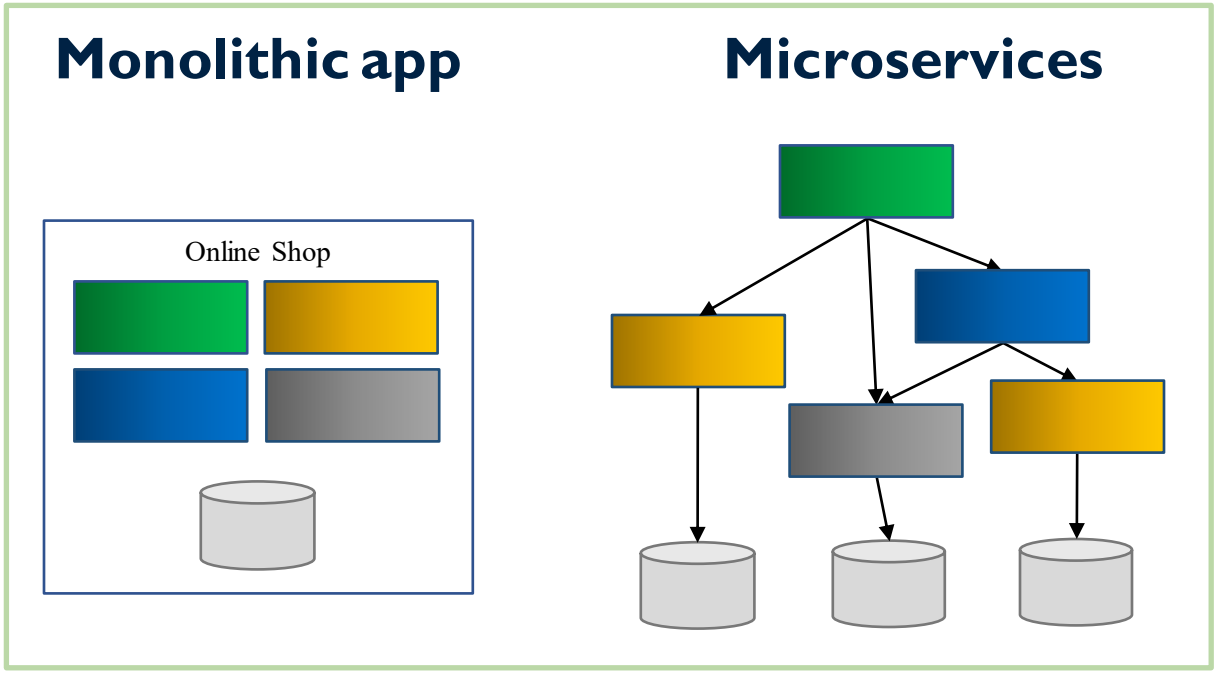
Lukewarm Serverless Functions: Characterization and Optimization

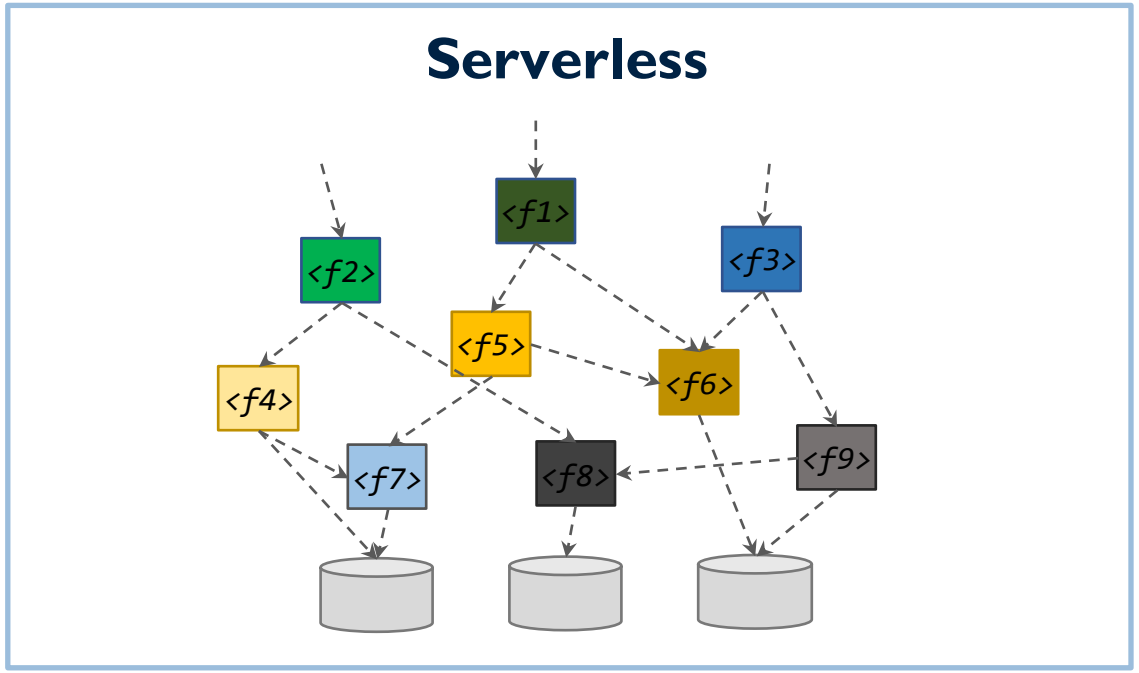
David Schall,

Artemiy Margaritov, Dmitrii Ustiugov,

Andreas Sandberg, Boris Grot

Cloud Applications: from Monoliths to Serverless





Conventional cloud deployments:

