

A GLIMPSE INTO DISTRIBUTED SYSTEMS

briefly, slowly, humbly

Xiaoguang Sun
Moca Team, Hoolai Games Ltd.



- ▶ Tyler Treat  @tyler_treat  github.com/tylertreat  bravenewgeek.com
- ▶ Amazon
- ▶ Google
- ▶ Open Source Community

CREDITS

Over 50% of content come from Tyler Treat's Introduction to Distributed System presentation, many thanks to his work, greatly reduced my load to prepare the tech talk.

- ▶ Thirty thousand foot view
 - ▶ Why
 - ▶ Fallacies
 - ▶ Characteristics
 - ▶ Pitfalls
- ▶ Above the sea
 - ▶ Byzantine Generals and Consensus
 - ▶ Brain split
 - ▶ Consistency
 - ▶ Scaling
- ▶ Fancy world
 - ▶ Dynamo/GFS/BigTable
 - ▶ Map Reduce/MPP/Streaming Processing

TOPICS

In the beginning God created the heavens
and the earth.
World begins...



What you are seeing and hearing these days, mostly derived from fundamental works
other people have done decades ago.

Paxos: 1989

Lamport timestamps: 1978

Vector clock: 1988

"A distributed system is one in which the failure of a computer you
didn't even know existed can render your own computer
unusable."

- Leslie Lamport

For example, alipay system outage makes you impossible to make a purchase on your computer or other connected devices.

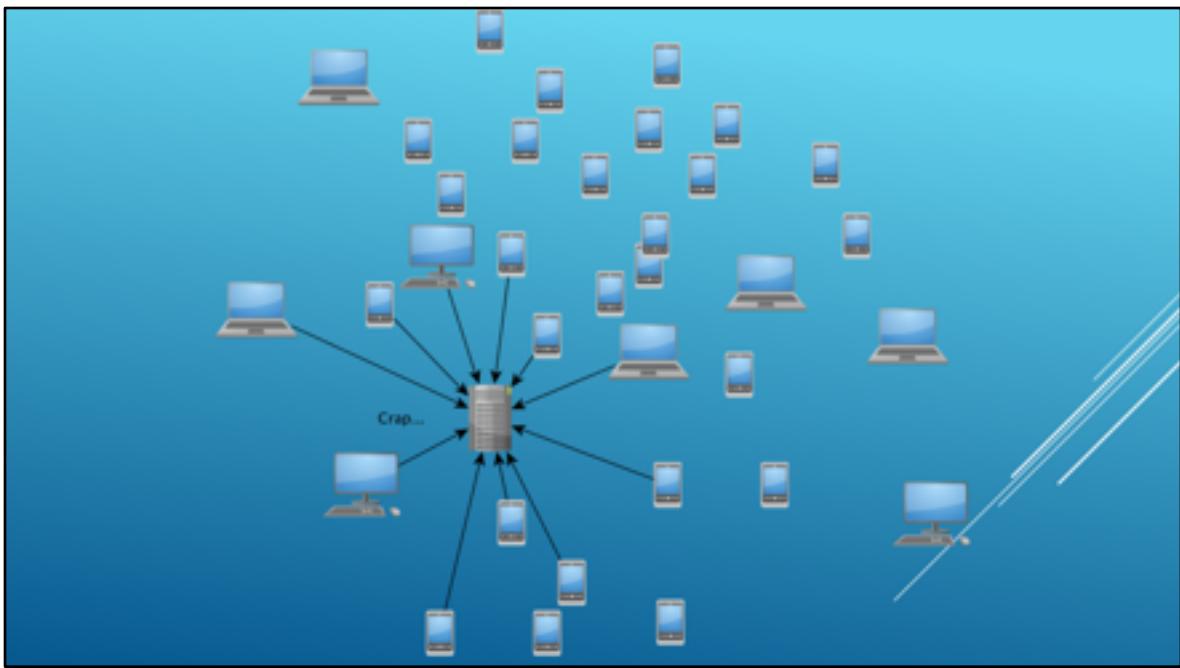
"A **distributed system** is a collection of **independent** computers that behave as a **single coherent system**."

- Andrew Tanenbaum

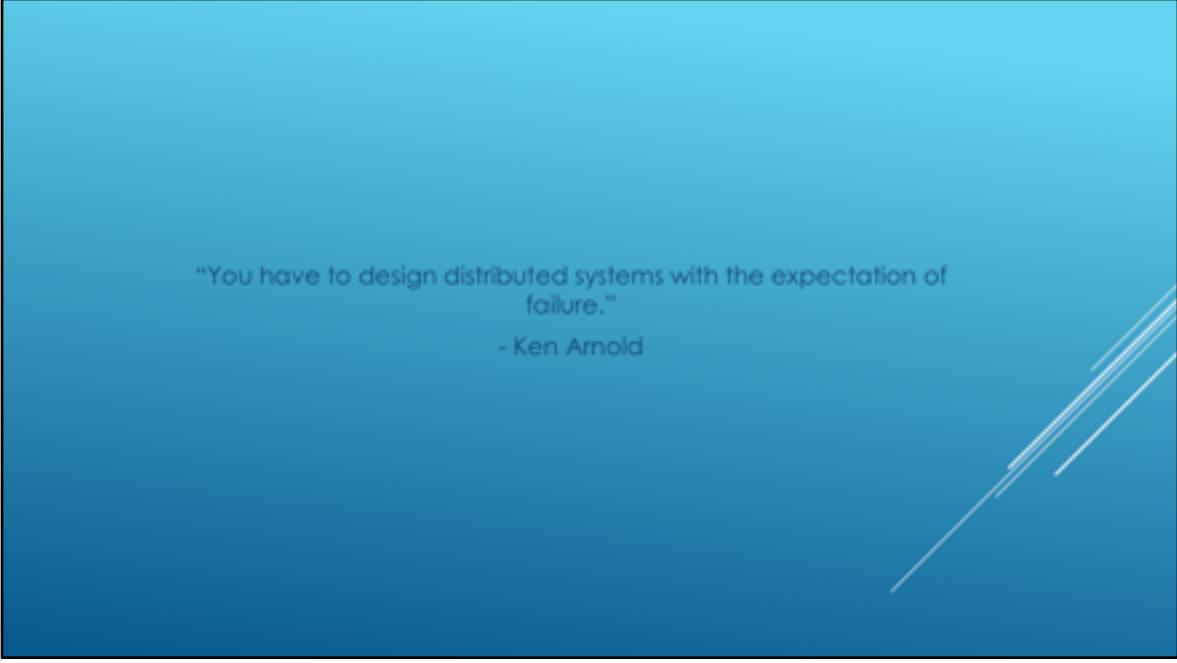
Tanenbaum was the professor when Linus was studying operating system.

