CAP Theorem
for Distributed Systems

Distributed and Cloud Computing

Jay Urbain, Ph.D.

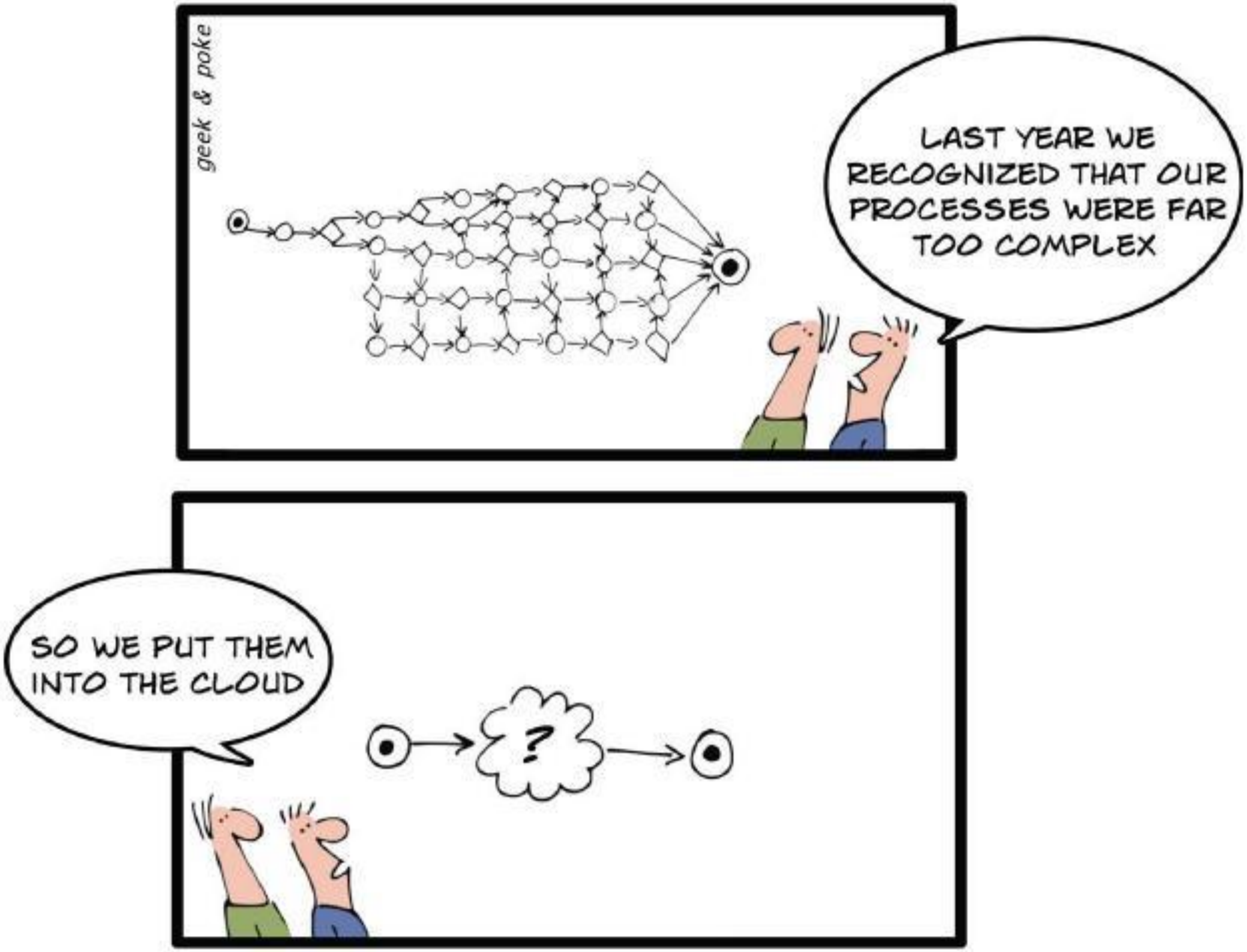
Credits:

- https://en.wikipedia.org/wiki/CAP_theorem
- Syed Sadat Nazrul, https://towardsdatascience.com/cap-theorem-and-distributed-database-management-systems-5c2be977950e

