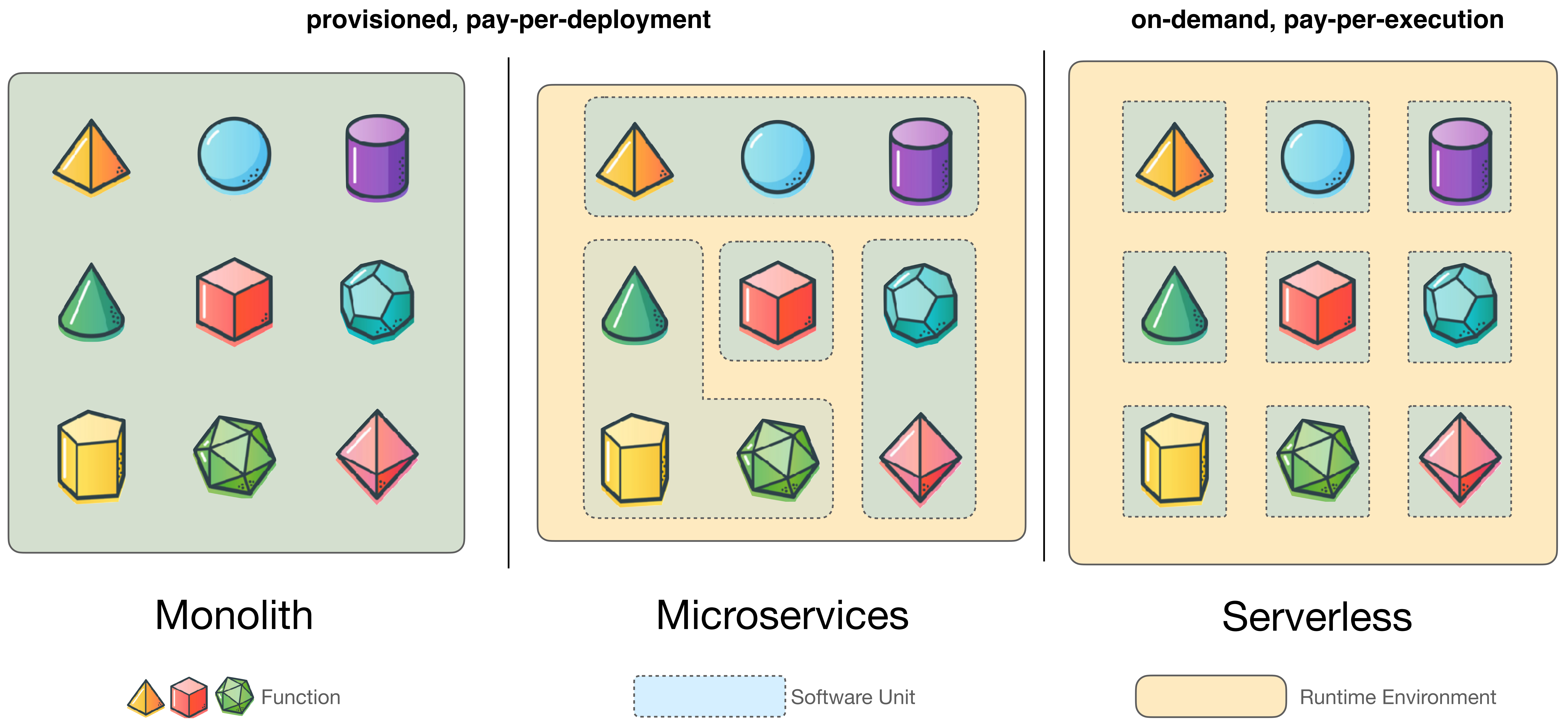
Challenges of Serverless: can languages help?

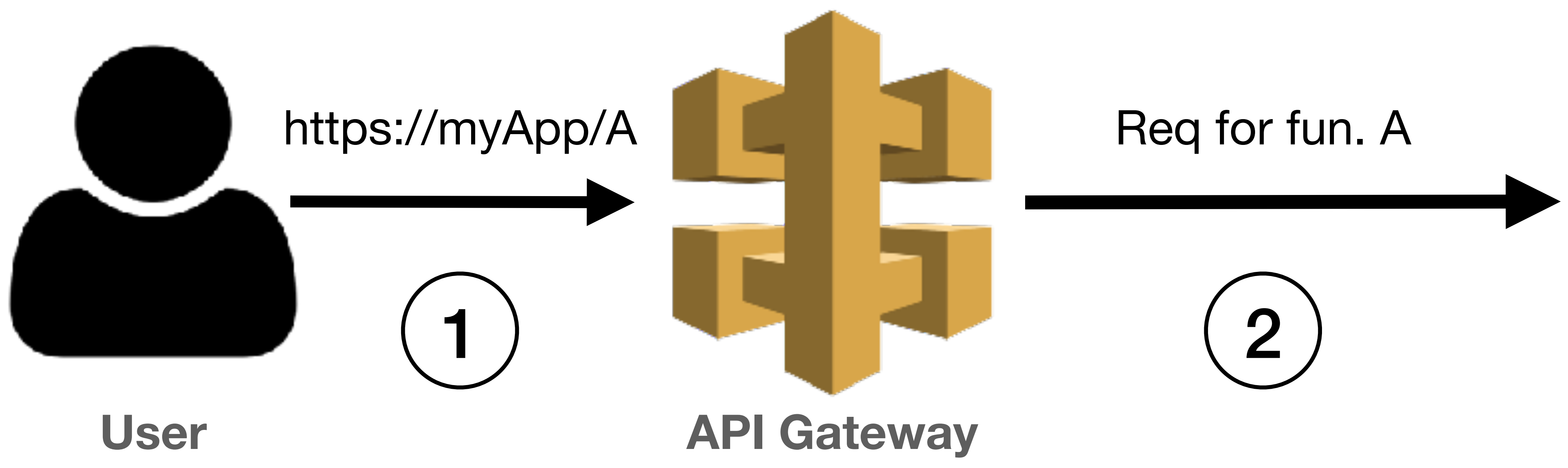
Saverio Giallorenzo

Università di Bologna (IT) INRIA (FR)

Serverless (and Microservices)

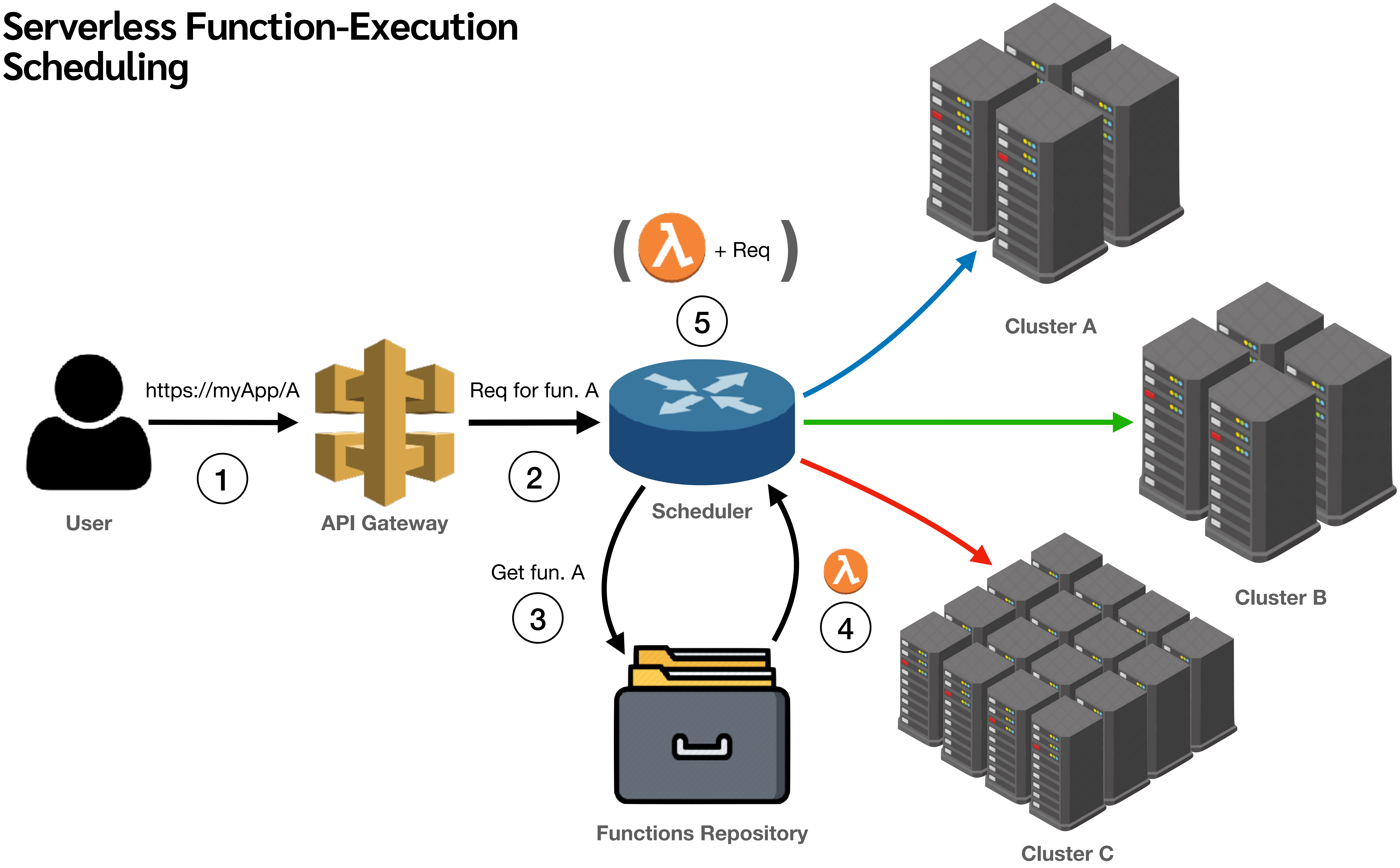


Serverless != CGI




```
# 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
```

