

# The **State** of Serverless Computing or, Fixing **Dysfunction**-as-a-Service

Chenggang Wu

RISE Lab, UC Berkeley

QCon New York 06/24/2019

UC Berkeley: CS 61A  
Fall 2018



# Berkeley

Division of  
Data Sciences



# Making **Programmers** Productive

**Key Question:** Where will code be run? In the ☁️!

# Background: Serverless Computing

# What is serverless computing?

*Serverless computing is a programming abstraction that enables users to upload programs, run them at any scale, and pay only for resources used.*

# Functions-as-a-Service (FaaS)

- AWS Lambda, Google Cloud Functions, OpenWhisk (IBM), Azure Functions, OpenLambda, OpenFaaS, kNative...
- Optimized for **simplicity** – register functions, enable triggers, and scale transparently

## Function code [Info](#)

Code entry type

Edit code inline

Runtime

Python 3.6

```
File Edit Find View Go Tools Window
Environment
  election
    lambda_function.py
  lambda_function.py
1
2 import boto3
3 import pickle
4 import random
5 import time
6
7 client = boto3.resource('dynamodb')
8 table = client.Table('vsreekanti')
9 candidates = client.Table('candidates')
10 THRESHOLD = 50
11
12 def main(thisid):
13     print("My invocation's id is: " + thisid)
14     thisid = int(thisid)
15     vote_round_count = 0
16     am_leader = False
17
18     while True:
19         time.sleep(.25)
```

## Add triggers

Choose a trigger from the list below to add it to your function.

API Gateway

AWS IoT

Alexa Skills Kit

Alexa Smart Home

Application Load Balancer

CloudFront



Add triggers from the list on the left

# Academic Interest in Serverless

## Occupy the Cloud: Distributing

Eric Jonas, Qifan Pu, Shivaram Venkataraman  
University of California, Berkeley  
{jonas, qifan, shivaram, istaitian}@cs.berkeley.edu

### ABSTRACT

Distributed computing remains inaccessible to a large number of users, in spite of many open source platforms and extensive commercial offerings. While distributed computation frameworks have moved beyond a simple map-reduce model, many users are still left to struggle with complex cluster management and configuration tools, even for running simple embarrassingly parallel jobs. We argue that stateless functions represent a viable platform for these users eliminating cluster management overhead, fulfilling the promise of elasticity. Furthermore, using our prototype implementation, PyWren, we show that this model is general enough to implement a number of distributed computing models such as BSP, efficient MapReduce, and MapReduce with straggler mitigation. Extrapolating from recent work on disaggregated storage, we argue that serverless computing is a natural fit for data processing.

### CCS CONCEPTS

• Computer systems organization  
• Computing methodologies

### KEYWORDS

Serverless, Distributed Computing, AWS Lambda, PyWren

### ACM Reference Format:

Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, Benjamin Recht. University of California, Berkeley {jonas, qifan, shivaram, istaitian}@cs.berkeley.edu. 2017. Occupy the Cloud: Distributed Computing for the 99%. In *Proceedings of SoCC '17, Santa Clara, CA, USA, September 24–27, 2017*, 7 pages.  
<https://doi.org/10.1145/3127479.3128601>

## 1 INTRODUCTION

Despite a decade of availability, the twin promises of scale and elasticity [2] remain out of reach for a large number of cloud computing users. Academic and commercially-successful platforms (Apache Hadoop, Apache Spark) with tremendous corporate backing (Amazon, Microsoft, Google) still present high barriers to entry for the average data scientist or scientific computing user. In fact, taking advantage of elasticity remains challenging for even sophisticated users, as the majority of these frameworks were designed to first

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions.acm.org](http://permissions.acm.org).  
*SoCC '17, September 24–27, 2017, Santa Clara, CA, USA*  
© 2017 Copyright held by the owner/authors(s). Publication rights licensed to Association for Computing Machinery.  
ACM ISBN 978-1-4503-5028-0/17/09...\$15.00

## Encoding, Fast Low-Latency Video Processing U

Sadjad Fouladi, Riad S. Wafar, Karthikeyan Vasuki Balasubramanian, Anirudh Sivaraman, George J. Rini, George J. Rini  
Stanford University, University of California San I

### Abstract

We describe ExCamera, a system that can edit, transform, and encode a video, including 4K and VR material, with low latency. The system makes two major contributions.

First, we designed a framework to run general-purpose parallel computations on a commercial “cloud function” service. The system starts up thousands of threads in seconds and manages inter-thread communication.

## Cloud Programming A Berkeley View

Eric Jonas, Johann Schlegel, Anurag Khandelwal, Qifan Pu, Karl Krauth, Neeraja Yegorov, Ion Stoica

### server

### ABSTRACT

Serverless computing offers the potential to an autoscaling, pay-as-you-go manner. In critical gaps in first-generation serverless computing: notably data-centric and distributed also open source and custom hardware. We make current serverless offerings a bad fit and particularly bad for data systems innovating pinpointing some of the main shortfalls of architectures, we raise a set of challenges w

## Serverless Computing

Joseph M. Hellerstein, Jose Foweraker

{hellerstein, jmf}

### ABSTRACT

Serverless computing offers the potential to an autoscaling, pay-as-you-go manner. In critical gaps in first-generation serverless computing: notably data-centric and distributed also open source and custom hardware. We make current serverless offerings a bad fit and particularly bad for data systems innovating pinpointing some of the main shortfalls of architectures, we raise a set of challenges w

## Peeking Behind the Curtains of Serverless Platforms

Liang Wang<sup>1</sup>, Mengyuan Li<sup>2</sup>, Yinqian Zhang<sup>2</sup>, Thomas Ristenpart<sup>3</sup>, Michael Swift<sup>1</sup>

<sup>1</sup>UW-Madison, <sup>2</sup>Ohio State University, <sup>3</sup>Cornell Tech

### Abstract

Serverless computing is an emerging paradigm in which an application’s resource provisioning and scaling are managed by third-party services. Examples include AWS Lambda, Azure Functions, and Google Cloud Functions. Behind these services’ easy-to-use APIs are opaque, complex infrastructure and management ecosystems. Taking on the viewpoint of a serverless

Serverless computing originated as a design pattern for handling low duty-cycle workloads, such as processing in response to infrequent changes to files stored on the cloud. Now it is used as a simple programming model for a variety of applications [14, 22, 42]. Hiding resource management from tenants enables this programming model, but the resulting opacity hinders adoption for many potential users, who have expressed concerns about security in terms of the quality of isolation,

▲ i\_phish\_cats 4 months ago | parent | favorite | on: Serverless Computing: One Step Forward, Two Steps ...

Oh look a bunch of academics telling developers how to do their job, again.

## Contents

- 1 Introduction to Serverless Computing
- 2 Emergence of Serverless Computing
  - 2.1 Contextualizing Serverless Computing
  - 2.2 Attractiveness of Serverless Computing
- 3 Limitations of Today’s Serverless Computing
  - 3.1 Inadequate storage for fine-grained data
  - 3.2 Lack of fine-grained coordination
  - 3.3 Poor performance for standard workloads
  - 3.4 Predictable Performance
- 4 What Serverless Computing Should Be
  - 4.1 Abstraction challenges
  - 4.2 System challenges
  - 4.3 Networking challenges
  - 4.4 Security challenges
  - 4.5 Computer architecture challenges
- 5 Fallacies and Pitfalls

available to the general public, managed as a service.

Despite that potential, we have yet to have seen radical ways. The cloud today is largely a platform for standard enterprise data services. Creative developers need programming frameworks to leverage the cloud’s power.

New computing platforms have typically programming languages and environments: it is difficult to identify the new programming cloud. And whether cause or effect, the result in practice: the majority of cloud services are easier-to-administer clones of legacy enterprise object storage, databases, queuing systems. Multitenancy and administrative simplicity are able goals, and some of the new services have in their own right. But this is, at best, only offered by millions of cores and exabytes of storage.

Recently, public cloud vendors have begun offering interfaces under the banner of serverless. Interest is growing, Google search trends show the term “serverless” recently matched the history of the phrase “Map Reduce” or “MapReduce” also been a significant uptick in attention to it from the research community [13, 6, 27, 14].

