The Design, Productization, and Evaluation of a Serverless Workflow-Management System

Erwin van Eyk

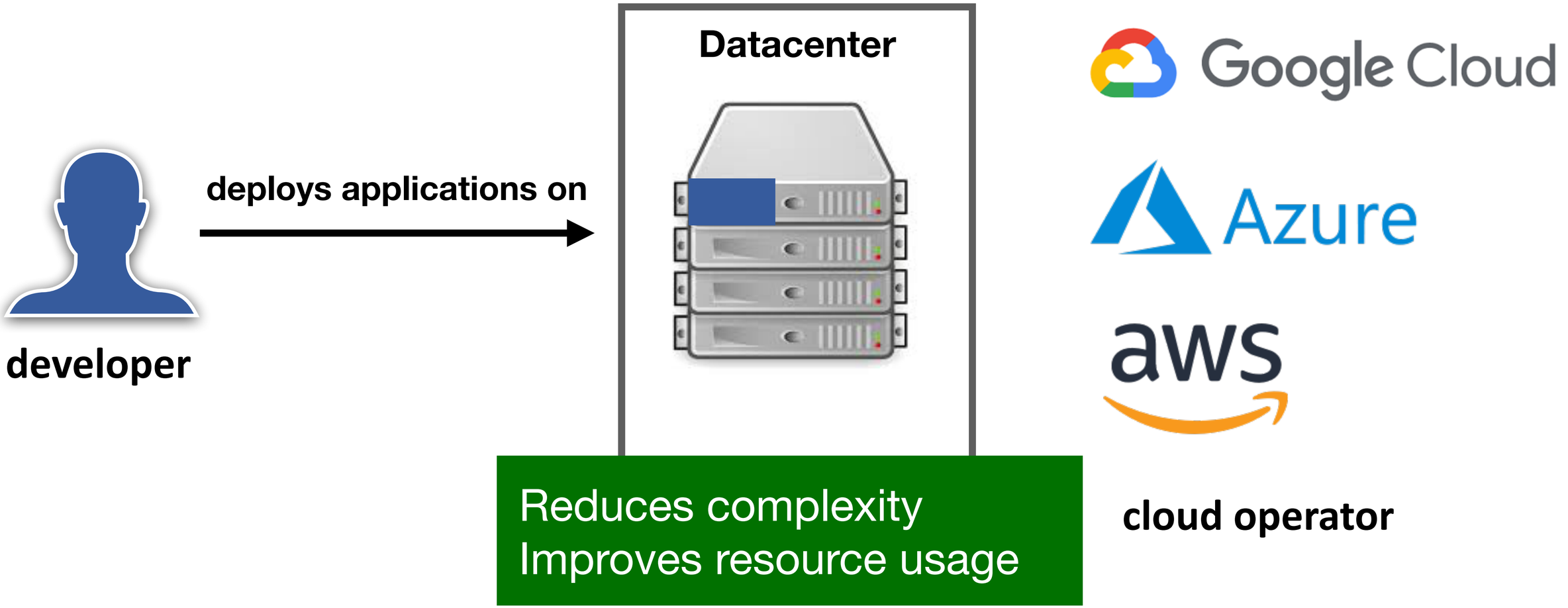
TU Delft

PLATFORM9

@Large Research
Massivizing Computer Systems

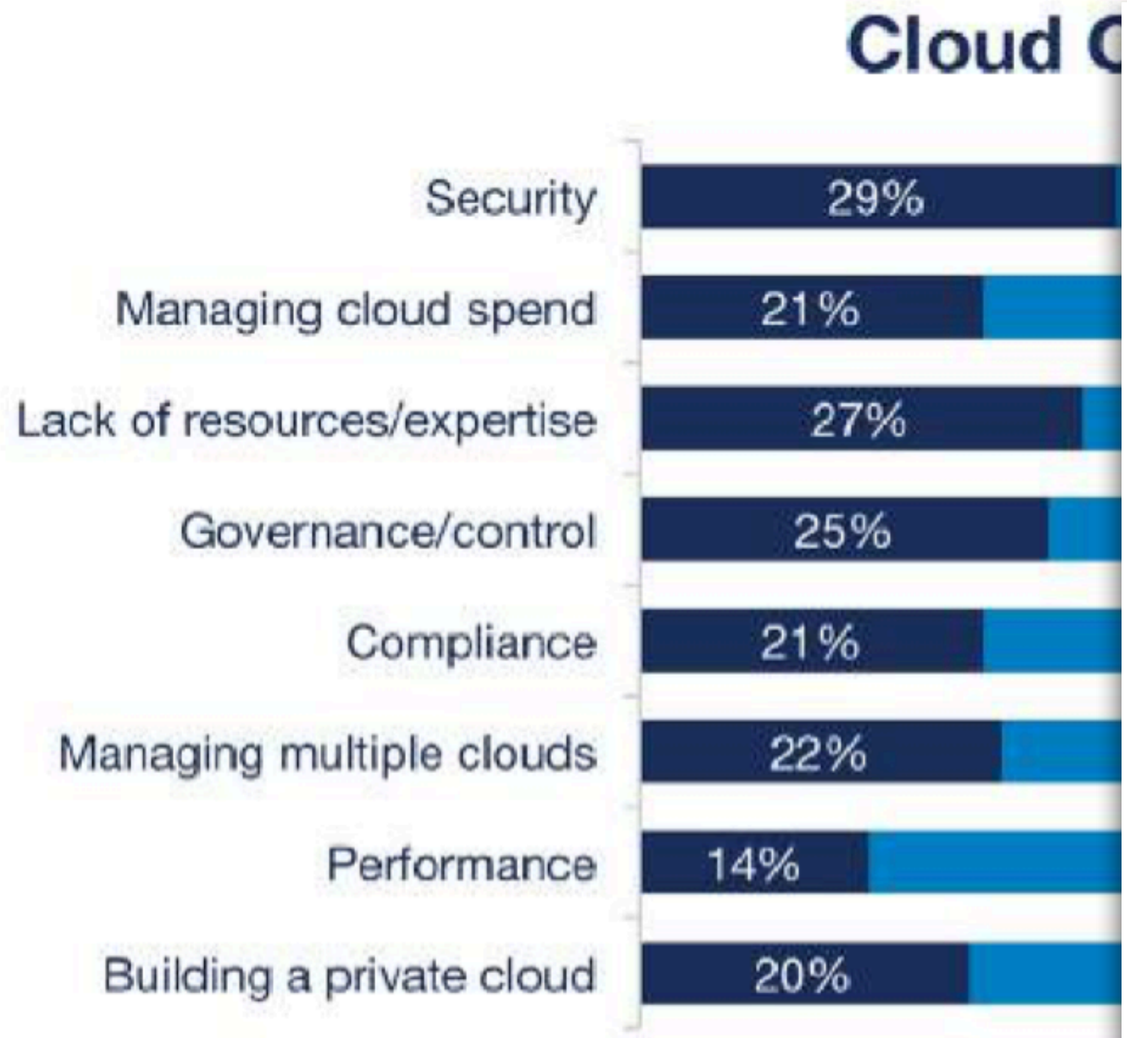
spec RG Cloud

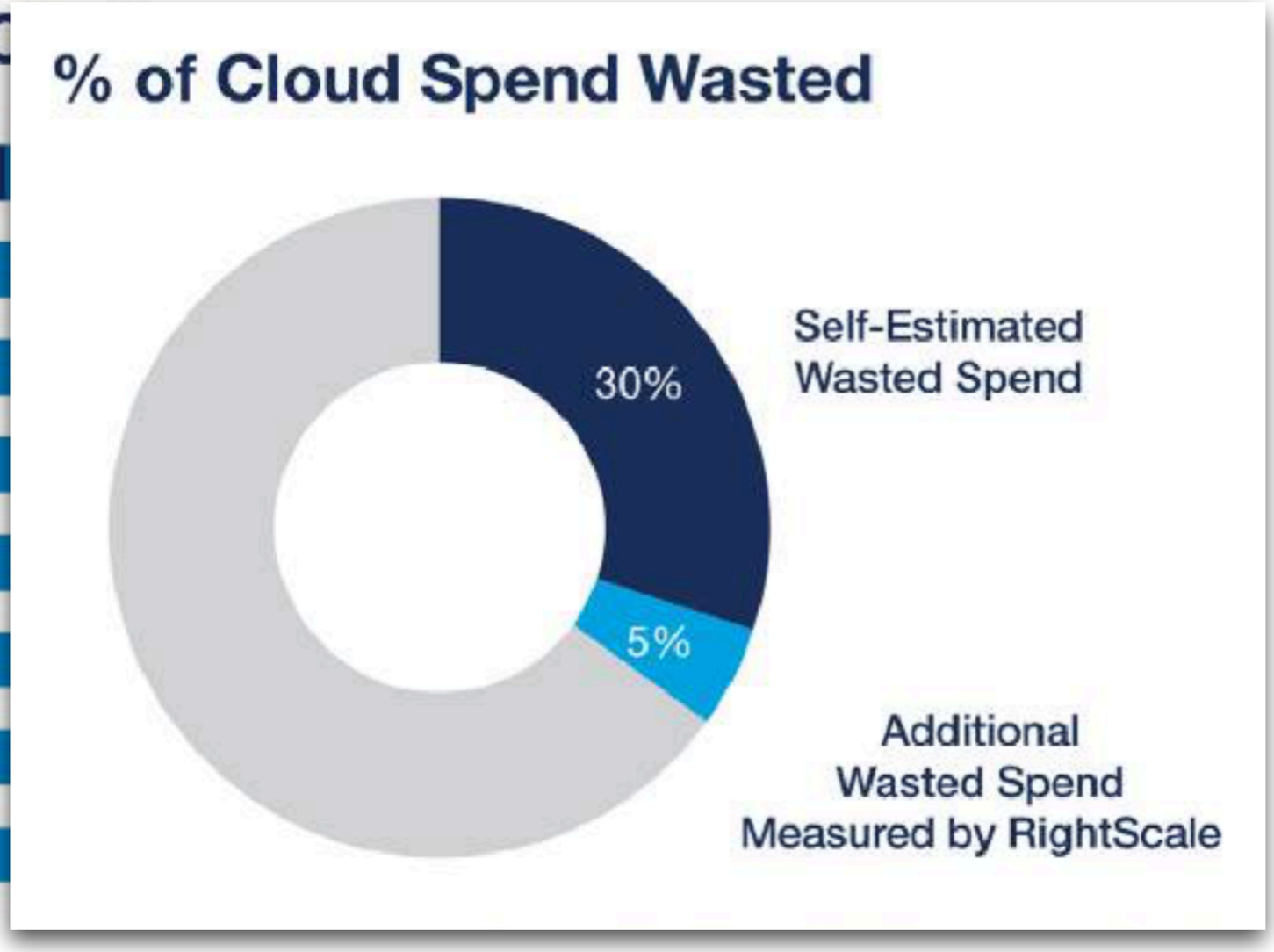
Cloud Computing



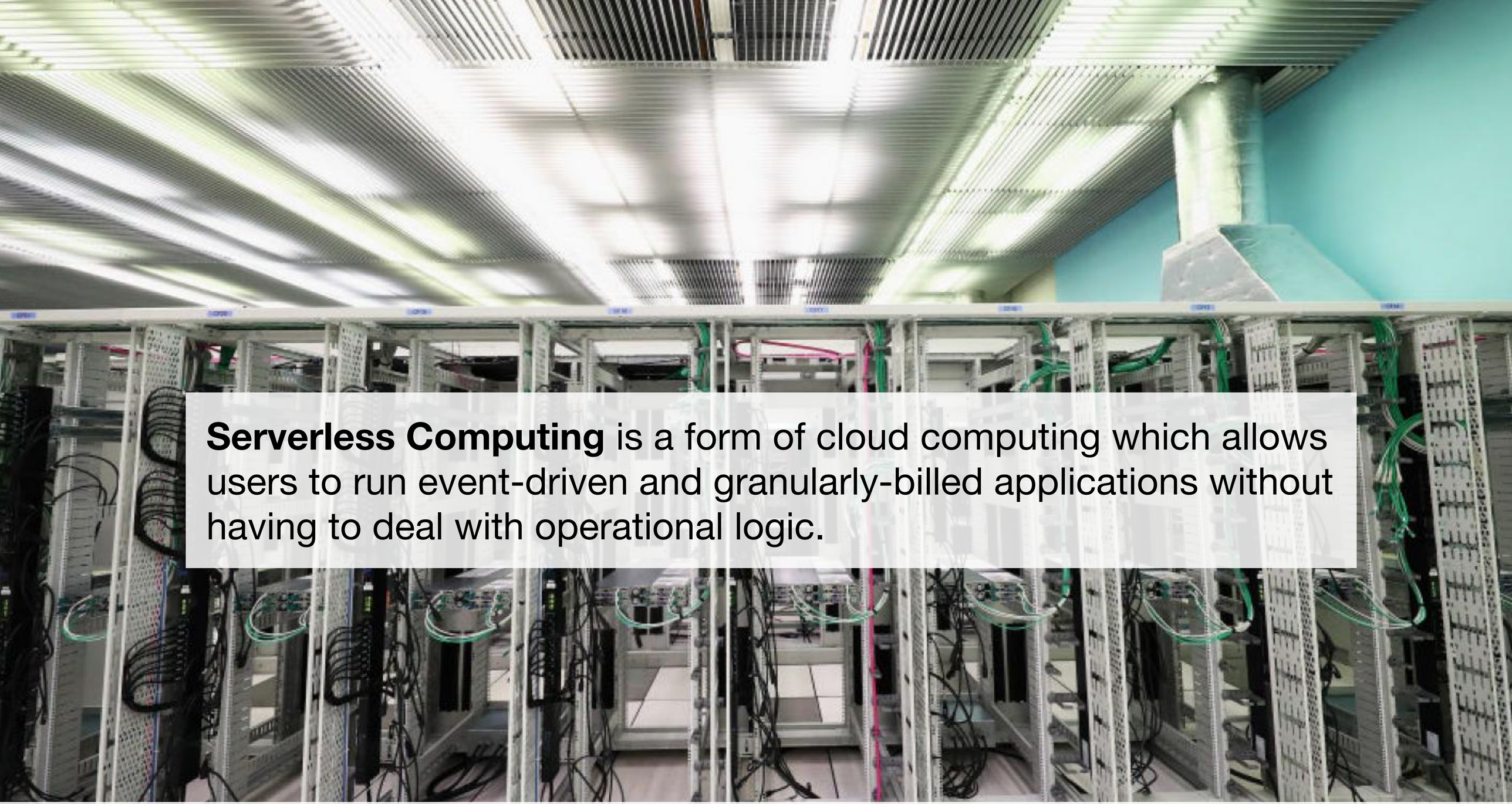
Van Eyk, Erwin, et al. "Serverless is more: From paas to present cloud computing." IEEE Internet Computing 22.5 (2018): 8-17.

Complexity and resource utilization remain key challenges





Source: RightScale 2018 State of the Cloud Report



Serverless Computing is a form of cloud computing which allows users to run event-driven and granularly-billed applications without having to deal with operational logic.

van Eyk, Erwin, et al. "The SPEC cloud group's research vision on FaaS and serverless architectures." WOSC, 2017.

Division of operational concerns

cloud operator manages…

developer manages…

DIY	Cloud / IaaS	serverless
Application	Application	Application
Application Middleware	Application Middleware	Application Middleware
Cluster Resource Mgt	Cluster Resource Mgt	Cluster Resource Mgt
Virtualization	Virtualization	Virtualization
Hardware	Hardware	Hardware

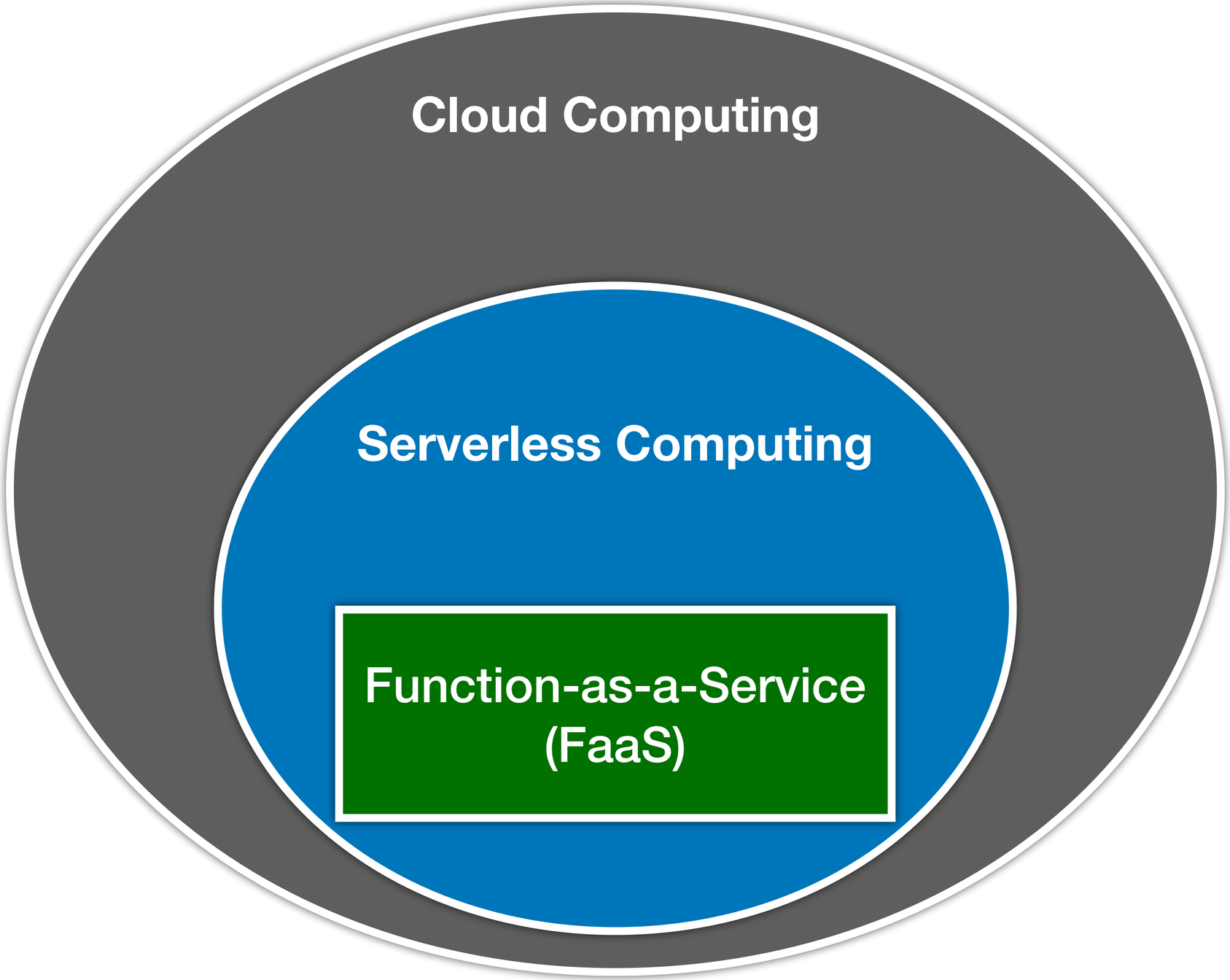
The “Serverless” market

currently: **$ 5** billion

by 2023: **$ 15** billion

(predicted)

source: https://www.marketsandmarkets.com/Market-Reports/serverless-architecture-market-64917099.html



van Eyk, Erwin, et al. "The SPEC cloud group's research vision on FaaS and serverless architectures." WOSC@MIDDLEWARE, 2017.

Function-as-a-Service (FaaS)

input

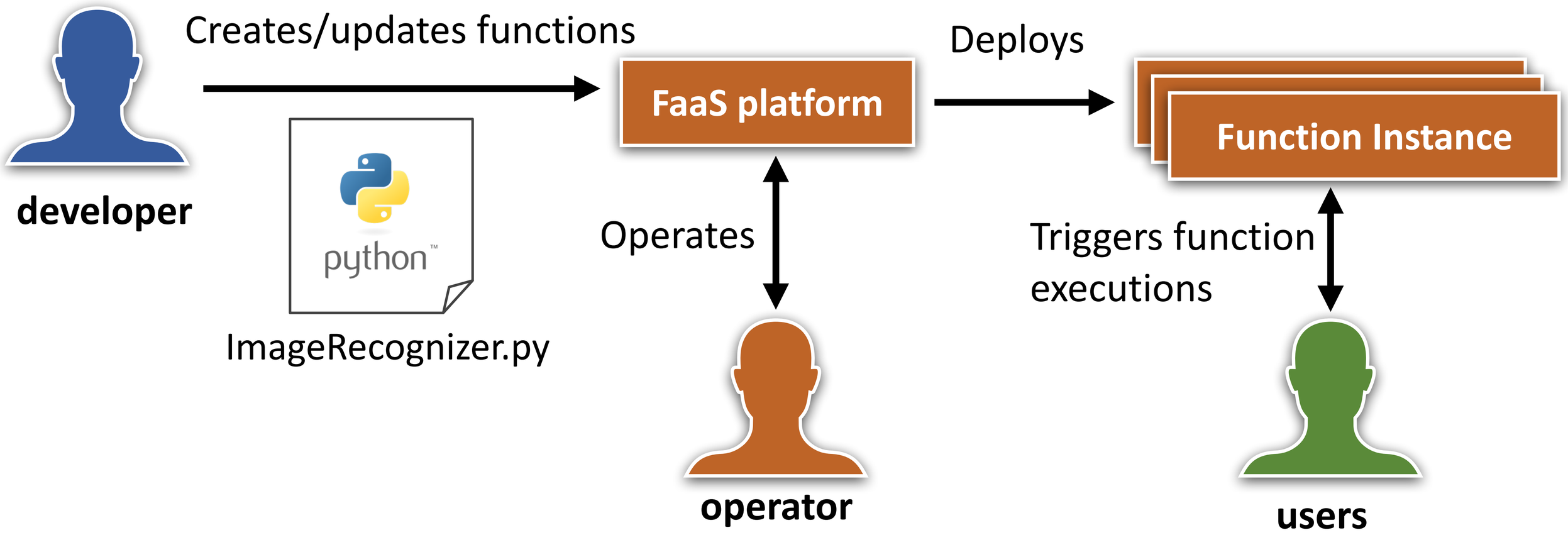
ImageRecognizer

output

cat

a serverless function

Function-as-a-Service (FaaS) in a nutshell



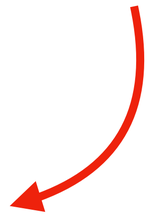
Van Eyk, Erwin, et al. "The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms." IEEE Internet Computing (2019, under submission).

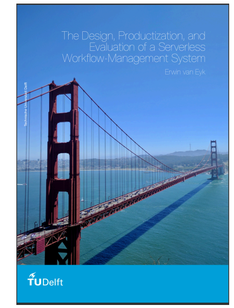
New technology, new problems

- Undefined terminology
- Lacking fundamental models and principles
- Absent real-world data and workload traces
- Missing well-established systems, tooling and processes

A multi-level approach

This thesis!





spec RG Cloud SM

serverless

@Large Research

Massivizing Computer Systems

and others

High-level, community problems

Terminology

Challenges & perspectives

FaaS reference architecture

Conceptual problems (in related domains)

Emergence of serverless

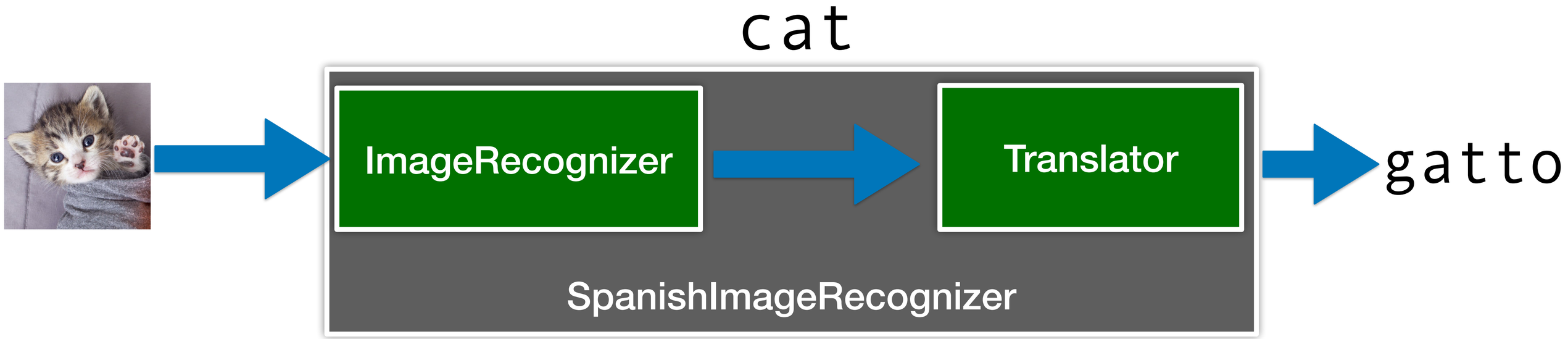
Scheduling in FaaS

A specific conceptual and technical problem

Serverless function composition

Serverless function composition

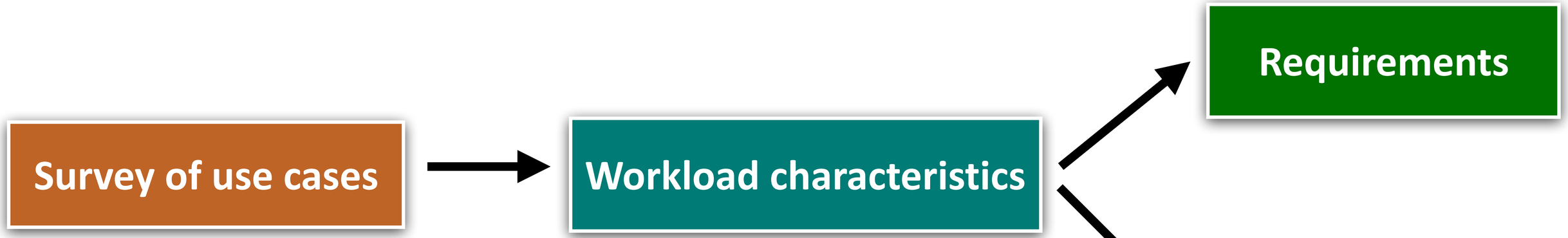
“Orchestrating existing functions into new more complex, composed functions.”



RQ: How to support serverless function composition through a distributed systems approach?

Requirements analysis

RQ1: What are the requirements to support serverless function compositions?



- Improves existing use cases
- Enables long-running processes

- Frequent executions
- Short runtimes (seconds)
- Long-tailed runtimes (cold starts!)

Requirements

1. Support long-running processes.
2. Minimize performance overhead.
3. Ensure reliable executions.
4. Scale to frequent workflow invocations.

Constraints

1. Treat FaaS functions as black-boxes.
2. Rely only on common FaaS functionality.
3. Follow the serverless function development and execution model.

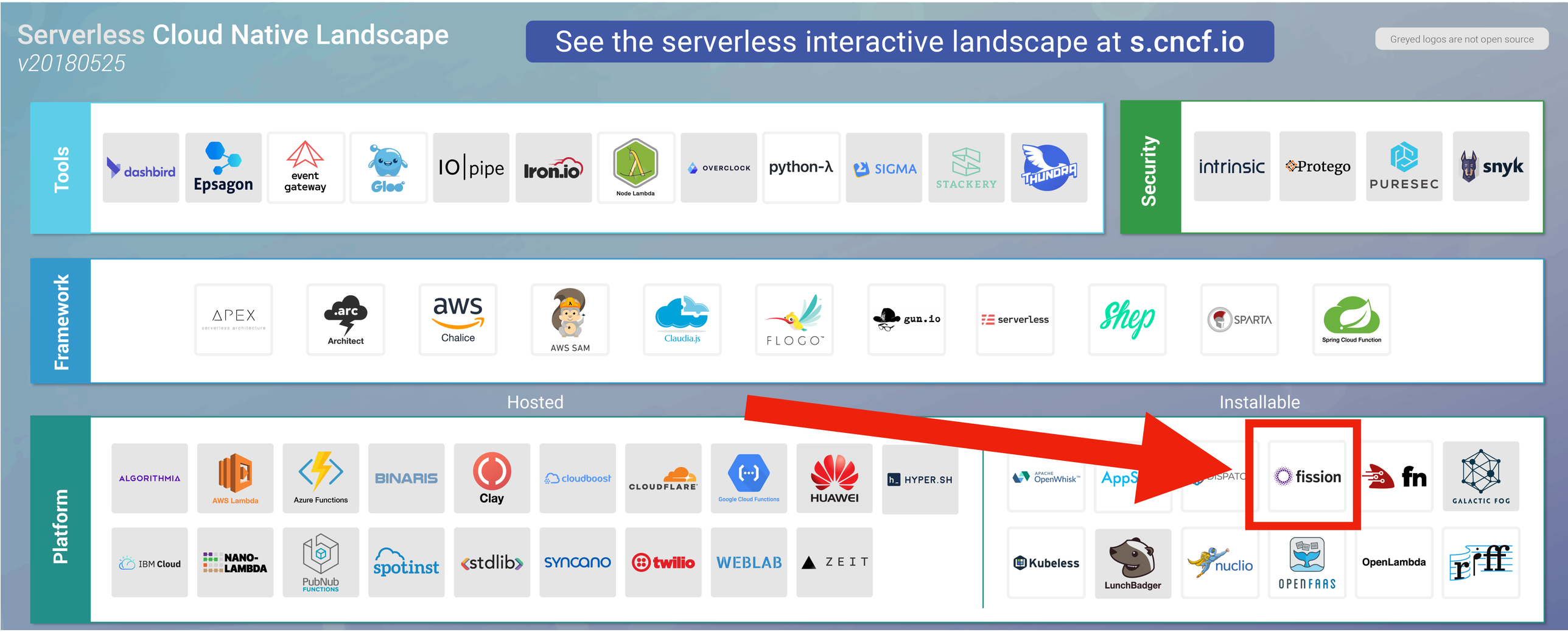
(Summarised; see thesis for the complete requirements and constraints)

Surveying the State-of-the-Art

RQ2: How do approaches for the composition of cloud functions compare?