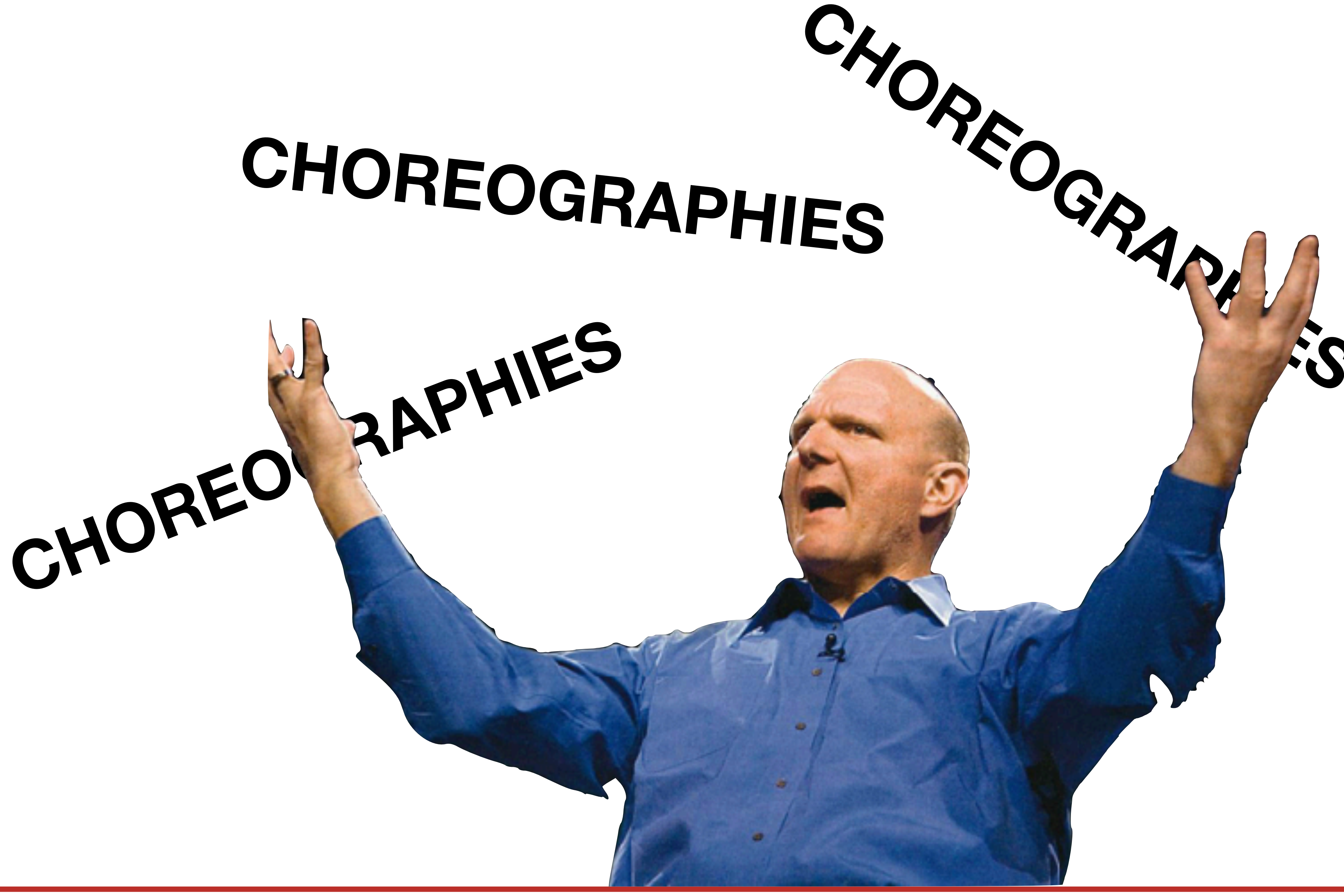
Serverless Function-Execution Scheduling



Open Problems in Serverless and Ideas

Few programmers can correctly write down their program semantics in a sequential language while also accounting for **parallel interleaving**, **message reordering**, **partial failures**, and **dynamic autoscaling** deployment.

Availability protocols are frequently interleaved into program logic in ways that make them tricky to test and evolve.



From “New Directions in Cloud Programming”

Open Problems in Serverless and Ideas

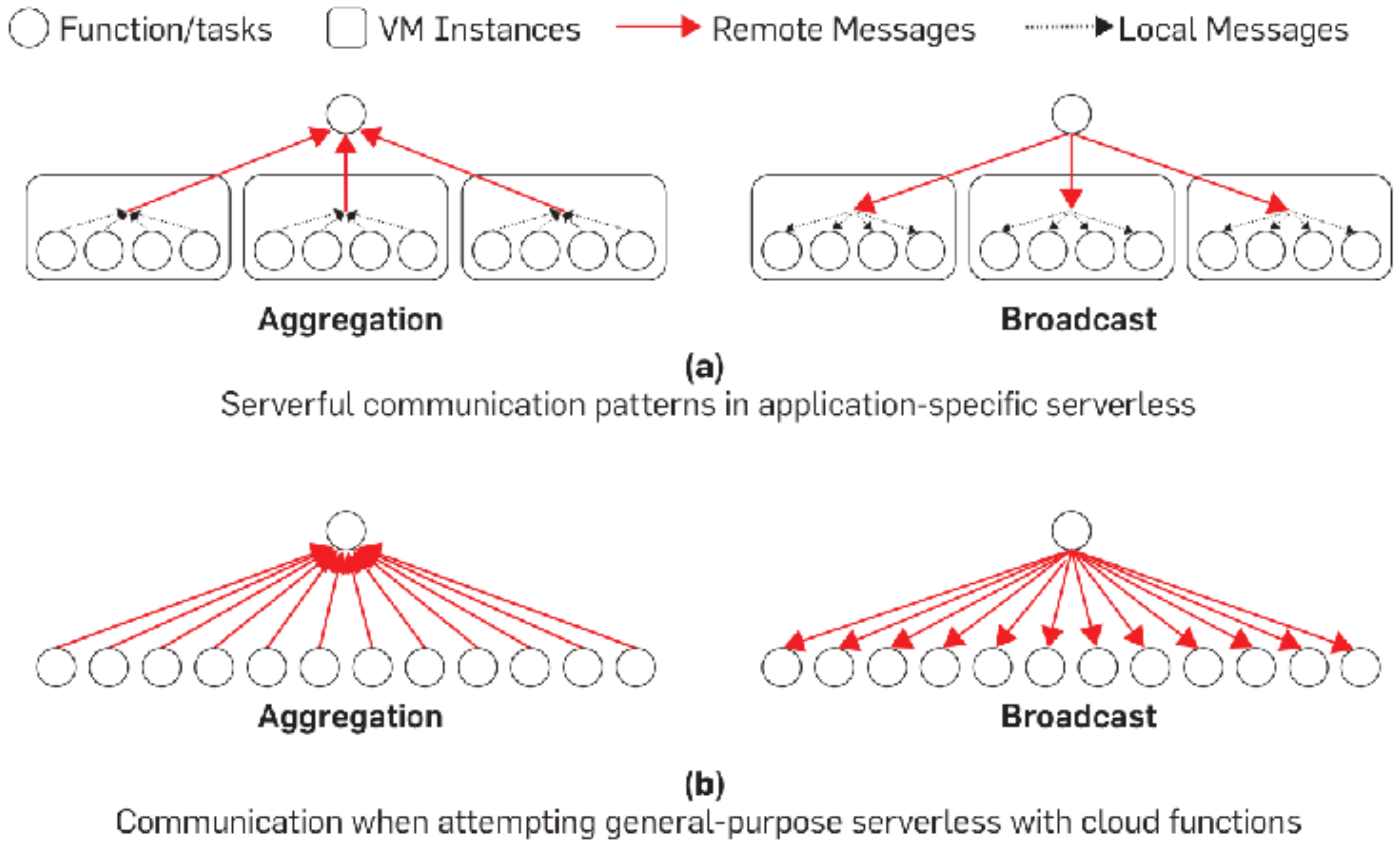
State management/security: cloud applications often need to **share or exchange short-lived/ephemeral state** among its components, e.g., application-wide caches, indexes, lookup tables, intermediate results. In serverless, this is usually solved via object storage and key-value stores, but the logic of the distributed system becomes fragmented, even when we neglect to consider access control, availability, consistency, etc.

