Jiangjie (Becket) Qin
LinkedIn

Mission Critical Messaging with Apache Kafka

InfoQ

促进软件开发领域知识与创新的传播

关注InfoQ官方信息

及时获取QCon软件开发者大会演讲视频信息

ArchSummit 全球架构师峰会 2016

[北京站] 2016年12月2日-3日

咨询热线：010-89880682

QCon 全球软件开发大会

[北京站] 2017年4月16日-18日

咨询热线：010-64738142

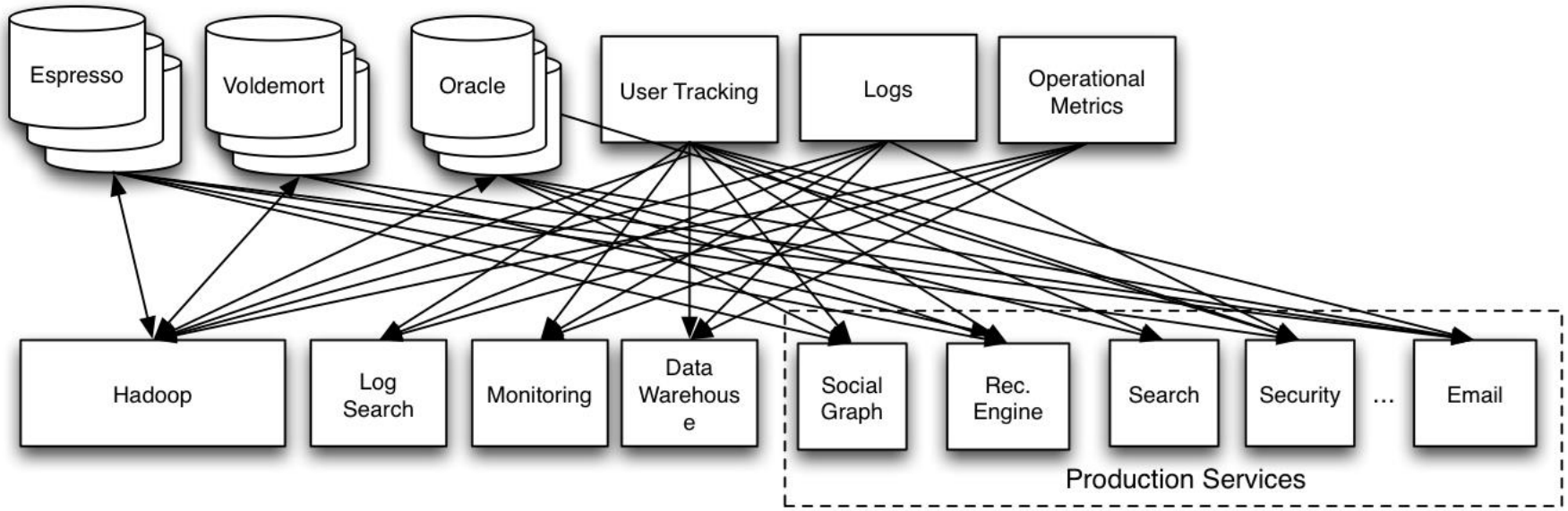
Agenda

- Introduction to Apache Kafka
- Kafka based replication in Espresso
- Message Integrity guarantees
- Performance
- Large message handling
- Security
- Q&A

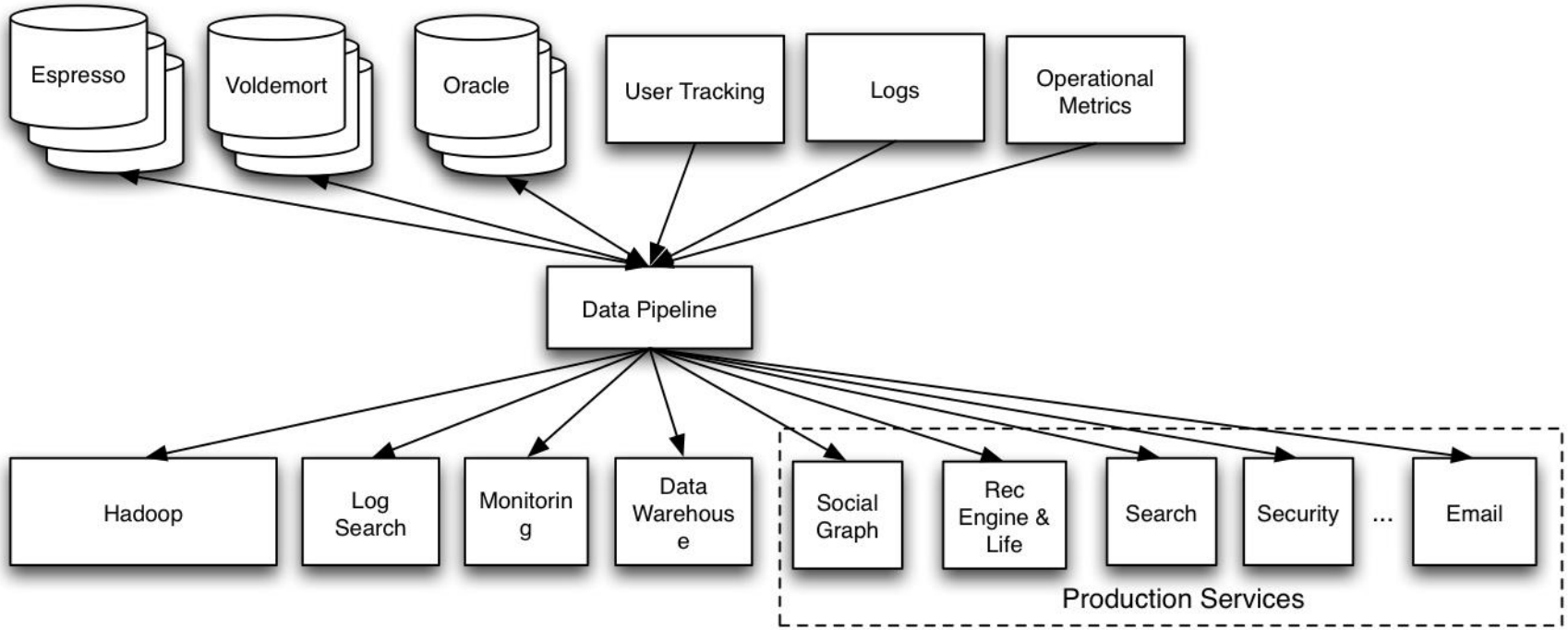
Agenda

- Introduction to Apache Kafka
- Kafka based replication in Espresso
- Message Integrity guarantees
- Performance
- Large message handling
- Security
- Q&A

