
KubeCon
CloudNativeCon
Europe 2023
TiKV


# On the Hunt for Etcd Data Inconsistencies

*Marek Siarkowicz, Google*

# Agenda



01 What are data inconsistencies?

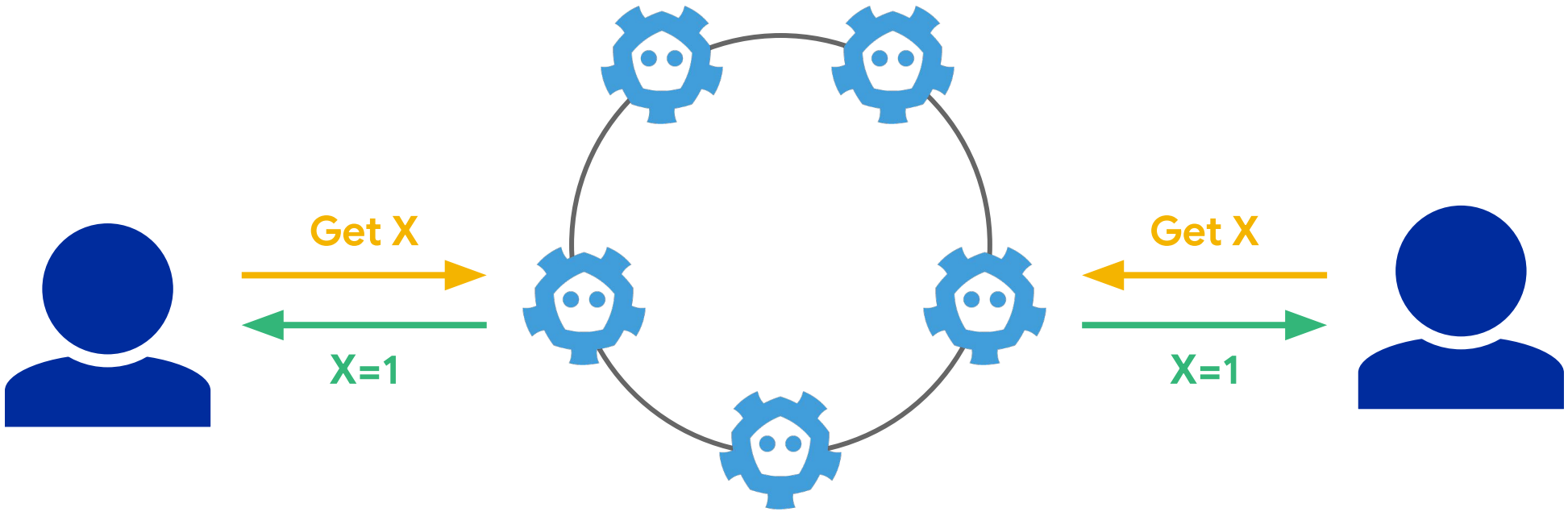
02 How to hunt data inconsistencies?

