Scheduling of Serverless Functions

Latest researches and approaches

By: Aryan Ebrahimpour

Professor: Dr. Entezari Maleki

دانشگاه علم و صنعت ایران

Table of Contents

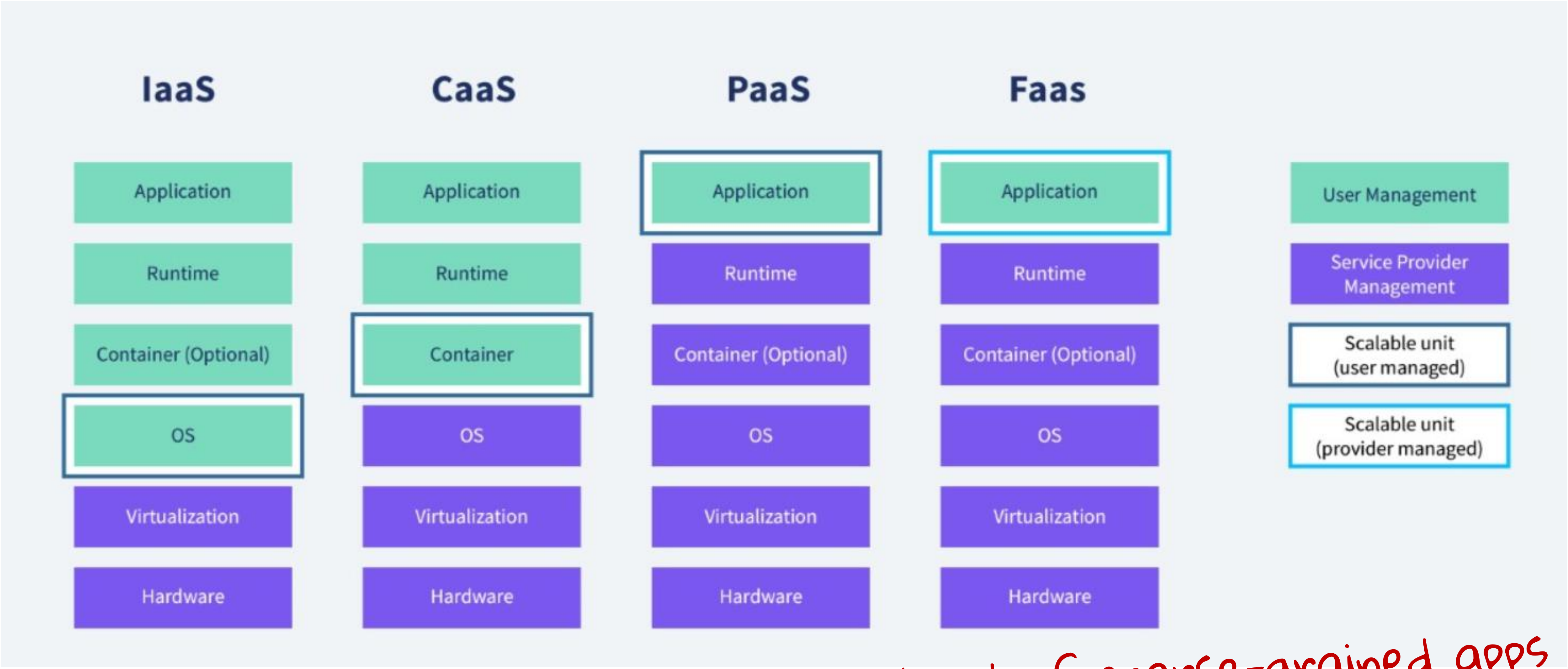
What are FaaS and Serverless?

Provider-User scopes, and FaaS vs BaaS

What are FaaS and Serverless?

- FaaS stands for “Function as a Service”.
- Its popularity comes from popularity of Containers and Microservices.
- Industrialized by Amazon with Amazon Lambda back in 2014.

What are FaaS and Serverless?



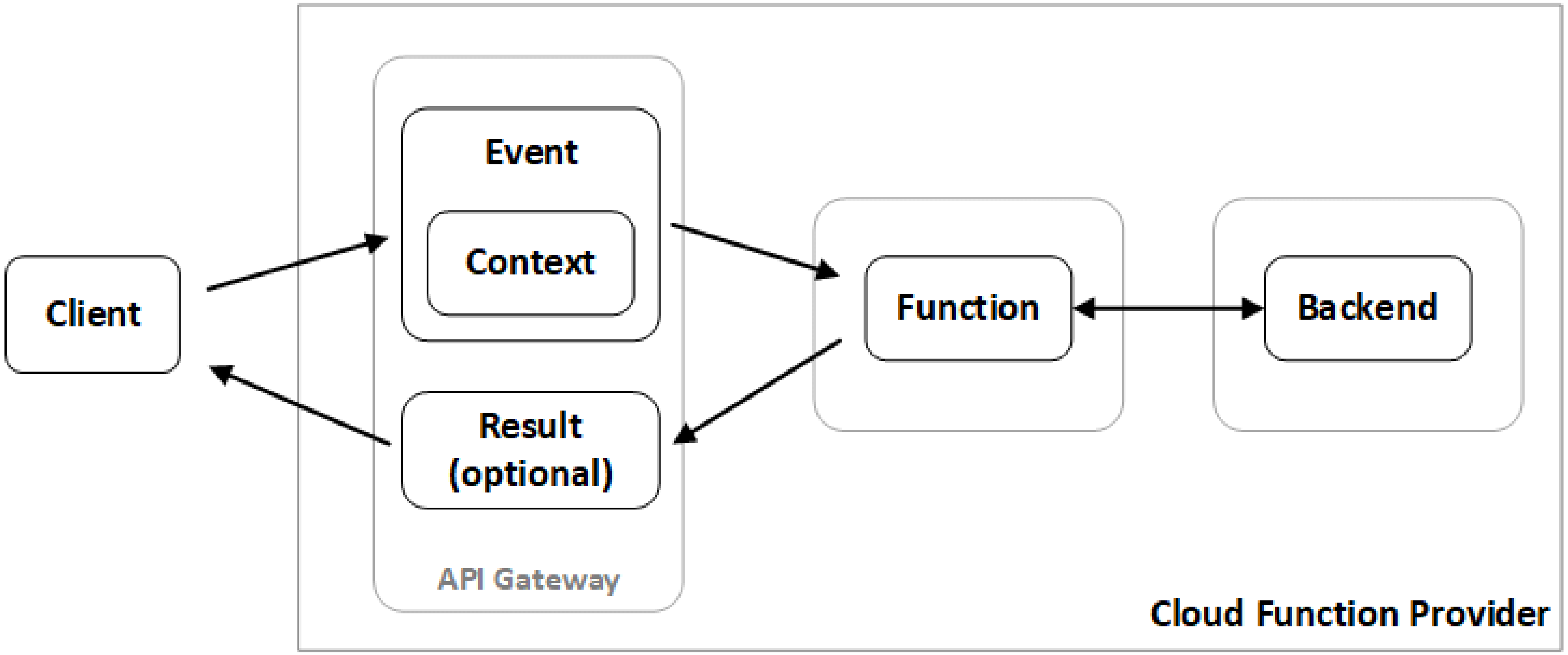
fine-grained functions instead of coarse-grained apps

What are FaaS and Serverless?

- The terms “Serverless” and “FaaS” are used interchangeably.
- However they are not quiet the same.
- Serverless doesn’t mean you don’t have servers.
- FaaS is one type of Serverless computing.

What are FaaS and Serverless?

