Gray Failure: The Achilles' Heel of Cloud-Scale Systems

Ryan Huang[1], Ze Li[2]

University of Michigan[1] *Microsoft Azure[2]*

MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

Microsoft

March 20th 2024

SREcon24 Americas

Outline

Principles + Practices

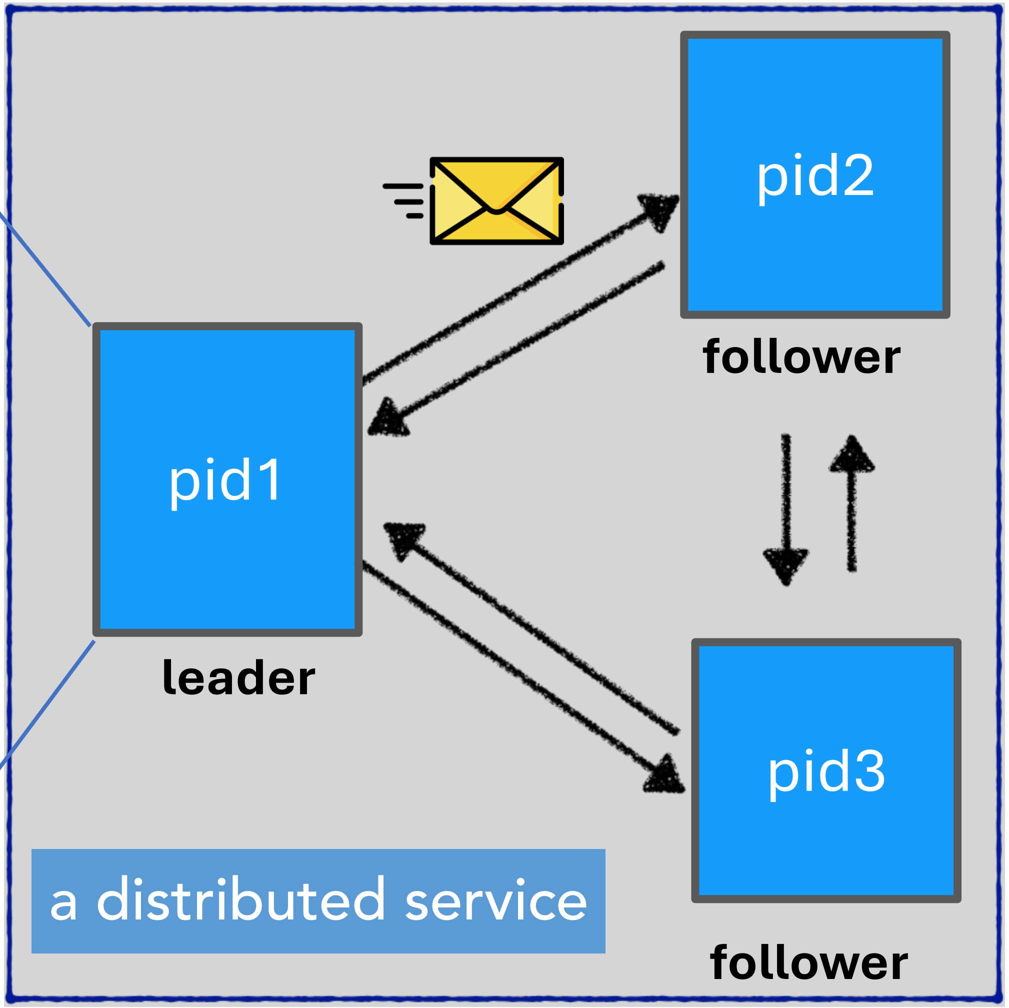
System + Data-Driven Approaches

Open Challenges

Large systems are built with clean abstractions

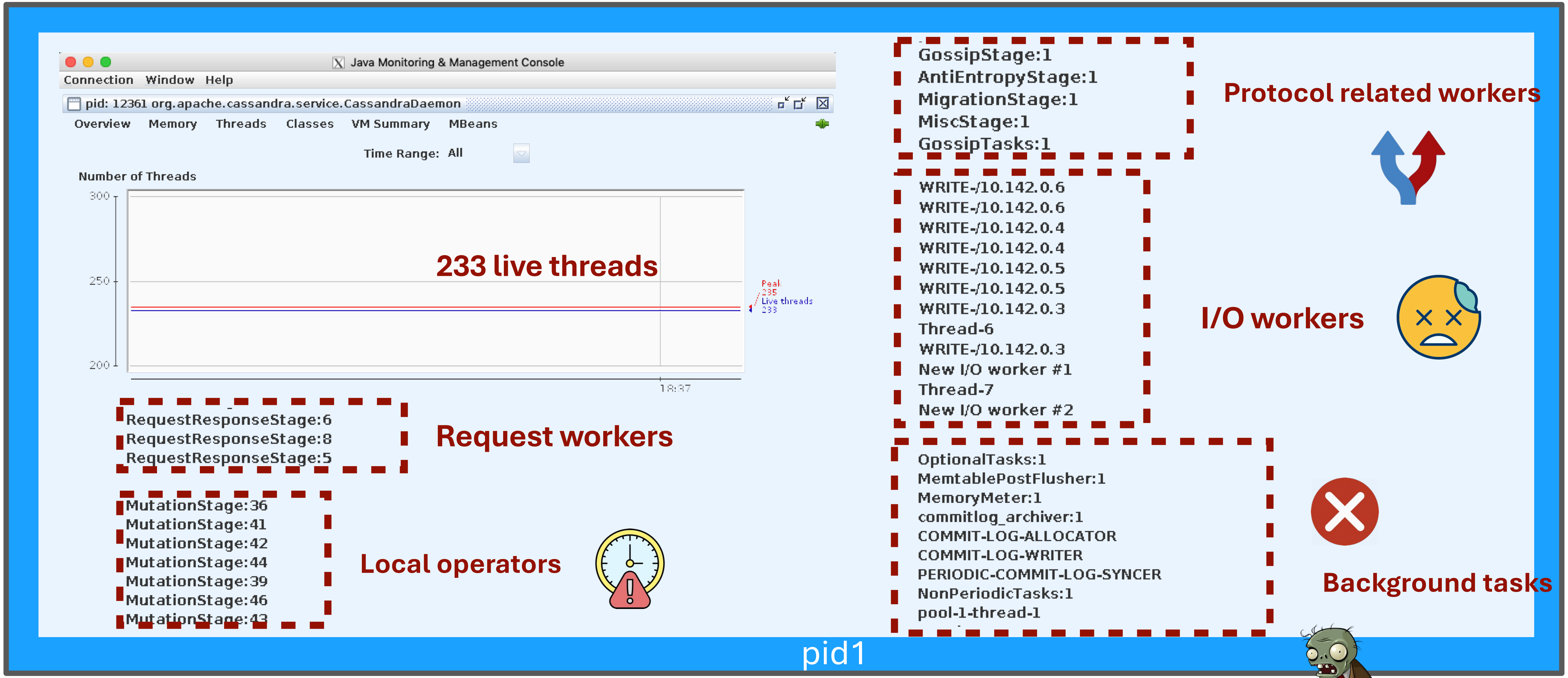
1. Abstract away the messy code into uniform "nodes"/processes

```
if (isDatanode) { //replication or move
  replicaHandler =
      datanode.data.createTemporary(storageType, storageId, block, isTransfer: false);
} else {
  switch (stage) {
  case PIPELINE_SETUP_CREATE:
    replicaHandler = datanode.data.createRbw(storageType, storageId,
        block, allowLazyPersist);
    datanode.notifyNamenodeReceivingBlock(
        block, replicaHandler.getReplica().getStorageUuid());
    break;
  case PIPELINE_SETUP_STREAMING_RECOVERY:
    replicaHandler = datanode.data.recoverRbw(
        block, newGs, minBytesRcvd, maxBytesRcvd);
    block.setGenerationStamp(newGs);
    break;
  case PIPELINE_SETUP_APPEND:
    replicaHandler = datanode.data.append(block, newGs, minBytesRcvd);
    block.setGenerationStamp(newGs);
    datanode.notifyNamenodeReceivingBlock(
        block, replicaHandler.getReplica().getStorageUuid());
    break;
  case PIPELINE_SETUP_APPEND_RECOVERY:
    replicaHandler = datanode.data.recoverAppend(block, newGs, minBytesRcvd);
    block.setGenerationStamp(newGs);
    datanode.notifyNamenodeReceivingBlock(
        block, replicaHandler.getReplica().getStorageUuid());
    break;
  case TRANSFER_RBW:
  case TRANSFER_FINALIZED:
    // this is a transfer destination
    replicaHandler = datanode.data.createTemporary(storageType, storageId,
        block, isTransfer);
    break;
  default: throw new IOException("Unsupported stage " + stage +
        " while receiving block " + block + " from " + inAddr);
  }
}
replicaInfo = replicaHandler.getReplica();
this.dropCacheBehindWrites = (cachingStrategy.getDropBehind() == null) ?
```



2. Model assorted interactions as clean messages

But software in practice is not “clean”

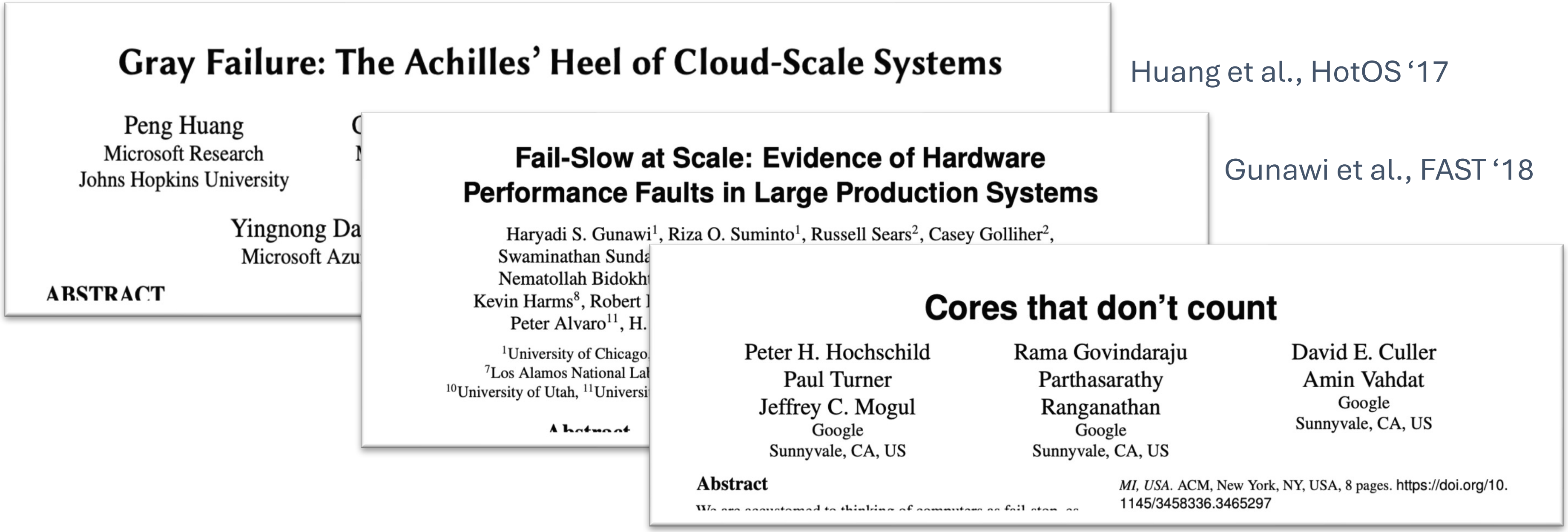


What appears “alive” may be experiencing serious issues

Rise of gray failures

A component appears to be working but is broken

- Occur across software and hardware stack



Gray Failure: The Achilles' Heel of Cloud-Scale Systems

Peng Huang
Microsoft Research
Johns Hopkins University

Yingnong Da
Microsoft Azu

ABSTRACT

Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems

Haryadi S. Gunawi[1], Riza O. Suminto[1], Russell Sears[2], Casey Golliher[2],
Swaminathan Sunda
Nematollah Bidokht
Kevin Harms[8], Robert
Peter Alvaro[11], H.