- Virtual machines that stay up for long periods of time
- User is billed even when the service is idle

Serverless cloud deployments:

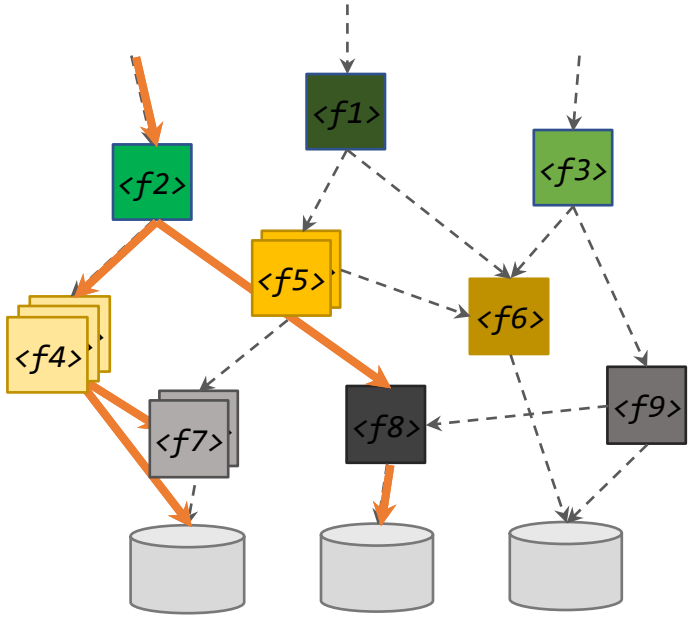
- Functions are invoked on-demand
- No invocations → no cost 🙂
- > 50% of cloud customers use serverless [Datadog 2022]

Serverless is big … and growing!

Serverless Basics

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 **scaling down to zero**
  - Zero is not possible for monoliths & microservices
- 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 ☺



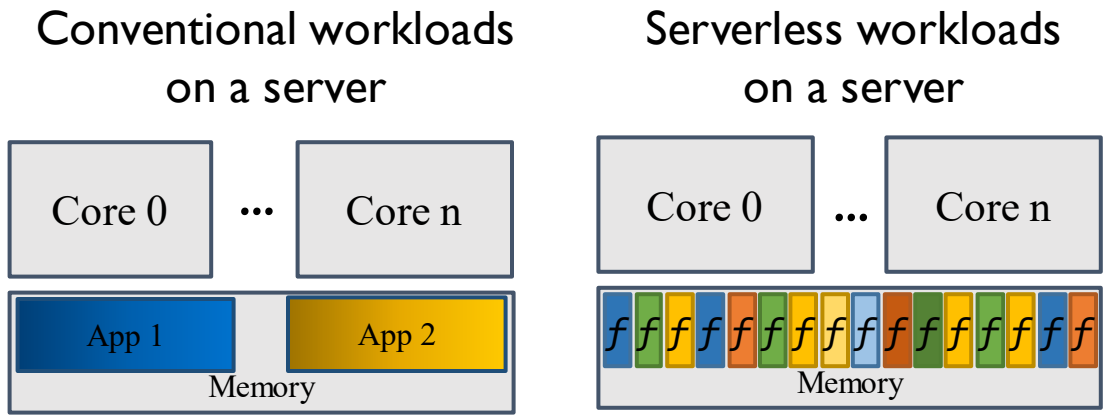
Serverless Characteristics on a Server

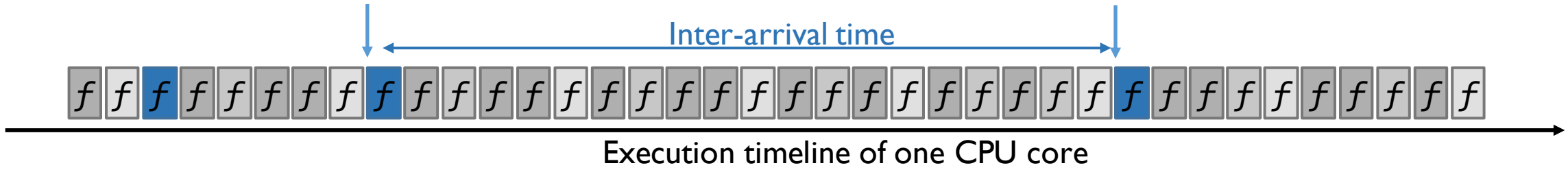
Unique characteristics:

- Short function execution times: a few ms or less
  - Contrast: Linux scheduling quantum: 10-20ms
- Small memory footprint: tens of MB
- Sporadically invoked (seconds or minutes)
  [Microsoft Azure @ATC20]

Implication:

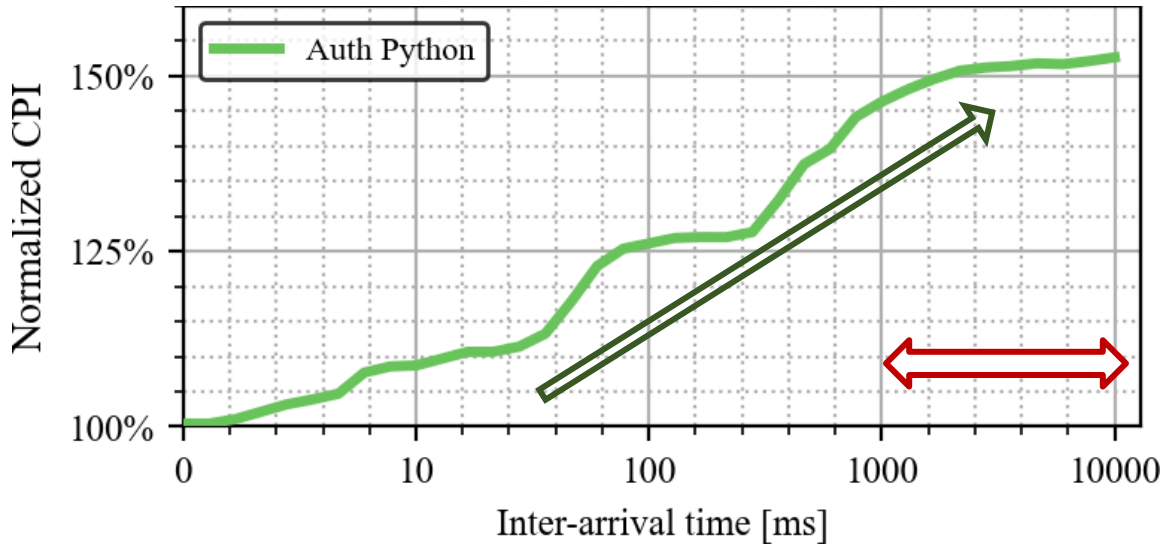
- Extreme multi-tenancy: Thousands of functions resident on a server
- **Huge degree of interleaving** between two invocations of the same function





What are the implications for microarchitecture?

Effect of Interleaving



Typical IAT

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

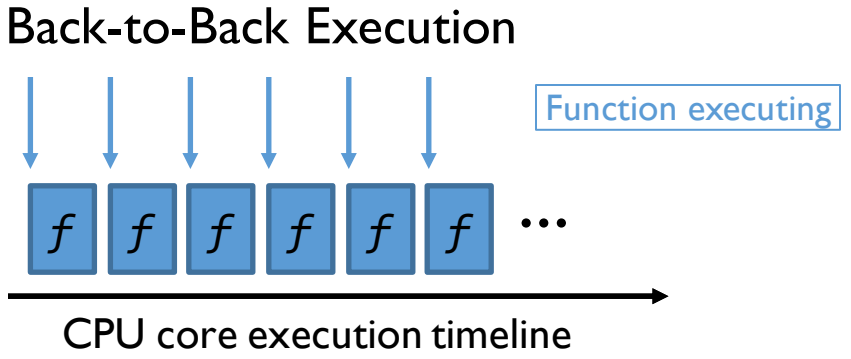
What causes the slowdown?

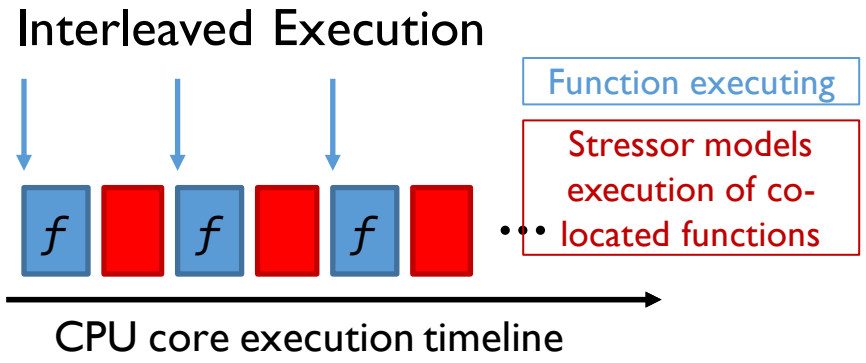
Characterization Methodology

Compare back-to-back to interleaved execution of one function
- Function-under-test runs isolated
- Interleaving modelled by stressor
  - Same effect as interleaved execution of co-located functions

