Allocation Priority Policies for Serverless Function-execution Scheduling Optimisation

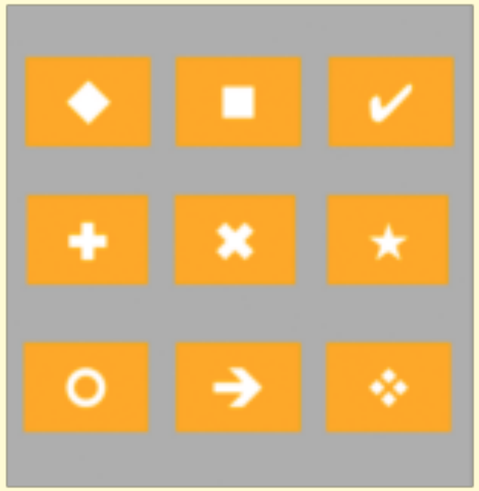
Giuseppe de Palma[1], Saverio Giallorenzo[1,2], and Jacopo Mauro[3]

[1]Università di Bologna, (IT) [2]INRIA, (FR) [3]University of Southern Denmark (DK)

Serverless (and Microservices)

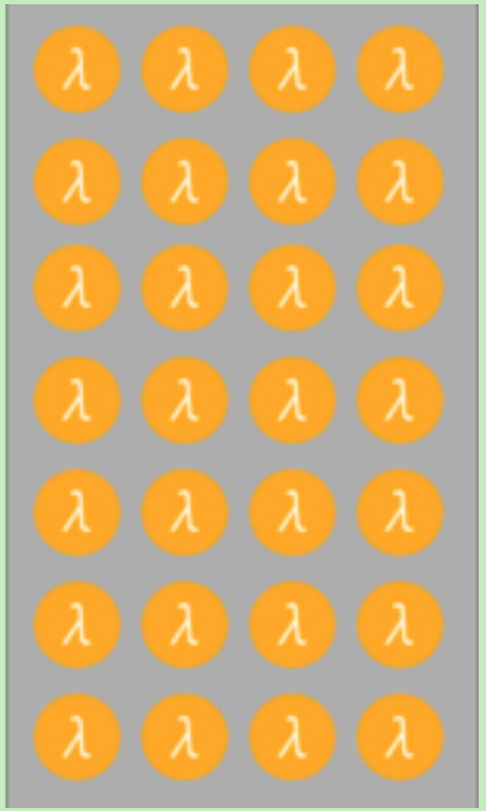
provisioned, pay-per-deployment

Monolith



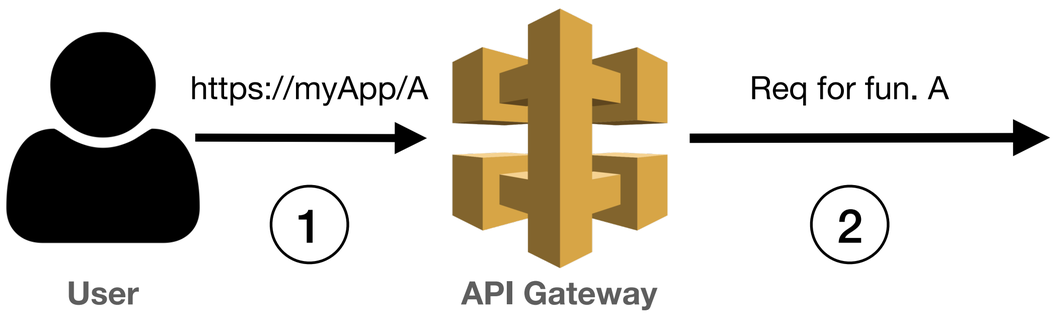
Microservices

on-demand, pay-per-execution



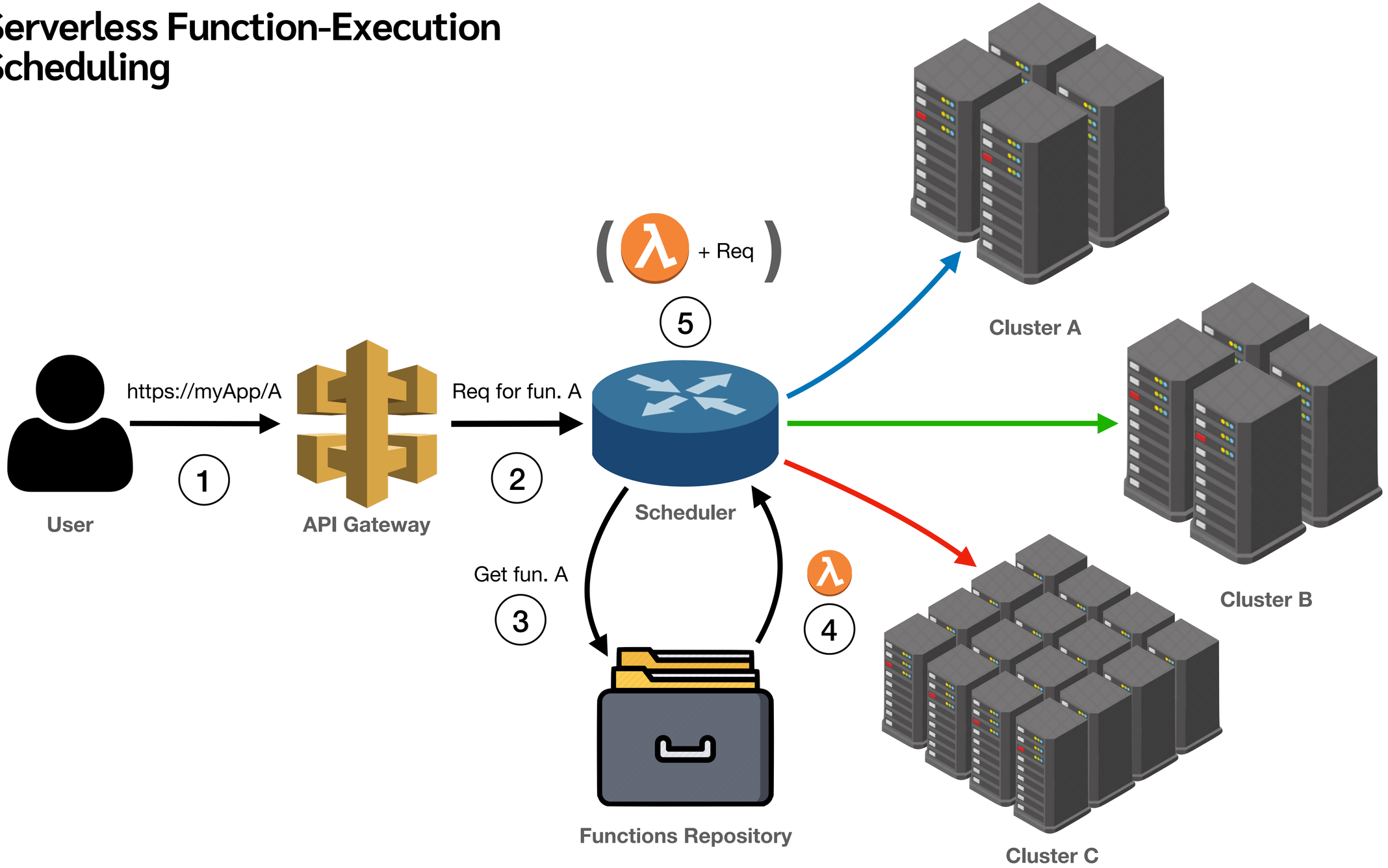
