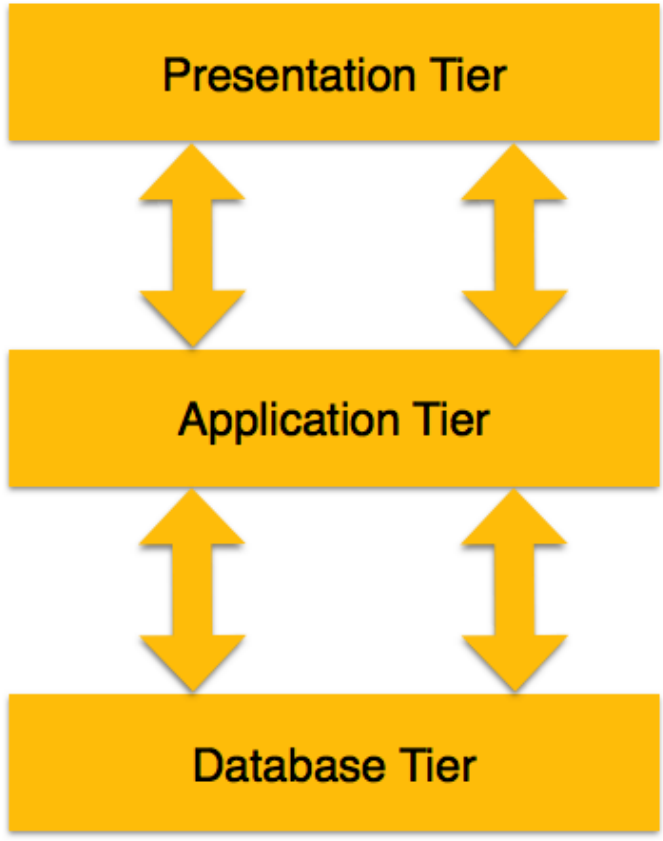
SPOUD

From Distributed Logs to Database Replication

Dr. Samuel Benz

How to achieve scalability, fault tolerance and consistency in distributed systems?

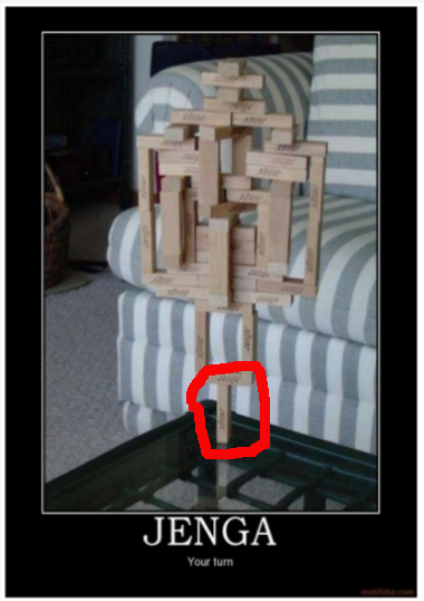
Distributed applications in theory...



. . . in practice

Why do we see such architectures?

Distributed Stateful Components



state (vs. stateless)

shared $>$ 1 client (isolation)

mutable $>$ 0 writer (concurrency)

distributed $>$ 1 DB (consistency)

geographically $> 50km$ (latency)