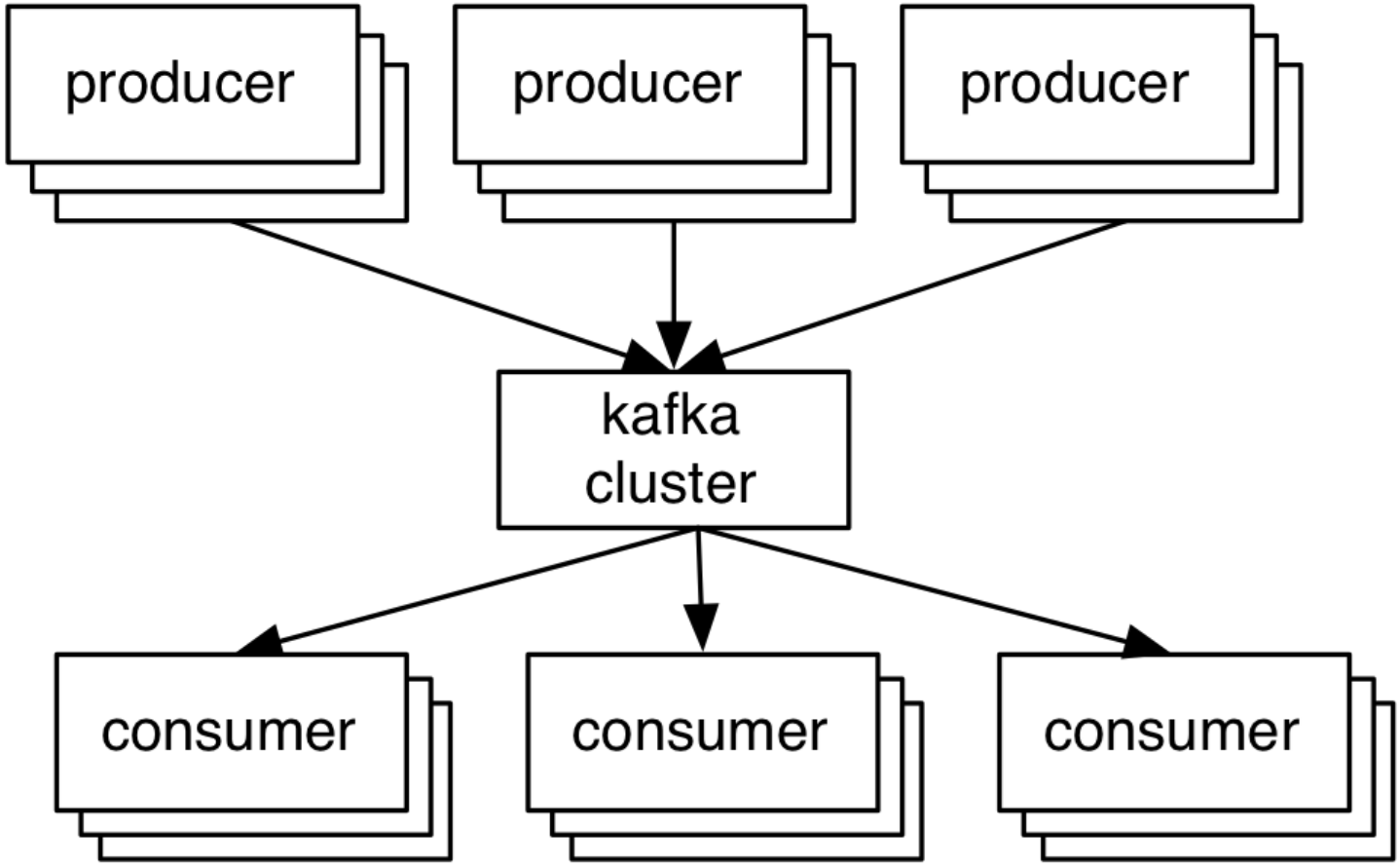
One way to aggregate data



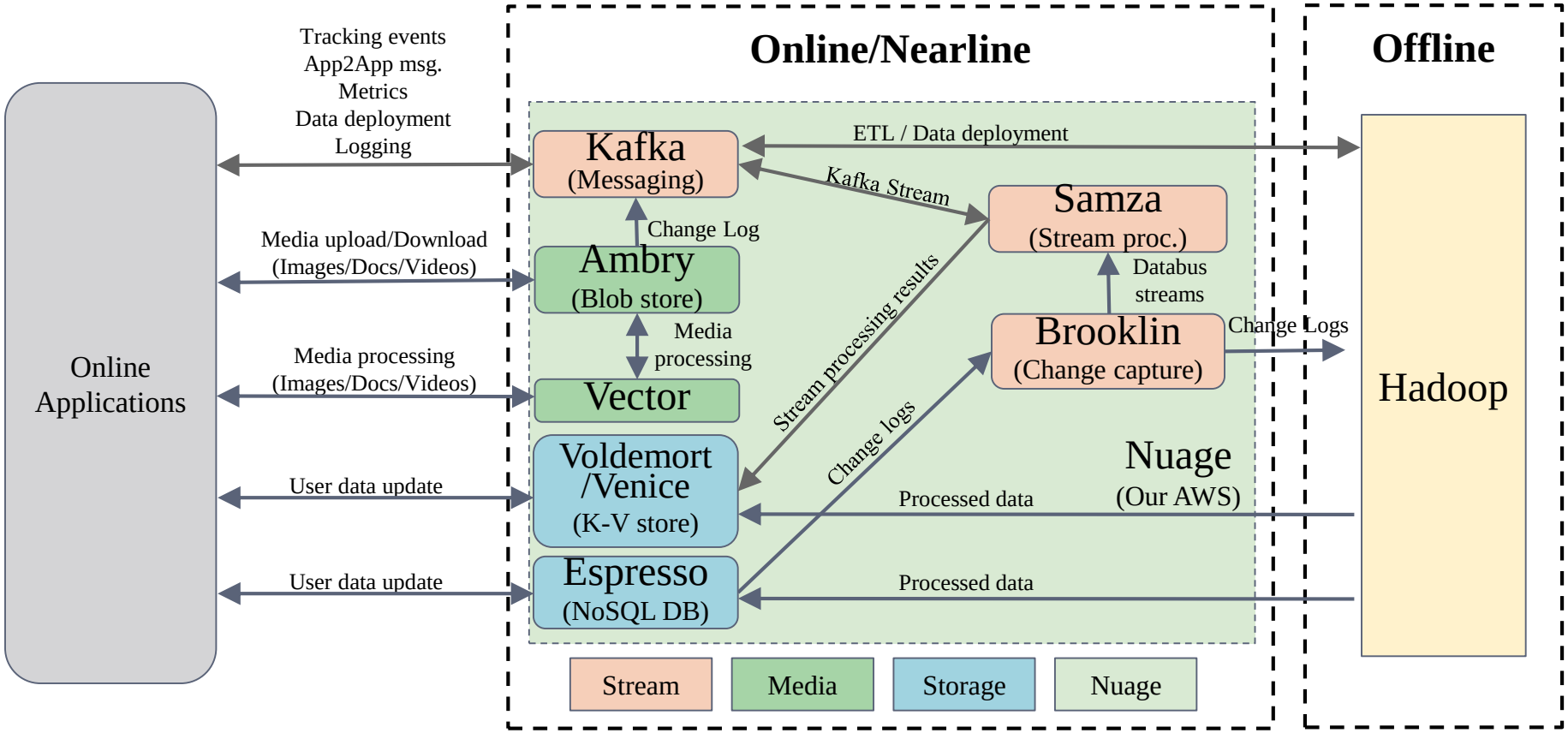
Another way to aggregate data



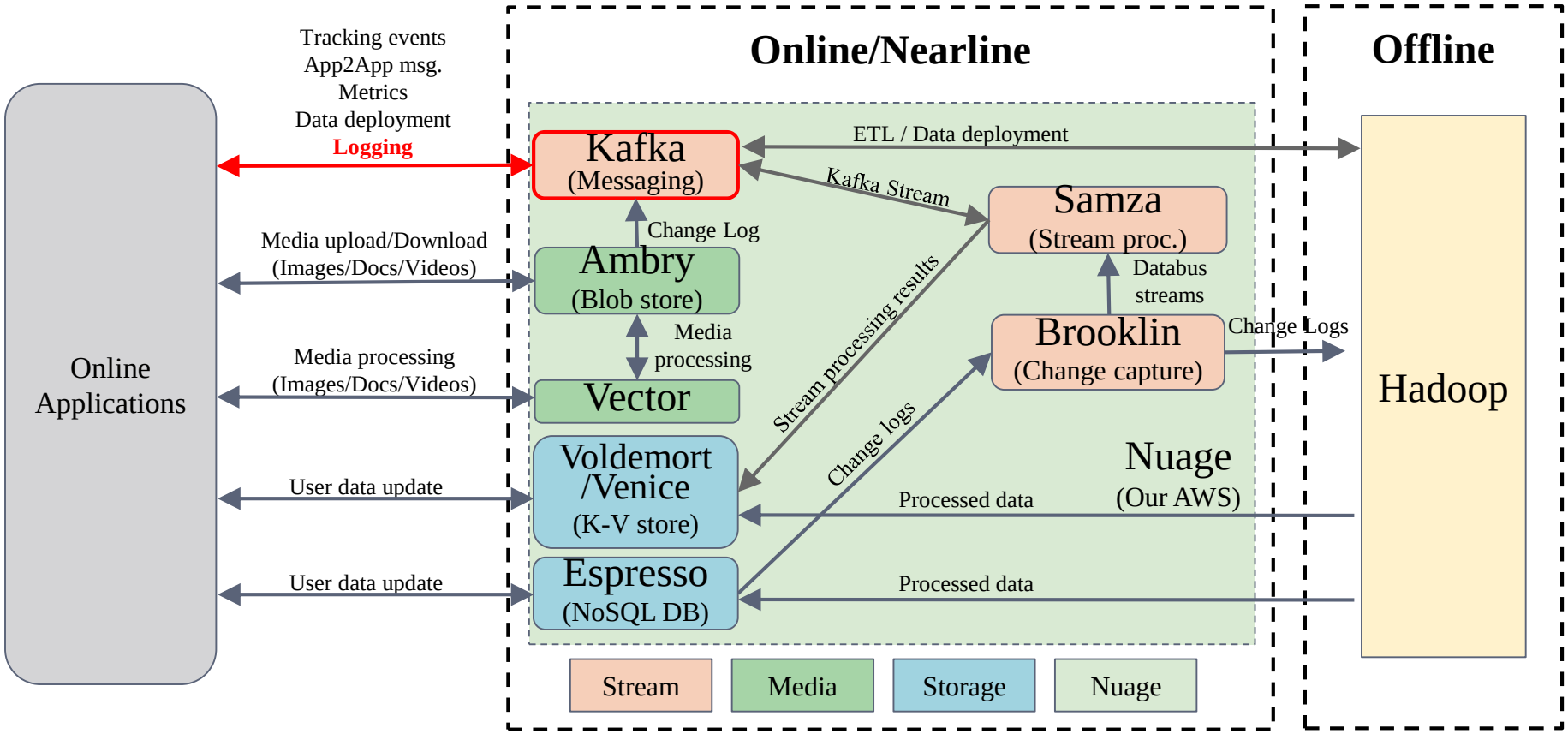
Apache Kafka



The big picture - Kafka@LinkedIn

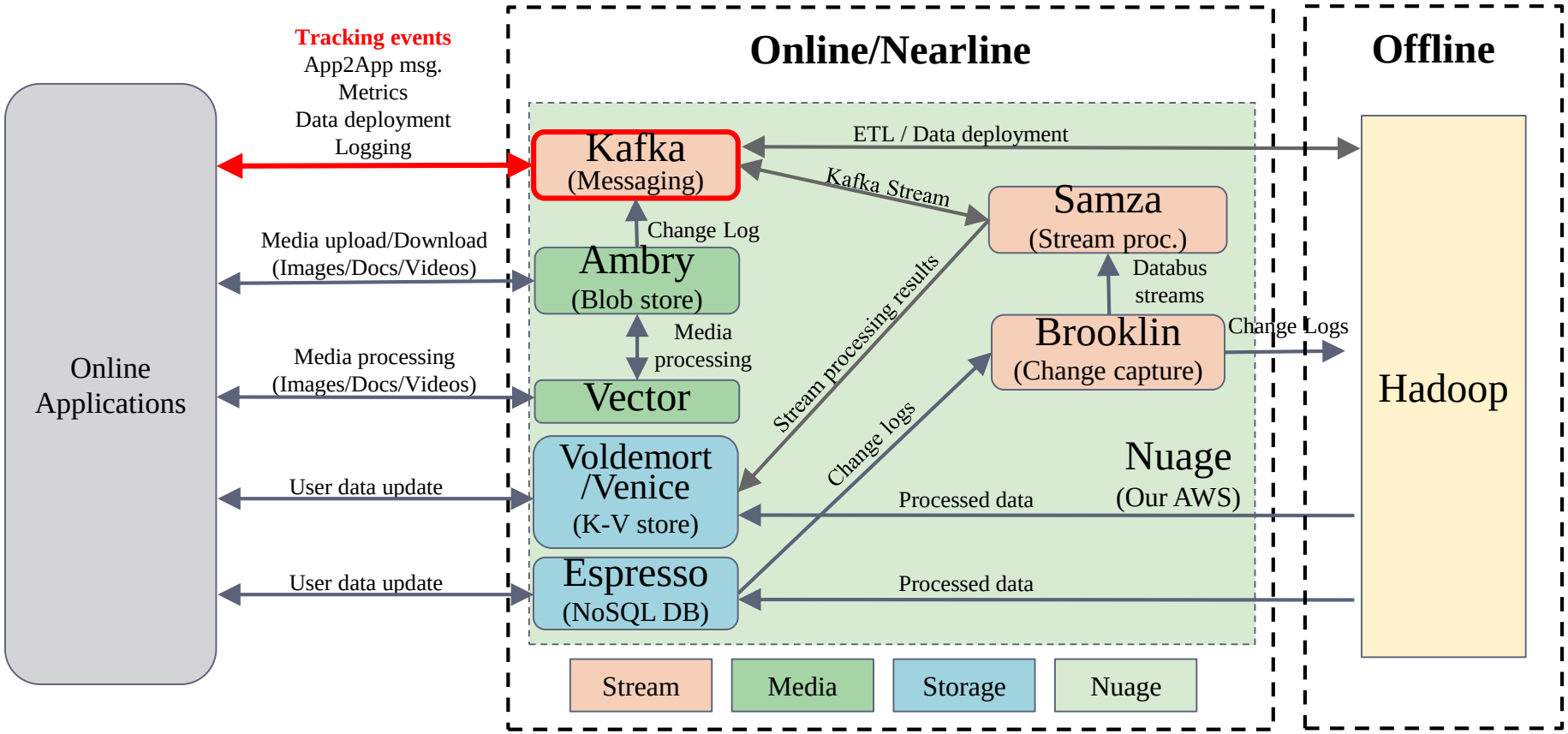


The big picture - Kafka@LinkedIn

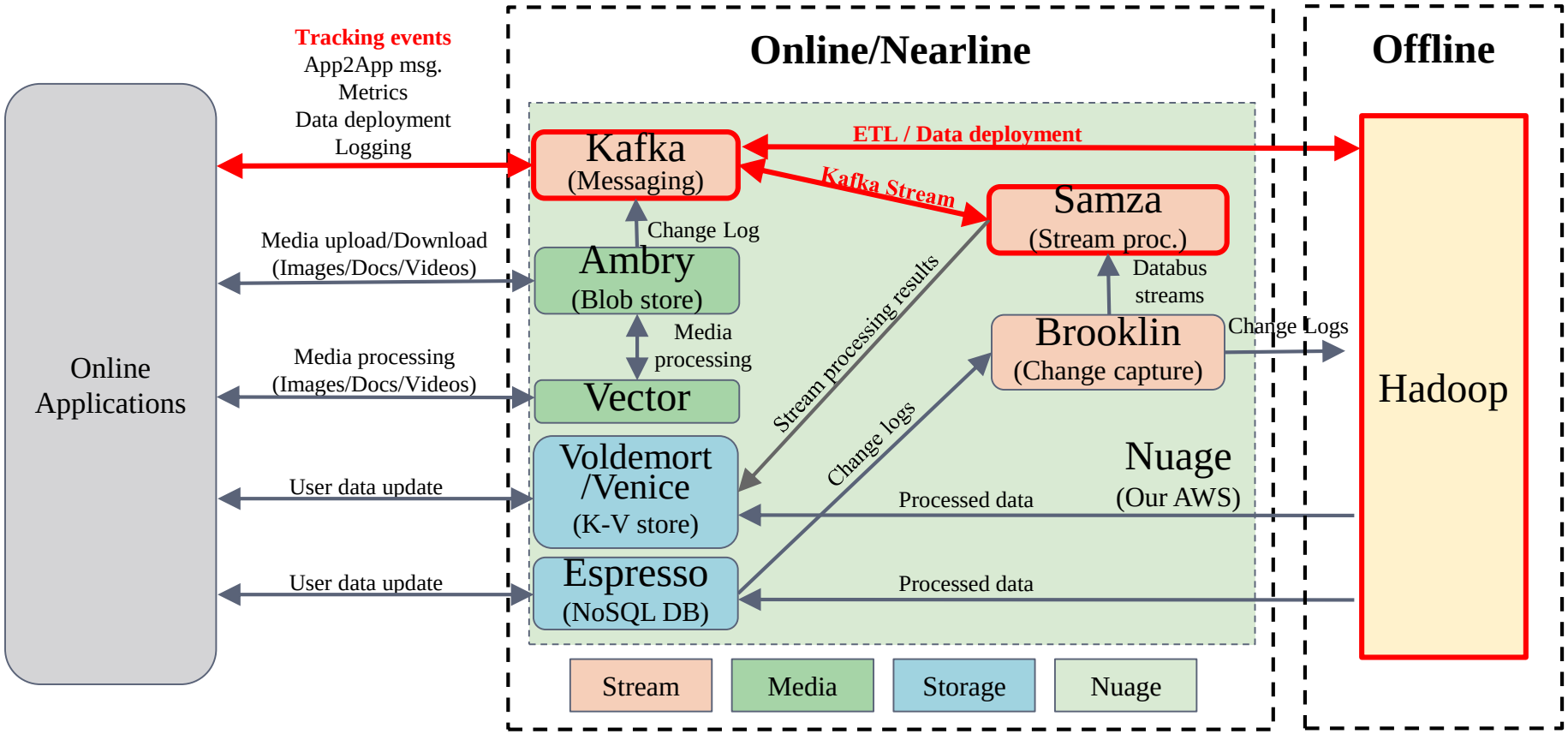


下午16:40，百宴厅1，LinkedIn基于Kafka和ElasticSearch的实时日志分析

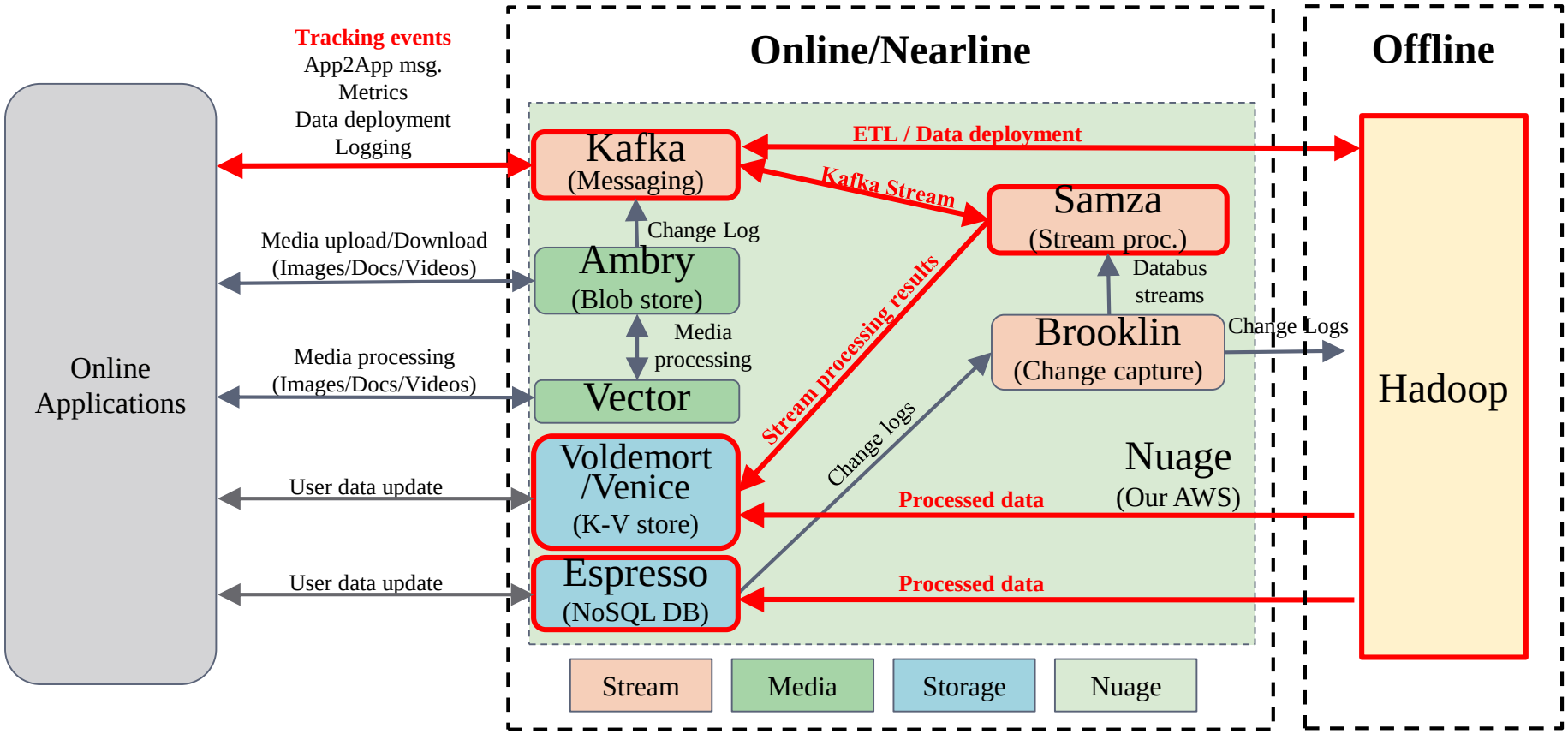
The big picture - Kafka@LinkedIn



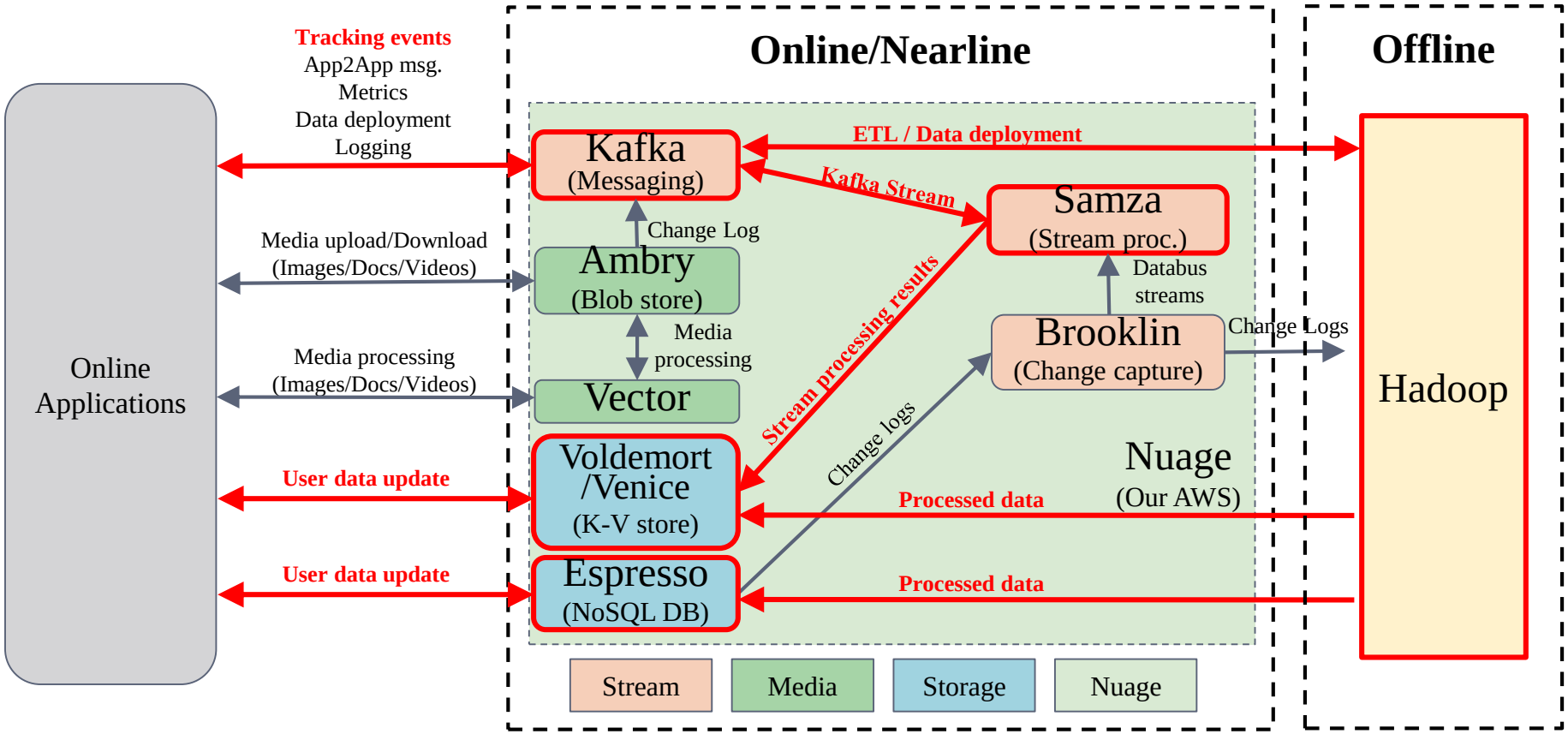
The big picture - Kafka@LinkedIn



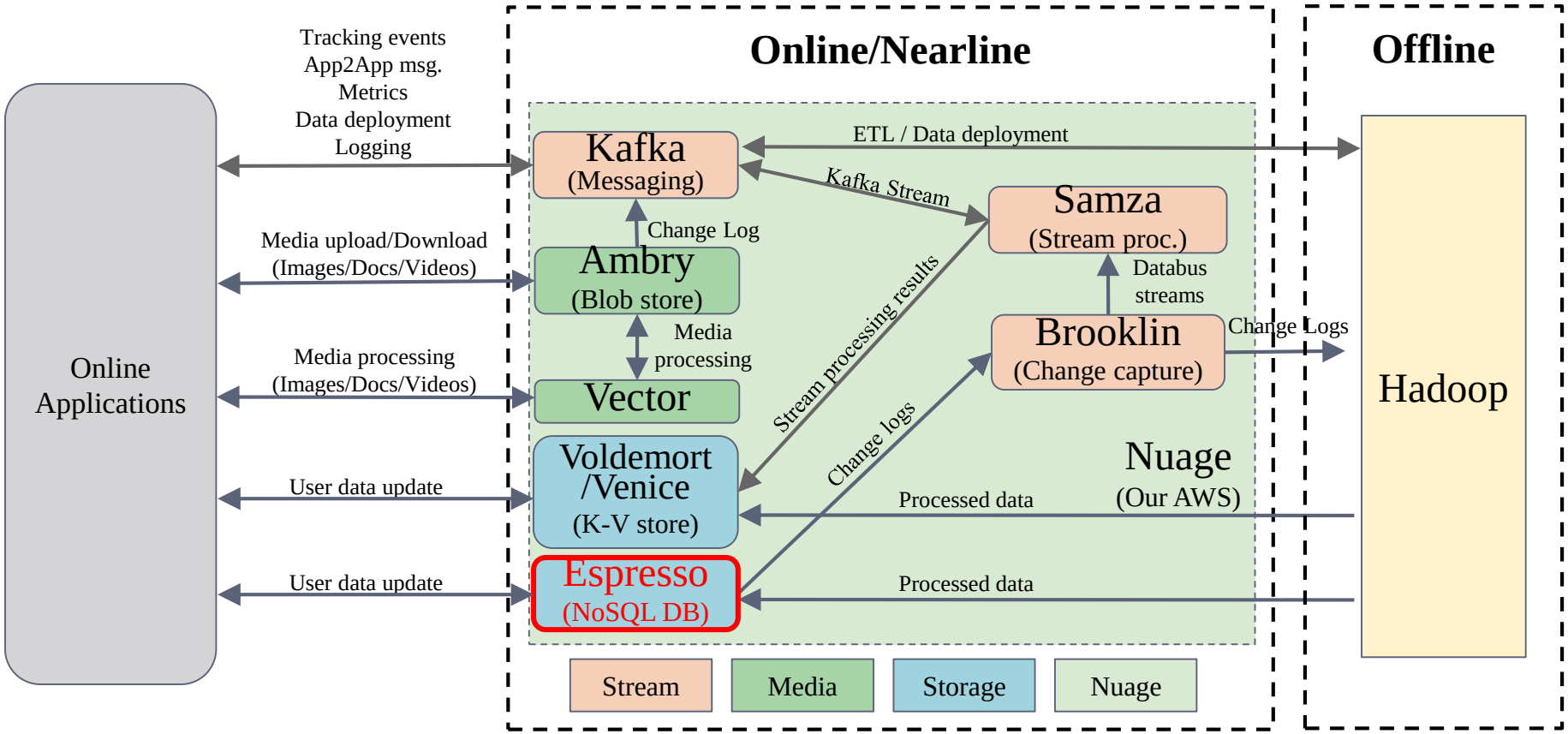
The big picture - Kafka@LinkedIn



