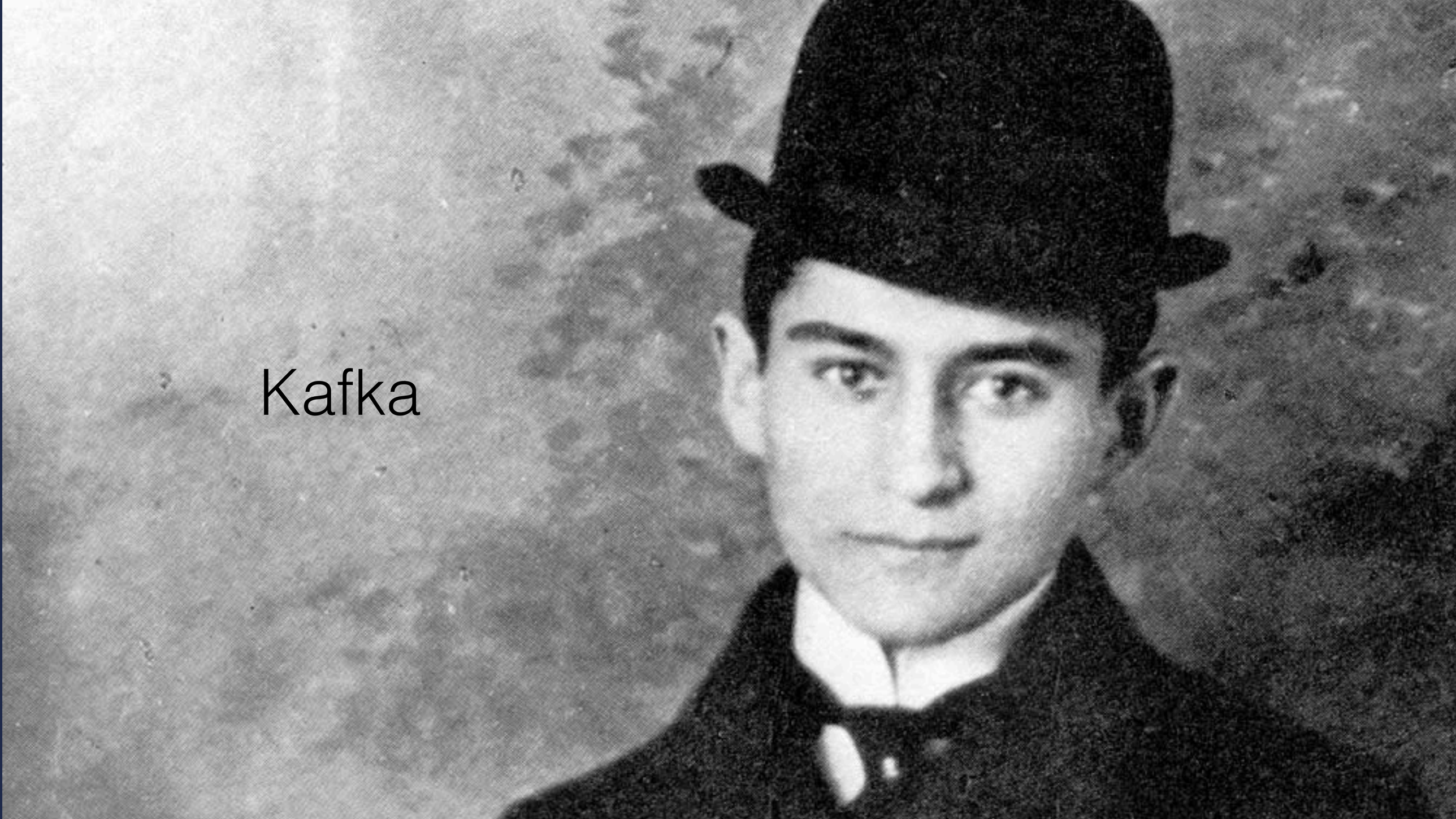
Kafka



Plan

- Kafka basics
- Demo
- Kafka internals

LinkedIn

confluent

2013

Apache Kafka is a distributed publish-subscribe messaging system

2014 – 2015

Apache Kafka is a distributed, partitioned, replicated commit log* service

*an ordered, immutable sequence of messages

2016 – now

Apache Kafka is a distributed streaming platform

Distributed streaming platform
lets you…

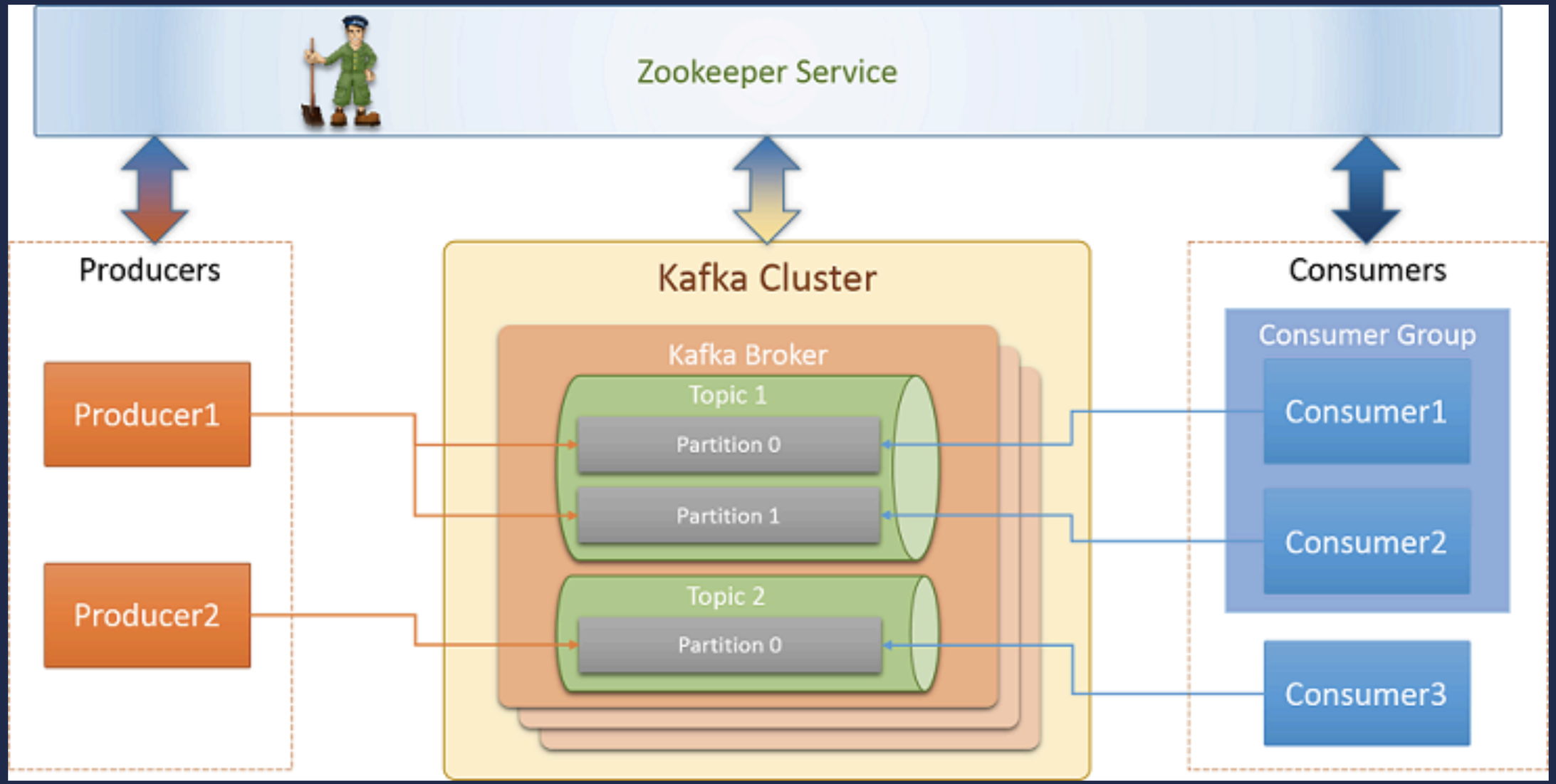
- publish and subscribe to streams of records
- store streams of records in a fault-tolerant way
- process streams of records

