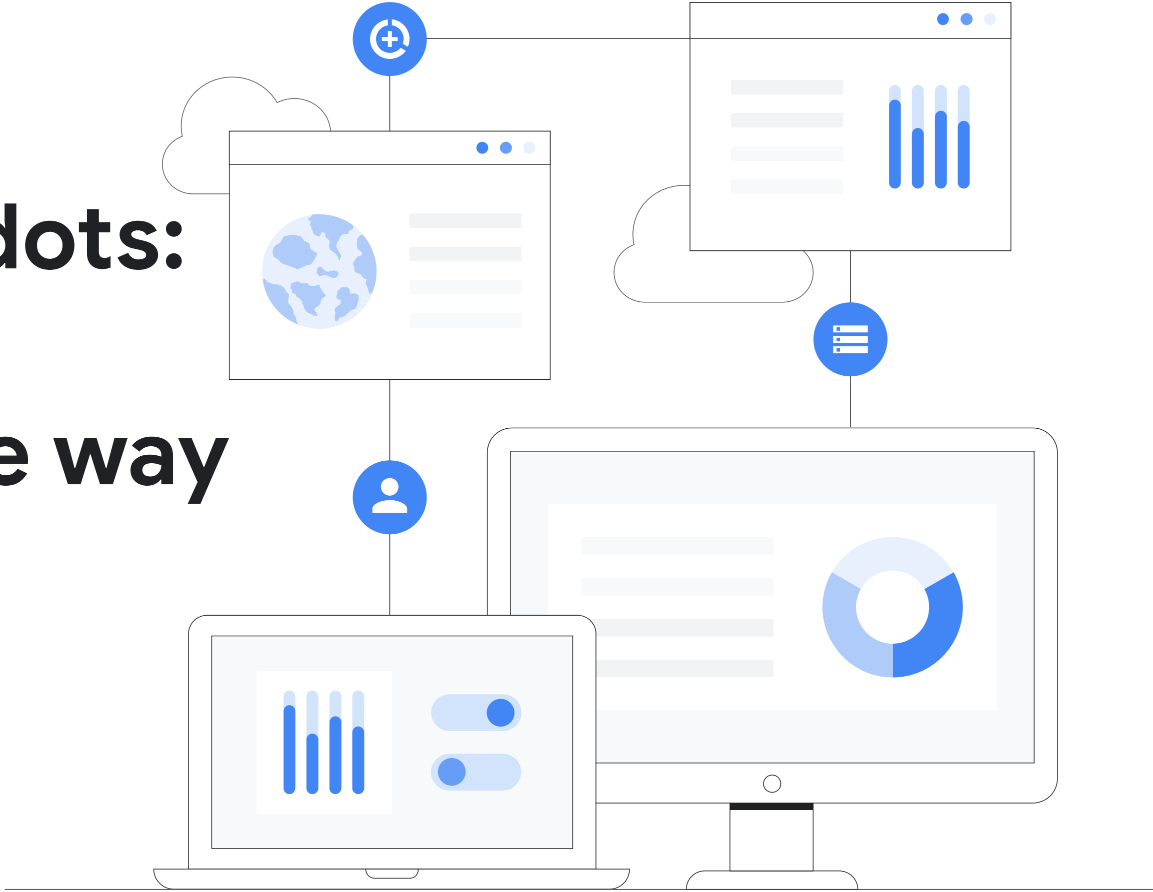
HPTS 2022

Gong Show

Google Cloud

Connecting the dots: Evolution and Revolution on the way to the Cloud



Sailesh Krishnamurthy
VP, Engineering

Google Cloud

Two approaches circa ~2002

Evolution

Many smaller databases

Migrating existing workloads

Evolve existing RDBMS engines

Revolution

Fewer massive databases

Newer, transformative workloads

Scale over relational semantics

What changed in the last ~20 years ?

Databases: Evolution and Revolution

Google Cloud SQL

- Control planes to manage RDBMS
- Hosted in commodity VMs
- Backed by generic block storage

Google AlloyDB

- Database-optimized storage
- Offload IO, improve costs/latency
- ***Compute / Storage separation***

Google Bigtable

- Partition for scale, solve IR problems
- NoSQL/KV instead of xact, relational, SQL
- ***Compute / Storage separation***

Google Spanner

- Global cross-partition transactions
- Full ANSI SQL, relational semantics
- ***Compute / Storage separation***

Data Warehouses: Evolution and Revolution

Hoist MPP DW to Cloud

- Control planes on MPP RDBMS
- Storage co-located with compute
- ***Data partitioning usually "sticky"***

Cloud MPP

- ***Compute / Storage separation***
- Autoscale compute
- Data planes still use MPP RDBMS

Dremel: "Online Map Reduce"

- Build for scale
- Forgo relational/SQL semantics
- Flex compute shape thru containers
- ***Compute / Storage separation***

BigQuery

- Full ANSI SQL support
- Enterprise security and governance
- Serverless relational data warehouse
- ***Compute / Storage separation***

Key Takeaways

Did anybody say "separation of compute and storage" ??

Cross-pollination of ideas is great for our community

- Differing motivations have driven continuous innovation
- The worlds of "revolution" and "evolution" are now converging

New opportunities

- Cloud customers demand more integrated services
- Analytics and Transactional systems can leverage each other

Google's unique approach is highly differentiated:

- Build infrastructure at unprecedented scale
- Reuse with external and internal customers

Thank you

Petalith

Memory is the treasury and guardian of all things - Cicero

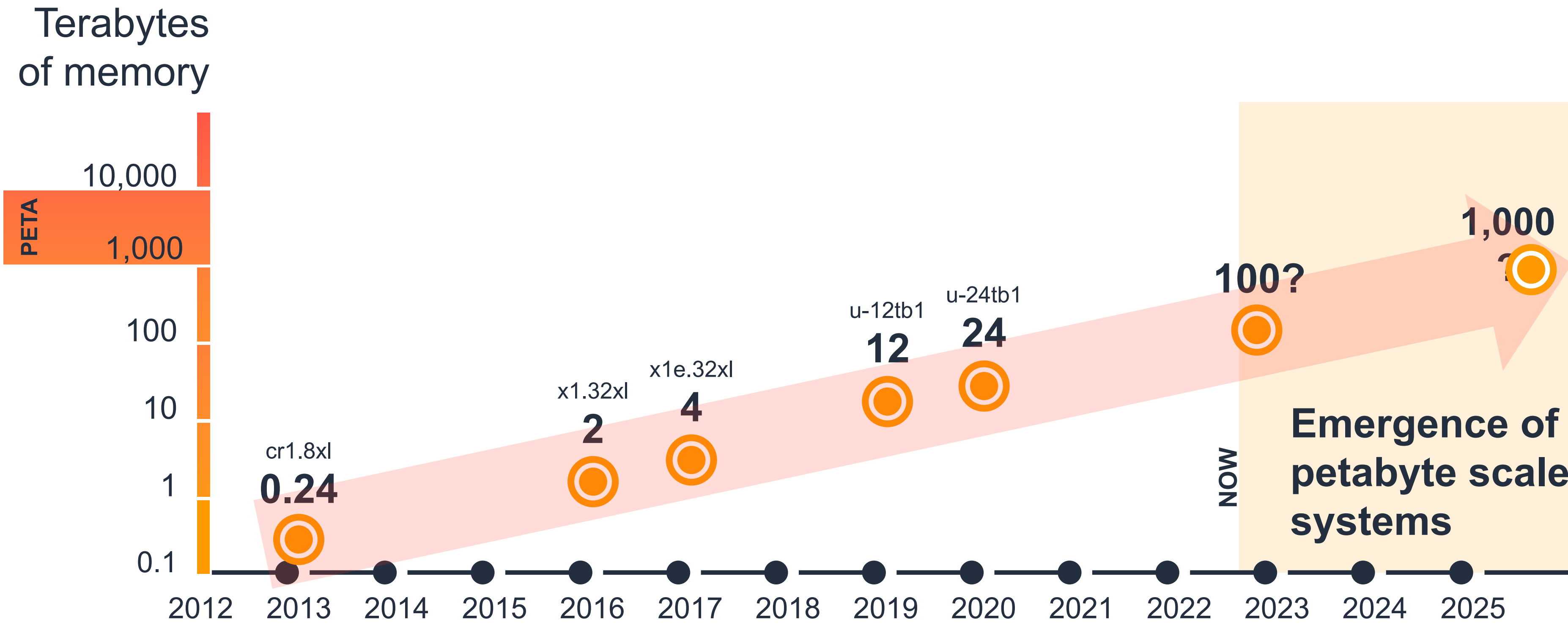
Adrian Cockcroft - OrionX.net
HPTS 2022

What is big data?

Data that doesn't fit in memory on one machine

Currently about **24 TB**

Biggest Memory Sizes Trend



What limits speedup?

Amdahl's Law – the serial portion of a workload

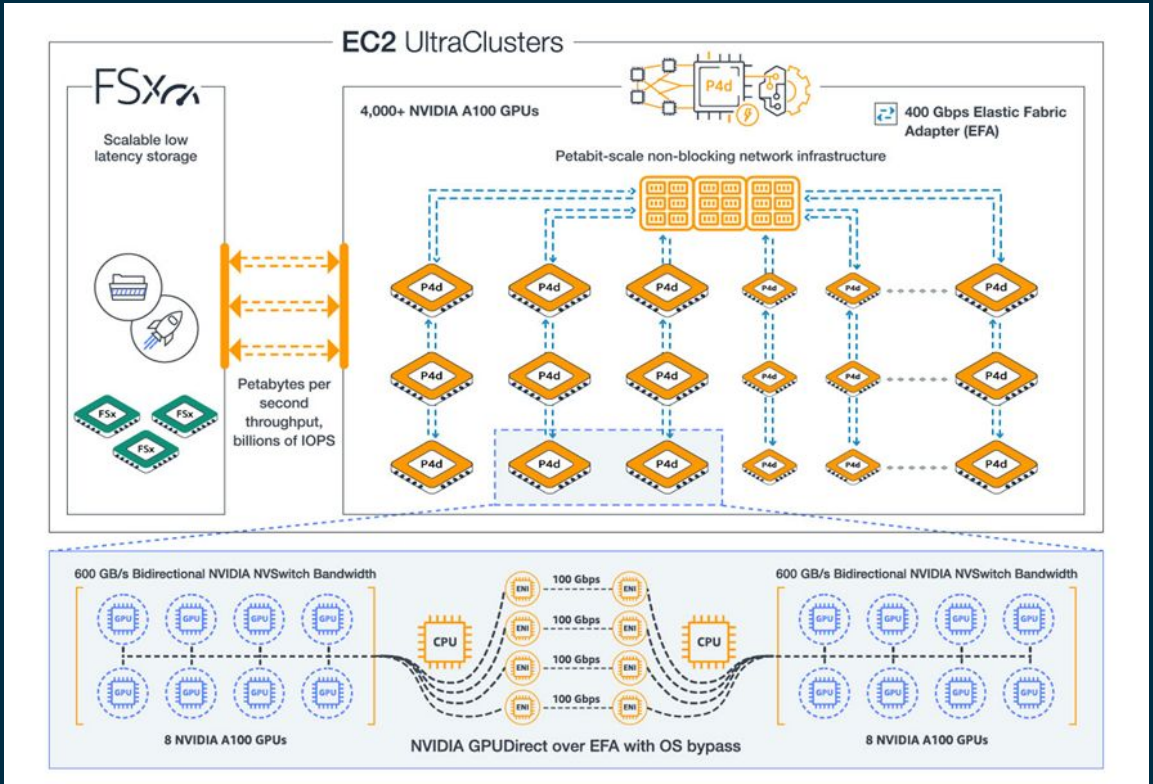
In a distributed system **The Communication**

"the overall performance improvement gained by optimizing a single part of a system is limited by the fraction of time that the improved part is actually used" – Gene Amdahl - 1967

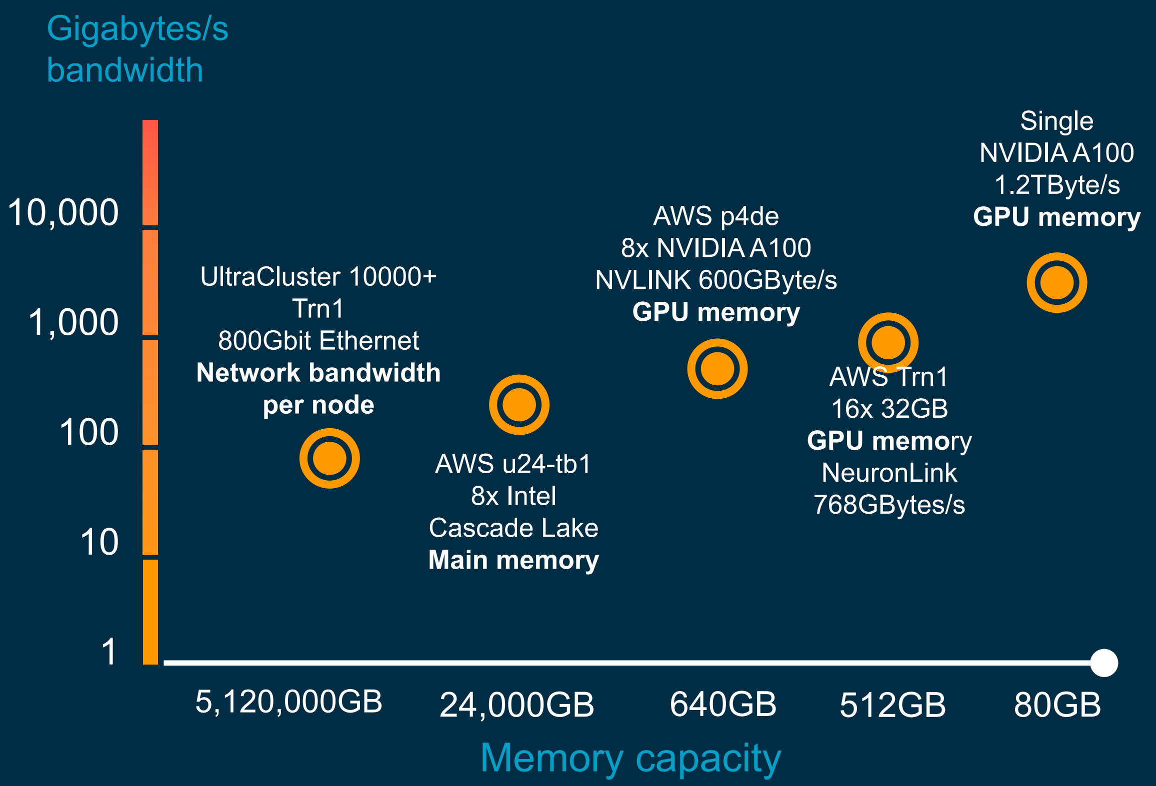
NETWORKING

Ethernet capacity in a single instance

EC2 UltraClusters (2022)
10000+ Trainium GPUs on 800GBit links into petabit scale fabric



Memory capacity vs. bandwidth



Engulf your data in memory to reduce overhead, if it fits

How do we communicate?

By encoding transmitting receiving and decoding

How do we do it? **Really inefficiently!**

Send an email with an idea and wait

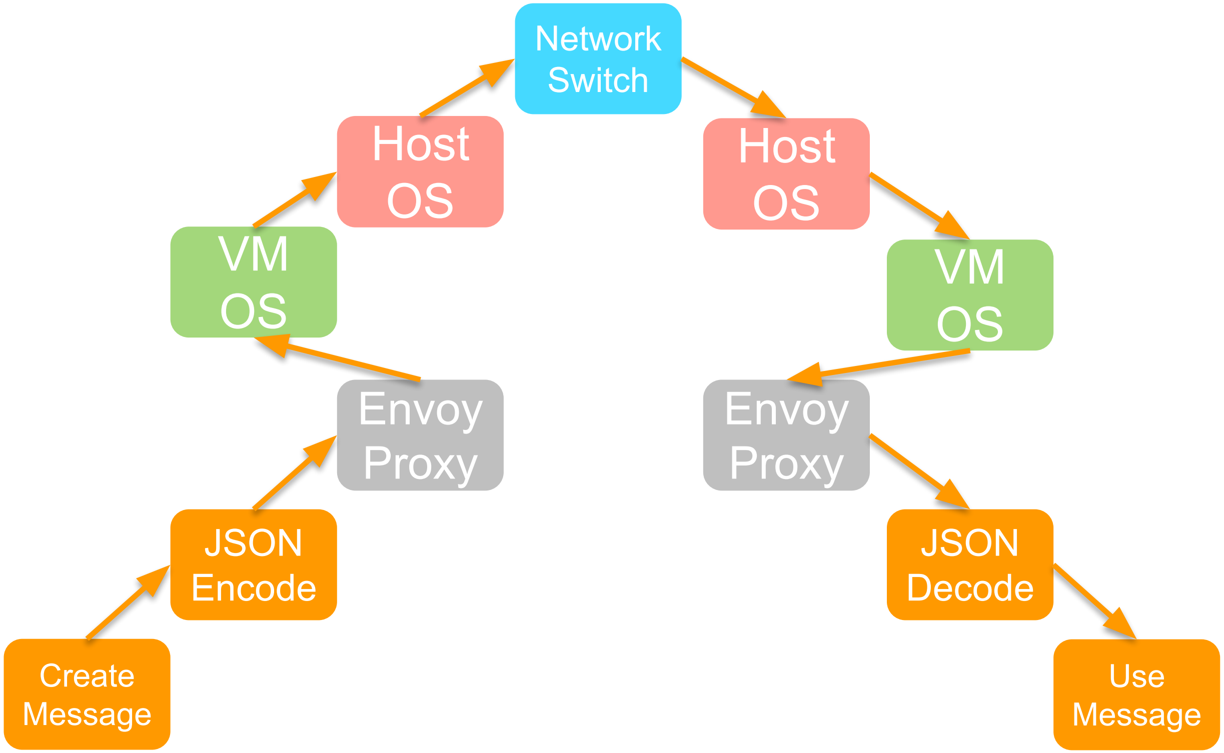
A bit better? Direct Conversation (e.g. at HPTS) Including eye contact and body language protocols

How do systems communicate?

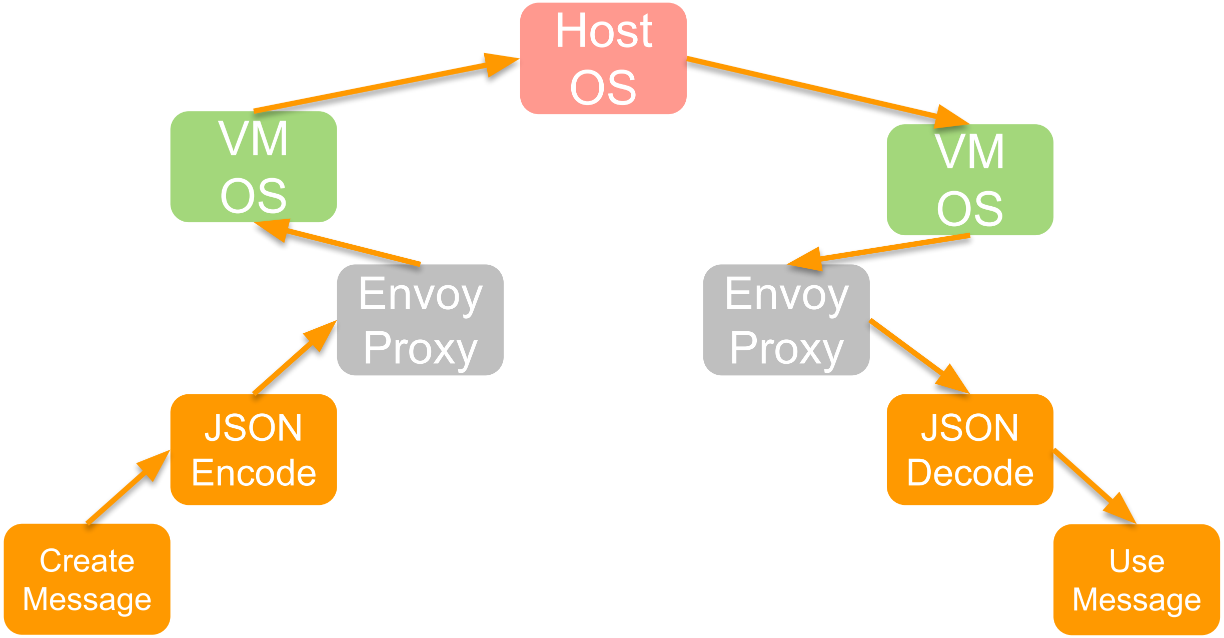
By encoding transmitting receiving and decoding

How do systems do it? **Really inefficiently!**

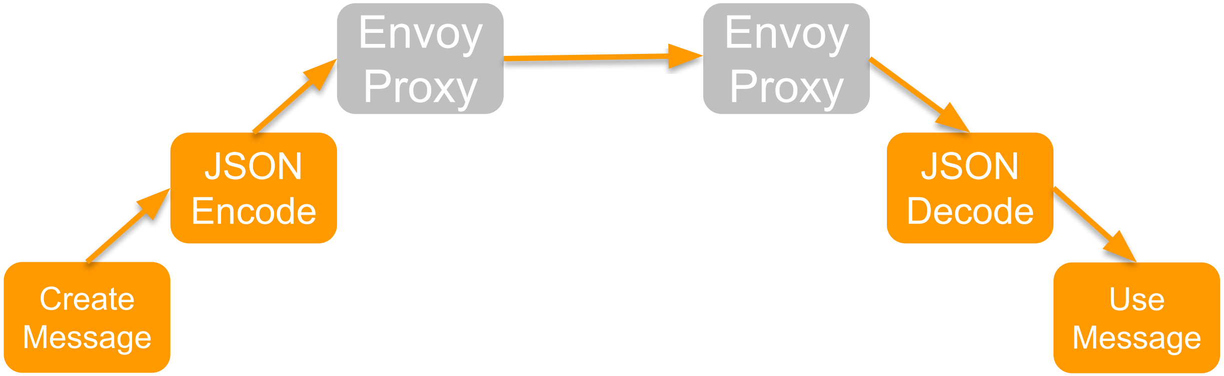
Typical containerized microservice call pattern
Every step makes a new copy of the message



