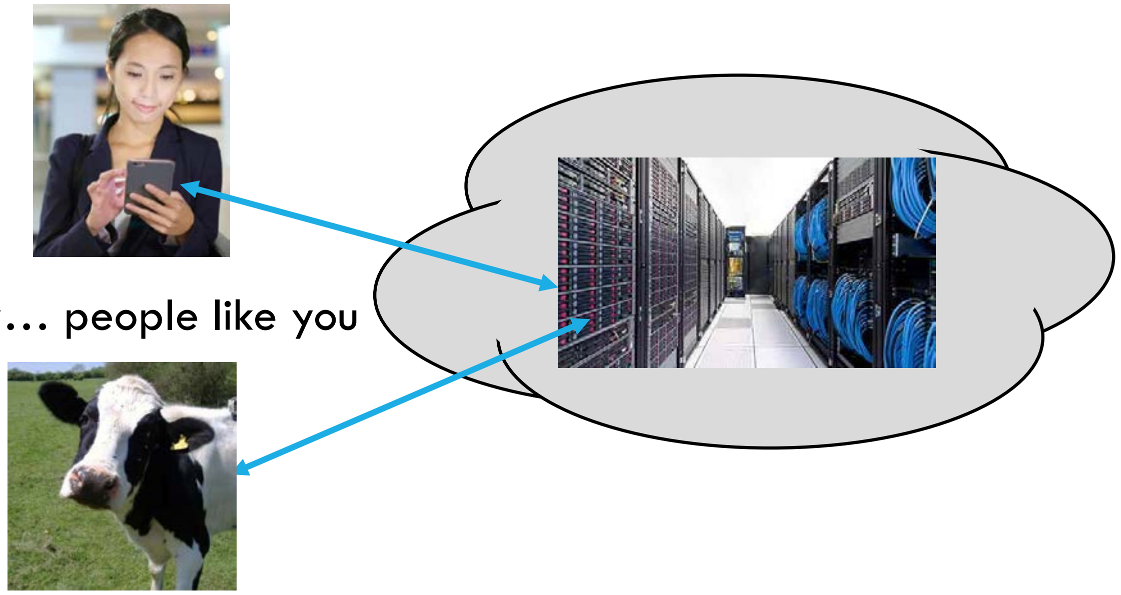
CS 5412/LECTURE 1 TOPICS IN CLOUD COMPUTING

Ken Birman
Spring, 2019

CLOUD COMPUTING


Today… people like you

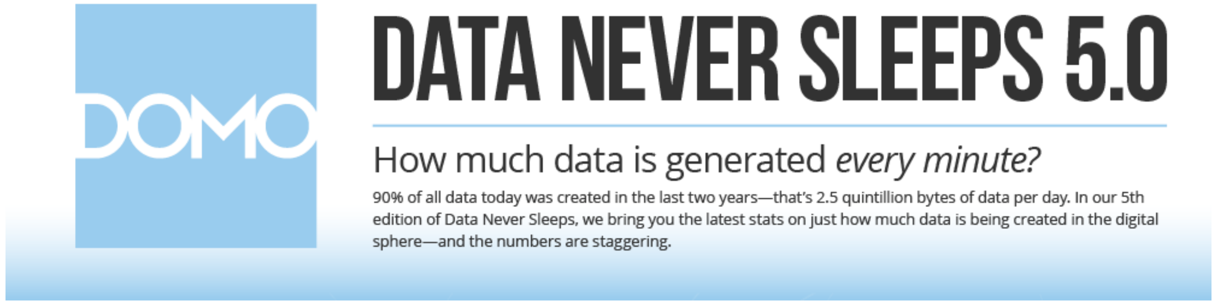
Tomorrow… Bessie!

CS5412 is…

- A deep study of a big topic.
- In spring 2019 our focus will be on “smart farming” in Azure IoT Edge.
- The farming focus leverages a Cornell and Microsoft interest (and an Azure product area) and makes it real.

Fog computing!

250 PB/DAY!

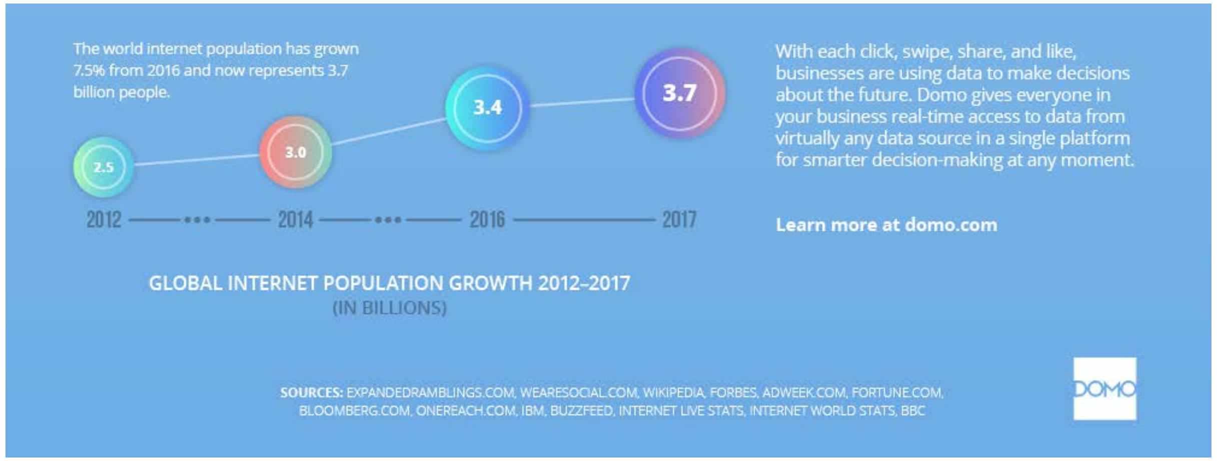


DOMO

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.

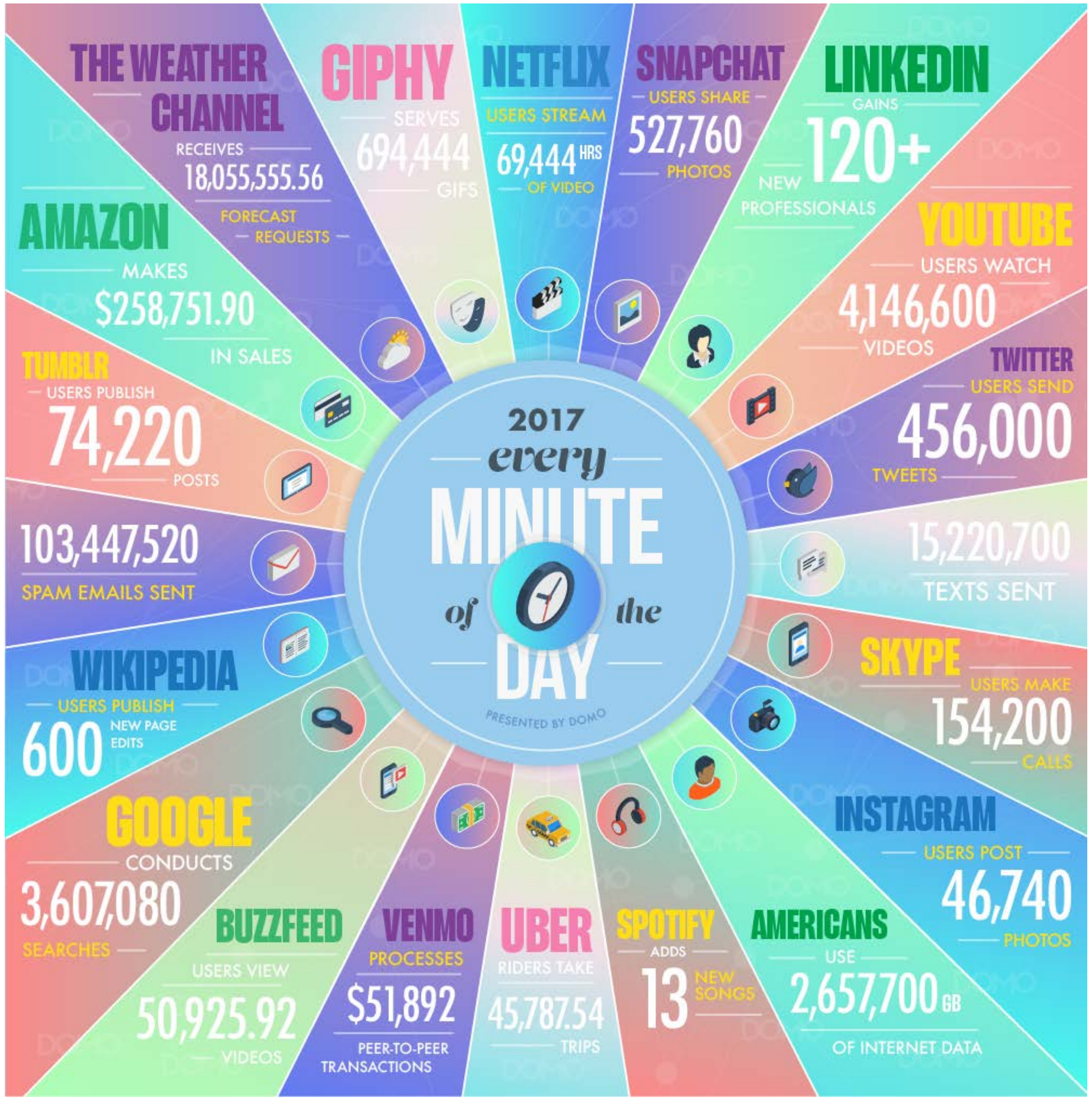
GLOBAL INTERNET POPULATION GROWTH 2012–2017 (IN BILLIONS)

