LearnQuest

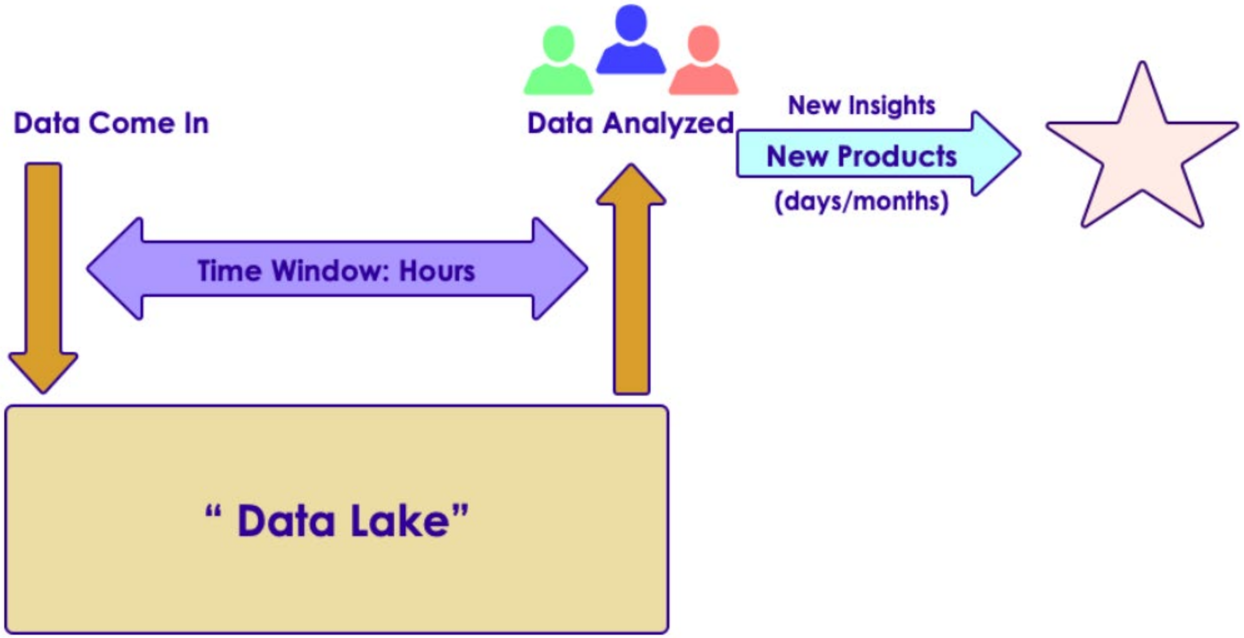
Presents

Streaming with Kafka

www.LearnQuest.com

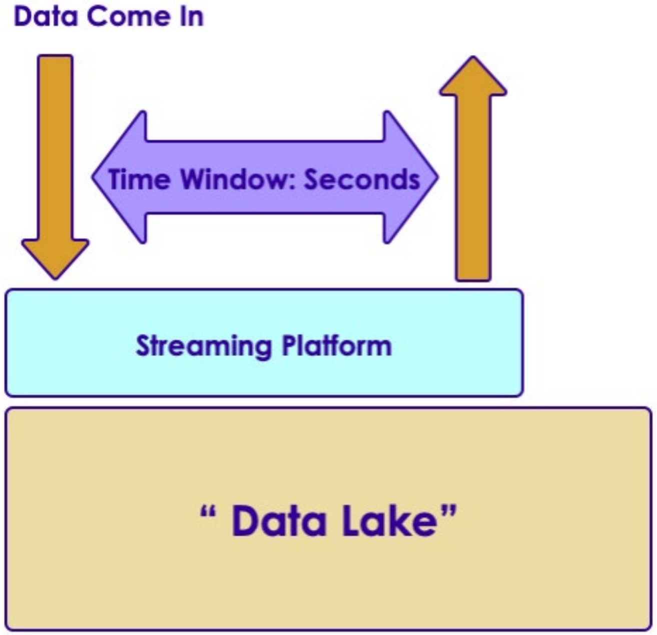
Big Data Evolution: Version 1

- Decision times: batch (hours / days)
  - ✓ Use cases: Modeling, ETL, Reporting



Moving Towards Fast Data: Version2

- Decision time: (near) real time
  - ✓ Seconds (or milliseconds)
- Use Cases
  - ✓ Alerts (medical/security)
  - ✓ Fraud detection
- Streaming is becoming more prevalent
  - ✓ Connected Devices
  - ✓ Internet of Things
- Beyond Batch
  - ✓ We need faster processing and analytics

