IEMS5730/IERG4330/ESTR4316
Spring 2024

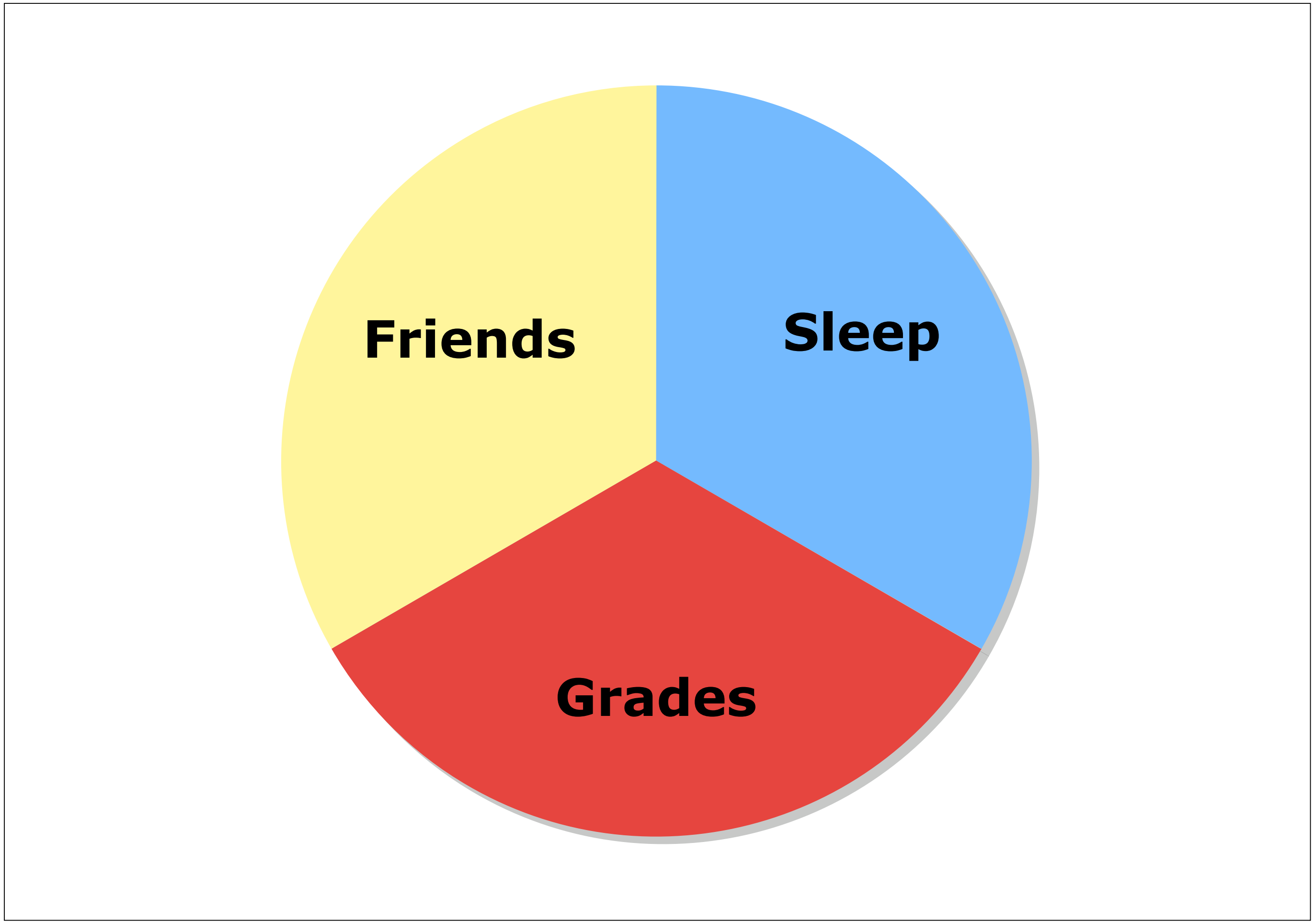
The CAP Theorem, ACID vs. BASE

Prof. Wing C. Lau
Department of Information Engineering
wclau@ie.cuhk.edu.hk

Acknowledgements

- The slides used in this chapter are adapted from the following sources:
  - CS5412 Cloud Computing, by Ken Birman, Cornell
  - CS498 Cloud Computing, by Roy Campbell and Reza Farivar, UIUC.
  - CS525 Advanced Distributed Systems, by Indranil Gupta, UIUC
  - Slides by Daniel J. Abadi, Yale University
  - Perry Hoekstra, Jiaheng Lu, Avinash Lakshman, Prashant Malik, and Jimmy Lin, "NoSQL and Big Data Processing, BigTable, Hbase, Cassandra, Hive and Pig"
-

The MIT Theorem



Eric Brewer's Conjecture

- In a famous 2000 keynote talk at ACM PODC, Eric Brewer (Berkeley) proposed that "you can have just two of the *Consistency*, *Availability* and *Partition Tolerance*"
  - He argues that data centers need very snappy response, hence availability is paramount
  - And they should be responsive even if a transient fault makes it hard to reach some service. So they should use cached data to respond faster even if the cached entry can't be validated and might be stale!
- Conclusion: weaken consistency for faster response

