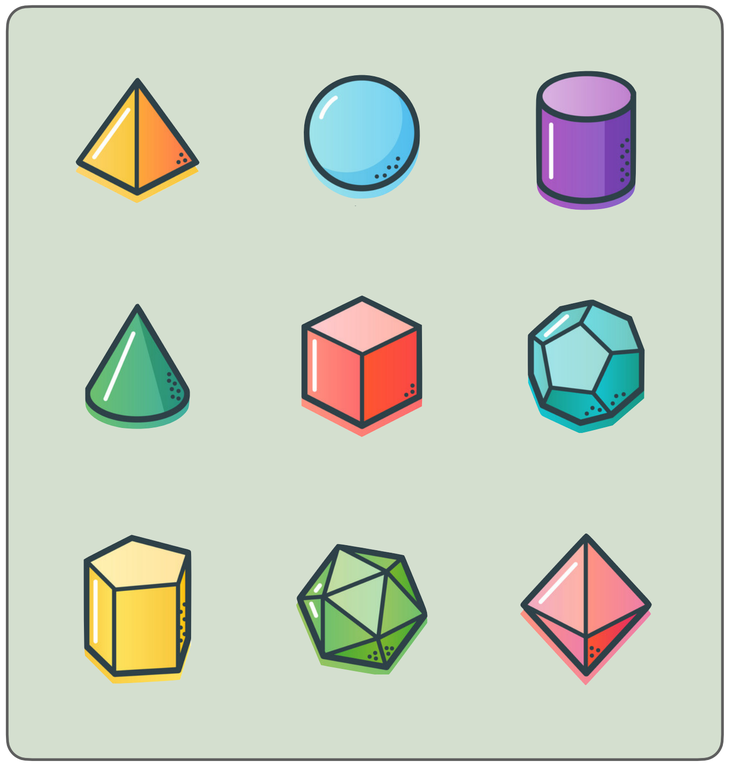
# A Unifying, Lightweight Platform for Microservice and Serverless Deployments

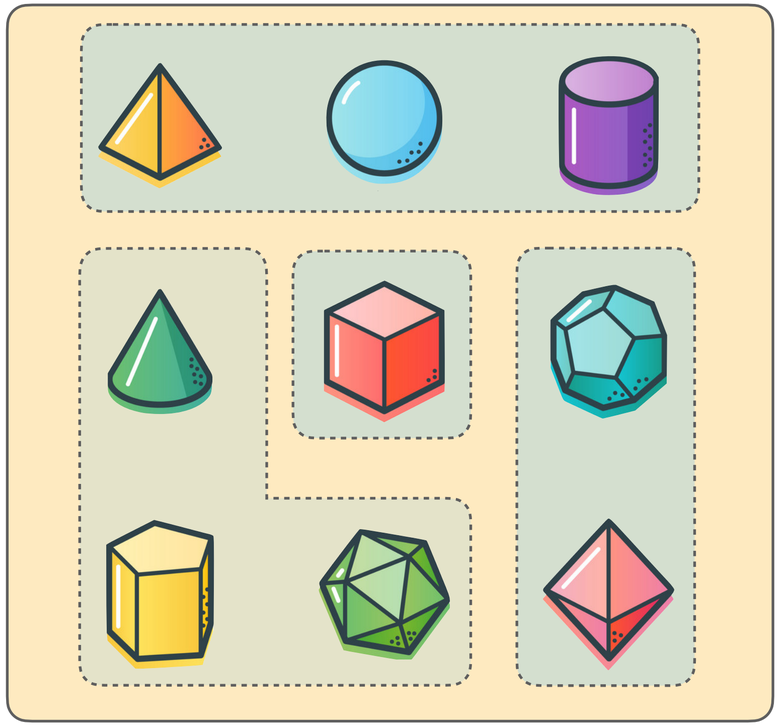
Saverio Giallorenzo[1,2], Claudio Guidi[3], Luca Tagliavini[1]

[1]Università di Bologna (IT) [2]FOCUS Team, INRIA (FR) [3]ItalianaSoftware S.r.l. (IT)