1. FaaS platforms
2. Serverless function composition approaches
3. Workflow management systems
4. Workflow languages

(1) FaaS platform survey

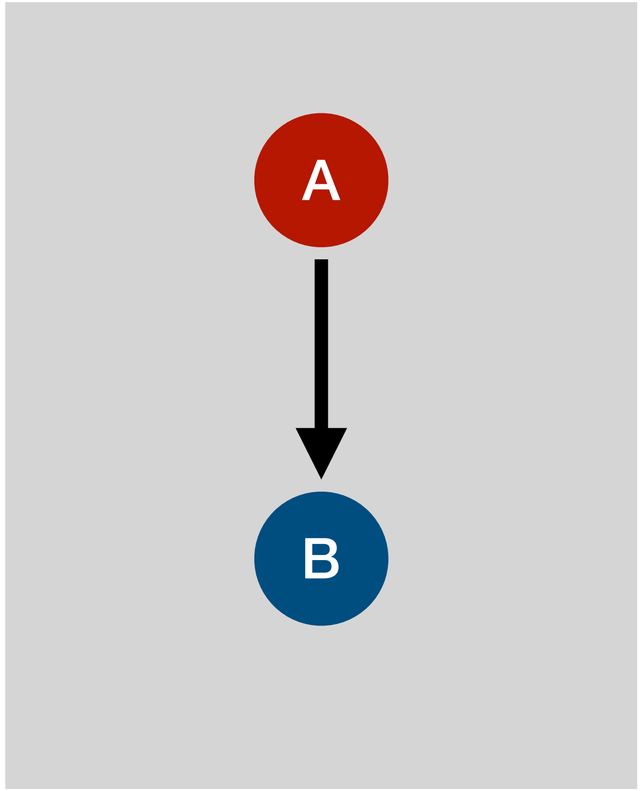


source: https://github.com/cncf/wg-serverless

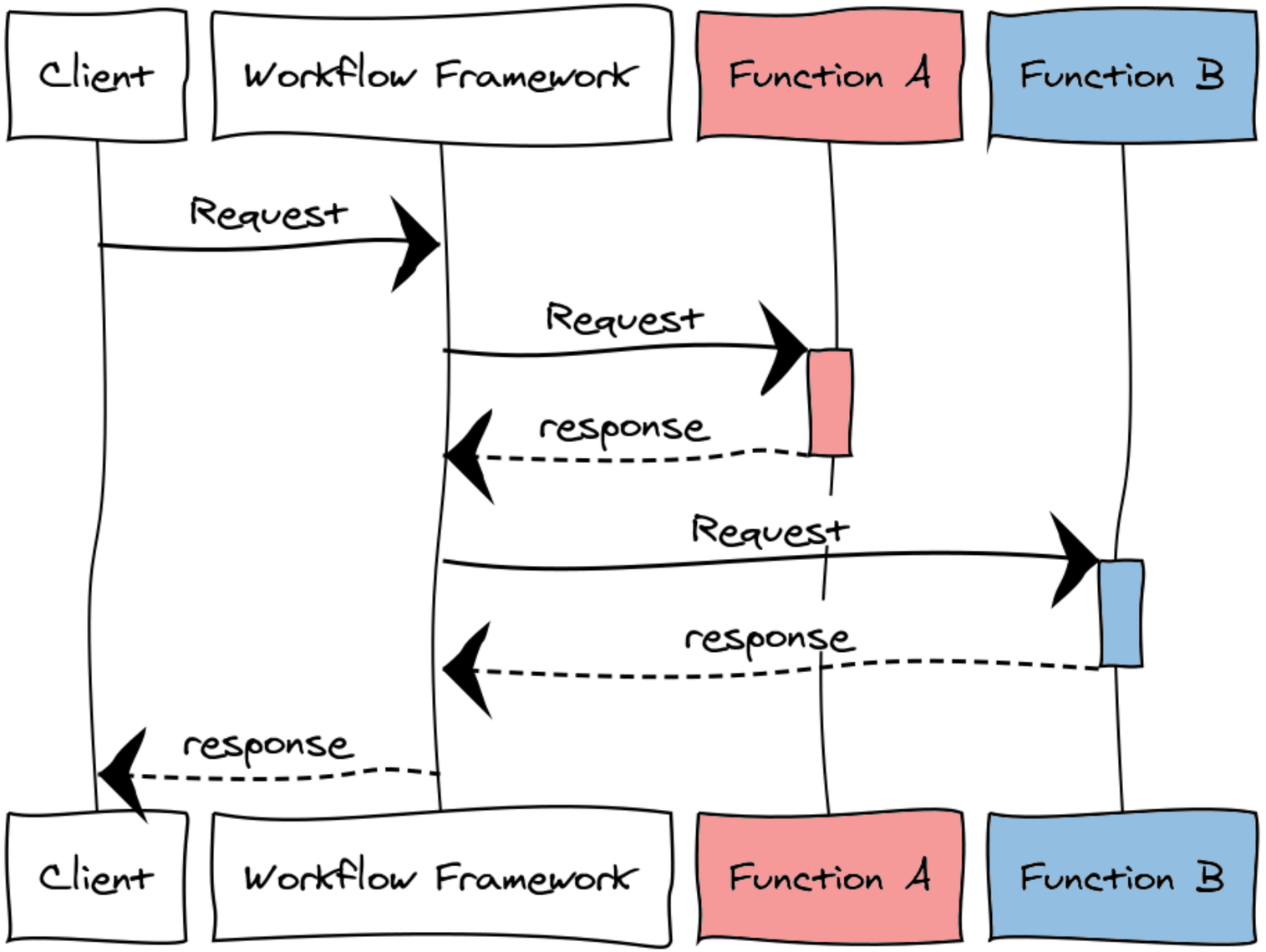
(2) Approaches to function composition survey

1. Direct composition
2. Compiled composition
3. Coordinator-based composition
4. Event-driven composition
5. Workflow-based composition

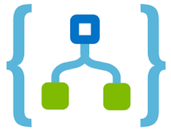
Workflow-based composition

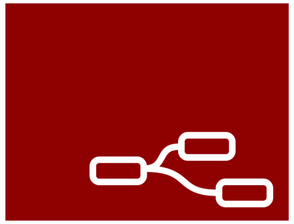


Workflow definition



(3+4) Survey of workflow systems and languages


Azure Logic Apps




AWS Step Functions

- Most serverless-like WMS are closed-source.
- Workflow languages are often tightly coupled to the WMS.
- Workflow languages contain system-level assumptions.
- Reuse common language syntax and system architecture.

Design of a serverless workflow management system

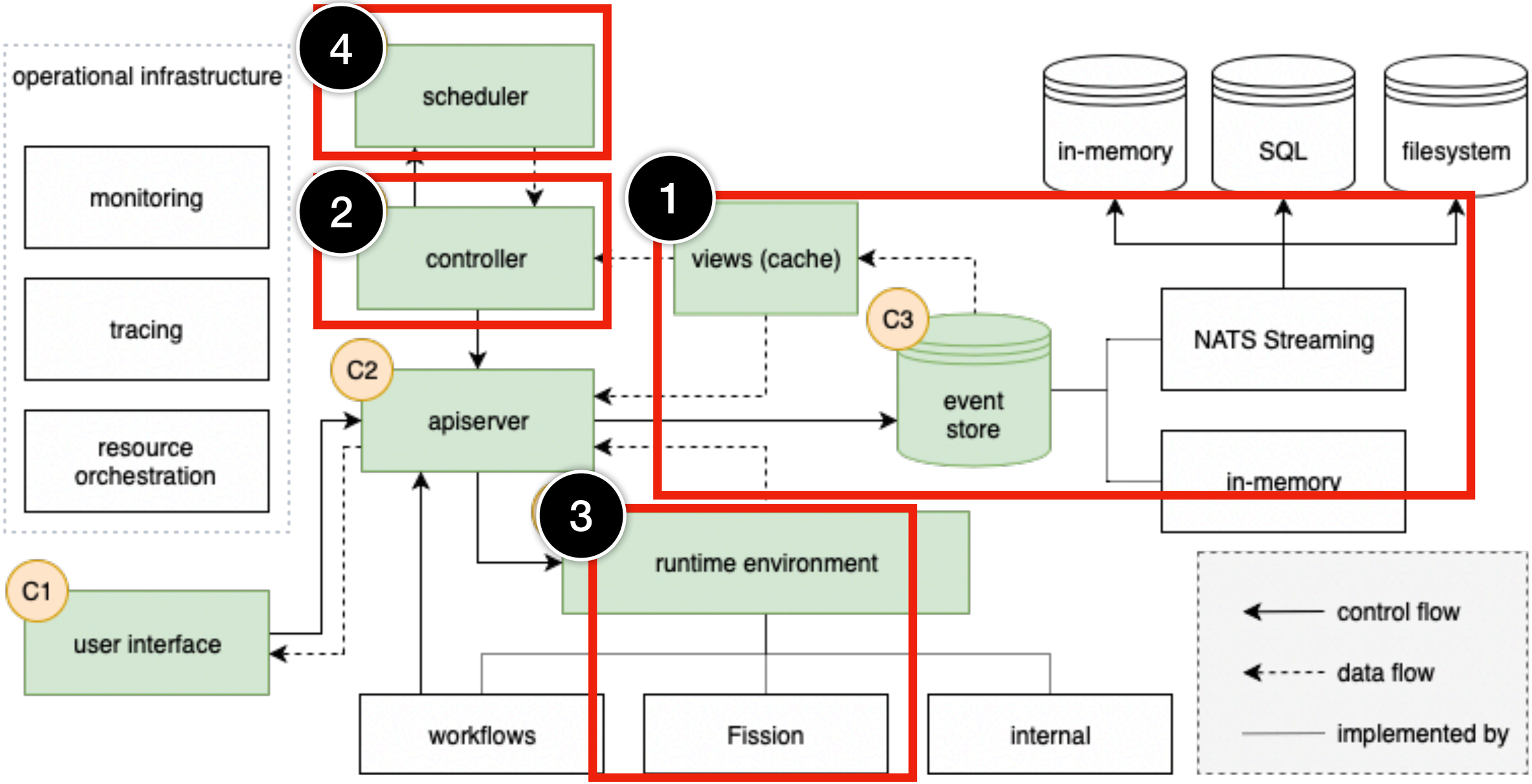
RQ3: How to design a system for composing cloud functions?

Serverless Workflow Language

(see thesis)

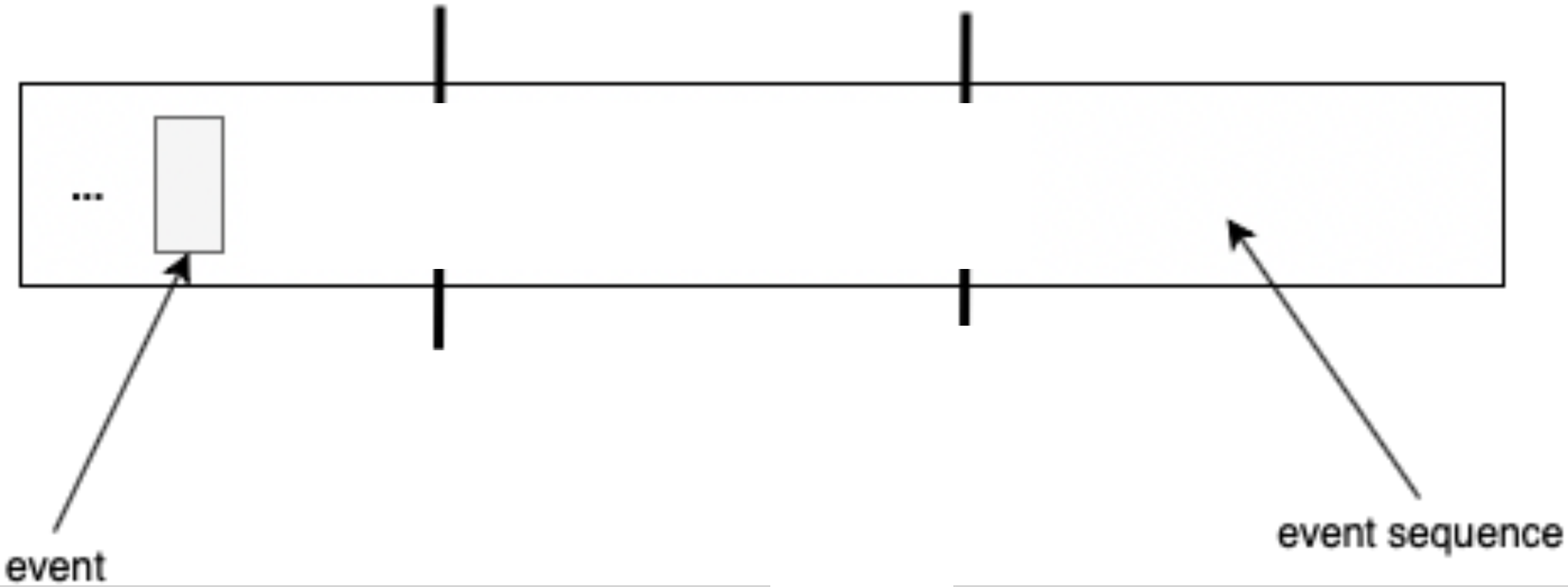
Serverless workflow management system

Fission Workflows



Event sourcing and persistence

The workflows are stored as a sequence of events



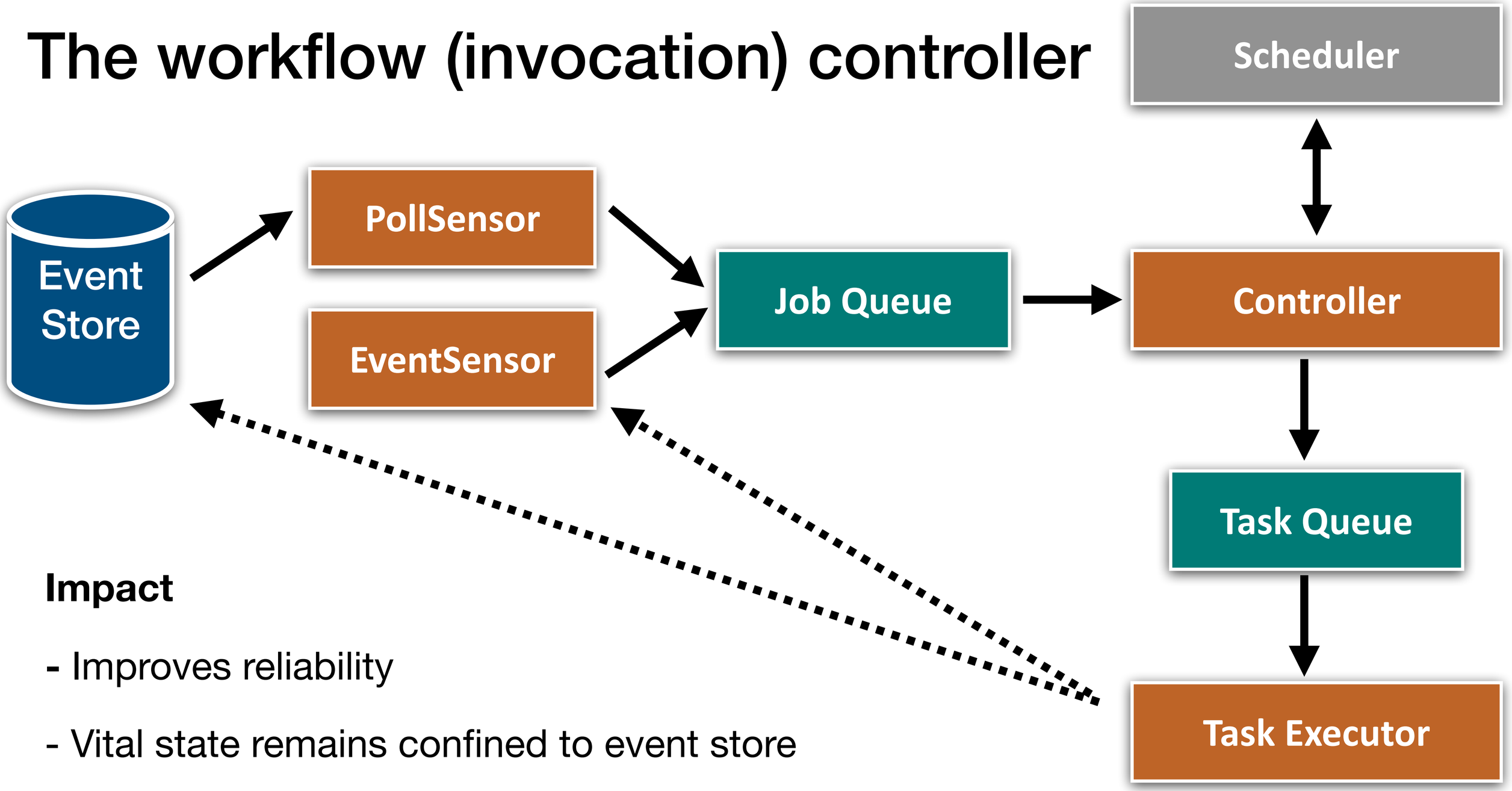
Impact

- Retain the time dimension of the data
- Simple, append-only persistence model

Event store implementations

- In-memory
- NATS streaming

The workflow (invocation) controller



Impact

- Improves reliability

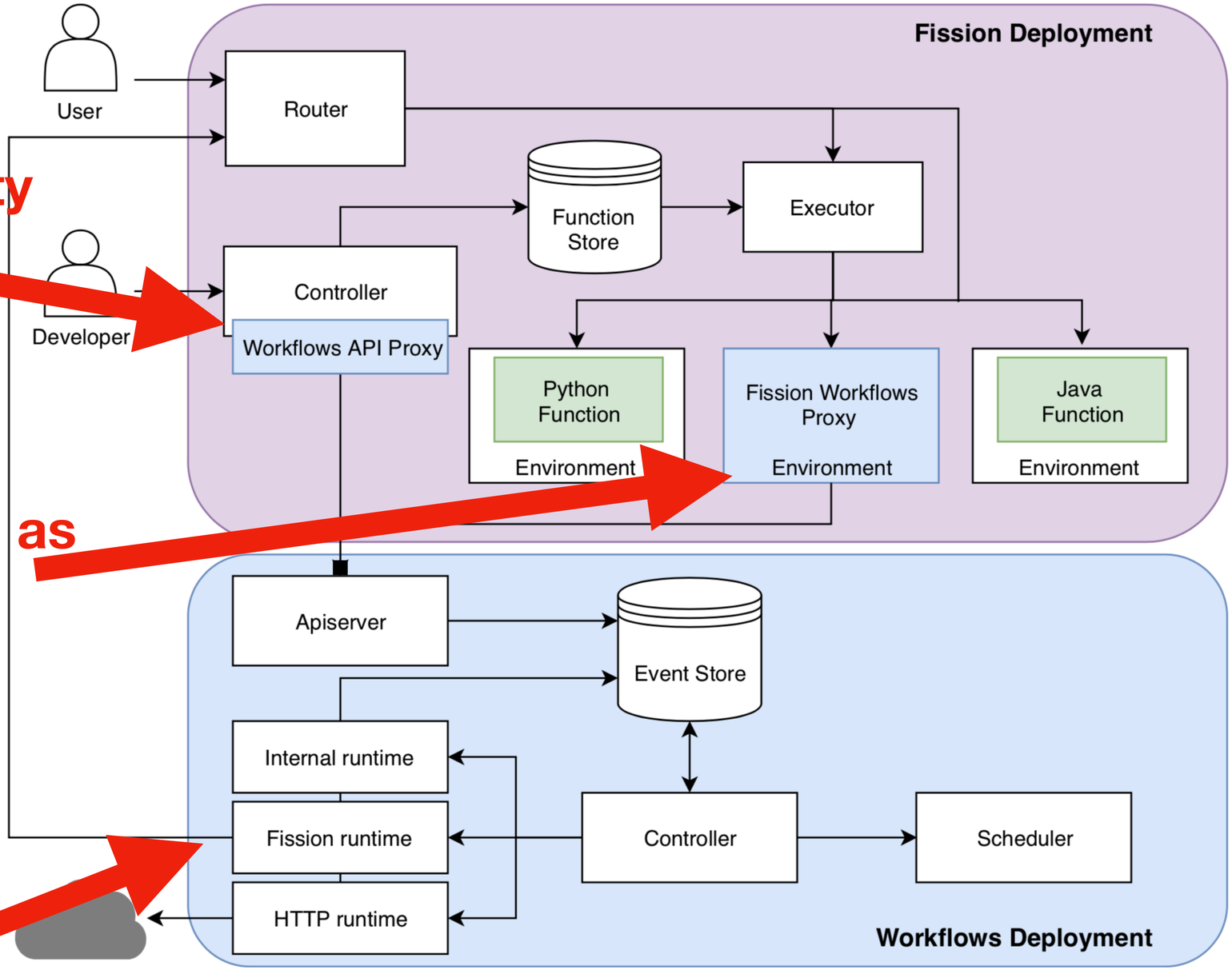
- Vital state remains confined to event store

Integration with the FaaS platform

FaaS can support workflow-specific functionality (optional)

Workflows can be executed as any serverless function

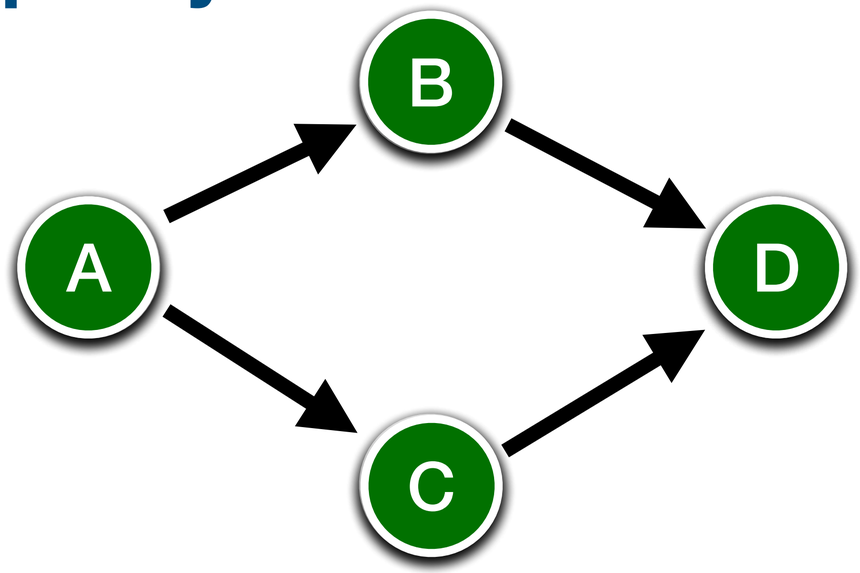
Fission Workflows uses the same user-level FaaS API

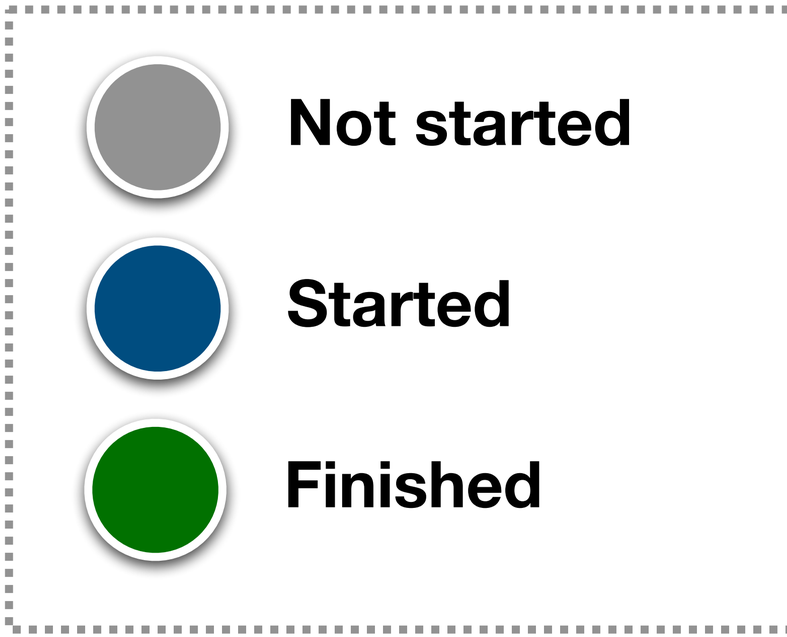


Workflow scheduling

Motivation: use workflows to mitigate cold starts of serverless functions.

horizon policy

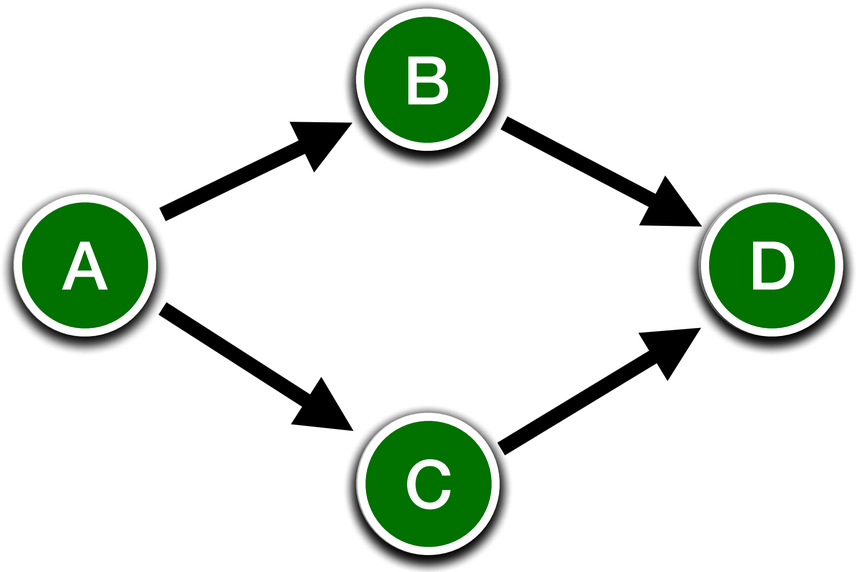




Prewarming: predictively deploying serverless functions based on expected demand.

Workflow scheduling

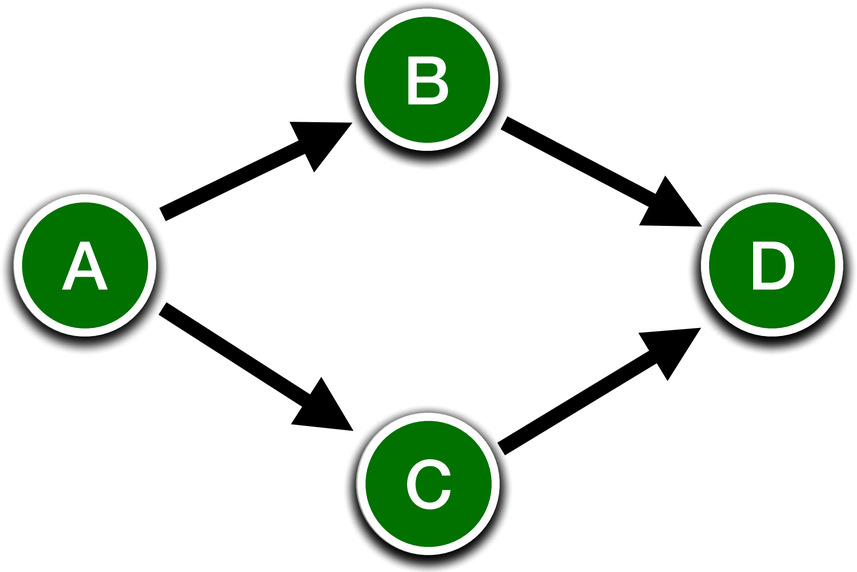
prewarm-all policy



Prewarm all non-active tasks in the workflow

Workflow scheduling

prewarm-horizon policy



Prewarm all tasks that are “up next” in the workflow

Experimental evaluation of the prototype

RQ4: How to evaluate systems for function composition experimentally?