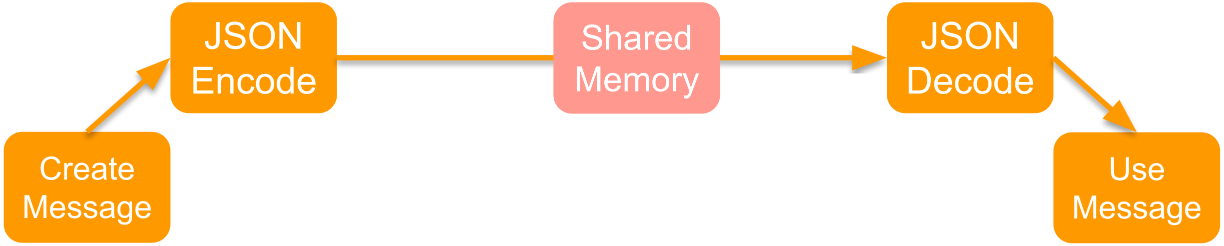
Shortcut the network



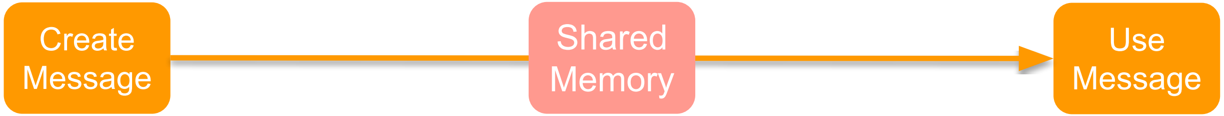
Use Memory as the Network (no app code changes)



Use Memory as the Network (Repackaged container sidecar)



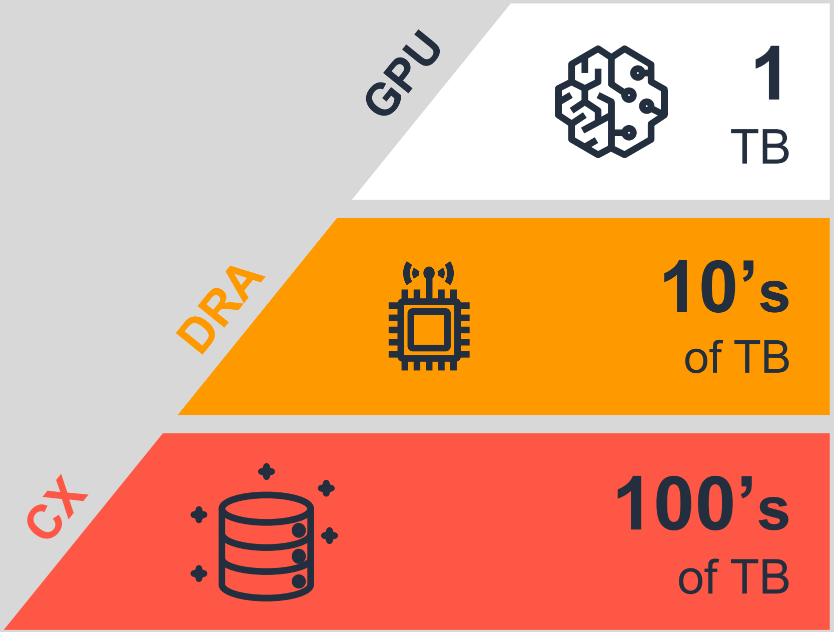
Use Memory as the Network
(No need to encode/decode)

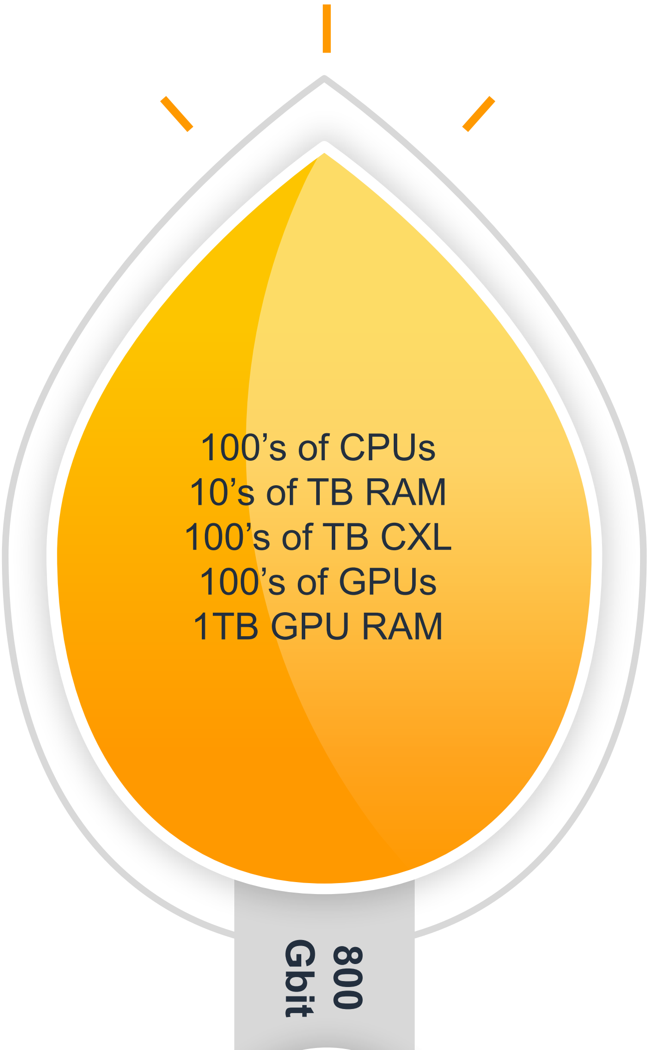


IN 2023 OR SO…

Large scale system

New memory hierarchy to manage



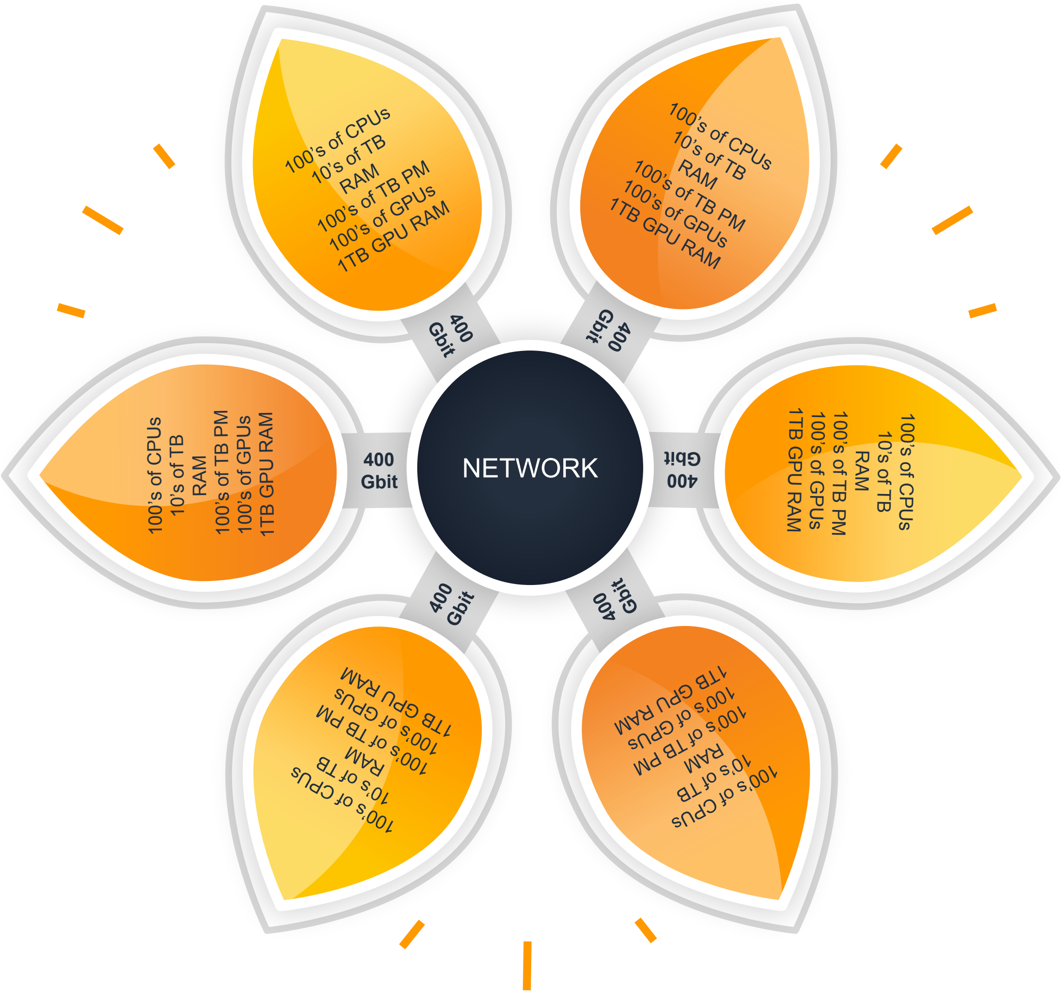


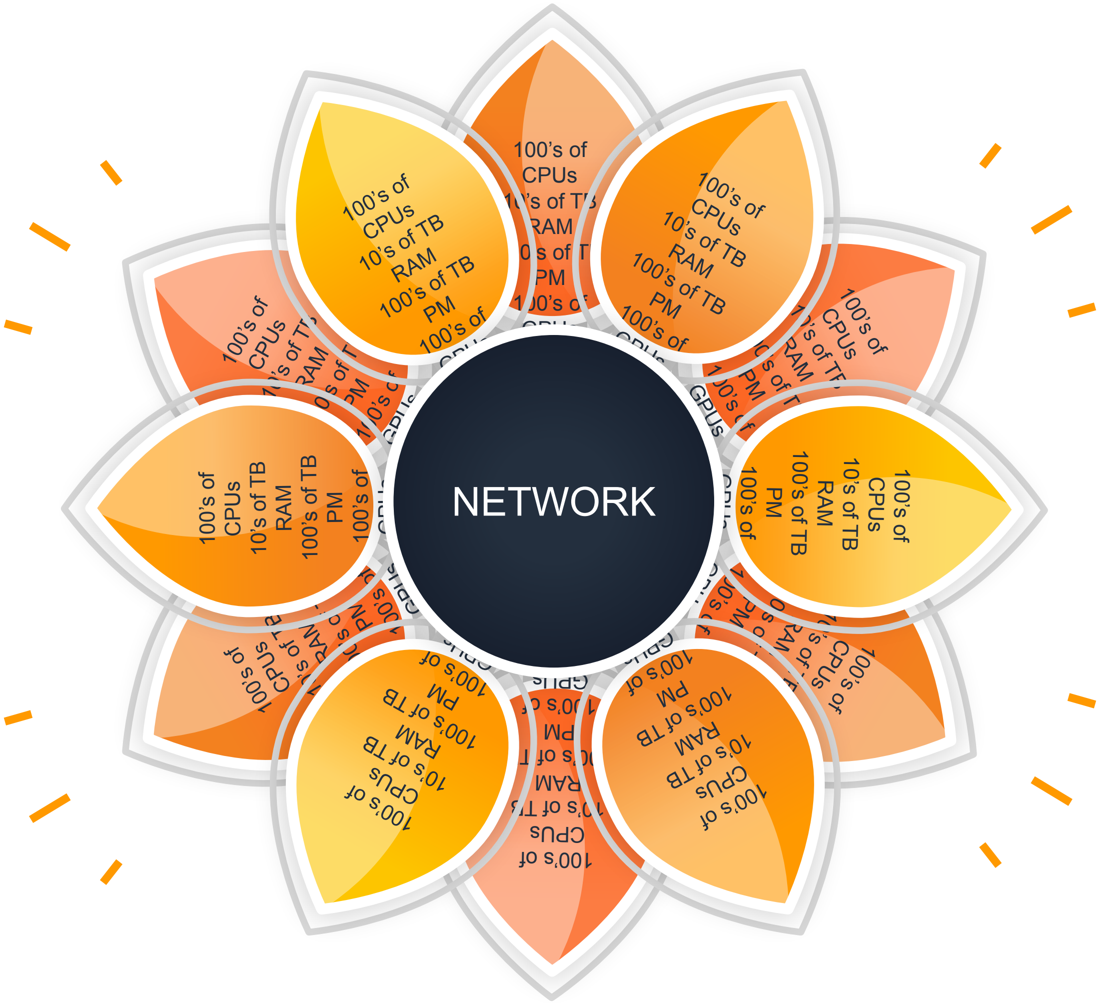
Petabyte scale architecture replicates data THREE WAYS for durability

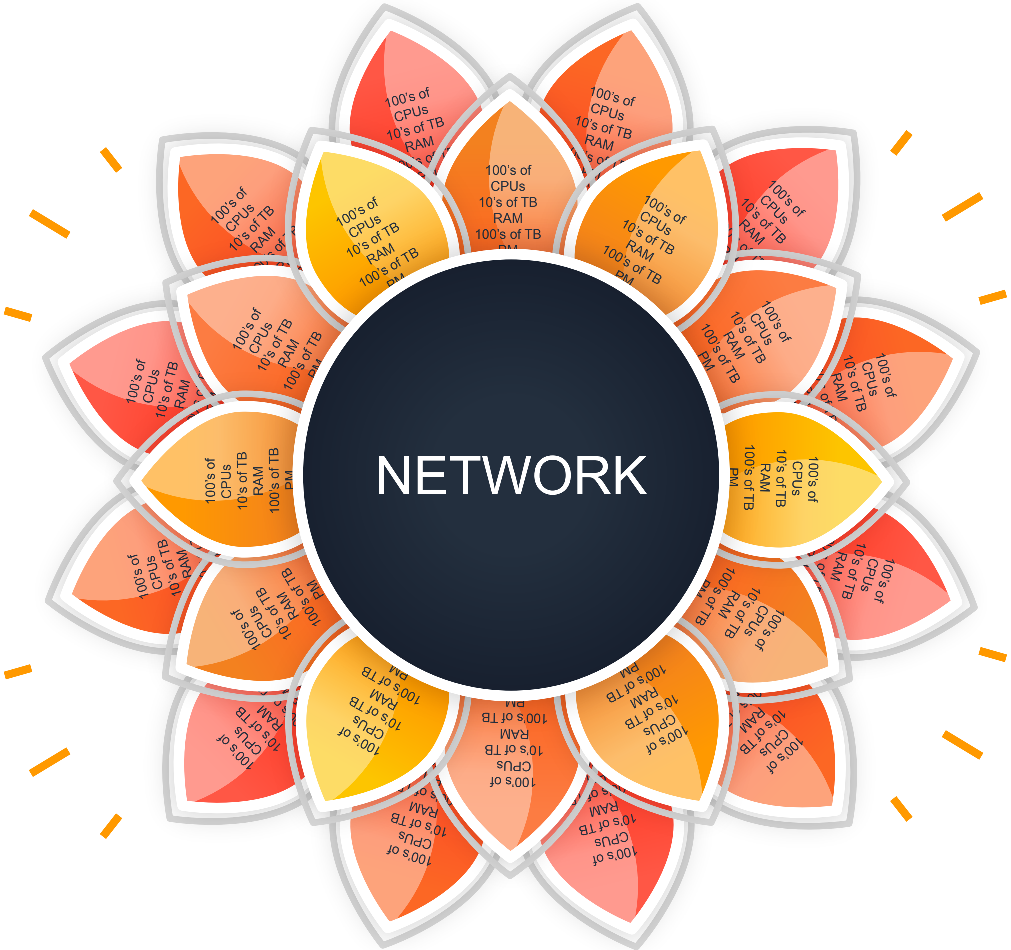
Via a local switch for lowest latency

Petabyte scale architecture replicates data SIX WAYS for resilience

Across three availability zones – like Aurora




NETWORK
100’s of
CPUs
10’s of TB
RAM
100’s of TB
PM
100’s of
GPUs



Petabyte scale architecture replicates data NINE WAYS for resilience across zones and regions

Follows the Netflix architecture for global reach and disaster recovery

Questions

What is the right operating system architecture to support this?

What is the right way to integrate and operate cloud services?

What is the right way to construct a petabyte scale application?

When will we enter the petalithic era?

Petalith

THIS IS A TEASER

I have a lot more ideas

I have been thinking about this for **10+** YEARS

Research

I want to encourage a research project that will end up as a cross-industry open source initiative like Tensorflow or Kubernetes

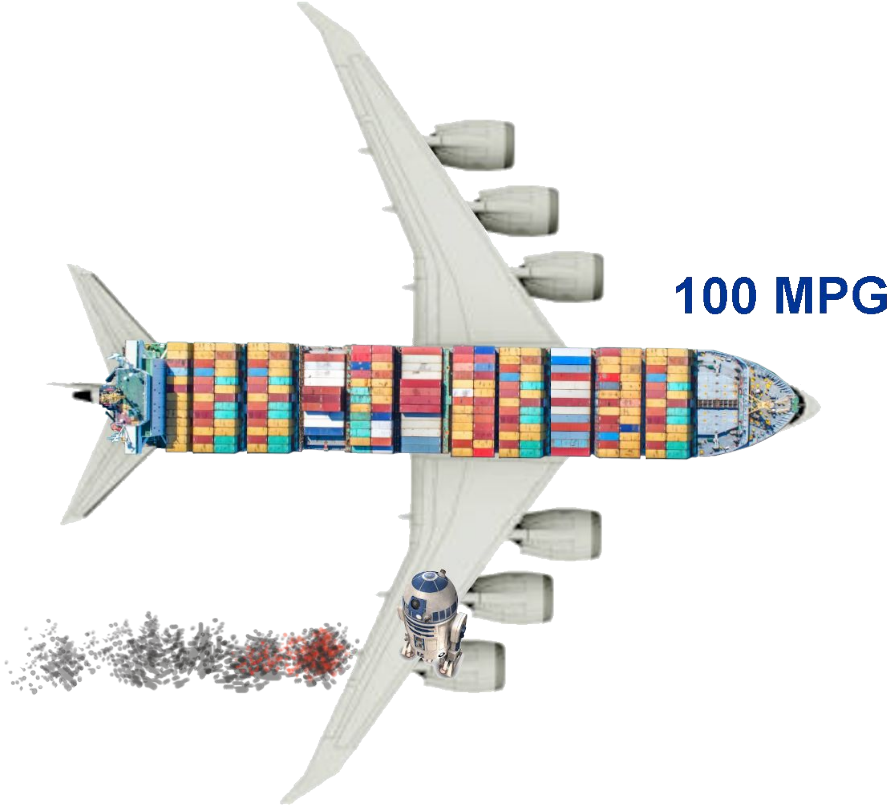
Petalith

THANK YOU

Unavoidable Trade-offs of Distributed Storage Systems in the Cloud

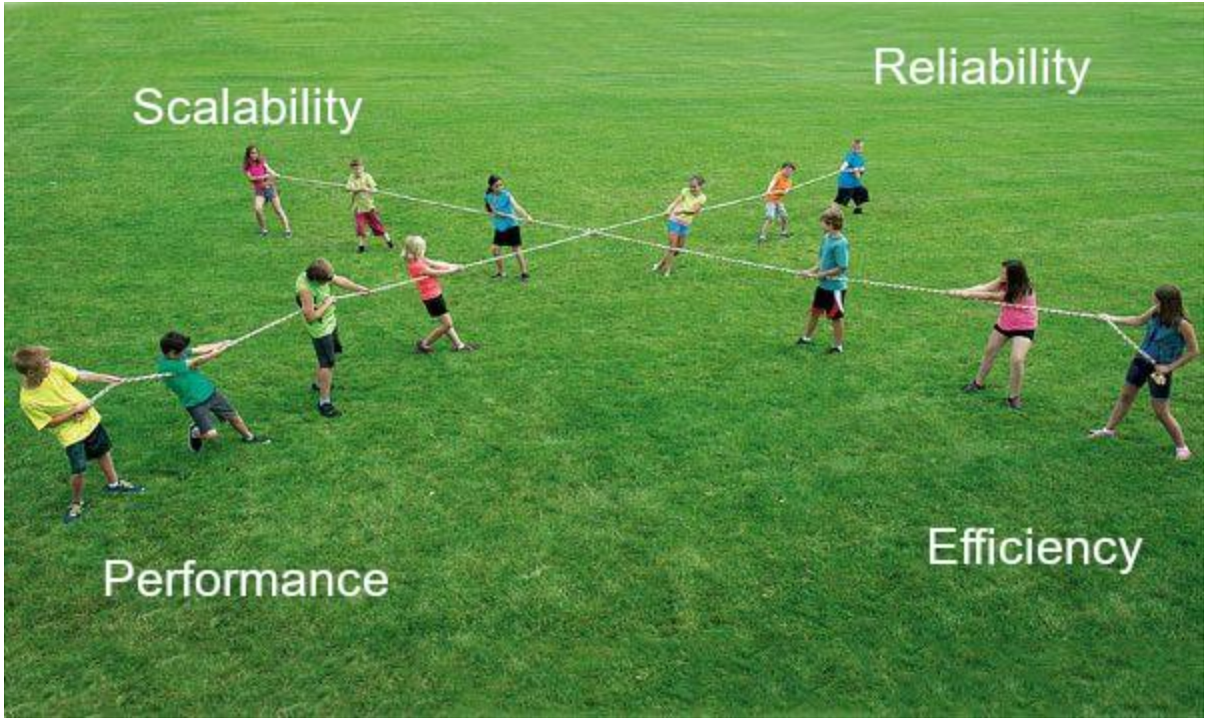
Aleksey Charapko
University of New Hampshire

What Systems Do we Want?



