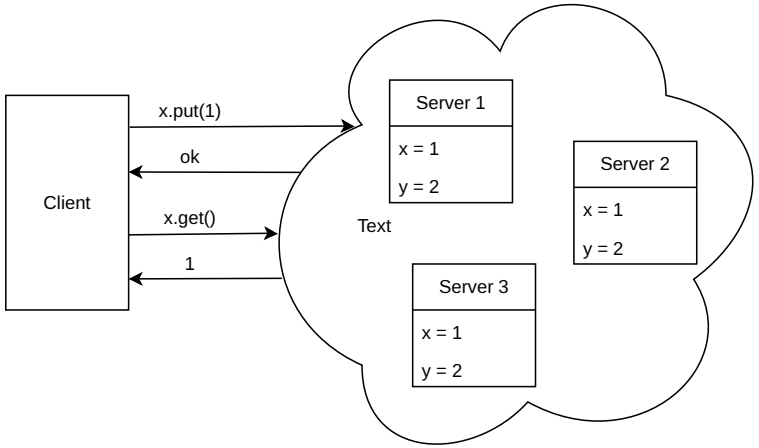
Replicated Data

Distributed Data

> A distributed system is a collection of independent computers that appears to its users as a single coherent system.

Andrew S. Tanenbaum and Maarten van Steen. 2006. Distributed Systems: Principles and Paradigms (2nd Edition).



When to use it

Reason 1: reliability, i.e., fault tolerance.

Reason 2: performance, i.e., very large data.

With retries we execute operations at most once.

Now we need to execute operations exactly once.

Later we will execute operations at least once.

Examples: distributed databases, key-value stores, collaborative editors, blockchains, ...

State Machine Replication

Fred B. Schneider. 1990. Implementing fault-tolerant services using the state machine approach: a tutorial.

Mealy machine: states, operations, outputs, deterministic transition.

Goal: all replicas execute the same operations in the same order.

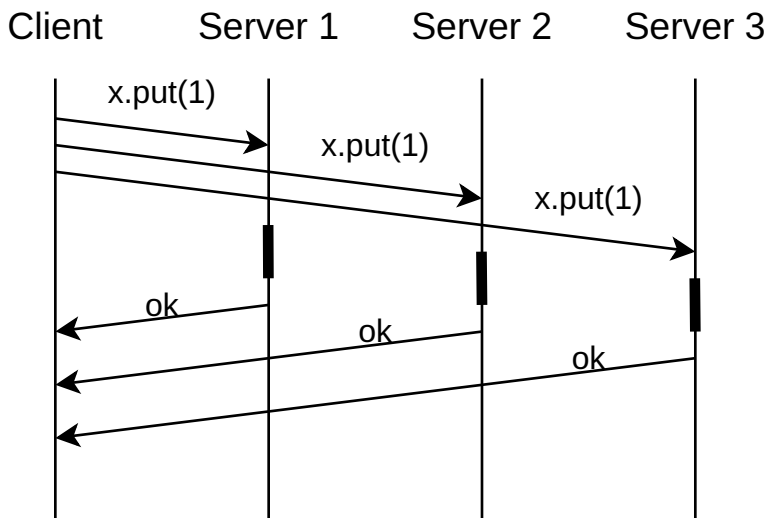
Reliability: every live process reliably receives every operation.

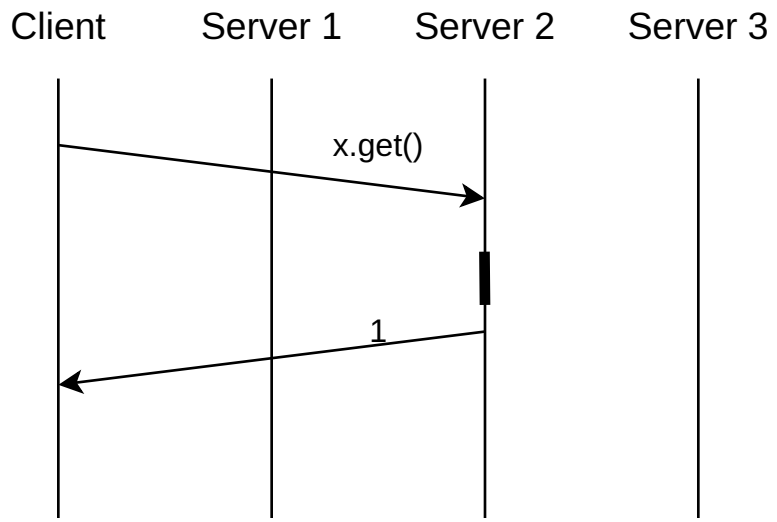
Ordering: every live process applies them in the same order.