Goal: Evaluate the workflow system prototype based on the following metrics:

- Reliability

- Performance

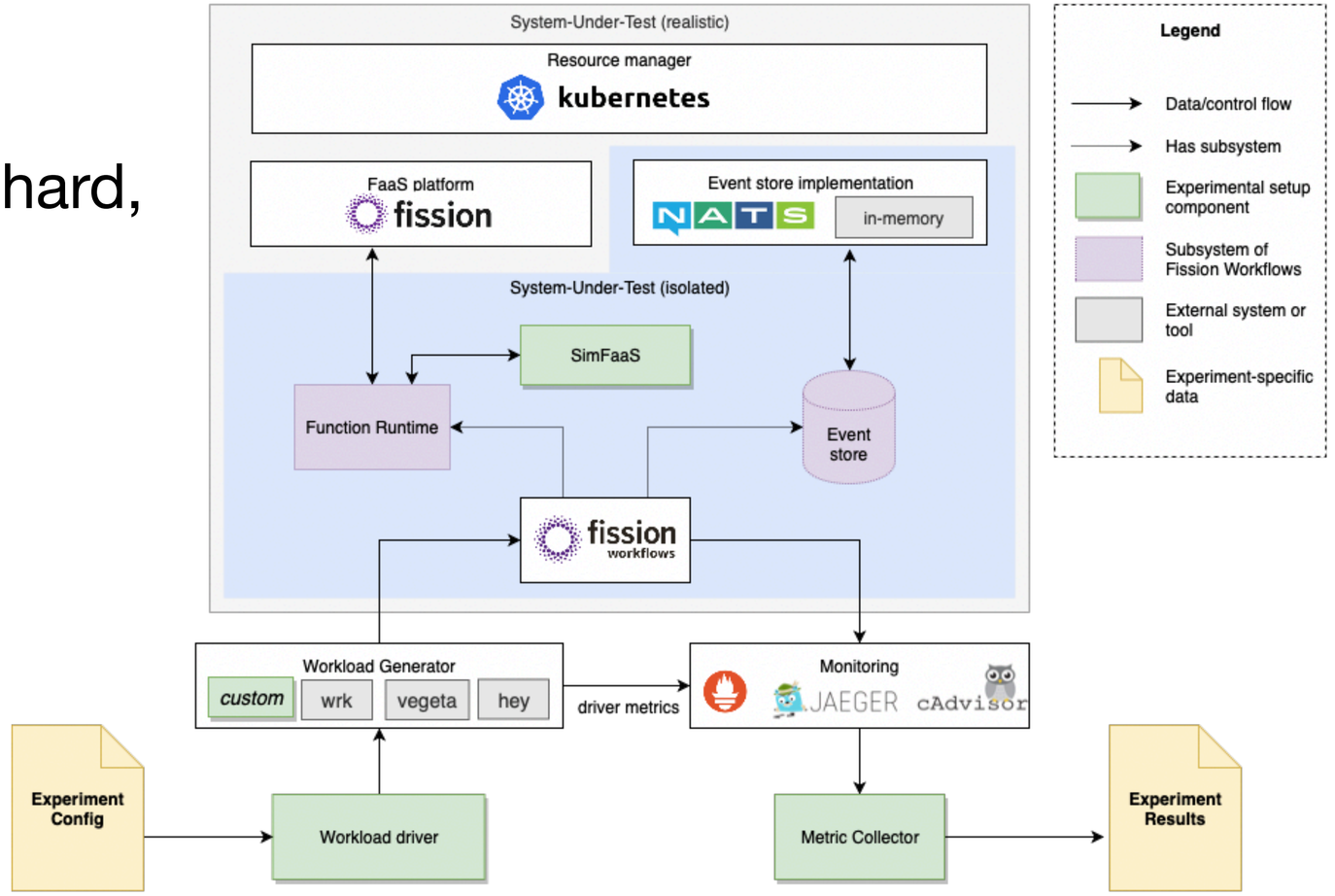
- Cost

Experimental setup

Developing distributed systems is hard, testing them is even harder.

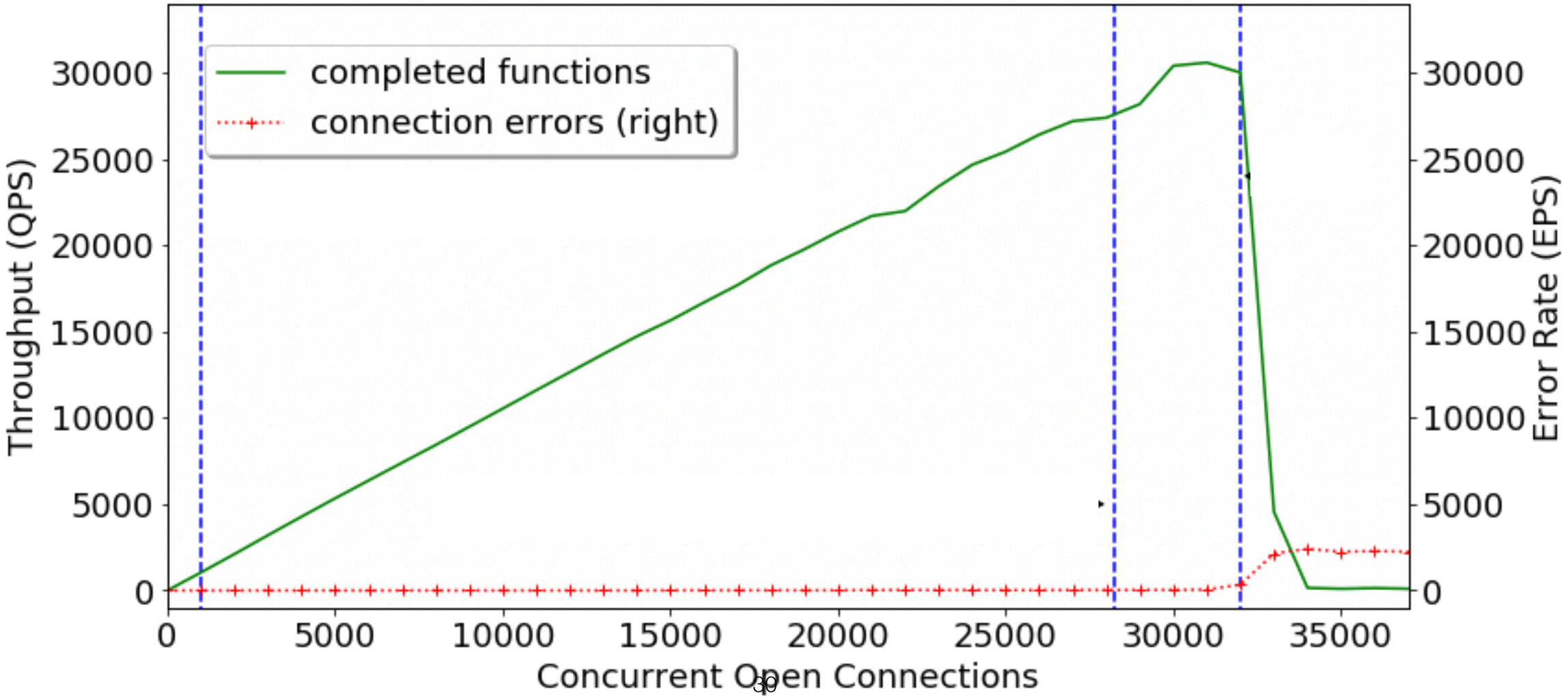
Two types of experiments

- **Isolated** experiments
- **Realistic** experiments

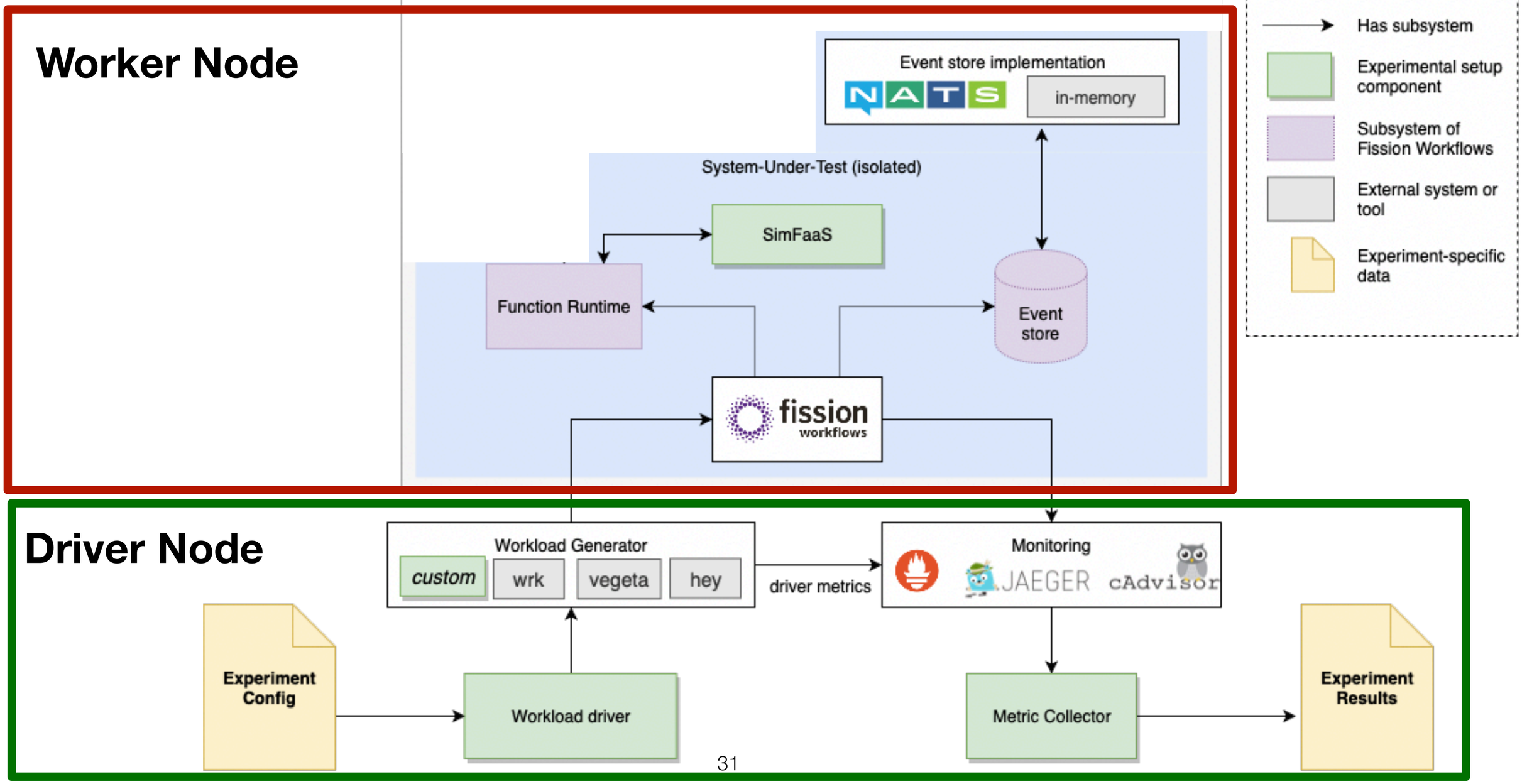


SimFaaS: a FaaS emulator

Goal: versatile FaaS emulation, which scales (with negligible performance overhead) well beyond the workloads in the other experiments.



Isolated experiment setup



Overview of the isolated experiments

Fault tolerance experiments

- Unavailable event store
- Unavailable FaaS runtime
- **Fail-stop crashes of Fission Workflows with different configurations**

Scalability experiments

- Event store implementation performance overhead
- Workflow submission
- Workflow throughput
- Workflow parallelism
- Workflow length

Scheduling experiments

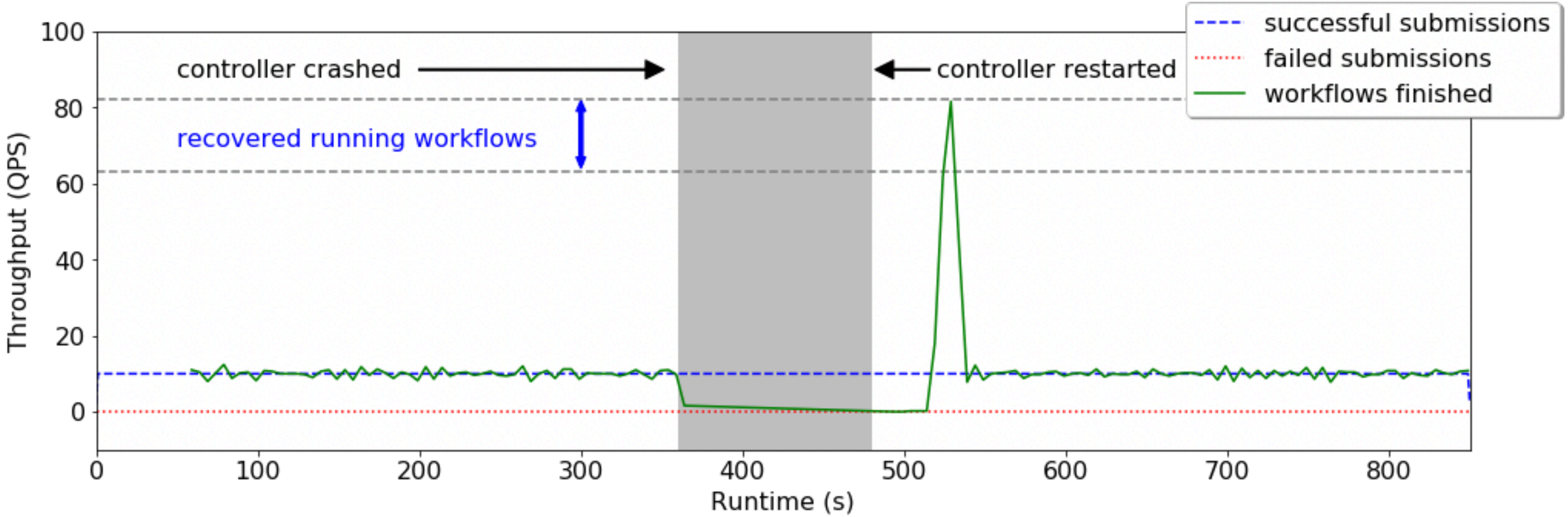
Fault-tolerance experiments

Goal: Evaluate the fault-tolerance of the prototype under different failure scenarios.

Scenarios evaluated:

- Unavailable event store
- Unavailable FaaS runtime
- **Fail-stop crashes of Fission Workflows**

Fault-tolerance: crash of the controller



Scheduling experiments

Goal: Evaluate the performance and resource consumption of the (prewarm-focused) scheduling policies:

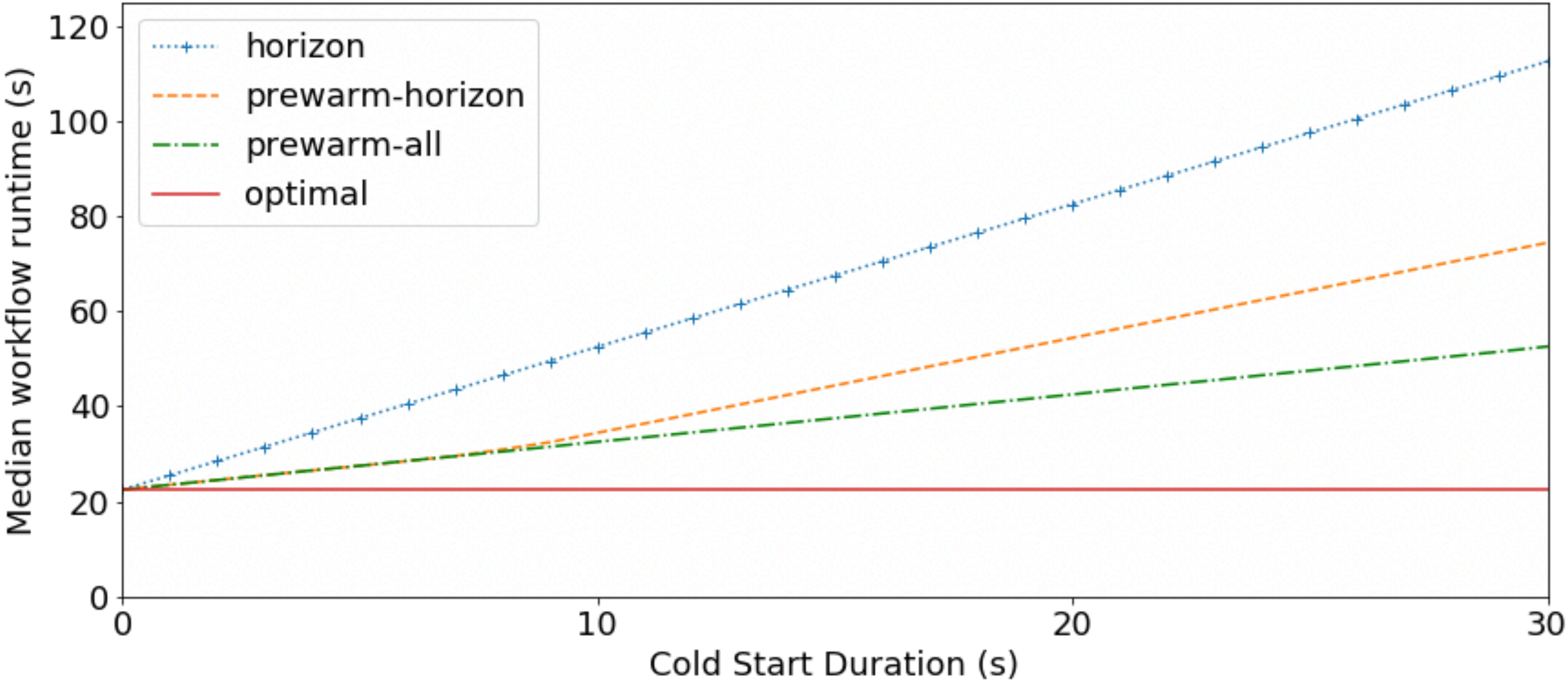
- **horizon**

- **prewarm-horizon**

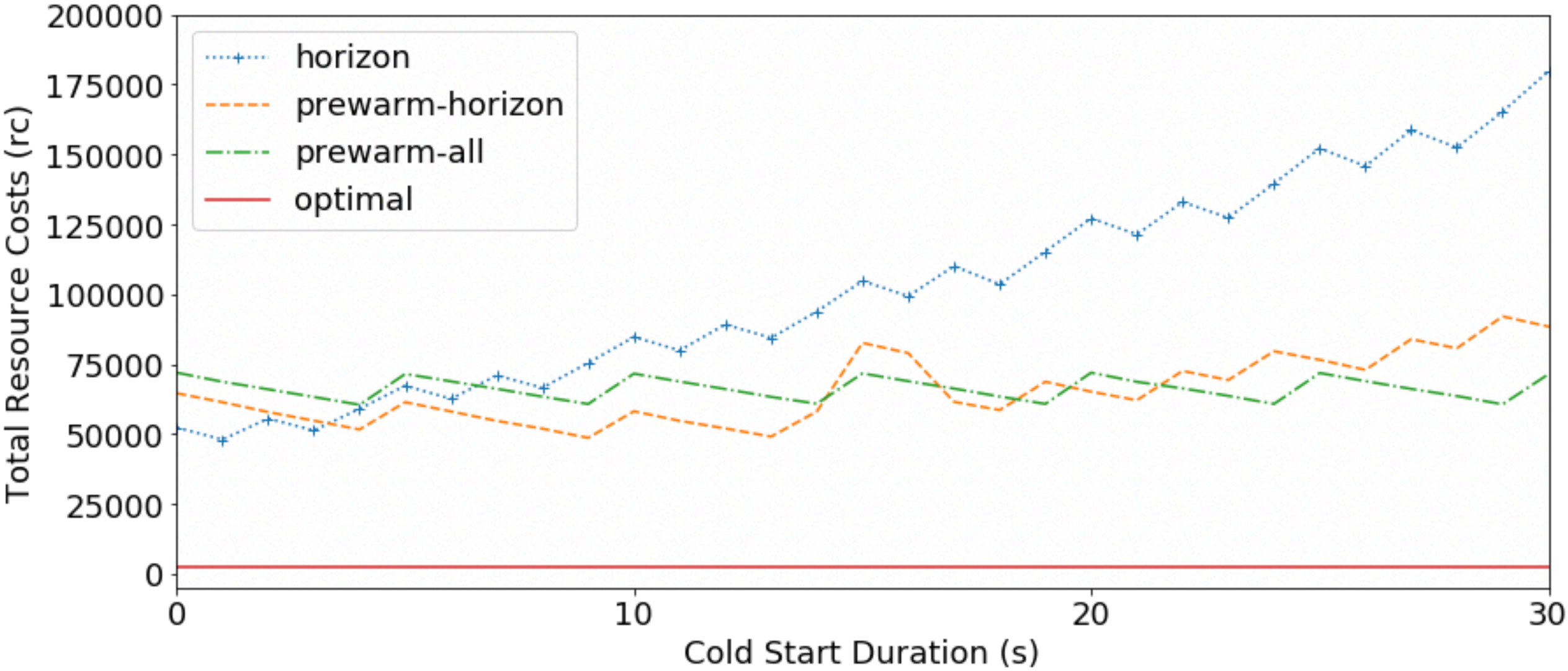
- **prewarm-all**

Approach: Use SimFaaS to emulate and control cold starts and resource usage.

Performance of scheduling policies



Resource consumption of scheduling policies

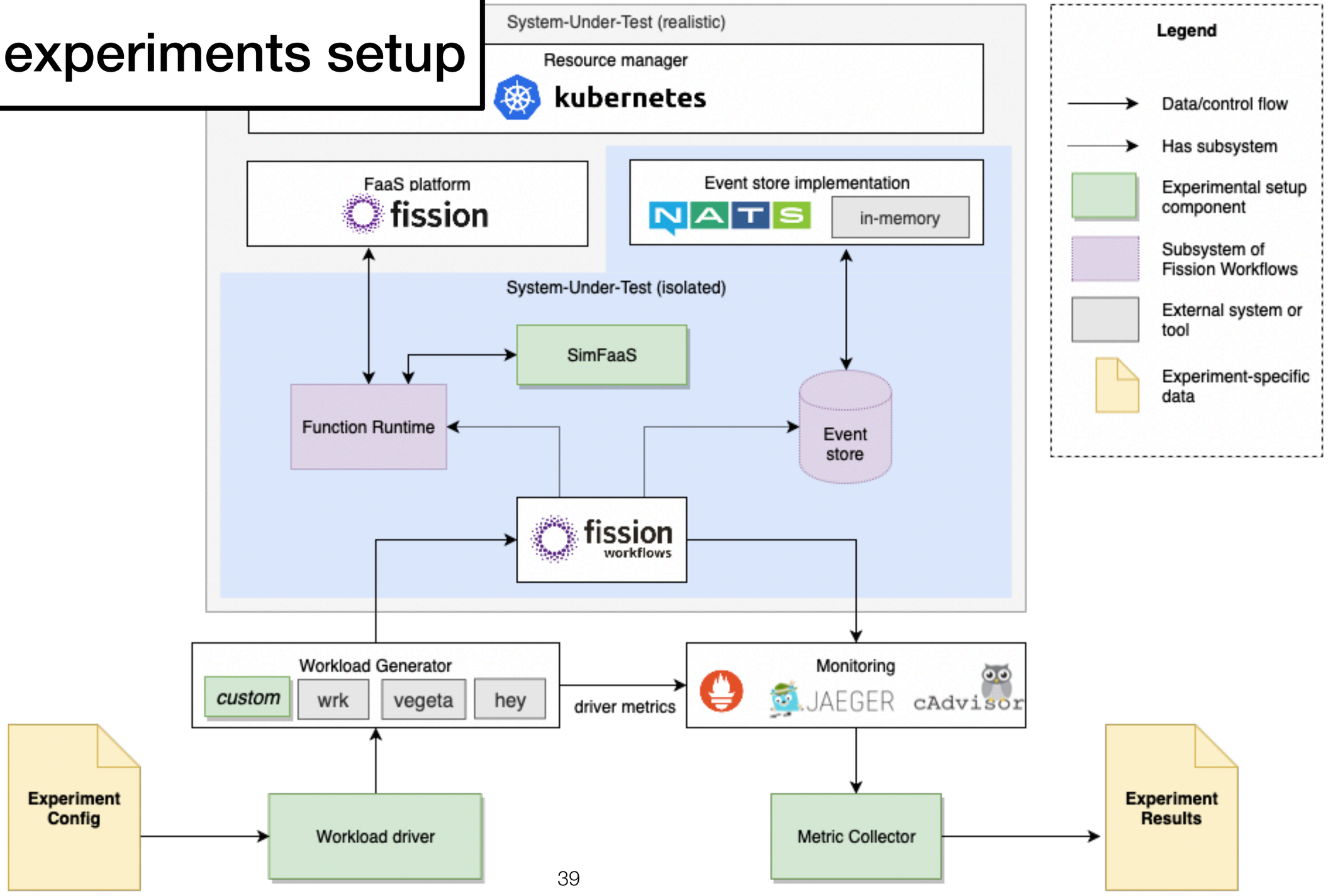


Realistic experiments

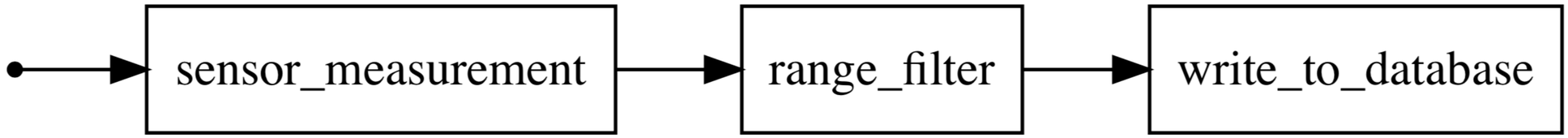
Goal: evaluate the performance and cost of Fission Workflows under realistic circumstances, comparing it to the state-of-the-art.

