# 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 ☁️!

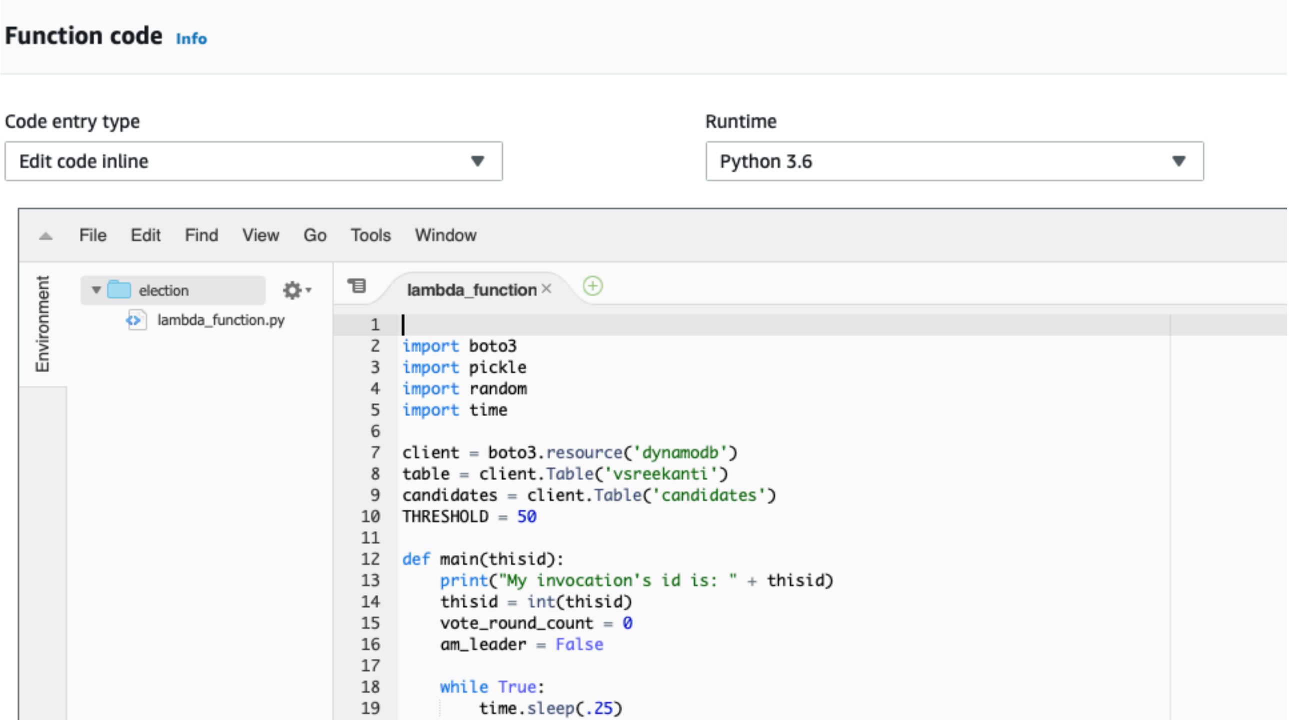
# Background: Serverless Computing

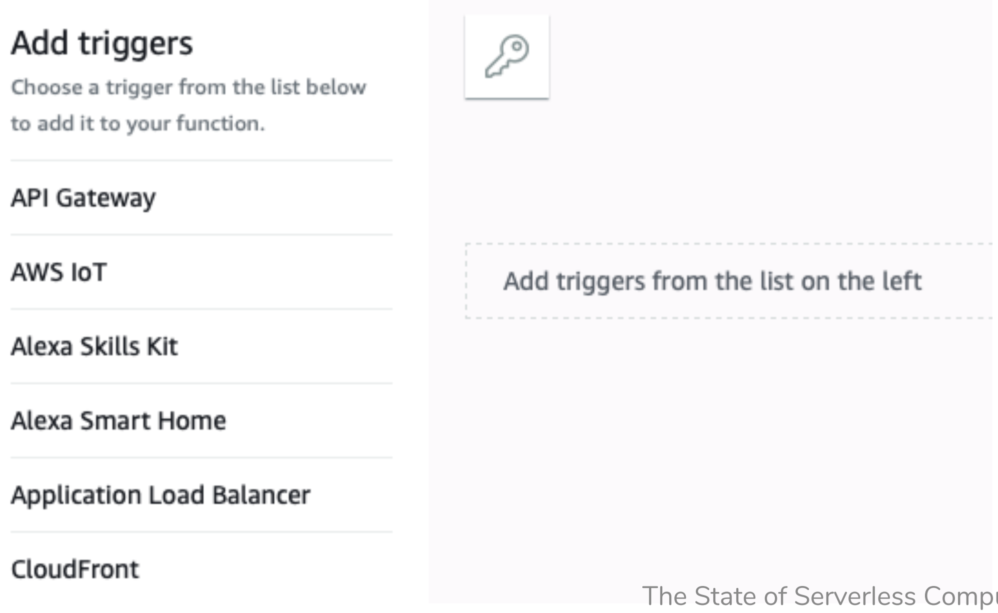
# What is serverless computing?

