Jiangjie (Becket) Qin
LinkedIn

Mission Critical Messaging with Apache Kafka

InfoQueue

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

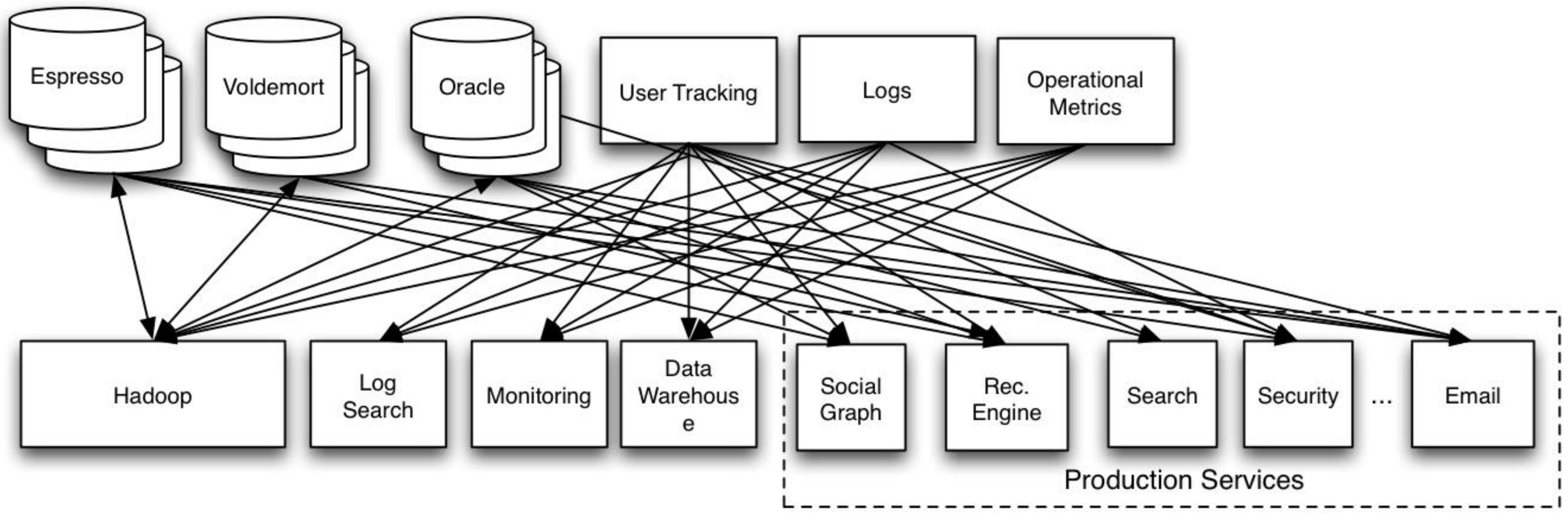
Agenda

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

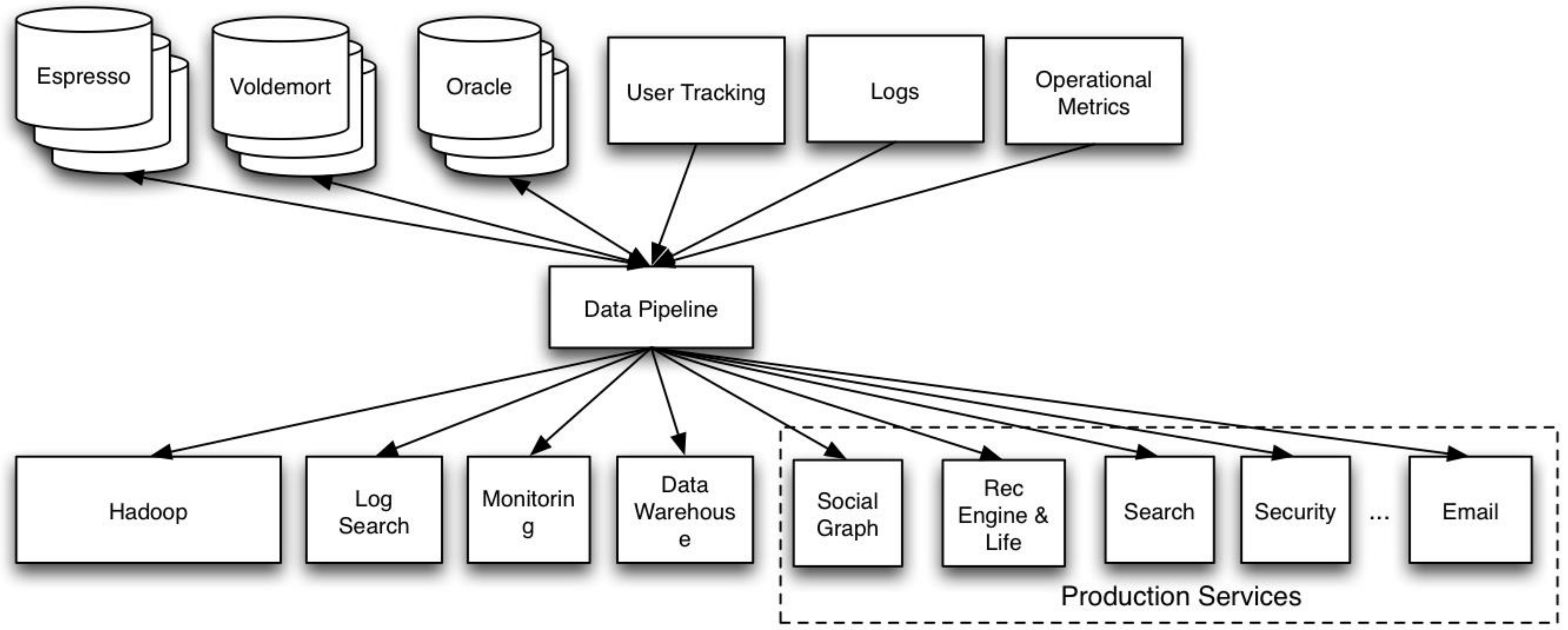
Agenda

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

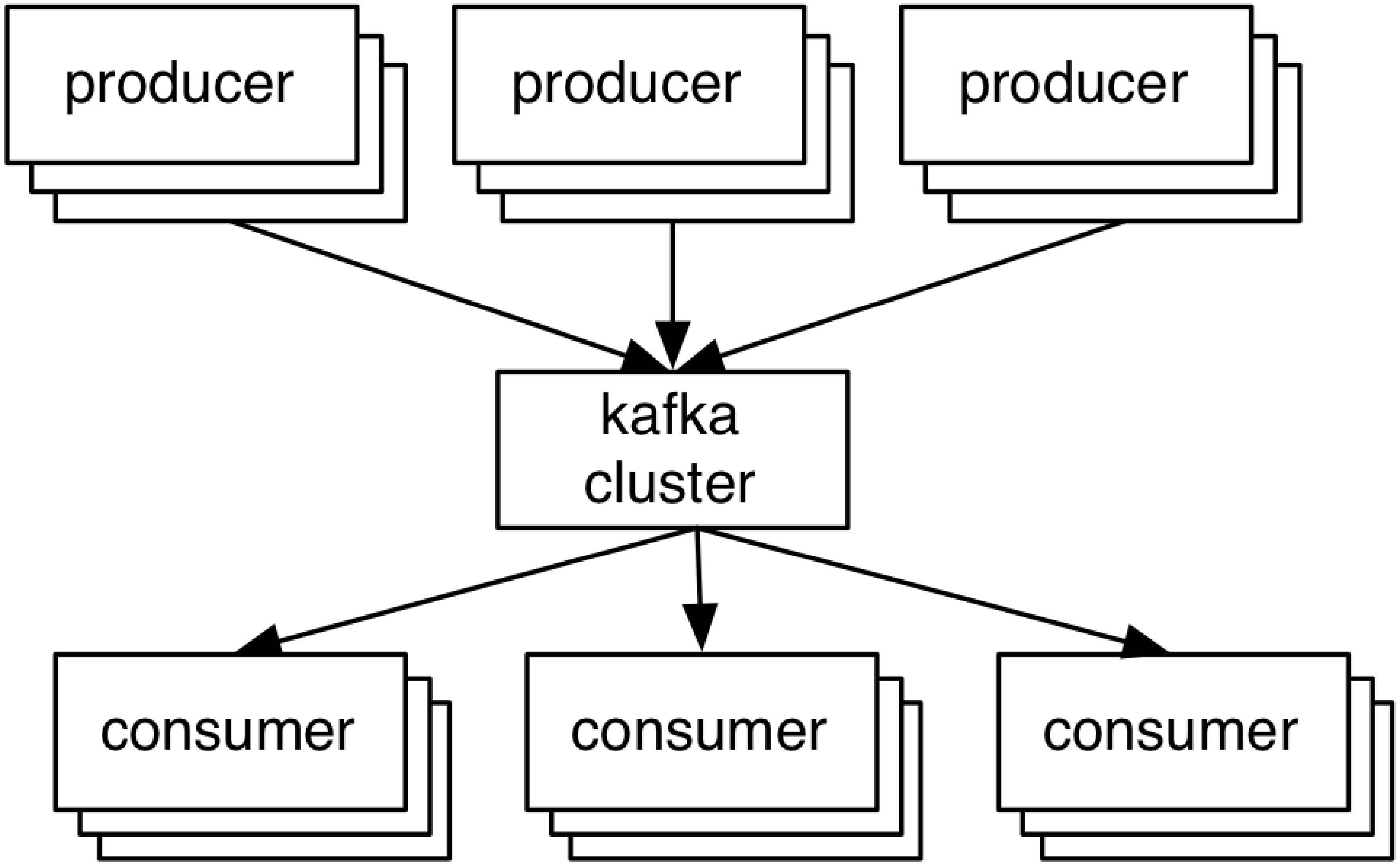
One way to aggregate data