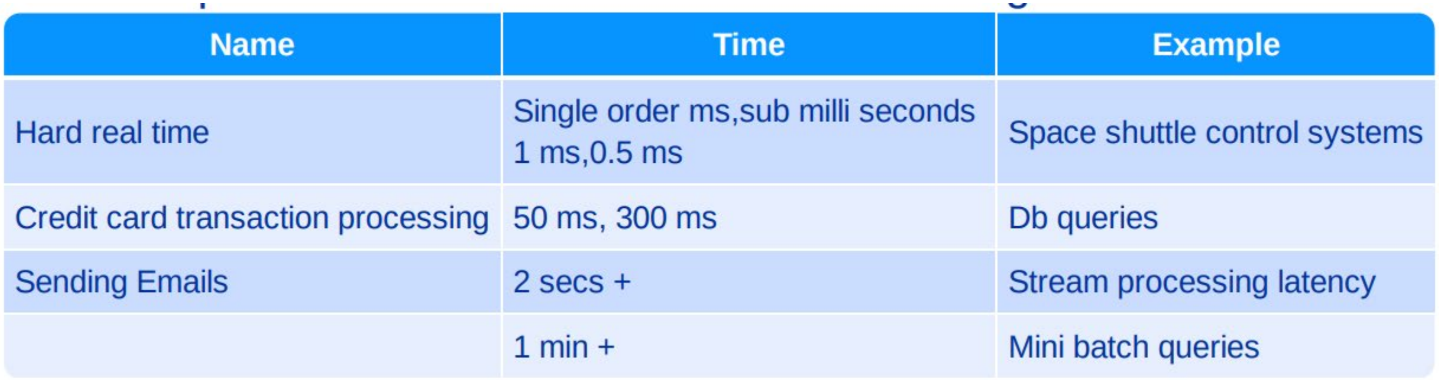


Streaming Use Cases

- ▶ Netflix
  - ✓ Recommendations 450 billion events/day
- ▶ Weather Company
  - ✓ Analyze weather sensor data
  - ✓ Billions of events/day
  - ✓ Multi-Petabyte (PB) traffic daily

Real Time / Near Real Time

- The 'real' real time is in milliseconds order
  - ✓ DB query returns in 2 ms
- ‘Near real time' is seconds
  - ✓ We can process an event within 3 seconds of its generation time