Key concepts

- Kafka is run as a cluster on one or more servers (a.k.a brokers)
- The Kafka cluster stores streams of records in categories called topics
- Each record consists of a key, a value, and a timestamp

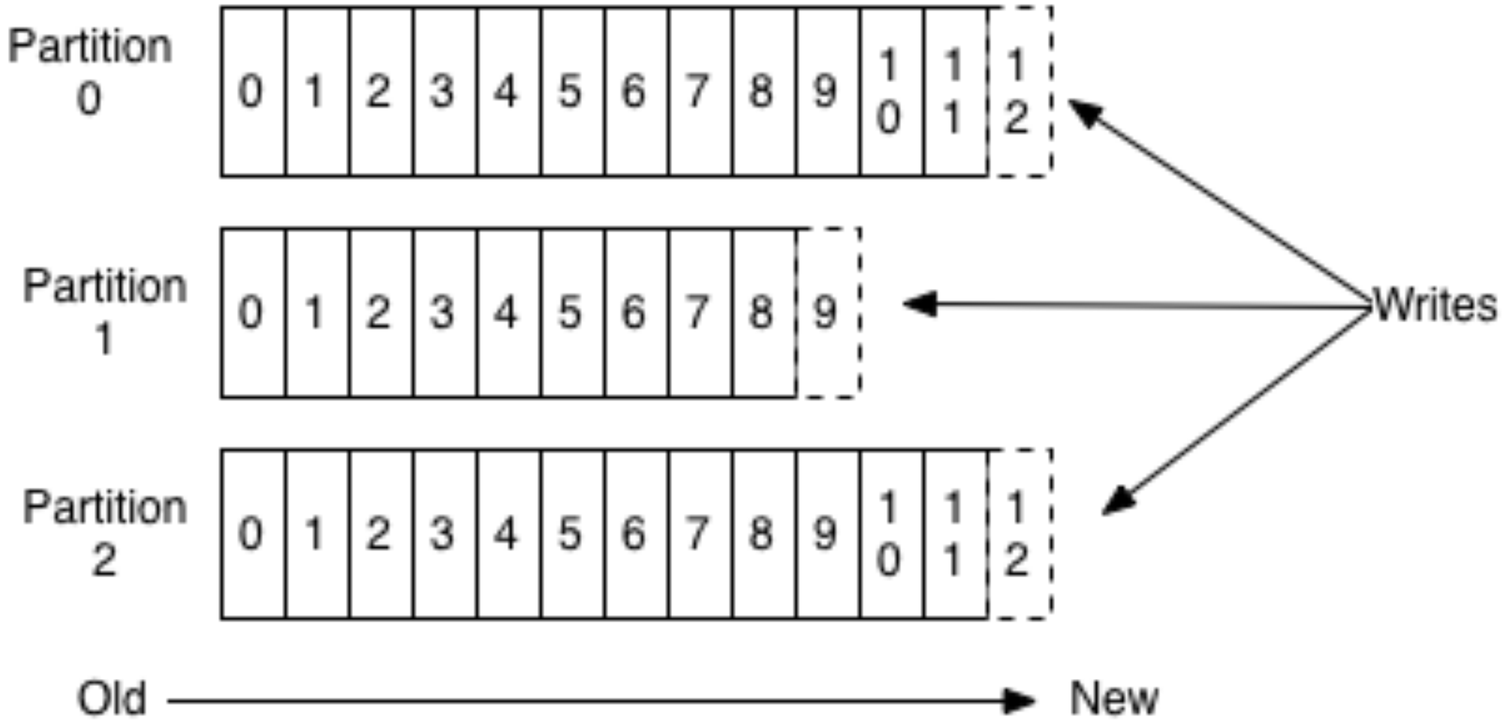
Core APIs

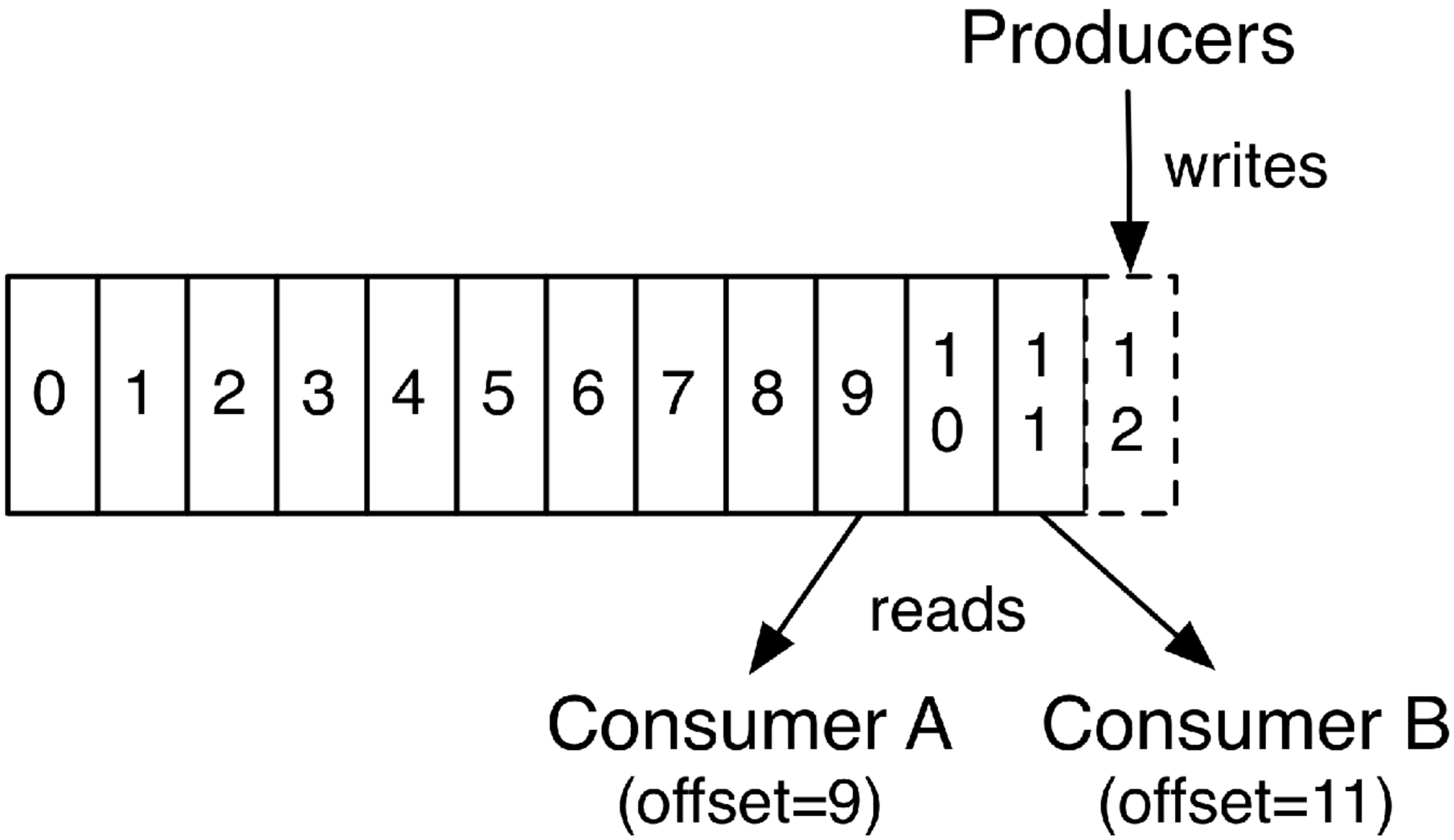
- Producer API
- Consumer API
- Streams API
- Connector API