Running example: a key-value store with operations `put` and `get`.

Active Replication

Client sends puts to all replicas and waits for acknowledgment and sends gets to any replica.



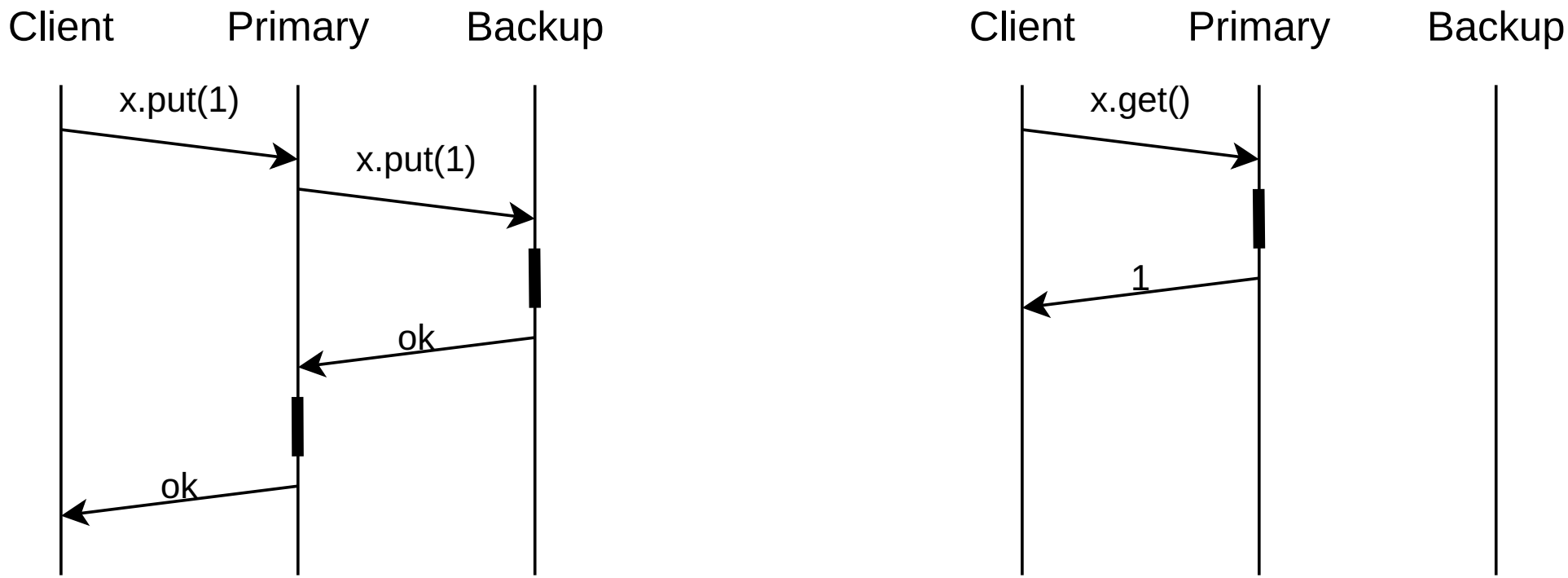


Put: write and acknowledge.

Get: read and respond.

Primary-Backup Replication

Client sends puts and gets to primary.