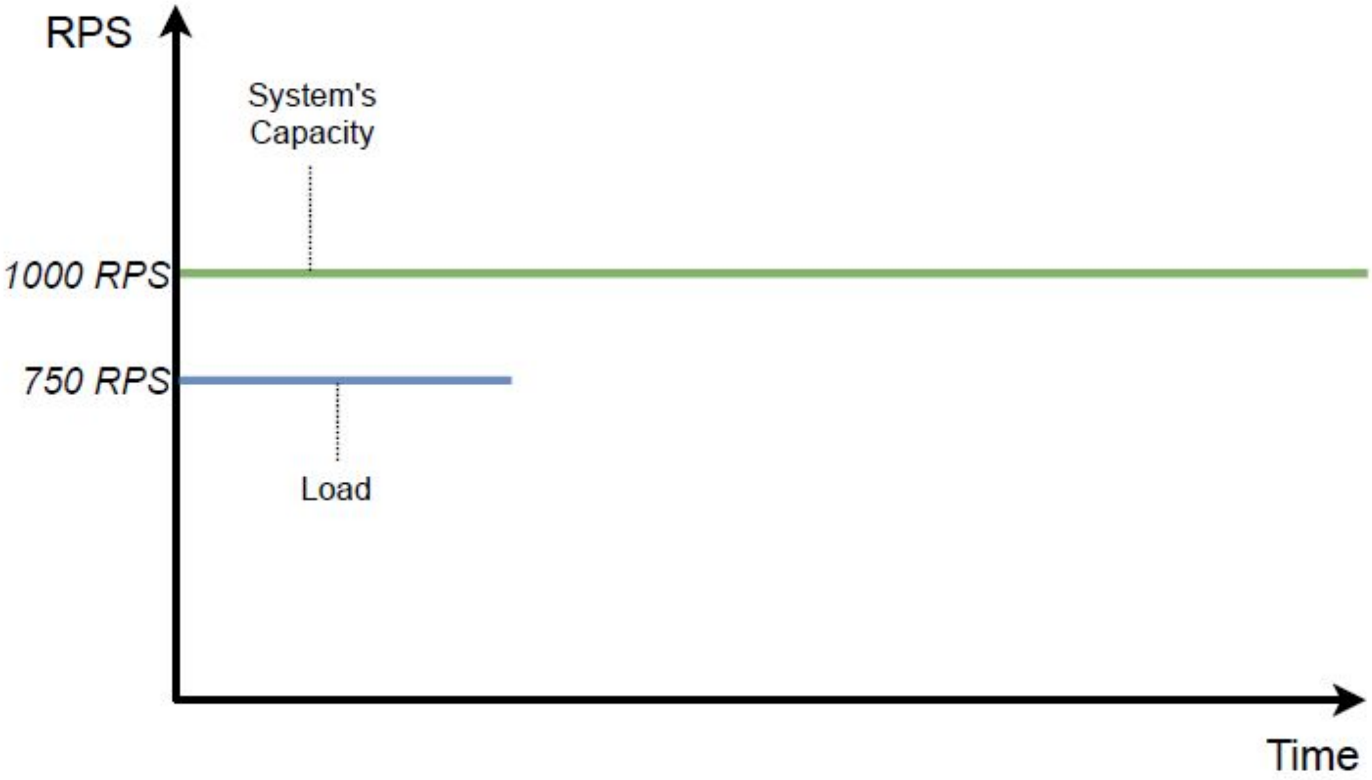
- Performant
- Efficient
- Reliable
- Maintainable
- Secure
- …

Design Tradeoffs

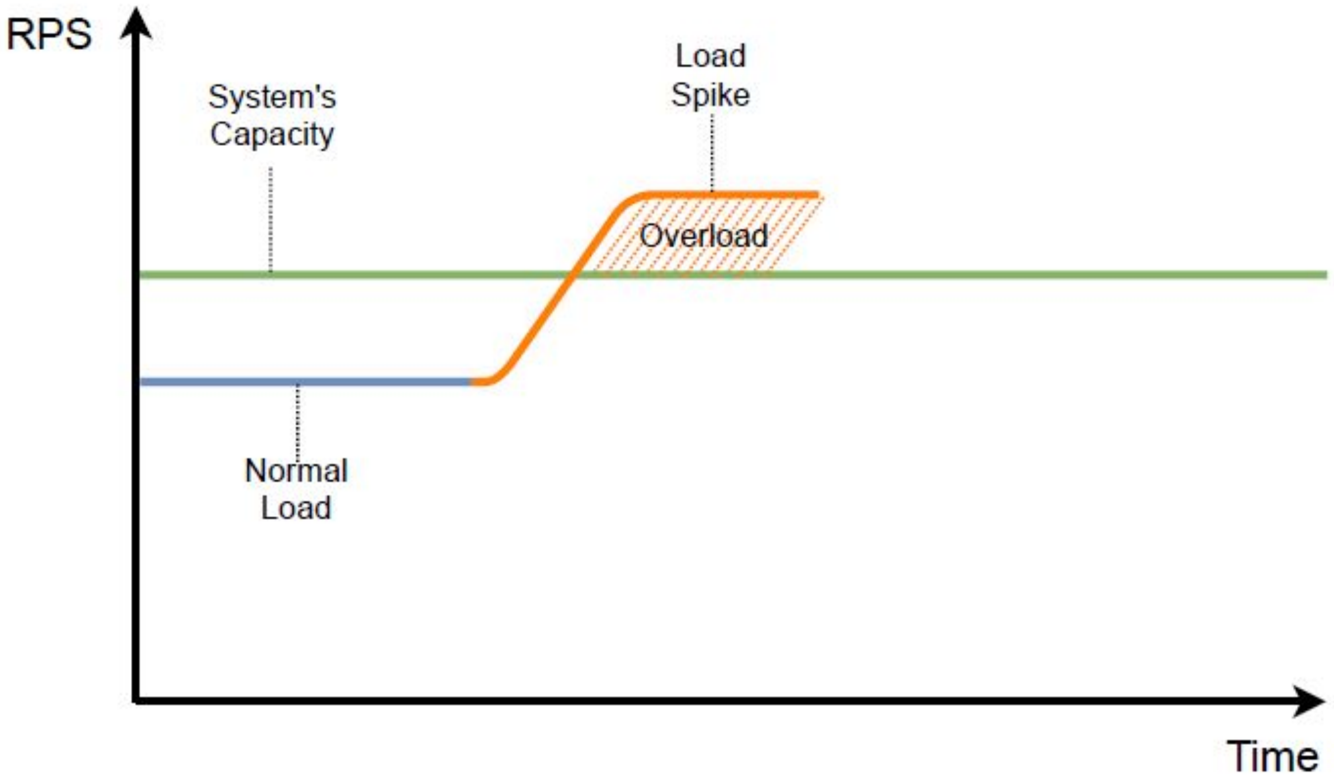


Something must give when one design consideration is in higher priority.

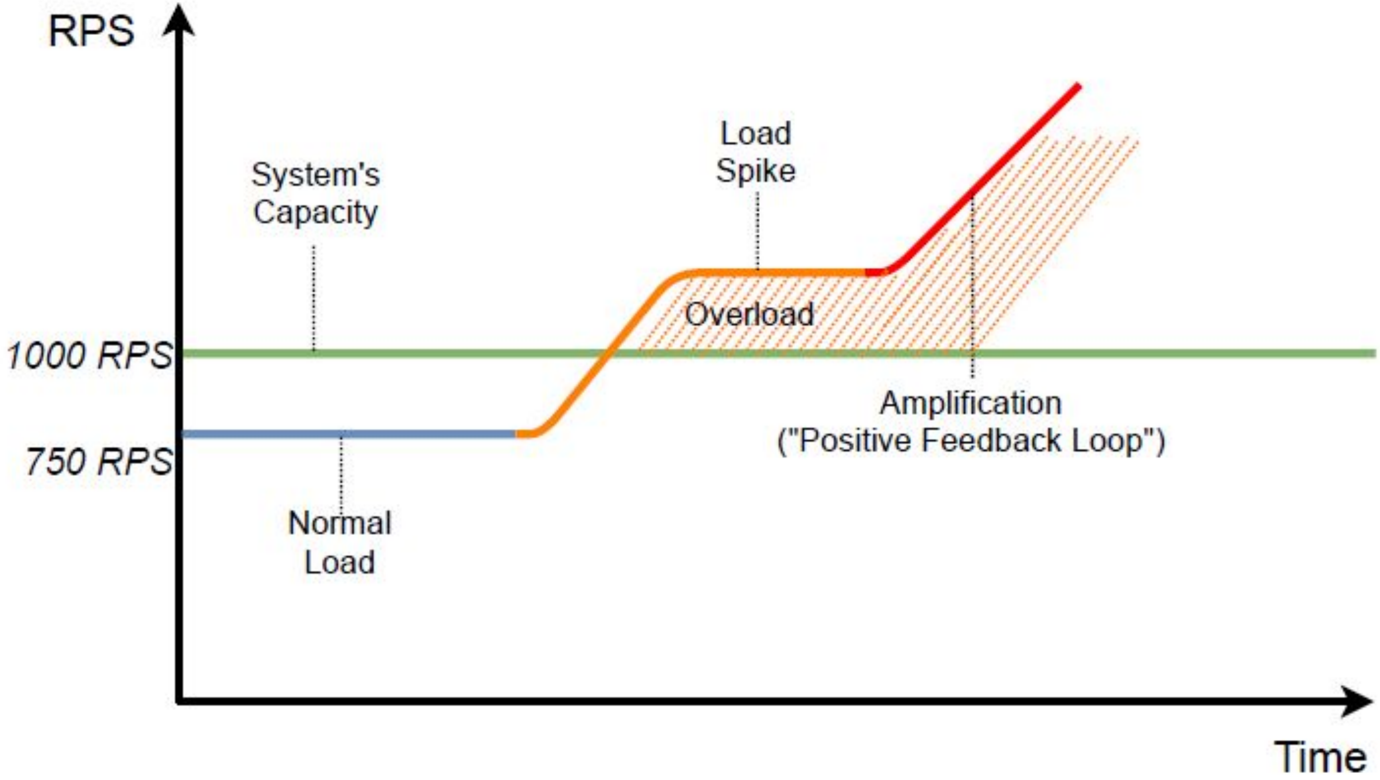
Metastable Failures



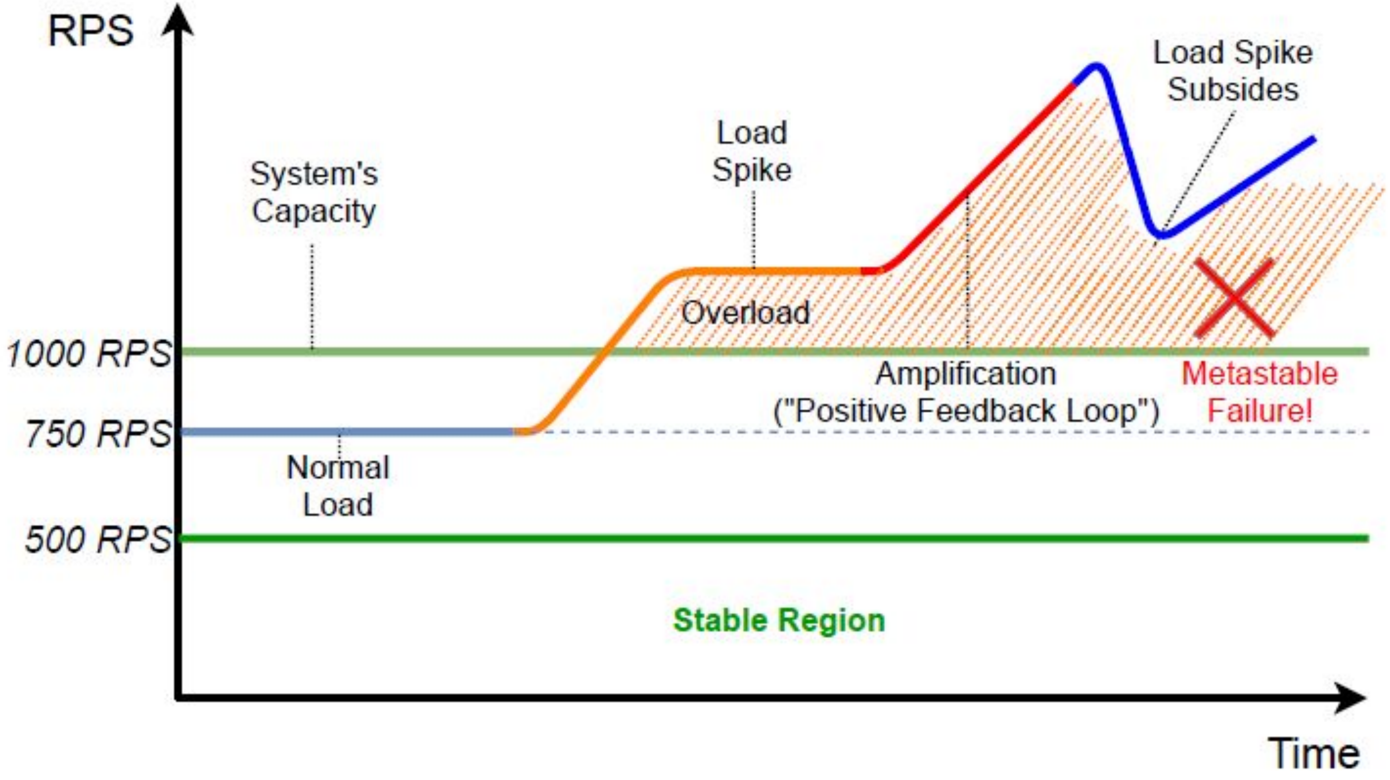
Metastable Failures



Metastable Failures



Metastable Failures



Tradeoffs Examples in Metastable Failures

- Running too close to capacity leaves no “wiggle” room to handle triggers
- Aggressive timeouts & retries to minimize latency on transient failures
- High performance gradients -- over optimized common path to the detriment of the exception path

The Compiler Is the Database

Bruce Lindsay

Firestore: The NoSQL Serverless Database for the Application Developer

Ram Kesavan
Google

SNL sketch
10/1/2022

Serverless Use Case: Extreme Edition

- BeReal
  - Negligible traffic for much of the day
  - Everyone (in a continent) is notified together
  - Everyone uploads their picture in the next 2 min
  - And you view/comment on your friends' pictures
- A how-to (blog link)
  - Created a POC prototype using Google Cloud
  - Simplified auth, storage, notifications, etc.
  - Firestore is the backing database
  - Serverless scale-out and pricing

Firestore: NoSQL Serverless Database

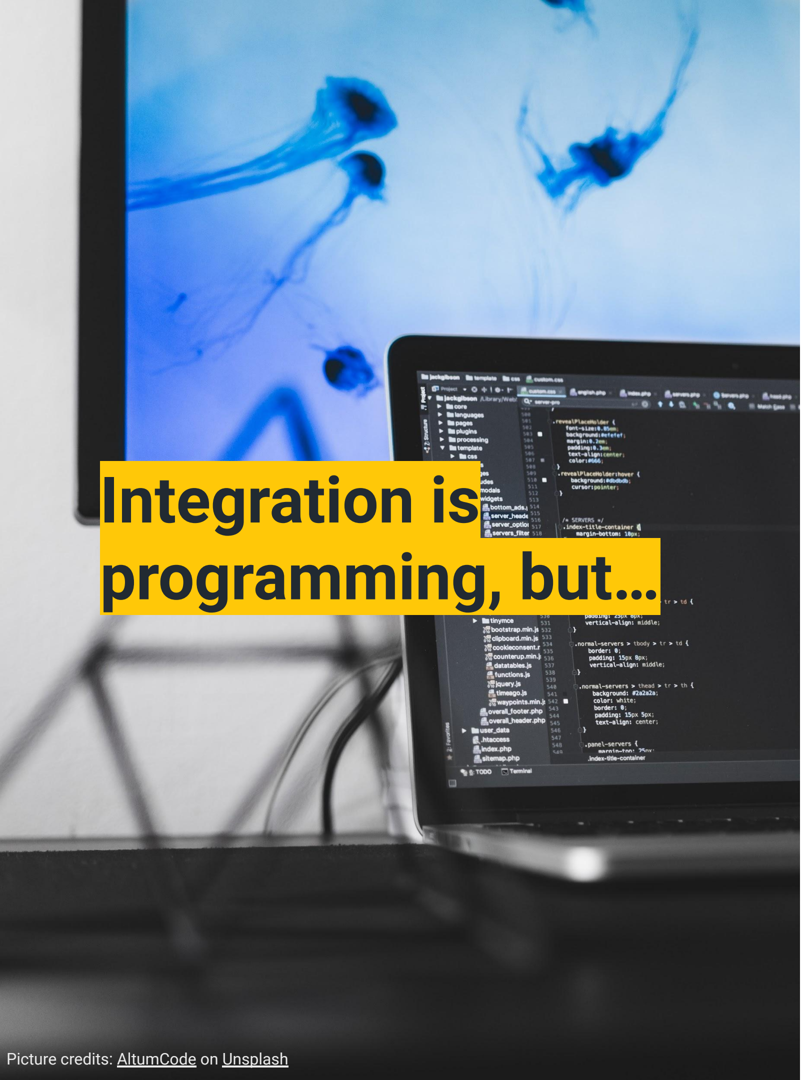
- Firebase client-side SDK libraries
  - Greatly simplifies coding for the app developer
  - Maintains an on-device cache to hide latency to Firestore
  - Offline access reduces to a variant of the default case
- Strong consistency is simpler to code to
  - Spanner storage: ACID semantics, availability, reliability, and scaling
  - Notification stack: updates to continuous queries from each mutation
  - Pay as you use pricing with a (daily) free-tier
- Highly popular
  - 250k+ monthly active application developers
  - 3.5m+ databases have been created
  - Powers 1B+ monthly active end-users

Ballerina in the House

An open-source programming language for network services

Eric Newcomer, CTO

October, 2022



Picture credits: AltumCode on Unsplash

Integration is programming, but…

A visual representation of integration logic is important to communicate with business users.

Domain specific languages (DSLs) have dominated because they provide the right abstractions for integration programming, albeit with limitations when it comes to “regular code” parts of the problem.

Integration programming has lost software engineering best practices because it lives in a closed universe.

