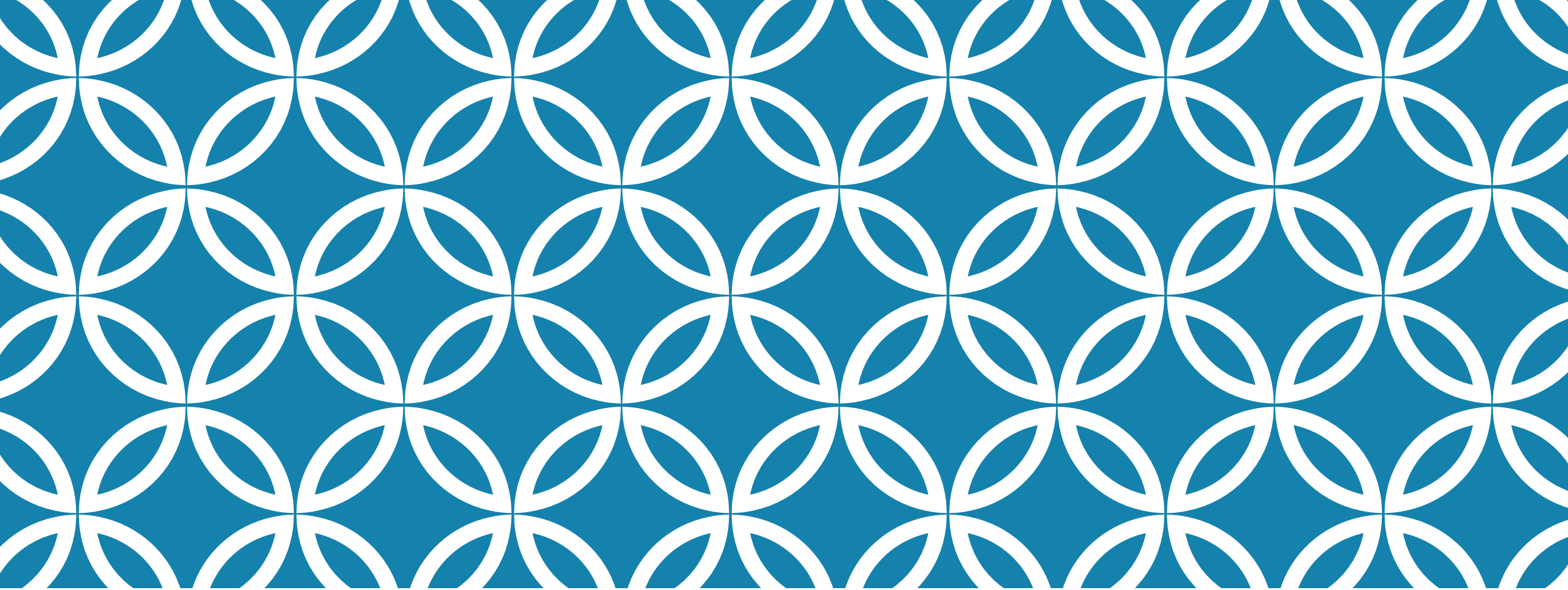


AVAILABILITY ZONES AND DATA REDUNDANCY SERVICES

Professor Ken Birman

CS4414/5416 Lecture 20

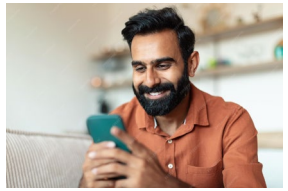
KEY IDEAS FOR TODAY

We know about clouds mapping to data centers at multiple regions

But what if a data center needs upgrades?

