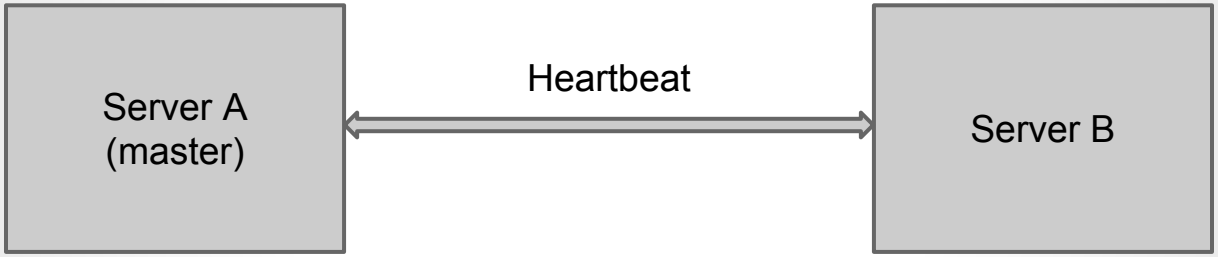
Distributed Consensus Algorithms

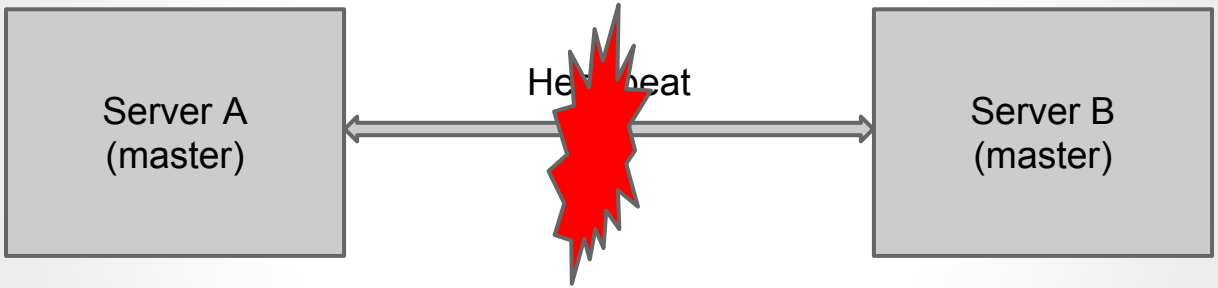
for extreme reliability

Laura Nolan (Google)

CONSENSUS

The road to true and lasting bliss


Server A
(master)
Heartbeat
Server B


Server A
(master)
Server B
(master)

