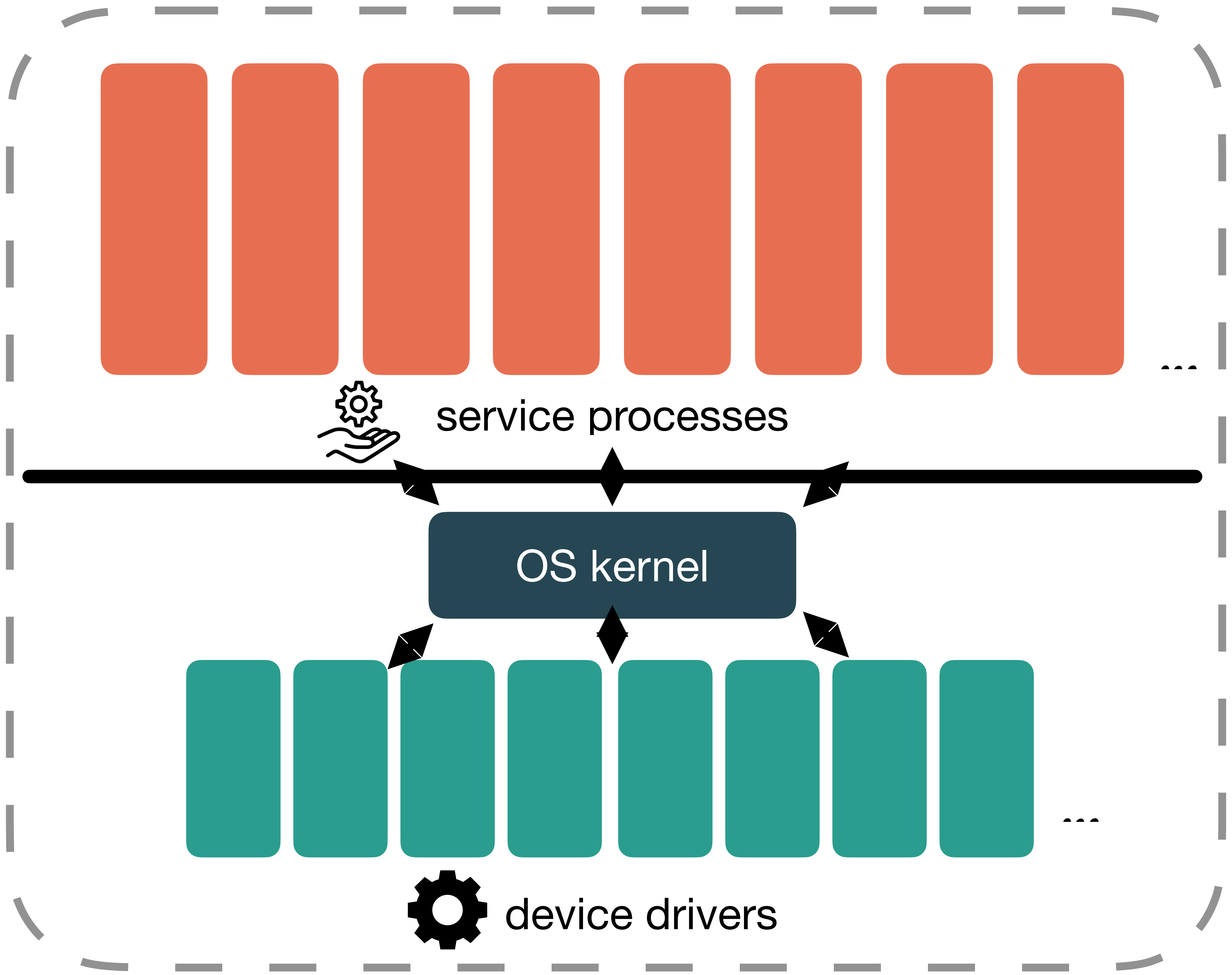
RESIN: A Holistic Service for Dealing with Memory Leaks in Production Cloud Infrastructure

Chang Lou, Cong Chen, Peng Huang, Yingnong Dang, Si Qin,
Xinsheng Yang, Xukun Li, Qingwei Lin, Murali Chintalapati

Johns Hopkins University
Microsoft Research
Microsoft Azure

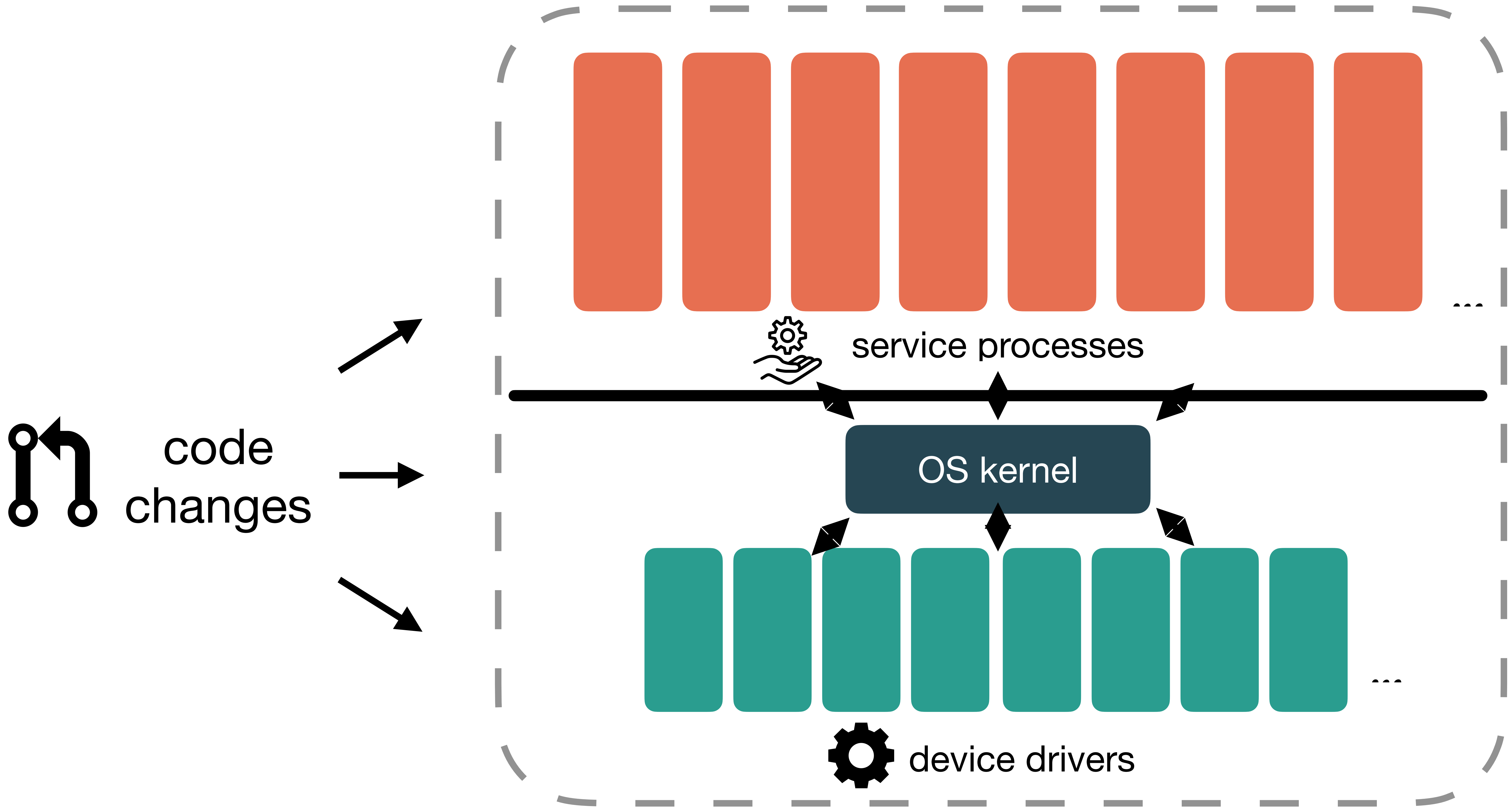
OSDI 2022

Memory leak is a notorious issue in cloud



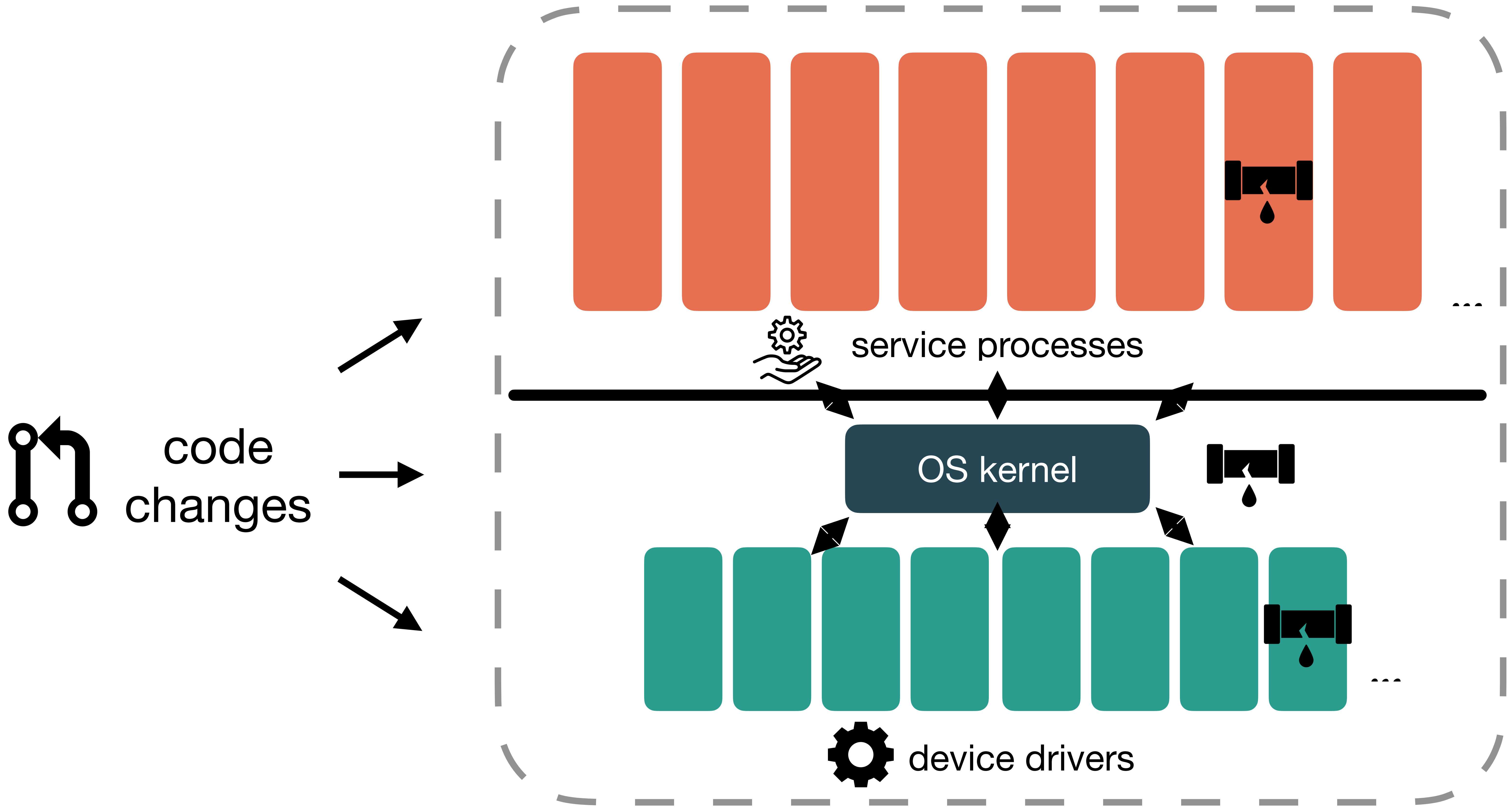
host software components

Memory leak is a notorious issue in cloud



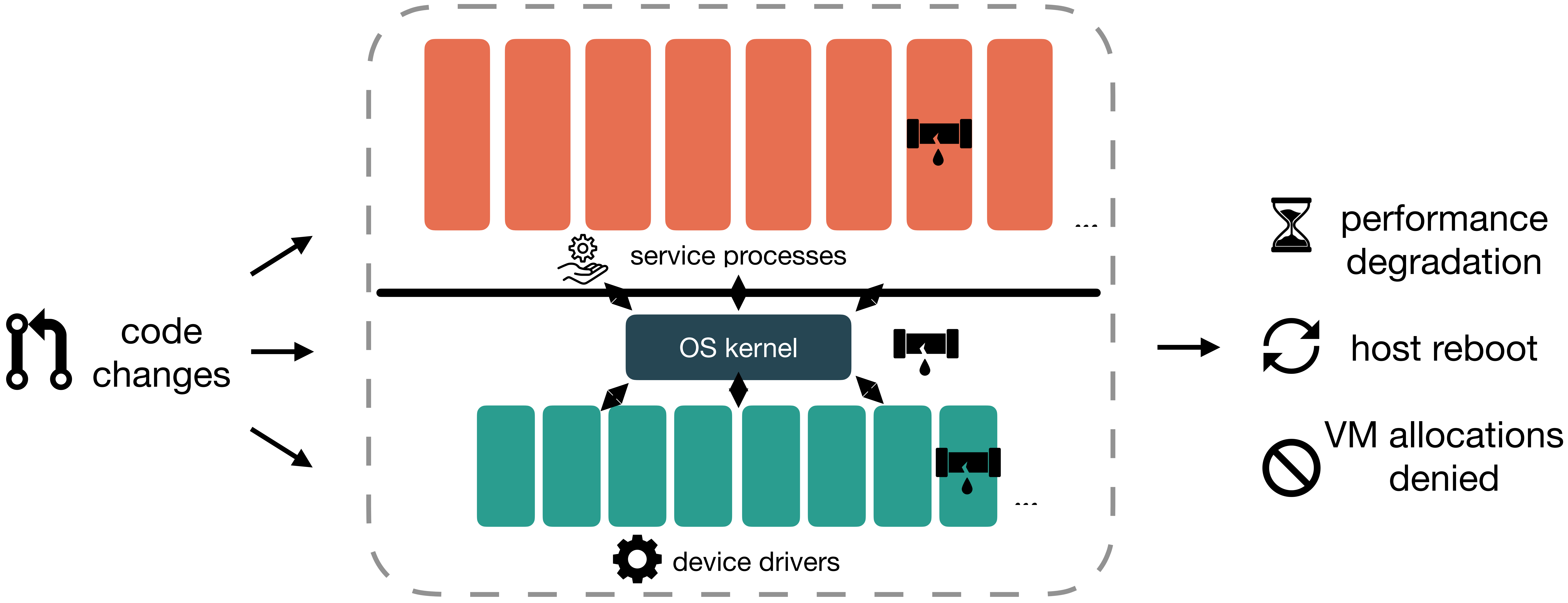
host software components

Memory leak is a notorious issue in cloud



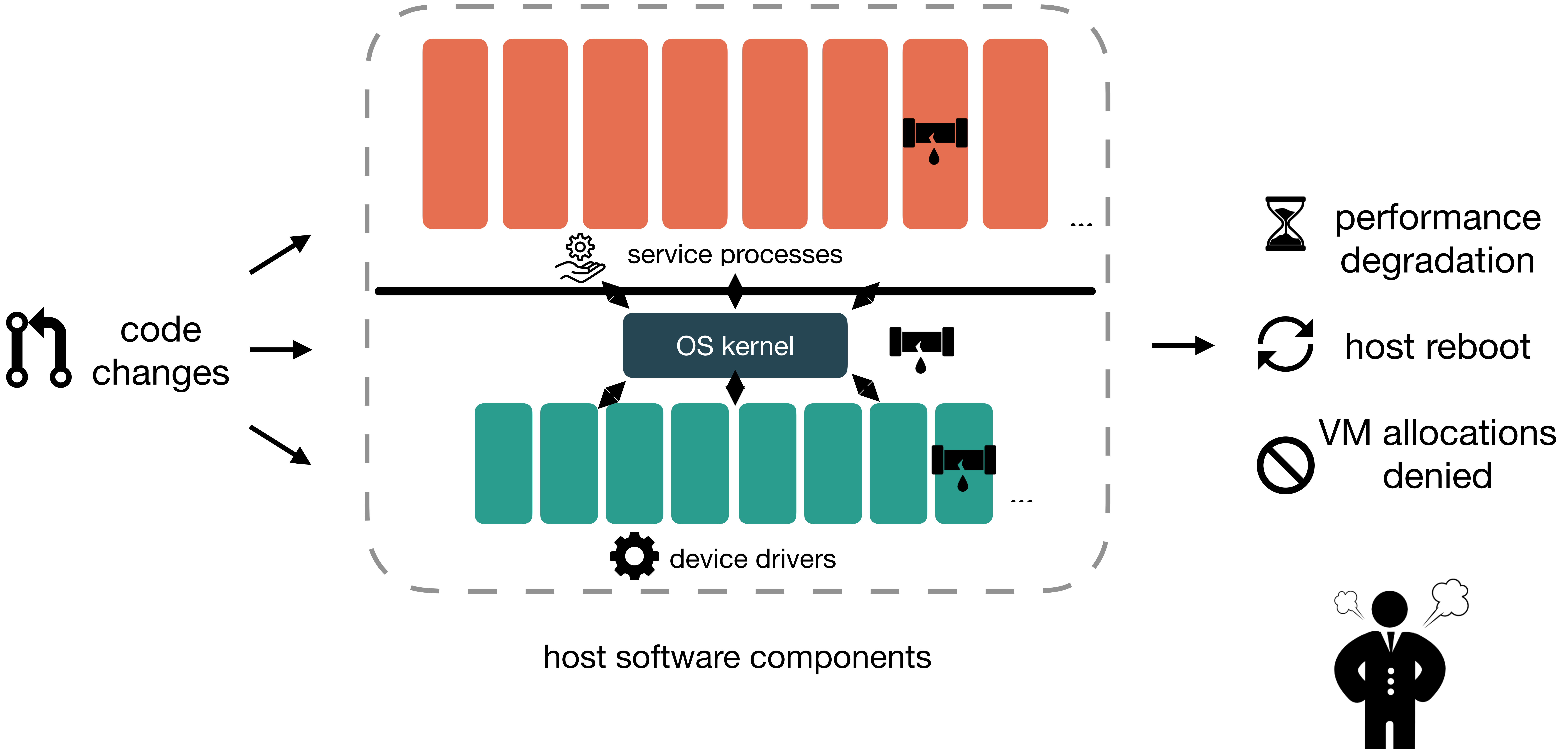
host software components

Memory leak is a notorious issue in cloud



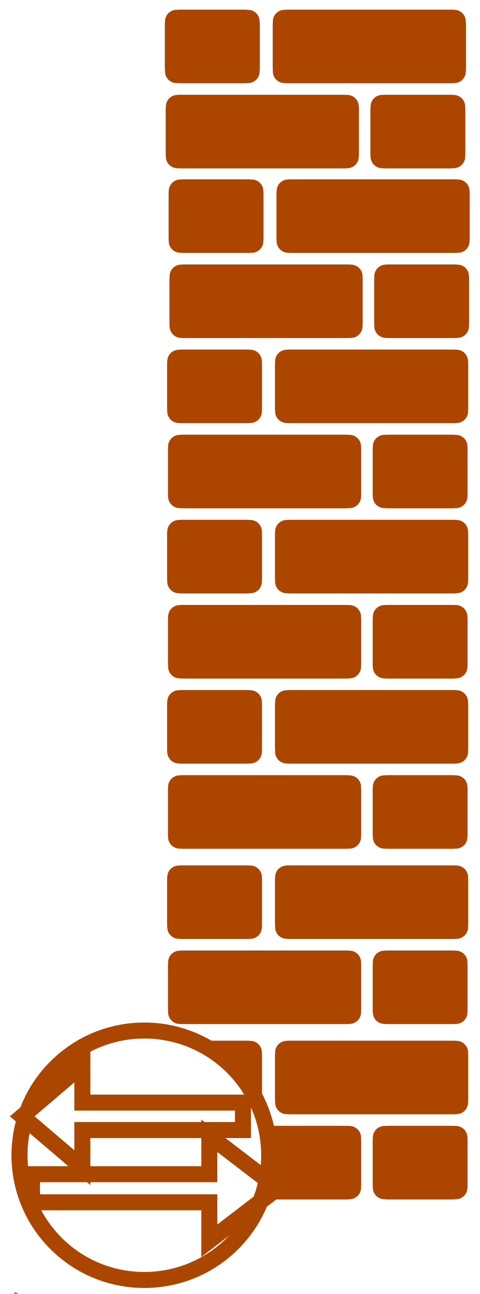
host software components

Memory leak is a notorious issue in cloud

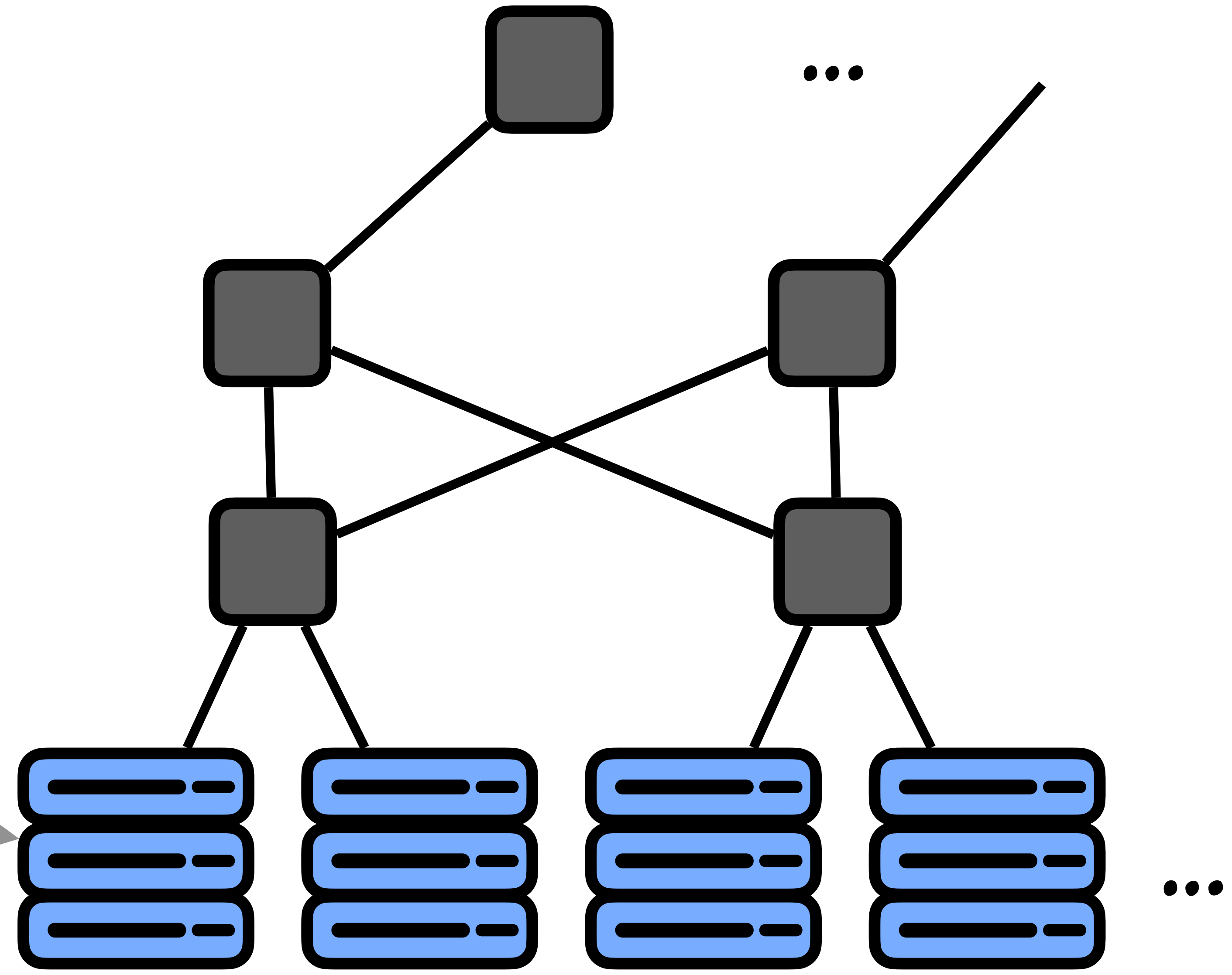


A production memory leak in cloud

ID	PERM	PROT	SRC	DEST
1	✓	TCP	ANY	192.168.1.21
2	✓	TCP	ANY	192.168.1.22
3	✓	UDP	ANY	192.168.1.23
4	✗	ANY	ANY	192.168.1.27

