What We Got Wrong

Lessons from the Birth of Microservices at Google

March 4, 2019

LIGHTSTEP

Part One: The Setting

Still betting big on the
Google Search Appliance

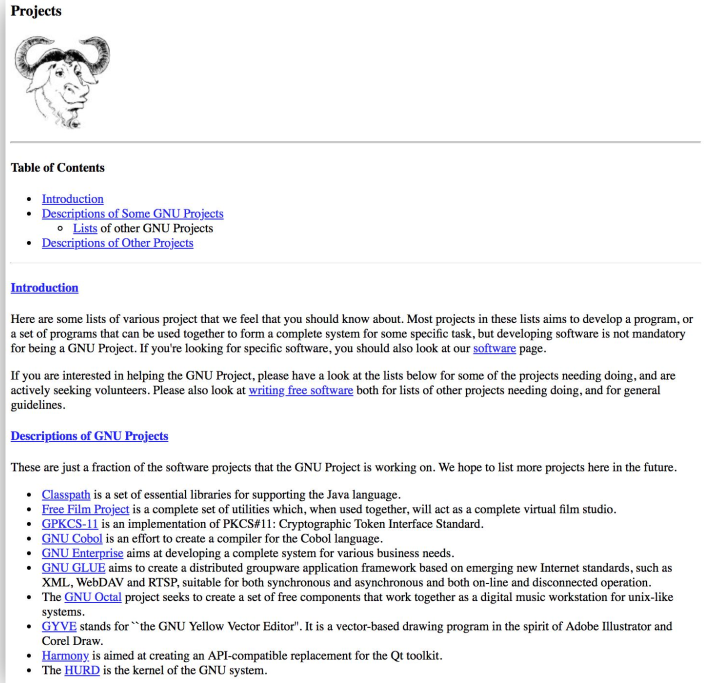
“Those Sun boxes are so expensive!”

“Those linux boxes are so unreliable!”

“Let’s see what’s on GitHub first…”

– literally nobody in 2001

“GitHub” circa 2001



Projects

Table of Contents

- [Introduction]
- [Descriptions of Some GNU Projects]
  - [Lists] of other GNU Projects
- [Descriptions of Other Projects]

Introduction

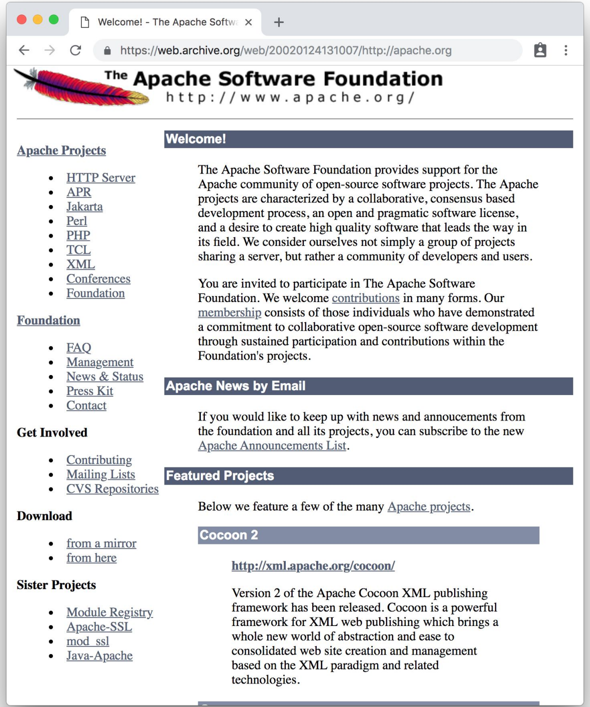
Here are some lists of various project that we feel that you should know about. Most projects in these lists aims to develop a program, or a set of programs that can be used together to form a complete system for some specific task, but developing software is not mandatory for being a GNU Project. If you're looking for specific software, you should also look at our software page.

If you are interested in helping the GNU Project, please have a look at the lists below for some of the projects needing doing, and are actively seeking volunteers. Please also look at writing free software both for lists of other projects needing doing, and for general guidelines.

Descriptions of GNU Projects

These are just a fraction of the software projects that the GNU Project is working on. We hope to list more projects here in the future.

- Classpath is a set of essential libraries for supporting the Java language.
- Free Film Project is a complete set of utilities which, when used together, will act as a complete virtual film studio.
- GPKCS-11 is an implementation of PKCS#11: Cryptographic Token Interface Standard.
- GNU Cobol is an effort to create a compiler for the Cobol language.
- GNU Enterprise aims at developing a complete system for various business needs.
- GNU GLUE aims to create a distributed groupware application framework based on emerging new Internet standards, such as XML, WebDAV and RTSP, suitable for both synchronous and asynchronous and both on-line and disconnected operation.
- The GNU Octal project seeks to create a set of free components that work together as a digital music workstation for unix-like systems.
- GYVE stands for ``the GNU Yellow Vector Editor''. It is a vector-based drawing program in the spirit of Adobe Illustrator and Corel Draw.
- Harmony is aimed at creating an API-compatible replacement for the Qt toolkit.
- The HURD is the kernel of the GNU system.



Welcome! - The Apache Softw

https://web.archive.org/web/20020124131007/http://apache.org

The Apache Software Foundation
http://www.apache.org/

Apache Projects

- HTTP Server
- APR
- Jakarta
- Perl
- PHP
- TCL
- XML
- Conferences
- Foundation

Foundation

- FAQ
- Management
- News & Status
- Press Kit
- Contact

Get Involved

- Contributing
- Mailing Lists
- CVS Repositories

Download

- from a mirror
- from here

Sister Projects

- Module Registry
- Apache-SSL
- mod_ssl
- Java-Apache

Welcome!

The Apache Software Foundation provides support for the Apache community of open-source software projects. The Apache projects are characterized by a collaborative, consensus based development process, an open and pragmatic software license, and a desire to create high quality software that leads the way in its field. We consider ourselves not simply a group of projects sharing a server, but rather a community of developers and users.

You are invited to participate in The Apache Software Foundation. We welcome contributions in many forms. Our membership consists of those individuals who have demonstrated a commitment to collaborative open-source software development through sustained participation and contributions within the Foundation's projects.

Apache News by Email

If you would like to keep up with news and annoucements from the foundation and all its projects, you can subscribe to the new Apache Announcements List.

Featured Projects

Below we feature a few of the many Apache projects.

Cocoon 2

http://xml.apache.org/cocoon/

Version 2 of the Apache Cocoon XML publishing framework has been released. Cocoon is a powerful framework for XML web publishing which brings a whole new world of abstraction and ease to consolidated web site creation and management based on the XML paradigm and related technologies.

