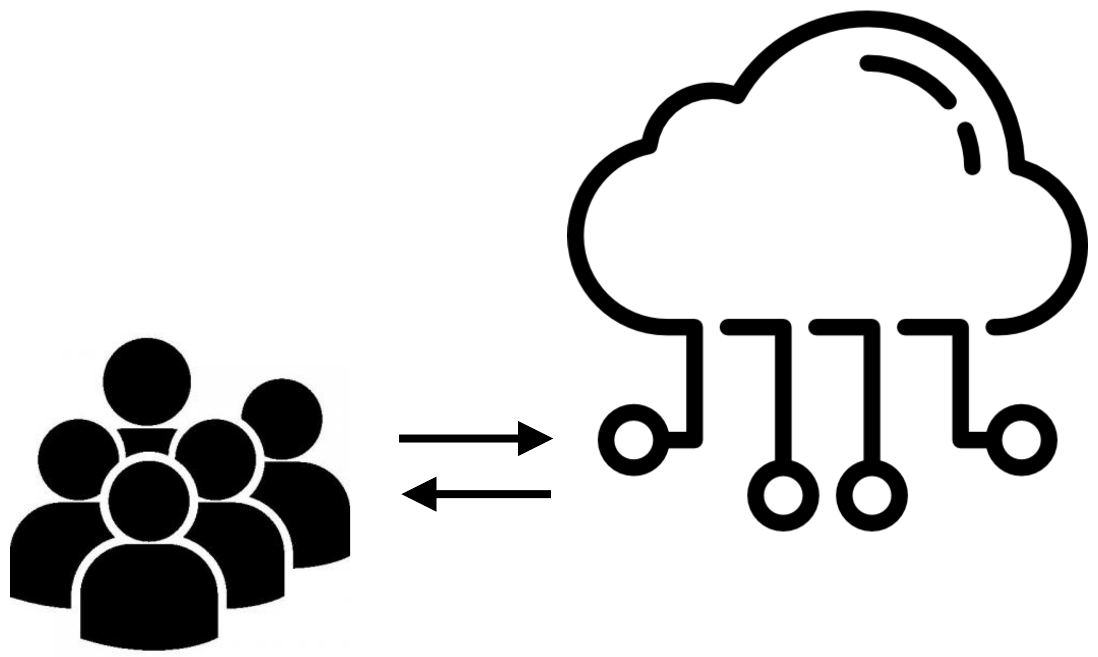
Demystifying and Checking Silent Semantic Violations in Large Distributed Systems

Chang Lou, Yuzhuo Jing, Peng Huang

JOHNS HOPKINS UNIVERSITY

OSDI 2022

Distributed systems provide rich semantics

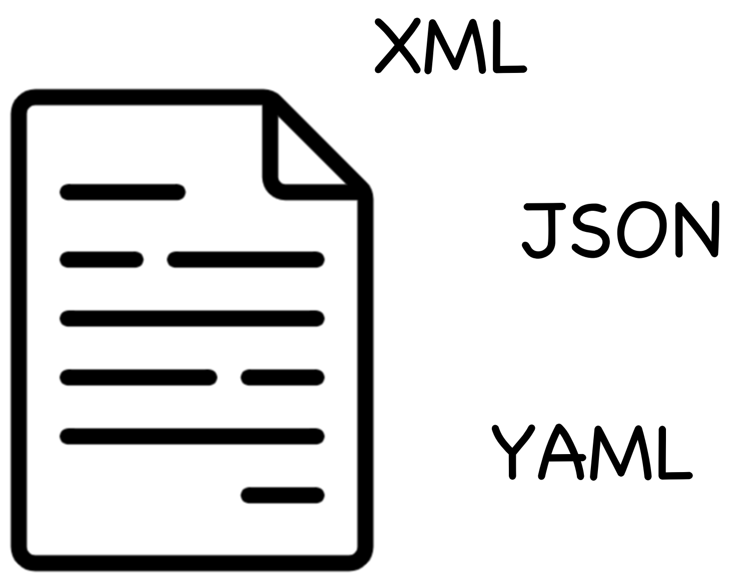


Client APIs

watch, kill, prune, reconnect..

Component guarantees

message ordering, redundancy, ACID..

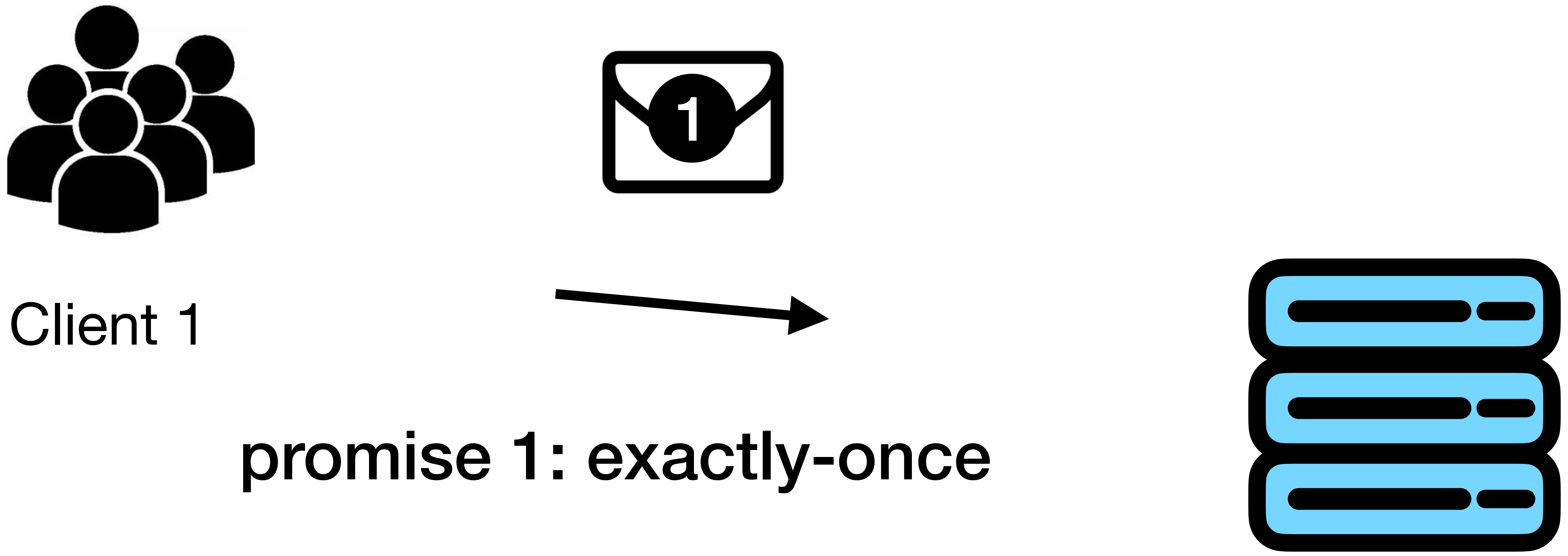


Configurations

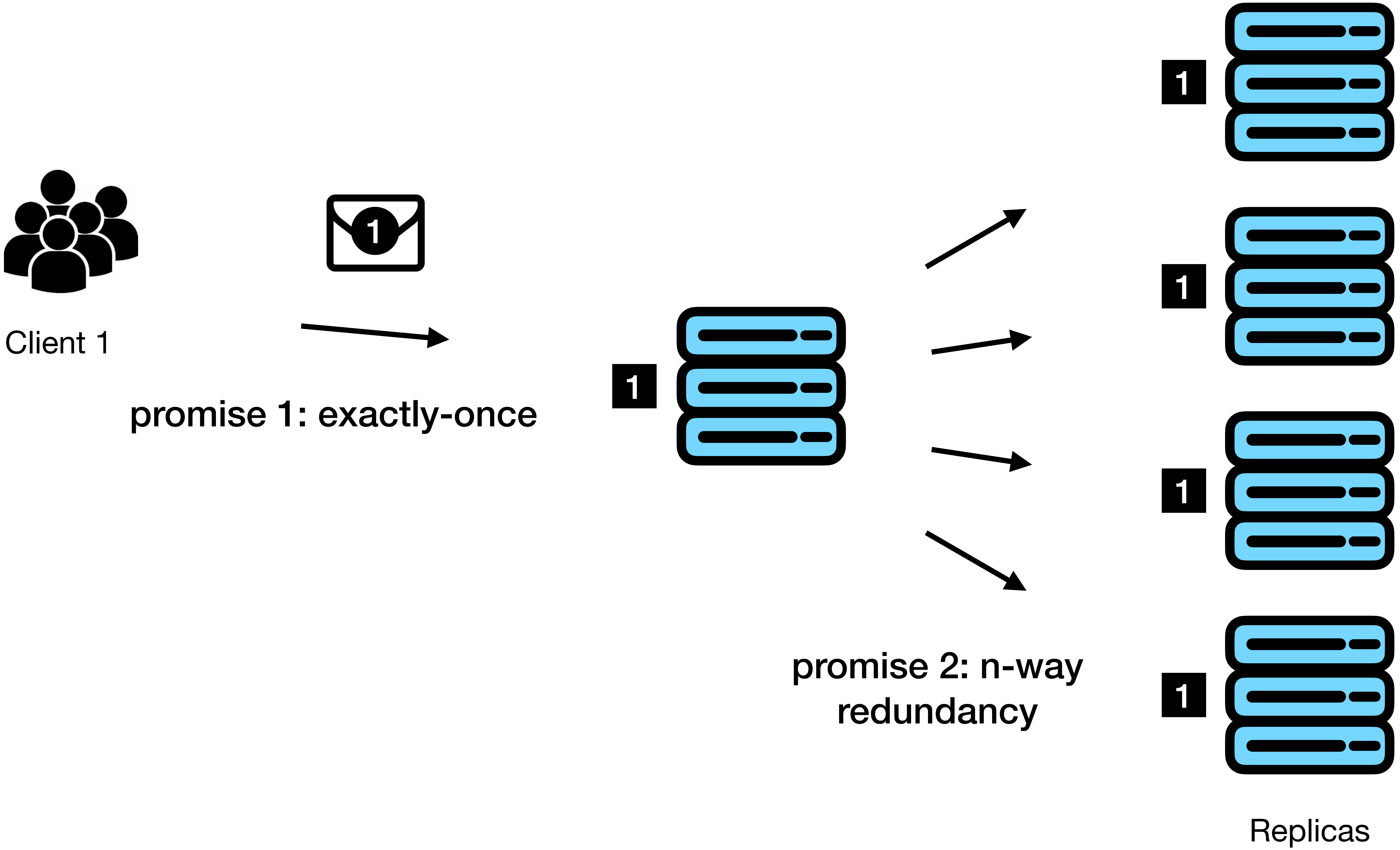
tickTime, snapCount, maxClientCnxns..

