Event processing with Kafka for the Pythonista

Pradeep Gowda

pradeep@btbytes.com

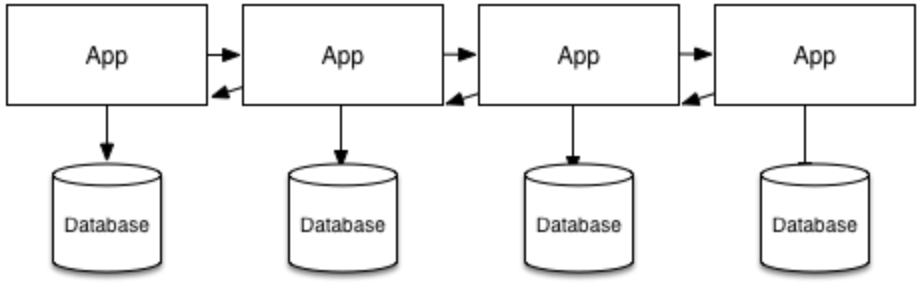
October 11, 2016

Indypy

Survey

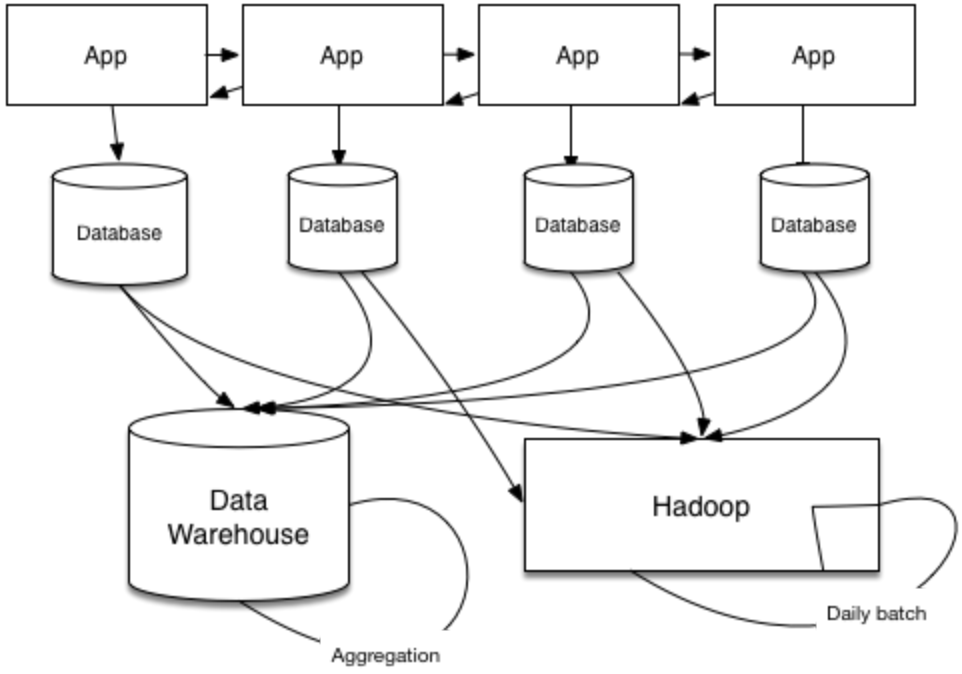
The nature of data