Serverless

Serverless != CGI

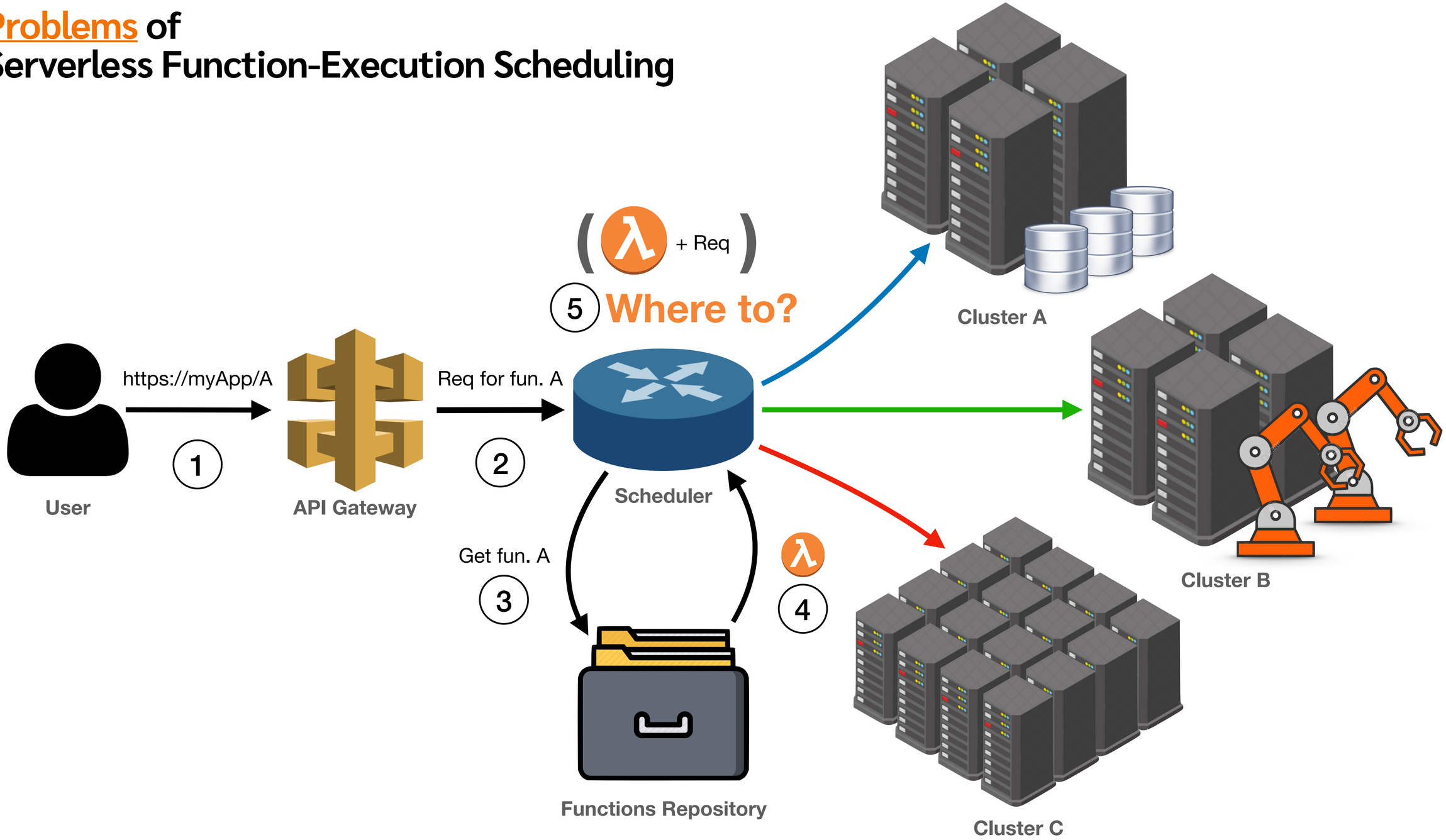



```
# A serverless cgi-bin!
# https://www.hawksworx.com/cgi-bin/hello/friend
[[redirects]]
  from = "/cgi-bin/hello/:name"
  to = "/.netlify/functions/hello?name=:name"
  status = 200
```