This article is published under a Creative Commons license (<http://creativecommons.org/licenses/by/3.0/>), which permits reuse in any medium as well as allowing derivative works to be created from the original work to the author(s) and CIDR 2019.

with highlights including that AWS Lambda adopts a bin-packing-like strategy to maximize VM memory utilization, that severe contention between functions can arise in AWS and Azure, and that Google had bugs that allow customers to use resources for free.

## 1 Introduction

Cloud computing has allowed backend infrastructure maintenance to become increasingly decoupled from application development. Serverless computing (or function-as-a-service, FaaS) is an emerging application deployment architecture that completely hides server management from tenants (hence the name). Tenants receive minimal access to an application’s runtime configuration. This allows tenants to focus on developing their functions — small applications dedicated to specific tasks. A function usually executes in a dedicated *function instance* (a container or other kind of sandbox) with restricted resources such as CPU time and memory. Unlike virtual machines (VMs) in more traditional infrastructure-as-a-service (IaaS) platforms, a function instance will be launched only when the function is invoked and is put to sleep immediately after handling a request. Tenants are charged on a per-invocation basis, without paying for unused and idle resources.

23, 35, 37, 40]; the need to improve application performance [28, 40]; and the ability to shed light on performance [10–12, 29–31]. We made to shed light on tenant and security [33, 34], as, as we will show, fail to

We therefore perform the most in-depth study of resource management and performance isolation to date in three popular serverless computing providers: AWS Lambda, Azure Functions, and Google Cloud Functions (GCF). We first use measurement-driven approaches to partially reverse-engineer the architectures of Lambda and Azure Functions, uncovering many undocumented details. Then, we systematically examine a series of issues related to resource management: how quickly function instances can be launched, function instance placement strategies, function instance reuse, and more. Several security issues are identified and discussed.<sup>1</sup> We further explore how CPU, I/O and network bandwidth are allocated among functions and the ensuing performance implications. Last but not least, we explore whether all resources are properly accounted for, and report on two resource accounting bugs that allow tenants to use extra resources for free. Some highlights of our results include:

- AWS Lambda achieved the best scalability and the lowest coldstart latency (the time to provision a new function instance), followed by GCF. But

<sup>1</sup>We responsibly disclosed our findings to related parties before this paper was made public.



# Industrial Interest in Serverless

The Seattle Times



the guardian

Zillow

BUSTLE



The Coca-Cola Company

SERVERLESS ROBOT

THOMSON REUTERS

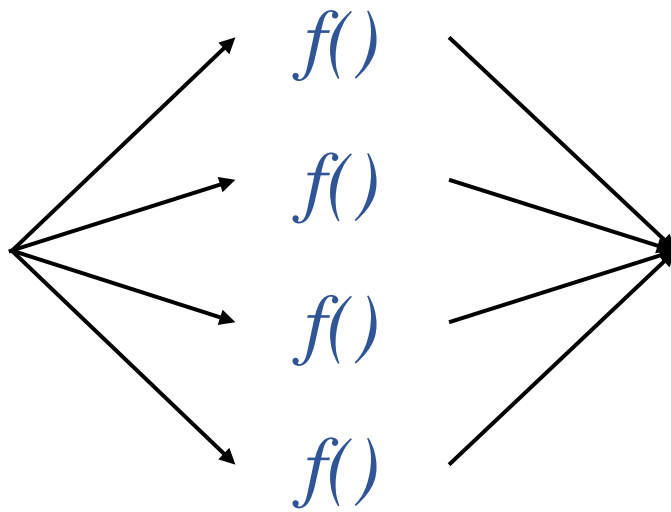
FINRA

AUTODESK

mlbam

NETFLIX

# What is FaaS good at today?



Embarrassingly parallel tasks



Workflow orchestration

Wait! What about...



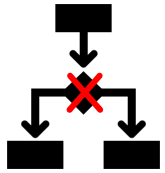
- There are other serverless services, too!
  - e.g., Google Cloud Dataflow, AWS Athena, Snowflake...
- Often referred to as **Backend-as-a-Service** (BaaS)

We're primarily interested in **generality**.

# Limitations on FaaS Today



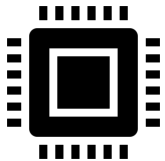
Limited execution lifetimes



No inbound network connections



IO is a bottleneck



No specialized hardware

But that's <sup>NOT</sup> okay: Everything is <sup>NOT</sup> functional!

- Functional programs don't have side effects or mutable state!



AWS Lambda

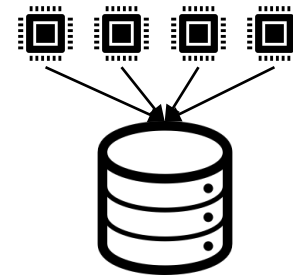
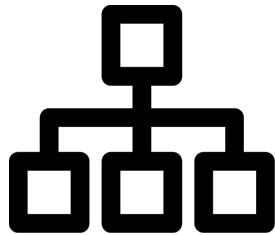


- And it is called *AWS Lambda*

# Dysfunction-as-a-Service

- FaaS is not designed for functional programming because **real applications share state**

$f(g(x))$



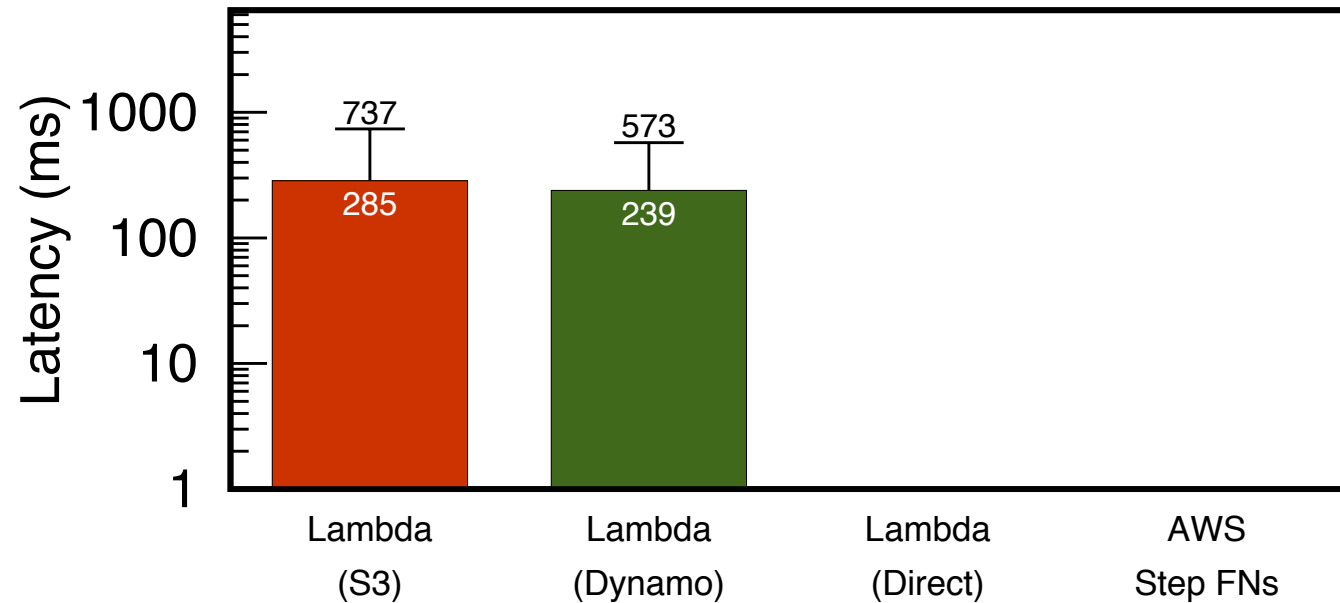
- FaaS is poorly suited for all of these