*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

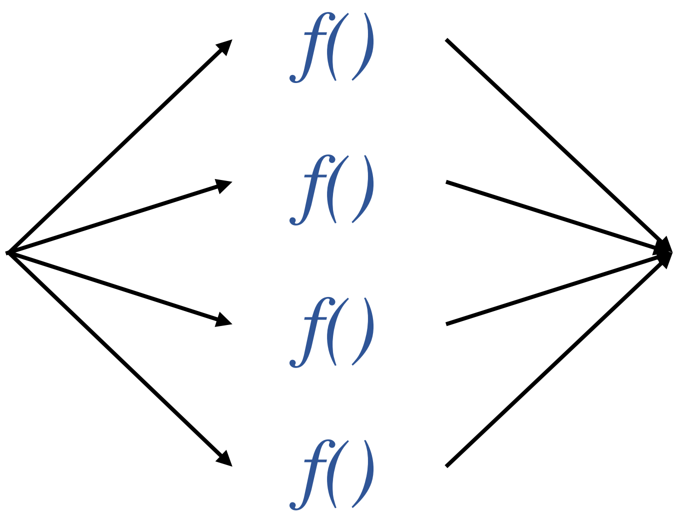


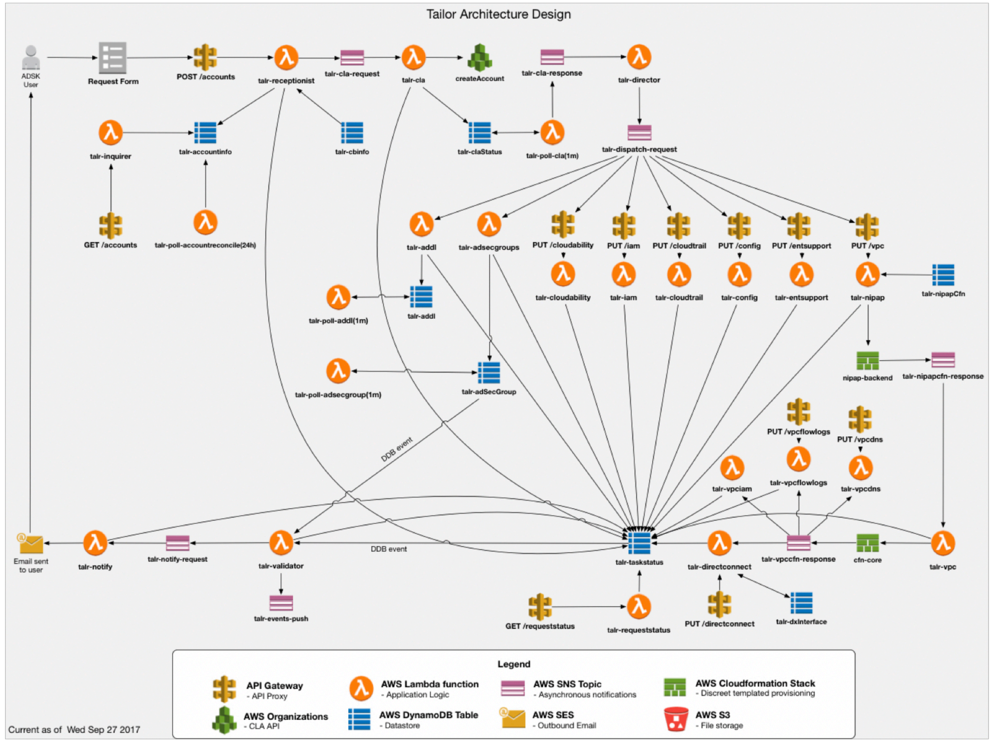
# Academic Interest in Serverless

# Industrial Interest in Serverless

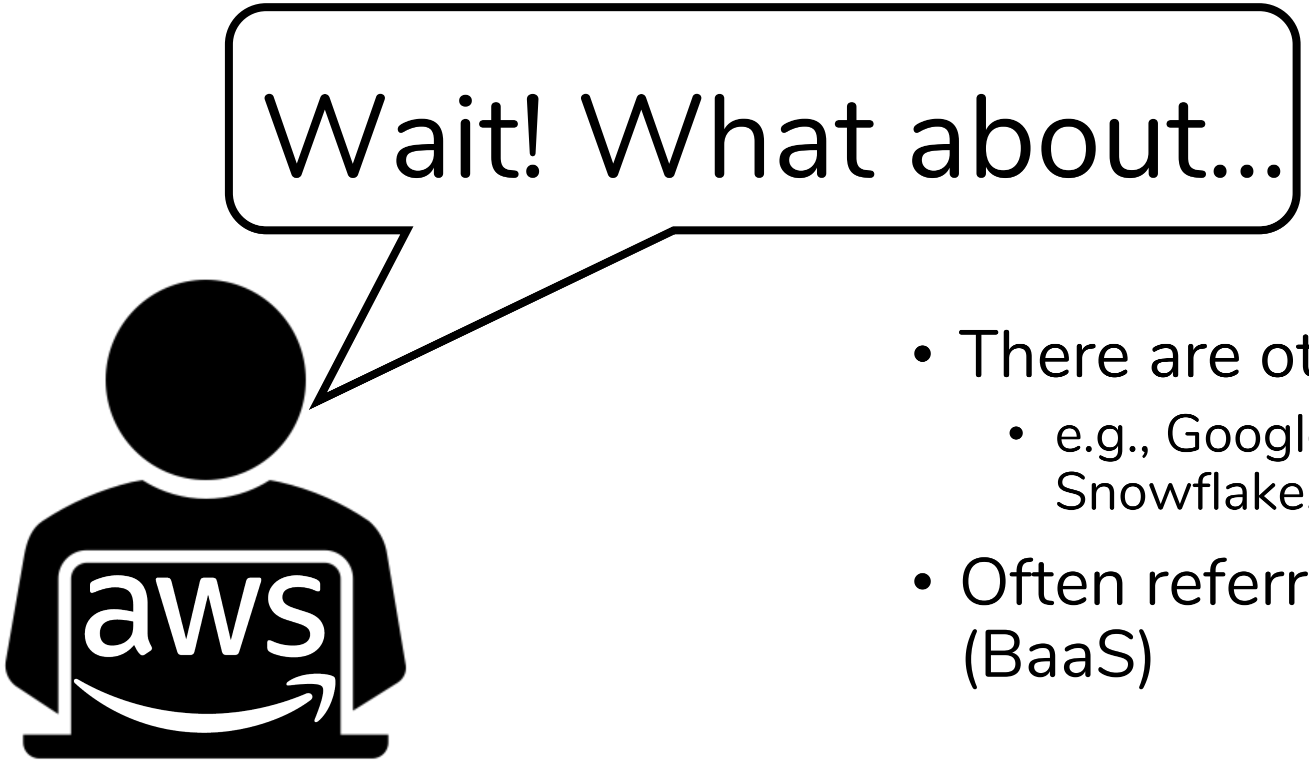
# What is FaaS good at today?




Embarrassingly parallel tasks




Workflow orchestration




- 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 NOT okay: Everything is NOT functional!

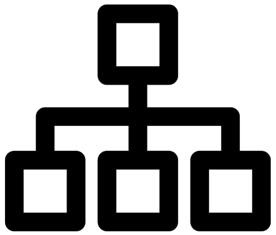
- Functional programs don't have side effects or mutable state!

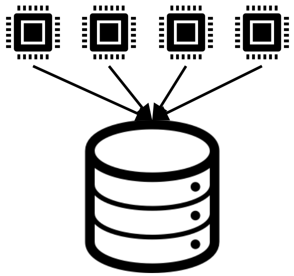
- And it *is called* AWS Lambda

# Dysfunction-as-a-Service

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

$$f(g(x))$$





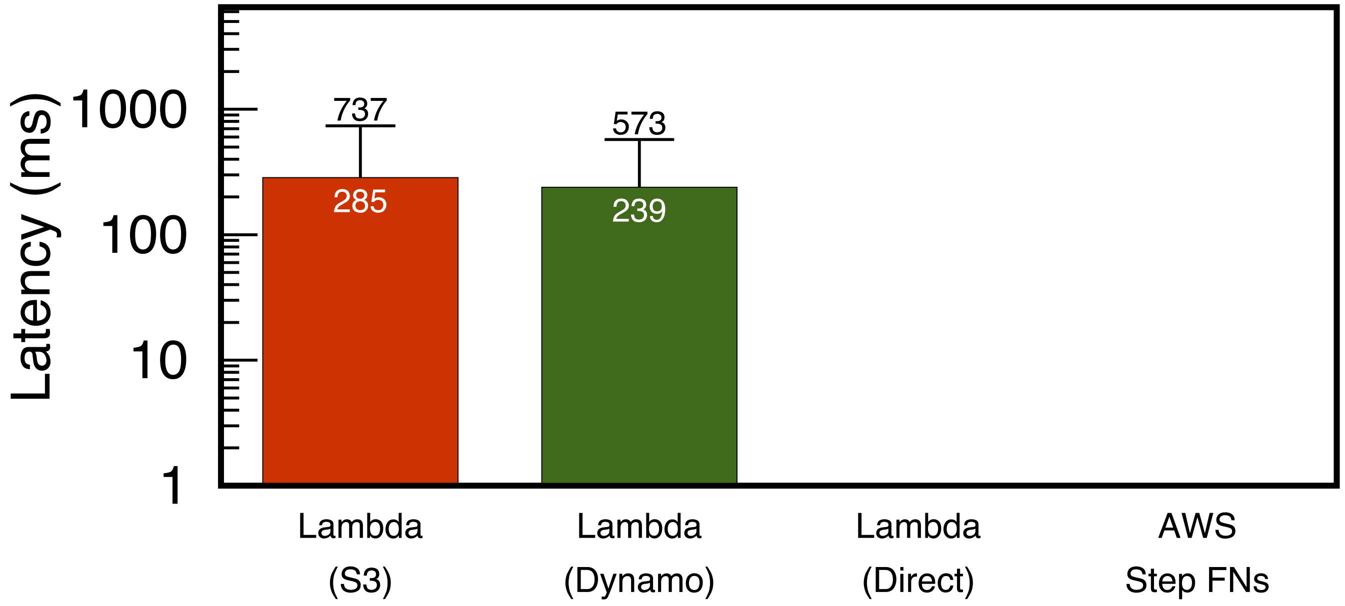
- FaaS is poorly suited for all of these