One possibility is to have stateful cloud functions (e.g., wrapping caches)[SKC]

From “What Serverless computing is and should become”

Open Problems in Serverless and Ideas

Scheduling: users cede control over where functions run and, as a consequence, providers **cannot support direct communication**. This **prevents** users to express serverful optimisations, e.g., **passing state between functions** running in the same machine, which instead requires the two functions to run through at least two trips (push and pull) from some shared storage.



The shortcomings of this become more relevant, e.g., in the case of scatter/gather patterns.

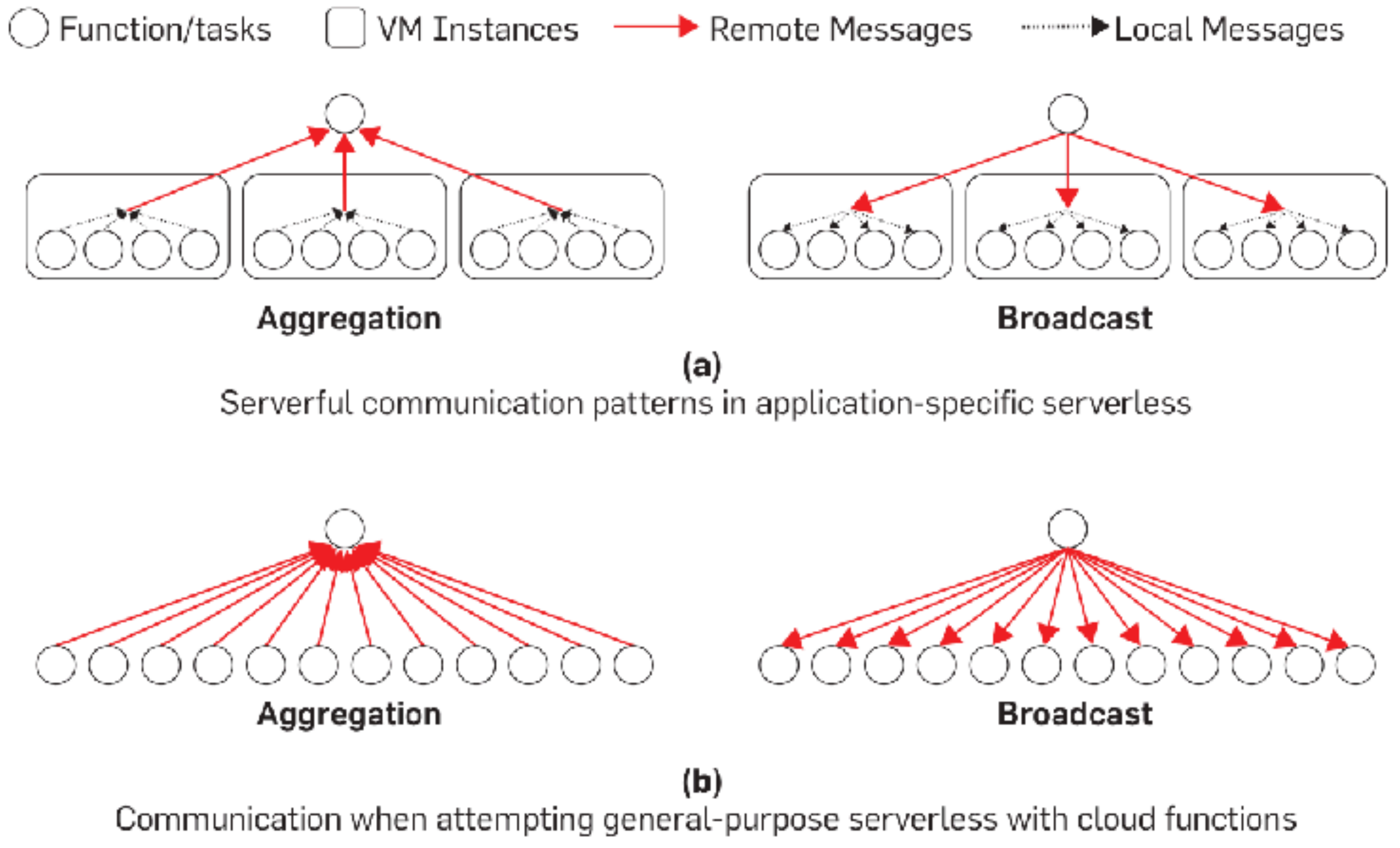
A solution here is to let languages express this information (from developers) and provide it to systems like APP so they can figure out the most efficient (as in, cheapest, fastest, greenest, etc.) way to achieve that.

From “What Serverless computing is and should become”

Open Problems in Serverless and Ideas

A solution here is to let languages express this information (from developers) and provide it to systems like APP so they can figure out the most efficient (as in, cheapest, fastest, greenest, etc.) way to achieve that.

From “What Serverless computing is and should become”



…an environment where programmers can specify multi-objective performance targets for execution, e.g., tradeoffs between billing costs, latency, and availability.

From “New Directions in Cloud Programming”

Open Problems in Serverless and Ideas

Simple COVID-19 Tracker App in Pythonic HydroLogic

```
class Person: (pid: int, country: string,
        contacts: Set(&Person), covid: bool, vaccinated: bool,
        key=pid, partition=country)
table people: Person
var vaccine_count: int

on add_person(pid: int):
  people.merge(Person(pid)) # monotonic mutation
  return OK

on add_contact(p: Person, p1: Person):
  p.contacts.merge(p1) # monotonic mutation
  p1.contacts.merge(p) # monotonic mutation
  return OK

query transitive(p: Person, p1: Person): # monotonic query
    {(p, p1) for p in people for p1 in p.contacts}
    {(p, p2) for (p, p1) in transitive for p2 in p1.contacts}

on trace(p: Person):
  return (p2 for (p, p2) in transitive(p, _)

on diagnosed(pid: int):
  people[pid].covid.merge(true) # monotonic mutation
  send alert(p: Person) {p for p in trace(pid)}

from covid_xmission_model import covid_predict
on likelihood(pid: int):
  return covid_predict(people[pid])

on vaccinate(pid: int, consistency={serializable;
                 vaccine_count >= 0; people.has_key(pid)}):
  people[pid].vaccinated.merge(True) # monotonic mutation
  vaccine_count := vaccine_count - 1 # NON-monotonic mutation
  return OK

availability:
    default: { domain = AZ, failures = 2 }
    likelihood: { domain = AZ, failures = 1 }

target:
    default: { latency = 100ms, cost = 0.01units }
    likelihood: { processor = GPU, cost = 0.1units }
```

Hydrologic, from “New Directions in Cloud Programming”

Open Problems in Serverless and Ideas

Simple COVID-19 Tracker App in Pythonic HydroLogic

```
class Person: (pid: int, country: string,
        contacts: Set(&Person), covid: bool, vaccinated: bool,
        key=pid, partition=country)
table people: Person
var vaccine_count: int

on add_person(pid: int):
  people.merge(Person(pid)) # monotonic mutation
  return OK

on add_contact(p: Person, p1: Person):
  p.contacts.merge(p1) # monotonic mutation
  p1.contacts.merge(p) # monotonic mutation
  return OK

query transitive(p: Person, p1: Person): # monotonic query
    {(p, p1) for p in people for p1 in p.contacts}
    {(p, p2) for (p, p1) in transitive for p2 in p1.contacts}

on trace(p: Person):
  return (p2 for (p, p2) in transitive(p, _)

on diagnosed(pid: int):
  people[pid].covid.merge(true) # monotonic mutation
  send alert(p: Person) {p for p in trace(pid)}

from covid_xmission_model import covid_predict
on likelihood(pid: int):
  return covid_predict(people[pid])

on vaccinate(pid: int, consistency={serializable;
              vaccine_count >= 0; people.has_key(pid)}):
  people[pid].vaccinated.merge(True) # monotonic mutation
  vaccine_count := vaccine_count - 1 # NON-monotonic mutation
  return OK

availability:
    default: { domain = AZ, failures = 2 }
    likelihood: { domain = AZ, failures = 1 }

target:
    default: { latency = 100ms, cost = 0.01units }
    likelihood: { processor = GPU, cost = 0.1units }
```

Hydrologic considers an **event loop** that runs on successive *snapshots* of the overall program state, including new inbound messages to be handled.

Each iteration of the loop uses the developer's program specification to compute new results from the snapshot and atomically updates the state of the program and the end of the loop.

This is a kind of "single-node" model that leaves to other part of the specification (bottom) the task to define their deployment-related information.

E.g., they can indicate a certain number of endpoints for a specific function and define load-balancing and replication policies to meet a given SLA on response rate.

Hydrologic, from "New Directions in Cloud Programming"

Open Problems in Serverless · Ideas

- Can we use Jolie to deploy a microservices architecture as a serverless one? Could the language help in understanding/capturing the characteristics of **serverless** and **serverful** architectures?