# Quantifying The Pain of FaaS

How FaaS Disappoints Famous Computer Scientists

# Even Functional Programming is Slow!

$$f(g(x))$$

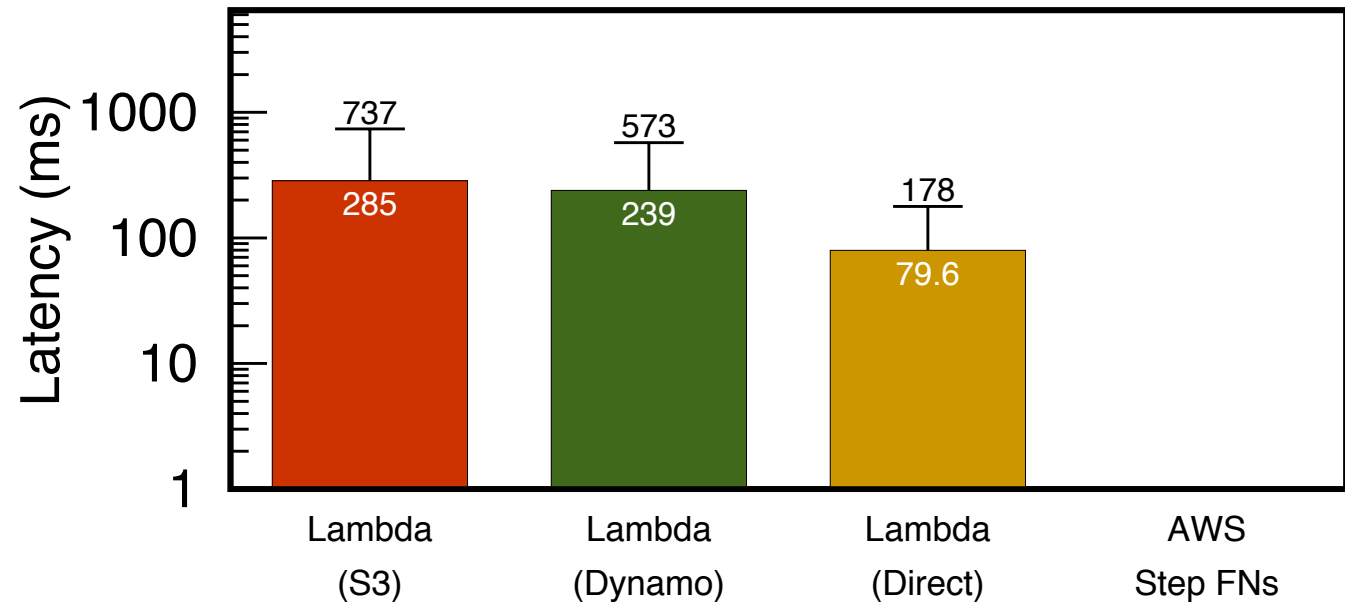


Median and 99<sup>th</sup> percentile latencies for composing two arithmetic functions on AWS Lambda.



# Even Functional Programming is Slow!

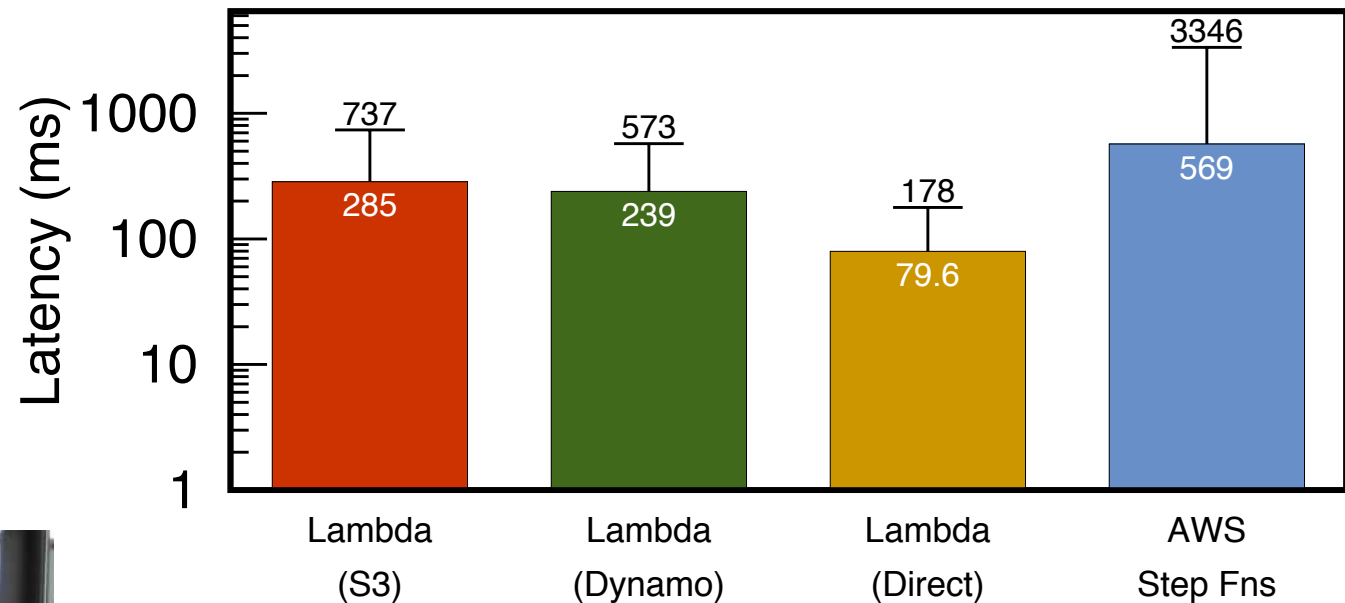
$$f(g(x))$$



Median and 99<sup>th</sup> percentile latencies for composing two arithmetic functions on AWS Lambda.

# Even Functional Programming is Slow!

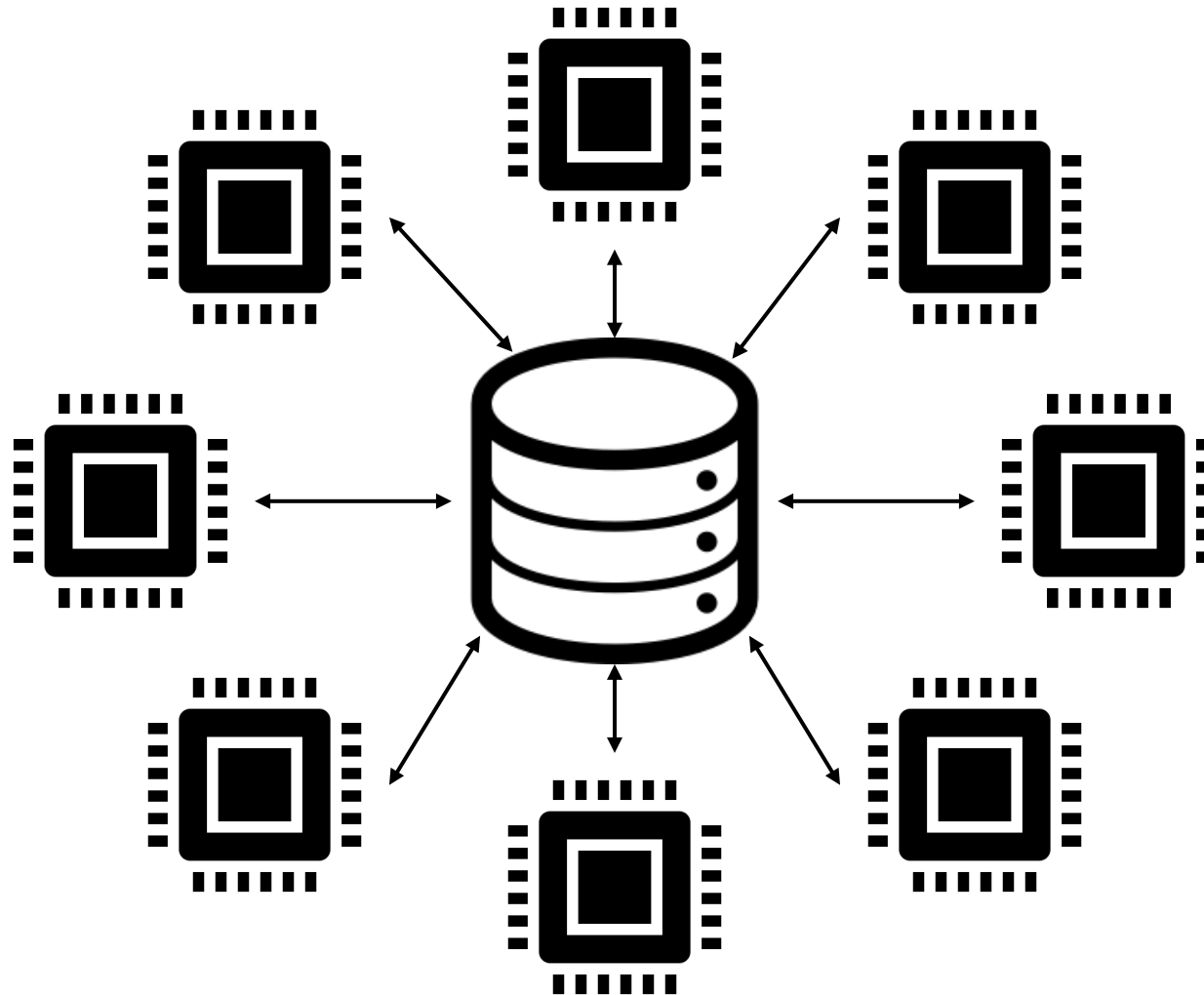
$$f(g(x))$$



Median and 99<sup>th</sup> percentile latencies for composing two arithmetic functions on AWS Lambda.