Willet

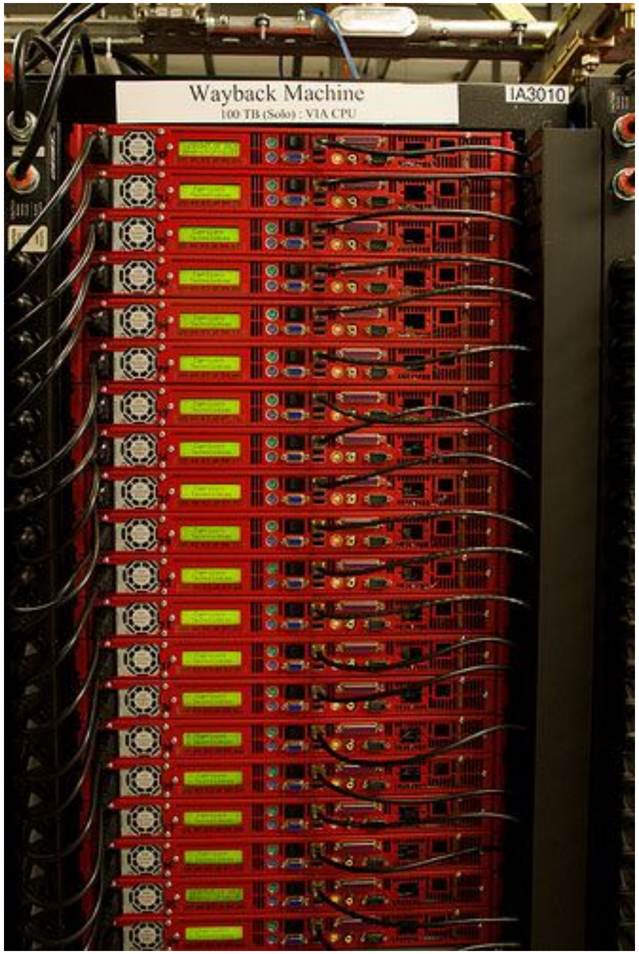
"Of course I'm insecure. I store vital computer information in a cloud!"

Graphic thanks to Migrating Your Applications and Processes to the Cloud: Practical Checklist


geek & poke
LAST YEAR WE RECOGNIZED THAT OUR PROCESSES WERE FAR TOO COMPLEX
SO WE PUT THEM INTO THE CLOUD
?

Topics

- Technologies for Scalable Distribution
  - **Cap Theorem**
  - Consistent Hashing
  - Vector Clocks
  - Sloppy Quorum
  - Merkle Hash Trees
- Dynamo: Amazon’s Highly Available Key-value Store



How much data is a lot?

640K ought to be enough for anybody.

ALBUQUERQUE N MEX
DEPT. OF CORRECT.
APD 105 519
12 13 77
WILLIAM H. GATES

ALBUQUERQUE N MEX
APD 105 519
12 13 77

How much data?

- The Google Search index contains hundreds of billions of web pages and is well over several petabytes in size.
- Google also indexes private data like email, photos, docs, etc. The private index has been growing rapidly along with the web index.
- Google also constructs knowledge resources like Knowledge Graph for enhancing search results.
  - https://www.google.com/search/howsearchworks

- Facebook's Social graph has ~2B monthly users. Each user can generate 100M Bytes of data per month. https://medium.com/@johnrobb/facebook-the-complete-social-graph-b58157ee6594

- As of December 2020, the Wayback Machine contained over 70 petabytes of data https://en.wikipedia.org/wiki/Wayback_Machine .

- CERN's LHC (Large Hadron Collider): 30 PB a year https://home.cern/resources/faqs/facts-and-figures-about-lhc#:~:text=The%20CERN%20Data%20Centre%20stores,are%20permanently%20archived%2C%20on%20tape.

**A petabyte (PB) is 10^{15} bytes of data, 1,000 terabytes (TB) or 1,000,000 gigabytes (GB).*

google.com/takeout

- Google offers an option to download all of the data it stores about you. A while back I requested to download it. It and the file is 12.7GB.

