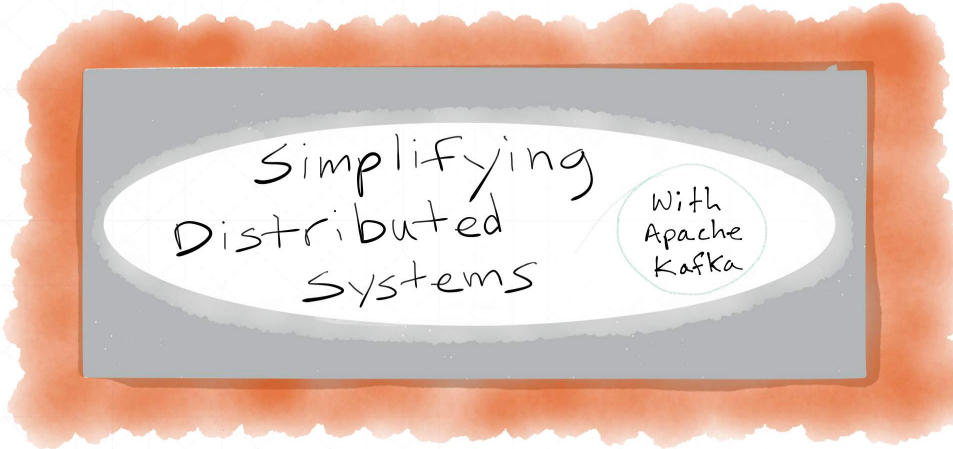
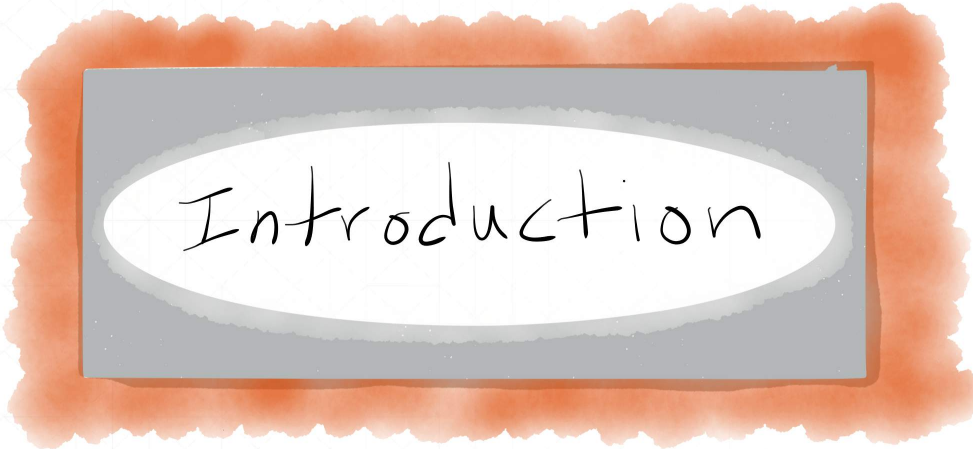
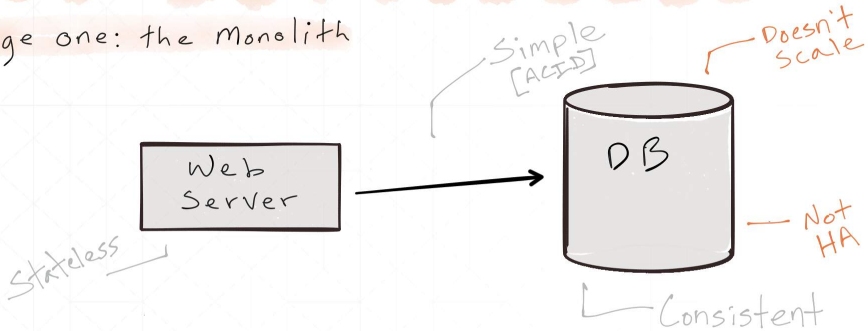

Simplifying
Distributed
Systems
With
Apache
Kafka


Introduction

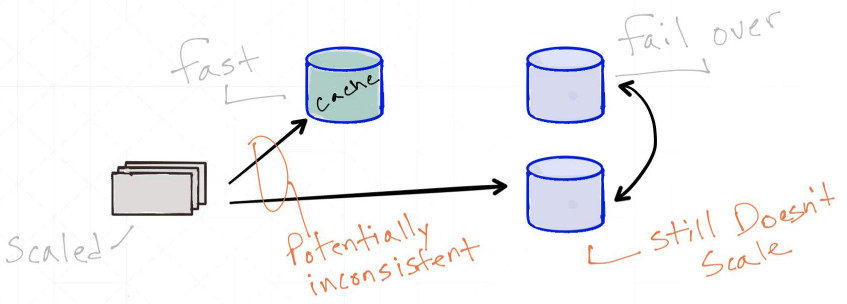
# Why Distributed?

