Practical Tooling for Serverless Computing

Josef Spillner <josef.spillner@zhaw.ch>
Service Prototyping Lab (blog.zhaw.ch/icclab)
Zurich University of Applied Sciences

Dec 5, 2017 | UCC‘17 | Austin, Texas, USA

Your Tutorial Agenda

Session 1

08.00-09.30 (90‘)	Tutorial Basics	
	I	Serverless Foundations
	II	Function Developer Tools

Session 2

10.00-12.00 (120‘)	III	Function Execution Tools
	IV	Research Challenges
	Discussion	

Your Tutorial Instructor

Josef Spillner <josef.spillner@zhaw.ch>

- works at Zurich University of Applied Sciences
- lectures Internet Service Prototyping, Python programming to undergraduates & masters of advanced studies
- performs research in the Service Prototyping Lab

- sometimes, publishes on FaaS / Serverless topics
  - CARLA‘17 serverless HPC paper
  - four 2017 preprints on arXiv related to tools
  - ongoing work on tracing/debugging as well as serverless developer survey
- co-authored «Architectural Transformations in Network Services and Distributed Systems»

[LS16]

Service Engineering Research Area
ICCLAB SPLAB

Your Tutorial Equipment

Required Installation

Local accounts → help yourself (git, docker, ...)
Cloud accounts → ask for tutoX account & ssh into 160.85.4.155

Supplementary Material:

Script: https://drive.switch.ch/index.php/s/Upjd0aXCypZjMnZ

Your Tutorial Equipment

Dependencies

- git, python3, ... (probably pre-installed on Ubuntu 16.04)

- python3-flask
- python3-boto3

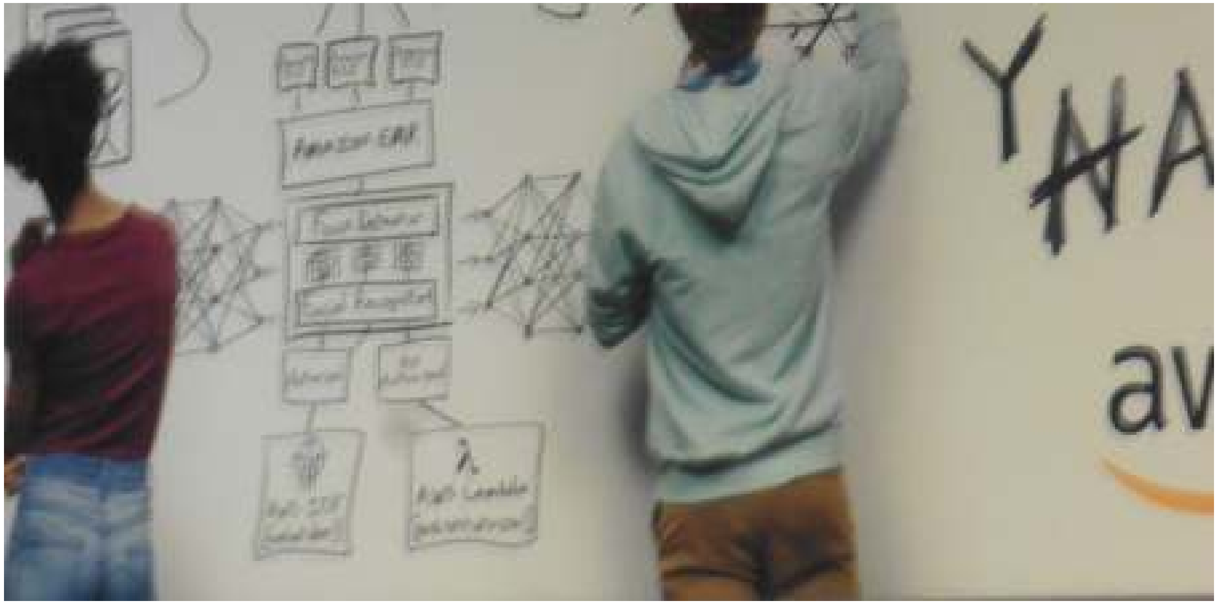
- openjdk-8-jdk
- maven
- openjfx

- gcc

- awscli
- unzip

Part I - Serverless Foundations

Industry Perspective: JFK 2 Days Ago



Industry Perspective: Cloud Apps

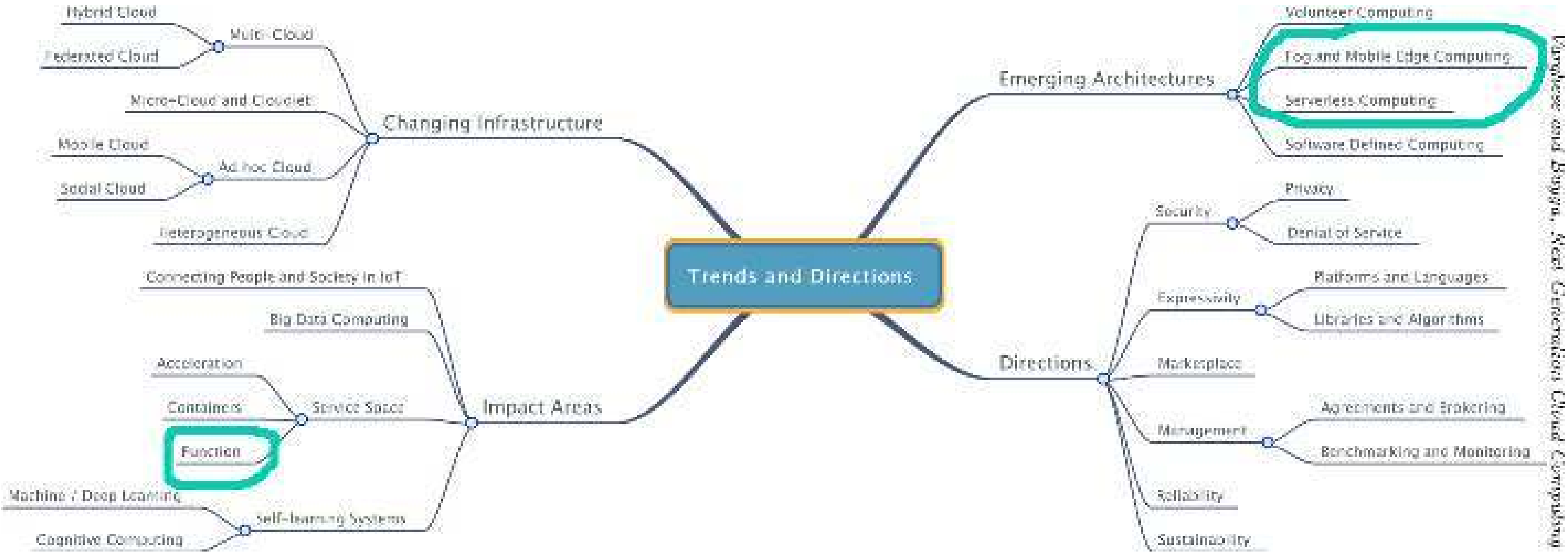
[https://github.com/cncf/landscape, Oct'17]

Service Engineering Research Area
ICCLAB SPLAB

Industry Perspective: FaaS

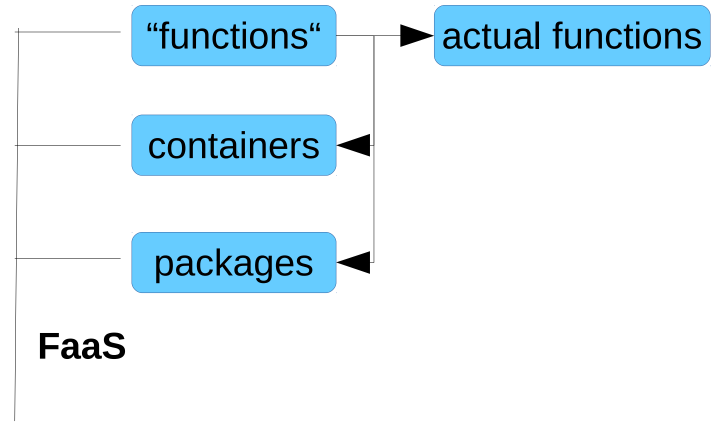
[Astasia Myers, Memory Leak, Oct'17]

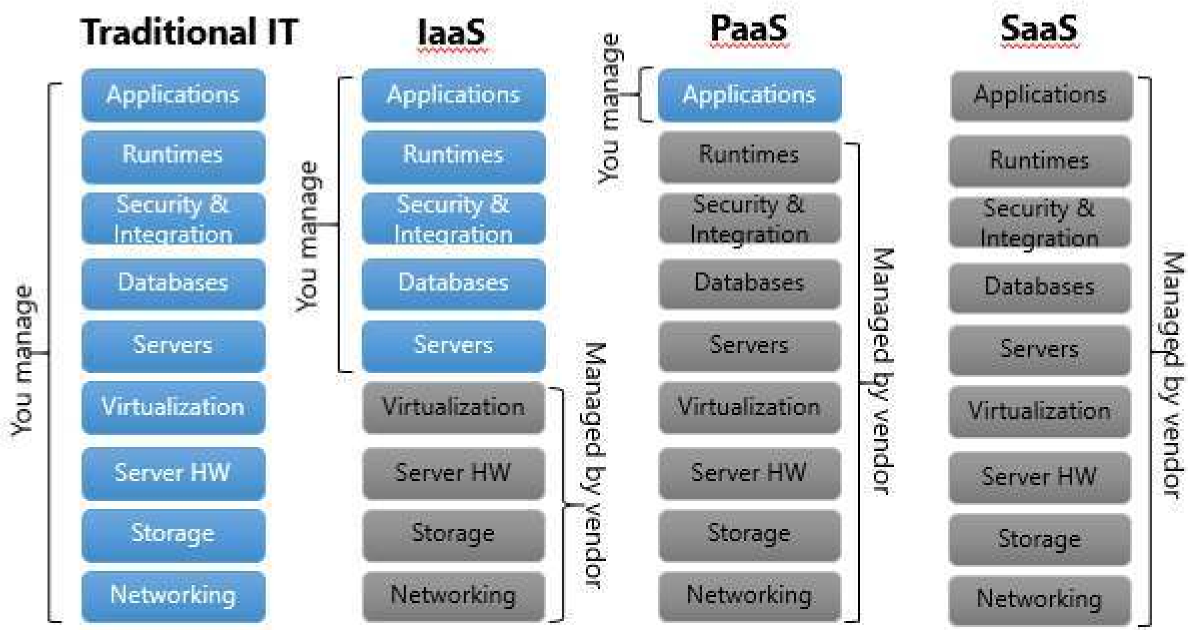
Academic Perspective: Clouds & FaaS



What is FaaS?

- running functions in the cloud (hosted functions)
- real “pay per use“ (per invocation, per load x time unit, e.g. GHz/100ms)
- seemingly “serverless“

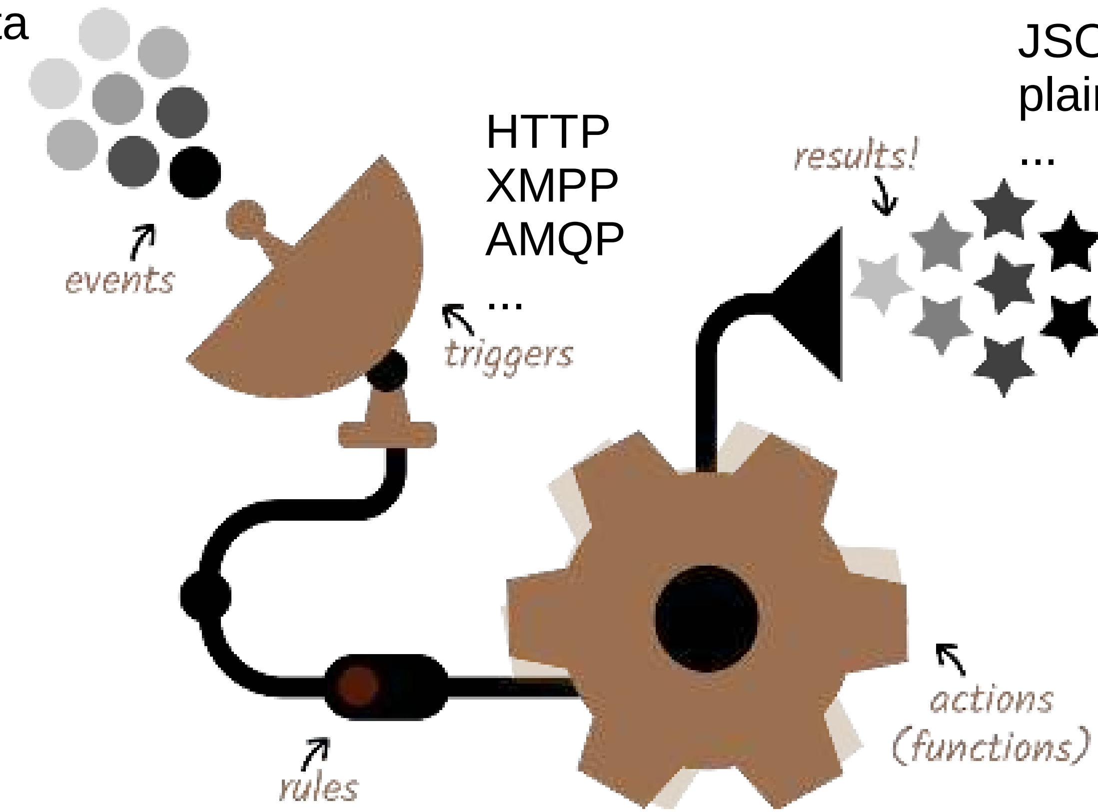




[mazikglobal.com]

FaaS Process

monitoring event
sensor data
log entry
git push
...



[openwhisk.org]

HTTP
XMPP
AMQP
...

JSON
plain text
...

max 1 per hour
triggers/actions
default params
...

your
Python/Java/...
functions!

FaaS in Formal Terms

- programming model
  - functions or methods in diverse programming languages
  - with specific signatures (parameters, return values)
  - sometimes, executable implementations, e.g. containers
- deployment model
  - upload of source files or compiled binaries
  - configuration of entrance handler, memory allocation, etc.
- execution model
  - time limit, e.g. 5 minutes
  - pay-per-use microbilling, e.g. per invocation + 100ms duration
- roughly equal to serverless computing: marketing term
  - for Function-as-a-Service ecosystems