Consistency Models in Relational DBMS

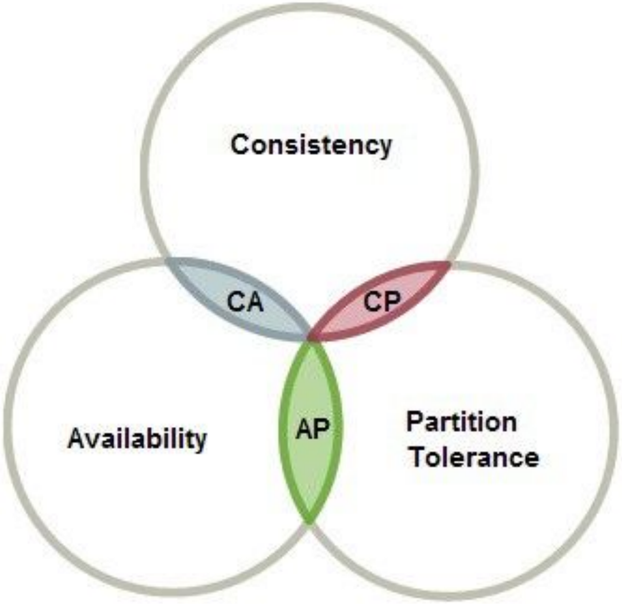
- RDBMS transactions are typically described as “ACID” transactions.
  - Atomic: The transaction is indivisible – either all the statements in the transaction are applied to the database, or none are.
  - Consistent: The database remains in a consistent state before and after transaction execution.
  - Isolated: While multiple transactions can be executed by one or more users simultaneously, one transaction should not see the effects of other concurrent transactions.
  - Durable: Once a transaction is committed, its changes are expected to persist.

Scalable Databases

- In the past, when we wanted to store more data or increase our processing power, the common option was to *scale vertically* (get more powerful machines) or further optimize the existing code base.
- With the advances in parallel processing and distributed systems, it is more common (more scalable and economic) to expand *horizontally*, or have more machines to do the same task in parallel.

Distributed Databases / Networked Systems

- In order to effectively pick the correct data solution, consideration of the CAP Theorem is necessary.
- **CAP Theorem is a concept that a distributed database system can only have 2 of the 3: Consistency, Availability and Partition Tolerance.**



CAP Theorem

Networked shared-data system

- Consistency
- Availability
- Partitions (i.e., across partitions)

Consistency

- A single up-to-date version of the data
- Inconsistency can arise:
  - Data is not updated properly (order is known)
  - Data is updated in multiple places (order is unknown)

Availability

- Data is always able to be updated and viewed.

Partitions

- Your data is split across multiple network-connected nodes that can have communication failures.

What’s CAP really saying?

- You cannot have a perfectly available consistent system if you also have partitions

Pretty idyllic scenario...

What’s CAP really saying?

- You cannot have a perfectly available consistent system if you also have partitions
    - “Get rid of partitions!”

What’s CAP really saying?

- You cannot have a perfectly available consistent system if you also have partitions
    - “Get rid of partitions!” -- No, need to scale.

What’s CAP really saying?

- You cannot have a perfectly available consistent system if you also have partitions
  - “Get rid of partitions!” -- No, need to scale.
  - “Get rid of availability!”

What’s CAP really saying?

- You cannot have a perfectly available consistent system if you also have partitions
    - “Get rid of partitions!” -- No, need to scale.
    - “Get rid of availability!” -- Yeah! ACID

What’s CAP really saying?

- You cannot have a perfectly available consistent system if you also have partitions
  - “Get rid of partitions!” -- No, need to scale.
  - “Get rid of availability!” -- Yeah! ACID
  - “Get rid of consistency!”

What’s CAP really saying?

- You cannot have a perfectly available consistent system if you also have partitions
    - “Get rid of partitions!” -- No, need to scale.
    - “Get rid of availability!” -- Yeah! ACID
    - “Get rid of consistency!” -- Yeah! BASE*

*BASE - Basically Available, Soft State, Eventually consistent.

