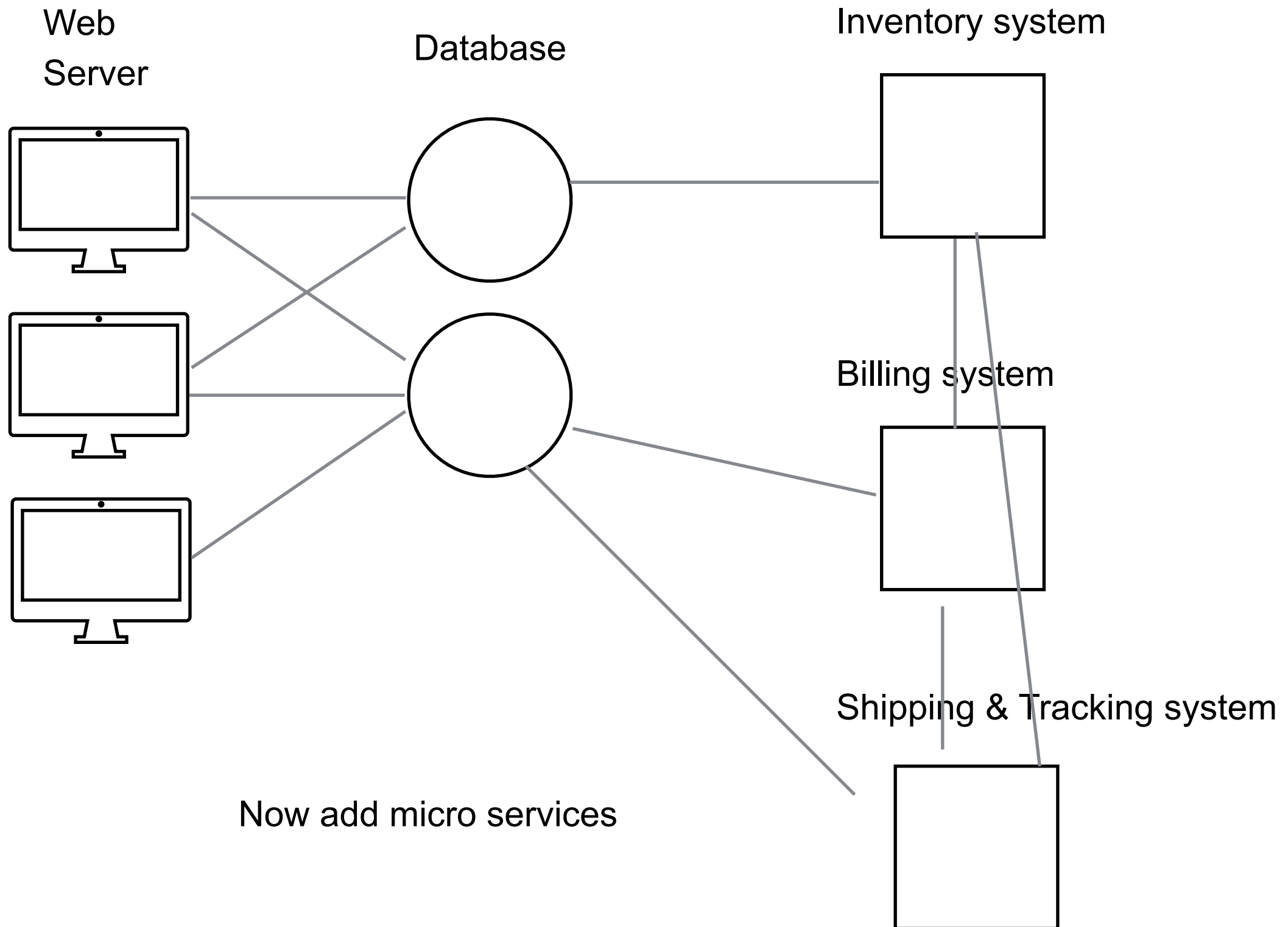


CS 649 Big Data: Tools and Methods  
Fall Semester, 2022  
Doc 26 Kafka  
Apr 19, 2022

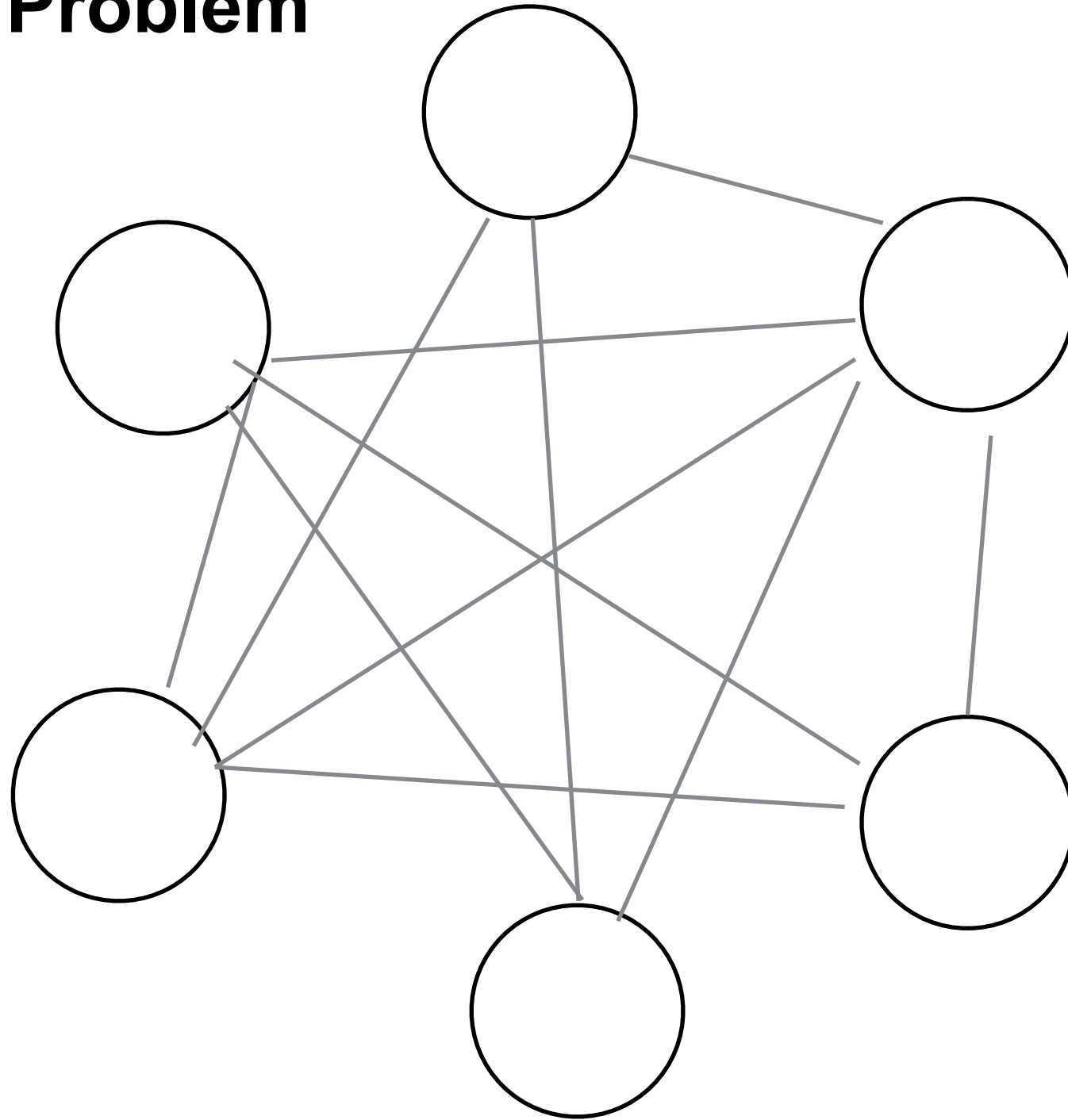
Copyright ©, All rights reserved. 2022 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/openpub/>) license defines the copyright on this document.

# Streaming

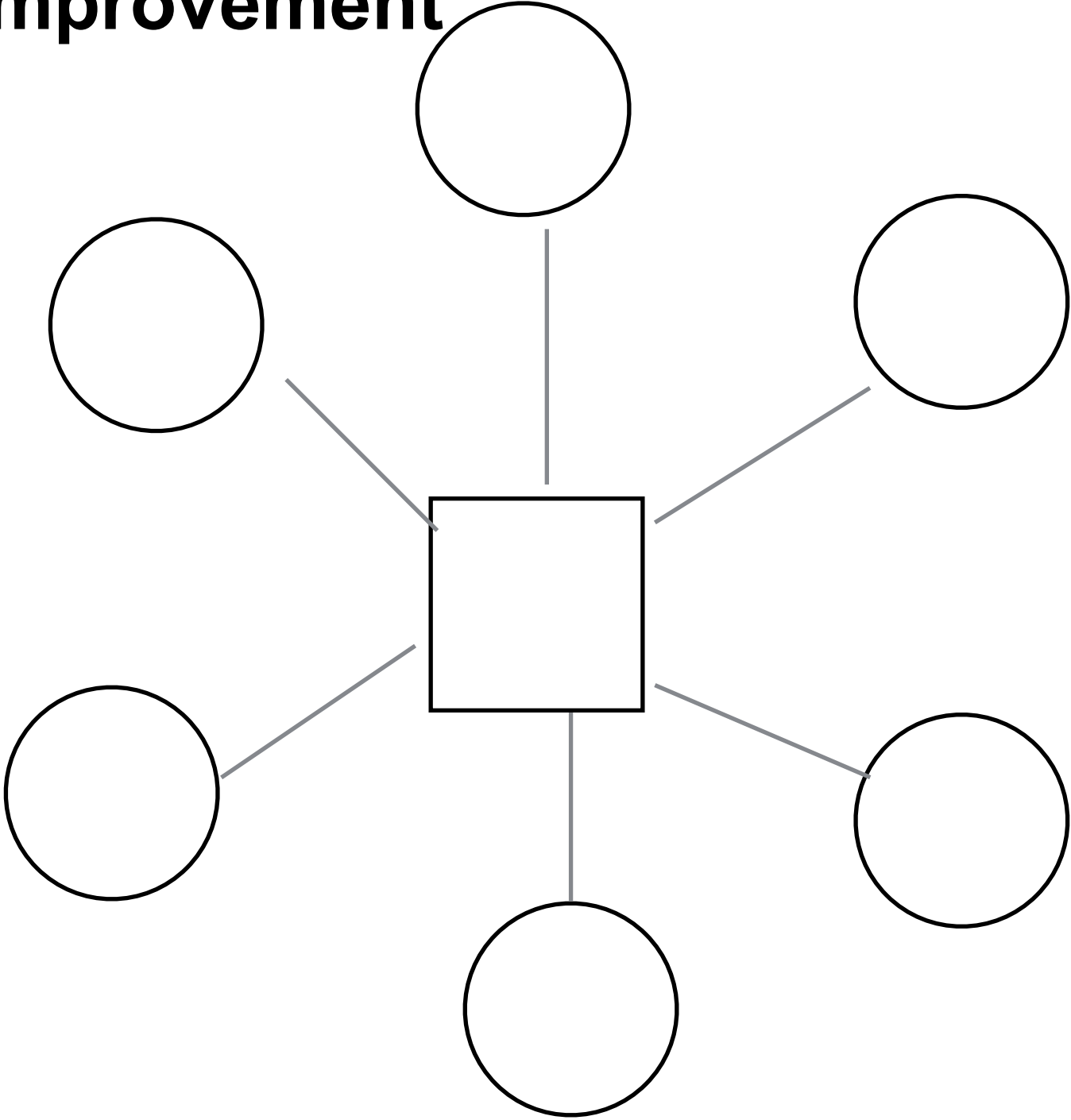
# The Problem



# The Problem



# An Improvement



# The Log

What every software engineer should know about real-time data's unifying abstraction

