Peeking Behind the Curtains of Serverless Platforms

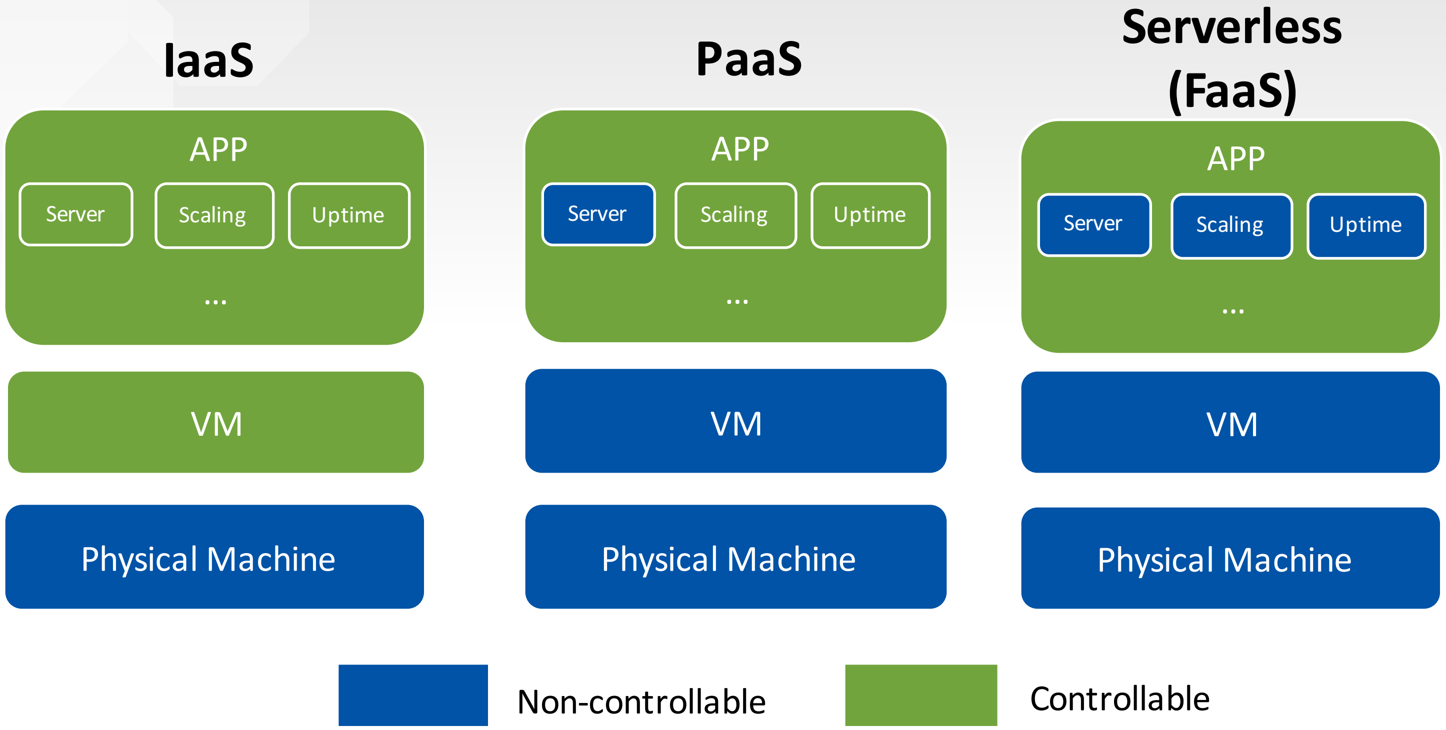
Liang Wang[1], Mengyuan Li[2], Yinqian Zhang[2],
Thomas Ristenpart[3], Michael Swift[1]
[1] UW-Madison, [2] The Ohio State University, [3] Cornell Tech