Semantics encode various promises

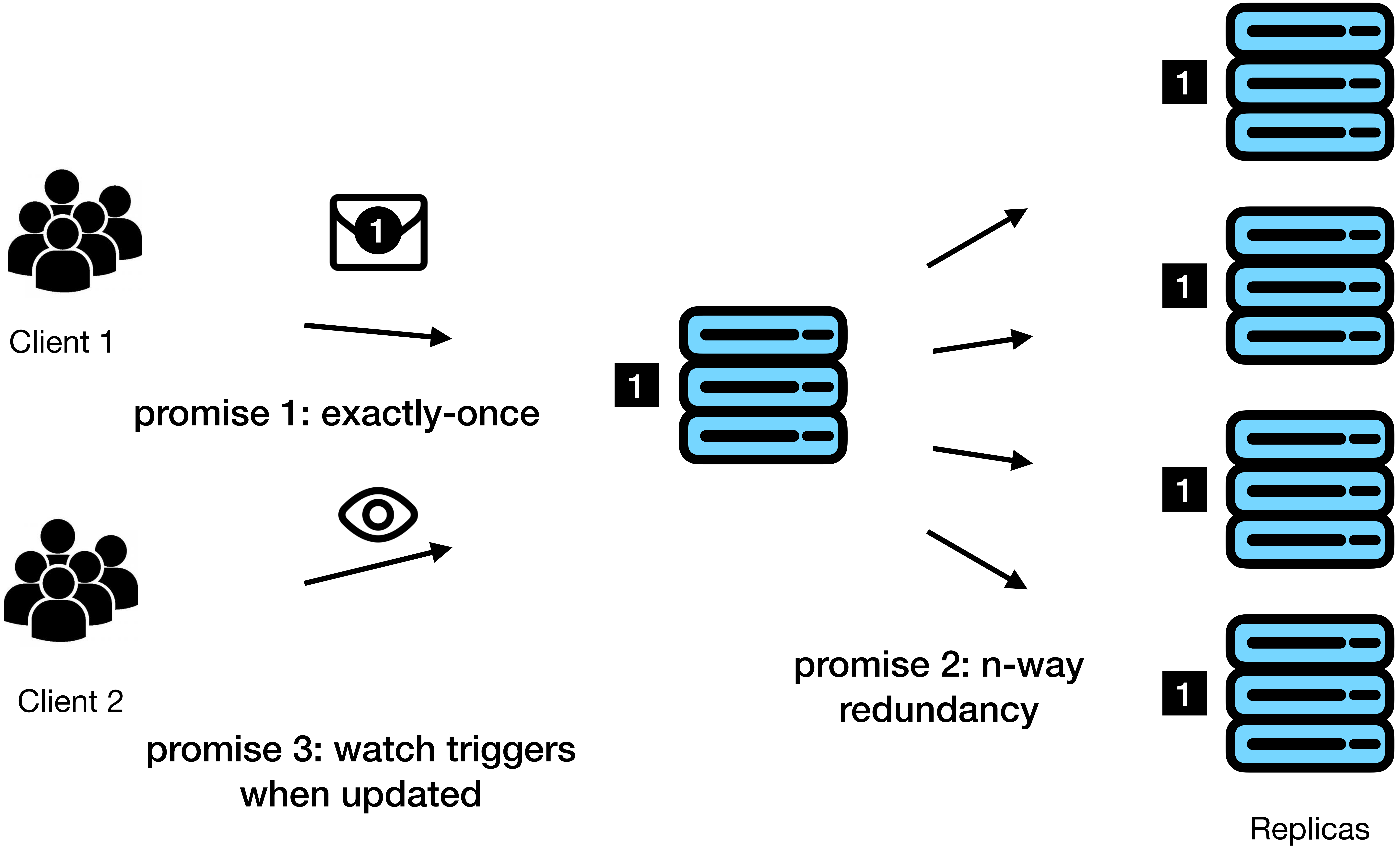
Semantics encode various promises



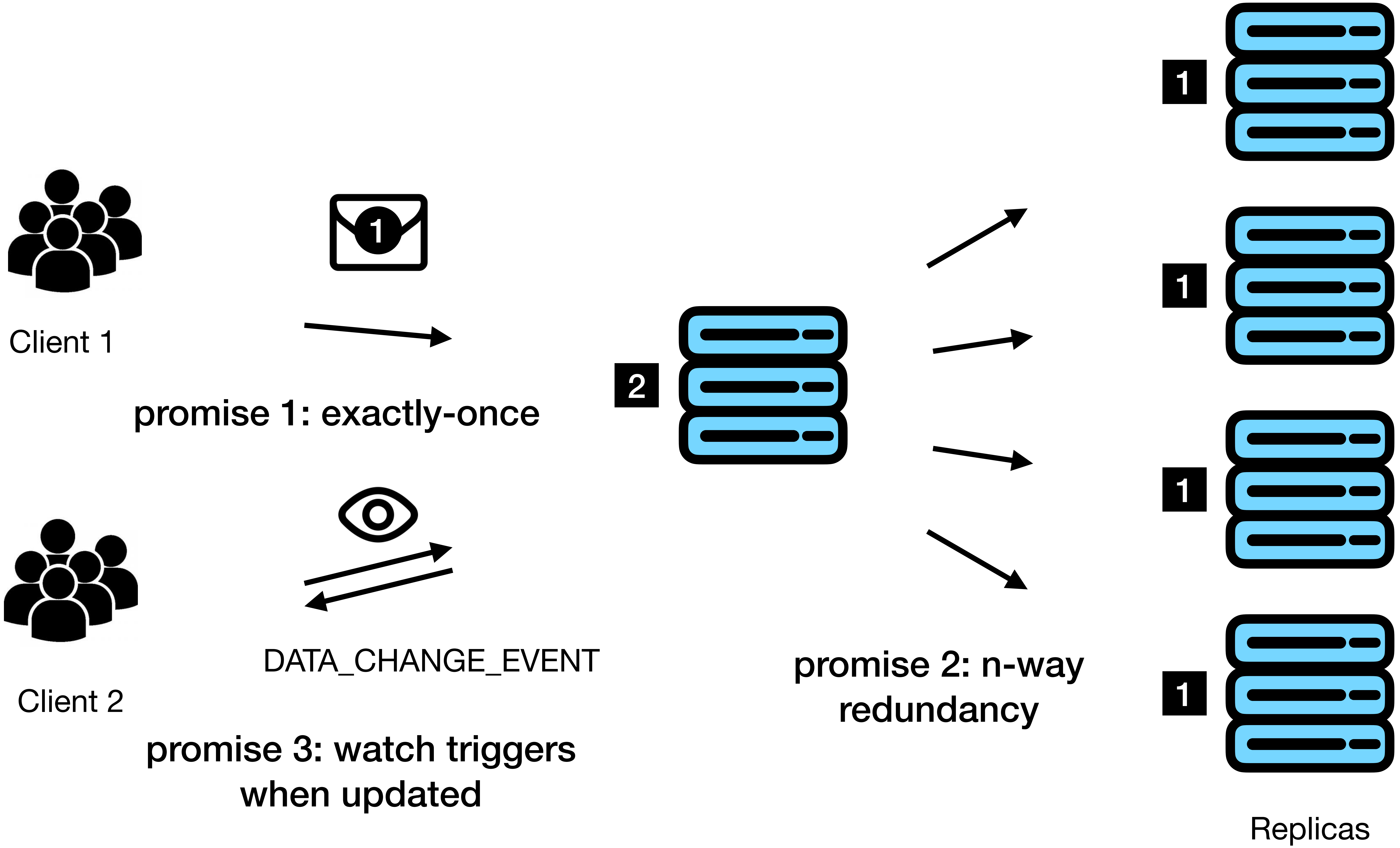
Semantics encode various promises



Semantics encode various promises

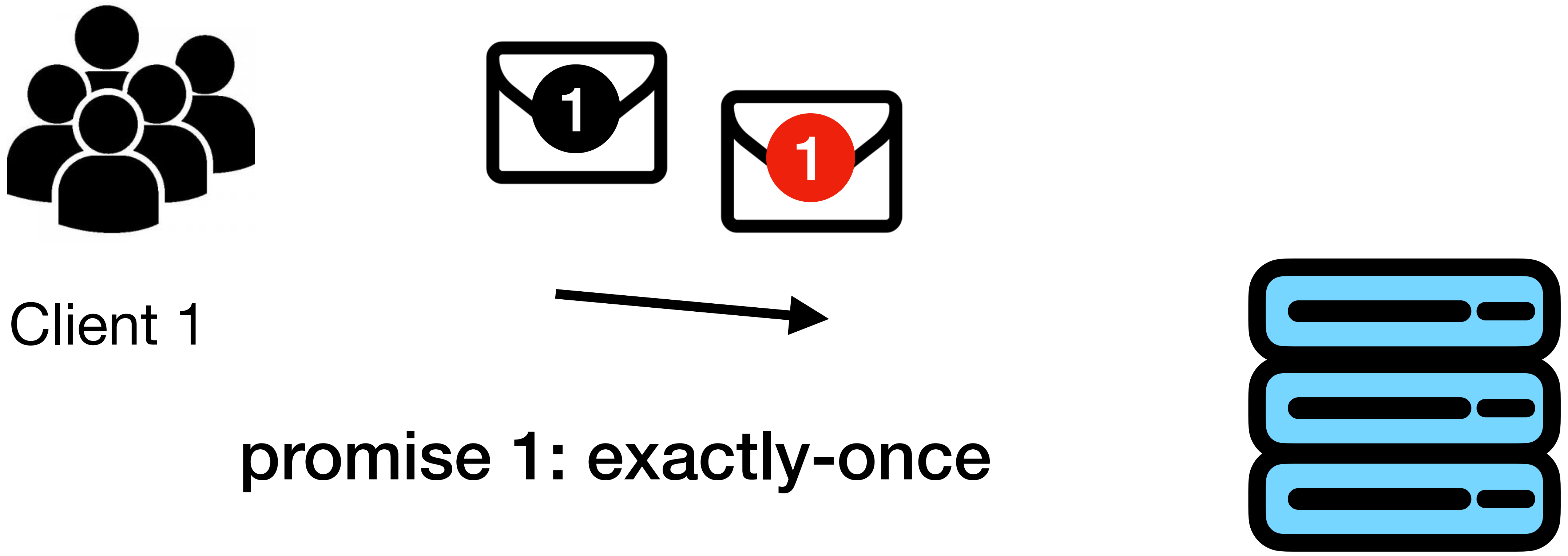


Semantics encode various promises

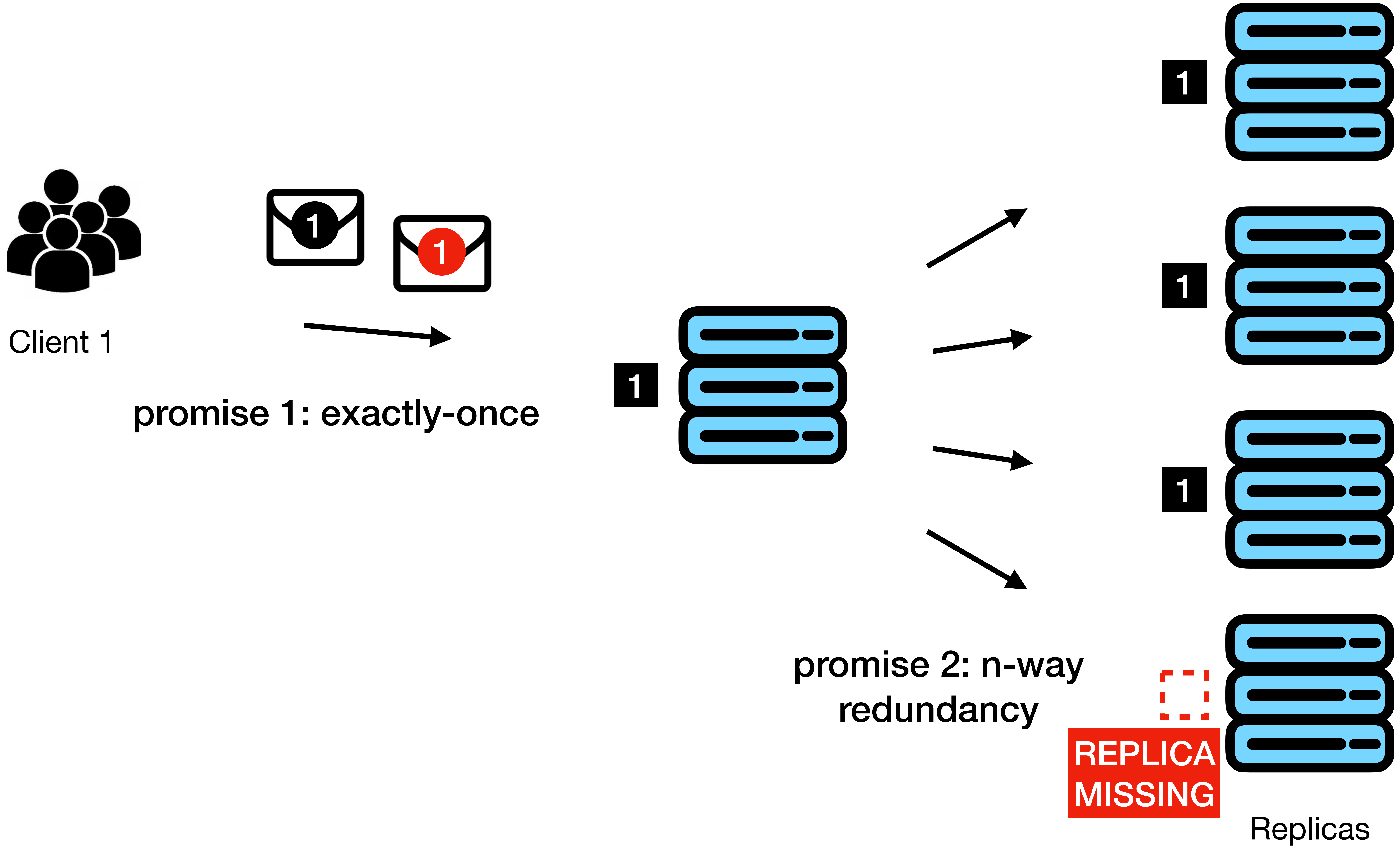


When a promise is not a promise

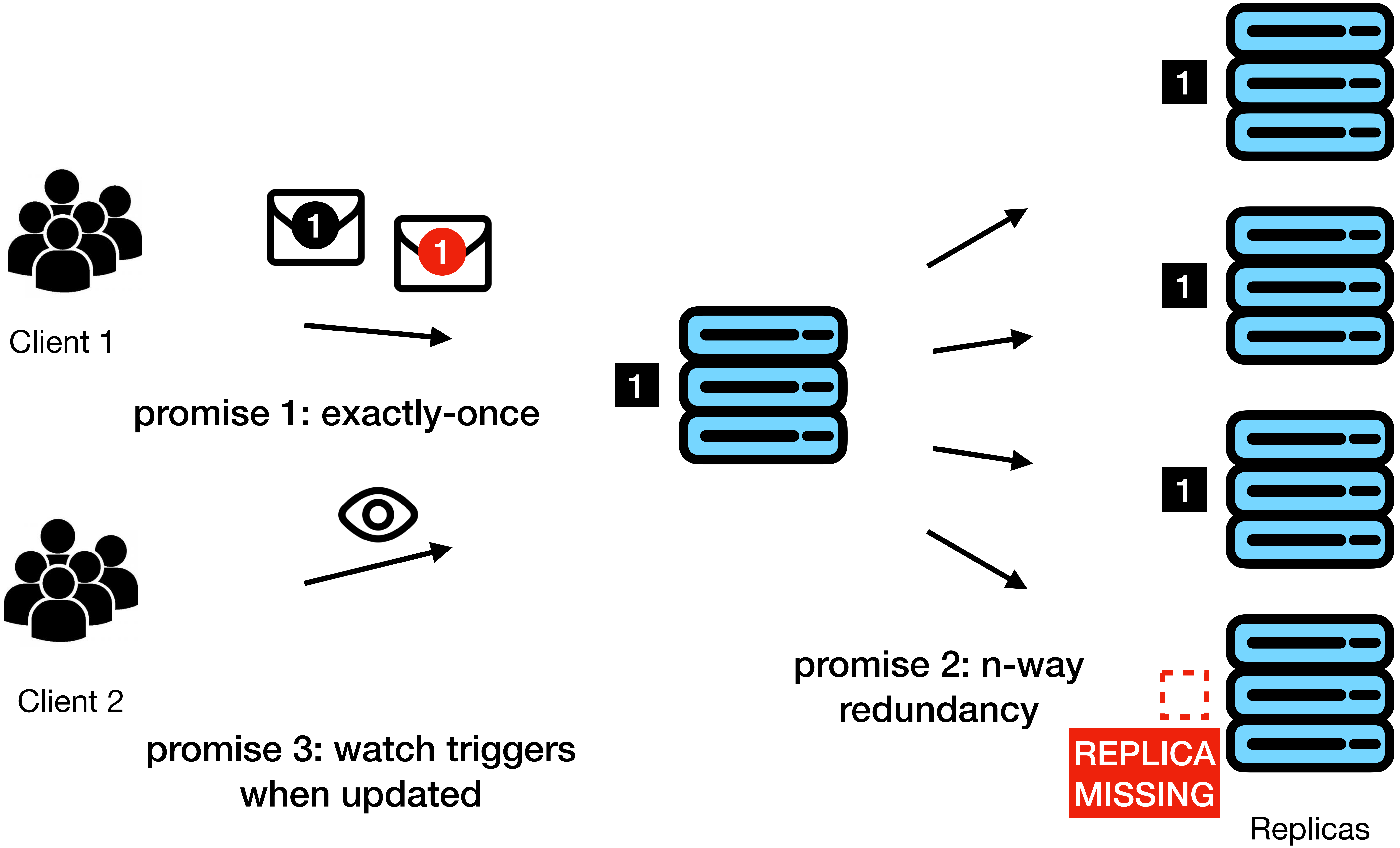
When a promise is not a promise



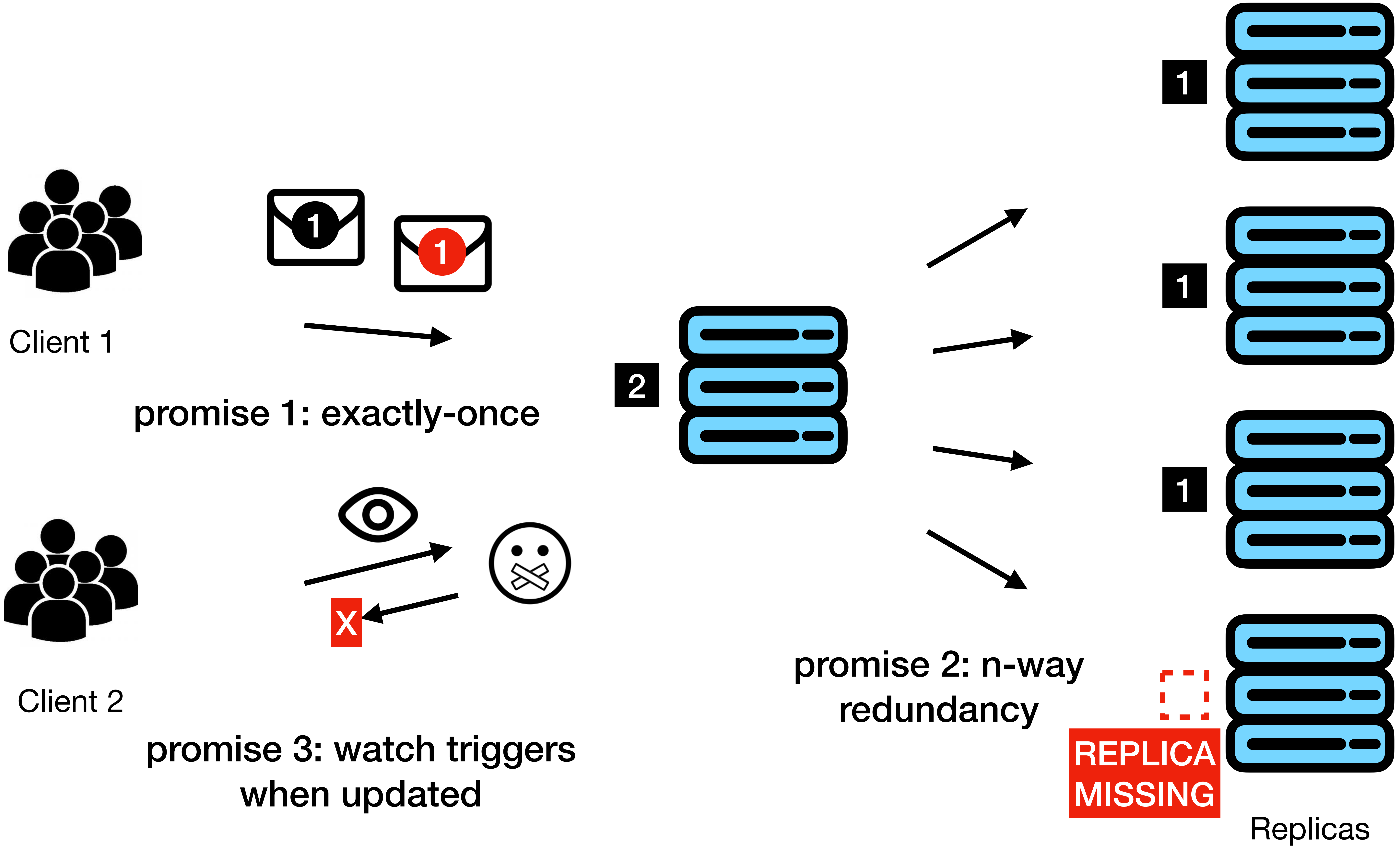
When a promise is not a promise



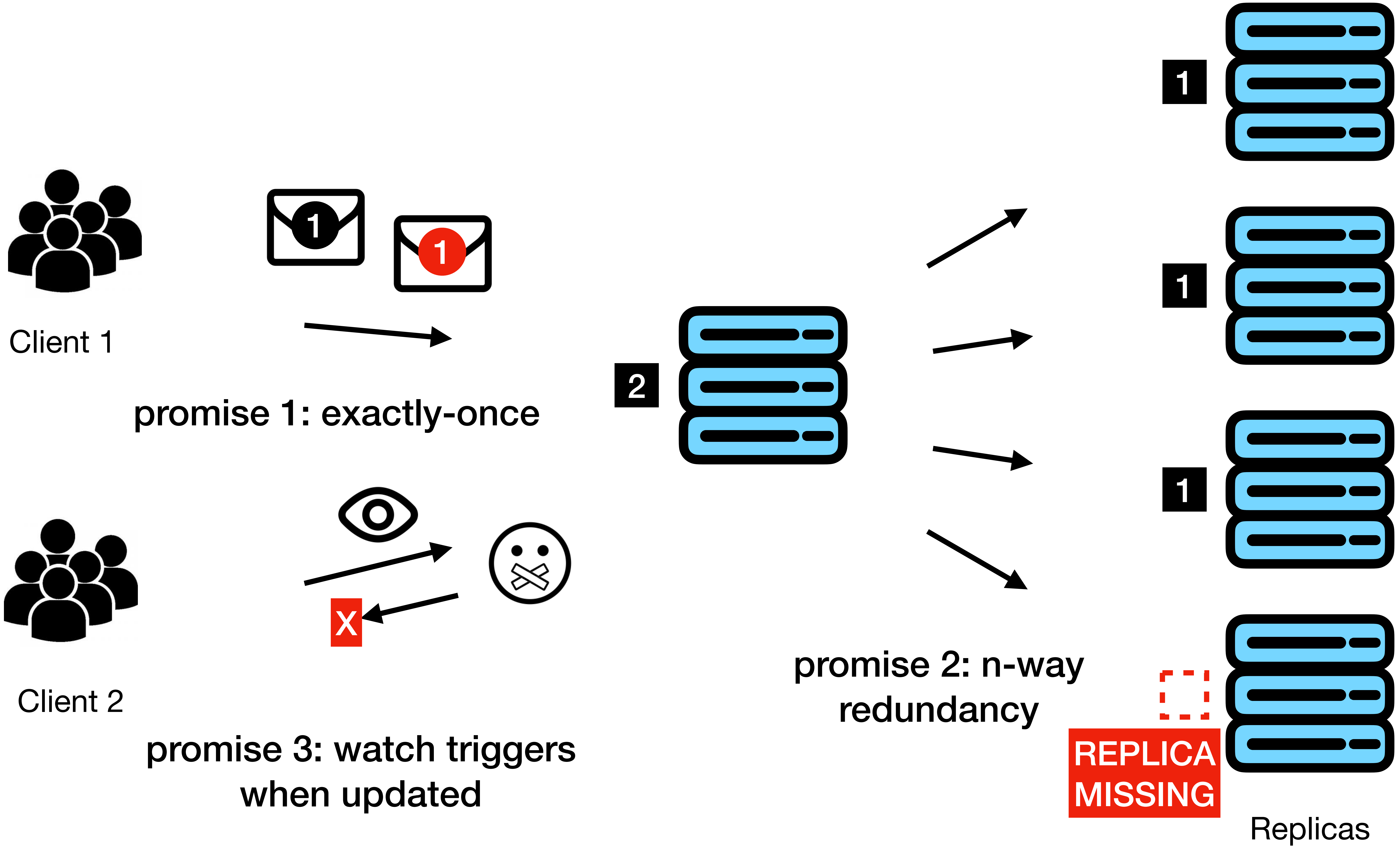
When a promise is not a promise



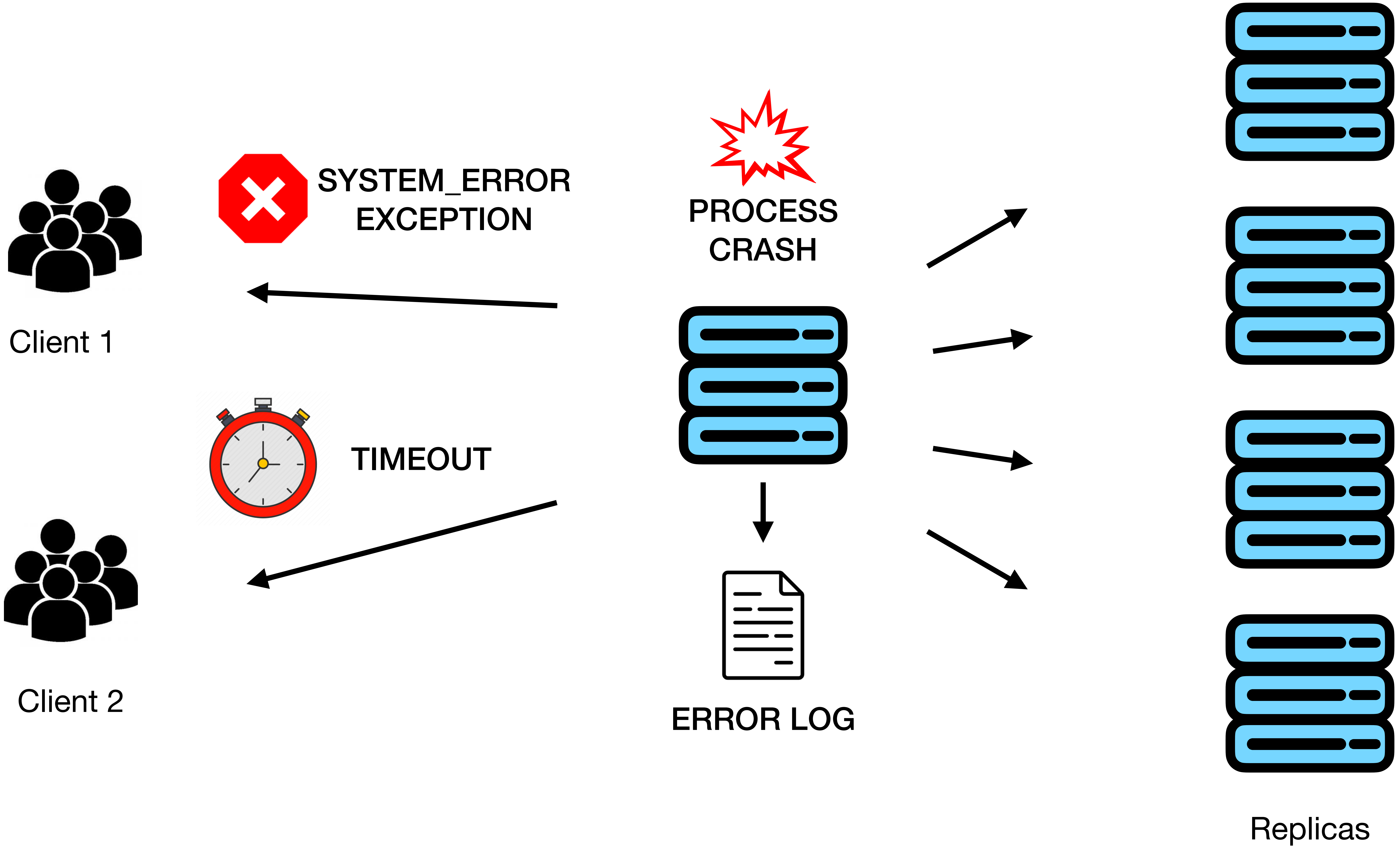
When a promise is not a promise



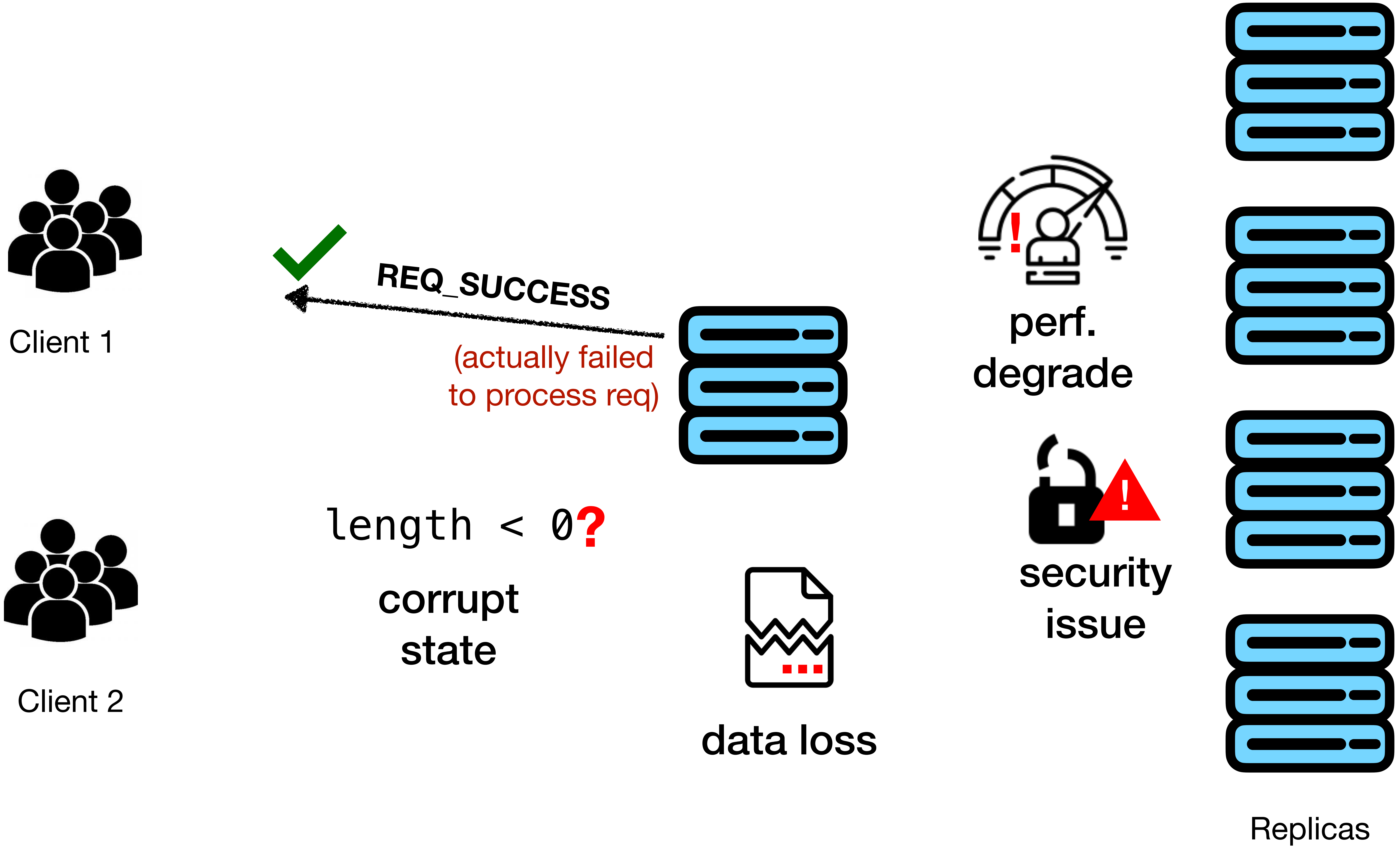
Semantic violations



Existing work focus on failures w/ explicit errors



Silent semantic violations



Contributions

1. A study on **109** real-world silent semantic violations
    - cases collected from **9** popular distributed systems
2. A detection solution: Oathkeeper
    - automatically **infer** semantic rules from **past failures**
    - enforce the rules at runtime to **detect** new failures

Study methodology

- Study on real-world incidents from nine distributed systems
  - randomly sampled **747** user-reported failures in total