<https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying>

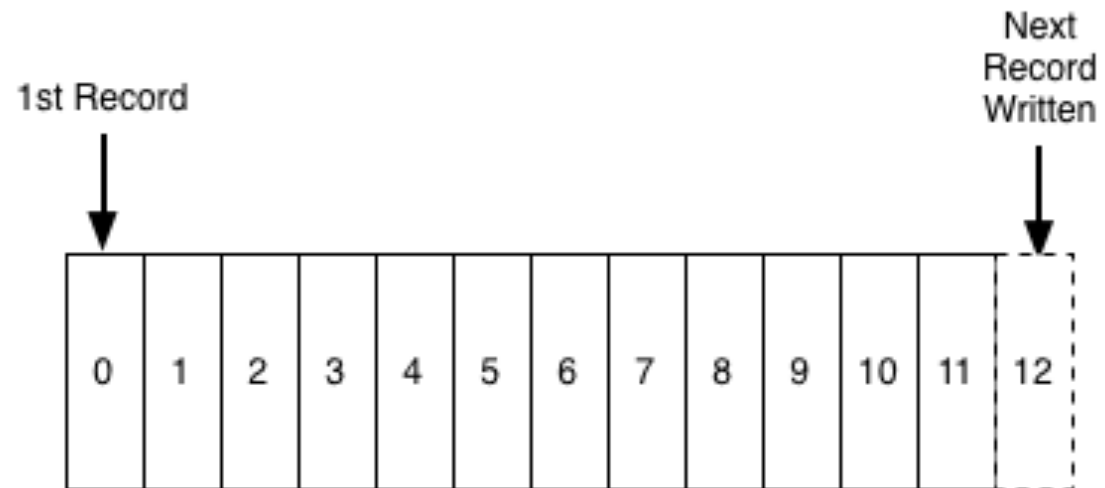
<http://tinyurl.com/qc43s5j>

Jay Kreps

Lead developer of Kafka

Why logs are the basic data structure for distributed computing

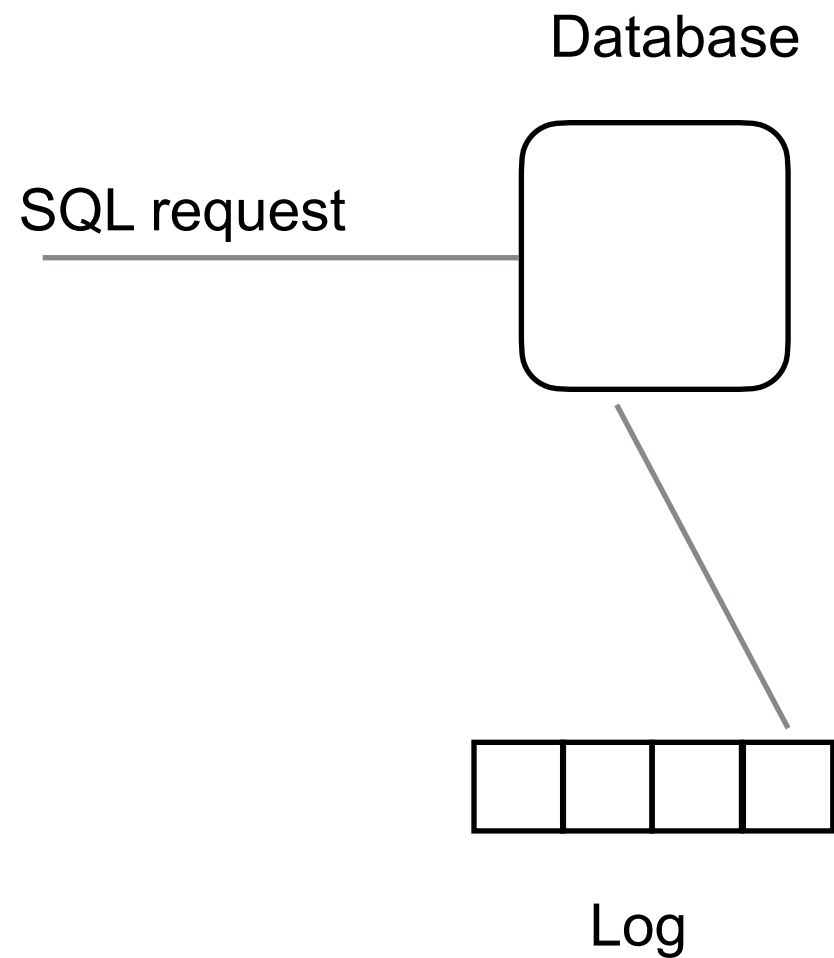
# What Is a Log?



Sequence number becomes timestamp

# Databases and Logs

How to make actions ACID



Log each operation  
Use the log to recover past state

Physical logging  
Log the state of the row that is changing

Logical logging  
Log the SQL statement



# The log contains Database History

With a complete log file we can

- Recreate the current state of the database

- Recreate the database for any time in the past

Used by in-memory databases

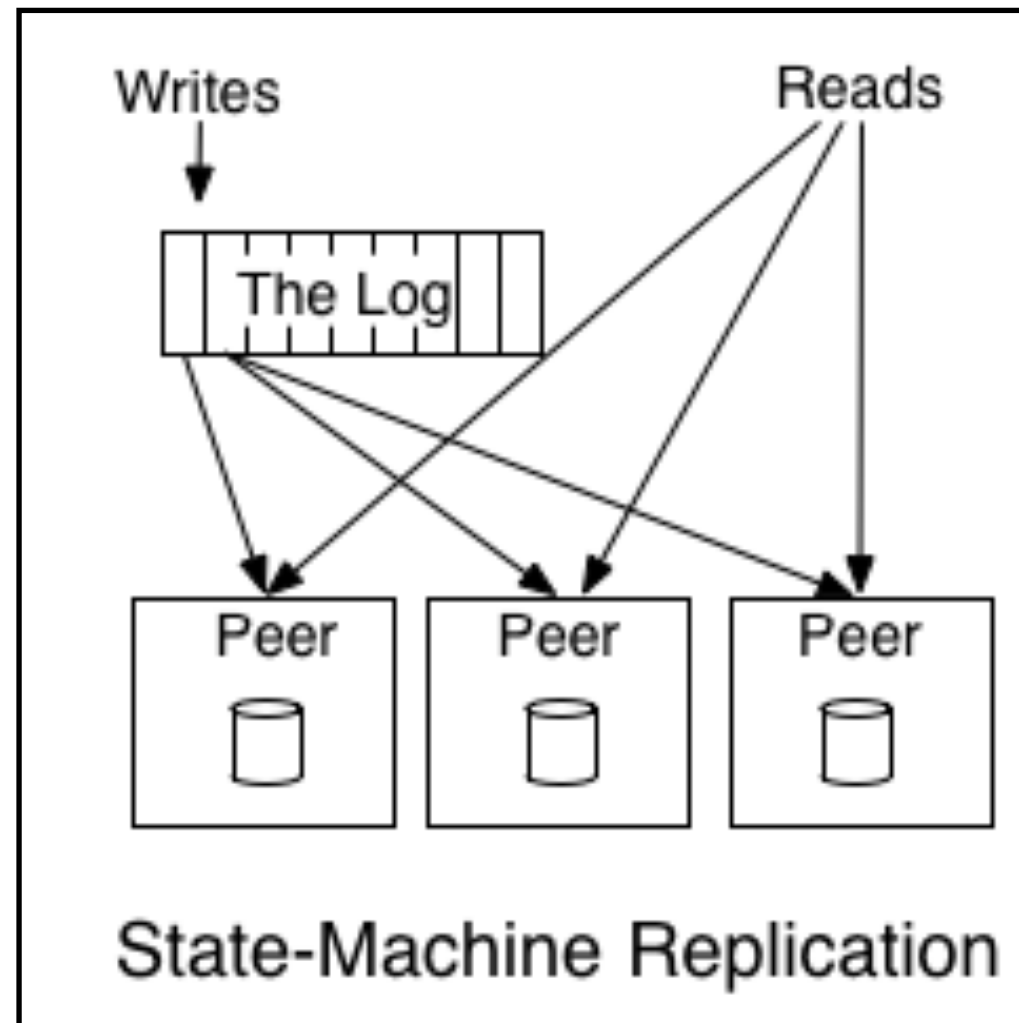
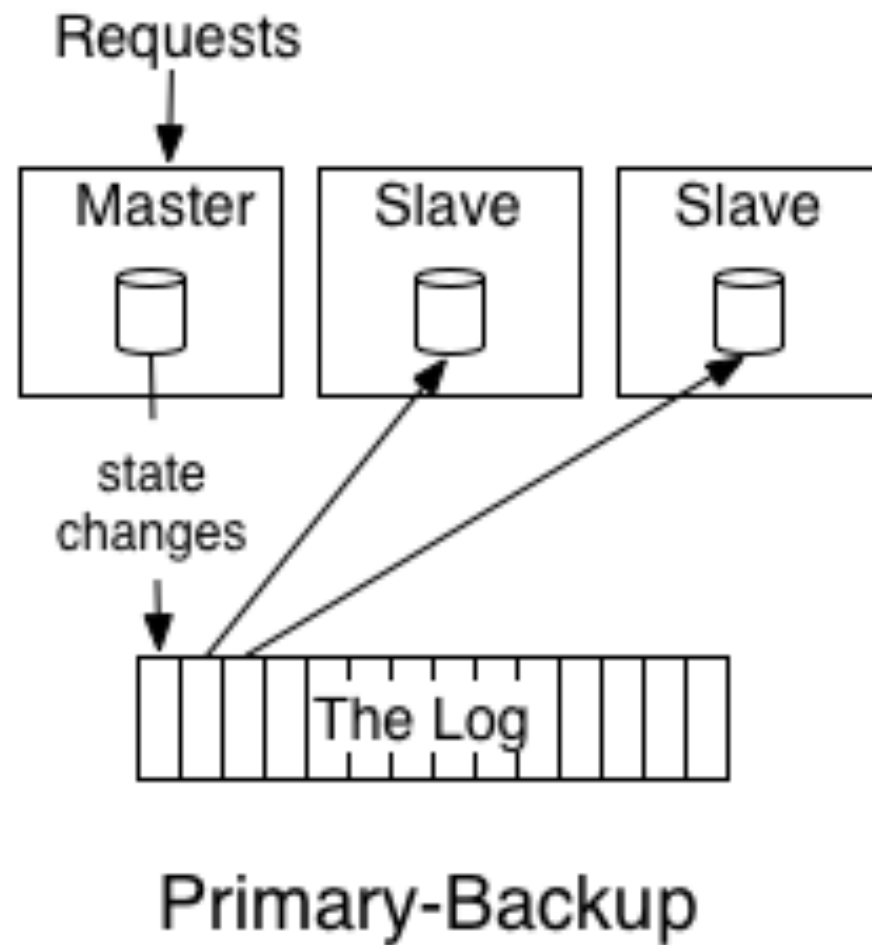
- Keep current state of tables in memory