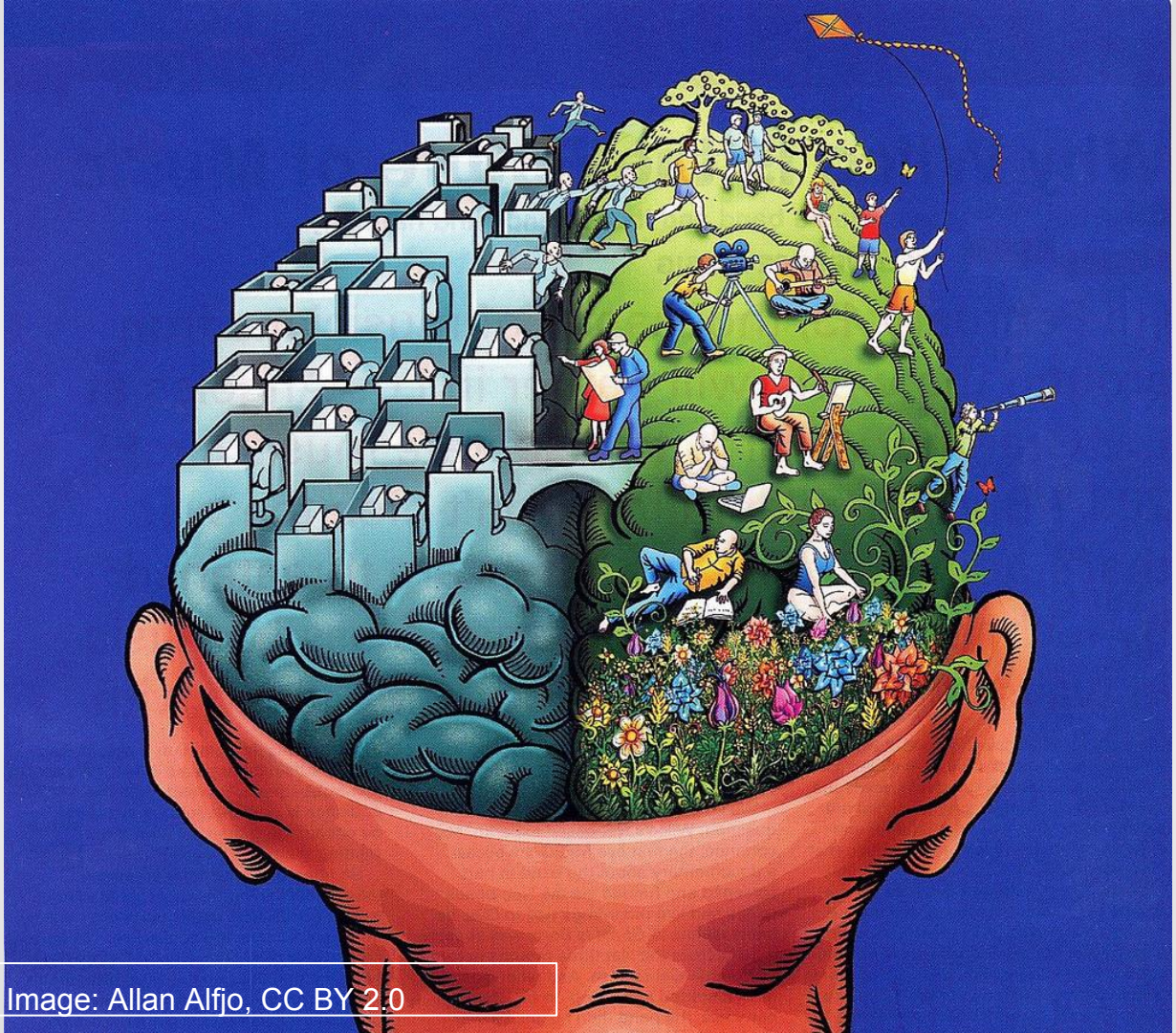
