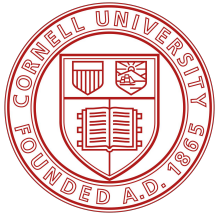
PRONGHORN

Effective Checkpoint Orchestration for Serverless Hot-Starts

Sumer Kohli[1*], Shreyas Kharbanda[2*], Rodrigo Bruno[3], Joao Carreira[4], Pedro Fonseca[5]

1 

2 

3 

4 

5 

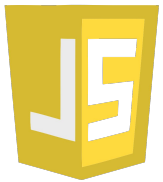
Serverless Computing

AWS Lambda

Azure Functions

Google Cloud Functions

JavaScript



Java

Stateless functions

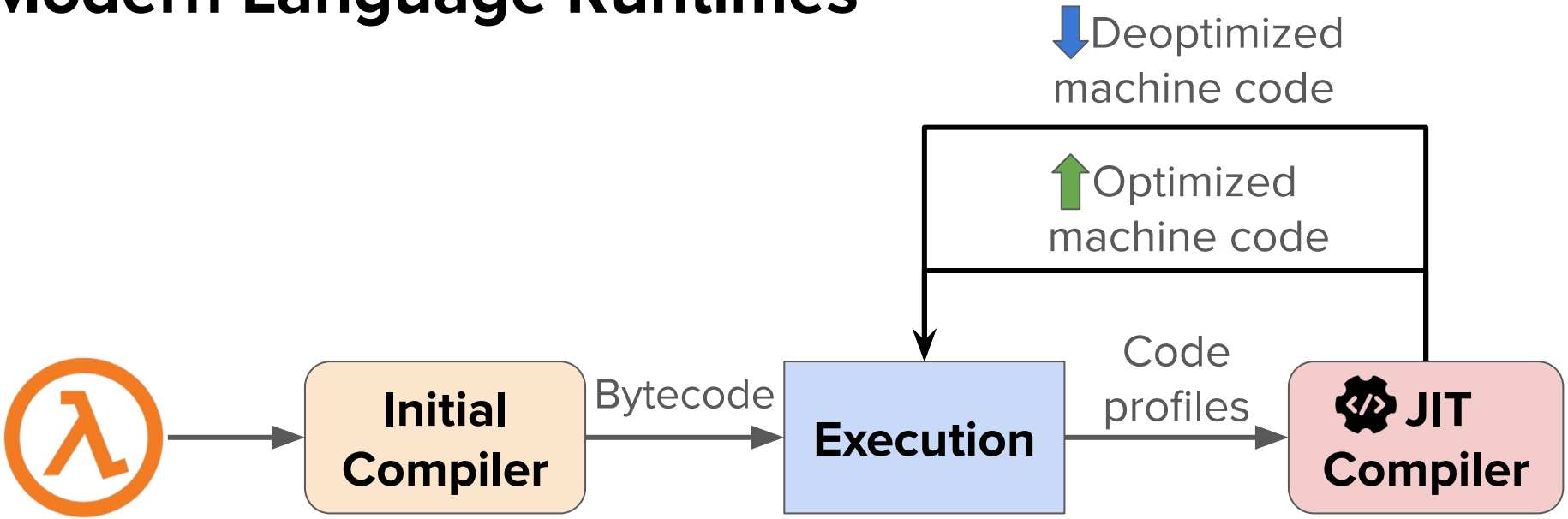
Sandboxed environments

Automatic infrastructure management

High-level languages

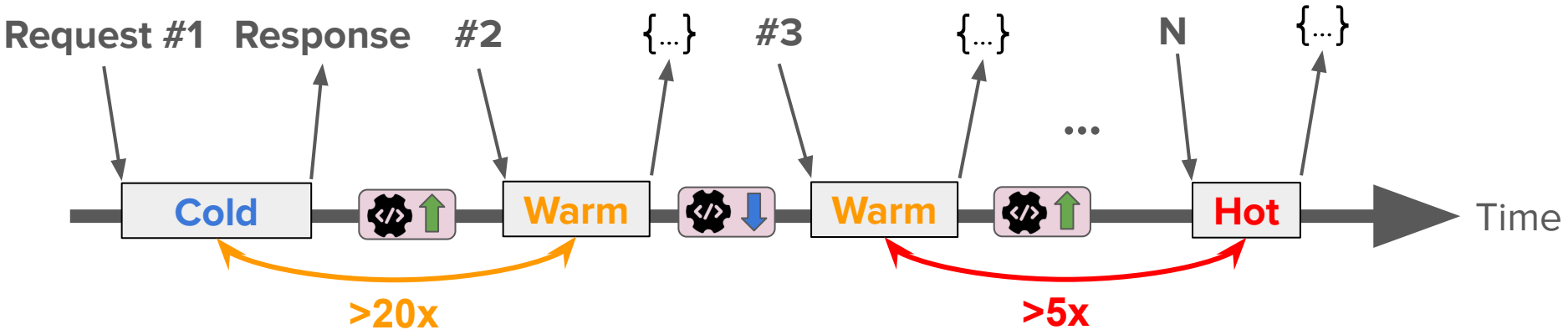
Complex runtimes

Modern Language Runtimes



JIT compilation is critical for performance.

JIT Optimization Speed



JITs can take hundreds of requests to fully optimize code.

