Data ingestion

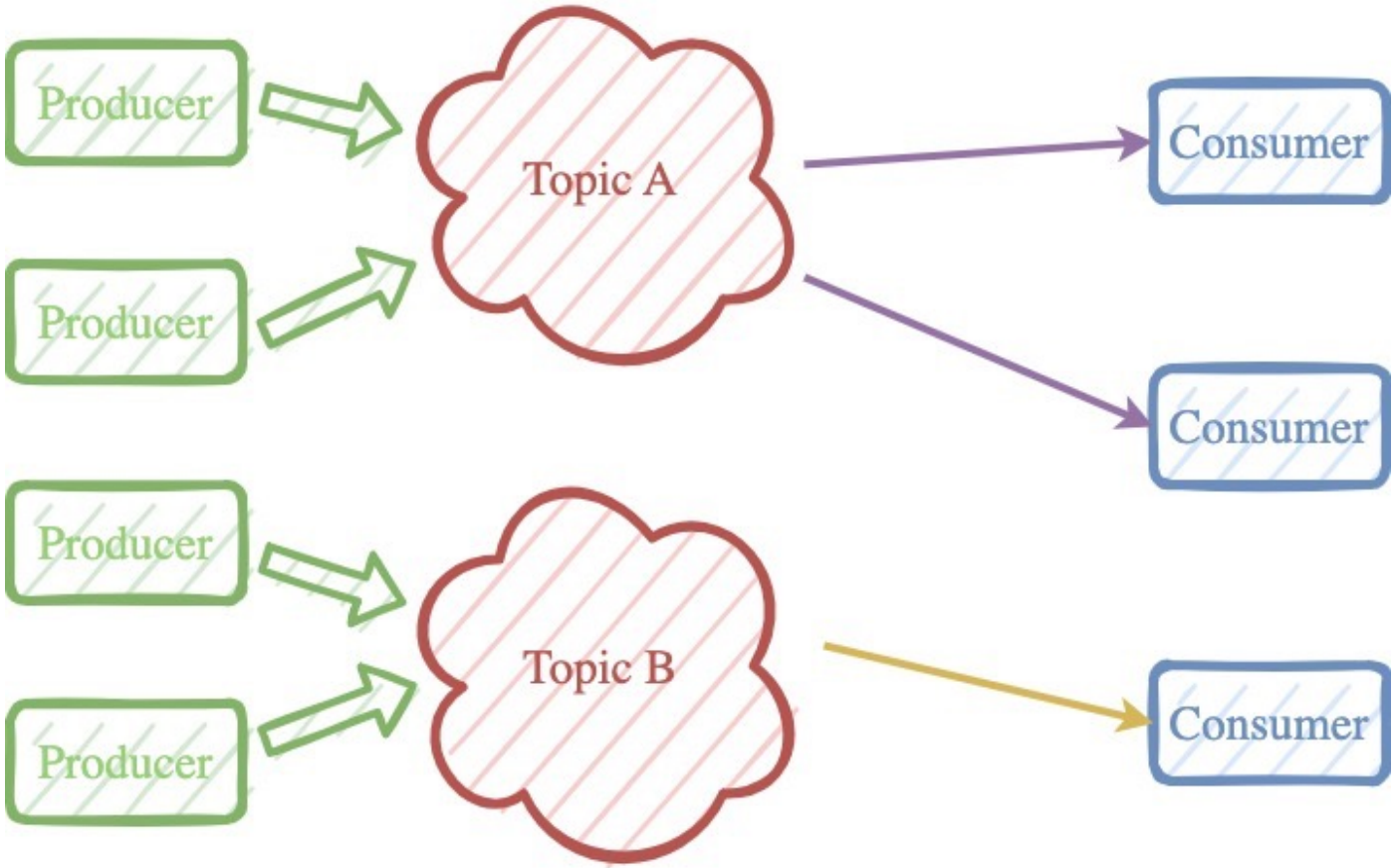
Apache Kafka

Introduction

- Kafka is an event streaming platform
  - Publish-subscribe model
  - Push/pull mode
- It allows for
  - Publishing and reading events
  - Storing streams reliably
  - Processing streams
- It’s usually deployed on a cluster
  - Scale-out elasticity
  - Fault tolerance