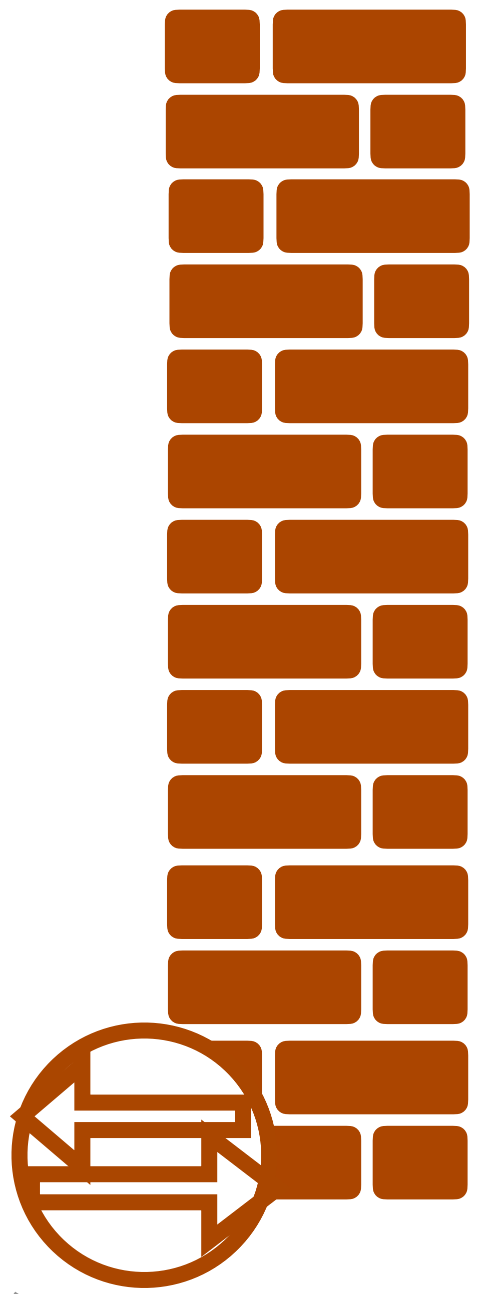


Firewall service



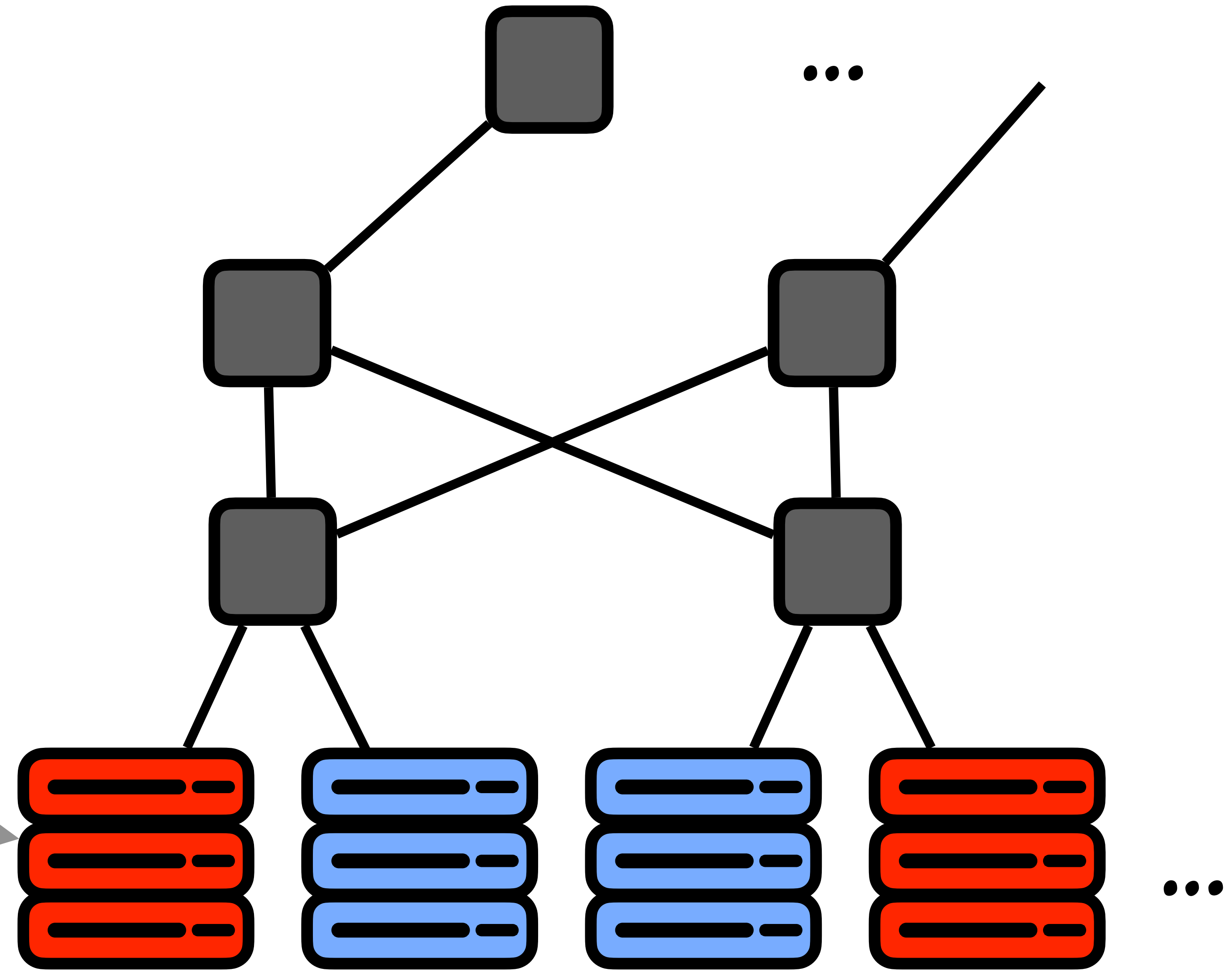
Host machines in Azure clusters

A production memory leak in cloud



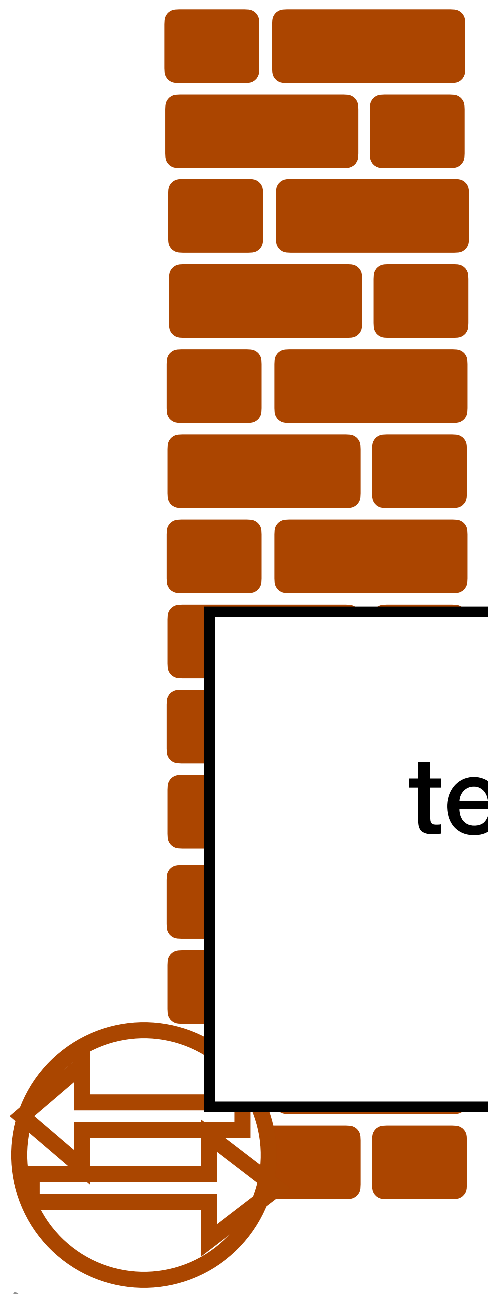
ID	PERM	PROT	SRC	DEST
1	✓	TCP	ANY	192.168.1.21
2	✓	TCP	ANY	192.168.1.22
3	✓	UDP	ANY	192.168.1.23
4	✘	ANY	ANY	192.168.1.27
5	✘	ANY	ANY	192.168.1.44
6	✘	ANY	ANY	192.168.1.44
7	✘	ANY	ANY	192.168.1.44
8	✘	ANY	ANY	192.168.1.44
9	✘	ANY	ANY	192.168.1.44
10	✘	ANY	ANY	192.168.1.44
11	✘	ANY	ANY	192.168.1.44
12	✘	ANY	ANY	192.168.1.44
			⋯	rule object leaked!
..	✘	ANY	ANY	192.168.1.44

Firewall service



Host machines in Azure clusters

A production memory leak in cloud

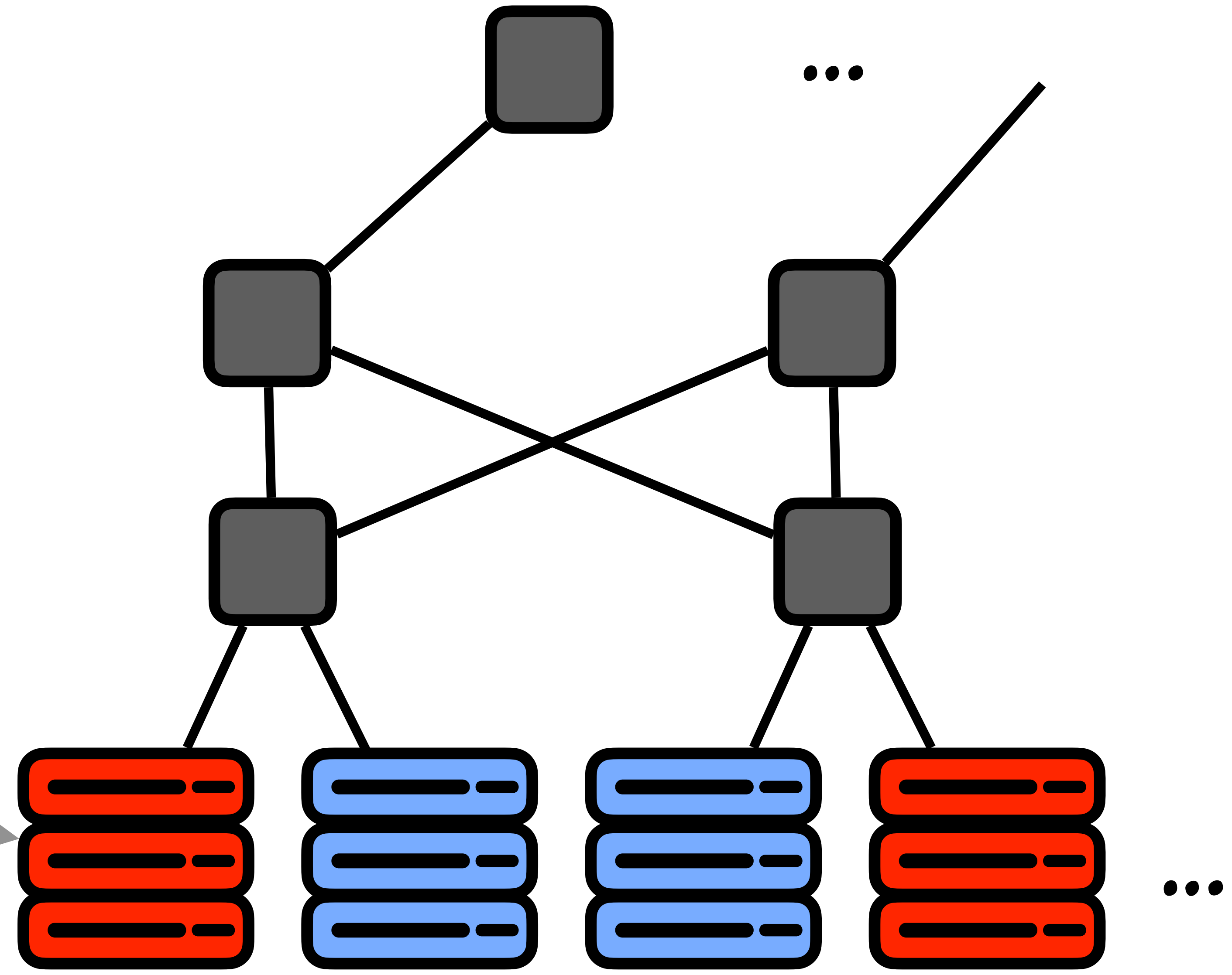


ID	PERM	PROT	SRC	DEST
1	✓	TCP	ANY	192.168.1.21
2	✓	TCP	ANY	192.168.1.22
3	✓	UDP	ANY	192.168.1.23
4	✗	ANY	ANY	192.168.1.27
5	✗	ANY	ANY	192.168.1.44
6	✗	ANY	ANY	192.168.1.44
7	✗	ANY	ANY	192.168.1.44
8	✗	ANY	ANY	192.168.1.44
..	✗	ANY	ANY	192.168.1.44

testing and static checker
report no bug ✓

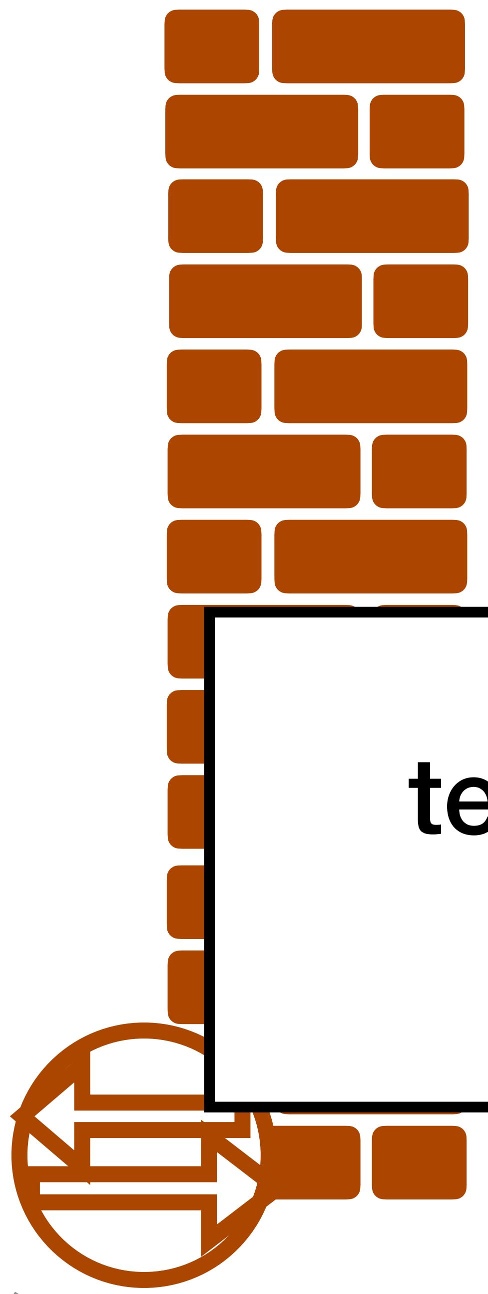
ed!

Firewall service



Host machines in Azure clusters

A production memory leak in cloud

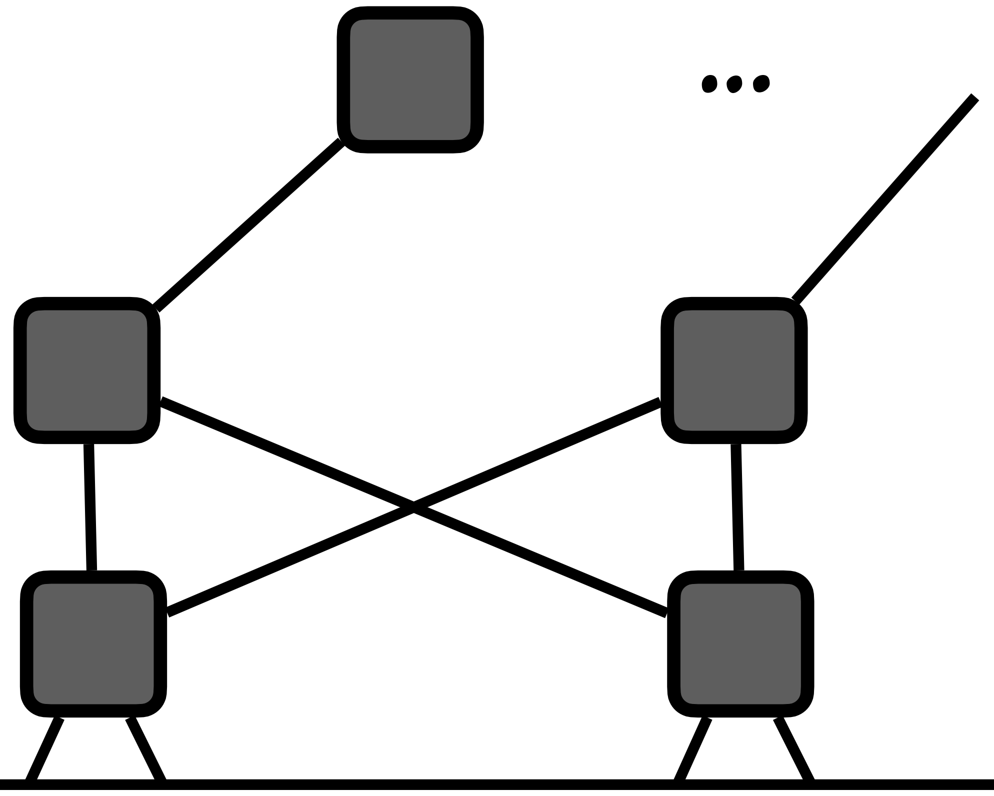


ID	PERM	PROT	SRC	DEST
1	✓	TCP	ANY	192.168.1.21
2	✓	TCP	ANY	192.168.1.22
3	✓	UDP	ANY	192.168.1.23
4	✗	ANY	ANY	192.168.1.27
5	✗	ANY	ANY	192.168.1.44
6	✗	ANY	ANY	192.168.1.44
7	✗	ANY	ANY	192.168.1.44
8	✗	ANY	ANY	192.168.1.44
..	✗	ANY	ANY	192.168.1.44

testing and static checker report no bug ✓

ed!

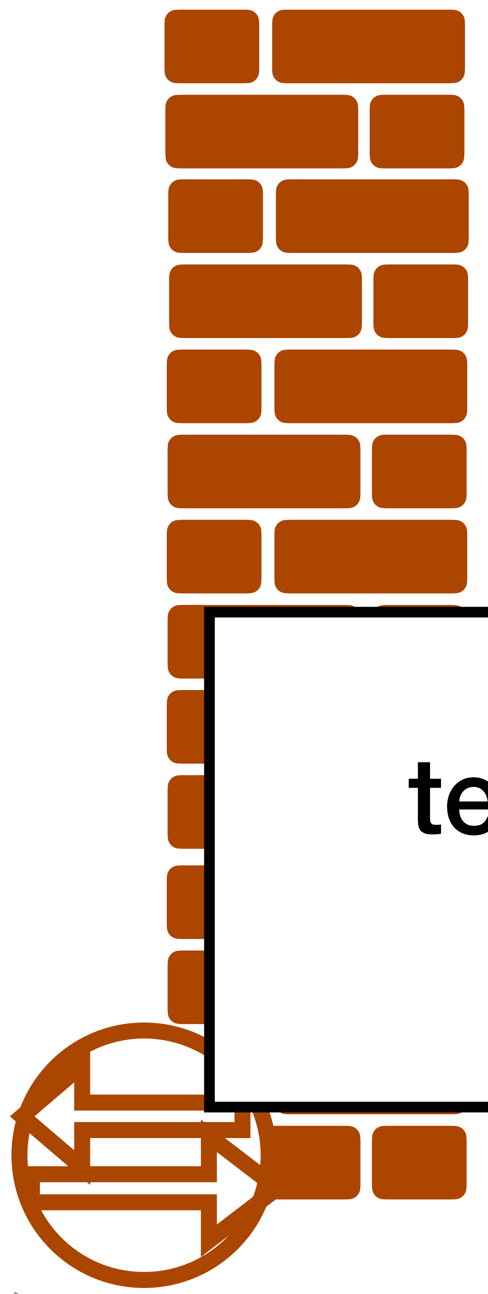
Firewall service



only leak on some hosts

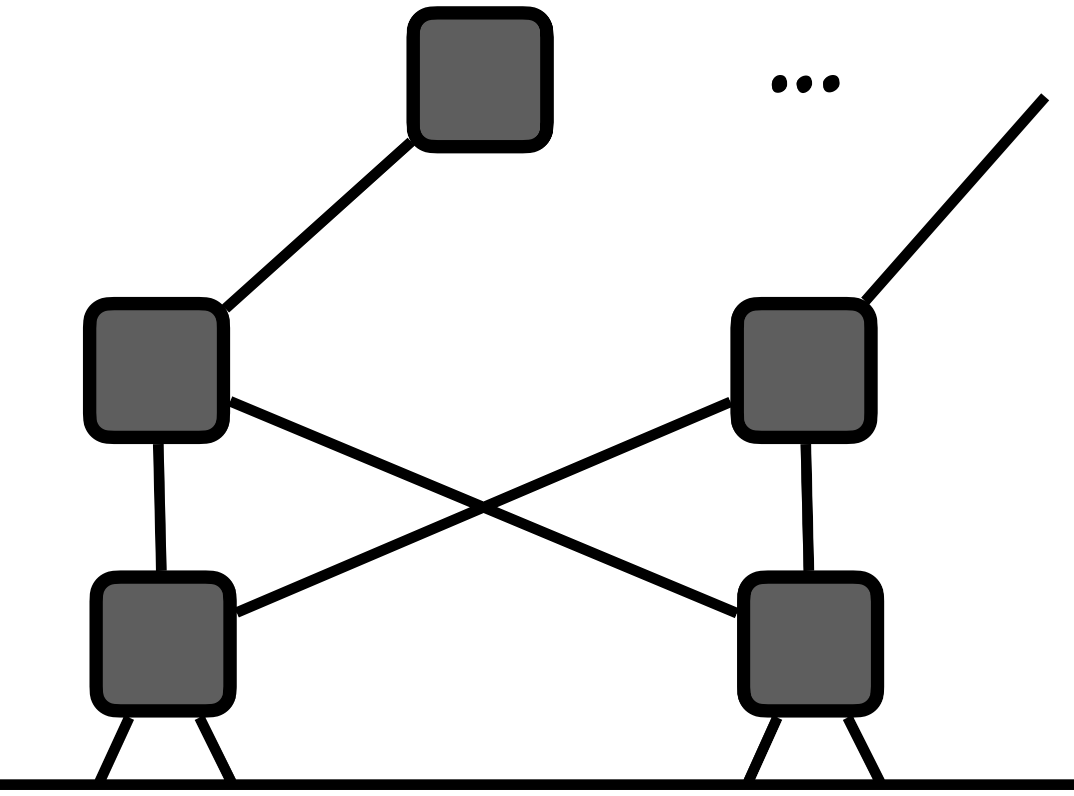
Host machines in Azure clusters

A production memory leak in cloud



ID	PERM	PROT	SRC	DEST
1	✓	TCP	ANY	192.168.1.21
2	✓	TCP	ANY	192.168.1.22
3	✓	UDP	ANY	192.168.1.23
4	✗	ANY	ANY	192.168.1.27
5	✗	ANY	ANY	192.168.1.44
6	✗	ANY	ANY	192.168.1.44
7	✗	ANY	ANY	192.168.1.44
8	✗	ANY	ANY	192.168.1.44
..	✗	ANY	ANY	192.168.1.44

testing and static checker report no bug ✓



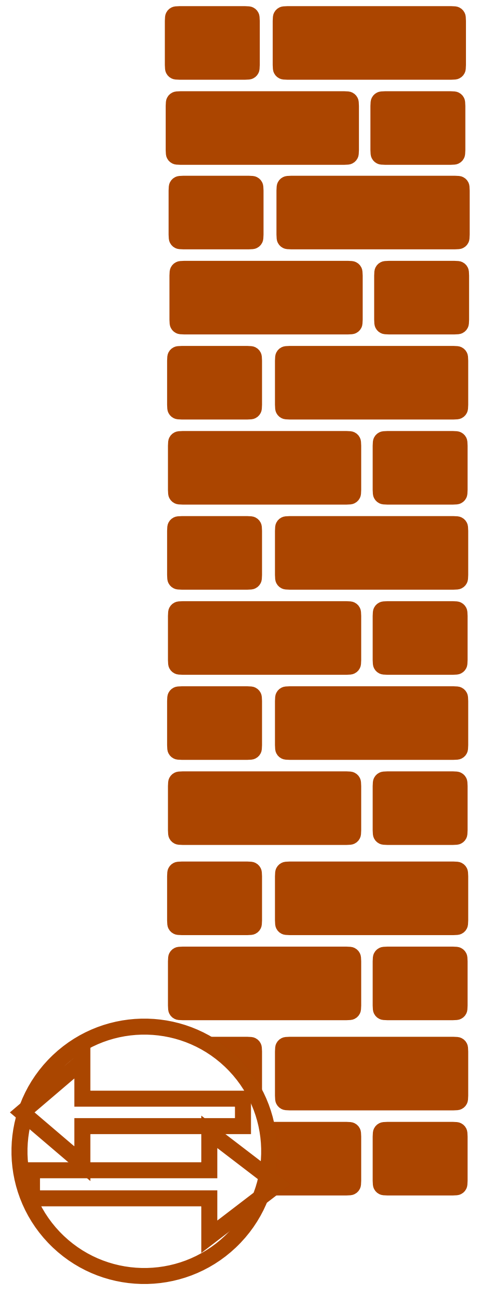
only leak on some hosts

Firewall service

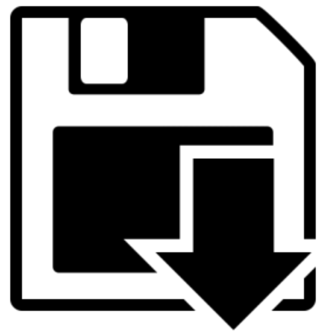
Host machines in Azure clusters

The leak is cross-component

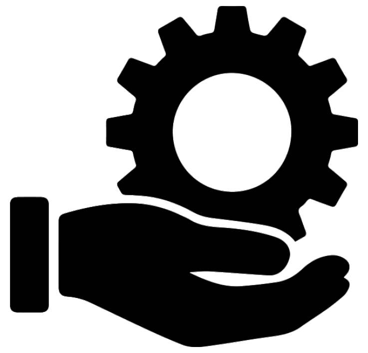
ID	PERM	PROT	SRC	DEST
1	✓	TCP	ANY	192.168.1.21
2	✓	TCP	ANY	192.168.1.22
3	✓	UDP	ANY	192.168.1.23
4	✘	ANY	ANY	192.168.1.27



Firewall service



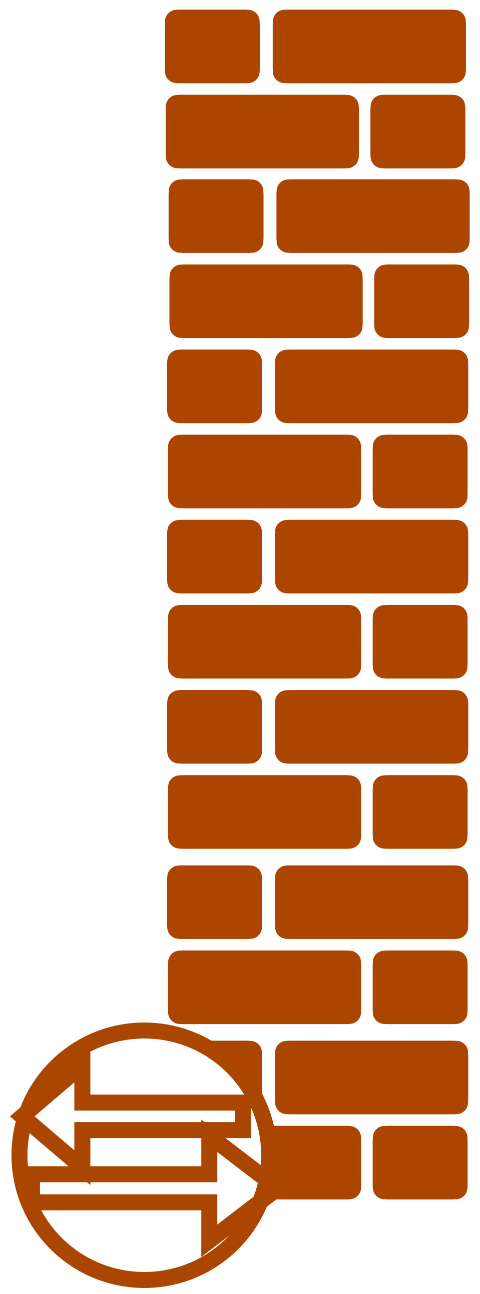
load configured rules



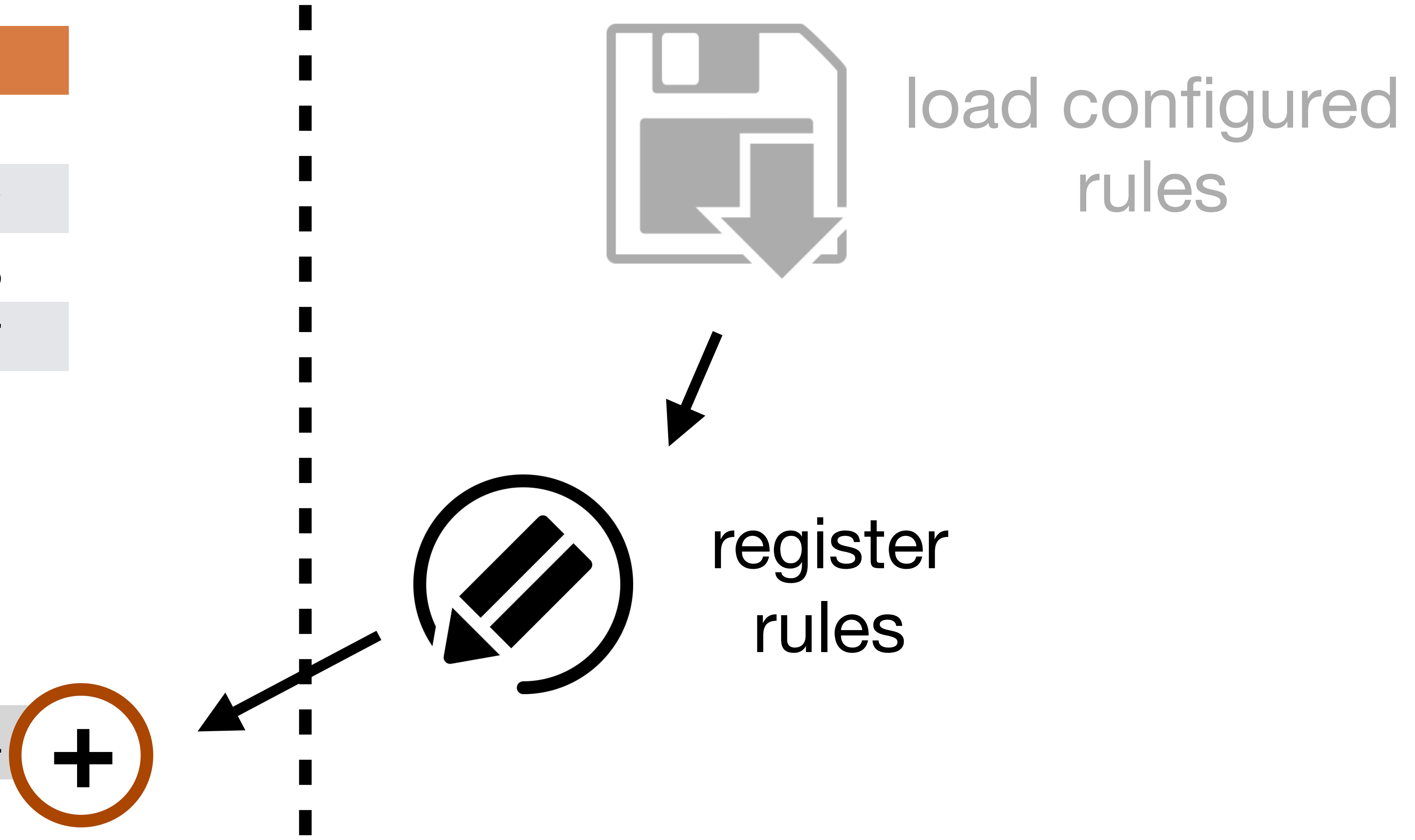
Config agent

The leak is cross-component

ID	PERM	PROT	SRC	DEST
1	✓	TCP	ANY	192.168.1.21
2	✓	TCP	ANY	192.168.1.22
3	✓	UDP	ANY	192.168.1.23
4	✗	ANY	ANY	192.168.1.27
5	✗	ANY	ANY	192.168.1.44