# Quantifying The Pain of FaaS

## How FaaS Disappoints Famous Computer Scientists

# Even Functional Programming is Slow!

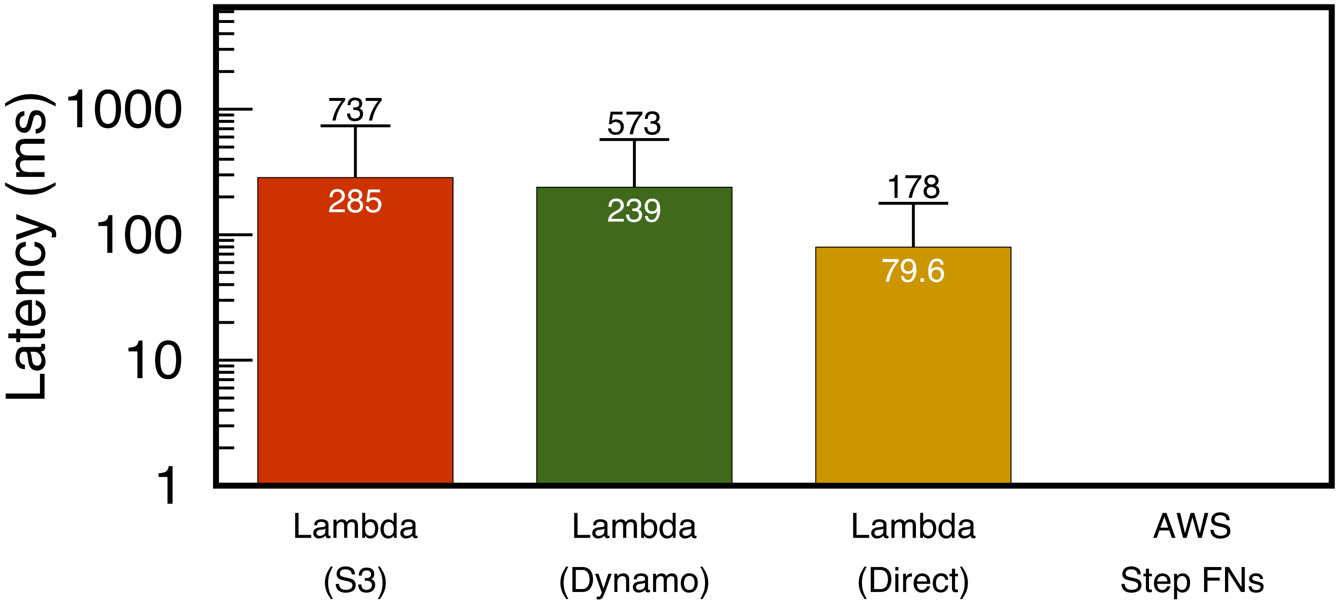
$$f(g(x))$$




Median and 99th percentile latencies for composing two arithmetic functions on AWS Lambda.

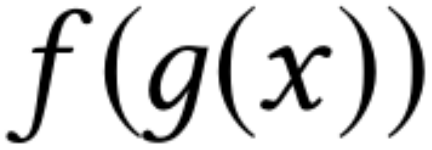
# Even Functional Programming is Slow!

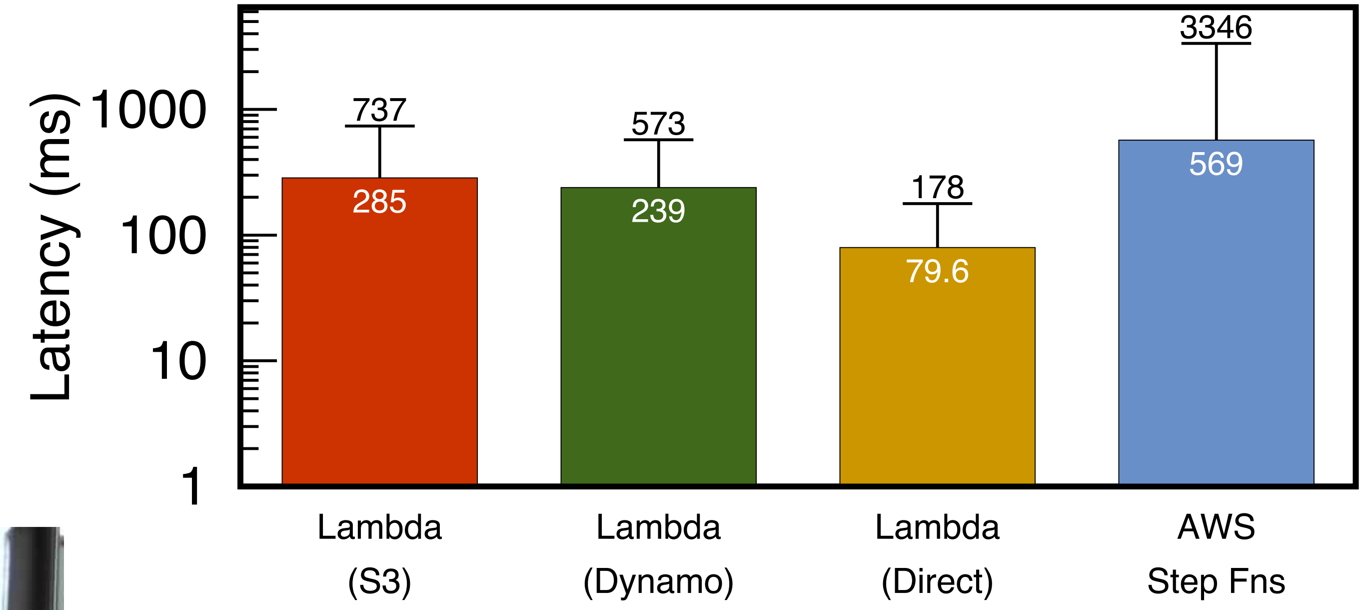
$$f(g(x))$$




Median and $99^{th}$ percentile latencies for composing two arithmetic functions on AWS Lambda.

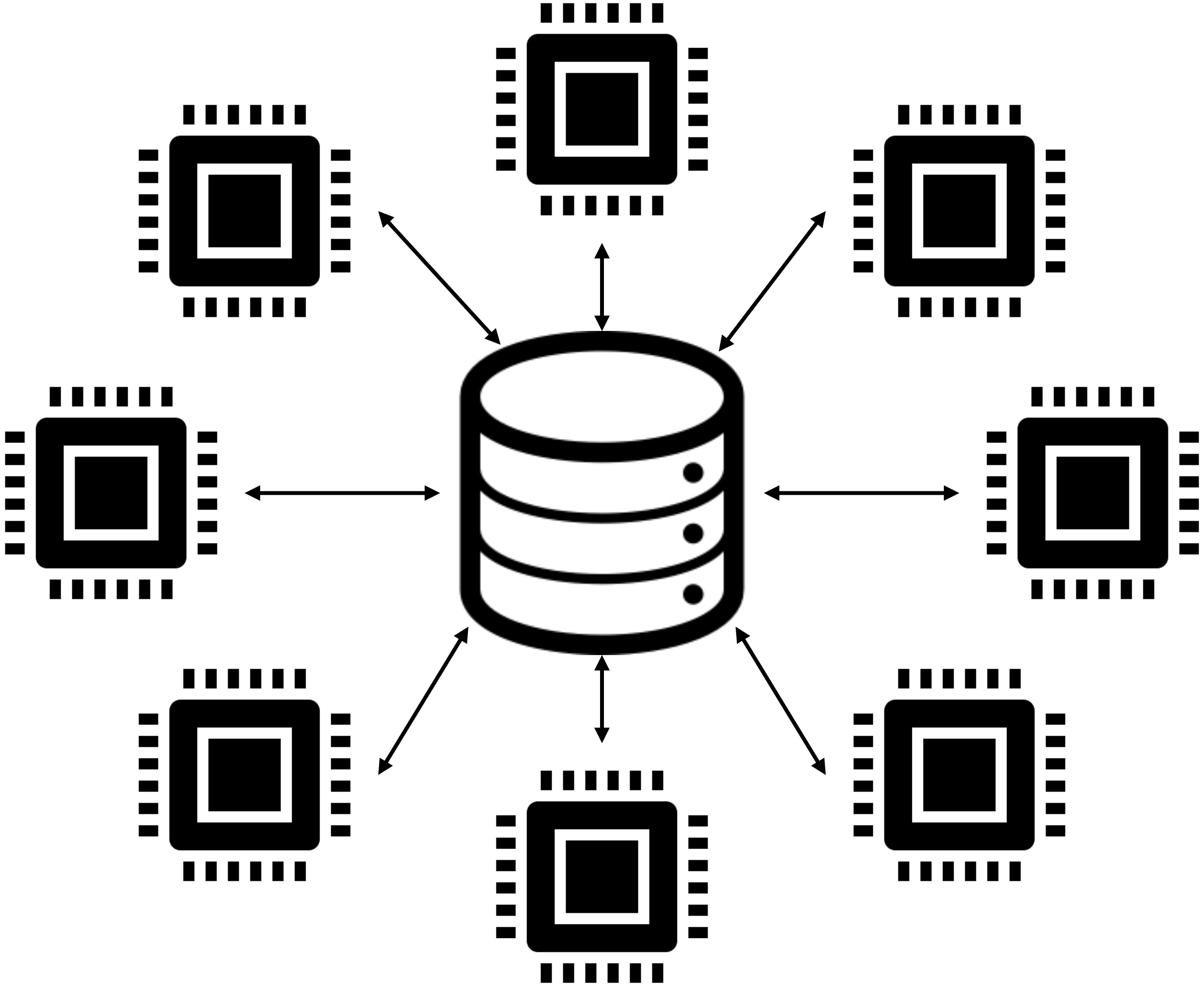
# Even Functional Programming is Slow!



