Understanding Open Source Serverless Platforms: Design Considerations and Performance

Junfeng Li, Sameer G. Kulkarni,
K. K. Ramakrishnan, Dan Li

Open Source Serverless Platforms

nuclio

OPENFAAS

Kubeless

fission

fn

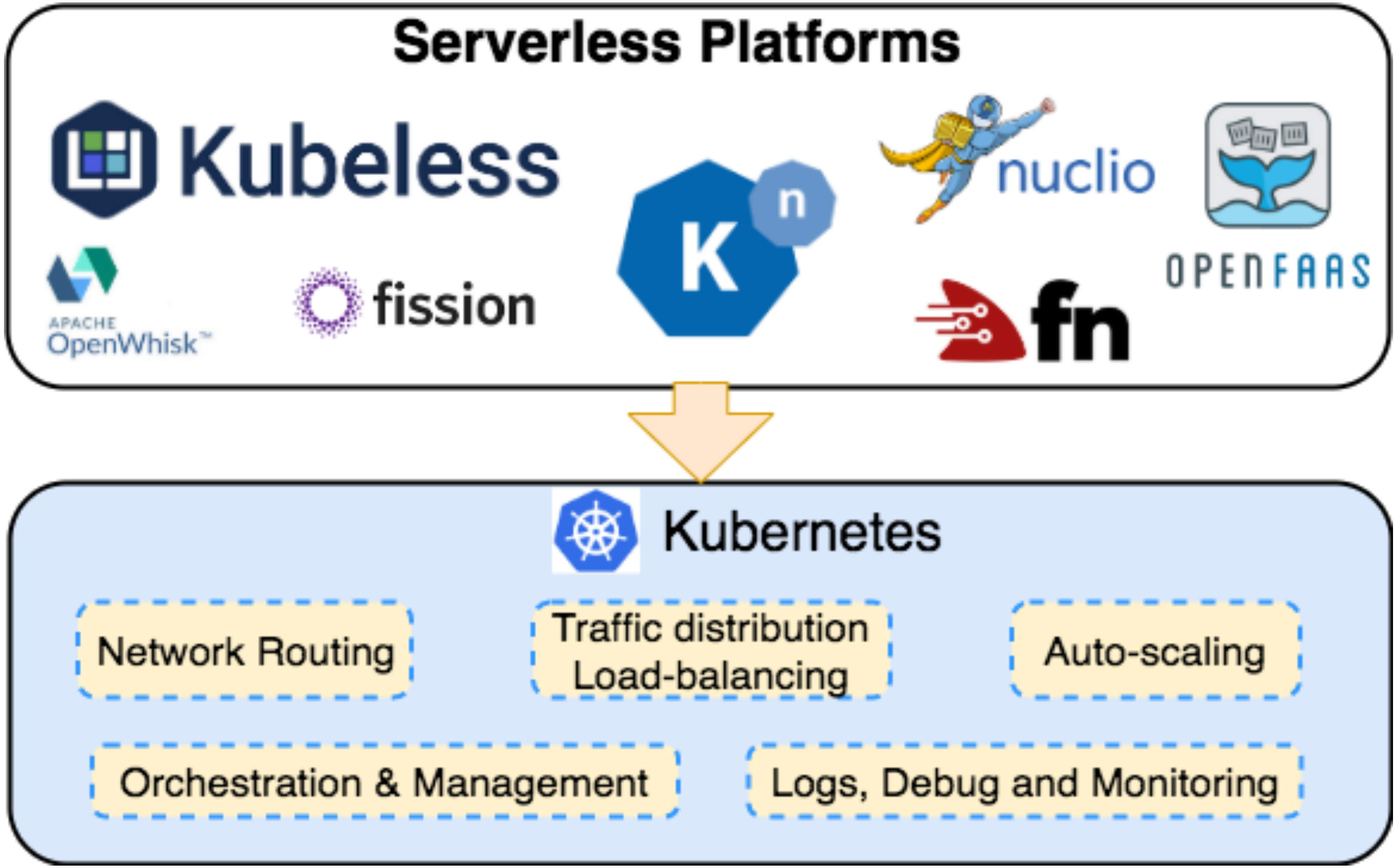
Spring Cloud Function

APACHE OpenWhisk

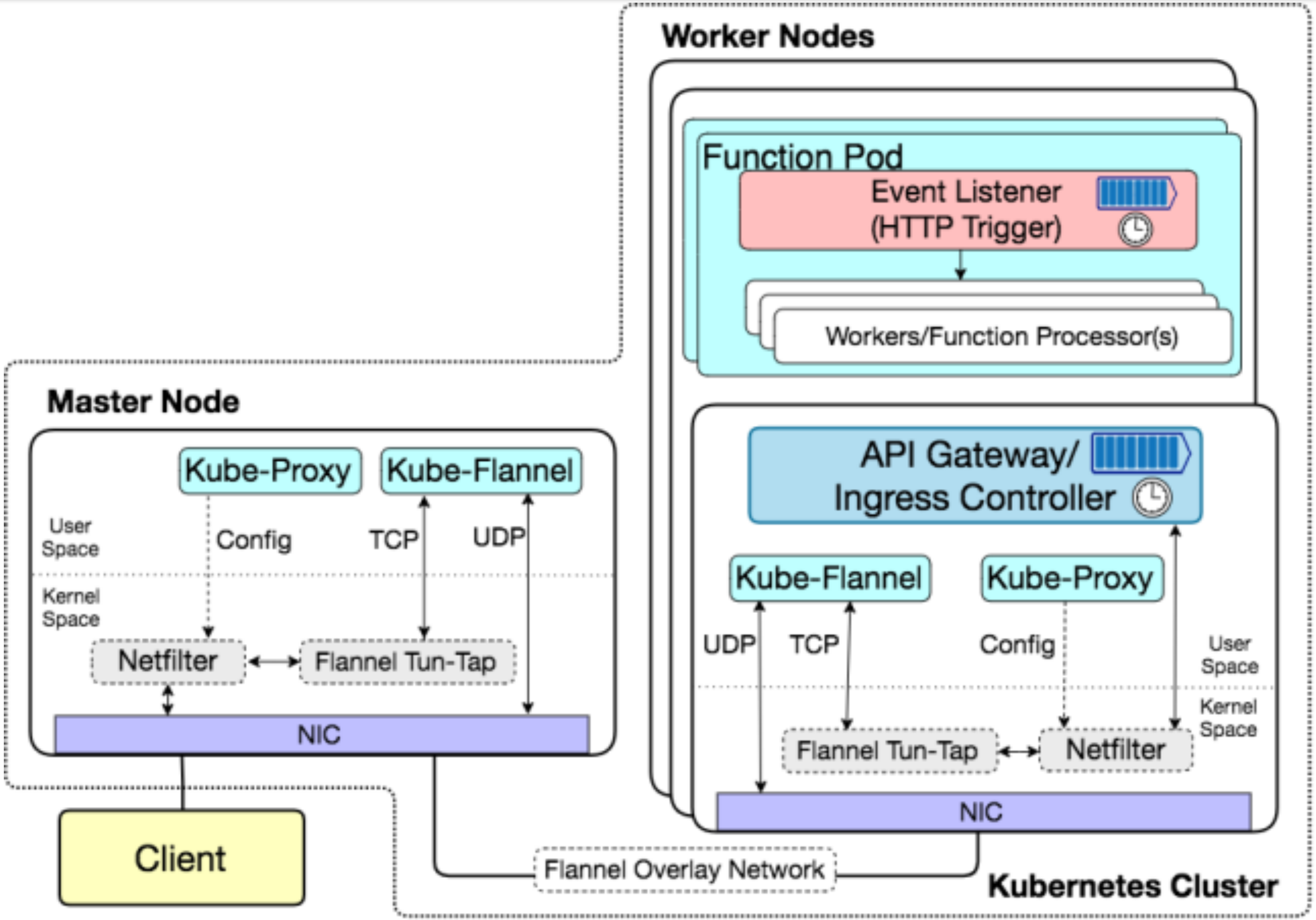
Motivation and Goals

- ❖ To develop an understanding on the open source serverless platforms:
  - ➢ Do measurements to understand the impact of key configuration parameters of different components (platform, gateway, controller and function)
- ❖ Evaluate and compare the performance of open source serverless platforms:
  - ➢ Different workloads
  - ➢ Different auto-scaling modes

Dependence on Kubernetes



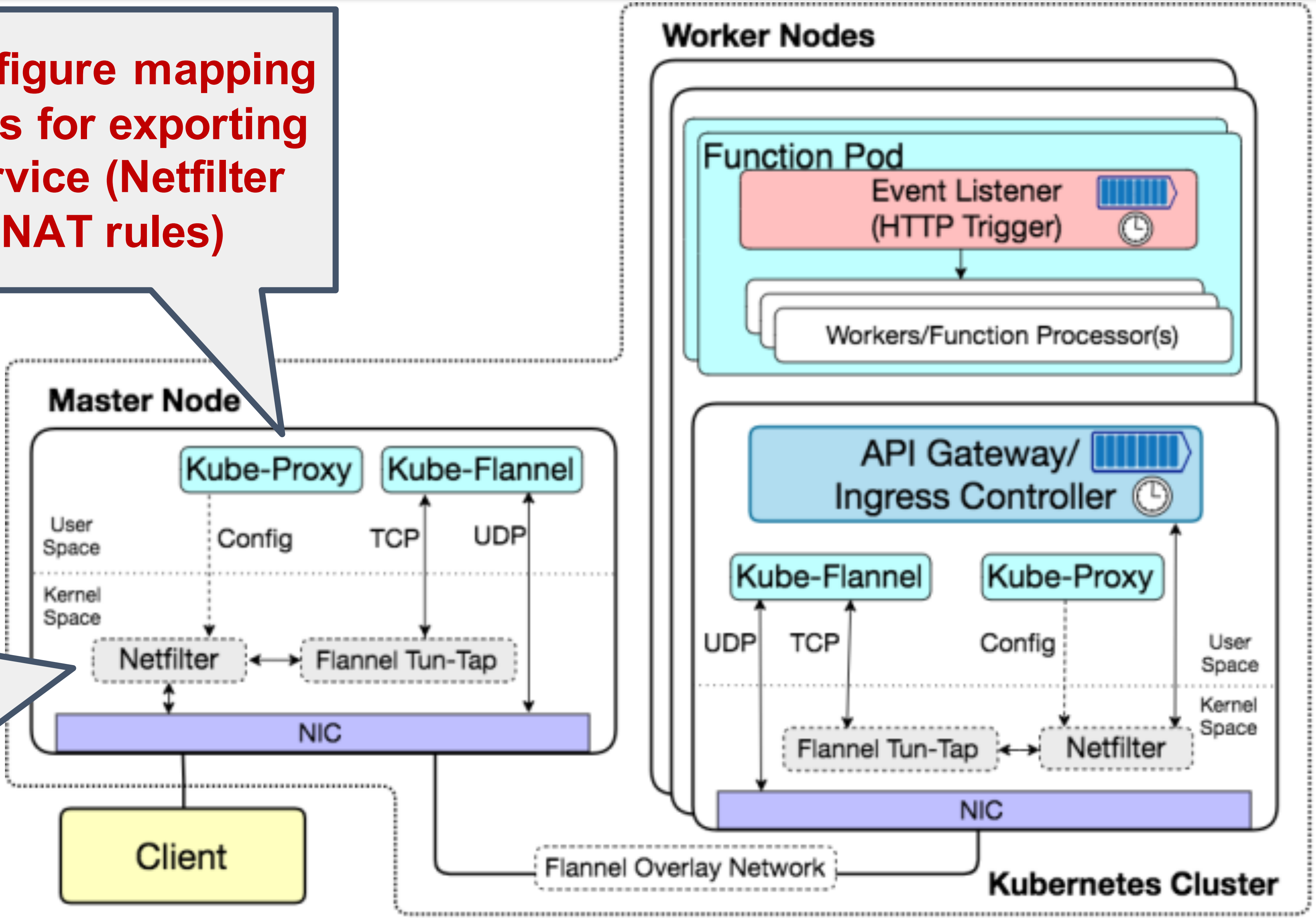
Service Exporting and Routing



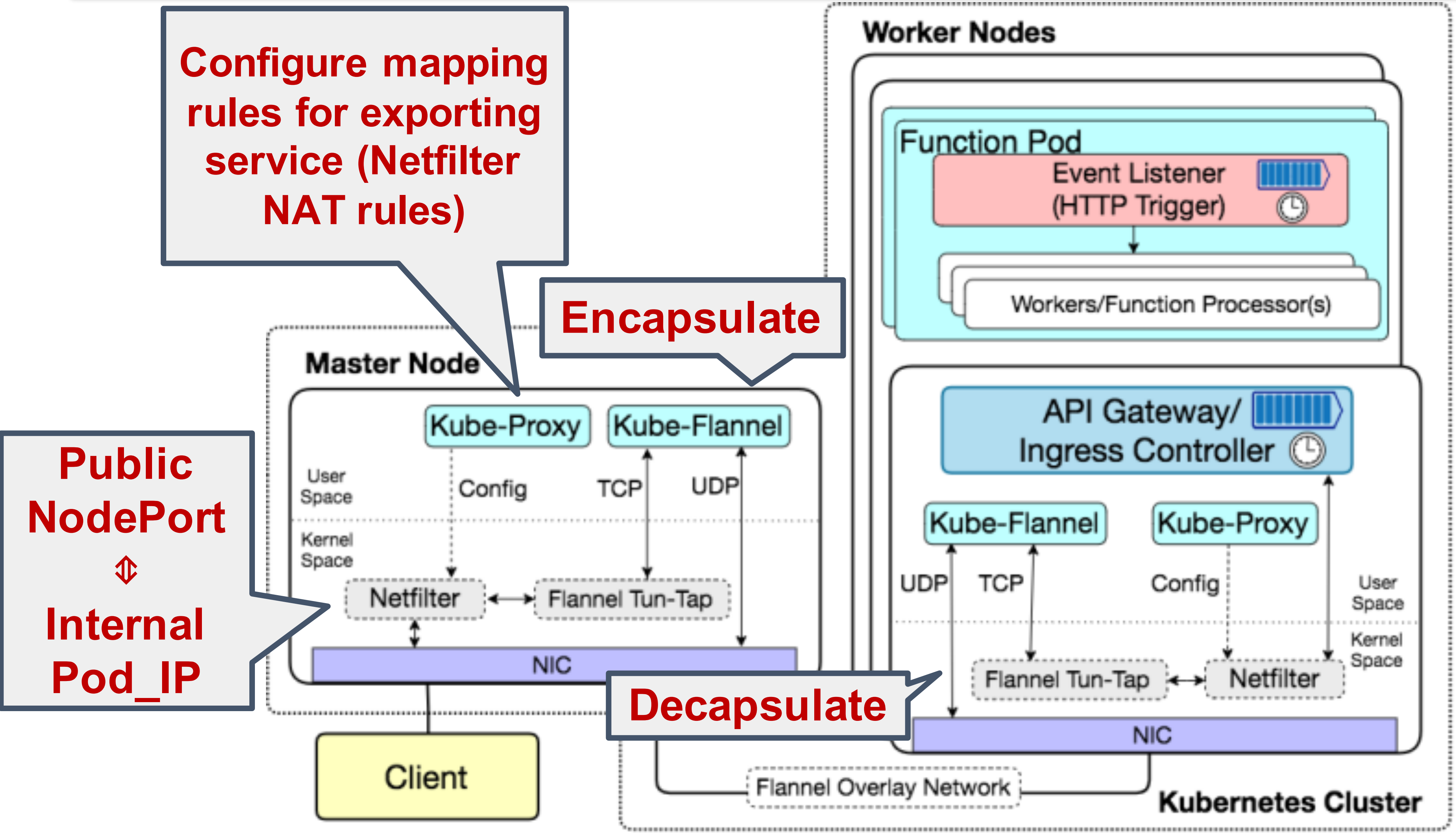
Service Exporting and Routing

Configure mapping rules for exporting service (Netfilter NAT rules)

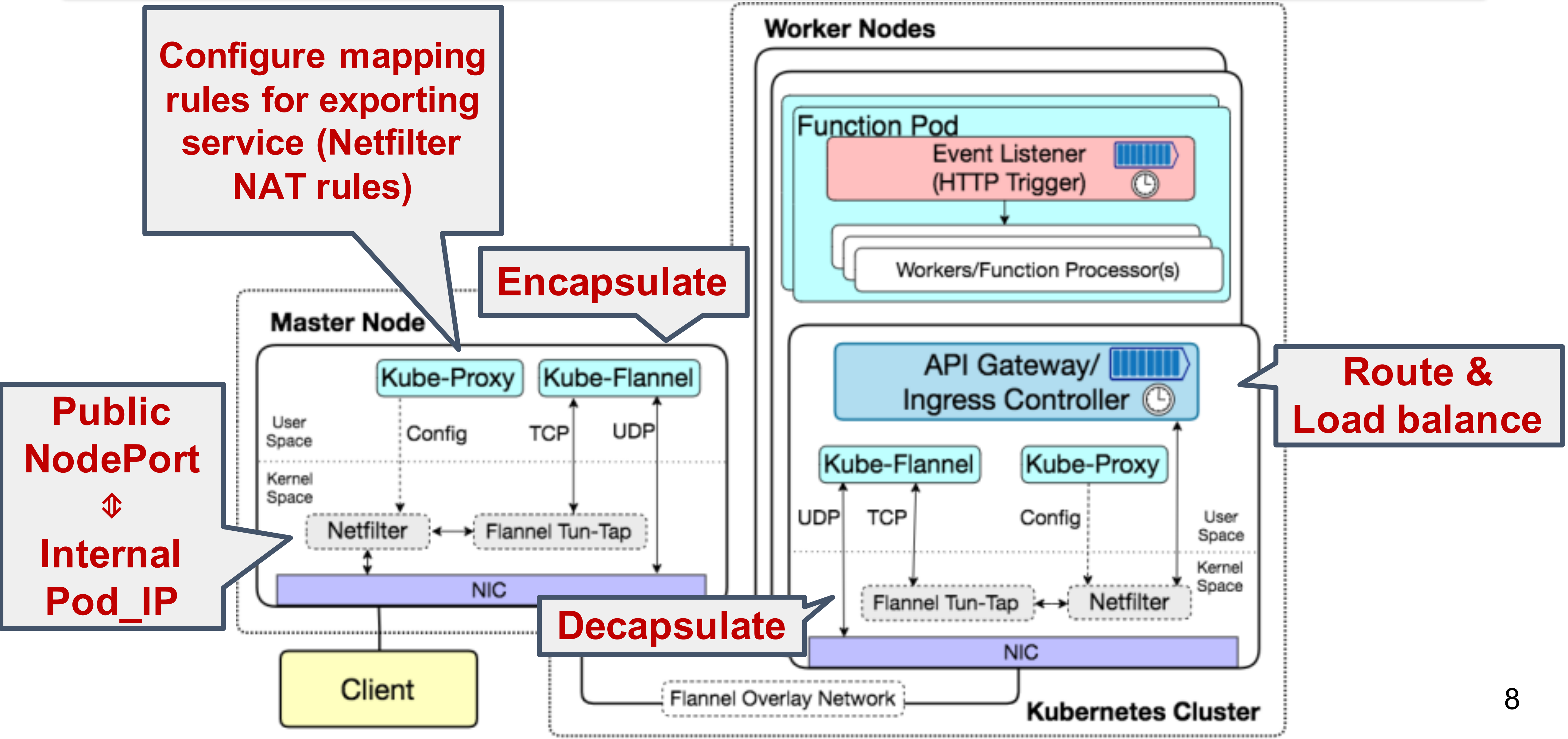
Public NodePort ⇕ Internal Pod_IP



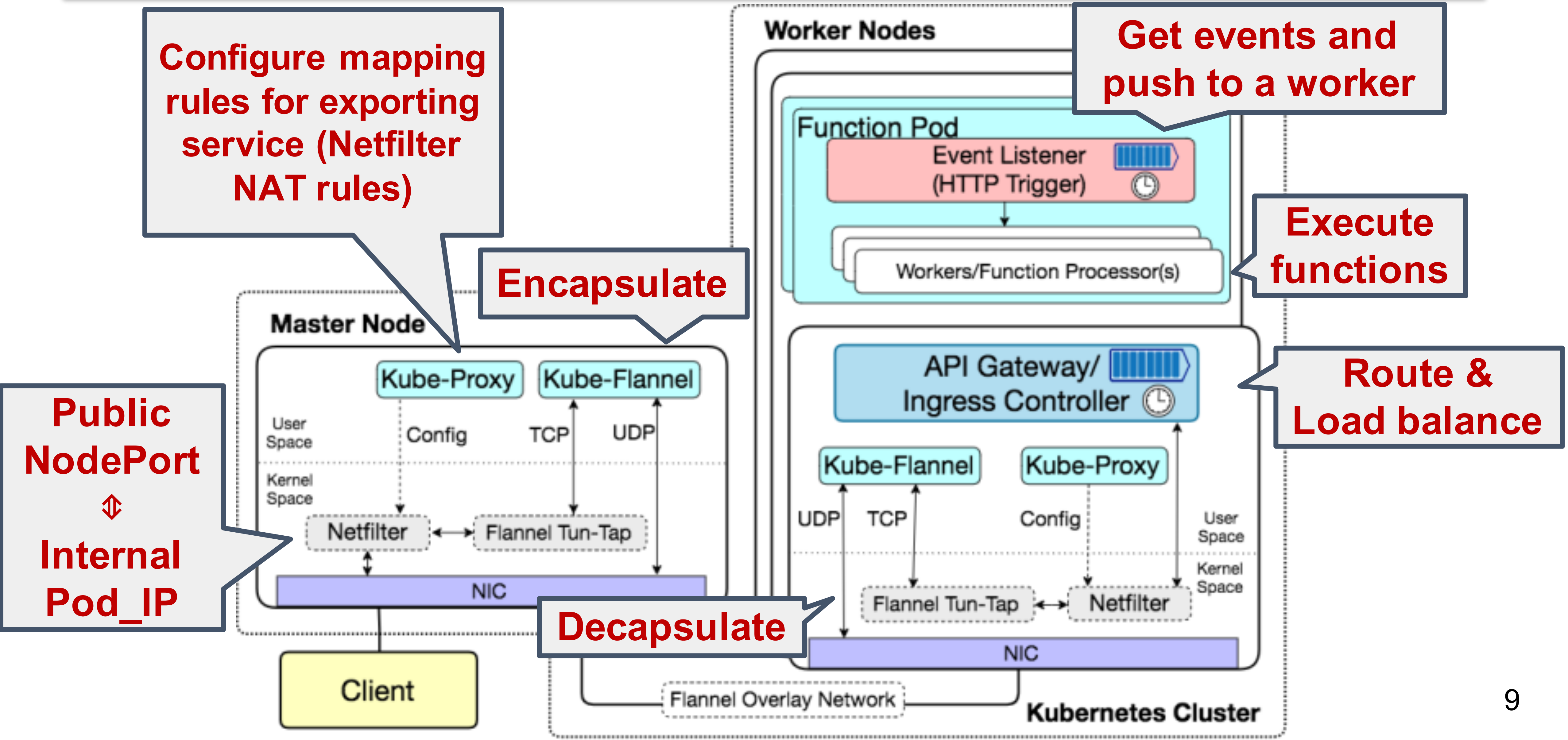
Service Exporting and Routing



Service Exporting and Routing



Service Exporting and Routing



Motivation and Goals

- To develop an understanding on the open source serverless platforms:
  - Do measurements to understand the impact of key configuration parameters of different components (gateway, controller and function)
- Evaluate and compare the performance of open source

nuclio

v1.1.16

OPENFAAS

Gateway: v0.17.0
Faas-netes: v0.8.6
Faas-cli: v0.9.2

v0.8

Kubeless

v1.0.4

Experiment Setup

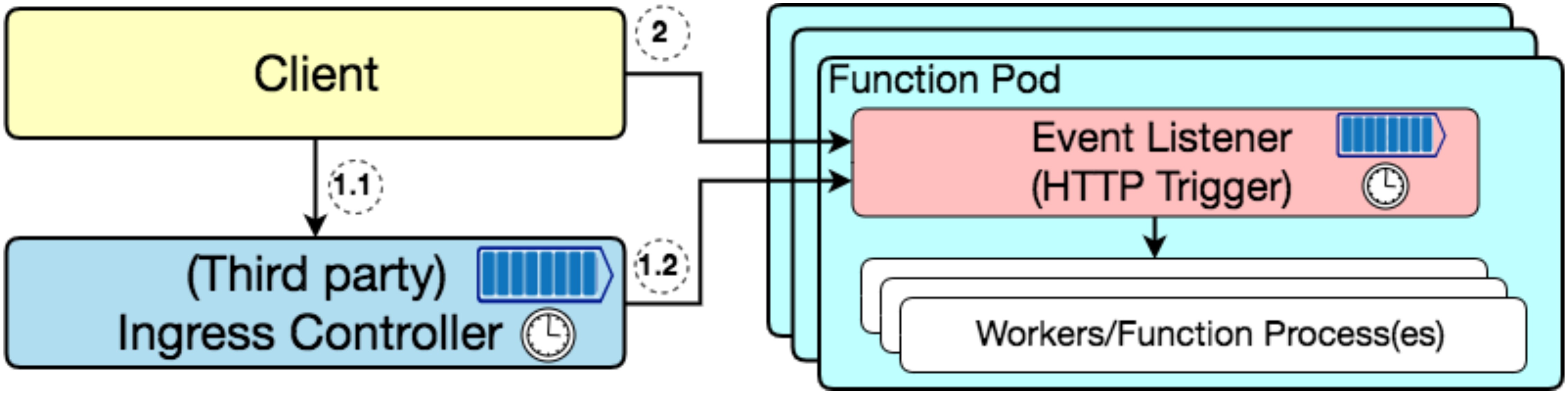
- Topology: Kubernetes cluster (1 master, 2 workers) on CloudLab[1]
  - Hardware: Intel Xeon E5-2640 v4 @ 20 Hyperthread cores.
  - Operating System: Ubuntu 16.04.1 LTS
  - Kubernetes v1.16.1, Docker v18.09.2

- Functions and Workload:
  - Python ‘Hello-world’ function
  - Python ‘HTTP’ function
  - Workload Generator: wrk

[1] Duplyakin, Dmitry, et al. "The design and operation of CloudLab." 2019 USENIX Annual Technical Conference (USENIX ATC 19). 2019.

