Kubernetes reliability and an avenue to reliable cloud infrastructures

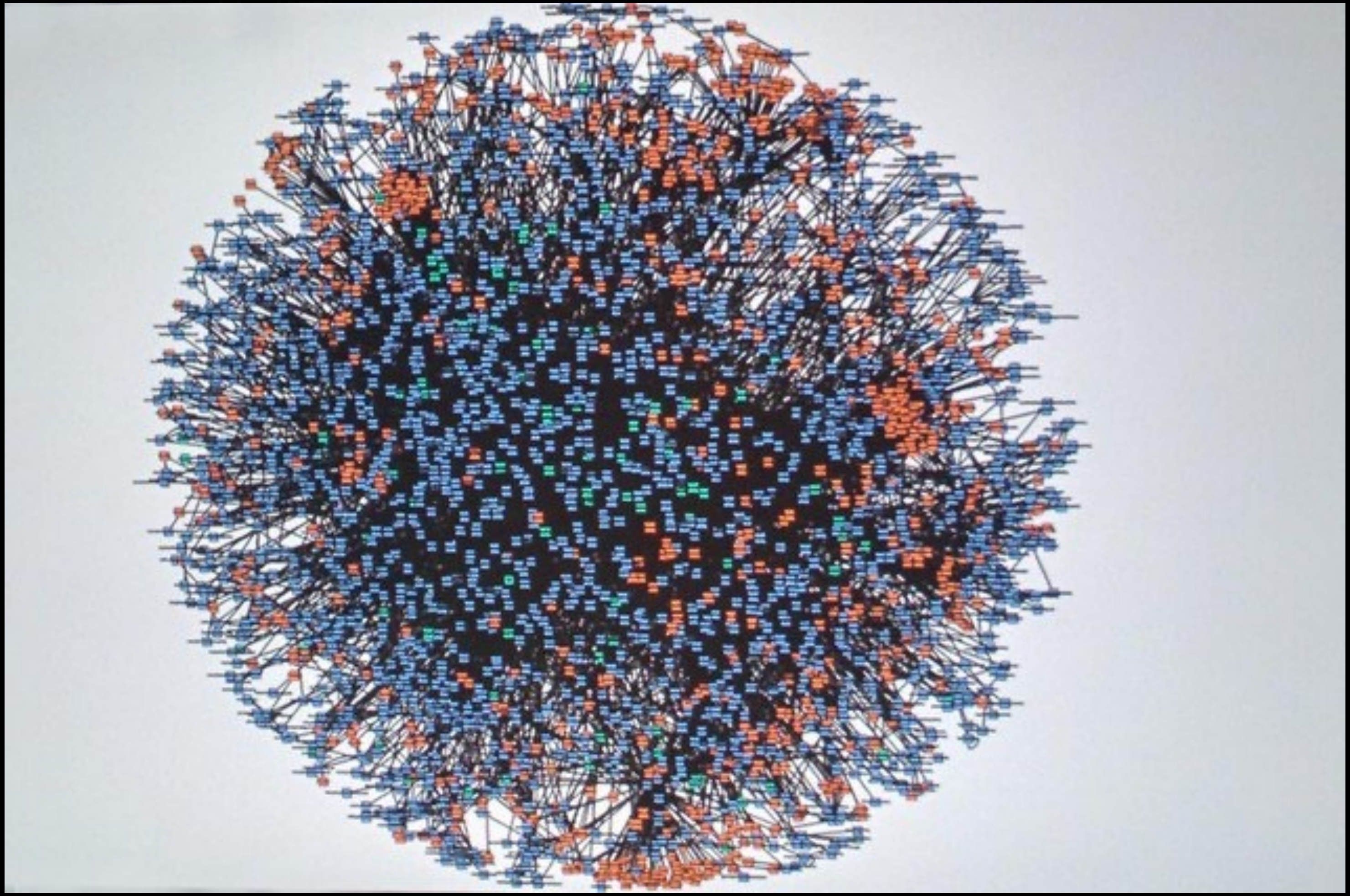
Tianyin Xu
University of Illinois Urbana-Champaign
tyxu@illinois.edu

IEEE Emerging Technology Reliability Roundtable
Lisbon, Portugal, May 2024

$ whoami

- Assistant Professor of Computer Science, UIUC (2018 – now)
  - Research Interests: Software and Systems Reliability
- Visiting Scientist, Facebook Core Systems (2017–2018)
  - Dealing with datacenter-level disasters
- PhD from UC San Diego (2011 – 2017)
  - Wrote a thesis about defending against configuration errors

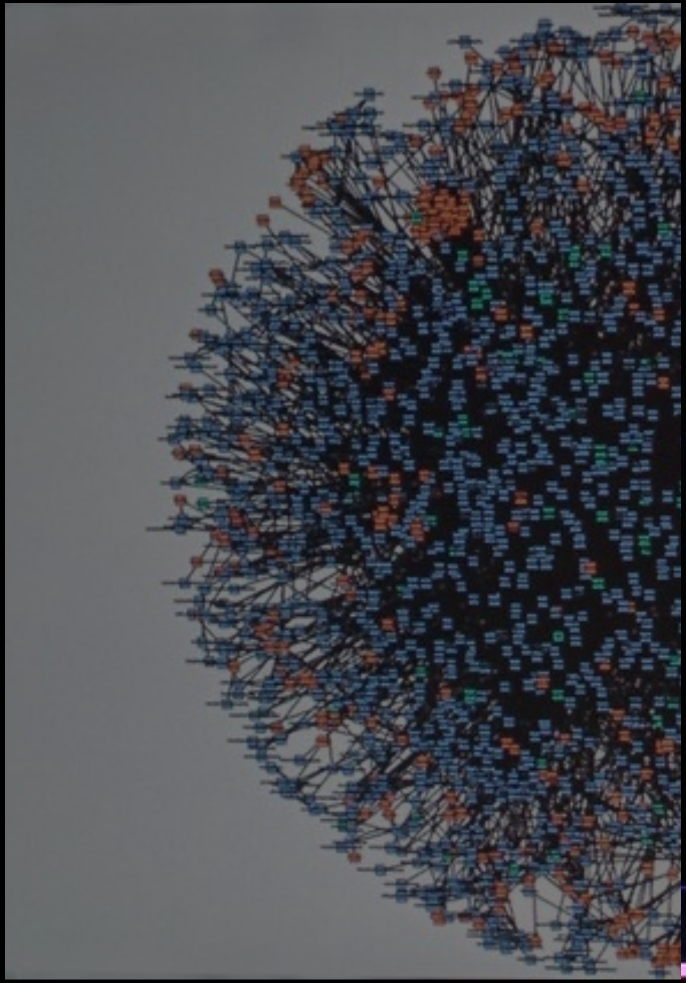
Emerging cloud computing paradigms



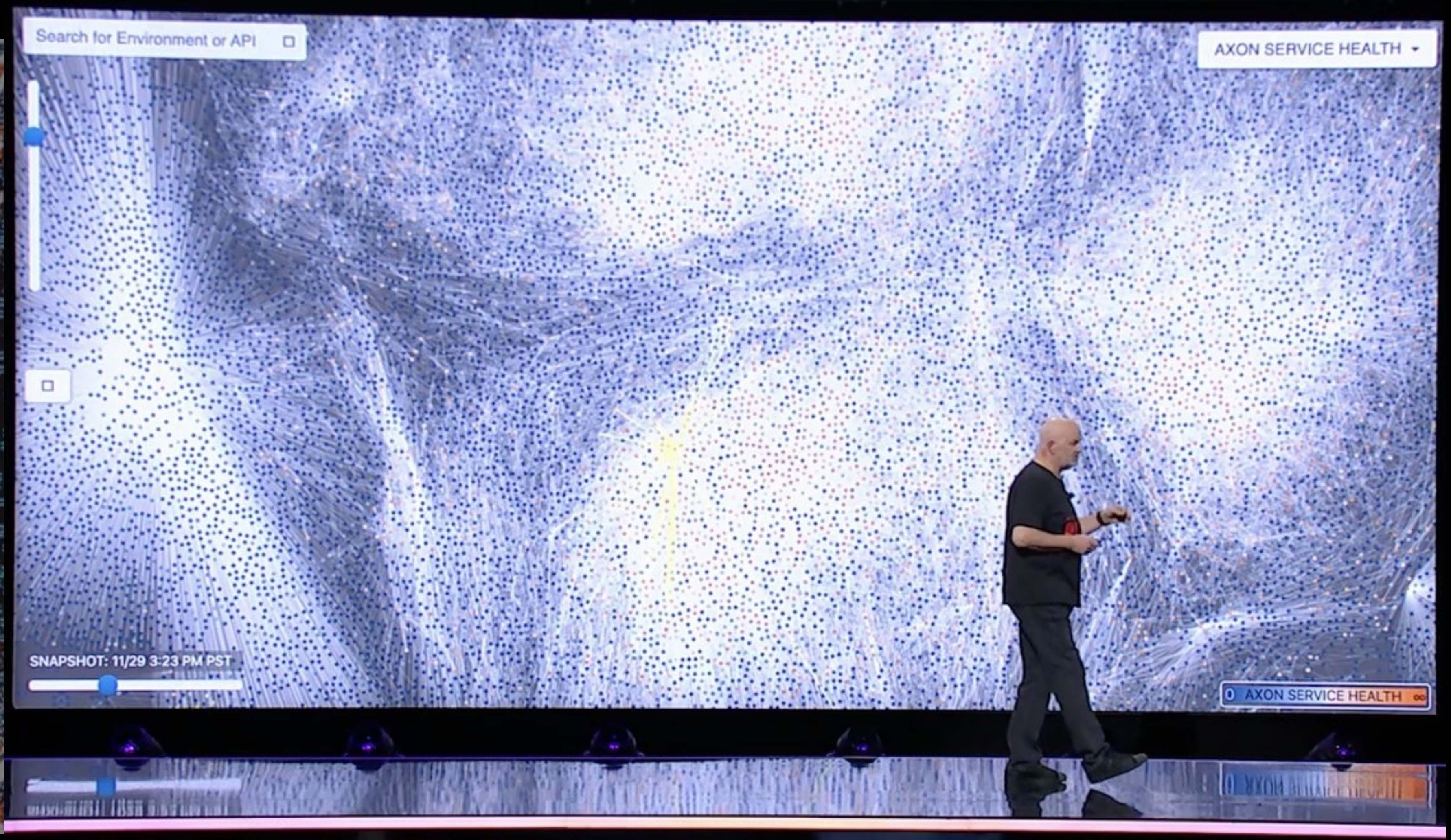
Real-time graph of microservice dependencies at amazon.com in 2008.