  - confirmed **268** cases as silent semantic violations
  - performed in-depth studies on **109** cases

System	Category	Lang.	Sampled Total	Sampled Confirmed	Studied
Cassandra	Database	Java	54	25	12
CephFS	File System	C++	123	37	12
ElasticSearch	Search	Java	46	26	10
HBase	Database	Java	80	32	14
HDFS	File System	Java	52	22	14
Kafka	Streaming	Scala	92	39	13
Mesos	Cluster Manager	C++	47	21	12
MongoDB	Database	C++	151	30	10
ZooKeeper	Coordination	Java	102	36	12
Total	/	/	747	268	109

Major findings

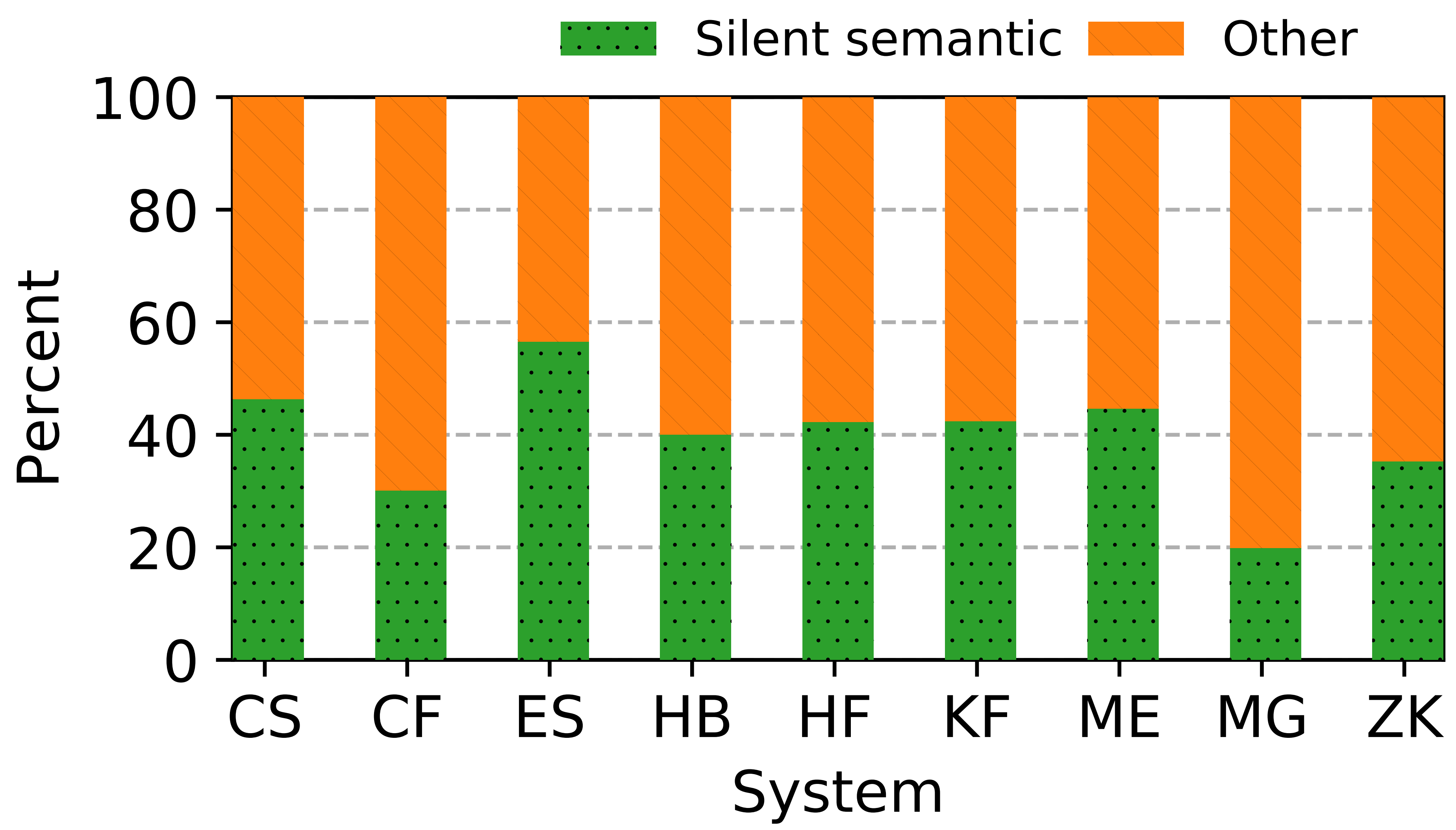
- [**Prevalence**] How common are silent semantic violations in production?
- [**Age of semantics**] How long has the violated semantics existed?
- [**Testing**] Is semantics covered by tests and why did not expose issue?
- [**Root cause**] Can we find common bug patterns for static checking?
- [**Timing**] When do semantic violations happen?
- ...

Prevalence

- Myth: are silent semantic violations **rare** in production?

Prevalence

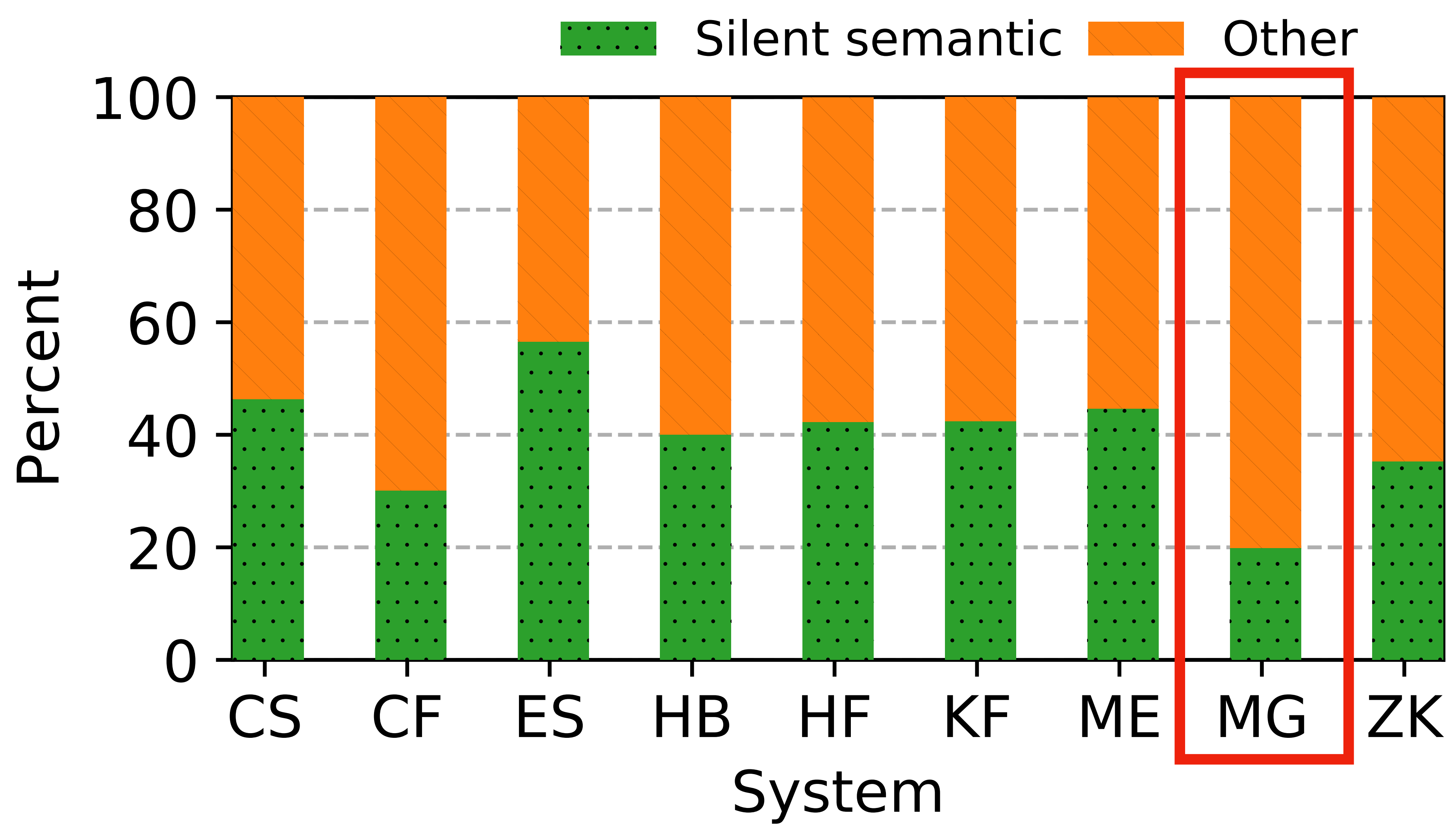
- Myth: are silent semantic violations **rare** in production?
- Finding 1: silent semantic violations are **prevalent**
  - occupy 39% of cases for all types of failures



Prevalence

- Myth: are silent semantic violations **rare** in production?