"pre-streaming" data architecture

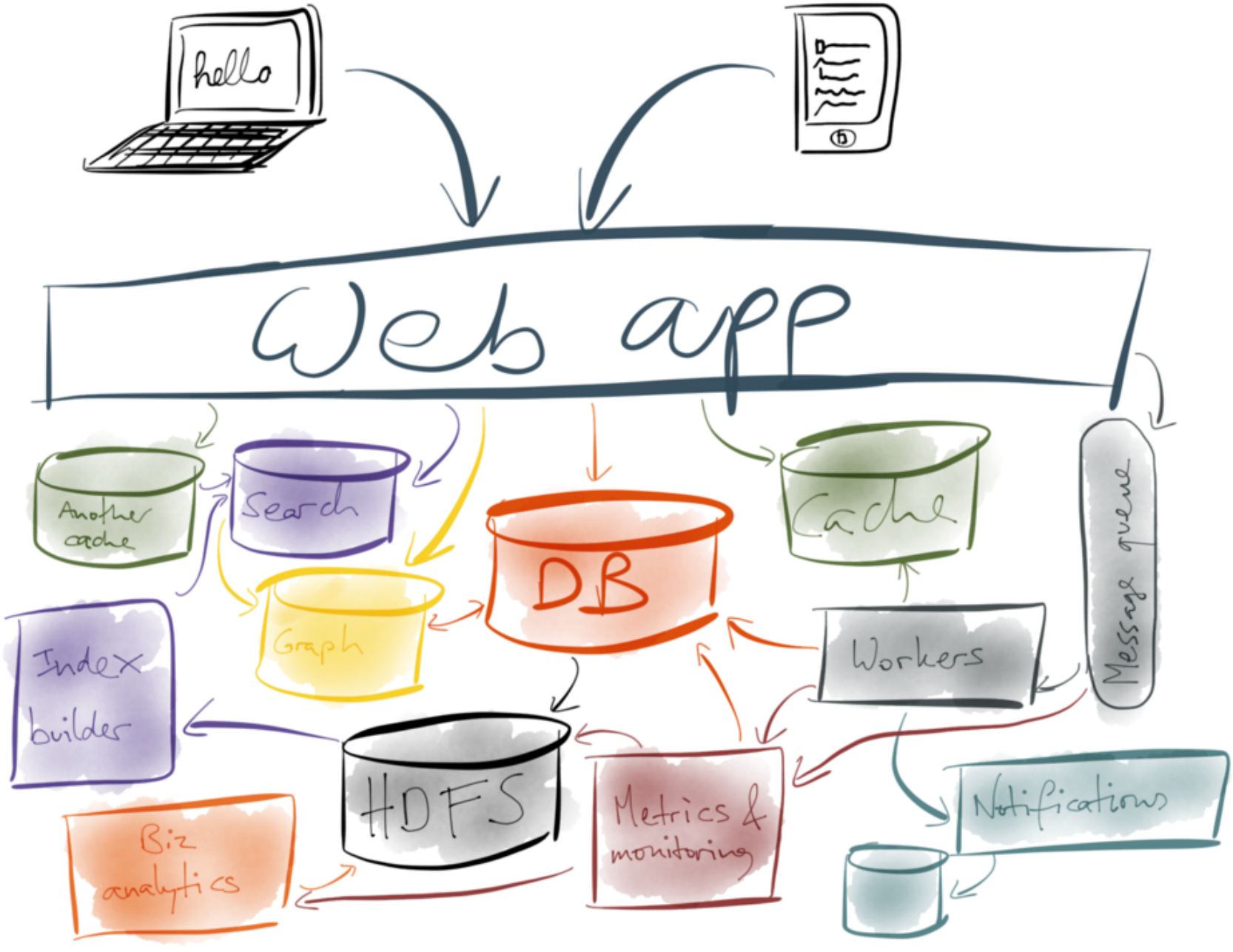


(3-layers, n-layers)

Problem: Pipeline sprawl

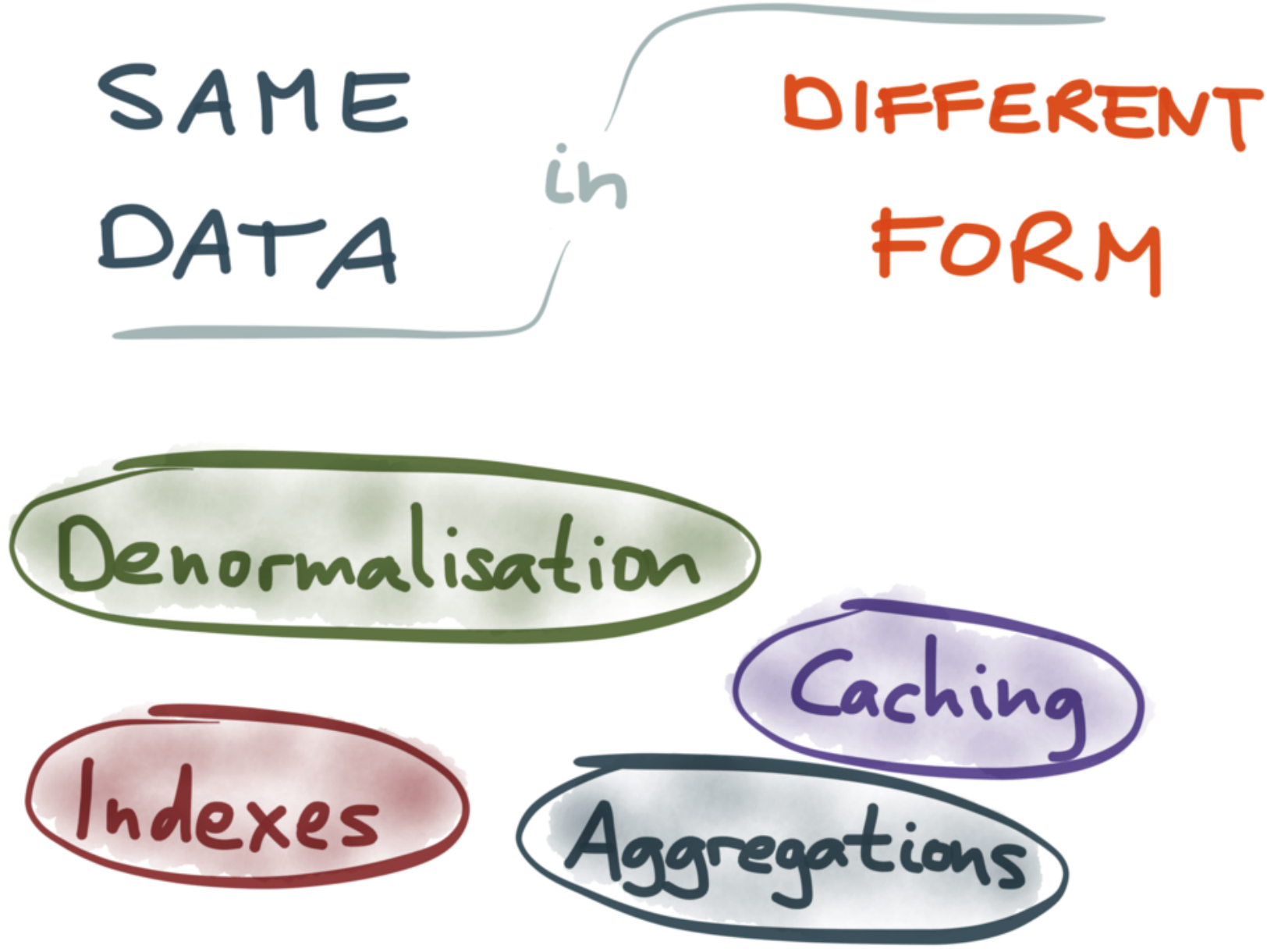


Problem: Everything is synchronous



How did we end up here?