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

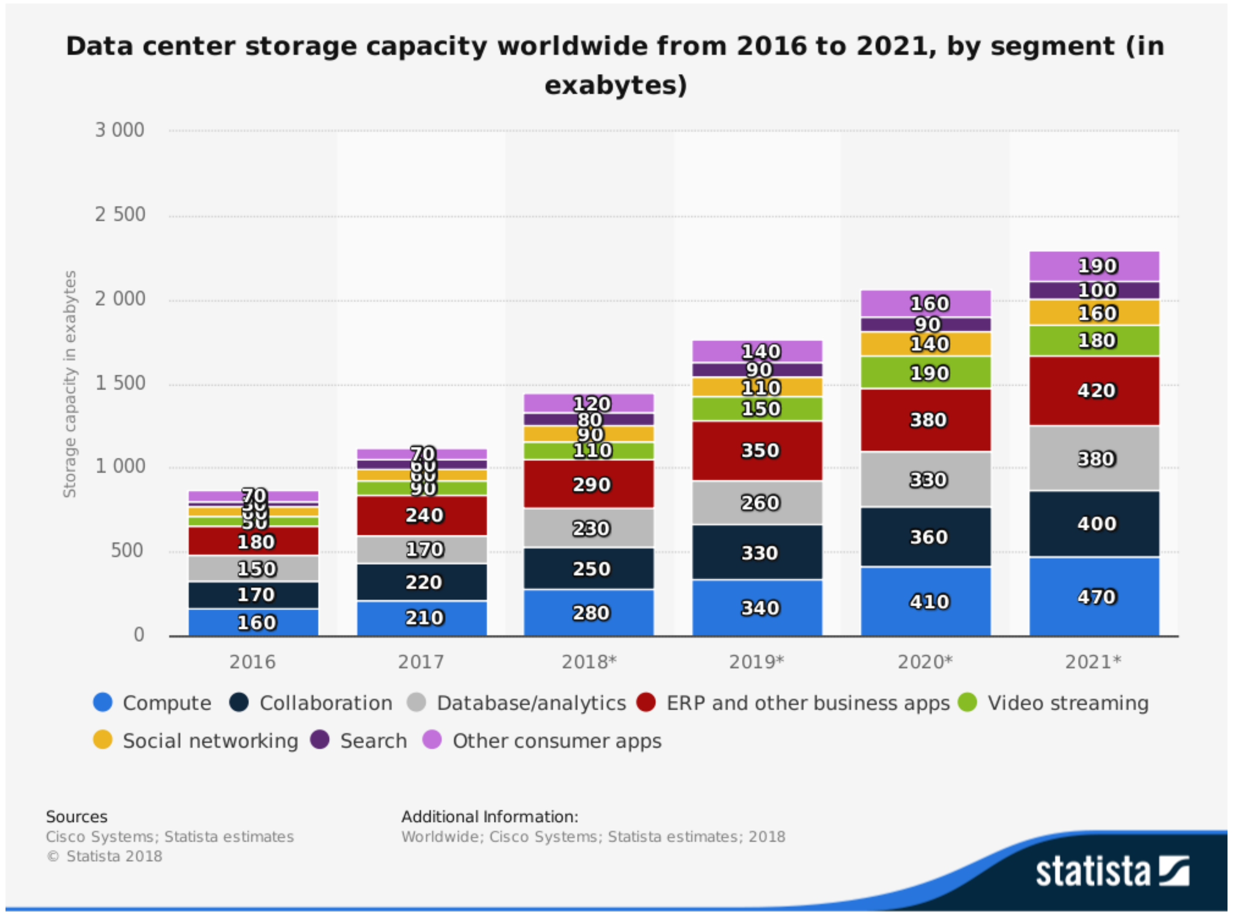

2017 every MINUTE of the DAY
PRESENTED BY DOMO
THE WEATHER CHANNEL RECEIVES 18,055,555.56 FORECAST REQUESTS
GIPHY SERVES 694,444 GIFS
NETFLIX USERS STREAM 69,444 HRS OF VIDEO
SNAPCHAT USERS SHARE 527,760 PHOTOS
LINKEDIN GAINS 120+ NEW PROFESSIONALS
YOUTUBE USERS WATCH 4,146,600 VIDEOS
TWITTER USERS SEND 456,000 TWEETS
15,220,700 TEXTS SENT
SKYPE USERS MAKE 154,200 CALLS
INSTAGRAM USERS POST 46,740 PHOTOS
AMERICANS USE 2,657,700 GB OF INTERNET DATA
SPOTIFY ADDS 13 NEW SONGS
UBER RIDERS TAKE 45,787.54 TRIPS
VENMO PROCESSES $51,892 PEER-TO-PEER TRANSACTIONS
BUZZFEED USERS VIEW 50,925.92 VIDEOS
GOOGLE CONDUCTS 3,607,080 SEARCHES
WIKIPEDIA USERS PUBLISH 600 NEW PAGE EDITS
103,447,520 SPAM EMAILS SENT
TUMBLR USERS PUBLISH 74,220 POSTS
AMAZON MAKES $258,751.90 IN SALES

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)

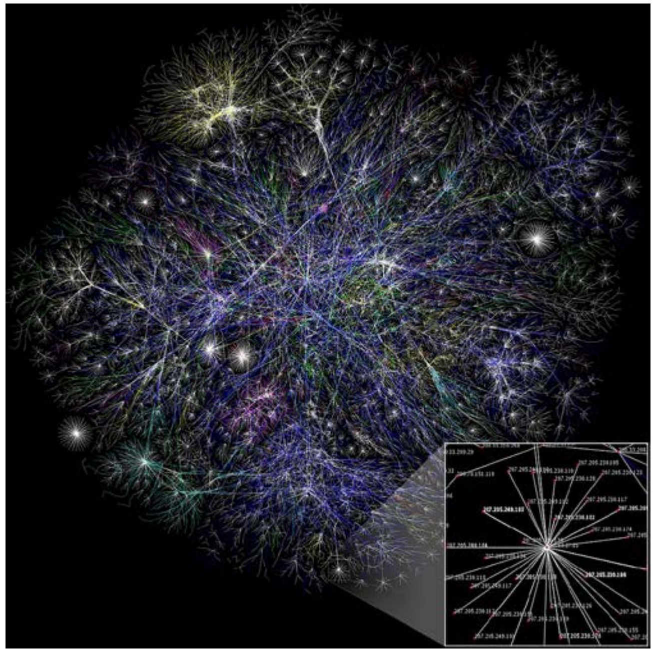
The Internet has about 2B websites, and of these, 644M have "active content"

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



YOU TELL ME…

Total size of all the digital information acquired about you per day?	• 2 GB/day
Total amount of stored data in the Internet?	• 400 Exobytes
How many web sites in the Internet today?	• 1B
Speed of an Internet backbone link?	• 100 Gbps per fiber
Speed of an Internet backbone router?	• 322 Tbps



CLOUD PROVIDERS NEED TO THINK “BIG”!

Google: 40,000 queries per second (1.2 Trillion per year)

YouTube: 1.9B active users per month, viewing 5B videos per day

Facebook: 2.23B active users, 8B video views,15M photos uploaed per day

Cloud: Nearly 4B of the world’s 7B accessed cloud resources in 2018

… the scale of computing to support these stories is just surreal!

… AND THEY NEED TO THINK "PARALLEL"

At these scales, no computer can keep up.

By the nature of the cloud, it *has* to be massively parallel!

YOU TELL ME…

How much DRAM in a datacenter server?	• 512GB-12TB
How fast is a single CPU in a NUMA machine?	• 1.8Ghz
How many cores does a NUMA server hold?	• 72 cores
How many threads per core when hyperthreading is enabled?	• 2
How many servers per rack?	• 48
How many servers total in a datacenter?	• ~500,000
How deep is a typical datacenter COS/SPINE routing tree?	• 6 layers
How fast is a datacenter network today?	• 56Gbps
How big is 1-way node to node latency (due purely to the network)?	• 1.25us
What is a typical round-trip latency for a datacenter RPC?	• 100us-25ms

YOU TELL ME...

How much storage capacity in a server's RAID SSD drive?	• 800 GB
How much in a RAID configuration?	• 100 TB
How much storage in a cutting edge rotating disk (HDD)?	• 15 TB
How much capacity in a memory-mapped Optane drive?	• 16 GB
Peak PCIe bus data transfer speeds, per bus?	• 8.5GB/s
… peak transfer rate for an <u>single</u> SSD unit?	• 200 MB/s
… access delay for a block of SSD storage?	• 100us
Seek time for an HDD?	• 2.5-10ms

YOU TELL ME...

Size of one email in HTML encoding?	• 10KB
… a typical raw photo?	• 4MB
… that same typical photo, in a compressed format?	• 250KB
… a typical encoded 3-5 minute music video?	• 10MB/min
… a full length movie?	• 4GB
Maximum standard IP packet size?	• 1KB
Jumbo frame size?	• 8KB

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

YAHOO EXPERIMENT

A sprint to render your web page!

At Yahoo, they tried an "alpha/beta" experiment

Customers who saw fast web page rendering (below 100ms) were happy.

For every 100ms delay, *purchase rates noticeably dropped*.