- Can we use choreographies to define a **server*** architecture, e.g., specifying/constraining parts of the interactions with a specific semantics, but leave the compiler/deployer decide the “best” strategies for the other ones (e.g., sets of messages, queues, etc)

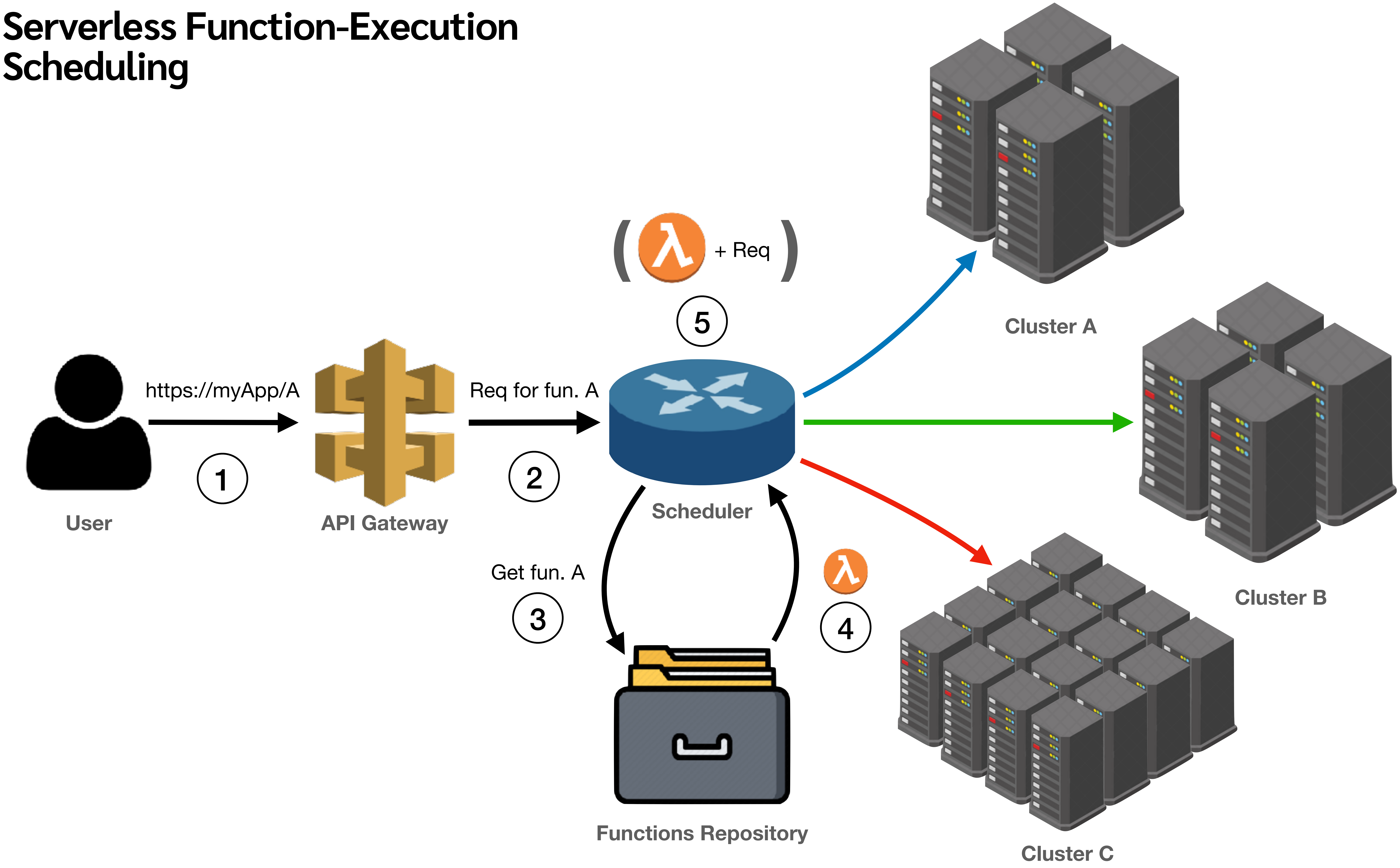
    - For the constrained parts, can we replicate (as in “preserve”) the platform-specific semantics in either setting, e.g., by synthesising proxy services?

Allocation Priority Policies for Serverless Function-execution Scheduling Optimisation

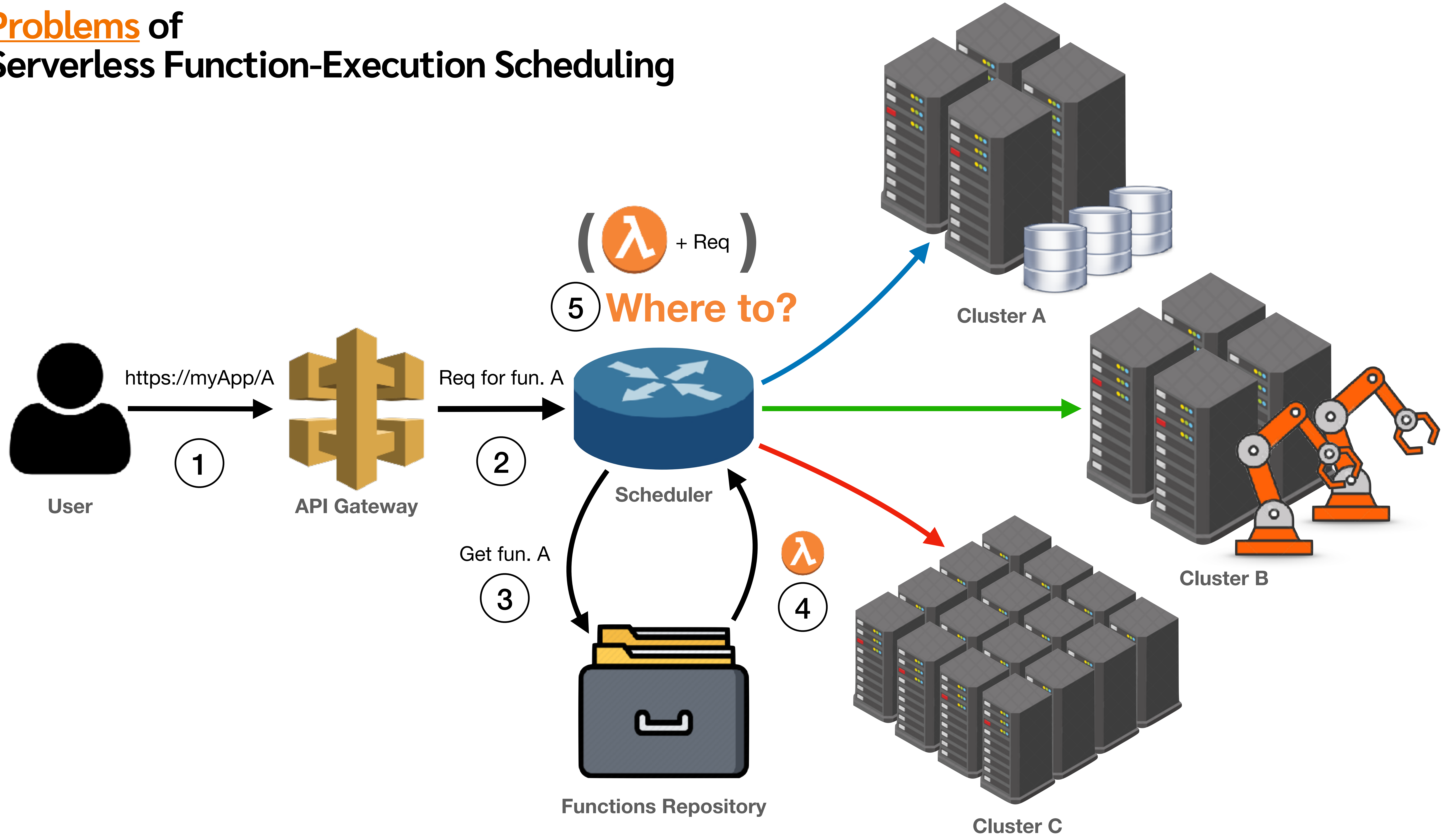
Giuseppe de Palma[1], Saverio Giallorenzo[1,2], Jacopo Mauro[3] and Gianluigi Zavattaro[1,2]