- multiple downstream users
- evolution of uses over time not anticipated at design time
- previous solutions to address this -- data lake, OLAP


SAME DATA
in
DIFFERENT FORM
Denormalisation
Caching
Indexes
Aggregations

All your data is event streams

Event data example

1. User views a product

```
{
    "product_id": 1234,
    "time": "2016-10-05T12:12:12",
    "user_id": 3298,
}
```

Event data example (continued..)

2. User checks-in to a new location
3. User catches a poke-thing
4. Log file

```
64.242.88.10 - - [07/Mar/2004:17:29:59 -0800] "GET /twiki/
64.242.88.10 - - [07/Mar/2004:17:31:39 -0800] "GET /twiki/
64.242.88.10 - - [07/Mar/2004:17:35:35 -0800] "GET /twiki/
```

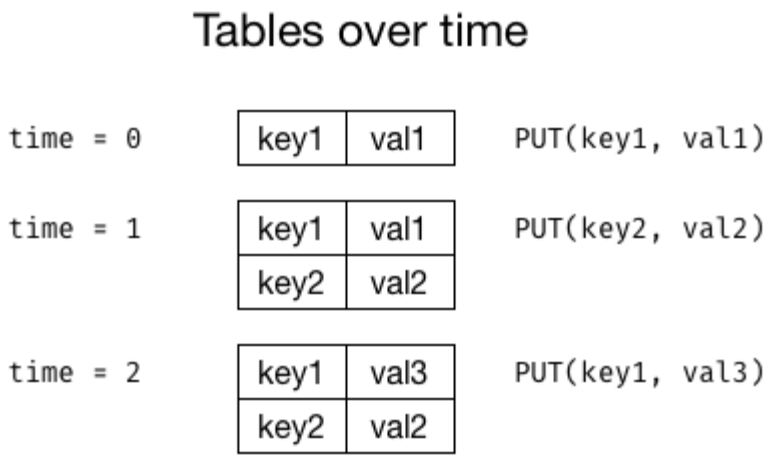
5. Sensors

What are tables?

Tables

key1	val3
key2	val2

Tables over time



Sounds like ...

Write-Ahead Logging (WAL)

- The Write-Ahead Logging Protocol:
  - ① Must force the log record for an update *before* the corresponding data page gets to disk.
  - ② Must write all log records for a Xact *before commit*.
- #1 guarantees Atomicity.
- #2 guarantees Durability.

- Exactly how is logging (and recovery!) done?
  - We'll study the ARIES algorithms.

WAL-first rule

EnterpriseDB®
The Enterprise Postgres Company

- The log record for an operation is always written to disk before the affected data pages
  - WAL first rule!
- In case of a crash, the WAL is replayed to reconstruct the unsaved changes

WAL Summary

EnterpriseDB
The Enterprise Postgres Company

Write-Ahead Logging is critical for:

- Durability
  - Once you commit, your data is safe
- Consistency
  - No corruption on crash

Advanced features

EnterpriseDB®
The Enterprise Postgres Company

But there's more!

The Write-Ahead Log *also* allows:

- Online backup
- Point-in-time Recovery
- Replication

What is Kafka?

Kafka is a modern distributed Platform for data streams

Three properties of a streaming platform

ie., Kafka

1. Publish and subscribe to stream of records (similar to Queues)
2. Store stream of records in a fault-tolerant way
3. Process streams of records as they occur

What is it used for?