Reliable and Scalable Stateful Services

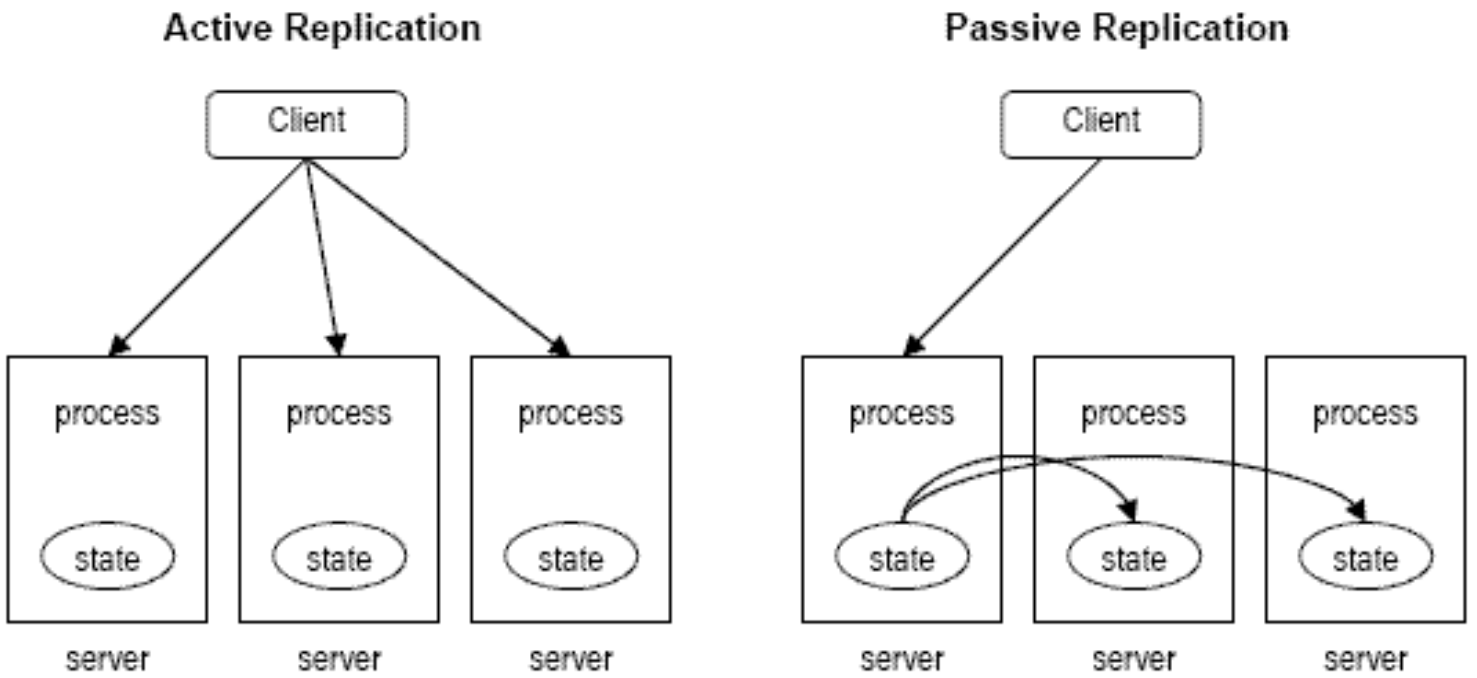
Problem

1. Scalability:
   - Size: Internet scale services
   - Location: Access latency
   - Administration: Multiple organizational units
2. Fault-Tolerance

