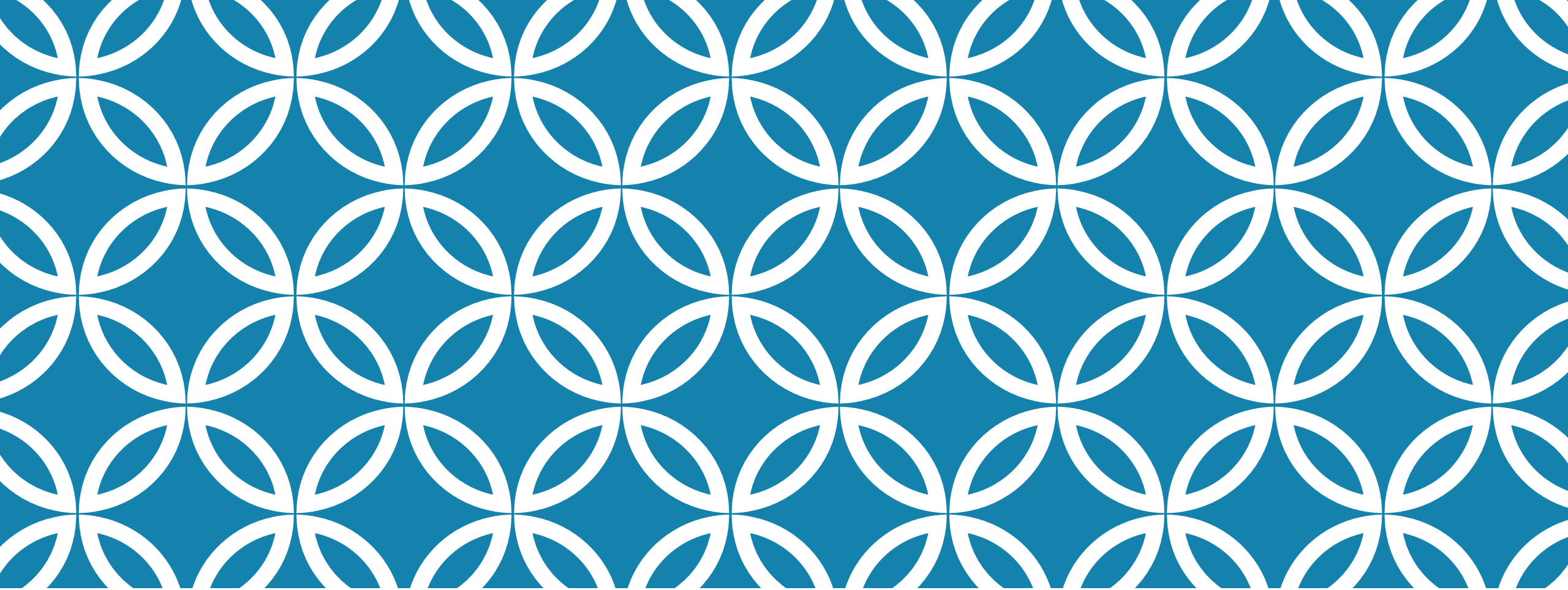


CLOUD MICROSERVICE ARCHITECTURES

Professor Ken Birman

CS4414/5416 Lecture 19

KEY IDEAS FOR TODAY

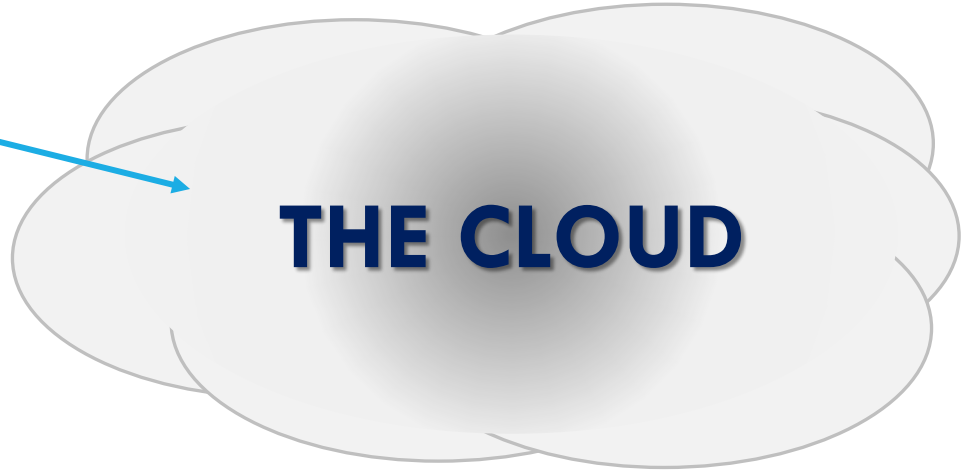
- We discussed several really large cloud infrastructure services
- You build new applications by introducing much smaller bits of code (lambdas) that leverage those existing tools. This leads to a so-called microservice model, sometimes written μ-service
- Exciting new trends? Agentic AI uses this same infrastructure

Early first-tier services were web servers. But we broke them into components. The microservice approach emerged because monolithic web servers were becoming too large and sluggish.

Today, the first tiers are portals to web sites/services accessed via http(s)/GRPC, backed by **microservices**

Our focus in today's lecture is on the main elements of the cloud computing microservices ecosystem

WHY DO WE EVEN CALL IT THE "CLOUD"?

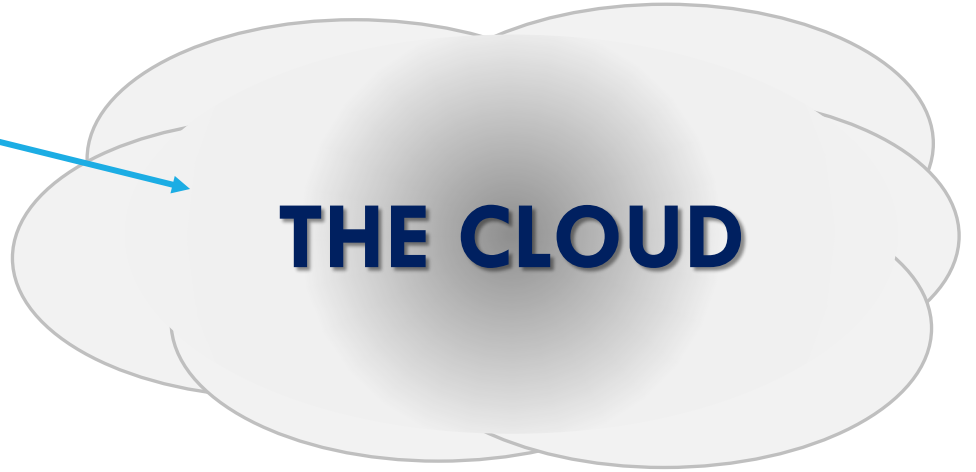


Computing happens somewhere else, not on your PC or mobile device

A big goal is to prepare students for careers at the companies building and running the cloud (Google, Microsoft, Amazon, Facebook…).

Cloud computing is a *technology* course: we ask how things really work, not a *concepts* course that might be more centered on capabilities and use cases (in fact we will see lots of those, but they aren't our main topic)

THE CLOUD UNDERPINS MODERN COMPUTING



Physical: The cloud is a global deployment of massive data centers connected by ultra-fast networking, designed for scalability and robustness.

Logical: A collection of tools and platforms that scale amazingly well. The platforms matter most; as a developer, they allow you to extend/customize them to create your application as a "personality" over their capabilities.

Conceptual: A set of scalable ideas, concepts and design strategies.

WHO INVENTED THE CLOUD?

Jeff Dean

Sanjay Ghemawat

Google's Jeff Dean and Sanjay Ghemawat get my "vote".

- Jeff Dean was a University of Washington PhD student. We know him well and he often visits Cornell. Now he is the head of Google Brain.
- Sanjay Ghemawat was a Cornell ugrad, then MIT PhD. He is a Senior Fellow in Google's systems infrastructure area.

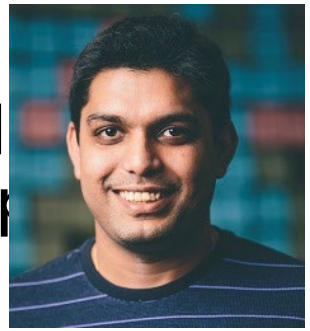
Both Jeff and Sanjay are famous for simple and robust ways to scale things up (and writing about them). This was the key to the modern cloud.

