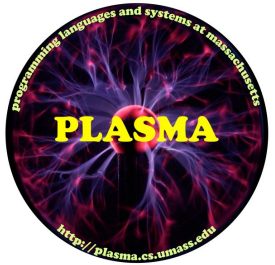
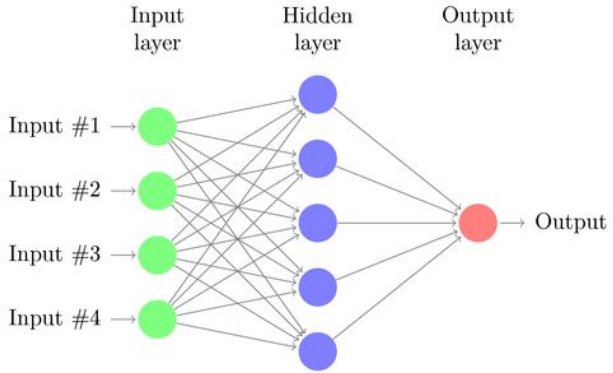


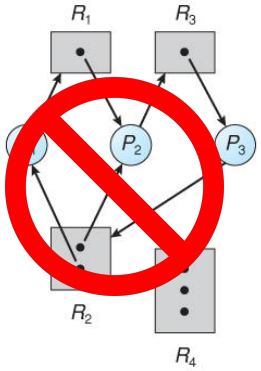
Formal Foundations of Serverless Computing

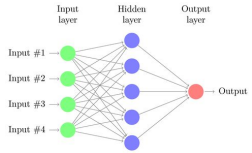
Abhinav Jangda, Donald Pickney, Samuel Baxter, Joseph Spitzer, Breanna Devore-McDonald, Yuriy Brun, Arjun Guha

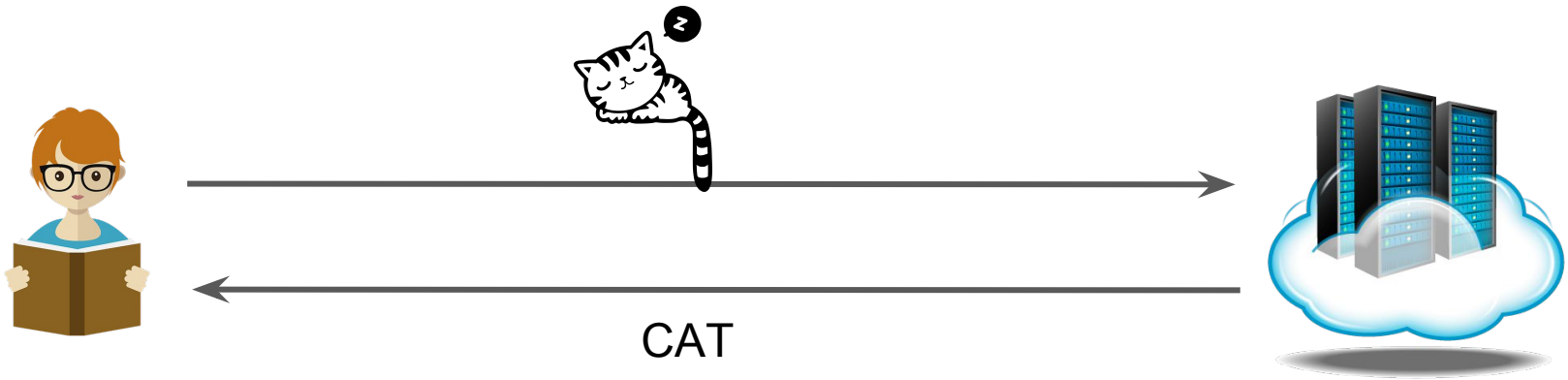

Input layer
Hidden layer
Output layer
Input #1
Input #2
Input #3
Input #4
Output


PyTorch


Xen™


R_1
R_3
P_2
P_3
R_2
R_4


Input layer
Hidden layer
Output layer
Input #1
Input #2
Input #3
Input #4
Output


CAT

Serverless Platforms

Google Cloud Platform

Azure

Supported Languages

Swift

```
model = CatOrDogModel ();
function catOrDog (req, res) {
     if (req.type === train) {
          model.train (req.data);
     } else {
          res.write (model.predict (req.data));}}
```