LIGHTSTEP

Engineering constraints

- Must DIY:
  - Very large datasets
  - Very large request volume
  - Utter lack of alternatives
- Must scale horizontally
- Must build on commodity hardware that fails often

Google eng cultural hallmarks, early 2000s

- Intellectually rigorous
- “Autonomous” (read: often chaotic)
- Aspirational

Part Two: What Happened

Cambrian Explosion of Infra Projects

Eng culture idolized epic infra projects (for good reason):

- GFS
- BigTable
- MapReduce
- Borg
- Mustang (web serving infra)
- SmartASS (ML-based ads ranking+serving)


The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google

ABSTRACT

1. INTRODUCTION

Categories and Subject Descriptors

General Terms

Keywords

Convergent Evolution?

Common characteristics of the most-admired projects:

- Identification and leverage of horizontal scale-points
- Well-factored application-layer infra (RPC, discovery, load-balancing, eventually tracing, auth, etc)
- Rolling upgrades and frequent (~weekly) releases

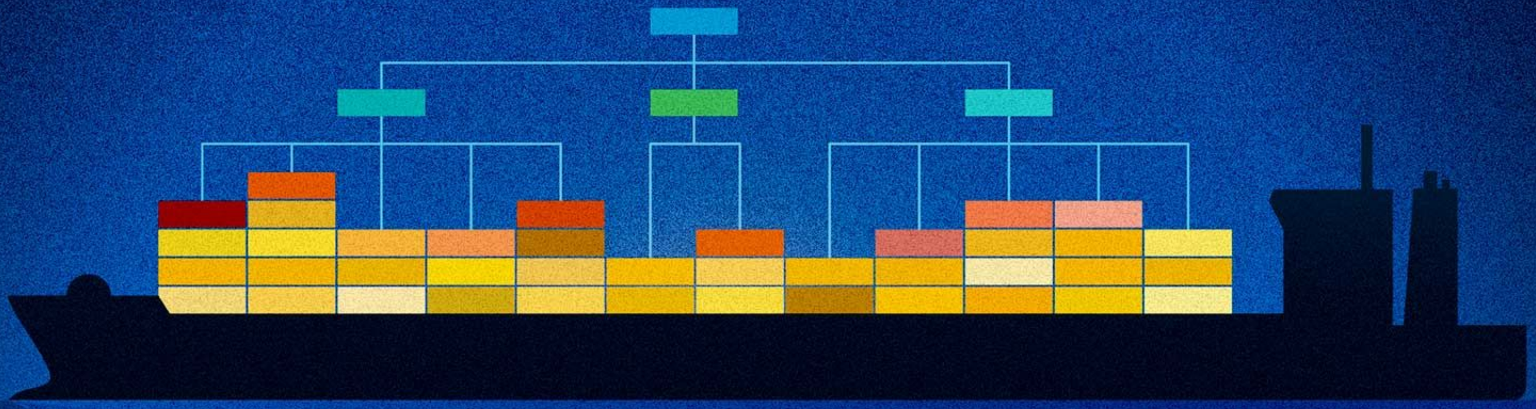
Sounds kinda familiar…

Part Three: Lessons

Lesson 1

Know Why

Org design, human comms, and microservices



You will inevitably ship your org chart

Accidental Microservices

- Microservices motivated by planet-scale technical requirements
- Ended up with something similar to modern microservice architectures …
- … but for different reasons (and that eventually became a problem)

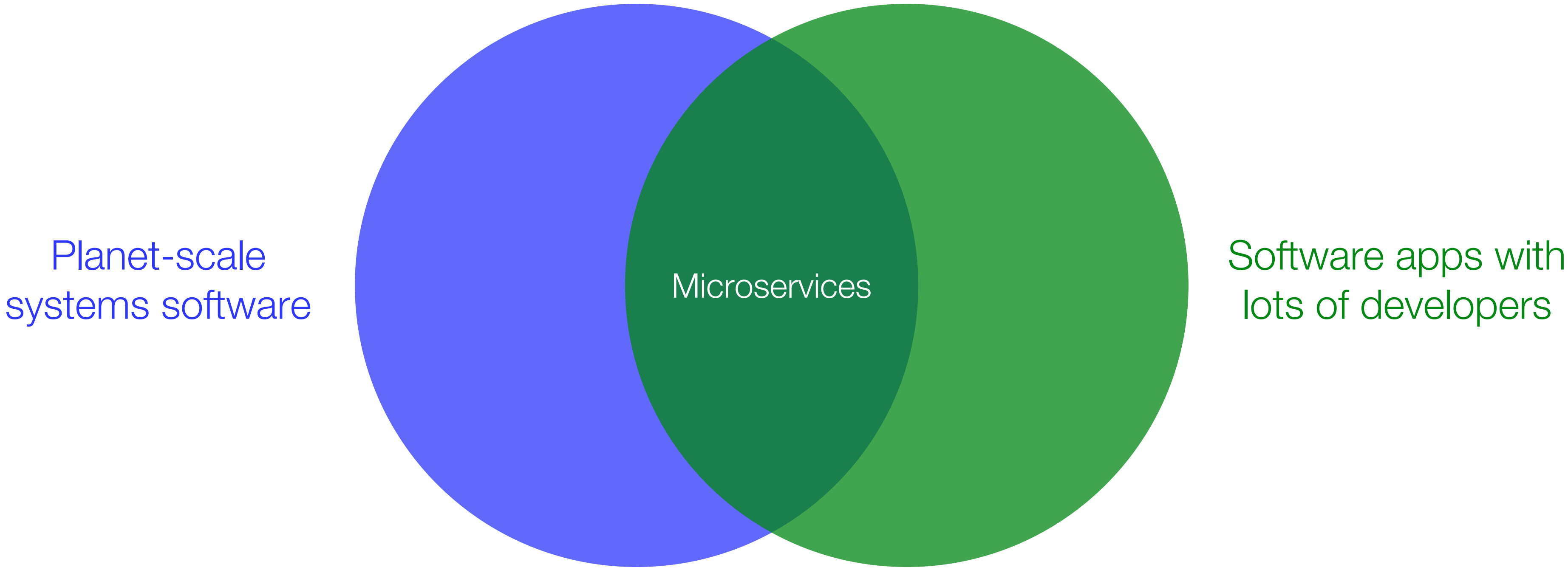
What’s best for Search+Ads is best for all!

What's best for Search+Ads is best for ~~all!~~ just the massive, planet-scale services

“But I just want to serve 5TB!!”