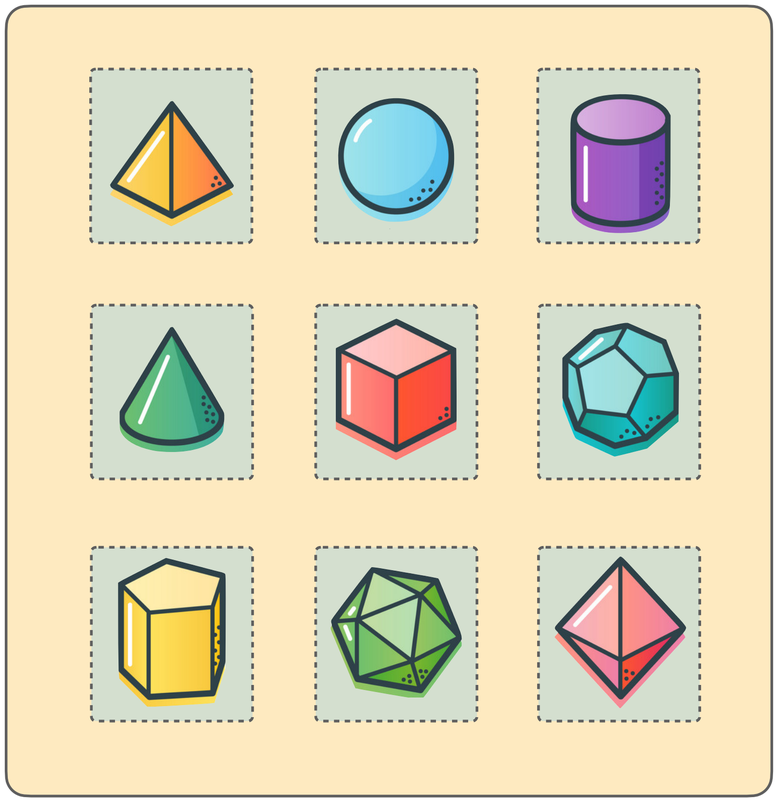
# A Gentle Introduction to Serverless



Monolith



Microservices



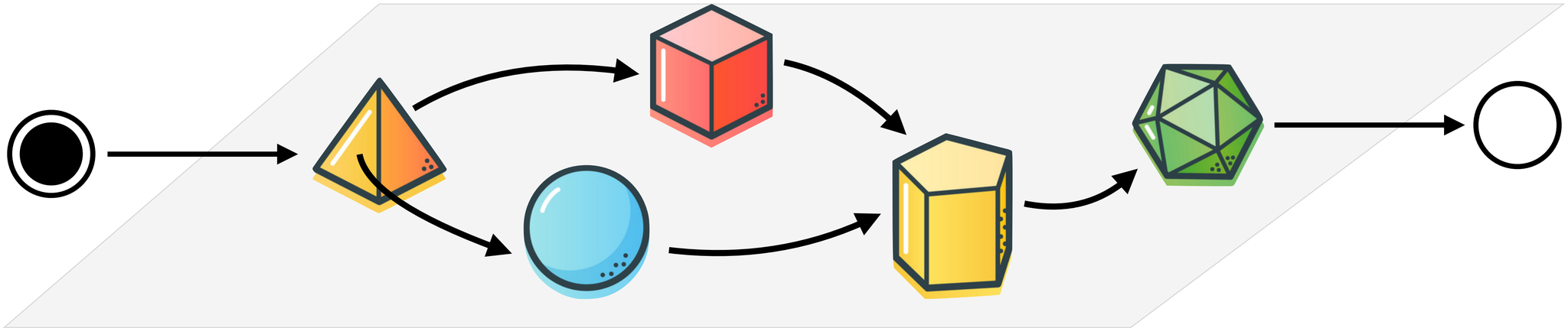
Serverless

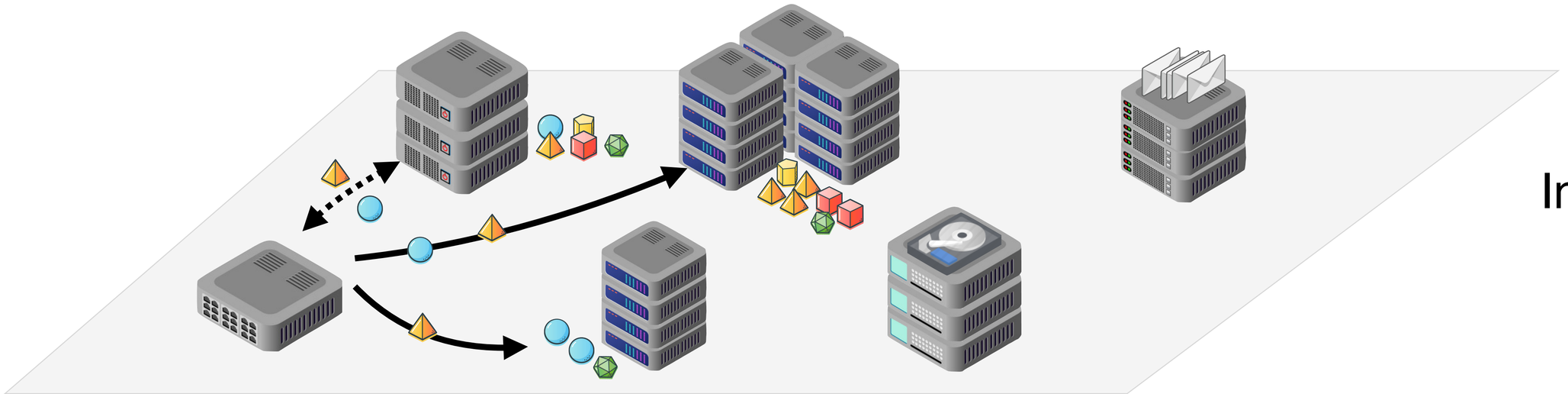
Function

Software Unit

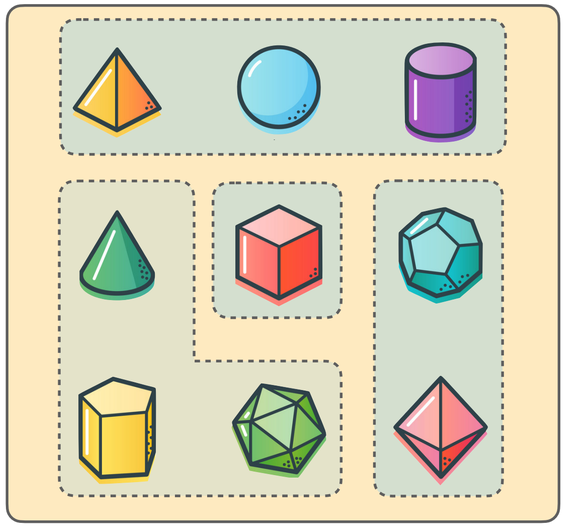
Runtime Environment

# A Gentle Introduction to Serverless



# Why unifying Microservices and Serverless?



## Microservices

**Pro**

- Resource-efficient on sustained traffic loads
- Marginal cold-start problems

**Cons**

- Waste resources when idle
- Complex deployment and scaling logic

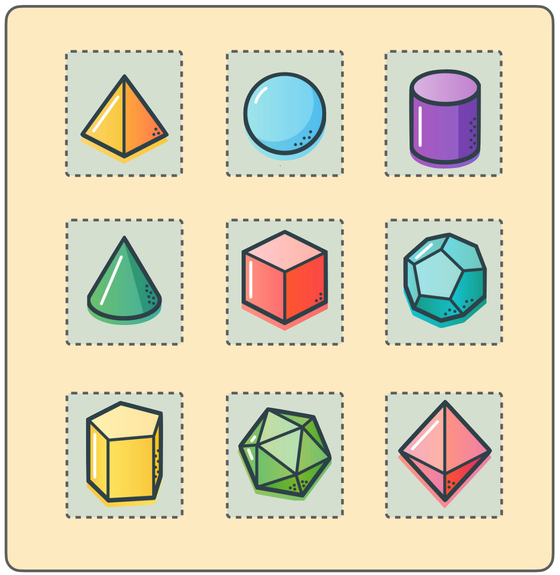
## Serverless

**Pro**

- Resource-efficient on intermittent traffic loads
- No deployment/scaling issue (platform-managed)

