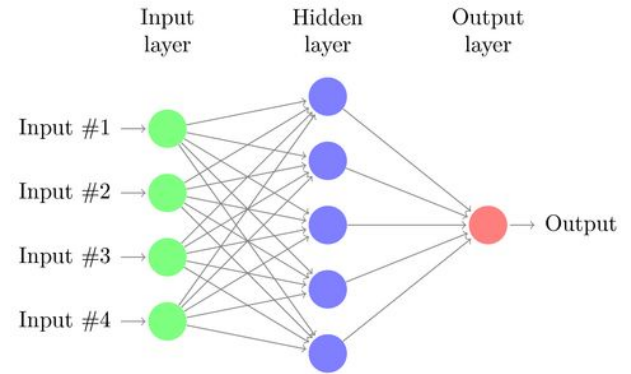


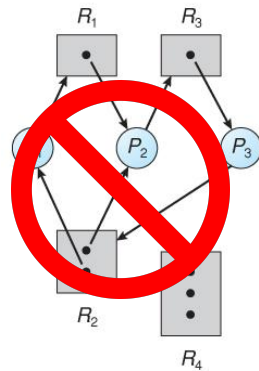


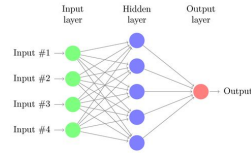
Formal Foundations of Serverless Computing

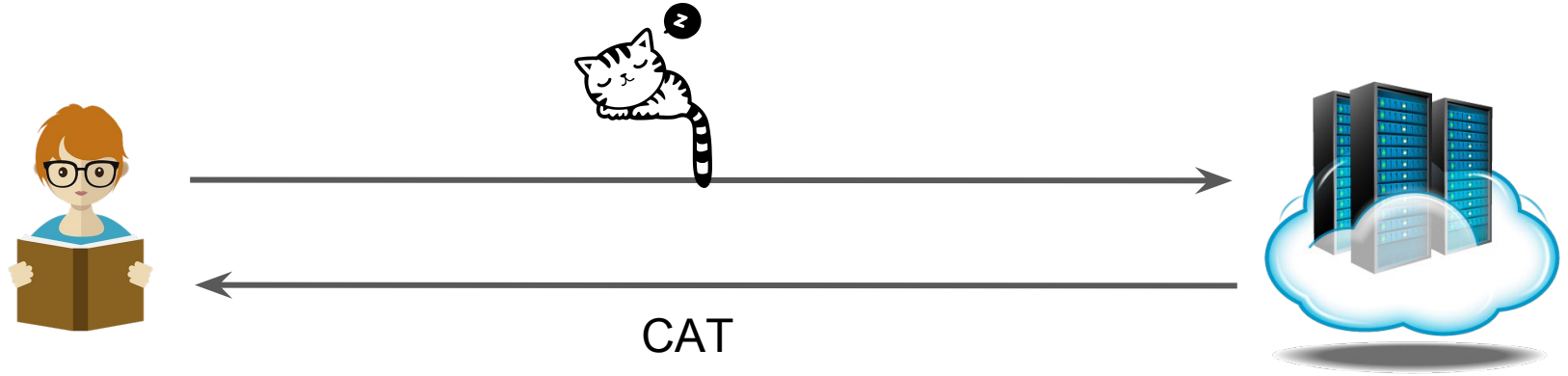
Abhinav Jangda, Donald Pickney, Samuel Baxter, Joseph Spitzer,
Breanna Devore-McDonald, Yuriy Brun, Arjun Guha