Store trained model

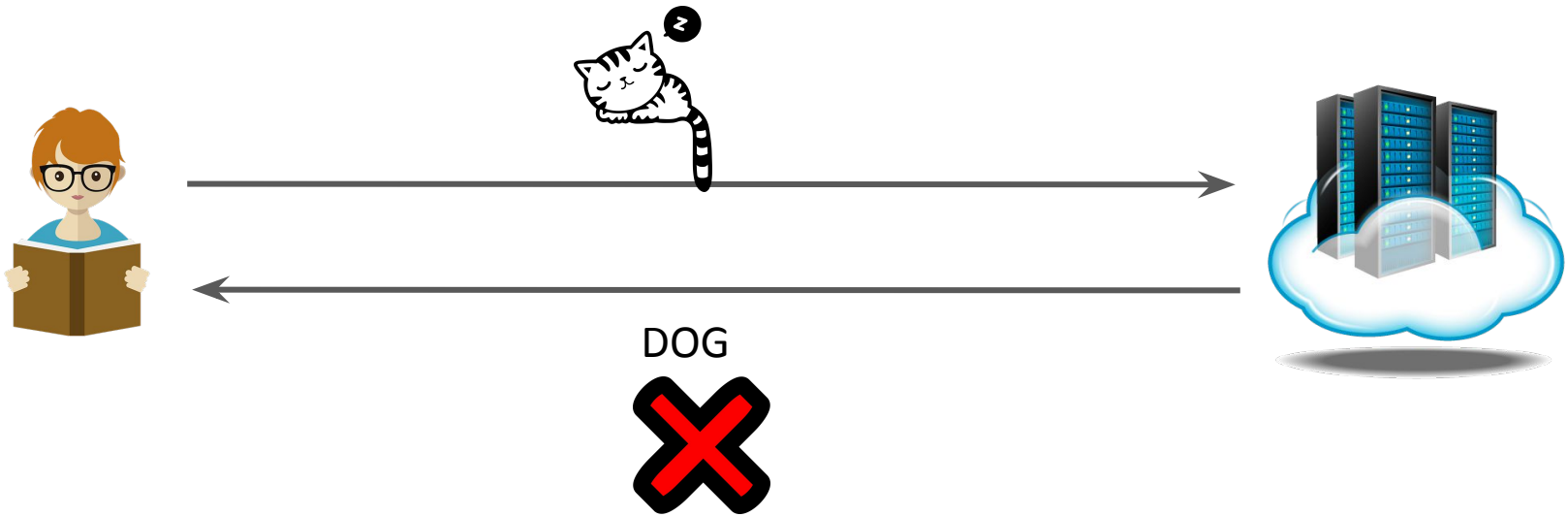
Takes a HTTP Req

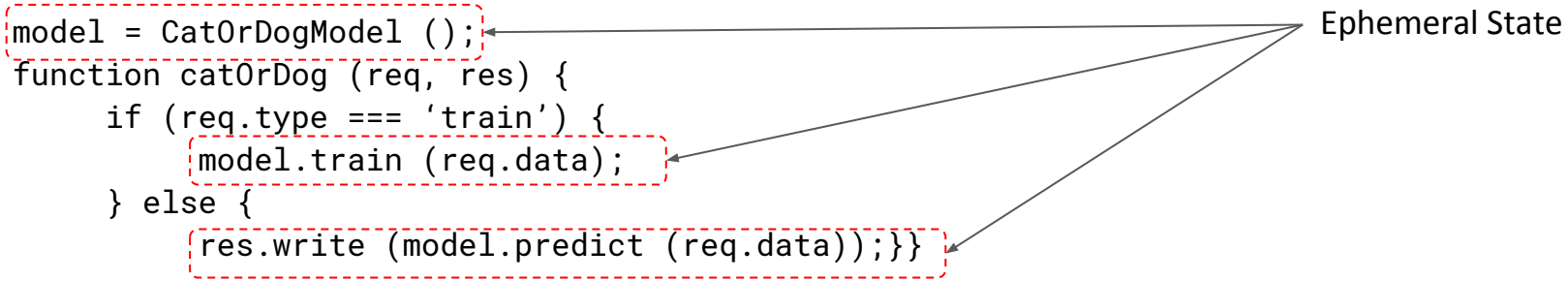
Train Model

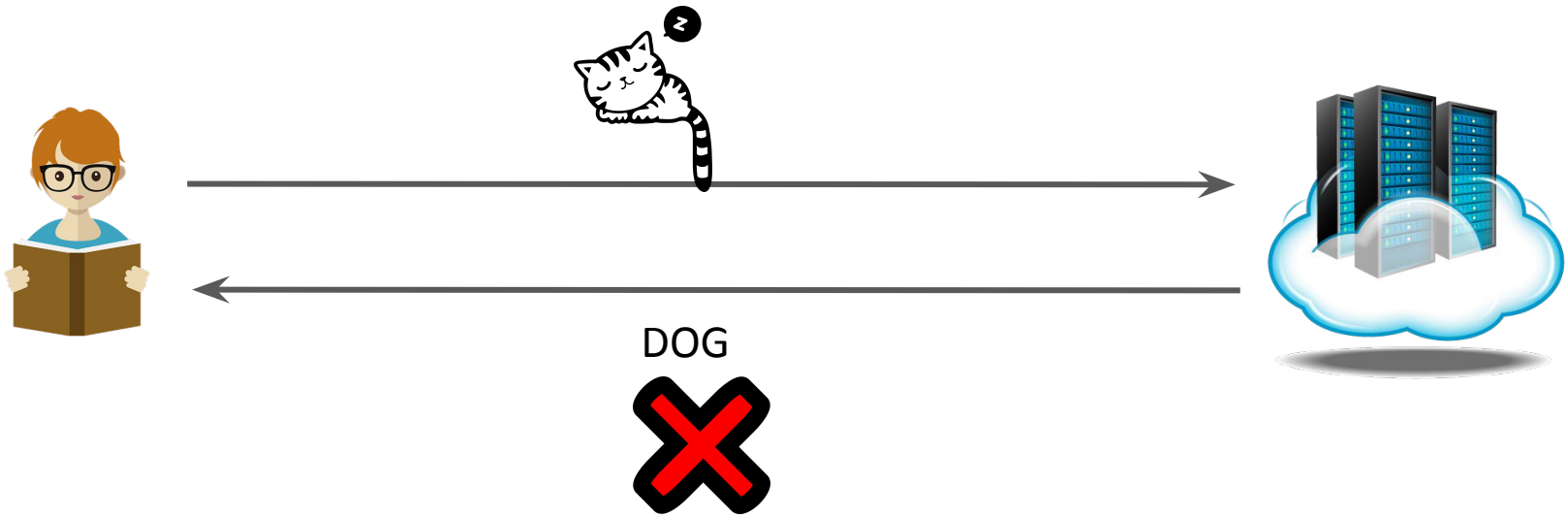
Predict Cat or Dog

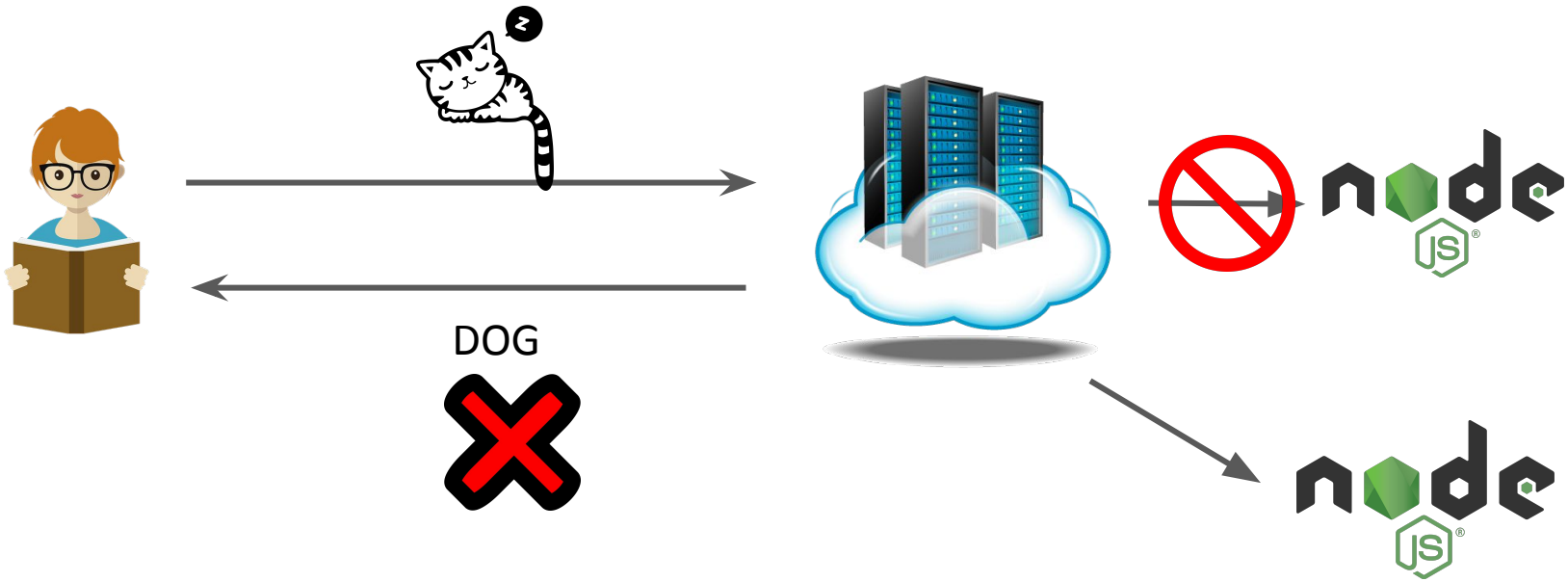
Play golf dataset

Independent variables				Dep. var
OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY
sunny	85	85	FALSE	Don't Play
sunny	88	90	TRUE	Don't Play
overcast	83	78	FALSE	Play
rain	70	96	FALSE	Play
rain	68	80	FALSE	Play
rain	65	70	TRUE	Don't Play
overcast	64	65	TRUE	Play
sunny	72	95	FALSE	Don't Play
sunny	69	70	FALSE	Play
rain	75	80	FALSE	Play
sunny	75	70	TRUE	Play
overcast	72	90	TRUE	Play
overcast	81	75	FALSE	Play
rain	71	80	TRUE	Don't Play