I WAS THERE TOO…

- I didn't invent the cloud, but many of my students had huge roles.
- Personally, I created the "self-healing" software that ran the trading floors of the New York Stock Exchange and the Swiss Exchange for 10+ years. The US military uses this technology too.
- Designed the French portion of the European Air Traffic control system, control and created the core software. They've used it since 1996.
- Oracle and Microsoft both use a technology I invented to track the status of their clusters and data centers.
- Recently, I helped create the New England smart grid (for ISONE and NYPA), and helped the Air Force figure out how to leverage the cloud.

SOME OF MY PAST STUDENTS BECAME CLOUD COMPUTING SUPERSTARS

Werner Vogels was in my group until 2005.
He has been CTO of Amazon since 2006.



Ranveer Chandra is Chief Scientist for Azure Global, head
Networking Research and responsible for their FarmBeats p

Yee-Jiun Song: VP Engineering, Facebook

Qi Huang: Facebook video/photo delivery

Dalia Malkhi: CTO for Diem, Facebook's digital cur

EVERYTHING IS BIG IN THE CLOUD

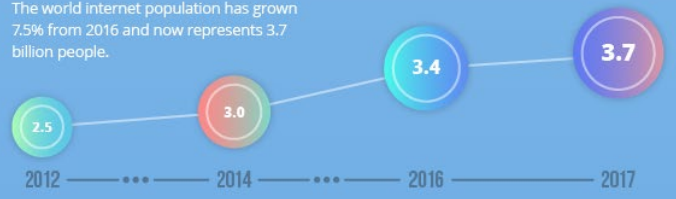
DOMO

DATA NEVER SLEEPS 5.0

How much data is generated *every minute?*

90% of all data today was created in the last two years—that's 2.5 quintillion bytes of data per day. In our 5th edition of Data Never Sleeps, we bring you the latest stats on just how much data is being created in the digital sphere—and the numbers are staggering.

The world internet population has grown 7.5% from 2016 and now represents 3.7 billion people.



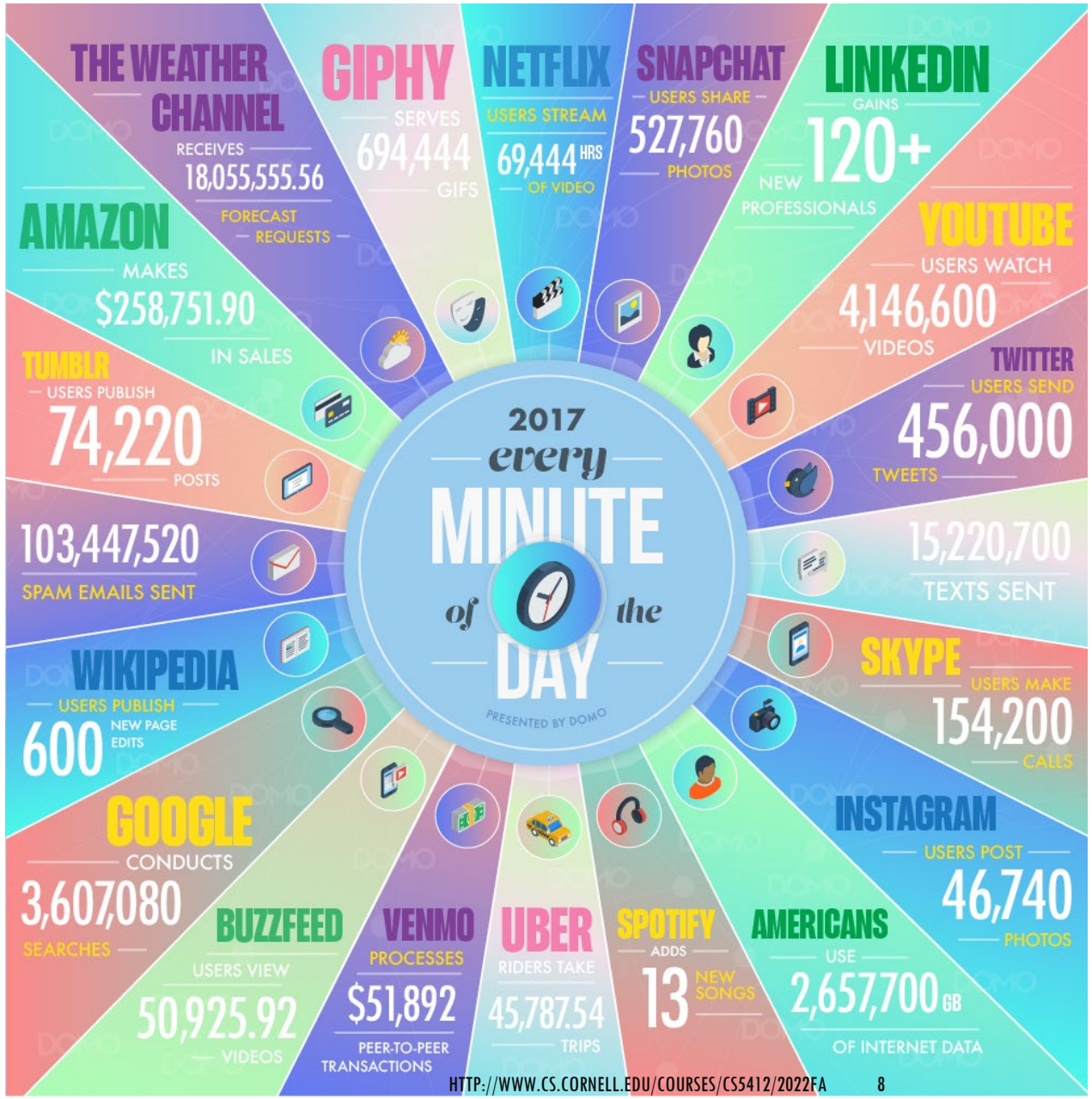
GLOBAL INTERNET POPULATION GROWTH 2012–2017
(IN BILLIONS)

With each click, swipe, share, and like, businesses are using data to make decisions about the future. Domo gives everyone in your business real-time access to data from virtually any data source in a single platform for smarter decision-making at any moment.

Learn more at domo.com

SOURCES: EXPANDEDRAMBLINGS.COM, WEARESOCIAL.COM, WIKIPEDIA, FORBES, ADWEEK.COM, FORTUNE.COM, BLOOMBERG.COM, ONEREACH.COM, IBM, BUZZFEED, INTERNET LIVE STATS, INTERNET WORLD STATS, BBC