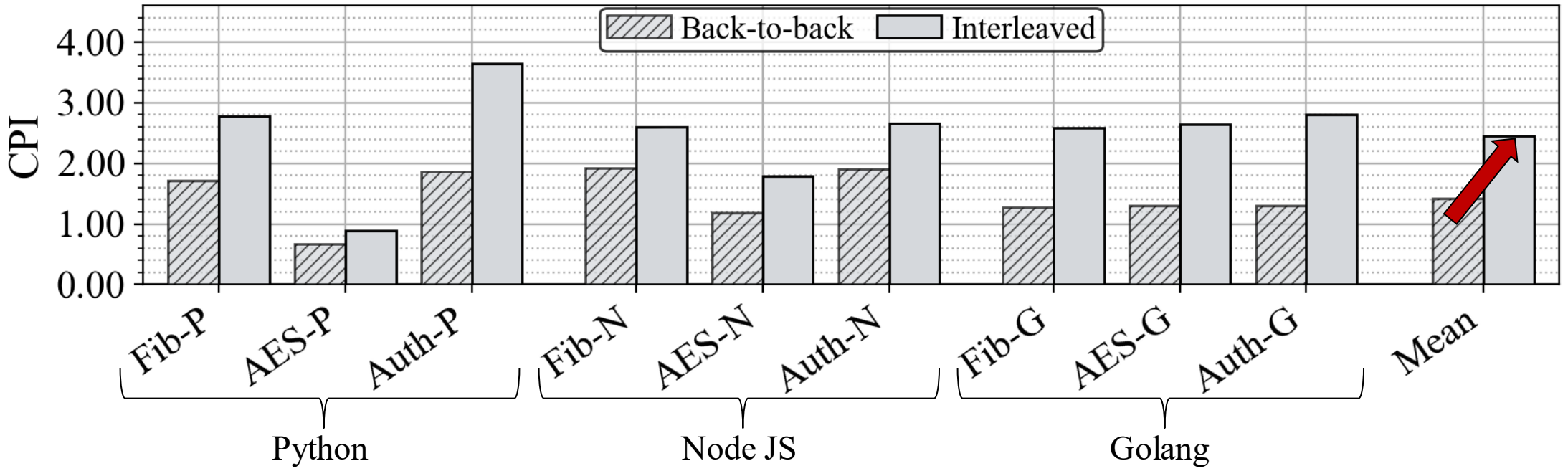
Use **Top-Down Methodology** for analysis
- Machine: Intel Broadwell CPU
  (10 cores, SMT disabled, 32KB L1-I/D, 256KB L2/core, 25MB LLC)
- Collect CPU performance counters

Workloads: 20 serverless functions
- Large variety in functionality and runtimes
  - 14 function types in three languages
  - Including compiled, JIT-ed and interpreted languages
- Publicly available https://github.com/ease-lab/vSwarm