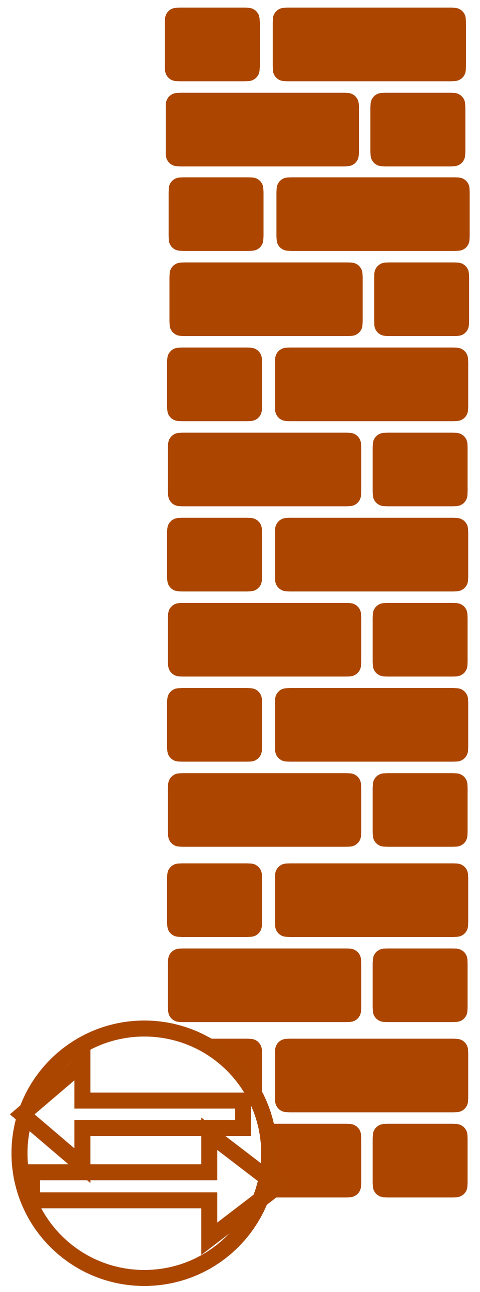


Firewall service

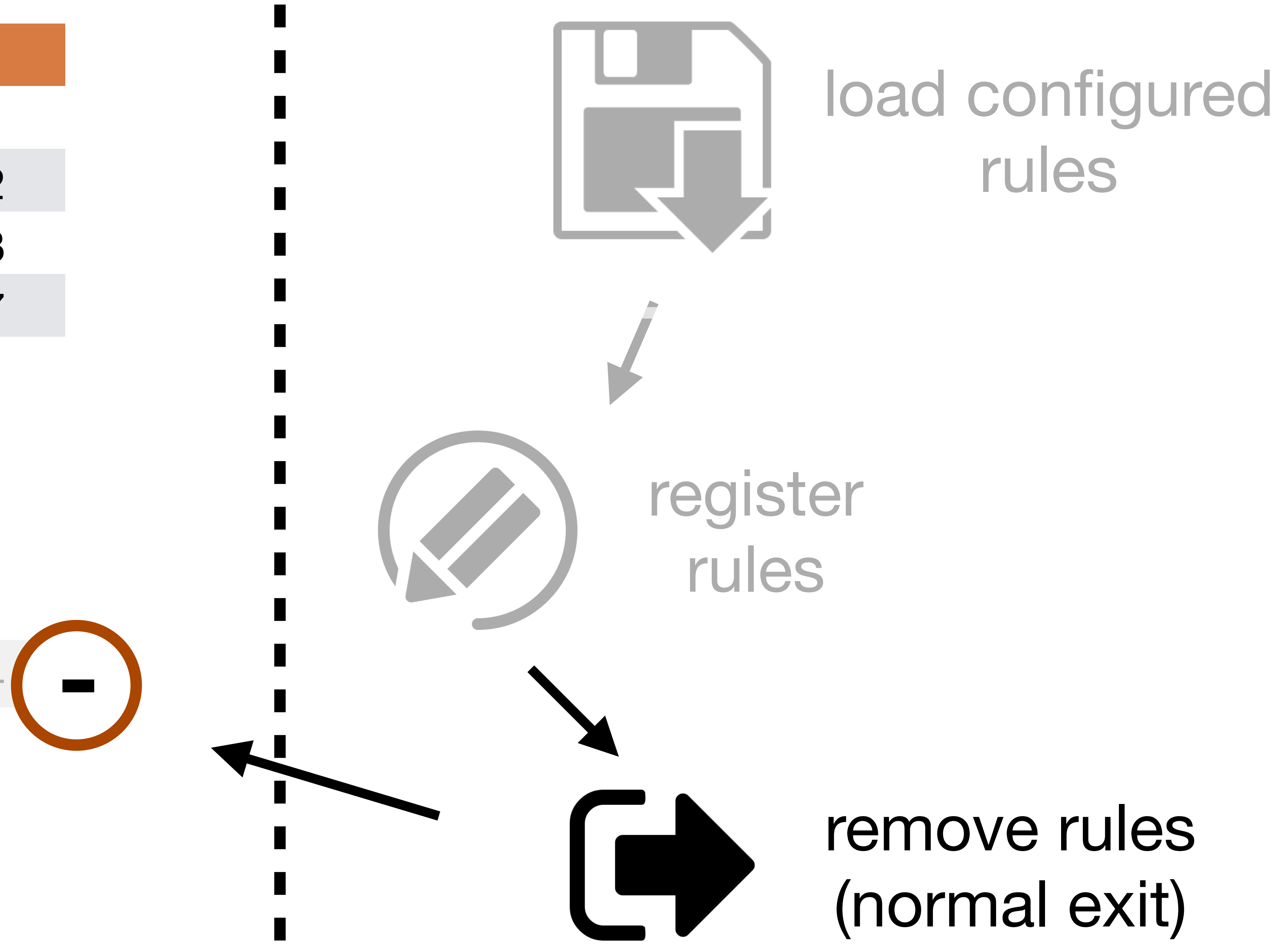


Config agent

The leak is cross-component



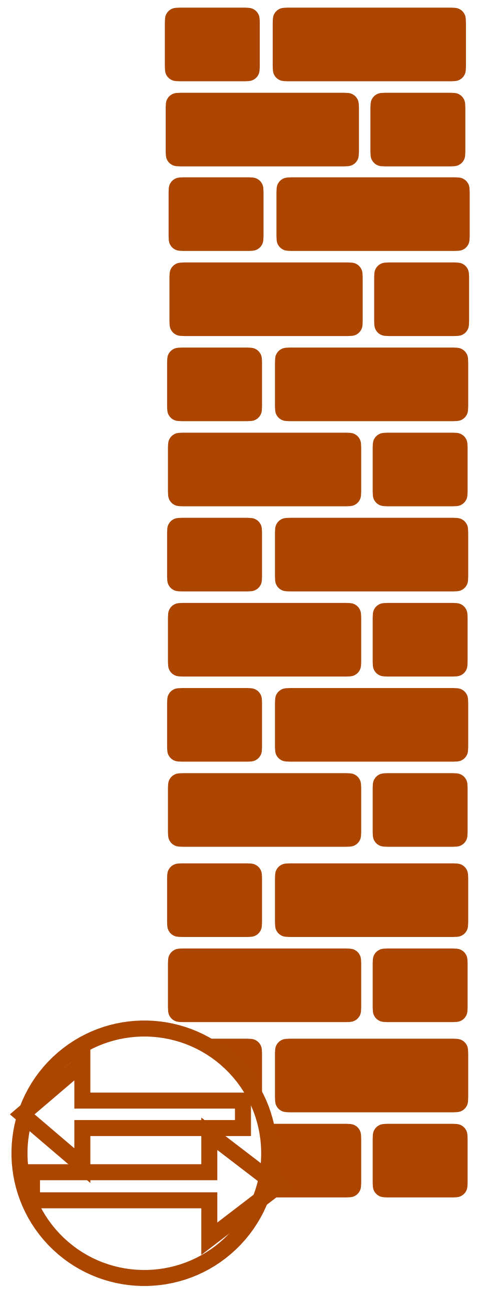
ID	PERM	PROT	SRC	DEST
1	✓	TCP	ANY	192.168.1.21
2	✓	TCP	ANY	192.168.1.22
3	✓	UDP	ANY	192.168.1.23
4	✗	ANY	ANY	192.168.1.27
5	✗	ANY	ANY	192.168.1.44



Firewall service

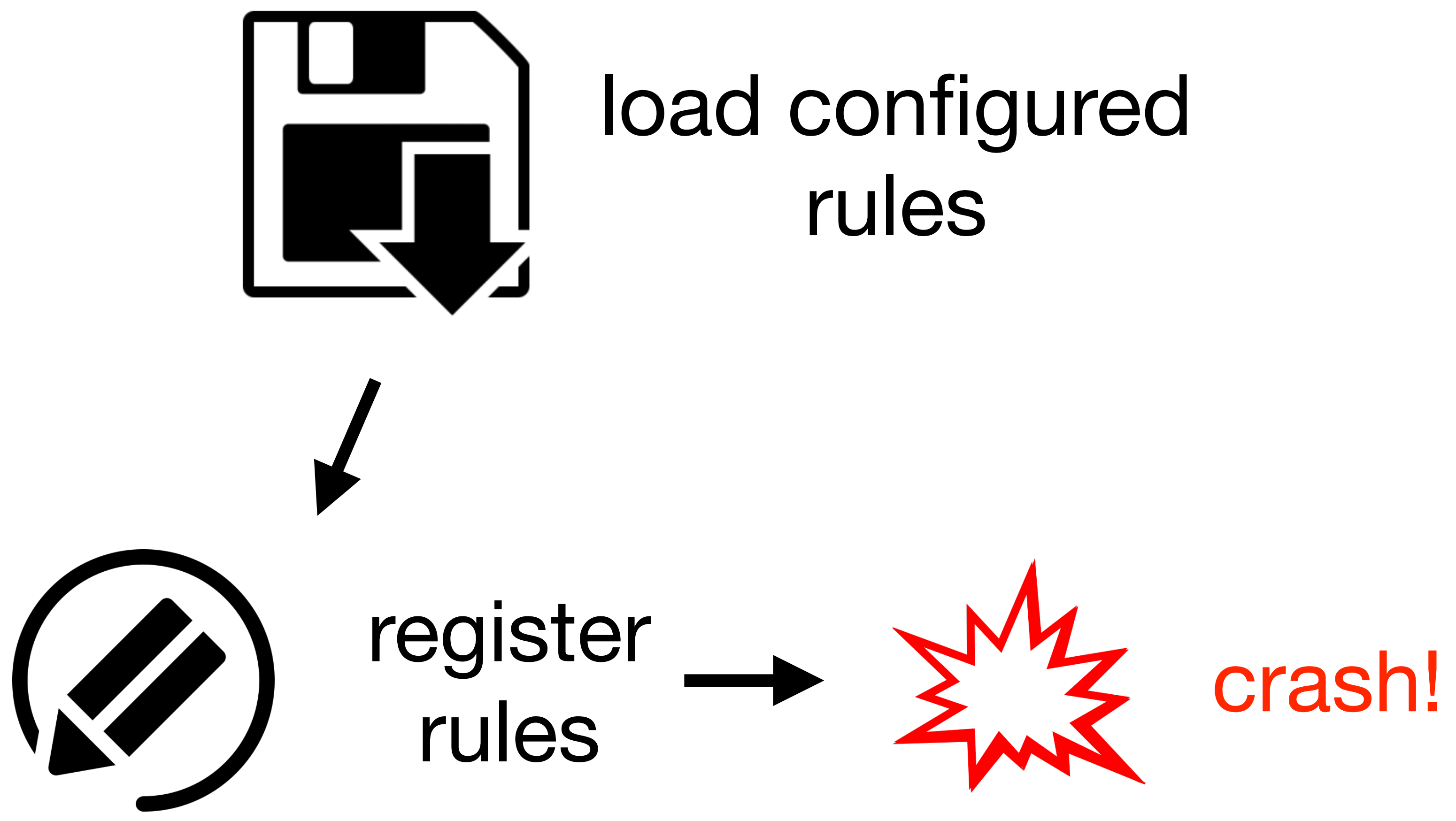
Config agent

The leak is cross-component



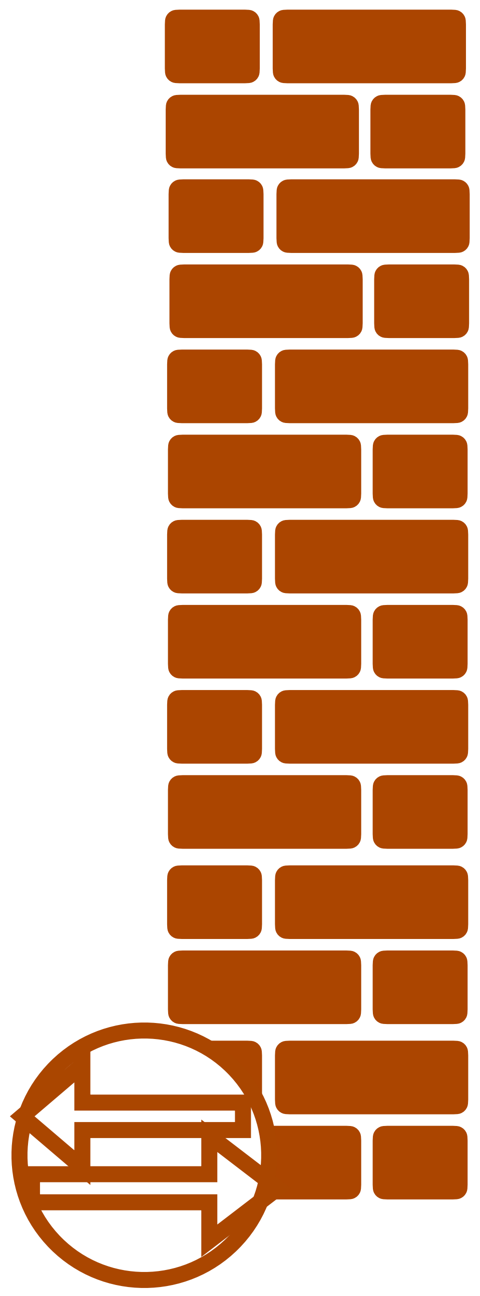
ID	PERM	PROT	SRC	DEST
1	✓	TCP	ANY	192.168.1.21
2	✓	TCP	ANY	192.168.1.22
3	✓	UDP	ANY	192.168.1.23
4	✗	ANY	ANY	192.168.1.27
5	✗	ANY	ANY	192.168.1.44

Firewall service



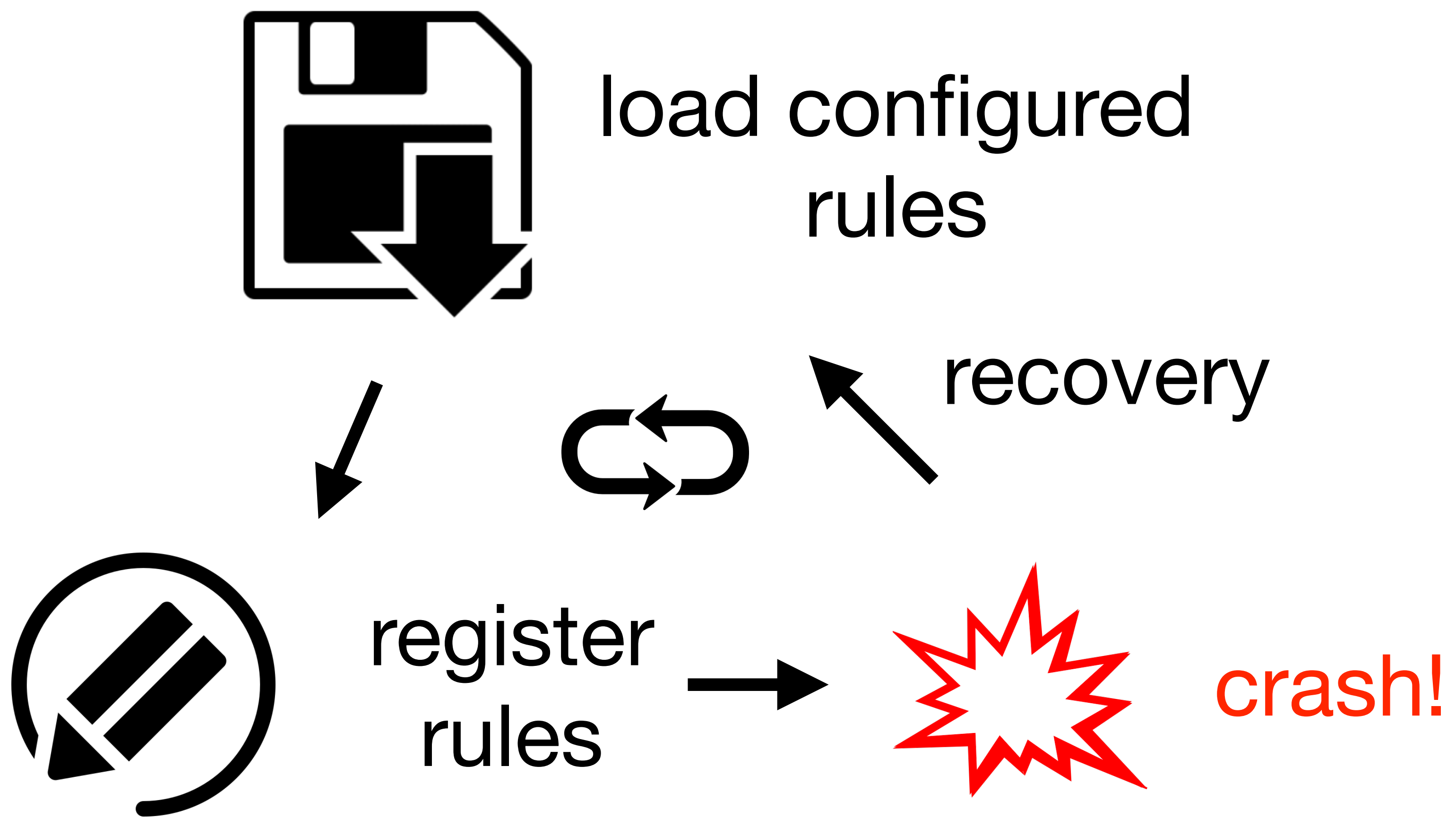
Config agent

The leak is cross-component



ID	PERM	PROT	SRC	DEST
1	✓	TCP	ANY	192.168.1.21
2	✓	TCP	ANY	192.168.1.22
3	✓	UDP	ANY	192.168.1.23
4	✗	ANY	ANY	192.168.1.27
5	✗	ANY	ANY	192.168.1.44
6	✗	ANY	ANY	192.168.1.44
7	✗	ANY	ANY	192.168.1.44
8	✗	ANY	ANY	192.168.1.44
9	✗	ANY	ANY	192.168.1.44
10	✗	ANY	ANY	192.168.1.44
11	✗	ANY	ANY	192.168.1.44
12	✗	ANY	ANY	192.168.1.44
			…	7,092,866 more rows
..	✗	ANY	ANY	192.168.1.44

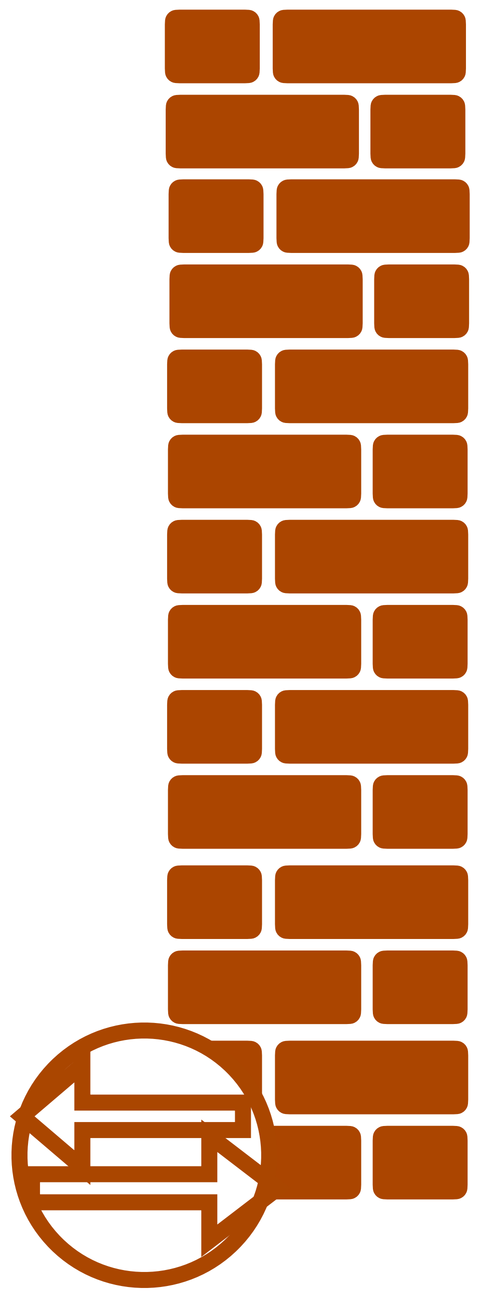
~80 bytes



Firewall service

Config agent

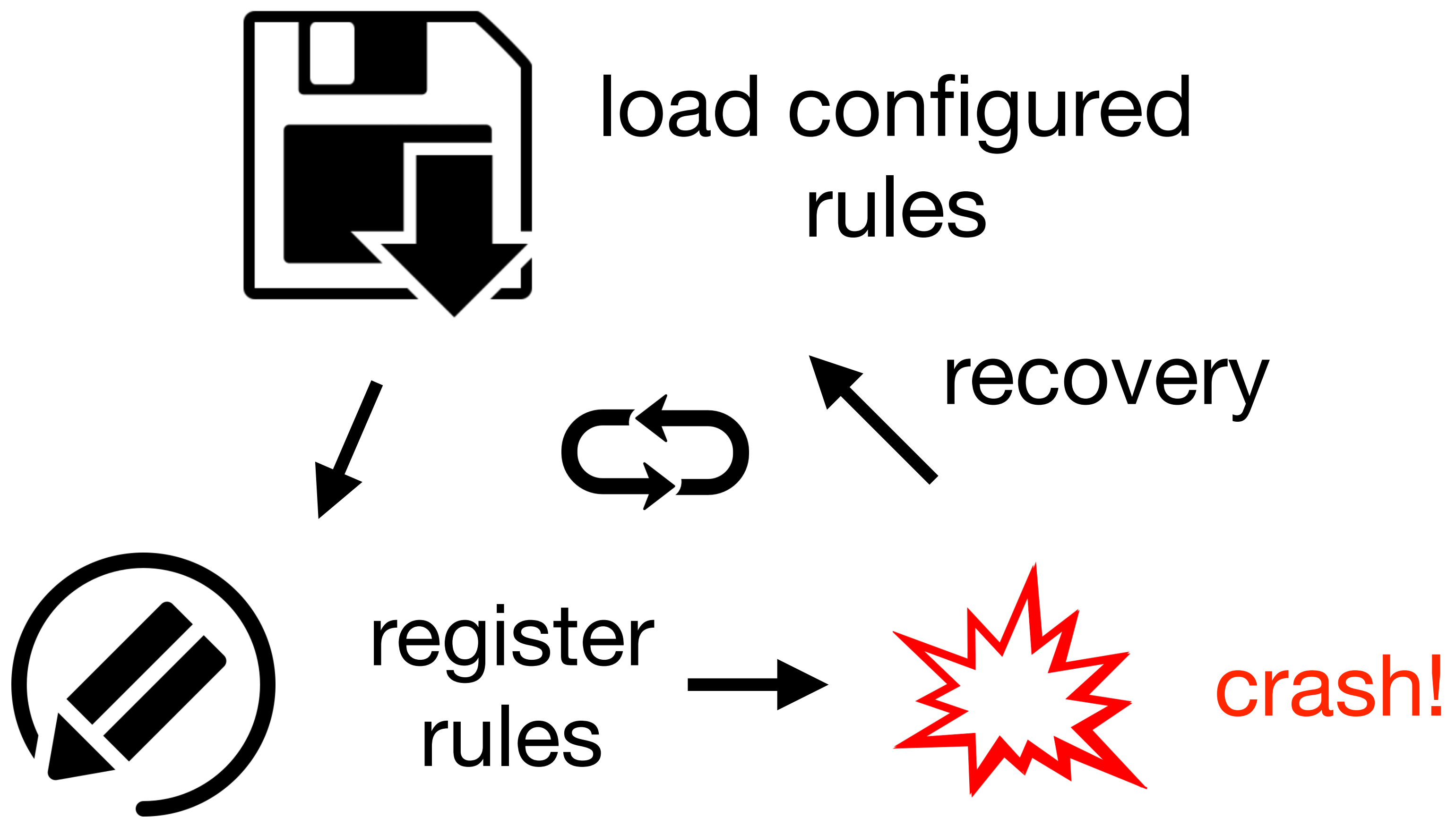
The leak is configuration-triggered



ID	PERM	PROT	SRC	DEST
1	✓	TCP	ANY	192.168.1.21
2	✓	TCP	ANY	192.168.1.22
3	✓	UDP	ANY	192.168.1.23
4	✗	ANY	ANY	192.168.1.27
5	✗	ANY	ANY	192.168.1.44
6	✗	ANY	ANY	192.168.1.44
7	✗	ANY	ANY	192.168.1.44
8	✗	ANY	ANY	192.168.1.44
9	✗	ANY	ANY	192.168.1.44
10	✗	ANY	ANY	192.168.1.44
11	✗	ANY	ANY	192.168.1.44
12	✗	ANY	ANY	192.168.1.44
			…	7,092,866 more rows
..	✗	ANY	ANY	192.168.1.44

~80 bytes

Firewall service

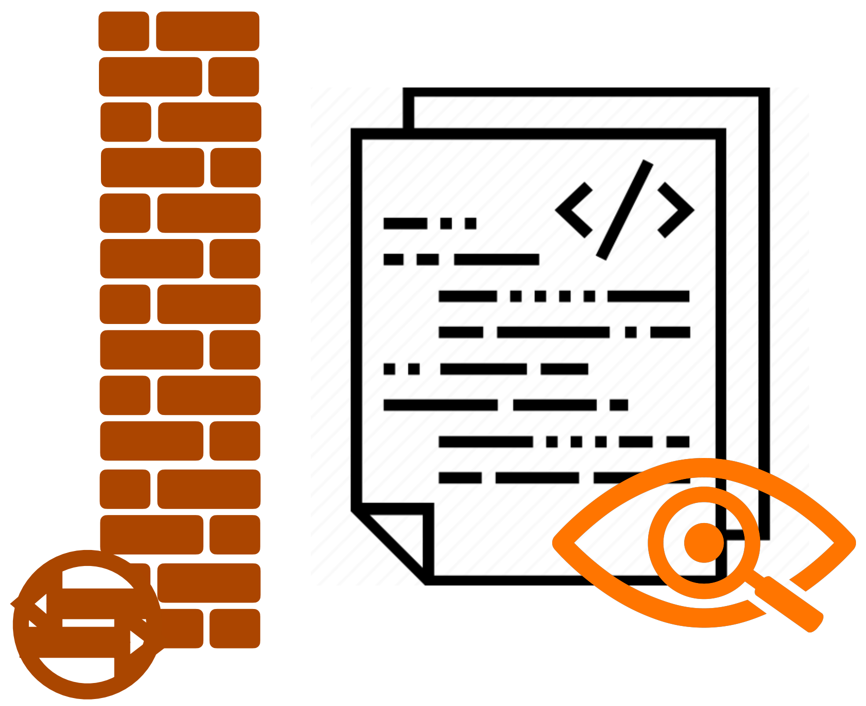


(bug triggered with enabled feature in config only)

Config agent

Why detection is still a problem in cloud?

