When Serverless Meets Servers

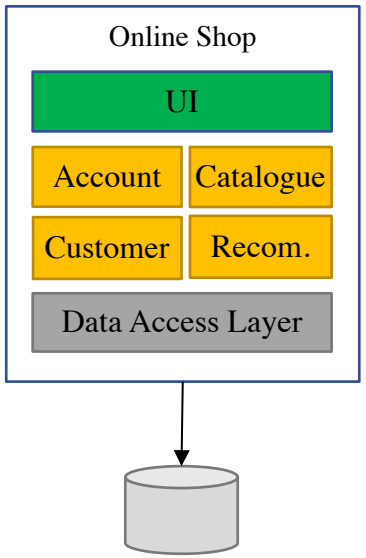
Boris Grot

Edinburgh Architecture & Systems Lab (EASE)

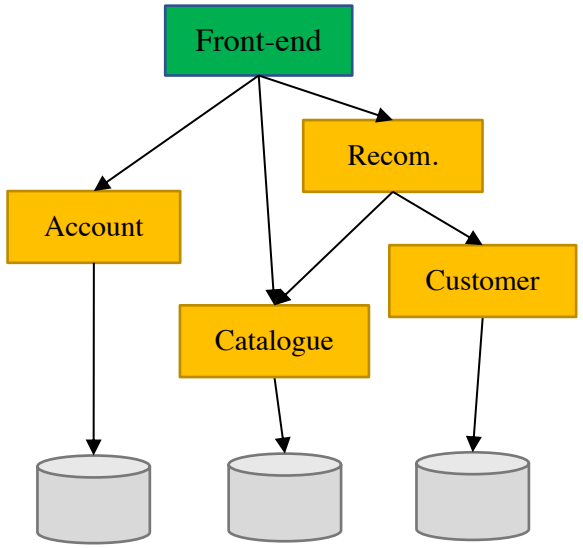
University of Edinburgh

Cloud Applications: from Monoliths to Serverless

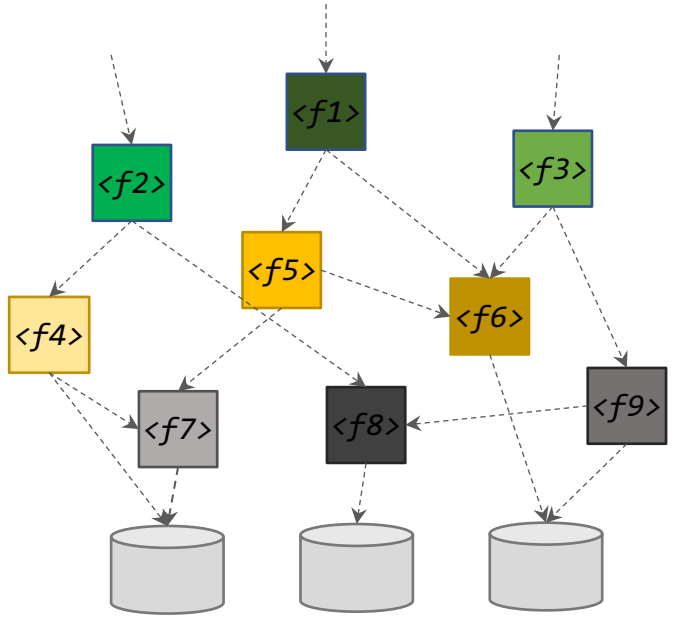
Monolithic app



Microservices



Serverless

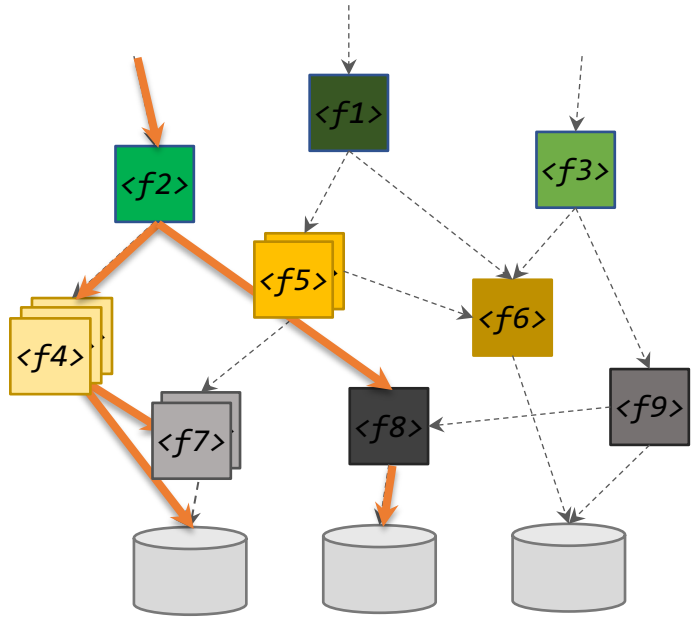


Trend toward **greater modularity** and **disaggregation** in cloud applications

Serverless 101

Datacenter application organized as a collection of stateless functions

- Functions invoked on-demand
  - via triggers (e.g., user click) or by another function
- Functions are stateless: facilitates on-demand scale-in/scale-out
- Developers: pay only per invocation (CPU+memory), not idle time 🙂
  - Key difference from monoliths & microservices!
  - Financial incentive to reduce function footprint
- Cloud providers: high density and utilization at the server level 🙂



What are the implications of the serverless model?