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

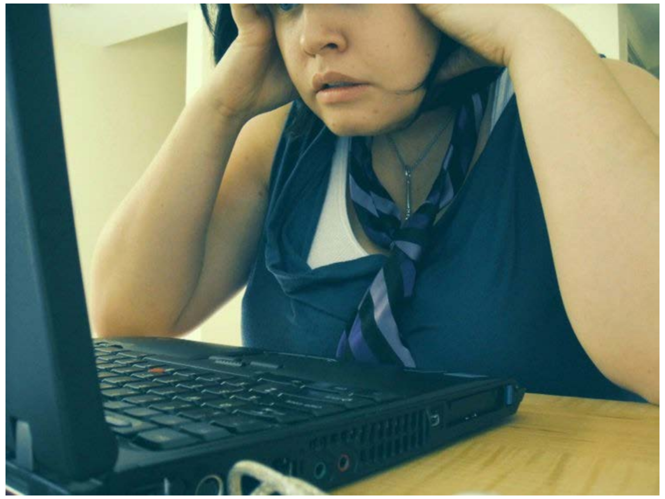
STARTING AROUND 2006, AMAZON LED IN REINVENTING DATA CENTER COMPUTING

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 relastic 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.

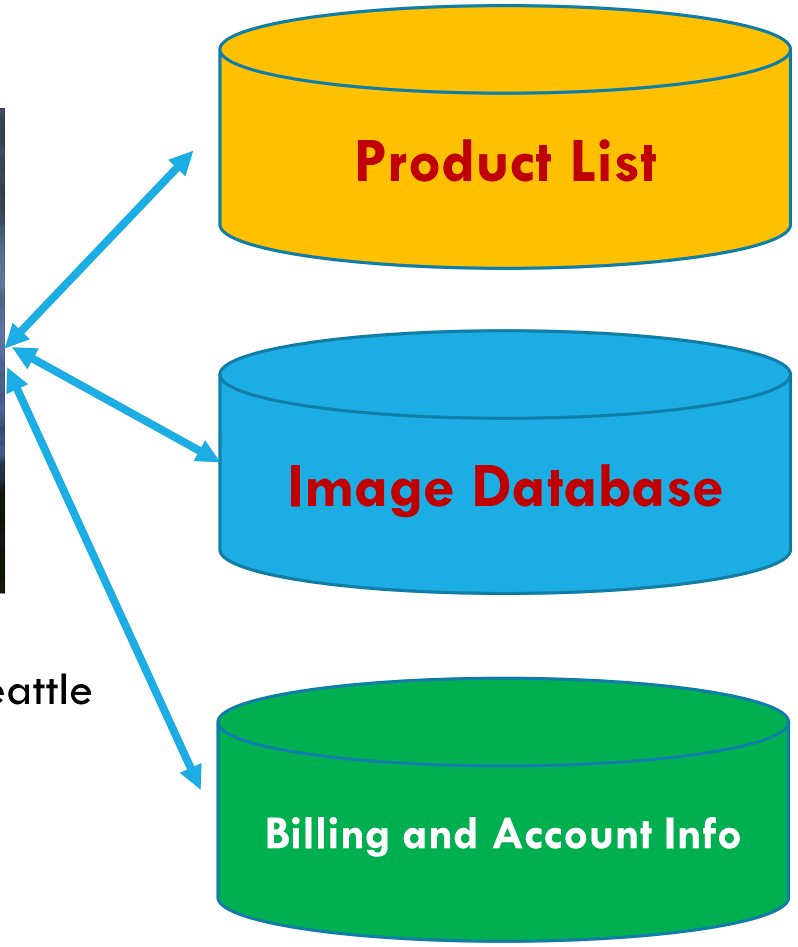
OLD APPROACH (2005)



Computers were mostly desktops

Internet routing was pretty static, except for load balancing

Web Server
built the page… in Seattle



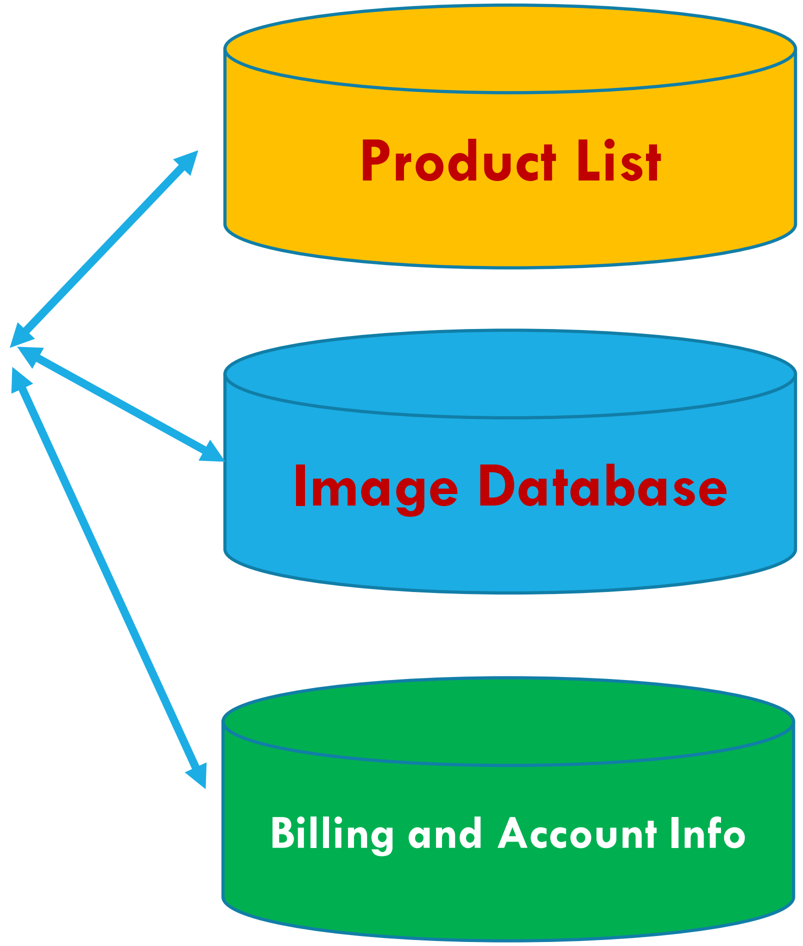
Databases held the real product inventory

NEW APPROACH (2008)

Computers became lightweight, yet faster

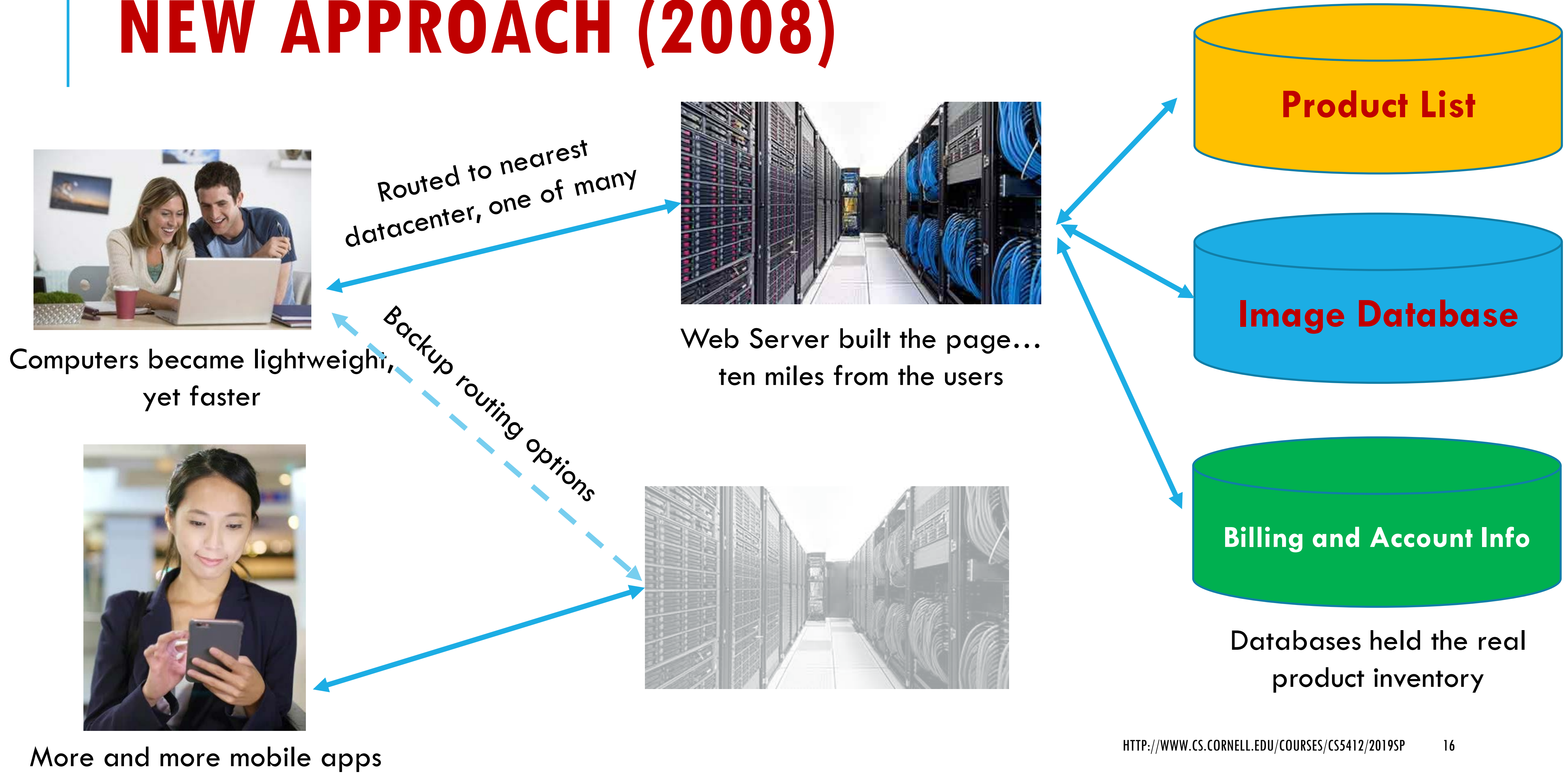
Routed to nearest datacenter, one of many