Brewer's Conjecture became The CAP Theorem

- Started as a conjecture, in 2002 was "proven"* and became a theorem, but some researchers still argue that the "proof" is incomplete^

The CAP theorem, also known as Brewer's theorem, states that it is impossible for a distributed system to simultaneously provide all three of the following guarantees:

- Consistency (all nodes see the same data at the same time)
- Availability (a guarantee that every request to a non-failing node receives a response about whether it was successful or failed)
- Partition tolerance (the system continues to operate despite arbitrary message loss or failure of part of the system)

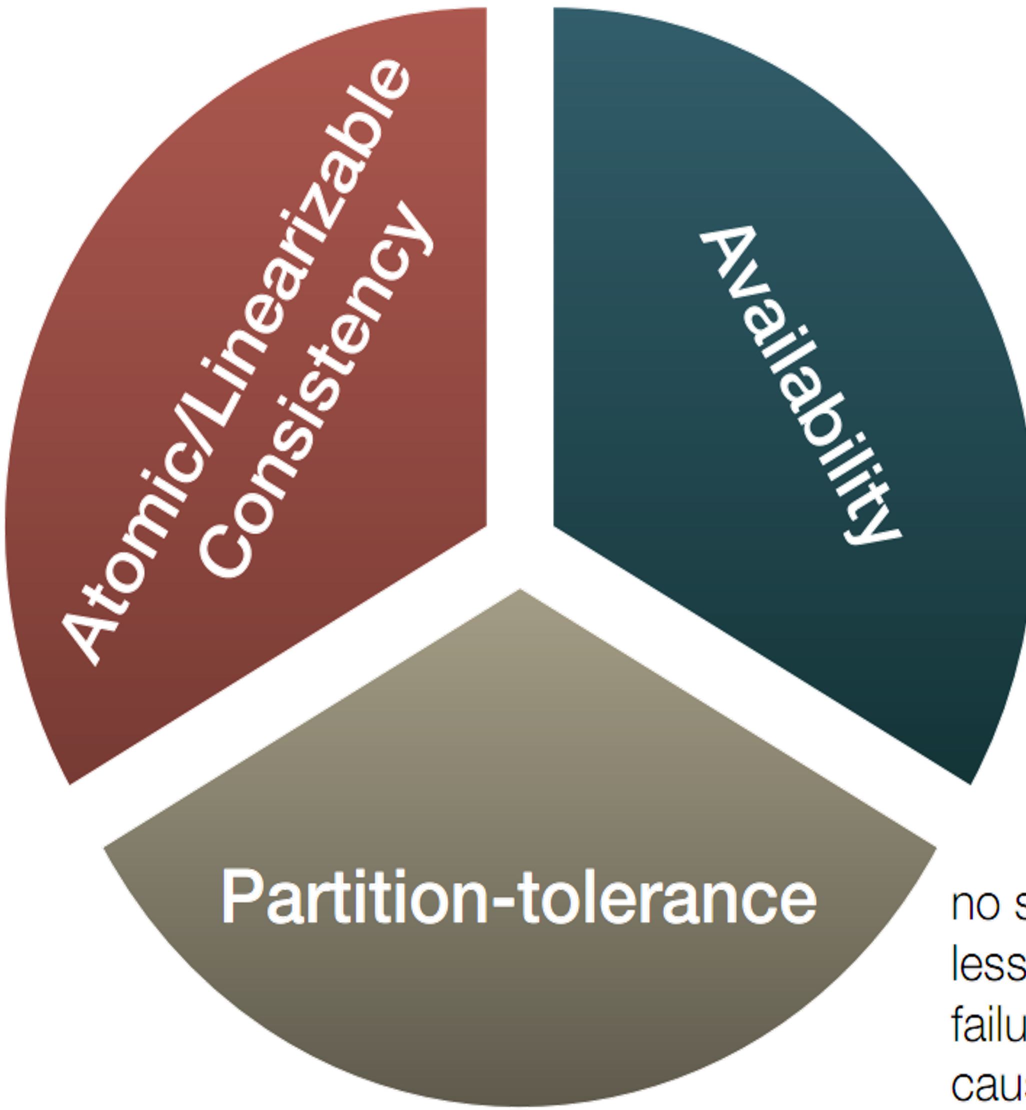
* Nancy Lynch and Seth Gilbert, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", *ACM SIGACT News*, Volume 33 Issue 2 (2002), pg. 51-59.

^ Mark Burgess, "Deconstructing the `CAP theorem' for CM and DevOps"

Definitions

Seth Gilbert and Nancy Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. ACM SIGACT News, vol. 33, no. 2, 2002, p. 51-59.