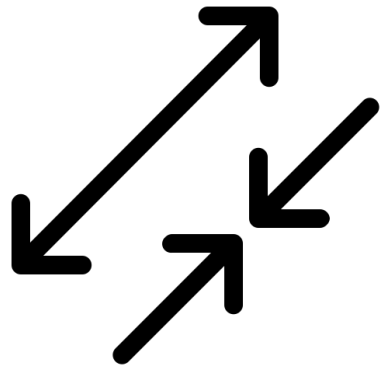
# Shared Mutable State



# Shared Mutable Storage



# ~~QCon~~ Serverless Storage



Autoscaling

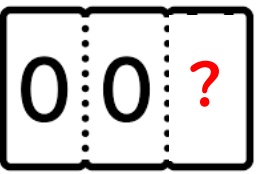
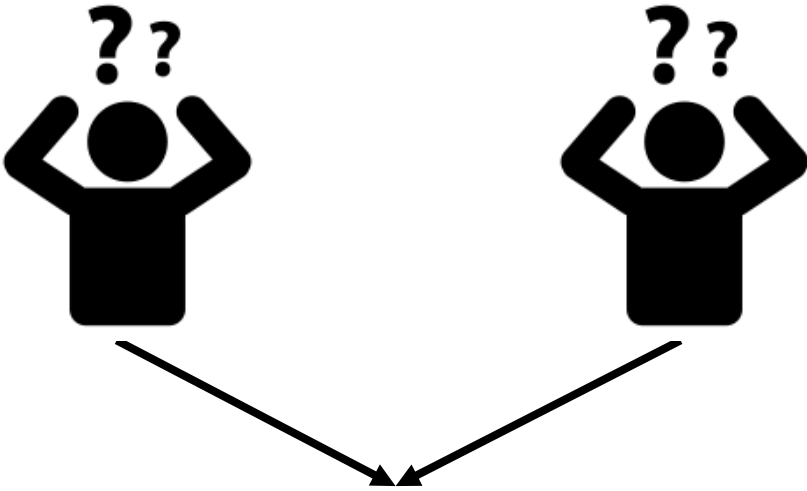


Tradeoff!