State of Serverless Clouds Today

Good: Programming & deployment simplicity; pay-per-use cost model

Bad: Poor performance & low efficiency

- Frequent scaling due to traffic changes → cold start delays, overprovisioning
- Functions are stateless → communication bottlenecks inherent
- Massive degree of function interleaving on a server → poor uarch efficiency
- …

Ugly: Proprietary serverless stacks across cloud providers

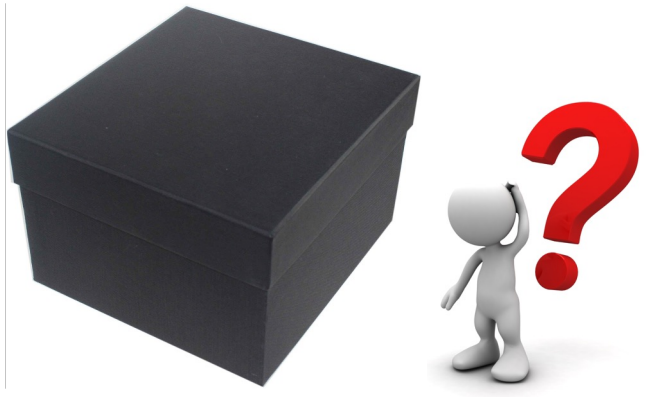
How to study and innovate?

Big challenges are big opportunities for research!

State-of-the-Art in Serverless Experimentation

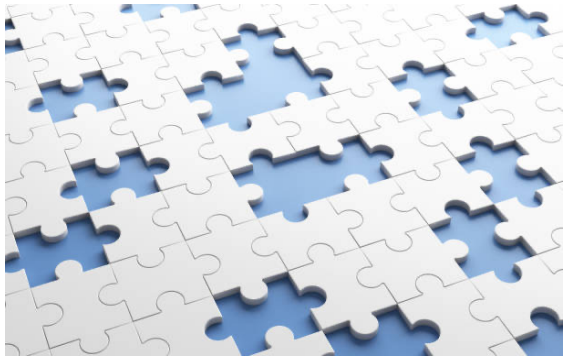
Industry

Bleeding-edge but **proprietary** serverless stacks



Research/academia

Incomplete or **non-representative**



Need for a full-stack open-source framework for serverless research

Idea: Integrate Open-Source Components from across the Industry

Cluster scheduler & Function-as-a-Service API
(Google, Cloud Native Computing Foundation)

Kubernetes + Knative

MicroVM (Amazon, Google)

Firecracker gVisor

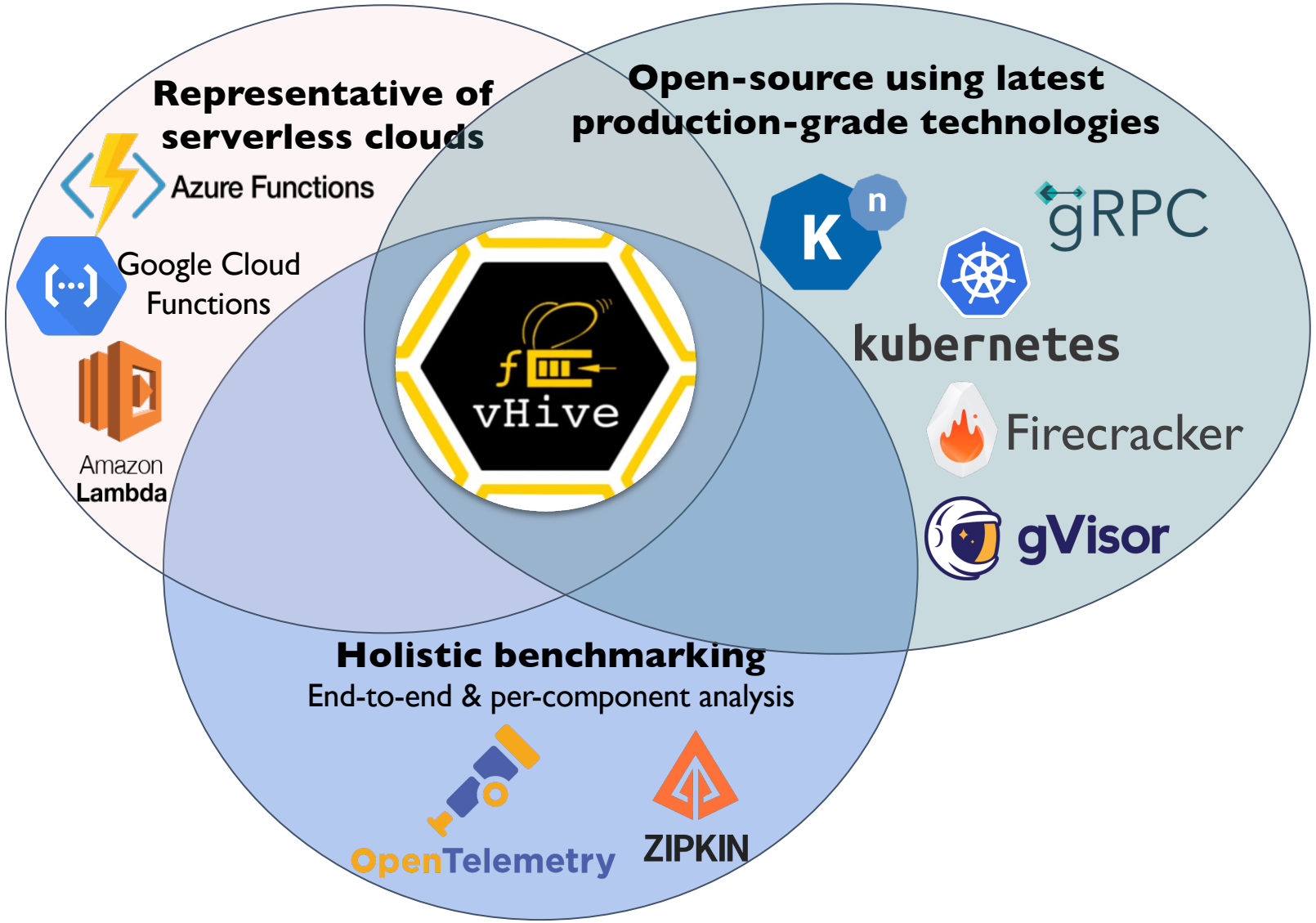
Host management, container runtime
(Cloud Native Computing Foundation)