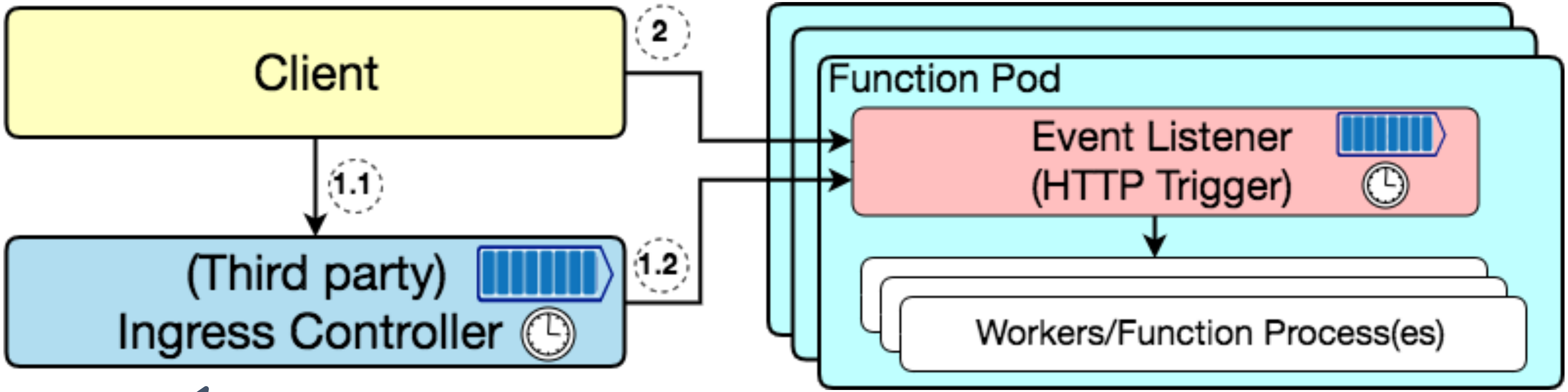
Nuclio

❖ Working model



Nuclio

❖ Working model

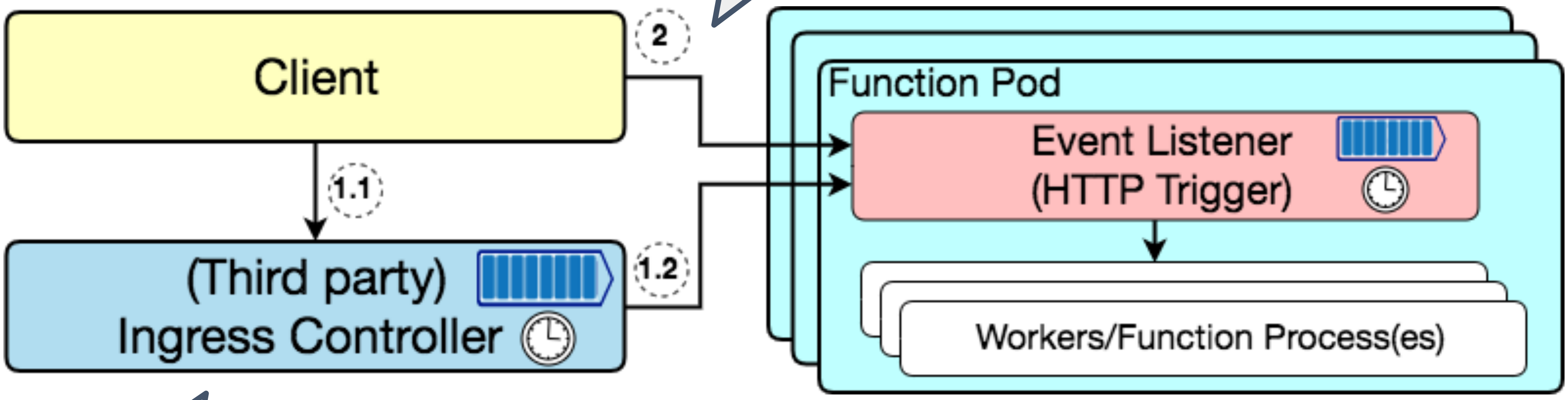


Invoke through IG

Nuclio

❖ Working model

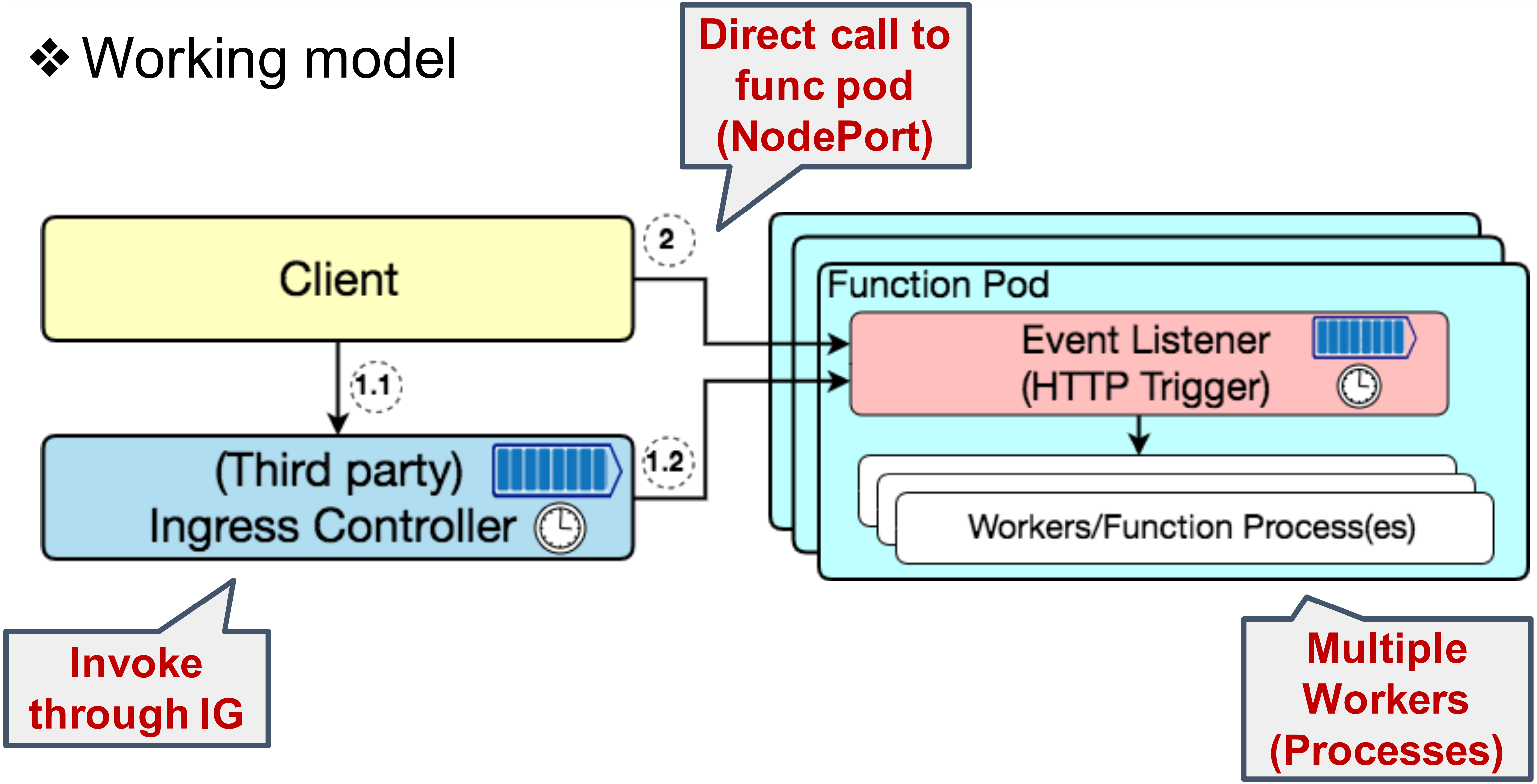
Direct call to func pod (NodePort)



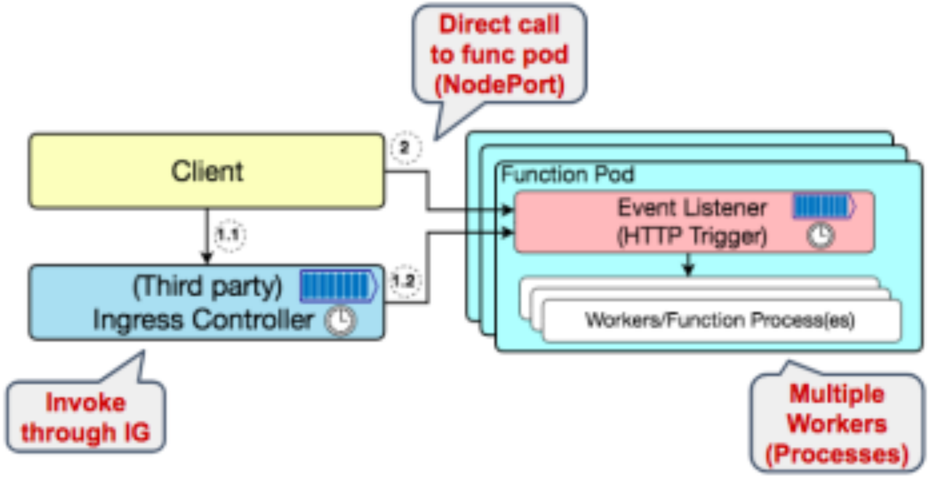
Invoke through IG

Nuclio

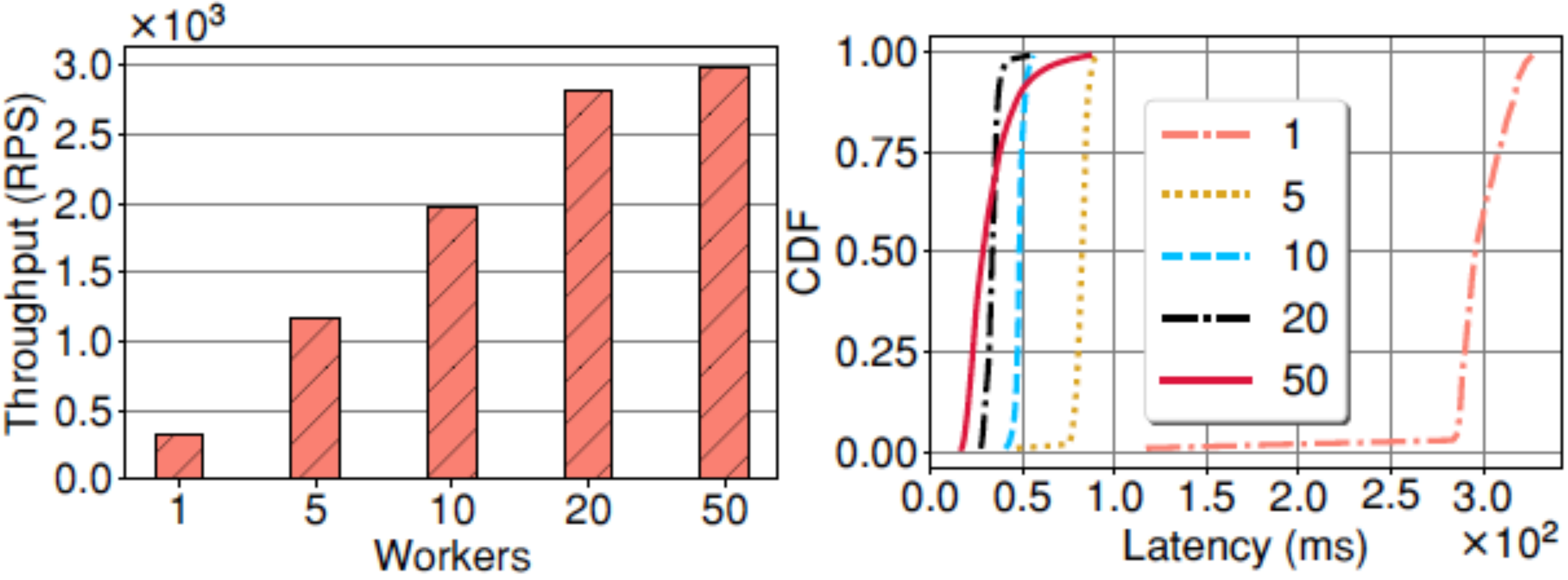
❖ Working model



Nuclio



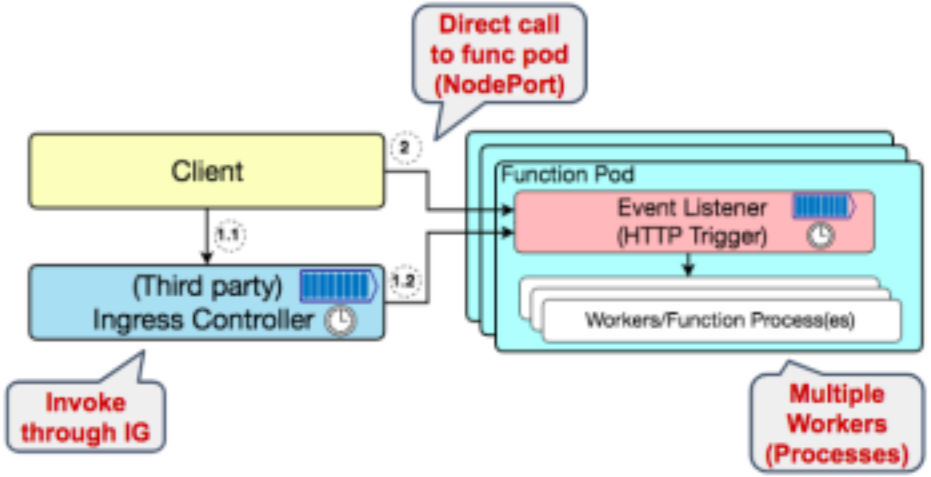
❖ Salient parameter: the number of workers within one pod



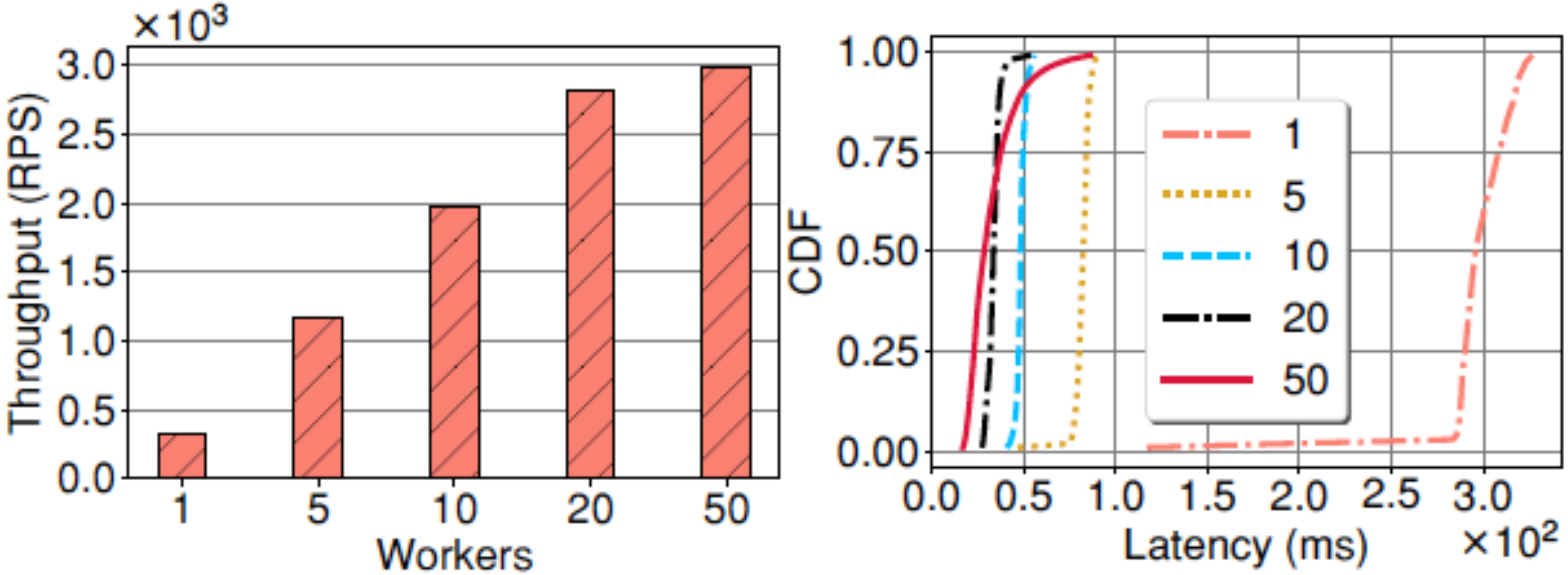
(a) Throughput in requests/second. (b) Latency in ms.

Nuclio

Performance increases as the number of workers increases.



❖ Salient parameter: the number of workers within one pod

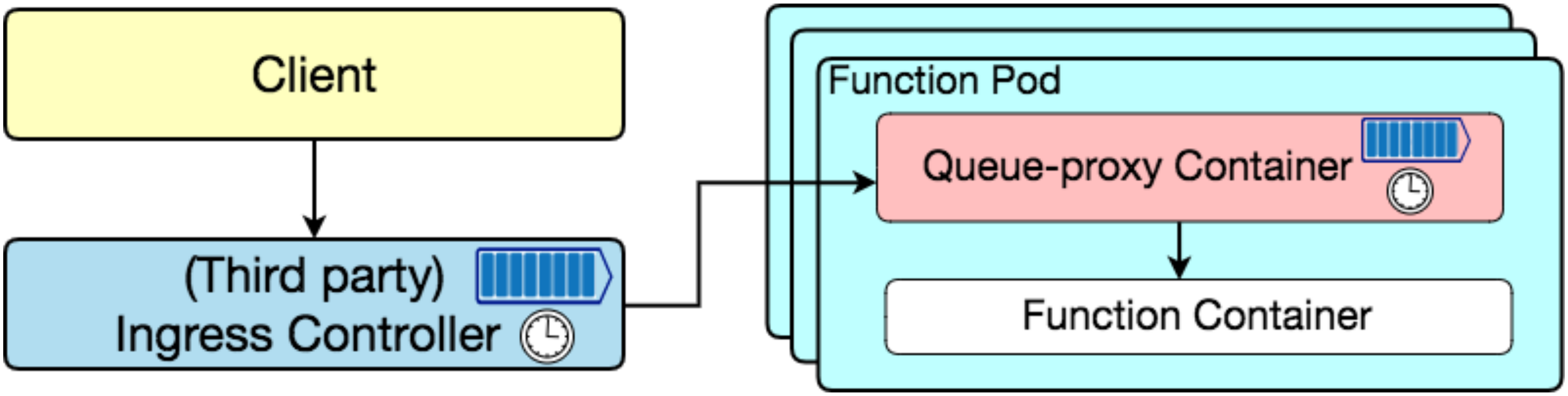


(a) Throughput in requests/second.

(b) Latency in ms.

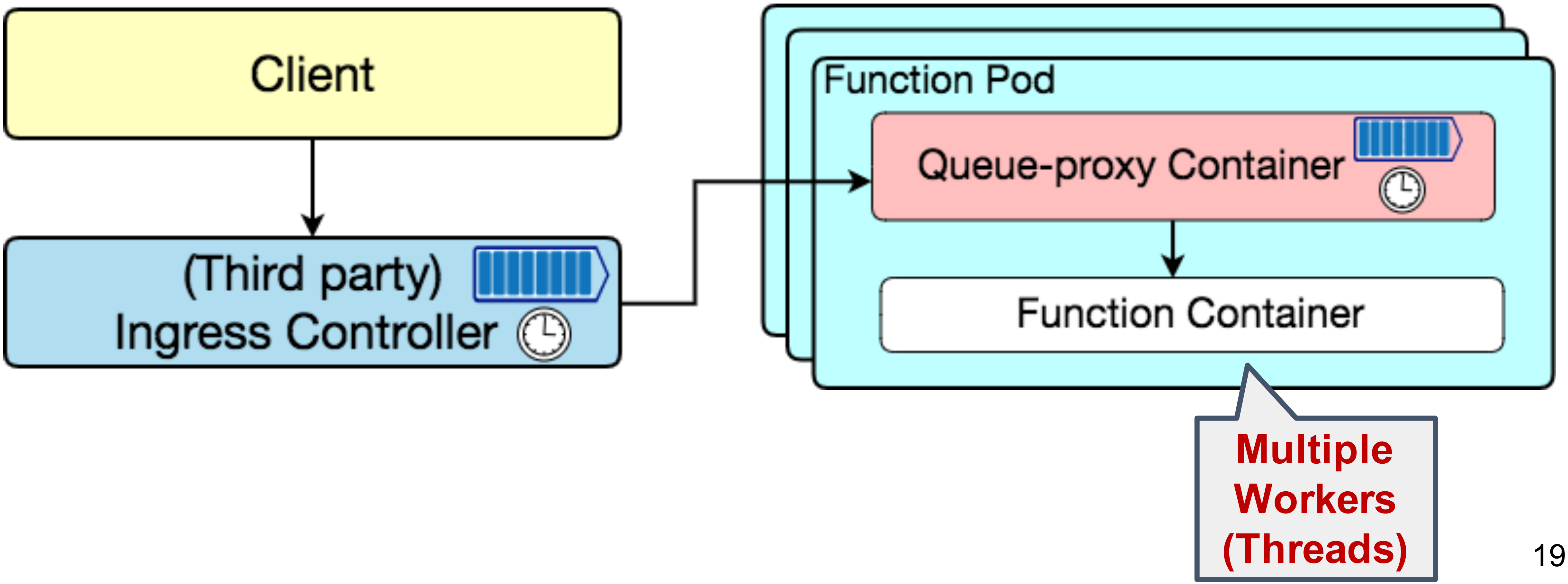
Knative

❖ Working model

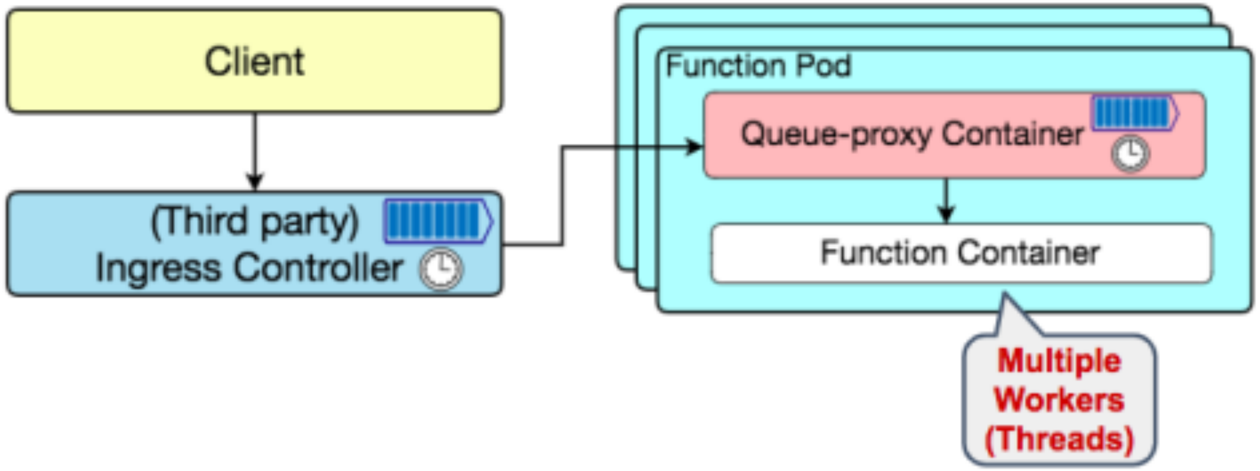


Knative

- ❖ Working model



Knative



❖ Salient parameter: the number of workers within one pod

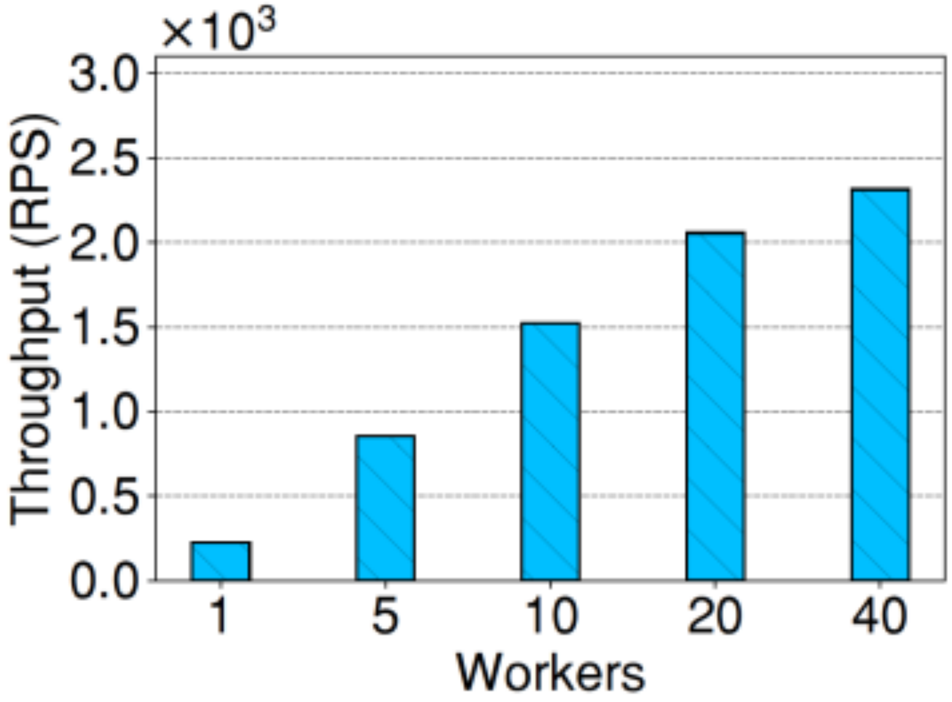


Fig. Throughput of Knative.

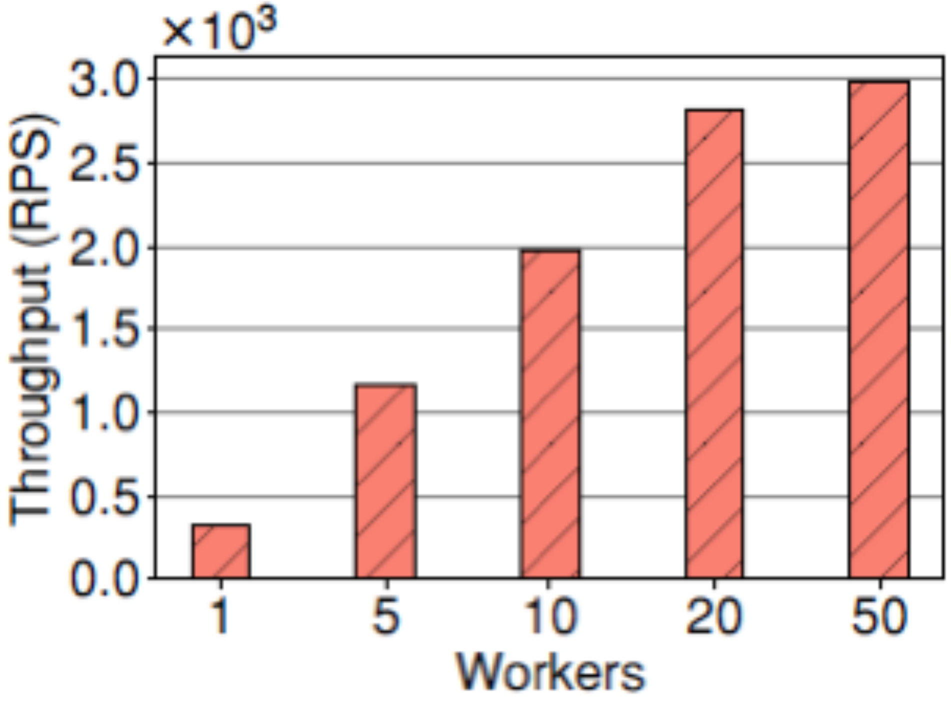
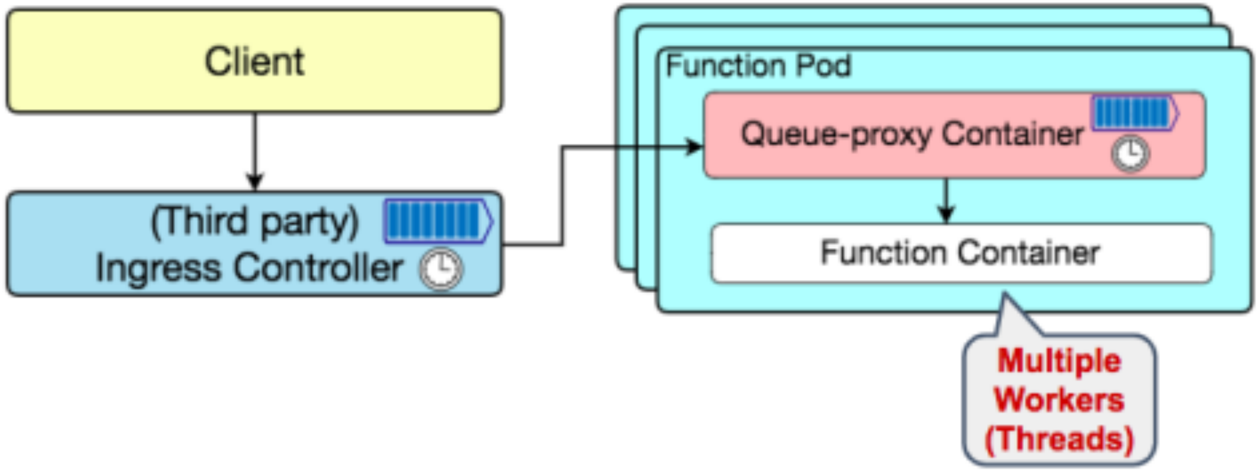


Fig. Throughput of Nuclio.