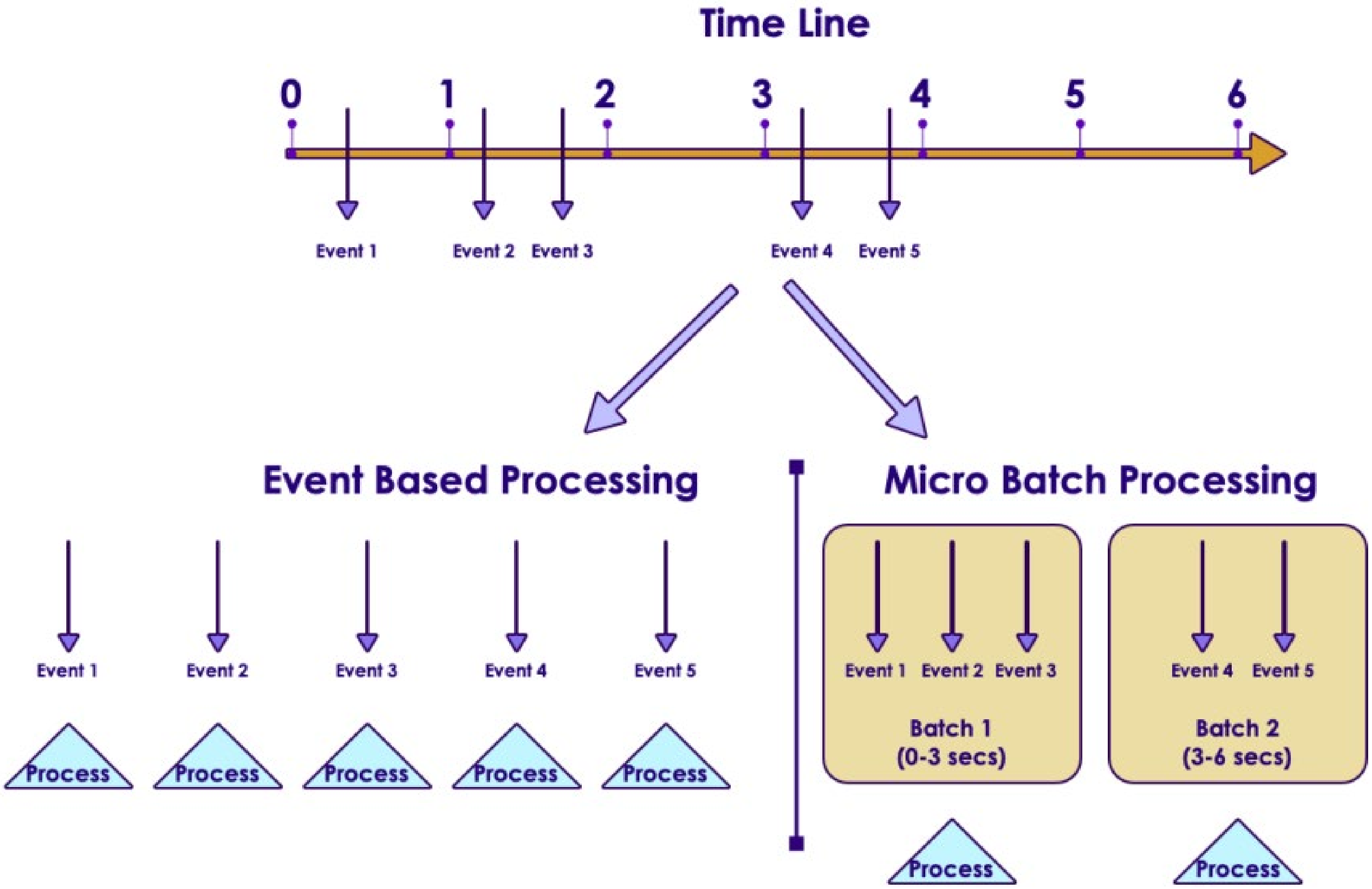


Name	Time	Example
Hard real time	Single order ms,sub milli seconds 1 ms,0.5 ms	Space shuttle control systems
Credit card transaction processing	50 ms, 300 ms	Db queries
Sending Emails	2 secs +	Stream processing latency
	1 min +	Mini batch queries