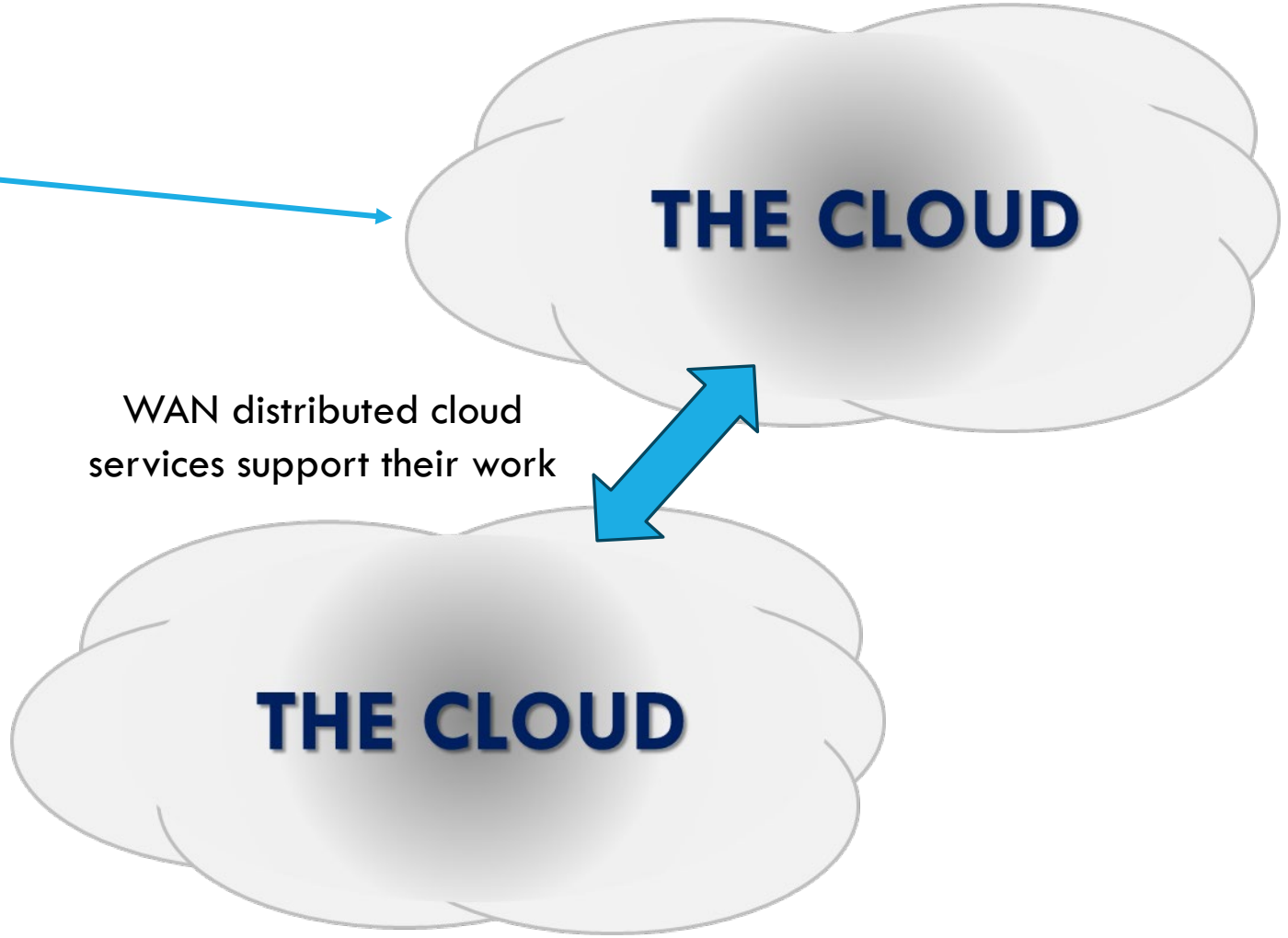
Our focus in today's lecture is on how we can replicate within a single region to enable scaling but also fault-tolerance at the scale of the entire data center!

