Log vs ETL

Data-centric, event-driven approach
for building distributed high-throughput applications with Apache Kafka

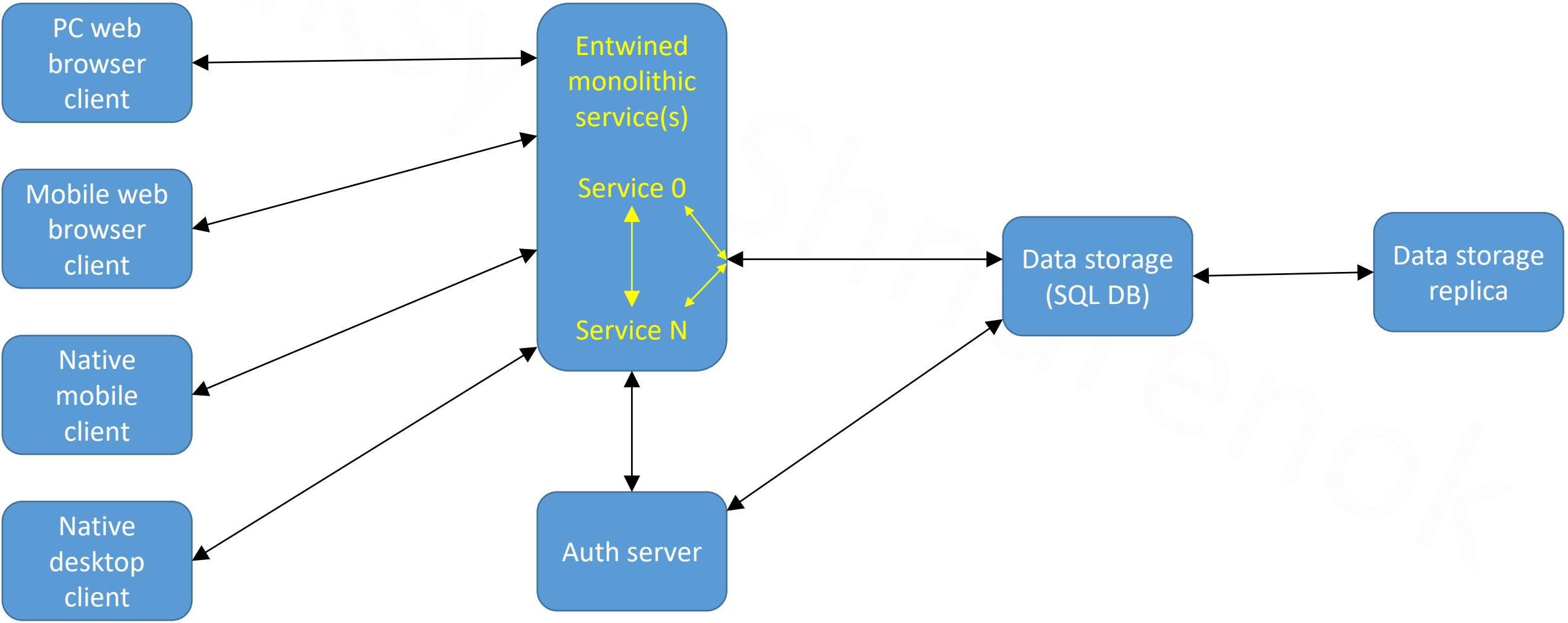
Slides online

https://github.com/MaxCrank/LogVsEtlKafka

Distributed applications

- More than 1 device/machine to perform tasks
- Access over network
- Therefore, it’s a vast majority of modern and legacy business-oriented applications

Traditional simple system design



Traditional simple system features

- Business logic is the core, making services a heart of the application
- High throughput is probably not required
- Data flow is secondary
- Data is batched for periodic replication