– tech lead for a small service team

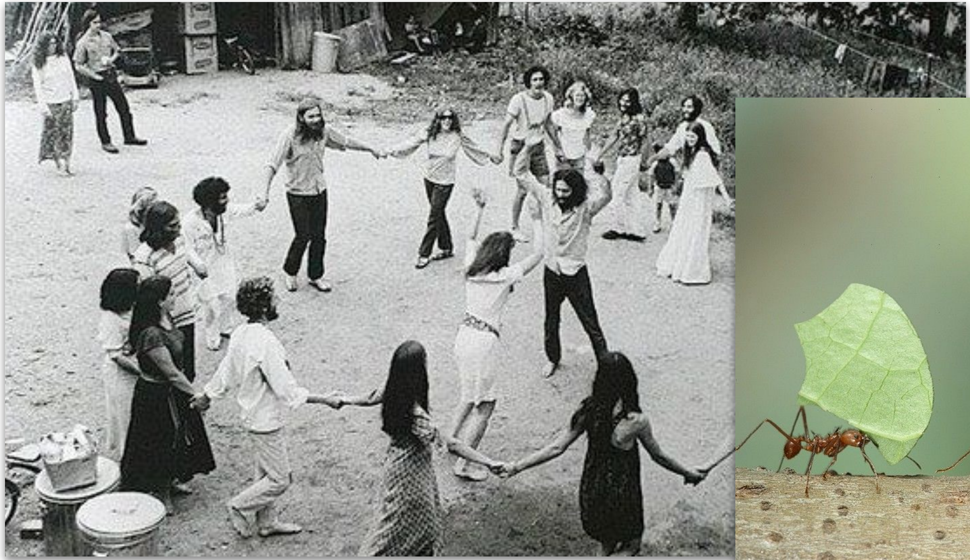
Architectural Overlap



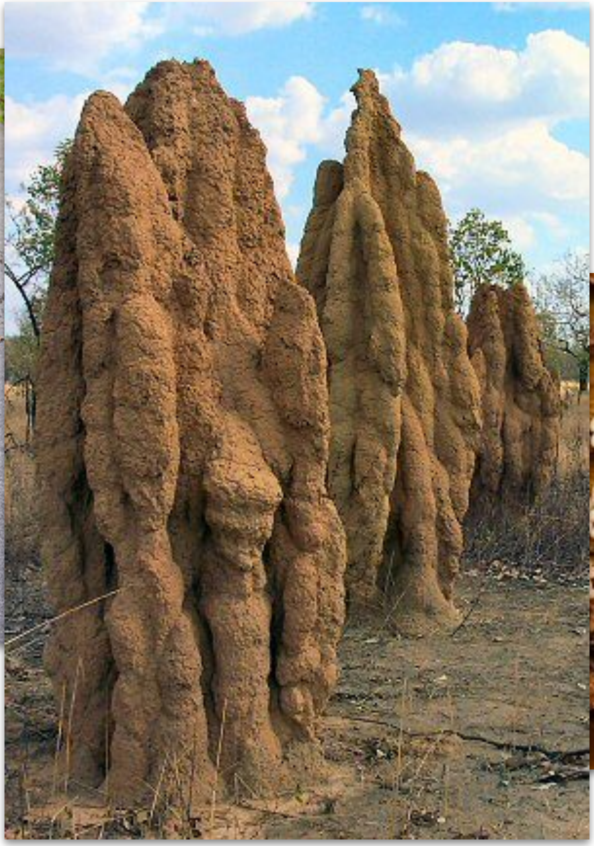
Lesson 2

“Independence” is
not an Absolute

Hippies vs Ants

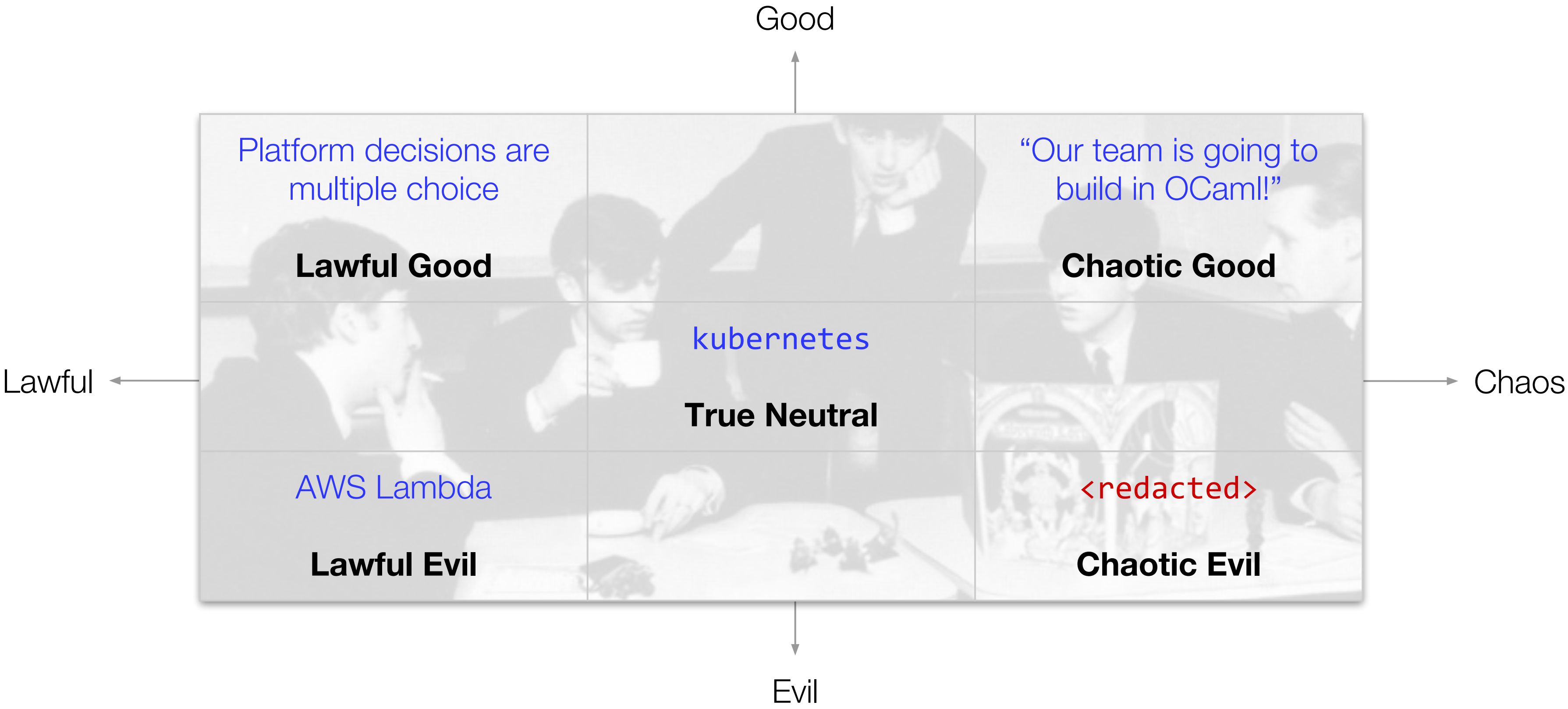


More Ants!