containerd

Communication (Google)

gRPC

The vHive eco-system



vHive: an open-source serverless stack
github.com/ease-lab/vhive

Representative of today's clouds

- Knative FaaS API, Firecracker & gVisor MicroVMs, Kubernetes
- First to support Firecracker snapshots

Robust methodology & performance analysis tools

vSwarm: a serverless benchmark suite
github.com/ease-lab/vSwarm

Comprehensive real-world benchmarks

- ML training & inference, video analytics & encoding, MapReduce, distributed compilation
- Varied runtimes & function composition patterns
- Data transfers via different mediums (inline, S3)

Gem5-runnable container images

- Enables full-system microarchitectural simulation

vHive in action:

Understanding & Accelerating Lukewarm Invocations

[ISCA'22]

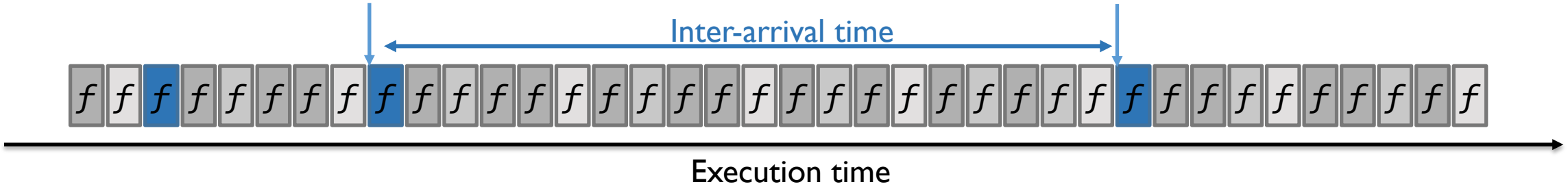
Serverless on a Server

Unique characteristics:

- Short function execution times: a few ms or less is common
  - Contrast: Linux scheduling quantum: 10-20ms
- Small memory footprint: as low as 128MB per instance
- Relatively infrequent invocations (seconds or minutes) [Microsoft Azure @ATC20]