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

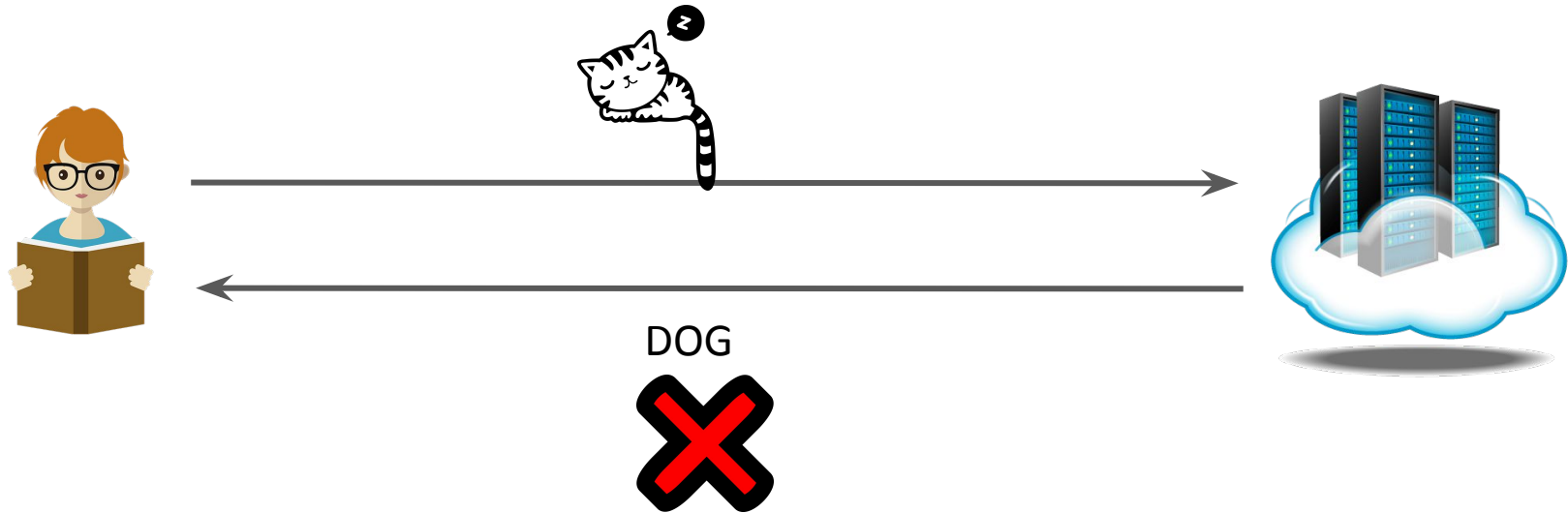
Train Model

Predict Cat or Dog



Play golf dataset

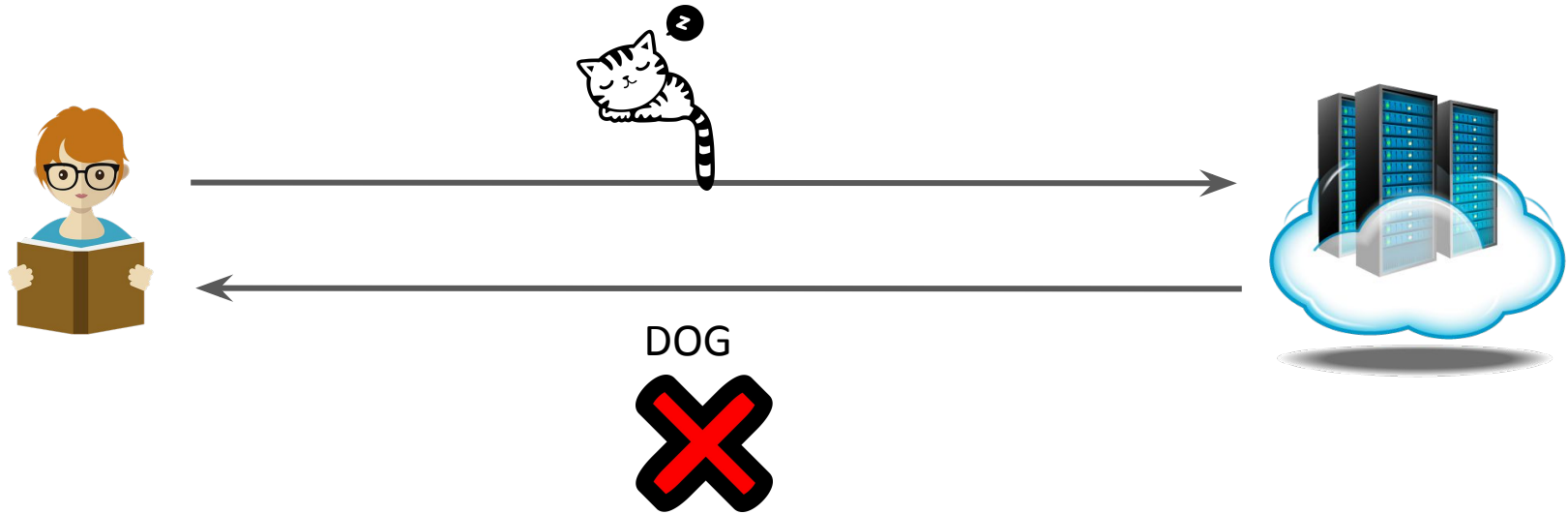
Independent variables				Dep. var
golfercode	teampartcode	timeofday	dayofweek	Play
1	1	1	1	Yes
1	1	2	1	No
1	1	3	1	No
1	1	4	1	No
1	1	5	1	No
1	1	6	1	No
1	1	7	1	No
1	1	8	1	No
1	1	9	1	No
1	1	10	1	No
1	1	11	1	No
1	1	12	1	No
1	1	13	1	No
1	1	14	1	No
1	1	15	1	No
1	1	16	1	No
1	1	17	1	No
1	1	18	1	No
1	1	19	1	No
1	1	20	1	No
1	1	21	1	No
1	1	22	1	No
1	1	23	1	No
1	1	24	1	No
1	1	25	1	No
1	1	26	1	No
1	1	27	1	No
1	1	28	1	No
1	1	29	1	No
1	1	30	1	No
1	1	31	1	No
1	1	32	1	No
1	1	33	1	No
1	1	34	1	No
1	1	35	1	No
1	1	36	1	No
1	1	37	1	No
1	1	38	1	No
1	1	39	1	No
1	1	40	1	No
1	1	41	1	No
1	1	42	1	No
1	1	43	1	No
1	1	44	1	No
1	1	45	1	No
1	1	46	1	No
1	1	47	1	No
1	1	48	1	No
1	1	49	1	No
1	1	50	1	No
1	1	51	1	No
1	1	52	1	No
1	1	53	1	No
1	1	54	1	No
1	1	55	1	No
1	1	56	1	No
1	1	57	1	No
1	1	58	1	No
1	1	59	1	No
1	1	60	1	No
1	1	61	1	No
1	1	62	1	No
1	1	63	1	No
1	1	64	1	No
1	1	65	1	No
1	1	66	1	No
1	1	67	1	No
1	1	68	1	No
1	1	69	1	No
1	1	70	1	No
1	1	71	1	No
1	1	72	1	No
1	1	73	1	No
1	1	74	1	No
1	1	75	1	No
1	1	76	1	No
1	1	77	1	No
1	1	78	1	No
1	1	79	1	No
1	1	80	1	No
1	1	81	1	No
1	1	82	1	No
1	1	83	1	No
1	1	84	1	No
1	1	85	1	No
1	1	86	1	No
1	1	87	1	No
1	1	88	1	No
1	1	89	1	No
1	1	90	1	No
1	1	91	1	No
1	1	92	1	No
1	1	93	1	No
1	1	94	1	No
1	1	95	1	No
1	1	96	1	No
1	1	97	1	No
1	1	98	1	No
1	1	99	1	No
1	1	100	1	No
1	1	101	1	No
1	1	102	1	No
1	1	103	1	No
1	1	104	1	No
1	1	105	1	No
1	1	106	1	No
1	1	107	1	No
1	1	108	1	No
1	1	109	1	No
1	1	110	1	No
1	1	111	1	No
1	1	112	1	No
1	1	113	1	No
1	1	114	1	No
1	1	115	1	No
1	1	116	1	No
1	1	117	1	No
1	1	118	1	No
1	1	119	1	No
1	1	120	1	No
1	1	121	1	No
1	1	122	1	No
1	1	123	1	No
1	1	124	1	No
1	1	125	1	No
1	1	126	1	No
1	1	127	1	No
1	1	128	1	No
1	1	129	1	No
1	1	130	1	No
1	1	131	1	No
1	1	132	1	No
1	1	133	1	No
1	1	134	1	No
1	1	135	1	No
1	1	136	1	No
1	1	137	1	No
1	1	138	1	No
1	1	139	1	No
1	1	140	1	No
1	1	141	1	No
1	1	142	1	No
1	1	143	1	No
1	1	144	1	No
1	1	145	1	No
1	1	146	1	No
1	1	147	1	No
1	1	148	1	No
1	1	149	1	No
1	1	150	1	No
1	1	151	1	No
1	1	152	1	No
1	1	153	1	No
1	1	154	1	No
1	1	155	1	No
1	1	156	1	No
1	1	157	1	No
1	1	158	1	No
1	1	159	1	No
1	1	160	1	No
1	1	161	1	No
1	1	162	1	No
1	1	163	1	No
1	1	164	1	No
1	1	165	1	No
1	1	166	1	No
1	1	167	1	No
1	1	168	1	No
1	1	169	1	No
1	1	170	1	No
1	1	171	1	No
1	1	172	1	No
1	1	173	1	No
1	1	174	1	No
1	1	175	1	No
1	1	176	1	No
1	1	177	1	No
1	1	178	1	No
1	1	179	1	No
1	1	180	1	No
1	1	181	1	No
1	1	182	1	No
1	1	183	1	No
1	1	184	1	No
1	1	185	1	No
1	1	186	1	No
1	1	187	1	No
1	1	188	1	No
1	1	189	1	No
1	1	190	1	No
1	1	191	1	No
1	1	192	1	No
1	1	193	1	No
1	1	194	1	No
1	1	195	1	No
1	1	196	1	No
1	1	197	1	No
1	1	198	1	No
1	1	199	1	No
1	1	200	1	No
1	1	201	1	No
1	1	202	1	No
1	1	203	1	No
1	1	204	1	No
1	1	205	1	No
1	1	206	1	No
1	1	207	1	No
1	1	208	1	No
1	1	209	1	No
1	1	210	1	No
1	1	211	1	No
1	1	212	1	No
1	1	213	1	No
1	1	214	1	No
1	1	215	1	No
1	1	216	1	No
1	1	217	1	No
1	1	218	1	No
1	1	219	1	No
1	1	220	1	No
1	1	221	1	No
1	1	222	1	No
1	1	223	1	No
1	1	224	1	No
1	1	225	1	No
1	1	226	1	No
1	1	227	1	No
1	1	228	1	No
1	1	229	1	No
1	1	230	1	No
1	1	231	1	No
1	1	232	1	No
1	1	233	1	No
1	1	234	1	No
1	1	235	1	No
1	1	236	1	No
1	1	237	1	No
1	1	238	1	No
1	1	239	1	No
1	1	240	1	No
1	1	241	1	No
1	1	242	1	No
1	1	243	1	No
1	1	244	1	No
1	1	245	1	No
1	1	246	1	No
1	1	247	1	No
1	1	248	1	No
1	1	249	1	No
1	1	250	1	No
1	1	251	1	No
1	1	252	1	No
1	1	253	1	No
1	1	254	1	No
1	1	255	1	No
1	1	256	1	No
1	1	257	1	No
1	1	258	1	No
1	1	259	1	No
1	1	260	1	No
1	1	261	1	No
1	1	262	1	No
1	1	263	1	No
1	1	264	1	No
1	1	265	1	No
1	1	266	1	No
1	1	267	1	No
1	1	268	1	No
1	1	269	1	No
1	1	270	1	No
1	1	271	1	No
1	1	272	1	No
1	1	273	1	No
1	1	274	1	No
1	1	275	1	No
1	1	276	1	No
1	1	277	1	No
1	1	278	1	No
1	1	279	1	No
1	1	280	1	No
1	1	281	1	No
1	1	282	1	No
1	1	283	1	No
1	1	284	1	No
1	1	285	1	No
1	1	286	1	No
1	1	287	1	No
1	1	288	1	No
1	1	289	1	No
1	1	290	1	No
1	1	291	1	No
1	1	292	1	No
1	1	293	1	No
1	1	294	1	No
1	1	295	1	No
1	1	296	1	No
1	1	297	1	No
1	1	298	1	No
1	1	299	1	No
1	1	300	1	No
1	1	301	1	No
1	1	302	1	No
1	1	303	1	No
1	1	304	1	No
1	1	305	1	No
1	1	306	1	No
1	1	307	1	No
1	1	308	1	No
1	1	309	1	No
1	1	310	1	No
1	1	311	1	No
1	1	312	1	No
1	1	313	1	No
1	1	314	1	No
1	1	315	1	No
1	1	316	1	No
1	1	317	1	No
1	1	318	1	No
1	1	319	1	No
1	1	320	1	No
1	1	321	1	No
1	1	322	1	No
1	1	323	1	No
1	1	324	1	No
1	1	325	1	No
1	1	326	1	No
1	1	327	1	No
1	1	328	1	No
1	1	329	1	No
1	1	330	1	No
1	1	331	1	No
1	1	332	1	No
1	1	333	1	No
1	1	334	1	No
1	1	335	1	No
1	1	336	1	No
1	1	337	1	No
1	1	338	1	No
1	1	339	1	No
1	1	340	1	No
1	1	341	1	No
1	1	342	1	No
1	1	343	1	No
1	1	344	1	No
1	1	345	1	No
1	1	346	1	No
1	1	347	1	No
1	1	348	1	No
1	1	349	1	No
1	1	350	1	No
1	1	351	1	No