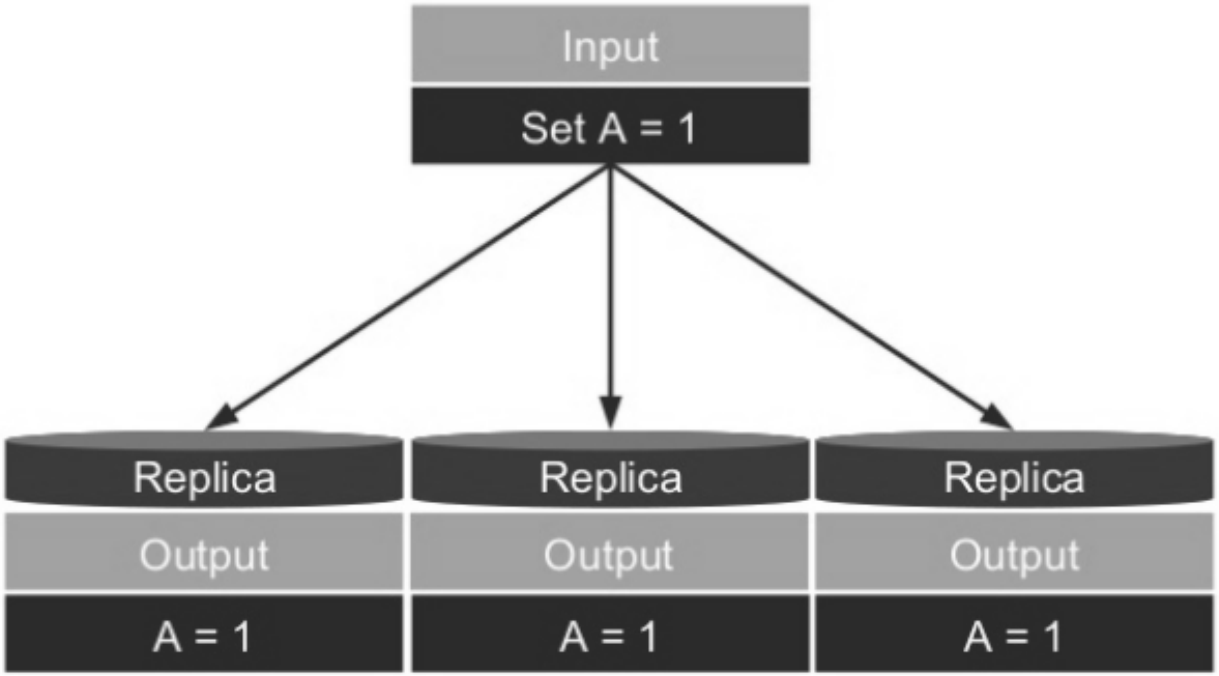
Solution

1. Distributed Data: **Replication**
2. Distributed Computing: Coordination

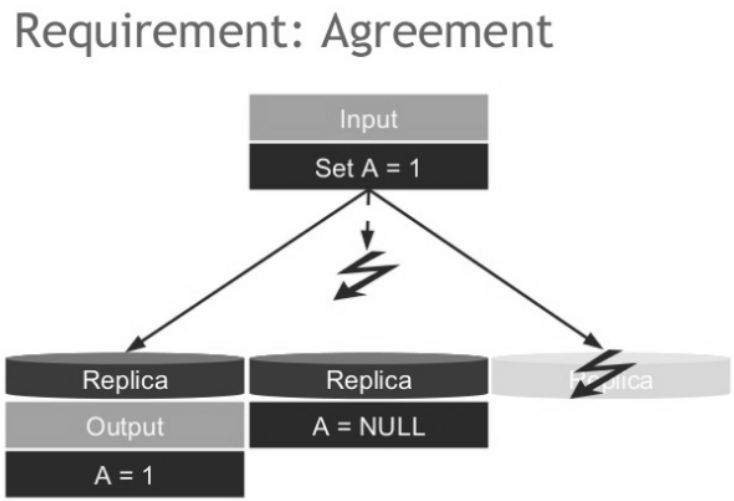
Different Types of Replication

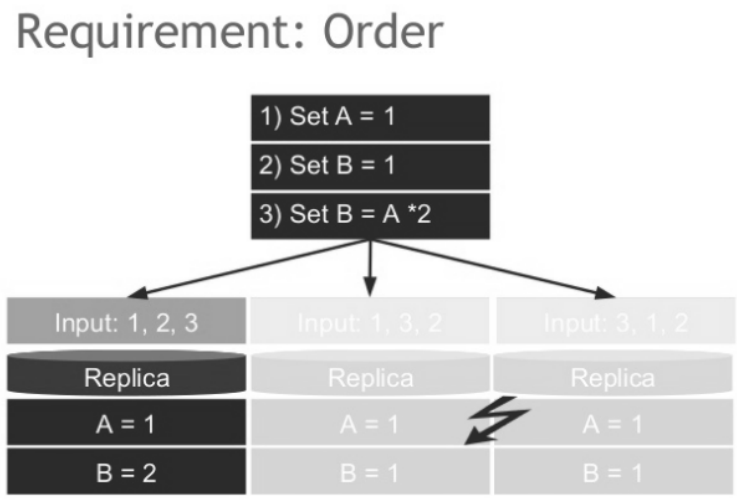


State Machine Replication → Fault-tolerance

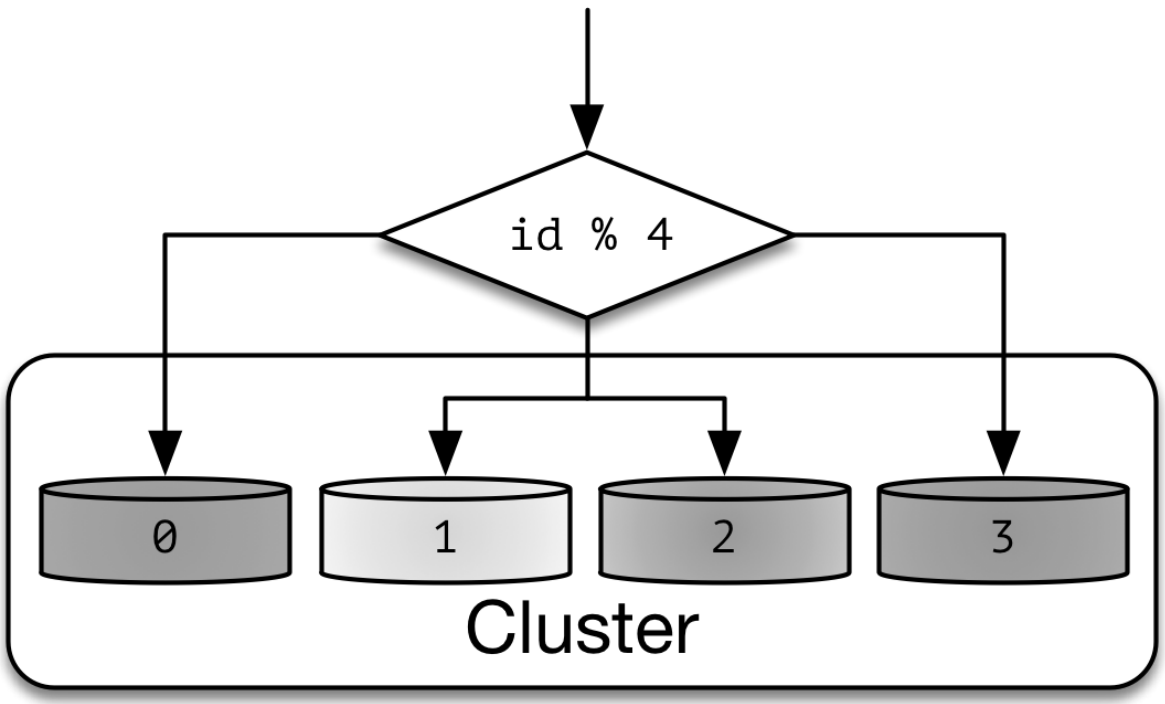


State Machine Replication → Consistency





Partitioning → Scalability



Consistent Partitioning

1. The system ensures strong consistency within partitions and **"best-effort"** across partitions.
2. The system ensures strong consistency using **2PC** across partitions.
3. The system orders commands before executing them or checks their order after executing the commands (**Atomic Multicast**).

Simple Coordination Problem

Consensus Problem

Fundamental Result

No algorithm can solve consensus in an asynchronous system despite a single crash.

FLP impossibility result (after Fischer, Lynch, and Paterson, 1985)

Consensus and Atomic Broadcast

In a crash-stop failure model **consensus** is defined as follows:

1. **Termination:** Every correct process eventually decides.
2. **Agreement:** No two correct processes decide differently.
3. **Uniform integrity:** Every process decides at most once.
4. **Uniform validity:** If a process decides v, then v was proposed by some process.

Additionally **Atomic Broadcast**:

5. **Total order:** If two correct processes p and q deliver two messages m and m', then p delivers m before m' if and only if q delivers m before m'.

[Chandra *et al*. Unreliable failure detectors for reliable distributed systems. 1996.]

Consensus and Atomic Broadcast

In a crash-stop failure model **consensus** is defined as follows:

1. **Termination:** Every correct process eventually decides.
2. **Agreement:** No two correct processes decide differently.
3. **Uniform integrity:** Every process decides at most once.
4. **Uniform validity:** If a process decides v, then v was proposed by some process.

Additionally **Atomic Broadcast**:

5. **Total order:** If two correct processes p and q deliver two messages m and m', then p delivers m before m' if and only if q delivers m before m'.

[Chandra *et al*. Unreliable failure detectors for reliable distributed systems. 1996.]