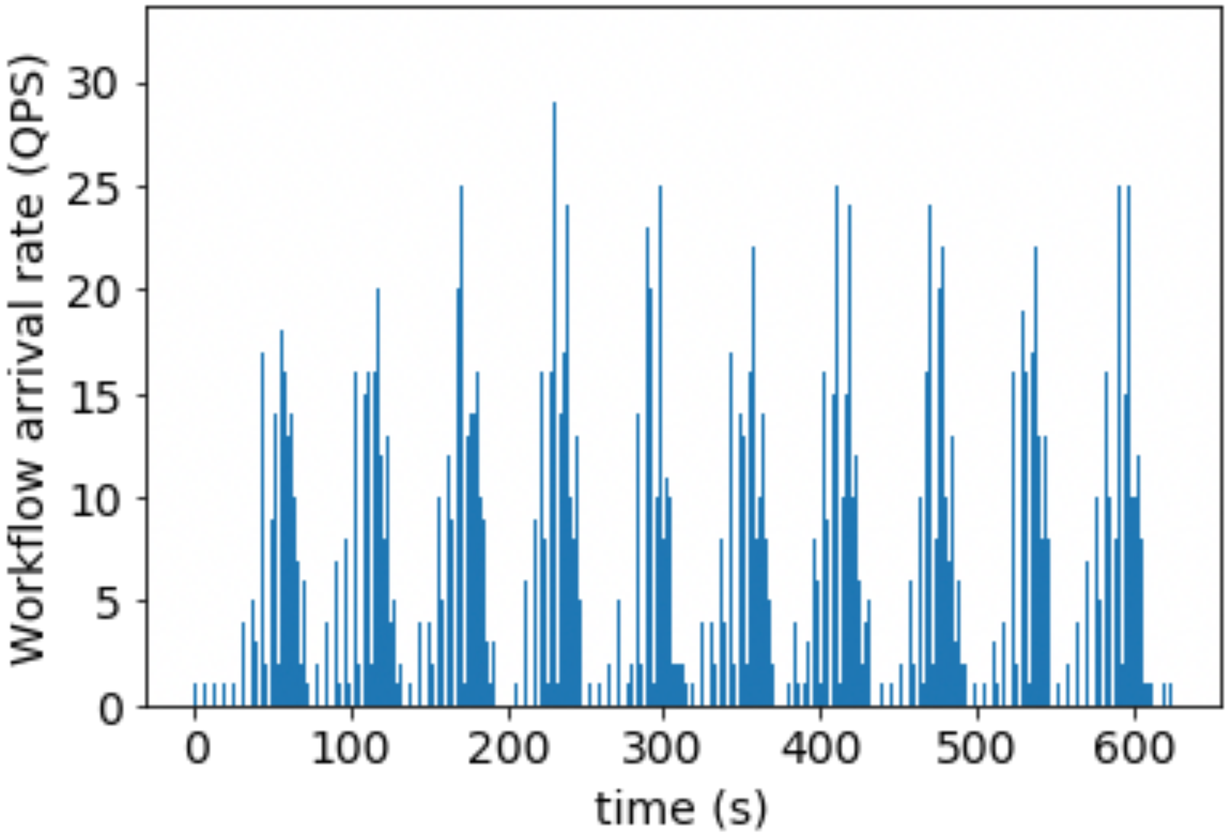
Provider	Workflow system	FaaS platform
Google Cloud	Google Cloud Composer	Google Cloud Functions
Microsoft Azure	Azure Logic Apps	Azure Functions
AWS	AWS Step Functions	AWS Lambda
Fission (Workflows)	fission workflows	fission

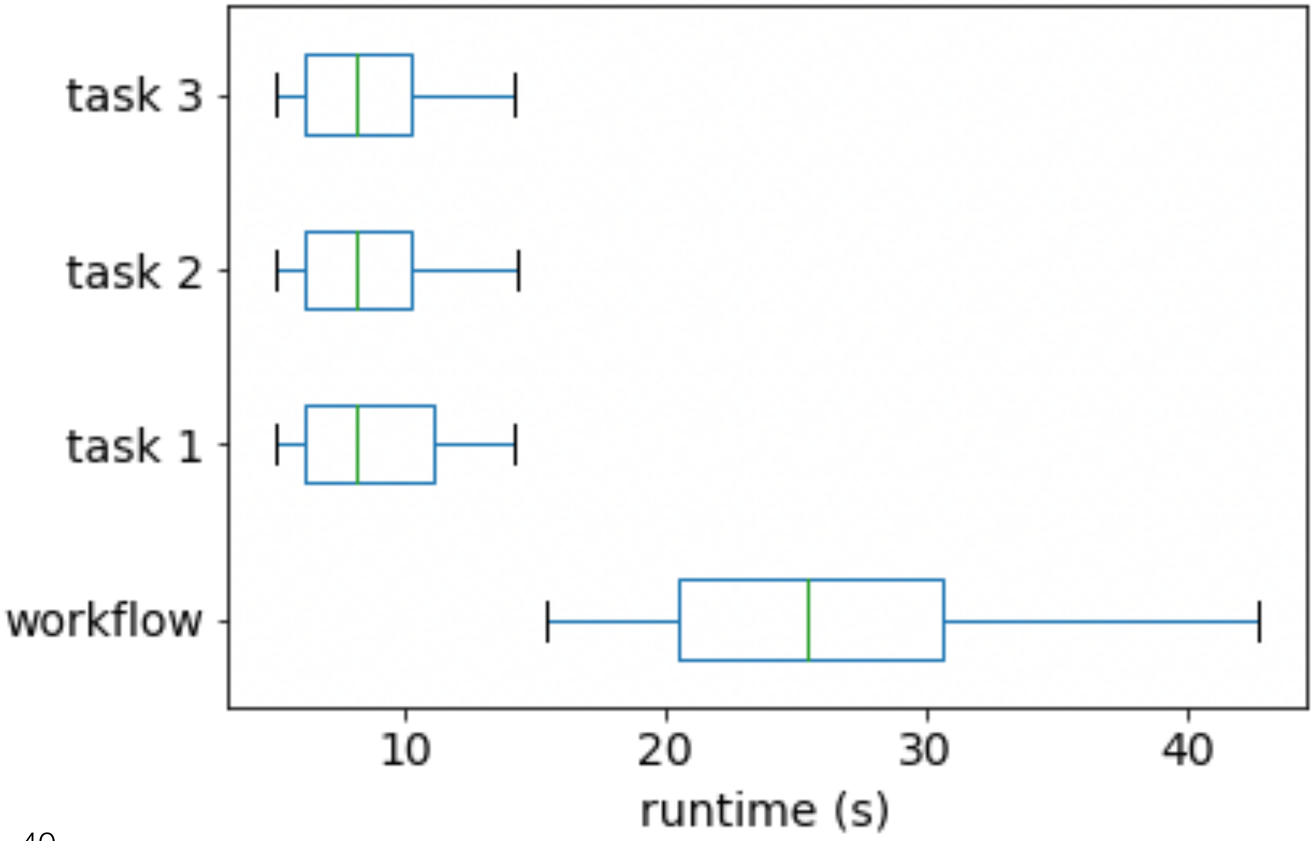
Realistic experiments setup



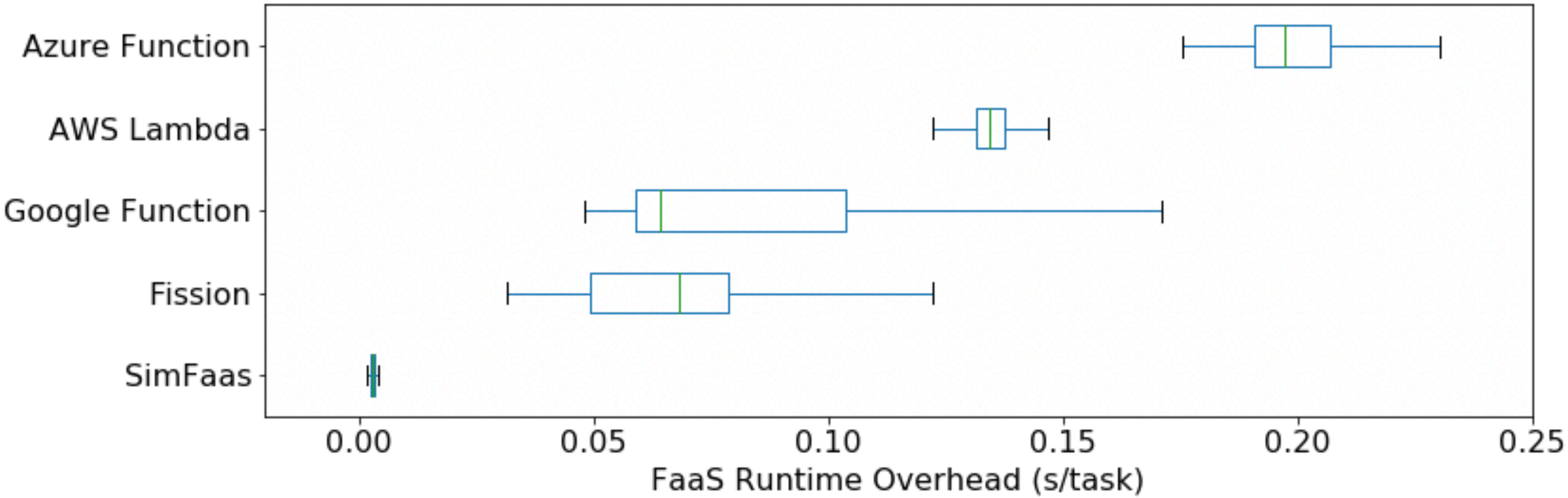
Chronos workload



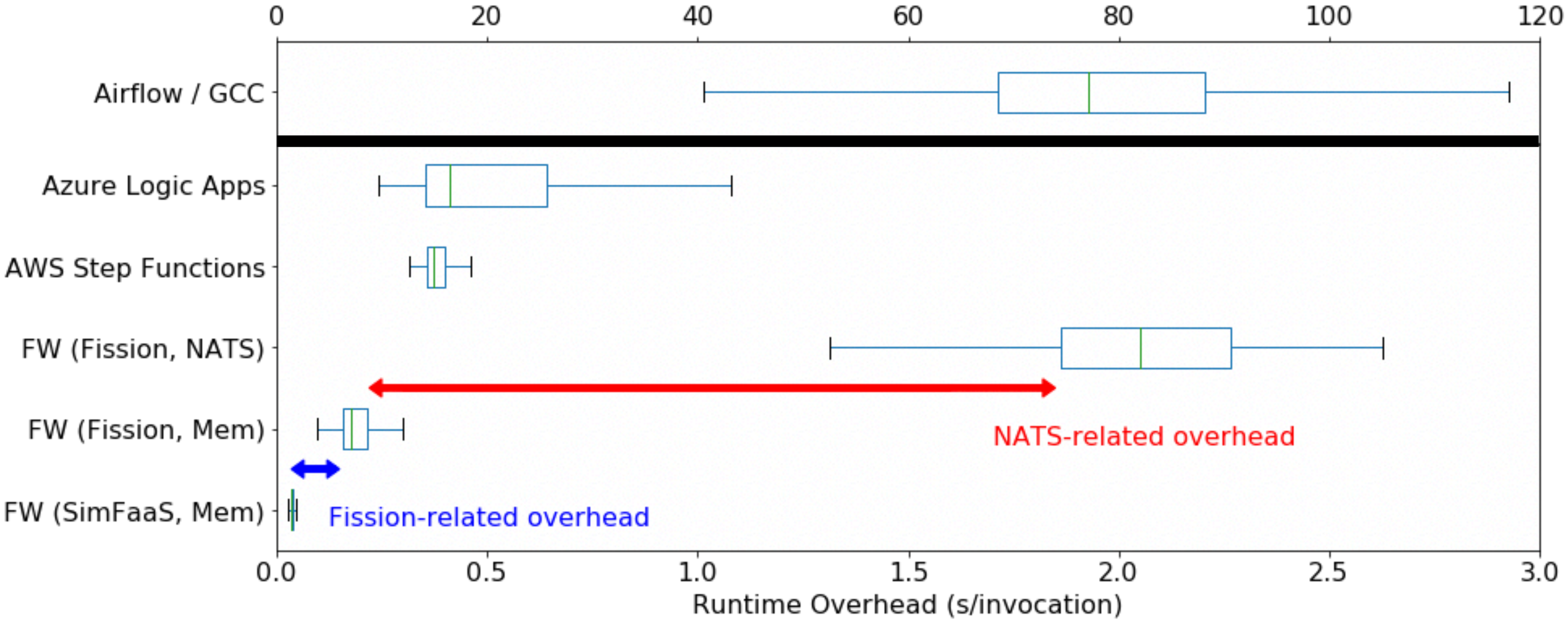




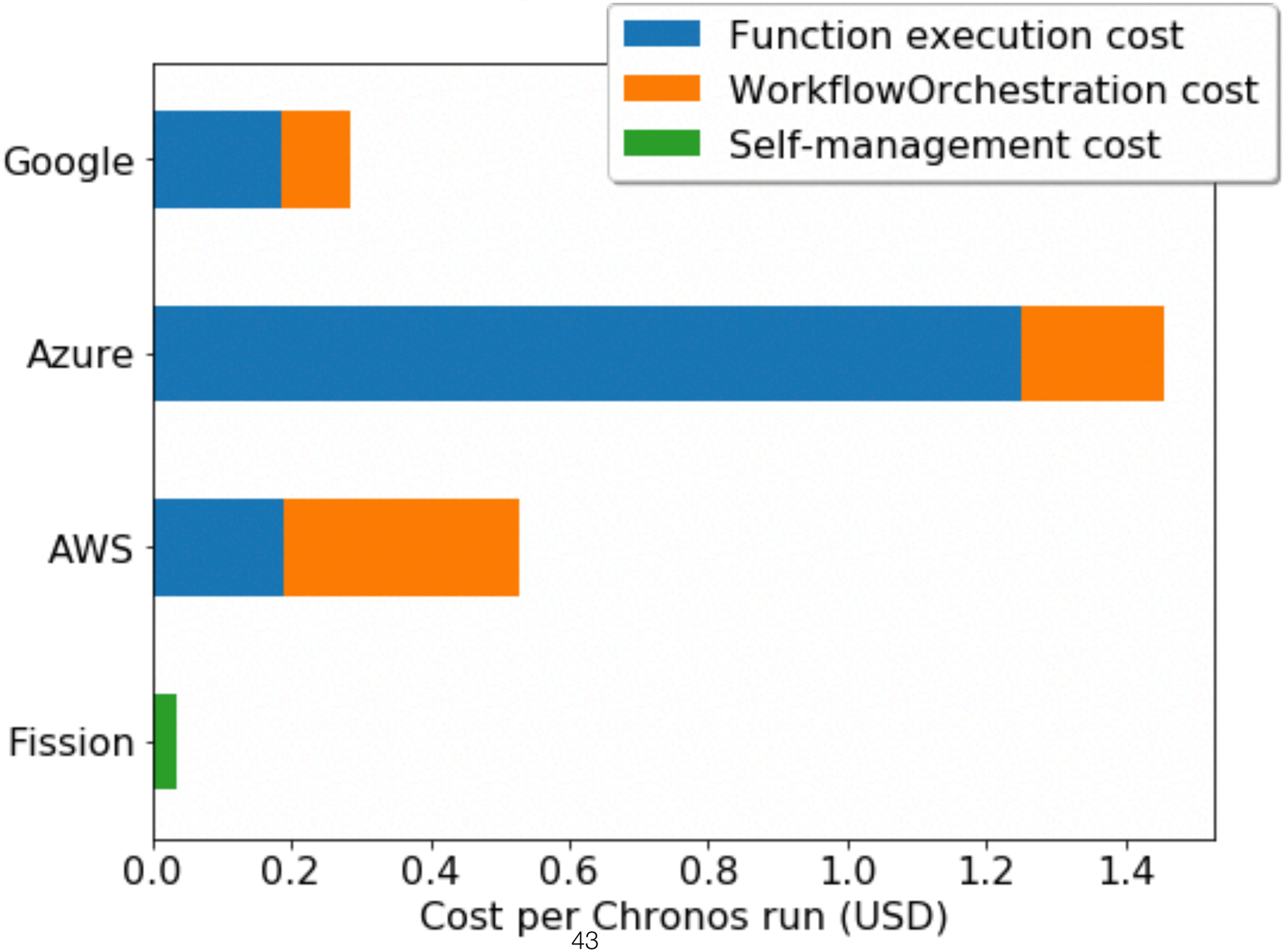
Realistic experiment: FaaS overhead



Realistic experiment: performance



Realistic experiment: cost



Process

~59k lines of code

~150 hours of experiments
~105 GB of experiment data

~689 cups of coffee

10 publications

4 as lead author

60 citations

source: https://scholar.google.nl/citations?user=5l4JxcAAAAAJ&hl=nl&oi=ao (accessed June 2019)

Internship @ Platform9

TC

Login

Startups
Apps
Gadgets
Videos
Audio
Extra Crunch NEW
Newsletters
—
Events
Advertise
More

Search

Apple
Transportation
Enterprise
Cybersecurity 101

Platform9's Fission Workflows makes it easier to write complex serverless applications

Frederic Lardinois @fredericl / 2 years ago

Comment

InfoQ

En | 中文 | 日本 | Fr | Br

1,274,004 May unique visitors

Development | Architecture & Design | AI, ML and Data Engineering | Culture & Methods | DevOps | Videos with Transcripts NEW

FEATURED: Streaming | Machine Learning | Reactive | Microservices | Containers | Observability | Continuous Delivery | QCon is

InfoQ Homepage > Articles > Four Techniques Serverless Platforms Use To Balance Performance And Cost

CLOUD

Four Techniques Serverless Platforms Use to Balance Performance and Cost

LIKE 1 BOOKMARKS

FEB 13, 2019 • 25 MIN READ

by

Erwin van Eyk

FOLLOW

reviewed by

Richard Seroter

FOLLOW

Key Takeaways

- The cost and performance models are two of the key drivers of the popularity of serverless and Function-as-a-Service (FaaS).
- Cold starts have gone down a lot, from multiple seconds to 100s of milliseconds, but there is still much space for improvement.
- There are various techniques that are being used to improve the performance of serverless functions, most of which focus on reducing or avoiding cold starts.
- These optimizations are not free; it is a trade-off between performance and cost, which depends on the requirements of your application.

RELATED CONTEN

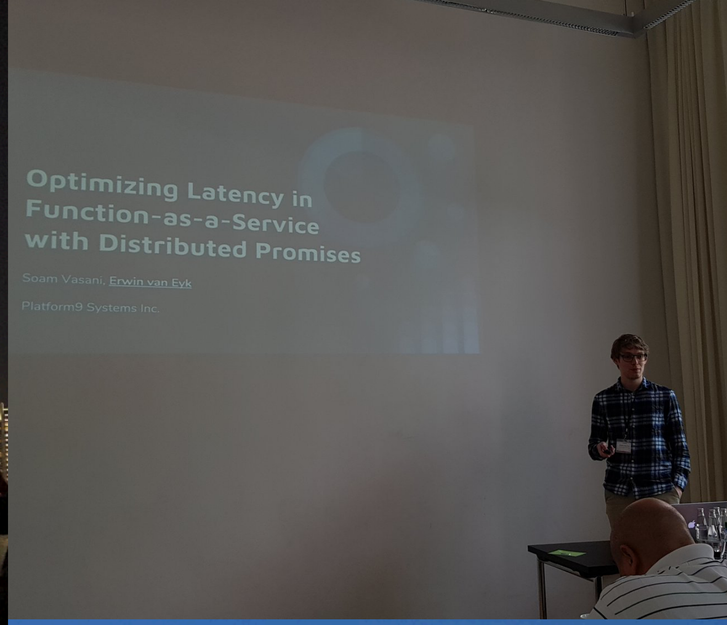
Akamai IoT Edge Connec Serverless Edge Platforn
JUN 15, 2019

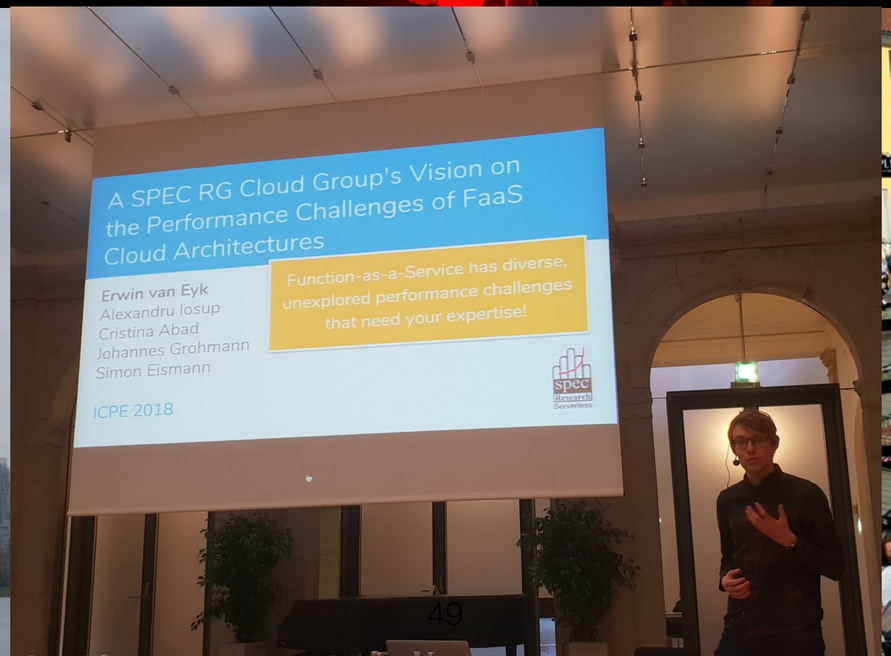
Cloudflare Releases Wor Value Store at the Edge
MAY 28, 2019

Building Responsiv Serverless, Event-D
JAN 05, 2019

Industry publications


KubeCon
CloudNativeCon
Europe 2019
Serverless Operations
From dev to production
Erwin van Eyk
PLATFORM9
fission
@Large Research
spec


Optimizing Latency in
Function-as-a-Service
with Distributed Promises
Soam Vasani, Erwin van Eyk
Platform9 Systems Inc.


A SPEC RG Cloud Group's Vision on
the Performance Challenges of FaaS
Cloud Architectures
Erwin van Eyk
Alexandru Iosup
Cristina Abad
Johannes Grohmann
Simon Eismann
Function-as-a-Service has diverse,
unexplored performance challenges
that need your expertise!
ICPE 2018
spec
Research
Serverless

Talks

Conclusion

1. Workflows are key to enabling serverless function composition.
2. Fission Workflows demonstrates the possibilities of serverless workflows; and highlights the opportunities of prewarming.
3. The prototype is on-par performance-wise, and cheaper than the state-of-the-art.
4. Industry interest in the Fission Workflows product emphasises the need for serverless workflow systems.

More research is needed in serverless computing!

Thanks!

fission workflows

https://github.com/fission

SimFaaS

https://github.com/erwinvaneyk/simfaas

Thesis + slides

https://erwinvaneyk.nl/thesis

Erwin van Eyk
Software Engineer @ Platform9 Systems
Researcher @ AtLarge Research
Co-chair @ SPEC CLOUD RG Serverless

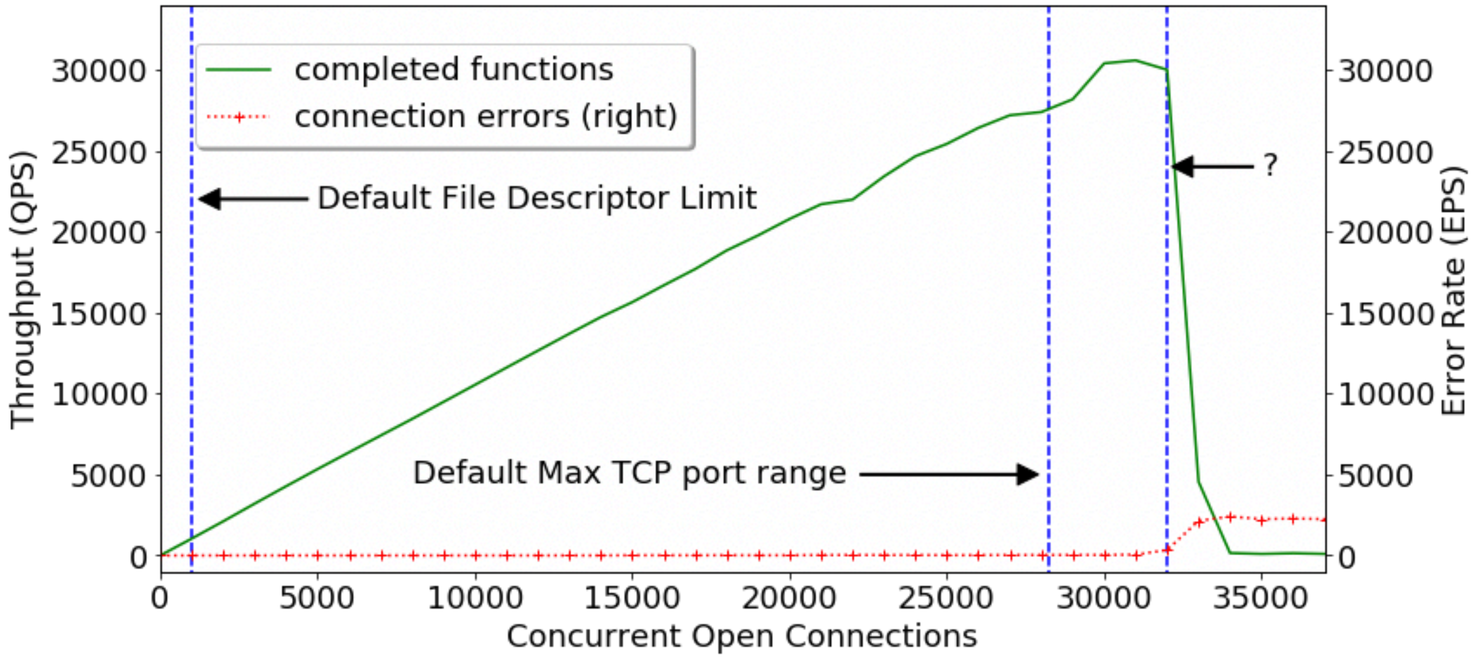
@erwinvaneyk
erwinvaneyk@gmail.com
https://erwinvaneyk.nl

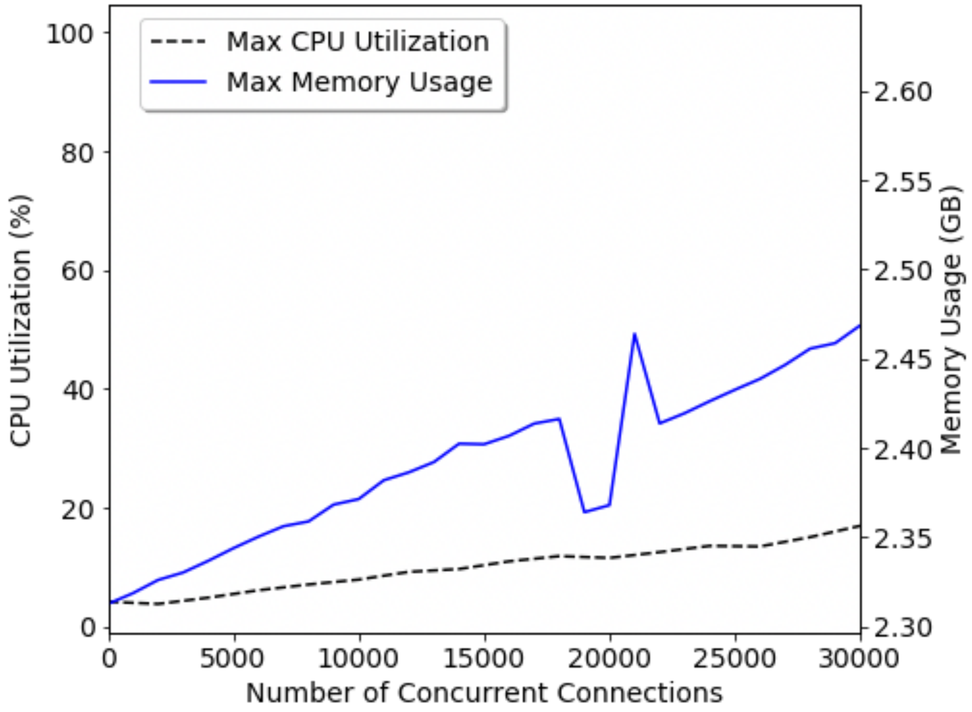
Additional Slides

Photo credits

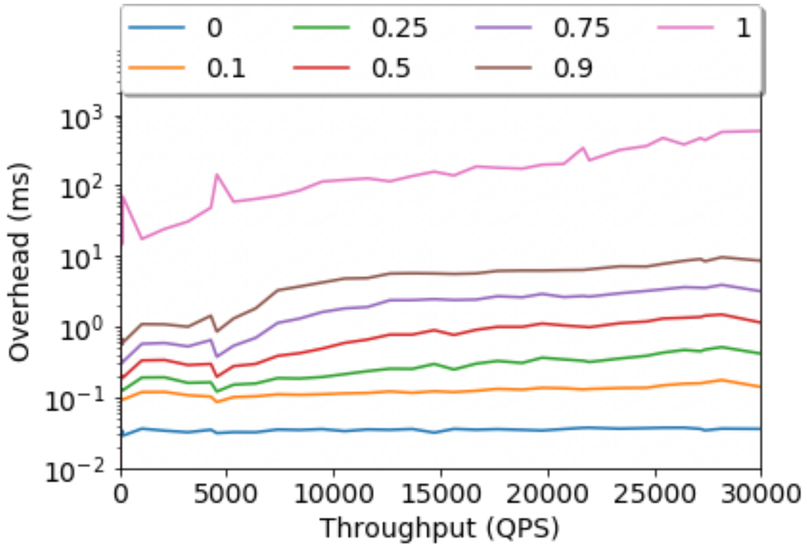
- Empty datacenter: https://www.datacenterknowledge.com/manage/wave-data-center-consolidation-different-first-one
- Copyright of logos used (Google, AWS, Azure, SPEC RG CLOUD, TU Delft, Platform9, Python) belongs to the respective organizations
- Logo for SWL: https://pixabay.com/vectors/hexagon-symbol-gui-internet-2307350/

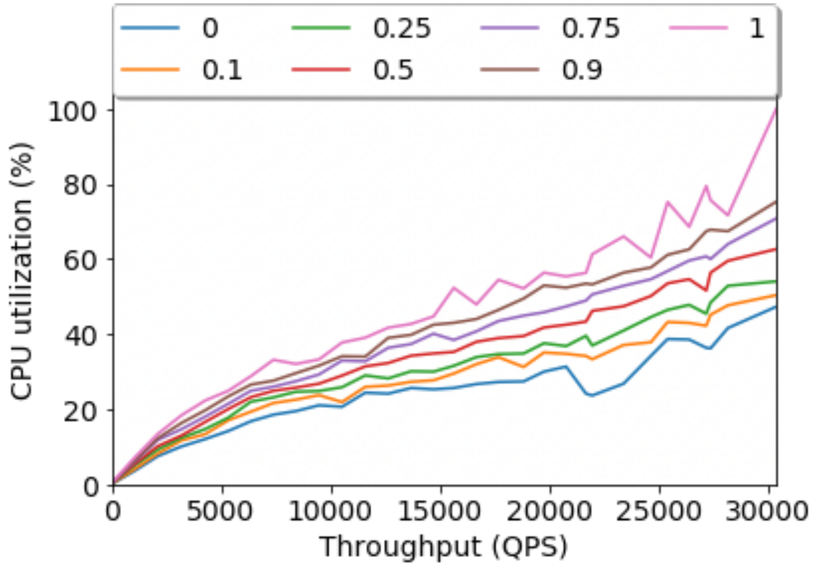
Performance overhead of SimFaaS (1)