- ▶ Availability serve every single request
- ▶ Fault Tolerance resilient to failures
- ▶ Throughput parallel computation
- ▶ Architecture decoupled, focused service
- ▶ Economics scale-out is manageable and cost-effective

WHY



This is why



"You have to design distributed systems with the expectation of failure."

- Ken Arnold

Shit happens, So

- ▶ The network is reliable
 - ▶ Message delivery is never guaranteed
 - ▶ Best effort
 - ▶ Is it worth it
 - ▶ Resiliency/redundancy/failover



FALLACY #1

Now, let's see what common issues could be which you might not realize their existence at all.

We can only try our best to deliver, no guarantee from network itself.

- ▶ Latency is zero.
 - ▶ Physics, Physics, Physics
 - ▶ LAN to WAN deteriorates quickly
 - ▶ Minimize network calls
 - ▶ Design asynchronous systems.



FALLACY #2

Distance from China to USA is 11640 KM, a round trip is 23280 KM. Light travels about 213000 KM per second within optical fiber. So it takes about 0.1 second for light to travel. TCP 3 way handshake itself already take 0.15 second.

What we can do?

Minimize network calls

Design asynchronous systems that do not block. But remember, latency doesn't go away, these can only improve overall throughput not making a single request perform beyond limits.

- ▶ Bandwidth is infinite.
 - ▶ Out of our control
 - ▶ Limit message sizes
 - ▶ Use message queueing



FALLACY #3

What we can do?

Limit message size, more compact protocol, e.g. binary protocol instead of text based. Compression. Design protocol to help compression algorithm.

Use message queueing, decouple producer and consumer, so burst traffic doesn't blow the whole system off.

- ▶ The network is secure
 - ▶ Everyone is out to get you
 - ▶ Build in security from day 1
 - ▶ Multi-layered
 - ▶ Encrypt, penetration test, train developers.



FALLACY #4

Security precaution at every layer

1. IPSec at IP layer
2. SSL at transportation layer
3. Multifactor authentication
4. Single sign-on
5. Fraud detection
6. Transaction signing and encryption

- ▶ Topology doesn't change
 - ▶ Network topology is dynamic
 - ▶ Don't statically address hosts
 - ▶ Collection of services, not nodes
 - ▶ Service discovery



FALLACY #5

Packets of a single TCP connection might go through completely different route to get you. TCP stack does the heavy lifting for you.

People who have developed udp, mcast which under the hood is also udp, applications know.

- ▶ There is one administrator
 - ▶ May integrate with third-party systems
 - ▶ Is it our problem or theirs?
 - ▶ Conflicting policies/priorities
 - ▶ Third parties constrain; weigh the risk



FALLACY #6

One administrator knows everything

Never true

- ▶ Transport cost is zero
 - ▶ Monetary and practical costs
 - ▶ Building/maintaining a network is not trivial
 - ▶ The "perfect" system might be too costly



FALLACY #7

No one is rich --- enough --- to ignore the cost.

- ▶ The network is homogeneous
 - ▶ Networks are almost never homogeneous
 - ▶ Third-party integration
 - ▶ Consider interoperability
 - ▶ Avoid proprietary protocols



FALLACY #8

Remember starcraft back to college? It uses IPX/SPX.

Heard of infiniband? It infiniband! Though you can still use TCP over it, performance wise sucks. Like driving Maserati on sideway

Application layers? Chaos!

These problems apply to LAN and WAN systems

(Single data center and cross data center)

No one is safe.

"Anything that can go wrong will go wrong"

- Murphy's law



- ▶ Fault tolerant nodes can fail
- ▶ Available serve all the requests, all the time
- ▶ Scalable behave correctly with changing topologies
- ▶ Consistent state is coordinated across nodes
- ▶ Secure access is authenticated
- ▶ Performance it's fast!

CHARACTERISTICS

Why fault tolerant? nodes can fail

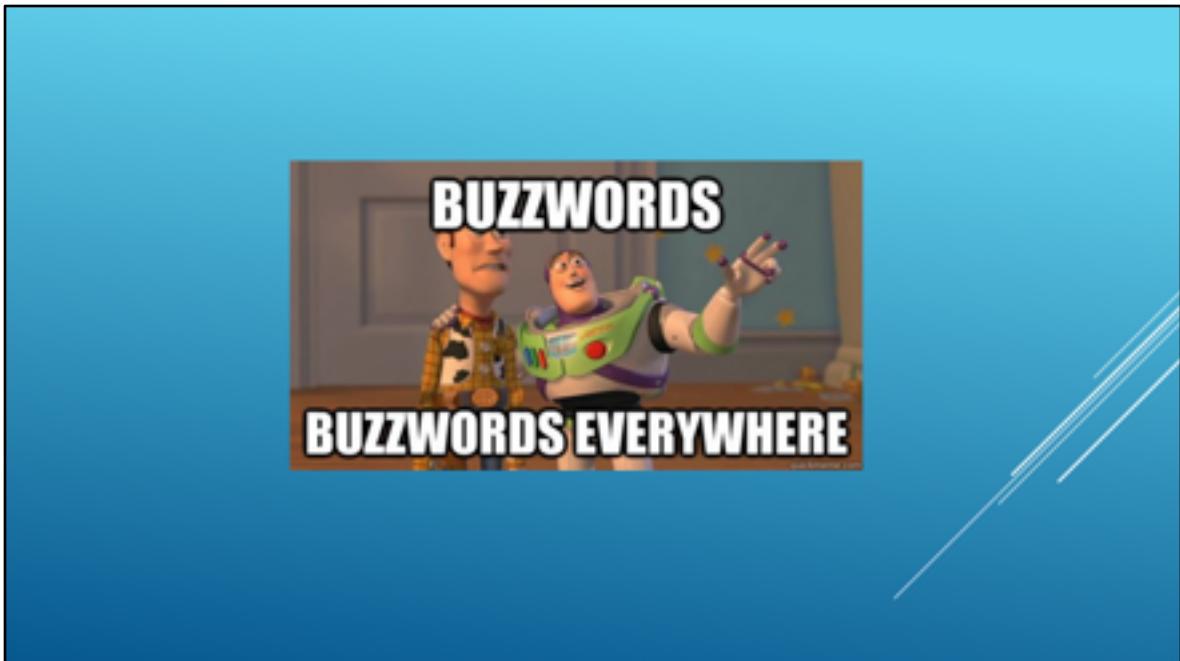
Why available? We want to serve all the requests, all the time

Why scalable? We want system behave correctly with changing topologies

Why consistent? We want the state to be coordinated across nodes

Why secure? We want every access to be authenticated

Why performance? We like fast

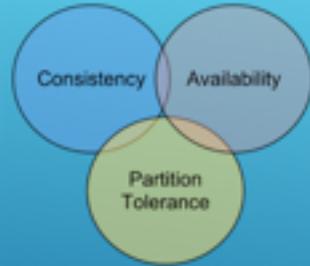


Hunting for buzzwords

Distributed systems are all
about **trade-offs**.