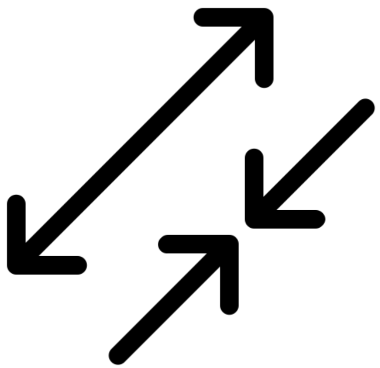

Median and 99th percentile latencies for composing two arithmetic functions on AWS Lambda.

# Shared Mutable State



# Shared Mutable Storage

# Serverless Storage

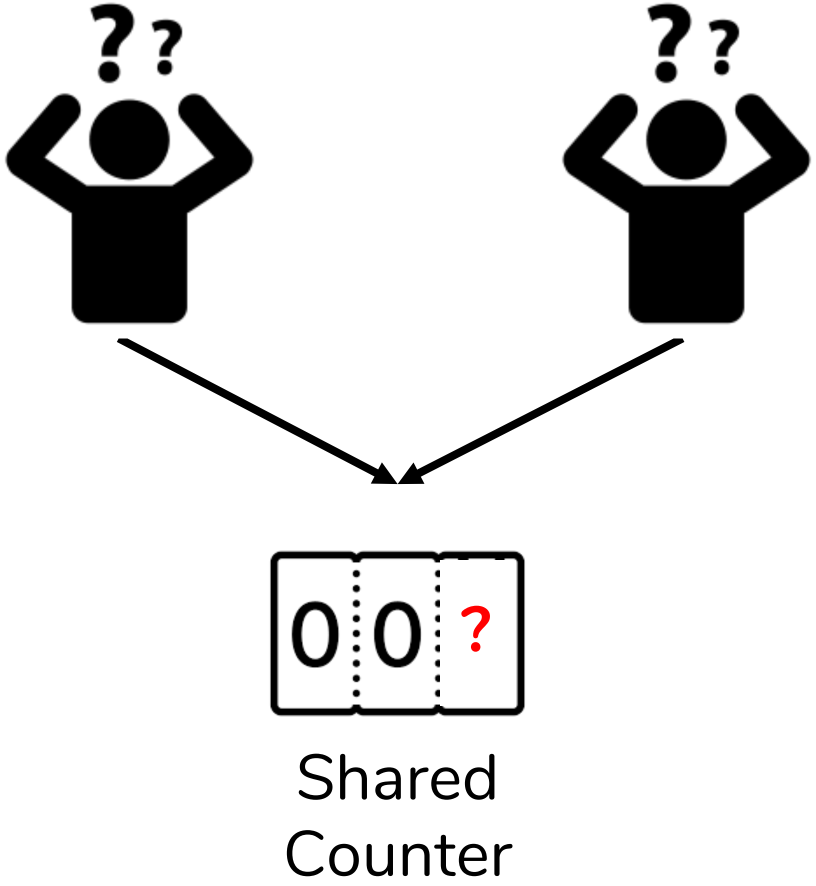


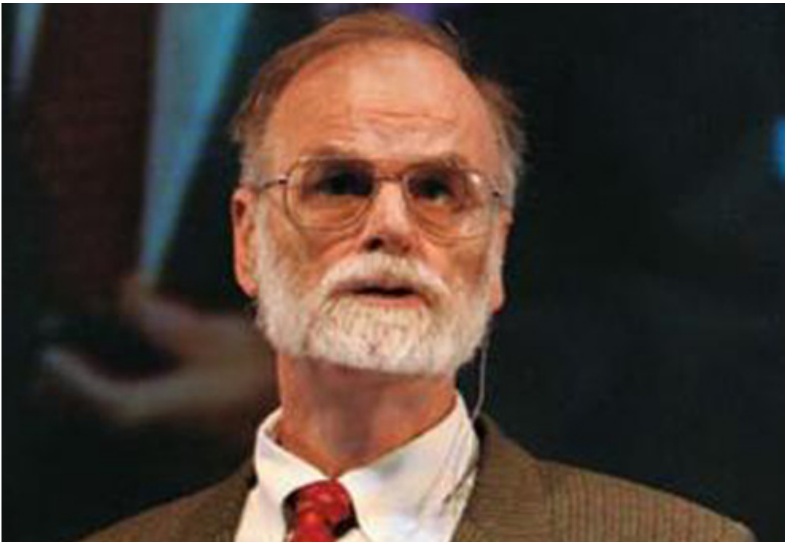
Autoscaling

Tradeoff!

