Serverless Computing Basics

Yiying Zhang

Outline

- Overview and major offerings
- Example applications
- Limitations and discussion
- Serverless computing in the wild

- Project progress report due this Thur 5/9
- Quiz 2 next Monday 5/13

Acknowledgement: Some of the slides are from Ali Ghodsi and Ion Stoica's Berkeley lecture notes in 2018, Tyler Harter's HotCloud'16 OpenLambda talk, and Mohammad Shahrad's ATC'20 talk

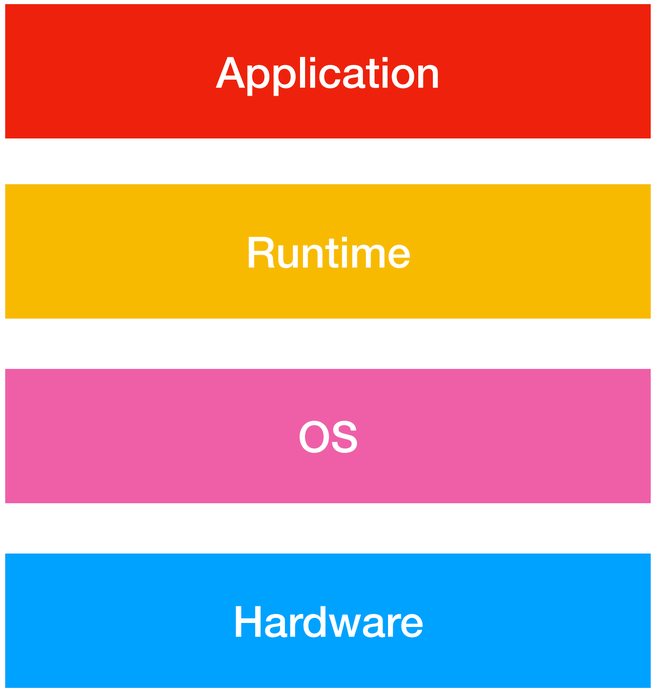
What is Serverless Computing?

- Computing without servers?
- Running applications without the need to manage servers?
- Running functions instead of containers/VMs?
- Infinite scaling?

- The truth: no clear, agreed definition, i.e., no one really knows

One Perspective: How Cloud and Virtualization Evolved

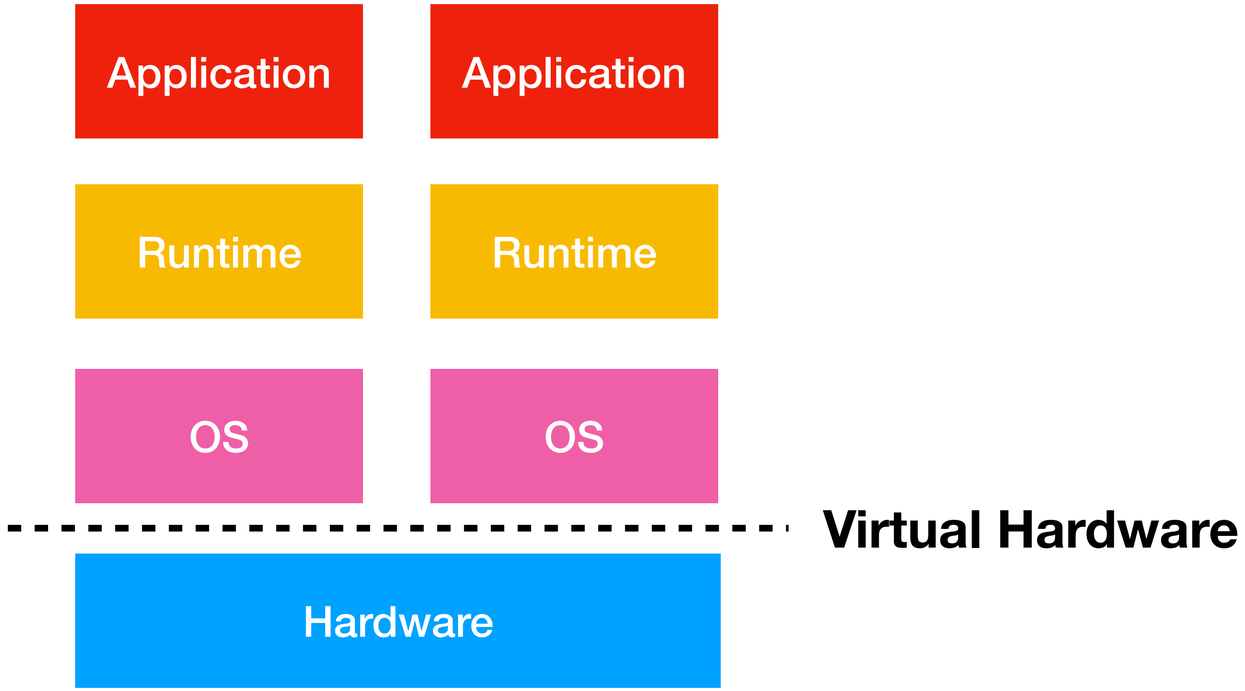
Classic Application Execution



Weak Virtualization

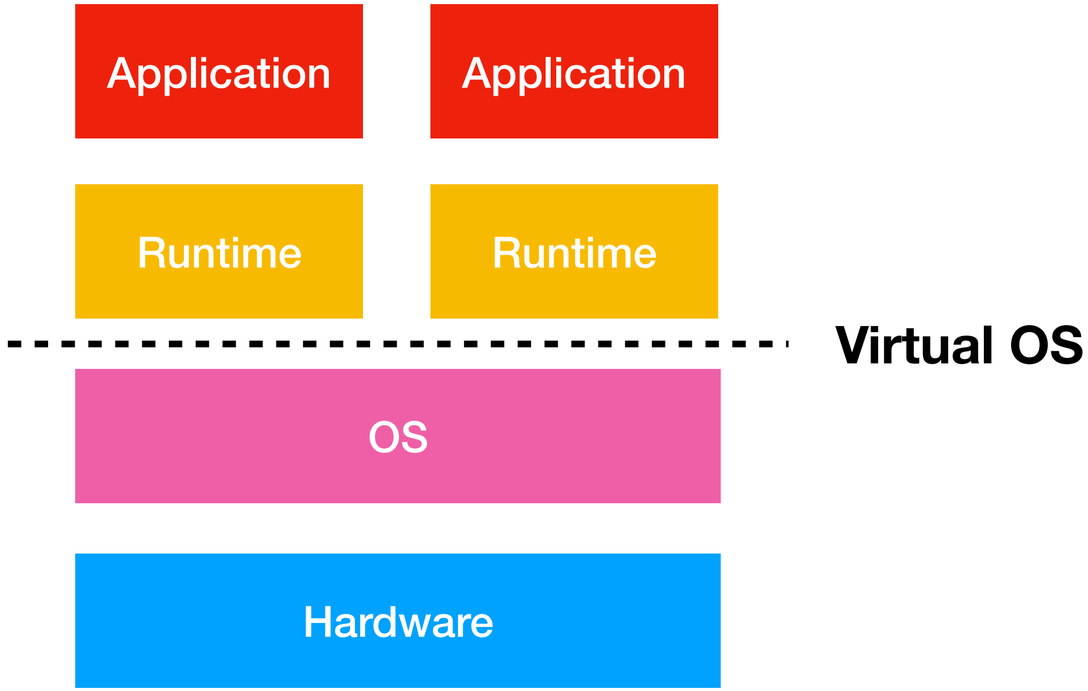
No Sharing

1st Generation: Virtual Machines



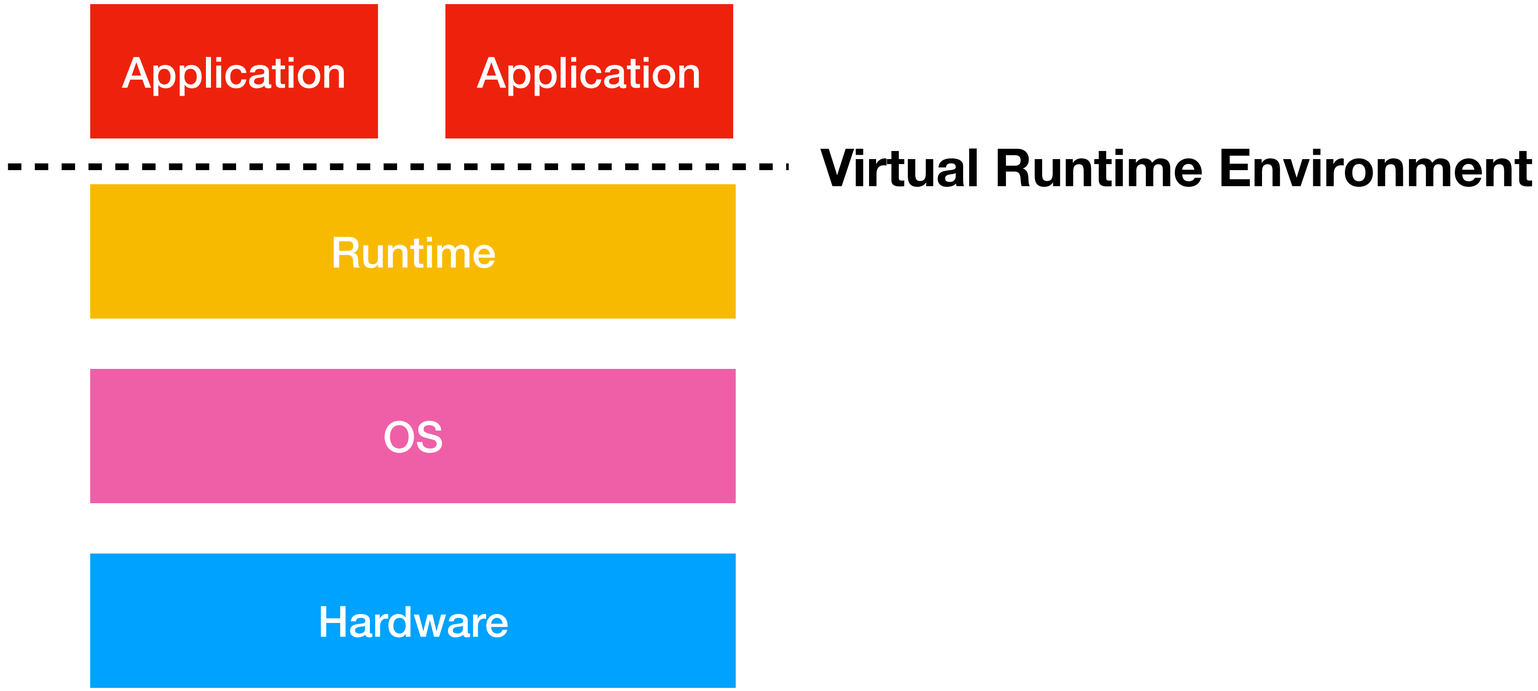
Shared Physical Hardware

2nd Generation: Containers



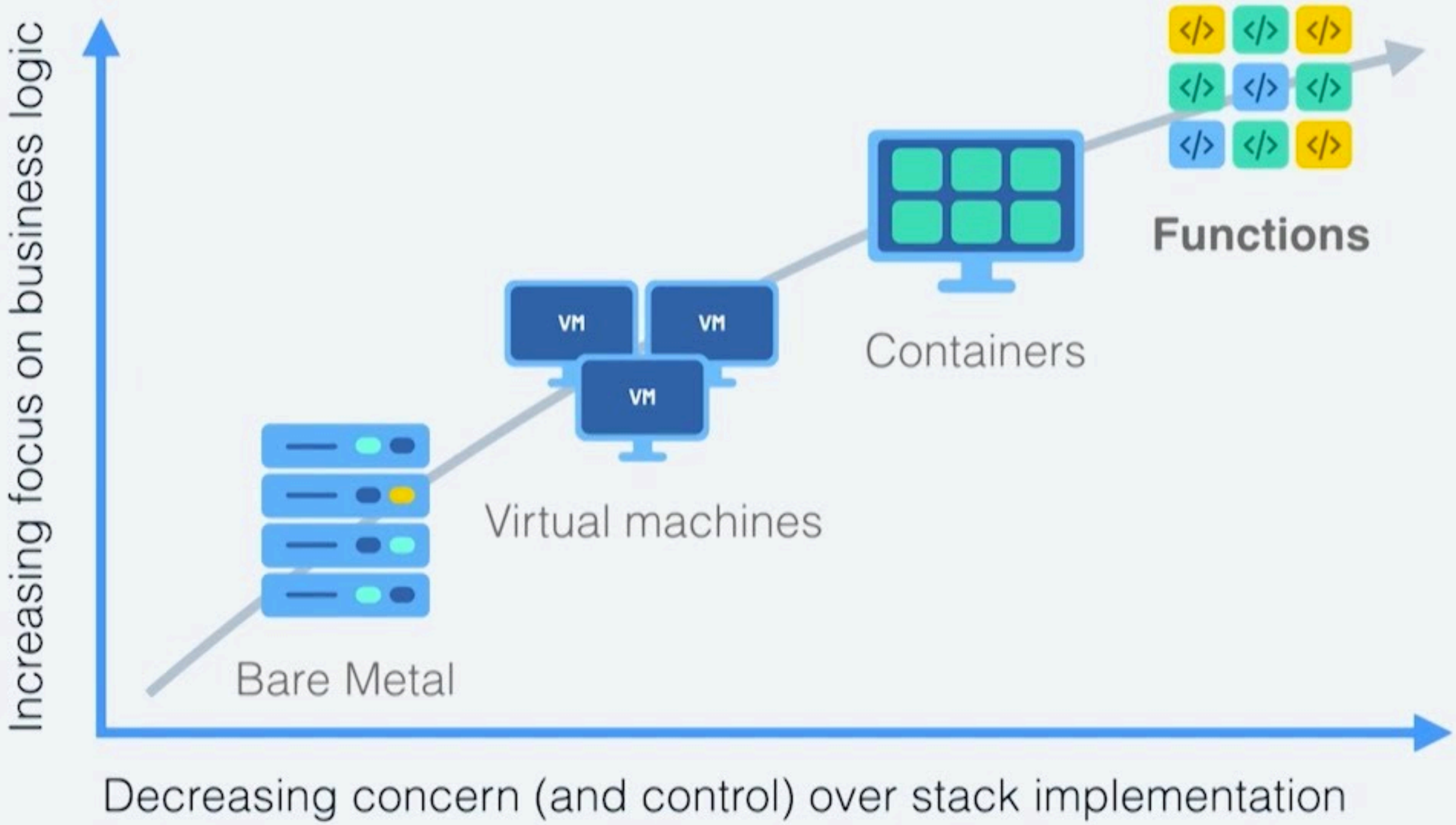
Shared Physical Hardware and OS

3rd Generation: Serverless Computing



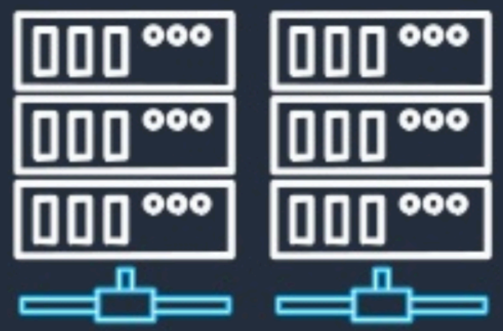
Shared Physical Hardware, OS, and Runtime

Evolution of serverless



© 2017 IBM Corporation | Interconnect 2017

Serverless means …



No server or container management

Flexible scaling

High availability

No idle capacity

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What is the essence of “Serverless Computing”? Or, what do people really like about it?