- Finding 1: silent semantic violations are **prevalent**
  - occupy 39% of cases for all types of failures




```
…
invariant(!msg->empty());
invariant(msg->operation() == dbMsg);
invariant(msg->dataSize() >= sizeof(uint32_t));
DataView(msg->data()).write(flags);
…
```

MongoDB has lowest ratio

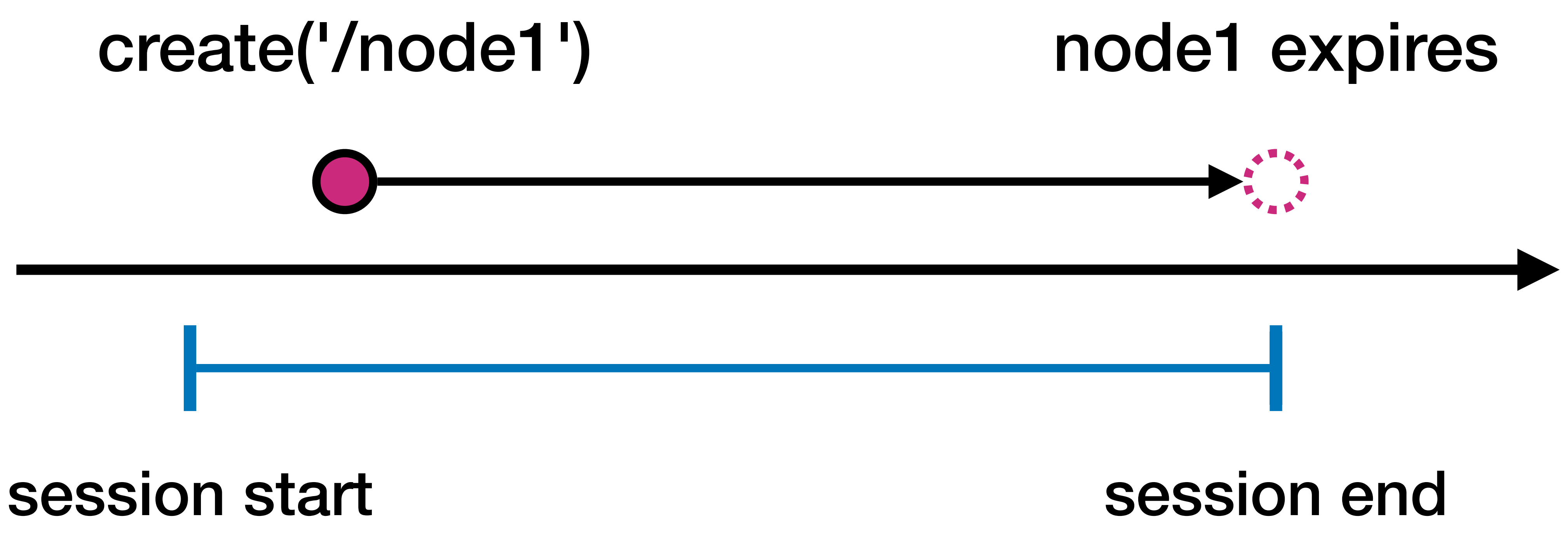
Age of semantics

- Myth: violated semantics are fragile because they are **new**?

Age of semantics

- Myth: violated semantics are fragile because they are **new**?

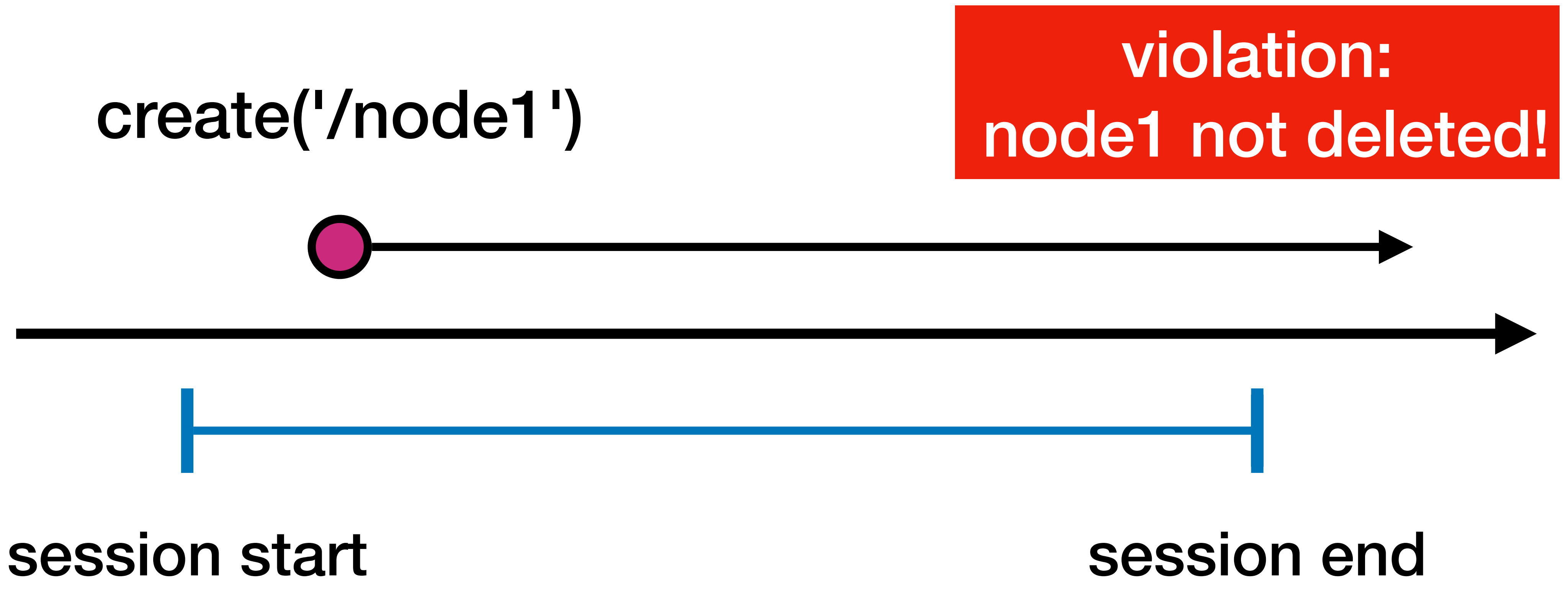
- Finding 2: **68%** of the studied failures violate **old** semantics
  - "old" means semantics exist since the first major release of the system
  - same semantics is repeatedly violated, *e.g.*, ZooKeeper ephemeral node

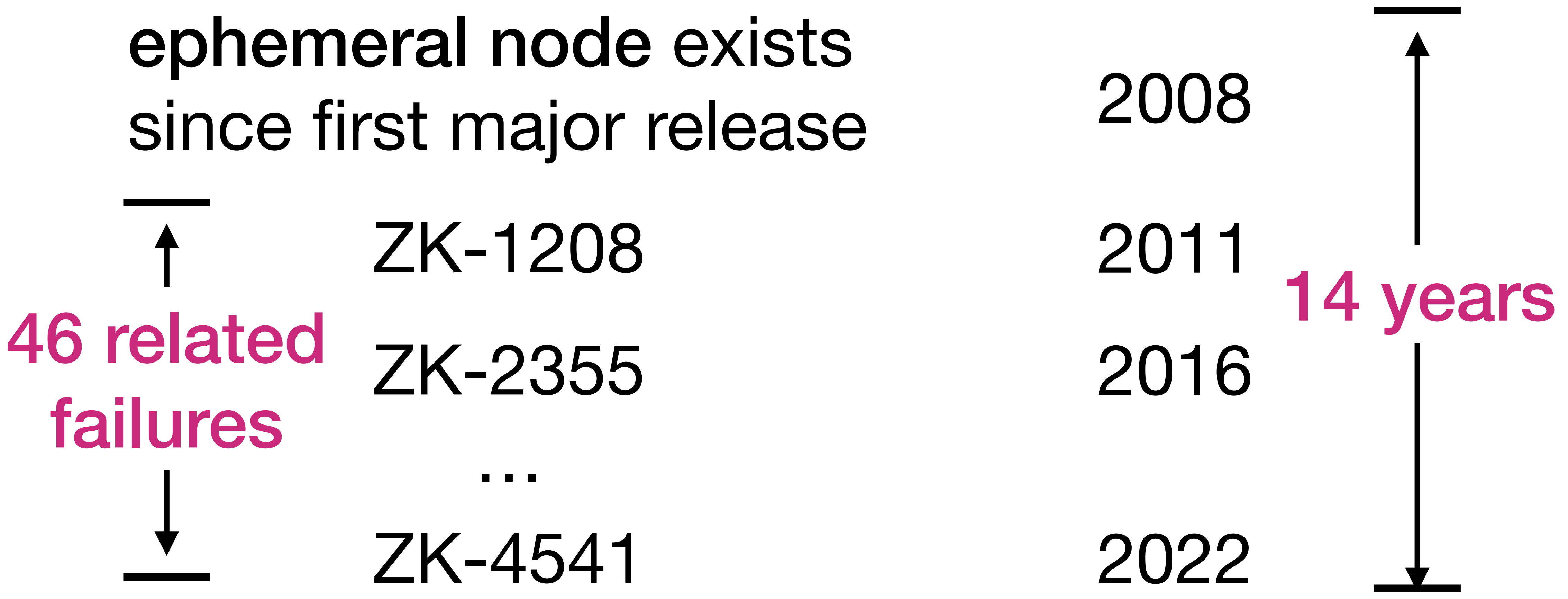


Age of semantics

- Myth: violated semantics are fragile because they are **new**?

- Finding 2: **68%** of the studied failures violate **old** semantics
  - "old" means semantics exist since the first major release of the system
  - same semantics is repeatedly violated, *e.g.*, ZooKeeper ephemeral node





Testing

- Myth: does violated semantics have poor testing?

Testing

- Myth: does violated semantics have poor testing?
- Finding 3: **73%** of violated semantics are covered by **existing** tests

Testing

- Myth: does violated semantics have poor testing?

- Finding 3: **73%** of violated semantics are covered by **existing** tests
  - lack operations, arguments, timing to expose new violations

Testing

- Myth: does violated semantics have poor testing?

- Finding 3: **73%** of violated semantics are covered by **existing** tests
  - lack operations, arguments, timing to expose new violations

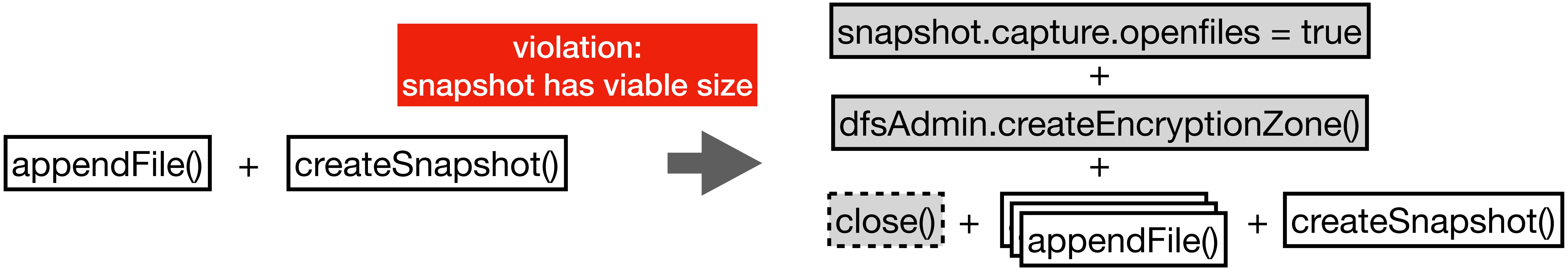
appendFile() + createSnapshot()

triggering conditions in existing test

Testing

- Myth: does violated semantics have poor testing?

- Finding 3: **73%** of violated semantics are covered by **existing** tests
  - lack operations, arguments, timing to expose new violations

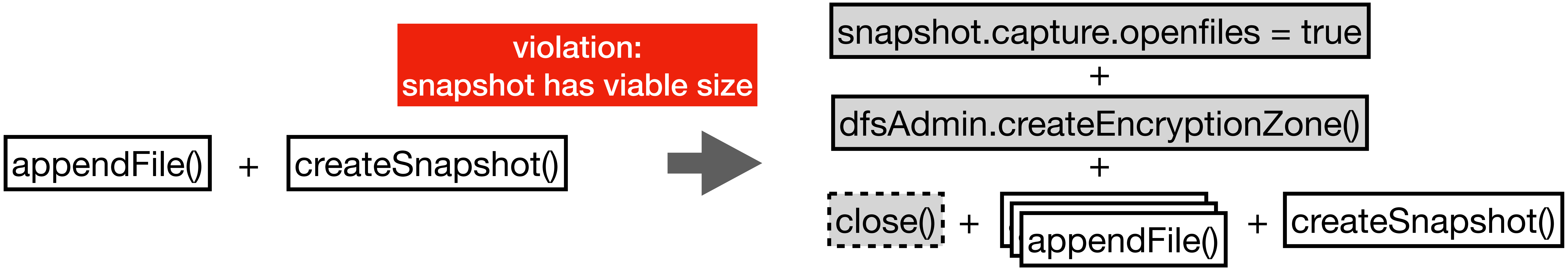


triggering conditions in existing test **triggering conditions for new violation**

Testing

- Myth: does violated semantics have poor testing?

- Finding 3: **73%** of violated semantics are covered by **existing** tests
  - lack operations, arguments, timing to expose new violations
  - existing efforts of writing tests do not effectively prevent future violations



triggering conditions in existing test

triggering conditions for new violation

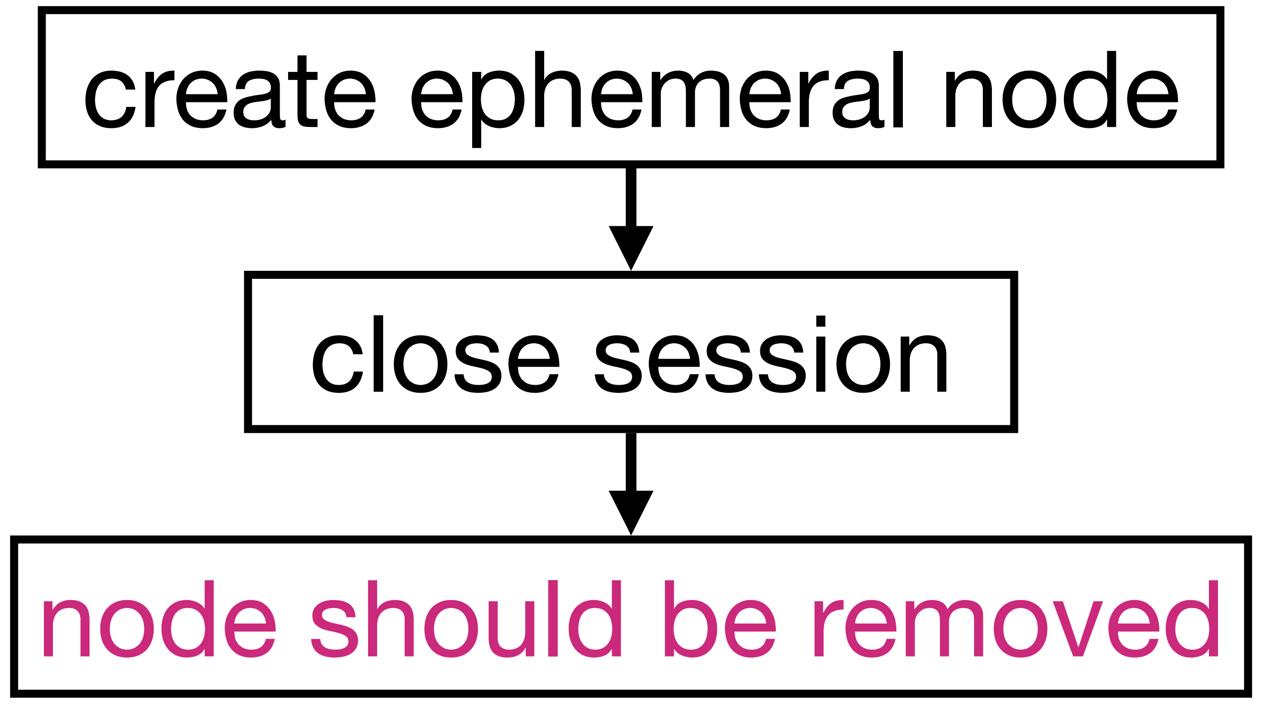
Root causes

- Myth: do same semantic violations have similar causes?

Root causes

- Myth: do same semantic violations have similar causes?