[1]University of Chicago,
[7]Los Alamos National Lal
[10]University of Utah, [11]Universi

Cores that don't count

Peter H. Hochschild, Paul Turner, Jeffrey C. Mogul — Google, Sunnyvale, CA, US
Rama Govindaraju, Parthasarathy Ranganathan — Google, Sunnyvale, CA, US
David E. Culler, Amin Vahdat — Google, Sunnyvale, CA, US

Abstract

MI, USA. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3458336.3465297

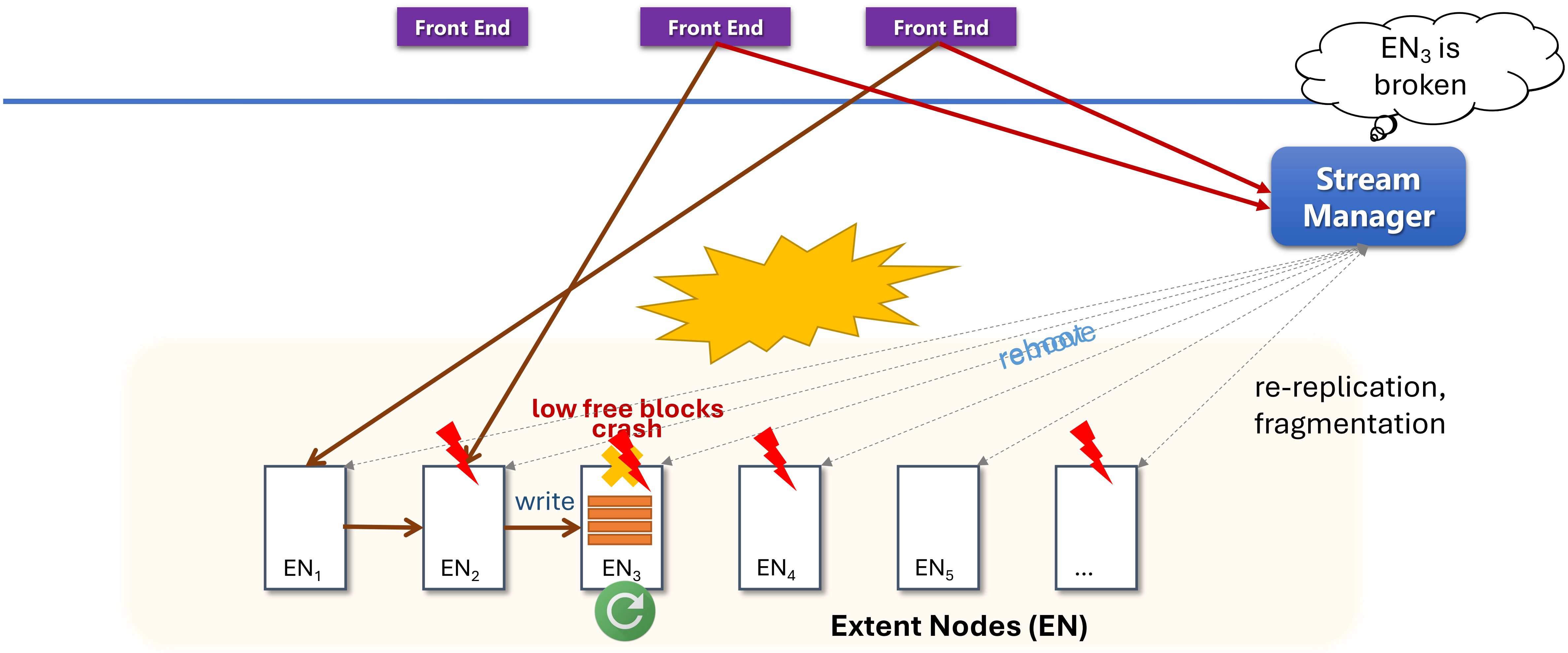
Huang et al., HotOS '17

Gunawi et al., FAST '18

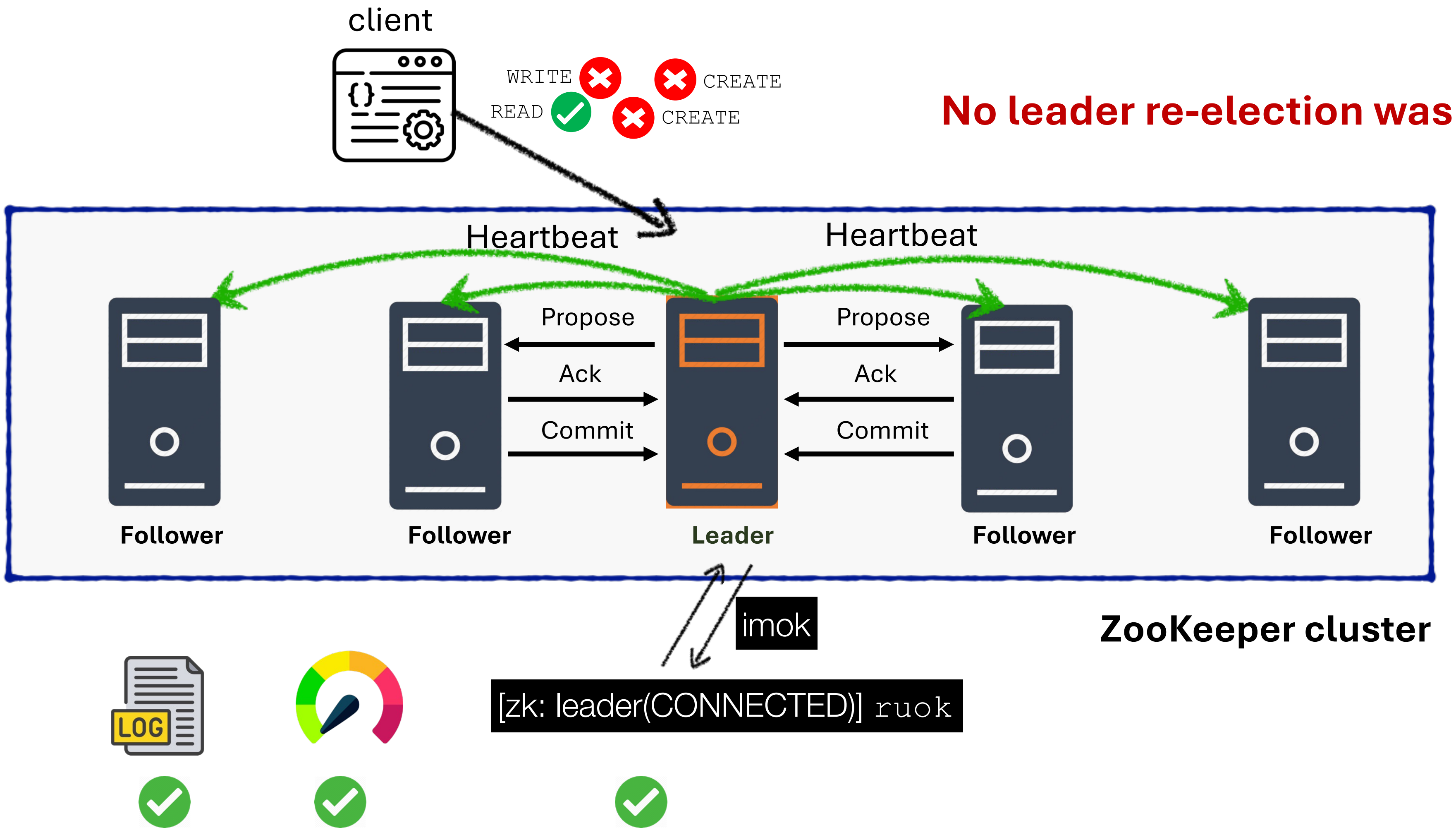
Hochschild et al., HotOS '21

- A wide variety of subtle symptoms and root causes
  - e.g., exception, zombie thread, thrashing, flaky I/O, random packet loss, silent corruption

Case 1: Distributed storage service



Case 2: Distributed coordination service



No leader re-election was triggered!

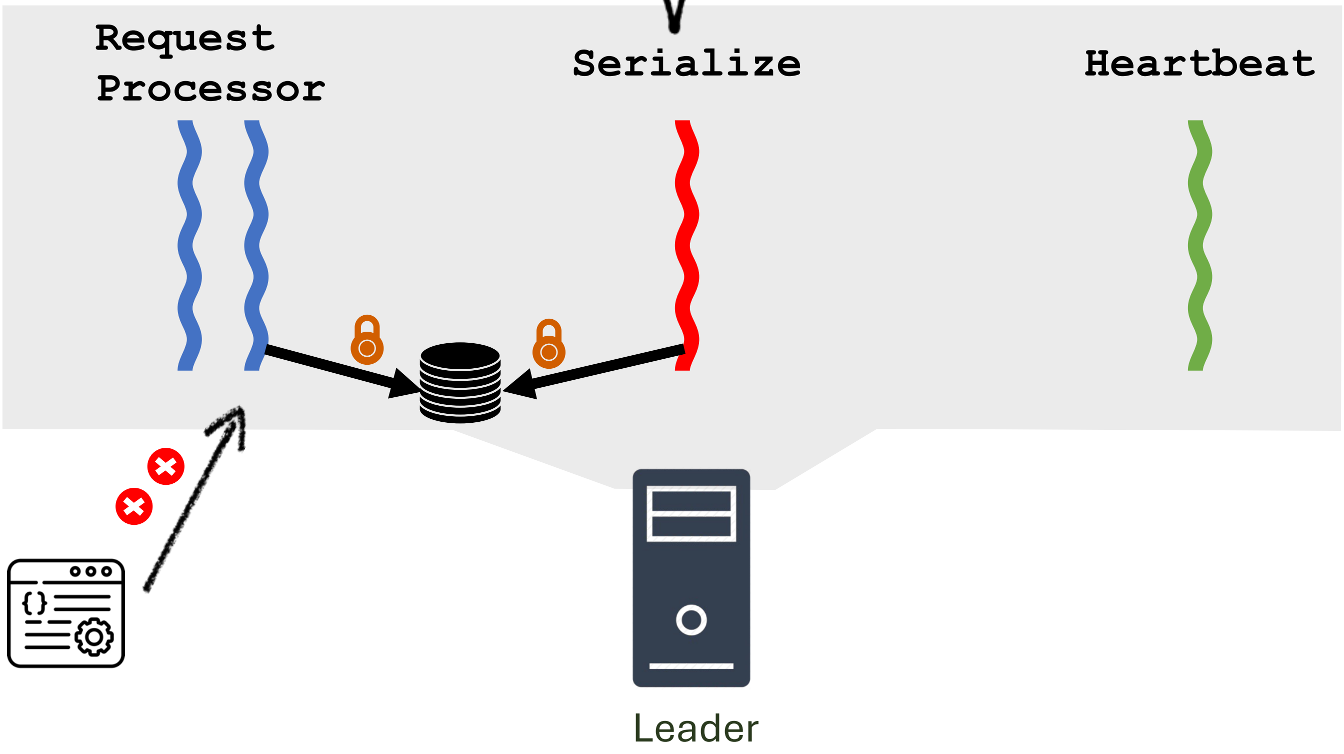
APACHE ZooKeeper™

ZooKeeper cluster

Failure root cause

```
synchronized (node) {
    output.writeString(path, "path");
    output.writeRecord(node, "node");
}
```

Stuck due to transient network issue



https://www.usenix.org/conference/srecon16/program/presentation/nadolny

The many faces of gray failure

“ *A performance issue.* ”

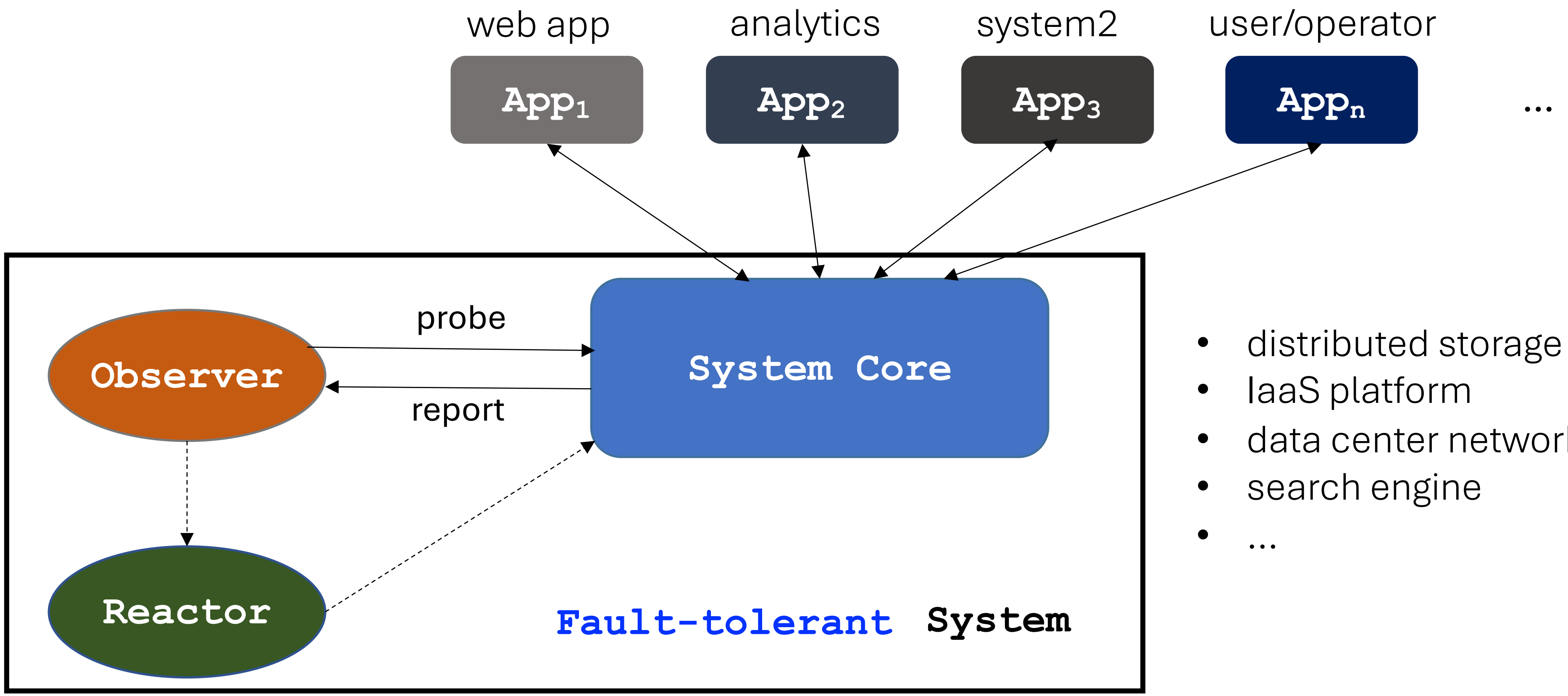
“ *A Heisenbug, sometimes it occurs and sometimes it does not.* ”

“ *The system is failing slowly, e.g., memory leak.* ”

“ *An increasing number of transient errors in the system, which results in reduced system capacity.* ”

……

An abstract model



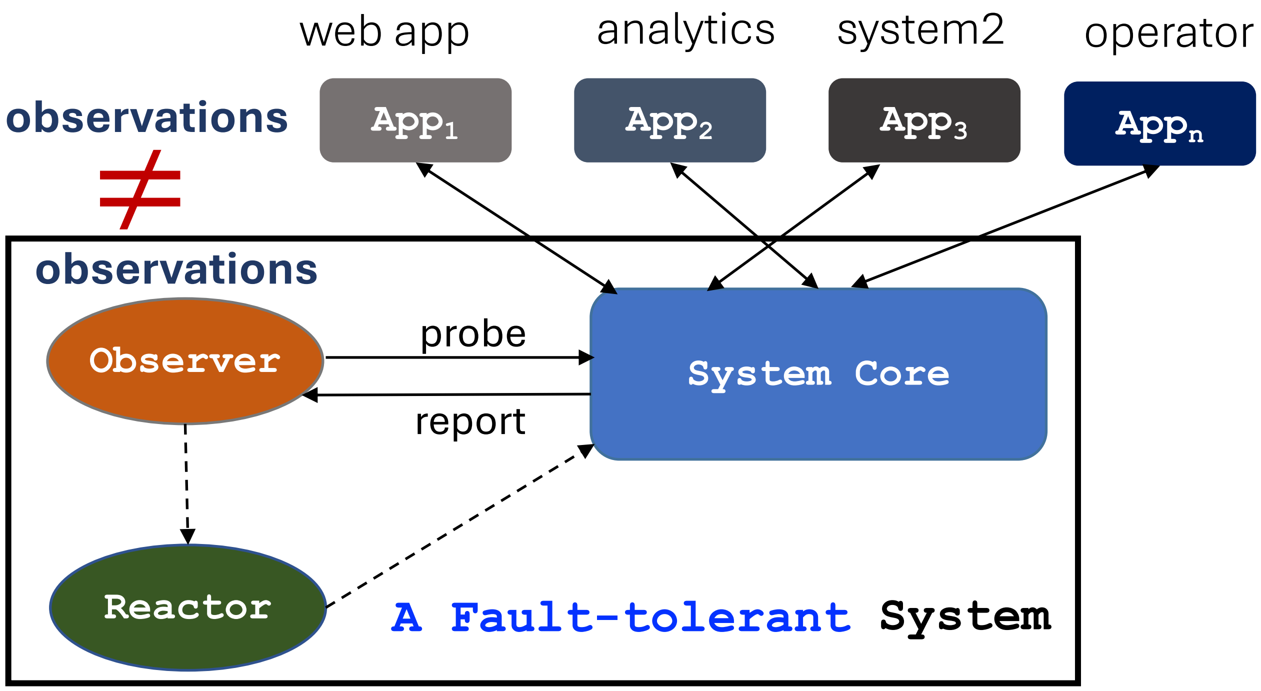
- distributed storage system
- IaaS platform
- data center network
- search engine
- ...