Department of Computer Sciences
UNIVERSITY OF WISCONSIN–MADISON

CORNELL TECH

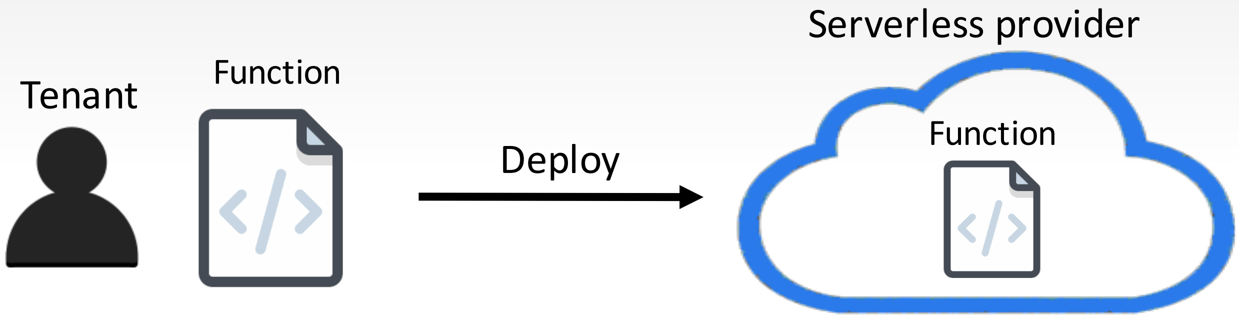
THE OHIO STATE UNIVERSITY

Providers do more, tenant do less



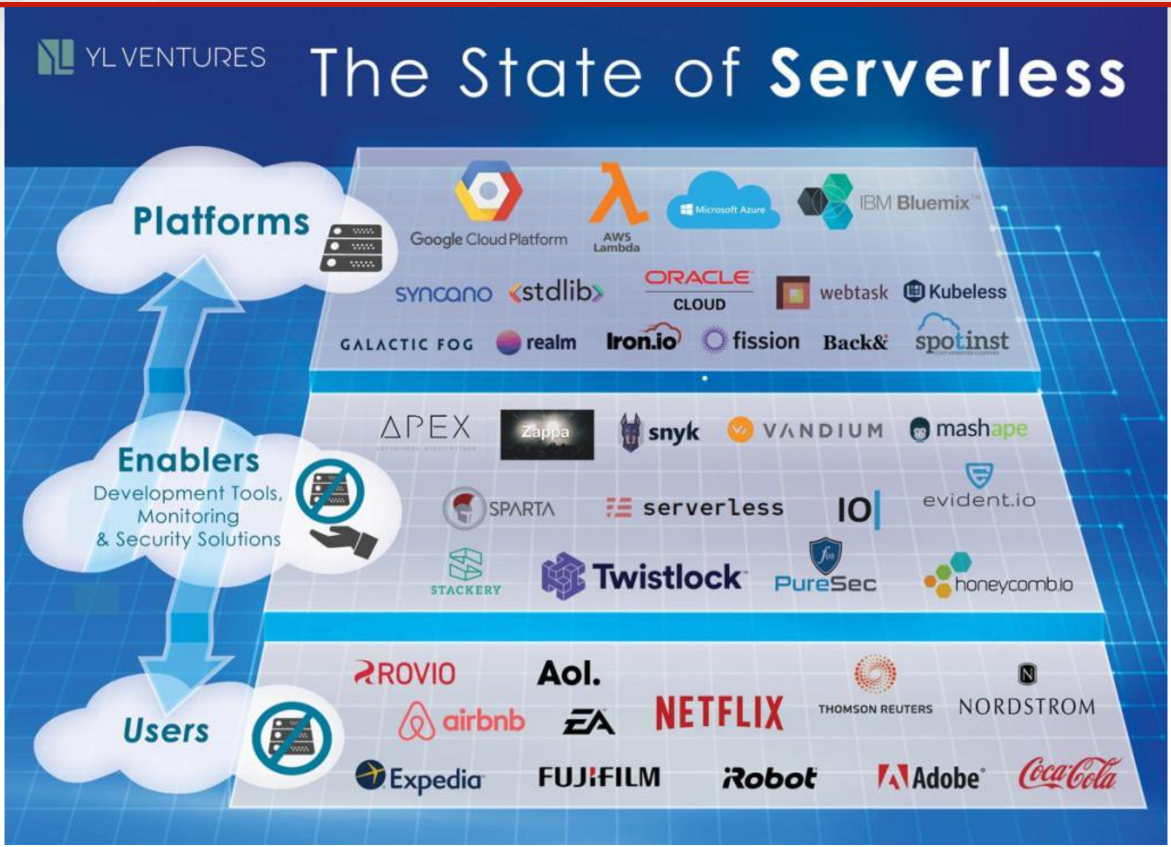
Benefits of serverless

Function: Standalone, small application dedicated to specific tasks



- Minimal configuration
- No efforts on server management
- Low cost

Serverless ecosystem

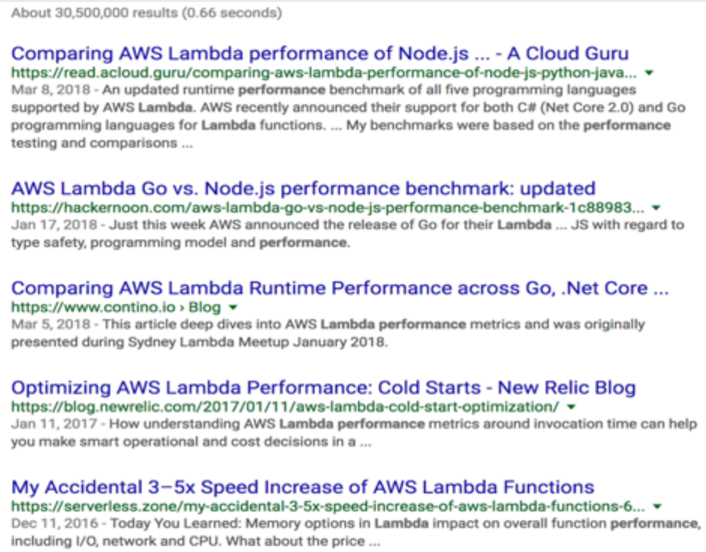


Source: https://venturebeat.com/2017/10/22/the-big-opportunities-in-serverless-computing/

Lots of questions about serverless

- Are applications resistant to DDos attacks in serverless?
- Are functions secure in serverless?
- Can serverless providers deliver guaranteed performance?

...



We need better methodology and more systematic measurement to answer these questions

Contributions

- In-depth study of resource management and performance isolation in

AWS Lambda Azure Functions Google Cloud Functions

- Identify opportunities to improve serverless platforms
    - AWS: Bad performance isolation, function consistency issue, …
    - Azure: Unpredictable performance, tenant isolation issues, …
    - Google: Resource accounting bug, ...

- Open-source measurement tool

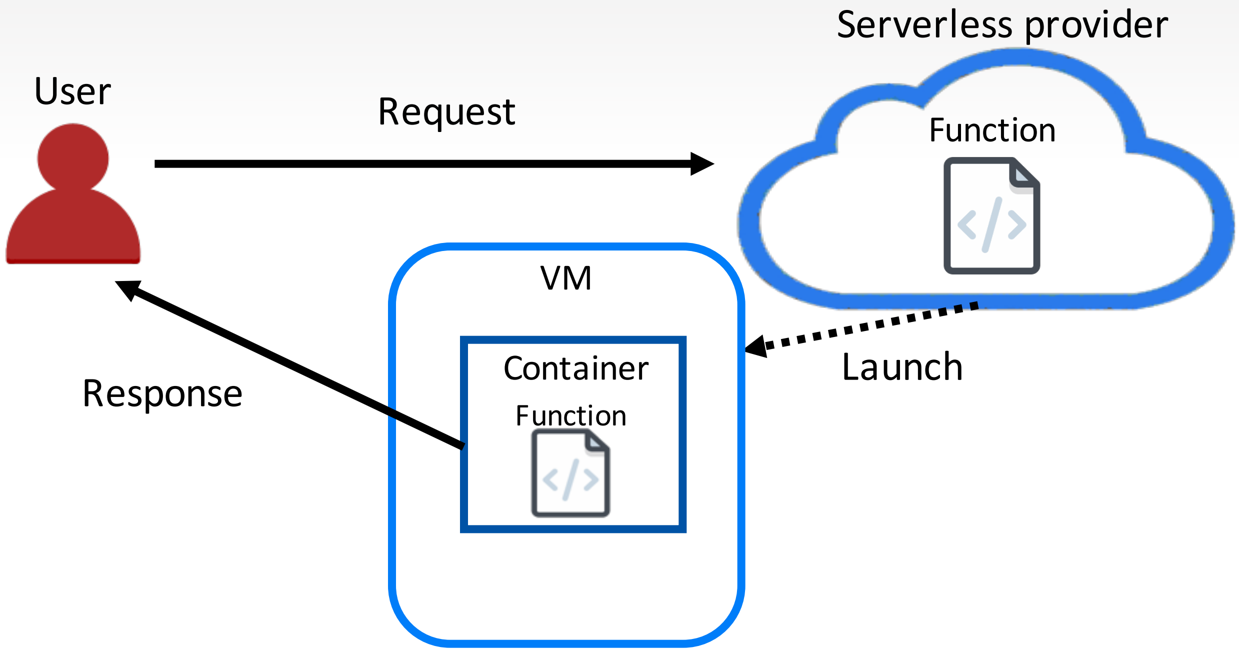
(https://github.com/liangw89/faas_measure)

Overview

- Background
- Methodology
- Highlighted results
    - Serverless architectures
    - Resource scheduling
    - Performance isolation
    - Bugs