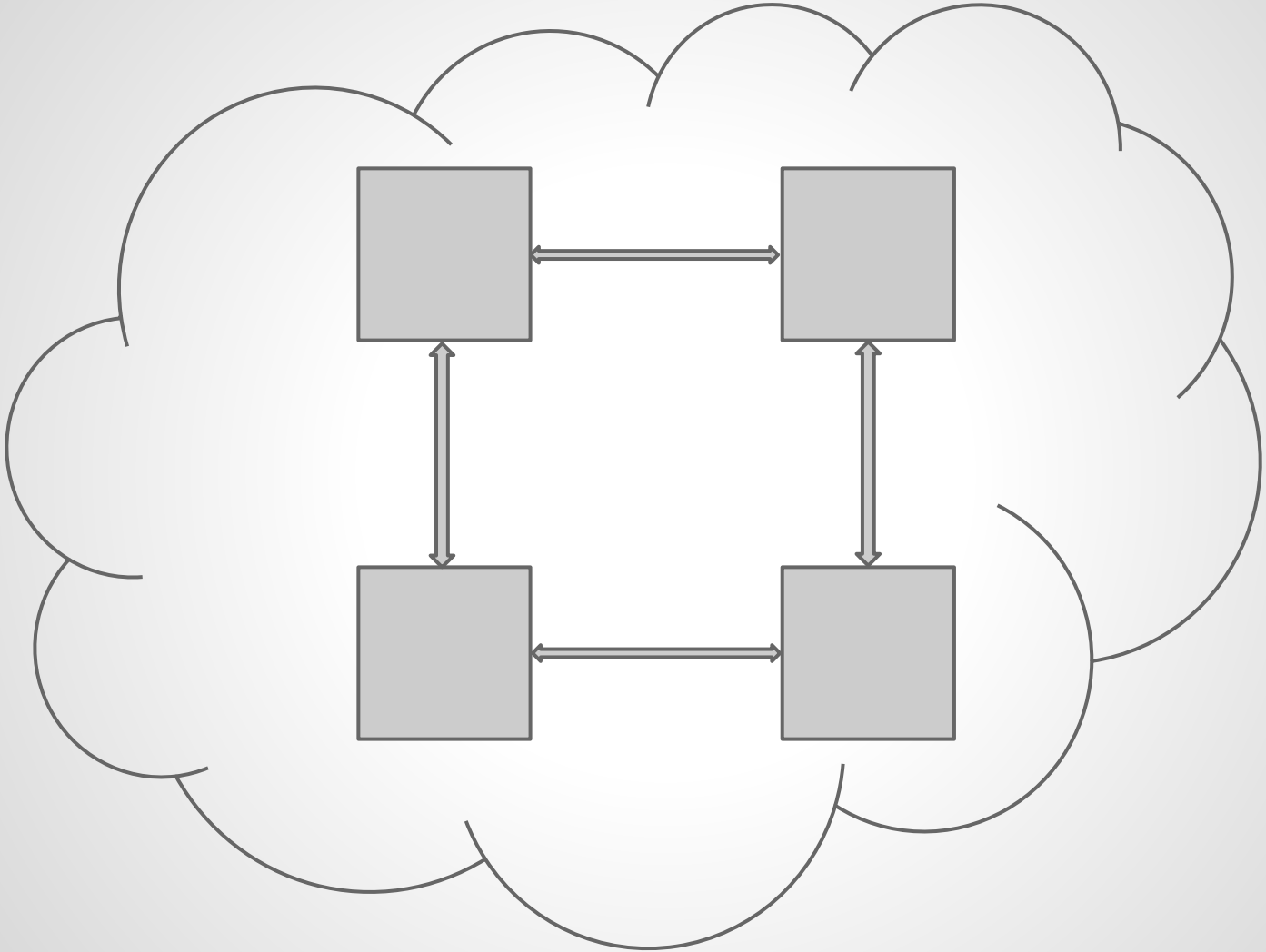


