Availability

Scalability of Web Systems – Fall 2017

Björn Þór Jónsson

Definition

$$A = \frac{E[\text{uptime}]}{E[\text{uptime}] + E[\text{downtime}]}$$

If we define the status function $X(t)$ as

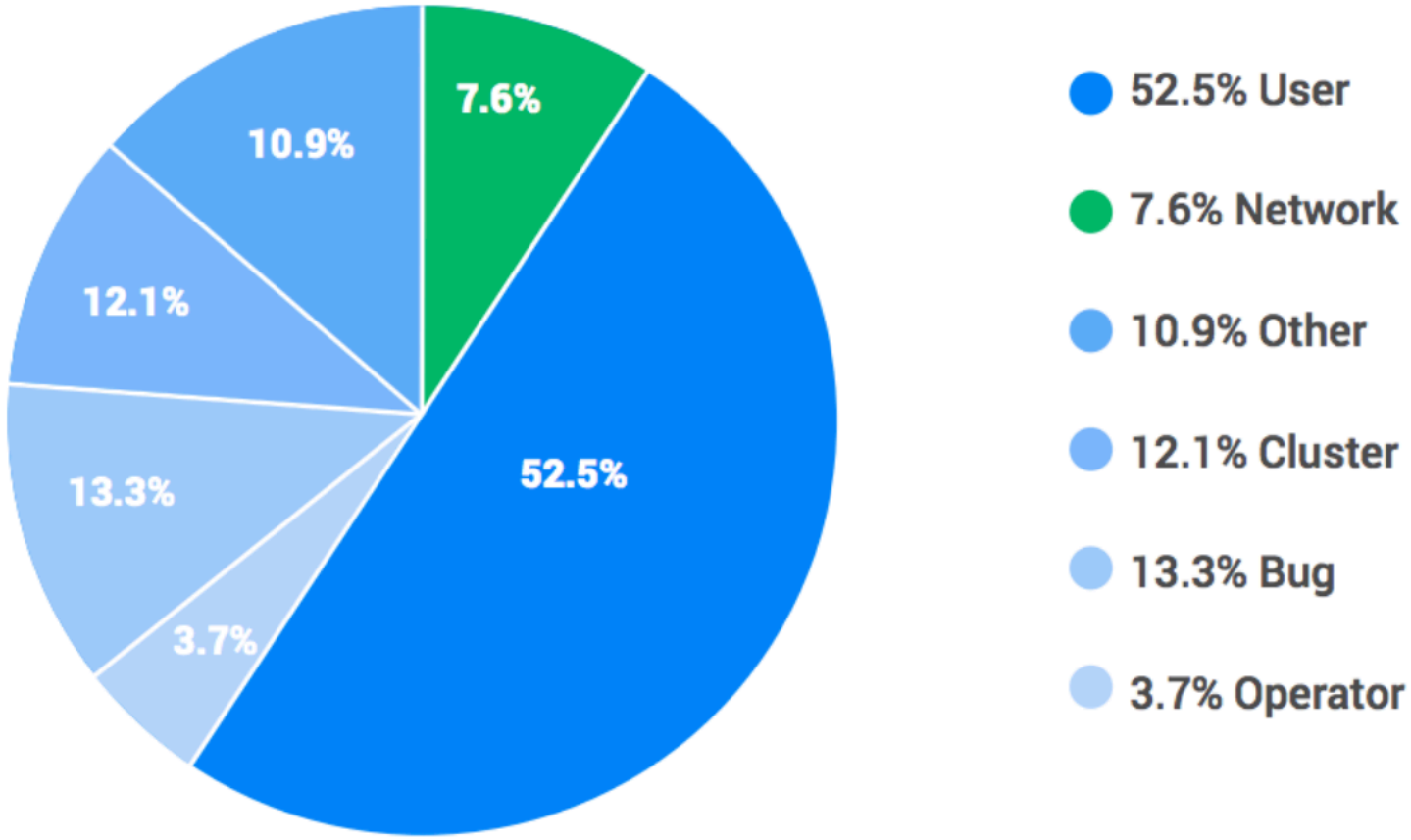
$$X(t) = \begin{cases} 1, & \text{sys functions at time } t \\ 0, & \text{otherwise} \end{cases}$$

therefore, the availability $A(t)$ at time $t > 0$ is represented by

$$A(t) = \Pr[X(t) = 1] = E[X(t)].$$

https://en.wikipedia.org/wiki/Availability

Failure Reasons



https://research.google.com/pubs/pub45855.html

Failure to Follow Best Practices in…

- Monitoring of the relevant components
- Requirements and procurement
- Operations
- Avoidance of network failures
- Avoidance of internal application failures
- Avoidance of external services that fail
- Physical environment
- Network redundancy
- Technical solution of backup
- Process solution of backup
- Physical location
- Infrastructure redundancy
- Storage architecture redundancy

https://en.wikipedia.org/wiki/High_availability

Costs of Unavailability

- Costs of lost business
- Cost of lost reputation
- Cost of penalties
- Cost of lost worker hours
- Recovery costs

Availability %	Downtime per year	Downtime per month	Downtime per week	Downtime per day
90% ("one nine")	**36.5 days**	**72 hours**	**16.8 hours**	**2.4 hours**
95% ("one and a half nines")	18.25 days	36 hours	8.4 hours	1.2 hours
97%	10.96 days	21.6 hours	5.04 hours	43.2 minutes
98%	7.30 days	14.4 hours	3.36 hours	28.8 minutes
99% ("two nines")	**3.65 days**	**7.20 hours**	**1.68 hours**	**14.4 minutes**
99.5% ("two and a half nines")	1.83 days	3.60 hours	50.4 minutes	7.2 minutes
99.8%	17.52 hours	86.23 minutes	20.16 minutes	2.88 minutes
99.9% ("three nines")	**8.76 hours**	**43.8 minutes**	**10.1 minutes**	**1.44 minutes**
99.95% ("three and a half nines")	4.38 hours	21.56 minutes	5.04 minutes	43.2 seconds
99.99% ("four nines")	**52.56 minutes**	**4.38 minutes**	**1.01 minutes**	**8.64 seconds**
99.995% ("four and a half nines")	26.28 minutes	2.16 minutes	30.24 seconds	4.32 seconds
99.999% ("five nines")	**5.26 minutes**	**25.9 seconds**	**6.05 seconds**	**864.3 milliseconds**
99.9999% ("six nines")	**31.5 seconds**	**2.59 seconds**	**604.8 milliseconds**	**86.4 milliseconds**
99.99999% ("seven nines")	**3.15 seconds**	**262.97 milliseconds**	**60.48 milliseconds**	**8.64 milliseconds**
99.999999% ("eight nines")	**315.569 milliseconds**	**26.297 milliseconds**	**6.048 milliseconds**	**0.864 milliseconds**
99.9999999% ("nine nines")	**31.5569 milliseconds**	**2.6297 milliseconds**	**0.6048 milliseconds**	**0.0864 milliseconds**