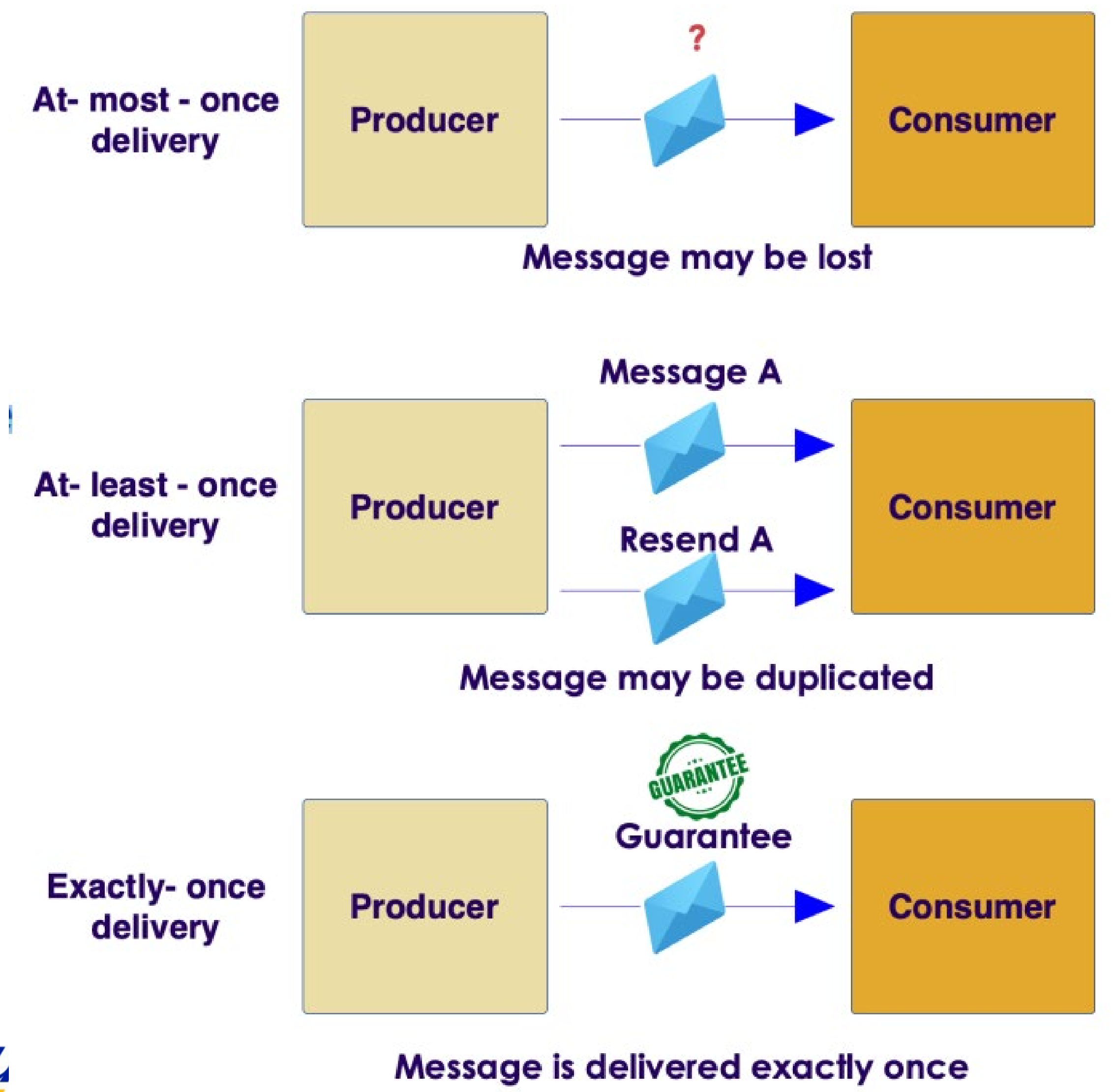
Streaming Concepts

- Processing model
  - ✓ Event based or micro batch based
- Processing guarantees
  - ✓ At least once
  - ✓ At most once
  - ✓ Exactly once
- State management
  - ✓ Event time vs. Arrival time
- Window Operations
- Back-pressure adjustment

Event Based Vs. Batch

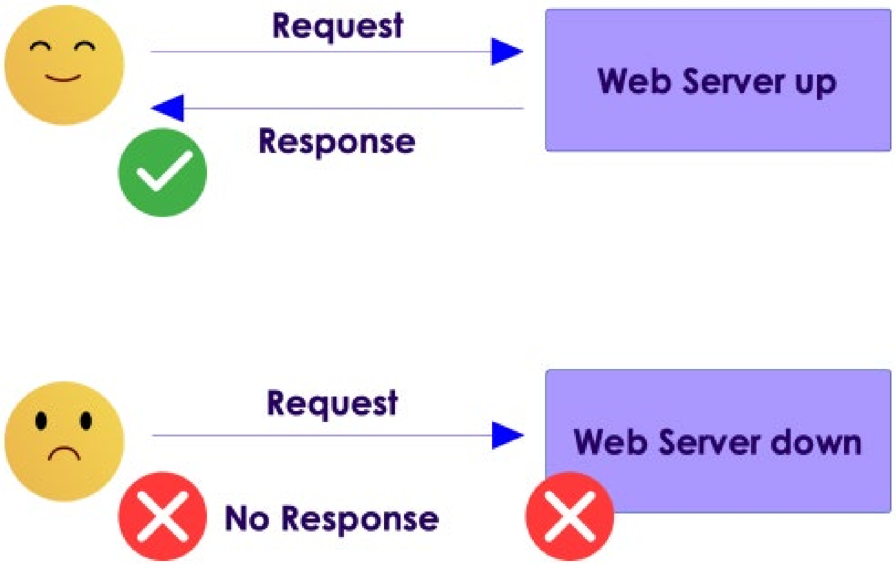


Processing Guarantees



At Most Once

- Event is sent only once
  - ✓ No duplicate processing
  - ✓ Events can be dropped due to crashes or heavy load
  - ✓ E.g. Web requests (if the web server is busy, requests are dropped)



At Least Once

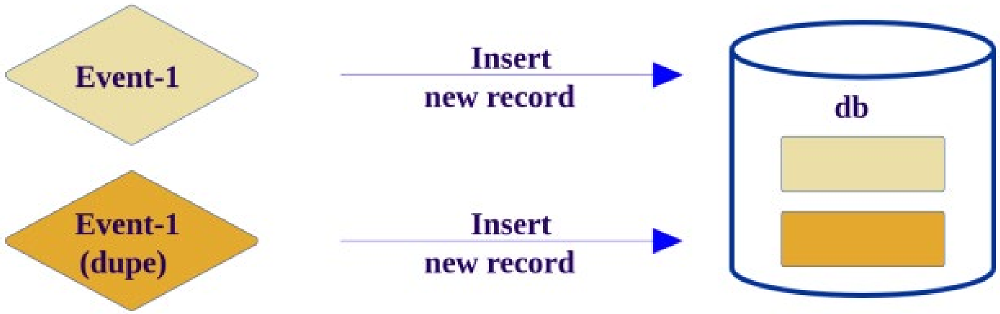
- All events are guaranteed to be processed (no dropped events)
  - ✓ However, events can be processed more than once
  - ✓ In case of failure recovery, events can be re-sent and processed again.
- Most common implementation
  - ✓ Frameworks: All (Storm, Spark, NiFi, Samza, Flink)

Handling Duplicate Events

- A resilient streaming system, has to be ready to handle duplicate events
- We have 2 scenarios:
  - ✓ First one, we are inserting a new record for each event received. This will result in duplicate records in the database
  - ✓ Second one, we are checking to see if the event is processed already, only if not, then a new record is inserted
- Second approach is more resilient, can deal with duplicate events
  - ✓ This is called idempotent processing (no side effects for duplicate events)