Image: Allan Alfjo, CC BY 2.0



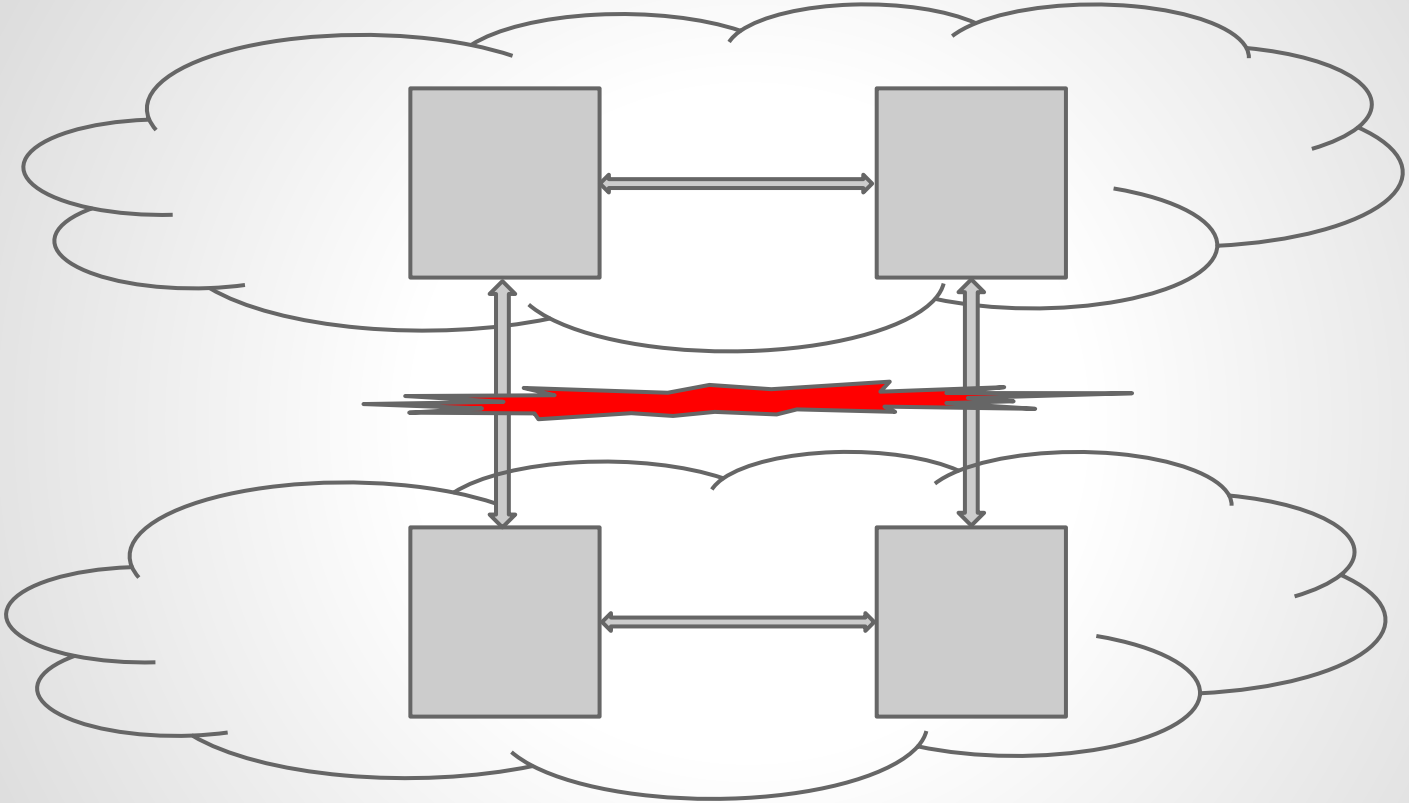


Image: John (irishwildcat - flickr), CC BY 2.0

Image: Drew from Zhodague, CC BY 2.0

The distributed consensus problem deals with reaching agreement among a group of processes connected by an unreliable communications network.

Distributed Consensus: a brief history

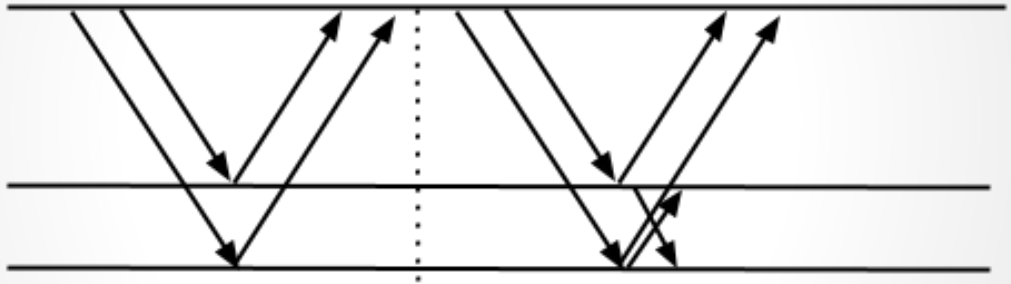
- 1985: FLP impossibility paper
- Late 1980s: Leslie Lamport invents Paxos on a dare
- 1990s: everyone* ignores Paxos (confused)
- 2001: 1985 FLP impossibility paper wins Dijkstra prize
- Distributed systems become pretty important
- 2006: Chubby paper published
- 2009: Zookeeper released
- 2010s: explosion of research; etcd and doozer released