- Management-free
  - No need to handle creation, failure, replication, etc.
- Autoscaling
  - Spin up/down functions quickly based on load
- Only pay for what you use
  - Event triggered

“ I didn’t have to worry about building a platform and the concept of a server, capacity planning and all that “yak shaving” was far from my mind... However, these changes are not really the exciting parts. The killer, the gotcha is the billing by the function...

This is like manna from heaven for someone trying to build a business. Certainly I have the investment in developing the code but with application being a variable operational cost then I can make a money printing machine which grows with users…"

source: https://hackernoon.com/why-the-fuss-about-serverless-4370b1596da0

What is Today’s Serverless Computing Like?

- Largely offered as Function as a Service (FaaS)
  - Cloud users write functions and ship them
  - Cloud provider runs and manages them
- Still runs on servers
- Have attractive features but also many limitations

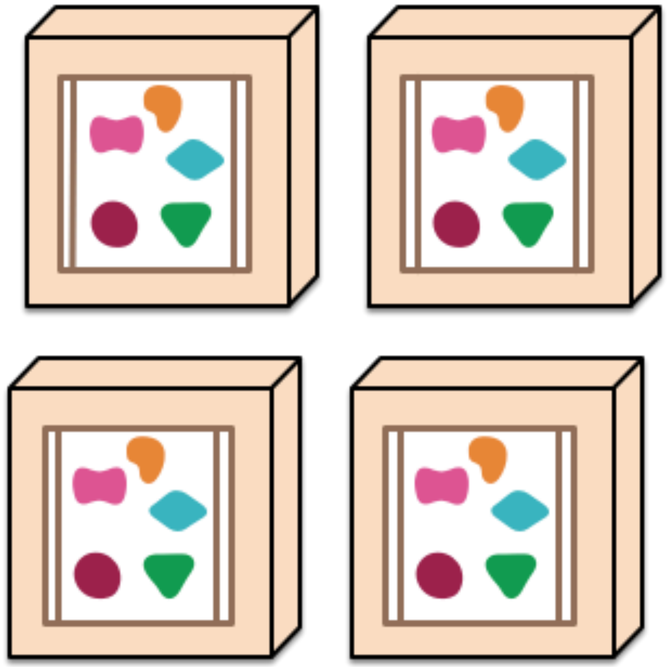
A Related Topic: Microservice

- A software architecture that develops an application as a suite of small services, each of which can be deployed and scaled independently
- When one (micro)service is in large demand, can scale it up
- Different (micro)services can be written and managed by different teams
- Changing one (micro)service will not affect the others

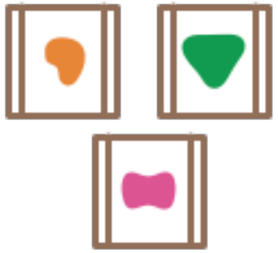
A Related Topic: Microservice

A monolithic application puts all its functionality into a single process...

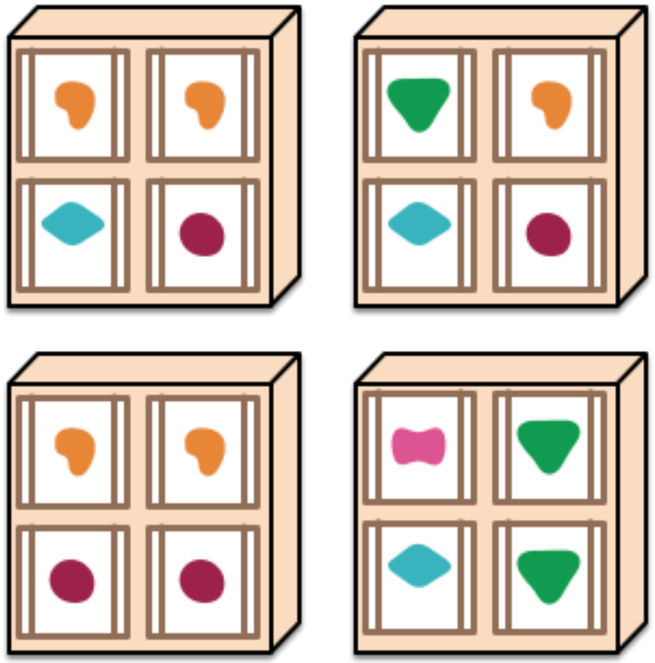
... and scales by replicating the monolith on multiple servers



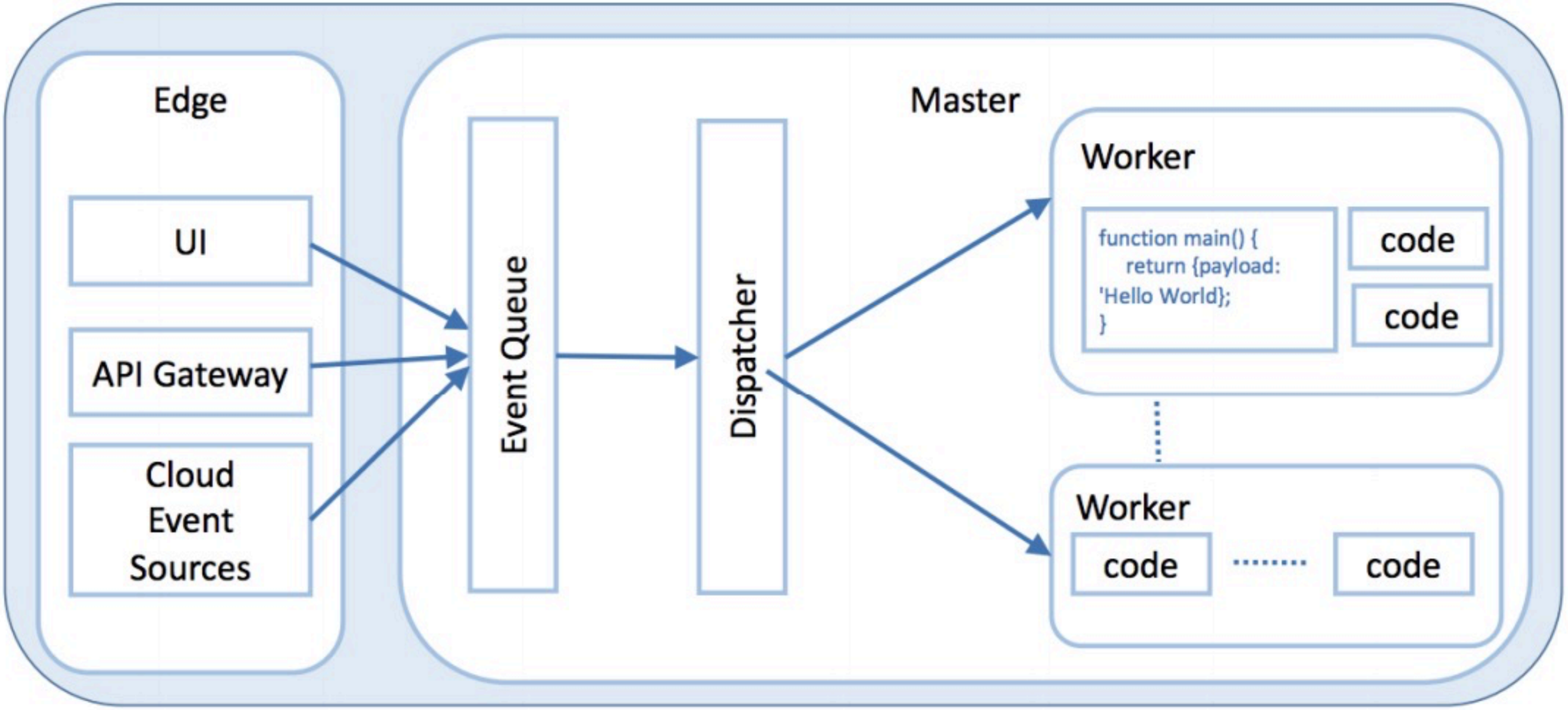
A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



Basic Architecture



First serverless app: BigQuery

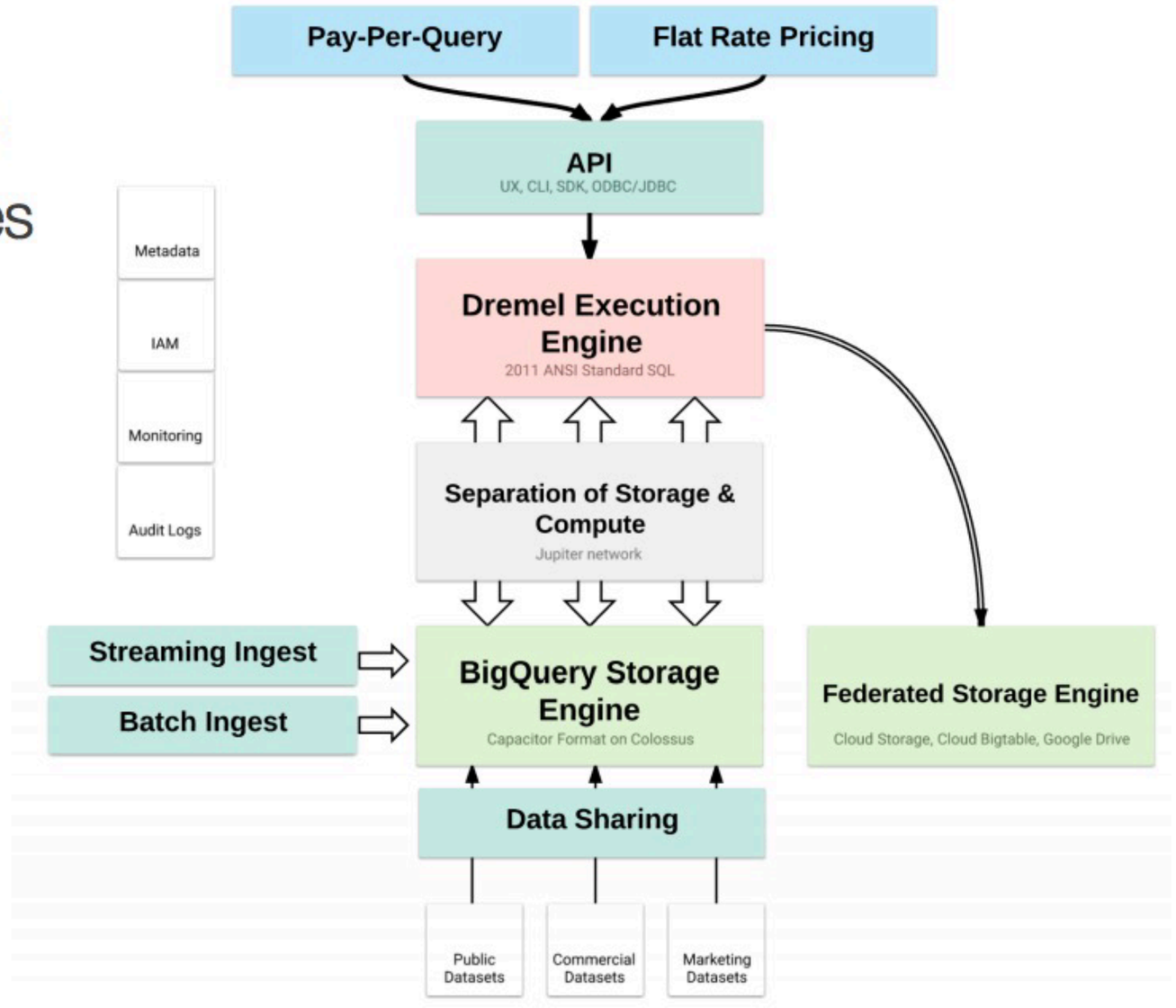
Google Big Query

Fully managed Data Warehouse

- “Arbitrarily” large data and queries
- Pay per bytes being processed
- No concept of cluster

Other similar systems

- AWS Athena
- Snowflake
- ...