- 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.

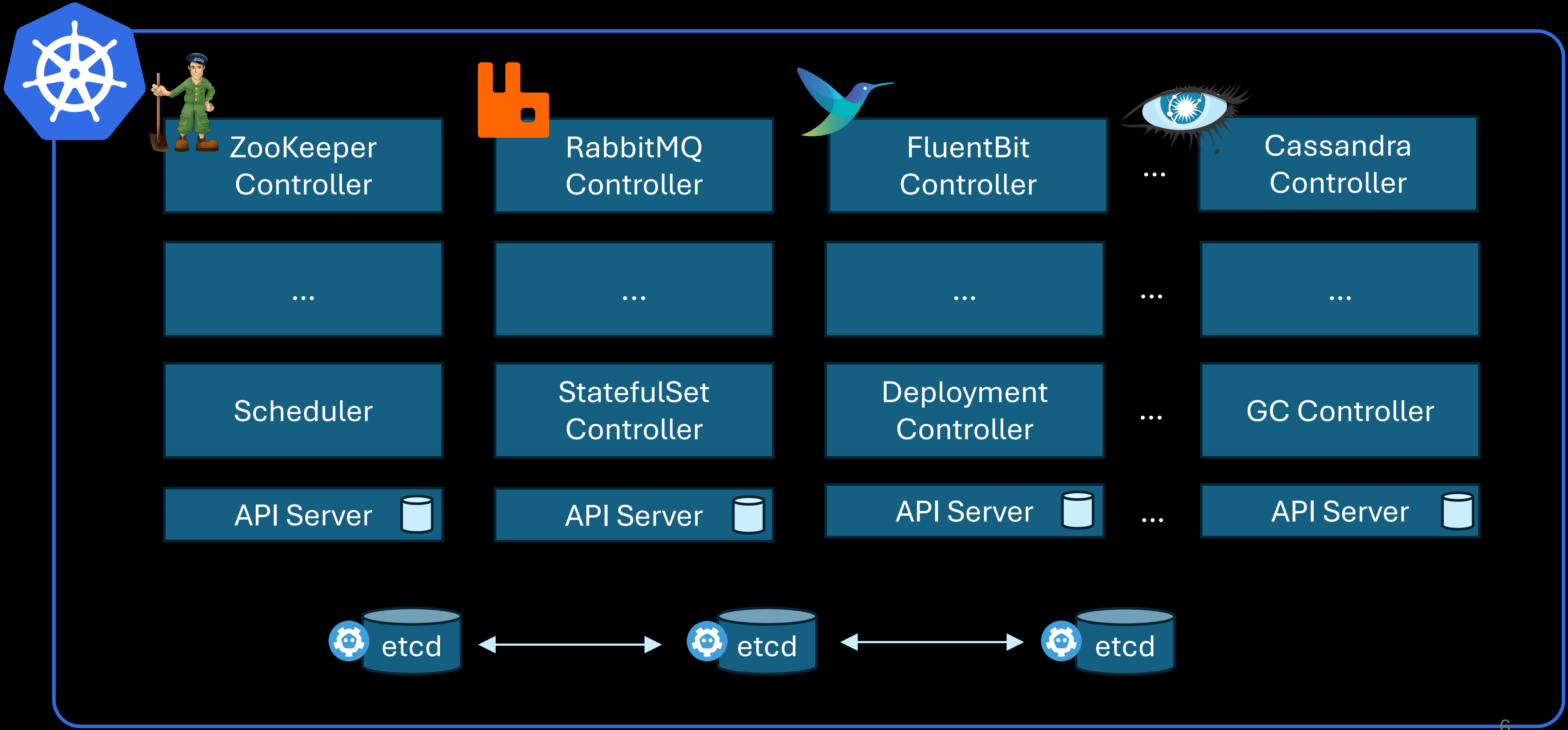
Observations

- Individual services become simpler and more fine-grained
  - Opportunities for testing, analysis, and verification
- Cross-service 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

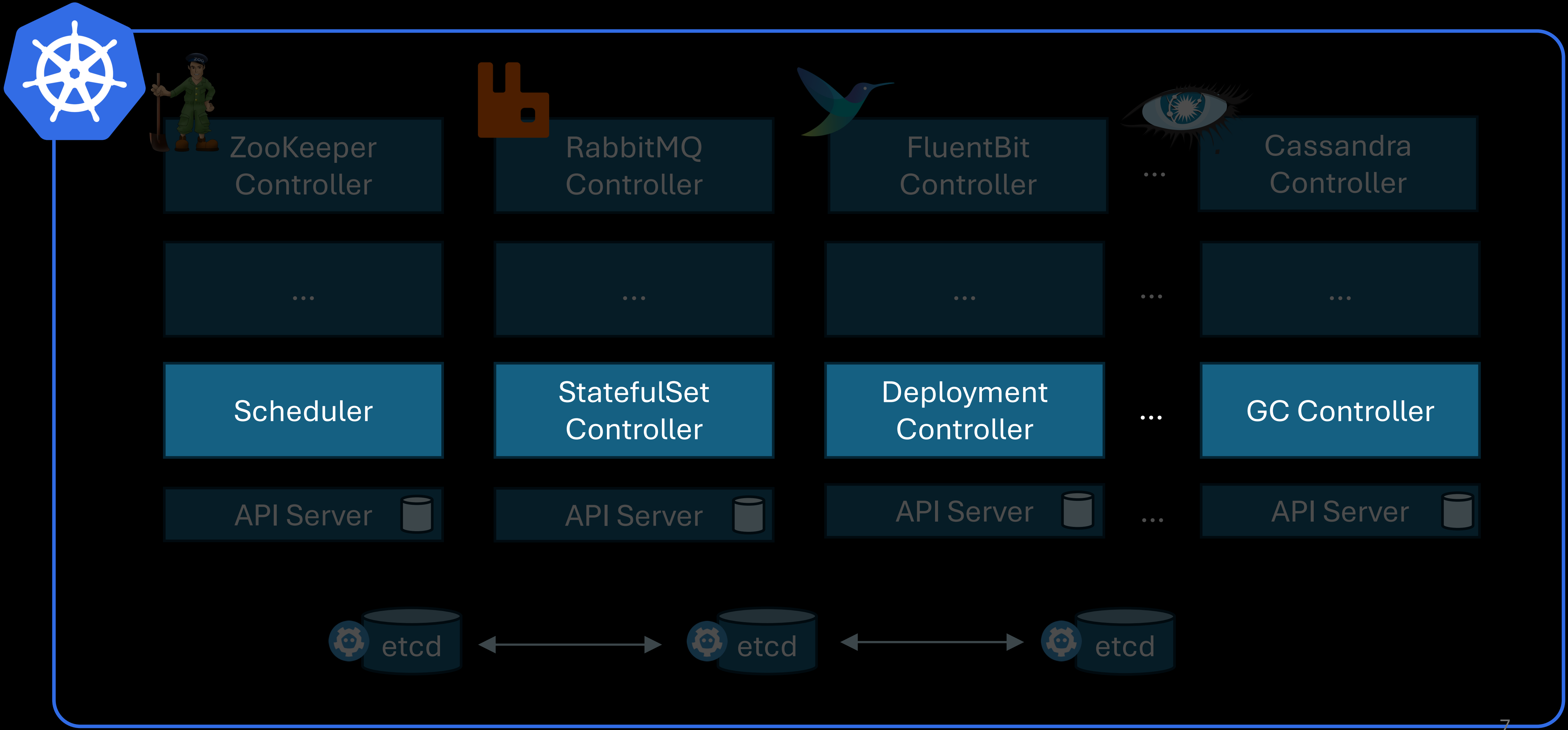
Testing and verification of cloud systems

- Testing and model checking existing cloud systems
    - Finding and fixing bugs in systems code
    - Specifying systems (e.g., using TLA+)
- Building formally verified systems with correctness guarantees
    - Proved safety and liveness

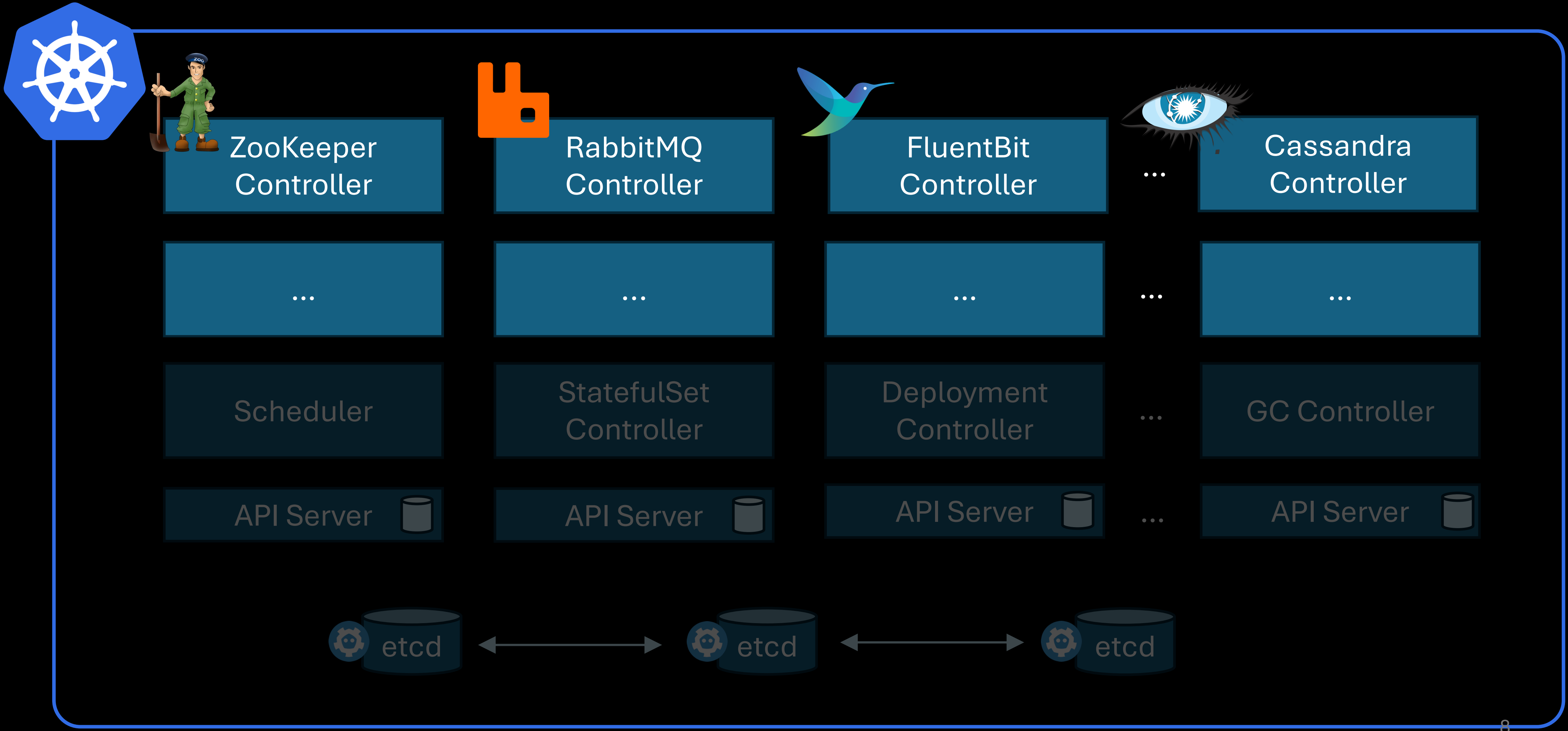
Kubernetes as a running (microservice) system