BASE Model

BASE

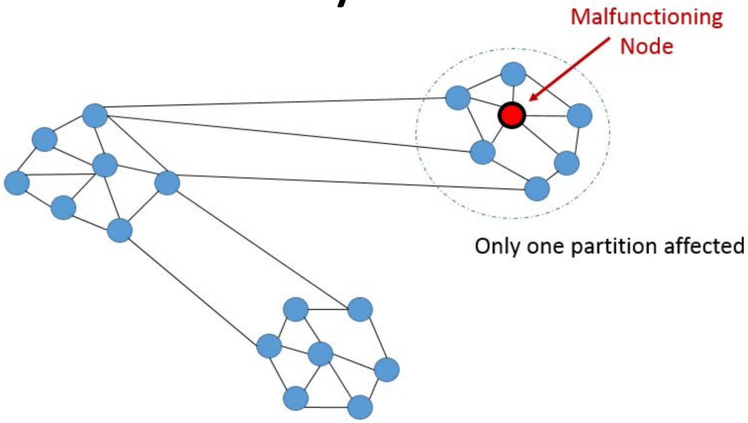
- **Basically Available** – Rather than enforcing immediate consistency, BASE-modelled NoSQL databases will ensure availability of data by spreading and replicating it across the nodes of the database cluster.

- **Soft State** – Due to the lack of immediate consistency, data values may change over time. The BASE model breaks off with the concept of a database which enforces its own consistency, delegating that responsibility to developers.

- **Eventually Consistent** – The fact that BASE does not enforce immediate consistency does not mean that it never achieves it. However, until it does, data reads are still possible (even though they might not reflect the reality).

Partition Tolerance

- The system continues to run, despite the number of messages being delayed by the network between nodes.
- A system that is partition-tolerant can sustain any amount of network failure that doesn't result in a failure of the entire network.
- Data records are sufficiently replicated across combinations of nodes and networks to keep the system up through intermittent outages.
- When dealing with modern distributed systems, Partition Tolerance is not an option. It's a necessity. Hence, we have to trade between Consistency and Availability.



High Consistency

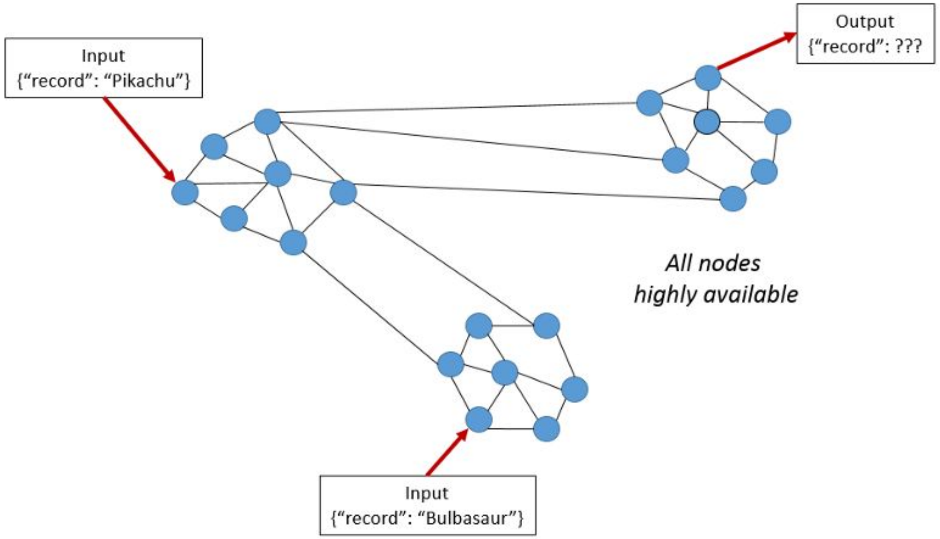
- All transactions see the same data at the same time.
- Performing a *read* operation will return the value of the most recent *write* operation causing all nodes to return the same data (consistency isolation level).
- A system has consistency if a transaction starts with the system in a consistent state, and ends with the system in a consistent state.
- In this model, a system can (and does) shift into an inconsistent state during a transaction, but the entire transaction gets rolled back if there is an error during any stage in the process.

