LSU ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

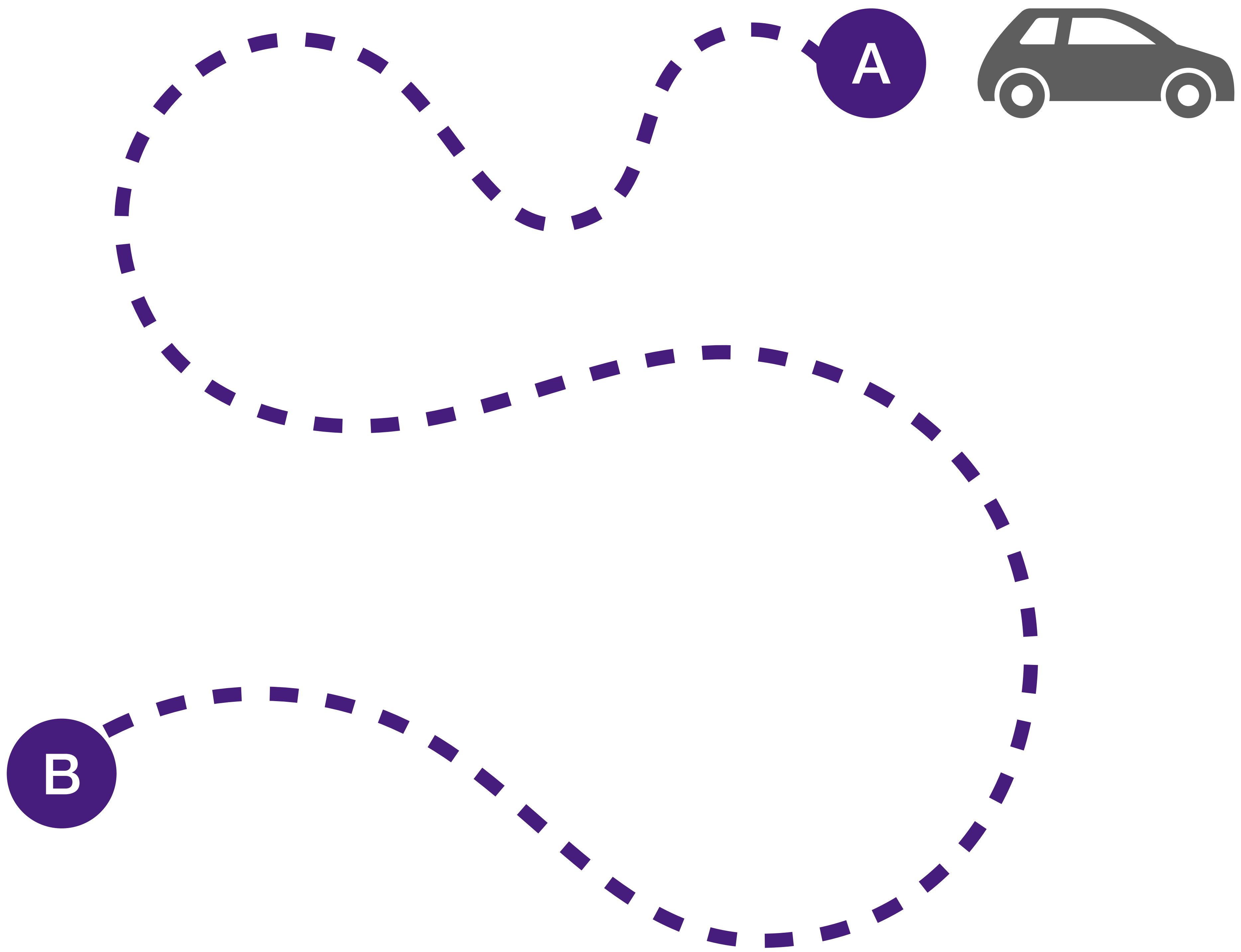
RainbowCake: Mitigating Cold-starts in Serverless with Layer-wise Container Caching and Sharing

Hanfei Yu[1], Rohan Basu Roy[2], Christian Fontenot[1], Devesh Tiwari[2], Jian Li[3], Hong Zhang[4], Hao Wang[1], Seung-Jong Park[5]

Louisiana State University[1], Northeastern University[2], Stony Brook University[3], University of Waterloo[4], Missouri University of Science and Technology[5]