Handling Duplicate Events

Scenario-1: Duplicate records created



Event-1

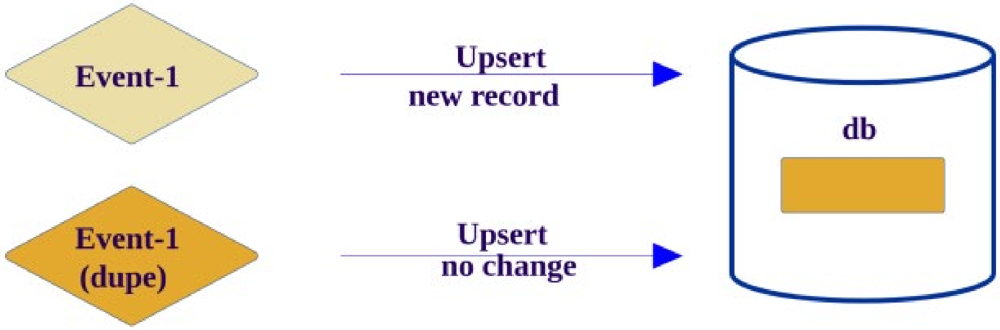
Insert
new record

Event-1
(dupe)

Insert
new record

db

Scenario-2: No Duplicate records created



Event-1

Upsert
new record

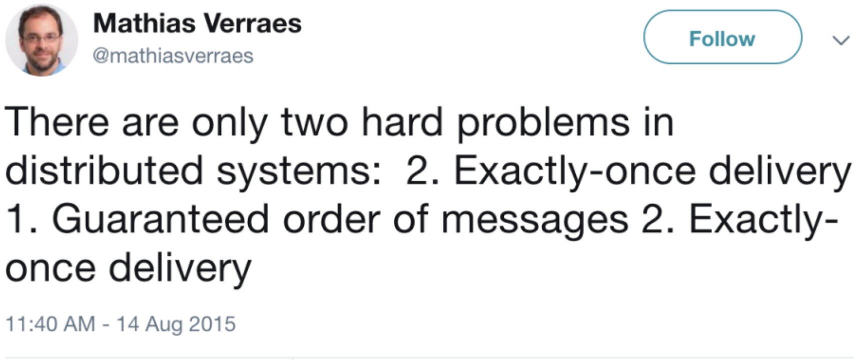
Event-1
(dupe)

Upsert
no change

db

Exactly Once

- Events are guaranteed to be processed once and only once
  - ✓ No dropped events
  - ✓ No duplicate processing
  - ✓ Frameworks: Storm (with Trident), Flink, Spark, Samza
- Sample applications
  - ✓ Credit card processing



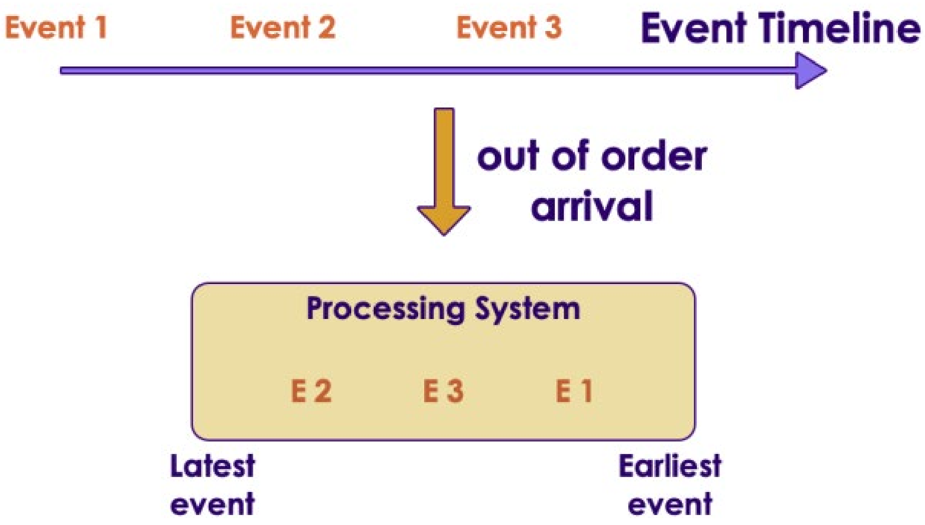
Mathias Verraes
@mathiasverraes

There are only two hard problems in distributed systems: 2. Exactly-once delivery 1. Guaranteed order of messages 2. Exactly-once delivery