A CLOUD WITH REGIONS



Rohan, based in Seattle, is collaborating with Anita, based in Singapore

WAN distributed cloud services support their work



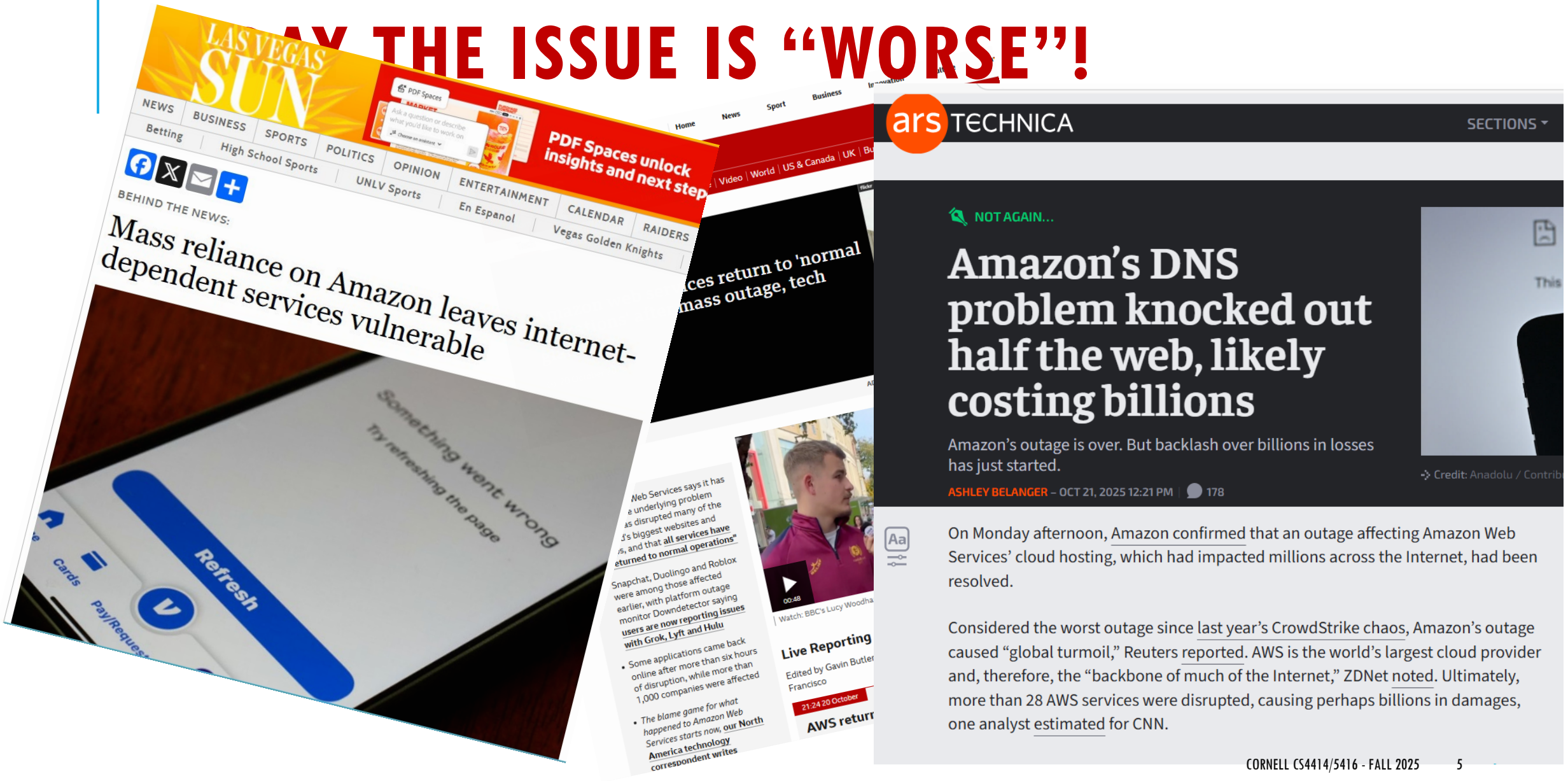
THE NEED FOR AVAILABILITY ZONES

Companies like Amazon and Microsoft faced a problem early in the cloud build-out: servicing a data center can require turning it off!

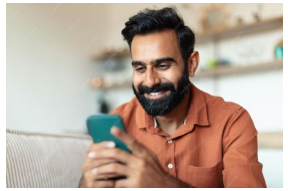
Why?

- Some hardware components are too critical to service while active, like the datacenter power and cooling systems, or the "spine" of routers.
- Some software components can't easily be upgraded while running, like the datacenter management system.
- In fact there is a very long list of cases like these

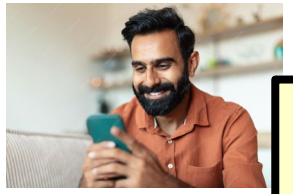
AY THE ISSUE IS "WORSE"!



A CLOUD WITH REGIONS AND AVAILABILITY ZONES



A CLOUD WITH REGIONS AND AVAILABILITY ZONES



Two or more data centers comprise an **availability zone.** Cloud AZ data replication tools automate many tasks for AZ-available services

AVAILABILITY ZONES

Basics: Instead of one mega-sized datacenter, they build two or three.

Often, all are active. In that mode they spread load over them, so everything stays busy.

But this also gives them an option for shutting one down entirely to do upgrades (and with three, they would still be able to "tolerate" a major equipment failure in one of the two others).

HOW CAN WE BUILD MICROSERVICES FOR THIS AZ MODEL?

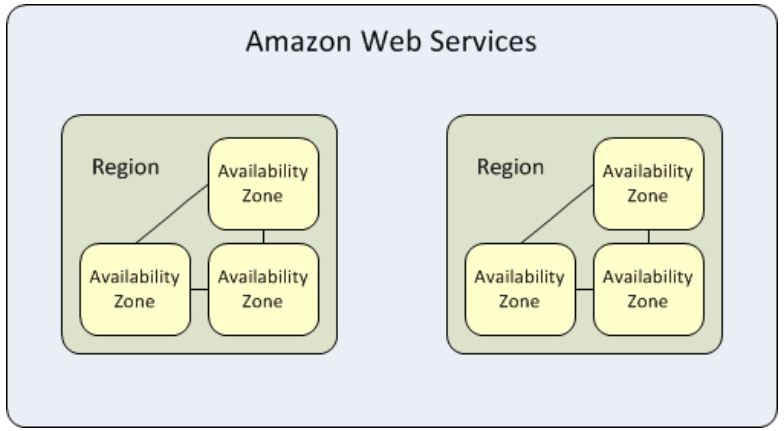
Normally, your microservice would just run in one of the datacenters, so the data it "learns" or "manages" for customers would normally be local to that datacenter.

We've talked about replication inside a single datacenter or compute cluster. Can we somehow replicate across availability zones? Then we could "fail over" if one data center needs to shut down but the others are up and running normally.

AVAILABILITY ZONES – AMAZON AWS

AWS Edge is less capable

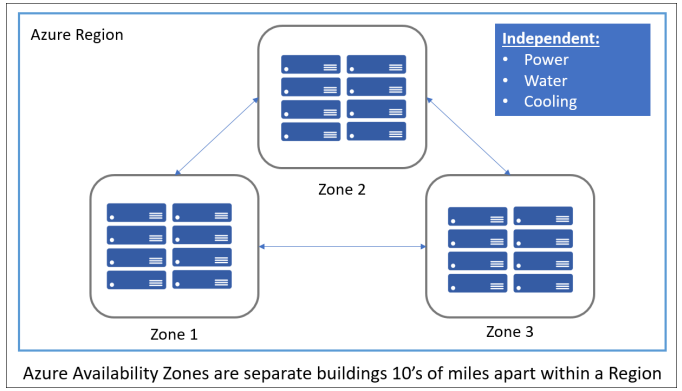
The AWS "region" has an availability zone structure



AVAILABILITY ZONES – MICROSOFT AZURE

Notice the blue circles.

Those are regions with three zones.



Azure Availability Zones are separate buildings 10's of miles apart within a Region

MODELS FOR HIGH-AVAILABILITY REPLICATION

Within a datacenter you can just make TCP connections and build your own chain replication solution, or download Derecho and configure it.

But communication between datacenters is tricky for several reasons. Those same approaches might not work well, or perhaps not at all.

CONNECTIVITY LIMITATIONS

Every datacenter has a firewall forming a very secure perimeter: applications cannot pass data through it without following the proper rules.