Serverless Function-Execution Scheduling

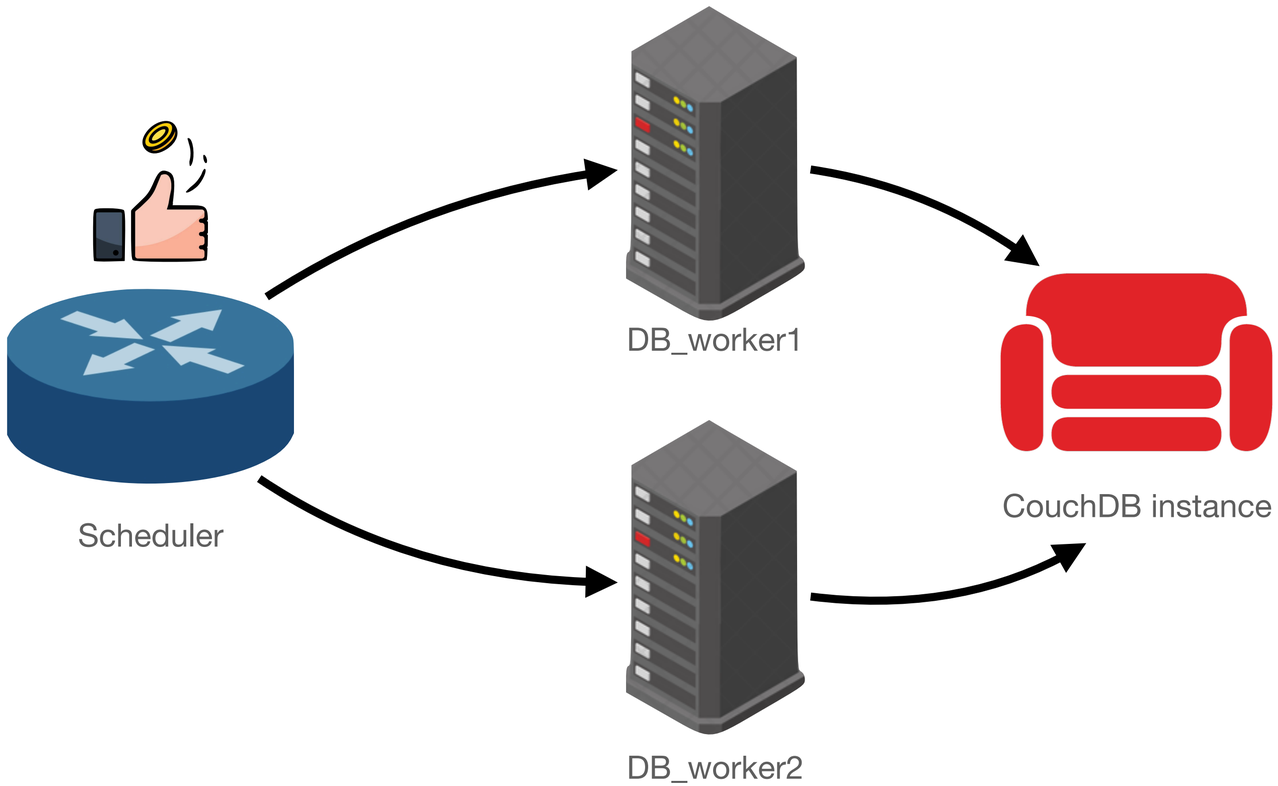


Problems of Serverless Function-Execution Scheduling



The APP Language • First Example

```
couchdb_query:
  - workers:
    - DB_worker1
    - DB_worker2
  strategy: random
  invalidate: ↩
    capacity_used: 50%
  followup: fail
```



The APP Language • Syntax

$$policy\_tag \in Identifiers \cup \{\texttt{default}\} \qquad worker\_label \in Identifiers \qquad n \in \mathbb{N}$$