AWS Lambda

- An event-driven, serverless computing FaaS platform introduced in 2014
- Functions can be written in Node.js, Python, Java, Go, Ruby, C#, PowerShell
- Each function allowed to take 128MB - 10GB memory and up to 15min
- Max 50-3000 concurrent functions
- Connected with many other AWS services

Lambda Function Triggering and Billing Model

- Run user handlers in response to events
  - web requests (RPC handlers)
  - database updates (triggers)
  - scheduled events (cron jobs)
- Pay per function invocation
  - No charge when no functions run (no triggering event)
  - Billed by duration of function, configured memory size, and # of functions
    - charge *actual_time * memory_cap*

Serverless Applications

Event source

Changes in data state

Requests to endpoints

Changes in resource state

Lambda function

Node.js
Python
Java
C# (.NET Core & Core 2.0)
Go
Ruby
Powershell
BYR – Bring your own Runtime

Services (anything)

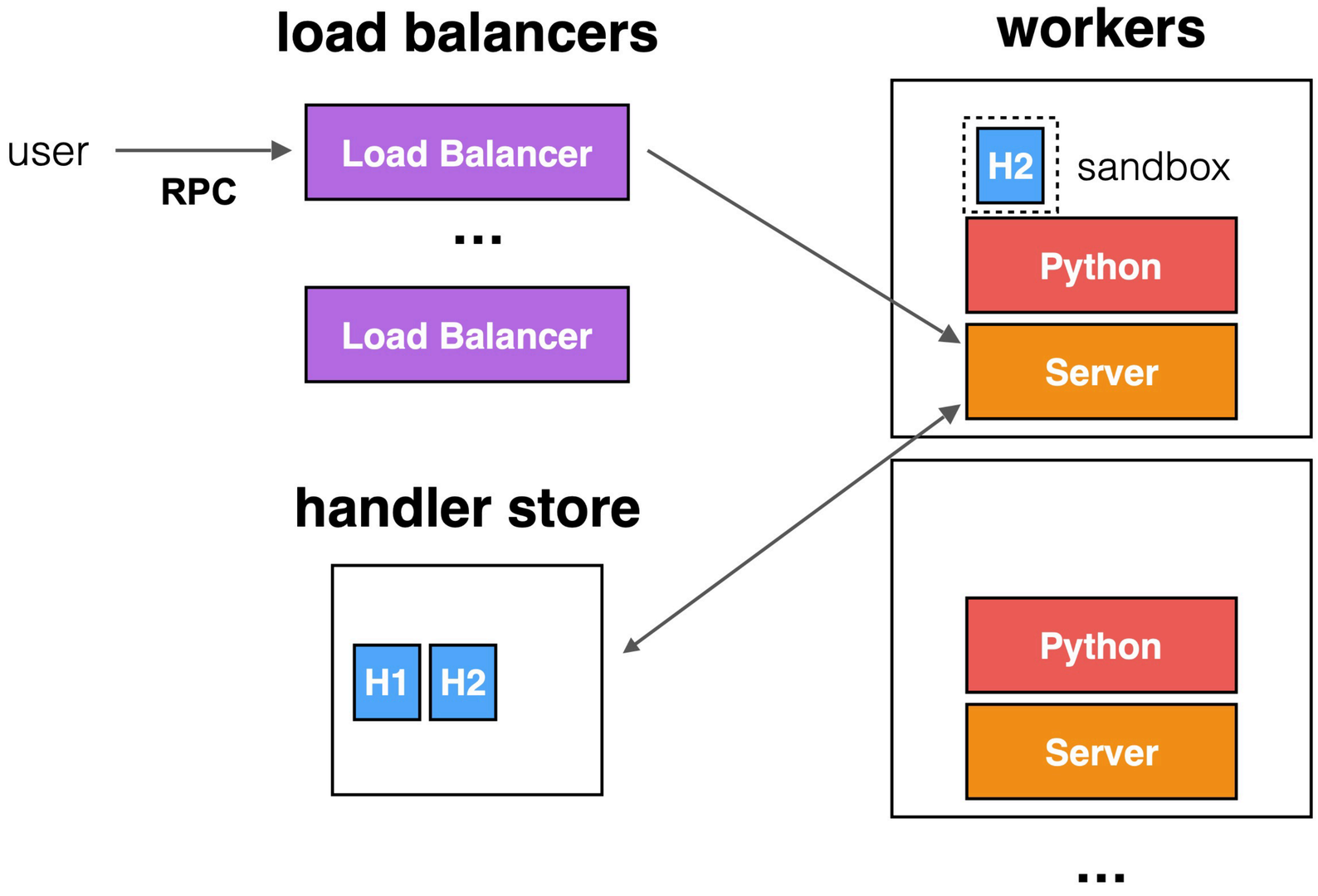
© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

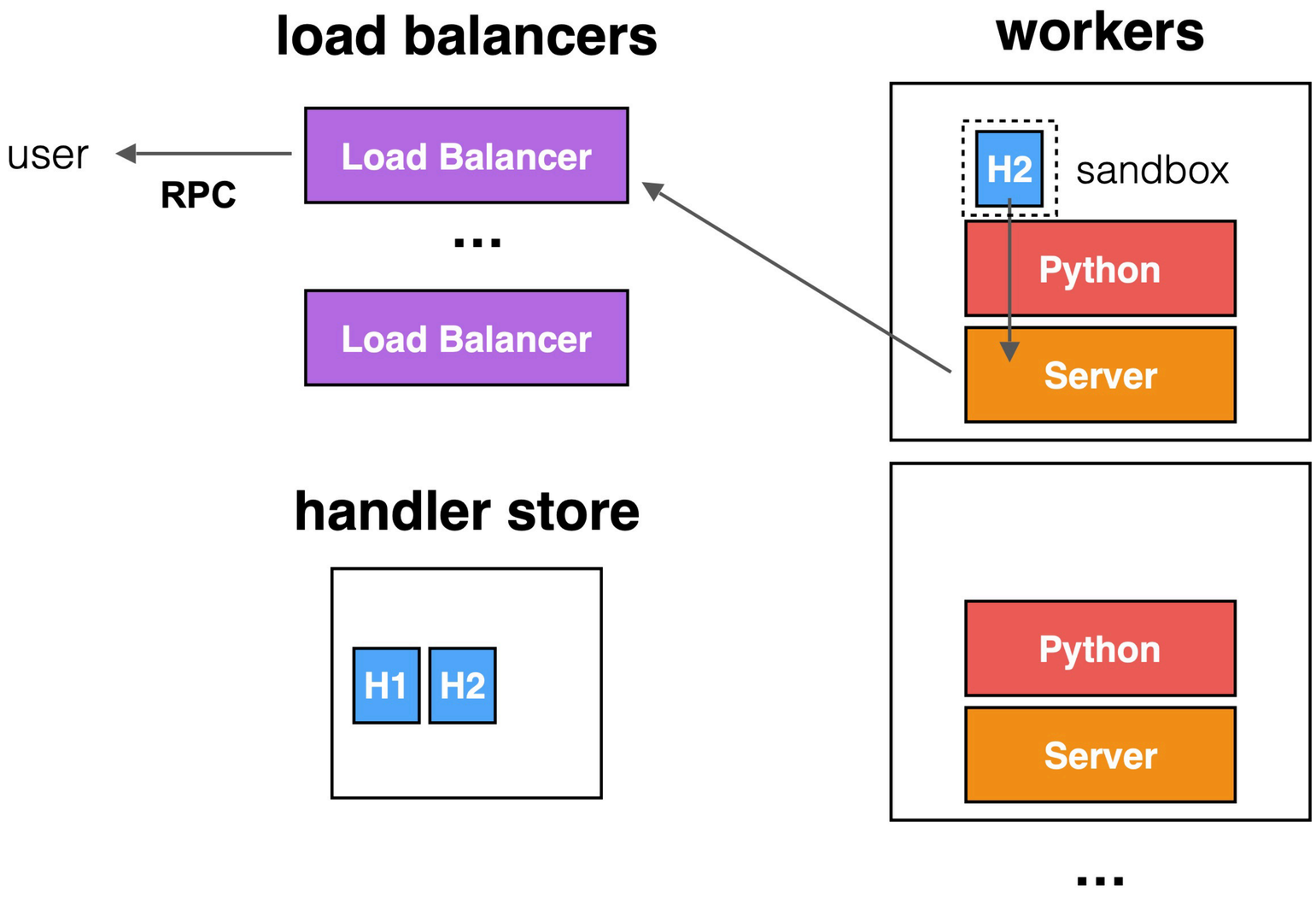
Internal Execution Model

- Developers upload function code to a *handler store* (and associate it with a URL)
- Events trigger functions through RPC (to the URL)
- Load balancers handle RPC requests by starting *handlers* on *workers*
  - Calls to the same function are typically sent to the same worker(s)
- Handlers sandboxed in containers
  - AWS Lambda reuses the same container to execute multiple handlers when possible

Execution Model



Execution Model



Azure Functions

- Debuted in 2016
- Support Python, C#, Java, JavaScript and PowerShell
- Three plans:
  - Consumption, Premium, and Dedicated
  - Dedicted: functions run on dedicated VMs and scaled manually (unlimited duration)
  - Max duration for consumption and premium: 10min and 60min
  - Max memory: 1.5GB - 14GB (depending on plans)

Google Cloud Functions

- Support Node.js, Python, and Go
- Memory size 128MB - 2GB
- Max function duration 9min
- Max number of functions per project: 1000
- Bill CPU and memory separately

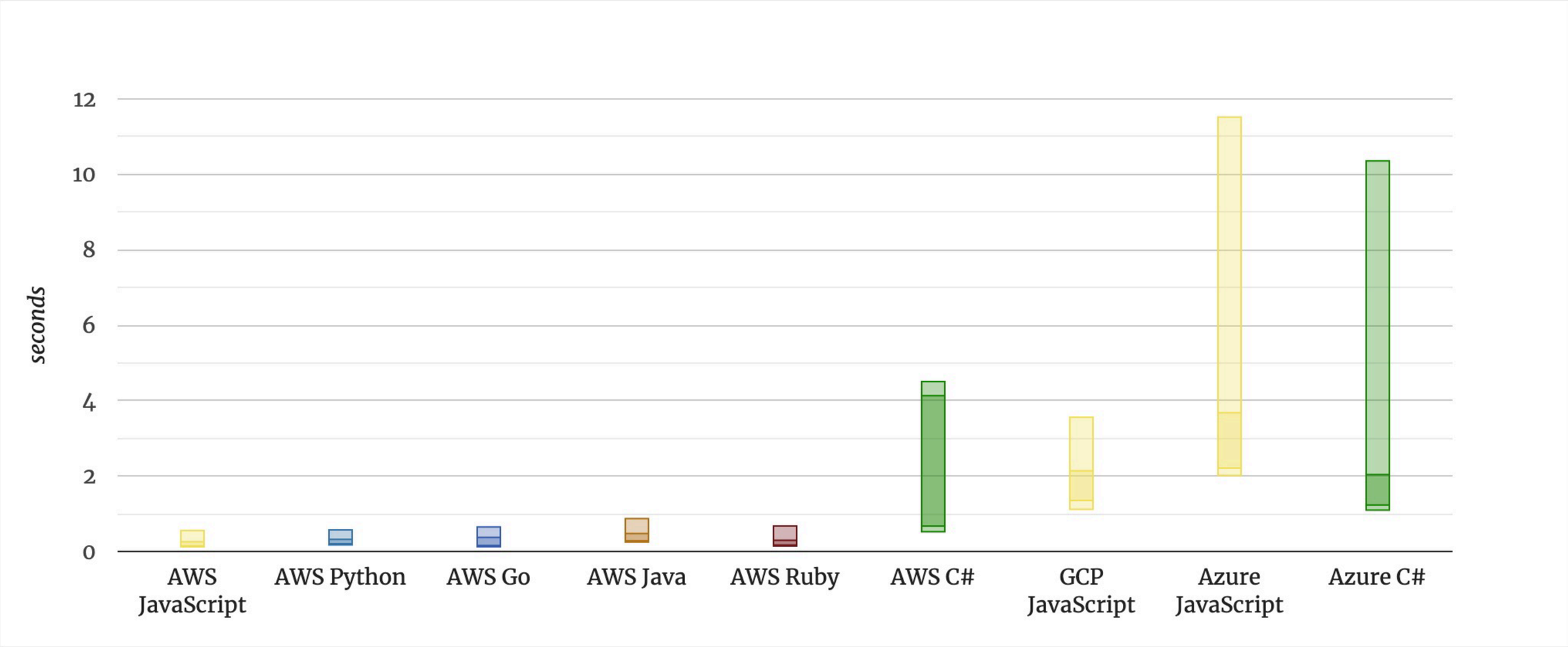
Open-Source Projects

- OpenWhisk
  - Originally developed by IBM and the core technology behind IBM Cloud Functions
- Cloudburst
  - Stateful serverless framework built on top of the Anna key-value store

Limitations of Today's Serverless Offerings

- Difficult and slow to manage states
  - Have to use (slow) cloud storage or other VMs!
- No easy or fast way to communicate across functions
  - Have to go through cloud storage or other services
- Functions can only use limited resources, and resources have "fixed ratios"
- No control over function placement or locality
  - e.g., starting functions on "cold" machines can be slow
- Billing model does not fit all needs