Before we start, please remember, tradeoff, tradeoff and tradeoff

- ▶ Presented in 1998 by Eric Brewer
- ▶ Impossible to guarantee all the three:
 - ▶ Consistency
 - ▶ Availability
 - ▶ Partition tolerance



CAP THEOREM

- ▶ Linearizable – there exists a total order of all state updates and each update appears atomic
- ▶ Mutexes make operations appear atomic
- ▶ When operations are linearizable, we can assign a unique "timestamp" to each one in total order
- ▶ A system is consistent if every node shares the same total order
- ▶ Consistency which is both global and instantaneous is impossible

CONSISTENCY

Way to consistency.

The truth is, instantaneous global consistency is impossible. Write takes time

- ▶ Eventual consistency: replicas allowed to diverge, eventually converge
- ▶ Strong consistency: replicas can't diverge requires linearizability

CONSISTENCY

So what we can get.

Eventual consistency, replicas of an object could diverge at some point, but converge eventually is going to happen. Dynamo is an example.

Strong consistency, replicas can't diverge if linearizability is guaranteed. Distributed Transaction is one example.

- ▶ Every request received by a non-failing node must be served
- ▶ If a piece of data required for a request is unavailable, the whole system is considered unavailable
- ▶ 100% availability is a myth

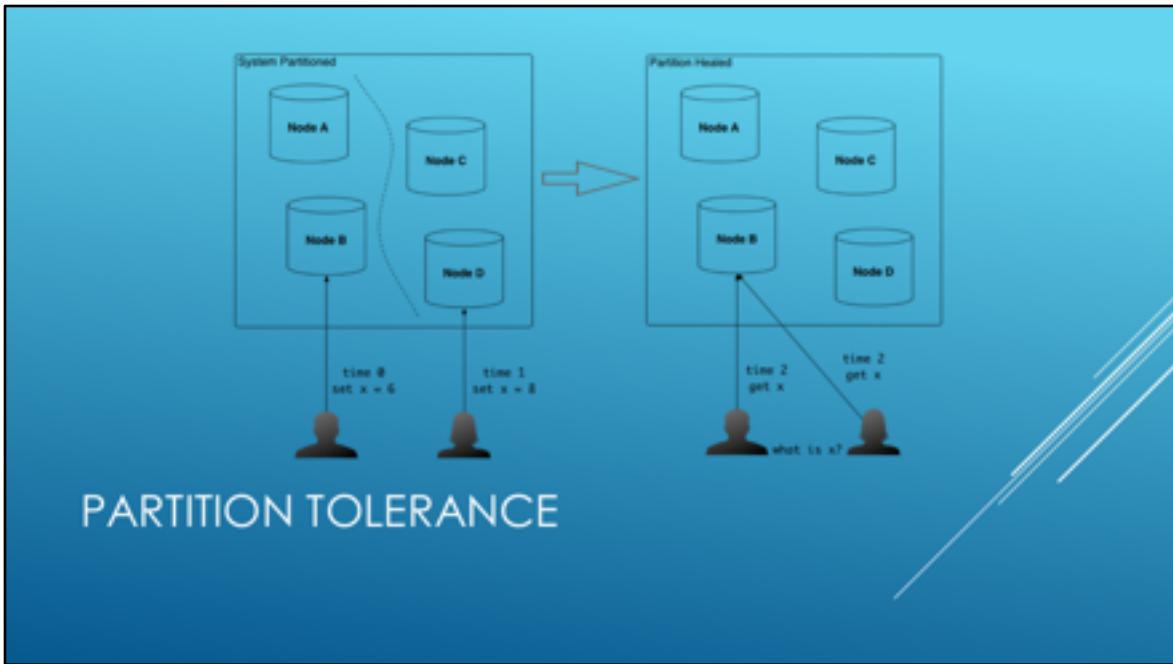
AVAILABILITY

Basically, serve every single request

- ▶ A partition is a split in the network - many causes
- ▶ Partition tolerance means partitions can happen
- ▶ CA is easy when your network is perfectly reliable
- ▶ Your network is **not** perfectly reliable

PARTITION TOLERANCE

CAP is only possible when your network is 100% reliable.



What will happen if we choose to sacrifice consistency and choose to continue serve requests when partition happens.

- ▶ Halting failure Machine stops
- ▶ Network failure Network connection breaks
- ▶ Omission failure Messages are lost
- ▶ Timing failure Clock skew
- ▶ Byzantine failure Arbitrary failure

PITFALLS

Literally every single thing you could ever imagine and be afraid of, will happen.

Machine could stop running cause service to halt

Network could break cause service to be unreachable.

Messages could get lost

Time skew could make timeout based failure detection shaky

Arbitrary failure, yes, it's called byzantine failure

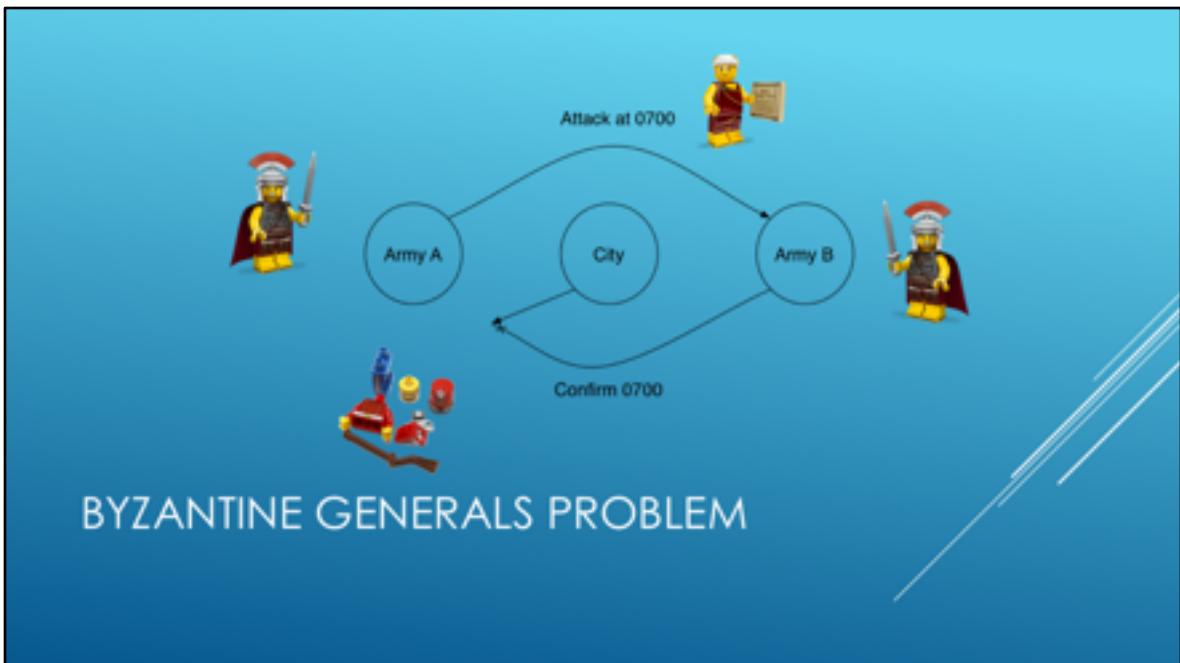
A little Deeper
Exploring some high-level concepts

Wow, concepts.