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

Serverless Function-Execution Scheduling

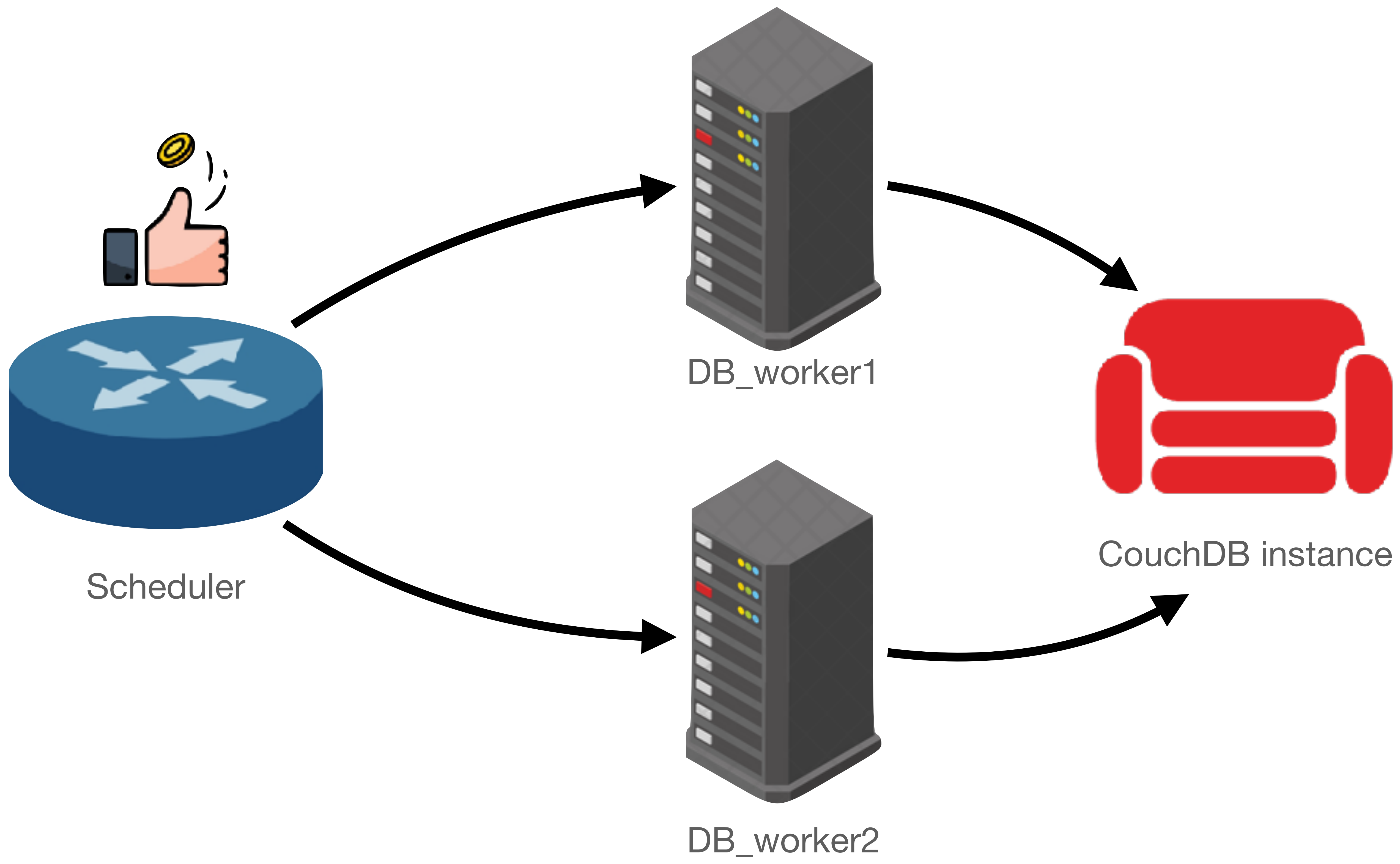


Problems of Serverless Function-Execution Scheduling



The APP Language • First Example

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

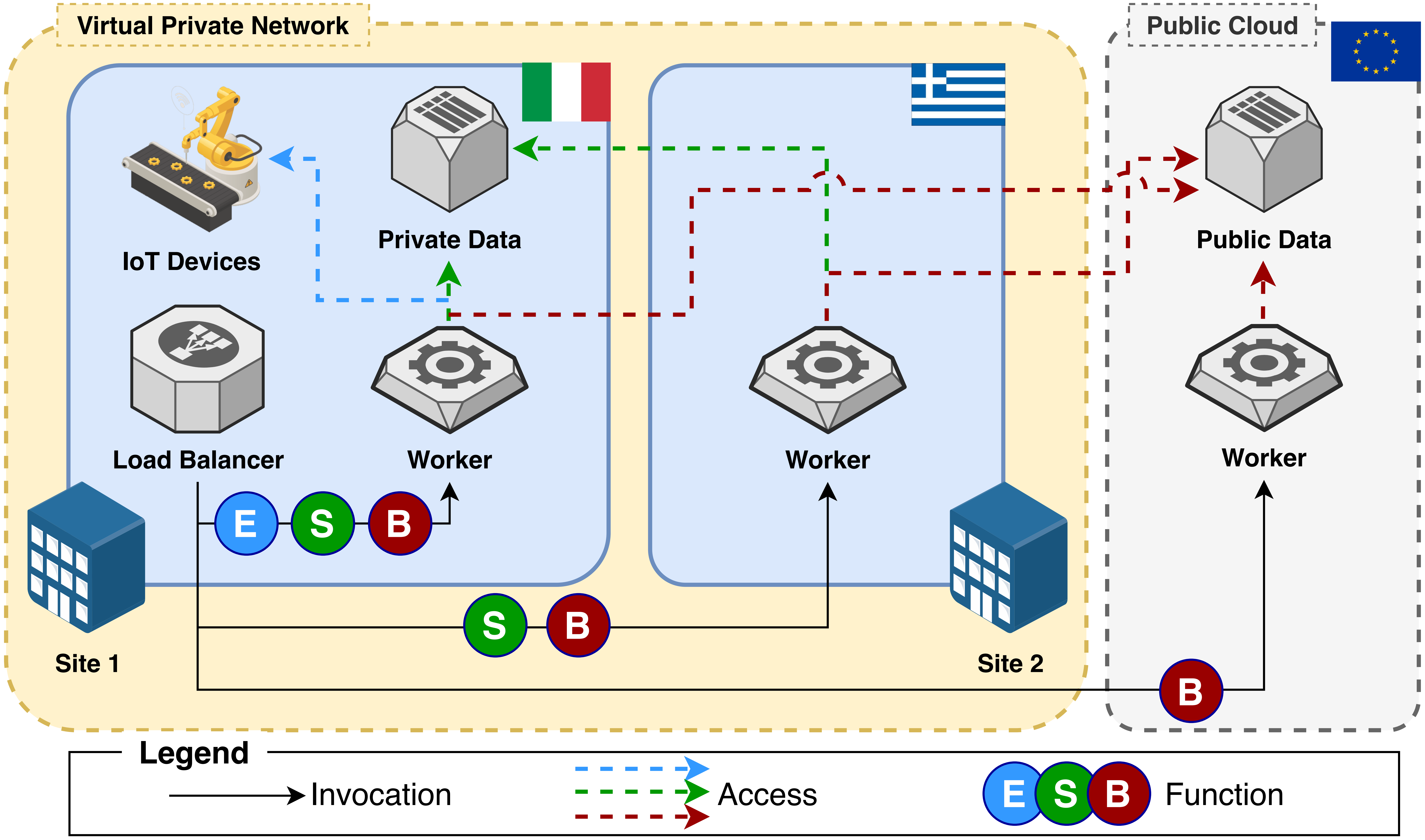


The APP Language • Syntax

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

$$
\begin{array}{lcl}
app & ::= & \overline{tag} \\
tag & ::= & policy\_tag \; \texttt{:} \; \overline{\texttt{-}\; block} \; followup? \\
block & ::= & \texttt{workers} \; [\; \texttt{"*"} \mid \overline{\texttt{-}\; worker\_label} \;] \\
 & & (\texttt{strategy} \; [\; \texttt{random} \mid \texttt{platform} \mid \texttt{best\_first} \;])? \\
 & & (\texttt{invalidate} \; [\; \texttt{capacity\_used} \; \texttt{:} \; n\texttt{\%} \mid \texttt{max\_concurrent\_invocations} \; \texttt{:} \; n \mid \texttt{overload} \;])? \\
followup & ::= & \texttt{followup} \; \texttt{:} \; [\; \texttt{default} \mid \texttt{fail} \;]
\end{array}
$$