Zone-redundant services are provided by AWS and Azure and others to help you mirror data across zones, or even communicate from your service in Zone A to a "sibling" in Zone B.

Direct connections via TCP would probably be blocked: it is easy to connect into a datacenter but hard to connect *out* from inside!

WHY DO THEY RESTRICT OUTGOING TCP?

The modern datacenter network can have millions of IP addresses inside each single datacenter.

But these won't actually be unique IP addresses if you compare across different data centers: the addresses *only make sense within a zone*.

Thus a computer in datacenter A often would not have an IP address visible to a computer in datacenter B, blocking connectivity!

HOW DOES A BROWSER OVERCOME THIS?

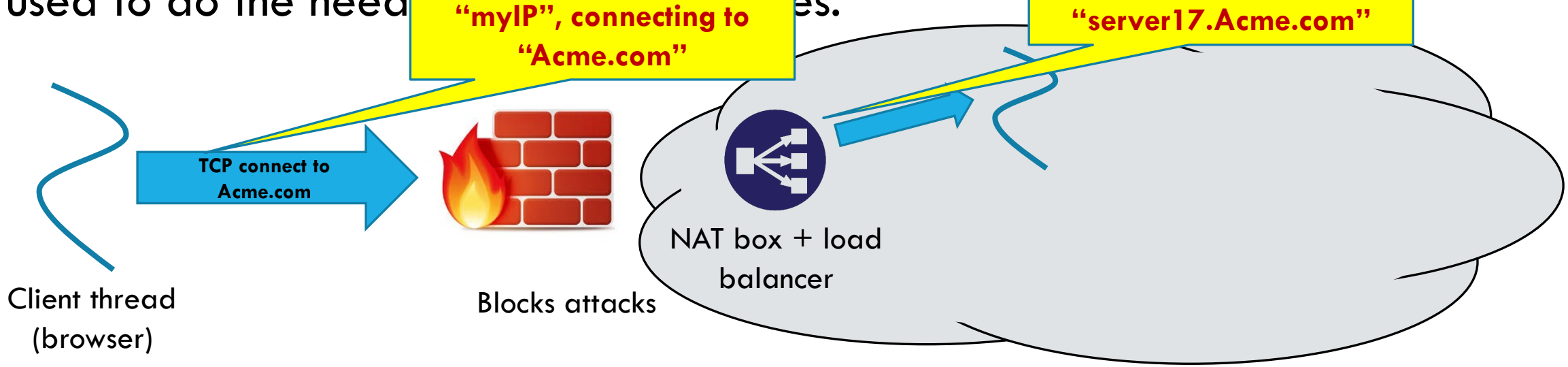
Your browser is on the *public* internet, not internal to a datacenter.

So it sends a TCP connect request to the datacenter over one of a small set of datacenter IP addresses covering the full datacenter.

- AWS, which hosts for many other sites, has a few IP addresses per site.
- Some systems mimic this approach, others have their own ways of ensuring that traffic to Acme.com gets to Acme's servers, even if hosted on their framework.

ARRIVING TRAFFIC: NAT BOX

On arrival, packets are scanned for possible BOT traffic or DDoS attacks, then the IP addresses are translated to "load balance" work over the proper servers. This would be costly, but high-spee re is used to do the need es.



HOW DOES IT PULL THIS TRICK OFF?

The NAT box maintains a table of “internal IP addresses (and port numbers) and the matching “external” ones.

As messages arrive, if they are TCP traffic, it does a table lookup and substitution, then adjusts the packet header to correct the checksum.

Thus “server17.Acme.com” cannot be accessed directly and yet your messages are routed to it, and vice versa.

BUT WITH GEOREPLICATION, THIS BLOCKS YOUR CODE FROM CONNECTING TO ITSELF

If you have machine A.Acme.com inside AWS or Azure, and then try to connect to B.Acme.com, it works inside a single datacenter.

- In fact you will be running in an "enterprice VLAN" or "VPC": what seems to be a private cloud.
- If you were to launch Ethereal or a similar sniffer you only see traffic from your own machines, not traffic from other datacenter tenants.

But if B was in a different datacenter, the connection simply won't work.

- Both A and B are behind NAT boxes, so neither can see the other!

OPTIONS?

Cloud vendors offer ways to make an A-B connection across datacenters in the same region (availability zone).

You need to use a special library they provide and otherwise, the connection would fail. And they charge for this service.

More common is to use some existing “Zone-aware” service

ZONE AWARE SERVICES ON AZURE

Linux Virtual Machines

Windows Virtual Machines

Virtual Machine Scale Sets

Managed Disks

Load Balancer

Public IP address

Zone-redundant storage

SQL Database

Event Hubs

Service Bus

VPN Gateway

ExpressRoute

Application Gateway

A FEW WORTH NOTING

Zone-aware storage is a storage mirroring option.

Files written in zone A would be available as read-only replicas in zone B. B sees them as a read-only file volume under path /Zone/A/…

This is a very common way to share information between data centers.

A FEW WORTH NOTING

The **Azure Service Bus** in its "Premium" configuration

- This is a message bus used by services within your VPC to communicate.
- Azure offers a configuration that automatically transfers data across zones under your control.
- Again, you need to follow their instructions to set it up. They charge but the performance and rate have historically favored this model.
- Basically, two queues hold messages, and then they use a set of side by side TCP connections to shuttle data in both directions. Very efficient!