JIT Optimization

JIT Deoptimization

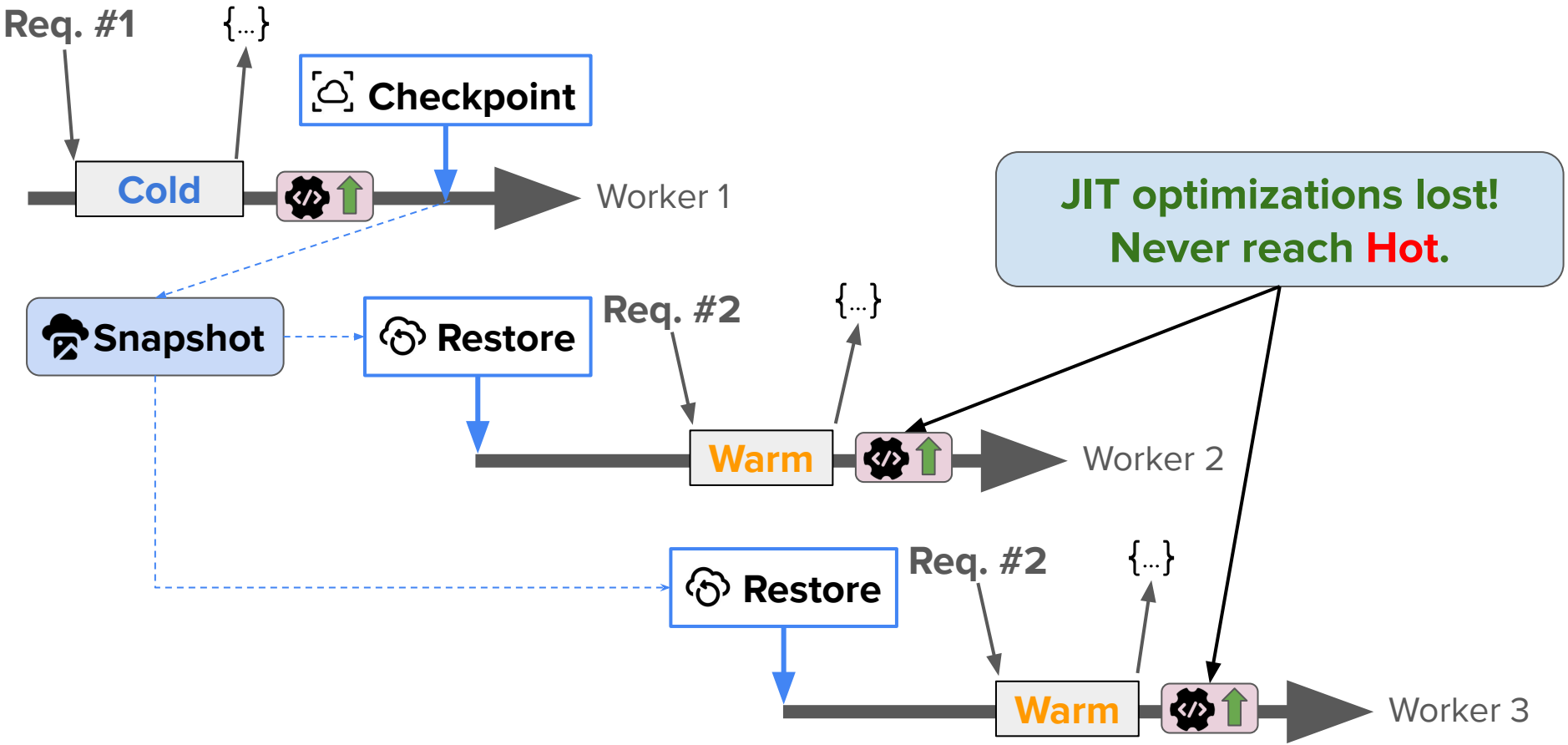
Waste of JIT Runtime Optimizations

Short-lived functions → Gathering profiles is difficult

Frequently-evicted workers → Most runtimes are used once

