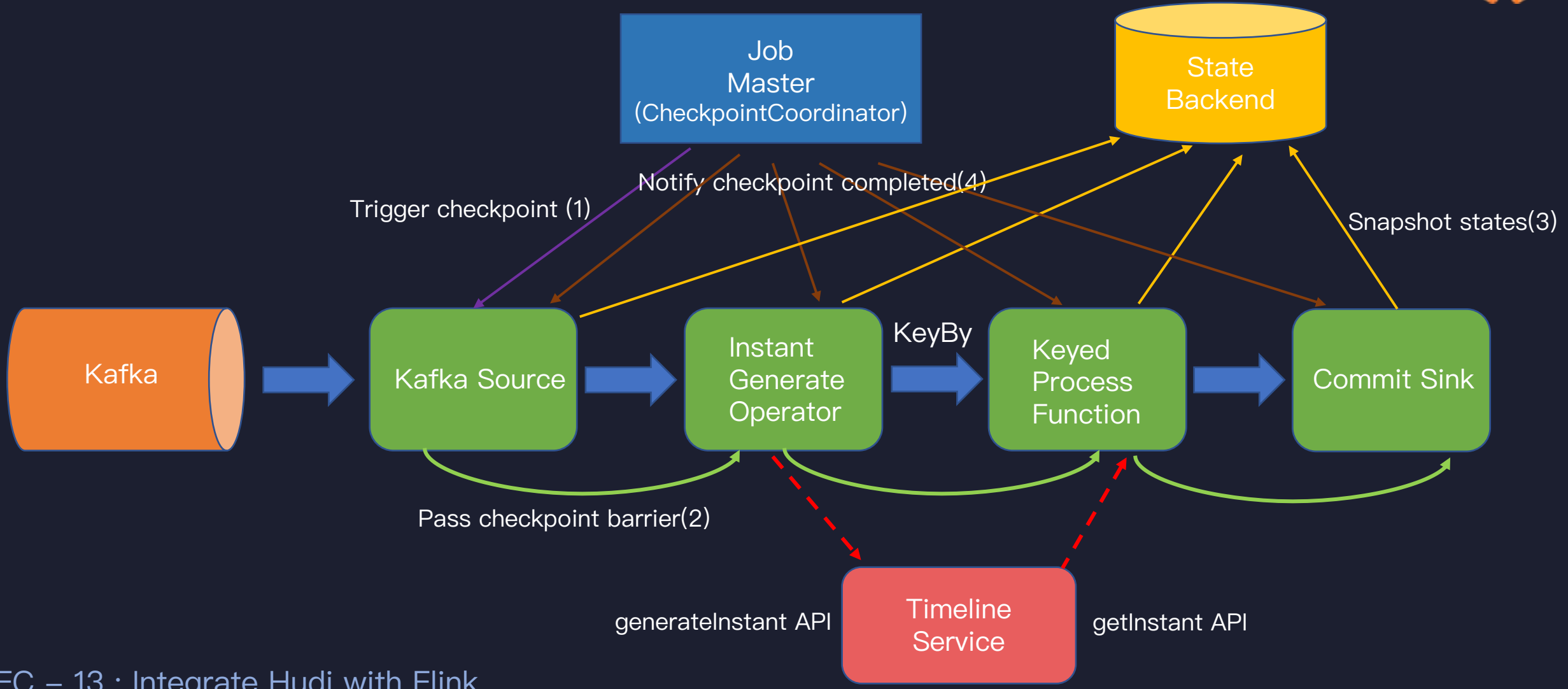
1. 按HoodieKey进行分区；
2. 去重（如需要）
3. tagLocation，标记输入数据位置（如需要）
4. 剖析数据，标记需要insert/update的数据，分桶
5. 修改instant状态为 inflight
6. 对数据分区
7. 执行写操作（lazy）
8. 更新索引

1. 初始化表（如需要）
2. 生成新的instant，置request状态

1、基于检查点完成通知来commit/rollback；
2、 instant状态变更为 completed

RFC – 13 : Integrate Hudi with Flink

Hudi 集成 Flink 实现真正的流式写入



RFC – 13 : Integrate Hudi with Flink

Hudi 与 Alluxio 集成做缓存加速

在压测时，数据量大于一定量级（2400W）后，使用alluxio+oss的查询速度超越了混合部署的HDFS查询速度，数据量大于1E后，查询速度开始成倍提升。到达6E数据后，相对于查询原生oss达到12倍提升，相对于查询原生HDFS达到8倍提升。提升的倍数取决于机器配置。

Alluxio优势：

1. Alluxio 支持层次化且透明的缓存机制；
2. Alluxio 支持读取时缓存 promote 模式；
3. Alluxio 支持异步写模式；
4. Alluxio 支持 LRU 回收策略；
5. Alluxio 拥有 pin 以及 TTL 特性；

T3 基于 Hudi 构建数据湖实践总结

性 能

- Hudi 跟 Flink 框架集成，提供更低延迟的流式数据写入，绑定检查点增强容错；
- Alluxio 加速 Hudi on OSS，性能平均提升近10倍；

功 能

- Hudi 跟 Spark 框架解耦，为业务端使用Java/Python访问数据湖提供基础；
- Hudi 跟 DolphinScheduler 调度框架集成，支撑更高效的增量处理能力；
- Hudi Write Commit集成Kafka，实现异步消息通知机制；
- Hudi 集成 Zeppelin，释放Ad-Hoc的查询、分析能力；

进行中

- Hudi 跟 Kylin 集成释放数据湖上 OLAP 的能力；
- Hudi 存取非结构化数据；

THANKS

QCon+ 案例研习社