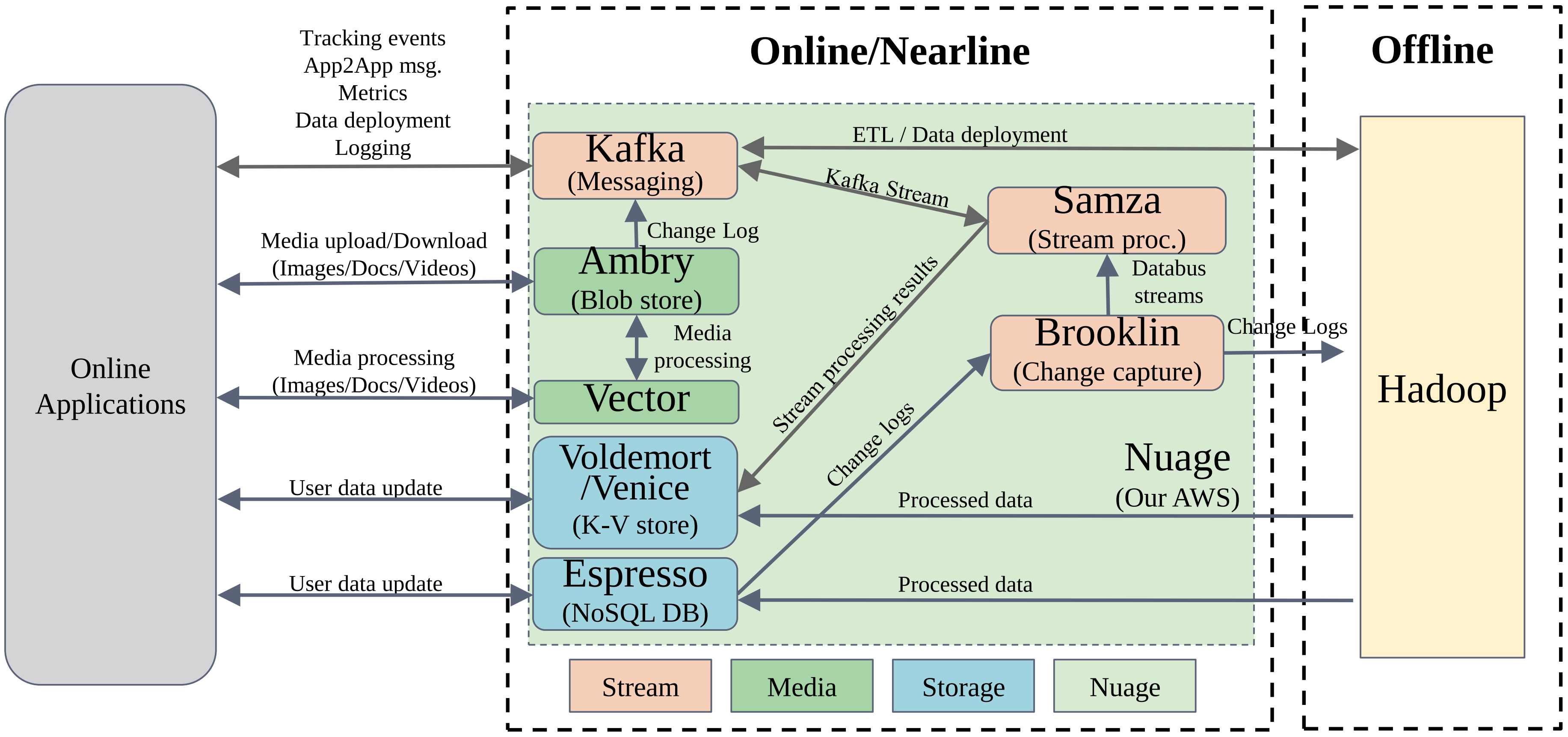
Another way to aggregate data



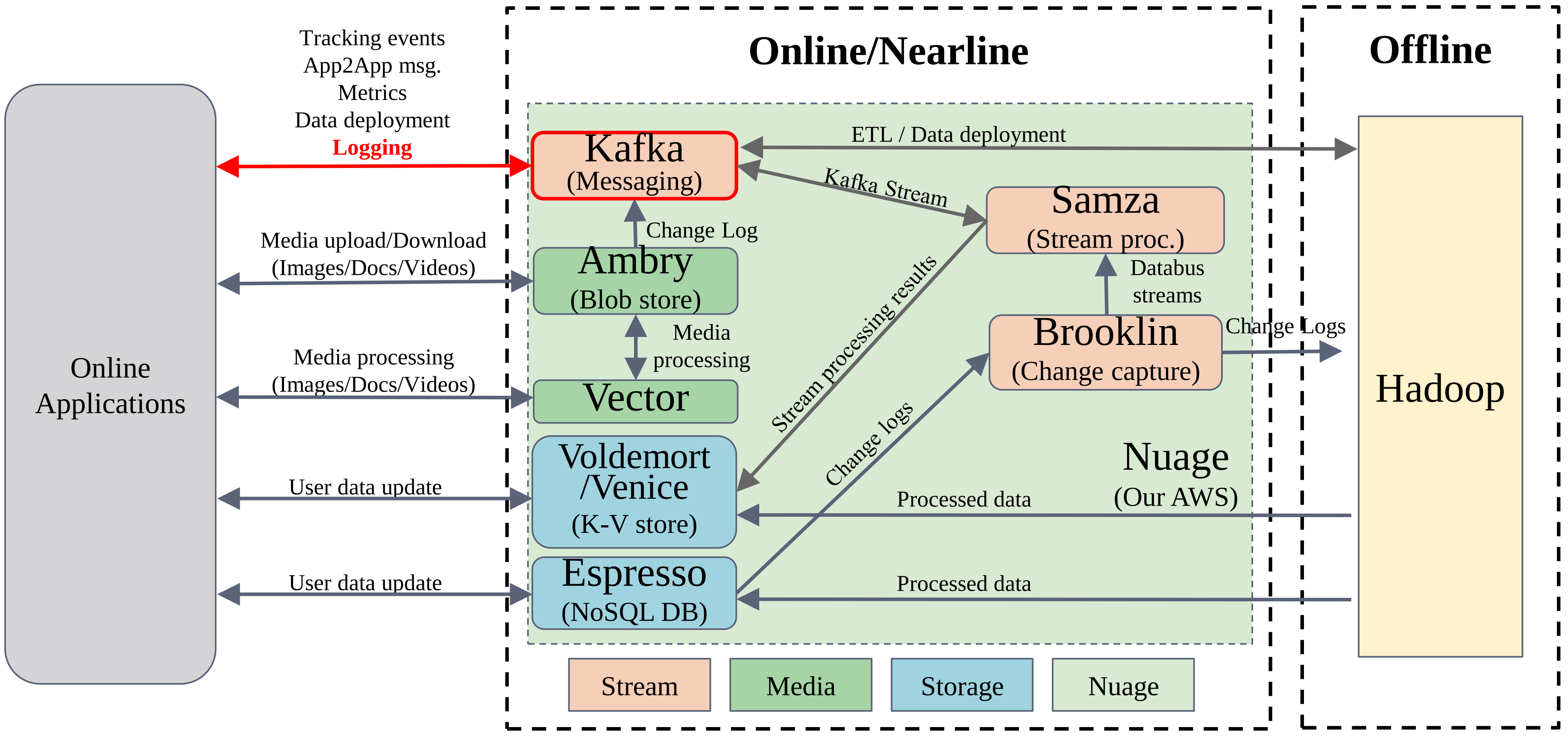
Apache Kafka



The big picture - Kafka@LinkedIn

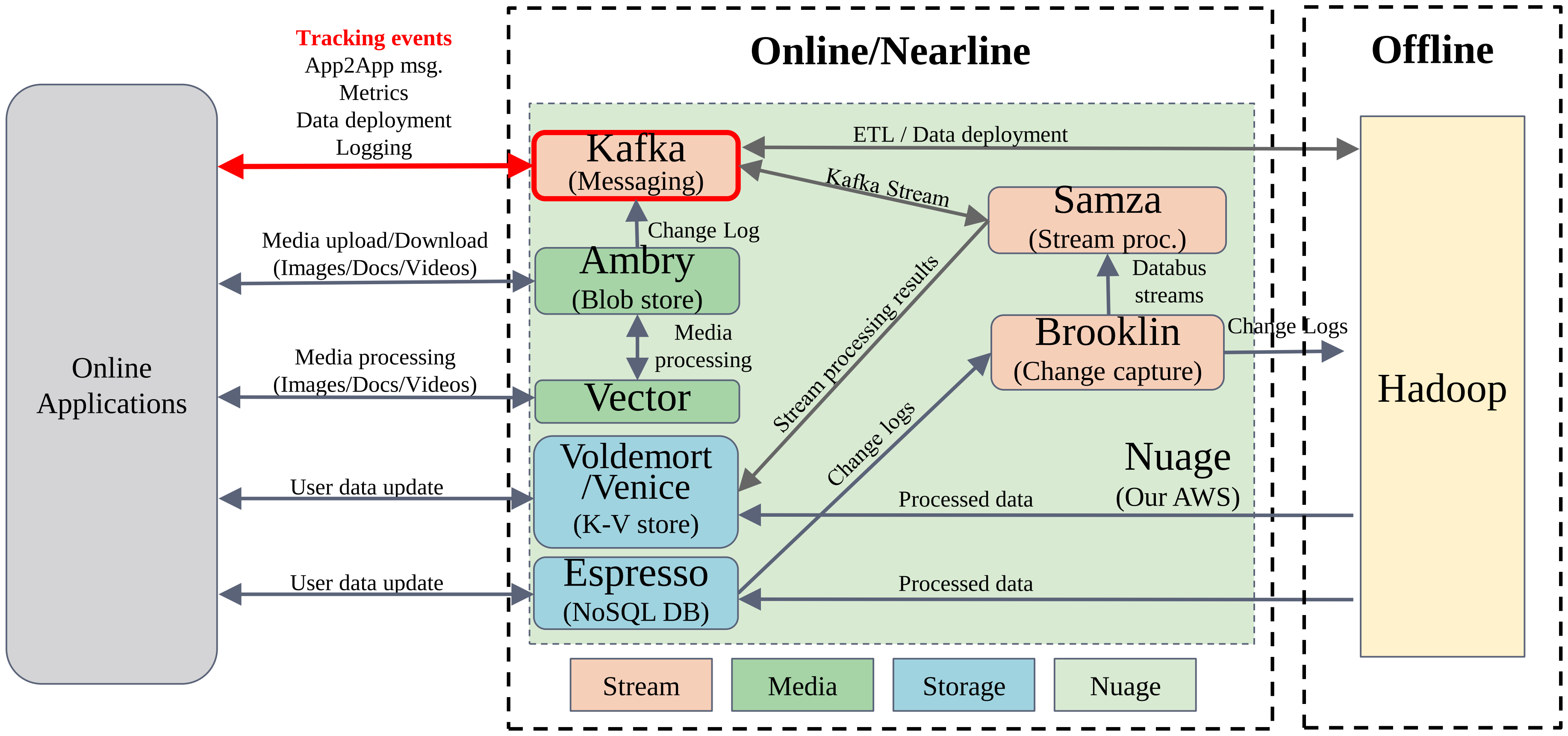


The big picture - Kafka@LinkedIn

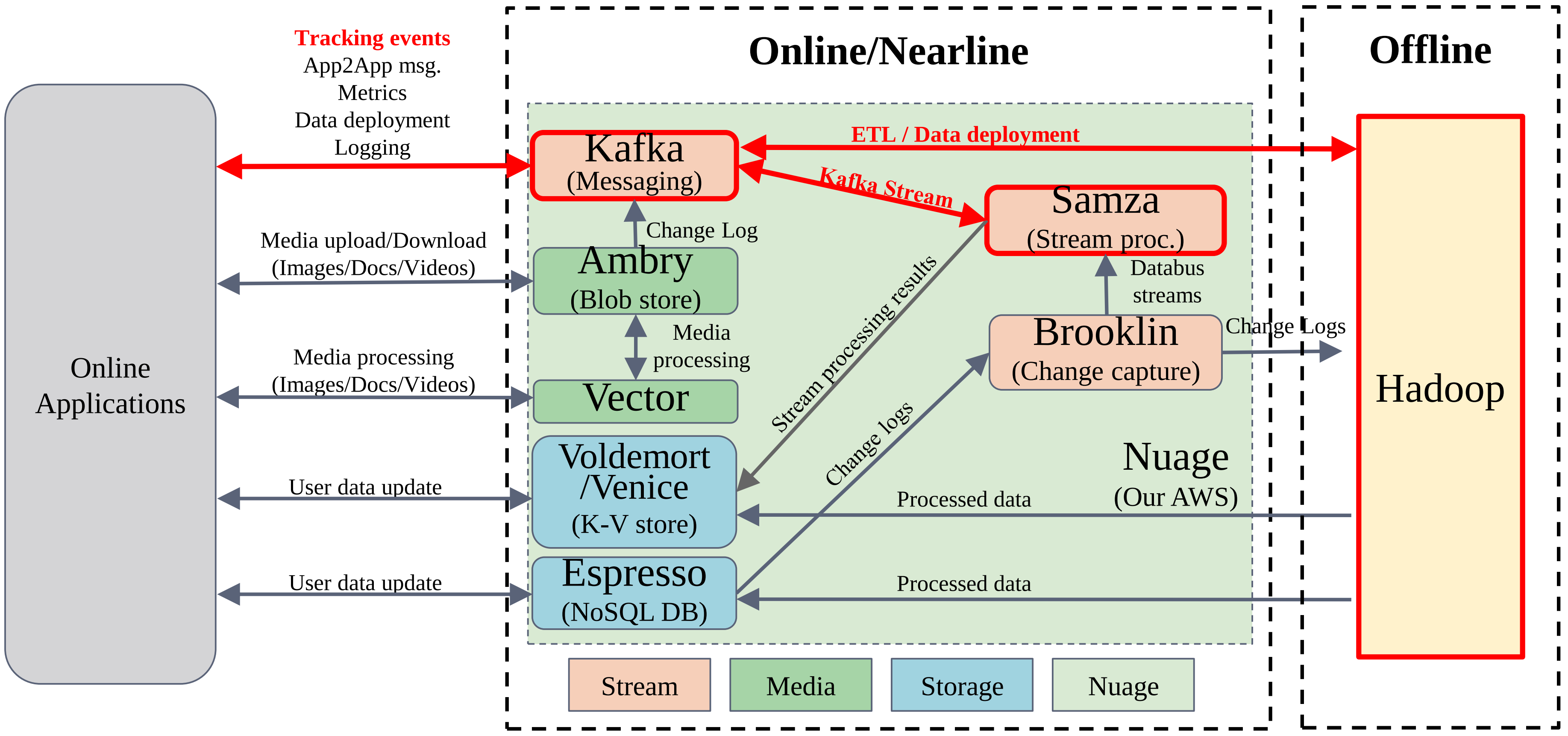


下午16:40，百宴厅1，LinkedIn基于Kafka和ElasticSearch的实时日志分析

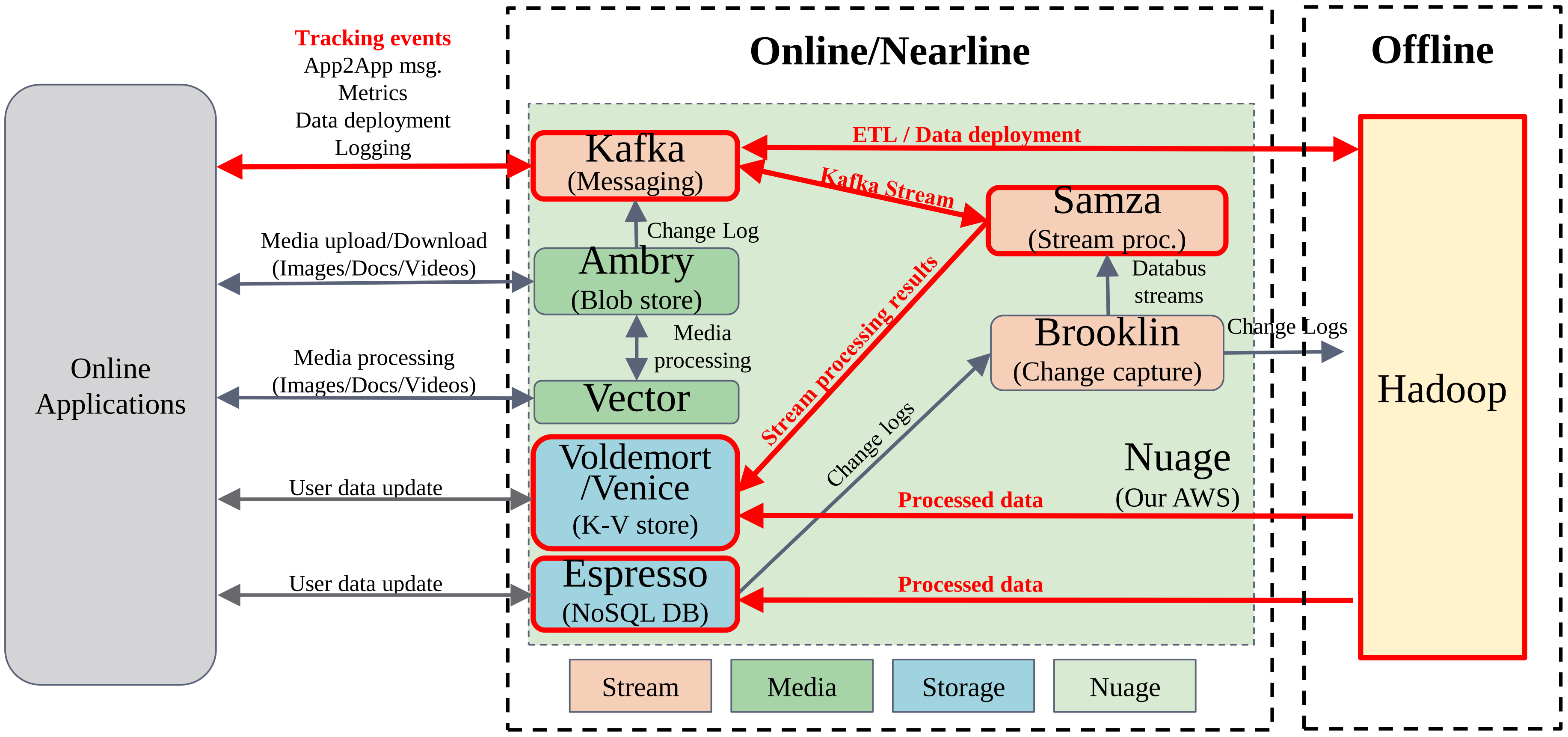
The big picture - Kafka@LinkedIn