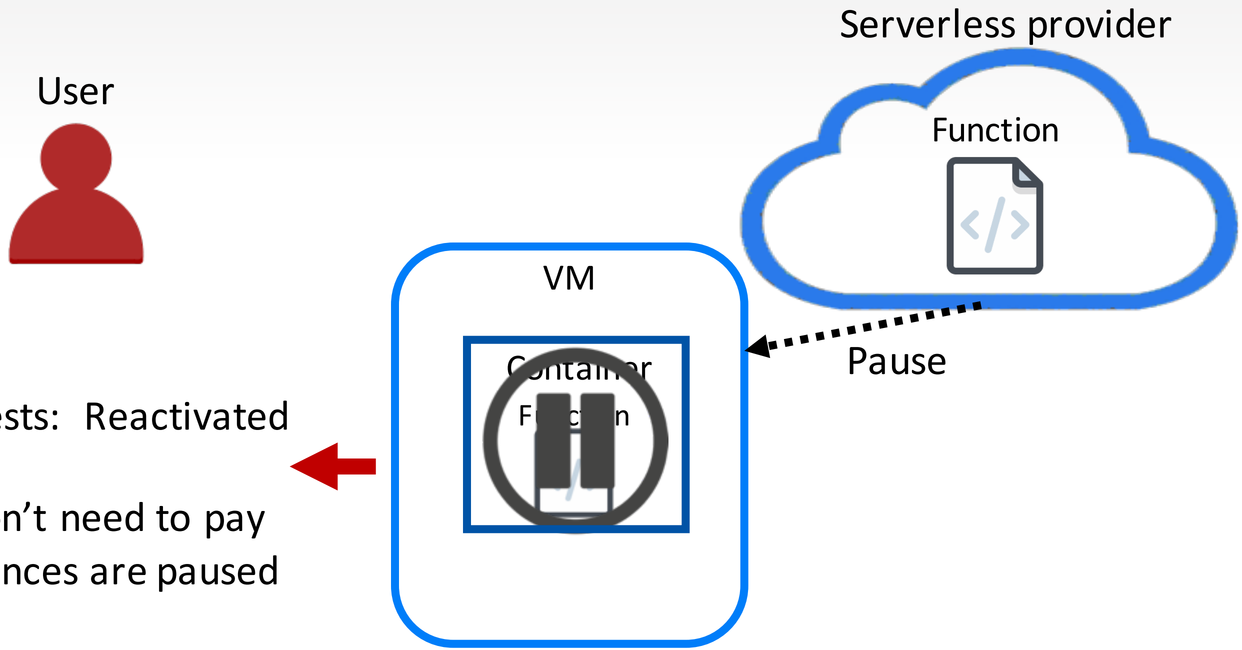
How serverless works

A function runs in a container (**function instance**) launched by the provider with limited CPU/memory/execution time



How serverless works

The function instance will be frozen after returning from invocation

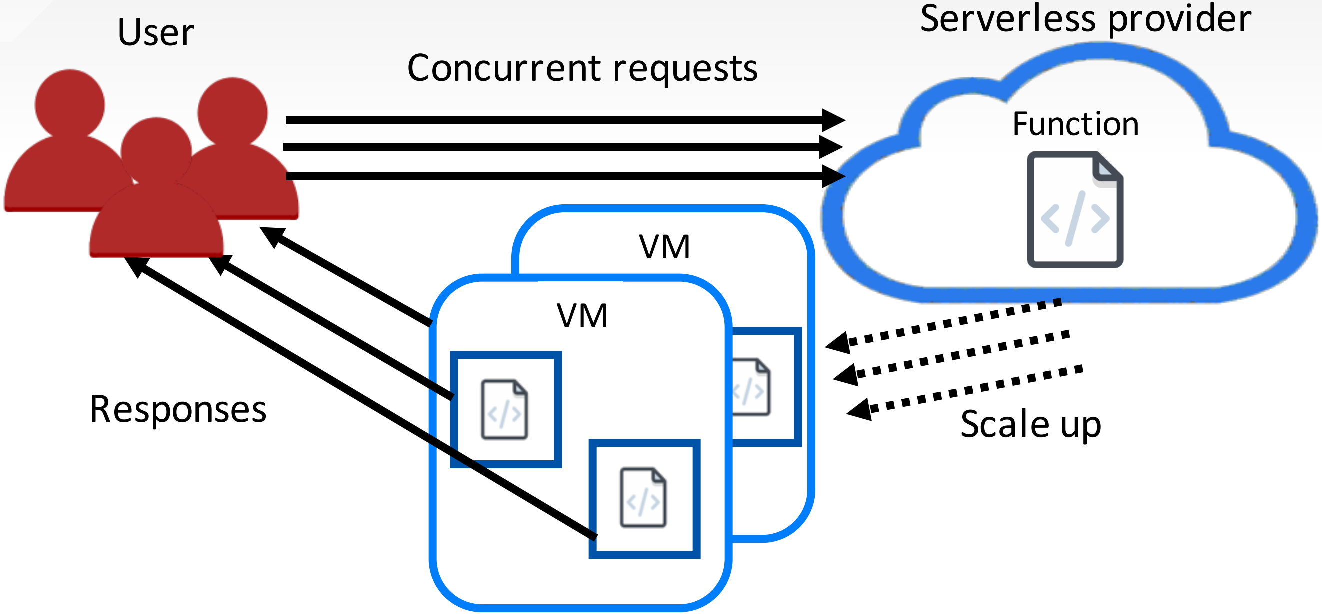


New requests: Reactivated

Tenants don’t need to pay while instances are paused

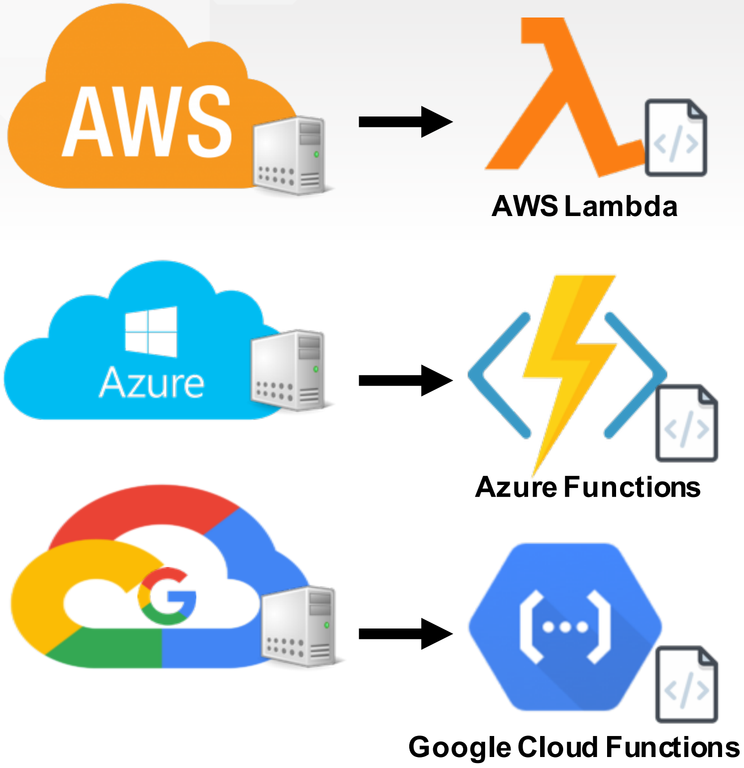
How serverless works

Providers manage backend infrastructures and resource for tenants



Methodology

Invoke measurement functions many times (50K+) under various settings from vantage points in the same cloud region



Measurement function

- Collect information via procfs/cmd/env
- Execute performance tests

Setting variables:

- Function memory
- Function language
- Request frequency
- Concurrent request

Time:

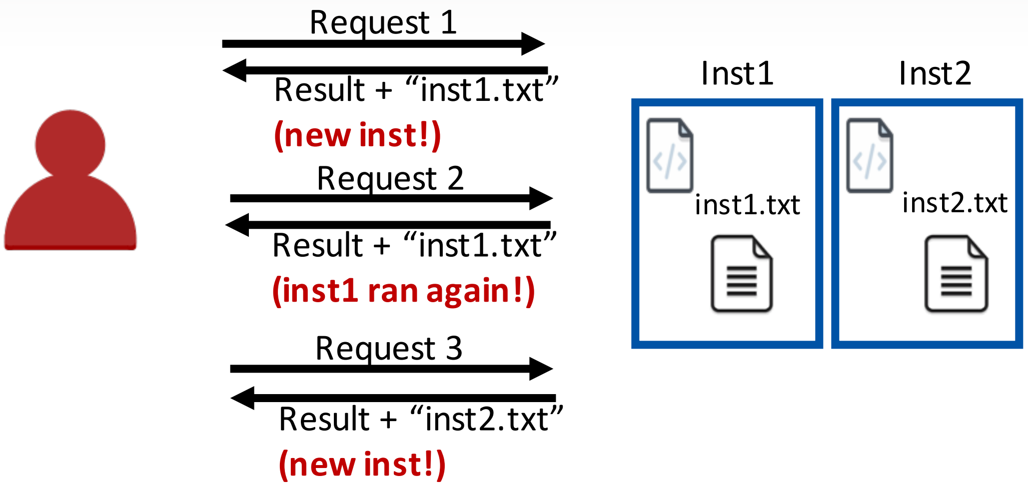
July–Dec 2017, May 2018

Tool 1: Map requests to instances

Which instance handled the request?

Instance identification:

Write a unique file on /tmp → persistent during instance lifetime



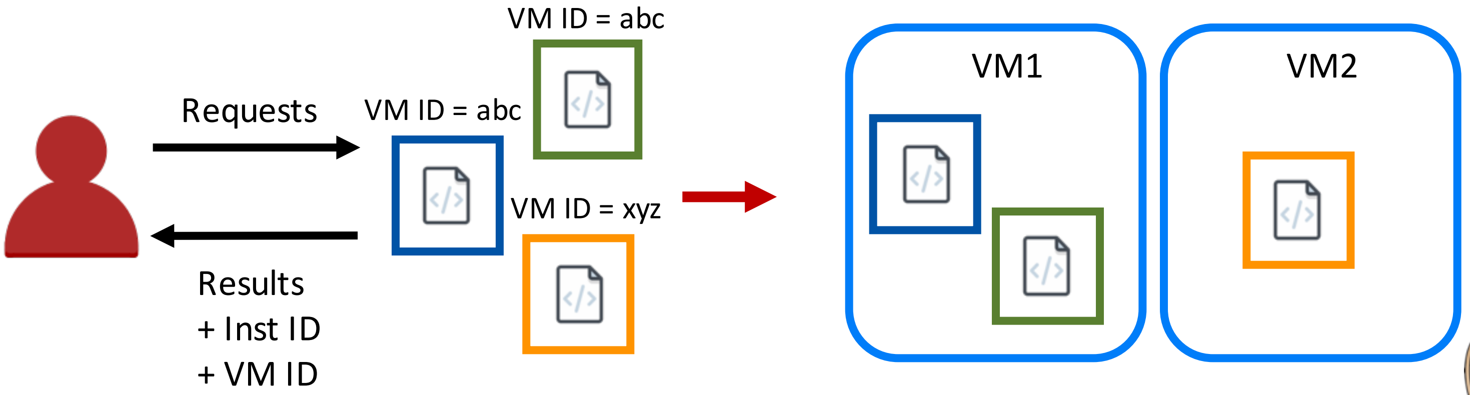
Tool 2: Map instances to VMs

Are instances on the same VM?

VM identification:

- **AWS**: An entry in the */proc/self/cgroup* `2:cpu:/sandbox-root-j88bAZ/:`
- **Azure**: The *WEBSITE_INSTANCE_ID* environment variable `WEBSITE_INSTANCE_ID:ae0f2957f1a770c2d97a 86ce1f2fb1e258e5a4cd25209618bef9e288c576675`
- **Google**: Unknown

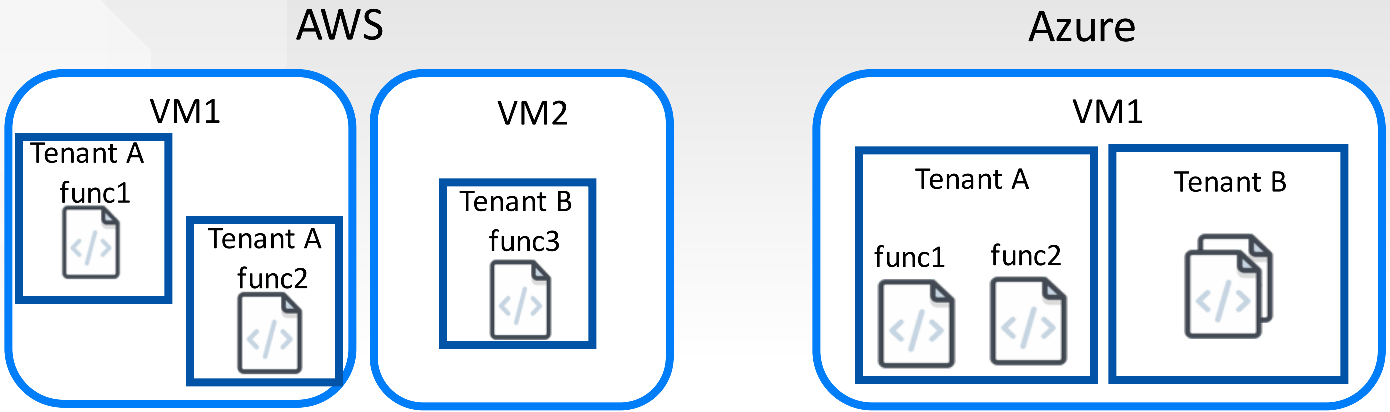
Verified via I/O-based and Flush-Reload coresidency tests



Highlighted results

- **Serverless architectures**
- Resource scheduling
- Performance isolation
- Bugs

Do multiple tenants' instances run on the same VM?



AWS: No → VM only hosts functions from single tenant

Azure:

- 2017: Yes → VM hosts functions from multiple tenants
- 2018: No. But other platforms still do this: **Spotinst, stdlib, webtask.io**