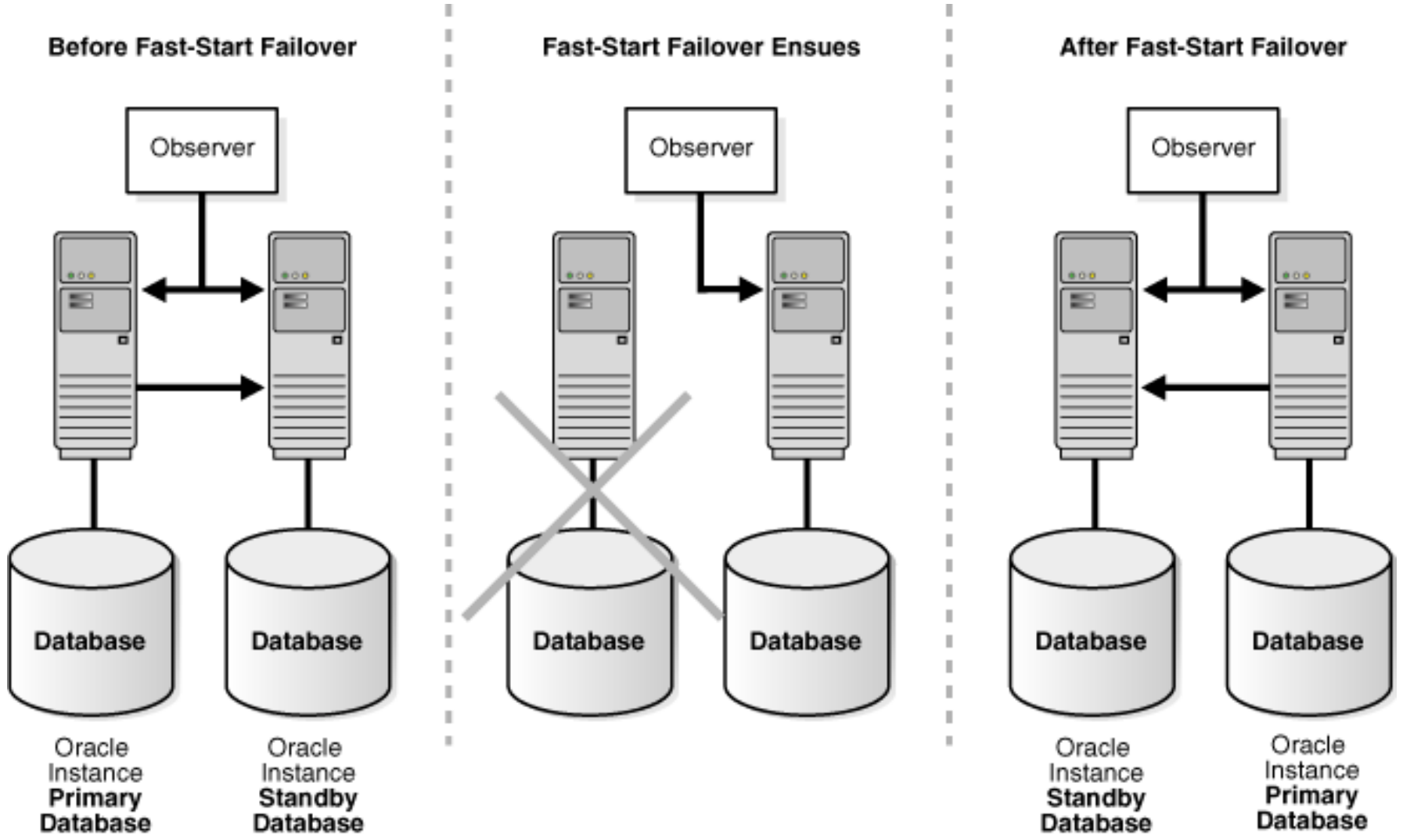
https://en.wikipedia.org/wiki/High_availability

System Design Principles for High Availability

- **Elimination of single points of failure.** This means adding redundancy to the system so that failure of a component does not mean failure of the entire system.
- **Reliable crossover.** In redundant systems, the crossover point itself tends to become a single point of failure. Reliable systems must provide for reliable crossover.
- **Detection of failures as they occur.** If the two principles above are observed, then a user may never see a failure. But the maintenance activity must.