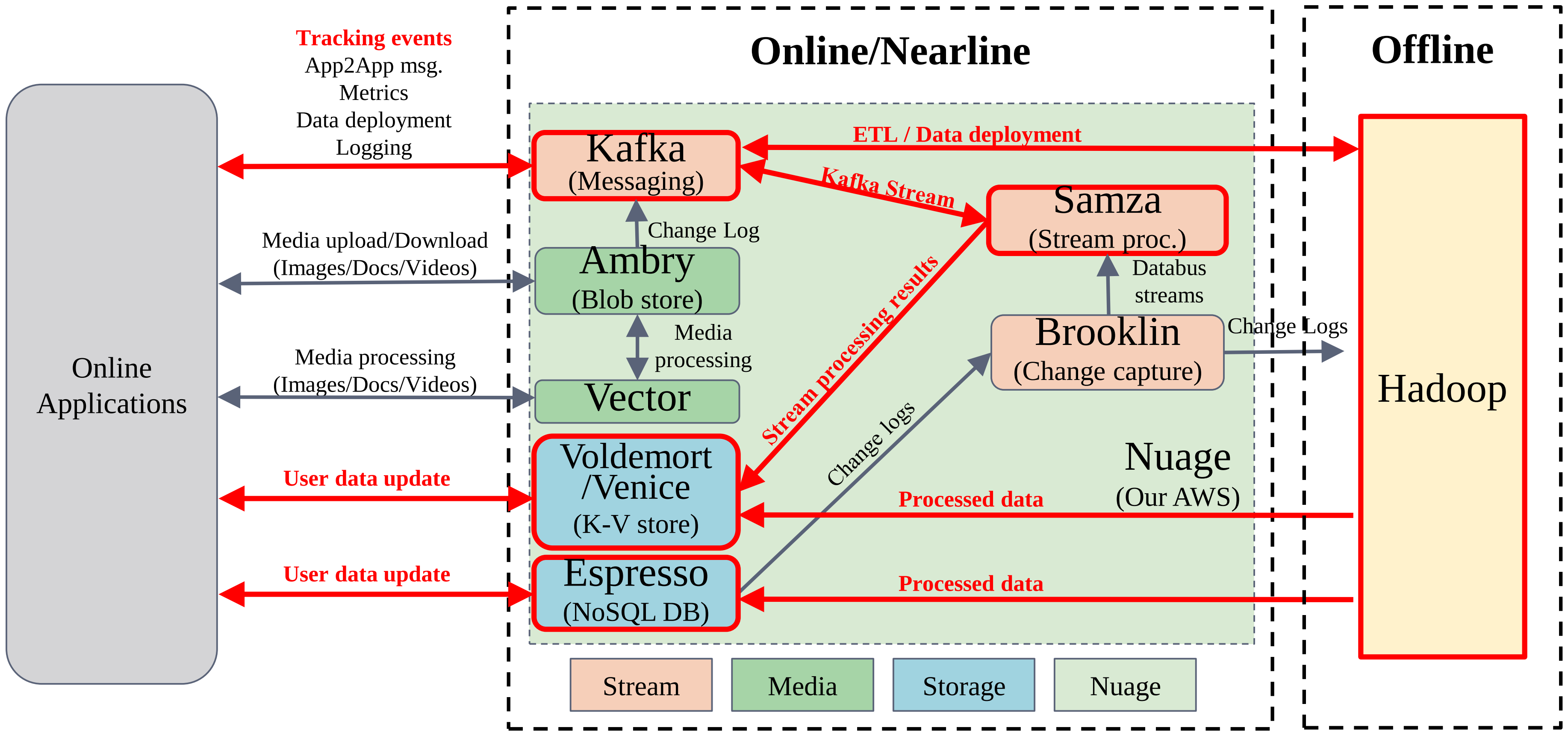
The big picture - Kafka@LinkedIn



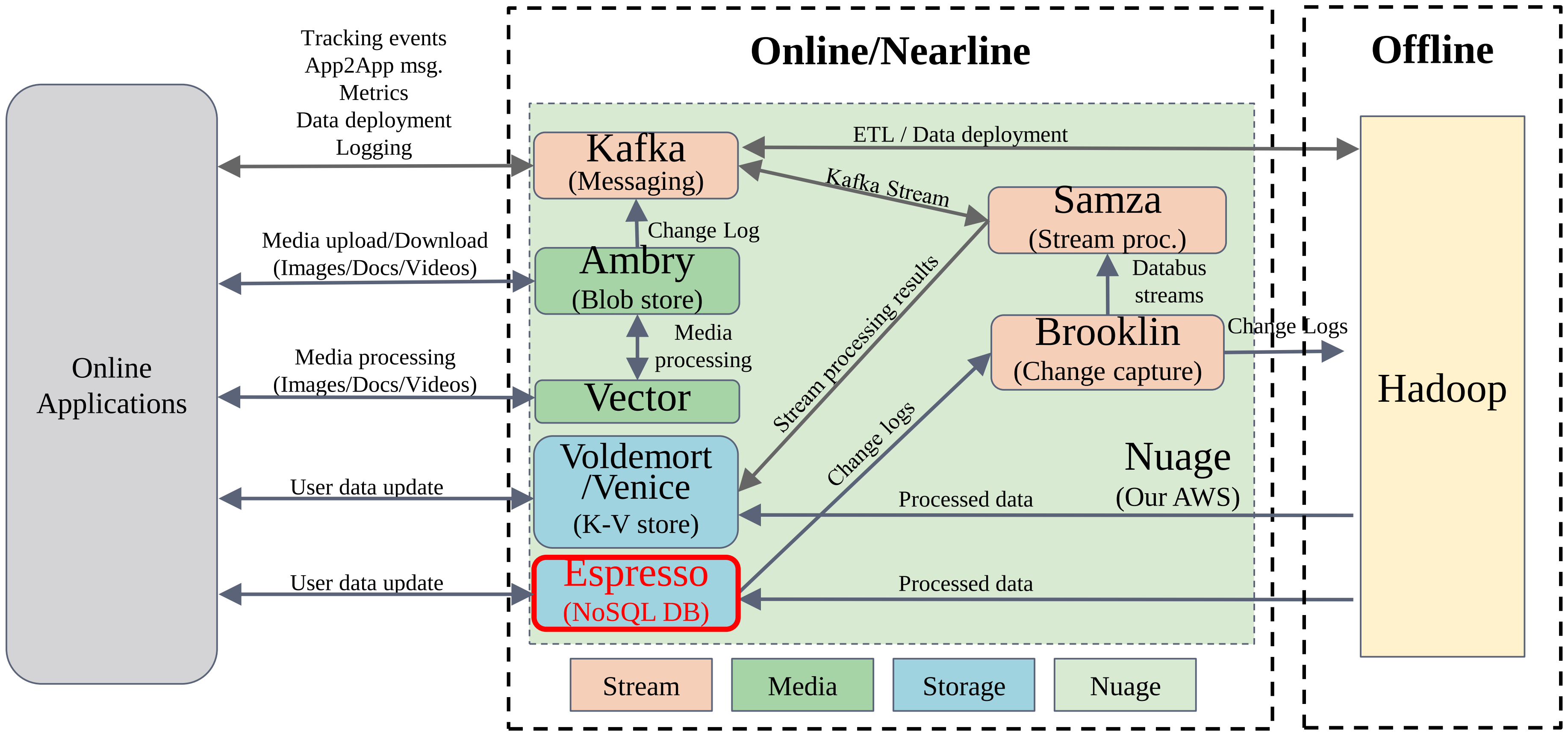
The big picture - Kafka@LinkedIn



The big picture - Kafka@LinkedIn



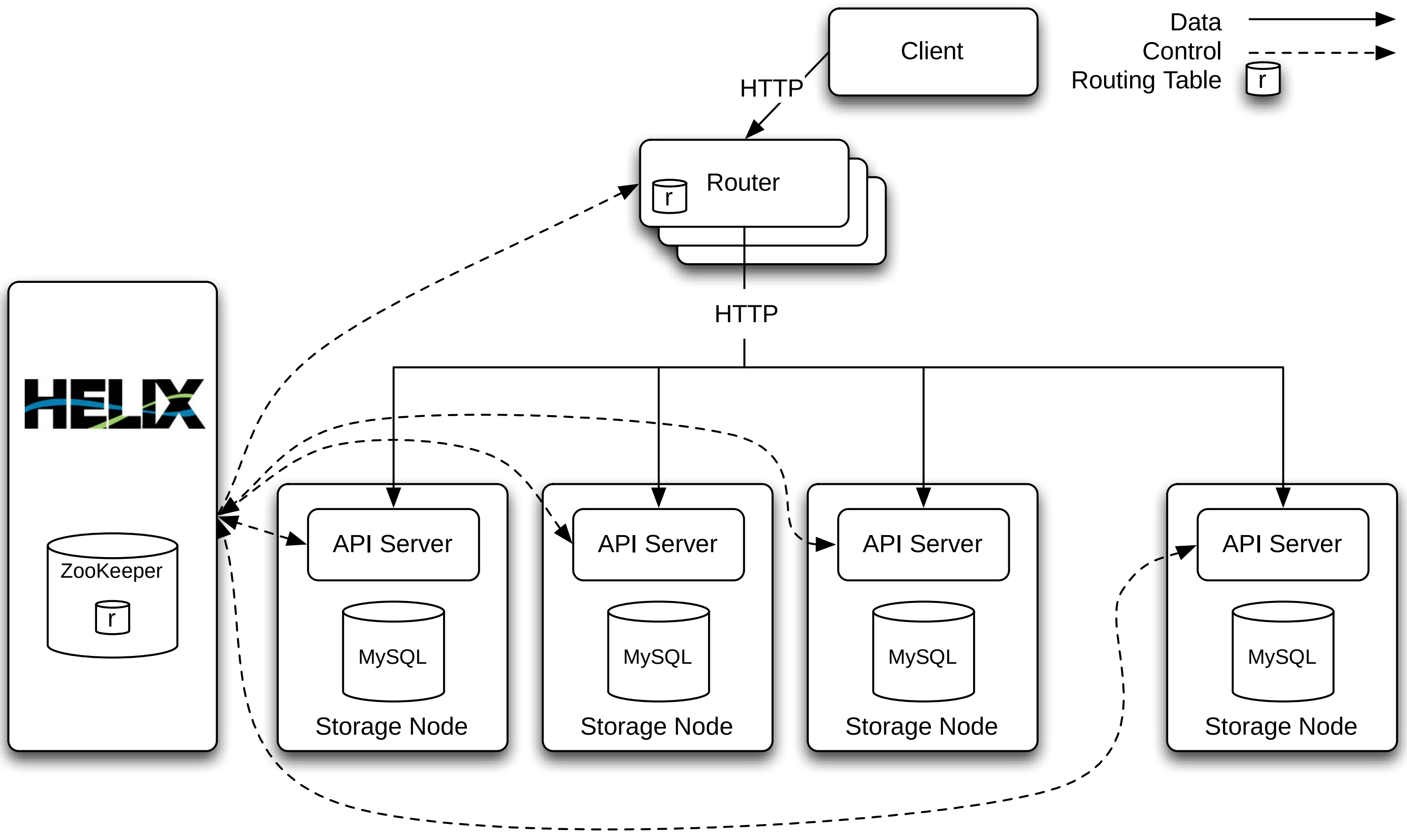
The big picture - Kafka@LinkedIn



Agenda

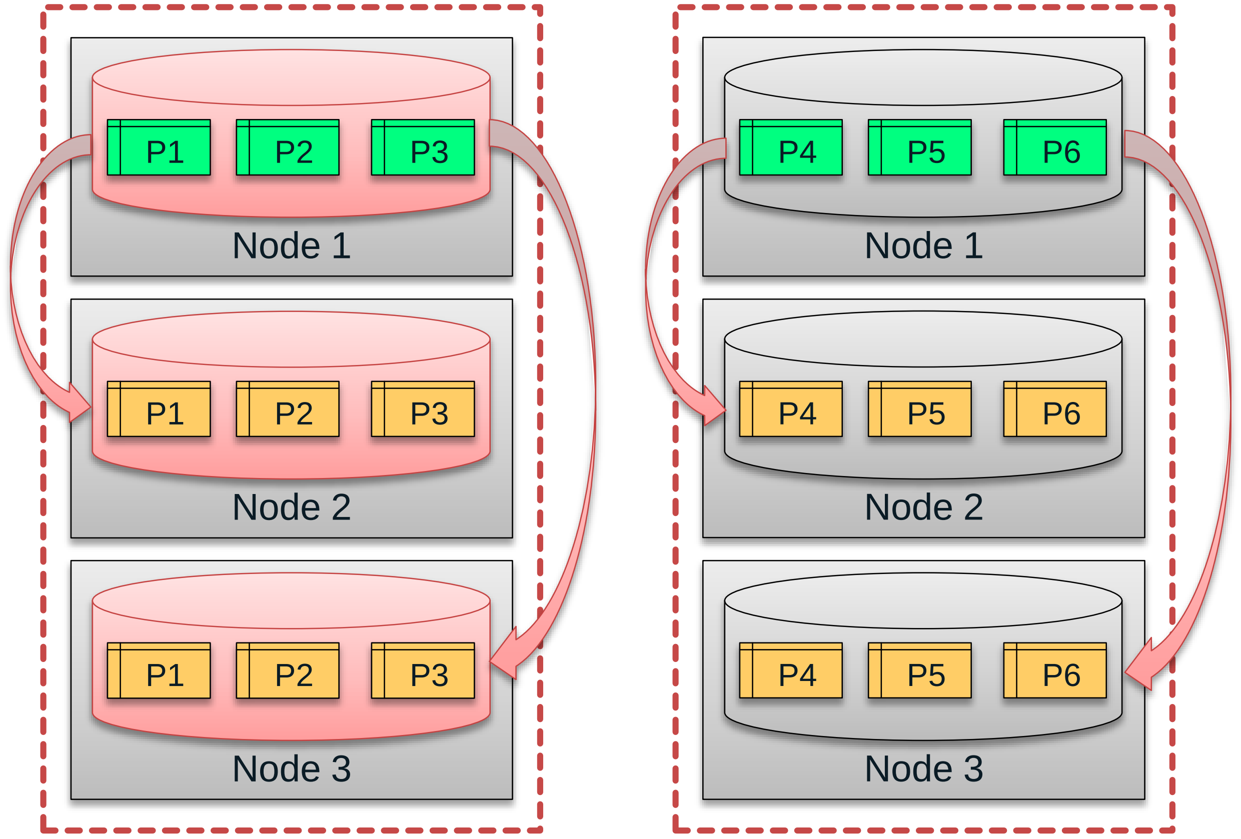
- Introduction to Apache Kafka
- Kafka based replication in Espresso
- Message Integrity guarantees
- Performance