∃ total order
∀ operations
so that they look
as if they were
completed at a
single instant



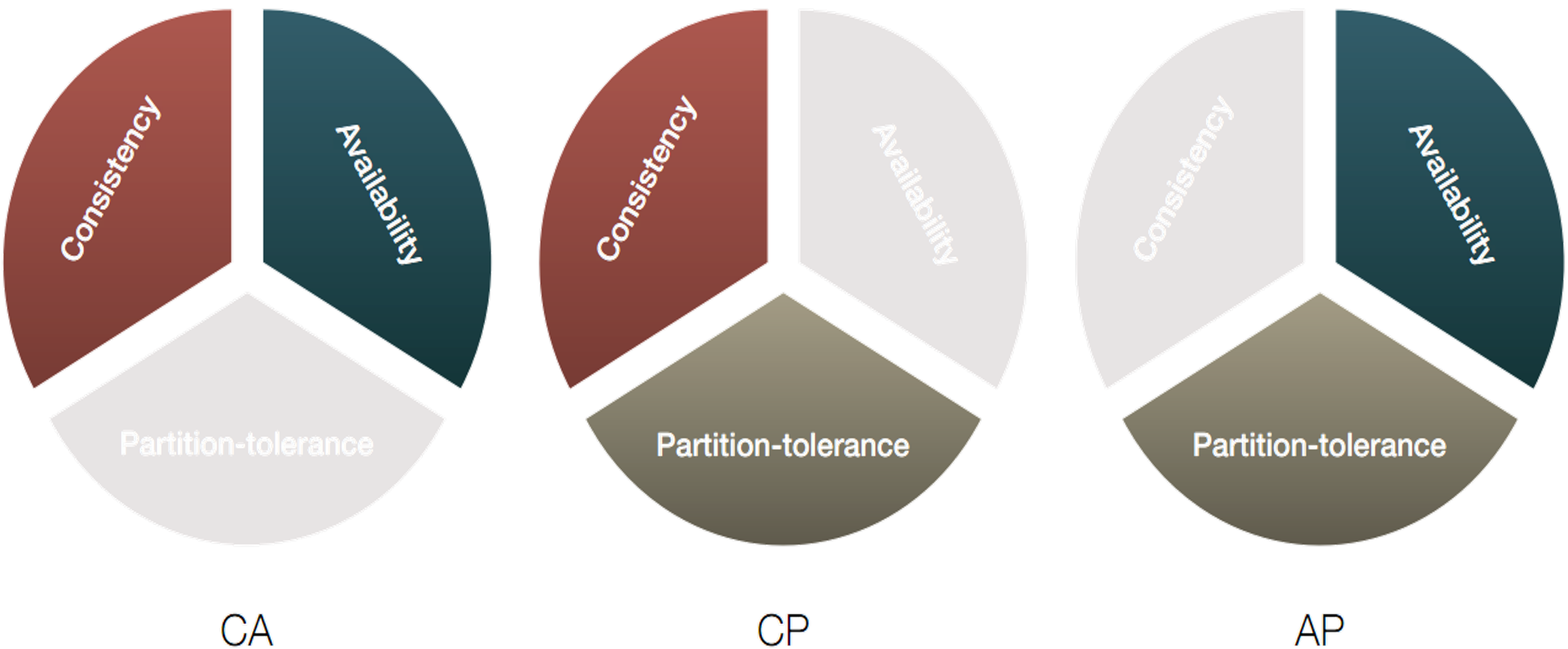
every request
received by a
non-failing
node must result
in a response

no set of failures
less than total network
failure is allowed to
cause the system to
respond incorrectly