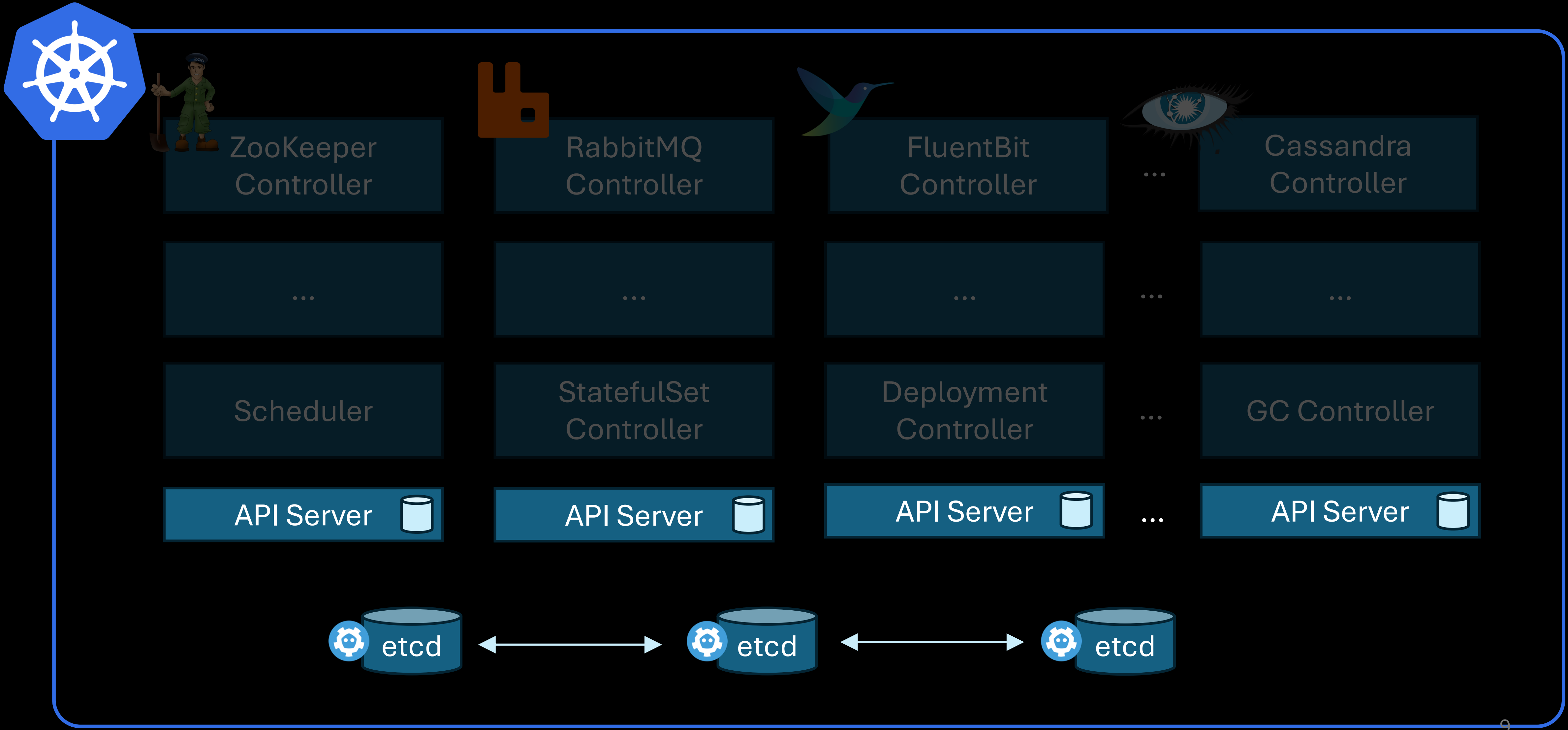
Kubernetes as a running (microservice) system



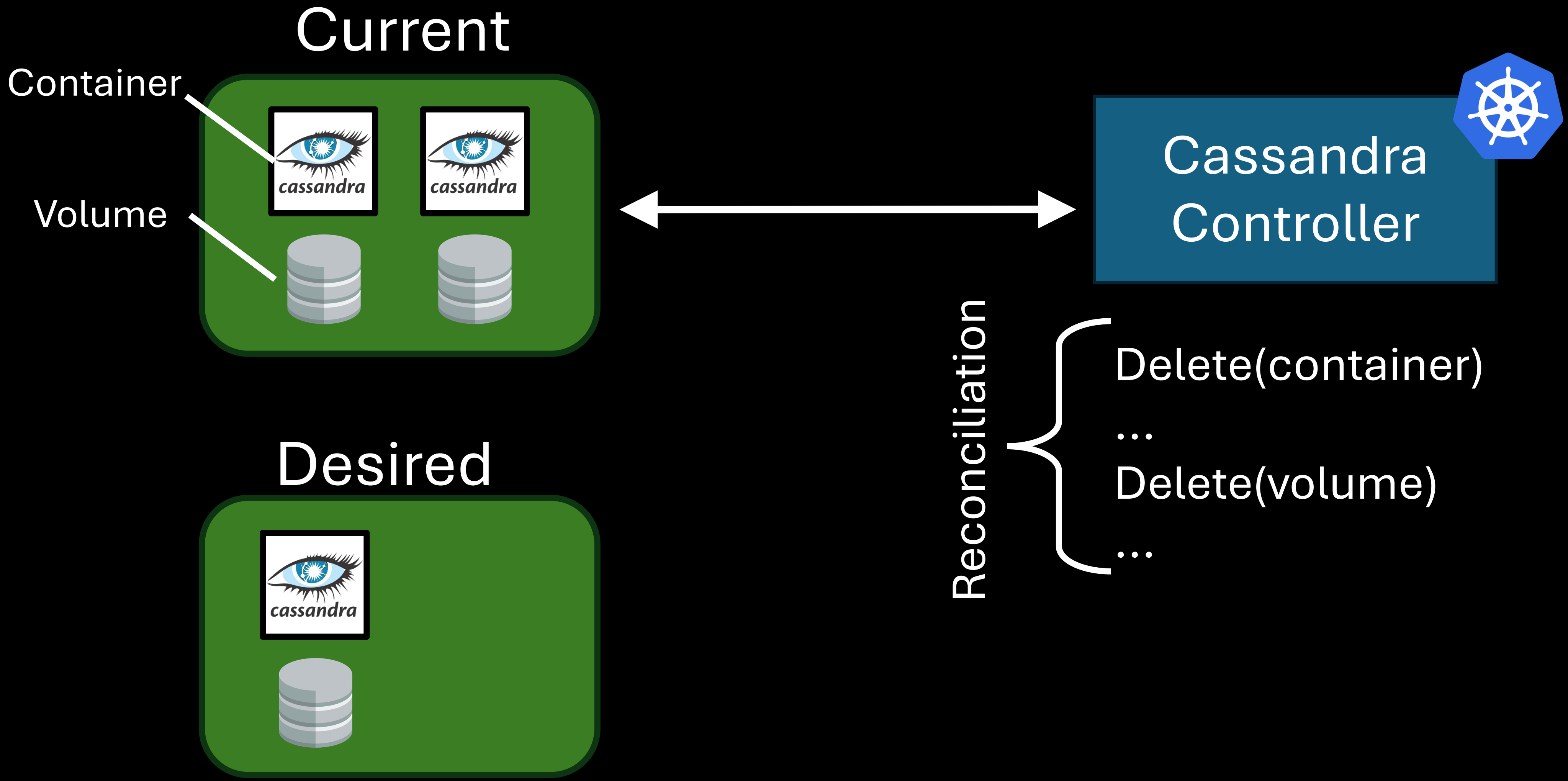
Kubernetes as a running (microservice) system