Knative



Performance improves, but relatively lower than Nuclio.

❖ Salient parameter: the number of workers within one pod

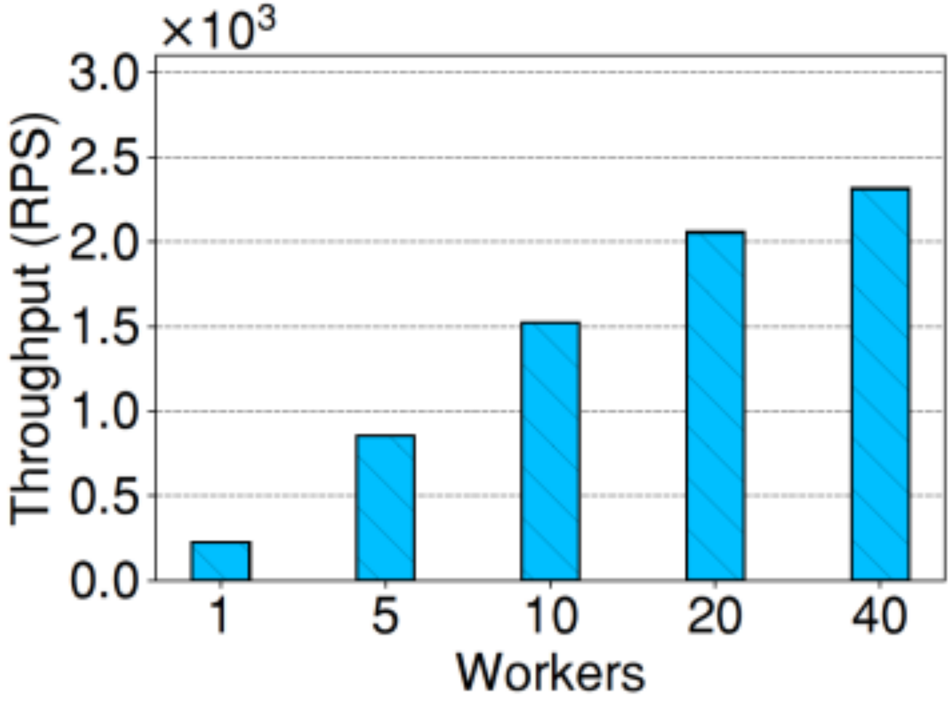


Fig. Throughput of Knative.

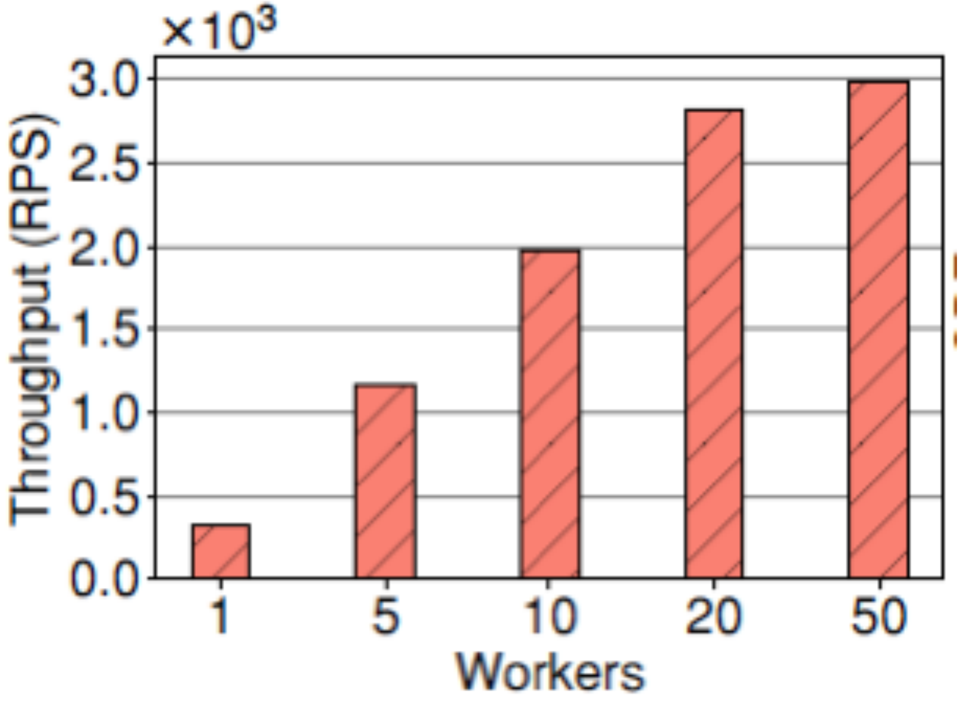
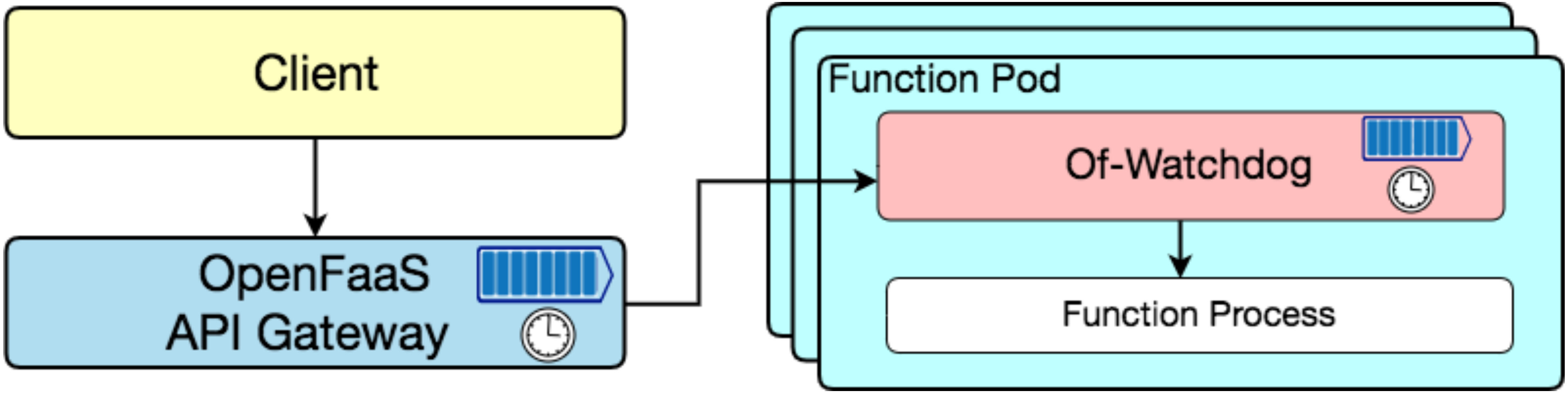


Fig. Throughput of Nuclio.

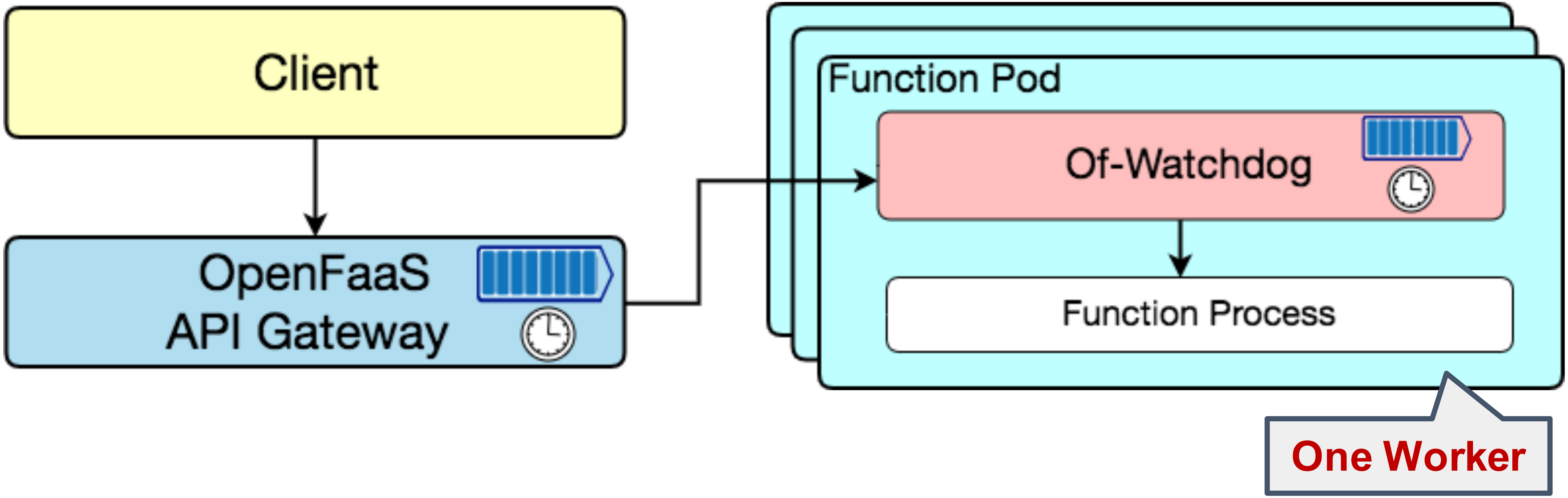
OpenFaaS

❖ Working model



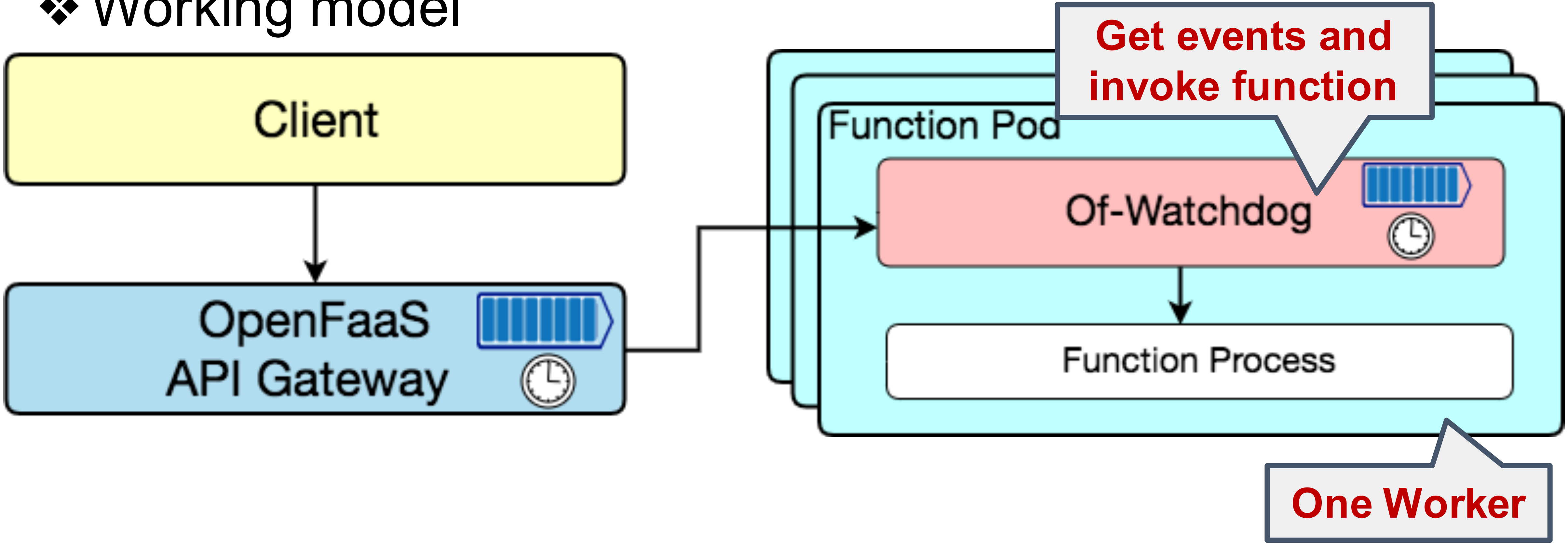
OpenFaaS

❖ Working model



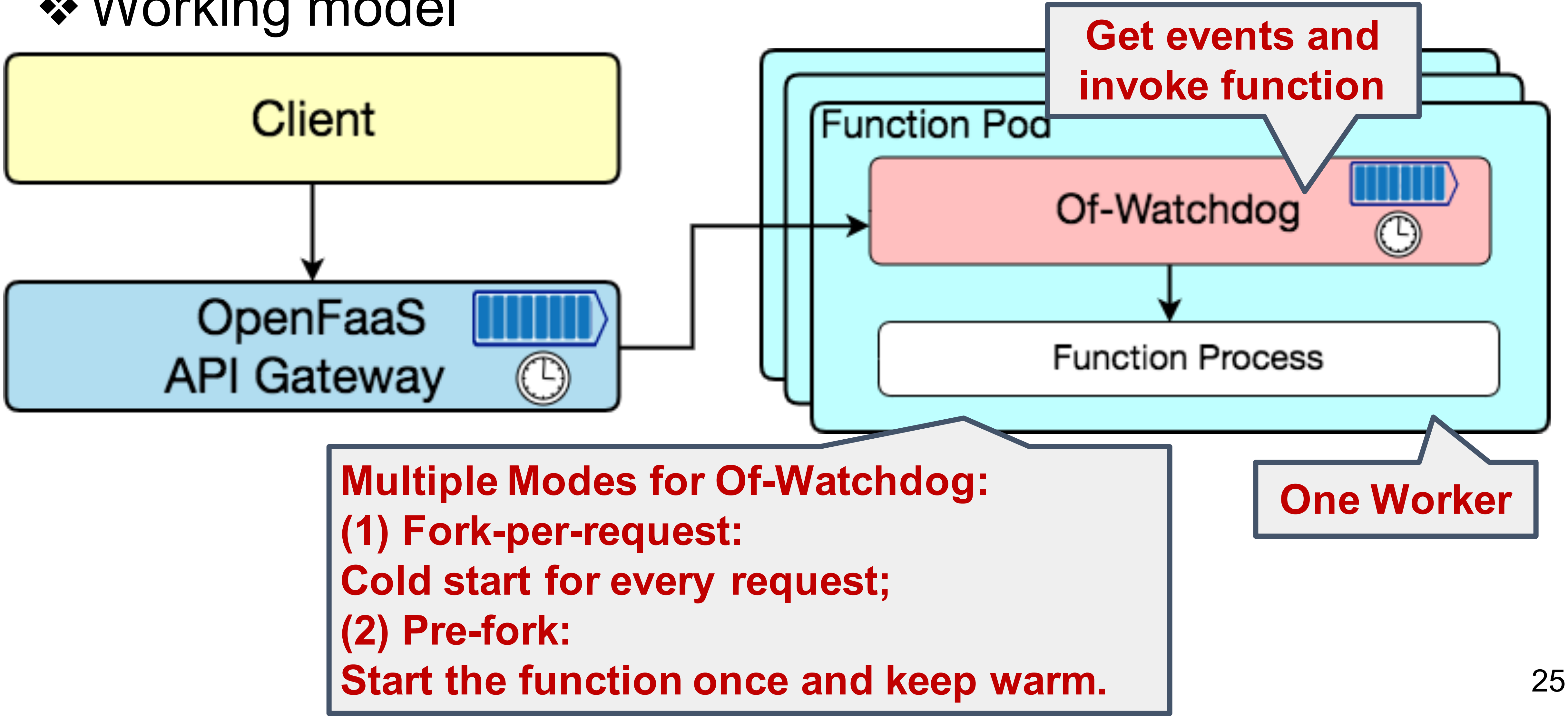
OpenFaaS

❖ Working model

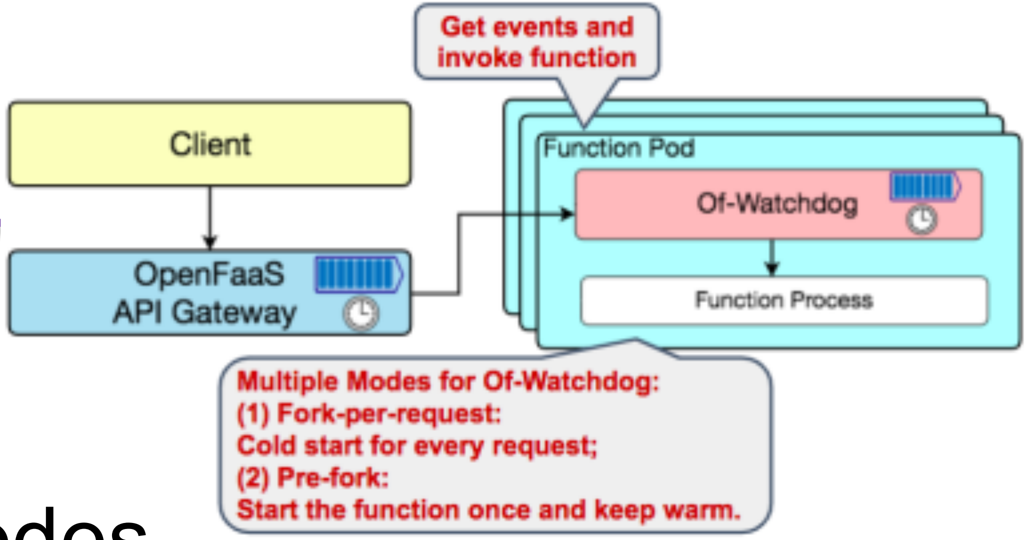


OpenFaaS

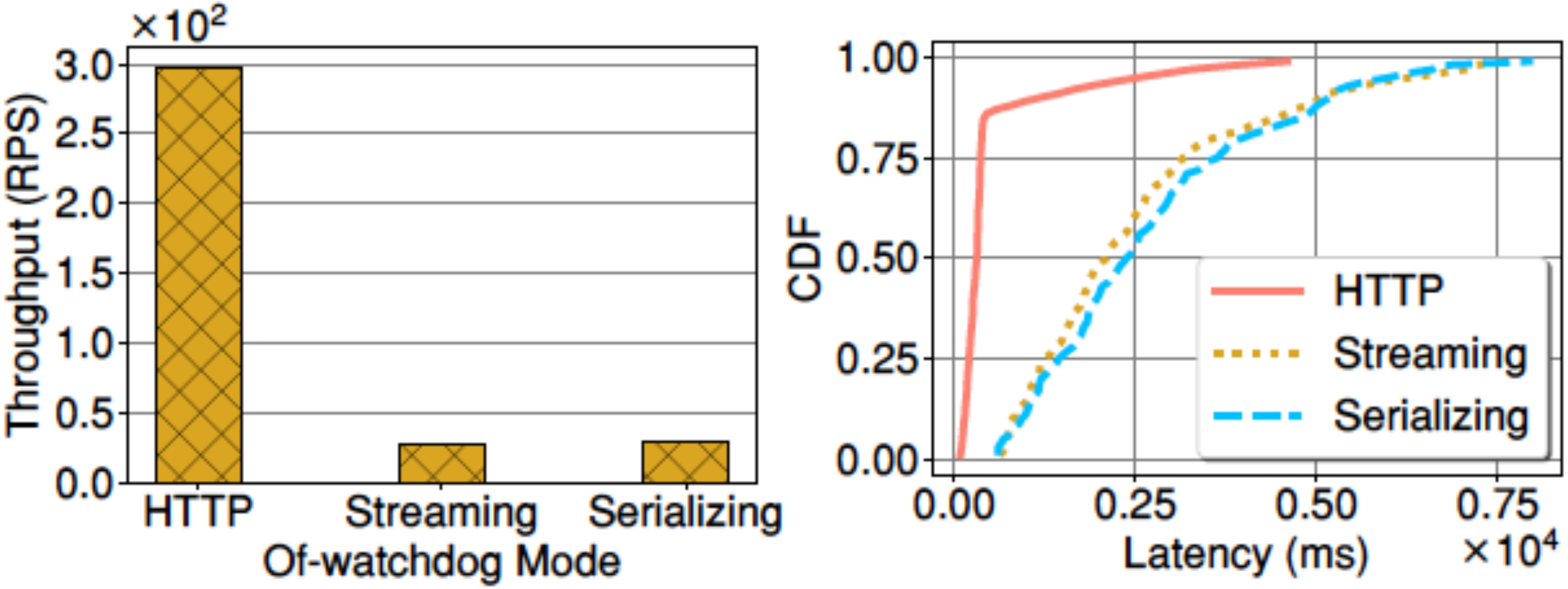
- Working model



OpenFaaS



❖ Salient parameter: of-watchdog modes



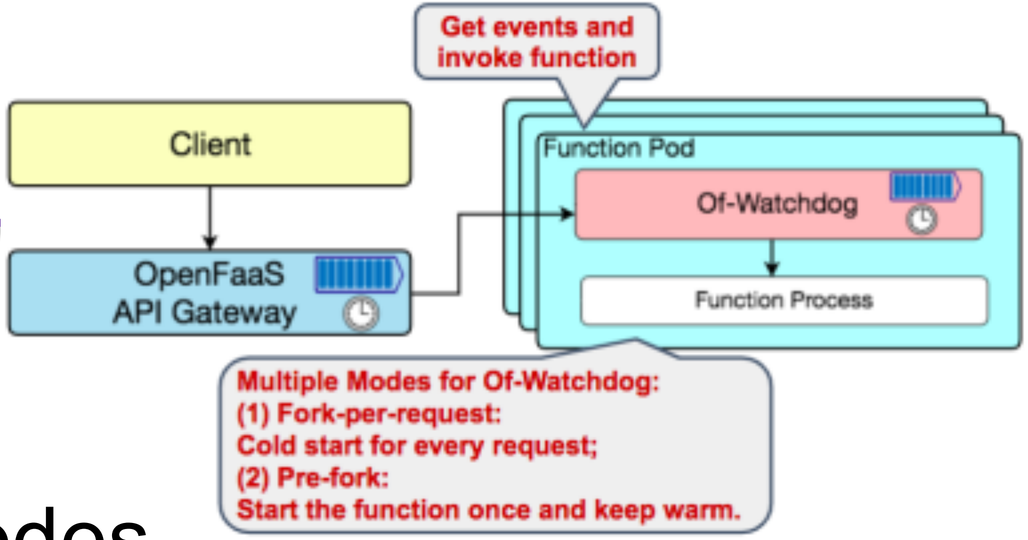
(a) Throughput in requests/second.

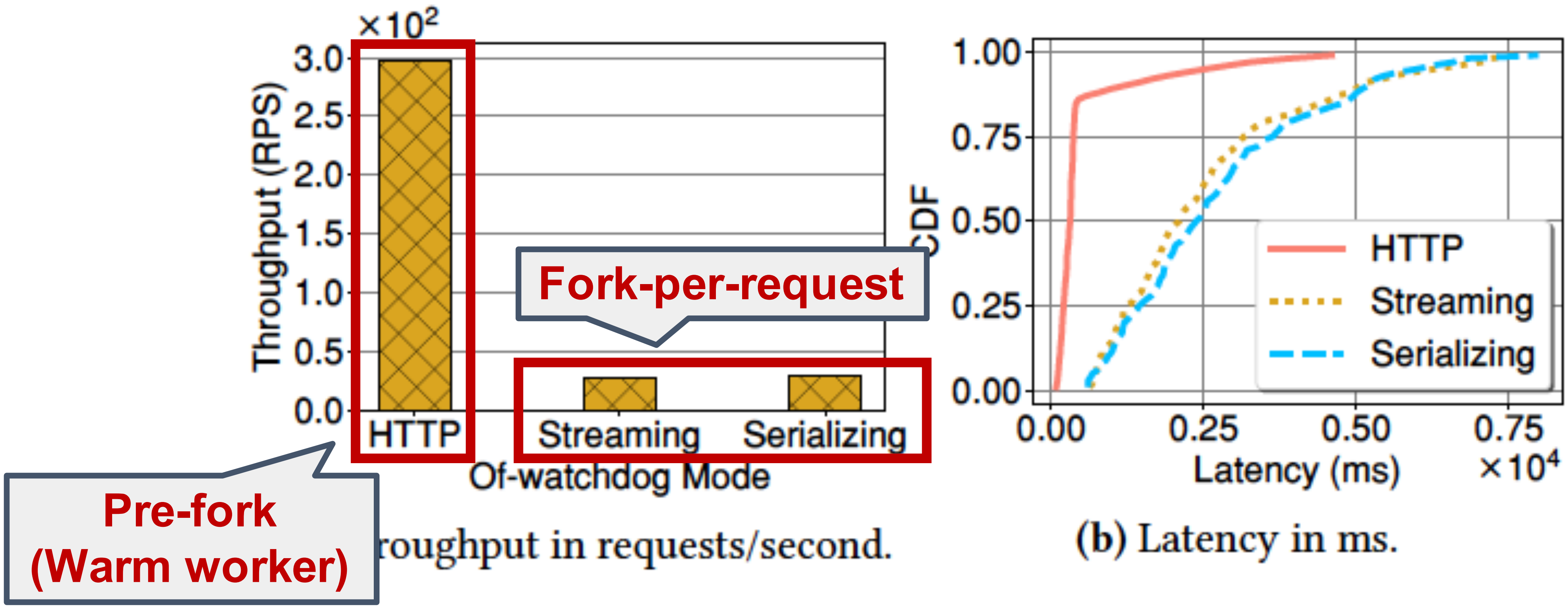
(b) Latency in ms.

OpenFaaS

"Pre-fork" mode has much better performance than "Fork-per-request".