- Large message handling
- Security

Espresso - A scalable NoSQL DB



MySQL Based Replication

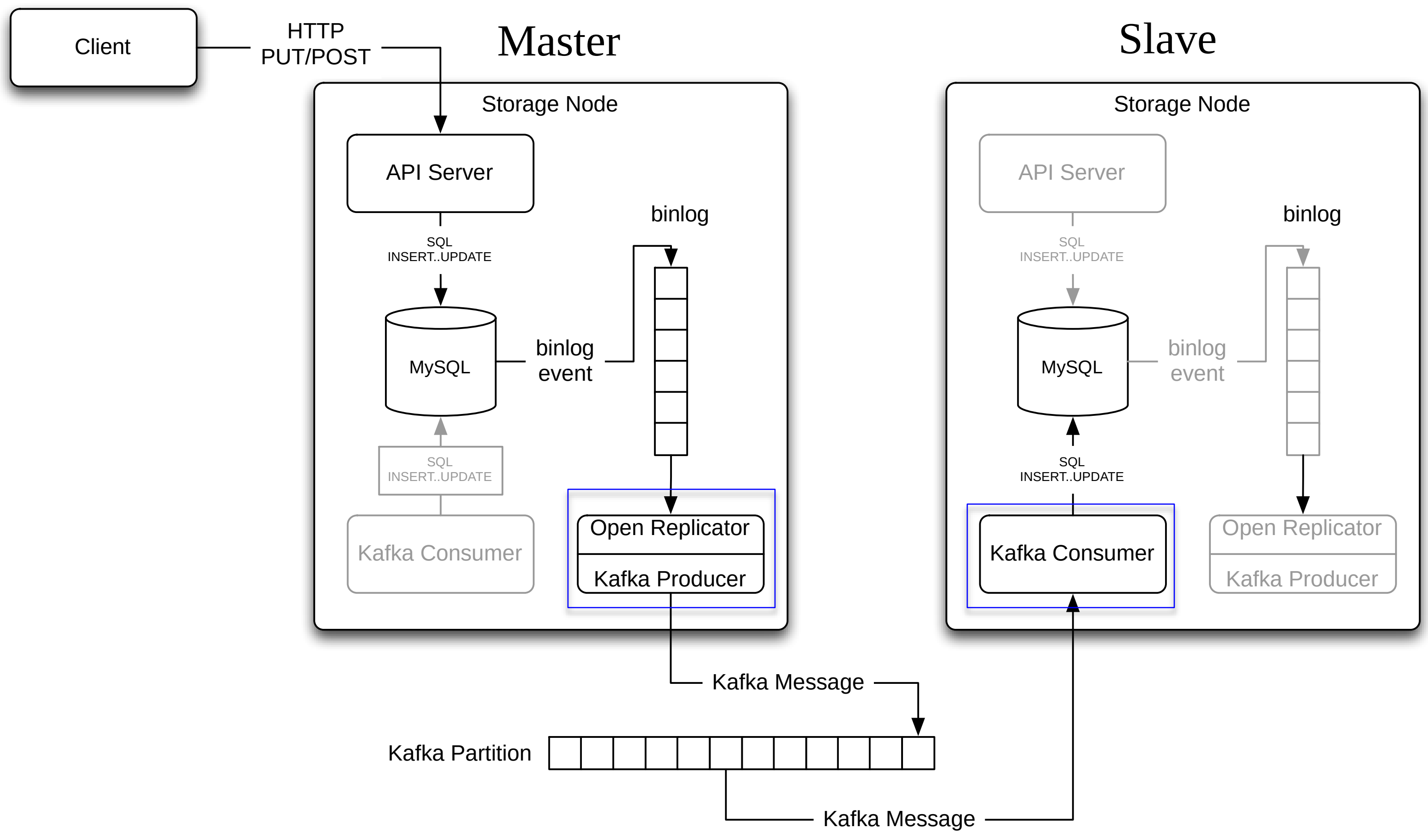
MySQL instance level replication



🟩	Master
🟨	Slave
🟥	Offline

Kafka Based Replication

Kafka based partition level replication



Mission Critical Messaging

- No message loss
- In-order delivery
- Exactly once semantic
- High throughput
- Low latency
- Handle large messages
- Security