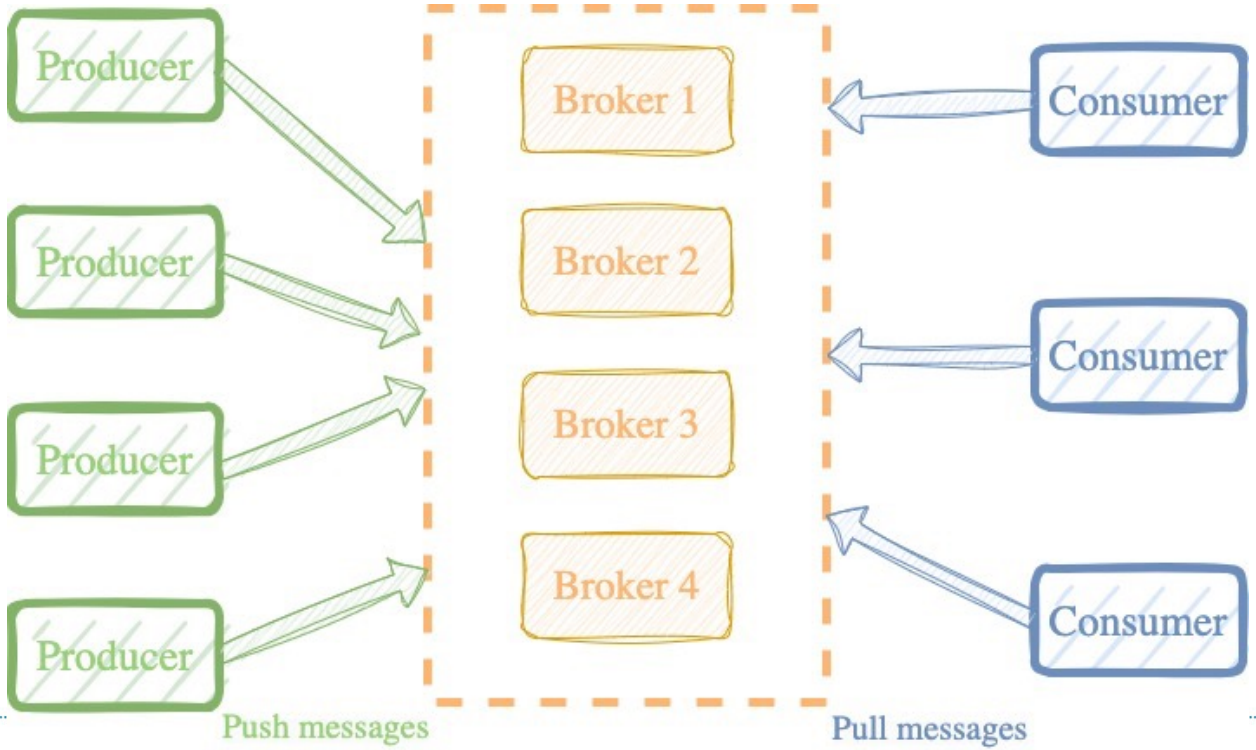
Concepts

- Events
  - Key, value, timestamp and optional metadata
  - Stored as **topics**
  - Cannot be modified
- Producers
  - Applications which publish (produce) events
- Consumers
  - Applications which subscribe (read) events
- No link between Producer and Consumers
  - Time and space decoupling