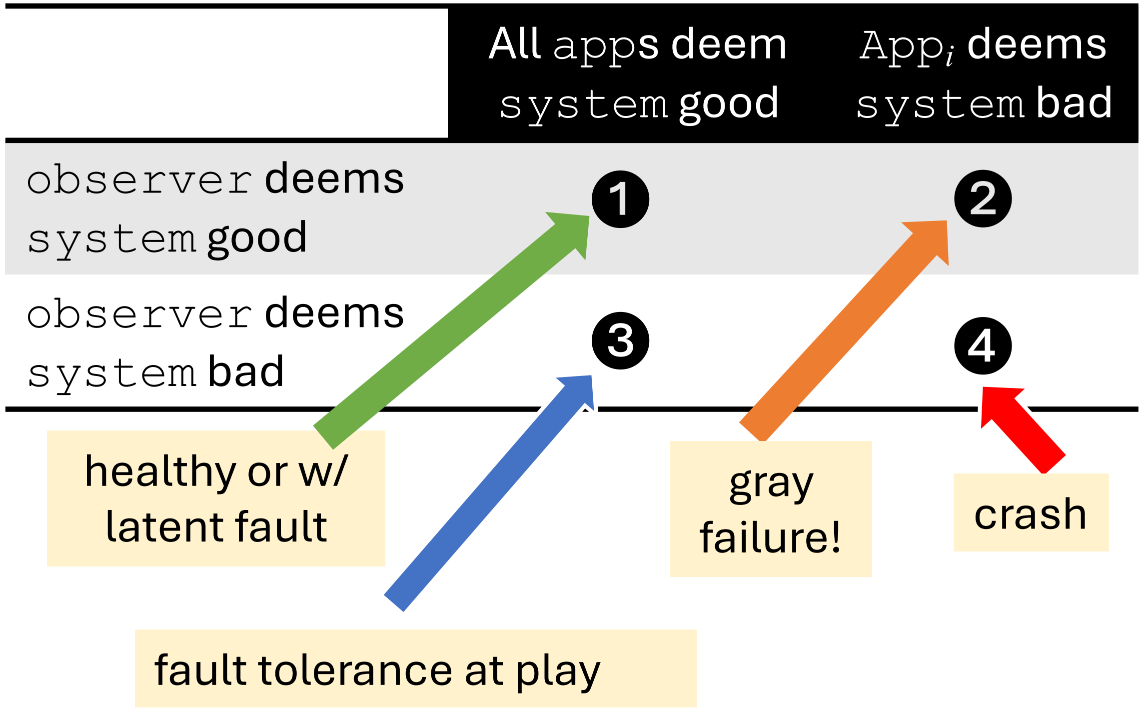
Note: these are logical entities

Key trait of gray failure: *differential observability*

different entities come into different conclusions about whether a system is working or not

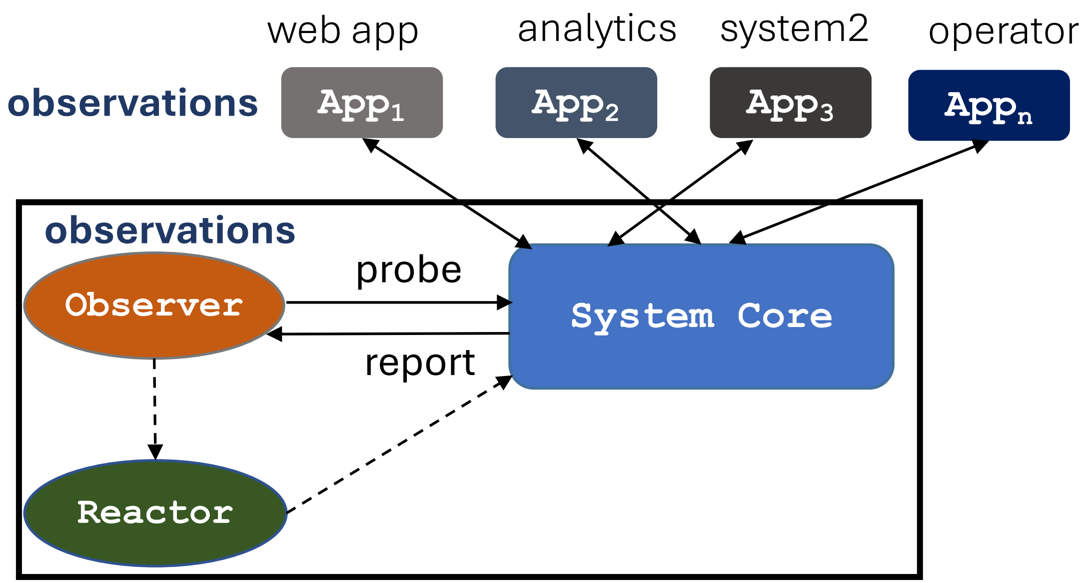
[HotOS '17]





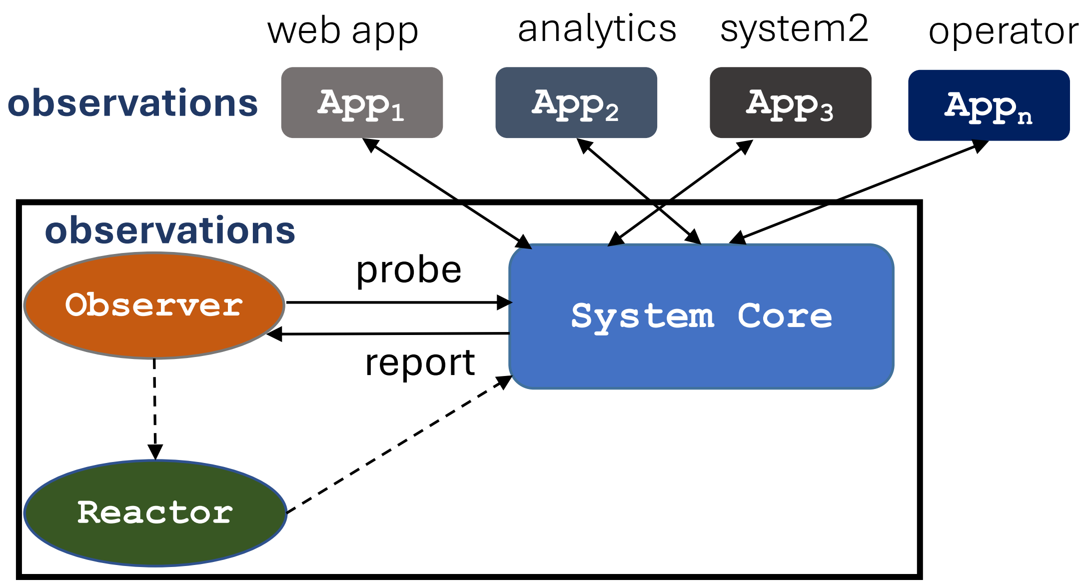
Take-away principles:

1. Close the observation gap
   - *Nines/heartbeats are not enough*
   - Multi-dimensional signals



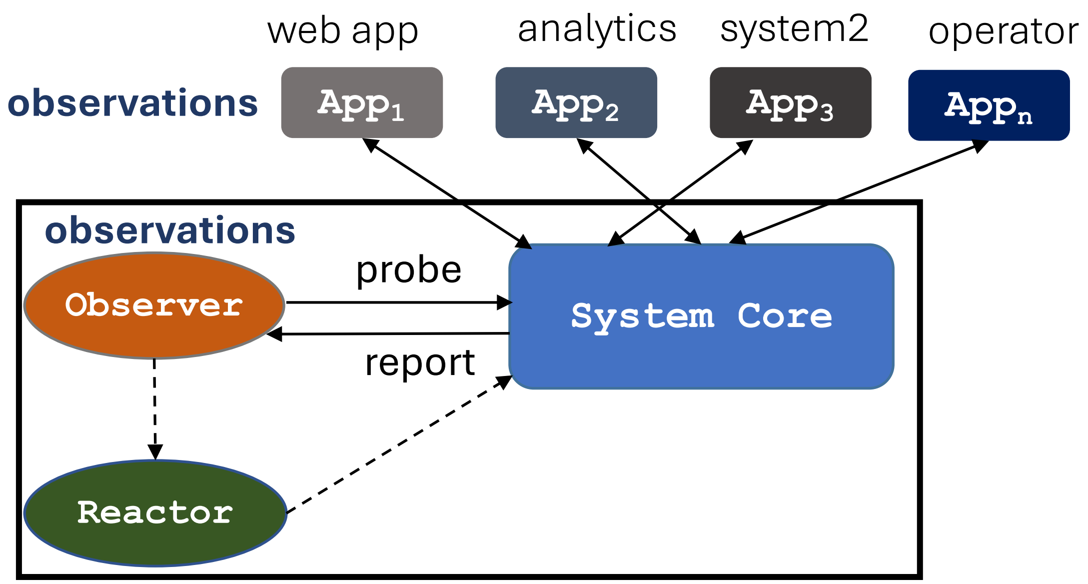
Take-away principles:

1. Close the observation gap
2. Approximate application view
    - Infeasible to eliminate differential observability due to multi-tenancy and modularity constraints
    - Use approximate measurements



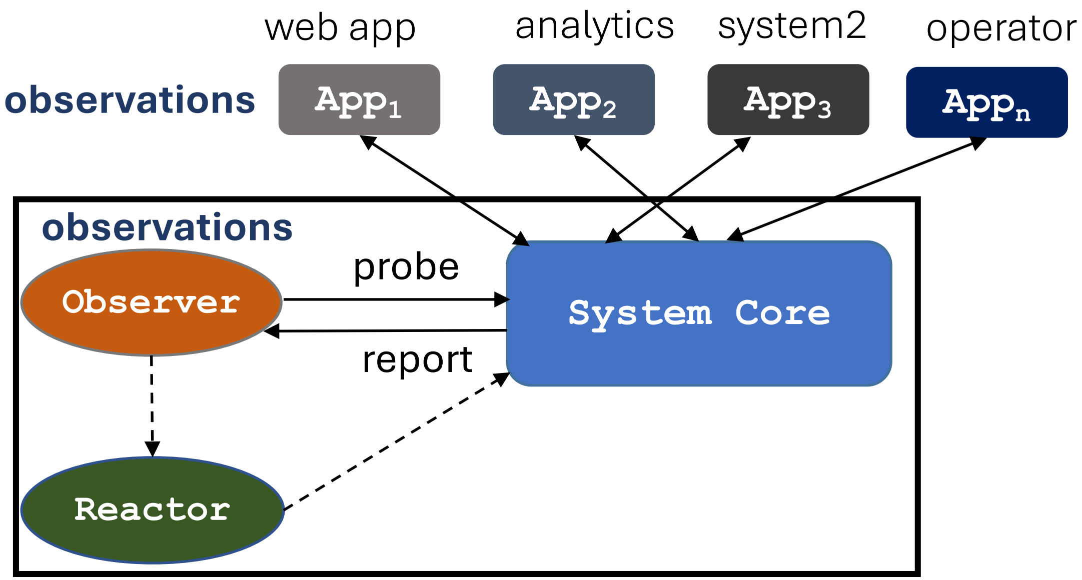
Take-away principles:

1. Close the observation gap
2. Approximate application view
3. Leverage the power of scale
   - Individual component only has a partial view
   - Break isolated observations
   - Address “blame game”