The big picture - Kafka@LinkedIn



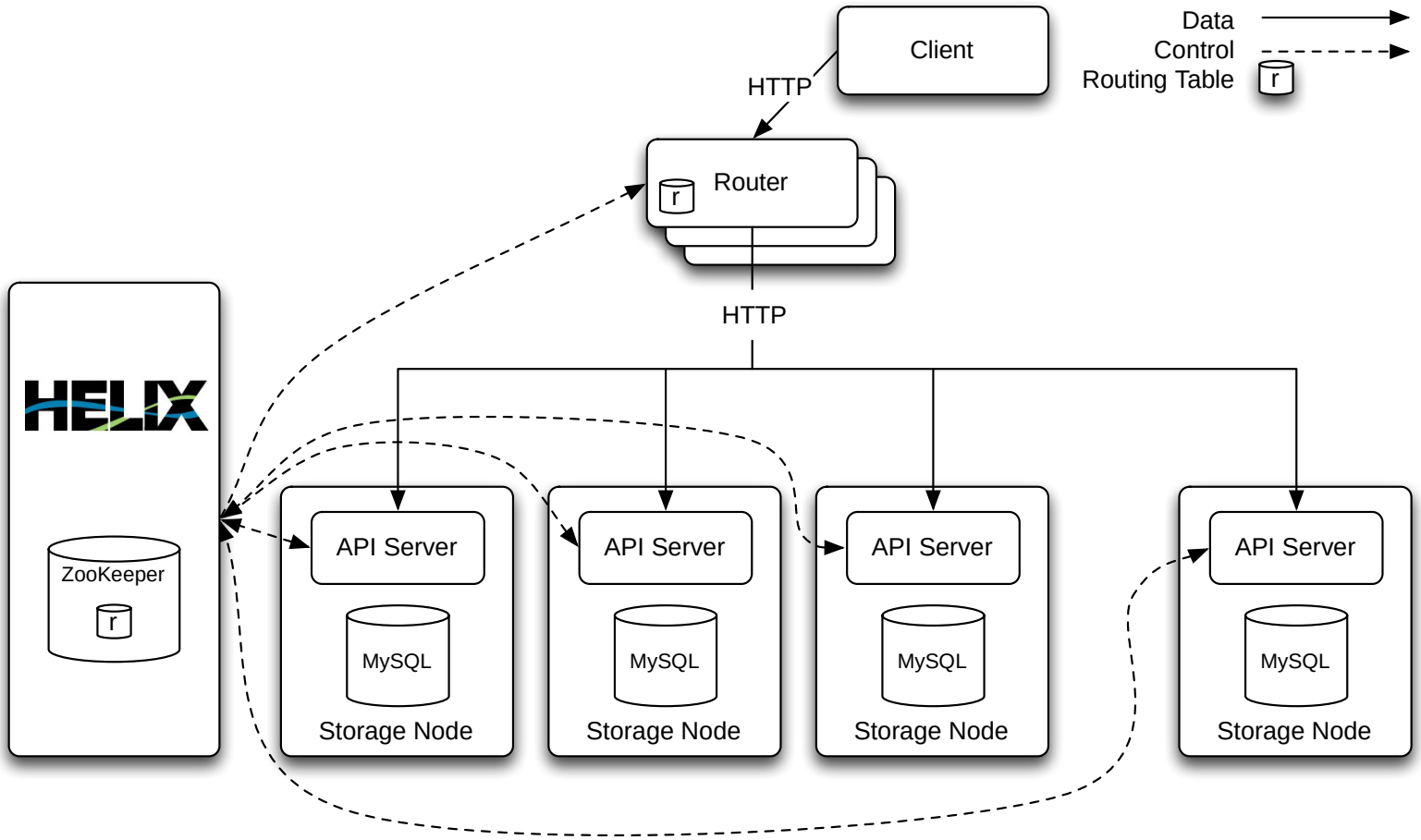
The big picture - Kafka@LinkedIn



Agenda

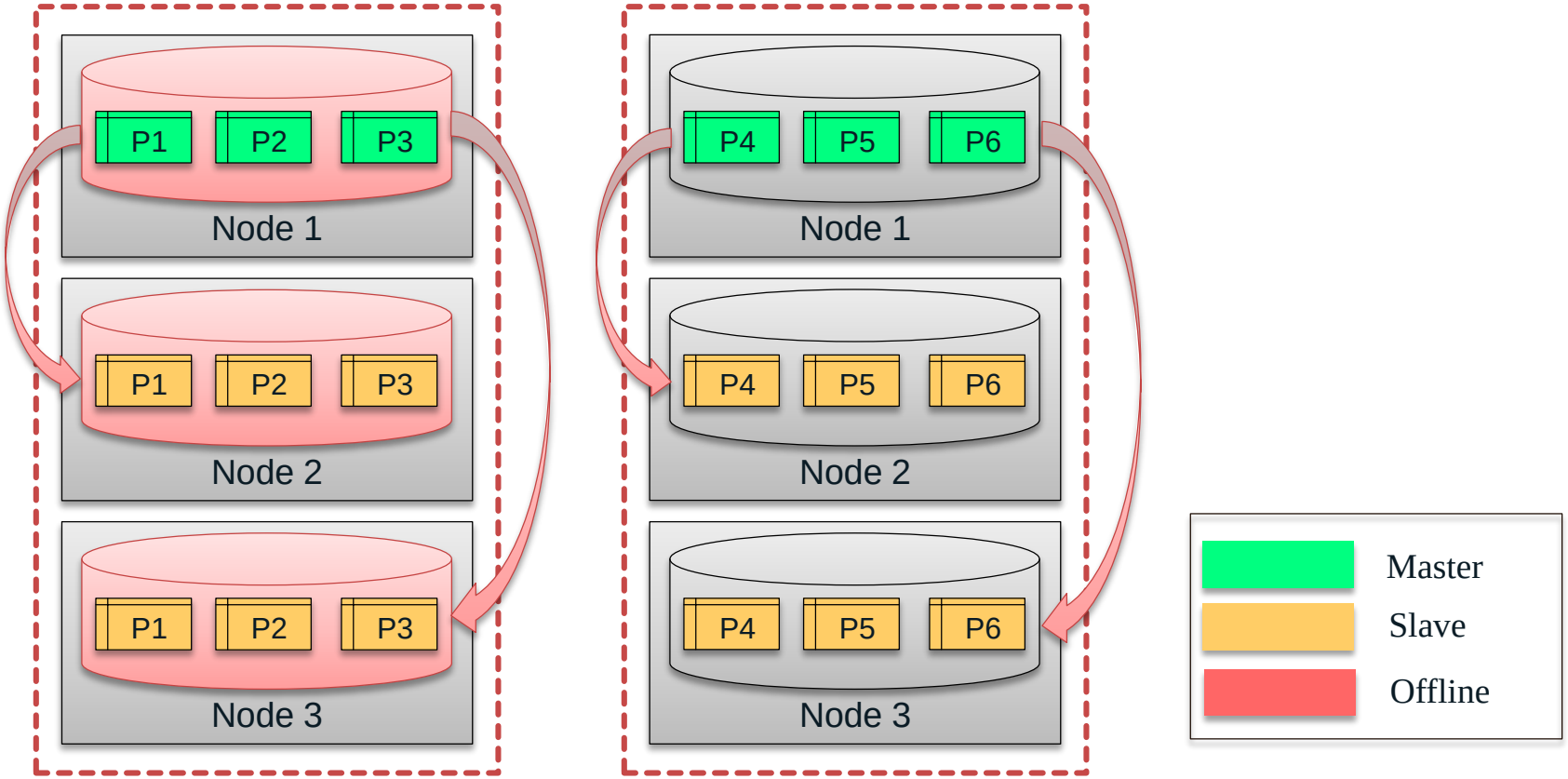
- Introduction to Apache Kafka
- Kafka based replication in Espresso
- Message Integrity guarantees
- Performance
- Large message handling
- Security

Espresso - A scalable NoSQL DB



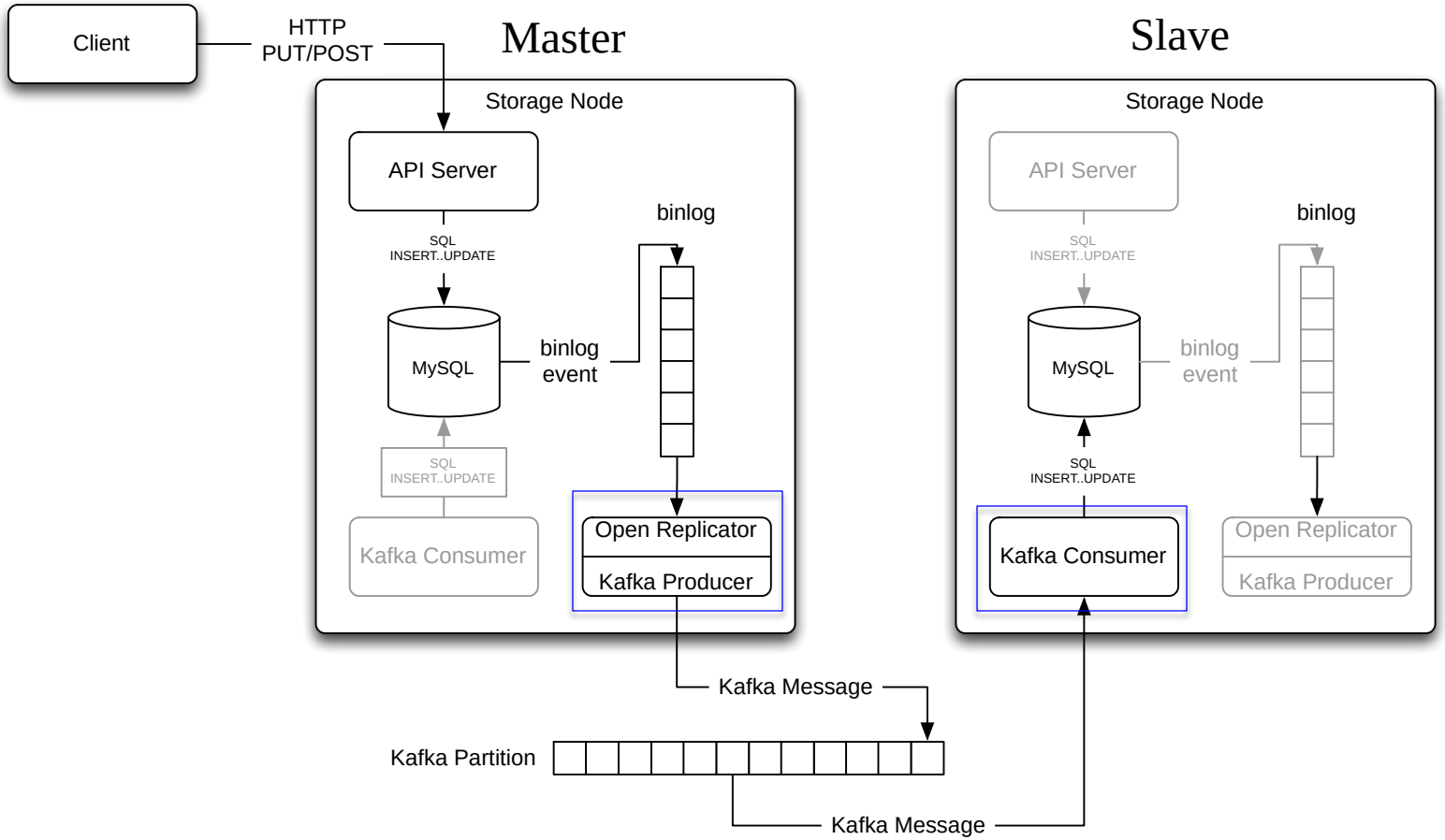
MySQL Based Replication

MySQL instance level replication



Kafka Based Replication

Kafka based partition level replication



Mission Critical Messaging

- No message loss
- In-order delivery
- Exactly once semantic

Message Integrity