DOMO

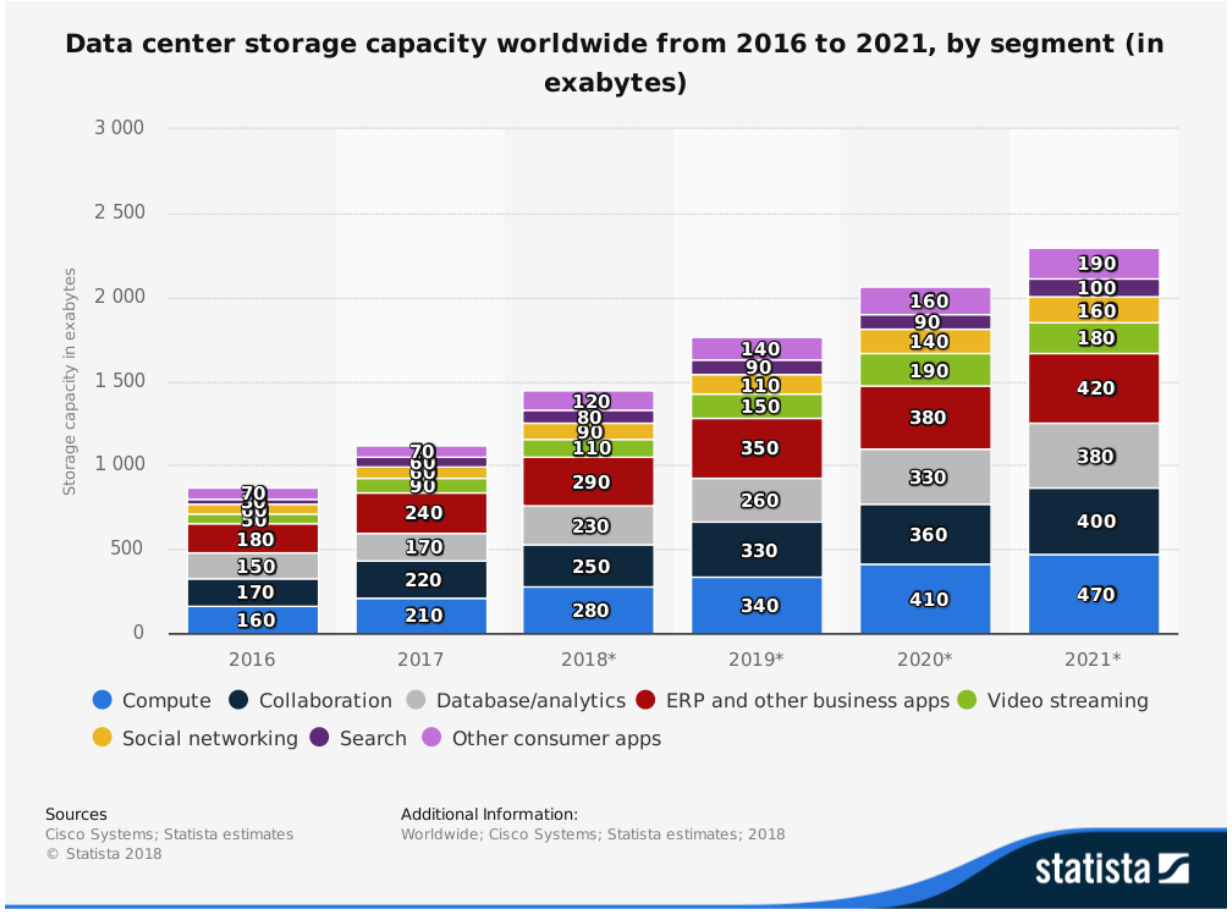


DATA IN THE CLOUD

1 Exabyte of data is 1,073,741,824 GB. (Your hard disk probably holds 64GB, but is way too slow by data-center standards)

The Internet has about 2B websites, and of these, 644M have "active content"

… and all of this is "pre Internet of Things"



HOW DID TODAY'S CLOUD EVOLVE?

Prior to ~2005, we had "data centers designed for high availability".

Amazon had especially large ones, to serve its web requests

- This is all before the AWS cloud model
- The real goal was just to support online shopping

Their system wasn't very reliable, and the core problem was scaling

- Like a theoretical complexity growth issue.
- Amazon's computers were overloaded and often crashed

WASN'T GOOGLE FIRST?

Google was still building their first scalable infrastructure in this period.

Because Amazon ran into scaling issues first, Google (a bit later) managed to avoid them.

- In some sense, Amazon dealt with these issues "in real time".
- Google had a chance to build a second system by learning from Amazon's mistakes and approaches.

YAHOO EXPERIMENT

A sprint to render your web page!

In the 2005 time period everyone was talking about an experiment done at Yahoo. It was an "alpha/beta" experiment about ad-click-through

- Customers who saw web page rendering faster than 100ms clicked ads.
- For every 100ms delay, *click-through rates noticeably dropped*.

Speed at scale determines revenue, and revenue shapes technology: an arms race to speed up the cloud.

MORE YAHOO FINDINGS

A sprint to render your web page!

Rending the ads first didn't help – in fact it hurt.

Customers wanted to see the "real content" first.

Rendering the ads after the content hurt too.

To get the best click-through rates, render your pages (ads included) fast!

EVERYONE HEARD THIS MESSAGE

At Amazon, Jeff Bezos spread the word internally.

He wanted Amazon to win this sprint.

The whole company was told to focus on ensuring that every Amazon product page would render with minimal delay. Unfortunately… as more and more customers turned up… Amazon's web pages ***slowed down***. This is a "crisis of the commons" situation.

THE CRISIS OF THE COMMONS

At the center of the village is a lovely grassy commons. Everyone uses it.

One day a farmer has an awesome idea. He lets his goats graze on the commons. This saves the time of herding them to his fields. This gains him hours that he uses to improve his goat cheese factory.

He earns extra money with his award-winning cheeses.

THE CRISIS OF THE COMMONS

… his neighbors love the idea! All of them decide to use the commons.

In no time all the grass is gone and the commons is reduced to dust.

For Amazon, success was like that. The first shoppers loved the site, but then "everyone" wanted to use it, and it overloaded and collapsed.

WHERE ARE THE COMMONS, IN A CLOUD?

In the cloud we need to think about all the internal databases and services "shared" by lots and lots of μ-service instances.

If we take the advice to "make everything as fast as possible", all those millions of first-tier μ-services will be greedy.

But what works best for one instance, all by itself, might overload the shared services when the same code runs side by side with huge numbers of other instances ("when we run at scale")

THE CLOUD AND THE "THUNDERING HERD"

In fact this is a very common pattern.

Something becomes successful at small scale, so everyone wants to try it.

But now the same code patterns that worked at small scale might break. The key to scalability in a cloud is *to use the cloud platform in a smart way.*

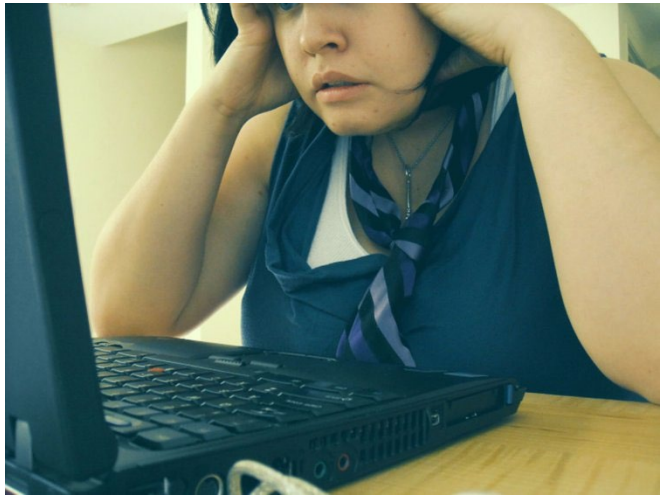
STARTING AROUND 2006, AMAZON LED IN REINVENTING DATA CENTER COMPUTING

Amazon reorganized their whole approach:

- Requests arrived at a "first tier" of very lightweight servers.
- These dispatched work requests on a message bus or queue.
- The requests were selected by "micro-services" running in elastic pools.
- One web request might involve tens or hundreds of μ-services!

They also began to guess at your next action and precompute what they would probably need to answer your next query or link click.

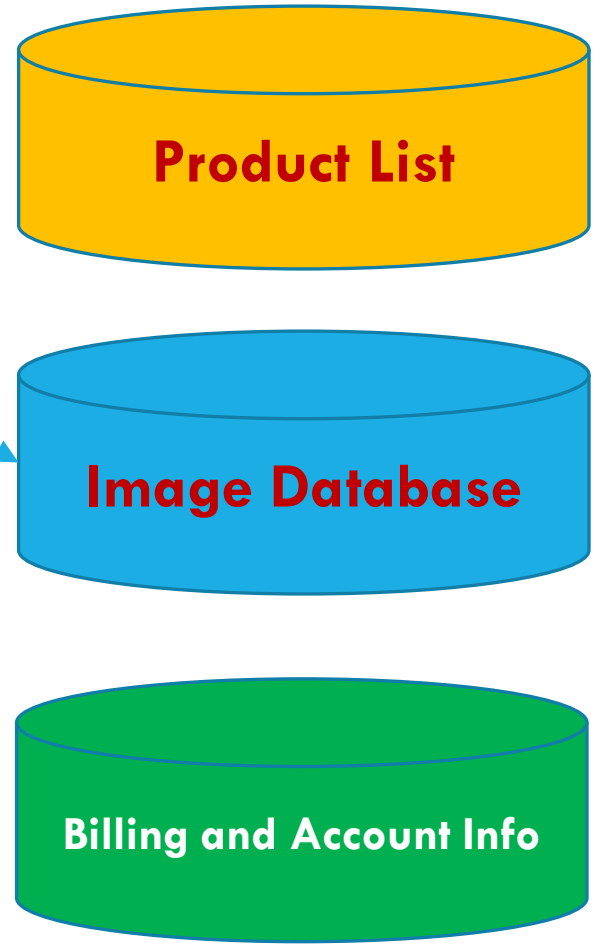
OLD APPROACH (2005)