Zookeeper Service
Producers
Kafka Cluster
Consumers
Producer1
Producer2
Kafka Broker
Topic 1
Partition 0
Partition 1
Topic 2
Partition 0
Consumer Group
Consumer1
Consumer2
Consumer3

Topics and Logs

Anatomy of a Topic



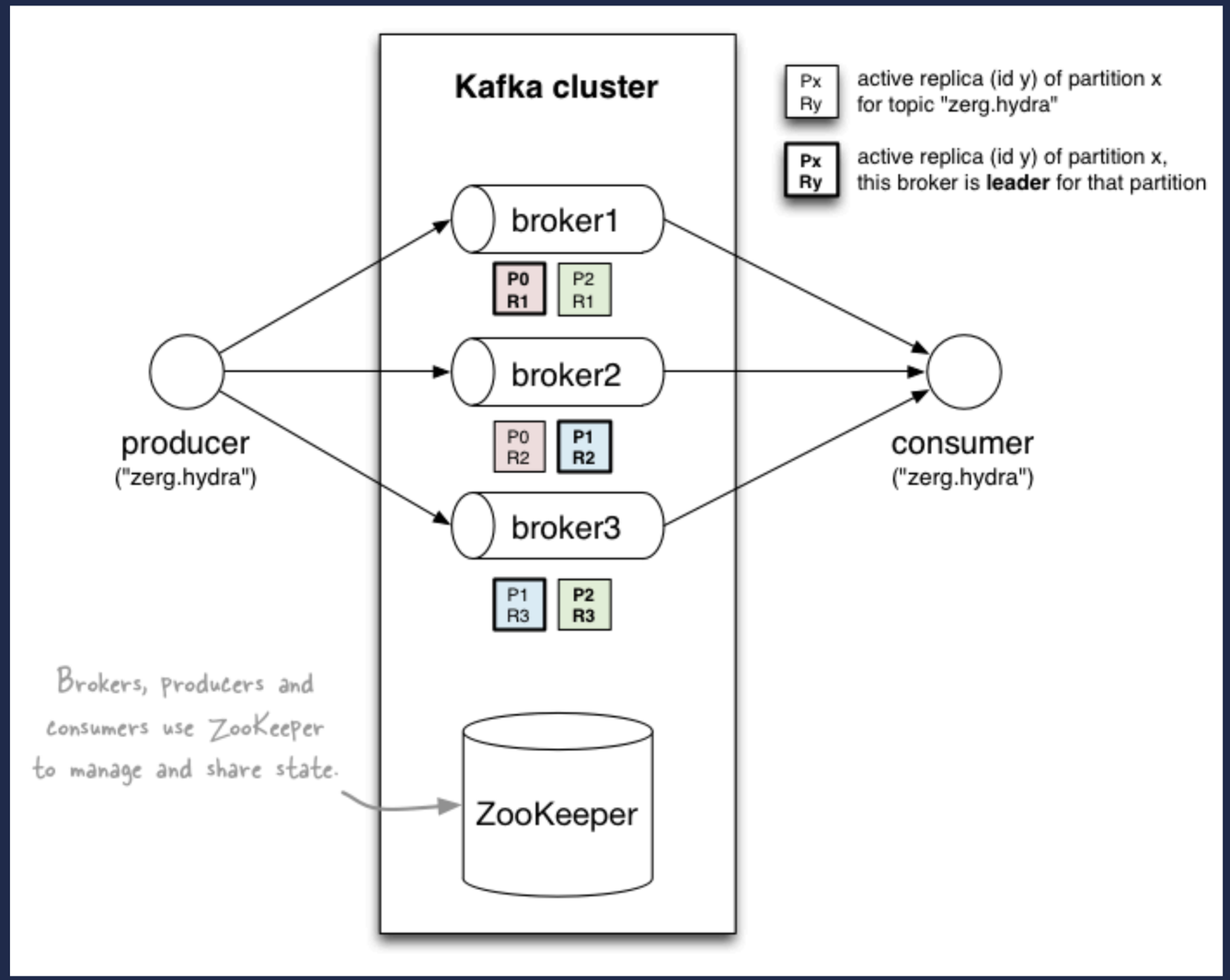

Producers
writes
0
1
2
3
4
5
6
7
8
9
10
11
12
reads
Consumer A
(offset=9)
Consumer B
(offset=11)

Replication

Replication

- Each partition has one server which acts as the "leader" and zero or more servers which act as "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