Dungeons and
Dragons!!

Microservices Platforming: D&D Alignment

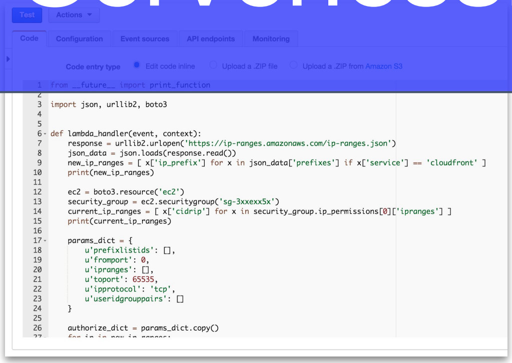


Lesson 3

Serverless Still Runs on Servers

An aside: what do these things have in common?

All 100% Serverless!



LIGHTSTEP

About “Serverless” / FaaS

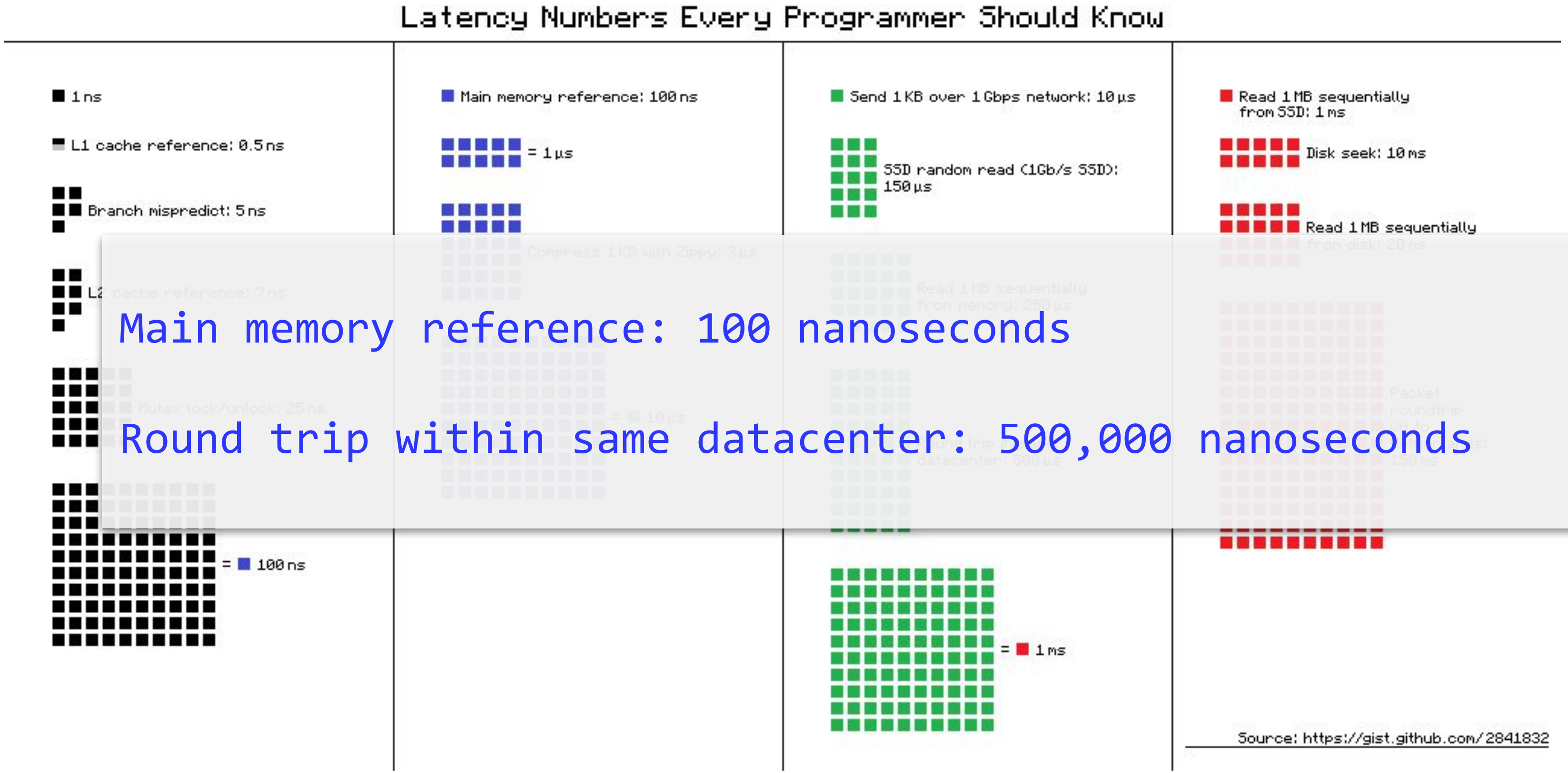
```
Numbers every engineer should know

Latency Comparison Numbers (~2012)
----------------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns                      14x L1 cache
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns                      20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy             3,000   ns        3 us
Send 1K bytes over 1 Gbps network       10,000   ns       10 us
Read 4K randomly from SSD*             150,000   ns      150 us          ~1GB/sec SSD
Read 1 MB sequentially from memory     250,000   ns      250 us
Round trip within same datacenter      500,000   ns      500 us
Read 1 MB sequentially from SSD*     1,000,000   ns    1,000 us    1 ms  ~1GB/sec SSD, 4X memory
Disk seek                           10,000,000   ns   10,000 us   10 ms  20x datacenter roundtrip
Read 1 MB sequentially from disk    20,000,000   ns   20,000 us   20 ms  80x memory, 20X SSD
Send packet CA->Netherlands->CA    150,000,000   ns  150,000 us  150 ms

Notes
-----
1 ns = 10^-9 seconds
1 us = 10^-6 seconds = 1,000 ns
1 ms = 10^-3 seconds = 1,000 us = 1,000,000 ns

Credit
------
By Jeff Dean:               http://research.google.com/people/jeff/
Originally by Peter Norvig: http://norvig.com/21-days.html#answers
```

About “Serverless” / FaaS



Main memory reference: 100 nanoseconds

Round trip within same datacenter: 500,000 nanoseconds

Real data!

velopment of new stateful services, which are the core of modern computing. Meanwhile, with innovation deterred, the cloud vendors increase market dominance for their proprietary solutions. This line of reasoning may suggest that serverless computing could

Hellerstein et al.: “Serverless Computing: One Step Forward, Two Steps Back”