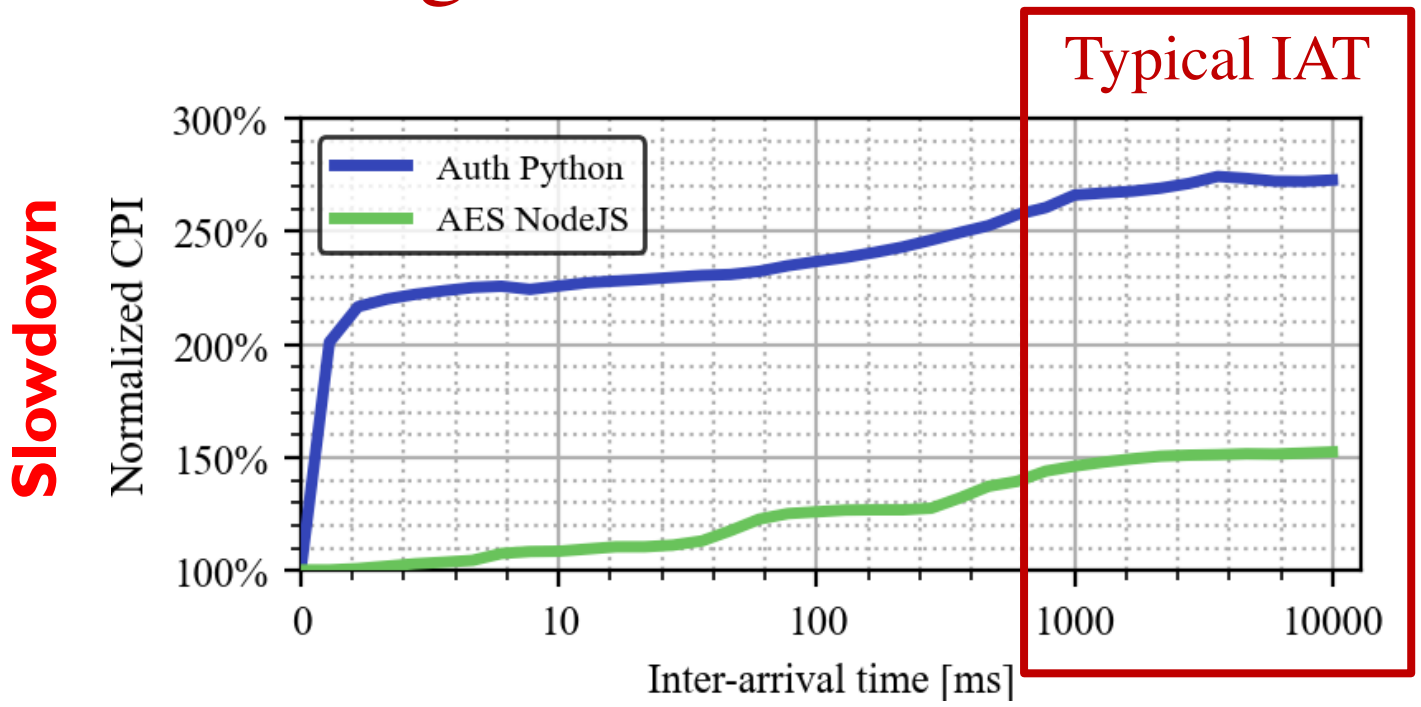
Implications:

- Thousands of functions resident on a server
- **Huge degree of interleaving** between two invocations of the same function



What are the implications for microarchitecture?

Effect of Interleaving



Longer inter-arrival times → Higher degree of interleaving → Higher CPI

Drastic increase in CPI for typical inter-arrival times (IATs)

- Up to 170% CPI increase for IAT > 1s

What causes the increase?

Characterization Methodology

Compare back-to-back to interleaved executions of a function

- Function-under-test runs isolated
- Interleaving modelled by a stressor

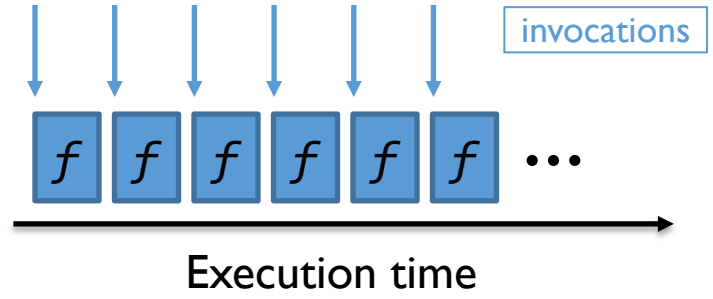
Use **Top-Down Methodology** for analysis

- Machine: Intel Broadwell CPU
  (10 cores, SMT disabled, 32KB L1-I/D, 256KB L2/core, 25MB LLC)
- Collect CPU performance counters

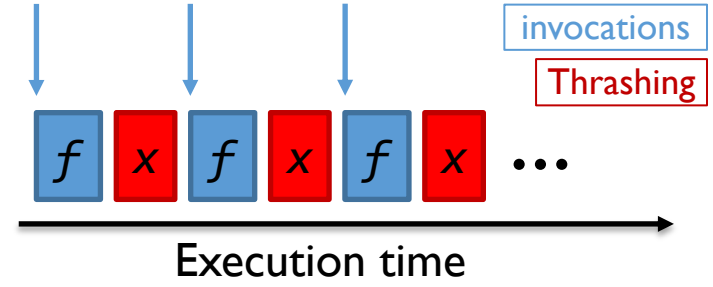
Serverless workloads: 20 functions

- Large variety in functionality and runtimes
- Compiled, JIT-ed and interpreted languages
- Publicly available https://github.com/ease-lab/vSwarm