Service Engineering Research Area
ICCLAB SPLAB

FaaS in Technical Terms

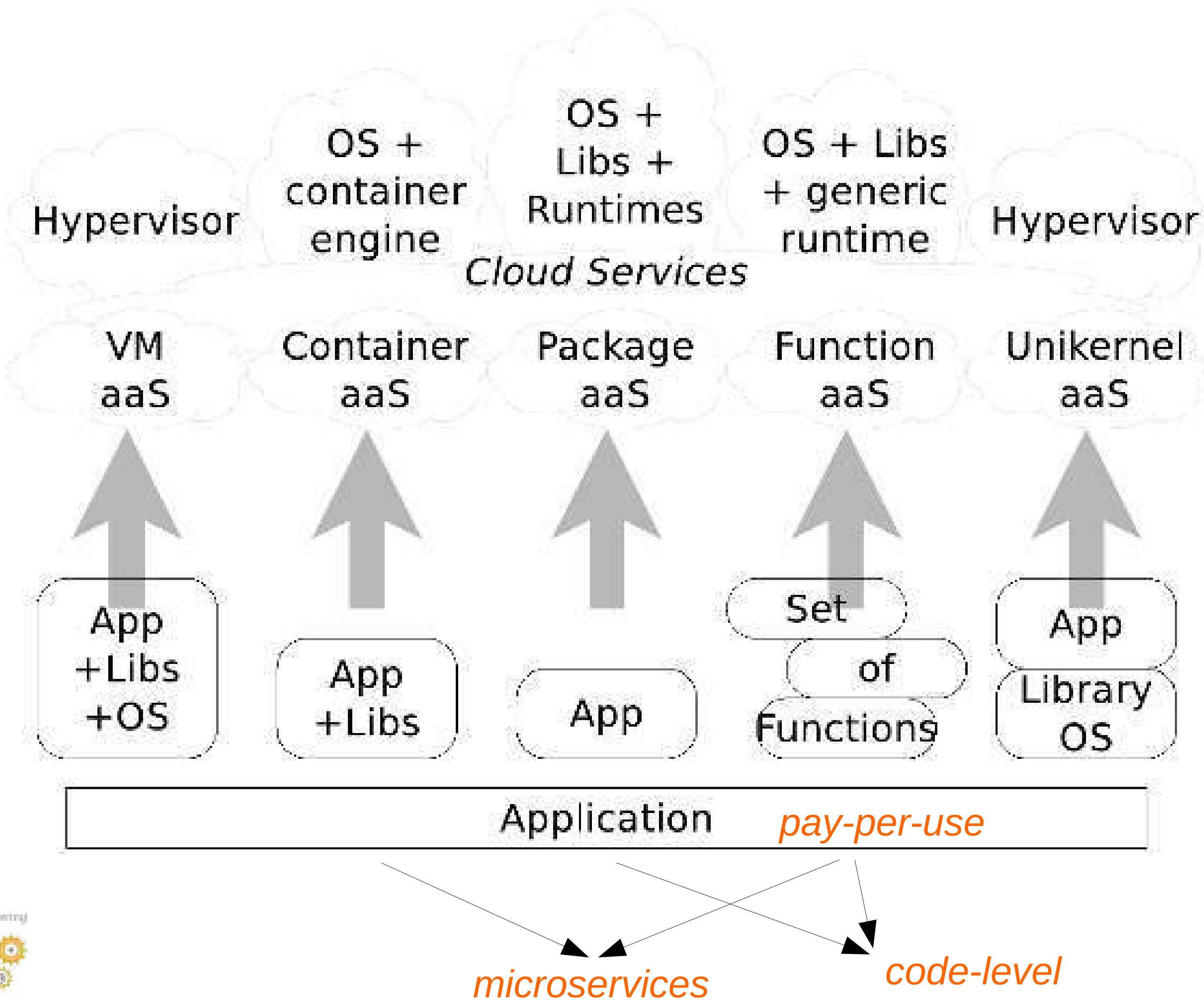
- Function = elementary unit
- App/Bundle = complex unit

provider-specific units

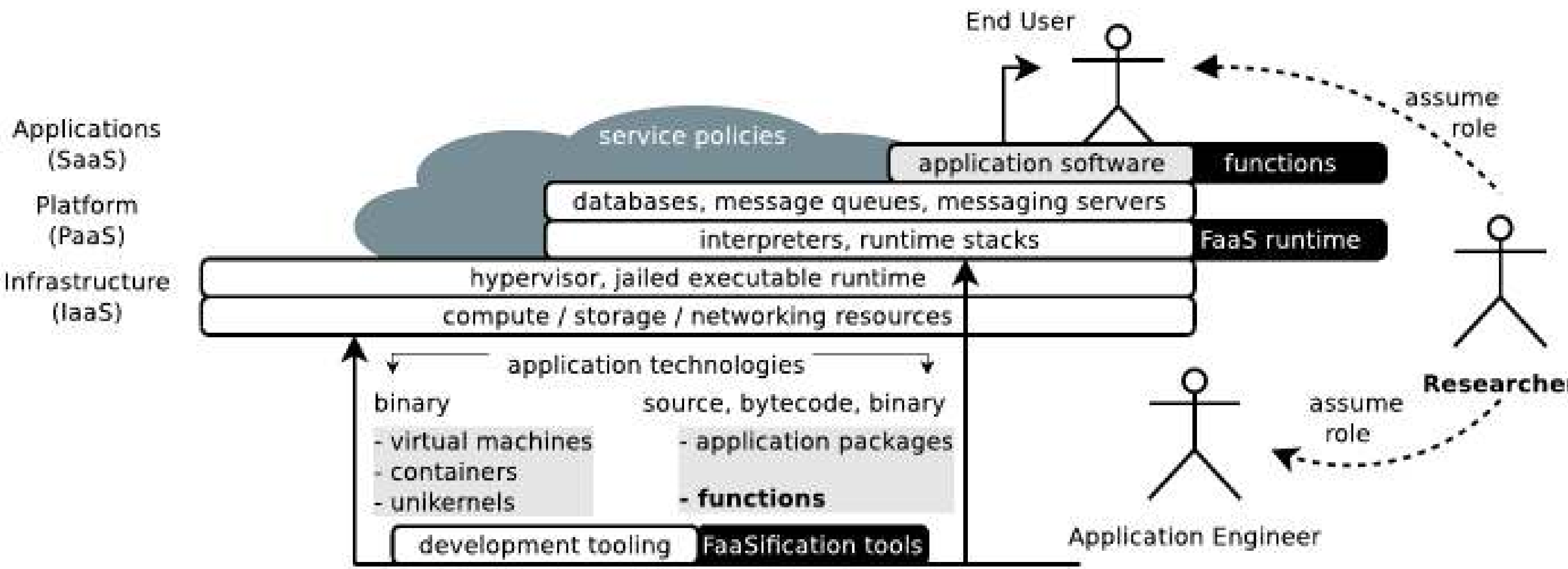
- Input
  - application-specific
  - context parameters, direct protocol access (if supported)
- Processing
  - up to the application
- Output
  - application-specific
  - state signalling, direct protocol access (if supported)

Service Engineering Research Area
ICCLAB SPLAB

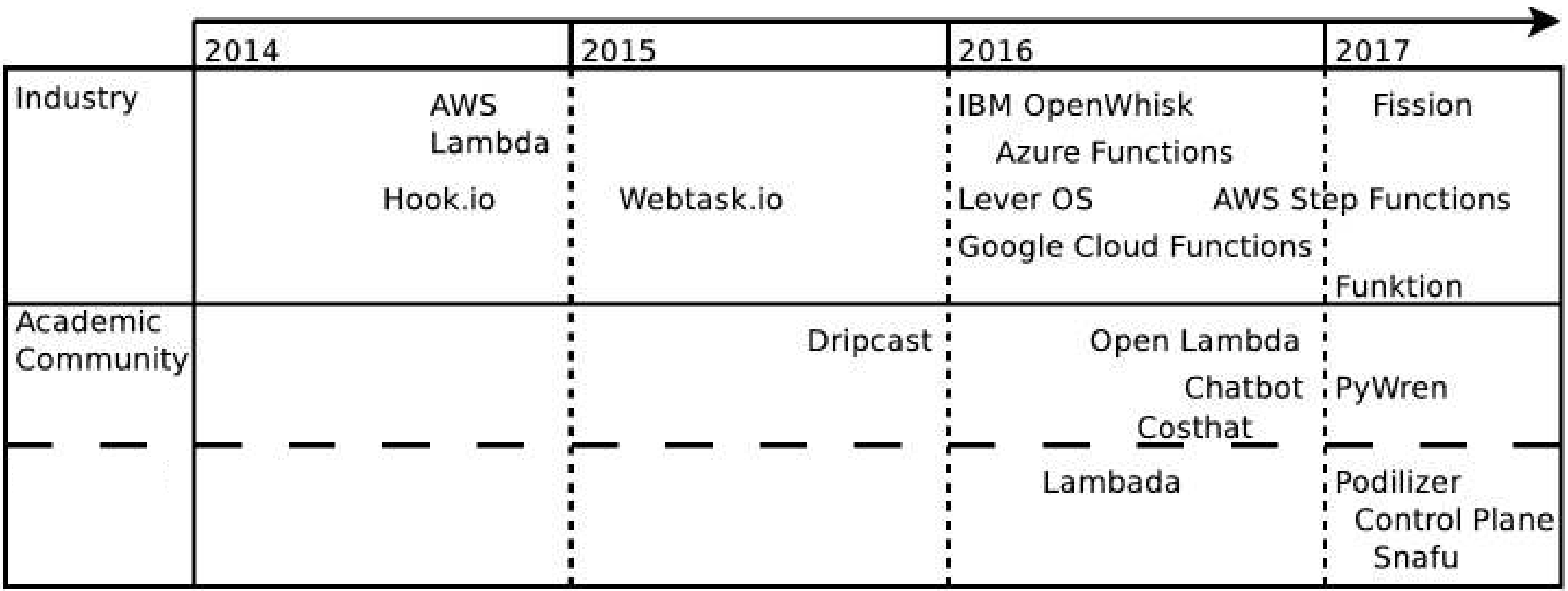
FaaS Positioning



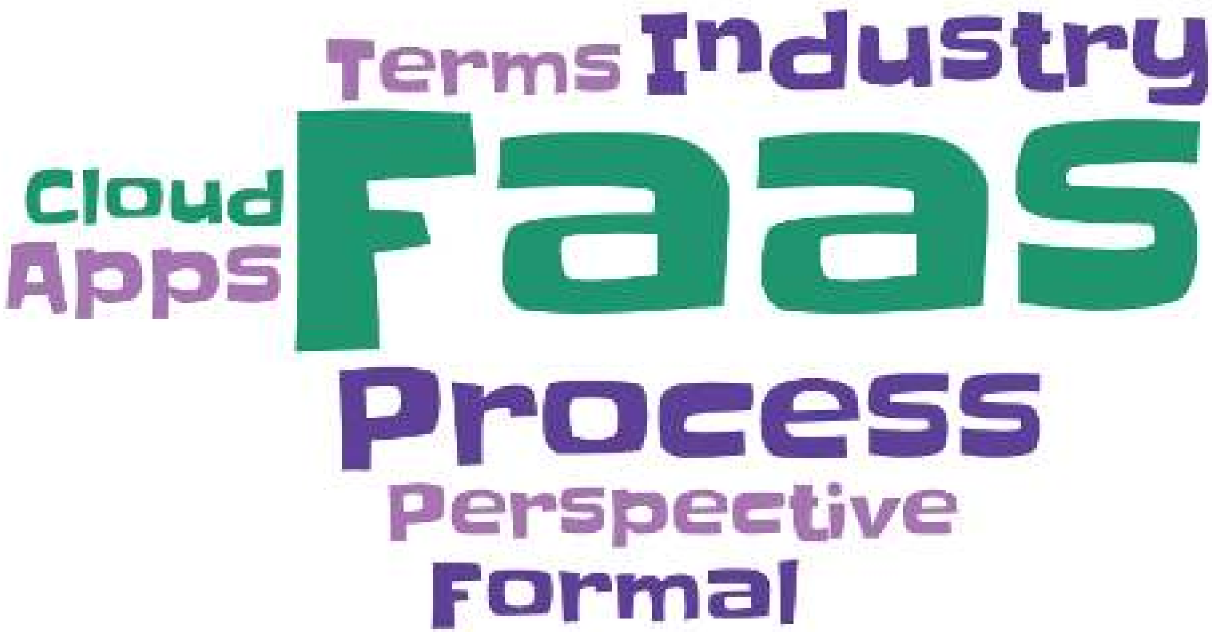
FaaS Positioning



FaaS Chronology



Wrap-Up Part I



Part II - Function Developer Tools

FaaS Synopsis: Python Examples

AWS Lambda:

```
def lambda_handler(event, context):
        "‘‘
        event: dict
        context: meta information object
        returns: dict, string, number, ...
        ‘‘‘
        # ...
        return “result“
```

OpenWhisk:

```
def handler(input):
        ‘‘‘
        input: dict
        returns: dict
        ‘‘‘
        # ...
        return {}
```

Fission:

```
def main():
        ‘‘‘
        input: via flask.request.get_data()
        returns: str
        ‘‘‘
        # ...
        return “result“
```

Azure Functions:

```
def main():
        from AzureHTTPHelper import\
        HTTPHelper
        input = HTTPHelper().post
        # ...
        open(os.environ[“res“], “w“).write(\
        json.dumps({“body“: “...“}))
main()
```

Further differences:

- function scoping (e.g. with/without export in JavaScript)
- function naming (mangling on client or service side)

FaaS Synopsis: JavaScript Examples

Provider	Syntax
AWS Lambda	exports.handler (ev, ctx, cb) > {cb(null, {});}
Google Cloud Functions	exports.fname = function(ev, cb){cb();}
Bluemix OpenWhisk	function main(<json>){return <json>}
Azure Functions	module.exports = function(ctx, req){ctx.req = <json>}
Webtask.io	module.exports = function(cb){cb(err, <json>);}
- Webtask.io advanced	module.exports function(ctx, cb){cb(err, <json>);}
Hook.io	module['exports'] function fname(hook){hook.res.end(<json>);}

Overlay Approach: PyWren

Improved conveyance of “serverless“ paradigm

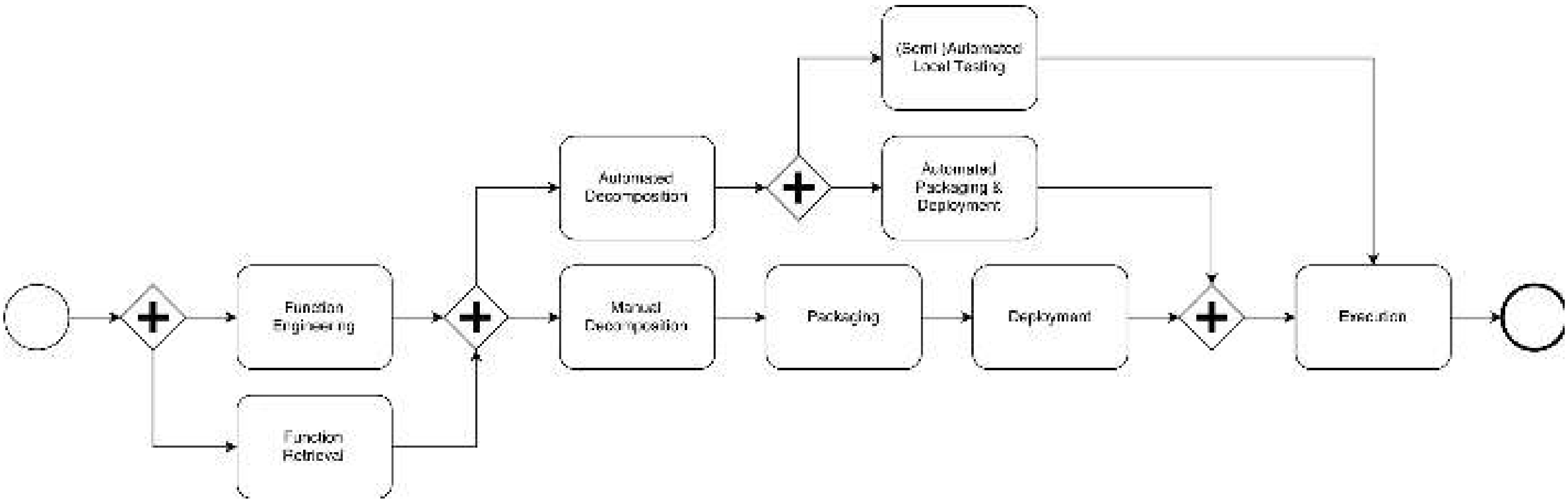
- no explicit deployment prior to execution
- rather, deploys while executing

```
def my_function(b):
  x = np.random.normal(0, b, 1024)
  A = np.random.normal(0, b, (1024, 1024))
  return np.dot(A, x)

pwex = pywren.default_executor()
res = pwex.map(my_function, np.linspace(0.1, 100, 1000))
```

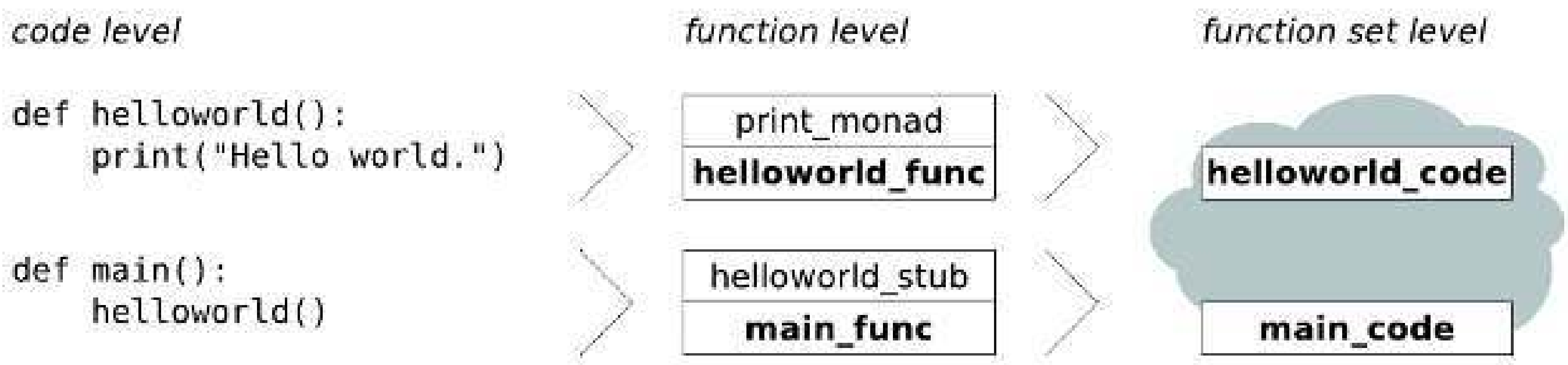
How it works:

- cloudpickle to AWS S3
- executes Lambda function which reads/writes from/to S3
- parallelisation through map functions