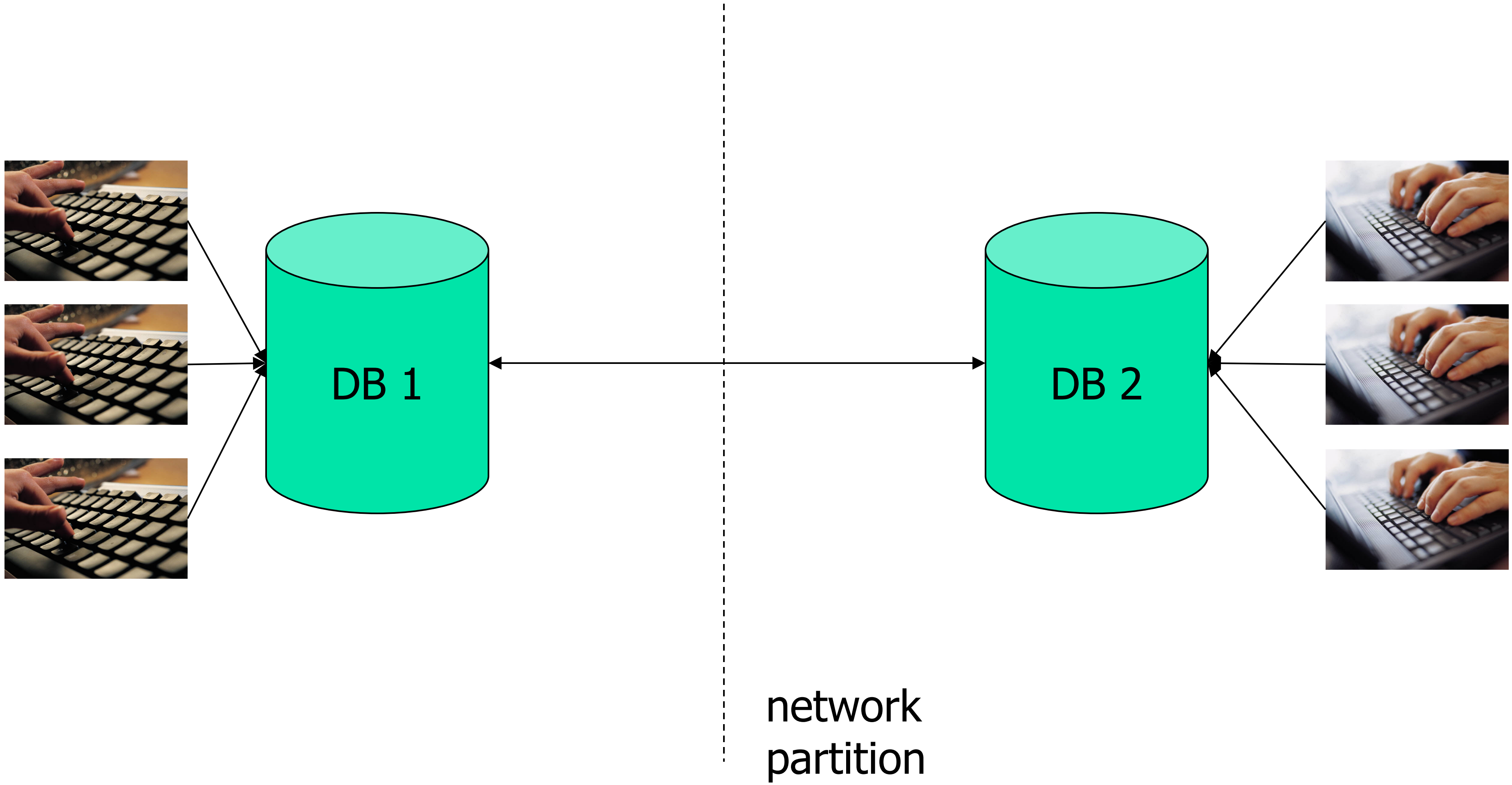
CAP Theorem

Eric A. Brewer. Towards Robust Distributed Systems. PODC Keynote, 2000.

In shared-data systems only two of the three CAP properties can be achieved at one moment in time.