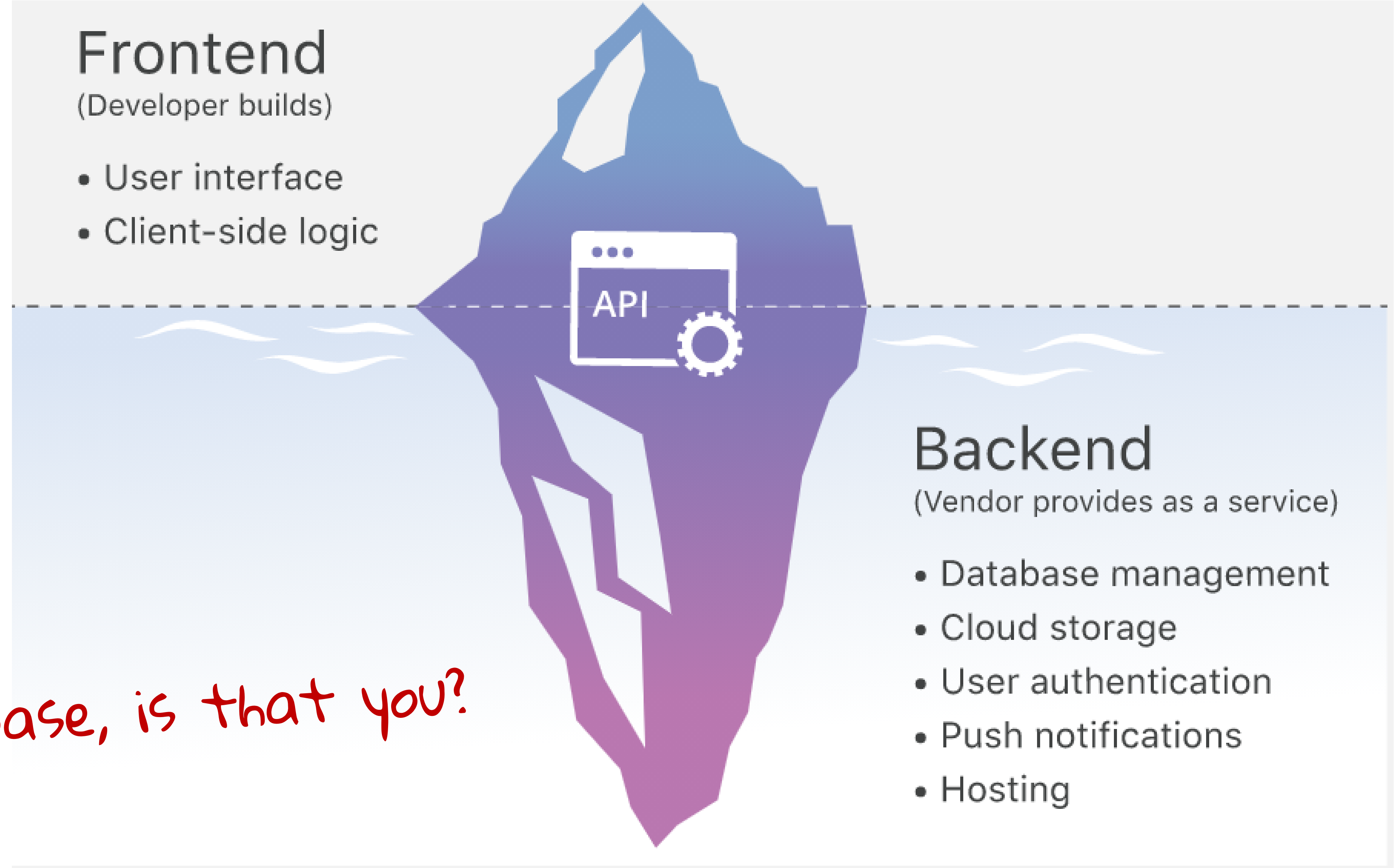
- FaaS is event-driven.



What are FaaS and Serverless?

- Don’t confuse it with “BaaS”.
- In BaaS, you’ll develop your front app, while having the backend already up and running in the cloud.
- Some say BaaS and FaaS are both serverless computing.

What are FaaS and Serverless?



Firebase, is that you?

Challenges

Why scheduling for FaaS matters?

Challenges

- You might say we already have scheduling approaches that actually work.
- Then why do we need FaaS specific scheduling systems?
- Other methods don’t seem to very well fit this model.

Challenges

- Cloud platforms offer cheap and scalable services.
- However FaaS should be even more scalable, and less costlier than other services.
- This will end up with challenges that provider should address.

Challenges

Cost

As it should be cheaper, scheduler should be even more efficient by not running for excess time. Warm up time is considerable.

Challenges

Isolation

Functions are stateless. They must be isolated from environment even more than other models.

Challenges

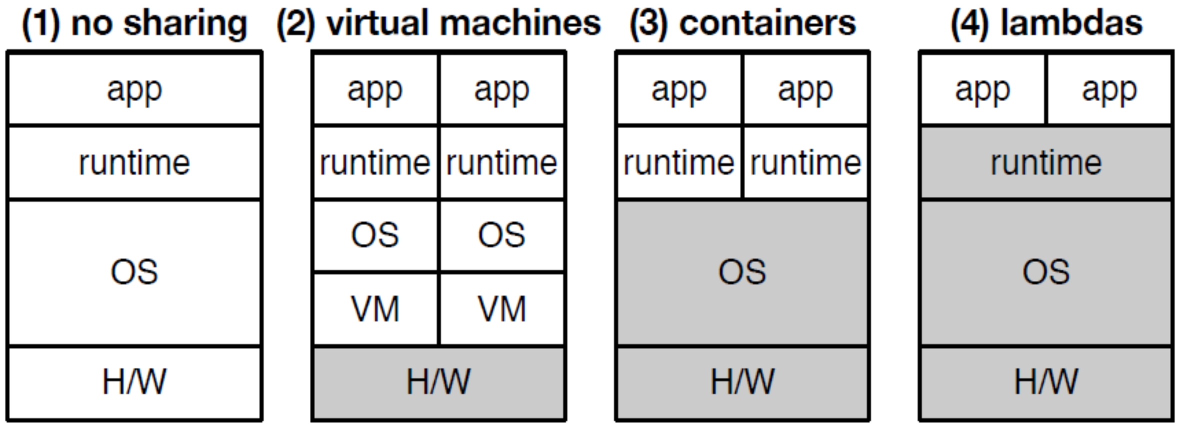
Orchestration and Load balancing

Prediction of function calls must be considered to have a better management on the number of up and running containers for running the function.

Challenges

Security

Many functions from different customers might be running on a single physical or virtual machines, and sometimes even on a container forked from the same base container. This leads to some security concerns.



Challenges

Scheduling

A good scheduling algorithm directly affects Cost, Response Time and Scalability of these cloud functions.

Challenges

📚 Others

There are also other challenges like vendor lock-in problem, old software, IDEs and dev tools integration, state management, etc.

However we focus on Scheduling.

Goals

What are we aiming to improve with these algorithms?

Goals

Cold-start Time

This time includes:

1. Fetching information
2. Creating container
3. Startup time
4. Loading packages time

Goals

📚 Response Time

Time spent since activating the function (usually from an HTTP event) until the final response.

Goals

Throughput

Number of responses received from the cloud function provider in a unit of time of a simulated environment.