- Each write operation is logged

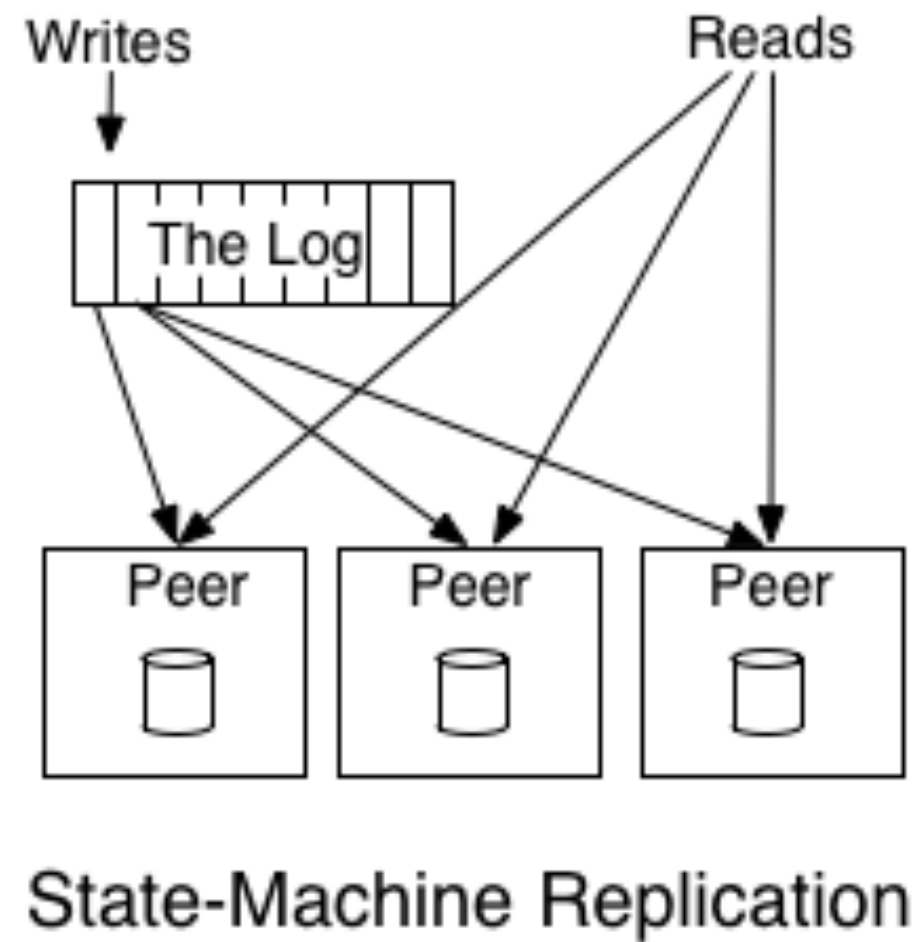
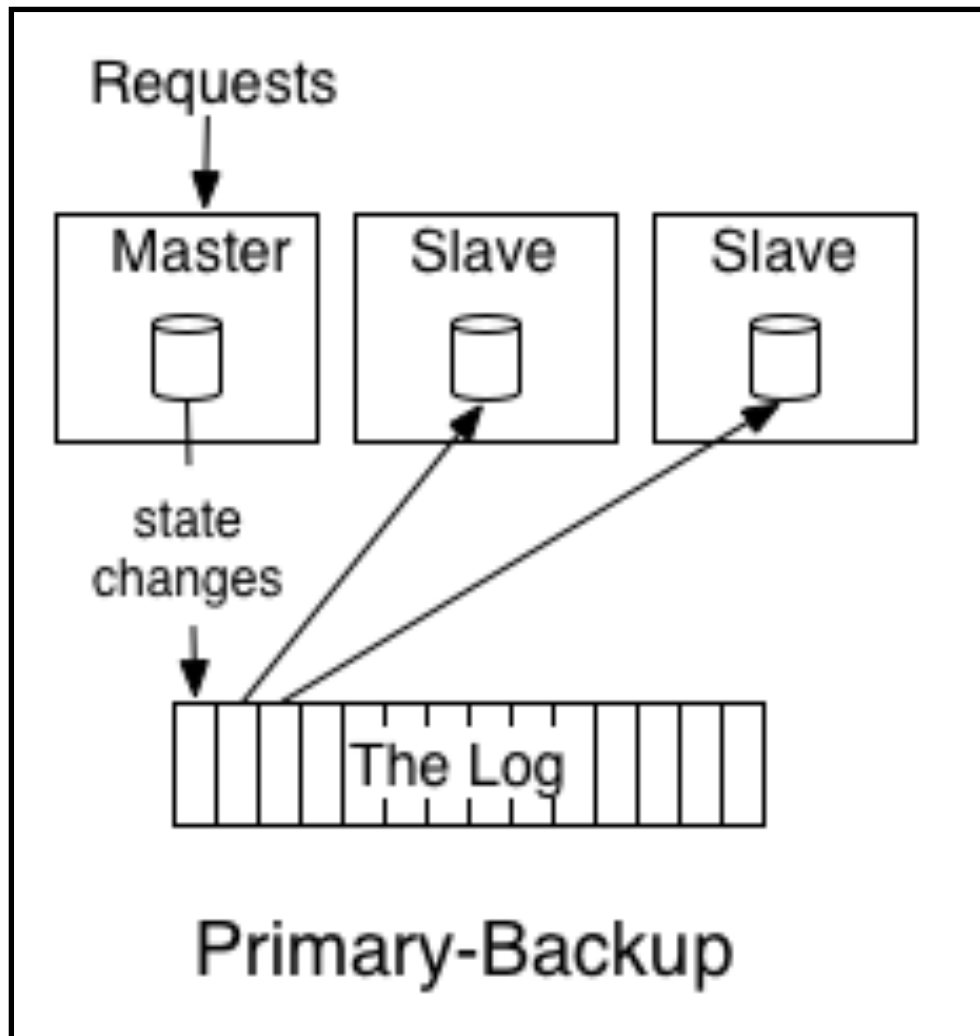
- To restart database replay log file

# Replicating Databases

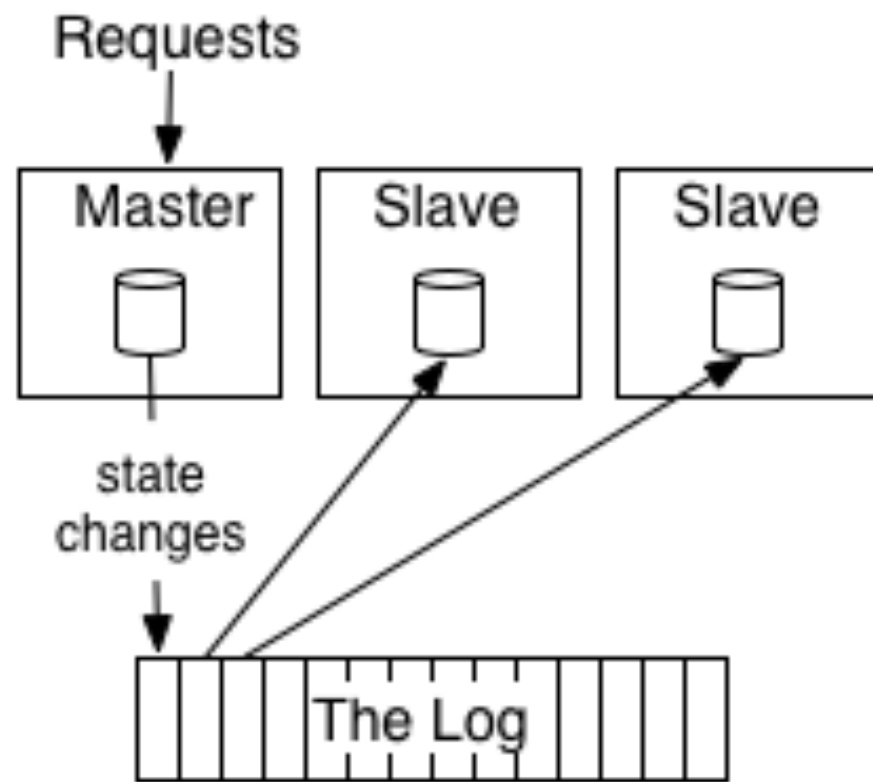
We can use the log to replicate the database