**Cons**

- Costly (and inefficient) under sustained traffic load
- Cold-start problems

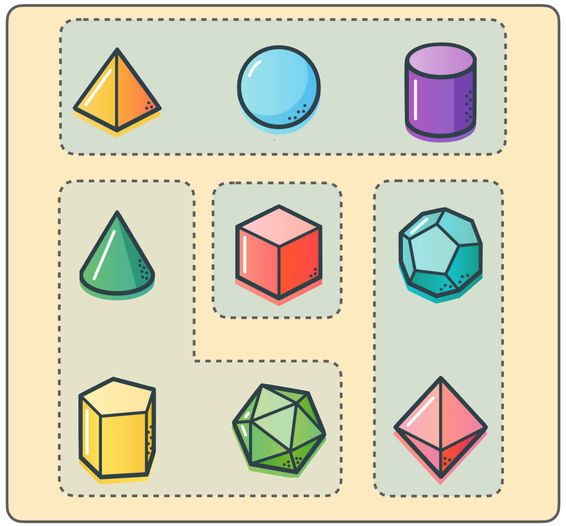


 Function

 Software Unit

 Runtime Environment

# Why unifying Microservices and Serverless?



## Microservices

**Pro**

- Resource-efficient on sustained traffic loads
- Marginal cold-start problems

**Cons**

- Waste resources when idle
- Complex deployment and scaling logic

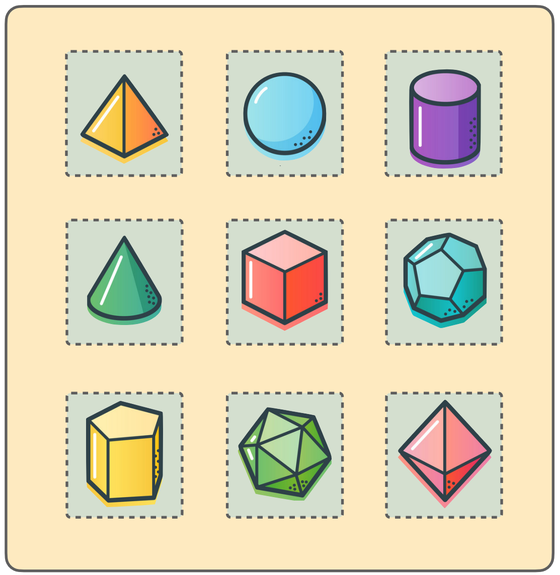
## Serverless

**Pro**

- Resource-efficient on intermittent traffic loads
- No deployment/scaling issue (platform-managed)

**Cons**

- Costly (and inefficient) under sustained traffic load
- Cold-start problems



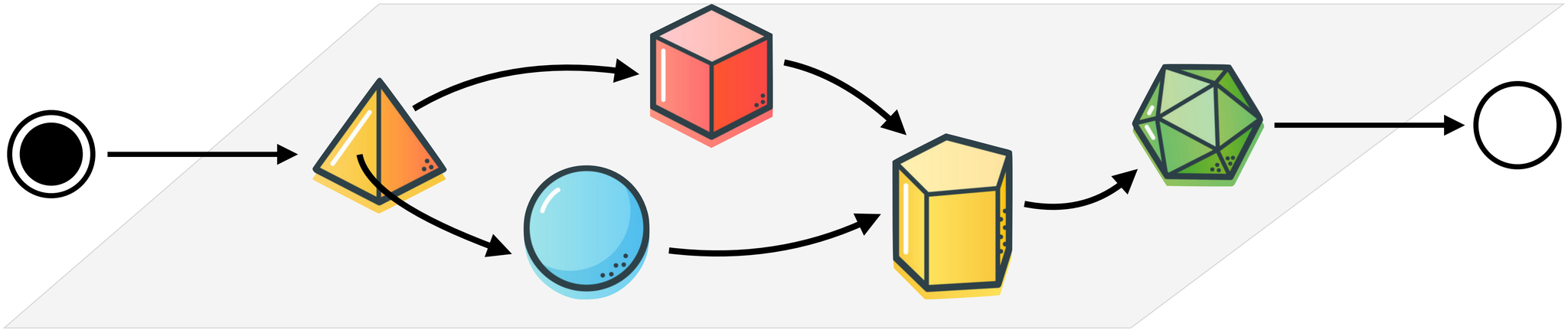
**What if we could have the best of both worlds?**