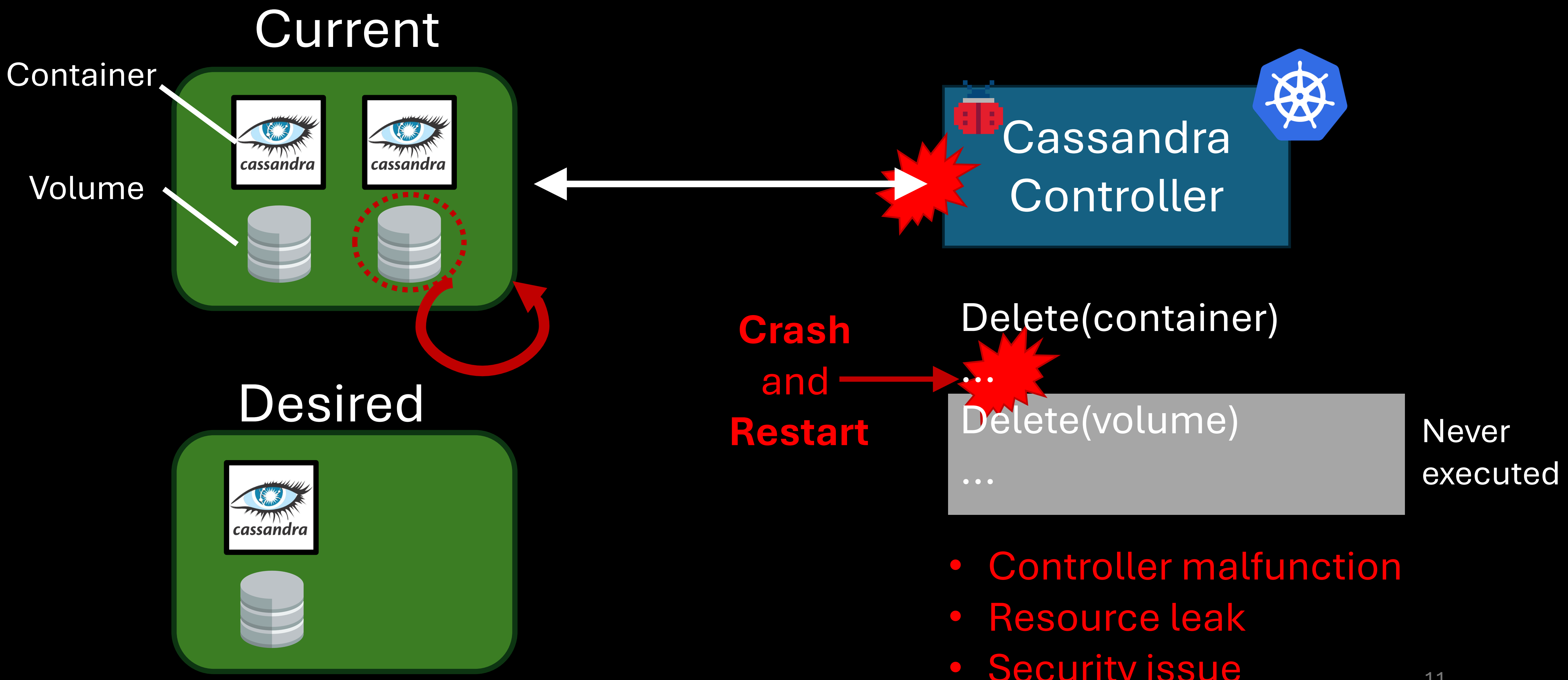
Kubernetes as a running (microservice) system



Challenge 1: Faults, delays, and asynchrony



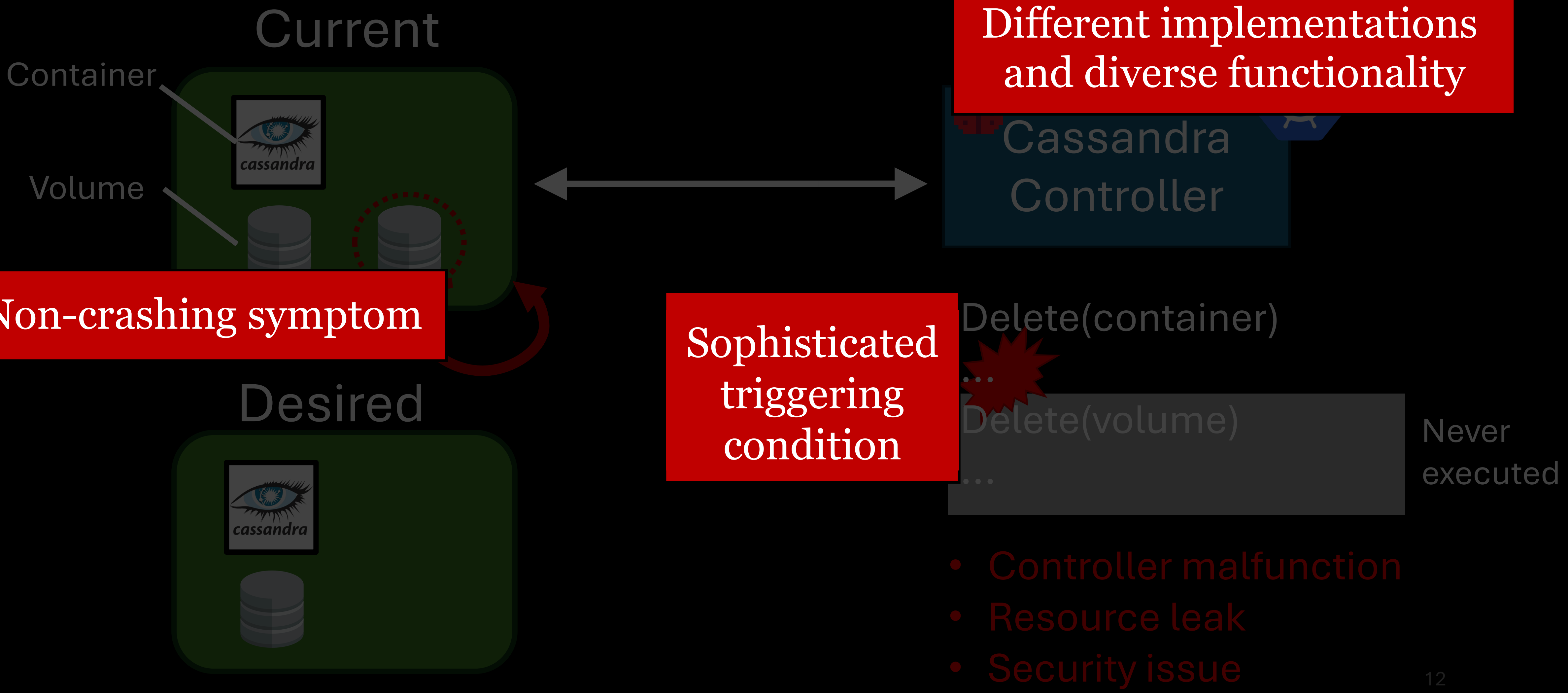
Challenge 1: Faults, delays, and asynchrony



```
Delete(container)
...
Delete(volume)
...
```