DOG

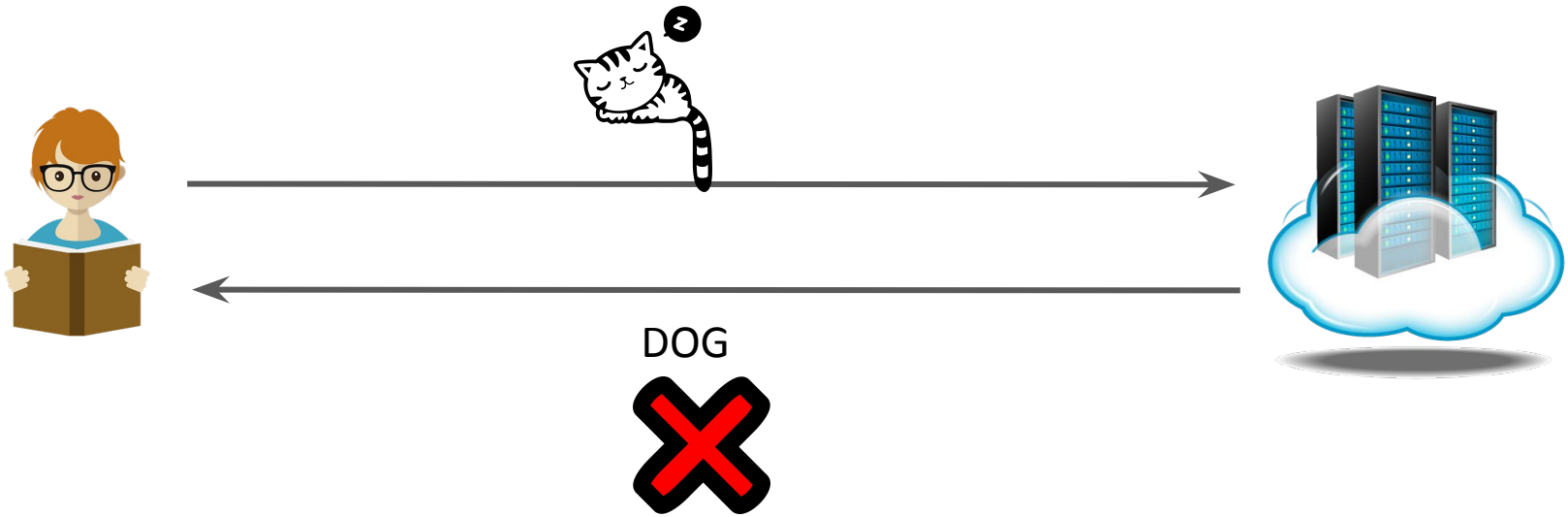

model = CatOrDogModel ();
function catOrDog (req, res) {
if (req.type === 'train') {
model.train (req.data);
} else {
res.write (model.predict (req.data));}}
Ephemeral State

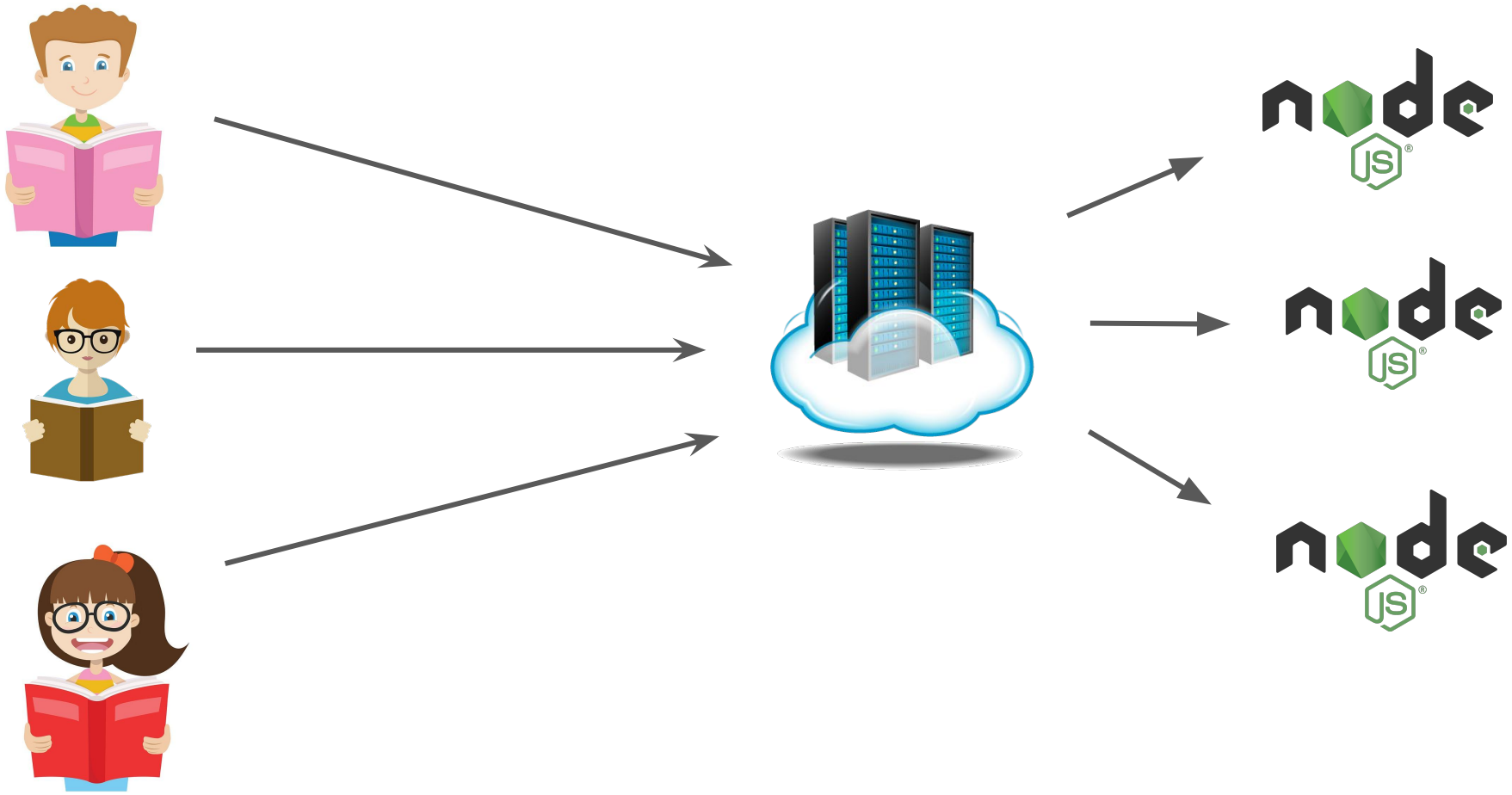

DOG


DOG

```
model = CatOrDogModel ();
function catOrDog (req, res) {
     if (req == ‘train’) {
          model.train (req.data);
     } else {
          res.write (model.predict (req.data));}}
```

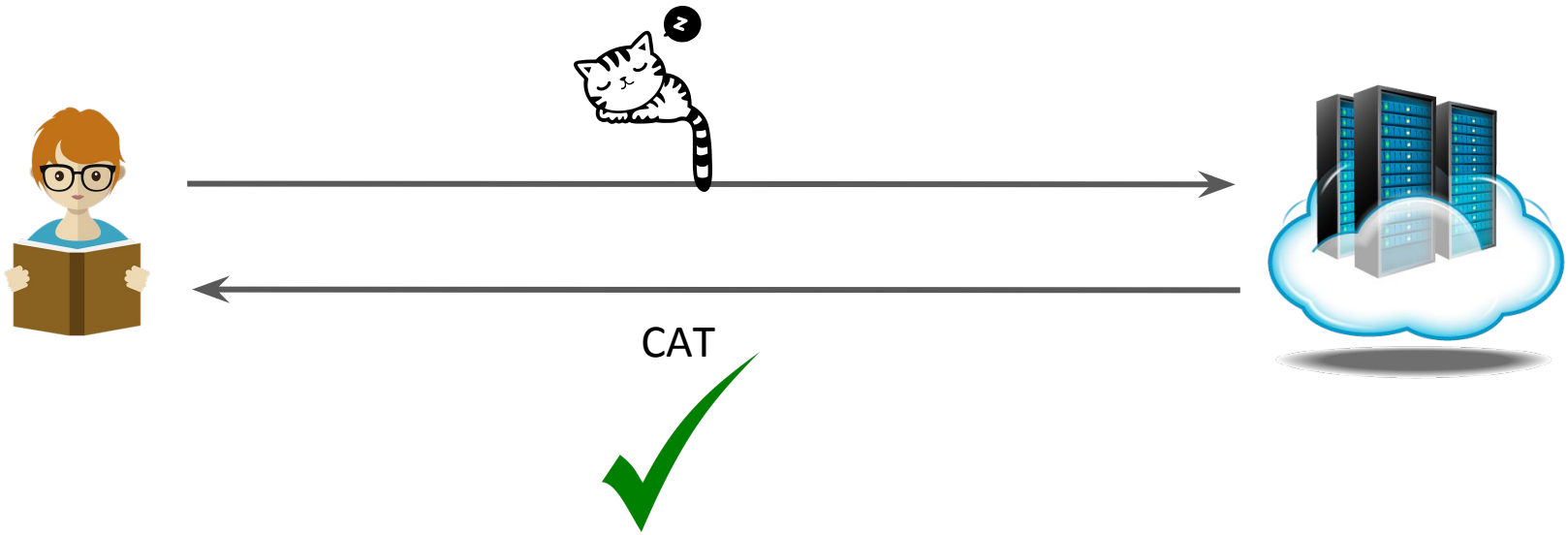
Single event processed to completion multiple times


DOG


node
JS
node
JS
node
JS

```
model = CatOrDogModel ();
function catOrDog (req, res) {
     if (req.type === 'train') {
          model.train (req.data);
     } else {
          res.write (model.predict (req.data));}}
```

More than one instances of function can be running in parallel.