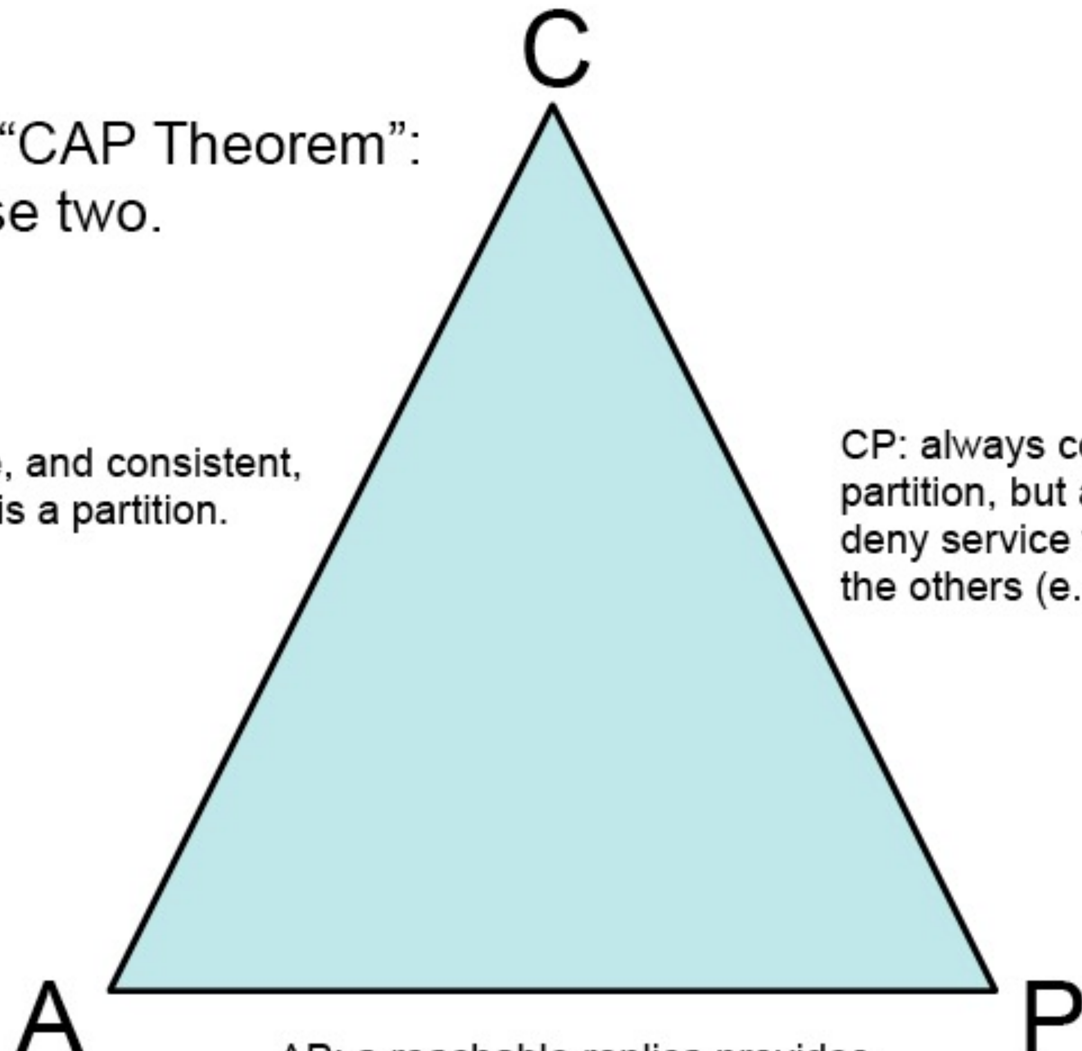
Intuition Behind Proof



Idea of the Proof for the CAP theorem

- Suppose a data center service is active in two parts of the country with a wide-area Internet link between them
- We temporarily cut the link ("partitioning" the network)
- And present the service with conflicting requests
- The replicas can't talk to each other so can't sense the conflict
- If they respond at this point, inconsistency arises

Fox&Brewer “CAP Theorem”:
C-A-P: choose two.



Partial List of Managed NoSQL services

	Model	CAP	Scans	Sec. Indices	Largest Cluster	Lear-ning	Lic.	DBaaS
HBase	Wide-Column	CP	Over Row Key	✗	~700	1/4	Apache	✗ (EMR)
MongoDB	Doc-ument	CP	yes	✓	>100 <500	4/4	GPL	mongoHQ
Riak	Key-Value	AP	✗	✓	~60	3/4	Apache	✗ (Softlayer)
Cassandra	Wide-Column	AP	With Comp. Index	✓	>300 <1000	2/4	Apache	instaclustr
Redis	Key-Value	CA	Through Lists, etc.	manual	N/A	4/4	BSD	Amazon ElastiCache

Source: Felix Gessert, "Cloud Databases in Research and Practice," Apr 2014, baqend.com/nosql.pdf

Partial List of Proprietary Database Service

	Model	CAP	Scans	Sec. Indices	Queries	API	Scale-out	SLA
SimpleDB	Table-Store	CP	Yes (as queries)	Auto-matic	SQL-like (no joins, groups, …)	REST + SDKs	✗	✗
Dynamo-DB	Table-Store	CP	By range key / index	Local Sec. Global Sec.	Key+Cond. On Range Key(s)	REST + SDKs	Automatic over Prim. Key	✗
Azure Tables	Table-Store	CP	By range key	✗	Key+Cond. On Range Key	REST + SDKs	Automatic over Part. Key	99.9% uptime
AE/Cloud DataStore	Entity-Group	CP	Yes (as queries)	Auto-matic	Conjunct. of Eq. Predicates	REST/ SDK, JDO,JPA	Automatic over Entity Groups	✗
S3, Az. Blob, GCS	Blob-Store	AP	✗	✗	✗	REST + SDKs	Automatic over key	99.9% uptime (S3)

Source: Felix Gessert, "Cloud Databases in Research and Practice," Apr 2014, baqend.com/nosql.pdf

Wait a Minute – Something doesn't seem right !

- The proof of the CAP Theorem actually only states:
  - If Network Partition occurs then one cannot get both Consistency and Availability at the same time, i.e.
    - (Network Partition) => not (Consistency and Availability)
    - It does not say anything when there is NO network partition
  - CA and CP systems are indistinguishable in practice (read the description in the edges of the triangle of the previous slide carefully): both behave the same without network partition ; but both show un-availability during partition

=> There are not really 3 different (i.e. CA, CP, AP) choices in practice

- Actual choice put forth by the Thm is AP vs. CA/CP
- The frequently quoted "At-most-2-out-of-3" claim of the CAP Theorem maybe slick but quite misleading*.

*Daniel J. Abadi, "Problems with CAP and Yahoo's little known NoSQL system," http://dbmsmusings.blogspot.hk/2010/04/problems-with-cap-and-yahoos-little.html

Problems with CAP

- Not as elegant as the MIT theorem
  - There are not three different choices!
  - CA and CP are indistinguishable
- Source of Confusion: Asymmetry of CAP properties
  - Some are properties of the system in general
  - Some are properties of the system only when there is a partition
- In any case, the CAP Theorem is frequently used as an excuse/justification to not bother with consistency
  - "Availability is really important to me, so CAP says I have to get rid of consistency"

CAP Examples

- CA/CP: Any consensus algorithm or state machine replication with a quorum required for service
  - Always consistent, even in a partition.
  - But the smaller (minority) partition will not be available during network partition.
- AP:
  - Always available if any replica is up and reachable, even during network partition.
  - But may not be consistent even without a partition.