Build real time

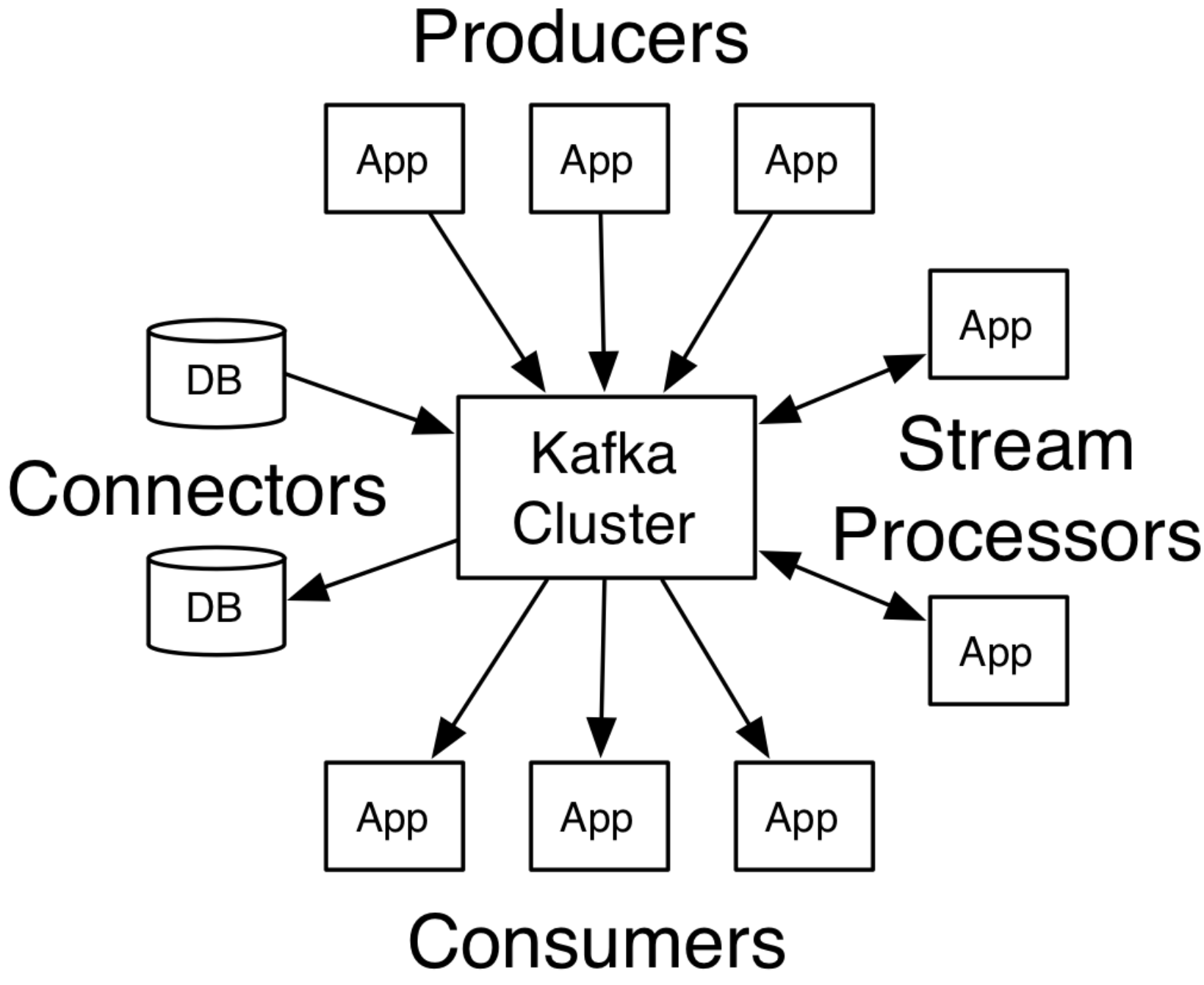
- streaming pipelines to ferry data between systems/apps
- streaming applications to transform/react to stream of data

Kafka concepts

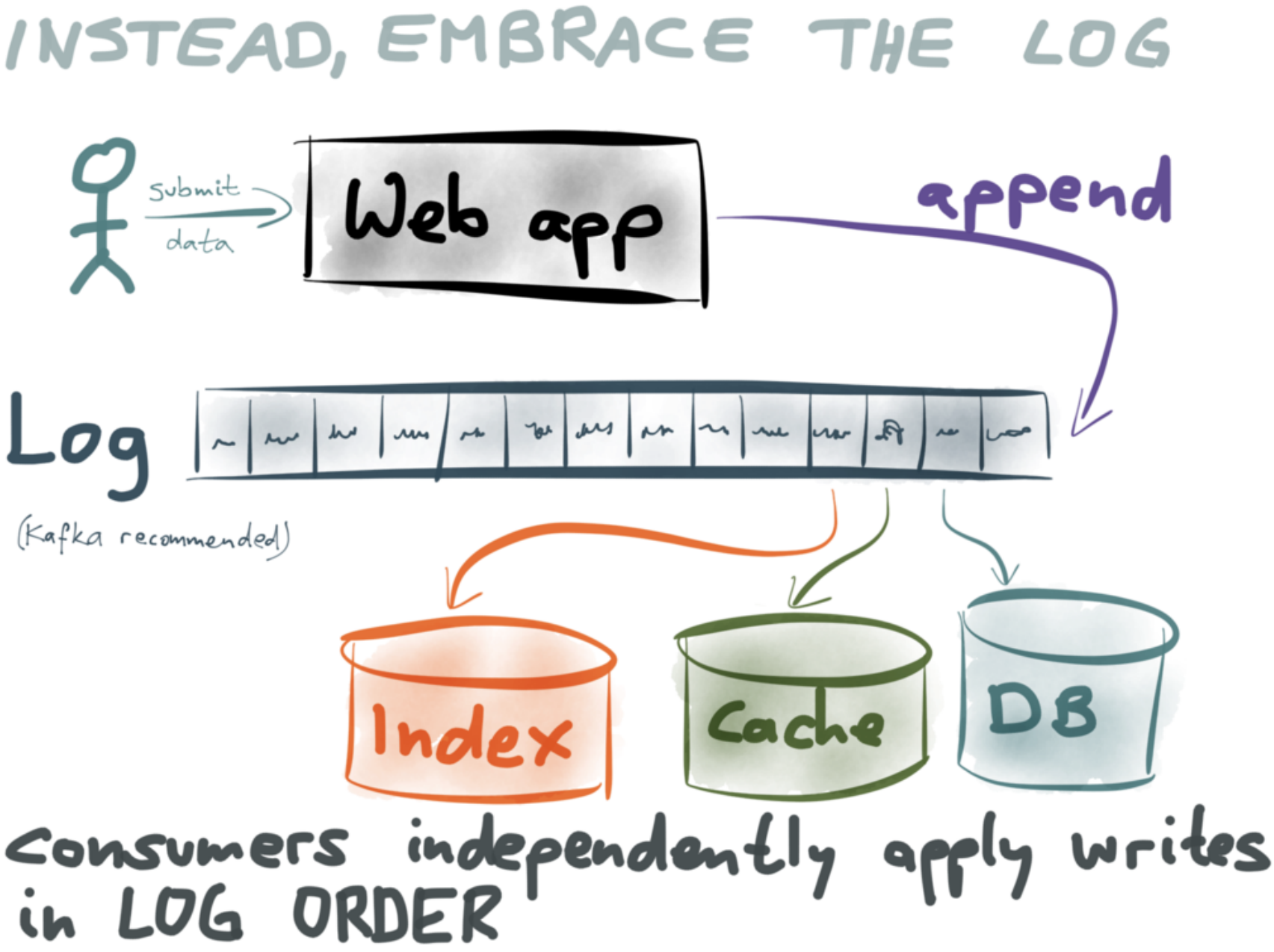
- Kafka is run as cluster of servers
- Kafka cluster stores streams of *records* in categories called *topics*
- Each record has this form — (`key`, `value`, `timestamp`)