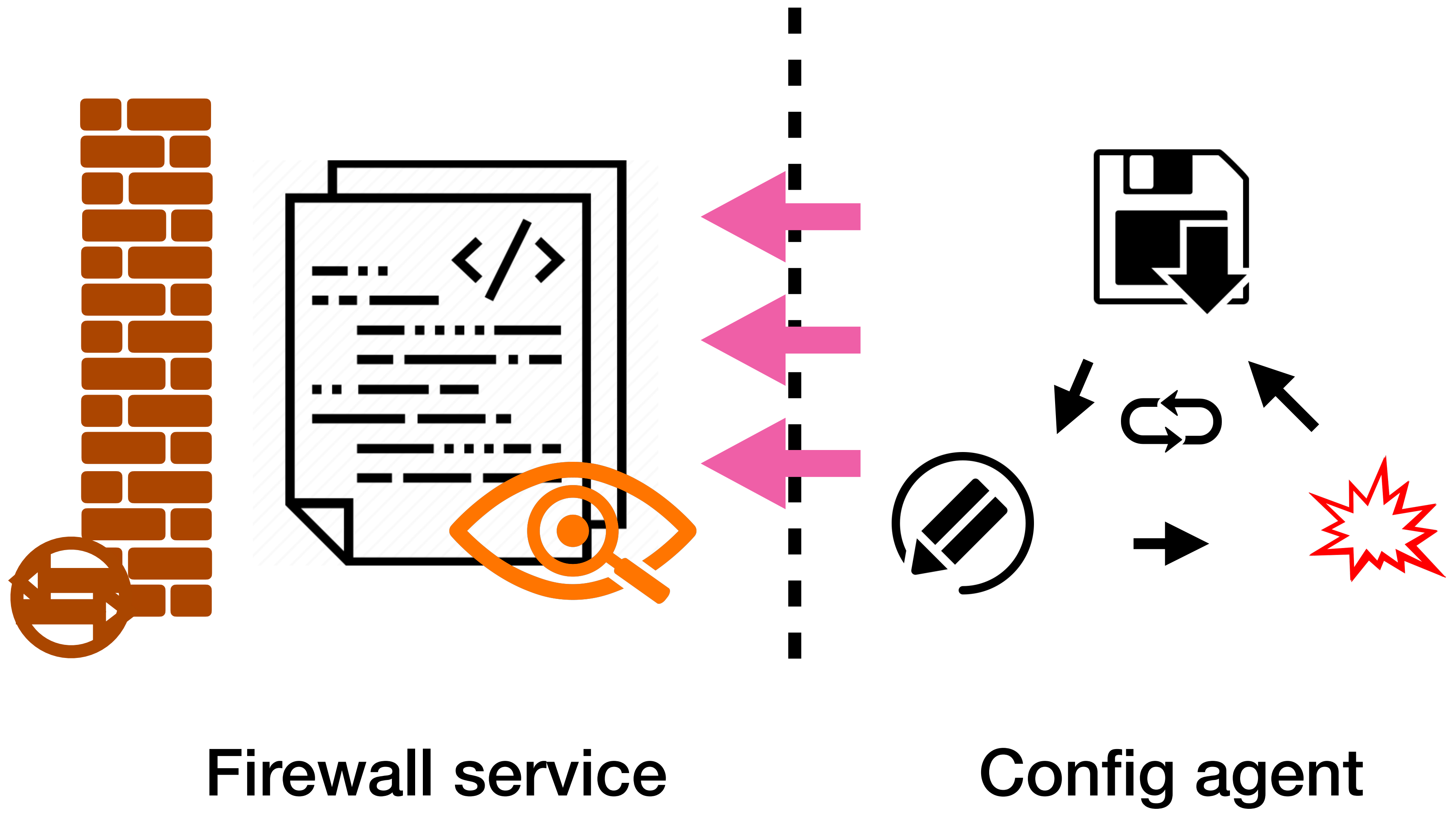
Why detection is still a problem in cloud?



Firewall service

- Practice 1: static approach
  - run static analysis on source codes
  - expose bugs without running programs

Why detection is still a problem in cloud?

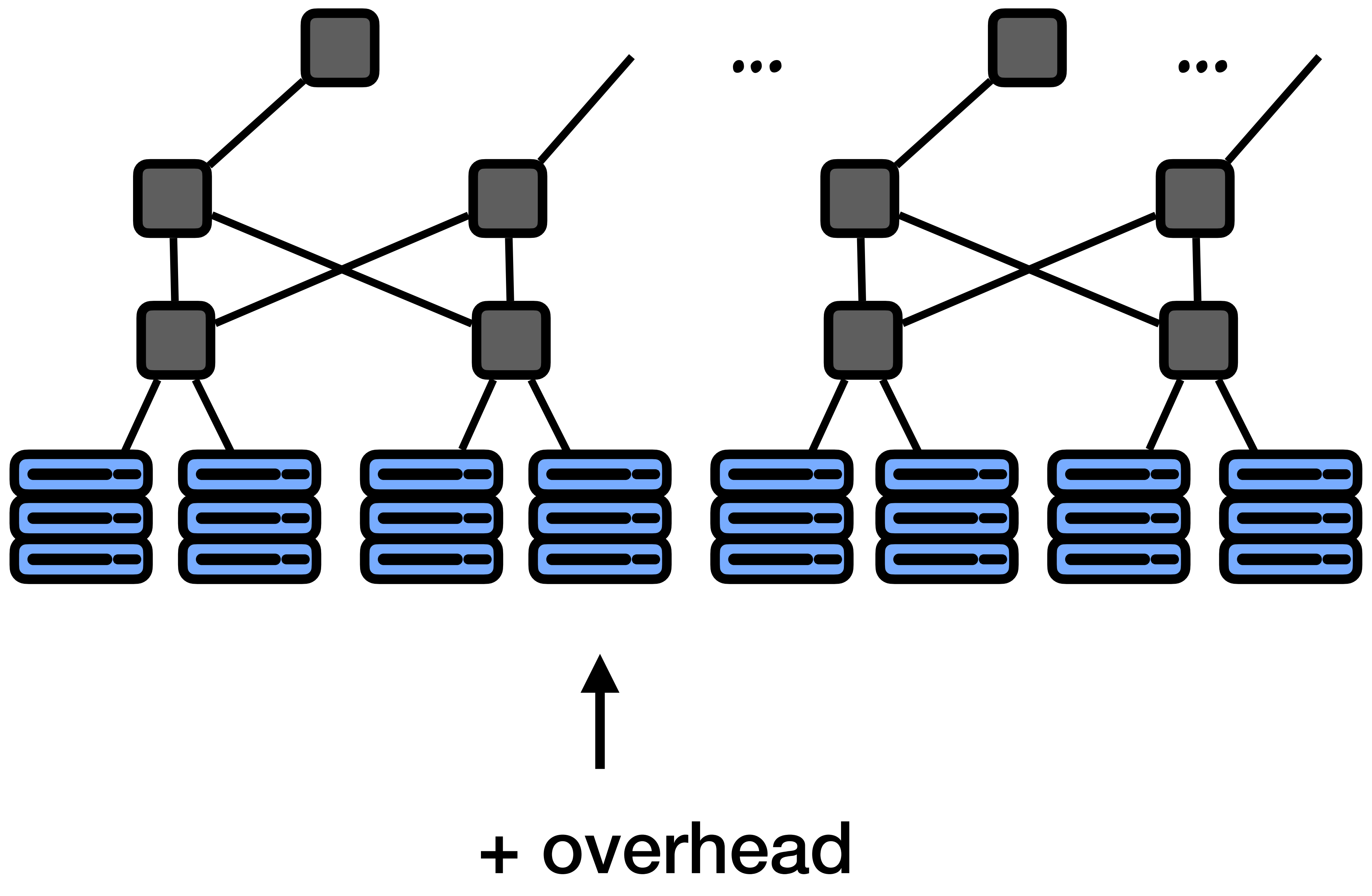


- Practice 1: static approach
  - run static analysis on source codes
  - expose bugs without running programs
- Limitations
  - no overhead, but not scalable or accurate

Why detection is still a problem in cloud?

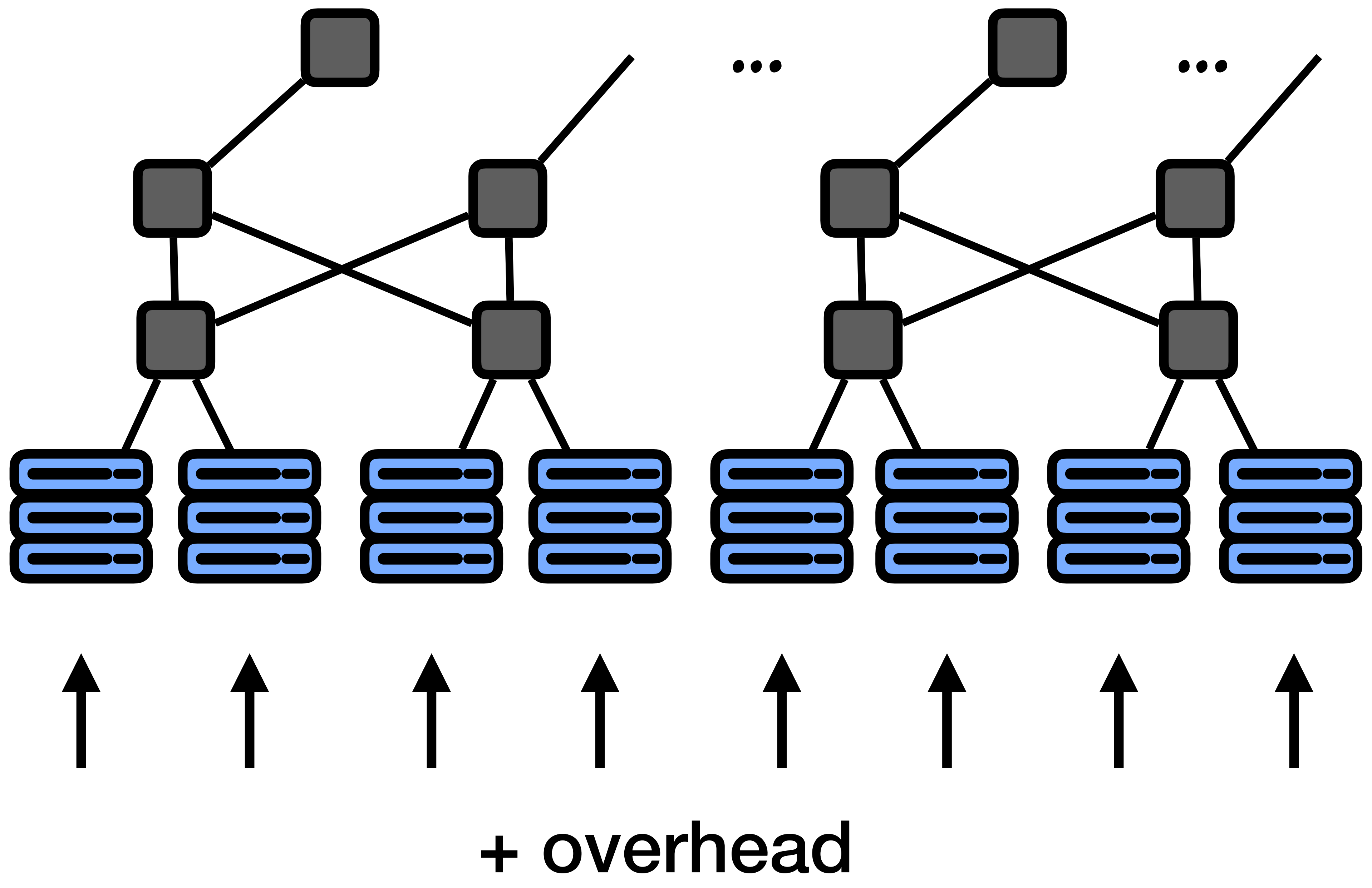
- Practice 2: dynamic approach
  - instrument programs and track the object lifetime at runtime to find leaked objects
  - detect leaks and pinpoint leaked objects

Why detection is still a problem in cloud?



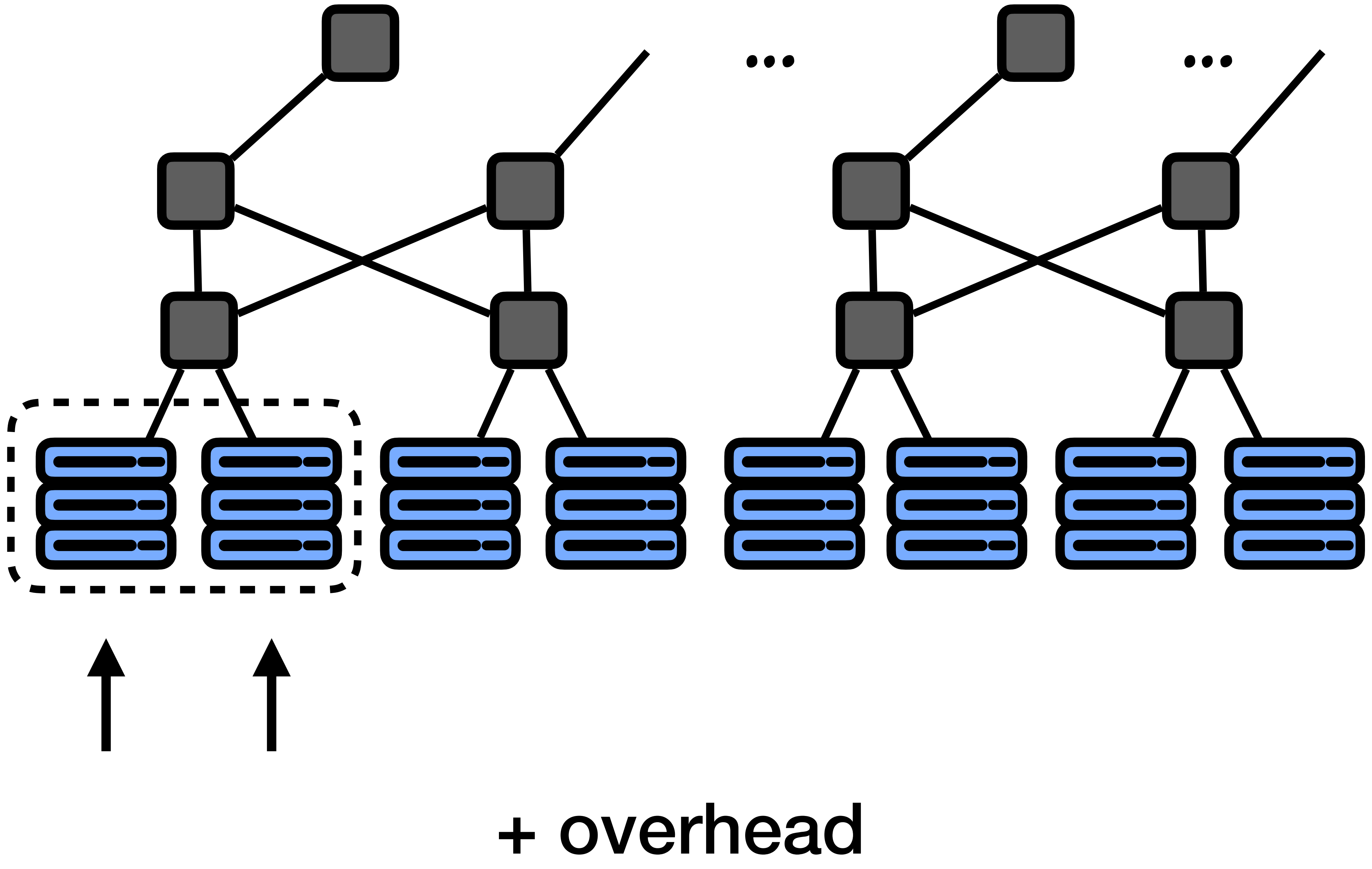
- Practice 2: dynamic approach
  - instrument programs and track the object lifetime at runtime to find leaked objects
  - detect leaks and pinpoint leaked objects

Why detection is still a problem in cloud?



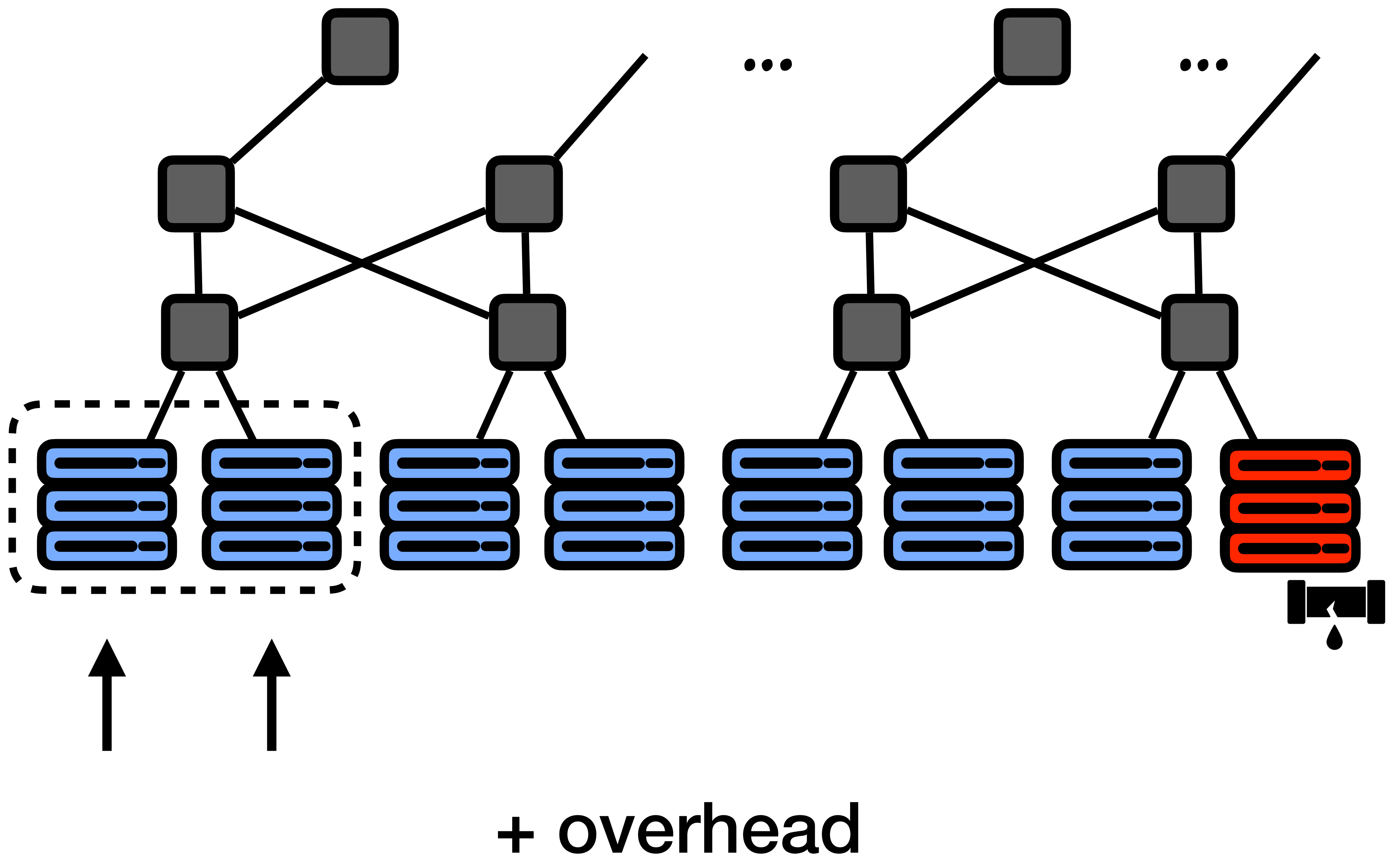
- Practice 2: dynamic approach
  - instrument programs and track the object lifetime at runtime to find leaked objects
  - detect leaks and pinpoint leaked objects

Why detection is still a problem in cloud?



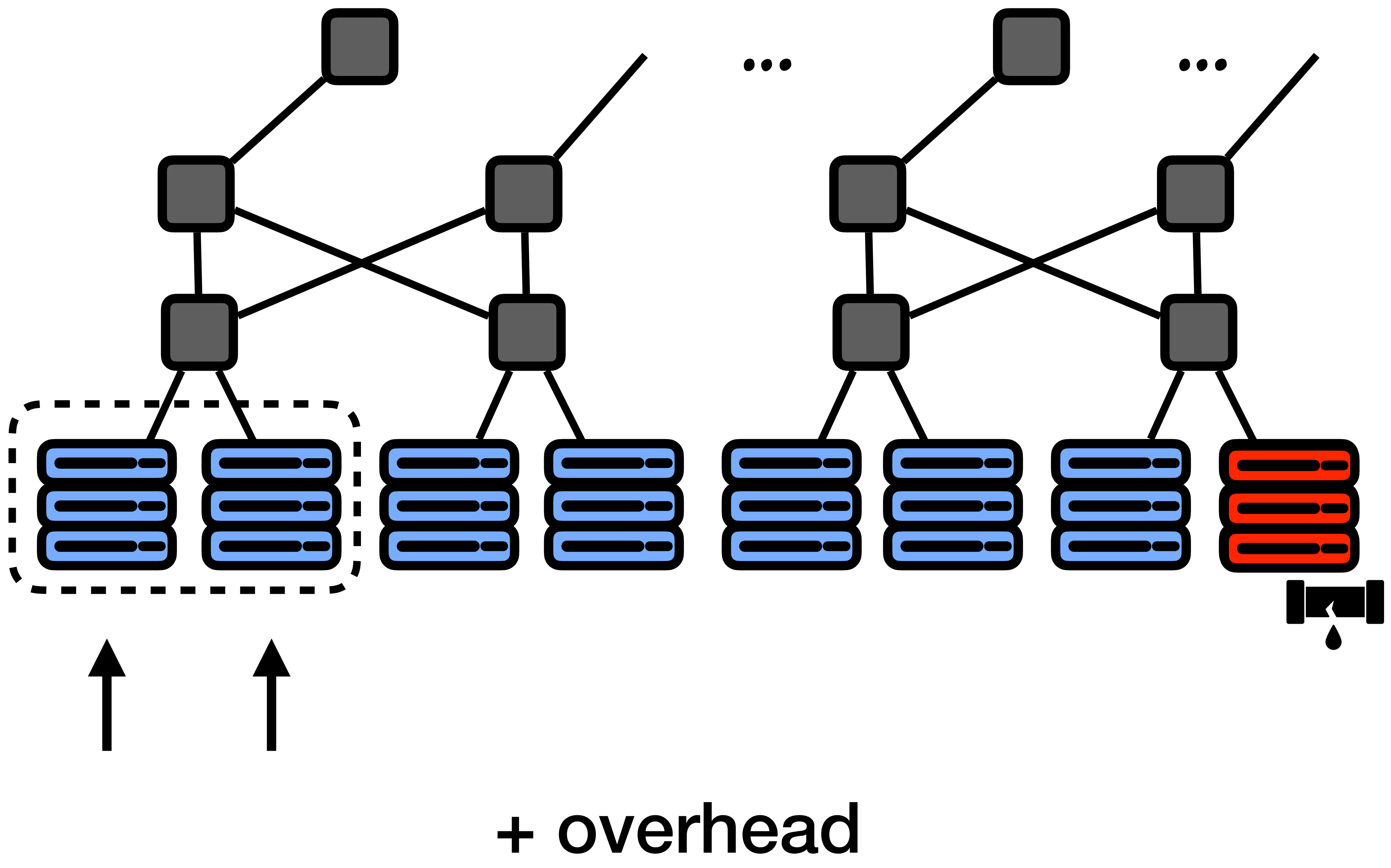
- Practice 2: dynamic approach
  - instrument programs and track the object lifetime at runtime to find leaked objects
  - detect leaks and pinpoint leaked objects

Why detection is still a problem in cloud?



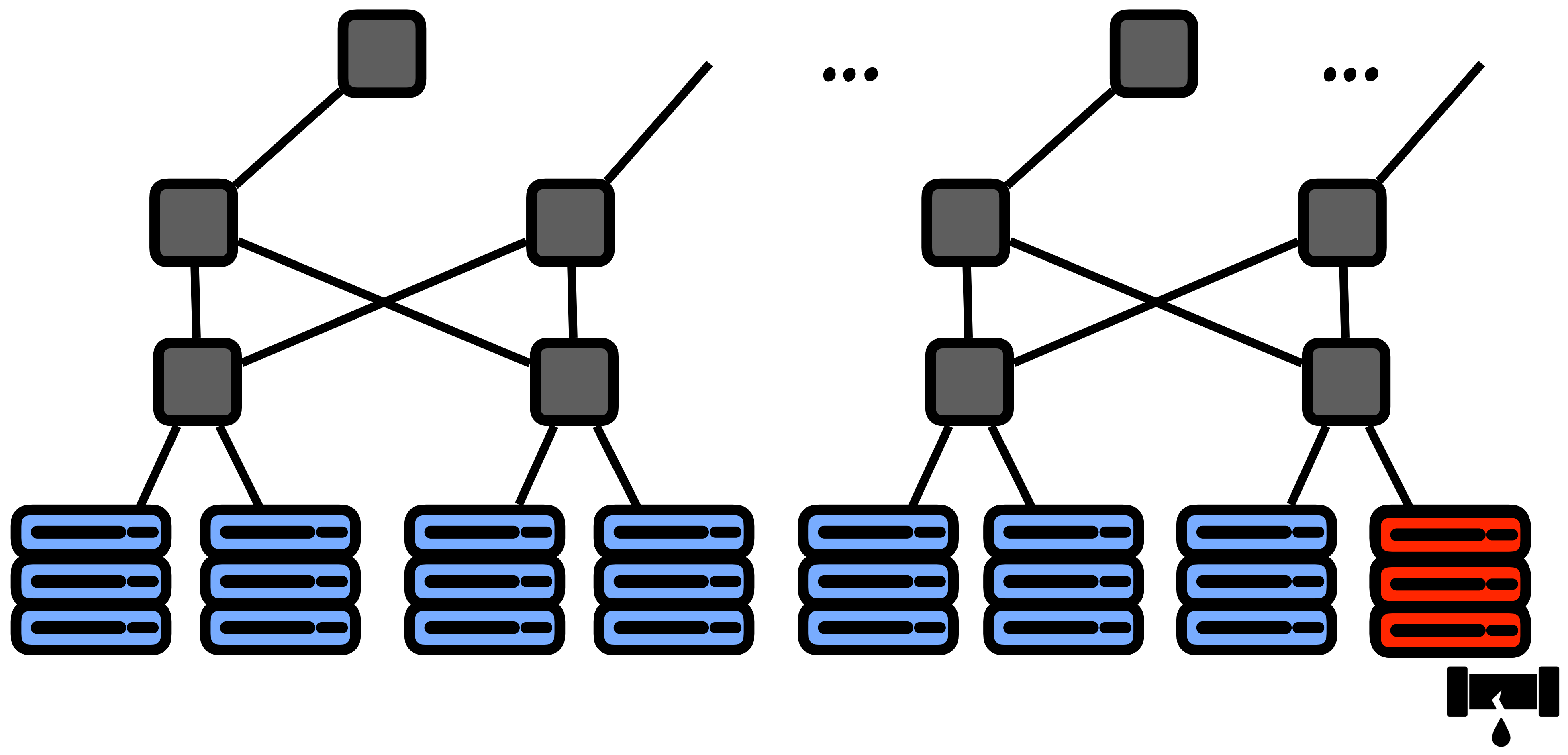
- Practice 2: dynamic approach
  - instrument programs and track the object lifetime at runtime to find leaked objects
  - detect leaks and pinpoint leaked objects

Why detection is still a problem in cloud?