Goals

Final cost

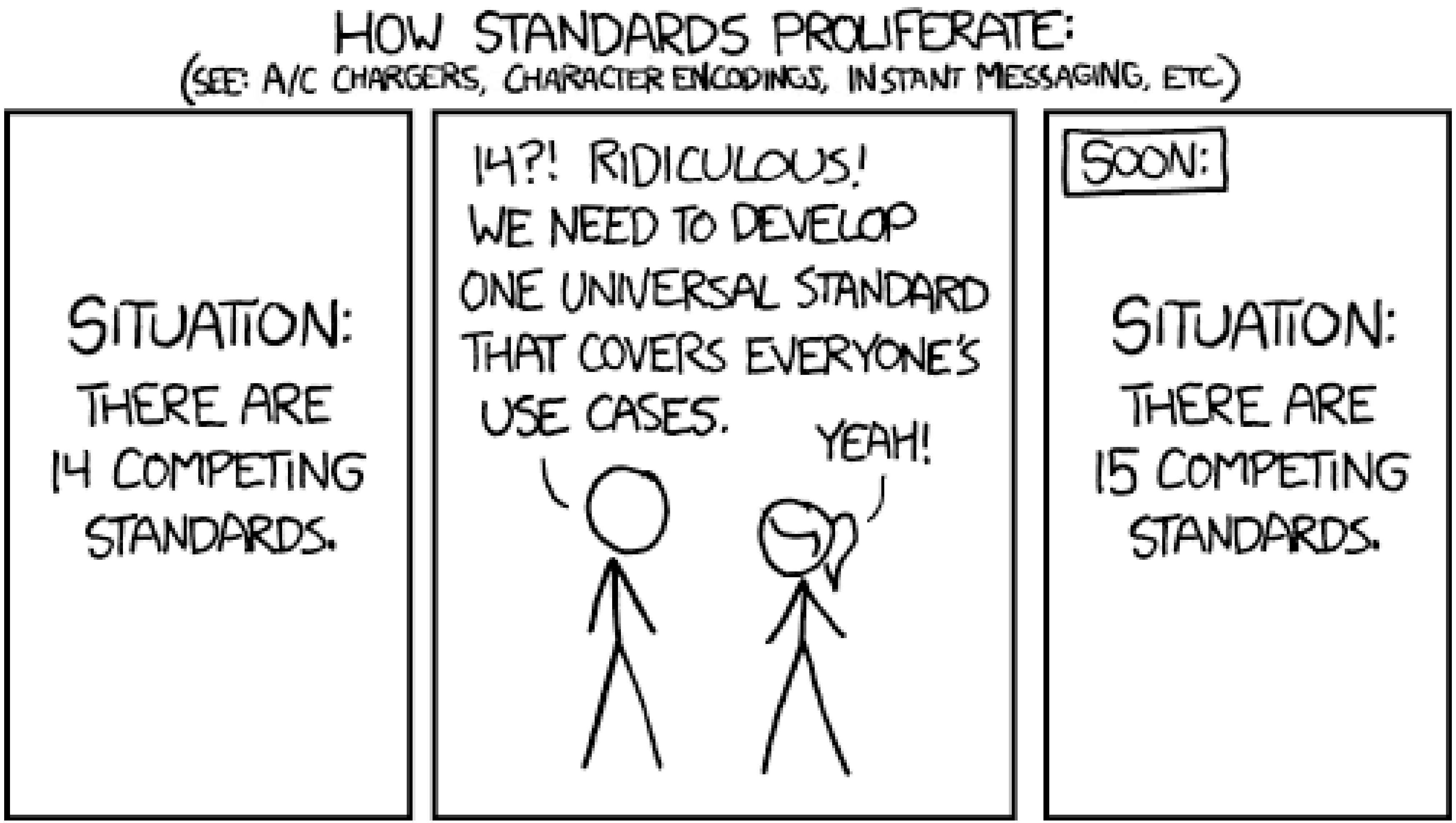
Even though it is highly affected by the other factors that are already mentioned, algorithm can also consider the user budget.

Platforms

Some of the platforms

Source	Site	Owner	Platform Name
proprietary	https://aws.amazon.com/lambda	Amazon	Amazon Lambda
proprietary	https://azure.microsoft.com/en-us/services/functions	Microsoft	Azure Functions
proprietary	https://cloud.google.com/functions	Google	GCP Functions
open source	https://kubeless.io	kubeless	kubeless
open source	https://www.openfaas.com	OpenFaaS Ltd	Open FaaS
proprietary	https://www.ibm.com/uk-en/cloud/functions	IBM	IBM Cloud Functions
open source	https://openwhisk.apache.org	Apache	Open Whisk
open source	https://fission.io	Platform9	Fission
open source	https://open.iron.io	Iron	IronFunctions
open source	https://fnproject.io	Oracle	Fn Project
open source	https://knix.io	Nokia Bell Labs	KNIX
open source	https://github.com/open-lambda	OpenLambda	Open-lambda
open source	https://knative.dev	Google	Knative
proprietary	https://www.alibabacloud.com/products/function-compute	Alibaba	AlibabaCloud Function Compute
proprietary	https://vercel.com/docs/serverless-functions/introduction	Vercel	Vercel Serverless Functions
open source	https://nuclio.io	Iguazio	Nuclio
proprietary	https://www.slappforge.com/sigma	SLAppForge Inc	Sigma

Why even bother knowing the platforms?



Approaches

In what ways people tried to address these challenges?

Approaches

- **Cold Start** approaches: address cold start via optimized package loading
- Optimization for **Scientific workflows**
- **Parallel** approaches

Cold-start approach

Approaches > Cold Start

- 📚 Loading packages and software dependencies take a lot of time for starting container.
- 📚 Technologies that are facing dependency-hell problems.

Approaches > Cold Start

- Harter et all., changed the Linux kernel to optimize containers performance.
- These container preload the packages that are needed to run the software, and lazy load the others.
- They’ve shown 76% of the startup time is related to package loading.
- They’ve succeeded to increase production cycle by 5x and development cycle 20x.
- Many systems can use this to boost their startup time.

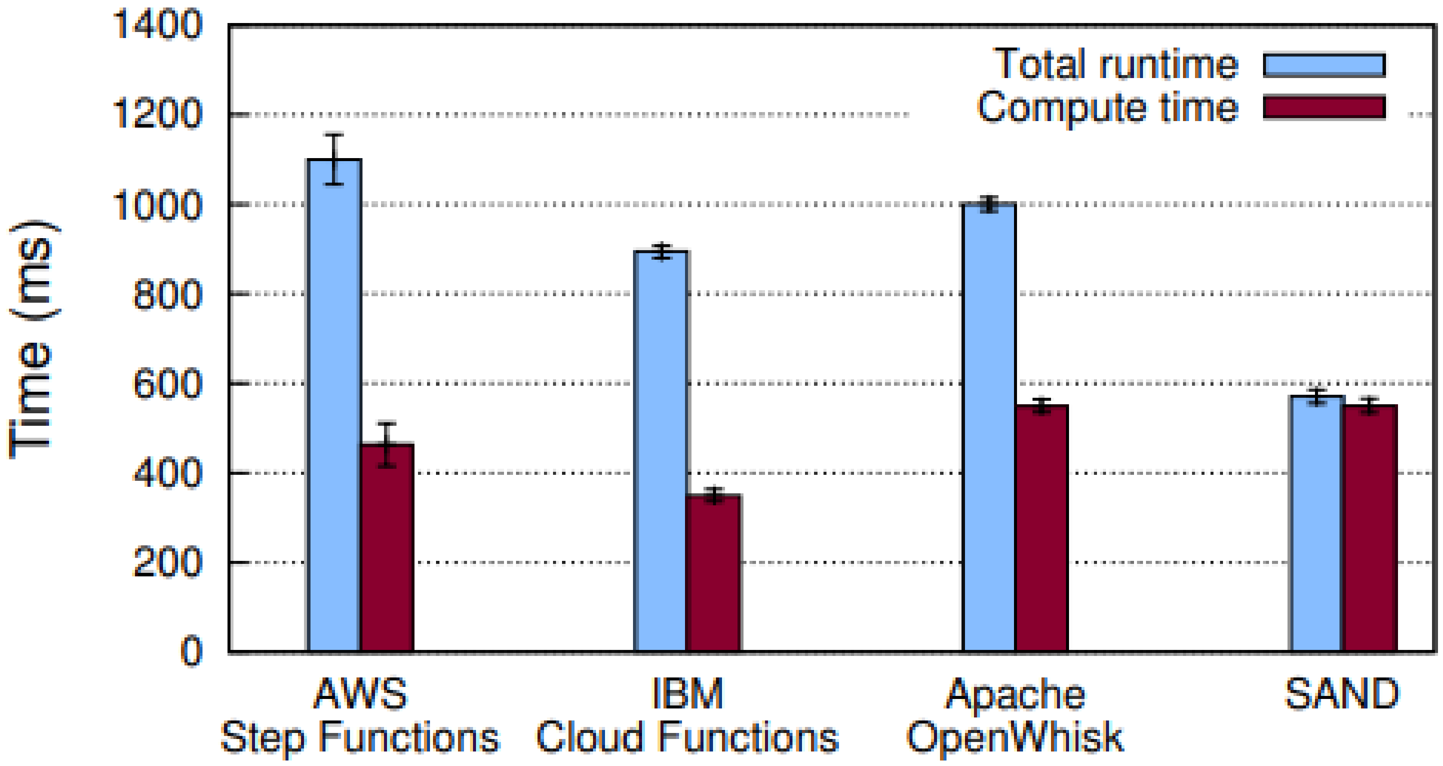
T. Harter et al., "Slacker: Fast distribution with lazy docker containers," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*, 2016, Santa Clara, United States, pp. 181-195.