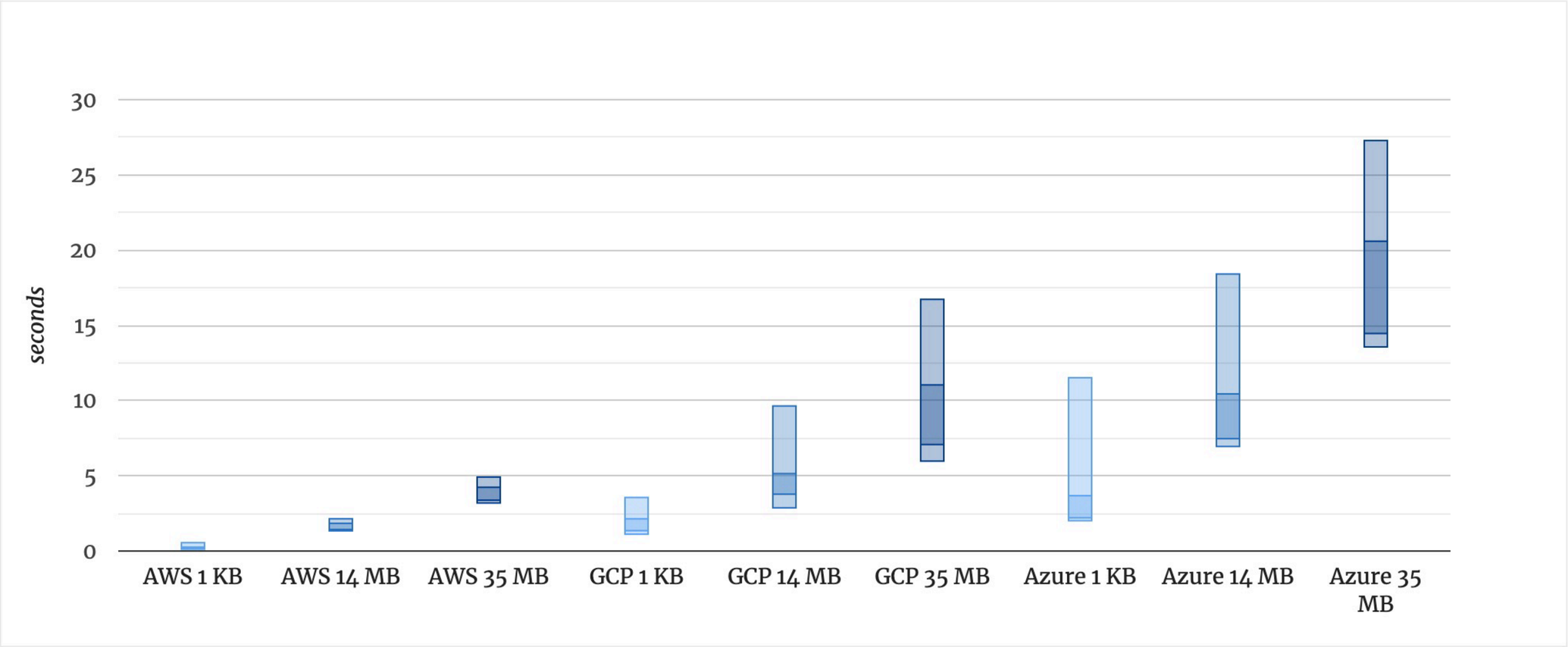
Cold Start Time of Different Languages/Offerings



Typical cold start durations per language

source: https://mikhail.io/serverless/coldstarts/big3/

Cold Start Time with Different Package Sizes



Comparison of cold start durations per deployment size (zipped)

source: https://mikhail.io/serverless/coldstarts/big3/

Good and Bad Use Cases

- Some good ones:
  - Parallel, independent, stateless tasks
  - Orchestrating functions
  - Function composition
- Some bad ones:
  - Stateful applications
  - Distributed applications and protocols
  - Applications that demand more resources (esp. memory)

Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider

Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini *

Microsoft Azure and Microsoft Research

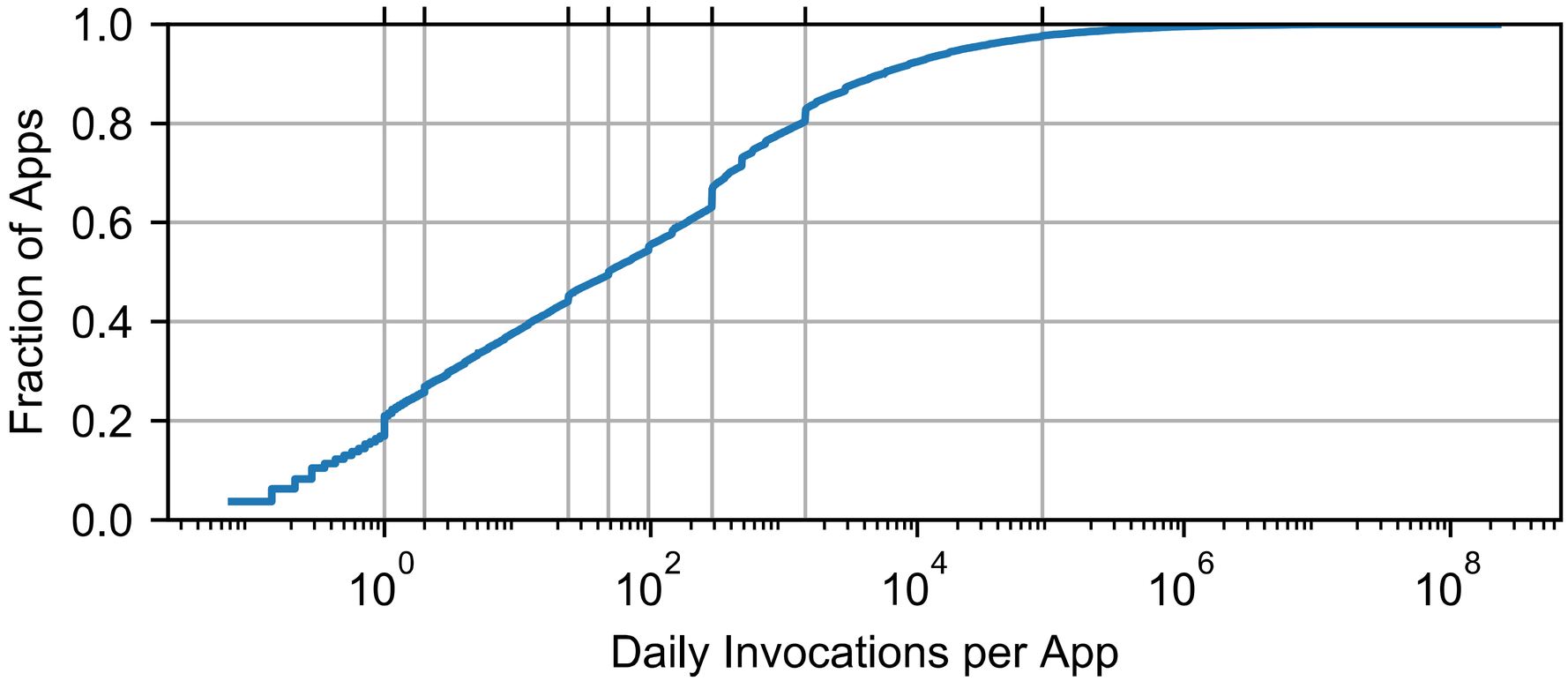
USENIX ATC 2020

- First characterization of production serverless workloads
  - Many new insights
  - Released production traces of Azure Functions:
    - https://github.com/Azure/AzurePublicDataset

Microsoft Azure

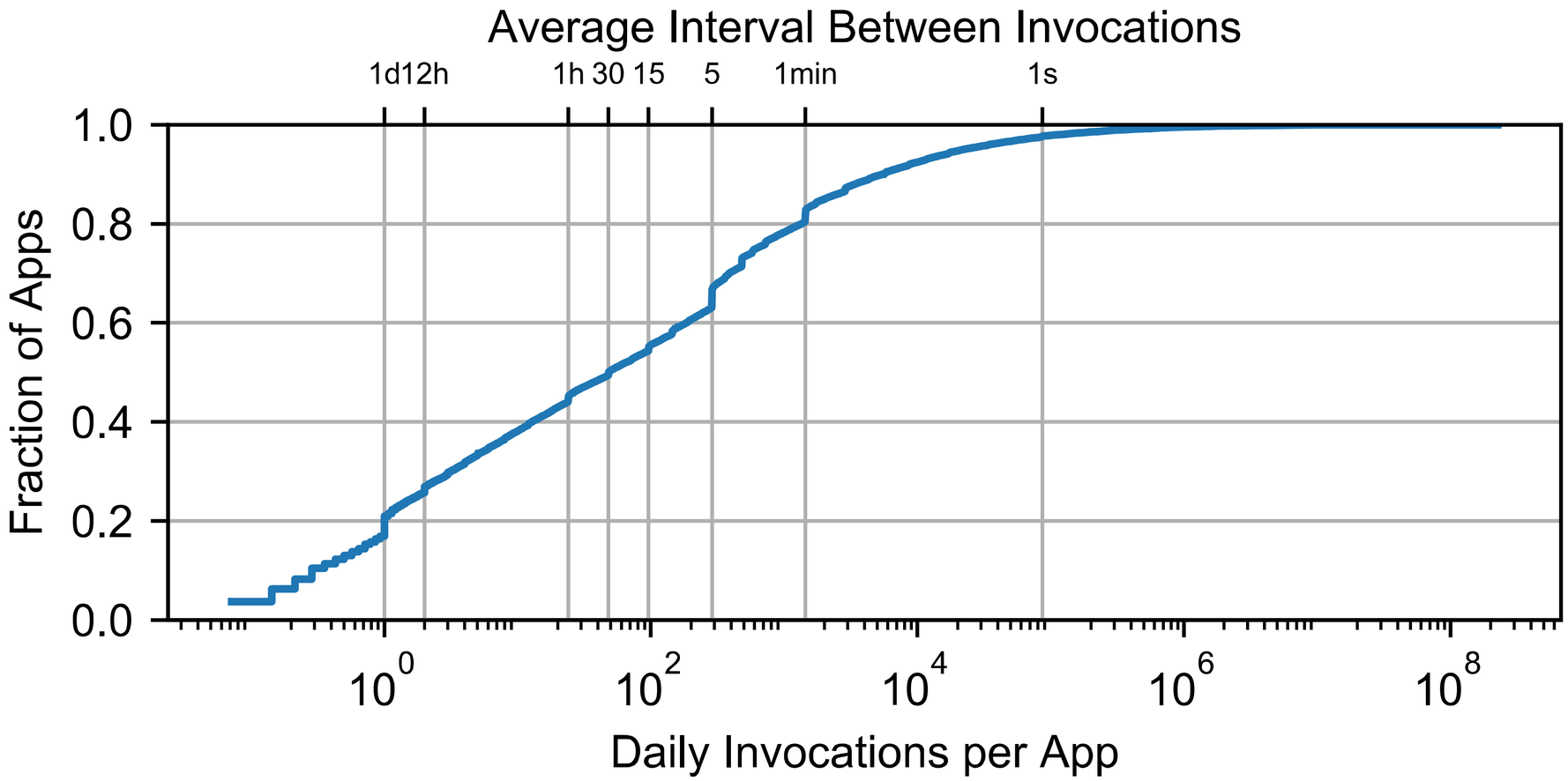
- A new adaptive serverless management scheme
  - New angle: going after eliminating cold starts instead of reducing cold start overhead
  - Improves the user experience, reduces the underlying resource usage
  - Deployed in production

Invocations per Application



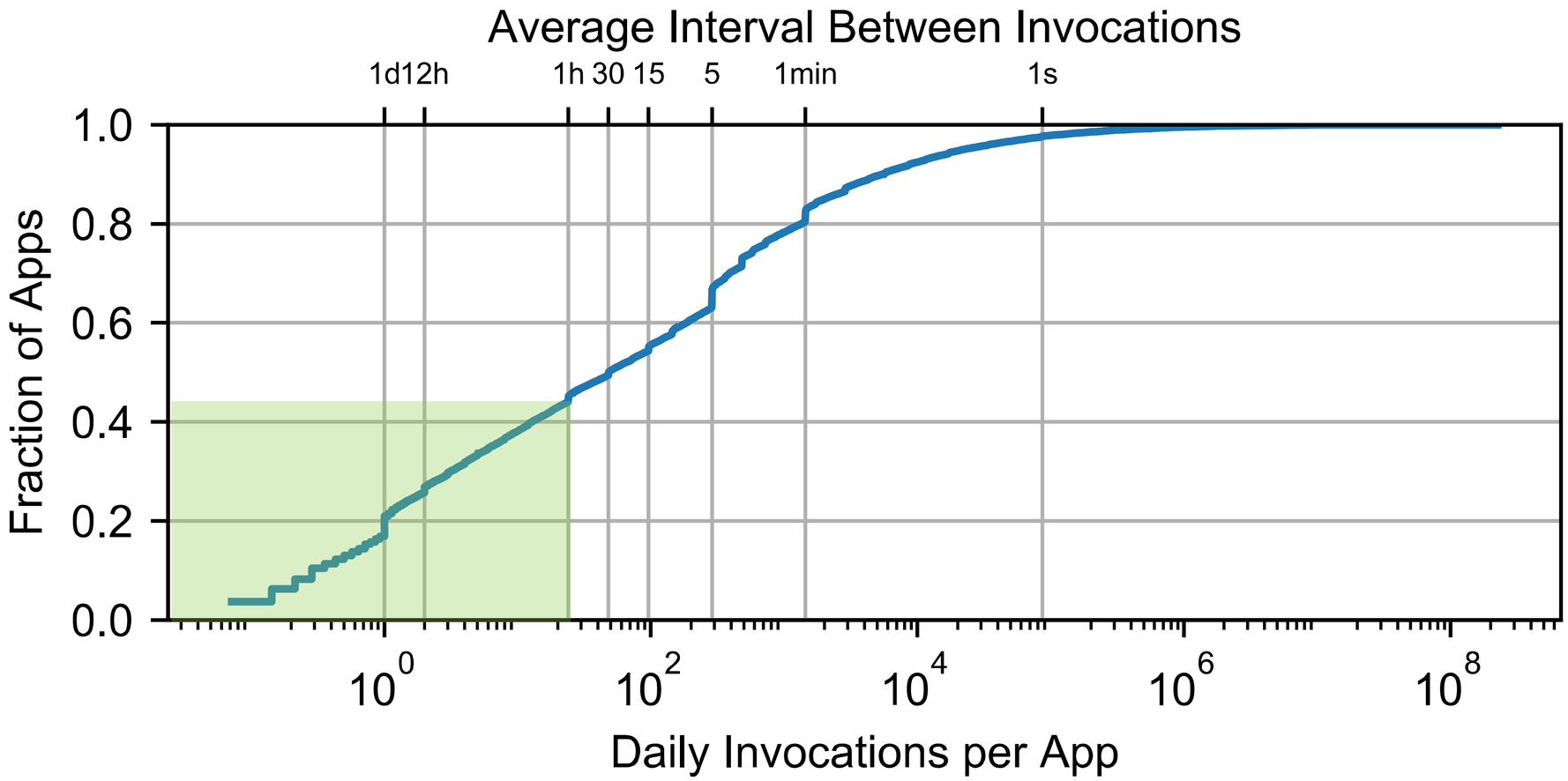
This graph is from a representative subset of the workload. See paper for details.