Web Server built the page… ten miles from the users



Databases held the real product inventory

NEW APPROACH (2008)

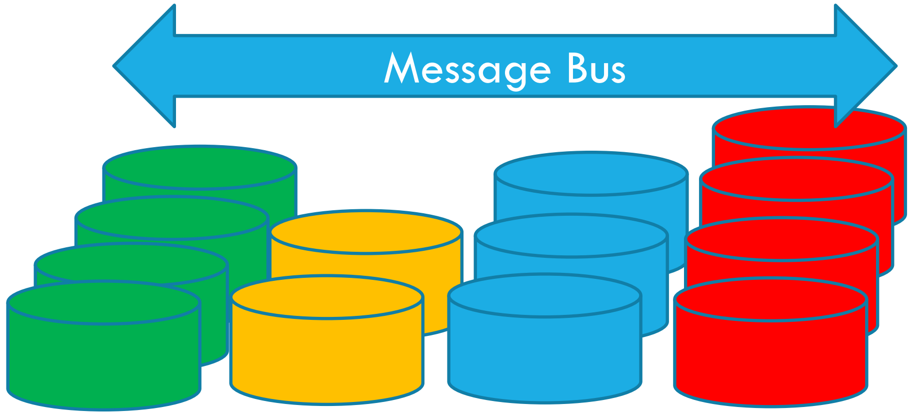

Routed to nearest datacenter, one of many
Backup routing options
Computers became lightweight, yet faster
More and more mobile apps
Web Server built the page… ten miles from the users
Product List
Image Database
Billing and Account Info
Databases held the real product inventory

NEW APPROACH (2008)

Desktops with snappier response

Routed to nearest datacenter, one of many

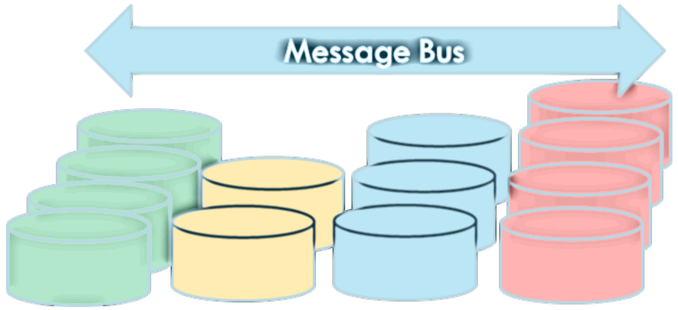
Web Server becomes simpler and does less of the real work



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)

TODAY’S CLOUD

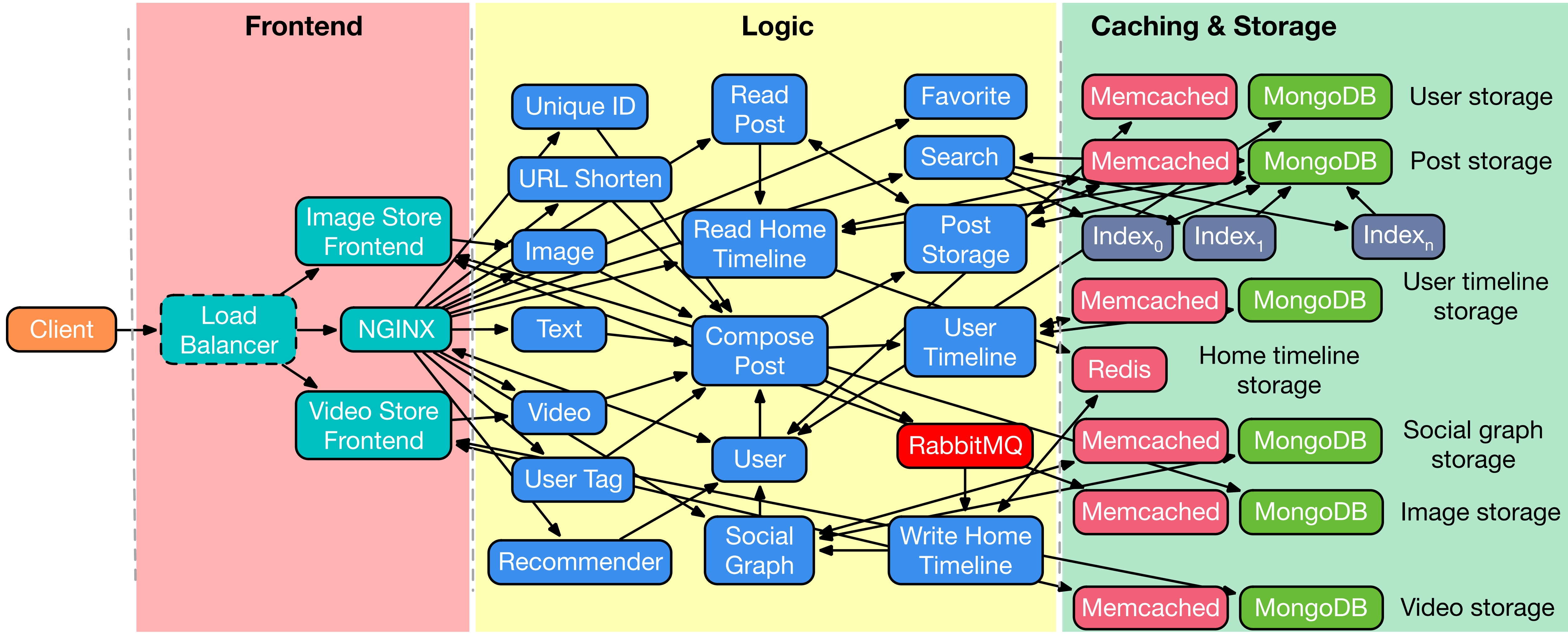
Tier one runs on very lightweight servers:

- They use very small amounts of computer memory
- 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.

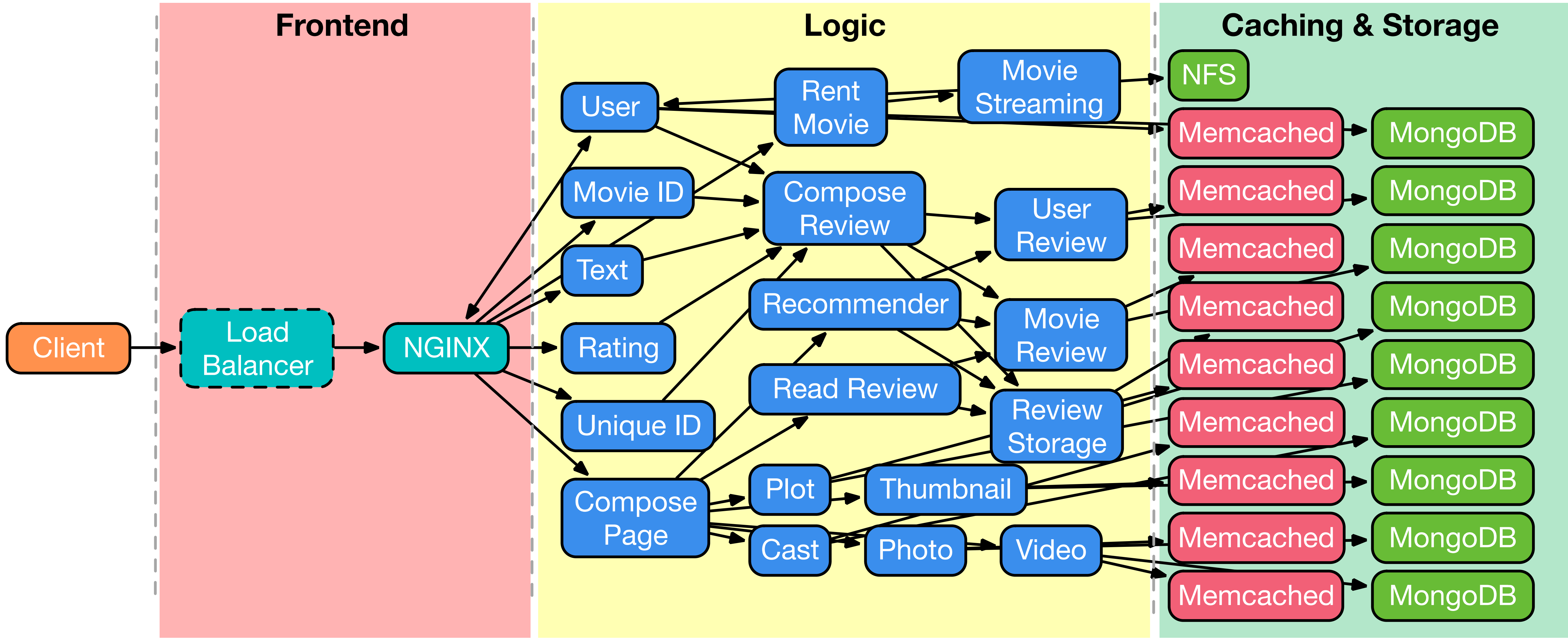
End-to-end Microservices (from Christina Delimitrou)

Social Network



End-to-end Microservices (from Christina Delimitrou)

Media Service



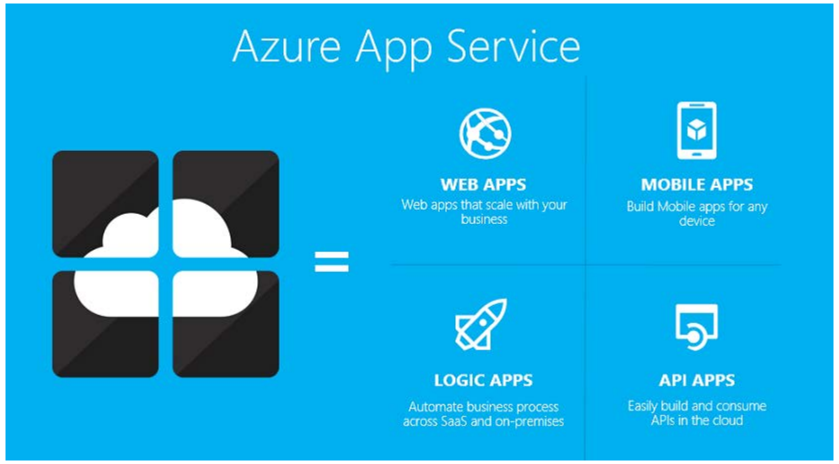
EACH MICROSERVICE IS A PARALLEL “POOL”!