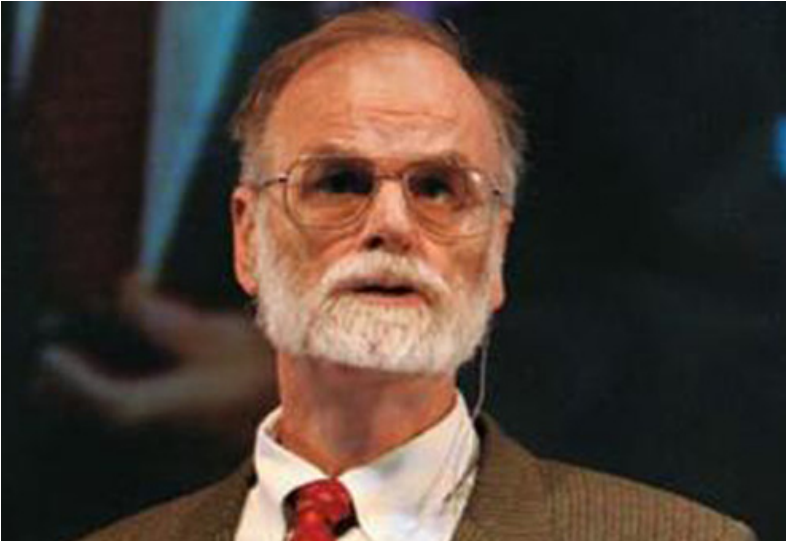


Low Latency

# (In)Consistency Guarantees



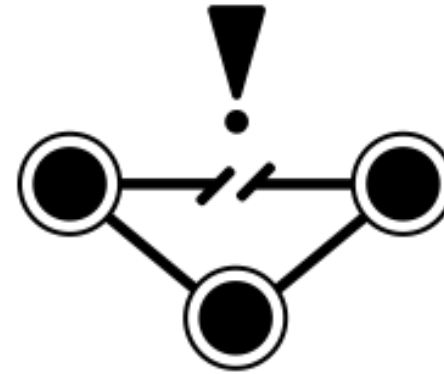
Shared Counter



# No Inbound Network Connections

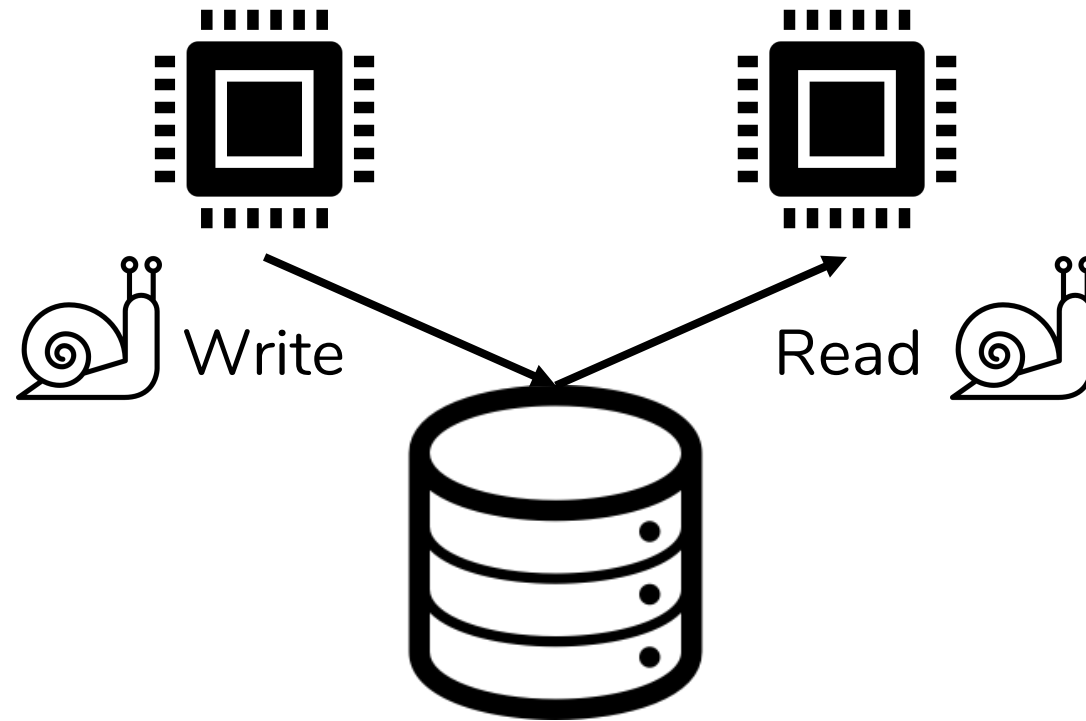


Enables Process  
Migration



Easy Fault  
Tolerance

# Indirect Communication





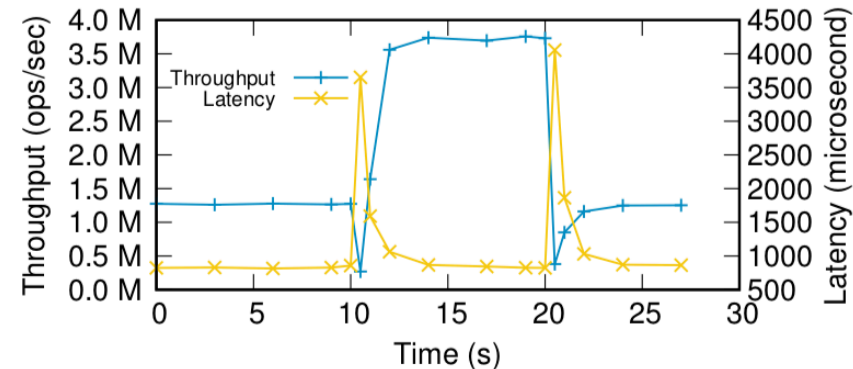
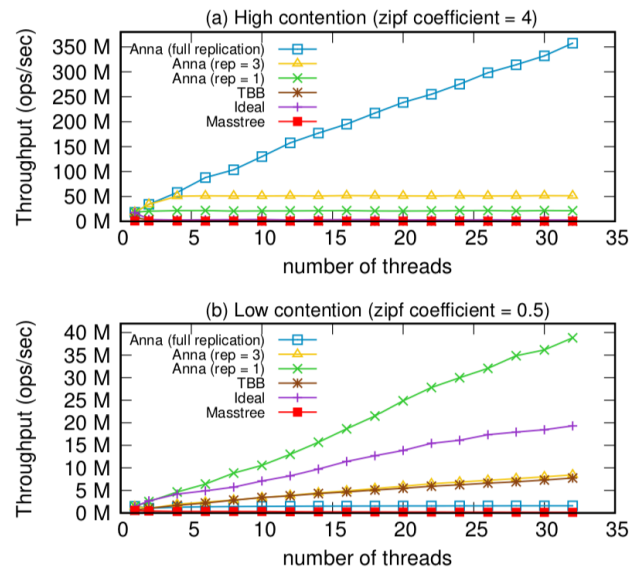
We can  
fix that!



# A Platform for Stateful Serverless Computing

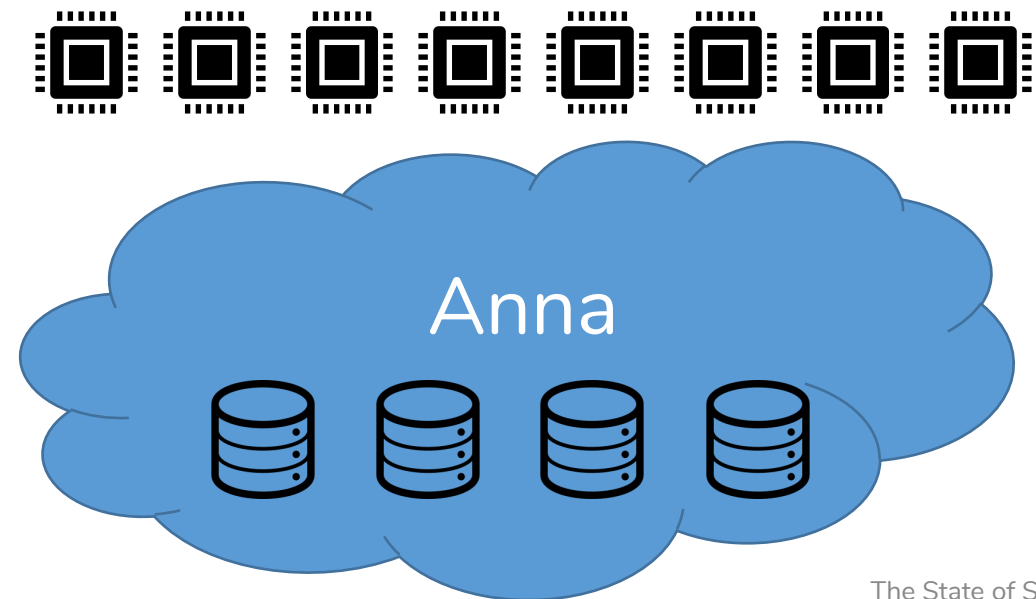
# Background: Anna

- High performance across orders of magnitude in scale
  - ✓ 10x faster than Redis/Cassandra in a geo-distributed deployment
- Autoscaling & cost-efficient
  - ✓ 500x faster than Amazon DynamoDB for the same cost

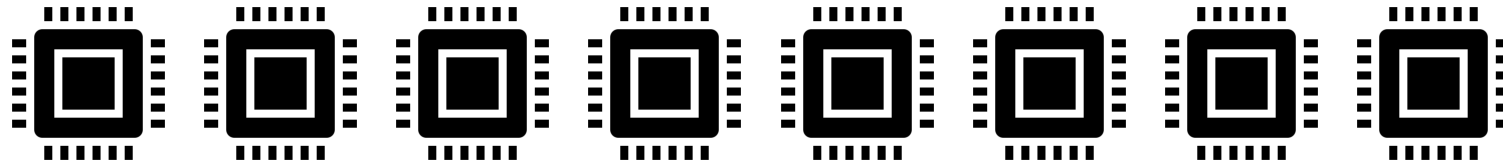


# Fluent: FaaS-over-Anna

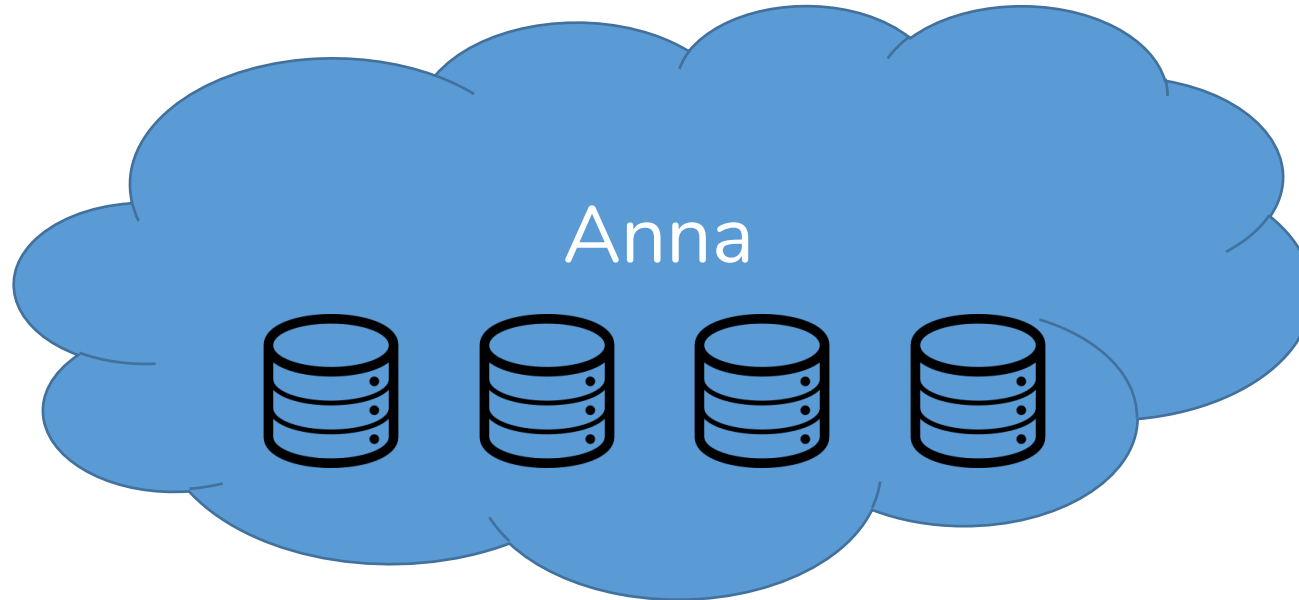
- Maintain disaggregation of compute & state
- Make serverless a viable option for stateful applications
- Use Anna for both storage and communication