- Weighs the elephants in the room
- Quantifies major issues, esp re service comms and function lifecycle

	Func. Invoc. (1KB)	Lambda I/O (S3)	Lambda I/O (DynamoDB)	EC2 I/O (S3)	EC2 I/O (DynamoDB)	EC2 NW (0MQ)
Latency	303ms	108ms	11ms	106ms	11ms	290μs
Compared to best	1,045×	372×	37.9×	365×	37.9×	1×

Serverless Computing: One Step Forward, Two Steps Back

Joseph M. Hellerstein, Jose Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov and Chenggang Wu
UC Berkeley
{hellerstein,jmfaleiro,jegonzal,jssmith,vikrams,atumanov,cgwu}@berkeley.edu

arXiv:1812.03651v1 [cs.DC] 10 Dec 2018

ABSTRACT

Serverless computing offers the potential to program the cloud in an autoscaling, pay-as-you go manner. In this paper we address critical gaps in first-generation serverless computing, which place its autoscaling potential at odds with dominant trends in modern computing: notably data-centric and distributed computing, but also open source and custom hardware. Put together, these gaps make current serverless offerings a bad fit for cloud innovation and particularly bad for data systems innovation. In addition to pinpointing some of the main shortfalls of current serverless architectures, we raise a set of challenges we believe must be met to unlock the radical potential that the cloud—with its exabytes of storage and millions of cores—should offer to innovative developers.

1 INTRODUCTION

Amazon Web Services recently celebrated its 12th anniversary, marking over a decade of public cloud availability. While the cloud began as a place to timeshare machines, it was clear from the beginning that it presented a radical new computing platform: the biggest assemblage of data capacity and distributed computing power ever available to the general public, managed as a service.

Despite that potential, we have yet to harness cloud resources in radical ways. The cloud today is largely used as an outsourcing platform for standard enterprise data services. For this to change, creative developers need programming frameworks that enable them to leverage the cloud’s power.

New computing platforms have typically fostered innovation in programming languages and environments. Yet a decade later, it is difficult to identify the new programming environments for the cloud. And whether cause or effect, the results are clearly visible in practice: the majority of cloud services are simply multi-tenant, easier-to-administer clones of legacy enterprise data services like object storage, databases, queueing systems, and web/app servers. Multitenancy and administrative simplicity are admirable and desirable goals, and some of the new services have interesting internals in their own right. But this is, at best, only a hint of the potential offered by millions of cores and exabytes of data.

Recently, public cloud vendors have begun offering new programming interfaces under the banner of *serverless computing*, and interest is growing. Google search trends show that queries for the term “serverless” recently matched the historic peak of popularity of the phrase “Map Reduce” or “MapReduce” (Figure 1). There has also been a significant uptick in attention to the topic more recently from the research community [13, 6, 26, 14]. Serverless computing offers the attractive notion of a platform in the cloud where developers simply upload their code, and the platform executes it on their behalf as needed at any scale. Developers need not concern themselves with provisioning or operating servers, and they pay only for the compute resources used when their code is invoked.

The notion of serverless computing is vague enough to allow optimists to project any number of possible broad interpretations on what it might mean. Our goal here is not to quibble about the terminology. Concretely, each of the cloud vendors has already launched serverless computing infrastructure and is spending a significant marketing budget promoting it. In this paper, we assess the field based on the serverless computing services that vendors are actually offering today and see why they are a disappointment as big as the cloud’s potential.

1.1 “Serverless” goes FaaS

To begin, we provide a quick introduction to *Functions-as-a-Service* (FaaS), the commonly used and more descriptive name for the core of serverless offerings from the public cloud providers. Because AWS was the first public cloud—and remains the largest—we focus our discussion on the AWS FaaS framework, Lambda; offerings from Azure and GCP differ in detail but not in spirit.

The idea behind FaaS is simple and straight out of a programming textbook. Traditional programming is based on writing functions, which are mappings from inputs to outputs. Programs consist of compositions of these functions. Hence, one simple way to program the cloud is to allow developers to register functions in the cloud, and compose those functions into programs.

Typical FaaS offerings today support a variety of languages (e.g., Python, Java, Javascript, Go), allow programmers to register functions with the cloud provider, and enable users to declare events that trigger each function. The FaaS infrastructure monitors the triggering events, allocates a runtime for the function, executes it, and persists the results. The user is billed only for the computing resources used during function invocation.

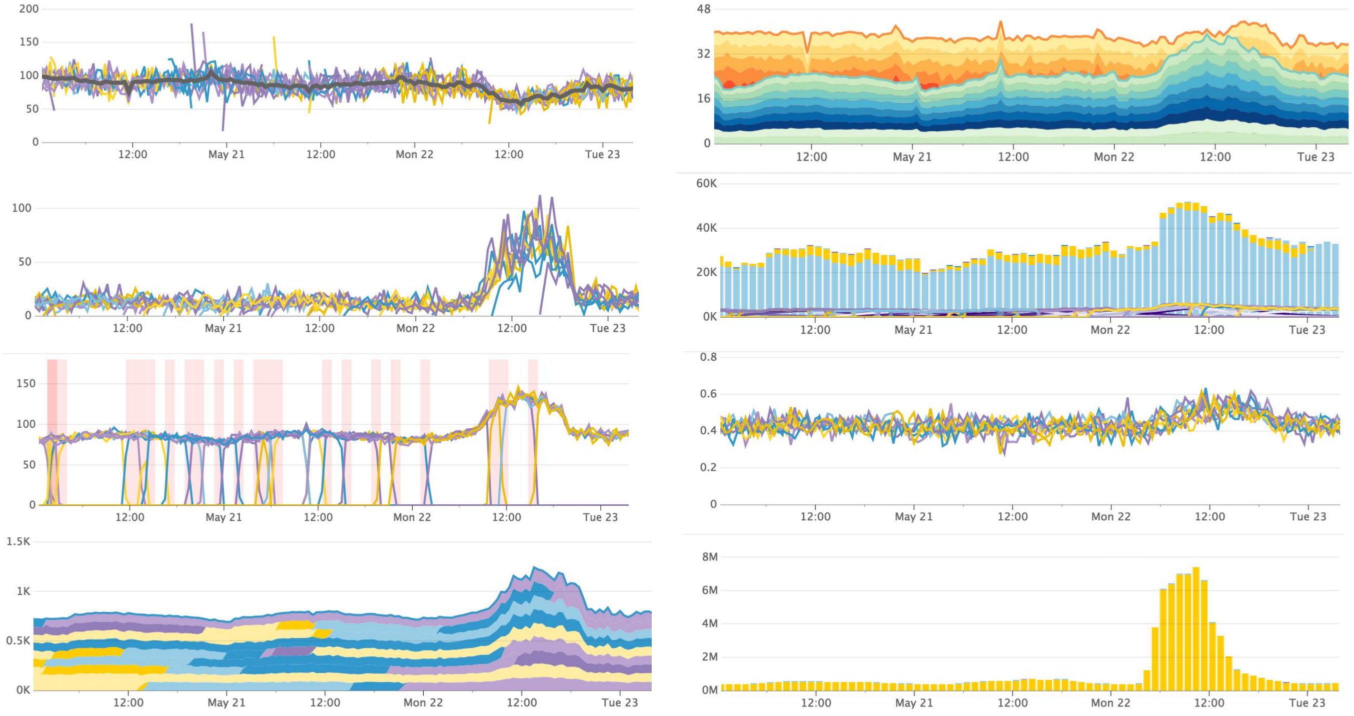
A FaaS offering by itself is of little value, since each function execution is isolated and ephemeral. Building applications on FaaS requires data management in both persistent and temporary storage, in addition to mechanisms to trigger and scale function execution. As a result, cloud providers are quick to emphasize that serverless is not only FaaS. It is FaaS supported by a “standard library”: the various multitenanted, autoscaling services provided by the vendor[1]. In the case of AWS, this includes S3 (large object storage), DynamoDB (key-value storage), SQS (queuing services), SNS (notification services), and more. This entire infrastructure is managed and operated by AWS; developers simply register FaaS code that