Computers were mostly desktops

Internet routing was pretty static, except for load balancing

Web Server
built the page… in Seattle



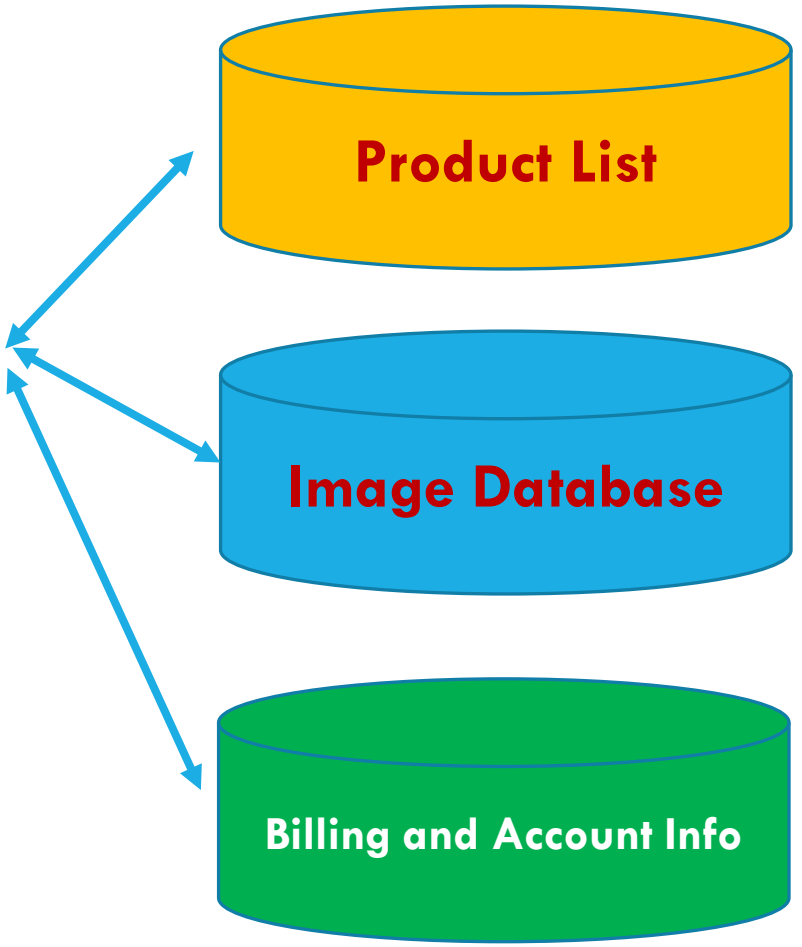
Databases held the real product inventory

NEW APPROACH (2008)

Computers became lightweight, yet faster

Routed to nearest datacenter, one of many

Web Server built the page… ten miles from the users



Databases held the real product inventory

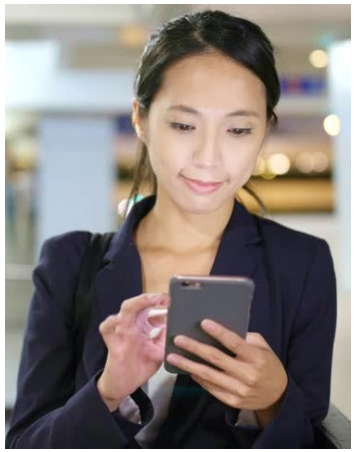
NEW APPROACH (2008)

Computers became lightweight, yet faster

Routed to nearest datacenter, one of many

Backup routing options

Web Server built the page… ten miles from the users



More and more mobile apps

Product List

Image Database

Billing and Account Info

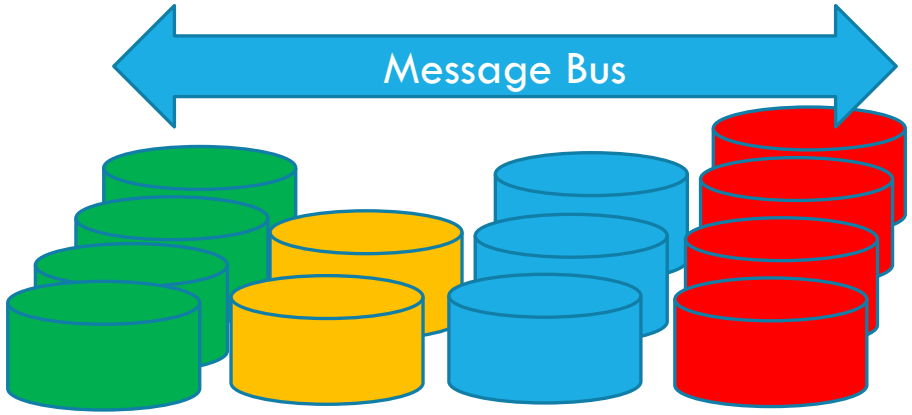
Databases held the real product inventory

NEW APPROACH (2008)

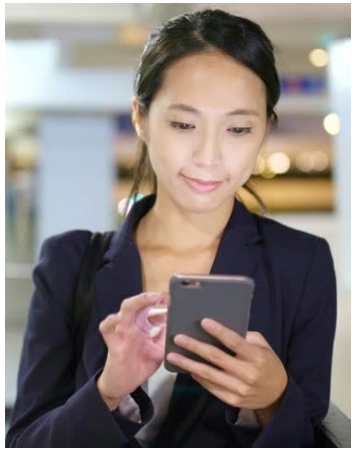
Desktops with snappier response

Routed to nearest datacenter, one of many

Web Server becomes simpler and does less of the real work

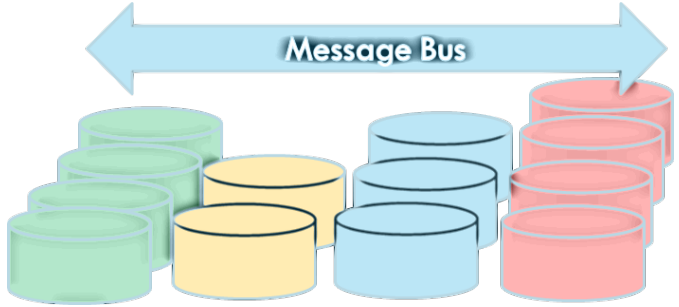


Racks of highly parallel workers do much of the data fetching and processing, ideally ahead of need… The old databases are split into smaller and highly parallel services.



More and more mobile apps

GeoReplication



TIER ONE / TIER TWO

We often talk about the cloud as a "multi-tier" environment.

Tier one: programs that generate the web page you see.

Tier two: services that support tier one. We will see one later (DHT/KVS storage used to create a massive cache)

Back end: Asynchronous/offline services not used directly by tier 1. These do important things but maybe no "in the moment".

TIERING EXAMPLE

A Facebook client program is an app that asks a tier 1 service to create a web page, perhaps for the user's web feed.

Tier 2 services include TAO itself, blob resizing and caching, maps, advertisement/recommendation generating, etc