# Fluent: FaaS-over-Anna

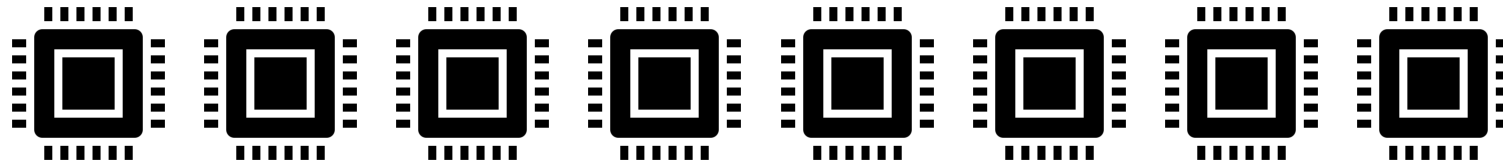


Network  
Boundary

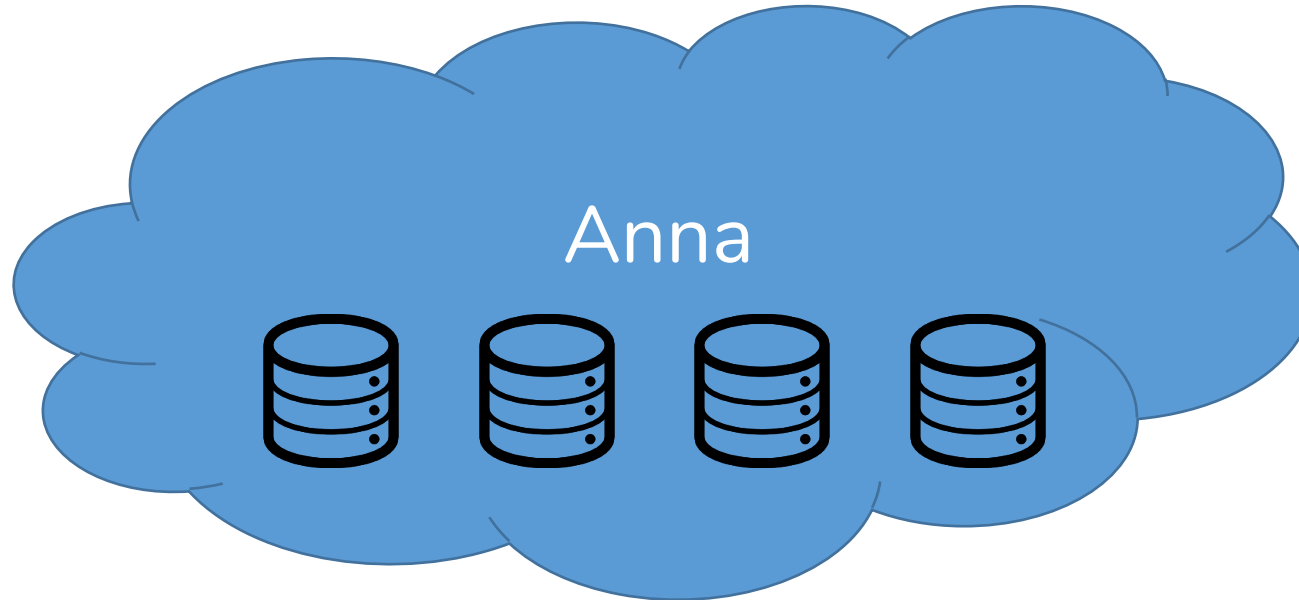


# Logical disaggregation with physical colocation

# Fluent: FaaS-over-Anna

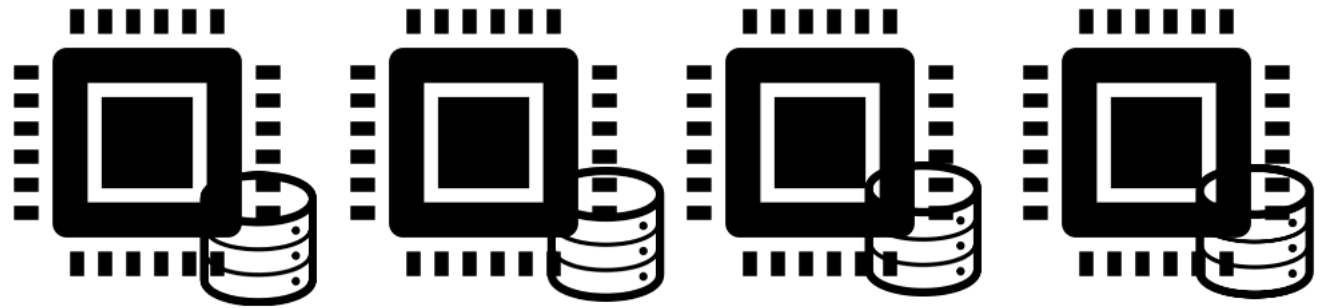


Network  
Boundary



# Key Idea: Caching

- Enable low-latency data access by caching data close to code execution
- Communication (and composition) is achieved via a fast-path on top of KVS puts and gets





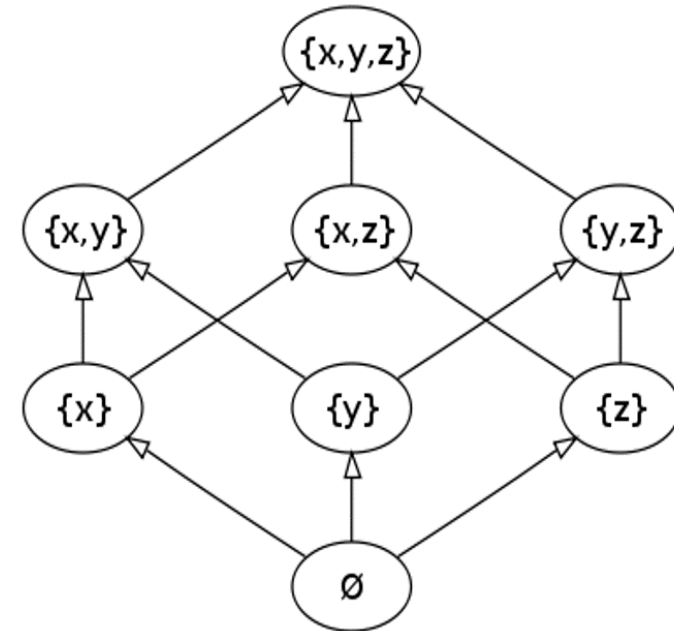
# Challenge: Cache Consistency

- tl;dr: we can provide a variety of coordination-avoiding consistency modes – which is better than S3 or DynamoDB!
- This is done by encapsulating program state in lattices



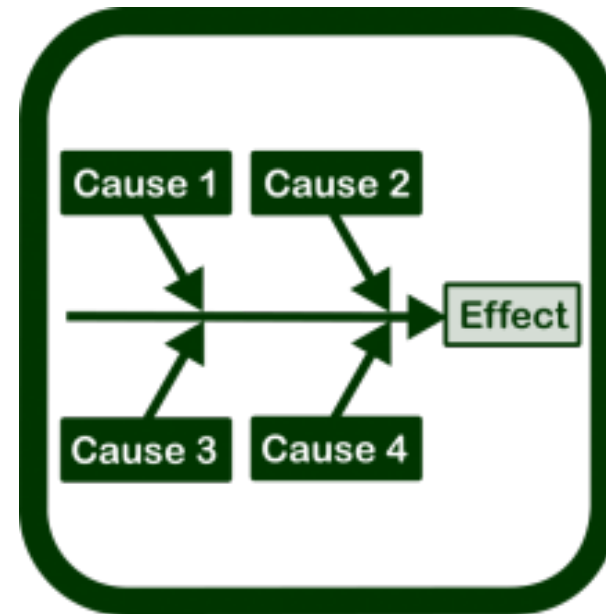
# Lattice

- Data structure that accepts incoming update in a way that is associative, commutative, and idempotent (ACI).
- Achieves eventual replica convergence