Invocations per Application



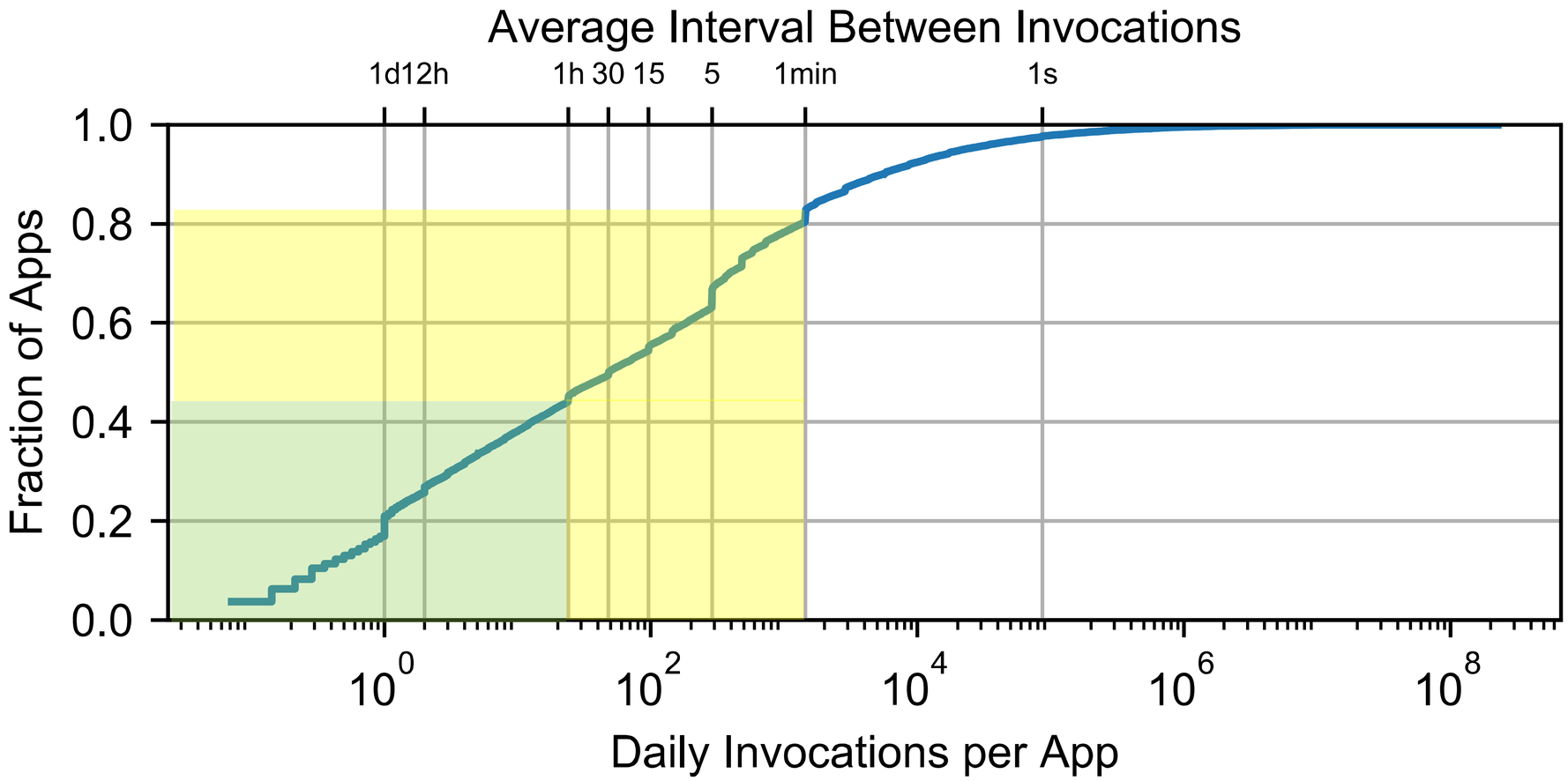
This graph is from a representative subset of the workload. See paper for details.

Invocations per Application



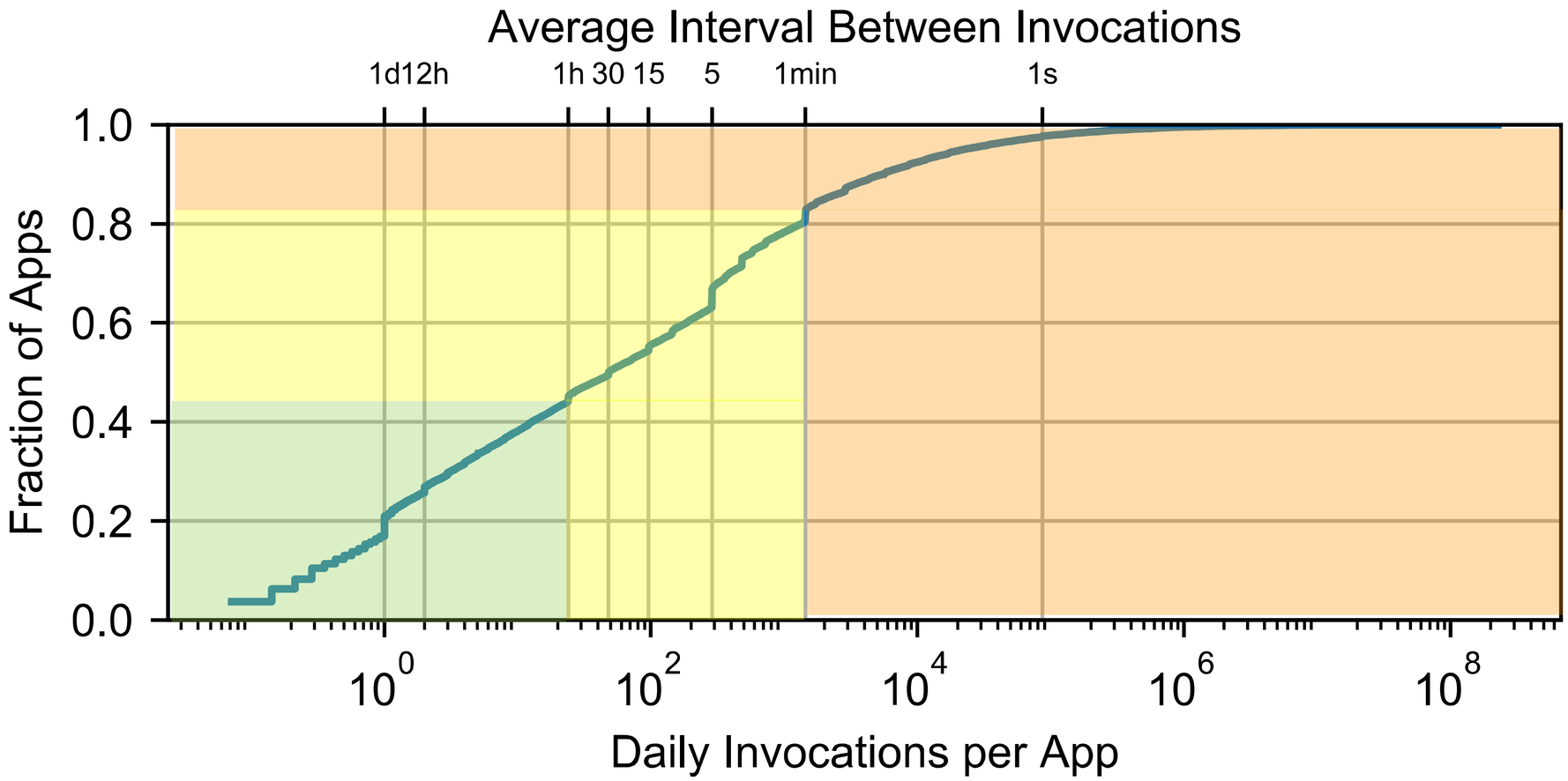
This graph is from a representative subset of the workload. See paper for details.

Invocations per Application



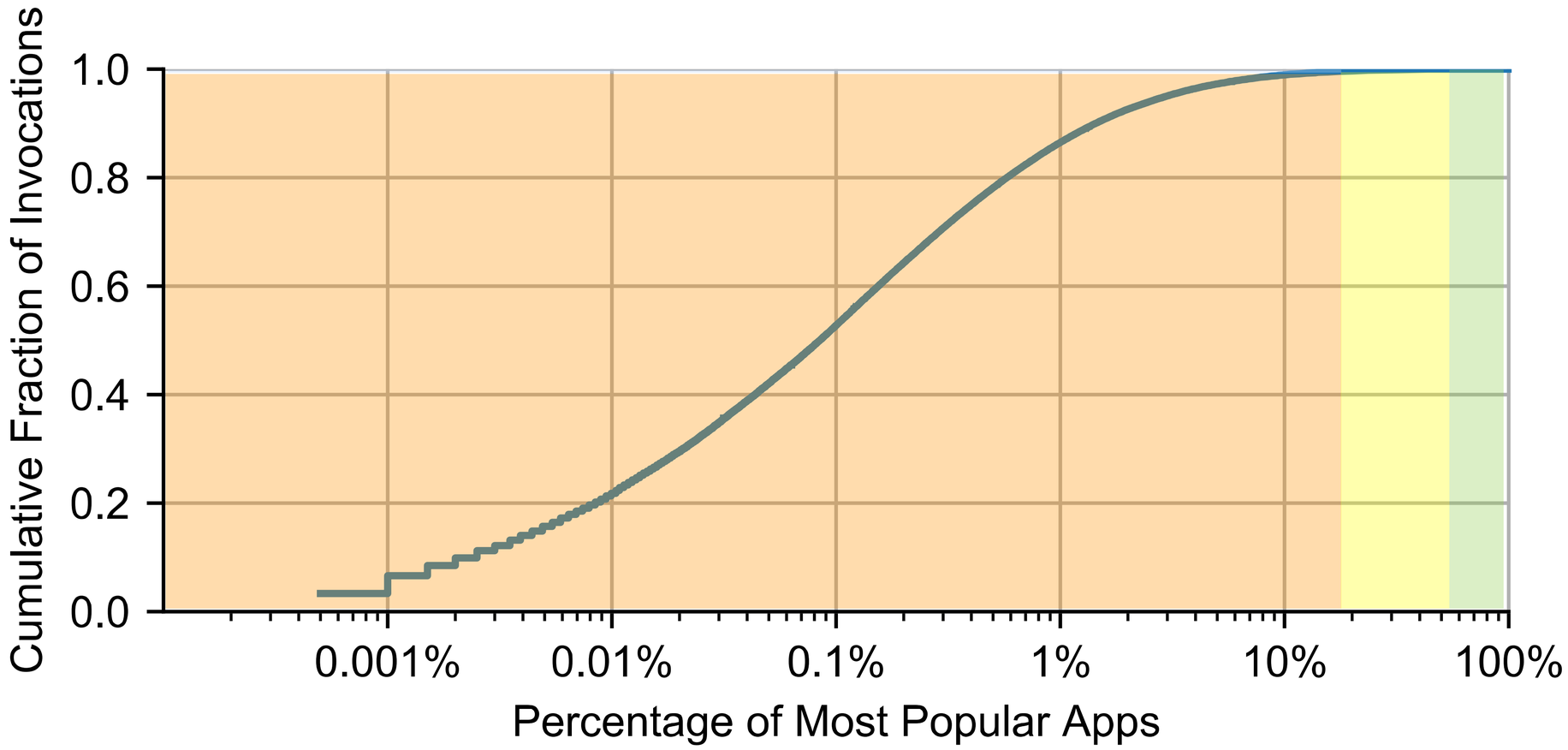
This graph is from a representative subset of the workload. See paper for details.

