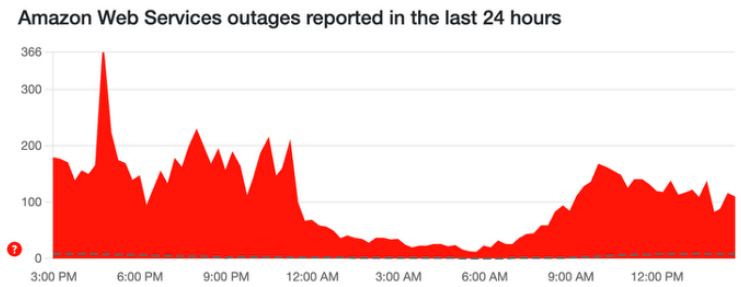
Simulating Failure Recovery *In Situ* for Production Distributed Systems

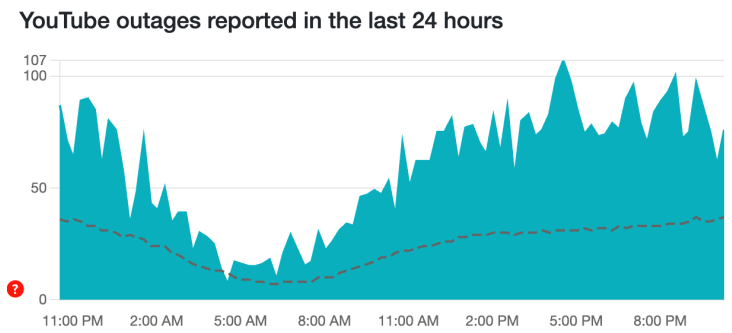
Zhenyu Li, Angting Cai, Chang Lou

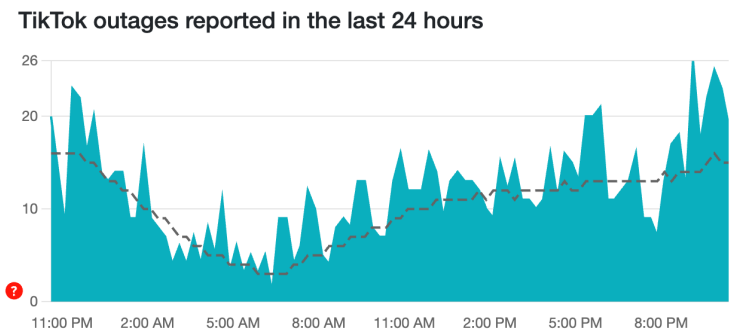
nsdi '26

University of Virginia

Failures are inevitable in cloud systems

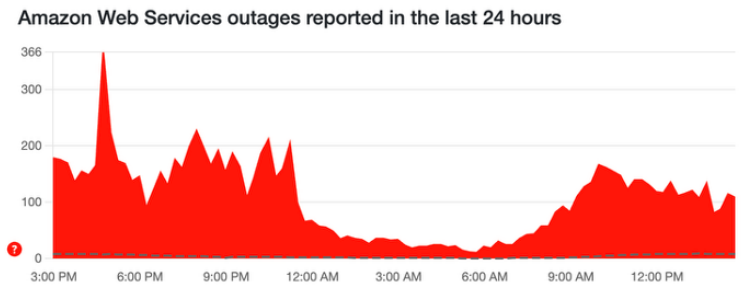


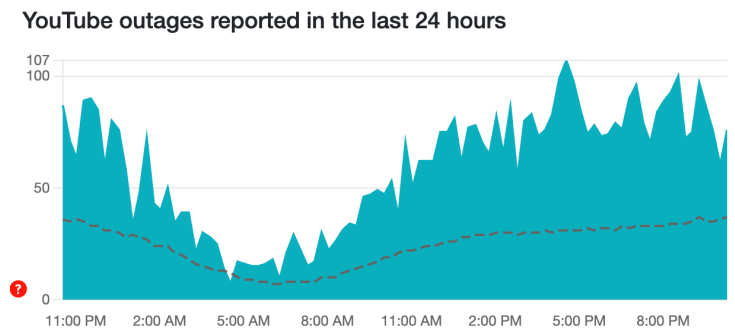


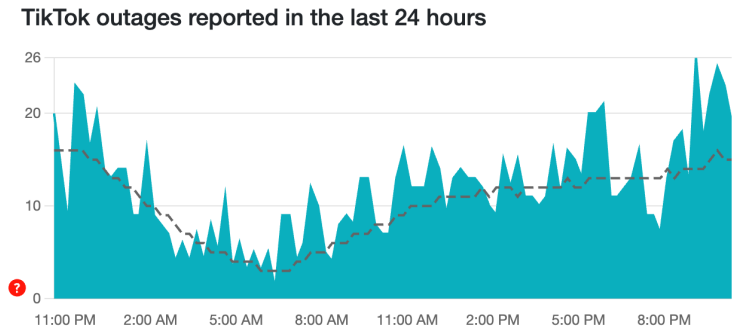


Statistics from downdetector.com

Failures are inevitable in cloud systems

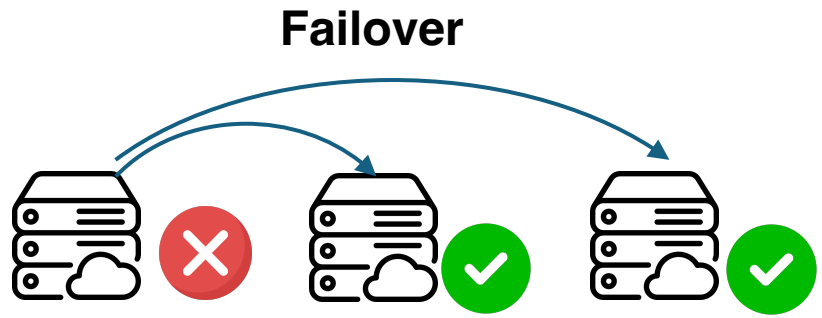


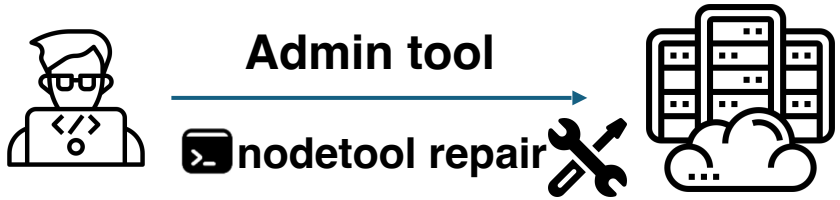




Statistics from downdetector.com

Correct recovery is critical





Ironically, recovery itself is error-prone in production

- Recovery failures: incorrect recovery actions that **fail to alleviate** failures, or even **amplify** consequences

Ironically, recovery itself is error-prone in production

- Recovery failures: incorrect recovery actions that **fail to alleviate** failures, or even **amplify** consequences

Kubernetes Liveness and Readiness Probes: How to Avoid Shooting Yourself in the Foot [1]

Nov 4, 2018

I expand on these ideas in my presentation Kubernetes Probes: How to Avoid Shooting Yourself in the Foot.

kubernetes

[1] https://blog.colinbreck.com/kubernetes-liveness-and-readiness-probes-how-to-avoid-shooting-yourself-in-the-foot/

Ironically, recovery itself is error-prone in production

- Recovery failures: incorrect recovery actions that **fail to alleviate** failures, or even **amplify** consequences

Kubernetes Liv
Readiness Prol
Avoid Shooting
Foot [1]

Nov 4, 2018

I expand on these ideas in my pres
to Avoid Shooting Yourself in the

Summary of Windows Azure Service Disruption on Feb 29th

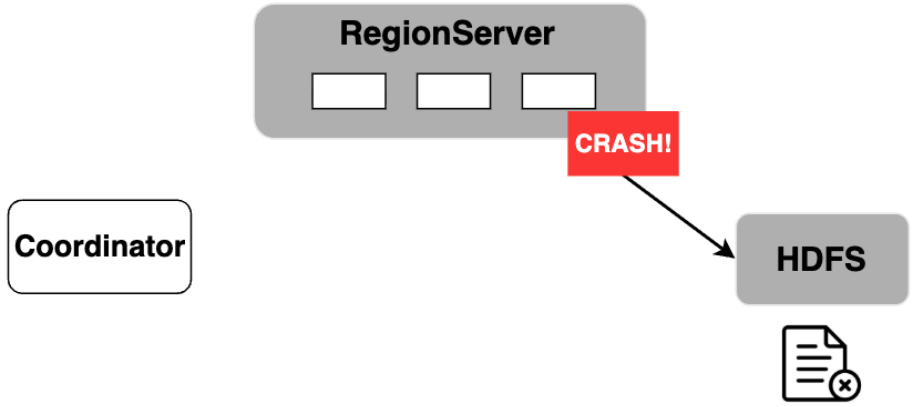
its standard autonomic failure recovery operations for a server in the HI state, the FC will service heal any VMs that were assigned to the failed server by reincarnating them to other servers. In a case like this, when the VMs are moved to available servers the leap day bug will reproduce during GA initialization, resulting in a cascade of servers that move to HI.

kubernetes

Azure

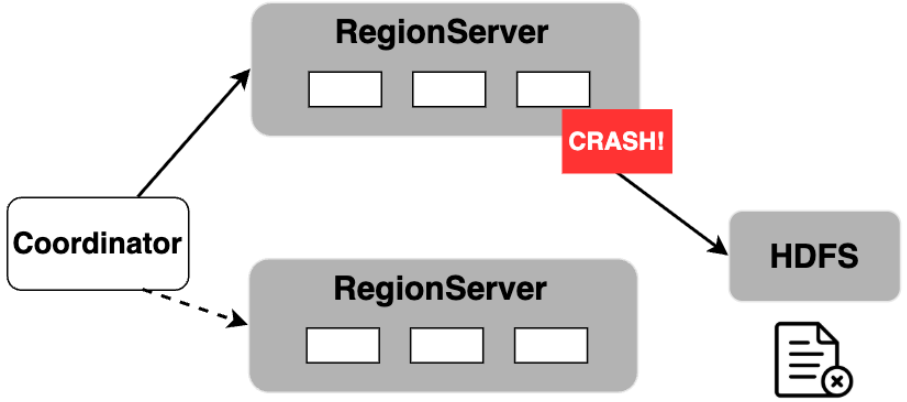
[1] https://blog.colinbreck.com/kubernetes-liveness-and-readiness-probes-how-to-avoid-shooting-yourself-in-the-foot/
[2] https://azure.microsoft.com/en-us/blog/summary-of-windows-azure-service-disruption-on-feb-29th-2012/

A real-world recovery failure in HBase production clusters



(1) RegionServer crashed while replicating a buggy log file

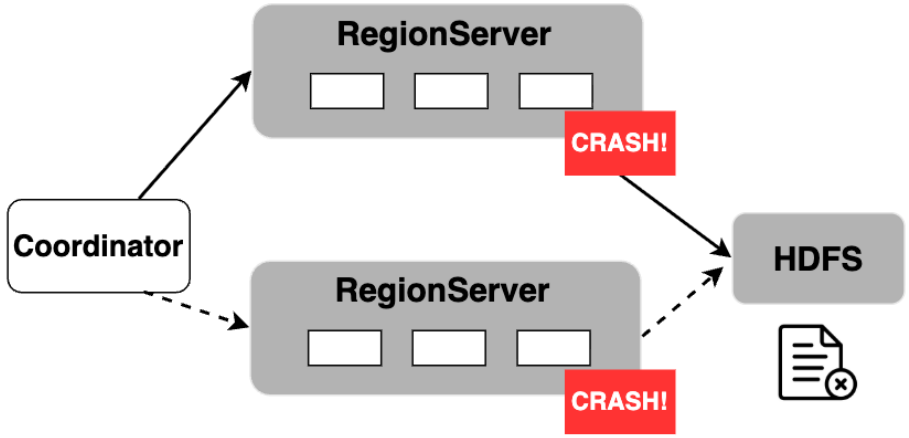
A real-world recovery failure in HBase production clusters



(1) RegionServer crashed while replicating a buggy log file

(2) Coordinator initiated the recovery

A real-world recovery failure in HBase production clusters

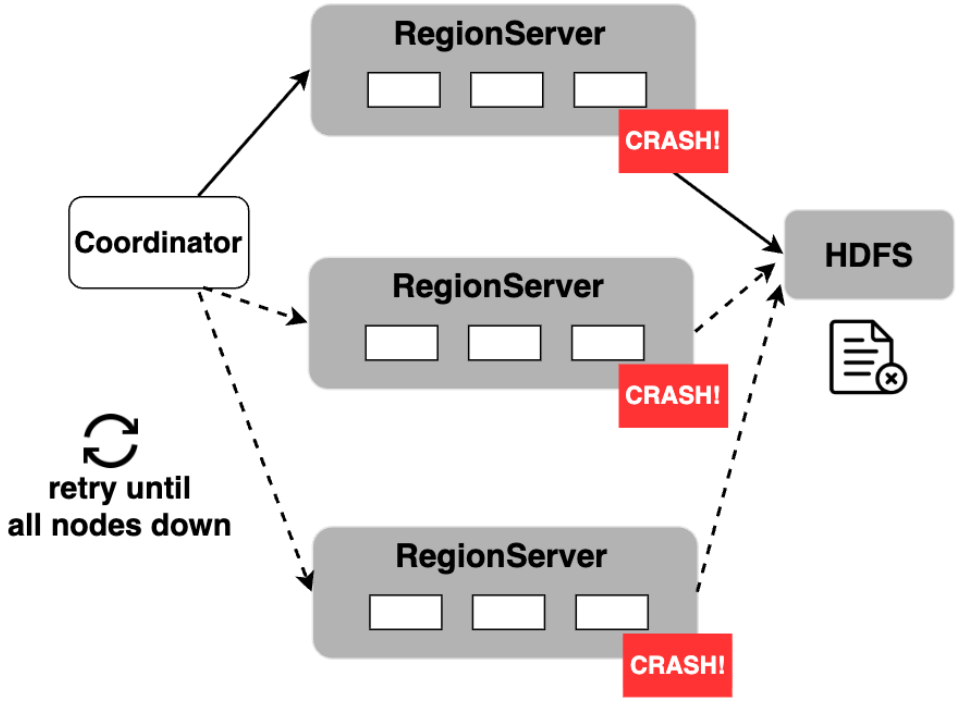


(1) RegionServer crashed while replicating a buggy log file

(2) Coordinator initiated the recovery

(3) RegionServer crashed again due to the resumed replication task!

A real-world recovery failure in HBase production clusters



(1) RegionServer crashed while replicating a buggy log file

(2) Coordinator initiated the recovery

(3) RegionServer crashed again due to the resumed replication task!

Without proper checking, recovery can become a deadly source of new failures!

(4) The whole cluster would be down with cascading failures**!**

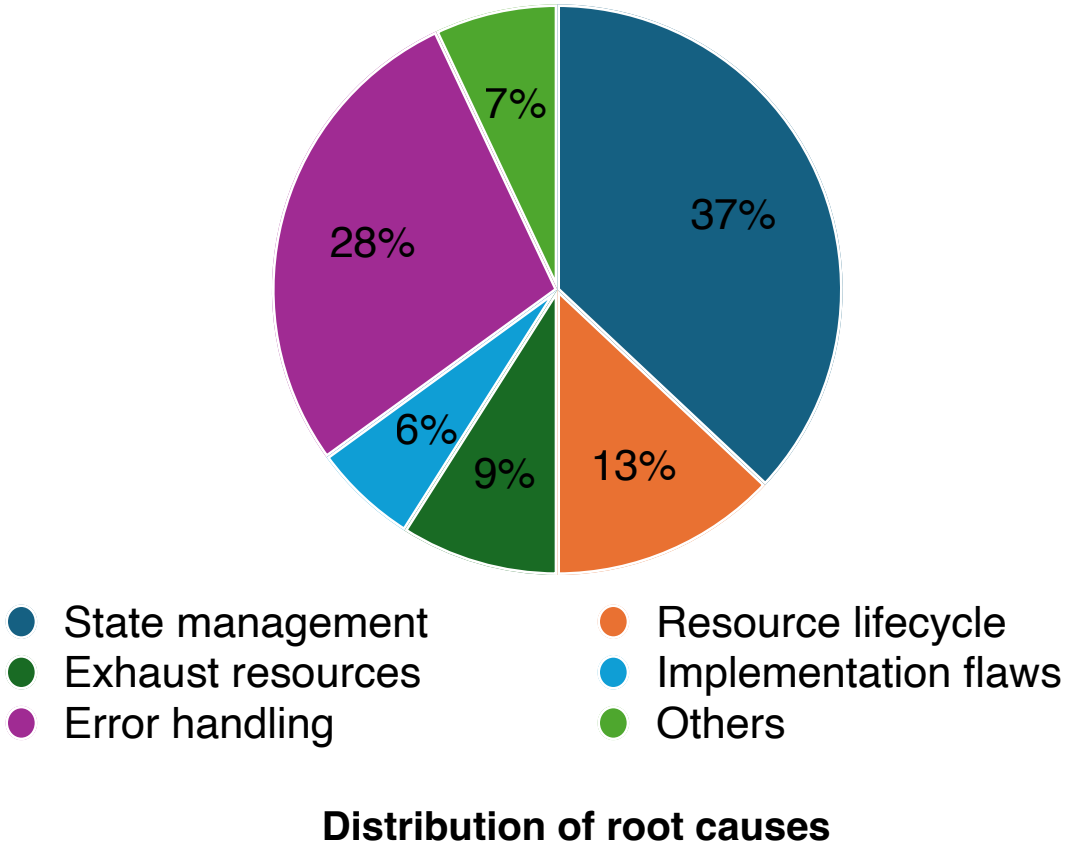
Failure study

- We studied **75** production incidents induced by recovery from **8 systems**
  - Recovery failures are **prevalent**
  - Recovery failures often lead to **severe consequences**

Software	Lang.	Category	Date	Sampled
HDFS	Java	File Sys.	2014-2024	10
HBase	Java	Database	2010-2025	15
YARN	Java	Resource Mgr.	2014-2025	5
TiDB	Go	Database	2019-2025	5
Kafka	Scala	Streaming	2017-2024	10
Cassandra	Java	Database	2011-2024	15
RabbitMQ	Erlang	Message Bkr.	2016-2024	5
Mesos	C++	Resource Mgr.	2015-2022	10

Study findings (selected)