11:40 AM - 14 Aug 2015

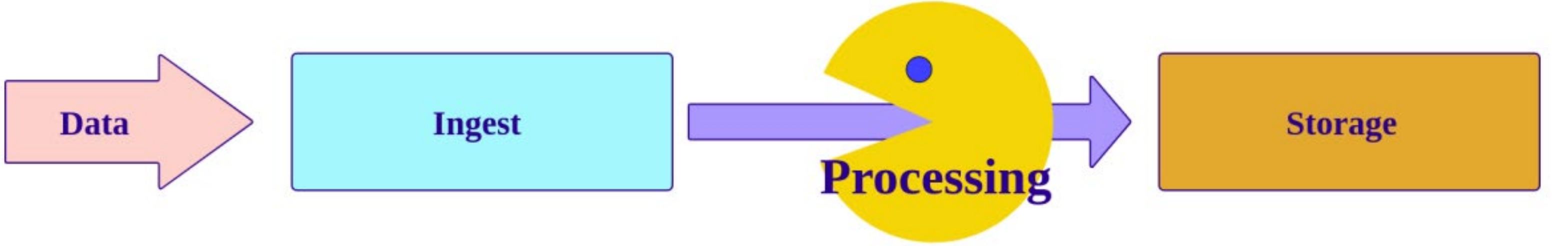
Event Time and Arrival Time

- Event Time: When the event occurred / generated
- Arrival Time: When event arrives for processing
- Event Time < Arrival Time
  - ✓ Some times events may arrive 'out of order' (due to network lag, outtage ..etc)



3 Tier Streaming Architecture

- Here is a simplified streaming architecture
- We see 3 distinct stages
  - ✓ Ingest stage captures data
  - ✓ Processing handles the data
  - ✓ And the processed data is stored in Storage layer



Ingest / Capture

- This layer:
  - ✓ Captures incoming data
  - ✓ Acts as a 'buffer' - smoothes out bursts So even if our processing offline, we won't loose data
- Choices
  - ✓ ***Kafka***
  - ✓ Queues (MQ, JMS ..etc)
  - ✓ Cloud based queues like Amazon Kinesis