Topic with partitions=3 and replicas=2

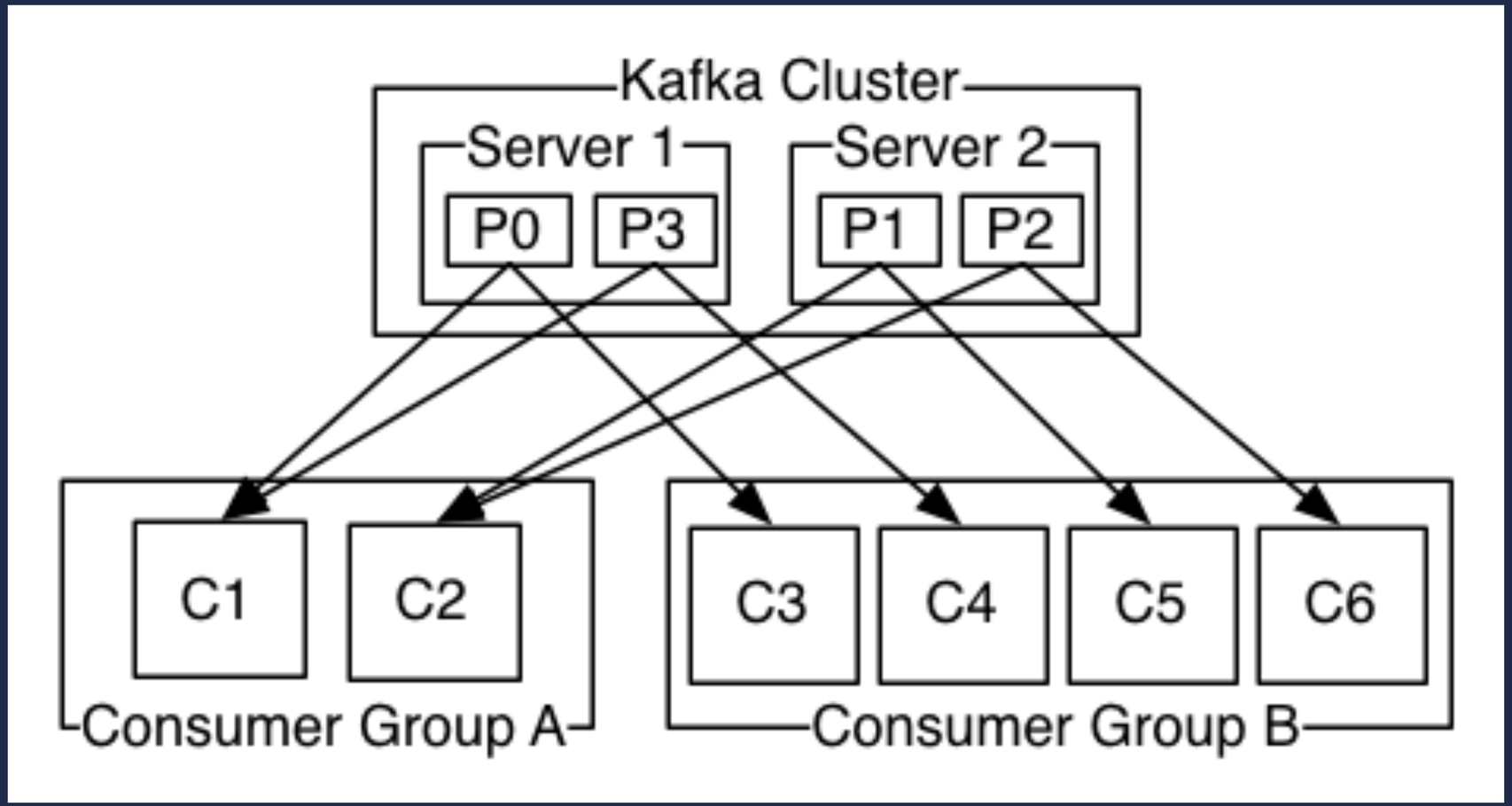


Producers

- publish data to the topics of their choice
- partitioning (round-robin, hash, based on key etc.)
- sync/async ack
- # of replicas in-sync

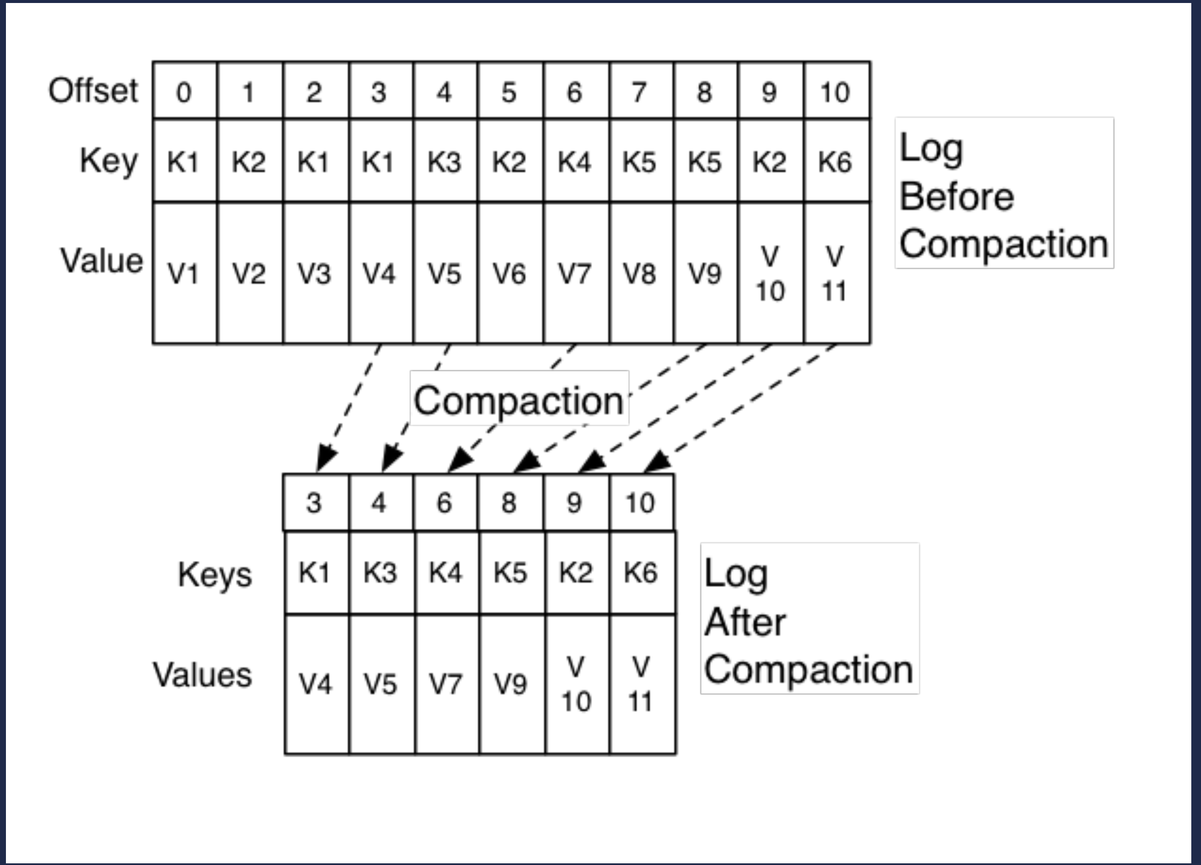
Consumers

- consumer groups
- if new instances join the group they will take over some partitions from other members of the group; if an instance dies, its partitions will be distributed to the remaining instances
- # of consumers in consumer group <= # of partitions in topic
- offsets


Kafka Cluster
Server 1
Server 2
P0
P3
P1
P2
C1
C2
C3
C4
C5
C6
Consumer Group A
Consumer Group B