Producer
Producer
Producer
Producer
Topic A
Topic B
Consumer
Consumer
Consumer

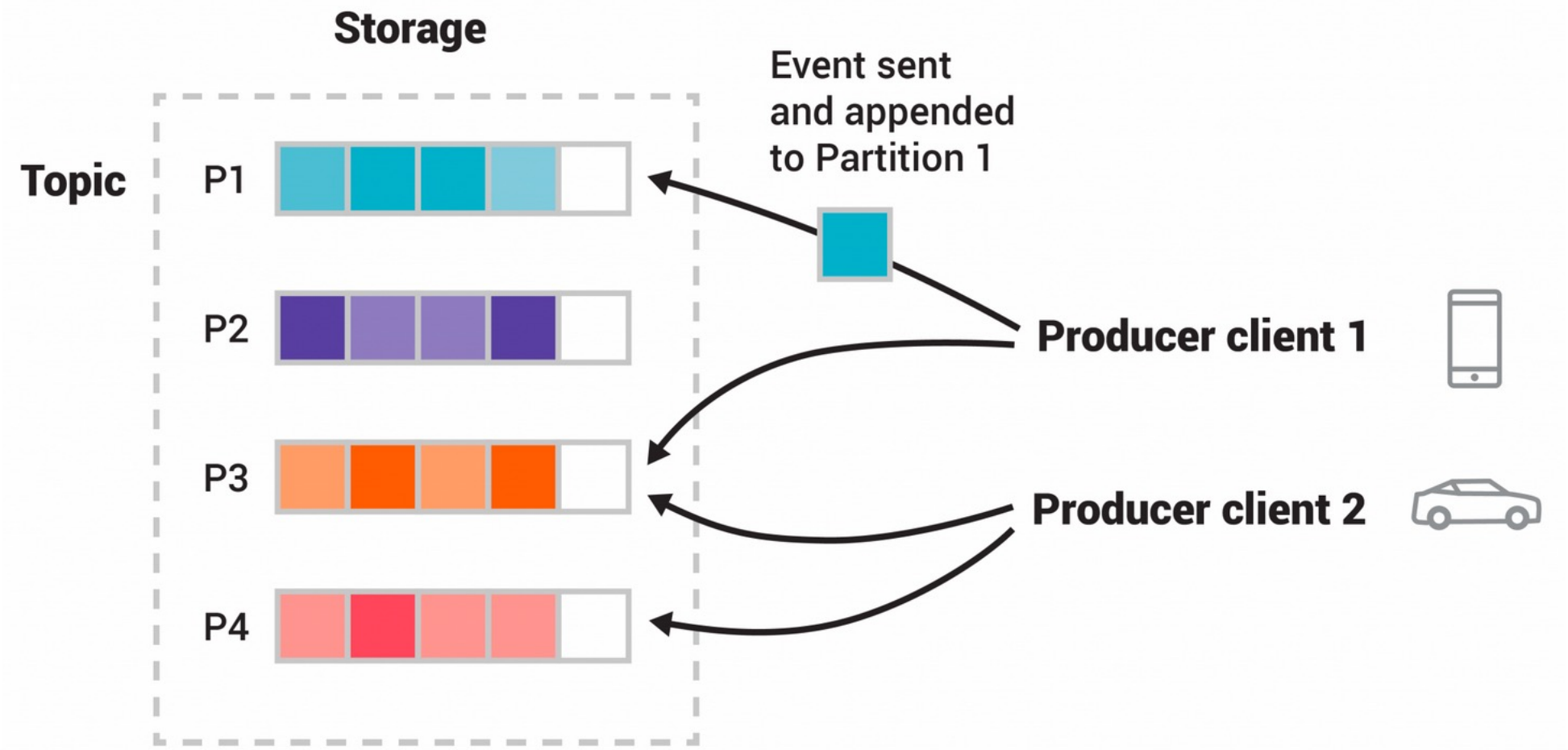
Architecture

- Kafka can run on
  - A single machine
  - A cluster
  - Multiple clusters
- Brokers
  - Core servers
  - Used for storing data
- Clients
  - Anything that connects to Kafka
  - Producer or Consumer


Producer
Producer
Producer
Producer
Broker 1
Broker 2
Broker 3
Broker 4
Consumer
Consumer
Consumer
Push messages
Pull messages