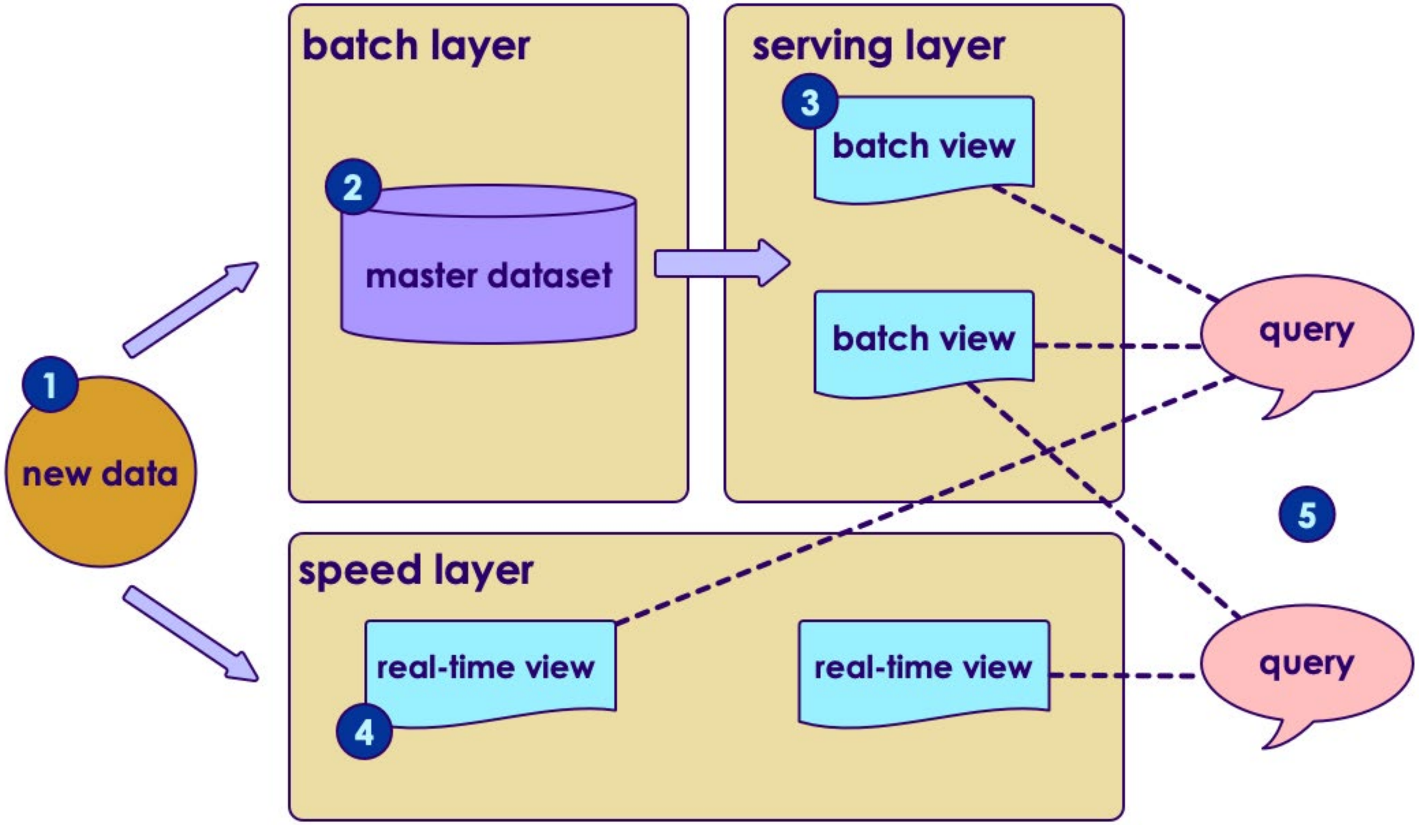
Processing

- We need to process events with low latency
  - ✓ (milliseconds to seconds)
- There are many stream/event processing frameworks available
  - ✓ Storm
  - ✓ Spark
  - ✓ NiFi
  - ✓ Samza
  - ✓ Flink
  - ✓ Beam

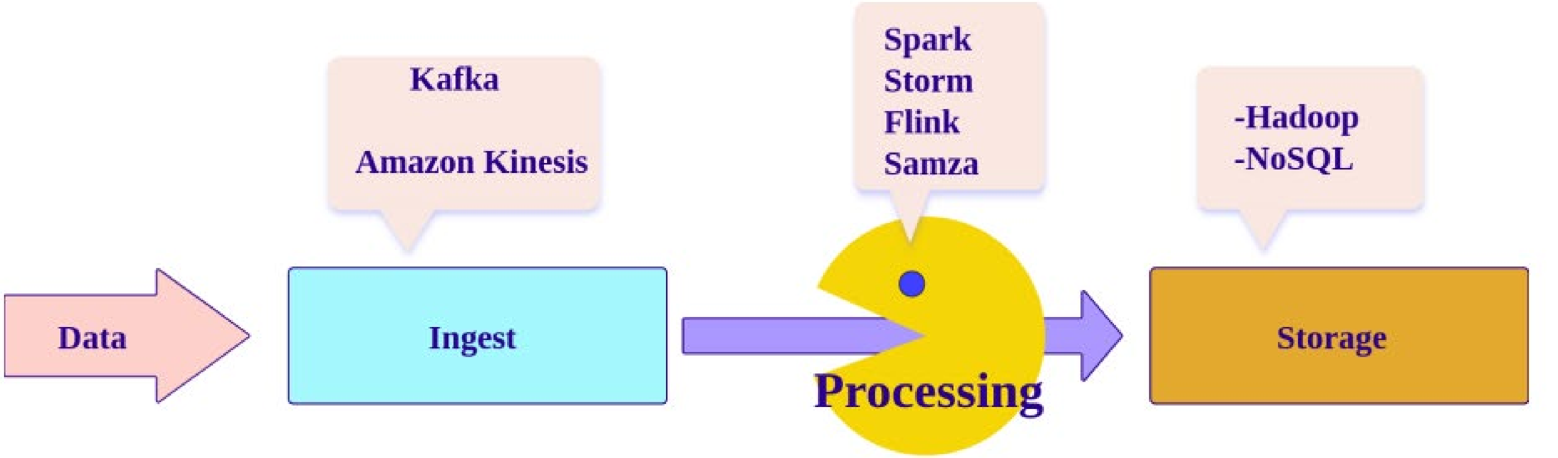
Storage

- After processing, they are stored for later retrieval
- Two choices:
  - ✓ Real time store
  - ✓ 'Forever' store
- Real Time Store
  - ✓ Need to absorb data in real time
  - ✓ Usually a NoSQL storage (HBase, Cassandra ...etc)
  - ✓ May contain subset of data (last 1 year ..etc)
- 'Forever store'
  - ✓ Needs to store massive amounts of data
  - ✓ Support analytics (usually batch)
  - ✓ Hadoop / HDFS