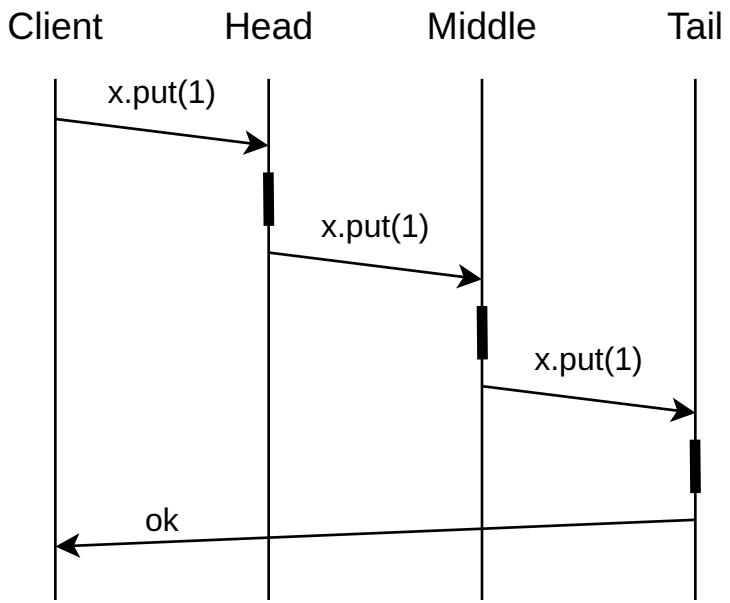


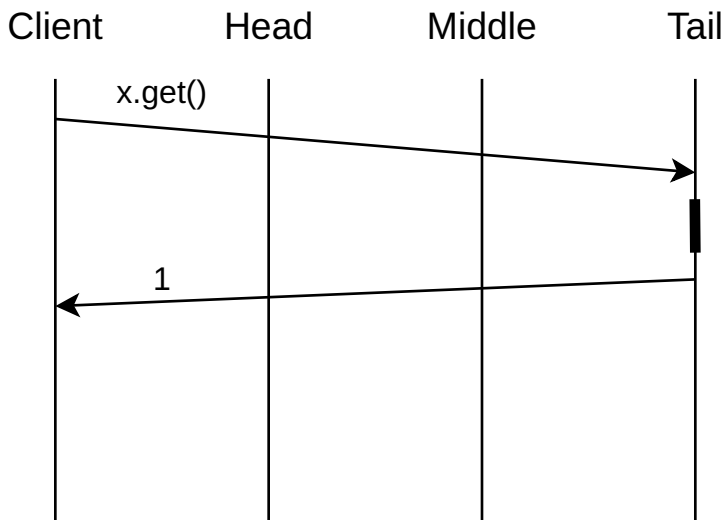
Put: write and wait for acknowledgment from backup.

Get: read and respond.

Chain Replication

Client sends puts to head and gets to tail.





Put: write and send on.

Get: read and respond.

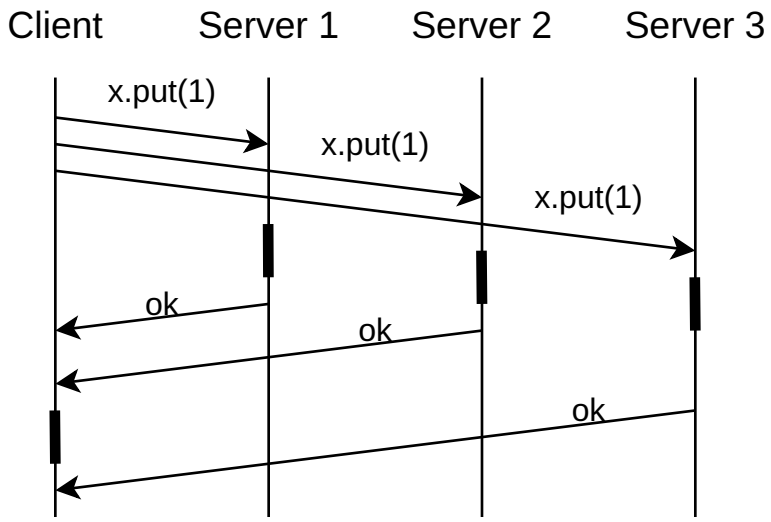
Quorum Protocols

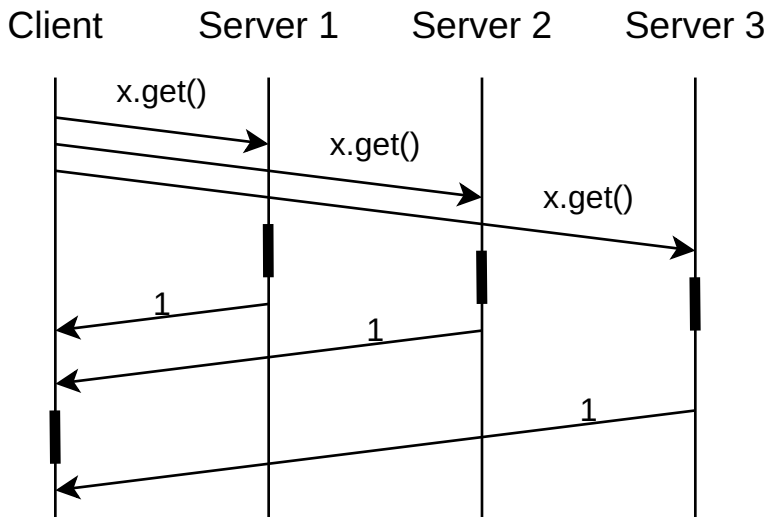
Client waits for enough acknowledgments. Tunable: R + W > N versus not.