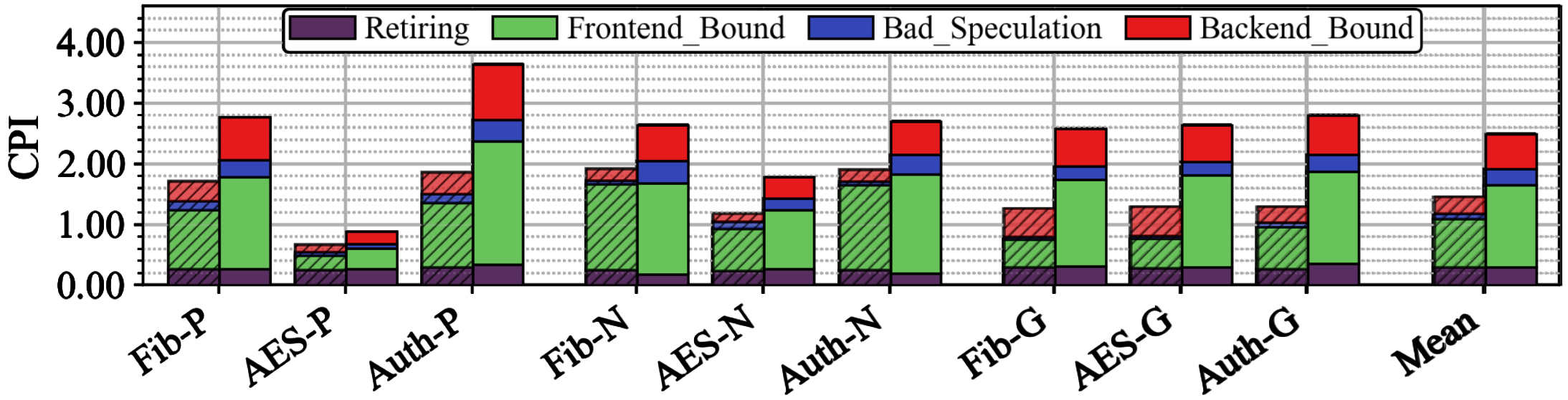


Performance Implications of Interleaving



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

Top-Down CPI Analysis

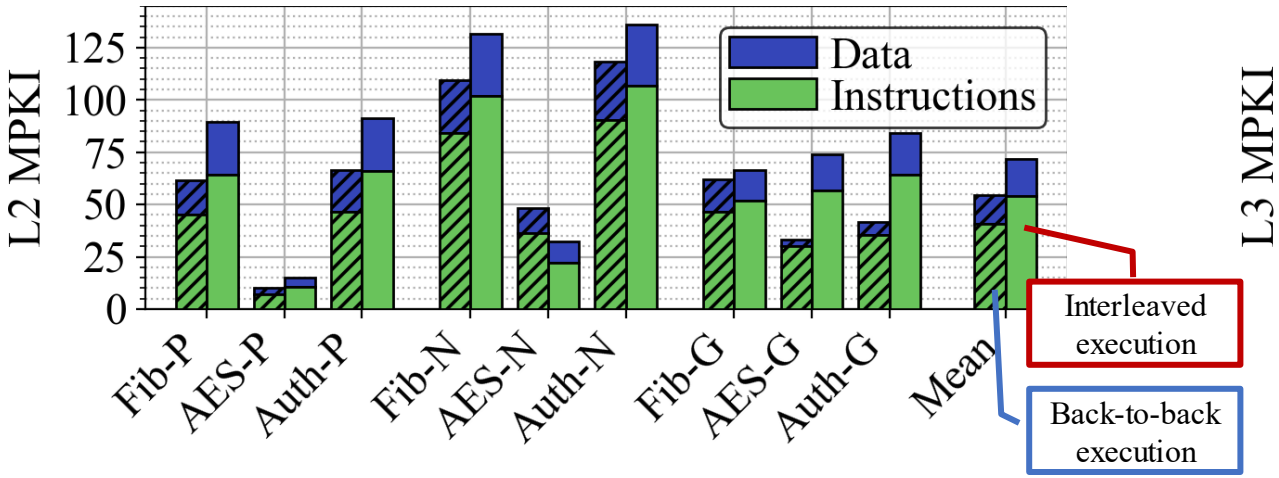


- Front-end stalls is the largest source of stalls
- 56% of additional stall cycles in interleaved execution come from **fetch latency**

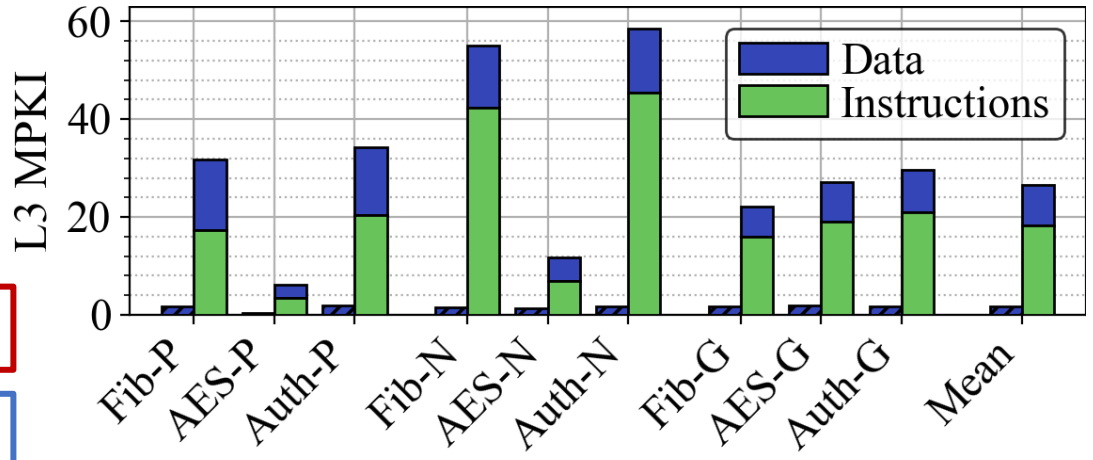
Instruction delivery a critical performance bottleneck for warm invocations

Instruction Fetch Pain Points

L2 Cache *(256KB/core)*



L3 Cache *(25MB)*



- Serverless workloads frequently miss in L2 cache
  - 50+ MPKI, on average
- Misses for instructions dominate
- Similar behaviour for both back-to-back and interleaved

- 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

Understand 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 of the same function
2. **Large instruction footprint**: 300KB-800KB
   - Contrast: L2 cache size: 256 KB
   - Deep software stacks result in large amount of code

Takeaways:

- **Large instruction** footprints → cannot be maintained on-chip under heavy **interleaving** ☹
- Same instructions accessed across invocations ☺

Can we exploit the high commonality to improve performance?