Low Latency

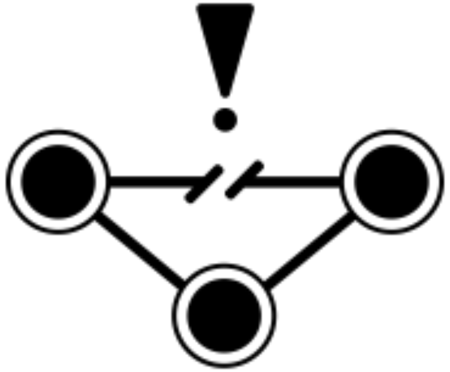
# (In)Consistency Guarantees



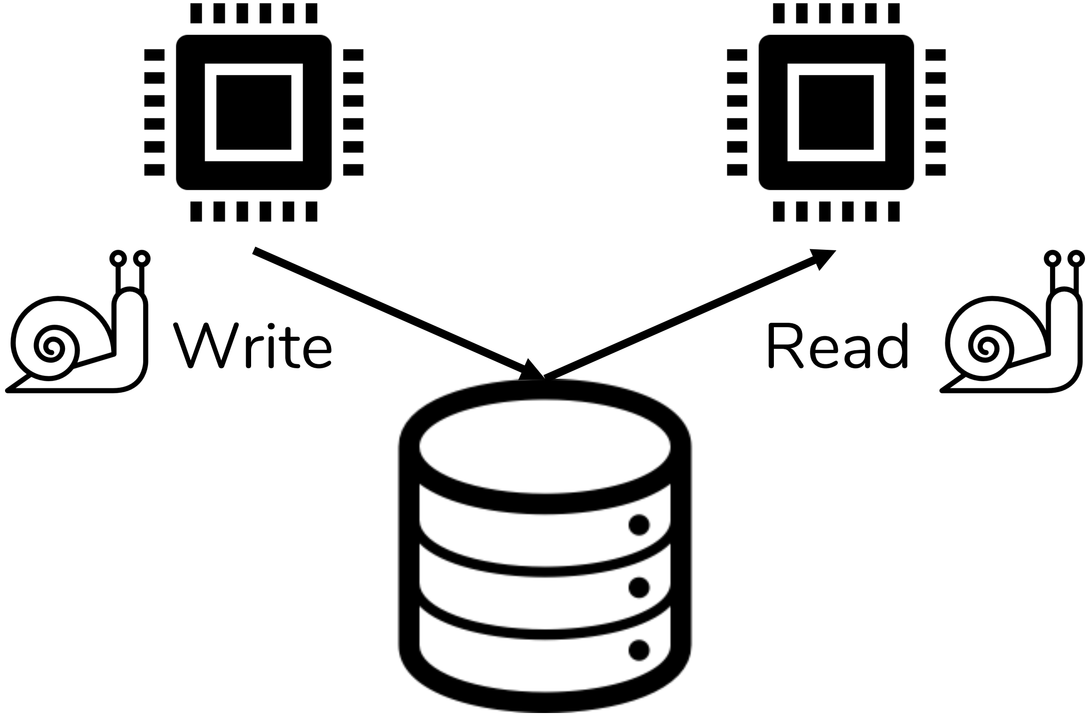
# No Inbound Network Connections

Enables Process
Migration



Easy Fault
Tolerance

# Indirect Communication

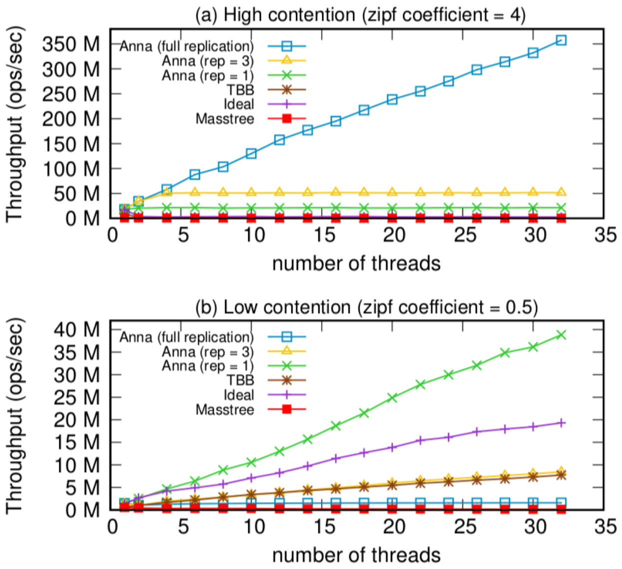


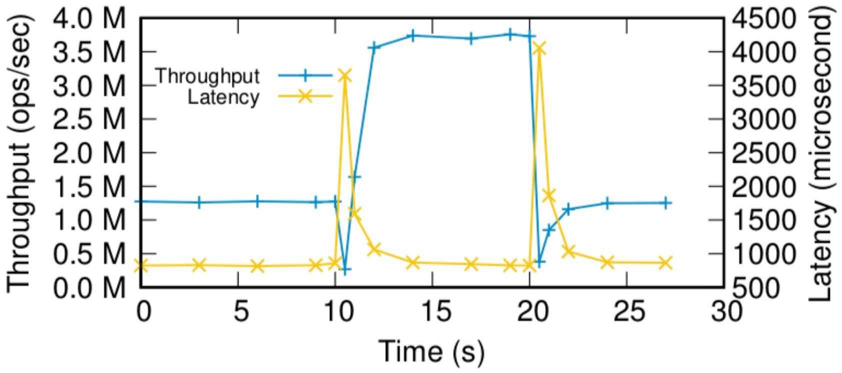

# We can fix that!

# 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

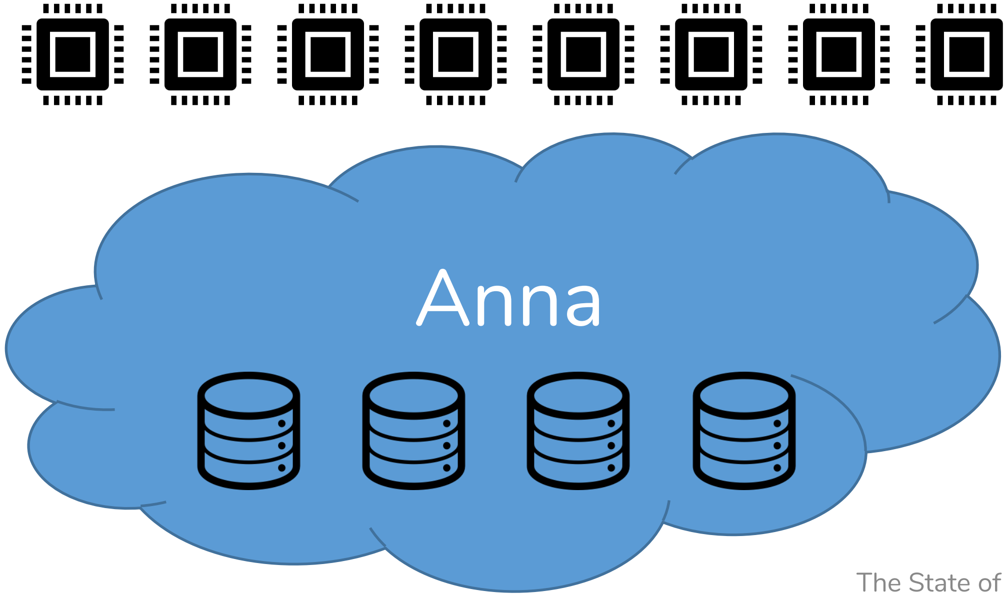


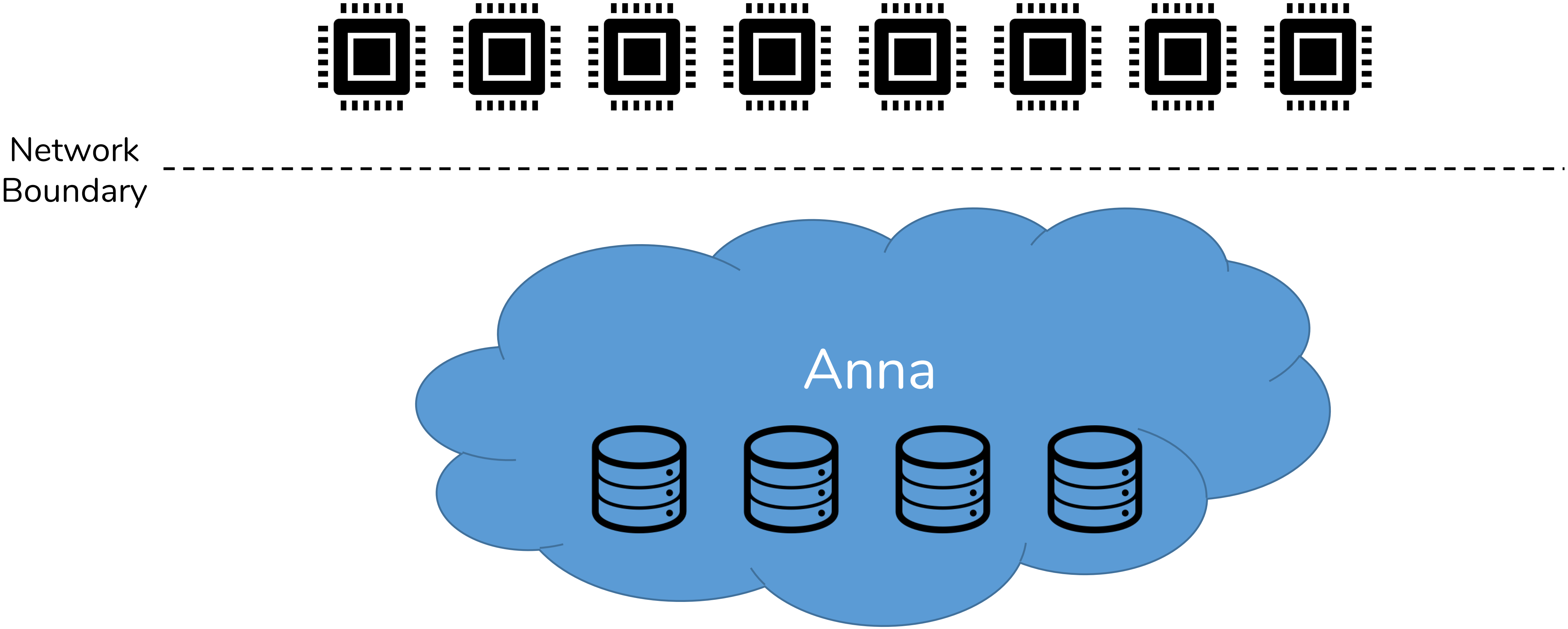

Chenggang Wu, Jose Faleiro, Yihan Lin, and Joseph M. Hellerstein. "Anna: A KVS for Any Scale." *IEEE Transactions on Knowledge and Data Engineering* (2019).
Chenggang Wu, Vikram Sreekanti, and Joseph M. Hellerstein. "Autoscaling Tiered Cloud Storage in Anna." *Proceedings of the VLDB Endowment* 12, no. 6 (2019): 624-638.