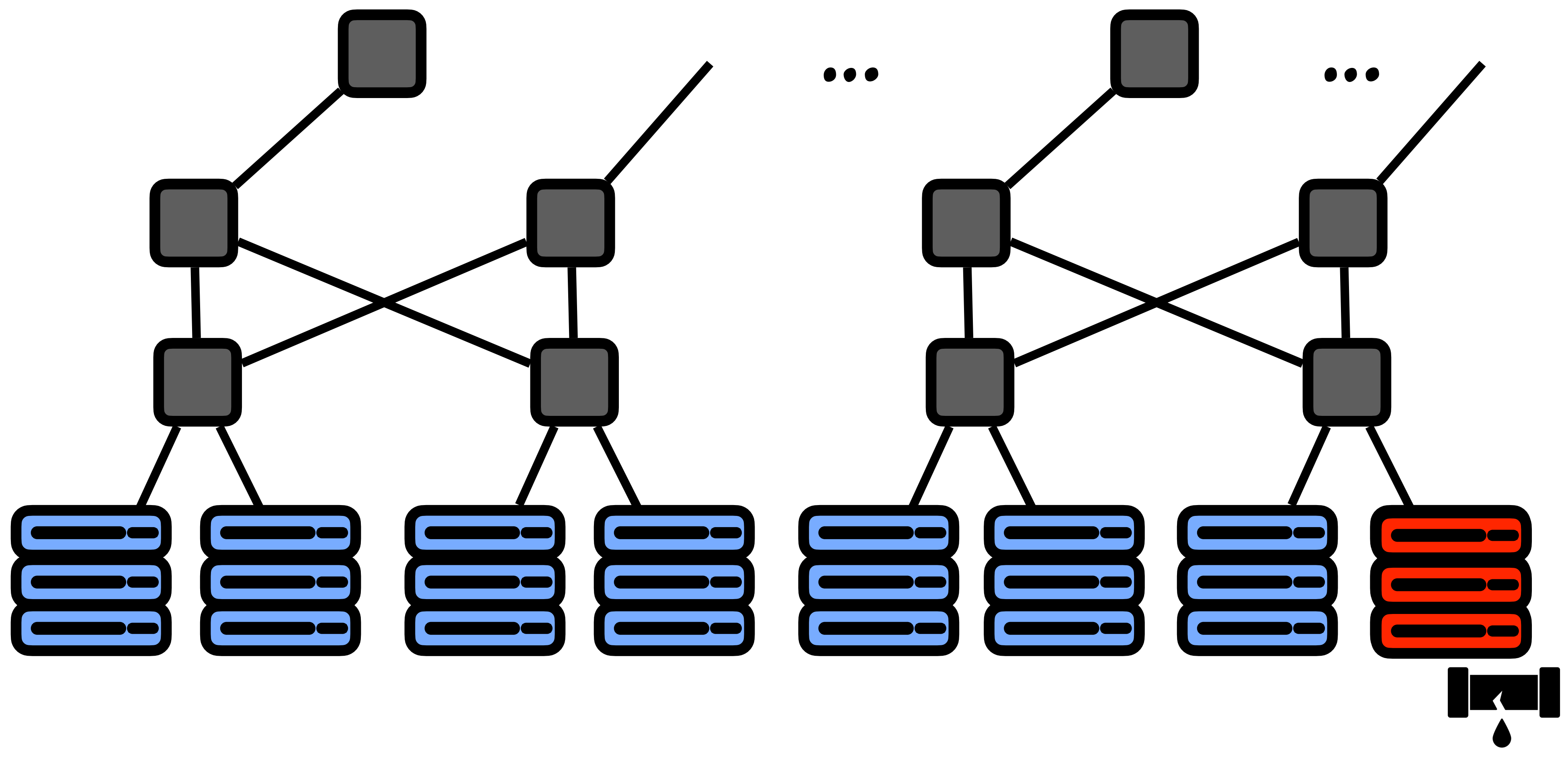
- Practice 2: dynamic approach
  - instrument programs and track the object lifetime at runtime to find leaked objects
  - detect leaks and pinpoint leaked objects
- Limitations
  - hard tradeoff among accuracy, scalability and overhead

Detecting memory leaks with RESIN



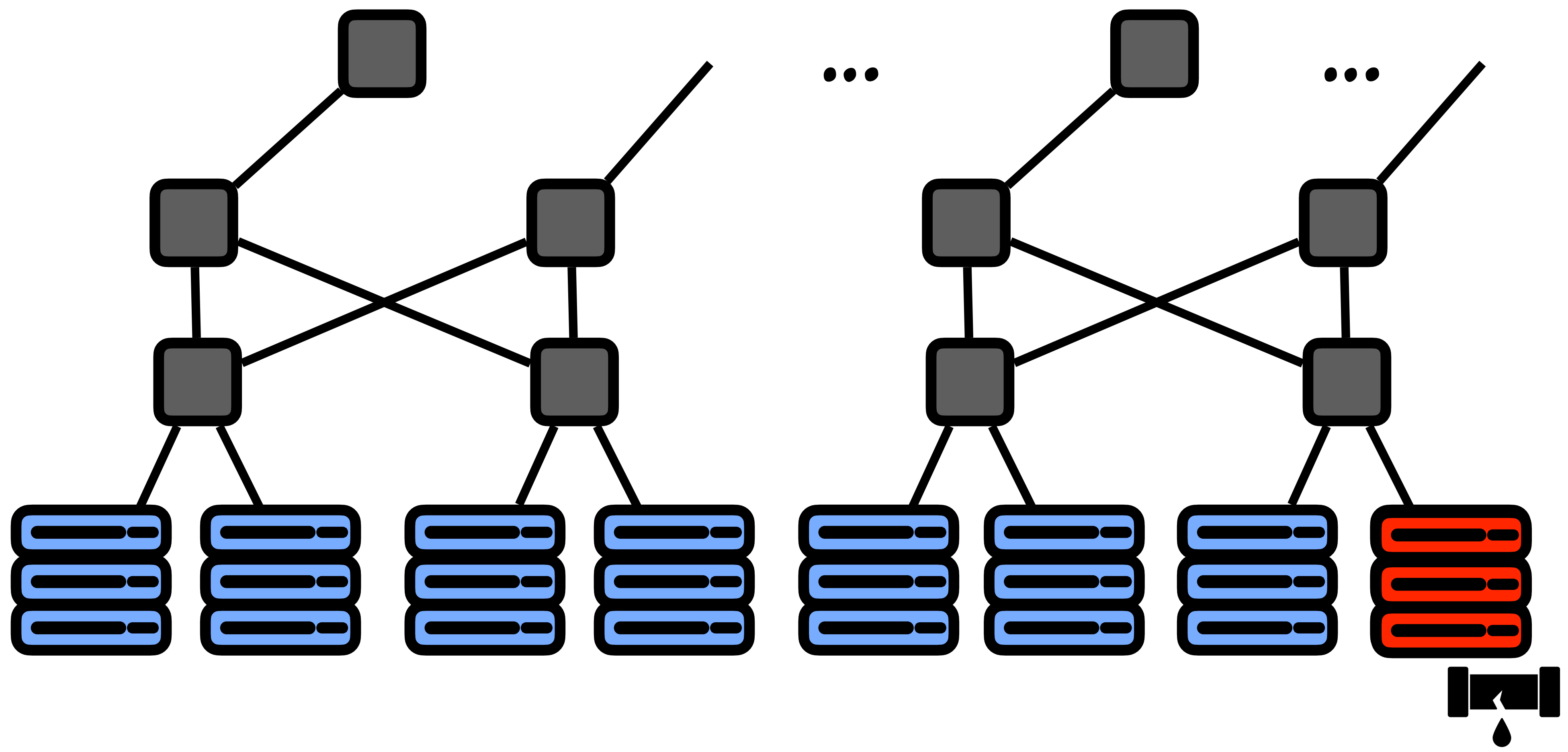
- Our response is RESIN
  - achieve accuracy, scalability and low overhead all together

Detecting memory leaks with RESIN



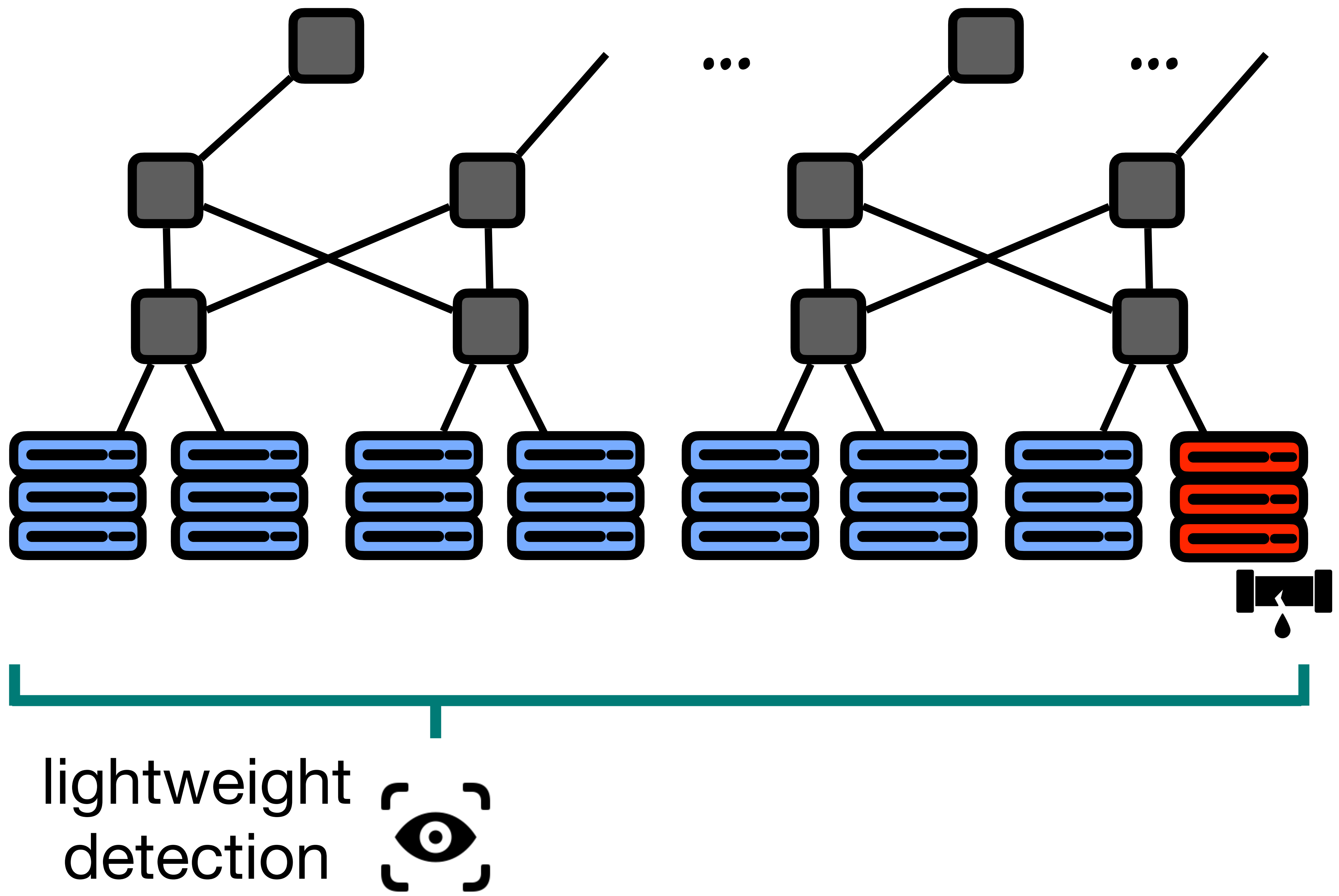
- Our response is RESIN
  - achieve accuracy, scalability and low overhead all together
- Insight 1
  - break mixed detecting and pinpointing

Detecting memory leaks with RESIN



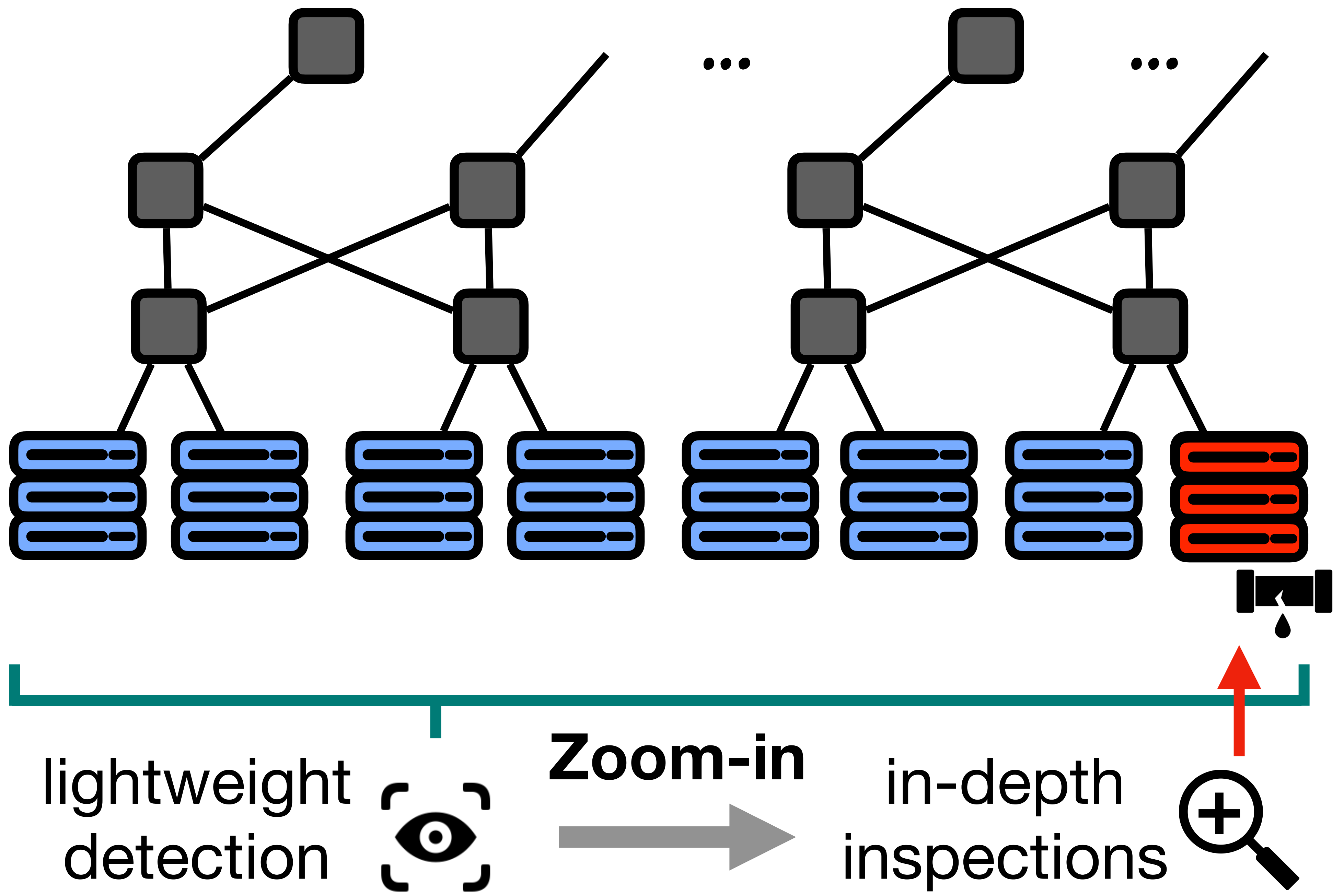
- Our response is RESIN
  - achieve accuracy, scalability and low overhead all together
- Insight 1
  - break mixed detecting and pinpointing
  - decompose detection to multi-stages

Detecting memory leaks with RESIN



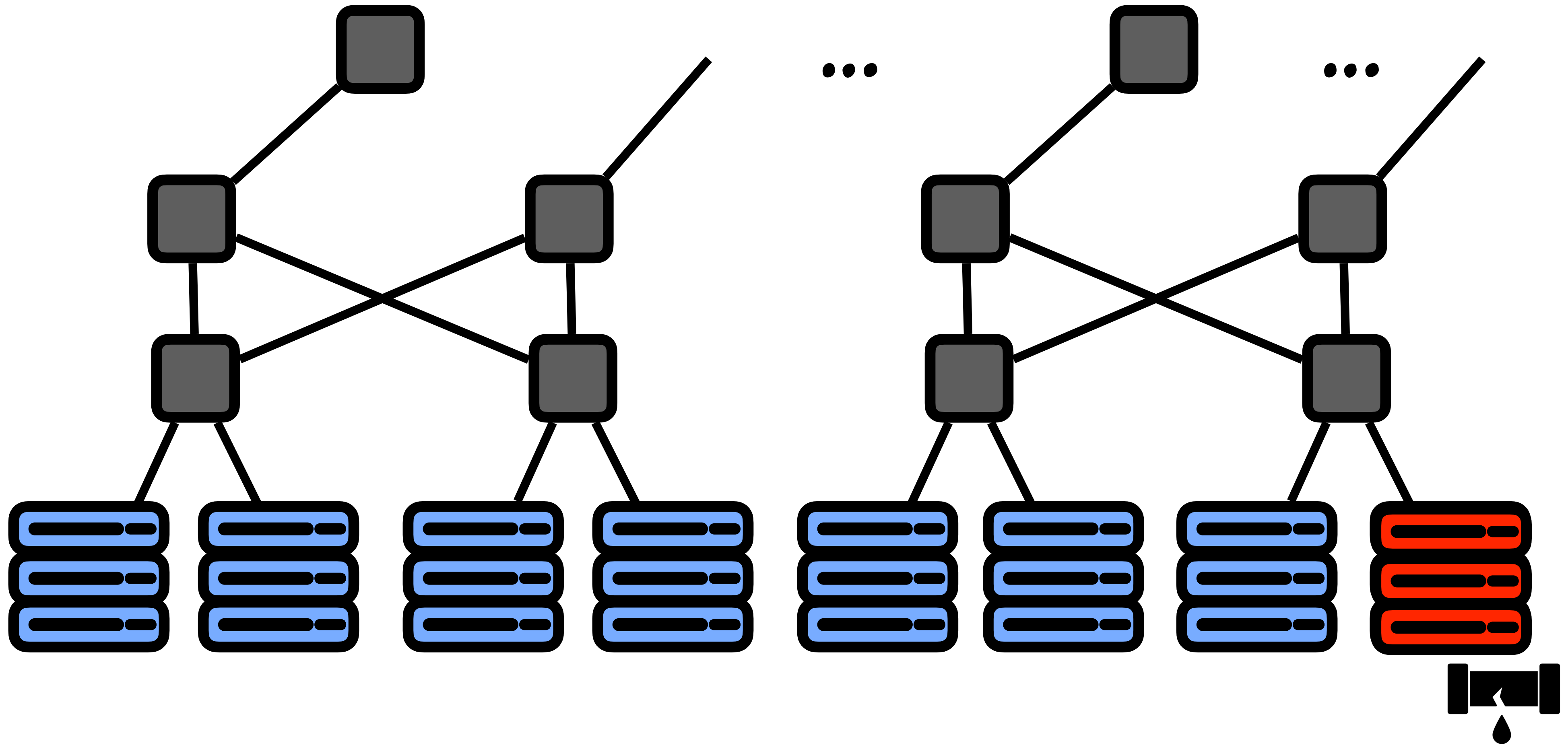
- Our response is RESIN
  - achieve accuracy, scalability and low overhead all together
- Insight 1
  - break mixed detecting and pinpointing
  - decompose detection to multi-stages

Detecting memory leaks with RESIN



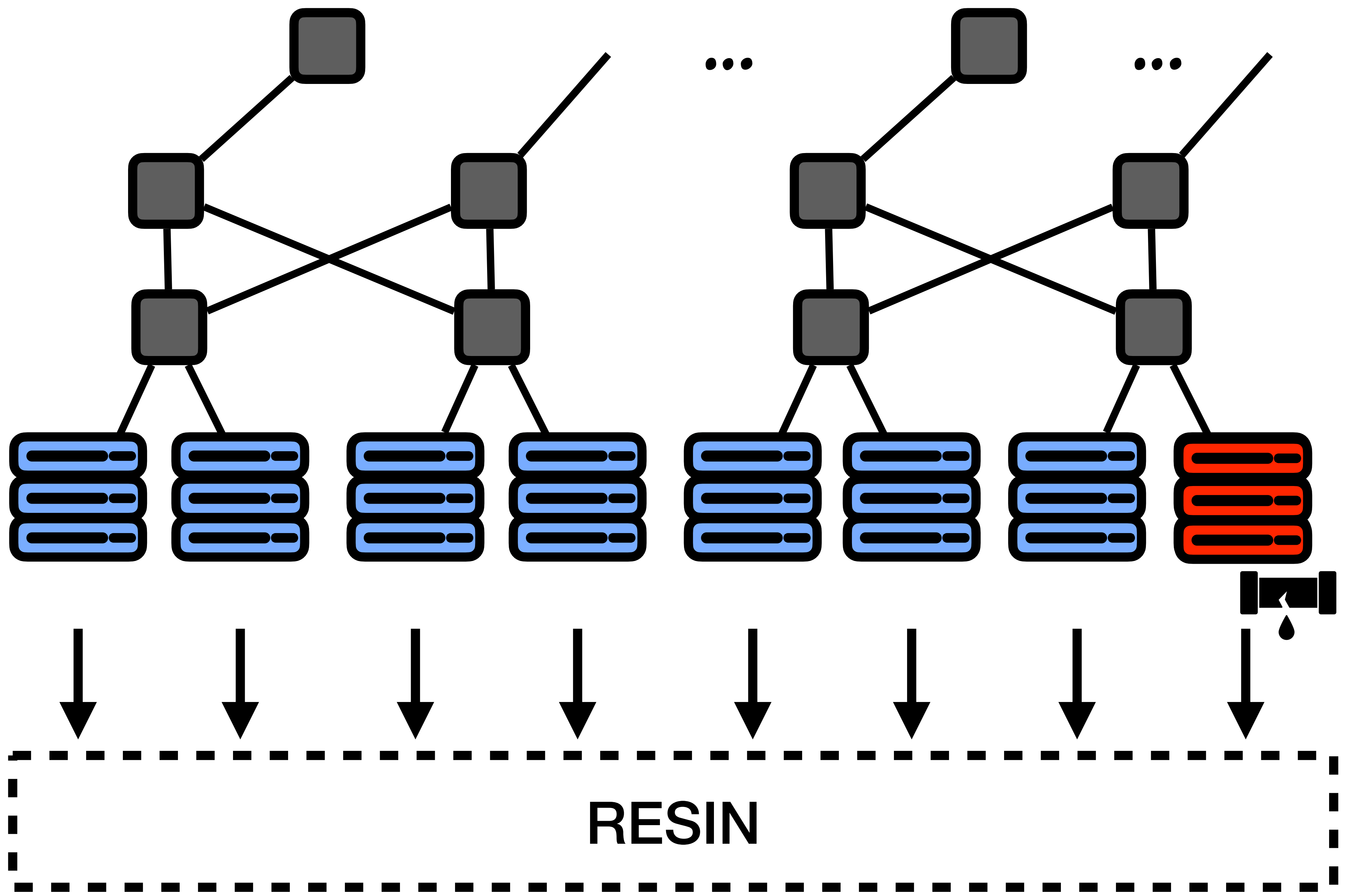
- Our response is RESIN
  - achieve accuracy, scalability and low overhead all together
- Insight 1
  - break mixed detecting and pinpointing
  - decompose detection to multi-stages

Detecting memory leaks with RESIN



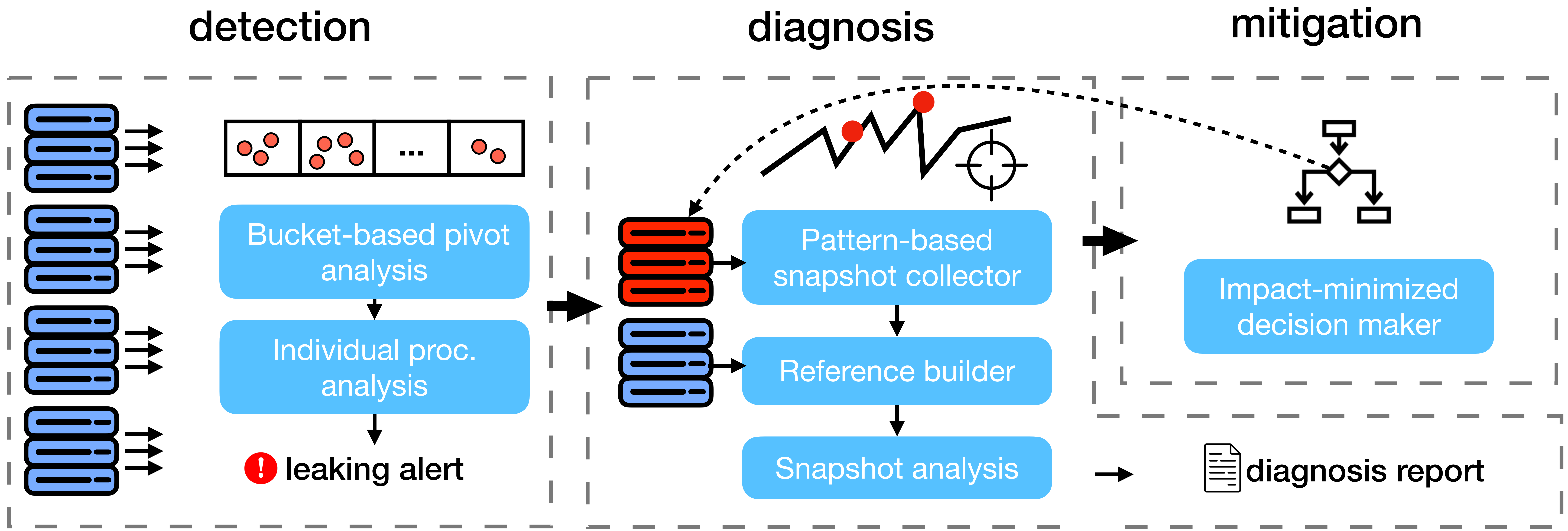
- Our response is RESIN
  - achieve accuracy, scalability and low overhead all together

Detecting memory leaks with RESIN



- Our response is RESIN
  - achieve accuracy, scalability and low overhead all together
- Insight 2
  - a centralized approach for all components
  - leverage power of scale to improve accuracy

RESIN overview



Outline

1. Motivation
2. Two-stage leak detection
3. Trace collection and diagnosis of detected leaks
4. In-production evaluation

Outline

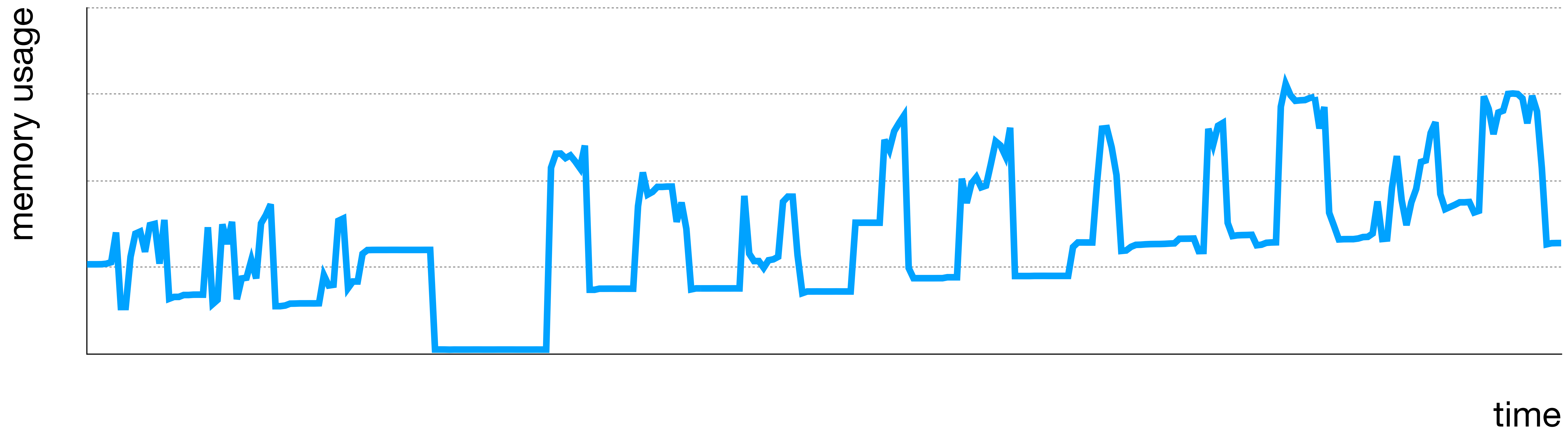
1. Motivation
2. Two-stage leak detection
   1. which component is leaking cluster-wide?
   2. on which hosts that component is leaking?
3. Trace collection and diagnosis of detected leaks
4. In-production evaluation

Detect leaking component

- A straightforward solution:
  - run anomaly detection on time-series data of memory usage for each host
- What are the challenges?

