

# Distributed Consensus

## an introduction

Tony Astolfi, June 2020









*How do we build reliable systems from unreliable components?*





*How do we build reliable systems from unreliable components?*

**Partial Failure**





*How do we build single coherent state from many divergent states?*





*How do we build single coherent state from many divergent states?*

**Asynchrony**



# Goals

- Use Case for Consensus
- Problem Definition and (De-)Composition
- Trade-off Analysis
- Common Protocol Mechanics
- Rules of Thumb & Advice
- Resources for Further Learning

# Consensus Protocols

## Crucial Building Block for Distributed Systems

- Broadly Used
  - Hadoop Ecosystem
    - ZooKeeper
    - Apache Cassandra, Kudu
  - Kubernetes
    - etcd
  - Cloud
    - Google Spanner
    - Amazon Elastic File System (EFS)



# Consensus Protocols

## Many Riffs on a Common Theme

- Viewstamped Replication [1988, 2012]
- Paxos [1990/1998, 2001, 2007]
- ZAB (ZooKeeper Atomic Broadcast) [2007, 2011]
- RAFT [2014]
- NO-Paxos [2016]
- Parallel RAFT [2018]
- Generalized Consensus [2019]



# Out of Scope

(caveat...)

- Weak consistency for greater:
  - scale/availability/throughput/etc. (Redis, BigTable)
- Big Data, sharding/slicing: data/problem too big for single machine
  - Consensus is often *part* of the solution (Cassandra, Spanner)
- “Byzantine” or untrusted consensus
  - Blockchains, cryptocurrency, etc. (Bitcoin, Ethereum)