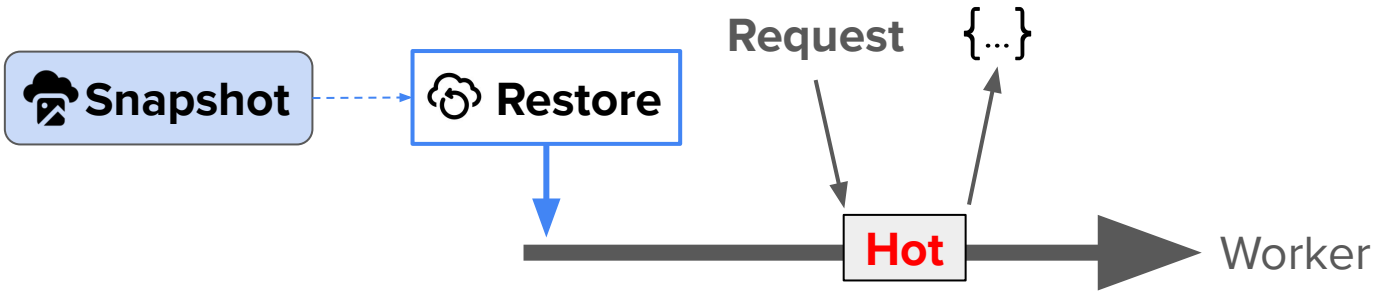
Fresh runtimes → Previous optimizations are lost

State of the Art: Checkpoint & Restore

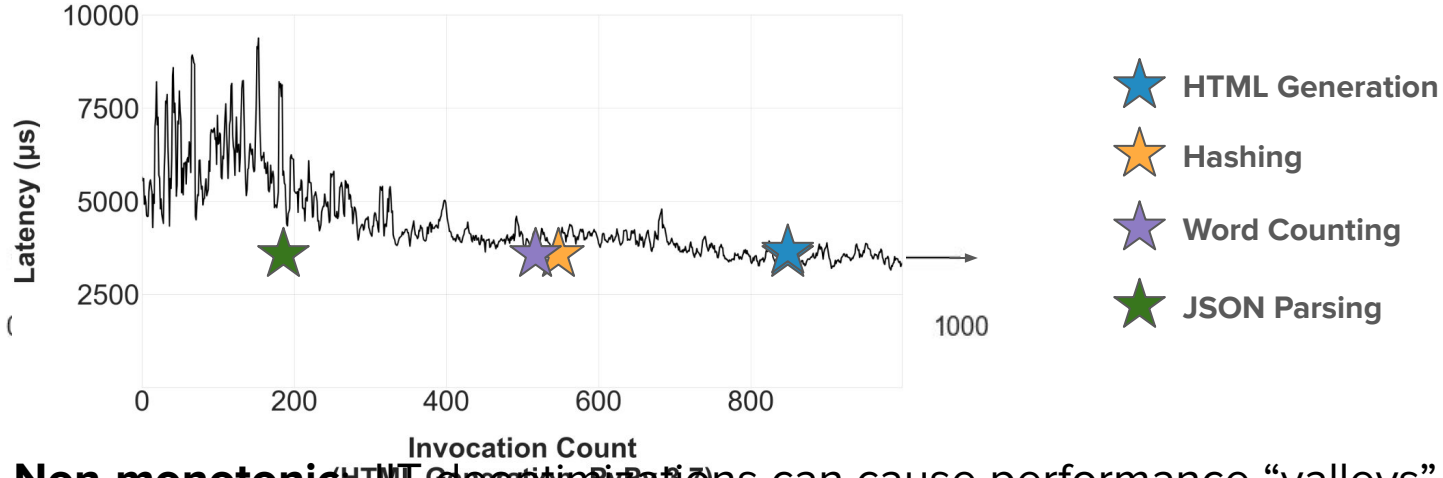


Pronghorn's Goal

Workers should start with fully-optimized code.