CAT

```
let Datastore = require('@google-cloud/datastore');
Function catOrDog(req, res) {
  let ds = new Datastore({ projectId: 'cat-app' });
  let dst = ds.transaction();
  dst.run(function() {
    let tId = ds.key(['Transaction', req.body.transId]);
    dst.get(tId, function(err, trans) {
    if (err || trans) {
      dst.rollback(function() { res.send(false); });
    } else if (req.body.type === 'train') {
      let data = ds.key(['model', req.body.data]);
      dst.get(data, function(err, model) {
      model.train (data)
      dst.save({ key: data, data: model });
      dst.save({ key: tId, data: {} });
      dst.commit(function() { res.send(true); });});});
    } else if (req == 'test') {
      let data = ds.key(['prediction', req.body.data]);
      dst.get(data, function(err, model) {
      prediction = model.predict (data)
      dst.save({ key: data, data: prediction });
      dst.save({ key: tId, data: {} });
      dst.commit(function() { res.send(prediction); });});});
    } else {
        dst.rollback(function(){res.send(false);});});}});}});};
```

=

```
model = CatOrDogModel ();
function catOrDog (req, res) {
      if (req.type == 'train') {
            model.train (req.data);
      } else {
            res.write (model.predict (req.data));}}
```

x 4

Serverless Programming is Hard

- Challenges
  - Ephemeral State
  - Need for transactions
  - Concurrency
- Need a deep understanding of serverless platforms to provide robust tools to programmers for writing correct code.

“Serverless functions are not functions in an ordinary sense”

Our Contributions

- Operational semantics to model essential details of serverless platforms.
- Three case studies to show that the semantics is useful:
  - Idealized Semantics
  - Key Value Store
  - Serverless Programming Language for composing functions

Operational Semantics of Serverless System

$$\textsc{Req}\ \frac{x \text{ is fresh}}{\mathcal{C} \xrightarrow{\mathsf{start}(x,v)} \mathcal{C}\mathbb{R}(f,x,v)}$$

$$\textsc{Cold}\ \frac{y \text{ is fresh} \qquad \mathsf{recv}(v,\sigma_0) = \sigma}{\mathcal{C}\mathbb{R}(f,x,v) \Rightarrow \mathcal{C}\mathbb{R}(f,x,v)\mathbb{F}(f,\mathbf{busy}(x),\sigma,y)}$$

$$\textsc{Warm}\ \frac{\mathsf{recv}(v,\sigma) = \sigma'}{\mathcal{C}\mathbb{R}(f,x,v)\mathbb{F}(f,\mathbf{idle},\sigma,y) \Rightarrow \mathcal{C}\mathbb{R}(f,x,v)\mathbb{F}(f,\mathbf{busy}(x),\sigma',y)}$$

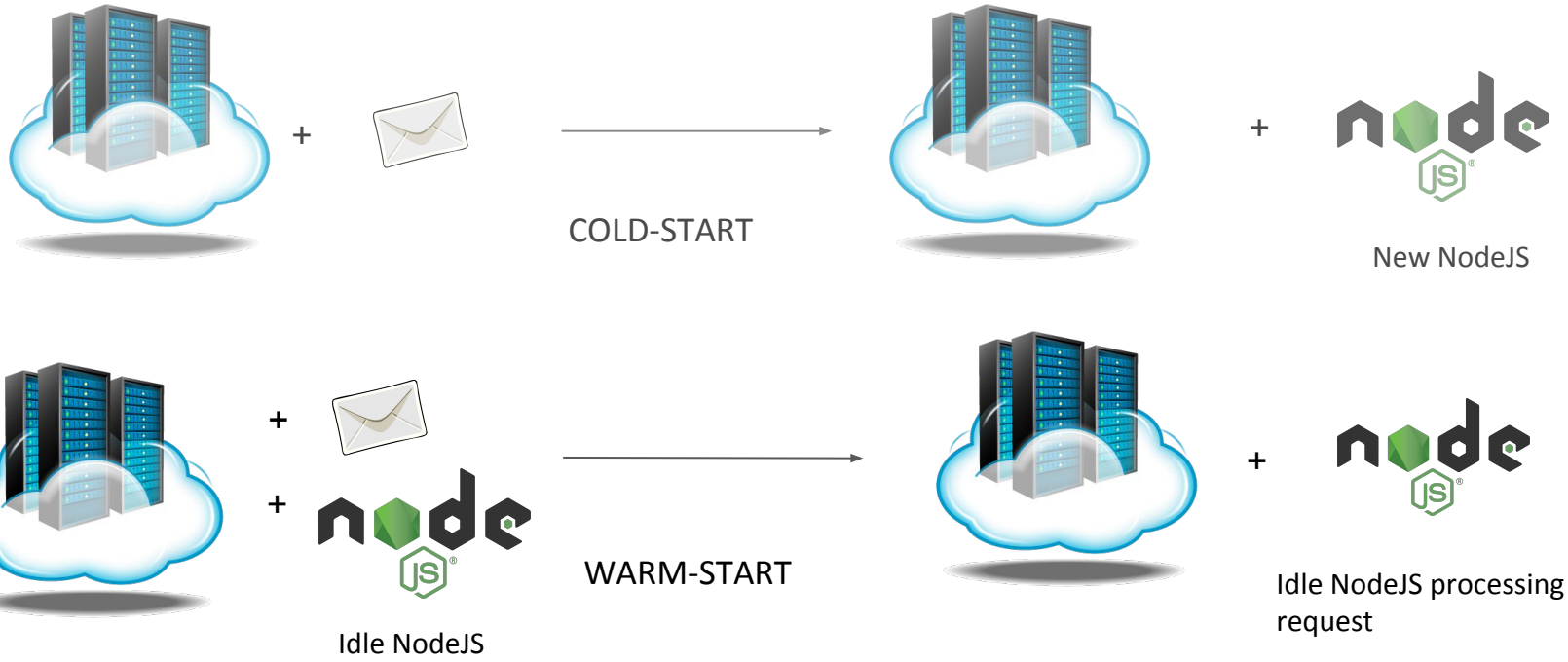
$$\textsc{Hidden}\ \frac{\mathsf{step}(\sigma) = (\sigma',\varepsilon)}{\mathcal{C}\mathbb{F}(f,\mathbf{busy}(x),\sigma,y) \Rightarrow \mathcal{C}\mathbb{F}(f,\mathbf{busy}(x),\sigma',y)}$$

$$\textsc{Resp}\ \frac{\mathsf{step}(\sigma) = (\sigma',\mathbf{return}(v'))}{\mathcal{C}\mathbb{R}(f,x,v)\mathbb{F}(f,\mathbf{busy}(x),\sigma,y) \xrightarrow{\mathsf{stop}(x,v')} \mathcal{C}\mathbb{F}(f,\mathbf{idle},\sigma,y)\mathbb{S}(x,v')}$$

$$\textsc{Die}\ \frac{}{\mathcal{C}\mathbb{F}(f,m,\sigma,y) \Rightarrow \mathcal{C}}$$