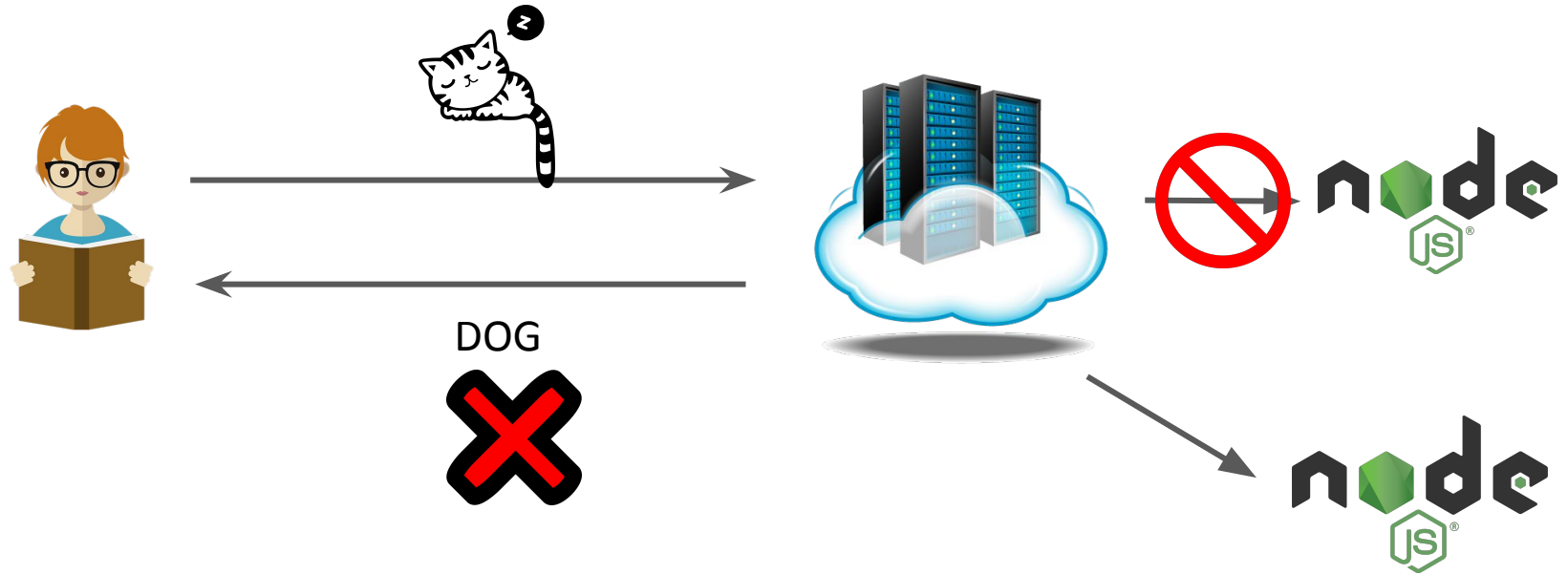


```
model = CatOrDogModel ();  
function catOrDog (req, res) {  
  if (req.type === 'train') {  
    model.train (req.data);  
  } else {  
    res.write (model.predict (req.data));  
  }  
}
```

Ephemeral State

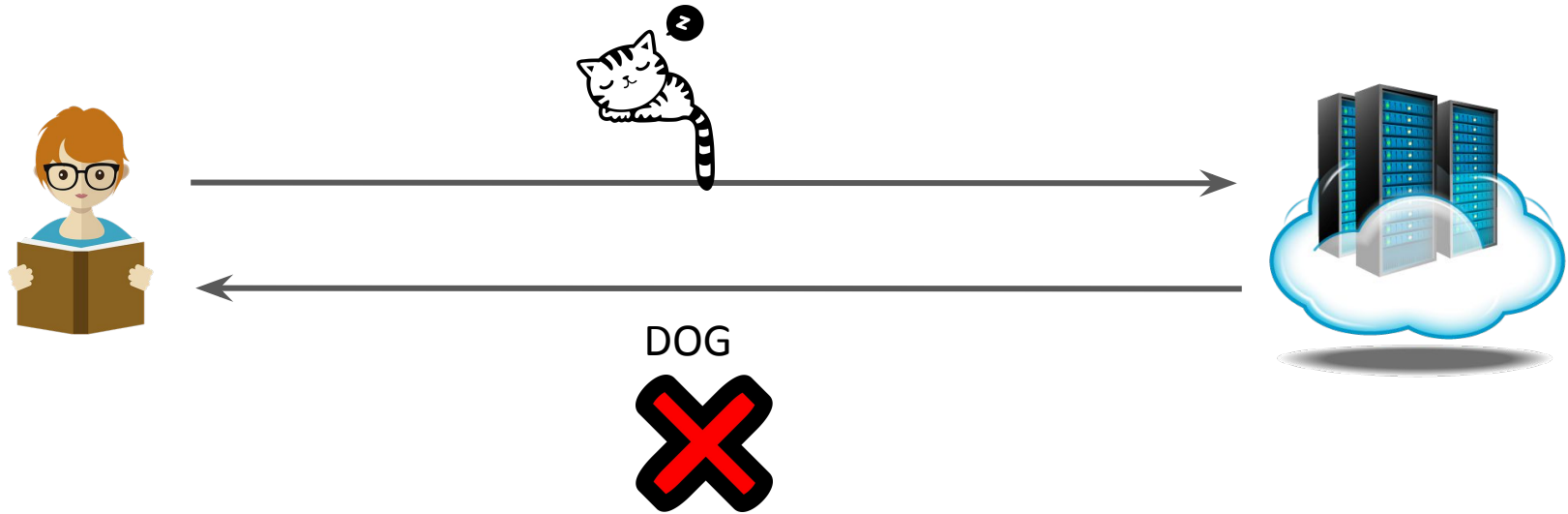
The diagram illustrates the concept of 'Ephemeral State' by highlighting three specific code snippets in the provided JavaScript code. Each snippet is enclosed in a red dashed rectangular box. Three black arrows originate from the right side of these boxes and point towards the text 'Ephemeral State' on the right. The first arrow points from the line 'model = CatOrDogModel ();'. The second arrow points from the line 'model.train (req.data);'. The third arrow points from the line 'res.write (model.predict (req.data));'.

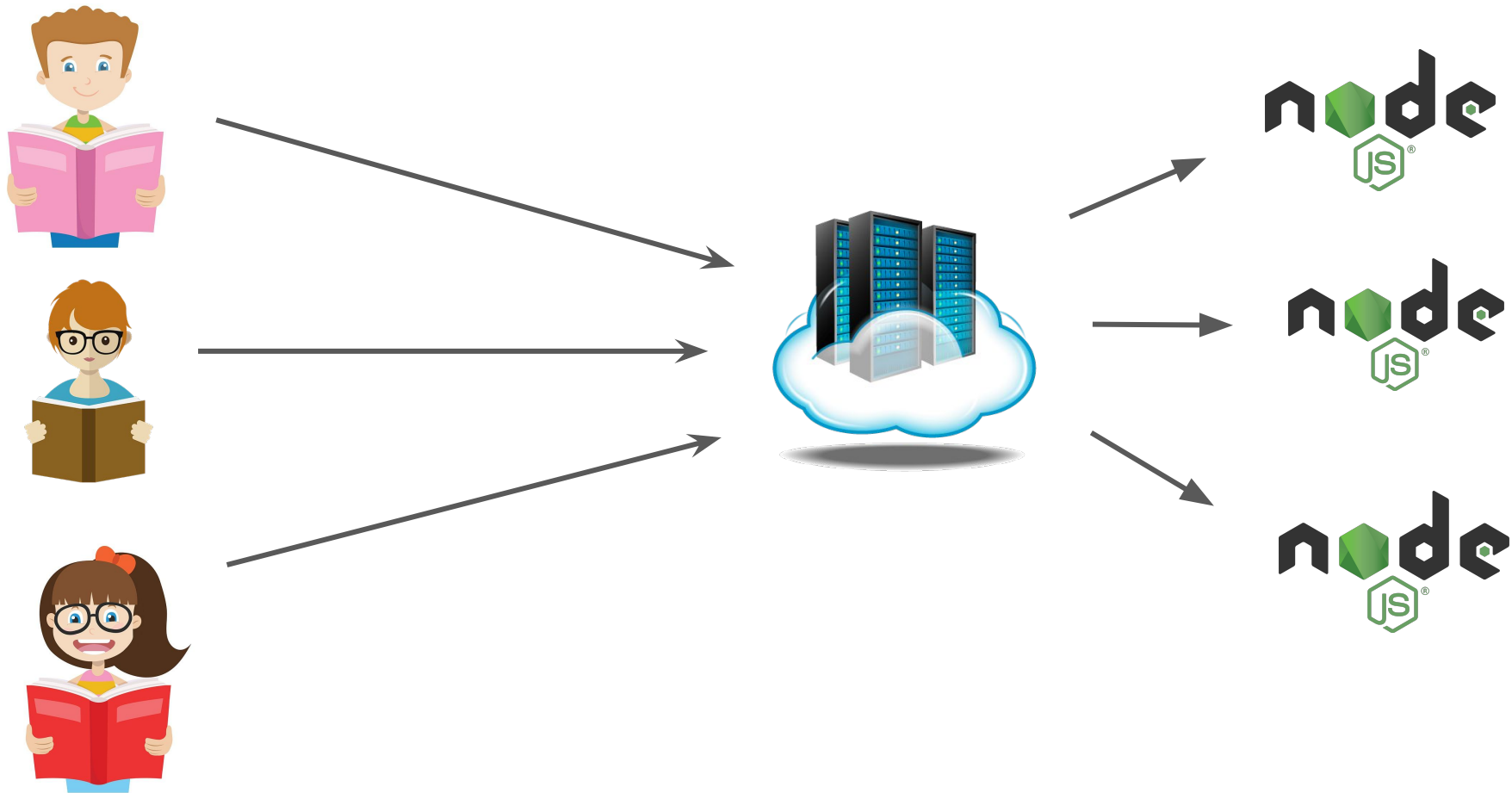




```
model = CatOrDogModel ();  
function catOrDog (req, res) {  
  if (req == 'train') {  
    model.train (req.data);  
  } else {  
    res.write (model.predict (req.data));  
  }  
}
```