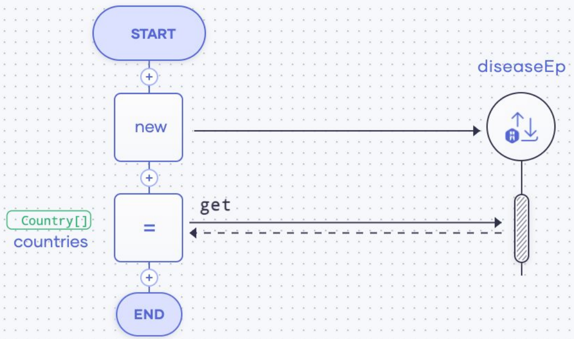
Produces network services

```
import ballerina/http;

configurable int port = ?;

type Country record {
   string country;
   int population;
   int cases;
   int deaths;
};

service / on new http:Listener(port) {
   resource function get countries()
               returns Country[] {
   }

   resource function get countries/[string country]()
               returns Country | http:NotFound {
   }

   resource function post
               countries(@http:Payload Country country)
               returns Country {
   }
}
```

- Application defines service objects and attaches them to Listeners
- Libraries provide protocol-specific Listeners, which receive network input and dispatch to service objects
- Service objects support two interface styles
  - remote methods, named by verbs, support RPC style
  - resources, named by method (e.g. GET) + noun, support RESTful style (used for HTTP and GraphQL)
- Types of service objects methods can used to generate interface descriptions e.g. OpenAPI, GraphQL
- Annotations on service objects enable easy cloud deployment

Consumes network services

```
import ballerina/http;

public function main() returns error? {
   http:Client diseaseData =
       check new (openDiseaseDataURL);
   Country[] countries =
       check diseaseData->get("/countries");
}
```



- Key enabler for sequence diagram view of network interactions
- Outbound network interactions represented by client objects
- Client objects have remote methods that represent outbound interactions with a remote system
- Distinct syntax for calls remote method
- Syntax restrictions make it possible to create a sequence diagram for any function

Data oriented

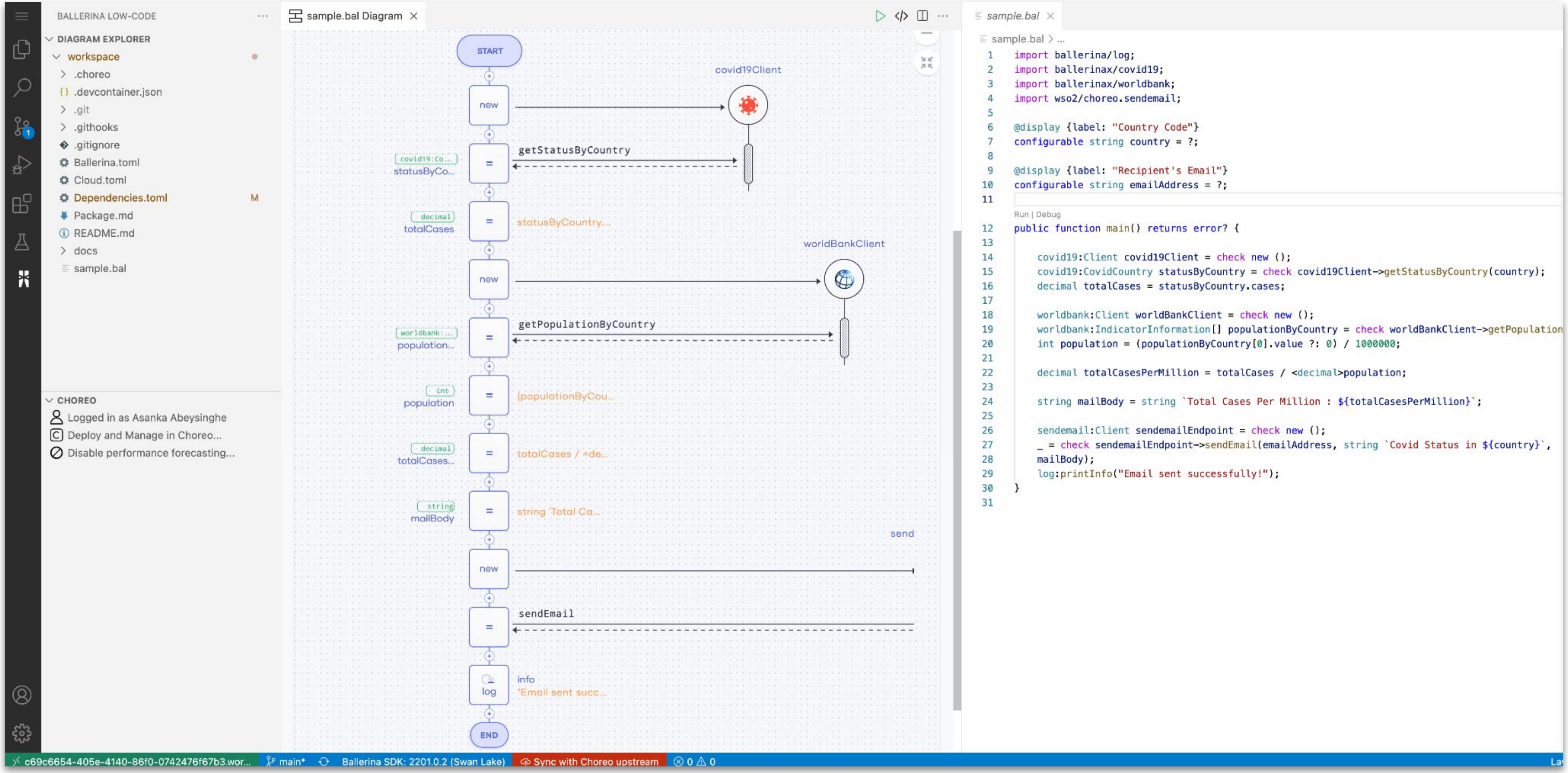
```
// Describes both the payload on the wire
//   and data in memory
type Country record {
   string country;
   int population;
   int cases;
   int deaths;
};

public function main() returns error? {
   http:Client diseaseData =
       check new (openDiseaseDataURL);
   Country[] countries =
       check diseaseData->get("/countries");
}
```

- Object-orientation bundles data with code: wrong approach for network interaction
- Ballerina emphasizes plain data - data that is independent of any code used to process the data
- Ballerina provides objects for internal interfaces, but is not object-oriented
- Ballerina's plain data maps straightforwardly to and from JSON
- Native data types for XML and JSON

Example service & resource syntax with primitives for sequence diagram

```
import ballerina/http;

var clientObj = client object {
    resource function get greeting/[string name]() returns string {
        return "Hello, " + name;
    }

    resource function post game/[string name]/[int players]() returns string
{
        return name + ": " + players.toString();
    }
};

public function main() {
    string name = "Mark";
    string result = clientObj->/greeting/[name];
    // Will print Hello, Mark
    io:println(result);

    [string, int] gameDetails = ["Chess", 2];
    result = clientObj->/game/[...gameDetails].post;
    // Will print Chess: 2
    io:println(result);
}
```

Sequence diagram and code - with round trip - in VS Code



“Swan Lake” Release Feb 2022

- GA quality - completion of long beta for new cloud features
- VS Code plugin enhanced for graphical code editing
  - Edit code -> generate sequence diagram
  - Edit sequence diagram -> generate code
  - Full round tripping
- Code to cloud syntax CL
  - Generate Docker files
  - Generate Kubernetes config
- Extended distributed API programming model
  - Open API (Swagger)
  - gRPC
  - AsyncAPI
- WSO2 Choreo PaaS product built using Ballerina Swan Lake

Ballerina implementations

jBallerina

- Toolchain written using Java
- Compiles to Java bytecodes and runs on a JVM
- Provides Java interoperability
- Current production version

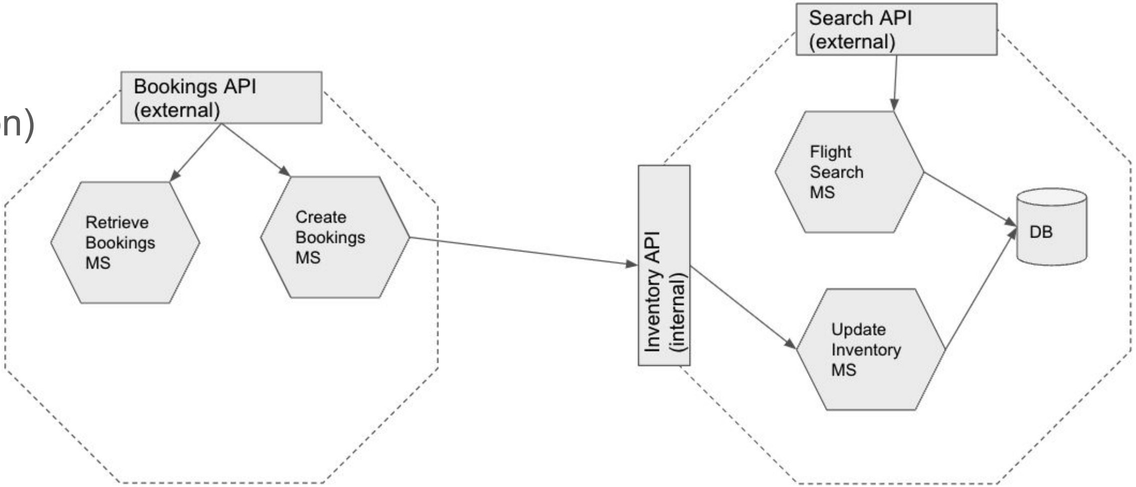
nBallerina

- Cross compilation to native binaries via LLVM
- Toolchain will be shared initially (compiler front-end still in Java) but fully bootstrapped soon
- Provides a C FFI
- ETA: (soon?!?)

Ballerina by Example

Upcoming features

- Persistence abstraction
- Long running transactions
- Workflow
- Data mapping tool (transformation)
- Domain services ->
    - gRPC inside the “domain”
    - HTTPS externally



Thanks!

Further info:

ballerina.io
choreo.io

Building to Buy

Joshua Leners
Two Sigma

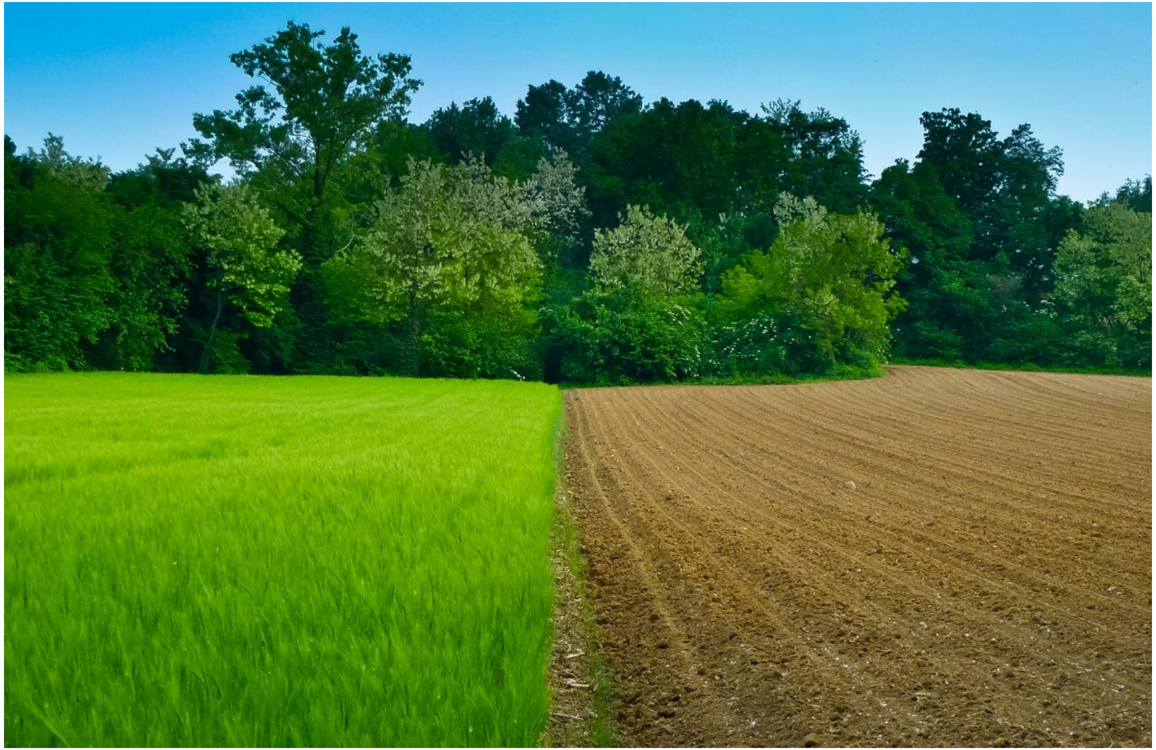
About me

Apostate systems researcher

Engineer at Two Sigma

2nd HPTS

Build vs Buy

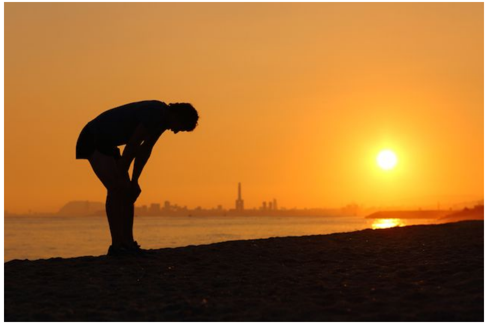


Everything changes

Keep building

Keep buying

Give up



Two Sigma - 2005

Can’t buy S3, HDFS, Cassandra, but we can read GFS paper

We can buy expensive appliances

Choice: Build our own (CelFS)

Two Sigma - 2015

Can’t buy cloud services (connectivity on the roadmap)

What is Celfs good at?

What is it bad at?

Two Sigma 2020

Can buy cloud 🎉

But our users have built to our APIs

And our users have built to our performance profile

What we've learned

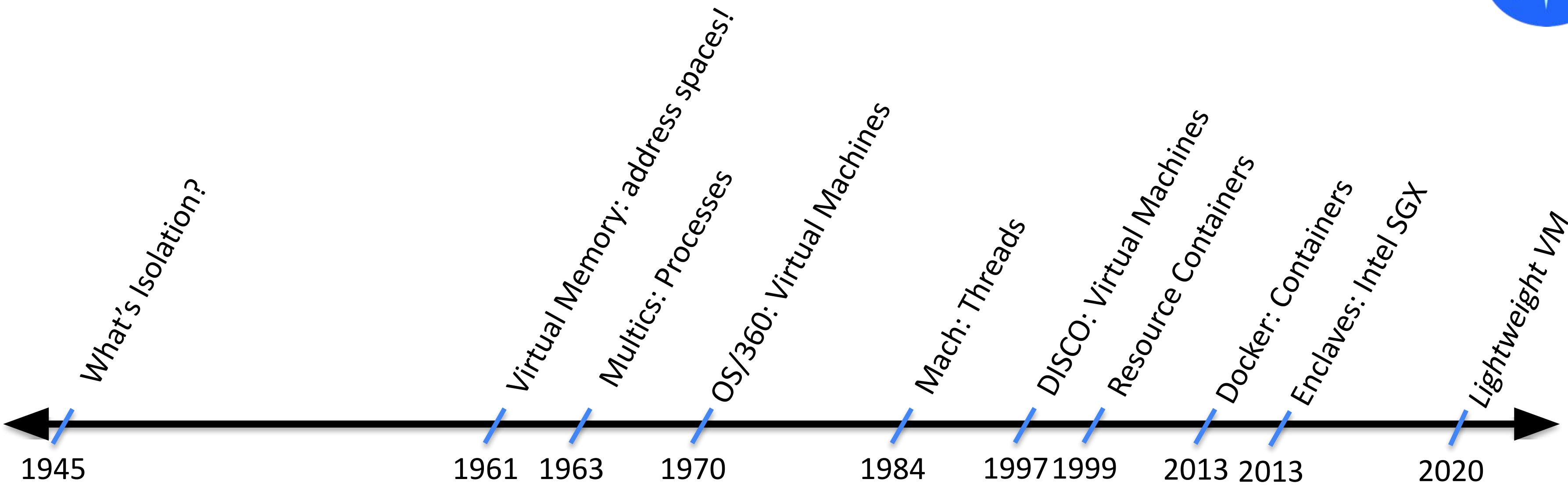
We're all buying, and it's more like subscription than not

Good integration skills are powerful

Déjà vu in OS Isolation

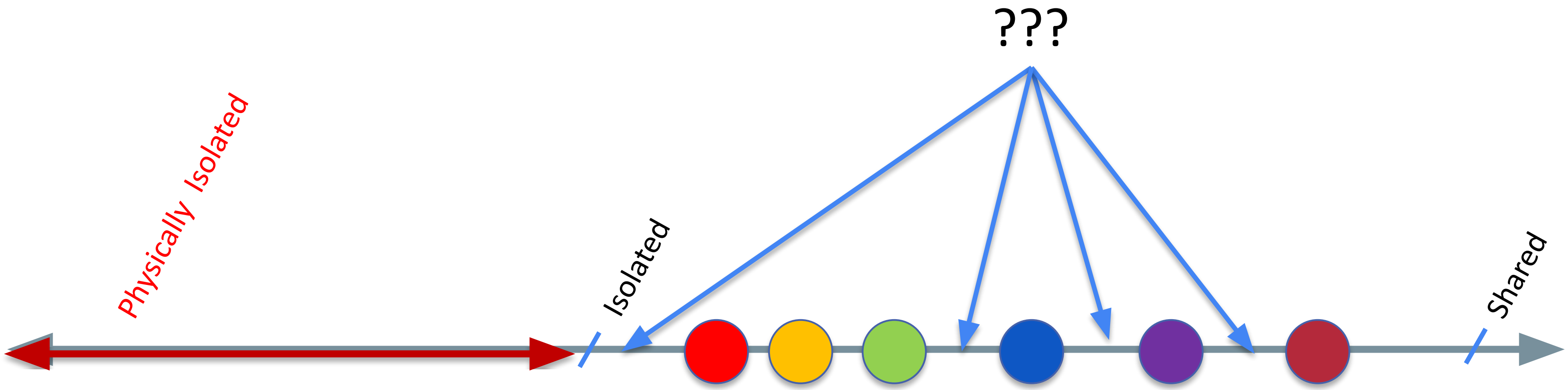
Sid Agrawal
University of British Columbia

Isolation over Time



Resource Isolation Is a spectrum

Consider the
memory resource



Problems with current scenario

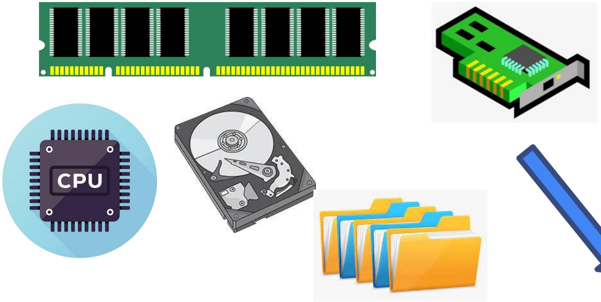
- **Isolation is incremental, but the implementation is not**
  - Increases the engineering cost
  - More bugs
  - Not everyone can afford to do this

Problems with current scenario

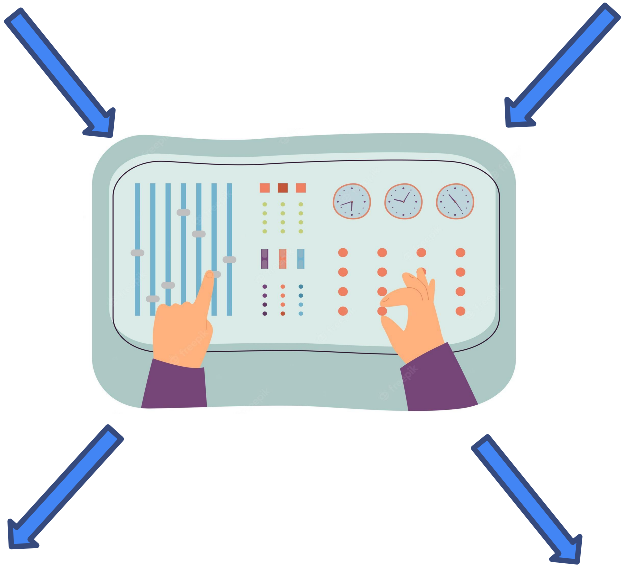
- **No holistic view of the isolation**
  - What is the level of isolation provided by a mechanism?
  - How to specify the desired level of isolation?
  - Too much isolation leads to <u>poor performance</u>.
  - Too less isolation leads to <u>security vulnerabilities</u>.

OSMosis

Identify Resources



Fine-grained
Access Control



Express Existing Mechanisms

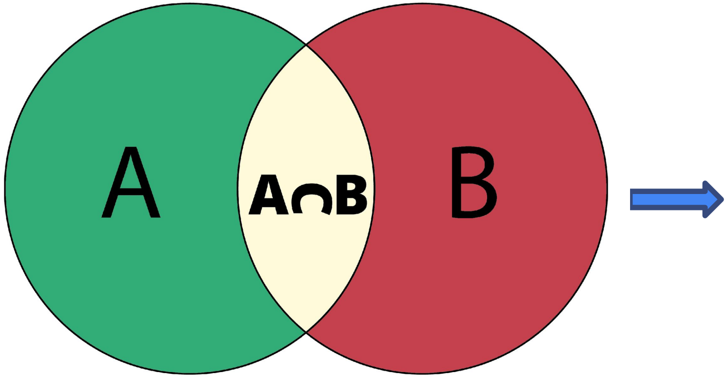
Enable Discovery of
new abstractions

Osmosis: Two Parts

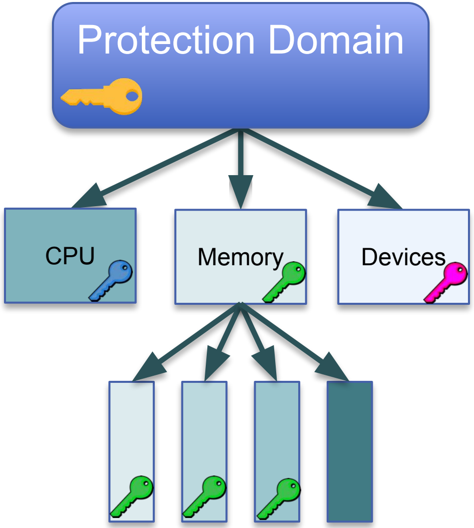
Precisely Defining what is shared (or not)

- Physical Resources
- Virtual Resources
- Underlying State (Kernel/VMM state)