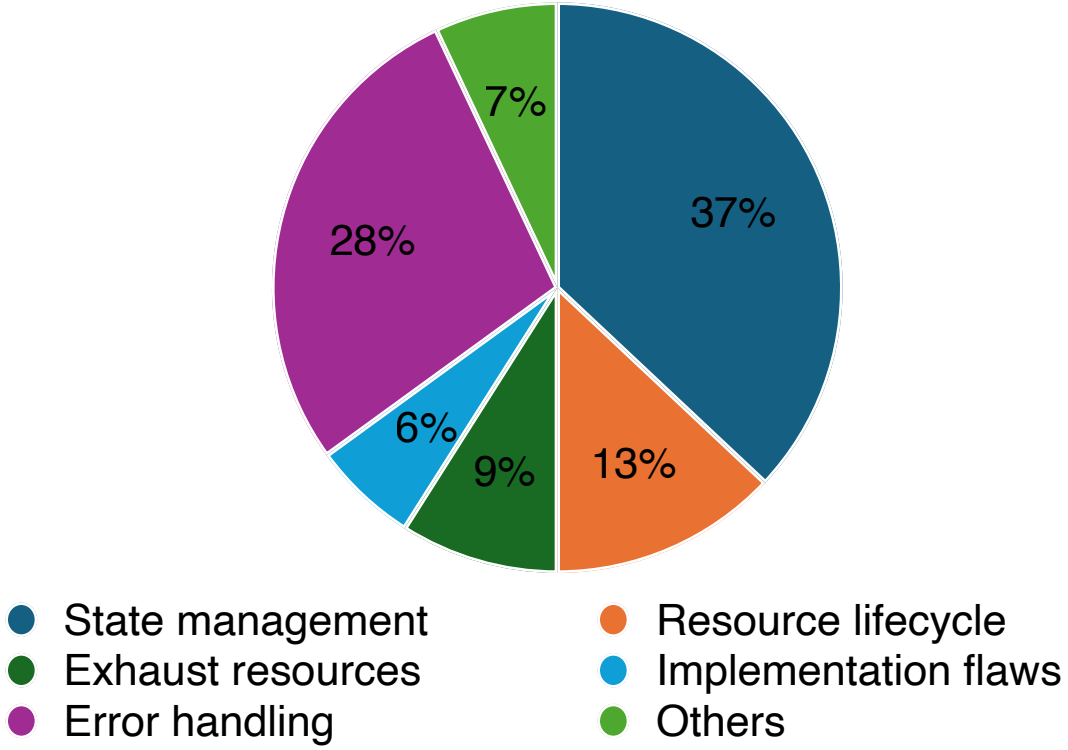
- Finding 1: The root causes of recovery failures are diverse



Distribution of root causes

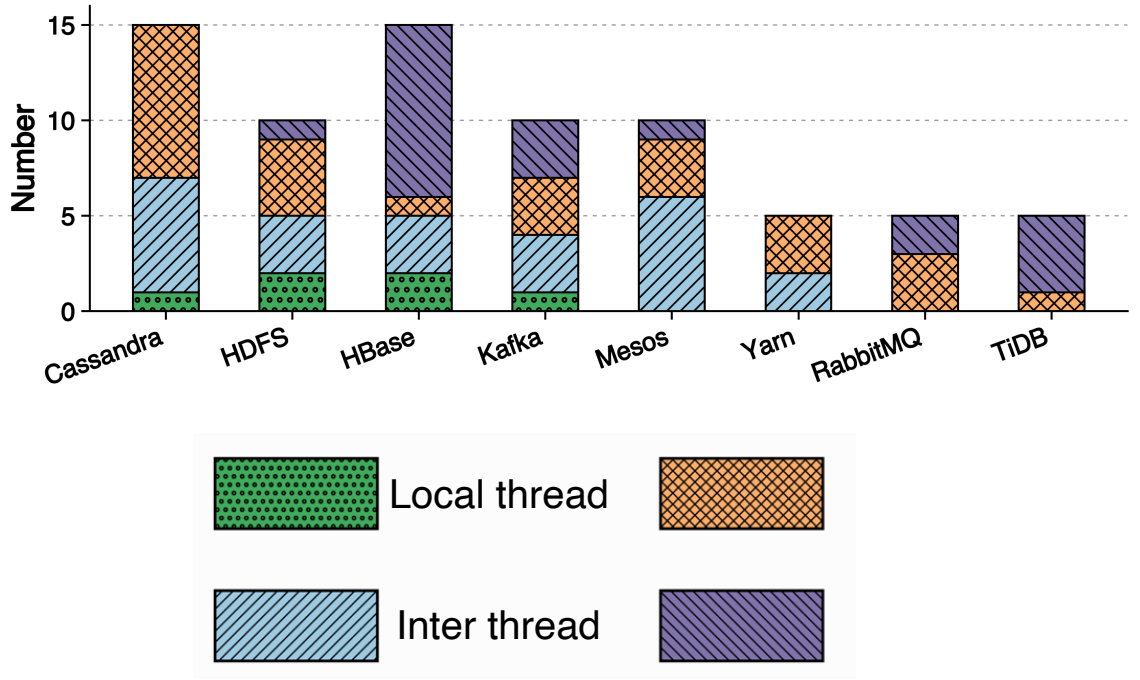
Study findings (selected)

- Finding 1: The root causes of recovery failures are diverse



Distribution of root causes

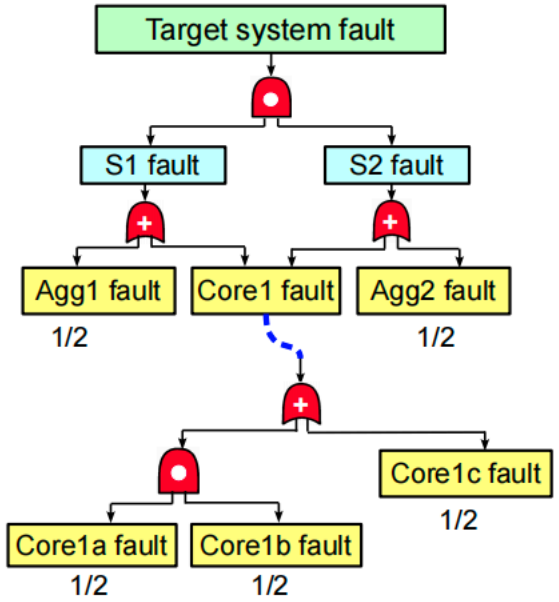
- Finding 2: Most recovery failures (92%) manifest beyond local thread scope



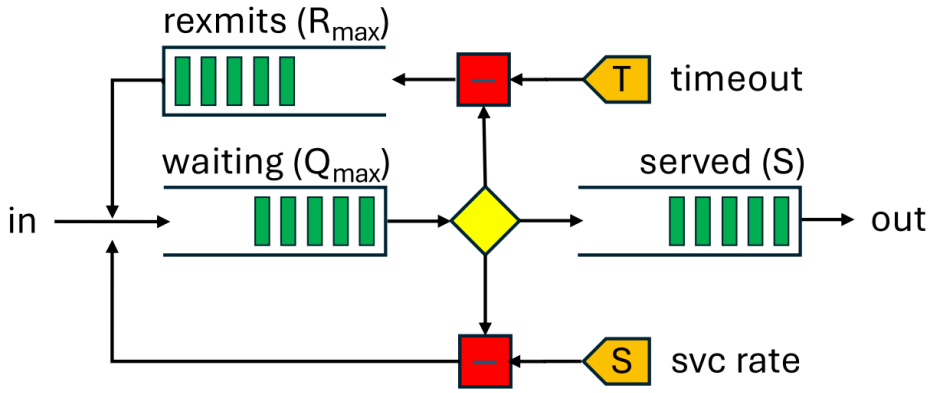
Distribution of the scope of recovery failures

Existing work

- **Offline** approaches model the system or test before deployment
  - Miss **low-level implementation bugs**
  - Miss **issues caused by dynamic interactions** (Finding 1, Finding 2)



CloudCanary [NSDI'20]

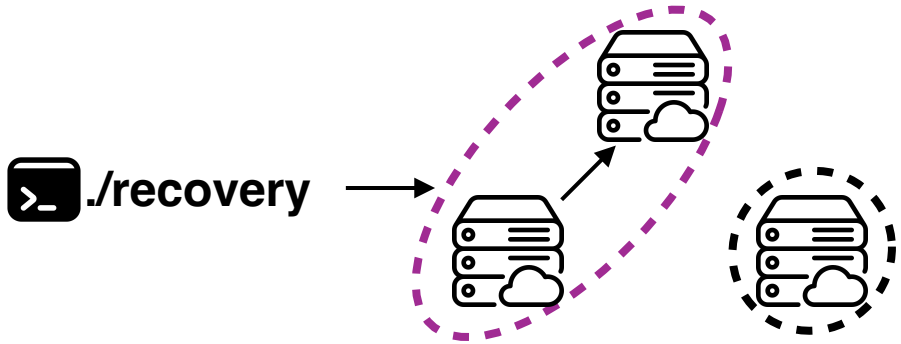


Metastability [HotNets'25]

Existing work

- **Offline** approaches model the system or test before deployment
  - Miss **low-level implementation bugs**
  - Miss **issues caused by dynamic interactions** (Finding 1, Finding 2)

- **Online** approaches predict failures using runtime information
  - Different node setups and workloads could cause **inaccuracy**
  - Lack built-in **safety mechanism**

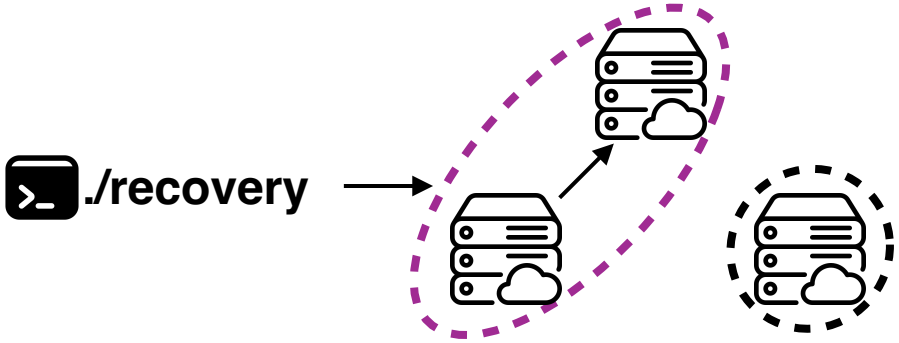


Narya [OSDI'20]

Existing work

- **Offline** approaches model the system or test before deployment
  - Miss **low-level implementation bugs**
  - Miss **issues caused by dynamic interactions** (Finding 1, Finding 2)

- **Online** approaches predict failures using runtime information
  - Different node setups and workloads could cause **inaccuracy**
  - Lack built-in **safety mechanism**

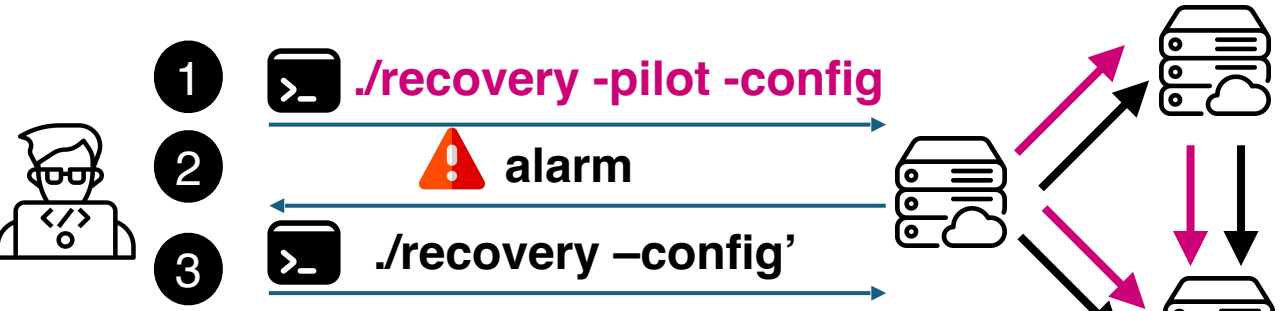


Narya [OSDI'20]

Can we simulate recovery accurately and safely?

Pilot Execution

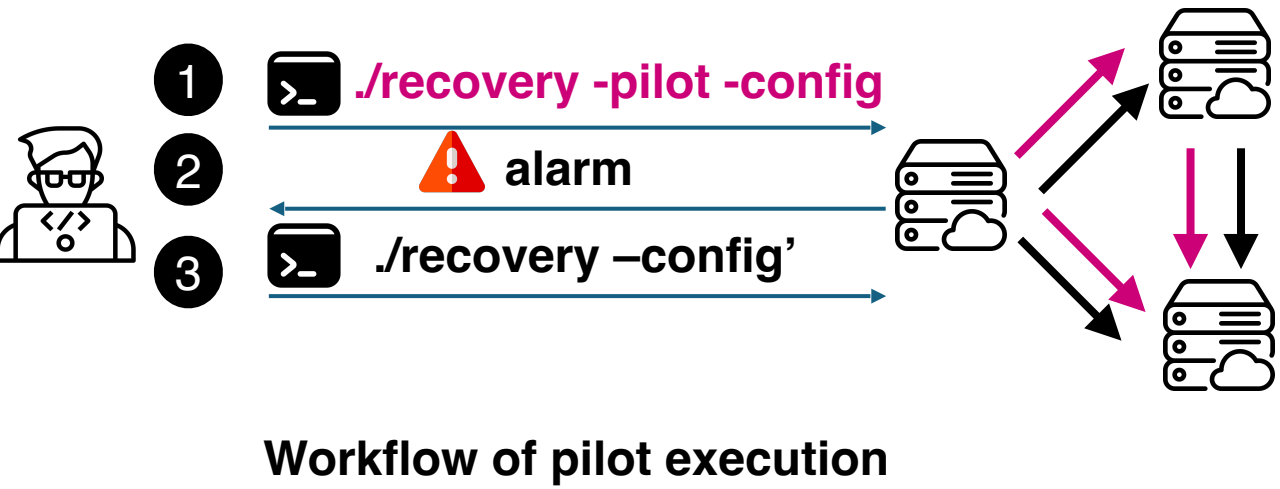
- Preview recovery before committing it, so operators can validate its consequences



Workflow of pilot execution

Pilot Execution

- Preview recovery before committing it, so operators can validate its consequences



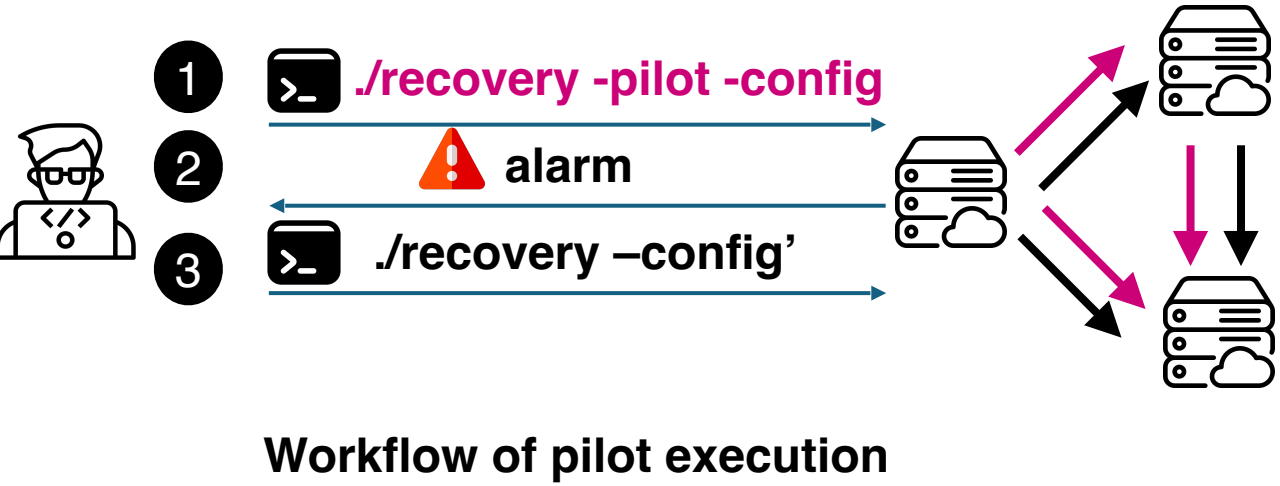
Workflow of pilot execution

Speculative execution treats early execution as an optimization

Pilot execution treats early execution as a safety check for recovery

Pilot Execution

- Preview recovery before committing it, so operators can validate its consequences

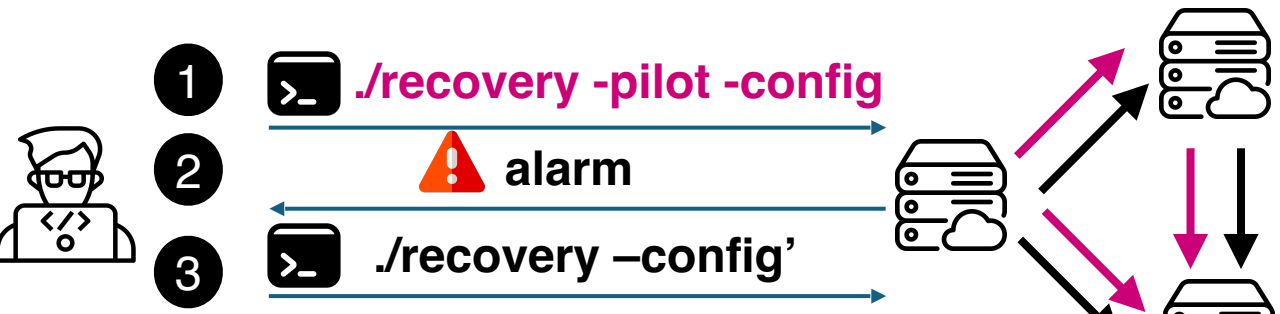


Workflow of pilot execution

Challenge1
Run in production without disruption

Pilot Execution

- Preview recovery before committing it, so operators can validate its consequences

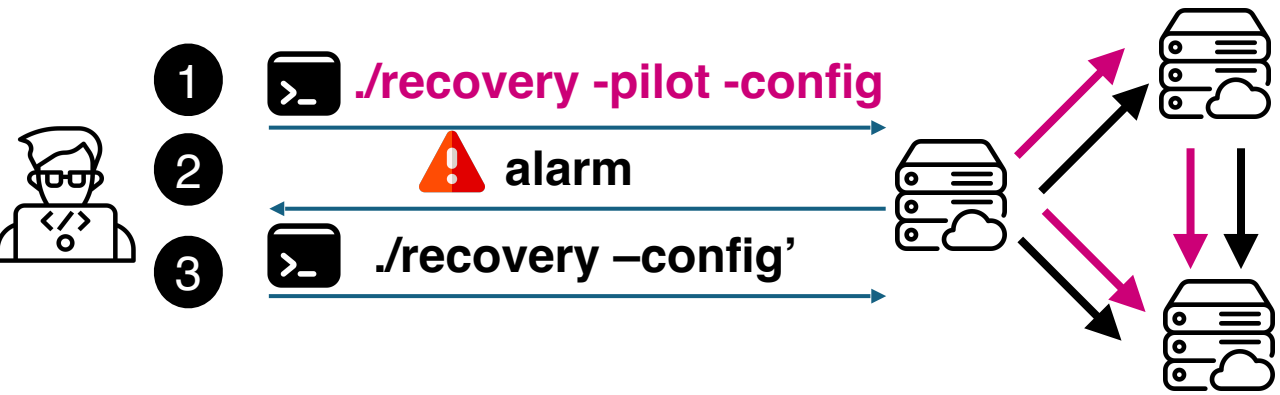


Workflow of pilot execution

Challenge1
Run in production without disruption
Isolate simulation with phantom threads

Pilot Execution

- Preview recovery before committing it, so operators can validate its consequences



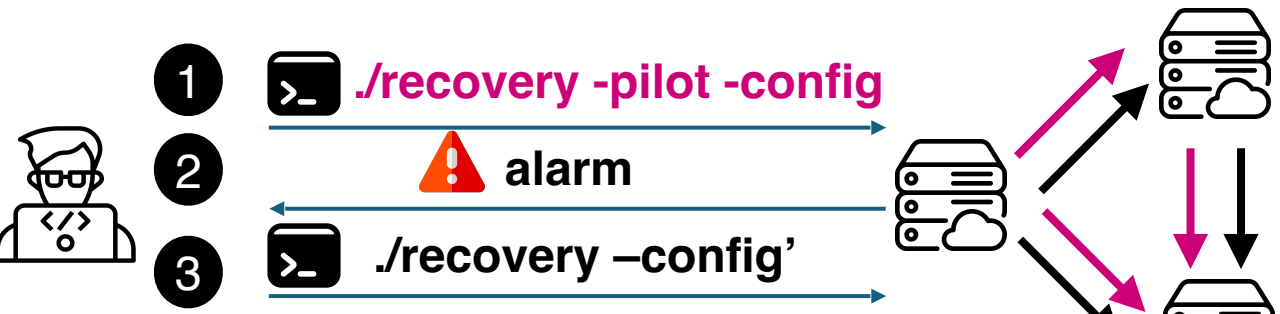
Workflow of pilot execution

Challenge1
Run in production without disruption
Isolate simulation with phantom threads