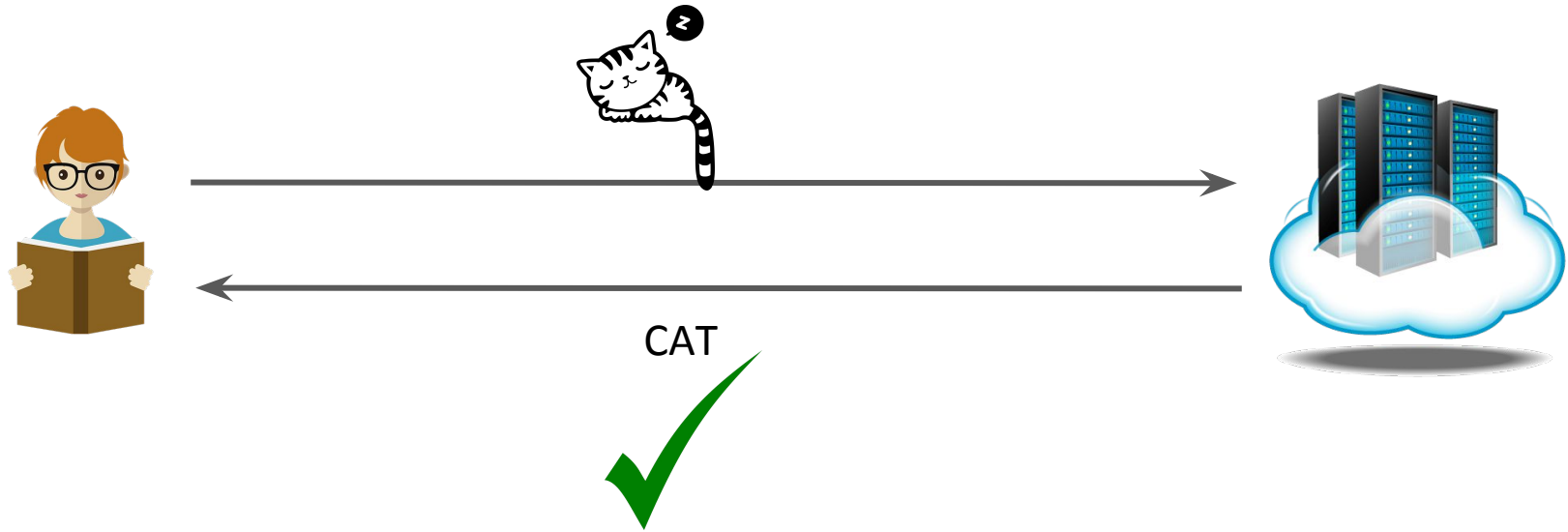
Single event processed to
completion multiple times





```
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.





```
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

$$\text{REQ} \frac{x \text{ is fresh}}{\mathcal{C} \xRightarrow{\text{start}(x,v)} \mathcal{C}\mathbb{R}(f, x, v)}$$

$$\text{COLD} \frac{y \text{ is fresh} \quad \text{recv}(v, \sigma_0) = \sigma}{\mathcal{C}\mathbb{R}(f, x, v) \Rightarrow \mathcal{C}\mathbb{R}(f, x, v)\mathbb{F}(f, \text{busy}(x), \sigma, y)}$$

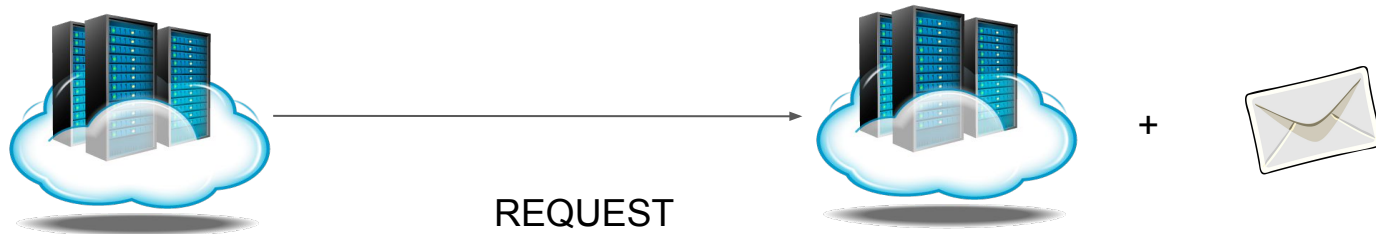
$$\text{WARM} \frac{\text{recv}(v, \sigma) = \sigma'}{\mathcal{C}\mathbb{R}(f, x, v)\mathbb{F}(f, \text{idle}, \sigma, y) \Rightarrow \mathcal{C}\mathbb{R}(f, x, v)\mathbb{F}(f, \text{busy}(x), \sigma', y)}$$

$$\text{HIDDEN} \frac{\text{step}(\sigma) = (\sigma', \varepsilon)}{\mathcal{C}\mathbb{F}(f, \text{busy}(x), \sigma, y) \Rightarrow \mathcal{C}\mathbb{F}(f, \text{busy}(x), \sigma', y)}$$

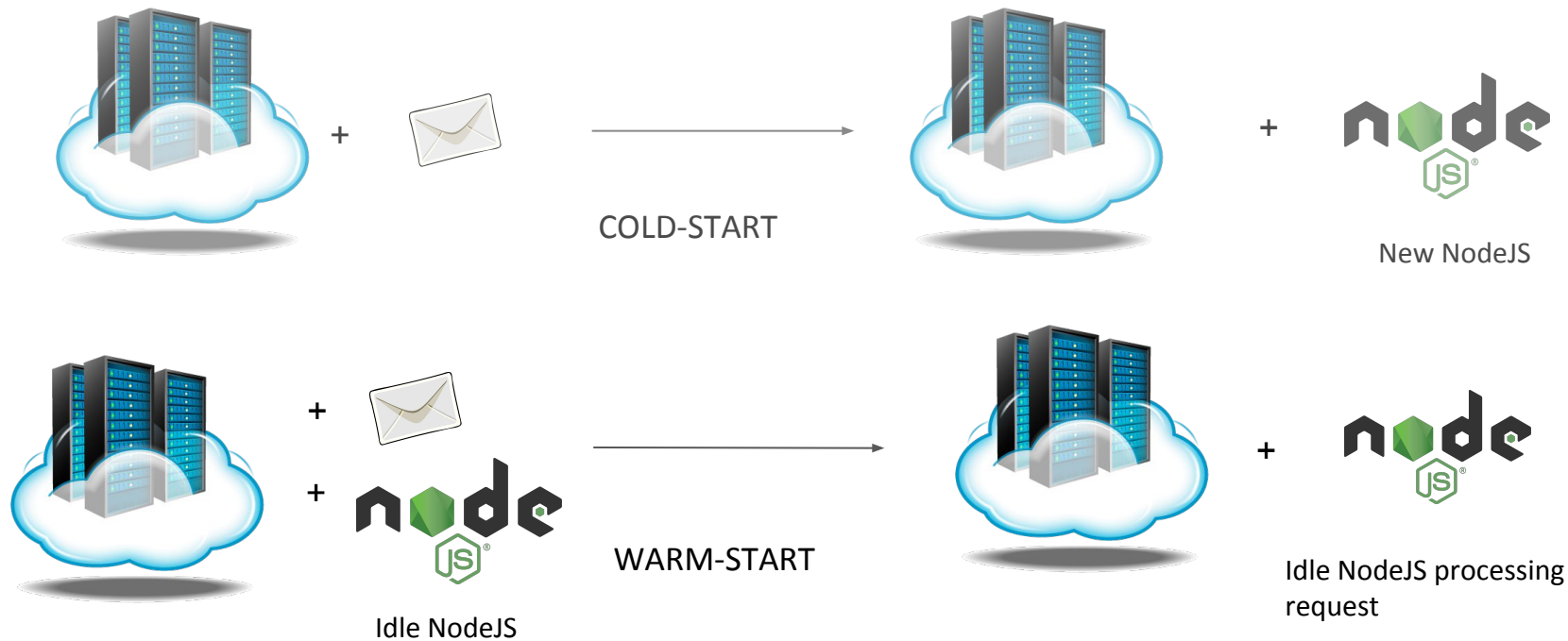
$$\text{RESP} \frac{\text{step}(\sigma) = (\sigma', \text{return}(v'))}{\mathcal{C}\mathbb{R}(f, x, v)\mathbb{F}(f, \text{busy}(x), \sigma, y) \xRightarrow{\text{stop}(x, v')} \mathcal{C}\mathbb{F}(f, \text{idle}, \sigma, y)\mathbb{S}(x, v')}$$