Invocations per Application



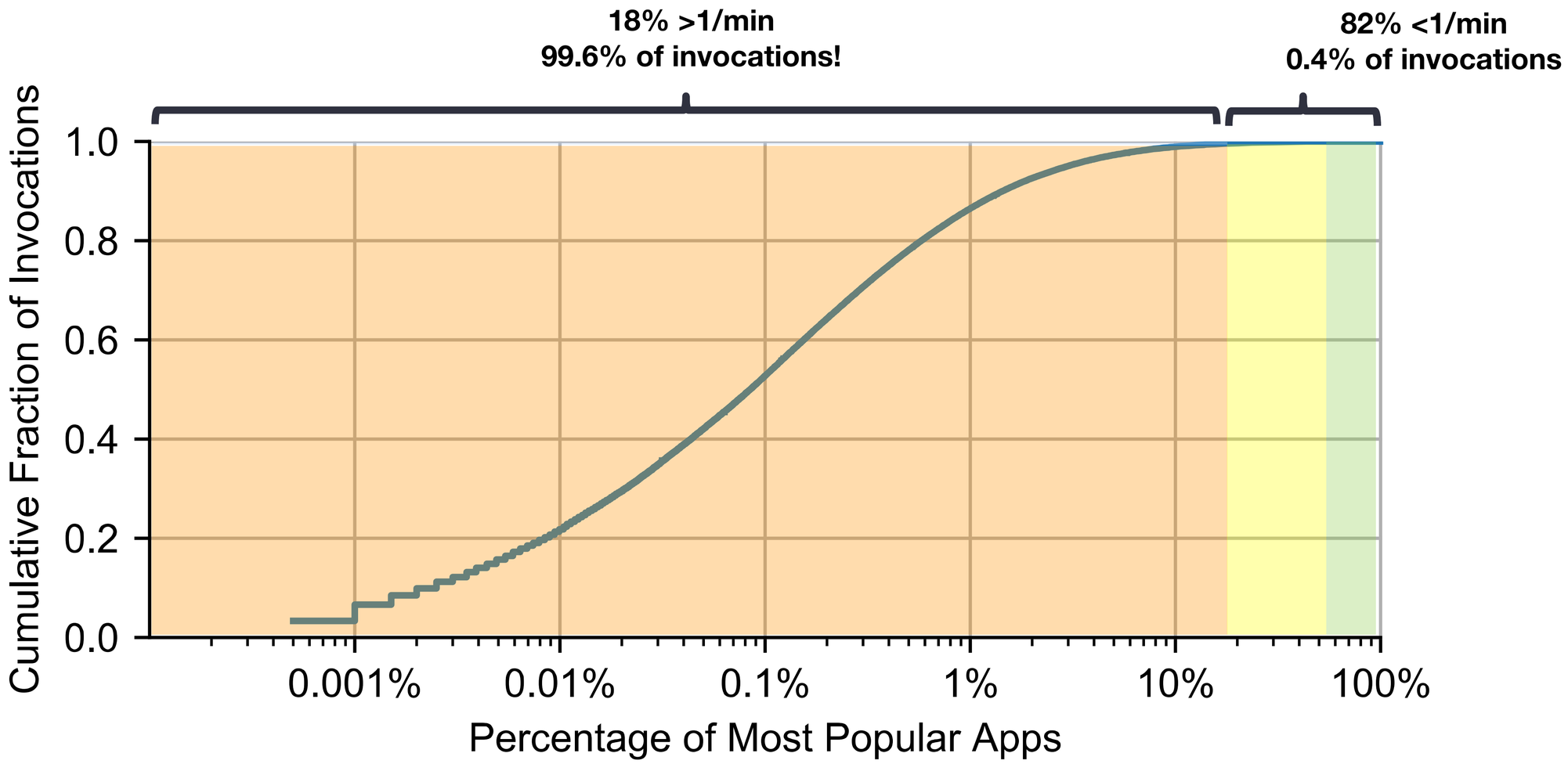
This graph is from a representative subset of the workload. See paper for details.

Invocations per Application



This graph is from a representative subset of the workload. See paper for details.

Invocations per Application



This graph is from a representative subset of the workload. See paper for details.

Who should you design/optimize the system for?

18% of applications that constitute 99.6% of invocations

- Can lead to immediate cost reductions for the provider.
- Keeps the big users happy and prevents them from migrating to competitors.

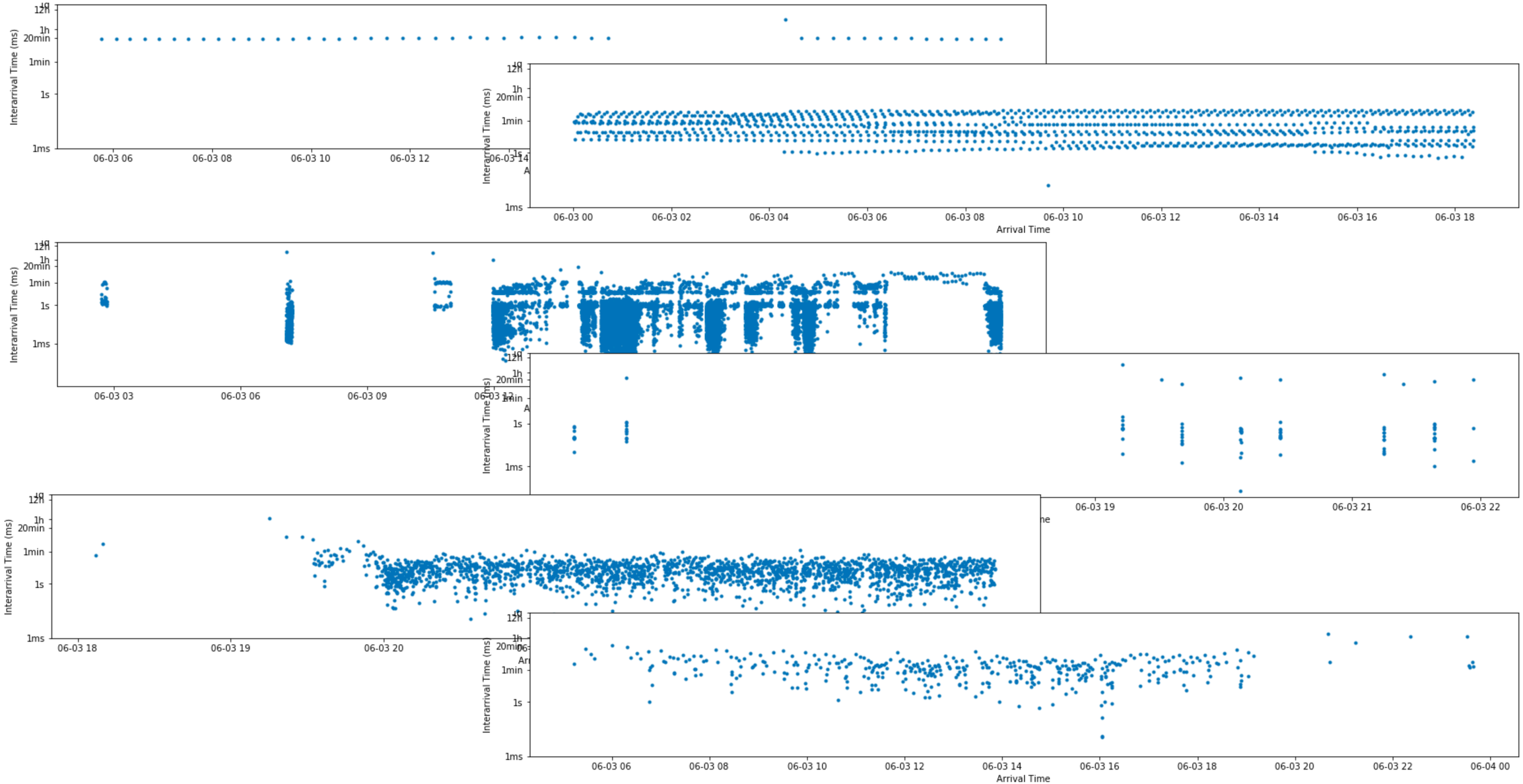
or

82% of applications that constitute 0.4% of invocations

- Can incentivize more users to shift to serverless. Potentially higher long-term profit.
- Seems reasonable when user valuation of service/ QoS is unknown.

Mixing both?

Average Invocations Do Not Tell the Entire Story



Performant Serverless

Short Execution

Low Cold Start Overhead

Fewer Cold Starts

Minimum Wait Time

Efficient Serverless

High Co-tenancy

Minimal Resource Wastage

High Cluster Utilization

It is all about trade-offs!

Cold Starts and Resource Wastage

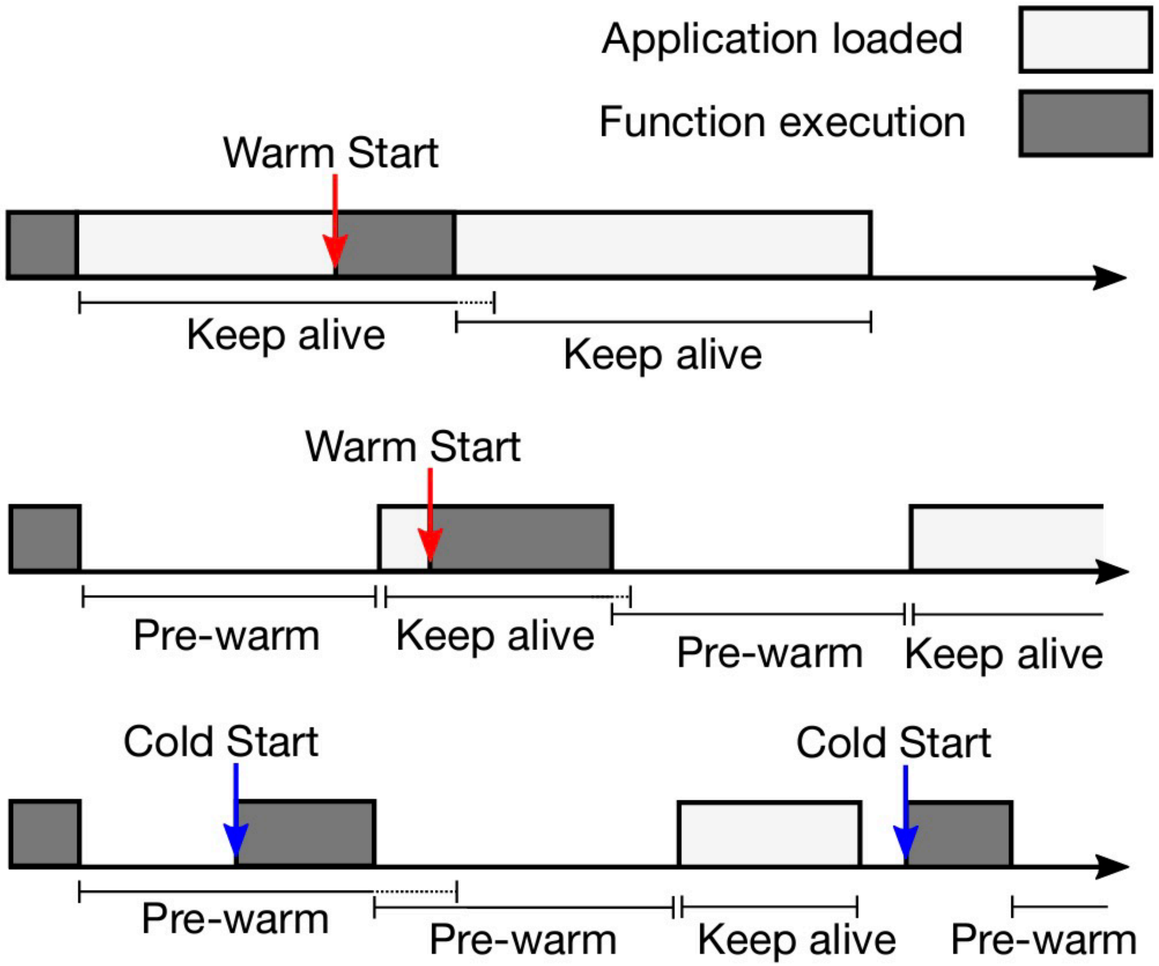
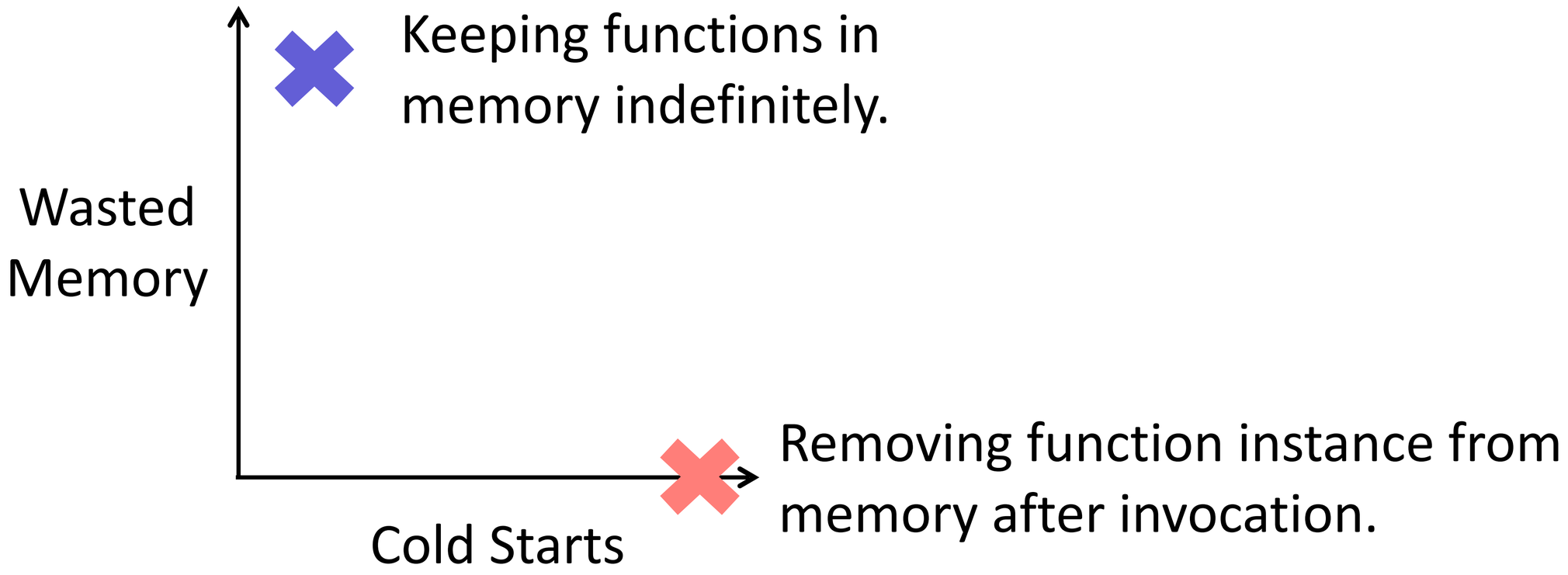


Figure 9: Timelines showing a warm start with keep alives and no pre-warming (top); a warm start following a pre-warm (middle); and two cold starts, before a pre-warm, and after a keep alive (bottom).