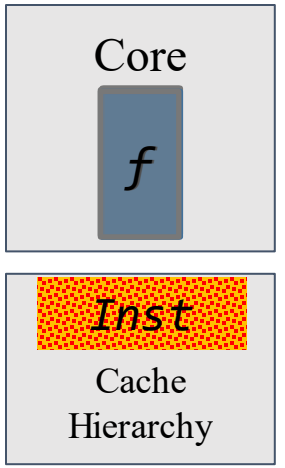
Addressing Cold On-chip Instruction State

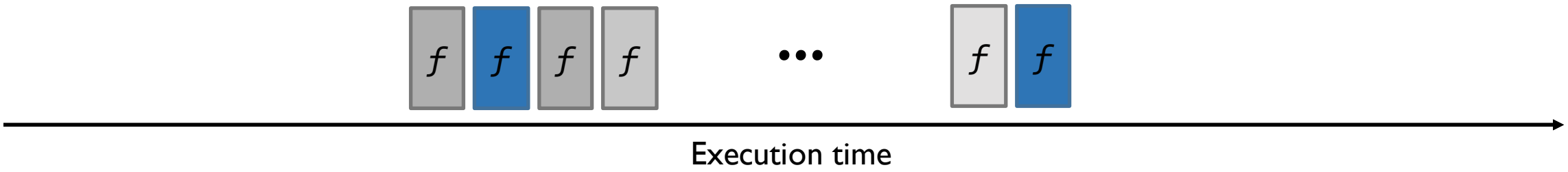
Basic Idea:

- **Exploit high commonality** of function invocations
  - Suggest prefetch opportunities

Mechanism:

- **Record instruction** working set of one invocation
- **Restore** the instruction working with the next invocation

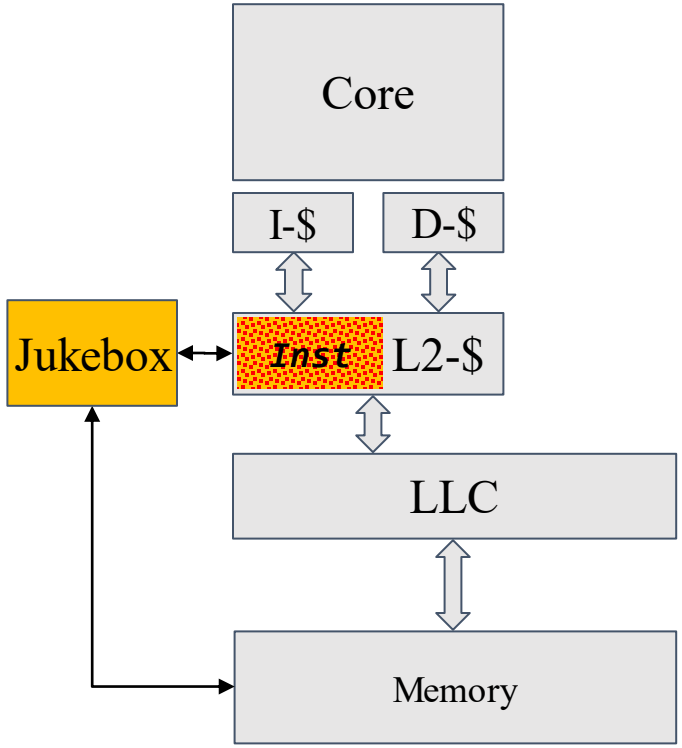




Jukebox: I-Prefetcher for Serverless

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 re-allocation
  - Prefetching prepopulates TLB