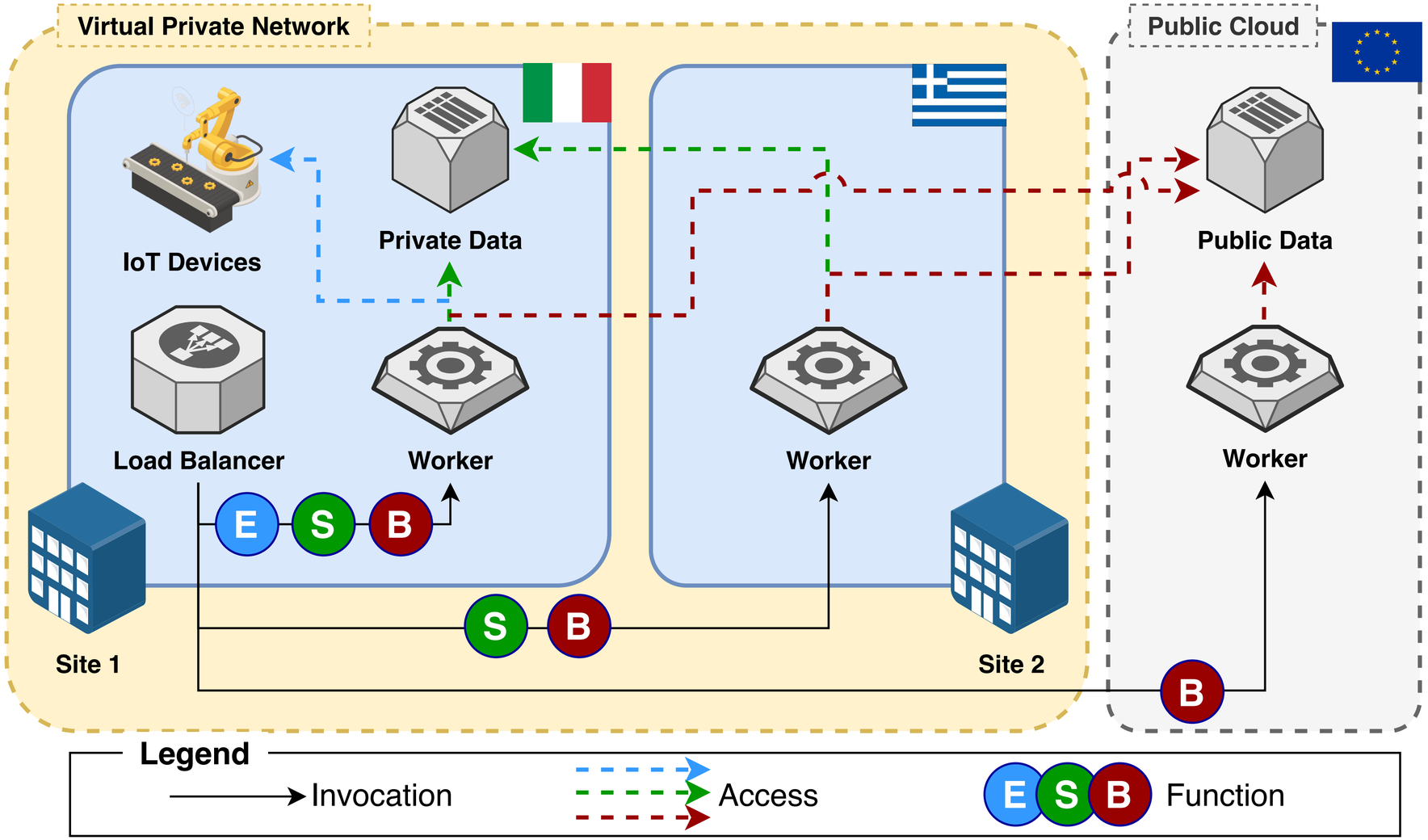
$$app ::= \overline{tag}$$

$$tag ::= policy\_tag : \overline{\texttt{-}\ block}\ followup?$$

$$block ::= \texttt{workers}\ [\ \texttt{"*"} \mid \overline{\texttt{-}\ worker\_label}\ ]\ (\texttt{strategy}\ [\ \texttt{random} \mid \texttt{platform} \mid \texttt{best\_first}\ ])?\ (\texttt{invalidate}\ [\ \texttt{capacity\_used} : n\% \mid \texttt{max\_concurrent\_invocations} : n \mid \texttt{overload}\ ])?$$