Approaches > Cold Start

- Akkus et al., proposed SAND. A model of isolation to address cold-start.
- They modeled the problem to Apps and Functions.
- Apps are in different containers while their functions are in the same container, but different processes by forking the warm container.
- Some other methods took this idea, and created a warm python interpreter with the most used libraries preloaded.

I.E. Akkus et al., "SAND: Towards High-Performance Serverless Computing," in Usenix Annual Technical Conference (18), July-2019, Boston, USA, pp. 923-935.

Approaches > Cold Start

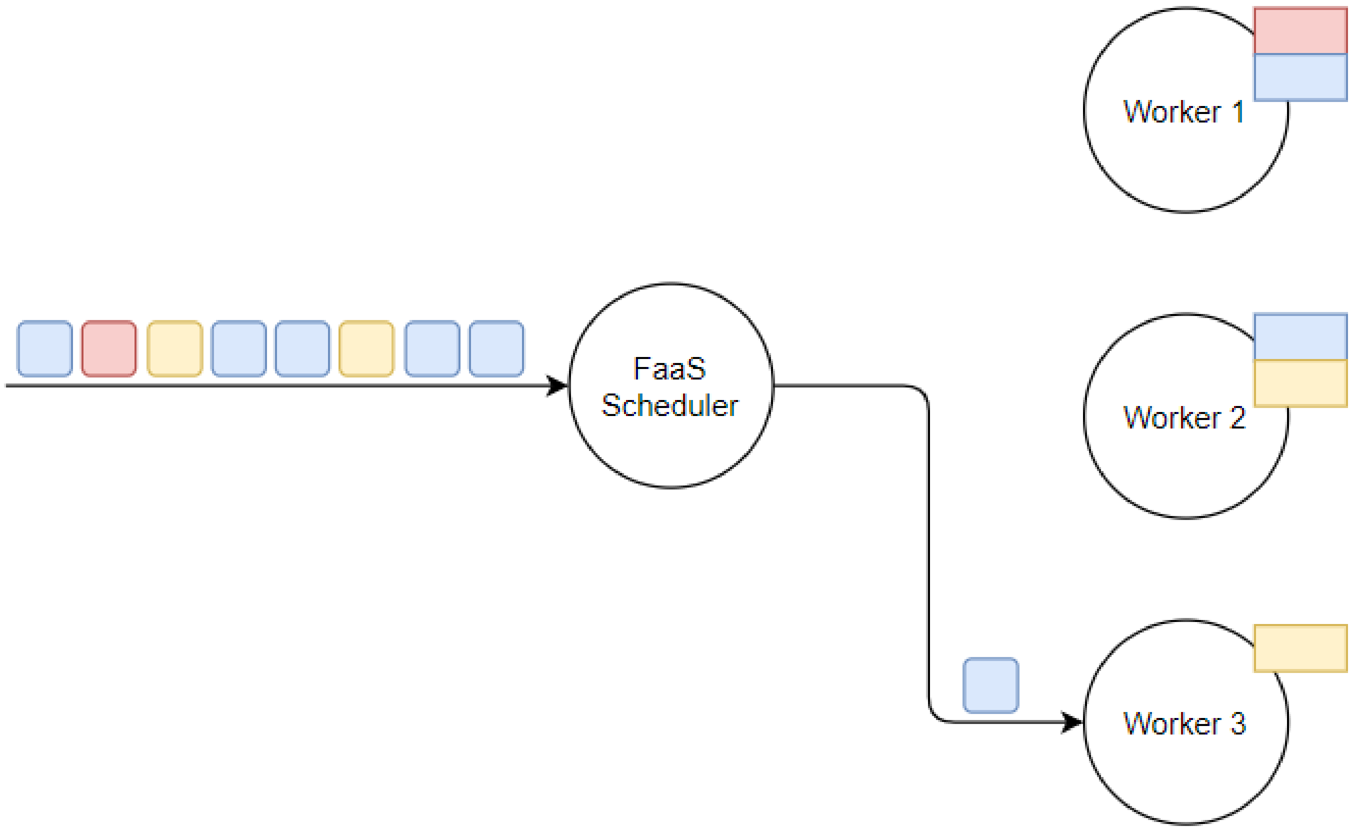


Approaches > Cold Start

- Aumala et al., also addressed huge packages and libraries problem.
- They bundled the packages with compute nodes instead of functions.
- Functions run on the nodes that already have the most heavy package loaded in cache.
- However, this method only considers the heaviest package.

G. Aumala et al., "Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms," in 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID) , May-2019, Larnaca, Cyprus, pp. 282-291.

Approaches > Cold Start



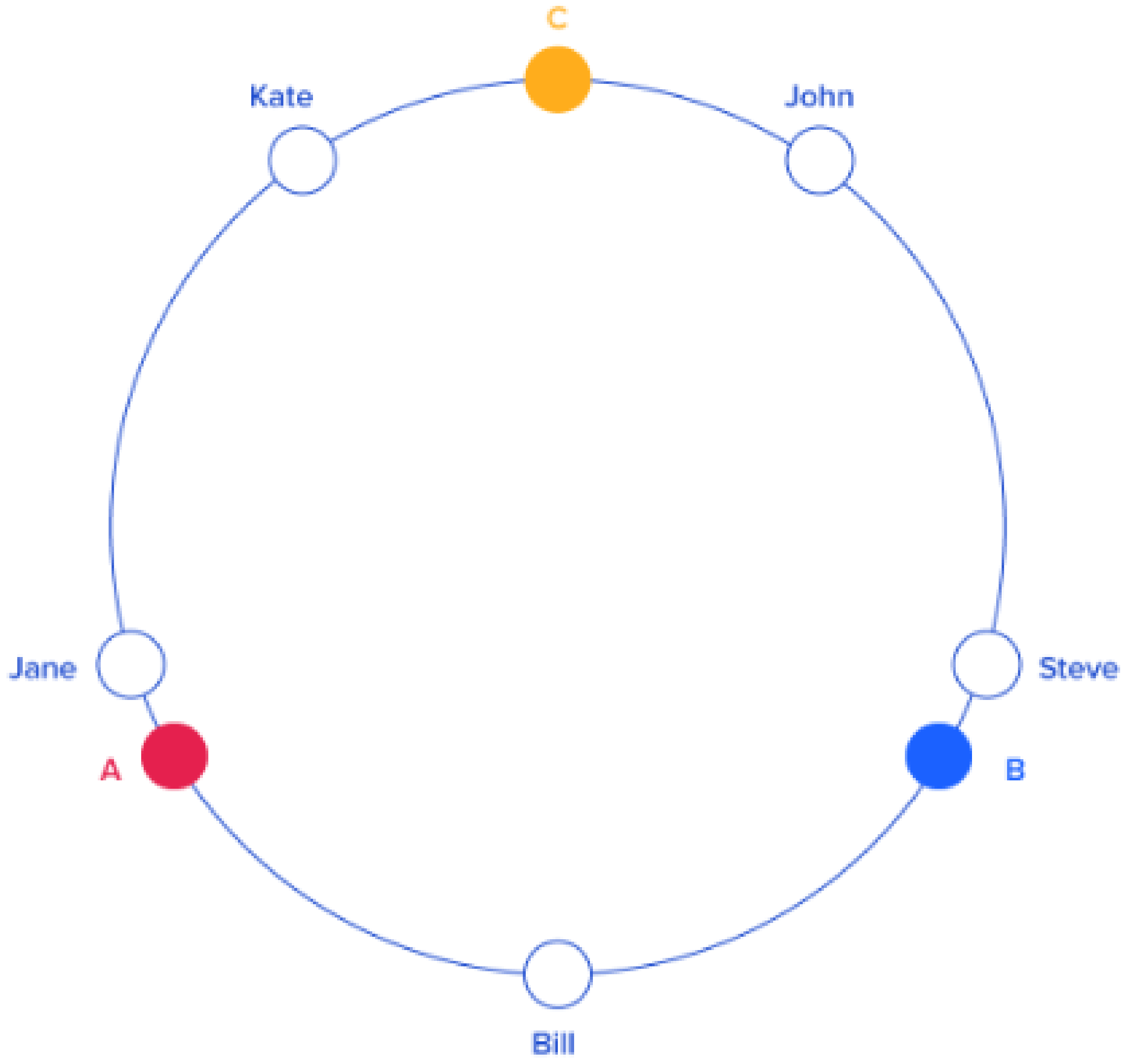
Approaches > Cold Start

- Aumala's method is based on two algorithms:
    1. Consistent Hashing
    2. Power of two choices technique

G. Aumala et al., "Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms," in 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID) , May-2019, Larnaca, Cyprus, pp. 282-291.

Approaches > Cold Start

consistent hashing,
scalable & easy to use



Approaches > Cold Start

- Aumala's method is based on two algorithms:
    1. Consistent Hashing
    2. Power of two choices technique
- Power of two choices selects two random nodes if there are multiple candidates, then picks the least loaded one.
- If the selected machine is overloaded already, algorithm fallbacks to the least loaded mechanism.

G. Aumala et al., "Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms," in 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID) , May-2019, Larnaca, Cyprus, pp. 282-291.

Algorithm 1: Package-aware scheduler algorithm (PASch)

Global data: List of workers, $W = \{w_1, ..., w_n\}$, and their load thresholds, $T = \{t_1, ..., t_n\}$, mapping function M

Input: Function, f, largest required package, p

if (p *is not null*)**then**

  `/* Get affinity workers */`

  $\langle a1, a2 \rangle = M(p)$

  `/* Select target with least load */`

  if ($load(w_{a1}) < load(w_{a2})$)**then**

    $A := a1$

  else

    $A := a2$

  `/* If target is not overloaded, we are done */`

  if ($load(w_A) < t_A$)**then**

    Assign f to w_A

    return

`/* Balance load */`

Assign f to least loaded worker, w_i

Algorithm 2: Mapping function, M; given a package, returns two affinity nodes.

Global data: A consistent hash implementation, *consistent*, and value to be added to the package ID to map a second affinity worker to it, *salt*

Input: Package id, p

Output: Affinity workers for p, $\langle a1, a2 \rangle$

`/* Get two affinity workers */`

$a1 = \text{consistentHash.get}(p)$

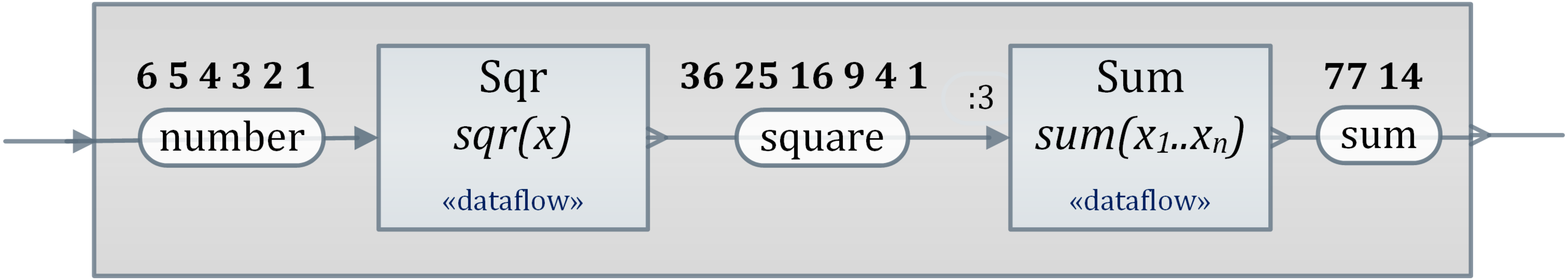
$a2 = \text{consistentHash.get}(p + salt)$

return $\langle a1, a2 \rangle$

Scientific Workflow approach

Approaches > Scientific Workflow

- As scientific workflows are important use case of FaaS, there are approaches to specifically address these type of cloud functions.
- But what are Scientific Workflows?
- They are a set of computations (usually mathematical) that are represented as DAGs.
- G = <T, E, Data> where T = {t1, t2, …} are tasks, and E are dependency edges.
- The E set ensures that a child task shouldn't start before its parent finishes.


6 5 4 3 2 1
number
Sqr
sqr(x)
«dataflow»
36 25 16 9 4 1
square
:3
Sum
sum(x1..xn)
«dataflow»
77 14
sum