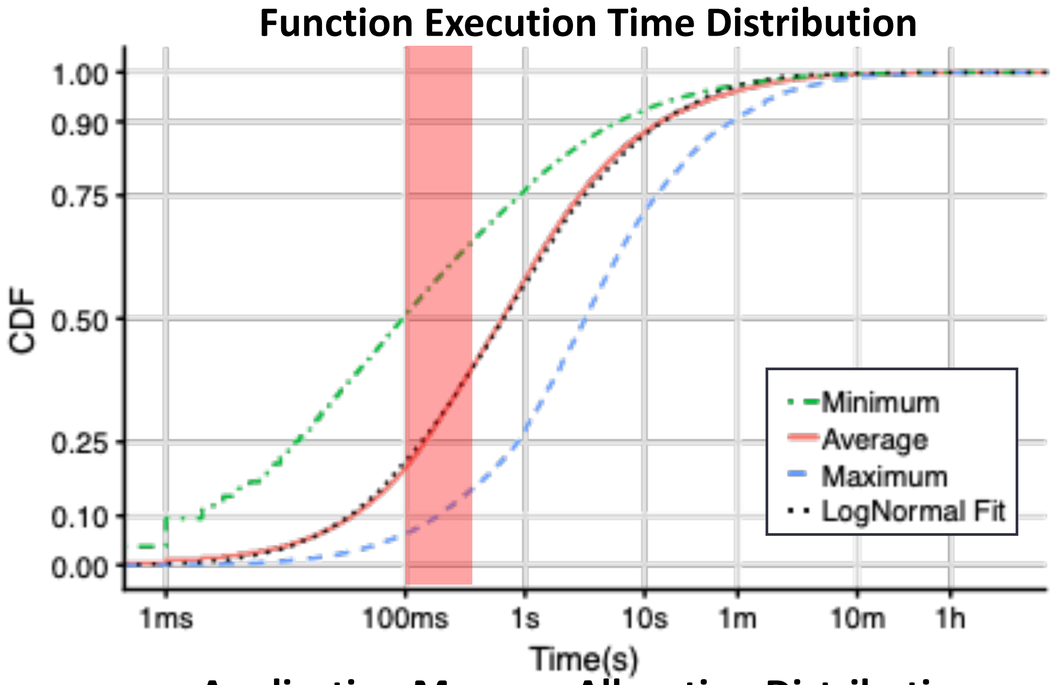
Cold Starts and Resource Wastage



Removing function instance from memory after invocation.

O(execution time) = O(cold start overhead)

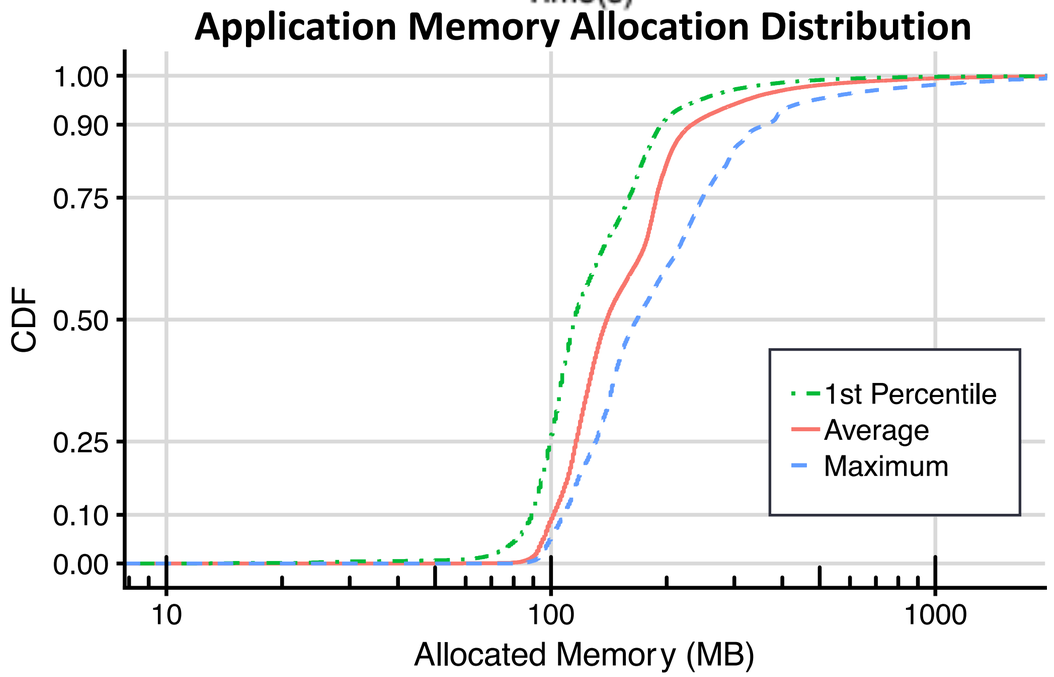
Cold starts cannot be neglected.



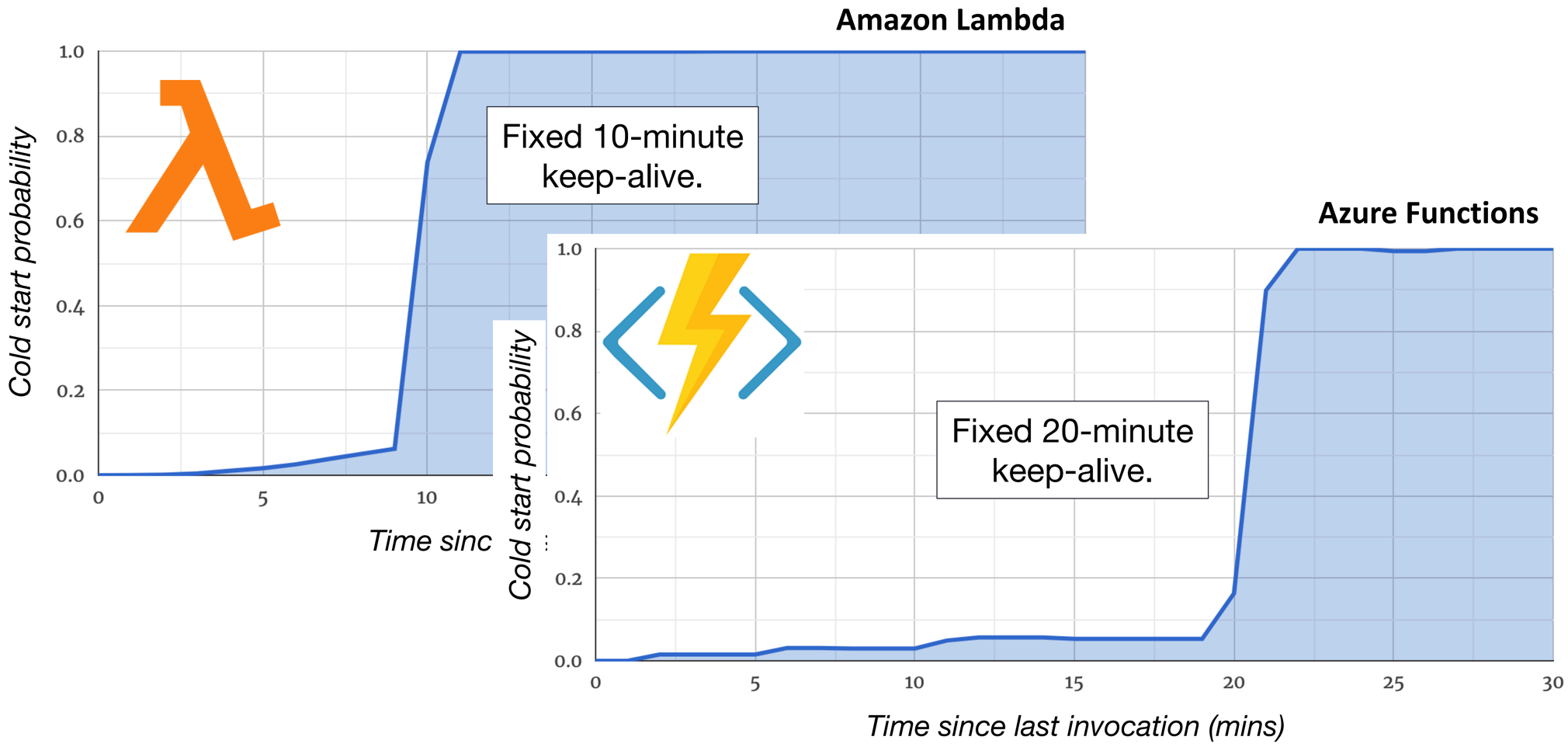
Keeping functions in memory indefinitely.

Memory usage not negligible.

Can't be kept in memory forever.



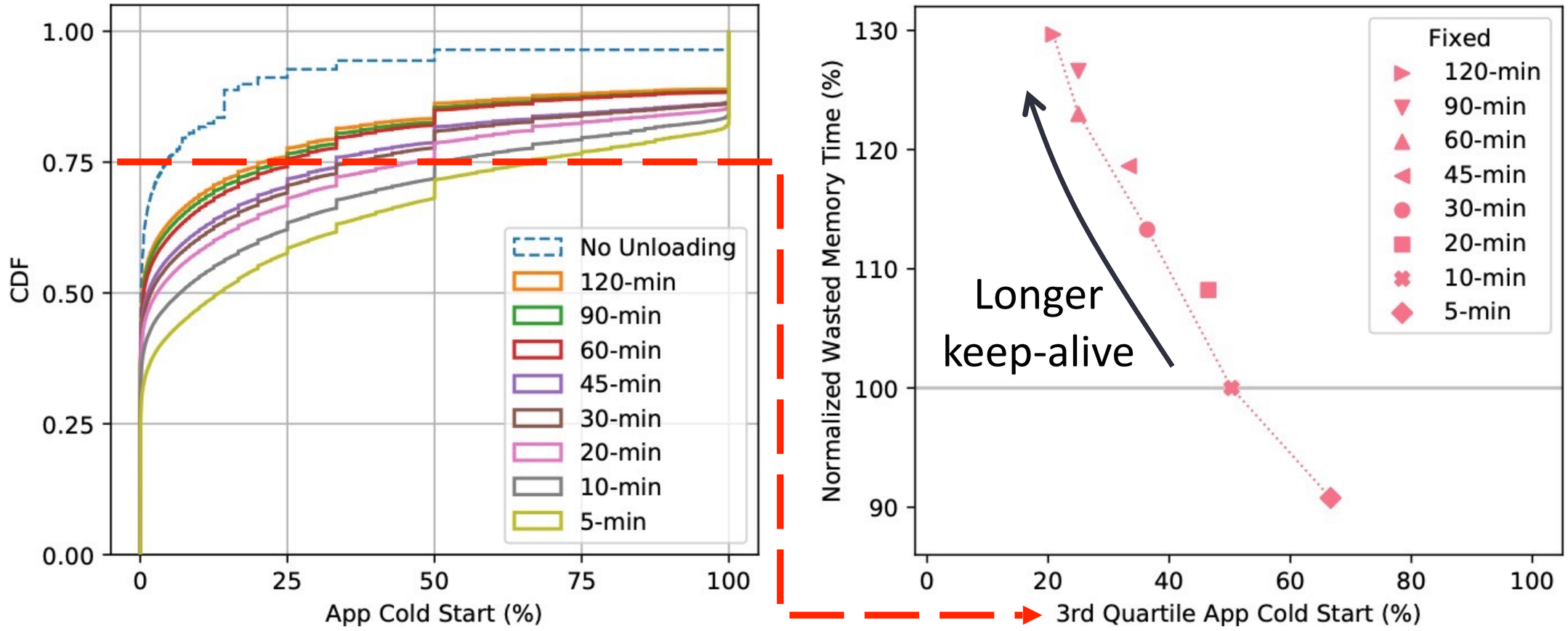
What do serverless providers do?