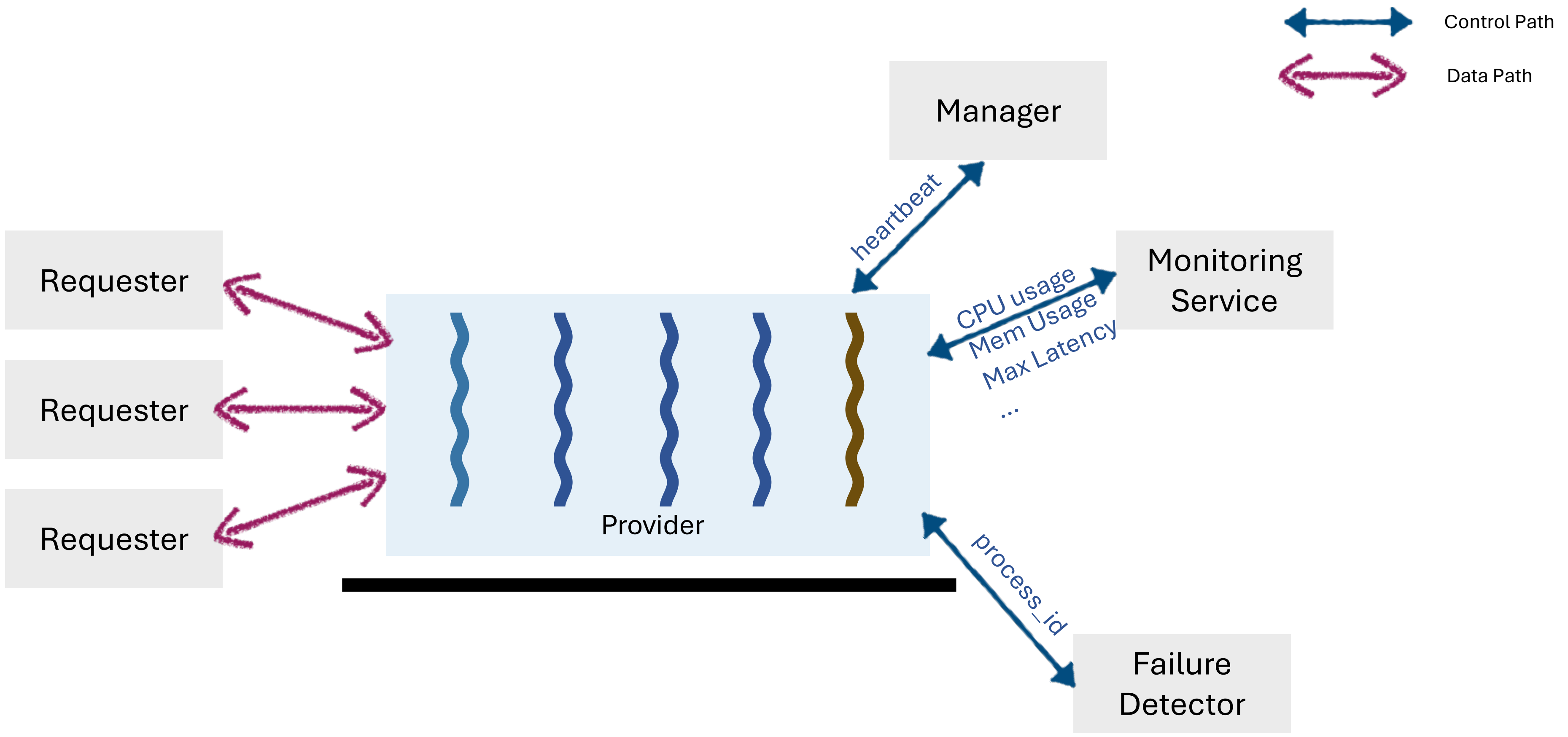


Take-away principles:

1. Close the observation gap
2. Approximate application view
3. Leverage the power of scale
4. Harness the temporal patterns
    - Evolution of gray failures over time



System approach to address gray failures

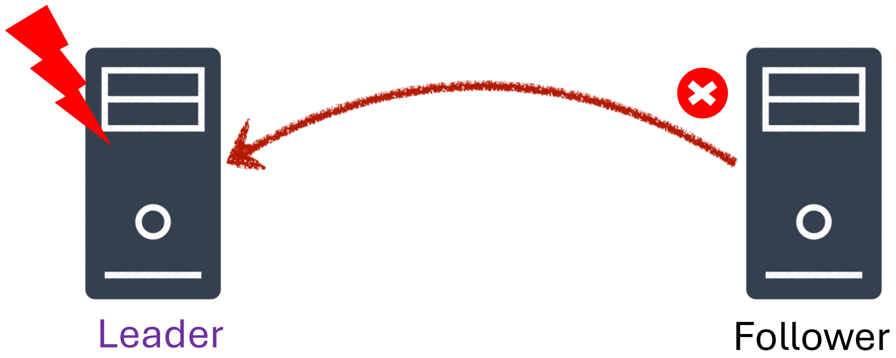


Insight: detect what the *requesters* see

A new approach: in-situ observers

Any system component can directly act as an *in-situ* observer

- during its execution, gather evidence about other components in situ



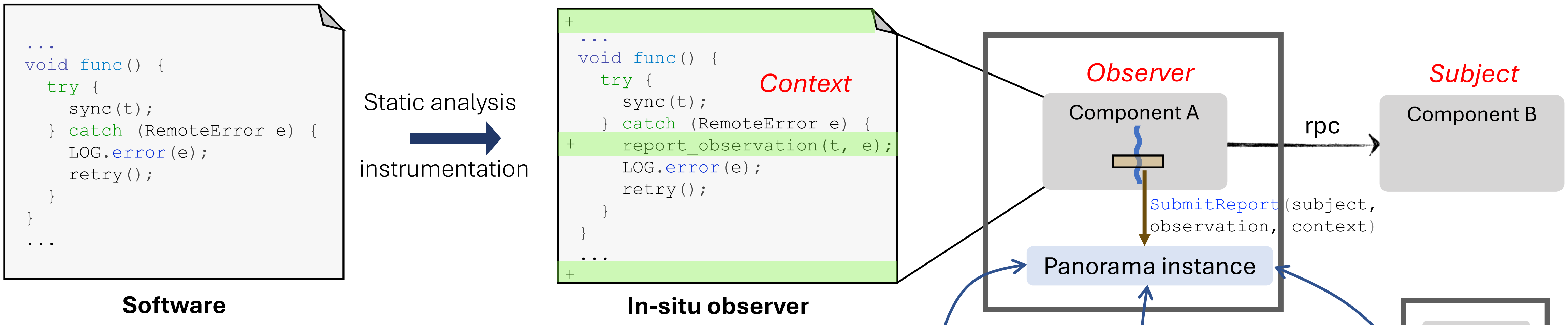

```
void syncWithLeader(long newLeaderZxid) {
   try {
        deserializeSnapshot(leaderIs);
        String sig = leaderIs.read("signature");
        if (!sig.equals("BenWasHere"))
            throw new IOException("Bad signature");
        } else {
            LOG.error("Unexpected leader packet.");
            System.exit(13);
        }
    catch (IOException e) {
        LOG.warn("Exception sync with leader", e);
        sock.close();
    }
}
```

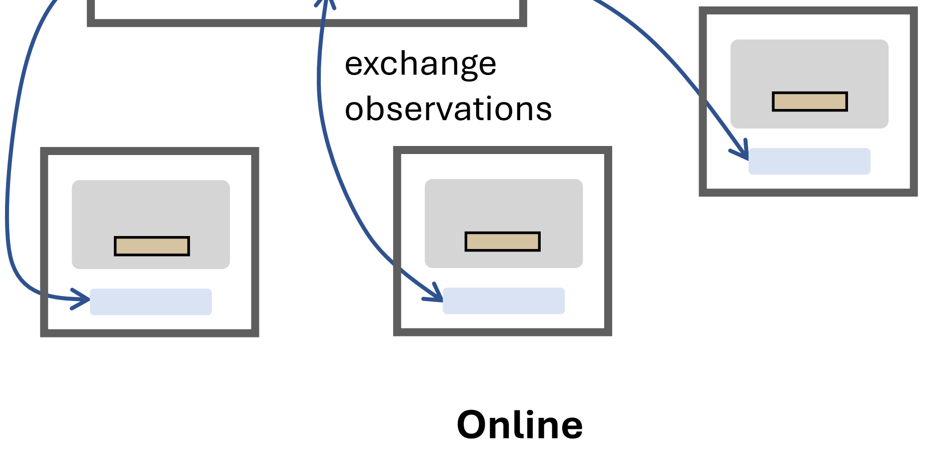
Challenge: modularity principle

- a component has incentives to handle others' errors, but may not for reporting
- need automated method to capture observations from existing code

Panorama: capturing system observability

[OSDI '18]





- A tool to convert a program into an observer
- Uniform observation abstractions
- A generic failure detection service for any component to participate

Convert component into in-situ observer

Goal: find instructions in a program that can potentially provide error evidence about *other programs*