Challenges on detecting memory leaks in cloud

- **Challenge 1:** noisy signals from environment
  - many different workloads in the cloud with dynamic characteristics
  - false positives easily incur



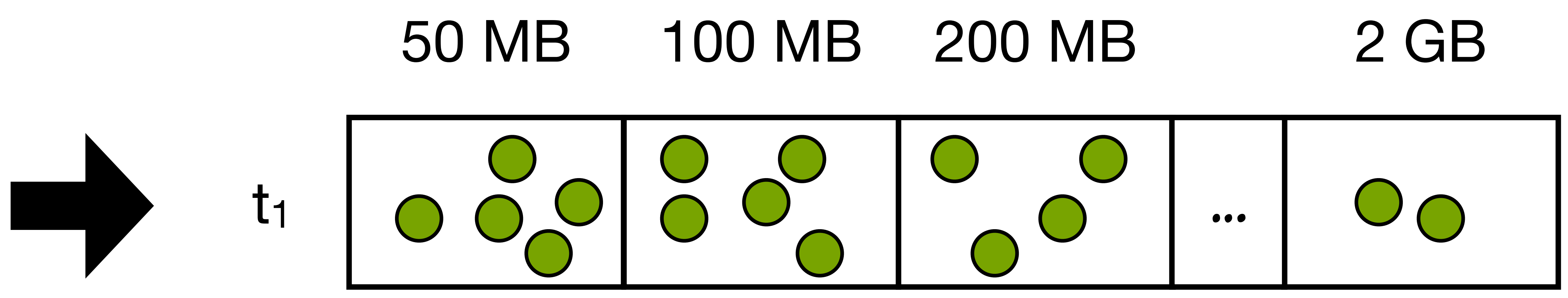
Challenges on detecting memory leaks in cloud

- **Challenge 1:** noisy signals from environment
  - many different workloads in the cloud with dynamic characteristics
  - detection false positives easily incur
- **Challenge 2:** slow leaks in long-running services
  - memory leaks often last over days or weeks
  - need to capture gradual changes meanwhile alerting in time
- **Challenge 3:** large profiling data volumes
  - need to analyze >10 TB memory usage data daily

Solution: bucket-based pivot analysis

- Each bucket is a collection of hosts with memory usage in a same range
  - this 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**)

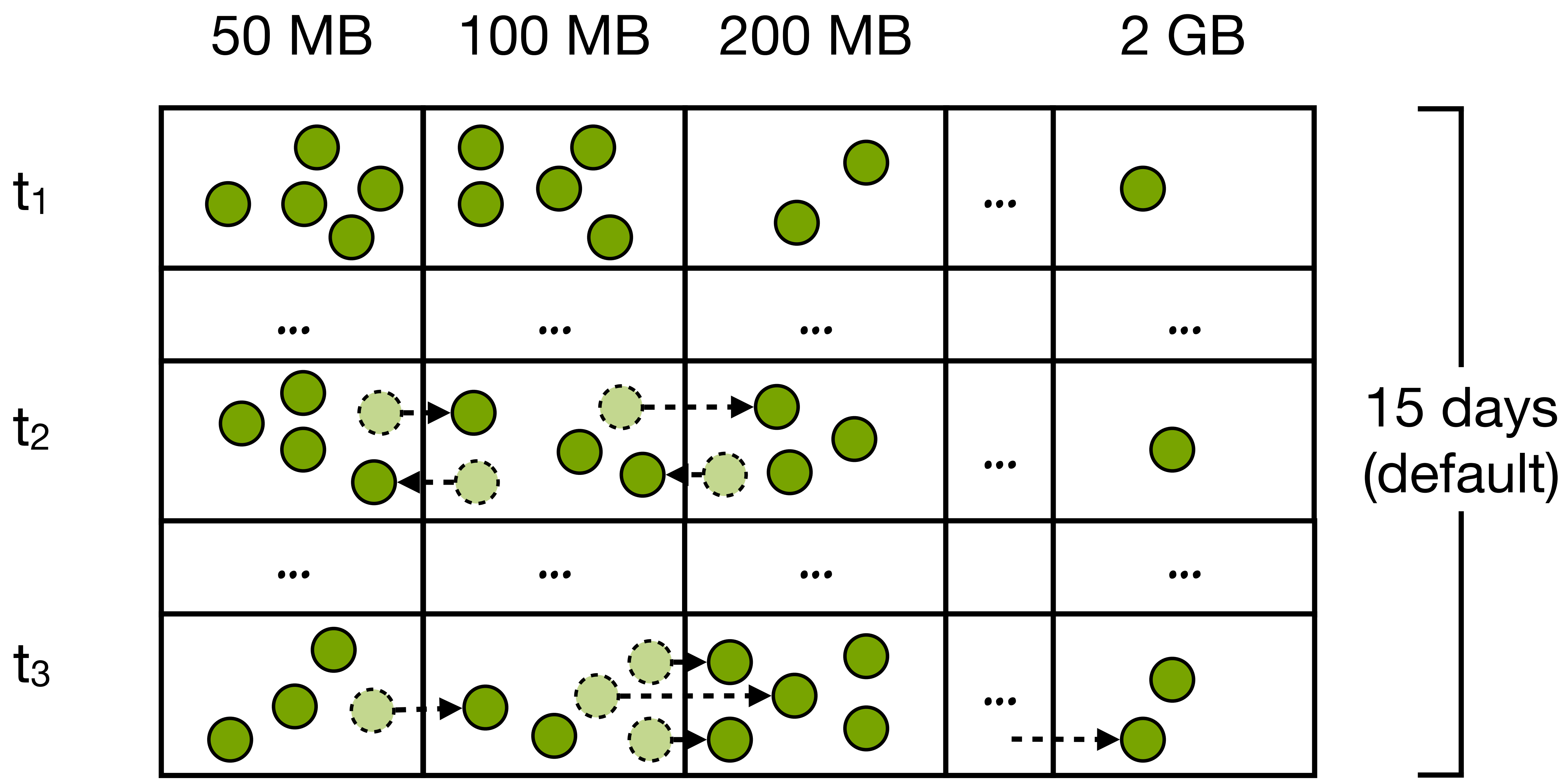
Time stamp	ImageName	Cluster	NodeId	PID	Private Usage	...
t_1	**firewall.exe**	NorthUS-1da	9das-sax1	254	2,334,720	
t_1	**firewall.exe**	NorthUS-9lp	9das-yq0c	979	90,413,120	
t_1	**firewall.exe**	Asia-b2	o1oz-bg75	1375	170,341,311	
t_1	...				...	



buckets of firewall.exe

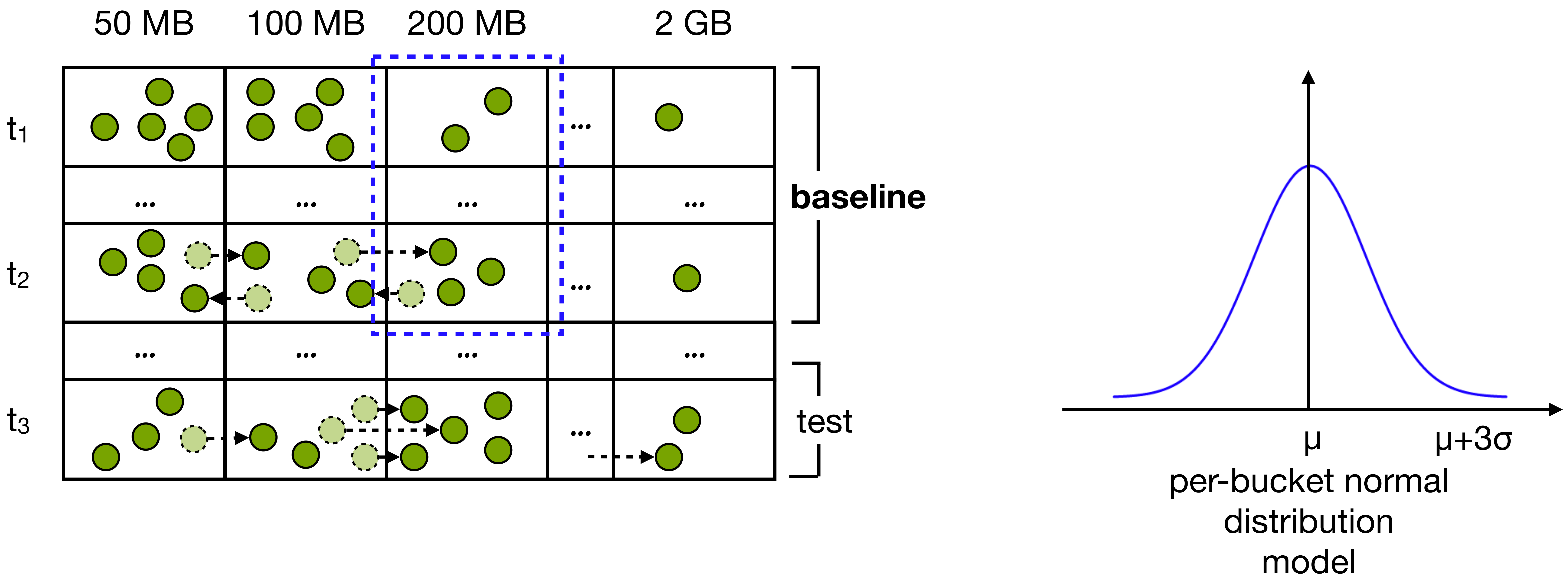
Solution: bucket-based pivot analysis

- Summaries from recent time-series data
  - able to detect slow leaks for weeks (**challenge 2**)



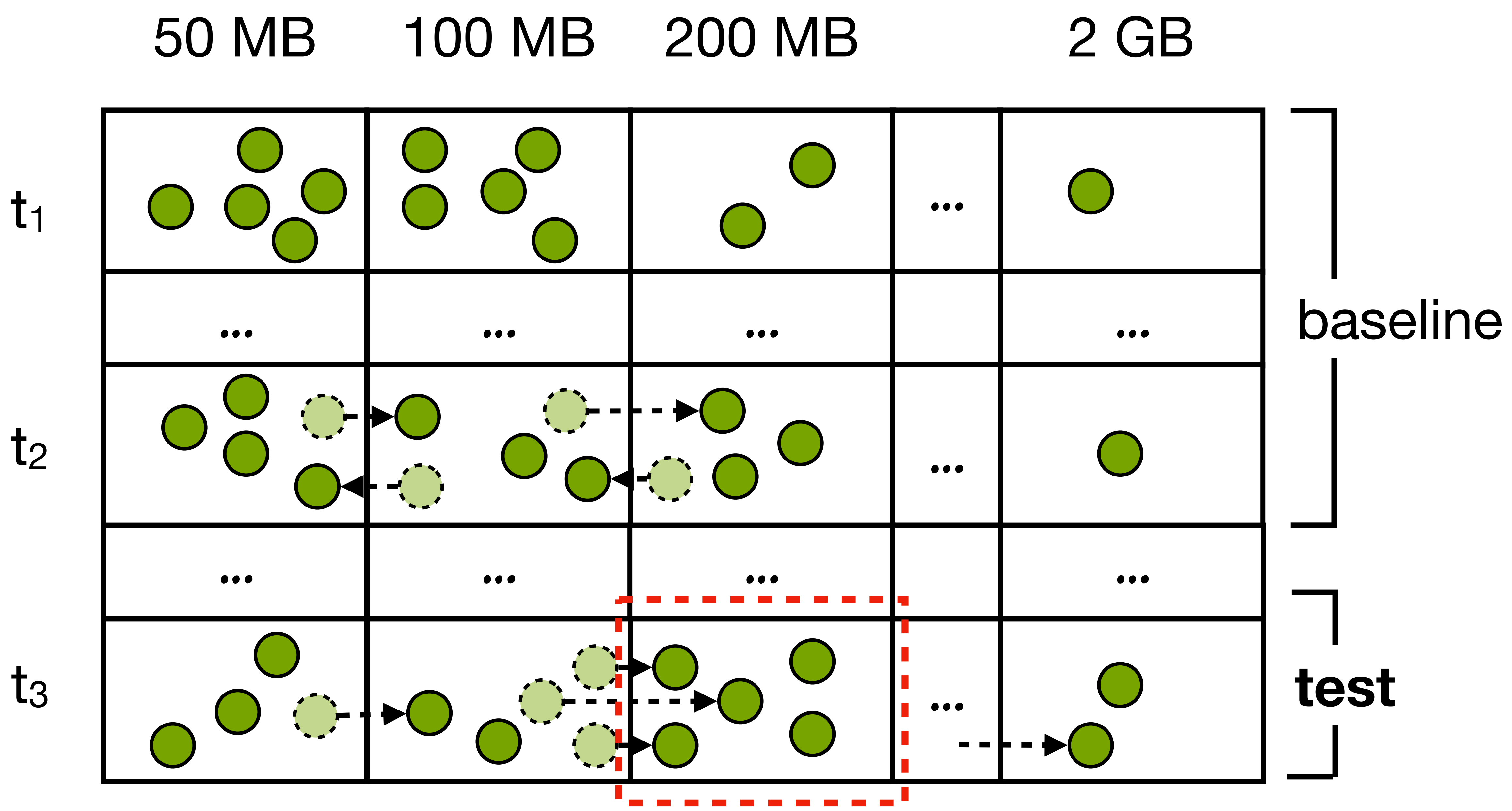
Solution: bucket-based pivot analysis

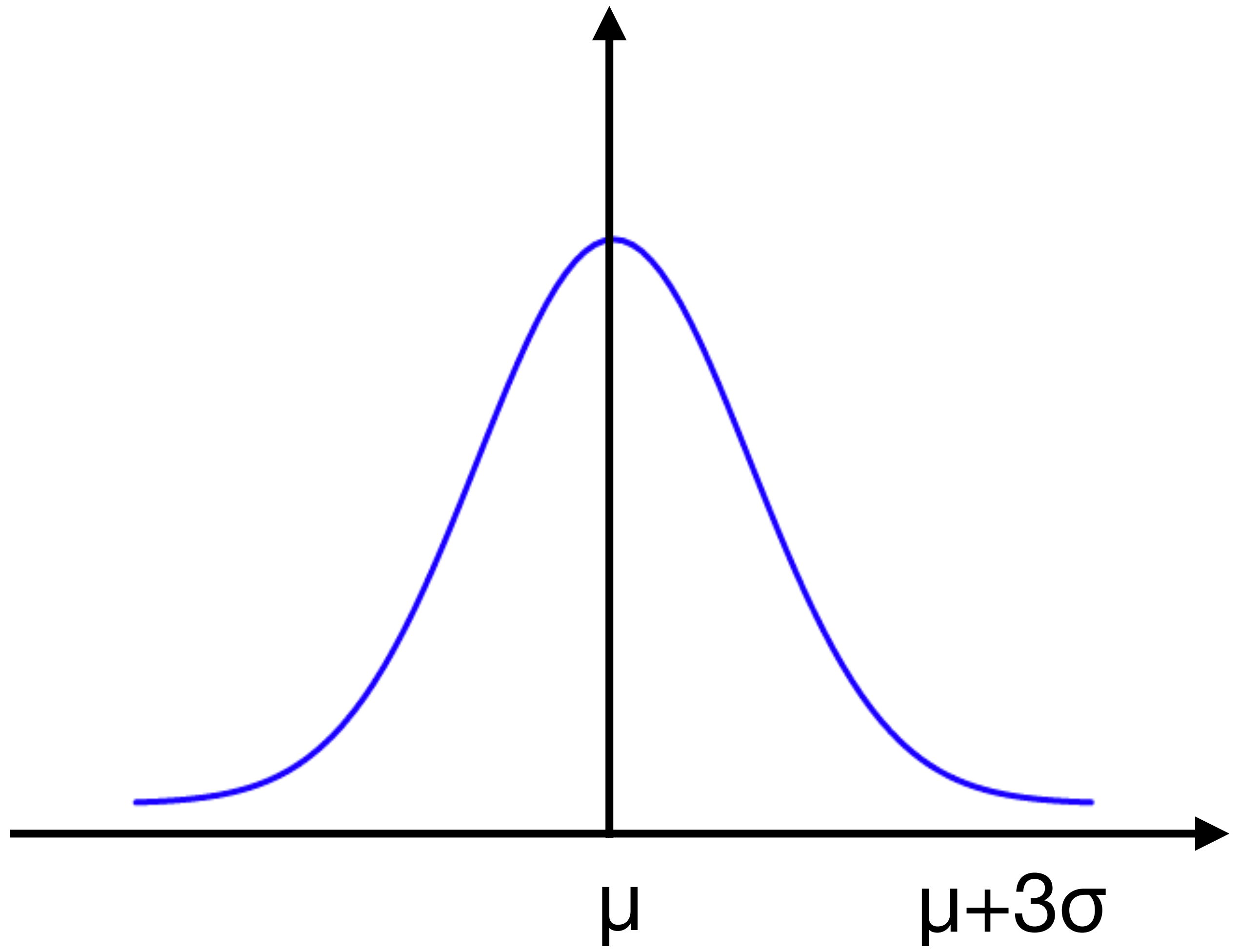
- Run anomaly detection against time series of bucket size
  - build normal distribution model from baseline range (2/3 portion)



Solution: bucket-based pivot analysis

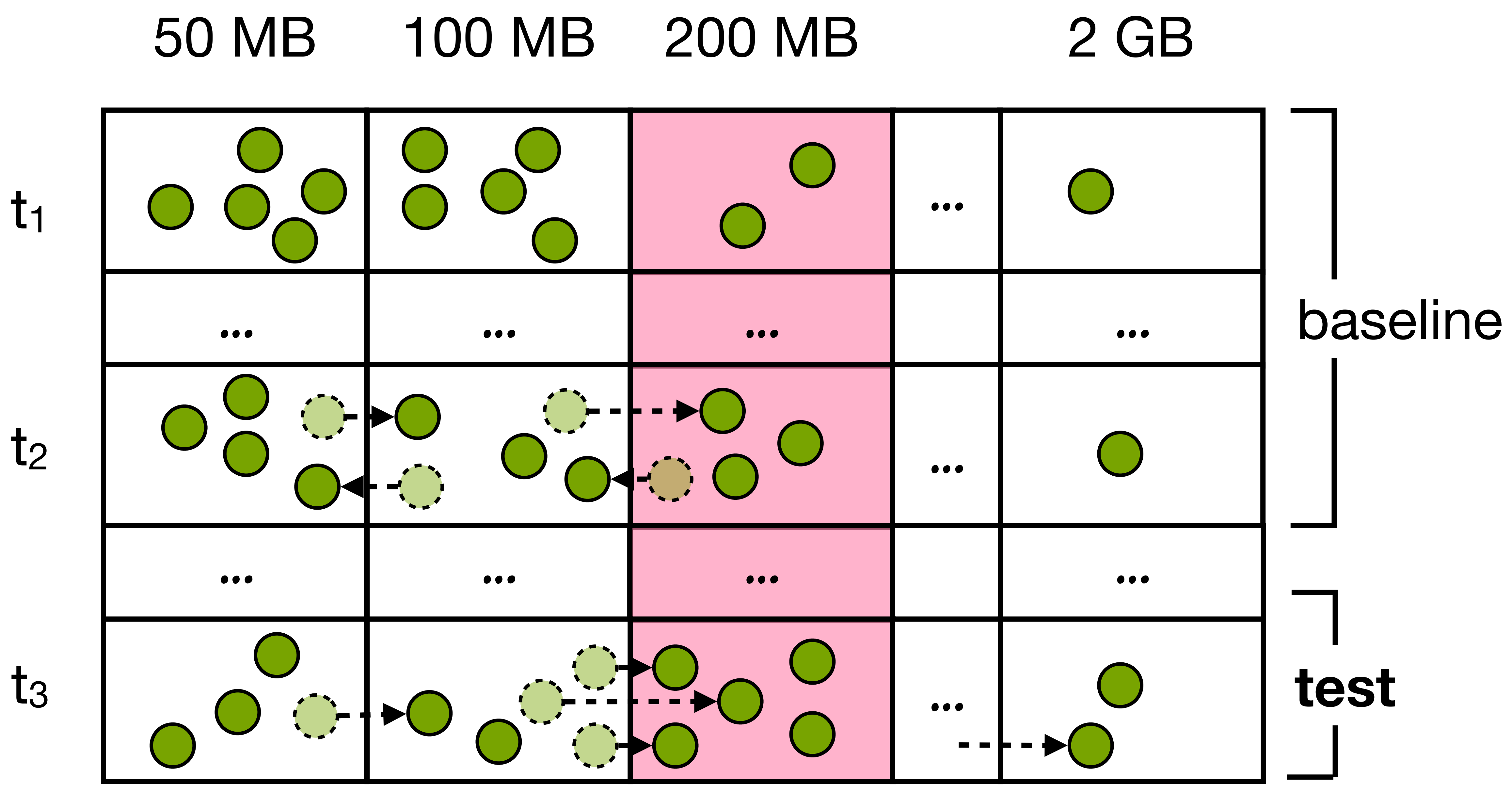
- Run anomaly detection against time series of bucket size
  - use the remaining data points as the test (1/3 portion)



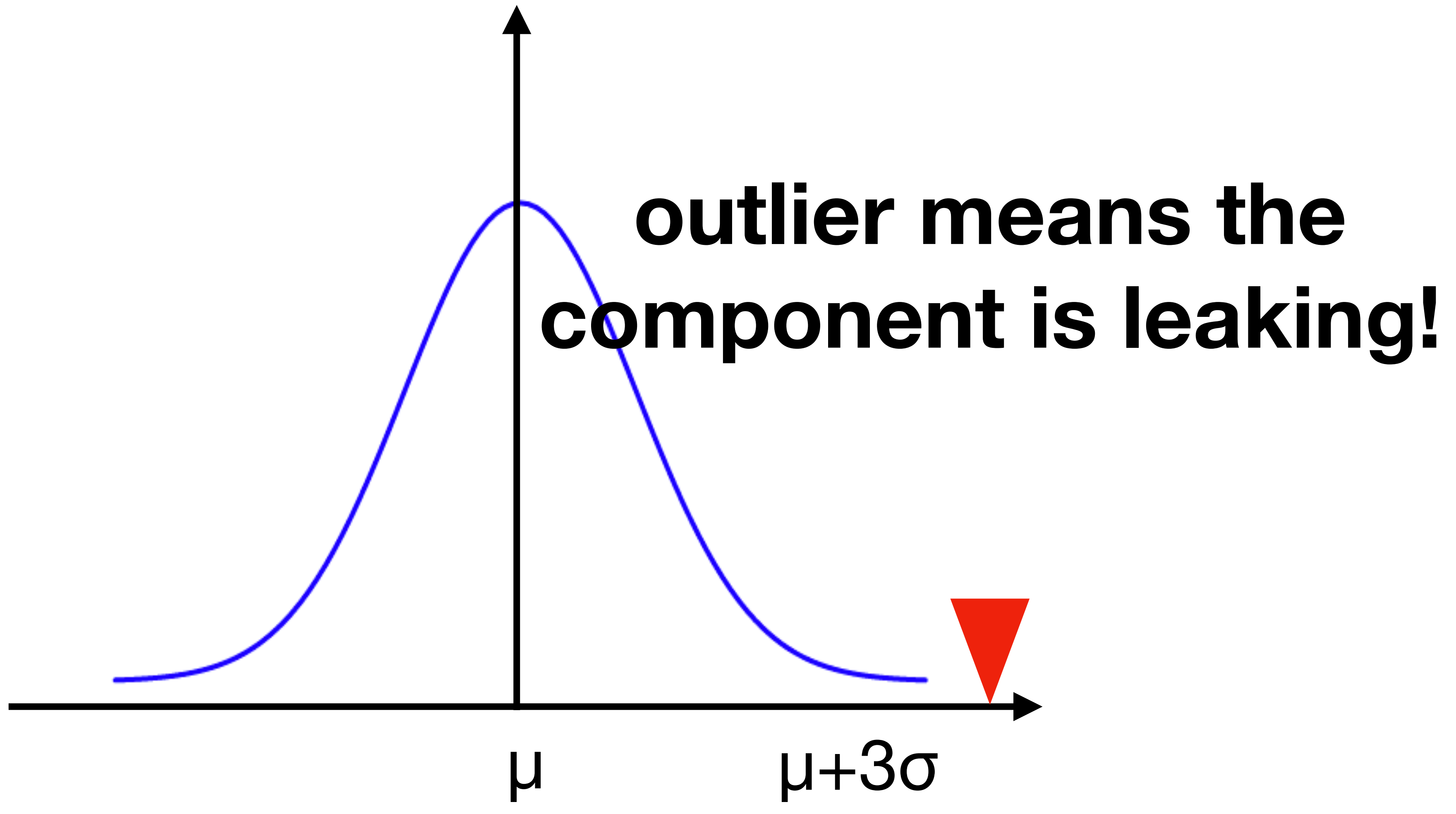


Solution: bucket-based pivot analysis

- Run anomaly detection against time series of bucket size
  - data points that exceed the $\mu + 3\sigma$ [1] of the baseline data are anomaly



alerting bucket!