**Why Consensus?**



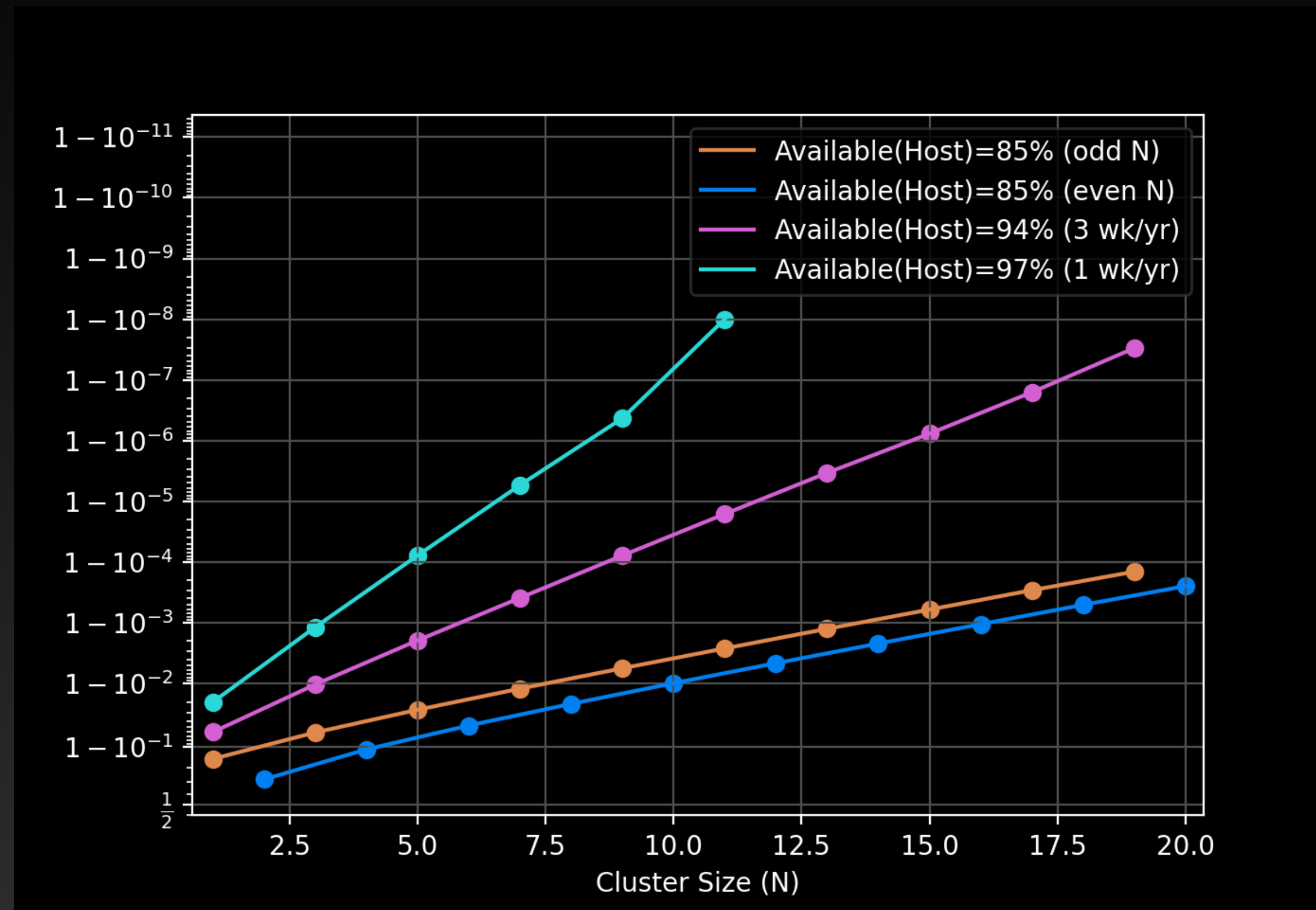
# Strong Consistency & Fault Tolerance

Preserve: Single-Node Simplicity

Add: Reliability ( $N = 2F + 1$ )



# Reliability Amplification



- Exponential improvement as a function of cluster size
- When host downtime is 1wk/yr & N=9, cluster is down less than 14s/yr!



# Problem Definition

What is the consensus problem?



# Problem Definition

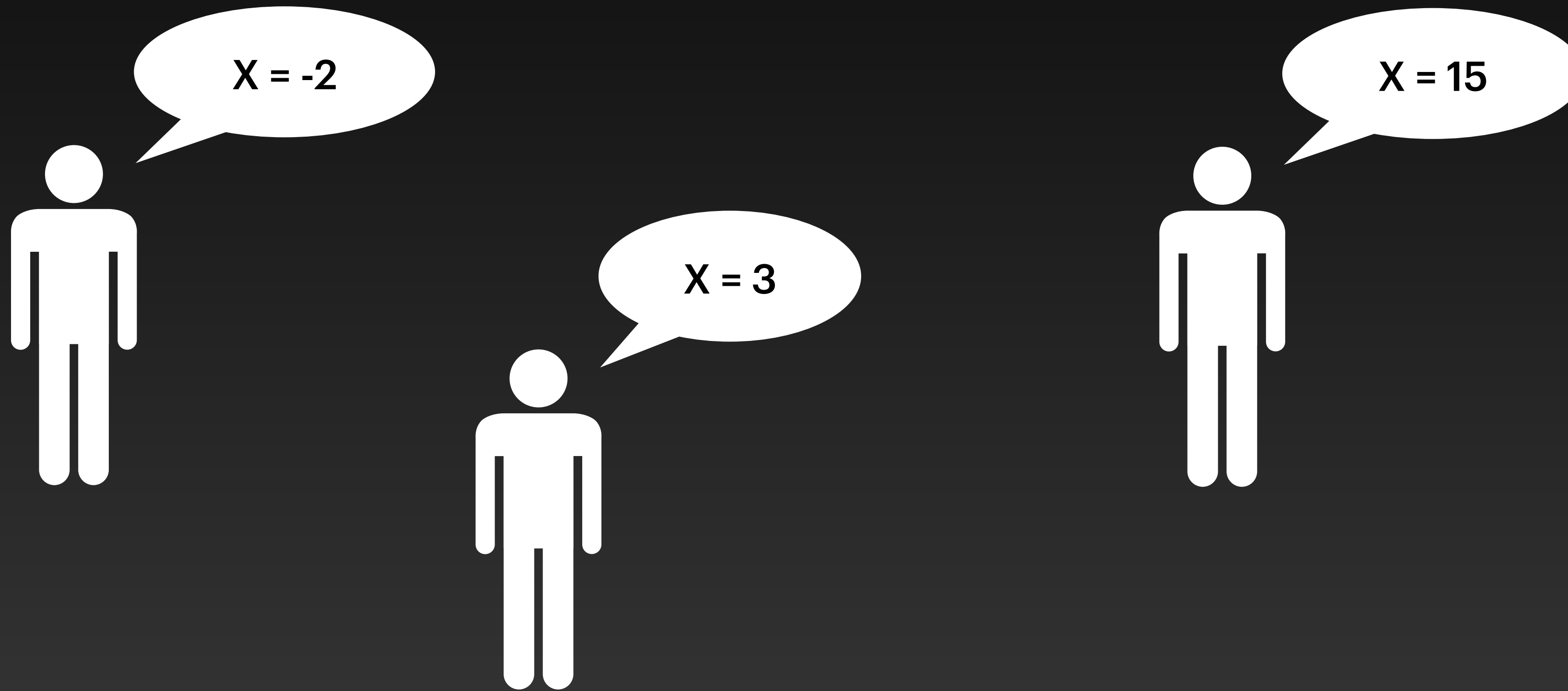
What is the consensus problem?