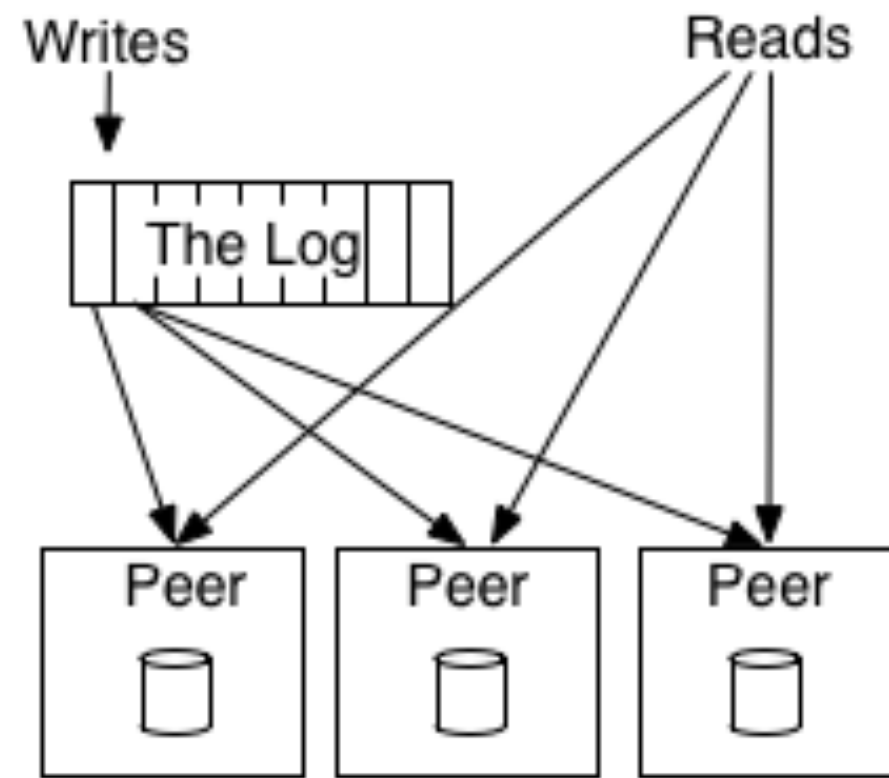
# Inverting the Structure



# Distributed Systems & Replication

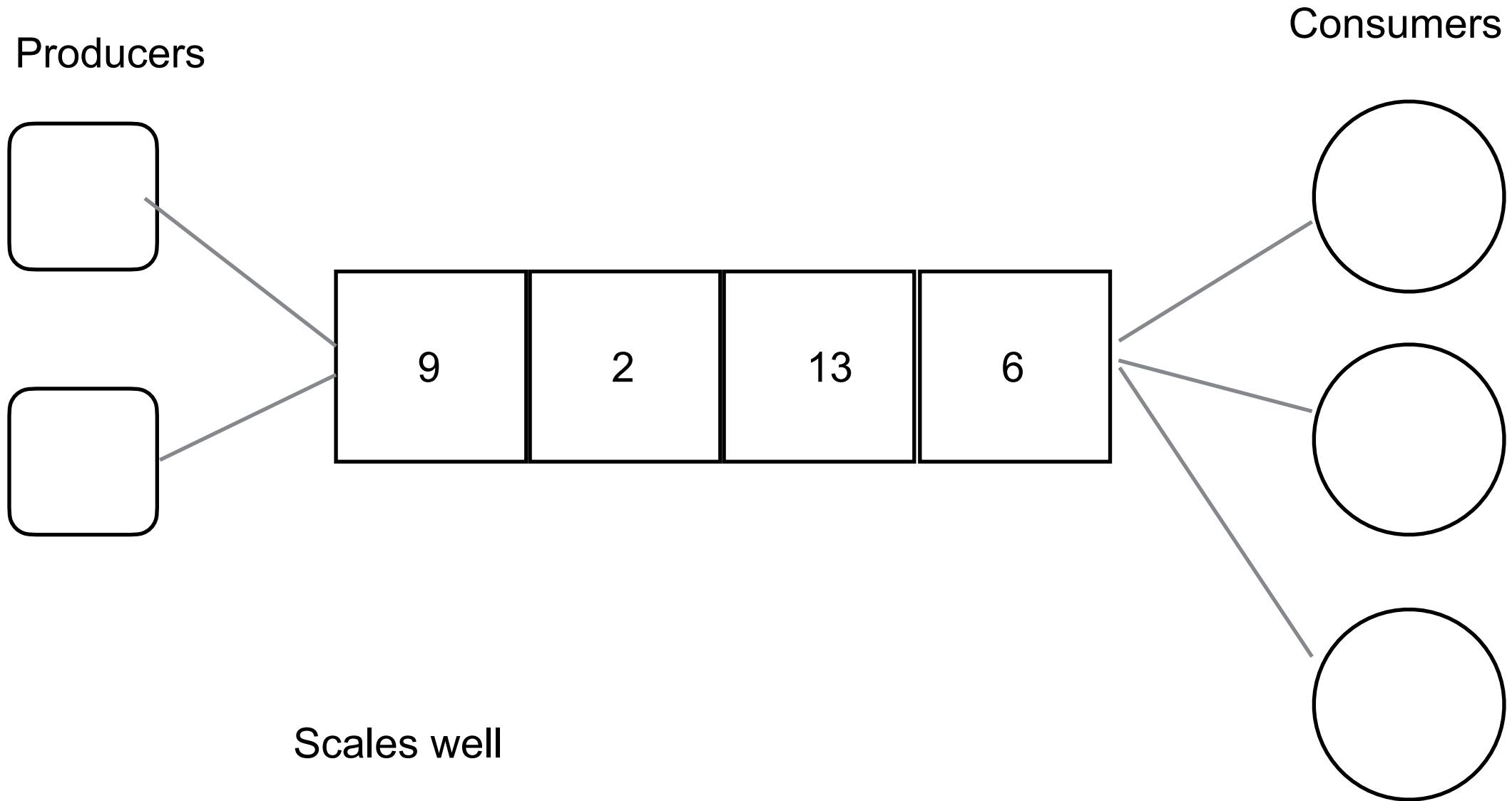


Primary-Backup



State-Machine Replication

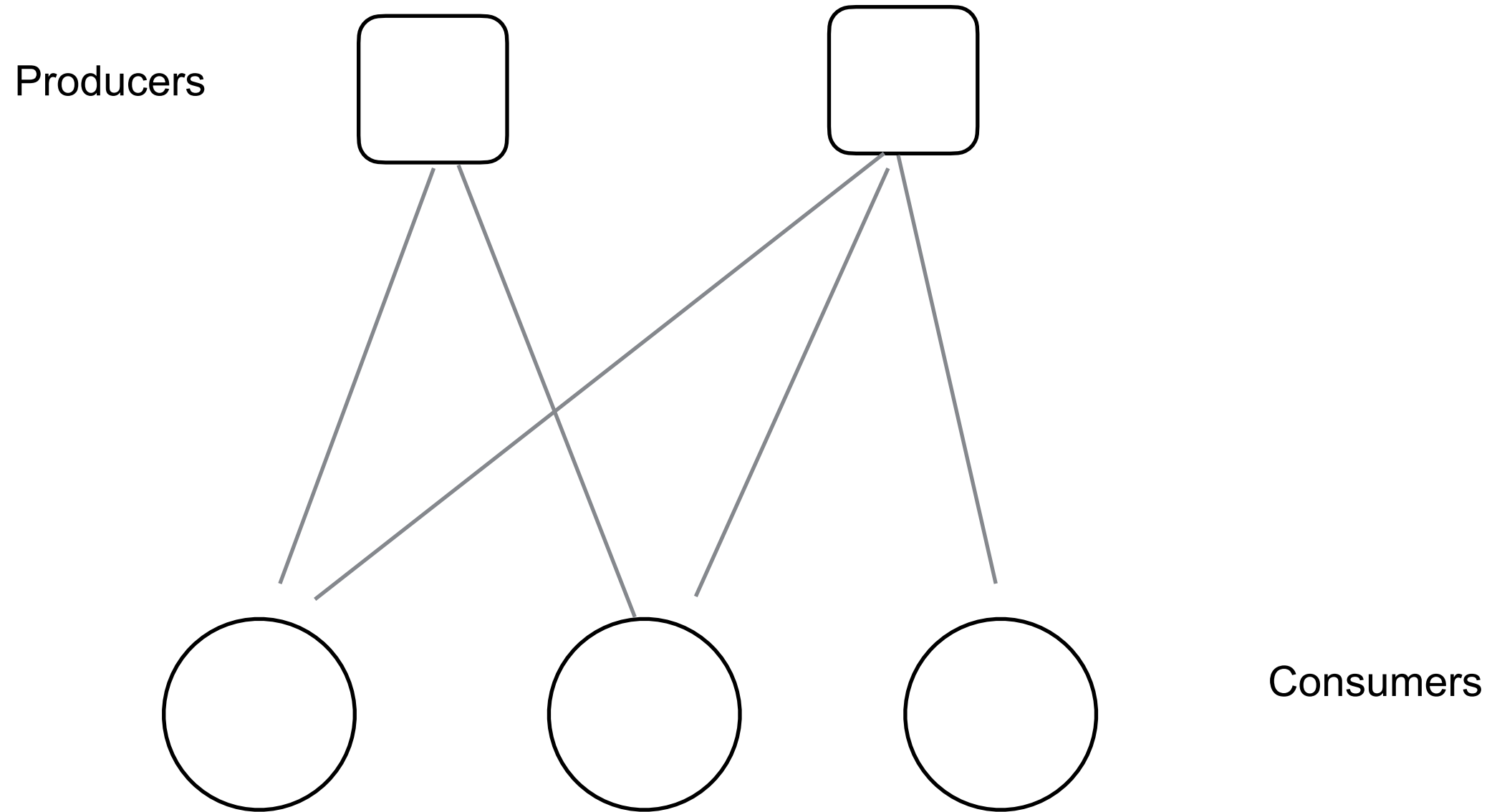
# Message System - Queue



Scales well

No multi subscribers - once read data is gone

# Message System Publish-Subscribe/Broadcast

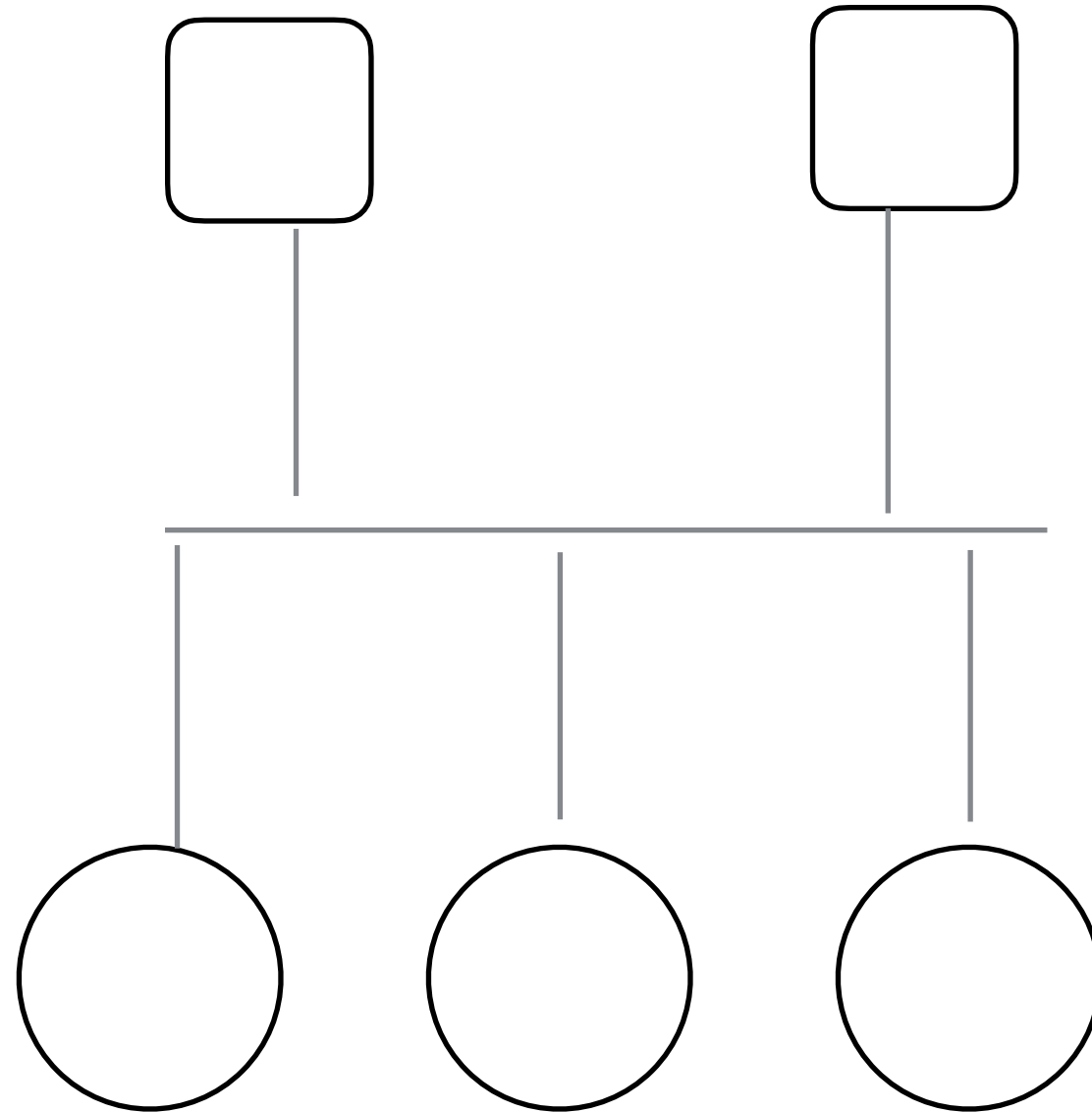


Multi subscribers

Does not scale

# Message System Publish-Subscribe/Broadcast

Producers



Consumers

Multi subscribers

Does not scale

# Message System Issues

Failure

Multi-machine

Load balancing

Redundancy



# Basic Messaging System Issue

How to handle failure

Publisher fails

Consumer fails

Messenger node fails

Messenger system fails

# Message Delivery Semantics

At most once

Messages may be lost but are never redelivered

At least once

Messages are never lost but may be redelivered

Exactly once

What you want

Each message is delivered once and only once

# Apache Kafka

Unified, high-throughput, low-latency platform for handling real-time data feeds

Started at LinkedIn

Named after Franz Kafka - author

Kafka is a writing system

Lead developer liked Kafka's works

Developers left LinkedIn formed Confluent

First release Jan 2011

Version 1.0 Nov 1, 2017

# Few Users of Apache Kafka

Cisco Systems

eBay

IPinYou

Netflix

- 36 Kafka clusters

- 700 billion messages per day

- Good article on problems running Kafka on AWS

The New York Times