Phase 1: Proposer sends Prepare message: with a new View number and a transaction number

Phase 2: Proposer sends Accept message with view and transaction numbers as well as the value proposed

Proposer

Acceptors



Acceptors respond with a Promise message: this means that the new view is accepted and proposals will not be accepted with a lower view number or transaction number

Acceptors respond by sending Accepted messages to all other members of the group (unless they have Promised a higher transaction number in the interim)

Other consensus algorithms

- Viewstamped Replication
- RAFT
- ZAB
- Mencius
- Many variants of Paxos (Fast Paxos, Egalitarian Paxos etc)

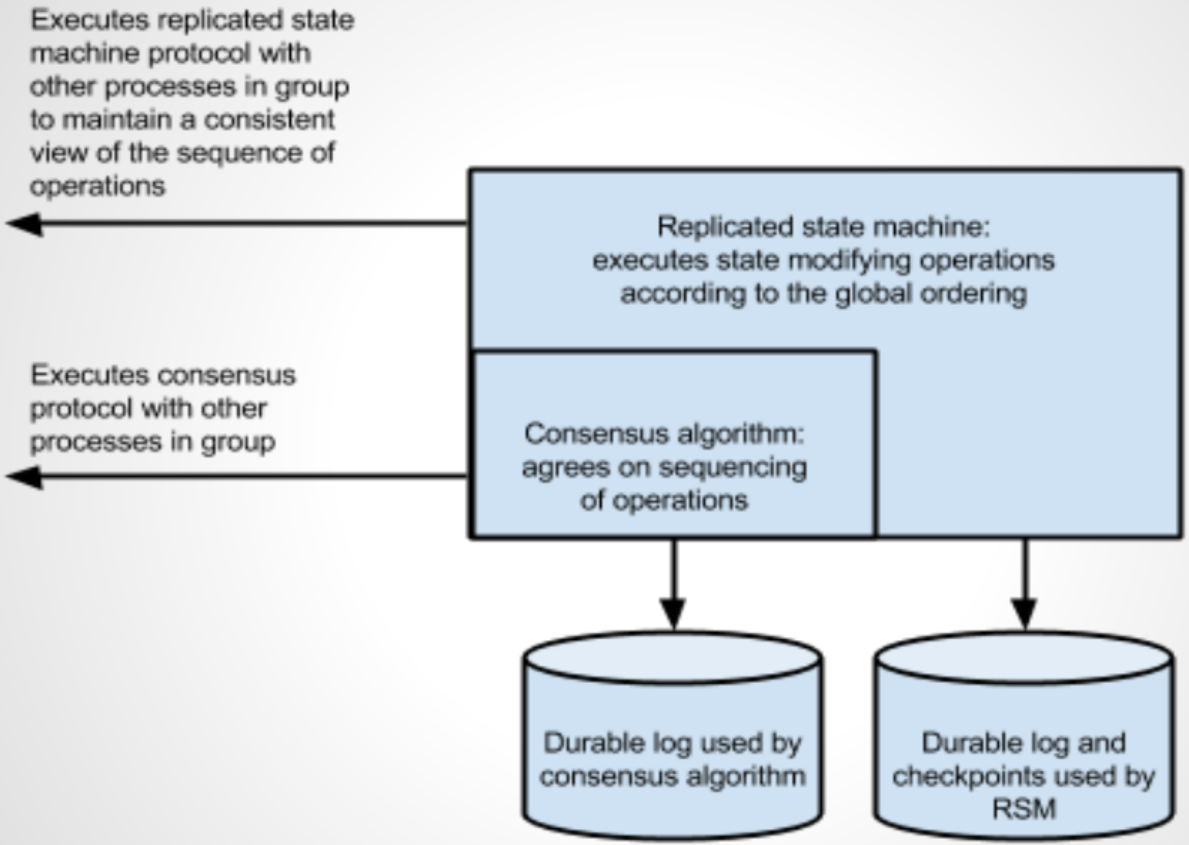
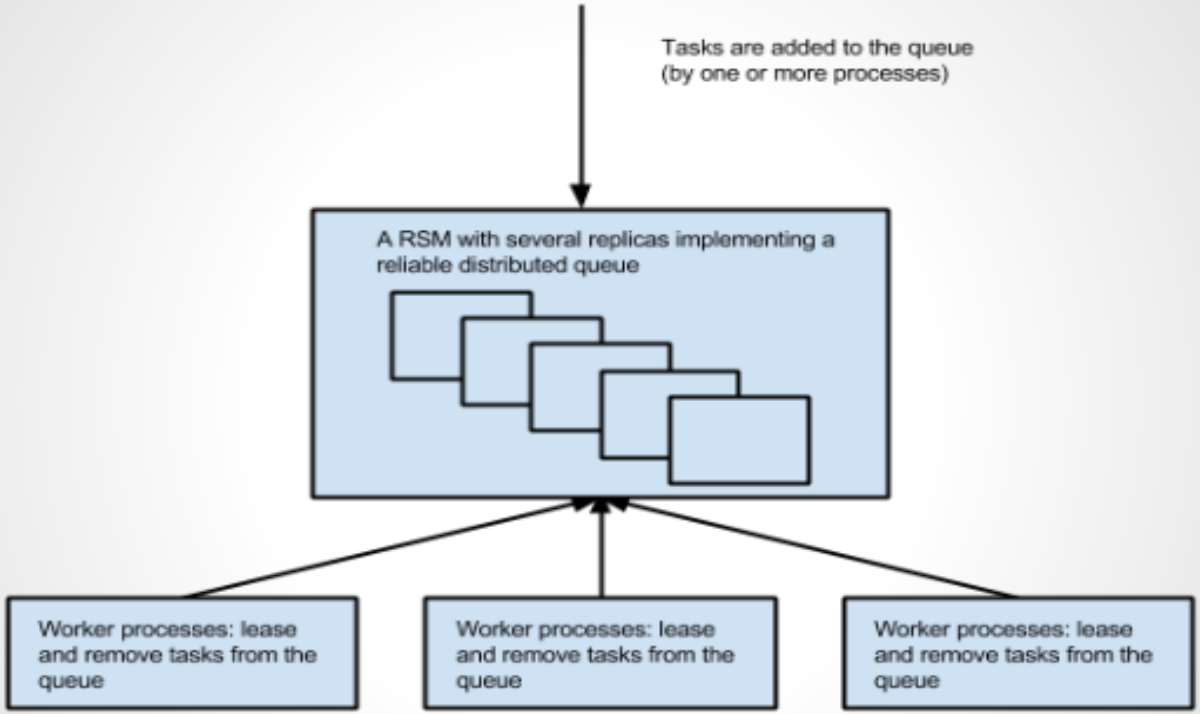