Consumer offset

Log compaction



Guarantees

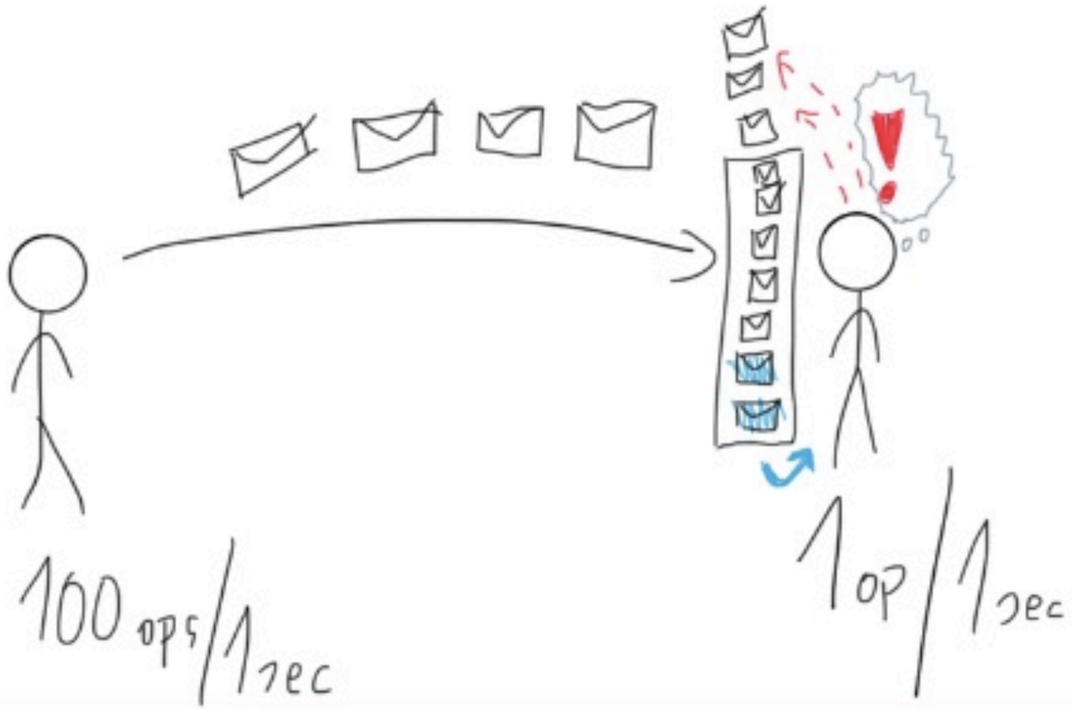
- Messages sent by a producer to a particular topic partition will be appended in the order they are sent
- A consumer instance sees records in the order they are stored in the log
- For a topic with replication factor N, Kafka will tolerate up to N-1 server failures without losing any records committed to the log

Kafka as a messaging system…

…handles back-pressure

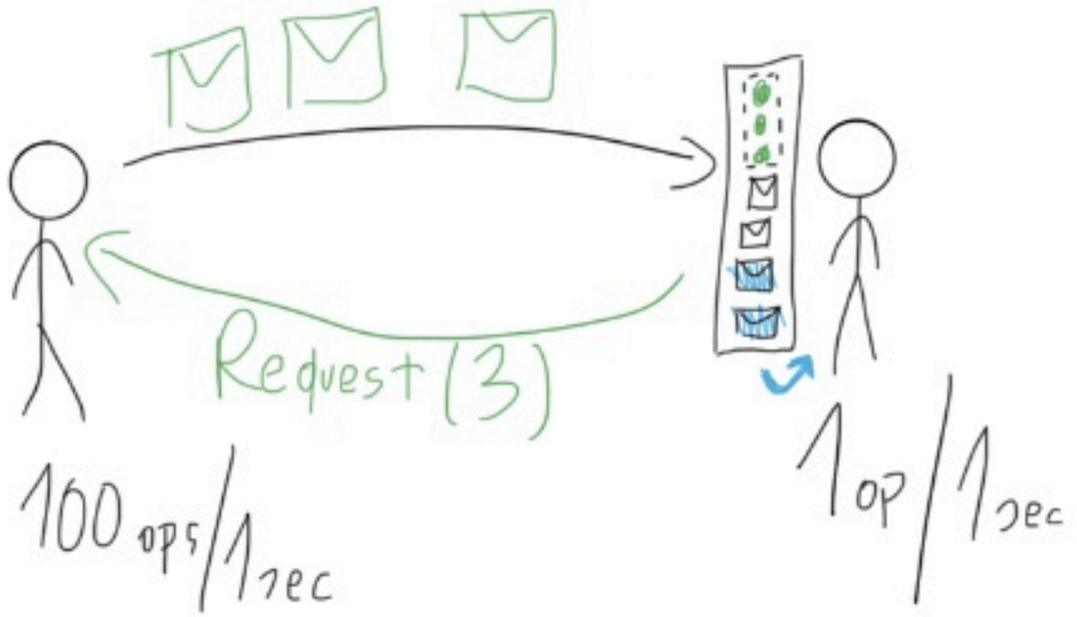
Back-pressure? Push + NACK model (b)

Increase buffer size…
Well, while you have memory available!



Back-pressure? RS: Dynamic Push/Pull

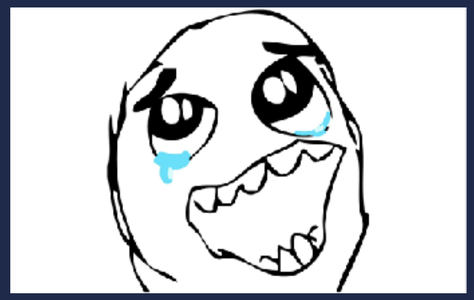
Fast Publisher will send at-most 3 elements.
This is pull-based-backpressure.