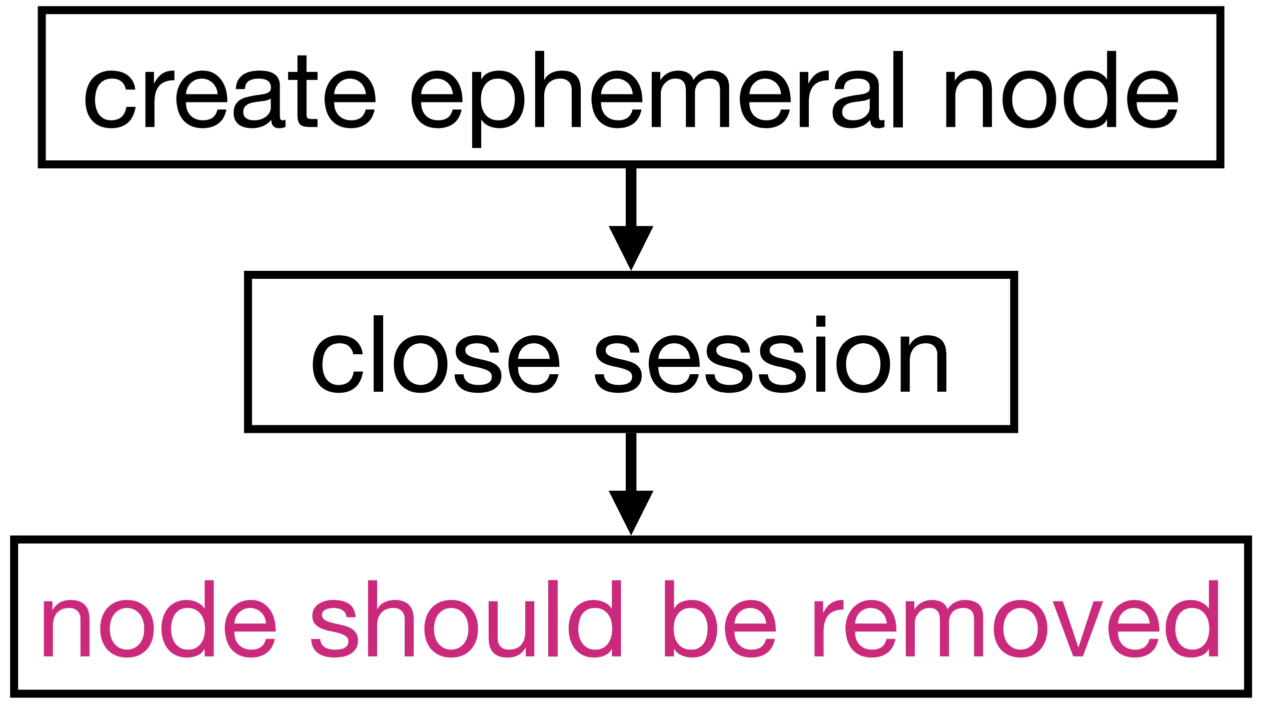
- Finding 4: root causes are **diverse**
  - even for failures violating the same semantics, the causes are often different



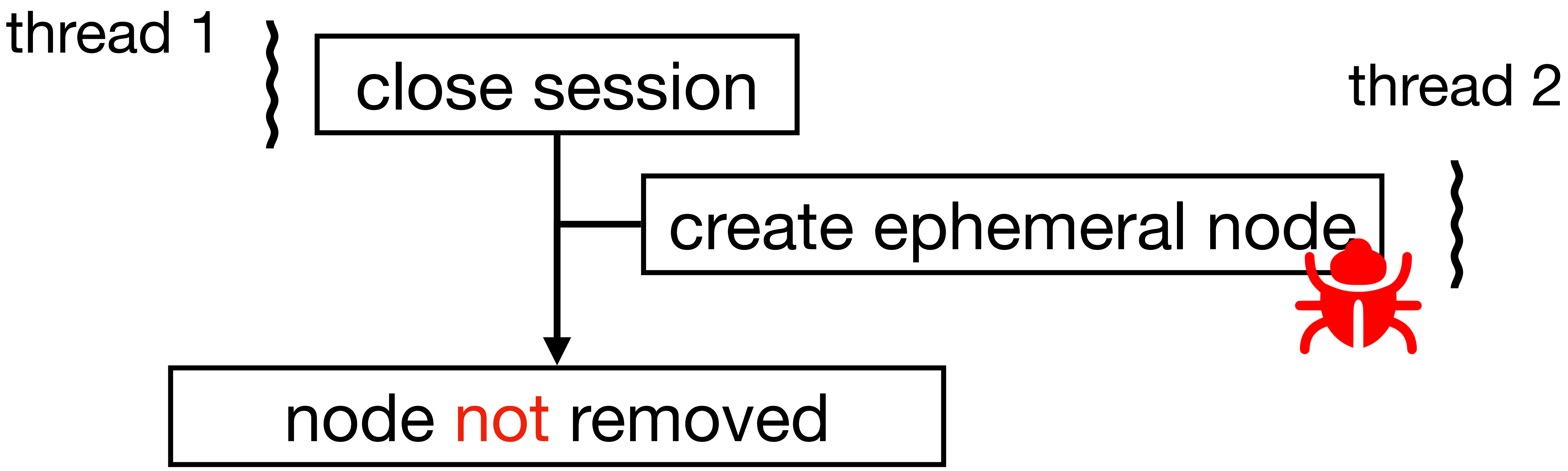
semantics: ephemeral node

Root causes

- Myth: do same semantic violations have similar causes?
- Finding 4: root causes are **diverse**
  - even for failures violating the same semantics, the causes are often different



semantics: ephemeral node

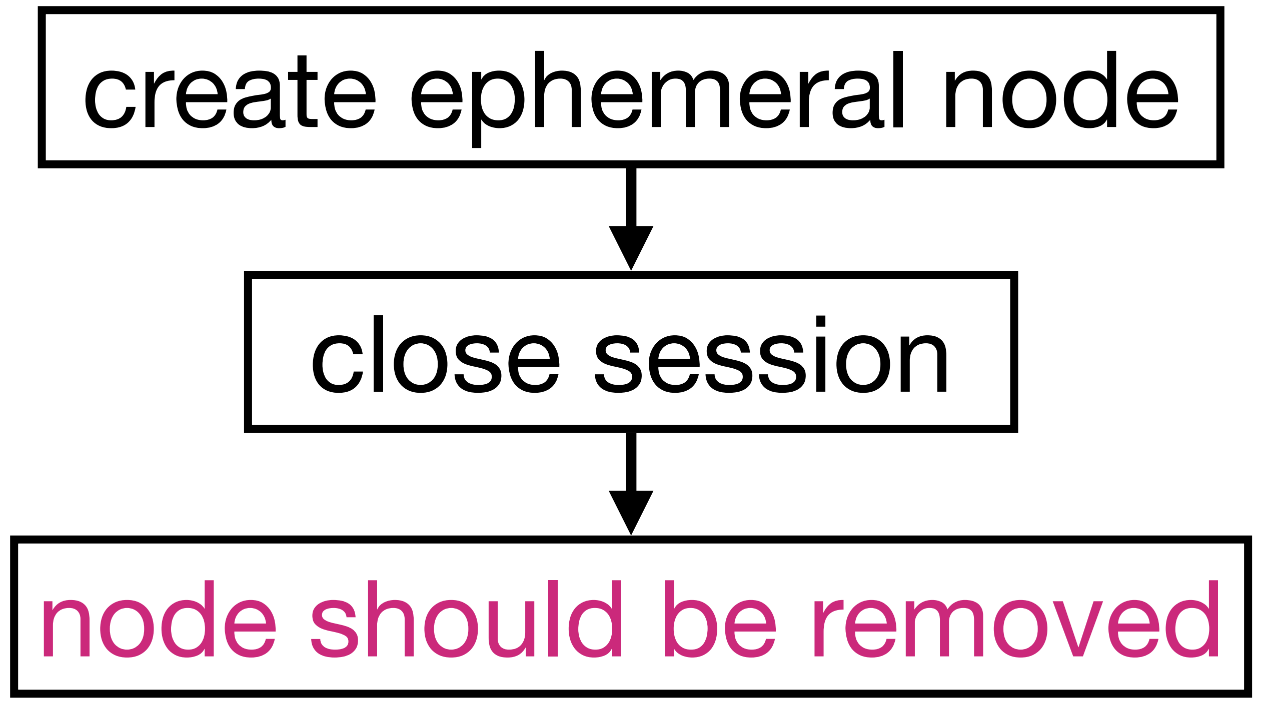


❶ ZK-1208: race condition

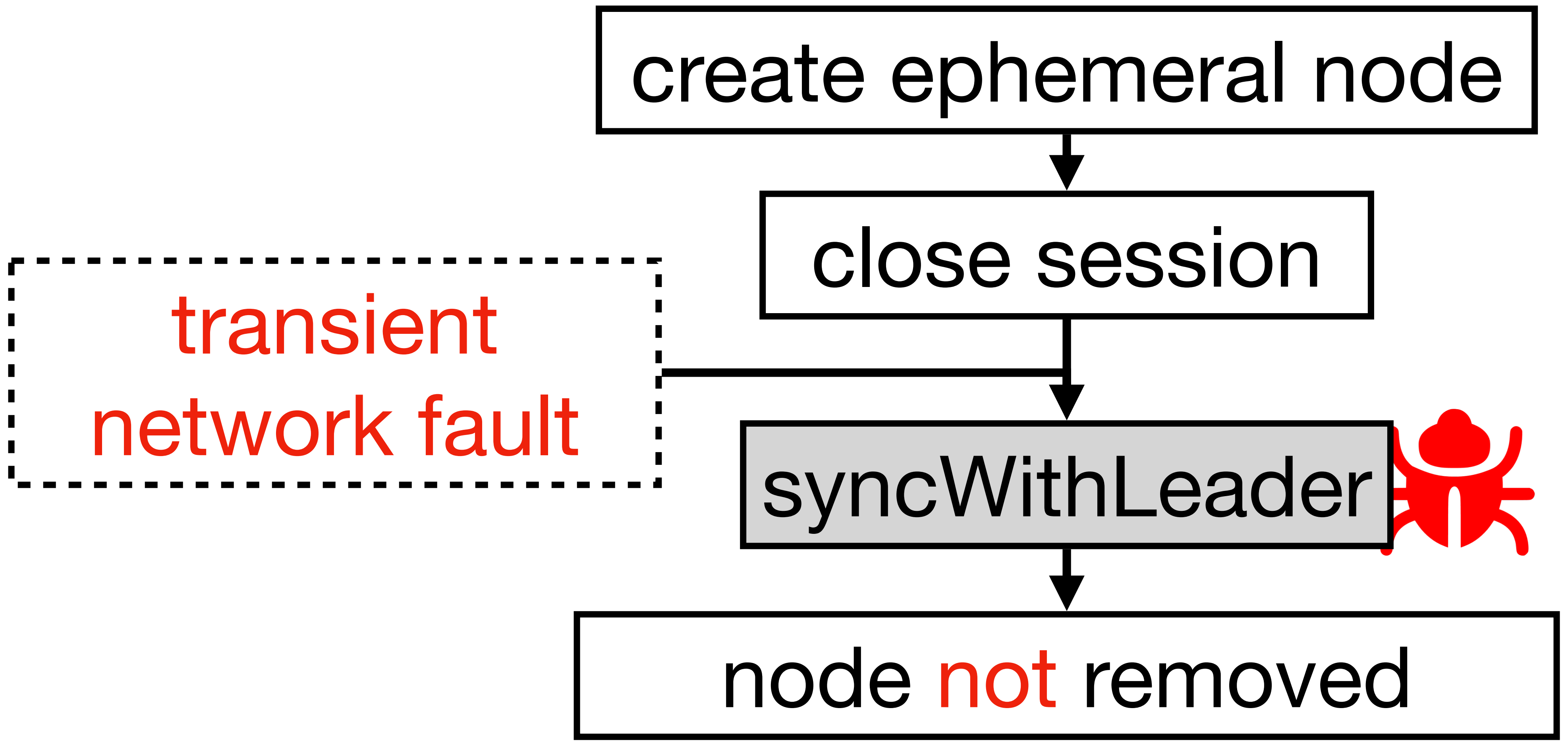
Root causes

- Myth: do same semantic violations have similar causes?

- Finding 4: root causes are **diverse**
  - even for failures violating the same semantics, the causes are often different



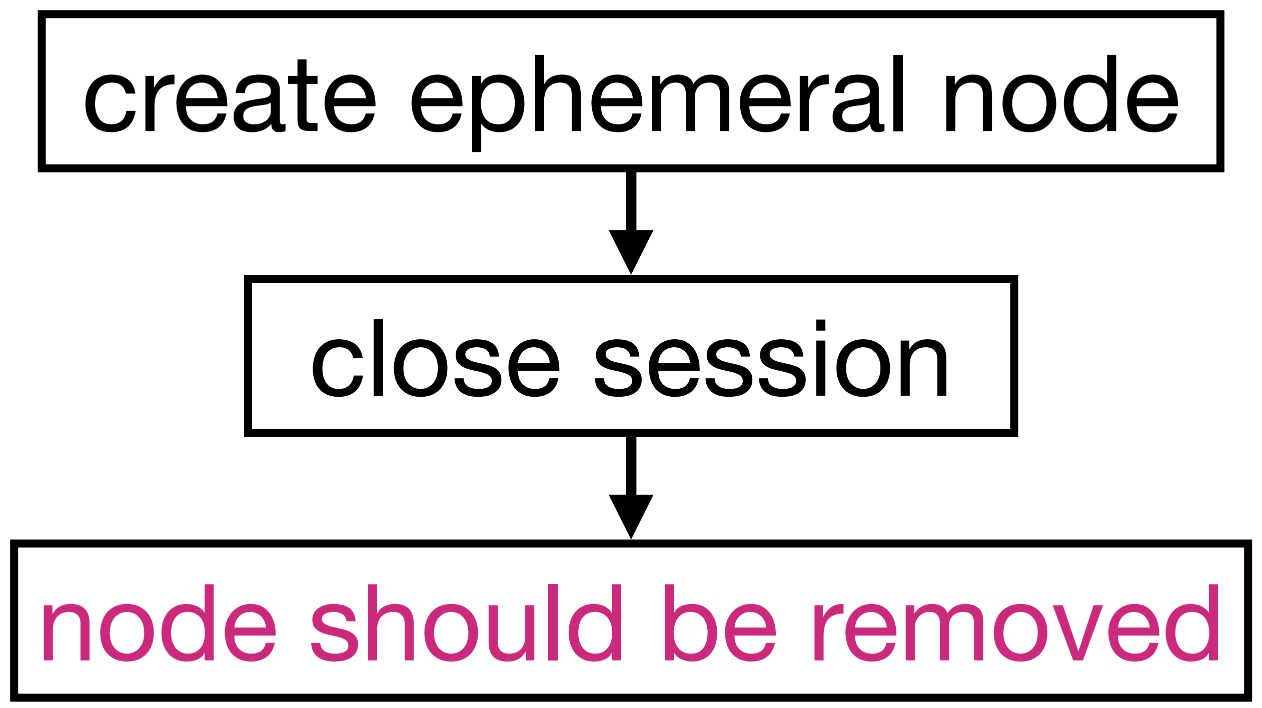
semantics: ephemeral node



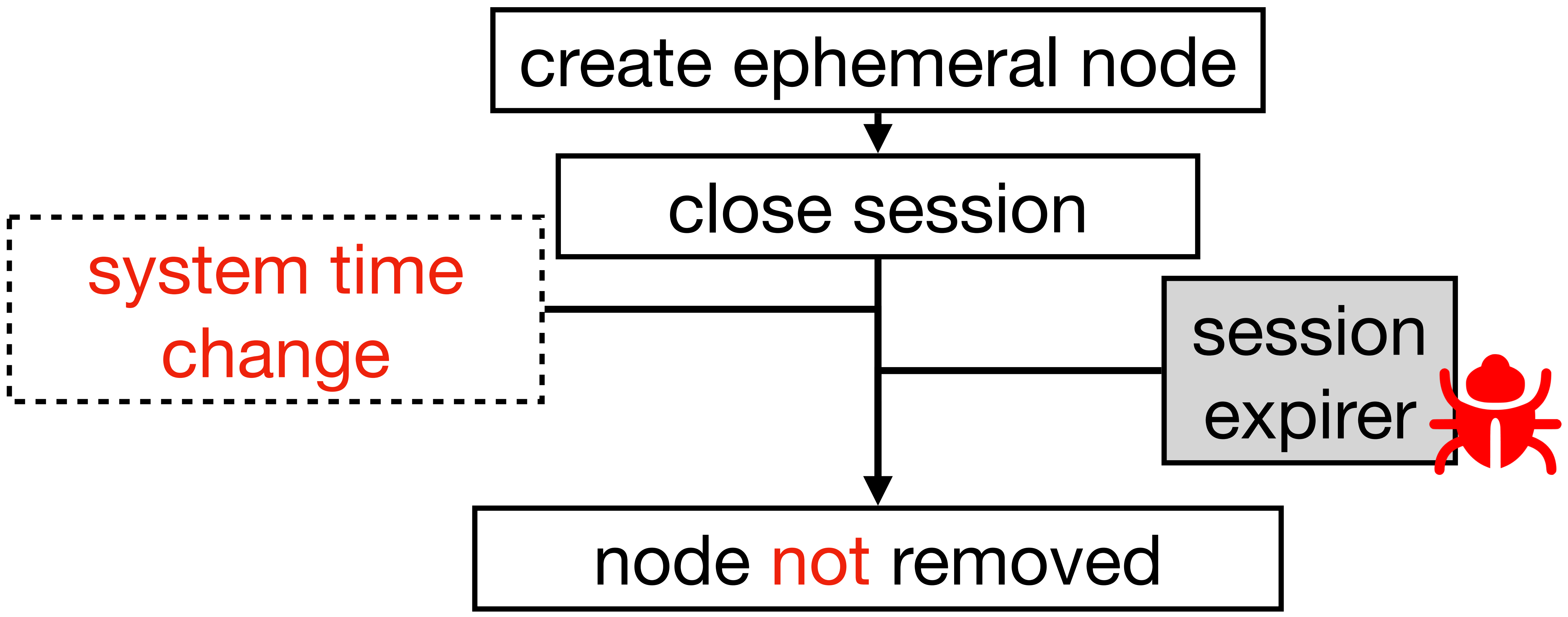
❷ ZK-2355: buggy error handling

Root causes

- Myth: do same semantic violations have similar causes?
- Finding 4: root causes are **diverse**
  - even for failures violating the same semantics, the causes are often different



semantics: ephemeral node



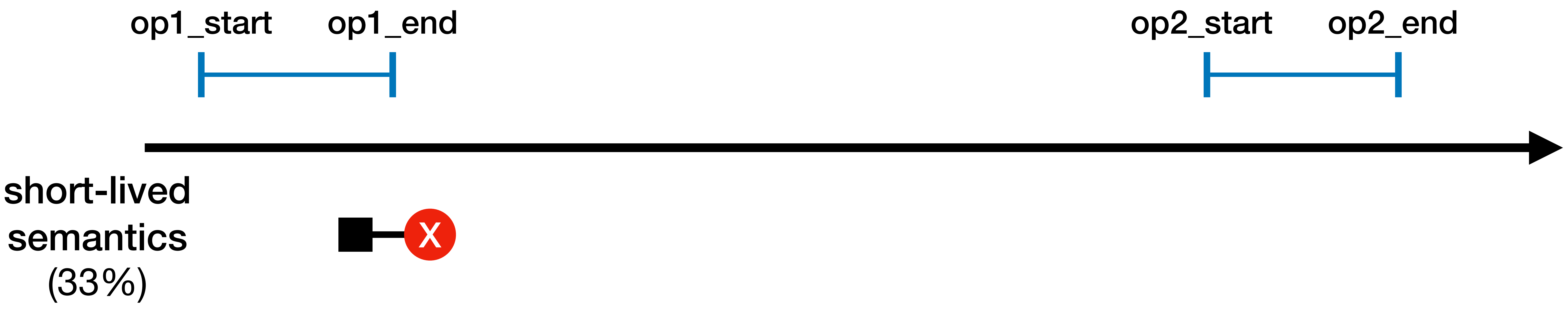
③ ZK-2774: skewed system time

Timing of violation

- Myth: appending a check after each operation can solve problem

Timing of violation

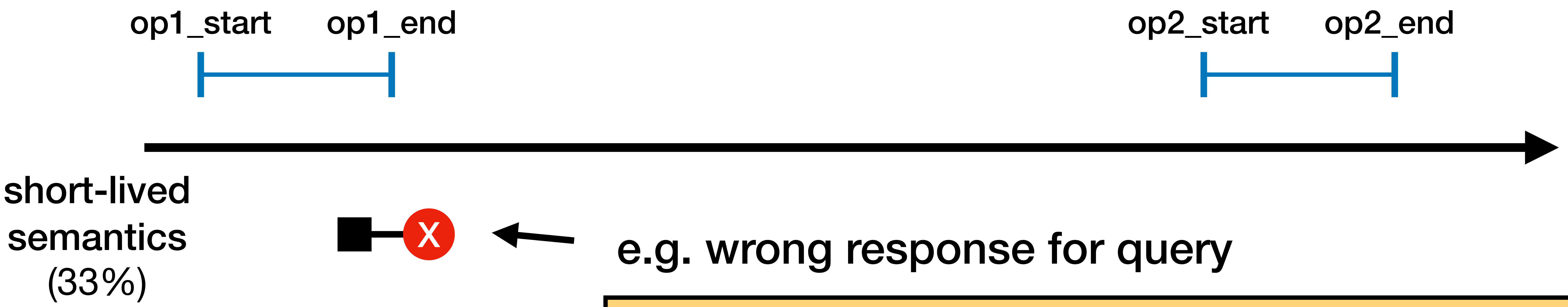
- Myth: appending a check after each operation can solve problem
- Finding 5: **67%** of cases violate **long-lived** semantics