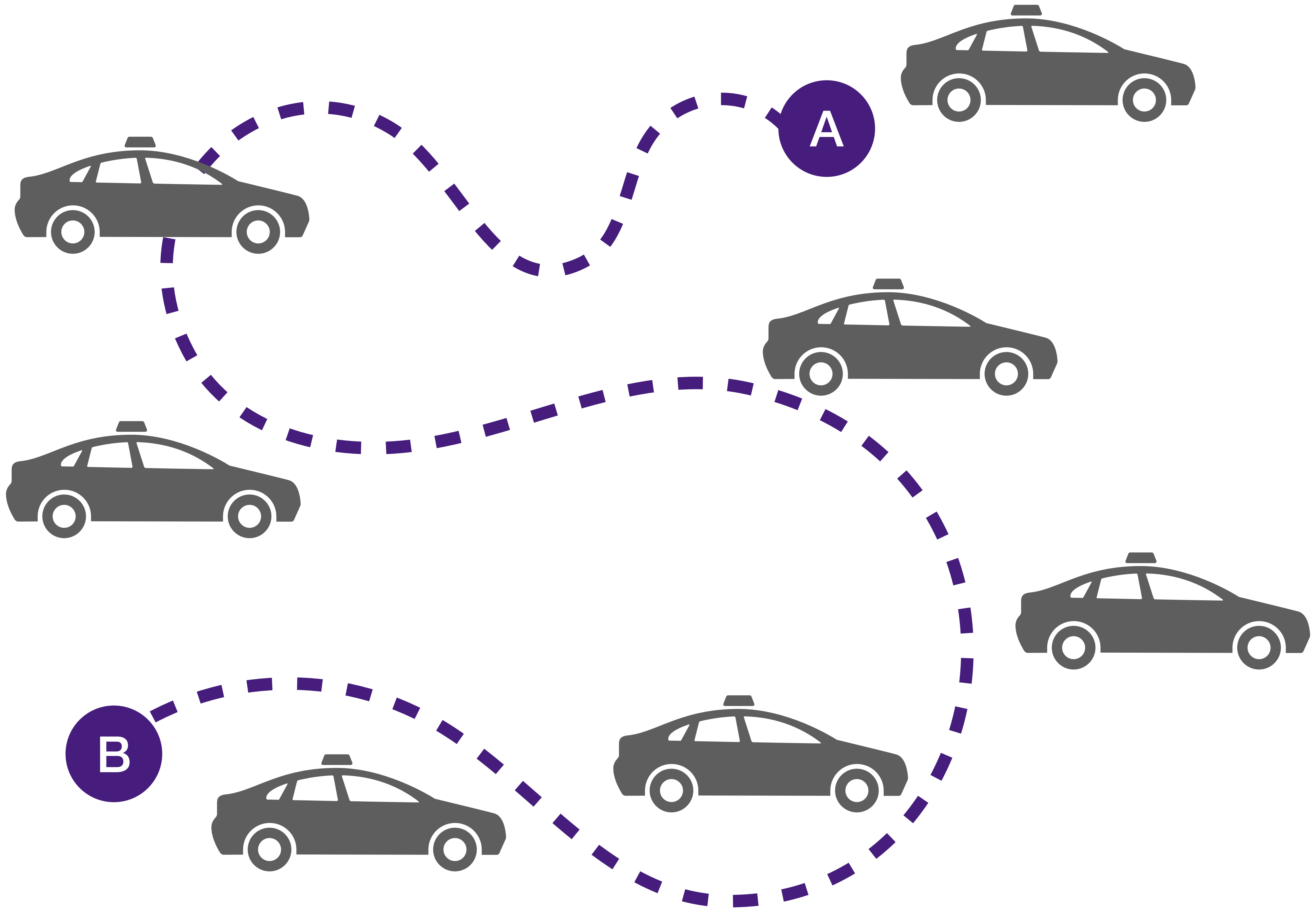
IntelliSys Lab

Cloud



Car rental

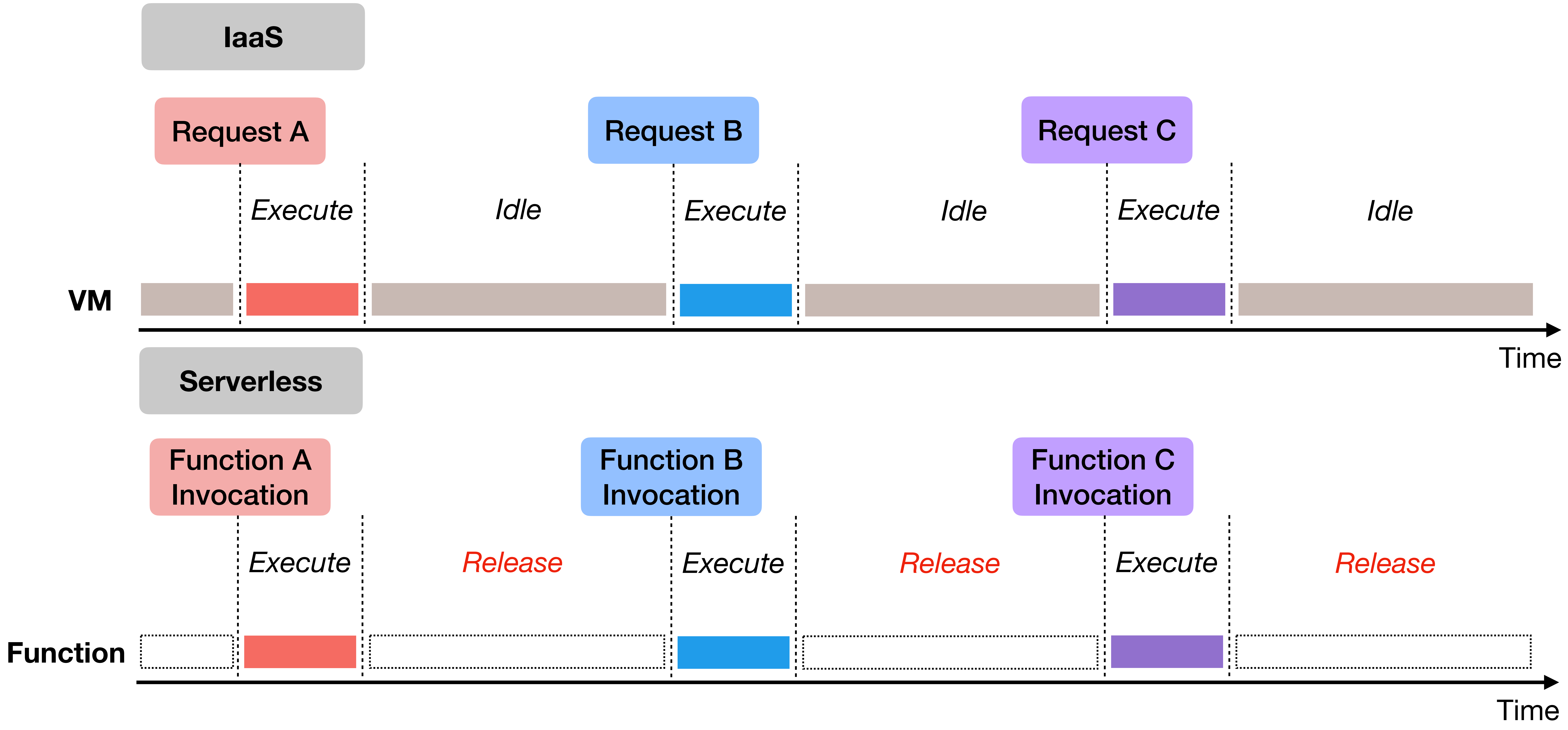
Serverless



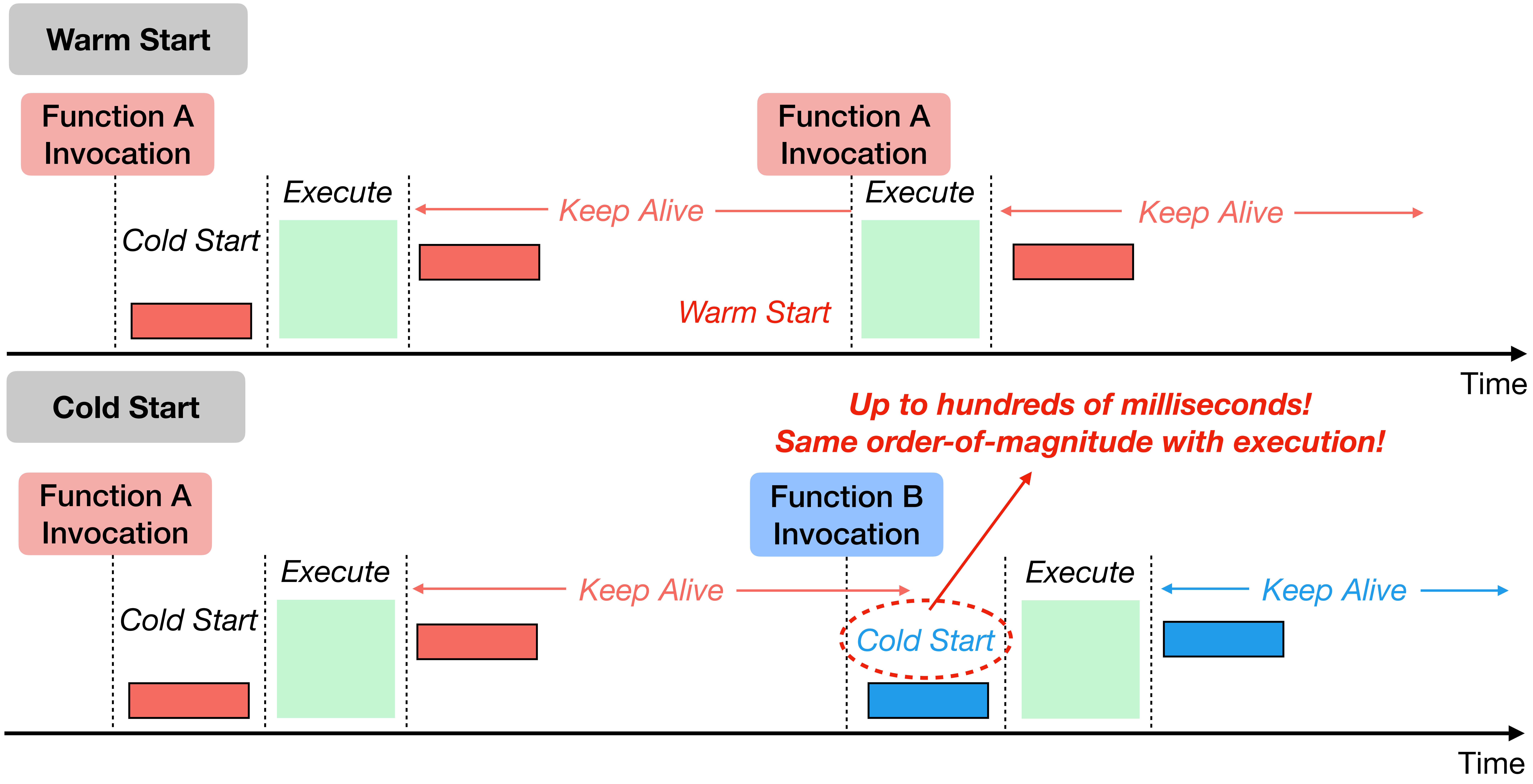
***Cruise* (Self-driving Taxi)**

From *A Serverless Vision for Cloud Computing*
by Prof. Ana Klimovic

IaaS Cloud vs. Serverless



Cold-start in Serverless



Why is Cold-start Hard to Handle

Highly volatile

50% functions

have varying

invocation patterns

Bursty workloads

Workload arrives

45% once per hour

81% once per minute

Hard-to-predict

80% functions

frequently experience

cold startups

Shahrad, Mohammad, et al. “Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider.” ATC’20