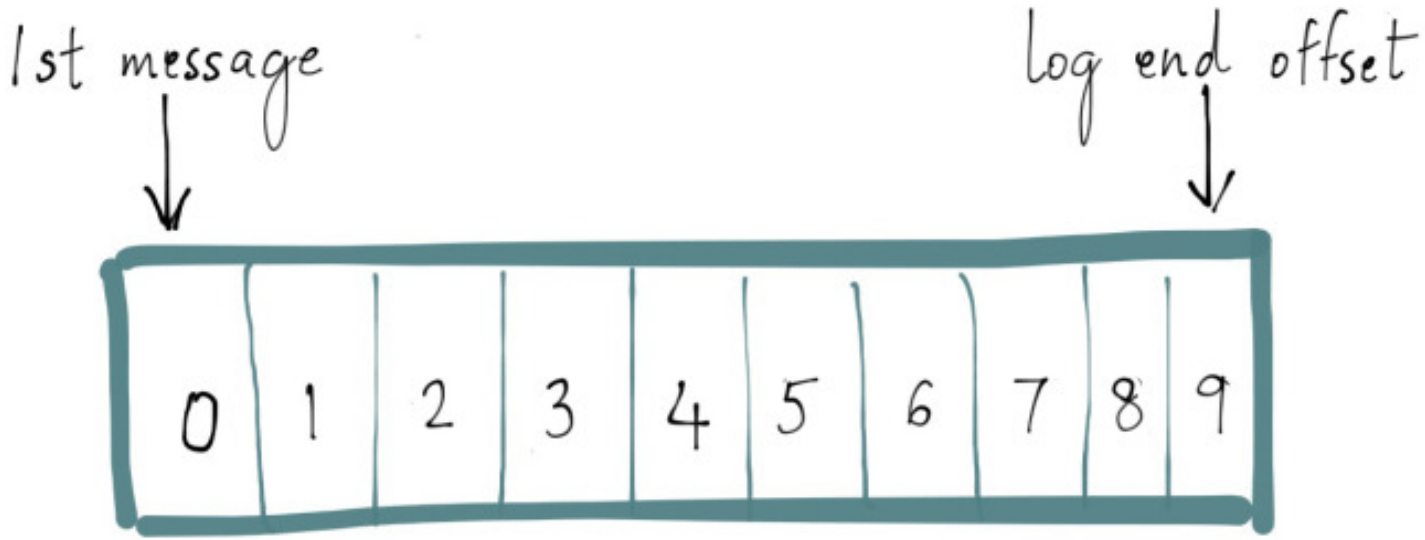
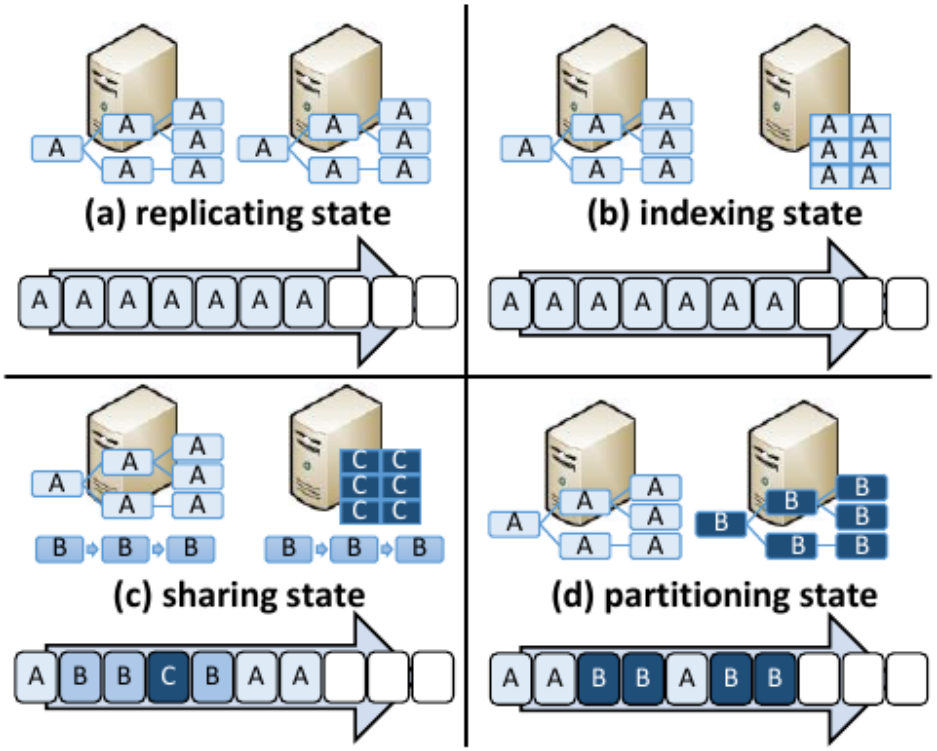