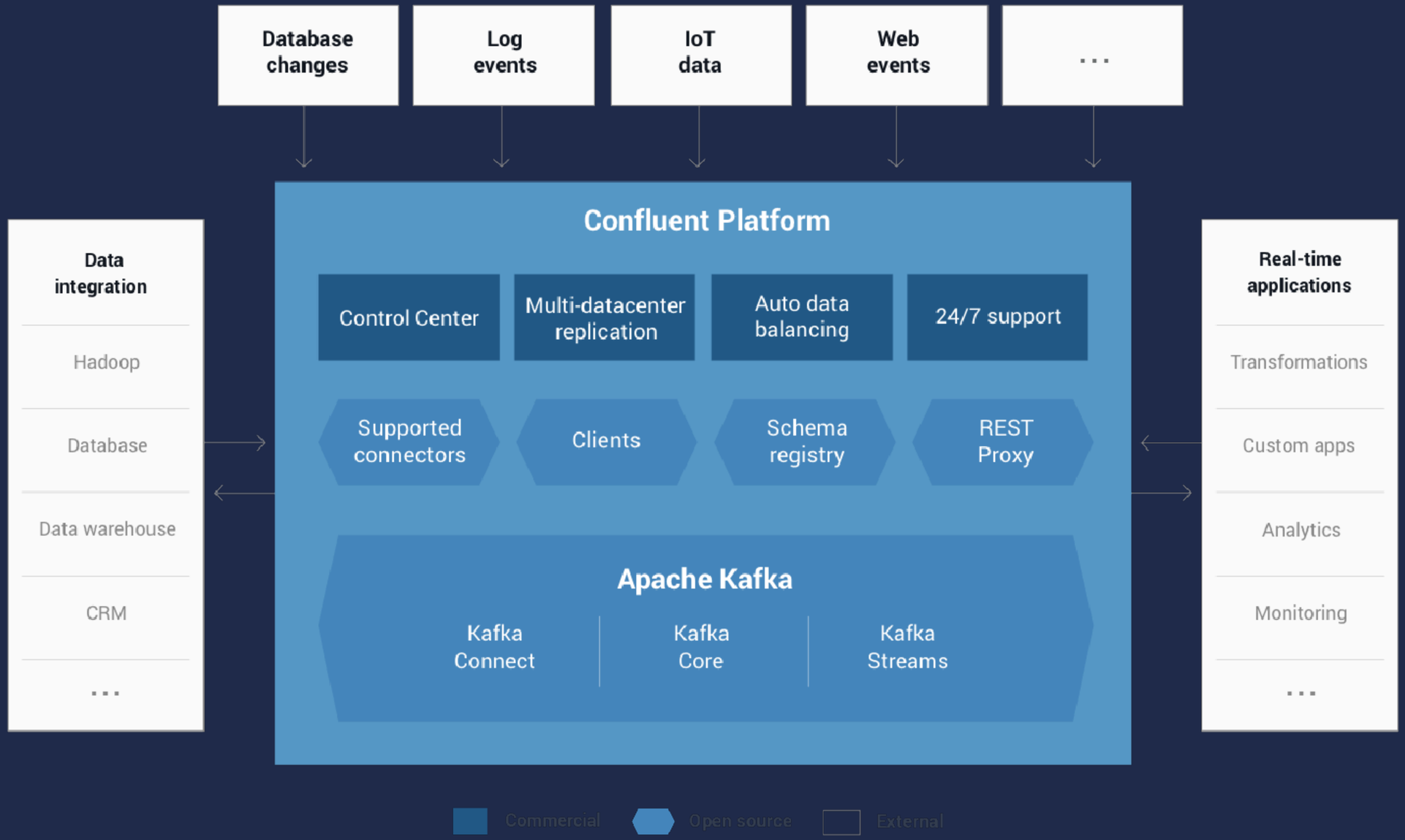
…is also a storage system

…is also a stream processor

Kafka is a backbone of Kappa architecture.
It can treat both past and future data the same way!




confluent


Database changes
Log events
IoT data
Web events
...
Confluent Platform
Control Center
Multi-datacenter replication
Auto data balancing
24/7 support
Supported connectors
Clients
Schema registry
REST Proxy
Apache Kafka
Kafka Connect
Kafka Core
Kafka Streams
Data integration
Hadoop
Database
Data warehouse
CRM
...
Real-time applications
Transformations
Custom apps
Analytics
Monitoring
...
Commercial
Open source
External

Demo

Producer example

```
from confluent_kafka import Producer

p = Producer({'bootstrap.servers': 'mybroker,mybroker2'})

for data in some_data_source:
    try:
        p.produce('mytopic', data.encode(‘utf-8'))
        p.poll(0)
    except BufferError as e:
        p.poll(10)
        p.produce('mytopic', data.encode(‘utf-8'))

p.flush()
```

Consumer example

```
from confluent_kafka import Consumer, KafkaError

c = Consumer({'bootstrap.servers': 'mybroker', 'group.id': 'mygroup','default.topic.config':
{'auto.offset.reset': 'smallest'}})

c.subscribe(['mytopic'])

running = True

while running:
    msg = c.poll()
    if not msg.error():
        print('Received message: %s' % msg.value().decode('utf-8'))
    elif msg.error().code() != KafkaError._PARTITION_EOF:
        print(msg.error())
        running = False

c.close()
```

Persistence

Don't fear the filesystem!

The performance of linear writes on 7200rpm SATA is about 600MB/sec but the performance of random writes is only about 100k/sec—a difference of over 6000X


PAGE CACHE
HD
HISTORY.COM
imgflip.com

A modern operating system provides read-ahead and write-behind techniques that prefetch data in large block multiples and group smaller logical writes into large physical writes.

Why not application cache?

Why not application cache?

- JVM memory overhead
- GC
- needs to be rebuilt after service restart
- OS does it better

All data is immediately written to a persistent log on the filesystem without necessarily flushing to disk. In effect this just means that it is transferred into the kernel's pagecache

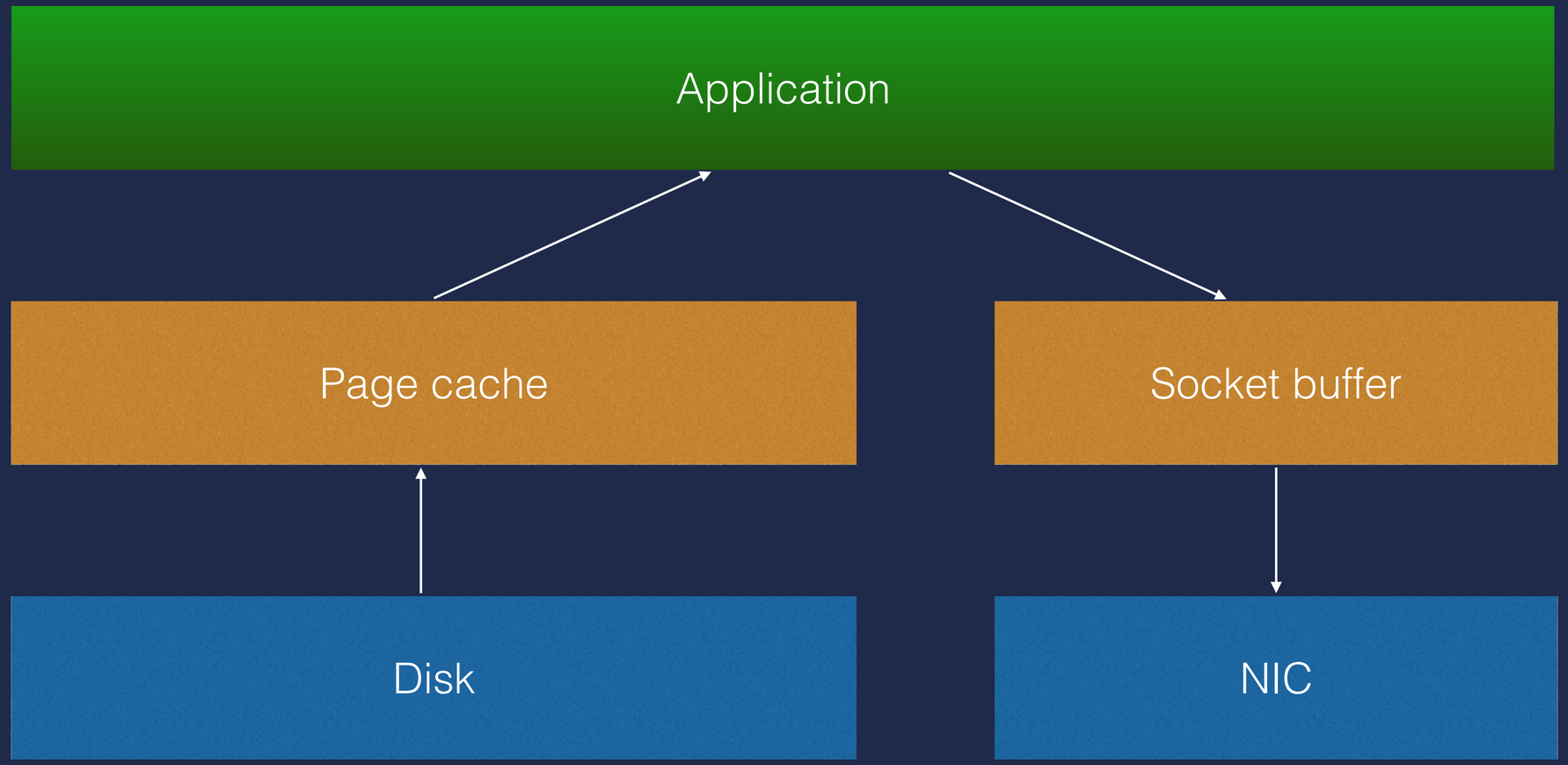
Networking

Networking is based on "message set" abstraction.
Messages are produced and consumed in chunks.