Challenge: When should we checkpoint?



Non-monotonic: JIT deoptimizations can cause performance “valleys”.

Non-bounded: JIT runtimes can take any number of invocations to fully optimize a function.

Non-deterministic: JIT runtimes can optimize the same code differently.

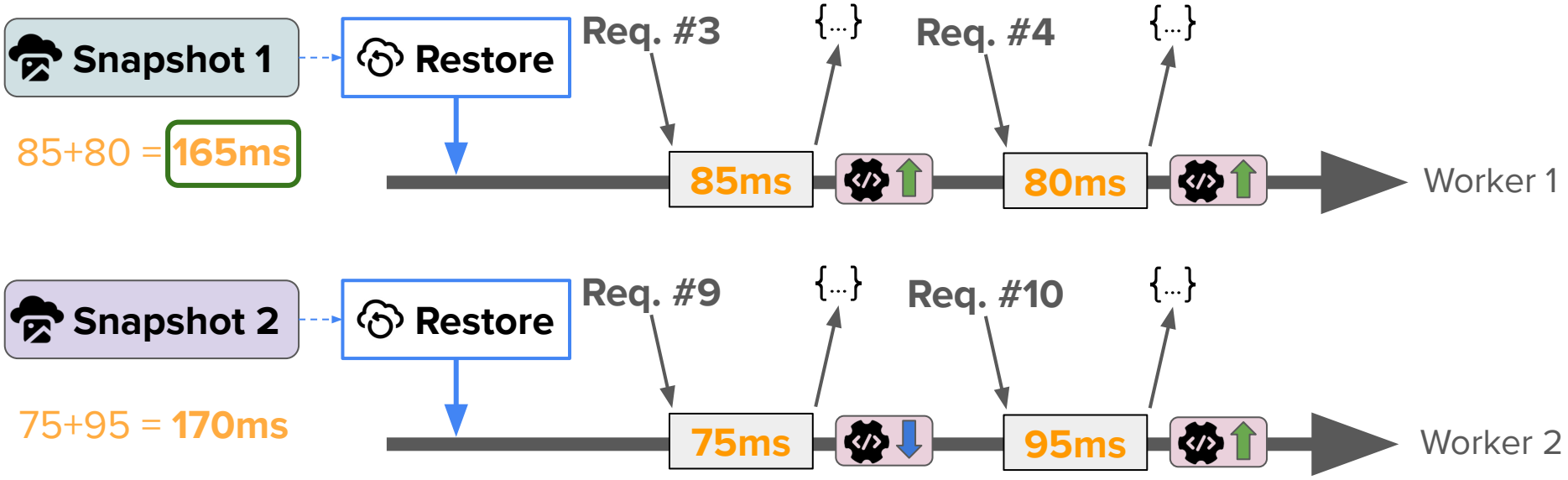
We **do not know** ahead of time when is the best time to checkpoint a function.

→

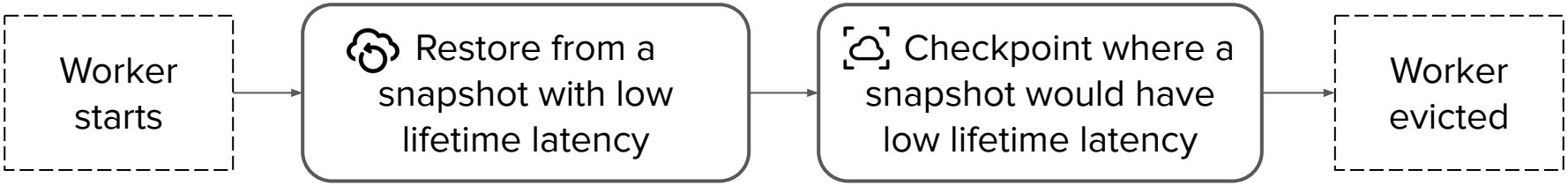
We need a **dynamic checkpointing policy**.

Pronghorn: Key Idea

We can grade snapshots by their **total lifetime latency**.