Use case

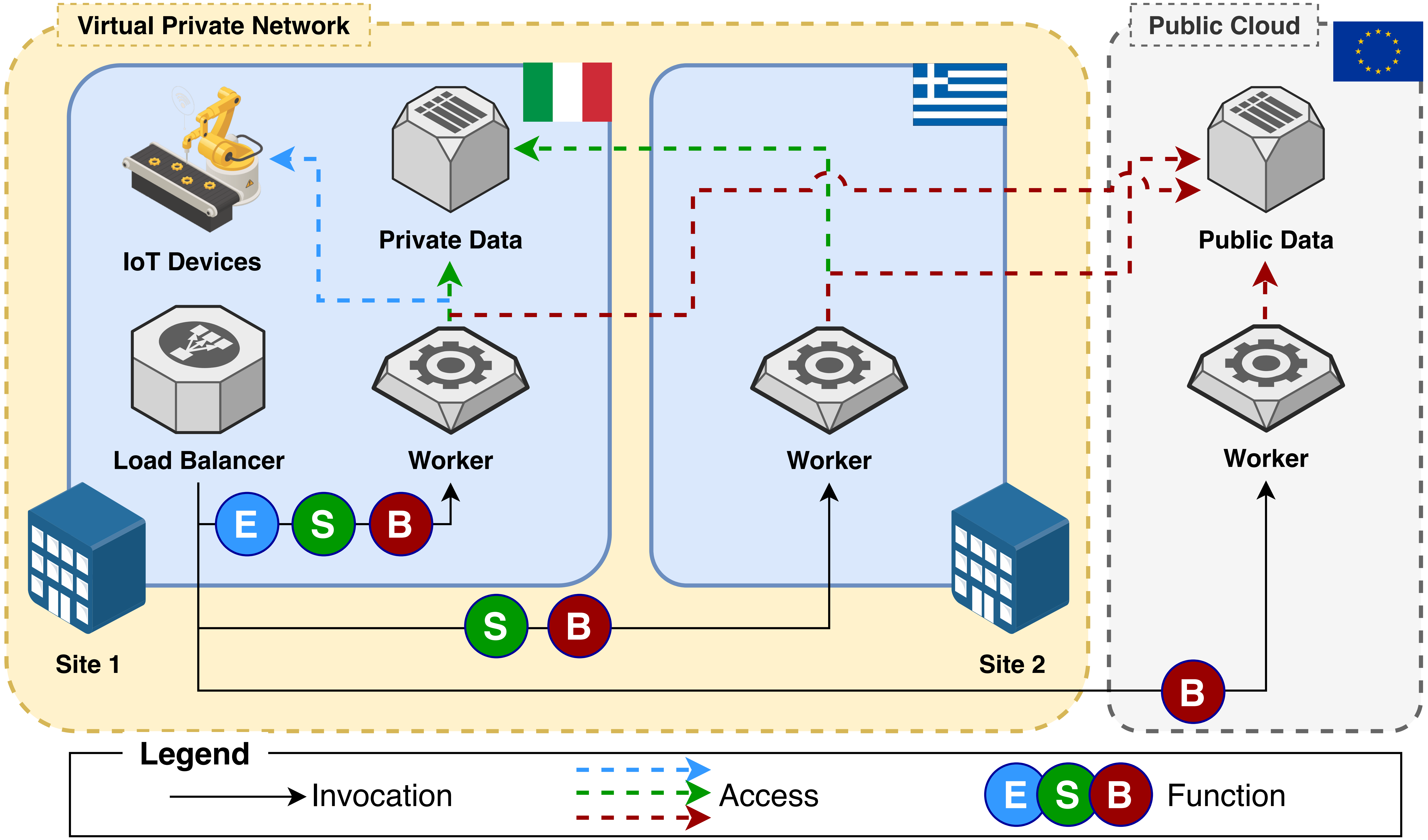

Virtual Private Network
Public Cloud
IoT Devices
Private Data
Public Data
Load Balancer
Worker
Worker
Worker
E
S
B
S
B
B
Site 1
Site 2
Legend
Invocation
Access
E
S
B
Function

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

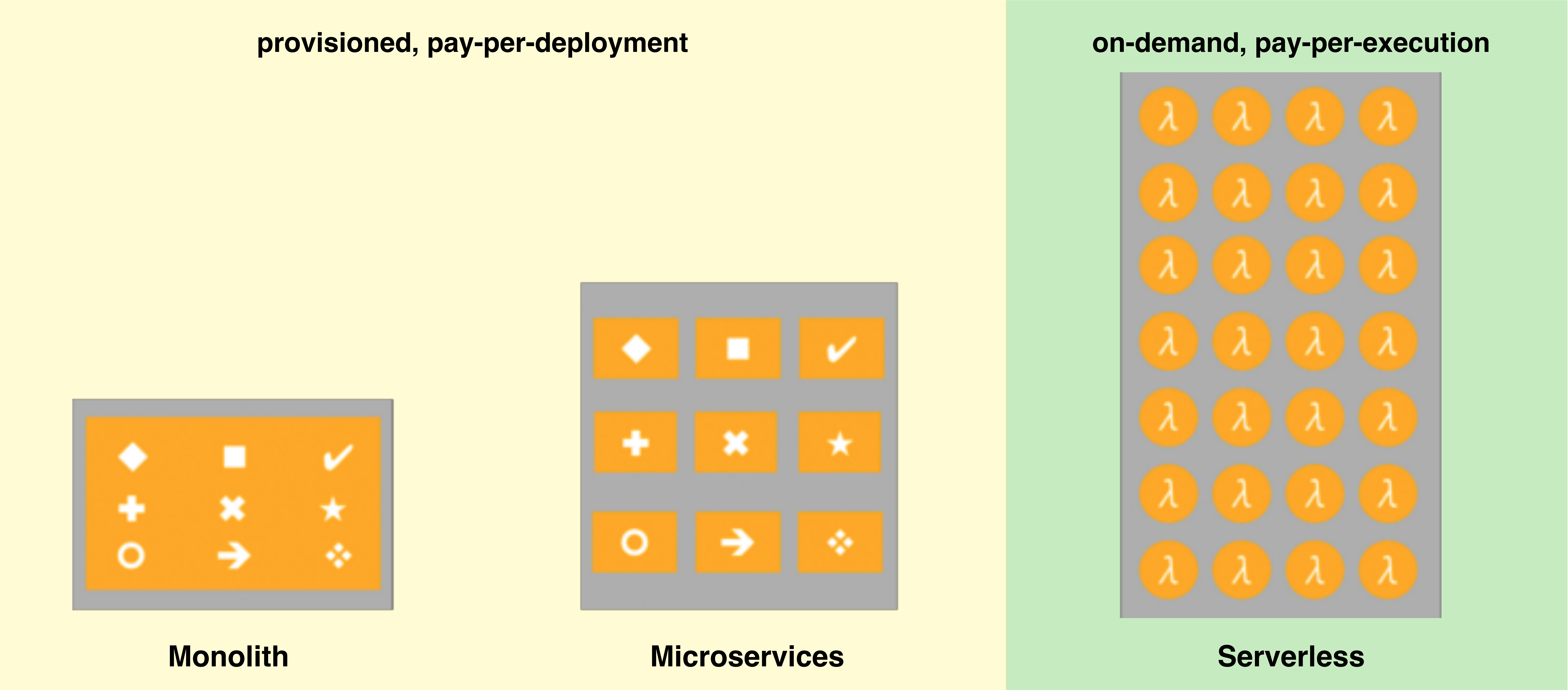
Table 2. 1000 invocations for each function in the vanilla OpenWhisk deployment.

Future Work

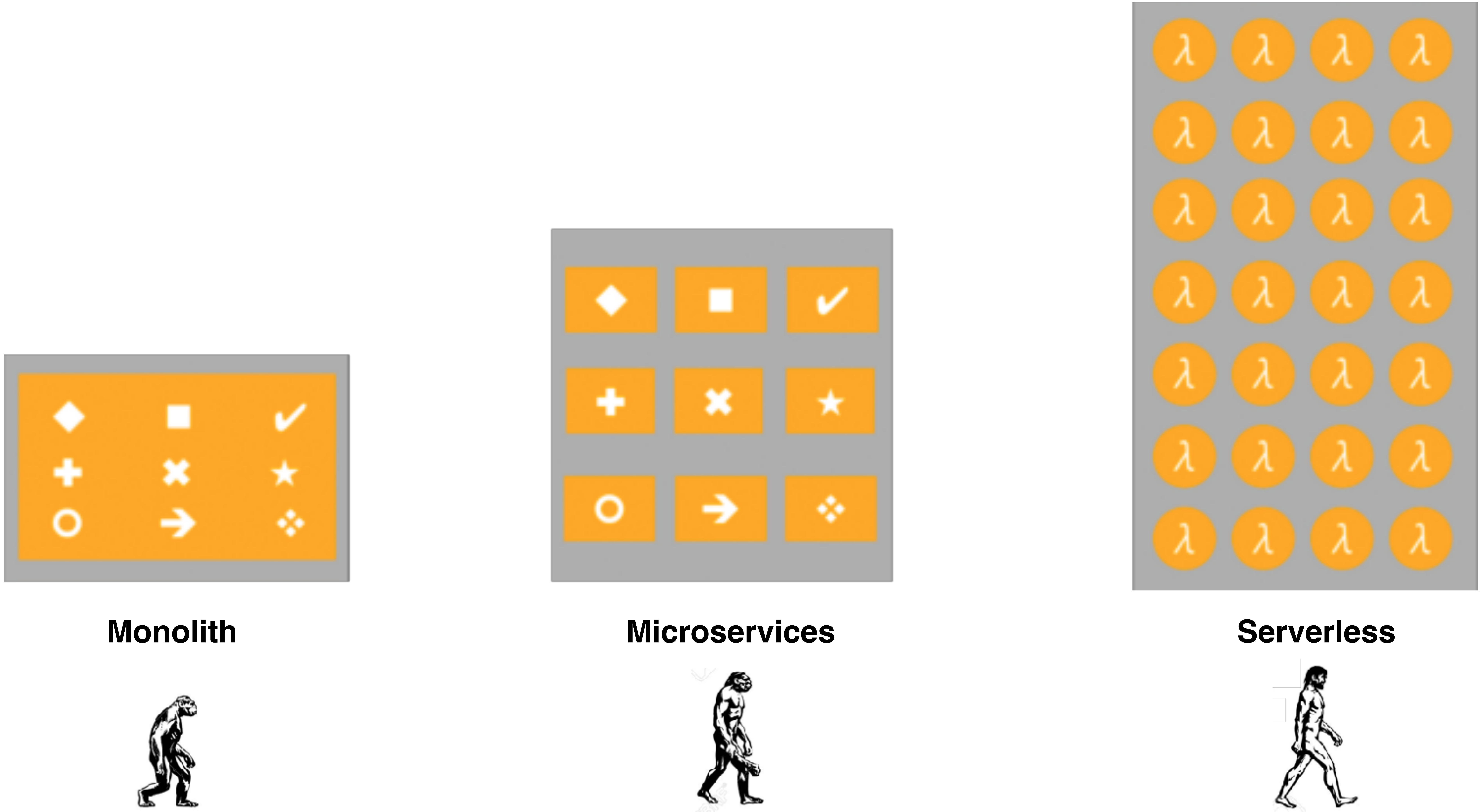
- Automatic configuration of priority policies (ML, heuristics, etc.);
- Extend our prototype to support pools of workers;
- Test the expressiveness of APP by capturing and implementing the policies presented other papers on Serverless scheduling;
- Extend APP to describe (and not just use) scheduling 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
L