- High throughput
- Low latency

Performance

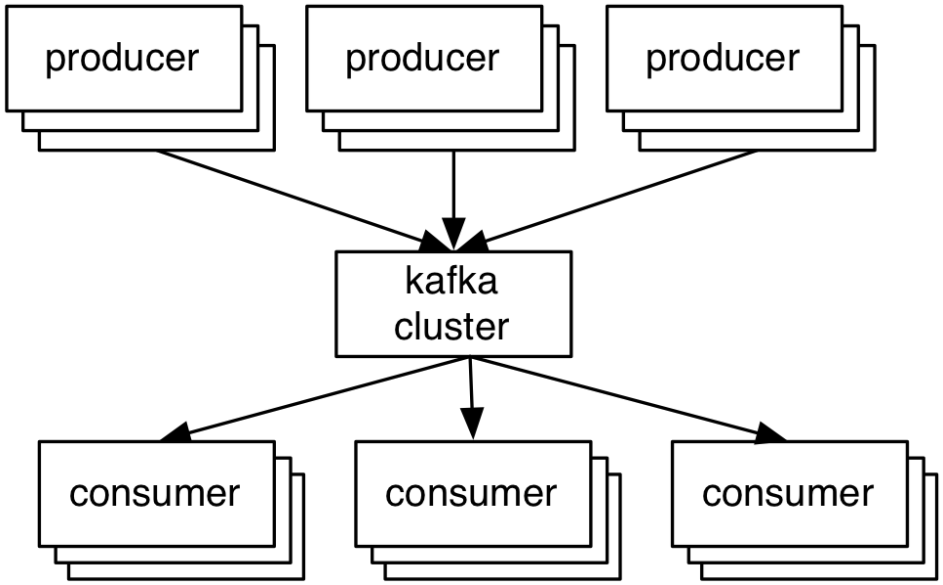
- Handle large messages
- Security

Agenda

- Introduction to Apache Kafka
- Kafka based replication in Espresso
- Message Integrity guarantees
- Performance
- Large message handling
- Security

Data Integrity

- No message loss
- In-order delivery
- Exactly once semantic



Produce

Replicate

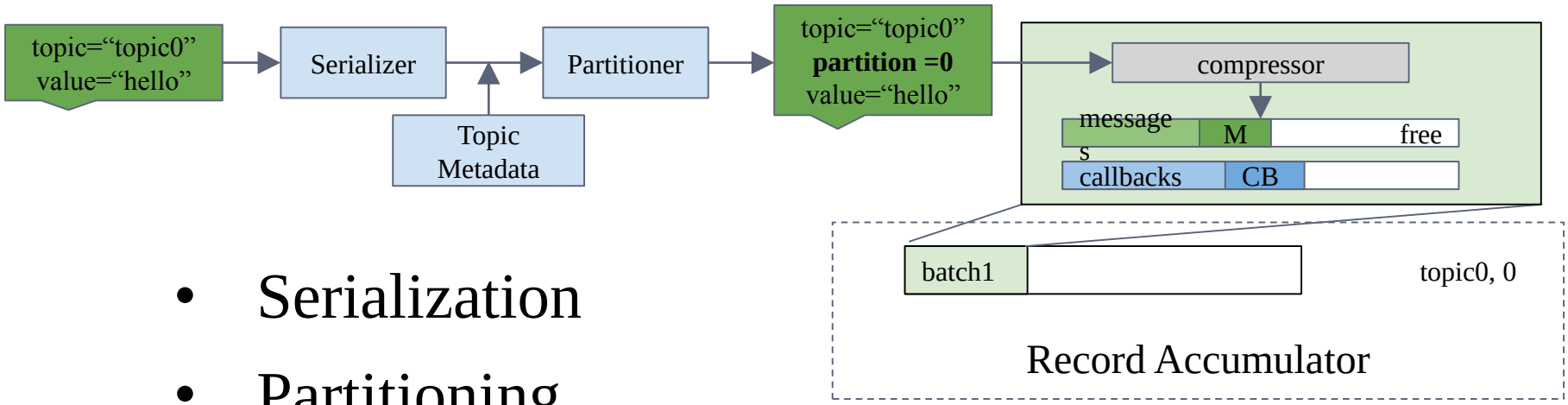
Consume

Produce

- A well implemented producer usually supports:
  - Batching the messages
  - Sending the messages asynchronously
- Example:
  org.apache.kafka.clients.producer.KafkaProducer