Message Integrity

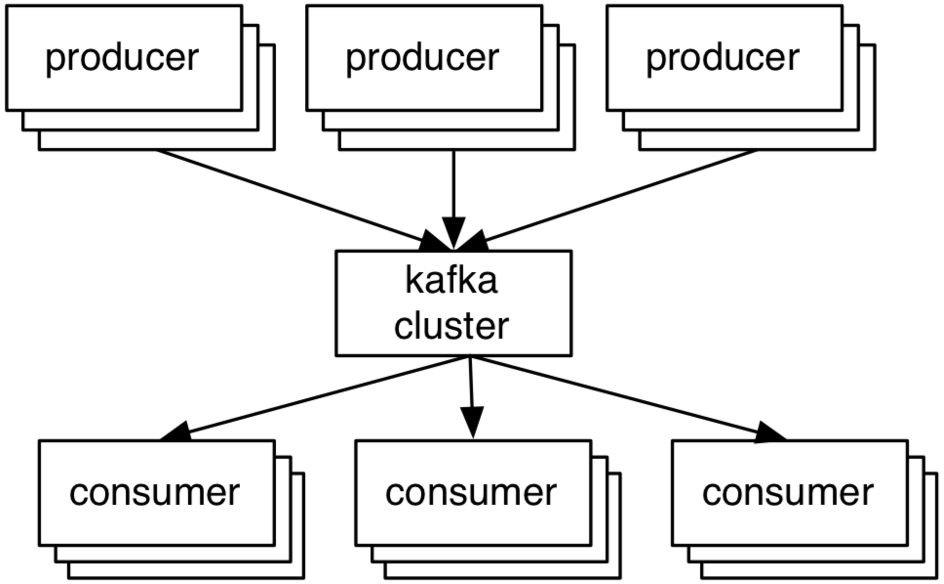
Performance

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

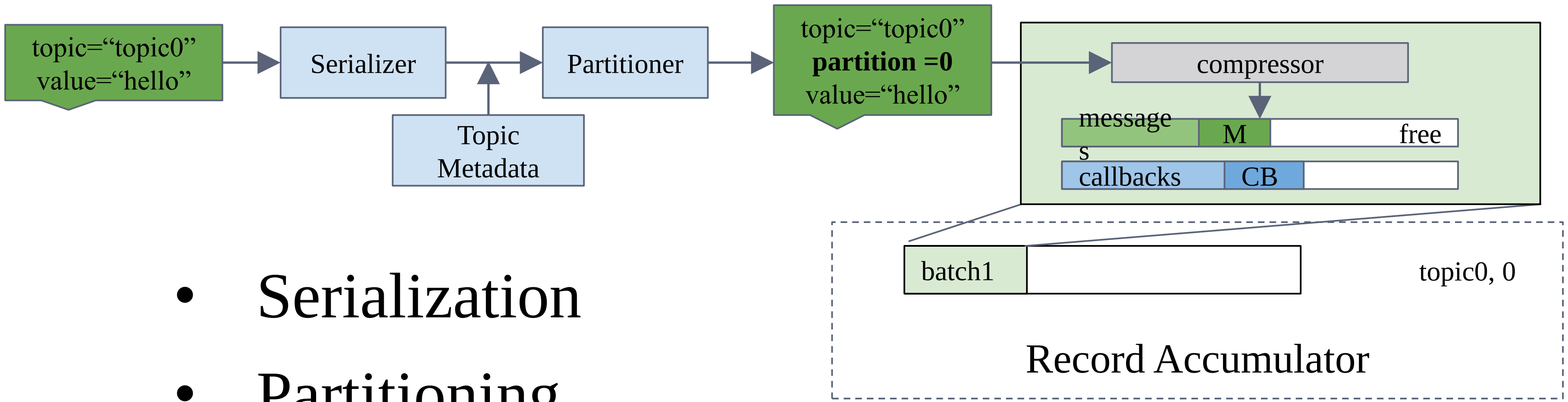
Produce

- 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);
```



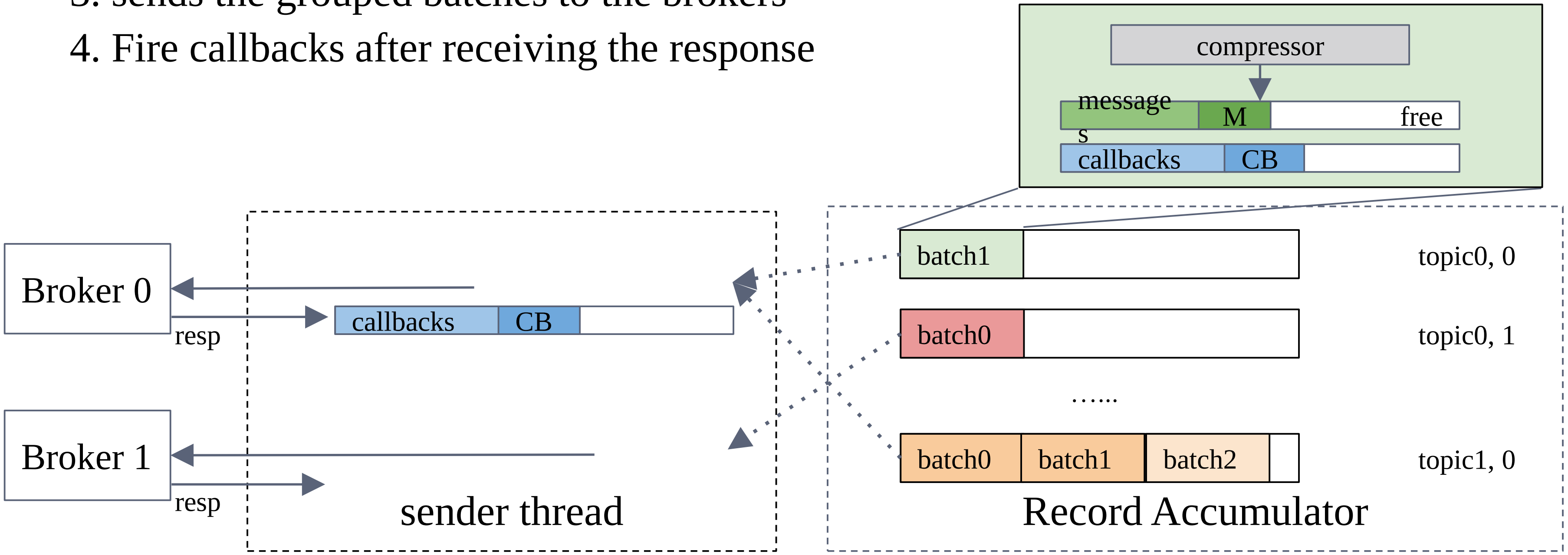
- Serialization
- Partitioning
- Compression

✓ Tasks done by the user thread

KafkaProducer

Sender:

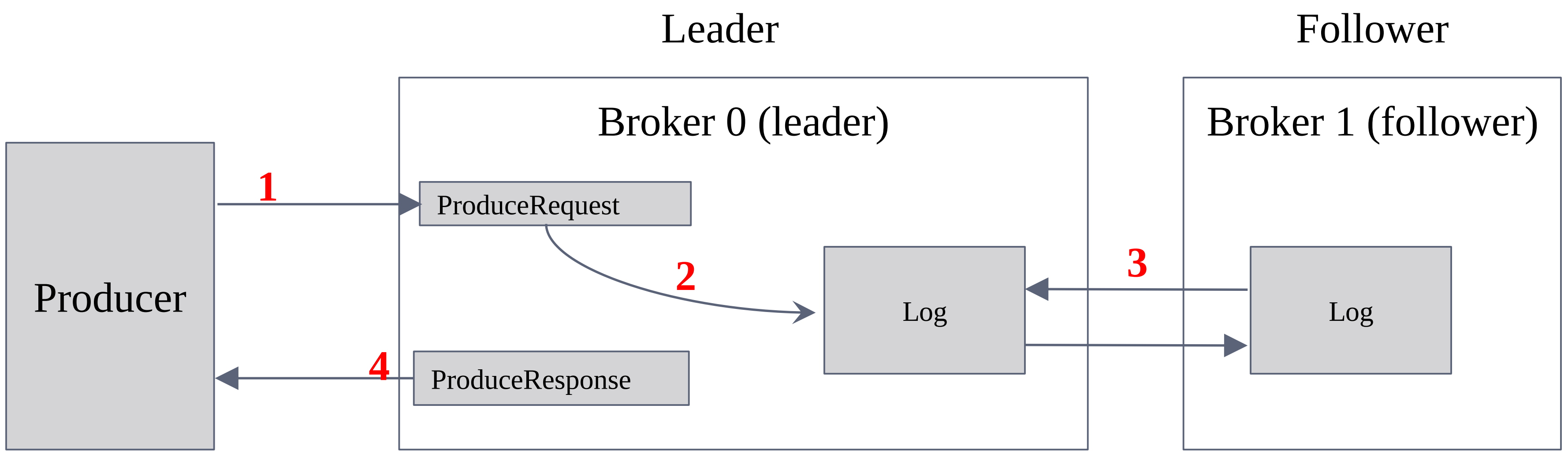
1. **polls** batches from the batch queues (one batch / partition)
2. **groups** batches based on the leader broker
3. sends the grouped batches to the brokers
4. Fire callbacks after receiving the response



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

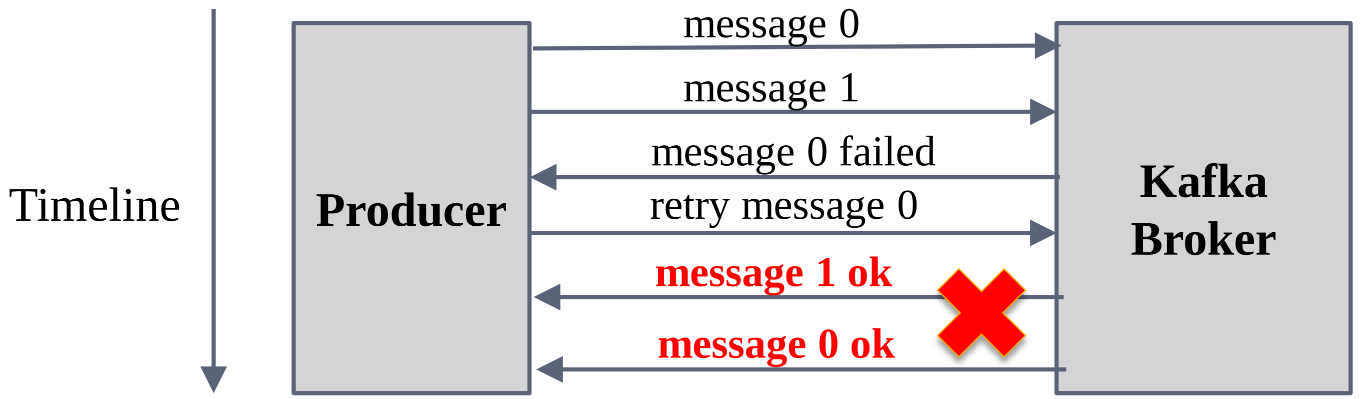
Durability guarantee

- In-Sync-Replica (ISR)
  - A replica that can keep up with the leader
- Semantic for `acks`

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

In order delivery

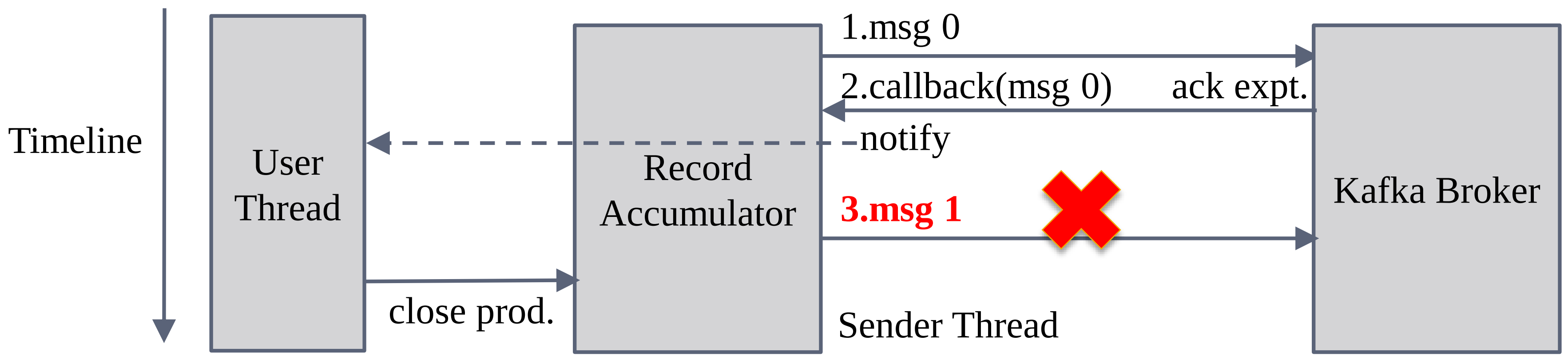
- max.in.flight.requests.per.connection = 1
  - Request pipelining



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

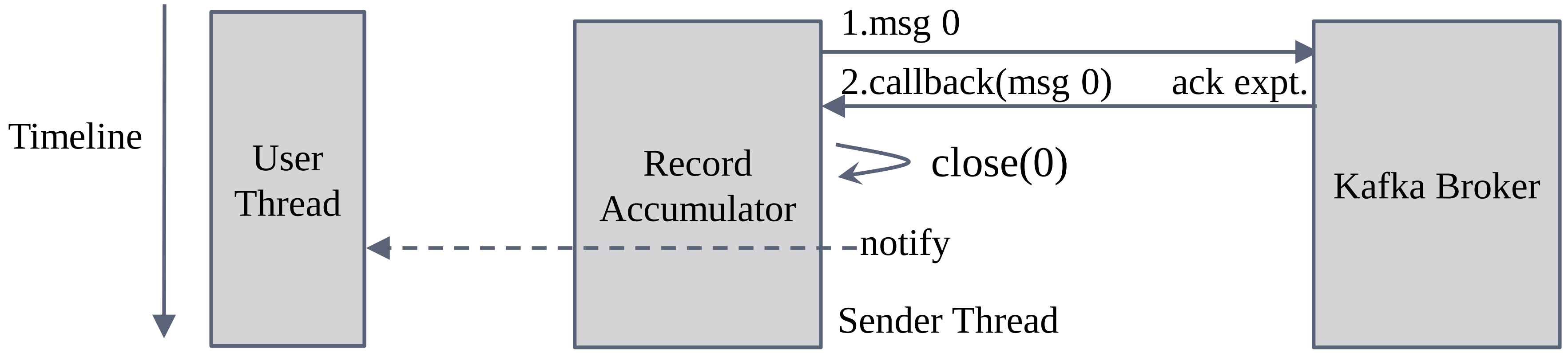
In order delivery

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



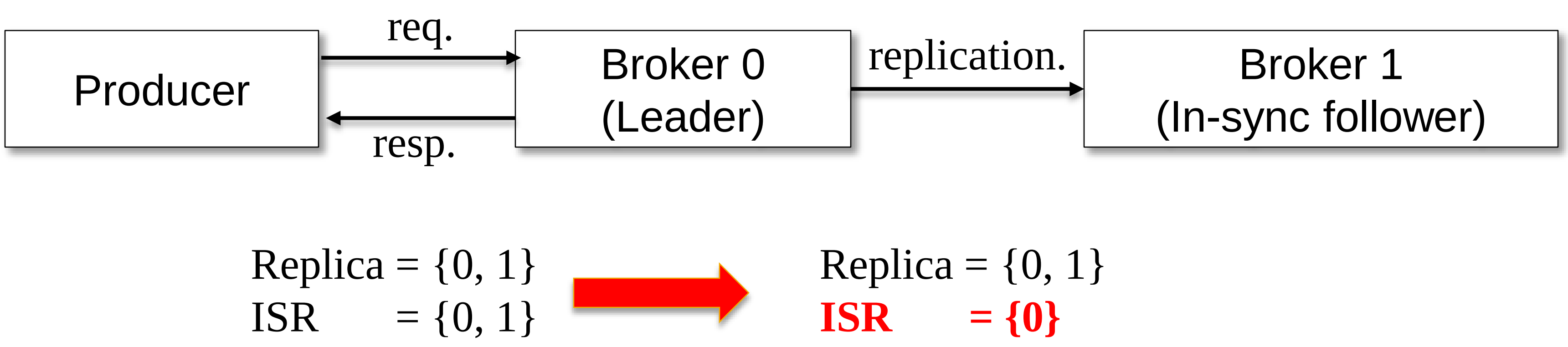