$$followup ::= \texttt{followup} : [\ \texttt{default} \mid \texttt{fail}\ ]$$

Use case

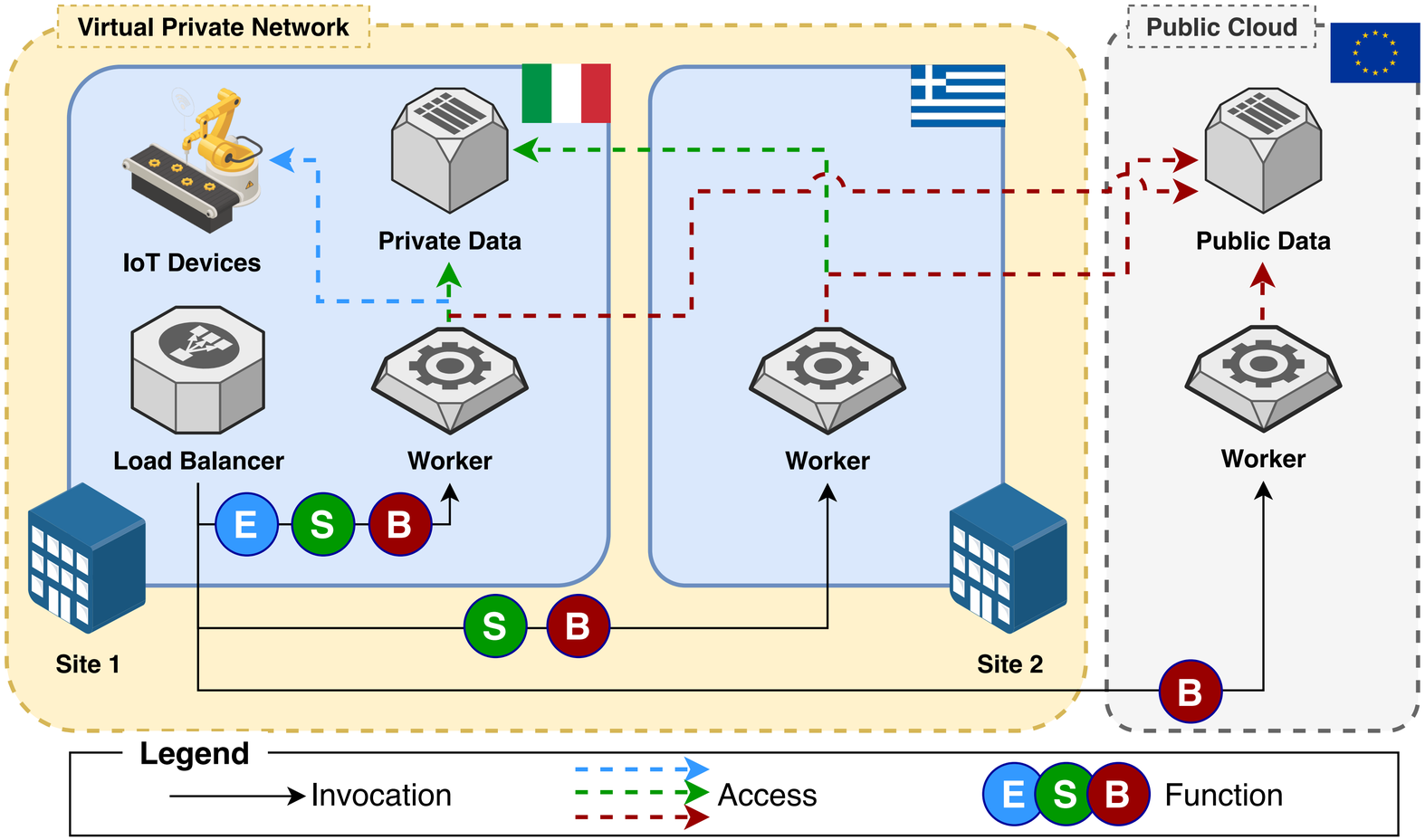

Virtual Private Network
Public Cloud
IoT Devices
Private Data
Public Data
Load Balancer
Worker
Worker
Worker
E
S
B
S
B
B
Site 1
Site 2
Legend
Invocation
Access
E S B Function

Use case - the APP deployment

```
Function_E:
  - workers:
    - worker_site1
  followup: fail

Function_S:
  - workers:
    - worker_site2
    - worker_site1
    strategy: random
  followup: fail

Function_B:
  - workers:
    - worker_public_cloud
    - worker_site2
    - worker_site1
  strategy: best_first
  followup: fail
```