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

Operational Semantics of Serverless System



Operational Semantics of Serverless System



Summary

Operational semantics models all details of serverless platforms:

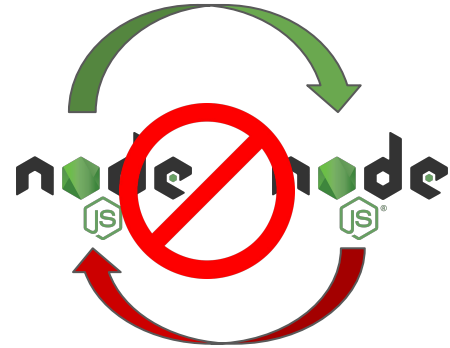
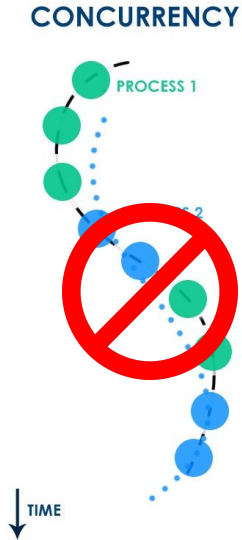
1. New instance creation
2. Instance reuse
3. Ephemeral and persistent state

Ideally: Provide programmers higher-level abstractions

1st Case Study: Idealized Serverless Semantics

Models an idealized serverless platform with:

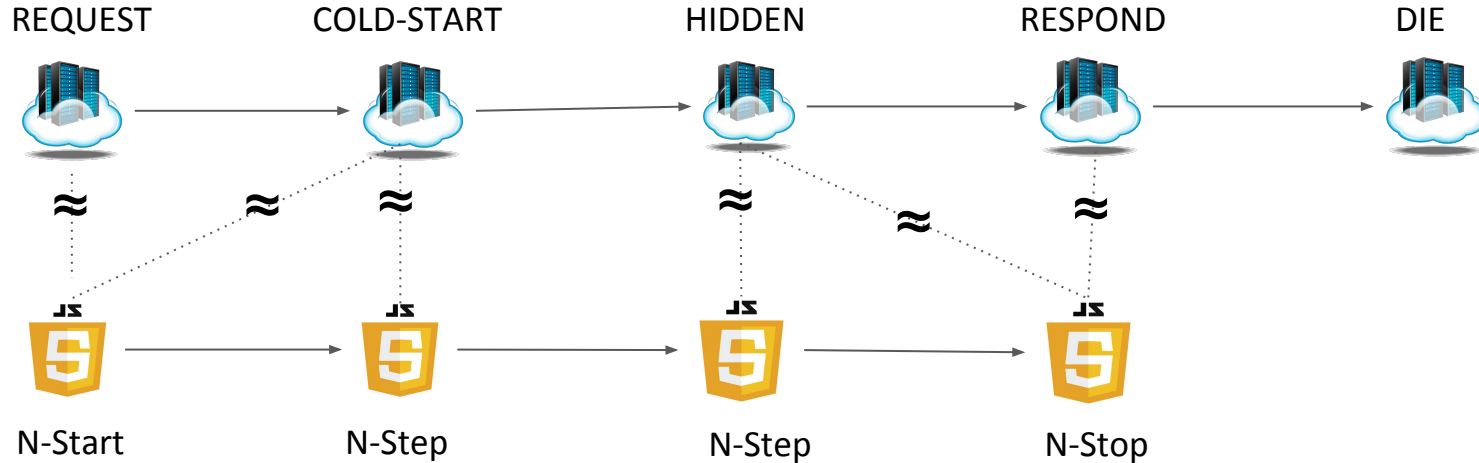
~~Failures~~



1st Case Study: Idealized Serverless Semantics

$$\begin{array}{l} \text{N-START} \frac{x \text{ is fresh} \quad \text{recv}(v, \sigma_0) = \sigma'}{\langle f, \text{idle}, \vec{\sigma} \rangle \xrightarrow{\text{start}(x, v)} \langle f, \text{busy}(x), [\sigma_0, \sigma'] \rangle} \\ \\ \text{N-STEP} \frac{\text{step}(\sigma) = (\sigma', \varepsilon)}{\langle f, \text{busy}(x), \vec{\sigma} \uparrow [\sigma] \rangle \mapsto \langle f, \text{busy}(x), \vec{\sigma} \uparrow [\sigma, \sigma'] \rangle} \\ \\ \text{N-STOP} \frac{\text{step}(\sigma) = (\sigma', \text{return}(v))}{\langle f, \text{busy}(x), \vec{\sigma} \uparrow [\sigma] \rangle \xrightarrow{\text{stop}(x, v)} \langle f, \text{idle}, [\sigma] \rangle} \end{array}$$

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



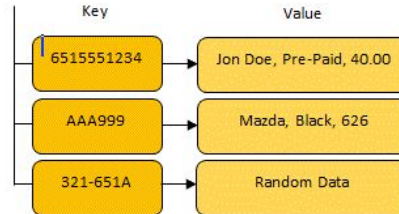
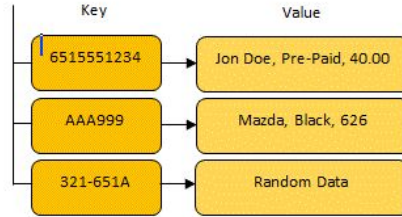
+