A FEW WORTH NOTING

Azure's **zone-redundant virtual network gateway**

- Used when you really do want a connection of your own, via TCP
- Setup is fairly sophisticated but they walk you through it.
- In effect, creates a special "pseudo-public" IP address for your services, which can then connect to each other. Not visible to other external users
- But performance might be balky: this isn't their most performant option. And they charge by the megabyte transferred over the links.

GEOREPLICATION

This term specifically refers to replication at global scale.

Availability zone aware programs are mostly said to be "mirrored" for high availability.

Georeplication arises in applications where a company like Google needs to offer services to clients who work globally.

HEAVY TAILED LATENCY

A big concern with georeplication is erratic delays.

Within an availability zone, the issue is minimal: the networks are short (maybe blocks, maybe a few miles), so latencies are tiny.

But with global WAN links, latencies can be huge and variation grows.

- Mean delay from New York to London: 90ms
- Mean delay from New York to Tokyo: 103ms

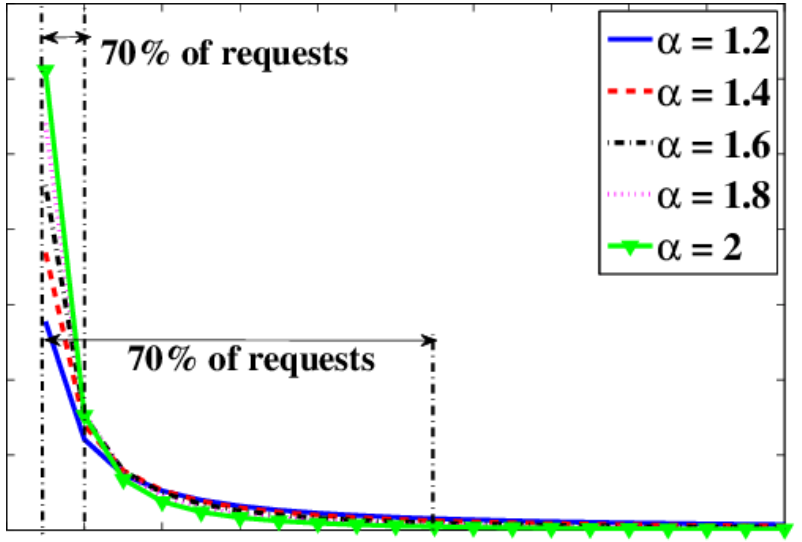
HEAVY TAILED LATENCY

Studies reveal "Zipf" latency distributions

Most traffic gets through very quickly

But some traffic takes extremely long. We say that these distributions are heavy-tailed if the "very slow" traffic adds up to a substantial portion of the total traffic.

The number of events in the i'th popularity bin is proportional to $1/i^{\alpha}$



Zipf's law

THE GOAL: TRANSACTIONS AT GLOBAL SCALE

If a team is collaborating on a project, but team members work at globally distributed locations, how can they avoid overwriting each-other's edits?

This argues for a way to check in edits to files and other data using a transactional approach.

ISSUE THIS RAISES

In protocols like Paxos, would we want to wait for *all* sites to respond, or just a quorum? Seemingly, a quorum is far better.

On the other hand, perhaps we would want all sites within an availability zone, but then wouldn't need to wait for geo-replicas to respond?

Leads to a three-level concept of Paxos stability.

- Locally stable in datacenter… Availability-zone stable… WAN stable

WHAT HAVE EXPERIMENTS SHOWN?

Basically, Paxos performs poorly with high, erratic latencies.

A big issue is that delay isn't symmetric:

- The path from Zone A to Zone B might be slow
- Yet the path from Zone B to Zone A could be fast at that same instant.

So the outgoing proposals experience one set of delays, and replies from Paxos members experience different delays. You end up waiting "for everyone"

GOOGLE SPANNER IDEA: TRUETIME

Google uses actual time as a way to build an asynchronous totally ordered data replication solution called Spanner.

Google **TrueTime** is a global time service that uses atomic clocks together with GPS-based clocks to issue timestamps with the following guarantee:

- For any two timestamps, T and T'
- If T' was fully generated before generation of T started,
- Then T > T'

WHY SUCH A TORTURED DEFINITION?

With the guarantee they offer, if some operation B was timestamped T, and could possibly have seen the effects of operation A with timestamp T', then we can order B and A so that A occurs before B.

The full API is very elegant and simple: TT.now(), TT.before(), TT.after(). TT.now() returns a range from TT.before() to TT.after().

The basic guarantee is that the true time is definitely in the range TT.now().

TT.before() is definitely a past time, and TT.after() is definitely in the future.

IMPLEMENTING TRUETIME

Google starts with a basic observation:

- Suppose clocks are synchronized to precision δ
- It is 10:00.000 on my clock, and someone wants to run transaction X.
- What is the largest possible timestamp any zone could have issued?

My clock could be slow, and some other clock could be fast.

So the largest (worst case) possible will be T+2 δ

MINIMIZING δ IN TRUETIME

Google uses a mix of atomic clocks and GPS synchronization for this. They synchronize against the mix every 30s, then might drift in between.

GPS can be corrected for various factors, including Einstein's relativistic time dilation, and they take those steps.

In the end their value of δ is quite small (0-6ms). So at actual time 10am, transaction X might get a TT.after() timestamp like **(10:00.006, zone-id, uid).**

- The zone id and uid are to break ties.