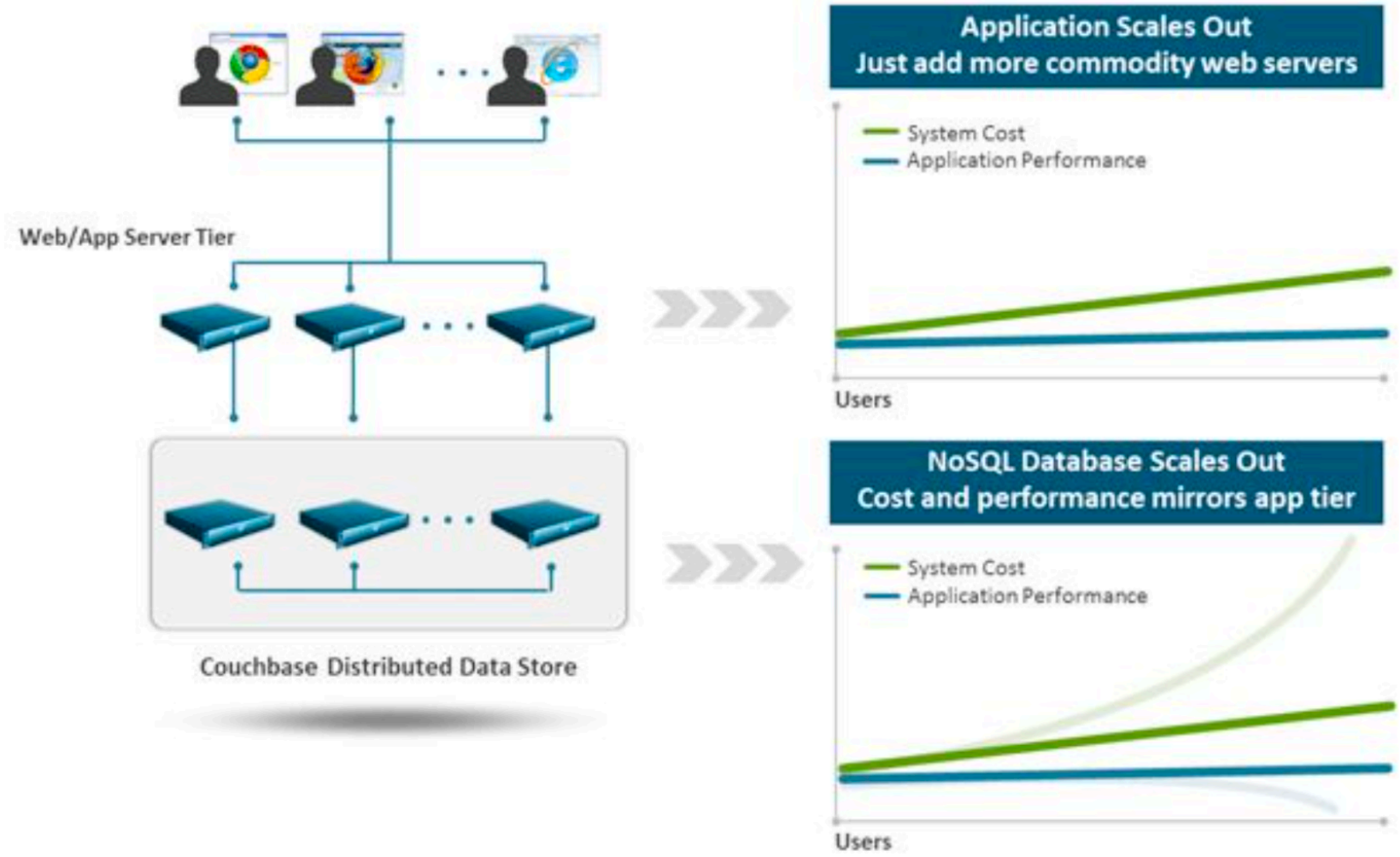
https://en.wikipedia.org/wiki/High_availability

Failover



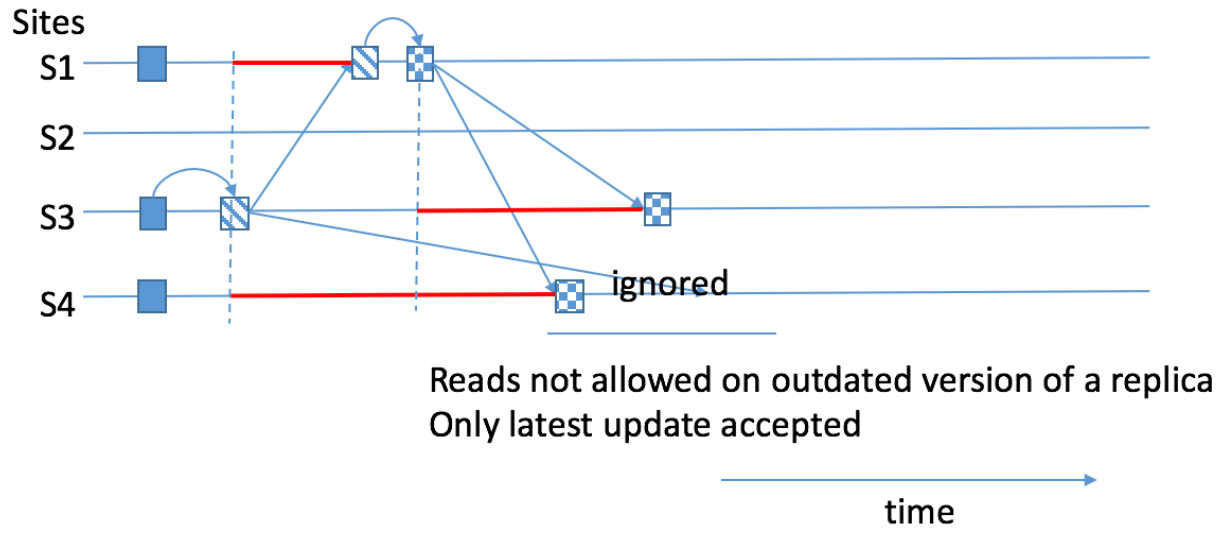
Failover

Redundancy


Web/App Server Tier
Couchbase Distributed Data Store
Application Scales Out
Just add more commodity web servers
System Cost
Application Performance
Users
NoSQL Database Scales Out
Cost and performance mirrors app tier
System Cost
Application Performance
Users