Use Capabilities to enable delegation and revocation of fine grained resources

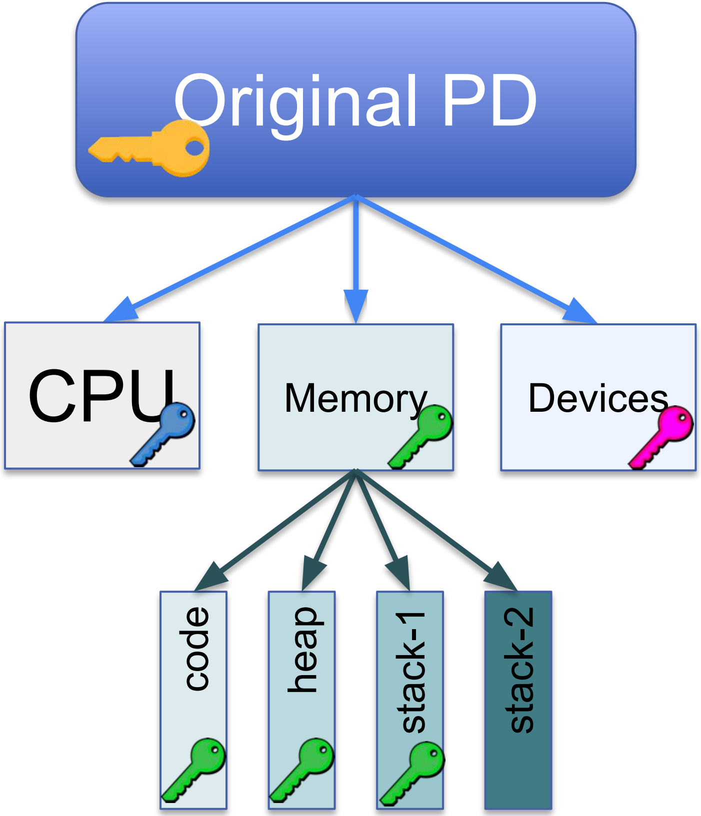


Model Sharing and Isolation



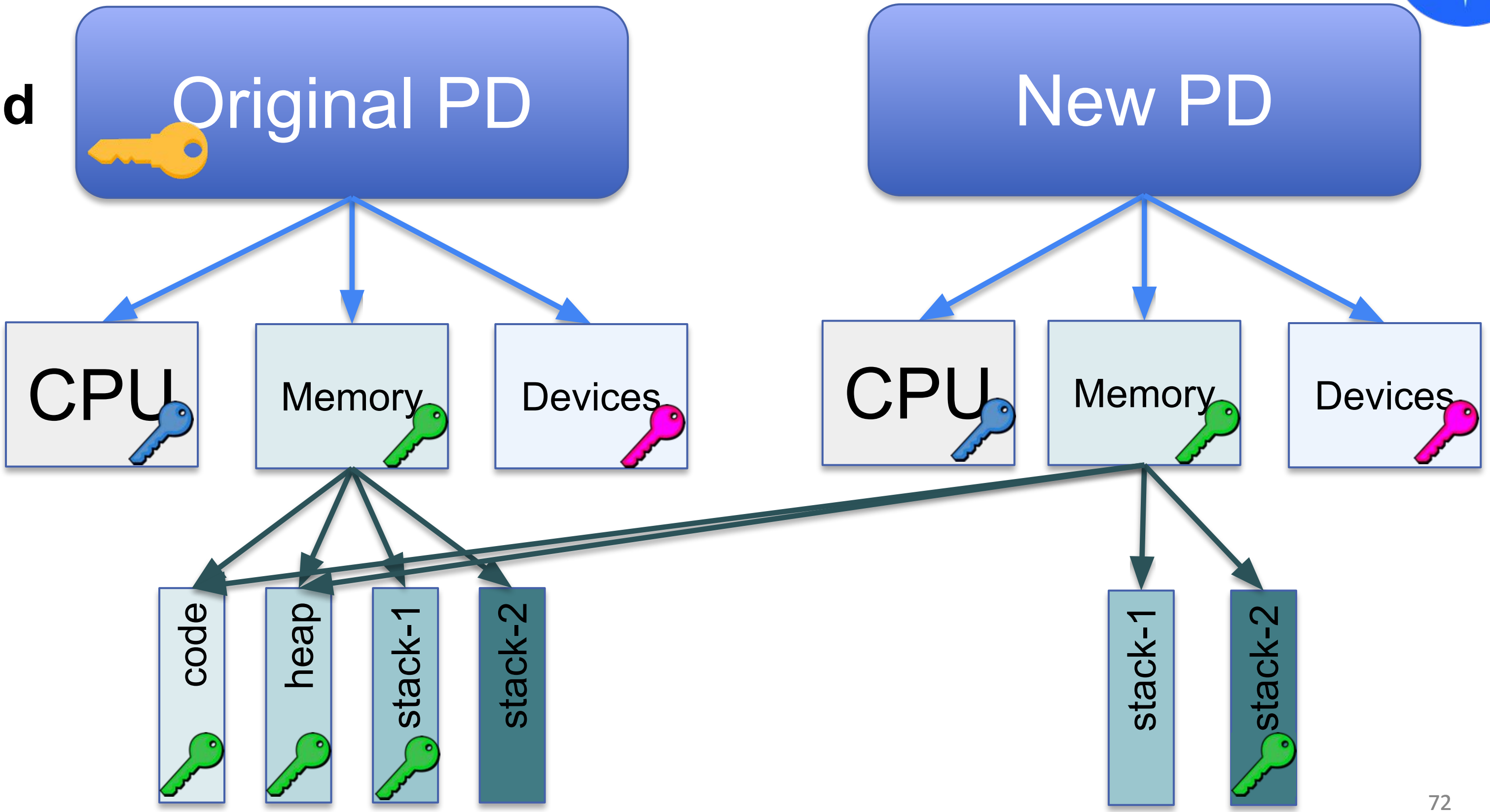
Realize with a Framework

OSmosis Framework



OSmosis Framework

Threads with Isolated Stacks



Open Questions

Shared capabilities or only copied capabilities?

How does our design change if we use something like CHERI?

Is seL4 the right substrate for this work?

Context-Mediated Transactions and Disaggregated Memory

Pankaj Mehra
Elephance Memory

Our Large Context is a Flower of 7-19 dims unrolled in time

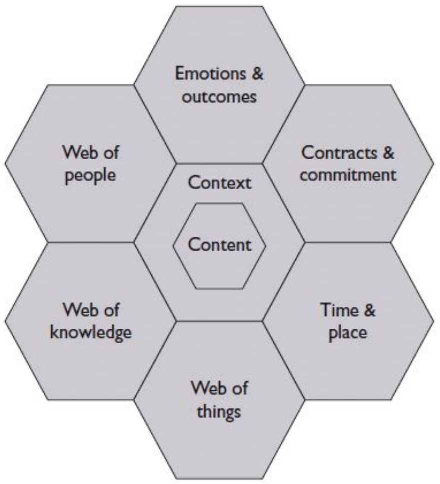
Graph [Databases] deliver contextualization to support new digital transformation initiatives… because messy data without context can dramatically slow down the AI process. *Noel Youhana (Forrester)* April 21, 2021

1995
McCarthy

Facts and Rules that axiomatize a situation and help us reason

Context as Ontology

Reasoning

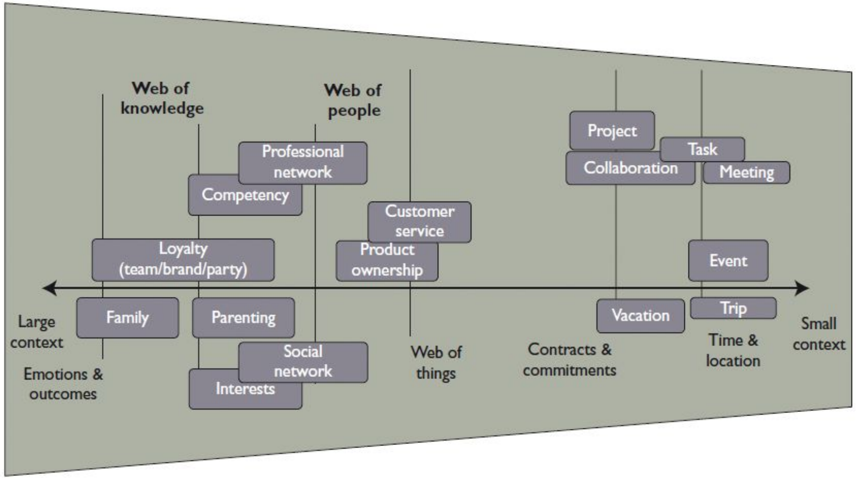


2010
Dey,Cooke

Context as Ontology

Cyber-physical

Context Engines for eCommerce & Advertising



2025
EKG

Context Graphs

Petabyte scale

In-Memory

Index-Free adjacency

Search → **Page Rank, RW**
Data: Crawled Content [GBs]

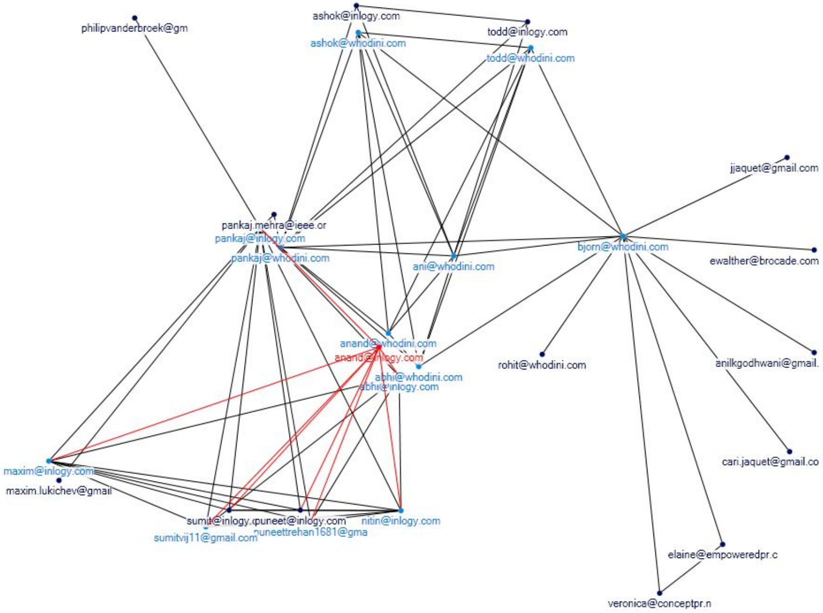
Recomm → **Subgraph Isomorphism**
Data: Users, Products Entities [TBs]

IPAs→ **WSD**
Data: Sound, Speech 10s TB

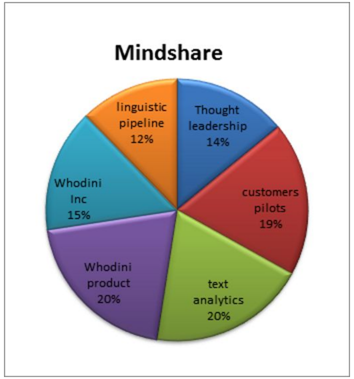
Fraud →
TBs/d x 30-90 d

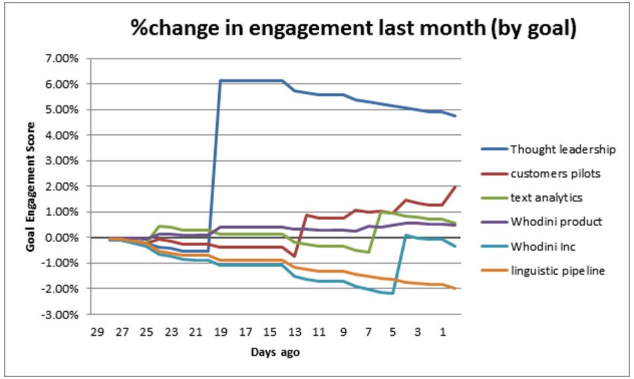
Hadoop | **RDF Triple Store** | **In-Memory Relational** | **In-Memory Graph DB & DNN**

Context Graphs



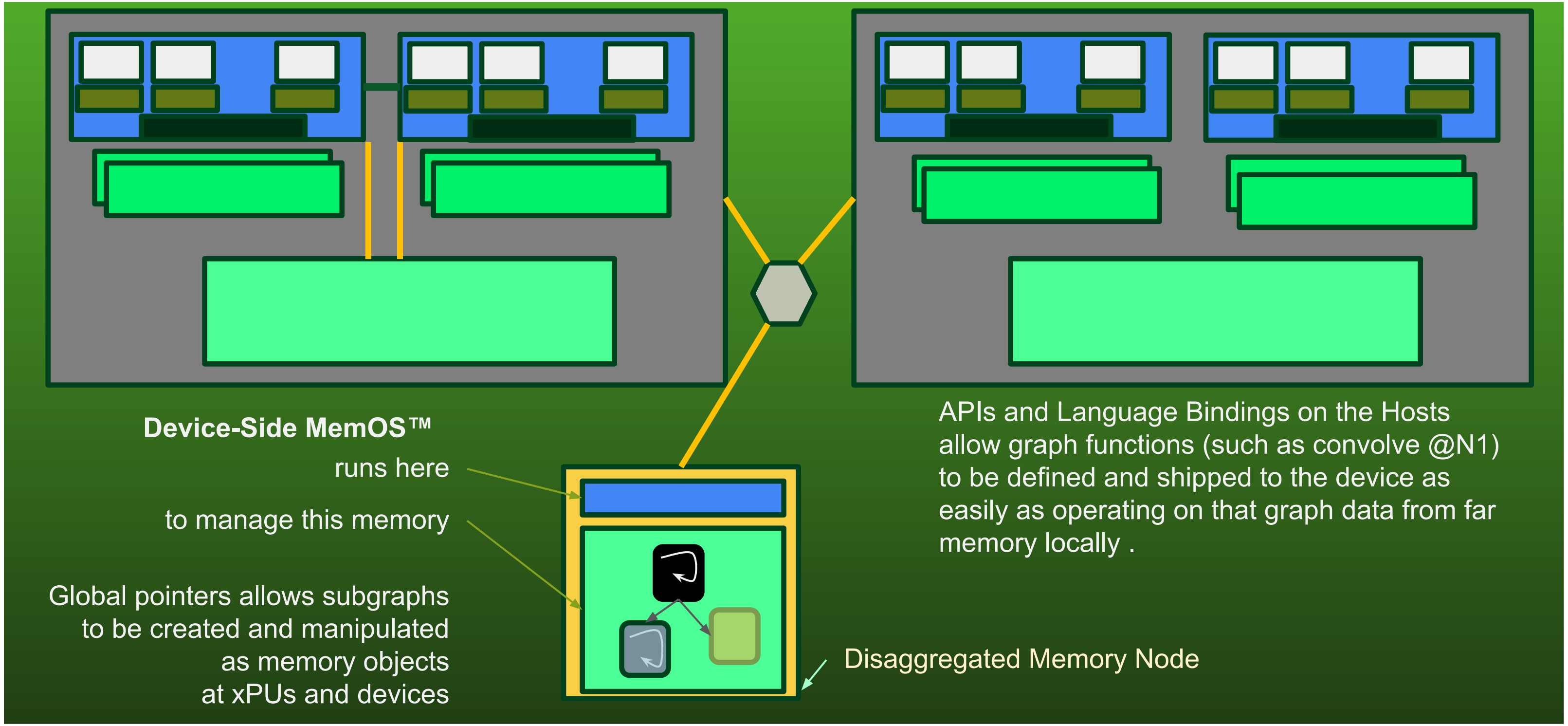
Helped to disambiguate words like *next iteration* by contextualizing and context-clustering on graphs of **Who said what to whom, when**



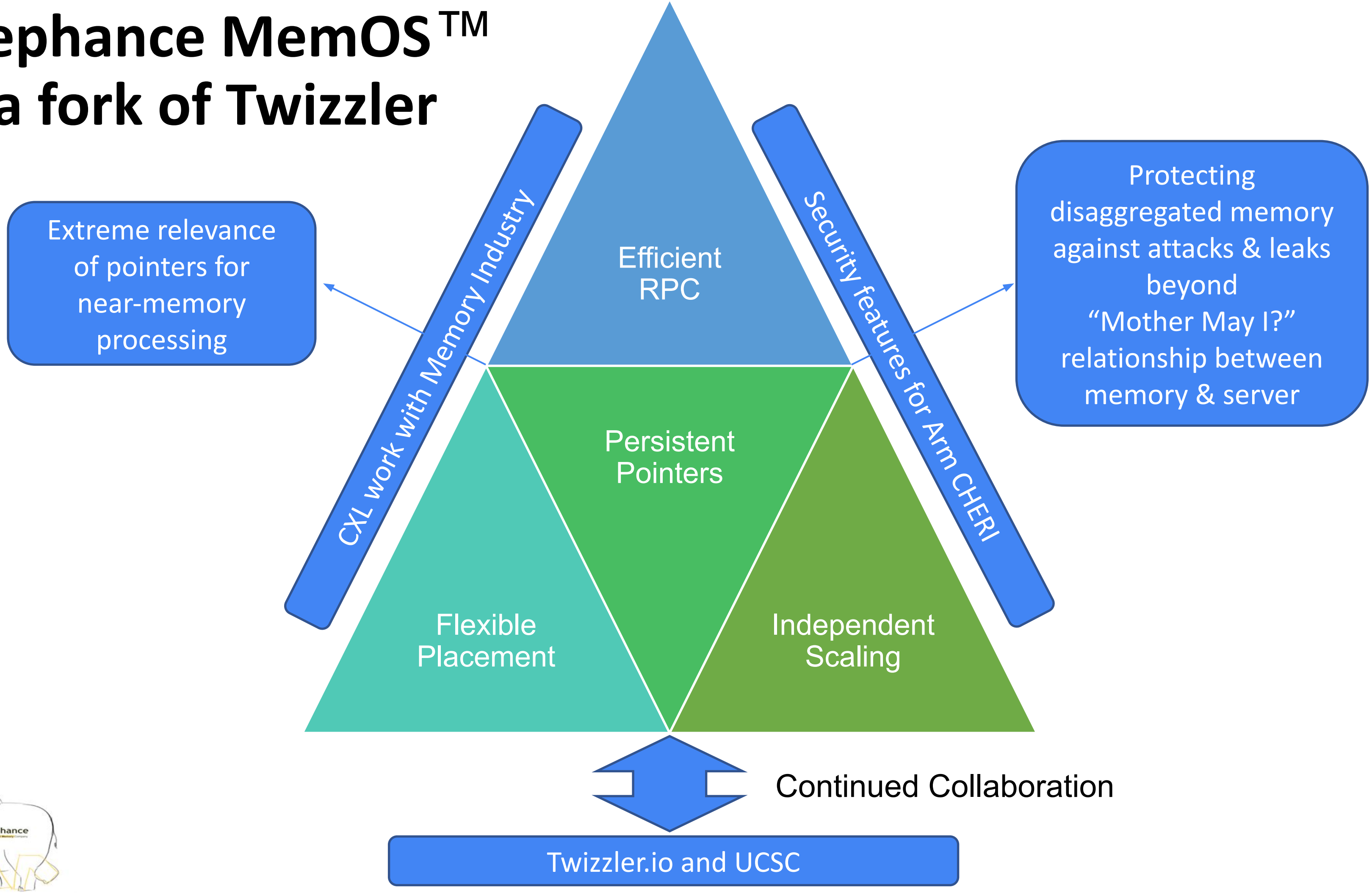


Example. Whodini ('10-13): Work Context extracted from email/calendar by applying Speech Act Theory + 47 algorithms against 210M data points / person / year from the 600,000 words each of us write every year!

Memory too is evolving in response to PB-scale use cases

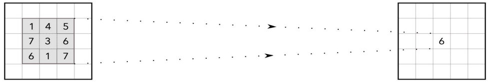


Elephance MemOS™ is a fork of Twizzler

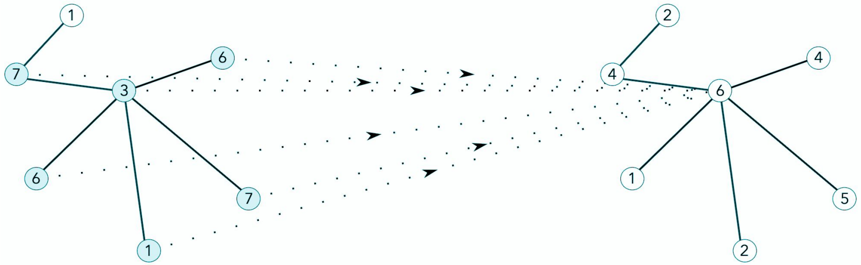


MemOS™ offloads MI graph operations from CPUs, GPUs

MI Operations
Pointer chasing
Convolution →
Clustering
In-DB ML Inf Op
Page Rank / RW
Connected Comps
Search-Accumulate
Shortest Paths
Filter-Aggregate
Compression



Convolution in CNN



Localized Convolution in GNNs

Far Memory without MemOS

CPU chases pointers,
Retrieves neighbor node,
Retrieves local property,
Calculates filter polynomial

Many RTTs,
Low goodput,
Cache pollution,
NW flooding,
Rule of 3 penalty

Far Memory with MemOS

DMN chases pointers,
Retrieves neighbor node,
Retrieves local property,
Calculates filter polynomial
Returns convolved values

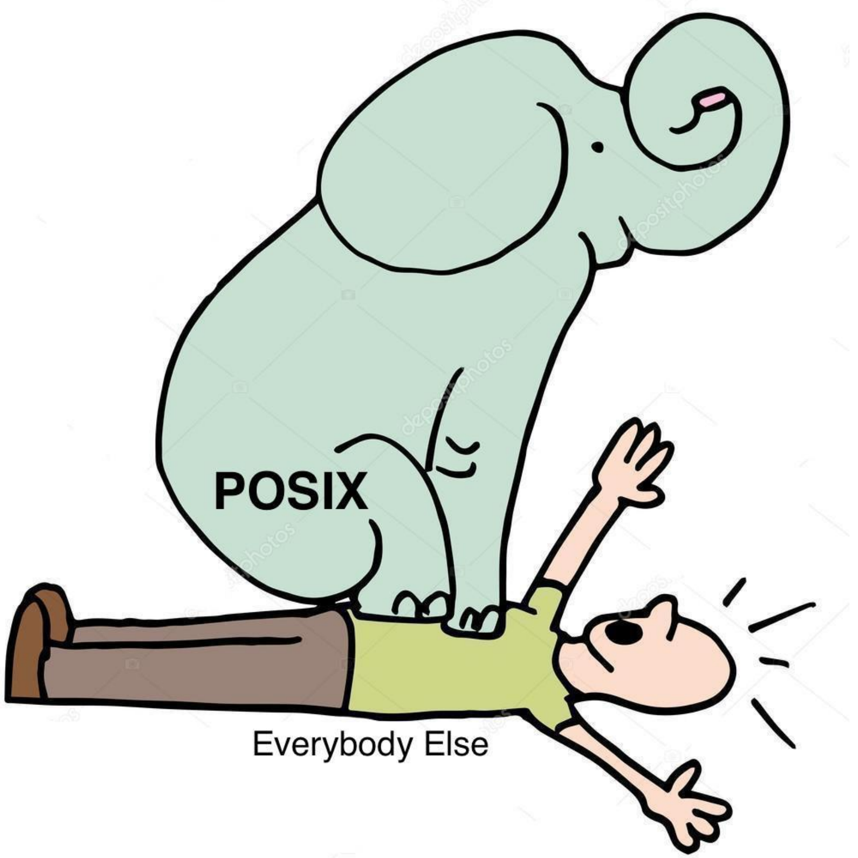
Low latency,
High goodput,
Independent scaling

The Elephant in the Room

George Neville-Neil
Elephance Memory

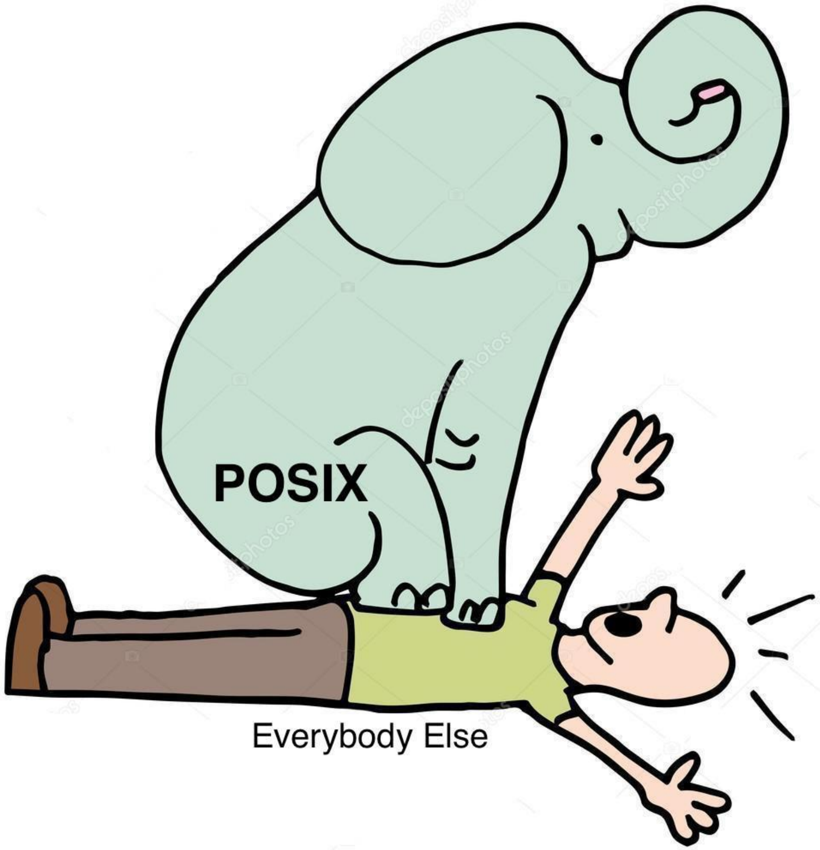
The Good

- Provides a consistent programming paradigm
- Led to unprecedented increase in the amount of software
- Better than the fragmented world of the 1960s-1970s
- Relatively open
  - (some caveats apply, void where prohibited, do not stick in ear.)