Use case - empirical results

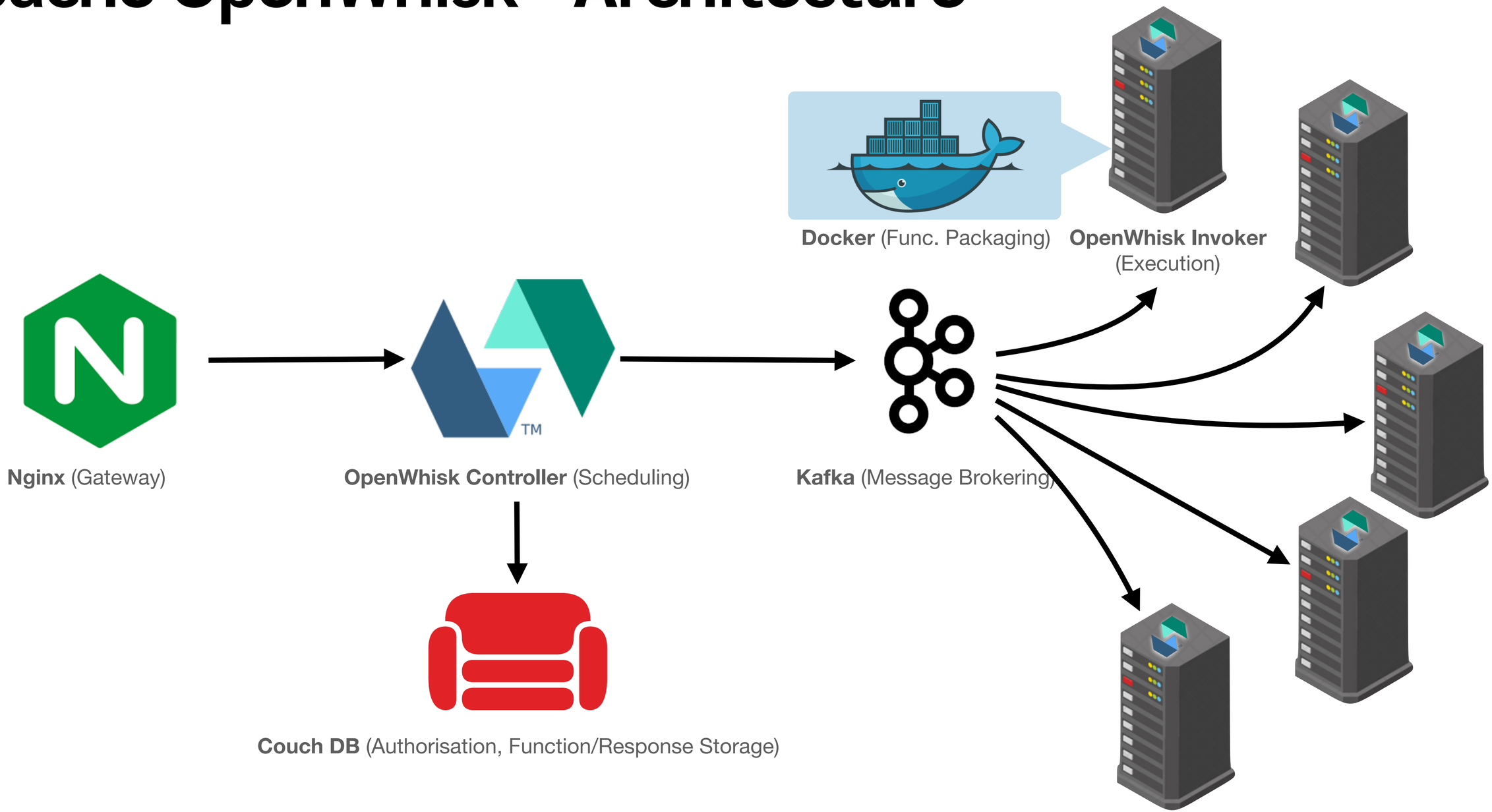
	Site 1	Site 2	Public Cloud	Average (ms)	95% Average (ms)
E	1000	0	0	1096.53	1019.03
S	466	534	0	149.18	90.86
B	0	90	910	105.18	64.62

Table 1. 1000 invocation for each function in the APP-based OpenWhisk deployment.