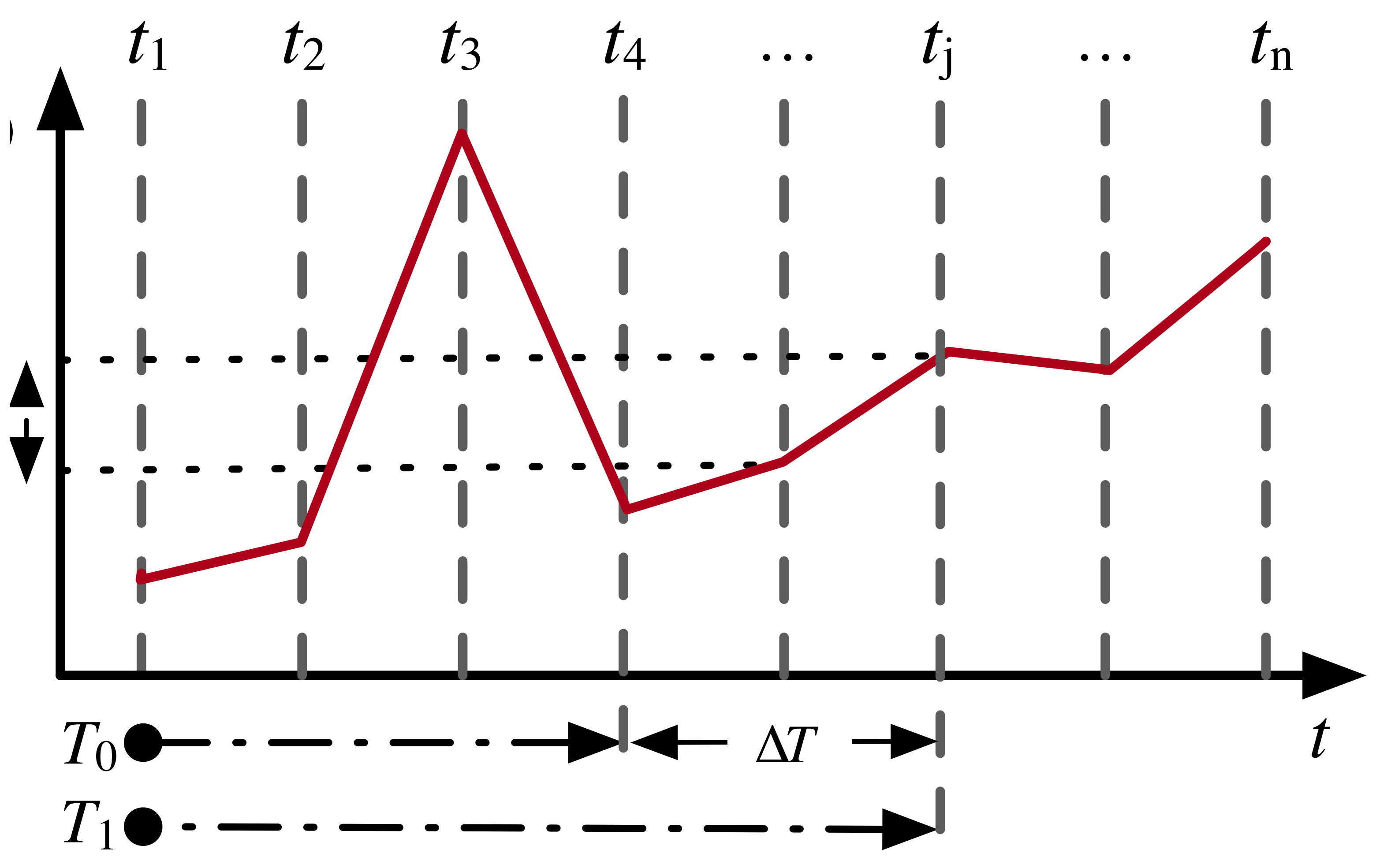


per-bucket normal distribution model

[1] mean and standard deviation of the distribution

Localizing leaking process

- Now we know which component is leaking
- Next question is, how to find on which host the component is leaking?
- Solution: **suspicious window analysis**
  - input: memory usage time-series data on each host
  - output: a list of suspected hosts with
    - leaking time windows
    - severity scores
- See algorithm details in our paper



Outline

1. Motivation
2. Two-stage leak detection
3. Trace collection and diagnosis of detected leaks
    1. what profiling traces are useful for diagnosis?
    2. what is the key challenge to collect traces?
    3. how to analyze the collected traces?
4. In-production evaluation

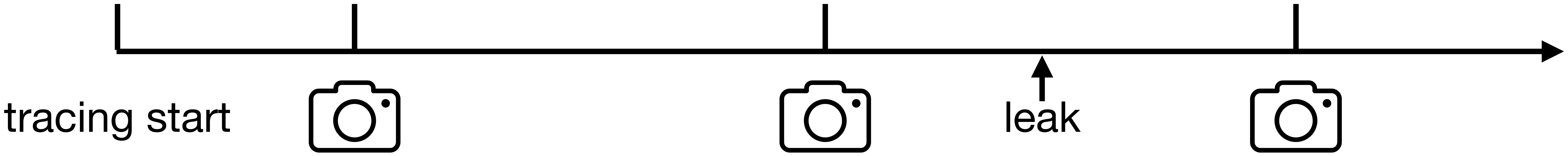
Profiling trace: heap snapshots

- 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

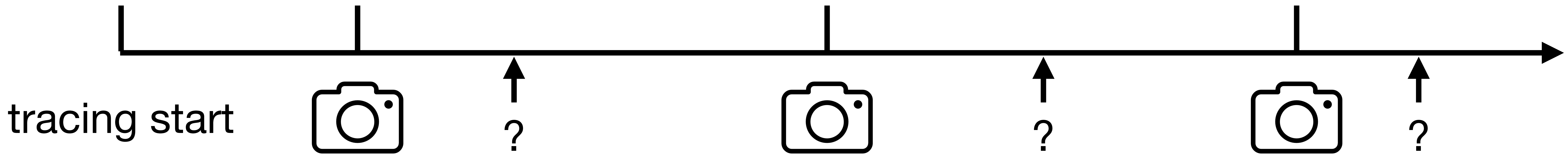
Alloc Addr	Stack Id	Size	RefCount
0x80000	1	64	2
0x90000	1	128	1

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



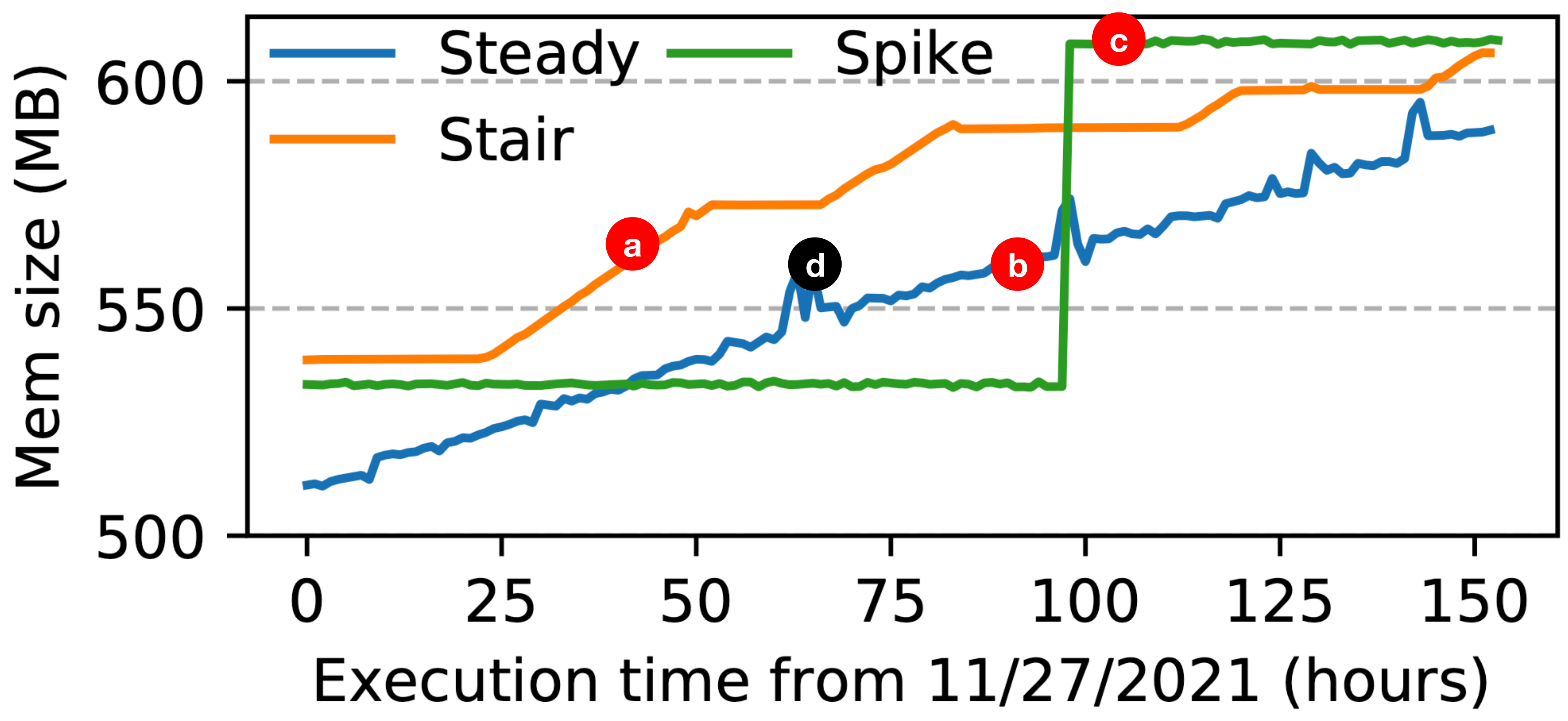
Challenge: decide trace collection timing

- Snapshot differencing requires accurate triggers for leak
- Strawman solution: setting threshold on memory usage difference
    - likely complete the tracing prematurely due to a memory usage spike
    - result in failure to capture the buggy allocation



Solution: collection based on growth pattern

- RESIN collect traces with pattern-based strategy
  - leaks usually exhibits consistent patterns across time
  - we classify the pattern of leak from historical data using simple linear regression
  - RESIN trigger completion based on collection strategy pre-defined for each pattern



Analyzing trace for diagnosis

1. Differentiate allocations between snapshots before and after leak
   - returns a list of allocations containing leaky allocation

Alloc Addr	Stack Id	Size	RefCount
0x80000	1	64	2
0xb0000	2	384	1
0xf0000	3	224	2
0xf0100	4	2560	2

$snapshot_2$

-

Alloc Addr	Stack Id	Size	RefCount
0x80000	1	64	2
0x90000	1	128	1
0xb0000	2	128	1

$snapshot_1$

=

Alloc Addr	Stack Id	Size	RefCount
0xb0000	2	256	1
0xf0000	3	224	2
0xf0100	4	2560	2

outstanding allocations

Analyzing trace for diagnosis

2. Sort the allocation list by size

- prioritize allocations whose memory usage is closer to estimated size
- challenge: the list still contains some noisy allocations, how to filter them?

Alloc Addr	Stack Id	Size	RefCount
0xb0000	2	256	1
0xf0000	3	224	2
0xf0100	4	2560	2

outstanding allocations

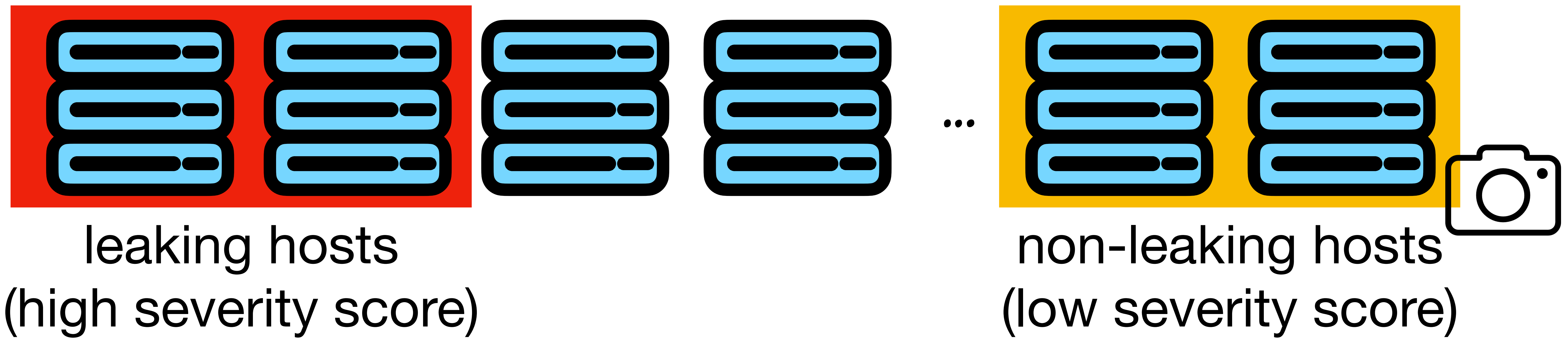
→

Alloc Addr	Stack Id	Size	RefCount
0xf0100	4	2560	2
0xb0000	2	256	1
0xf0000	3	224	2

outstanding allocations (sorted)

Solution: references from non-leaking hosts

- Collect reference snapshots to filter noises
  - fingerprint leaking processes and find its non-leaking hosts as references
    - (cluster_id, OS version, service version, date)
  - collect heap snapshots to retrieve stack traces from **normal** workloads



Analyzing trace for diagnosis

3. Filter likely noisy allocations

- remove allocations larger than estimated size or from reference snapshots
- output diagnosed stack trace as result

Alloc Addr	Stack Id	Size	RefCount
0xf0100	4	2560	2
0xb0000	2	256	1
0xf0000	3	224	2

outstanding allocations
(sorted)

Analyzing trace for diagnosis

3. Filter likely noisy allocations

- remove allocations larger than estimated size or from reference snapshots
- output diagnosed stack trace as result

top in reference snapshot allocation lists

Alloc Addr	Stack Id	Size	RefCount
~~0xf0100~~	~~4~~	~~2560~~	~~2~~
0xb0000	2	256	1
~~0xf0000~~	~~3~~	~~224~~	~~2~~

outstanding allocations (sorted)

Analyzing trace for diagnosis

3. Filter likely noisy allocations

- remove allocations larger than estimated size or from reference snapshots
- output diagnosed stack trace as result

top in reference snapshot allocation lists

Alloc Addr	Stack Id	Size	RefCount
~~0xf0100~~	~~4~~	~~2560~~	~~2~~
0xb0000	2	256	1
~~0xf0000~~	~~3~~	~~224~~	~~2~~

outstanding allocations (sorted)

stack trace

- ConfManager::ApplyUnlocked
  - Conf::Apply
    - FirewallRuleInfo::Create
      - Firewall::AddRule

Outline

1. Motivation
2. Two-stage leak detection
3. Trace collection and diagnosis of detected leaks
4. In-production evaluation

RESIN deployment status and scale

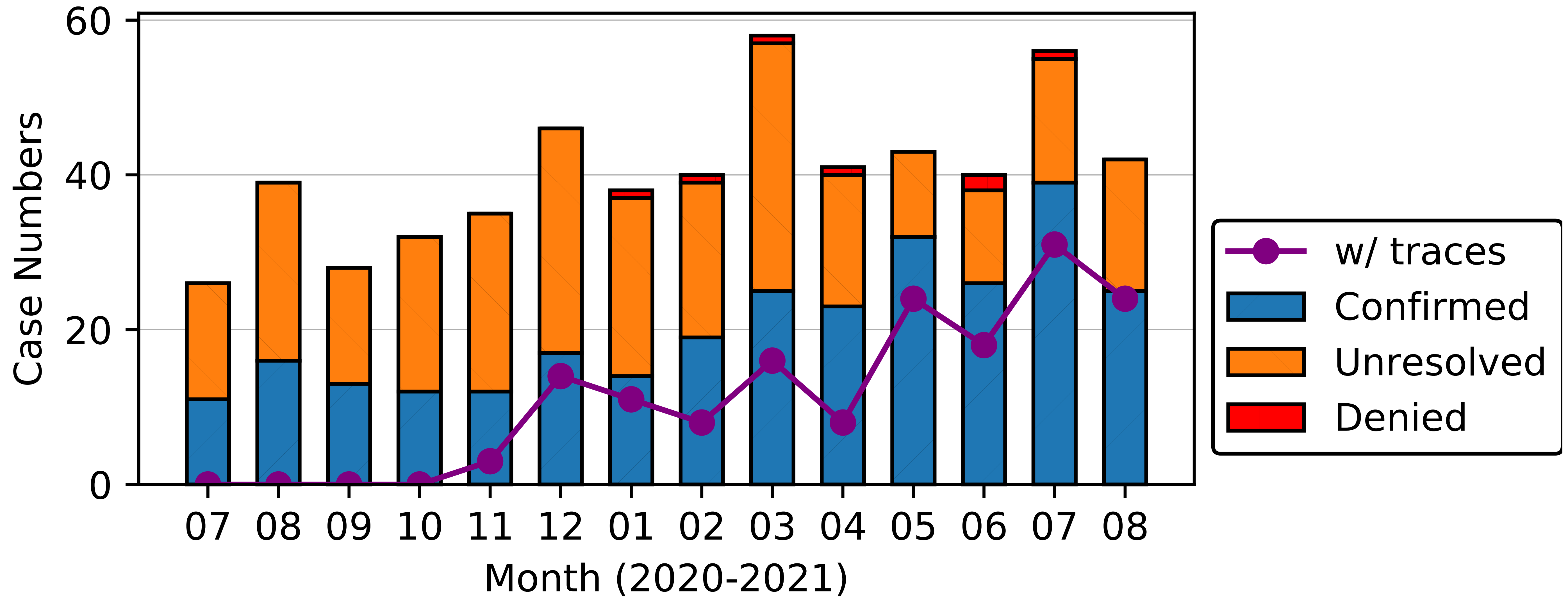
- Running in Azure production since late 2018
  - cover **millions** of hosts
  - detect leaks for **600+** host processes
  - detect leaks for **800+** kernel pool tags
  - the detection engine analyzes more than **10 TB** memory usage data daily
  - the diagnosis module collects **56** traces on average daily

In-production evaluation

- Our evaluation aims to answer questions:
  - (1) how effective is RESIN in addressing memory leaks in Azure?
  - (2) how accurate is the detection?
  - (3) can RESIN help developers diagnose leaks?
  - (4) what is the overhead of trace collection?
  - ...

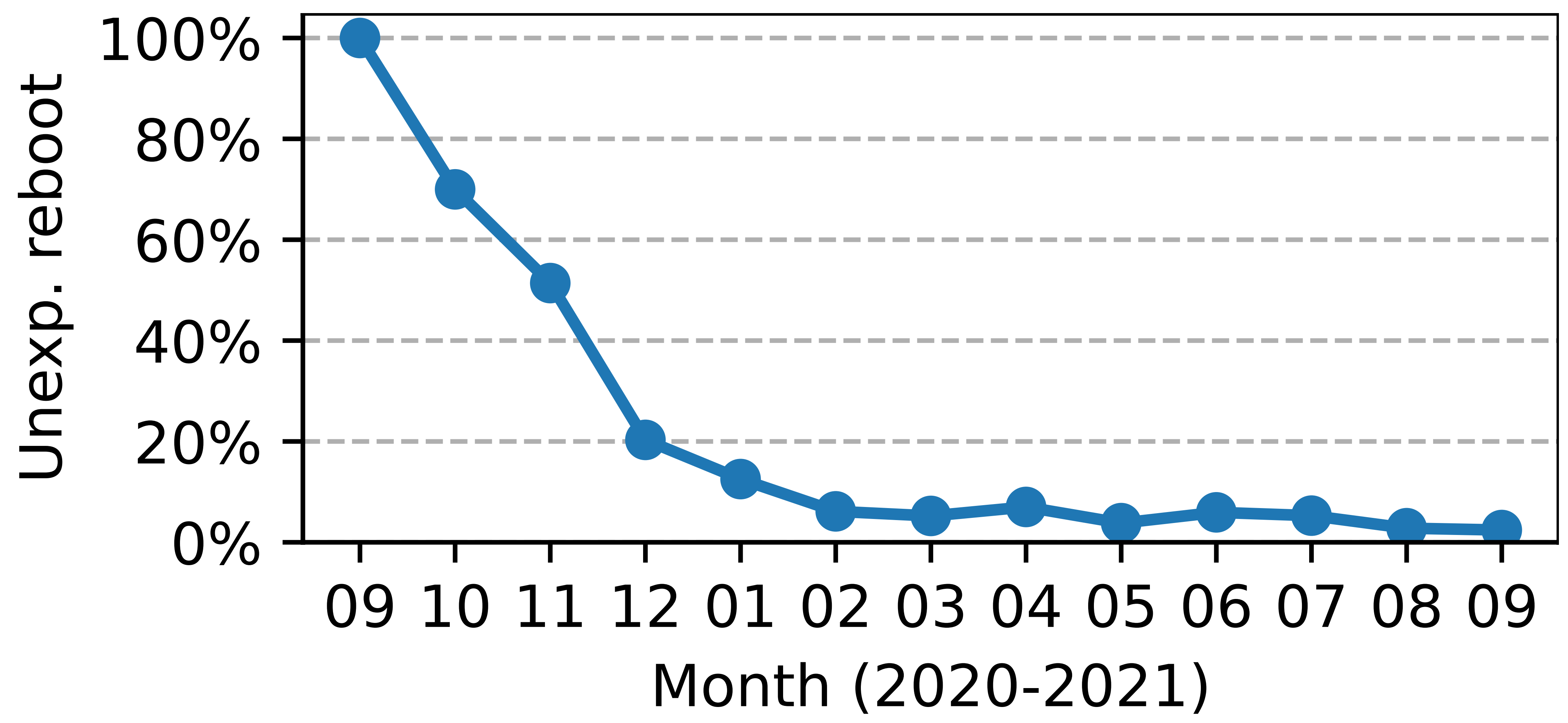
Evaluation setting

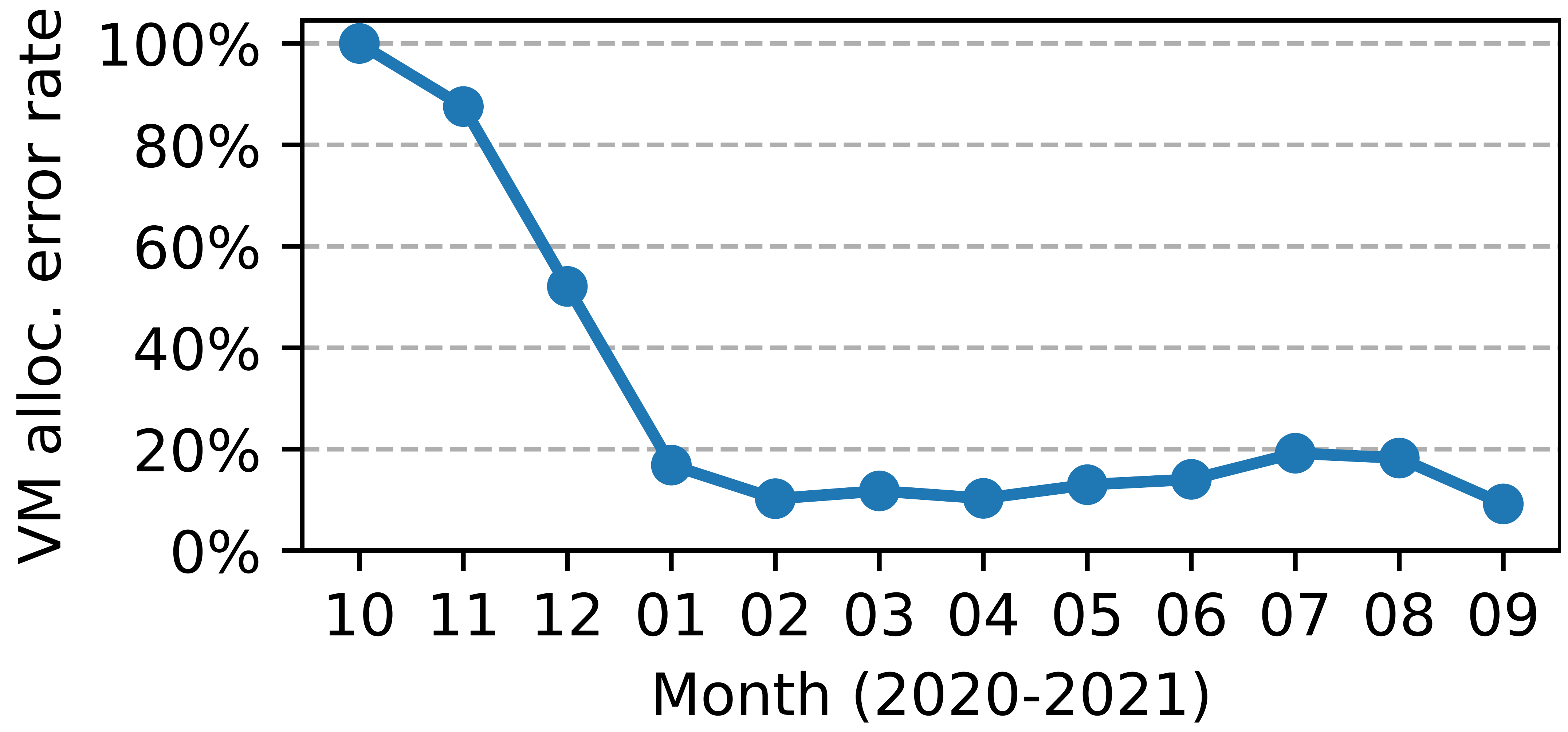
- We collected data from July 2020 to August 2021
  - the detection engine reports **564** tickets in total
  - developers explicitly resolved **291** (52%) tickets



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





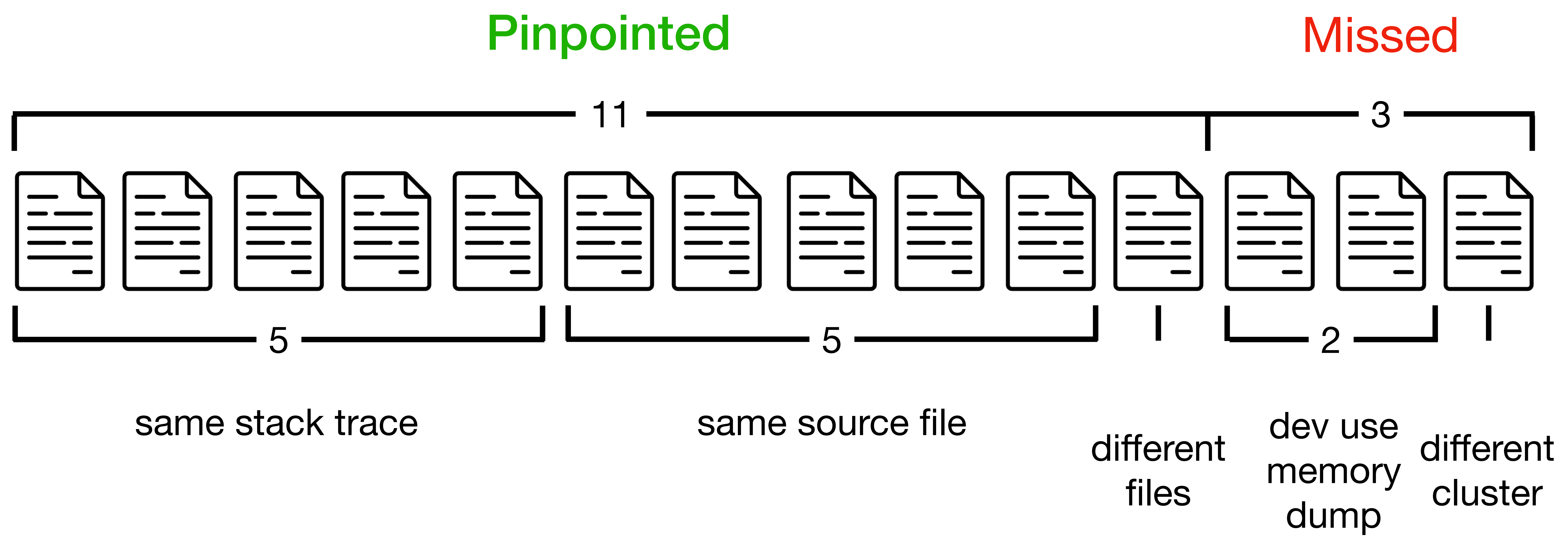
* data is normalized

How accurate is the detection?