Executes replicated state machine protocol with other processes in group to maintain a consistent view of the sequence of operations
Replicated state machine: executes state modifying operations according to the global ordering
Executes consensus protocol with other processes in group
Consensus algorithm: agrees on sequencing of operations
Durable log used by consensus algorithm
Durable log and checkpoints used by RSM


Image: 55thstreet at flickr, CC BY-ND 2.0


Tasks are added to the queue
(by one or more processes)
A RSM with several replicas implementing a reliable distributed queue
Worker processes: lease and remove tasks from the queue
Worker processes: lease and remove tasks from the queue
Worker processes: lease and remove tasks from the queue

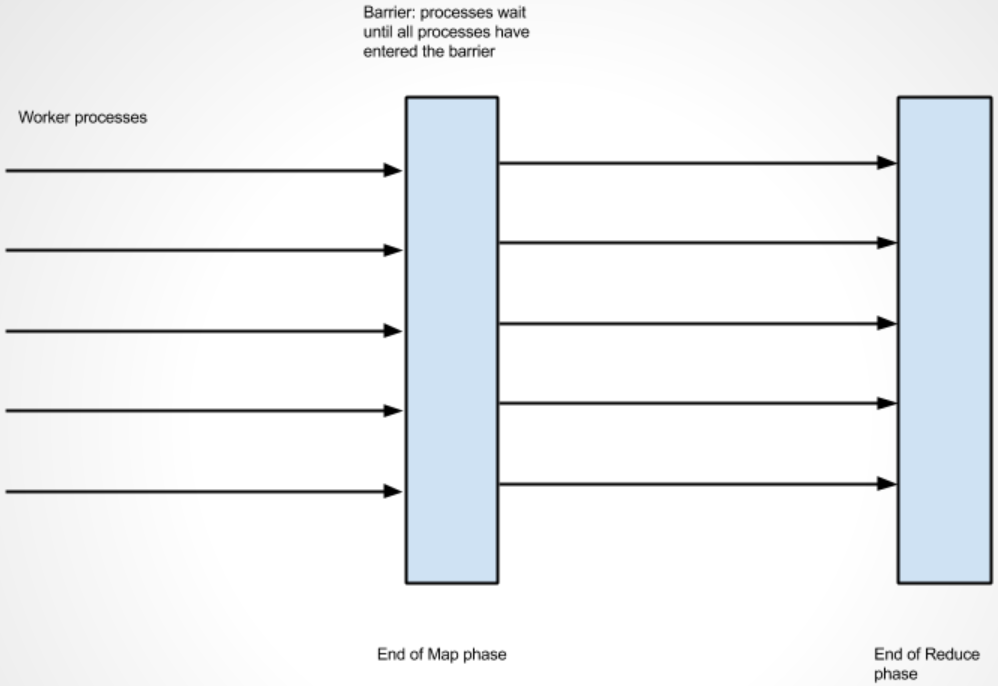
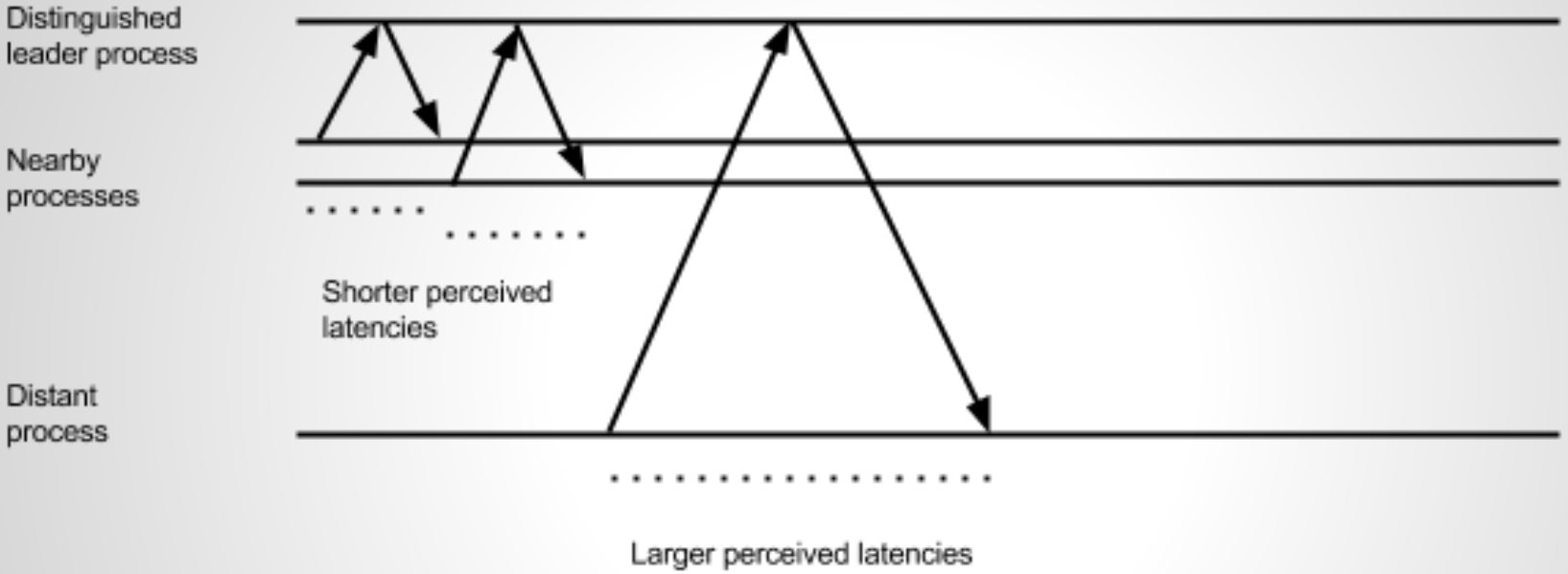
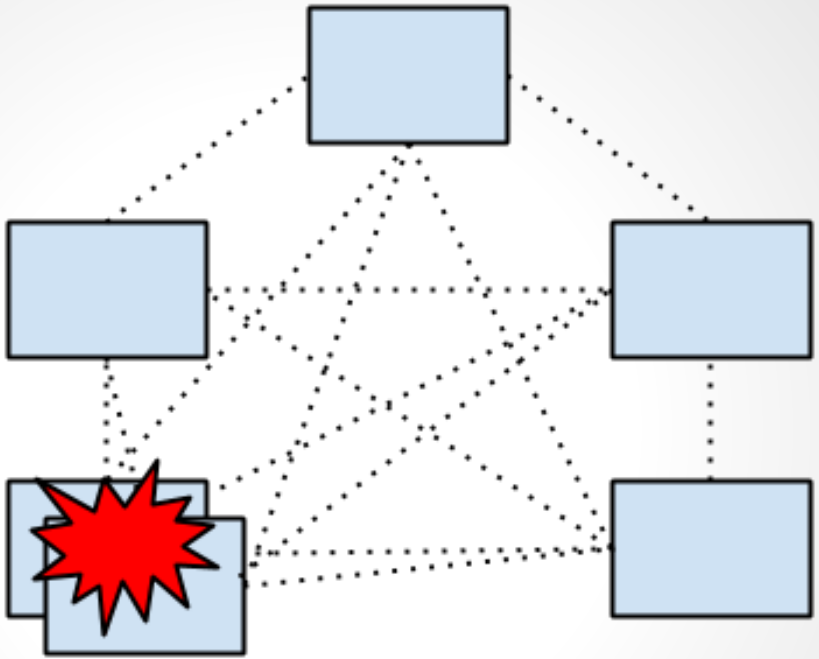

Barrier: processes wait until all processes have entered the barrier
Worker processes
End of Map phase
End of Reduce phase