- Controller malfunction
- Resource leak
- Security issue

Challenge 1: Faults, delays, and asynchrony



Different implementations and diverse functionality

Non-crashing symptom

Sophisticated triggering condition

- Controller malfunction
- Resource leak
- Security issue

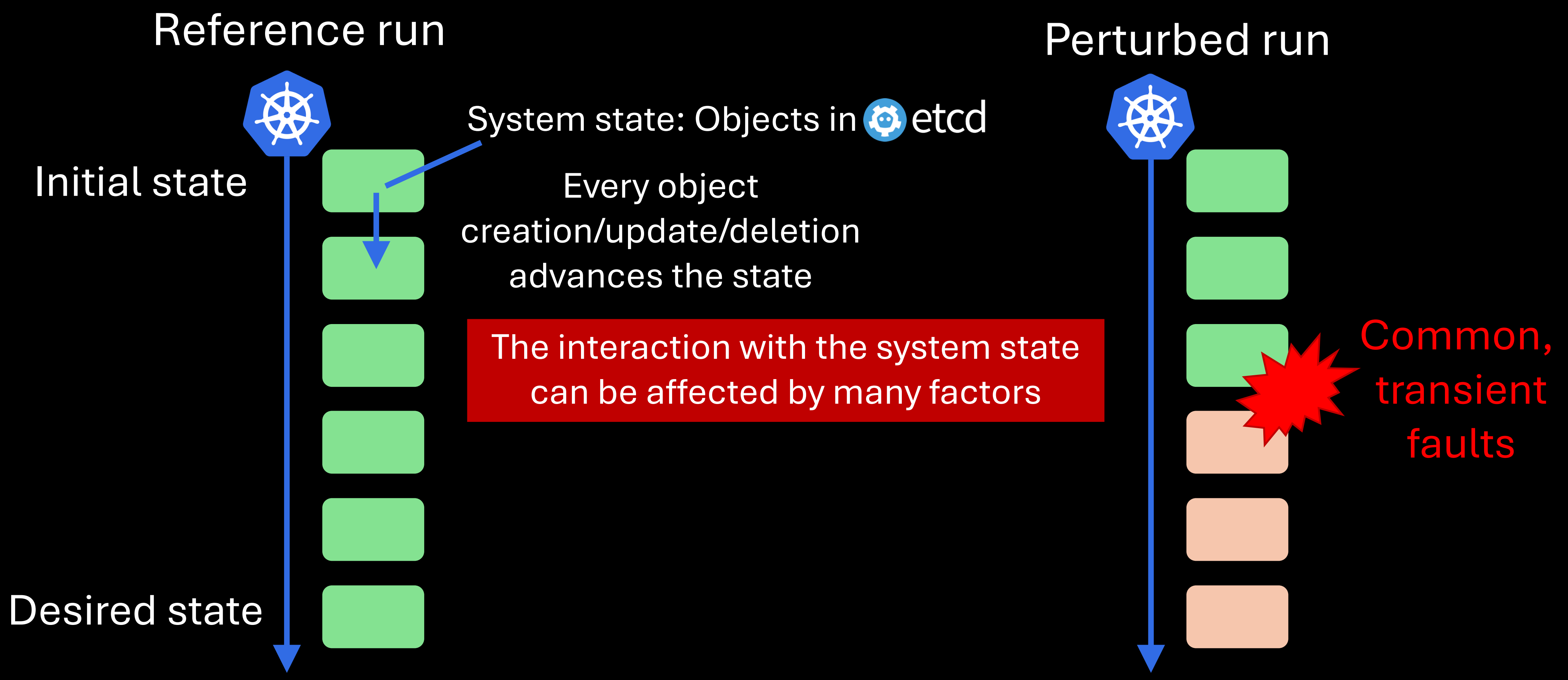
Sieve for automatic reliability testing

- 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

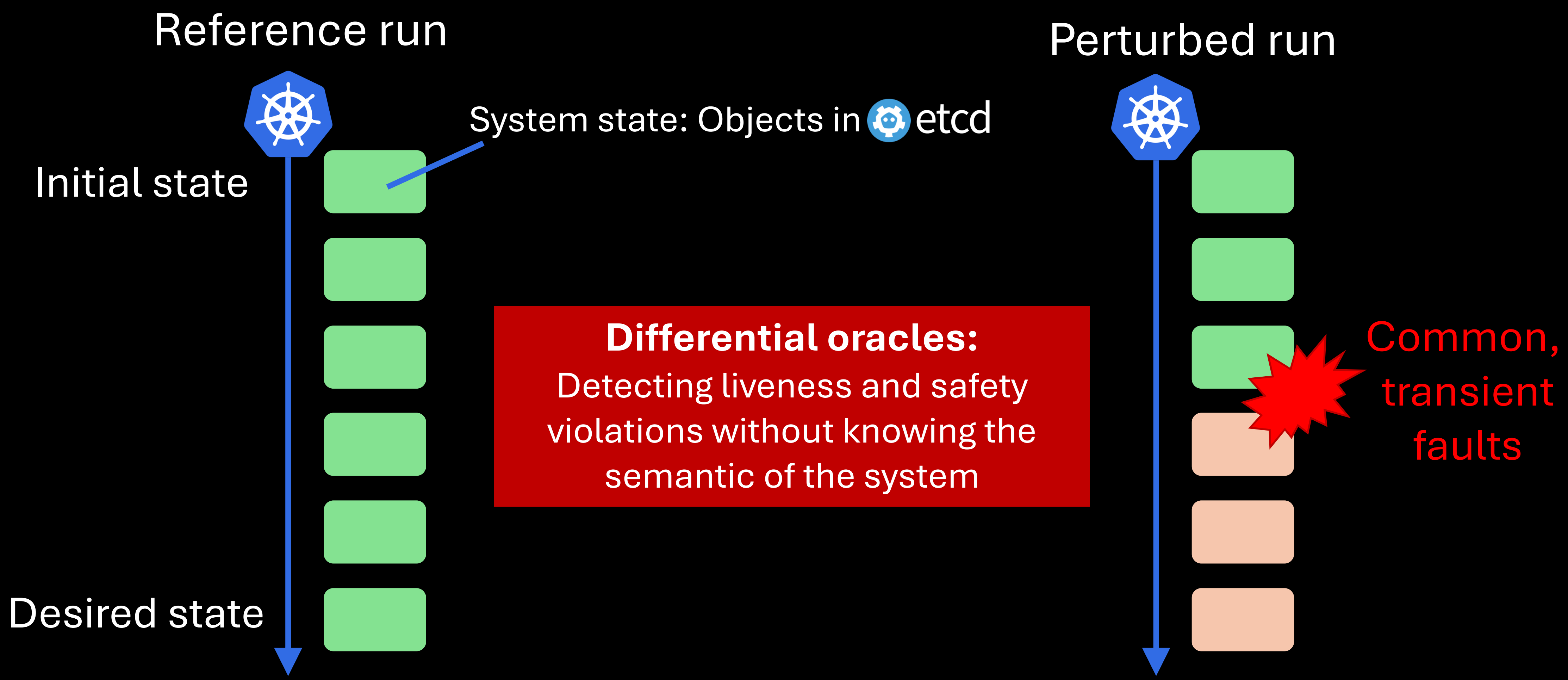
- Available: https://github.com/sieve-project/sieve

Perturbing the controller’s view of system states


Reference run
Perturbed run
System state: Objects in etcd
Initial state
Desired state
Every object creation/update/deletion advances the state
Common, transient faults

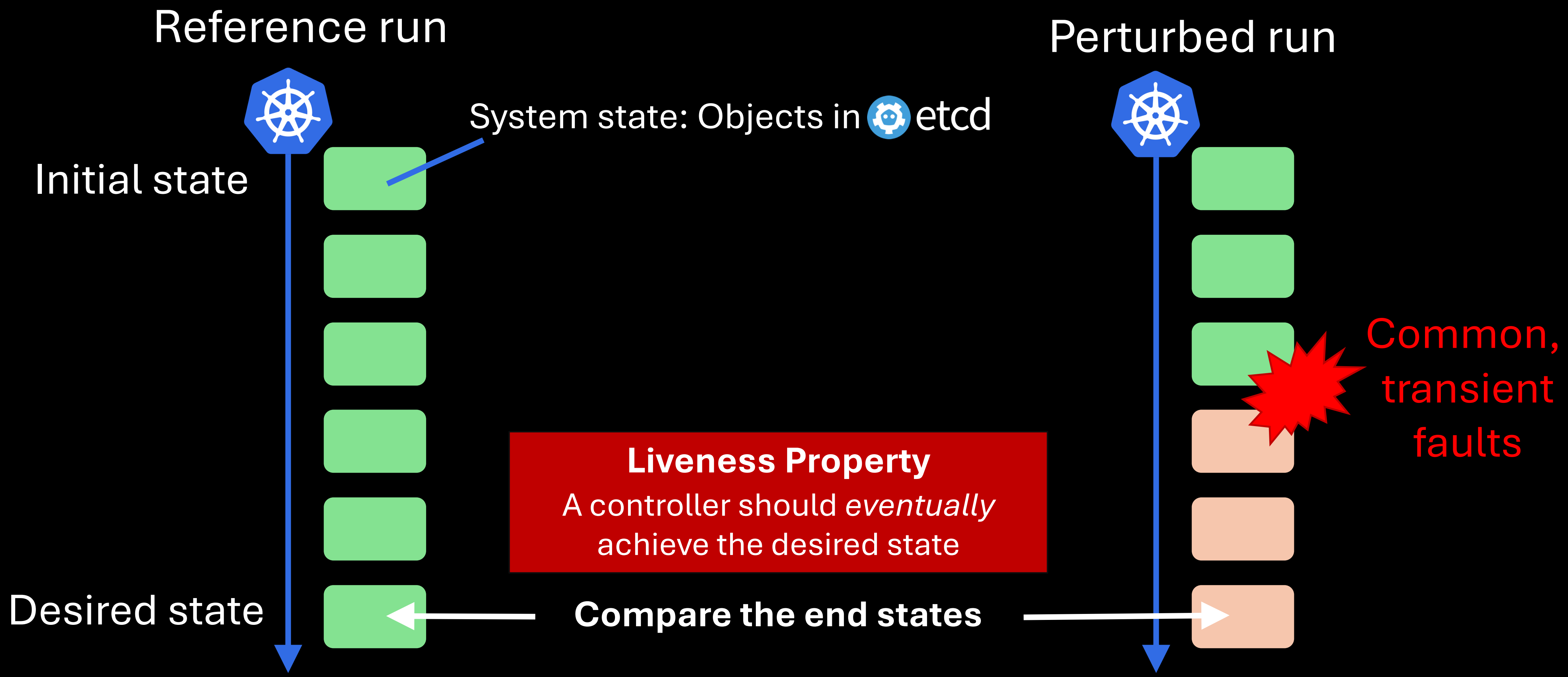
The interaction with the system state can be affected by many factors

Flagging buggy behavior with differential oracles



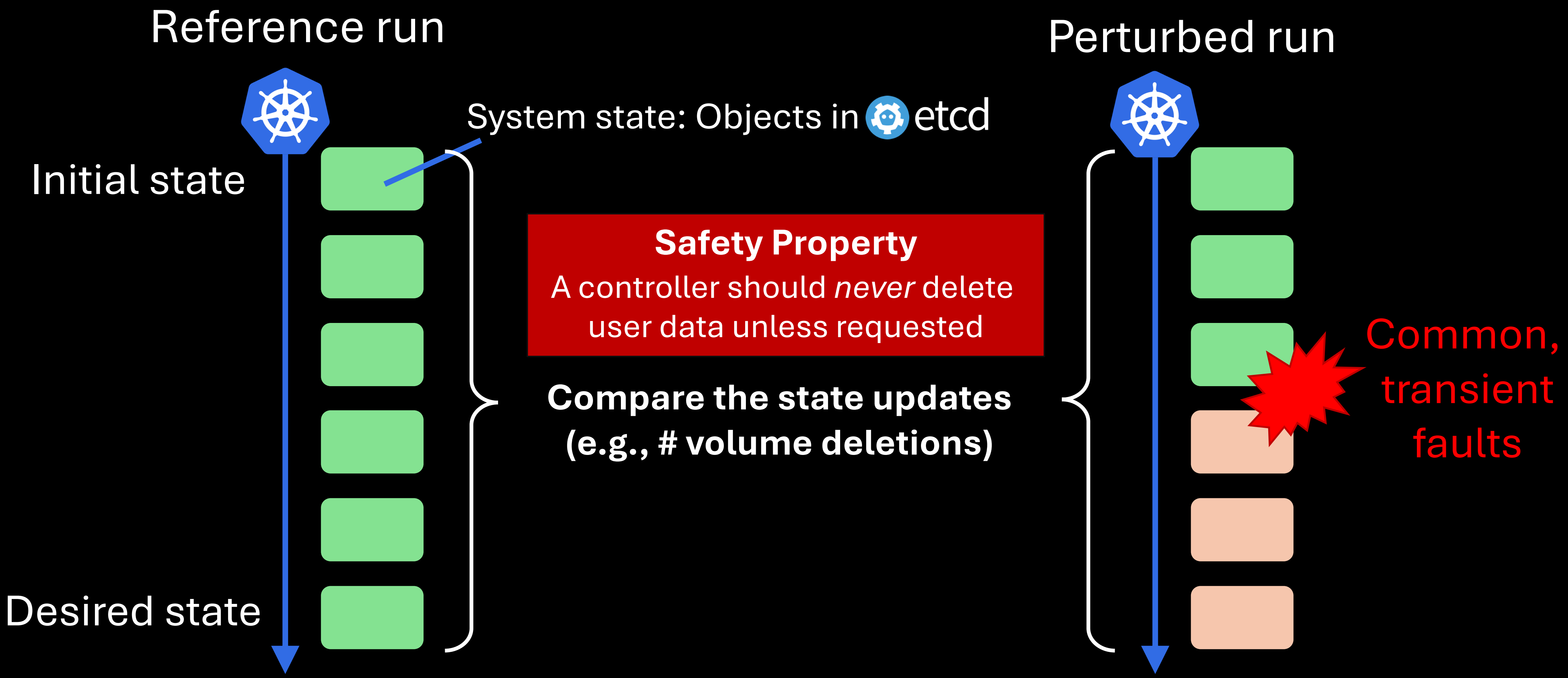
Differential oracles:
Detecting liveness and safety violations without knowing the semantic of the system