Pronghorn


Worker starts
Restore from a snapshot with low lifetime latency
Checkpoint where a snapshot would have low lifetime latency
Worker evicted

Pronghorn: Computing lifetime latency

Given:

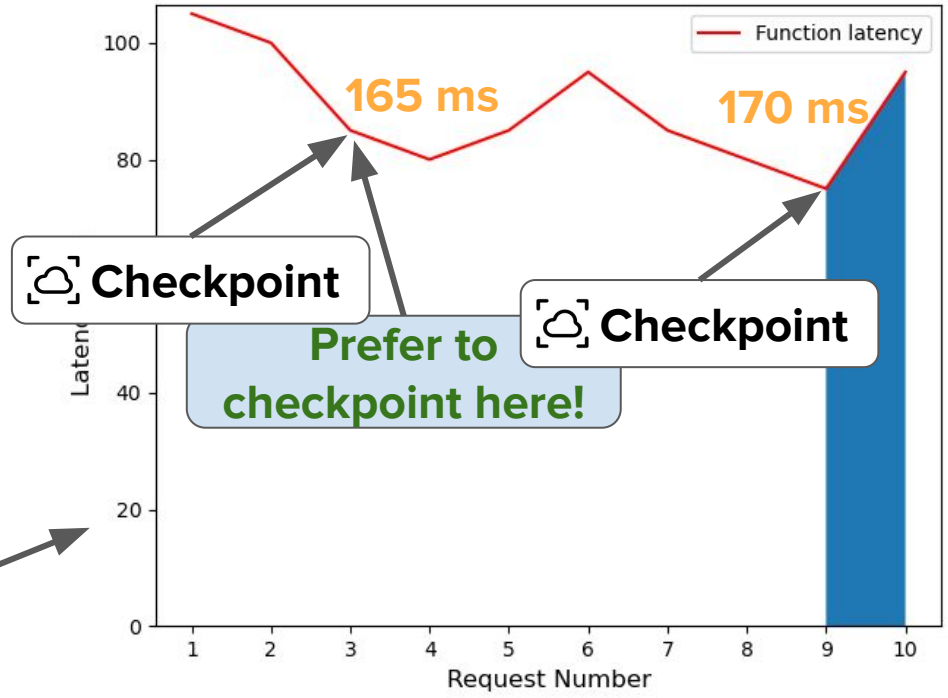
1. Worker lifetime
2. **Latency for each request #**

Known

Let worker lifetime be **2** requests

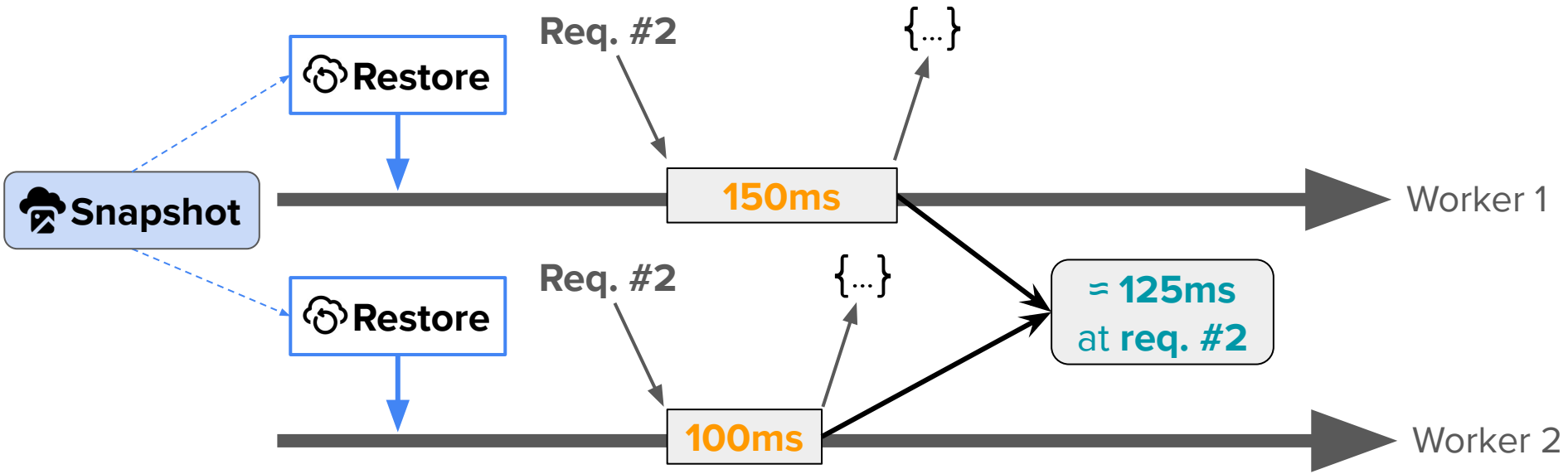
How do we get this graph?