Every one of those little nodes is itself a small elastic pool of processes

A microservice (μ-service) is a kind of program designed so that the data center can run 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!

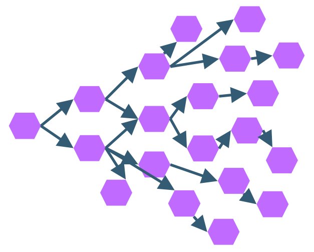
THESE POOLS ARE MANAGED AUTOMATICALLY


Azure App Service
WEB APPS
Web apps that scale with your business
MOBILE APPS
Build Mobile apps for any device
LOGIC APPS
Automate business process across SaaS and on-premises
API APPS
Easily build and consume APIs in the cloud

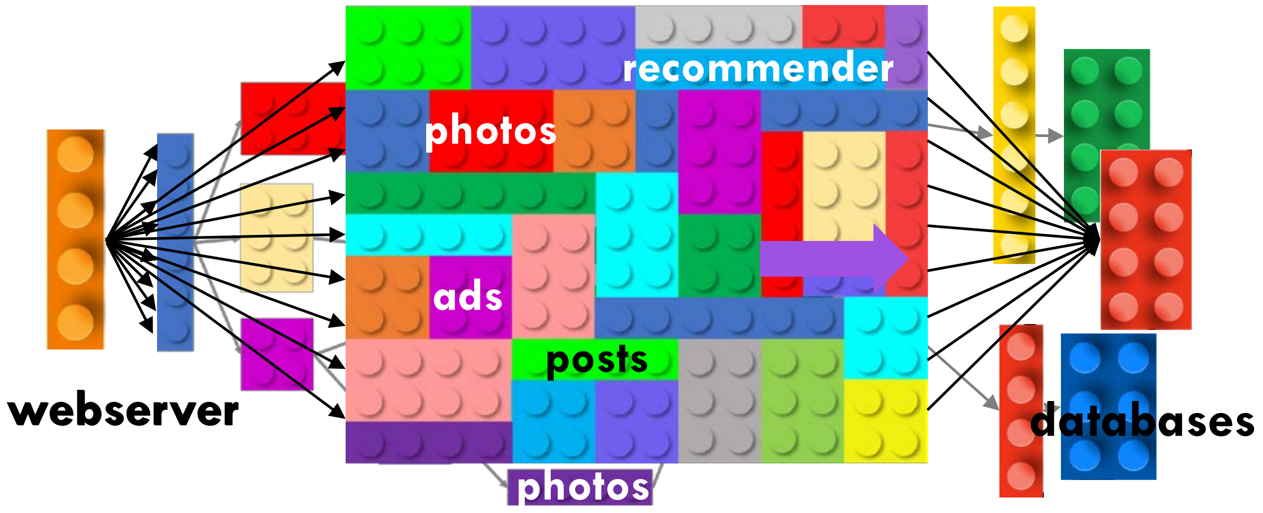
In Azure, for example, there is a tool called the "App Service" (we'll use it!)

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)



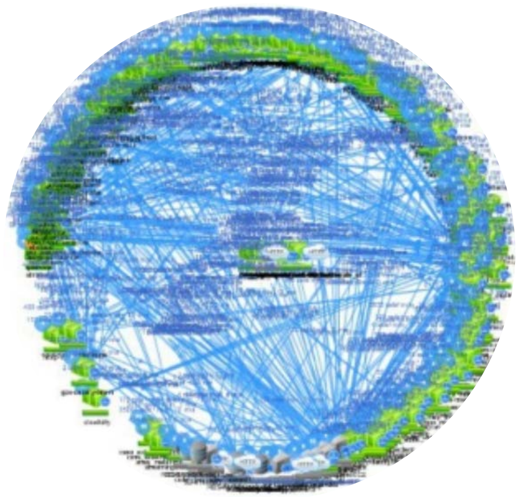
Monolith

Microservices

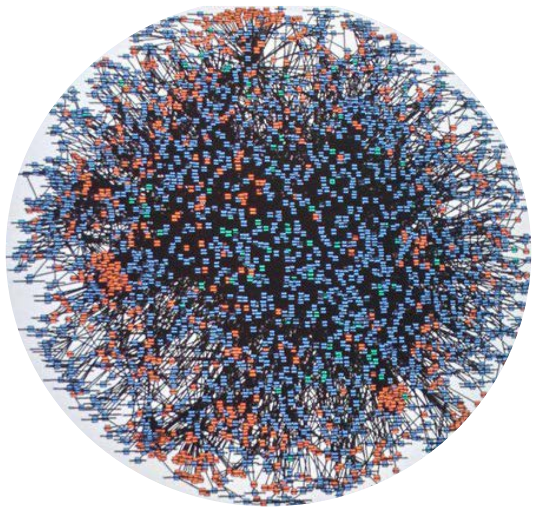
Advantages of µ-services:

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

Performance management (Delimitrou)


Netflix

Twitter

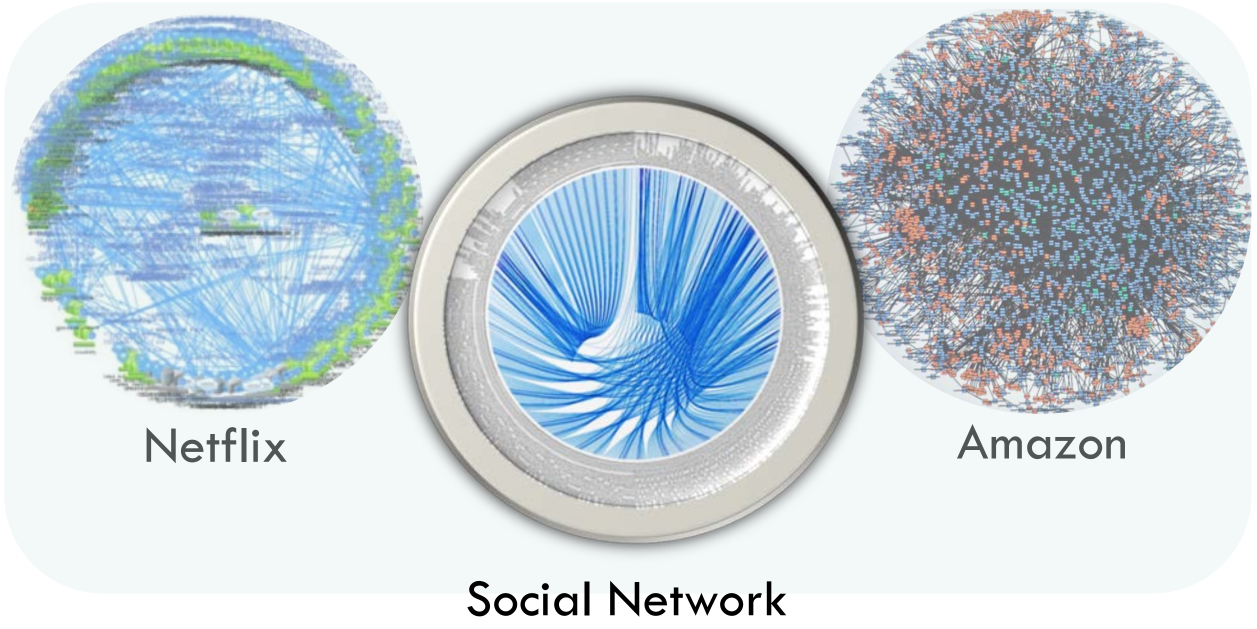

Amazon

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 WE JUST SEE?

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 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…

SCALABILITY ISSUES ARISE EVEN INSIDE A SINGLE μ-SERVICE INSTANCE

We've been acting as if each μ-service is a set of "processes" but ignoring how those processes were built.

In fact they will use parallel programming of some form because modern computers have NUMA architectures.

How do cloud developers think about this form of parallelism?

DEEP DIVE: BEST WAY 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.