- ▶ Consider a city under siege by two allied armies
- ▶ Each army has a general
- ▶ One general is the leader
- ▶ Armies must agree when to attack
- ▶ Must use messengers to communicate
- ▶ Messengers can be captured by defenders

BYZANTINE GENERALS PROBLEM

Let's have a day dream



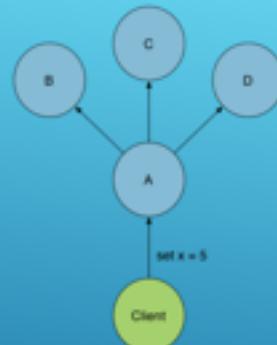
So what can we do

- ▶ Send 100 messages, attack no matter what
 - ▶ A might attack without B
- ▶ Send 100 messages, wait for acks, attack if confident
 - ▶ B might attack without A
- ▶ Messages have overhead
- ▶ Can't reliably make decision (proven impossible)

BYZANTINE GENERALS PROBLEM

Not so much

- ▶ Replace 2 generals with N generals
- ▶ Generals must agree on decision
- ▶ Solutions:
 - ▶ Multi-phase commit
 - ▶ State replication

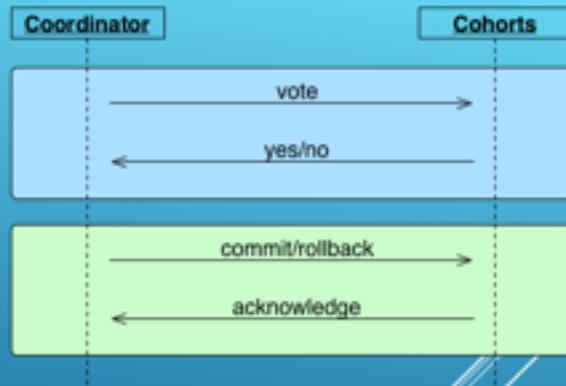


DISTRIBUTED CONSENSUS

Technology perspective

- ▶ Blocking protocol
- ▶ Coordinator waits for cohorts
- ▶ Cohorts wait for commit/rollback
- ▶ Can deadlock

Vote
Commit



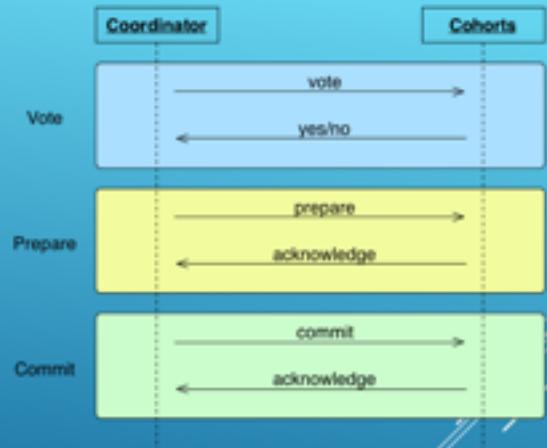
TWO-PHASE COMMIT

If coordinator dies in middle, cohorts wait.

Any cohort dies, coordinator waits forever.

In the case ack loses, the state of the particular cohort is in doubt. Cohorts or coordinator keep sending recover messages to each other

- ▶ Non-blocking protocol
- ▶ Abort on timeouts
- ▶ Susceptible to network partitions



THREE-PHASE COMMIT

Vote state: timeout causes abort

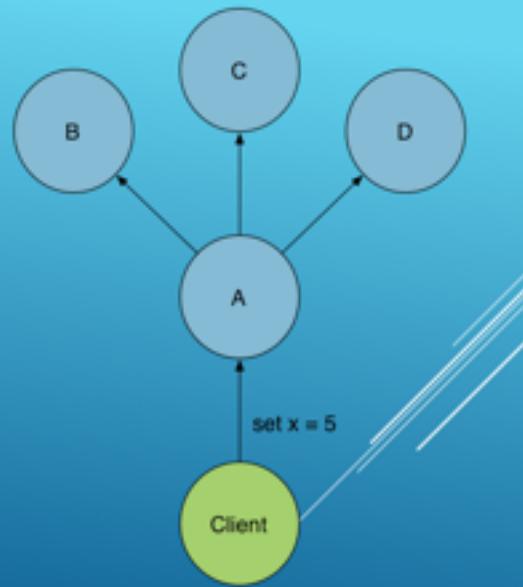
Prepare state: timeout causes state changing to commit on cohorts side as everyone would have made the commitment in vote state

Commit state: coordinator timeout lead to cohorts commit, cohorts timeout result in doubtful transaction and requires recovery protocol to clear the state

There is an enhanced three-phase commit that can survive from network partitions or quorum lost

- ▶ Paxos, Raft protocols
- ▶ Elect a leader
- ▶ All changes go through leader
- ▶ Each change appends log entry
- ▶ Each node has log replica