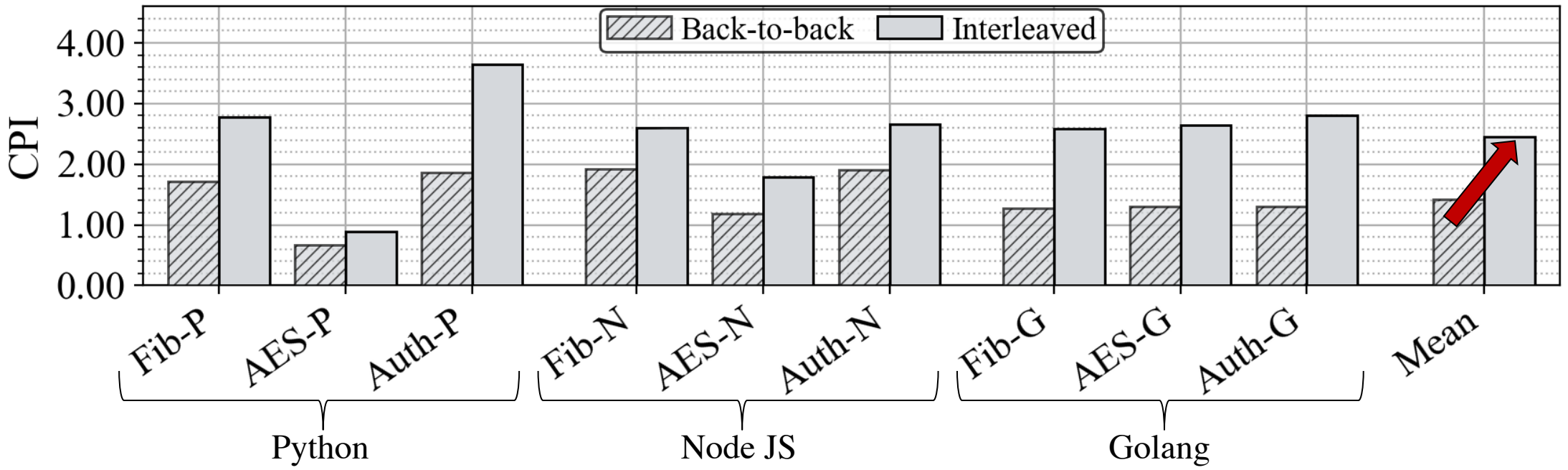
Back-to-Back Execution



Interleaved Execution

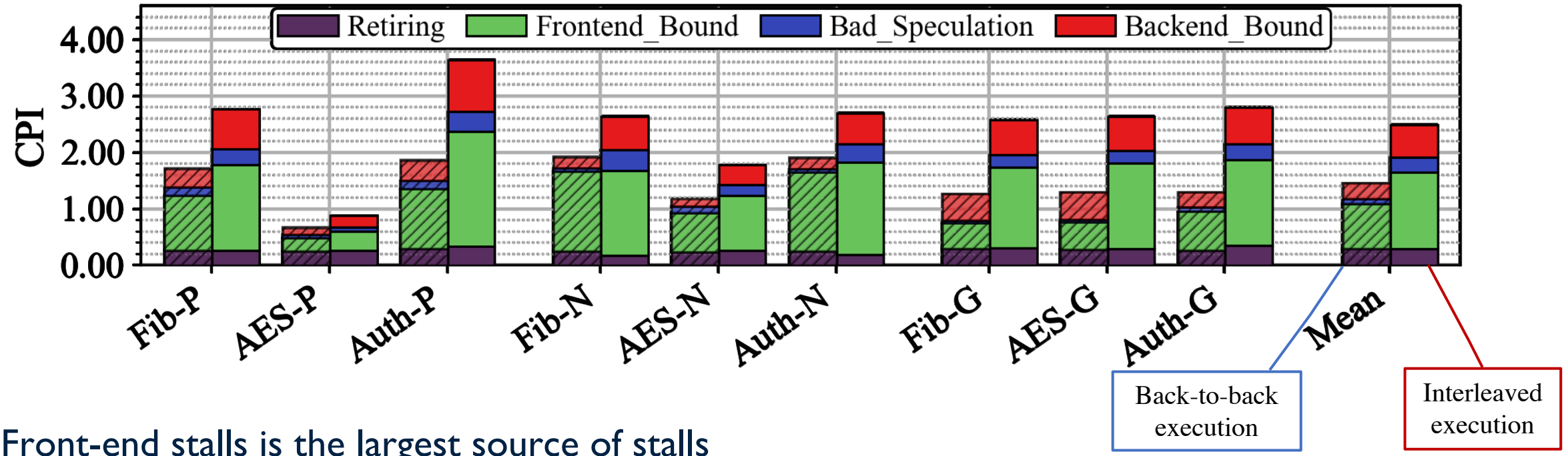


Understanding the Impact of Interleaving



- Interleaving increases the mean CPI by 70%
- Reason: Lukewarm execution
  - Function in memory, but no μ-arch state on-chip