Kafka APIs

1. Producer API
2. Consumer API
3. Streams API
4. Connector API

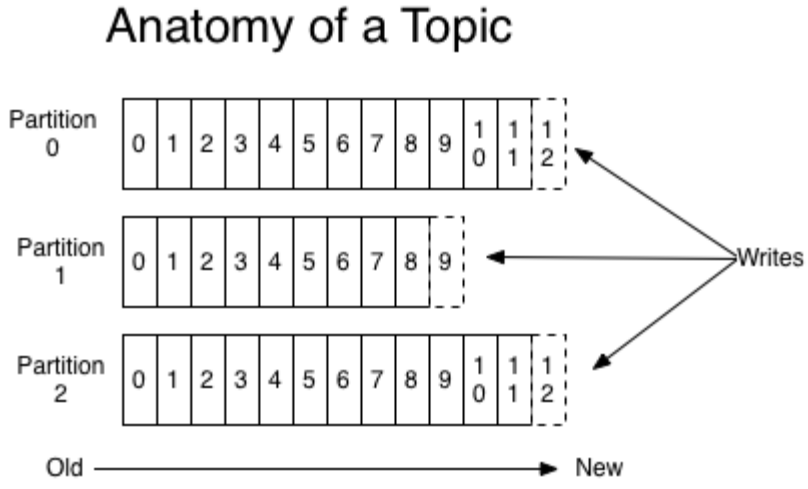

Producers
App
App
App
DB
Connectors
DB
Kafka
Cluster
App
Stream
Processors
App
App
App
App
Consumers

Streaming Data



Kafka Topic

- example of a topic: `poke-thing-capture`
- every topic is multi-subscriber. `0-n`