Challenge 2
Simulate recovery in distributed environment

Pilot Execution

- Preview recovery before committing it, so operators can validate its consequences



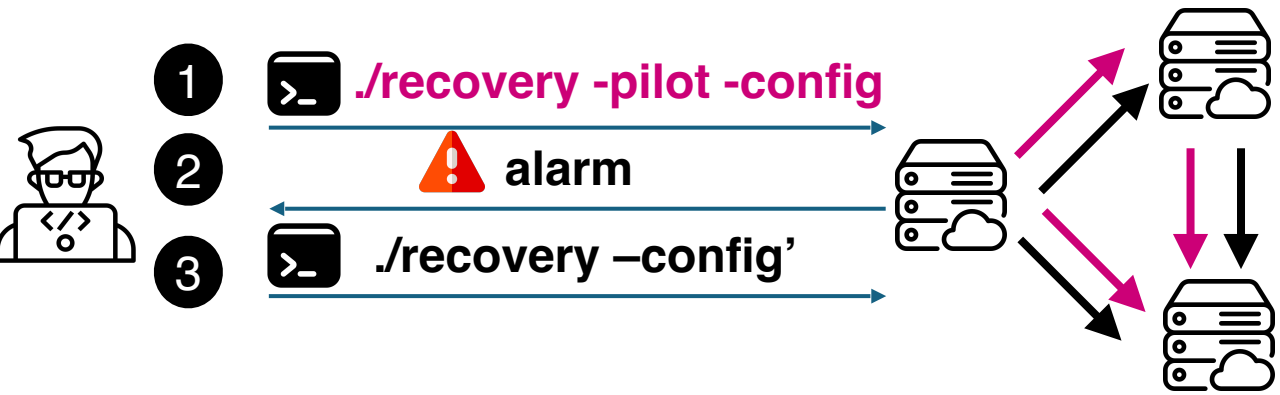
Workflow of pilot execution

Challenge1
Run in production without disruption
Isolate simulation with phantom threads

Challenge 2
Simulate recovery in distributed environment
Propagate execution with tracked metadata

Pilot Execution

- Preview recovery before committing it, so operators can validate its consequences



Workflow of pilot execution

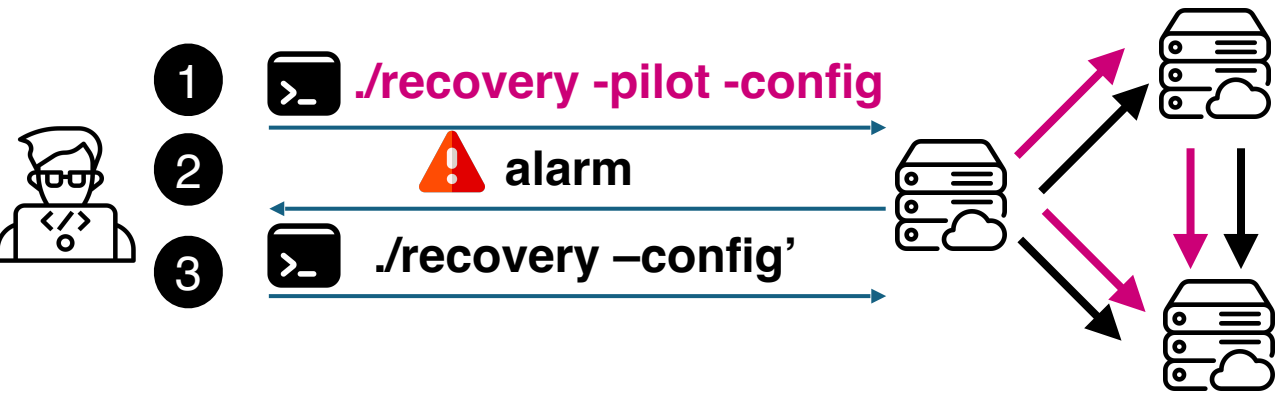
Challenge1
Run in production without disruption
Isolate simulation with phantom threads

Challenge 2
Simulate recovery in distributed environment
Propagate execution with tracked metadata

Challenge 3
Limit delay to the original recovery

Pilot Execution

- Preview recovery before committing it, so operators can validate its consequences



Workflow of pilot execution

Challenge1
Run in production without disruption
Isolate simulation with phantom threads

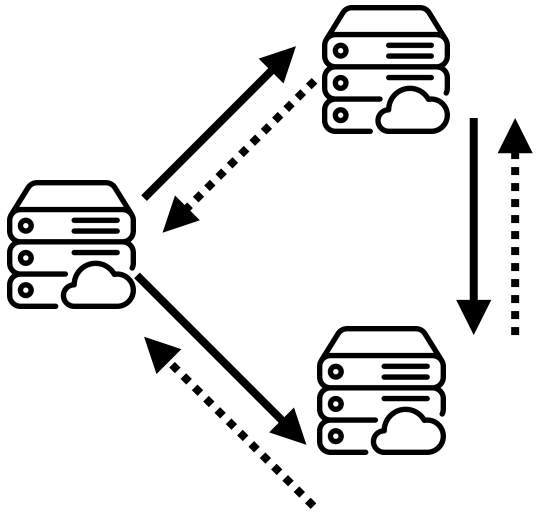
Challenge 2
Simulate recovery in distributed environment
Propagate execution with tracked metadata

Challenge 3
Limit delay to the original recovery
Accelerate original recovery with caching

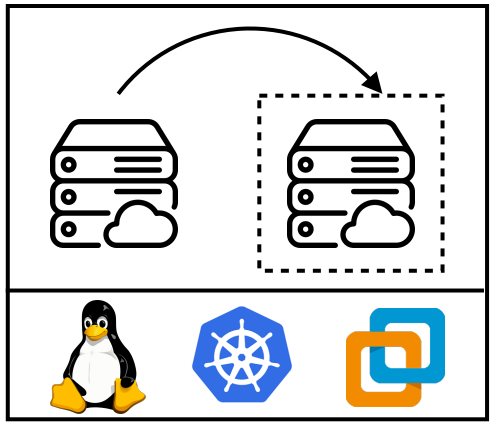
Challenge 1 Pilot execution should not disrupt production environment

Challenge 1 Pilot execution should not disrupt production environment

- Recovery spans distributed side effects, making rollback hard to implement
- Virtualization provides isolation, but incurs high overhead and weakens fidelity



Transactional rollback

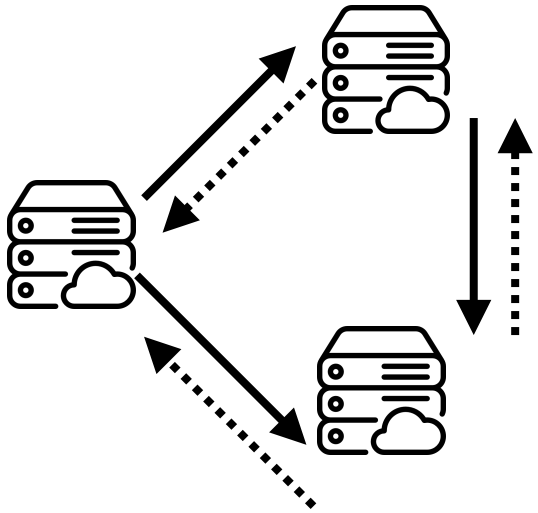


Isolate recovery in virtualized environment

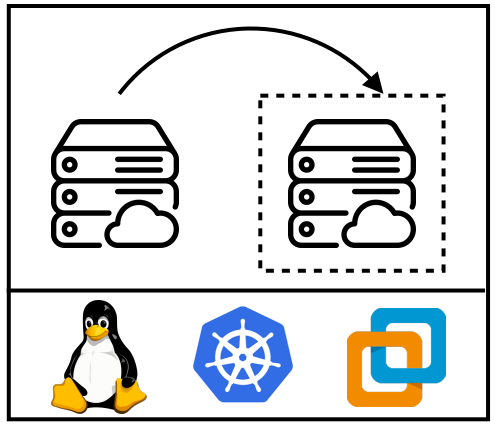
Challenge 1 Pilot execution should not disrupt production environment

- Recovery spans distributed side effects, making rollback hard to implement
- Virtualization provides isolation, but incurs high overhead and weakens fidelity

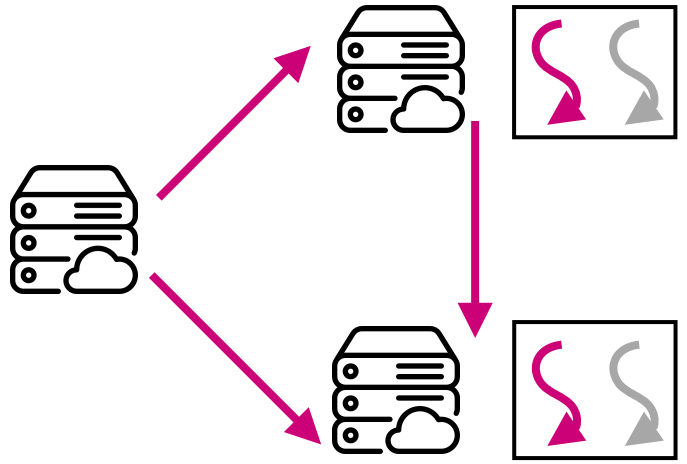
- **In-Situ simulation**: Use dedicated threads for pilot execution



Transactional rollback



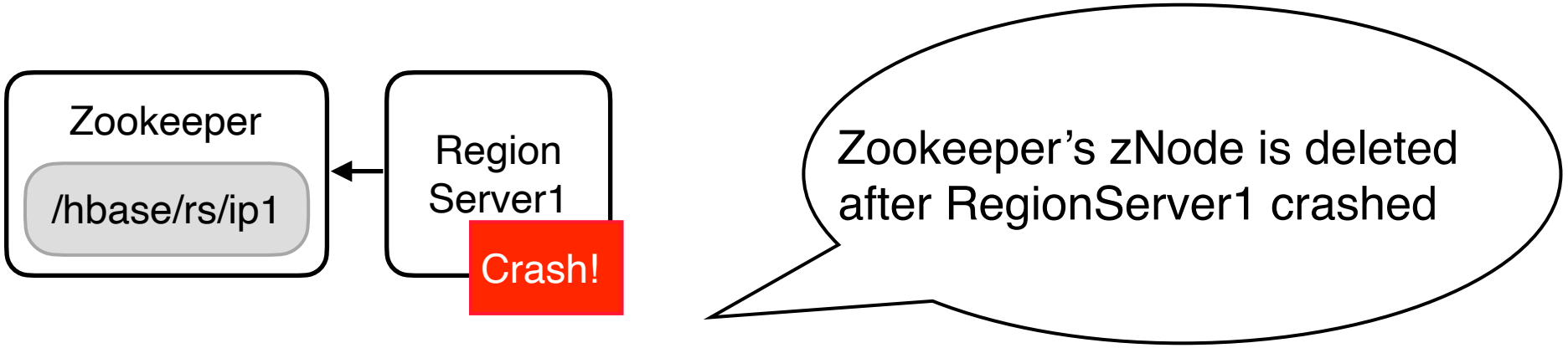
Isolate recovery in virtualized environment



In-Situ simulation

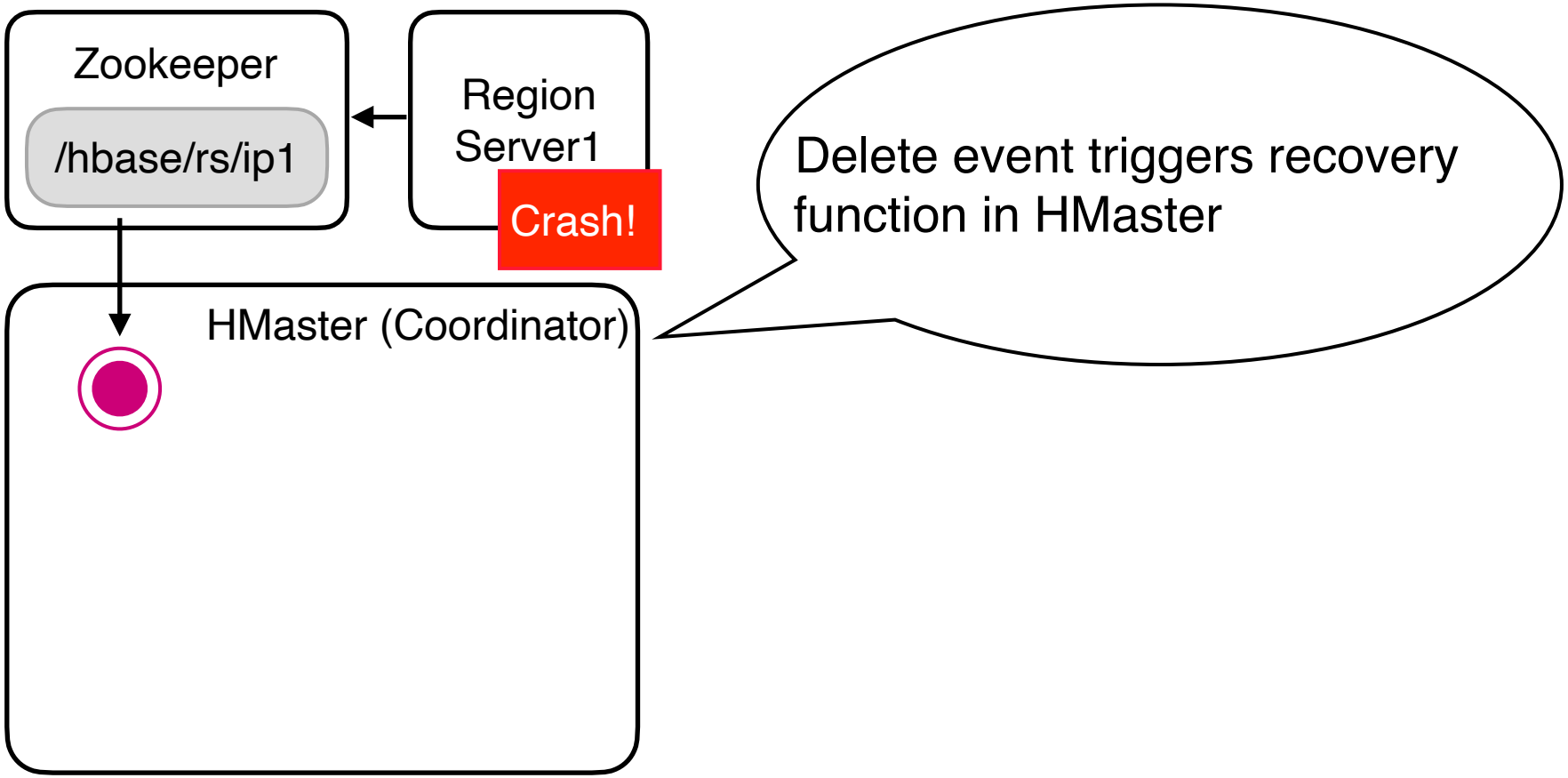
Initiate phantom threads

- Instrument the recovery entry point to launch pilot execution



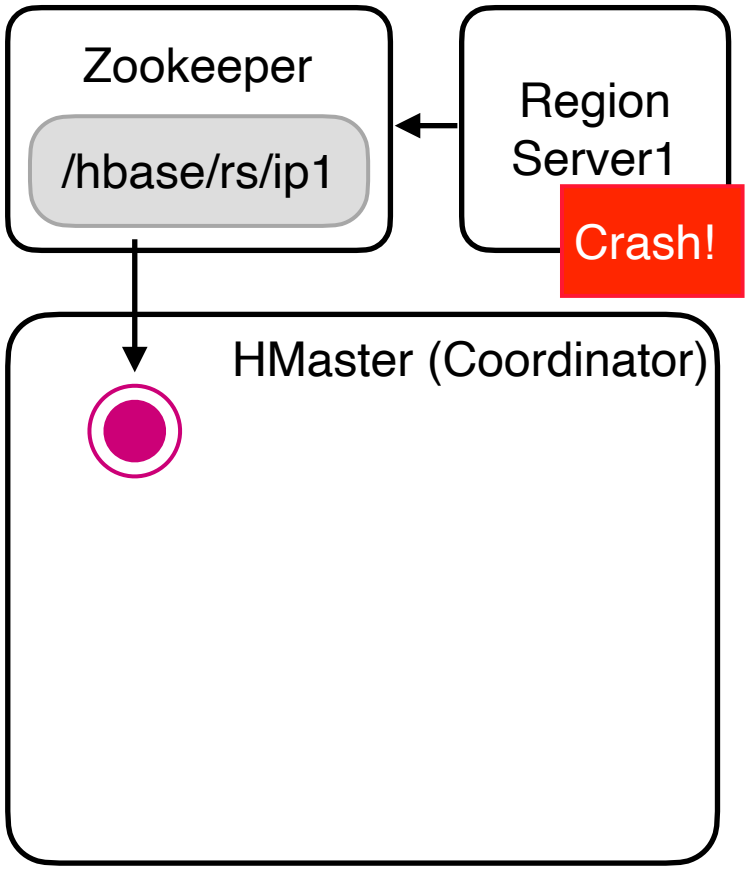
Initiate phantom threads

- Instrument the recovery entry point to launch pilot execution



Initiate phantom threads

- Instrument the recovery entry point to launch pilot execution

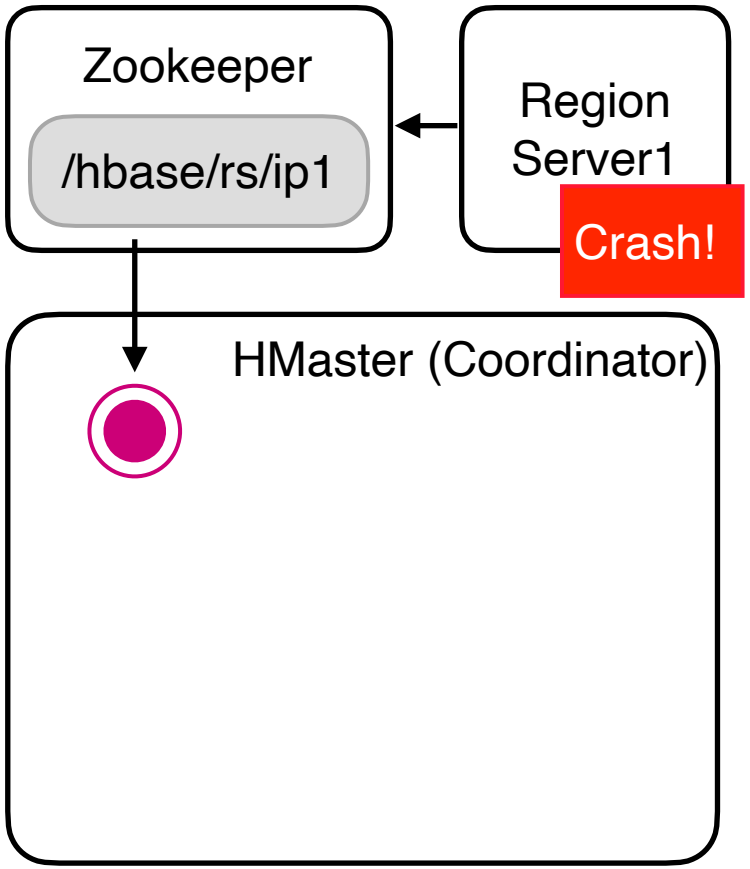


Entry point of HBase recovery

```
@ZooKeeperCallback
public void failover() {
  transferQueue(server);
  this.server = getFailoverTarget();
  //…
  File.write(path, content);
  ZooKeeperClient.deleteNode(path);
  //…
}
```