Challenge: such instructions are scattered in the source code

Program analysis to systematically instrument observation hooks

Step 1

locate boundary-crossing calls *(ob-boundary)*

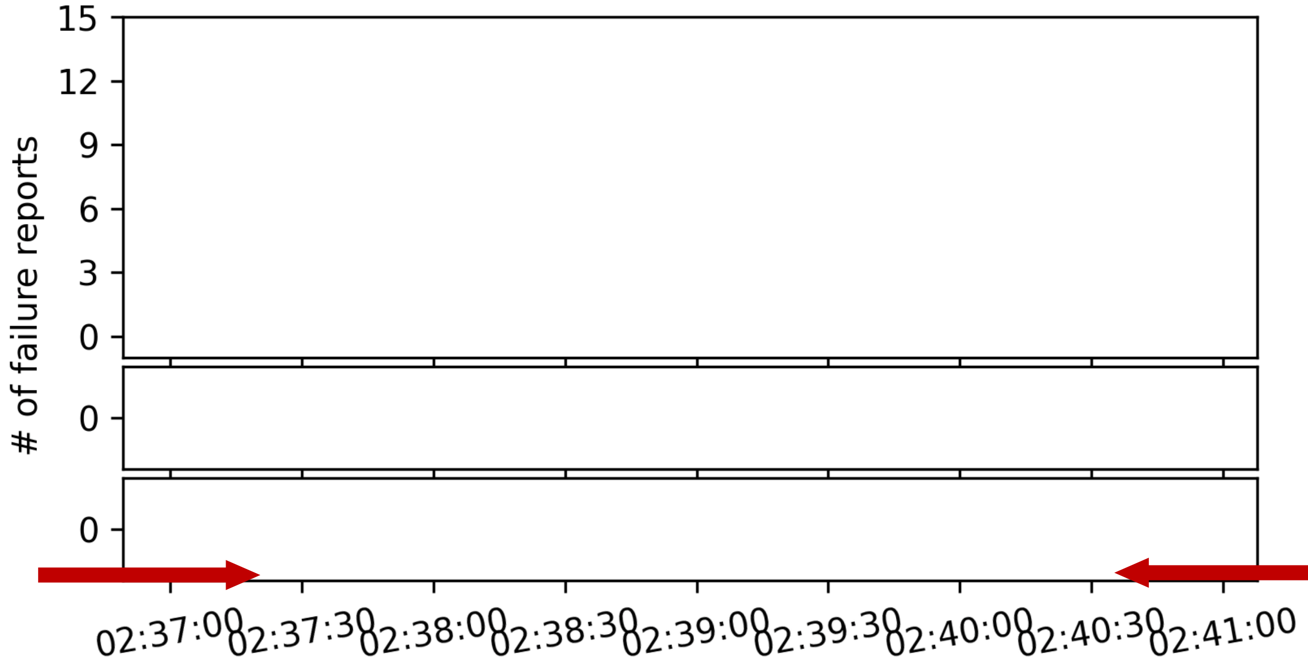
Step 2

identify the *observer* and the *subject*

Step 3

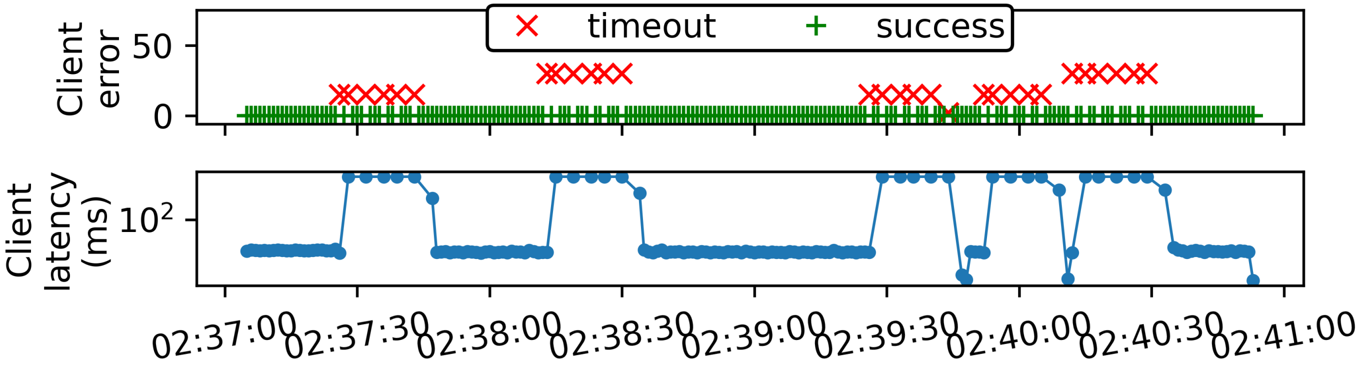
extract observation point *(ob-point)*

Detecting the ZooKeeper gray failure



failure starts

failure clears



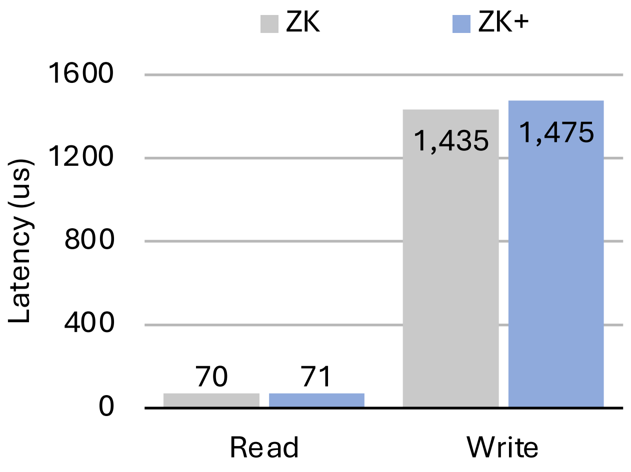
client view

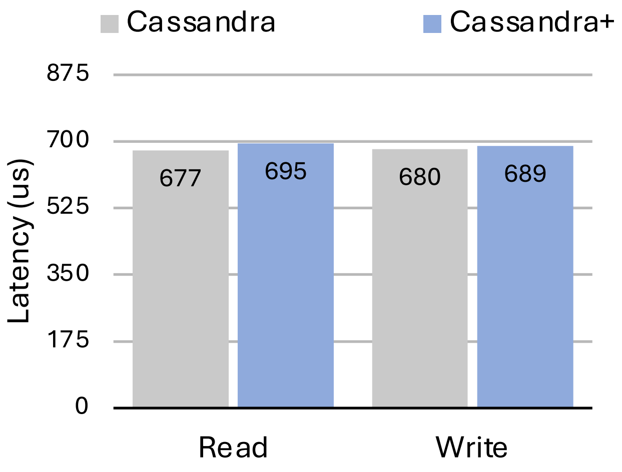
Latency overhead to observers

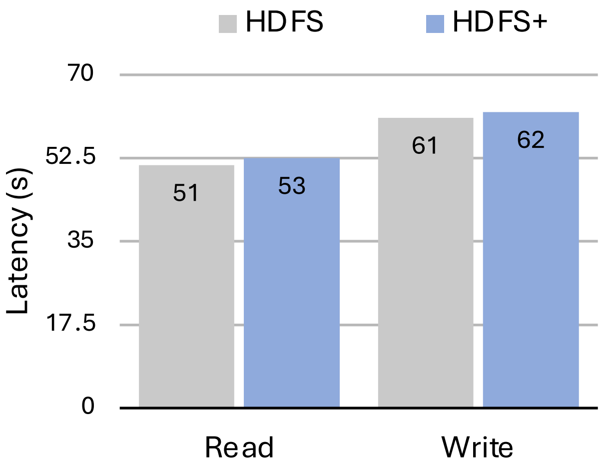
local RPC | library call | RPC

Report	ReportAsync	Judge	Propagate
114.6 μs	0.36 μs	109.0 μs	776.3 μs

main overhead perceived
by the in-situ observer





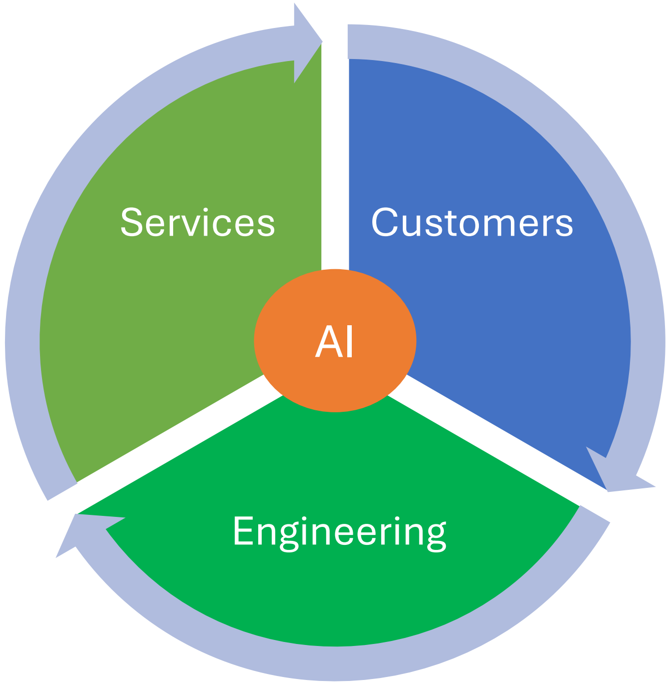


Less than 3% latency overhead

Case Studies: How Microsoft Azure Core AIOps Applies the Differential Observability Model

4 case studies to demonstrate the 4 principles in differential observability model

AIOps for Azure Core Infra Quality & Customer Experience



Integrating AI into how we build and operate Azure

Quality & Customer experience related AIOps projects in Compute

AI for Systems

- VM Pre-provisioning: Prediction + optimization) **WWW '23, IJCAI '20**
- Host resilience [Deep Learning + Multi-bandit] **OSDI '20**
- Disk/Memory Failure Prediction [Deep Learning + Assembly Tree] **OSDI '22**
- Spot VM Harvest optimization [Prediction + optimization] **AAAI '21**
- ....

AI for DevOps: Regression prevention and monitoring

- Safe Deployment and Change Management **NSDI '20, ICSE '23**
- Anomaly Detection + Correlation **KDD '21**
- Host health governance [Anomaly Detection + Correlation] **OSDI '22**
- Pre-production: Graph theory-based experiment design + A/B comparison
- ....

AI for Customers

- LLM and Chatbot
- Self-Help Recommendation Systems
- ...

Apply Differential Observability Model in Azure: From Data to Actions

Data

- Service
- System
- DevOps process

Insights

- Detect
- Diagnose
- Predict
- Optimize

Actions

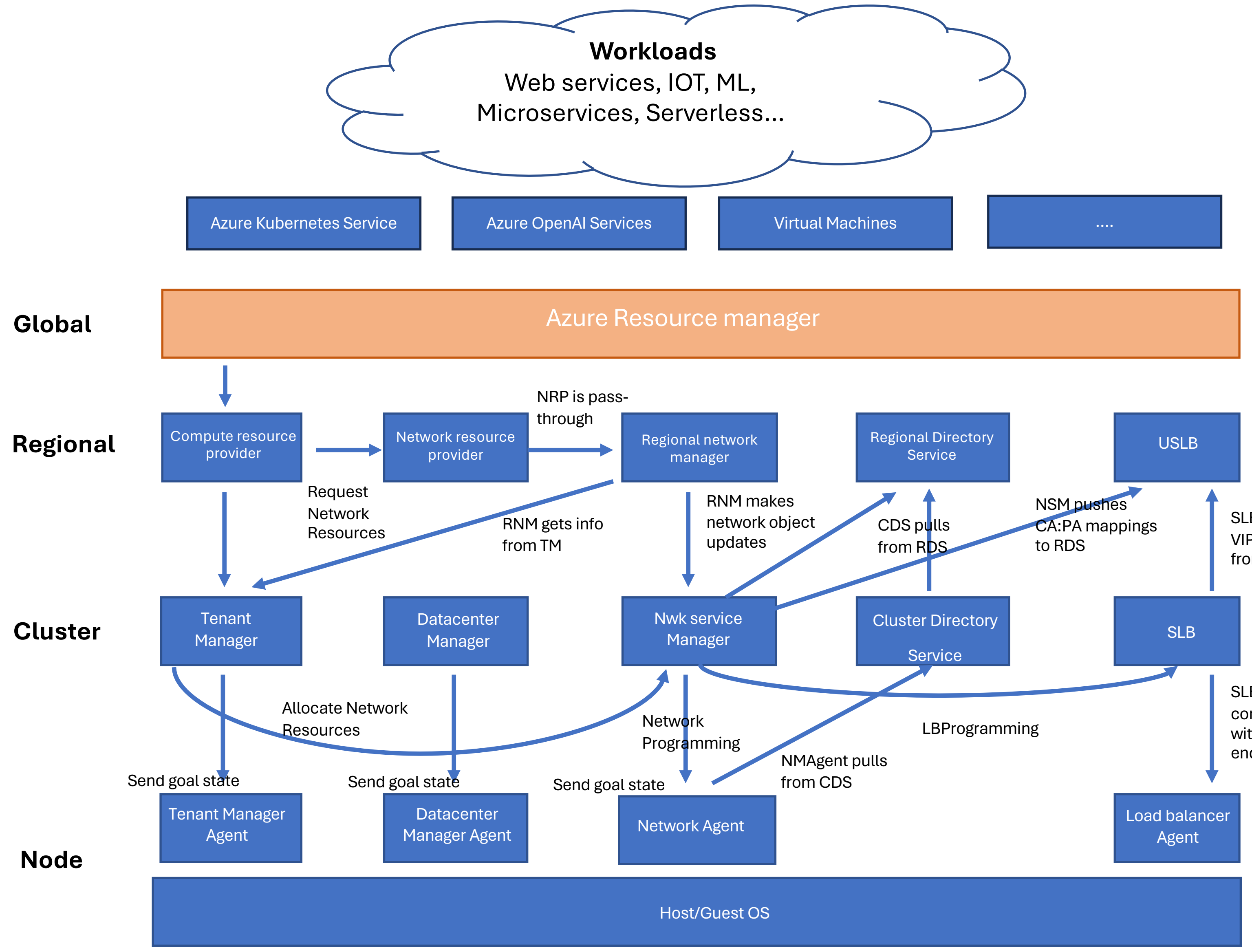
- Mitigate/Resolve
- Avert future pain
- Optimize resource allocation
- Improve architecture & process


1D
1B


6AZB

Heterogenous systems in hyperscale bring complexity in detection

- Microsoft Azure has 62 + regions and 200+ datacenters globally
- Complex interactions between agents in different cloud levels
- Need careful design on applying differential observability model in hyper-scale system



Closing the Observation Gap

Case Study 1: Applying Closing Observation Gap Principle in Guest and Host Insights Analysis

Closing the Observation Gap: Incorporating Guest VM Insights into Host Infra Monitoring

- Traditional monitoring in **cloud provider** is usually heavily focusing on infra side.

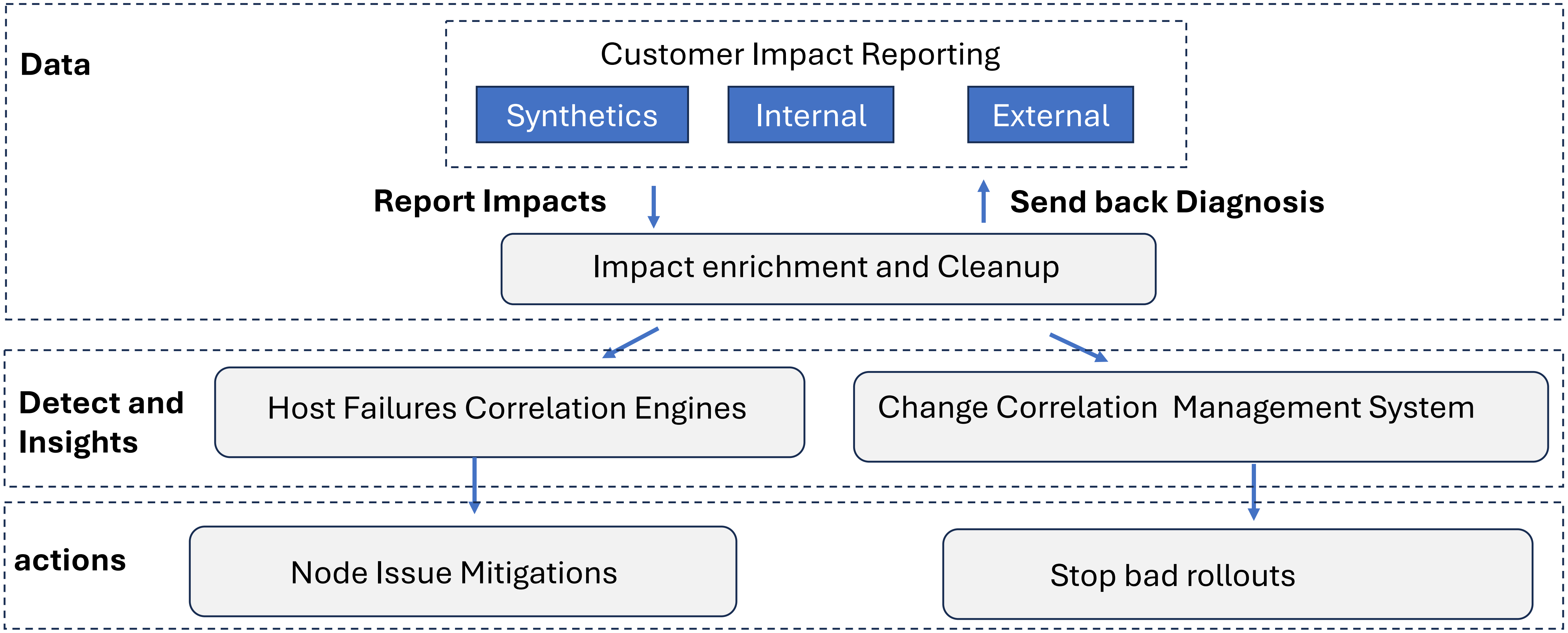
- Service Owner is responsible for service monitoring (e.g. Cassandra service has long read/write delay)

- Blame game between service Issue and host Issue

  - ❑ Time to mitigate for customer support tickets
  - ❑ Time to recover SLI/SLO regressions
  - ❑ Hard to ensure zero workload impacts on infra changes
  - ❑ Hard to meet the diverse workload SLI/SLO requirements

Closing the Observation Gap: Incorporating Guest Insights into Host Health Assessment and Diagnosis

- Empower workload owners to report the guest impacts: [Azure Impact Reporting REST API | Microsoft Learn]
- Run mission critical synthetics workload to understand the workload patterns



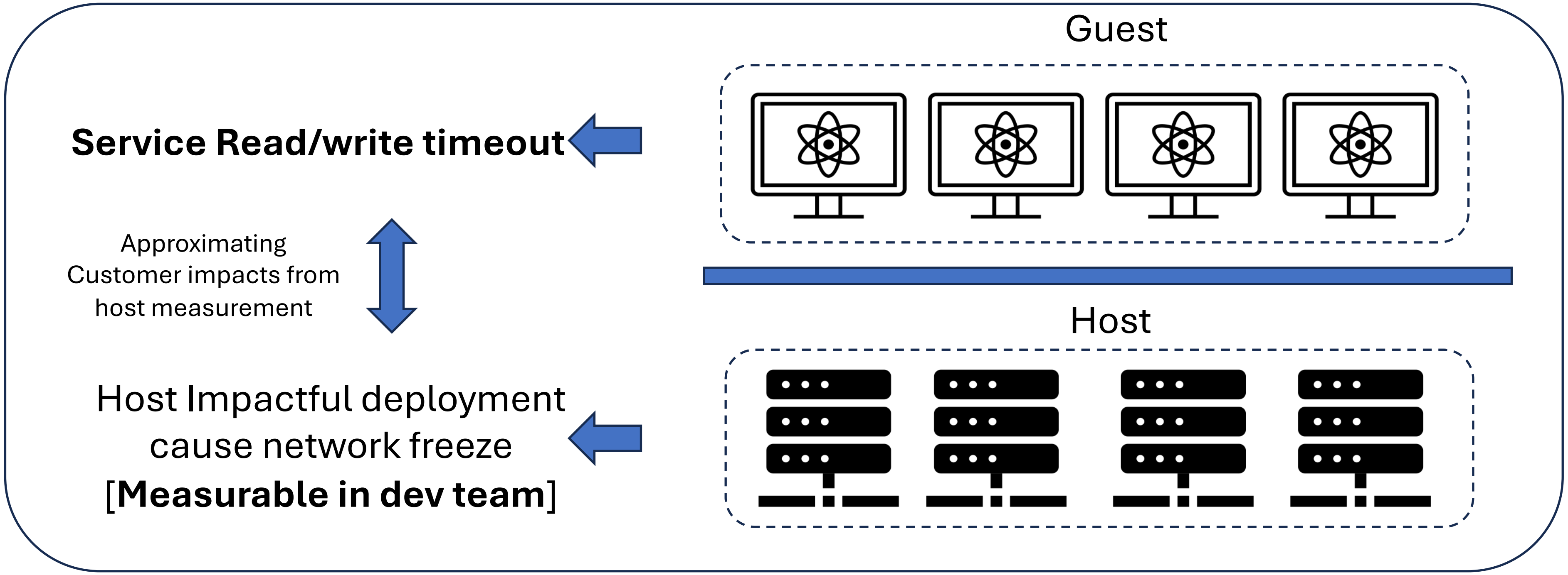
Approximate Application view

Case Study 2: Applying Approximate Application View Principle to Approximate Guest Impacts with Host Impacts

Approximating Customer Impacts based on host impact measurement.