Timing of violation

- Myth: appending a check after each operation can solve problem

- Finding 5: **67%** of cases violate **long-lived** semantics

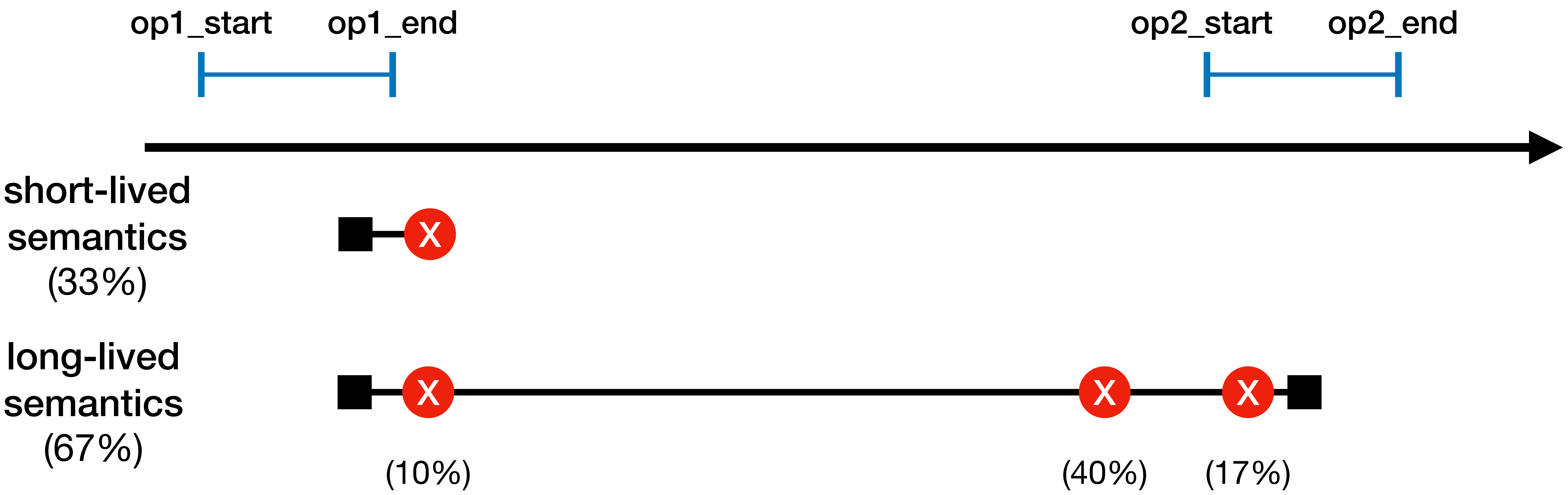


e.g. wrong response for query

> UPDATE cyclists SET name = '**Alex**' WHERE id = 11;
> SELECT name FROM cyclists WHERE id = 11;
Alice

Timing of violation

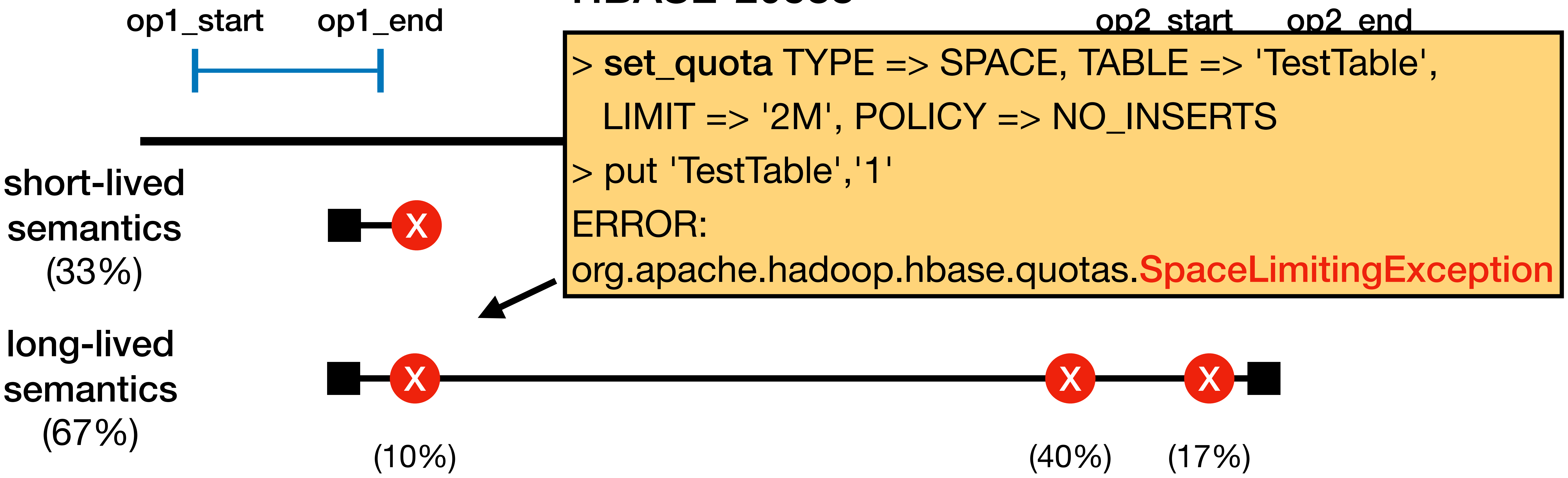
- Myth: appending a check after each operation can solve problem
- Finding 5: **67%** of cases violate **long-lived** semantics



Timing of violation

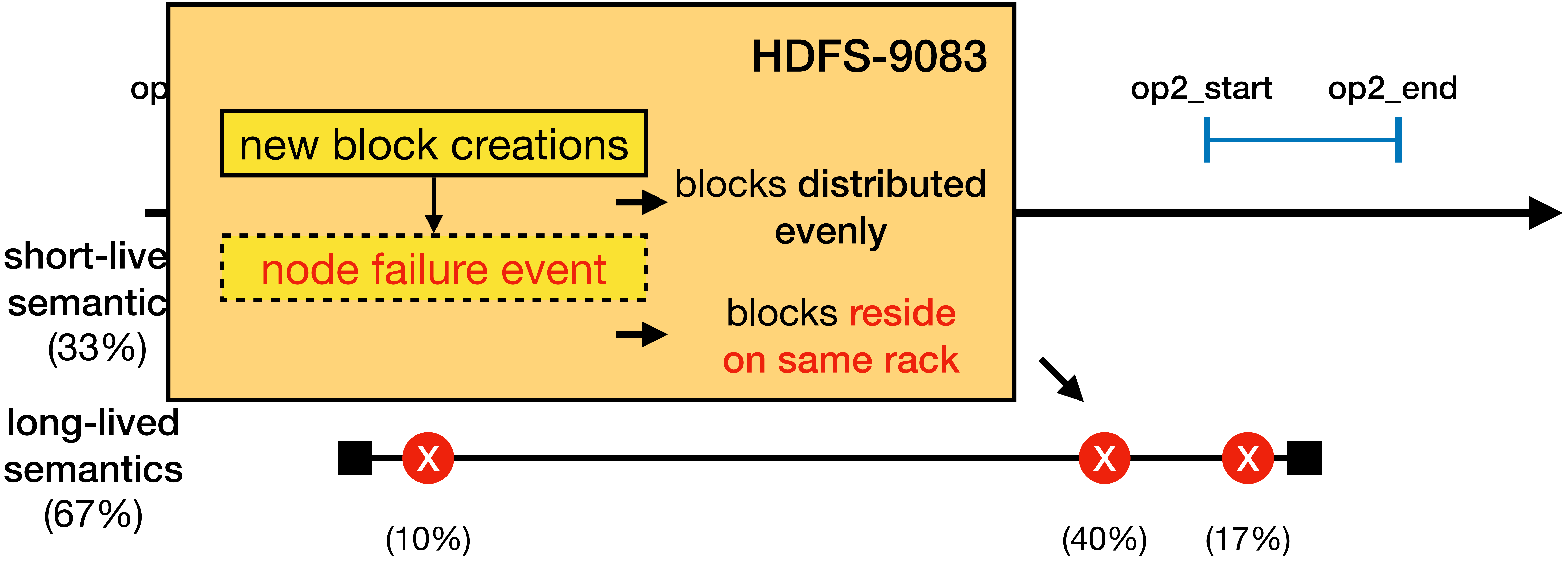
- Myth: appending a check after each operation can solve problem
- Finding 5: **67%** of cases violate **long-lived** semantics

HBASE-20588



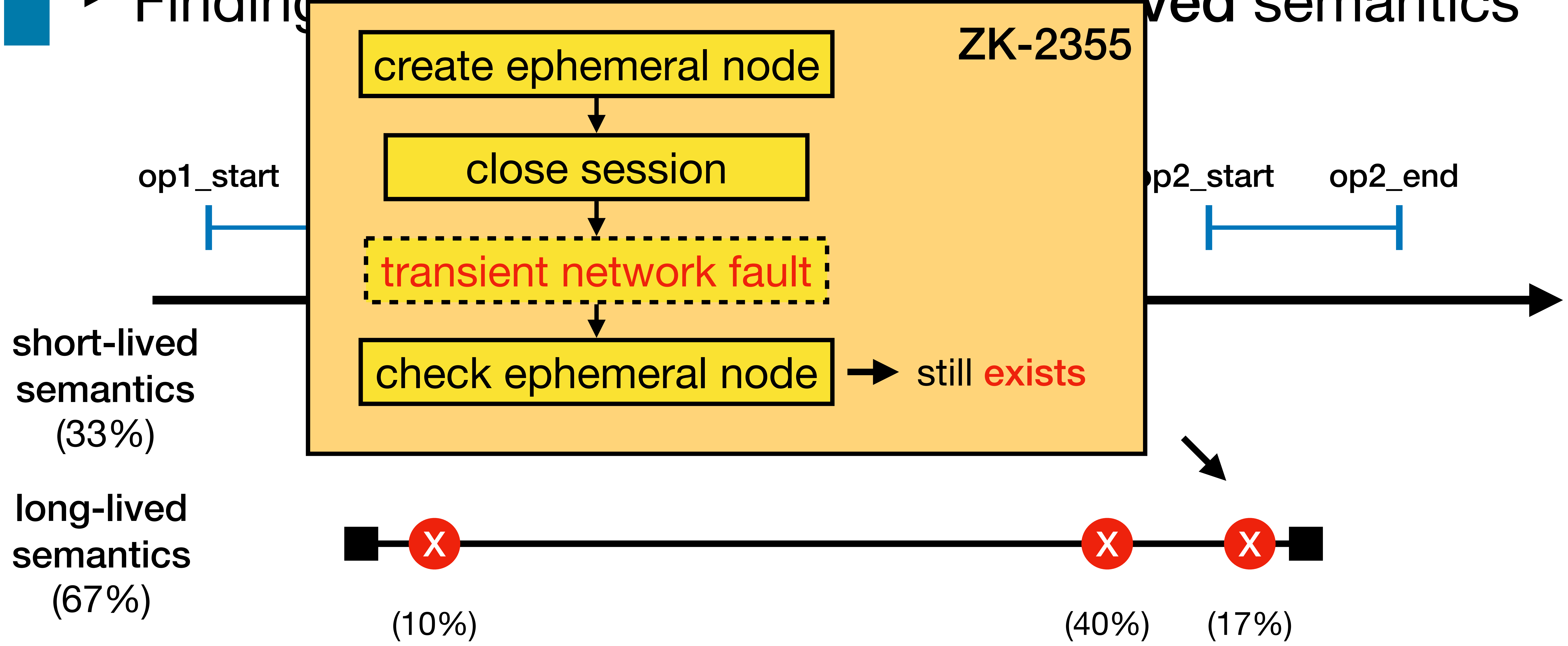
Timing of violation

- Myth: appending a check after each operation can solve problem
- Finding 5: **67%** of cases violate **long-lived** semantics



Timing of violation

- Myth: appending a check after each operation can solve problem
- Finding 5: 67% of cases violate long-lived semantics



Other findings

- Finding 6: sanity checks are insufficient
  - in 51% of the failures the buggy functions have some sanity checks
  - only 9% cases can be potentially detected by adding proper sanity checks
- Finding 7: local vs. distributed semantics
- Finding 8: safety vs. liveness semantics
- Finding 9: user observability
- ...

See the full list of findings in our paper

Oathkeeper: a semantic violation detection tool

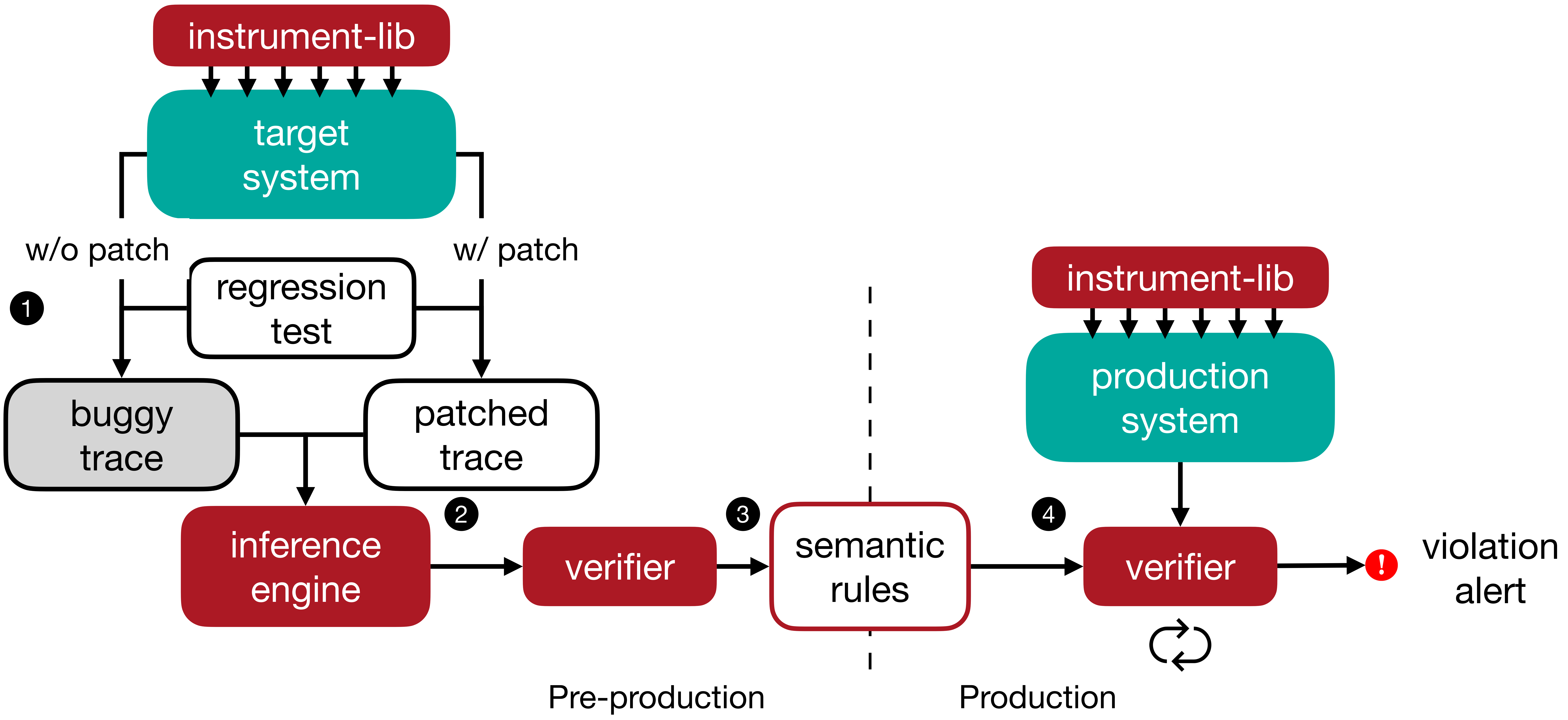
- **Motivating findings:**
  - the majority of studied failures violate old semantics
  - the testing coverage of these semantics is decent
  - the same semantics is repeatedly violated by different root causes
  - many failures violate long-lived semantics
  - ...

Oathkeeper: a semantic violation detection tool

- Motivating findings:
  - the majority of studied failures violate old semantics
  - the testing coverage of these semantics is decent
  - the same semantics is repeatedly violated by different root causes
  - many failures violate long-lived semantics
  - ...

- Key idea:
  - extract essence from semantic failure regression tests and enforce it

Oathkeeper workflow



How to express semantics?