Flagging buggy behavior with differential oracles



Liveness Property
A controller should *eventually* achieve the desired state

Flagging buggy behavior with differential oracles



Safety Property
A controller should *never* delete user data unless requested

Compare the state updates (e.g., # volume deletions)

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

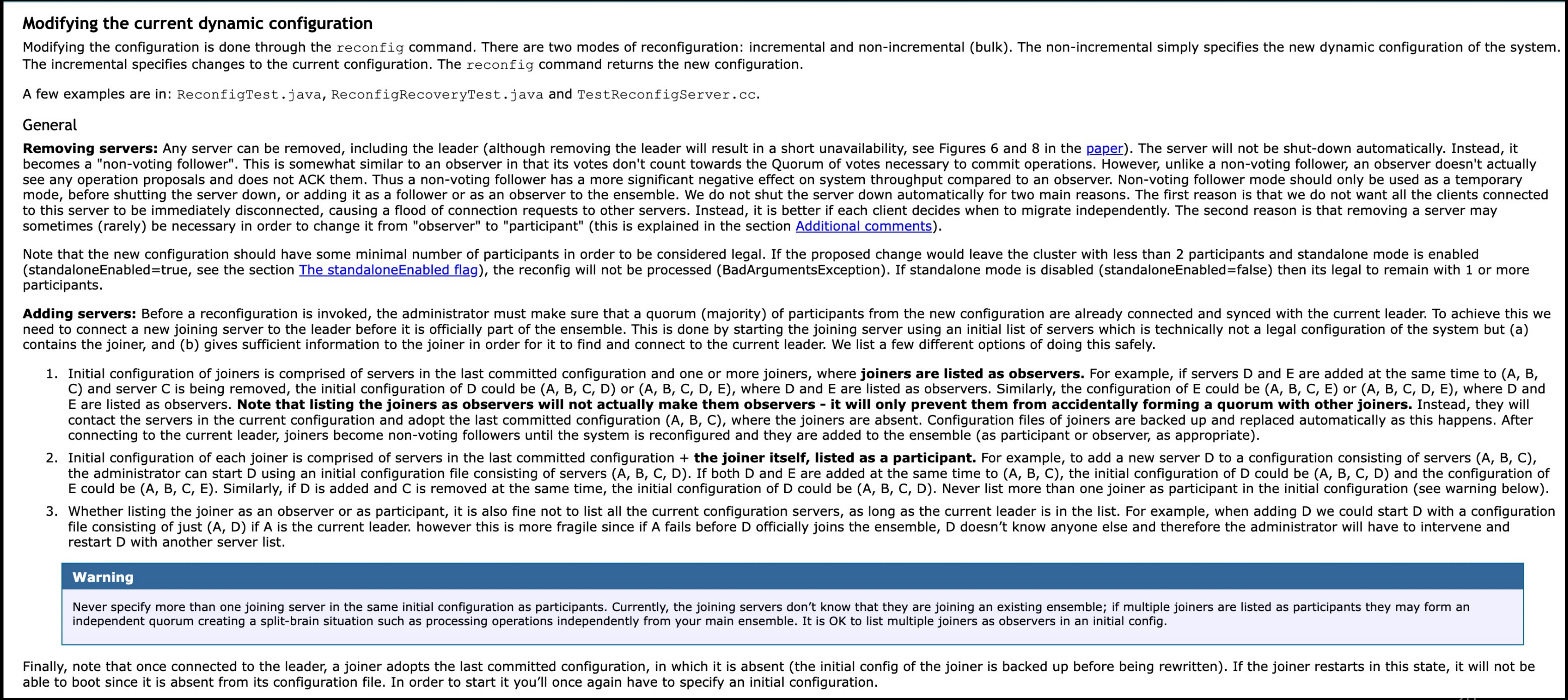
Challenge 2: Complex interface and configuration

- Hundreds to thousands of configuration parameters
  - “*cloud feels more about configuration management than software engineering*”

- High velocity (thousands) of configuration changes per day
  - outpacing source-code changes

- Fundamentally difficult to test all possible configurations
  - classic combinatorial complexity
  - Misconfigurations (the error space) are often not considered

Challenge 2: Complex interface and configuration



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.

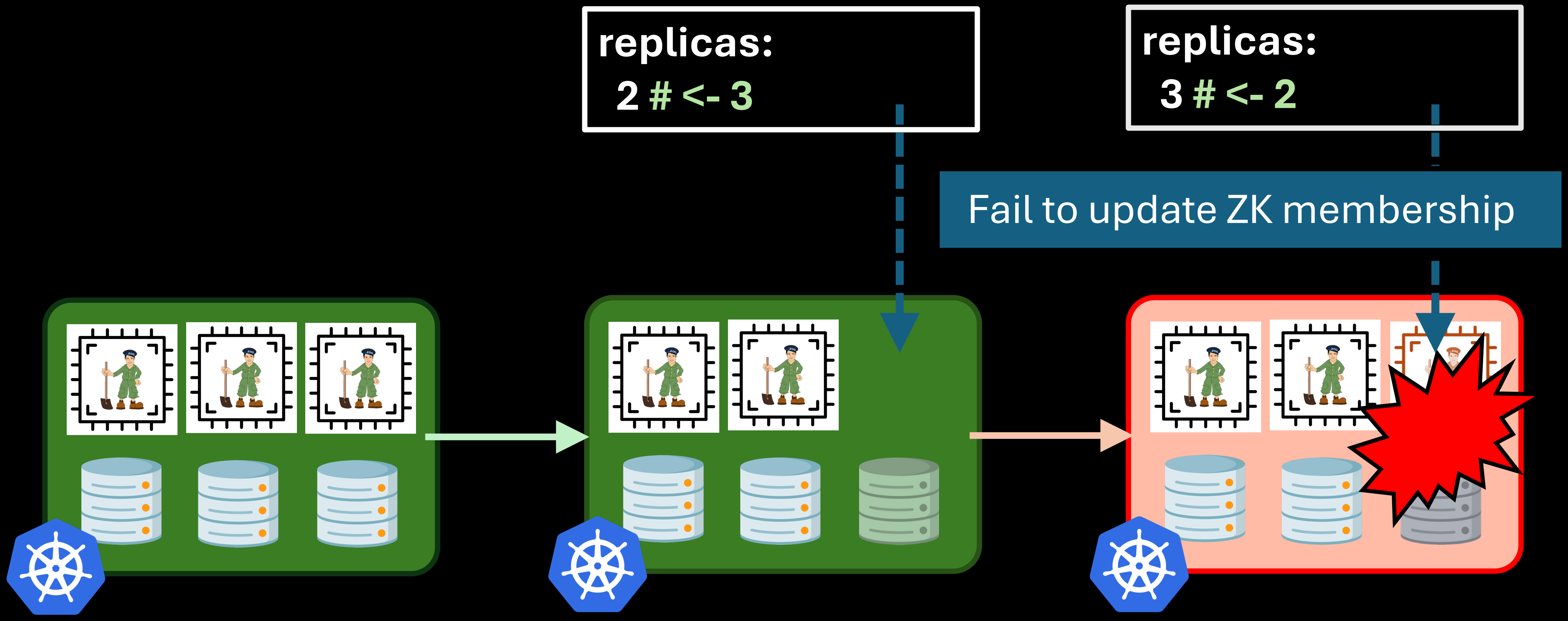
1. Initial configuration of joiners is comprised of servers in the last committed configuration and one or more joiners, where **joiners are listed as observers.** For example, if servers D and E are added at the same time to (A, B, C) and server C is being removed, the initial configuration of D could 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.

Challenge 2: Complex interface and configuration



Ctest: testing configuration changes with code

- Testing configuration changes *together with* code affected by the changes
  - can detect sophisticated misconfigurations (e.g., silent errors)
  - can detect dormant bugs triggered by *valid* configuration changes

- Configuration tests can be directly generated from existing tests
  - Mature software projects have abundant test suites
  - Reuse well engineered test logic and oracles

- Detected **96.9%** of real-world failure-inducing configuration changes

- Available: https://github.com/xlab-uiuc/openctest
  https://github.com/xlab-uiuc/ctest4j

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 **81** serious bugs in 12 popular Kubernetes controllers
  - 62 confirmed and 43 fixed

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

Can we build formally verified controllers that are practical?