 Function

 Software Unit

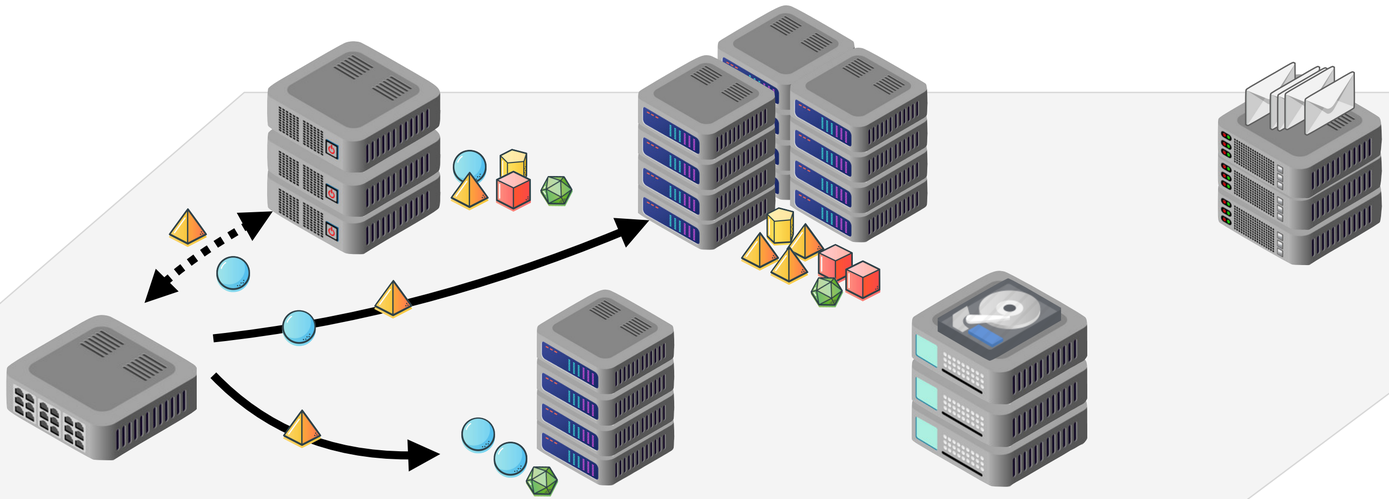
 Runtime Environment

# The Idea



Programming Layer

Jolie



Implementation Layer


One **specification**

```
interface TwiceInterface {
  requestResponse:
   twice( int )( int )
}
```

```
main
{
  twice( number )( result ) {
    result = number * 2
  }
}
```

One **behaviour**

Many **deployments**

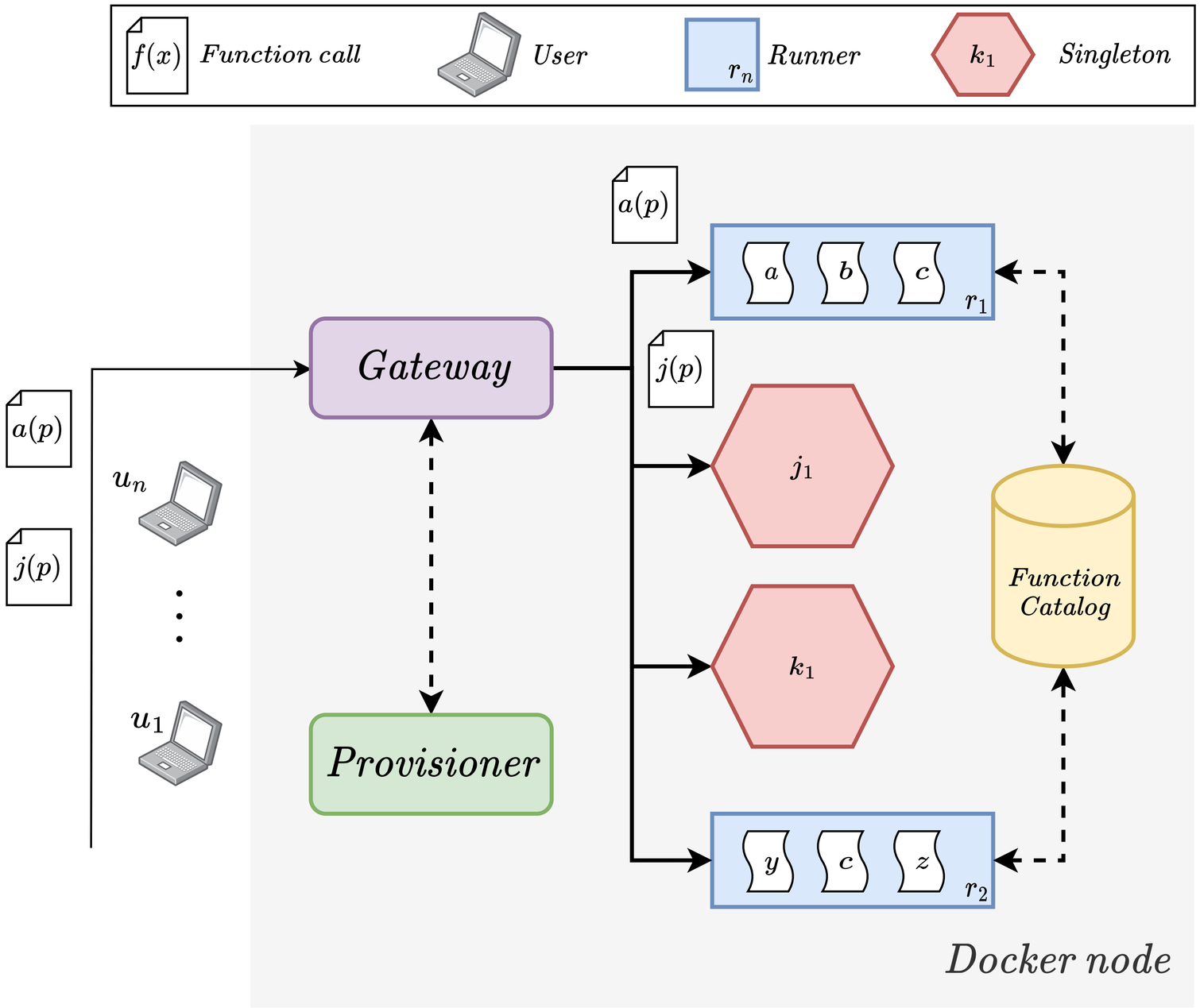
```
inputPort BluetoothPort {
  location: "bluetooth://…"
  protocol: JSON/RPC
  interfaces: TwiceInterface
}
```

```
inputPort WebServicePort {
  location: "socket://…"
  protocol: http
  interfaces: TwiceInterface
}
```

