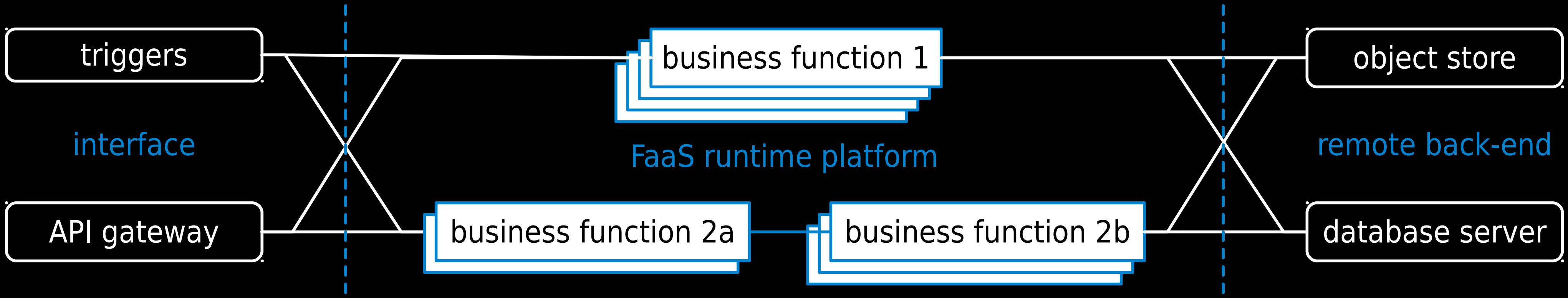
OFC: An Opportunistic Caching System for FaaS Platforms

Djob Mvondo	*Univ. Grenoble Alpes, ENS Lyon*
Mathieu Bacou	*Télécom SudParis, IP Paris*
Kevin Nguetchouang, Lucien Ngale	*ENSP Yaoundé*
Stéphane Pouget	*ENS Lyon*
Josiane Kouam	*Inria*
Renaud Lachaize	*Univ. Grenoble Alpes*
Jinho Hwang	*Facebook*
Tim Wood	*The George Washington University*
Daniel Hagimont	*University of Toulouse*
Noël De Palma	*Univ. Grenoble Alpes*
Bernabé Batchakui	*ENSP Yaoundé*
Alain Tchana	*ENS Lyon, Inria*

Context: Function-as-a-Service

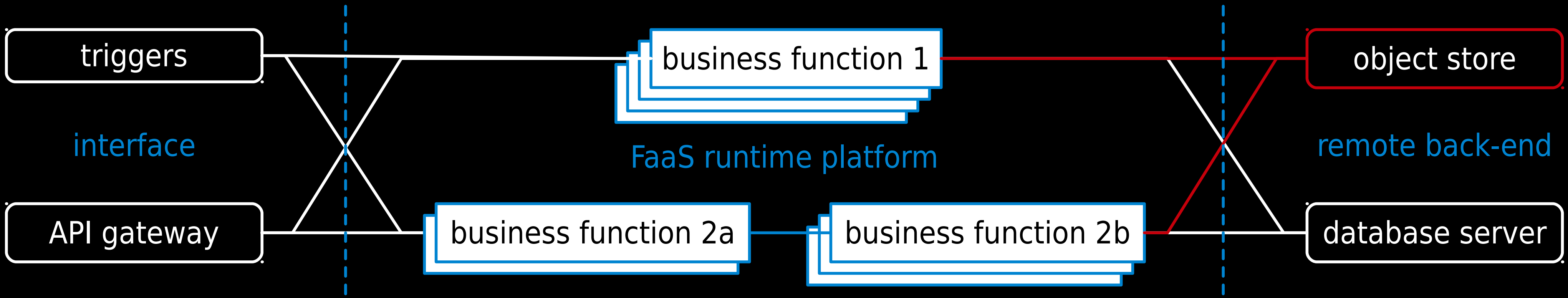
- Cloud-native applications
  - Built as collections of (chains of) functions
  - Rely on platform-provided back-end servers (serverless)
  - Mostly stateless by design



Function-as-a-Service architecture in a serverless cloud.

Extract-Transform-Load pattern[a]

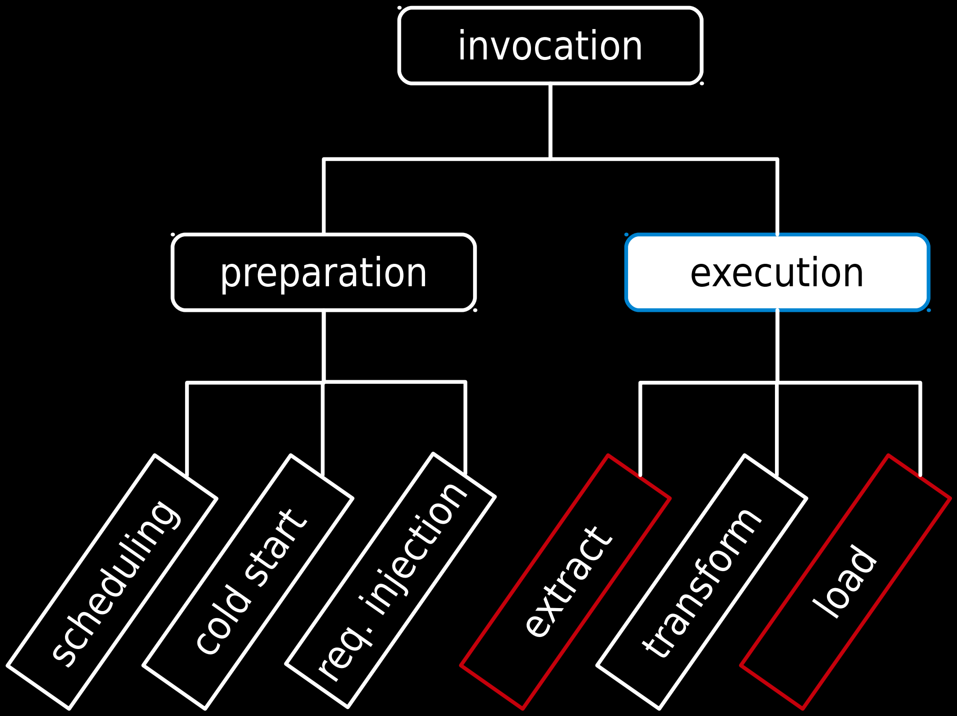
1. Extract (E) data from remote persistent storage (object store…)
2. Transform (T) by performing some computation (blur image…)
3. Load (L) result to remote persistent storage



Function-as-a-Service architecture in a serverless cloud.

a. H. Fingler et al. *USETL: Unikernels for Serverless Extract Transform and Load*. In APSys, 2019.

Performance issue: latency



FaaS performance issues in latency of function invocation, and concerns of our work.

- Storage access is a big issue with ETL
- Problem of data locality
  - Out-of-infrastructure remote storage
  - Even worse for pipelines

Related work

Caching, caching, and caching …

- Cloudburst[a]
- Infinicache[b]
- Pocket[c]
- ….

Existing works either require function modification or extra-resources (memory) to provision the cache layer

a. V. Sreekanti et al. *CloudBurst: stateful functions-as-a-service.* In VLDB Endowment, 2020.
b. A. Wang et al. *InfiniCache: Exploiting Ephemeral Serverless Functions to Build a Cost-Effective Memory Cache.* In FAST, 2020.
c. A. Klimovic et al. *Pocket: Elastic Ephemeral Storage for Serverless Analytics.* In OSDI, 2018.

Solution: caching in the FaaS age

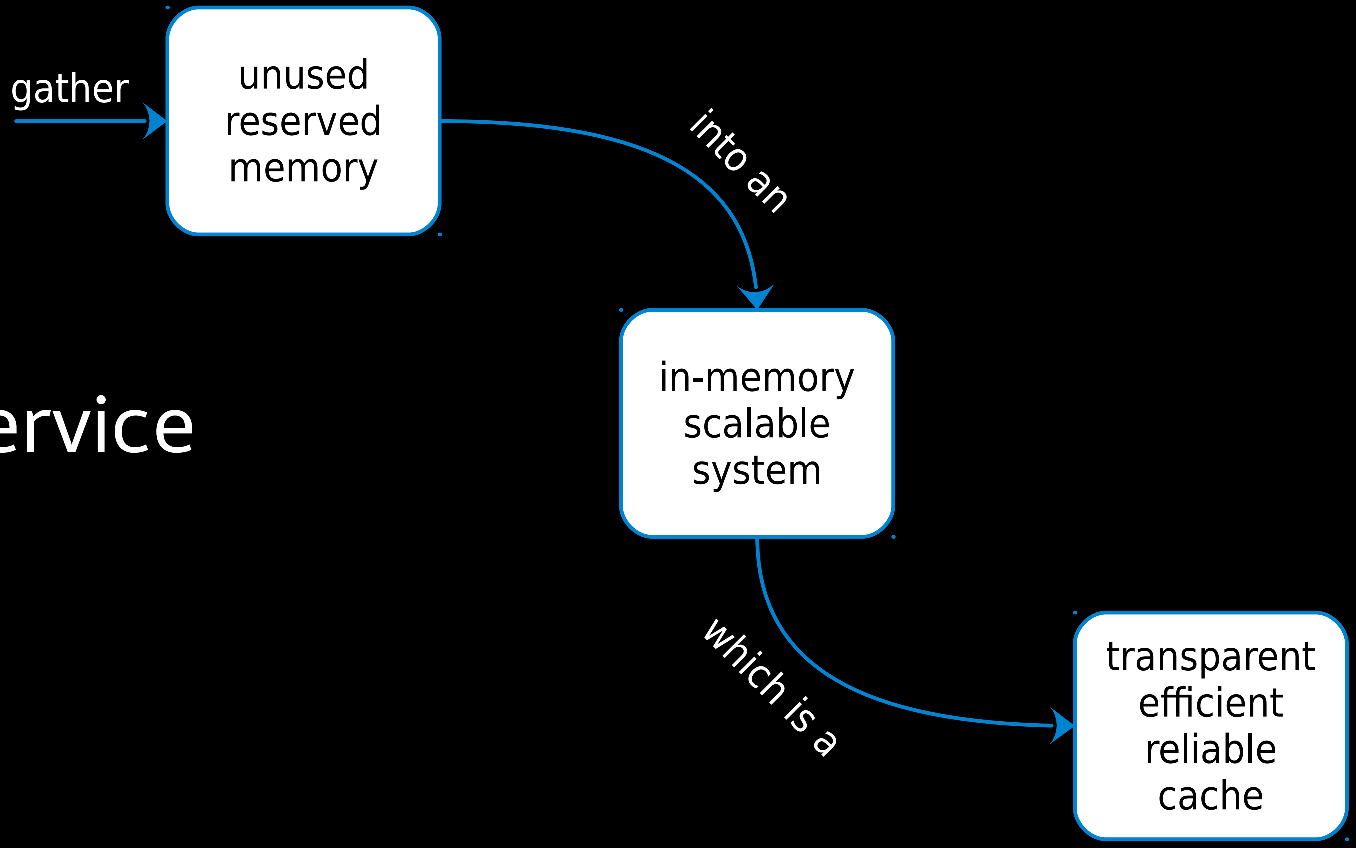
- Avoid remote storage with in-memory caching
- FaaS characteristics: very short latency, very elastic
- New challenges in the FaaS context:
  - How to provision memory for the cache?
  - How to make caching scale?
  - How to provide caching to functions?

OFC: Opportunistic FaaS Cache

Opportunistic

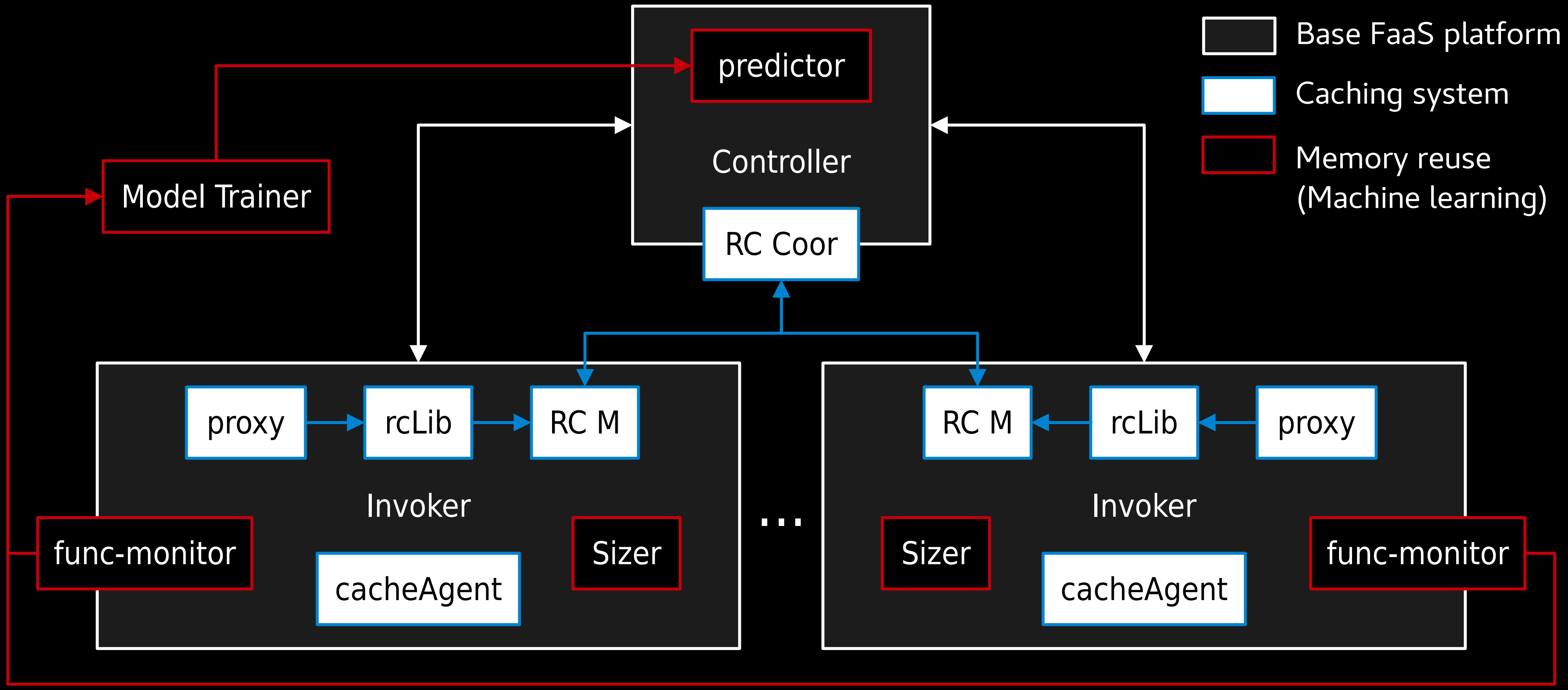
Function-as-a-Service

Cache



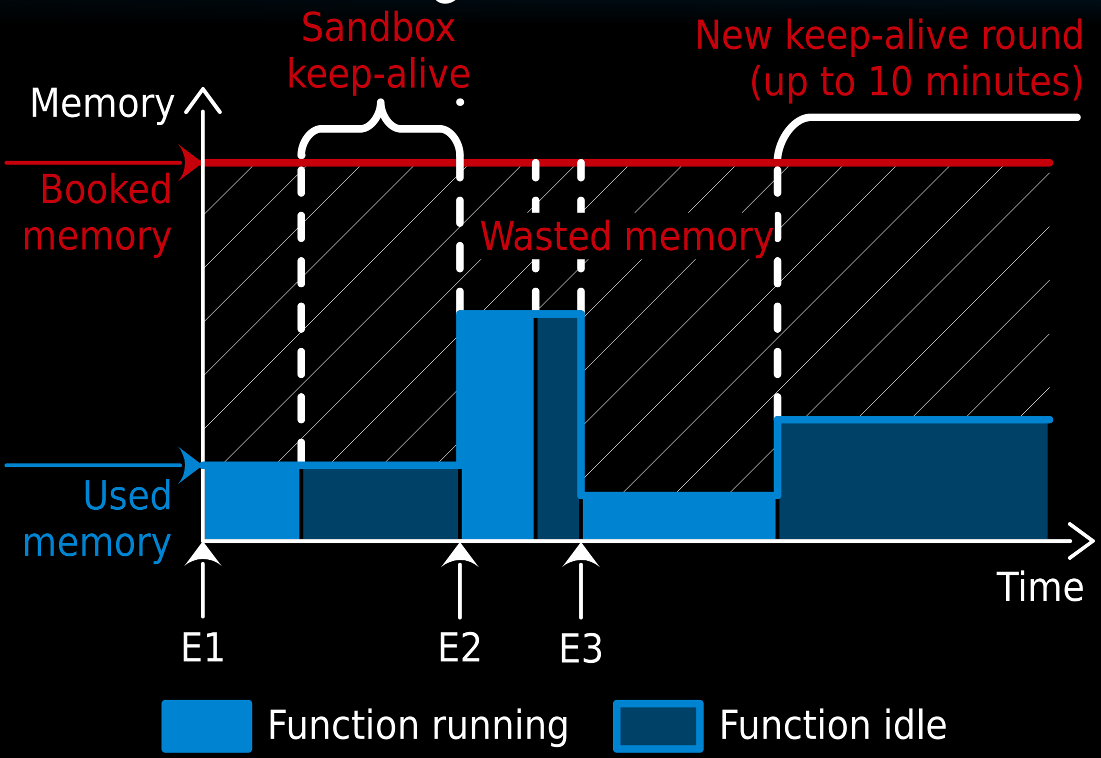
The three pillars of OFC.

OFC: Opportunistic FaaS Cache



Unused reserved memory

1. Over-provisioning by tenants to absorb workload variation[a]
   - 50% of functions reserve ≥512MB
   - 50% of functions use ≤29MB
2. Keep-alive policy: keep functions warm to reduce latency[b]
   - 81% invoked once per min. or less
   - Functions kept warm 10~20min (OpenWhisk, AWS Lambda)



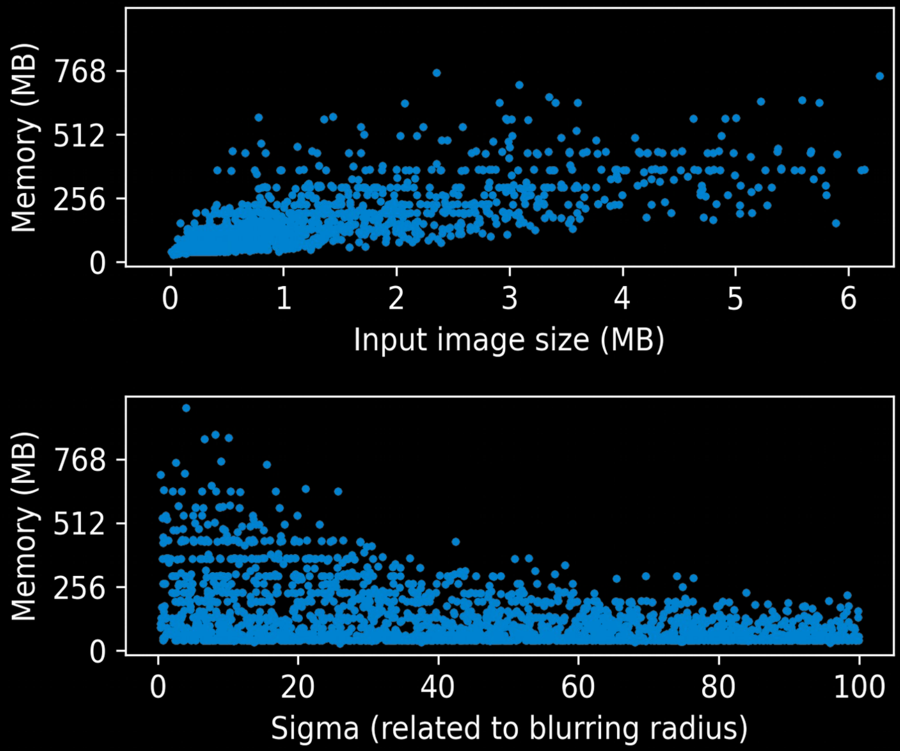
Timeline of a function sandbox illustrating wasted memory.

a. R. Ribensaft. *What AWS Lambda's Performance Stats Reveal* . Web source, 2020.
b. M. Shahrad et al. *Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider.* In USENIX ATC, 2020.

Predicting wasted memory

- How much memory is available to the cache?
  - Complex relation with data, parameters
- Use machine learning!
  - White-box functions
    - Parameters, inputs…
  - High invocation rate
    - Quick dataset gathering

Memory usage of an image blurring function



Relation between memory usage and function invocation parameters and input.

Learning memory usage, and more

- Constraints of the FaaS:
  - Learn and update models
    - Maintain training dataset
  - Learn from unknown features: bounds, sets of values?
    - Cannot compute from features
  - Prediction speed: on the critical path of the invocation
    - Predict in less than 1ms
- Classification instead of regression
  - Predict among 16MB intervals

- Decision trees: J48 (C4.5)
  - 92.7% accuracy for exact-or-over predictions
  - Model *accurate enough* for 95% of functions in less than 8h of lifetime
  - 13x faster at 99% than RandomForest
    - While being just as accurate

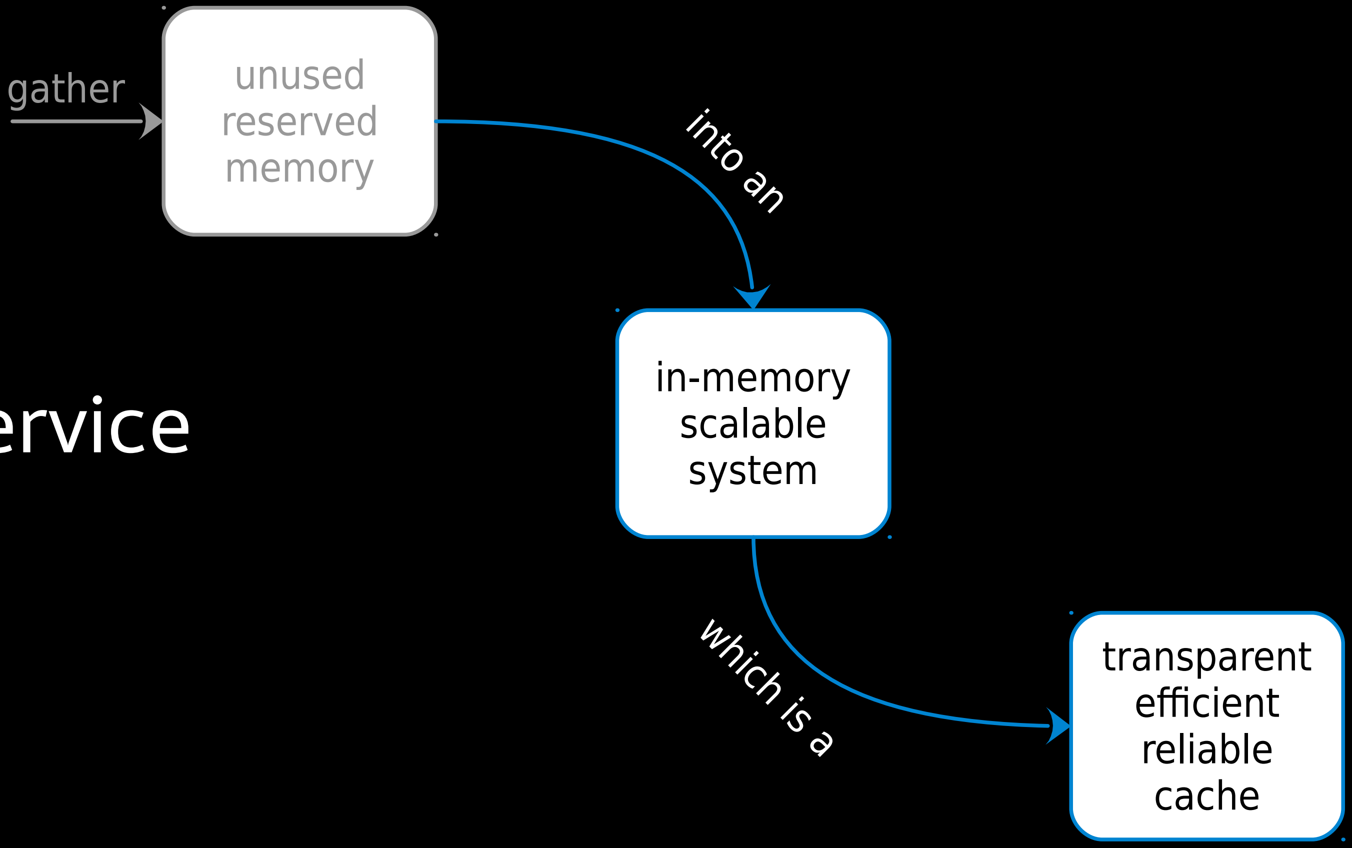
- ML also used to predict caching benefits
  - Keep only useful data in cache

OFC: Opportunistic FaaS Cache

Opportunistic

Function-as-a-Service

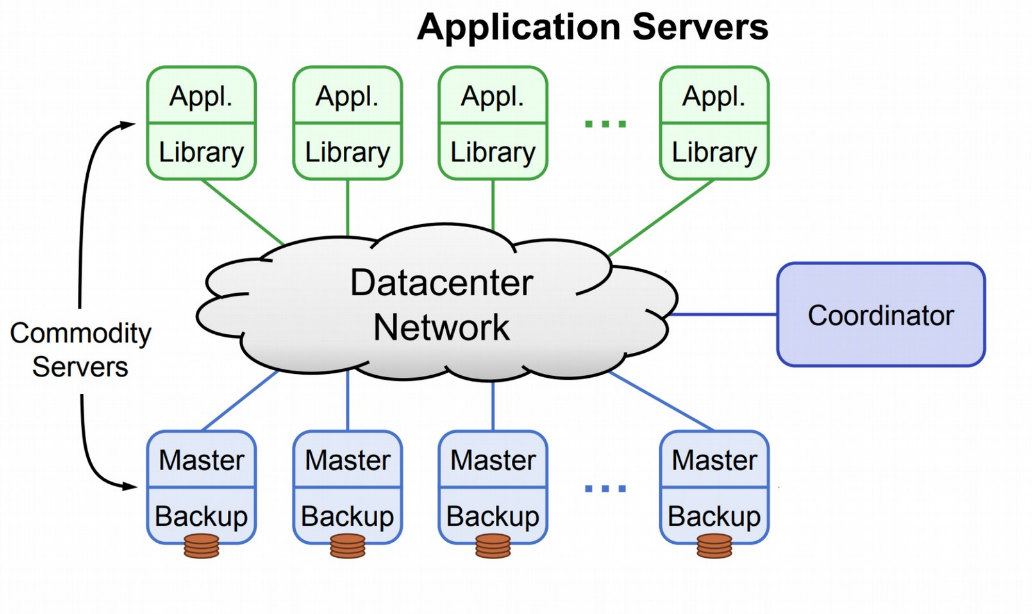
Cache



The three pillars of OFC.

OFC caching mecanisms overview

- OFC leverages RAMCloud[a]
  - Distributed
  - In-memory
  - Fault tolerant

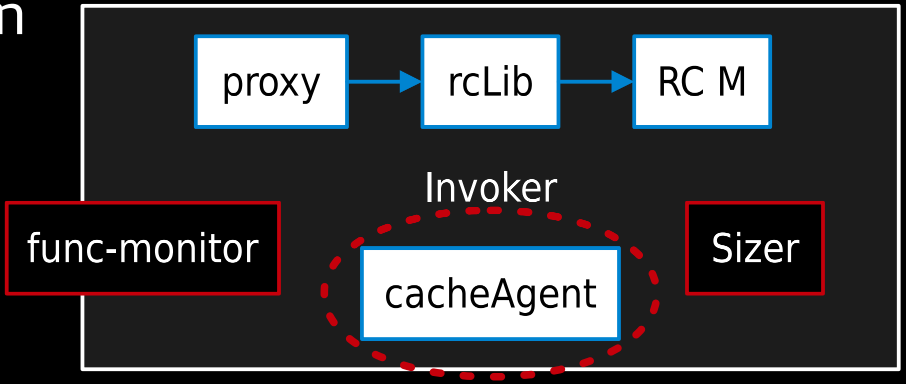


- RAMCloud can store objects up to 8MB. We updated this to 10MB.

a. J. Ousterhout at al. *The RAMCloud Storage System.* ACM Trans. On Comp. Sys, 2015.

OFC caching mecanisms overview

- On each invoker node:
  - RC M: RAMCloud cache master
  - CacheAgent: cache autoscaling
    - Scale the cache memory up/down
    - Monitor the cache pressure
    - Perform Garbage Collection



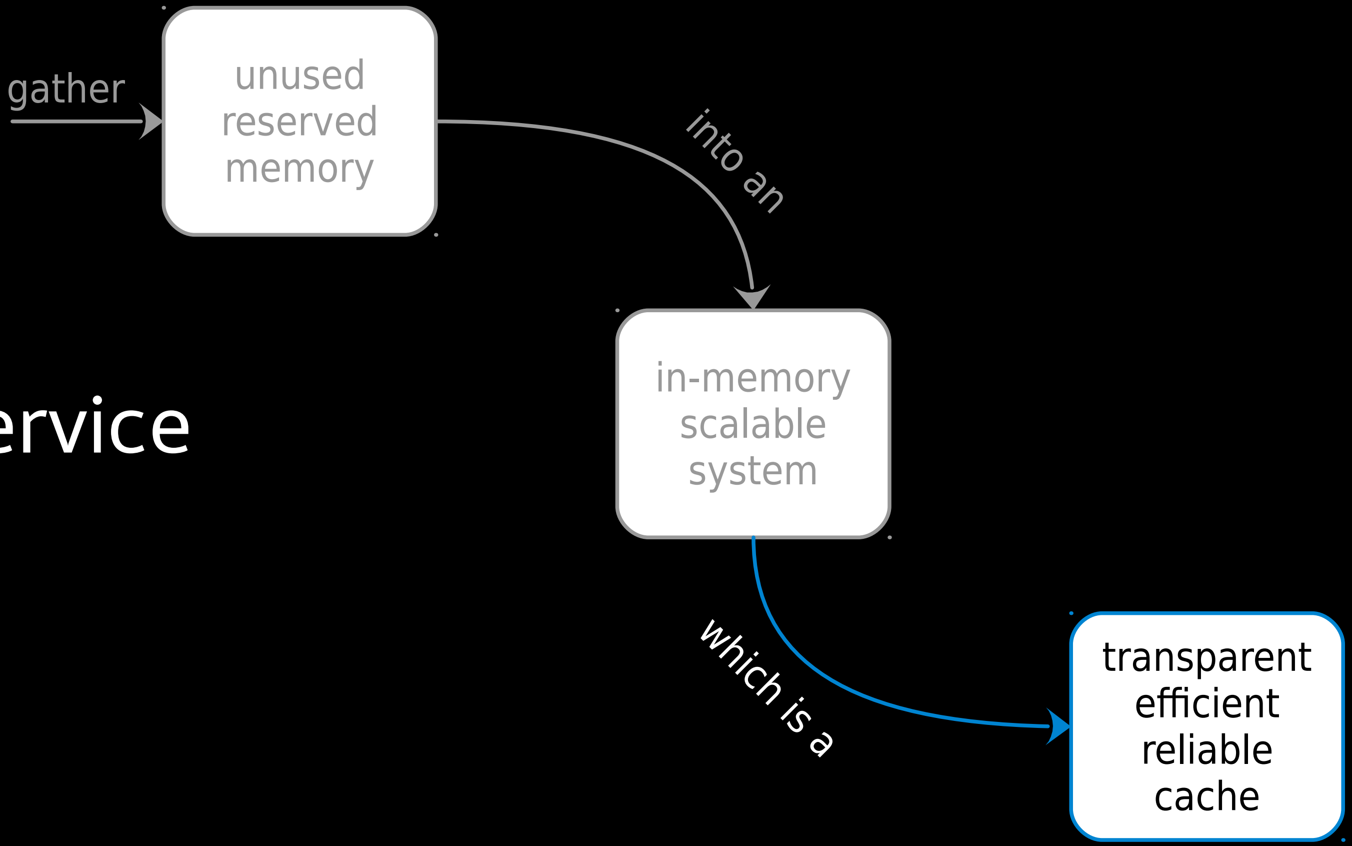
Cache autoscaling: the CacheAgent component.

OFC: Opportunistic FaaS Cache

Opportunistic

Function-as-a-Service

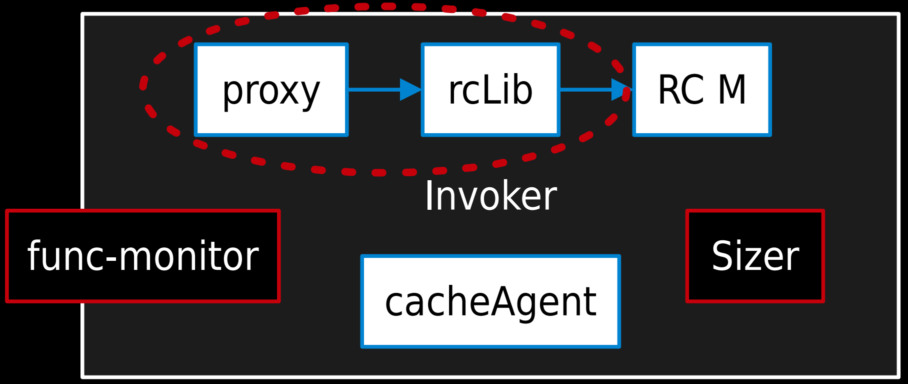
Cache



The three pillars of OFC.

OFC caching mecanisms overview

- A proxy transparently intercepts function calls to storage nodes.
  - Runtime interception
  - Routes request to cache API (rcLib)

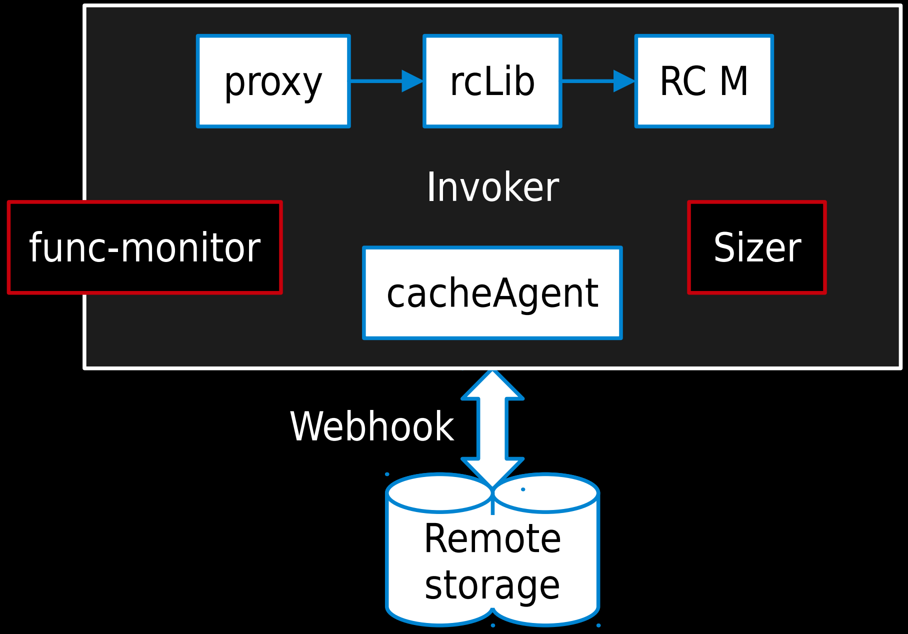


Transparent caching: proxy and rcLib.

OFC caching mecanisms overview

- RAMCloud library rcLib:
  - Persist data on the local cache
  - Ensure consistency with remote storage

- To ensure consistency with OFC, on storage node, a webhook checks for queries the cache for incoming read requests



Data persistence and consistency with remote storage.

OFC evaluation results

- Does OFC improve serverless functions latencies?
  - Single functions
  - Multi-stage functions

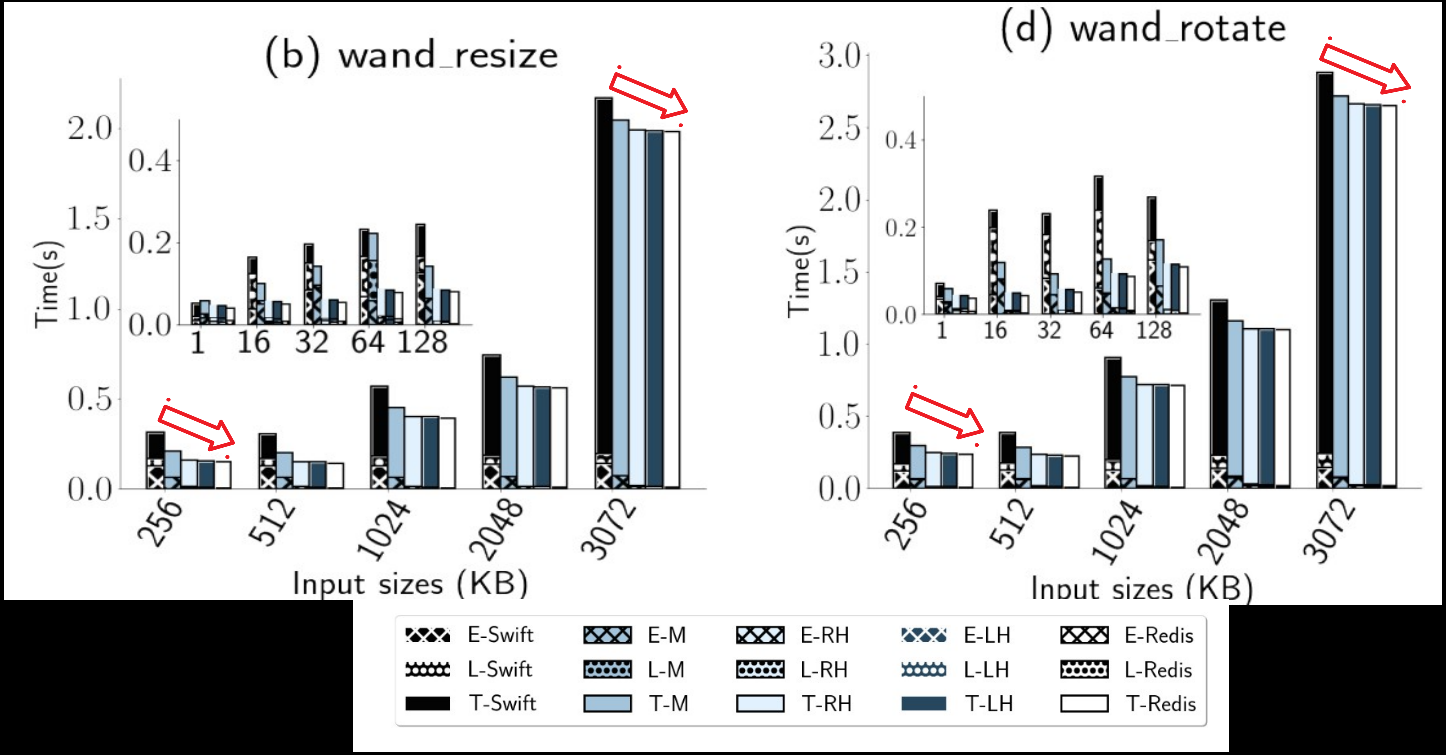
- Five scenarios
  1) Redis
  2) OFC Local Hit (LH)
  3) OFC Remote Hit
  4) Miss (M)
  5) Default (Swift)

Memory	512 GB
OS	Ubuntu 16.04.7 LTS
CPUs	2 Intel Xeon E5-2698v4 CPUs (20 cores/CPU)
Disk	480 GB SSD
Network	Intel Ethernet 10G 2P X520 Adapter

OFC evaluation results

- Single functions:

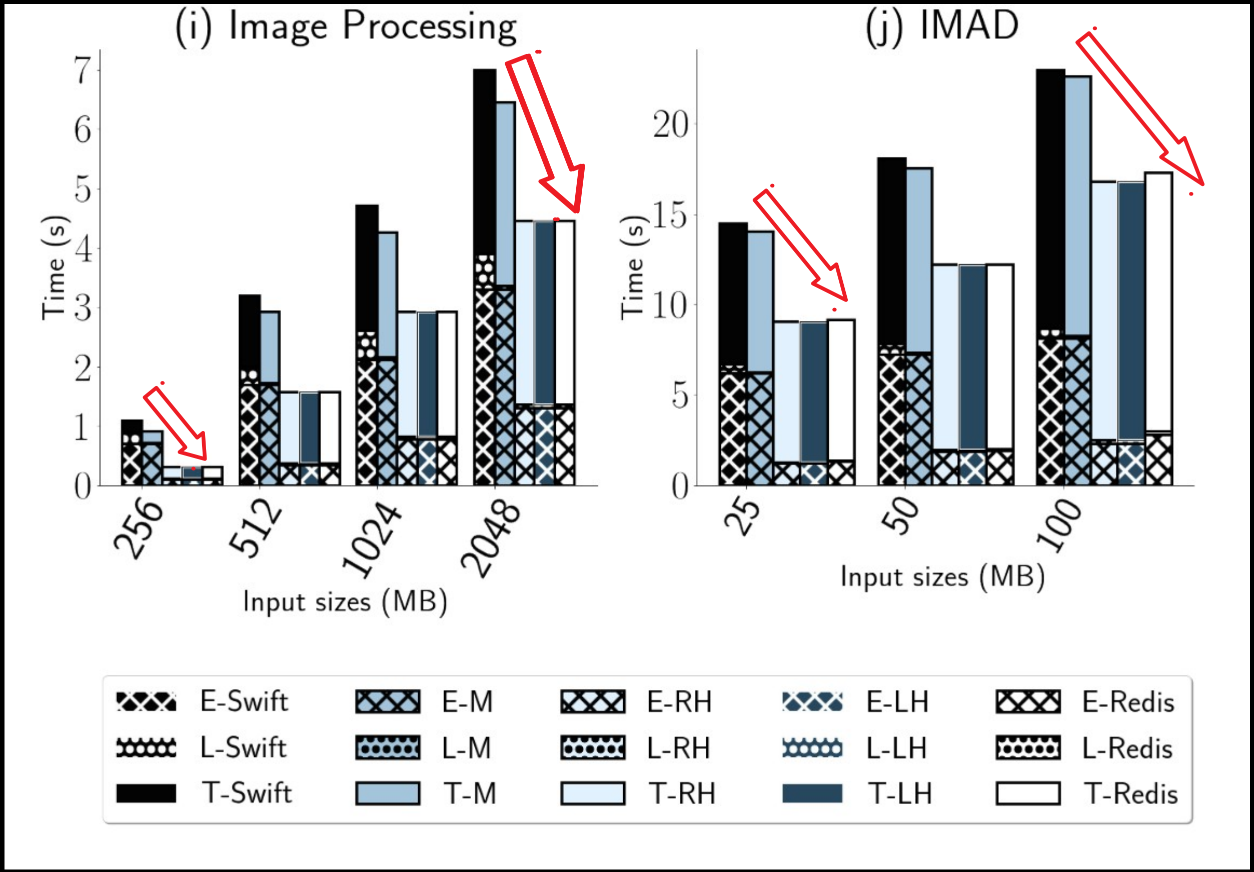
OFC overcomes Swift by up to **82%**



OFC evaluation results

- Multi-stage functions

OFC overcomes Swift by up to 60%



OFC: Conclusion

- OFC leverages ML and RAMCloud
  - Opportunistic caching layer for serverless functions
- OFC does not require function modification
  - Direct benefit for existing functions
- OFC ensures consistency between the platform's cache and the remote storage
- OFC achieves major latency improvements
  - Up to 82% for single functions
  - Up to 60% for multi-stage functions

Checkout OFC source code at
https://gitlab.com/lenapster/faascache/