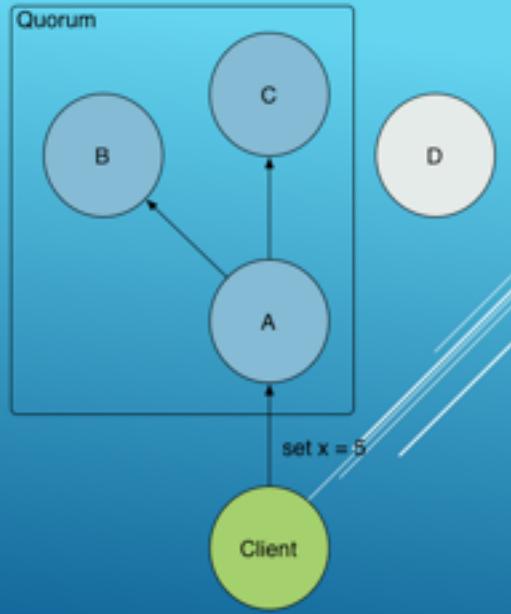
STATE REPLICATION

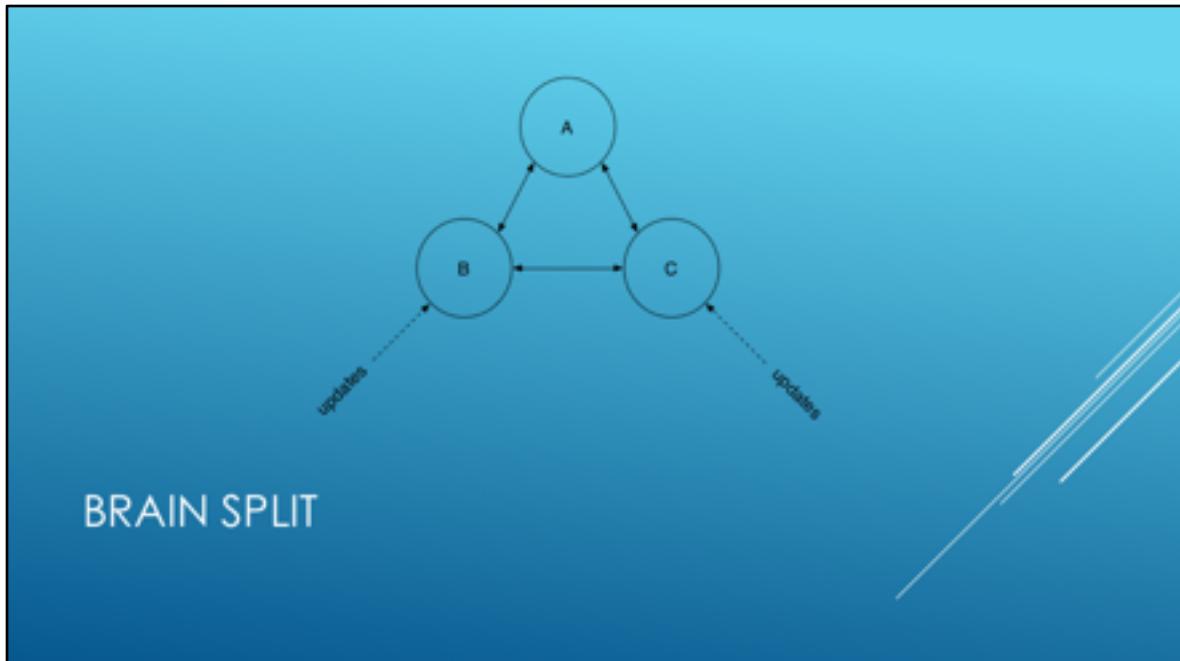


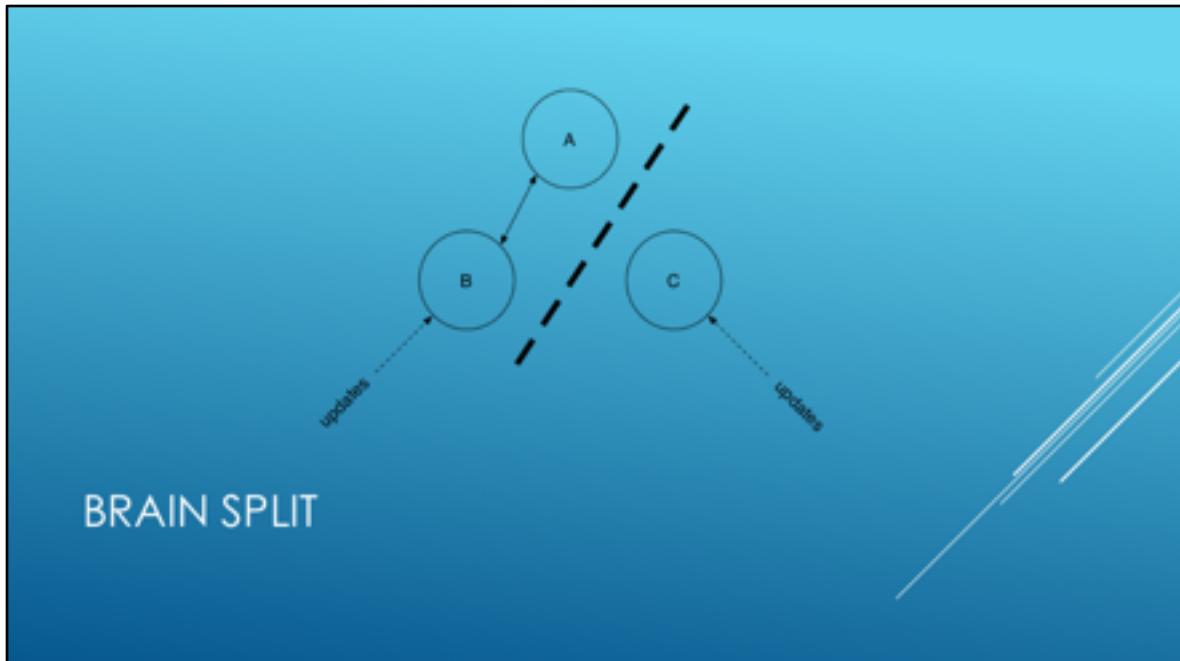
Yes, there are more.

- ▶ Must have quorum to proceed
- ▶ Commit once quorum acks
- ▶ Quorums mitigate partitions
- ▶ Logs allow state to be rebuilt

STATE REPLICATION







- ▶ Optimistic (AP) – Let partitions work as usual
- ▶ Pessimistic (CP) – Quorum partition works, fence others

BRAIN SPLIT

Tradeoff, tradeoff, tradeoff

Remember CAP theorem

Choose Availability + Partition Tolerance yields inconsistency

Choose Consistency + Partition Tolerance renders system unavailable for certain cases
depends on reliability requirement.

- ▶ Weak availability, low latency, stale reads
- ▶ Strong fresh reads, less available, high latency
- ▶ Hybrid
 - ▶ Weaker models when possible (like, followers, votes)
 - ▶ Stronger models when necessary
 - ▶ Tunable consistency models (Cassandra, Riak, etc)

HYBRID CONSISTENCY MODELS

Consistency Models:

Weak means better availability because you don't need to wait at least two or quorum to reply, but you could read old version of data

Strong requires at least quorum number of reads which means at least two for three replicas system. More roundtrips means higher latency, also means there must be at least quorum available replicas.