Programming Perspective



Transformation Overview

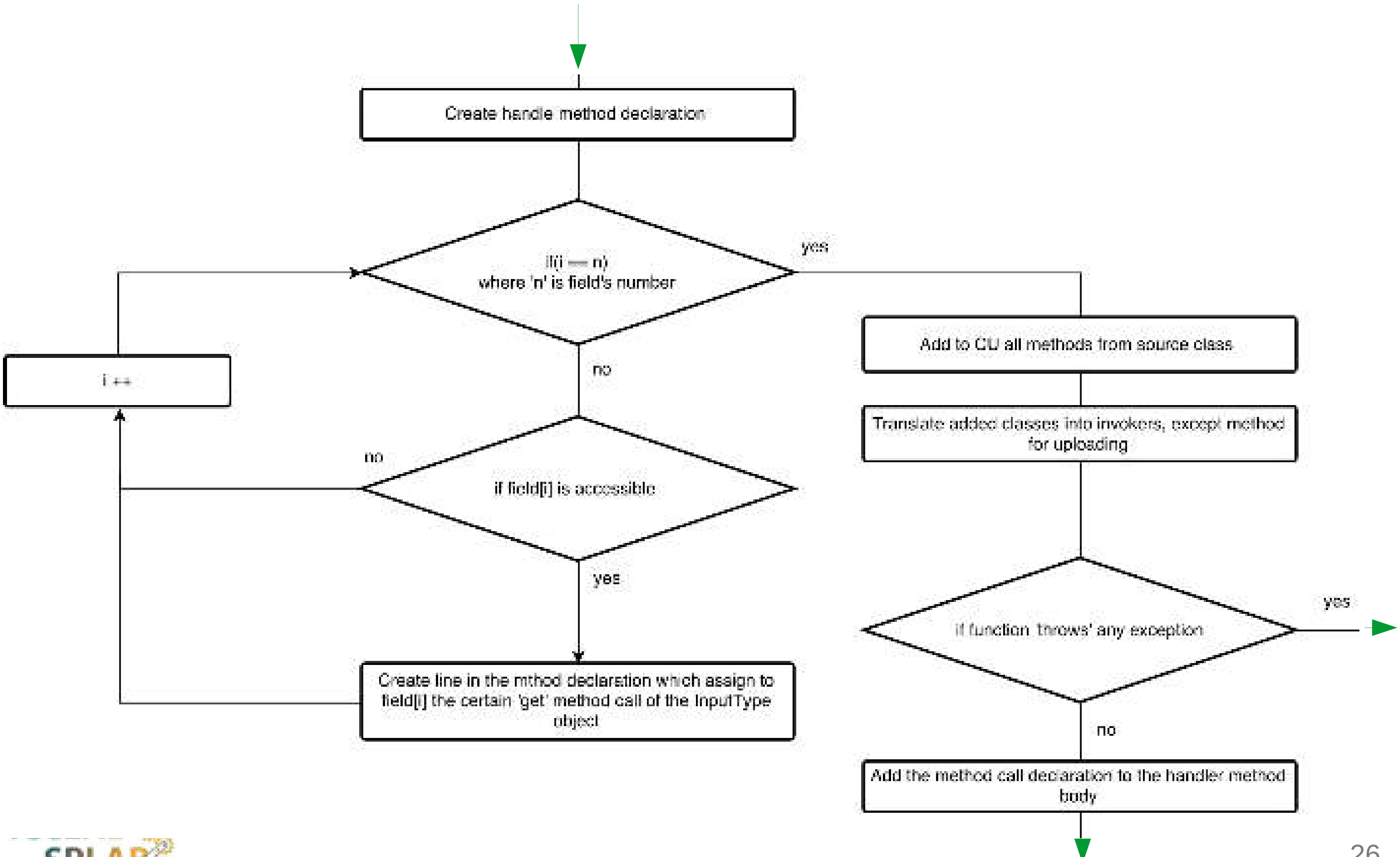


Transformation Rules

- entry points
  - no transformation of main function
- functions definitions
  - adapt to FaaS conventions: parameters, return value
  - scan recursively for function calls
  - export as function unit including dependencies
- function calls
  - if internal, rewire
  - if input/output, replace
  - otherwise, leave unchanged
- monads
  - functional programming with side effects (i.e. input/output as side channel)

ICCLAB SPLAB

Transformation Algorithm (Java Exc.)



FaaSification

Definition of "FaaSification"

→ Process of automated decomposition of software application into a set of deployed and readily composed function-level services.

<u>FaaSification := code analysis + transformation + deployment + on-demand activation</u>

Integration Categories:
- generic (code/function unit generation)
- single-provider integration
- multi-provider integration

Depth Categories:
- shallow (file to function)
- medium (function to lines)
- deep (line to many lines)

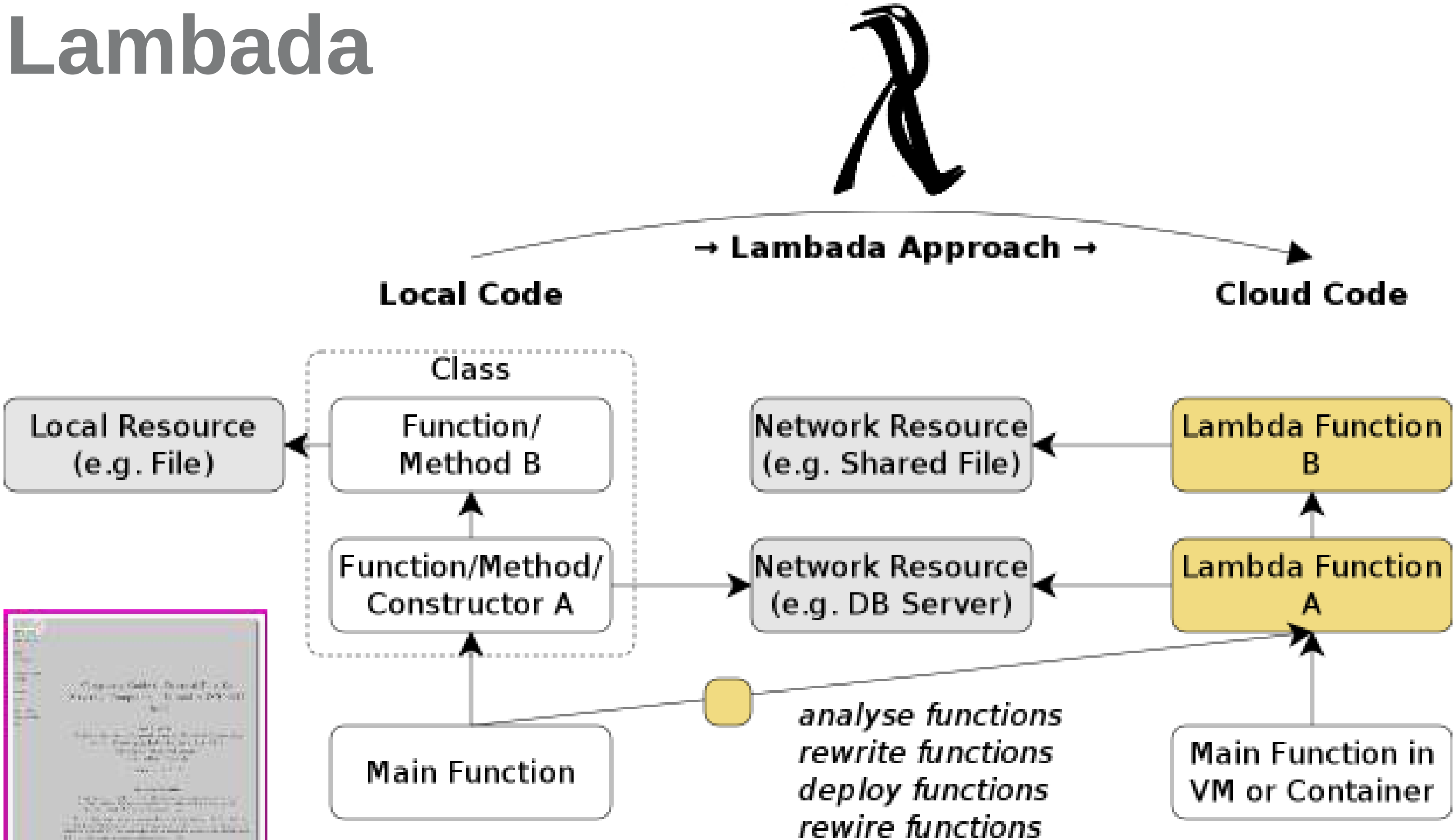
Decomposition Categories:
- static code analysis
- dynamic code analysis

"Lambdafication"
- targeting AWS Lambda

→ Lambada: FaaSification for Python
→ Podilizer, Termite: FaaSification for Java
(currently limited to Lambdafication)

ICCLAB SPLAB Service Engineering Research Area

Lambada



Nr. 1

ICCLAB SPLAB Service Engineering Research Area

Lambada

Code Analysis

Dependencies
- imported modules
- global variables
- dependency functions
  - defined in other module
  - defined in same module

Input/Output
- printed lines
- input statements
  - tainting
  - stateful function splitting

```
import time
import math

level = 12
counter = 0

def fib(x):
        global counter
        counter += 1
        for i in range(counter):
                a = math.sin(counter)
        if x in (1, 2):
                return 1
        return fib(x - 1) + fib(x - 2)

if __name__ == "__main__":
        fib(level)
```

Lambada

Code Transformation

Rewrite rules, via AST:

```
return 9
-------------------
return {“ret“: 9}
```

```
print(“hello“)
return 9
-----------------------------------------
return {“ret: 9“, “stdout“: “hello“}
```

```
local_func()
----------------------
local_func_stub()
```

Stubs, via templates:

```
def func_stub(x):
    input = json.dumps({“x“: x})
    output = boto3.client(“lambda“).invoke(FN=“func“, Payload=input)
    y = json.loads(output[“Payload“].read().decode(“utf-8“))
```

Lambada

Code Transformation

Stateful proxies for Object-Oriented Programming:

```
class Test:                              →    class Proxy:
     def __init__(self):                      def __new__(cls, clsname, p=True):
          self.x = 9                                    if p: # __new__ must return callable
                                                             return lambda: Proxy(clsname, False)
     def test(self):                                    else:
          return self.x * 2                                  return object.__new__(cls)

                                                   def __init__(self, clsname, ignoreproxy): ...
                                                   def __getattr__(self, name): ...
```

→ Test becomes Proxy(“Test“), Test() then invokes proxy
→ test() becomes remote_test({“x“: 9}) through network proxy class
→ automatically upon import of class

ICCLAB SPLAB

Lambada

Examples

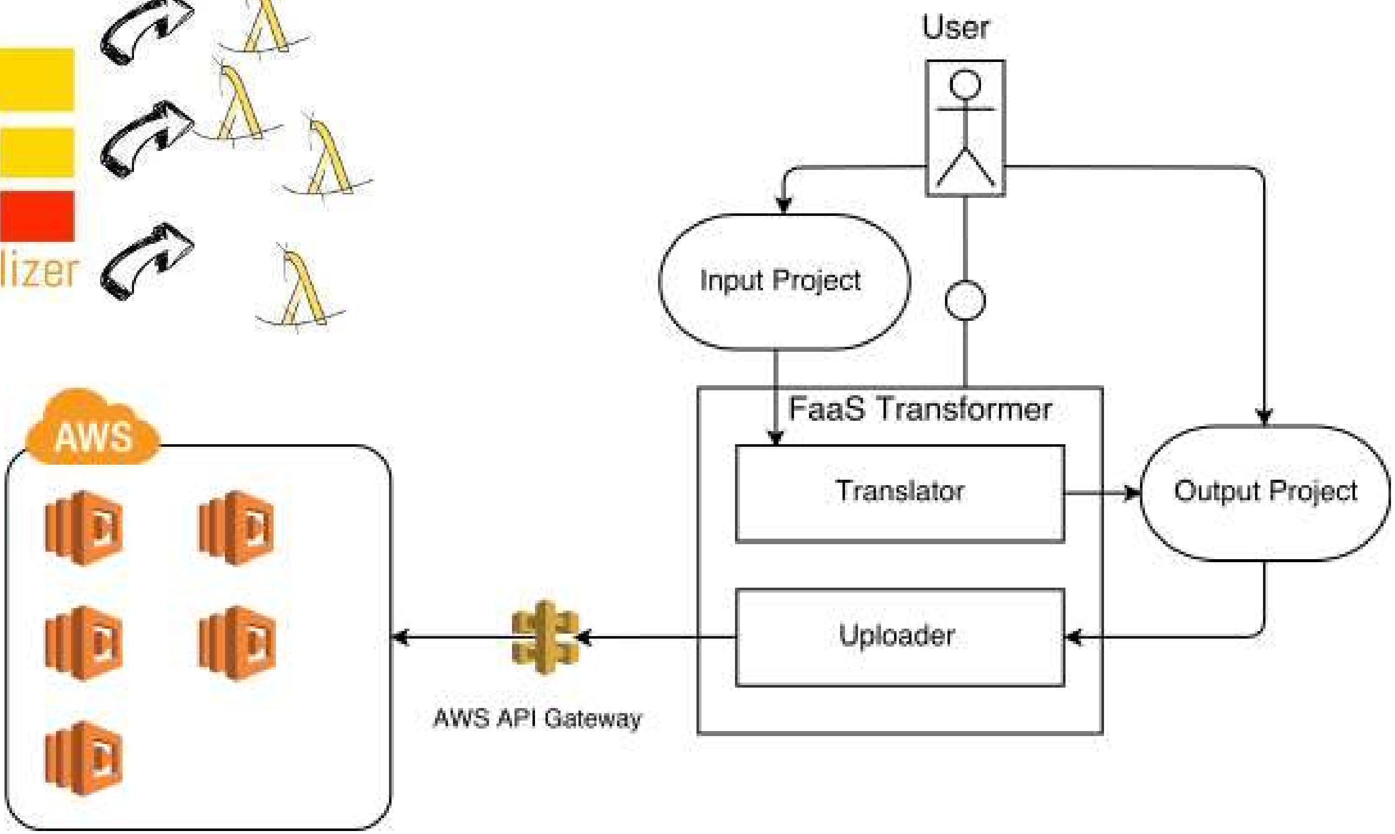
(not shown: monads, decorators)

```python
#!/usr/bin/env python3

import math
import csv

import lambada

def complextrig(v):
    print("// complextrig", v)
    return math.sin(v) + math.cos(v)

def calculate(values):
    return sum(map(complextrig, values))
```

```python
import csv
import math
import json
from boto3 import client as boto3_client
lambda_client = boto3_client('lambda')

# dep complextrig

def complextrig(v):
    msg = {"v": v}
    fullresponse = lambda_client.invoke(FunctionName="complextrig_lambda",
                                        Payload=json.dumps(msg))
    response = json.loads(fullresponse["Payload"].read())
    return response["ret"]


def calculate_lambda(event, context):
    values = event["values"]
    ret = sum(map(complextrig, values))
    return {'ret': ret}
```

Podilizer

Java application
Podilizer



User
Input Project
FaaS Transformer
Translator
Output Project
Uploader
AWS
AWS API Gateway

Podilizer

Examples

```java
package pkg;

class HelloWorld
{
    private String greeting()
    {
        return "Hello, world.";
    }

    public void greetworld()
    {
        System.out.println(this.greeting());
    }

    public static final void main(String args[])
    {
        new HelloWorld().greetworld();
    }
}
```

hosted function

local proxy method

Podilizer

Examples

```
public void greetworld() {
    String awsAccessKeyId = "# your Access Key Id provided by AWS";
    String awsSecretAccessKey = "# your Secret Access Key provided by AWS";
    String regionName = "# the AWS awsRegion for created Lambda functions. Example: 'awsRegion: eu-west-2'";
    String functionName = "pkg_HelloWorld_greetworld";
    Region region;
    AWSCredentials credentials;
    AWSLambdaClient lambdaClient;
    credentials = new BasicAWSCredentials(awsAccessKeyId, awsSecretAccessKey);
    lambdaClient = (credentials == null) ? new AWSLambdaClient() : new AWSLambdaClient(credentials);
    region = Region.getAwsRegion(Regions.fromName(regionName));
    lambdaClient.setAwsRegion(region);
    awsl.pkg.HelloWorld.greetworld.InputType inputType = new awsl.pkg.HelloWorld.greetworld.InputType();
    ObjectMapper objectMapper = new ObjectMapper();
    String json = "";
    try {
        json = objectMapper.writeValueAsString(inputType);
    } catch (JsonProcessingException e) {
        e.printStackTrace();
    }
   awsl.pkg.HelloWorld.greetworld.OutputType outputType = null;
    try {
        InvokeRequest invokeRequest = new InvokeRequest();
        invokeRequest.setFunctionName(functionName);
        invokeRequest.setPayload(json);
    outputType = objectMapper.readValue(byteBufferToString(
                lambdaClient.invoke(invokeRequest).getPayload(),
                Charset.forName("UTF-8")),awsl.pkg.HelloWorld.greetworld.OutputType.class);
    } catch(Exception e) {

        };
}
```