Initiate phantom threads

- Instrument the recovery entry point to launch pilot execution



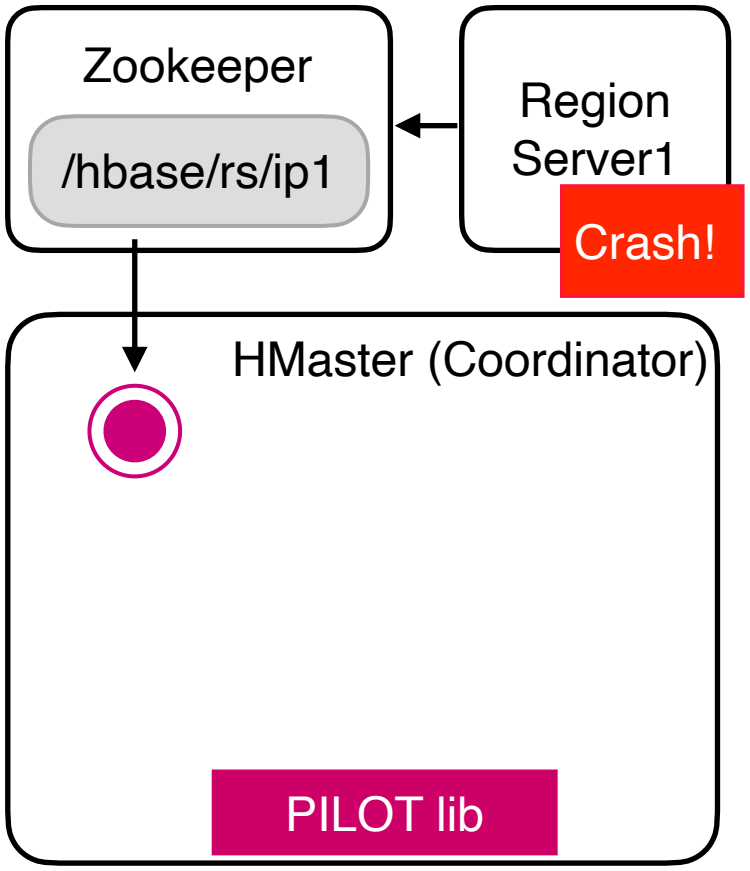
Entry point of HBase recovery

```
@ZooKeeperCallback
public void failover() {
  transferQueue(server);
  this.server = getFailoverTarget();
  //…
  File.write(path, content);
  ZooKeeperClient.deleteNode(path);
  //…
}
```

Divert into pilot execution

Initiate phantom threads

- Instrument the recovery entry point to launch pilot execution

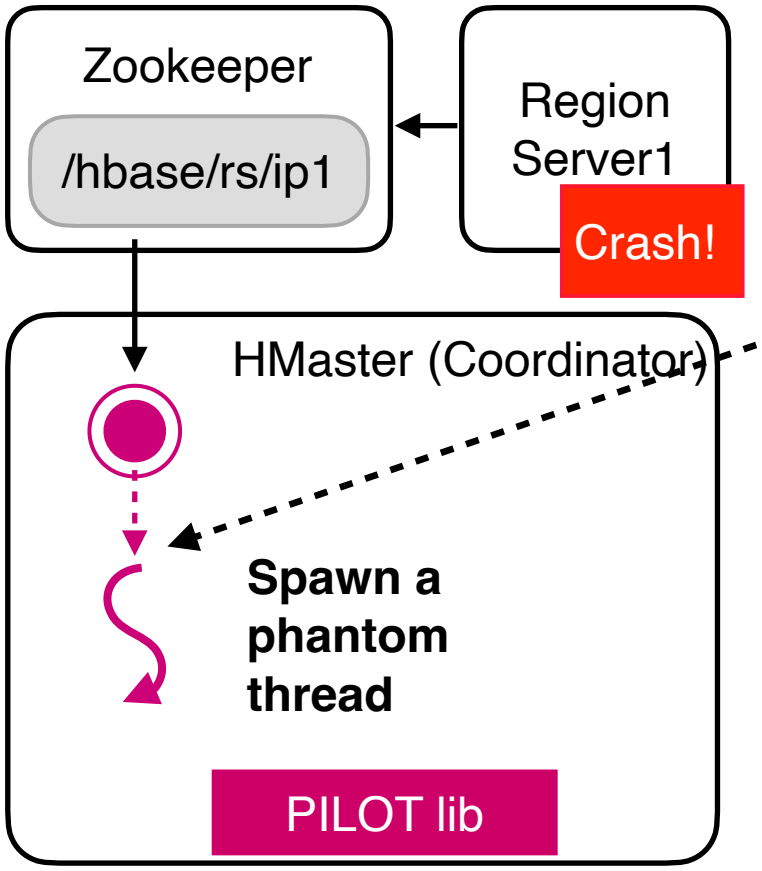


Entry point of HBase recovery

```
@ZooKeeperCallback
public void failover() {Divert into pilot execution
+    if (pilotMode) {
+        RunId id = Pilot.start(
+            this::failover$pilot, policy);
+        if (Pilot.waitFinishOrAbort(id)
+                == FAILED) {
+            Pilot.report(id);
+            return;
+        }
+    }
   //Original recovery code
}
+public void failover$pilot(){…}
```

Initiate phantom threads

- Instrument the recovery entry point to launch pilot execution



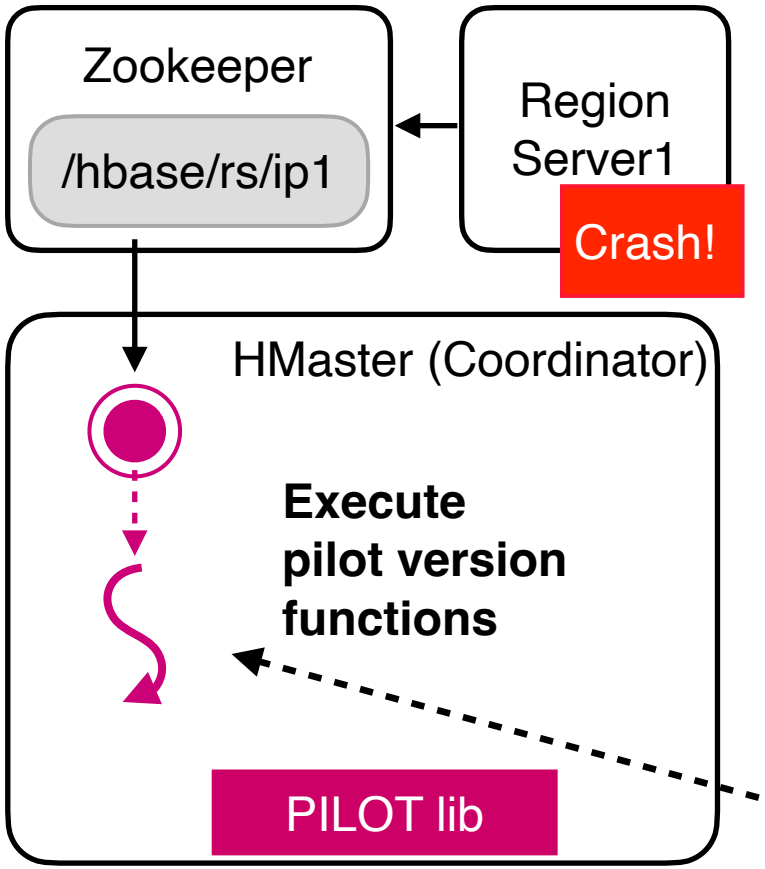
Entry point of HBase recovery

Divert into pilot execution

```
@ZooKeeperCallback
public void failover() {
+    if (pilotMode) {
+        RunId id = Pilot.start(
+            this::failover$pilot, policy);
+        if (Pilot.waitFinishOrAbort(id)
+                == FAILED) {
+            Pilot.report(id);
+            return;
+        }
+    }
   //Original recovery code
}
+public void failover$pilot(){…}
```

Initiate phantom threads

- Instrument the recovery entry point to launch pilot execution



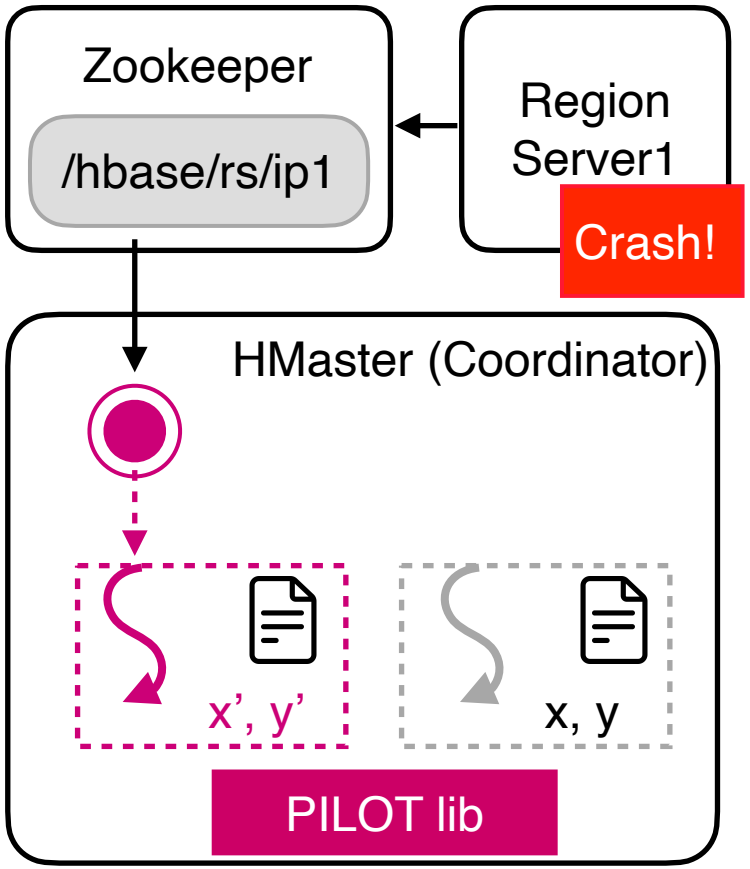
Entry point of HBase recovery

Divert into pilot execution

```
@ZooKeeperCallback
public void failover() {
+    if (pilotMode) {
+        RunId id = Pilot.start(
+            this::failover$pilot, policy);
+        if (Pilot.waitFinishOrAbort(id)
+                == FAILED) {
+            Pilot.report(id);
+            return;
+        }
+    }
   //Original recovery code
}
+public void failover$pilot(){…}
```

Isolate phantom threads

- Isolate side effects with pilot functions



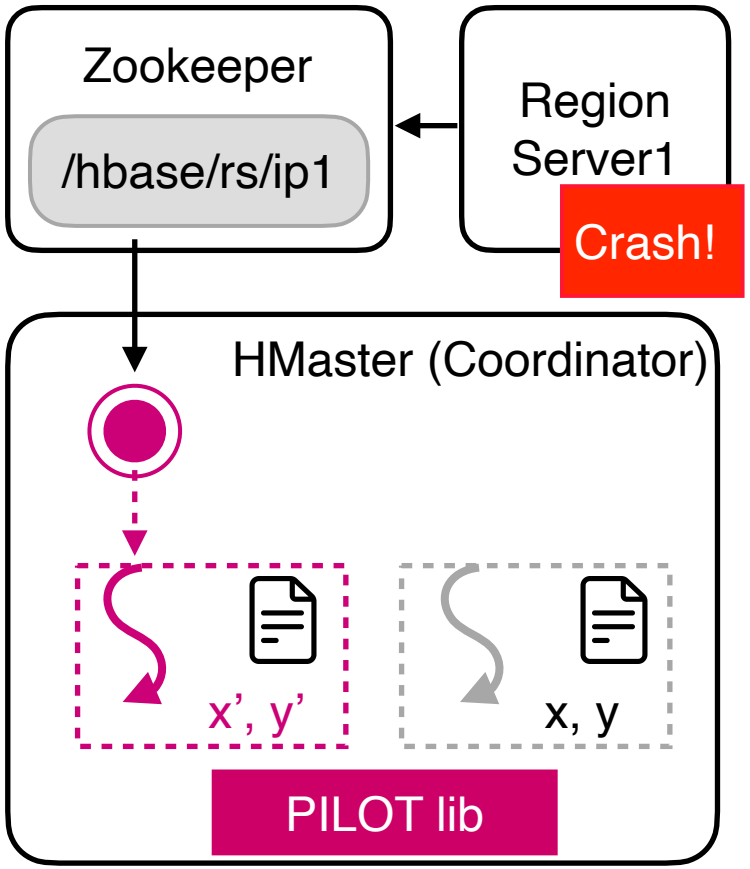
(1) State Isolation with Recursive Redirection

```
public class NodeFailoverWorker {
    public Server server;

    public void failover() {
        transferQueue(server);
        server = getFailoverTarget();
    }
}
```

Isolate phantom threads

- Isolate side effects with pilot functions



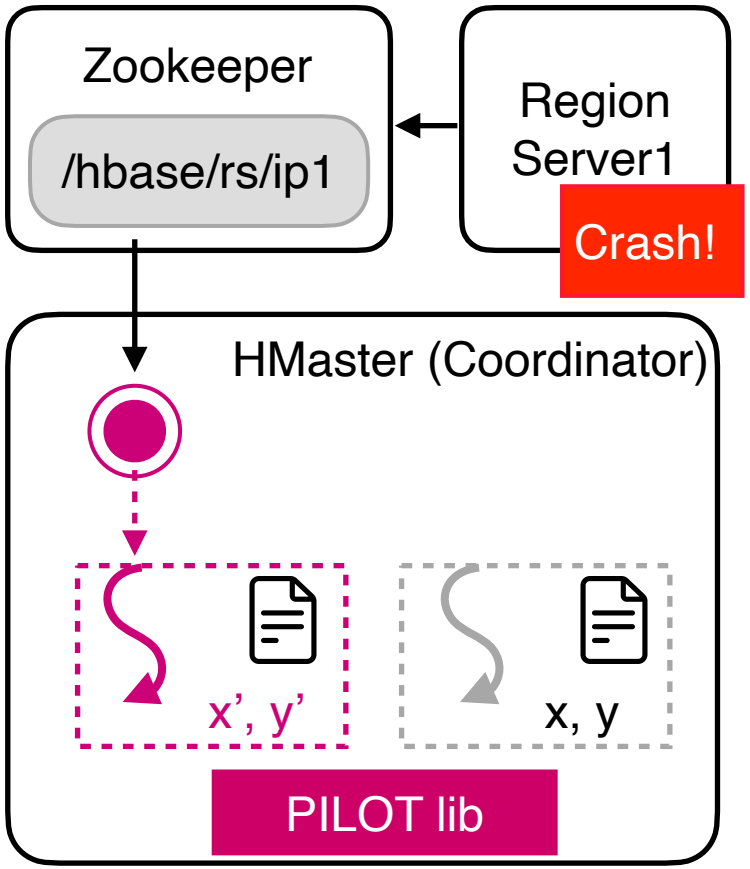
(1) State Isolation with Recursive Redirection

```
public class NodeFailoverWorker {
    public Server server;
+   public Server server$pilot;

    public void failover() {
        transferQueue(server);
        server = getFailoverTarget();
    }
}
```

Isolate phantom threads

- Isolate side effects with pilot functions



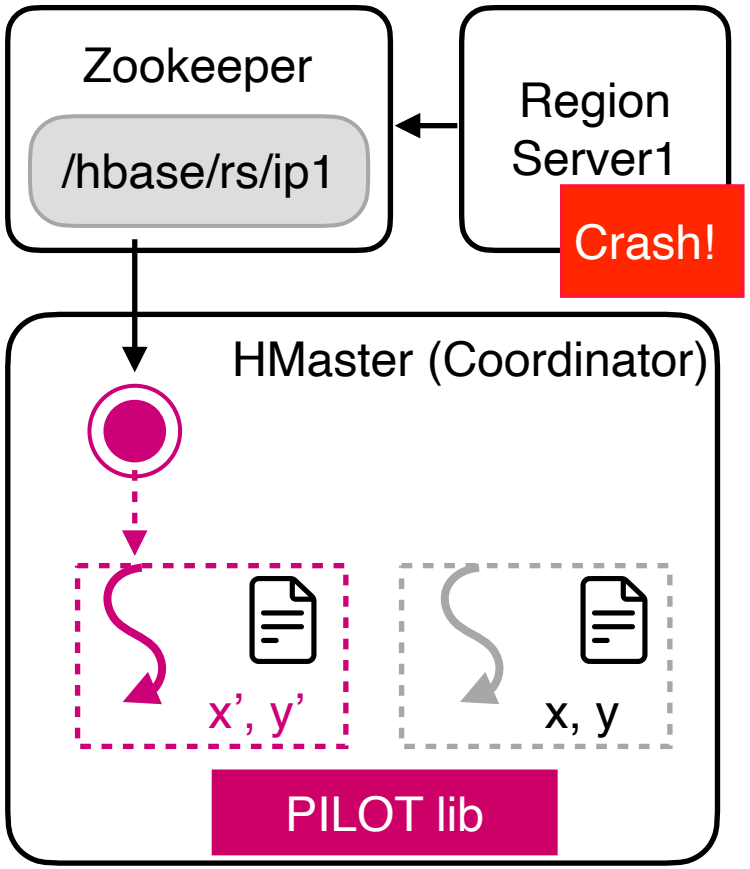
(1) State Isolation with Recursive Redirection

```
public class NodeFailoverWorker {
    public Server server;
+   public Server server$pilot;

+   public void failover$pilot() {
+       server$pilot =
+           PilotUtil.copy(server);
        transferQueue(server);
        server = getFailoverTarget();
    }
}
```

Isolate phantom threads

- Isolate side effects with pilot functions



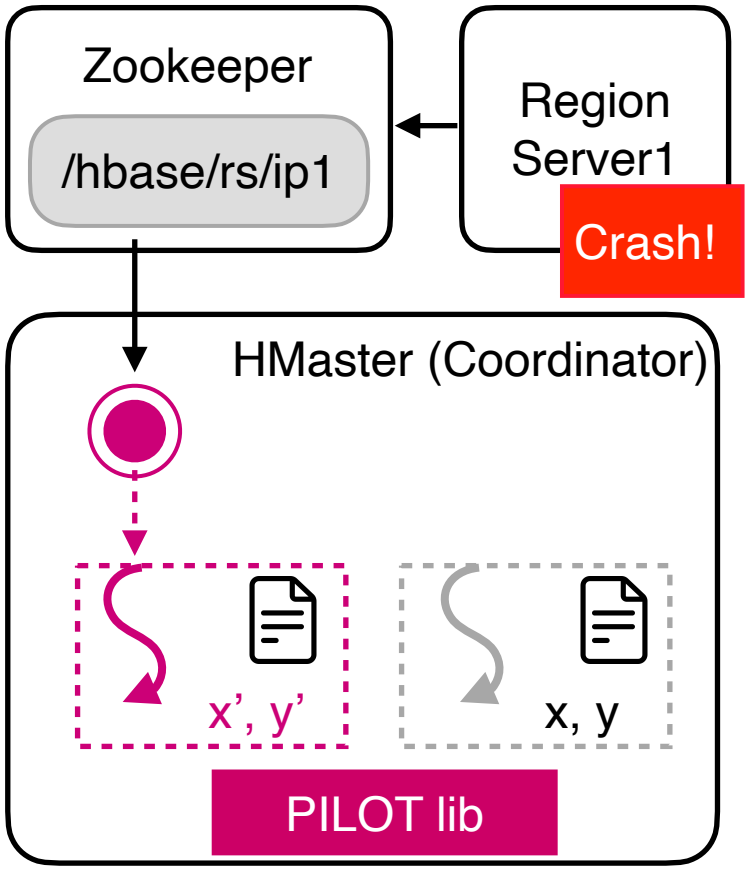
(1) State Isolation with Recursive Redirection

```
public class NodeFailoverWorker {
    public Server server;
+   public Server server$pilot;

+   public void failover$pilot() {
+       server$pilot =
+           PilotUtil.copy(server);
-       transferQueue(server);
-       server = getFailoverTarget();
+       transferQueue$pilot(server);
+       server$pilot =
+           getFailoverTarget$pilot();
    }
}
```