Predicates over key state variables:

$0 <= Sender <= N$
$\forall$ nodes i, j, $NodeState_i = NodeState_j$

Dinv[1]

$\forall l \in LockID, sizeof(Owners(l)) <= 1$

D3S[2]

Relationship among semantics-related events

(obtained from instrumentation)

```
public void serialize(...) {
+ logEvent(Type.Op,"serialize", ...);
  ...
```

```
Map<Long, HashSet<String>> ephemerals;
...
void killSession(long session, long zxid) {
   HashSet<String> list = ephemerals.remove(session);
+ logEvent(Type.State, "ephemerals", "killSession",
+                  ephemerals, ...);
...
```

Oathkeeper

[1] Inferring and asserting distributed system invariants. Grant et al. ICSE'18.
[2] D3S: Debugging deployed distributed systems. Liu et al. NSDI'08.

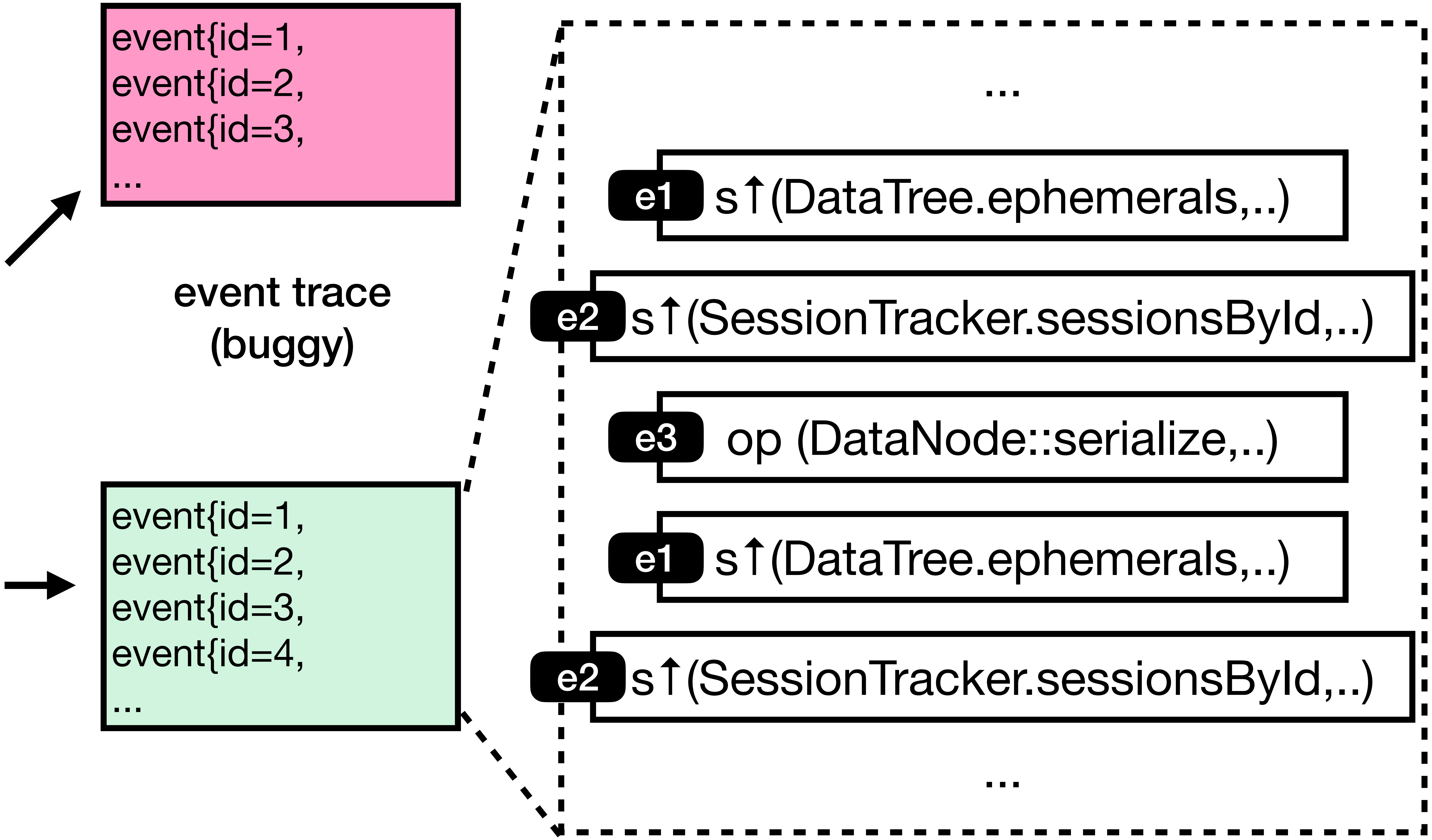
Emitting semantic event traces

```
void testCreateAfterCloseShouldFail() {

  // open a connection
  ConnectRequest conReq = new ...;
  // close connection
  RequestHeader h = new ...;
  // create ephemeral znode (race)
  CreateRequest createReq = new...;

  assertEquals(1, zk.getChildren("/").size()); }
```

regression test
(ZK-1208)



event trace
(patched)

[e1, e2, e3, e1, e2]

General semantic rule templates

- Relation examples summarized from study

Template	Example
$p \Rightarrow q$	decommission a datanode should trigger reconstruction
$s\uparrow \Rightarrow p$	when datanode changes, associated watcher notifies clients
$s\uparrow \Rightarrow k\uparrow$	after session disconnection, ephemeral node is removed
$(s = c) \oplus q$	deny new requests after connections reach maxClientCnxns
$p + \Delta t \Rightarrow q$	read-only server should not provide write access
$s\uparrow \rightarrow q$	inserted data should expire after the TTL is reached.
$p \Rightarrow \odot(s\uparrow, k\uparrow)$	after snapshot renaming, either new snapshot creation and old snapshot deletion both

full template list included in our tech report

Inference example: p ⇒ q

- Assume all rules hold and filter rules if counterexamples found

input	[e1, e2, e3, e1, e2]

Inference example: p ⇒ q

- Assume all rules hold and filter rules if counterexamples found

input [e1, e2, e3, e1, e2]

pre-scan

<e1,e2>	<e2,e1>	<e1,e3>	<e3,e1>	<e2,e3>	<e3,e2>
0	0	0	0	0	0

Inference example: p ⇒ q

- Assume all rules hold and filter rules if counterexamples found

input [e1, e2, e3, e1, e2]

		<e1,e2>	<e2,e1>	<e1,e3>	<e3,e1>	<e2,e3>	<e3,e2>
pre-scan		0	0	0	0	0	0
scan	e1	1	0	1	0	0	0
	e2	0	1	1	0	1	0
	e3	0	1	0	1	0	1
	e1	1	0	1	0	0	1
	e2	0	1	1	0	1	0

Inference example: p ⇒ q

- Assume all rules hold and filter rules if counterexamples found

input [e1, e2, e3, e1, e2]

		<e1,e2>	<e2,e1>	<e1,e3>	<e3,e1>	<e2,e3>	<e3,e2>
pre-scan		0	0	0	0	0	0
scan	e1	1	0	1	0	0	0
	e2	0	1	1	0	1	0
	e3	0	1	0	1	0	1
	e1	1	0	1	0	0	1
	e2	0	1	1	0	1	0
post-scan		e1 ⇒ e2			e3 ⇒ e1		e3 ⇒ e2

Validation example: p ⇒ q

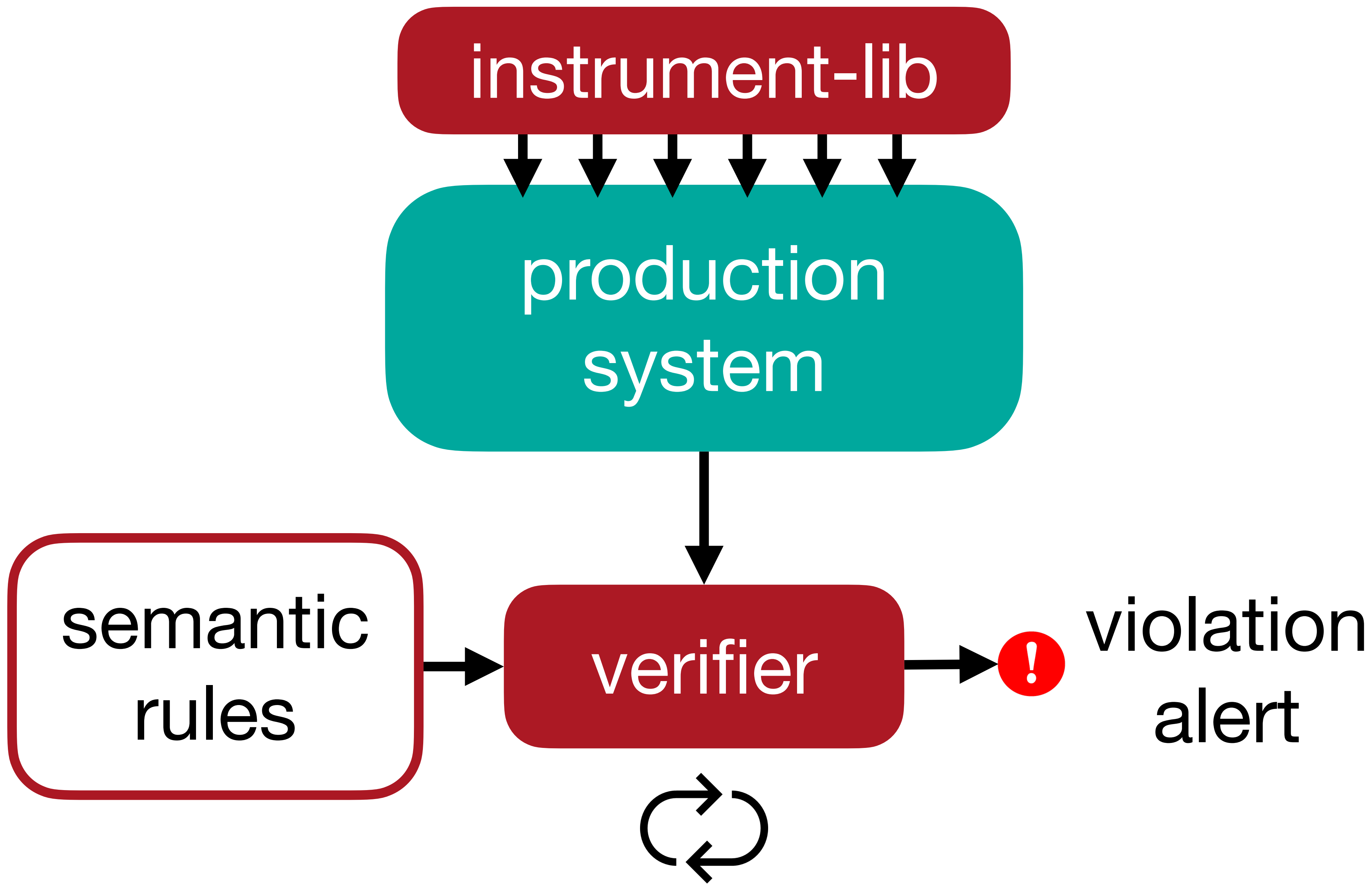
- Only preserve rules that are violated in buggy trace

input		[e1, e2, e3, e1]		
pre-scan		e1 ⇒ e2	e3 ⇒ e1	e3 ⇒ e2
		0	0	0
scan	e1	1	0	0
	e2	0	0	0
	e3	0	1	1
	e1	1	0	1
post-scan		e1 ⇒ e2		e3 ⇒ e2

Validation against all tests

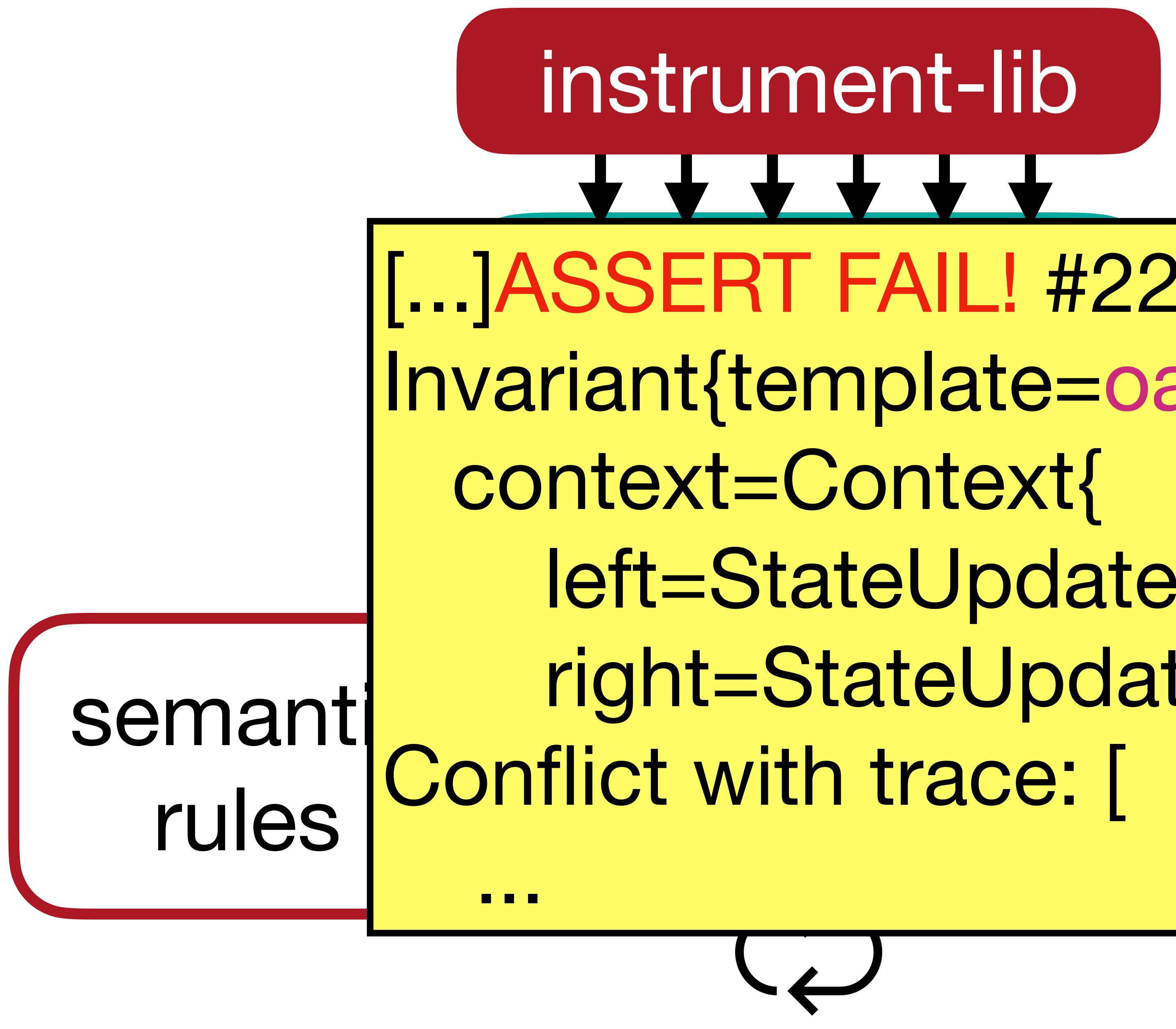
- False rules may still remain after validating against buggy trace
- The verifier further validates rules against traces from all tests
  - mark rules without counterexamples as **verified**

Runtime detection



- In production, the target system is deployed with verifier and instrumentation library
- Only rule-related functions are instrumented
- Deployed semantic rules periodically validate against the runtime trace
  - report alerts in the log with counterexamples

Runtime detection



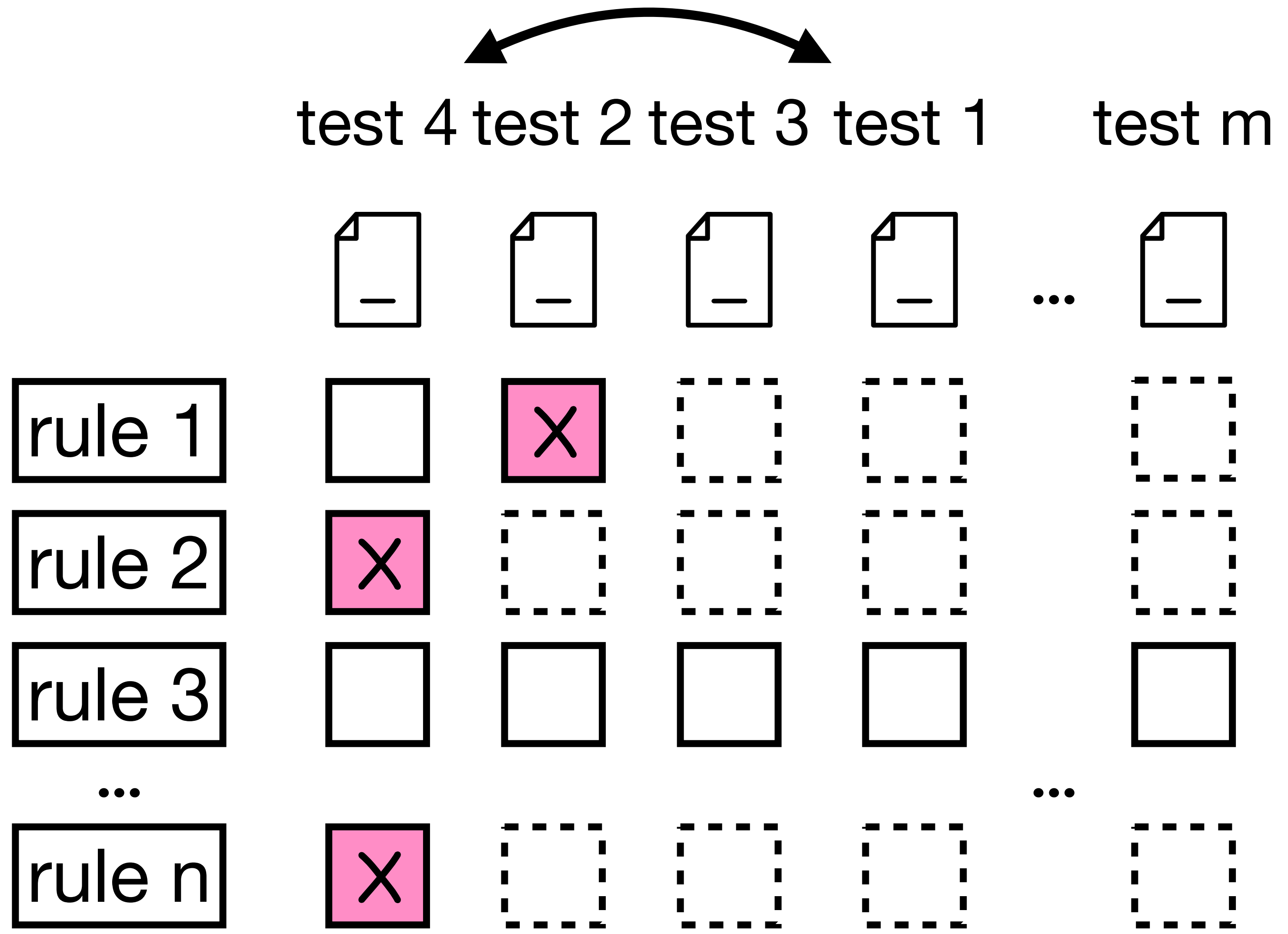
- In production, the target system is deployed with verifier and instrumentation library

```
[...]ASSERT FAIL! #220
Invariant{template=oathkeeper.runtime.template.StateUpdateImplyStateUpdateTemplate,
   context=Context{
       left=StateUpdateEvent{state='org.apache.zookeeper.server.DataTree.ephemerals'..},
       right=StateUpdateEvent{state='org.apache.zookeeper.server.SessionTracker.sessionsById'..}
Conflict with trace: [
    ...
```