High Availability

- Every request gets a response on success/failure.
- Achieving availability in a distributed system requires that the system remains operational 100% of the time.
- Every client gets a response, regardless of the state of any individual node in the system.

High Availability

- The availability metric is easy to measure: either you can submit read/write commands, or you cannot.
- Hence, the databases are time independent as the nodes need to be available online at all times.
- This means that we do not know if “Pikachu” or “Bulbasaur” was added first. The output could be either one.
- Hence why, high availability isn’t feasible when analyzing streaming data at high frequency.



CAP Theorem for Distributed Systems

- Distributed systems allow us to achieve a level of computing power and availability that were simply not available in the past.
- Our systems have higher performance, lower latency, and near 100% up-time in data centers that span the entire globe.
- Best of all, the systems of today are run on commodity hardware that is easily obtainable and configurable at affordable costs. **However, there is a price.**
- Distributed systems are more complex than their single-network counterparts.
- Understanding the complexity incurred in distributed systems, making the appropriate trade-offs for the task at hand (CAP), and selecting the right tool for the job is necessary with horizontal scaling.

Consistency Models in Non-relational DBMS

- One of the most significant differences between non-relational (*NoSQL*) databases and traditional RDBMS is the way in which *consistency* of data is handled.
- In a traditional RDBMS, *all users see a consistent view of the data.*
- Once a user commits a transaction, all subsequent queries will report *that transaction* and no-one will "see" partial results of a transaction. (*Note: dependent on the isolation level).*

Eventual Consistency

- A compromise between consistency and weak (no guarantees) consistency is *Eventual Consistency*.
- Eventual consistency: database may have some inconsistencies at a point in time, but it will eventually become consistent should all updates cease.
  - I.e., inconsistencies are transitory: eventually all nodes will receive the latest consistent updates.

Question

- Why not use higher level sync primitives?

Question

- Why not use higher level sync primitives?
    - This is the “A” in ACID

Question

- Why not use higher level sync primitives?
  - This is the “A” in ACID
  - Waiting on some kind of distributed lock violates perfect availability

Question

- Why not use higher level sync primitives?
  - This is the “A” in ACID
  - Waiting on some kind of distributed lock violates perfect availability
  - Attempting to coordinate indefinitely is choosing C over A

Why is CAP misleading?

Why is CAP misleading?

- Partitions are rare (so A and C should usually hold)
- Different subsystems of the same system can choose between various levels of A and C

What is a partition?

- Does it only happen when two or more nodes are split across a WAN?

What is a partition?

- Does it only happen when two or more nodes are split across a WAN?
  - No, even within a datacenter, nodes could become partitioned (but rare)
  - Client goes into offline mode (not rare)

Questions

What are the tradeoffs in a BASE system vs ACID system?

Questions

What are the tradeoffs in a BASE system vs ACID system?

- Consistency model.

Questions

What’s a good example of a ACID system?

What’s a good example of a BASE system?

Questions

What’s a good example of a ACID system?

- SQL databases like MySQL, SQLite, Microsoft SQL, Oracle follow ACID properties

What’s a good example of a BASE system?

- No SQL databases MongoDB, Cassandra, Redis follow BASE.

STOP

Remembrance Inc. – Ch. 1

Credits: Kaushik Sathupadi

http://ksat.me/a-plain-english-introduction-to-cap-theorem/

Remembrance Inc! - Never forget, even without remembering!

Ever felt bad that you forget so much? Don't worry. Help is just a phone away!

When you need to remember something, just call *555-55-REMEM* and tell us what you need to remember. For e.g., call us and let us know of your boss's phone number, and forget to remember it. when you need to know it back. Call back the same number[(555)-55-REMEM] and we'll tell you what's your boss's phone number.

Charges : only $0.10 per request

Ch. 2 : You scale up

Your venture gets funded by *YCombinator*. Your idea is so simple, needs nothing but a **paper notebook and phone**, yet it's so effective that it spreads like wildfire. You **start getting hundreds** of call every day.