KafkaProducer

```
User: producer.send(new ProducerRecord("topic0", "hello"),
                    callback);
```

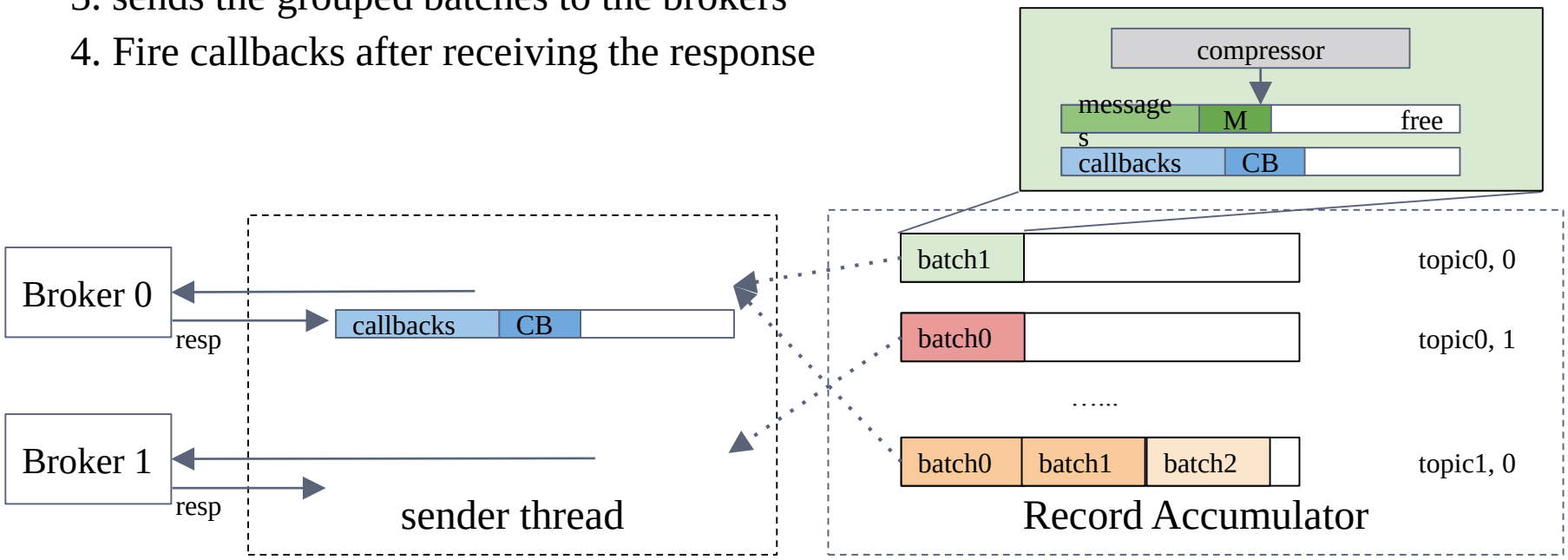


- Serialization
- Partitioning
- Compression
- ✓ Tasks done by the user thread

KafkaProducer

Sender:

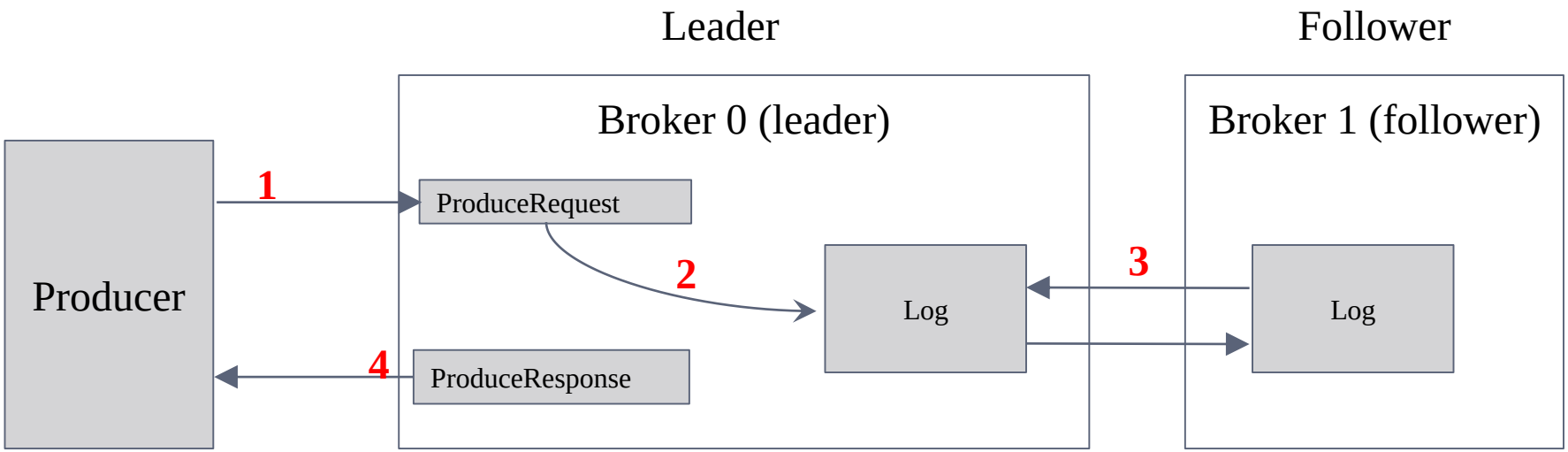
1. **polls** batches from the batch queues (one batch / partition)
2. **groups** batches based on the leader broker
3. sends the grouped batches to the brokers
4. Fire callbacks after receiving the response



Ensure No Message Loss

- Enable retries in the sender
  - (e.g. `retries=5`)
- Keep the messages not acked yet
  - Espresso only checkpoints at the transaction boundary after the callback is fired
- acks=all

acks=all



1. [Network] Send ProduceRequest
2. [Broker] Append messages to the leader's log
3. **[Broker] Replication (before sending the response)**
4. [Broker] ProduceResponse

Durability guarantee

- In-Sync-Replica (ISR) 如果一个replica能和leader保持同步的就叫 in-Sync的否则就是out of Sync
  - A replica that can keep up with the leader
    Leader Replica 肯定是In-Sync的
- Semantic for `acks`

no ack(0)---像UDP send出去后不需要response

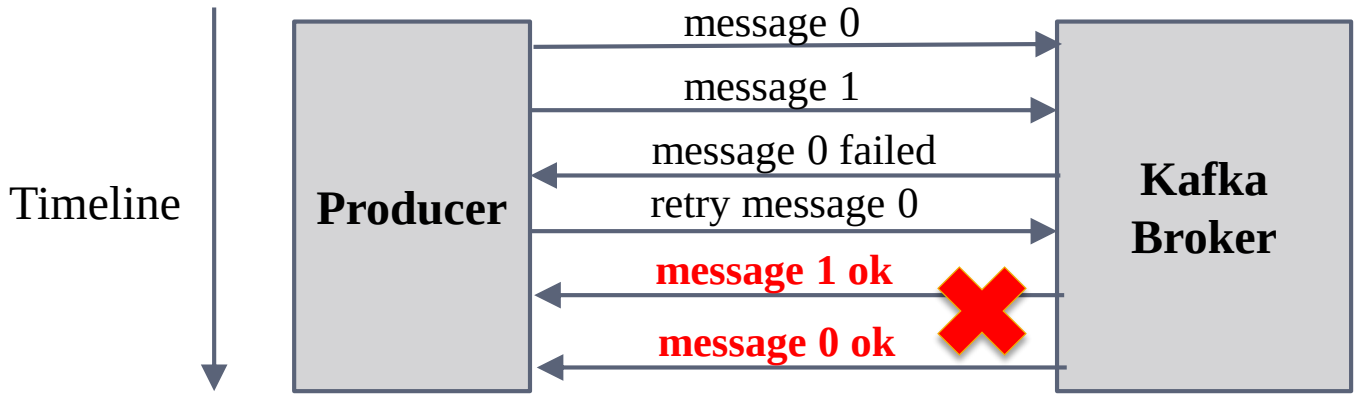
acks	Throughput	Latency	Durability
no ack(0)	high	low	No guarantee
Leader only(1)	medium	medium	leader
All ISR(-1)	low	high	All ISR

Leader only(1) ---需要等到leader把我这条消息append到tag log中去,然后把response返回给我，不需要等到broker把所有的repilica做完才把response返回给我

ALL ISR(-1)--这里指的并不是所有的replica都拿到消息，而是所有的In-Sync-Relica拿到消息

In order delivery

- max.in.flight.requests.per.connection = 1
  - Request pipelining 意思是需不需要等到上一个response才发下一个request



max.in.flight.requests.per.connection=**2**

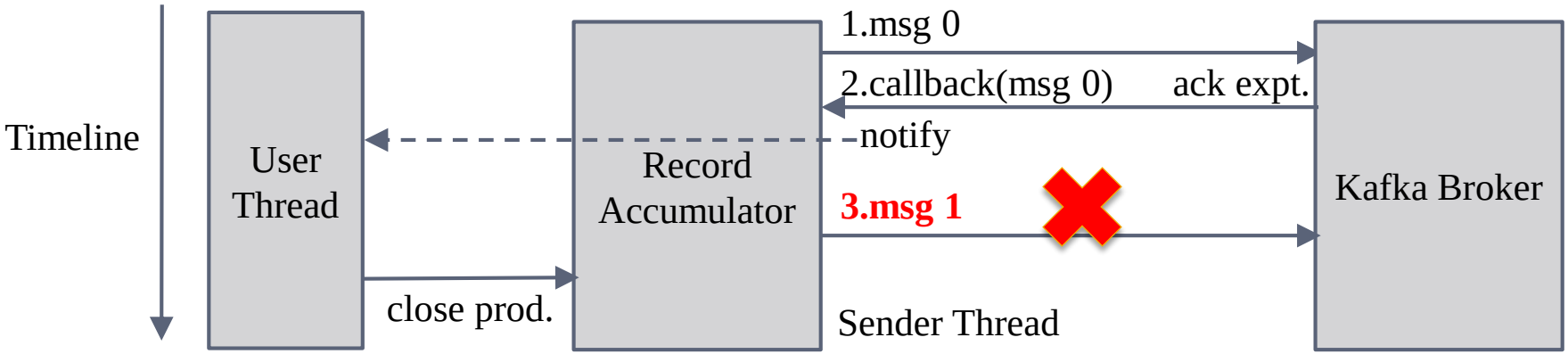
producer不用等msg 被ACK就直接发送msg1 导致消息发送顺序 错乱

In order delivery

send failure发生时需要注意需要保证消息发送顺序的话需要在callback里面把producer关掉，close时需要设置timeout=0，注意callback(producer到send pass in 的哪个callback)是在SendThread中执行的

- Close the producer **in the callback with 0 timeout on failures**
  - Callbacks are executed in the Sender thread

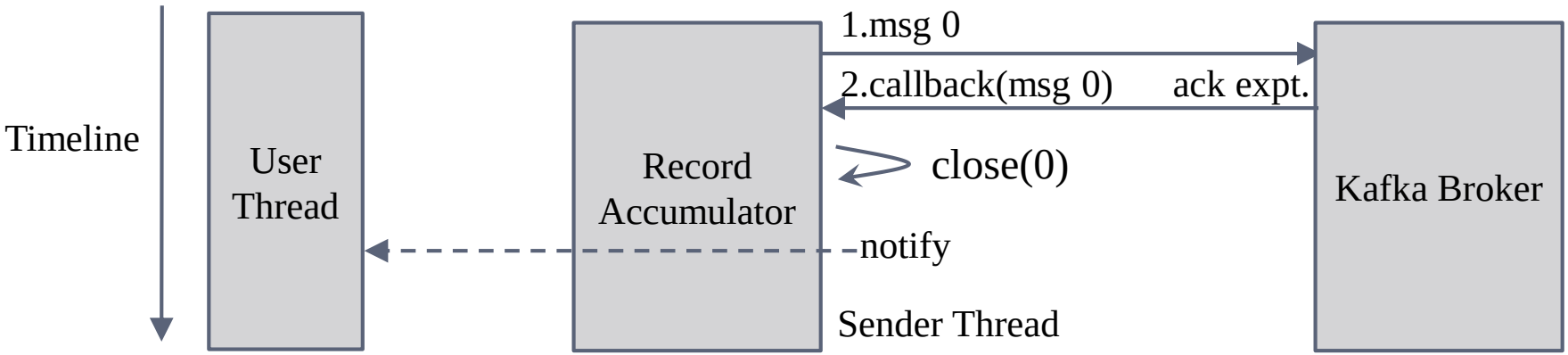
如果producer发送 msg 0失败 broker返回了一个Exception在callback上看到了这个exception，假如我们在callback上没把producer close掉，仅仅设置了一个flg，做了一个通知给UserTheard这条消息发送失败了，Sender Thread仍会接着往下执行，它并不知道UserThread已经接收到了一个fail，Sender Thread会接着执行，它会把下一条消息send出去,就会先看到msg1，发生顺序错乱