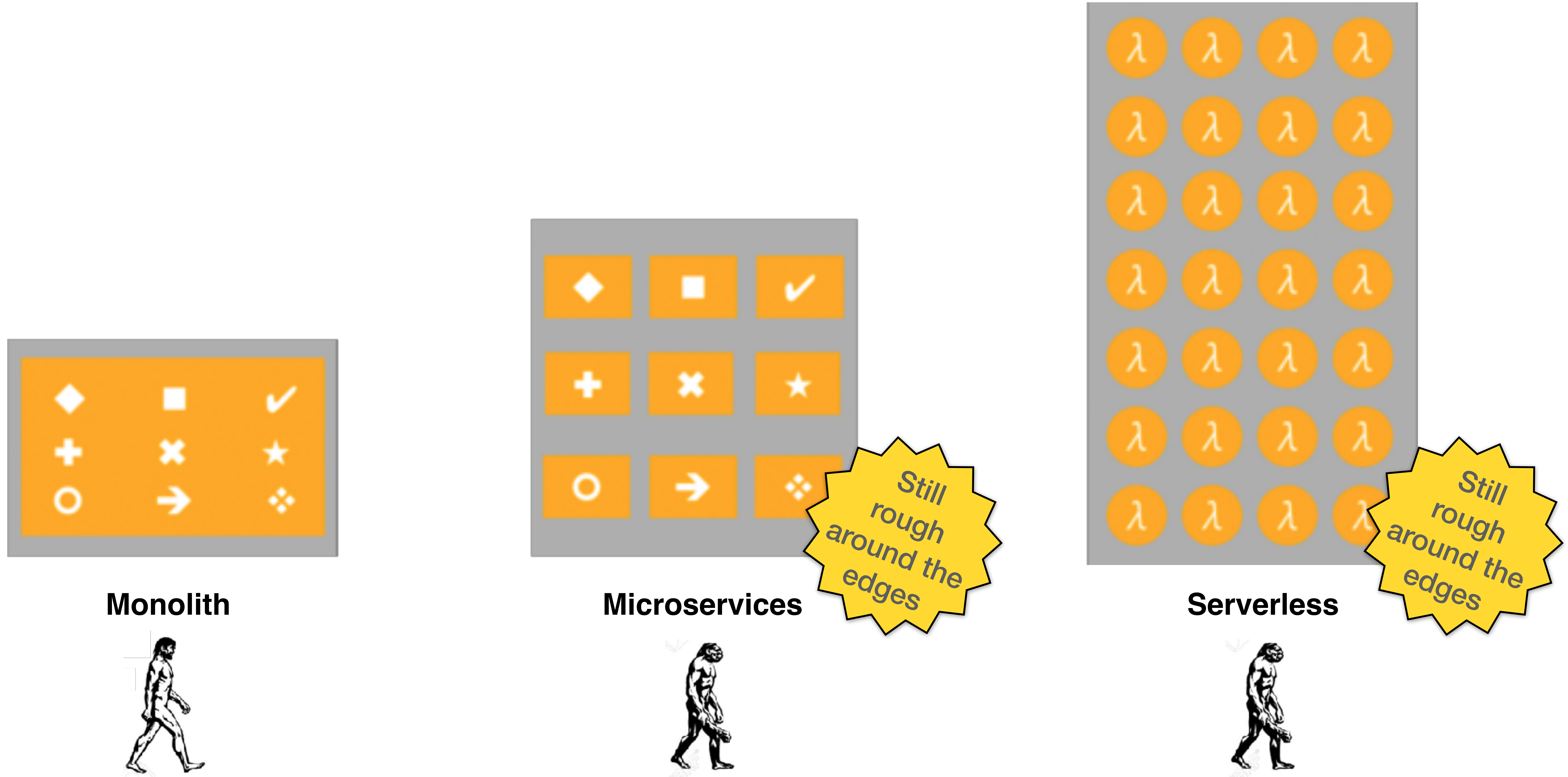
Serverless (and Microservices)



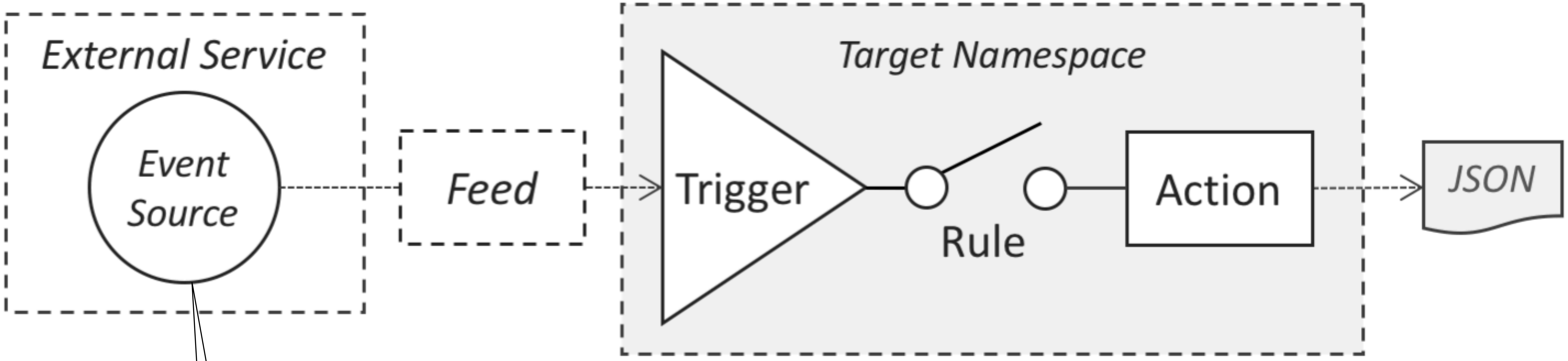
Serverless (and Microservices)