SPANNER'S USE OF TRUE TIME

Spanner is a transactional database system (in fact using a key-value structure, but that won't matter today).

Any application worldwide can generate a timestamped Spanner transaction.

These are relayed over the Google version message services. Their service ***delivers messages from Zone X to Zone Y in timestamp order.***

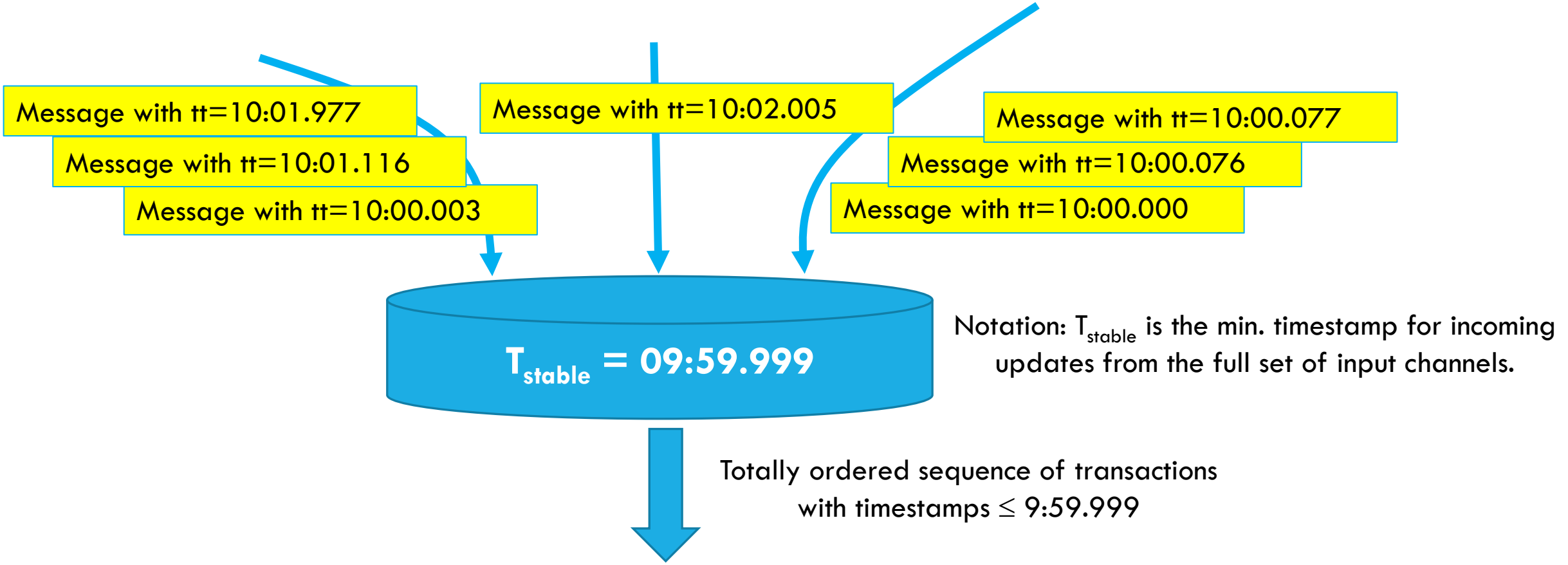
LIFE OF A SPANNER INSTANCE

Spanner has connections to perhaps hundreds of remote zones.

Transactions flow in on these connections, and it stores them until every zone has sent some transaction with a timestamp of value T_{stable} or larger.

Then it can apply all transactions with timestamps $\leq T_{stable}$, in order.

LIFE OF A SPANNER INSTANCE



A NICE FEATURE OF SPANNER

Think back to how CASD took worst-case assumptions and then created a **slow** protocol: It always delays until the worst-case delay has passed.

But Spanner can safely assume that the links to all its data centers are working, and it just waits to hear from all of them. If a data center is taken offline, Spanner is told, so then it won't wait for that link.

Thus Spanner can make ordering decisions "as soon as possible".

WHY DOES THIS WORK?

If Spanner has received messages with timestamps $> T_{stable}$ from every zone, *it has seen every transaction with* timestamp $\leq T_{stable}$!

- This is because the connections deliver messages in order, by timestamp.
- If an earlier transaction were to show up, it would violate this rule.

So, if it now places those transactions into timestamp order (breaking ties by (zone-id, uid)), they can be applied to the global database in total order.

WHAT LIMITS SPANNER?

One limiting factor is that although the zone-to-zone data transfers run over large numbers of parallel TCP connections, the messages need to be put into order to obtain this "monotonicity" property.

A second limiting factor is Zipf-like latency with heavy tails: Spanner will often have to wait for "laggards".

At global scale the effect can be significant (many seconds).

PAXOS ALL OVER AGAIN?

With the classic implementation of Paxos, we have a back-and-forth interaction *that traverses every link several times in both directions. Paxos experiences delays repeatedly.*

With Google Spanner, there are global asynchronous flows but no back-and-forth coordination of this kind. *So Spanner experiences delays once.*

Intriguing observation: Derecho's version of Paxos "behaves" like Spanner.

WHAT LIMITS SPANNER?

Google researchers report that the most frustrating issue is that a transaction cannot even be processed *locally* without waiting this way!

In some situations, a speculative result may be acceptable: "If there are no conflicts, my transaction would run and you would win the auction!"