- Publish content to applications & system to make it available to readers

Spotify

- Used to send logs from all hosts to Hadoop cluster

- Reduced transfer time from 4 hours to 10 seconds

Uber

Walmart

Yelp

# Related Tools

Apache ZooKeeper

Distributed hierarchical key-value store

KSQL

Streaming SQL for Apache Kafka

# Kafka Versions

1.0.0

Nov 1, 2017

0.11.0.0

June 2017

Guaranteed delivery

0.10.0.0

May 2016

Streams API

1.0.0

Improved Streams API

Better metrics on Kafka performance

Supports Java 9

Faster encryption

Tolerates disk failure better

JBOD broker tolerates one disk failure

Improved throughput on transactions

# Kafka - Key Capabilities

Publish and subscribe to streams of records

Message queue or enterprise messaging system

Store streams of records in a fault-tolerant way

Process streams of records as they occur

Kafka is run as a cluster on one or more servers

The Kafka cluster stores streams of records in categories called topics

Each record consists of a key, a value, and a timestamp



# Core API

## Producer

Allow an application to publish stream of records to 1 or more topics

## Consumer

Allow an application to subscribe to 1 or more topics and read stream of records

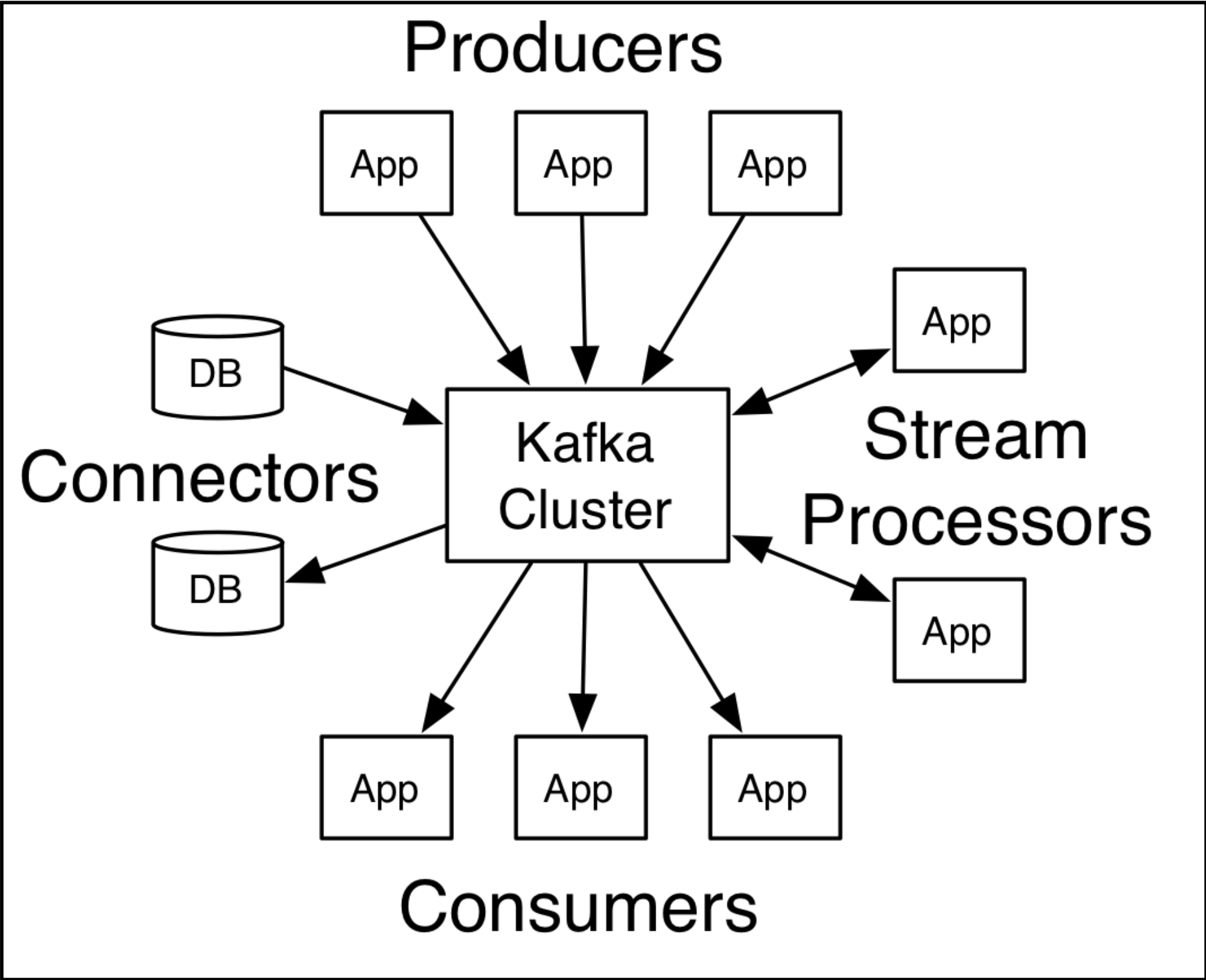
## Streams

An application consume an input stream and produce an output stream

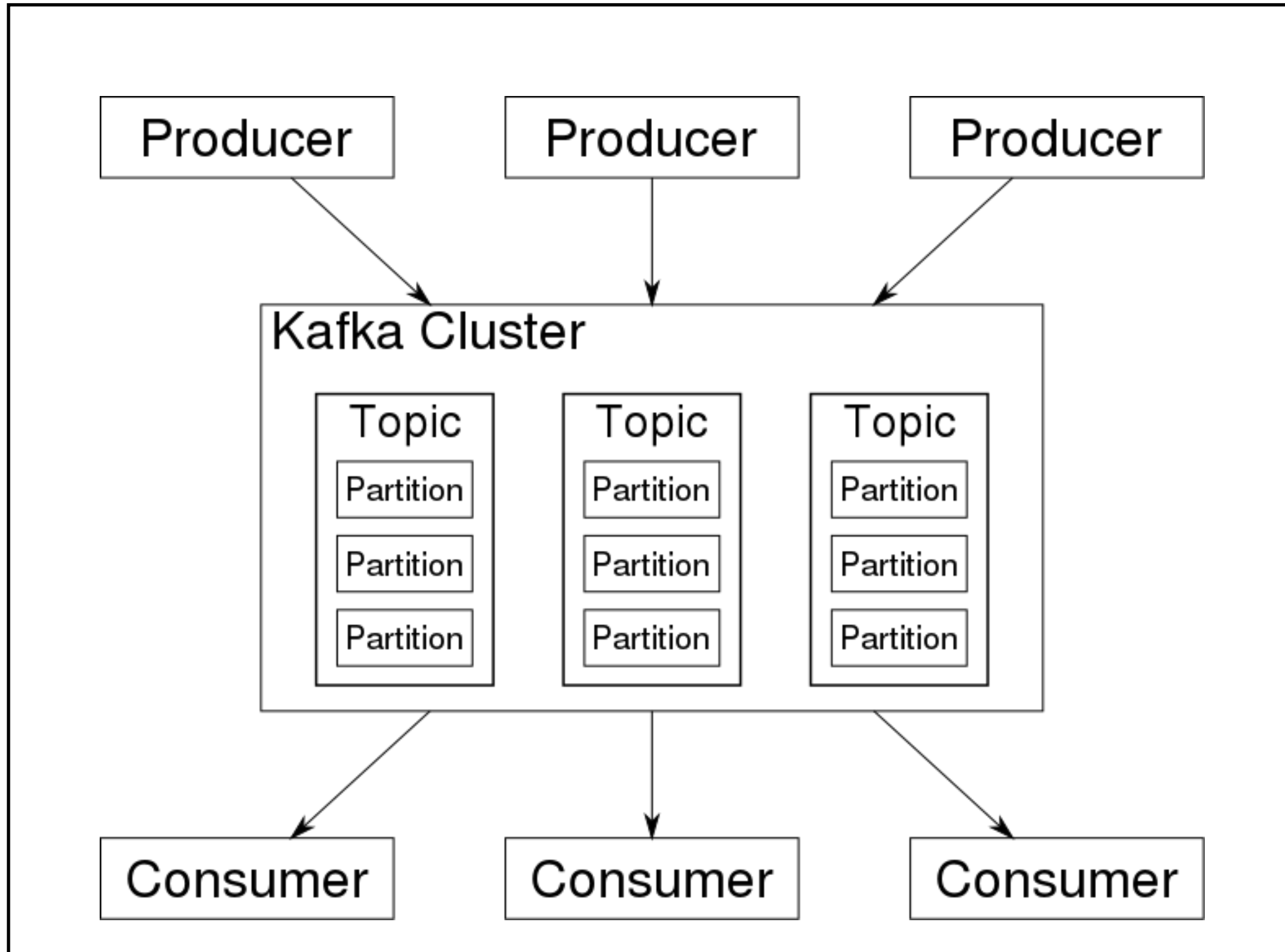
## Connector

Producer or consumer connect Kafka topic to existing data systems

Connect to database could capture every change to a table