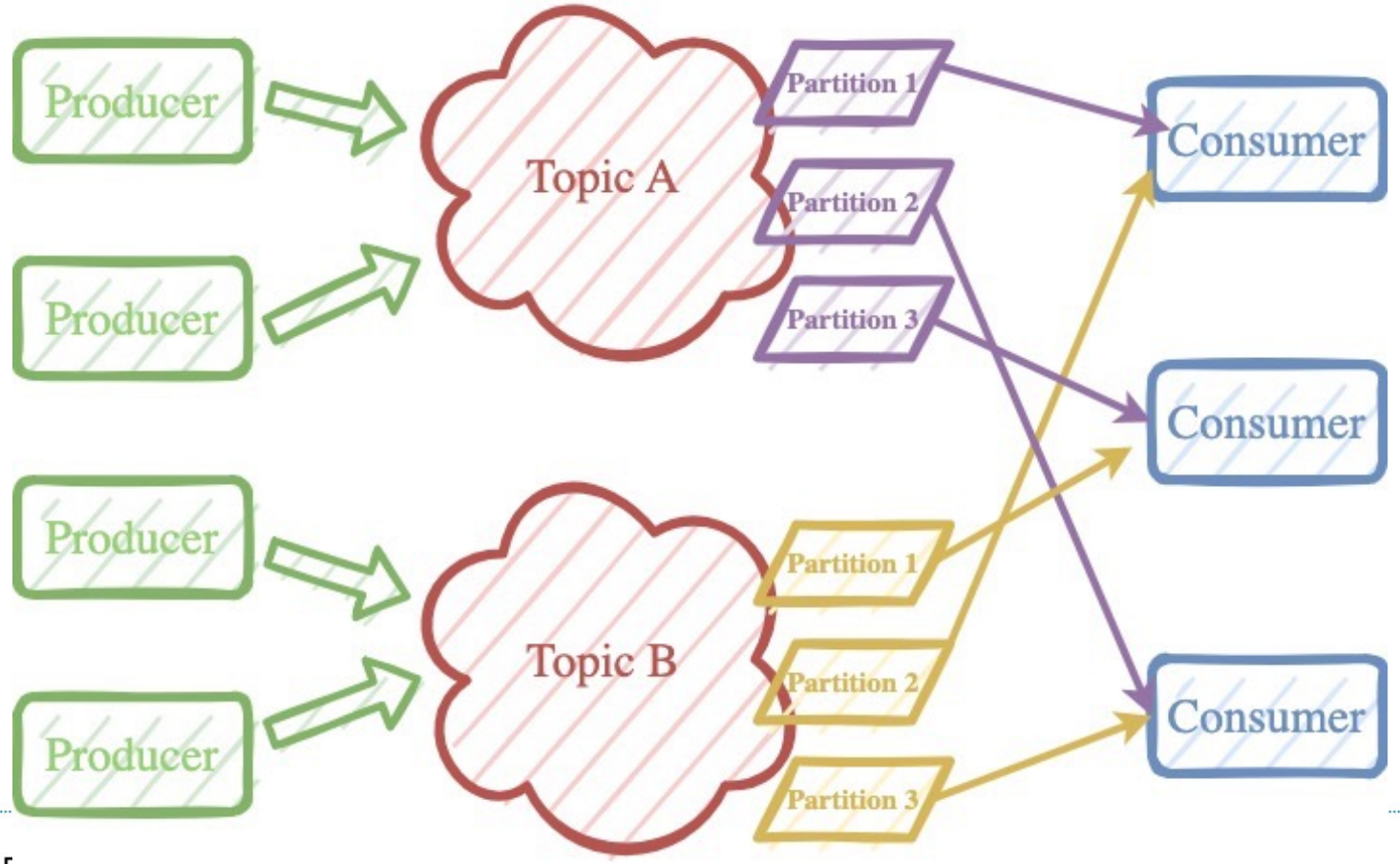
Topics

- Multi-producer and multi-subscriber
  - 0, 1 or more producer and subscriber
- Events
  - Appended to a topic
  - Not discarded after reading
    - Per-topic retention policy
  - Have a unique record ID (offset) in their partition
- Topics are partitioned
  - Based on the event's key (if provided)
  - New partitions created when needed
  - Partition are distributed, increasing parallelism
- Replication factor set per topic