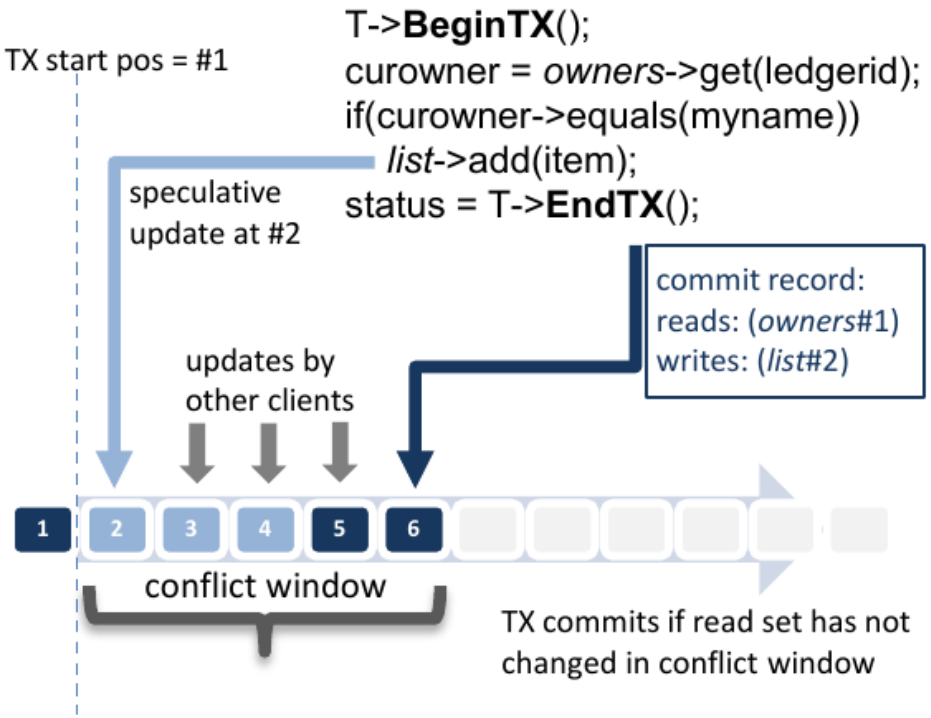