Example latency curve

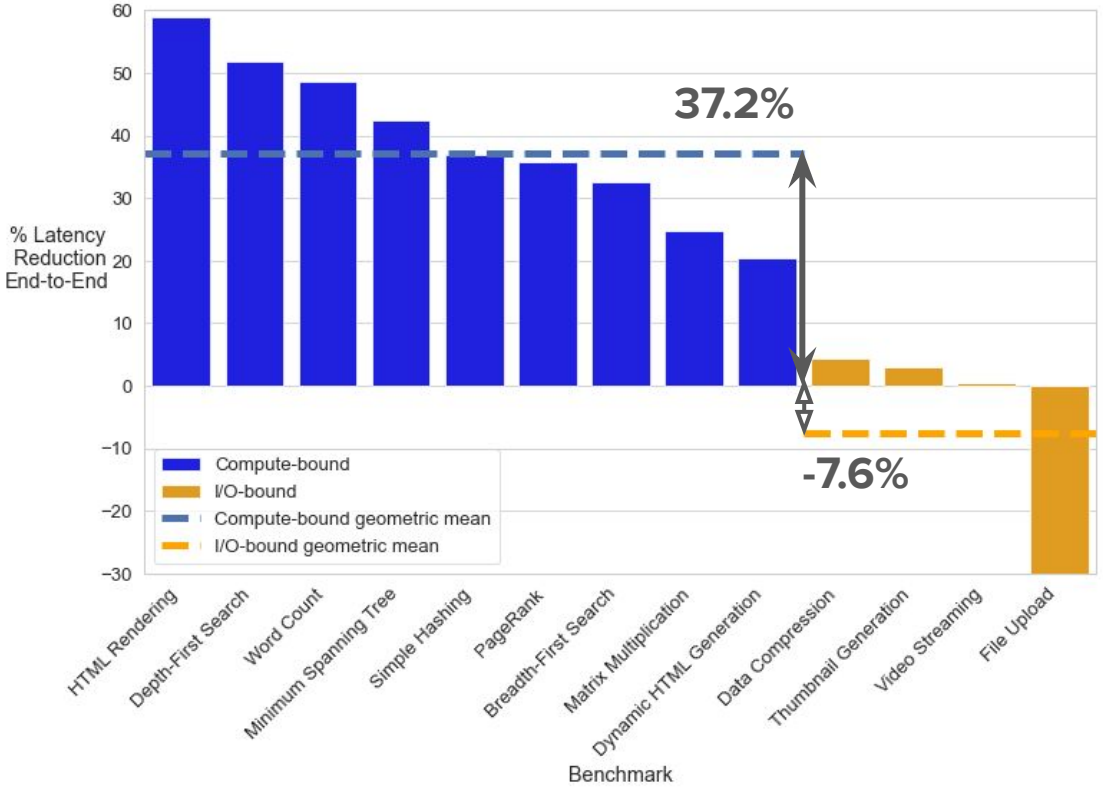


Pronghorn: Estimating function latency

We can **profile requests** to estimate **latency at each request number.**



End-to-end Benchmark Results



Integrated into a **widely-used serverless platform** (OpenFaaS)

Improves compute-bound workloads by **37.2% on average** vs. existing systems

Generally **matches I/O-bound function performance** of existing systems

Benchmarks adapted from ServerlessBench, FaaSDom, and SeBS.

Cost Analysis & Mitigation

Checkpoint

60–105 ms
worker downtime

→ Off critical path
Controlled by cloud provider

Storage

Size of
snapshot pool

→ Configurable pool size

Network

2x bandwidth
usage

→ Reverts to SOTA after
checkpointing phase

Pronghorn's costs are either off the critical path or configurable.

Key Takeaways

1. We must carefully choose **when** to checkpoint functions for hot-starts.
2. Our **dynamic checkpointing policy** reduces latency by 37%.
3. Compatible with **any** JIT runtime or serverless framework.

Any questions?

PRONGHORN: https://github.com/rssys/pronghorn-artifact

Contact me at sumer@cs.stanford.edu

(Slide images courtesy of project webpages and flaticon.com)