[1] This might be better termed a "proprietary library", but the analogy to C's stdlib is apropos: not officially part of the programming model, but integral in practice.

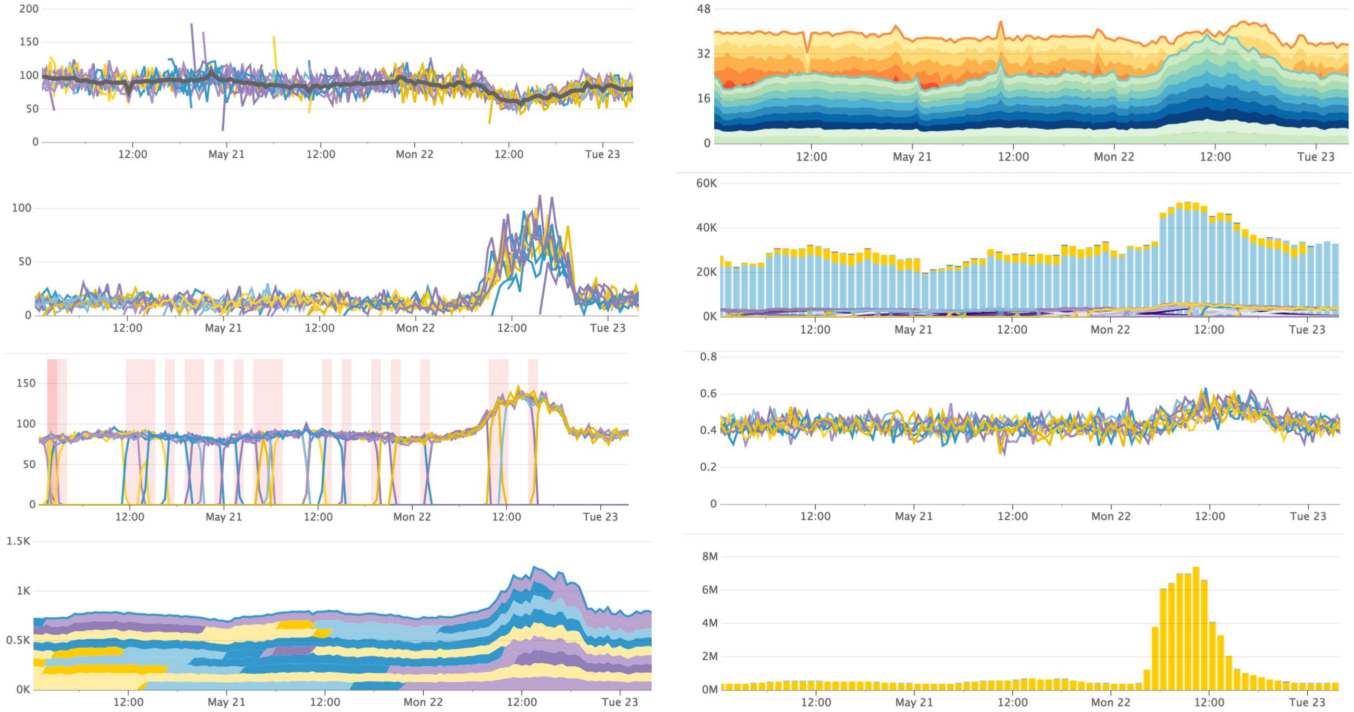
Lesson 4

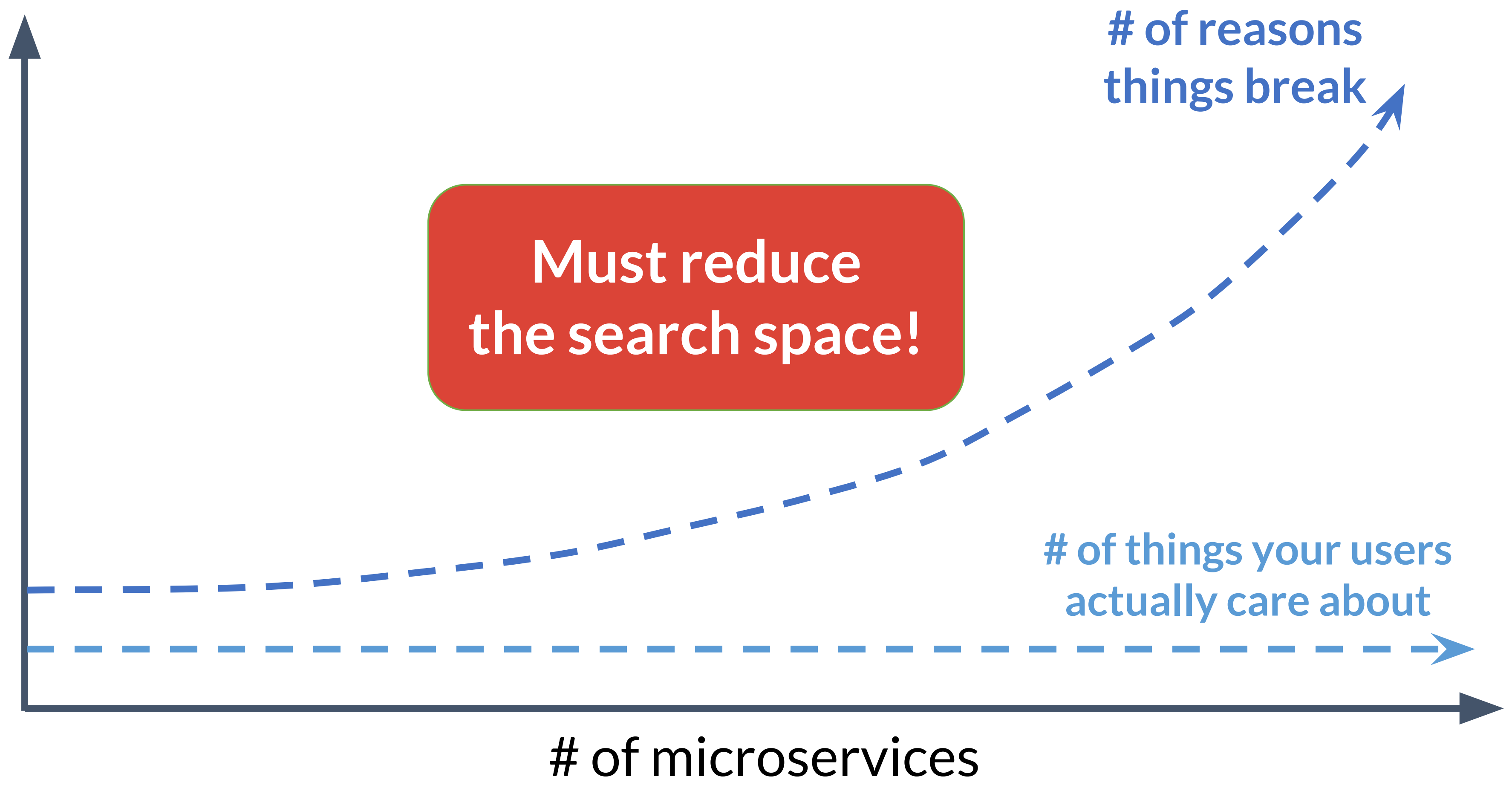
Beware Giant Dashboards

We caught the regression!



… but which is the culprit?

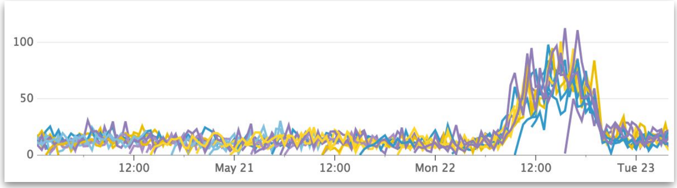




# of reasons things break
Must reduce the search space!
# of things your users actually care about
# of microservices

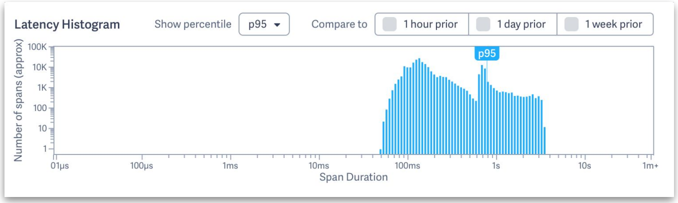
All of observability in two activities

1. Detection of critical signals (SLIs)
2. Explaining variance



variance over time

“Visualizing everything that might vary” is **a *terrible* way to explain variance.**