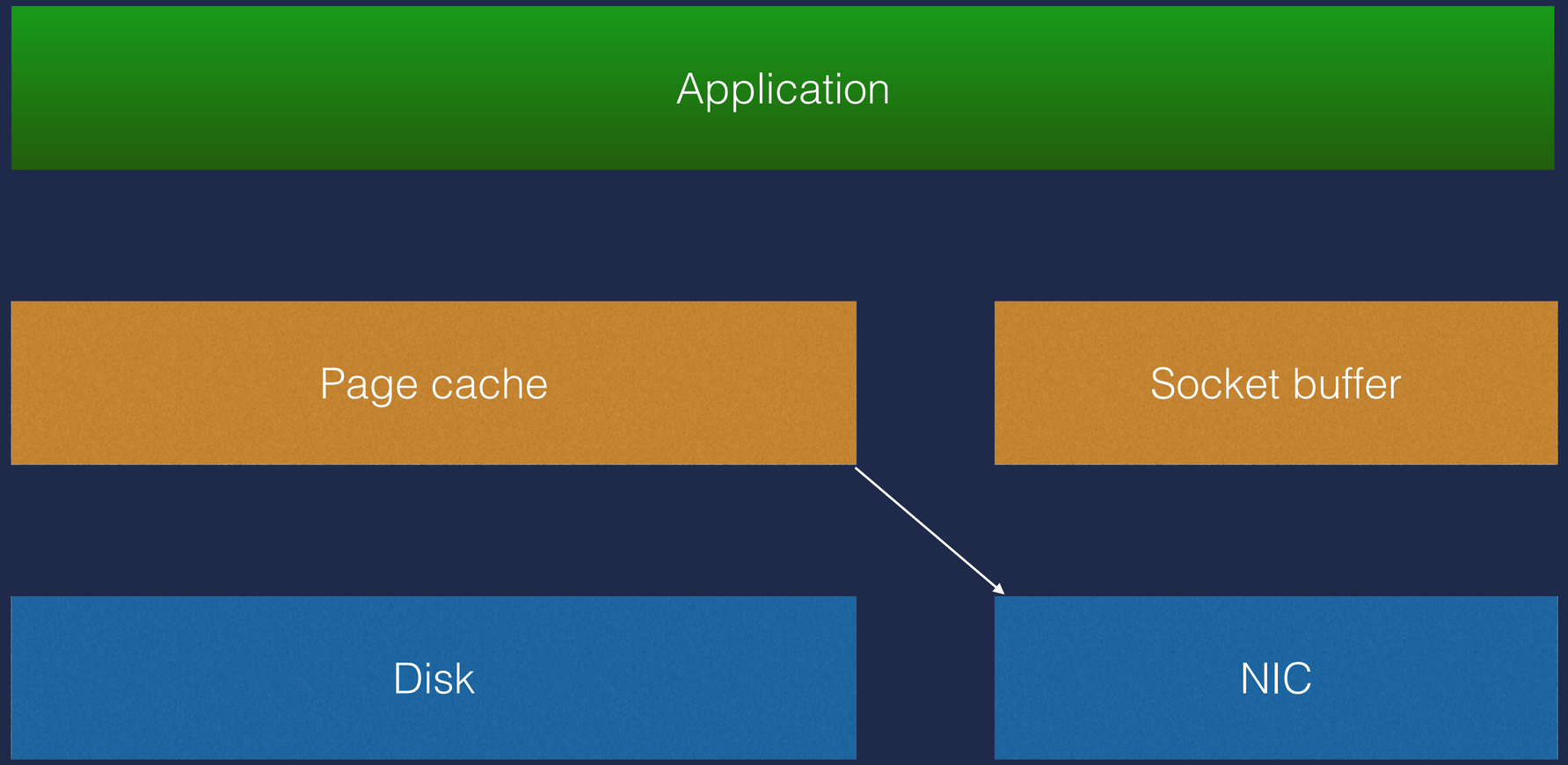
To avoid byte copying producer, broker, and consumer employ a standardized binary message format.

Modern unix operating systems offer a highly optimized code path for transferring data out of pagecache to a socket

Common data path



Kafka data path



Zookeeper

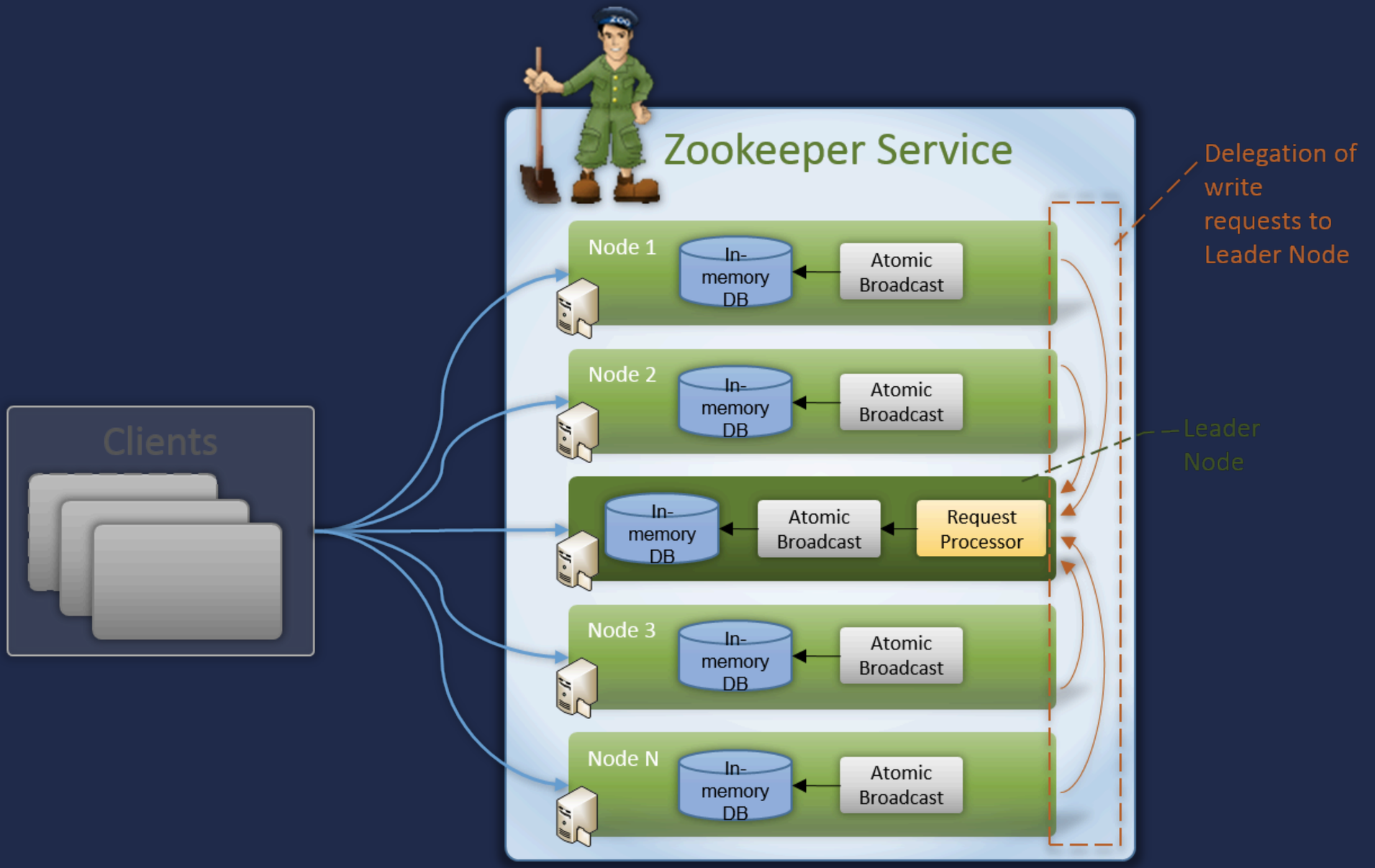
“Because coordinating distributed systems is a Zoo”