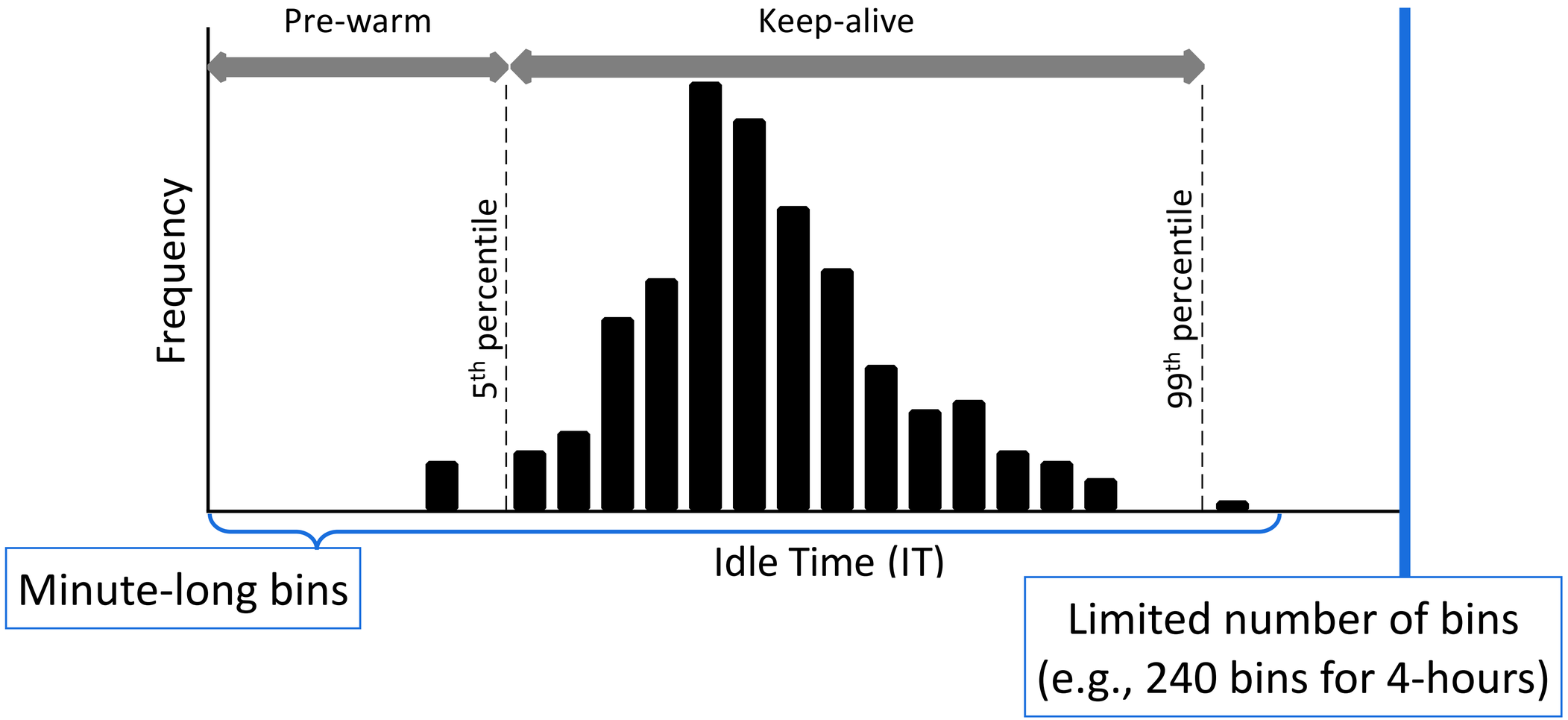
Mikhail Shilkov, Cold Starts in Serverless Functions, https://mikhail.io/serverless/coldstarts/

Fixed Keep-Alive Policy

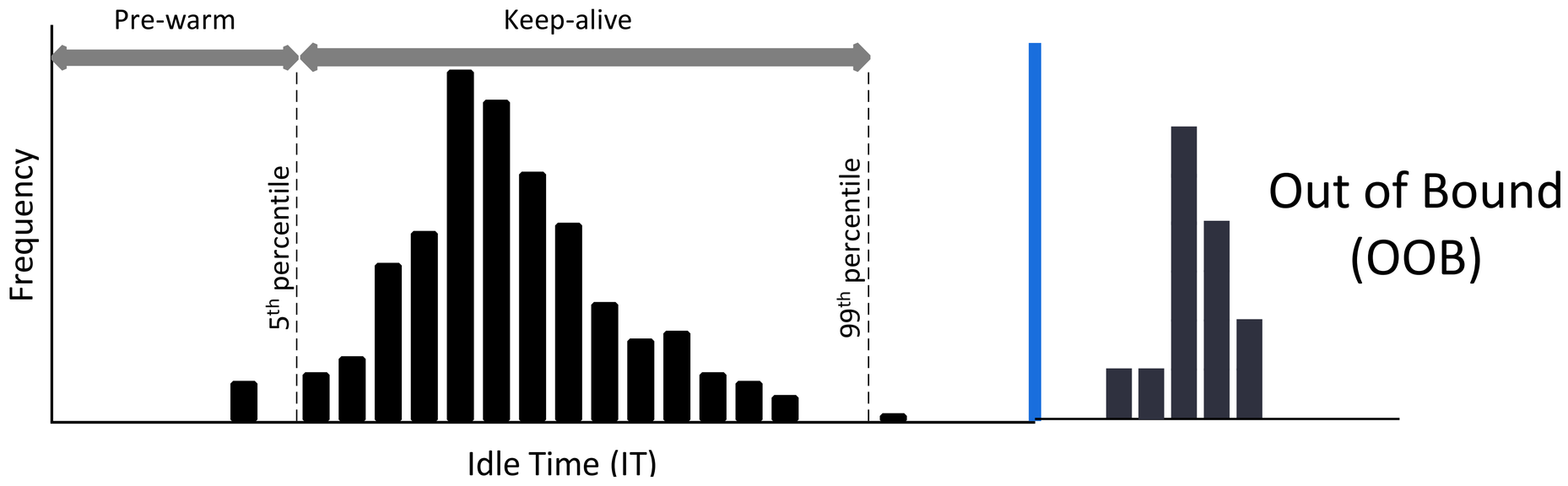
Results from simulation of the entire workload for a week.



A Histogram Policy To Learn Idle Times



The Hybrid Histogram Policy

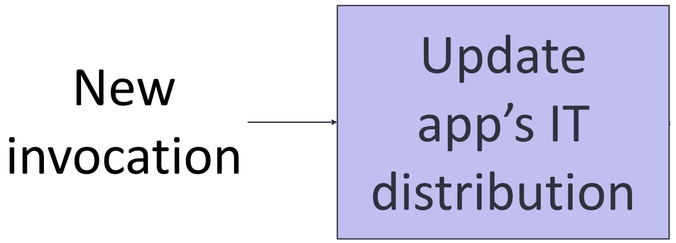


We can afford to run complex predictors given the low arrival rate.

A histogram might be too wasteful.

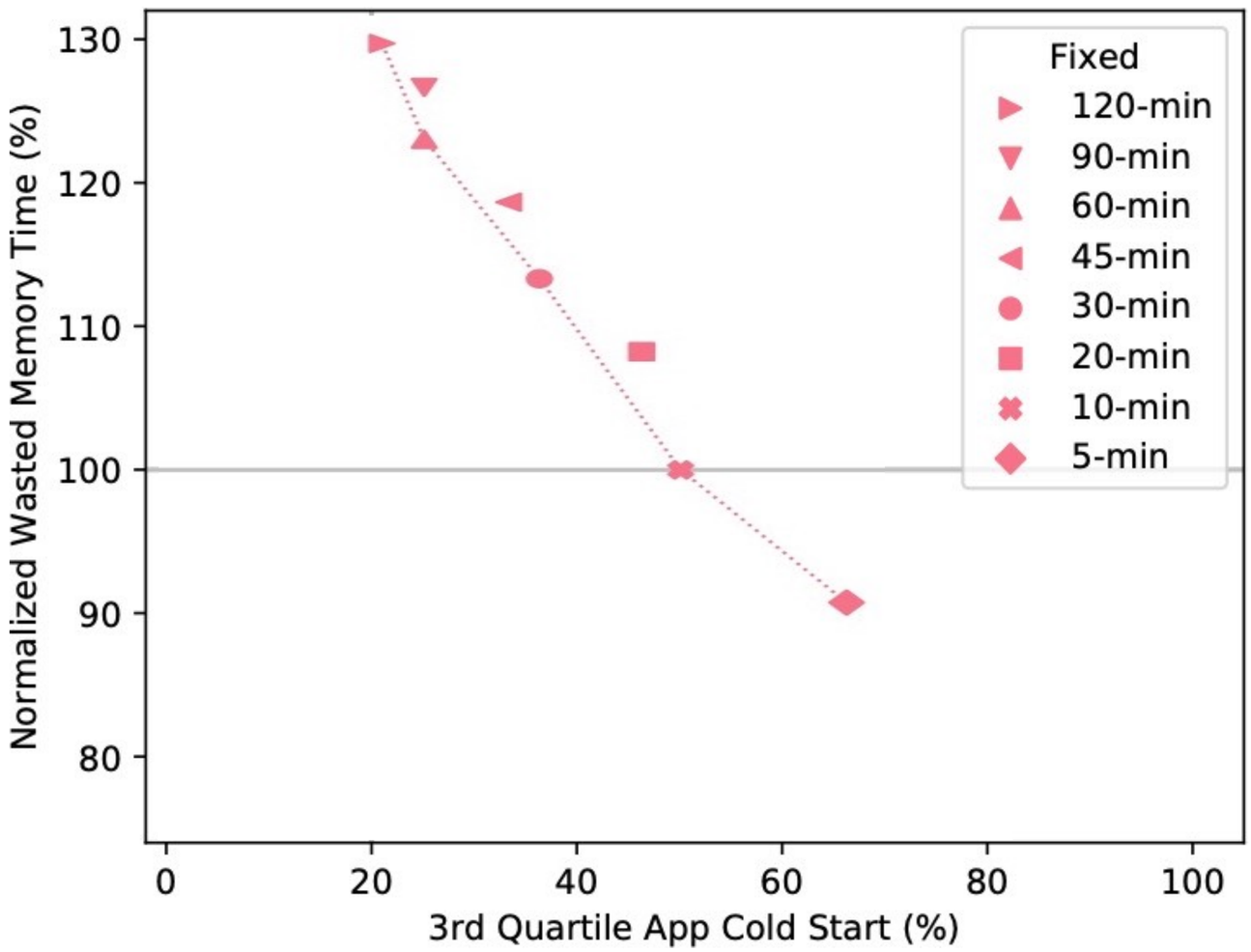
Time Series Forecast

Decision Tree for the Hybrid Histogram Policy



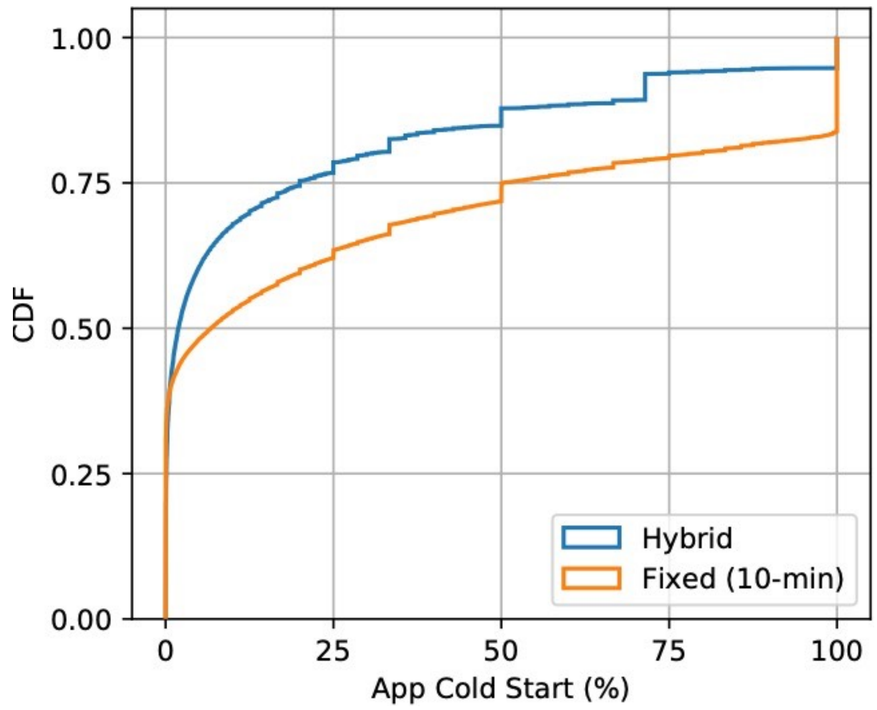
ARIMA: Autoregressive Integrated Moving Average

More Optimal Pareto Frontier

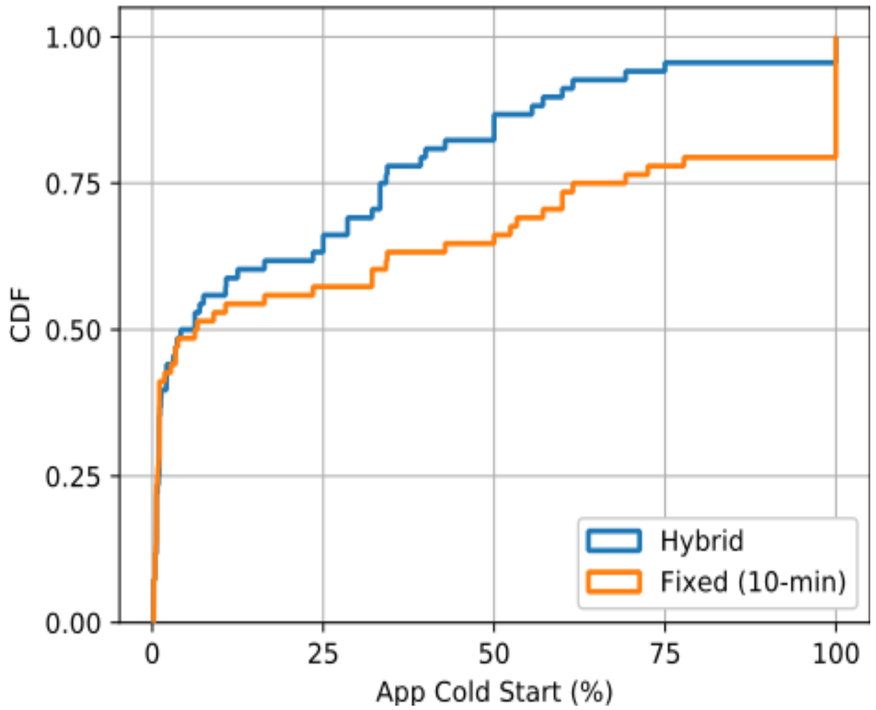


4-Hour Hybrid Histogram

Simulation



Experimental



Container memory reduction: 15.6%

Average exec time reduction: 32.5%

99th–percentile exec time reduction: 82.4%

Latency overhead: < 1ms (~800 *μs*)

Production possible thanks to the dedication of Íñigo Goiri, Gohar Chaudhry, and the Azure Functions team.