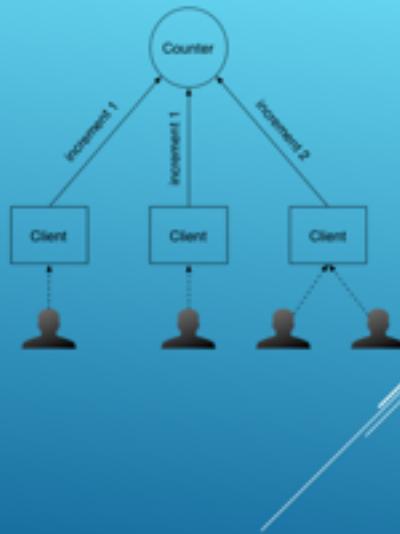
Of cause, tradeoffs exist.

- ▶ Sharing **mutable data** at large scale is difficult
- ▶ Solutions
 - ▶ Immutable data
 - ▶ Last write wins
 - ▶ Application level conflict resolution
 - ▶ Causal ordering (e.g. vector clocks)
 - ▶ Distributed data types (CRDTs)

SCALING SHARED DATA

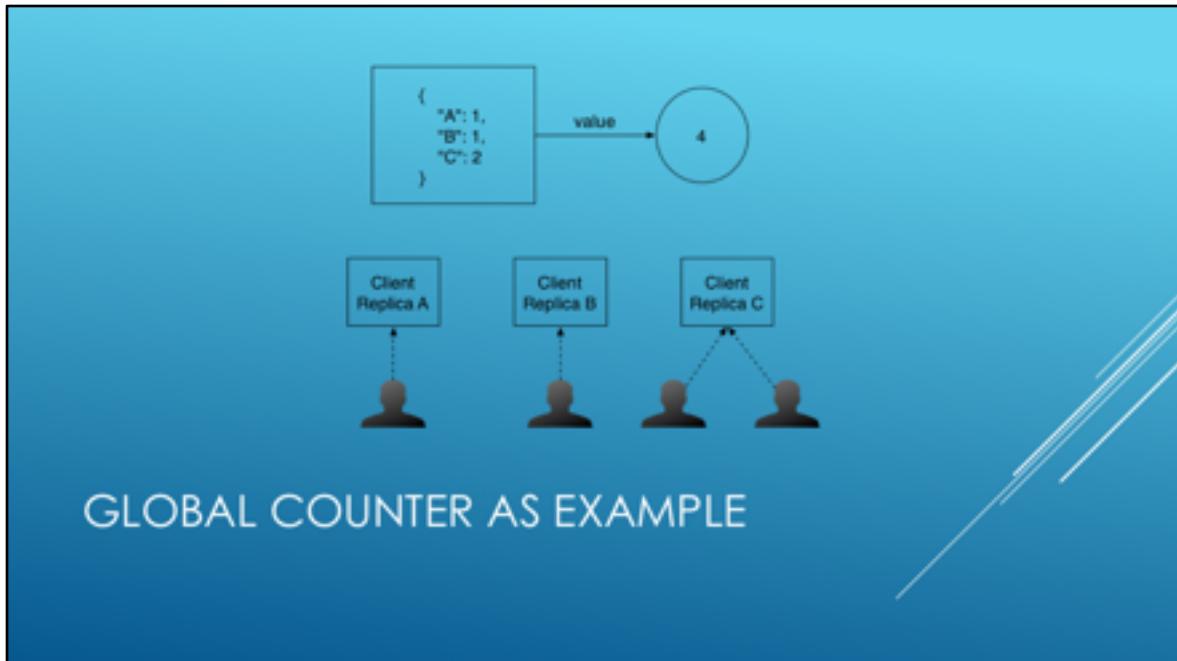
Imagine a shared, global counter

"Get, add, and put"
transaction will not scale



SCALING SHARED DATA

Mutex required



Set of operations being applied on the counter sequentially.

- ▶ Conflict-free Replicated Data Type
- ▶ Convergent: state-based
- ▶ Commutative: operations-based
- ▶ E.g. Distributed sets, lists, maps, counters
- ▶ Update concurrently without writer coordination

CRDT

Required properties: Convergence, commutation

- ▶ CRDTs always converge (provably)
- ▶ Operations commute
- ▶ Highly available, eventually consistent
- ▶ **Always reach consistent state**
 - ▶ Requires knowledge of all clients
 - ▶ Must be associative, commutative, and idempotent

CRDT

- ▶ Add to set is associative, commutative, idempotent
 - ▶ $\text{add}("a"), \text{add}("b"), \text{add}("a") \Rightarrow \{"a", "b"\}$
- ▶ Adding an removing item is not
 - ▶ $\text{add}("a"), \text{remove}("a") \Rightarrow \{\}$
 - ▶ $\text{remove}("a"), \text{add}("a") \Rightarrow \{"a"\}$
- ▶ CRDTs require interpretation of common data structure with limitations

CRDT

Why regular set as it is, is not conflict free in replicated environment.

- ▶ Use two sets, one for adding, one for removing
- ▶ Elements can be added once and removed once
- ▶ {

```
"a" : ["a", "b", "c"],  
"r" : ["a"]  
}  
=> {"b", "c"}  
add("a"), remove("a") => {"a" : ["a"], "r" : ["a"]} => {}  
remove("a"), add("a") => {"a" : ["a"], "r" : ["a"]} => {}
```

TWO-PHASE SET

Make it CRDT. Remember, it has limitation, not regular set in our mind.

Fancy world
Looks a lot like heaven,
tastes much like hell

WHERE
A LOT
MEANS
A LOT
MORE



- ▶ Object store
 - ▶ Amazon Dynamo
 - ▶ Riak
- ▶ File system
 - ▶ Google GFS
 - ▶ HDFS
- ▶ Structured store
 - ▶ Google BigTable
 - ▶ Hbase
- ▶ Block store
 - ▶ Amazon EBS
 - ▶ Ceph

A LOT OF STORES

- ▶ Bring to public at SOSP 2007
- ▶ Always writable at first place, can be tuned by N/W/R
- ▶ Decentralized
- ▶ Data partitioned using consistent hashing
- ▶ Flat namespace

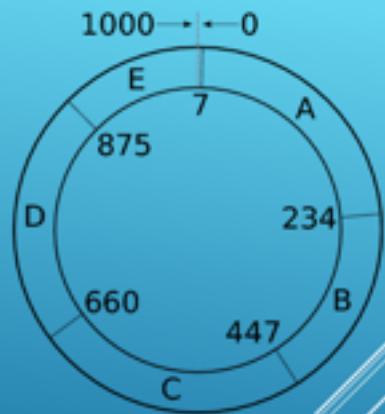
DYNAMO

Resize of hash table causes K/N keys to be relocated on average.