Does CAP apply deeper in the cloud?

- The principle of wanting speed and scalability certainly is universal
- But many cloud services have strong consistency guarantees that we take for granted but depend on
- Marvin Theimer at Amazon explains:
  - Avoid costly guarantees that aren't even needed
  - But sometimes you just need to guarantee something
  - Then, be clever and engineer it to scale
  - And expect to revisit it each time you scale out 10x

Cloud services and their properties

Service	Properties it guarantees
Memcached	No special guarantees
Google's GFS	File is current if locking is used
BigTable	Shared key-value store with many consistency properties
Dynamo	Amazon's shopping cart: eventual consistency
Databases	Snapshot isolation with log-based mirroring (a fancy form of the ACID guarantees)
MapReduce	Uses a "functional" computing model within which offers very strong guarantees
Zookeeper	Yahoo! file system with sophisticated properties
PNUTS	Yahoo! database system, sharded data, spectrum of consistency options
Chubby	Locking service… very strong guarantees

Is there a conclusion to draw?

- One thing to notice about those services…
  - Most of them cost 10's or 100's of millions to create!
  - Huge investment required to build strongly consistent and scalable and high performance solutions
  - Oracle's current parallel database: billions invested
- CAP isn't about telling Oracle how to build a database product…
  - CAP is a warning to you that strong properties can easily lead to slow services
  - But thinking in terms of weak properties is often a successful strategy that yields a good solution and requires less effort

Going beyond CAP: PACELC*

- There are other costs to consistency (besides availability in the face of network partitions)
  - Overhead of synchronization schemes
  - Latency
    - If workload is geographically partitionable
      - Latency is not so bad
    - Otherwise
      - No way to get around at least one round-trip message
- PACELC
  - In the case of a partition (P), does the system choose availability (A) or consistency (C)?
  - Else (E), does the system choose latency (L) or consistency (C)?

*Daniel J. Abadi, "Consistency tradeoffs in modern distributed database system design," IEEE Computer Magazine, Feb. 2012.

Examples

- PA/EL
  - Dynamo, SimpleDB, Cassandra, Riptano, CouchDB, Cloudant
- PC/EC
  - ACID compliant database systems
- PA/EC
  - GenieDB
  - See CIDR paper from Wada, Fekete, et. al.
    - Indicates that Google App Engine data store (eventual consistent option) falls under this category
- PC/EL: Existence is debatable
  - Strengthening (instead of weakening) consistency when there is a partition doesn't seem to make sense

Core problem?

- When can we safely sweep consistency under the rug?
  - If we weaken a property in a safety critical context, something bad can happen!
  - Amazon and eBay do well with weak guarantees because many applications just didn’t need strong guarantees to start with!
  - By embracing their weaker nature, we reduce synchronization and so get better response behavior
- But what happens when a wave of high assurance applications starts to transition to cloud-based models?

Scalable Cloud Services often have a Tiered Architecture

- Tier 1: Very lightweight, responsive "web page builders" that can also route (or handle) "web services" method invocations. Limited to "soft state".
- Tier 2: (key,value) stores and similar services that support Tier 1. Basically, various forms of caches.
- Inner tiers: Online services that handle requests not handled in Tier 1. These can store persistent files, run transactional services. But we shield them from load.
- Back end: Runs offline services that do things like indexing the web overnight for use by tomorrow morning's Tier-1 services.