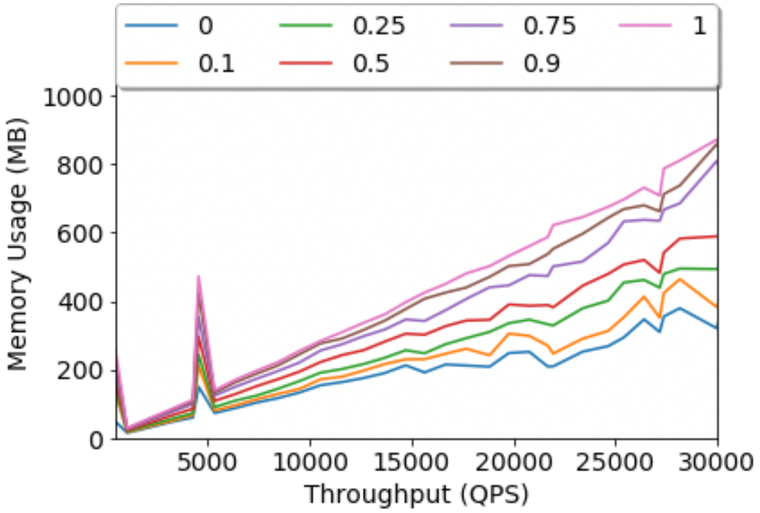


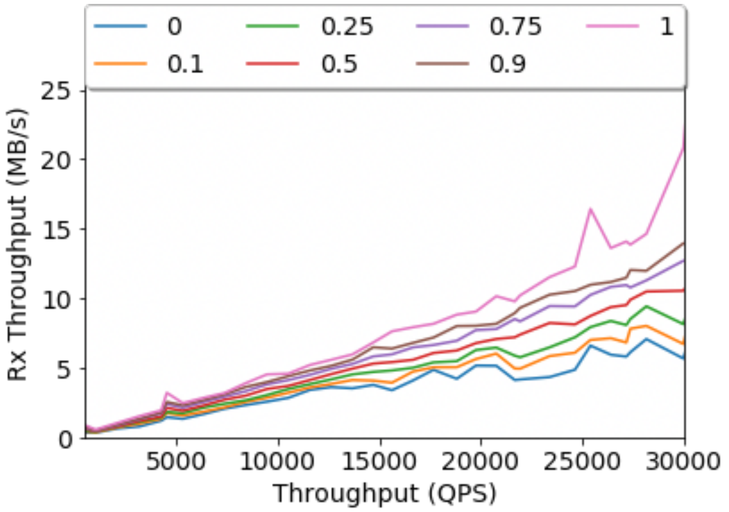


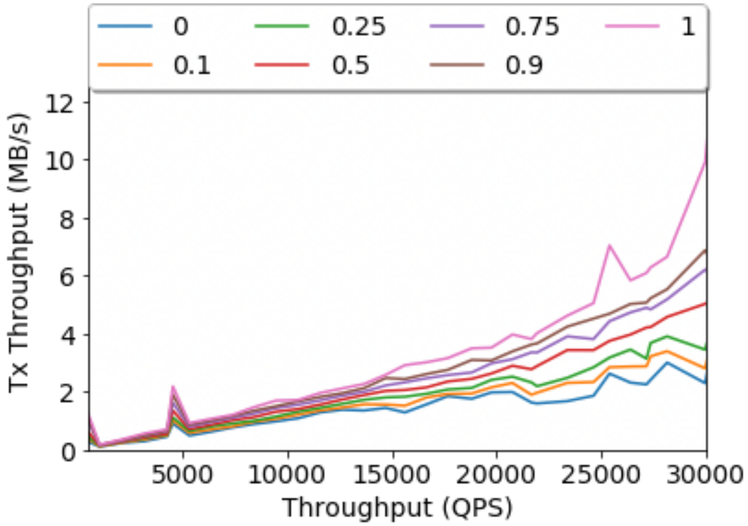
Performance overhead of SimFaaS (2)



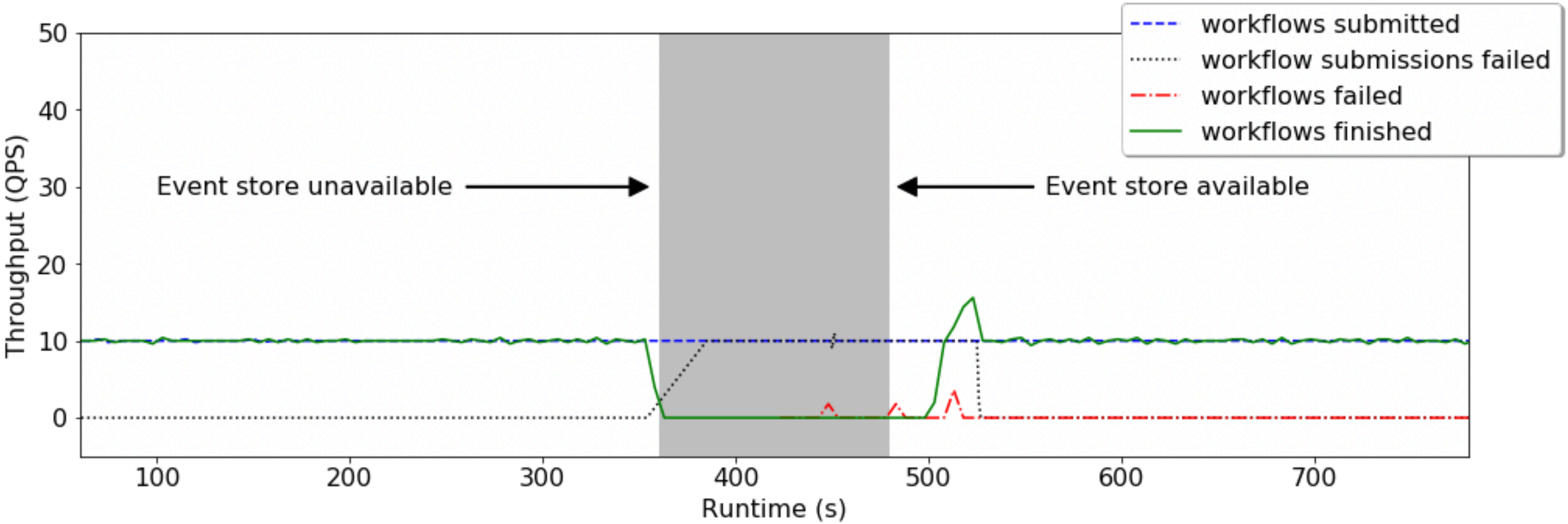




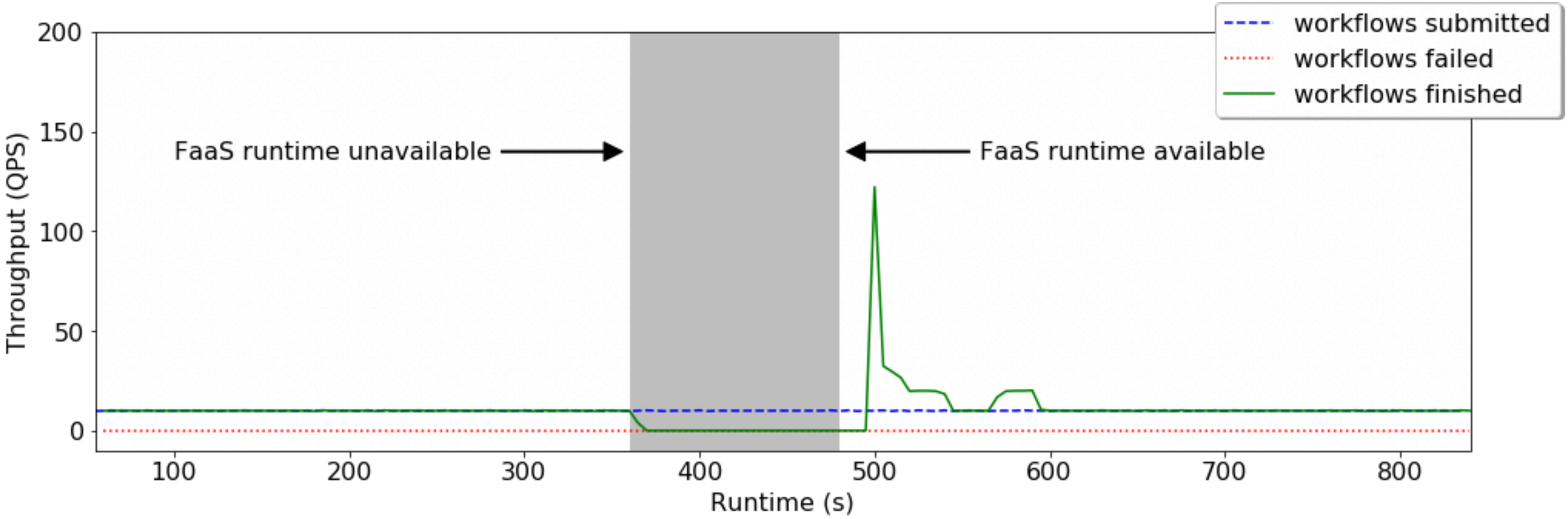




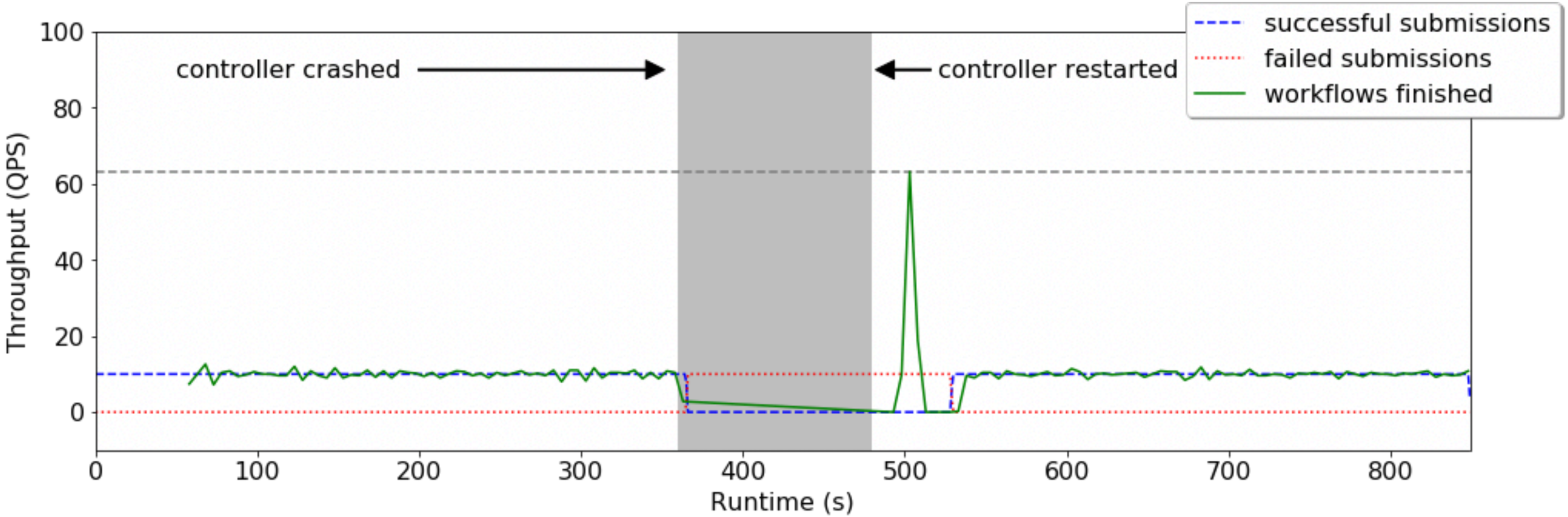
Fault-tolerance: event store unavailability



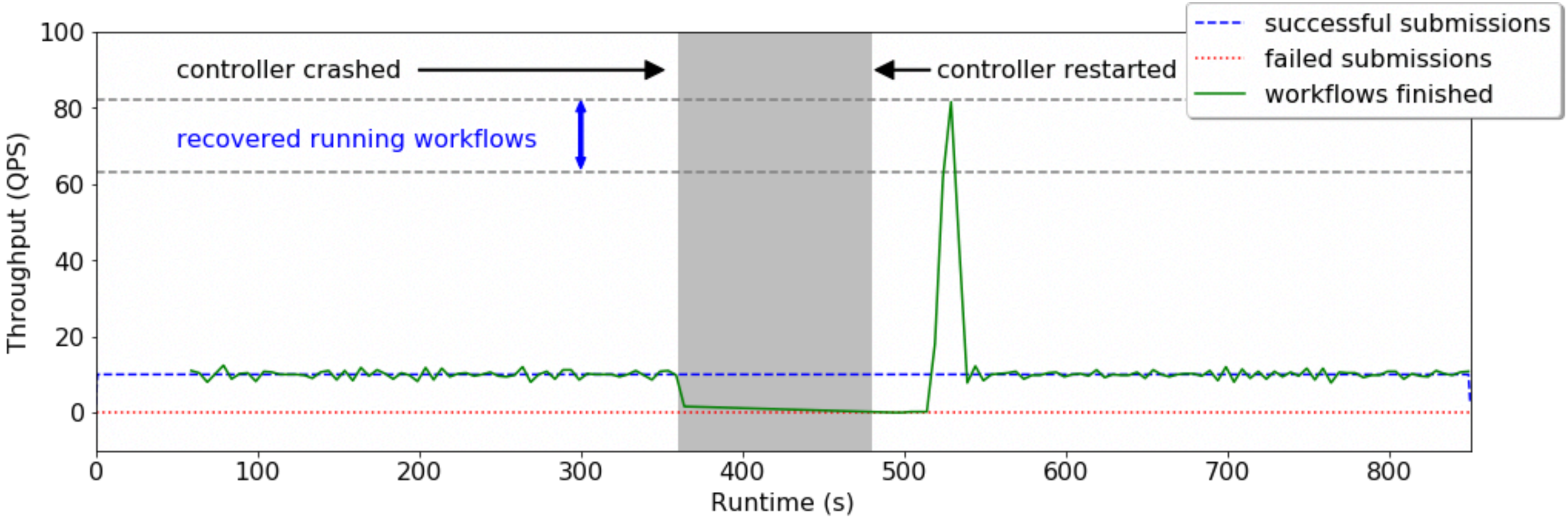
Fault-tolerance: FaaS runtime unavailability



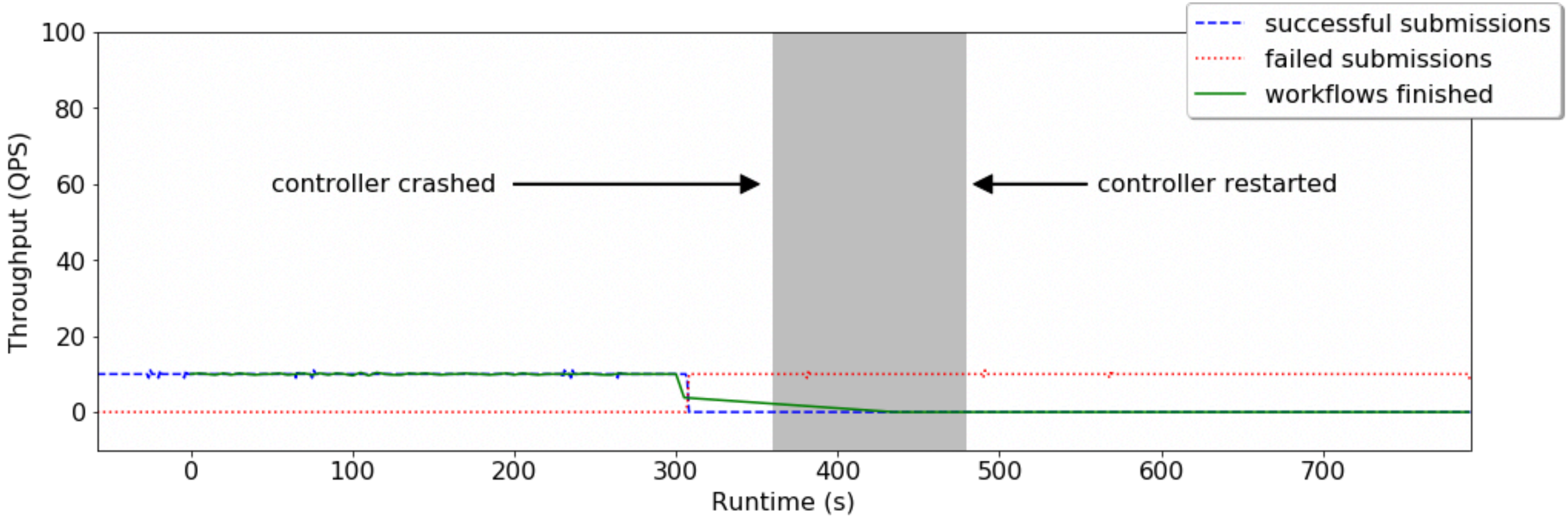
Fault-tolerance: crash of bundled workflow engine



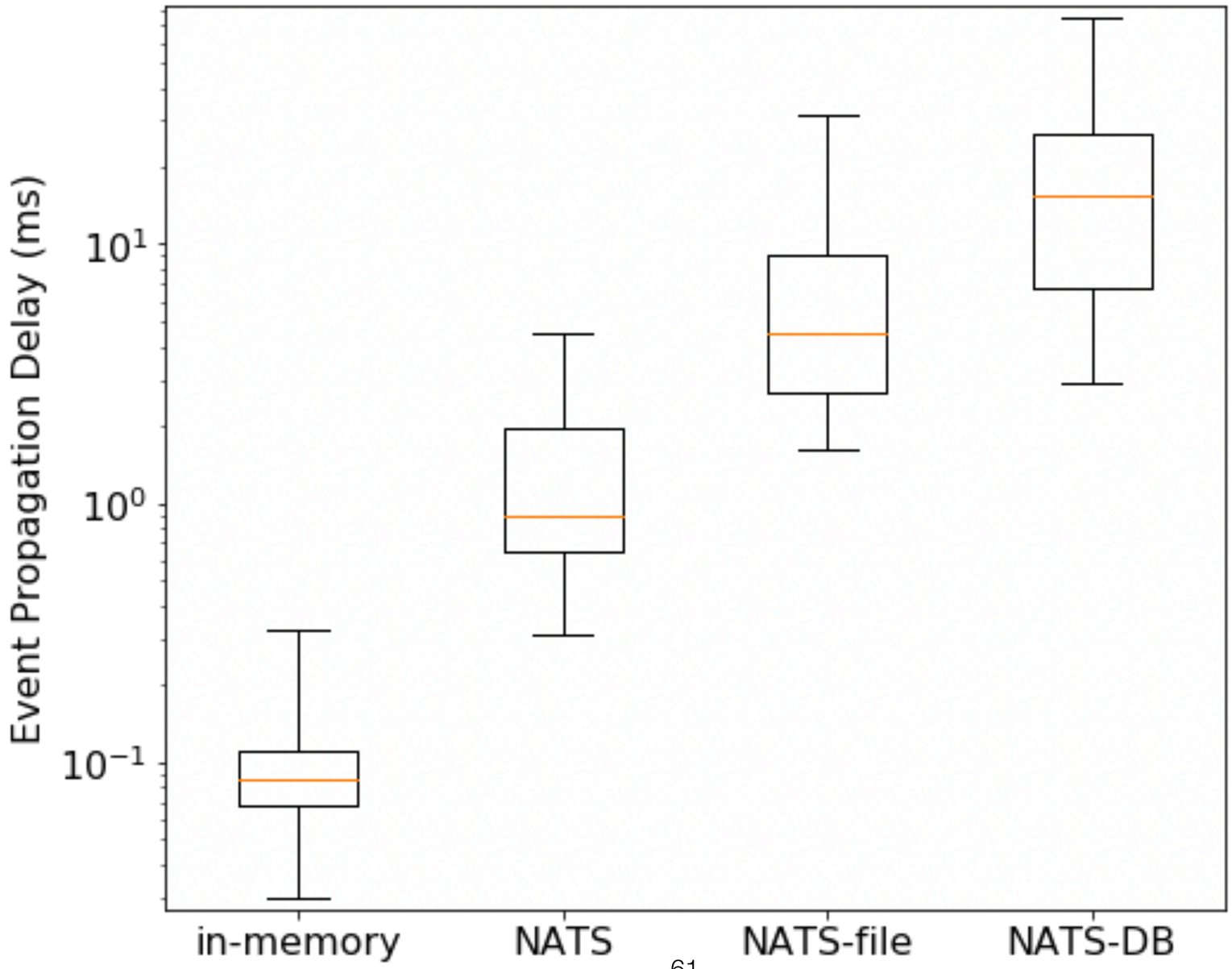
Fault-tolerance: crash of distributed workflow engine



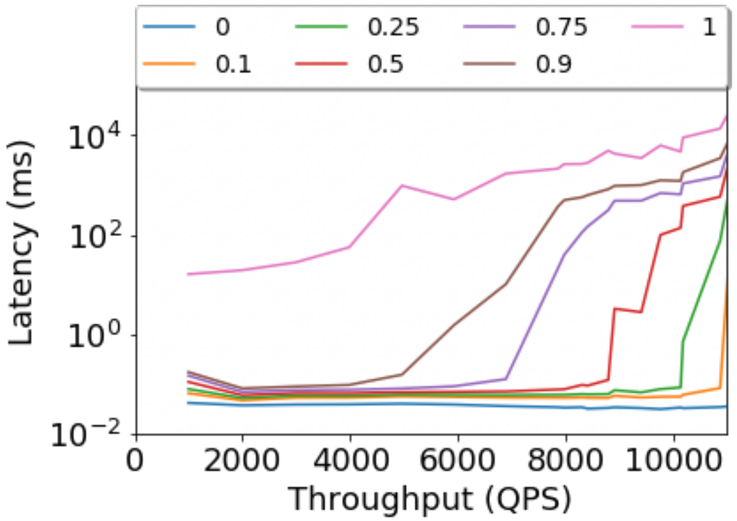
Fault-tolerance: crash of workflow engine with an in-memory event store

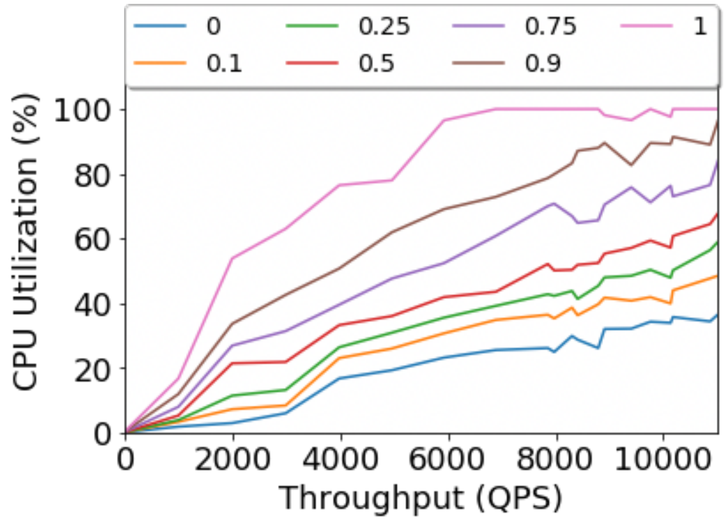


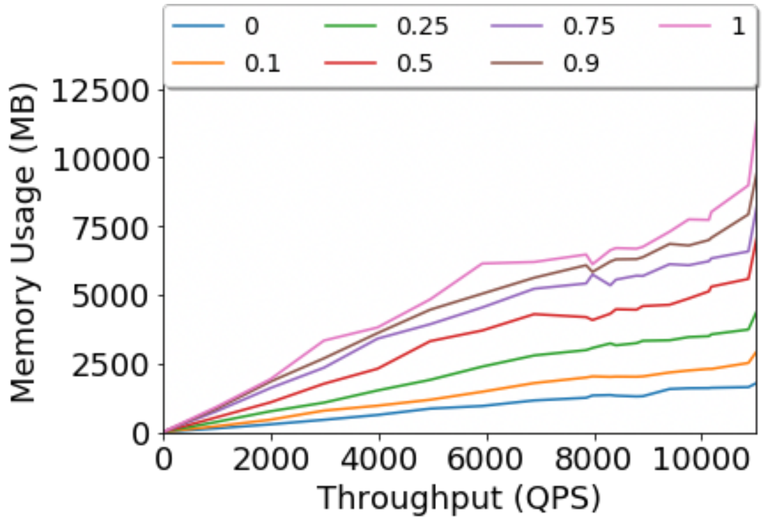
Performance overhead of event store implementations

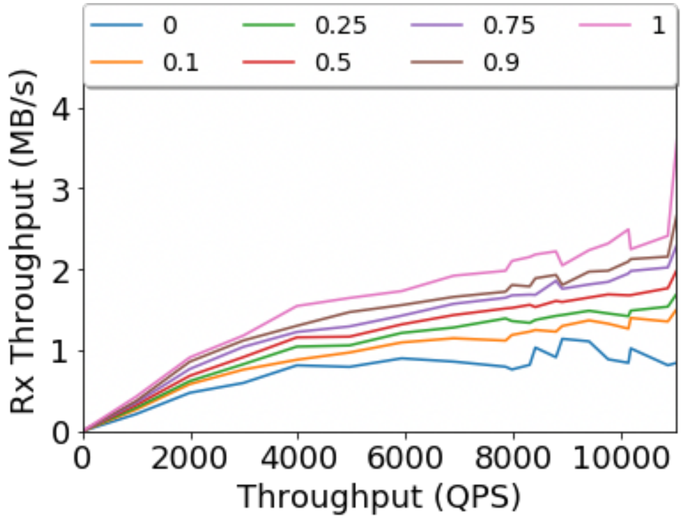


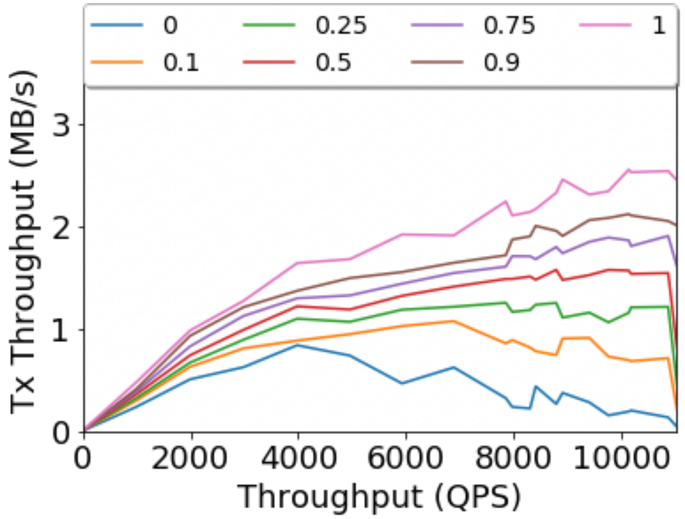
Scalability: workflow invocation submission