In order delivery

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



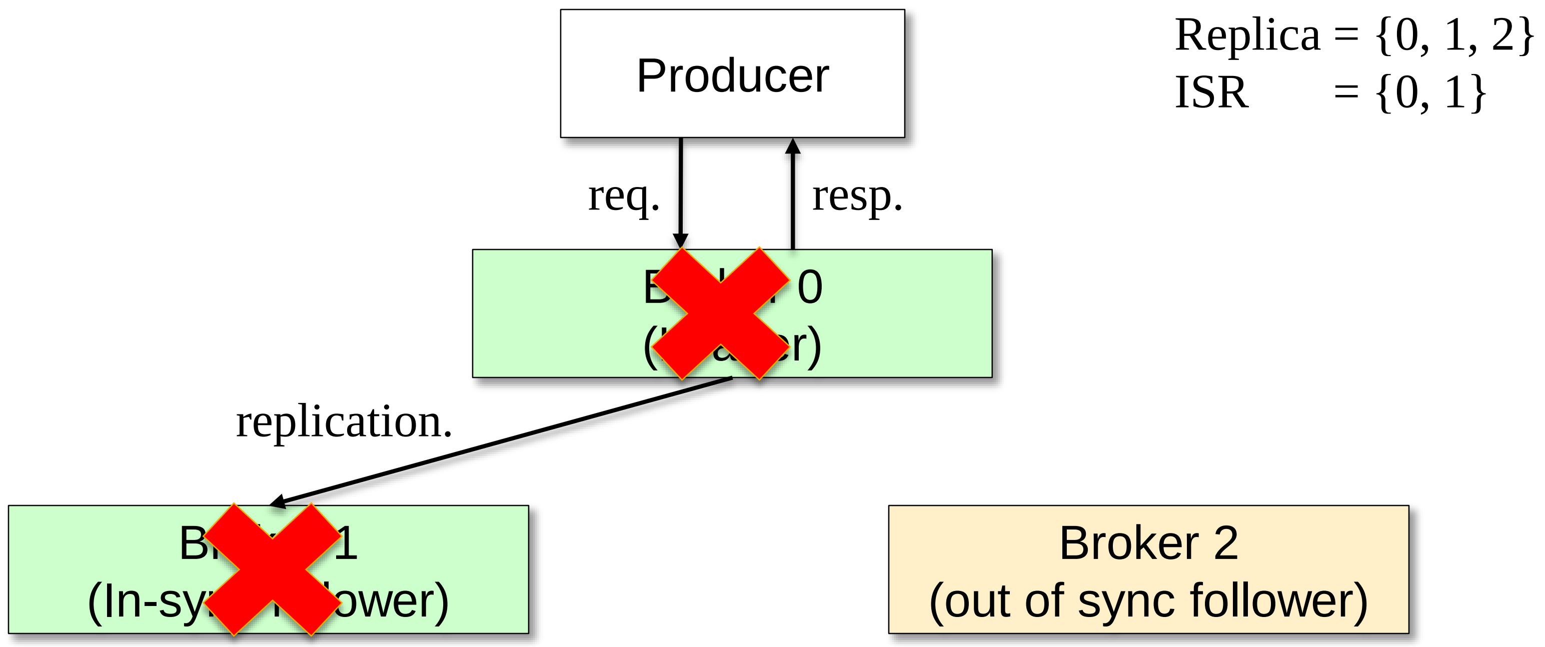
Broker

- min.isr=2
  - If acks=all, at least 2 copies of messages are required on the broker
  - replication.factor needs be 3 to tolerate single broker failure



Broker

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

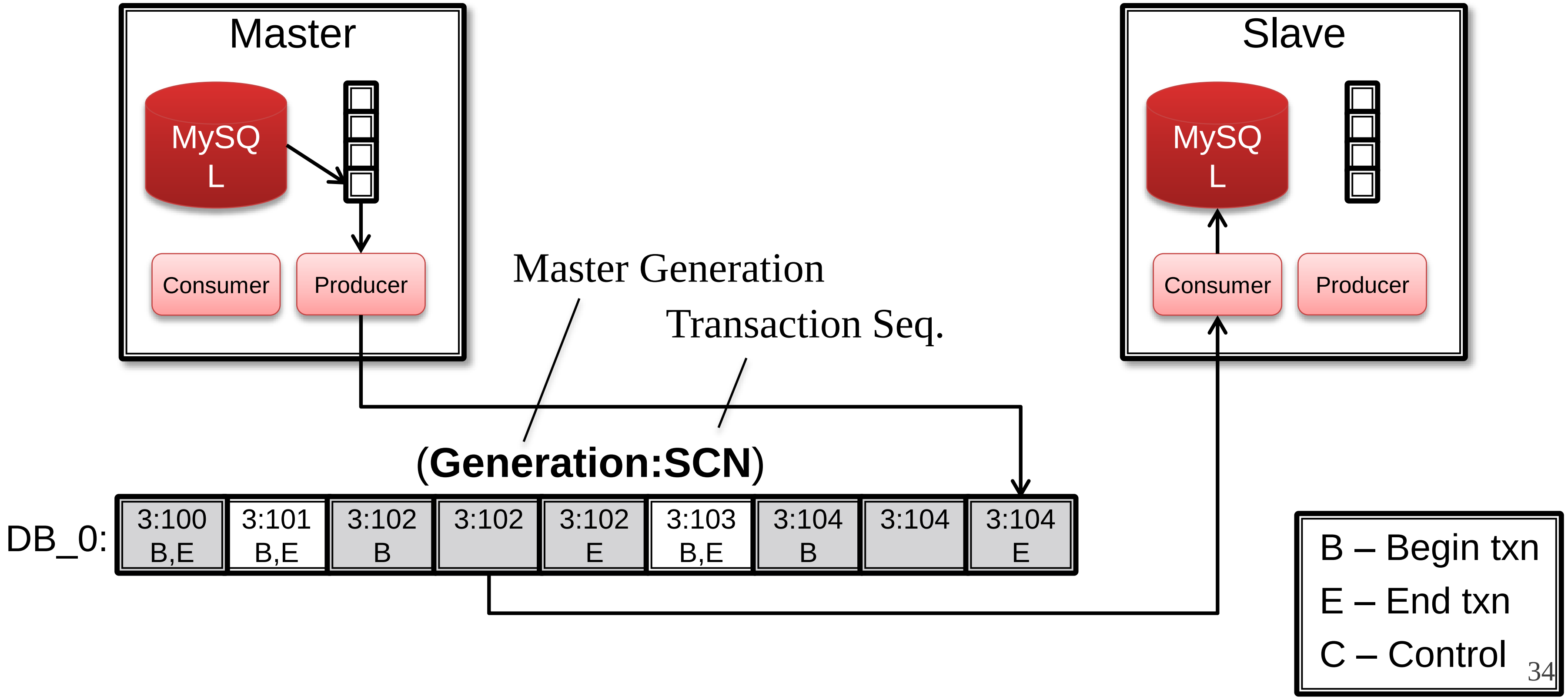


Consumer

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

Consumer

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



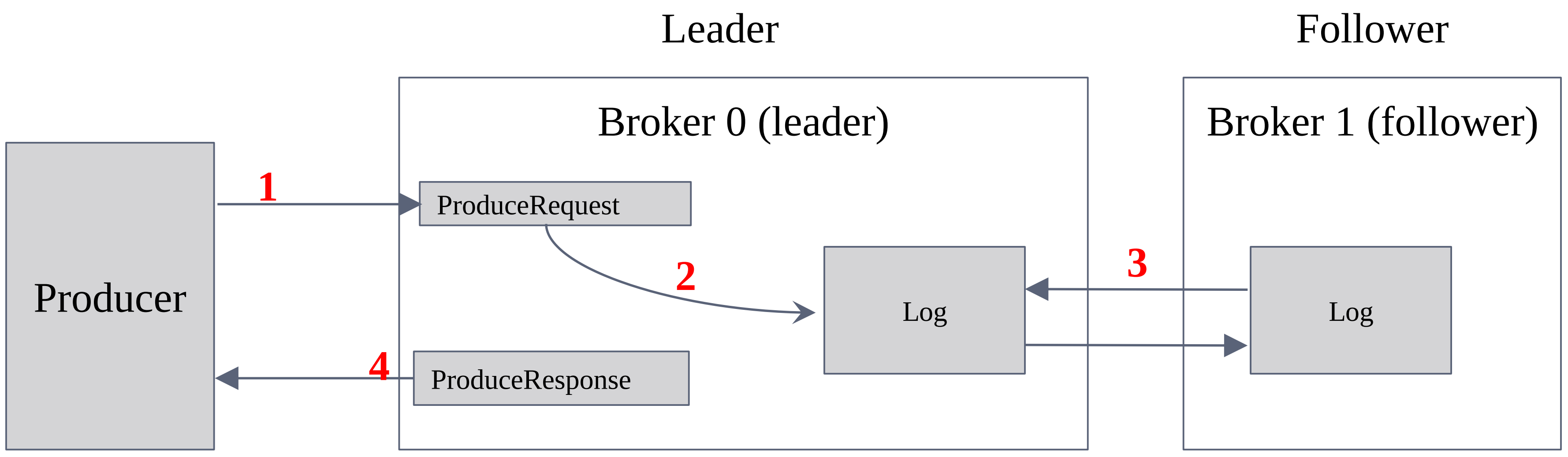
Agenda

- 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)

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

Latency when acks=all

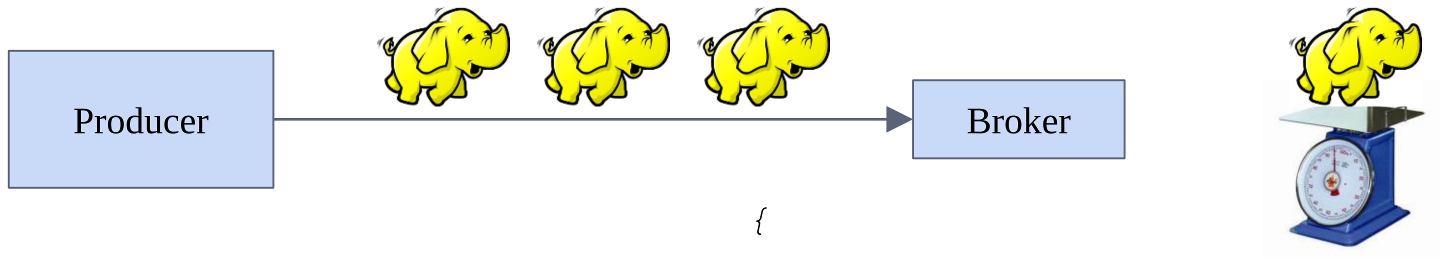
- Kafka replication is a pull model
- Increase **num.replica.fetchers**
  - Parallel fetch
- Not perfect solution
  - Diminishing effect (1/N)
  - Scalability concern
    - Replica fetchers per broker = (Cluster_Size - 1) * num.replica.fetchers

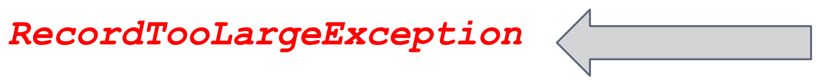
Agenda

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

What is a large message

- Kafka has a limit on the maximum size of a single message
  - Enforced on the compressed wrapper message if compression is used