❖ Salient parameter: of-watchdog modes



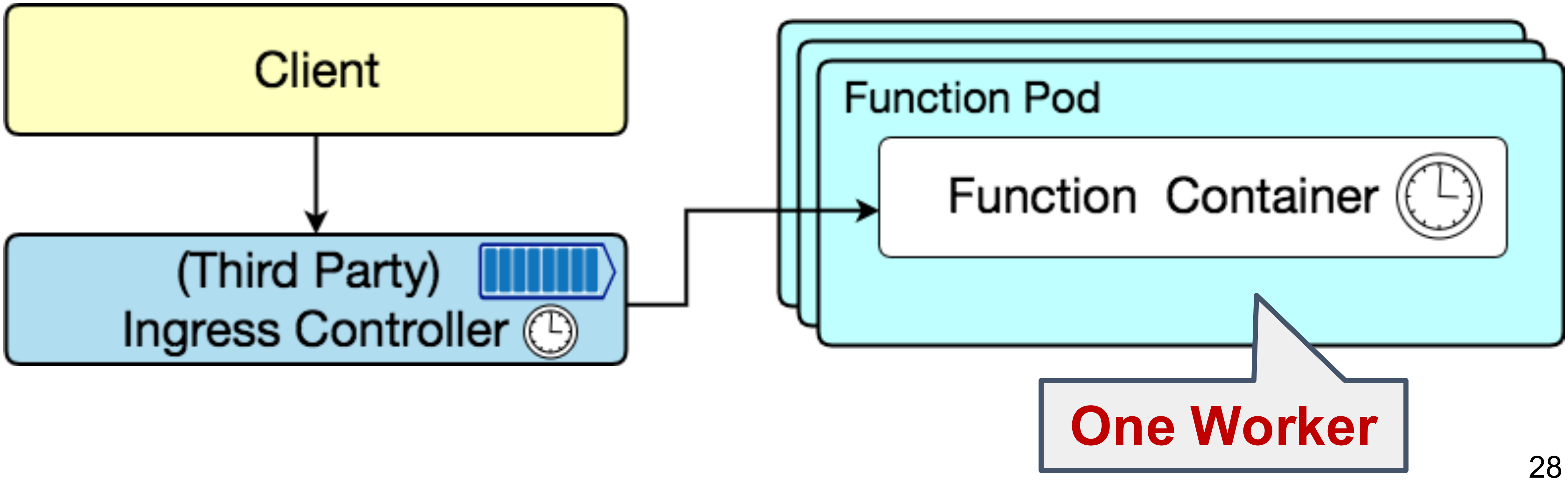


roughput in requests/second.

(b) Latency in ms.

Kubeless

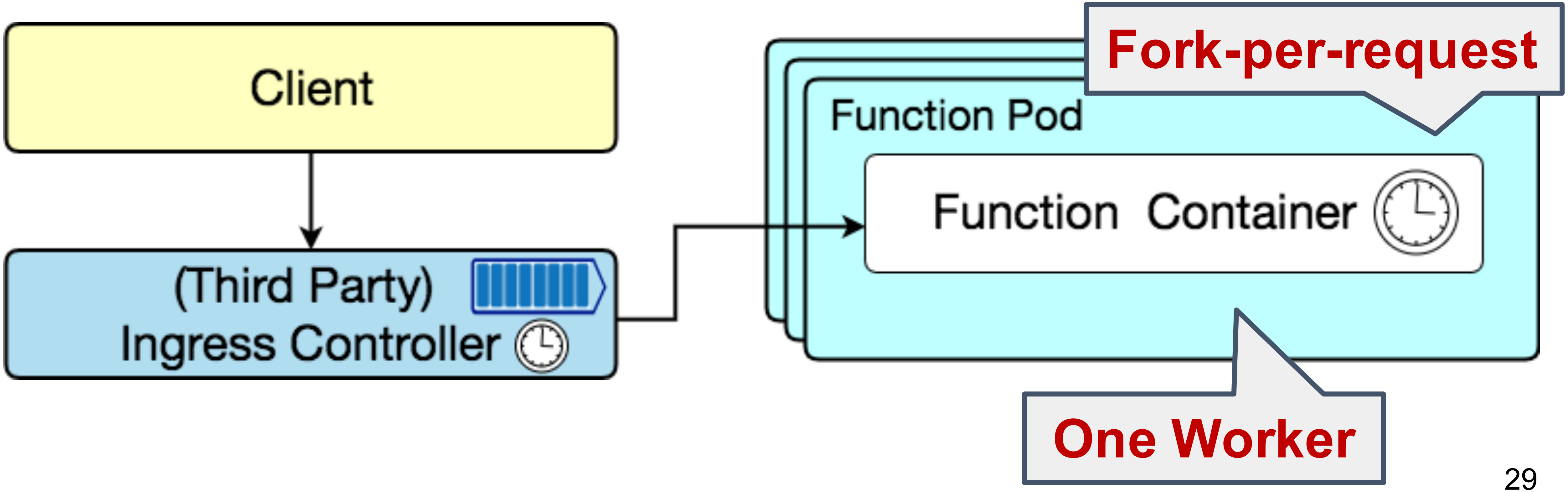
❖ Working model



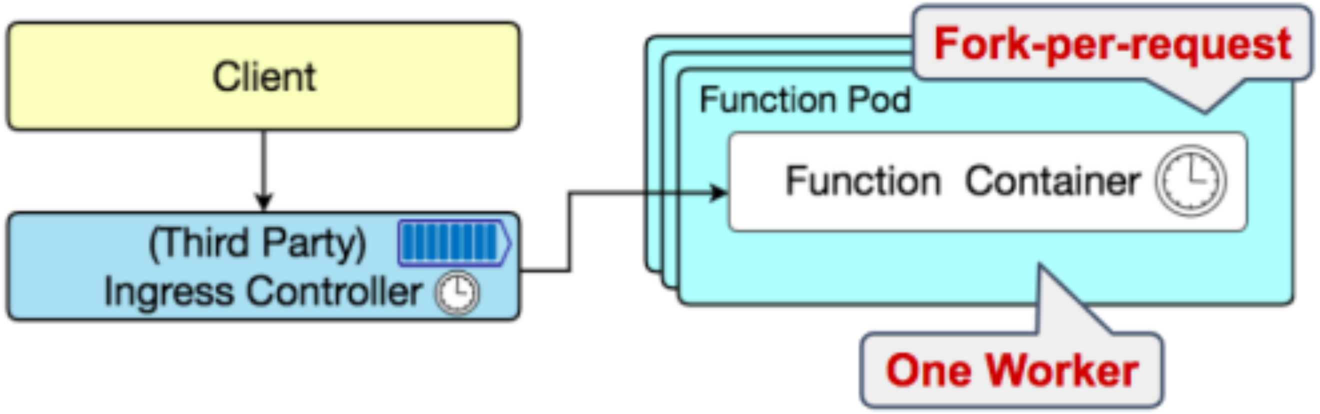
Kubeless

- Working model

Kubeless only supports "Fork-per-request" mode.



Kubeless



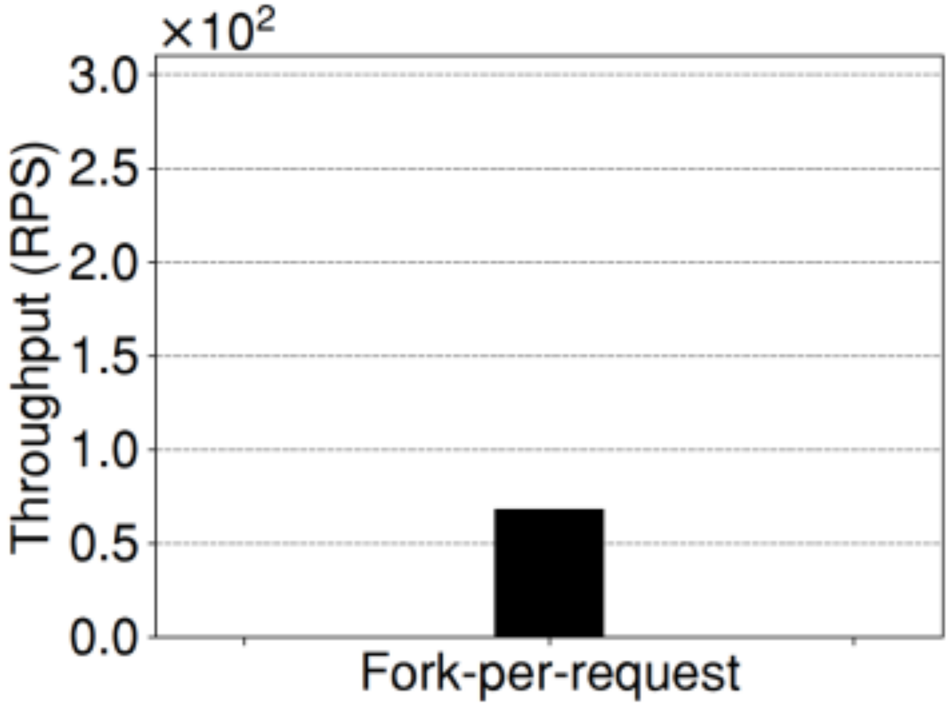


Fig. Throughput of Kubeless.

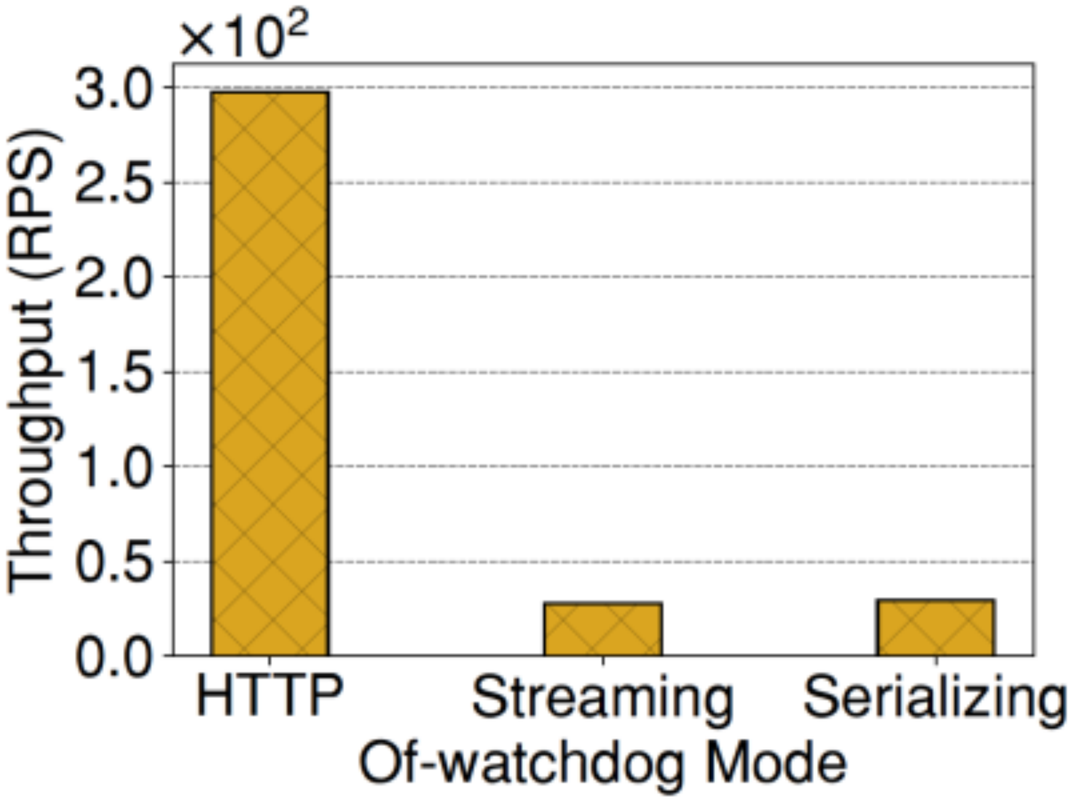
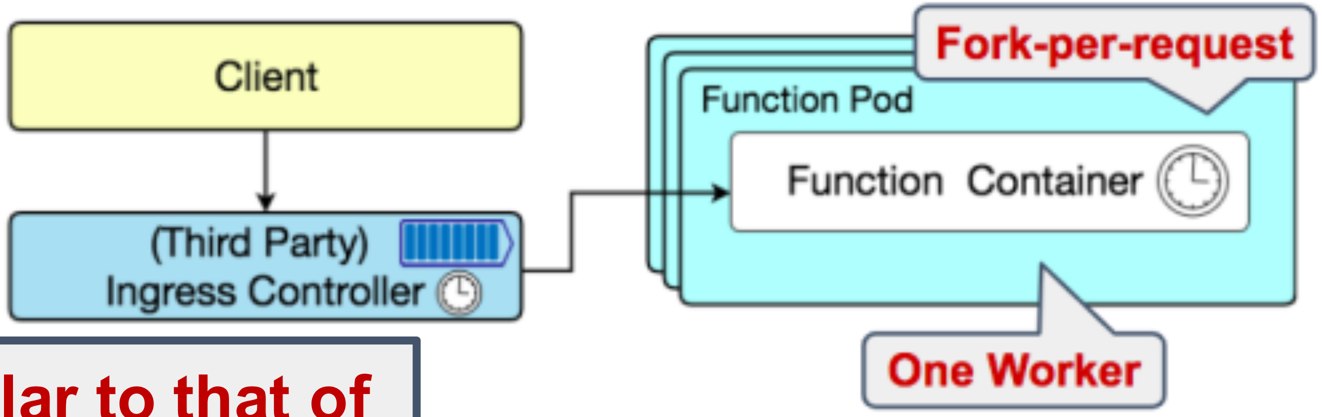


Fig. Throughput of OpenFaaS.

Kubeless



The performance of Kubeless is similar to that of OpenFaaS in "Fork-per-request" mode.

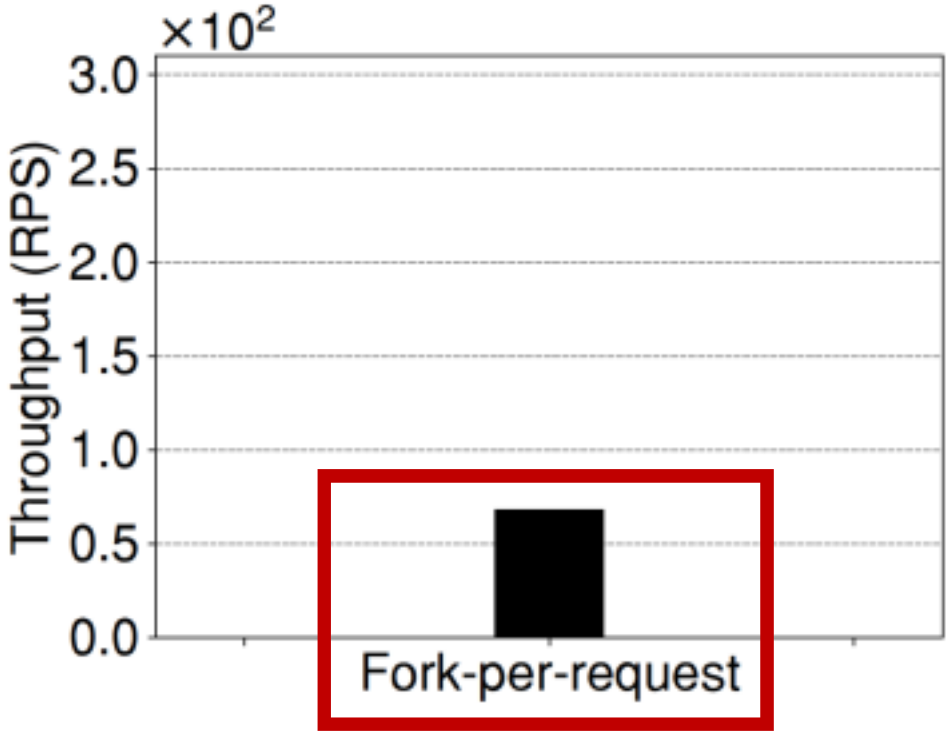


Fig. Throughput of Kubeless.

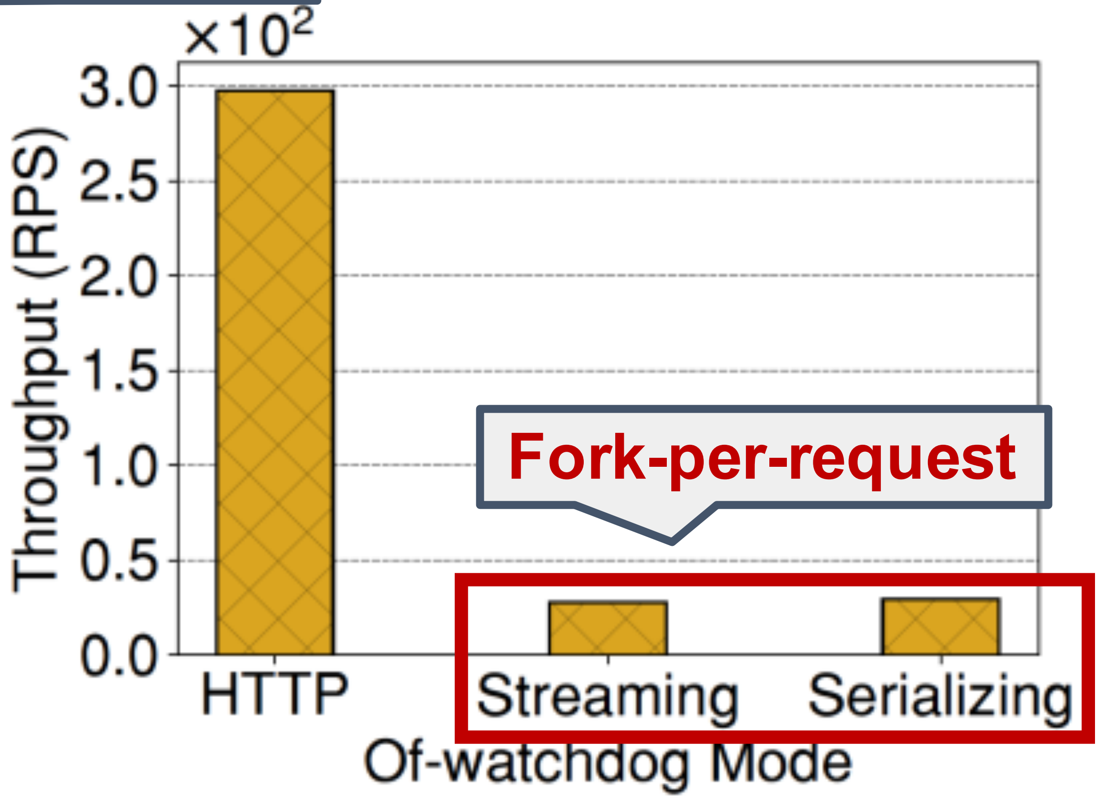


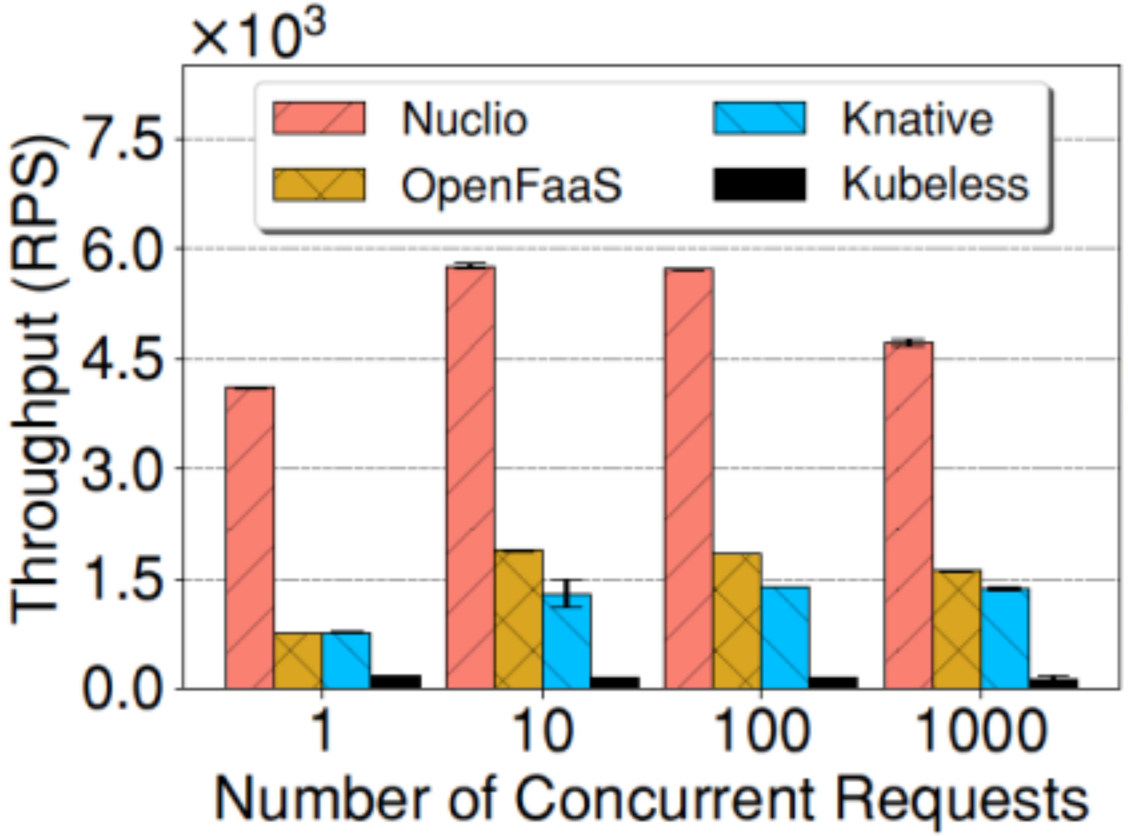
Fig. Throughput of OpenFaaS.

Motivation and Goals

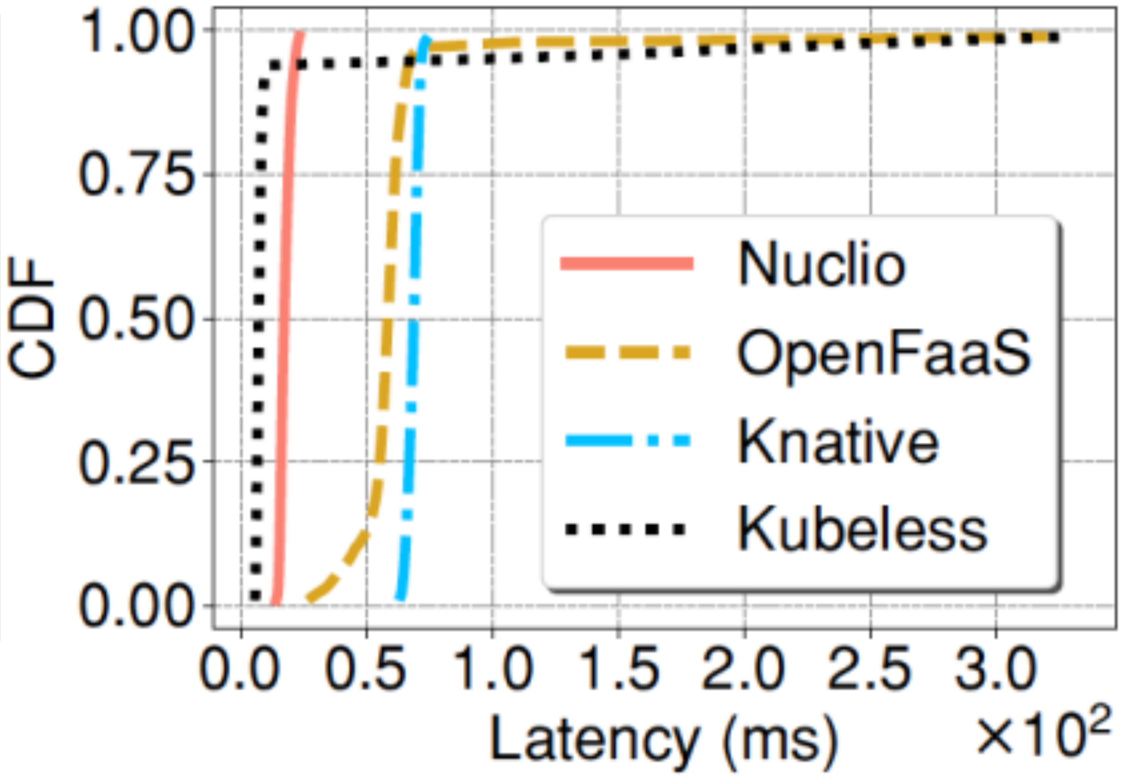
- To develop an understanding on the open source [illegible] configuration parameters of different components (platform, gateway, controller and function)
- Evaluate and compare the performance of open source serverless platforms:
  - Different workloads
  - Different auto-scaling modes

Performance

Baseline: helloworld function (Return "hello")



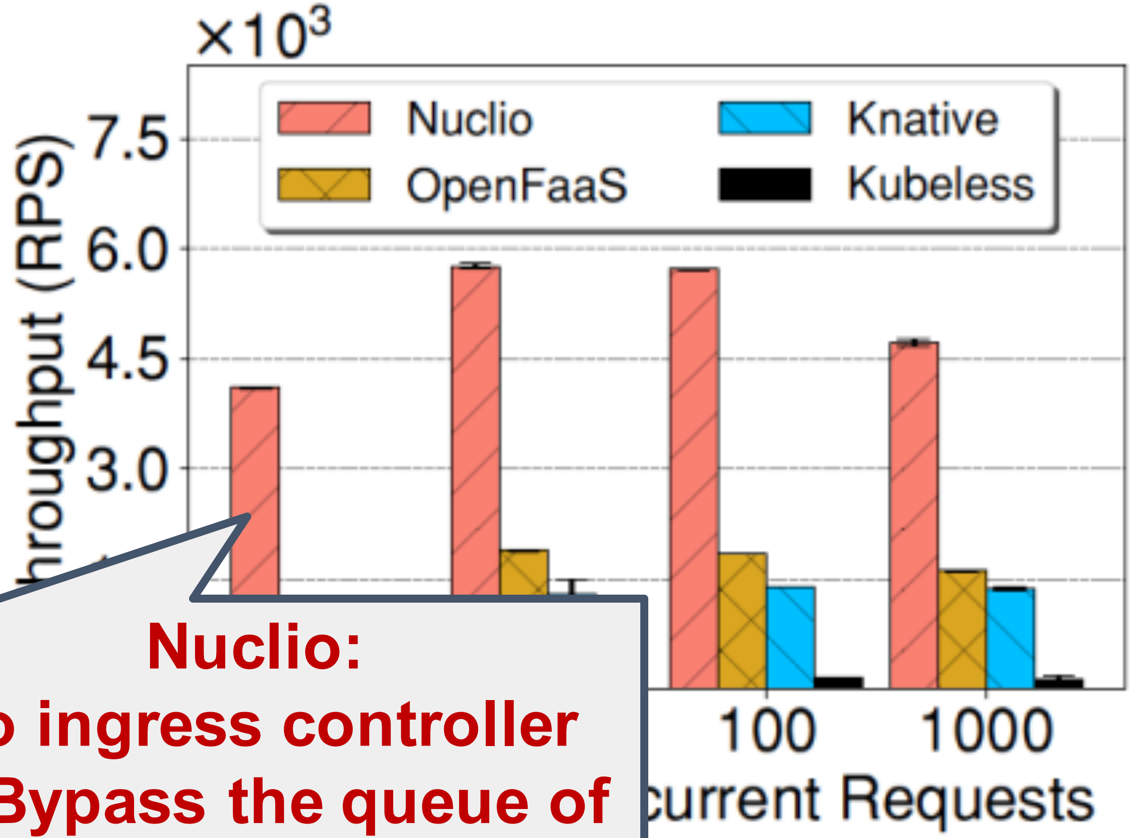
(a) Avg. throughput.



(b) Latency for concurrency of 100.

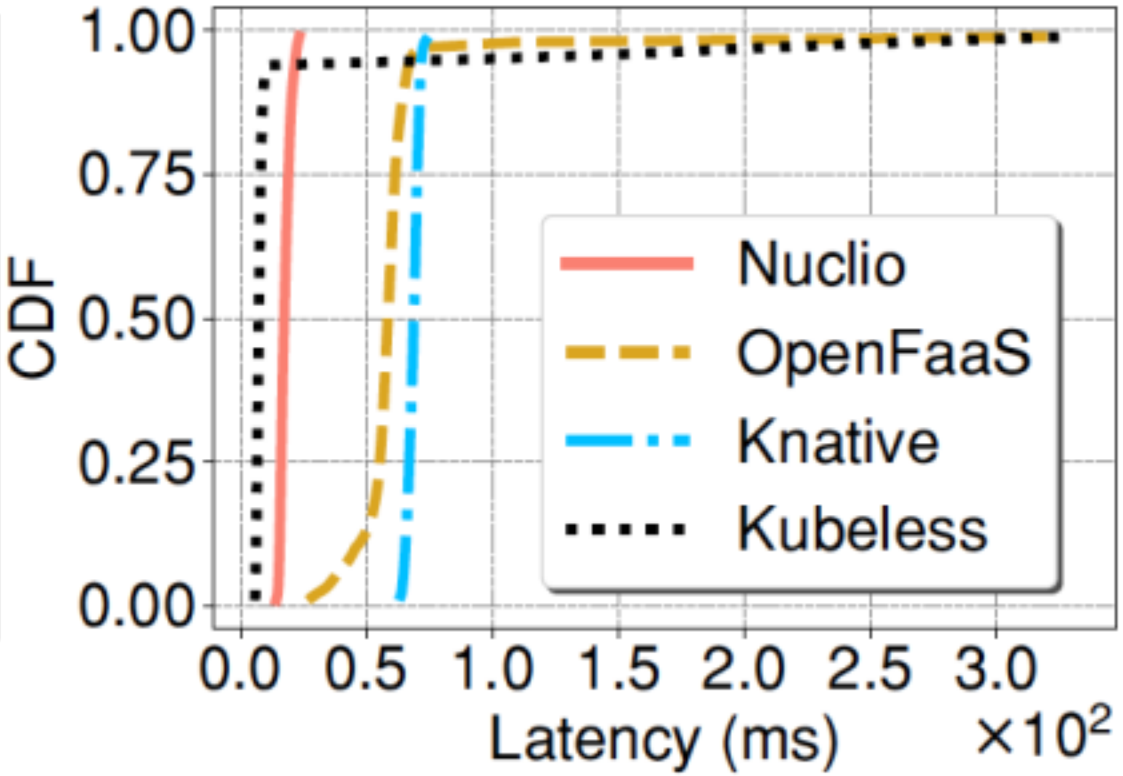
Performance

Baseline: helloworld function (Return "hello")



ughput.

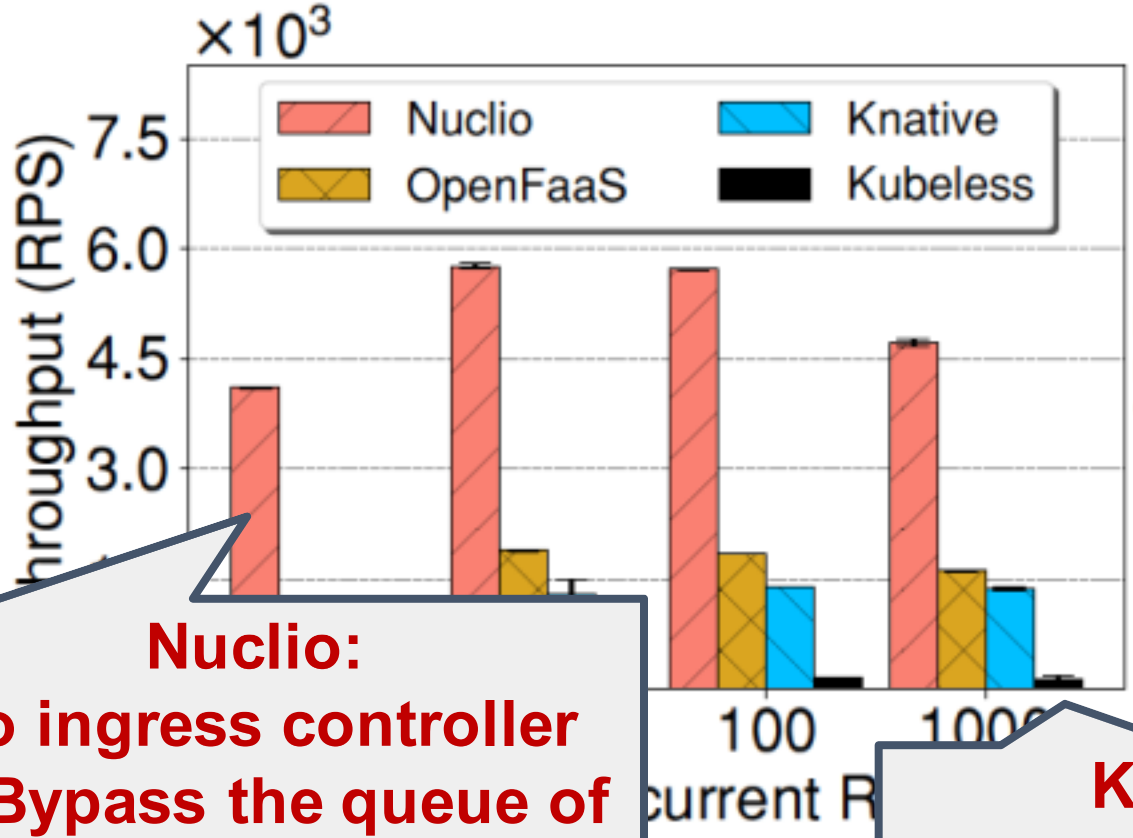
Nuclio:
No ingress controller
⇒ Bypass the queue of ingress controller
⇒ Highest throughput



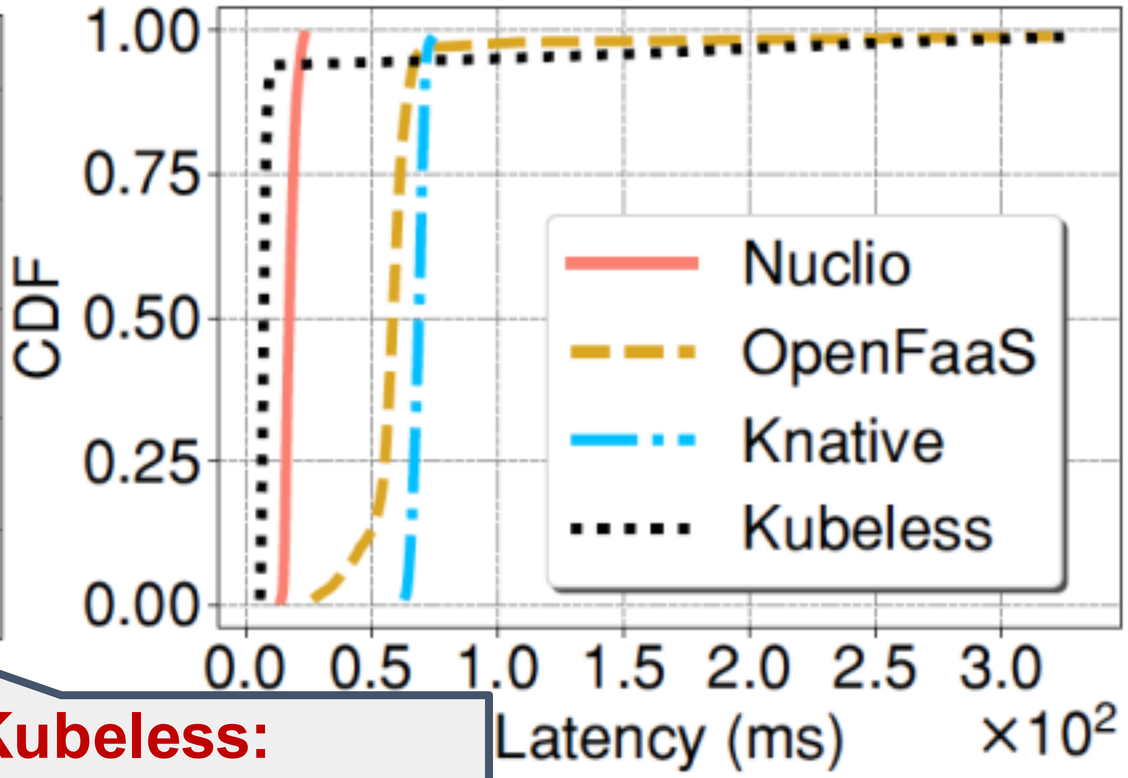
(b) Latency for concurrency of 100.

Performance

Baseline: helloworld function (Return "hello")



ughput.



concurrency of 100.

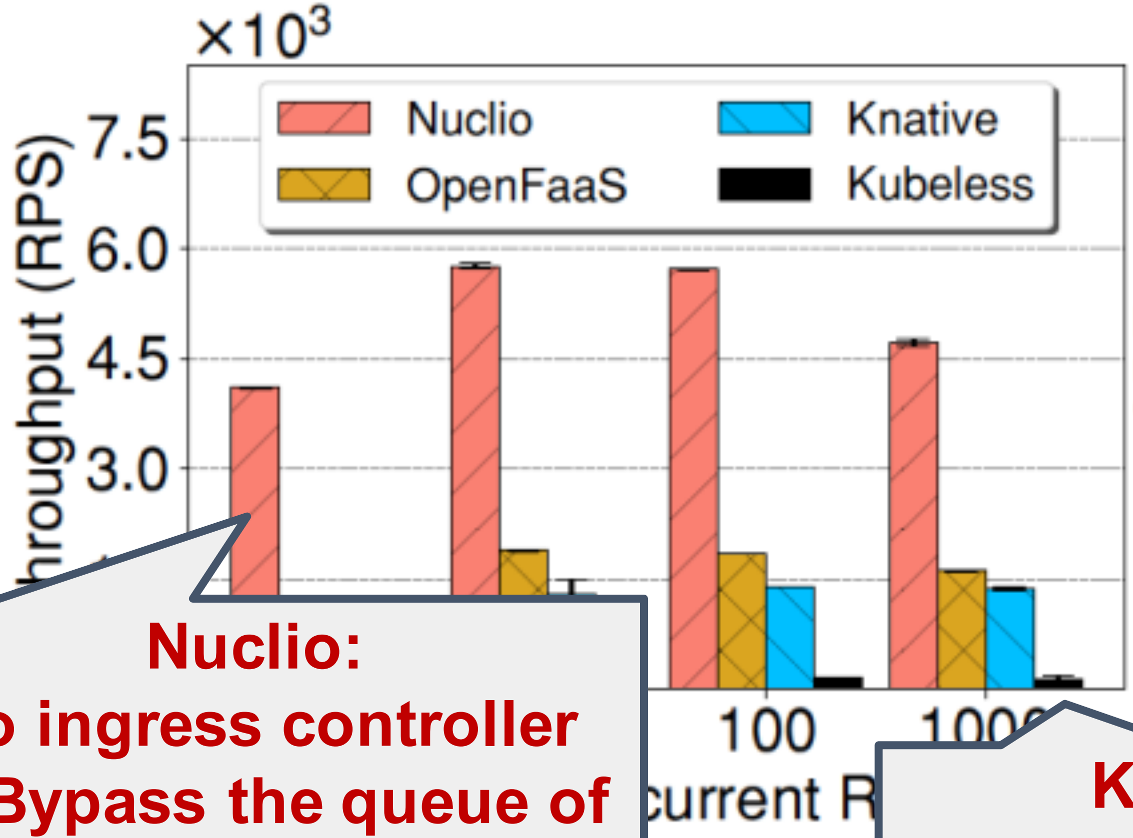
Nuclio:
No ingress controller
⇒ Bypass the queue of ingress controller
⇒ Highest throughput

Kubeless:
Fork-per-request
⇒ Lowest throughput

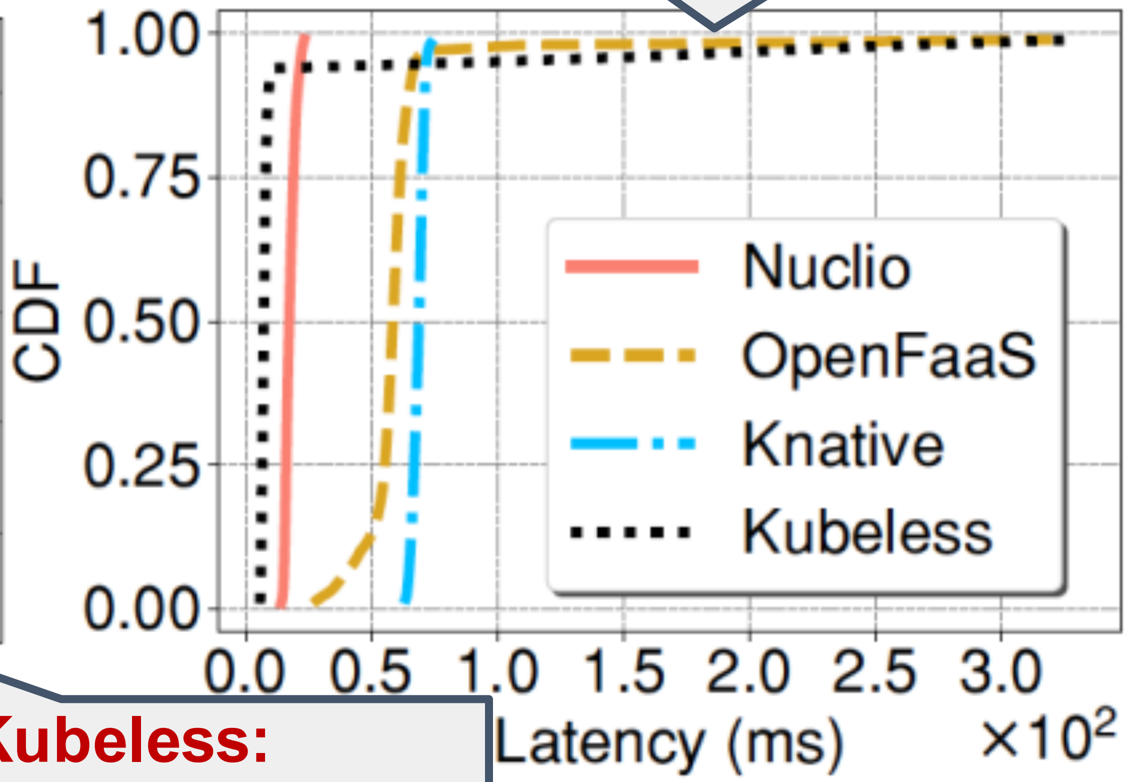
Performance

Baseline: helloworld function (Return “hello”)

Queuing Up ⇒ Long tail



Nuclio:
No ingress controller
⇒ Bypass the queue of ingress controller
⇒ Highest throughput

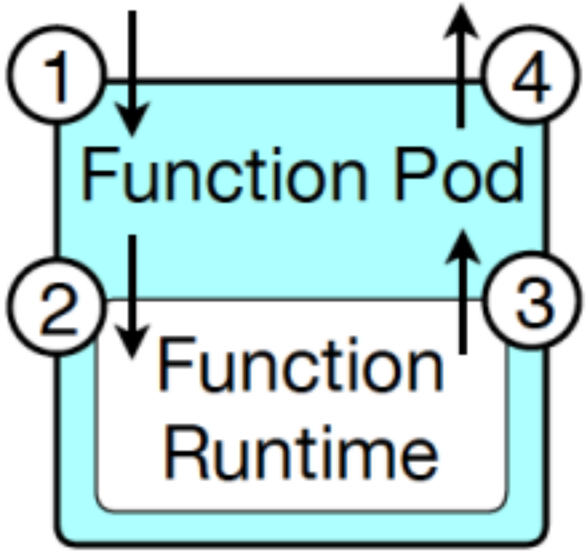


Kubeless:
Fork-per-request
⇒ Lowest throughput

ughput. concurrency of 100.

Performance

Latency breakdown of helloworld function:



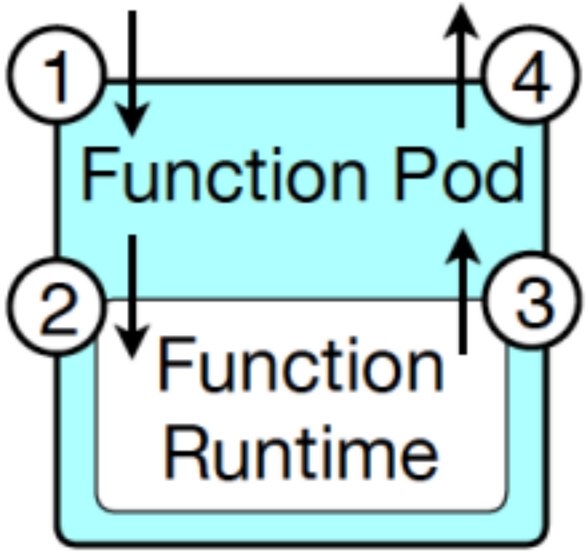
Process	1→2	2→3	3→4
Nuclio	0.63	0.001	0.54
OpenFaaS	1.32	0.001	0.93
Knative	1.30	0.001	0.62
Kubeless	4.96	0.001	2.63

Figure 10. Latency breakdown (ms) parts of serverless execution.

Performance

Latency breakdown of helloworld func

Same Python Runtime
⇒ Same execution time



Process	1→2	2→3	3→4
Nuclio	0.63	0.001	0.54
OpenFaaS	1.32	0.001	0.93
Knative	1.30	0.001	0.62
Kubeless	4.96	0.001	2.63

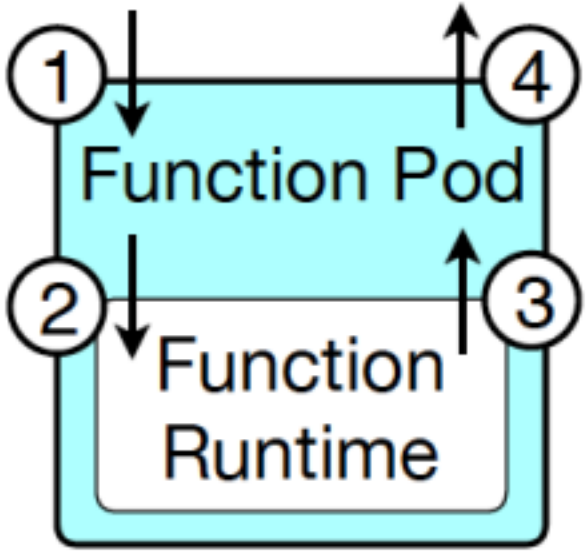
Figure 10. Latency breakdown (ms) parts of serverless execution.

Performance

Latency breakdown of helloworld func

Same Python Runtime
⇒ Same execution time

Platform Specific



Process	1→2	2→3	3→4
Nuclio	0.63	0.001	0.54
OpenFaaS	1.32	0.001	0.93
Knative	1.30	0.001	0.62
Kubeless	4.96	0.001	2.63

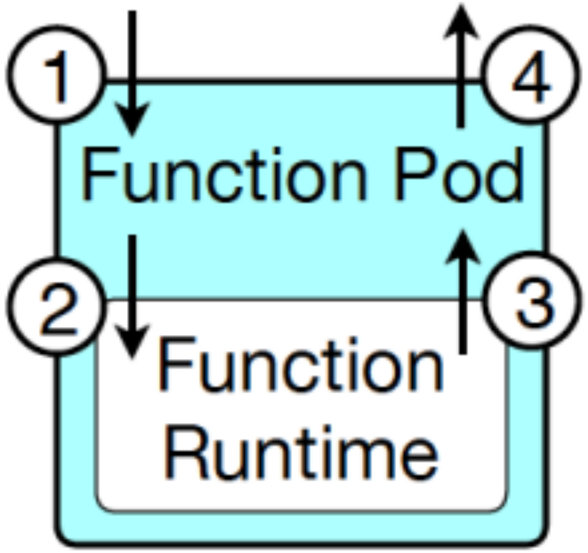
Figure 10. Latency breakdown (ms) parts of serverless execution.

Performance

Latency breakdown of helloworld func

Same Python Runtime
⇒ Same execution time

Platform Specific



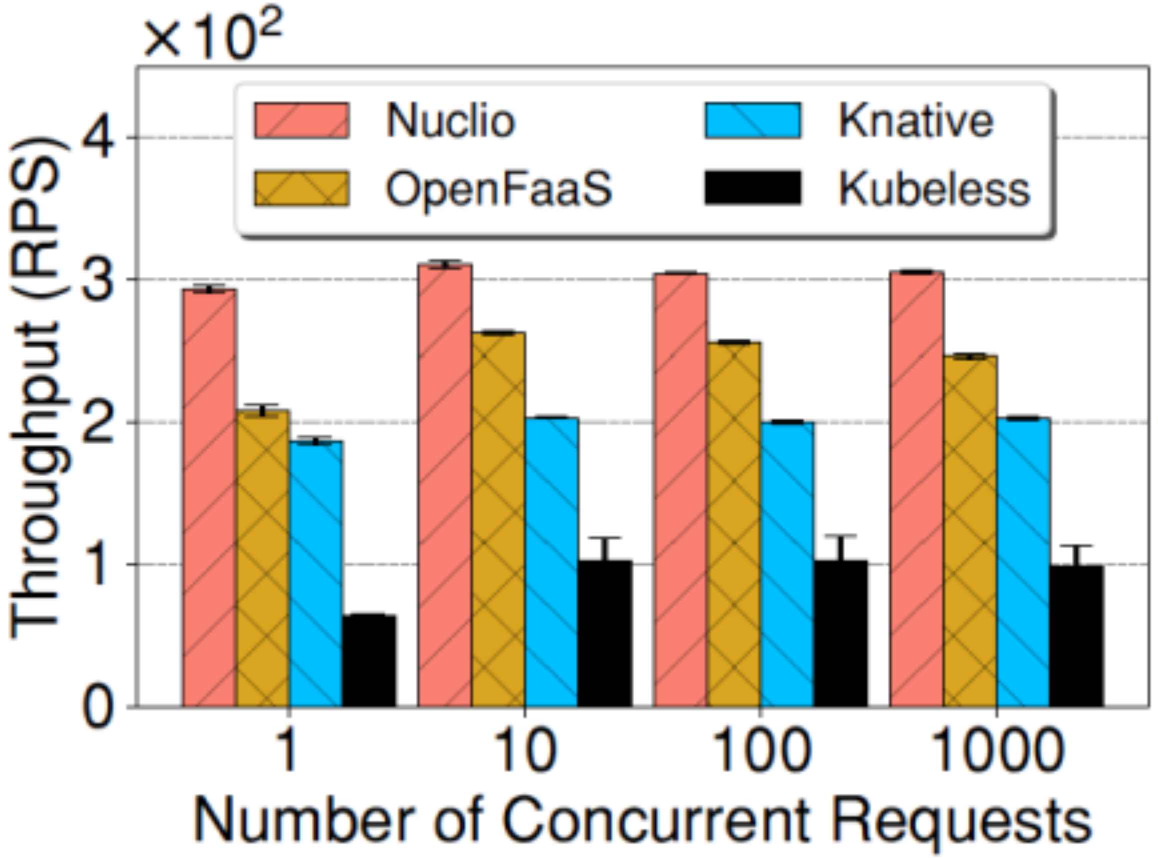
Process	1→2	2→3	3→4
Nuclio	0.63	0.001	0.54
OpenFaaS	1.32	0.001	0.93
Knative	1.30	0.001	0.62
Kubeless	4.96	0.001	2.63

Figure 10. Latency breakdown (ms) parts erverless execution.

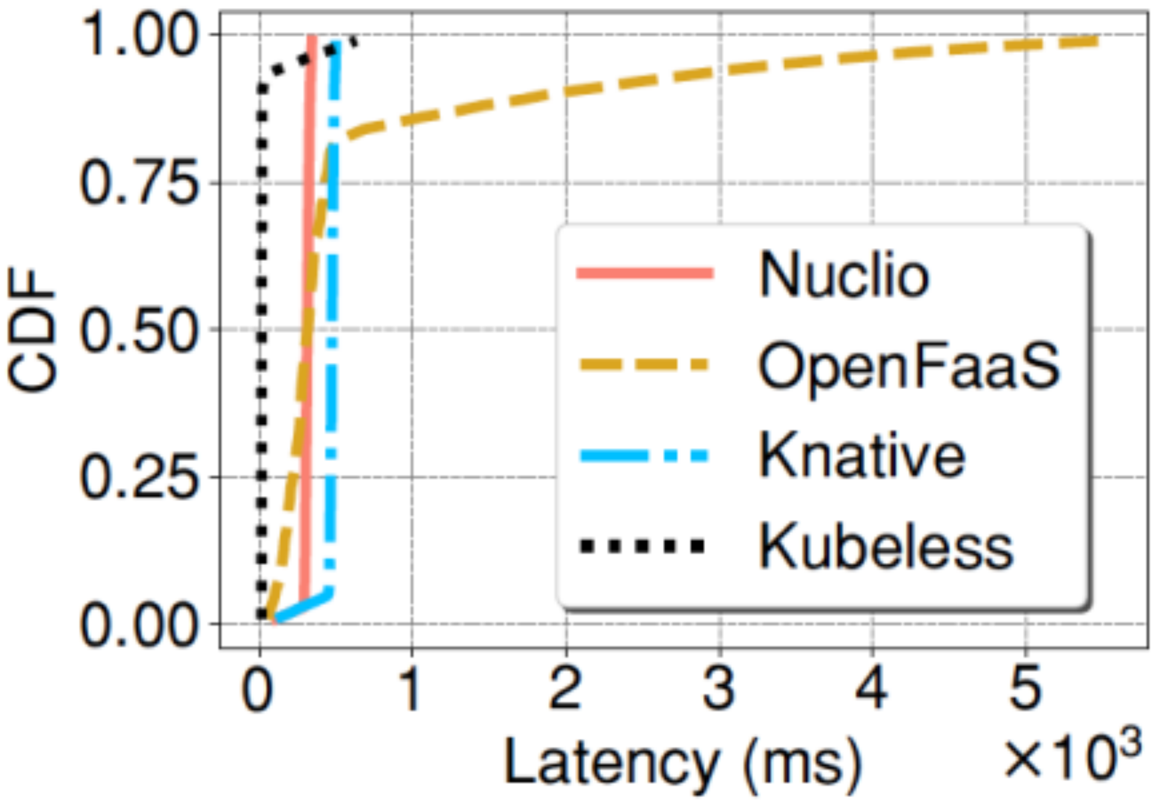
Fork-Per-Request
(Cold Start All the Time)

Performance

HTTP Workload: fetch a web page (5 Byte) from a local server



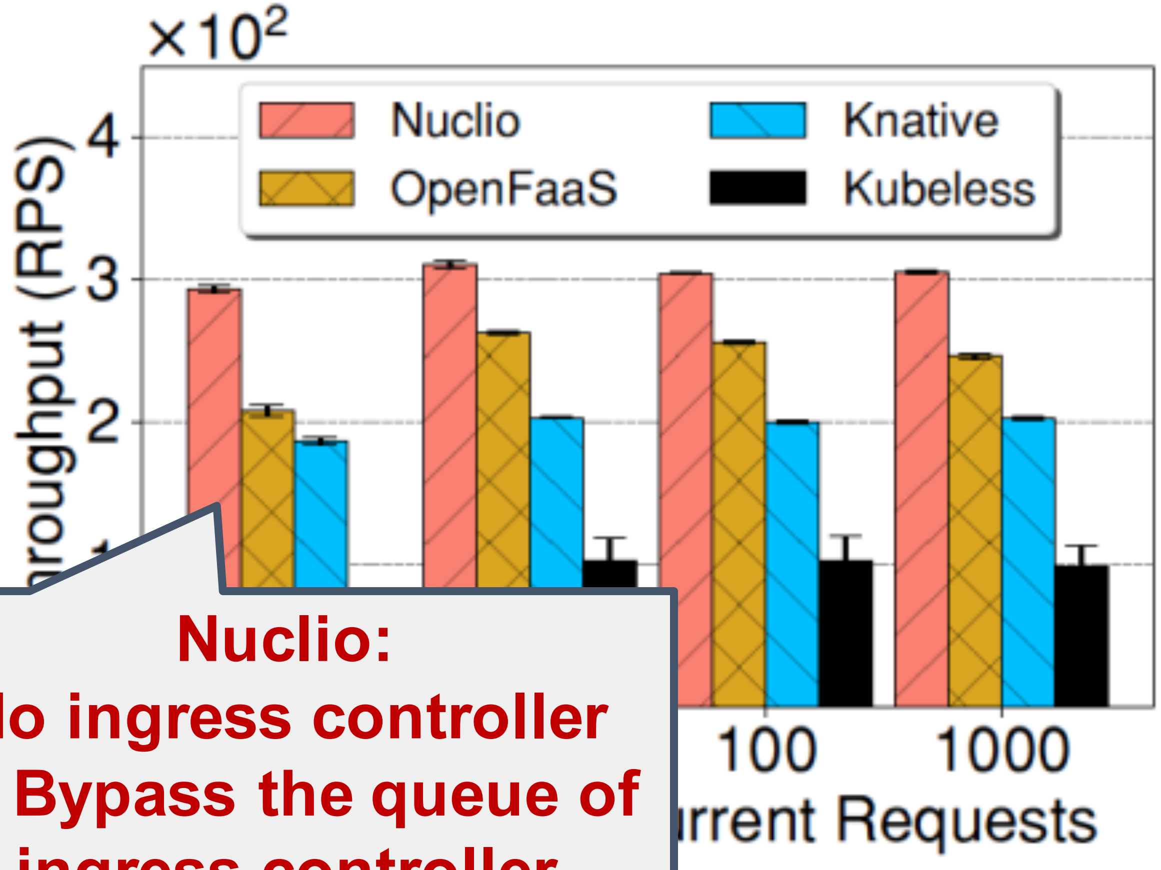
(a) Avg. throughput.



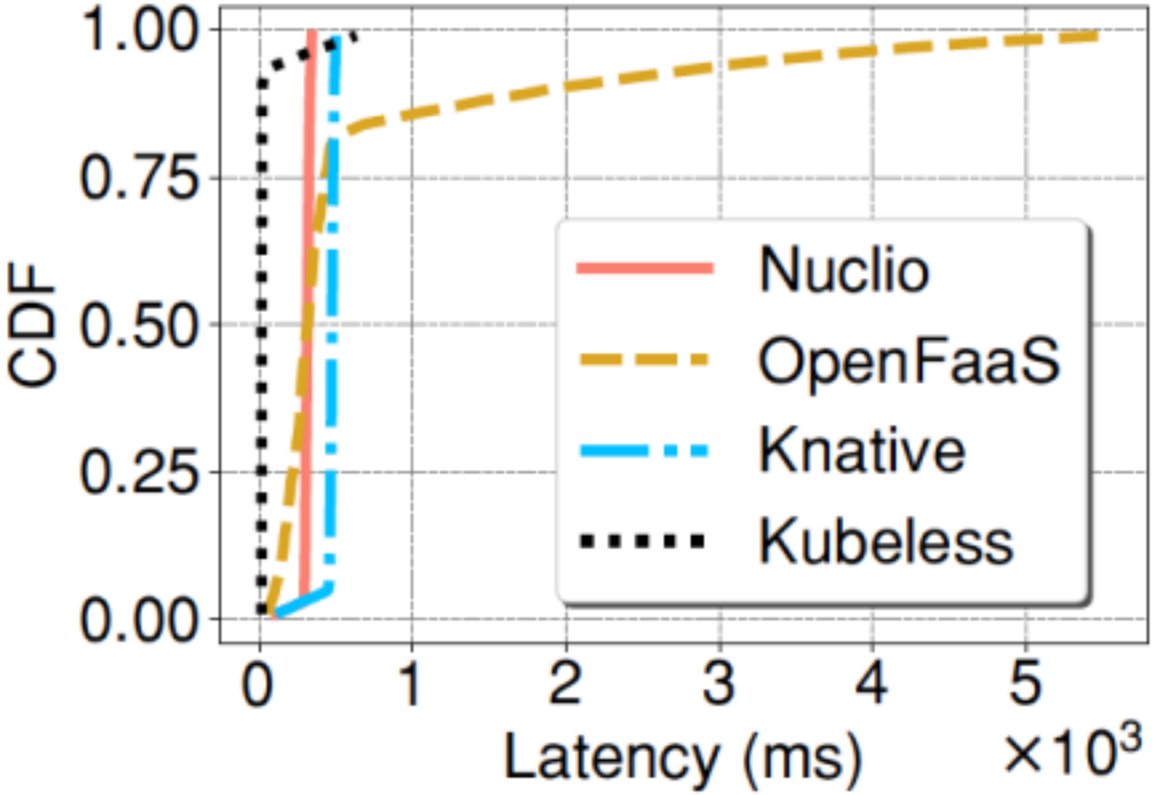
(b) Latency for concurrency of 100.

Performance

HTTP Workload: fetch a web page (5 Byte) from a local server



ughput.

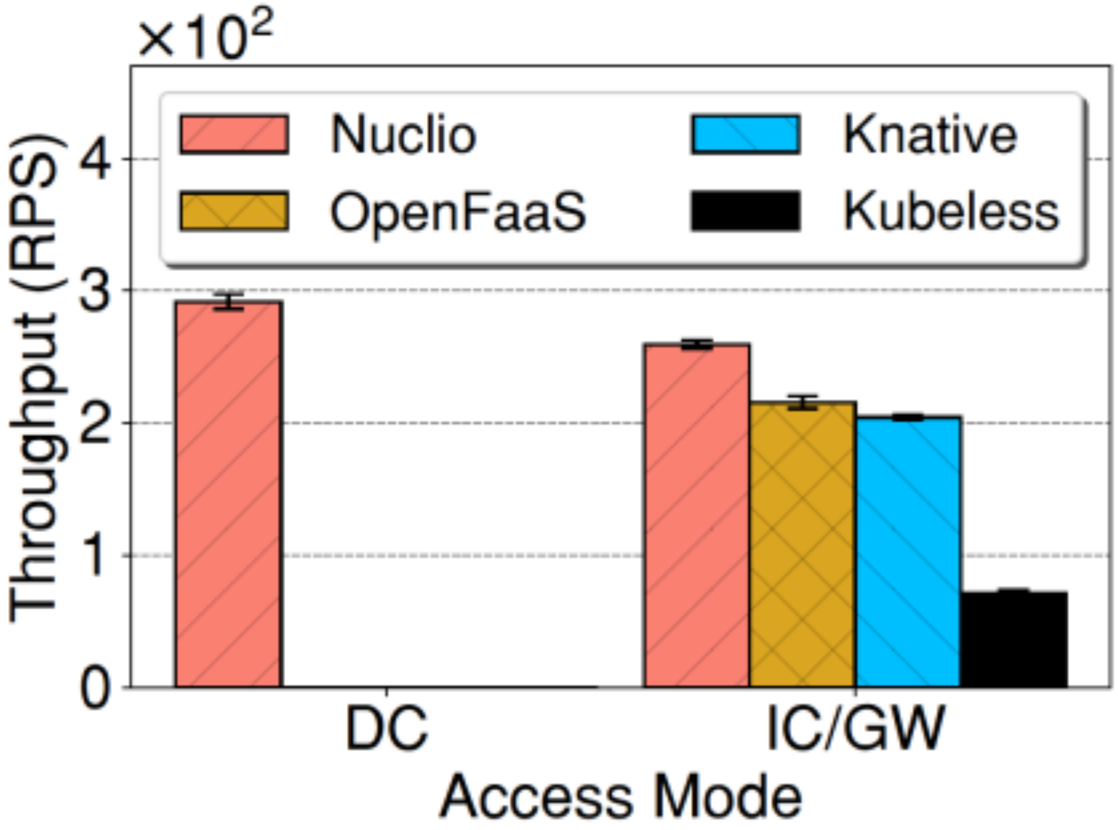


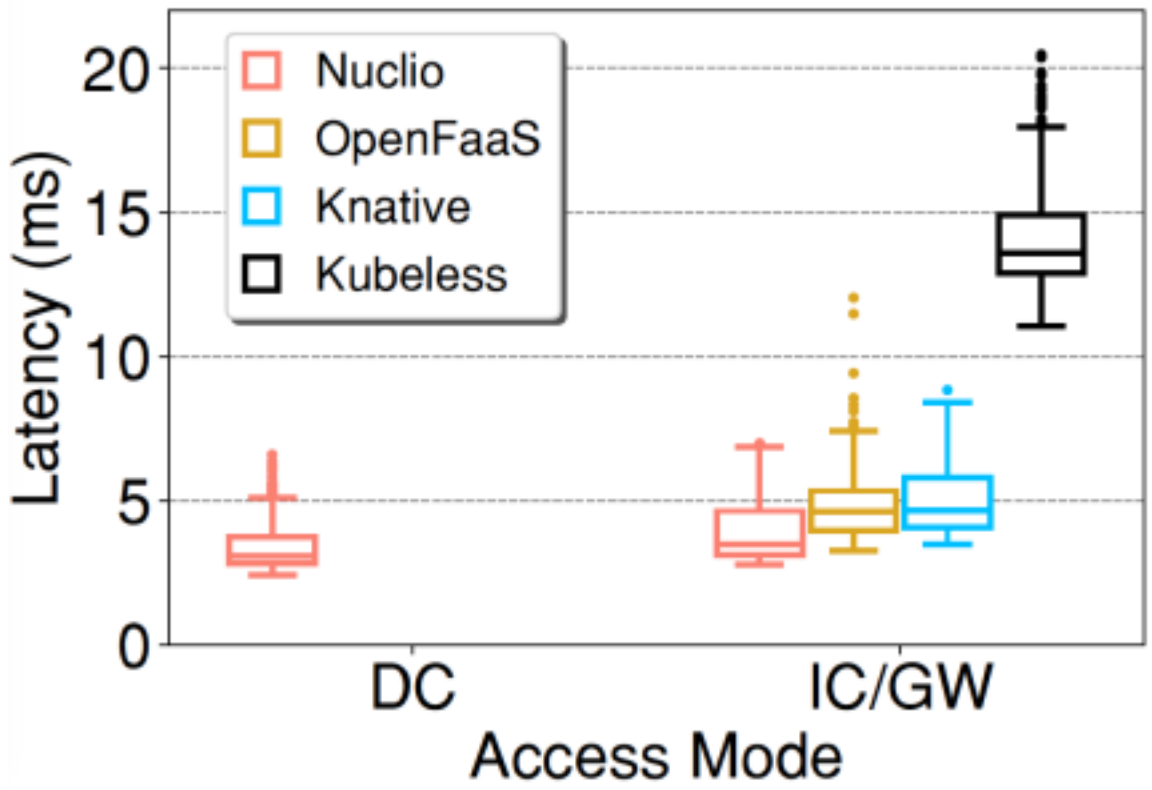
(b) Latency for concurrency of 100.

Nuclio:
No ingress controller
⇒ Bypass the queue of ingress controller
⇒ Highest throughput

Performance

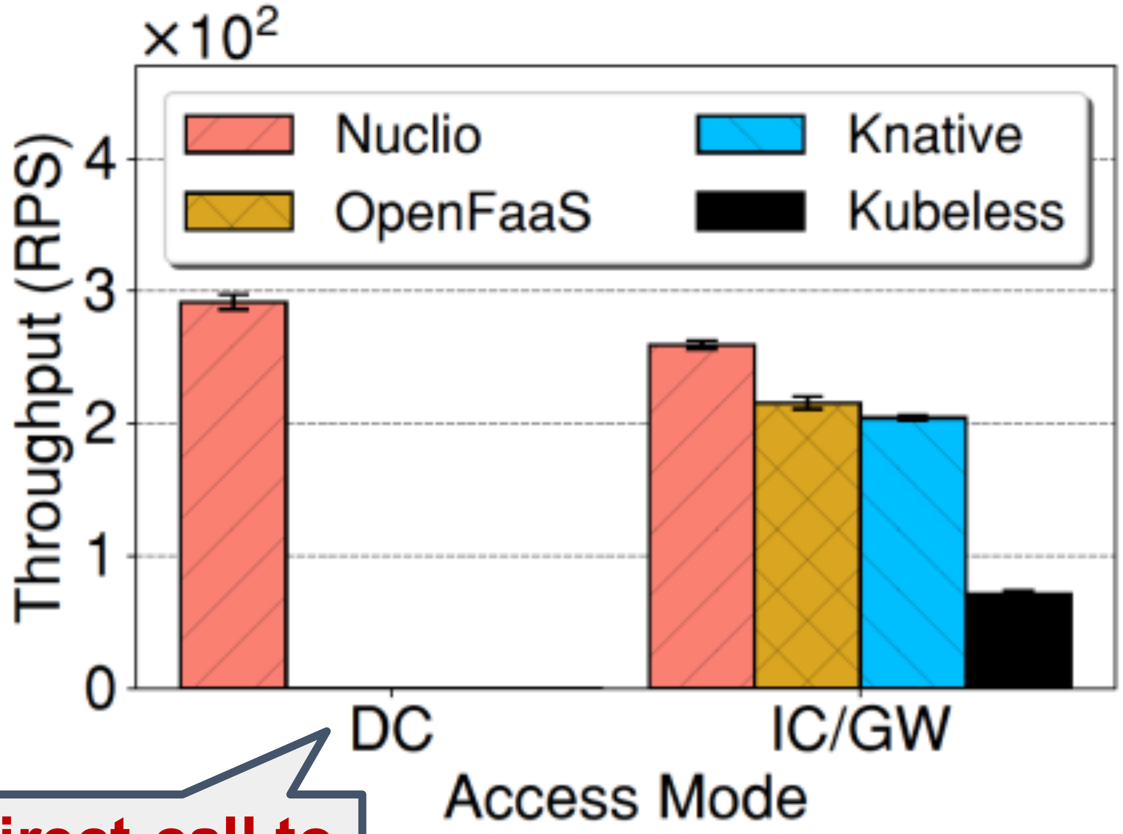
Different modes of exporting services:

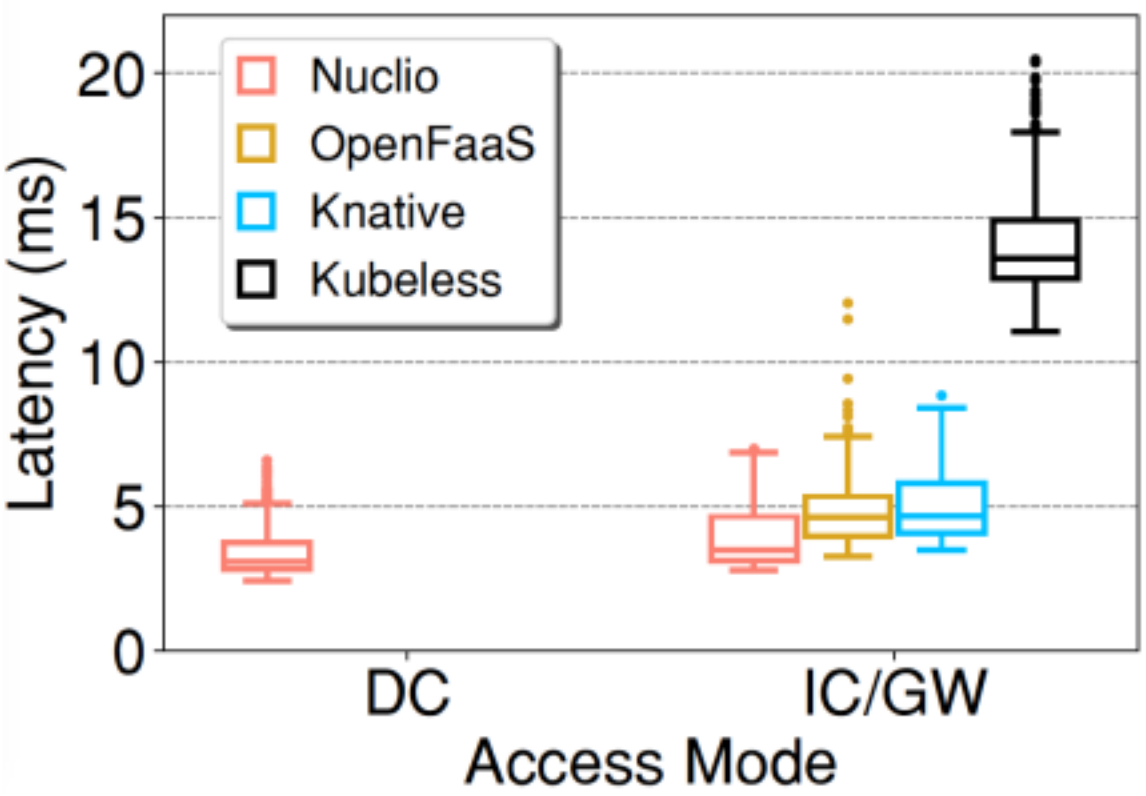




Performance

Different modes of exporting services:

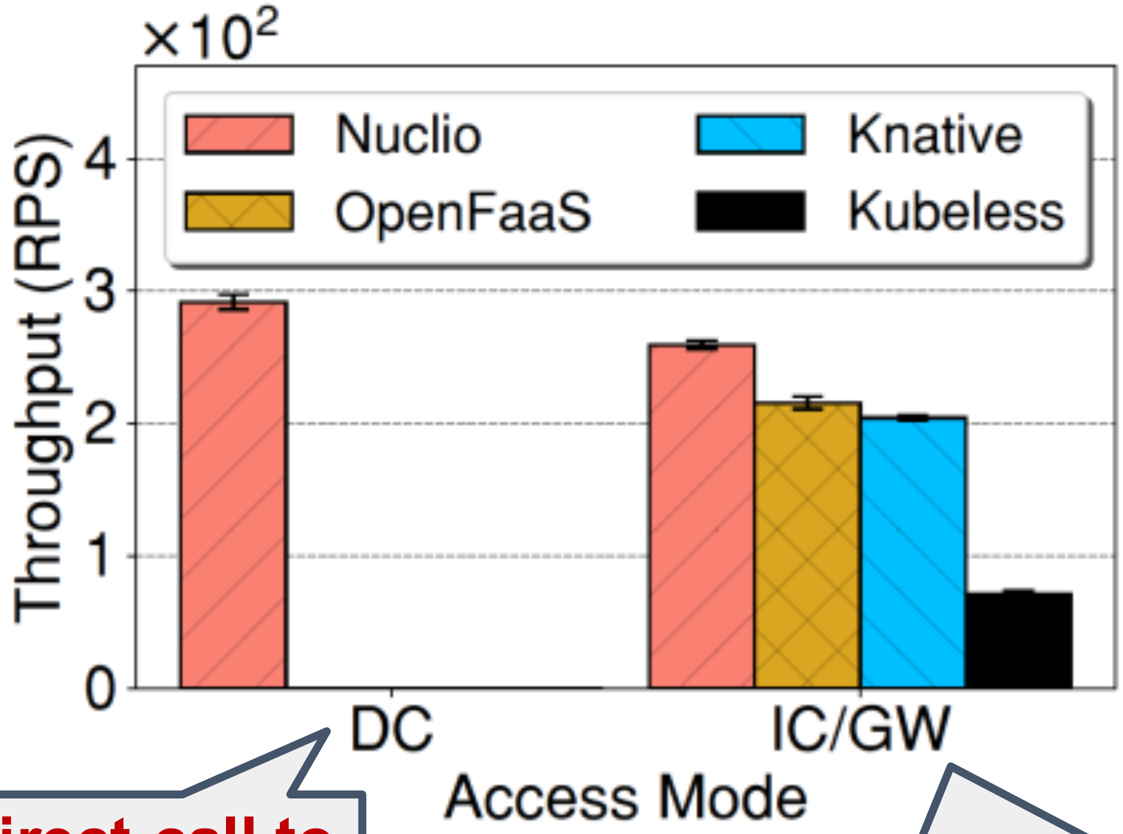


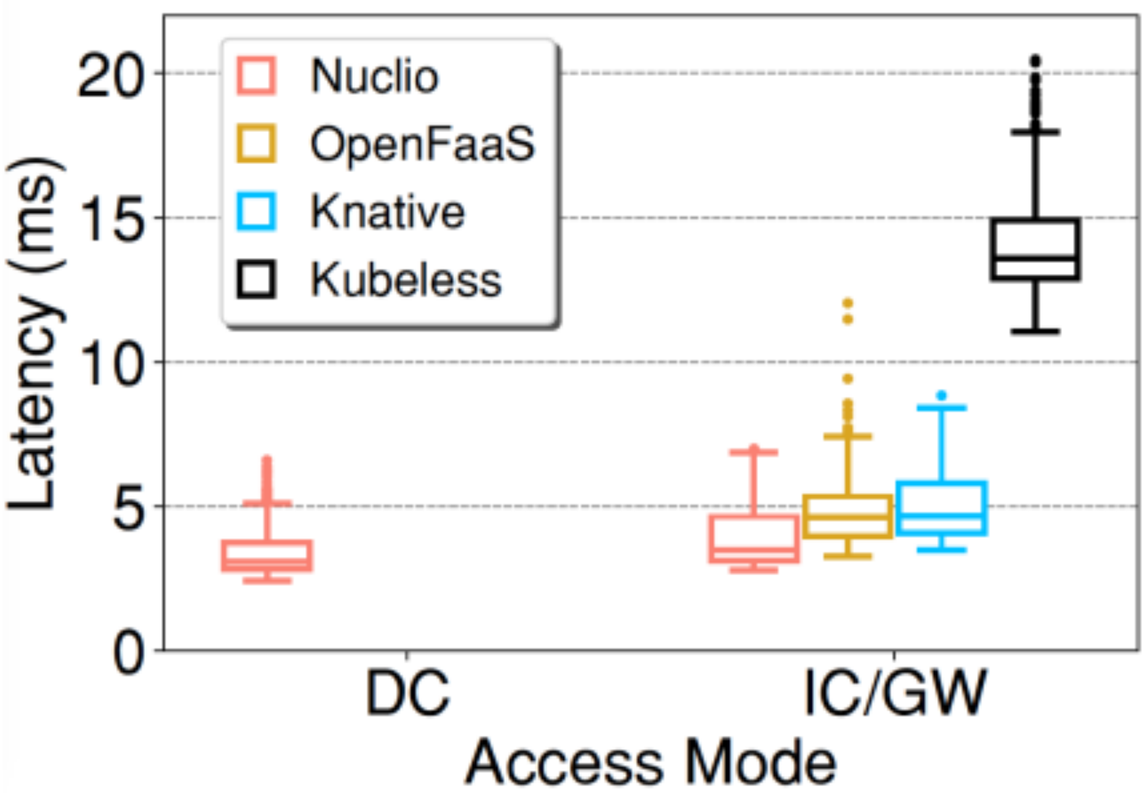


Direct call to func pod (NodePort)

Performance

Different modes of exporting services:





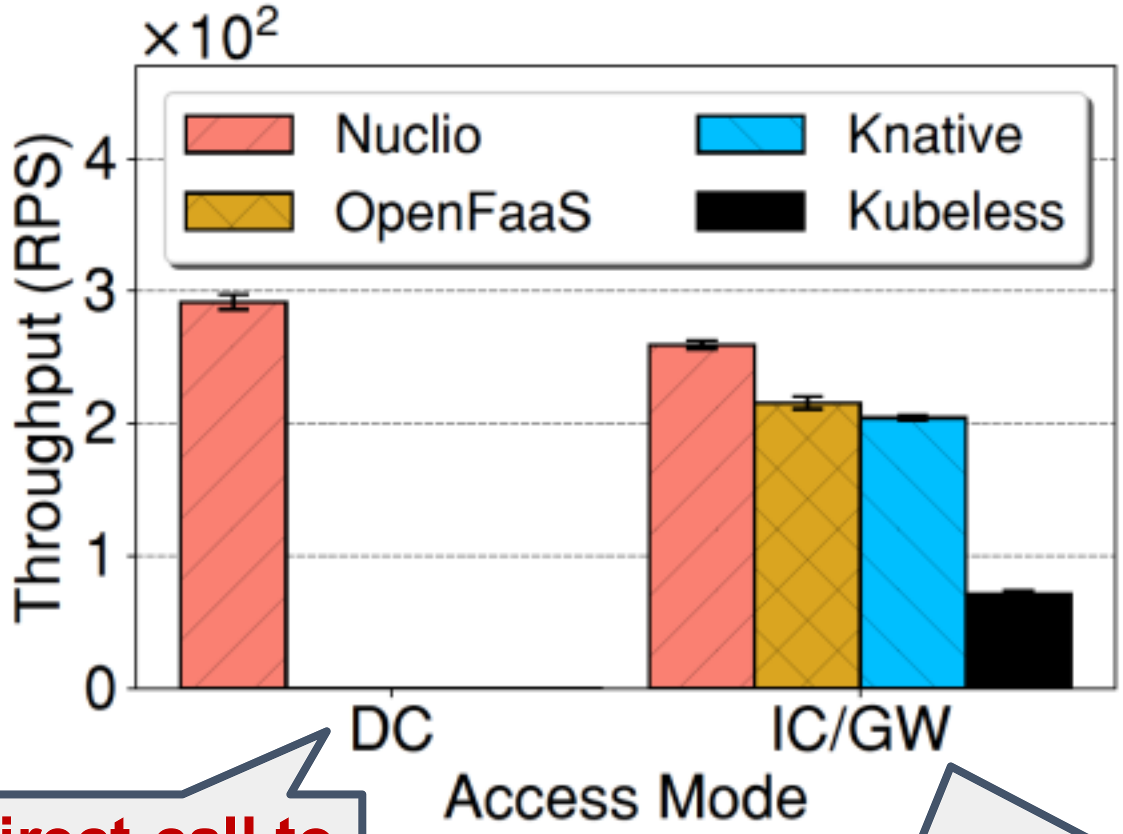
Direct call to func pod (NodePort)

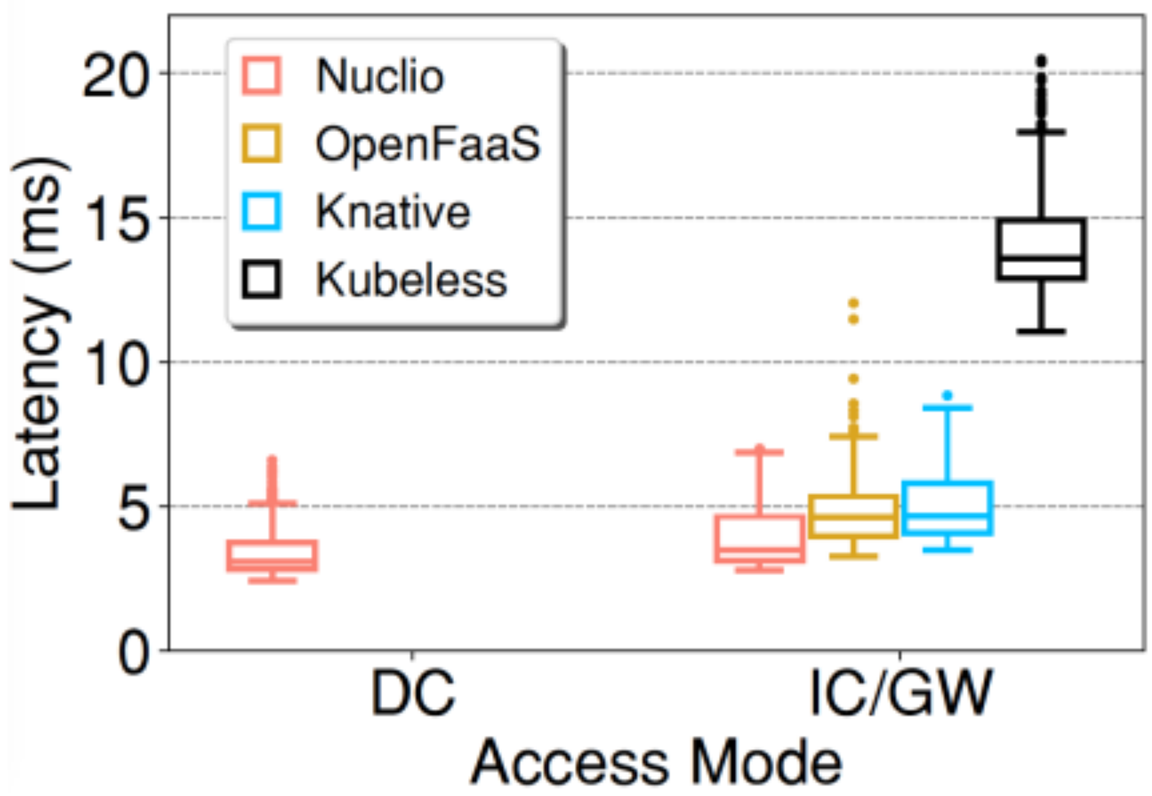
Invoke through IC/GW

Performance

Different modes of exporting services:

IC/GW: Overhead of Ingress Controller/API Gateway.