Zookeeper

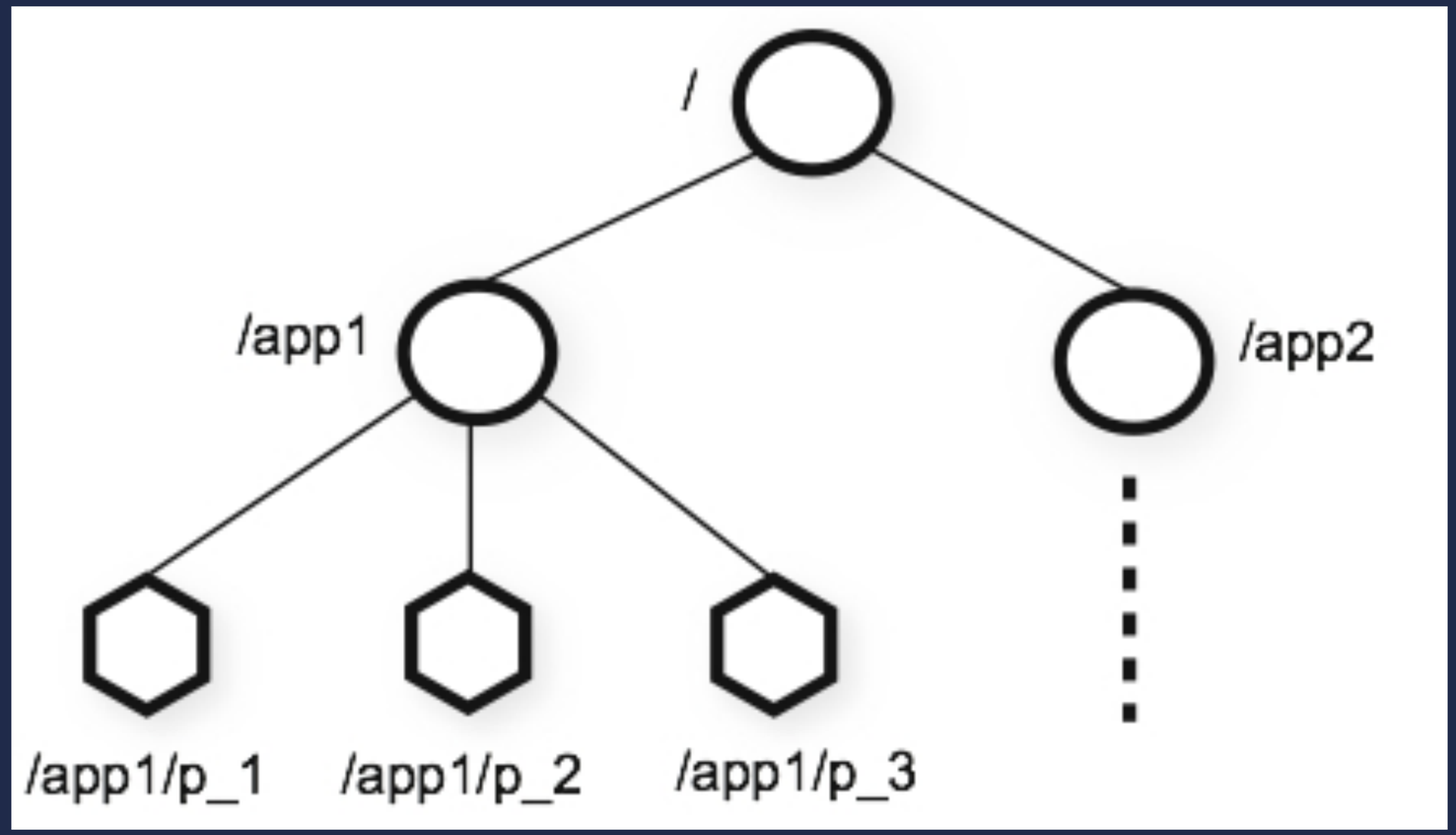
ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

Zookeeper implements

- consensus
- group management
- leader election
- presence protocols


Zookeeper Service
Clients
Node 1
In-memory DB
Atomic Broadcast
Node 2
In-memory DB
Atomic Broadcast
In-memory DB
Atomic Broadcast
Request Processor
Node 3
In-memory DB
Atomic Broadcast
Node N
In-memory DB
Atomic Broadcast
Delegation of write requests to Leader Node
Leader Node

znodes



Kafka znodes

Broker Node Registry

/brokers/ids/[0...N] -->

{"jmx_port":...,"timestamp":...,"endpoints":
[...],"host":...,"version":...,"port":...}

(ephemeral node)

Broker Topic Registry

/brokers/topics/[topic]/partitions/[0...N]/state -->

{"controller_epoch":...,"leader":...,"version":...,"leader
_epoch":...,"isr":[...]}

(ephemeral node)

Consumer Id Registry

/consumers/[group_id]/ids/[consumer_id] -->

{"version":...,"subscription":
{...:...},"pattern":...,"timestamp":...}

(ephemeral node)

Consumer offsets (if used)

/consumers/[group_id]/offsets/[topic]/[partition_id]
-->

offset_counter_value

(persistent node)

Partition Owner registry

/consumers/[group_id]/owners/[topic]/[partition_id]
-->

consumer_node_id

(ephemeral node)

Cluster Id

/cluster/id -->

cluster_id

(persistent node)

Kafka Connect

Bundled Connectors

```
$ bin/confluent list connectors
connect is [DOWN]
Bundled Predefined Connectors (edit configuration under etc/):
  elasticsearch-sink
  file-source
  file-sink
  jdbc-source
  jdbc-sink
  hdfs-sink
  s3-sink
```

Incremental Query Modes

- Incrementing Column
- Timestamp Column
- Timestamp and Incrementing Columns
- Custom Query
- Bulk

KSQL is a streaming SQL engine that enables stream processing against Apache Kafka

Questions