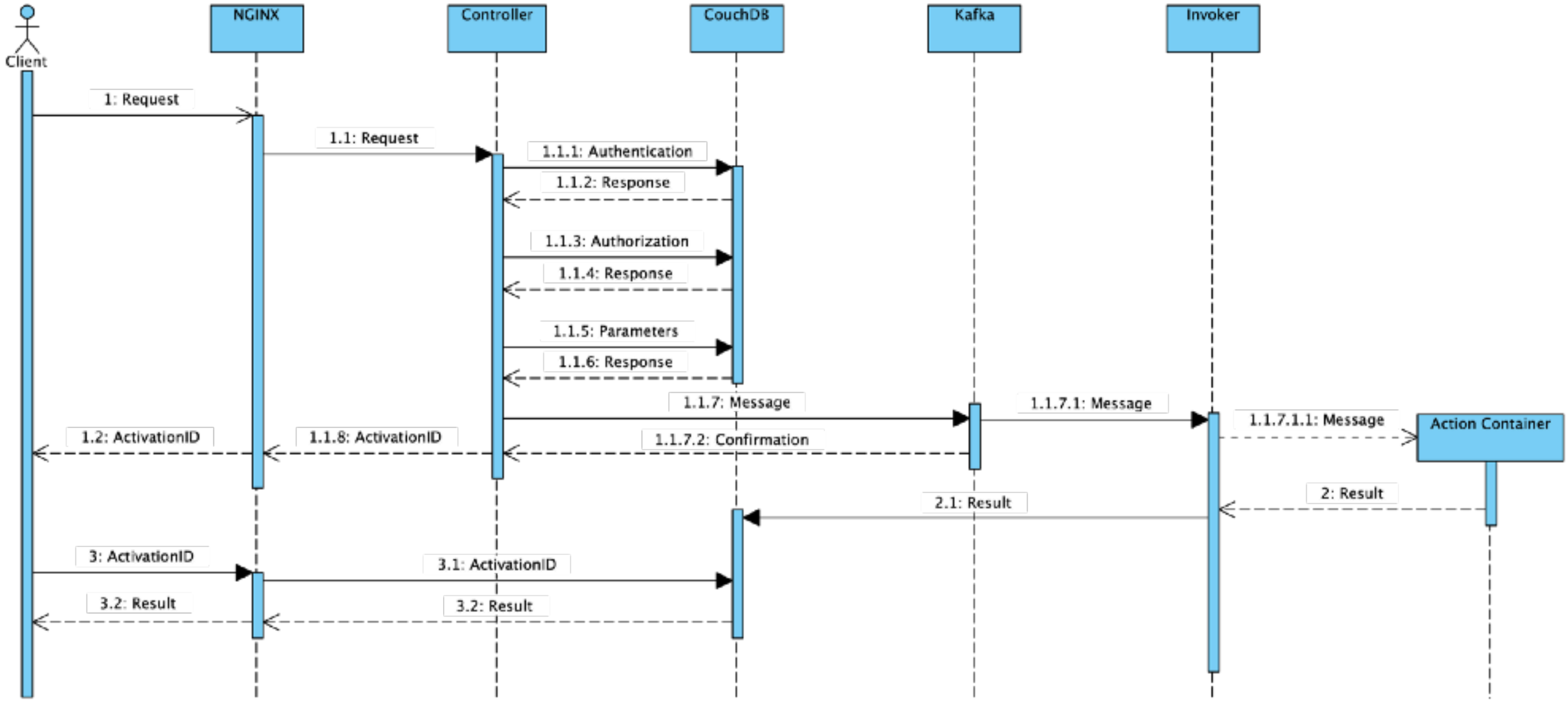
		Site 1	Site 2	Public Cloud	Average (ms)	95% Average (ms)
OW1	E	1000	0	0	1159.90	1025.52
OW2	S	19	981	0	385.30	302.08
OW3	B	185	815	0	265.69	215.793

Table 2. 1000 invocations for each function in the vanilla OpenWhisk deployment.

Apache OpenWhisk • Architecture



Apache OpenWhisk



Future Work

- Automatic configuration of priority policies (ML, heuristics, etc.);
- Extend our prototype to support pools of workers;
- Test APP's expressiveness by capturing and implementing the policies presented other papers on Serverless scheduling;
- Extend APP to describe (and not just use) scheduling algorithms and support the creation of user-defined libraries;
- Formalise the semantics of APP, useful for both a rigorous specification and to automatically reason on the properties of APP-defined deployments.


THANK
YOU