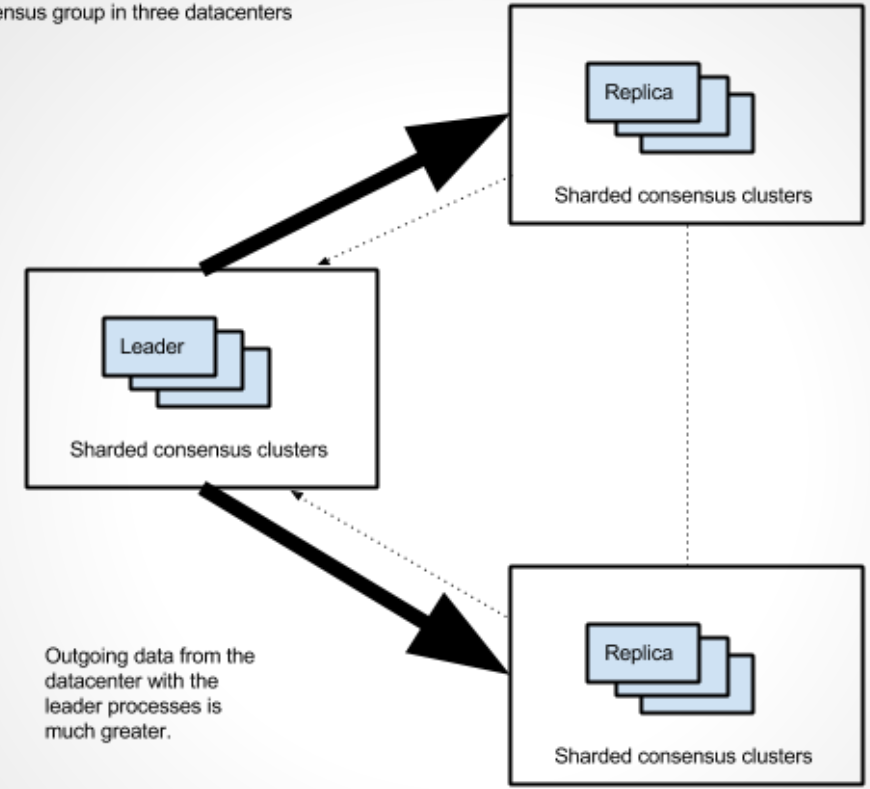
Image: lamantin at flickr, CC BY 2.0


Distinguished leader process
Nearby processes
Shorter perceived latencies
Distant process
Larger perceived latencies

Two replicas in a single datacenter: leaves only a quorum with no redundancy if failure occurs here

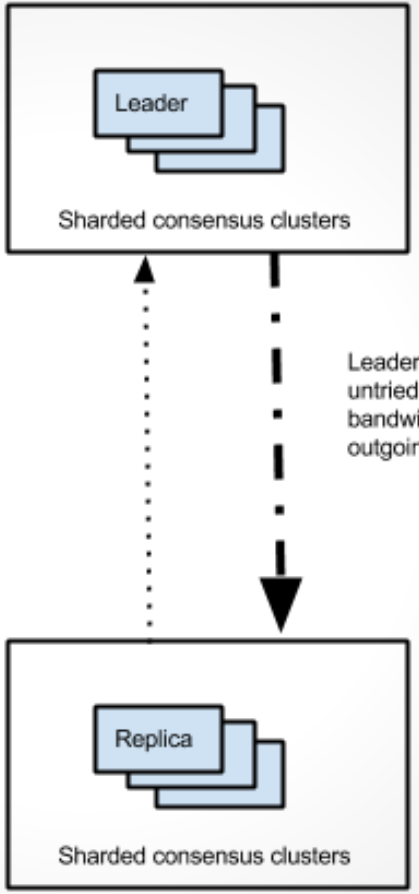


A highly-sharded consensus system running with replicas for each consensus group in three datacenters



Outgoing data from the datacenter with the leader processes is much greater.

A highly-sharded consensus system running with replicas for each consensus group in three datacenter: one fails



Leaders fail over en-masse to another, untried datacenter: insufficient bandwidth is available there for their outgoing traffic.