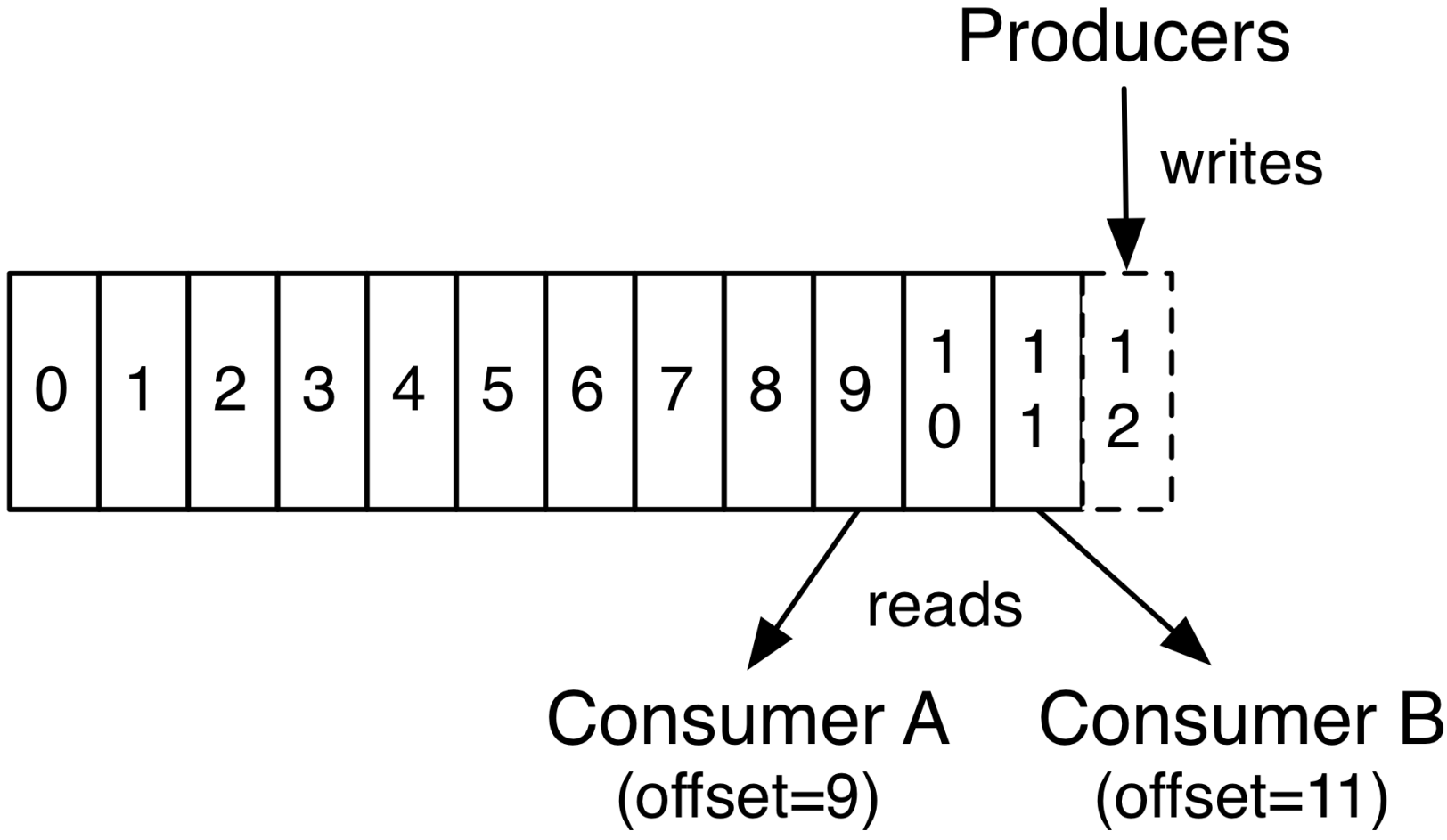


- Partition = immutable sequence of records.
- one topic may be split into multiple partitions
- ... **Append only**
- retained for `x` period (default= `1d`)

Producer

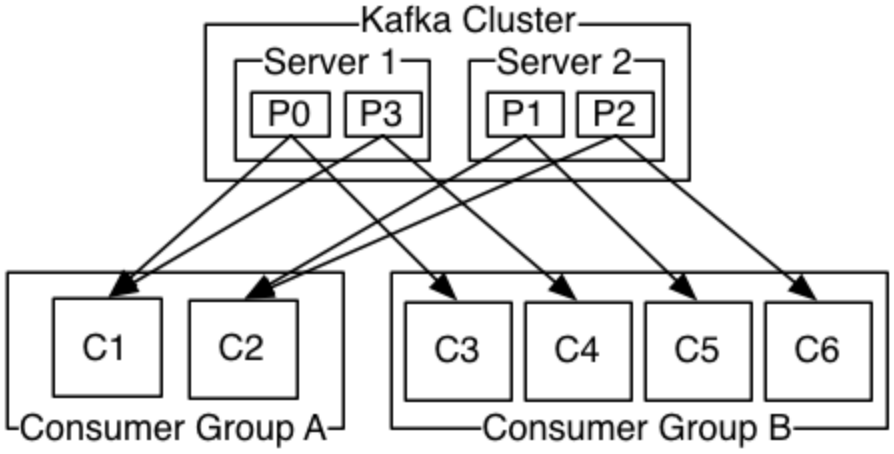
- Producers publish data to topics

Consumers



- Every consumer has an "offset".
- This offset is controlled by the consumer

Consumer Groups



- records load balanced over consumer instances

Total ordering

- Kafka only provides a total order over records within a partition, not between different partitions in a topic.

Distribution

- The partitions of the log are distributed over the servers in the Kafka cluster
- Each partition is replicated across a configurable number of servers for fault tolerance.
- Each partition has one server which acts as the "leader" and zero or more servers which act as "followers".

Leaders and followers

- The leader handles all read and write requests for the partition while the followers passively replicate the leader.
- If the leader fails, one of the followers will automatically become the new leader.
- Each server acts as a leader for some of its partitions and a follower for others so load is well balanced within the cluster.
- c.f. **zookeerer**

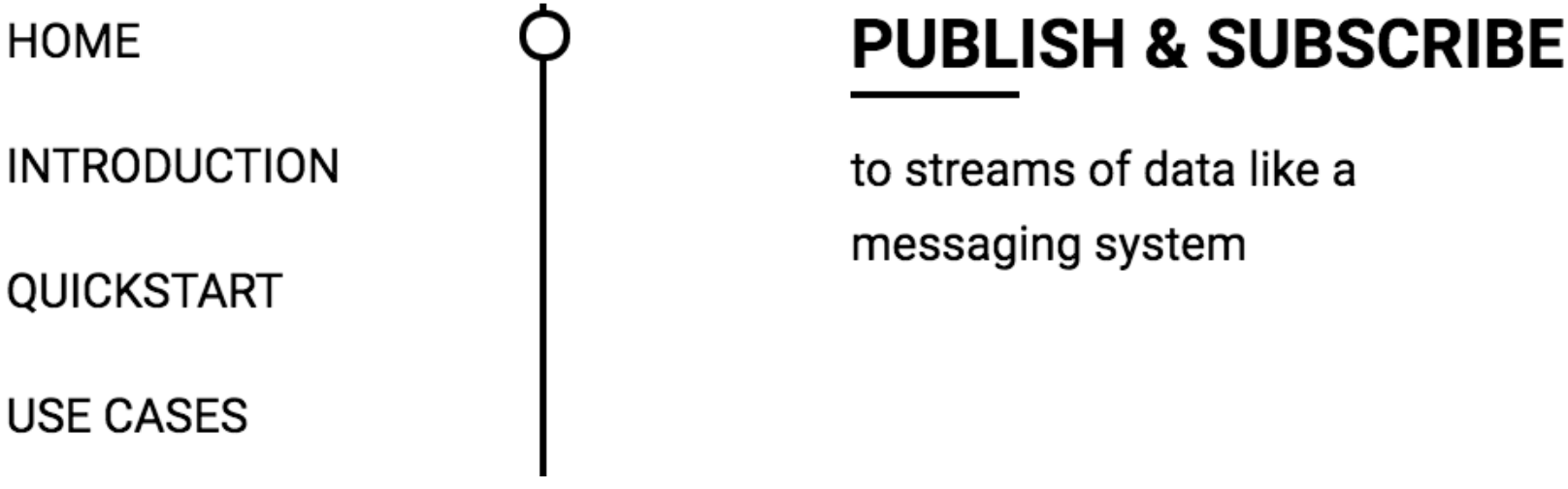
Kafka as ...