Guest Insights data may not always be available

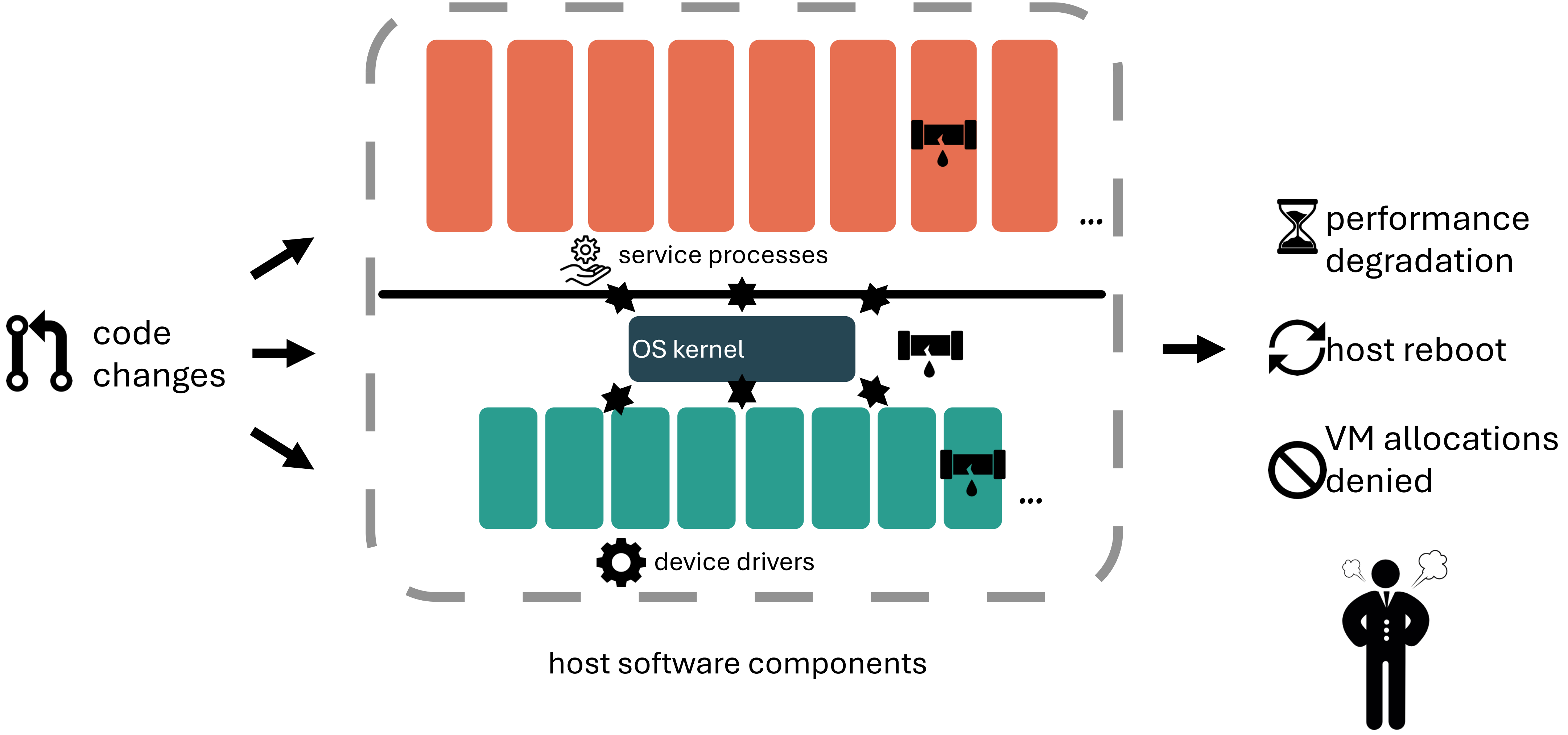
- Compliance and security issues
- High resource consumption for collection certain telemetries



Harnessing temporal patterns

Case Study 3: Applying Harnessing Temporal Patterns Principle in Memory Leak Detection

Memory leak is notorious in cloud and cause gray failures



Challenges of leak detection in cloud

Noisy signals from environment

- many different workloads in the cloud with dynamic characteristics
- easily incur false positives

Slow leaks in long-running services

- memory leaks often last over days or weeks
- need to identify gradual changes

Large profiling data volumes

- need to analyze >10 TB memory usage data daily

Why is leak detection still challenging in cloud?

Extensive work in memory leak detection

Practice 1: static approach

- statically analyze the source code
- no runtime overhead

Limitations

- inaccurate and not scalable to large systems

Practice 2: dynamic approach

- instrument programs and track the object lifetime at runtime
- more accurate

Limitations

- intrusive and high overhead

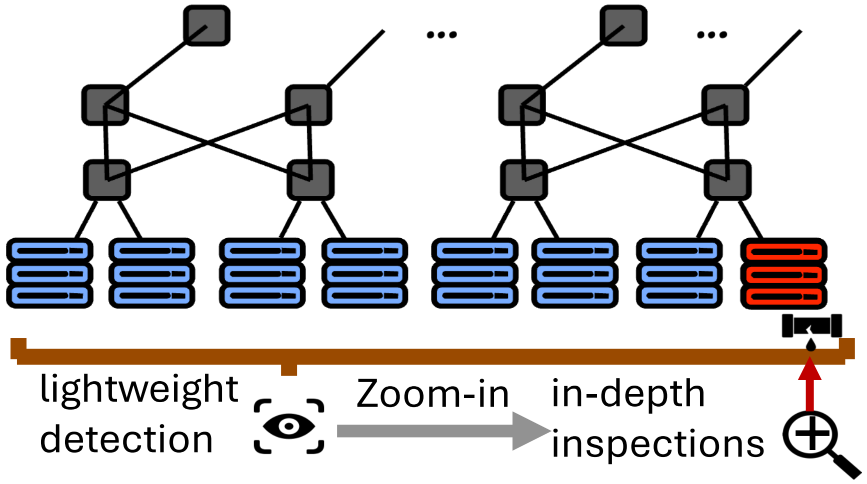
Hard trade-offs among **accuracy**, **overhead**, and **scalability**

RESIN: exploiting temporal patterns

[OSDI '22]

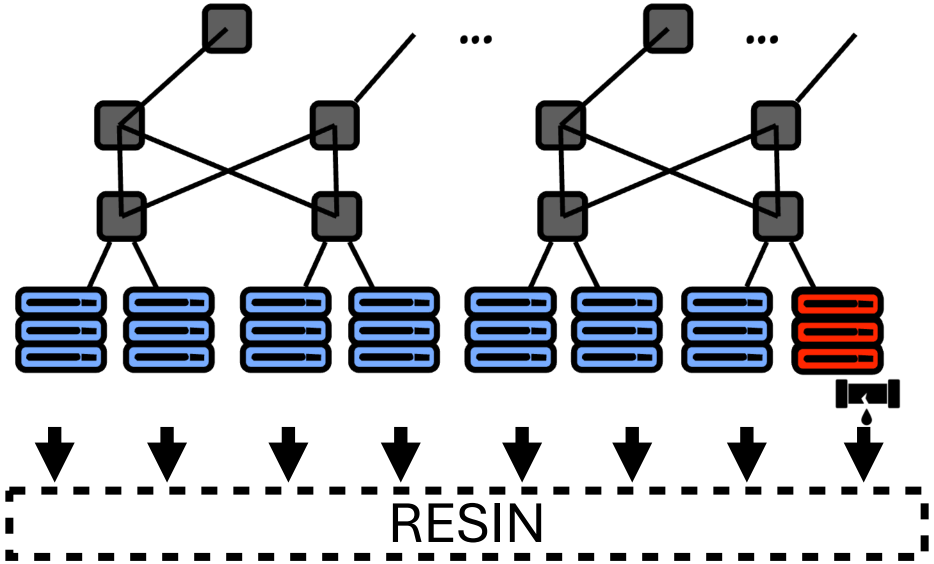
Insight 1:

- separate detection and pinpointing problems
- decompose detection to multi-stages



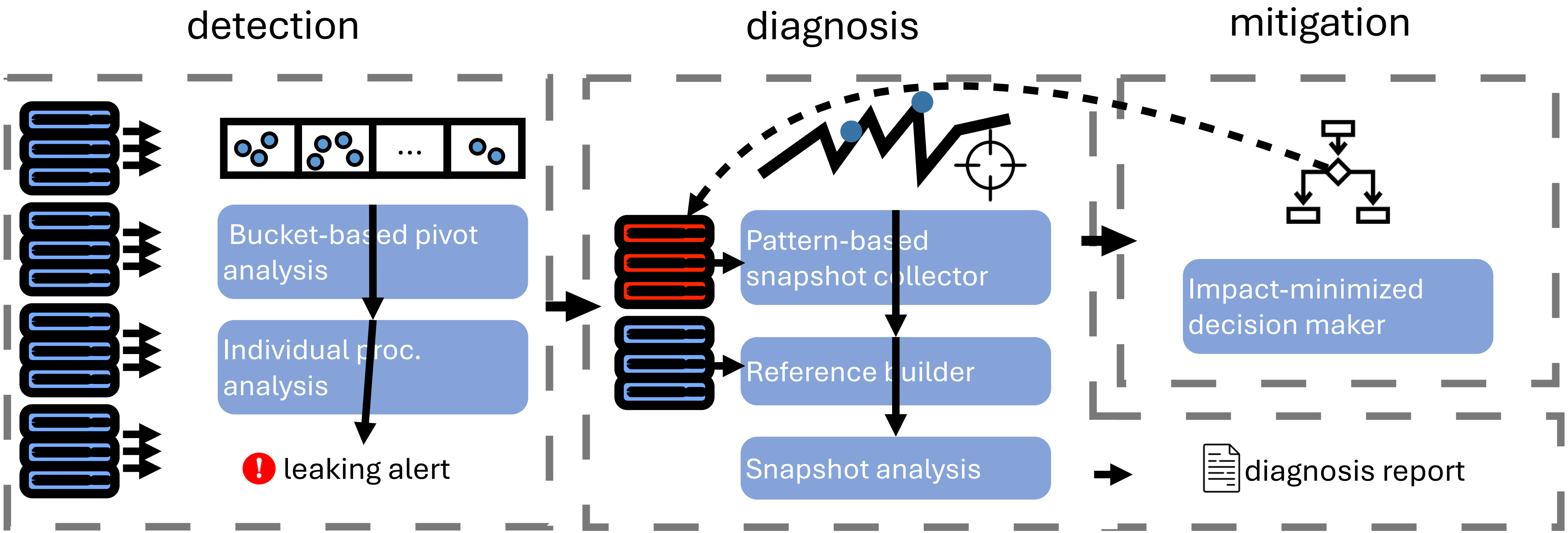
Insight 2:

- a centralized approach for all components
- leverage temporal patterns at scale to improve accuracy



Achieve high accuracy, scalability, and low overhead

Overview of RESIN



Bucket-based pivot analysis

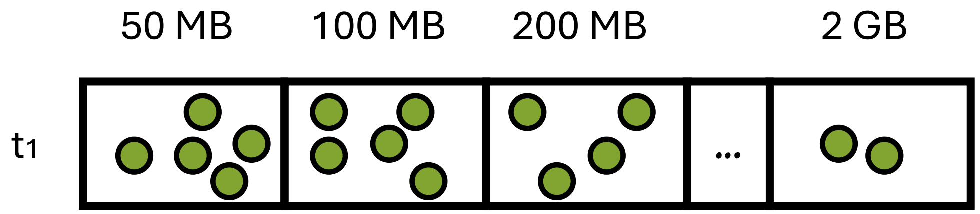
Each bucket is a collection of hosts with memory usage in a same range

- bucketization is done per component
- e.g., 50MB-bucket includes hosts running firewall services with usage 50MB-100MB

Insight: monitor trend of bucket size instead of individual component usage

- robust to tolerate noises due to workload effect (challenge 1)
- scalable to large clusters with massive hosts (challenge 3)

Time stamp	ImageName	Cluster	NodeId	PID	Private Usage	...
t_1	**firewall.exe**	NorthUS-1da	9das-sax1	254	2,334,720	
t_1	**firewall.exe**	NorthUS-9lp	9das-yq0c	979	90,413,120	
t_1	**firewall.exe**	Asia-b2	o1oz-bg75	1375	170,341,311	
t_1	...				...	