Google: Unknown

Cross-tenant VM sharing make applications vulnerable to side-channel attacks

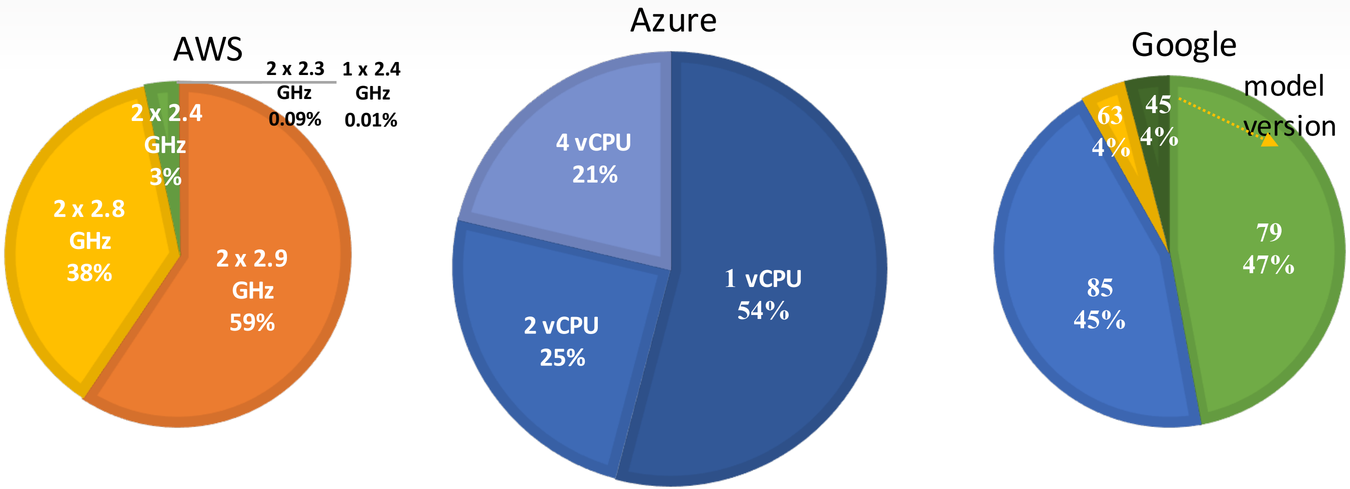
Do VMs have the same configurations?

Methodology: Examine procfs and env variables of the host VMs of 50 K function instances

AWS: 5 CPU configurations (1 or 2 vCPUs, 4 CPU models)

Azure: 9 configurations (1 or 2 or 4 vCPUs, 4 CPU models)

Google: 4 configurations (4 CPU models)



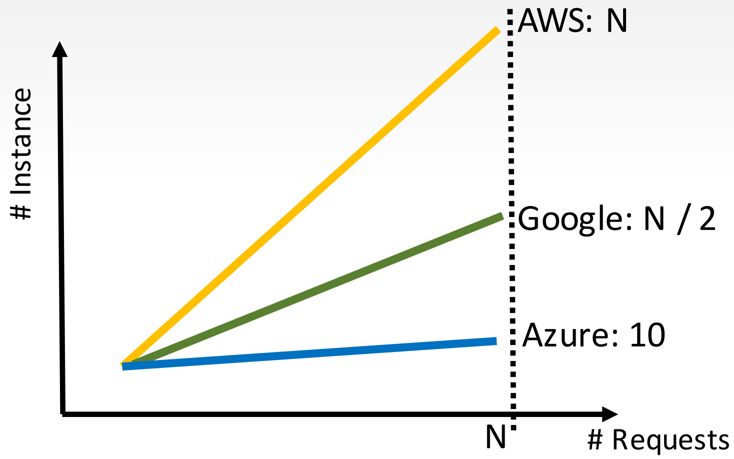
Different types of VMs could result in different instance performance

Highlighted results

- Serverless architectures
- **Resource scheduling**
- Performance isolation
- Bugs

Can the platforms effectively handle concurrent requests?

Methodology: send N concurrent requests and examine the number of instances running concurrently



Azure/Google: Don't deliver promised scalability

How long does it take to launch an instance?

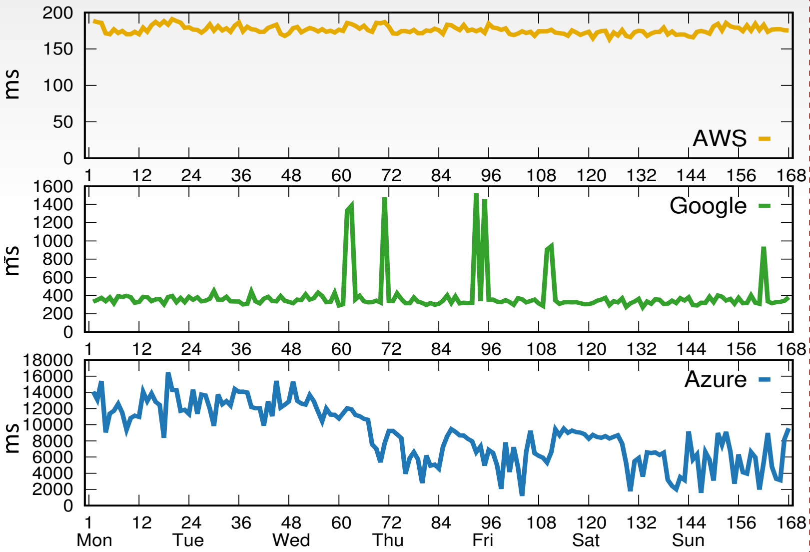
Median coldstart latency of 1000 instances

AWS: 160 ms

Google: 500 ms (2017)
→ 2000 ms (2018)

Azure: 3600 ms (2017)
→ 300 ms (2018)

Median coldstart latency per hour over 7 days (2017)



Coldstart might affect tail latencies

Highlighted results

- Serverless architectures
- Resource scheduling
- **Performance isolation**
- Bugs

What can affect performance?

- **CPU share**: fraction of 1000-ms time period for which the instance can use CPU
- **IO throughput**: Write 512 KB of data to the local disk 1,000 times (via dd or scripts)
- **Network throughput**: Use iperf3 to run the throughput test for 10 seconds

Factors affecting performance:

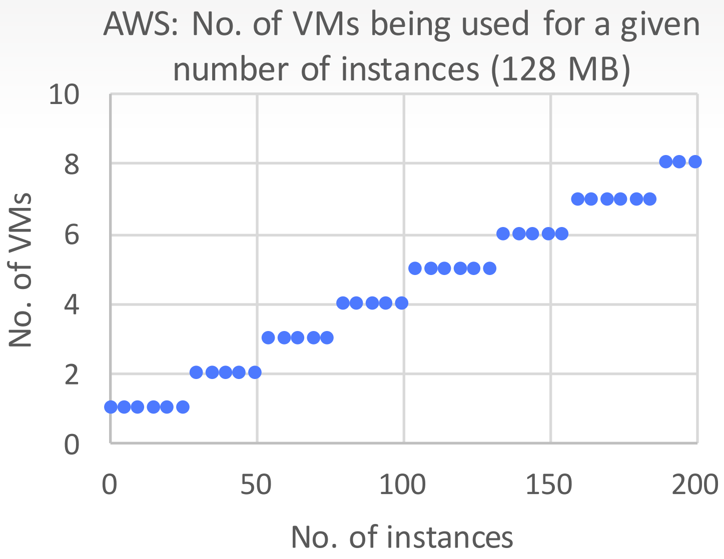
	AWS	Azure	Google
Coresidency	**Yes**	**Yes**	Unknown
VM configuration	No	Yes	No