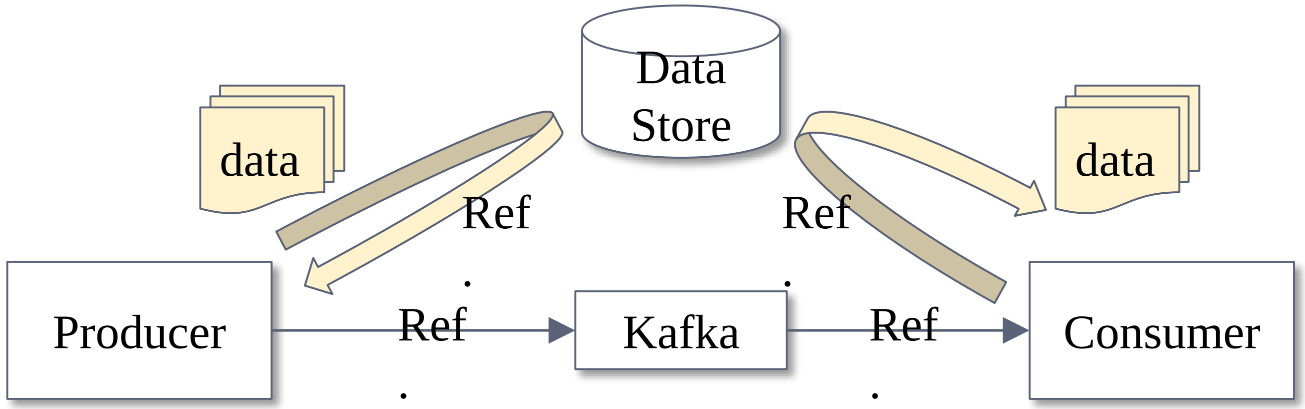
```
{
   …
   if (message.size > message.max.bytes)
         reject!
   …
}
```

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

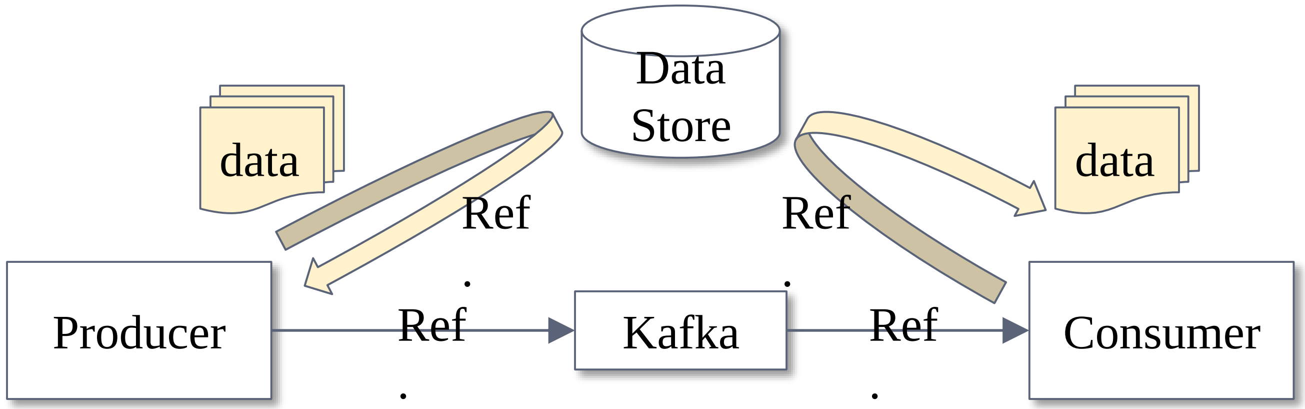
- Reference based messaging



What works

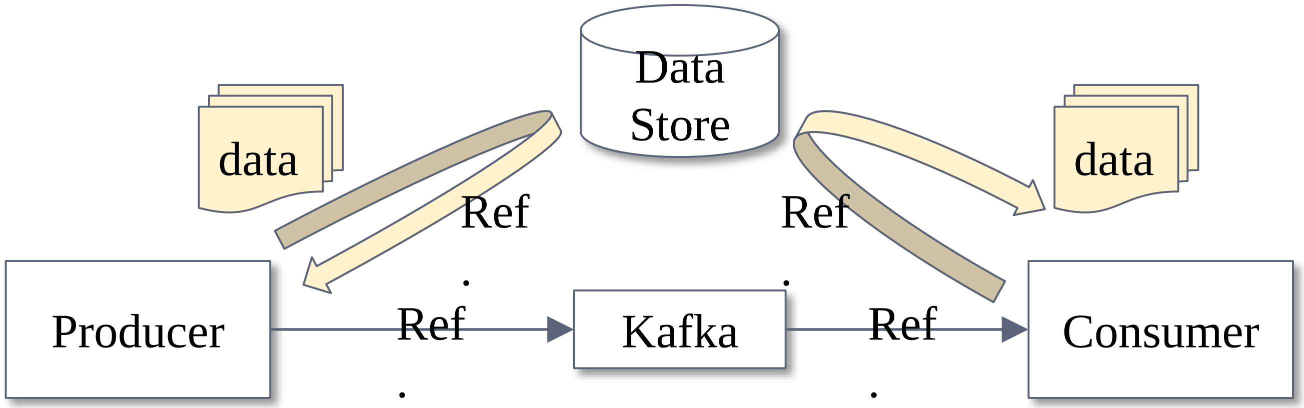
- Unknown maximum row size
- Strict no data loss
- Strict message order guarantee

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

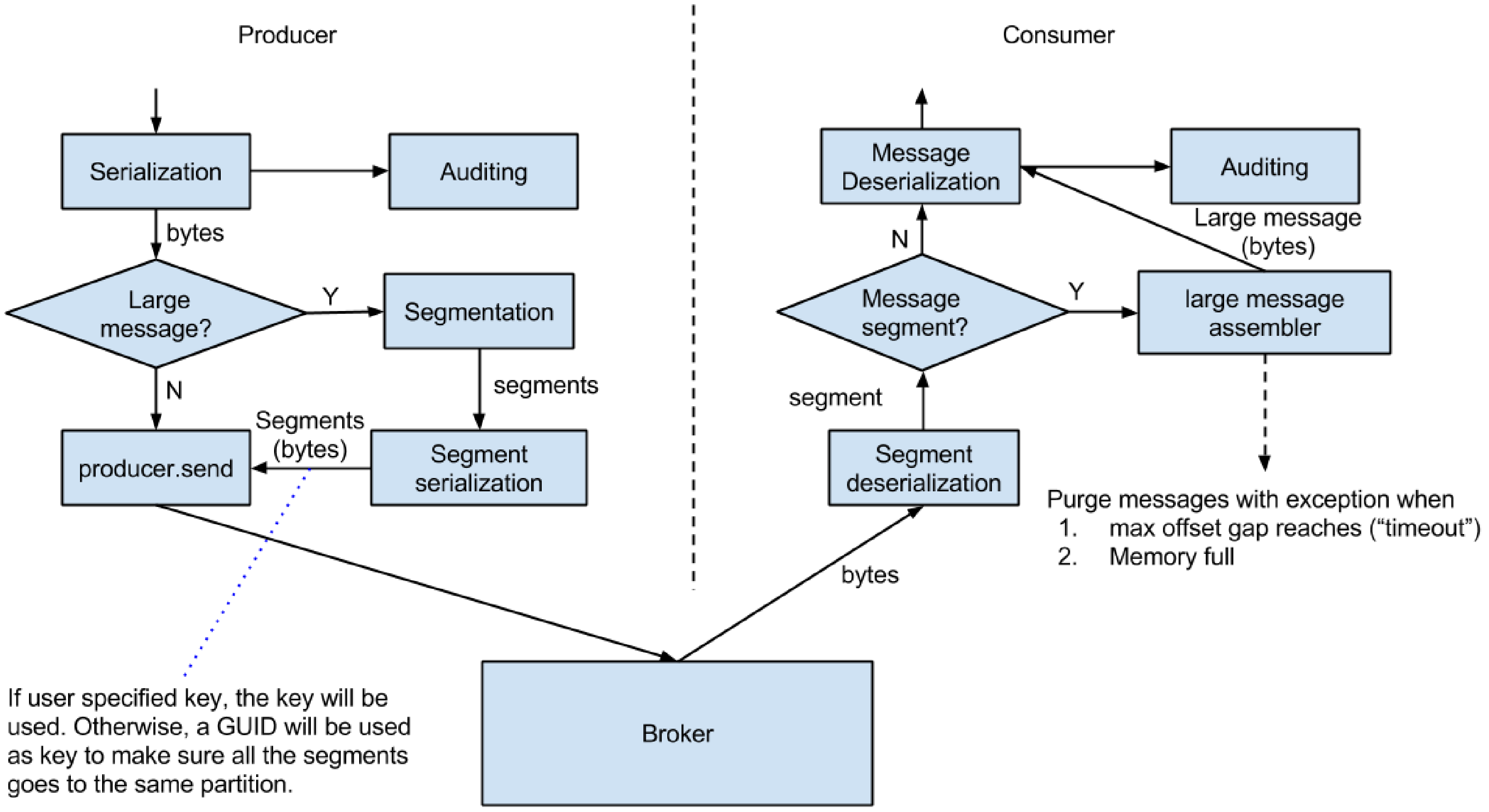


What does not work

- 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



In-line large message handling



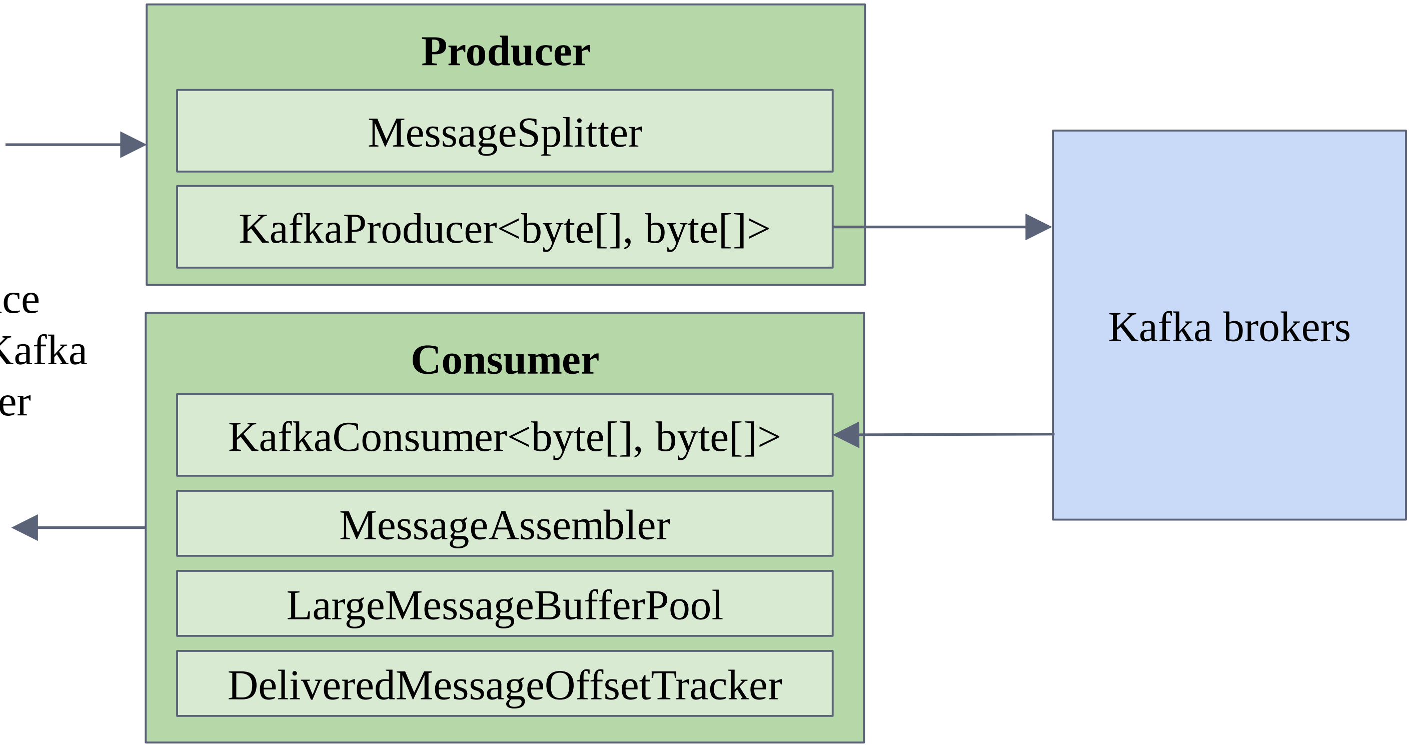
Purge messages with exception when
1. max offset gap reaches ("timeout")
2. Memory full

If user specified key, the key will be used. Otherwise, a GUID will be used as key to make sure all the segments goes to the same partition.

In-line large message support

	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

In-line large message support


Producer
MessageSplitter
KafkaProducer<byte[], byte[]>
Kafka brokers
Consumer
KafkaConsumer<byte[], byte[]>
MessageAssembler
LargeMessageBufferPool
DeliveredMessageOffsetTracker

Compatible interface with open source Kafka producer / consumer

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