In order delivery

- Close the producer **in the callback with 0 timeout on failures**

同样发送 msg0失败，但我们在callback中把producer关闭掉，那么SenderThread就不会再把已存在Buffer中的消息继续发送消息



Broker

- min.isr=2

  min.isr=2 最小的In-Sync-Replica,意思是要保证In-Sync-Replica至少要有两个才认为消息是安全的

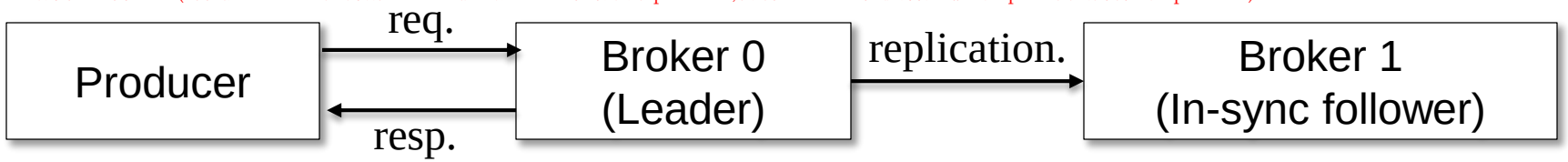
  - If acks=all, at least 2 copies of messages are required on the broker

    等到所有的In-Sync-Replica都拿到了我的消息后才可以返回 producer response

  - replication.factor needs be 3 to tolerate single broker failure

    这种情况下 tolerance 需要将replication.factor设置为3

如果Broker1挂掉的话replica还是有2个但 isr就只还有Broker 0了，如果这时不设置min.usr=2的话，如果producer send 一个request给Broker0因为isr只有broker0，此时的acks=all就相当于acks=1(leader only)，也就是只要leader append后就可以把消息返回了。但实际上在breaker端只有一个备份。如果broker0再死掉那么这条消息就丢失。之所以需要设置min.isr=2的原因也再这里(即要是Broker 1挂掉那Broker0就会返回一个错误给producer,告诉Producer我没有足够的replica来支持你的replication)



Replica = {0, 1}
ISR = {0, 1}

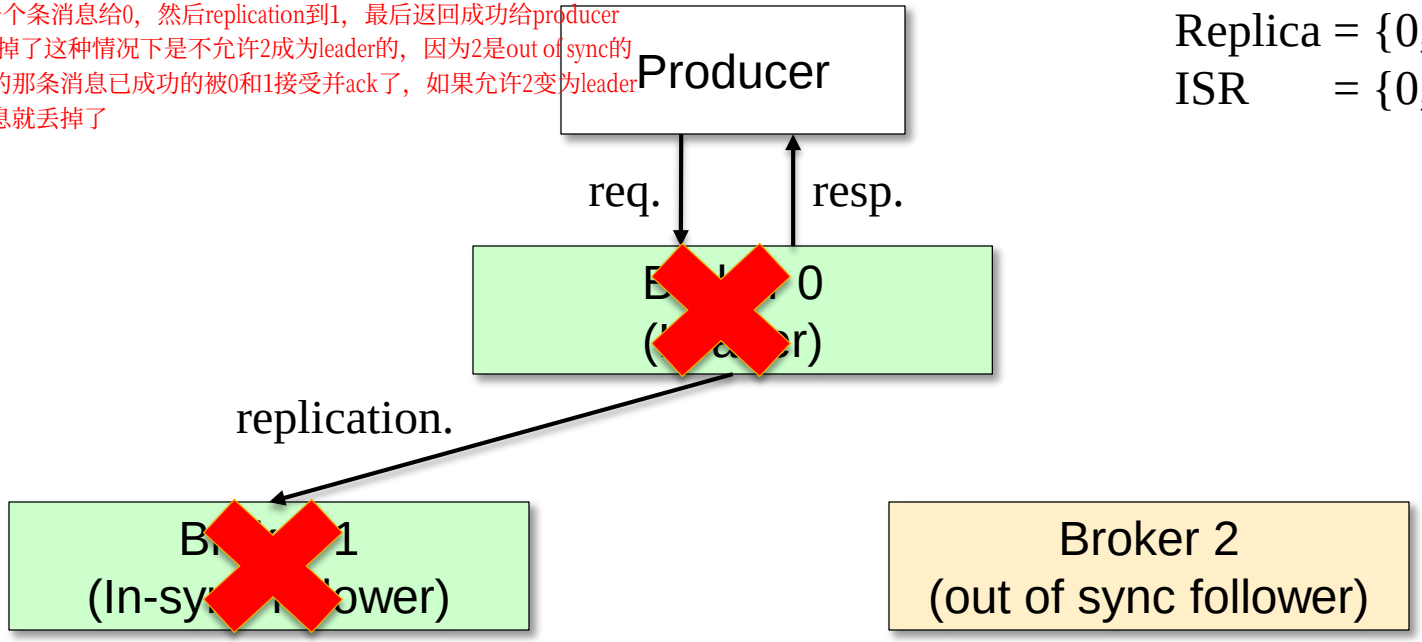
Replica = {0, 1}
ISR = {0}

Broker

- unclean.leader.election.enable= false
  - Only in-sync replicas can become leader

unclear.leader.election.ennable的意思是是否允许一个out of sync replica变成leader

producer发送一个条消息给0，然后replication到1，最后返回成功给producer
假如0 和1都死掉了这种情况下是不允许2成为leader的，因为2是out of sync的
因为之前发送的那条消息已成功的被0和1接受并ack了，如果允许2变为leader
的话那那条消息就丢掉了



Replica = {0, 1, 2}
ISR = {0, 1}

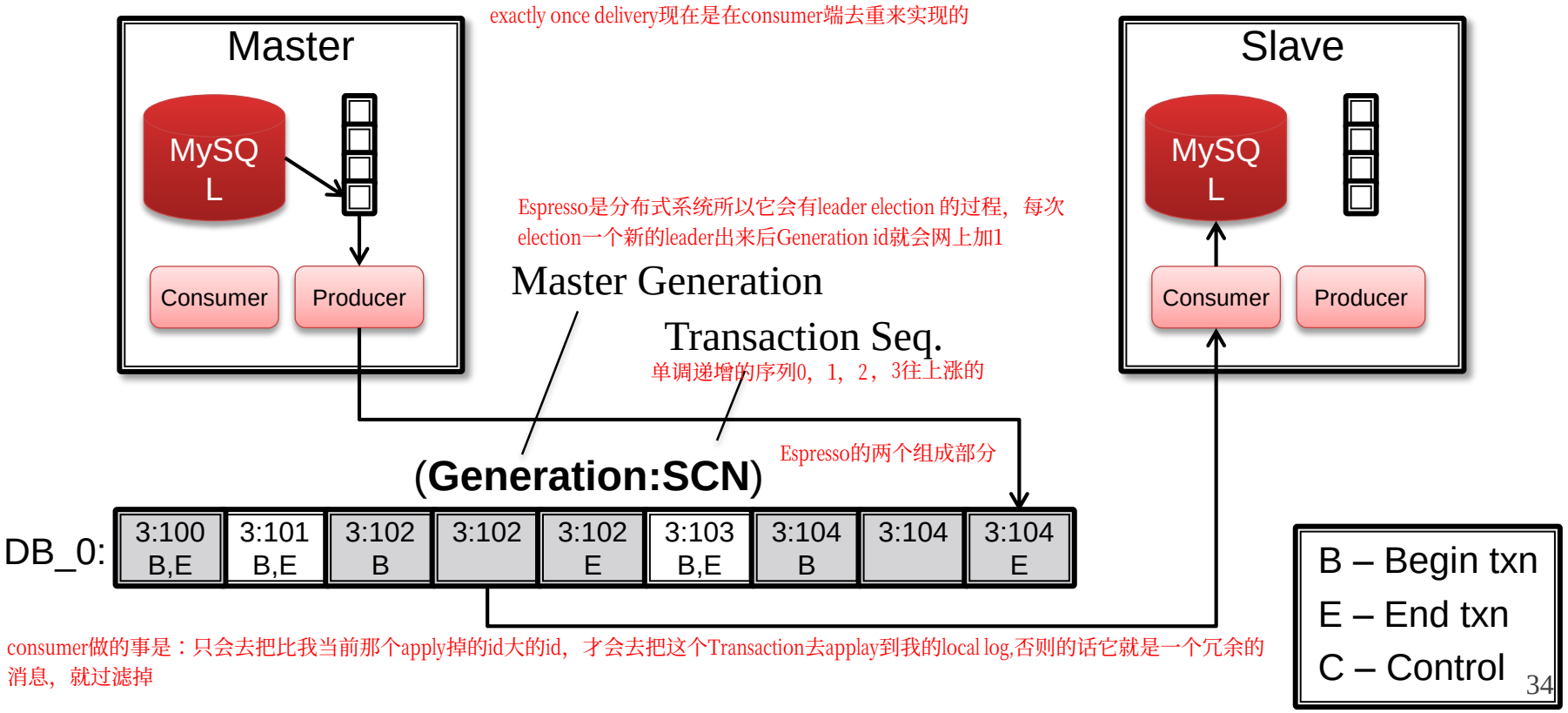
Consumer

- Disable auto offset commit
- Manually commit offset
  - only commit offsets after successfully processing the messages

需要注意的是不要提前的去commit offset，需要等到消息完全处理完后再去commit offset
建议把 auto offset commit disable

Consumer

- Exactly once delivery (Espresso)
  - Consumer only apply higher **Generation:SCN**



Agenda

- Introduction to Apache Kafka
- Kafka based replication in Espresso
- Message Integrity guarantees
- Performance
- Large message handling
- Security

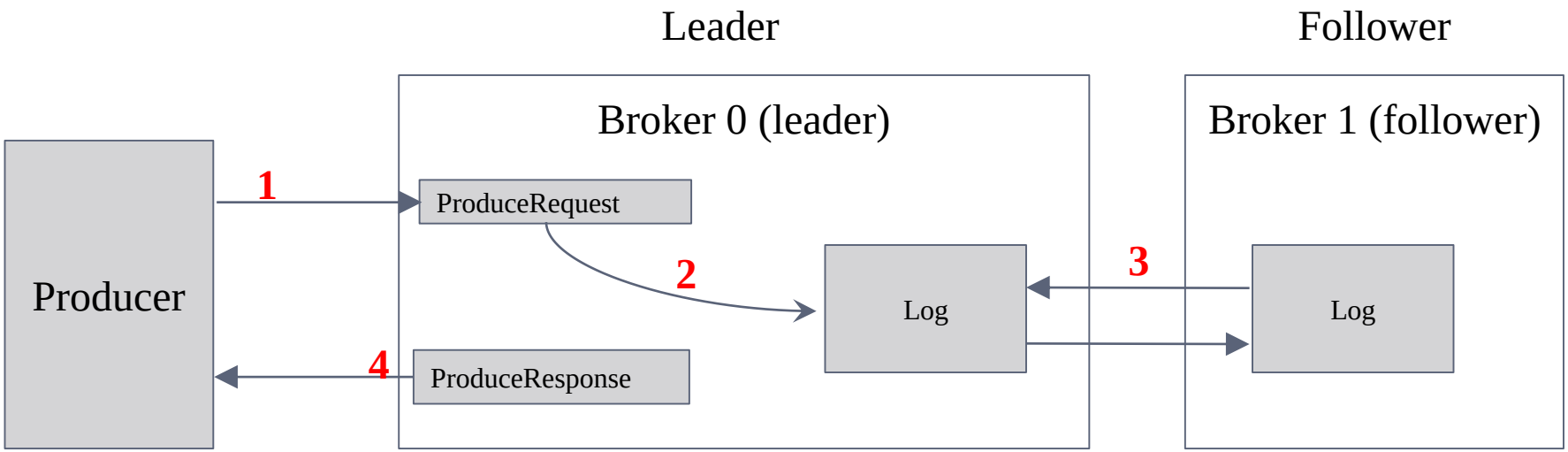
Performance Tuning

- Performance tuning is case by case
  - Traffic pattern sensitive
  - Application requirement specific
- Producer performance is more interesting
  - **Especially for acks=all**
  - See more
    - Producer performance tuning for Apache Kafka

      (http://www.slideshare.net/JiangjieQin/producer-performance-tuning-for-apache-kafka-63147600)

性能调优方面更关心Producer端，尤其是acks=all时，因为你的整个延迟上会包含一个replication的延迟

Latency when acks=all



1. [Network] Send ProduceRequest
2. [Broker] Append messages to the leader's log
3. **[Broker] Replication (synchronously)** → increases latency
4. [Broker] ProduceResponse

acks=all 就会涉及到一个同步的repliction,会增加latency

Latency when acks=all

解决方法是增加num.replica.fetchers

- Kafka replication is a pull model
- Increase **num.replica.fetchers**
  - Parallel fetch

增加了replication的并发度

- Not perfect solution
  - Diminishing effect (1/N)

边界效应递减的原因：比如一个replica fetcher我找到了两个时那latency的降幅是50%，从2-3个时呢也许就只有10%l了

  - Scalability concern

kafka中的replica fetcher是对broker而言的

    - Replica fetchers per broker = (Cluster_Size - 1) * num.replica.fetchers

eg:50个node 的kafka cluster 那把我们的replica fetcher设置为32，那结果是在每一个broker上他会有49*32那么多个replica fetchers，对于大的cluster来说并不适用了