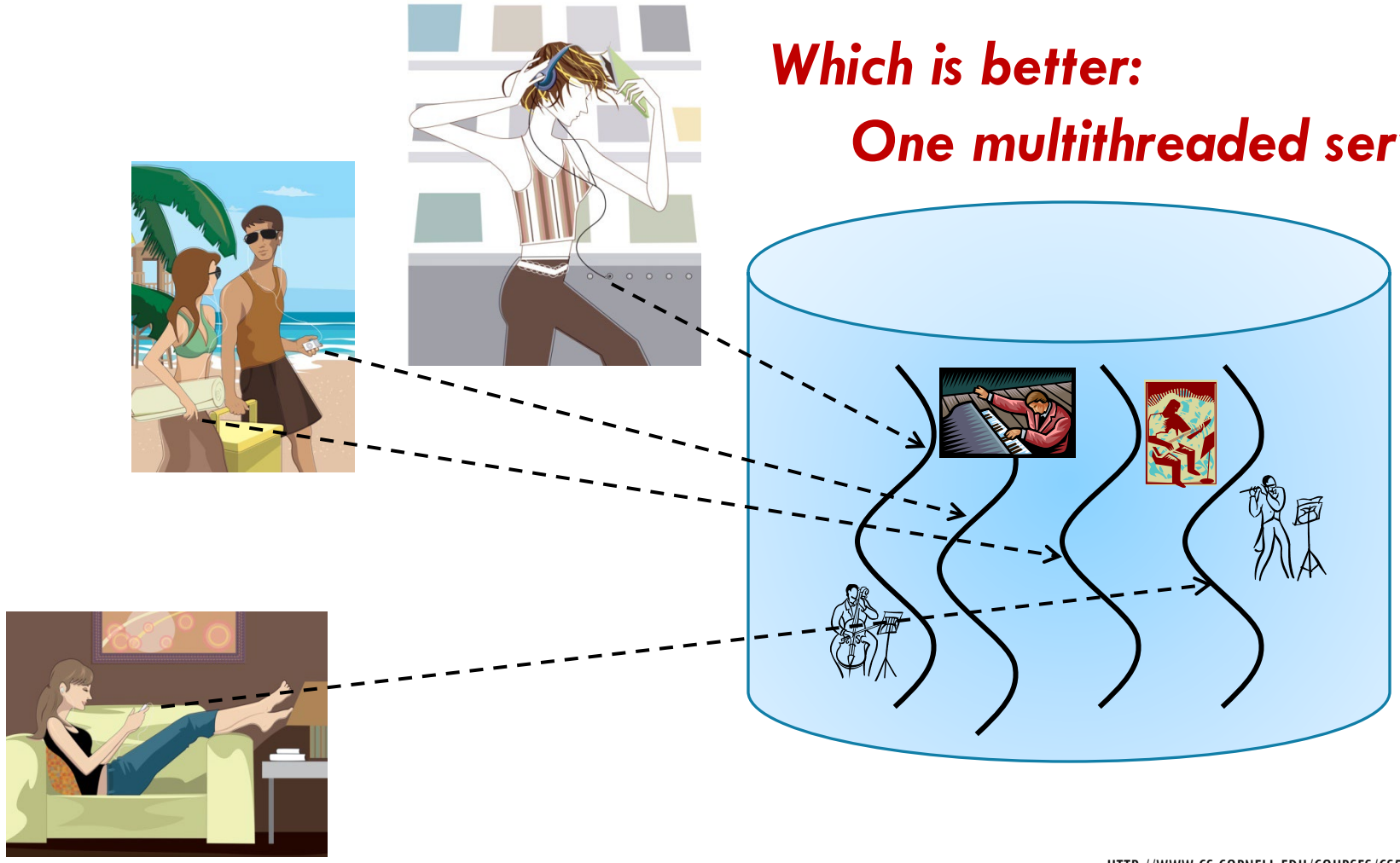
Back end includes Haystack, the TAO database, cloud storage, etc

OLD DEBATE: HOW TO LEVERAGE PARALLELISM?

Not every way of scaling is equally effective. Pick poorly and you might make less money!

To see this, we'll spend a minute on just one example.

EXAMPLE: CLOUD HOSTED MUSIC SERVICE



Which is better:

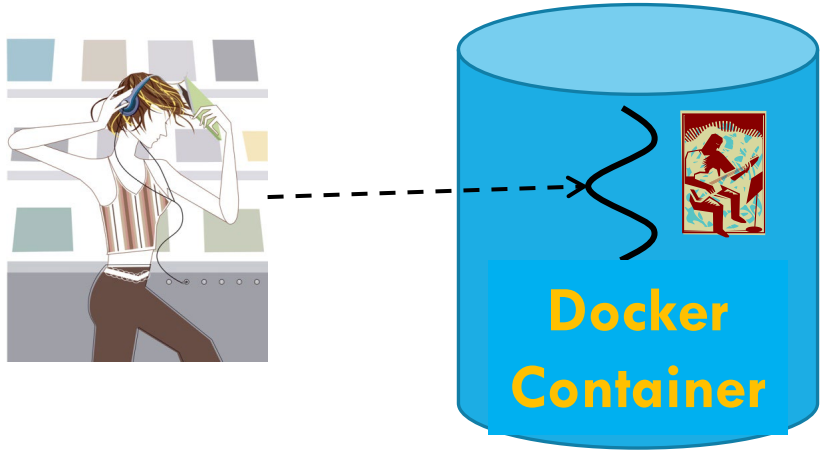
One multithreaded server, per node?

WHICH WOULD YOU PICK?

Basically, we have four options:

1. Keep my server busy by running one multithreaded application on it
2. Keep it busy by running N unthreaded versions of my application as virtual machines, sharing the hardware
3. Keep it busy by running N side by side processes, but don't virtualize
4. Keep it busy by running N side by side processes using containers

THE WINNER IS…



One "μ-player" per user, but many containerized instances per machine

Best is "container virtualization" with one server process dedicated to each distinct user.

A single cloud server might host hundreds of these servers. But they are easy to build: you create one music player, and then tell the cloud to run as many "instances" as required

WHY FAVOR CONTAINER VIRTUALIZATION?

Code is much easier to write. Most people can write a program to play music for a single client – this same insight applies to other programs, too!

Very easy for the cloud itself to manage: containers are cheap to launch and also to halt, when your customer disconnects

The approach also matches well with modern NUMA computer hardware

BUT YOU'VE TAKEN CS4414/5416!

Couldn't you do better?

… and yes, you really could! We've already learned more than most web solution developers ever hear about

But in web settings, if the application is fast enough (that 100ms thing), usually we just declare success and move on!

TODAY'S CLOUD IS OPTIMIZED FOR THIS PATTERN

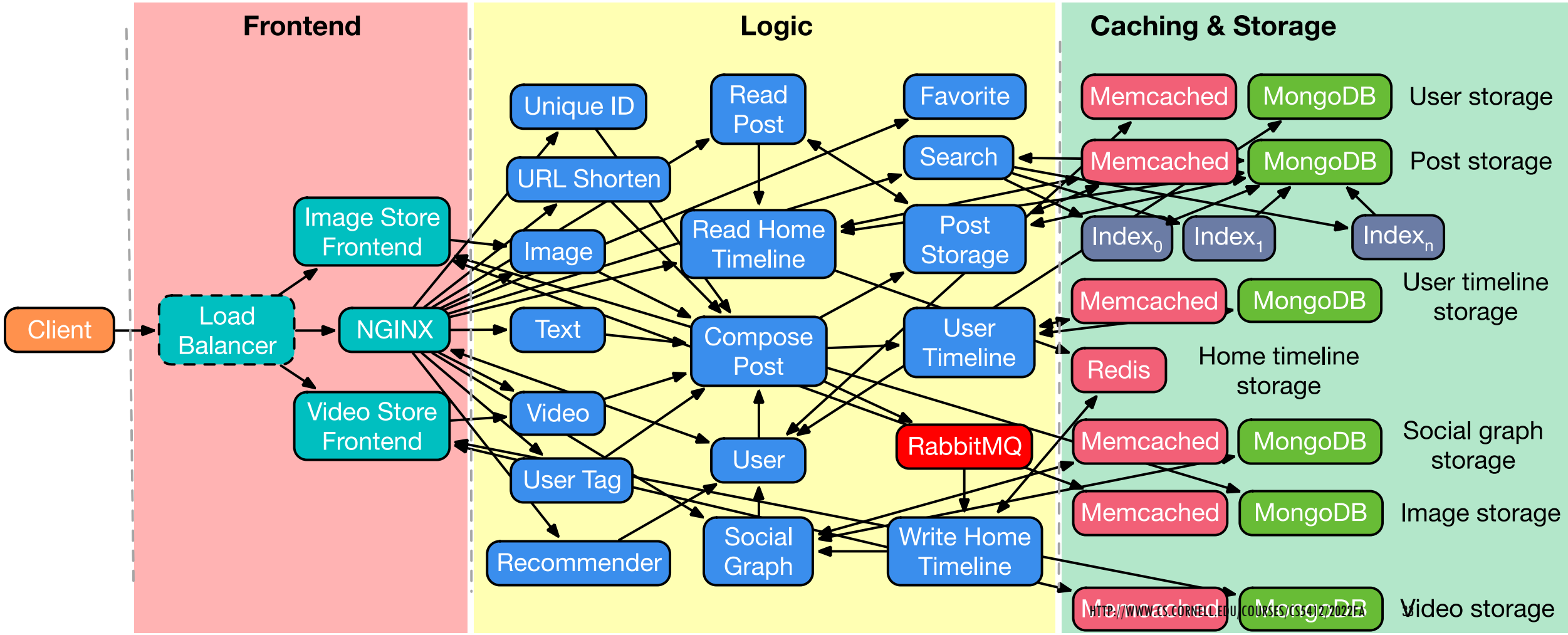
Tier one is specialized for very lightweight servers:

- The services uses very small amounts of computer memory so that each tier one computer can host a huge number of instances
- They don't need a lot of compute power either
- They have limited needs for storage, or network I/O

Tier two μ-Services specialize in various aspects of the content delivered to the end-user. They may run on somewhat "beefier" computers.