[How did this happen]

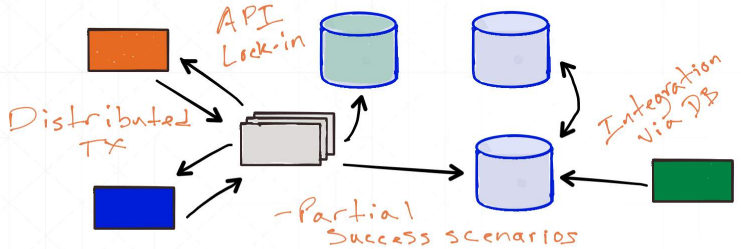
## Stage one: the Monolith



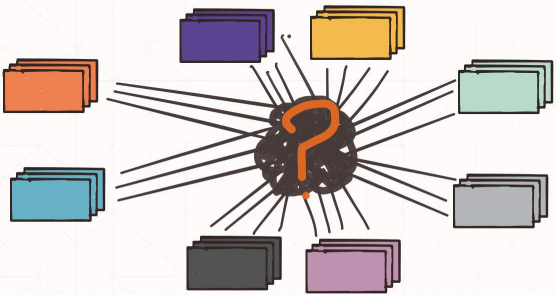
# Stage 2: HA/Scale



# Stage 3: Integrations



# So on...and so on



Microservices?

# The Problems

- Point-to-Point Communication
- Multi Dispatch updates
- Distributed Transactions
- Integration Via database
- API Coupling

# Apache Kafka

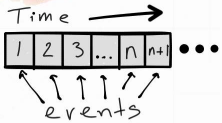
- Originally built by LinkedIn
- Apache top level project since 2012
- Used by many noteable companies
- Modeled as a distributed transaction log

# Design

## Append only log

- Commonly used in Database [see MVCC]
- ordered
- Analog to "stream"

The log



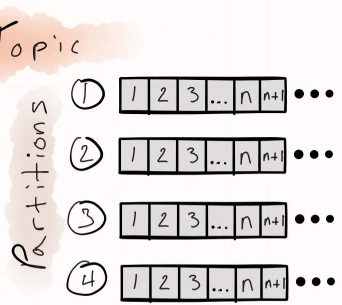
# More Design

A topic has 1-N partitions

Topics can be replicated

Partitions can reside on different brokers

Scales based on partition count

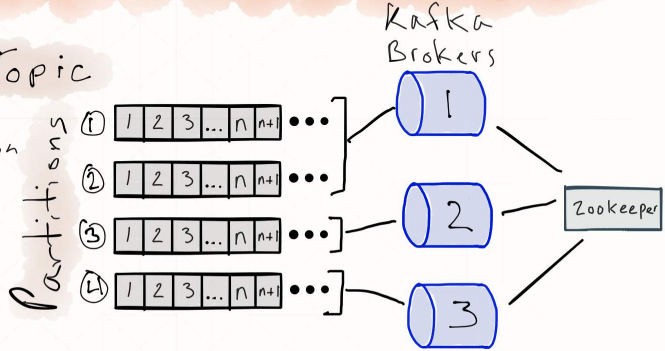


# Clusters

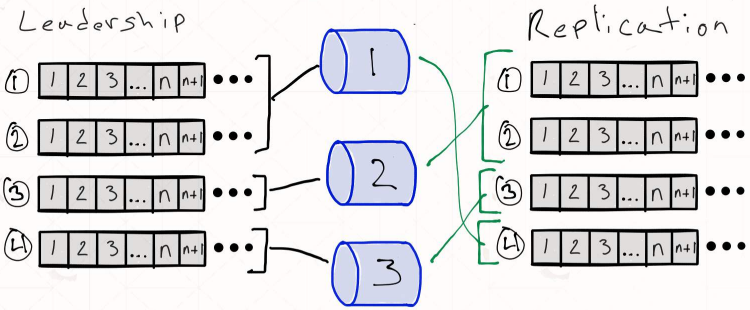
Zookeeper used for:

- leader election
  - partitions
- Configuration
  - Broker
  - Topics
  - Consumers