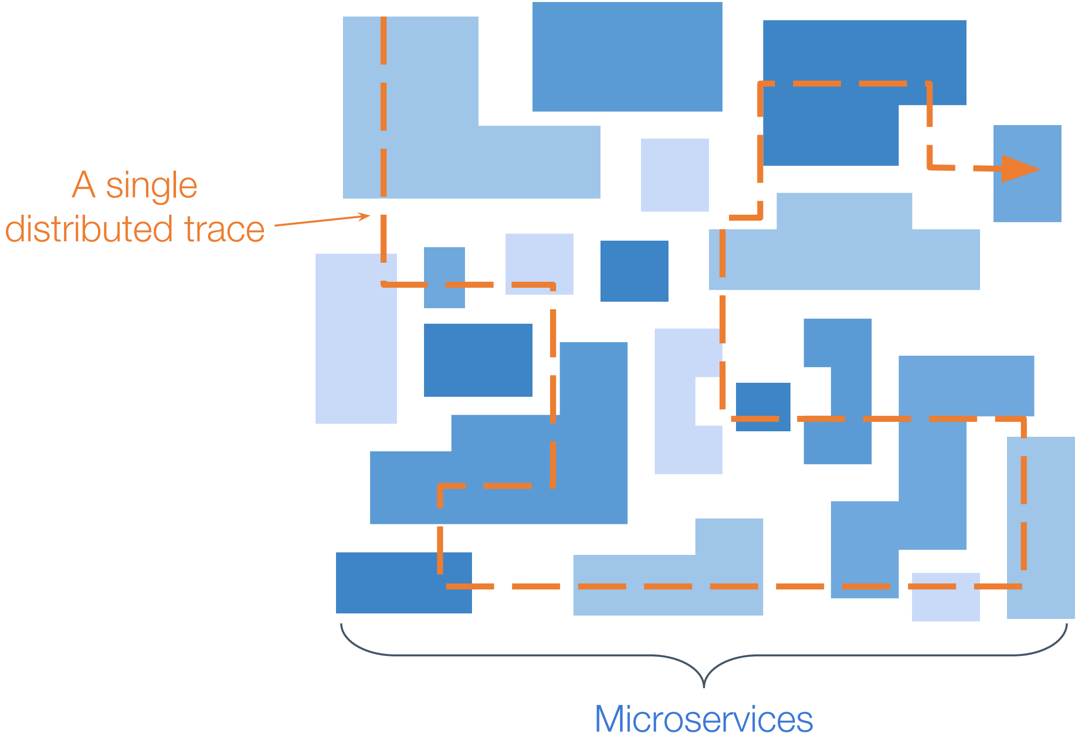


variance in the latency distribution

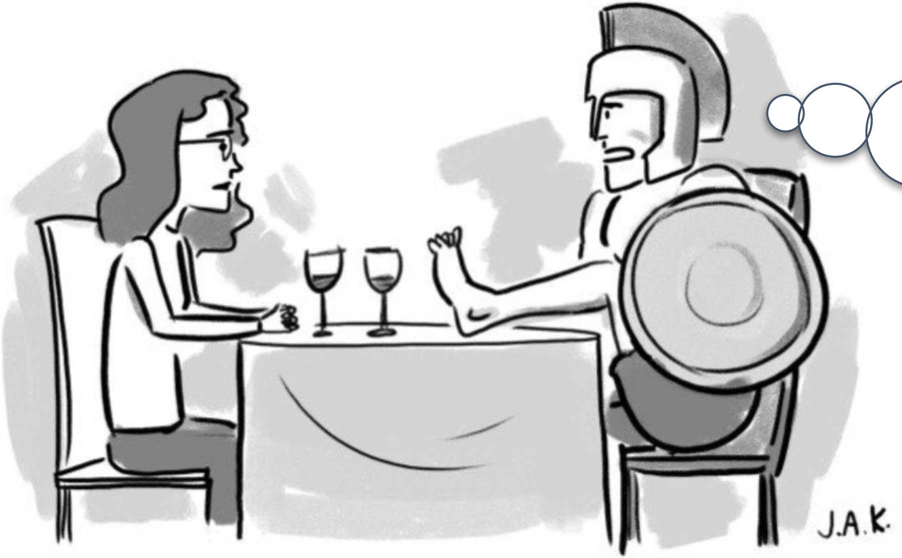
Lesson 5

Distributed Tracing is more than Distributed Traces

Distributed Tracing 101

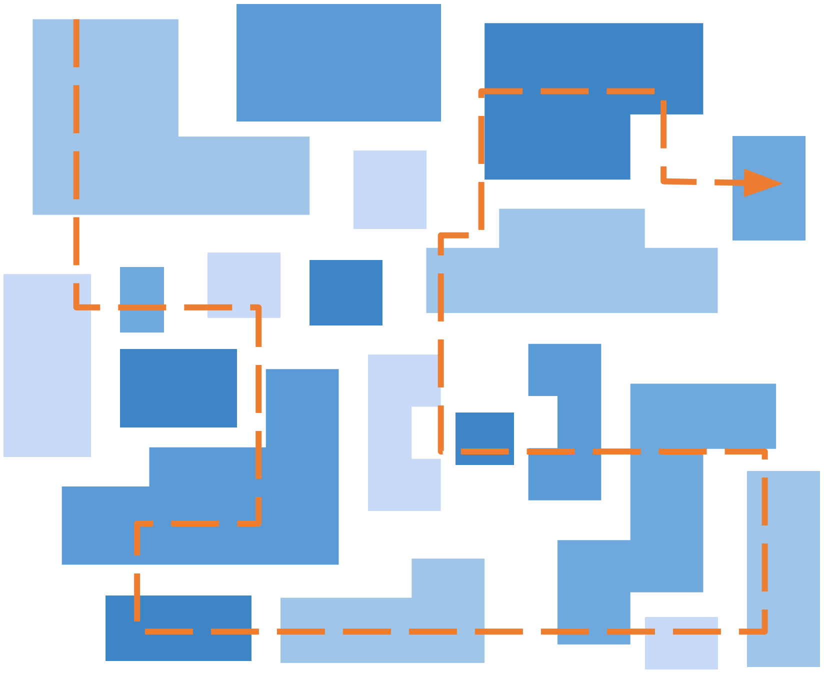


There are some things I need to tell you…



"I'm ready to be vulnerable."

Trace Data Volume: a reality check



```
    app transaction rate
x    # of microservices
x   cost of net+storage
x    weeks of retention

------------------------

     way too much $$$$
```

The Life of Trace Data: Dapper

Stage	**Overhead affects…**	**Retained**
Instrumentation Executed	App	100.00%
Buffered within app process	App	000.10%
Flushed out of process	App	000.10%
Centralized regionally	Regional network + storage	000.10%
Centralized globally	WAN + storage	000.01%

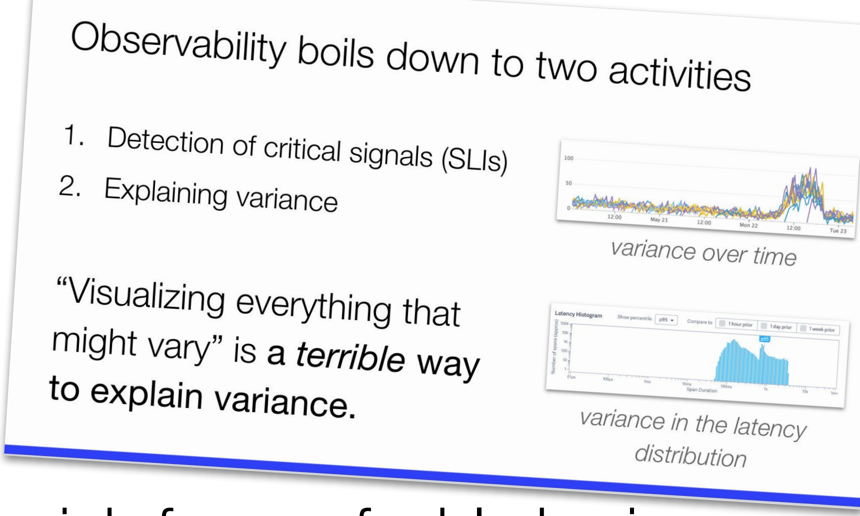
LIGHTSTEP

The Life of Trace Data: ~~Dapper~~ "Other Approaches"

Stage	Overhead affects…	Retained
Instrumentation Executed	App	100.00%
Buffered within app process	App	100.00%
Flushed out of process	App	100.00%
Centralized regionally	Regional network + storage	100.00%