(proxy method)

SPLAB

Podilizer

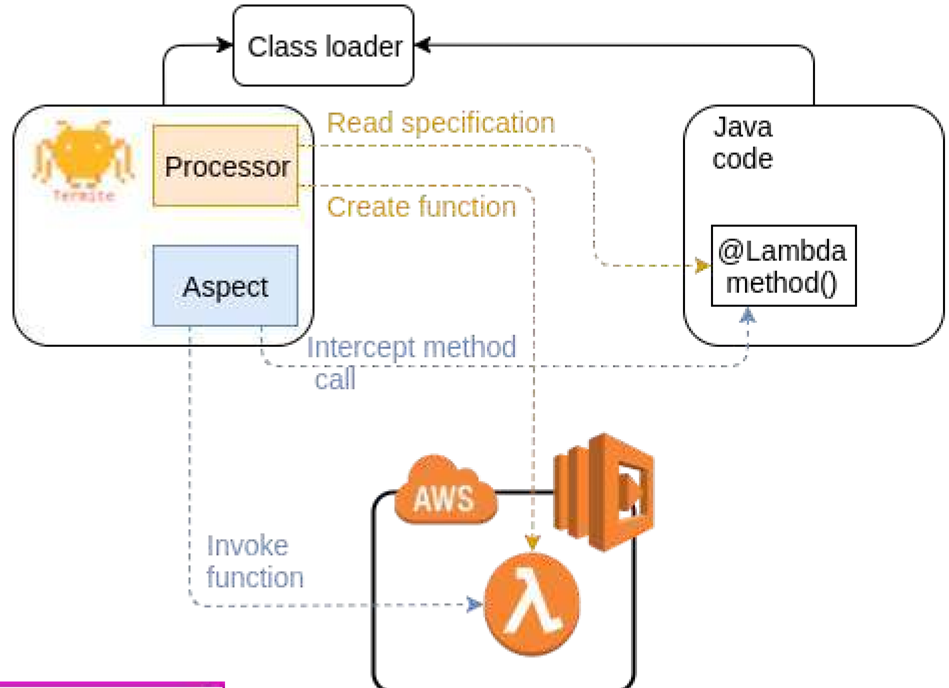
Examples

```
import java.nio.ByteBuffer;
import java.nio.charset.Charset;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import pkg.*;

public class LambdaFunction implements RequestHandler<InputType, OutputType> {

    public OutputType handleRequest(InputType inputType, Context context) {
        greetworld();
        {
            OutputType outputType = new OutputType();
            return outputType;
        }
    }

    private String greeting() {

    public static final void main(String args[]) {
        new HelloWorld().greetworld();
    }

    public static String byteBufferToString(ByteBuffer buffer, Charset charset) {
        byte[] bytes;
        if (buffer.hasArray()) {
            bytes = buffer.array();
        } else {
            bytes = new byte[buffer.remaining()];
            buffer.get(bytes);
        }
        return new String(bytes, charset);
    }

    public void greetworld() {
        System.out.println(this.greeting());
    }
}
```

(hosted function)

Termite

Annotations:
@Lambda(region=...,
memory=...,timeout=...)

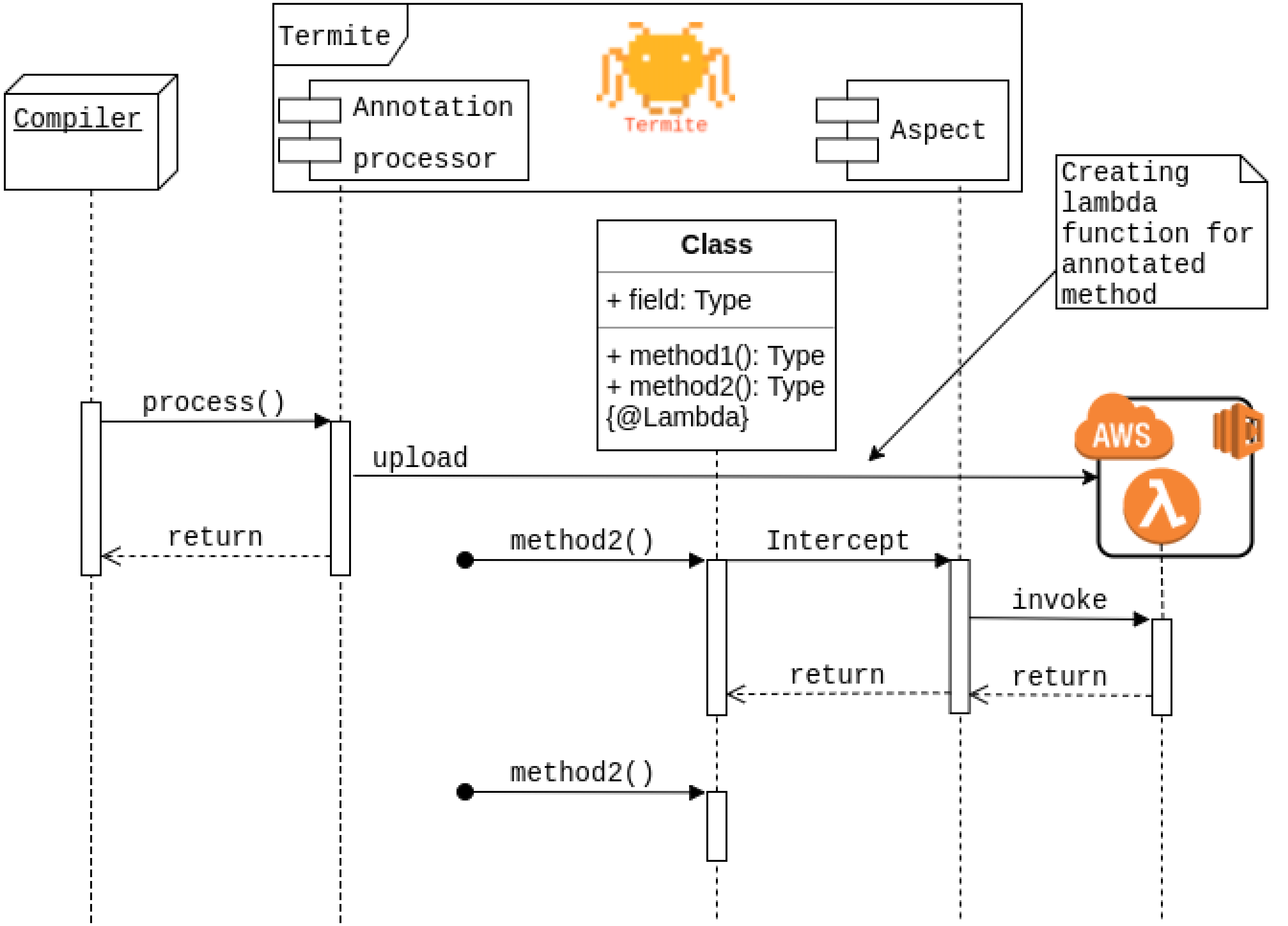


Nr. 2

Termite

Workflow

(Maven
+ AspectJ
integration)



Wrap-Up Part II


HAVE A
BREAK

Part III - Function Execution Tools

The FaaS Space

AWS Lambda

Google Cloud Platform
Functions

OpenWhisk

Azure

webtask

hook.io

OVH Functions

Chalice
[Lambda]

Zappa
[Lambda]

PyWren
[Lambda]

Dawson
[Lambda]

Apex
[Lambda]

ICCLAB
SPLAB

Serverless Framework
[Lambda, OW, GCF, AF]

FaaS (Docker)

Kubeless

Funktion

Fission

Picasso

Fn

Docker-LambCI

Effe

Lever OS

OpenLambda

Snafu

Whisk-Mocha
[OpenWhisk]

X-Ray
[Lambda]

Step Functions
[Lambda]

MR Refarg
[Lambda]

LambDash
[Lambda]

Podilizer

Termite

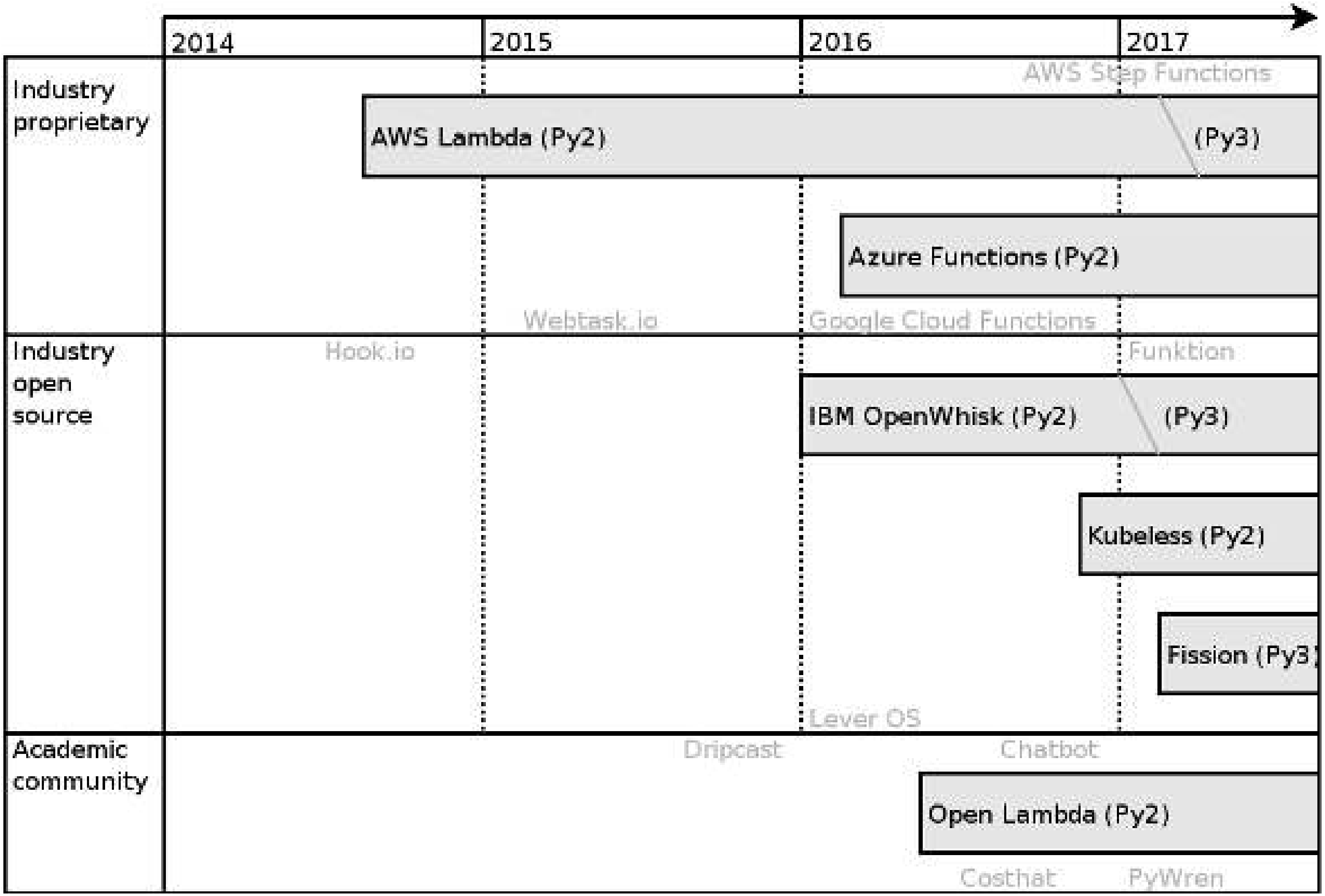
Lambada

Runtime Overview: Providers&Stacks

Implementation	Languages	Availability
AWS Lambda	Node.js, Java, Python / C#	Service
Google Cloud Functions	Node.js	Service
Apache OpenWhisk	Node.js, Swift, Docker* / Python	OSS
→ IBM Cloud Functions	-"-	Service
Azure Functions	Node.js, C# / F#, Python, PHP, ...	Service
Webtask.io	Node.js	OSS + Service
Hook.io	Node.js, ECMAScript, CoffeeScript	OSS + Service
Effe	Go	OSS
OpenLambda	Python	Academic + OSS
LambCI Docker-Lambda	Node.js	OSS (re-engineered)
Lever OS	Node.js, Go	OSS
Fission	Node.js, Python	OSS
Funktion	Node.js	OSS
Kubeless	Python	OSS
IronFunctions	Node.js, Java, Python, Go, ...	OSS
→ Fn	-"-	OSS

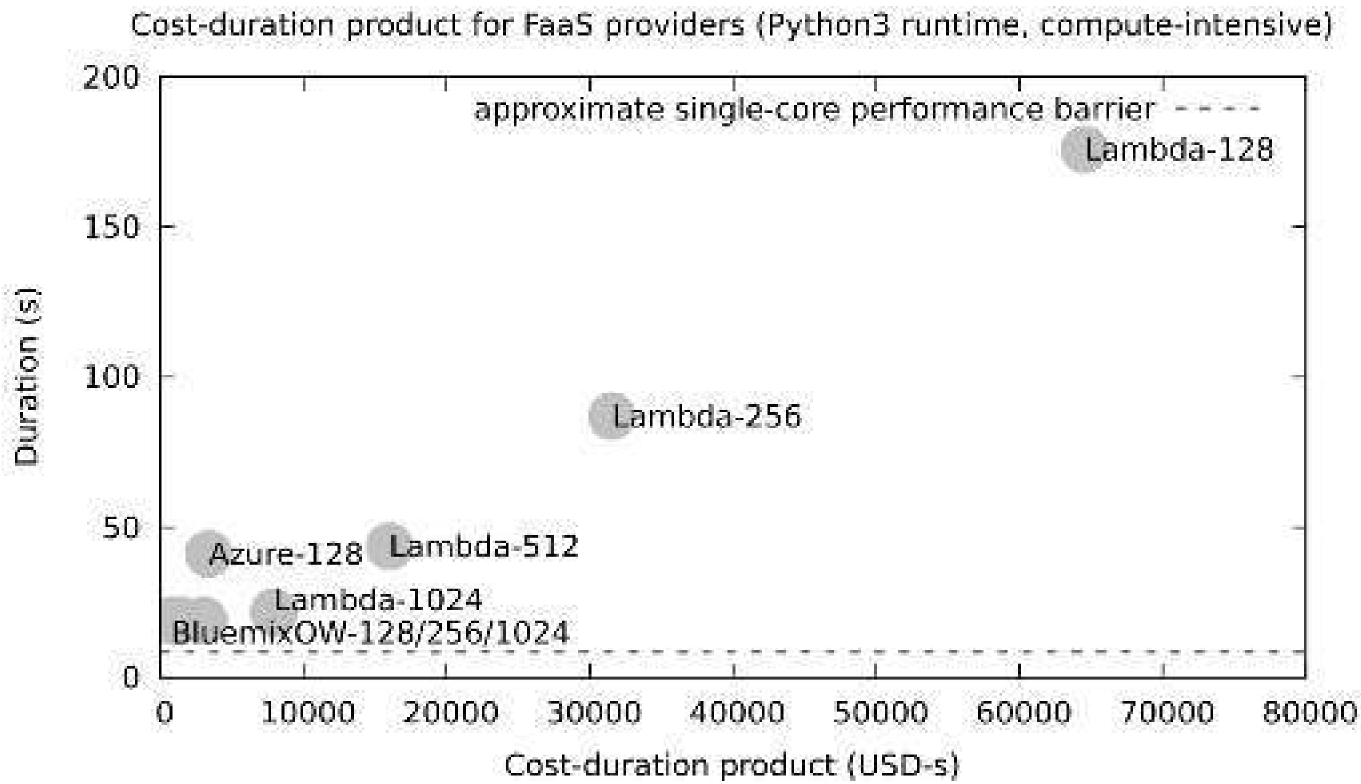
ICCLAB SPLAB Service Engineering Research Area

Runtime Overview: Python Examples



Runtime Overview: Provider Pricing

Service	Monthly Free Tier	Cost per Call	Cost per Load
AWS Lambda	1e6 calls, 4e5 load	2.00e-7	1.667e-5
Google Cloud Functions	unknown, service is in alpha stage		
IBM OpenWhisk	unknown, service is in beta stage		
Azure Functions	1e6 calls, 4e5 load	2.00e-7	1.600e-5
Webtask.io	–	1.68e-7 (paid plan)	–
Hook.io	–	2.00e-3 (small plan)	–