Isolate phantom threads

- Isolate side effects with pilot functions



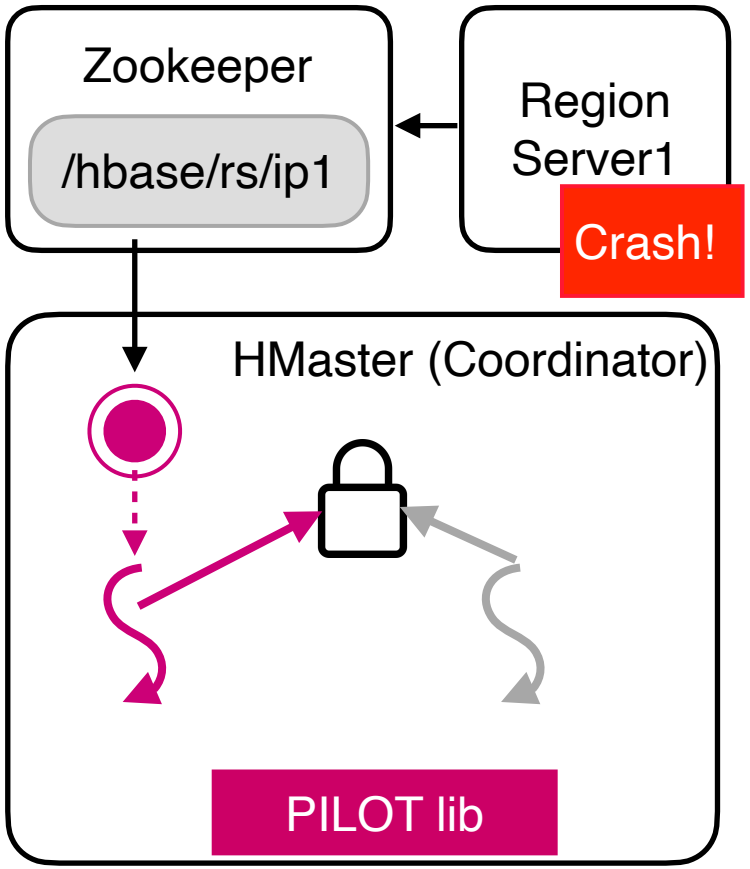
(1) State Isolation (2) I/O (3) Third-Party Service

```
public class NodeFailoverWorker {
    public Server server;
+   public Server server$pilot;

+   public void failover$pilot() {
+       server$pilot =
+           PilotUtil.copy(server);
-       transferQueue(server);
-       server = getFailoverTarget();
+       transferQueue$pilot(server);
+       server$pilot =
+           getFailoverTarget$pilot();
    }
}
```

Manage phantom threads

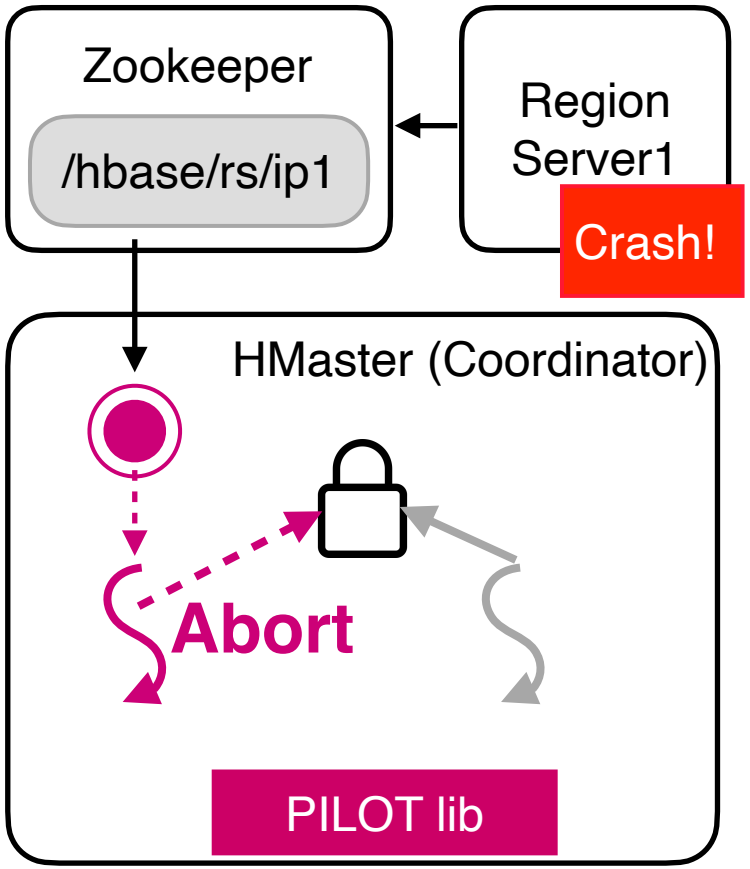
- Avoid disrupting production



- Phantom threads and original threads may compete

Manage phantom threads

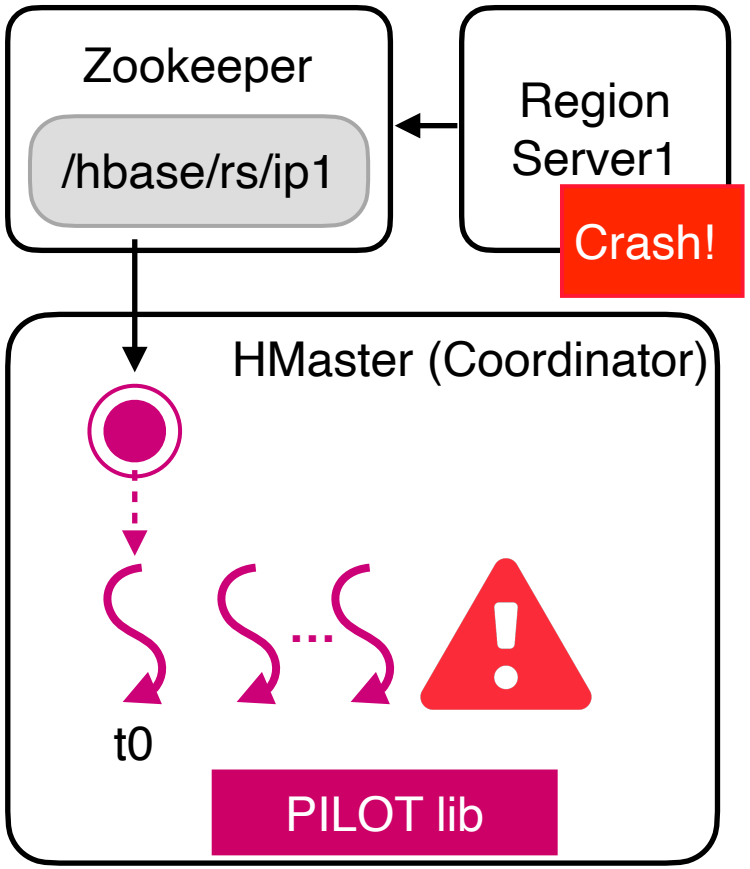
- Avoid disrupting production



- Phantom threads and original threads may compete
  - Phantom threads always yield to original threads

Manage phantom threads

- Avoid disrupting production



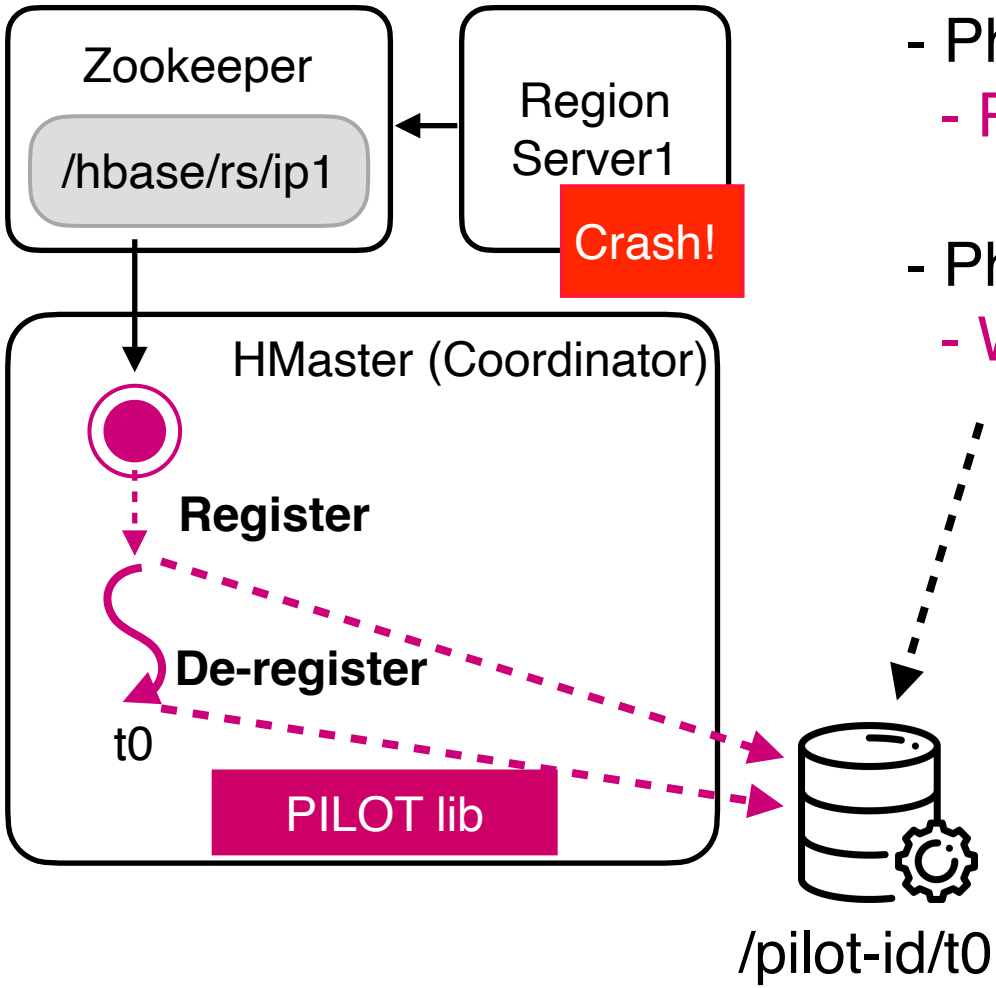
/pilot-id/t0

- Phantom threads and original threads may compete
  - Phantom threads always yield to original threads

- Phantom threads may leak or outlive the pilot run

Manage phantom threads

- Avoid disrupting production

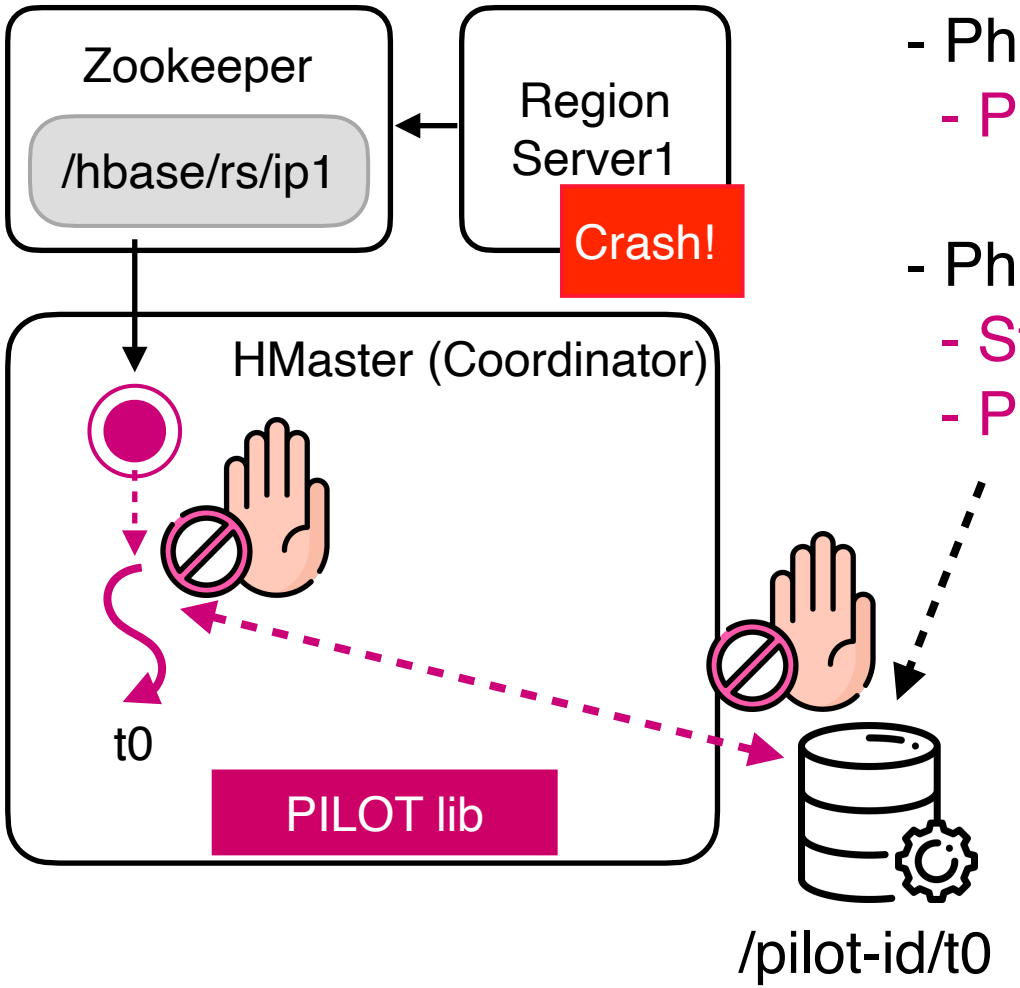


- Phantom threads and original threads may compete
  - Phantom threads always yield to original threads

- Phantom threads may leak or outlive the pilot run
  - Watch status registry to check if finished or not

Manage phantom threads

- Avoid disrupting production

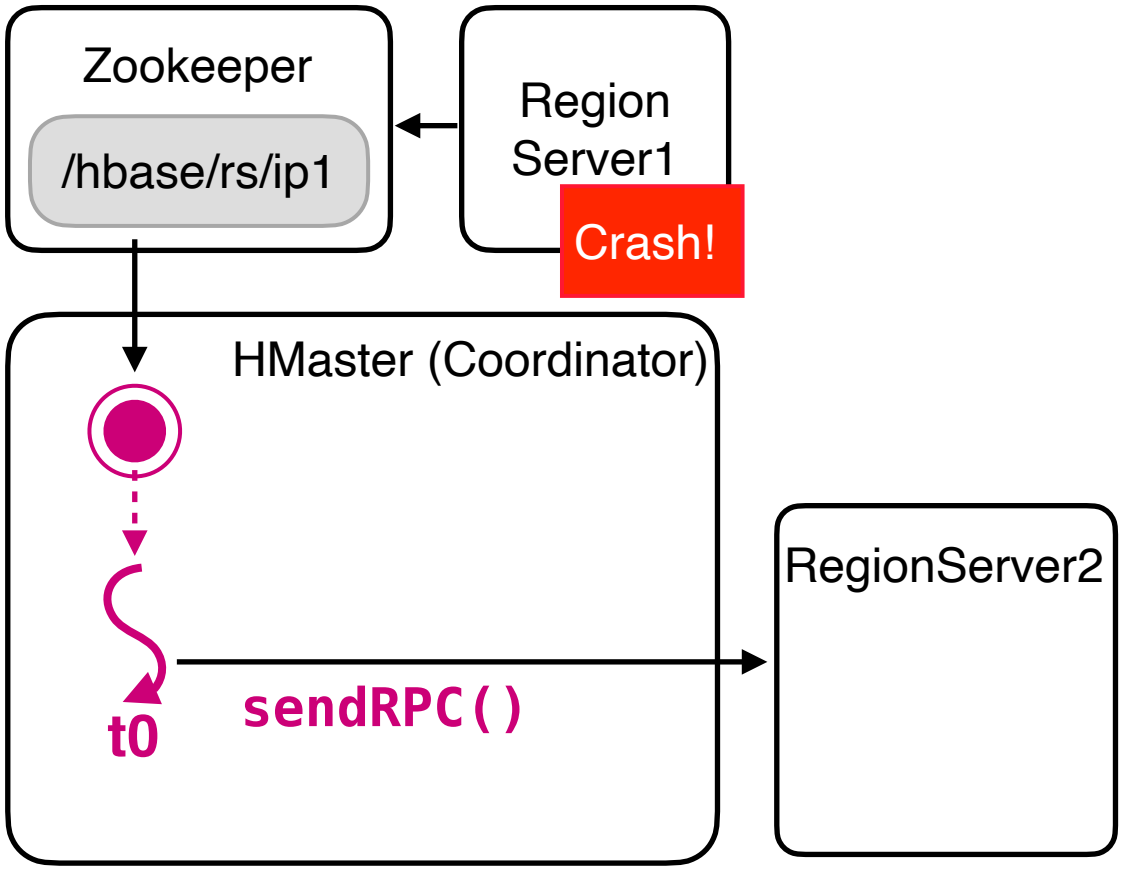


- Phantom threads and original threads may compete
  - Phantom threads always yield to original threads

- Phantom threads may leak or outlive the pilot run
  - Status registry can trigger cleanup callbacks
  - Prevent storms of phantom threads

Challenge 2 Simulate recovery in distributed environment

- **Recovery includes a lot of inter-thread and inter-process interactions**
  - e.g. RPC, async executor, condition variable, etc…