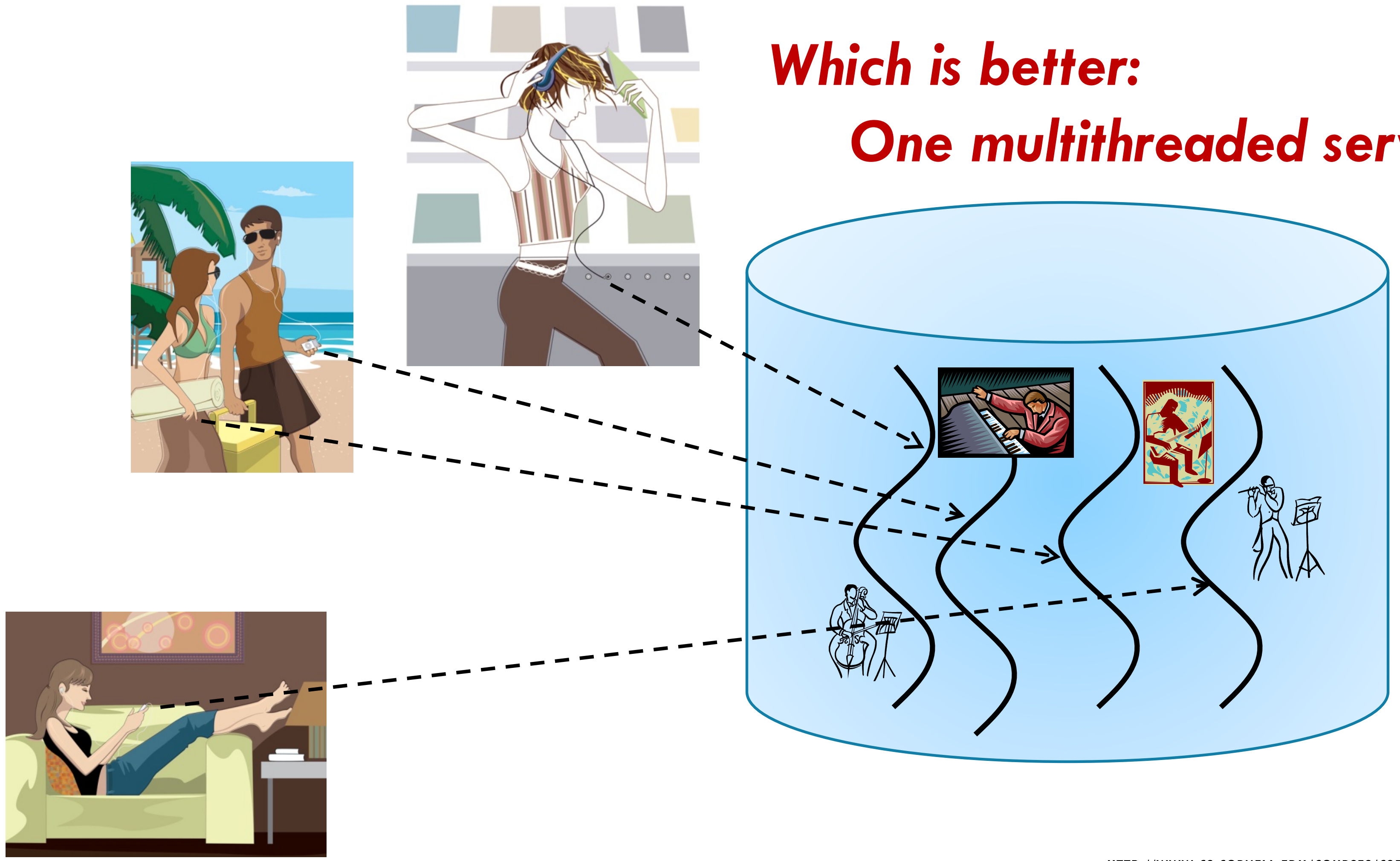
This may feel like a small detour but actually is typical of CS5412

Slight digression

TIER-ONE FOCUSES ON EASY STORIES

Which is better:

One multithreaded server, per node?



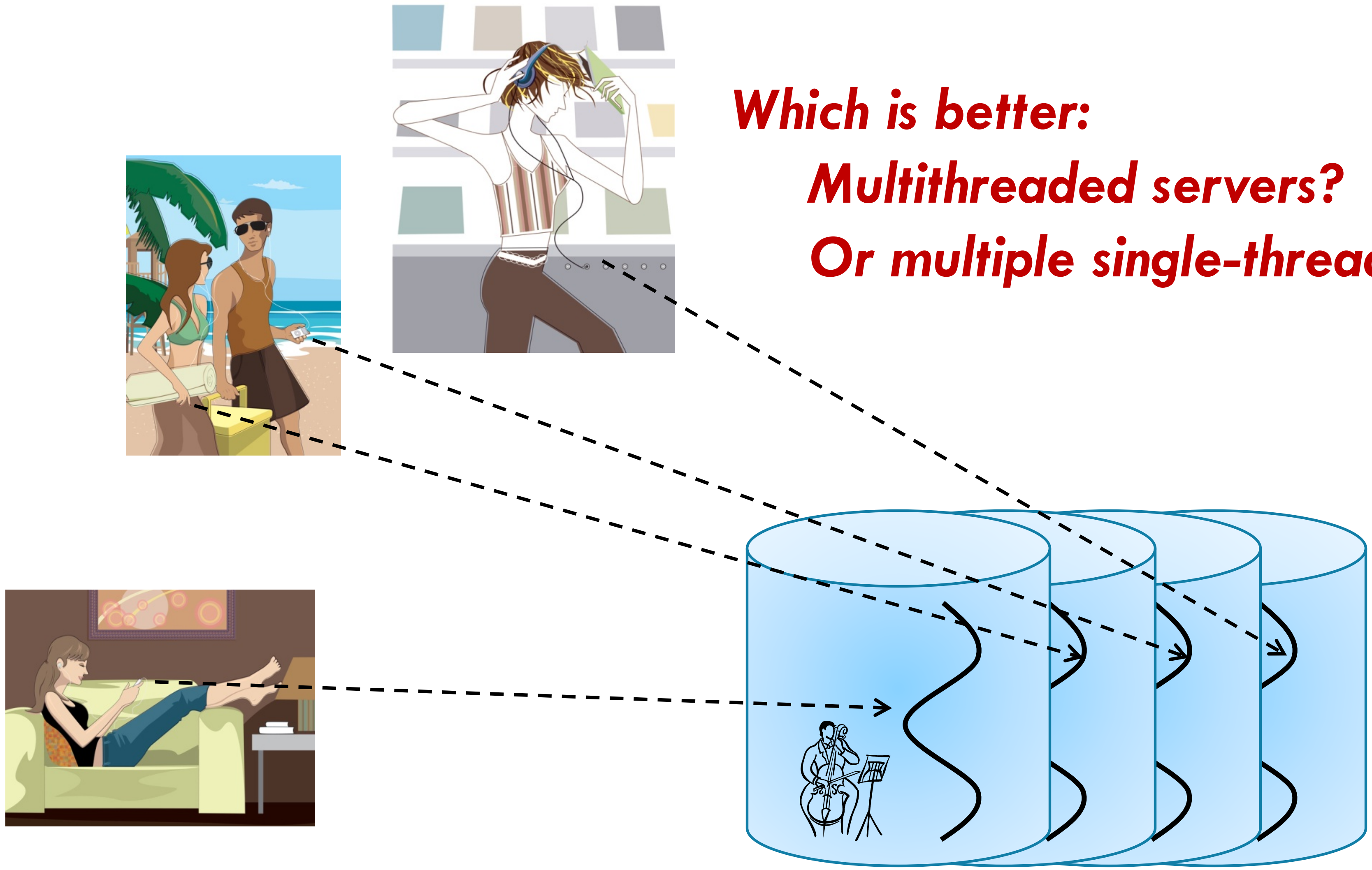
Slight digression

TIER-ONE FOCUSES ON EASY STORIES

Which is better:

Multithreaded servers?

Or multiple single-threaded servers?



Slight digression

WHAT YOU LEARNED IN O/S COURSE

Probably, you just took a class where the big focus was concurrency and threaded programs, and probably they taught you to go for multithreading

The story you heard was something like this:

- Because of Moore's law, modern computers are NUMA multiprocessors.
- To leverage that power, create lots of threads, link with a library like "pthreads", and request that your program be allocated multiple cores.
- Use thread synchronization/critical sections to ensure correctness.

Slight digression

BUT IS THIS THE RIGHT CHOICE?

First, we should identify other design options, even ones that look dumb at first glance.

Then we can evaluate based on a variety of considerations:

- Expected speed and scaling (more is good)
- Complexity of the solution (more is bad)
- Cost of the solution (more is bad)