# Kafka - Topic, Partitions, Logs



# Topics, Partitions, Logs

## Topic

A category for a stream of records

Can have 0, 1, or many consumers subscribe to it

Can have multiple partitions

## Partition

Log of records for a topic

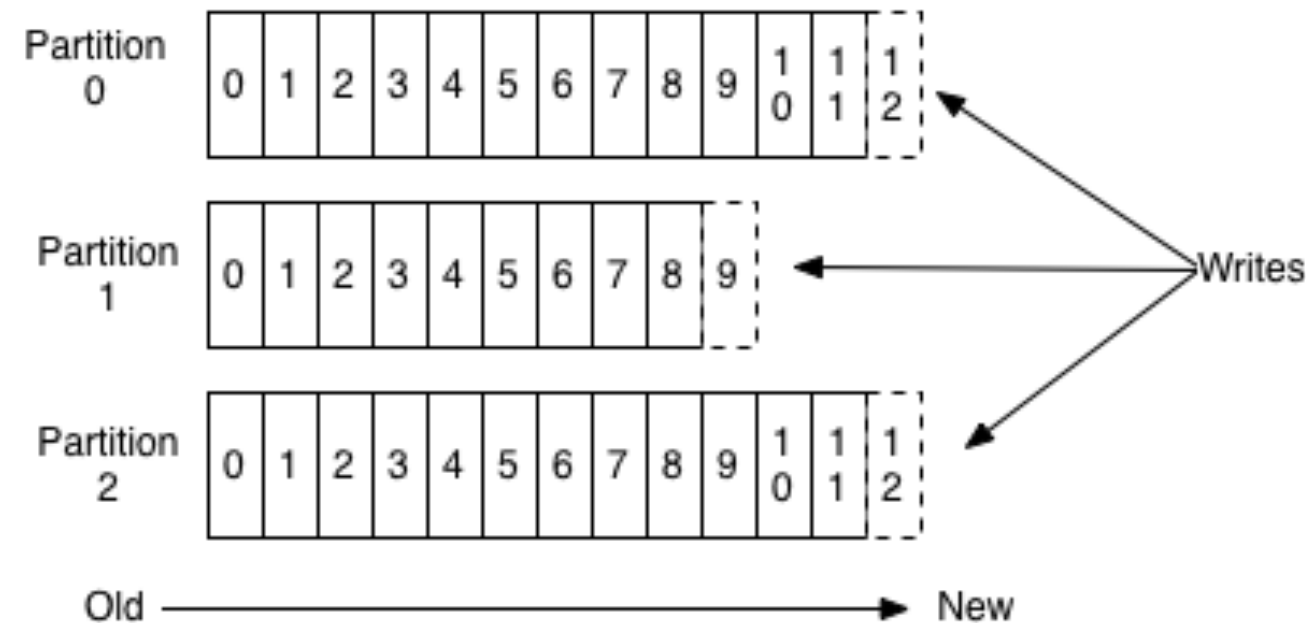
Stored on disk

Records are given sequential id

Each partition on different machine

Each partition replicated

## Anatomy of a Topic



# Retaining records

Records are retained for a fixed time

Configurable

Records are not deleted after a read

Records are deleted at end of retention period even if not read

Number of records in single partition

Does not affect performance

Single partition must fit on one machine

# Consumers & Partitions

Kafka server maintains offset for each consumer

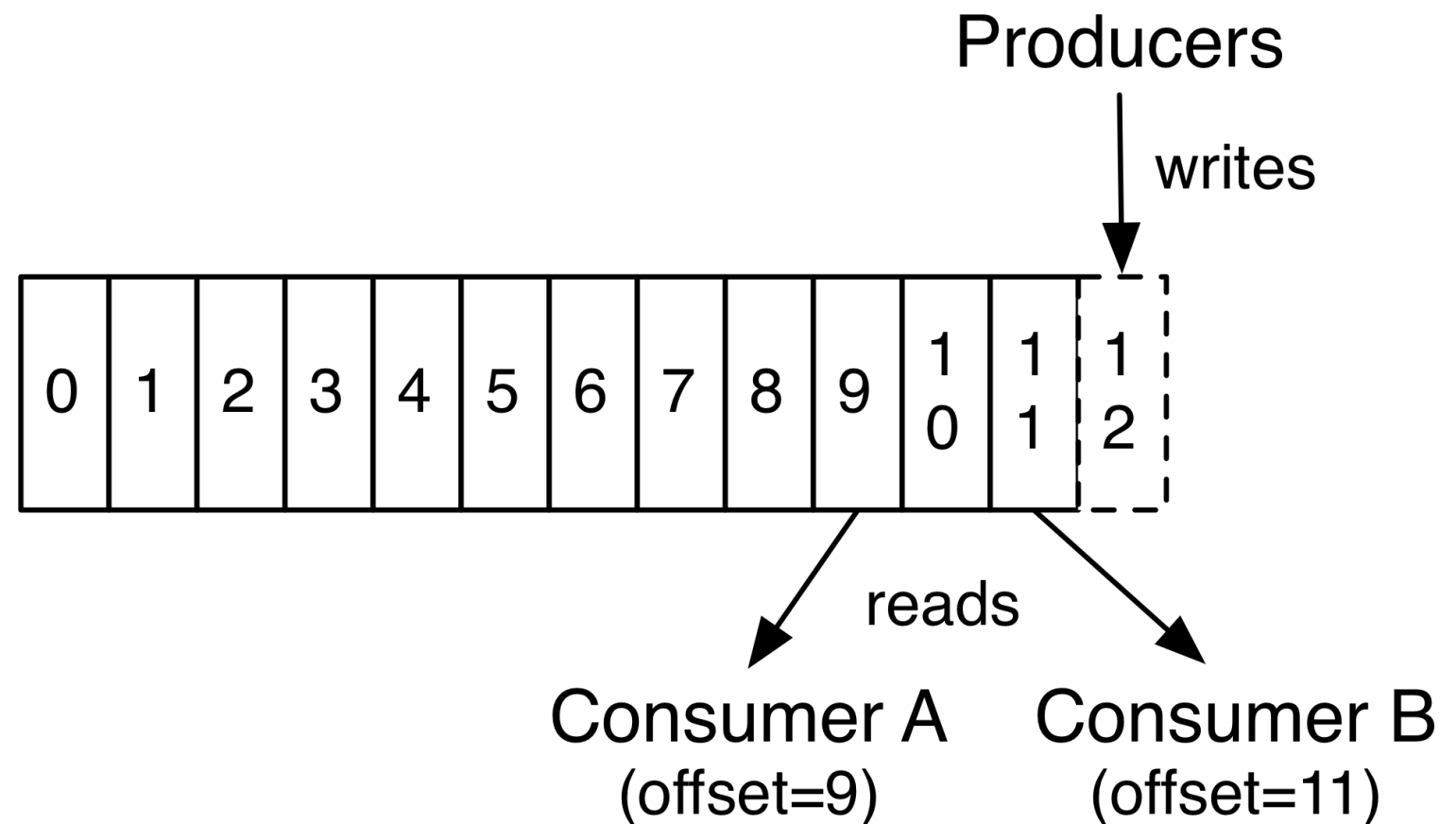
- Last read record

- Makes consumers cheap to support

Consumer controls offset

- Can go back

- Skip ahead



# Distribution of Partitions

Each Kafka server handles a share of partitions

Typically there are many topics

Each partition

Replicated on multiple machines

One machine is leader of partition

Others are followers

Each machine is leader for some partitions

When leader fails follower becomes leader

# Producers

Publish data to topics of their choice

Producer chooses with partition to write to

- Round robin

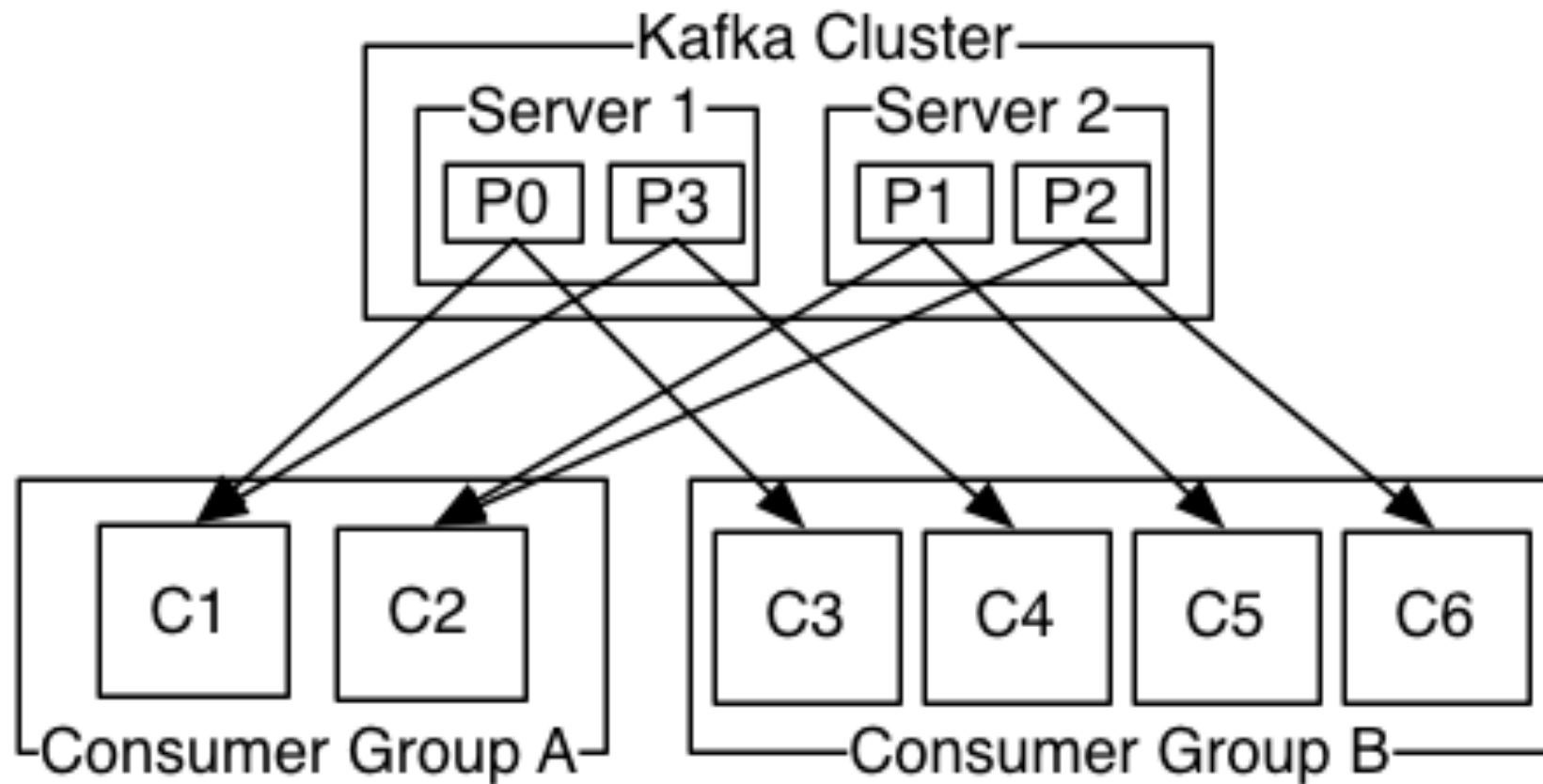
- Select by key



# Consumers

Consumers have a consumer group name

Each record in a topic is delivered to one consumer in each subscribing consumer group