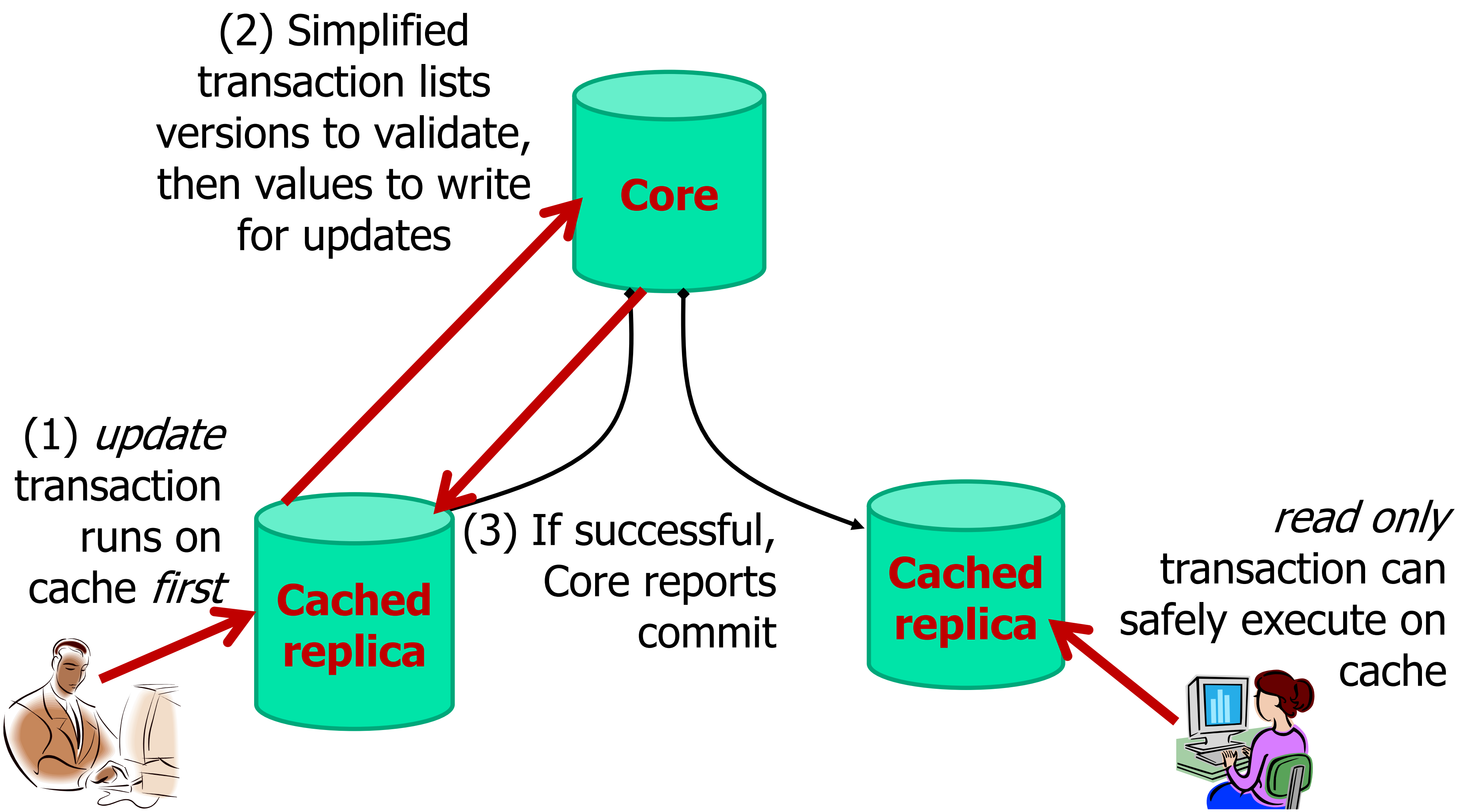
Is inconsistency a bad thing?

- How much consistency is really needed in the first tier (front-end portion) of the cloud?
  - Think about YouTube videos. Would consistency be an issue here?
  - What about the Amazon “number of units available” counters. Will people notice if those are a bit off?
- Puzzle: can you come up with a general policy for knowing how much consistency a given thing needs?

Consistency: Two “views”

- Client sees a snapshot of the database that is internally consistent and “might” be valid
- Internally, database is genuinely consistent, but the states clients saw aren’t tracked and might sometimes become invalidated by an update
- Inconsistency is tolerated because it yields such big speedups, although some clients see “wrong” results

A picture of how this works



Core issue: How much contention?

- Root challenge is to understand
  - How many updates will occur
  - How often those updates conflict with concurrent reads or with concurrent updates
- In most of today's really massive cloud applications either contention is very rare, in which case transactional database solutions work, or we end up cutting corners and relaxing consistency
- This has resulted in many practitioners declaring consistency in clouds dead!

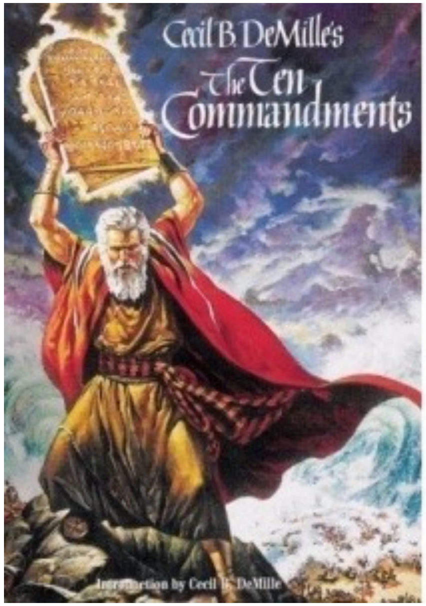
The Wisdom of the Sages

eBay's Five Commandments

- As described by Randy Shoup at LADIS 2008

Thou shalt…

1. **Partition Everything**
2. **Use Asynchrony Everywhere**
3. **Automate Everything**
4. **Remember: Everything Fails**
5. **Embrace Inconsistency**



Vogels at the Helm

- Werner Vogels is CTO at Amazon.com…
- He was involved in building a new shopping cart service
  - The old one used strong consistency for replicated data
  - New version was build over a DHT, like Chord, and has weak consistency with eventual convergence

- This weakens guarantees… but
  - ***Speed matters more than correctness***

James Hamilton's advice

- Key to scalability is decoupling, loosest possible synchronization
- *Any* synchronized mechanism is a risk
  - His approach: create a committee
  - Anyone who wants to deploy a highly consistent mechanism needs committee approval

.... They don't meet very often

Consistency

Consistency

- Two kinds of consistency:
  - Strong consistency – ACID(Atomicity Consistency Isolation Durability)
  - Weak consistency – BASE(Basically Available Soft-state Eventual consistency)

ACID Transactions

- A traditional DBMS is expected to support "*ACID transactions*," processes that are:
  - *Atomic* : Either the whole process is done or none is.
  - *Consistent* : Database constraints are preserved.
  - *Isolated* : It appears to the user as if only one process executes at a time.
  - *Durable* : Effects of a process do not get lost if the system crashes.