Top-Down CPI Analysis

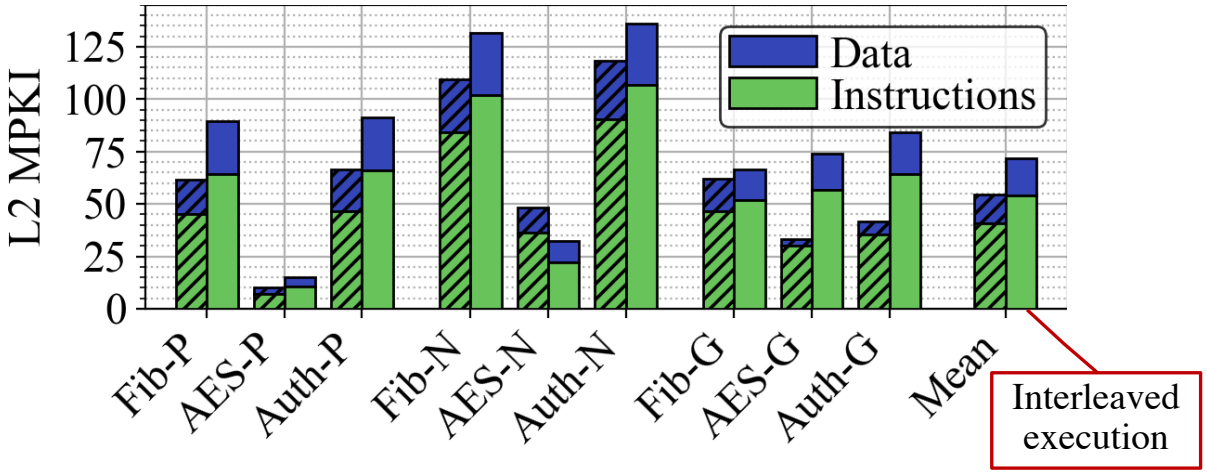


- Front-end stalls is the largest source of stalls
- 56% of additional stall cycles in interleaved execution come from **fetch latency**

Instruction delivery a critical performance bottleneck for warm invocations

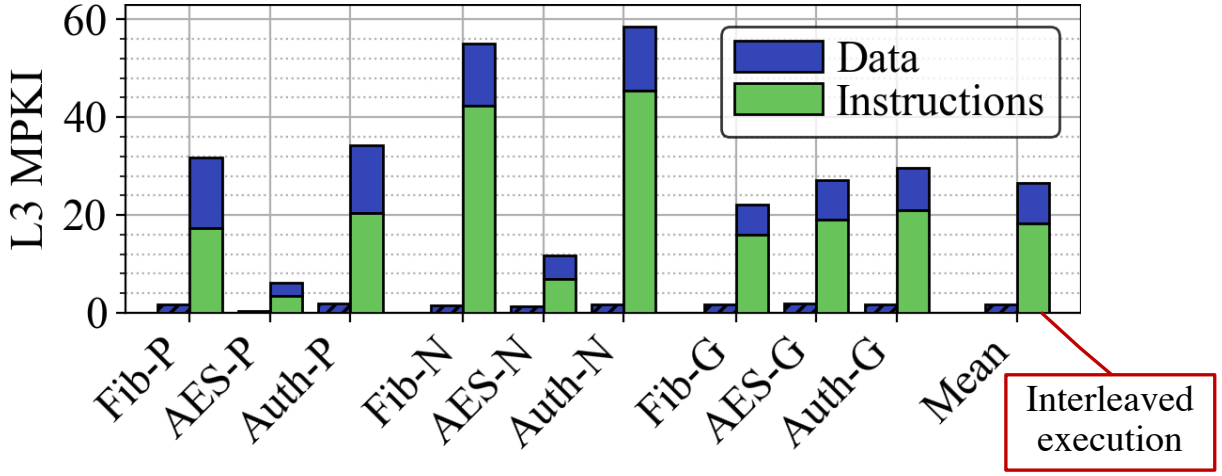
Instruction Fetch Pain Points

L2 Cache *(256KB/core)*



- Serverless workloads frequently miss in L2 cache
  - (50+ MPKI, on average)
- Dominated by instruction misses
- Similar for both back-to-back and interleaved

L3 Cache *(25MB)*



- Almost no L3 instruction misses for back-to-back execution
- Frequent L3 misses for instructions under interleaving (18 MPKI)
  - Instructions fetched from main memory → high stall cycles

L3 instruction misses hurt performance under interleaving

Understanding Instruction Misses

Studied instruction traces from 25 consecutive invocations of each function.

Compared **instruction footprint** & **commonality** at cache-block granularity across invocations

Two key insights:

1. **High commonality** across invocations
   - > 85% of cache blocks are the same in all invocations
2. **Large instruction footprint**: 300KB-800KB
   - Deep software stacks result in large amount of code

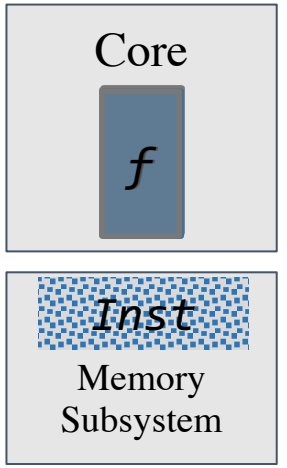
Identified a common problem for serverless functions:

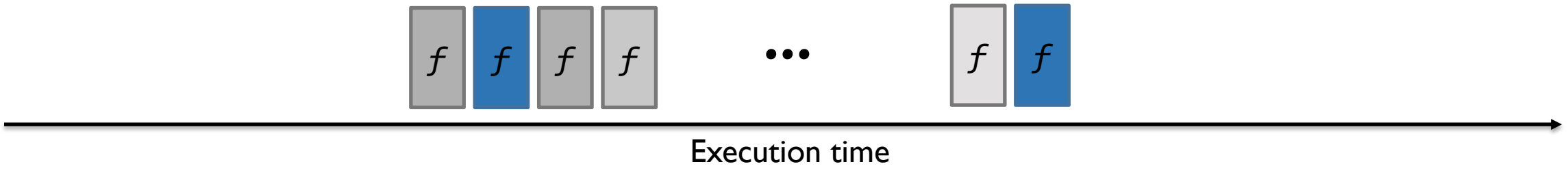
→ **Large instruction** footprints cannot be maintained on-chip under heavy **interleaving**