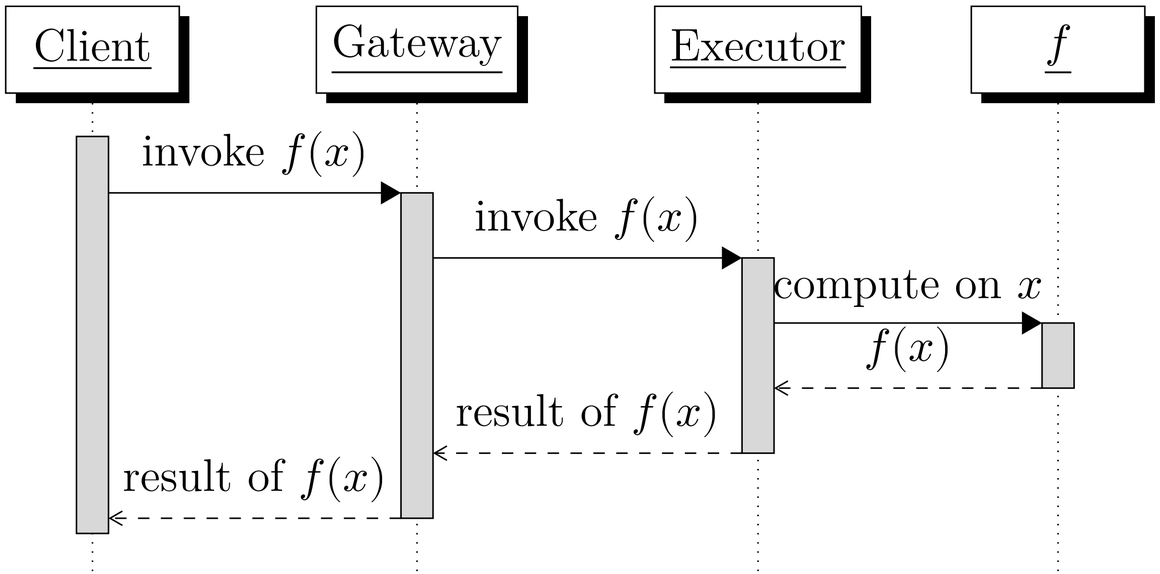
```
inputPort LocalPort {
  location: "local"
  interfaces: TwiceInterface
}
```

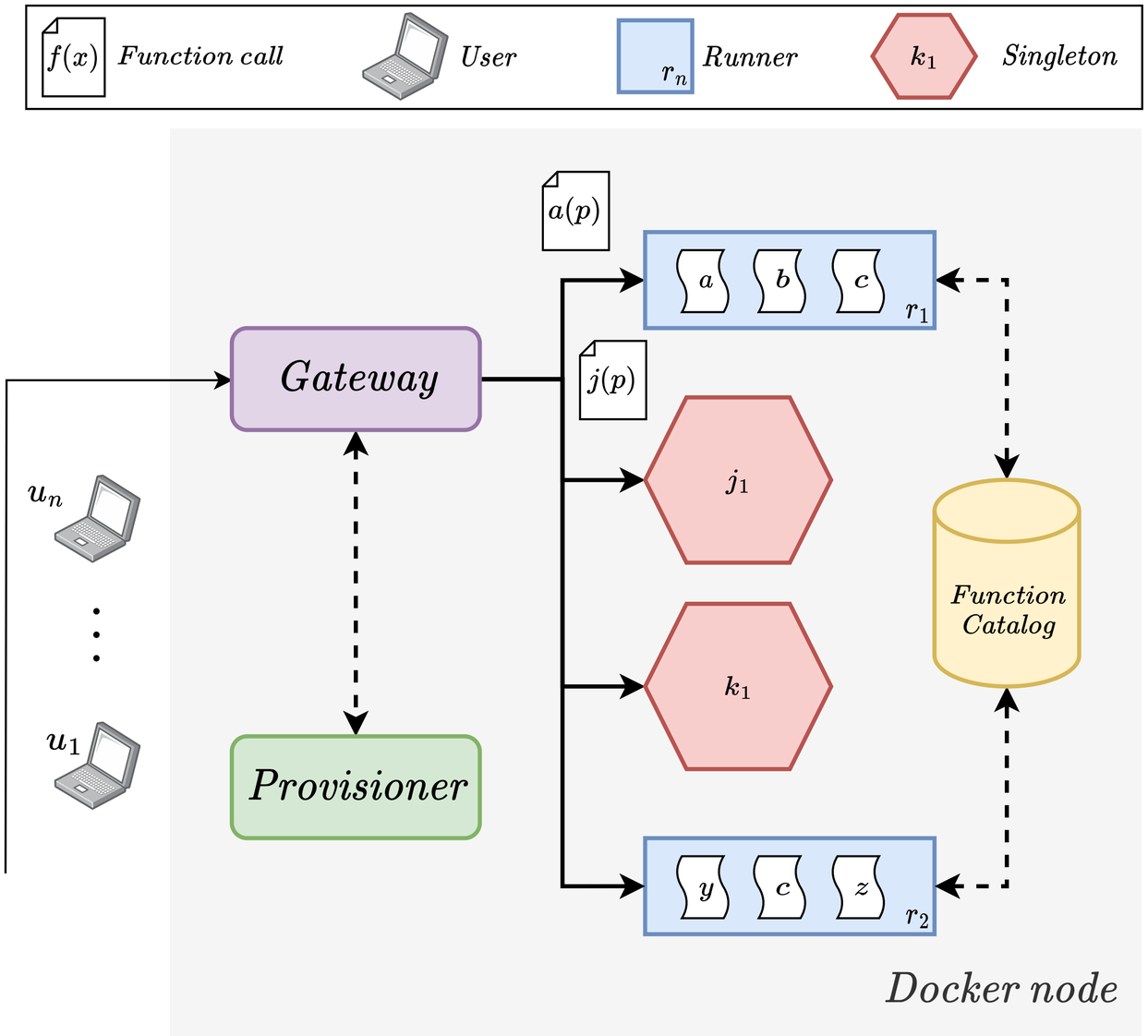
```
inputPort IOTPort {
  location: "socket://myhost:8000"
  protocol: mqtt {
    broker = "socket://broker.com:1883"
   }
  interfaces: TwiceInterface
}
```