CAP Theorem

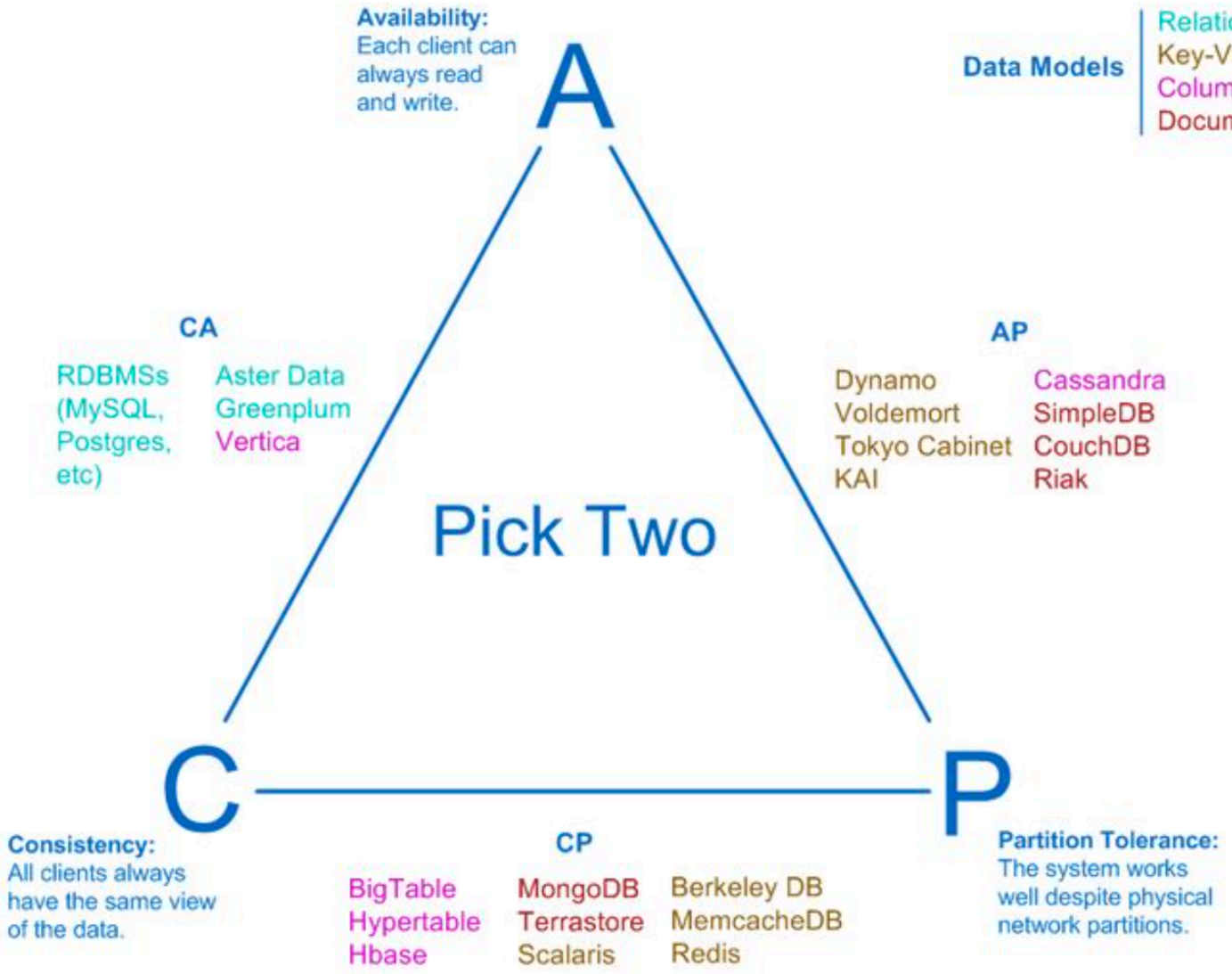
- **C** = Atomic **C**onsistency
  - Readers read most recent update (PAXOS)
- **A** = High **A**vailability
  - An answer is always returned
- **P** = Ability to Tolerate Network **P**artitions
  - The network becomes disconnected



Typical Formulation

- You can only get **two** of
  Consistency
  Availability and
  Partition Tolerance

Visual Guide to NoSQL Systems



Easily Provable Formulation

- In a **Partitioned** network, choose between **Availability** and **Consistency**

- Proof: Simple thought experiment

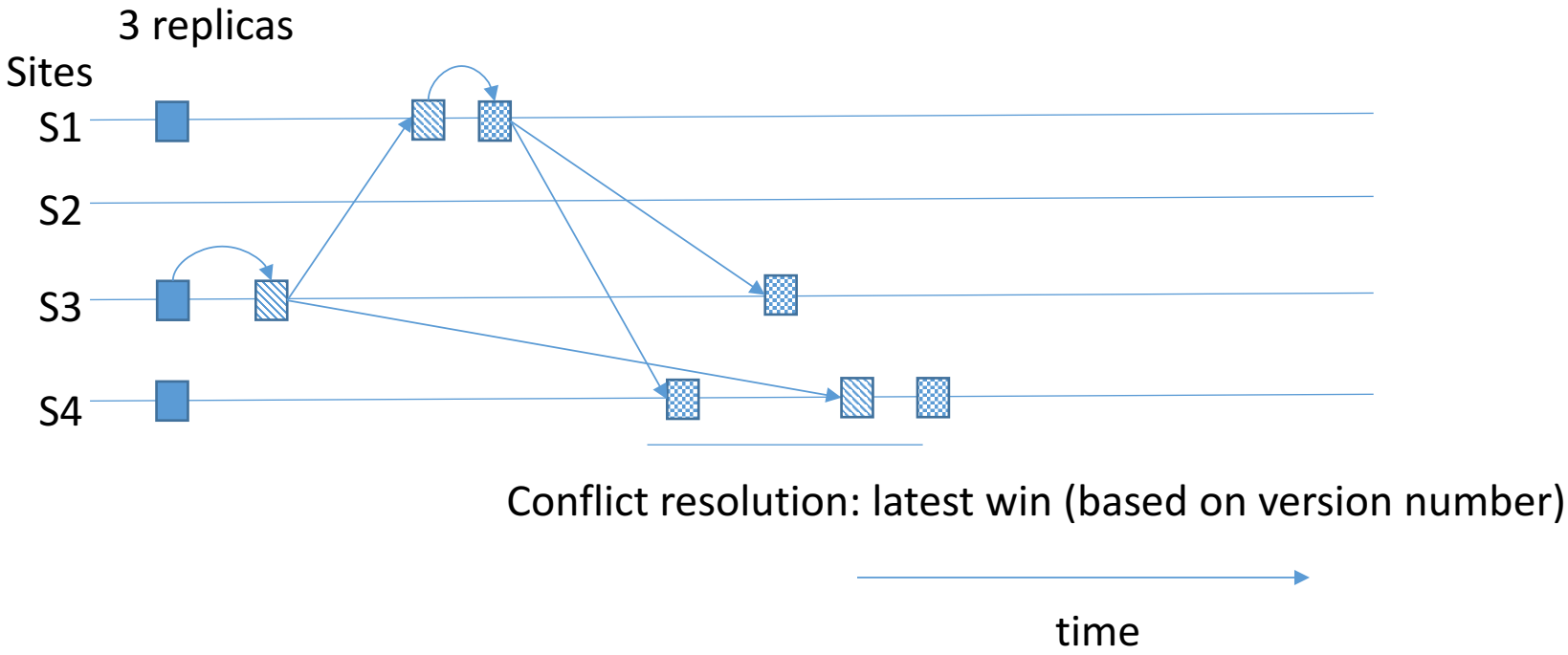
Gilbert and Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News, Volume 33 Issue 2 (2002)

Now What?