# Fluent: FaaS-over-Anna

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

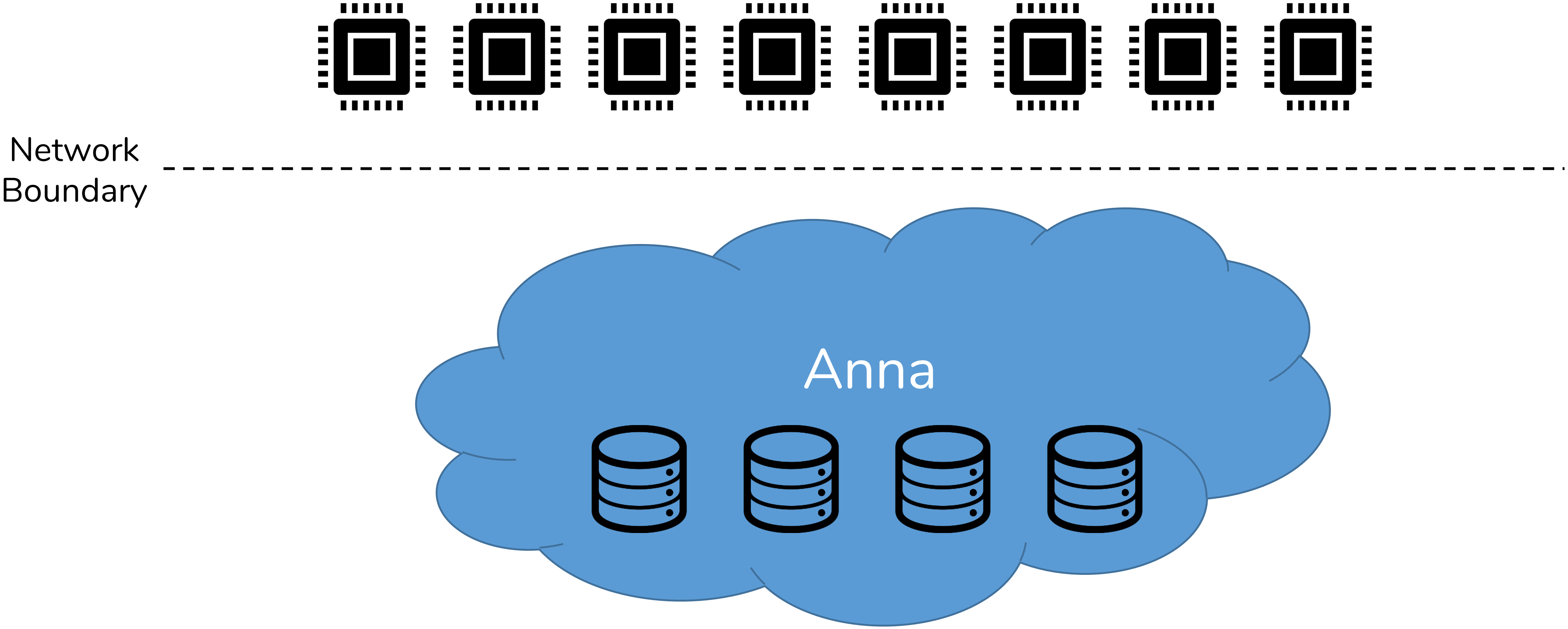


# Fluent: FaaS-over-Anna



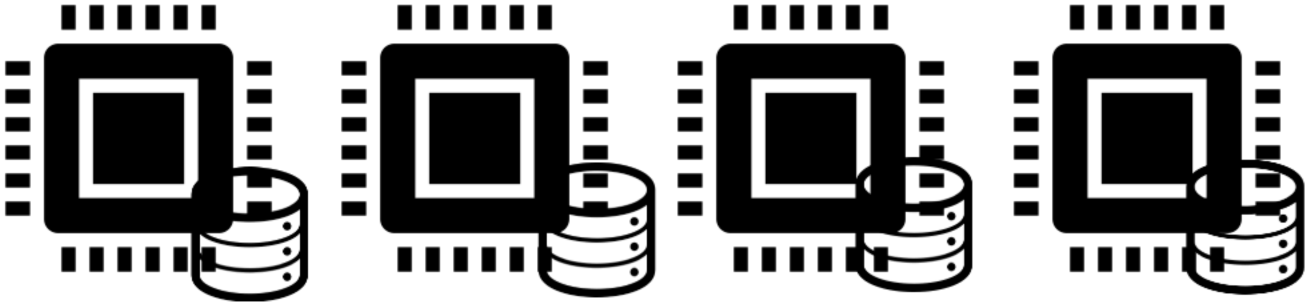
# Logical disaggregation with physical colocation