WHAT YOU LEARNED IN O/S COURSE

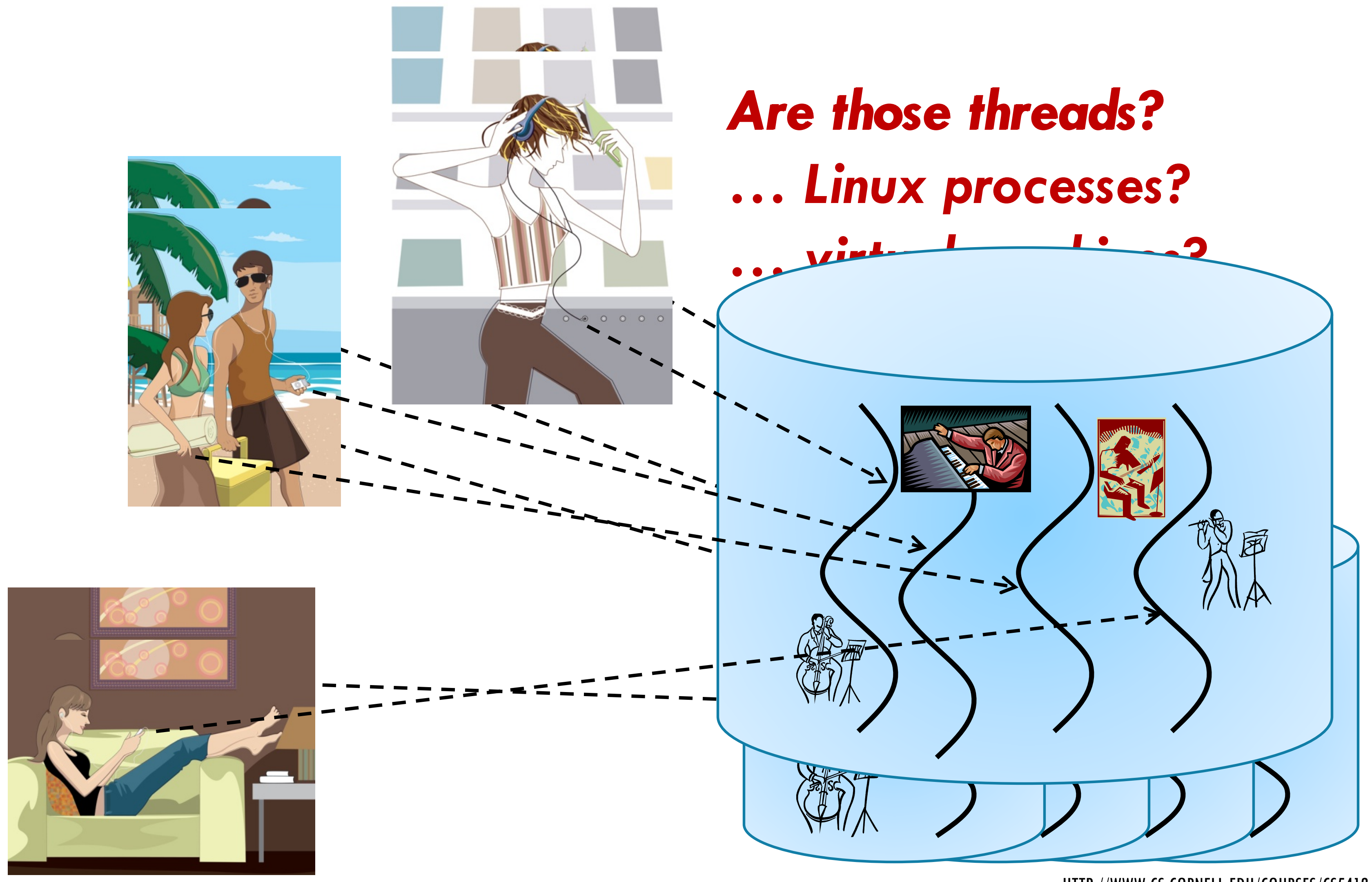
Another thing you learned about was the *virtual machine* approach.

With true virtualization, programs run on private virtual machines.

Today, a recent alternative is “containers”, which give the illusion of a private virtual machine in a Linux process address space, not a true VM.

Slight digression

… EVEN OUR "EASY" CLOUD POSES CHOICES!



Are those threads?

… Linux processes?

… virtual machines?

Slight digression

HOW WOULD YOU DECIDE BETWEEN THEM?

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

Slight digression

CHOICES AS A TABLE

Option	Speed	Complexity	Costs
1. Multithreaded server	Poor	Poor	Development: expensive. Use of resources: best. But may be hard to administer.
2. Single-thread + VM	Poor	Good	Inexpensive development, but inefficient use of memory resources, high overheads
3. Single-threaded process model	Very good if interference can be avoided	Least complex!	Inexpensive development, but administering to ensure that the processes won't somehow interfere can be tricky.
4. Single-threaded, containers	**Best of all.**	**Just like a single-threaded process model.**	**Inexpensive development. The approach helps by protecting containerized apps from most forms of interference.**

Slight digression

WHY CONTAINER MODEL "WINS"…

We want the edge of the cloud to be as cost-effective as possible.

Development and management complexity is one kind of cost.

Also think about CPU load, memory, and context switching overheads:

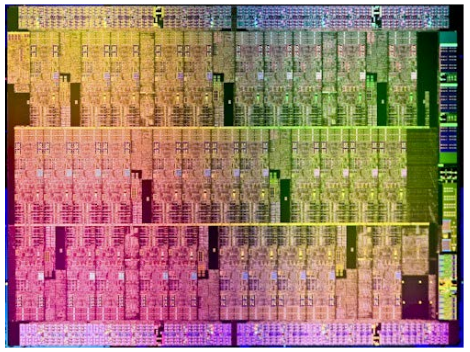
- Best would be a single program with multiple threads
- Containers offer isolation and can share code pages, saving memory

Slight digression

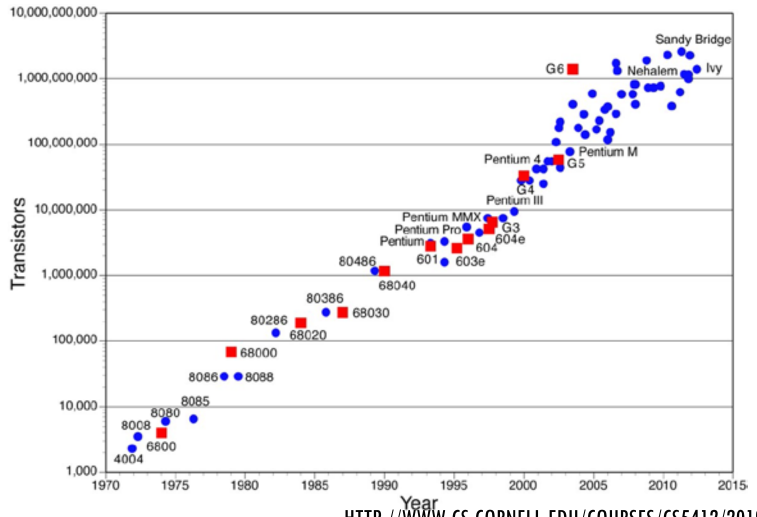
WHY DOESN'T A MULTI-THREADED SOLUTION PERFORM BEST?

This is almost always a surprise to CS5412 students. To appreciate the issue, we need to understand more about modern server hardware

Early days of the web were before we fell off Moore's curve. Today's servers are *NUMA machines* with many cores.



32-core Intel Aubrey chip. Some servers have as many as 128 cores today!



Slight digression

NUMA ARCHITECTURE

A NUMA computer is really a small rack of computers on a chip

Each has its own L2 cache, and small groups of them share DRAM.

With, say, 12 cores you might have 4 DRAM modules serving 3 each.

- Accessing your nearby DRAM is rapid
- Accessing the other DRAM modules is much slower, 15x or more

NUMA hardware provides cache consistency and locking, but costs can be quite high if these features have much work to do.

Slight digression

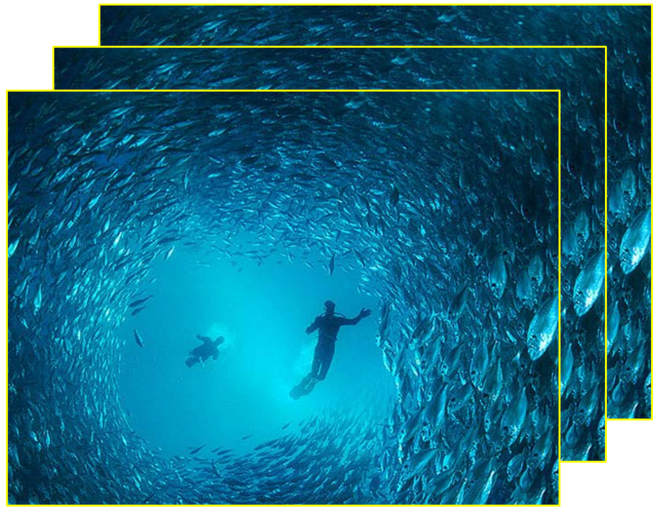
MULTITHREADING ON A NUMA.

On a NUMA architecture, many threaded programs *slow down* on > 1 cores! Many reasons:

- Locking and NUMA memory coherency,
- Weak control over "placement" (which memory is on which DRAM?),
- Higher scheduling delays,
- Issues of reduced L2 cache hit rate

Slight digression

DEEP DIVE ON THAT QUESTION



What about true virtualization?

- In effect, we will have perhaps 12 VMs. Our programs might still have threads, but we will mostly use just one core per program (or per VM)
- Now we won't have memory contention: each program is isolated. On the other hand, we use more memory, because sharing is harder.
- But the virtualization layer causes page-table translation slowdown, and I/O operations might also be slower. DMA might not work.

Slight digression

CONTAINERS WIN!

A container is a normal Linux process with a library that mimics a full VM.

- The system looks “private” but without full virtualization.
- Eliminates the 10% or so performance overheads seen with true VMs.
- Also, containers launch and shut down much faster than a full VM, because we don’t need to load the whole OS.
- We won’t see NUMA memory contention problems.
- Security and “isolation” are nearly as good as for VMs.

Popular options? Kubenetes and Docker.

Slight digression

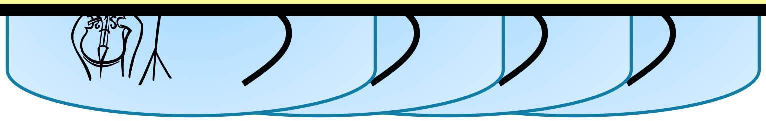
AHA!

Build <u>single threaded</u> server, optimized to run on behalf of a single client.

Run lots of copies on each NUMA server (perhaps hundreds).

Use containers for isolation, container O/S smart about DRAM memory issues.

- *Share read-only pages only between cores that share the same DRAM*
- *Make one copy per DRAM for read-only shared data, like code pages!*



Slight digression

SECONDARY QUESTION THIS POSES

If we have a huge number of "separate" tasks running, in distinct containers, maybe different servers, how consistent should the data they use be?