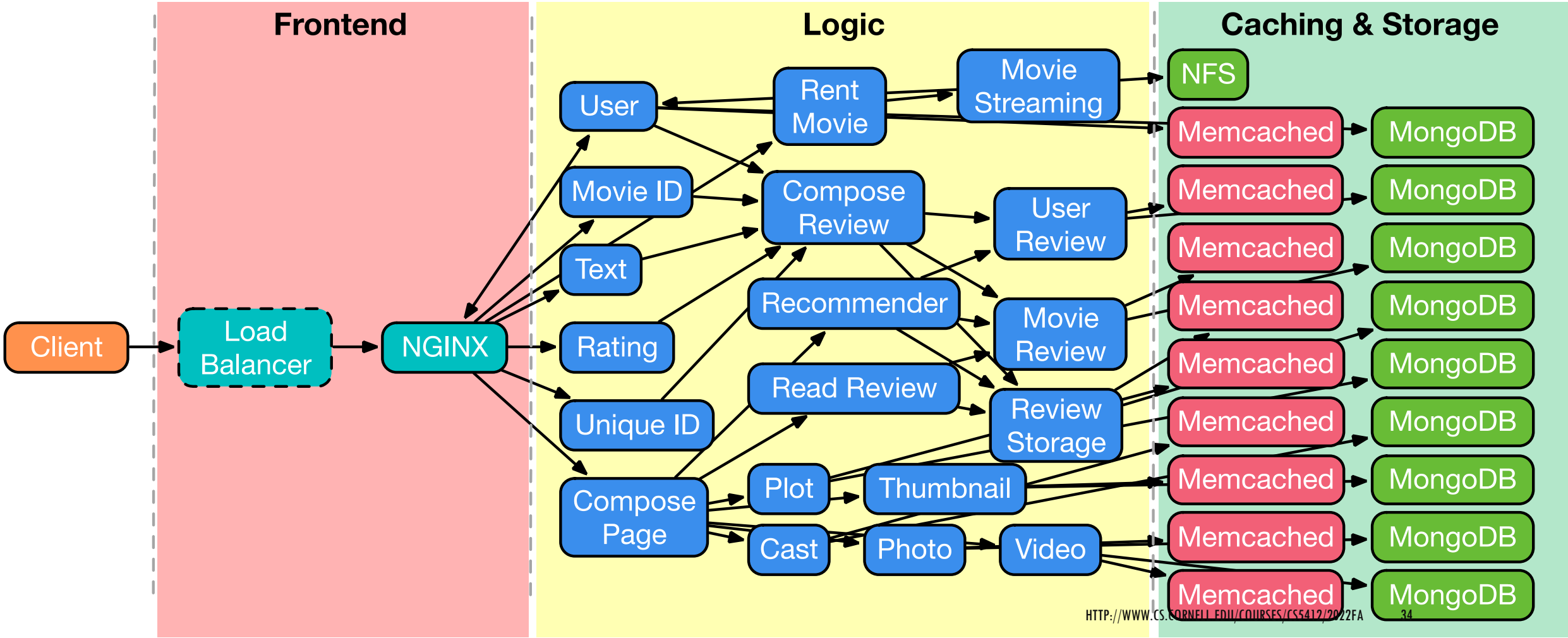
End-to-end Microservices (from Christina Delimitrou)

Social Network



End-to-end Microservices (from Christina Delimitrou)

Media Service



WHAT EXACTLY ARE THE LITTLE BOXES?

They are Linux programs that adhere to standardized methods to talk to the cloud hosting framework and to one-another. We call them microservices (μ-services) if they follow this pattern.

We typically package them as Linux containers for portability, but the code is often as small as a C++ lambda or an object supporting some C++ interface (or Python, Java, C#, whatever)

In these graphics, each box is a label for a **pool** with a variable number of instances of the container, dynamically resized by the cloud

MORE ON THIS ELASTIC POOL CONCEPT

In those graphs, each individual node really is a variable size pool of instances.

Thus, each microservice (μ-service) centers on a program designed so that the data center can shut it down entirely, launch one instance… or many instances, "elastically", to deal with dynamically varying demand.

The idea here is that any instance can handle any request equally well, so there is no need for very careful "routing" of specific requests to specific instances. This lets the data center adapt to changing loads easily!

A MICROSERVICE STARTS AS A PROGRAM

The pieces in our Lego picture might have **originated** as modules in some single program, or been created separately, but by today those modules have been ported into the target setting.

But each target setting can be very different. Clouds are operated by many big vendors like Azure, AWS and Google, but some of their customers operate application-clouds hosted on the larger clouds. *A hierarchy of clouds!*

EACH VENDOR HAS ITS OWN SPECIALITIES!

Each cloud has evolved by emphasizing distinct roles

For example, AWS and Azure are leaders in hosting customizable web sites and services accessed via https/GRPC

The idea is that by leveraging existing**their** microservices, **your** coding obligation is dramatically reduced – in fact an off-the-shelf AI can generate many standard kinds of solutions.

EACH VENDOR HAS ITS OWN SPECIALITIES!

In contrast… companies like Oracle, Snowflake and Databricks all are basically big-data clouds.

Their customers bring immense amounts of data and need to extract various forms of knowledge. Of course these cloud also are specialists in data security!

Their services and tools help you describe and execute your data mining objectives (to a growing degree, with machine intelligence)

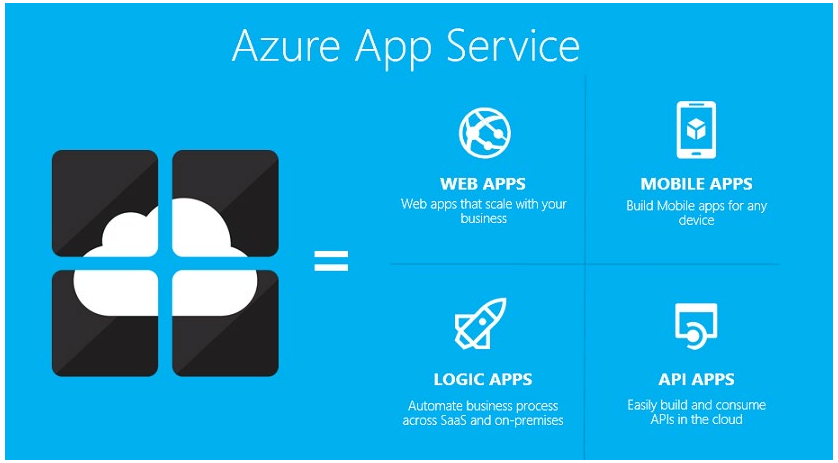
TODAY WE'LL FOCUS ON WEB HOSTING (WHICH IS THE SAME AS AGENTIC ML/AI HOSTING)

This focus fits with the topics of our course.

If you work in this area, you'll want to learn more about all of the different styles of clouds, and your code will probably talk to multiple clouds.

But learning any one, deeply, takes months of training

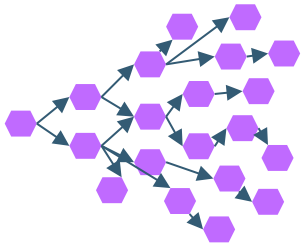
MANAGEMENT OF μ-SERVICE POOLS IS AUTOMATED



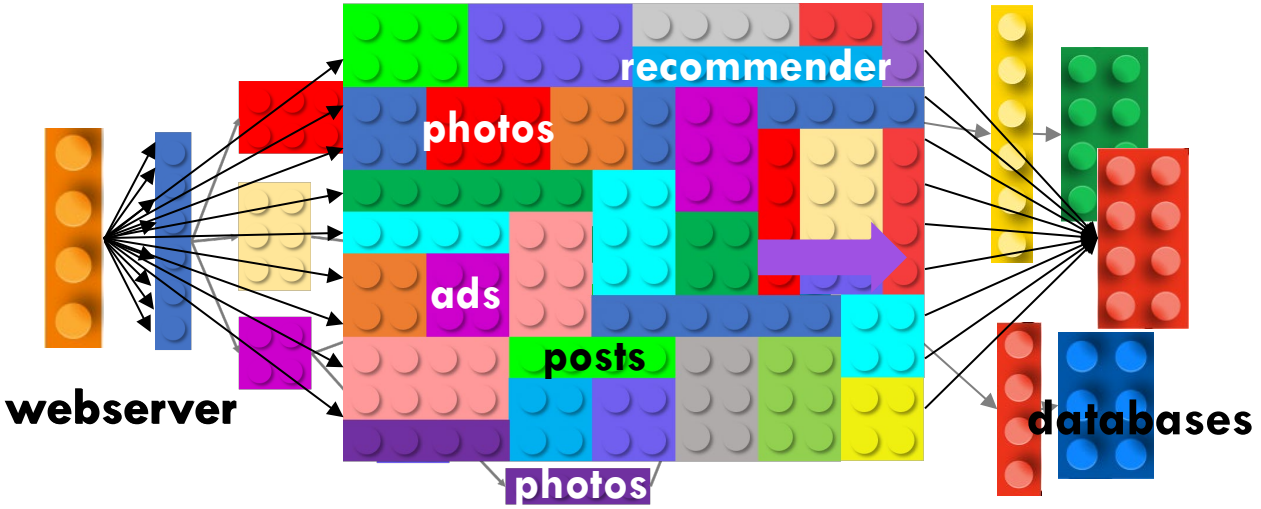
In Azure, for example, there is a tool called the "App Service" (we'll use it!)