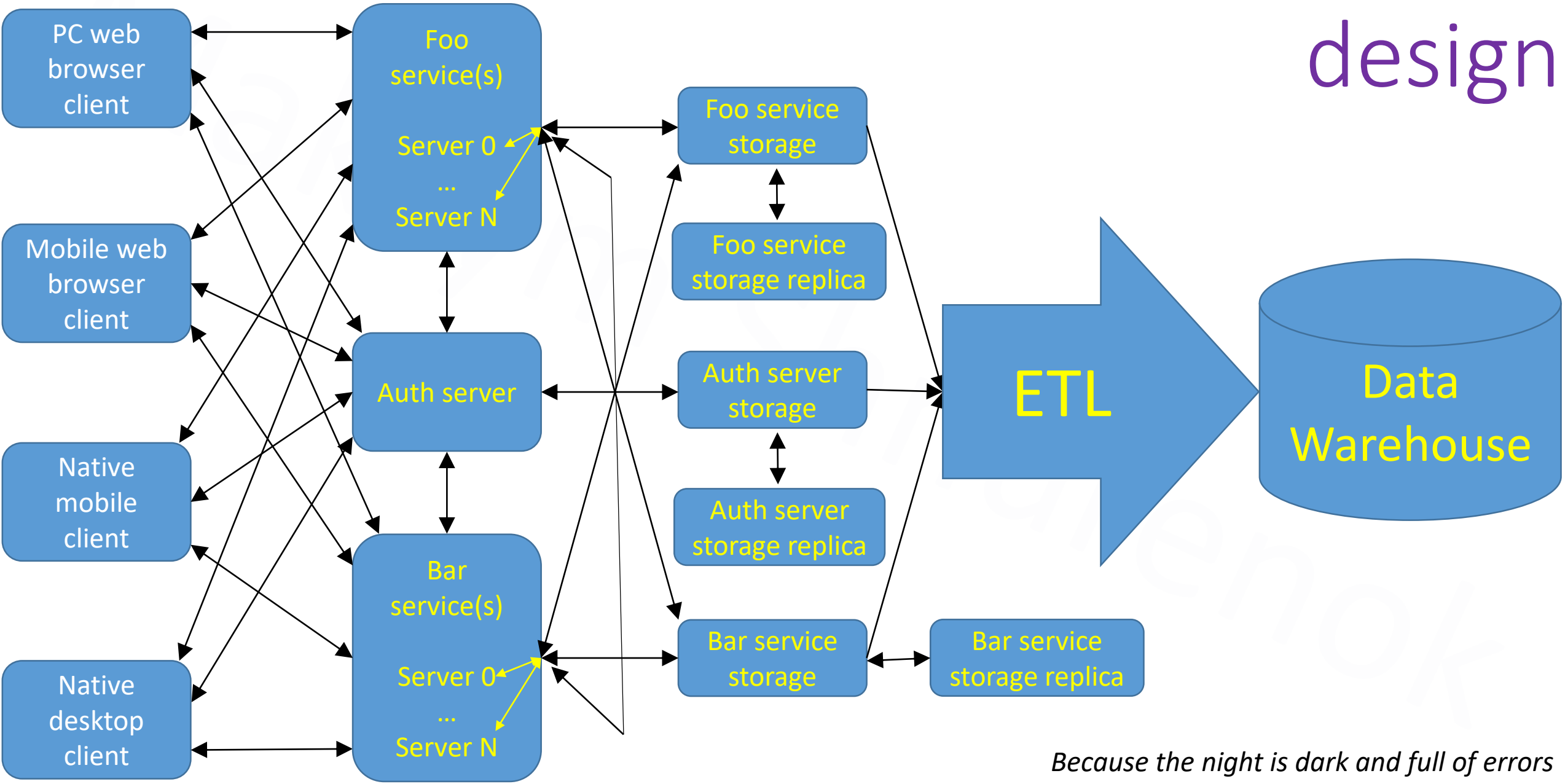
Traditional simple system problems

- Requires intense manual interference in case of data loss to restore the state (either if services or DB are down)
- Complete data recovery is impossible in typical case that depends on the replication period

Traditional simple system. So what can we do about it?

- Setup the safest type of replication possible depending on your system's throughput requirements to minimize replicated data loss
- If you haven't done it yet, handle "server is down" and scaling scenarios to prevent input data loss
- If you need both high throughput and resilience, consider system's design improvement

Complex service-oriented system design



Because the night is dark and full of errors

What is ETL?

1. Extract: get data from all separate sources and validate it
2. Transform (unify): clean storage-specific data, apply business rules, aggregate (or disaggregate) data
3. Load data into warehouse for further processing by BI, reporting, diagnostic services etc.