Serverless (and Microservices) • Readiness

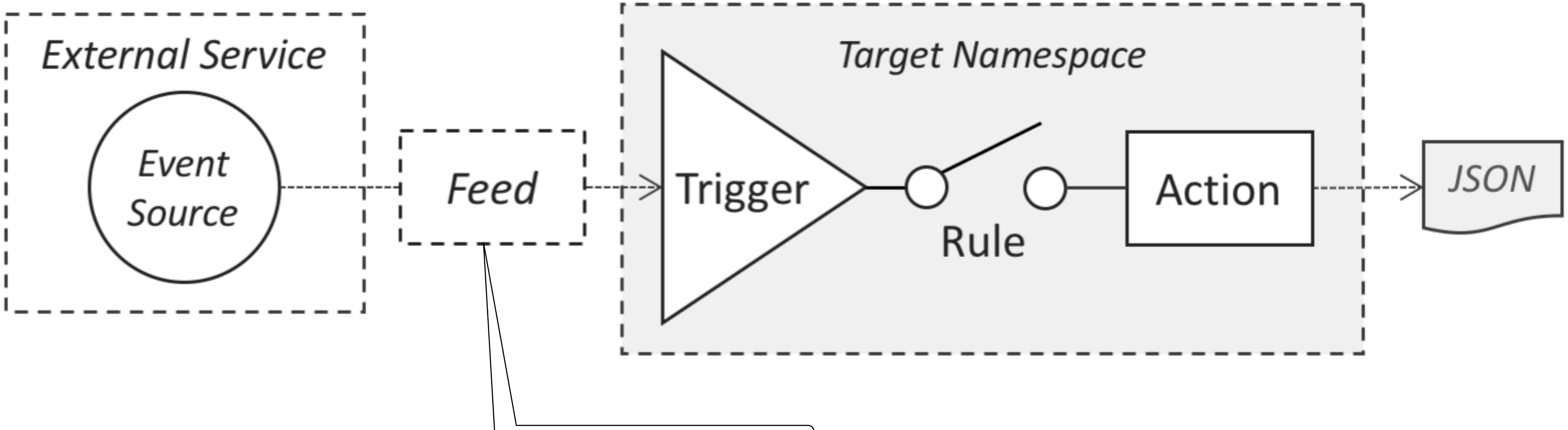


Apache OpenWhisk • Programming Model



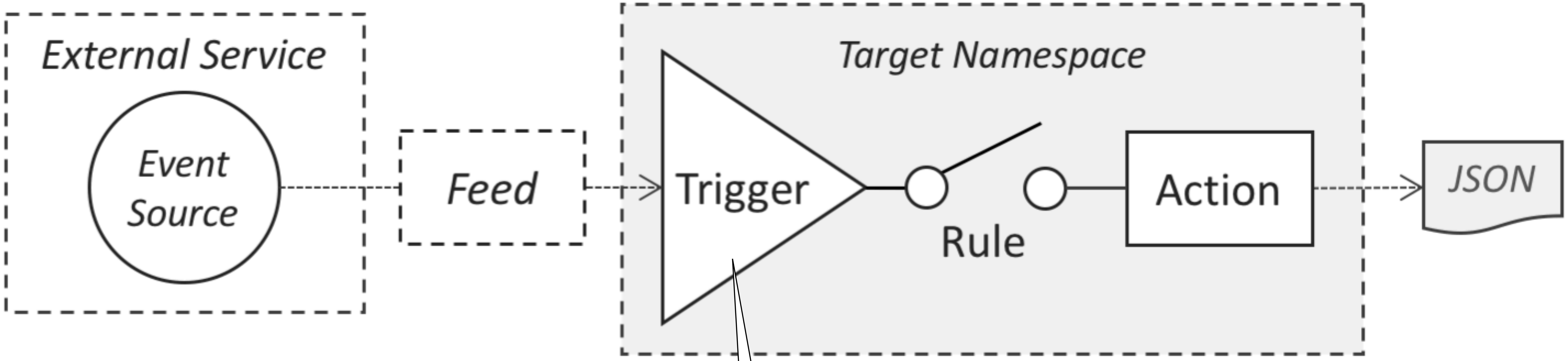
Services that generate events carrying data (requests) or indicating changes on data (to act upon).

Apache OpenWhisk • Programming Model



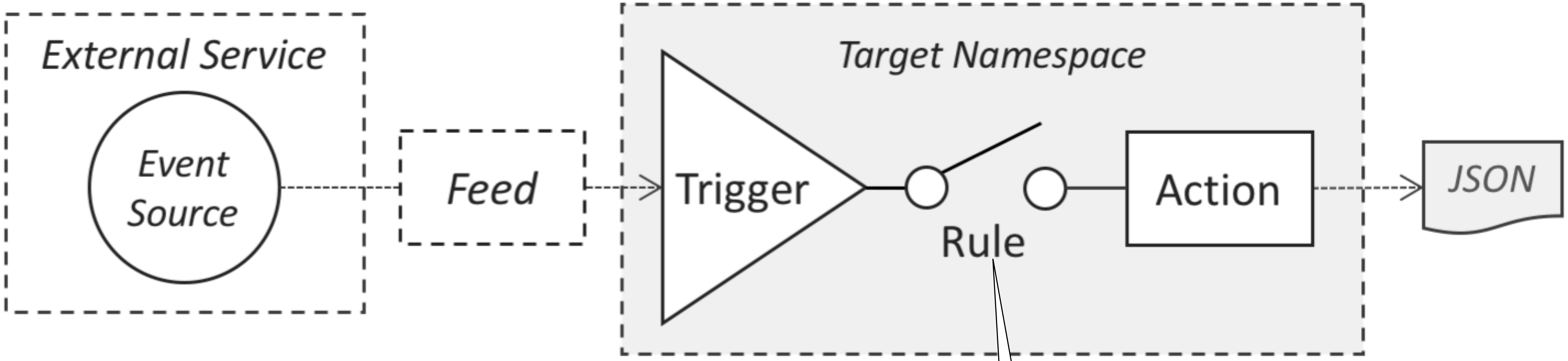
An event broker, receiving and handling the stream of events for some triggers.

Apache OpenWhisk • Programming Model



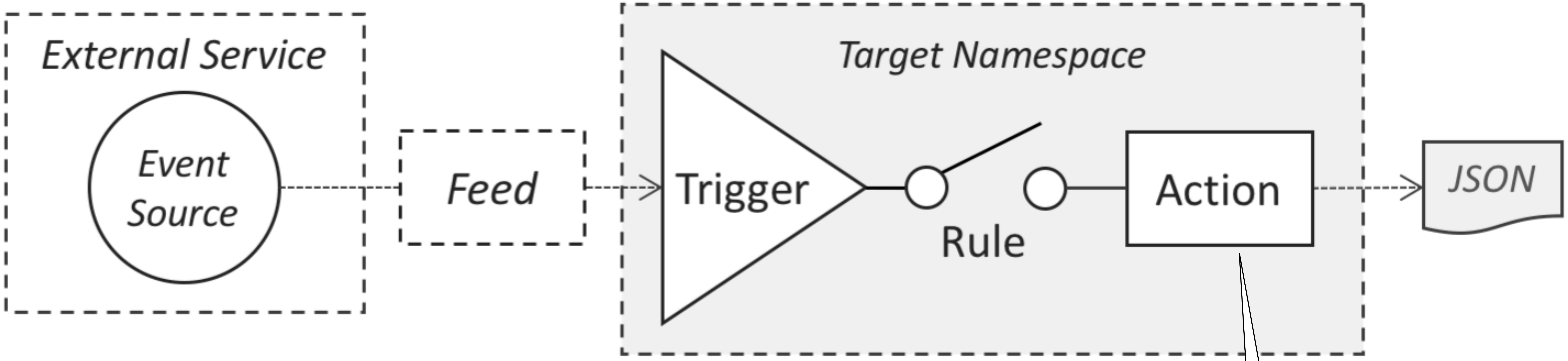
A named channel for classes of events

Apache OpenWhisk • Programming Model



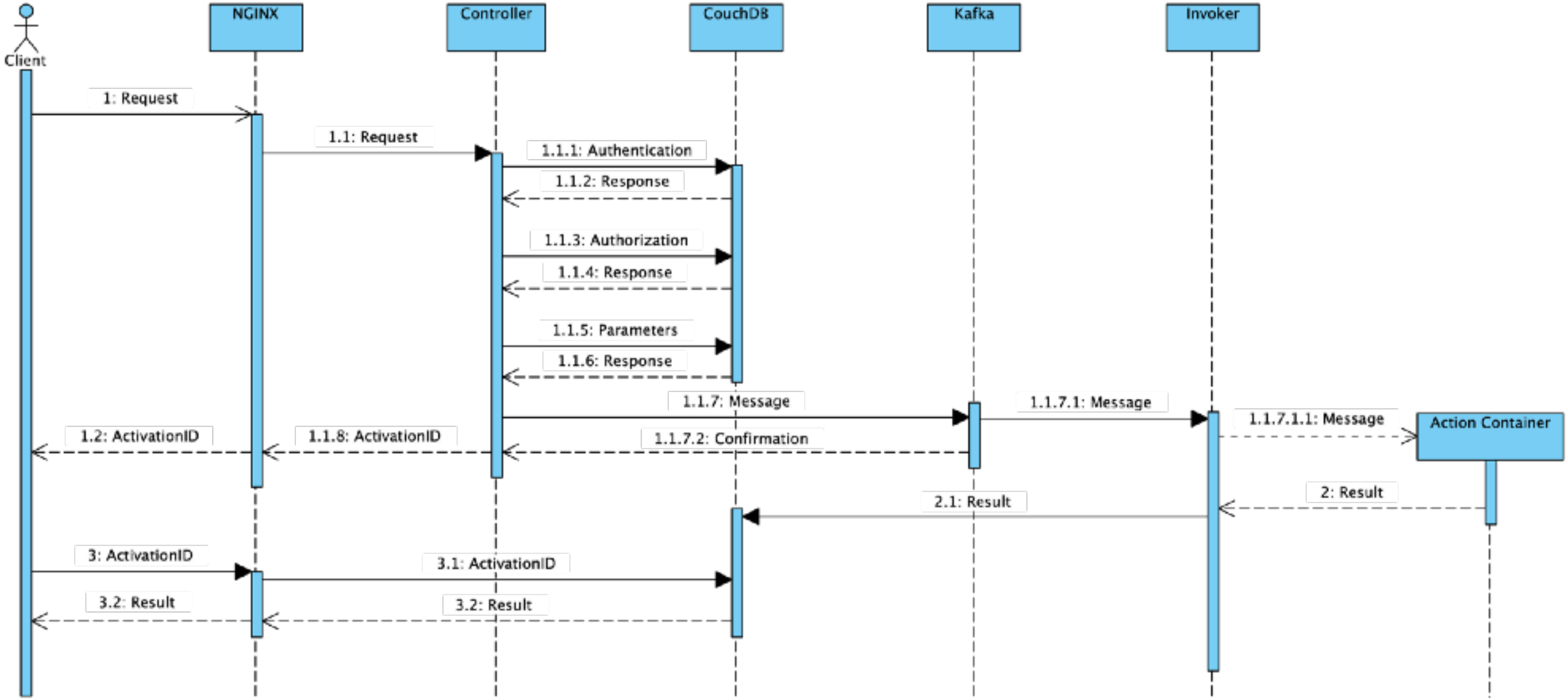
Associates one trigger with one action. Each time a trigger event is fired, the action is invoked.

Apache OpenWhisk • Programming Model



Stateless functions that run on the OpenWhisk platform

Apache OpenWhisk



Apache OpenWhisk • Architecture