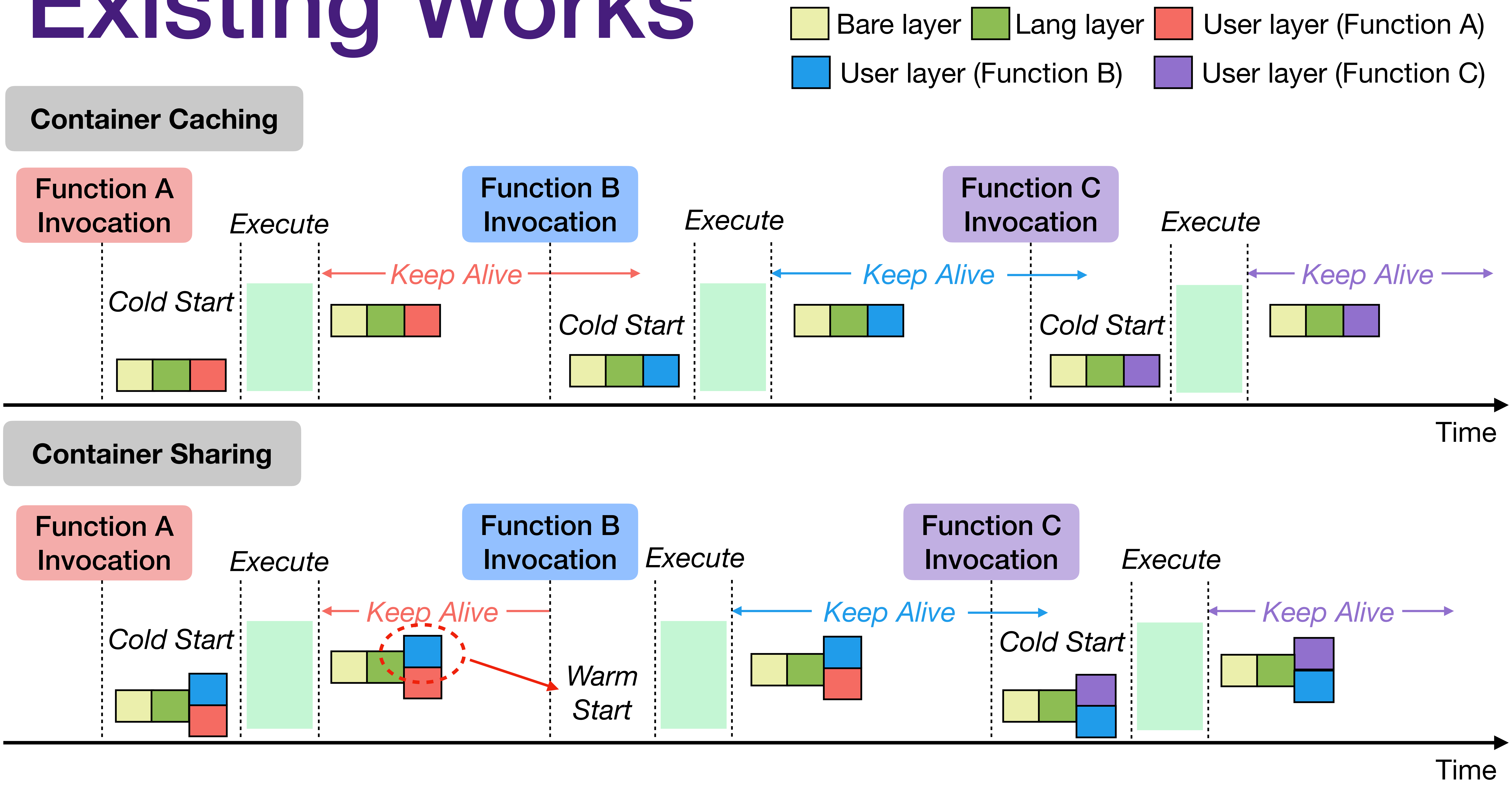
Design Space

Cold-start Mitigation

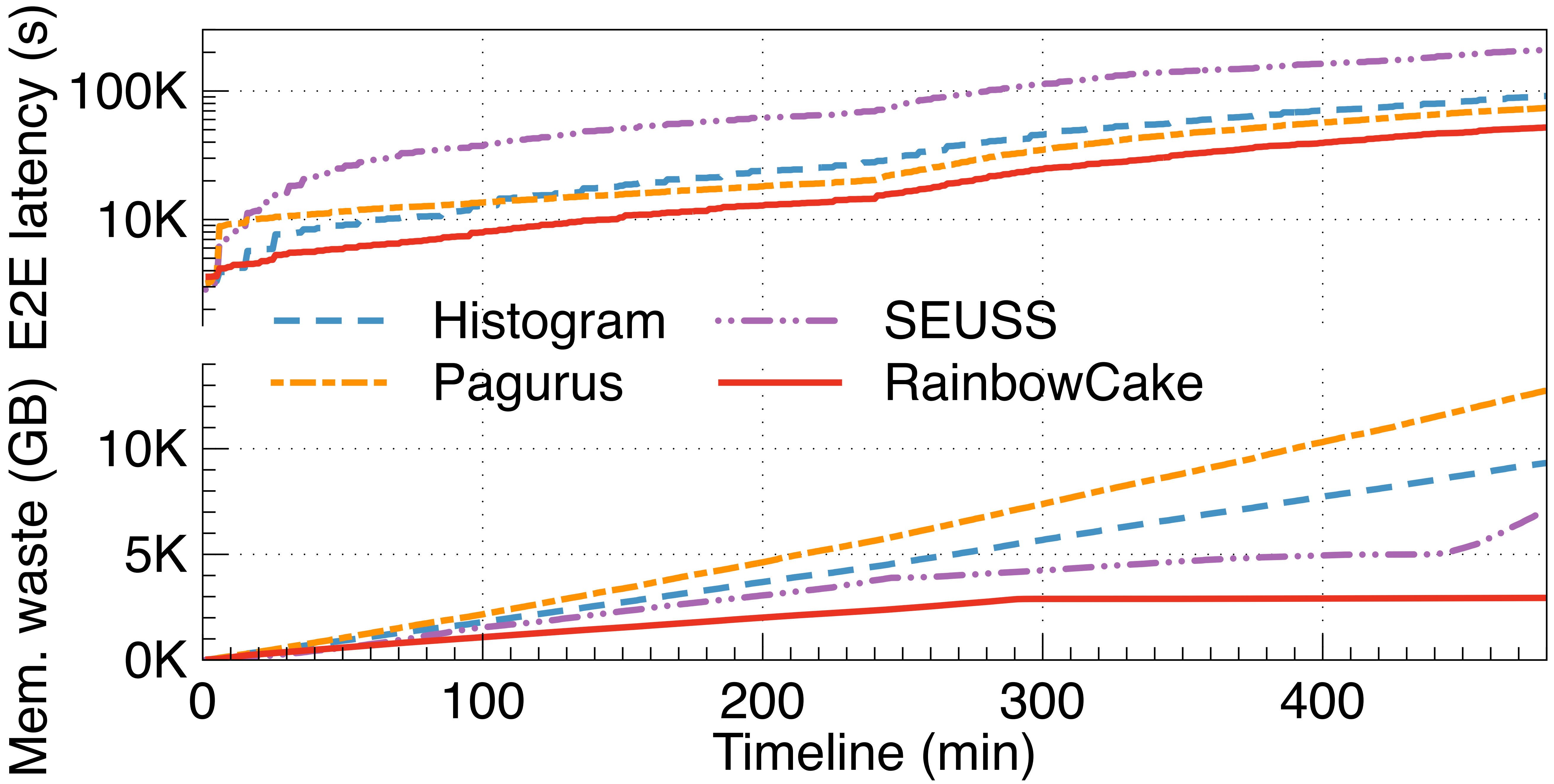
- **New Container Techniques** (Firecracker [1], gVisor [27])
- **Container Caching**
  - **Checkpoint** (Catalyzer [21], Prebaking [55])
  - **Full Caching** (FaaSCache [25], IceBreaker [48])
  - **Partial Caching** (SEUSS [13], FlashCube [39])
- **Container Sharing** (SOCK [43], Pagurus [37])
- **Memory Optimization** (Medes [49])

Detailed references in our paper:
Yu, Hanfei, et al. “RainbowCake: Mitigating Cold-starts in Serverless with Layer-wise Container Caching and Sharing.” ASPLOS’24

Existing Works



Limitations of Existing Works



Can we achieve less cold-starts and low memory waste at the same time?