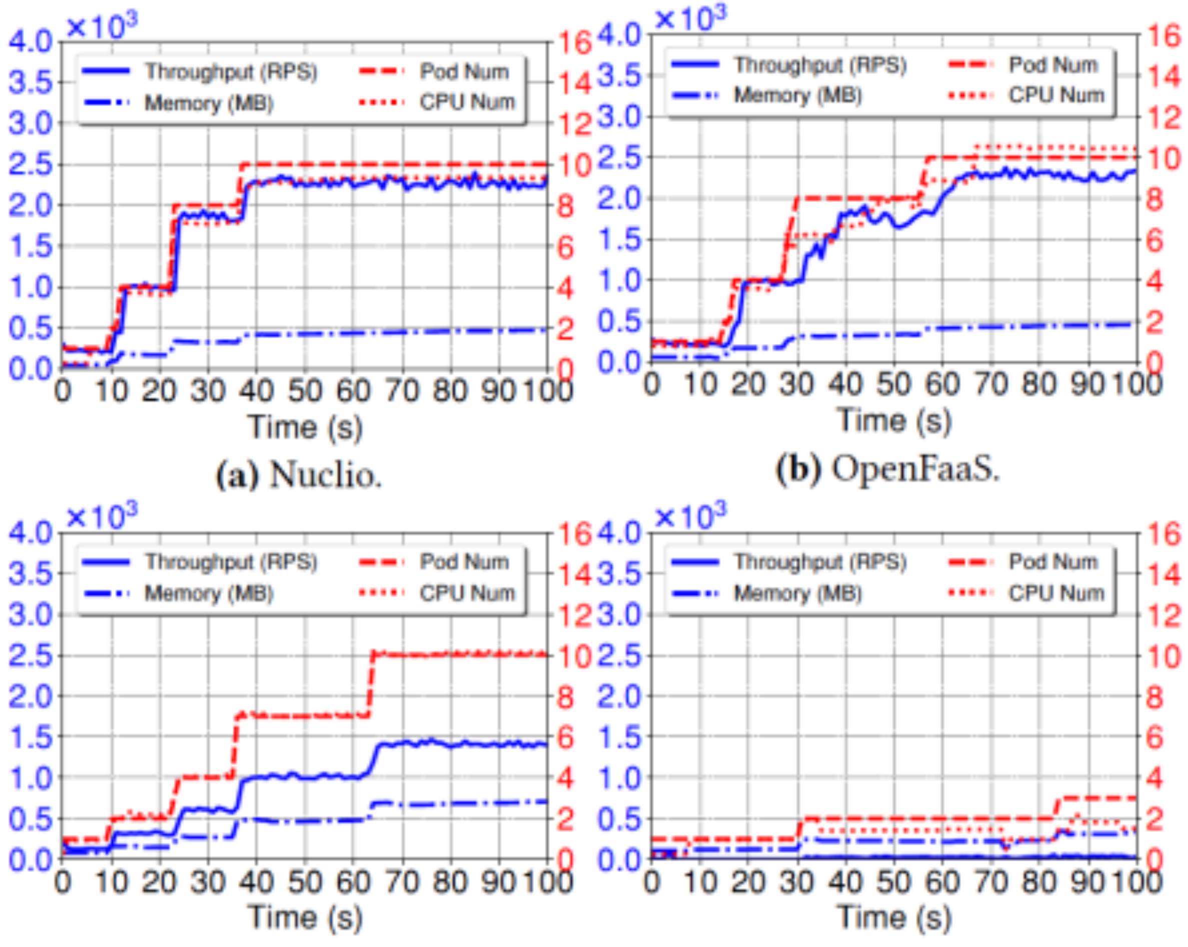


Direct call to func pod (NodePort)

Invoke through IC/GW

Performance: Auto-scaling

Resource-based auto-scaling:

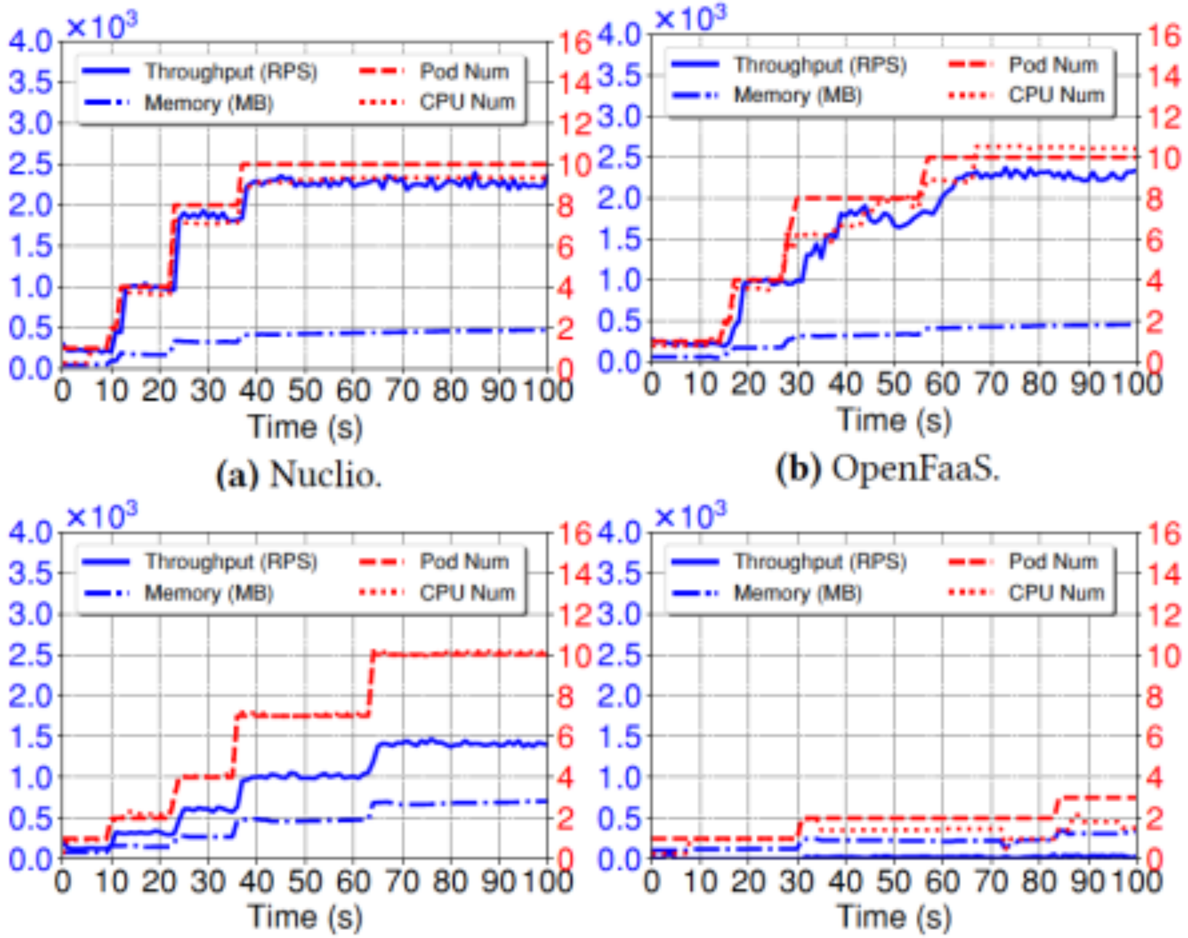


(a) Nuclio. (b) OpenFaaS. (c) Knative. (d) Kubeless.

Resource-based auto-scaling depends on Kubernetes HPA (Horizontal Pod Autoscaler)

Performance: Auto-scaling

Resource-based auto-scaling:



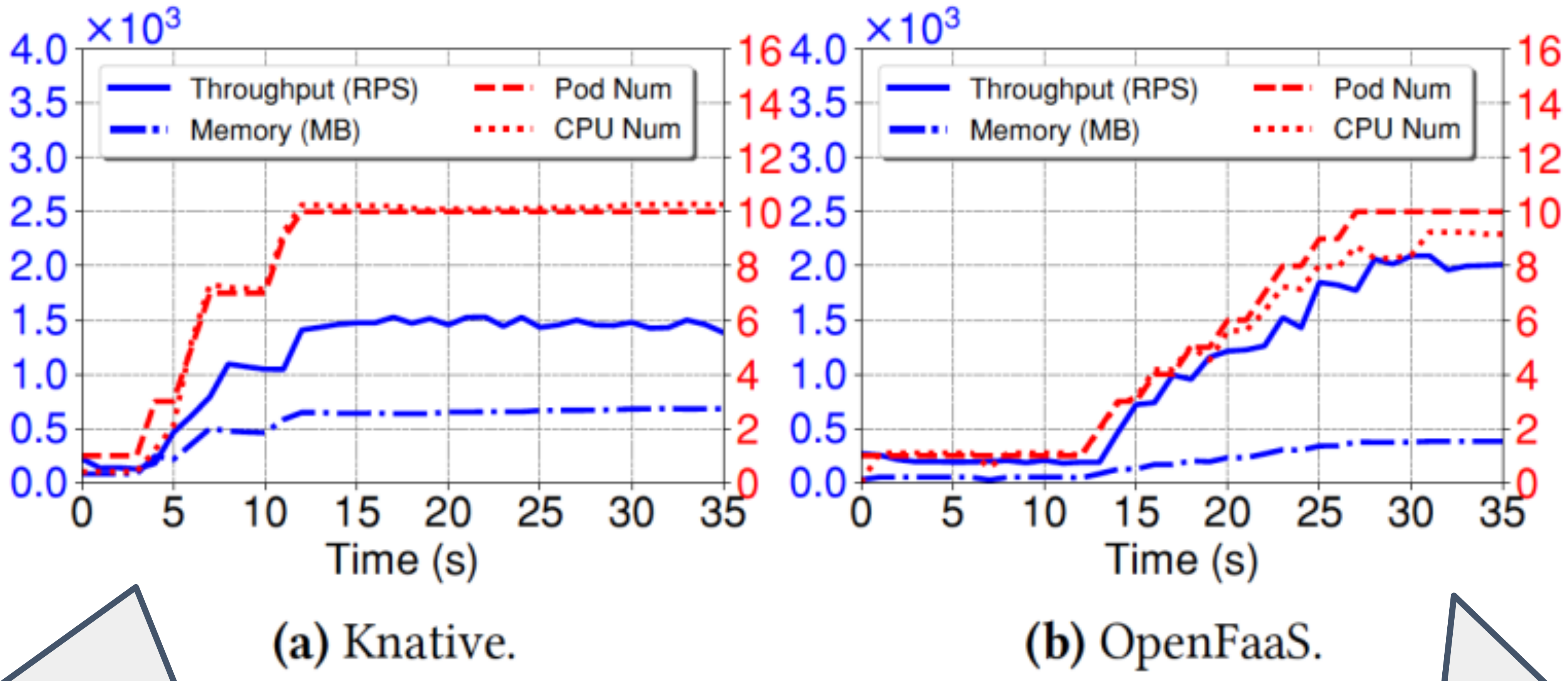
(a) Nuclio. (b) OpenFaaS. (c) Knative. (d) Kubeless.

Resource-based auto-scaling depends on Kubernetes HPA (Horizontal Pod Autoscaler)

In spite of the same function and HPA, platform characteristics govern auto-scaling.
(Different performance
⇒ Different resource utilization
⇒ Different auto-scaling rate)

Performance: Auto-scaling

Workload-based auto-scaling:



o-scaling with steady wor

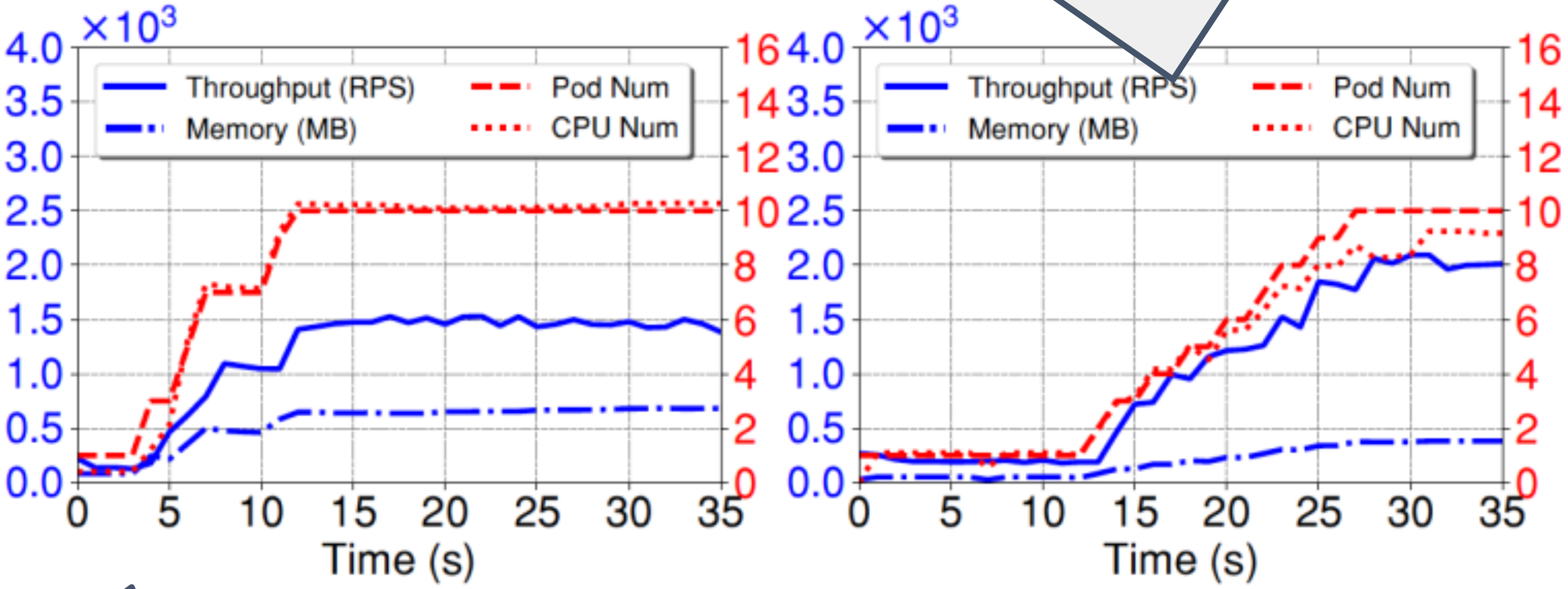
Concurrency-based

RPS-based

Performance: Auto-scaling

Workload-based auto-scaling:

Prometheus reacts slowly
⇒ Slow scaling



(a) Knative.

(b) OpenFaaS.

Concurrency-based

...o-scaling with steady wor...

RPS-based

Performance: Auto-scaling

Issues about load balancing for OpenFaaS:

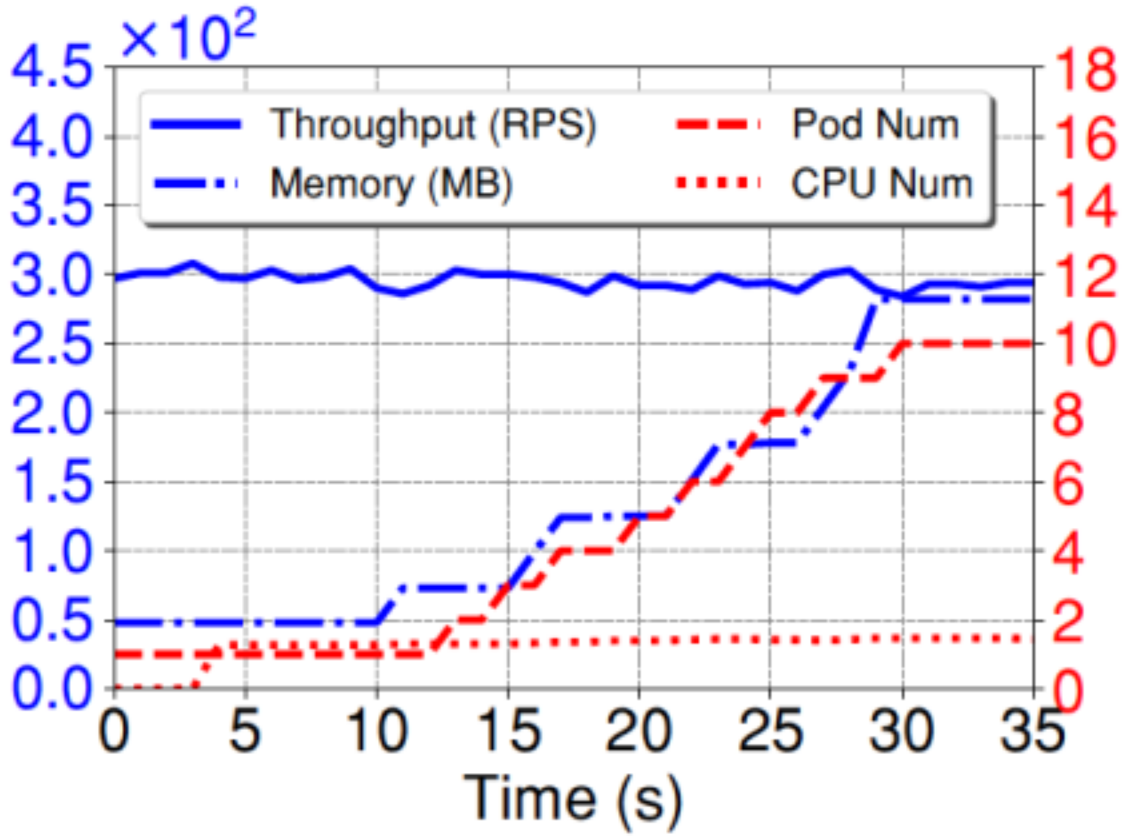


Fig. RPS-based auto-scaling in OpenFaaS

Performance: Auto-scaling

Issues about load balancing for OpenFaaS:

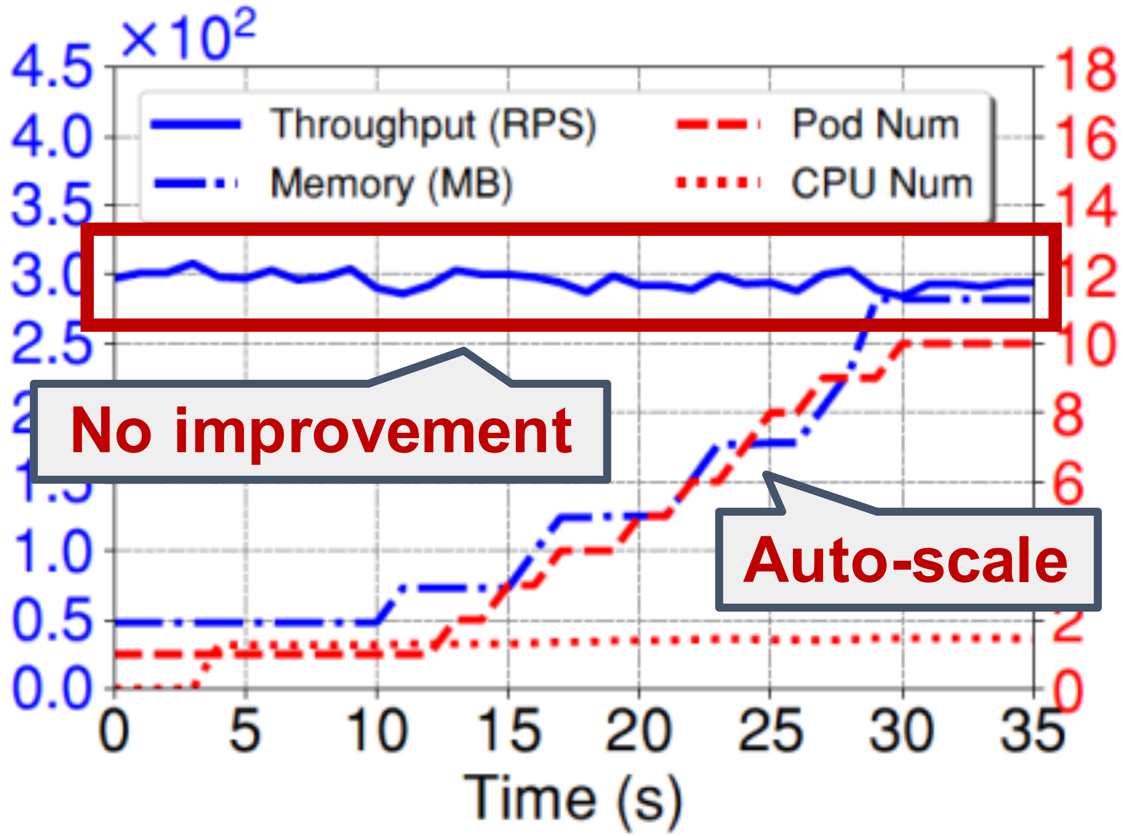


Fig. RPS-based auto-scaling in OpenFaaS

Behavior: Auto-scaling happens but NO performance improvement!

Performance: Auto-scaling

Issues about load balancing for OpenFaaS:

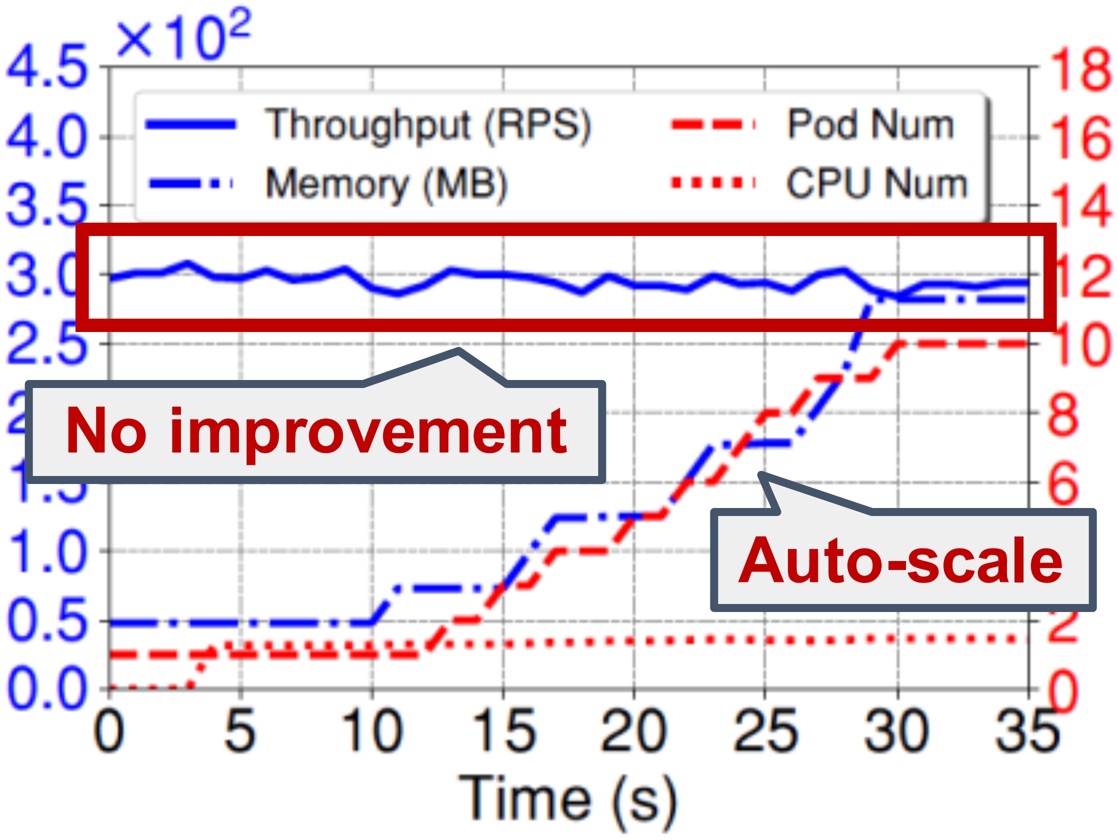


Fig. RPS-based auto-scaling in OpenFaaS

Behavior: Auto-scaling happens but NO performance improvement!

<u>Load-balancing Issue!</u>

If client enables keep-alive, OpenFaaS does not set up connections with newly created function pods, which hinders performance improvement.

Performance: Auto-scaling

Issues about Concurrent-based auto-scaling:

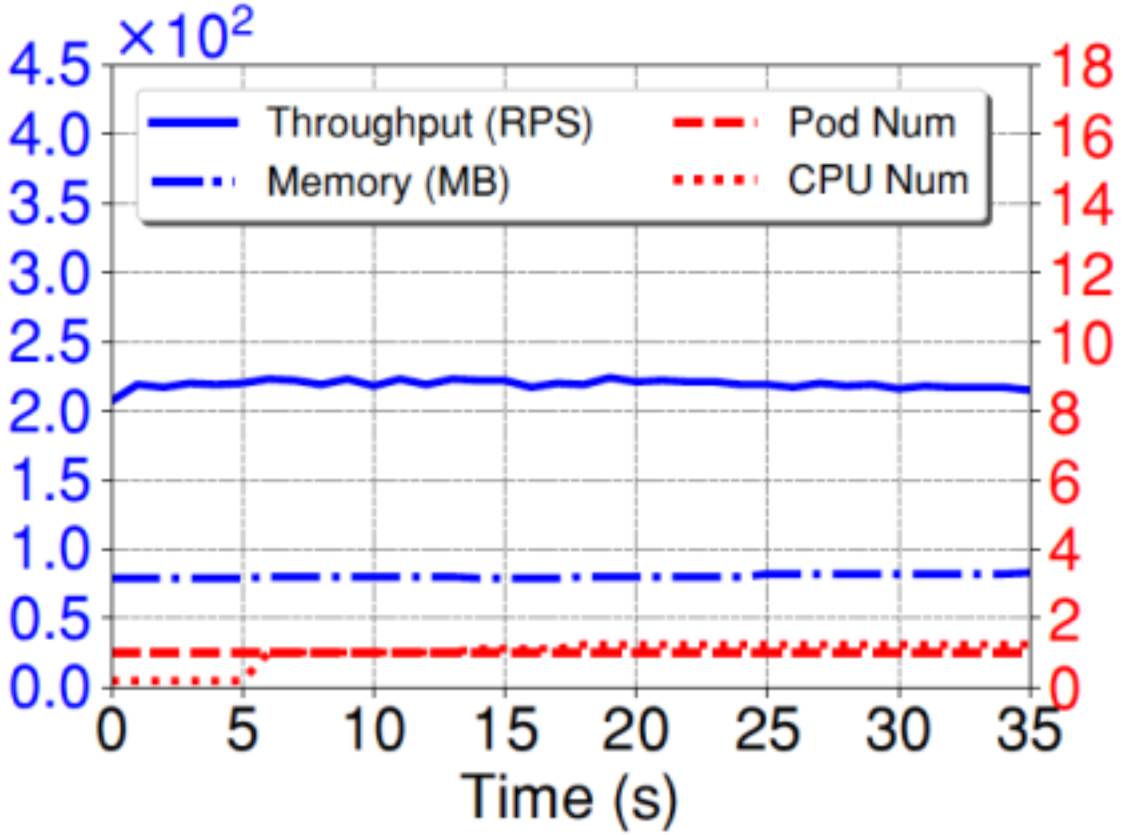


Fig. Conc-based auto-scaling in Knative

Traffic: Conc=9, RPS=400
Configuration: Conc_Threshold=10

Performance: Auto-scaling

Issues about Concurrent-based auto-scaling:

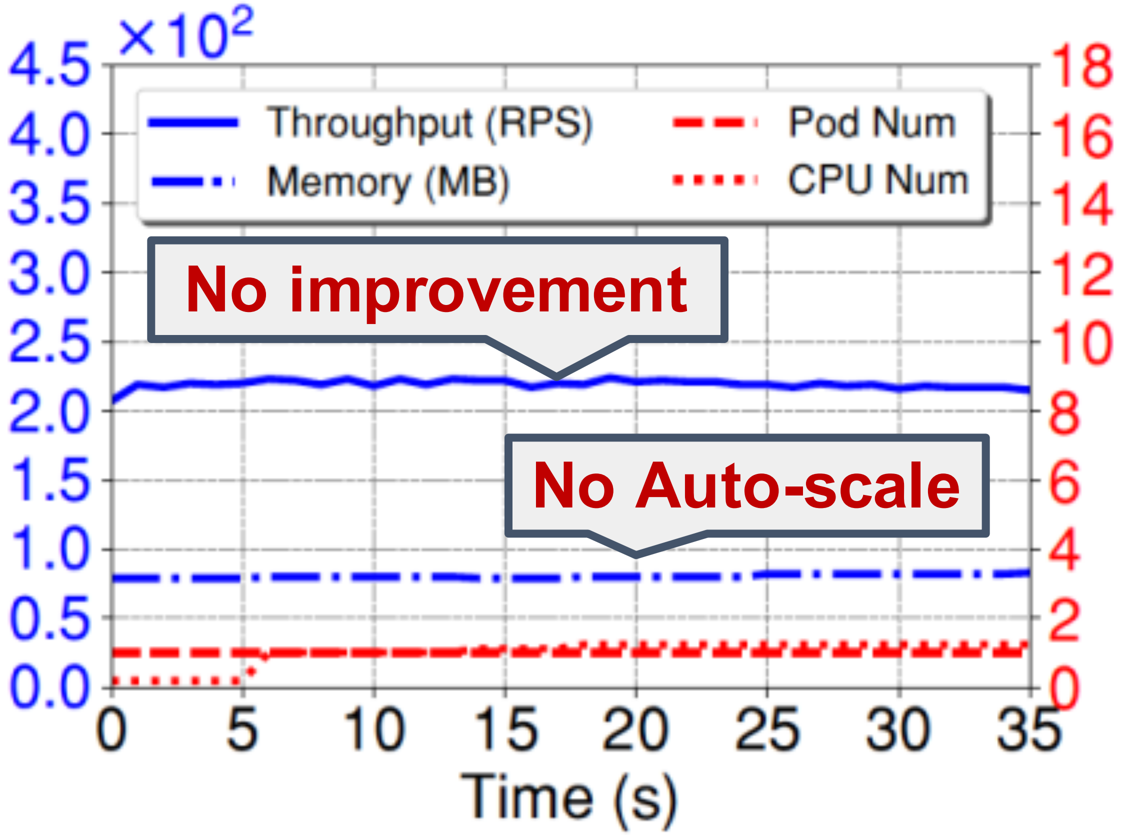


Fig. Conc-based auto-scaling in Knative

Traffic: Conc=9, RPS=400
Configuration: Conc_Threshold=10

Behavior: No Auto-scaling!
No able to scale to 400 RPS (Actual RPS=~220)

Performance: Auto-scaling

Issues about Concurrent-based auto-scaling:

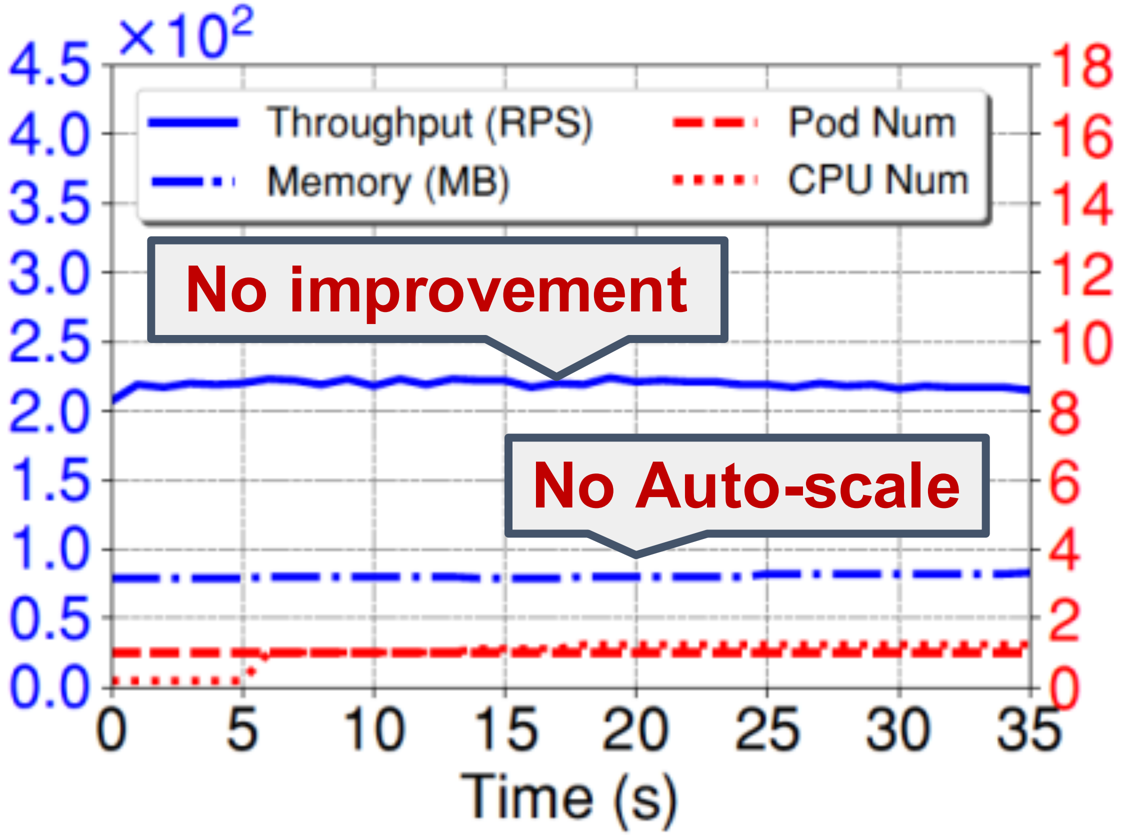


Fig. Conc-based auto-scaling in Knative

Traffic: Conc=9, RPS=400
Configuration: Conc_Threshold=10

Behavior: No Auto-scaling!
No able to scale to 400 RPS (Actual RPS=~220)

Misconfiguration inhibits auto-scaling.
(Conc. does not exceed threshold.
⇒ No auto-scaling with workload of low concurrency but high RPS.)

Key Observations

- Function processing:
  - Multiple workers within one function pod contribute to performance improvement.
  - Pre-fork mode (warm worker) increases the throughput and reduces the latency.

- Load balancing:
  - Plays an important role in the performance and scalability.
  - Coupling routing with load balancing can adversely affect the performance -- Needs greater attention!

Key Observations

- ❖ Autoscaling:
  - ➢ For resource-based auto-scaling, in spite of the same function and HPA, platform characteristics govern auto-scaling.
  - ➢ Misconfiguration of auto-scaling rules can severely degrade the performance and system utilization.
  - ➢ Current Auto-scaling approaches are based only on the total processed requests, while the dropped requests are missed out. -- Incoming request rate needs to be accounted for.

Backup Slides

Serverless 2020 and Beyond

- Load balancing:
  - Improper load balancing results in poor performance improvement -- Needs greater attention!
- Autoscaling:
  - Misconfiguration of auto-scaling rules can severely degrade the performance and system utilization.
  - Current Auto-scaling approaches are based only on the total processed requests, while the dropped requests are missed out. -- Incoming request rate needs to be accounted for.

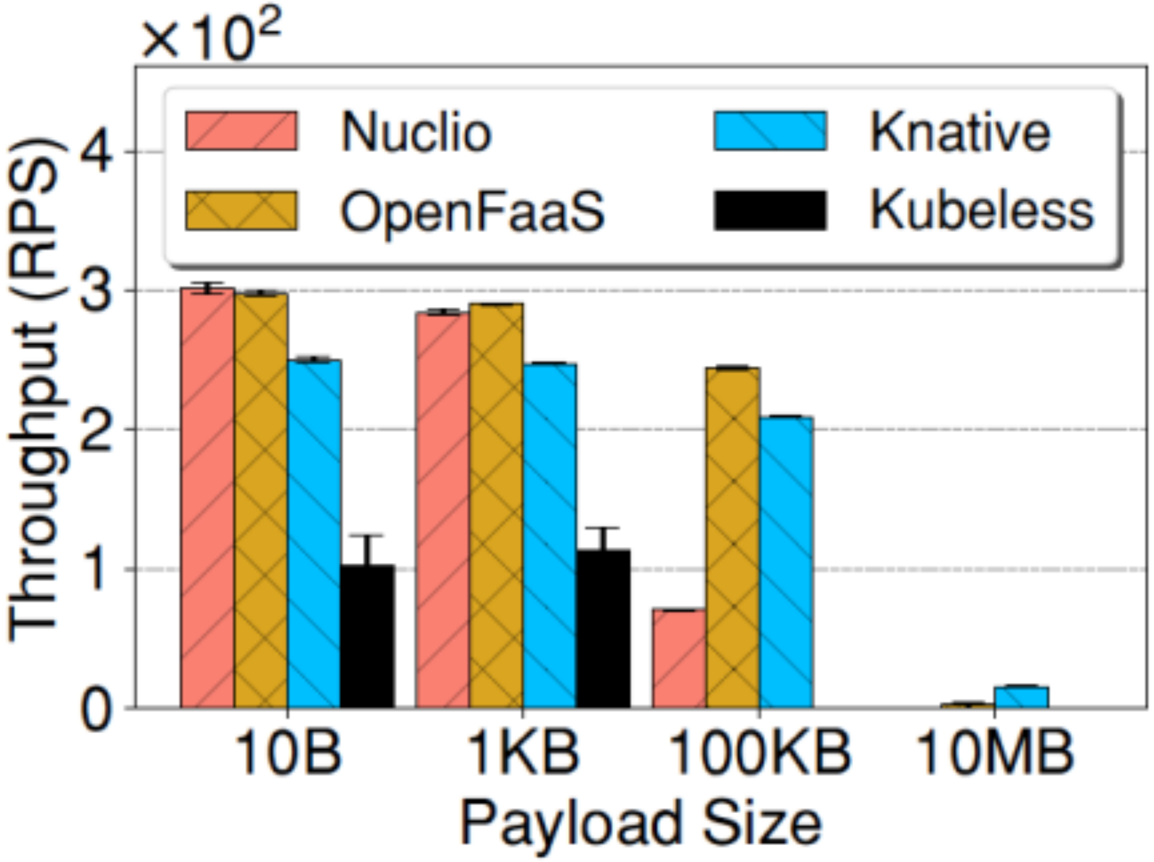
Motivation

- To understand how the serverless platforms work?
- What is the impact of configuration parameters?
- What is the performance of serverless platforms?
- What is the behavior of auto-scaling?

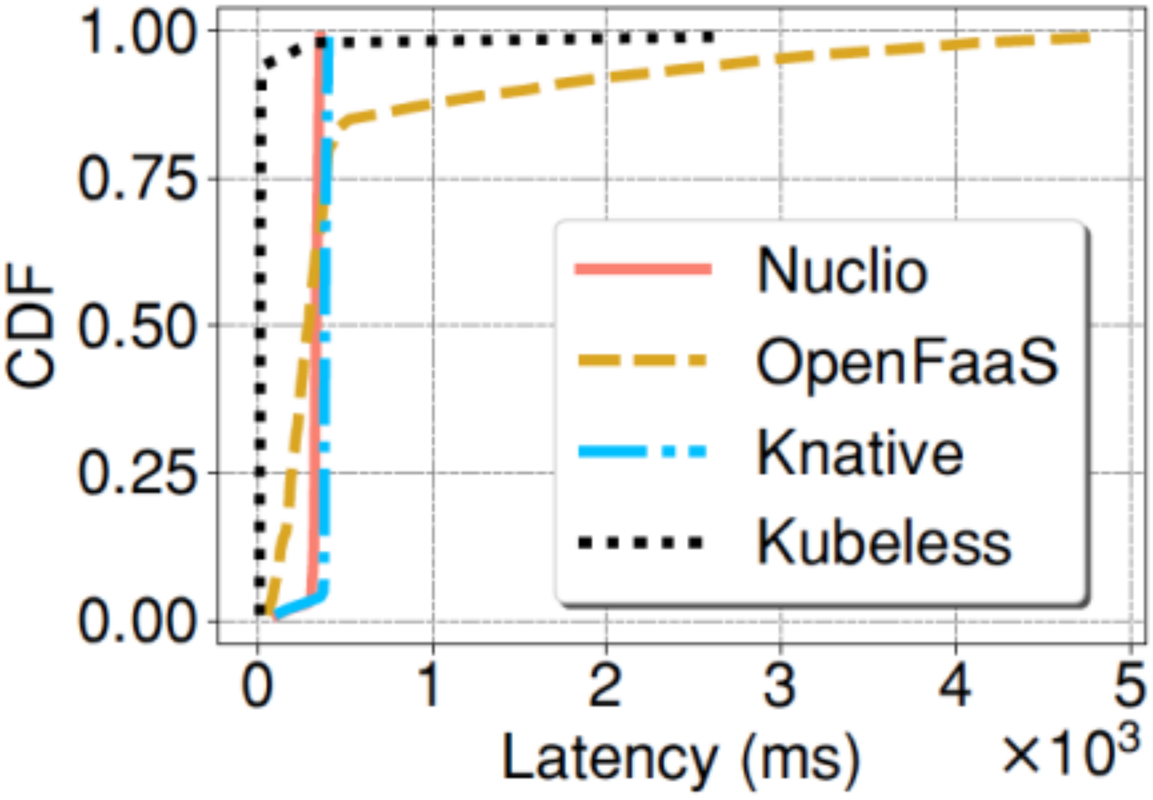
Thank you!

Performance

HTTP Workload: fetch a web page of different sizes



(a) Throughput in requests/second.

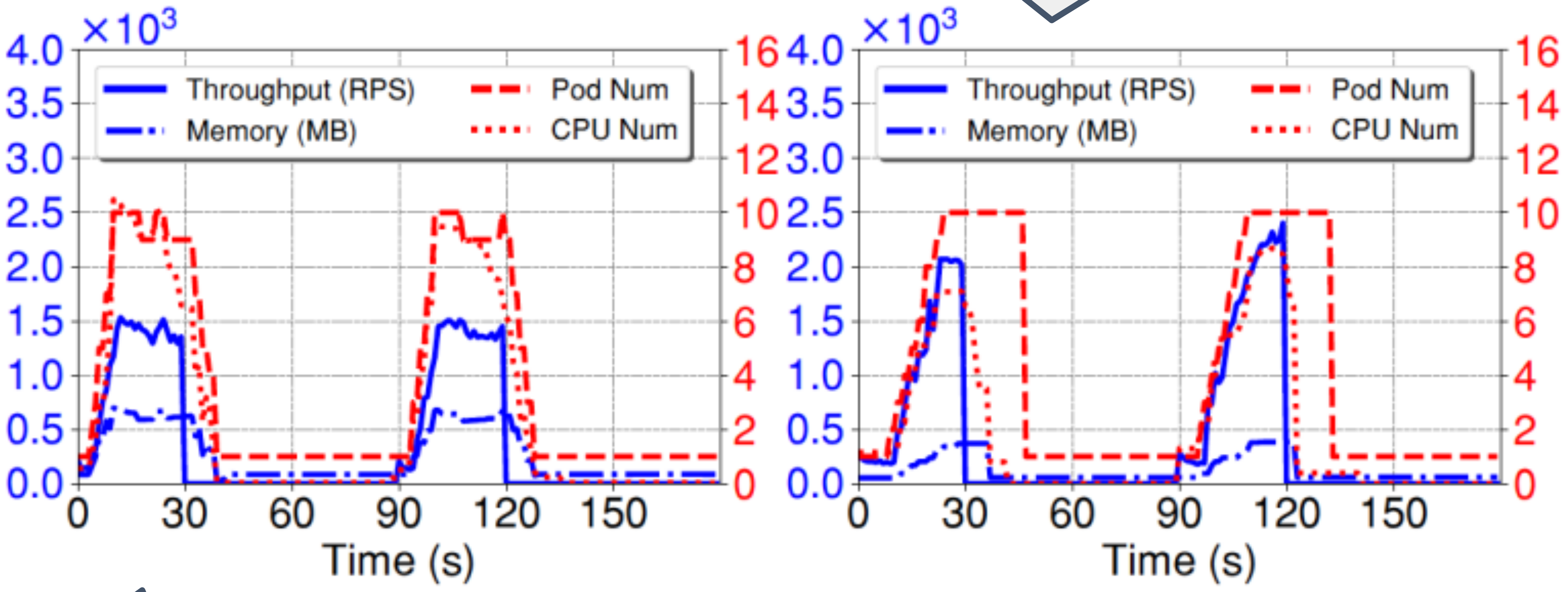


(b) Latency for 1KB payload.

Performance: Auto-scaling

Prometheus ⇒ React slowly

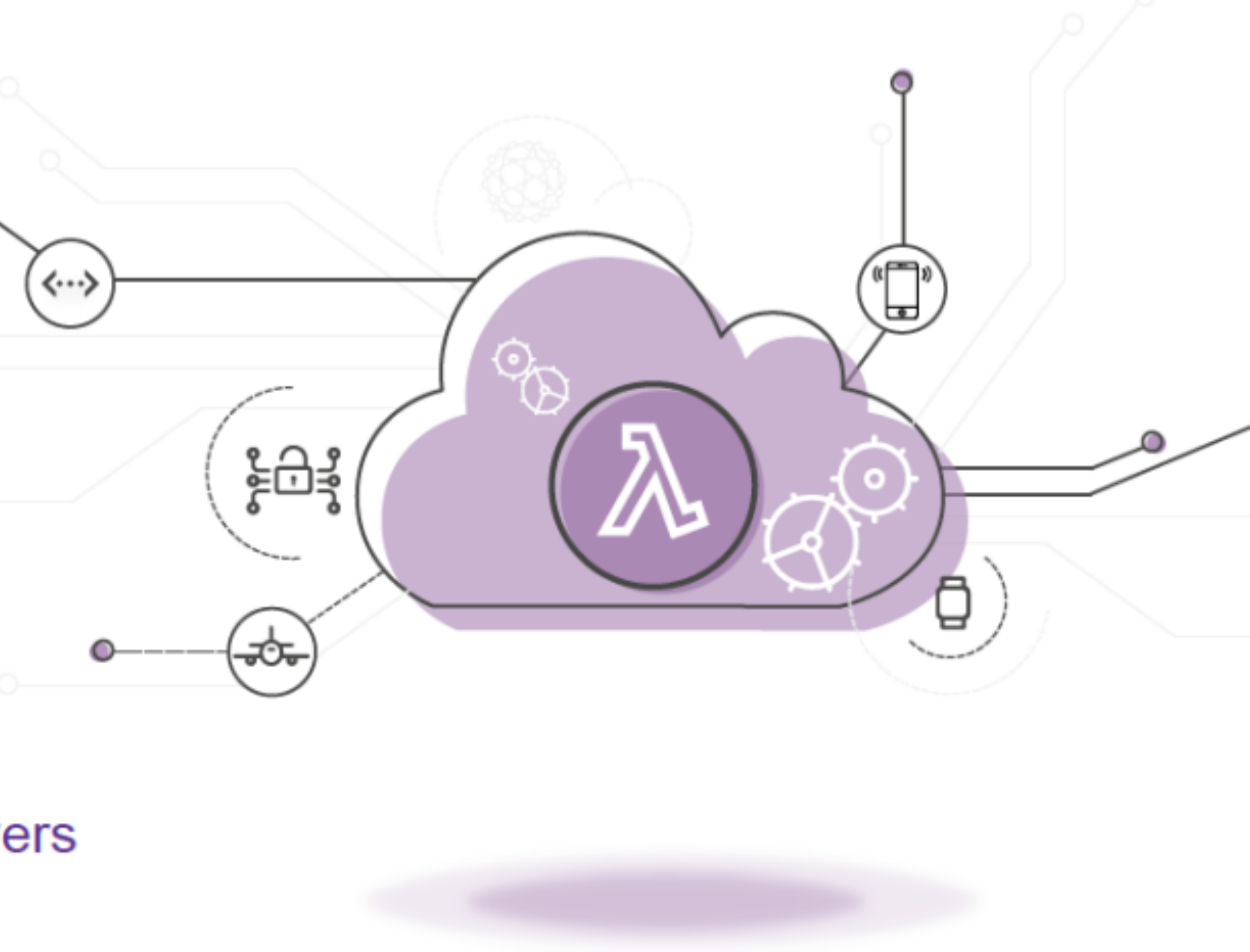
Workload-based auto-scaling: bursty w… ad



(a) Knative. (b) OpenFaaS.

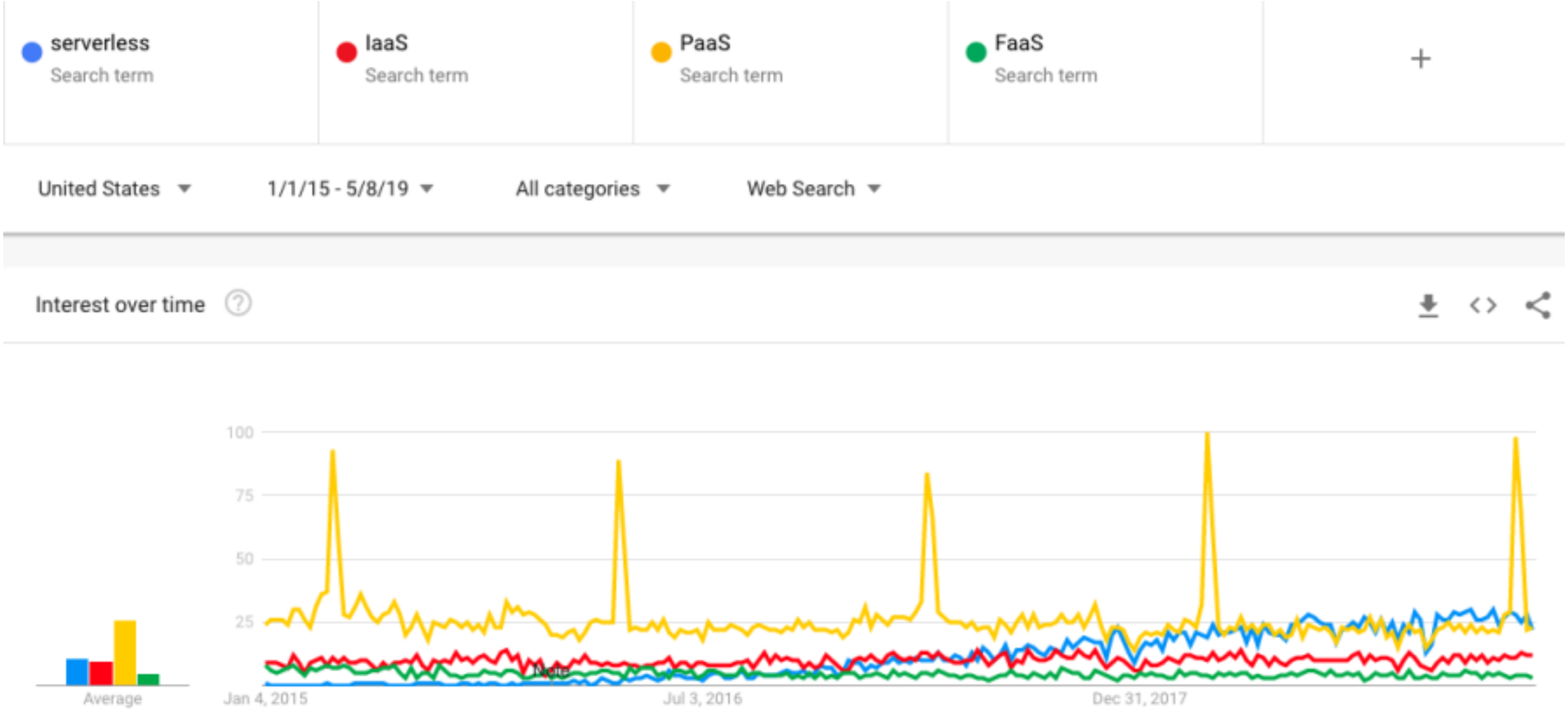
Concurrency-based o-scaling with bursty worl **RPS-based**

What is serverless?



Build and run applications without thinking about servers

Serverless Computing: The New Hotness



https://trends.google.com/trends/explore?date=2015-01-01%202019-05-08&geo=US&q=serverless,IaaS,PaaS,FaaS