- Messaging system
    - queuing (-once you read the data, it's gone)
    - pub/sub
- Storage system
- Stream processing

Use cases

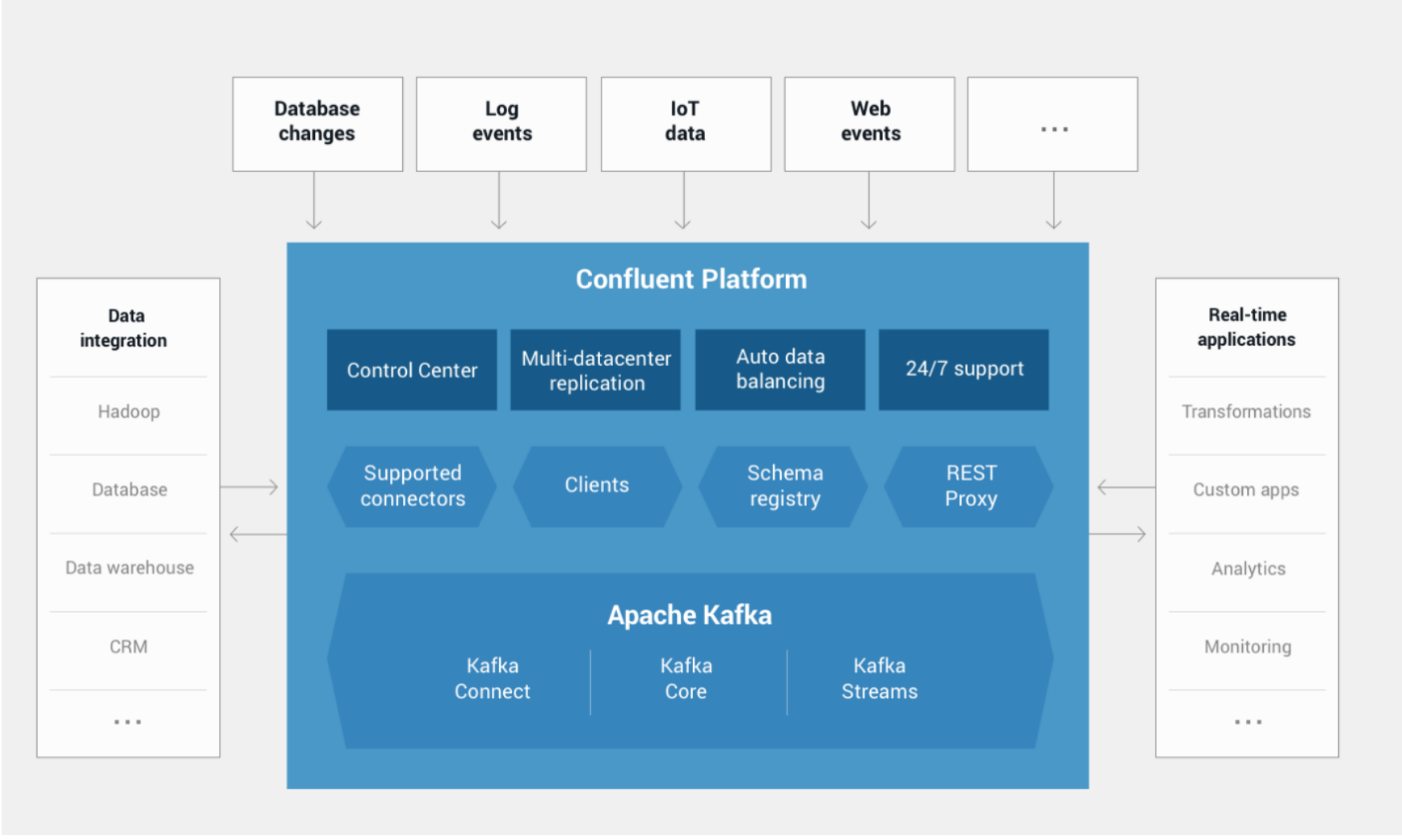
- Messaging (Ã la RabbitMQ)
- Website activity tracking
- Metrics
- Log aggregation
- Stream processing
- Event Sourcing
- Commit log (c.f log compaction)