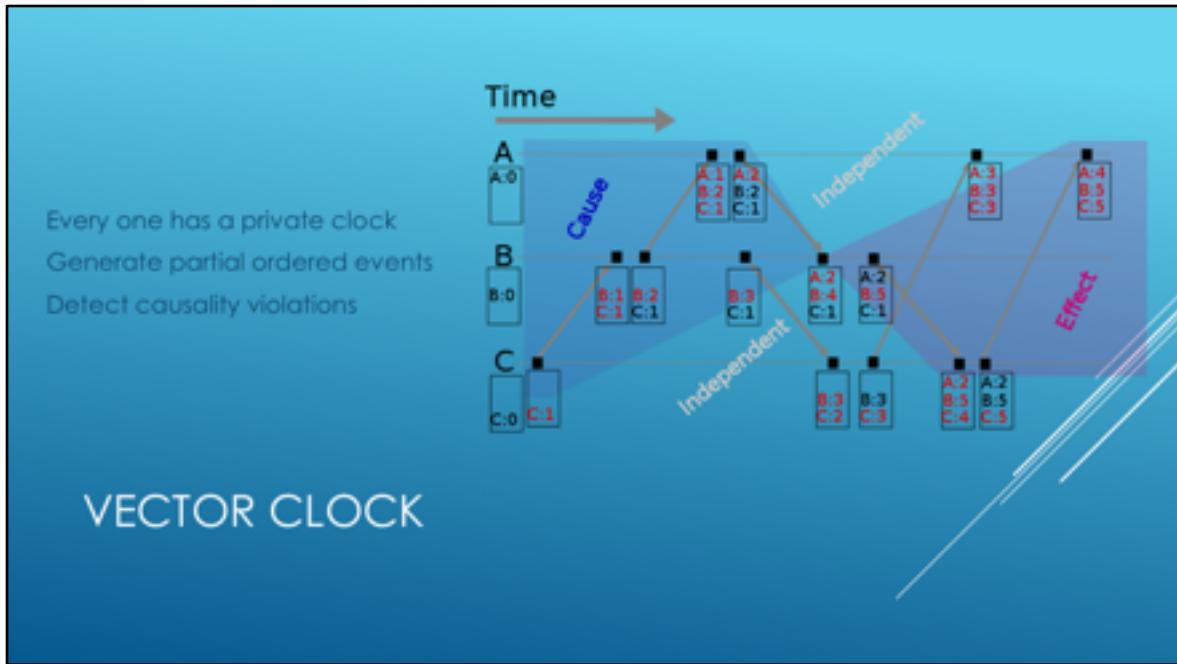
K : size of key space

N : number of partitions

$\text{key}' = F(\text{key})$



CONSISTENT HASHING



e.g.

Event 1->{B:1, C:1} is older than event 2->{A:2, B:5, C:1} because for each process in {A, B, C}.

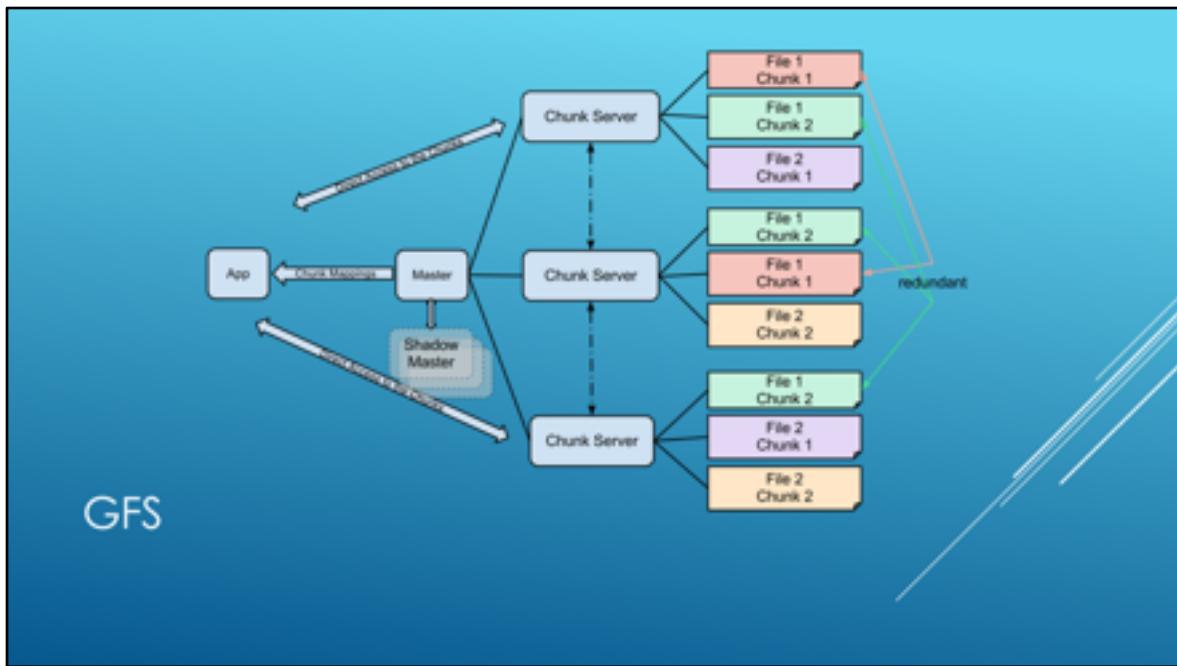
A:0 < A:2

B:1 < B:5

C:1 <= C:1

- ▶ Bring to public at SOSP 2003
- ▶ Meta/Chunk structured
- ▶ File system style hierarchical structure
- ▶ Targets for large files
- ▶ Fits offline processing better

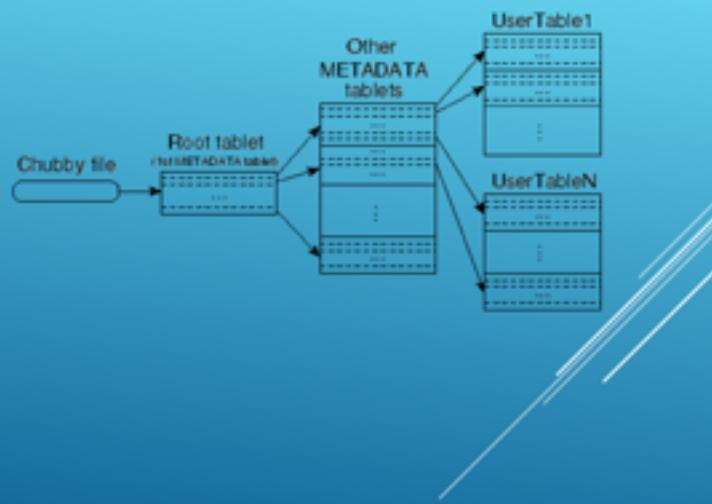
GFS



- ▶ Bring to public at OSDI 2006
- ▶ Structured but schema free
- ▶ Serves both throughput oriented and latency sensitive tasks well

BIGTABLE

- ▶ Layered on top of chubby, GFS
- ▶ Memtables and SSTables as storage engine



BIGTABLE

- ▶ As simple as object storage with integer key and fixed size value
- ▶ As complicate as Elastic Block Store
 - ▶ Snapshot
 - ▶ Resizable volume
 - ▶ Extremely low latency
 - ▶ Extremely high throughput

EBS

- ▶ Map Reduce
- ▶ MPP
- ▶ Of course, there are more. Different models, different implementations and different tradeoffs.