UNIVERSITÉ CÔTE D'AZUR

https://kafka.apache.org/documentation/#intro_topics

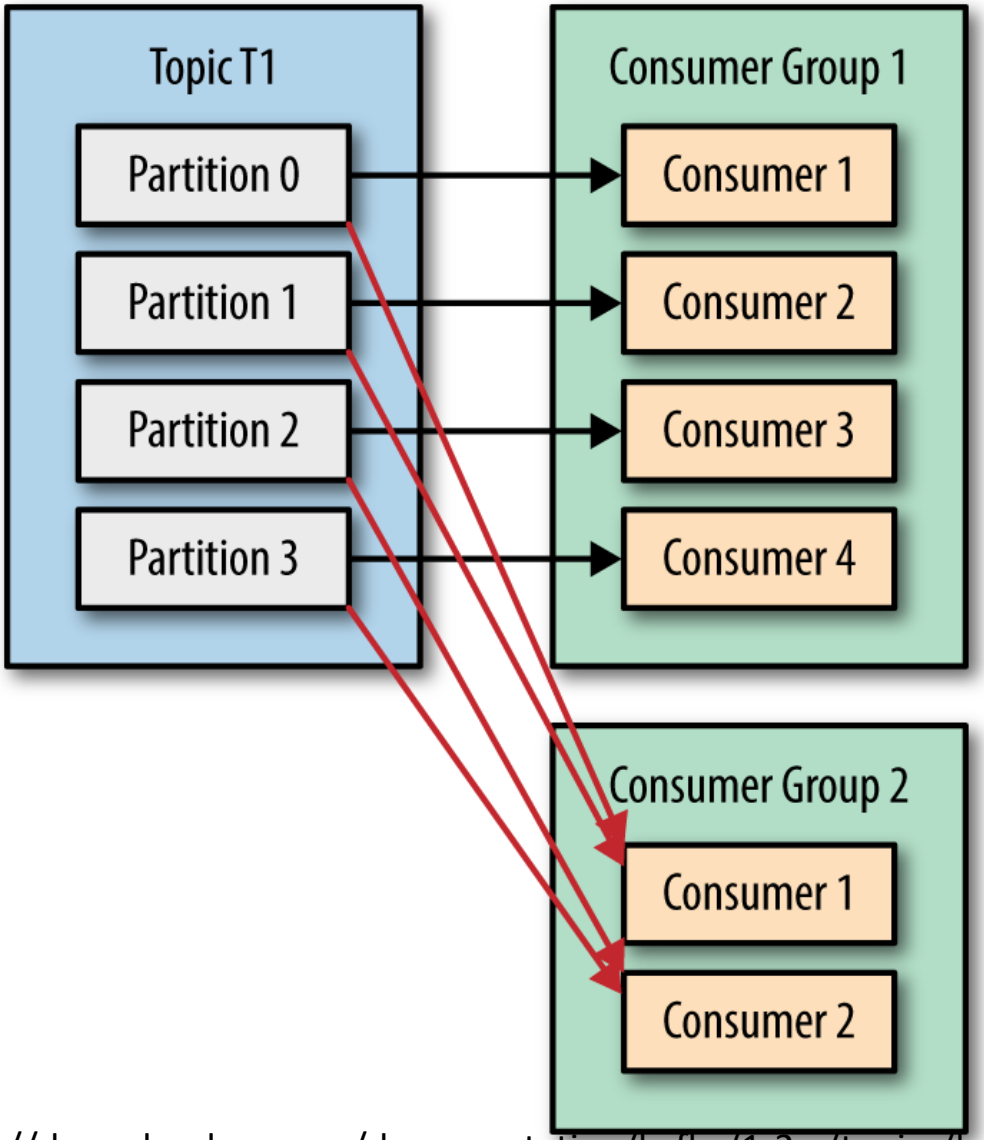

Producer
Producer
Producer
Producer
Topic A
Topic B
Partition 1
Partition 2
Partition 3
Partition 1
Partition 2
Partition 3
Consumer
Consumer
Consumer

Consumer groups

- Mechanism for scaling consumers
- A single logical consumer
  - Multiple consumers instances reading from the same topic
  - Share the same group ID
- Partitions are dispatched to different consumers of the same group
  - 2 consumers of the same group **cannot** read the same partition
  - Some consumers might be idle if not enough partitions