You see **that more and more of your customers have to wait in the queue to speak to you**. Most of them even hang up, tired of the waiting. Besides when you were sick the other day and could not come to work you lost a whole day of business. Not to mention all those dissatisfied customers who wanted information on that day.

You decide it's time for you to scale your service and bring in your wife to help you.

Your start with a simple plan:

- You and your wife both get an extension phone
- Customers still dial (555)–55-REMEM and need to remember only one number
- A PBX will route the customers call to whoever is free

Ch. 3 : “Bad Service”

Two days after you implemented the new system, you get a call from your trusted customer John. This is how it goes:

John: Hey

You: Glad you called “Remembrance Inc!”. What can I do for you?

John: Can you tell me when is my flight to Boston?

You: Sure... 1 sec sir

(You look up your notebook, wow! there is no entry for “flight date” in John’s page)!!!!!

You: Sir, I think there is a mistake. You never told us about your flight to Boston

John: What! I just called you guys yesterday! (cuts the call!)

How did that happen? Could John be lying? You think about it for a second and the reason hits you! Could John’s call yesterday reached your wife? You go to your wife’s desk and check her notebook. Sure enough it’s in there. You tell this to your wife and she realizes the problem too.

What a terrible flaw in your distributed design! **Your distributed system is not consistent!** There could always be a chance that a customer updates something which goes to either you or your wife, and when the next call from the customer is routed to another person there will not be a consistent reply from Remembrance Inc!

Ch. 4: Fix the Consistency problem

Whenever any one of us get a call for an update, **before completing the call we tell the *other* person (transaction: consistency).**

This way **both of us note down any updates**.

When there is a call for search, we don't need to talk with the other person. Since both of us have the latest updated information in both of our notebooks we can just refer to it.

Problem: **"update" request has to involve both of us, and we cannot work in parallel during that time**. E.g. when you get an update request and telling me to update too, I cannot take other calls. But that's okay because **most calls we get anyway are "search"** (a customer updates once and asks many times) . Besides, we cannot give wrong information at any cost.

"Neat" your wife says, "but there is one more flaw in this system that you haven't thought of: **What if one of us doesn't report to work on a particular day**? On that day, then, we won't be able to take "any" update calls, because the other person cannot be updated!

We will have an Availability problem , I.e., if an update request comes to me I will never be able to complete that call because even though I have written the update in my note book, I can never update you. So I can never complete the call!"

Ch. 5: You come up with the greatest solution ever!

You begin to realize a little bit on **why a distributed system might not be as easy as you thought** at first. Is it that difficult to come up with a solution that could be both "**Consistent and Available**?"

- Whenever any one of us gets a call for an update before completing the call, if the other person is ***available*** we tell the other person. This way both of us note down any updates.
- But if the other person is not available (doesn't report to work) we send the other person an email about the update.
- The next day when the other person comes to work after taking a day off, **He first goes through all the emails**, updates his note book accordingly.. before taking his first call.

Genius! Your wife says. I can't find any flaws in this systems. Let's put it to use. *Remembrance Inc*! is now both ***consistent*** and ***available***!

Ch. 6: Your wife gets angry

Everything goes well for a while. Your system is ***consistent***. Your system works well even when one of you doesn't report to work.

But what if both of you report to work and one of you doesn't update the other person?

Remember all those days you've been waking your wife up early with your Greatest-idea-ever-bullshit?

What if your wife decides to take calls, but is too angry with you and decides not to update you for a day? Your idea totally breaks! Your idea so far is **good for *consistency* and *availability* but is not *Partition Tolerant!***

The big idea:

You can decide to be ***partition tolerant*** and ***consistent*** by deciding not to take any calls until you patch things up with your wife. Then your system will not be ***available*** during that time…

Or

You can decide to be ***partition tolerant*** and ***available*** to your customers, but ***not consistent*** with your wife's data.