$$X = ?$$



# Problem Definition

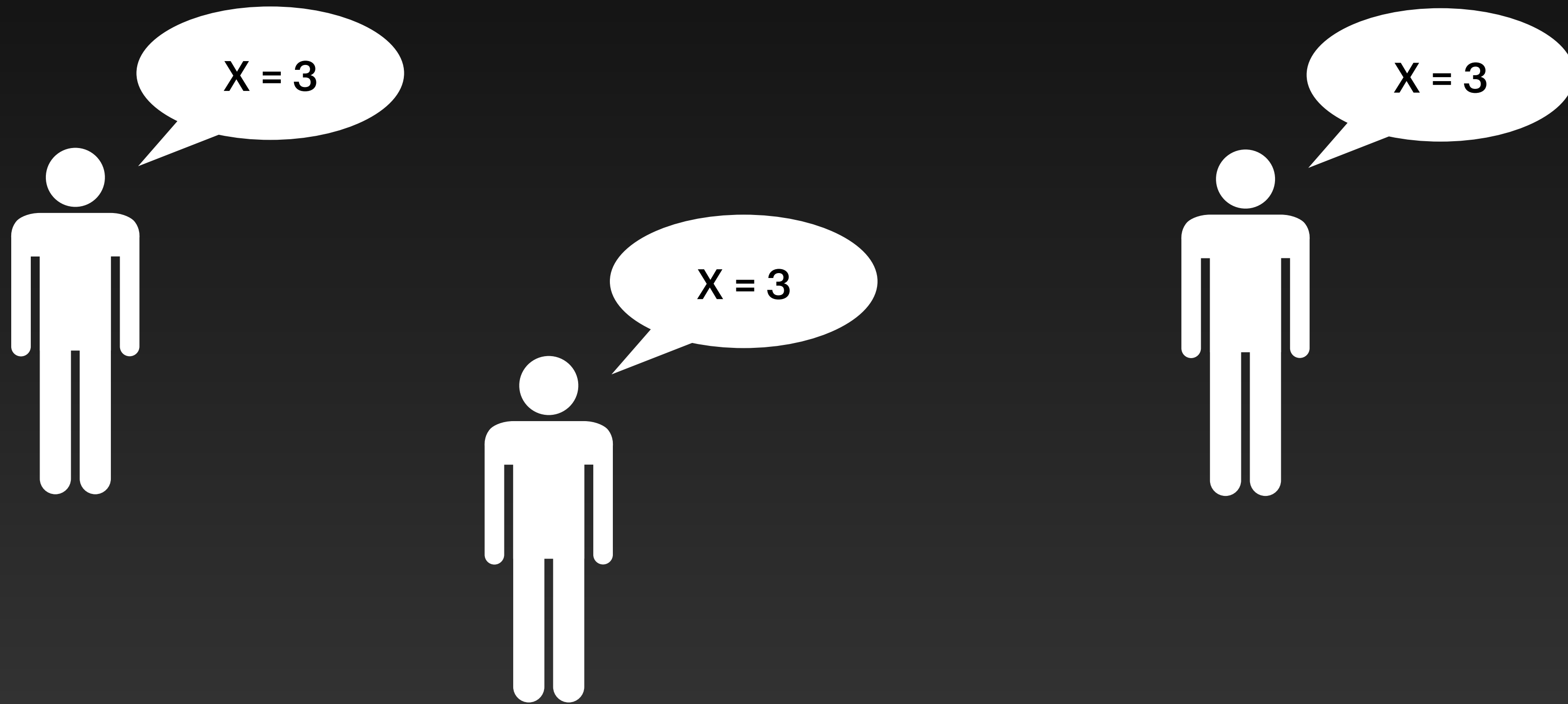
What is the consensus problem?