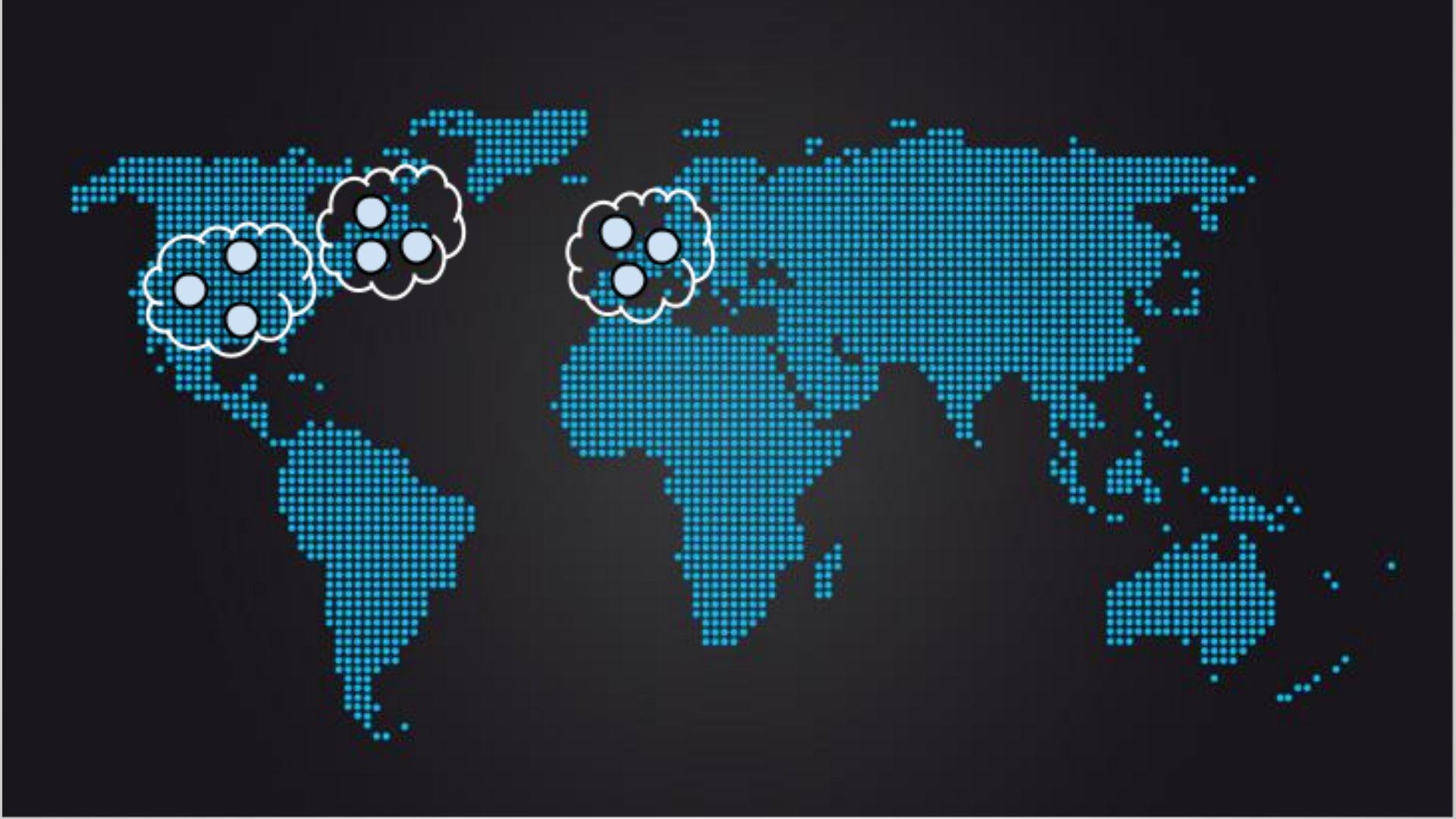


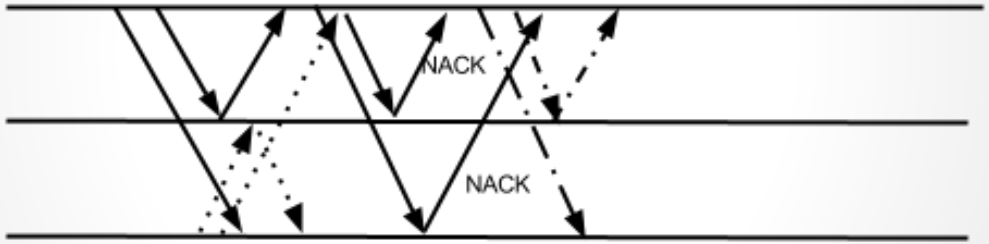
Image: uwdigitalcollections at flickr, CC BY 2.0

Process 1 sends Prepare message with a new View number and a transaction number. Process 2 responds with a Promise message.

Process 1 sends Accept for its proposal but Process 2 and 3 cannot accept its proposal because Process 3 has Proposed in the interim and Process 2 has promised.

Process 1 makes another attempt, with a higher transaction and view number. Process 2 promises, which means that Process 3's proposal can not be accepted. The cycle can repeat indefinitely.

Processes in the consensus group



Process 3 sends a conflicting Prepare message, to which Process 2 responds with a Promise message. Process 1 does not receive the message (or it is delayed).

Monitoring

- Number of instances up
- Health/status - healthy, lagging/catching up, unhealthy
- Mastership changes
- Transaction ID - is it increasing
- Plus usual things such as errors, request latency distributions

Further Reading

How to build a highly-available system using Distributed Consensus, Butler Lampson [http://goo.gl/pPp1Tz]

The Consensus Protocols series by Henry Robinson:

- Two-phase commit [http://goo.gl/xobNF6]
- Three-phase commit [http://goo.gl/wMl4ig]
- Paxos [http://goo.gl/jPpwHf]

Paxos Made Live, Tushar Chandra et al [http://goo.gl/Vaps3V]