Or

You can decide to be ***consistent*** and ***available*** to your customers, but ***not partition tolerant.***

Ch. 6: Conclusion

CAP Theorem:

When you are designing a distributed system you cannot achieve all three of Consistency, Availability and Partition tolerance. You can pick only two of:

- **Consistency**: Your customers, once they have updated information with you, will always get the most updated information when they call subsequently. No matter how quickly they call back.
- **Availability**: Remembrance Inc. will always be available for calls.
- **Partition Tolerance**: Remembrance Inc. will work even if there is a communication loss between you and your wife!

Epilogue: Eventual Consistency with a run around clerk

Eventual consistency:

- You can have a run around clerk, who will update other's notebook when one of you or your wife's note books is updated.
- The greatest benefit of this is that, the clerk can work in the background, and one of your or your wife's "update" doesn't have to block, waiting for the other one to update.
- This is how many **NoSql** systems work, one node updates itself locally and a background process synchronizes all other nodes accordingly.
- The only problem is that you will **loose consistency** for some time. E.g., a customer's call reaches your wife first and before the clerk has a chance to update your notebook , the customer calls back and it reaches you. Then he won't get a consistent reply.
- Not at all a bad idea if such cases are limited. E.g., assuming a customer won't forget things so quickly that he calls back in 5 minutes. Depends on your application!

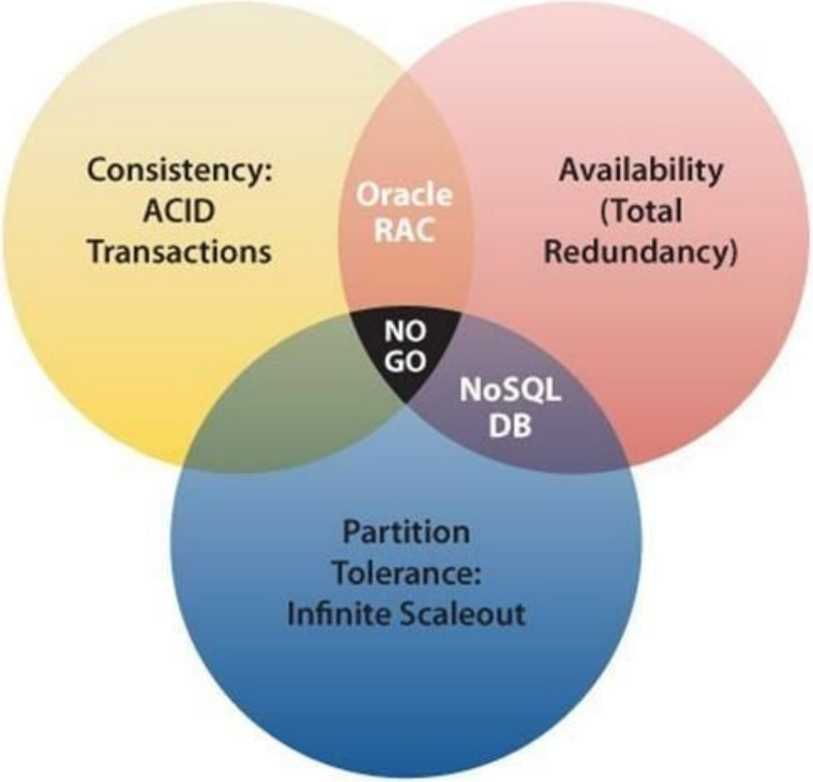
- *Note: MVCC – Multi-valued Concurrency Control*