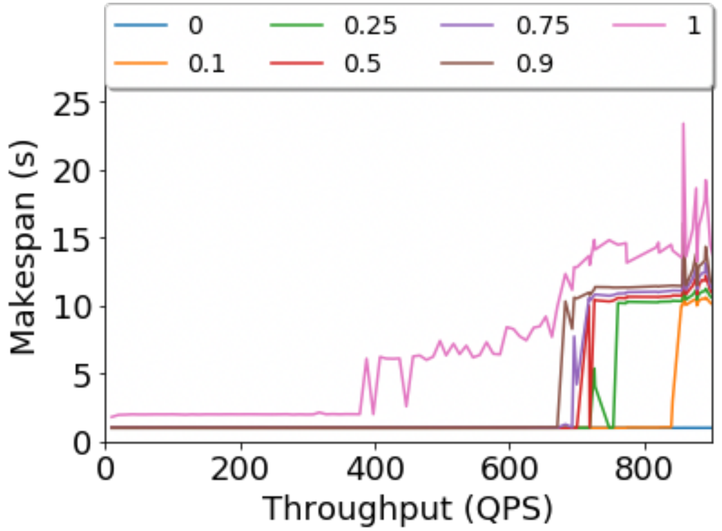


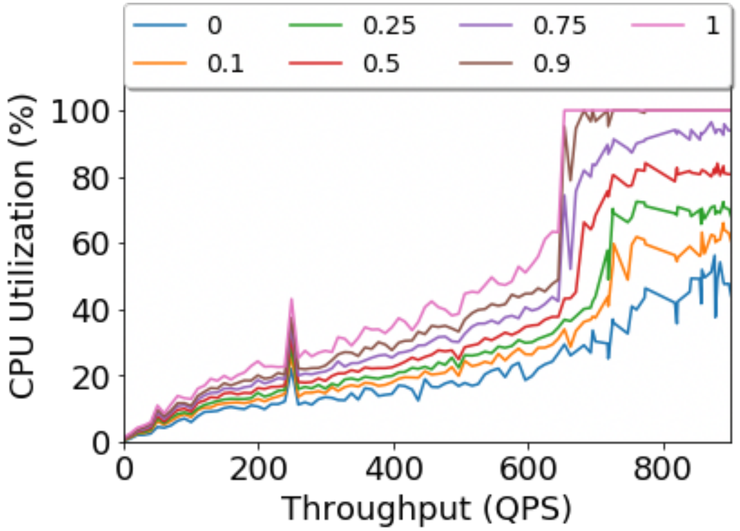


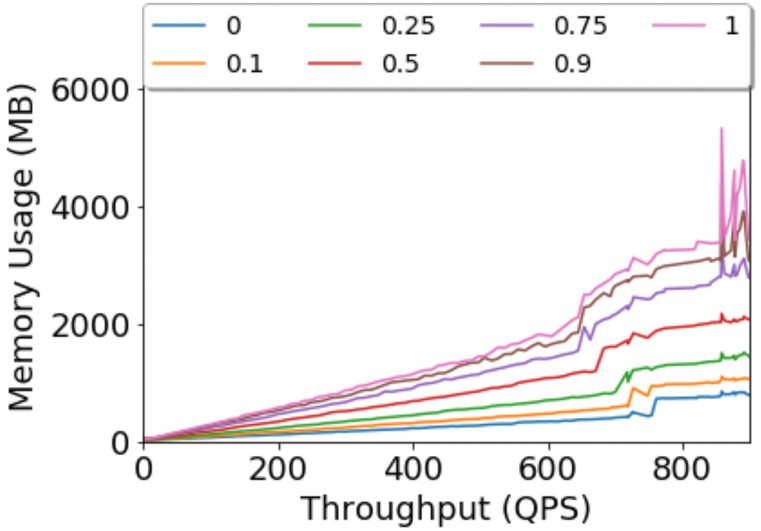


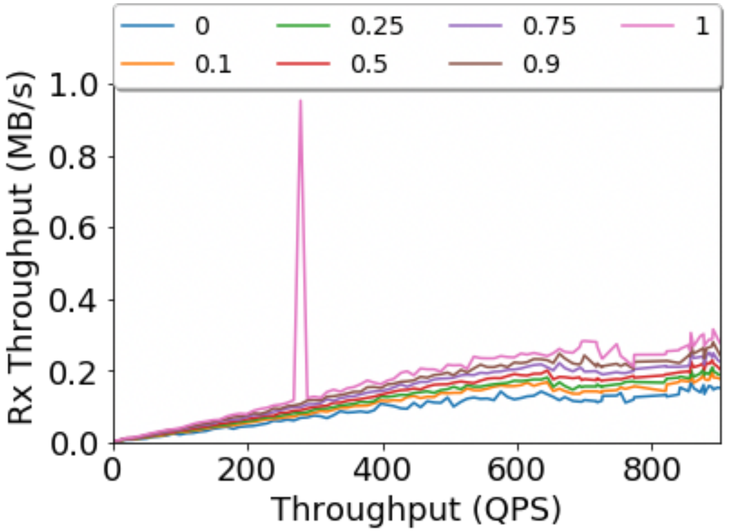


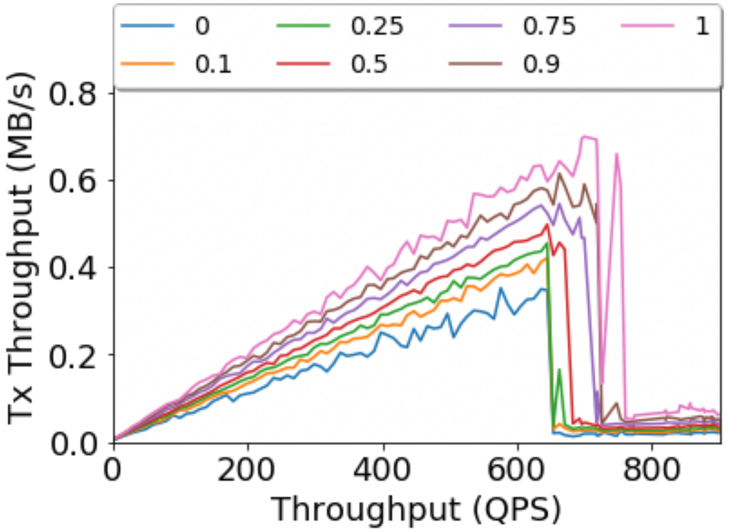
Scalability: 1-task workflows



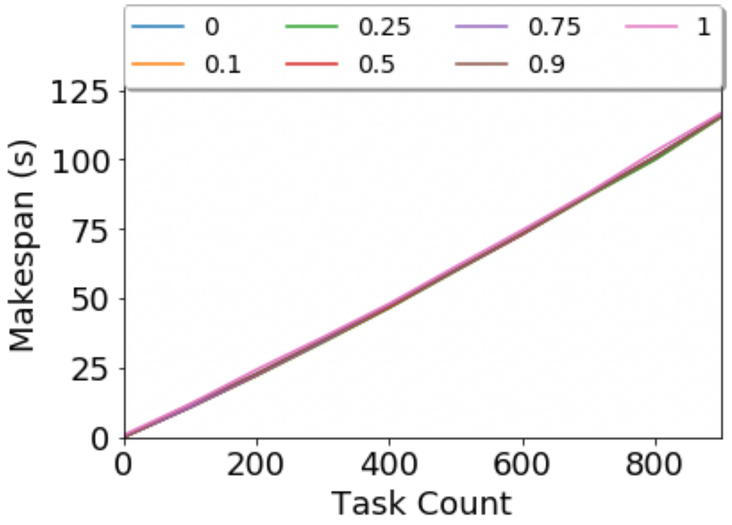


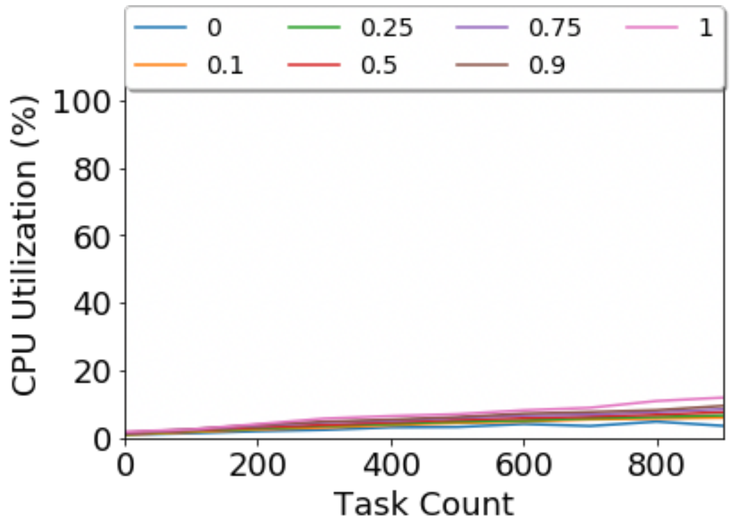


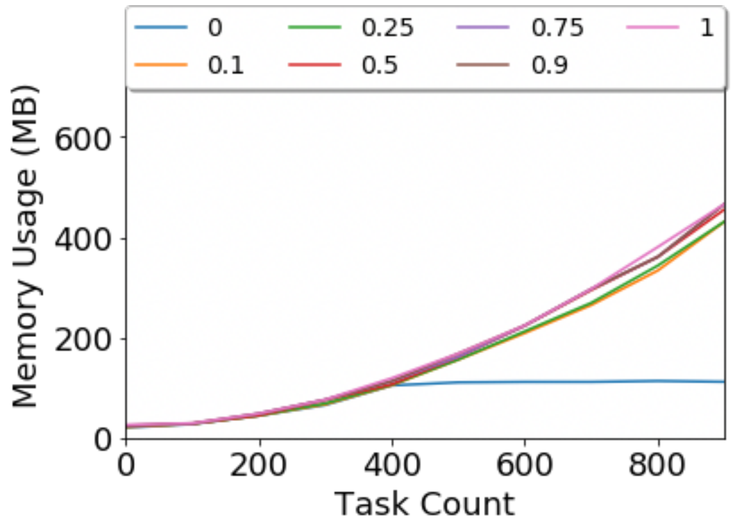


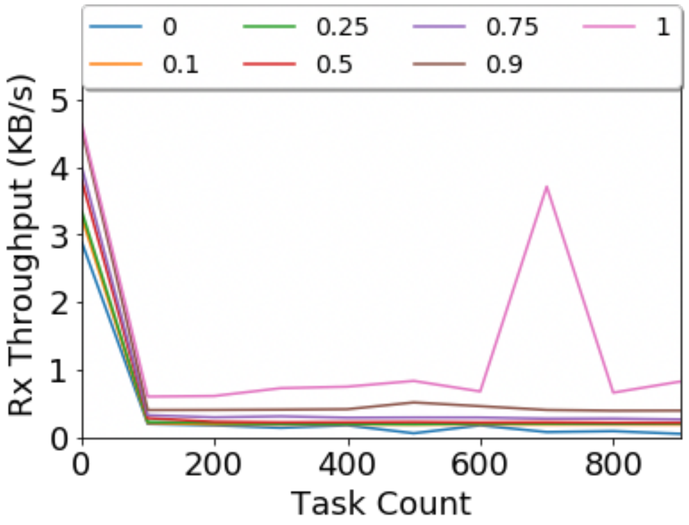


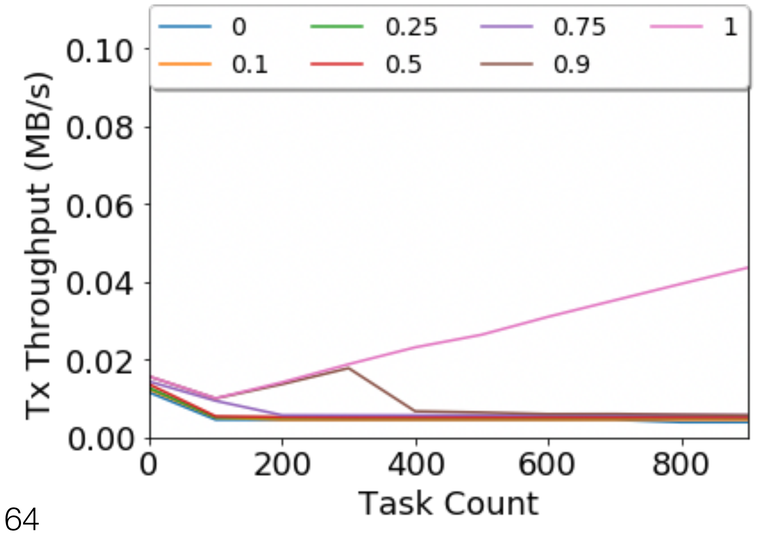
Scalability: serial/long-running workflows



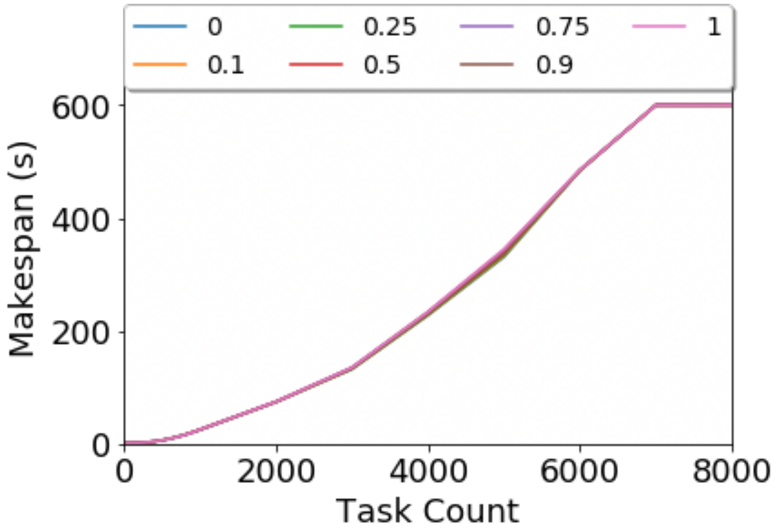


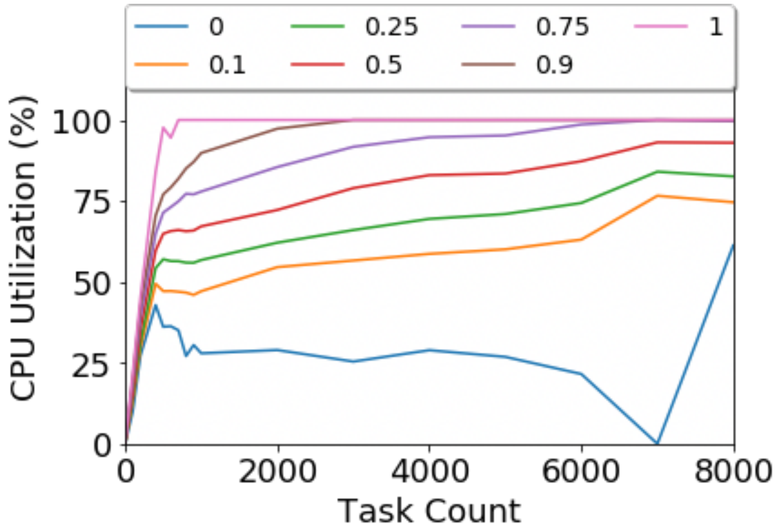


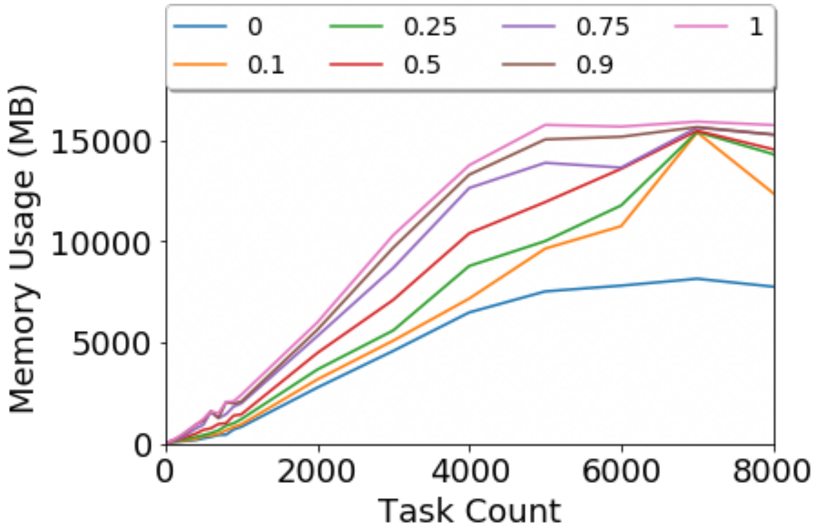


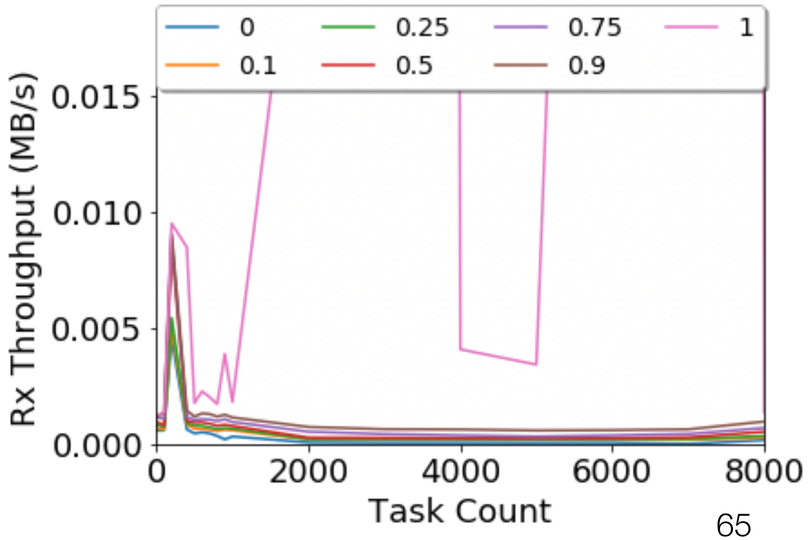


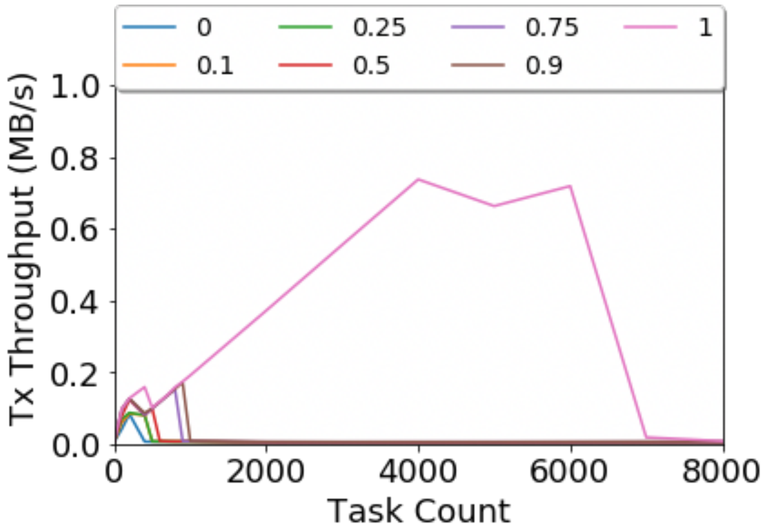
Scalability: parallel workflows



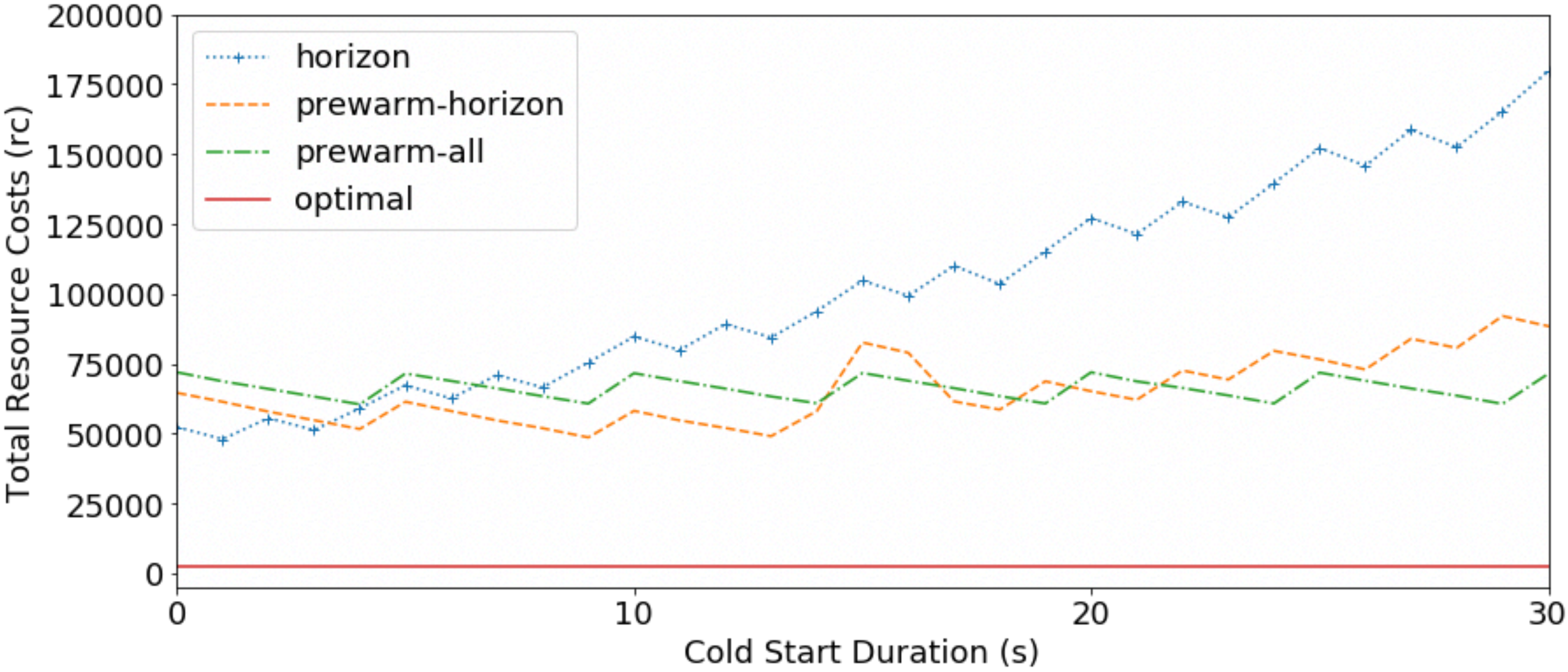




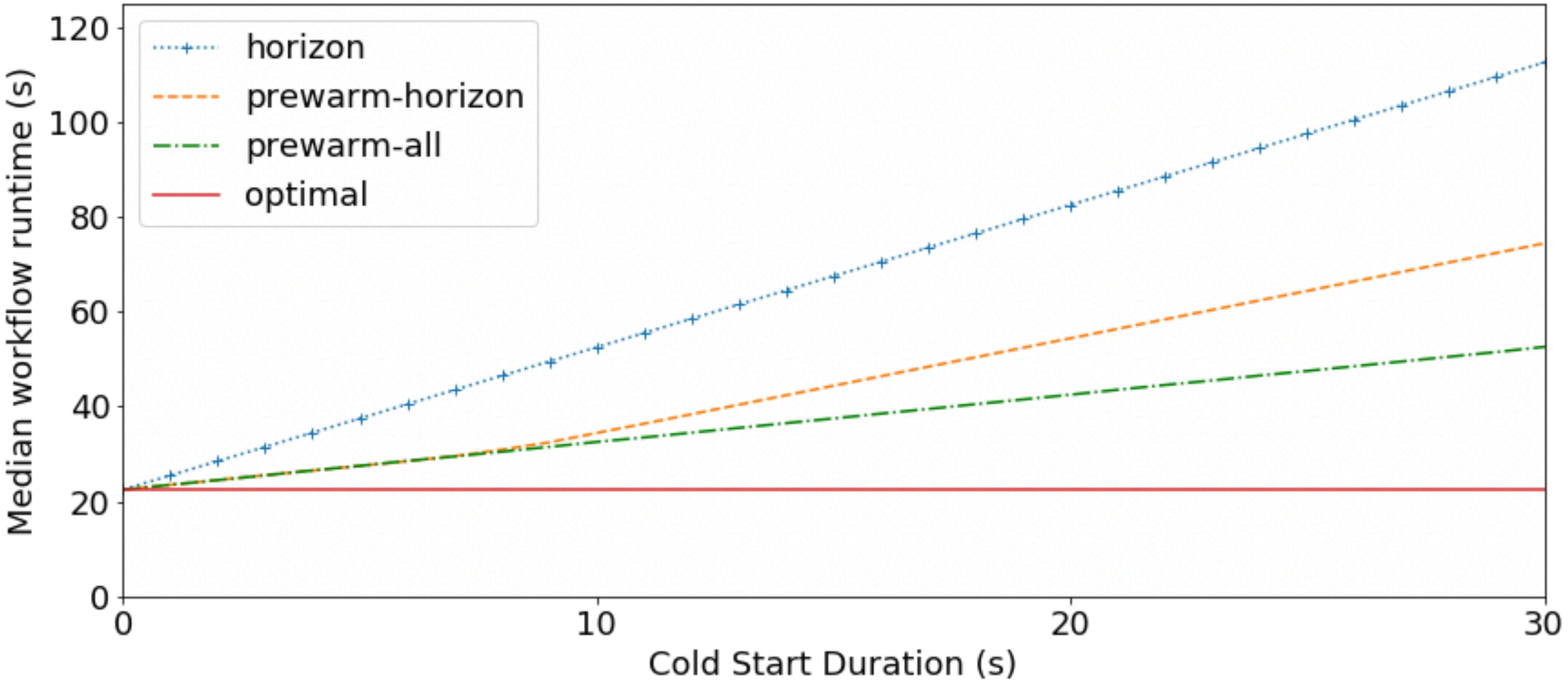




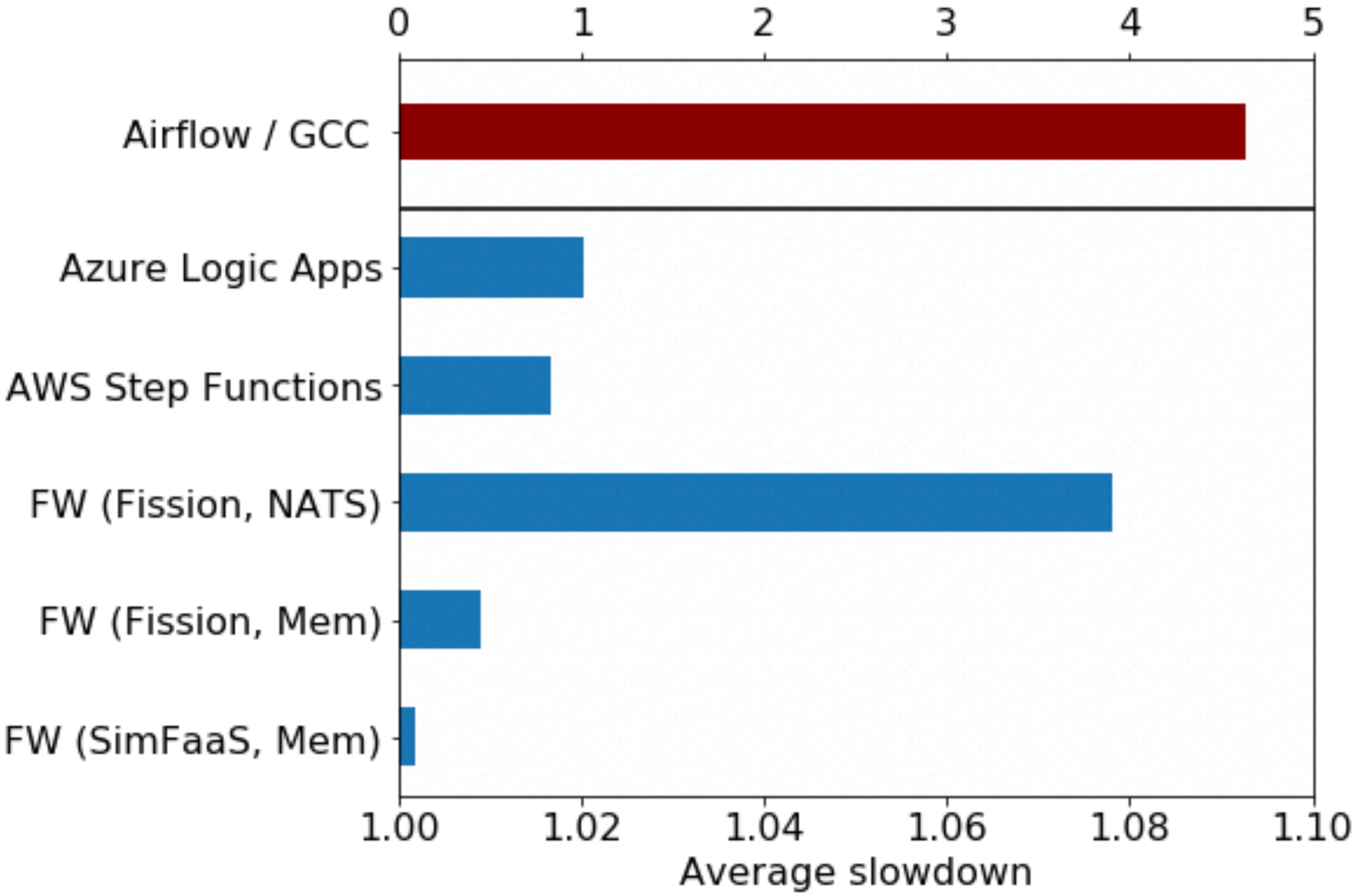
Resource consumption of scheduling policies



Performance of scheduling policies



Slowdown of the serverless workflow engines



Cost breakdown of the evaluated serverless platforms

Cost item	Fission Workflows	AWS	Azure	Google Cloud
Compute nodes	4	-	-	4
Node cost per hour	0.0475	-	-	0.0475
Other hourly costs	-	-	-	0.4092
Function cost per second	-	0.00000208	0.000014	0.00000203
Function cost per execution	-	0.0000002	0.000000169	0.0000004
Orchestration cost per workflow	-	0.0001	0.00006	-
Experiment duration (seconds)	651.0	653.2	650.5	650
Total Costs	0.0343	0.5278	1.455	0.293

Fission Workflow definition example

```
apiVersion: 1
output: WhaleWithFortune
tasks:
  GenerateFortune:
    run: fortune

  WhaleWithFortune:
    run: whalesay
    inputs:
      body: "{ output('GenerateFortune') }"
    requires:
    - GenerateFortune
```

Survey of FaaS platforms

FaaS Platform	F1	F2	F3	F4	F5	F6
AWS Lambda	○	●	○	○	●	●
Azure Functions	○	●	○	○	◐	●
Google Cloud Functions	○	●	○	○	◐	●
OpenWhisk	◐	●	○	●	◐	●
OpenFaaS	●	◐	○	○	◐	◐
OpenLambda	●	○	●	○	○	◐
Kubeless	●	◐	◐	○	◐	●
Fission	●	◐	●	●	◐	●
Funktion	●	○	○	○	○	●

Survey of composition approaches

	Constraints				Requirements						
Approach	C1	C2	C3	C4	R1	R2	R3	R4	R5	R6	R7
Direct	●	●	●	●	●	○	○	○	●	●	●
Compiled	●	○	●	●	◐	○	●	●	○	○	●
Coordinator	●	●	●	●	●	○	○	○	●	●	●
Event-Driven	●	●	○	◐	○	●	●	●	○	◐	◐
Workflows	●	●	●	◐	●	●	◐	●	●	●	●