Runtime Overview: Provider Instances

Amazon EC2

accelerated

Model	vCPU	CPU Credits / hour	Mem (GiB)	Storage
t2.nano	1	3	0.5	EBS-Only
t2.micro	1	6		EBS-Only
t2.small	1	12	2	EBS-Only
t2.medium	2	24	4	EBS-Only
t2.large	[illegible]	36	[illegible]	EBS-Only
[illegible]	4	[illegible]	[illegible]	[illegible]

general-purpose

Model	vCPU	Mem (GiB)	Storage	Dedicated EBS Bandwidth (Mbps)
c4.large	2	3.75	EBS-Only	[illegible]
c4.xlarge	4	7.5	EBS-Only	750
c4.2xlarge	8	15	EBS-Only	1,000
c4.4xlarge	[illegible]	30	EBS-Only	2,000

compute-optimised

AWS Lambda

Lambda @ Edge

Memory (MB)
128
192
256
320
384
448
512
576

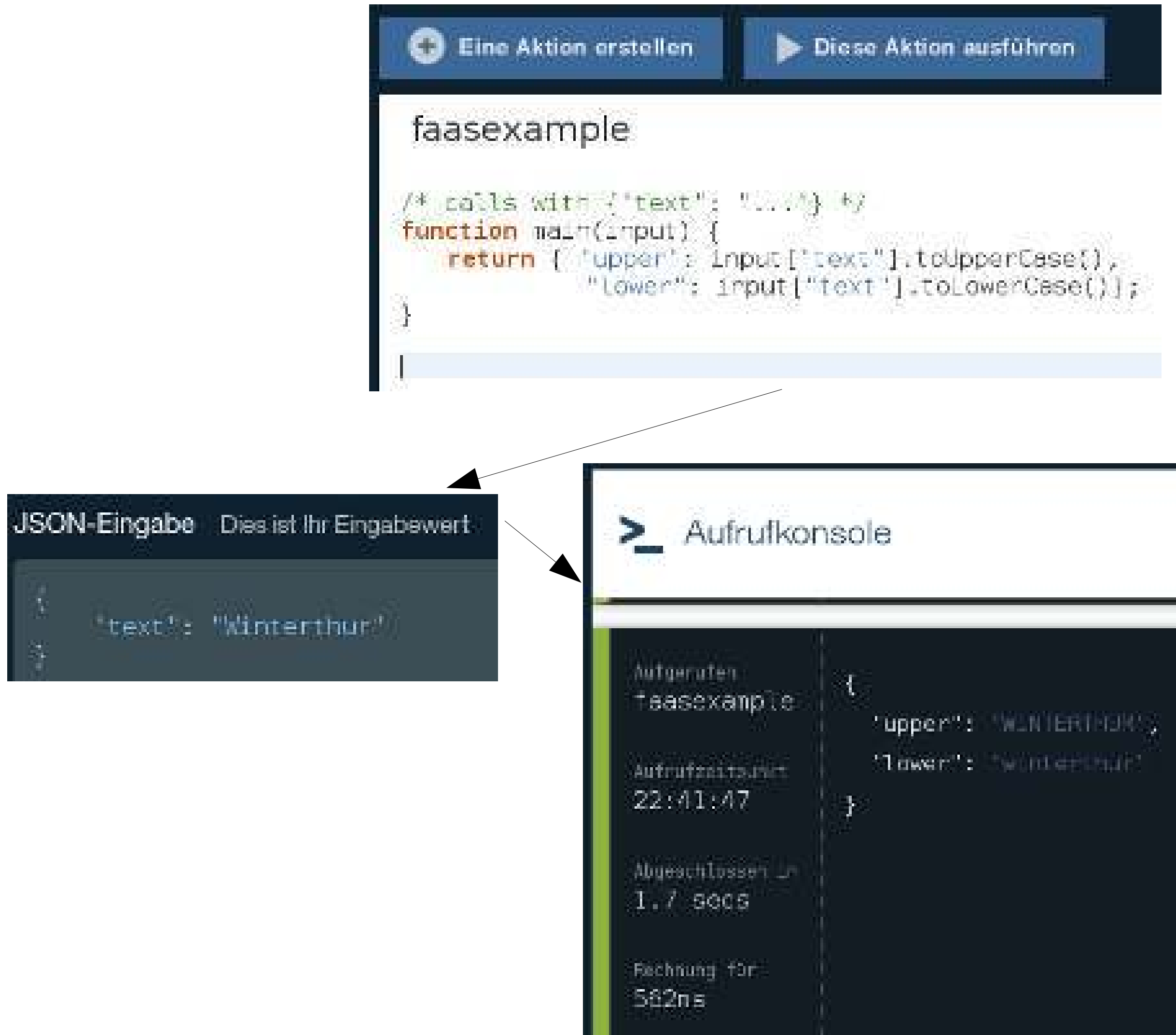
(CPU performance proportional to memory allocation)

Runtime Examples: IBM Cloud

Bluemix OpenWhisk & JavaScript/Node.js

Constraints
- code size 48 MB
- payload size 1 MB

Configuration
- runtime environment (from list)
- memory 128-512 MB; default 256
- timeout 0.1-300 s; default 60
- authoring: template or blank document

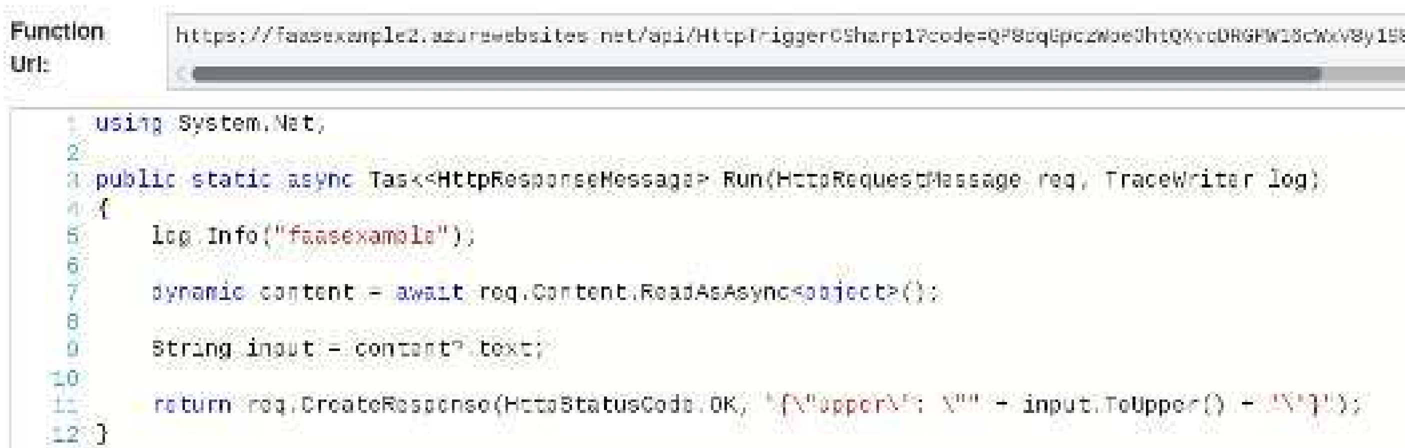


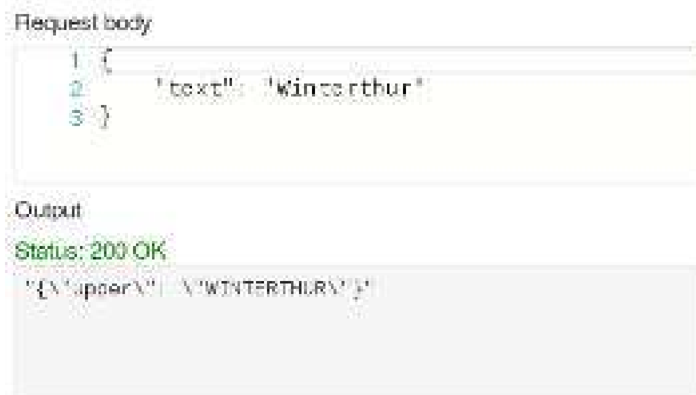
Runtime Examples: Azure Functions

Azure Functions

Constraints
- undocumented

Configuration
- runtime environment (from list)
- memory 128-1536 MB; default 256





Runtime Examples: Reality Check

OpenWhisk

~~$ wsk~~

does not compile

~~# openlambda~~

~~$ bin/admin~~

(invoke read error)

Azure

~~$ az~~

not scriptable

~~$ kubeless~~

breaks minikube

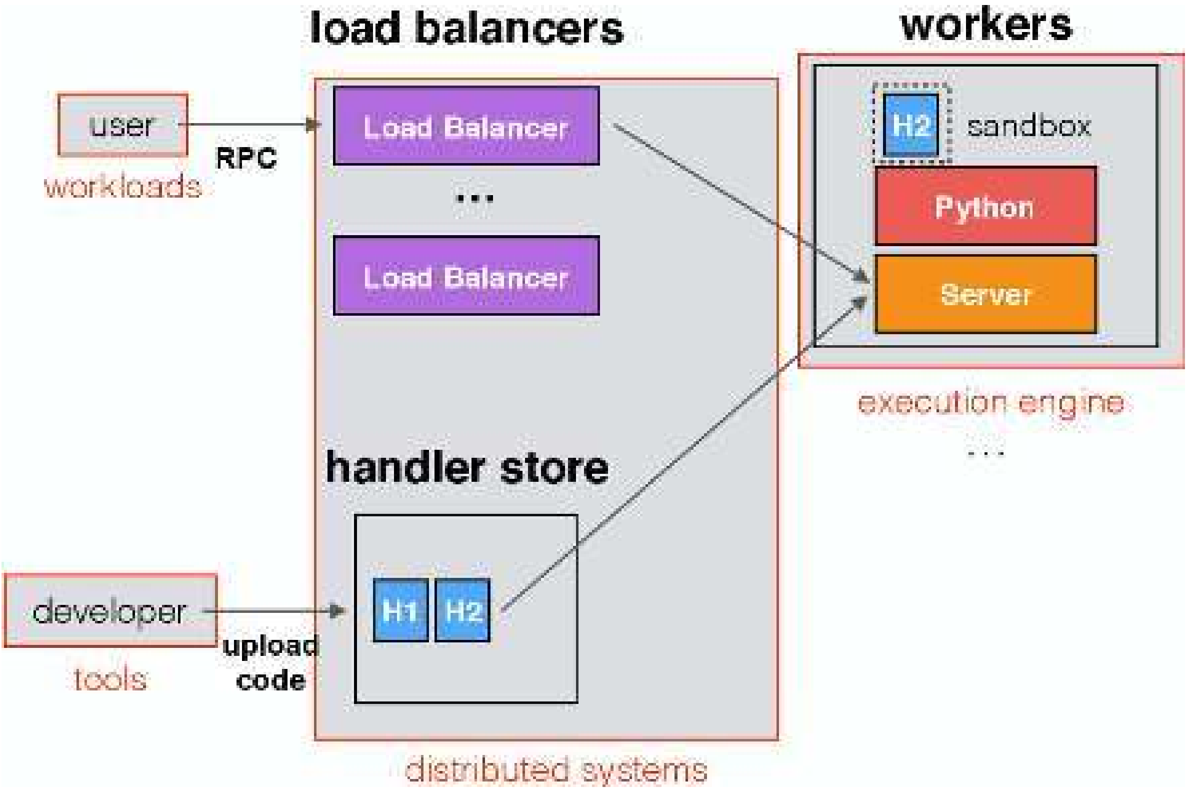
AWS Lambda

~~$ aws lambda~~

requires account

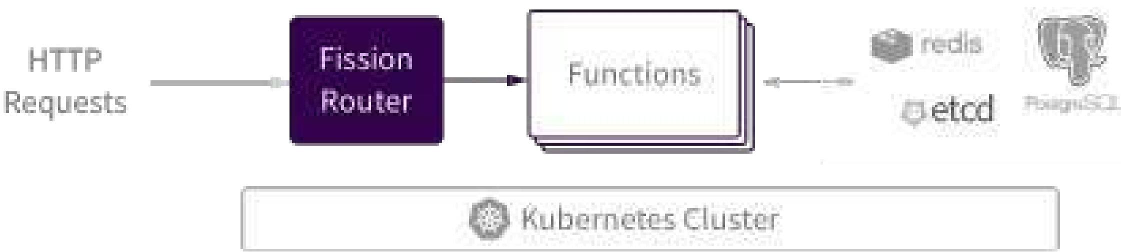
$ fission

Runtime Examples: OpenLambda



Nr. 5

Runtime Examples: Fission

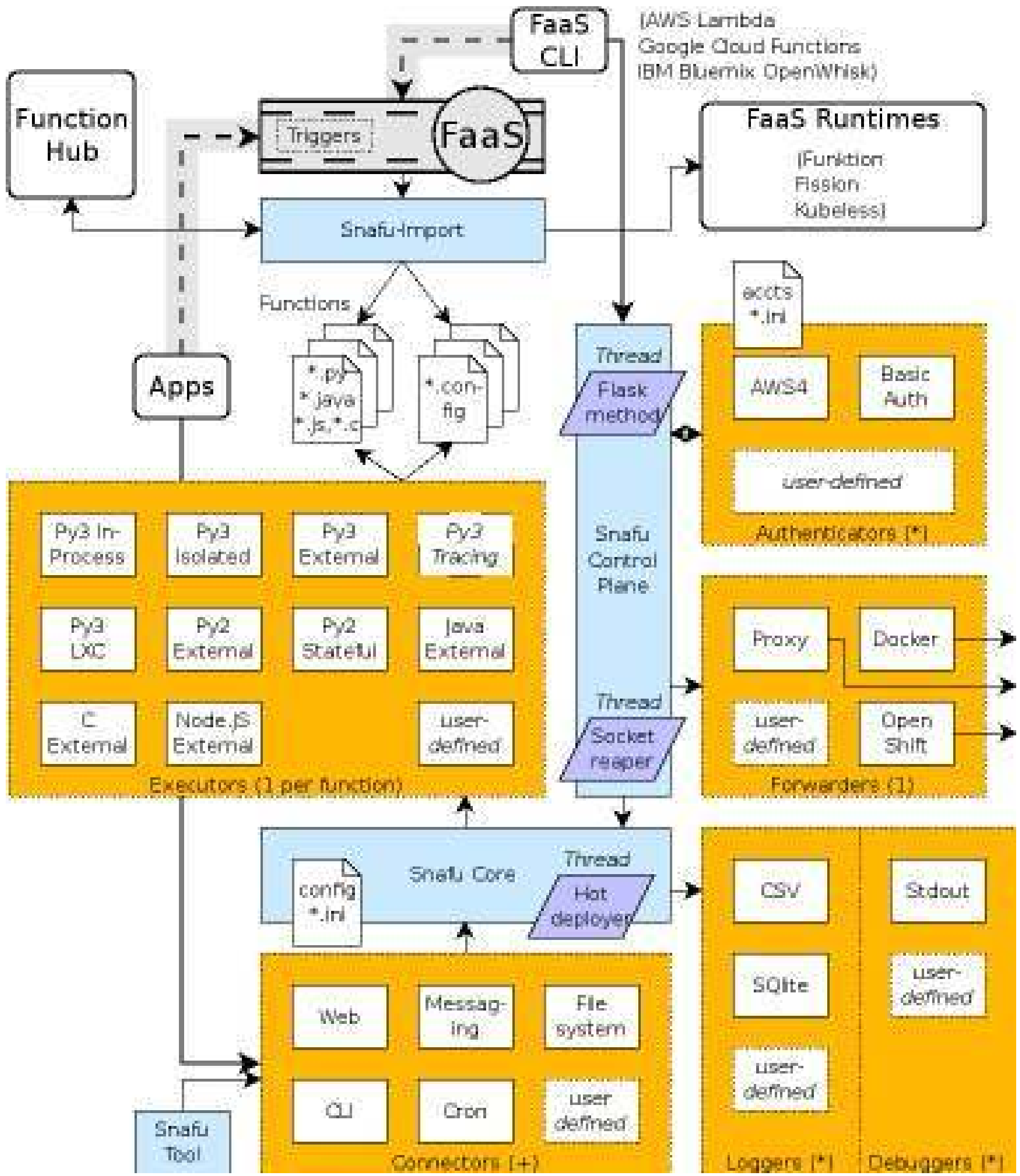


Nr. 4

Runtime Examples: Snafu

“Swiss Army Knife“ of Serverless Computing

Nr. 3



Snafu in Commercial Clouds

APPUiO

- OpenShift atop IaaS providers or on-premise (Cloudscale, AWS, ...)
- composite application deployment w/ templates