CAP Theorem

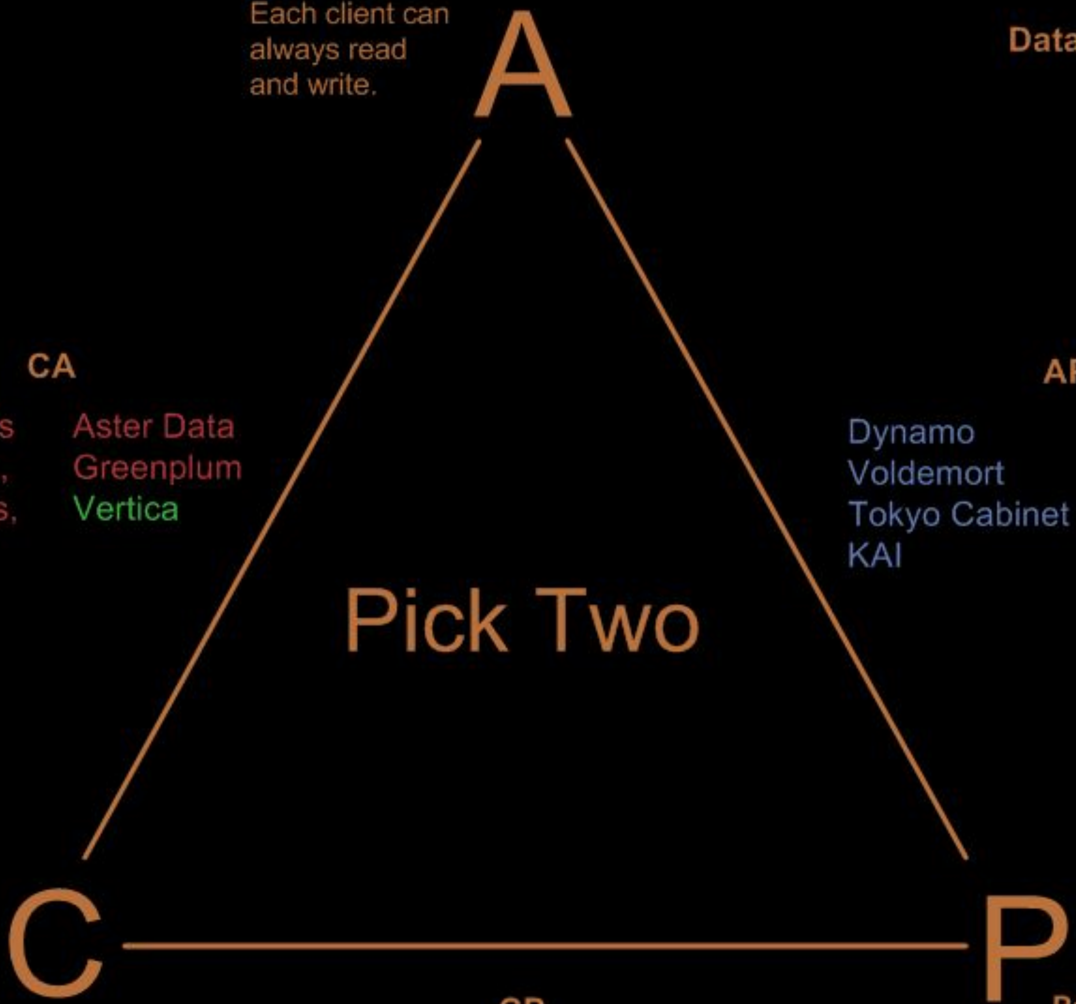
- In 2000, Eric Brewer outlined the CAP (Brewer's) Theorem.
- The CAP theorem states that in a distributed database system, you can only have at most two of the following three characteristics:
  - Consistency: All nodes in the (computing) cluster see exactly the same data at any point in time.
  - Availability: Failure of a node does not render the database inoperative.
  - Partition tolerance: Nodes can still function when communication with other groups of nodes is lost.

CAP Theorem

- Interpretation and implementations of CAP theorem vary, but most of the NoSQL database systems favor *partition tolerance and availability over strong consistency.*


Consistency: ACID Transactions
Oracle RAC
Availability (Total Redundancy)
NO GO
NoSQL DB
Partition Tolerance: Infinite Scaleout

Visual Guide to NoSQL Systems



Availability:
Each client can
always read
and write.

Data Models

Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented

CA

RDBMSs
(MySQL,
Postgres,
etc)

Aster Data
Greenplum
Vertica

AP

Dynamo
Voldemort
Tokyo Cabinet
KAI

Cassandra
SimpleDB
CouchDB
Riak

Consistency:
All clients always
have the same view
of the data.

CP

BigTable
Hypertable
Hbase

MongoDB
Terrastore
Scalaris

Berkeley DB
MemcacheDB
Redis

Partition Tolerance:
The system works
well despite physical
network partitions.