Operational Semantics of Serverless System

Operational Semantics of Serverless System



Summary

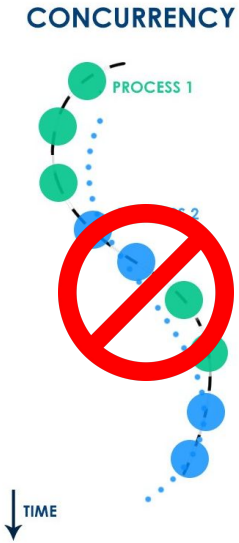
Operational semantics models all details of serverless platforms:

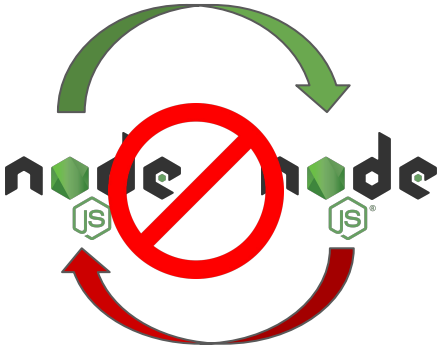
1. New instance creation
2. Instance reuse
3. Ephemeral and persistent state

Ideally: Provide programmers higher-level abstractions

1st Case Study: Idealized Serverless Semantics

Models an idealized serverless platform with:





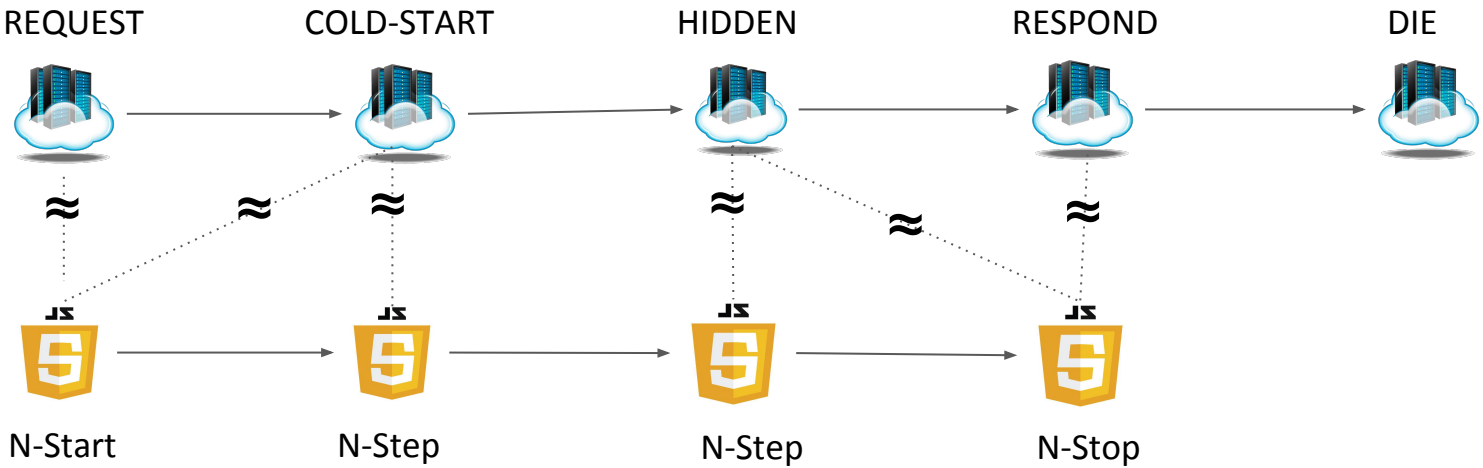
1st Case Study: Idealized Serverless Semantics

$$\textsc{N-Start}\ \frac{x \text{ is fresh} \qquad \mathsf{recv}(v, \sigma_0) = \sigma'}{\langle f, \mathbf{idle}, \vec{\sigma} \rangle \xmapsto{\mathsf{start}(x,v)} \langle f, \mathbf{busy}(x), [\sigma_0, \sigma'] \rangle}$$

$$\textsc{N-Step}\ \frac{\mathsf{step}(\sigma) = (\sigma', \varepsilon)}{\langle f, \mathbf{busy}(x), \vec{\sigma} +\!\!+ [\sigma] \rangle \mapsto \langle f, \mathbf{busy}(x), \vec{\sigma} +\!\!+ [\sigma, \sigma'] \rangle}$$

$$\textsc{N-Stop}\ \frac{\mathsf{step}(\sigma) = (\sigma', \mathbf{return}(v))}{\langle f, \mathbf{busy}(x), \vec{\sigma} +\!\!+ [\sigma] \rangle \xmapsto{\mathsf{stop}(x,v)} \langle f, \mathbf{idle}, [\sigma] \rangle}$$

1st Case Study: Idealized Serverless Semantics



Theorem: If certain conditions are met, then Idealized Semantics is weakly bisimilar to Operational Semantics.

2nd Case Study: Semantics with Key Value Store

+

Key	Value
6515551234	Jon Doe, Pre-Paid, 40.00
AAA999	Mazda, Black, 626
321-651A	Random Data

≈

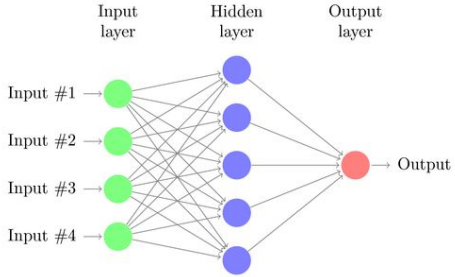
+

Key	Value
6515551234	Jon Doe, Pre-Paid, 40.00
AAA999	Mazda, Black, 626
321-651A	Random Data

3rd Case Study: Serverless Programming Language

1. Extend Operational Semantics with a serverless composition language
2. Inspired by OpenWhisk Conductor and IBM Composer

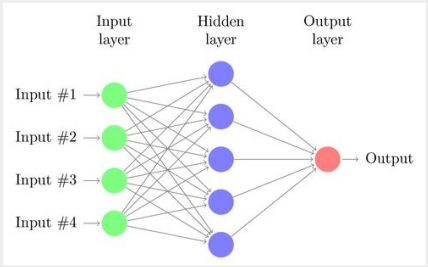
Serverless Compositions



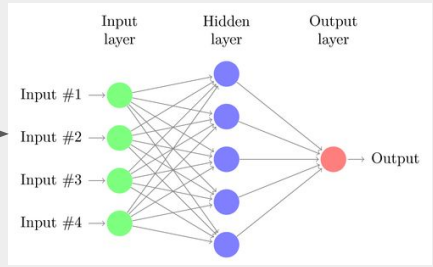
BlackOrWhite

Black

Serverless Compositions



CatOrDog



BlackOrWhite

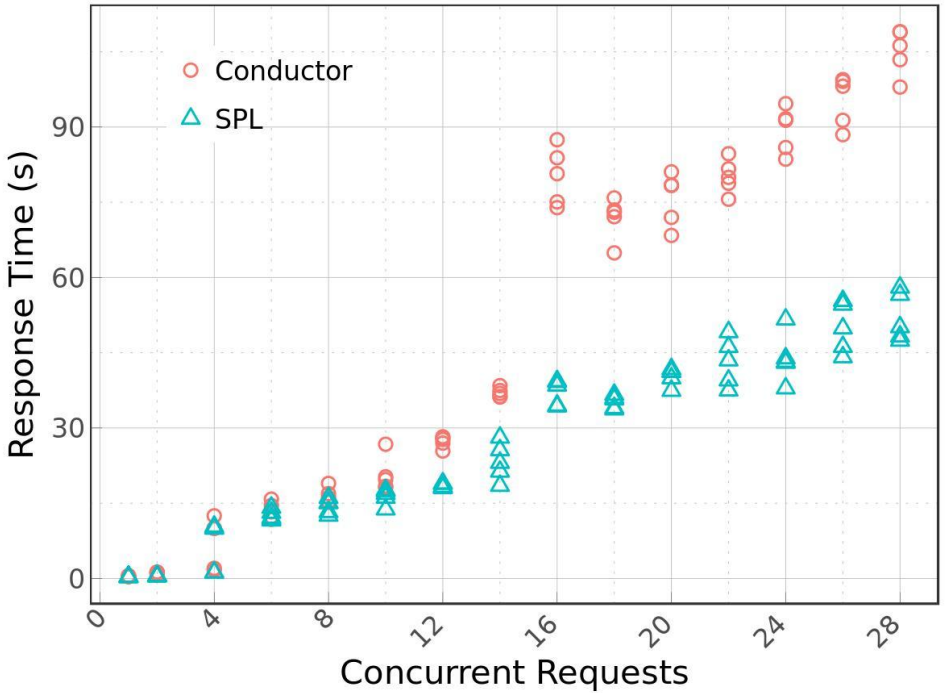
Black Cat

3rd Case Study: Serverless Programming Language

```
a <- invoke catOrDog (in);
return invoke blackOrWhite (a.animal);
```