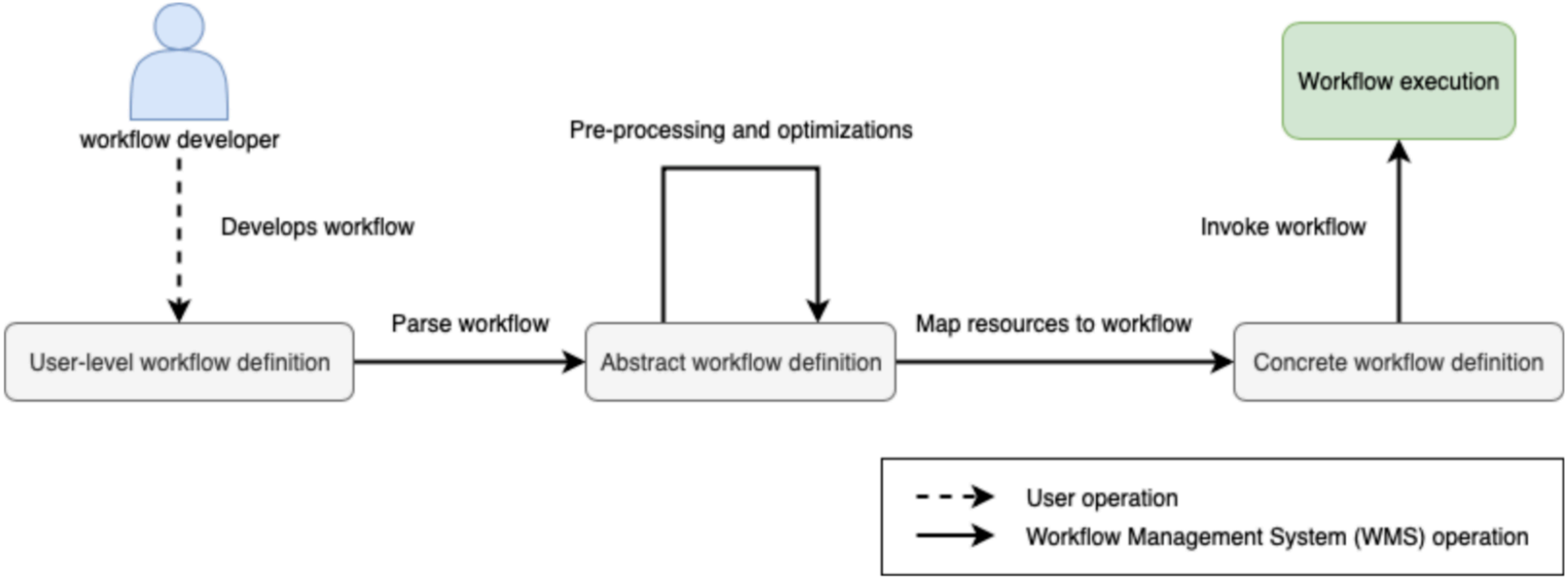
Survey of WMSs

Approach	Constraints				Requirements								
	C1	C2	C3	C4	R1	R2	R3	R4	R5	R6	R7	OSS	Docs
NodeRED	●	◐	◐	◐	●	◐	○	○	○	○	●	●	●
Openstack Mistral	◐	○	◐	○	◐	●	◐	●	○	●	●	●	●
AWS Step Functions	◐	●	●	●	◐	●	●	●	◐	●	●	○	○
Azure Logic Apps	●	◐	◐	●	●	●	●	●	◐	●	●	○	○
Pywren	●	○	●	○	○	●	●	○	○	●	●	●	◐
Apache Airflow	○	◐	○	○	◐	●	◐	●	○	◐	◐	●	●
Azkaban	◐	○	◐	◐	○	●	◐	●	○	◐	●	●	○
Luigi	○	○	◐	◐	◐	●	○	●	●	◐	●	●	●
Pegasus	◐	◐	○	◐	○	●	○	●	●	●	●	●	◐

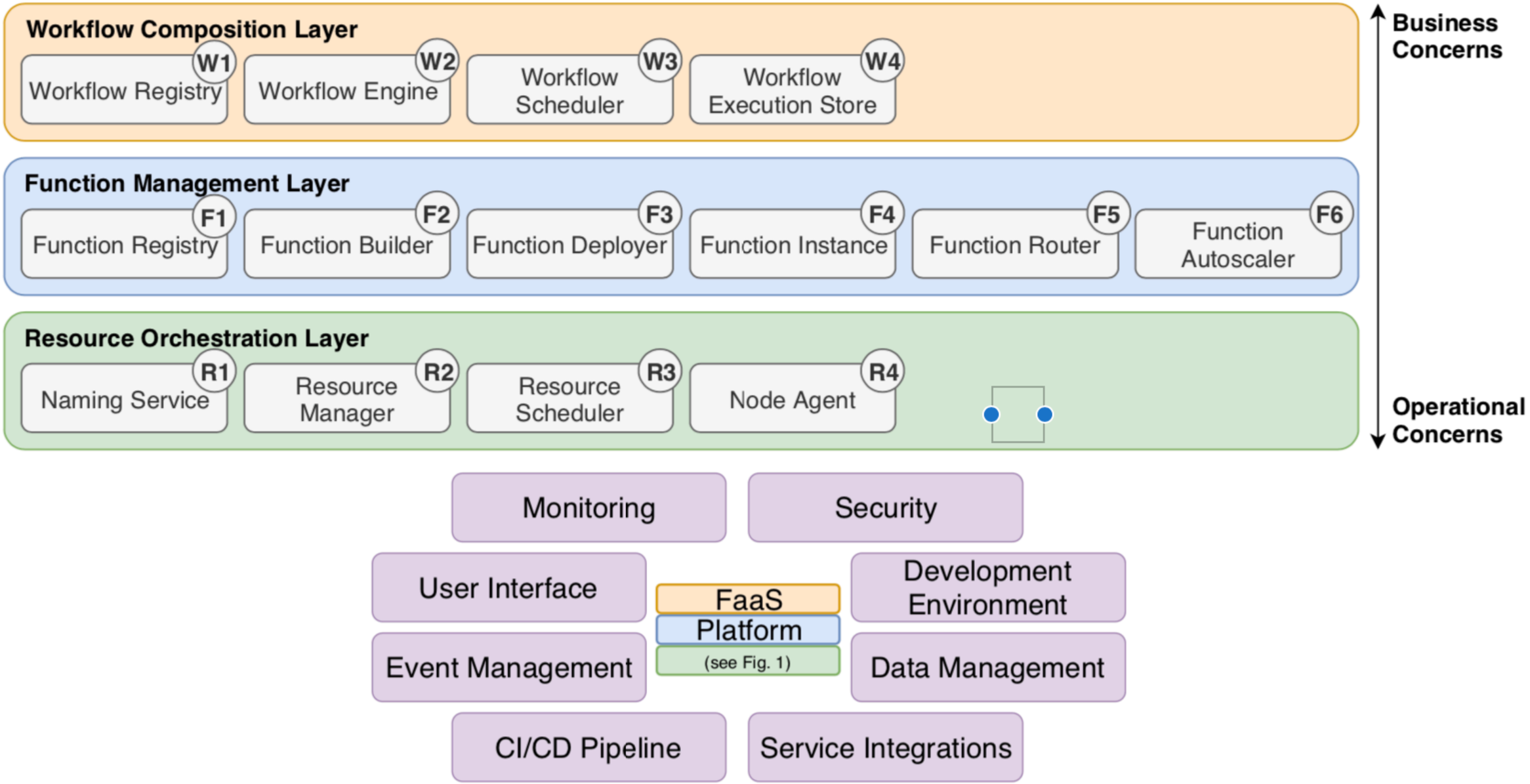
Survey of workflow languages

Workflow Language	Format	FaaS functions	Complex Control Flows	Readability	Model fit
BPMN 2.0	XML	○	●	○	○
WS-BPEL	XML	○	●	○	○
YAWL	XML	○	●	○	○
WDL	Custom	○	○	●	◐
CWL	YAML, JSON	●	○	●	◐
Amazon States Language	JSON	●	◐	◐	●
Openstack Mistral Language	YAML	◐	◐	●	◐
Apache Airflow's Workflow format	Python	◐	●	●	◐
Azure Workflow Definition Language	JSON	●	●	◐	◐
Pegasus DAX	XML	◐	○	○	◐
Azkaban Language	Custom	◐	○	◐	◐
NodeRED Flow File	JSON	○	○	○	○
Luigi's Workflow Format	Python	○	●	●	◐
Amazon Simple Workflow Framework	Java	◐	●	○	○