(Partial) container caching

- Pro: low memory waste
- Con: insufficient startup latency reduction

Container sharing

- Pro: less cold-starts
- Con: high memory waste

RainbowCake

Cold-start Mitigation

- **New Container Techniques** (Firecracker [1], gVisor [27])
- **Container Caching**
  - **Checkpoint** (Catalyzer [21], Prebaking [55])
  - **Full Caching** (FaaSCache [25], IceBreaker [48])
  - **Partial Caching** (SEUSS [13], FlashCube [39])
- **Container Sharing** (SOCK [43], Pagurus [37])
- **Memory Optimization** (Medes [49])

RainbowCake (Partial Caching + Container Sharing)

Design Goals

Mitigate cold-starts with minimal resource waste ← Fine-grained layer-wise breakdown

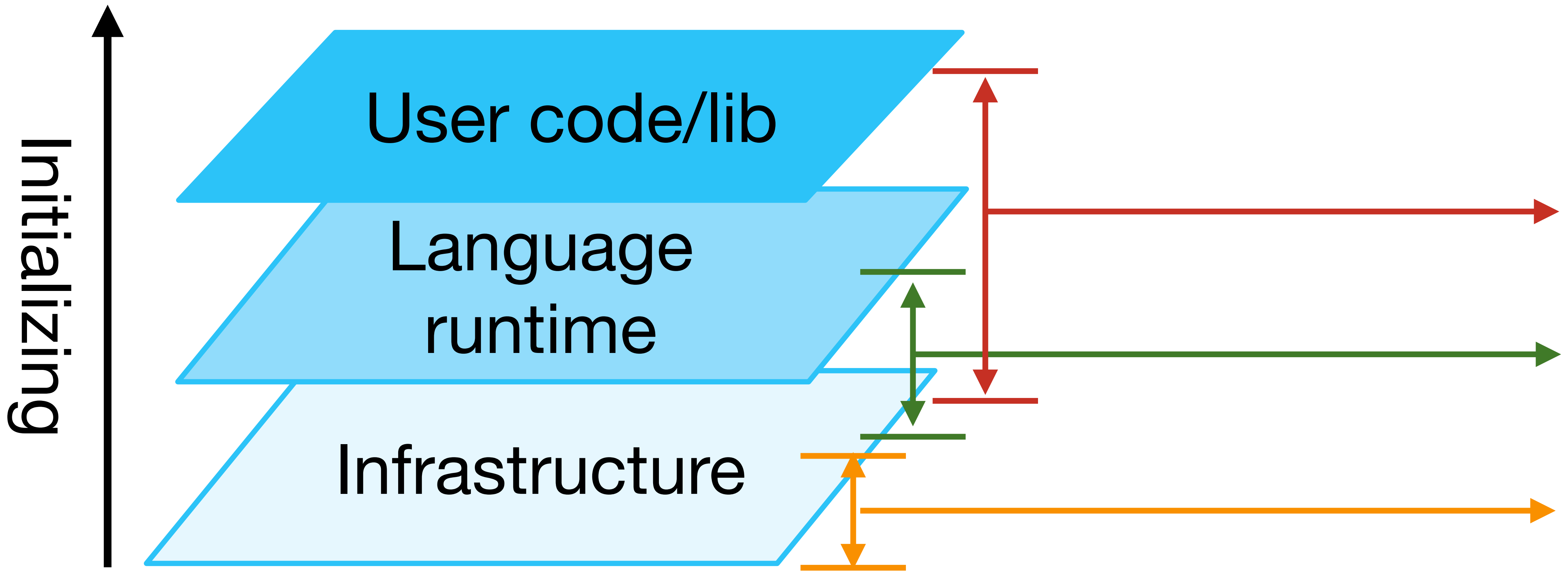
Tolerance to burstiness and mispredictions ← Sharing-aware layer pre-warming and keep-alive

Lightweight and high scalability ← Generic layer design for compatibility

Layered Container Structure

Function startup goes through three layers:

- **Bare layer:** infrastructure, environment, and utility preparation
- **Lang layer:** language runtime creation
- **User layer:** load user code and any necessary libraries