Serverless Programming Language (SPL)

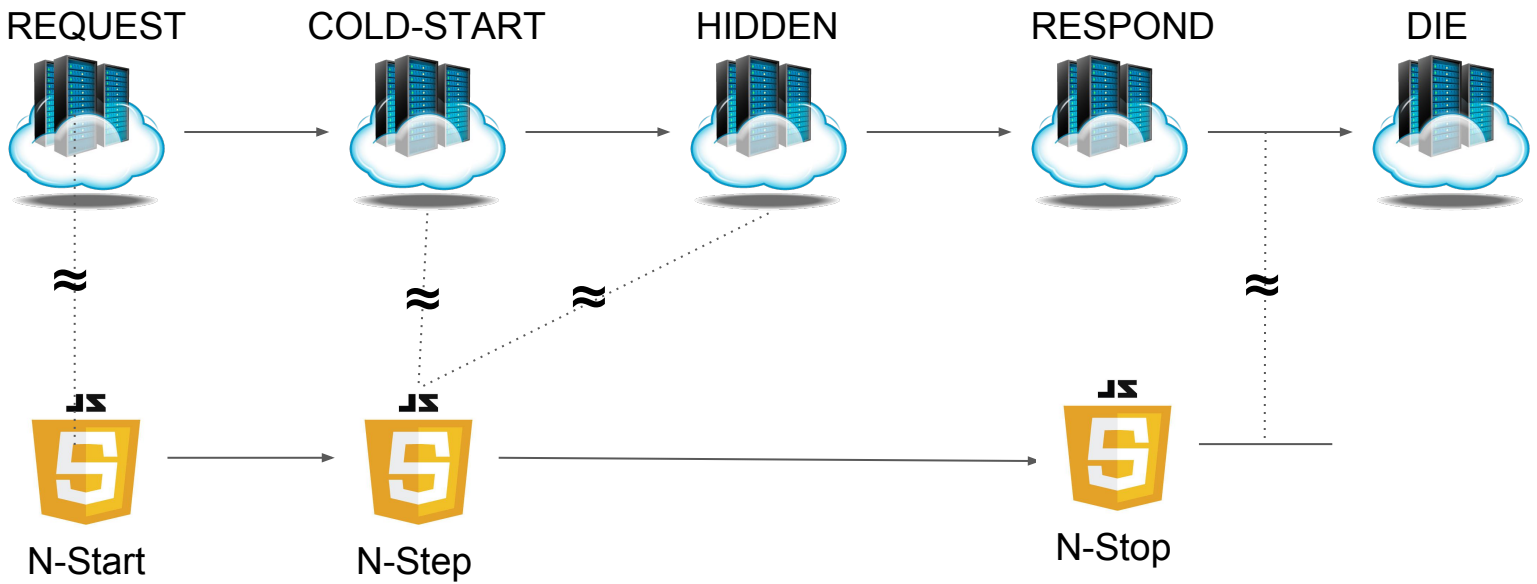
1. OpenWhisk Conductor is executed in a docker container.
2. SPL's JSON Transformation can be executed directly in OpenWhisk Controller.



Contributions

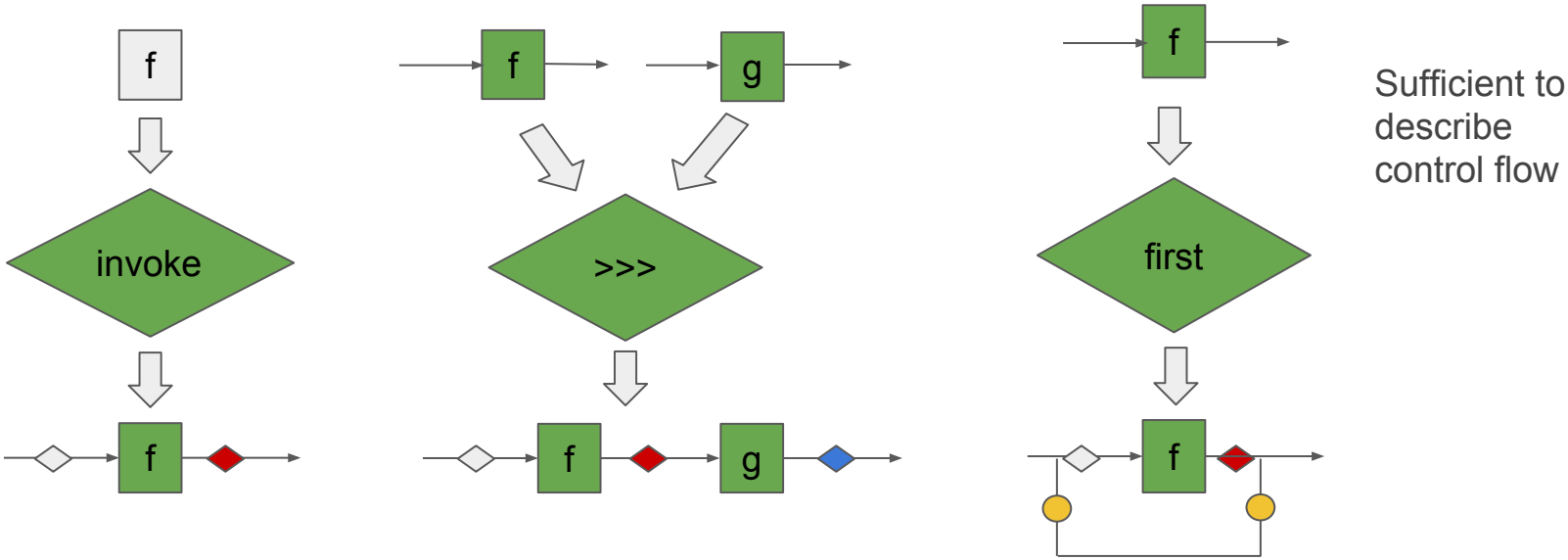
1. Writing correct code for serverless platform is hard
2. We present Operational Semantics of serverless platform
3. We present 3 case studies to show these semantics are useful
   a. Naive Semantics are abstractions of operational semantics
   b. Operational Semantics are extended with a Key Value store
   c. Serverless Programming Language can be used to compose existing serverless functions efficiently.

1st Case Study: Naive Serverless Semantics



Theorem: If certain conditions are met, then Naive Semantics is weakly bisimilar to Operational Semantics.