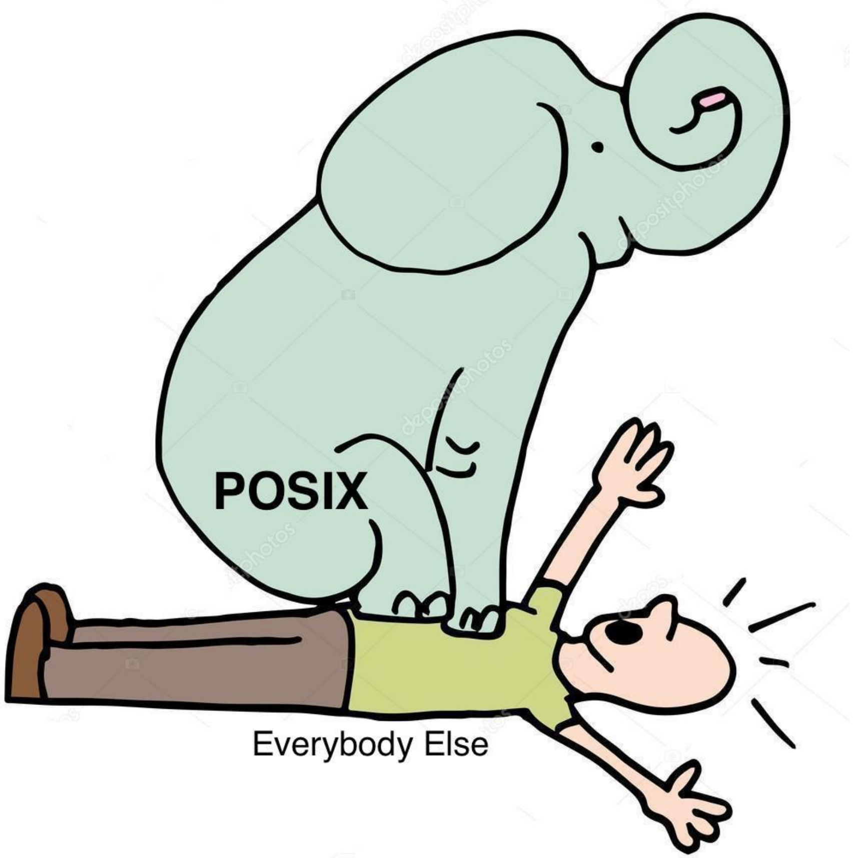
The Not So Good

- Plumbing is too visible
- Hidden assumptions
- Narrows thinking about how we program
- Twists systems to be more like itself.
- A drag on innovation.



Thoughts to Consider

- How do current computers actually work?
- What do current programmers really want?
- What other models are possible?
- Data Oriented Programming
- Re-think the plumbing
  - Don't just hide it



Elephance

BigQuery in 4 minutes and 30 seconds

Justin Levandoski
Google

BigQuery

A serverless, highly scalable, and cost-effective cloud data warehouse

Fully managed, serverless, clusterless

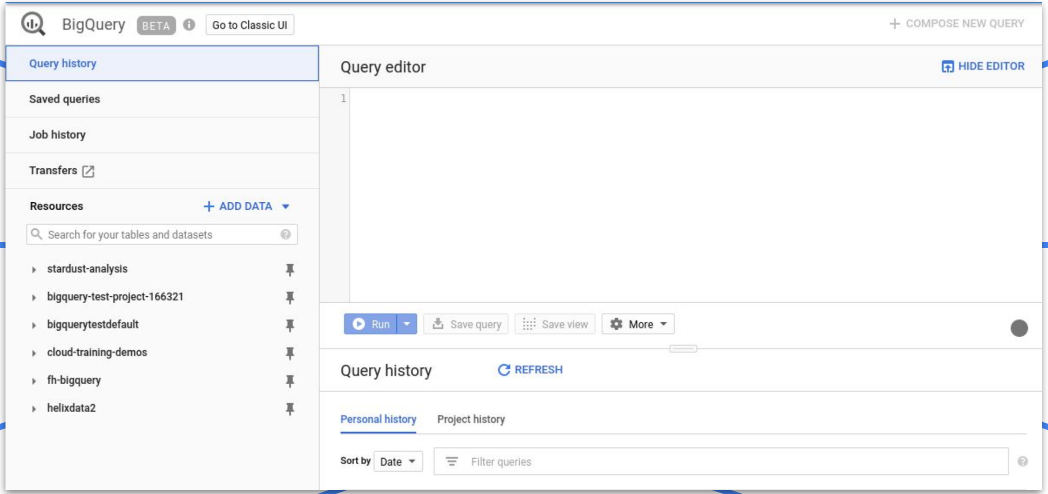
24/7 Service with > 99.99% uptime

Predictable Performance

Petabyte-scale storage and queries

Encrypted, durable

Real-time analytics on streaming data



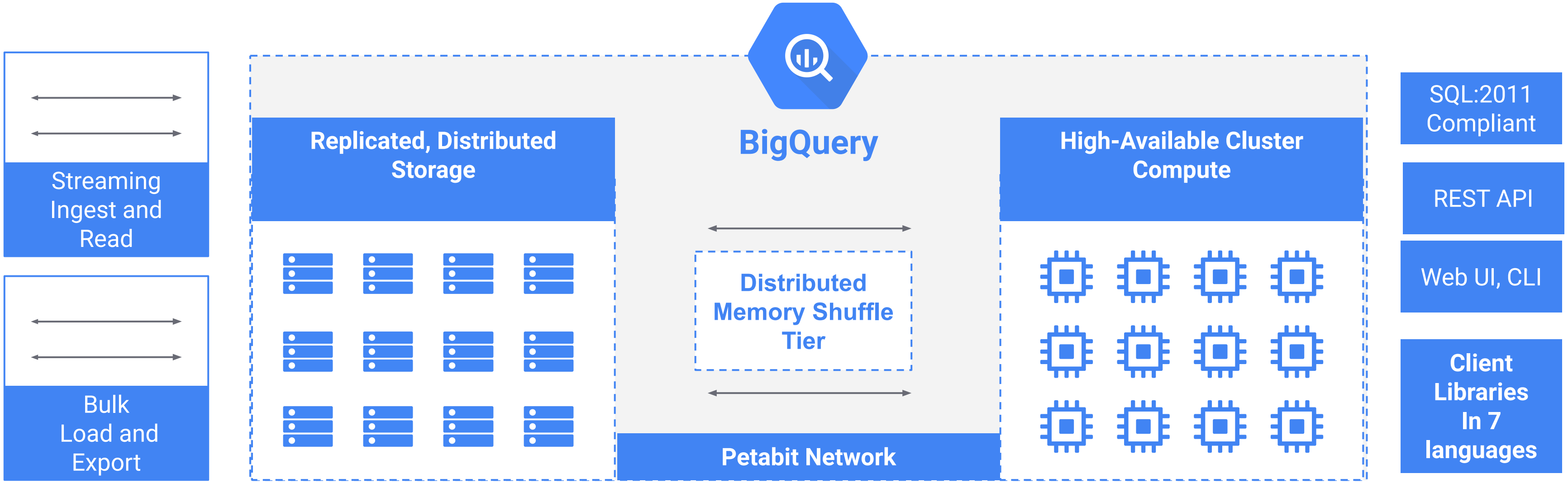
Integrated ML

Easy to use SQL without hints

“BigQuery was serverless before serverless was a thing.”

-Mosha Pasumansky

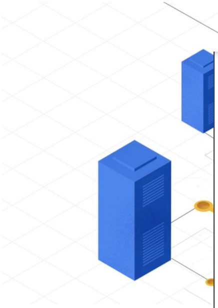
BigQuery Architecture



Google Infrastructure

STORAGE & DATA TRANSFER

Colossus under the hood: a peek into Google's scalable storage system



Dean Hildebrand
Technical Director, Office of the CTO, Google Cloud

Denis Serenyi
Tech Lead, Google Cloud Storage

April 19, 2021

You trust Goo
the same und
the same stor
popular produ

That foundati
ecosystem of

Large-scale cluster management at Google with Borg

Abhishek Verma† Luis Pedrosa‡ Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes
Google Inc.

Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture and features, important design decisions, a quantitative analysis of some of its policy decisions, and a qualitative examination of lessons learned from a decade of operational experience with it.

1. Introduction

The cluster management system we internally call Borg admits, schedules, starts, restarts, and monitors the full range of applications that Google runs. This paper explains how.

Borg provides three main benefits: it (1) hides the details of resource management and failure handling so its users can focus on application development instead; (2) operates with very high reliability and availability, and supports applications that do the same; and (3) lets us run workloads across tens of thousands of machines effectively. Borg is not the first system to address these issues, but it's one of the few operating at this scale, with this degree of resiliency and completeness. This paper is organized around these topics, con-

† Work done while author was at Google.
‡ Currently at University of Southern California.

EuroSys'15, April 21–24, 2015, Bordeaux, France.
Copyright is held by the owner/author(s).
ACM 978-1-4503-3238-5/15/04.
http://dx.doi.org/10.1145/2741948.2741964

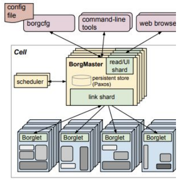


Figure 1: The high-level architecture of Borg. *Only a t of the thousands of worker nodes are shown.*

cluding with a set of qualitative observations we l
from operating Borg in production for more than

2. The user perspective

Borg's users are Google developers and system a
tors (site reliability engineers or SREs) that run
applications and services. Users submit their wo
in the form of *jobs*, each of which consists of o
tasks that all run the same program (binary). Eac
in one Borg *cell*, a set of machines that are ma
unit. The remainder of this section describes the
tures exposed in the user view of Borg.

2.1 The workload

Borg cells run a heterogenous workload with two
The first is long-running services that should "
down, and handle short-lived latency-sensitive r
few μs to a few hundred ms). Such services ar
end-user-facing products such as Gmail, Google
web search, and for internal infrastructure serv
BigTable). The second is batch jobs that take fr
seconds to a few days to complete; these are muc
sitive to short-term performance fluctuations. The
mix varies across cells, which run different mixes
tions depending on their major tenants (e.g., som
quite batch-intensive), and also varies over time:

Dremel: A Decade of Interactive SQL Analysis at Web Scale*

Sergey Melnik, Andrey Gubarev,
Jing Jing Long, Geoffrey Romer,
Shiva Shivakumar, Matt Tolton,
Theo Vassilakis
Original authors of VLDB 2010 Dremel paper

Hossein Ahmadi, Dan Delorev,
Slava Min, Mosh
Jeff S
Googl
dremel-tot-pape

ABSTRACT

Google's Dremel was one of the first systems that combined a set of architectural principles that have become a common practice in today's cloud-native analytics tools, including disaggregated storage and compute, in situ analysis, and columnar storage for semistructured data. In this paper, we discuss how these ideas evolved in the past decade and became the foundation for Google BigQuery.

PVLDB Reference Format:
Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, Hossein Ahmadi, Dan Delorey, Slava Min, Mosha Pasumansky, and Jeff Shute. Dremel: A Decade of Interactive SQL Analysis at Web Scale. *PVLDB*, 13(12): 3461-3472, 2020.
DOI: https://doi.org/10.14778/3415478.3415568

1. INTRODUCTION

Dremel is a distributed system for interactive data analysis that was first presented at VLDB 2010 [32]. That same year, Google launched BigQuery, a publicly available analytics service backed by Dremel. Today, BigQuery is a fully-managed, serverless data warehouse that enables scalable analytics over petabytes of data.[1] It is one of the fastest growing services on the Google Cloud Platform.

A major contribution of papers originating from the industry in the past decade, including the Dremel paper, is to demonstrate what kind of systems can be built using state-of-the-art private clouds. This body of work both reduced the risk of exploring similar routes and identified viable directions for future research. Introducing the journal version of the paper [33], Mike Franklin pointed out that it was "eye-opening" to learn that Google engineers routinely analysed massive data sets with processing throughputs in the range of 100 billion records per second [20]. His main take-away was that simply throwing hardware at the problem was not sufficient. Rather, it was critical to deeply understand the structure of the data

*Invited contribution for the VLDB 2020 Test of Time Award given to the VLDB 2010 paper "Dremel: Interactive Analysis of Web-Scale Datasets" [32]
[1]https://cloud.google.com/bigquery

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415568

and how it would be used. I
tion that the data volumes d
come relevant to more organi
ing edge" becomes common
various opportunities for opt

This paper focuses on Dre
ciples. Much of the overall
some of these principles tur
now considered best practic
ogy trends highlighted in the
Database Research [1], the
are:

- *SQL*: [1] reports that a
style APIs as the predo
Dremel's initial SQL-
compliant SQL backed
with other Google pro
- *Disaggregated compu*
verged on an architect
to analyze data in clou
compute from storage,
- *In situ analysis*: Data
lar, in which a variety
data, to curate it or ex
the results back in the
ational systems. Drem
shared data access uti
data processing system
with SQL-based analy
- *Serverless computing*:
sources, the industry
provide extreme elast
managed internal serv
pay-per-use economic
ported to BigQuery.
- *Columnar storage*: W
mercial data analytic
Dremel introduced a n
eralized the applicabi
tional and semistructu

This paper is structured
we explain the original mot

[2]https://cloud.google.com/sp

Spanner: Google's Globally-Distributed Database

James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, Dale Woodford

Google, Inc.

Abstract

Spanner is Google's scalable, multi-version, globally-distributed, and synchronously-replicated database. It is the first system to distribute data at global scale and support externally-consistent distributed transactions. This paper describes how Spanner is structured, its feature set, the rationale underlying various design decisions, and a novel time API that exposes clock uncertainty. This API and its implementation are critical to supporting external consistency and a variety of powerful features: non-blocking reads in the past, lock-free read-only transactions, and atomic schema changes, across all of Spanner.

1 Introduction

Spanner is a scalable, globally-distributed database designed, built, and deployed at Google. At the highest level of abstraction, it is a database that shards data across many sets of Paxos [21] state machines in datacenters spread all over the world. Replication is used for global availability and geographic locality; clients automatically failover between replicas. Spanner automatically reshards data across machines as the amount of data or the number of servers changes, and it automatically migrates data across machines (even across datacenters) to balance load and in response to failures. Spanner is designed to scale up to millions of machines across hundreds of datacenters and trillions of database rows.

Applications can use Spanner for high availability, even in the face of wide-area natural disasters, by replicating their data within or even across continents. Our initial customer was F1 [35], a rewrite of Google's advertising backend. F1 uses five replicas spread across the United States. Most other applications will probably replicate their data across 3 to 5 datacenters in one geographic region, but with relatively independent failure modes. That is, most applications will choose lower latency over higher availability, as long as they can survive 1 or 2 datacenter failures.

Spanner's main focus is managing cross-datacenter replicated data, but we have also spent a great deal of time in designing and implementing important database features on top of our distributed-systems infrastructure. Even though many projects happily use Bigtable [9], we have also consistently received complaints from users that Bigtable can be difficult to use for some kinds of applications: those that have complex, evolving schemas, or those that want strong consistency in the presence of wide-area replication. (Similar claims have been made by other authors [37].) Many applications at Google have chosen to use Megastore [5] because of its semi-relational data model and support for synchronous replication, despite its relatively poor write throughput. As a consequence, Spanner has evolved from a Bigtable-like versioned key-value store into a temporal multi-version database. Data is stored in schematized semi-relational tables; data is versioned, and each version is automatically timestamped with its commit time; old versions of data are subject to configurable garbage-collection policies; and applications can read data at old timestamps. Spanner supports general-purpose transactions, and provides a SQL-based query language.

As a globally-distributed database, Spanner provides several interesting features. First, the replication configurations for data can be dynamically controlled at a fine grain by applications. Applications can specify constraints to control which datacenters contain which data, how far data is from its users (to control read latency), how far replicas are from each other (to control write latency), and how many replicas are maintained (to control durability, availability, and read performance). Data can also be dynamically and transparently moved between datacenters by the system to balance resource usage across datacenters. Second, Spanner has two features that are difficult to implement in a distributed database: it

“Serverless” Design Principles and Advantages

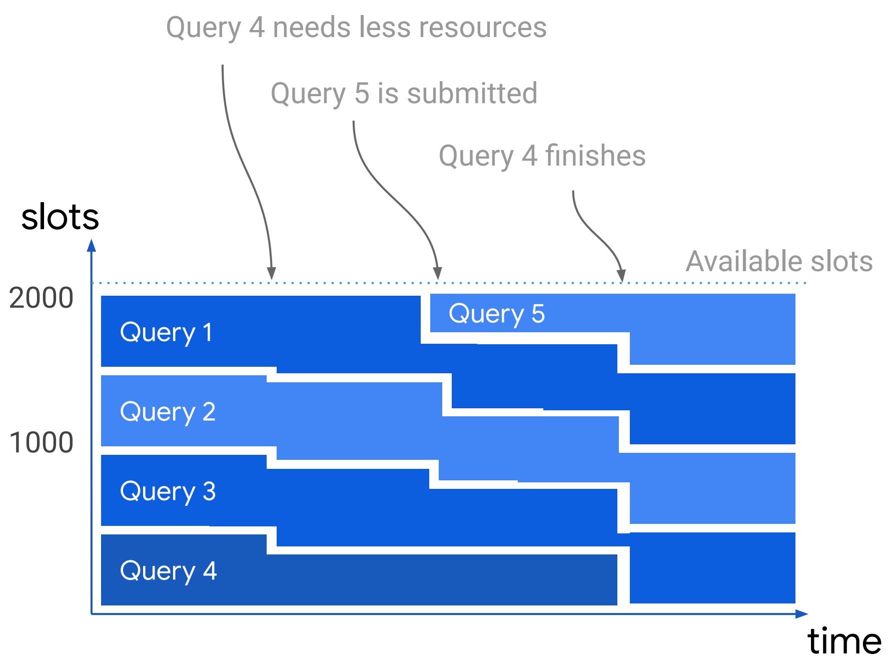
- **Disaggregation of compute, storage, memory**
    - On-demand scaling of each resource
    - On-demand sharing of resources
    - Adapts well to ***multi-tenant*** usage at lower cost
- **Fault tolerance and restartability**
    - At scale assume everything is unreliable/slow
    - Query subtasks are deterministic and repeatable
    - Multiple copies of same task dispatched to avoid stragglers

In Memory Shuffle

- **BigQuery implements a disaggregated memory-based shuffle**
  - RAM/disk managed separately from compute tier
  - Reduced shuffle latency by <u>order-of-magnitude</u>
  - Enables <u>order-of-magnitude</u> larger shuffles
  - Reduced <u>resource cost by 20%</u>
  - Avoid resource fragmentation, stranding, poor isolation

- **Persistence in shuffle layer**
  - Checkpoint query execution state
  - Allows flexibility in scheduling + execution (preemption of workers)

Dynamic Scheduling



Dynamic Query Execution

- **Dynamic (Re)Partitioning**: load balance and adjust parallelism while adapting to any query or data shape and size

- **Dynamic join processing:**
    - Example – start with shuffle join, but cancel and switch to broadcast join if data sizes warrant it

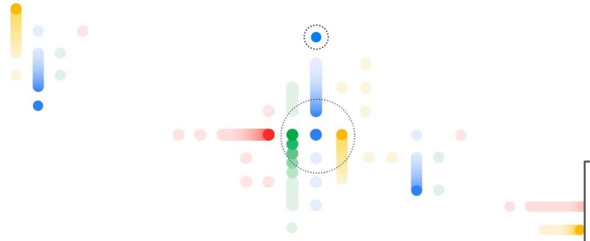
made available on the SIGMOD Web site.