The App Service manages a big collection of compute resources in the cloud. Developers can install your own services in it (as "containers"). Configuration files tell it when to launch them for you, automatically.

Among the features is a way for it to watch the queue of requests and automatically add instances or shut instances down to match loads.



Motivation for μ-services (Delimitrou)

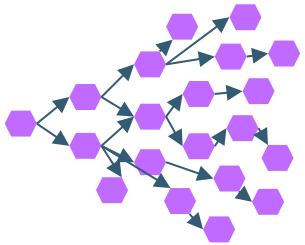


Monolith

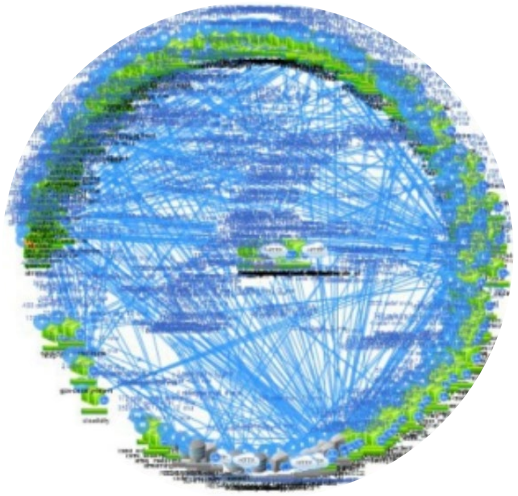
Microservices

Advantages of μ-services:

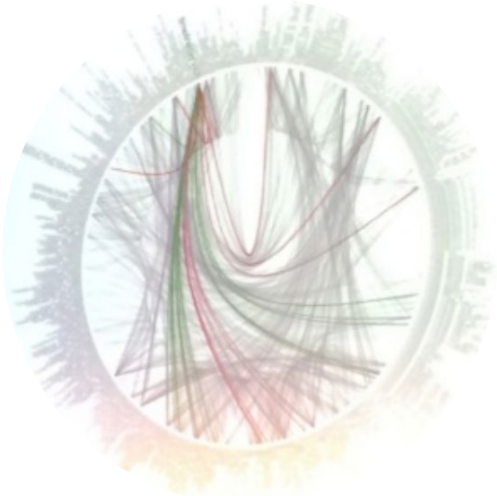
- Modular → easier to understand
- Speed of development & deployment
- On-demand provisioning, elasticity
- Language/framework heterogeneity



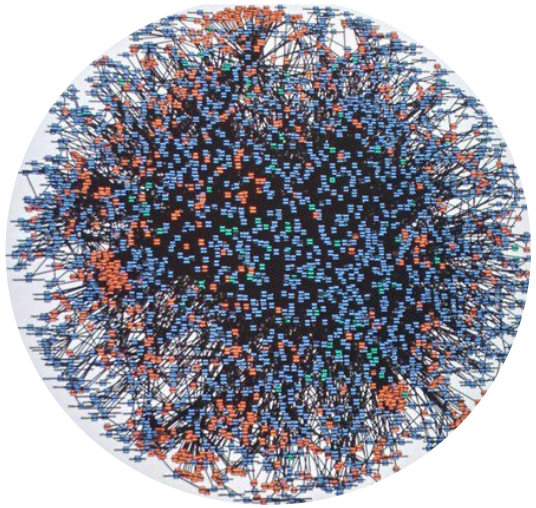
Elasticity management (one role of the App Service)



Netflix



Twitter



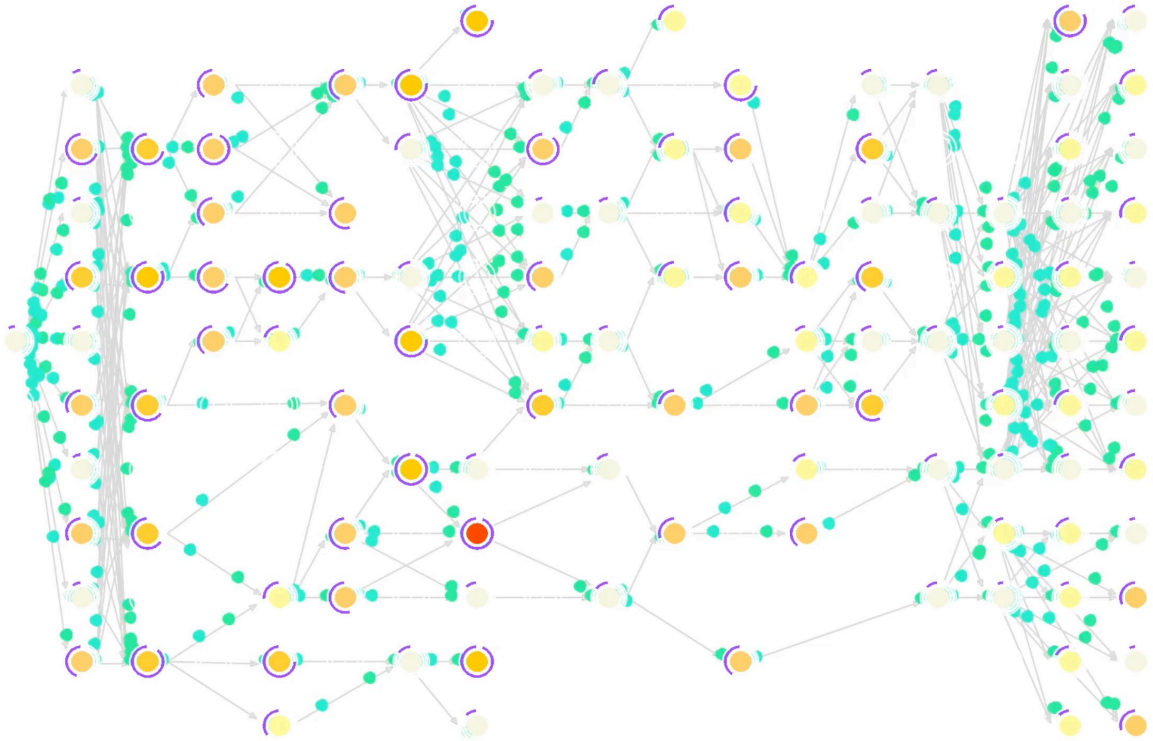
Amazon

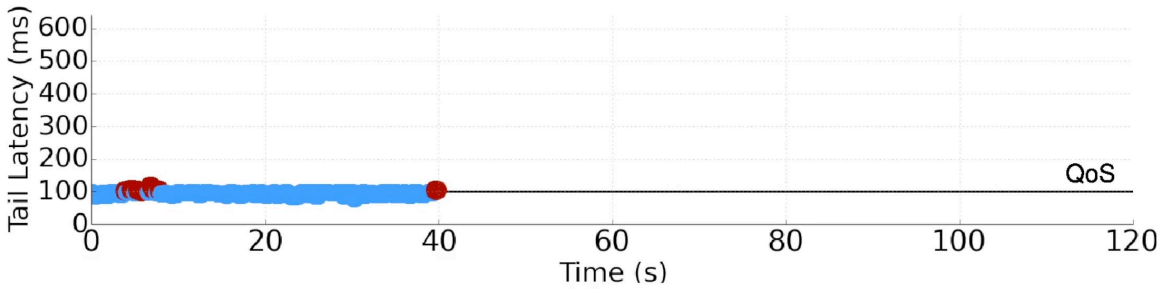
Brings many benefits… but complicates cluster management & performance debugging

Dependencies cause cascading QoS violations

Difficult to isolate root cause of performance unpredictability

Performance visualization





Dependencies cause cascading QoS violations

Empirical performance debugging → too slow, bottlenecks propagate

Long recovery times for performance

WHAT DID THE ANIMATION REVEAL?