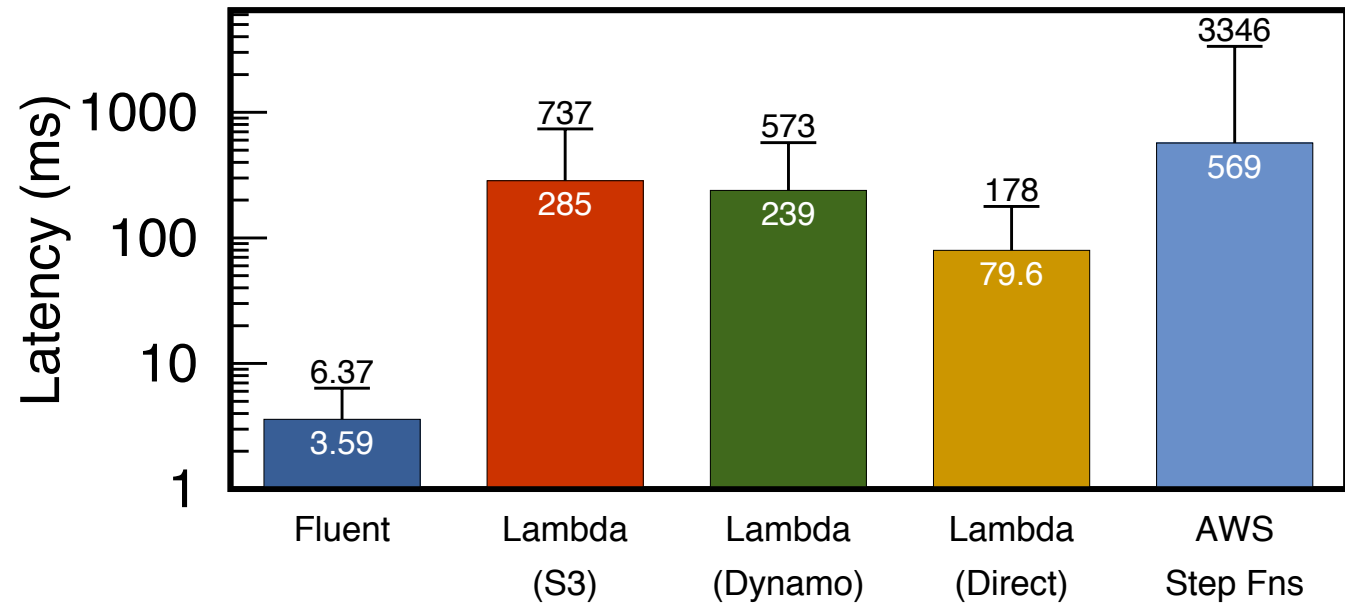
# Causal Consistency

- Strongest consistency level that doesn't require coordination
- Causally-related updates will be revealed in an order that respects causality
- In addition, guarantee
  - Repeatable read
  - Atomic visibility



# Function Composition, Revisited

$$f(g(x))$$

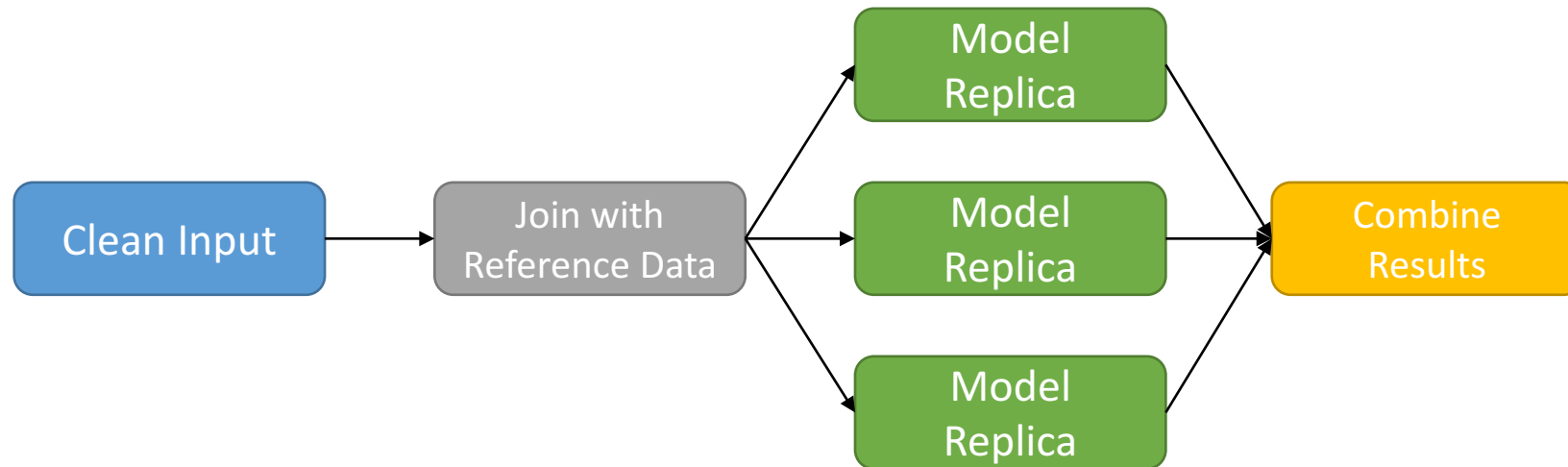


Median and 99<sup>th</sup> percentile latencies for composing two arithmetic functions on AWS Lambda and Fluent.

# Case Study: Prediction Serving

# Prediction Serving

- Generate predictions from pretrained machine learning models



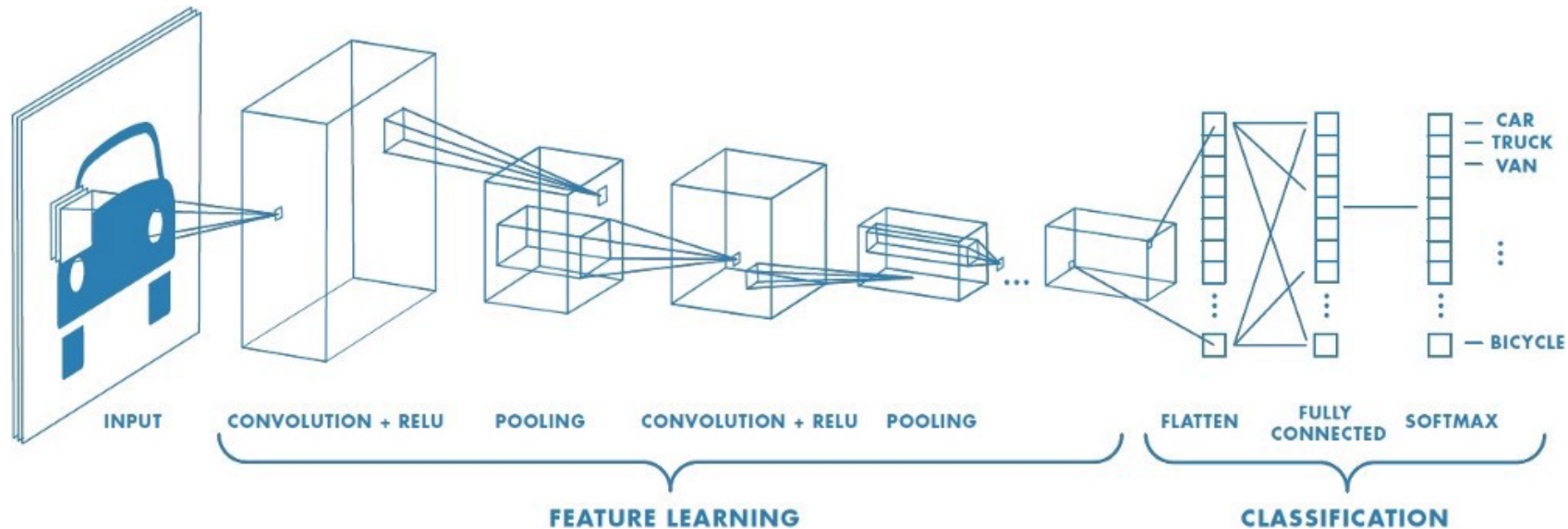
- At first blush, a great fit for serverless infrastructure