Consumer groups - 2

- Single consumer group
  - Add/remove consumers to scale
  - When leaving/joining group, need to rebalance
    - Reassign partitions to consumers
- Multiple consumer groups
  - Can work on same or different topics
  - Same topic
    - Separation of processing logic



https://docs.cloudera.com/documentation/kafka/1-2-x/topics/kafka.html

Offsets

- *Current Offset*
  - Position of the next message to read in partition
  - Per consumer group
- Purpose
  - Ordering : consumer group read messages in order they were produced
  - Fault tolerance : if a consumer crashes, another *from the same group* can continue (but from where?)
  - Replayability : can go back to previous offset to read older messages
  - Scalability : multiple consumer groups can read from the same topic without interfering

Offsets

- *Commit Offset*
  - Position of last “officially” processed data
  - Set by a consumer
    - Automatic (time based)
    - Manual
- Purpose
  - Indicates processed messages (as opposed to just read)
- When rebalancing, start from the last commited offset

High availability

- Topics are replicated
  - Replication factor per topic
- Replicated partitions stored on different brokers
- For each partition, a leader is elected
  - Broker which handles client request and manages replicas
- If the leader fails
  - New leader elected among replicas
  - Another replicas added if possible

Data ingestion

Storm-Kafka integration

UNIVERSITÉ
CÔTE D'AZUR

Rational

- Use Kafka as a source for data … or a sink
- Storm already provides basic Spouts and Bolts for this
  - org.apache.storm.kafka.bolt.KafkaBolt
  - org.apache.storm.kafka.spout.KafkaSpout
- Can be used in any topology
  - KafkaBolt shouldn't be used for processing,
  - Need some configuration to connect to Kafka
- 2 dependencies
  - kafka-clients
  - storm-kafka-client

KafkaBolt

- Configuration
  - Specified with org.apache.kafka.clients.producer.ProducerConfig and java.util.Properties
  - Must contain
    - Address of a broker (BOOTSTRAP_SERVERS_CONFIG)
    - Topic name (CLIENT_ID_CONFIG)
    - Kafka classes to serialize key and value

KafkaBolt

- Instantiation
  - Specify type of Key and Value as bounded type
  - Specify mapping between Storm fields (input) and Kafka key-value
- Use of call chaining

```
KafkaBolt<String, String> bolt = new KafkaBolt<String, String>()
  .withProducerProperties(prop)
  .withTopicSelector(new DefaultTopicSelector(topicName))
  .withTupleToKafkaMapper(new FieldNameBasedTupleToKafkaMapper<>("key", "lambda"));
```

KafkaSpout

- More complex than Bolts
  - Will be part of a consumer group
  - Manage commits
  - Transform Kafka *key-value* into Tuples
- Configuration
  - Specified with org.apache.kafka.clients.consumer.ConsumerConfig, java.util.Properties and org.apache.storm.kafka.spout.KafkaSpoutConfig

KafkaSpout

- Configuration is created with *KafkaSpoutConfig.builder(…)* and call chaining
- Configuration
  - Address of a broker
  - Consumer ID (*ConsumerConfig.GROUP_ID_CONFIG*)
  - How to convert Kafka data to tuples with a *ByTopicRecordTranslator*
  - Which topics to register to

KafkaSpout – topic translator

- The topic translator specifies
  - What *Values* to extract from a Kafka record (*org.apache.kafka.clients.consumer.ConsumerRecord<K,V>*)
  - What *Fields* to emit
  - On which stream (*default* if not specified)
  - On which topic to apply a specific translation logic (all if not specified)
    - The same translator can be used for different topics with different logic

```
ByTopicRecordTranslator<String, String> trans = new ByTopicRecordTranslator<>(
    (r) -> new Values(r.topic(), r.partition(), r.offset(), r.key(), r.value()),
    new Fields("topic", "partition", "offset", "key", "value"), TOPIC_0_1_STREAM);
```

```
trans.forTopic(TOPIC_2,
    (r) -> new Values(r.topic(), r.partition(), r.offset(), r.key(), r.value()),
    new Fields("topic", "partition", "offset", "key", "value"), TOPIC_2_STREAM);
```