The cloud scheduler watched each μ-service pool (each is shown as one dot, with color telling us how long the task queue was, and the purple circle showing how CPU loaded it is).

The picture didn't show how many instances were active – that makes it too hard to render. But each pool had varying numbers of instances. The App Server was automatically creating and removing instances.

Each time the scheduler realized that it should add instances to a slow service, some of the "deadline violations" went away.

WHAT ARE SOME VENDOR-SUPPLIED MICROSERVICES?

The list is endless! There are literally tens of thousands!

Common elements include fast sharded storage for structured data (tables, spreadsheets, databases), for photos or videos ("blobs"), and object/file storage. Databases and caching services are popular, too. And some microservices help you talk to other clouds!

There are also tools to help with communication from layer to layer: Kafka is one we've mentioned a few times.

WHAT DOES IT MEAN TO "ADD INSTANCES"?

For some applications (ones with NUMA threading for parallelism) we add instances by launching new threads on additional cores.

For others, we literally run two or more identical copies of the same program, on different computers! They use a "load balancer" to send requests to the least loaded instances.

And you can even combine these models…

STANDARDS FOR ELASTICITY

A very common approach focuses on **docker** container virtualization. A docker container is like a snapshot including a program (or several) and the files they depend on, and a kind of virtual image of the O/S environment they expect

Kubenetes is a widely popular docker container management framework. To define a new pool, you fill in a kind of form that it uses to manage the pool elastically according to your spec.

BUT ARE POOLS OF INSTANCES REALLY BEST?

This is just one of a few ways to get parallelism

Let's look at some of the choices and try to understand why the cloud favors the approach we just saw on the Delimitrou visualization.

HOW DOES LOAD BALANCING WORK?

Often we use a service like Kafka between the tier 1 app and the tier 2 service.

- Requests are enqueued in a "topic" (in fact a replicated shard instance managed by the Apache key-value store)
- Using a form of transaction, a tier 2 server can dequeue a request, process it, and reply. All of this is fault-tolerance.
- The client is notified, reads the reply, adds that data to the web response (page) it is constructing

WHY IS THIS LOAD-BALANCED?

New work goes to a service instance with capacity to do another request

The microservices management framework monitors the Kafka queue and if a backlog forms, launches more service instances. It downsizes if instances are sitting around doing nothing.

But the approach assumes that launching instances is very cheap

HOW DOES AGENTIC AI CHANGE THE GAME?

Clearly, we focus on queries that are some multimodal combination of text, images, voice, other data formats (radar, fMRI and CAT scans, ultrasound…)

We have new kinds of ecosystem elements such as LLMs for knowledge retrieval or inference, RAG databases, etc

And new kinds of security/privacy considerations.

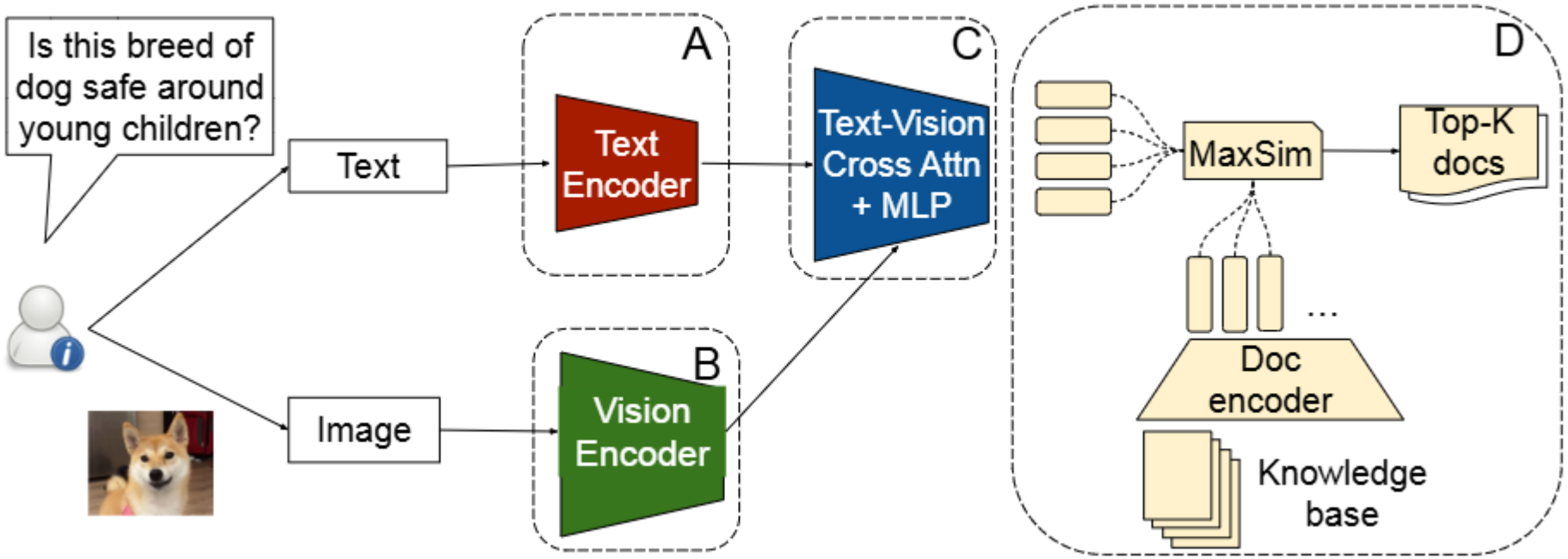
HOW DOES AGENTIC AI CHANGE THE GAME?

LLM/LRM hosting is different from traditional microservice hosting because **the models (and RAG databases) are huge – 10's of GB**

We can't spin this kind of container up or down quickly, so we need to create high performance "hosting specialist" systems (examples include Ray Serve and Torch Serve) that optimize to perform well despite this constraint.

Cornell's Vortex project is an example of a research effort in this area

ML VIEWED AS A MICROSERVICE PIPELINE



PreFMLR knowledge retrieval, finds documents relevant to a query.

… WHERE IS THE AGENTIC AI ASPECT?

Some sort of app is engaged with the actual end-user. Perhaps, a voice app on a mobile phone.

The app dispatches an AI query, but it could be an AI task request. Agentic AI covers both cases.

The ML as a service pipeline receives queries and performs tasks like retrieving relevant data (not always generating speech or pictures).

… WHERE ARE THE MASSIVE ML MODELS?

The ML microservice components each have an ML model. The code is pretty small but the ML model and perhaps other objects could be huge (tens or hundreds of gigabytes).

- E.g. a RAG database and its index
- Often we also need a GPU accelerator preloaded with the desired ML computational kernels.

This makes it harder to elastically resize a pool of instances.

SUMMARY?

The modern cloud is all about pipelines and graphs of components that talk to one-another, often using Kafka or a similar connector.

Elasticity enables load balancing

ML as a service will be challenging because it doesn't fit the elasticity concept easily.

SELF-TEST QUESTION

We saw that for a case like a music or video player, it might be easier to build a single-threaded container than to have each server instance handle multiple clients.

What overheads would this single-threading choice imply? List as many as you can think of.

SELF-TESTING QUESTION

When you use Facebook, which is very heavy on images and videos, it is almost uncanny how quickly your feed loads and how fast you can scroll around in it without delay.

What sorts of challenges do you think Facebook must have overcome to achieve that seamless experience?

If you were doing a Facebook clone, what microservices might you want to create (or leverage) to support it?

SELF-TEST QUESTIONS

Suppose that we create a web-hosted system for some sort of application like Amazon shopping, and it involves a set of new microservices.

The solution is popular, but it sees bursts of load and the deployment dynamically varies in response. You configure it for elasticity.

Would you expect your microservice pools to have the same number of replicas, or could one need a lot and another, very few?

SELF-TEST QUESTION

In your new job on the agentic AI interface team for purpleair.com, you need to create a collection of intelligent APIs.

Purpleair.com has a lot of value in its database of sensor data and its historical tracking of this data over time.

How would you design your agentic APIs to protect this valuable information while also giving very high value to purpleair.com's customers and partner companies?