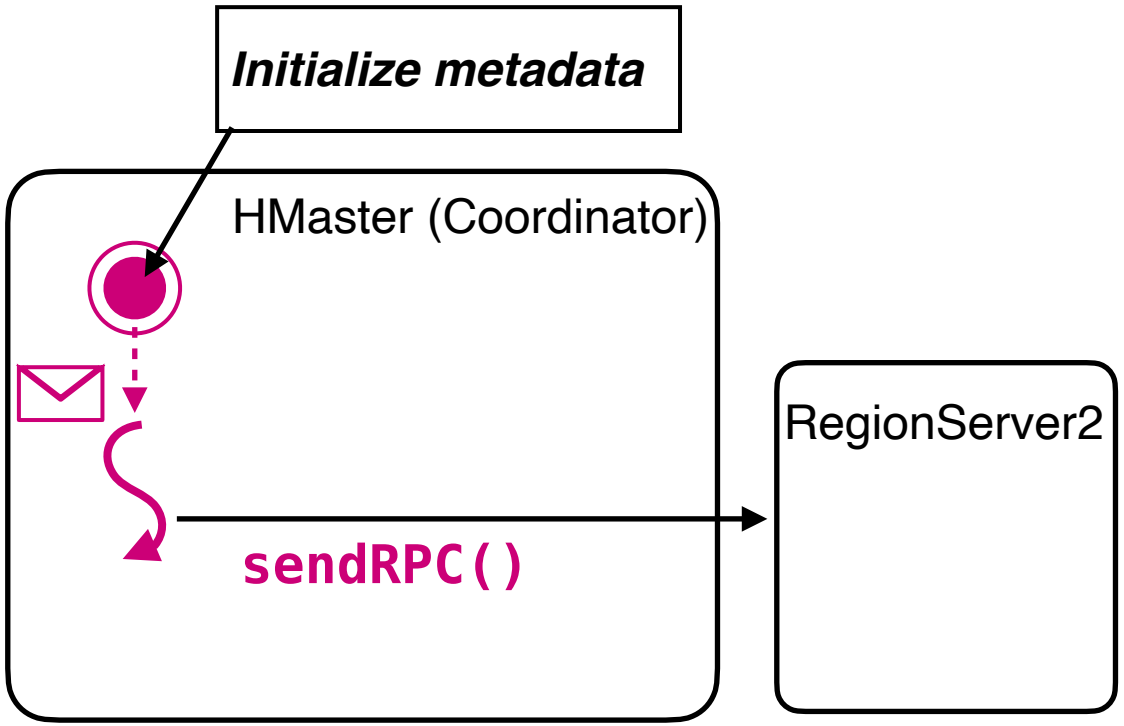


How to handle propagation?

```
public void failover$pilot() {
 Region region = getRegion();
 sendRPC(region);
}
```

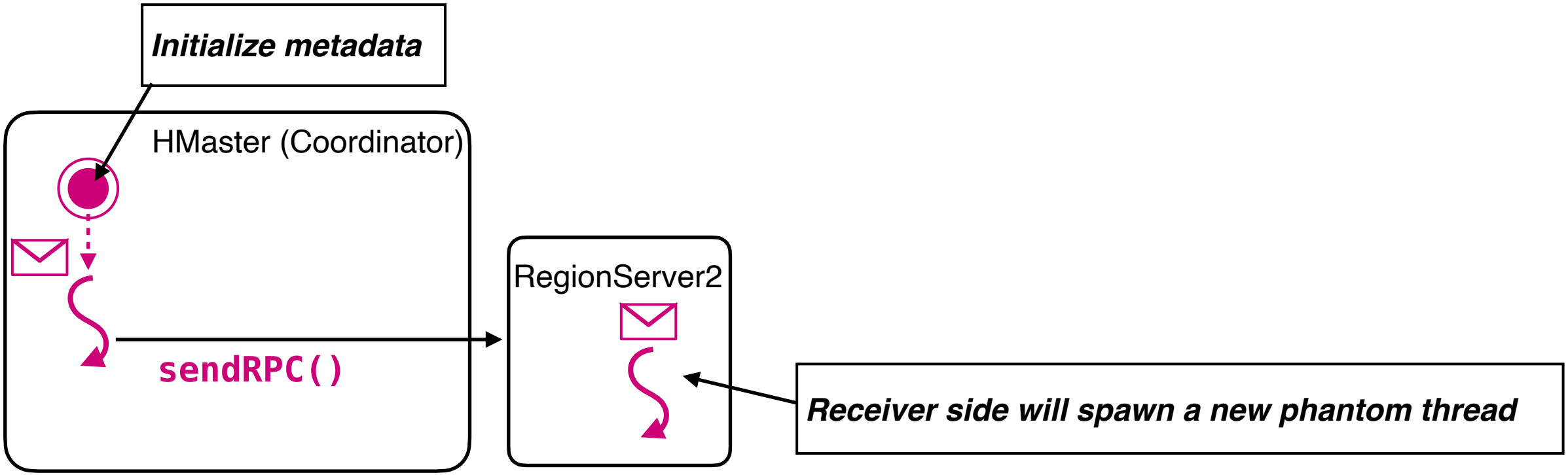
Challenge 2 Simulate recovery in distributed environment

- **Recovery includes a lot of inter-thread and inter-process interactions**
  - e.g. RPC, async executor, condition variable, etc…
- **How to ensure the whole recovery process is simulated by phantom threads?**
  - **Propagate pilot execution with tracked metadata**



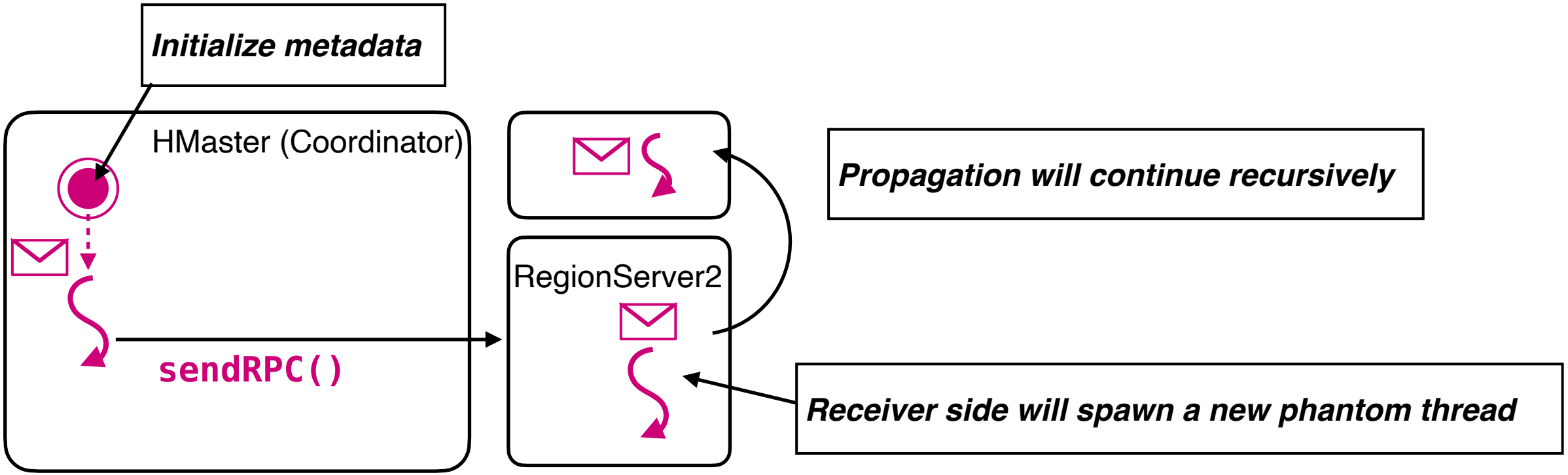
Challenge 2 Simulate recovery in distributed environment

- **Recovery includes a lot of inter-thread and inter-process interactions**
  - e.g. RPC, async executor, condition variable, etc…
- **How to ensure the whole recovery process is simulated by phantom threads?**
  - **Propagate pilot execution with tracked metadata**



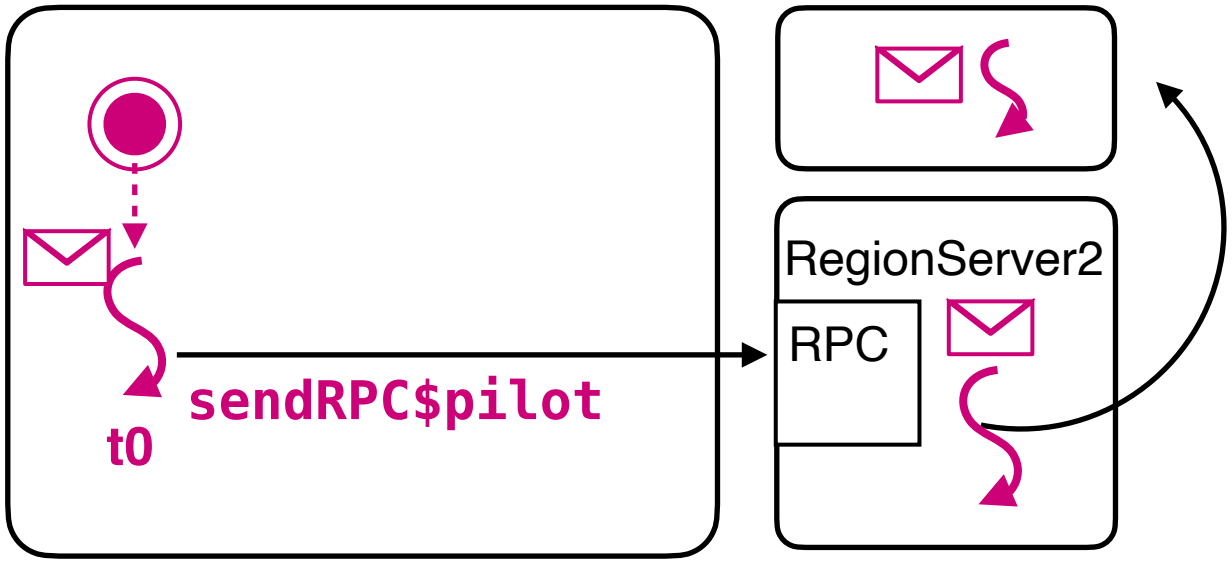
Challenge 2 Simulate recovery in distributed environment

- **Recovery includes a lot of inter-thread and inter-process interactions**
  - e.g. RPC, async executor, condition variable, etc…
- **How to ensure the whole recovery process is simulated by phantom threads?**
  - **Propagate pilot execution with tracked metadata**



Propagate execution with the tracked metadata

- **PilotContext: Pilot Run ID (Global Identifier) + Span metadata**
  - Tracing infrastructure already supports common propagation scenarios

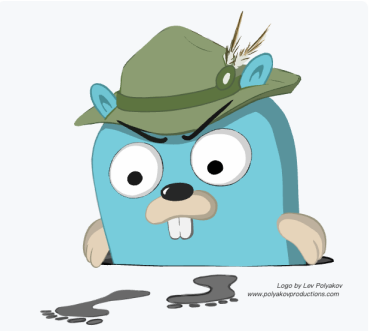


OpenTelemetry

slack join chat → | Unit Tests passing | codecov 97% | Project+Community stats
license scan passing | openssf scorecard 8.2 | openssf best practices silver
CLOMonitor Report A | Artifact Hub jaegertracing

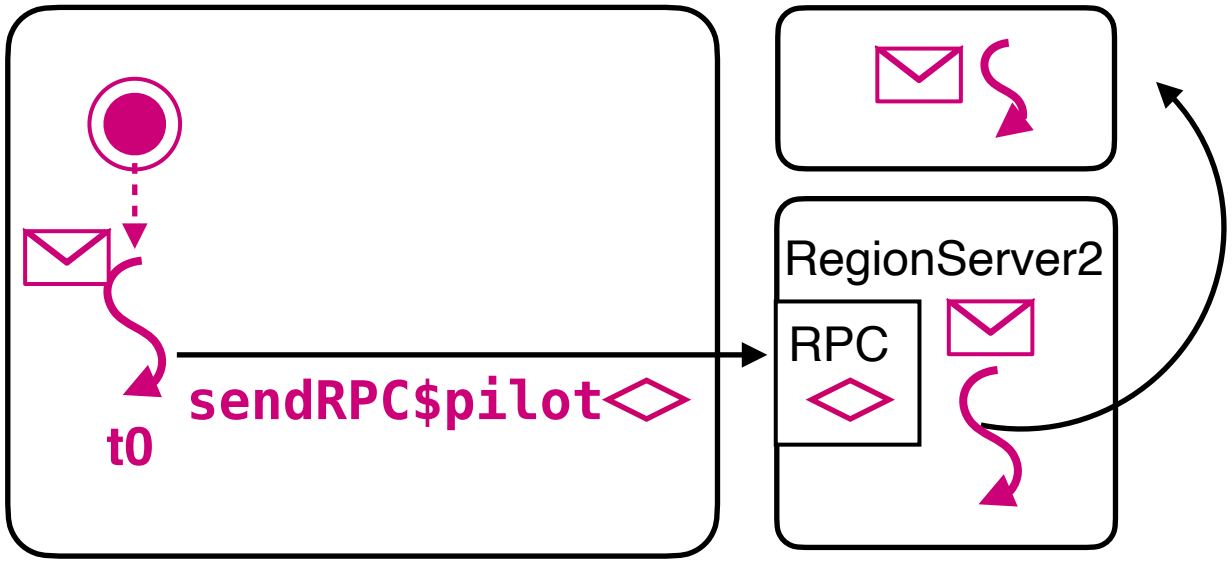
CLOUD NATIVE COMPUTING FOUNDATION
GRADUATED PROJECT

Jaeger - a Distributed Tracing System



Propagate execution with the tracked metadata

- **PilotContext: Pilot Run ID (Global Identifier) + Span metadata**
  - Tracing infrastructure already supports common propagation scenarios
  - Our instrumentation piggybacks on existing tracing frameworks with hooks ◇

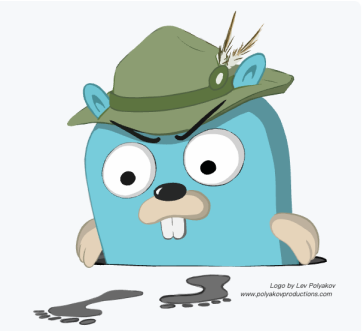


OpenTelemetry

slack join chat → | Unit Tests passing | codecov 97% | Project+Community stats
license scan passing | openssf scorecard 8.2 | openssf best practices silver
CLOMonitor Report A | Artifact Hub jaegertracing

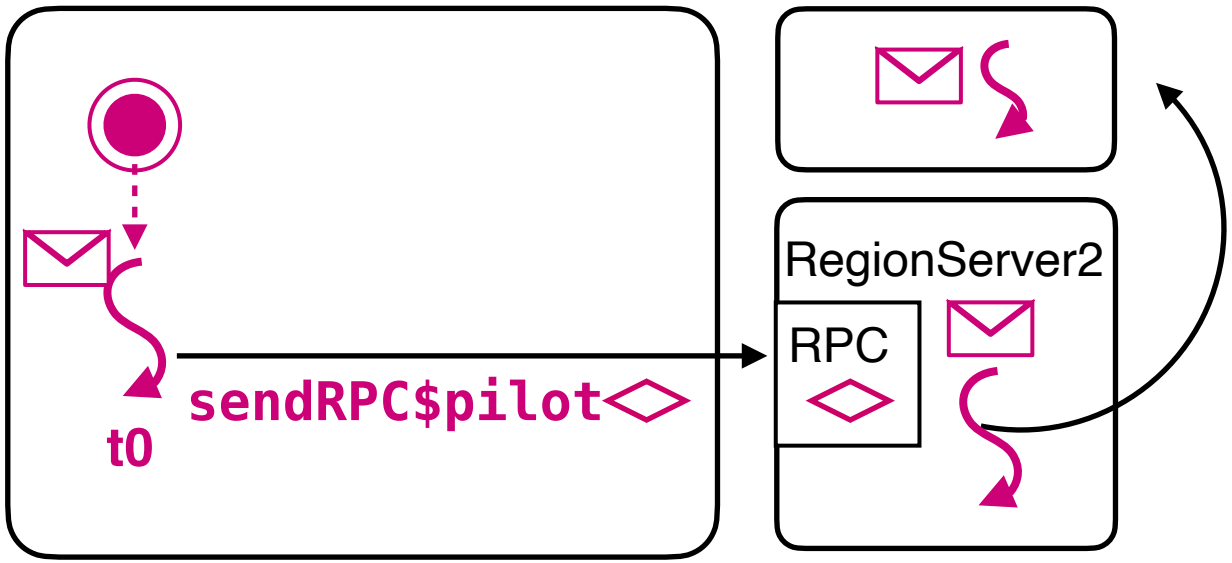
CLOUD NATIVE COMPUTING FOUNDATION
GRADUATED PROJECT

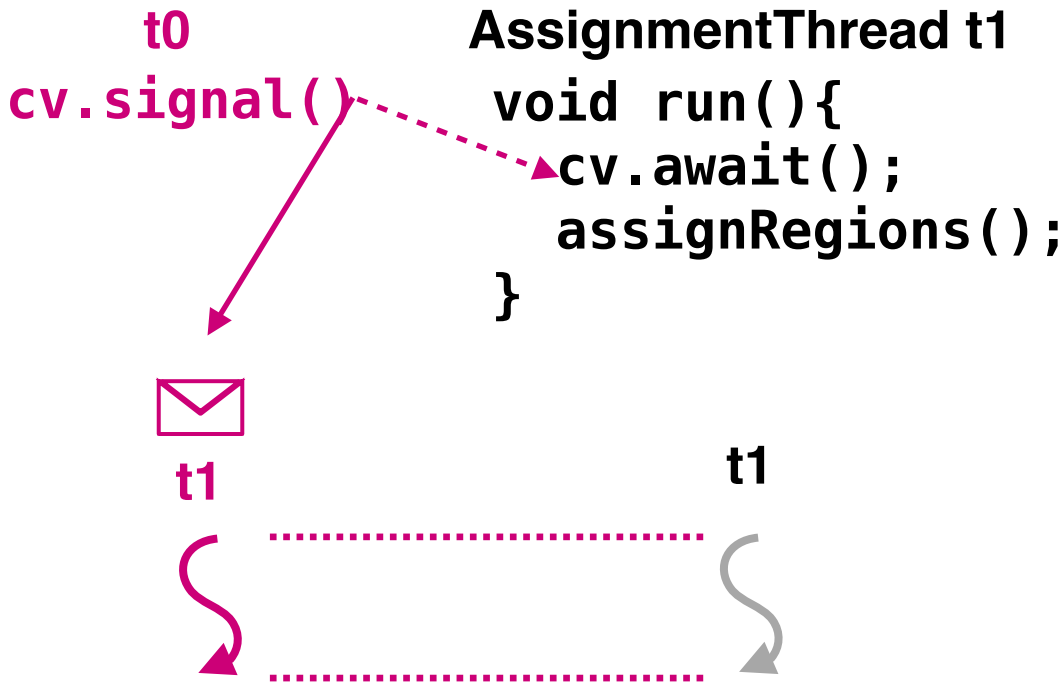
Jaeger - a Distributed Tracing System



Propagate execution with the tracked metadata

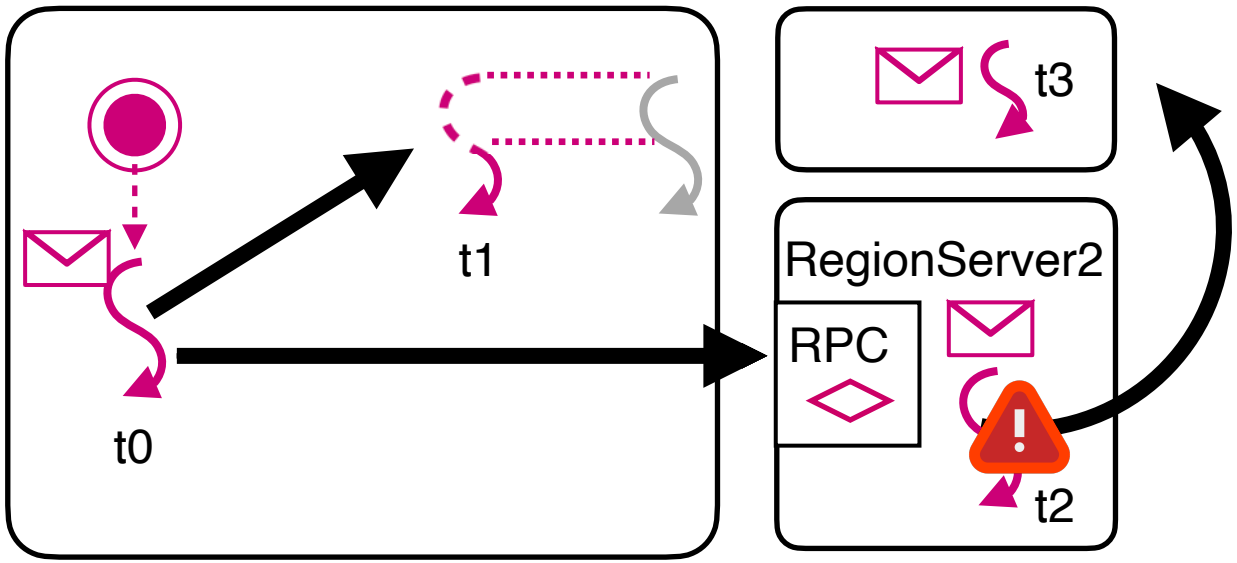
- **PilotContext: Pilot Run ID (Global Identifier) + Span metadata**
  - For some cases (e.g., cond vars), we simulate the affected behavior explicitly