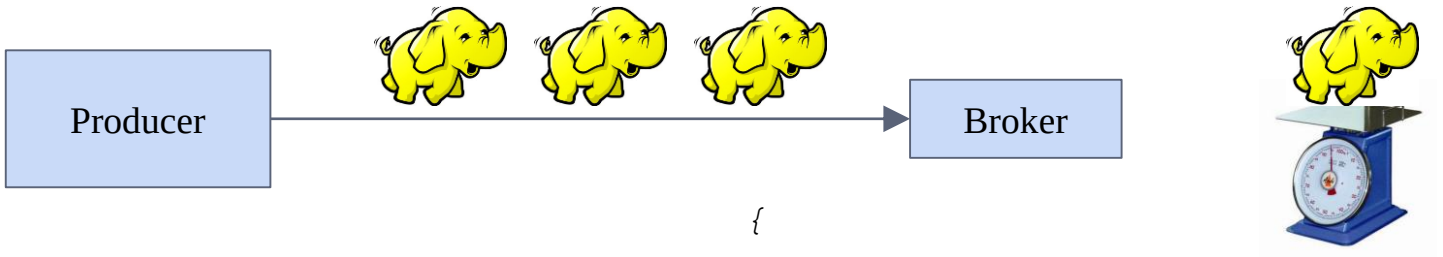
Agenda

- Introduction to Apache Kafka
- Kafka based replication in Espresso
- Message Integrity guarantees
- Performance
- Large message handling
- Security

What is a large message

大消息的处理

- Kafka has a limit on the maximum size of a single message
  kafka中消息的大小是有一个阈值的
  - Enforced on the compressed wrapper message if compression is used



消息太大 会跑出这样的异常

首先超大消息会对broker来说是一个麻烦的事情，会占用很大的内存，造成内存压力，即席，内存的分配都会产生一些问题
其次 非常长昂贵，serialization和deserialization,decompression, append to log都会花更长的时间，整个过程都会花更多的资源，导致整体的延迟变高
第三 在绝大数的应用下一个比较合理的最大的消息size 的threshold是能够满足绝大多数的use case

Why limit message size

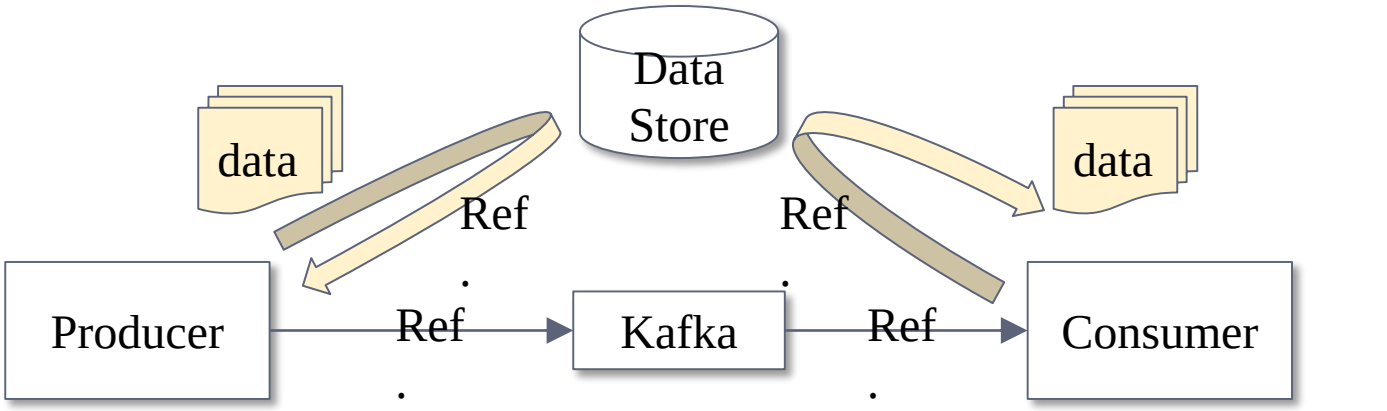
- Increase the **memory pressure** in the broker
- Large messages are **expensive** to handle and could **slow down the brokers**.
- A reasonable message size limit can handle vast **majority of the use cases**.

Typical solution

- Reference based messaging

解决超大文件的一个方案

先把超大的文件存在额外的DataStore，然后返回引用给Producer



Consumer端根据ref拿到真正的数据

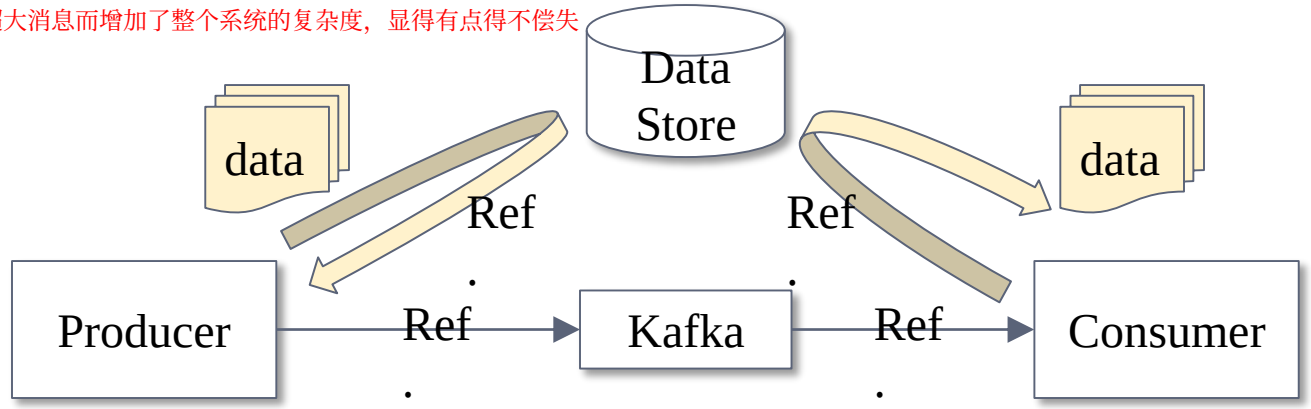
What works

- Unknown maximum row size
- Strict no data loss
- Strict message order guarantee

上述解决方案的可行性

Works fine as long as the durability of the data store can be guaranteed.

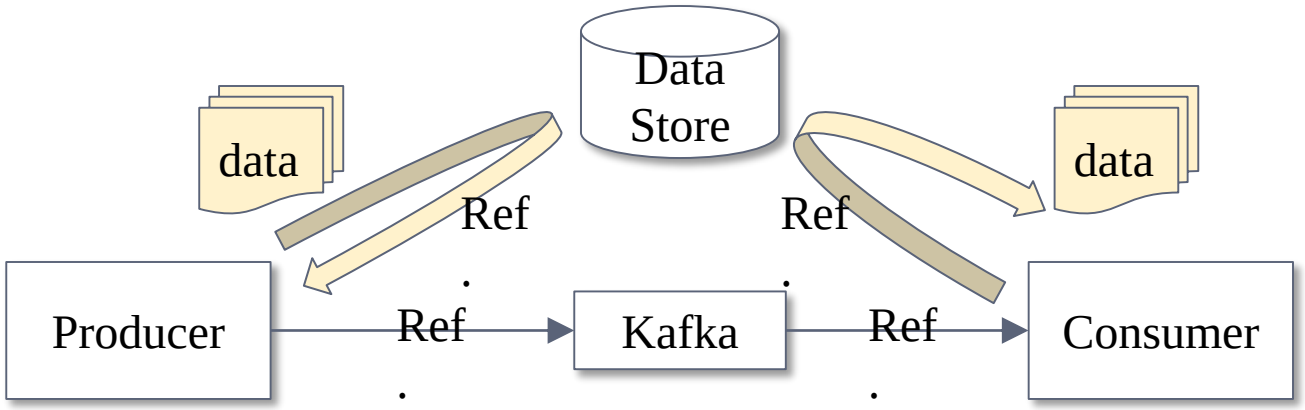
但是超大消息在绝大多数的应用中的数量占比还是比较少的，为了
处理这些少量的超大消息而增加了整个系统的复杂度，显得有点得不偿失



What does not work

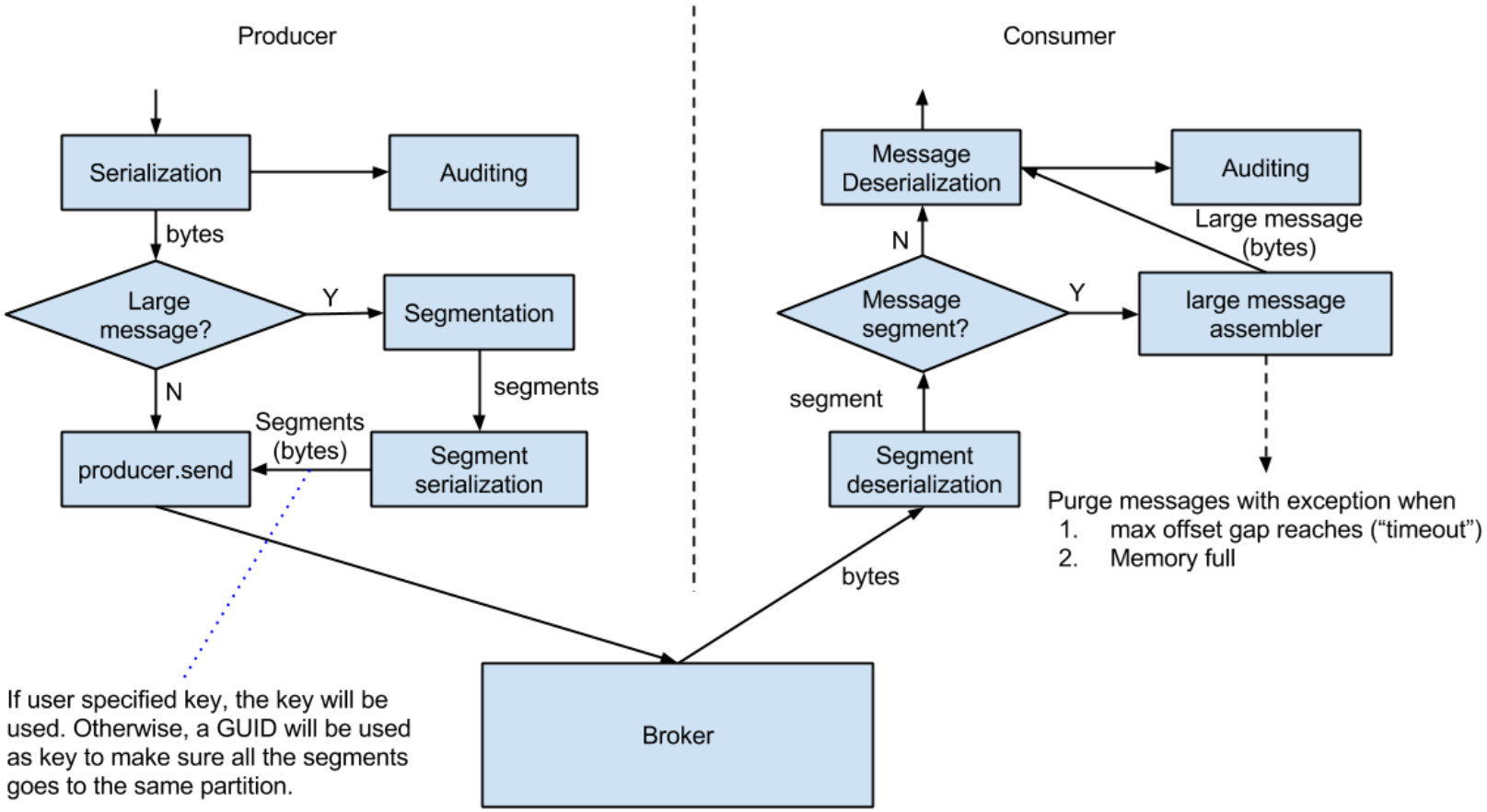
- Replicates a data store by using another data store....
- Sporadic large messages
- Low end to end latency
  - There are more round trips in the system.
  - Need to make sure the data store is fast

上述解决方案不好的地方



In-line large message handling

最终采用的方式(超大文件处理)，把大消息切成很多小段，然后把这些小段作为单独的消息send到broker,在consumer端把这些掉段全部consume下来后，再把它assembler回去