# Problem Definition

What is the consensus problem?





# Requirements

- **Non-Triviality:** Any decided value must be proposed as input
- **Consistency:** The decided value is agreed upon and immutable
- **Liveness:** Some decision is eventually made



How do we go from “the value of  $X$  as a service” to something useful?



# Replicated State Machine Model



# State Machine Design Principles



# State Machine Design Principles

*“Design graphs, not graph algorithms” - Skiena*



# State Machine Design Principles

*“Design state machines, not consensus algorithms”*

- MUST be deterministic!
- Trim the log!
  - Snapshot + deltas
  - Write snapshots incrementally or in the background
  - Log-structured or append-only data structures
- Write-ahead-log (WAL) techniques
  - Group commands by declaring *intent*
  - Commit at the end



On a Good Day...



On a Bad Day...



# This is complicated!

## Why Study Consensus?

- Know what the design-tradeoffs might be so you can pick the right dependency to take. (What “lunches” aren’t free?)
- Know how to configure the system (avoid even N, cluster sizing)
- Know how to build proper monitoring (failure margin, leader elections)
- *Build vs Buy should be a choice!*



# Is it a good deal?

## Cost/Benefit Trade-offs

Cost	Benefit
Latency: 3-4 delays	Better Latency Distribution
Single-Node Bottleneck	Simplicity for Apps
Operational Complexity: Config, Leader Elections	No Planned Outages for Upgrades/Maintenance
Resources: N x Single Host	(very) High Availability



# More Resources

- Integrating into larger systems
  - Google Chubby (*inspiration for “Paxos-as-a-Service:” ZooKeeper, etcd, ...*)
  - Google Spanner (*consensus within database shards, atomic clocks for transactions*)
  - Eris (*NO Paxos successor; another approach to cross-cluster transactions*)
  - Tapir (*high level: consistent transactions, low level: inconsistent log replication*)
- Alternative/Complimentary
  - Google File System (GFS) (*small, consistent metadata + big inconsistent data*)
  - Amazon Dynamo (*weak consistency model, consistent hashing & gossip*)
  - Tango/Corfu (*Shared-Log-as-a-Service; replicated state machine model w/o consensus*)

**Thank you!**  
(Questions, please!)



# Levels of Build/Buy

- Level 0: No Consensus (high-availability or consistency: pick one)
- Level 1: Depend on a System that depends on Consensus
- Level 2: Depend on a Consensus-as-a-Service system (ZK, etcd, ...)
- Level 3: Depend on an existing implementation (design state machines...)
- Level 4: Implement an existing protocol (RAFT)
- Level 5: Invent a new protocol (dangerous & hard, but worth it at scale)

# Live Reconfiguration

Change cluster membership without interrupting service

- Key Idea: Make membership part of the state machine
  - Protocol must be aware
  - Membership commands require quorum in new and old configs
- Use Cases:
  - Scale up/down
  - Replace unhealthy machine
  - Re-balance load/resources



**END OF DECK**