Lambda Architecture



Streaming Stack - Summary

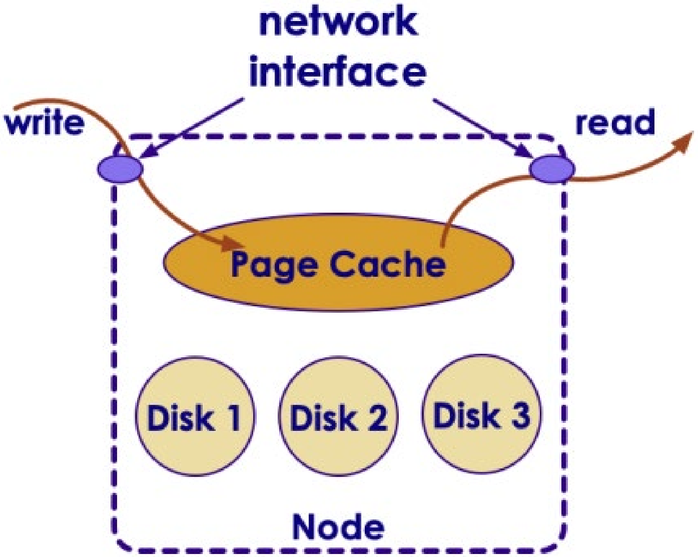


Apache Kafka

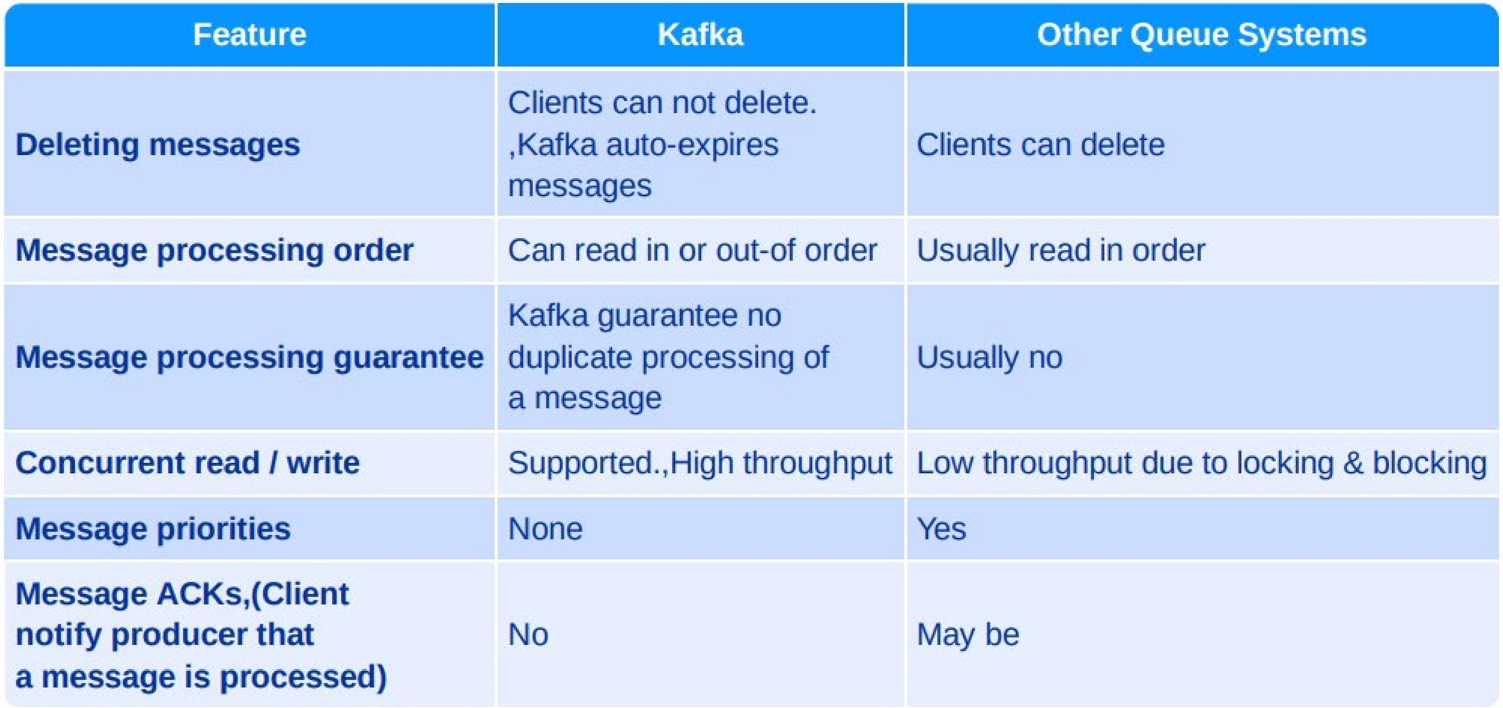
- Kafka is a Publisher / Subscriber (Pub-Sub) messaging system
  - ✓ Distributed
    - Scales seamlessly
  - ✓ High throughput
    - Capable of handling billions of messages per day
  - ✓ Replicated
    - Safeguards data in case of machine failures
- Created @ LinkedIn in 2010
  - ✓ Now Apache Project (Open Source)

Why Is Kafka Very Fast?

- Write: Disk writes are buffered in page cache
- Read: The data from page cache can be transferred to network interface very efficiently
- 99% of the time data is read from page cache, no disk access at all



Kafka Features



Feature	Kafka	Other Queue Systems
Deleting messages	Clients can not delete. ,Kafka auto-expires messages	Clients can delete
Message processing order	Can read in or out-of order	Usually read in order
Message processing guarantee	Kafka guarantee no duplicate processing of a message	Usually no
Concurrent read / write	Supported.,High throughput	Low throughput due to locking & blocking
Message priorities	None	Yes
Message ACKs,(Client notify producer that a message is processed)	No	May be

Questions