Centralized globally	WAN + storage	on-demand

LIGHTSTEP

But wait, there's more!



- Visualizing individual traces is necessary but not sufficient
- Raw distributed trace data is too rich for our feeble brains
- A superior approach:
  - Ingest 100% of the raw distributed trace data
  - Measure SLIs with high precision (e.g., latency, errors)
  - Explain variance with *biased* sampling and "real" stats

Meta: more detail in my other talk today and Weds keynote

Almost Done…

Let’s review…

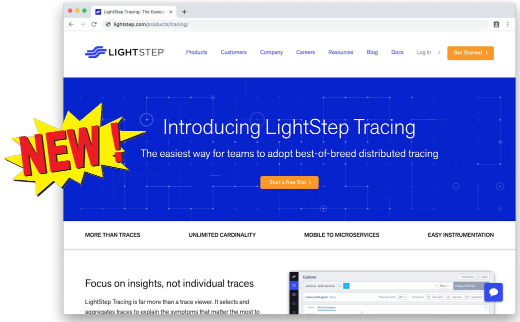
- Two drivers for microservices: what are you solving for?
    - Team independence and velocity
    - “Computer Science”
- Understand the appropriate scale for any solution
- Hippies vs Ants
- Services can be too small (i.e., “the network isn’t free”)
- Observability is about *Detection* and *Refinement*
- “Distributed tracing” must be more than “distributed traces”

LIGHTSTEP

Thank you!

Ben Sigelman, Co-founder and CEO
twitter: @el_bhs
email: bhs@lightstep.com

PS: LightStep announced something cool today!



I am friendly and would love to chat… please say hello, I don't make it to Europe often!