How instances are placed on VMs

AWS: Bin-packing; use at most **3328** MB VM memory

Azure: Random

Google: Unknown

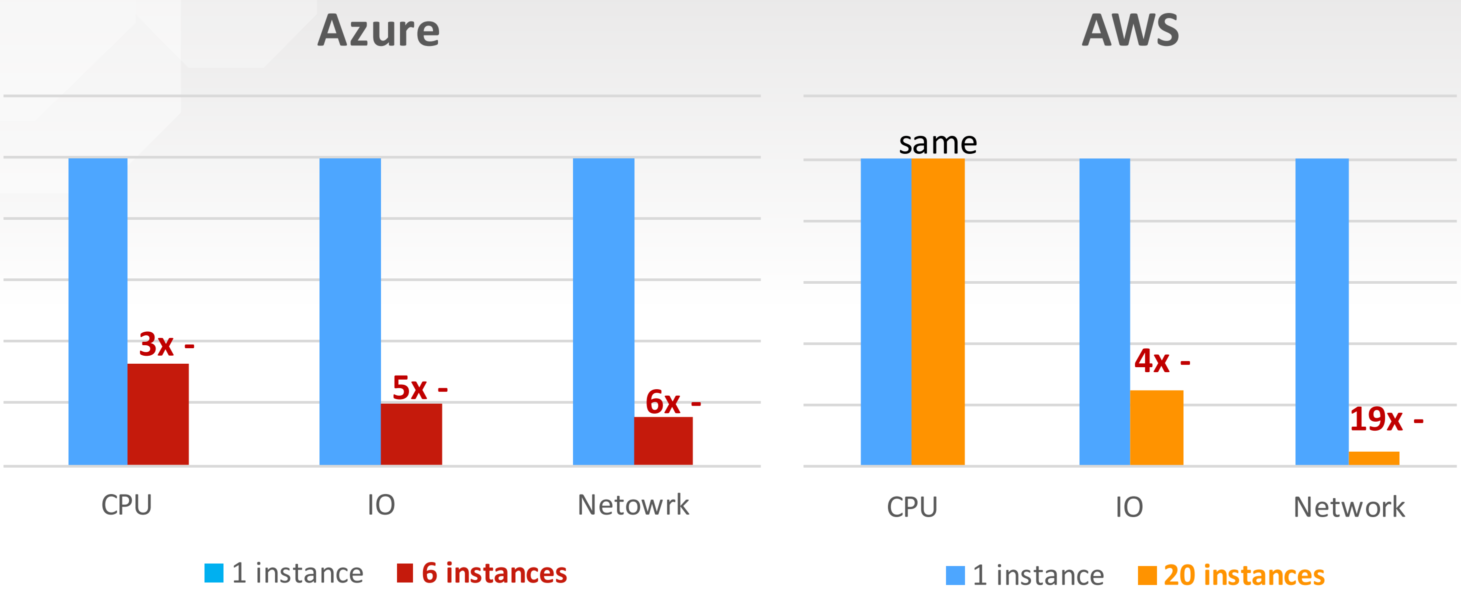


25 * 128 MB insts: 1 VM
50 * 128 MB insts: 2 VMs
...
200 * 128 MB insts: 8 VMs

AWS Lambda VM memory utilization: 85-100%

AWS: Easy for instances from the same tenant to be coresident

Coresident instances contend for VM resources

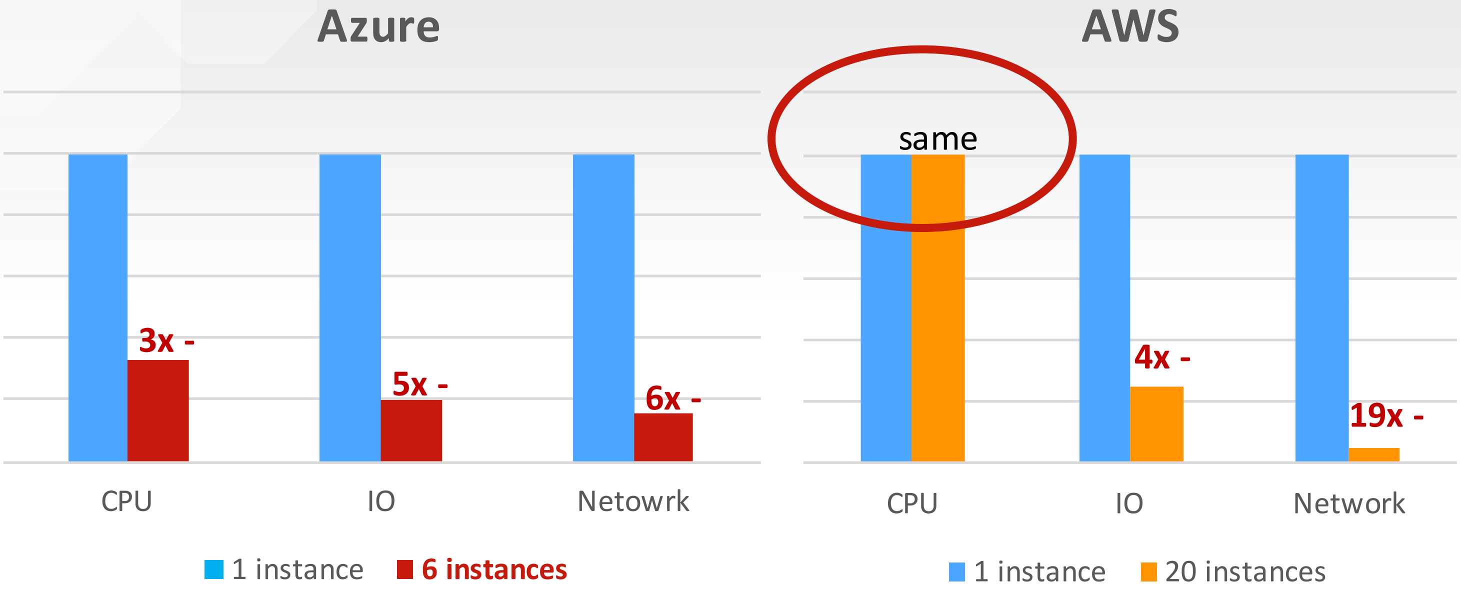


(Estimated based on the median performance across coresident instances, over 50 rounds)

Resources are allocated per VM

More co-residency decreases resources per function

Coresident instances contend for VM resources



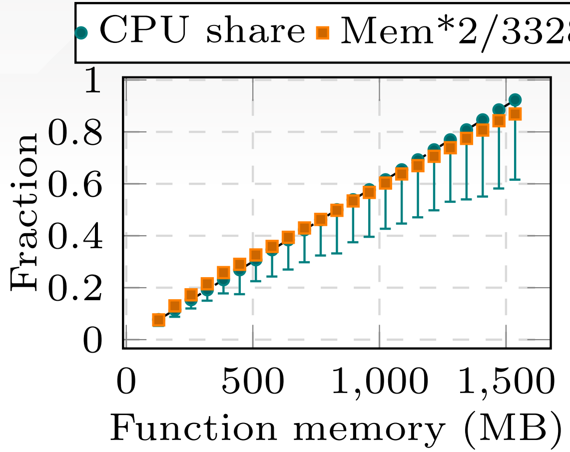
(Estimated based on the median performance across coresident instances, over 50 rounds)

Resources are allocated per VM

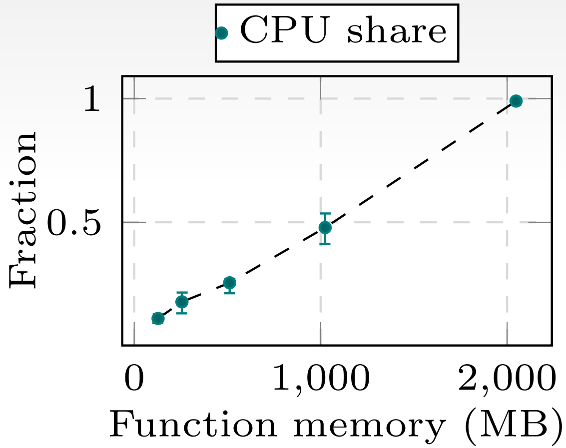
More co-residency decreases resources per function

AWS/Google: CPU share is proportional to memory

AWS: Functions of 128 MB memory can use CPU for 80 ms in 1000 ms
Functions of 1.5 GB memory can use CPU for 900 ms in 1000 ms



AWS



Google

More memory --> More CPU --> Better performance

What can affect performance?