Addressing Cold On-chip Instruction State

Basic Idea:

- **Exploit high commonality** of function invocations
  - Prefetch common instruction state
- **Record instruction** working set of one invocation
- **Restore** the instruction working with the next invocation

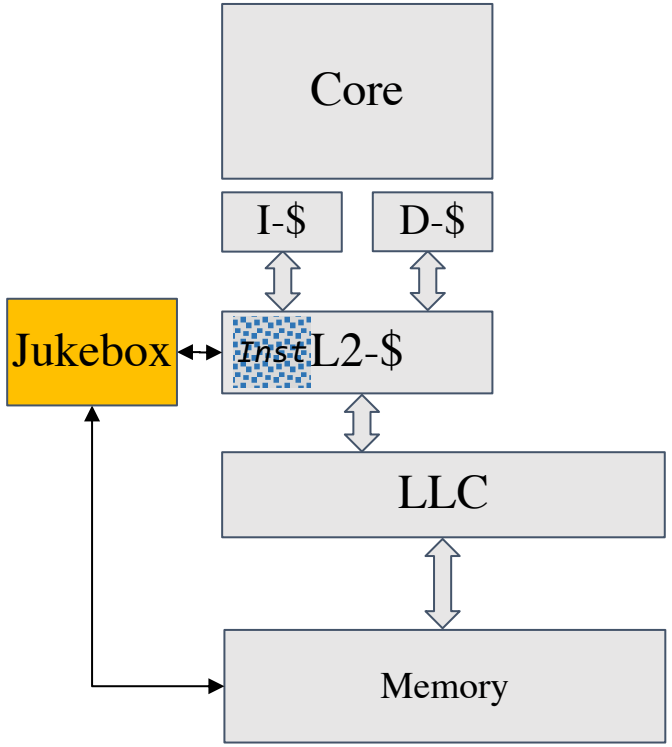




Jukebox: I-Prefetcher for Serverless

Jukebox: record-and-replay instruction prefetcher for lukewarm serverless function invocations

- **Record: L2 misses** using a spatio-temporal encoding
  - Stores records in main memory
- **Replay**: prefetch the recorded addresses **into the L2**
- Fully decoupled from the core
  - Triggered by function invocation
- Operates on virtual addresses
  - Not affected by page reallocation
  - Prefetching prepopulates TLB