# Fluent: FaaS-over-Anna



# 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

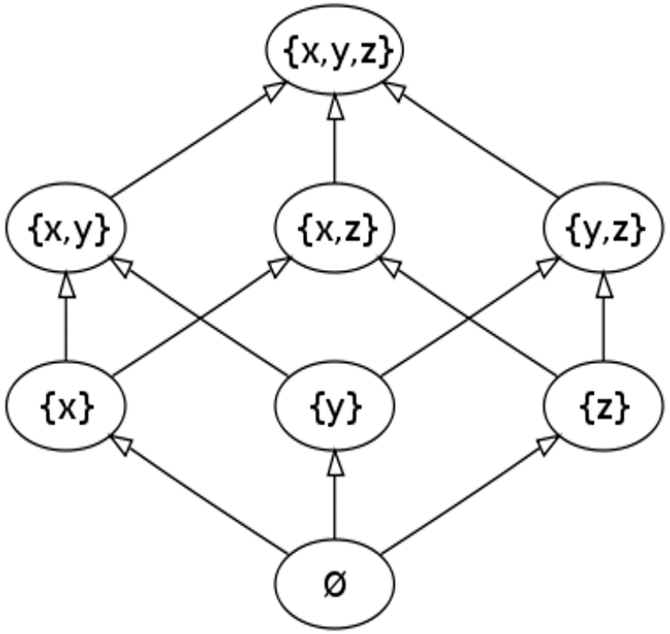


# Challenge: Cache Consistency

- 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

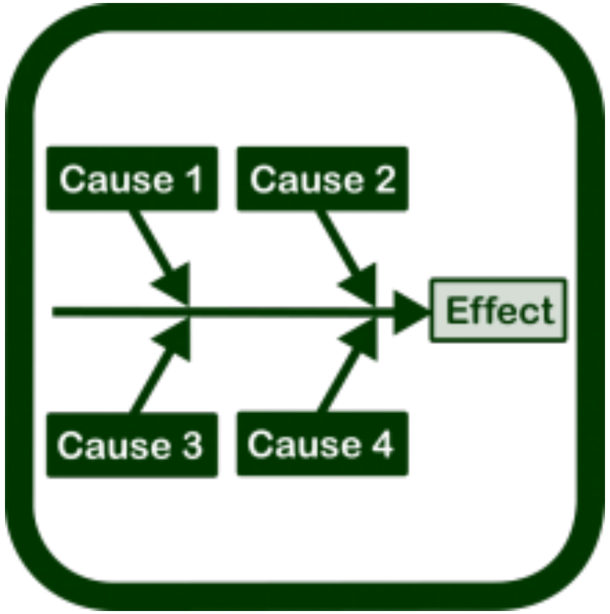
# Lattice

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



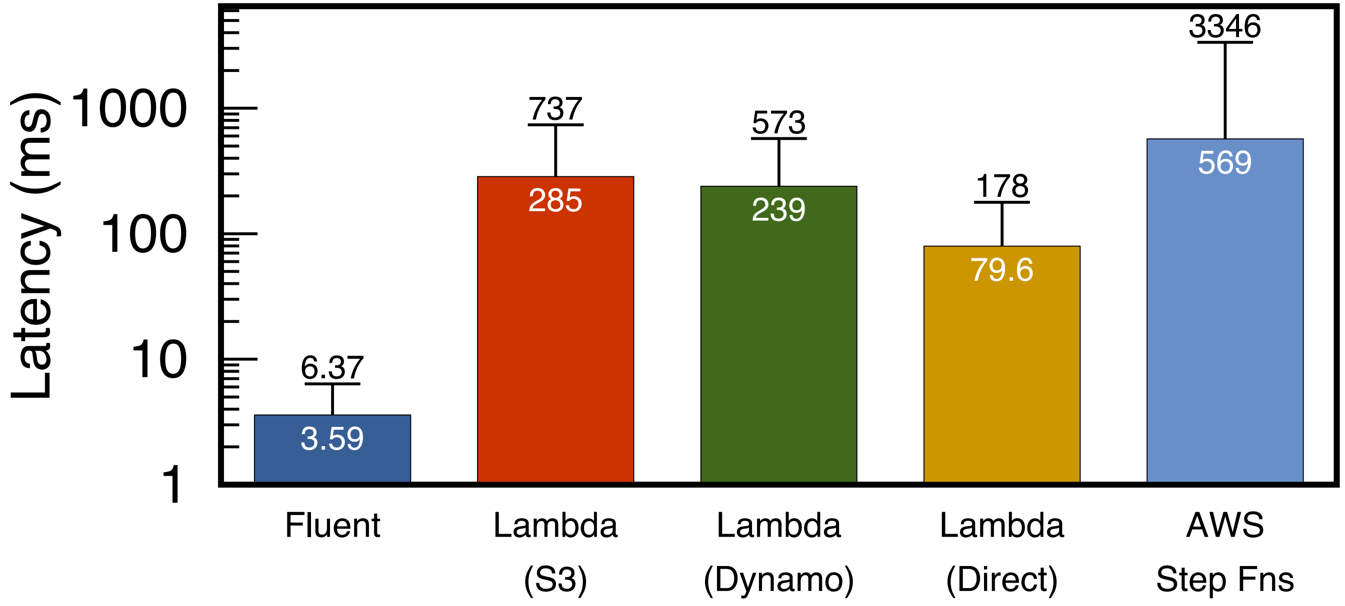
# Causal Consistency

- 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

$$f(g(x))$$

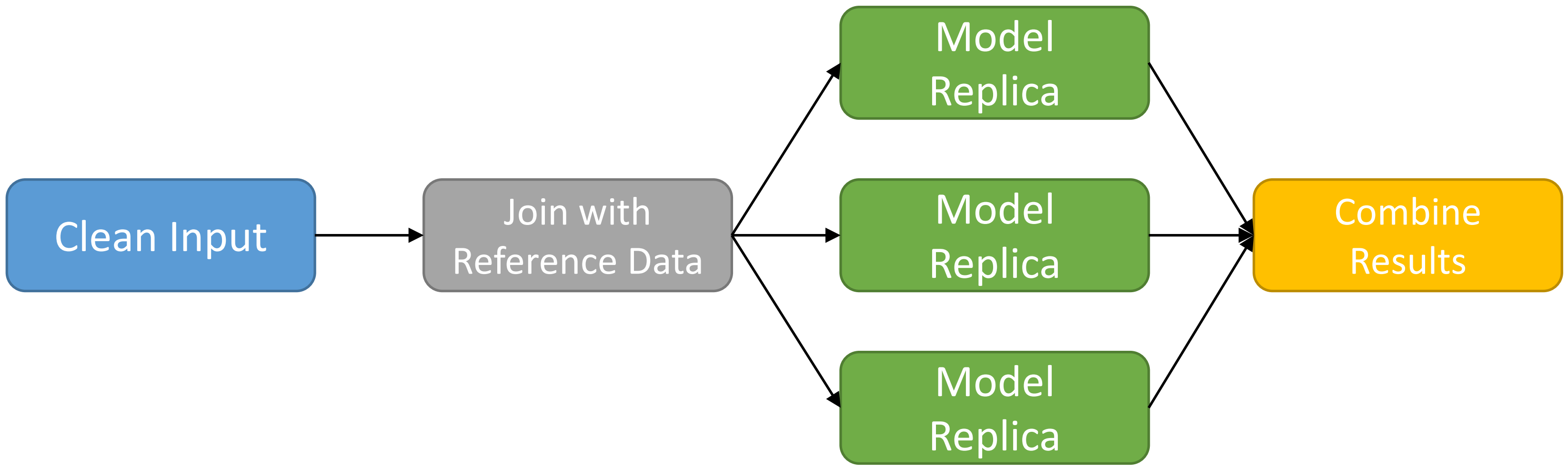



Median and 99th percentile latencies for composing two arithmetic functions on AWS Lambda and Fluent.