KafkaSpout – creating config and Spout

```
KafkaSpoutConfig config= KafkaSpoutConfig.builder(bootstrapServers, new String[]{TOPIC_0, TOPIC_1, TOPIC_2})
.setProp(ConsumerConfig.GROUP_ID_CONFIG, "kafkaSpoutTestGroup")
.setRecordTranslator(trans)
.setOffsetCommitPeriodMs(10_000)
.setFirstPollOffsetStrategy(EARLIEST)
.setMaxUncommittedOffsets(250)
.build();
```

```
final TopologyBuilder tp = new TopologyBuilder();
tp.setSpout("kafka_spout", new KafkaSpout<>(spoutConfig), 1);
```

Wrapping it all together

Lambda architectures

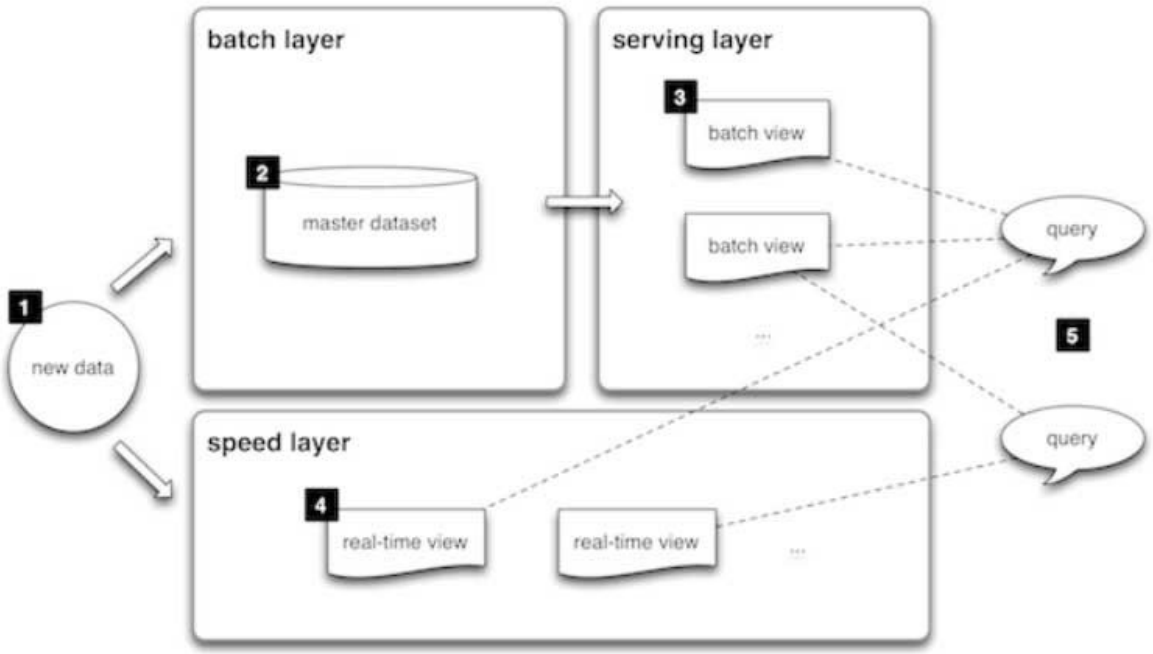
UNIVERSITÉ CÔTE D'AZUR

Recap

- We have seen architectures to
  - Store and process Big Data
  - Process Fast Data
- Constraints and objectives are different
  - Different framework with common parts (HDFS,…)
- But some use-cases process Big and Fast Data
  - Quickly analyze and react to new data
  - Perform complex analytics later