03 Etcd robustness tests

04 Let’s hunt!

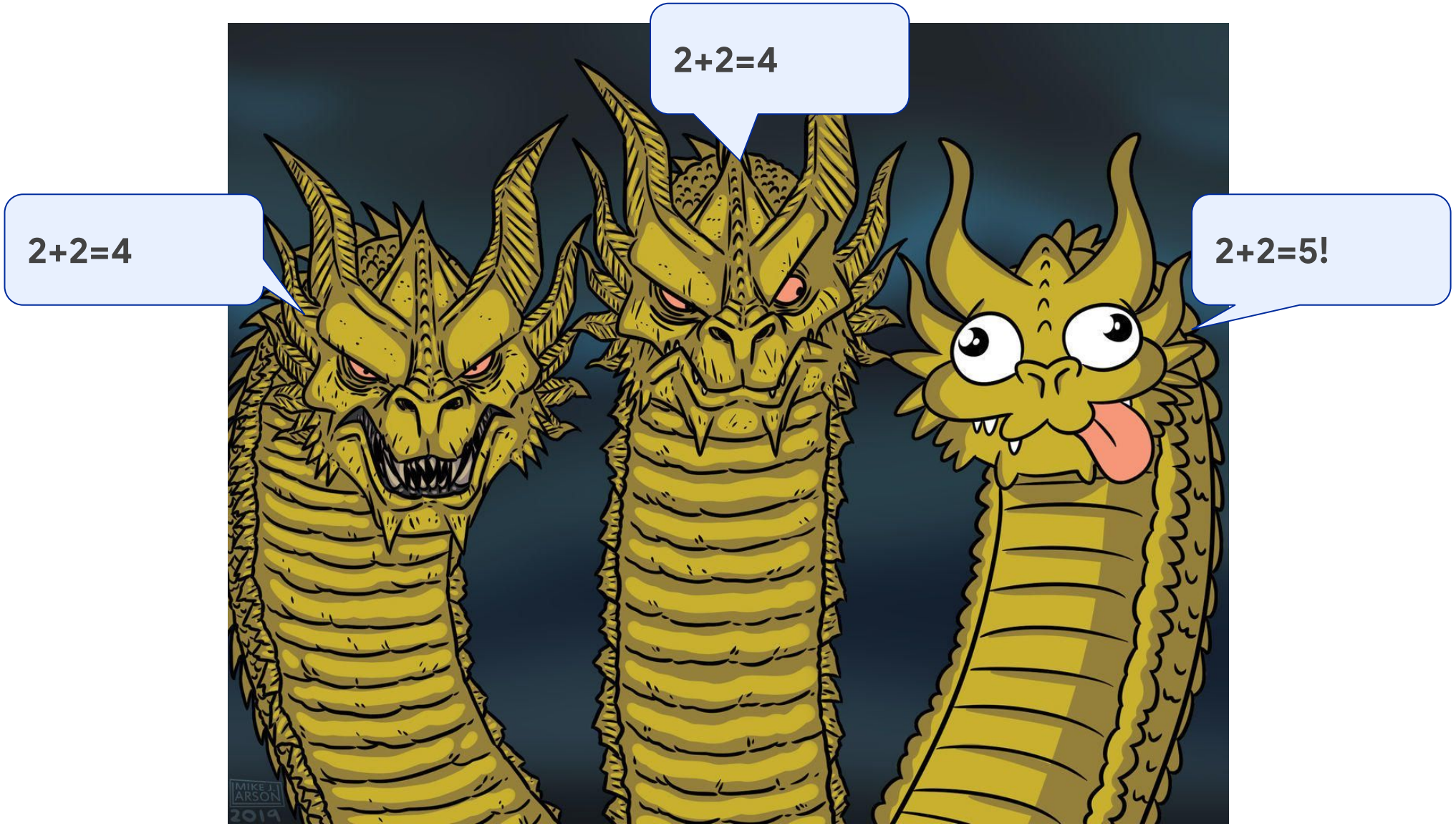
# Distributed consensus





# Data inconsistency





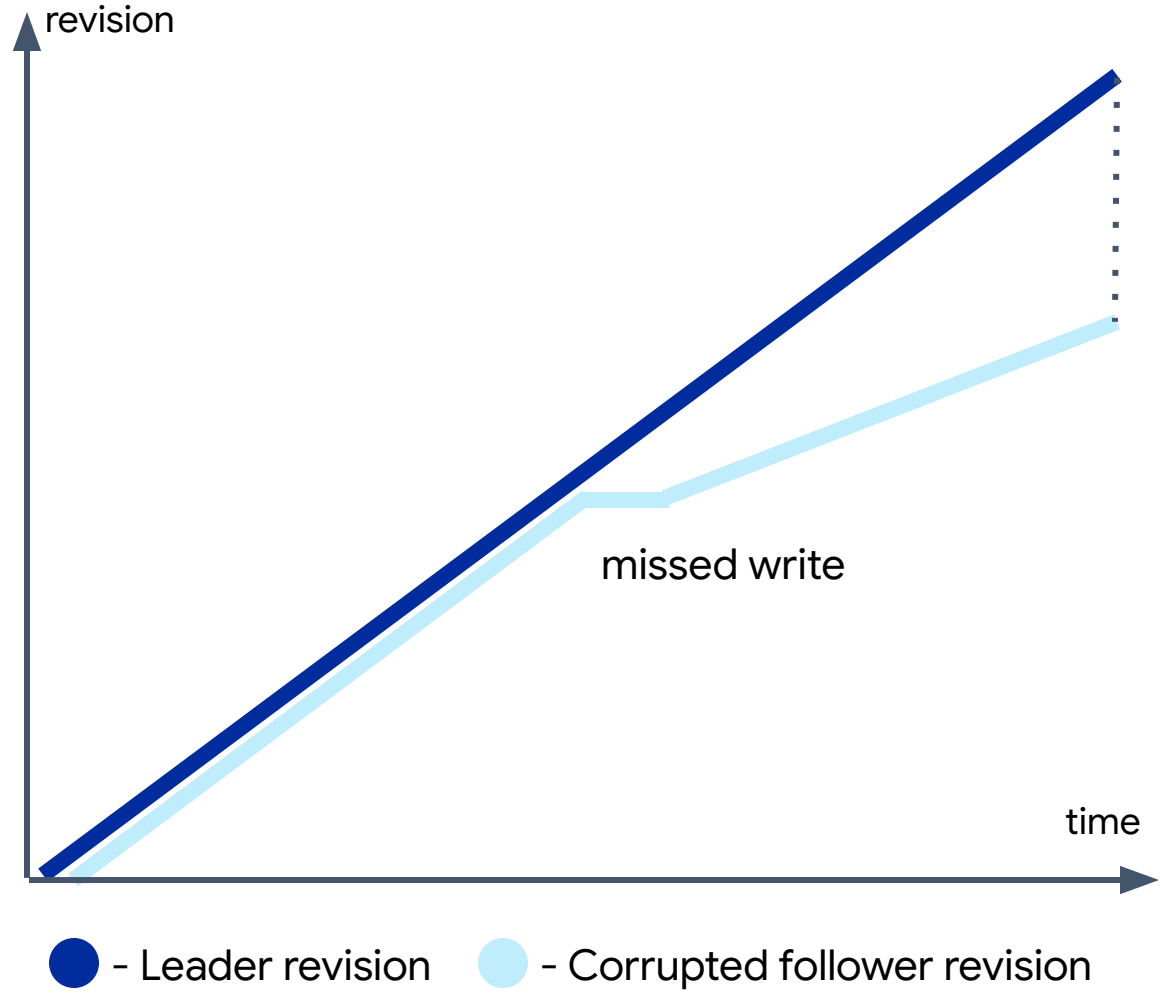
# Kubernetes Production Example



**Issues observed:**

- Nodes status flapping between Read and NotReady
- Random apiserver request failures and timeouts
- Webhooks timeout
- Requests fail to authenticate
- Kube-system addons are crashing
- One out of three apiservers returned stale data