# Guarantees

Topic partition appends messages from same producer in the order they are sent

A consumer instance sees records in the order they are stored in partition log

For a topic with replication factor  $N$ ,

we will tolerate up to  $N-1$  server failures without losing any records committed to the log

# What does not happen

Individual partitions are ordered by when messages arrive

There is no order kept between partitions in the same topic

Producer

Sends M1 to partition A

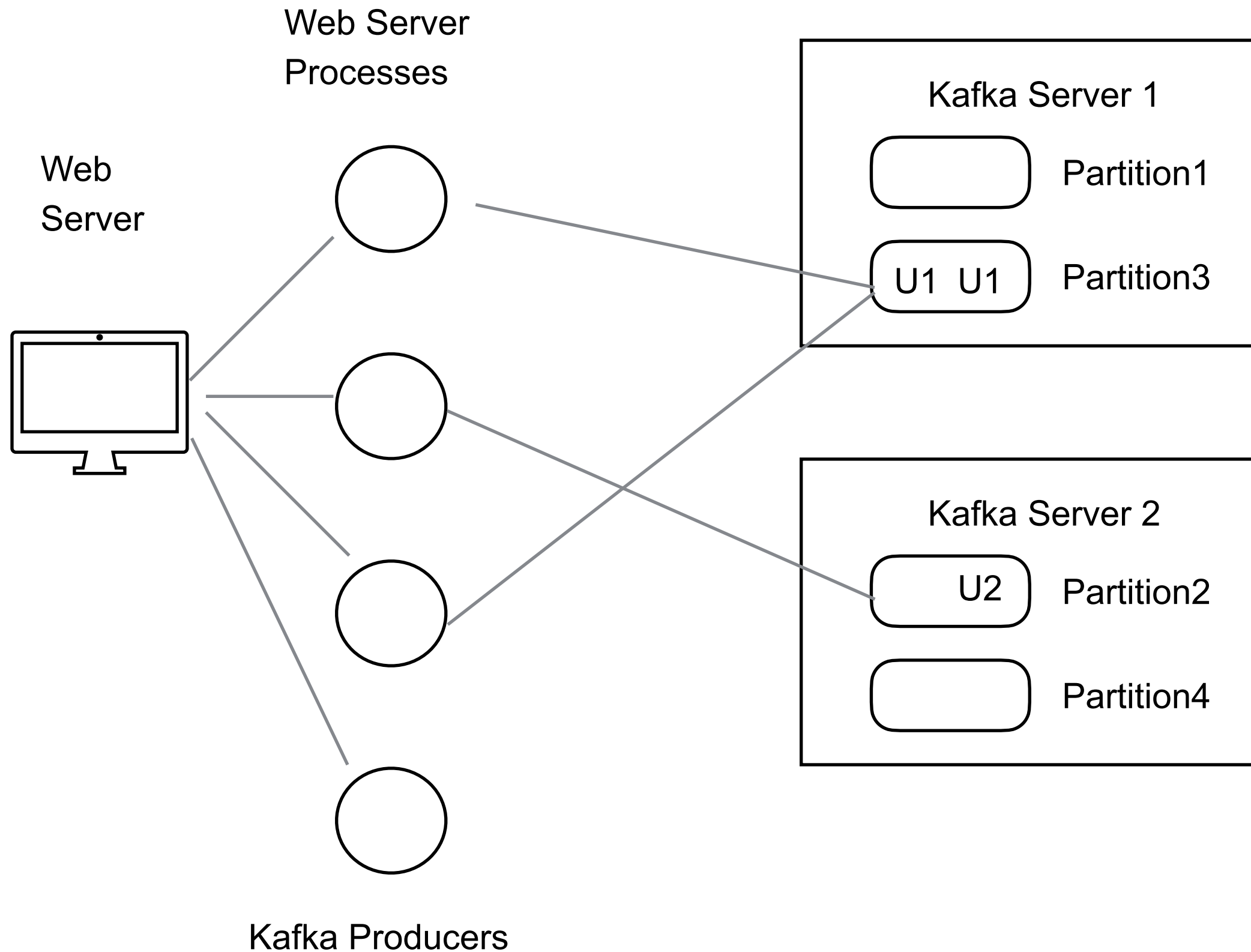
Then send M2 to partition B in same topic

Kafka does keep track that M1 arrived first

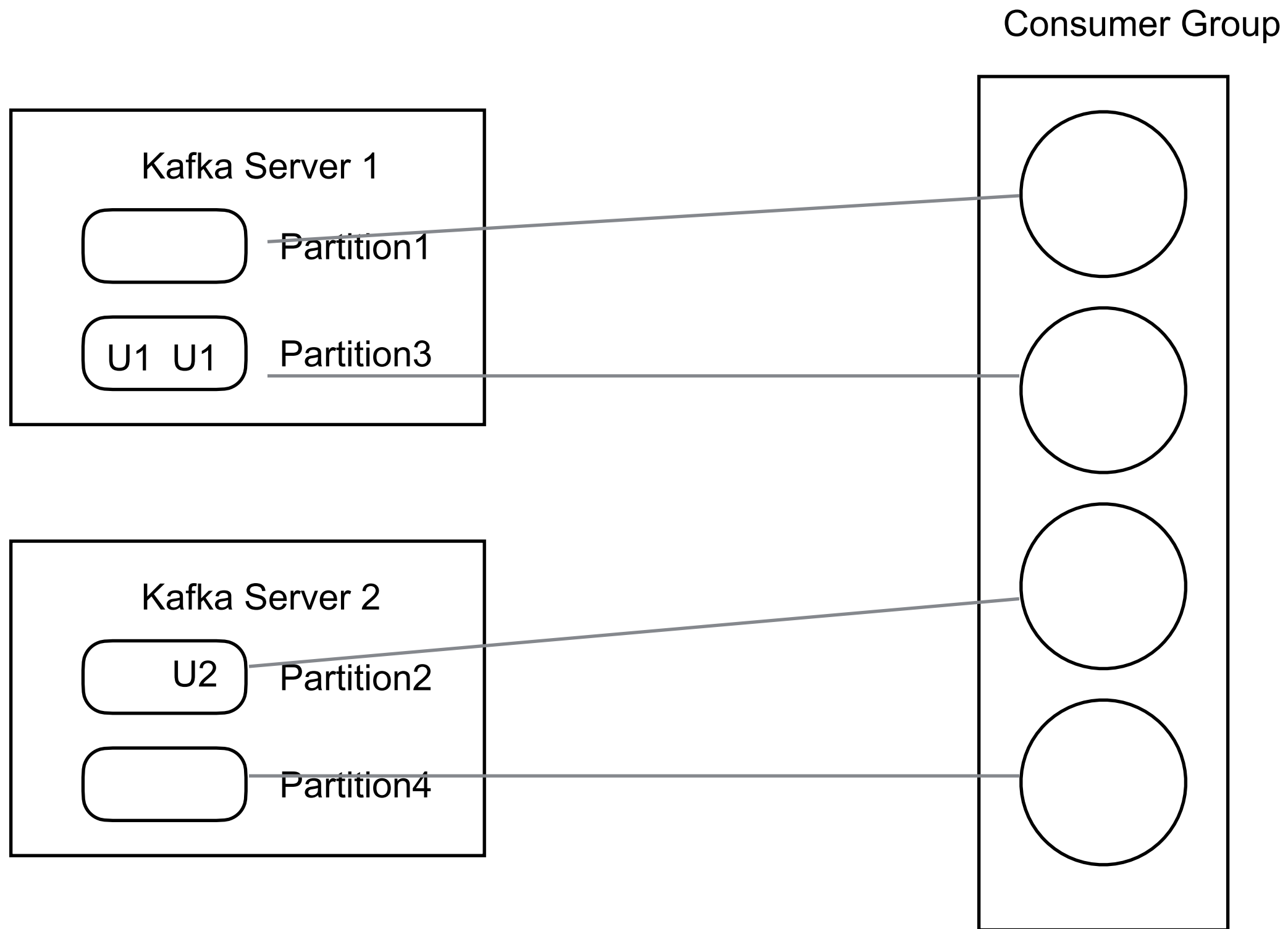
Partition A and B could be on different machines

Per-partition ordering + the ability to partition data by key is sufficient for most applications