Complex service-oriented system features compared to a simple one

- High throughput is achieved with scaling
- Resilience is also achieved with scaling (using costly ETL tools)
- The core, service-centric concept remains the same
- Data becomes more important

Complex service-oriented system problems compared to a simple one

- Increased complexity and high coupling
- Infrastructure maintenance costs
- High latency for end-users

Complex service-oriented system. So what can we do about it?

- Evolve to Log-centric, event-driven system – make it do “ETL in reverse”!

Data is the core

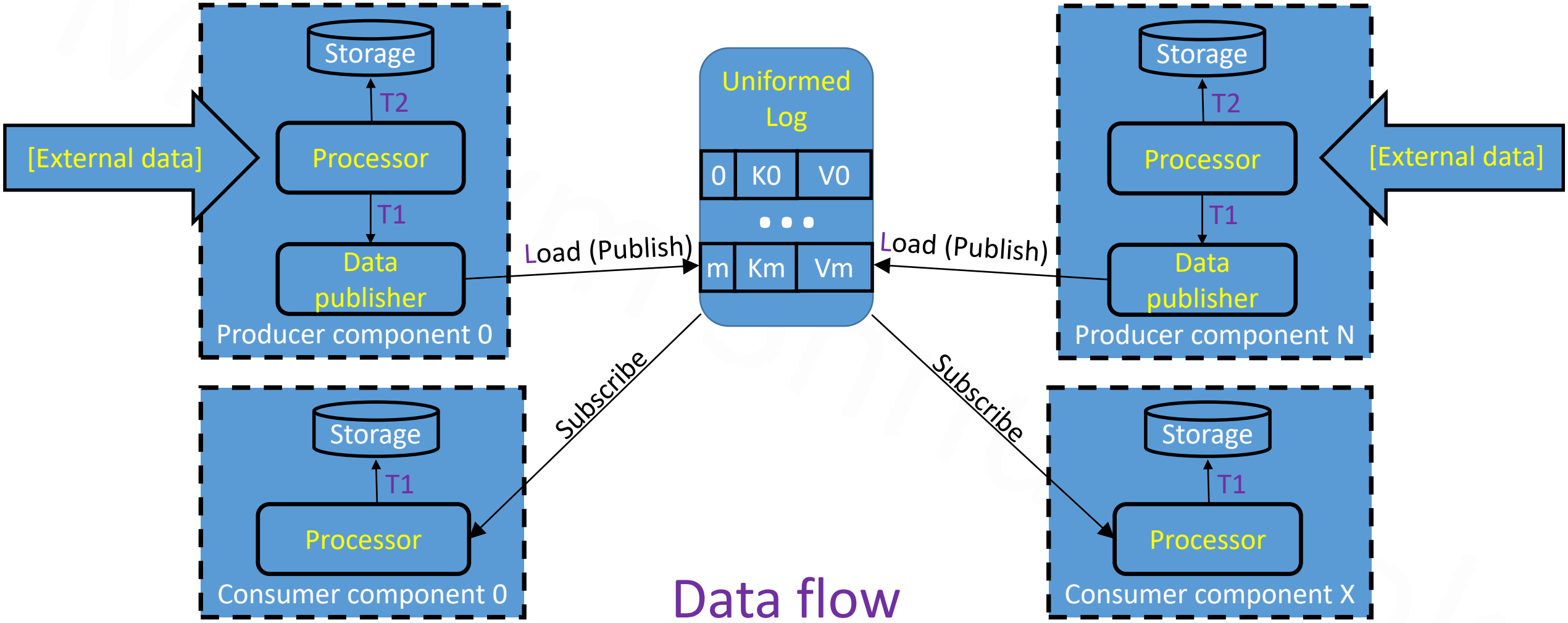
- Data represents the state of the system
- Business relies on data (data loss equals to money loss)
- Taking care of data is the key to resilience and fault-tolerance

What is Log?

baby don't hurt me, don't hurt me no more

- Log is represented as ordered key-value storage, where first record means the very start of the system – opposite to time-based concepts, only the order matters
- Log holds the state and is a *single source of truth* for the system
- All incoming data is prepared and moved to the Log right after receiving (*before* processing)

The Log concept



Data flow

1. Producer system component can either consume external data or produce it independently.
2. Producer system component cleans and structures data for uniformity, then puts it to the Log.
3. Then, Producer system component can process the data and use its' own storage.