**Root Cause:** One etcd member missed 1 write



# State of v3.5.0



Latest minor etcd release came after **3 years of development.** It resulted in release with multiple data inconsistencies and correctness issues:

- data inconsistency on crash
- loss of durability on crash

An multiltiple unconfirmed reports:

- Data inconsistency
- Stale reads
- Split brain
- Lost update

**Etcd doesn't have a tests capable to detect this class of issues**

## v3.5 data inconsistency postmortem

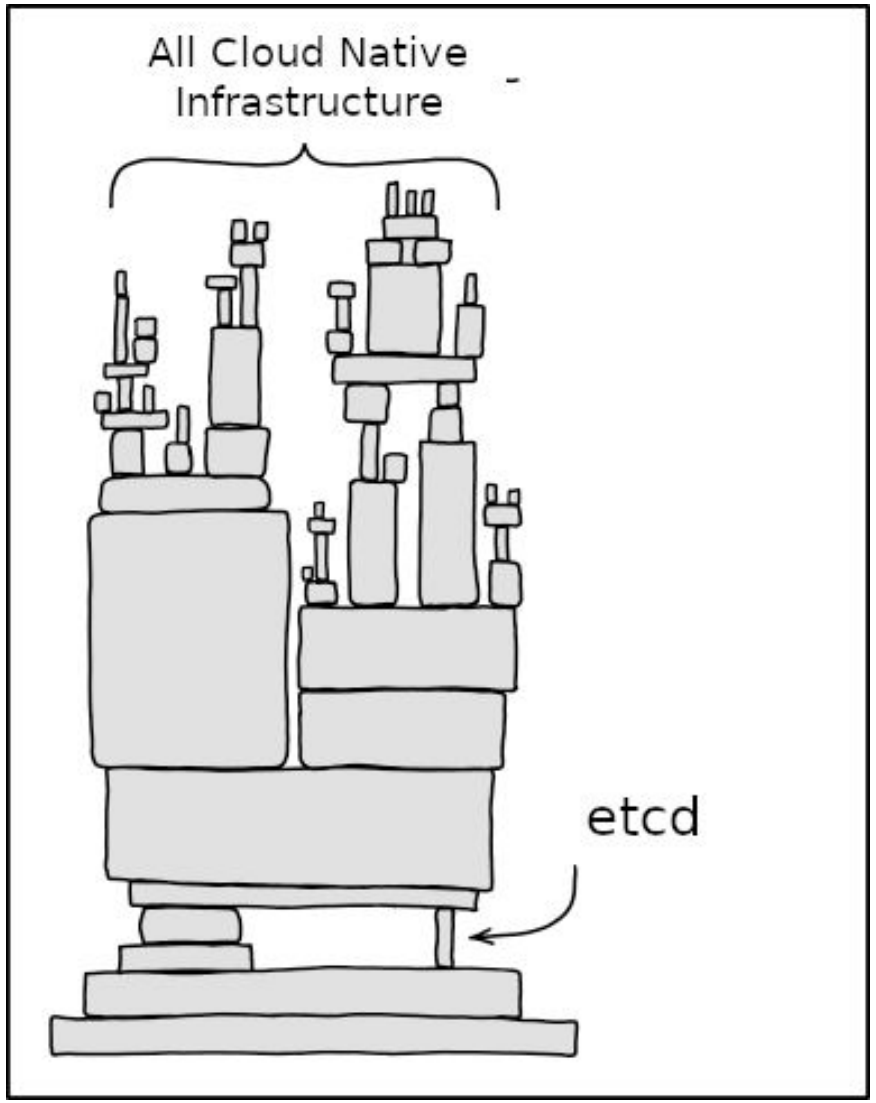
| | |
|---|---|
| Authors | serathius@ |
| Date | 2022-04-20 |
| Status | published |

### Summary

| | |
|---|---|
| Summary | Code refactor in v3.5.0 resulted in consistent index not being saved atomically. Independent crash could lead to committed transactions are not reflected on all the members. |
| Impact | No user reported problems in production as triggering the issue required frequent crashes, however issue was critical enough to motivate a public statement. Main impact comes from loosing user trust into etcd reliability. |

# Etcd reliability is important






Source: https://xkcd.com/2347/

# How to hunt data inconsistencies?

KubeCon CloudNativeCon Europe 2023

## etcd functional tests

Framework built by the previous etcd maintainers to address the same problem. Put traffic on etcd cluster while injecting failures. After test validate if members are inconsistent.

Problems:

- Very rigid and over designed
- Non-strict and flaky
- Doesn't check correctness, just consistency

More in #15102 and #14820

## Jepsen

Generic framework build to validate safety of distributed databases, queues, consensus systems. Developed by Kyle Kingsbury (@aphyr) that offers paid analysis of your system. Validated etcd in 2014 and 2020.

Problems:

- Written in Closure
- Non-AWS setups are broken
- Not approachable. Requires domain knowledge
- To heavy to be run in CI
- Hard to reach owner

More in #14890

# Requirements



## Adequate

- Validate generic properties
- Reproduces historical etcd issues

## Accurate

- Strict validation of tested scenarios
- Failures can be easily attributed to etcd or test issue

## Maintainable

- Run in our continuous integration
- Use existing e2e testing framework
- Simplify reproduction based on individual failure

Etcd API guarantees

# Robustness - ability of system to maintain correctness under any condition

Normal operation, upgrade, network failure, disk failure etc

# What does “any condition” mean?



- Expected failure - once per cluster day
  - Network packet loss
  - Clock drift
  - Slow IO
- Maintenance - once per cluster week
  - **Graceful shutdown**
  - Upgrade
  - Downgrade
  - Cluster membership change
  - **Defrag**
- Standard failure - once per cluster year
  - **Forceful shutdown**
  - **Network partition**
  - Storage failure
- Obscure failure - once per 100 cluster years
  - Silent Data Corruption
  - Memory/CPU corruption
  - Network packet manipulation