Lambda Architecture

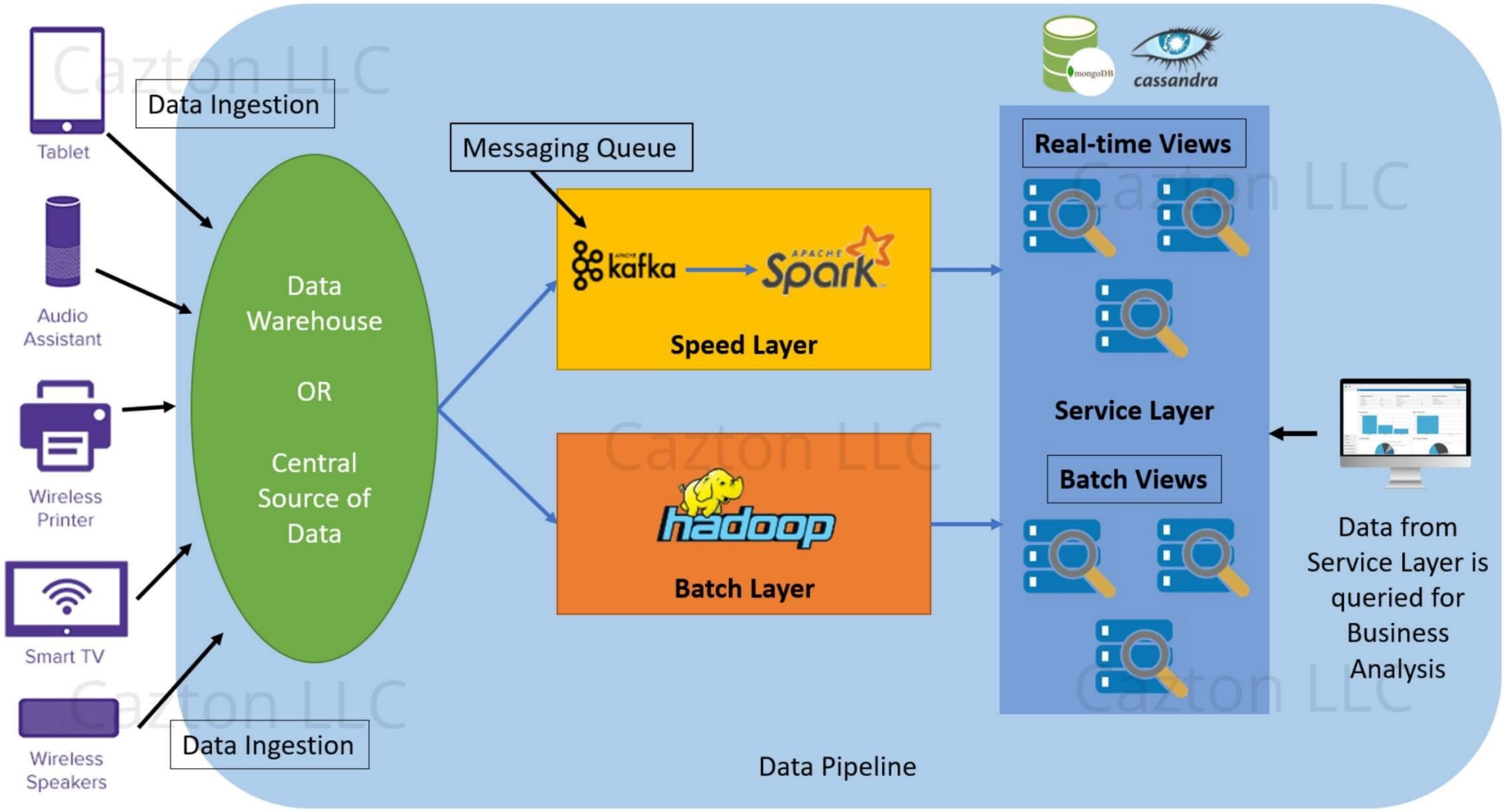
- An architecture for both Fast and Big Data



https://databricks.com/glossary/lambda-architecture

Layers

- Batch layer
  - Manage the immutable dataset
  - Pre-compute queries to create batch views
- Serving layer
  - Precompute batch views so they can be queried efficiently
- Speed layer
  - Analyze recent data only



https://cazton.com/consulting/enterprise/lambda-architecture