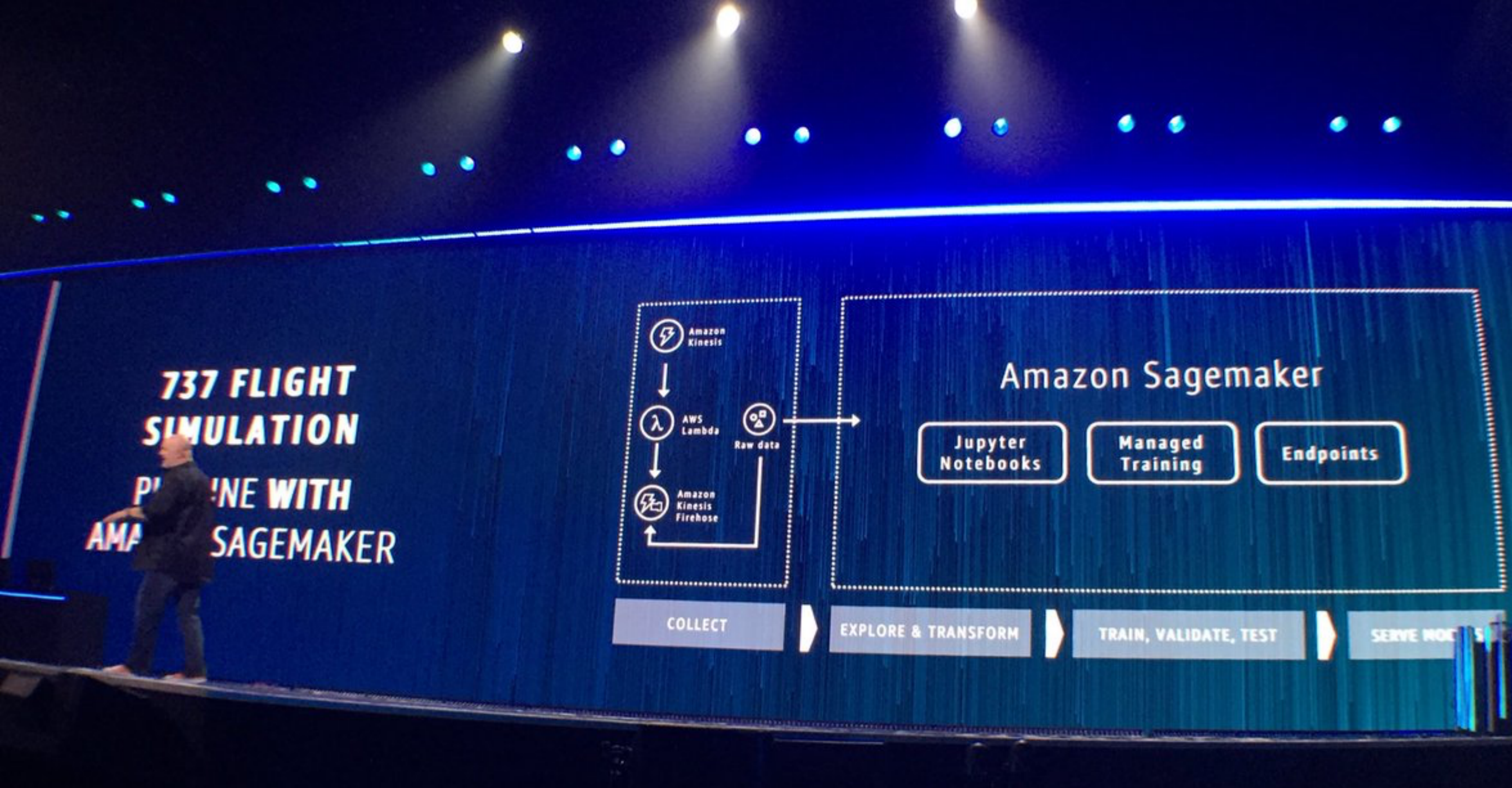
# Case Study: Prediction Serving

# Prediction Serving

- Generate predictions from pretrained machine learning models

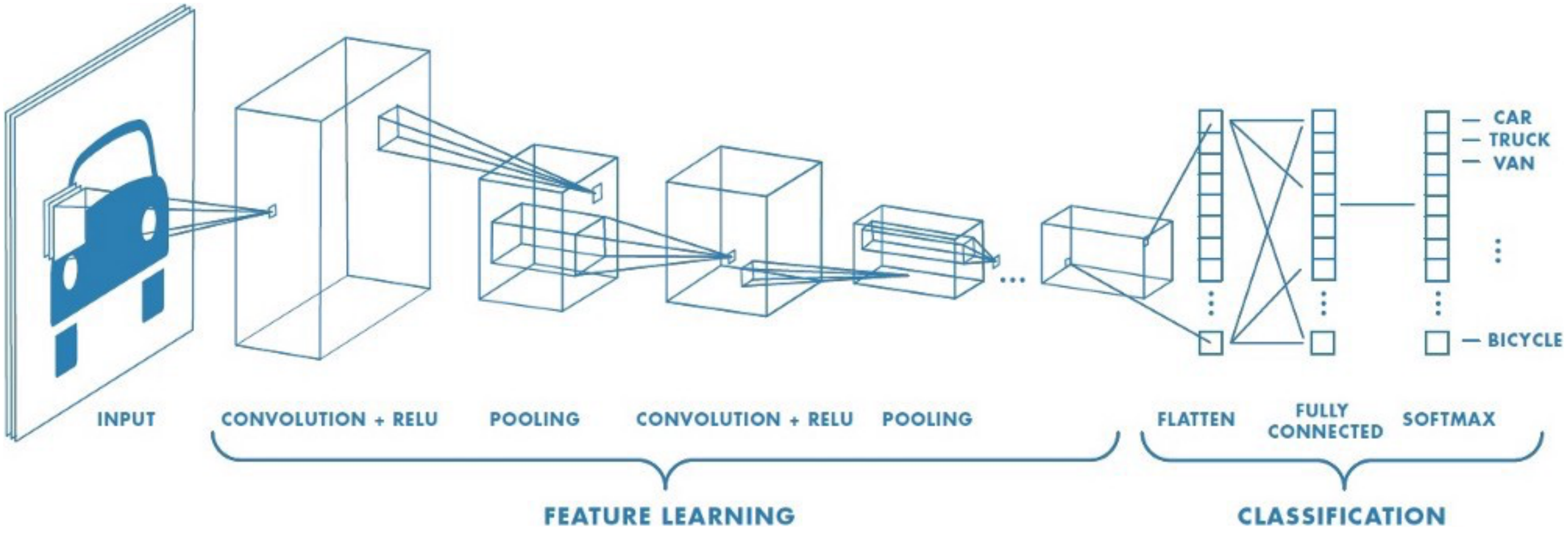



- At first blush, a great fit for serverless infrastructure

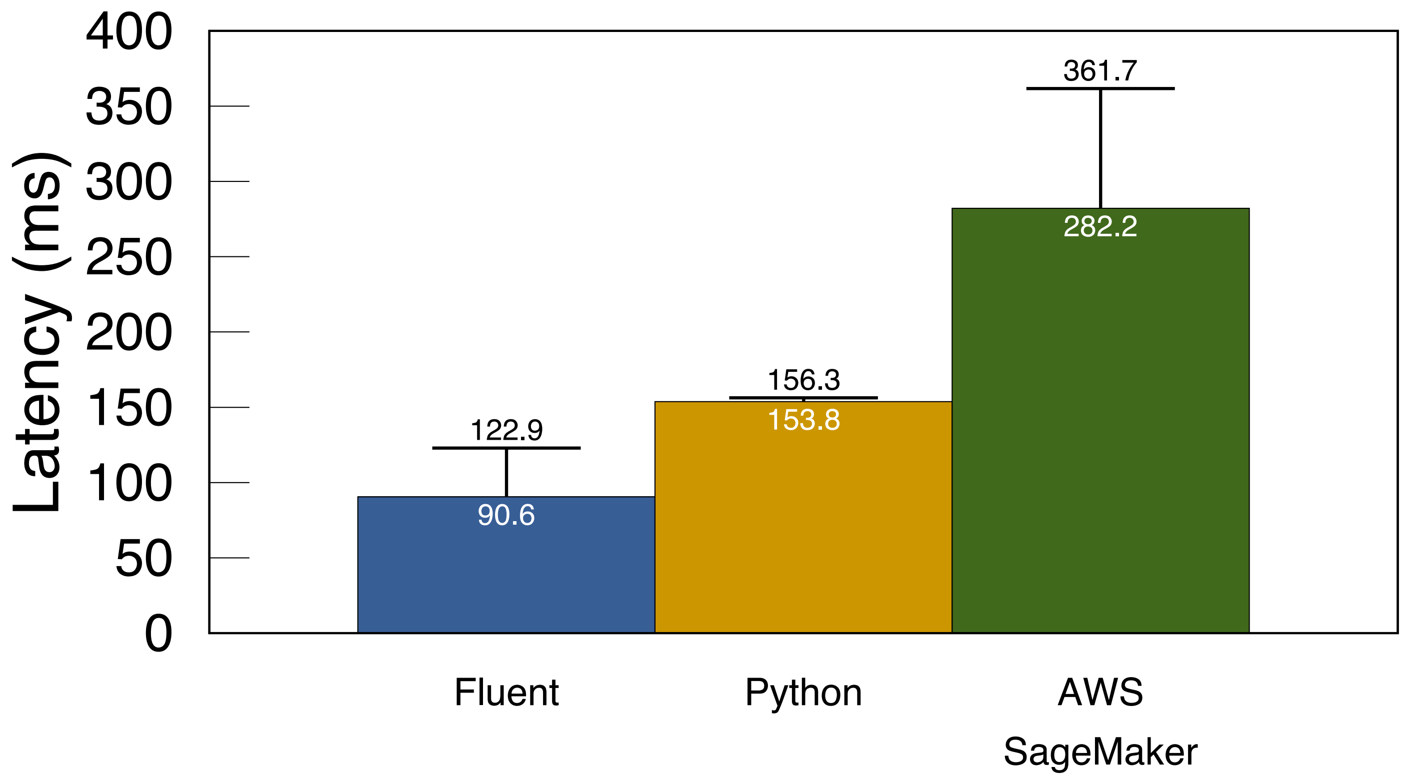

737 FLIGHT
SAGEMAKER
Amazon Kinesis
AWS Lambda
Raw data
Amazon Kinesis Firehose
Amazon Sagemaker
Jupyter Notebooks
Managed Training
Endpoints
COLLECT
EXPLORE & TRANSFORM
TRAIN, VALIDATE, TEST

# Background: SqueezeNet

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



# Prediction Serving



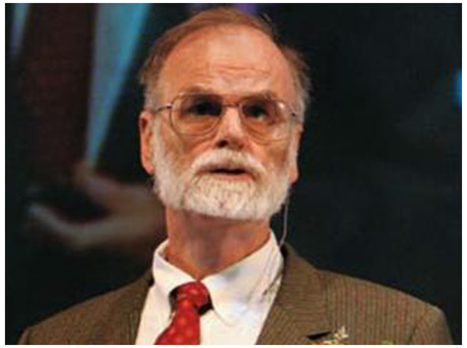

Median and 99$^{th}$ percentile latencies for SqueezeNet on Fluent and AWS SageMaker.

# The Future of Cloud Programming

# Looking Back: Disappointed Computer Scientists

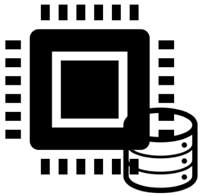
- Functional programming is slow
- Communication through slow storage
- Poor consistency guarantees



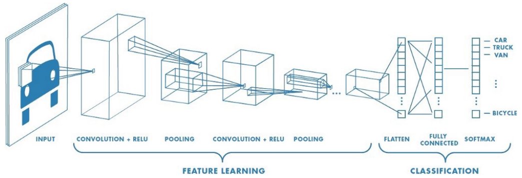



# Making FaaS Functional

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



$$f(g(x))$$

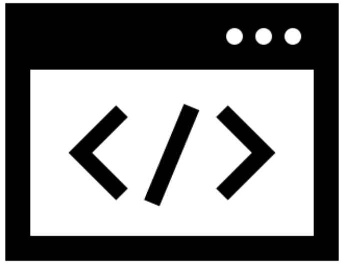



A step on our road towards a programmable cloud.

# 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

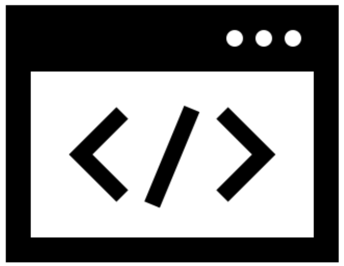
# Moving Forward from FaaS



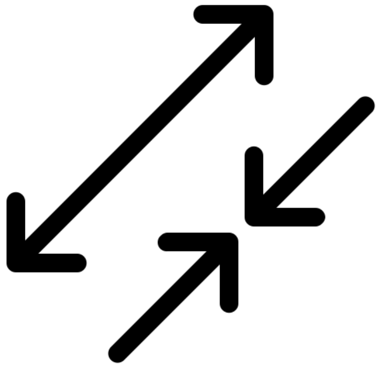
Building
Developer
Tools

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

cgwu@berkeley.edu

fluent-project/fluent