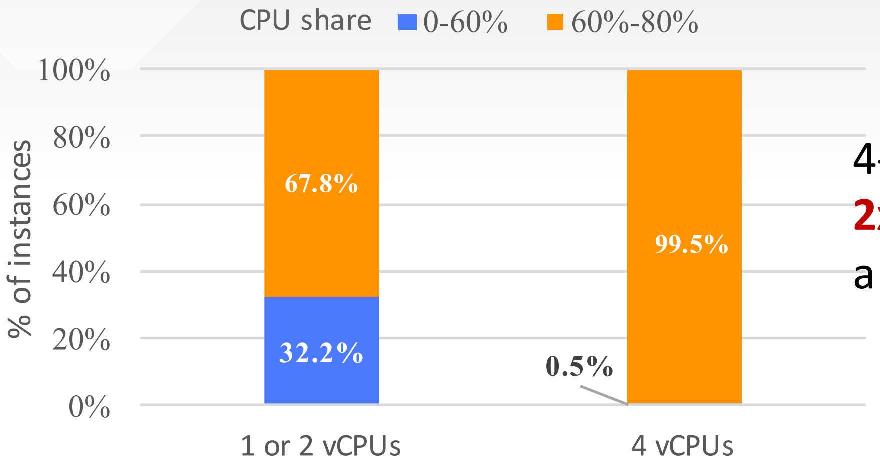
- **CPU share**: fraction of 1000-ms time period for which the instance can use CPU
- **IO throughput**: Write 512 KB of data to the local disk 1,000 times (via dd or scripts)
- **Network throughput**: Use iperf3 to run the throughput test for 10 seconds

Factors affecting performance:

	AWS	Azure	Google
Coresidency	Yes	Yes	Unknown
VM configuration	No	**Yes**	No

Azure: VM configurations affect performance

Azure:



4-vCPU VMs get **1.5x** IO throughput,
2x network throughput,
and more CPU than other types of VMs

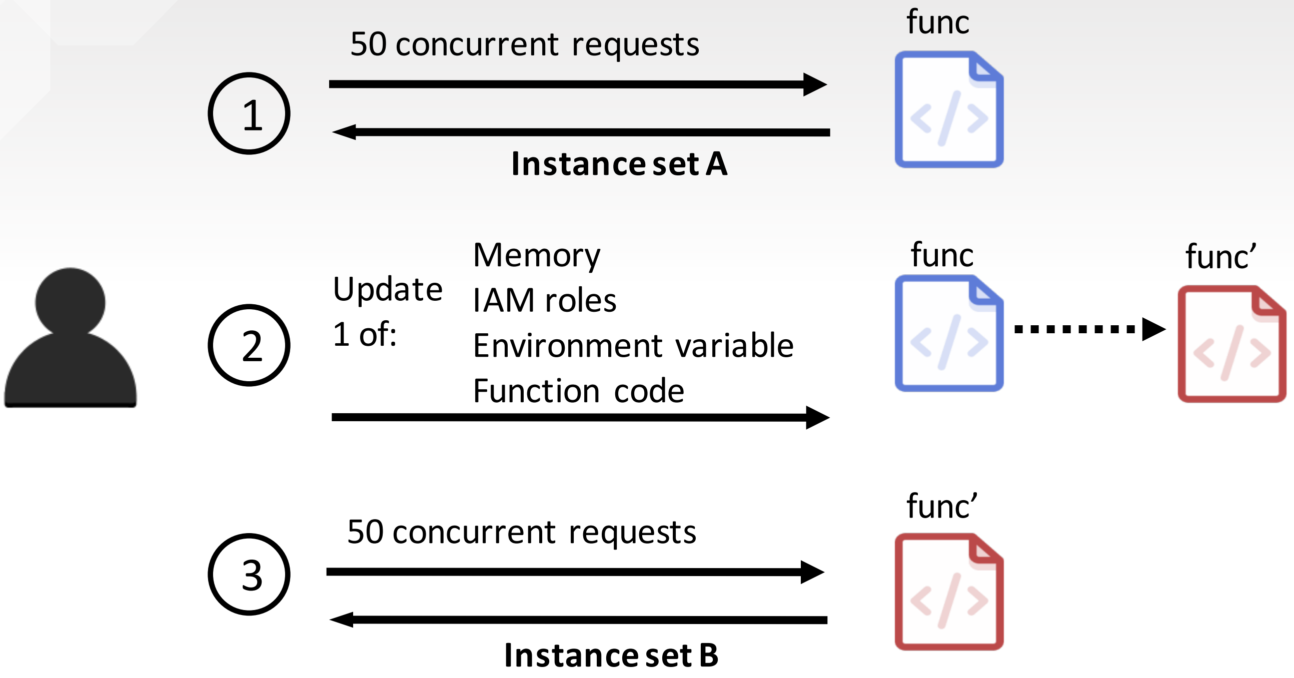
Same function + fewer resources
= longer running time = more money

Highlighted results

- Serverless architectures
- Resource scheduling
- Performance isolation
- **Bugs**

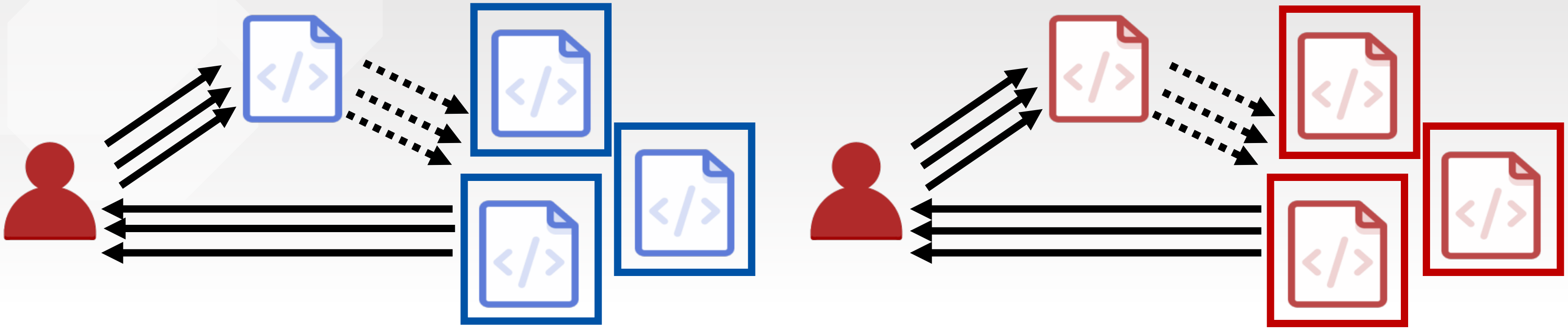
Can AWS propagate function updates correctly?