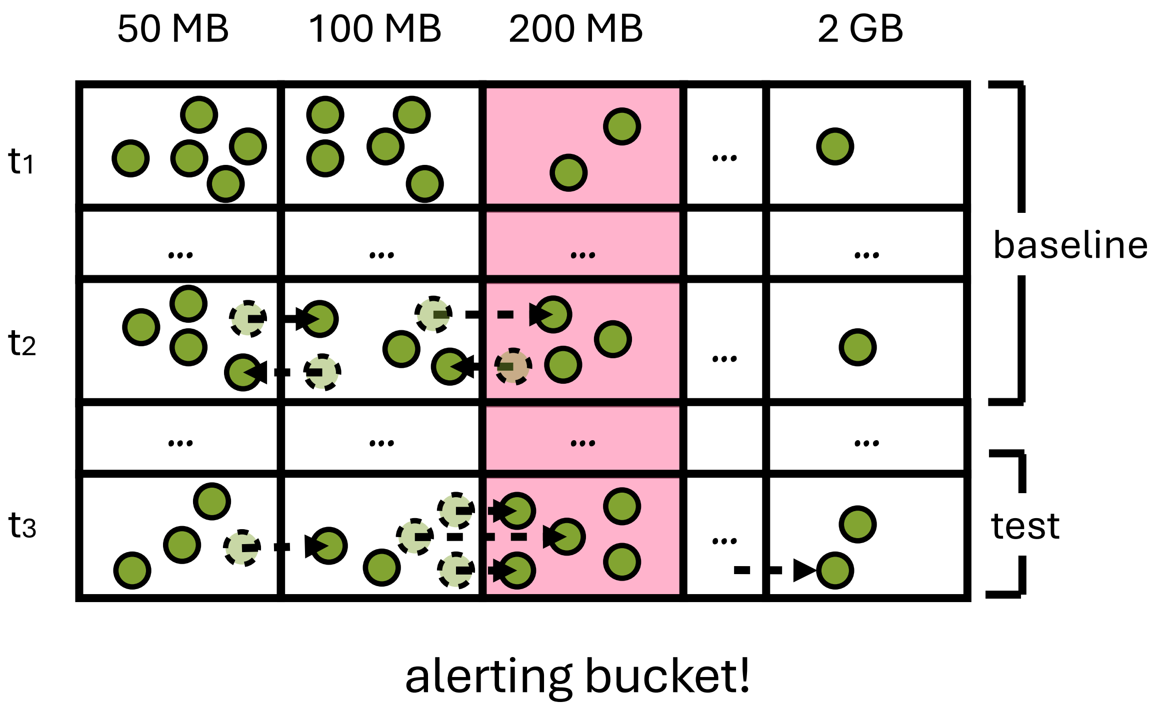


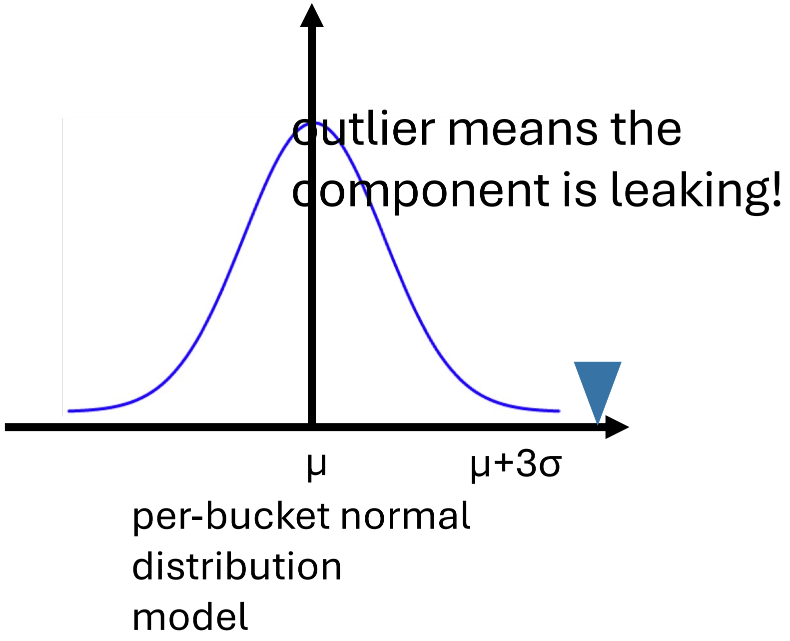
buckets of firewall.exe

Bucket-based pivot analysis

Run anomaly detection against time series of bucket size

- data points that exceed the $\mu + 3\sigma$ [1] of the baseline data are anomaly





[1] mean and standard deviation of the distribution

Second-stage detection: live heap snapshots

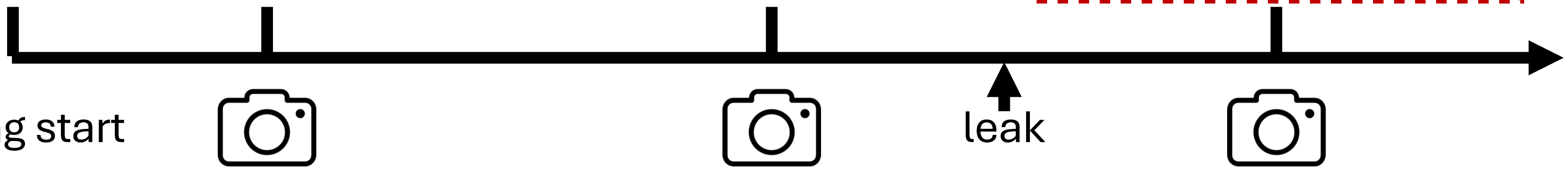
RESIN diagnoses leaks by capturing heap snapshot traces

- wait for leak allocation happens again to trigger completion
- differentiate snapshots before and after memory leak allocation

Alloc Addr	Stack Id	Size	RefCount
0x80000	1	64	2

Alloc Addr	Stack Id	Size	RefCount
0x80000	1	64	2
0x90000	1	128	1

Alloc Addr	Stack Id	Size	RefCount
0x80000	1	64	2
0x90000	1	128	1
0xf0000	2	32	2



RESIN deployment status and scale

Running in Azure production since late 2018

- cover **millions** of hosts
- detect leaks for **600+** host processes
- detect leaks for **800+** kernel pool tags
- the detection engine analyzes more than **10 TB** memory usage data daily
- the diagnosis module collects **56** traces on average daily

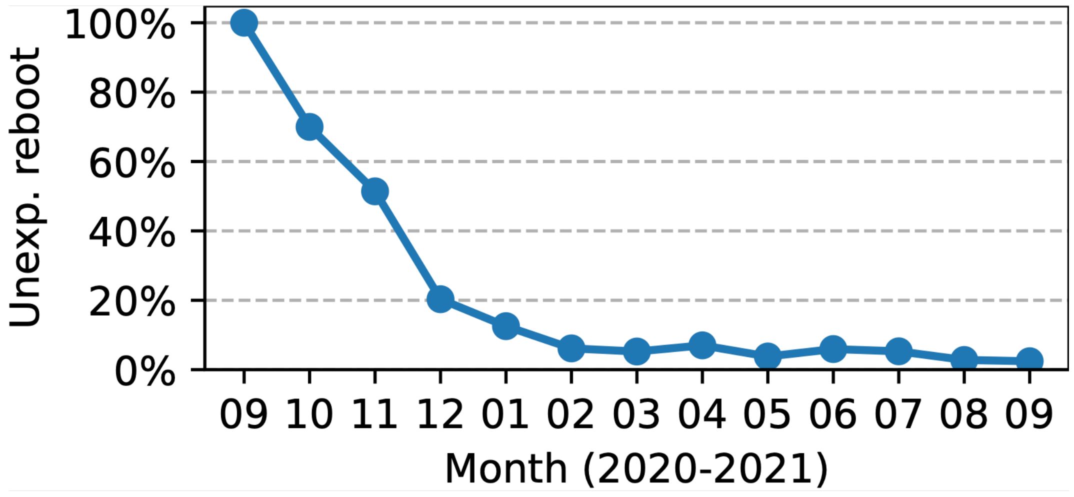
How effective is RESIN?