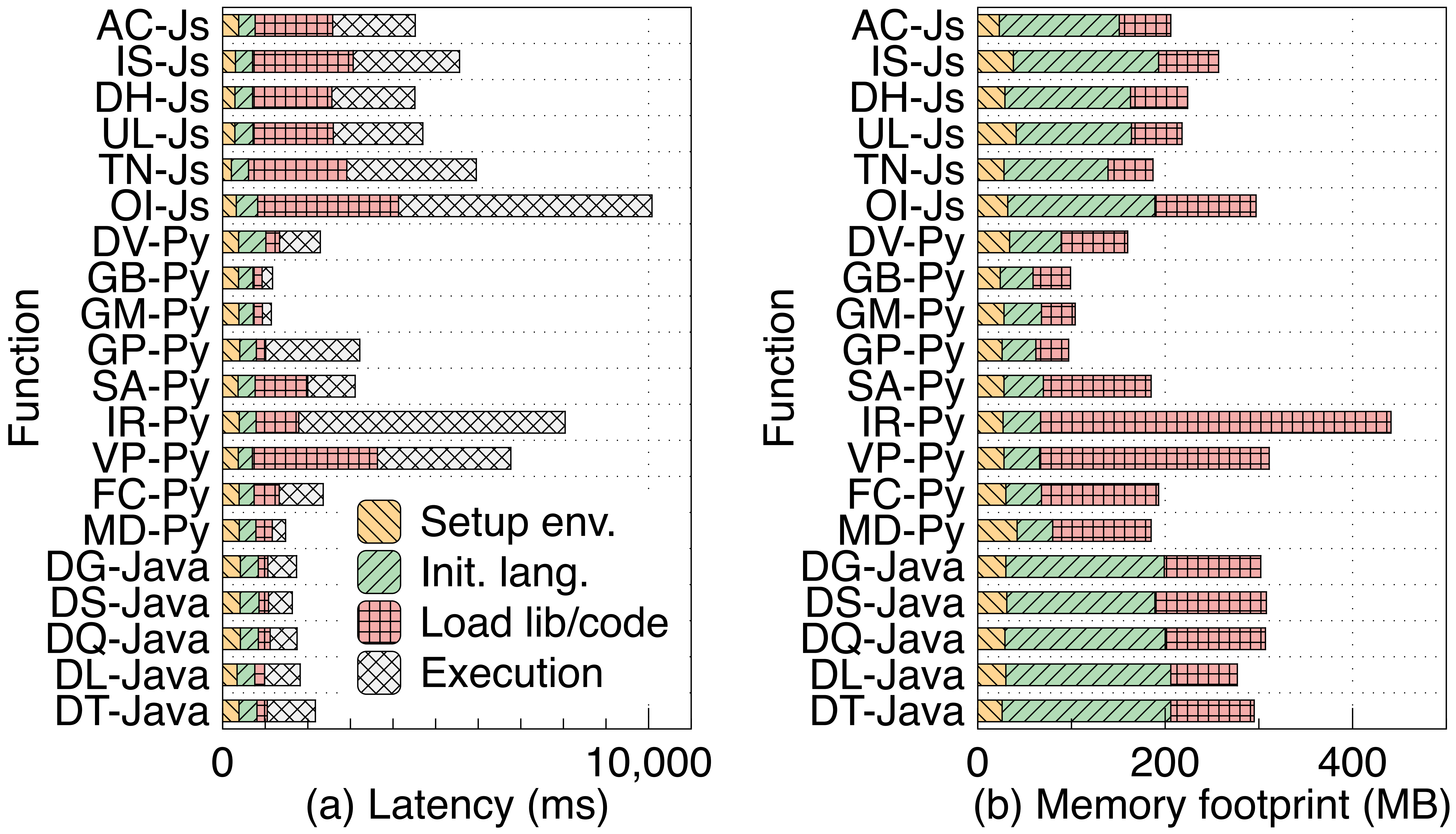


	Memory	Latency	Compatibility
User container	High	Low	Low
Lang container	Medium	Medium	Medium
Bare container	Low	High	High

Characterization of Three Layers

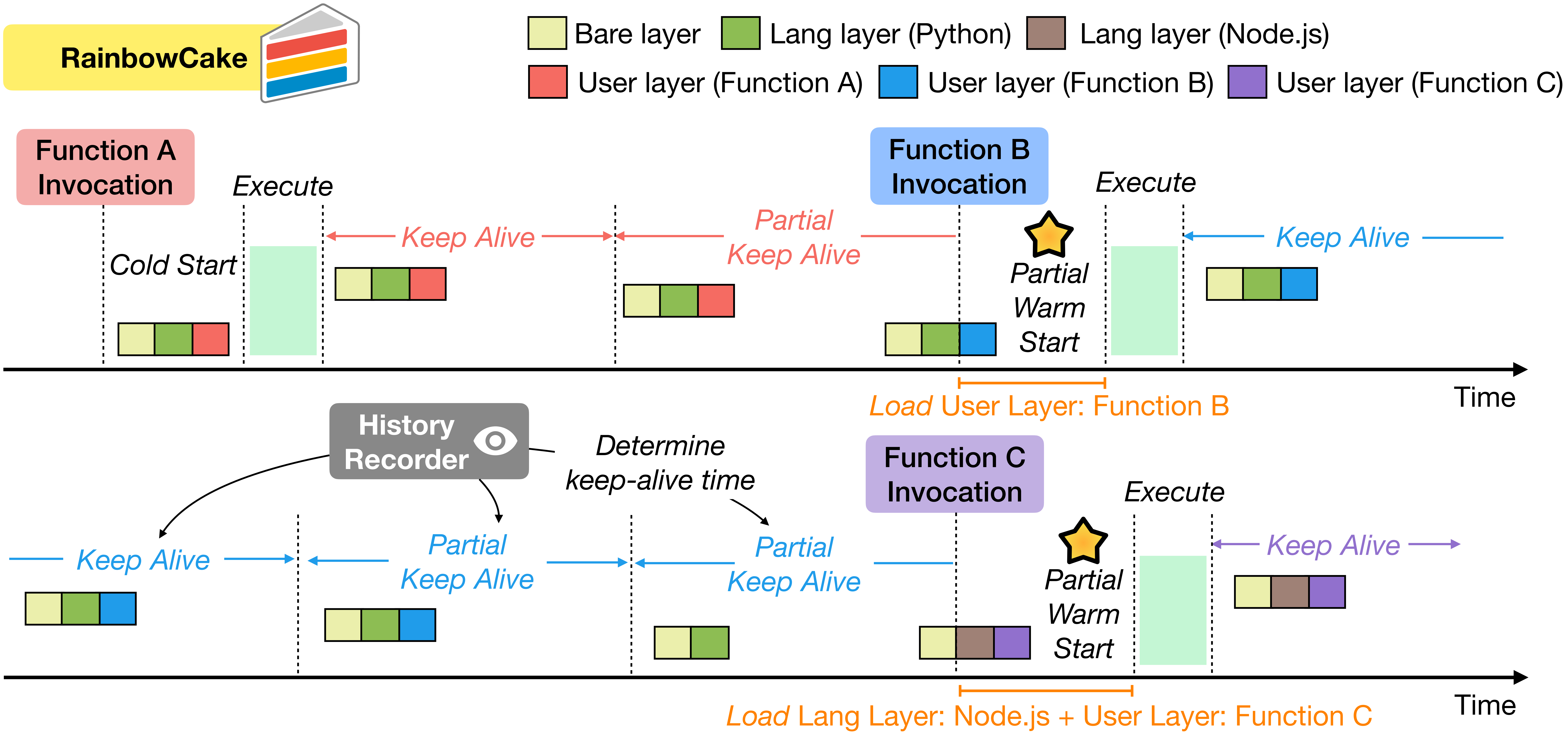
We evaluate 20 realistic functions from three serverless benchmark suites

Layered structures can be observed for all functions



Copik, Marcin, et al. “SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing.” Middleware’21
Cordingly, Robert, et al. "Implications of Programming Language Selection for Serverless Data Processing Pipelines.” CBDCom’20
Shahrad, Mohammad, et al. "Architectural Implications of Function-as-a-Service Computing." MICRO’19

RainbowCake Workflow



RainbowCake Pre-warming

Algorithm 1: *RainbowCake*'s Pre-warming

```
async def SchedulePrewarm(function_id, IAT):
    Sleep(IAT) /* Wait until next request */
    if Available(function_id) is False then
        /* Pre-warm if no warm ones */
        PrewarmContainer(function_id, type=User)
    else
        /* Skip if warm containers exist */
        pass
    return
while function invocation arrives do
    function_id ← function.get_id()
    next_IAT ← Poisson(function_id, type=User)
    /* Asynchronous execution */
    SchedulePrewarm(function_id, next_IAT)
```