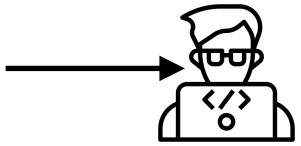
Propagate execution with the tracked metadata

- **PilotContext: Pilot Run ID (Global Identifier) + Span metadata**
  - Span relationship and alarm will be provided to the operator



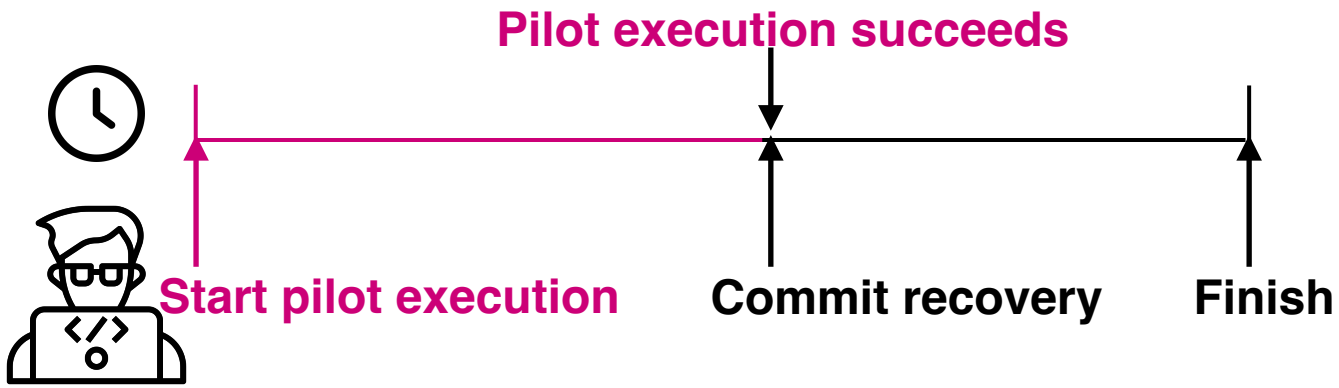

```
catch(IOException e){
-Runtime.halt();
+Pilot.report();
}
```

Alarm: t2 crashes with WAL-xxx



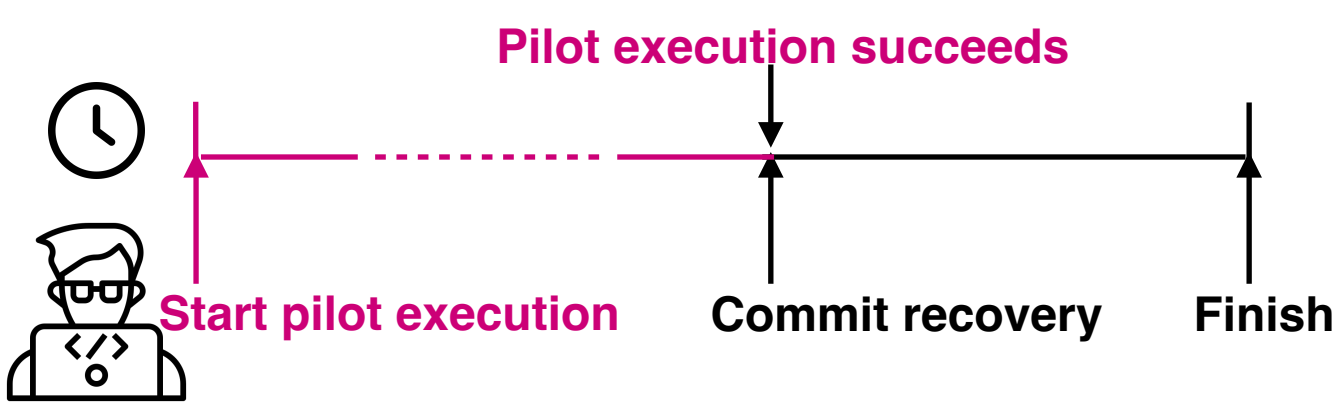
Challenge 3 Avoid delaying original recovery

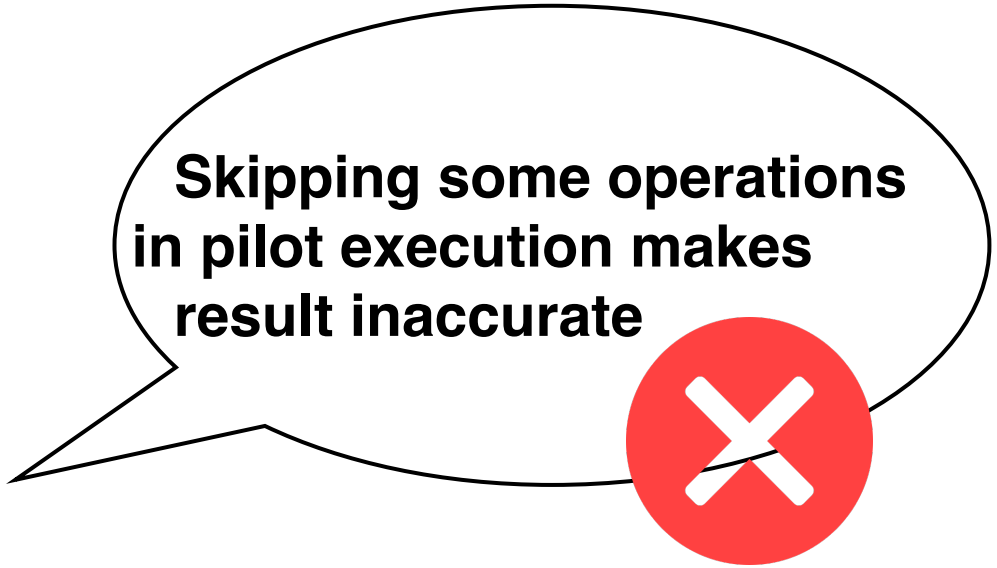
-Without optimization, end-to-end recovery time will be doubled



Challenge 3 Avoid delaying original recovery

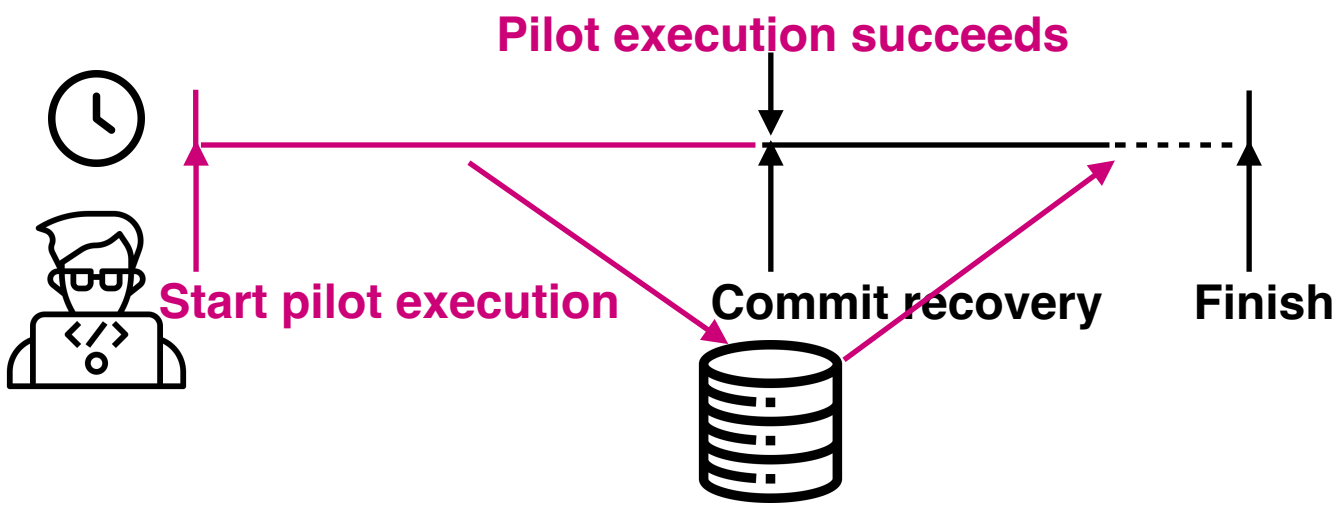
-Without optimization, end-to-end recovery time will be doubled





Challenge 3 Avoid delaying original recovery

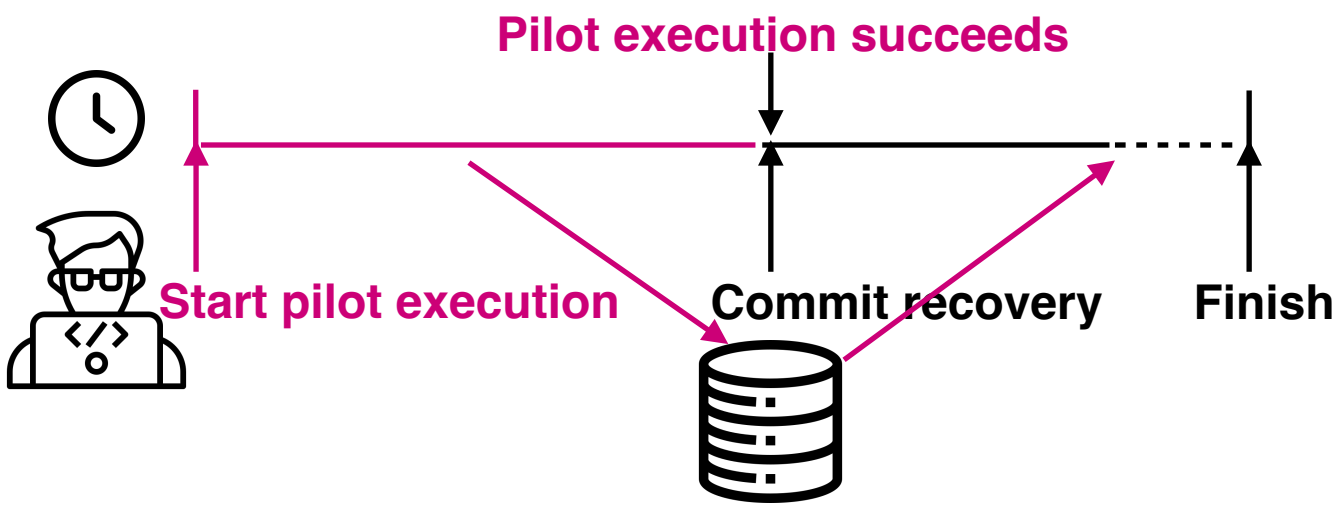
- Without optimization, end-to-end recovery time will be doubled



Pilot lib caches I/O and compute-intensive results

Challenge 3 Avoid delaying original recovery

-Without optimization, end-to-end recovery time will be doubled

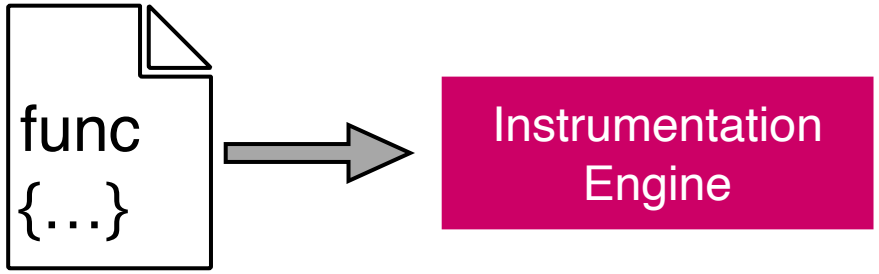


Pilot lib caches I/O and compute-intensive results

-Solution: accelerate original recovery with caching

PILOT: System Support for Pilot Execution

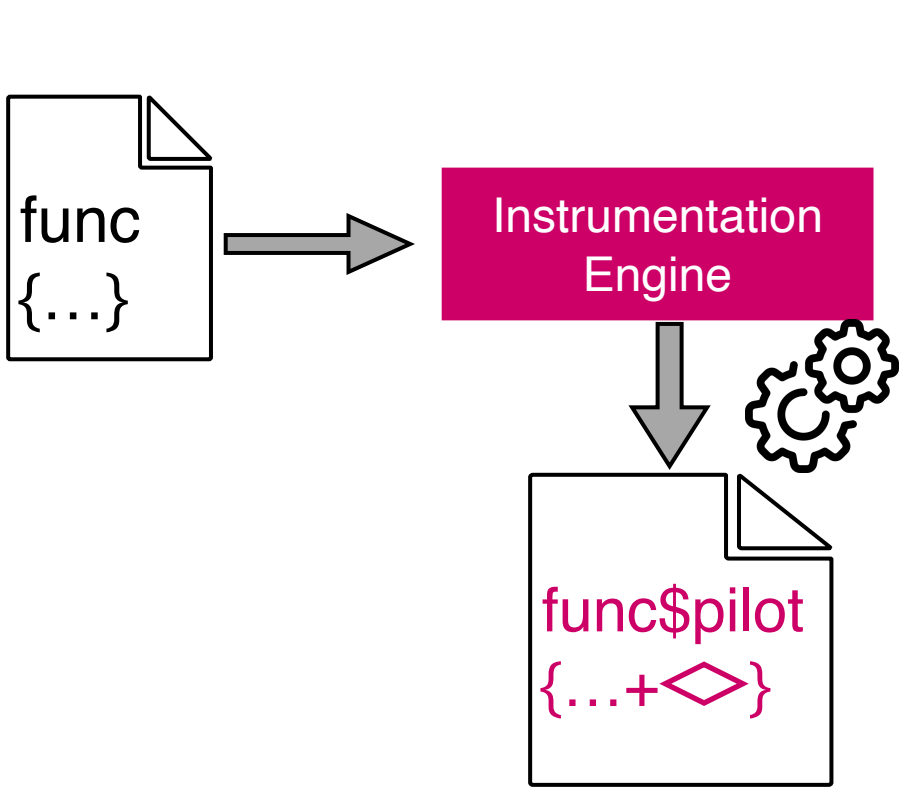
- We implement PILOT, a framework that makes pilot execution easy to adopt for existing distributed systems
- PILOT contains an **instrumentation engine** and a **runtime library**



Offline

PILOT: System Support for Pilot Execution

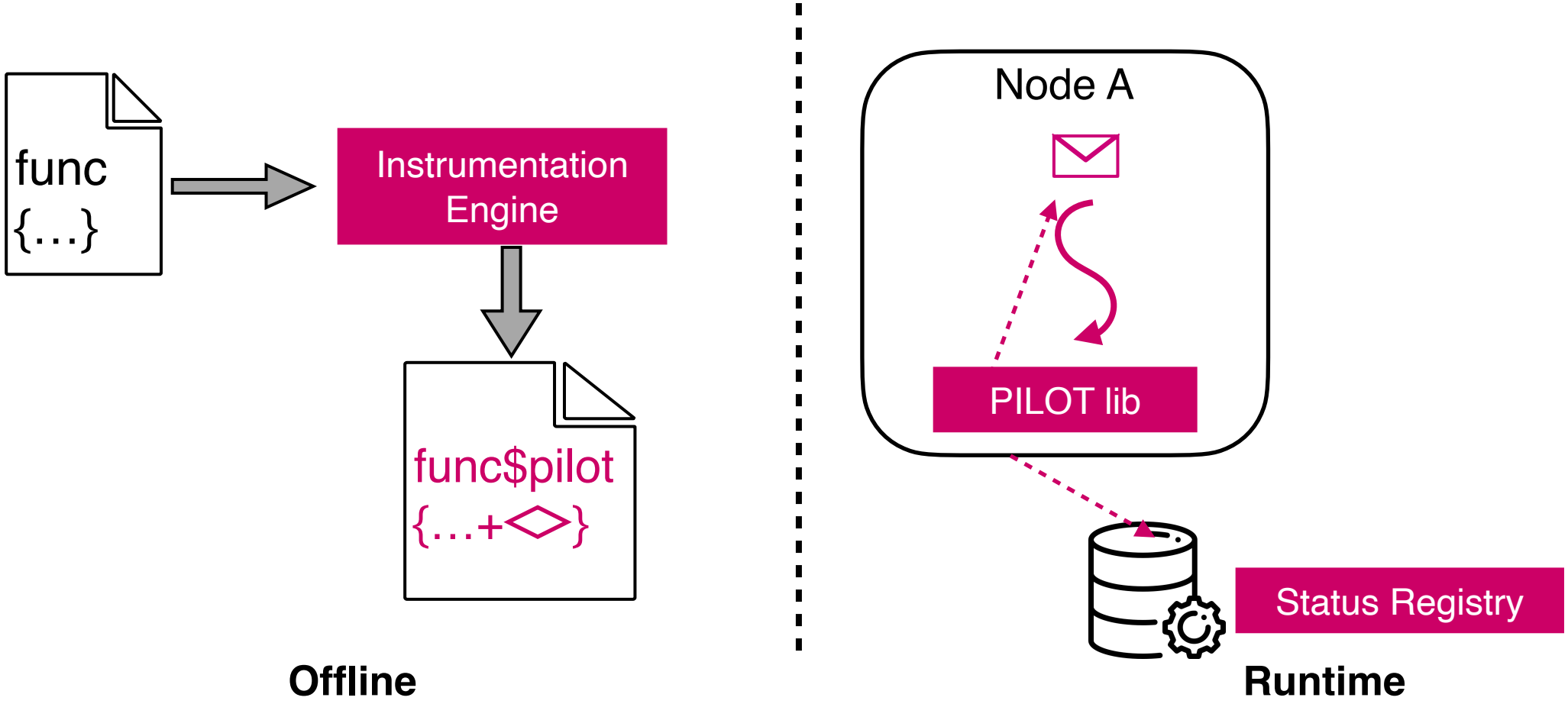
- PILOT contains an **instrumentation engine** and a **runtime library**
- Instrumentation generate pilot version functions and add hooks