# 737 FLIGHT SIMULATION PIPELINE WITH AMAZON SAGEMAKER



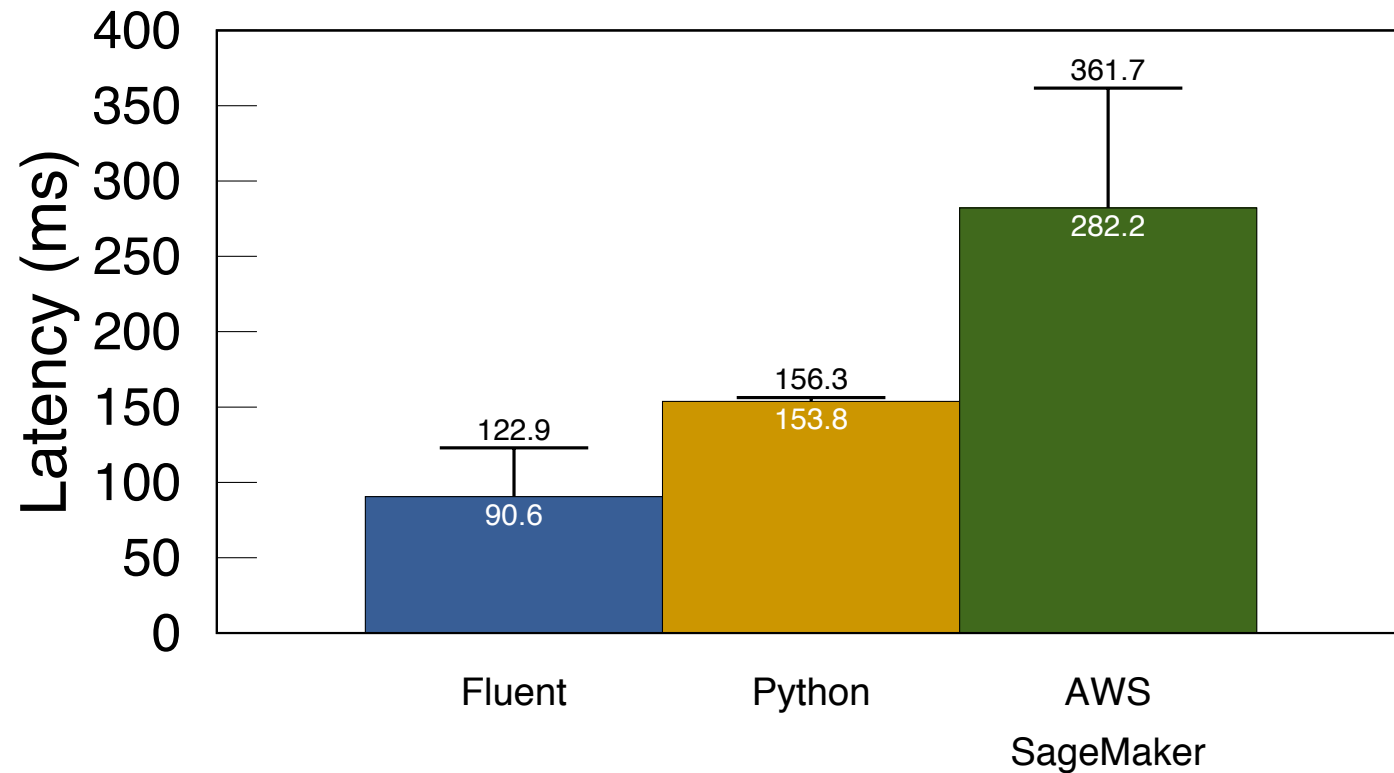
# Background: SqueezeNet

- State-of-the-art image classification model (developed at Berkeley!)





# Prediction Serving

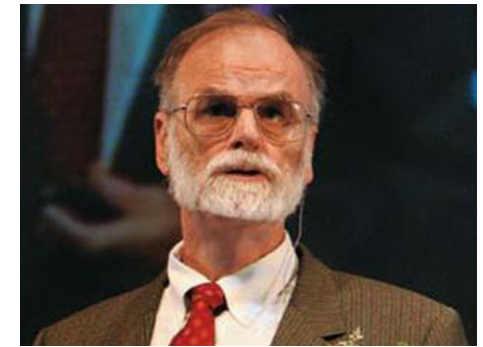


Median and 99<sup>th</sup> percentile latencies for SqueezeNet on Fluent and AWS SageMaker.

# The Future of Cloud Programming

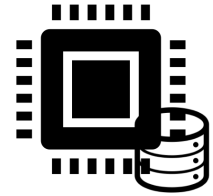
# Looking Back: Disappointed Computer Scientists

- Functional programming is slow
- Communication through slow storage
- Poor consistency guarantees

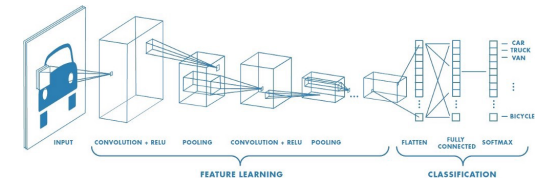


# Making FaaS Functional

- Embrace state
- Easy things become better
- Hard things become easy



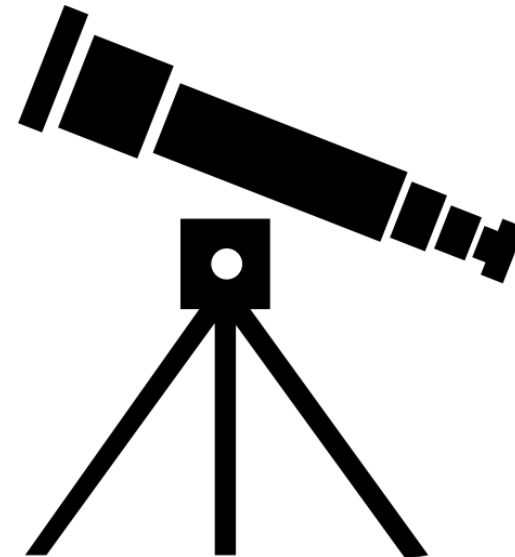
$$f(g(x))$$



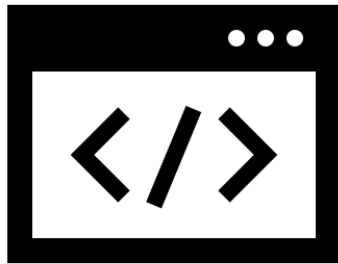
A step on our road towards a **programmable cloud**.

# Our Vision

- Serverless will change the way that we write software and the way that programming infrastructure works
- Cloud-native [programming models](#)
  - Enable users to take advantage of millions of cores and petabytes of RAM



# Moving Forward from FaaS

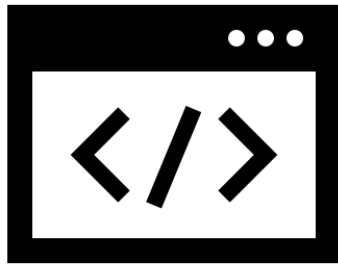


Building  
Developer  
Tools

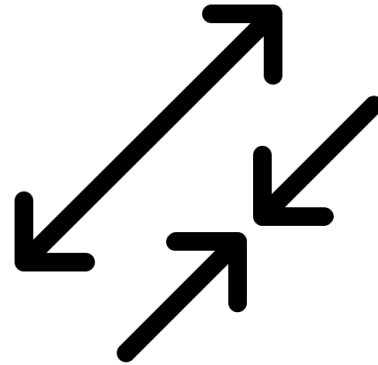
UC Berkeley: CS 61A  
Fall 2018



# Moving Forward from FaaS



Building  
Developer  
Tools



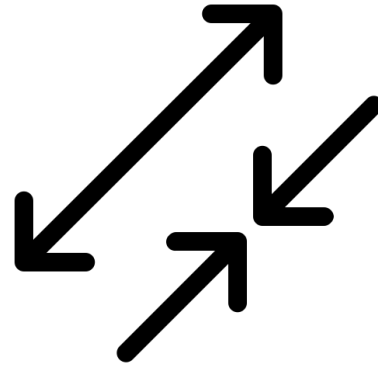
Developing  
Autoscaling  
Policy



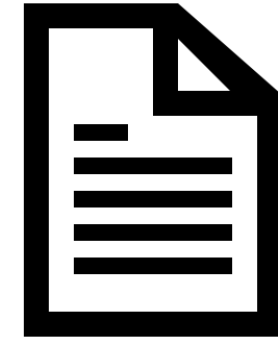
# Moving Forward from FaaS



Building  
Developer  
Tools

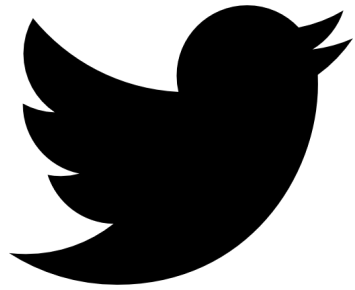


Developing  
Autoscaling  
Policy

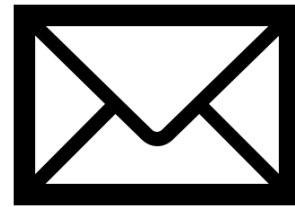


Designing  
SLOs & SLAs

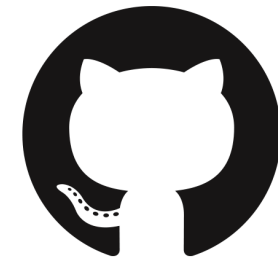
# Thanks!



[@cgwu0530](https://twitter.com/cgwu0530)



[cgwu@berkeley.edu](mailto:cgwu@berkeley.edu)



[fluent-project/fluent](https://github.com/fluent-project/fluent)