# Example - Logging User Web Activity



# Example - Logging User Web Activity



# If Need Strict Ordering of Messages

Use one partition for the topic

Means only one client per consumer group on that topic

# Kafka Performance

On three cheap machines

<http://tinyurl.com/ng2h9uh>

Setup

6 machines

Intel Xeon 2.5 GHz processor with six cores

Six 7200 RPM SATA drives

32GB of RAM

1Gb Ethernet

Kafka cluster - 3 machines

Zookeeper - 1 machine

Generating load - 3 machines

Test uses 100 byte messages

6 partitions

# Producer Throughput

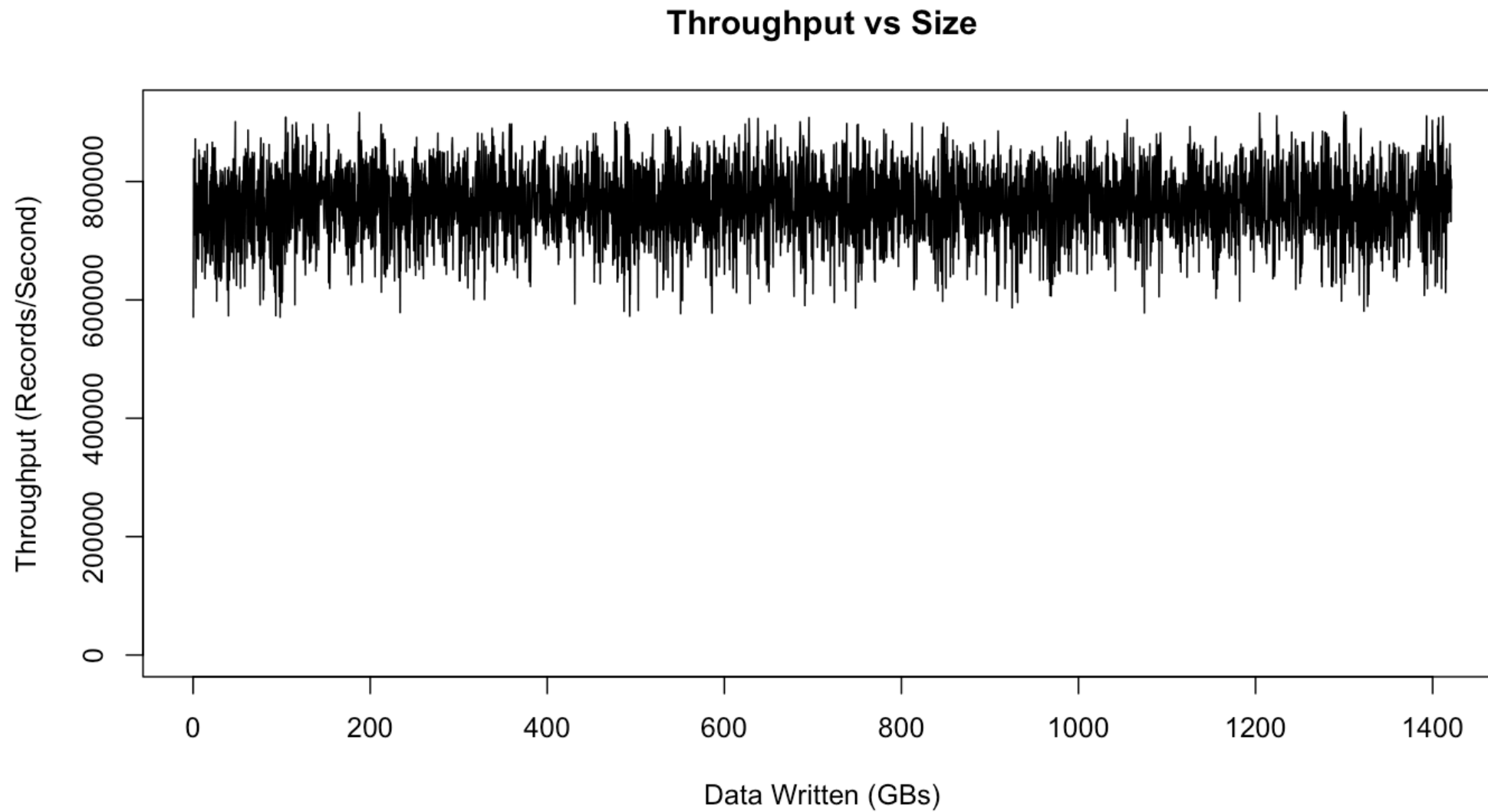
	Records/Second	MB/sec
1 producer, no replication of partition	821,557	78.3
1 producer, 3 async replication	786,980	75.1
1 producer, 3 sync replication	421,823	40.2
3 producers, 3 async replication	2,025,032	193.0

7,290,115,200 Records/Hour



# Producer Throughput Versus Stored Data

Does the amount of stored data affect performance?



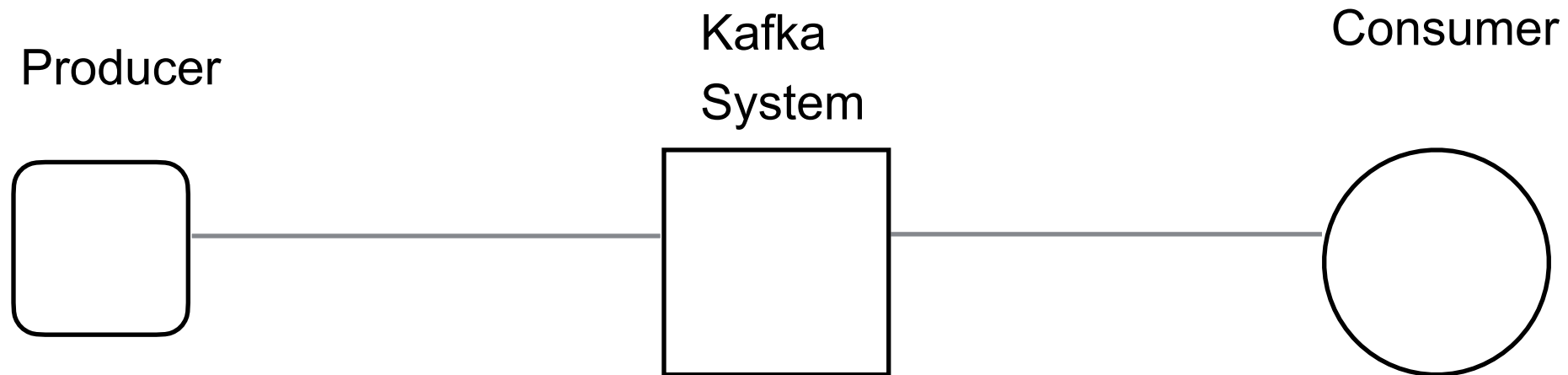
Persisting messages is  $O(1)$

# Consumer Throughput

	Records/Second	MB/sec
1 consumer	940,521	89.7
3 consumers - same topic	2,615,968	249.5

6 partitions, 3x async replicated

# End-to-end Latency



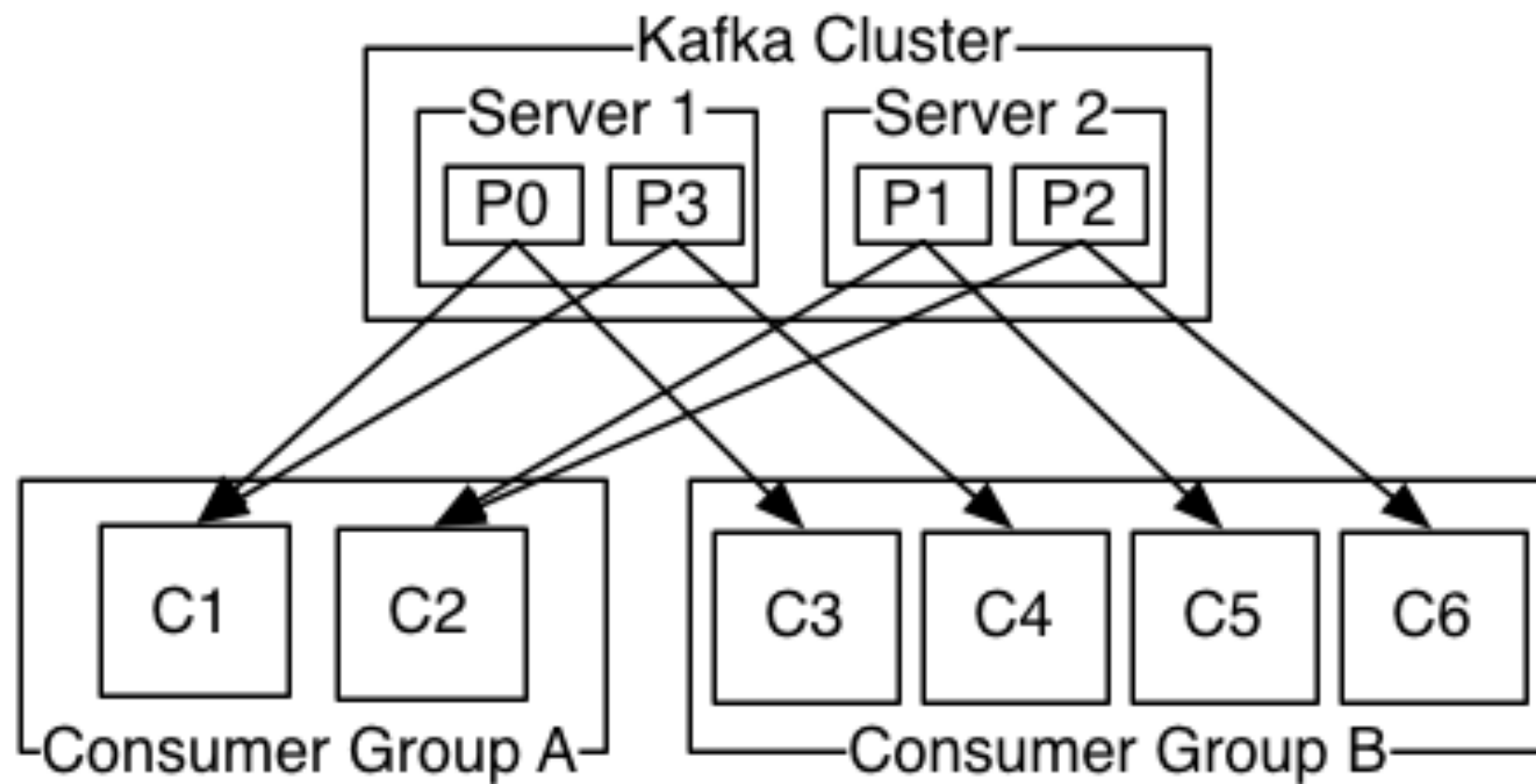
2ms - median

3ms - 99th percentile

14ms - 99.9 percentile

# What Makes Kafka Fast - Partitions

Allows concurrent writes & reads on same topic



# What Makes Kafka Fast - Messages

Messages

Binary format

Batched

Compressed

Producer convert message into binary

Kafka treats message as bits

Consumer needs to know how to unpack message

Producer supplies schema

Adds schema to ZooKeeper

bit 0~2:

0: no compression

1: gzip

2: snappy

3: lz4

# What Makes Kafka Fast - Use the Filesystem

Linear read/writes are very fast on hard drives

JBOD configuration six 7200rpm SATA drives 600MB/sec

Modern OS (Linux)

Heavily optimized

read-ahead, write-behind

Will use all free memory for disk cache

JVM

Memory overhead of objects is high

Java garbage collection becomes slow as heap data increases

So

Write data to disk

Use as little memory as possible

Let OS use nearly all memory for disk cache

28-30GB cache on 32GB machine

# What Makes Kafka Fast - sendfile

Normal path to send a file on network

- 1 OS copies file Disk -> page cache in kernel space
- 2 Application copies data: page cache -> user-space
- 3 Application copies data: user-space -> socket buffer in kernel space
- 4 OS copies data: socket buffer -> NIC buffer

Using sendfile to send a file on network

1. OS copies file: Disk -> NIC buffer

Using OS pagecache + sendfile means when consumers are mostly caught up

Files served from cache