N: number of replicas

W: votes required for write

R: votes required for read



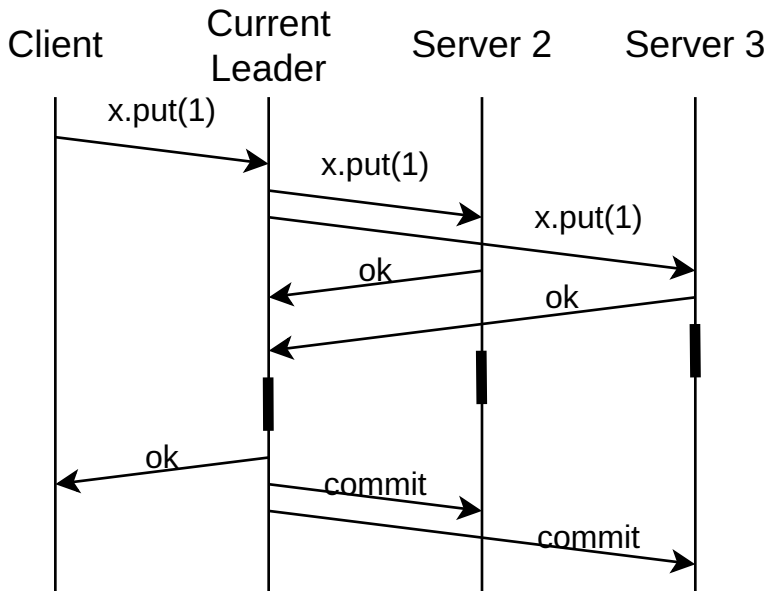


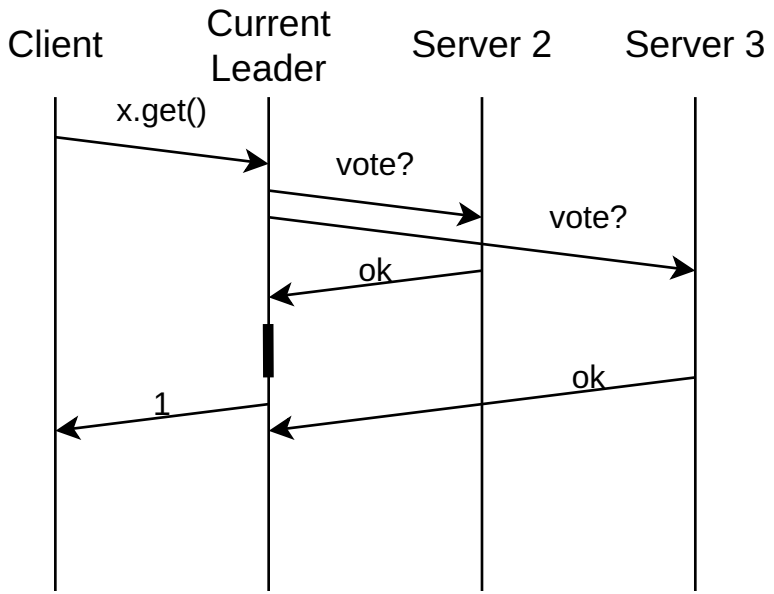
Put: write and acknowledge.

Get: read and respond.

Consensus Protocols

Client sends operation to any server.





Very complicated.

Paxos: Leslie Lamport. 1998. The part-time parliament.

Raft: Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm.

Machine Crashes

Active replication: tolerates 0 crashes for writes (N - 1) crashes for reads.

Primary backup replication: tolerates 1 crash with backup taking over.

Chain replication: tolerates (N - 1) crashes with bypassing in chain.

Quorum replication: tolerates (N - W) crashes for writes and (N - R) crashes for reads.

Consensus protocol: tolerates f failures when N = 2 * f + 1

Concurrent Operations

Only possible with mutliple cients.

Active replication: replicas receive and execute operations in different order.

Primary backup replication: all clients talk to same server which orders operations.

Chain replication: clients talk to the current head which orders operations.

Quorum replication: replicas receive and execute operations in different order.

Consensus protocol: clients talk to any server but current leader decides order.

Ordering Schemes

etcd (CoreOS): consensus-based total order via Raft

Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm.

Dynamo (Amazon): vector clocks and reconciliation

Giuseppe DeCandia, et al. 2007. Dynamo: amazon's highly available key-value store.

Spanner (Google): real time with global positioning system and atomic clocks

James C. Corbett, et al. 2013. Spanner: Google’s Globally Distributed Database.

Calvin (Yale): sequencer processes

Alexander Thomson, et al. 2012. Calvin: fast distributed transactions for partitioned database systems.