Kafka is a top level Apache project — http://kafka.apache.org/



- Runs on the JVM
- Written in Scala
- Apache Community Development - A Maturity Model for

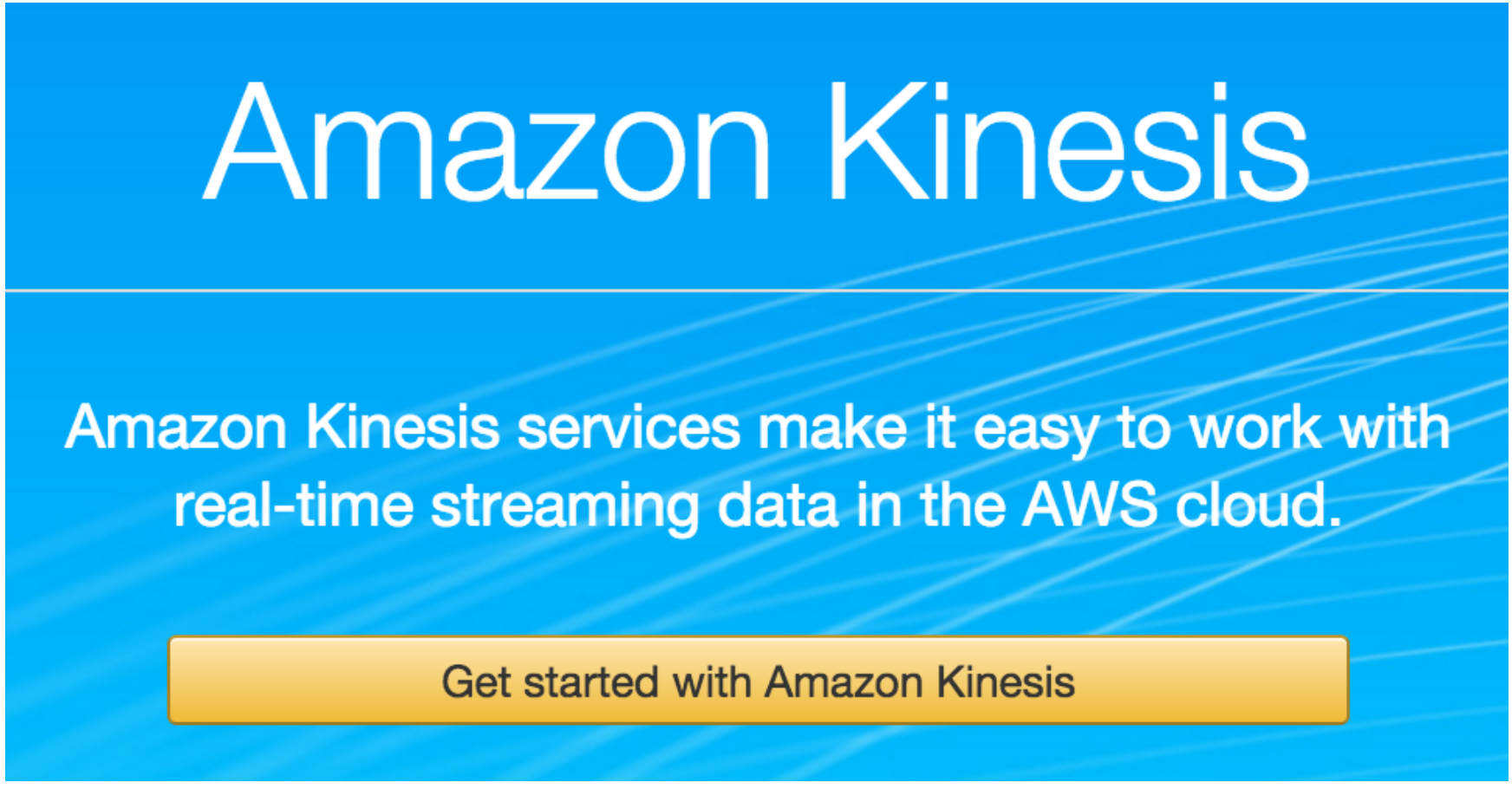
Confluent.io



People:

- Jay Kreps
- Neha Narkhede

Amazon Kinesis



- Firehose — load massive volumes of streaming data into AWS
- Analytics — analyze streaming data with standard SQL
- Streams — Build custom applications that process or analyze streaming data

Three key differences

1. Scale
2. Guarantees
3. Stream Processing

Scale

- Hundreds of MBs of Througput
- Store TBs of data
- Commodity Hardware
- `O(1)` writes
- Scalability of a file system

Guarantees

- Strict ordering (M1, M2)
- A consumer instance sees records in the order they are stored in the log.
- Handling server failures. (For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any records committed to the log.)

Distributed by design

- Replication
- Fault tolerance
- Partitioning
- Elastic scaling

Real life example

Linkedin

- `>1.2 trillion` messages per day written
- `>3.4 trillion` messages per day read
- `1PB` of stored data

Demo

Console demo

See `notes.txt`

Python demo

```
pip install kafka-python
```

Output:

```
See example-log.txt
```

Thank you!

Image Credits —

- @martinkl,
- @jaykreps
- kafka project
- Readings in Database Systems, 3rd Ed
- Heikki Linnakangas