Jukebox records and replays L2 instruction working sets

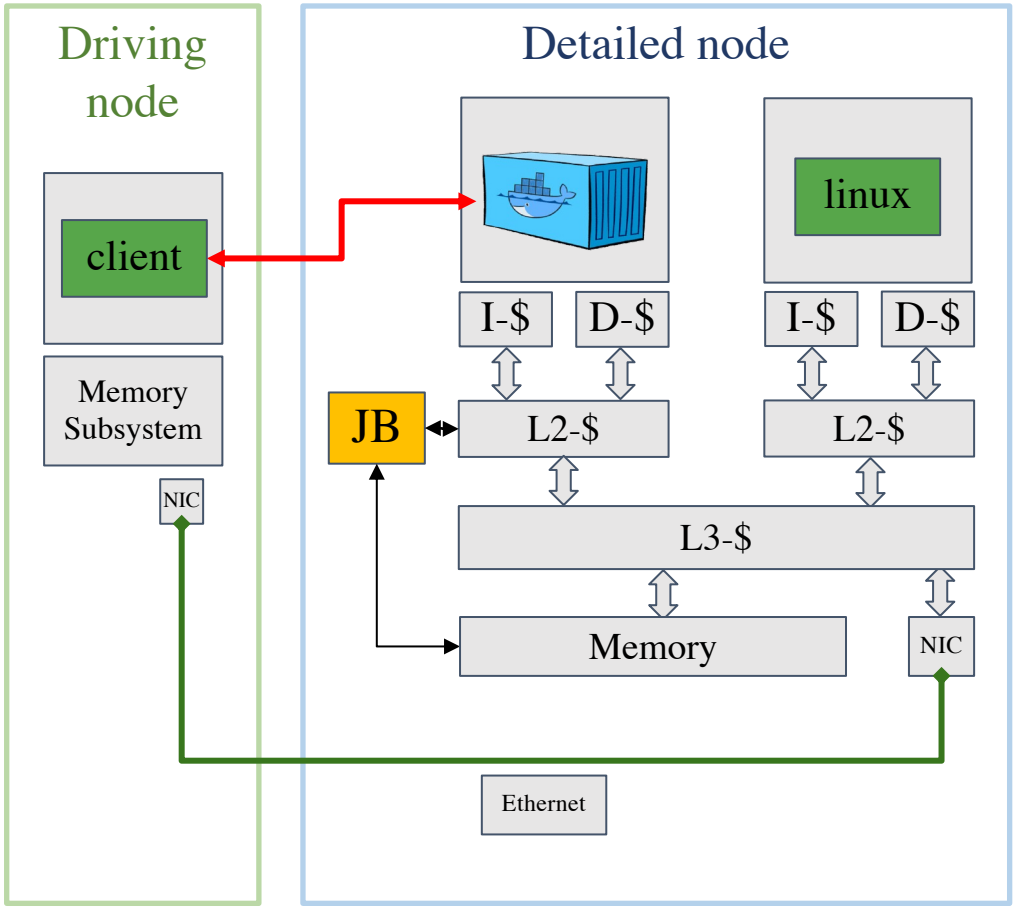
Evaluation Infrastructure

Use **gem5** simulator for evaluating Jukebox

- Detailed model of the server node
  - *Dual core Skylake-like CPU model*
  - *32KB L1-I/D, 1MB L2/core, 8MB L3*
- Secondary node for driving invocations.
- Functions run in isolation
- **Cycle accurate** simulation of the **full system**
  - Exact same software stack as on real hardware

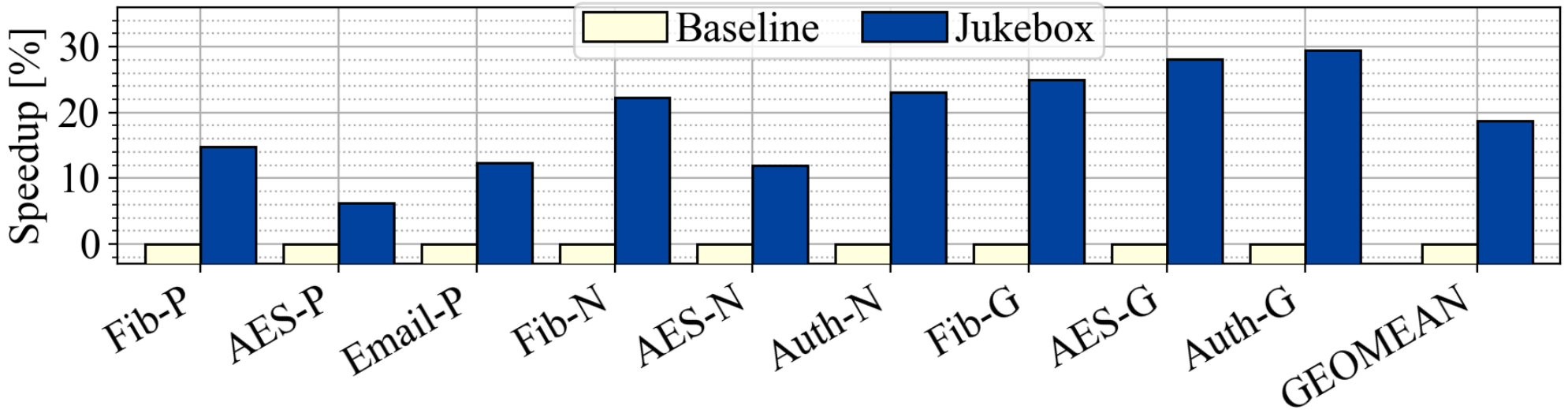
  (Ubuntu 20.04, kernel: 5.4, same container images)
  - **First support for containers in gem5**
    - Publicly available:

      https://github.com/ease-lab/vSwarm-u



Representative infrastructure for detailed evaluation

Jukebox: Performance Improvements



Jukebox's recording and replaying of instruction working sets:

- Improves performance by 18%, on average
  - Consistent improvement across benchmarks
- Covers >85% of off-chip instruction misses
- Requires only 32KB of metadata per function instance

Jukebox is simple & effective

Summary

Serverless functions present new challenges for modern CPUs

→ Need a **representative infrastructure** to study serverless stacks: **vHive**

→ **Lukewarm execution:** function in memory, but no μ-arch state on-chip

Characterisation reveals a severe front-end bottleneck in lukewarm executions

→ Large instruction footprints cannot be maintained on-chip under heavy function interleaving

→ Frequent off-chip misses for instructions expose the CPU to long-latency stalls

Jukebox: Record-and-replay instruction prefetcher for lukewarm serverless functions

→ Simple and effective solution for cold on-chip instruction state

→ Improves performance by 18% with 16KB of in-memory metadata per instance

Acknowledgements

Students & interns

Dmitrii Ustiugov (now @ETH)
David Schall
Artemiy Margaritov
Shyam Jesalpura
Theodor Amariucai
Harshit Garg
Plamen Petrov
Michal Baczun
Yuchen Niu
Amory Hoste
Bora M. Alper

External collaborators

Rustem Feyzhanov (Instrumental)
Francisco Romero (Stanford)
Marios Kogias (Imperial)
Edouard Bugnion (EPFL)
Ana Klimovic (ETH)

Industry supporters

Join our Serverless Research Community

https://github.com/ease-lab

Thank you!

Questions?