Fig 1: Partition's write-ahead log

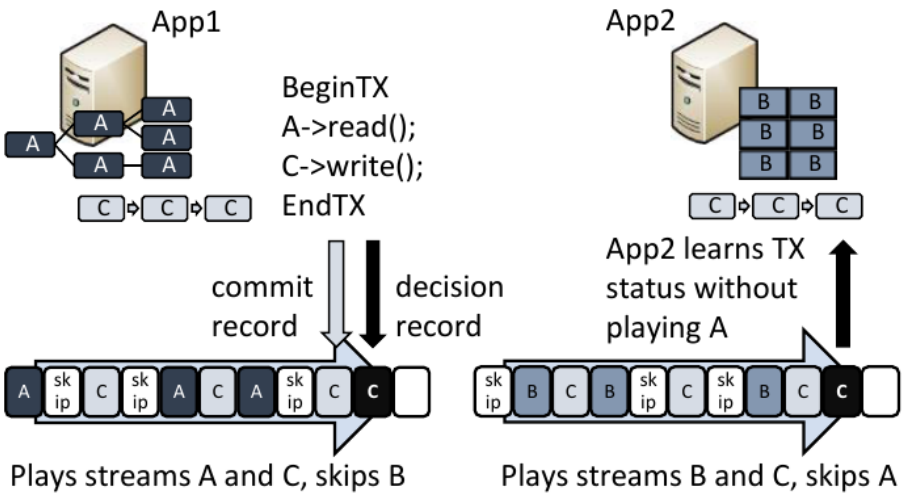
Distributed Log



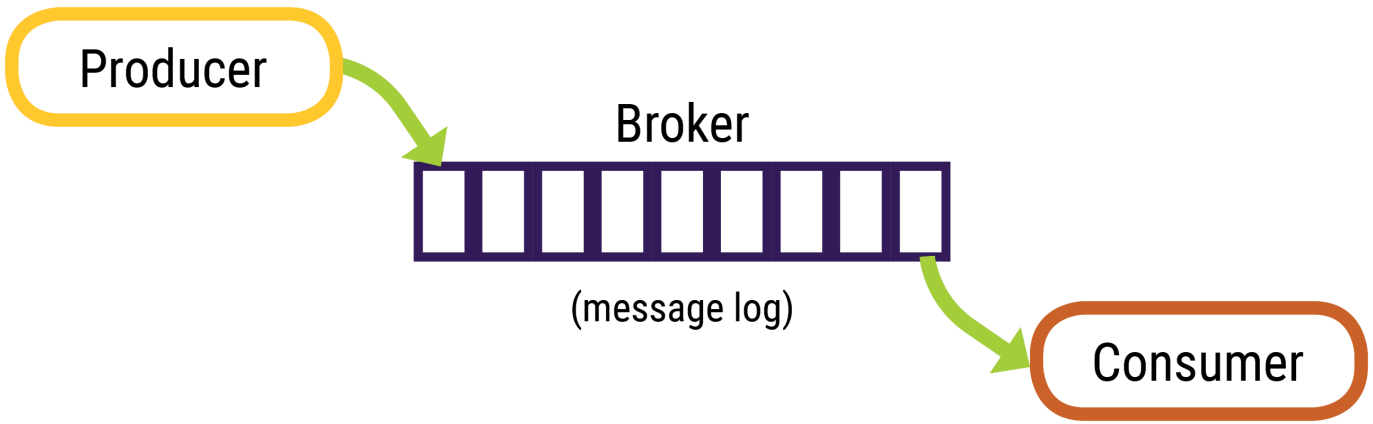
Distributed Transactions



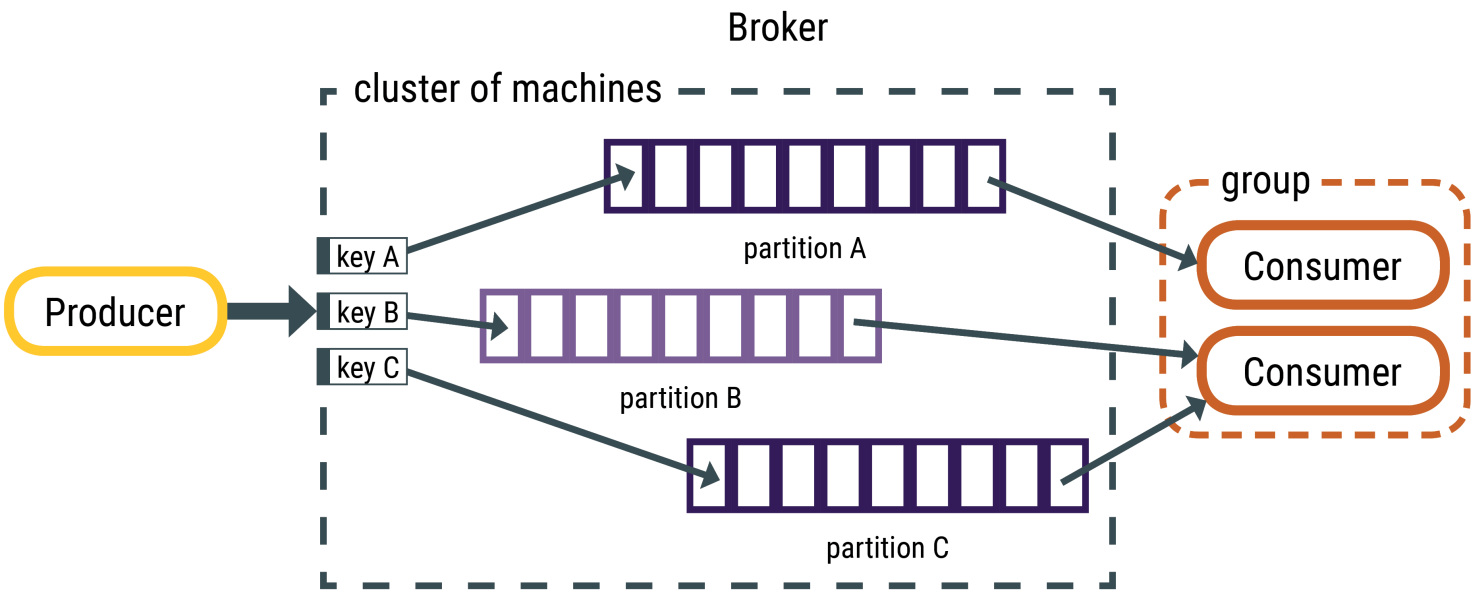
Distributed Data Structures



Kafka Consistency



Kafka Scalability



SPOUD