SWISS CONTAINER PLATFORM

Snafu: OpenShift template

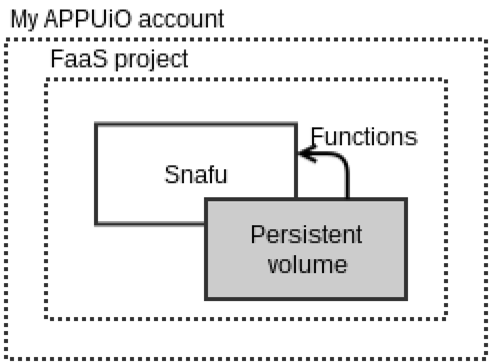
- deploys unprivileged Docker container(s)
- object hierarchy

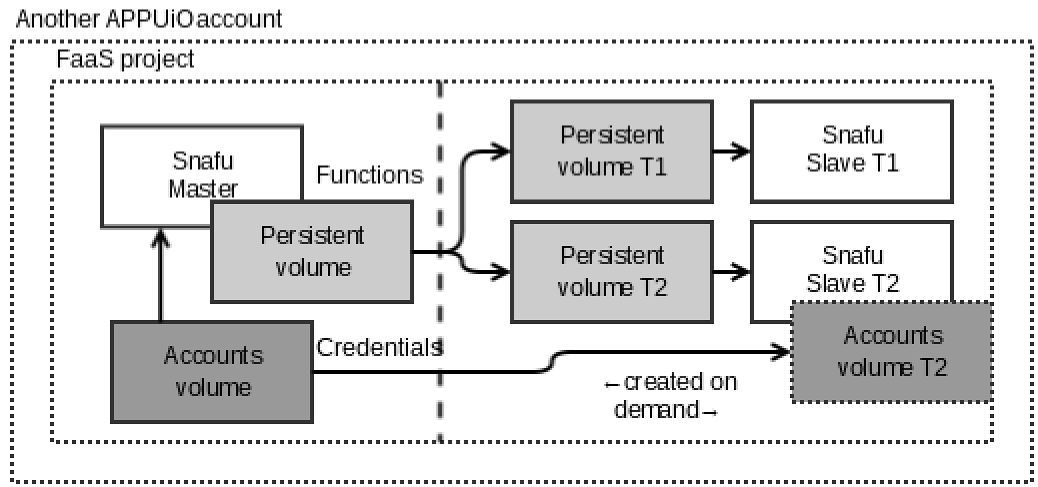
```
Template
    ImageStream
        DockerImage
    DeploymentConfig
        ImageStreamTag
    Service
    Route
        Service
    ConfigMap
```

```
...
containers:
- args:
    - --authenticator=aws
  command:
    - /opt/snafu-control
    - -s
    - /etc/snafu/snafu.ini
  image: snafu
...
```

Service Engineering Research Area ICCLAB SPLAB

Snafu in Commercial Clouds





Single-tenant mode
- unauthenticated
- single account
- multiple accounts in same instance

Multi-tenant mode
- multiple isolated accounts
- proper scale-out per active account
- selective sharing of functions and accounts from master to tenant instances

Snafu in Research Clouds

CloudLab

Utah, Wisconsin, South Carolina, Massachusetts, ...

- Emulab testbed
- raw access from physical level
- programmable infrastructure via GENI libraries
- >50000 experiments

Snafu: RawPC setup

- Ubuntu 16.04 base
- Snafu repository clone

Experiments:

- boot time → 5-7 min

European Grid Initiative Federated Cloud

Jülich, Fraunhofer SCAI, GRnet, INFN, Cesnet, GWDG, ...

- diverse open stacks and standards
- virtual machines, containers
- >250000 VM instances per year

Snafu: Kubernetes VM

- deployment descriptor
- based off OpenShift templates

Experiments:

- deployment time → 5 s

CloudLab Deployment

```
# partially generated

import geni.portal as portal
import geni.rspec.pg as pg
import geni.rspec.emulab as emulab
pc = portal.Context()

request = pc.makeRequestRSpec()

node = request.RawPC('node')
node.disk_image = 'urn:publicid:IDN+emulab.net+image+emulab-ops: \
  UBUNTU16-64-STD'
node.Site('Site 1')

node.addService(pg.Execute(shell="sh", command="sudo git clone \
  http://github.com/serviceprototypinglab/snafu /local/snafu"))

pc.printRequestRSpec(request)
```

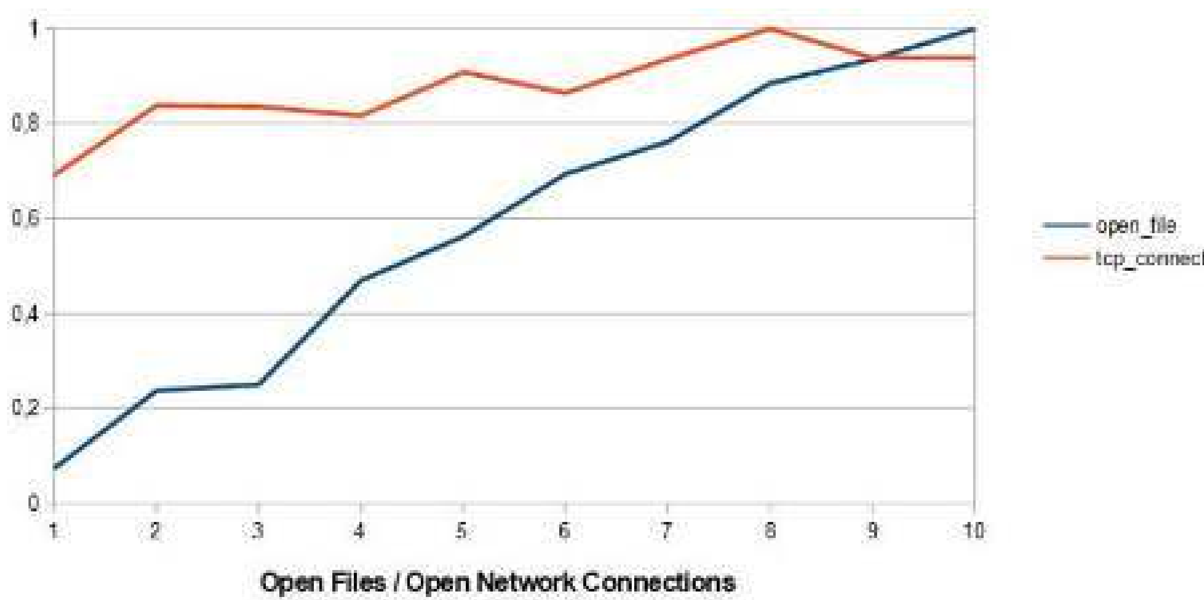
EGI Deployment

```
# on local system
kubectl config set-cluster egi --server=https://HOST.fedcloud-tf. \
  fedcloud.eu
kubectl config set-context egi --cluster=egi --user=josef
kubectl config use-context egi

wget -qc https://raw.githubusercontent.com/serviceprototypinglab/ \
  snafu/master/openshift/snafu-control-template.yaml
oc -n zhaw-test1 process -f snafu-control-template.yaml -o yaml \
  > snafu-control-deployment.yaml
scp snafu-control-deployment.yaml YOU@HOST.fedcloud-tf.fedcloud.eu

# remotely on server
kubectl create -f snafu-control-deployment.yaml

# on local system
aws --endpoint-url https://faas.fedcloud-rt.fedcloud.eu:31000 \
  lambda list-functions
```

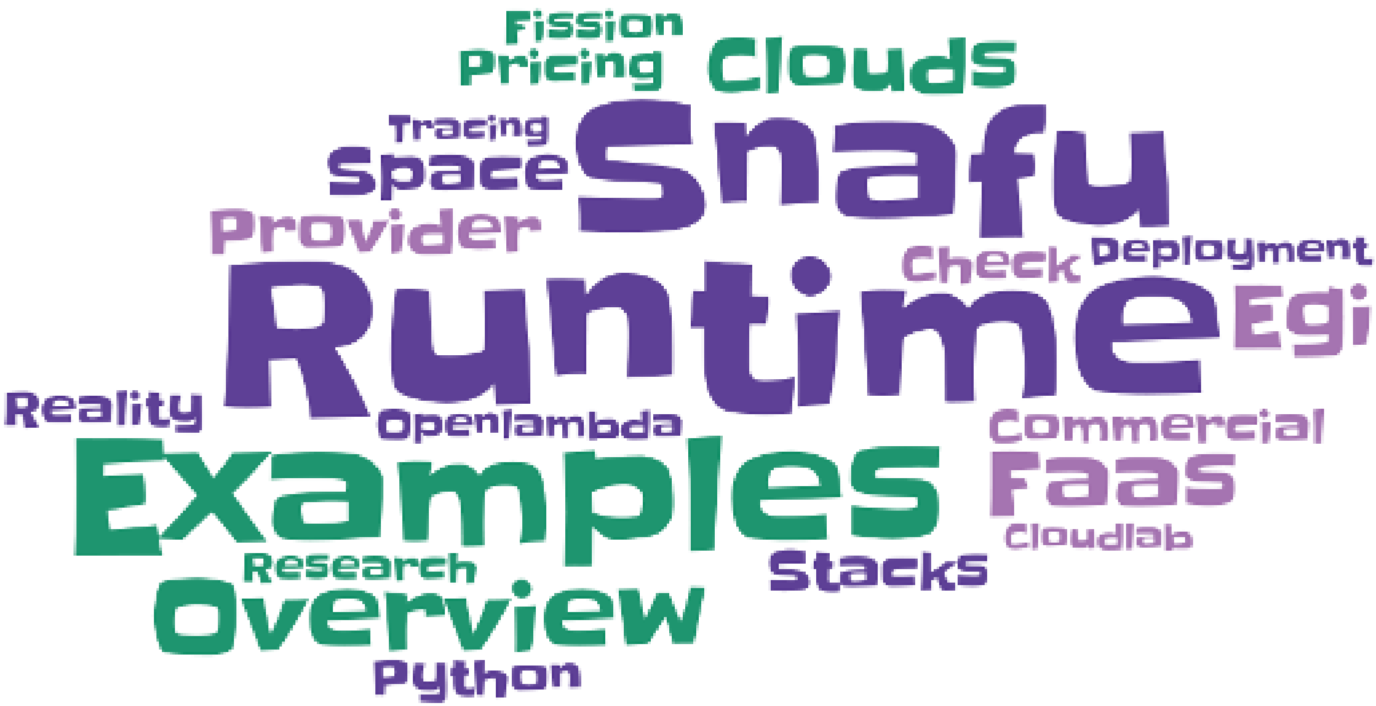
Tracing in Snafu

Trace	Method	Root Access
Tracking function calls	investigating stackframe	NO
Function execution time	investigating stackframe	NO
Network Calls	psutil	YES
Writes and reads to disk	psutil	NO
Performance metrics	psutil	NO

preliminary results



Wrap-Up Part III



Part IV - Research Challenges

ICCLAB SPLAB Service Engineering Research Area

Research Overview

refine by year
2017 (17)
2016 (8)
2015 (4)
2013 (1)
2012 (2)
2011 (3)
2010 (4)
2009 (1)
2008 (2)
2007 (2)

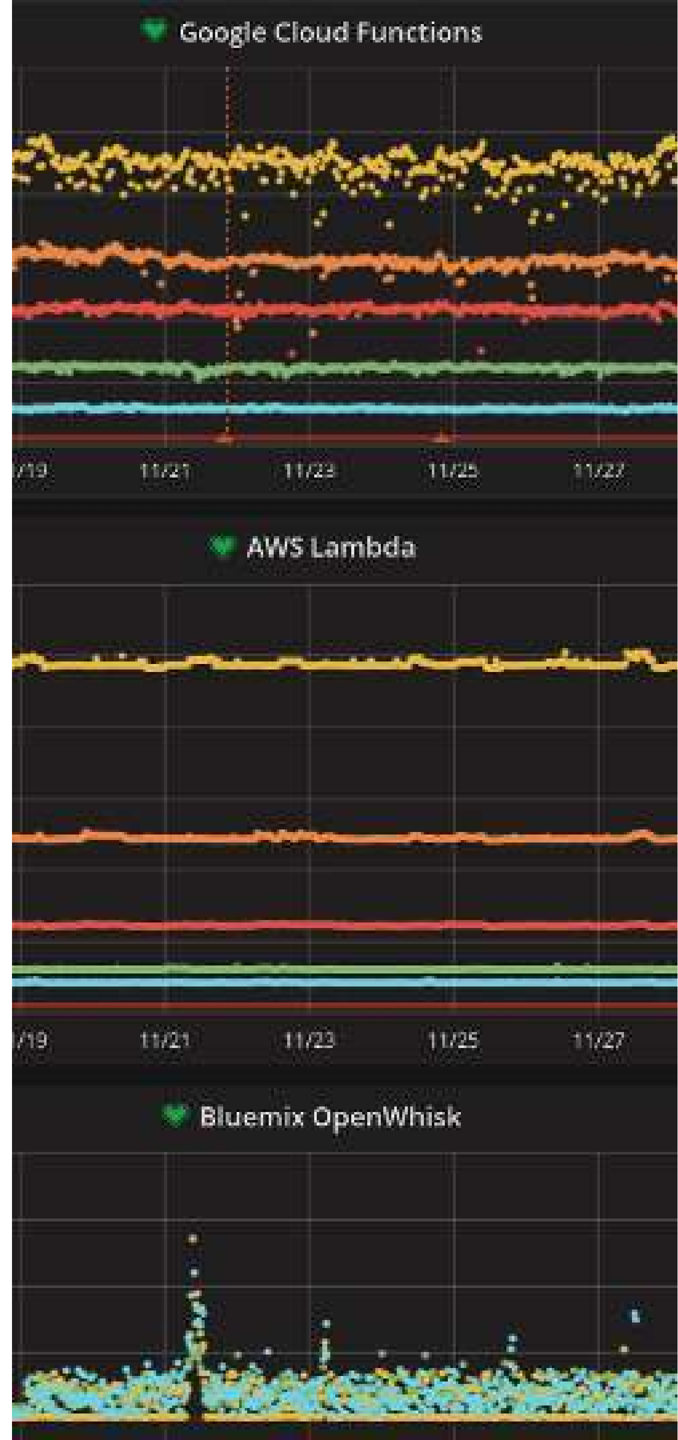
CostHat

$$c(s,\zeta) = \sum_{(s_e,n)\in w_s^\zeta} c^\lambda(s_e,n)$$

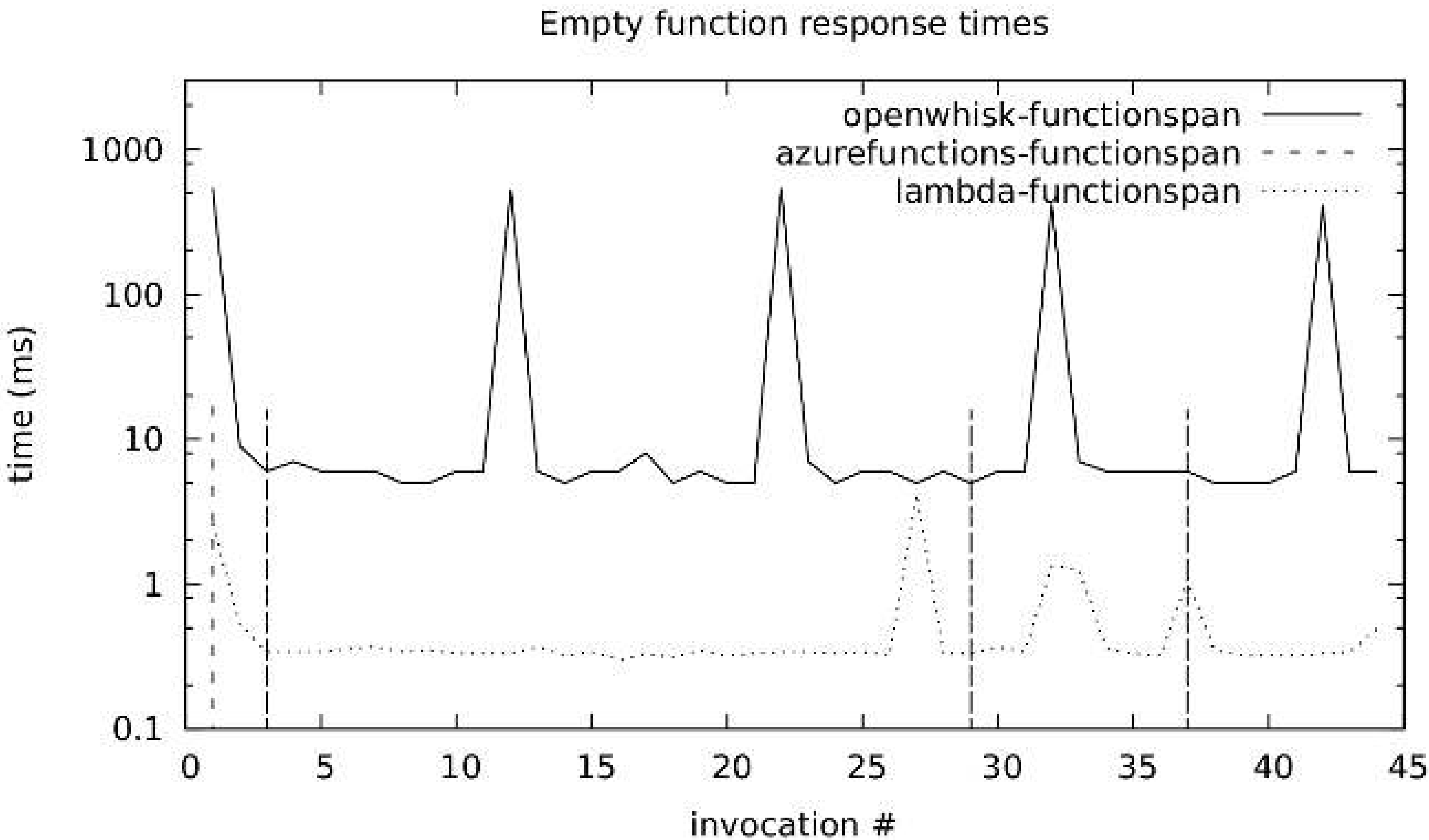
$$c^\lambda(e,n) = n * \left(\sum_{c_f \in CF_\lambda} c^\lambda_{c_f}(e) \right) + c^\lambda_{c_\xi}(e)$$