But in other settings, consistency is absolutely needed, so there is no choice.

TRICKS TO MINIMIZE IMPACT

If a zone hasn't been sending any transactions, it should "pause".

- Send a special "End of sequence" message
- Cease to send new transactions
- Other zones no longer need to wait.

Later, to resume, it will have to get permission to restart:

- Send a "resume request"
- Every other zone must acknowledge this before new transactions can start.

OTHER OPTIONS?

With a "primary zone" model, we can shard our database and assign each shard a primary owner (only the owner zone can update that shard).

Then you can make the rule that the primary owner can always do transactions on shards that it owns, without waiting.

But if *any* transaction would need to access shards for which it isn't primary, than *all* must use the Spanner ordering mechanism (you can't mix methods).

WHAT LIMITS THE PRIMARY ZONE METHOD?

In some situations it is hard to know what shards a transaction would access before the transaction code actually executes.

For example, the keys a transaction will touch might be a function of the data it reads in some initial step. So until it executes, we don't know the key set, and can't know if those will all be local shards.

Spanner is really aimed at cases like these.

BEYOND ZONE AWARENESS

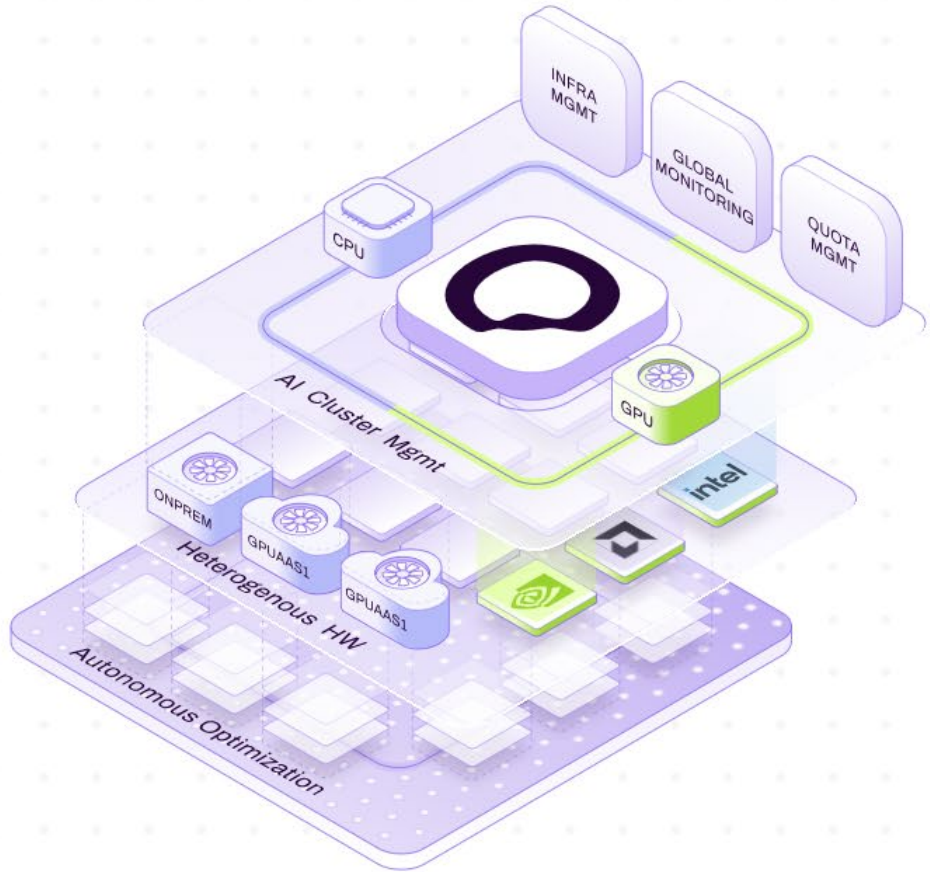
In CS4414/5416 we've learned about

- Fault tolerance (high availability) for individual services: use replication to make duplicates, then "fail over" if a new view from Zookeeper, Derecho, etc reports a disruption
- Today: AZ redundancy: Same idea, multiple data centers
- Today: Georeplication: Same idea, global redundancy
- And beyond our course? Exostellar!

Self-managed, Heterogeneous AI Infrastructure Orchestration

Exostellar does the thinking - scaling GPUs and CPUs, tuning workloads, and boosting ROI so AI developers and IT teams don't have to.

See how it works ↗



A CORNELL SPINOFF!

Hakim Weatherspoon

Robbert van Renesse
Zhiming Shen, Hakim

Exostellar offers a transparent way to

- Migrate virtual clouds from one data center to another (use case? Chasing the best "spot pricing")
- Move "primary zone" to follow the sun (use case? Your customers are working in Bangalore but sleeping in Seattle…)
- Migrate from one vendor to another (use case? The next massive outage like the AWS one)

MORE REMARKS ABOUT IP ADDRESSES

We have seen that with modern cloud systems, you could connect to Acme.com and be routed to Amazon.com instead.

In fact the modern cloud has considerable control over this, and you can influence that layer.

This was actually the layer of AWS that failed on October 20 and caused such widespread disruption... But something always fails, eventually. A wise developer has a plan for "all forms of outages"

HOW CAN WE PROTECT AGAINST ALL FORMS OF OUTAGES?