- Given the CAP theorem, what can we do?

- Eventual Consistency
- PACELC (Abadi, 2012)
- Harvest and Yield (Fox and Brewer, 1999)
- Spanner (Google, 2012) and CockroachDB

Replica Consistency – Eventual Consistency



Useful Formulation = PACELC

- In a **partitioned** network, choose between **Availability** and **Consistency**
- **Else** (regular operation), choose between **Latency** and **Consistency**

Daniel Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story", IEEE Computer 45(2): 37-42 (2012)

Tunable Consistency

- Not a binary decision
  - N replicas, R reads, W writes
- R = W = 1 gives eventual consistency
- R + W > N (+Paxos) gives strong consistency
  - Reduced likelihood of non-availability
- Can choose values between…

Harvest and Yield

- **Yield:** The probability of completing a request. How many 9's of availability can your system provide?
- **Harvest:** How complete is the answer that you are providing to your user?

- Define your systems in terms of how you can degrade harvest or yield to fit it's usage and your customer's needs
  - Graceful degradation…
  - Strategy 1: Probabilistic degradation
    - Example: Inktomi
  - Strategy 2: Application Decomposition and Orthogonal Mechanisms
    - Microservices!
    - Example: Farmville

http://mauricio.github.io/pwl-harvest-yield/#/

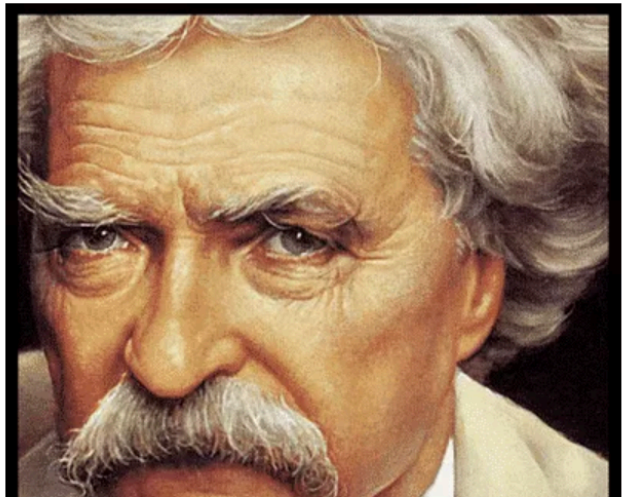
Network Partitions Will Happen!

Mark Twain

Quote: "The only two certainties in life are death and taxes."

This, seems to be more of a misattribution problem than that of misquotation. Mark Twain wasn't really the one who had coined this phrase, it was actually coined by Edward Ward in his 1724 'Dancing Devils' where he wrote 'Death and Taxes, they are certain.'

Then then there was Christopher Bullock had written in his 'Cobbler of Preston' (1716), ''Tis impossible to be sure of anything but death and taxes!'



Or Will They?

Amazon Web Services' secret weapon: Its custom-made hardware and network

BY DAN RICHMAN on January 19, 2017 at 10:49 am

2 Comments | Share 15 | Tweet | Share 2.5k | Reddit | Email

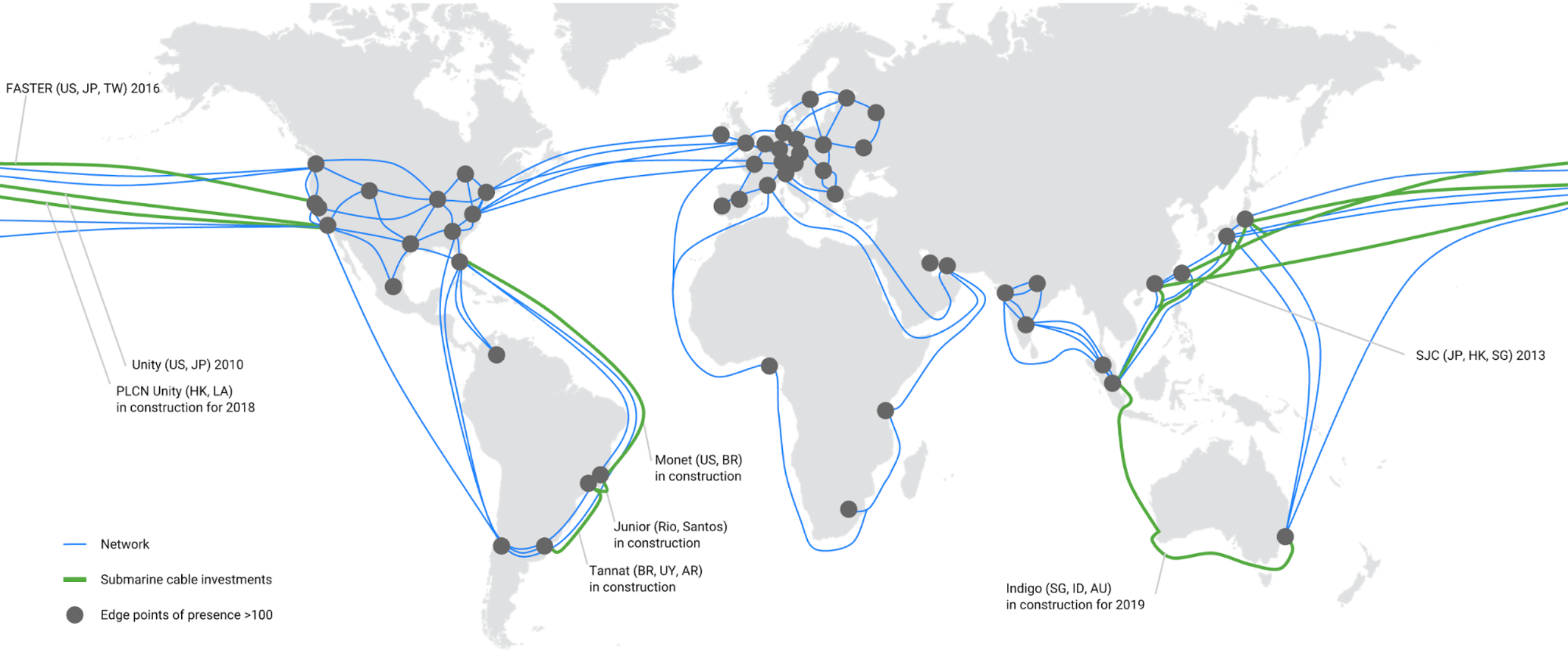
James Hamilton, an AWS VP and Distinguished Engineer, revs up for his talk at re:Invent 2016 in Las Vegas. (GeekWire Photo / Dan Richman)

"We run our own custom-made routers, made to our specifications, and we have our own protocol-development team," Hamilton said. "It was cost that caused us to head down our own path, and though there's a big cost (improvement) . . . the biggest gain is in reliability." This custom-made gear "has one requirement, from us, and we show judgment and keep it simple. As fun as it would be to have a lot of tricky features, we just don't do it, because we want it to be reliable."