Performance

Read Latency

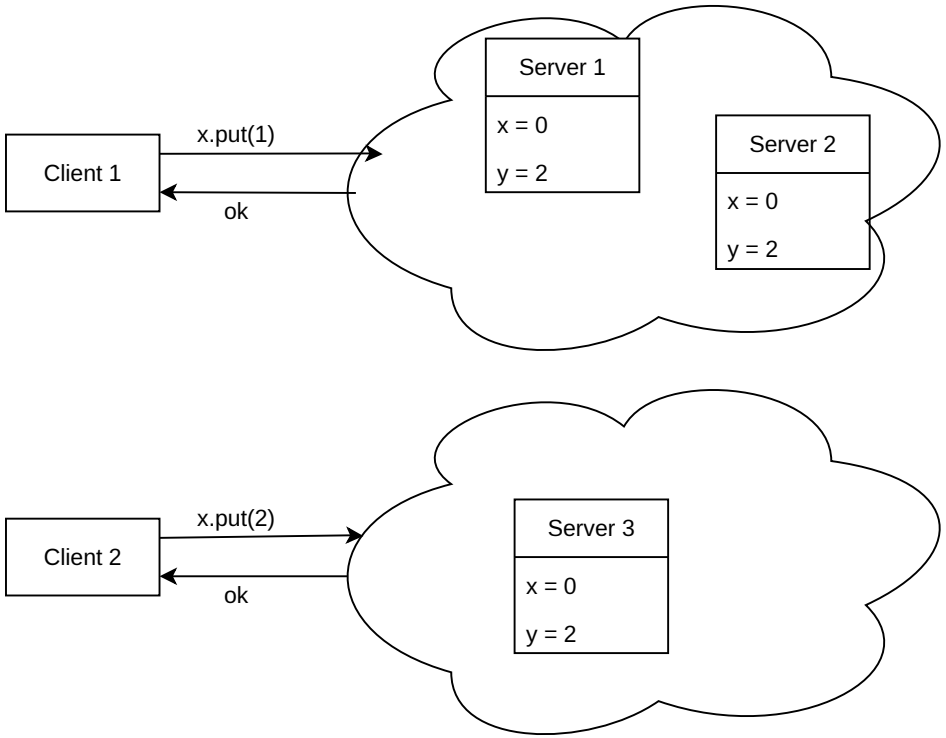
Active replication: low
Primary backup replication: low
Chain replication: low
Quorum replication: tunable
Consensus protocol: medium

Write Latency

Active replication: low
Primary backup replication: medium
Chain replication: high
Quorum replication: tunable
Consensus protocol: high

Network Partitions

The network splits into multiple parts that can not communicate anymore.



CAP theorem: during the partition, choose between consistency and availability.

Consistency Levels

Linearizability: Every operation appears to execute atomically at some point between invocation and response, respecting real-time order.

Sequential Consistency: All operations appear in the same total order to all processes, preserving each process's program order.

Causal Consistency: Operations that are related via happened-before are seen in the same order by all processes, concurrent operations may be seen differently.

Consistency

Active replication: sequential consistency by using logical clocks.

Primary backup replication: linearizable by single sequencer.

Chain replication: linearizable by single sequencer.

Quorum replication: sequential consistency by using logical clocks.

Consensus protocol: linearizable by using lots of things.

Temporal Logic

Propositional logic extended with two modalities: □ (always) and ◇ (eventually).

Axioms:

```
□(P → Q) → (□P → □Q)
□P → P
□P → □□P

◇(P → Q) → (◇P → ◇Q)
P → ◇P
◇◇P → ◇P
```

Safety properties: bad things never happen.
Liveness properties: good things eventually happen.

https://lamport.azurewebsites.net/tla/science-book.html

TLA+

https://lamport.azurewebsites.net/tla/tla.html

```
----------- MODULE HourClock ---------------
EXTENDS Naturals

VARIABLES hour

Init == hour = 1

Next == hour' = IF hour = 12 THEN 1 ELSE hour + 1

Spec == Init /\ [][Next]_hour
=========================================
```

Designed by Leslie Lamport, first appeared 1999.

Quint

https://quint-lang.org

```
var balances: str -> int

pure val ADDRESSES = Set("alice", "bob", "charlie")

action withdraw(account, amount) = {
  balances' = balances.setBy(account, curr => curr - amount)
}

val no_negatives = ADDRESSES.forall(addr =>
  balances.get(addr) >= 0
)
```

Designed by the Informal Systems team, first released 2022.

Modern syntax and tooling for TLA+.

demo/basics.qnt