Topic Partitions



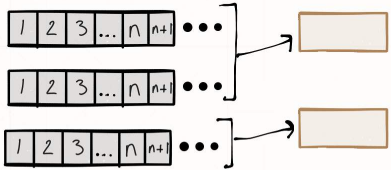
# Clusters, Replication



# Consumers

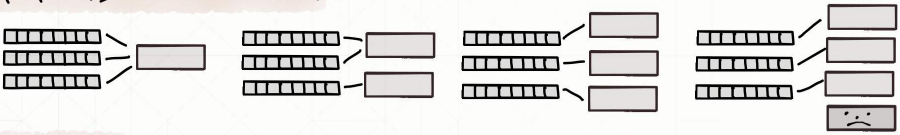
Grouped by group id

- Many consumers per group
- Consumer count <= # of partitions
  - Extra will be idle
- Each consumer in group assigned 1 or more partions until all are assigned.
- Automatic partition reassignment on consumer count change

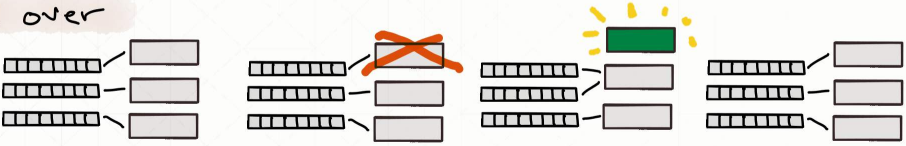


# Consumers

## Partitions to consumers

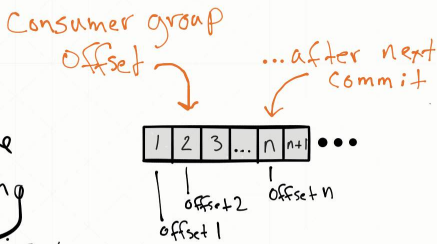


## fail over



# Consumers

Data is never "Consumed" it remains on the broker according to retention policy

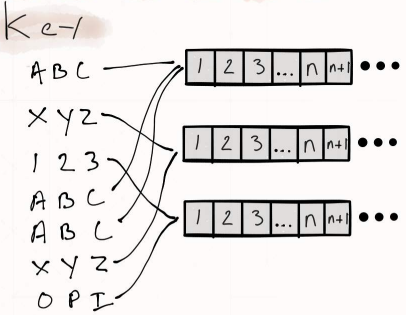


# Producers

Keyed/unkeyed messages
- Same key always goes to same partition.*

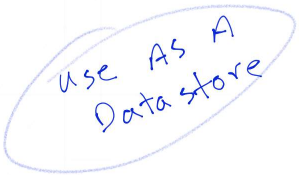
Configurable delivery guarantees
- ACK 0 - no guarantee
- ACK All - all replicants received
- ACK N - N replicants received

Attempts to batch for increased throughput
- Configurable

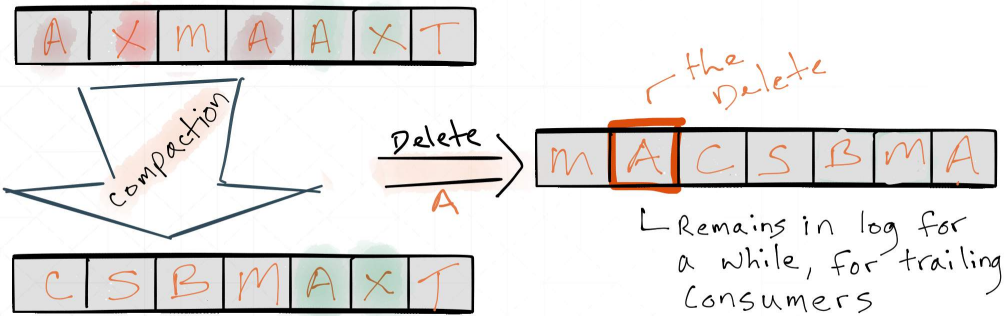


# Log Compaction

- Based on keyed messages
- Always retains last message for a key
- Empty payload removes key

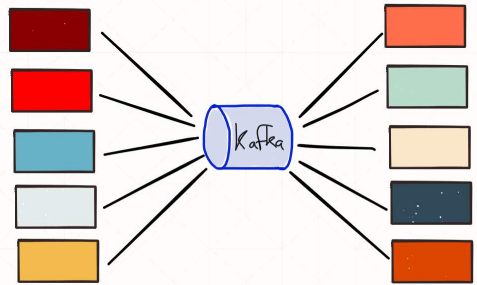


# Log Compaction



# Detangling

Replace P-2-P with an event hub.