Second International Workshop on Serverless Computing (WoSC) 2017

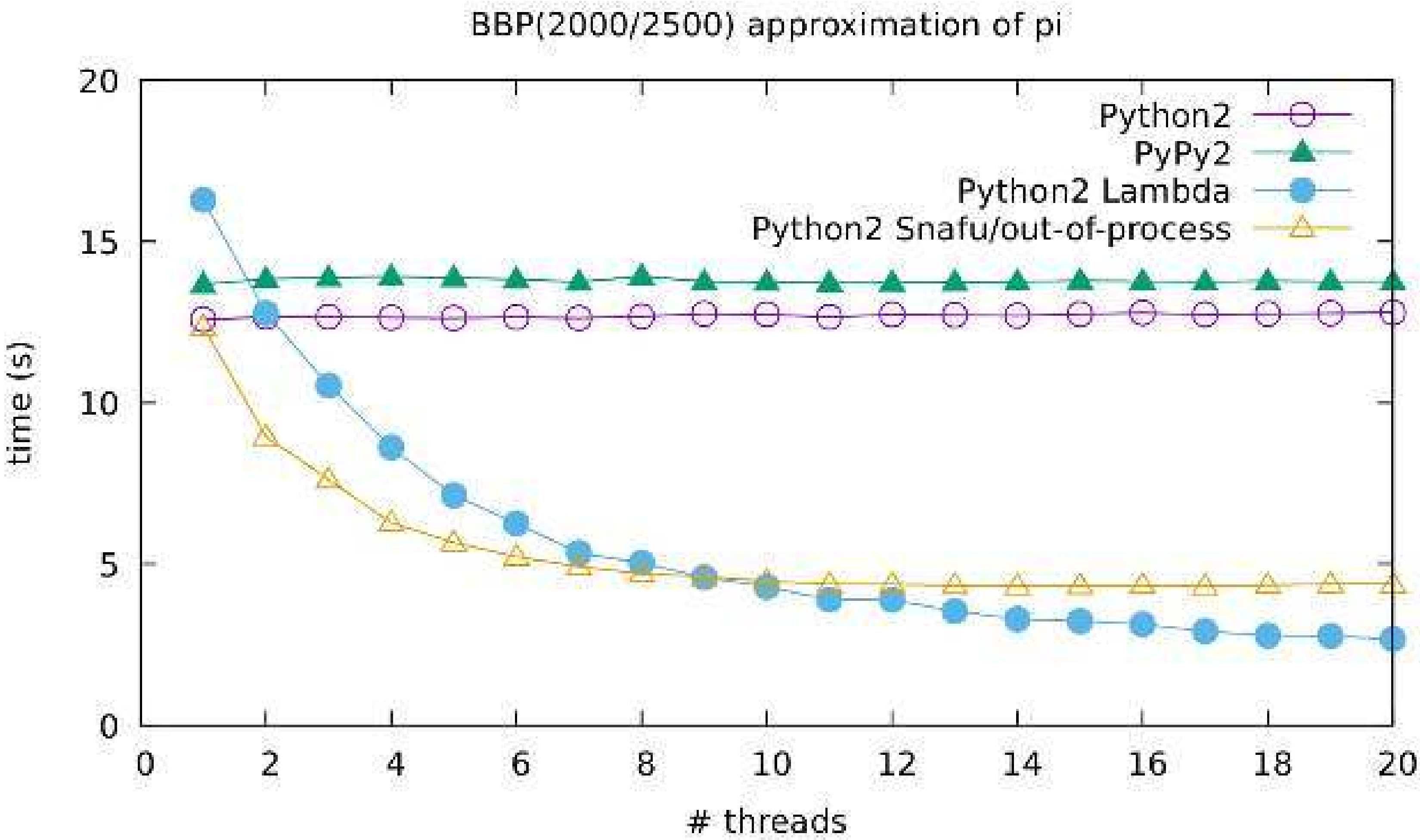
Part of Middleware 2017.