IT'S ALL ABOUT COMPUTING



Conceptually very simple

Two major functions:

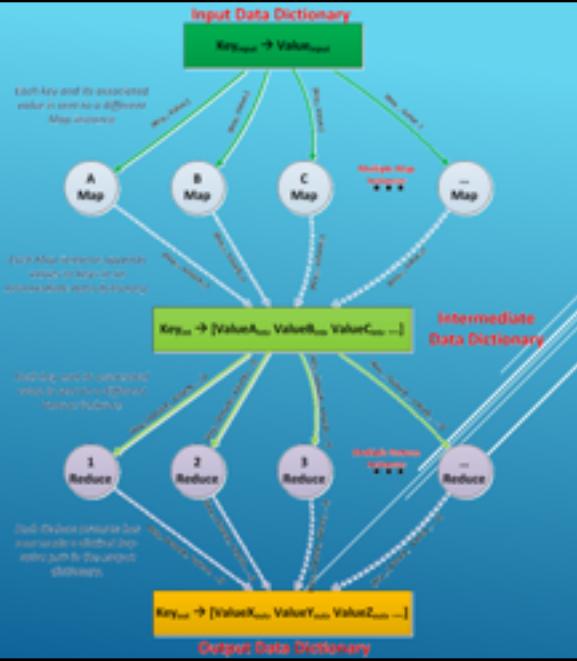
- ▶ map $f(K, V) = \sum_n((K_n, V_n))$
- ▶ reduce $f(K, \sum_n(V_n)) = \sum_m(V_m)$

Some utility functions:

- ▶ partition $f(K) = \text{reducer}$
- ▶ compare $f(l, r) = \begin{cases} -1, & l < r \\ 0, & l = r \\ 1, & l > r \end{cases}$
- ▶ read $f(\text{input}) = \sum_n((K_n, V_n))$
- ▶ write $f(K, \sum_n(V_n)) = \text{output}$

Where Sigma means collection here

MAP/REDUCE



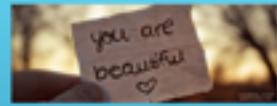
Attention: Where sigma doesn't mean sigma as it is in mathematic. It means collection here.

- ▶ Computing with declarative languages, usually SQL
- ▶ Used to be expensive proprietary software suites only
- ▶ Open Source player is emerging, e.g. Impala.

MASSIVELY PARALLEL PROCESSING



- ▶ Everything sounds beautiful, looks terrific, smells like teen spirit
- ▶ However, tastes bitter than bitter
- ▶ Deemed life long journey



A JOURNEY FOREVER

CONFIDENTIAL

Distributed architectures allow us to build
highly available, fault-tolerant systems.

RECAP

We can't live in this **fantasy land** where
everything works **perfectly** all of the time.

RECAP

Shit happens – network partitions,
hardware failure, GC pauses, latency,
dropped packets

RECAP

Build **resilient** systems.

RECAP

Design for **failure**.

RECAP



Consider the trade-off between
consistency and **availability**.

RECAP

Partition tolerance is not an option, it's required.

RECAP

Use **weak consistency when possible**,
strong when necessary.

RECAP

Sharing data at scale is hard, let's go shopping.

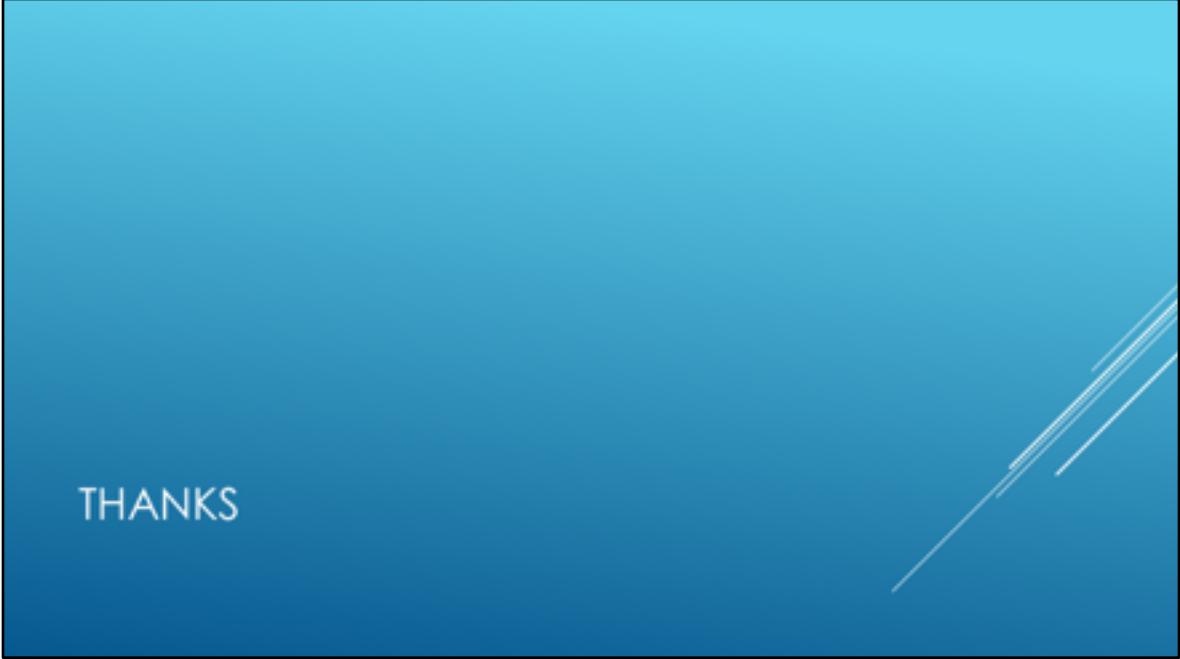
RECAP

State is hell.

RECAP

- ▶ [Reliable Distributed Systems: Technologies, Web Services and Applications](#)
- ▶ [Paxos Made Simple](#)
- ▶ [Dynamo: Amazon's Highly available Key-value store.](#)
- ▶ [MapReduce: Simplified Data Processing on Large Clusters](#)
- ▶ [BigTable: A Distributed Storage System for Structured Data](#)
- ▶ Many more...
- ▶ [There is one more thing](#)

FURTHER READINGS



THANKS