In-line large message support

解决超大文件 的两种方式中消息的对比

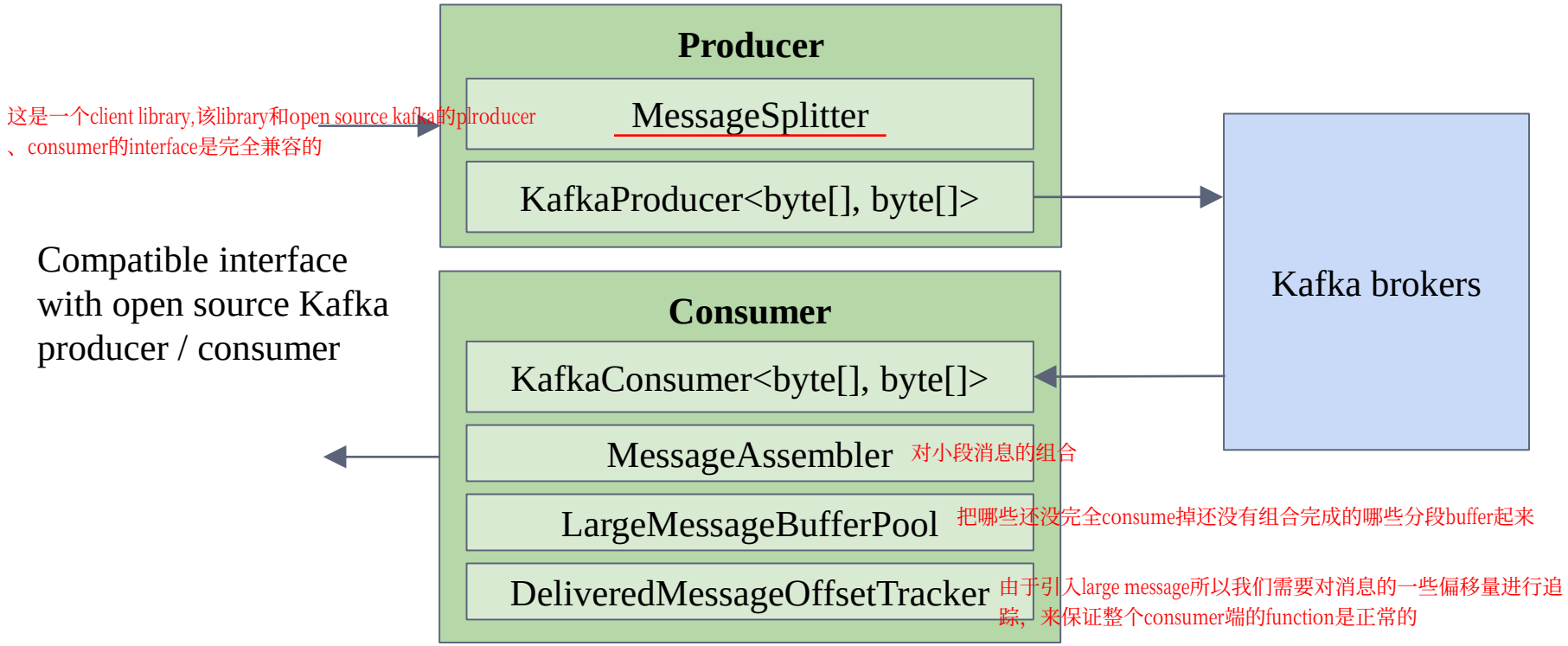
最大的优势在于简单 只使用kafka的一套东西，不需要额外的

	Reference Based Messaging	In-line large message support
Operational complexity	Two systems to maintain	Only maintain Kafka
System stability	Depend on : ▪ The consistency between Kafka and the external storage ▪ The durability of external storage	Only depend on Kafka
Cost to serve	Kafka + External Storage	Only maintain Kafka
End to end latency	Depend on the external storage	The latency of Kafka
Client complexity	Need to deal with envelopes	Much more involved
Functional limitations	Almost none	Some limitations

缺点：1.client端尤其是在consumer端增加了复杂度
2.在consumer端会对已有的功能产生一些限制

In-line large message support

所以最后做的是这样一个框架



Some details

in-line large message handler的具体实现

- Many interesting details 由很多需要关注的细节
  - The offset of a large message
  - Offset Tracking
  - Rebalance and duplicates handling
  - Producer callback
  - Memory management
  - Performance overhead
  - Compatibility with existing messages

Some details

- Many interesting details
  - The offset of a large message
  - Offset Tracking
  - Rebalance and duplicates handling
  - Producer callback
  - Memory management
  - Performance overhead
  - Compatibility with existing messages

Offset of a large message

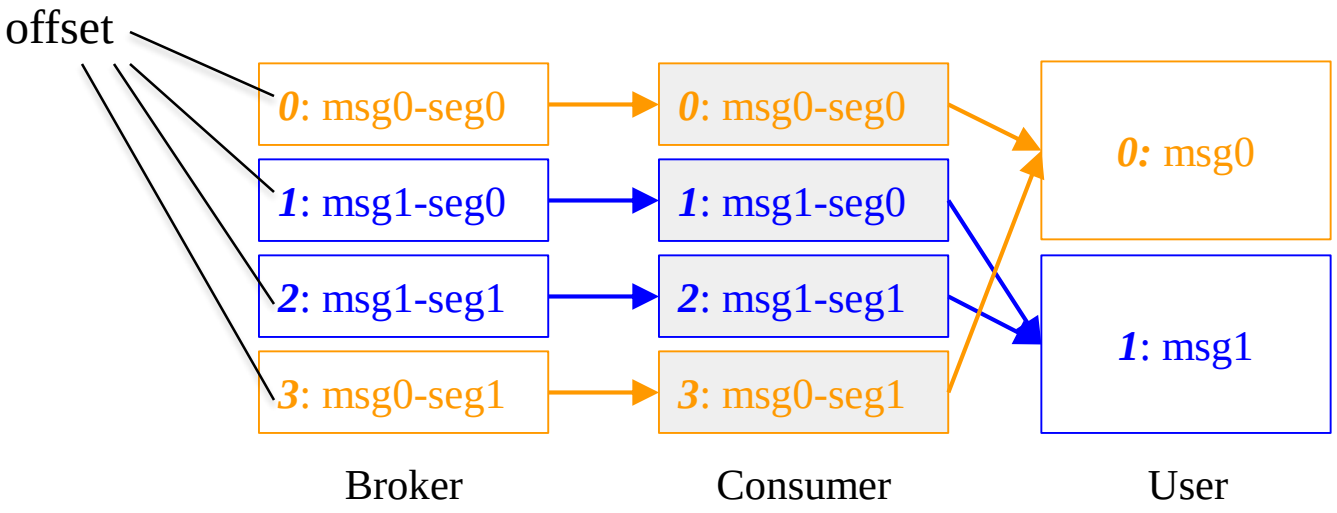
offset是kafka 中消息在log中的逻辑位置

- Each message in Kafka has an **Offset**
  - The logical sequence in the log
- Two options for large message's offset
  - The offset of the **first** segment
  - The offset of the **last** segment

大消息切分为小消息后有很多的offset，当把小消息返回给用户时只能返回一个offset给用户，到底返回哪一个呢？

The offset of a large message

- offset of a large message = offset of **first** segment
  - First seen first serve
  - Expensive for in-order delivery



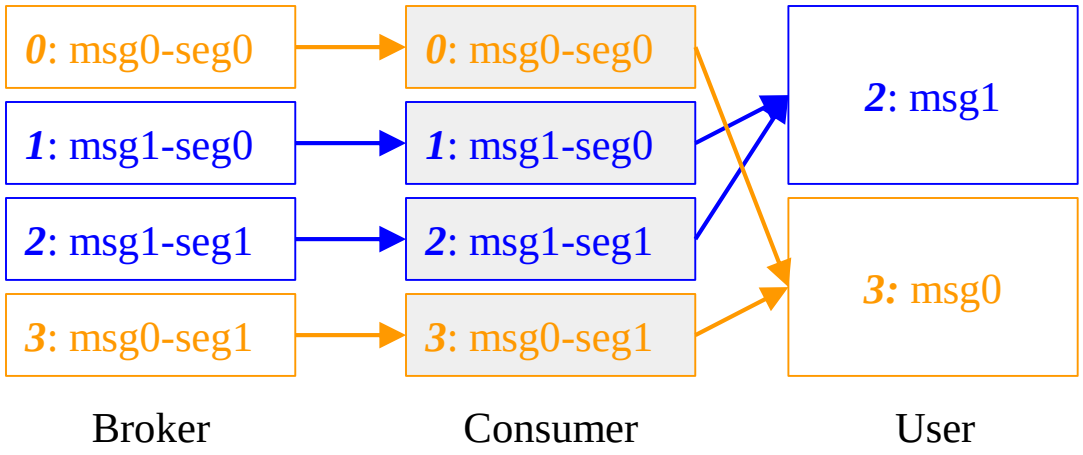
尽管已经把msg1的所有分段都consume下来了但是不能把消息发送给用户，因为需要考虑发送的delivery order是正常的，因为这个时候msg0的offset已经固定为0，msg1的offset一定为1，因为我们使用的是第一个分段的offset作为large message的offset,所以我们不能把offset为1的msg1先给delivery，我们必须先delivery msg0，因为它的offset为0，所以我们必须要等到第四个消息被consumer下来，然后在这个时候之后我们才能把msg0 telly of fetch 0 delivery 给user,然后才把msg1 telly of fetch 1 delivery给user

Max number of segments to buffer: 4

The offset of a large message

- offset of a large message = offset of **last** segment
  - Less memory consumption
  - Better tolerance for partially sent large messages.
  - Hard to seek()

不需要等msg0就可以把msg1给user



Max number of segments to buffer: 3

More details

- Offset Tracking
- Rebalance and duplicates handling
- Producer callback
- Memory management
- Performance overhead
- Compatibility with existing messages

大消息处理

- http://www.slideshare.net/JiangjieQin/handle-large-messages-in-apache-kafka-58692297

Open Source Clients Library

- Our client library will be open sourced shortly
  - Large message support
  - Auditing

Agenda

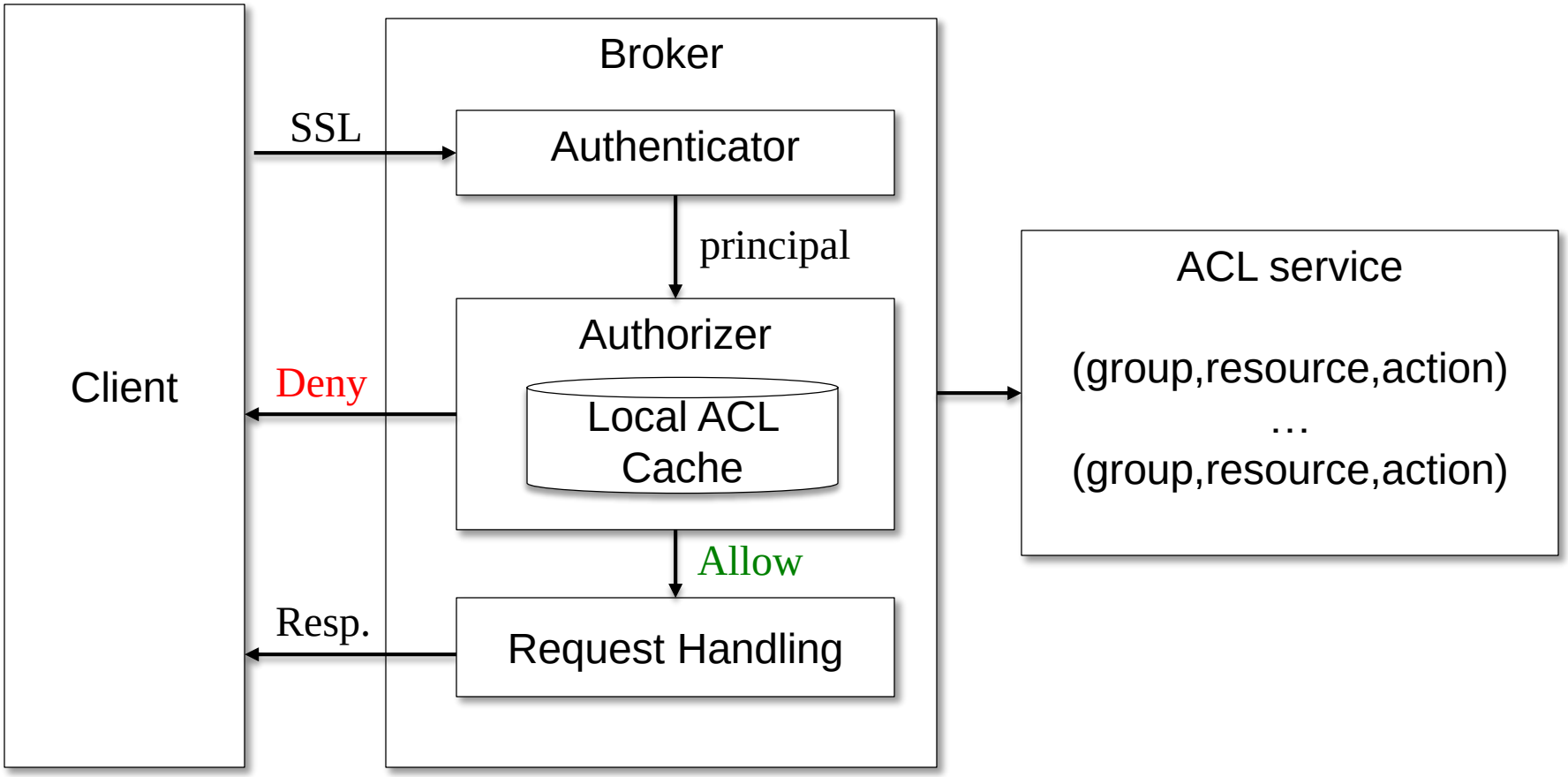
- Introduction to Apache Kafka
- Kafka based replication in Espresso
- Message Integrity guarantees
- Performance
- Large message handling
- Security

Security in Kafka

- Authentication (SSL, Kerberos, SASL)
- Authorization (Unix-like permission)
  - **Resource**: Cluster, Topic, Group 现在主要指的是consumer group将来可以指producer group
  - **Operation**: Read, Write, Create, Delete, Alter, Describe, Cluster Operation, All
- TLS encryption

Our solution

Authorizer performance is important



Q & A