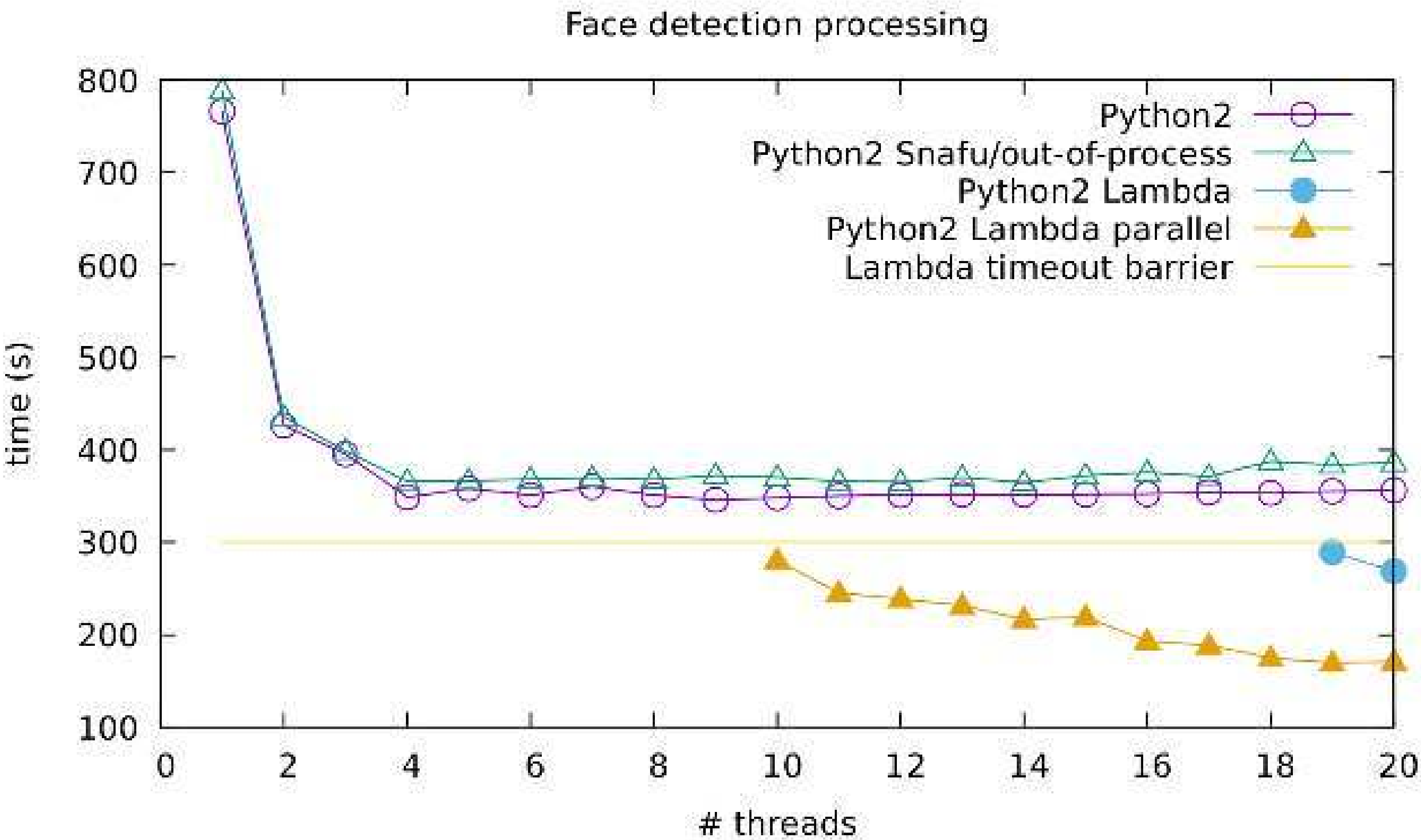
Challenge: Performance



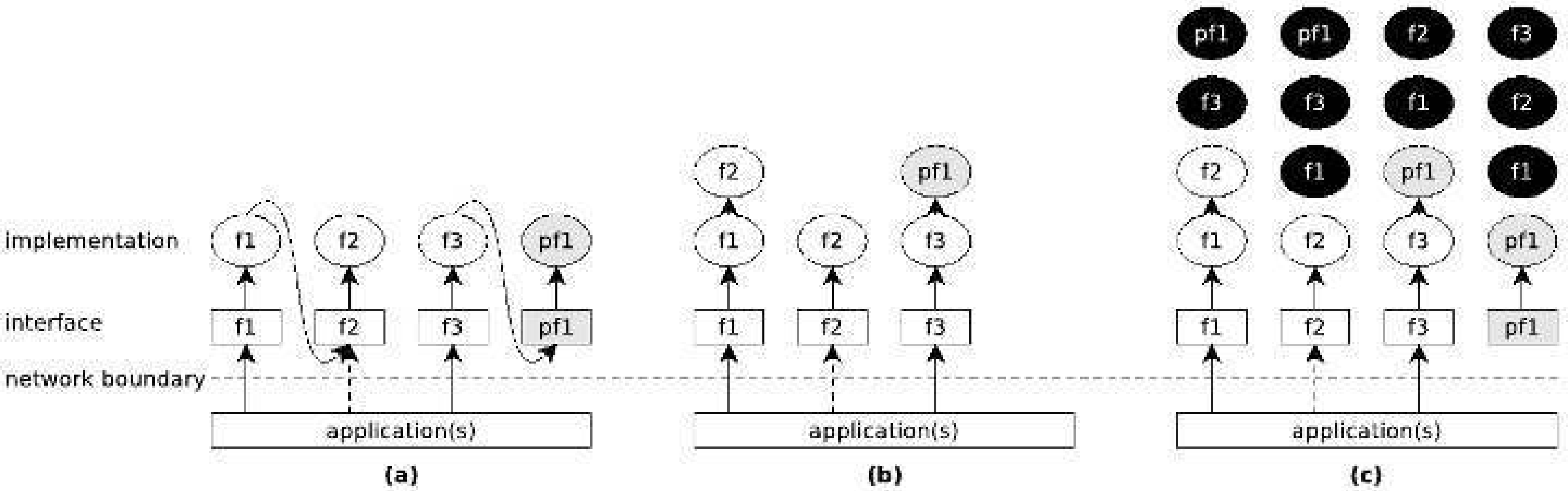
Challenge: Performance



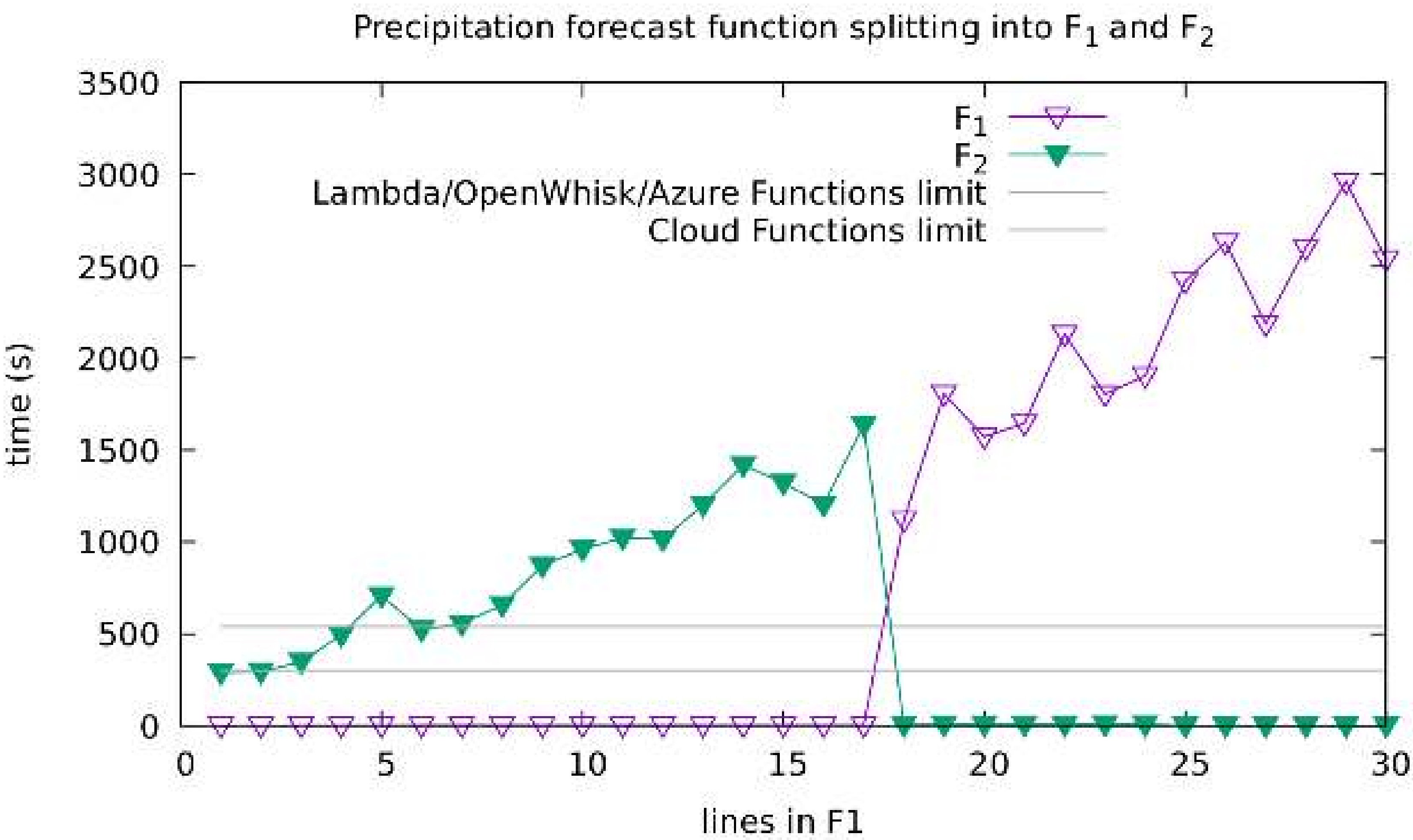
Challenge: Performance



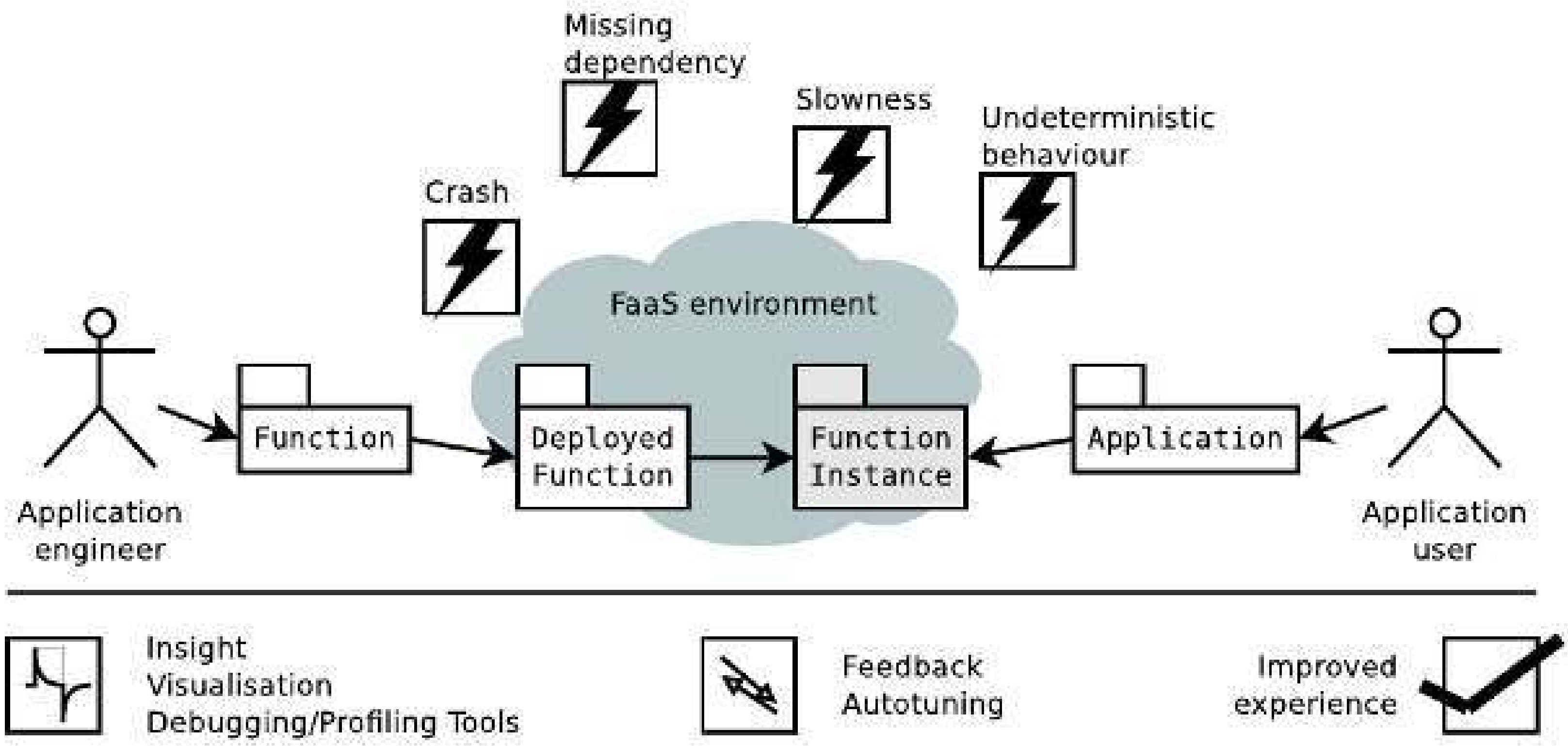
Challenge: FaaSification



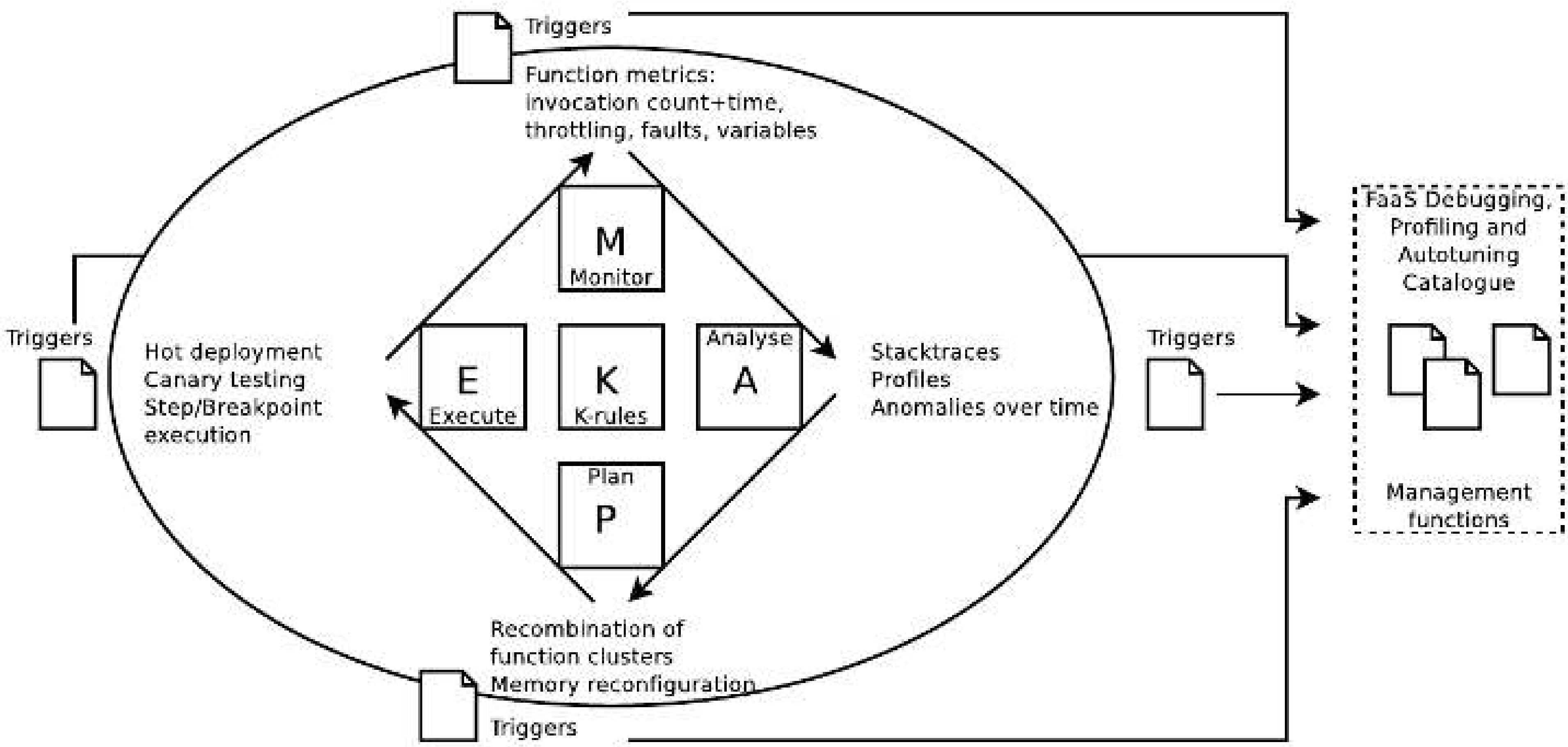
Challenge: Deep FaaSification



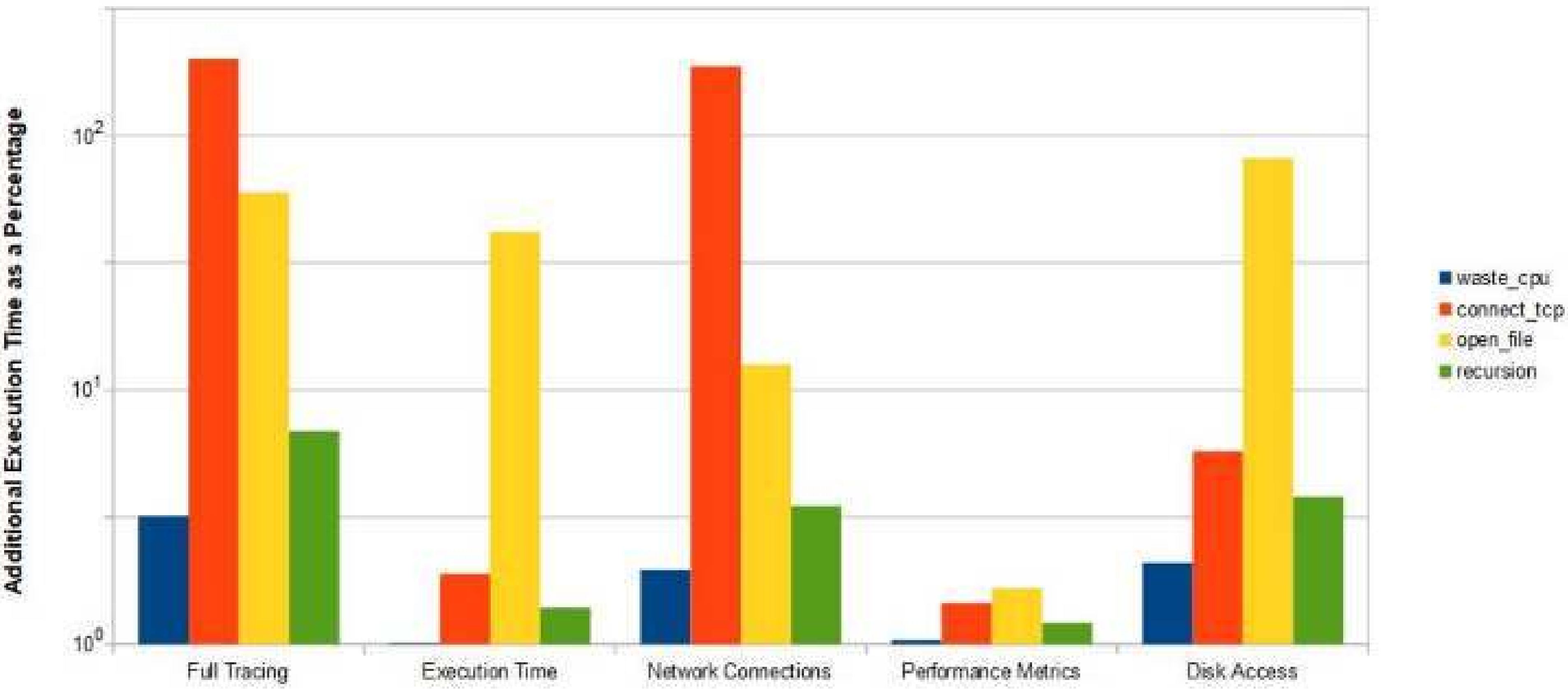
Challenge: Debugging



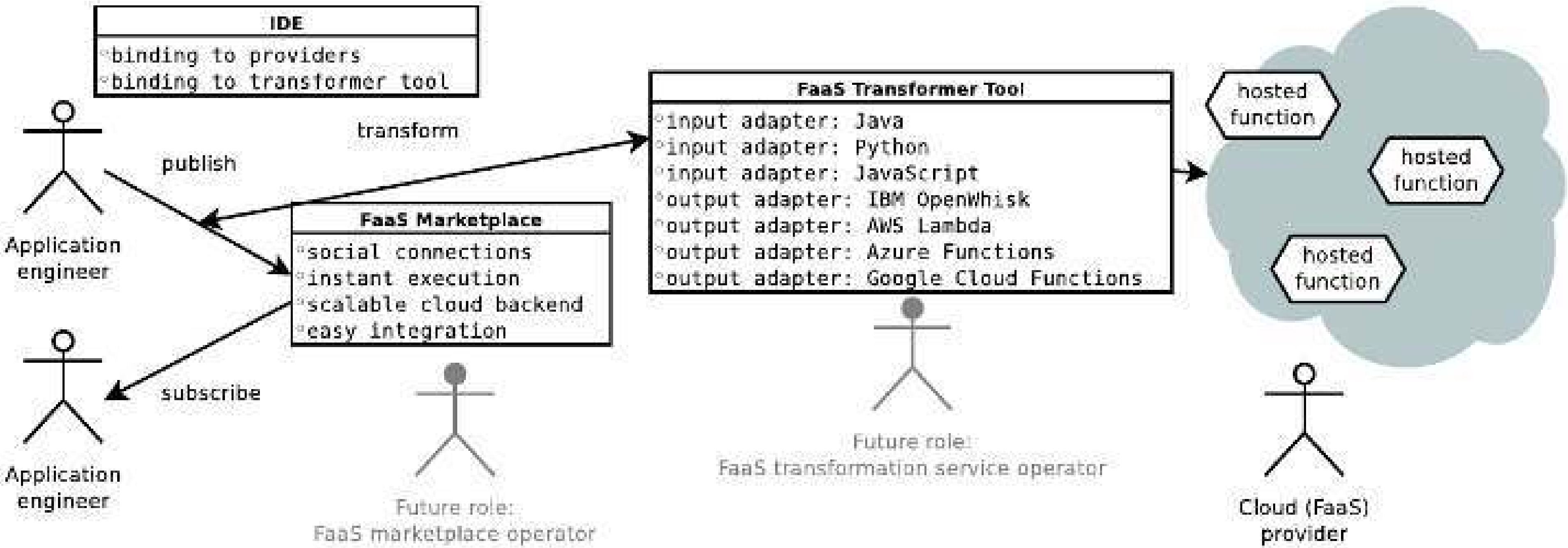
Challenge: Debugging



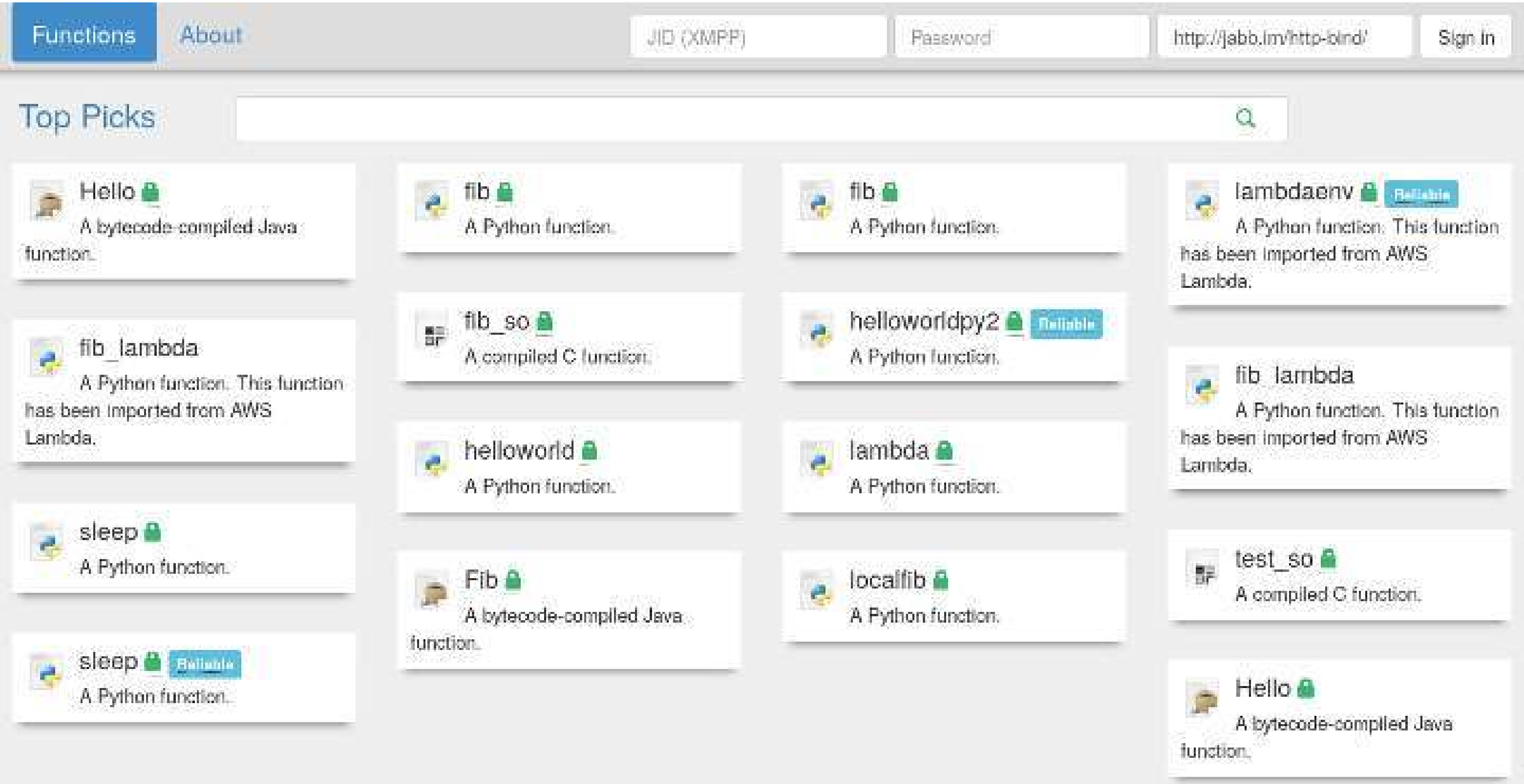
Challenge: Debugging



Challenge: Ecosystem



Challenge: Ecosystem



Wrap-Up Part IV

Discussion

Further Reading and FaaS Fun

Lama, Lambackup:
- https://arxiv.org/abs/1701.05945

Podilizer:
- https://arxiv.org/abs/1702.05510

Snafu:
- https://arxiv.org/abs/1703.07562

Lambada
- https://arxiv.org/abs/1705.08169

On arXiv Analytics:

On GitHub:

[github.com/serviceprototypinglab]

Service Engineering Research Area
ICCLAB
SPLAB