Strong consistency: The tasks should share a single database

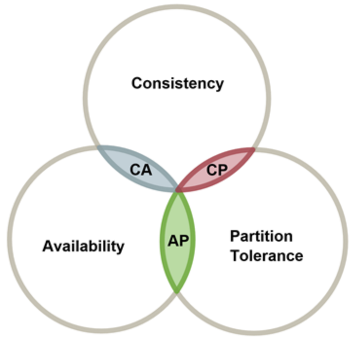
Weaker consistency:

- The tasks have some kind of read-only data, computed "last night".
- They can also enqueue update tasks for offline processing.
- The tasks might also guess at the effect of the updates, but the offline version will "win" if a conflict occurs.

IN THE CLOUD, NOT EVERY SUBSYSTEM NEEDS THE STRONGEST GUARANTEES

At Berkeley, Eric Brewer captured this insight with a “theorem”

CAP stands for “Consistency, Availability and Partition Tolerance”



Basically, Eric argues that:

- The theoretically “best” solution often brings heavy costs.
- Consistency is one example: conflicting database updates can be forced into an agreed order, but this takes time and involves node-node dialog (hence **¬P**).
- Remember that to earn the most money, you need the fastest possible responses. Eric concluded that this means you might have to relax consistency: **CAP**.

DEFINITIONS (SLIGHTLY INFORMAL!)

Consistency: The system responds using *the most current values* (updates). *Conflicting updates are performed in some system-selected order by all replicas.* Queries thus will see a "single system" and will be up to date.

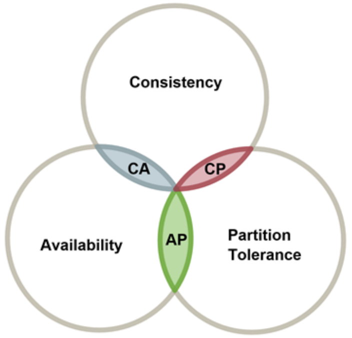
Availability: The system is *rapidly responsive*, and will *self-repair* if some single component fails, restoring normal functionality asap. Of course fault-tolerance isn't always possible: if too many components fail all at once, availability is lost.

Partition Tolerant: A data center can have network issues, or entire services can be down. Yet *as seen from outside, the cloud should continue to respond even if its first-tier services are temporarily unable to talk to some inner services* they would normally depend upon.

IN THE CLOUD, NOT EVERY SUBSYSTEM NEEDS THE STRONGEST GUARANTEES

At Berkeley, Eric Brewer captured this insight with a “theorem”

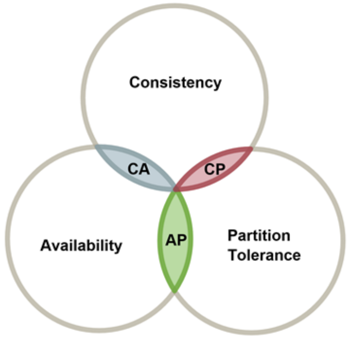
CAP is short for “Consistency, Availability and Partition Tolerance”



Basically, Eric argues that:

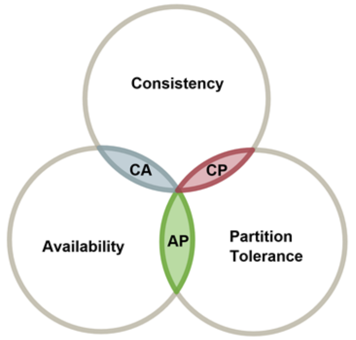
- The theoretically “best” solution often brings heavy costs.
- Consistency is one example: conflicting database updates can be forced into an agreed order, but this takes time and involves node-node dialog (hence $\neg$**P**).
- Remember that to earn the most money, you need the fastest possible responses. Eric concluded that this means you might have to relax consistency: **~~C~~AP**.

ERIC BREWER'S CAP THEOREM



- Relax consistency (**C**),
- Gain faster response (**A**).
- Generate responses even when unable to talk to back-end servers (**P**).

BASE METHODOLOGY



BASE ≡ "CAP in practice"

Invented at eBay, adopted by Amazon and others

- **B**asic **A**vailability, **S**oft State and **E**ventual Consistency

"Use CAP. You can clean up later."

How BASE works: cache data but don't worry about cache entries getting stale (hey, they were valid a little while ago).

TODAY'S CLOUD IS A CAP+BASE "WORLD"

By and large, cloud systems try to manage with weak consistency.

But for IoT this may not be good enough! We will see why in future lectures

So one of our primary challenges will be to understand when we need stronger properties, and how to obtain the needed guarantees at the lowest cost! Fortunately, there are good options today.

… CLOUD COMPUTING ISN’T AN O/S COURSE

Even so, we need to keep doing these deep dives to understand how to make the best use of a large data center to do cloud computing cost-effectively at massive scale.

In the two examples we’ve looked at today, we saw how these goals can lead to an O/S choice.

We will dip down into many “technology areas” this spring, but always driven by this need to understand cost-effective scalable computing.

HOW MANY IOT DEVICES? HOW MUCH DATA?

23.5B connected devices today, will rise to 75B by 2025.

A single camera or video could generate GB/s of "new data".   An microphone capturing high-quality audio produces hundreds of MB/s. We definitely can't transfer all of this to the cloud, or process all of it.

So the world of IoT will require "new ideas" simply because the bulk of the information resides out on the edge and not in the cloud.

PUZZLE THIS CREATES

IoT is introducing new dimensions of scalability never seen before!

These force us to think about shifting some tasks, like deciding which data is interesting, from deep inside today's cloud to the edge.

But will that shift break the things that make the cloud scalable?

SOME TOPICS WE WILL TALK ABOUT

Azure's IoT Edge

- ➢ Sensors and actuators: what are they?
- ➢ How smart are they likely to be?
- ➢ Customizing the IoT Edge
- ➢ Filtering and transforming data

Fault Tolerance

Challenges of dealing with real-time data

- ➢ Time synchronization, temporal storage
- ➢ Concepts of consistency for the cloud edge

Azure's Intelligent IoT Cloud

- ➢ Details of the μ-services concept
- ➢ Customizing the intelligent cloud
- ➢ Roles played by edge μ-services
- ➢ Hardware accelerators for intelligence

Big data analytics to support IoT use cases

- ➢ The Apache ecosystem: Zookeeper, Hadoop, Pig, Hive, HBase, etc.
- ➢ Spark and its RDD model.

ORGANIZATIONAL STUFF

Spring 2019

ORGANIZATIONAL TOPICS/FAQ

Projects and pairing with digital agriculture students/scenarios.

Prelim

Wait -list for getting in

CS5999 (MENG PROJECT CREDITS)

Some people expand their CS5412 projects by adding 3 credits of CS5999, which allows them to count the project towards MEng project credits.

But this means *six hours more work per week, starting this week!*

Those projects are always more ambitious, harder to build, and we closely monitor to make sure that they extra hours really were reflected in extra accomplishments.

MEETING PEOPLE

© Can Stock Photo

We'll have a few "meet and match" events in the first weeks.

Many projects are done by people who only know each other in passing. But this is true in industry as well!

The best teams often bring people together with very different specializations because doing so gives you coverage of more aspects.

WHY ARE WE FOCUSED ON AZURE?

Microsoft is leading among companies with high quality, professional products for cloud + IoT, so this drives a focus on their platform.

If you love AWS for some reason, you can do similar things, but will often find yourself looking to see how Microsoft did it, then trying to copy them.

So we will emphasize Azure, used from Linux (not Windows).

WHY NOT AWS OR GOOGLE?

Actually, we don't mind if you prefer to use some other cloud, but we are trying to offer some concrete help, and we can't help with three cloud options all at once.

These days Azure is just using a standard Ubuntu Linux as its default and we can create prebuilt containers for you to run with the software we want you to use already installed.

You can still code in whatever language you prefer.

WHAT ABOUT PROGRAMMING LANGUAGES?

These days, C++ 17 is probably the first choice for IoT application development, among people who care about performance.

But you can work in any language you like, at a slight cost.

For our class, we won't tell you what language to use.

HOW TO LEARN AZURE?

As it turns out, Azure is “recipe-oriented”:

- There is a standard way to build scalable applications, and there is a form-fill style of programming to match this standard approach.
- You can create new μ-services, but it takes a bit more sophistication.
- Only the recommended styles of programming will give good outcomes.
- Experts can do anything, but nobody is an expert after six weeks!

Sounds, nasty, right?

VISUAL STUDIO 2017 AND VS CODE

We use cloud-oriented interactive development environments (IDEs).

For Azure, Visual Studio 2017 or a related editor called Visual Studio Code are best (VS Code is a good choice if you favor C++ in one of the Azure/Linux options, while Visual Studio is best for the Windows .NET)

In both editors, you can edit, compile, run, debug, profile, etc. But the key thing is that both have built-in templates for common patterns!

VS Code

EXAMPLES?

In Visual Studio 2017, you can just tell it to create a C# program that will:

- … accept RPC requests over the Internet (via Windows Comm. Framework, WCF, which can use a built-in message format, or SOAP/REST)
- … create a client program that can issue those RPC requests.
- … implement a function to run in the Azure IoT Edge Function Server, or the Azure Intelligent Edge for IoT.
- … etc (the list is very long)

Using these tools, they provide the template and you just fill in the blanks!

CAN I USE JAVA+ECLIPSE?

Yes, but it might be harder.

Microsoft and Amazon each favor their own IDEs, as noted (Cloud 9" is the preferred IDE option on Amazon AWS). Both already understand cloud programming.

With Eclipse you might have to find and install the relevant "templates" more or less one by one, and this takes more expertise. But if you prefer this approach you can find instructions easily on the web.