Methodology:



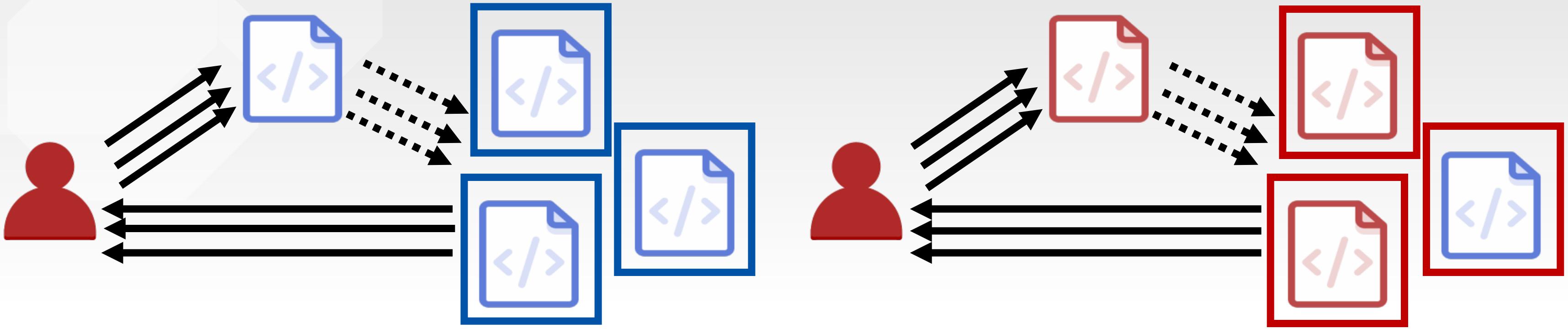
Did any instances in set B run func instead of func’?

AWS: Inconsistent function usage



3.8% (out of 20K) ran an inconsistent or outdated function

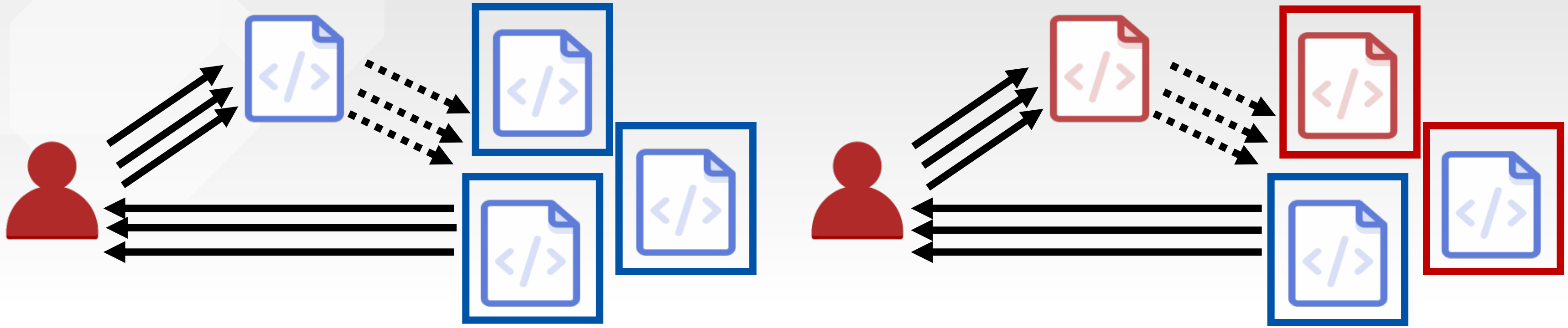
AWS: Inconsistent function usage



3.8% (out of 20K) ran an inconsistent or outdated function

- Case 1: New instances ran outdated functions (0.1%)

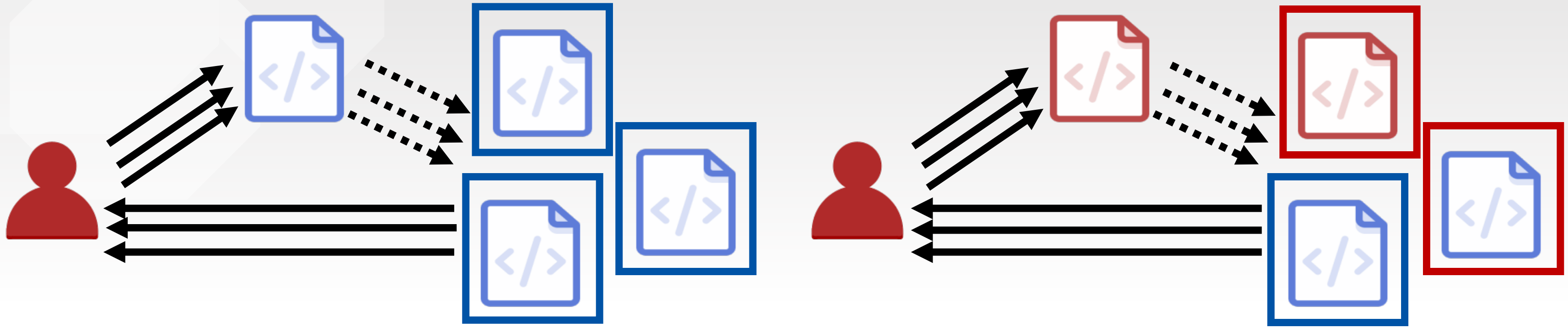
AWS: Inconsistent function usage



3.8% (out of 20K) ran an inconsistent or outdated function

- Case 1: New instances ran outdated functions (0.1%)
- Case 2: Requests handled by the instances for outdated functions (3.7%)

AWS: Inconsistent function usage



3.8% (out of 20K) ran an inconsistent or outdated function

- Case 1: New instances ran outdated functions (0.1%)
- Case 2: Requests handled by the instances for outdated functions (3.7%)

Inconsistent responses to users

Google: Stealthy background process

Processes can run after function invocation concluded

Method:

Nodejs will execute line B without waiting for user_task returns

```
exports.handler = function handler(req, res) {
            // run asynchronous task here.
line A:     user_task();
            // send back results.
line B:     res.status(http_code).send(user_data);
}
```

- Processes can stay alive for to 21 hours
- No billing → **Use extra resources for free!**

Google: Stealthy background process

Processes can run after function invocation concluded

Method:

Nodejs will execute line B without waiting for user_task returns

```
exports.handler = function handler(req, res) {
            // run asynchronous task here.
line A:     user_task();
            // send back results.
line B:     res.status(http_code).send(user_data);
}
```

Google should monitor the resource usage of the entire function instance rather than the Nodejs processes

Summary

- In-depth measurement study that discover various issues in three serverless computing platforms
    - Unpredictable performance
    - Bad performance isolation
    - Consistency issues

- Performance baselines and design considerations for future design of serverless platforms

- Responsible disclosure