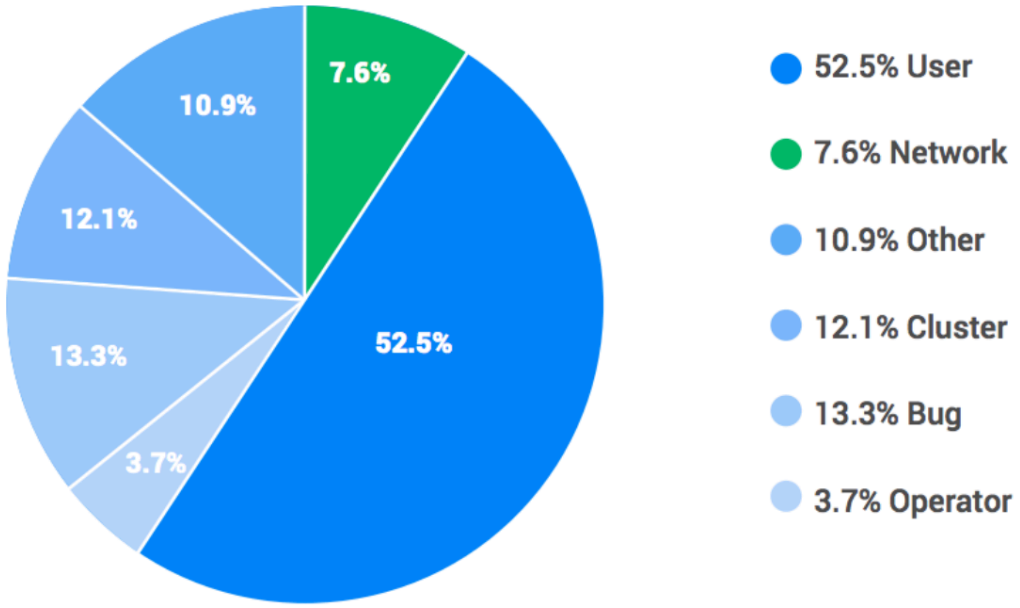
https://www.geekwire.com/2017/amazon-web-services-secret-weapon-custom-made-hardware-network/

Google Cloud Submarine Cable Investments

Google Cloud's well-provisioned global network is comprised of hundreds of thousands of miles of fiber optic cable and seven submarine cable investments



Are Network Partitions a Problem?



The Network category, under 8%, is where partitions and networking configuration problems appear. There were no events in which a large set of clusters were partitioned from another large set of clusters. Nor was a Spanner quorum ever on the minority side of a partition. We did see individual data centers or regions get cut off from the rest of the network. We also had some misconfigurations that underprovisioned bandwidth temporarily, and we saw some temporary periods of bad latency related to hardware failures. We saw one issue in which one direction of traffic failed, causing a weird partition that had to be resolved by bringing down some nodes. So far, no large outages were due to networking incidents.

https://research.google.com/pubs/pub45855.html

Google’s Cloud Spanner

https://cloud.google.com/spanner/

No-Compromise Relational Database Service

Cloud Spanner is the world’s first fully managed relational database service to offer both strong consistency and horizontal scalability for mission-critical online transaction processing (OLTP) applications. With Cloud Spanner you enjoy all the traditional benefits of a relational database; but unlike any other relational database service, Cloud Spanner scales horizontally to hundreds or thousands of servers to handle the biggest transactional workloads.

Comparison

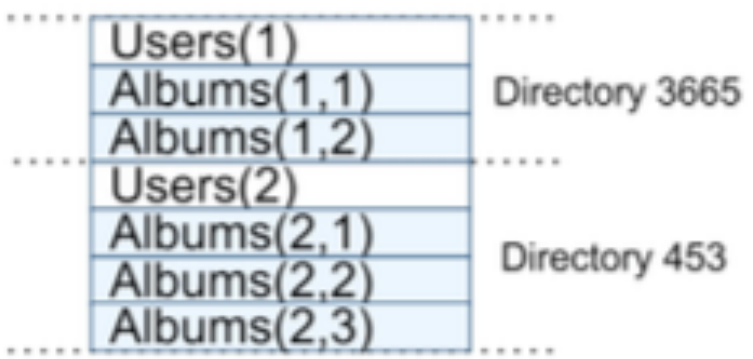
	CLOUD SPANNER	TRADITIONAL RELATIONAL	TRADITIONAL NON-RELATIONAL
Schema	✓ **Yes**	✓ Yes	✗ No
SQL	✓ **Yes**	✓ Yes	✗ No
Consistency	✓ **Strong**	✓ Strong	✗ Eventual
Availability	✓ **High**	✗ Failover	✓ High
Scalability	✓ **Horizontal**	✗ Vertical	✓ Horizontal
Replication	✓ **Automatic**	↻ Configurable	↻ Configurable