Eventual Consistency

- When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
- For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service
- Known as BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency), as opposed to ACID

All ACID implementations have costs

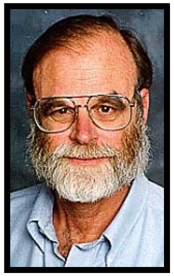
- Locking mechanisms involve competing for locks and there are overheads associated with how long they are held and how they are released at Commit

- Snapshot isolation mechanisms using locking for updates but also have an additional *version* based way of handling reads
  - Forces database to keep a history of each data item
  - As a transaction executes, picks the versions of each item on which it will run

- So… there are costs, not so small

Dangers of Replication

[The Dangers of Replication and a Solution . Jim Gray, Pat Helland, Dennis Shasha. Proc. 1996 ACM SIGMOD.]



- Investigated the costs of transactional ACID model on replicated data in "typical" settings
  - Found two cases
    - Embarrassingly easy ones: transactions that don't conflict at all (like Facebook updates by a single owner to a page that others might read but never change)
    - Conflict-prone ones: transactions that sometimes interfere and in which replicas could be left in conflicting states if care isn't taken to order the updates
  - Scalability for the latter case will be *terrible*
- Solutions they recommend involve sharding and coding transactions to favor the first case

Approach?

- They do a paper-and-pencil analysis
  - Estimate how much work will be done as transactions execute, roll-back
  - Count costs associated with doing/undoing operations and also delays due to lock conflicts that force waits
- Show that even under very optimistic assumptions slowdown will be $O(n^2)$ in size of replica set (shard)
- If approach is naïve, $O(n^5)$ slowdown is possible!

This motivates BASE

[D. Pritchett. BASE: An Acid Alternative. ACM Queue, July 28, 2008.]

- Proposed by eBay researchers
  - Found that many eBay employees came from transactional database backgrounds and were used to the transactional style of "thinking"
  - But the resulting applications didn't scale well and performed poorly on their cloud infrastructure
- Goal was to guide that kind of programmer to a cloud solution that performs much better
  - BASE reflects experience with real cloud applications
  - "Opposite" of ACID

A “methodology”

- BASE involves step-by-step transformation of a transactional application into one that will be far more concurrent and less rigid
  - But it doesn’t guarantee ACID properties
  - Argument parallels (and actually cites) CAP: they believe that ACID is too costly and often, not needed
  - BASE stands for “**Basically Available Soft-State Services with Eventual Consistency**”.

Terminology

- **Basically Available**: Like CAP, goal is to promote rapid responses.
  - BASE papers point out that in data centers partitioning faults are very rare and are mapped to crash failures by forcing the isolated machines to reboot
  - But we may need rapid responses even when some replicas can't be contacted on the critical path

Terminology

- **Basically Available**: Fast response even if some replicas are slow or crashed
- **Soft State Service**: Runs in first tier
  - Can’t store any permanent data
  - Restarts in a “clean” state after a crash
  - To remember data either replicate it in memory in enough copies to never lose all in any crash or pass it to some other service that keeps “hard state”

Terminology

- **Basically Available**: Fast response even if some replicas are slow or crashed
- **Soft State Service**: No durable memory
- **Eventual Consistency**: OK to send "optimistic" answers to the external client
  - Could use cached data (without checking for staleness)
  - Could guess at what the outcome of an update will be
  - Might skip locks, hoping that no conflicts will happen
  - Later, if needed, correct any inconsistencies in an offline cleanup activity

Before BASE… and after

- Code was often much too slow, and scaled poorly, and end-user waited a long time for responses
- With BASE
  - Code itself is way more concurrent, hence faster
  - Elimination of locking, early responses, all make end-user experience snappy and positive
  - But we do sometimes notice oddities when we look hard

BASE side-effects

- Suppose an eBay auction is running fast and furious
  - Does every single bidder necessarily see every bid?
  - And do they see them in the identical order?
- Clearly, everyone needs to see the winning bid
- But slightly different bidding histories shouldn’t hurt much, and if this makes eBay 10x faster, the speed may be worth the slight change in behavior!

BASE side-effects

- Upload a YouTube video, then search for it
  - You may not see it immediately
- Change the “initial frame” (they let you pick)
  - Update might not be visible for an hour
- Access a FaceBook page when your friend says she’s posted a photo from the party
  - You may see an