⋮

≈

⋮

+

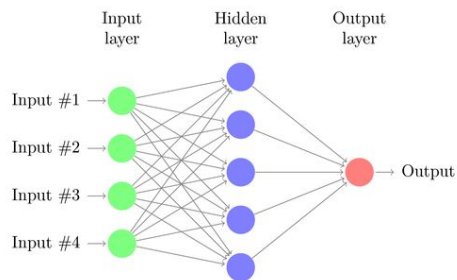


3rd Case Study: Serverless Programming Language

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



Serverless Compositions

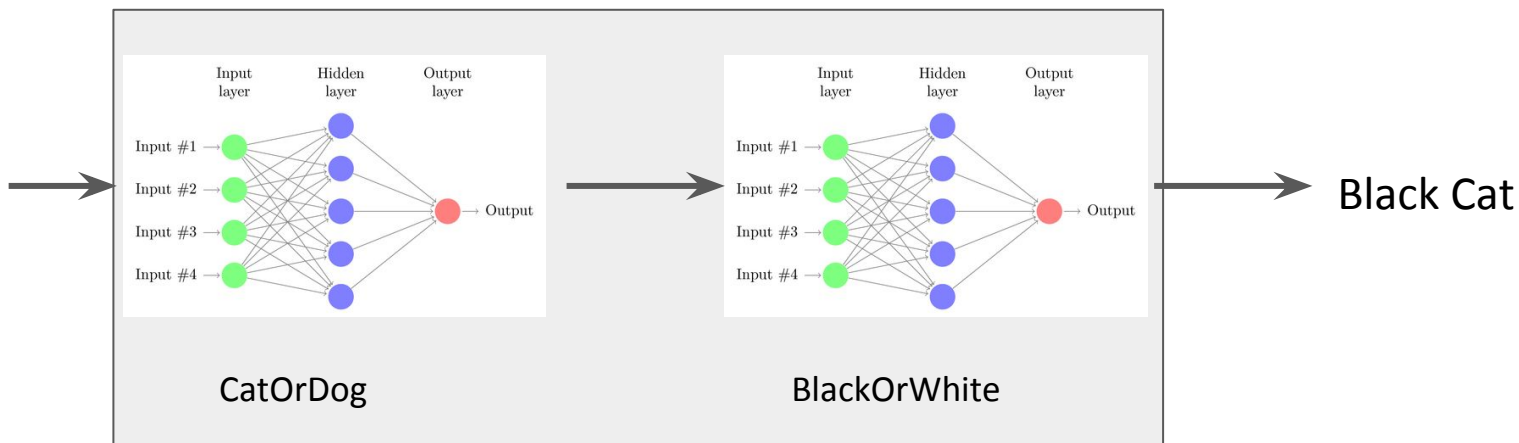


BlackOrWhite



Black

Serverless Compositions



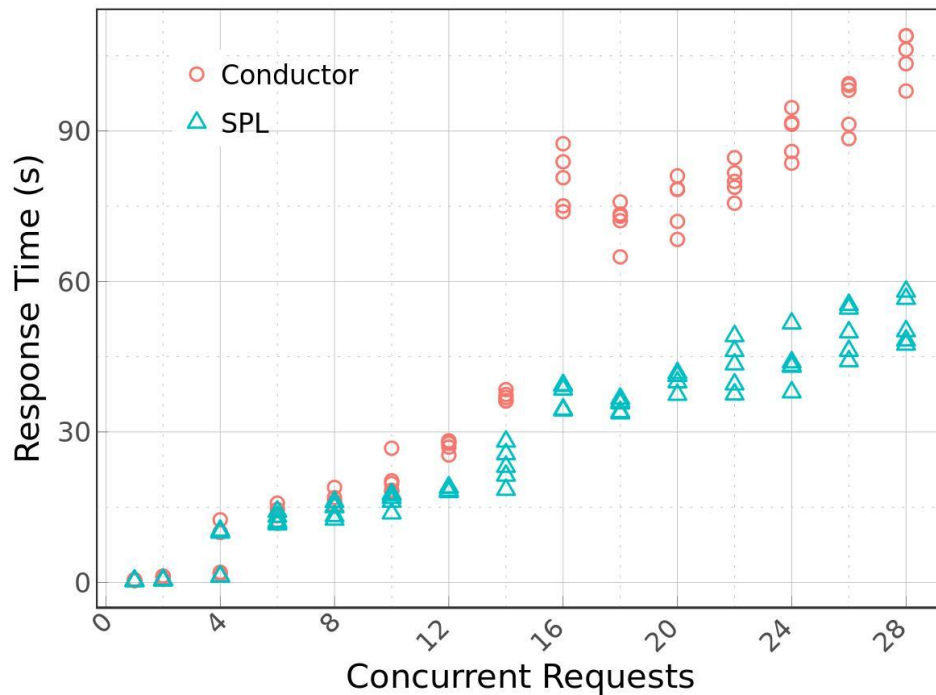
3rd Case Study: Serverless Programming Language

```
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.

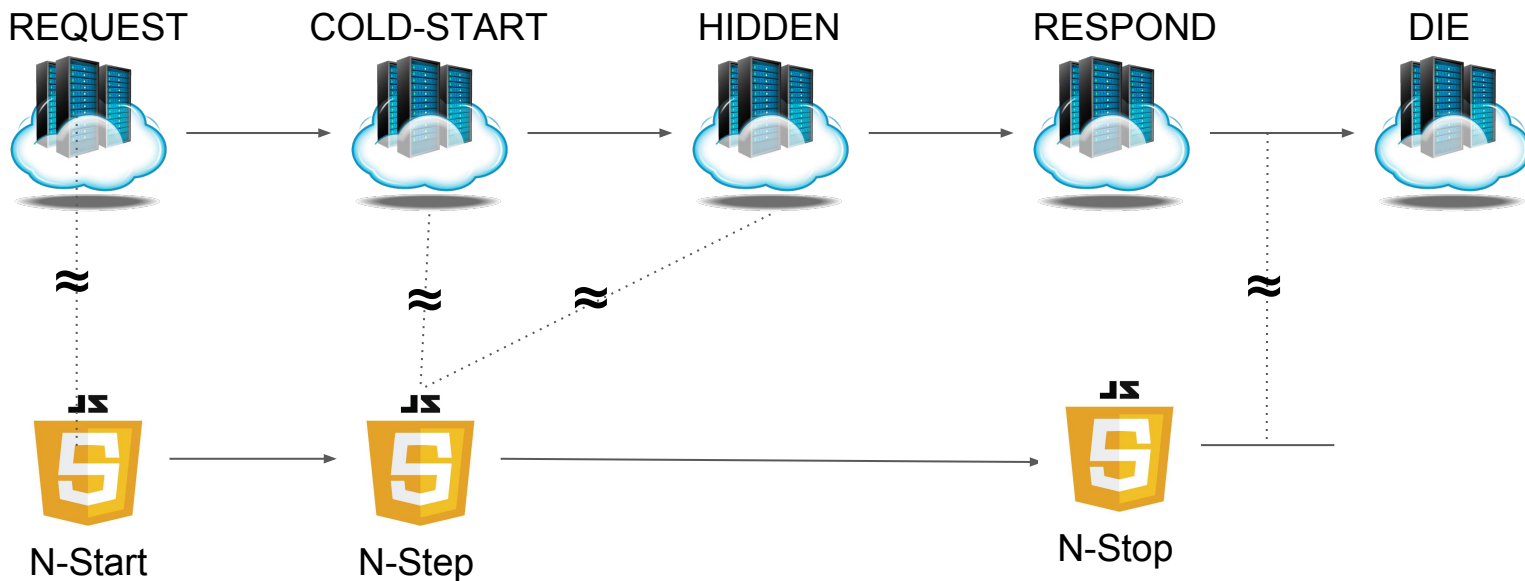


Contributions

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.

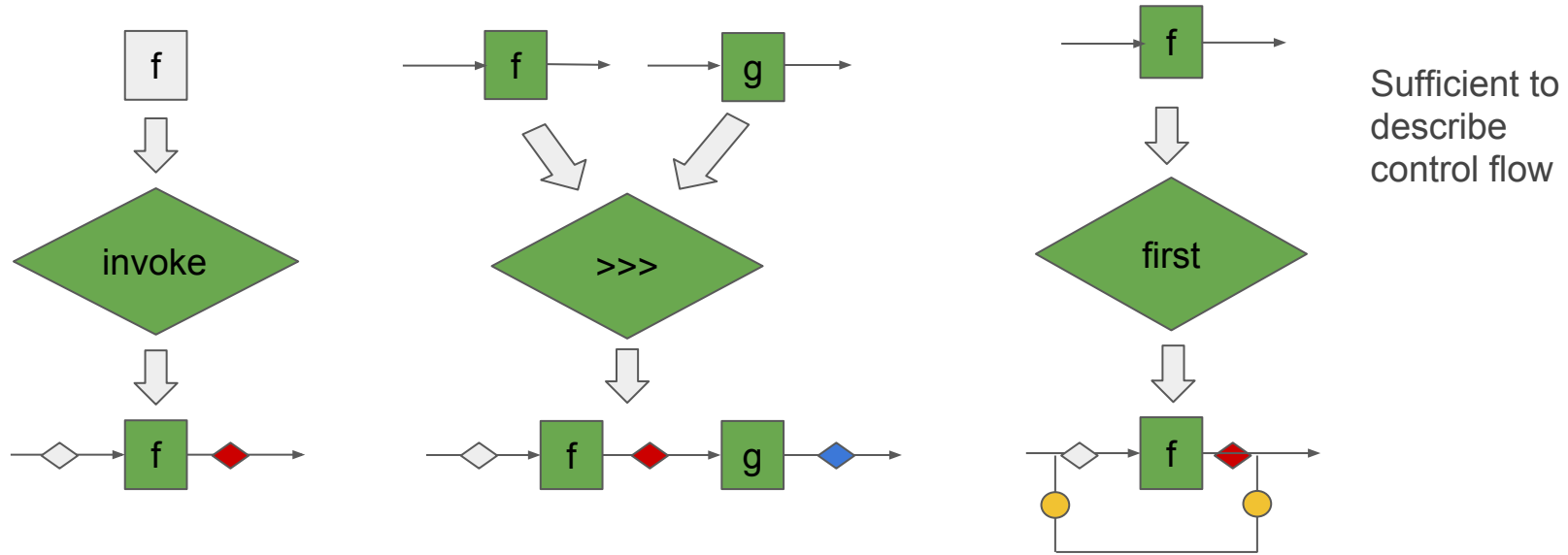


1st Case Study: Naive Serverless Semantics



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

3rd Case Study: Serverless Compositions



Issues