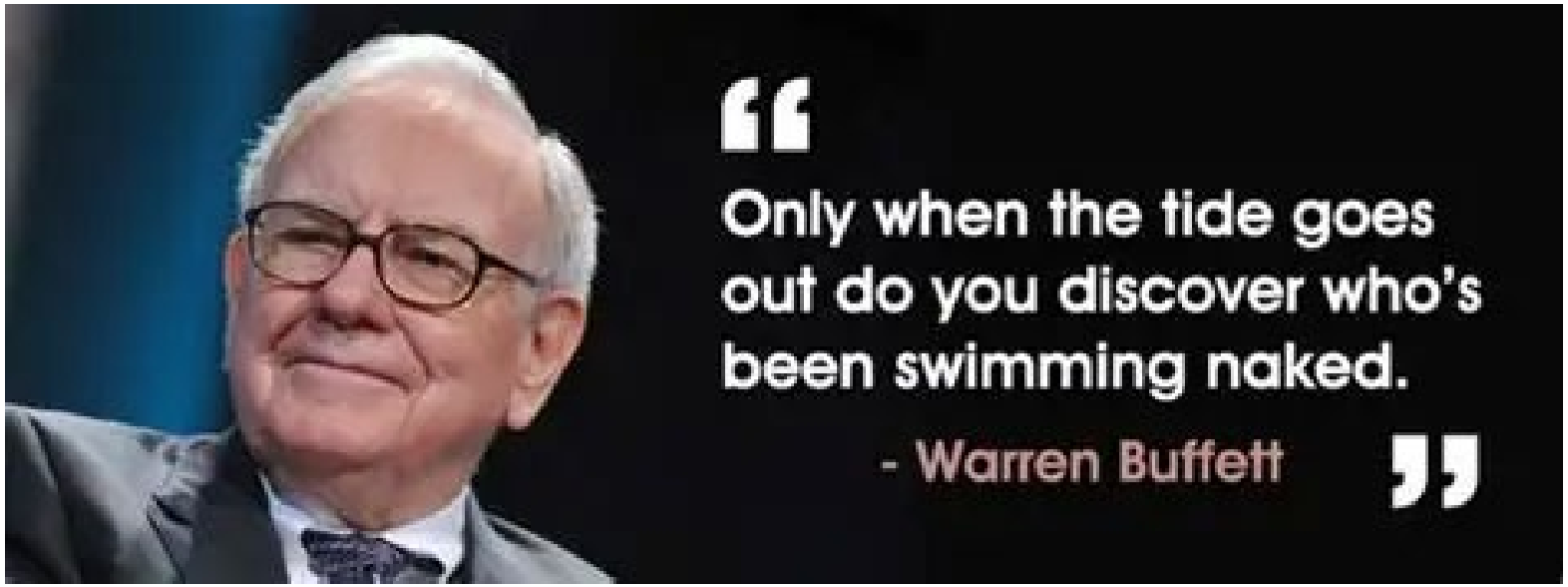
We saw the answer in the lecture on virtual synchrony, but it isn't specific to Derecho and Zookeeper

View membership as a *virtualized concept*.

The healthy portion of a system *manages itself* – and excludes elements that act strangely without trying to diagnose faults.

SUMMARY

Warren Buffett (the investor) put it really well:



SUMMARY

"...he doth bestride the narrow world/Like a Colossus..."

Replication (redundancy) can save your app when that unlucky day comes!

- **Availability Zone:** Just neighboring data centers. With some cost, you can use TCP and build "normal" protocols. Derecho should work this way.
- **Georeplication:** Using georeplication techniques like Google Spanner a system can have a robust point of presence in many physical regions… But don't cut corners: Researchers have experimented with Paxos at global scope, but it performs poorly due to high-latency links. **Use vendor-supported tools.**
- **Multi-vendor replication:** Here, your company is redundant across two or more cloud providers, such as AWS but also Azure (Exostellar, or similar). Now if something disrupts one cloud your application hops to the other and remains available!

SUMMARY

A well-designed web app

- Replicates data within an AZ, across regions and maybe even across providers
- The client side of the code automatically rolls over: if the primary point of contact is unresponsive, try the next option
- Very much like what we saw with Facebook's TAO… but **you need to implement this** when you build the app.

SELF-TEST ON WIDE AREA REPLICATION

We know all about fast replication with tools like Paxos or Derecho/Vortex.

Couldn't we just use these within an availability zone? If two clouds are right next to one-another, we get ultra-low latency.

If the answer is yes, what are the pros and cons of the AZ form of mirroring? If no, why not?

SELF-TEST

Many block chains use super-expensive wide area consensus protocols that tolerate Byzantine attacks.

But some blockchains are operated by enterprises that aren't expecting to see Byzantine attackers, ever.

Why not just use Spanner plus standard entanglement via encryption to build much faster WAN blockchains?

SELF-TEST

Consider fault-tolerance.

Be prepared (or prepare to be embarrassed)!

List the techniques we discussed. Then come up with a concrete example of a use case (one per technique) for each of them.

Make your examples as practical and real as you can. Not vague handwaving but a concrete application that would benefit

SELF-TEST

Spanner incurs a delay penalty to **serialize concurrent transactions** (create a single agreed ordering) initiated anywhere on earth.

A primary-owner model assigns each piece of data a primary location, then mirrors it for read-only access elsewhere. This approach doesn't need Spanner's globally-unique update ordering.

List two applications for which the primary-owner suffices. Now list two for which you really do need Spanner's properties.