- 7 false positives out of 291 resolved cases
  - caused by new software features or configuration changes
- 4 false negatives not covered in RESIN’s reports among 14 months
  - the leak bugs were captured by developers before causing noticeable impact

Can RESIN help developers diagnose leaks?

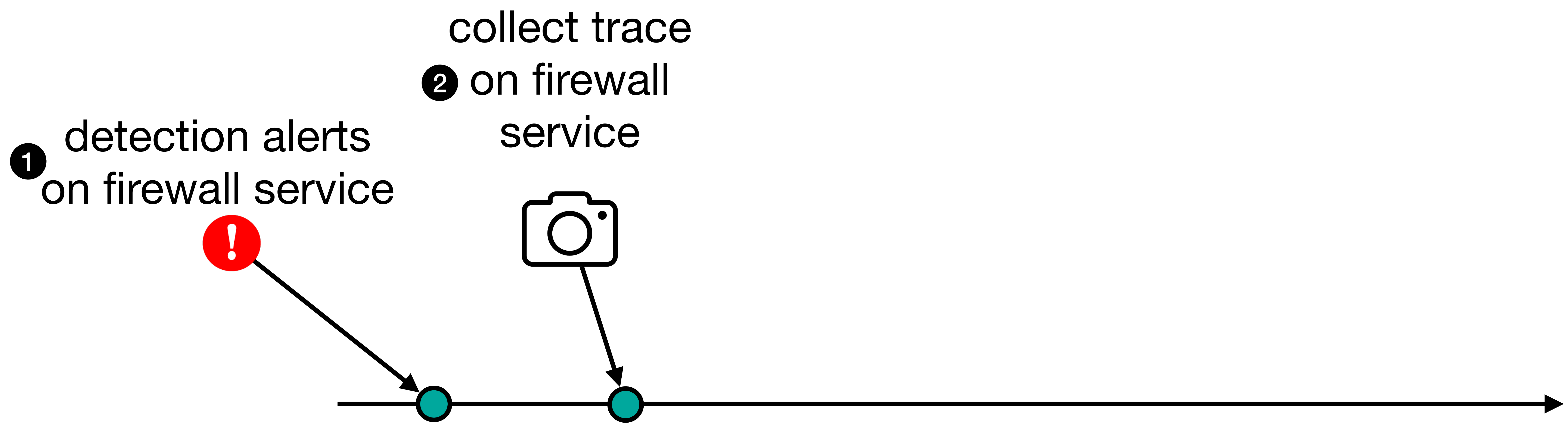
- RESIN collects traces and generates reports for **157** cases
  - we followed debugging **14** issues to validate diagnosis usefulness
  - directly pinpoint for **11** out of 14 cases
  - save developers days to weeks on diagnosis workloads



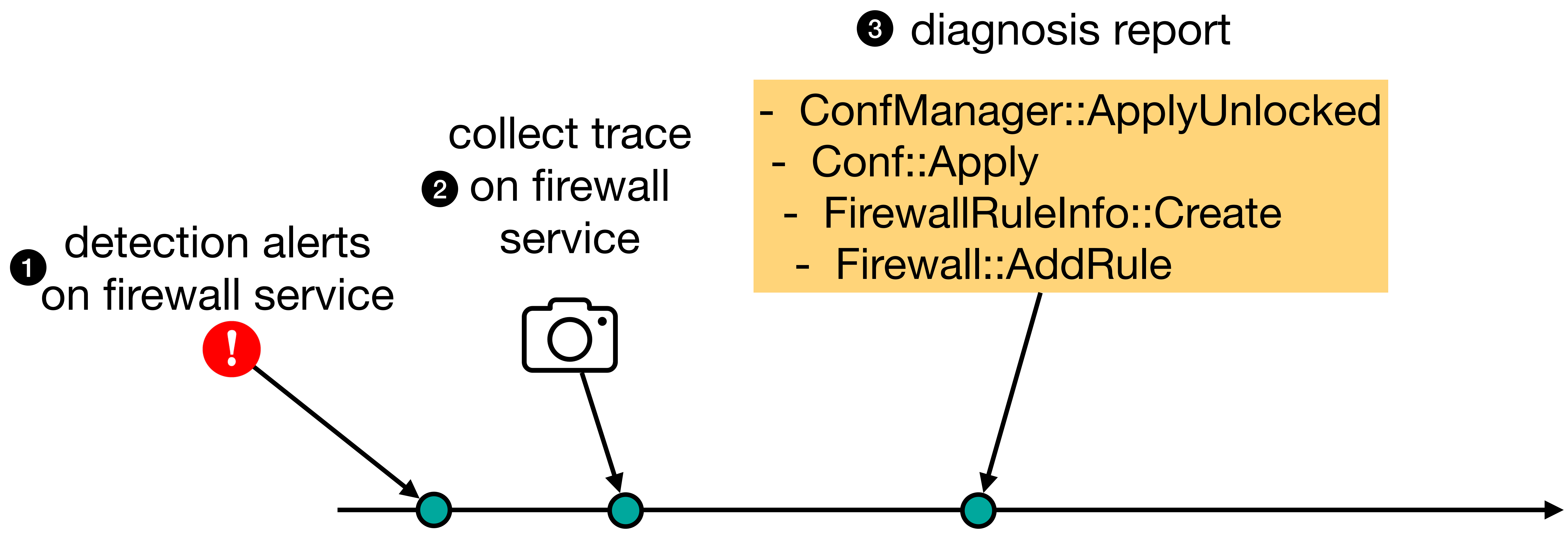
Diagnosis for the earlier firewall example

Diagnosis for the earlier firewall example

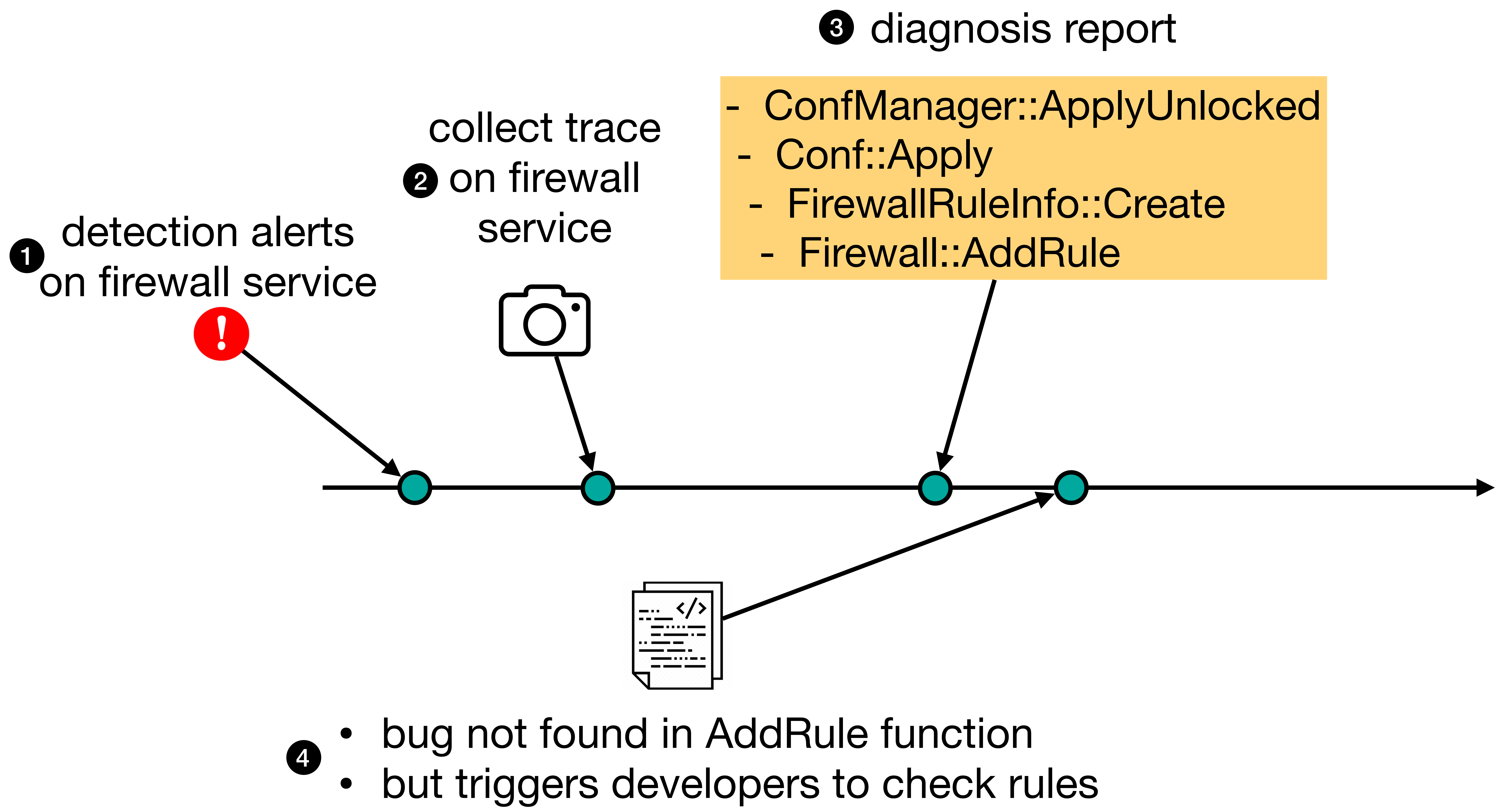
Diagnosis for the earlier firewall example



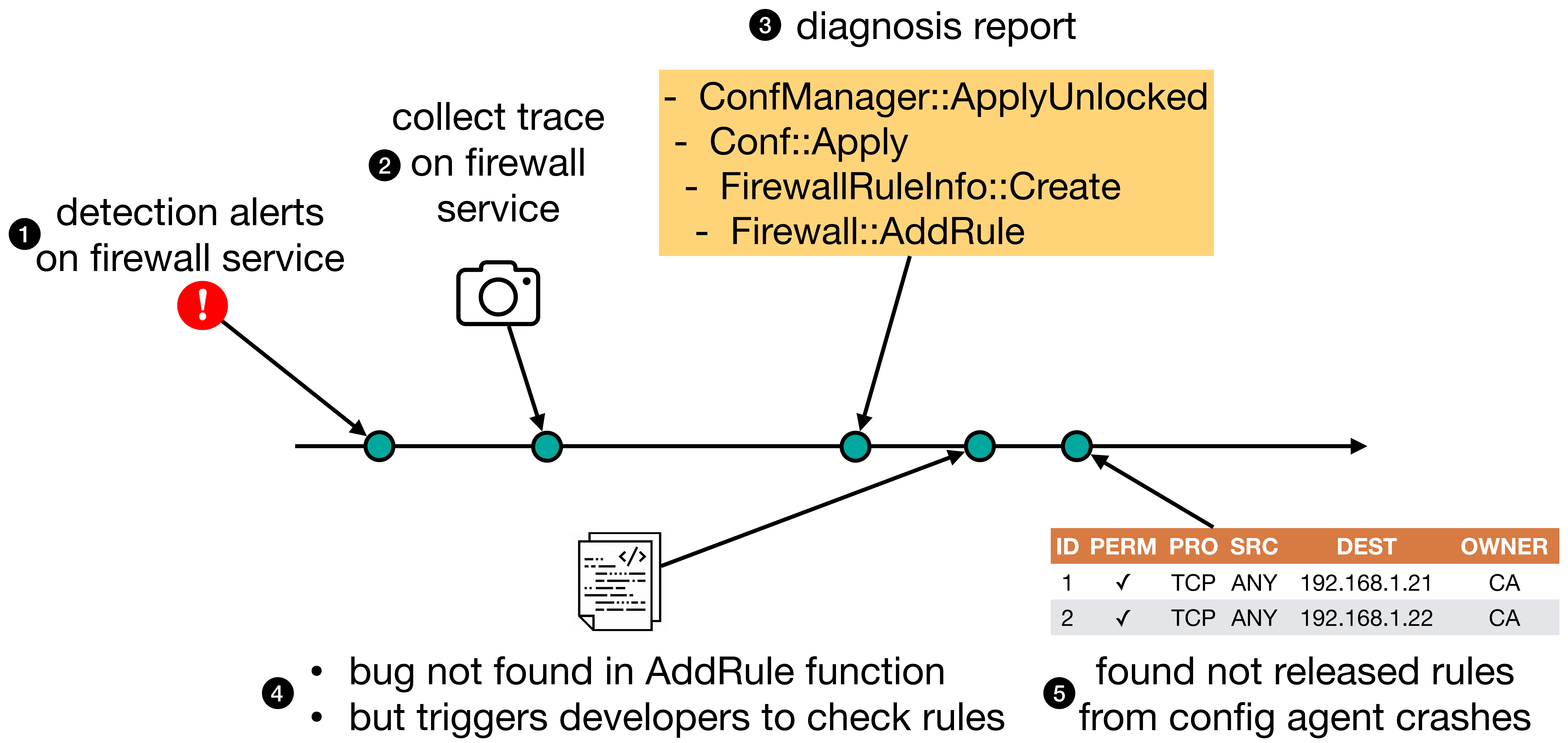
Diagnosis for the earlier firewall example



Diagnosis for the earlier firewall example

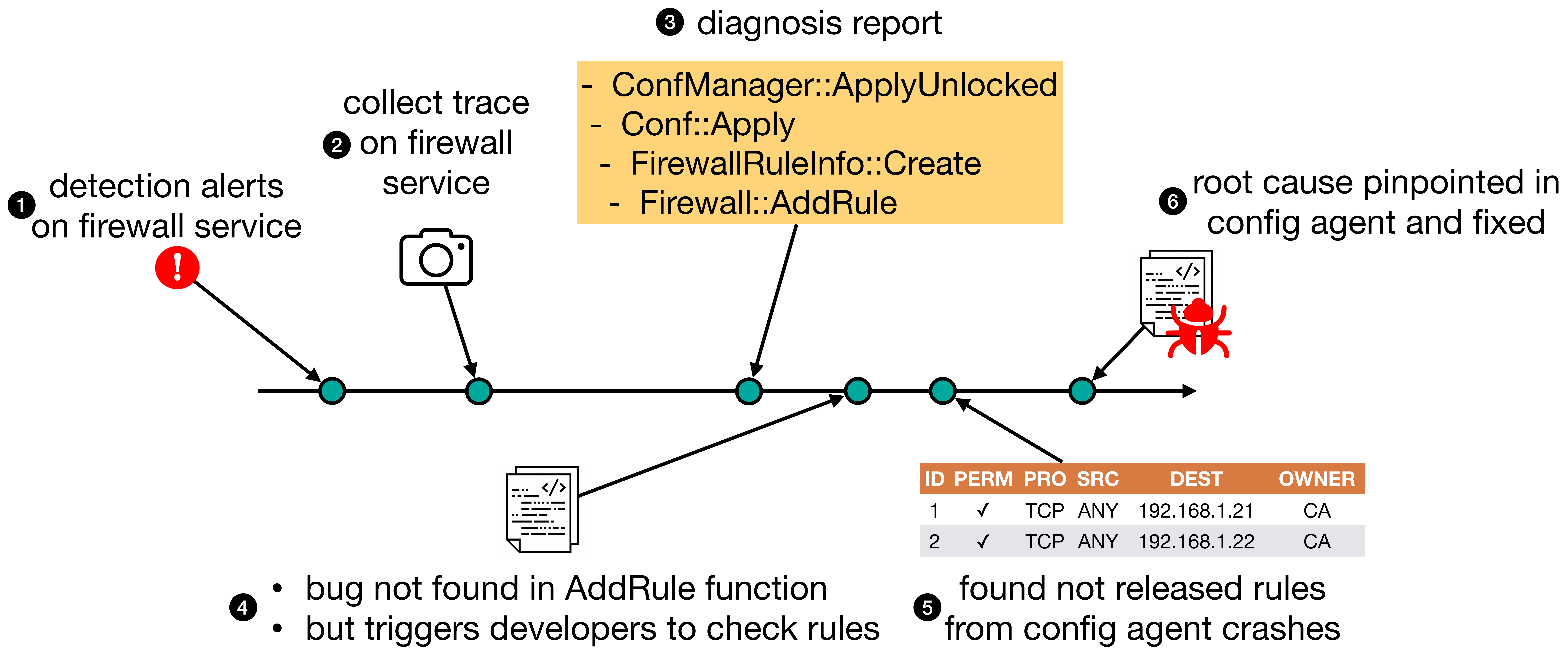


Diagnosis for the earlier firewall example



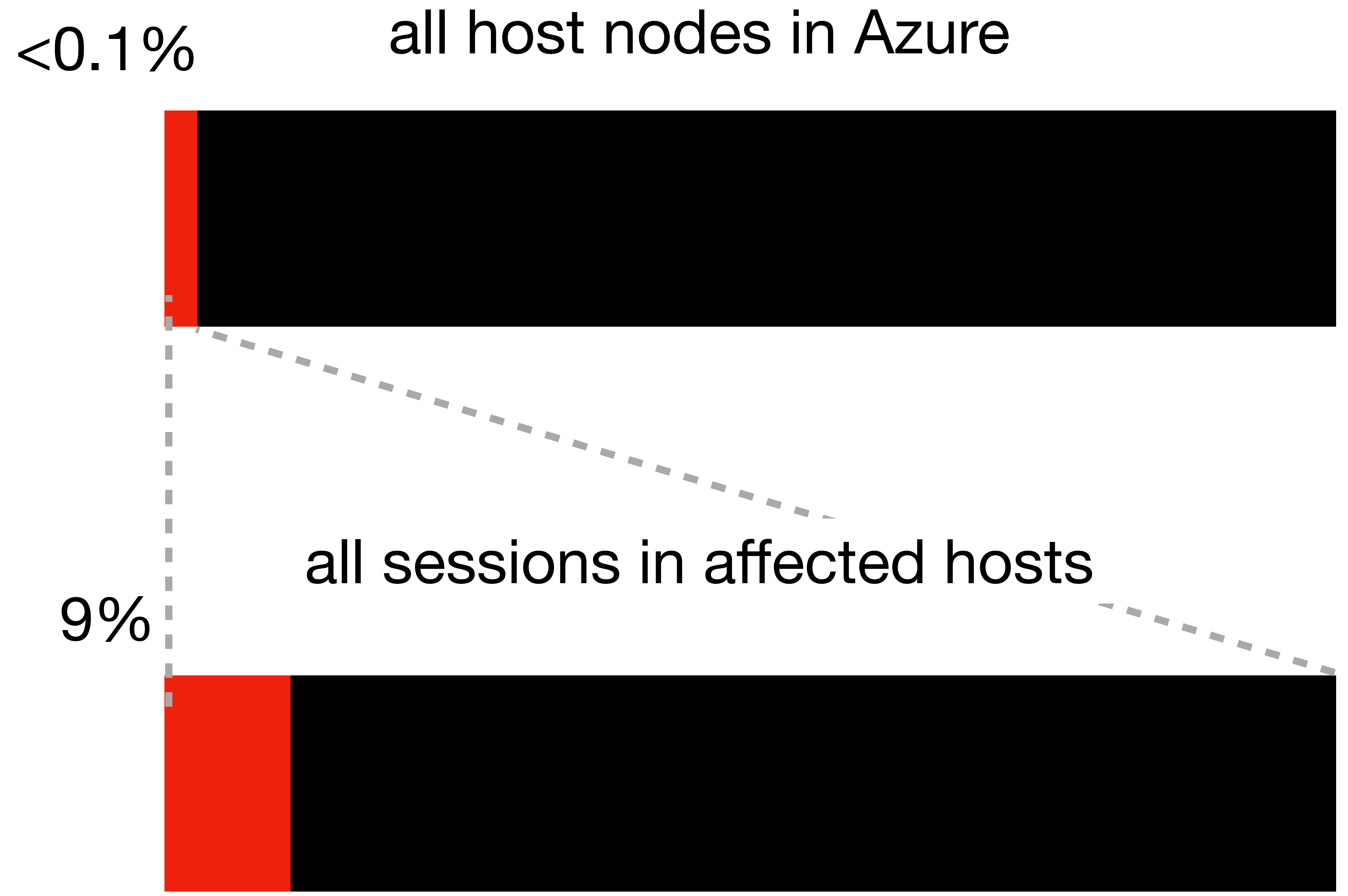
ID	PERM	PRO	SRC	DEST	OWNER
1	✓	TCP	ANY	192.168.1.21	CA
2	✓	TCP	ANY	192.168.1.22	CA

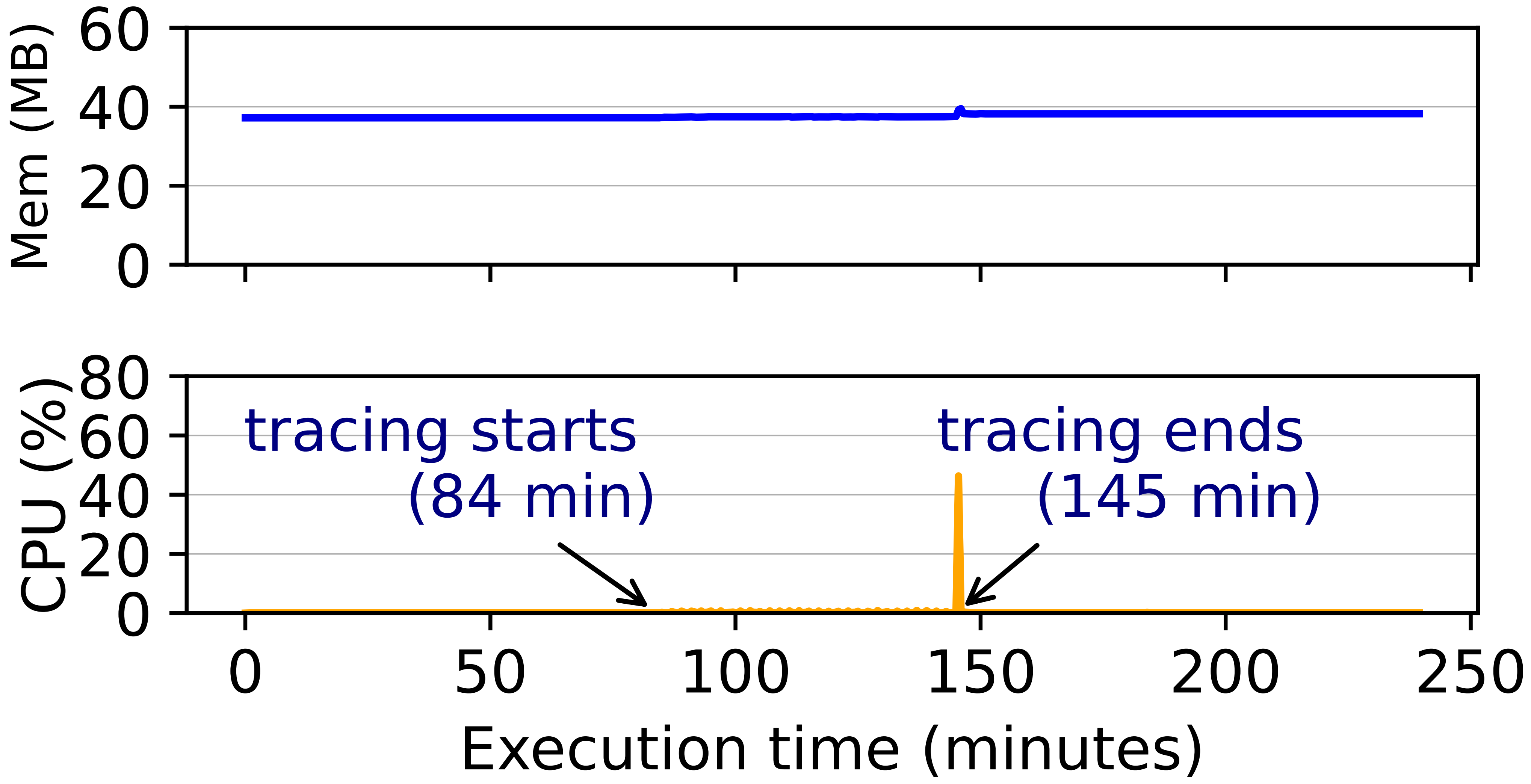
Diagnosis for the earlier firewall example



ID	PERM	PRO	SRC	DEST	OWNER
1	✓	TCP	ANY	192.168.1.21	CA
2	✓	TCP	ANY	192.168.1.22	CA

What is the overhead of trace collection?





Memory: + 1.93 MB

CPU: a spike lasting for seconds

End-to-end latency: +1 second (median)

Affected hosts: < 0.1% of all nodes

Affected sessions: < 9% on affected hosts

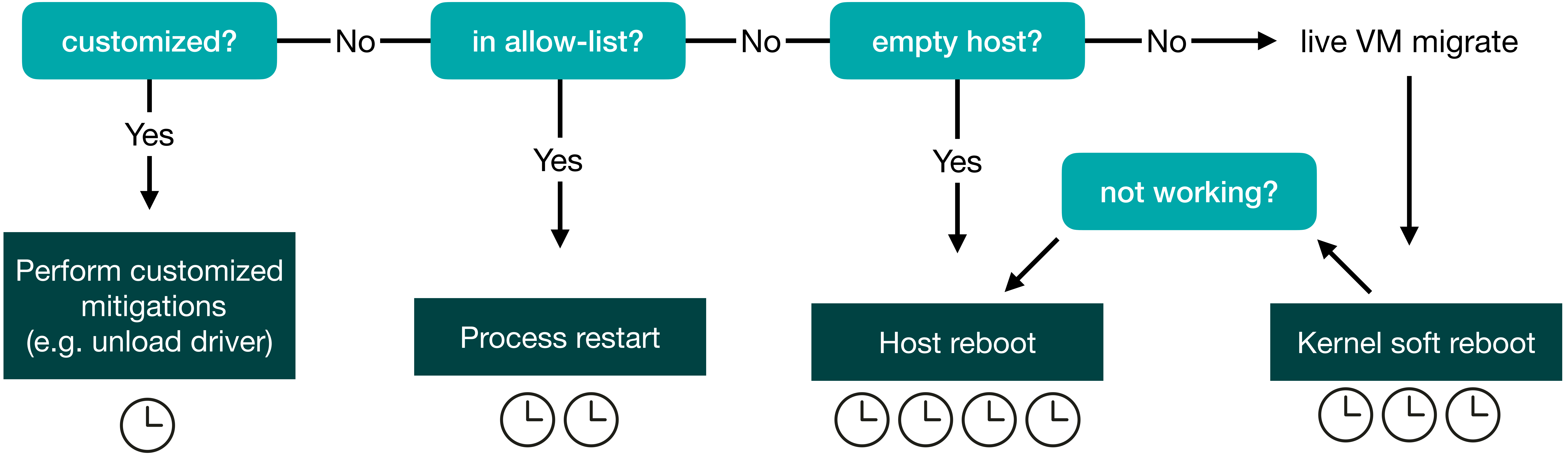
Conclusion

- Addressing memory leaks in cloud infrastructure is challenging
- RESIN, an end-to-end memory leak solution in production
  - divide-and-conquer to decompose the problem
  - multi-level solution with novel algorithms
- Running in Azure for more than 3 years
  - low-memory-induced VM reboots reduced 41×
  - new VM allocation errors reduced 10×

Backup slides

Decision tree based mitigation

- Goal: mitigate the memory leaks while **minimizing** the user impact



Mitigation duration