```
let accounts = new Map();
```

```
exports.bank = function(req, res) {
```

```
  accounts.set(req.body.name, req.body.value);
```

```
  res.send(true);
```

```
};
```

- Ephemeral State
- More than one instances of function can be running in parallel.
- Re-invoke functions when a failure is detected.
A single event may be processed to completion multiple times.

Operational Semantics of Serverless System

- Based on experiences with major serverless computing platforms.

Operational Semantics of Serverless System



+

$R(f, id, v)$



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

COLD-START

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



+

$R(f, id, v)$

+

$F(f, idle, \sigma)$



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

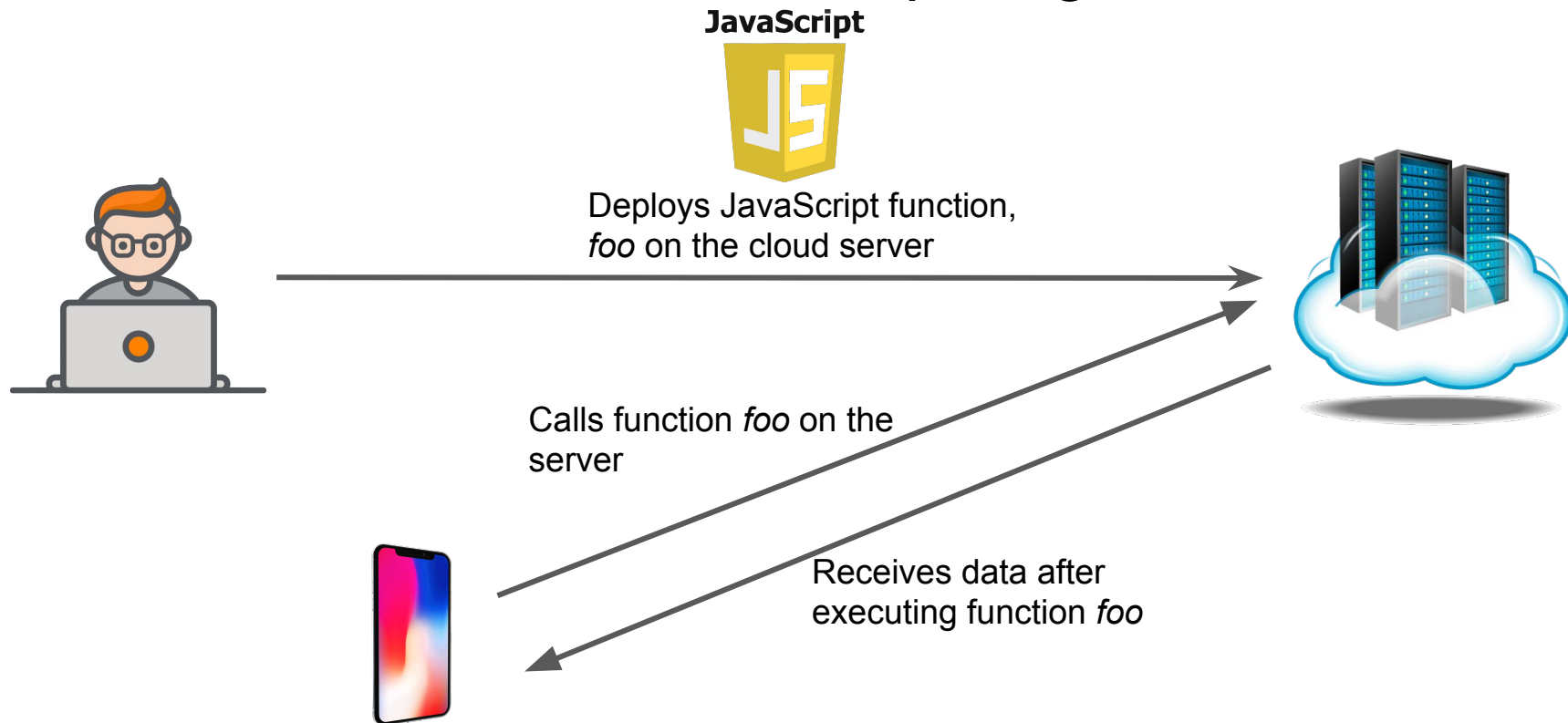
WARM-START

Start executing f on an *idle* function instance

Operational Semantics of Serverless System



Introduction to Serverless Computing





- Focus on application code only



- Compiles code
- Configures OS
- Manages resource allocation.

Serverless Function Example

```
let accounts = new Map();
```

Store account data

```
exports.bank = function(req, res) {
```

Takes a HTTP Req

```
  accounts.set(req.body.name, req.body.value);
```

Deposit the amount

```
  res.send(true);
```

```
};
```

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]
```

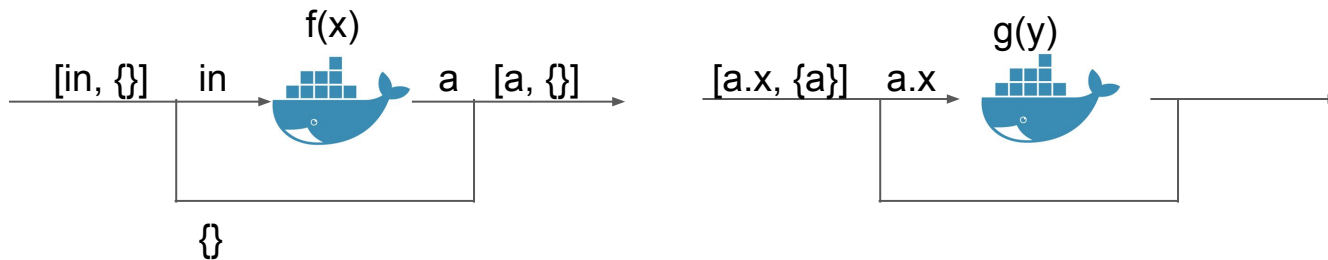
in is the input to the
composition

First element of array as the input
to the composition

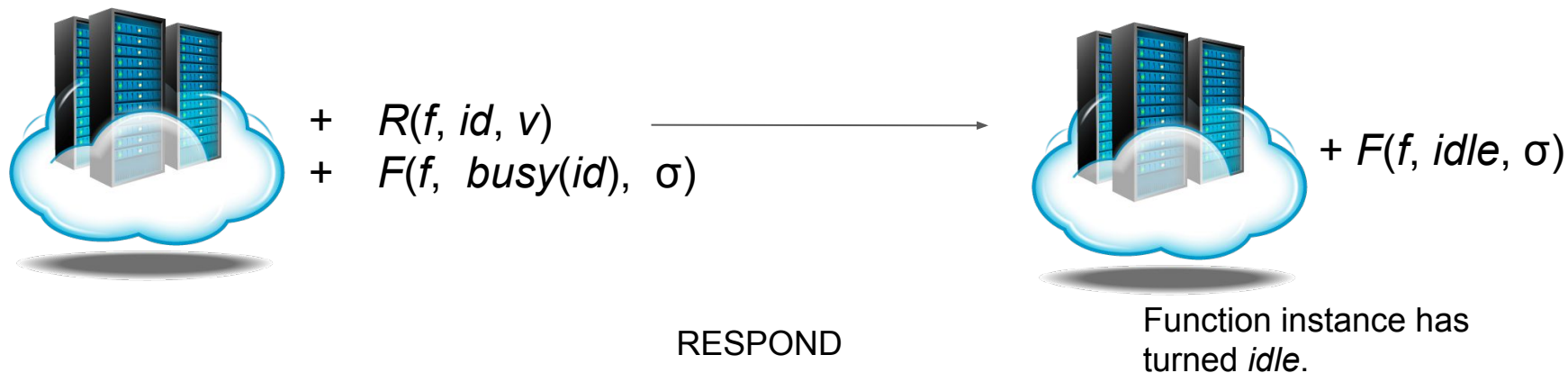
Perform JSON Transformations
to store the state of
composition