Data Model

- Nearly relational
  - "NewSQL"
- Key-Value store
  - PRIMARY KEY **required**
  - FOREIGN KEY not supported
- User-assisted sharding
  - INTERLEAVE
- SQL for queries
  - REST for updates

```
CREATE TABLE Users {
  uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
  uid INT64 NOT NULL, aid INT64 NOT NULL,
  name STRING
} PRIMARY KEY (uid, aid),
  INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```

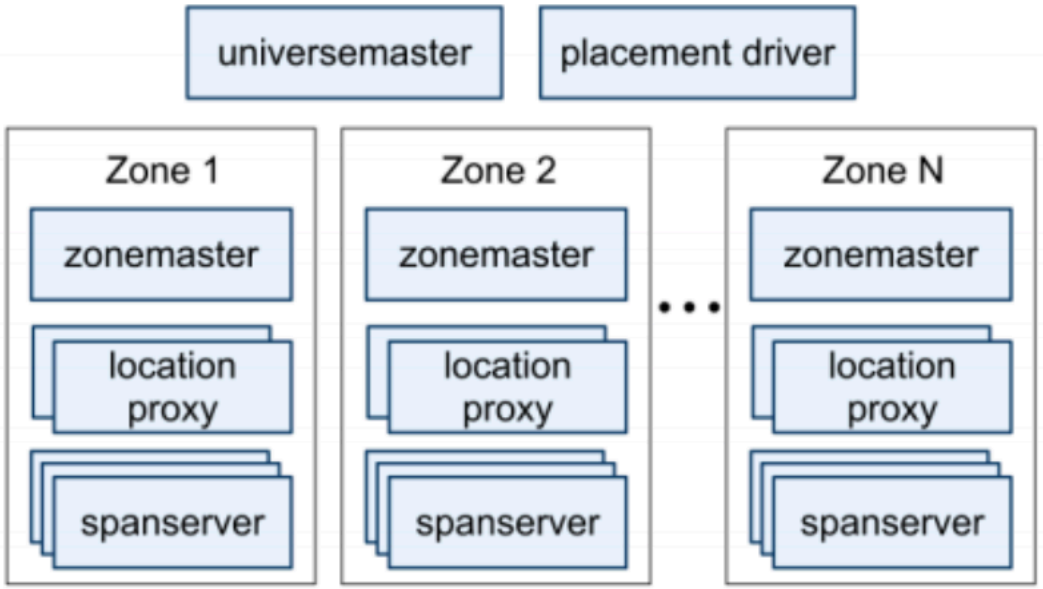


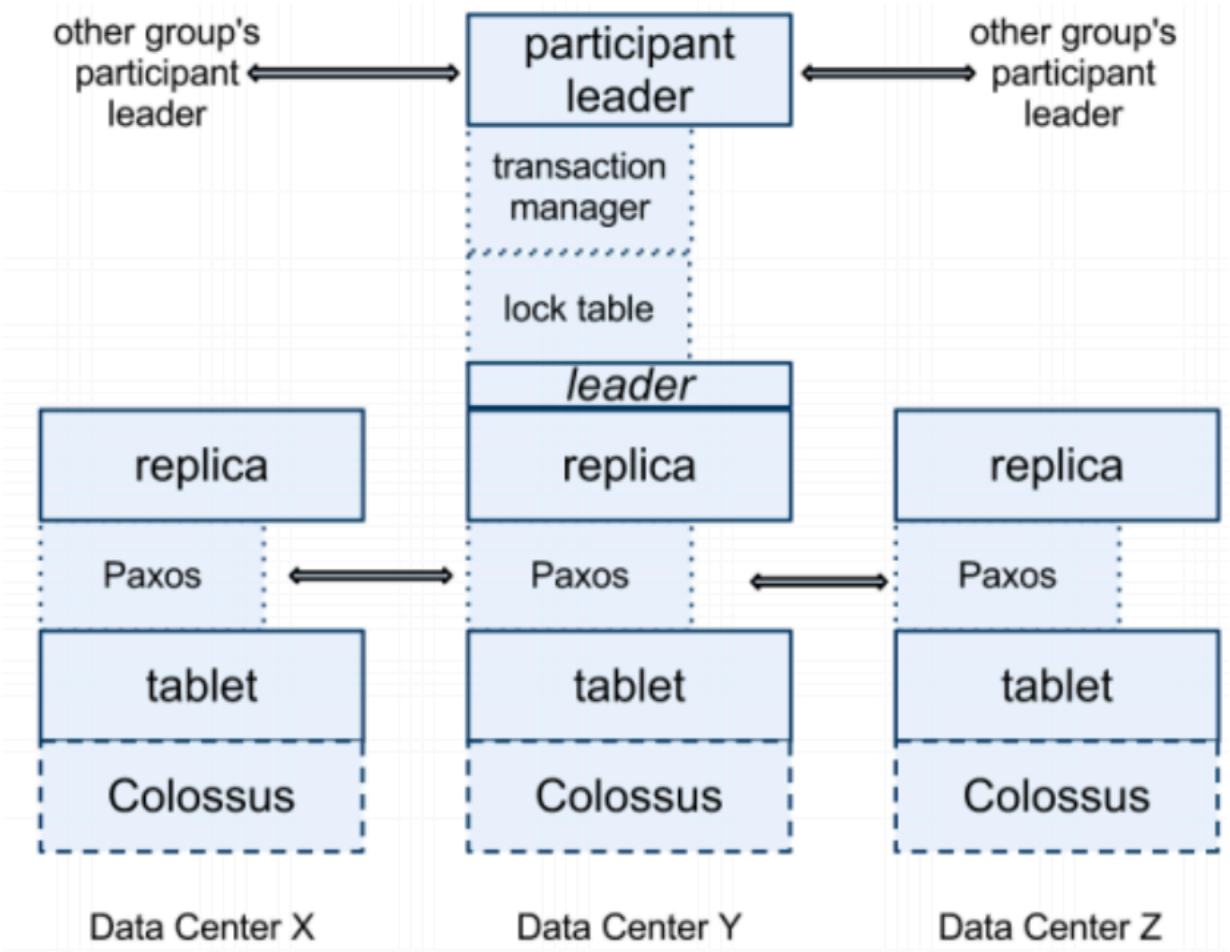
Performance by Structure

- Simple reads: 16K transactions/second, 25 ms/transaction
- Interleaved joins: 16K transactions/second, 25 ms/transaction
- Non-interleaved joins: 14 transactions/second, 14 sec/transaction
- N-i joins, with index: 14K transactions/second, 30 ms/transaction

https://opencredo.com/google-spanner-first-look/

Architecture Overview





Consistency

Method	Returns
$TT.now()$	$TTinterval$: $[earliest, latest]$
$TT.after(t)$	true if t has definitely passed
$TT.before(t)$	true if t has definitely not arrived

- Local consistency using locking and PAXOS
- External consistency from TrueTime.
  - Spanner's external consistency invariant is that for any two transactions, T_1 and T_2 (even if on opposite sides of the globe): if T_2 starts to commit after T_1 finishes committing, then the timestamp for T_2 is greater than the timestamp for T_1
- Spanner's use of TrueTime as the clock ensures the invariant holds. In particular, during a commit, the leader may have to wait until it is sure the commit time is in the past (based on the error bounds)

https://research.google.com/pubs/pub45855.html

TrueTime

Method	Returns
$TT.now()$	*TTinterval*: $[earliest, latest]$
$TT.after(t)$	true if t has definitely passed
$TT.before(t)$	true if t has definitely not arrived

TrueTime is implemented by a set of *time master* machines per datacenter and a *timeslave daemon* per machine. The majority of masters have GPS receivers with dedicated antennas; these masters are separated physically to reduce the effects of antenna failures, radio interference, and spoofing. The remaining masters (which we refer to as *Armageddon masters*) are equipped with atomic clocks. An atomic clock is not that expensive: the cost of an Armageddon master is of the same order as that of a GPS master. All masters' time references are regularly compared against each other. Each master also cross-checks the rate at which its reference advances time against its own local clock, and evicts itself if there is substantial divergence. Between synchronizations, Armageddon masters advertise a slowly increasing time uncertainty that is derived from conservatively applied worst-case clock drift. GPS masters advertise uncertainty that is typically close to zero.

http://static.googleusercontent.com/media/research.google.com/en/us/archive/spanner-osdi2012.pdf

Spanner claims to be consistent and available

Despite being a global distributed system, Spanner claims to be consistent and highly available, which implies there are no partitions and thus many are skeptical.[1] Does this mean that Spanner is a CA system as defined by CAP? The short answer is "no" technically, but "yes" in effect and its users can and do assume CA.

The purist answer is "no" because partitions can happen and in fact have happened at Google, and during (some) partitions, Spanner chooses C and forfeits A. It is technically a CP system. We explore the impact of partitions below.

Given that Spanner always provides consistency, the real question for a claim of CA is whether or not Spanner's serious users assume its availability. If its actual availability is so high that users can ignore outages, then Spanner can justify an "effectively CA" claim. This does not imply 100% availability (and Spanner does not and will not provide it), but rather something like 5 or more "9s" (1 failure in 10^5 or less). In turn, the real litmus test is whether or not users (that want their own service to be highly available) write the code to handle outage exceptions: if they haven't written that code, then they are assuming high availability. Based on a large number of internal users of Spanner, we know that they assume Spanner is highly available.

https://research.google.com/pubs/pub45855.html

Google Cloud Spanner: our first impressions

Pros

- Strongly ACID semantics
- Distributed, fault-tolerant system
- High performance, scales horizontally
- Fully hosted and managed solution that “just works”
- “Close enough” relational data model

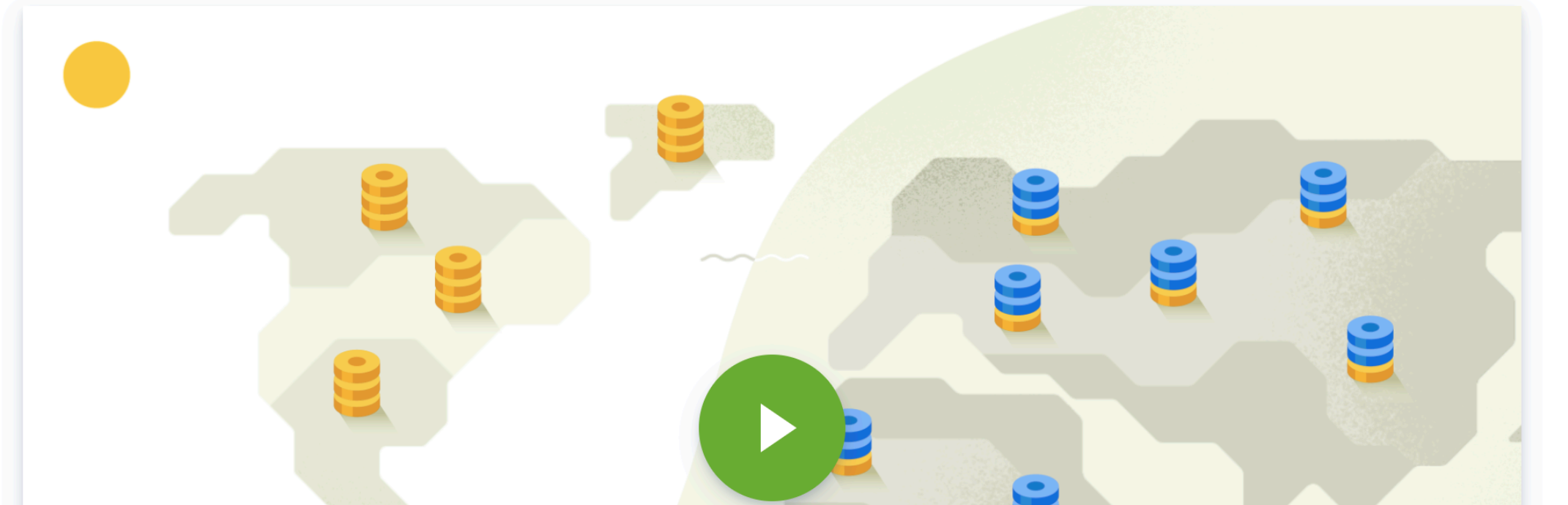
Cons

- Non-standard APIs
- Relative lack of tooling support
- Not “pure” RDBMS – limited ways to enforce referential integrity
- Data modeling has to be approached differently
- Vendor lock-in

https://opencredo.com/google-spanner-first-look/

CockroachDB

The SQL database for building global cloud services.
Watch the video to learn how it works:



Readings

- Fox, Brewer (1999). Harvest, Yield and Scalable Tolerant Systems. Workshop on Hot Topics in Operating Systems (HotOS).
- Gilbert, Lynch (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News, 33(2).
- Eric A. Brewer (2012). CAP twelve years later: How the "rules" have changed. IEEE Computer 45(2).
- Daniel Abadi (2012). Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. IEEE Computer 45(2).
- James C. Corbett et al. (2012). Spanner: Google's Globally-Distributed Database. Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI).

Take Away

- Redundancy used to facilitate availability
- CAP = tradeoff between **availability** and **consistency** when network is partitioned
- PACELC = tradeoff between **latency** and **consistency** otherwise

- Spanner = distributed CA system?
  - In theory: no
  - Effectively: yes
  - Due to Google's infrastructure…
- Many relational features – NewSQL
  - Many non-standard features also…
- CockroachDB may be a competitor