Anvil: building formally verified controllers

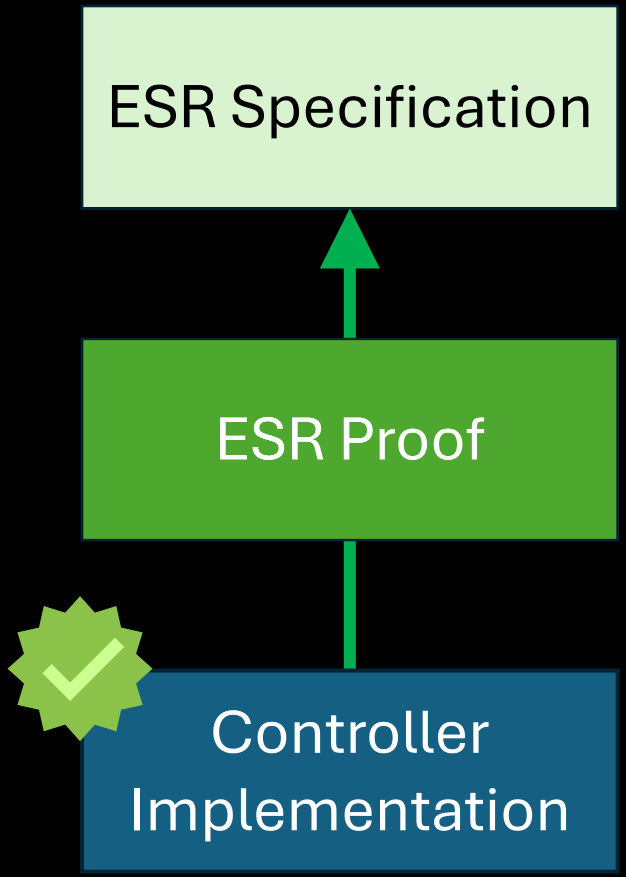
- A framework to help build practical and verified controllers
  - **Verified**: the controller implementation is formally verified
  - **Practical**: the verified controller can be deployed in any Kubernetes clusters

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

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"

Developing controllers with Anvil



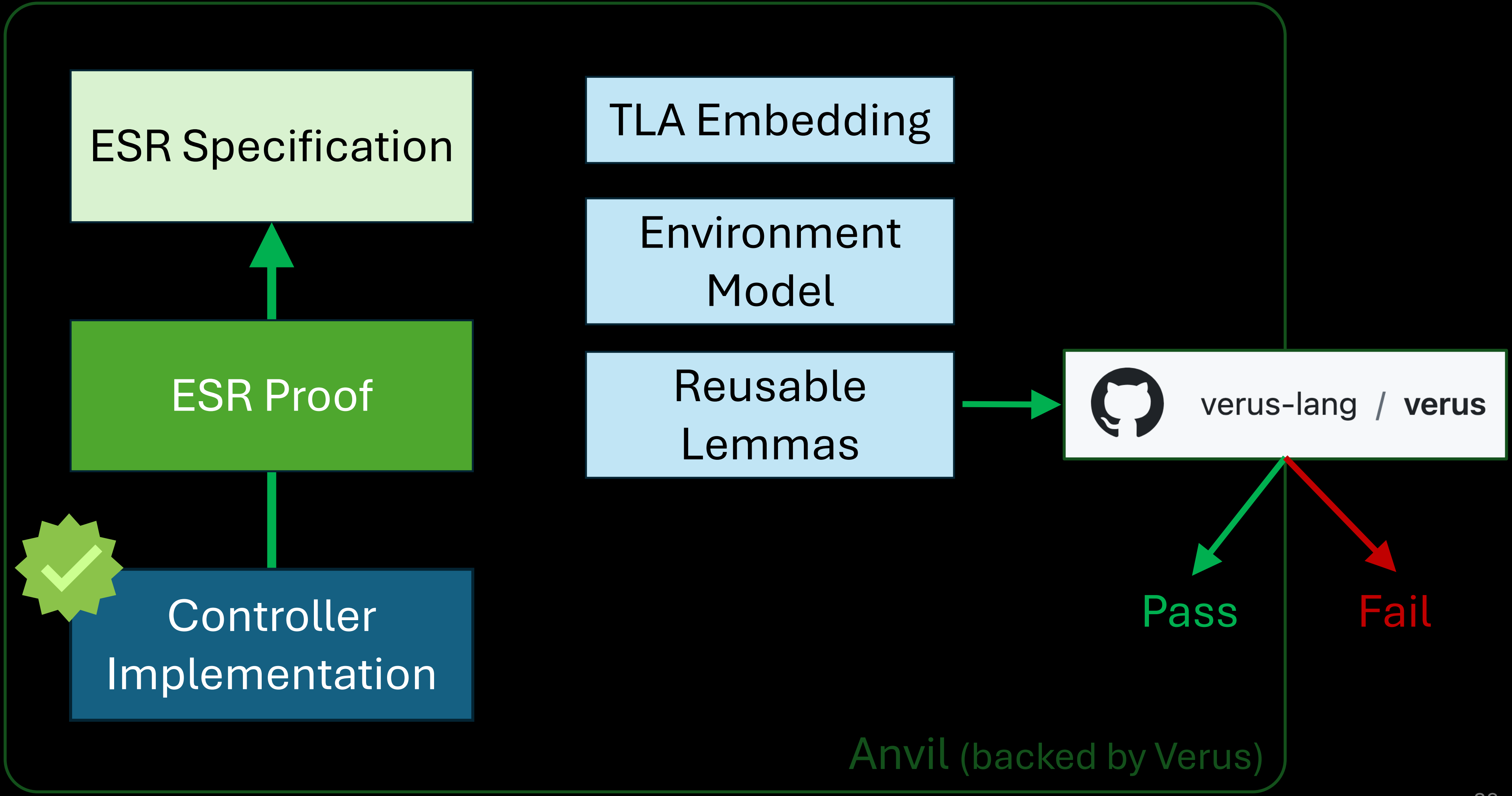
Also supports other properties: e.g., safety

Checked by the SMT solver

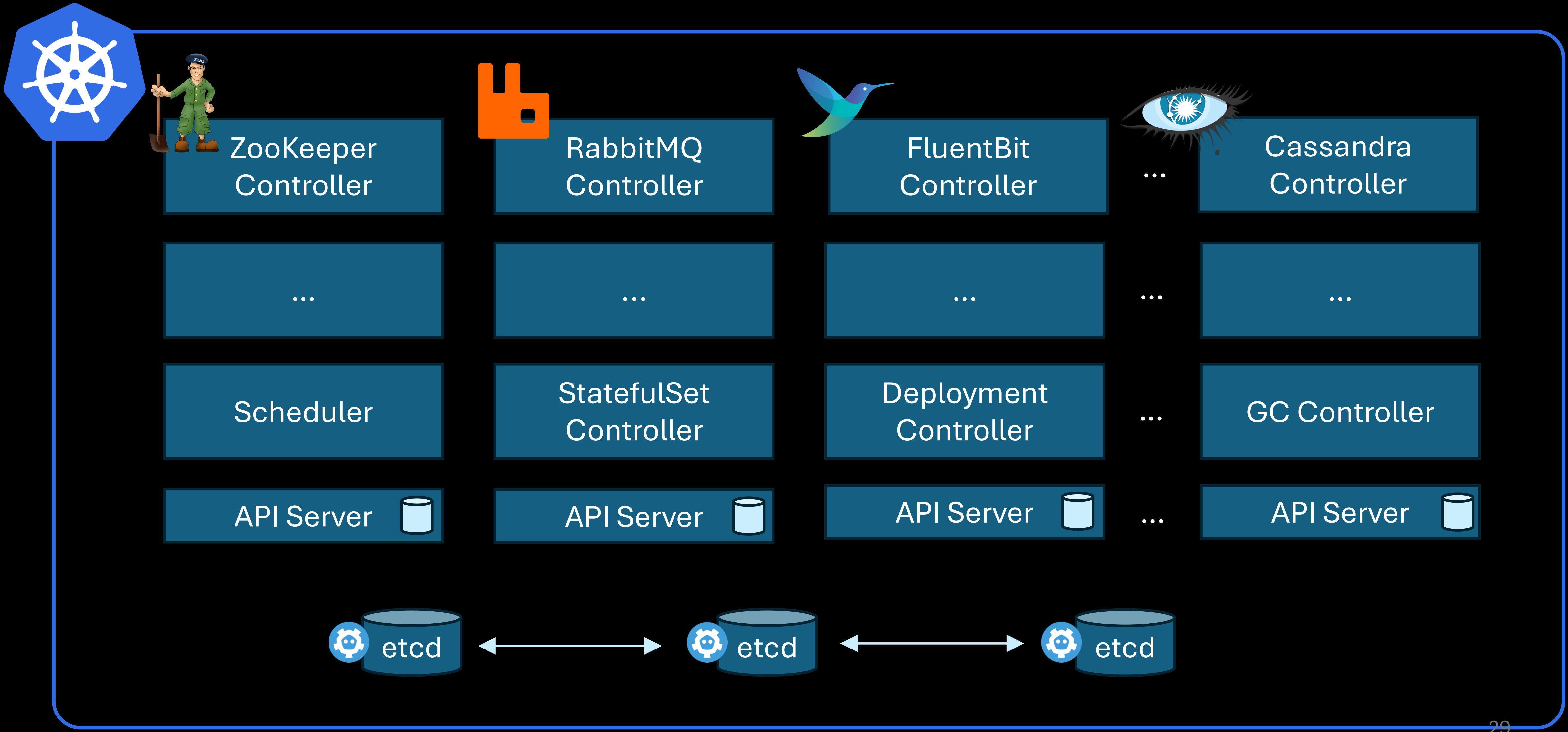
Written in Rust
Deployed in Kubernetes

Anvil (backed by Verus)