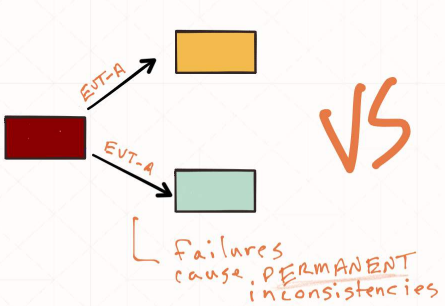



Scale topics or consumers as needed

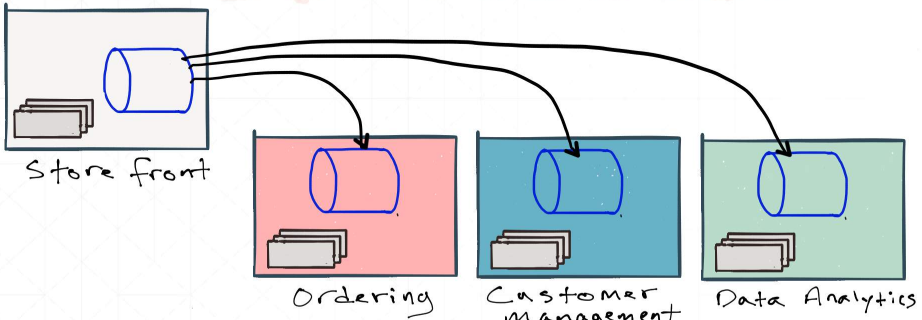
New services can consume the data they need.

Create custom DBs and caches suited to thier domain.

# Detangling


EVT-A
EVT-A
VS
failures cause PERMANENT inconsistencies
EVT-A
EVT-A
Kafka
EVT-A
Eventually Consistent

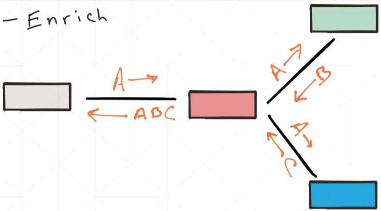
# System Integrations



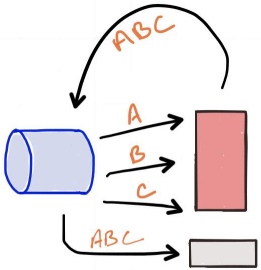
# DC Replication

# Stream Processing

- Many services are essentially stream processors
  - Transform
  - Enrich



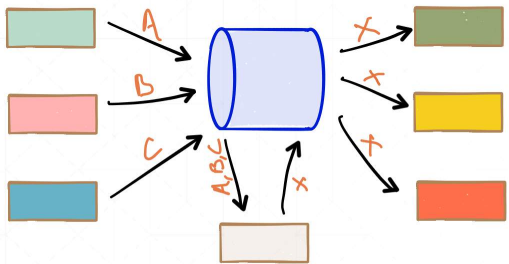

VS



# Stream Processing

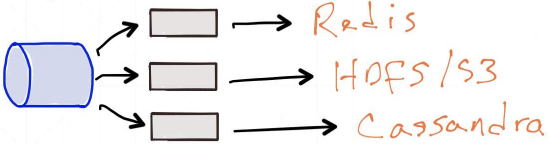
Transform messages into common formats centrally.

Speed up Integrations



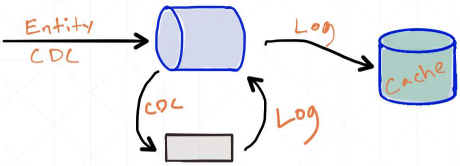
# Stream Export/Import

- Export into stores
  - Databases
  - Caches
  - Object stores
- Import from stores
  - Databases
    - mysql
    - Postgres
    - etc
  - filesystems



# Streams to Caches

— Compacted log (all data)

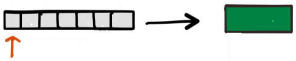



- Cache always up to date
- Load new cache by reading from start of Log
- Same for new DB (no more schema migrations)
- Small datasets can be in Process Memory

# Message Replay

- Great when things go sideways
- Can replay all retained messages or from any offset


fix and Replay



# Things to keep in Mind

- Data formats
  - smaller better (bandwidth)
  - well defined (schema)
  - Managed evolution
- Consumers
  - idempotent

# Things to keep in Mind

- Message guarantees
  - at most once
  - at least once
  - exactly once
    - requires consumer integration

You will get Dupes

# Things to keep in Mind

- Dead letters / Retry
  - Don't skip messages
  - poison Bad Data
  - fix broken services
  - replay if necessary

# Things to keep in Mind

- Kafka / Zookeeper configuration
  - Kafka can be CP or AP (CAP)
  - over provision partitions
    (Don't go nuts!)
    - repartitioning breaks ordering temporarily

# Things to keep in Mind

## Monitoring is critical!

- Brokers
- Zookeeper
- Consumers (Lag)
  - Burrow can help

# Questions?

john.holland@objectpartners.com
@jhollandus