# Jolie Functions (JFN)

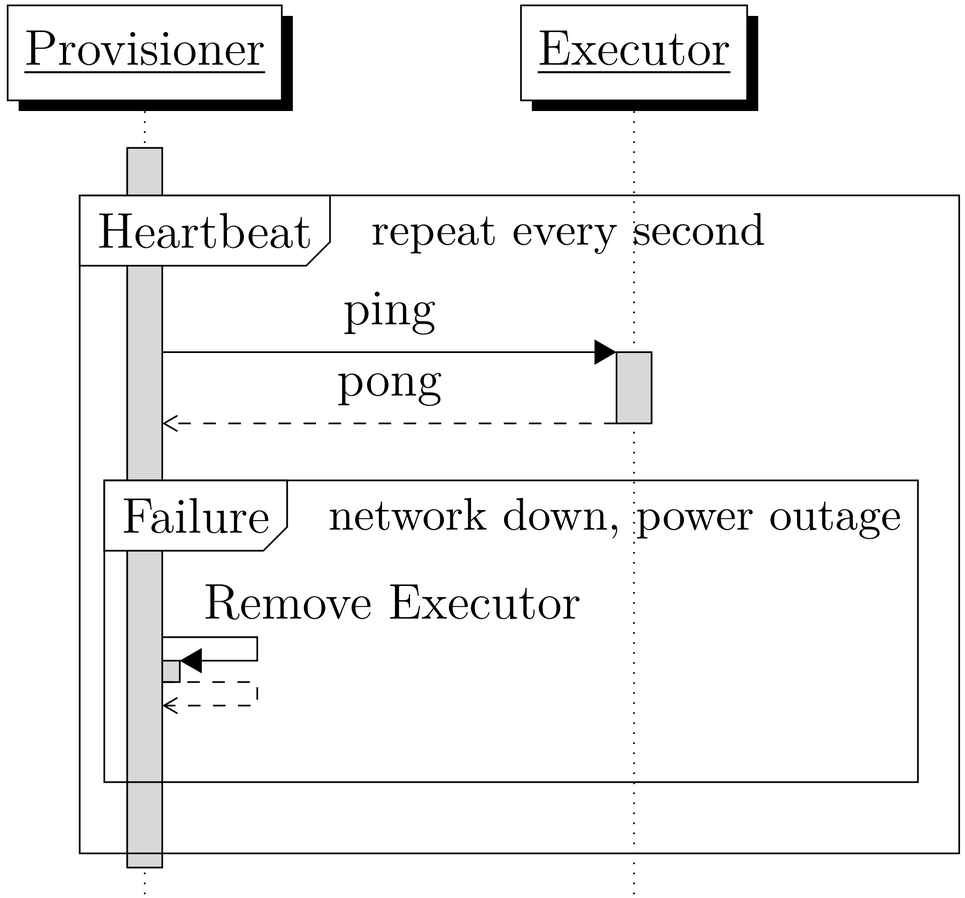


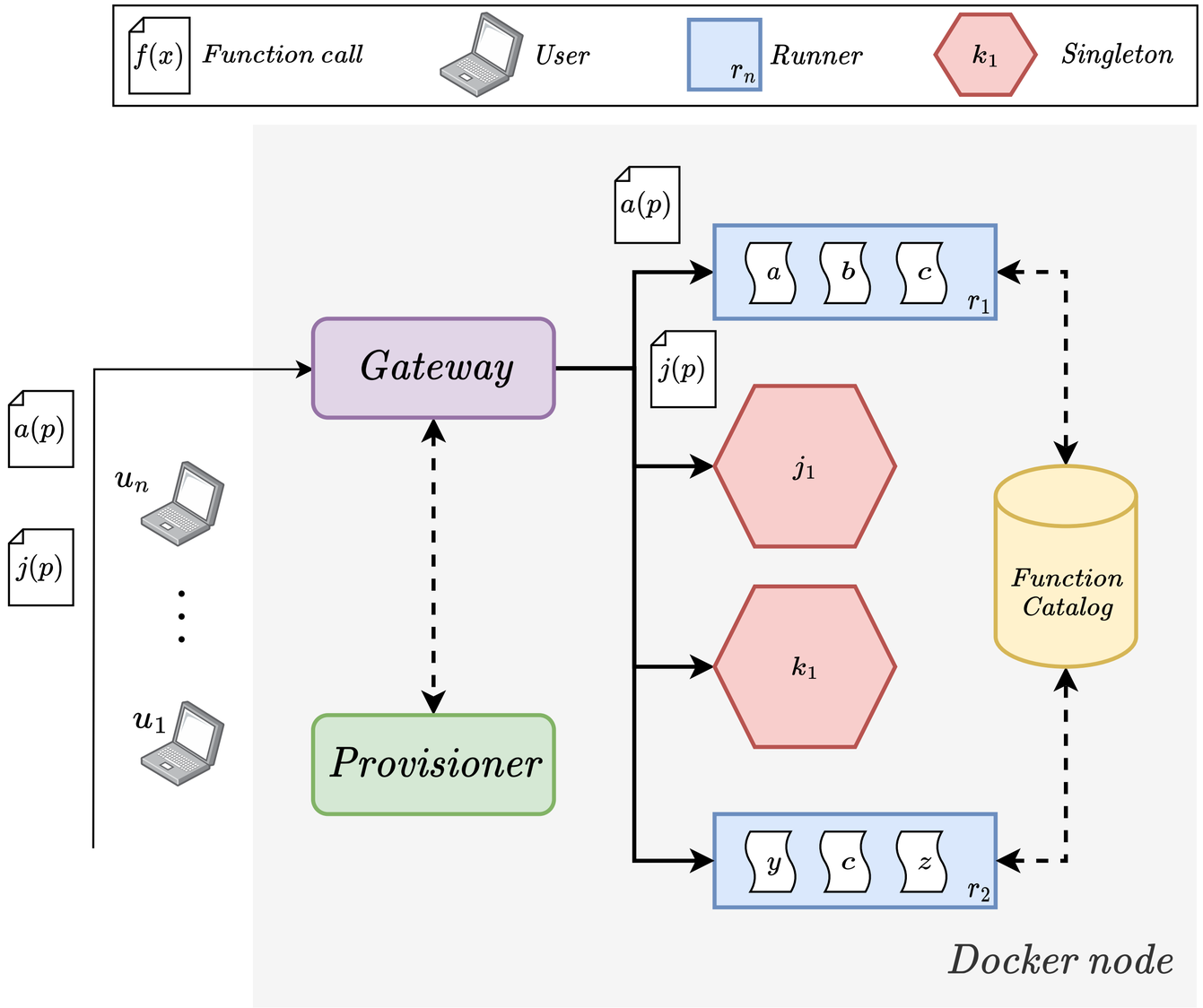
# JFN Flows


Client
Gateway
Executor
$f$
invoke $f(x)$
invoke $f(x)$
compute on $x$
$f(x)$
result of $f(x)$
result of $f(x)$

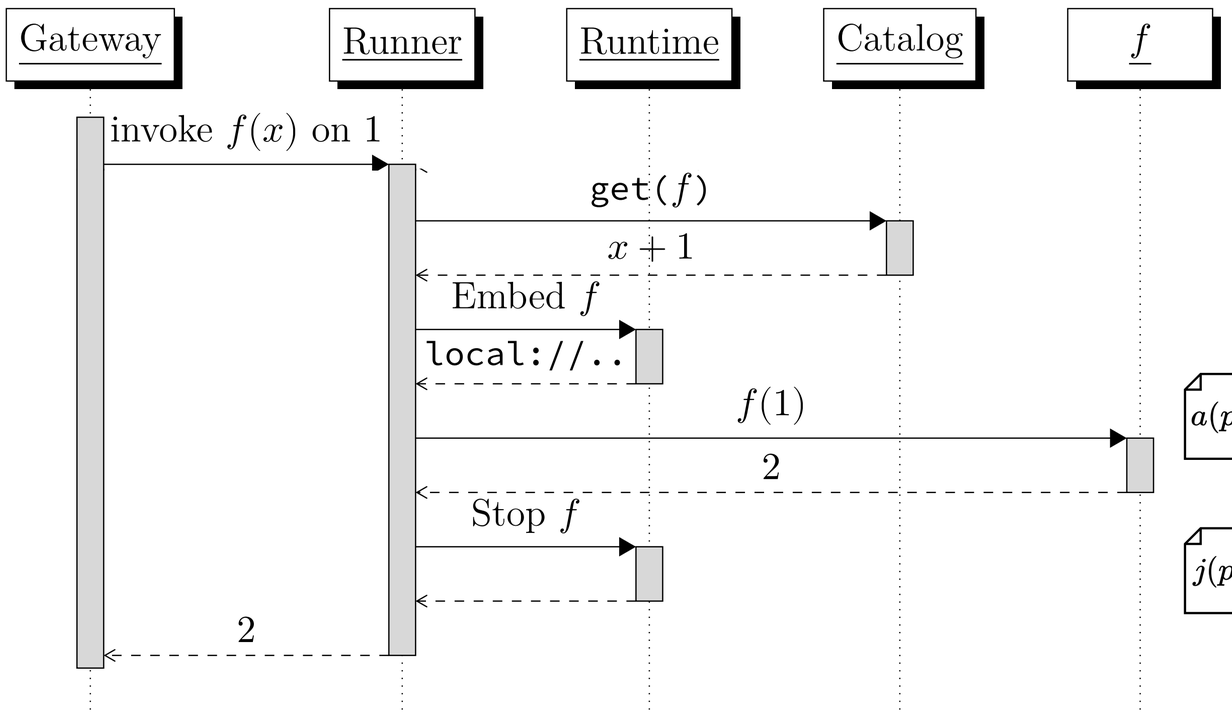

$f(x)$ Function call
User
$r_n$ Runner
$k_1$ Singleton
$a(p)$
$u_n$
$j(p)$
$u_1$
Gateway
Provisioner
$a(p)$
$a$ $b$ $c$ $r_1$
$j(p)$
$j_1$
$k_1$
Function Catalog
$y$ $c$ $z$ $r_2$
Docker node

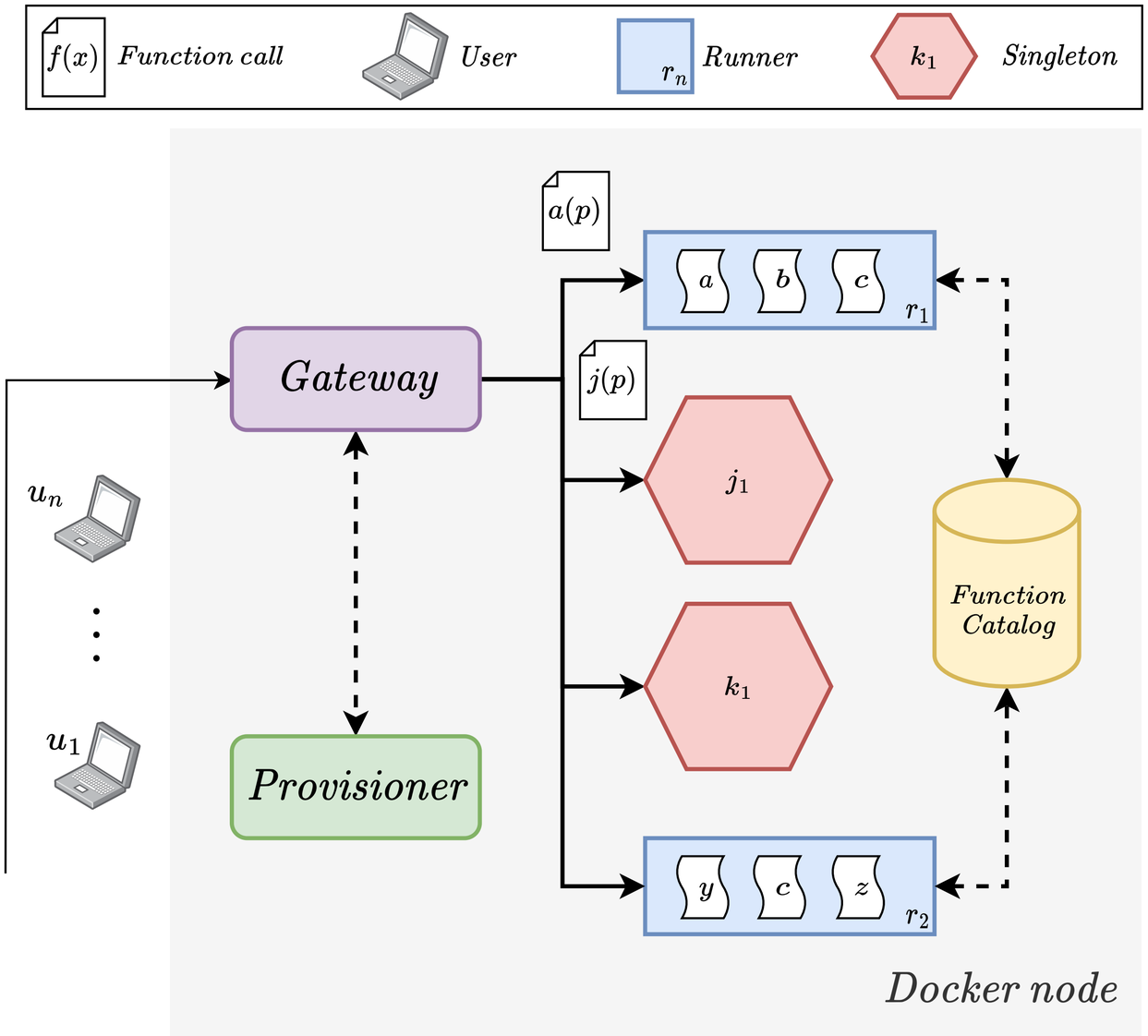
# JFN Flows


Provisioner
Executor
Heartbeat
repeat every second
ping
pong
Failure
network down, power outage
Remove Executor


f(x) Function call
User
r_n Runner
k_1 Singleton
a(p)
u_n
j(p)
u_1
Gateway
Provisioner
a(p)
a
b
c
r_1
j(p)
j_1
k_1
Function Catalog
y
c
z
r_2
Docker node

# JFN Flows


Gateway
Runner
Runtime
Catalog
$f$
invoke $f(x)$ on 1
get($f$)
$x+1$
Embed $f$
local://..
$f(1)$
2
Stop $f$
2


$f(x)$ Function call
User
$r_n$ Runner
$k_1$ Singleton
$a(p)$
$j(p)$
$u_n$
$u_1$
Gateway
Provisioner
$a$ $b$ $c$ $r_1$
$j_1$
$k_1$
$y$ $c$ $z$ $r_2$
Function Catalog
Docker node

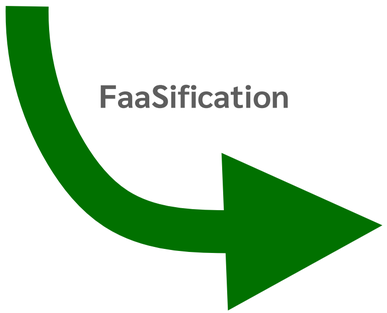
# JFN: From Microservices to Functions

```
[sort(array)(sorted){
  mid = (array.end - array.start) / 2
  sort@Self({ data << array.data, start = data.start, end
= mid })(array.data)
  sort@Self({ data << array.data, start = mid+1, end = end
})(array.data)
  merge@Self(array)(sorted)
}]
```

1. Identify the operation to expose as a function (e.g., **sort**)

2. Create a JFN function where:

- its body is the body of the operation (making sure to map the input/outputs)
- all calls to operations which have been moved to a separate function as follows:




```
[fn(request)(response){
  mid = (request.data.end - request.data.start) / 2
  op@Gateway({ op = "sort", data << {
    data << request.data.data,
    start = request.data.start,
    end = mid
  } })
  op@Gateway({ op = "sort", data << {
    data << request.data.data,
    start = mid+1,
    end = request.data.end
  } })
  op@Gateway({ op = "merge", data << { data << request.
data.data } })
```

# **Future Work** (implementation)

Increase the **scalability** of the architecture. Immediate targets:

- architectural: **Function Catalog** and **Provisioner**
- infrastructural: Kubernetes

Support the deployment of microservice packages (JAPs) to run:

- multi-file Jolie microservices
- Java and JavaScript microservices


THANK
YOU