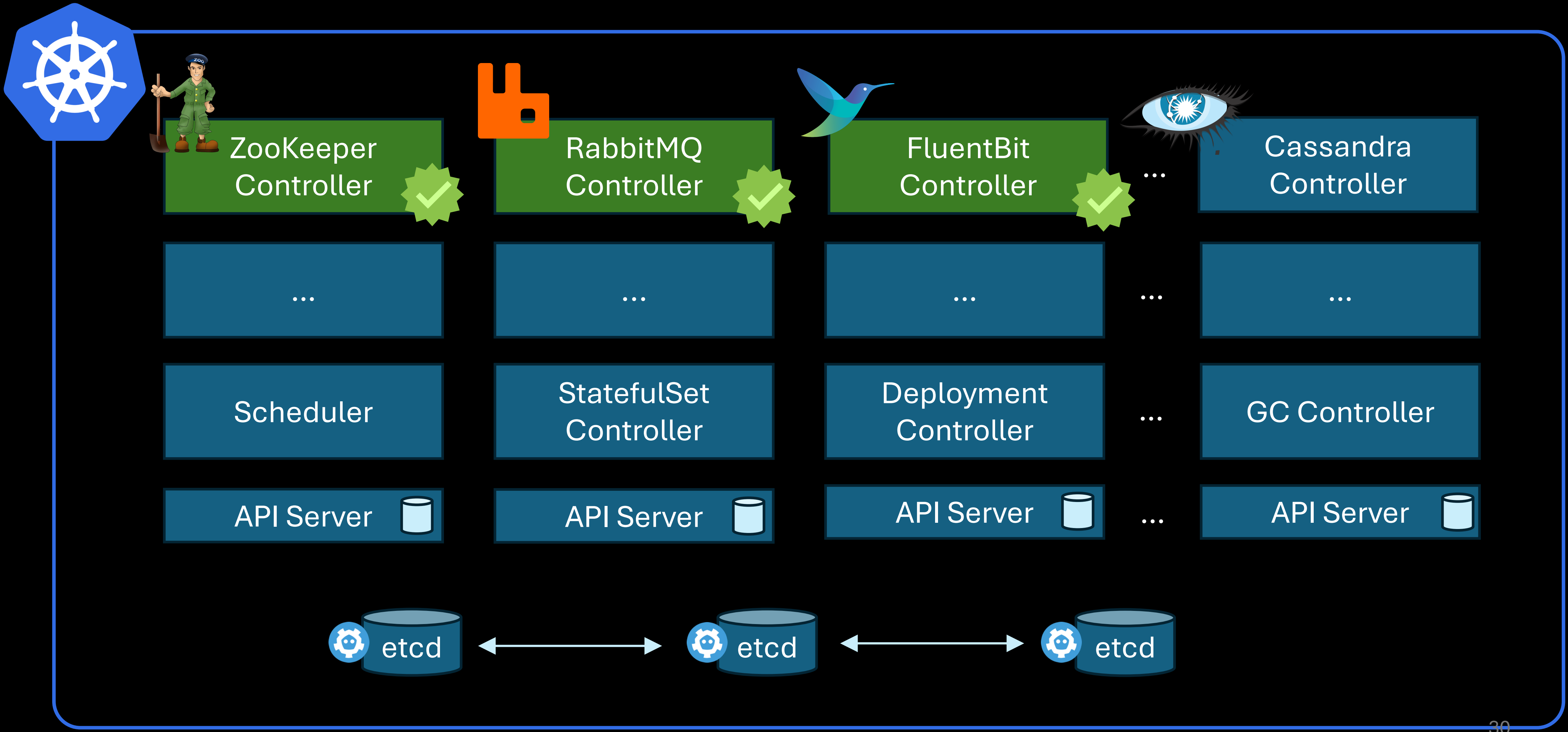
Developing controllers with Anvil



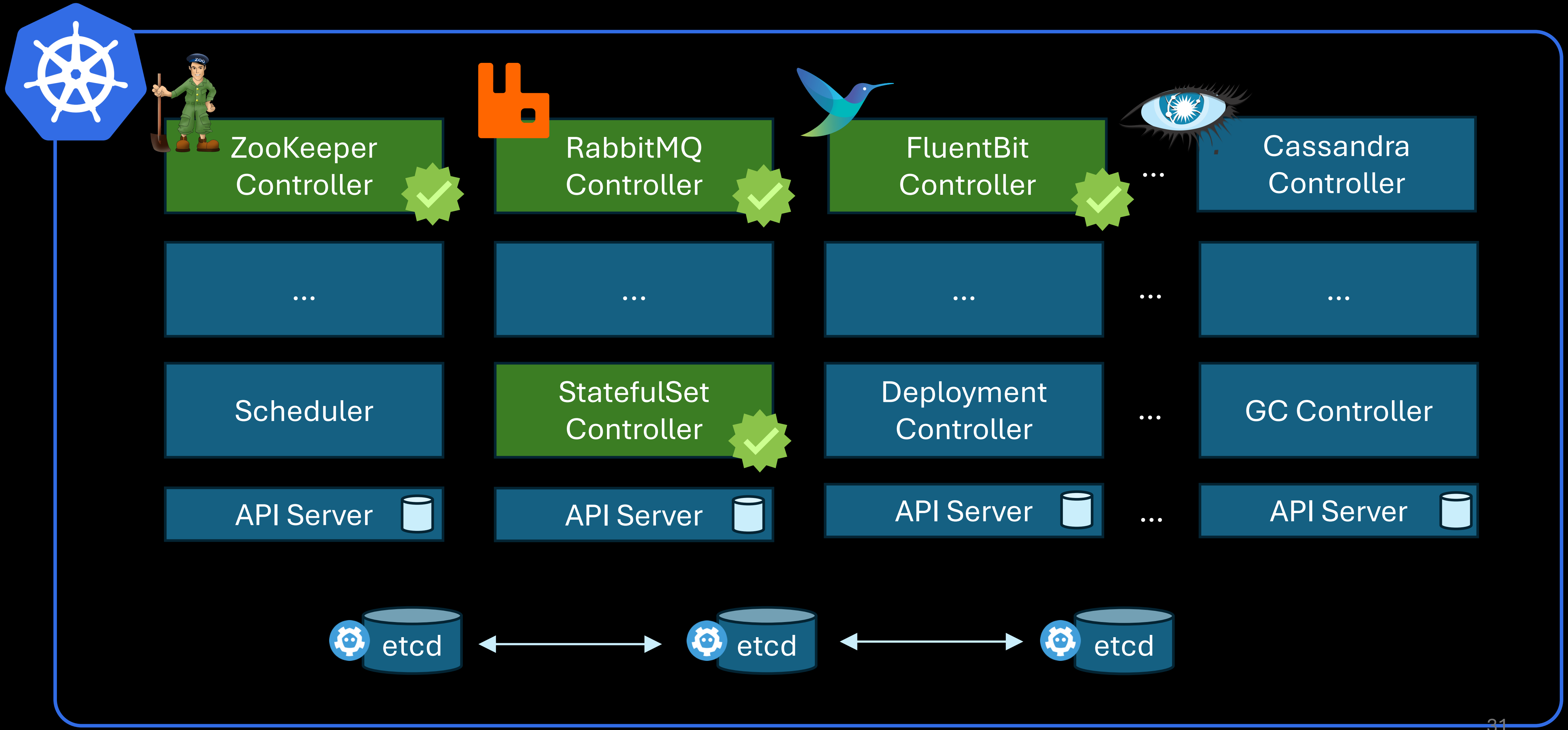
Towards truly reliable cloud infrastructures



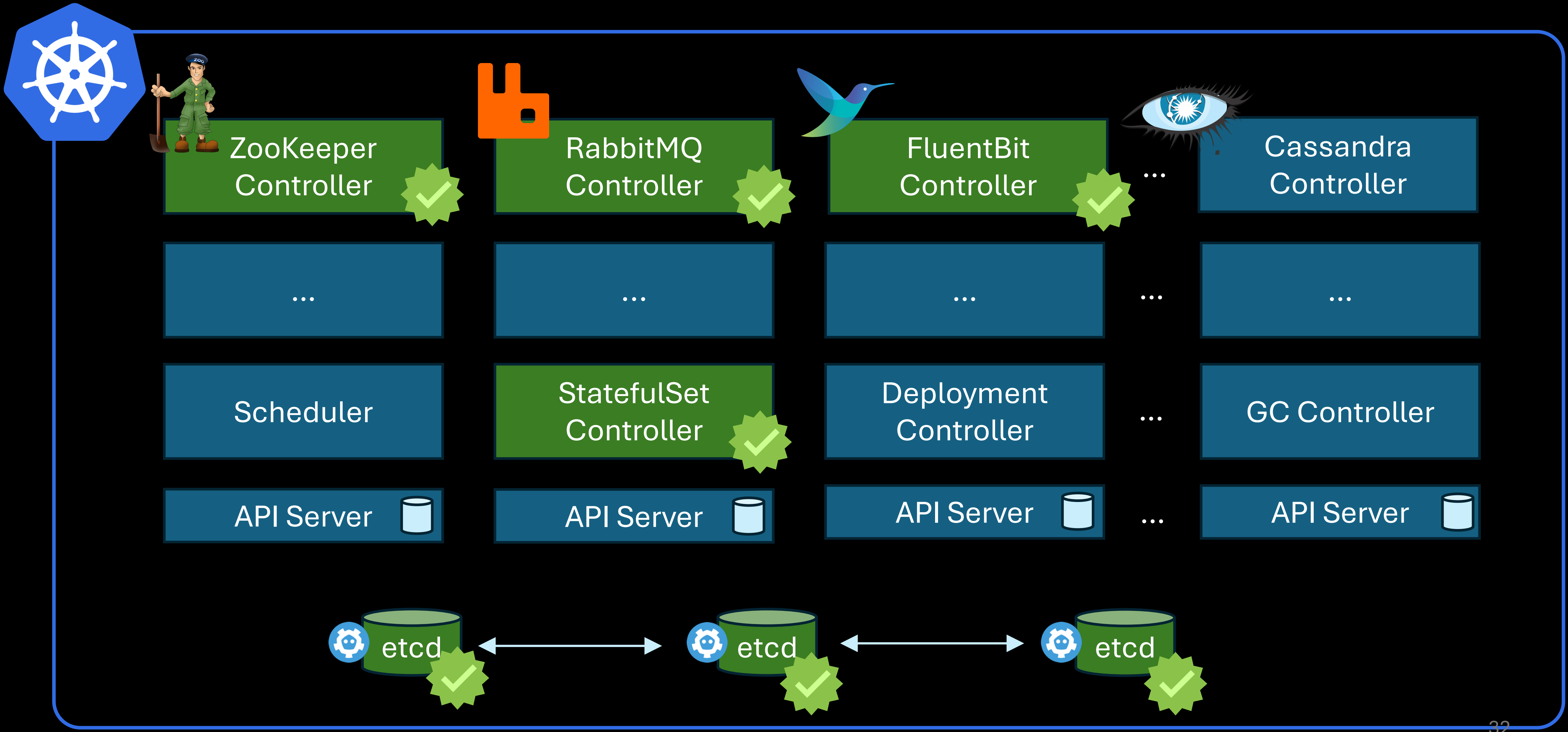
Towards truly reliable cloud infrastructures



Towards truly reliable cloud infrastructures



Towards truly reliable cloud infrastructures



Reference

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

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

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

[4] Anvil: Verifying Liveness of Cluster Management Controllers, OSDI, 2024. [paper] [project]

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

Reference

[6] Testing Configuration Changes in Context to Prevent Production Failures, OSDI, 2020. [paper] [project]

[7] Push-Button Reliability Testing for Cloud-Backed Applications with Rainmaker, NSDI, 2022 [paper]

[8] Early Detection of Configuration Errors to Reduce Failure Damage, 2016. [paper]

[9] Do Not Blame Users for Misconfigurations, SOSP, 2013. [paper]