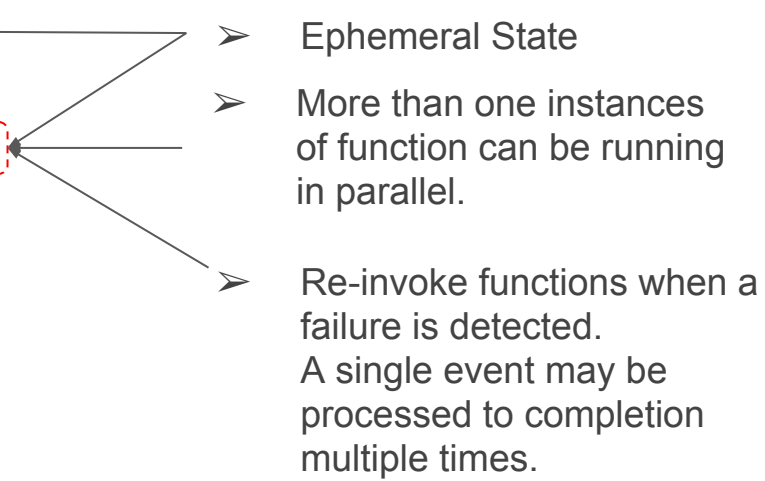
3rd Case Study: Serverless Compositions



Sufficient to describe control flow

Issues

```
let accounts = new Map();
exports.bank = function(req, res) {
      accounts.set(req.body.name, req.body.value);
       res.send(true);
  };
```



Operational Semantics of Serverless System

- Based on experiences with major serverless computing platforms.

Operational Semantics of Serverless System

\+ $R(f, id, v)$

COLD-START

\+ $R(f, id, v)$
\+ $F(f, busy(id), \sigma)$

A new Function instance of *f* with initial state σ for *id* has started

\+ $R(f, id, v)$
\+ $F(f, idle, \sigma)$

WARM-START

\+ $R(f, id, v)$
\+ $F(f, busy(id), \sigma)$

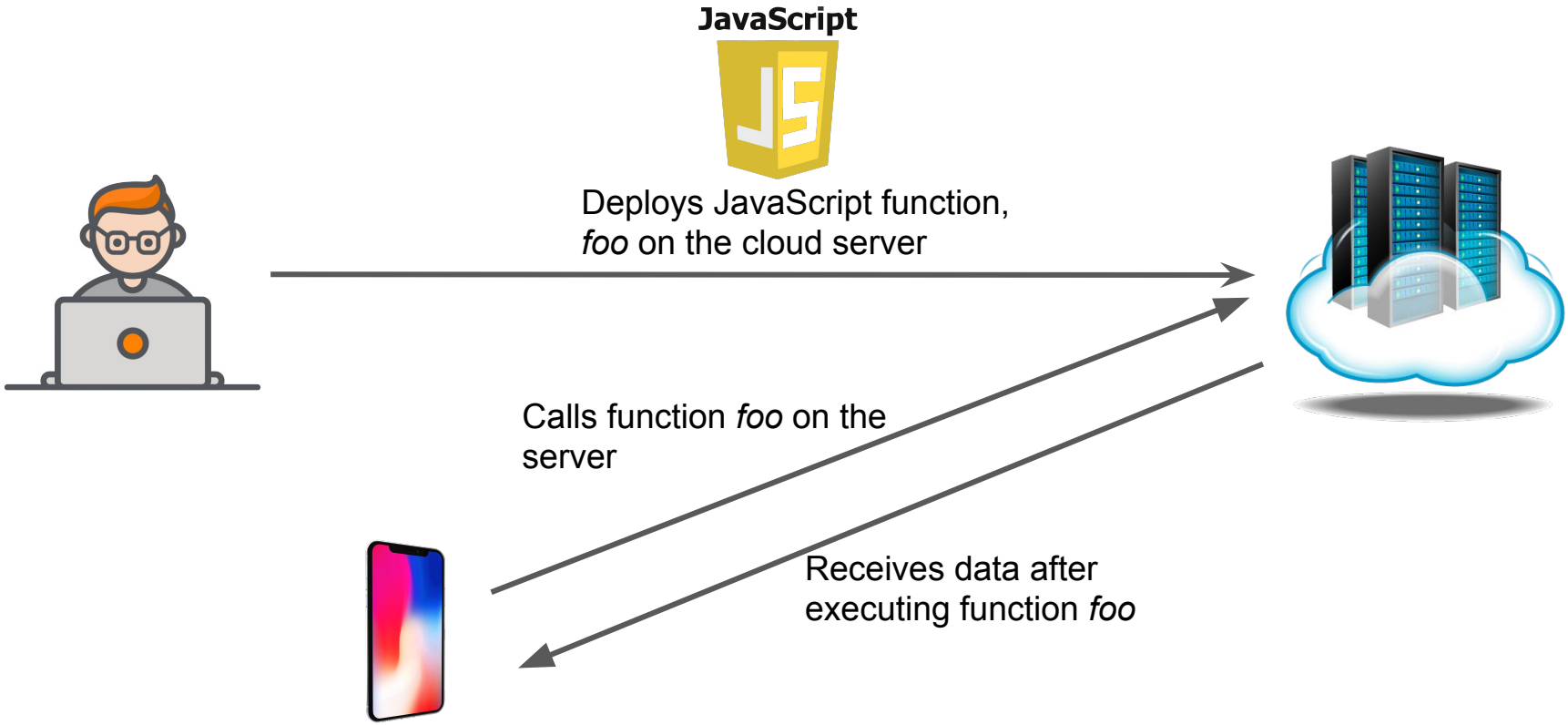
Start executing *f* on an *idle* function instance

Operational Semantics of Serverless System

+ $F(f, m, \sigma)$

DIE

Introduction to Serverless Computing



- Focus on application code only

- Compiles code
- Configures OS
- Manages resource allocation.

Serverless Function Example

```
let accounts = new Map();
exports.bank = function(req, res) {
        accounts.set(req.body.name, req.body.value);
        res.send(true);
    };
```

Store account data

Takes a HTTP Req

Deposit the amount

Serverless Programming Language

```
a <- invoke f (in);
return invoke g (a.x);
```

```
[in, {input: in}] >>> first (invoke f) >>> [a.x, {a: in, input: in}] >>>
first (invoke g) >>> in[0]
```

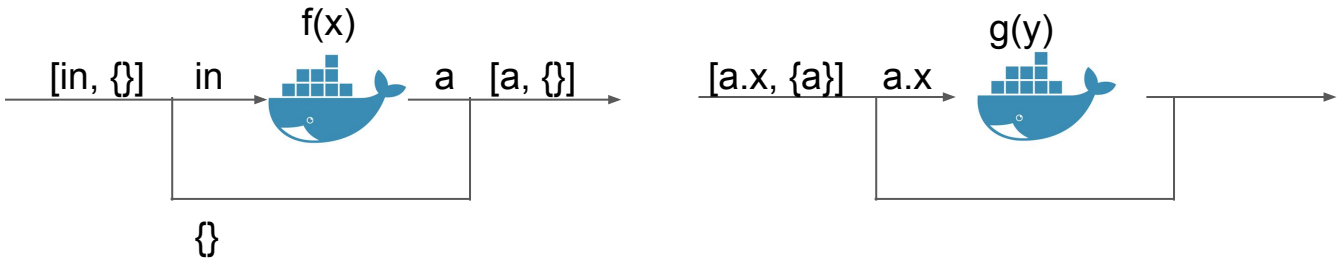
`in` is the input to the composition

First element of array as the input to the composition

Perform JSON Transformations to store the state of composition

Serverless Programming Language

```
a <- invoke f (in);
return invoke g (a.x);
```



Operational Semantics of Serverless System

\+ $R(f, id, v)$
\+ $F(f, busy(id), \sigma)$

RESPOND

\+ $F(f, idle, \sigma)$

Function instance has turned *idle*.