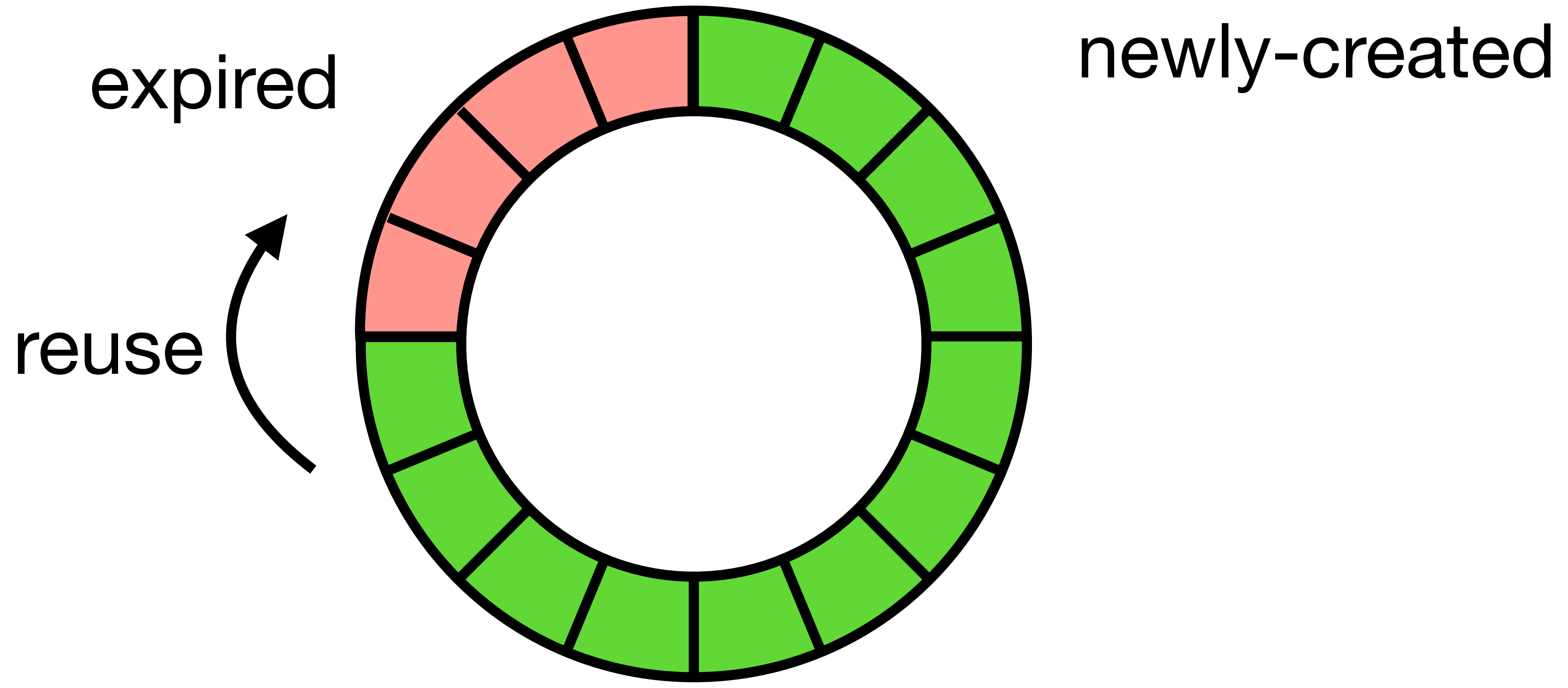
  - report alerts in the log with counterexamples

Optimizations

- "Survivor" mode for validation
  - prioritize running related tests to invalidate rules more efficiently
  - reduce validation processing time
- Ring buffer tracer for runtime
  - reuse expired event objects
  - effectively lower runtime overhead



survivor mode



ring buffer for tracer

Evaluation

- Integrated Oathkeeper with ZooKeeper, HDFS and Kafka
- We try to answer questions such as
  - can Oathkeeper check new violations from past failures?
  - is runtime checking accurate?
  - how fast can tool generate rules?
  - is runtime checking lightweight?

Extracted semantic rules

- We select old semantic failures and regression tests to reproduce
  - extracted 285 rules for ZooKeeper, 1,209 rules for HDFS, and 150 rules for Kafka

ZooKeeper	HDFS	Kafka
ZK-1046	HDFS-8950	KAFKA-9144
ZK-1208	HDFS-9204	KAFKA-9491
ZK-1412	HDFS-10192	KAFKA-9666
ZK-1573	HDFS-10536	KAFKA-9752
ZK-1754	HDFS-10968	KAFKA-9891
ZK-1755	HDFS-11960	KAFKA-9921
ZK-2680	HDFS-12862	KAFKA-10001
ZK-2797	HDFS-13120	KAFKA-10545
	HDFS-13192	
	HDFS-14504	

Detecting real-world failures

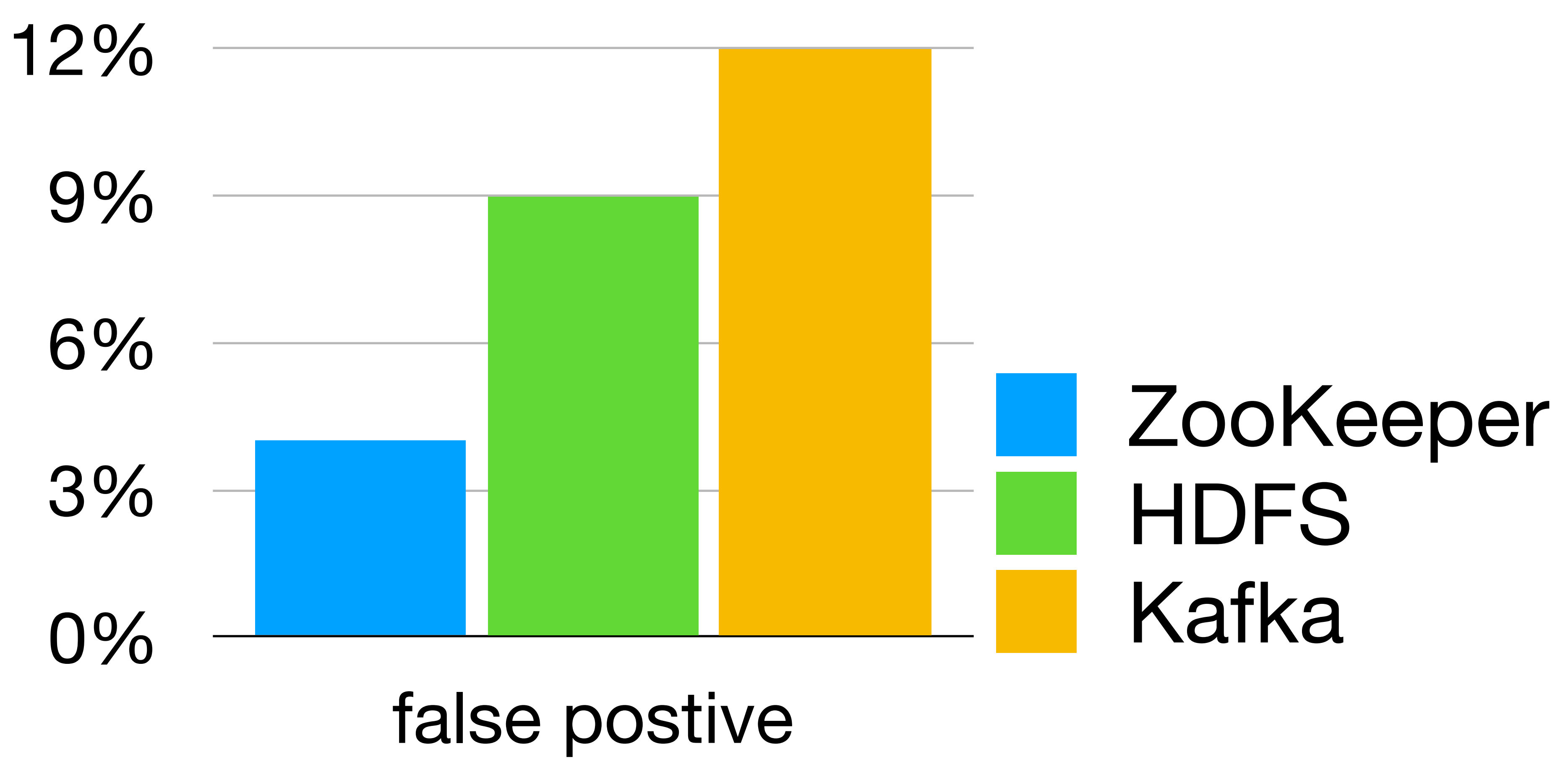
- Oathkeeper detects violations for 6 of 7 evaluated cases
  - use regression tests 9–34 months earlier than new failures
  - baseline checker based on Dinv[1] only detects 1 case

JIRA Id	Violated Semantics	Rules from
ZK-1496	ephemeral node should be deleted after session expired	ZK-1208
ZK-1667	watcher should return correct event when client reconnected	MISS
ZK-3546	container node should be deleted after children all removed	ZK-2705
HDFS-14699	failed block need to be reconstructed	HDFS-10968
HDFS-14317	edit log rolling should be activated periodically	HDFS-10536
HDFS-14633	file rename should respect storageType quota	HDFS-14504
KAFKA-12426	partition topic ID should be persisted into metadata file	KAFKA-10545

[1] Inferring and asserting distributed system invariants. Grant et al. ICSE'18.

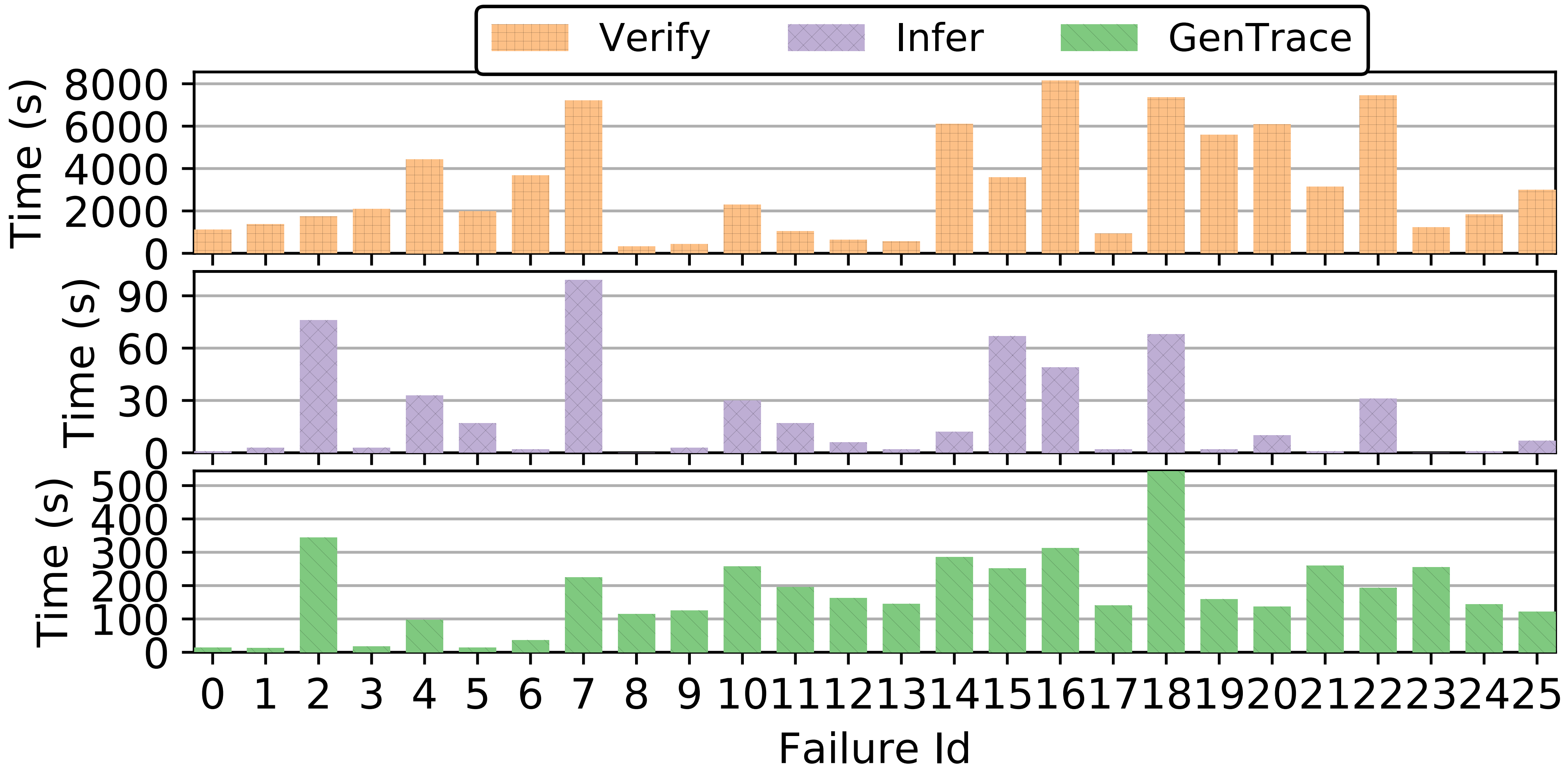
False positive

- Generated rules incur 4-12% false positive ratios
  - greatly benefits from the validation steps
  - can be further reduced by adding profile runs or a dynamic ban mechanism



Offline performance

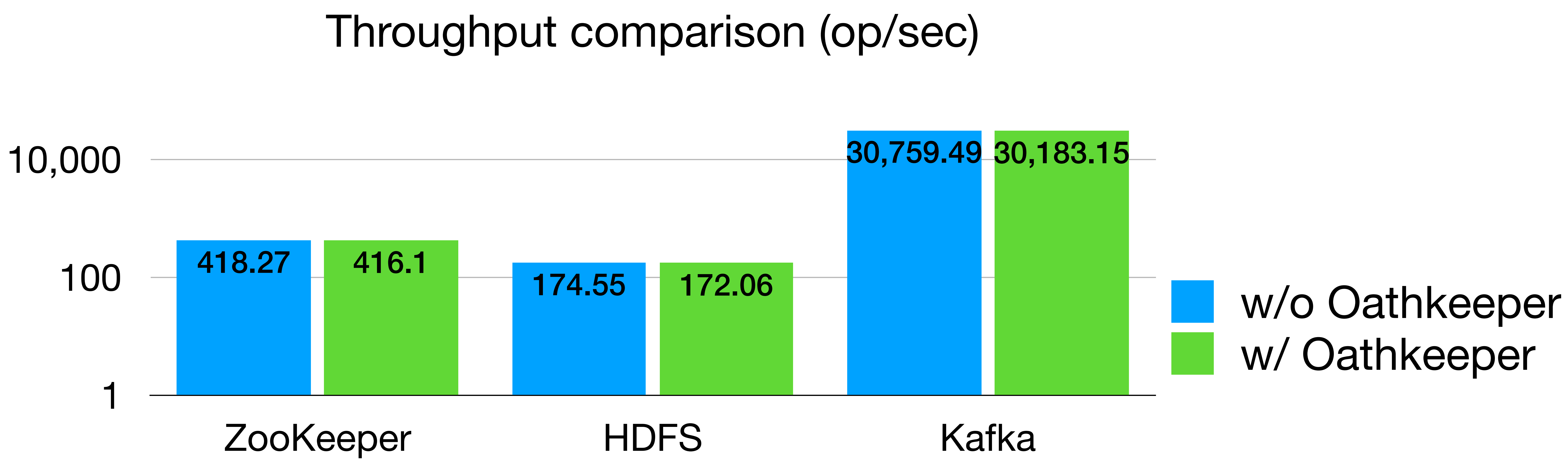
- Trace generation and inference usually take up to minutes
- Validation is most time-consuming part
  - survivor mode can reduce validation time by 38%



Phase	Median time (sec)
trace generation	153.5
inference	6.5
validation	2,196

Runtime overhead

- Oathkeeper adds ~1.27% overhead on throughput
  - overhead is mainly from the added instrumentation to emit traces
  - ring buffer optimization eliminates overhead by frequent GC



Conclusion

- Semantics in distributed systems can be violated silently
- Our study reveals interesting findings
  - same old semantics can be violated repeatedly in different scenarios
  - long-lived semantics require continuous monitoring
- Oathkeeper: a runtime detection tool
  - infer semantic rules from past failures to detect new violations

https://github.com/OrderLab/OathKeeper

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE | ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL | ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED