Seamless Offloading of Web App Computations From Mobile Device to Edge Clouds via HTML5 Web Worker Migration

Hyuk Jin Jeong

Seoul National University

SoCC 2019

Virtual Machine & Optimization Laboratory

Department of Electrical and Computer Engineering

Seoul National University

Computation Offloading

Mobile clients have limited hardware resources

- Require computation offloading to servers
- E.g., cloud gaming or cloud ML services for mobile

Traditional cloud servers are located far from clients

- Suffer from high latency

End device

60~70 ms
(RTT from our lab to the closest Google Cloud DC)

↔

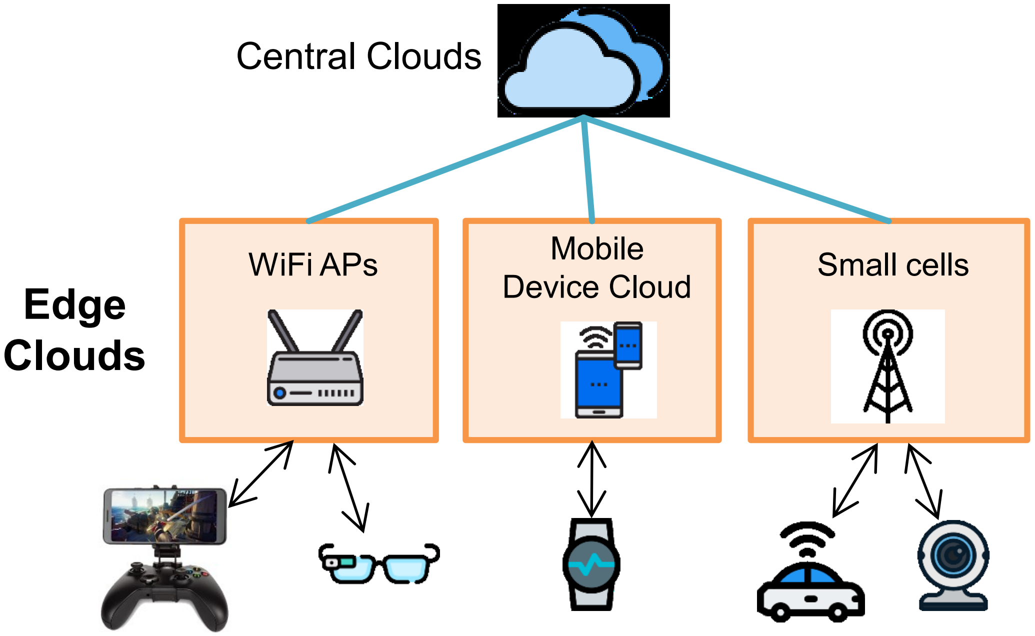
Latency<50 ms is preferred for time-critical games
[Kjetil Raaen, NIK 2014]

Cloud data center

Edge Cloud

Edge servers are located at the edge of the network

- Provide ultra low (~a few ms) latency



What if a user moves?

A Major Issue: User Mobility

How to seamlessly provide a service when a user moves to a different server?

- Resume the service at the new server
- What if *execution state* (e.g., game data) remains on the previous server?

This is a challenging problem

- Edge computing community has struggled to solve it
  - VM Handoff [Ha et al. SEC' 17], Container Migration [Lele Ma et al. SEC' 17], Serverless Edge Computing [Claudio Cicconetti et al. PerCom' 19]

We propose a new approach for web apps based on app migration techniques

Outline

Motivation

Proposed system

WebAssembly migration

Evaluation

MOTIVATION

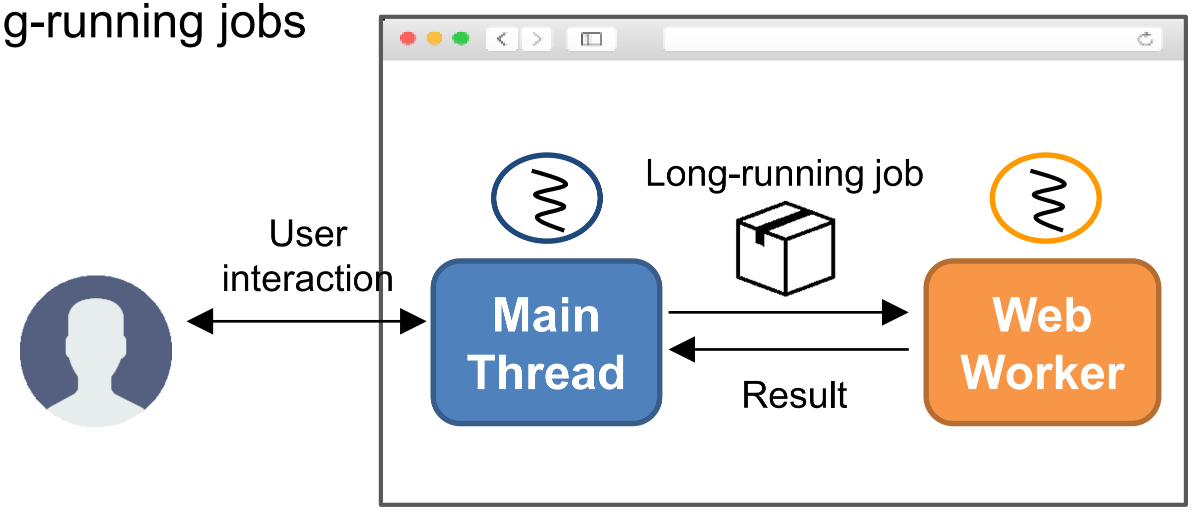
Background: Web Apps

Apps running on a web browser

- Widely used in mobile devices due to portability
  - E.g., WebView in Android and iOS, Tizen, LG WebOS
- Program logics are written in JavaScript or WebAssembly (wasm)
  - wasm: low-level instructions for web

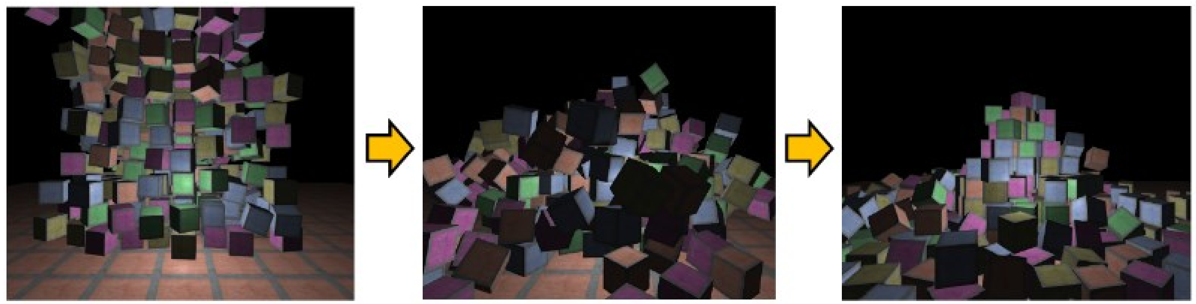
Web app threads

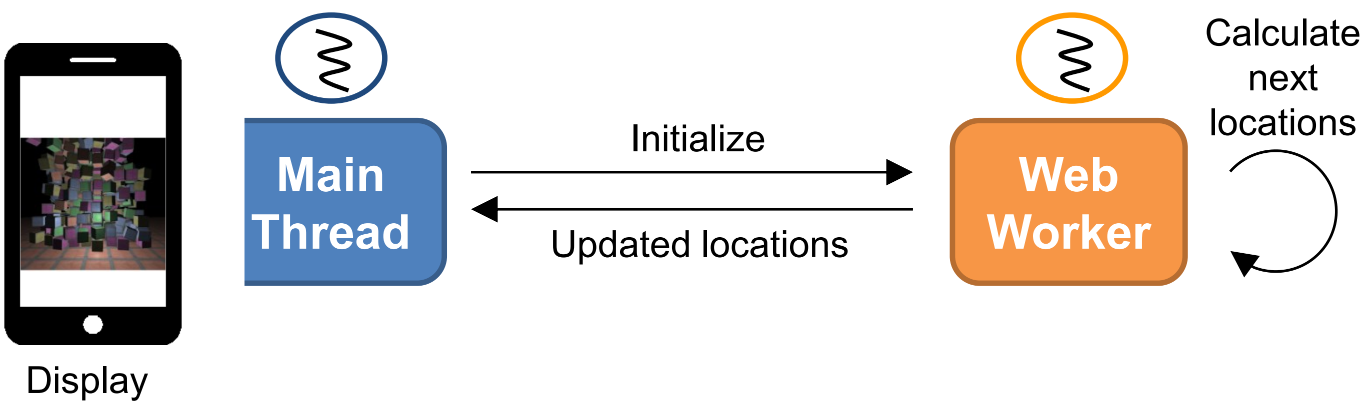
- Main thread: User interaction
- Web worker: Long-running jobs



Example: Physics Engine App

Web app simulating 3D cubes falling from the air





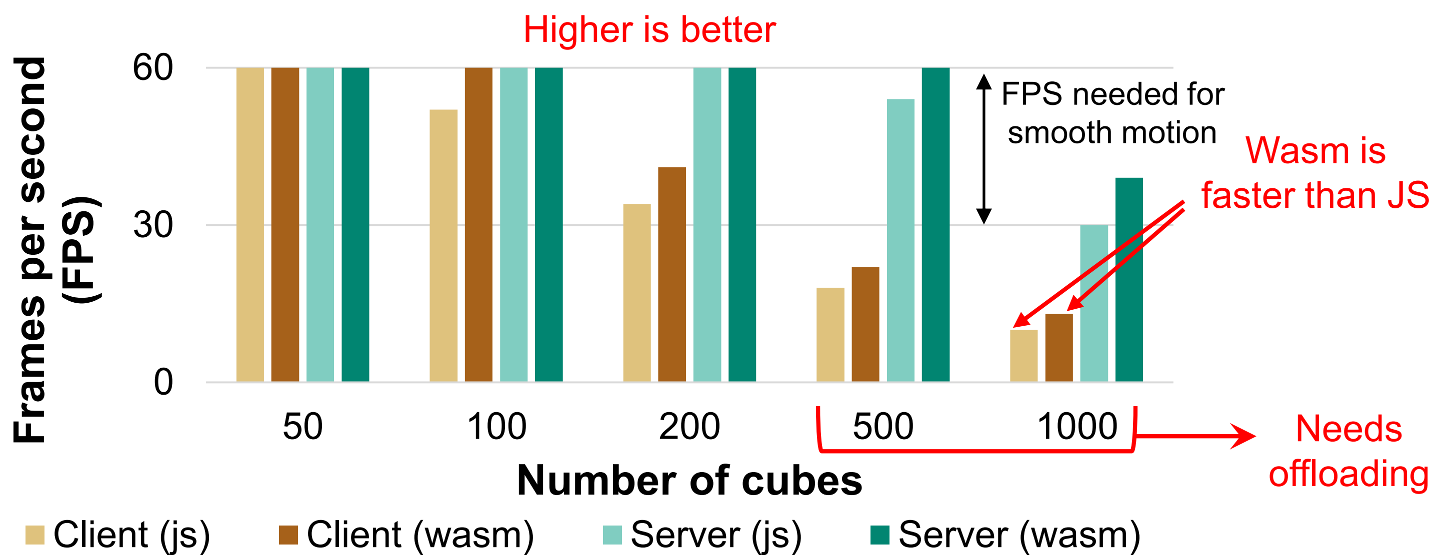
Cube locations

ID	x	y	z
1	13.4	44.1	99.1
2	52.6	79.5	10.5
…	…	…	…

Example: Physics Engine App

We ran the app on the server and the client and measured FPS

- Client: Odroid XU4 (ARM CPU 2.0 GHz, 2GB memory)
- Server: Desktop PC (x86 CPU 3.6 GHz, 16 GB memory)

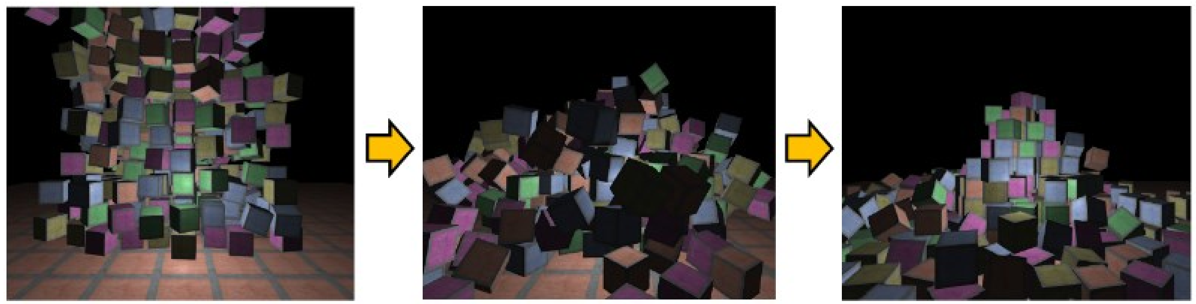


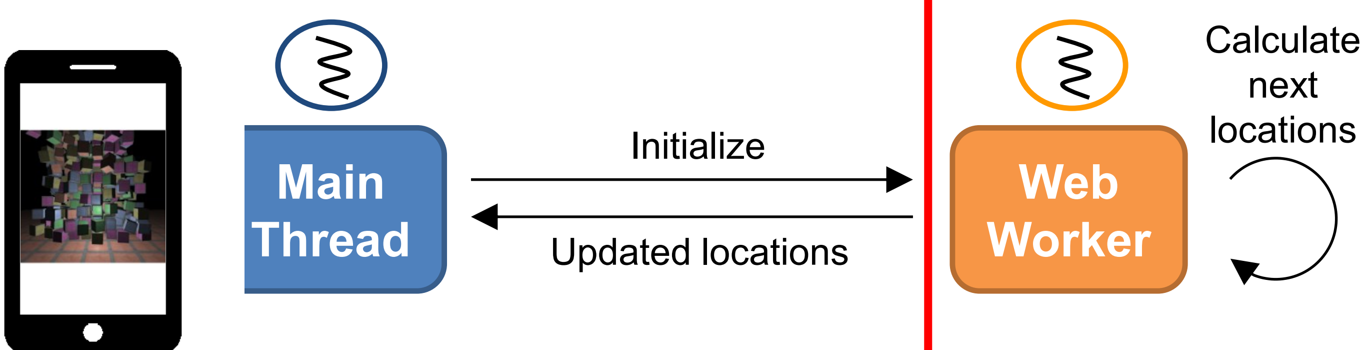
Observations

1. Wasm is faster than JS (20~30%)
2. Even with wasm, client-only is not enough when # of cubes ≥ 500

Example: Physics Engine App

Web app simulating 3D cubes falling from the air





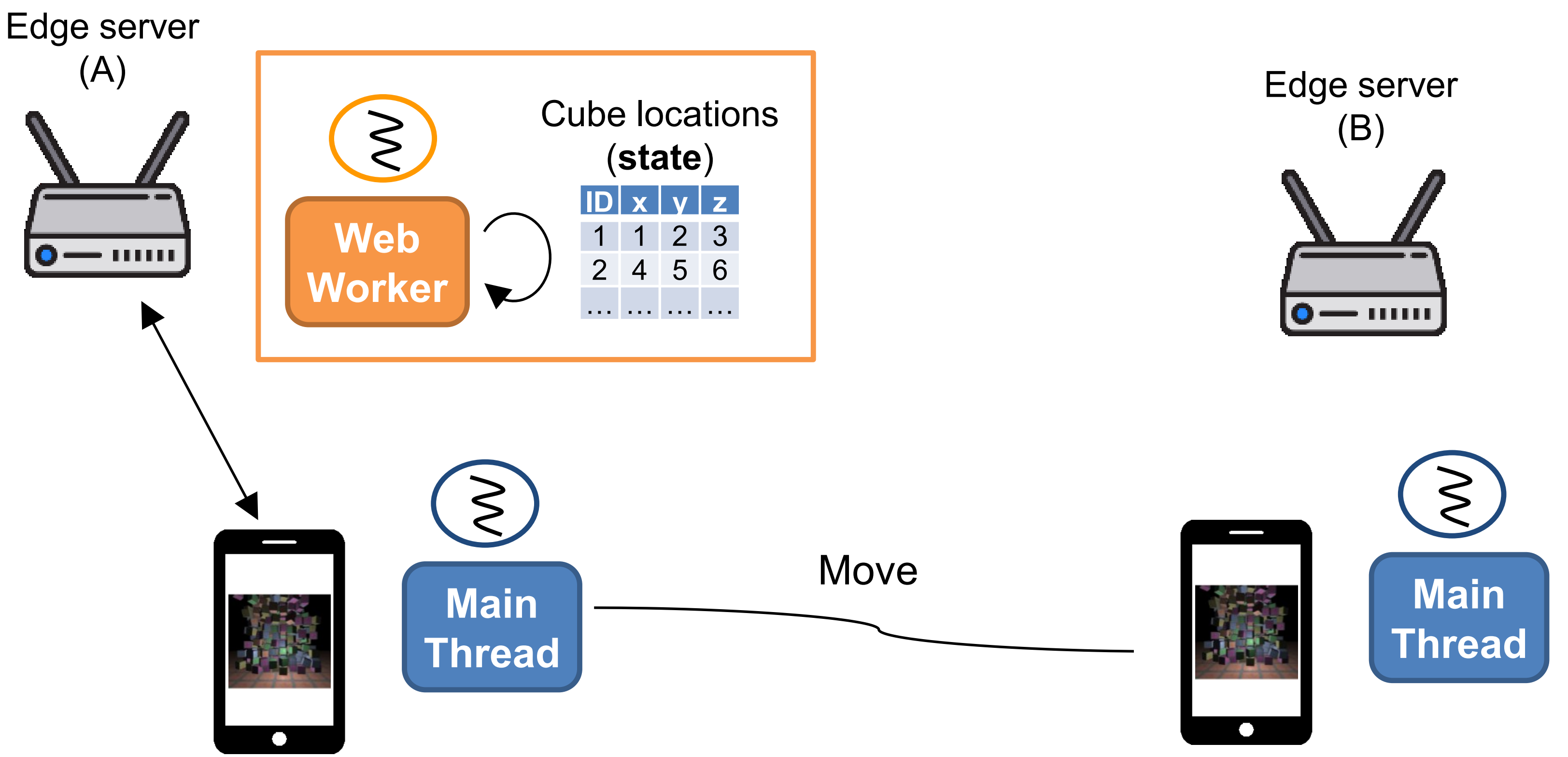
Cube locations **(state)**

ID	x	y	z
1	13.4	44.1	99.1
2	52.6	79.5	10.5
…	…	…	…

Computation-intensive

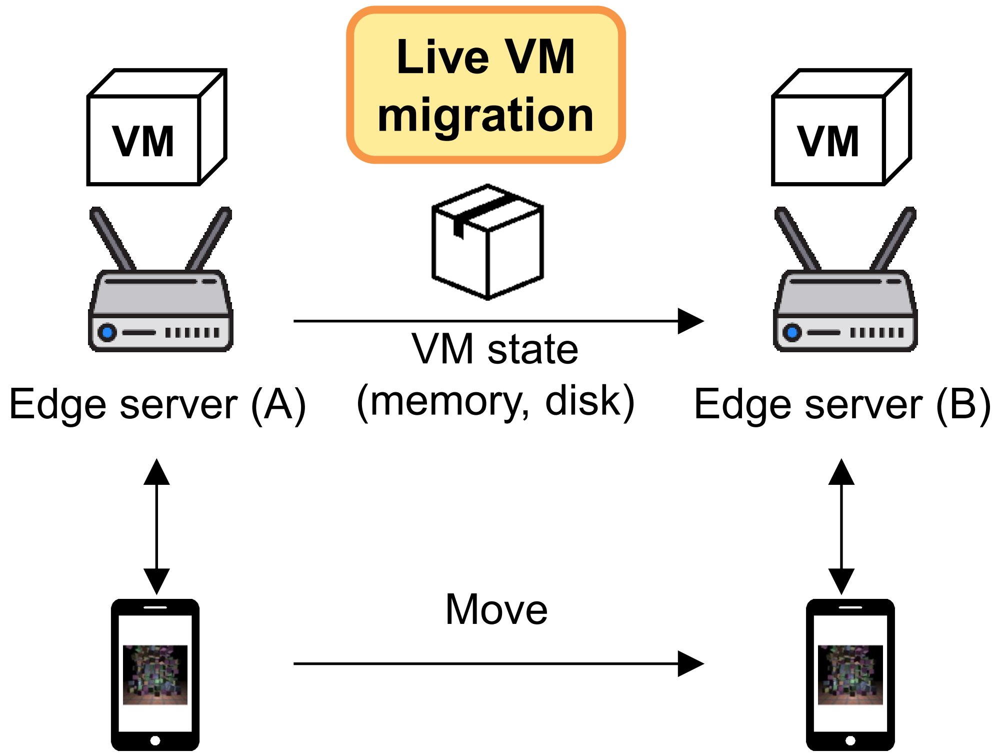
→ Do this on the server

Motivation: Mobile Scenario



How to continue service at the new edge server by seamlessly migrating previous edge's state?

Previous Approach (1): VM Handoff [Ha et al. SEC 2017]



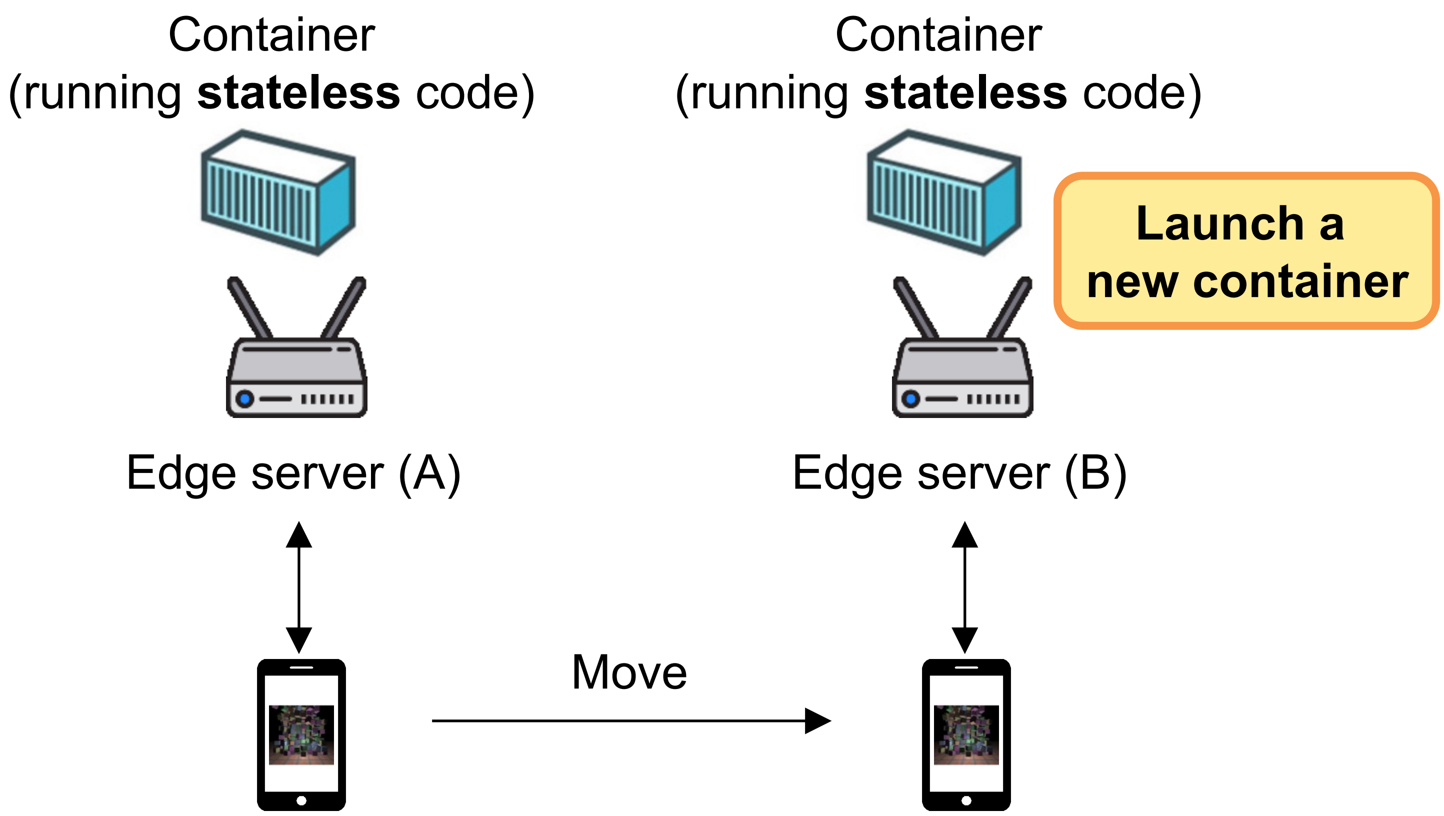
Issue

Live VM migration is heavy (due to a large base system)

- ~8 sec to migrate a Node.js instance

Previous Approach (2): Serverless computing

[Cicconetti et al. PerCom 2019]



Issue

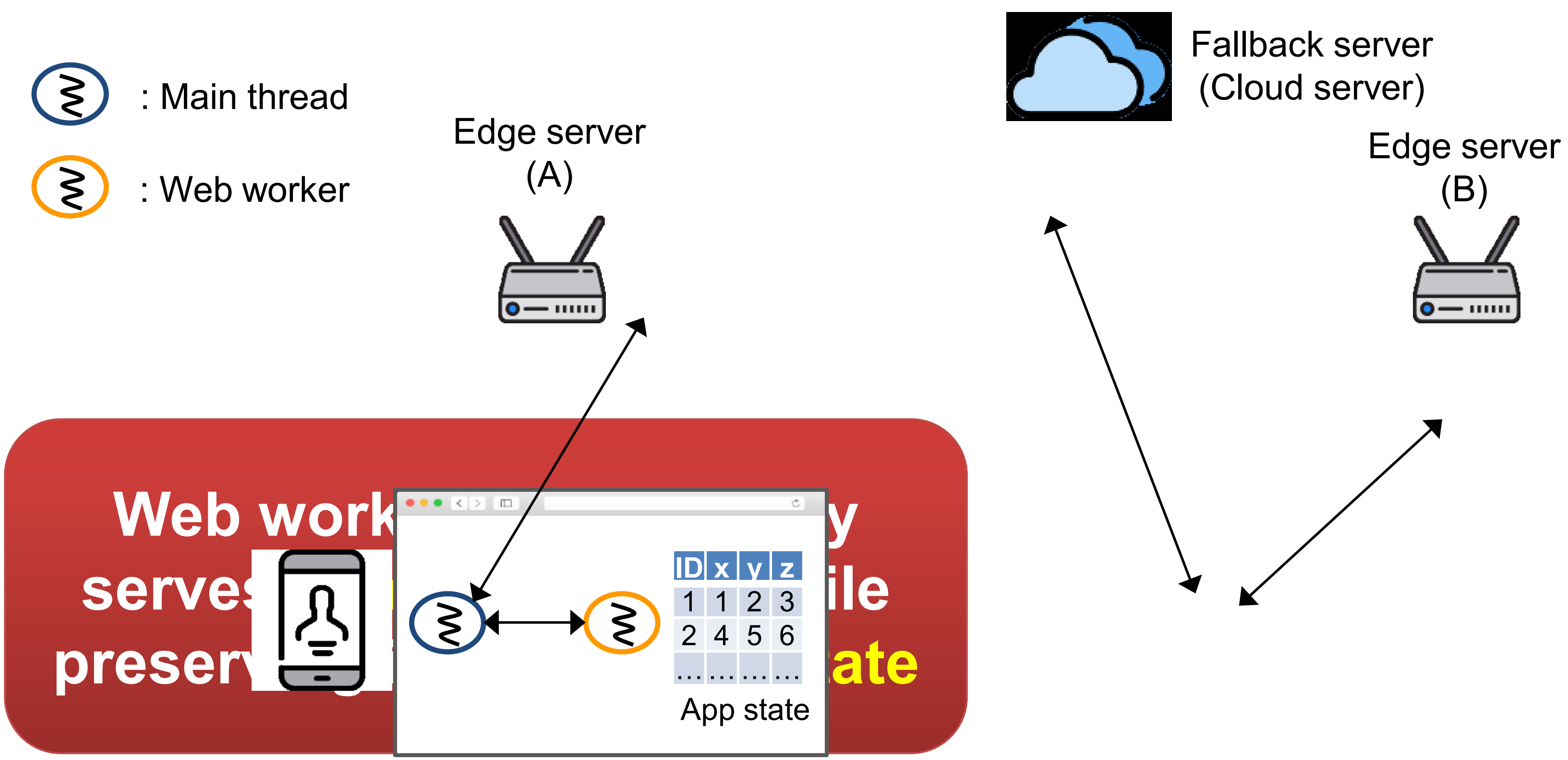
Effective only for short-lived, *stateless* jobs

- The worker in our physics app has *state* (cube locations)

Proposed Framework: Mobile Web Worker

We migrate a web worker across client, edge, and cloud

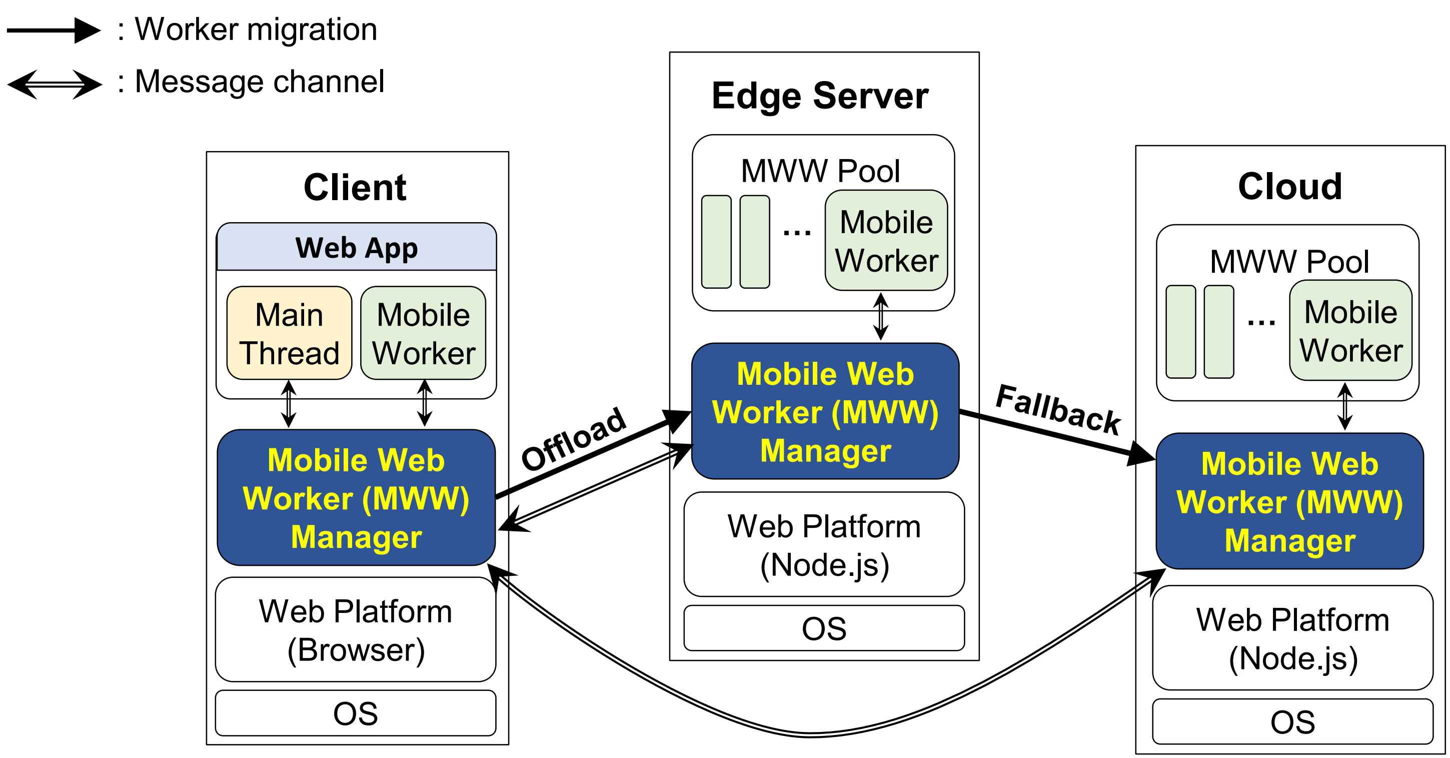
- Execution state is automatically migrated in an application level
  - No need to migrate base systems (OS or runtime) → Lightweight



Mobile Web Worker System

Mobile Web Worker (MWW) manager **controls migration of web workers and message passing with main thread**

- Directly captures and restores the web worker state
  - No VM-encapsulated black box



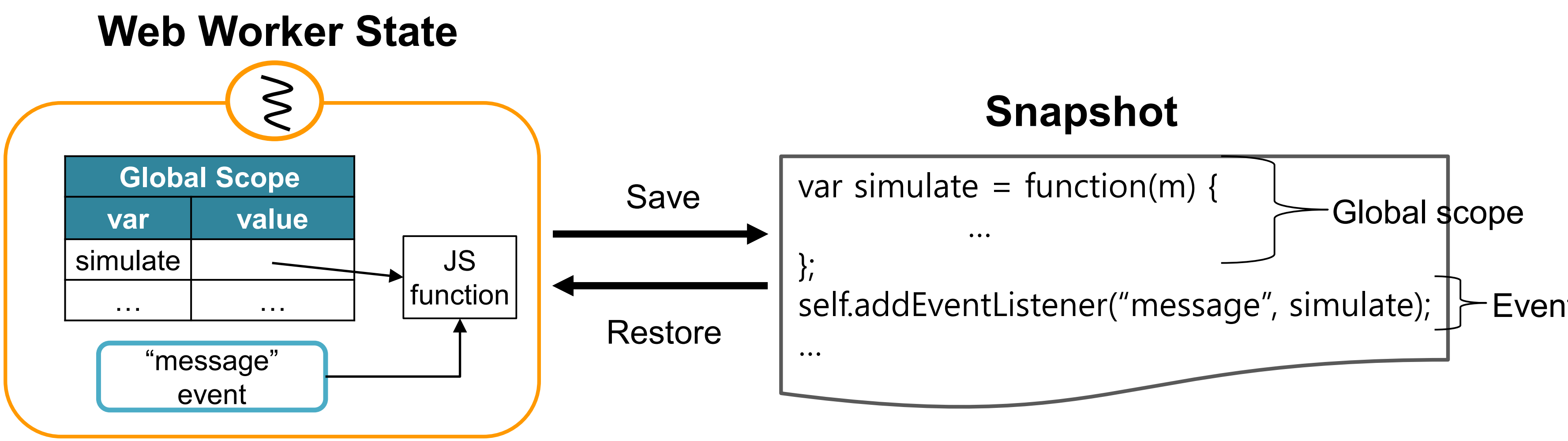
How to Migrate Web Worker State?

Web worker is a JS program, whose runtime state consists of

- JS scopes (variables, JS objects, functions) + events

These can be serialized into another JS code (*snapshot*) whose execution restores app state automatically [Oh et al. VEE '15] [Kwon et al. WWW '17]

- On any device equipped with a web platform



Issues on Web Worker Snapshot

Previous snapshot implementation does not properly migrate

1. Webassembly functions
2. Built-in objects

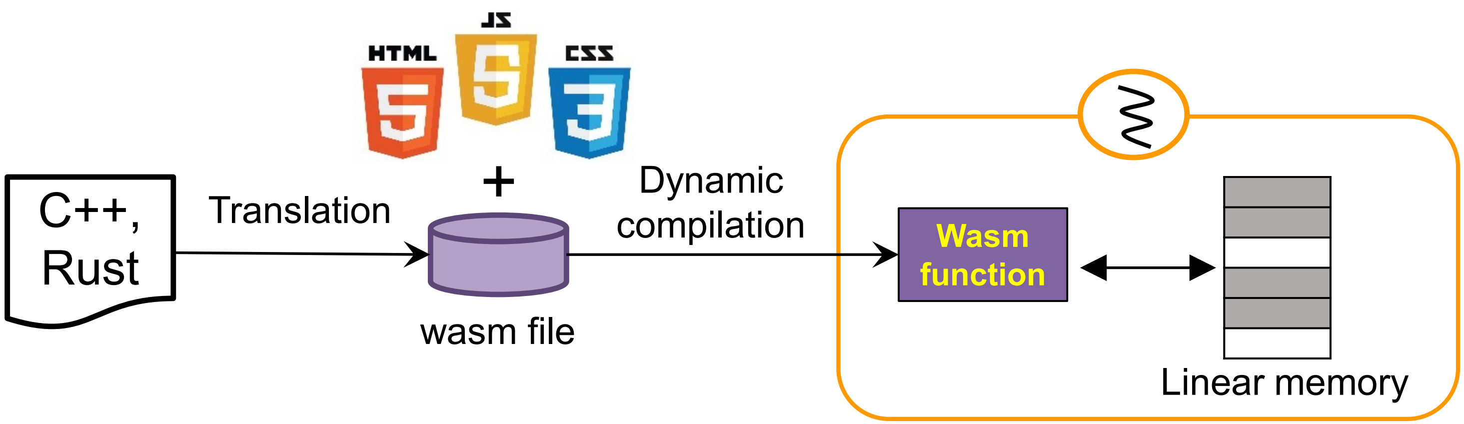
WEBASSEMBLY MIGRATION

Background: WebAssembly (Wasm)

Low-level instruction format for web for high performance

Wasm file is translated from high-level languages (ex: C++, Rust)

- Deployed with a web app source code
- Dynamically compiled when loaded onto the browser (or JS engine)
  - After compilation, wasm function and linear memory are created



Challenges on Wasm Migration

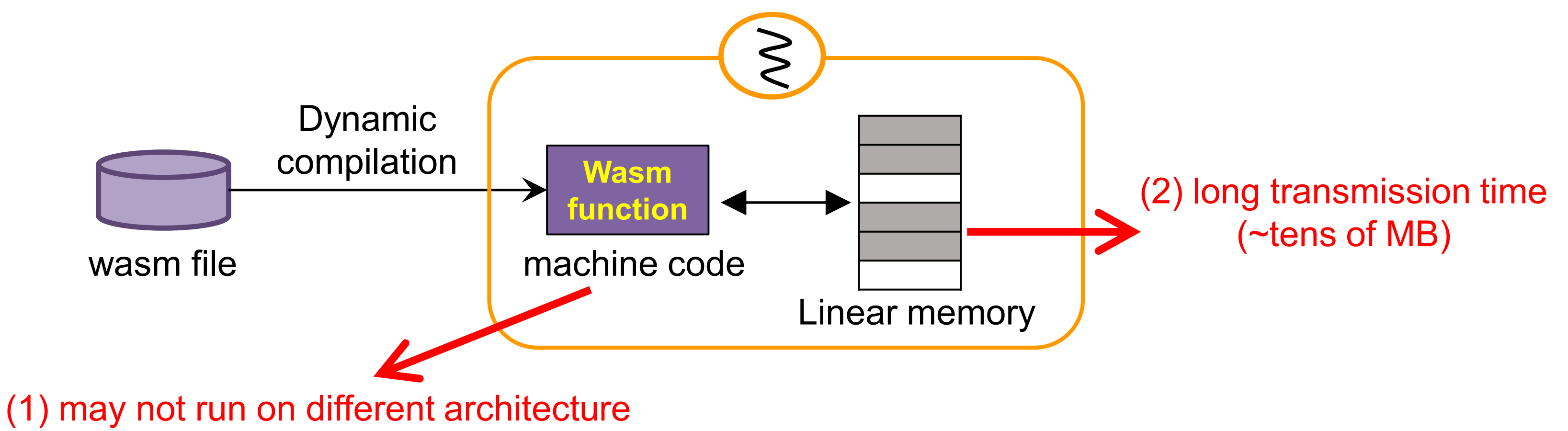
Wasm is difficult to serialize, because

(1) Wasm file is compiled into machine code when loaded

- Compiled machine code may not run on different architecture

(2) Wasm maintains a large memory (linear memory)

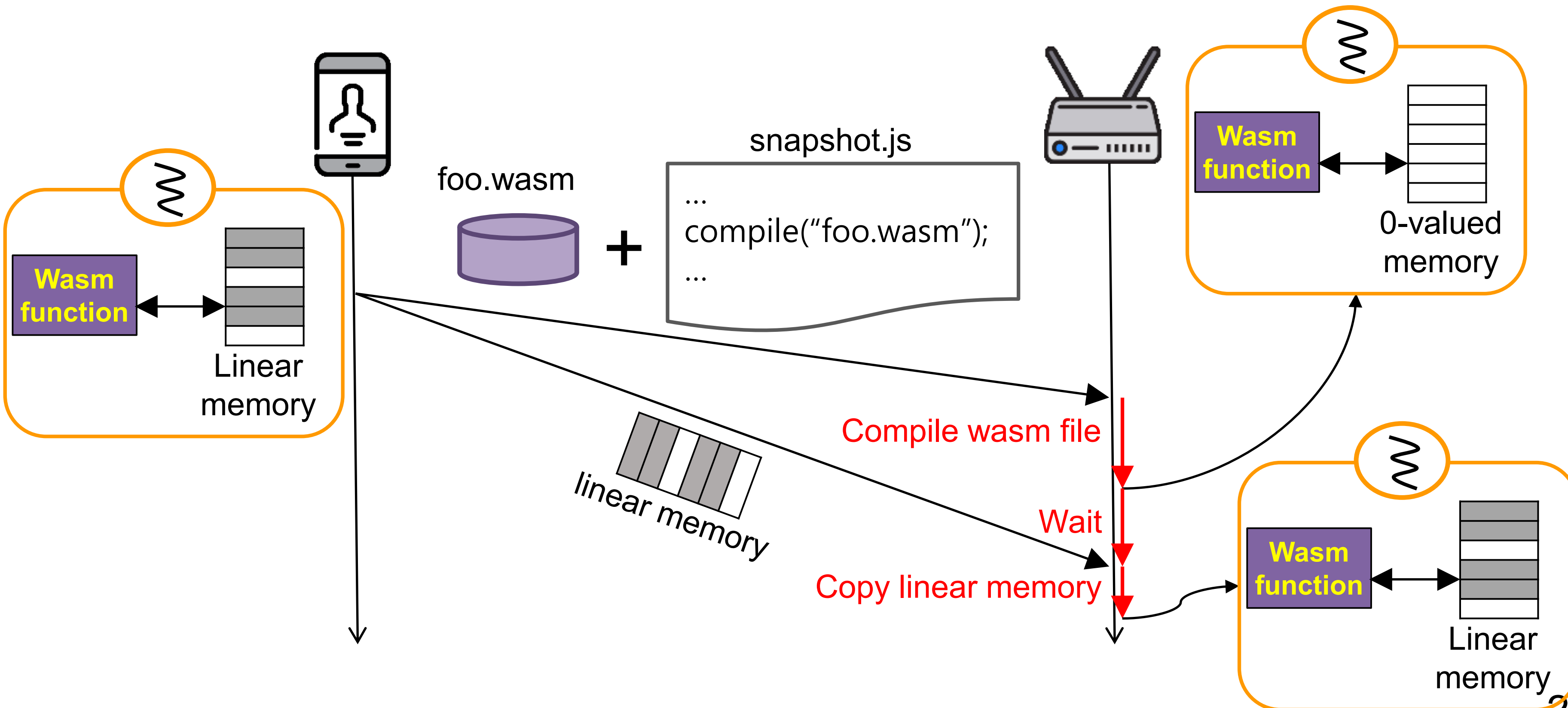
- Serious transmission and recovery overhead



Proposed Method for Wasm Migration

Send a wasm file along with the code that compiles it

Linear memory is *asynchronously* transmitted and lazily restored



EVALUATION

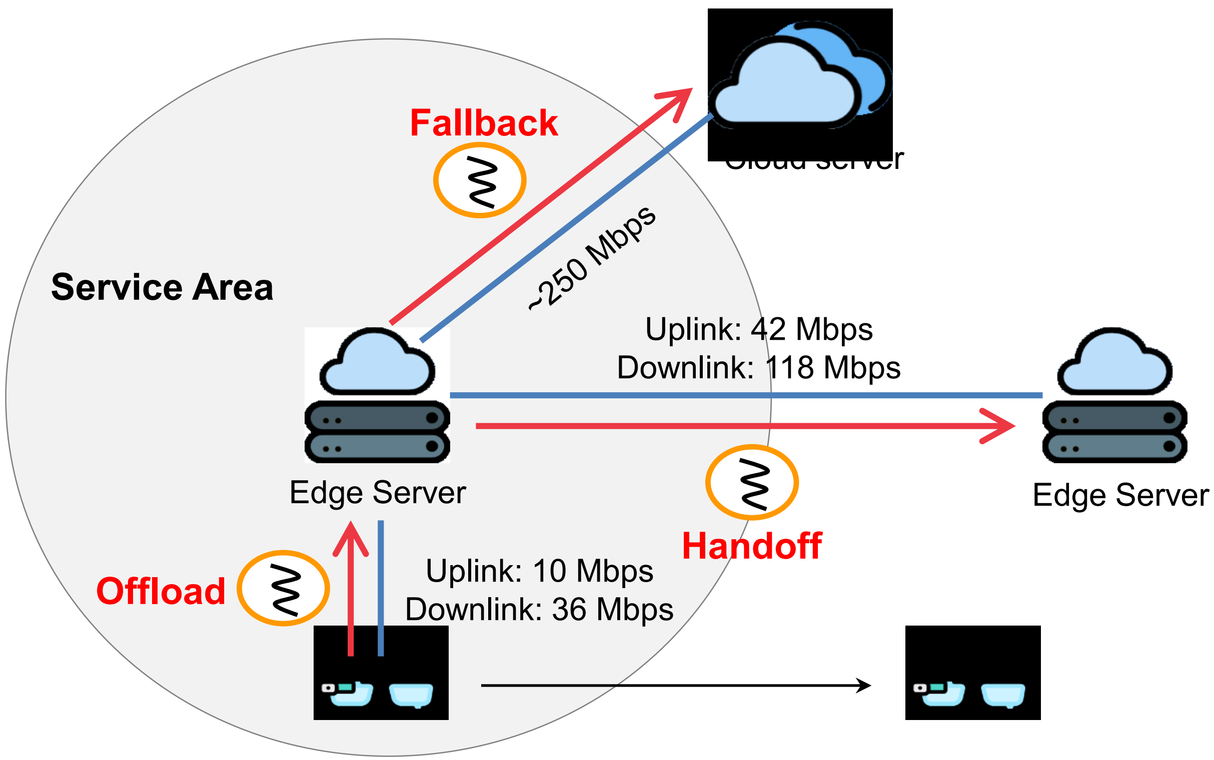
Evaluation Environment

Client: Odroid XU4 (ARM 2-core CPU 2.0 GHz and 2 GB Memory) with **chromium**

Edge server: PC (x86 4-core CPU 3.6 GHz and 16~32 GB Memory) with **Node.js**

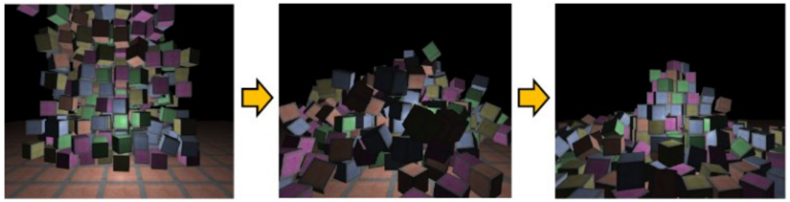
Cloud server: Google cloud (8 vCPU 2.0 GHz and 32 GB memory) with **Node.js**

Network: Average internet speed of US in April 2019 (mobile network, fixed broadband)

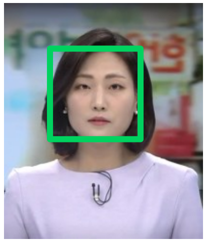


Test Applications

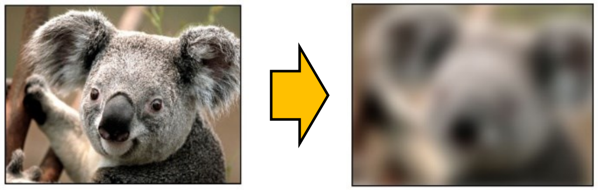
1. Physics simulation (ammo.js)



2. Face detection (OpenCV.js)



3. Blur filter (web-dsp)



Web Worker Migration Time

Migrating a web worker was significantly faster than migrating a Node.js VM instance

- Web worker migration does not need migration of base system

Mobile-to-edge took a long time for migration, due to

- low mobile network speed, slow mobile device
- But, it may happen infrequently

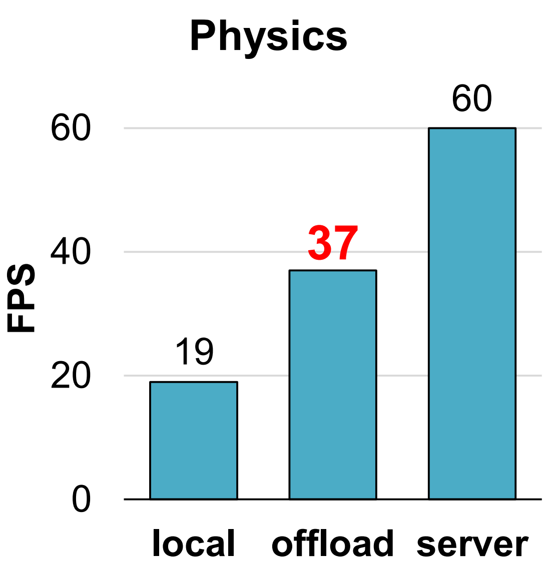
Migration time	VM migration	Web Worker Migration		
	Node.js instance	**physics simulation**	**face detection**	**blur filter**
mobile to edge	18.2	**3.1**	**11.9**	**0.39**
edge to edge	7.9	**1.0**	**3.8**	**0.15**
edge to cloud	7.7	**1.5**	**4.1**	**0.22**

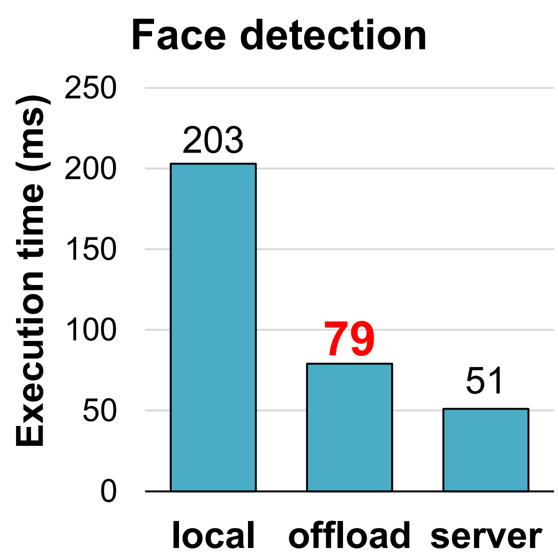
Unit: Second

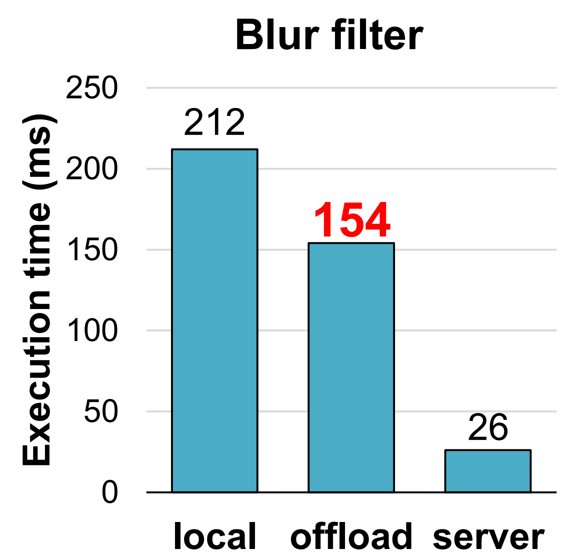
App Execution Performance

Offloading of wasm code significantly improved app performance

- Achieved 37 FPS in physic app
- Achieved 2.6x speedup in face app, and 1.4x in filter app
  - Low speedup in filter app is due to sending input/output images







Higher is better

Lower is better

Conclusion

We proposed a lightweight, state-preserving edge computing framework for web apps

The system migrates web worker using *snapshot*

Experiment showed promising results in both migration time and app performance

THANK YOU

Q & A