Some details

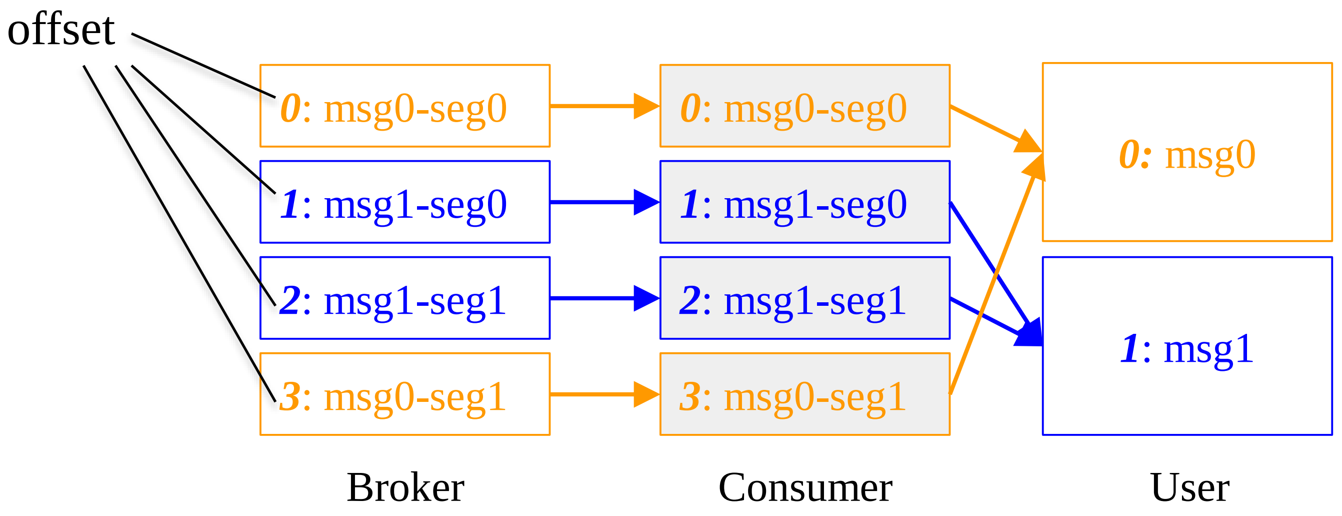
- Many interesting details
  - The offset of a large message
  - Offset Tracking
  - Rebalance and duplicates handling
  - Producer callback
  - Memory management
  - Performance overhead
  - Compatibility with existing messages

Offset of a large message

- 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

The offset of a large message

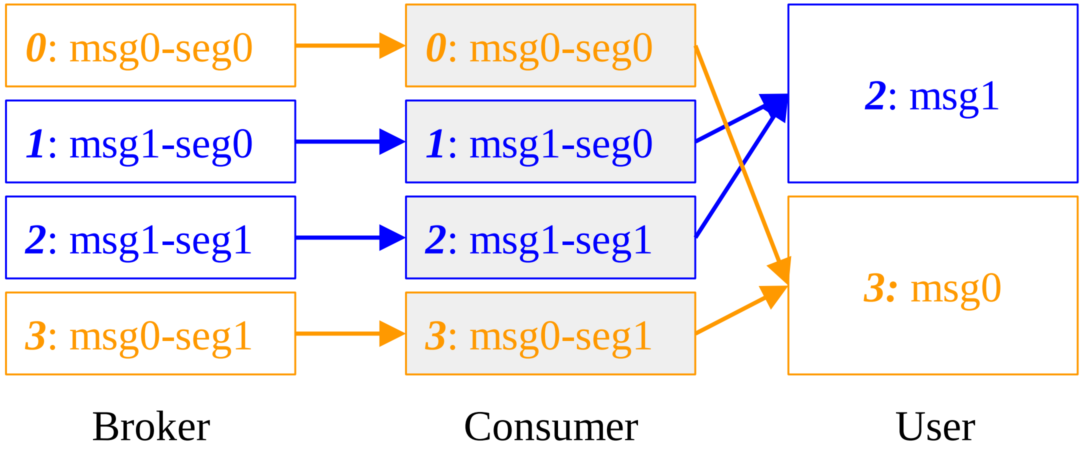
- offset of a large message = offset of **first** segment
    - First seen first serve
    - Expensive for in-order delivery



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()



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

- 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

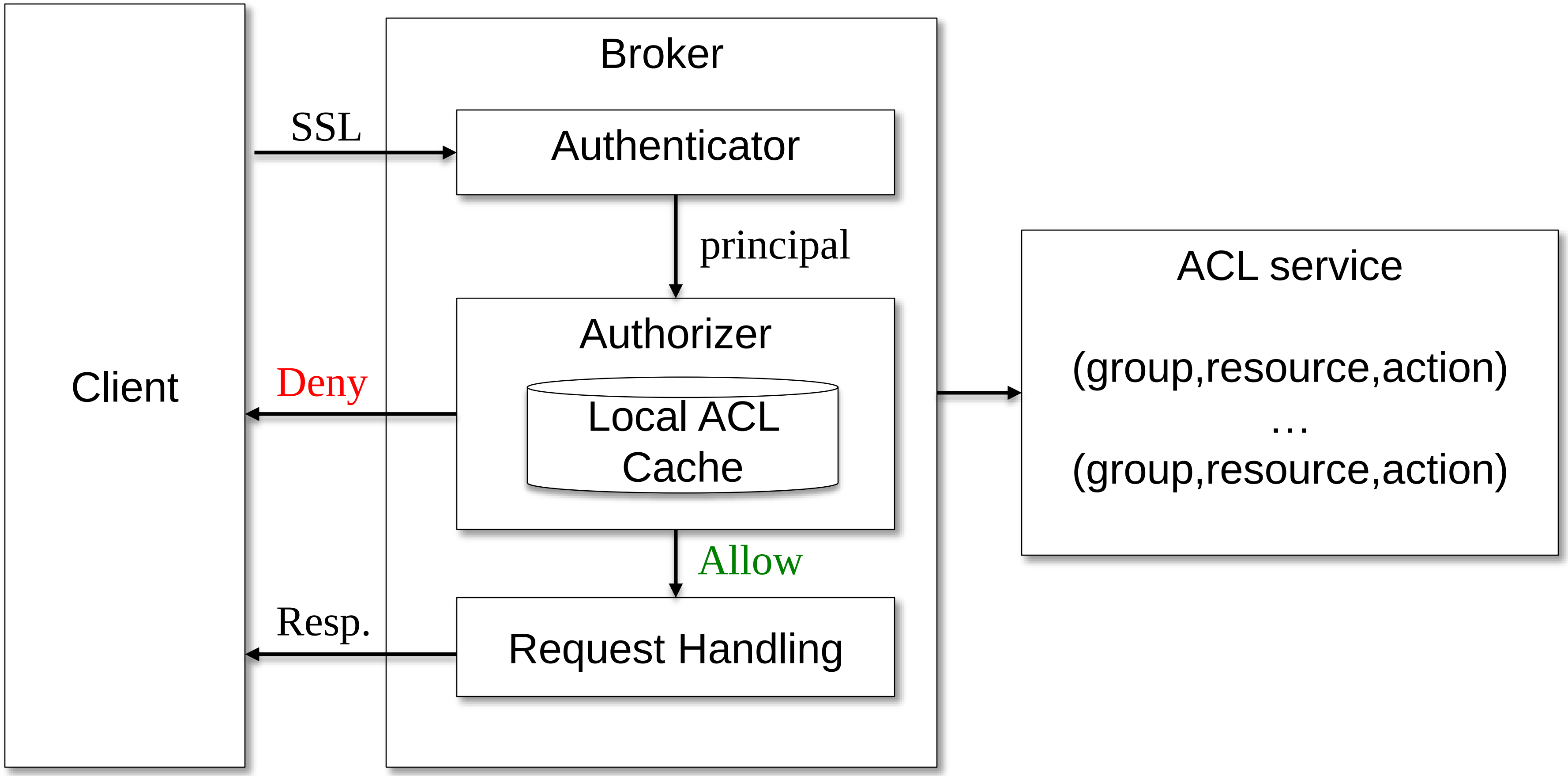
- Introduction to Apache Kafka
- Kafka based replication in Espresso
- Message Integrity guarantees
- Performance
- Large message handling
- Security

Security in Kafka

- Authentication (SSL, Kerberos, SASL)
- Authorization (Unix-like permission)
  - **Resource**: Cluster, Topic, Group
  - **Operation**: Read, Write, Create, Delete, Alter, Describe, Cluster Operation, All
- TLS encryption

Our solution

Authorizer performance is important



Q & A