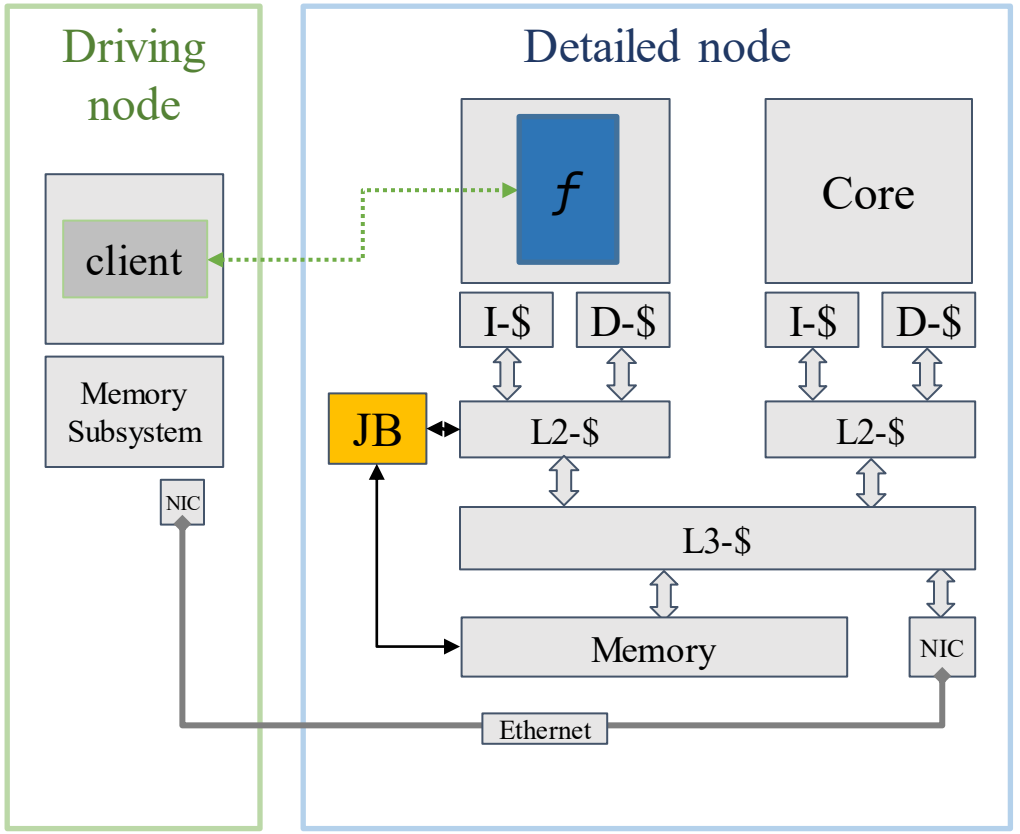


Jukebox records and replays L2 instruction working sets

Evaluation Infrastructure

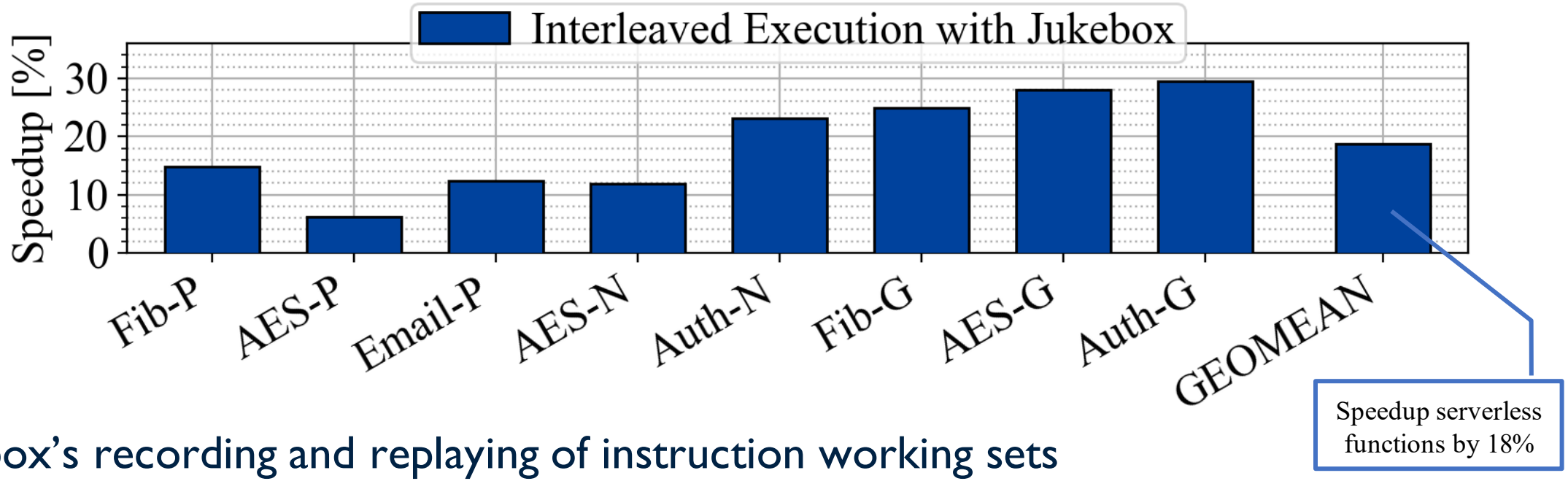
Use **gem5** simulator for evaluating Jukebox

- Detailed 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, same container images, full gRPC stack)
  - First support for containers in gem5
    - Publicly available:
      https://github.com/ease-lab/vSwarm-u



Representative infrastructure for studying serverless functions

Jukebox: Performance Improvements



Jukebox's recording and replaying of instruction working sets

- Speedup interleaved (lukewarm) execution by 18%, on average
  - Consistent for all benchmarks
- Covers > 85% L3 instruction misses
  - Effective in covering off-chip instruction misses
- Only 32KB metadata