Read / Write API + Streaming

- **Read API**
  - Read data in parallel directly from BQ storage
  - For consumption by Spark, Presto, Tensorflow, etc, etc...
- **Write API**
  - Industry-leading stream ingest support at scale
  - Exactly once semantics
  - Stream-level and cross-stream transactions

DATA ANALYTICS

Now in preview, BigQuery search features provide a simple way to pinpoint unique elements in data of any size



Srinidhi Raghavan
Software Engineer, Google Cloud

Christopher Crosbie
Product Manager, Google Cloud

Today, we are excited to announce the public preview of search indexes and related SQL SEARCH functions in BigQuery. This is a new capability in BigQuery that allows you to use standard BigQuery SQL to easily find unique data elements buried in unstructured text and semi-structured JSON, without having to know the table schemas in advance. By making row lookups in BigQuery efficient, you now have a powerful columnar store and text search in a single data platform. This allows for

DEVOPS & SRE

Introducing Cloud Logging - Log Analytics, powered by BigQuery

Charles Baer
Product Manager, Google Cloud

September 27, 2022

Logging is a critical part of the software development lifecycle allowing developers to debug their apps, DevOps/SRE teams to troubleshoot issues, and security admins to analyze access. Cloud Logging provides a powerful pipeline to reliably ingest logs at scale and quickly find your logs. Today, we're pleased to announce Log Analytics, a new set of features in Cloud Logging available in Preview, powered by BigQuery that allows you to gain even more insights and value from your logs.

Introducing Log Analytics

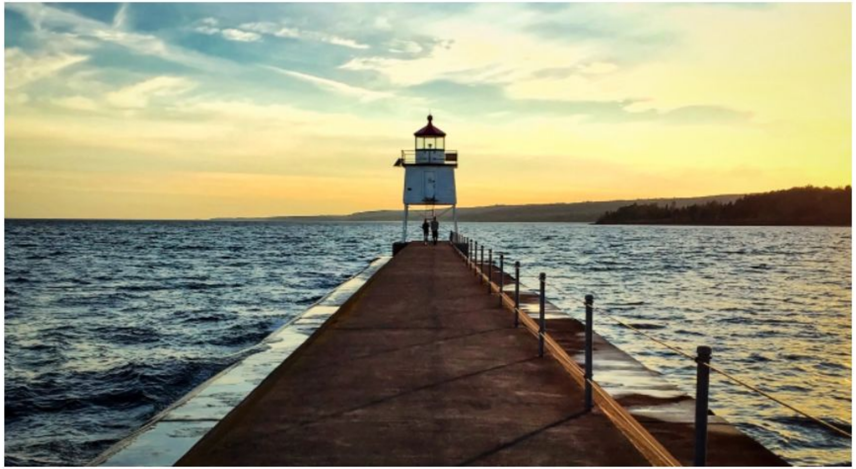
Log Analytics brings entirely new capabilities to search, aggregate, or transform logs at query time directly into Cloud Logging with a new user experience that's optimized for analyzing logs data through the power of BigQuery. BigQuery is a cost-effective, serverless, multi cloud data warehouse to power your data-driven innovation.

With Log Analytics, you can now harness SQL (see figure 1) and the capabilities of

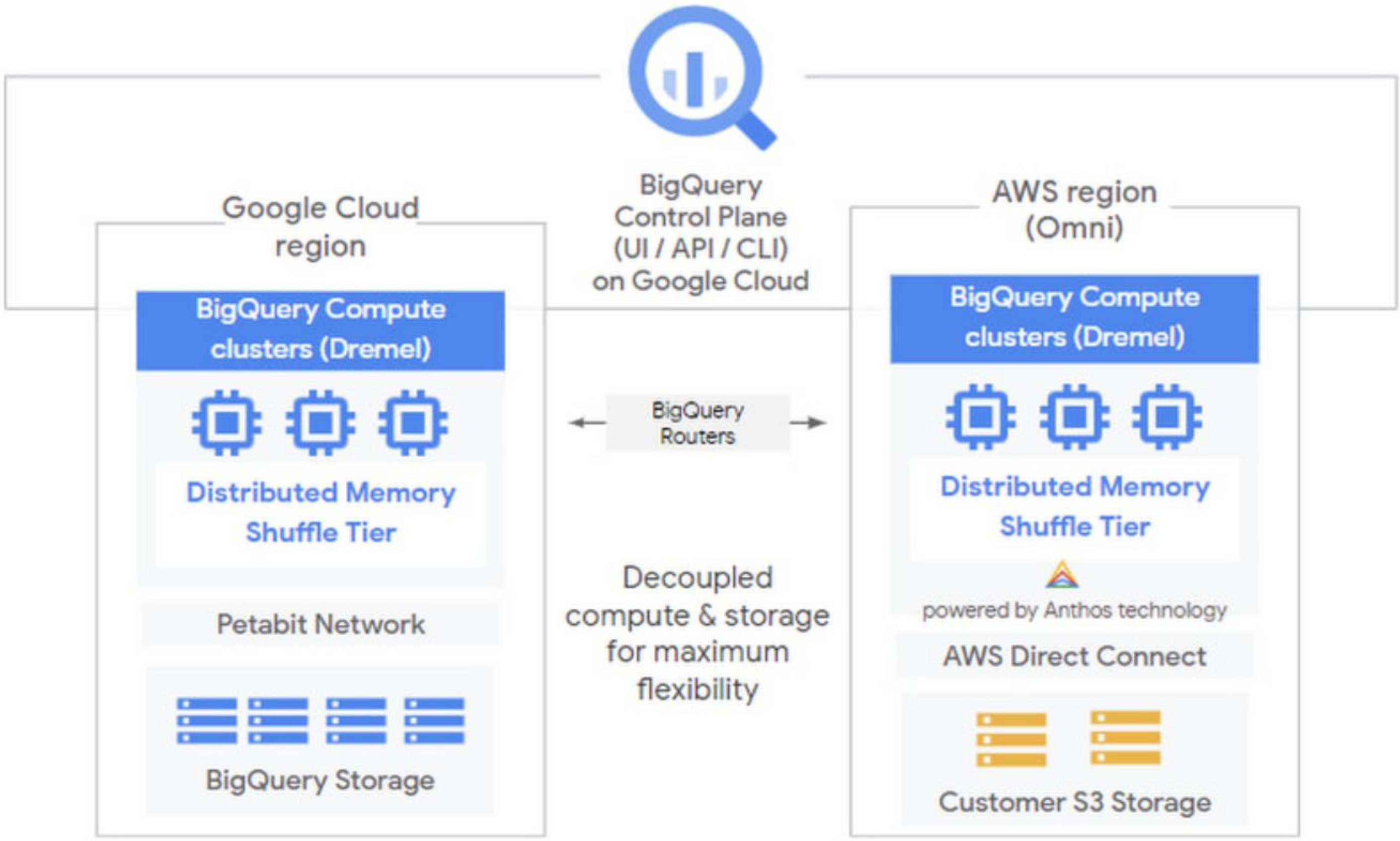
Google Cloud launches BigLake, a new cross-platform data storage engine

Frederic Lardinois @fredericl / 10:00 PM PDT • April 5, 2022

Comment



BigQuery Omni



BigQuery ML

Classification

- Logistic regression
- DNN classifier (TensorFlow)
- Boosted trees using XGBoost
- AutoML Tables

Regression

- Linear regression
- DNN regressor (TensorFlow)
- Boosted trees using XGBoost
- AutoML Tables

Model Import/Export

- TensorFlow models for batch and online prediction

Other Models

- k-means clustering
- Time series forecasting
- Recommendation: Matrix factorization

Unstructured Data

```sql
CREATE TABLE dataset1.images
WITH CONNECTION 'service_account1'
OPTIONS (uris=['gs://mybucket/*'])
```

filename	create_time	generation	...
image1.jpg	2021-11-04	2rba7gbp0	
image2.jpg	2021-11-05	gbp02rba7	
image3.jpg	2021-11-06	p02rbgbgb	

```sql
SELECT * FROM
  ML.PREDICT(MODEL cat_detector,
    SELECT _HANDLE FROM dataset1.images
    WHERE ENDSWITH(filename, 'jpg')
    AND create_time > TIMESTAMP('2021-1-1')
  )
```

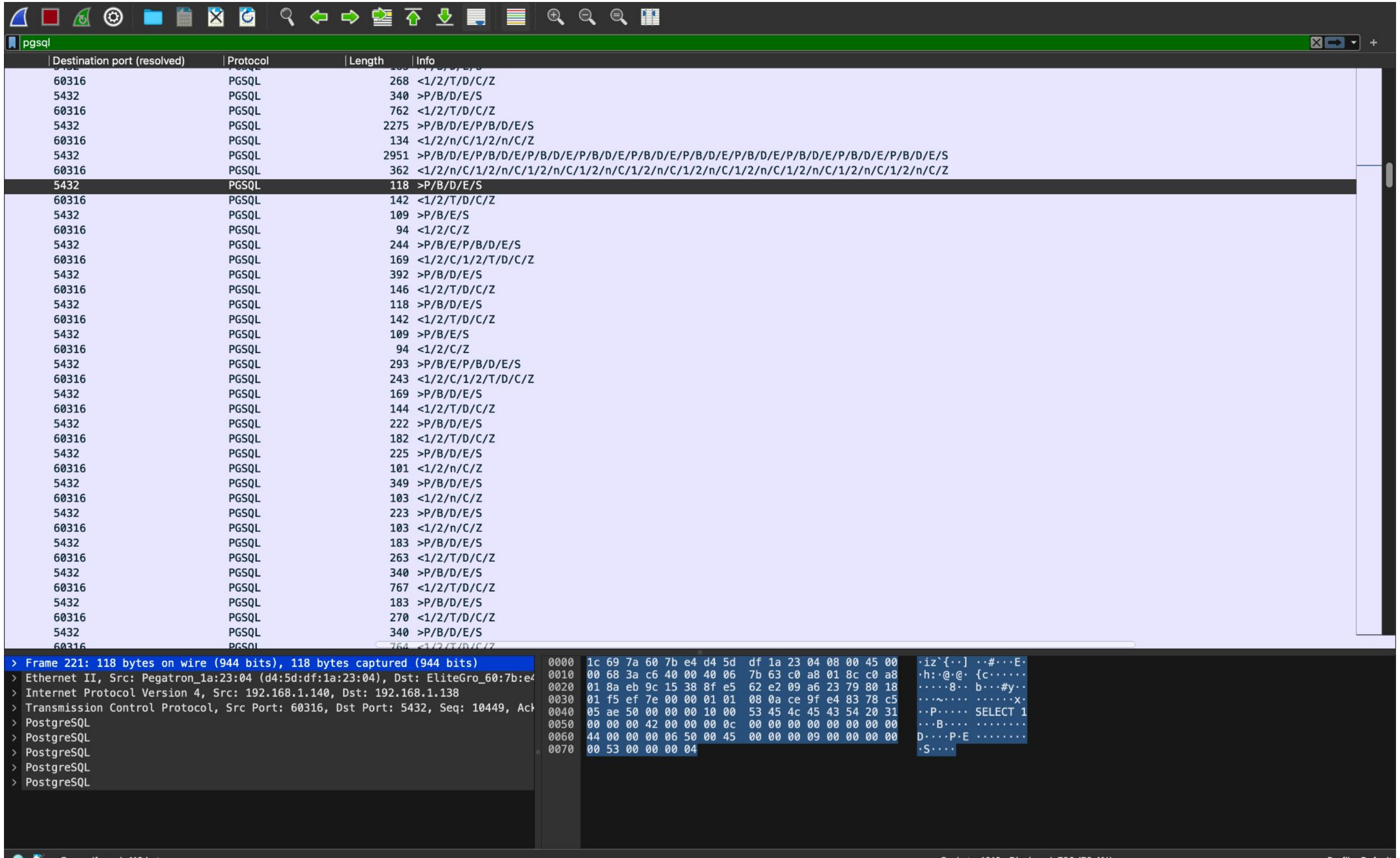
The Sugar-free Chocolate of **Databases**

Matt Butrovich
Carnegie Mellon University
#1 Ranked CMU-DB PhD Student

Eating Smarter

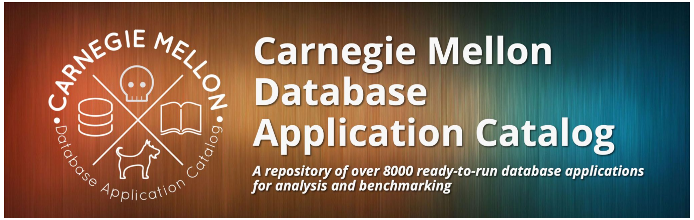
Food labels in the US are wild.

Staring at DBMS Traces

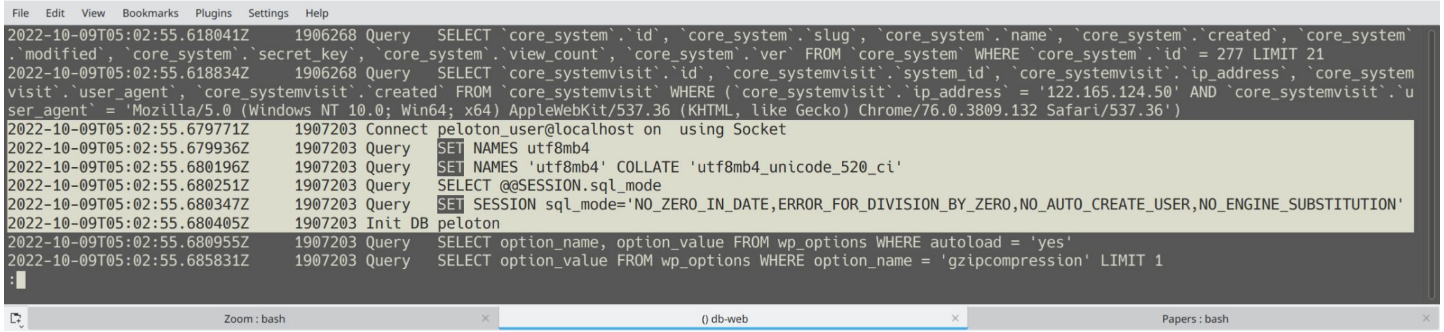


Zero Calorie Queries

- 1,462,909 queries from various workloads…
- CMDBAC data set shows that they are 27% of all queries!



- Look in your SQL logs and you'll see these queries over and over again!



File Edit View Bookmarks Plugins Settings Help

```
2022-10-09T05:02:55.618041Z	   1906268 Query	SELECT `core_system`.`id`, `core_system`.`slug`, `core_system`.`name`, `core_system`.`created`, `core_system`.`modified`, `core_system`.`secret_key`, `core_system`.`view_count`, `core_system`.`ver` FROM `core_system` WHERE `core_system`.`id` = 277 LIMIT 21
2022-10-09T05:02:55.618834Z	   1906268 Query	SELECT `core_systemvisit`.`id`, `core_systemvisit`.`system_id`, `core_systemvisit`.`ip_address`, `core_systemvisit`.`user_agent`, `core_systemvisit`.`created` FROM `core_systemvisit` WHERE (`core_systemvisit`.`ip_address` = '122.165.124.50' AND `core_systemvisit`.`user_agent` = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.132 Safari/537.36')
2022-10-09T05:02:55.679771Z	   1907203 Connect peloton_user@localhost on  using Socket
2022-10-09T05:02:55.679936Z	   1907203 Query	SET NAMES utf8mb4
2022-10-09T05:02:55.680196Z	   1907203 Query	SET NAMES 'utf8mb4' COLLATE 'utf8mb4_unicode_520_ci'
2022-10-09T05:02:55.680251Z	   1907203 Query	SELECT @@SESSION.sql_mode
2022-10-09T05:02:55.680347Z	   1907203 Query	SET SESSION sql_mode='NO_ZERO_IN_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION'
2022-10-09T05:02:55.680405Z	   1907203 Init DB peloton
2022-10-09T05:02:55.680955Z	   1907203 Query	SELECT option_name, option_value FROM wp_options WHERE autoload = 'yes'
2022-10-09T05:02:55.685831Z	   1907203 Query	SELECT option_value FROM wp_options WHERE option_name = 'gzipcompression' LIMIT 1
:
```

Zoom : bash	() db-web	Papers : bash

DBMS Proxies to the Rescue

- PgBouncer, RDS Proxy, ProxySQL
- Features:
    - Connection pooling
    - Query rewriting
    - Sharding
    - Query caching

What I Do

- **Tigger** is a proxy that pushes Application Layer (i.e., L7) DBMS protocol logic into kernel-space via eBPF.
- Perform things like transaction pooling and workload replication without ever going to user-space. **User-bypass**.

I am graduating in early 2024.
I will be expensive to hire.

https://mattbutrovi.ch

Coming out of Codd's shadow – search on unstructured data

Mehul A. Shah
mehul@aryn.ai
www.linkedin.com/in/mehulashah/

The Zeitgeist

- Unstructured data abounds in enterprises
  - growing 3X faster than structured
  - non-consumption: 90% of this data is “dark”

- Data lakes are all the rage
  - lots of attention on structured
  - docs, audio, images, videos, logs, genomes, …
  - don’t know what I have, where it is, and how to synthesize it

- I’ve been under a rock for 5 years
  - new large (transformer) models can … speak English, feed my dogs …
  - 10x / year parameter growth - disrupted overnight
  - open source - download 10GBs from internet

Aryn

The brilliance of Codd ...

1970s

queries in

relational calculus
(first-order logic)

relational
algebra

map data into

relational model

separate app from technology growth
lasted 50 years, 10^10X

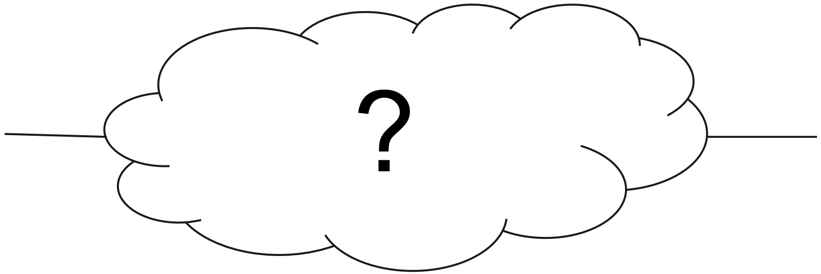
what about unstructured data?

Out of Codd's shadow ...

2020s

queries in

keywords
natural language
image search
recommendations



public data
modeled in

alphabet soup of
LLM - transformers

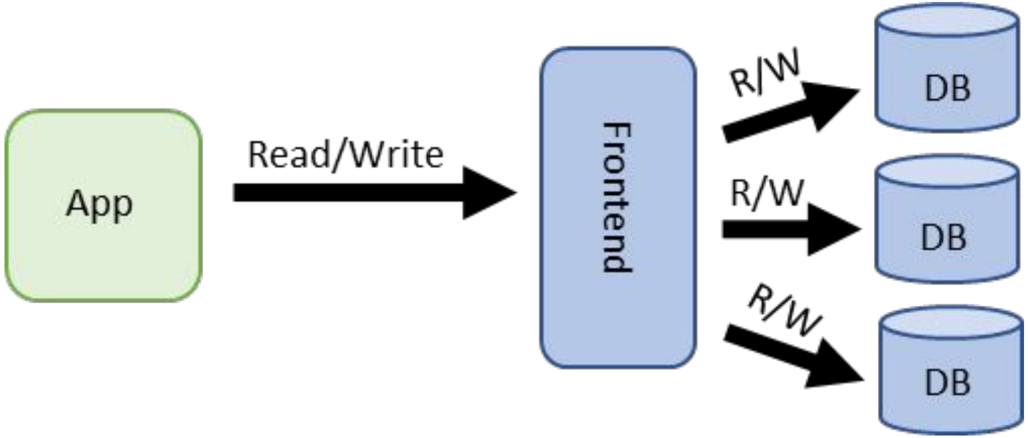
transformer learns the data and the queries
can we understand them? is there an algebra?
what happens for the next 50 years?