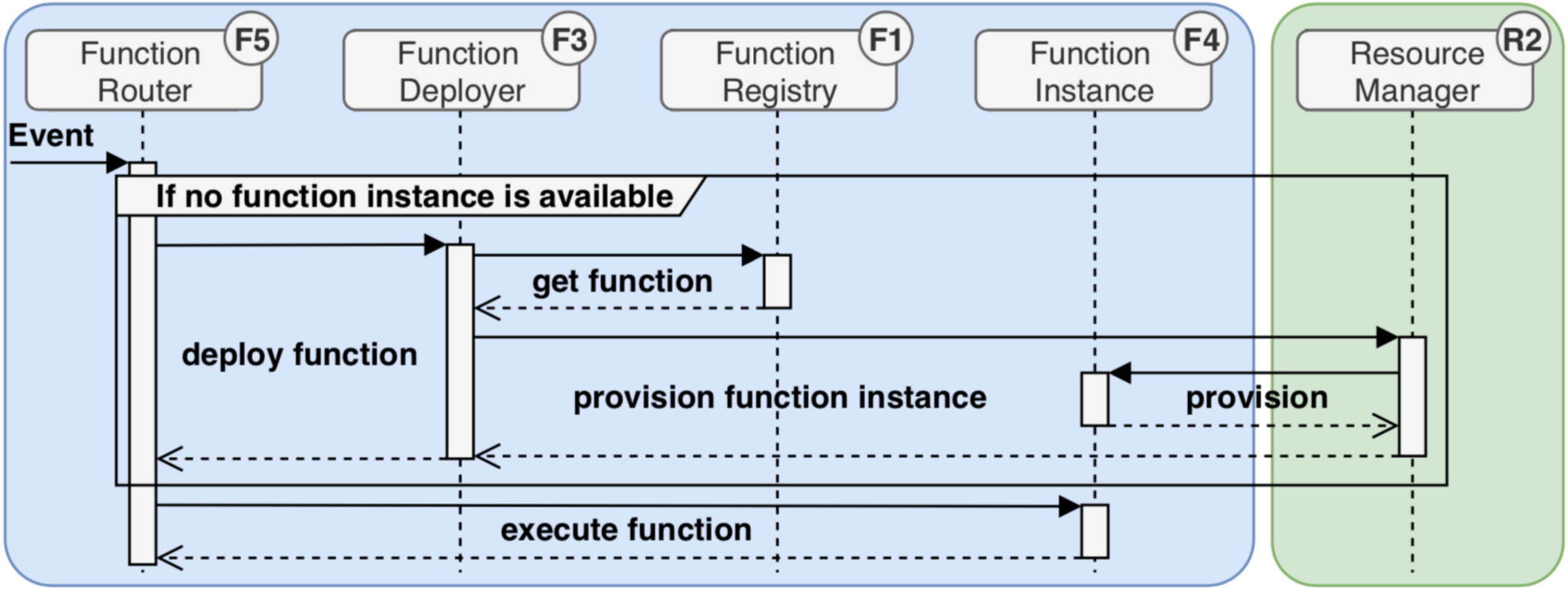
Workflow definition lifecycle



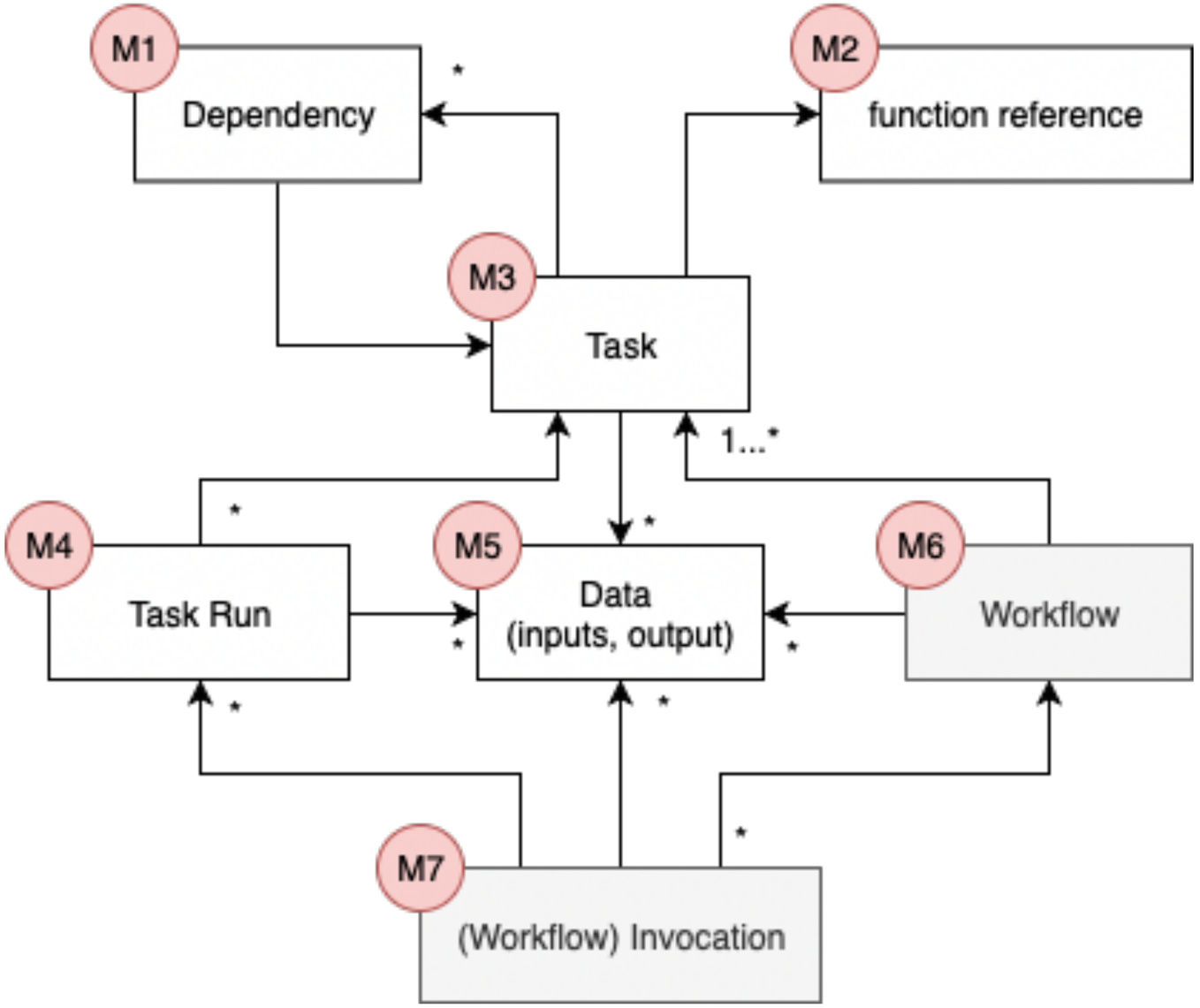
FaaS platform reference architecture



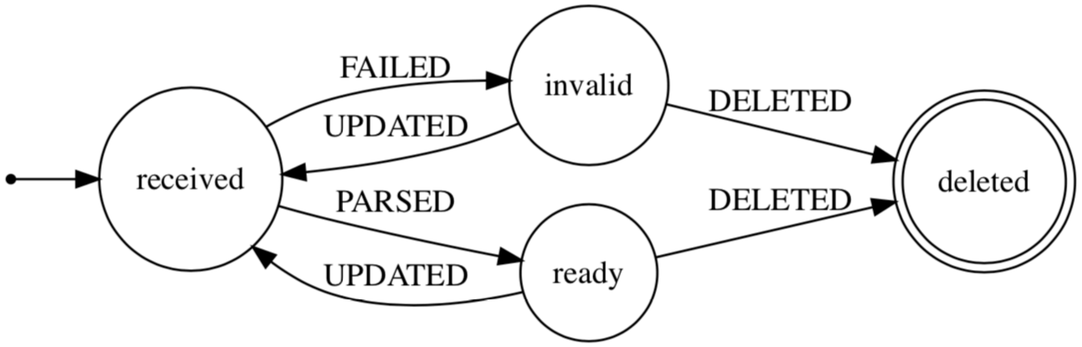
Serverless function execution



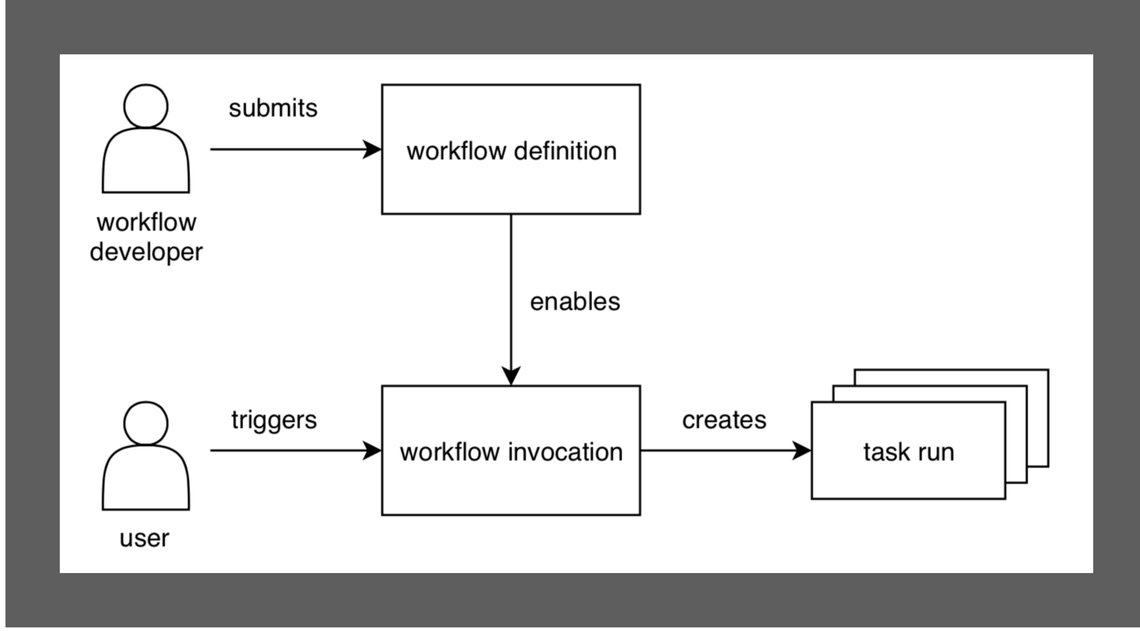
SWL: data model

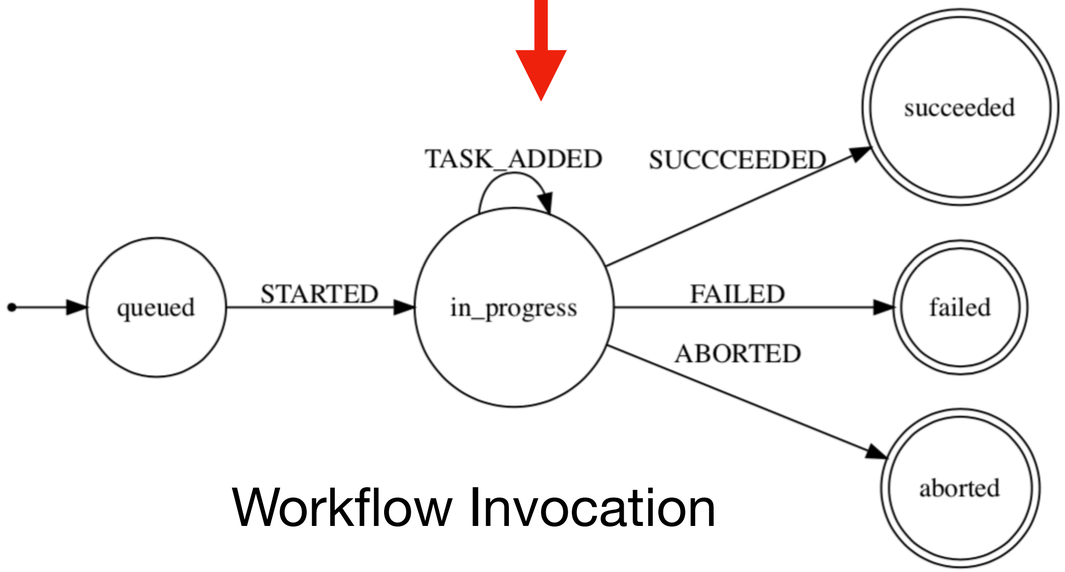


SWL: execution model

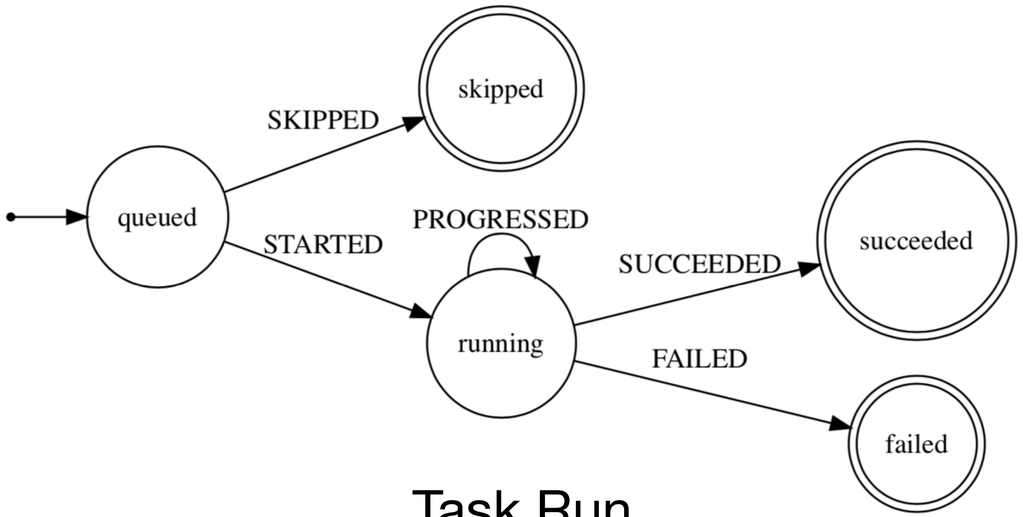


Workflow Definition



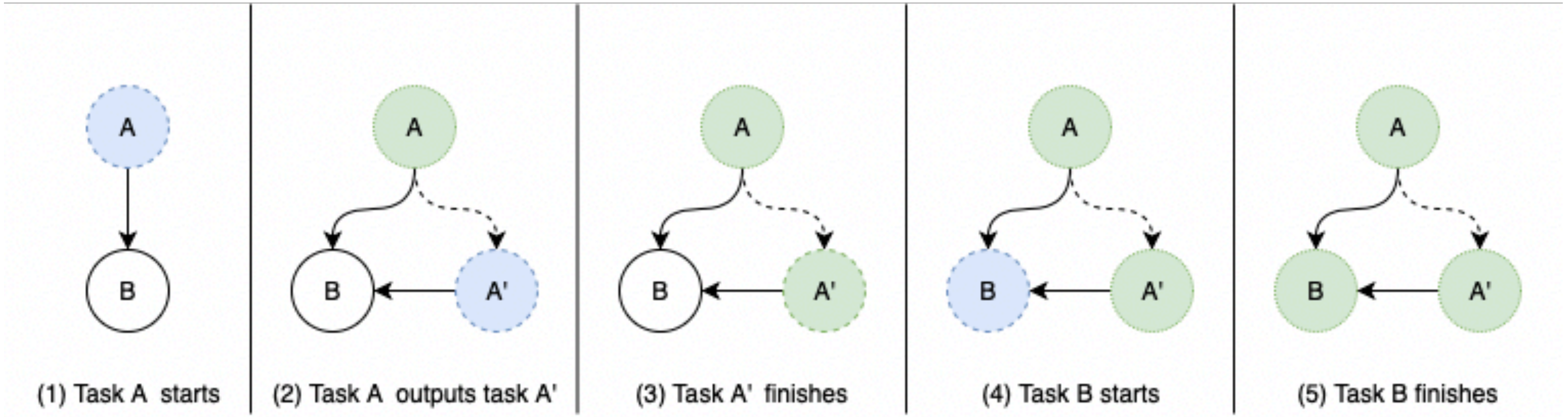


Workflow Invocation

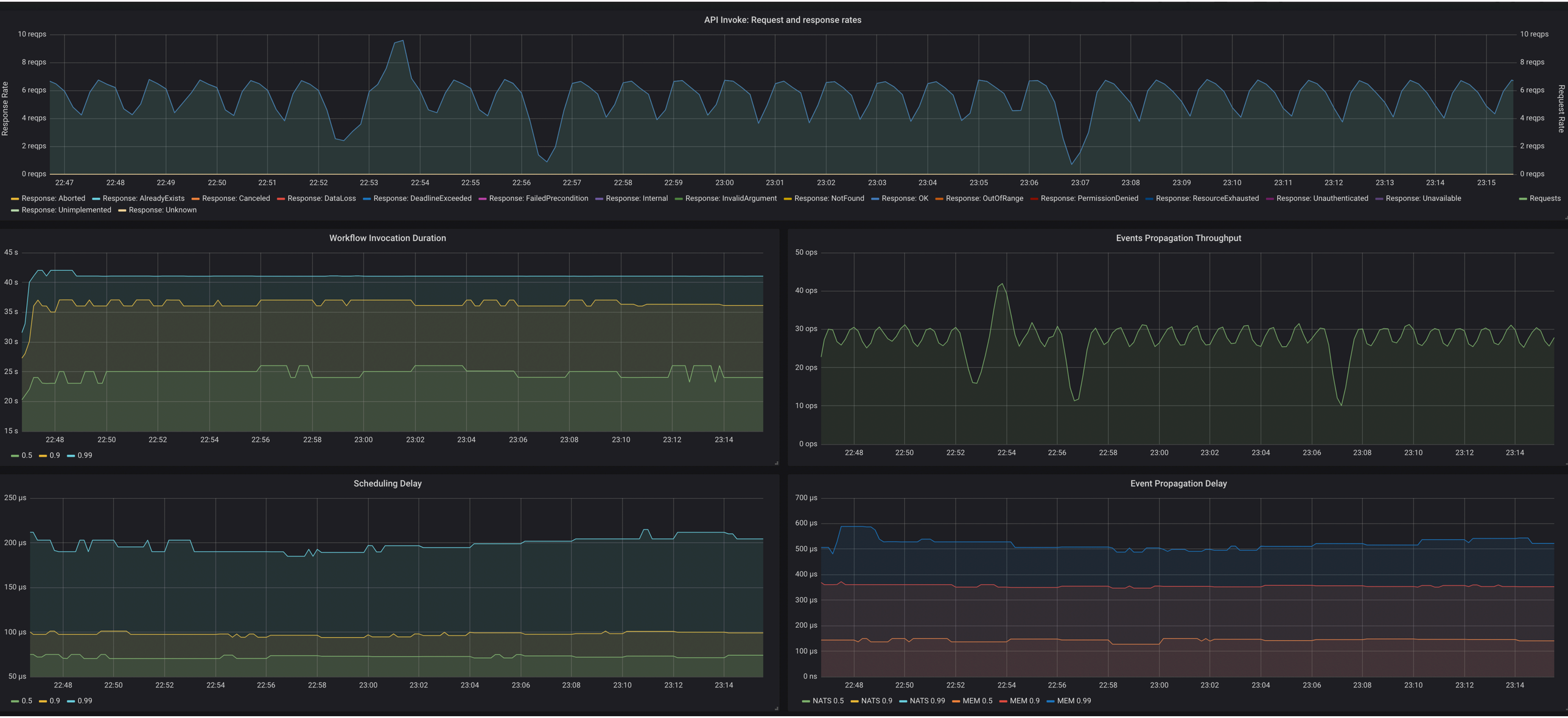


Task Run

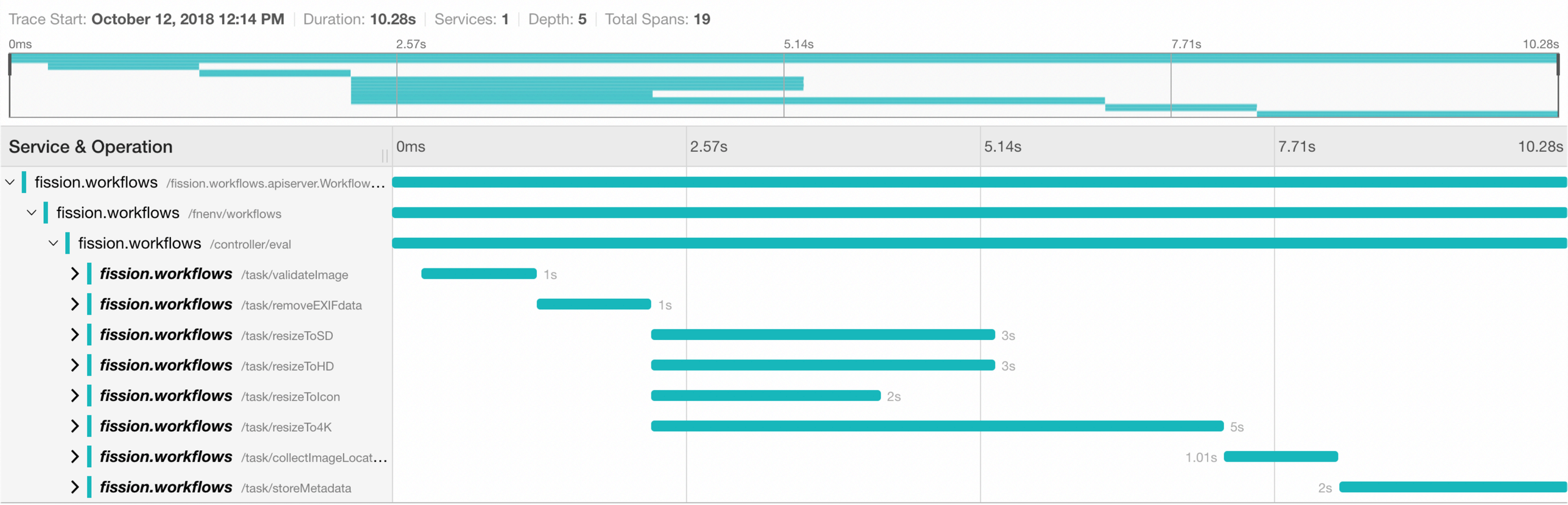
SWL: dynamic workflow support



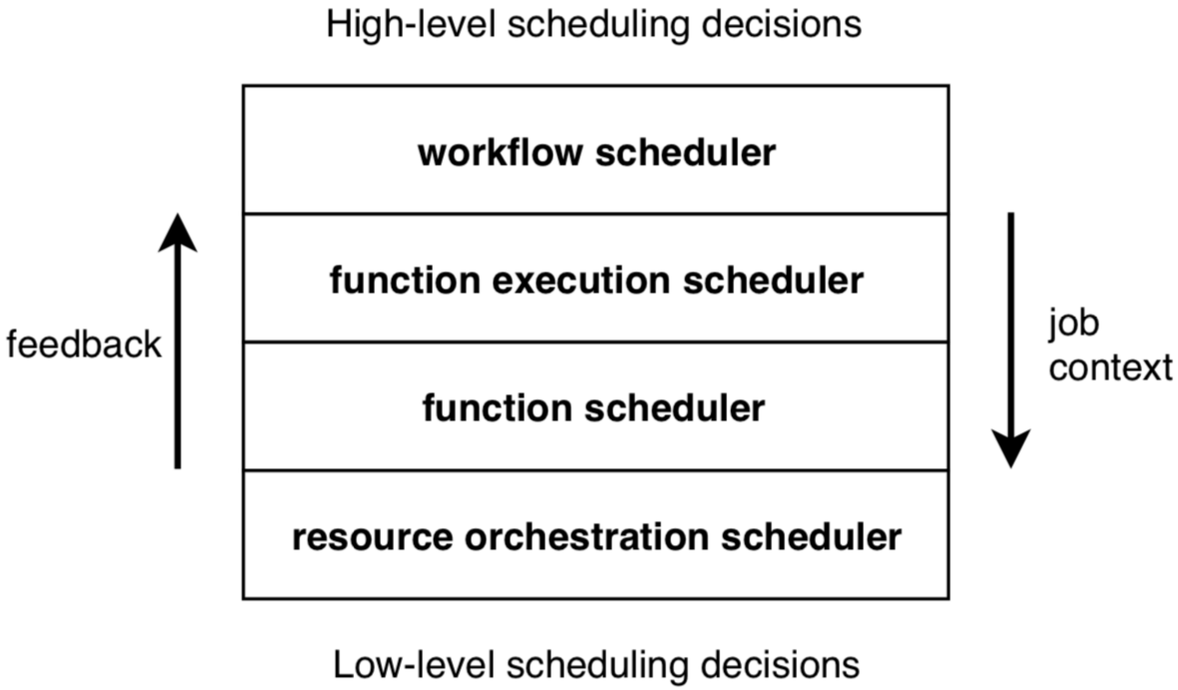
Fission Workflows monitoring support

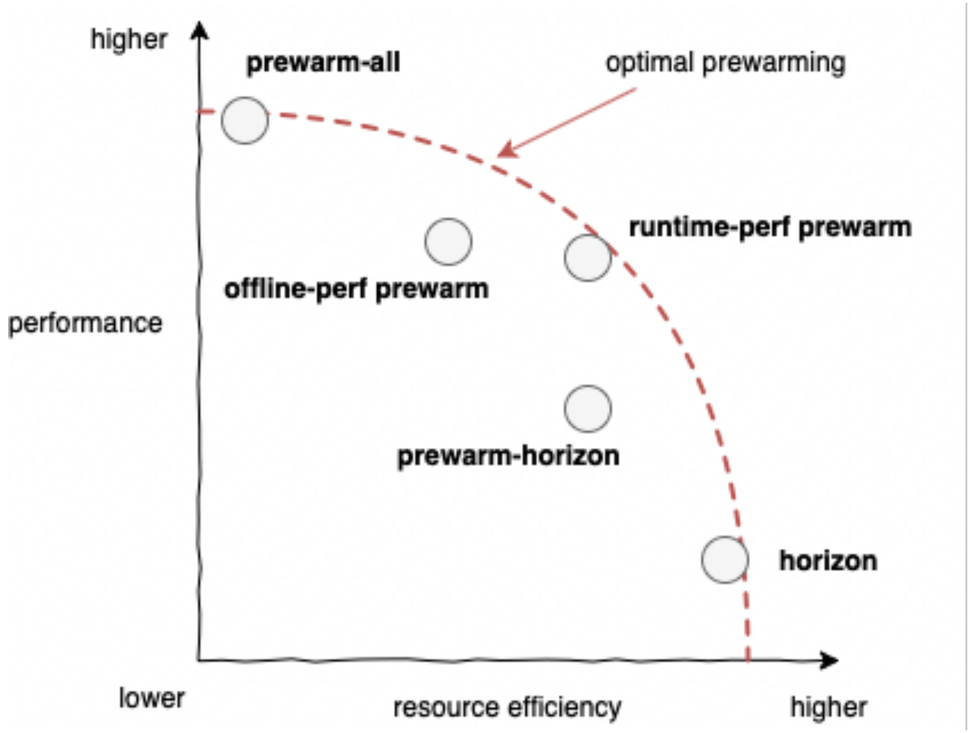


Fission Workflows tracing support



Serverless scheduling architecture and policies

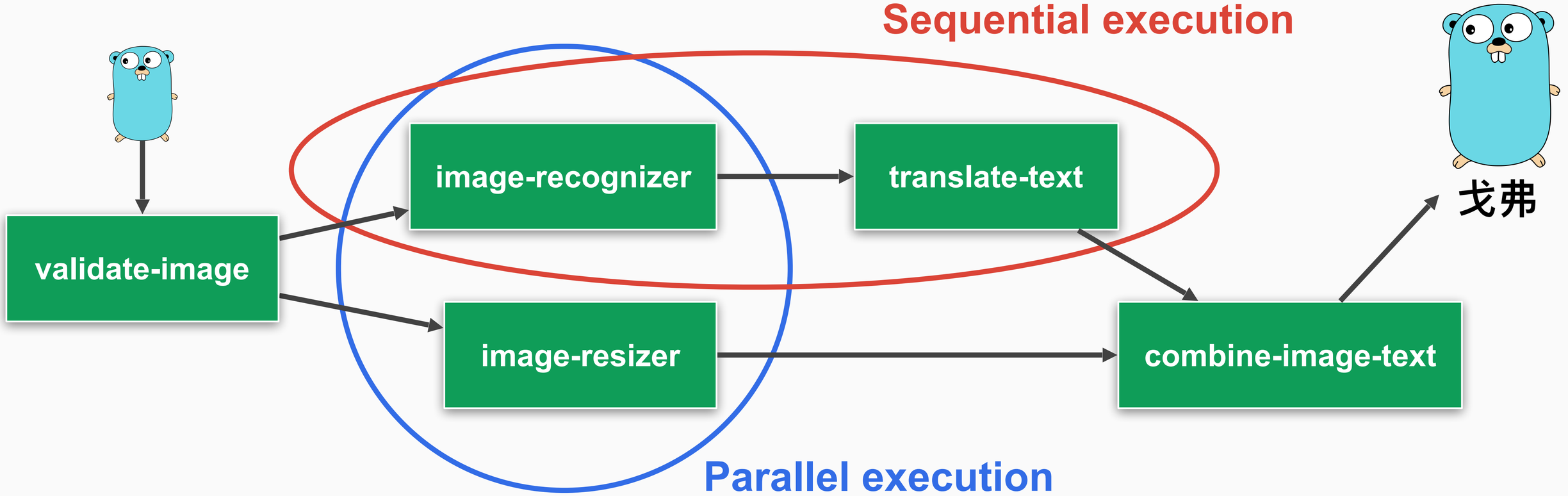




fission workflows

Function composition...

- Connect existing functions into complex function compositions
- Workflow engine takes care of the plumbing and provides fully monitorable, fault-tolerant function compositions with low overhead.



Why serverless computing?

Operational Model

Minimal operational logic

“Infinite” autoscaling

Built-in tooling: monitoring, tracing, health checking, etc.

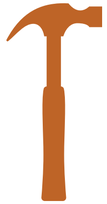
Cost Model

Pay based on usage

No upfront/periodic costs

Granular billing

Development Model

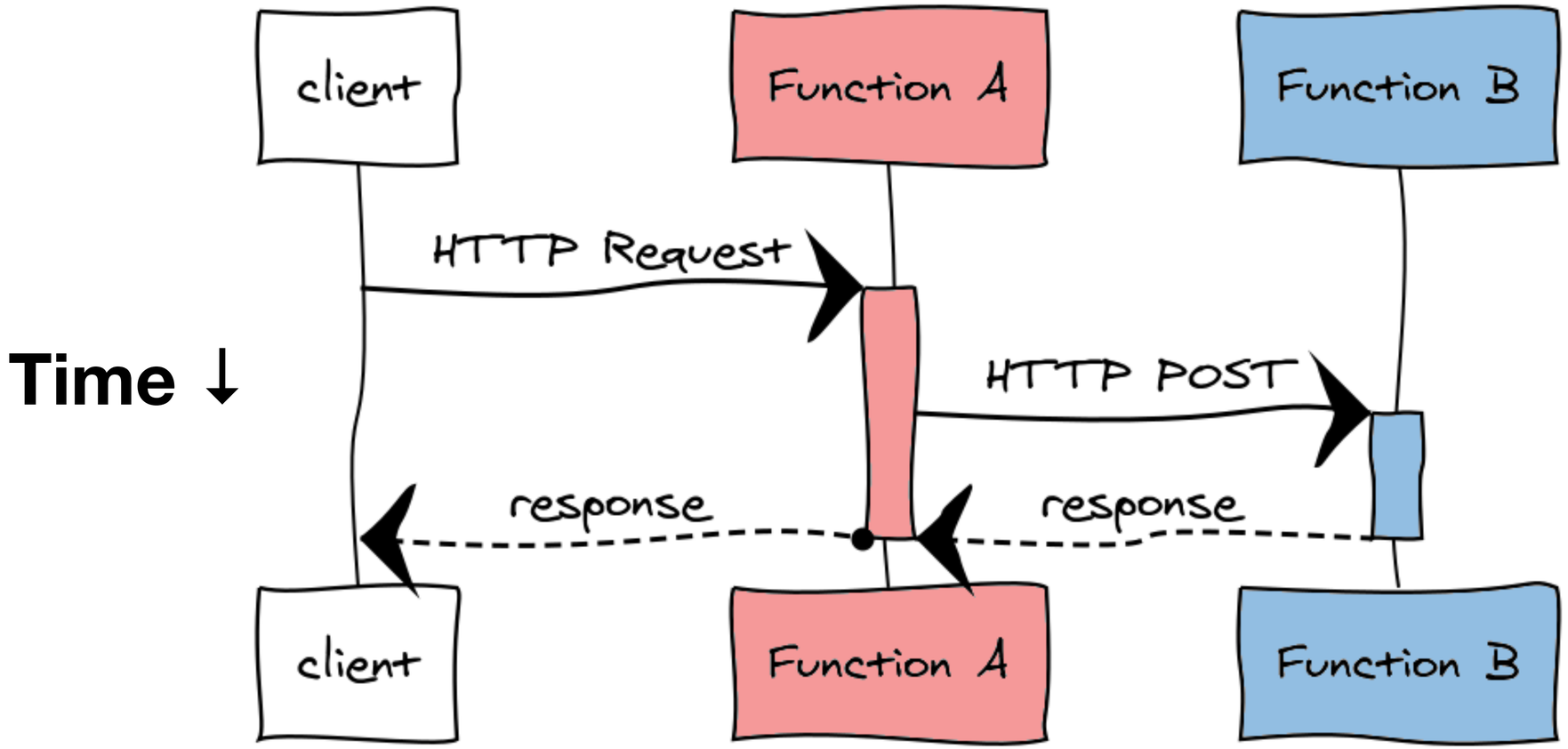


High-level abstractions

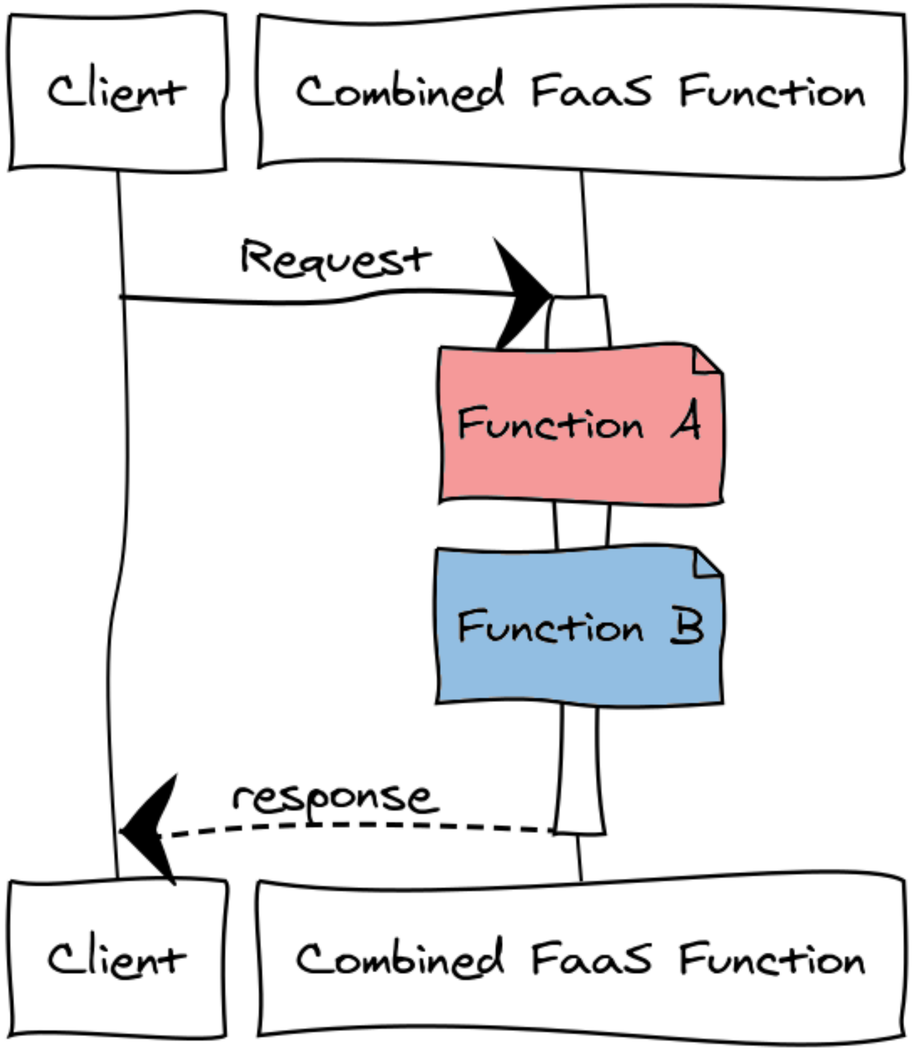
Pre-provided integrations

Language-agnostic

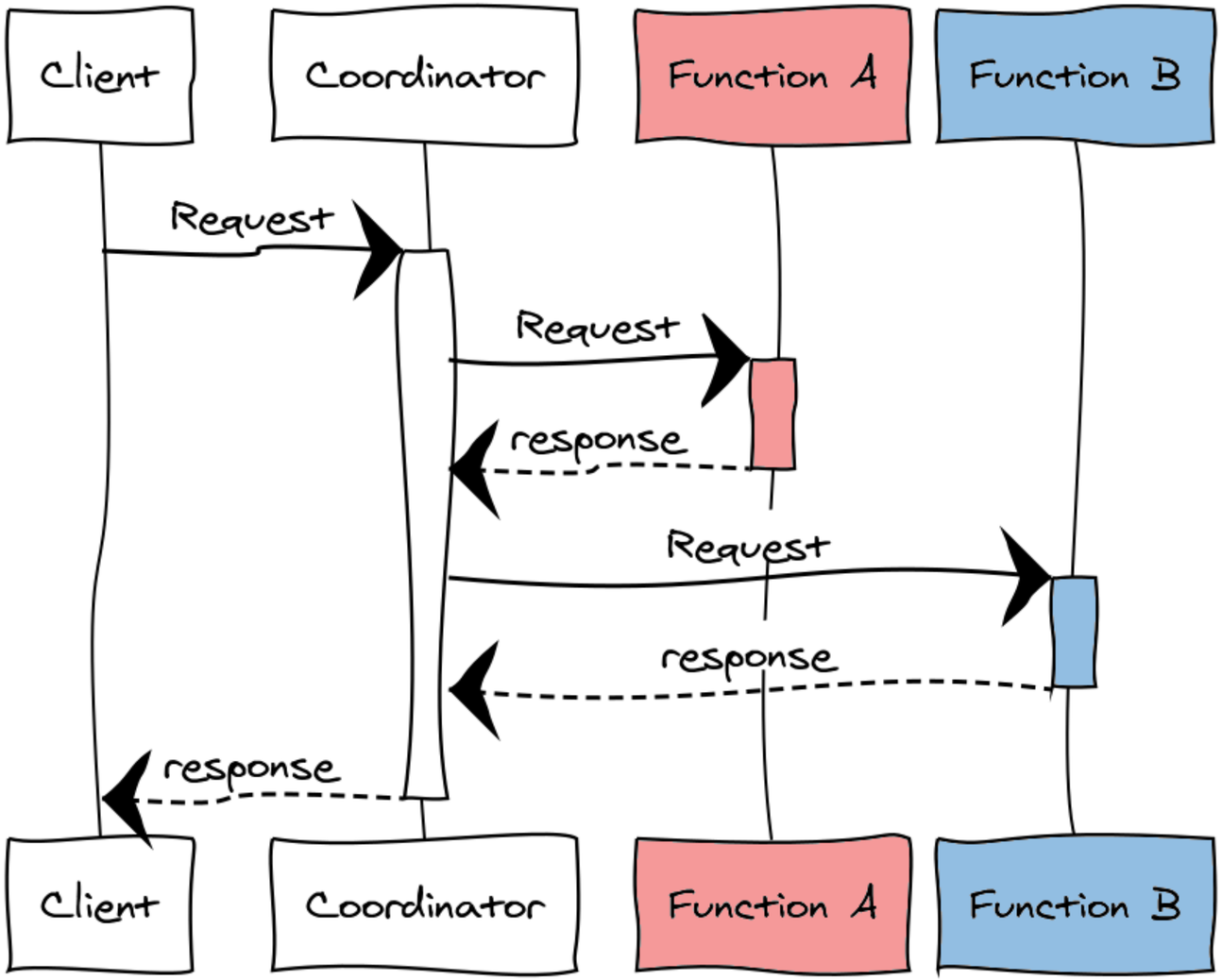
Direct composition



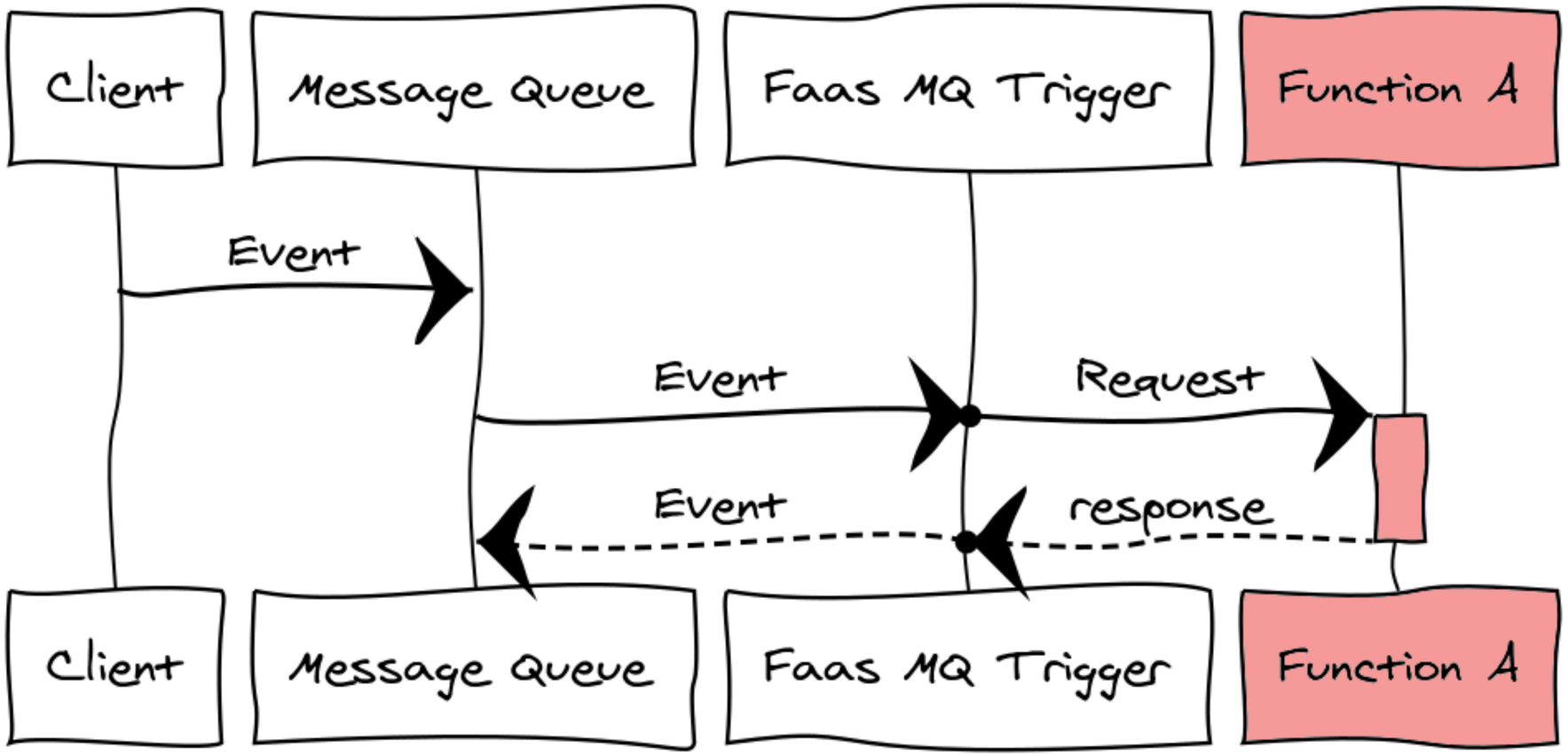
Compiled composition



Coordinator-based composition

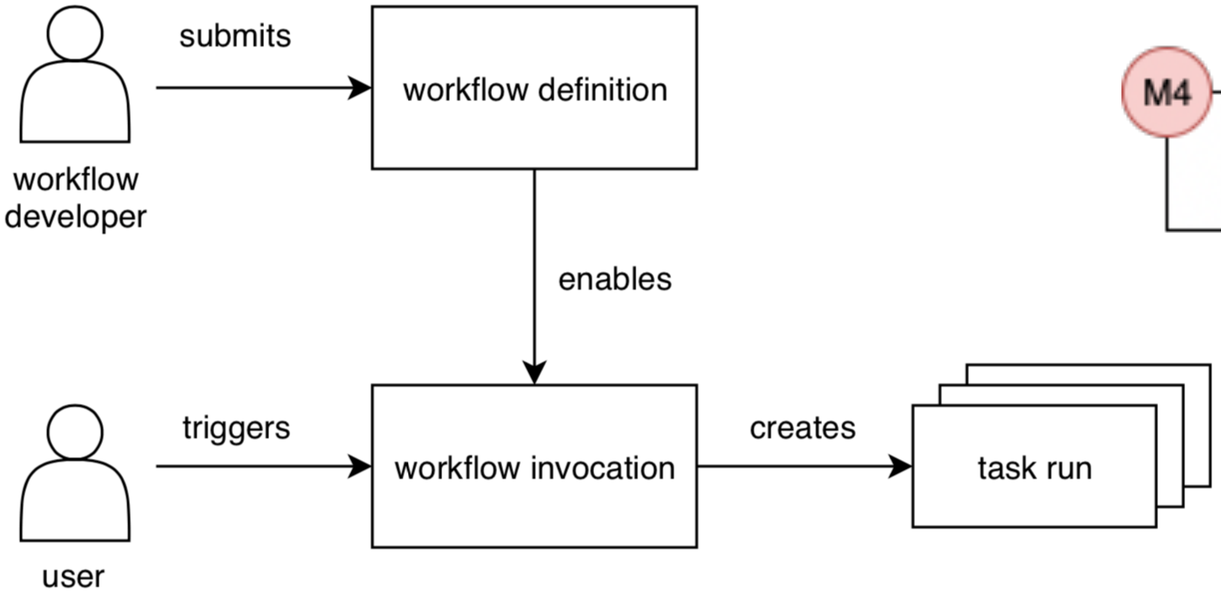


Event-driven composition

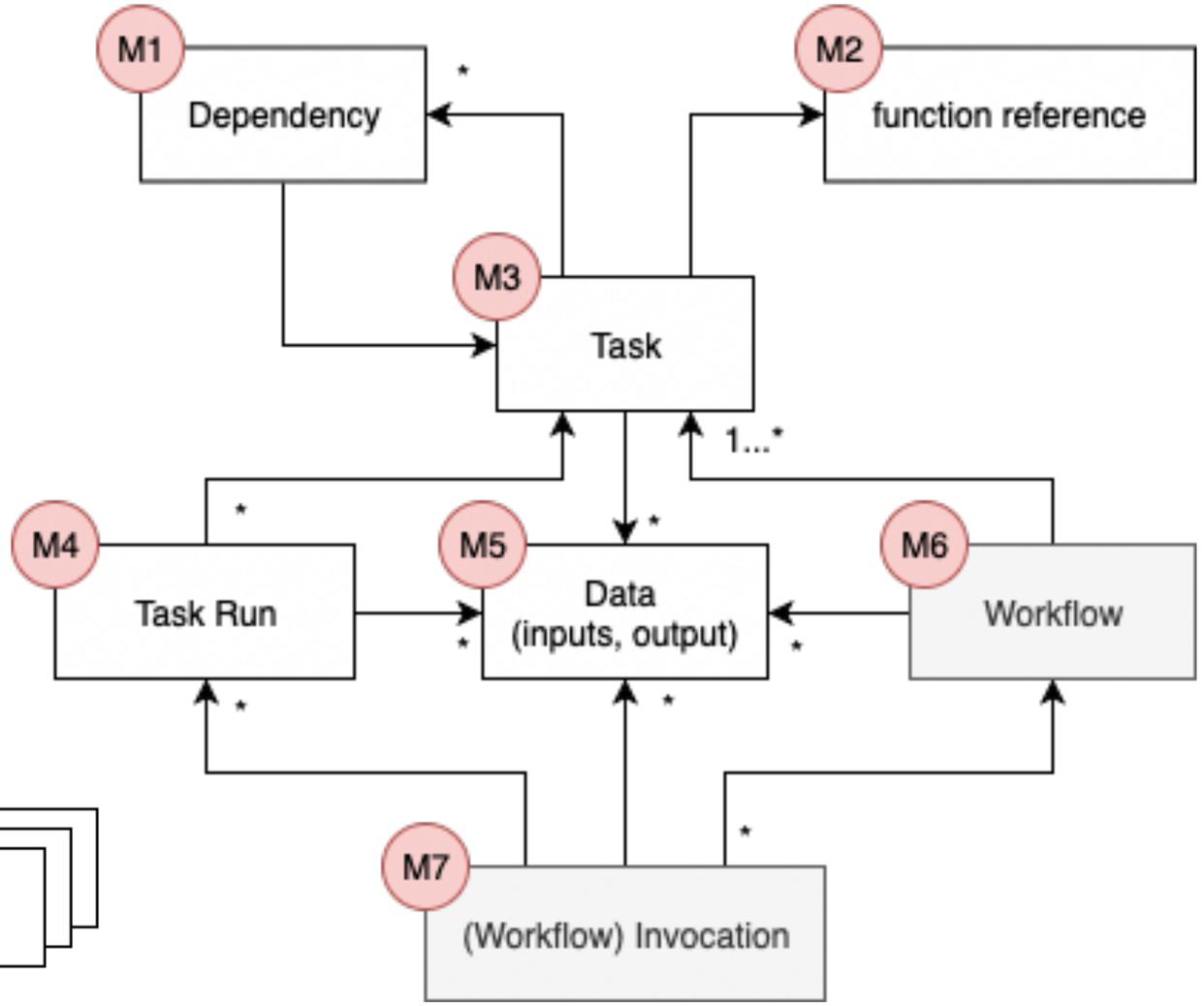


Serverless Workflow Language (SWL)

- Execution-level workflow language
- Minimal constructs
- Supports complex control flows



SWL execution model



SWL data model

SWL-YAML: a reference implementation of SWL

```
1  apiVersion: 1
2  output: WhaleWithFortune
3  tasks:
4    GenerateFortune:
5      run: fortune
6
7    WhaleWithFortune:
8      run: whalesay
9      inputs:
10       body: "{ output('GenerateFortune') }"
11     requires:
12     - GenerateFortune
```

- Uses a declarative format
- Follows syntax of state-of-the-art WMSs
- Supports JavaScript-like expressions