Offline

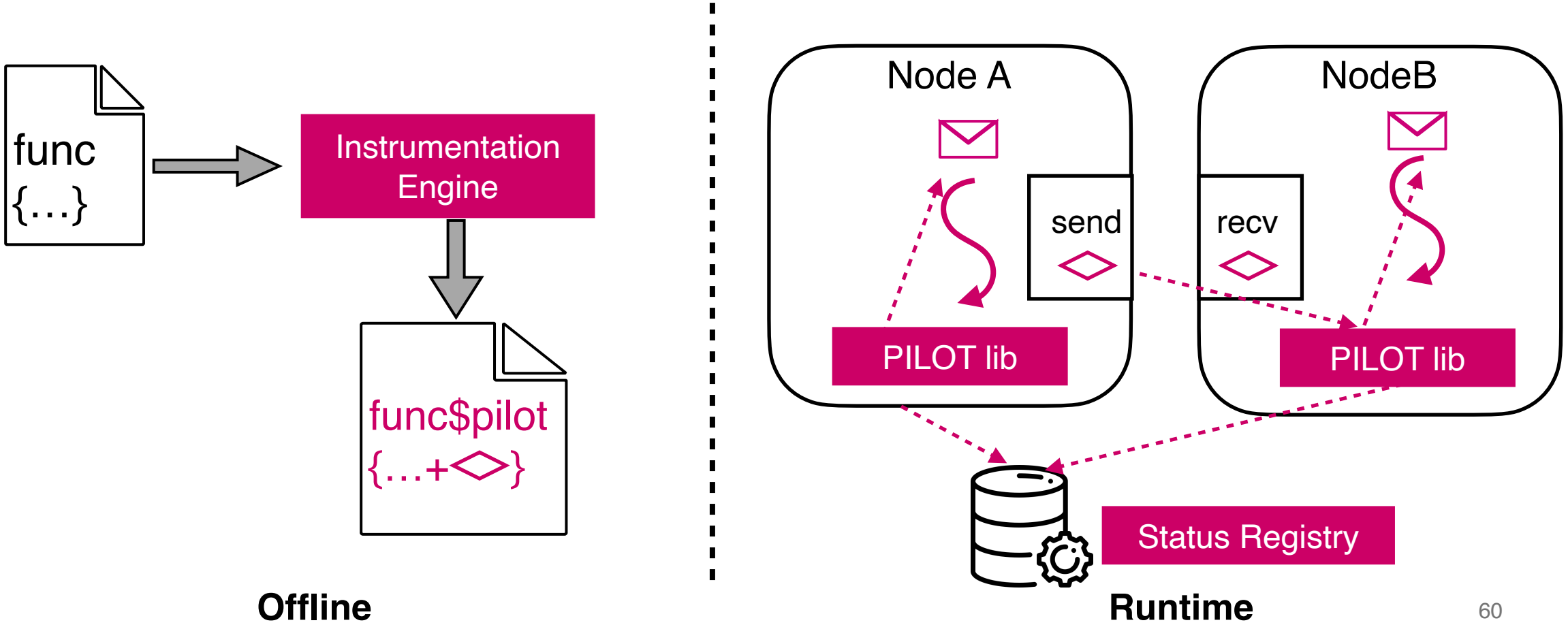
PILOT: System Support for Pilot Execution

- PILOT contains an **instrumentation engine** and a **runtime library**
- Instrumentation generate pilot version functions and add hooks
- PILOT runtime library **propagates** and **manages** pilot execution



PILOT: System Support for Pilot Execution

- PILOT contains an **instrumentation engine** and a **runtime library**
- Instrumentation generate pilot version functions and add hooks
- PILOT runtime library **propagates** and **manages** pilot execution



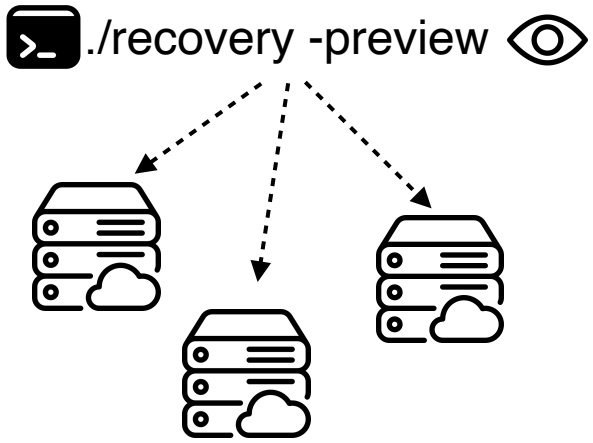
Evaluation: detection

JIRA Id.	Id.	Sys.Feature	Symptom
Solr-17515	SL1	Follower Recovery	State Inconsistency
Solr-10914	SL2	Follower Recovery	Partial Failure
Solr-6056	SL3	Core Recovery	Zombie(whole-cluster)
HDFS-16689	HF1	Namenode Failover	Service Unavailability
HDFS-9908	HF2	Error Handler	Cluster Initialization Failure
HDFS-14459	HF3	Error Handler	Incorrect Results
HDFS-10320	HF4	Datanode Recovery	Service Unavailability
HDFS-4937	HF5	Datanode Recovery	Partial Failure
Cassandra-13938	CA1	Node Recovery	Crash(Cascading)
Cassandra-14096	CA2	Node Recovery	Out-of-Memory
Cassandra-7560	CA3	Inconsistency	Partial Failure
Cassandra-6415	CA4	Inconsistency	Partial Failure
HBase-19980	HB1	Snapshot Recovery	State Inconsistency
HBase-13567	HB2	RegionServer Failover	State Inconsistency
HBase-25898	HB3	RegionServer Failover	Crash(Cascading)
YARN-7562	YN1	ResourceManager Failover	Service Unavailability
YARN-4347	YN2	ResourceManager Failover	Service Unavailability
YARN-6403	YN3	NodeManager Recovery	Crash
YARN-2816	YN4	NodeManager Recovery	Crash

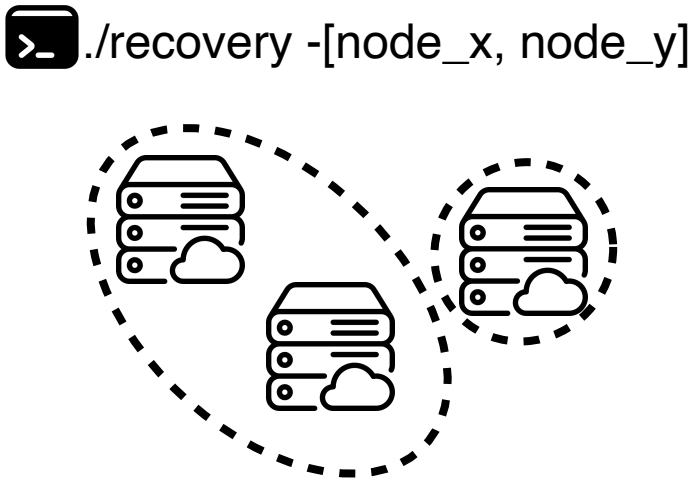
20 real-world recovery failures reproduced for evaluation

Evaluation: detection

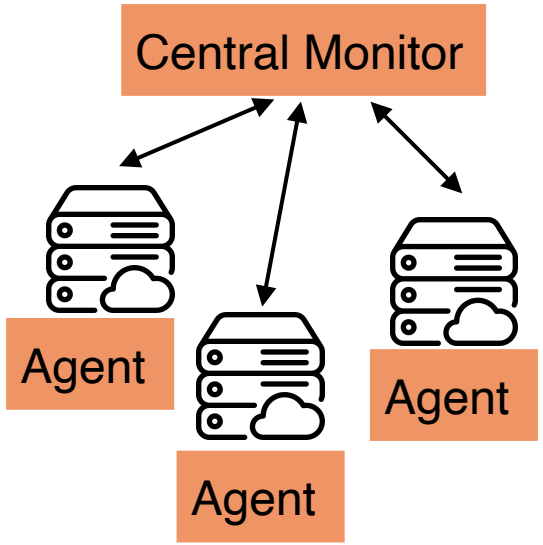
- We compare our approach with four baselines



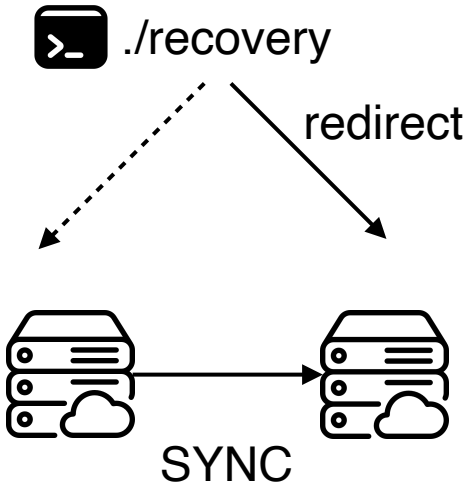
Built-in preview



A/B testing

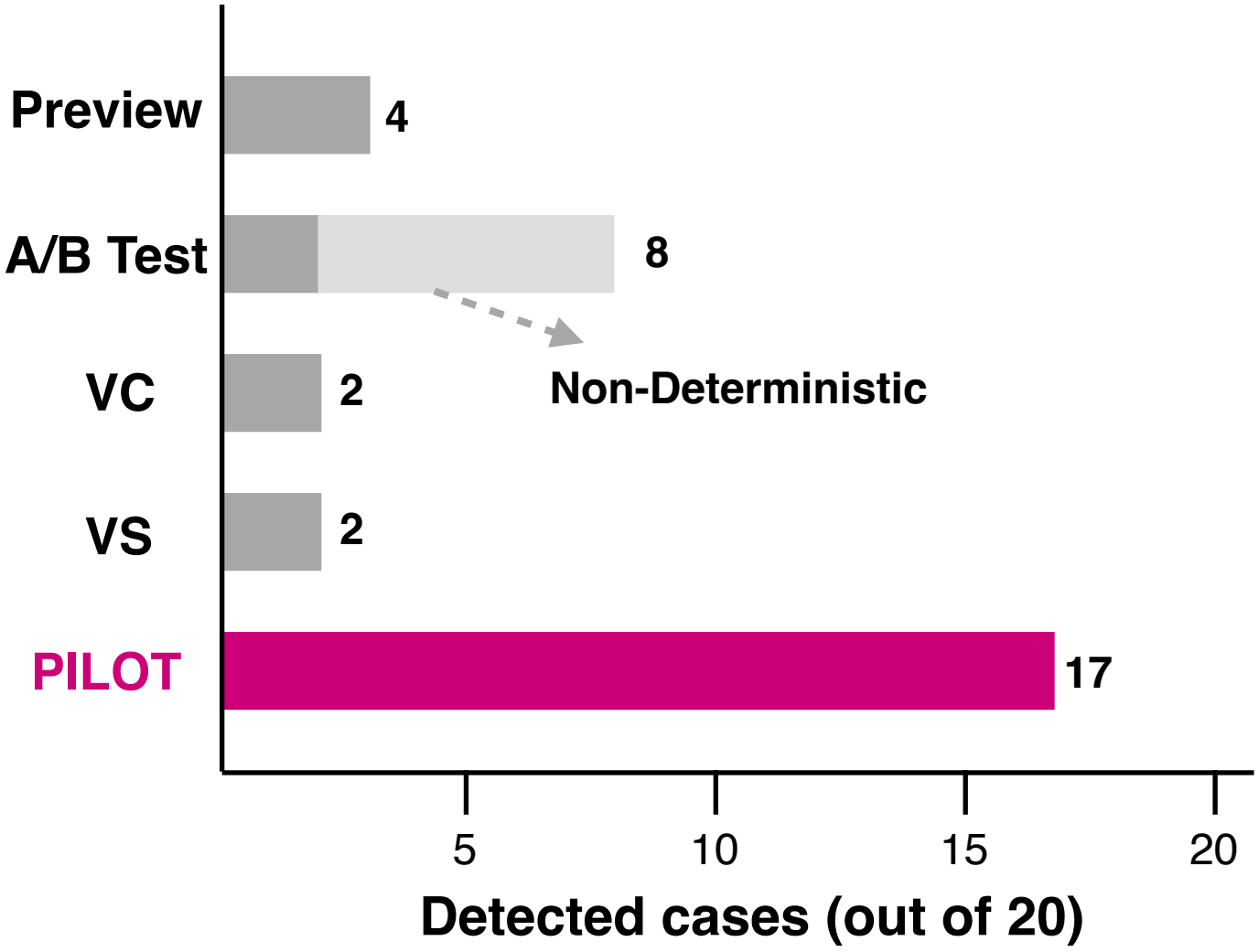


Correlate runtime logs
Vicious Cycle [ASE'23]

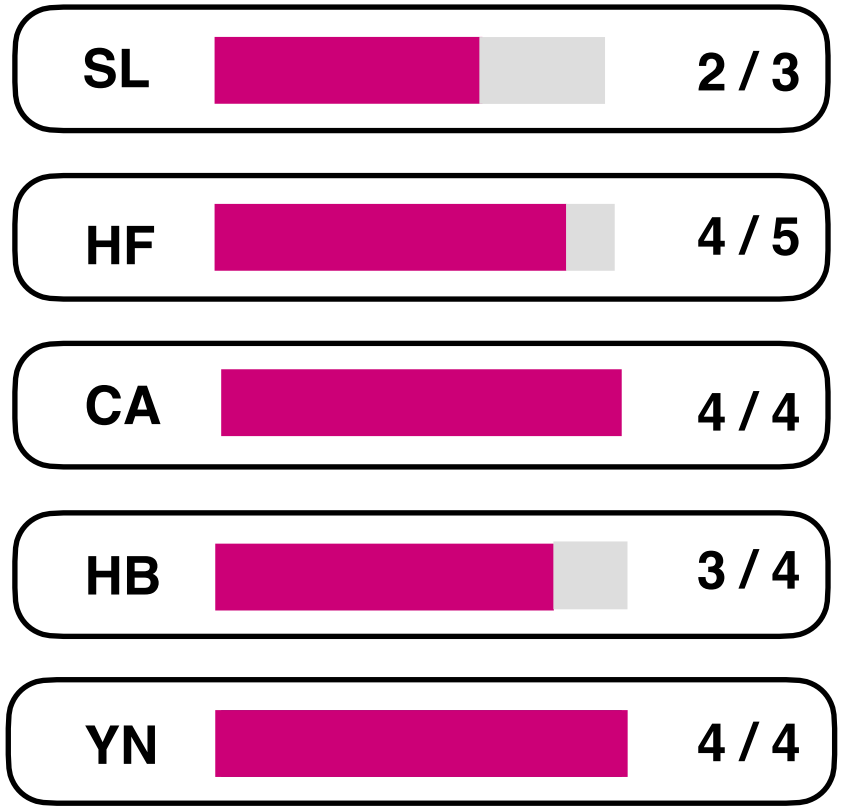


Replay traffic to masked components
Validation Slice [OSDI'04]

Evaluation: detection



17/20
cases detected by PILOT

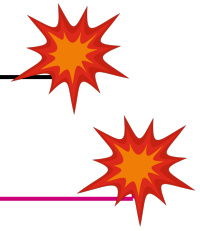


Evaluation: recovery time

Avg. Detection time of recovery failure

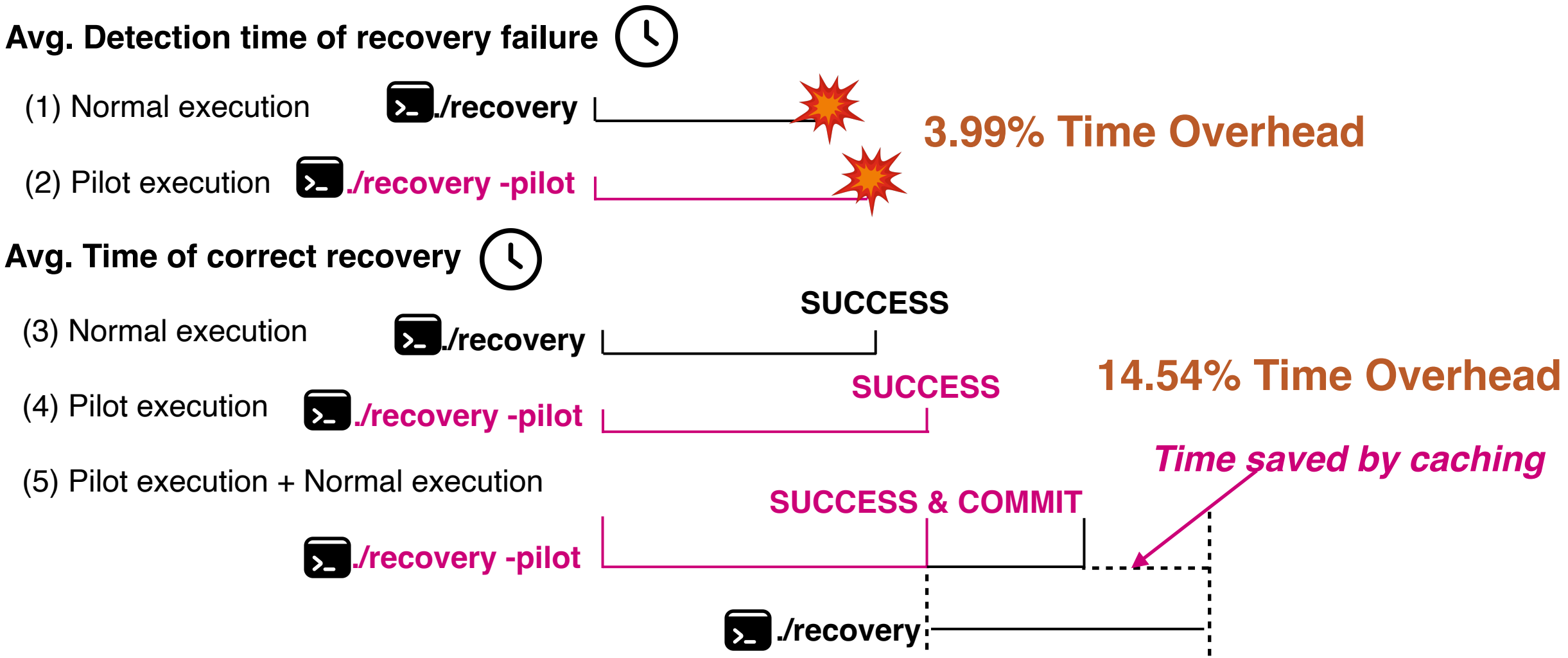
(1) Normal execution **./recovery**

(2) Pilot execution **./recovery -pilot**



3.99% Time Overhead

Evaluation: recovery time



PILOT Exposed a Critical HBase Recovery Bug

HBase / HBASE-28589

Server side DoNotRetryException not propagated to client

Edit | Add comment | Agile Board | More | Reopen Issue

Details

Type:	Bug	Status:	RESOLVED
Priority:	Critical	Resolution:	Fixed
Affects Version/s:	2.0.0, 2.4.0, 2.5.0, 2.6.0, 3.0.0	Fix Version/s:	2.7.0, 3.0.0-beta-2, 2.6.4, 2.5.13
Component/s:	IPC/RPC		
Labels:	pull-request-available		
Tags:	Retry, RPC, cascading failure, region server		
External issue URL:	https://github.com/apache/hbase/pull/7156		

Description

When an IOException occurs during response creation in ServerCall.setResponse(), the method only catches the IOException and logs a warning and sets the response to null. This causes the client to receive no response or experience connection issues without knowing what went wrong on the server side.

Conclusion

- Recovery in production distributed systems can be risky.
- Pilot execution turns recovery into a previewable "dry-run" action.
- In-situ dry-runs are feasible with phantom threads and context propagation.
- PILOT is a first step toward validatable cloud management.

Key results

- Studied **75** real-world recovery failures
- PILOT detects **17/20** real-world recovery failures
- PILOT adds modest **overhead** to original recovery

github.com/LiftLab-UVA/PilotExecution

vnr3ne@virginia.edu