How does the Log help with resilience and fault-tolerance?

- Each system component holds the last processed Log index
- In case of failure and restart, the system can restore data consistency (and therefore, state) by consuming latter Log entries that have appeared after failure
- It’s even theoretically possible to restore the state of the whole application from the very start only from persisted Log

How does the Log help with throughput?

- Log is filled with new records in real-time manner by flushing the data *before* processing
- Publish/Subscribe relations between system components and *the single source of truth* decouple them and eliminate the overhead of many-to-many relations
- Adding new Log entries cause *events* for subscribers, so such application is event-driven

Positive side-effects of Log-centric design

1. Another example of recently hyped *Lambda Architecture* with separate batch and speed layers (though, the Log concept established long before).
2. Extensibility – easy to add new decoupled, storage-agnostic components.
3. Readiness for business logic/context changes. *The single source of truth* provides ability for components to quit or add processing of any data by constant access to literally all the data available in the Log.
4. This makes it ideal for service-oriented/microservices architecture.

Looks good, but how to use the Log in practice?

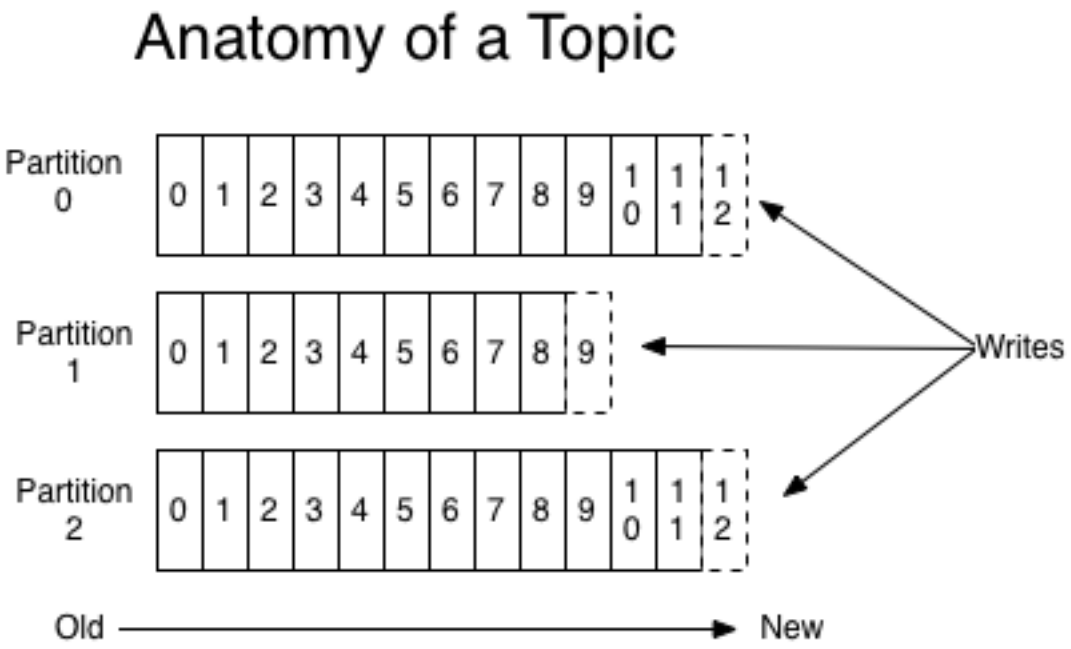
- What are the real-life instruments?
- How is it persisted so performance and availability remain high?
- Can it really handle high throughput, or be scaled (and how)?

Pic is from https://kafka.apache.org/

Apache Kafka – the real-life instrument for Log

- Self-described as *"a distributed streaming platform"*, it treats data in a stream manner
- High-performance TCP protocol
- Replicated, fault-tolerant data storage system
- Publish/Subscribe features to use as a regular message broker

The basics of Log in Kafka

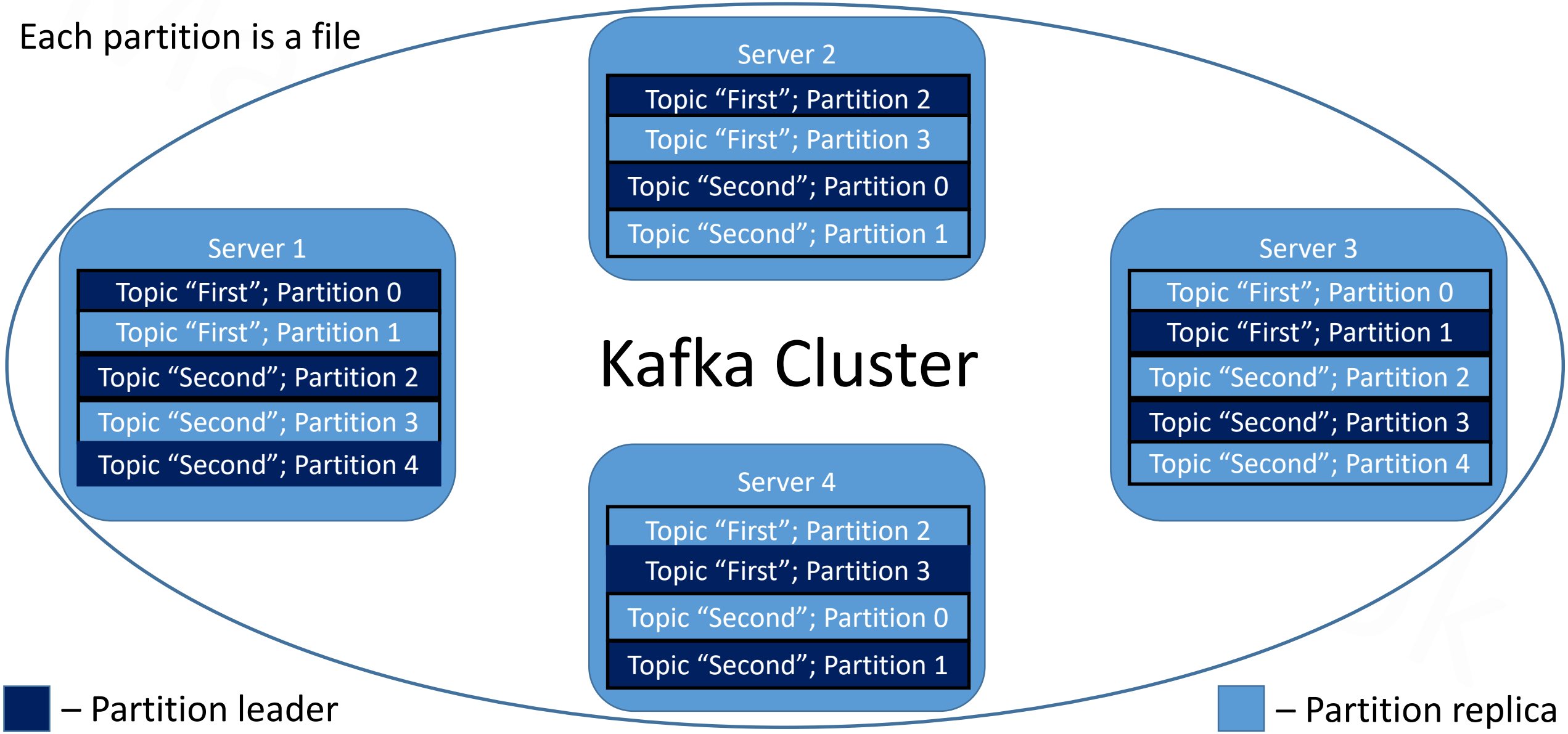


Pic is from https://kafka.apache.org/intro

- Data is saved to Topics
- Topics are divided to Partitions
- Partition store order-based records (this way, order is guaranteed only *within partitions*)

Log distribution in Kafka

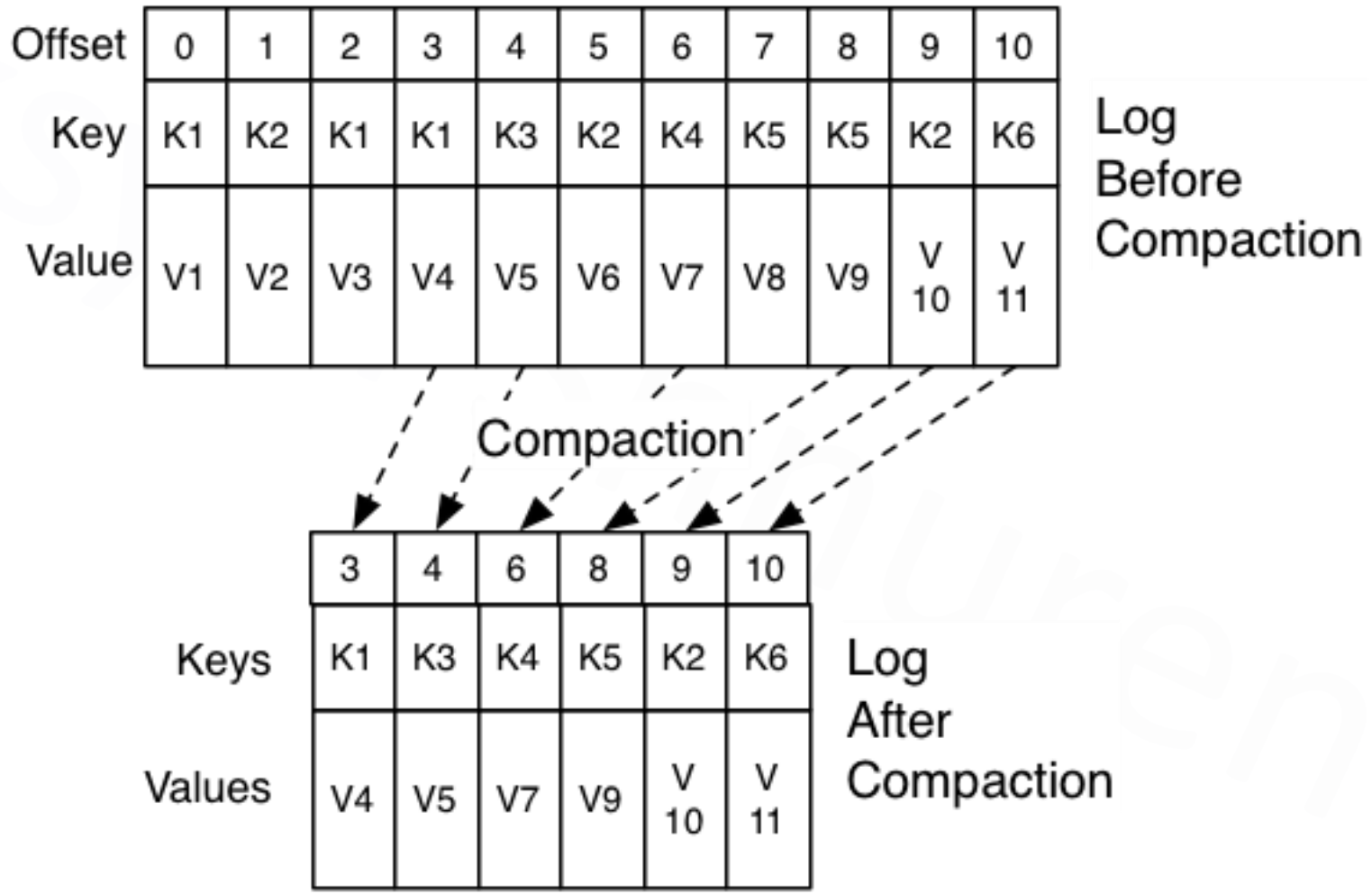
Each partition is a file



– Partition leader

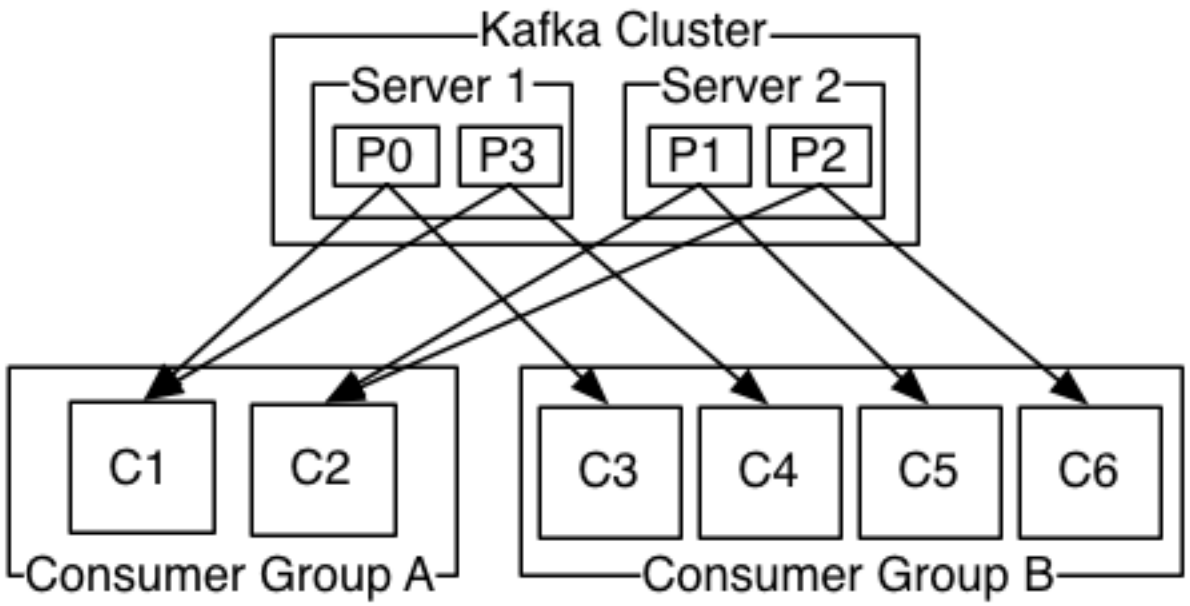
– Partition replica

Log compaction in Kafka



Pic is from https://kafka.apache.org/intro

Log subscribers in Kafka



Pic is from https://kafka.apache.org/intro

- Subscribers are divided to Consumer Groups with many consumer instances, respectively
- Each Partition is intended for a single consumer instance
- If a Topic is shared between more than one Consumer Group, load balancing is applied so that each consumer instance can perform

Fundamental Kafka features and decisions

- Data is transferred using low-level APIs for Virtual Memory page cache to avoid data copying duplication between disk and RAM, kernel and user space (i.e. *zero-copy*)
- Efficient message grouping and compression for transfer and persistence
- Data is considered as published only after replication is finished (i.e. after all partition replicas pull data from partition leader just as a regular Consumer/Subscriber)
- After a fail, restarted Log server must fully re-sync again to avoid possible data corruption/loss
- By design, consumers store partition offsets to the same place as the output data

Fundamental Kafka features and decisions

- Standardized binary messages shared between the producer, the broker and the consumer to avoid intermediate data transformation
- Linear writes with appending bytes to a single partition file instead of making separate files for each record to minimize disk seek operations
- Push is chosen for Producers and long-poll for Consumers in message broker
- CPU- or bandwidth-based quotas of broker resources usage for producers and consumers
- Flexible configuration for durability, synchronization, replication, delivery guarantees, timeouts, quotas, log compaction etc.

More on data publishing in Kafka

- Producer controls the choice of partition for data transfer
- By design, Producer transfers data to partition leader directly, sending the request to get its' location first, but may just send data to any partition if needed
- Producer can use semantic partitioning (i.e. by key) for data locality or just send data to random partitions

More on data consumption in Kafka

- Reminder: each partition has a single Consumer instance in a group at a time
- Reminder: consumer stores a current partition position/offset *with* output data (vs separate saving the position either before or after processing the input data)
- Quote: “This makes the equivalent of message acknowledgment very cheap” (vs getting the state from the Consumer and holding it on the Broker)
- It also makes possible to *rewind* consumption (vs standard queues with irreversible pop)
- “Exactly once” message delivery guarantees *for real* (i.e. fails are handled) along with less durable ones

References

- Log concept fundamentals: https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying
- First post of the series on efficient data processing: https://www.confluent.io/blog/data-dichotomy-rethinking-the-way-we-treat-data-and-services/
- Apache Kafka portal: https://kafka.apache.org/
- Apache Kafka performance: https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines