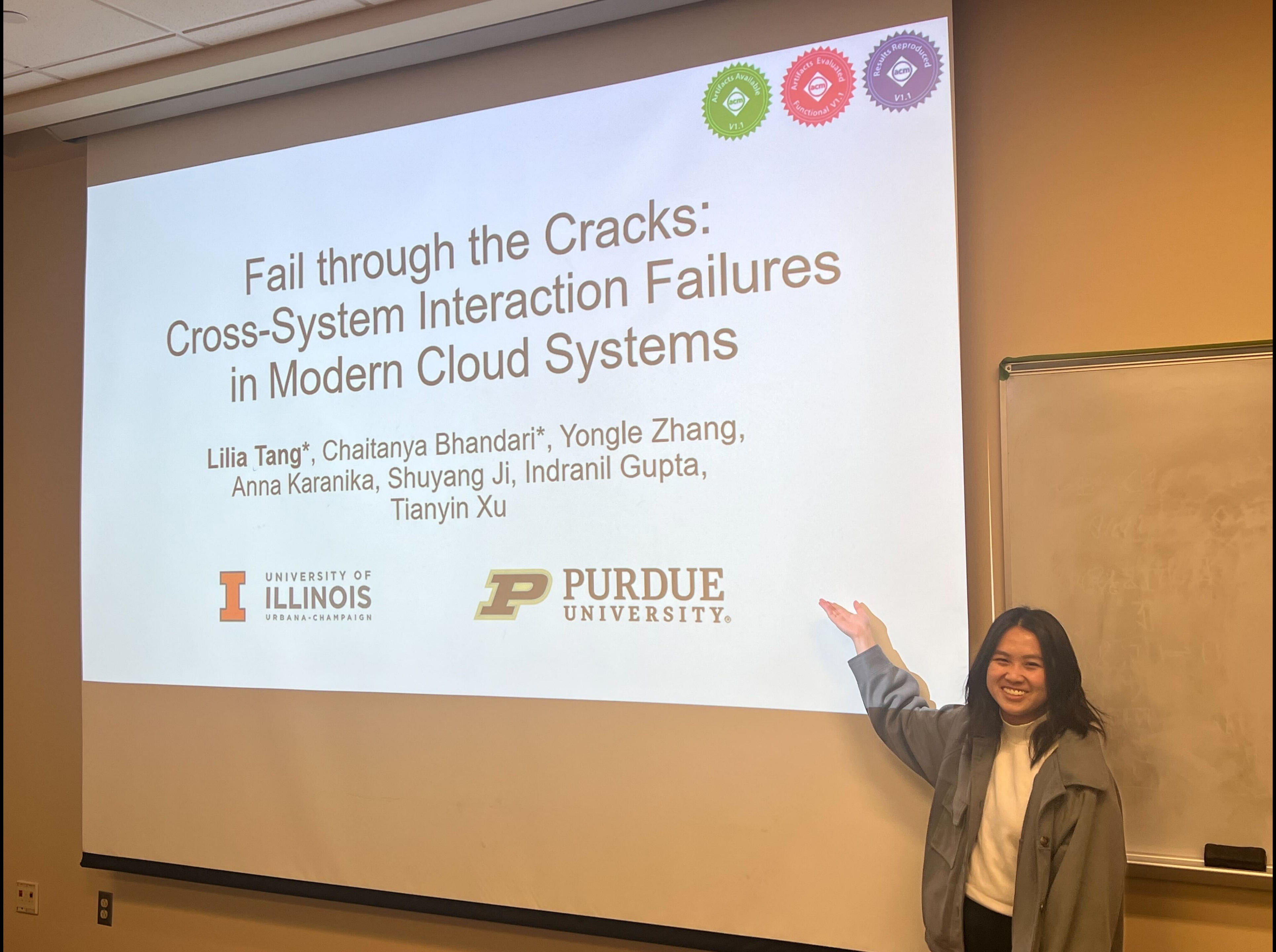
Cross-System Interaction Failures Don't Fail through the Cracks

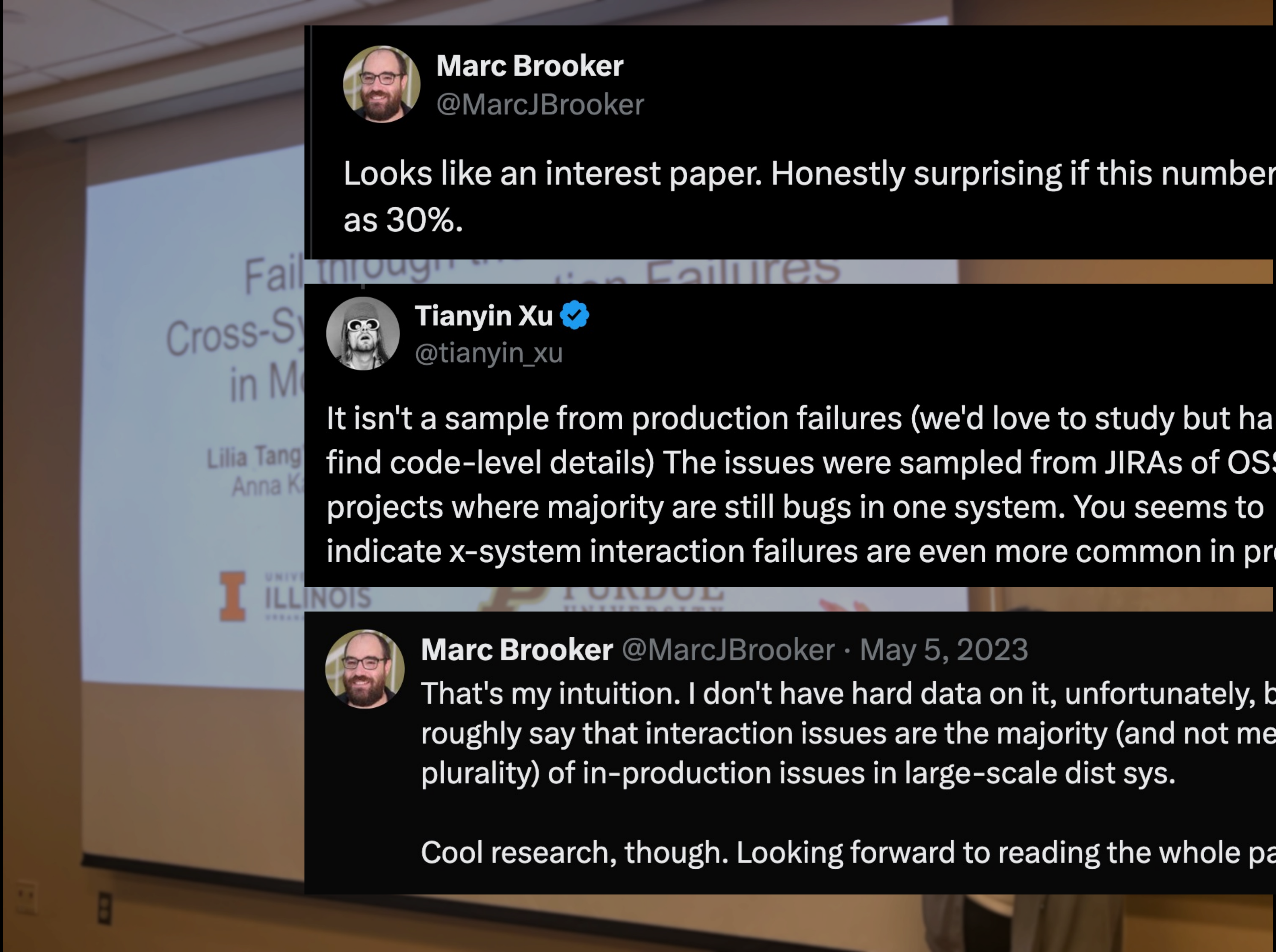
Tianyin Xu
University of Illinois Urbana-Champaign

Xudong Sun
University of Illinois Urbana-Champaign


Artifacts Available
V1.1
Artifacts Evaluated
Functional V1.1
Results Reproduced
V1.1
Fail through the Cracks:
Cross-System Interaction Failures
in Modern Cloud Systems
Lilia Tang*, Chaitanya Bhandari*, Yongle Zhang,
Anna Karanika, Shuyang Ji, Indranil Gupta,
Tianyin Xu
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
PURDUE UNIVERSITY

Indranil Gupta
@indygupta

The Dark Matter of Production Bugs! How bad are Cross-System Interactions? Out of 360 random issues, 120 issues were #CSI Failures!
Lilia Tang @liliatangxy will present paper @ #Eurosys2023! W/
@tianyin_xu @chaitybhandari @anna_karanika @jsy531
Paper: tinyurl.com/CSI-2023



Marc Brooker
@MarcJBrooker

Looks like an interest paper. Honestly surprising if this number is as low as 30%.

Tianyin Xu
@tianyin_xu

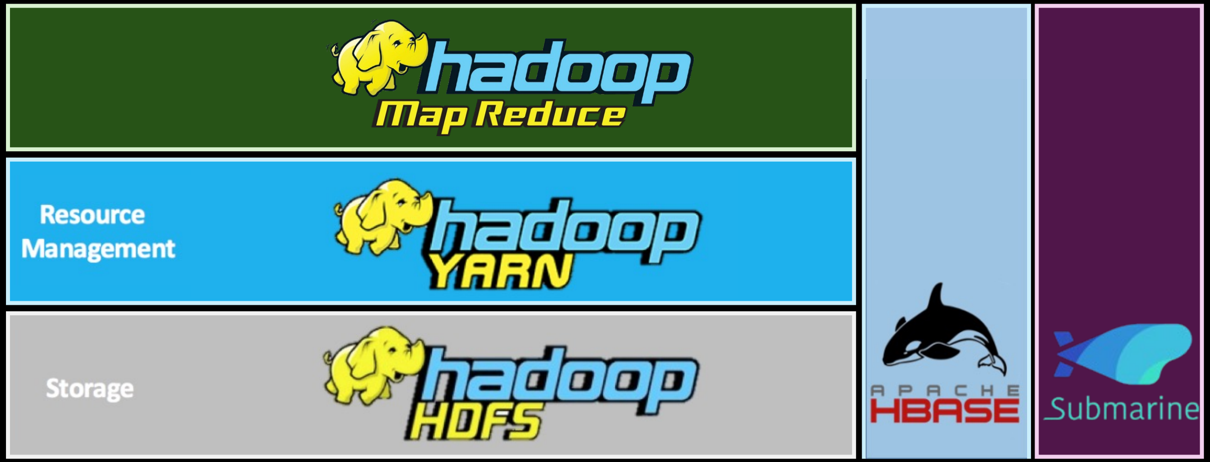
It isn't a sample from production failures (we'd love to study but hard to find code-level details) The issues were sampled from JIRAs of OSS projects where majority are still bugs in one system. You seems to indicate x-system interaction failures are even more common in prod? :0

Marc Brooker @MarcJBrooker · May 5, 2023

That's my intuition. I don't have hard data on it, unfortunately, but I would roughly say that interaction issues are the majority (and not merely plurality) of in-production issues in large-scale dist sys.

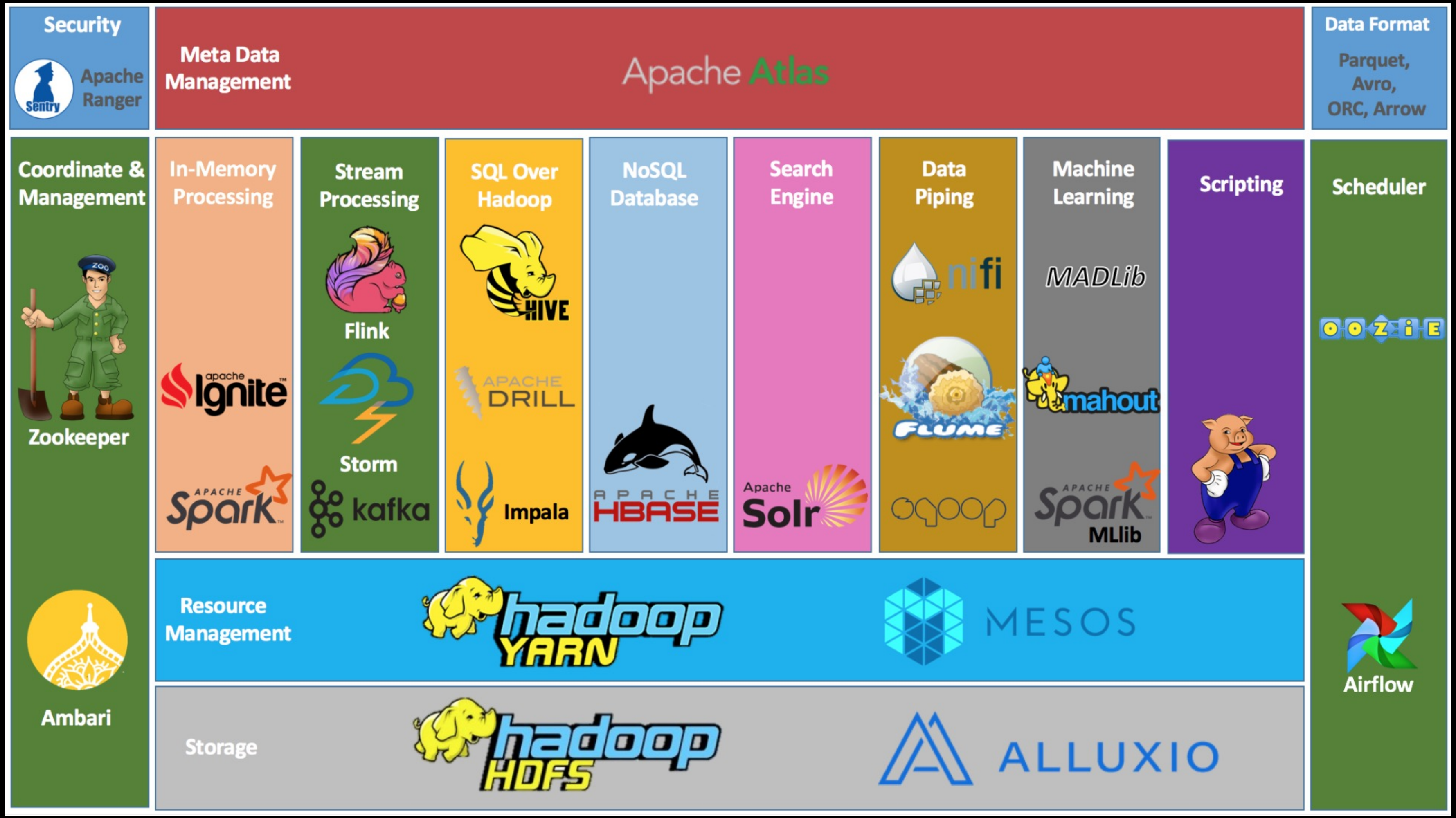
Cool research, though. Looking forward to reading the whole paper.

Our production stack is mostly an orchestration of many (!) interacting systems.

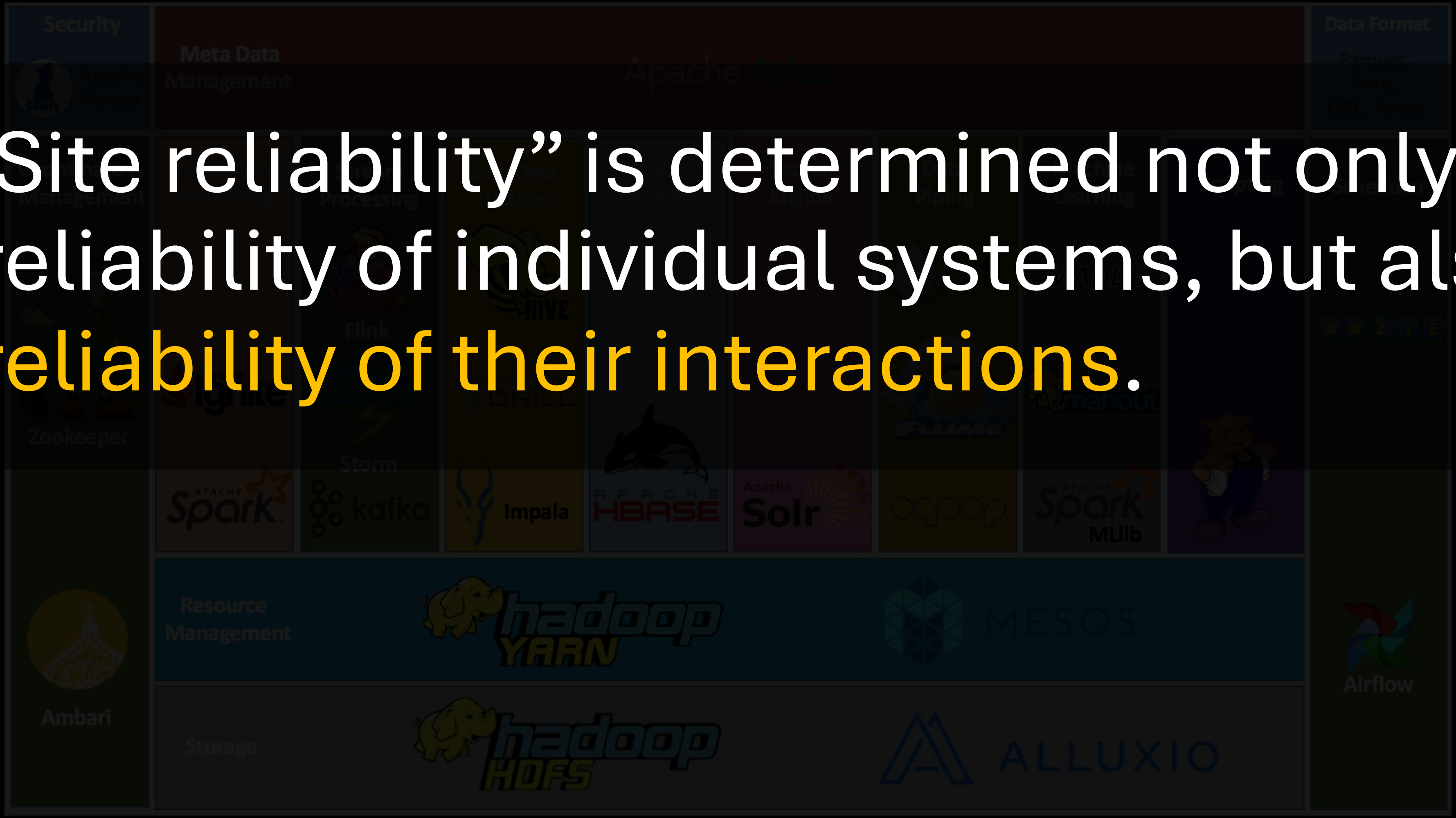


https://hadoop.apache.org/

Our production stack is mostly an orchestration of many (!) interacting systems.



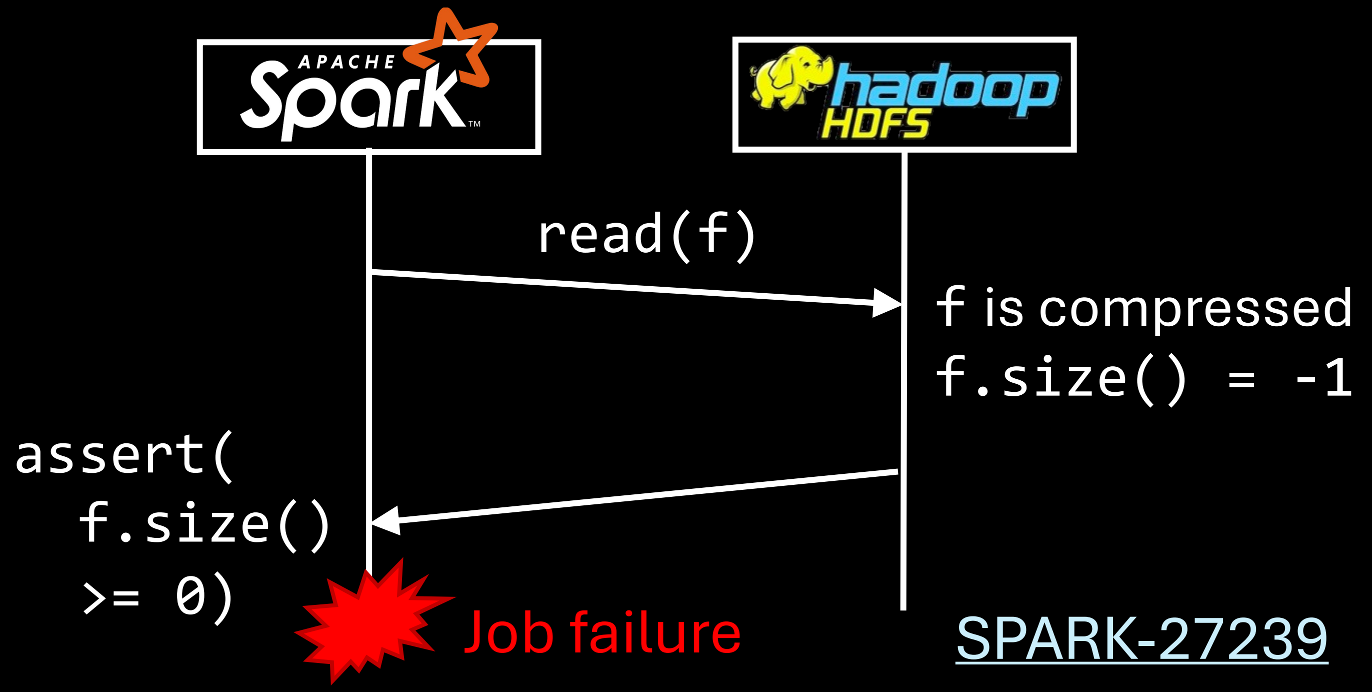
The production stack is mostly an orchestration of many (!) interacting systems.



“Site reliability” is determined not only by the reliability of individual systems, but also by the reliability of their interactions.

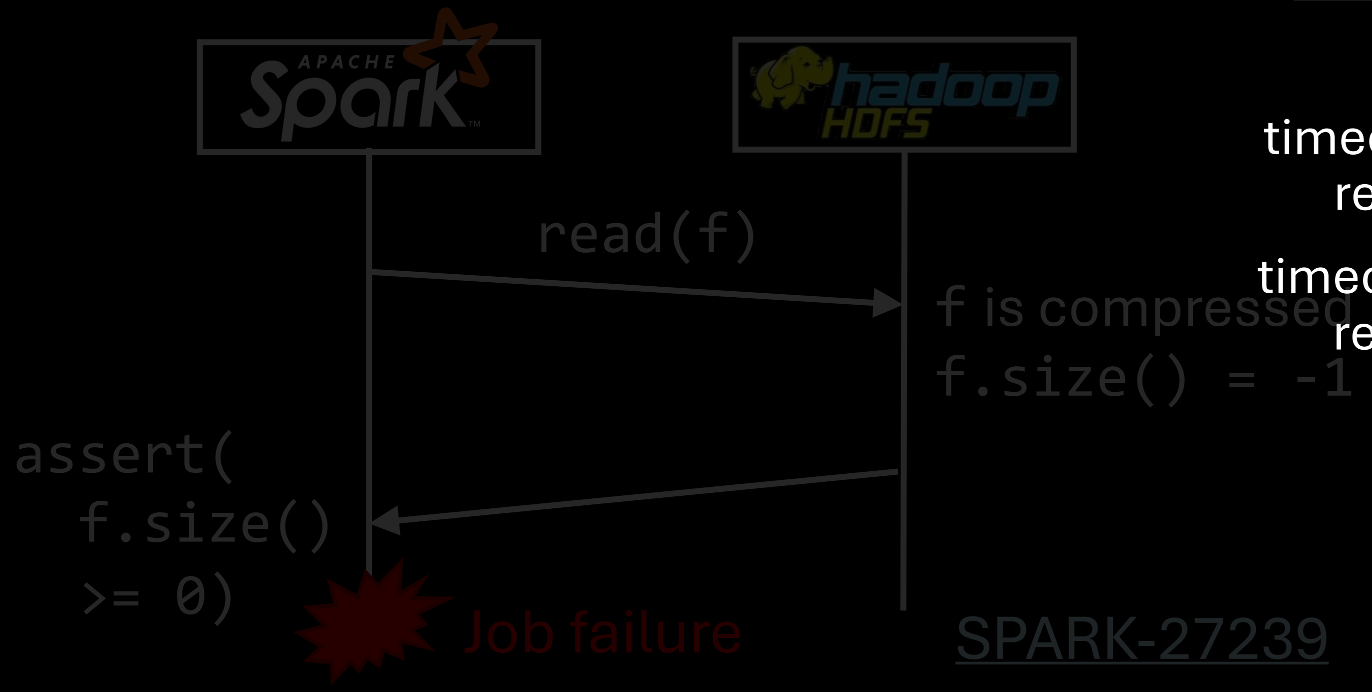
Interaction reliability is hard.

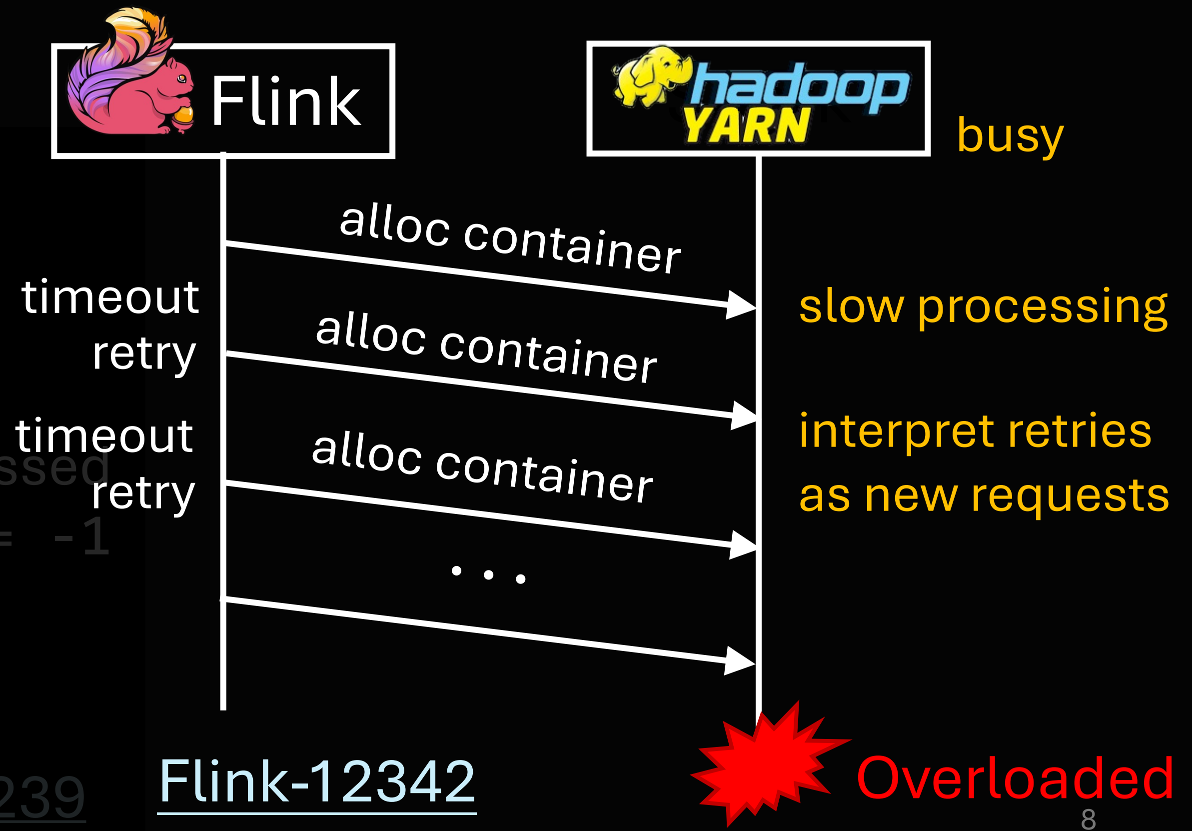
- Few formal description on cross-system interfaces
  - Not even “POSIX”



Interaction reliability is hard.

- Few formal description on cross-system interfaces
  - No “POSIX” any more
  - No spec on error paths



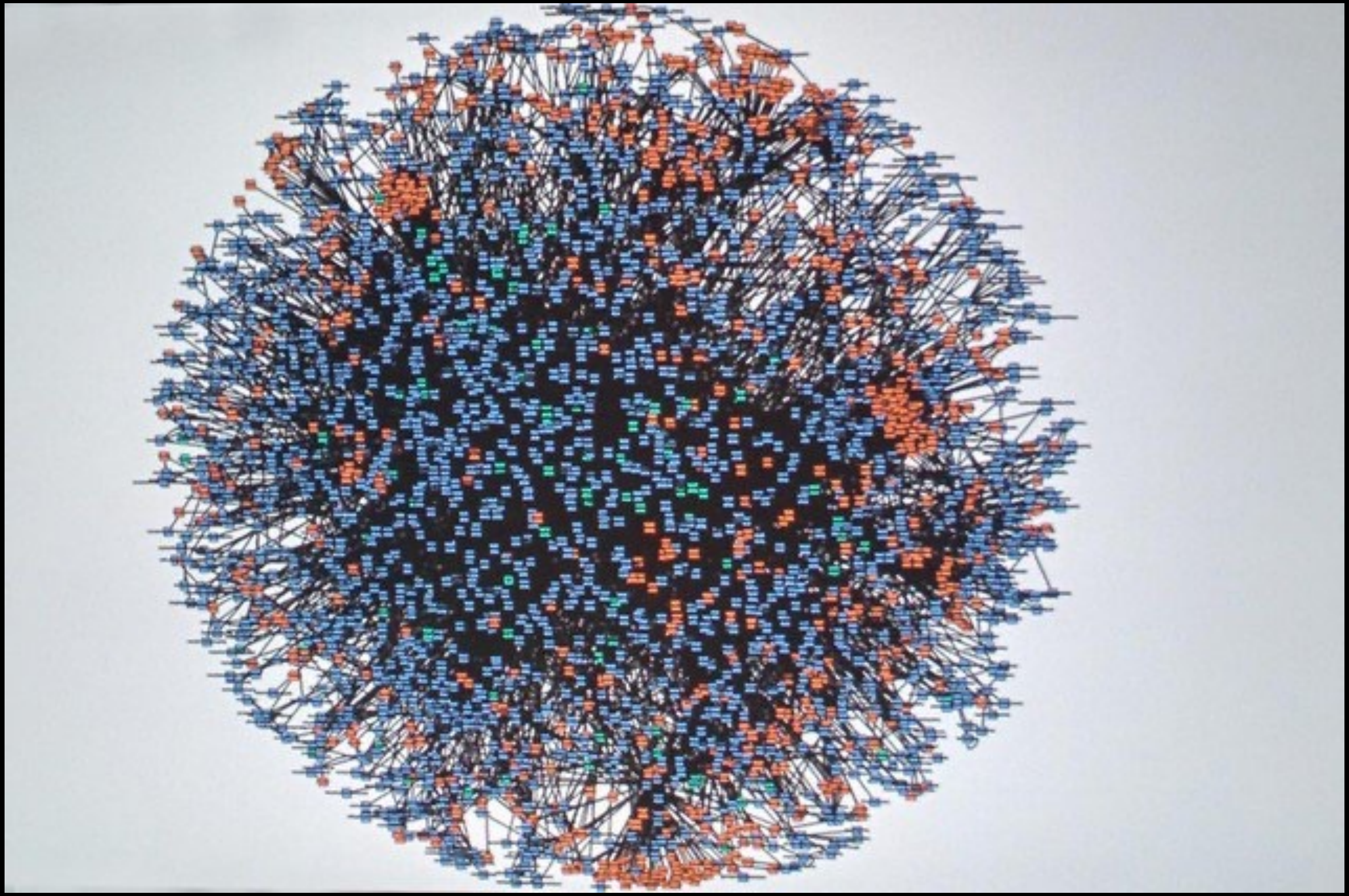


Interaction reliability is hard.

- Few formal description on cross-system interfaces
  - No “POSIX” any more
  - No spec on error paths

- High cost of reasoning about multiple systems *collectively*
  - New tools are needed to cross system/program boundaries
  - Search space grows exponentially

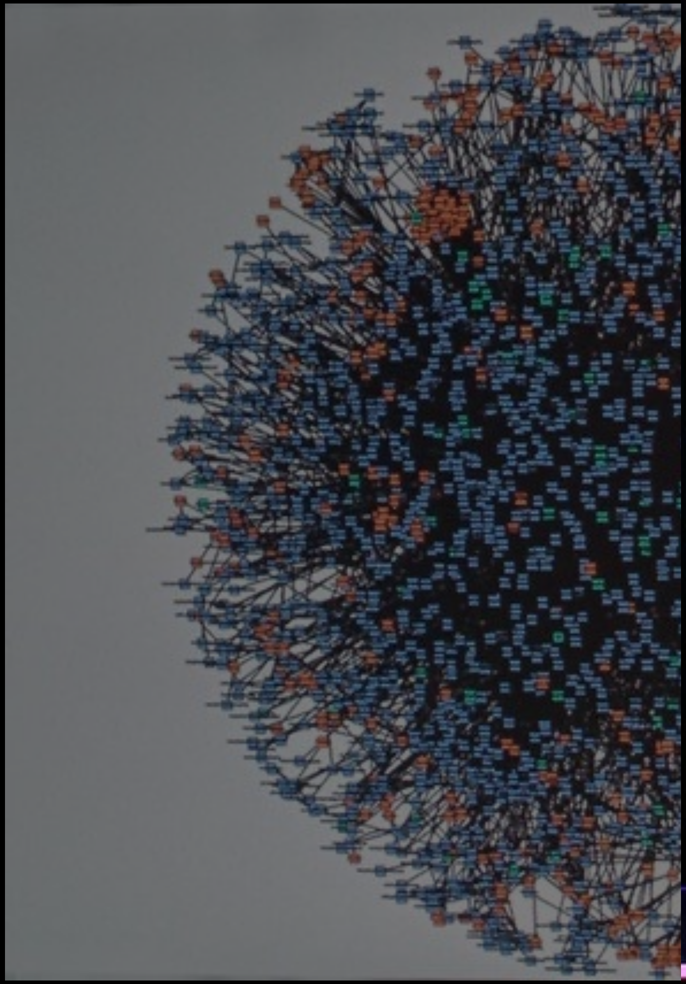
Magnified by emerging computing paradigms



Real-time graph of microservice dependencies at amazon.com in 2008.

- Microservice
- Serverless
- Sky computing
- Hybrid cloud

From the Death Star to the Galaxy



Real-time graph of microservice dependencies at amazon.com in 2008.

AWS Re:Invent 2023.

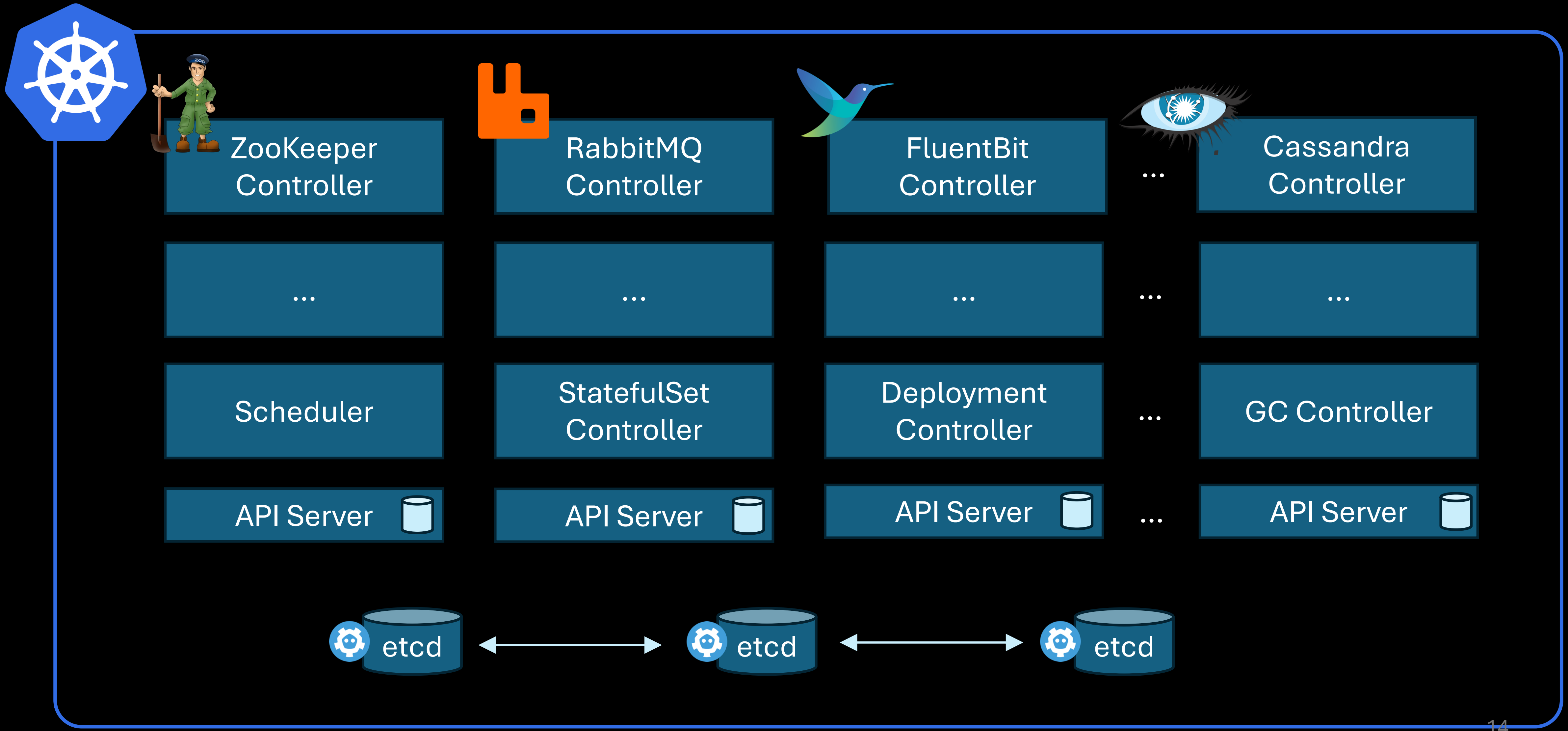
Summary (from the paper)

- Individual systems become simpler and more fine-grained
  - More friendly for testing, analysis, and verification
- Cross-system interactions become more complex and error-prone
  - New tools and practices are needed
- Traditional reliability tools are insufficient
  - Many only reason about control- and data-flow within a program

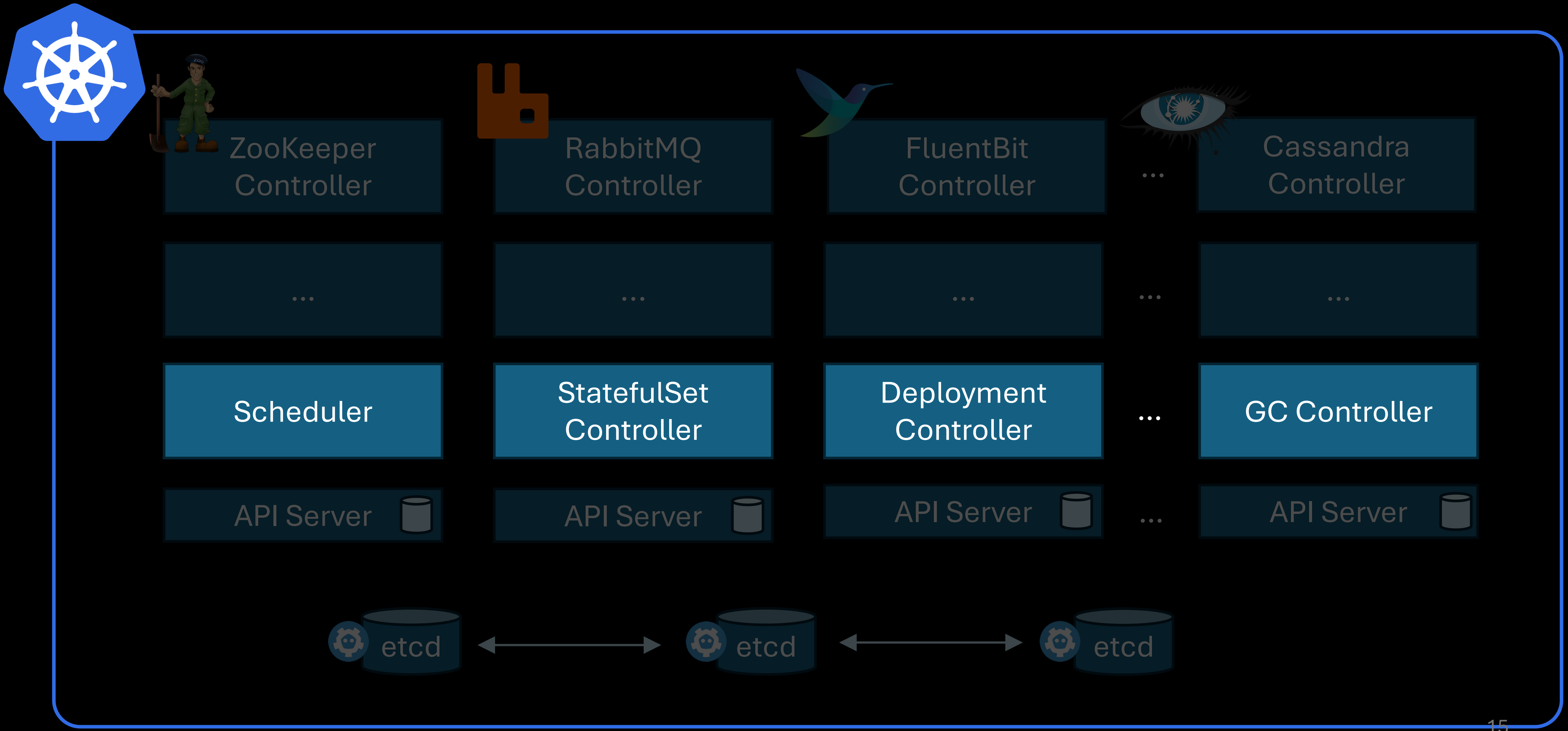
What can be done about it?

- Testing and verification of systems with interactions
  - Find bugs manifested through interactions
  - Build formally verified systems with guaranteed safety and liveness

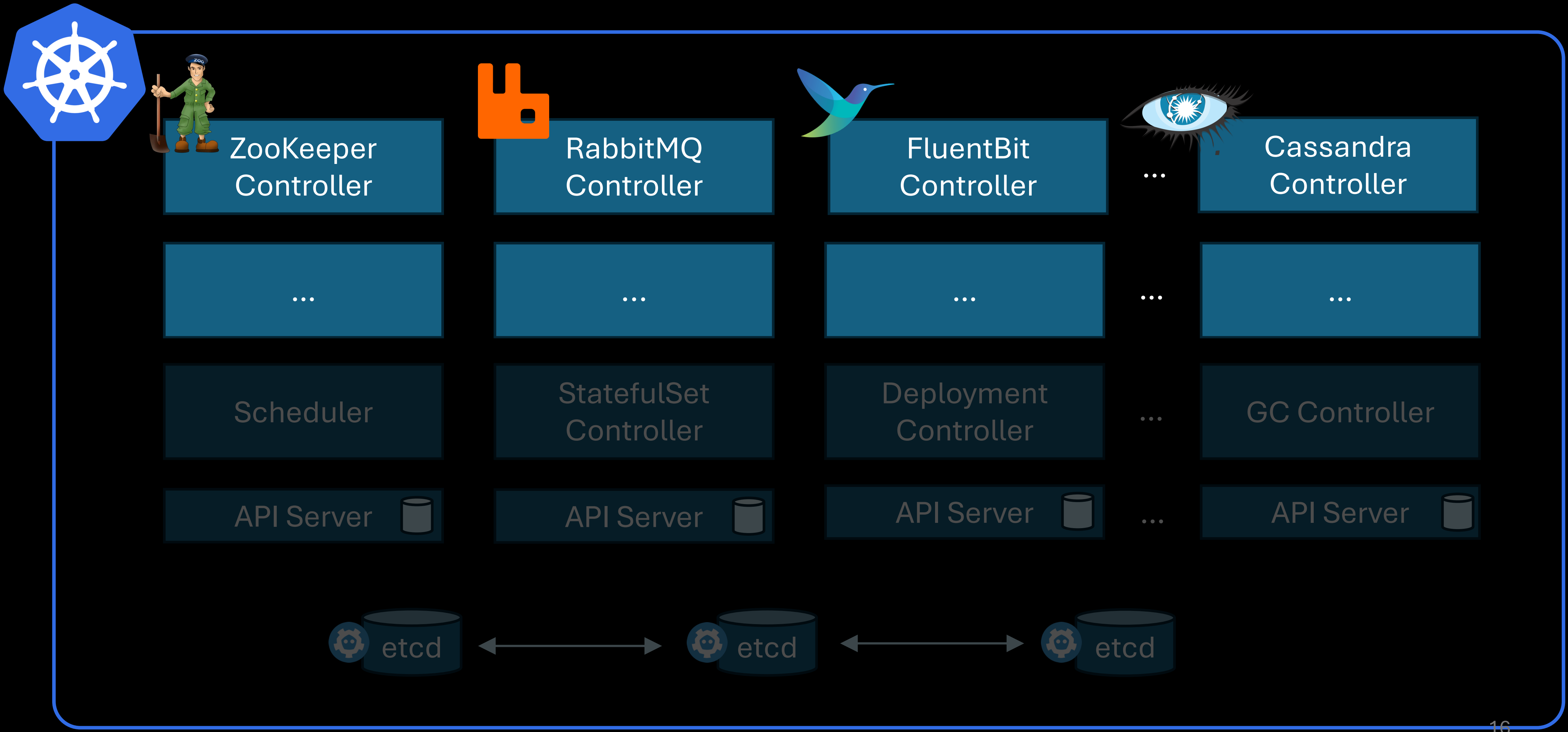
Kubernetes as a running (microservice) system



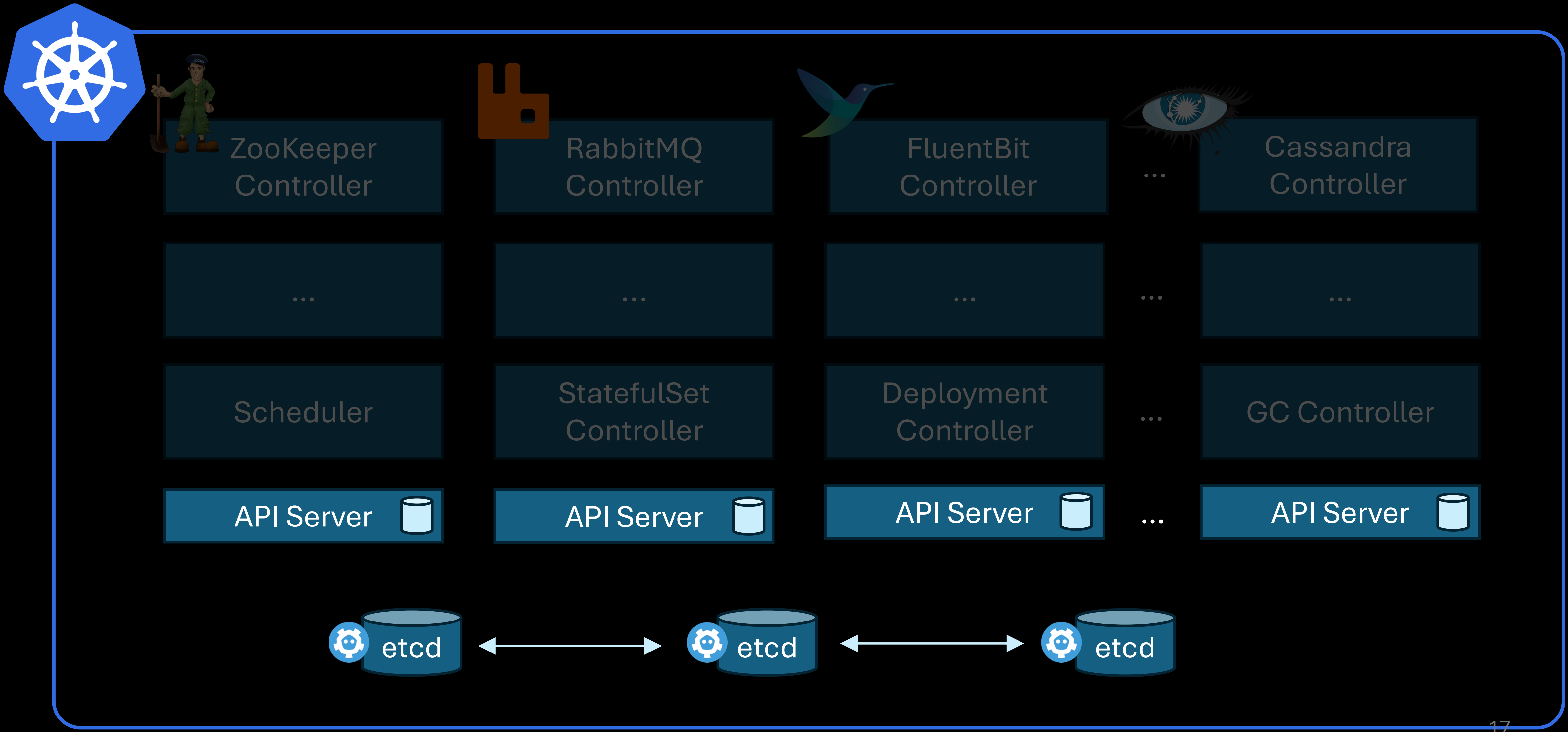
Kubernetes as a running (microservice) system



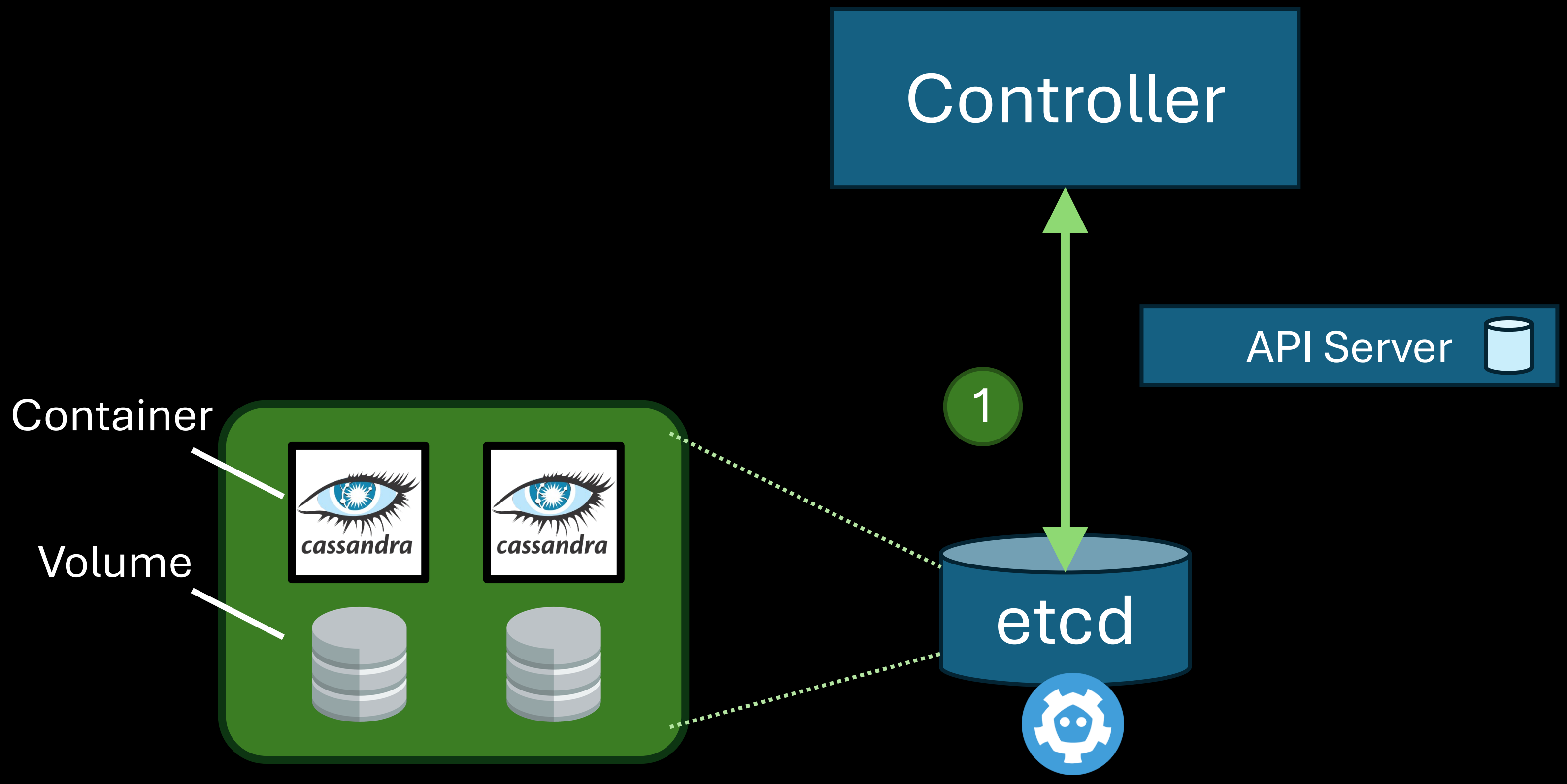
Kubernetes as a running (microservice) system



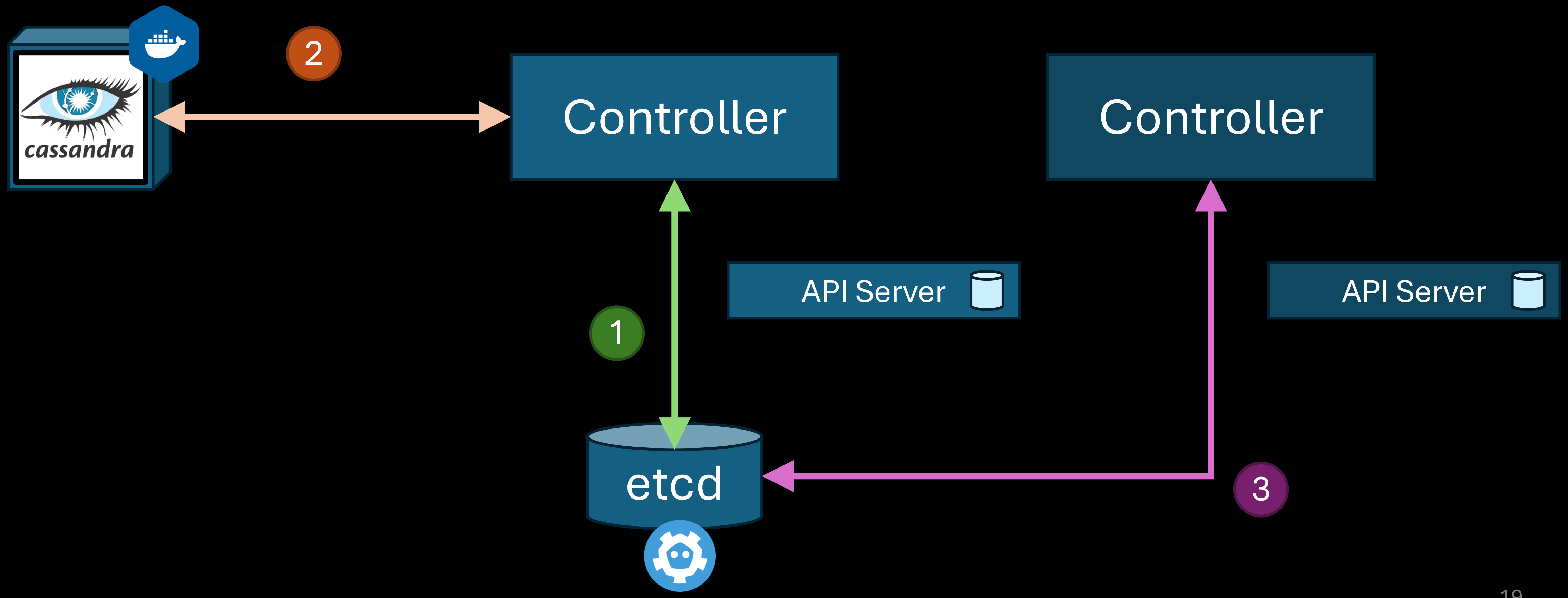
Kubernetes as a running (microservice) system



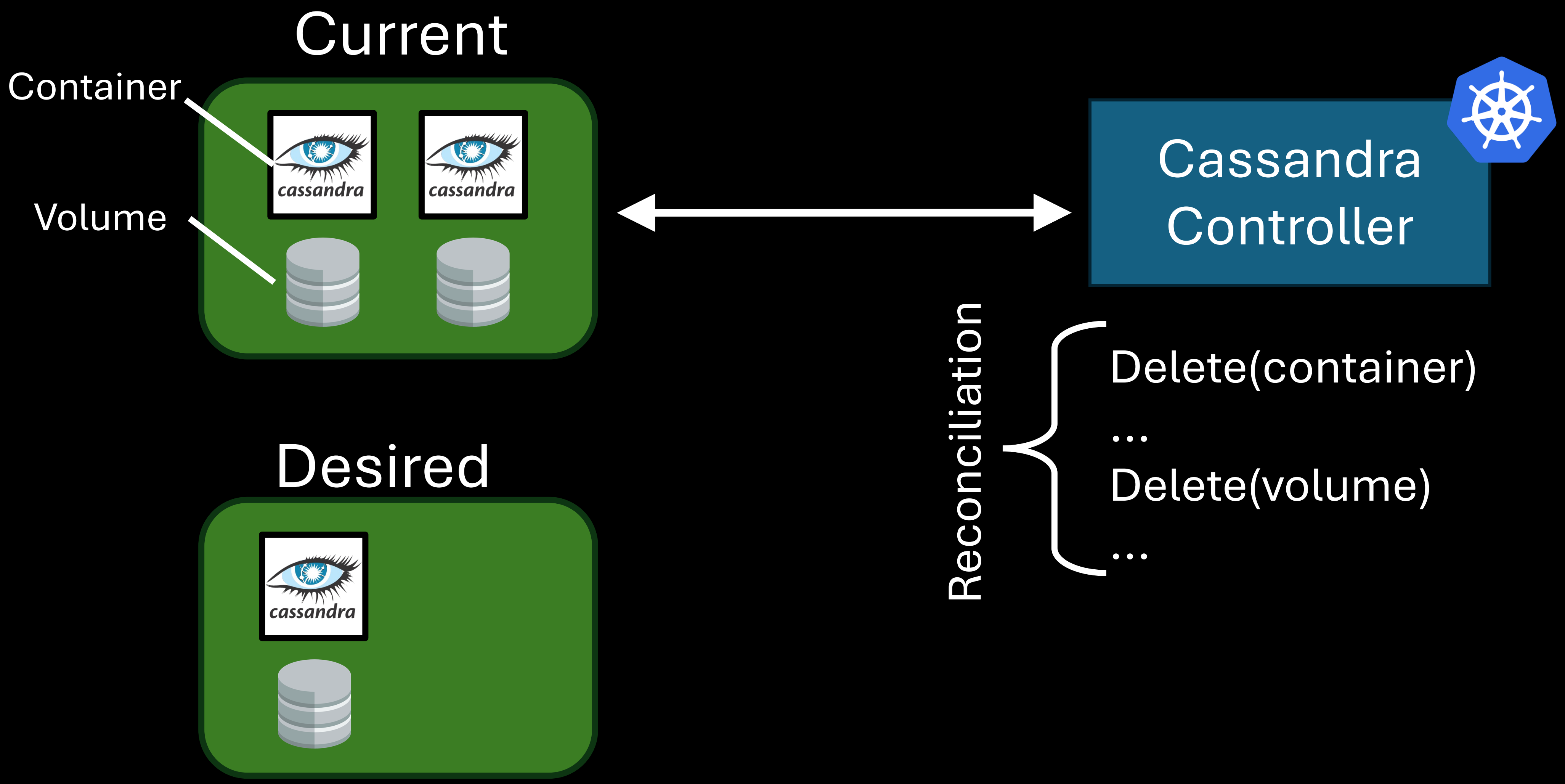
Controller and different types of interactions



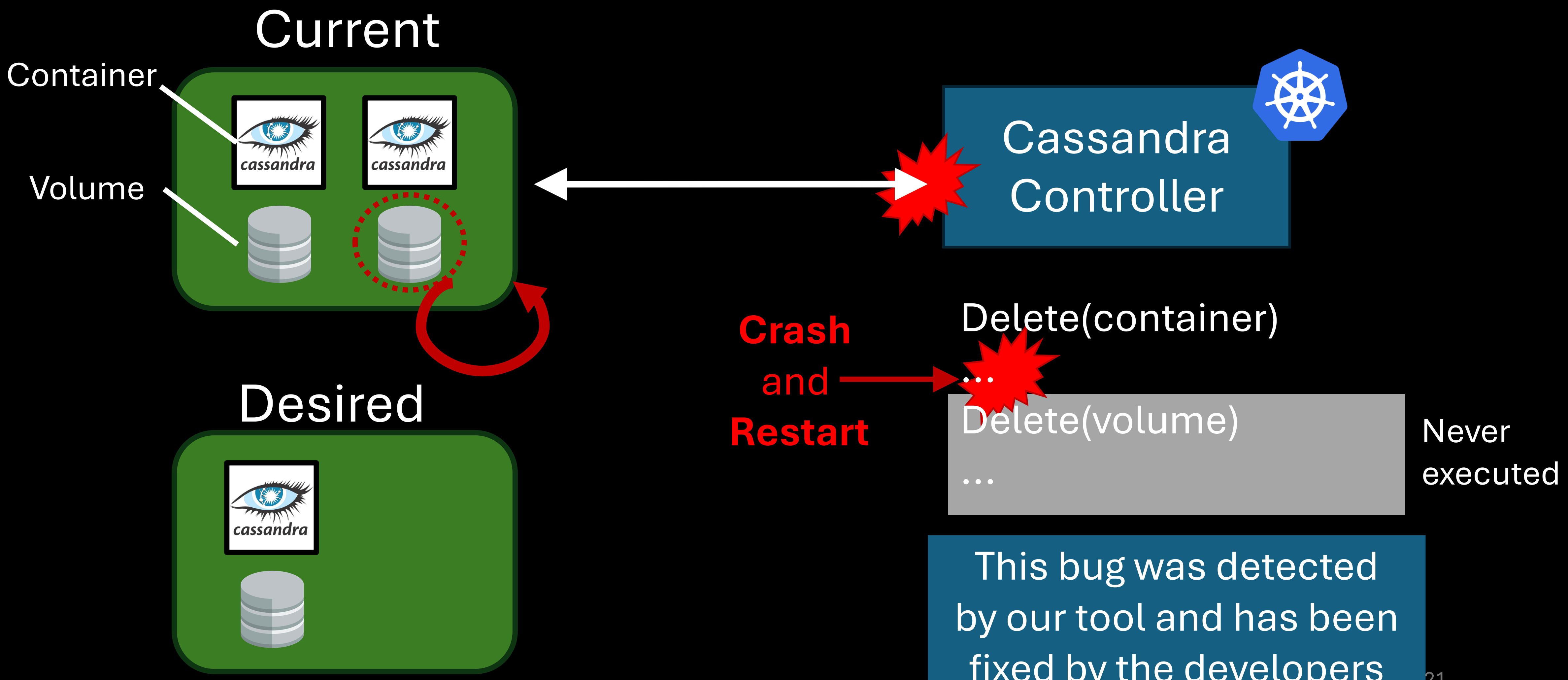
Controller and different types of interactions



Interaction between controller and system state



Unreliable interactions lead to disasters



This bug was detected by our tool and has been fixed by the developers

Sieve for automatic reliability testing

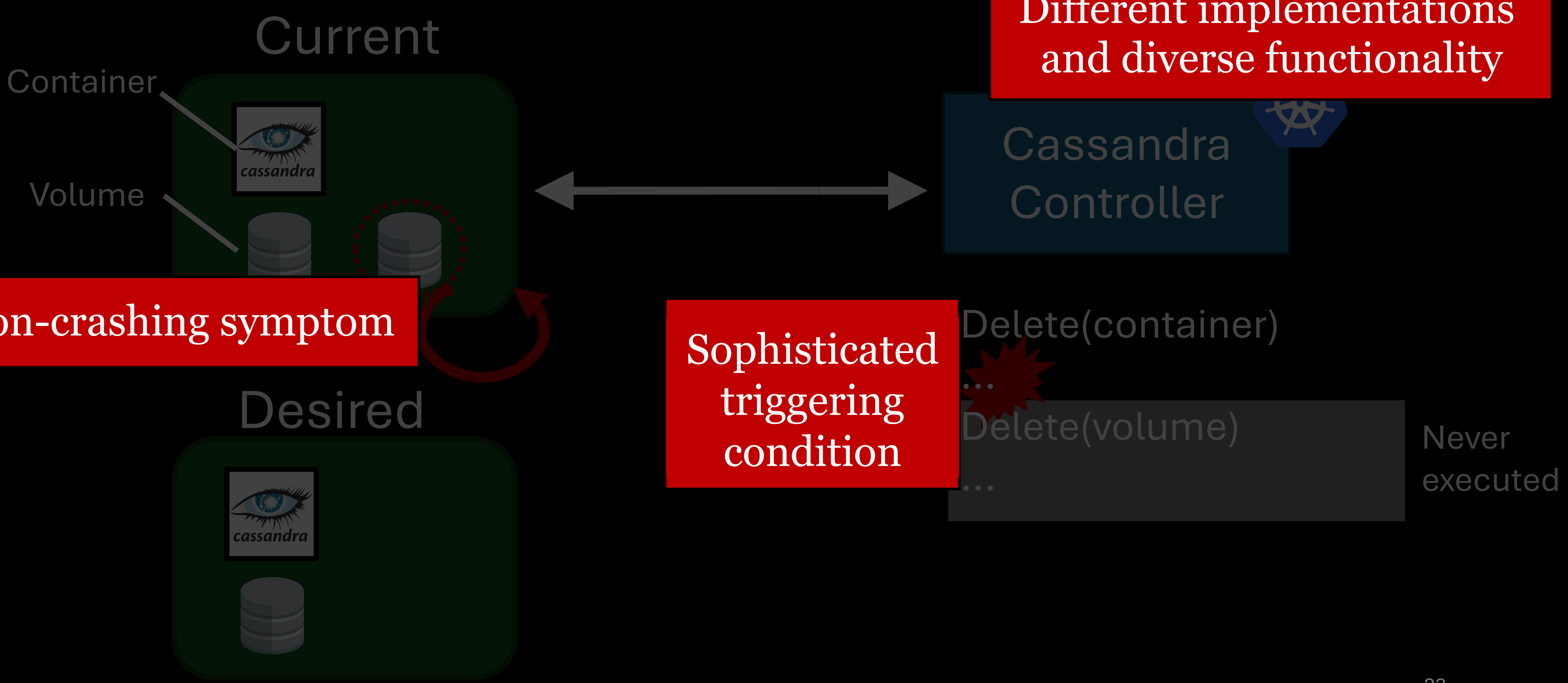
ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

- Key Idea: Perturbing the controller's interaction with the system state
  - **Usability:** Testing unmodified controllers
  - **Reproducibility:** Reproducing detected bugs reliably
- Detected 46 serious bugs in 10 popular Kubernetes controllers
  - **Severe consequences:** System outage, data loss, security issues, etc.
  - 35 confirmed and 22 fixed
- Available: https://github.com/sieve-project/sieve

Challenges of testing the interaction

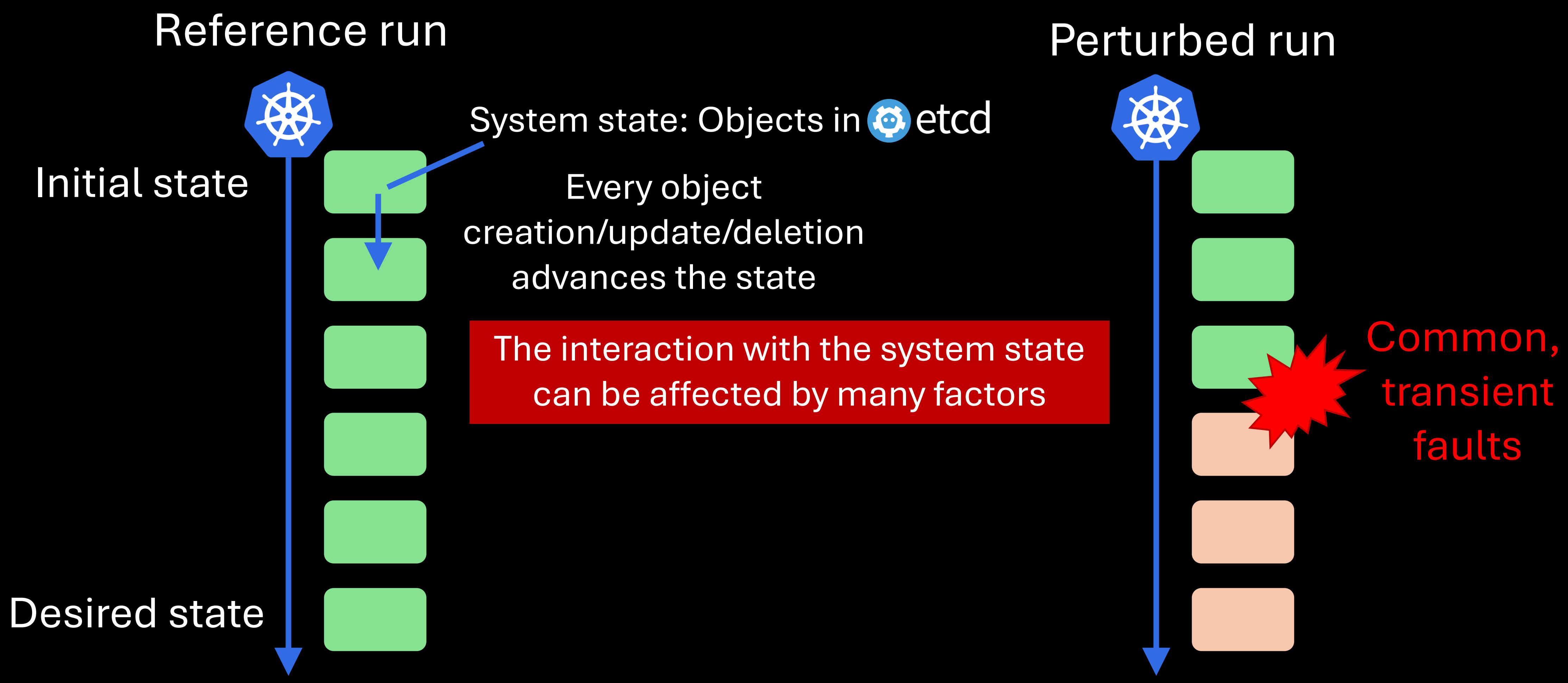


Different implementations and diverse functionality

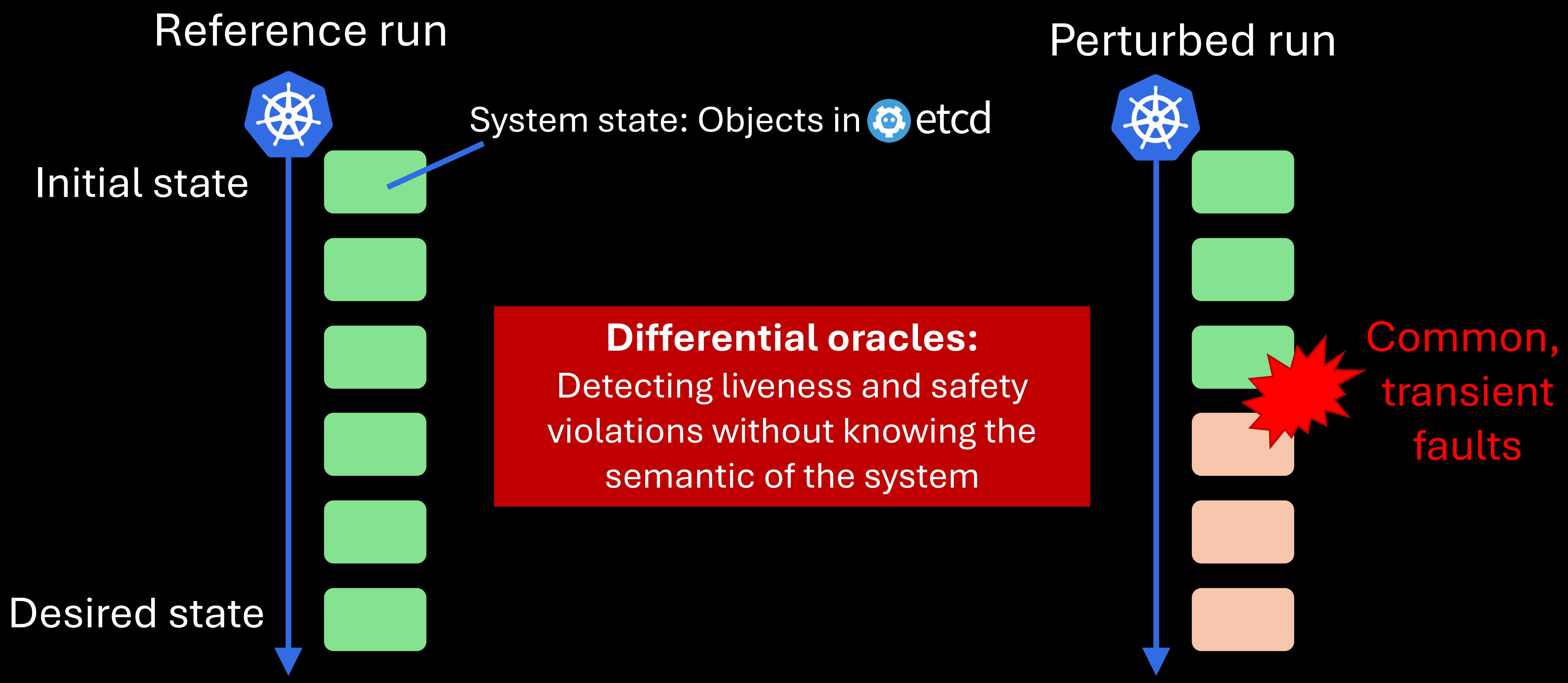
Non-crashing symptom

Sophisticated triggering condition

Perturb the controller's interaction with the state

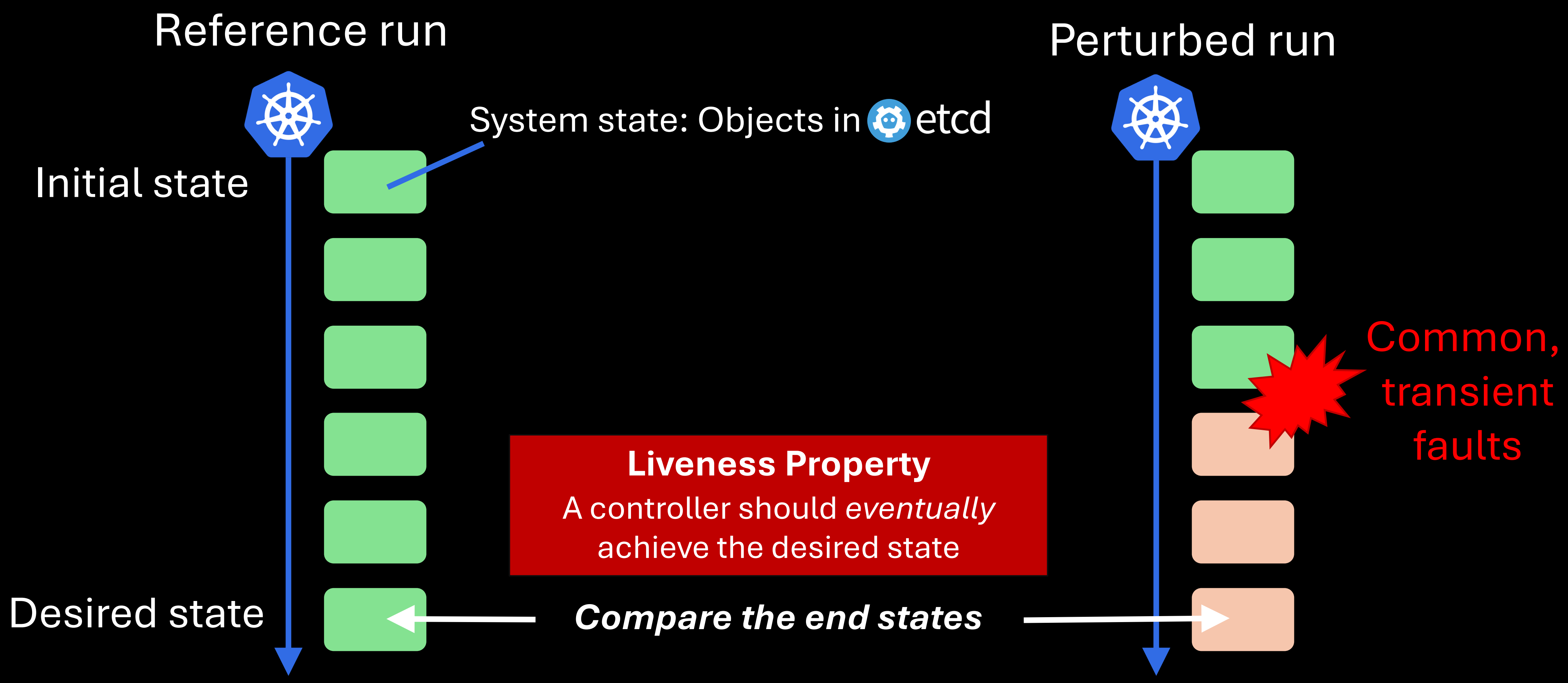


Flag buggy behavior with *differential* oracles

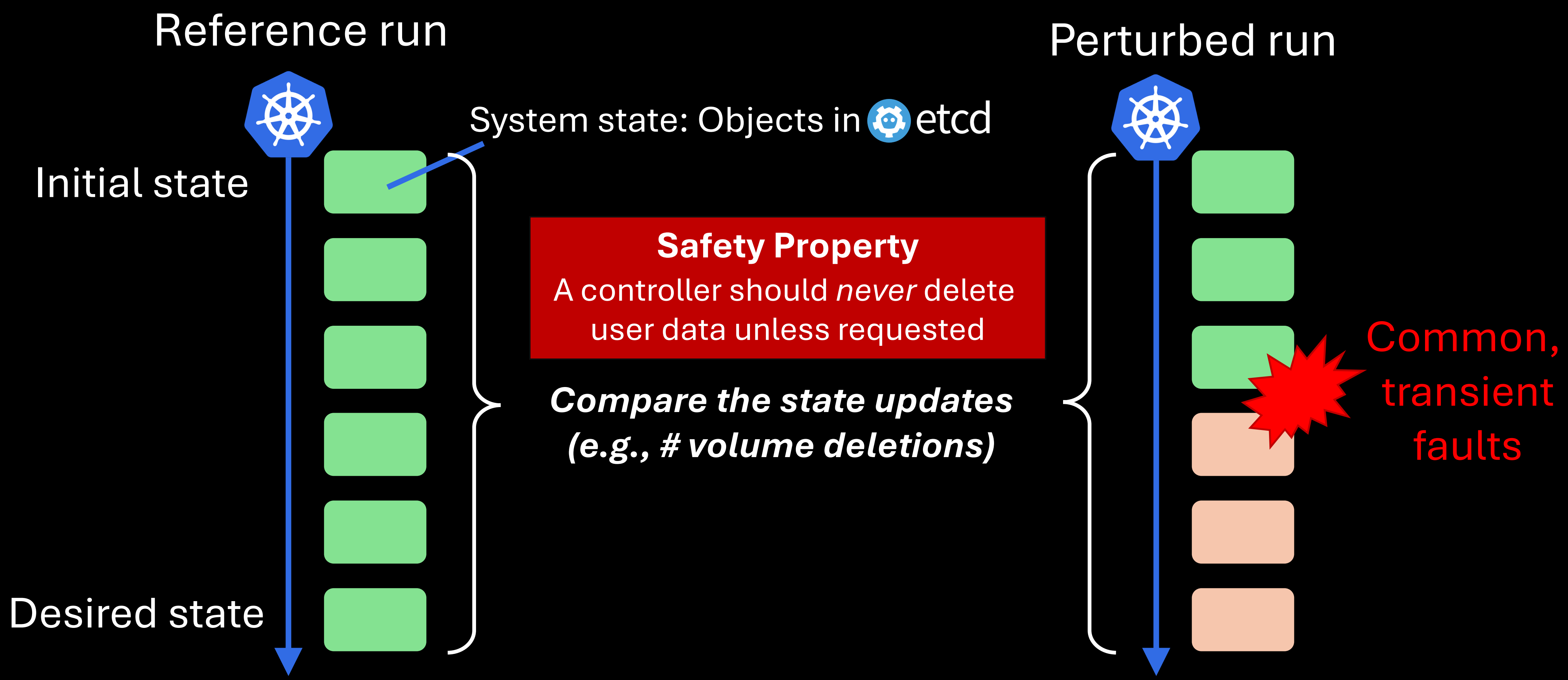


Differential oracles:
Detecting liveness and safety violations without knowing the semantic of the system

Flag buggy behavior with *differential* oracles



Flag buggy behavior with *differential* oracles



Exhaustive perturbation with different patterns

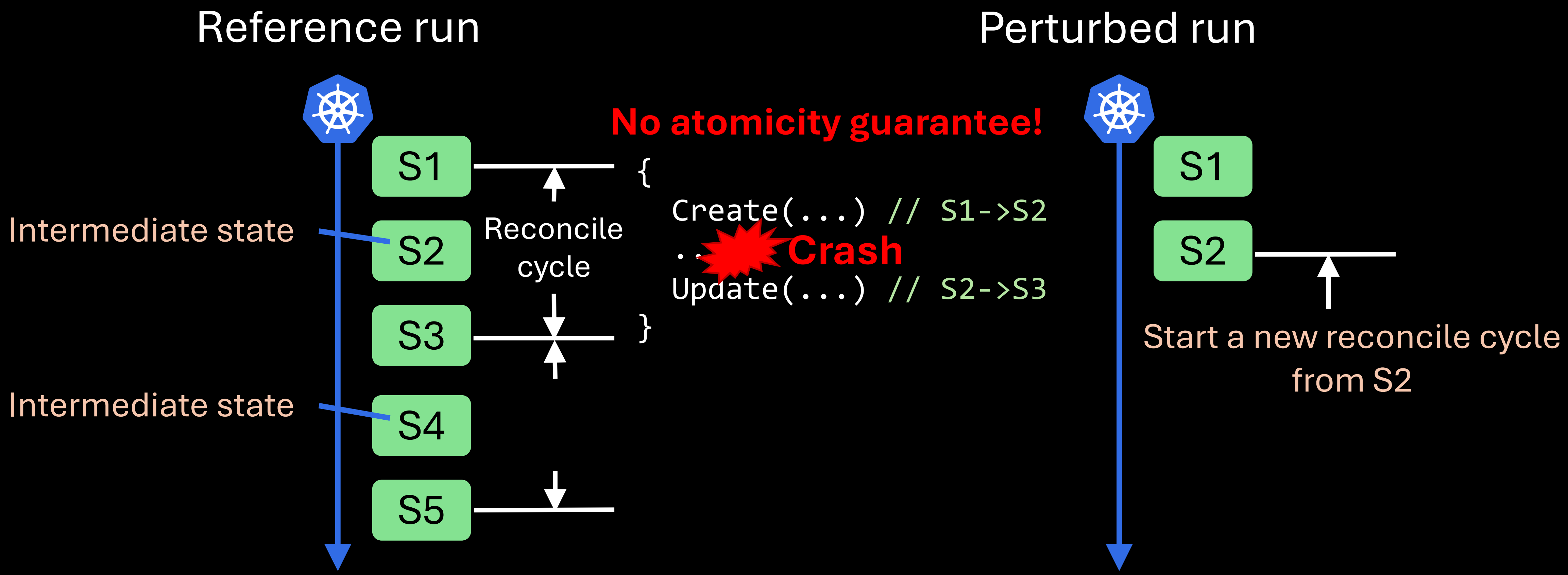
- Employ **three perturbation patterns**
  - Intermediate-state pattern
  - Stale-state pattern
  - Unobserved-state pattern

- **Exhaustively** test all bug-triggering perturbations
  - Systematically find all the targeted bugs
  - Inject faults with different timings

- Prune out ineffective perturbations to be **efficient**
  - Not every perturbation leads to bugs

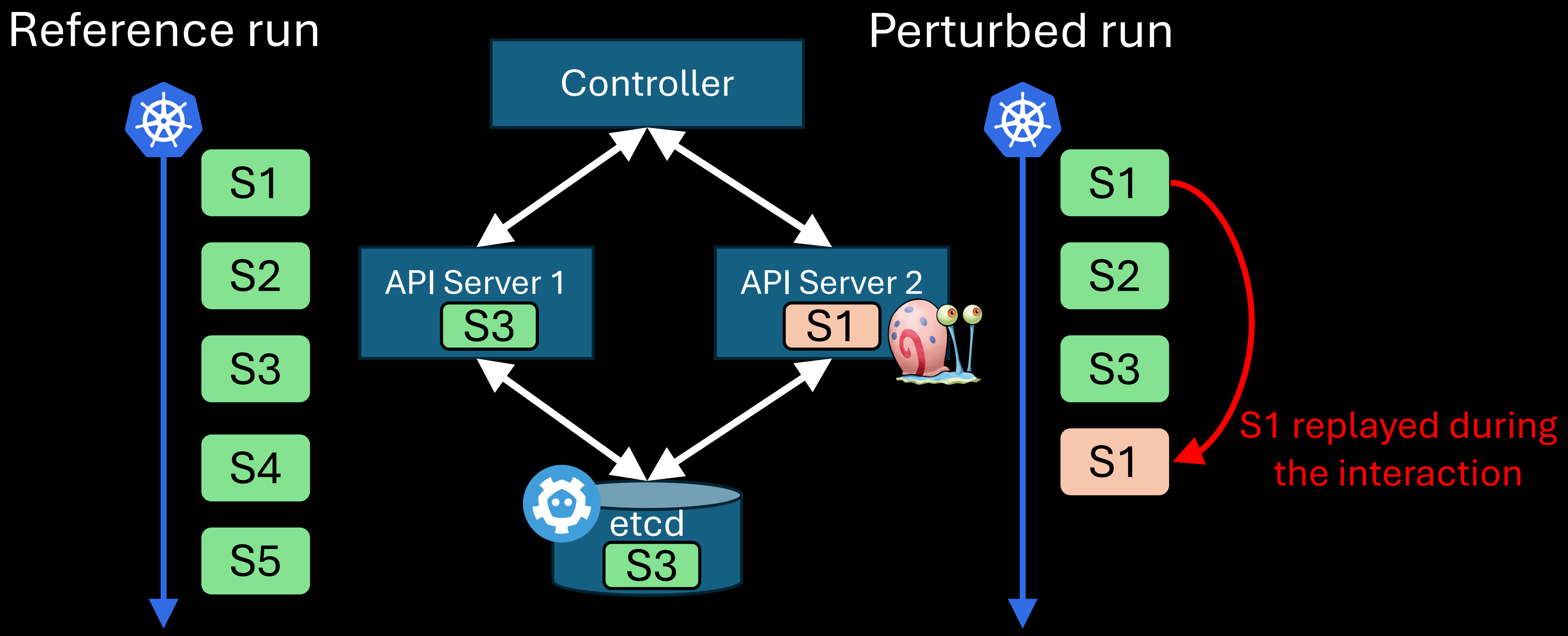
The intermediate-state pattern

The interaction fails in the middle, leaving the controller to handle some intermediate state



The stale-state pattern

The interaction is affected by staleness caused by asynchrony and caching



The stale-state pattern

The interaction is affected by staleness caused by asynchrony and caching

Pull requests 813 · Actions · Projects 6 · Security · Insights

Kubernetes is vulnerable to stale reads, violatir
#59848

Open · **smarterclayton** opened this issue on Feb 13, 2018 · 89 comments

smarterclayton commented on Feb 13, 2018 • edited

When we added resourceVersion=0 to reflectors, we didn't properly reason about its ir
can cause two nodes to run a pod with the same name at the same time when using m
pod safety guarantees on the cluster. Because a read serviced by the watch cache car
connects to that api server can read an arbitrarily old history. We explicitly use quorum

Scenario:

1. T1: StatefulSet controller creates `pod-0` (uid 1) which is scheduled to `node-1`
2. T2: `pod-0` is deleted as part of a rolling upgrade
3. `node-1` sees that `pod-0` is deleted and cleans it up, then deletes the pod in the a
4. The StatefulSet controller creates a second pod `pod-0` (uid 2) which is assigned t
5. `node-2` sees that `pod-0` has been scheduled to it and starts `pod-0`
6. The kubelet on `node-1` crashes and restarts, then performs an initial list of pods s
   an HA setup (more than one API server) that is partitioned from the master (watch
   watch cache returns a list of pods from before T2
7. `node-1` fills its local cache with a list of pods from before T2
8. `node-1` starts `pod-0` (uid 1) and `node-2` is already running `pod-0` (uid 2).

Kubernetes Podcast from Google

Episodes · About · Subscribe

#218 February 9, 2024

Kubernetes stale reads, with Madhav Jivrajani

Hosts: Abdel Sghiouar, Kaslin Fields

Episode #218: Kubernetes stale reads, with Madhav Jivrajani 43:36

Madhav Jivrajani is an engineer at VMware, a tech lead in SIG Contributor Experience and a GitHub Admin for the Kubernetes project. He also contributes to the storage layer of Kubernetes, focusing on reliability and scalability.

In this episode we talked with Madhav about a recent post on social media about a very interesting stale reads issue in Kubernetes, and what the community is doing about it.

Do you have something cool to share? Some questions? Let us know:

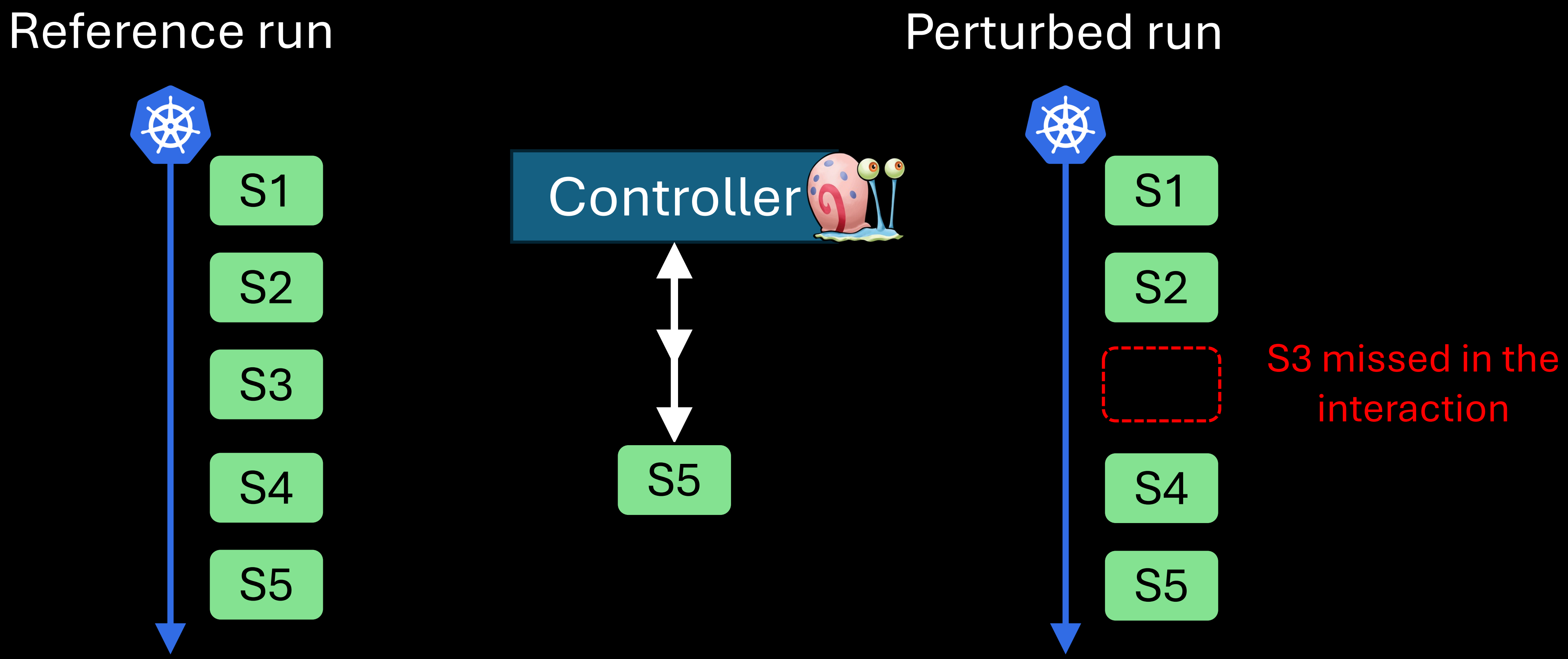
- web: kubernetespodcast.com
- mail: kubernetespodcast@google.com
- twitter: @kubernetespod

Backlog

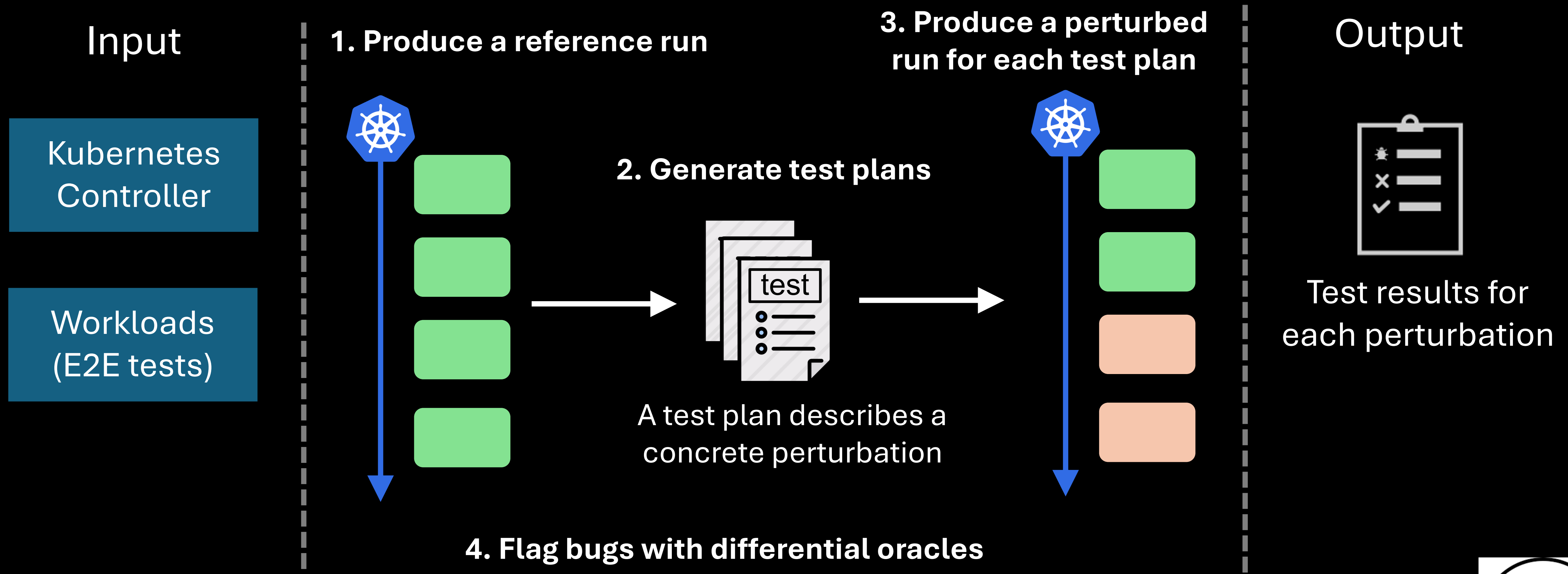
Milestone

The unobserved-state pattern

There are observability gaps in the interaction



Sieve: Testing interaction with the system state



Open source: https://github.com/sieve-project/sieve

Interaction between controller and application

- Controller reconfigures the managed applications
  - must respect application operation semantics

Interaction between controller and application

Modifying the current dynamic configuration

Modifying the configuration is done through the `reconfig` command. There are two modes of reconfiguration: incremental and non-incremental (bulk). The non-incremental simply specifies the new dynamic configuration of the system. The incremental specifies changes to the current configuration. The `reconfig` command returns the new configuration.

A few examples are in: `ReconfigTest.java`, `ReconfigRecoveryTest.java` and `TestReconfigServer.cc`.

General

Removing servers: Any server can be removed, including the leader (although removing the leader will result in a short unavailability, see Figures 6 and 8 in the paper). The server will not be shut-down automatically. Instead, it becomes a "non-voting follower". This is somewhat similar to an observer in that its votes don't count towards the Quorum of votes necessary to commit operations. However, unlike a non-voting follower, an observer doesn't actually see any operation proposals and does not ACK them. Thus a non-voting follower has a more significant negative effect on system throughput compared to an observer. Non-voting follower mode should only be used as a temporary mode, before shutting the server down, or adding it as a follower or as an observer to the ensemble. We do not shut the server down automatically for two main reasons. The first reason is that we do not want all the clients connected to this server to be immediately disconnected, causing a flood of connection requests to other servers. Instead, it is better if each client decides when to migrate independently. The second reason is that removing a server may sometimes (rarely) be necessary in order to change it from "observer" to "participant" (this is explained in the section Additional comments).

Note that the new configuration should have some minimal number of participants in order to be considered legal. If the proposed change would leave the cluster with less than 2 participants and standalone mode is enabled (standaloneEnabled=true, see the section The standaloneEnabled flag), the reconfig will not be processed (BadArgumentsException). If standalone mode is disabled (standaloneEnabled=false) then its legal to remain with 1 or more participants.

Adding servers: Before a reconfiguration is invoked, the administrator must make sure that a quorum (majority) of participants from the new configuration are already connected and synced with the current leader. To achieve this we need to connect a new joining server to the leader before it is officially part of the ensemble. This is done by starting the joining server using an initial list of servers which is technically not a legal configuration of the system but (a) contains the joiner, and (b) gives sufficient information to the joiner in order for it to find and connect to the current leader. We list a few different options of doing this safely.

1. Initial configuration of joiners is comprised of servers in the last committed configuration and one or more joiners, where **joiners are listed as observers.** For example, if servers D and E are added at the same time to (A, B, C) and server C is being removed, the initial configuration of D could be (A, B, C, D) or (A, B, C, D, E), where D and E are listed as observers. Similarly, the configuration of E could be (A, B, C, E) or (A, B, C, D, E), where D and E are listed as observers. **Note that listing the joiners as observers will not actually make them observers - it will only prevent them from accidentally forming a quorum with other joiners.** Instead, they will contact the servers in the current configuration and adopt the last committed configuration (A, B, C), where the joiners are absent. Configuration files of joiners are backed up and replaced automatically as this happens. After connecting to the current leader, joiners become non-voting followers until the system is reconfigured and they are added to the ensemble (as participant or observer, as appropriate).
2. Initial configuration of each joiner is comprised of servers in the last committed configuration + **the joiner itself, listed as a participant.** For example, to add a new server D to a configuration consisting of servers (A, B, C), the administrator can start D using an initial configuration file consisting of servers (A, B, C, D). If both D and E are added at the same time to (A, B, C), the initial configuration of D could be (A, B, C, D) and the configuration of E could be (A, B, C, E). Similarly, if D is added and C is removed at the same time, the initial configuration of D could be (A, B, C, D). Never list more than one joiner as participant in the initial configuration (see warning below).
3. Whether listing the joiner as an observer or as participant, it is also fine not to list all the current configuration servers, as long as the current leader is in the list. For example, when adding D we could start D with a configuration file consisting of just (A, D) if A is the current leader. however this is more fragile since if A fails before D officially joins the ensemble, D doesn’t know anyone else and therefore the administrator will have to intervene and restart D with another server list.

> **Warning**
>
> Never specify more than one joining server in the same initial configuration as participants. Currently, the joining servers don’t know that they are joining an existing ensemble; if multiple joiners are listed as participants they may form an independent quorum creating a split-brain situation such as processing operations independently from your main ensemble. It is OK to list multiple joiners as observers in an initial config.

Finally, note that once connected to the leader, a joiner adopts the last committed configuration, in which it is absent (the initial config of the joiner is backed up before being rewritten). If the joiner restarts in this state, it will not be able to boot since it is absent from its configuration file. In order to start it you’ll once again have to specify an initial configuration.

Interaction between controller and application

- Controller reconfigures the managed applications
  - must respect application operation semantics

- Must reason about **end-to-end** operation correctness
  - Unit tests are deficient

Acto: a push-button E2E testing tool

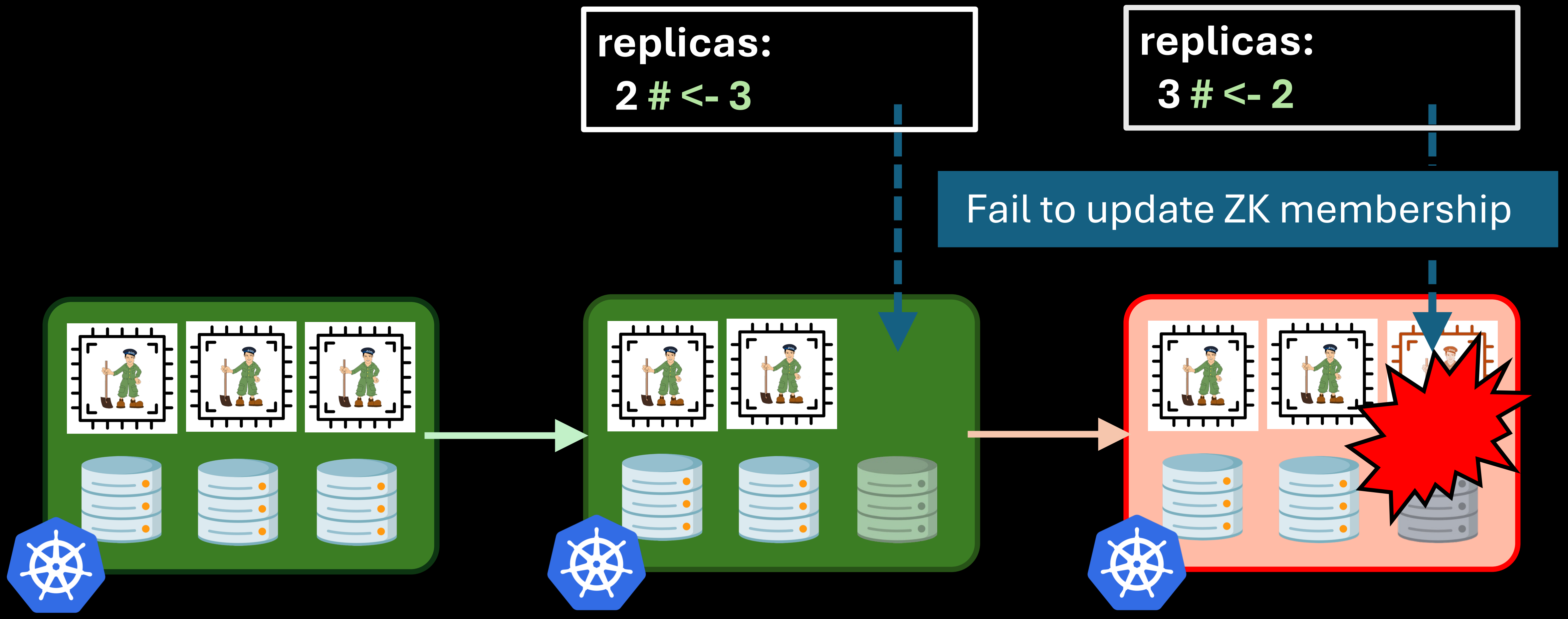
- Testing the controller *together with* the managed applications
  - complement unit tests

- Checking **end-to-end** correctness properties
  - *always* reconciling the managed application to its desired states
  - *always* recovering the application from undesired or error states
  - *always* being resilient to operation errors

- Detected **56** serious bugs in 11 popular Kubernetes controllers
  - 42 confirmed and 30 fixed

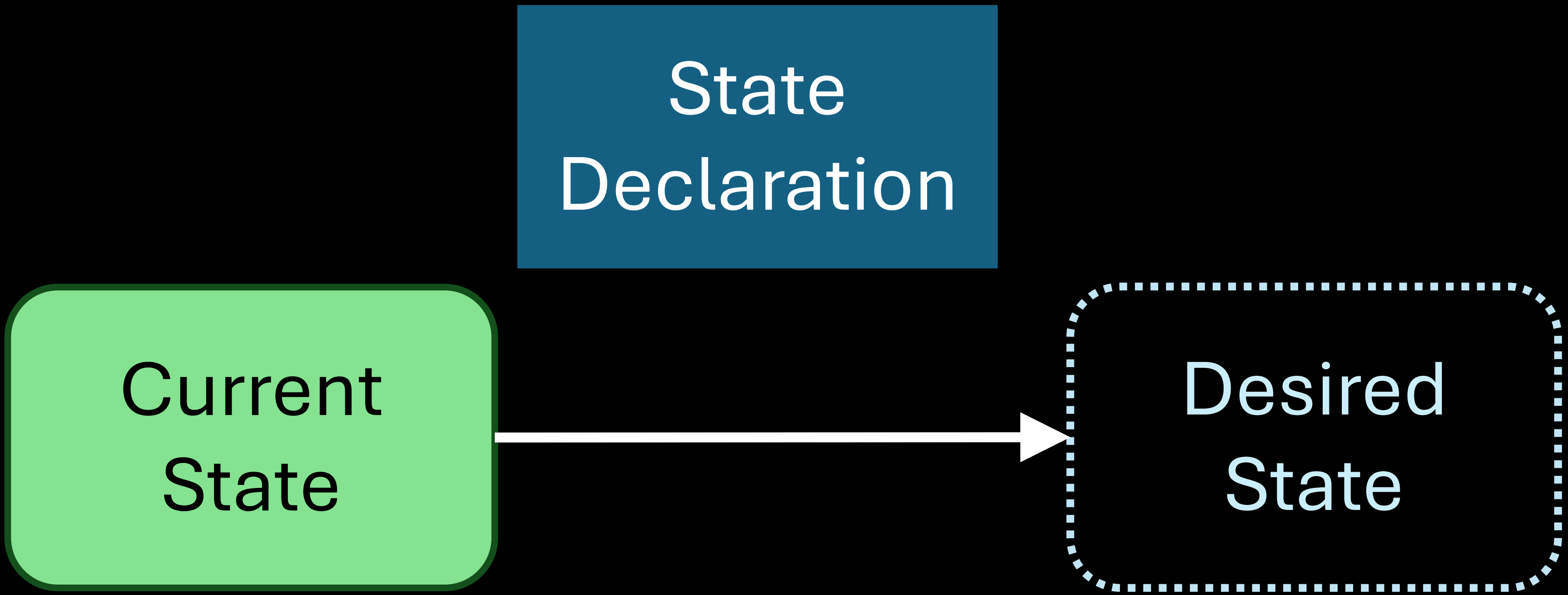
- Available: https://github.com/xlab-uiuc/acto

Interaction bugs detected by Acto



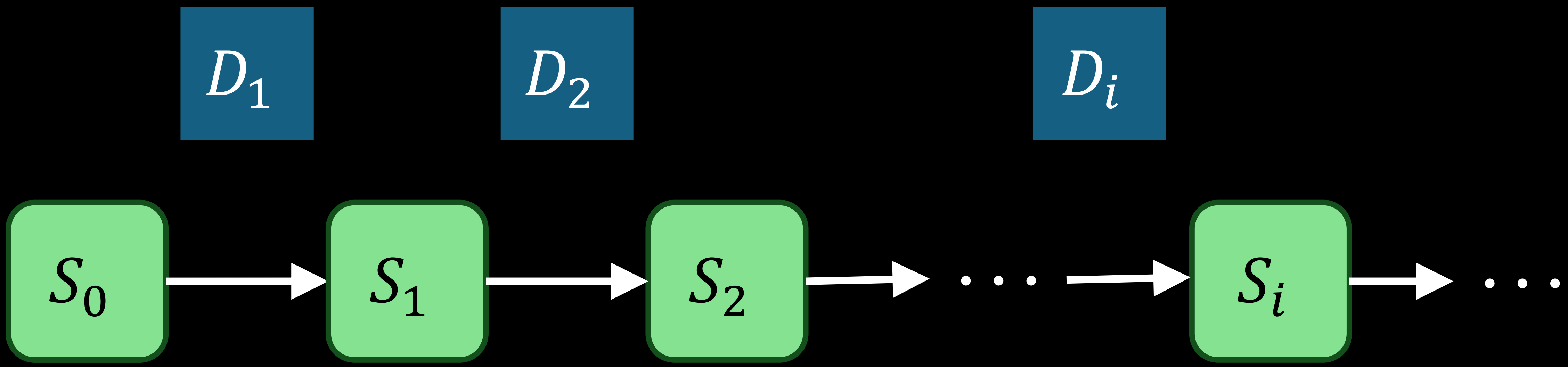
Basic idea: exploring different transitions of states

- Modeling operations as state transitions



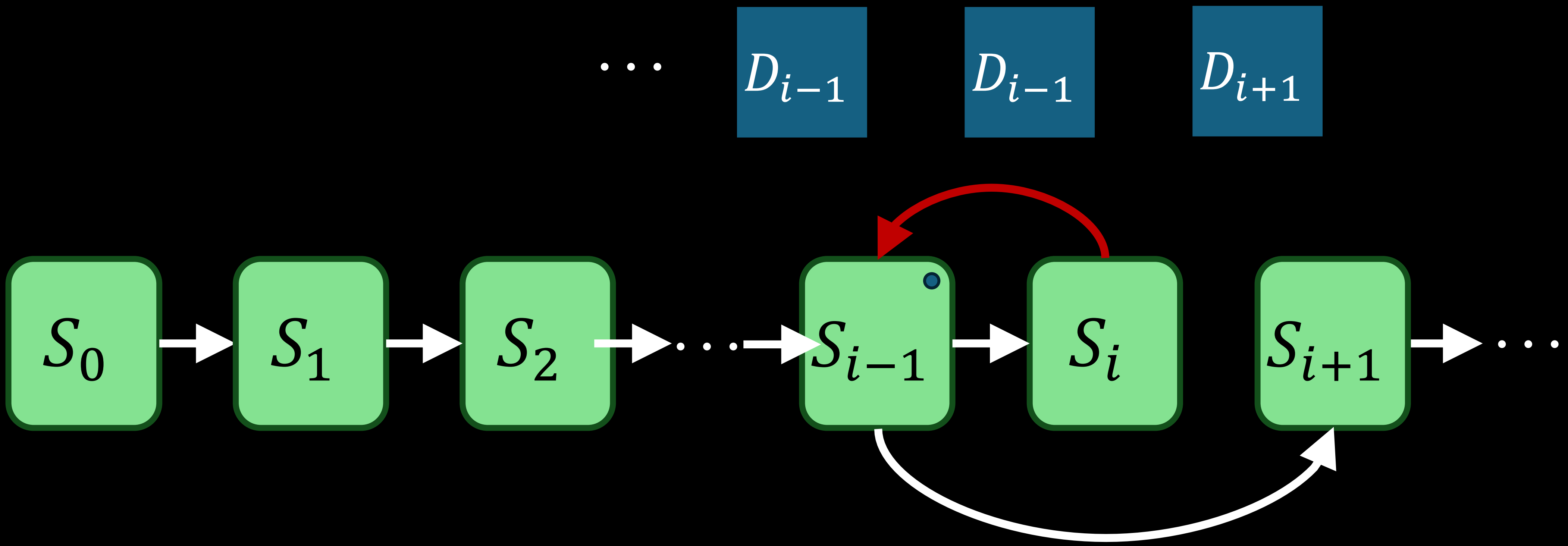
Basic idea: exploring different transitions of states

- Chaining state transitions into an operation sequence



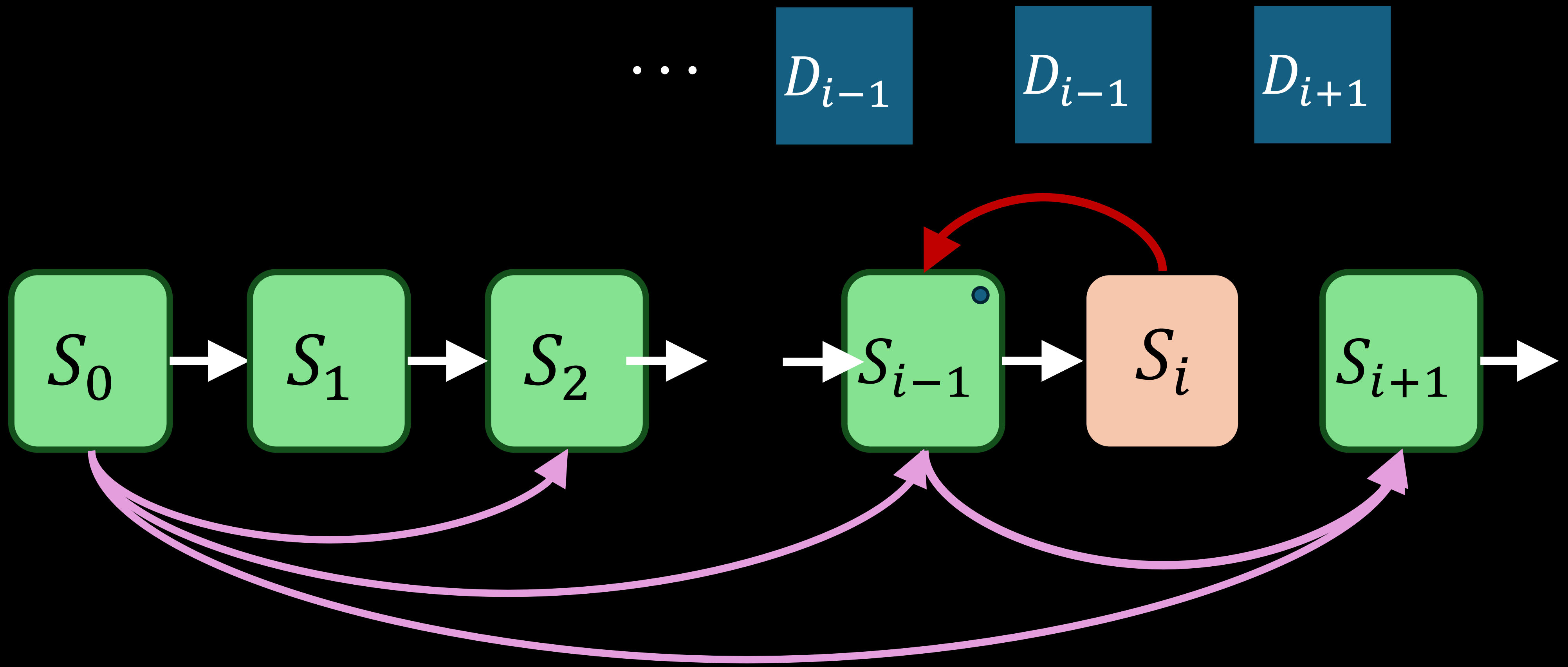
Basic idea: exploring different transitions of states

- Checking error-state recovery



Basic idea: exploring different transitions of states

- Checking the level-triggering principle



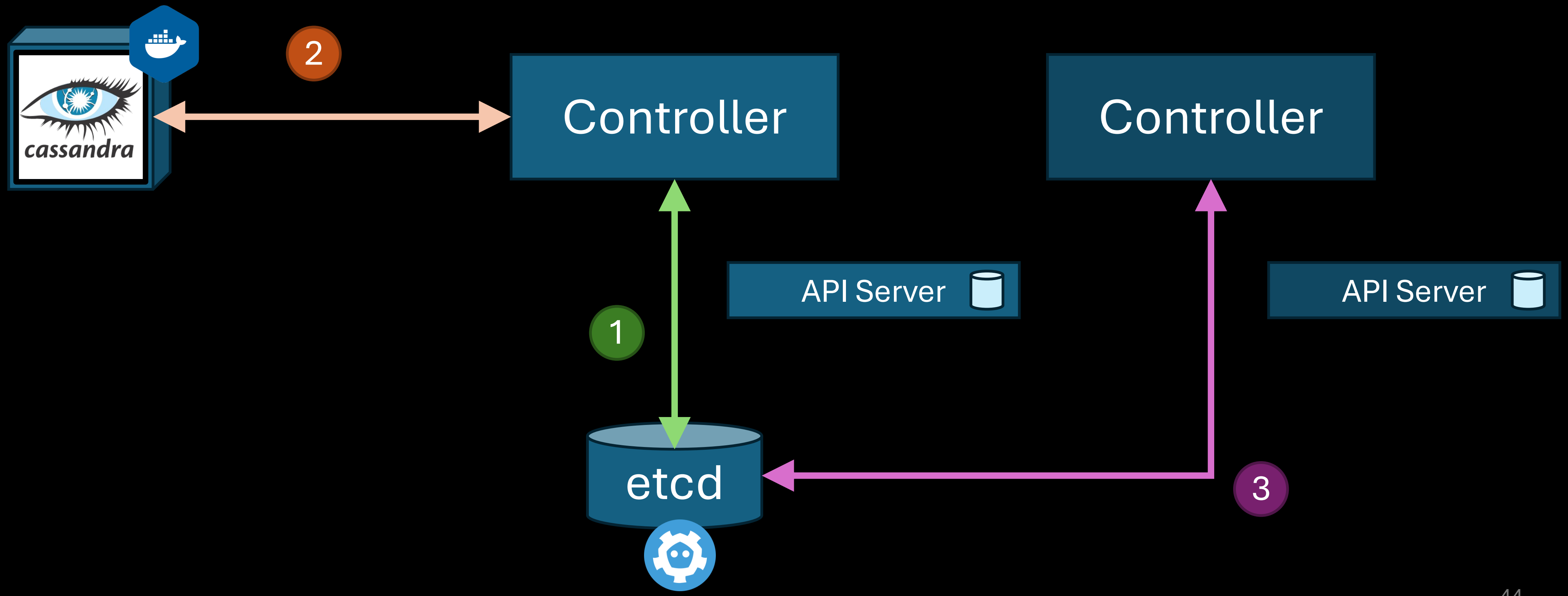
Secret sauces

- Automatic generation of comprehensive desired-state declarations
  - cover different operation scenarios
  - cover all the fields of the operation interface

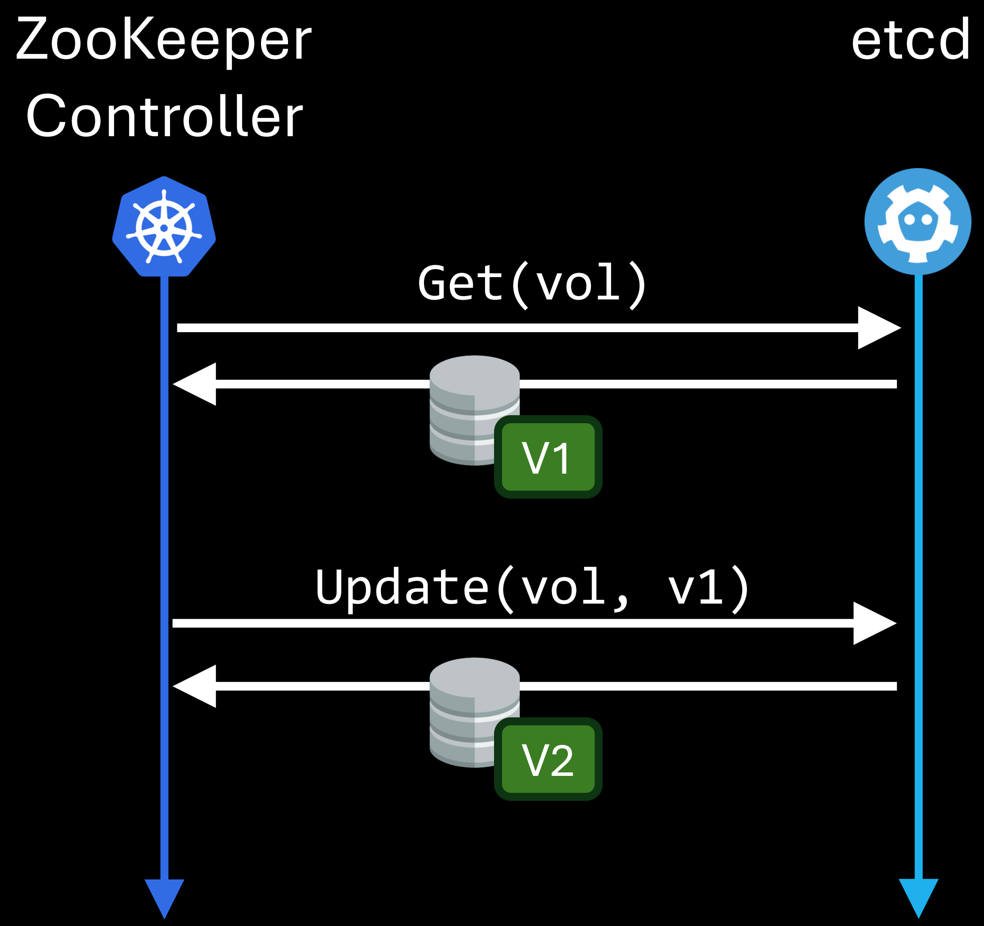
- Automatic test oracles for flagging undesired behavior
  - e.g., consistency and differential oracles

- **Open source**: https://github.com/xlab-uiuc/acto

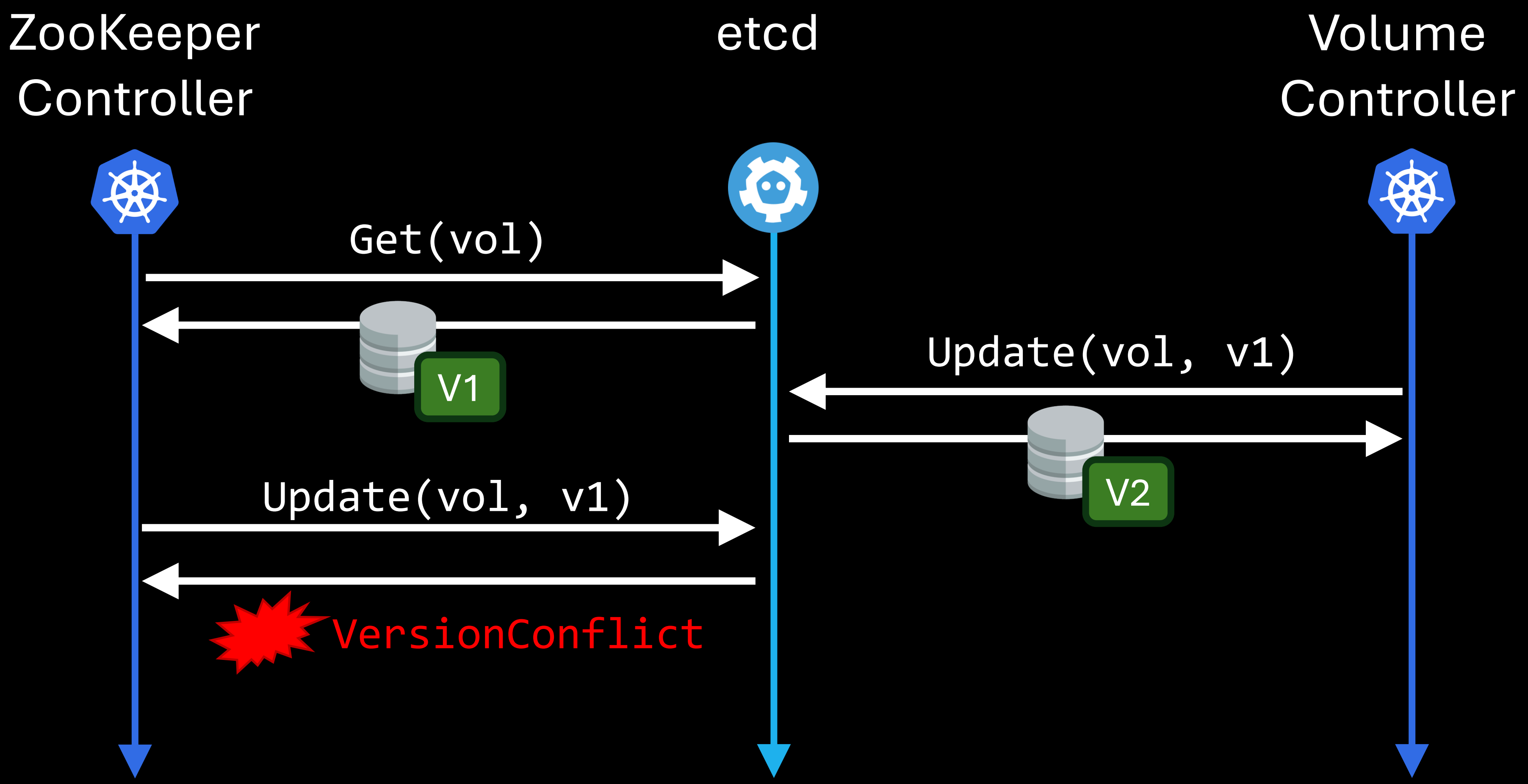
Verification? All types of interactions matter.



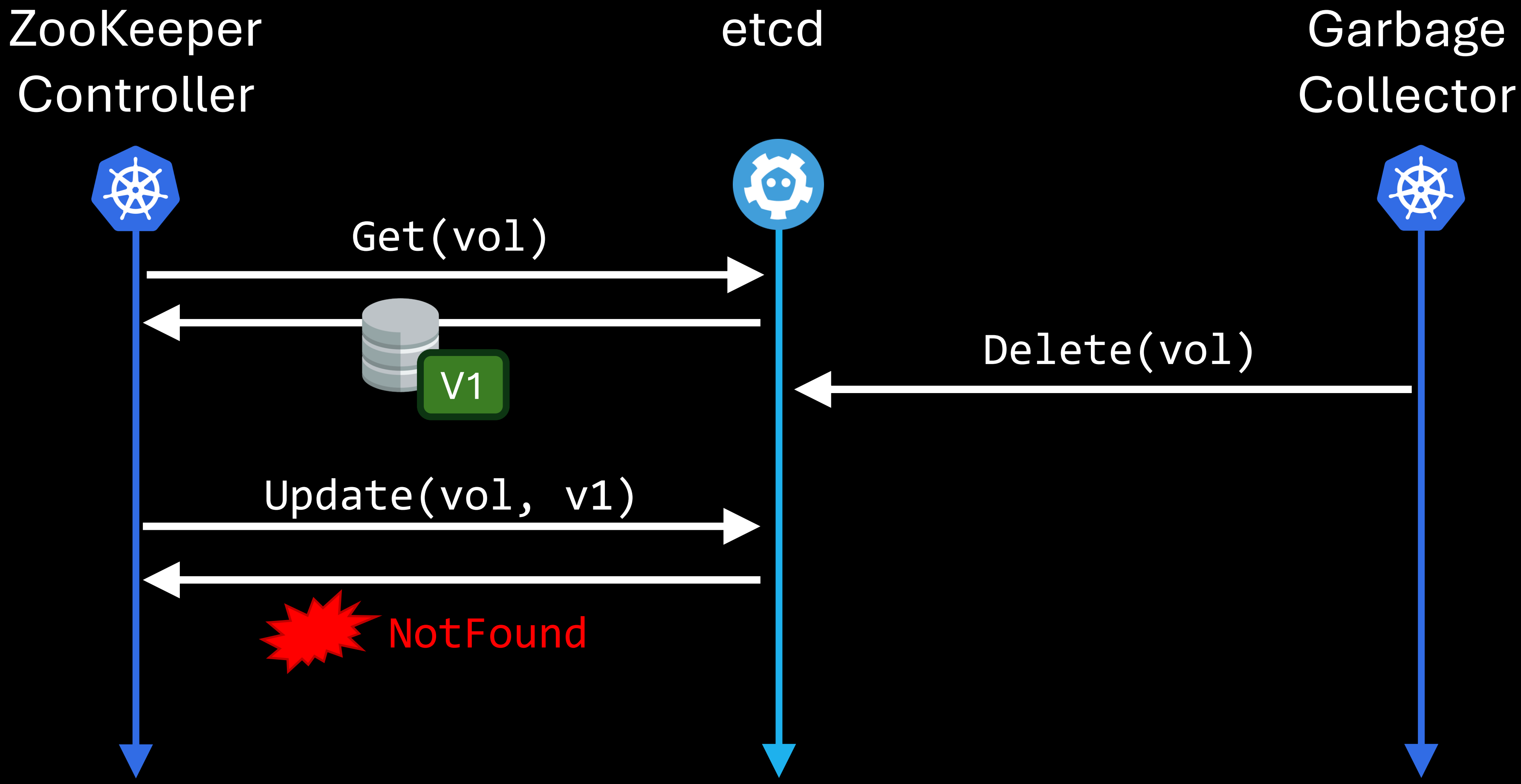
Interaction between controllers



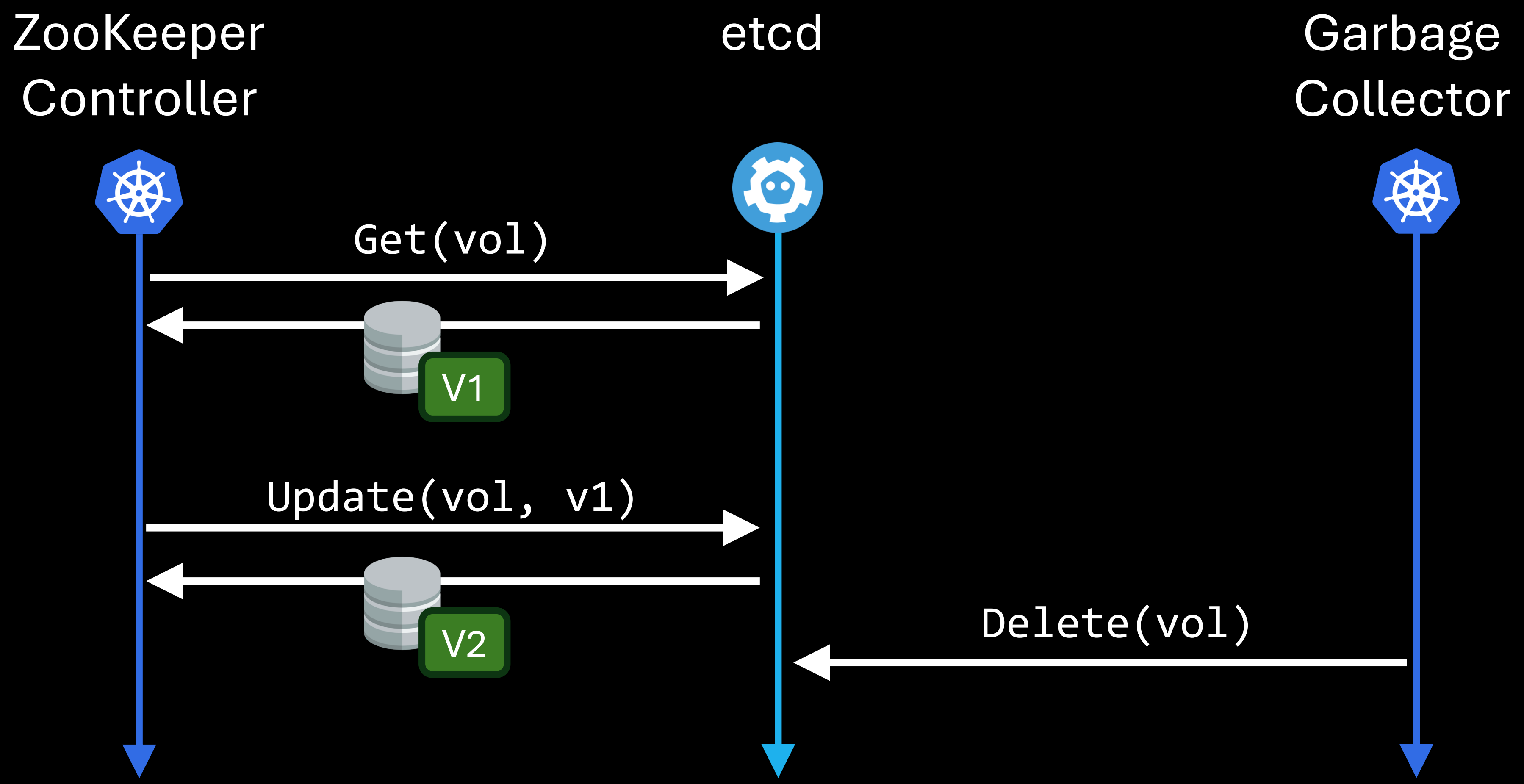
Interaction between controllers



Interaction between controllers



Interaction between controllers



Anvil: building verified Kubernetes controllers

- A framework to help build practical and verified controllers
  - **Verified**: the controller implementation is formally verified
  - **Practical**: the verified controller can be deployed in any Kubernetes clusters

- We have built three Kubernetes controllers using Anvil
  - Controllers for managing ZooKeeper, RabbitMQ and FluentBit
  - Feature parity and competitive performance

Modeling three types of interactions

- Interactions between the controller and the system state
  - API server and etcd that serves/stores the system state
- Interactions between the controller and the managed application
  - The managed application (customized by developers)
- Interactions between controllers
  - Built-in controllers that interact with the target controller
- Asynchrony and failures (e.g., controller crash, network delay)

Eventually Stable Reconciliation (ESR)

- A formal correctness specification for controllers
  - Generally applicable to diverse controllers
  - Powerful enough to preclude a broad range of bugs

- Formula: $\texttt{model} \models \forall d. \Box\texttt{desire}(d) \leadsto \Box\texttt{match}(d)$

- "If at some point the desired state stops changing, then the system state will eventually match the desired state, and always match it from then"

Reasoning about one step at a time

- P: the precondition for the controller to take one step
- Q: the postcondition after the controller takes one step
- $Step_c$: one step of the controller
- $Step_{any}$: one step of any component (including the controller) in the cluster

$$\frac{\{P\}Step_c\{Q\} \quad \{P\}Step_{any}\{P \vee Q\} \quad WeakFair(Step_c)}{P \leadsto Q} \; WF1 \text{ rule}$$

If volume exists with v1, eventually volume exists with v2

Combining steps together

$$\dfrac{\dfrac{\cdots}{P \leadsto Q}\ WF1\text{ rule} \qquad \dfrac{\cdots}{Q \leadsto R}\ WF1\text{ rule}}{P \leadsto R}\ Transitivity\text{ rule}$$

Developers can build up the leads-to ($\leadsto$) chain in this way to prove that the controller eventually reaches the desired state step by step, regardless of all possible interactions.

Towards a truly reliable cloud infrastructure

It takes a village to do the research.

Ramnatthan Alagappan
Chaitanya Bhandari
Tej Chajed
Aishwarya Ganesan
Michael Gasch
Jiawei Tyler Gu
Indranil Gupta
Jon Howell
Shuyang Ji
Yuxuan Jiang
Anna Karanika
Andrea Lattuada

Owolabi Legunsen
Wenqing Luo
Wenjie Ma
Zicheng Ma
Oded Padon
Lalith Suresh
Adriana Szekeres
Lilia Tang
Mandana Vaziri
Chen Wang
Wentao Zhang
Yongle Zhang

Reference

[1] Fail through the Cracks: Cross-System Interaction Failures in Modern Cloud Systems, EuroSys, 2023. [paper] [dataset]

[2] Reasoning about modern datacenter infrastructures using partial histories, HotOS, 2023 [paper]

[3] Automatic Reliability Testing for Cluster Management Controllers, OSDI, 2022. [paper] [project]

[4] Acto: Automatic End-to-End Testing for Operation Correctness of Cloud System Management, SOSP, 2023. [paper] [project]

[5] Anvil: Verifying Liveness of Cluster Management Controllers, OSDI, 2024. (to appear)