Operational Semantics of Serverless System

+ *F*(*f*, *idle*, σ)

S(*result*)

RESPOND

Operational Semantics of Serverless System

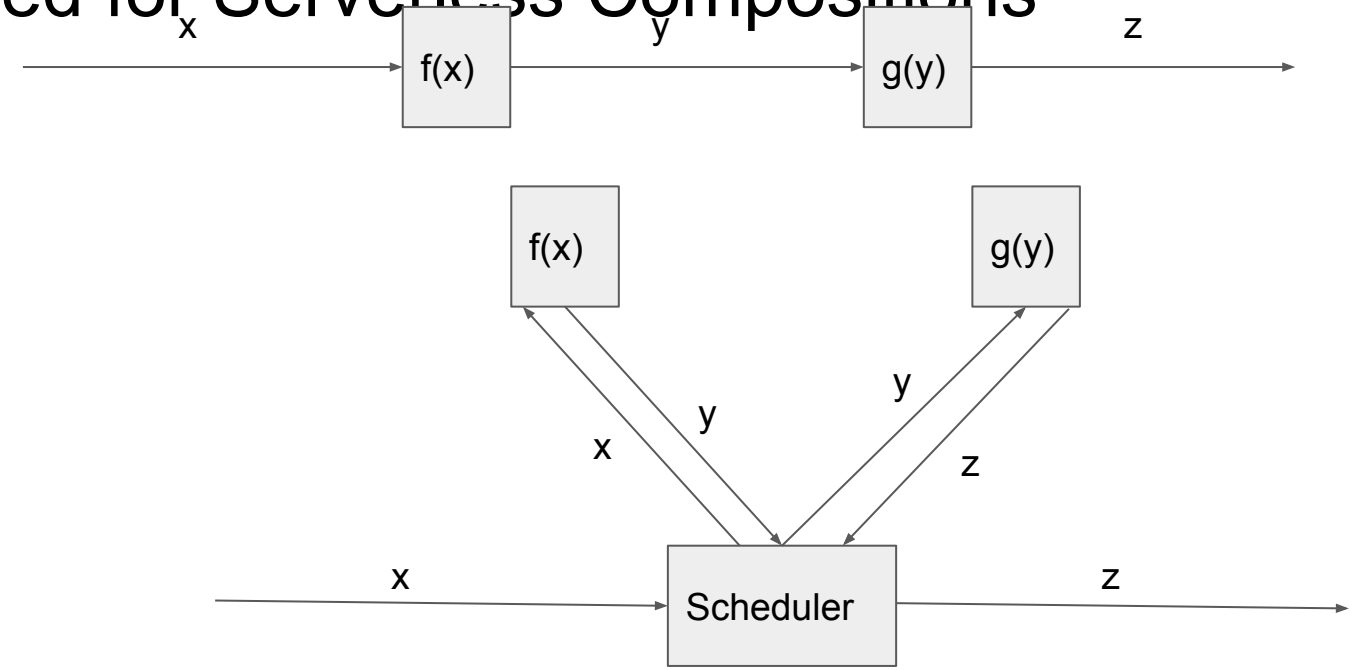
$+ \quad R(f, id, v)$
$+ \quad F(f, \ idle, \ \sigma)$

$+ \quad R(f, id, v)$
$+ \quad F(f, \ busy(id), \ \sigma)$

Start executing *f* on an *idle* function instance

WARM-START

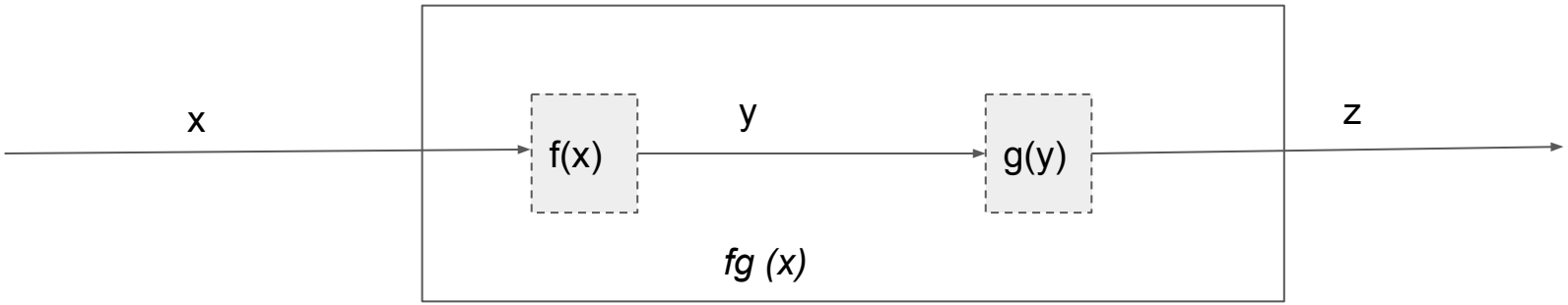
Need for Serverless Compositions



Scheduler spends most of the time idle. Leading to double billing

Need for Serverless Compositions

Merge f and g into one function



1. Hinders code-reuse
2. Does not work when
   a. source code is unavailable or
   b. the serverless functions are written in different languages

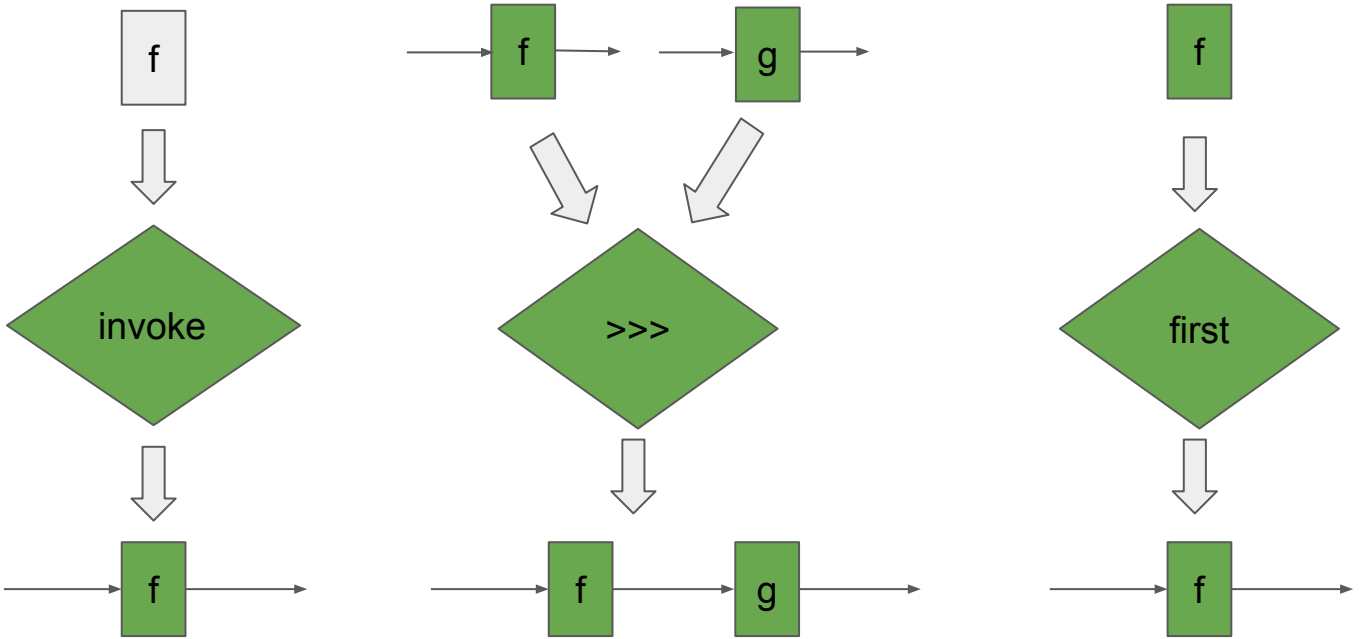
Need for Serverless Compositions

The solution should

- not lead to double billing
- does not hinders code-reuse
- be language agnostic

Serverless Programming Language

Composition primitives for serverless platforms based on Haskell Arrows



Sufficient to describe control flow

Serverless Programming Language

Implementations:

1. Integrated in a fork of OpenWhisk
2. Portable implementation to communicate with other public cloud providers

Serverless Programming Language

1. Comparison to OpenWhisk Conductor
2. Increasing Request Sizes
3. 6-core Intel Xeon E5-1650 with 64 GB RAM with Hyper-Threading enabled.