Asynchronous pre-warming event scheduling

RainbowCake Pre-warming

Algorithm 1: *RainbowCake*'s Pre-warming

```
async def SchedulePrewarm(function_id, IAT):
    Sleep(IAT) /* Wait until next request */
    if Available(function_id) is False then
        /* Pre-warm if no warm ones */
        PrewarmContainer(function_id, type=User)
    else
        /* Skip if warm containers exist */
        pass
    return
while function invocation arrives do
    function_id ← function.get_id()
    next_IAT ← Poisson(function_id, type=User)
    /* Asynchronous execution */
    SchedulePrewarm(function_id, next_IAT)
```

Asynchronous pre-warming event scheduling

Pre-warm a User container if no warm ones

RainbowCake Pre-warming

Algorithm 1: *RainbowCake*'s Pre-warming

```
async def SchedulePrewarm(function_id, IAT):
    Sleep(IAT) /* Wait until next request */
    if Available(function_id) is False then
        /* Pre-warm if no warm ones */
        PrewarmContainer(function_id, type=User)
    else
        /* Skip if warm containers exist */
        pass
    return
while function invocation arrives do
    function_id ← function.get_id()
    next_IAT ← Poisson(function_id, type=User)
    /* Asynchronous execution */
    SchedulePrewarm(function_id, next_IAT)
```

Asynchronously schedules pre-warming events

Pre-warm a User container if no warm ones

Otherwise, skip this pre-warming event

RainbowCake Pre-warming

Algorithm 1: *RainbowCake*'s Pre-warming

```
async def SchedulePrewarm(function_id, IAT):
    Sleep(IAT) /* Wait until next request */
    if Available(function_id) is False then
        /* Pre-warm if no warm ones */
        PrewarmContainer(function_id, type=User)
    else
        /* Skip if warm containers exist */
        pass
    return
while function invocation arrives do
    function_id ← function.get_id()
    next_IAT ← Poisson(function_id, type=User)
    /* Asynchronous execution */
    SchedulePrewarm(function_id, next_IAT)
```

Asynchronous pre-warming event scheduling

Pre-warm a User container if no warm ones

Otherwise, skip this pre-warming event

Whenever an invocation arrives

RainbowCake Pre-warming

Algorithm 1: *RainbowCake*'s Pre-warming

```
async def SchedulePrewarm(function_id, IAT):
    Sleep(IAT) /* Wait until next request */
    if Available(function_id) is False then
        /* Pre-warm if no warm ones */
        PrewarmContainer(function_id, type=User)
    else
        /* Skip if warm containers exist */
        pass
    return
while function invocation arrives do
    function_id ← function.get_id()
    next_IAT ← Poisson(function_id, type=User)
    /* Asynchronous execution */
    SchedulePrewarm(function_id, next_IAT)
```

Asynchronous pre-warming event scheduling

Pre-warm a User container if no warm ones

Otherwise, skip this pre-warming event

Whenever an invocation arrives

Fit Poisson distribution to predict next Inter-arrival time (IAT)

RainbowCake Keep-alive

Compute Time-to-live given a container and its predicted IAT

Algorithm 2: *RainbowCake*'s Keep-alive

```
def ComputeTTL(container, IAT):
    t ← container.get_startup_latency()
    m ← container.get_memory_footprint()
    β ← (α × t)/((1 − α) × m) /* Equation 6 */
    return Min(IAT, β)
while container timeouts do
    function_id ← container.get_function_id()
    layer ← container.get_type()
    if layer is Bare then
        /* Bare containers timeout */
        container.kill()
    else
        /* User or Lang containers timeout */
        container.downgrade()
        layer ← container.get_type()
        next_IAT ← Poisson(function_id, layer)
        TTL ← ComputeTTL(container, next_IAT)
        SetContainerTimeout(container, TTL)
```

RainbowCake Keep-alive

Compute Time-to-live given a container and its predicted IAT

Whenever a container ends its keep-alive period

Algorithm 2: *RainbowCake*'s Keep-alive

```
def ComputeTTL(container, IAT):
    t ← container.get_startup_latency()
    m ← container.get_memory_footprint()
    β ← (α × t)/((1 − α) × m) /* Equation 6 */
    return Min(IAT, β)
while container timeouts do
    function_id ← container.get_function_id()
    layer ← container.get_type()
    if layer is Bare then
        /* Bare containers timeout */
        container.kill()
    else
        /* User or Lang containers timeout */
        container.downgrade()
        layer ← container.get_type()
        next_IAT ← Poisson(function_id, layer)
        TTL ← ComputeTTL(container, next_IAT)
        SetContainerTimeout(container, TTL)
```

RainbowCake Keep-alive

Compute Time-to-live given a container and its predicted IAT

Whenever a container ends its keep-alive period

Terminate if a Bare container times out

Algorithm 2: *RainbowCake*'s Keep-alive

```
def ComputeTTL(container, IAT):
    t ← container.get_startup_latency()
    m ← container.get_memory_footprint()
    β ← (α × t)/((1 − α) × m) /* Equation 6 */
    return Min(IAT, β)

while container timeouts do
    function_id ← container.get_function_id()
    layer ← container.get_type()
    if layer is Bare then
        /* Bare containers timeout */
        container.kill()
    else
        /* User or Lang containers timeout */
        container.downgrade()
        layer ← container.get_type()
        next_IAT ← Poisson(function_id, layer)
        TTL ← ComputeTTL(container, next_IAT)
        SetContainerTimeout(container, TTL)
```

RainbowCake Keep-alive

Compute Time-to-live given a container and its predicted IAT

Whenever a container ends its keep-alive period

Terminate if a Bare container times out

Otherwise, fit Poisson distribution to predict next Inter-Arrival Time (IAT)

Algorithm 2: *RainbowCake*'s Keep-alive

```
def ComputeTTL(container, IAT):
    t ← container.get_startup_latency()
    m ← container.get_memory_footprint()
    β ← (α × t)/((1 − α) × m) /* Equation 6 */
    return Min(IAT, β)

while container timeouts do
    function_id ← container.get_function_id()
    layer ← container.get_type()
    if layer is Bare then
        /* Bare containers timeout */
        container.kill()
    else
        /* User or Lang containers timeout */
        container.downgrade()
        layer ← container.get_type()
        next_IAT ← Poisson(function_id, layer)
        TTL ← ComputeTTL(container, next_IAT)
        SetContainerTimeout(container, TTL)
```

Implementation

RainbowCake is prototyped on top of Docker and Apache OpenWhisk

Bare Layer	**Lang Layer**	**User Layer**	**Pre-warming & Keep-alive**
Docker environment	OpenWhisk's Java/Python/Node.js runtimes	OpenWhisk's Container Proxy	OpenWhisk's Container Pool

Evaluation

Testbed

3 nodes
140 AMD EPYC CPU cores
240 GB Memory

Metrics

Function response latency
Memory waste

Baselines

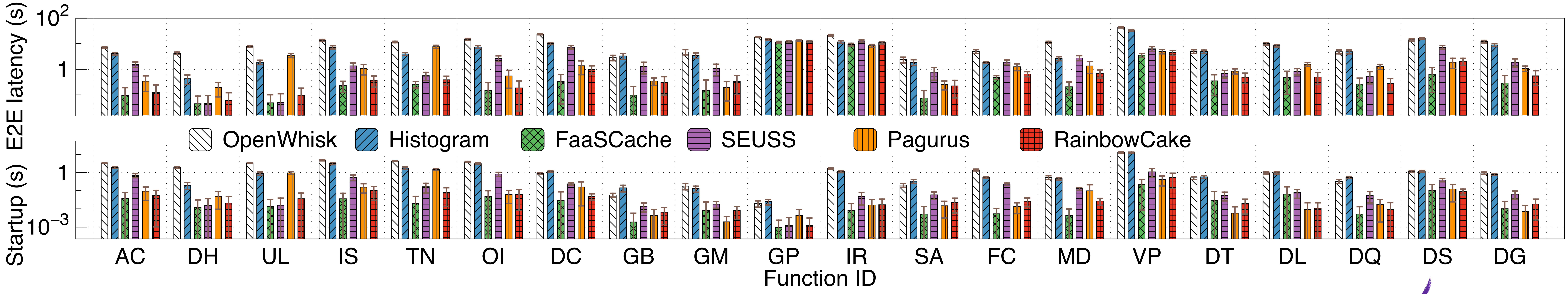
OpenWhisk default
Histogram
FaaSCache
SEUSS
Pagurus

Traces

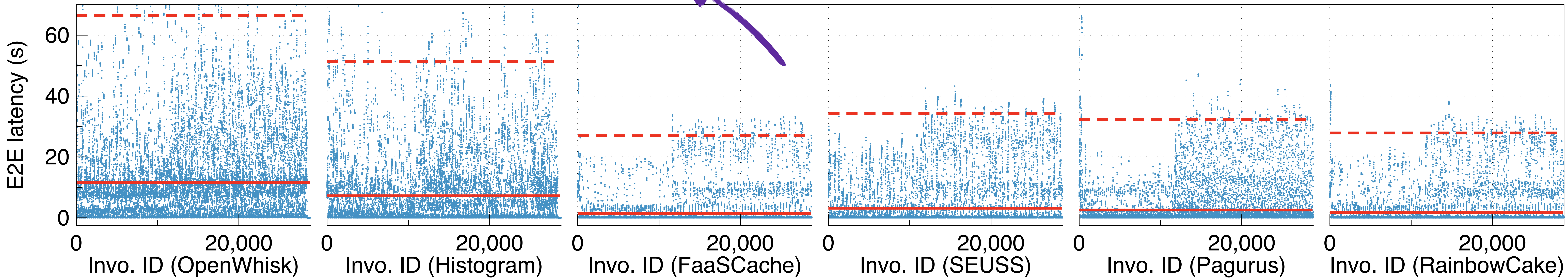
Azure Functions traces
8-hour workloads

Histogram: Shahrad, Mohammad, et al. “Serverless in the Wild: Characterizing and Optimizing the Serverless…” ATC’20
FaaSCache: Fuerst, Alexander, et al. "SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing." ASPLOS’21
SEUSS: Cadden, James, et al. "SEUSS: Skip Redundant Paths to Make Serverless Fast." EuroSys’20
Pagurus: Li, Zijun, et al. "Help Rather Than Recycle: Alleviating Cold Startup in Serverless Computing…” ACSOS 20

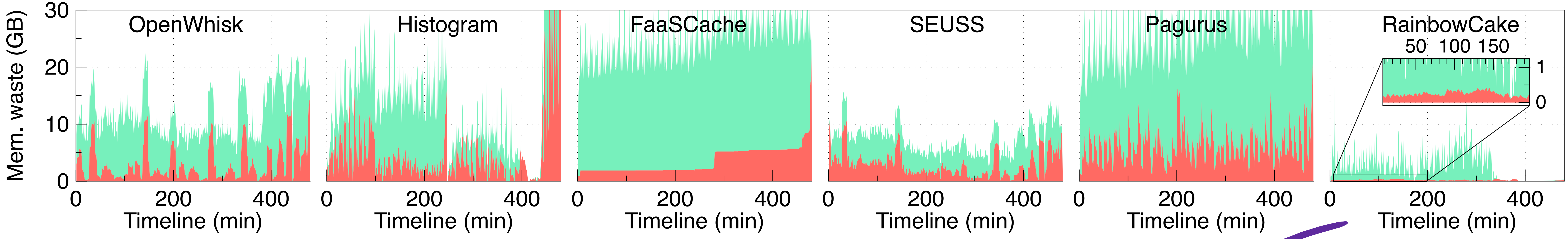
End-to-end Latency



RainbowCake achieves similar or better **function** and **invocation** latency than other baselines



Memory Footprint



RainbowCake significantly reduces **memory waste** compared to other baselines

Combining container caching and sharing

Layer-wise pre-warming and keep-alive decisions

Mitigating cold-starts with minimal memory waste

RainbowCake

68%

Function startup latency reduction

77%

Memory waste reduction

RainbowCake Code Repo:

https://github.com/IntelliSys-Lab/RainbowCake-ASPLOS24

Corresponding Author:

Hanfei Yu <hyu25@lsu.edu>

Hao Wang <haowang@lsu.edu>