Jukebox's idea is simple but very effective

Summary

Serverless functions present new challenges for modern CPUs

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

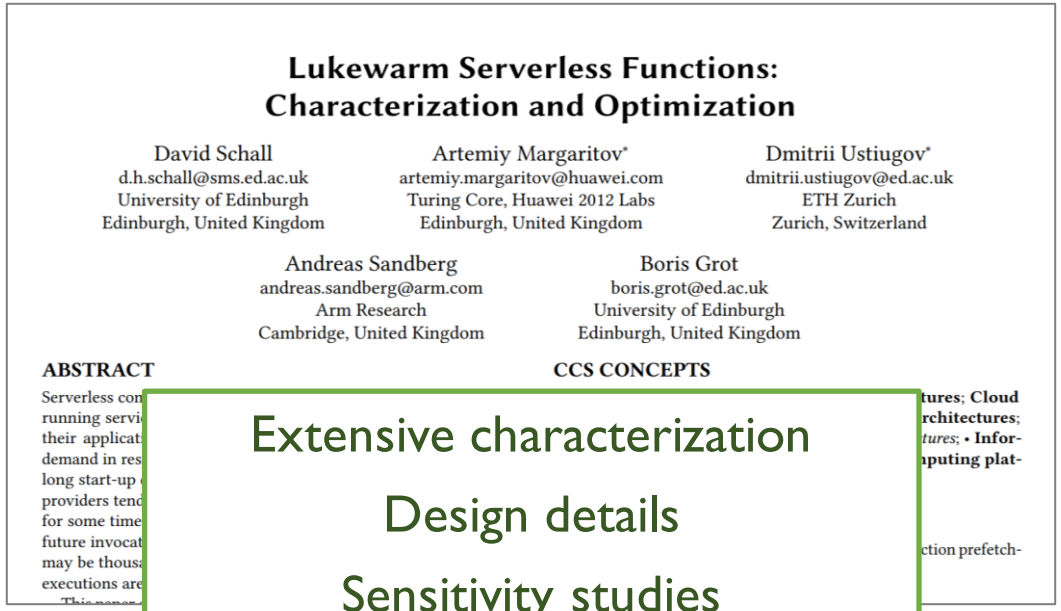
Characterisation reveals a severe front-end bottleneck in lukewarm executions

→ 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 32KB of in-memory metadata per instance

Thank you!



Lukewarm Serverless Functions: Characterization and Optimization

David Schall
d.h.schall@sms.ed.ac.uk
University of Edinburgh
Edinburgh, United Kingdom

Artemiy Margaritov*
artemiy.margaritov@huawei.com
Turing Core, Huawei 2012 Labs
Edinburgh, United Kingdom

Dmitrii Ustiugov*
dmitrii.ustiugov@ed.ac.uk
ETH Zurich
Zurich, Switzerland

Andreas Sandberg
andreas.sandberg@arm.com
Arm Research
Cambridge, United Kingdom

Boris Grot
boris.grot@ed.ac.uk
University of Edinburgh
Edinburgh, United Kingdom

ABSTRACT

Serverless con... running servi... their applicat... demand in res... long start-up ... providers tend... for some time... future invocat... may be thous... executions are...

CCS CONCEPTS

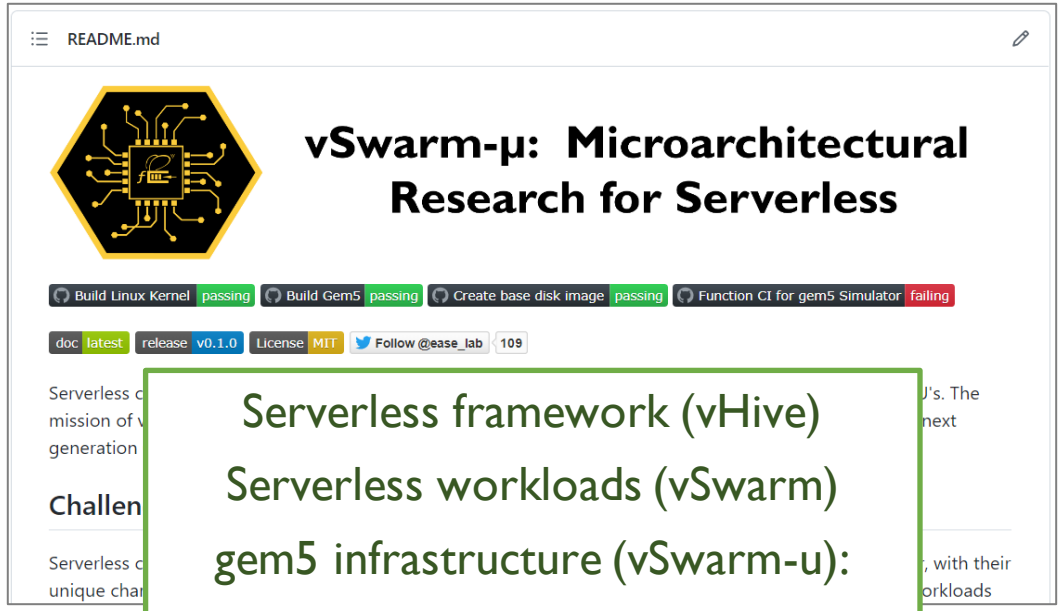
...tures; Cloud ...rchitectures; ...tures; • Infor... ...puting plat...

...ction prefetch-

Extensive characterization

Design details

Sensitivity studies



README.md

vSwarm-μ: Microarchitectural Research for Serverless

Build Linux Kernel passing | Build Gem5 passing | Create base disk image passing | Function CI for gem5 Simulator failing

doc latest | release v0.1.0 | License MIT | Follow @ease_lab 109

Serverless c... mission of v... generation ...

...U's. The ...next

Challen...

Serverless c... unique char...

..., with their ...orkloads

Serverless framework (vHive)

Serverless workloads (vSwarm)

gem5 infrastructure (vSwarm-u):

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