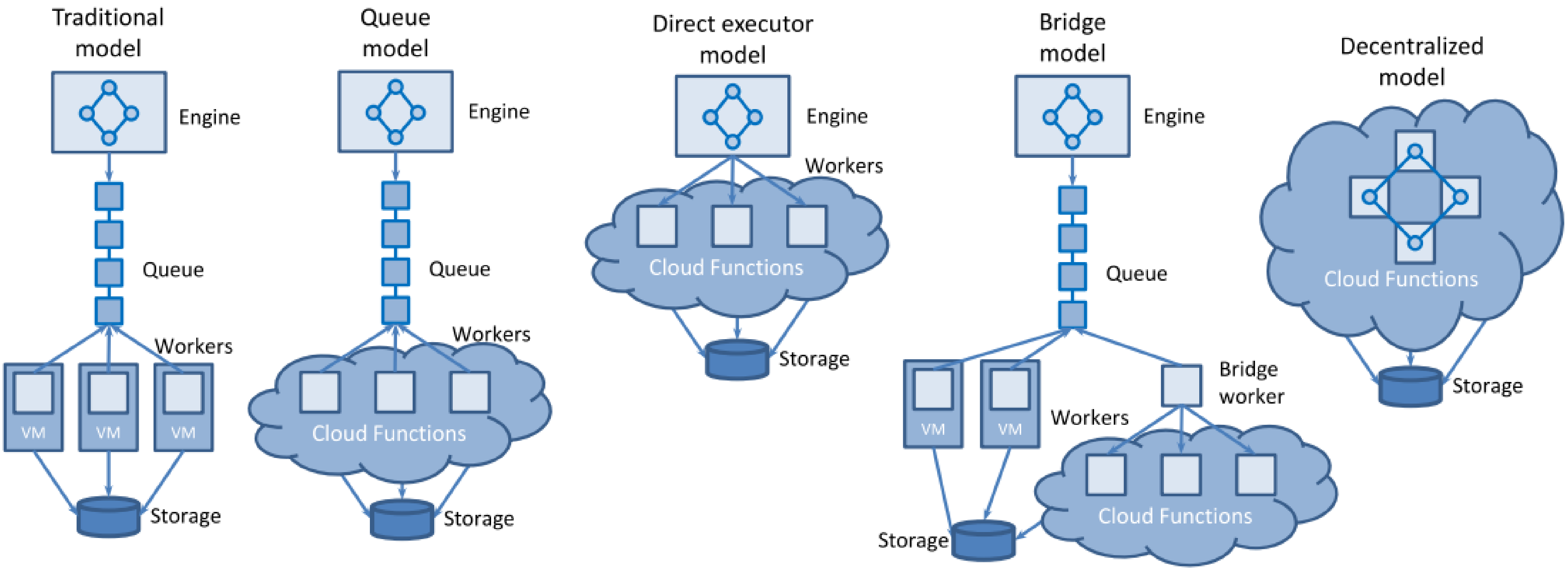
Approaches > Scientific Workflow

- Malawski et al., analyzed different providers to see if their cloud function services fit for scientific workflows.
- They developed a scientific workflow runner on them.
- They’ve shown 5 models and options for developing such systems.

M. Malawski et al., "Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions," Future Generation Computer Systems, vol. 110, pp. 502-514, September 2020.

Approaches > Scientific Workflow

5 models to handle scientific workflows



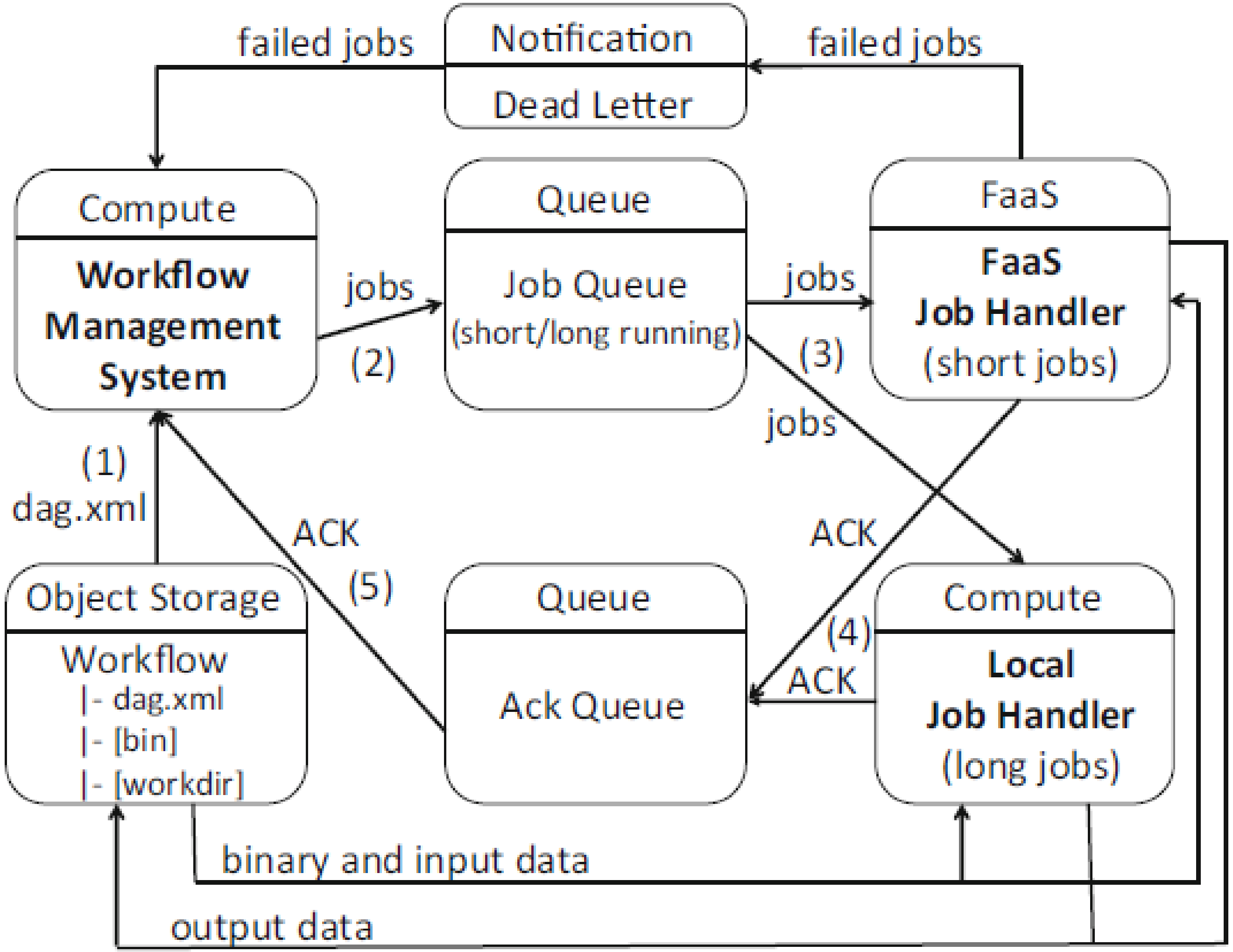
Approaches > Scientific Workflow

- Jiang et al., created DEWE (Distributed Elastic Workflow Execution), a scheduler for running scalable scientific workflows with high performance.
- It distinguishes the jobs to Short jobs and Long jobs.

M. Malawski et al., "Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions," Future Generation Computer Systems, vol. 110, pp. 502-514, September 2020.

Approaches > Scientific Workflow

Death letter contains diagnostics data



DEWE

Approaches > Scientific Workflow

- Kijak et al., in their paper “Challenges for scheduling scientific workflows on cloud functions” introduced a trade-off between deadline and budget.
- This made possible by Hyperflow engine.

J. Kijak et al., "Challenges for scheduling scientific workflows on cloud functions," in 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), Jul-2018, San Francisco, USA, pp. 460-467.

Approaches > Scientific Workflow

Require: DAG, time (D_{user}) and budget (B_{user})

Sort all tasks based on their level
if $B_{user} < Cost_{low}(DAG)$ **then**
  return no possible schedule
else if $B_{user} > Cost_{high}(DAG)$ **then**
  return schedule map on the most expensive resource
end if
Compute the sub-deadline value for each task
while there is unscheduled task **do**
  for $r \in$ resources **do**
    Calculate quality measure $Q(t_{cur}, r)$
  end for
  $r_{selected} \Leftarrow r$ with highest quality measure
  assign t_{cur} to $r_{selected}$
end while
return schedule map

$$l(t_i) = 1 + \max_{t_p \epsilon predecessors(t_i)} l(t_p),$$

$$Level^{j}_{execution} = \max_{l(t_i)==j} \{ET_{max}(t_i)\}$$

$$Level^{j}_{DL} = Level^{j-1}_{DL} + D_{user} * \frac{Level^{j}_{execution}}{\sum_{1 \le j' \le l(t_{exit})} Level^{j'}_{execution}}$$

$$S_{DL}(t_{cur}) = \{Level^{j}_{DL} | l(t_i) == j\}$$

$$Time_Q(t_{cur}, r) = \frac{\xi * S_{DL}(t_{cur}) - FT(t_{cur}, r)}{FT_{max}(t_{cur}) - FT_{min}(t_{cur})}$$

$$Cost_Q(t_{cur}, r) = \frac{Cost_{max}(t_{cur}) - Cost(t_{cur}, r)}{Cost_{max}(t_{cur}) - Cost_{min}(t_{cur})} * \xi$$

where

$$\xi = \begin{cases} 1 & \text{if } FT(t_{cur}, r) < S_{DL}(t_{cur}) \\ 0 & \text{otherwise} \end{cases}$$

$$Q(t_{cur}, r) = Time_Q(t_{cur}, r) * (1 - C_F) + Cost_Q(t_{cur}, r) * C_F$$

where C_F, a cost-efficient factor is a tradeoff factor defined as:

$$C_F = \frac{Cost_{low}(DAG)}{B_{user}}$$

Parallel approach

Approaches > Parallel

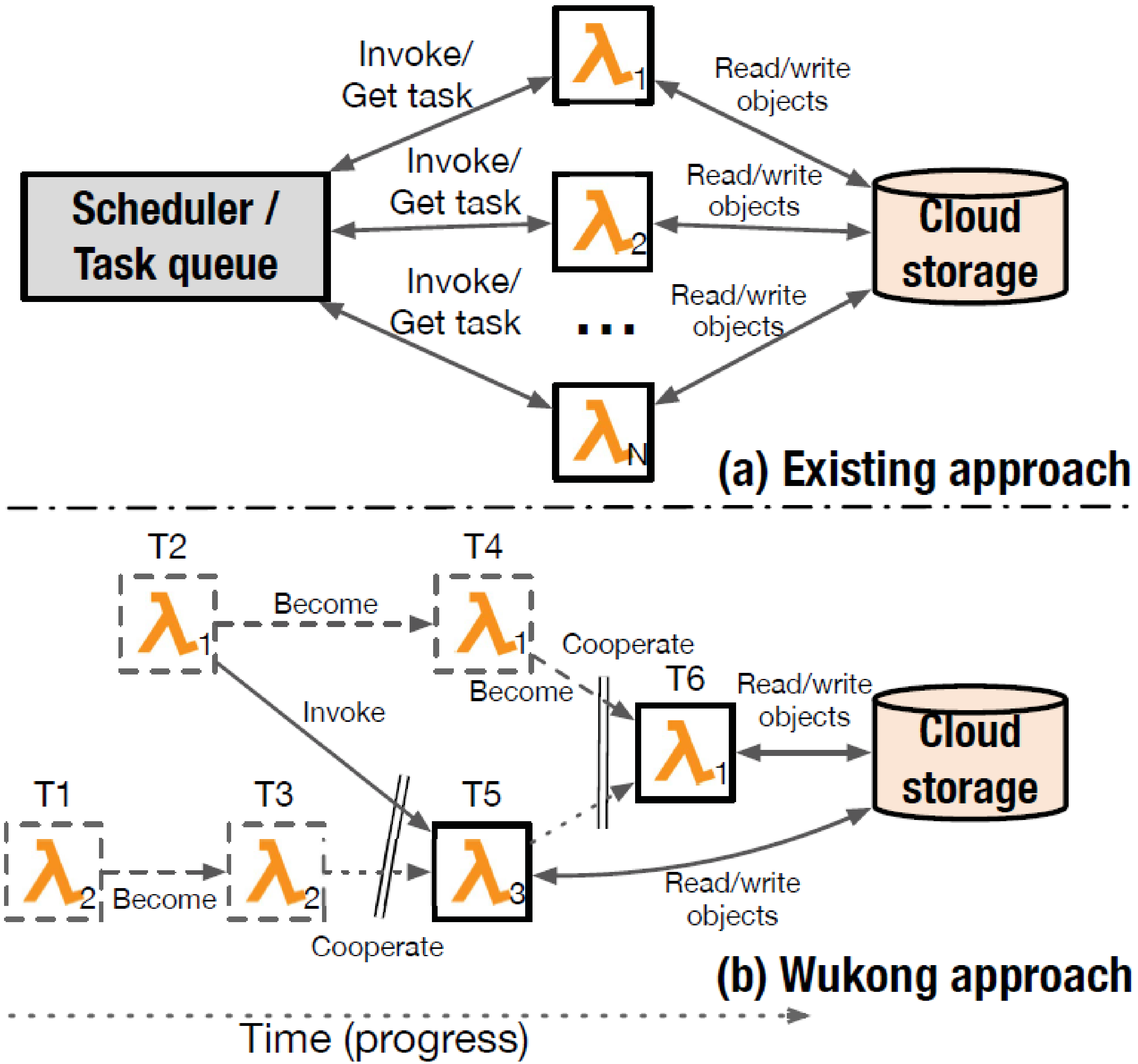
- Carver et al., introduced “Wukong”, a decentralized approach to schedule serverless functions.
- Distributed scheduling enables us to run scheduling on all runners in parallel.
- This has 4 benefits:
  - 1. Data Locality usage optimization
  - 2. Less network I/O
  - 3. Auto scaling
  - 4. Cost efficiency

B. Carver et al., "Wukong: a scalable and locality-enhanced framework for serverless parallel computing," in Proceedings of the 11th ACM Symposium on Cloud Computing, 2020, New York, USA, pp. pp. 1-15.

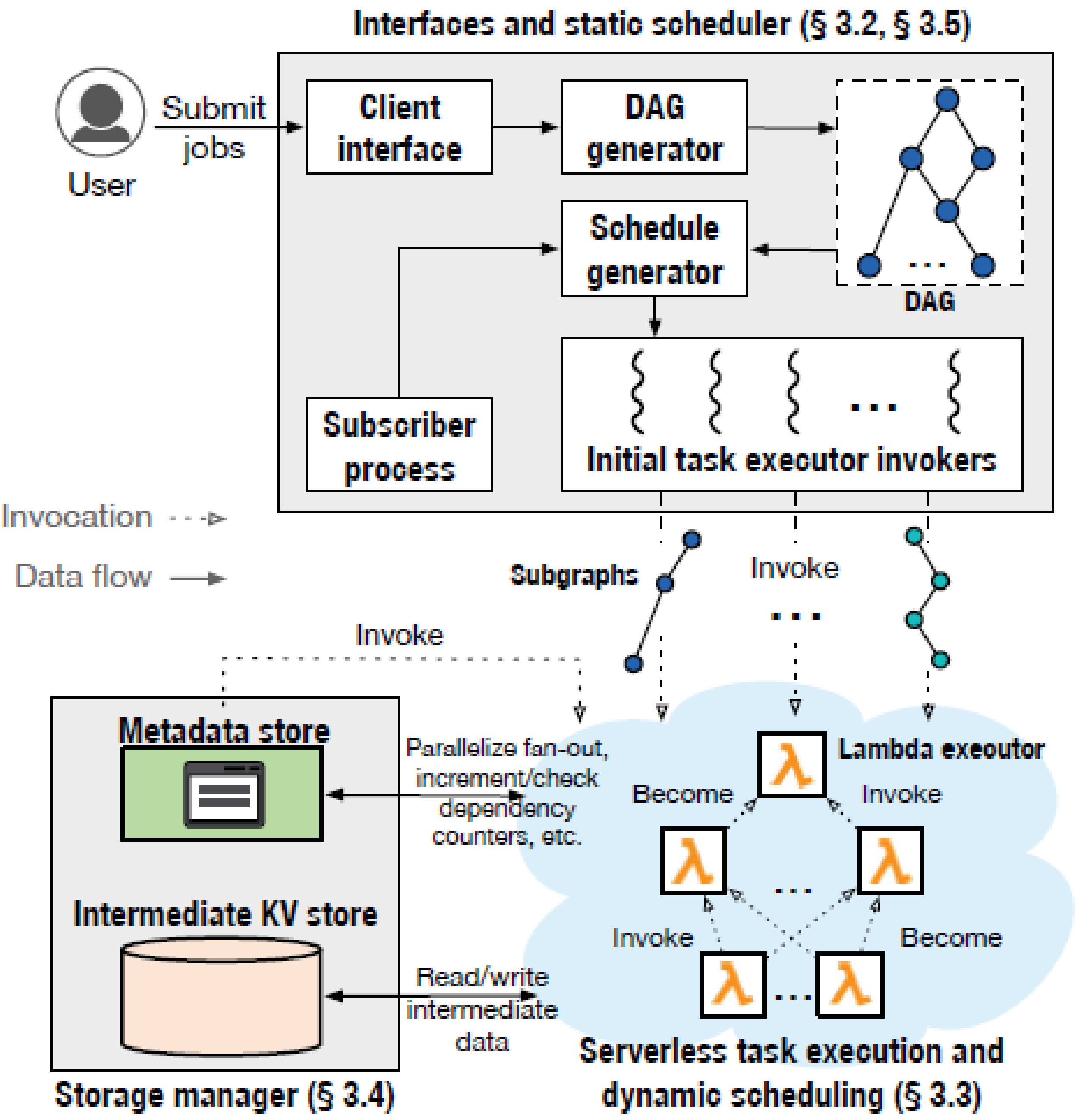
Approaches > Parallel

- They succeeded to run parallel tasks 68.17% faster
- They also reduced costs up to 92.96%

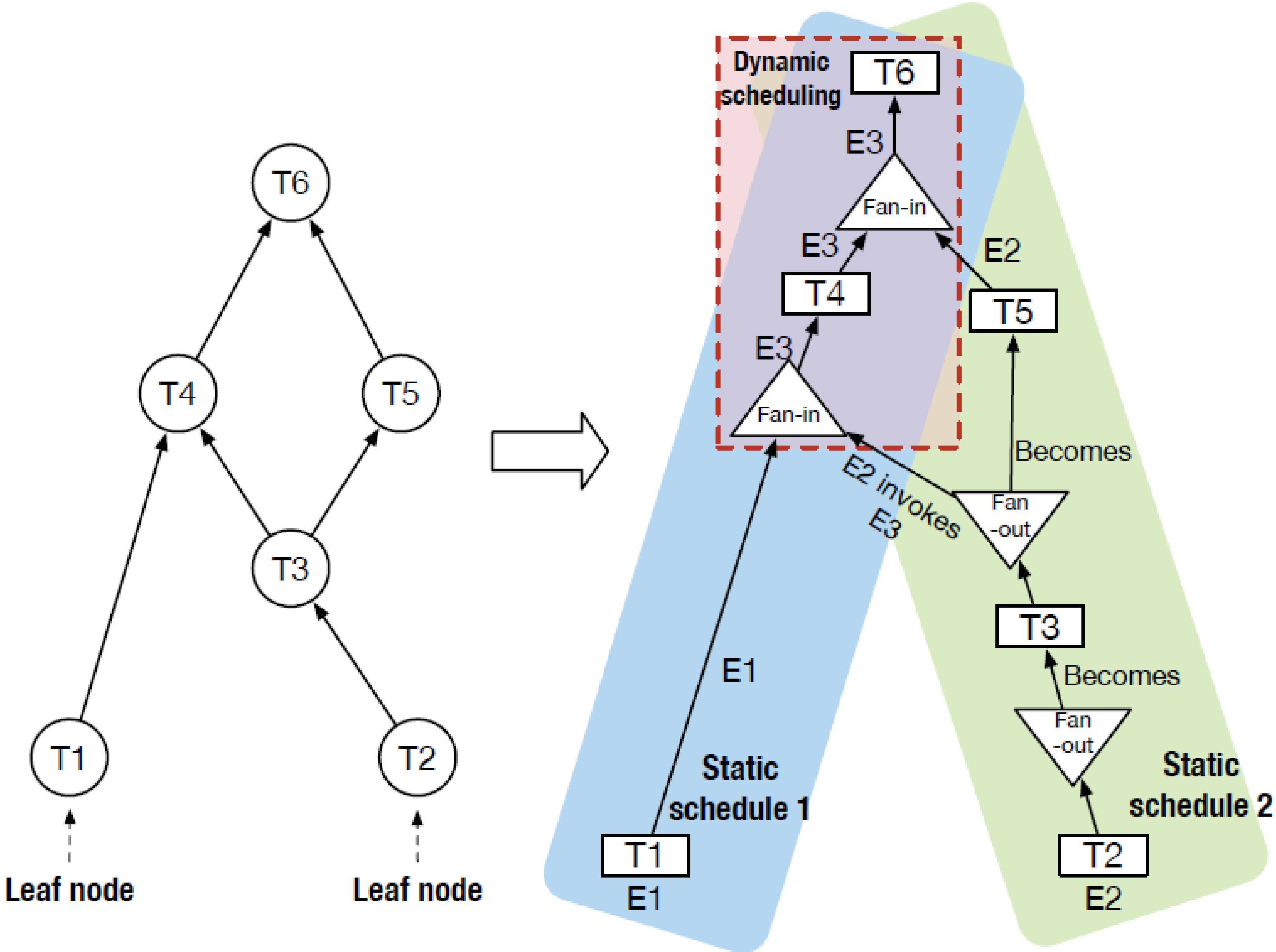
B. Carver et al., "Wukong: a scalable and locality-enhanced framework for serverless parallel computing," in Proceedings of the 11th ACM Symposium on Cloud Computing, 2020, New York, USA, pp. pp. 1-15.



Scheduler / Task queue
Invoke/ Get task
Invoke/ Get task
Invoke/ Get task
Read/write objects
Read/write objects
Read/write objects
Cloud storage
(a) Existing approach
T2
T4
Become
Cooperate
Become
T6
Invoke
Read/write objects
Cloud storage
T1
T3
T5
Become
Cooperate
Read/write objects
(b) Wukong approach
Time (progress)



Interfaces and static scheduler (§ 3.2, § 3.5)
User
Submit jobs
Client interface
DAG generator
Schedule generator
DAG
Subscriber process
Initial task executor invokers
Invocation
Data flow
Subgraphs
Invoke
Invoke
Metadata store
Parallelize fan-out, increment/check dependency counters, etc.
Intermediate KV store
Read/write intermediate data
Storage manager (§ 3.4)
Lambda executor
Become
Invoke
Invoke
Become
Serverless task execution and dynamic scheduling (§ 3.3)



(a) Static DAG **(b) Dynamic scheduling**

References

T. Harter et al., "Slacker: Fast distribution with lazy docker containers," in 14th USENIX Conference on File and Storage Technologies (FAST 16), 2016, Santa Clara, United States, pp. 181-195.

I.E. Akkus et al., "SAND: Towards High-Performance Serverless Computing," in Usenix Annual Technical Conference (18), July-2019, Boston, USA, pp. 923-935.

G. Aumala et al., "Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms," in 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID) , May-2019, Larnaca, Cyprus, pp. 282-291.

M. Malawski et al., "Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions," Future Generation Computer Systems, vol. 110, pp. 502-514, September 2020.

J. Kijak et al., "Challenges for scheduling scientific workflows on cloud functions," in 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), Jul-2018, San Francisco, USA, pp. 460-467.

B. Carver et al., "Wukong: a scalable and locality-enhanced framework for serverless parallel computing," in Proceedings of the 11th ACM Symposium on Cloud Computing, 2020, New York, USA, pp. pp. 1-15.

Thank you!

Any questions?

دانشگاه علم و صنعت ایران

☆ download here: https://avestura.dev/presentations