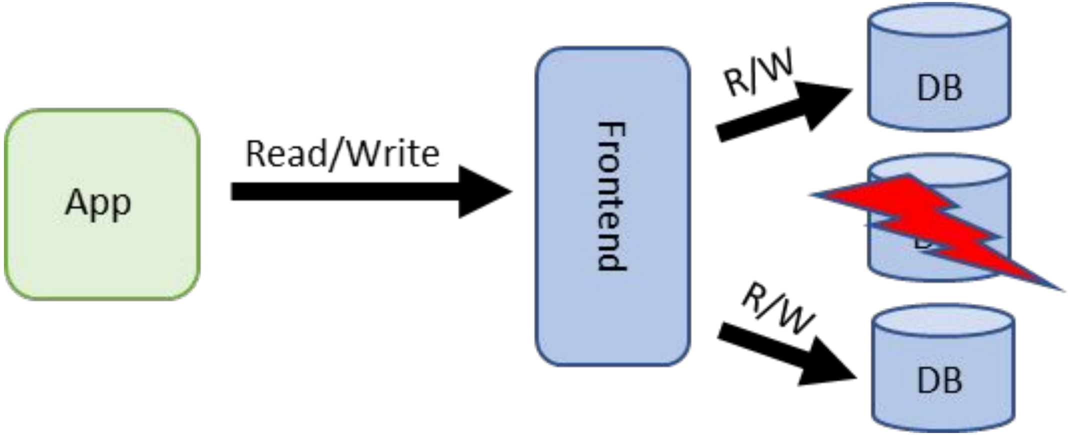
Stop Losing Sleep Over Losing Data

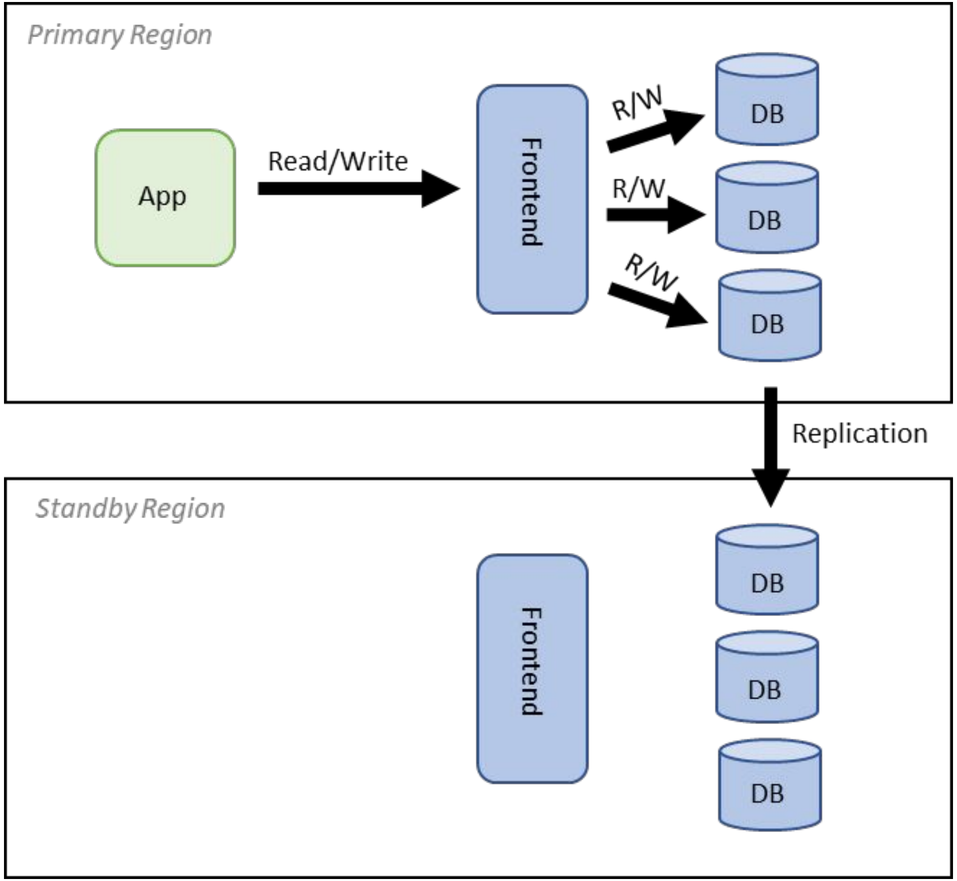
Doug Terry
Amazon Web Services

Customers increasingly fret about

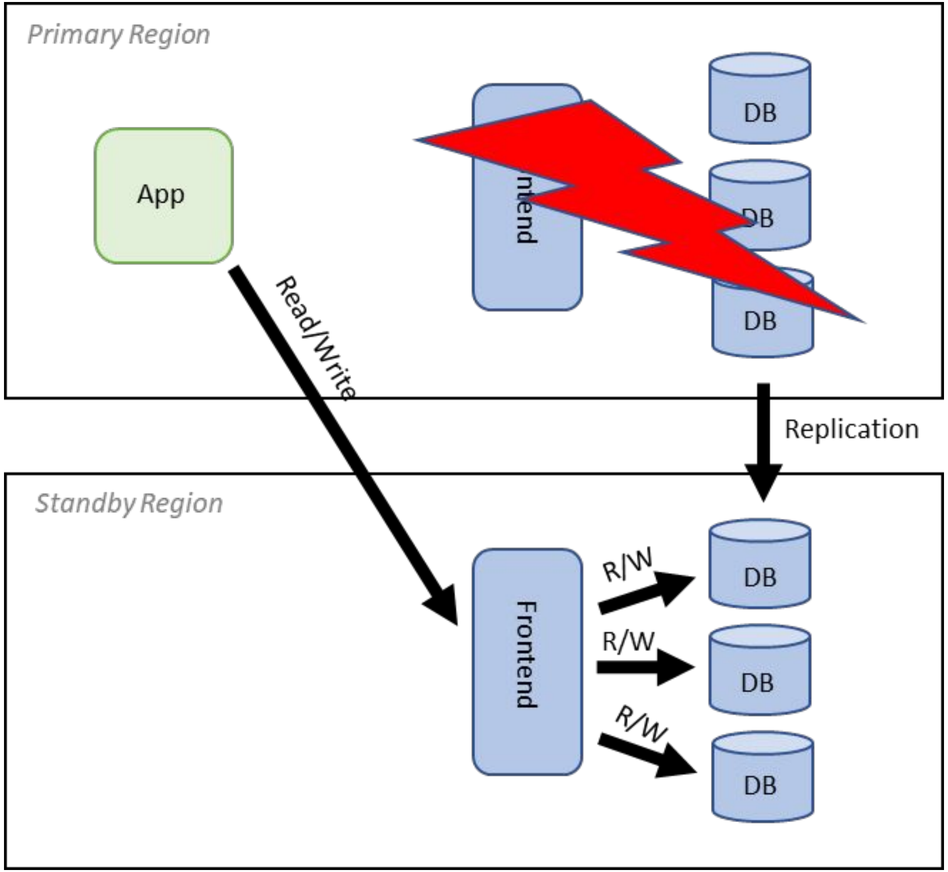
1. Data loss
2. Region failures


App
Read/Write
Frontend
R/W
R/W
R/W
DB
DB
DB


App
Read/Write
Frontend
R/W
DB
R/W
DB

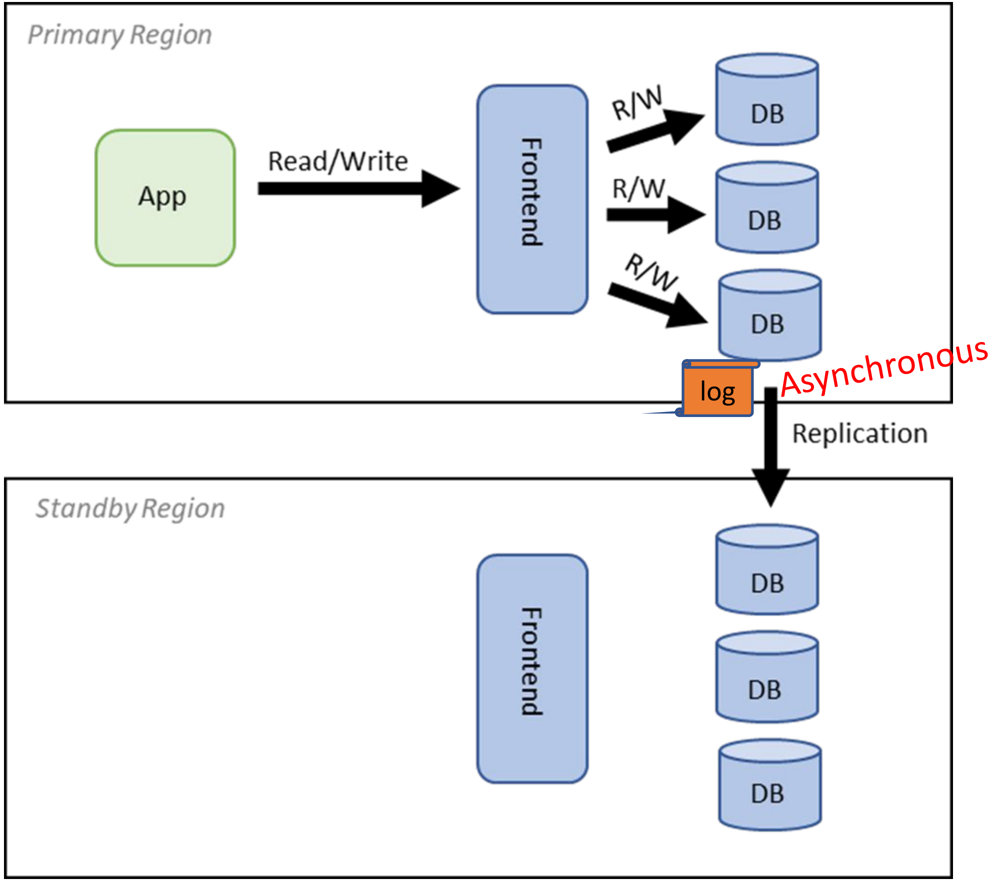


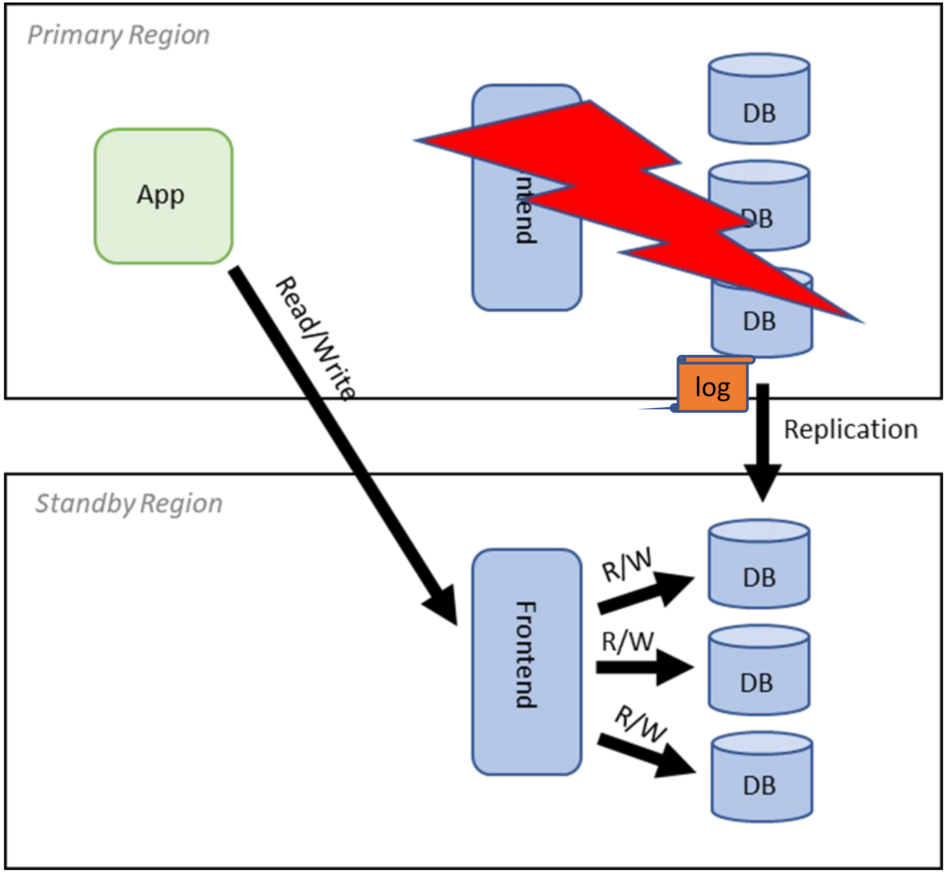
Primary Region
App
Read/Write
Frontend
R/W
R/W
R/W
DB
DB
DB
Replication
Standby Region
Frontend
DB
DB
DB



Primary Region
App
ntend
DB
DB
DB
Read/Write
Replication
Standby Region
Frontend
R/W
R/W
R/W
DB
DB
DB

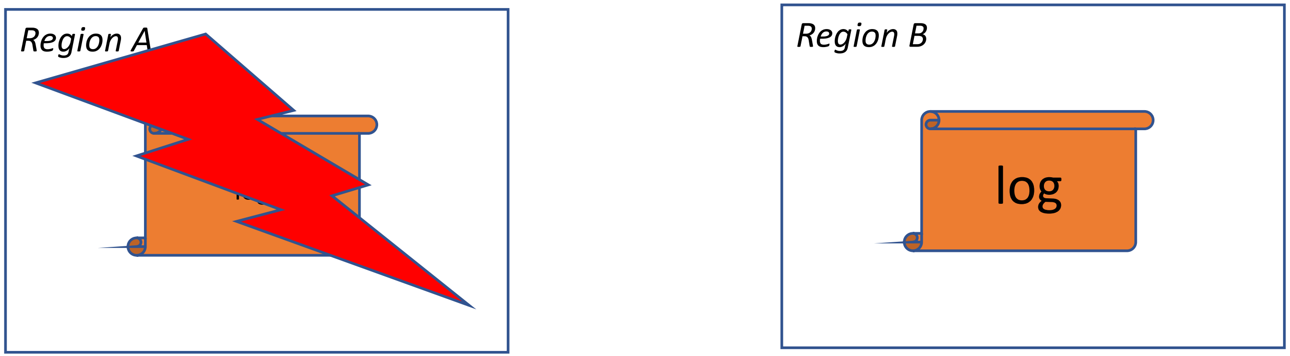
RPO has gotta go
Hell No RPO!
RPO


Primary Region
App
Read/Write
Frontend
R/W
R/W
R/W
DB
DB
DB
log
Asynchronous
Replication
Standby Region
Frontend
DB
DB
DB

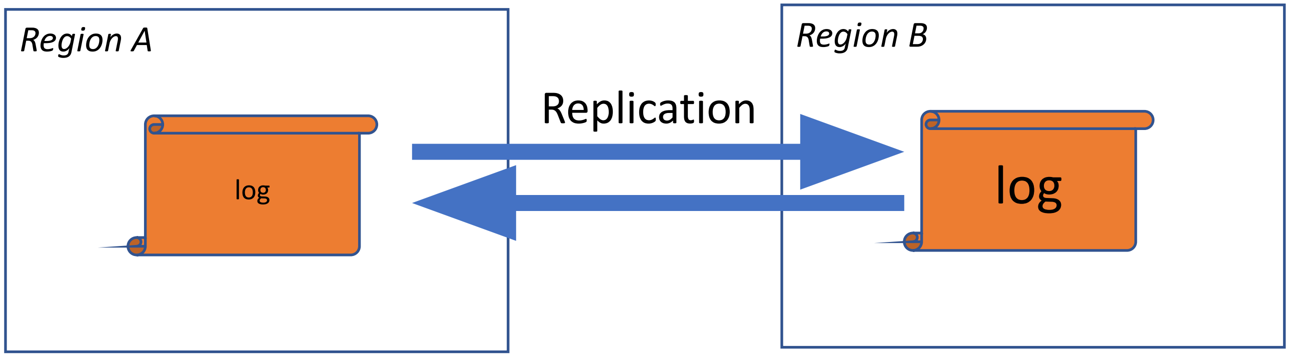

Primary Region
App
Read/Write
ntend
DB
DB
DB
log
Replication
Standby Region
Frontend
R/W
R/W
R/W
DB
DB
DB

What can we do about RPO?

Option 1: Accept it



Option 1: Accept it

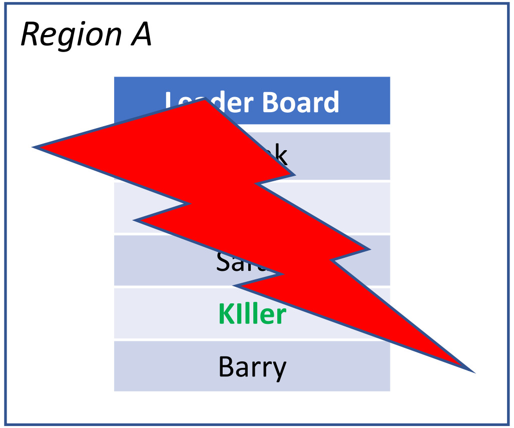


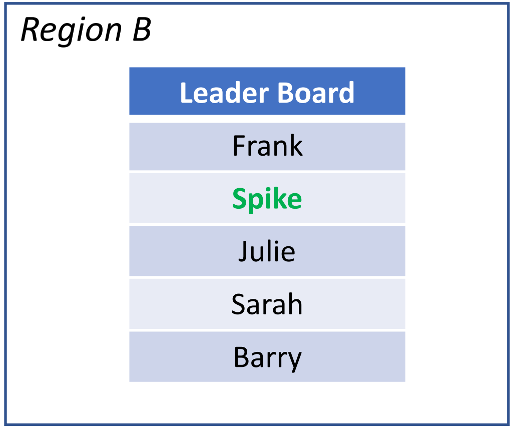
Option 2: Reconcile it

Region A

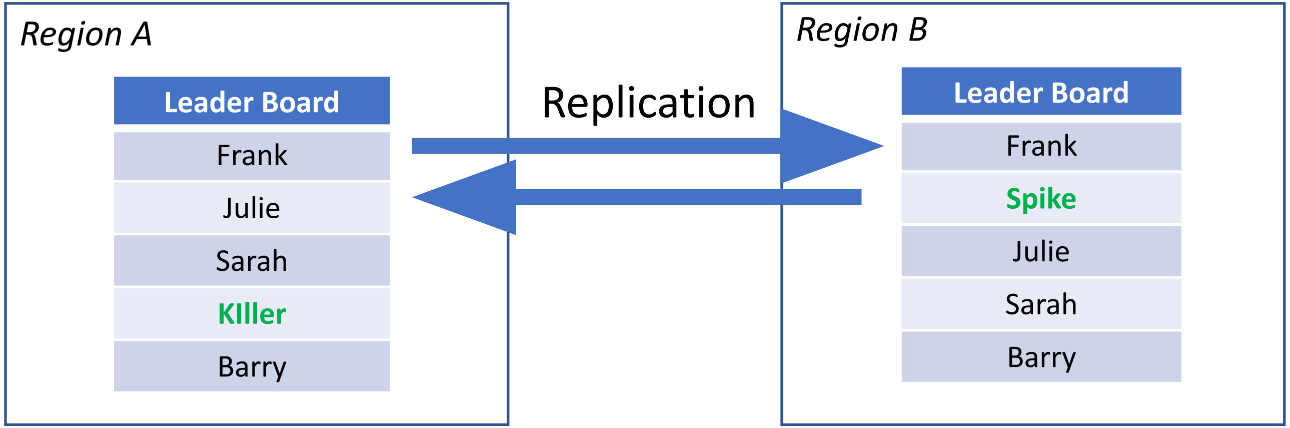
Leader Board
Frank
Julie
Sarah
KIller
Barry

Option 2: Reconcile it

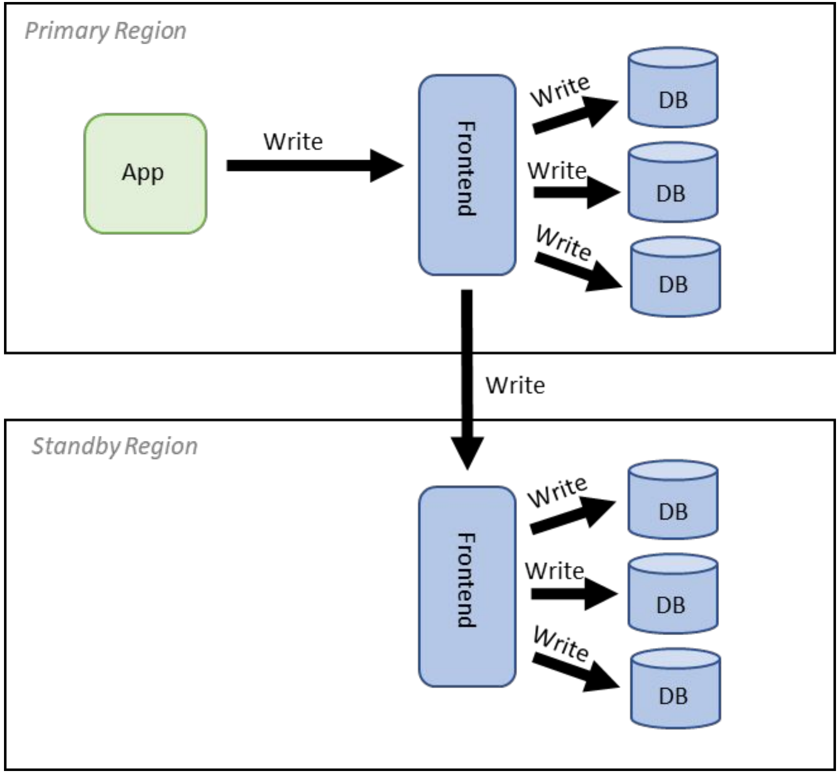




Option 2: Reconcile it



Option 3: Prevent it



What to do about RPO?

Option 1: Accept it

Option 2: Reconcile it

Option 3: Prevent it


PLEASE
BE
CAREFUL
ONLY
YOU CAN
PREVENT RPO
SMOKEY
SMOKEY

Thank you!