Serverless Programming Language

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



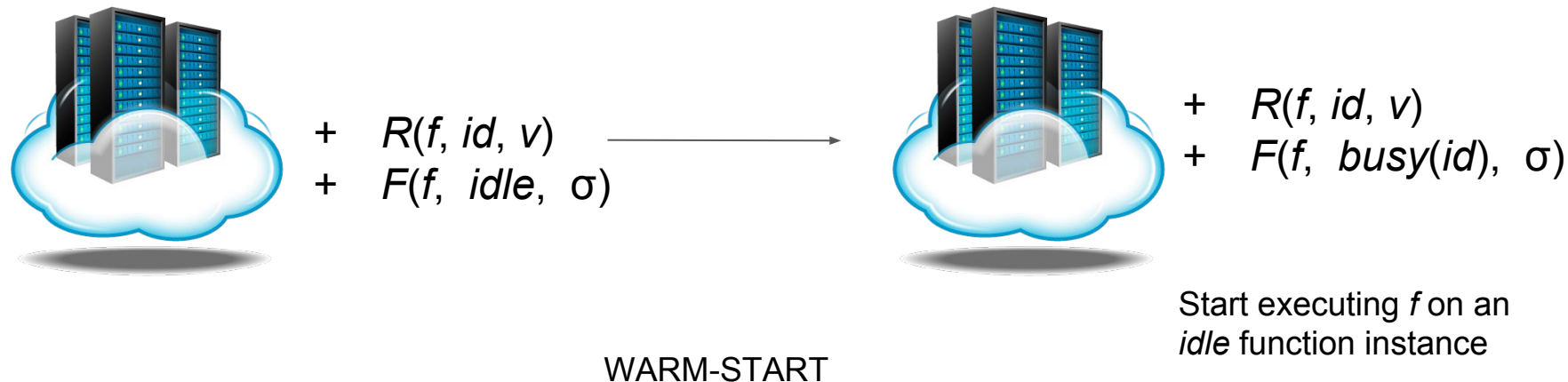
Operational Semantics of Serverless System



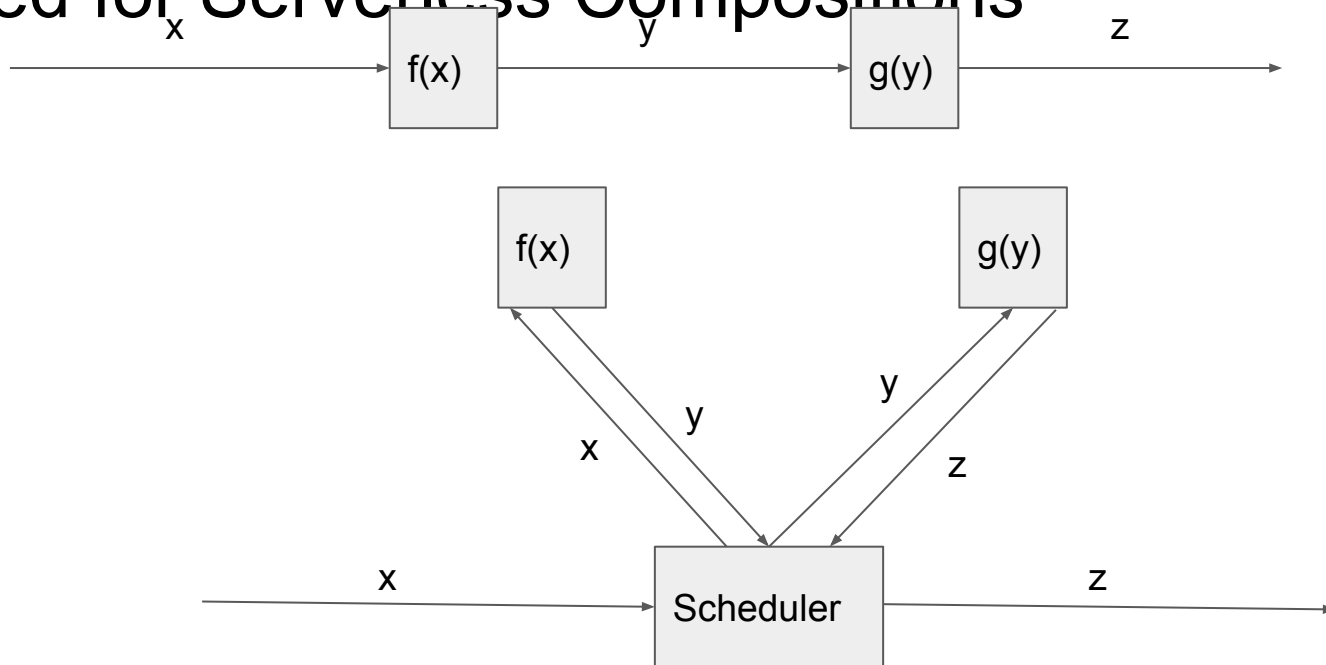
Operational Semantics of Serverless System



Operational Semantics of Serverless System



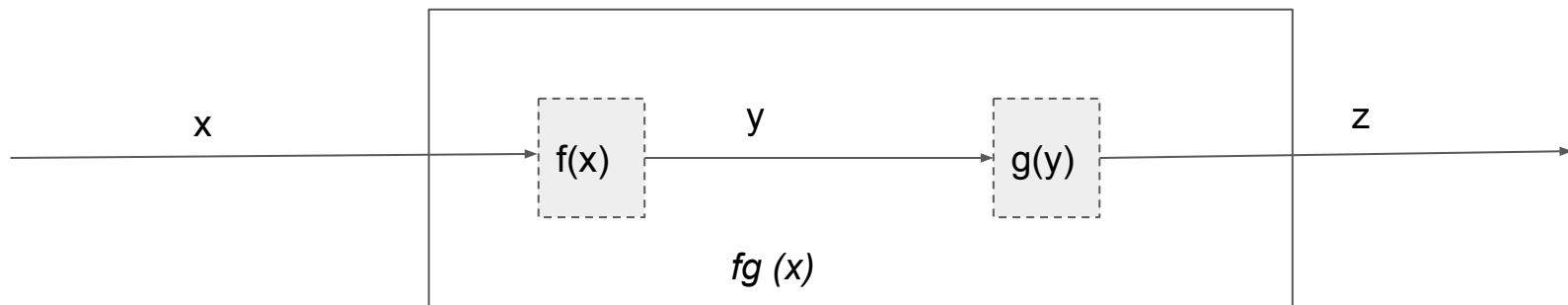
Need for Serverless Compositions



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

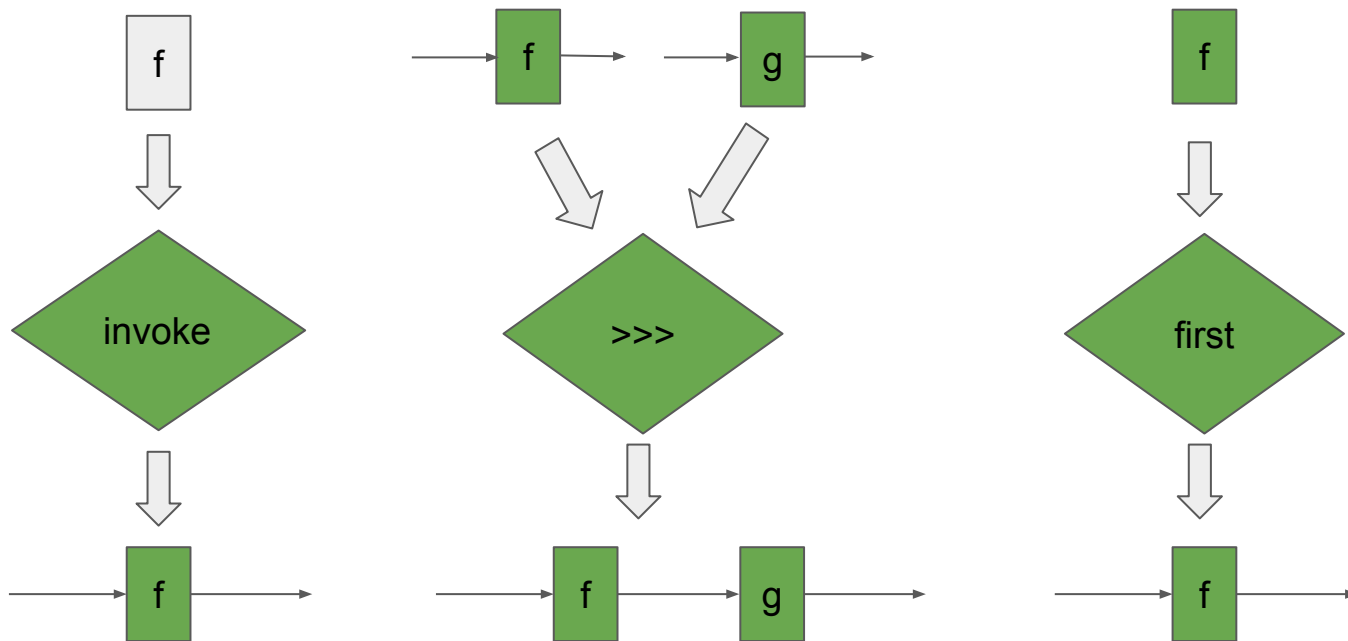
Need for Serverless Compositions

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.