**Bolded** cases are currently tested for etcd

Presented failure frequencies are an approximation

# What does correctness means?

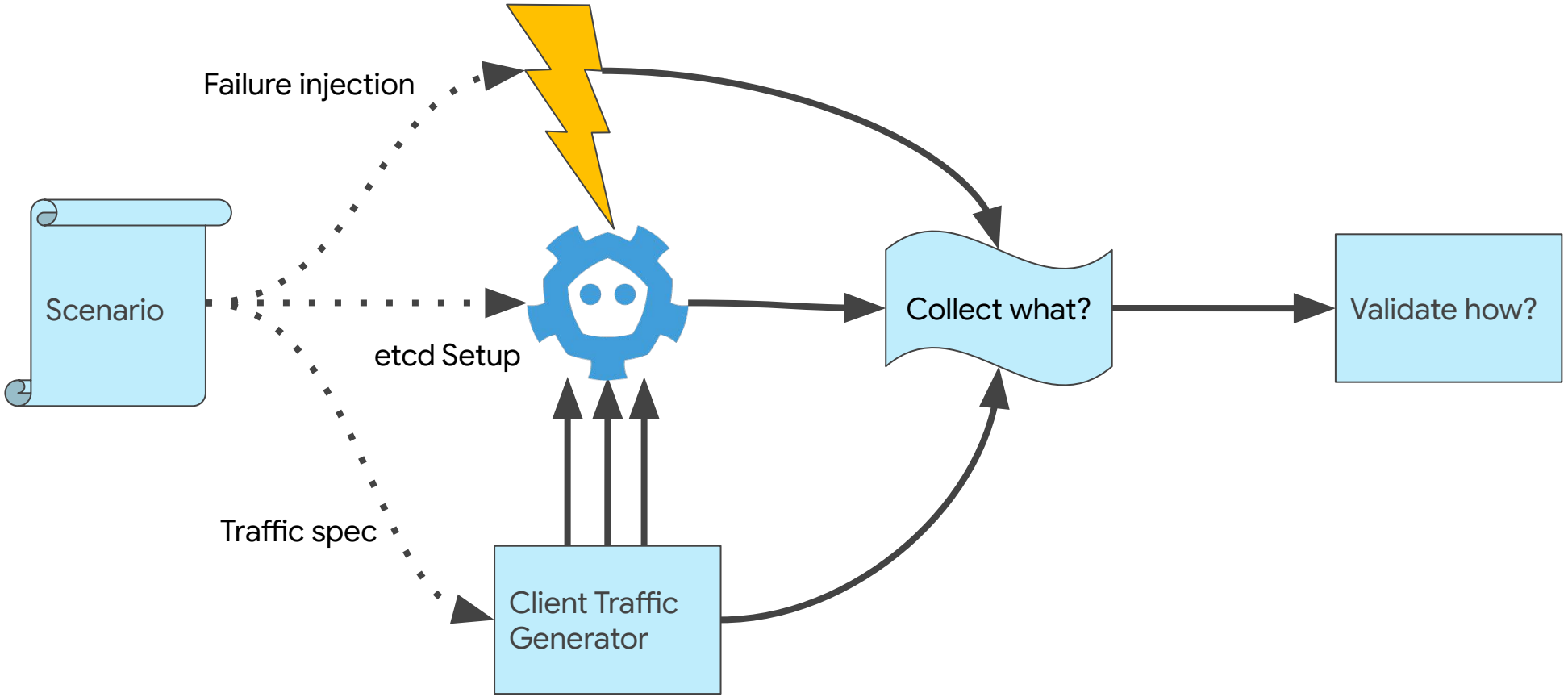


- Etcd upholds key value API guarantees about operations:
  - Atomic
  - Durable
  - Linearizable

- Etcd upholds watch guarantees about event streams:
  - Ordered
  - Reliable
  - Atomic

# How to validate correctness?

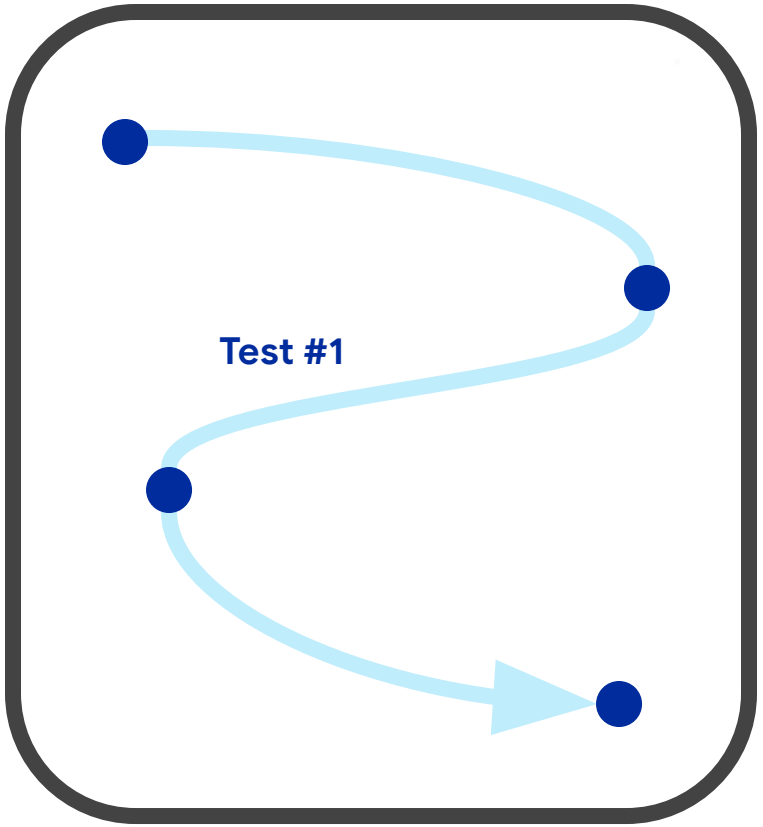




# Scripted vs exploratory testing

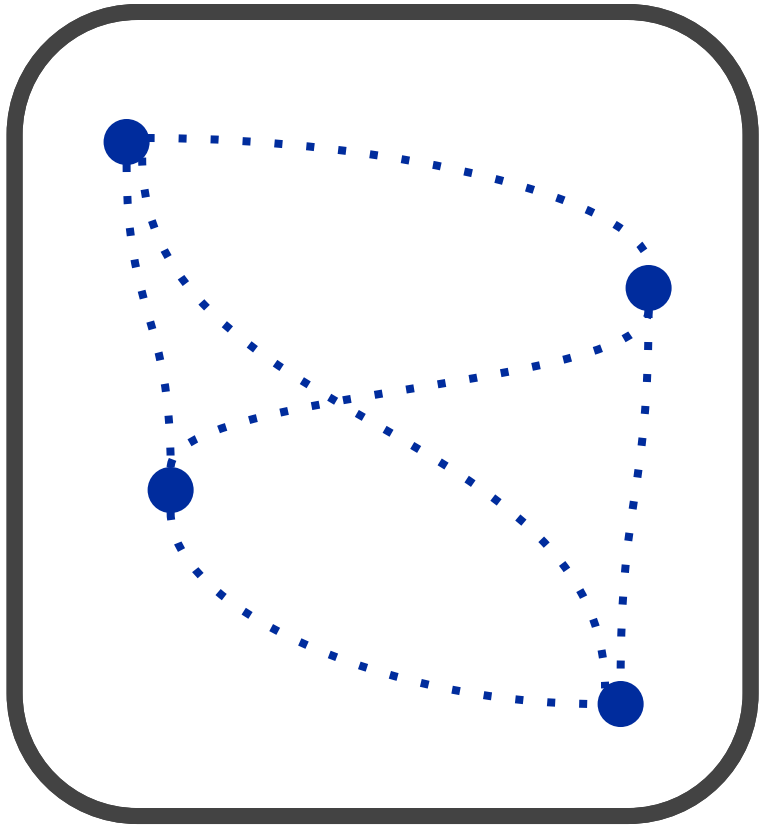


Predefined scenario




Validation at the end by comparing result vs expect

Defining possible paths and picking random



Validation ??

# Model-based testing



Uses a **model** to find discrepancies in real system behavior. A model

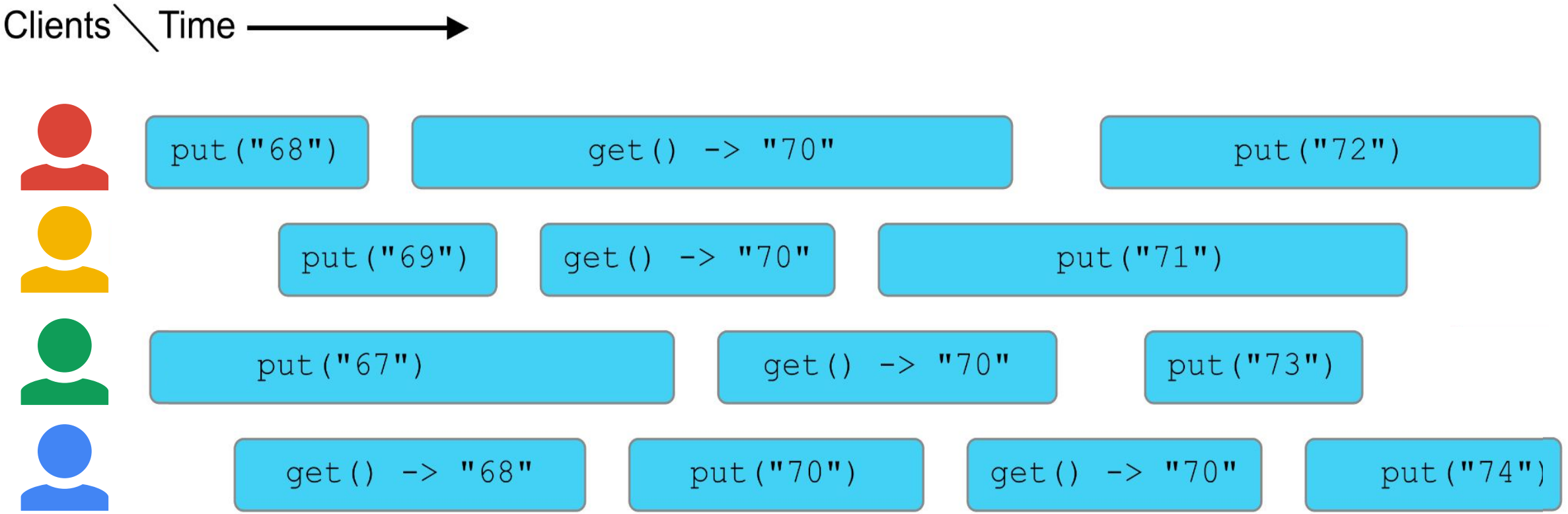
- Represents the desired behavior of the system
- Is a simplified implementation of a real system
- Implements subset of API that we want to test

Model-based tests take an **operation history** executed by the system and replay it on the model. Test passes if their responses match.