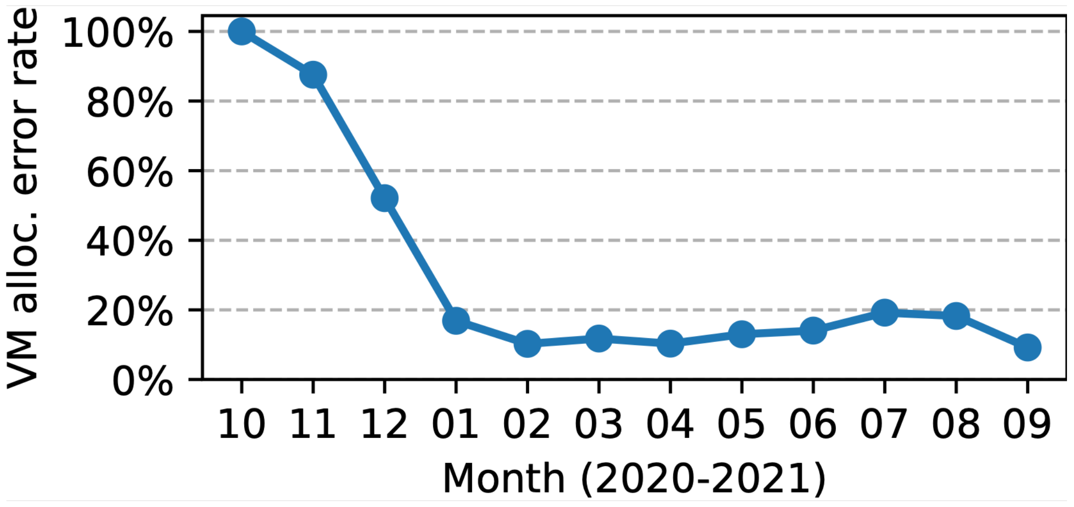
VM reboots reduced by 41x

- average number of reboots per 100,000 hosts per day due to low memory

VM allocation errors reduced by 10x

- ratio of erroneous VM allocation requests due to low memory





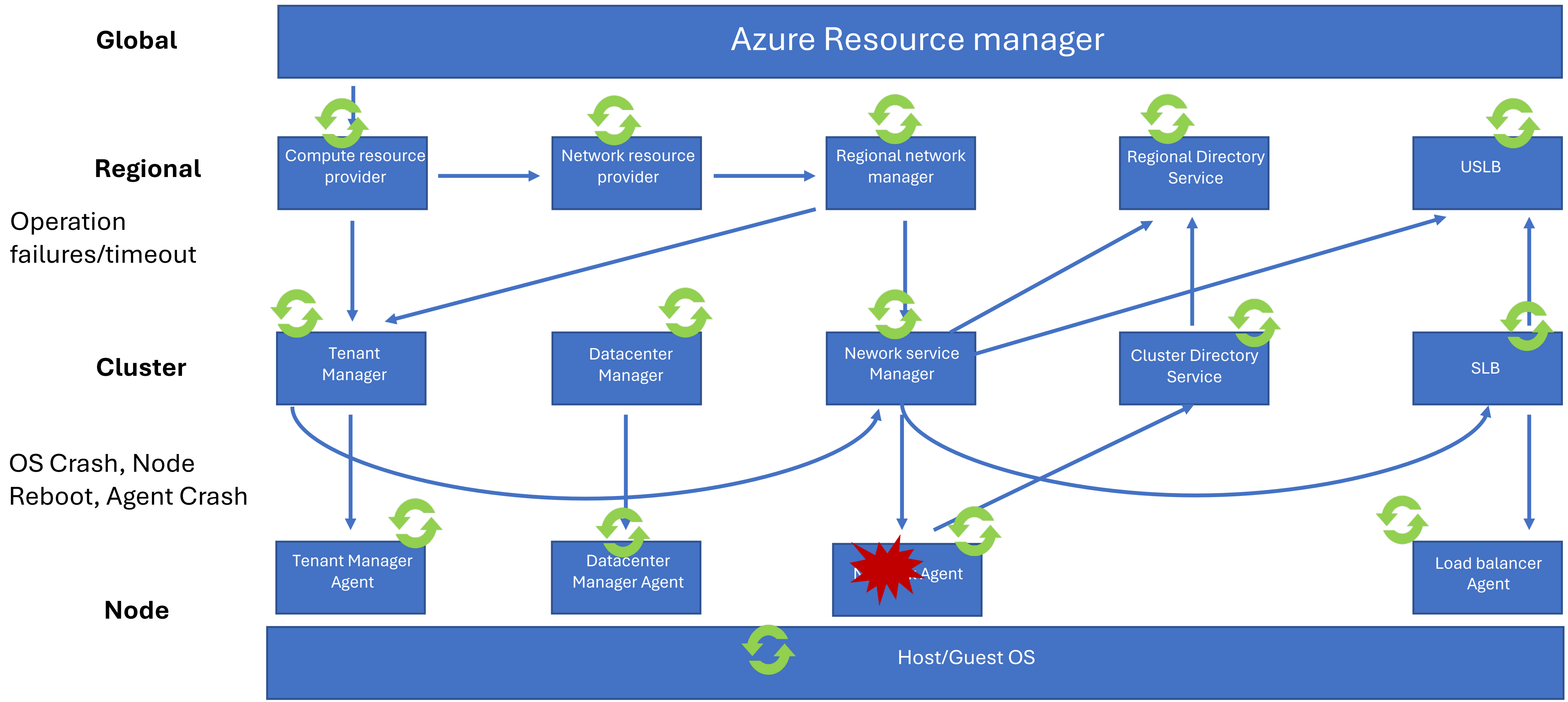
* data is normalized

Leveraging the power of scale

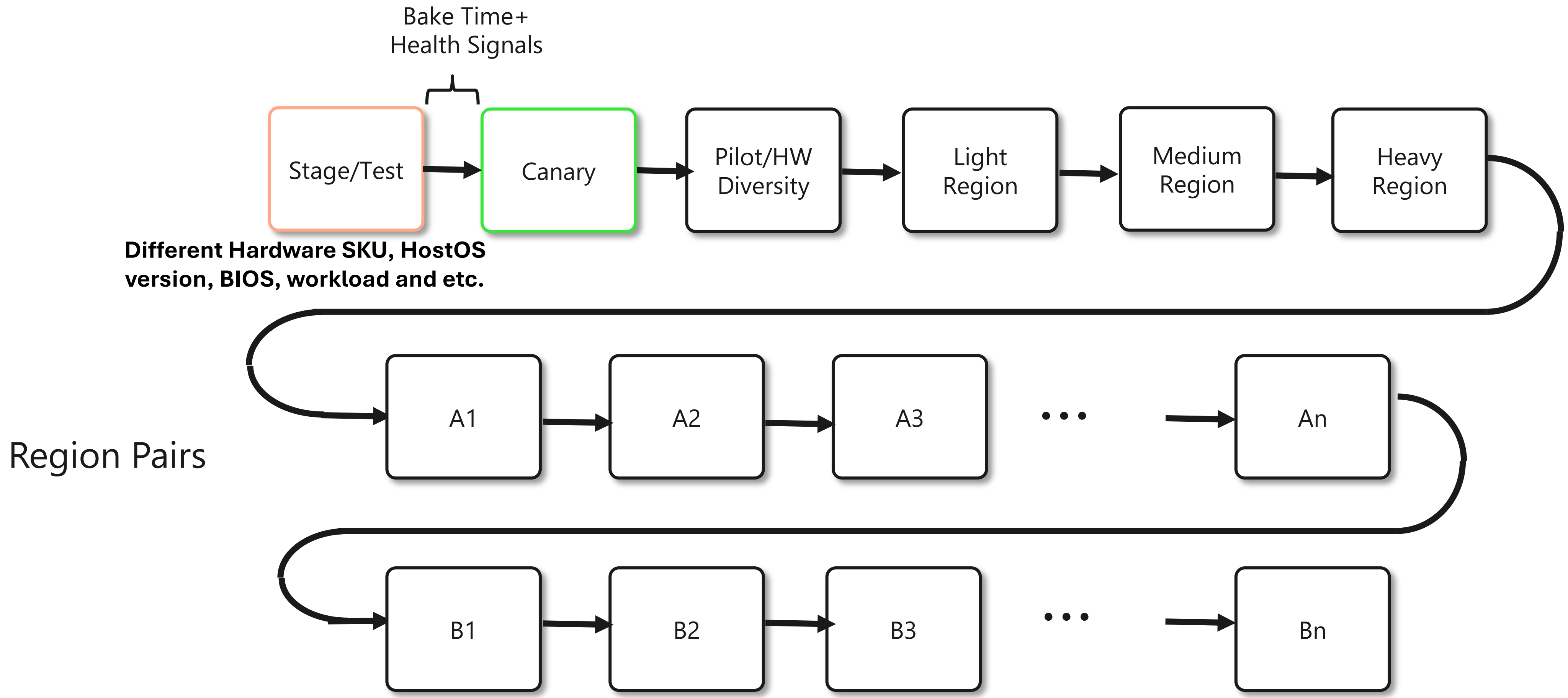
Case Study 4: Safe Deployment

Why is safe deployment challenging?

Microsoft



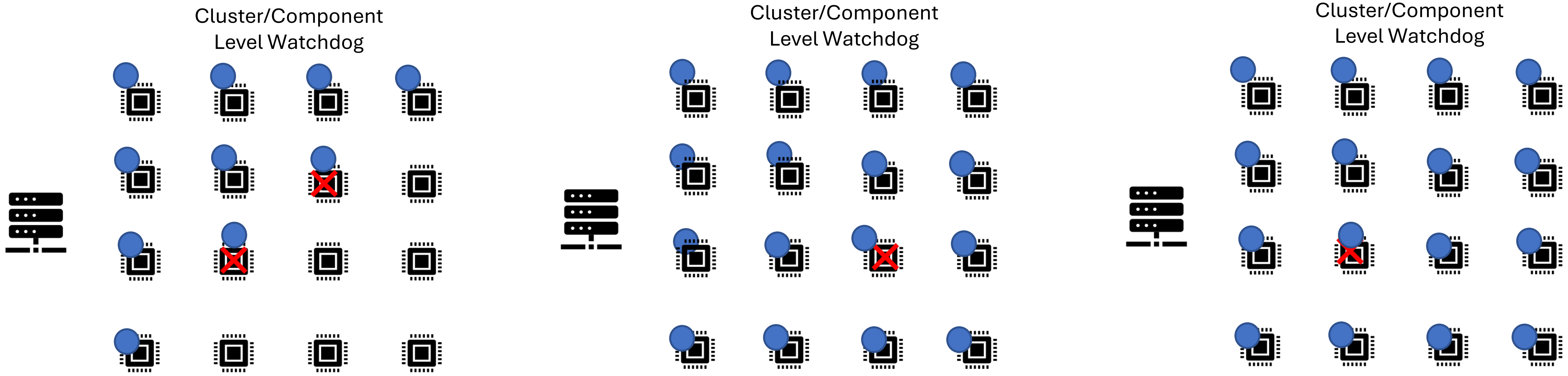
Existing practice: pre-qualification test and safe deployment policy



- Gradual rollout
- Manual go/nogo decision after baking at each step needed

Existing practice: local watchdog

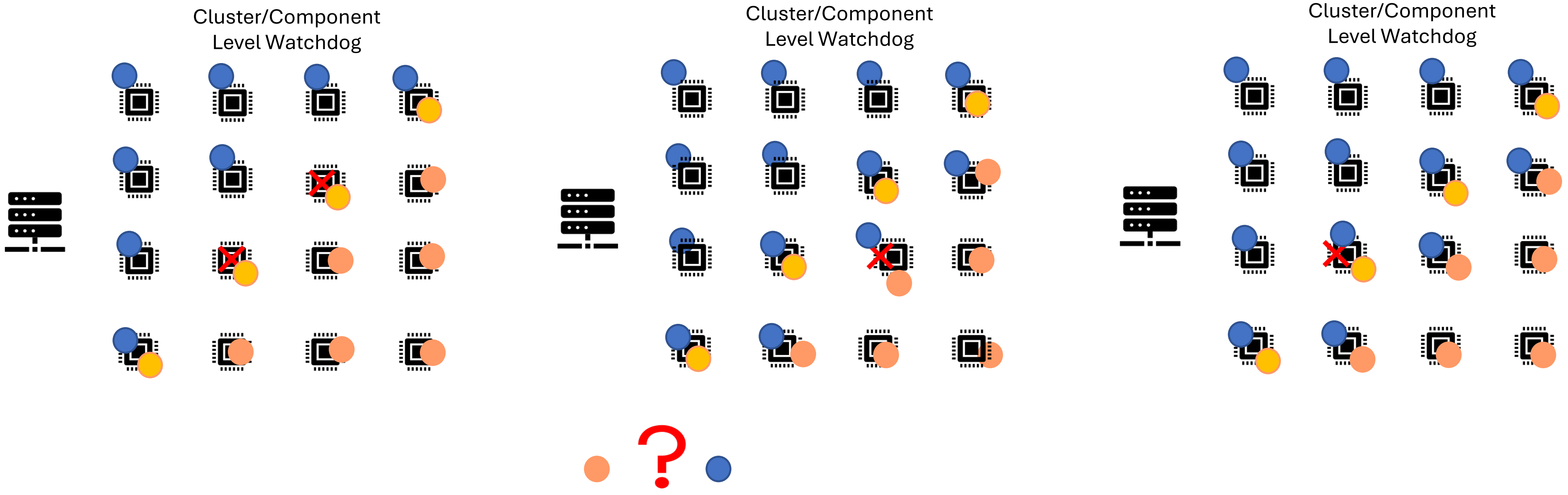
- Threshold-based anomaly detection model
- Cannot detect global issues that are minor in each cluster but severe across the fleet
- Cannot detect latent failures



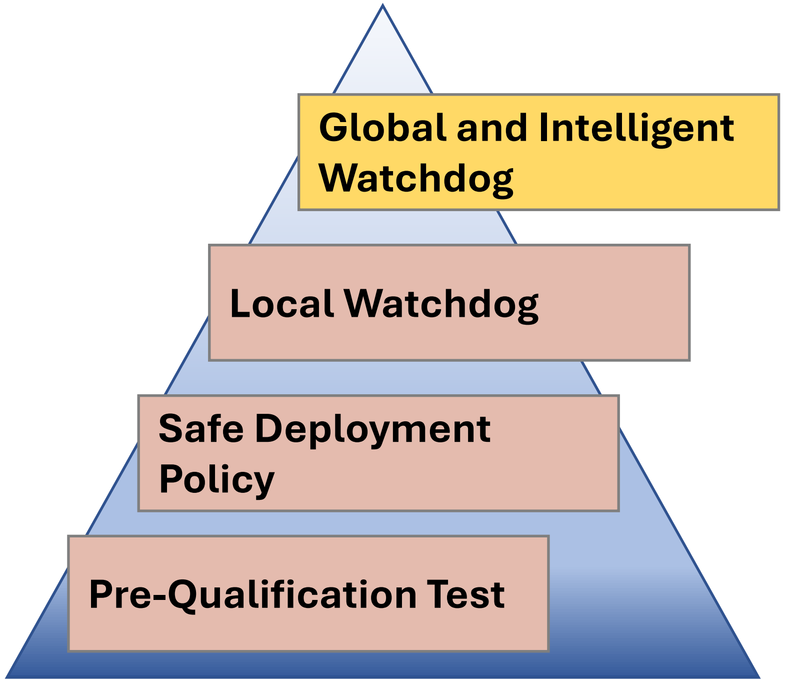
Rollout is stopped at cluster level with failures observed from over x nodes

Existing practice: local watchdog

- Threshold-based anomaly detection at cluster level
- Cannot detect issues that are minor in each cluster but severe across the fleet
- Cannot detect latent failures
- If multiple rollouts happened at the same time, it will randomly blame



Design Goals



- Take advantage of the differential observations across large scale of the cloud system
  - A deployment of an agent take weeks to go over the regions cluster by clusters
  - Different agents landed on a cluster at different time
- Make go/nogo decision recommendations for auto-stop and reduce the baking time

Overview of the Model

[Gandalf: NSDI '20]

- All failure events
- All update events

Anomaly Detection → Ensemble Voting → Spatial & Temporal Correlation → Exponential Decaying

Correlation process: Identify which rollouts are suspicious

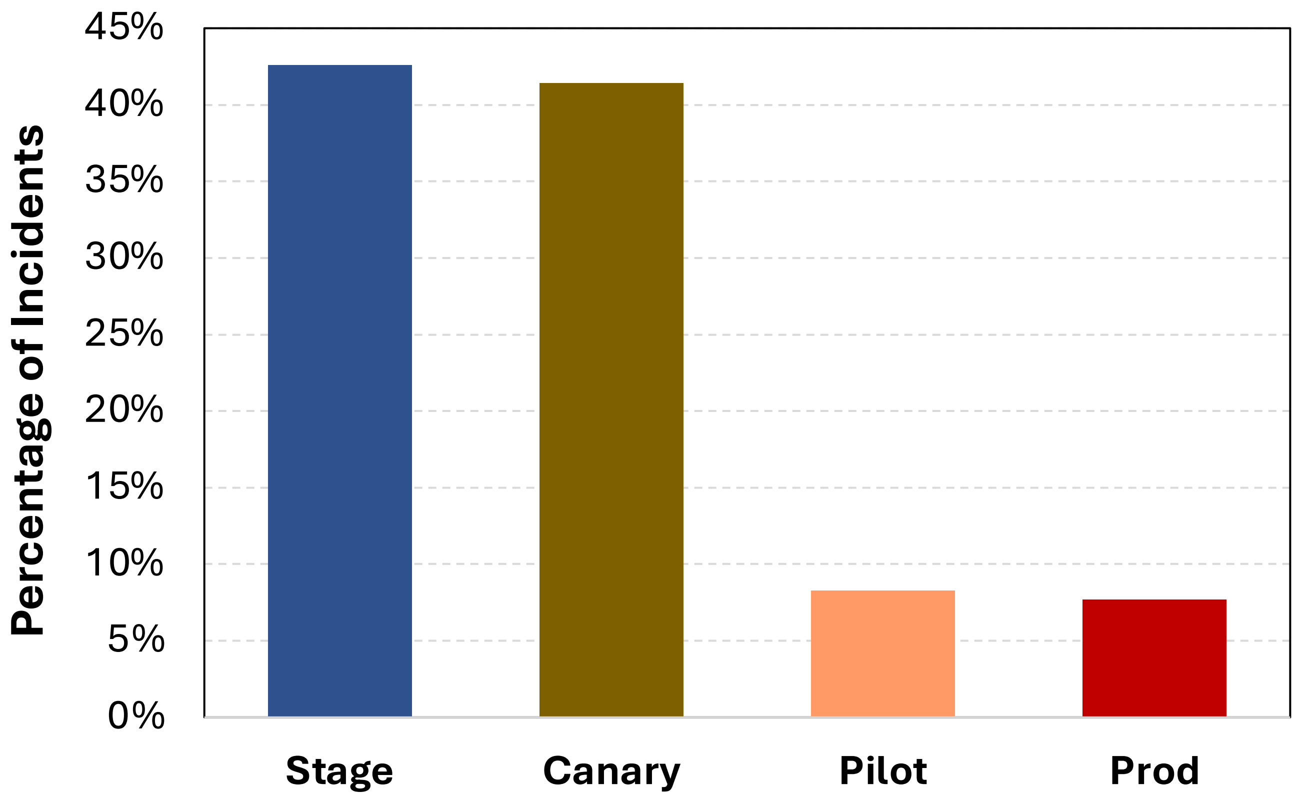
- Blamed Component
- Fault Signature

Feature extraction

Binary decision + supporting evidence

Decision process: Assess the customer impacts of the blamed components and failures

Percentage of issues detected in each environment.



Open challenges: hyperscale

Cascading effect and heterogeneity

- Complex dependencies across many layers
- Different VM types, h/w SKUs, s/w versions, workloads, etc. → different baselines of normal behavior

Right logs at right time

- Identify the right telemetry for logging
- Large volume of data across millions of VMs
- Various logging conventions in different h/w and s/w components

Preventive measure

- Prevent gray failures
- Risk management and change management
- Integrate differential observation model in testing

Noisy neighbors in shared tenant

- Identify the noisy neighbors
- Issue isolation

Conclusions

Key trait of gray failures is differential observability

No single silver bullet

Four principles

- Close the observation gap
- Approximate application view
- Leverage the power of scale
- Harness the temporal patterns

Require both system and data-driven approaches

Contact:
ryanph@umich.edu