As a key-value store etcd model can be just a **hash map with a counter** to simulate revision.

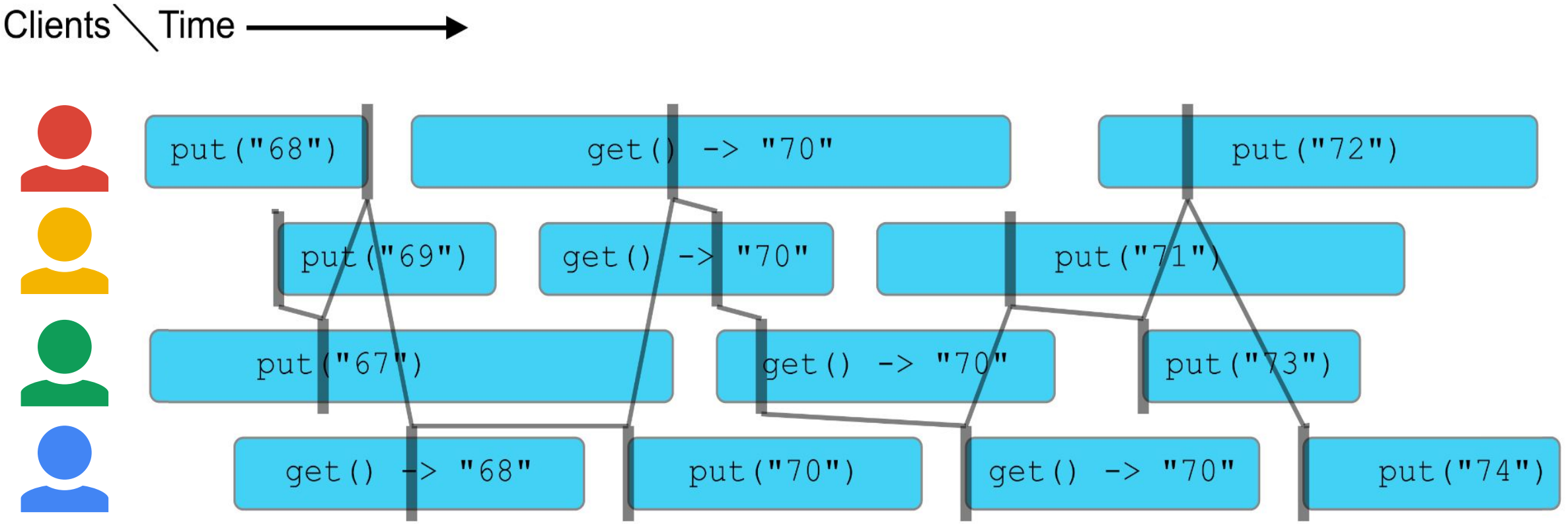
# Operation history



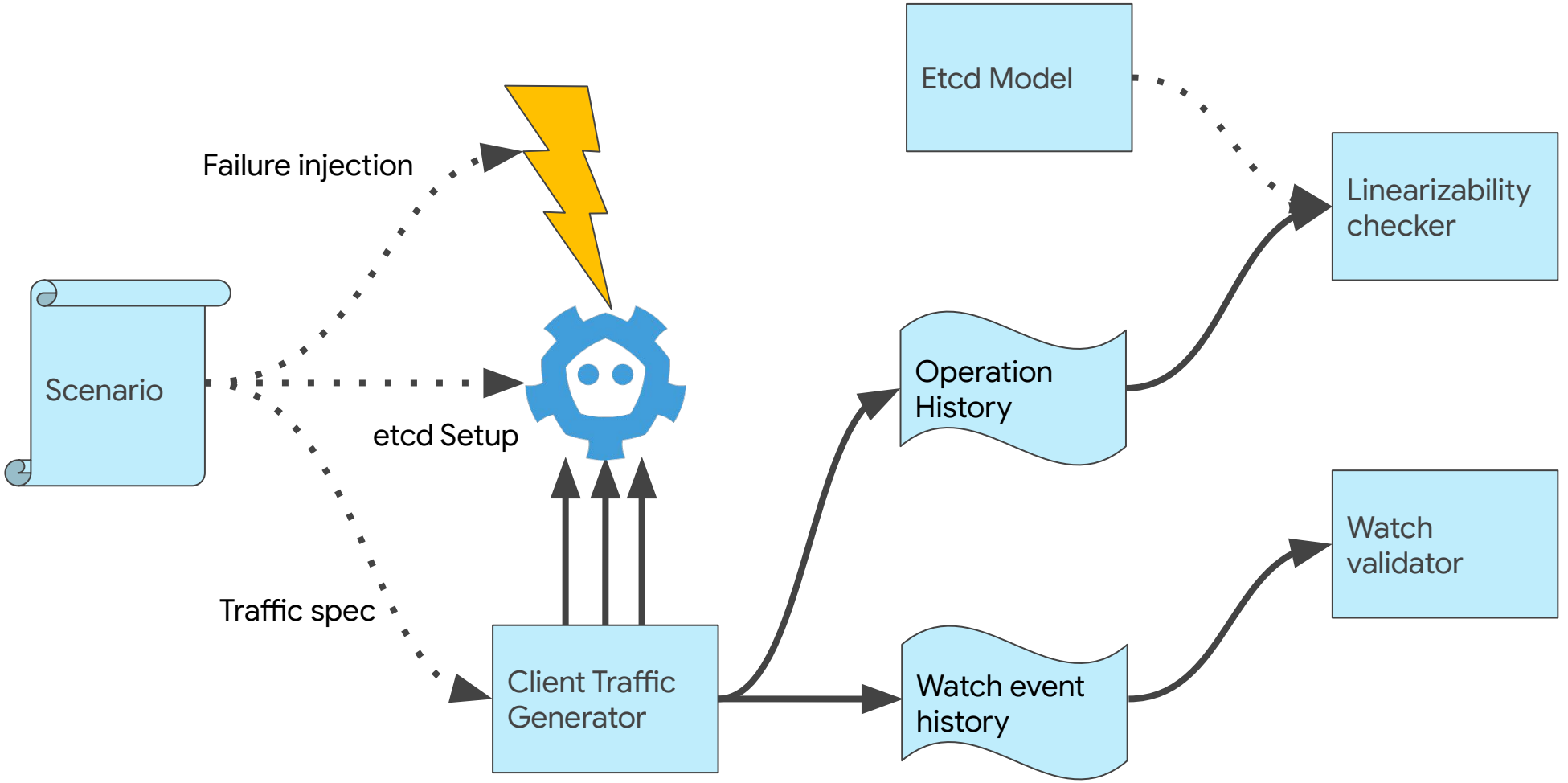


# Checking linearizability





# Etcd robustness tests



# Validating Kubernetes Contract

KubeCon CloudNativeCon Europe 2023

- Kubernetes has many unwritten has assumptions about etcd behaviors that are required by it.

- Many little known properties, like recently added:
  - **Renewable** - Progress notification guarantees that previous events have been already delivered.

This motivated writing down [The Implicit Kubernetes-Etcd Contract]

## The Implicit Kubernetes-ETCD Contract

authors: Marek Siarkowicz , Han Kang
created: Mar 29, 2023

### Background

Kubernetes natively uses etcd as its underlying store. There are a number of assumptions about the underlying store that Kubernetes depends on. Changes to these underlying features may cause Kubernetes to behave incorrectly, or even not work at all.

This document does **not** intend to suggest that Kubernetes will not make use of other etcd APIs in the future, nor does this document intend to posit an opinion about that. Goal of this document is to clarify the existing implicit contract between etcd and Kubernetes so it can be used to improve testing and documentation of both projects.

### Storage Requirements

Kubernetes expects that the storage backend is an MVCC key-value store with support for compaction and the ability to watch for changes to keys.

Due to apiserver support for partitioning etcd clusters by resource type, kubernetes underlying store might be sharded by resource type. Each ***resource type*** must guarantee:

- **Strict Serializability -** Operations (read, write, delete) are atomic and occur in a total order, consistent with real-time order of those operations. Read more about consistency.
- **Resource version** - Each object change needs to have an identifier establishing a total order of operations

# Results



- Revision inconsistency caused by panic during defrag #14685
- Watch response traveling back in time after network partition #15271
- Duplicate event revision on watch stream #15248

# Next steps



- Codify the whole Kubernetes-etcd contract
- Add more diverse failure conditions to robustness tests, like simulating disk failure
- Use same approach to validate bbolt, embedded key-value store used by etcd
- Validate operation history in all etcd tests
- Use etcd model in K8s tests

# Let’s hunt!

# Should you use model-based test?



Pros

- More generic approach to correctness
- Separates validation test phase
- Model is reusable

Cons

- Not needed by most applications
- Model can get quite complicated
- Fragile due to linearizability checking being NP-Complete problem

# New contributors are welcomed!

KubeCon CloudNativeCon Europe 2023

KubeCon CloudNativeCon North America 2021

Please scan the QR Code above
to leave feedback on this session