CAP theorem follow-up and miscellanea…

CSE 536 Spring 2026

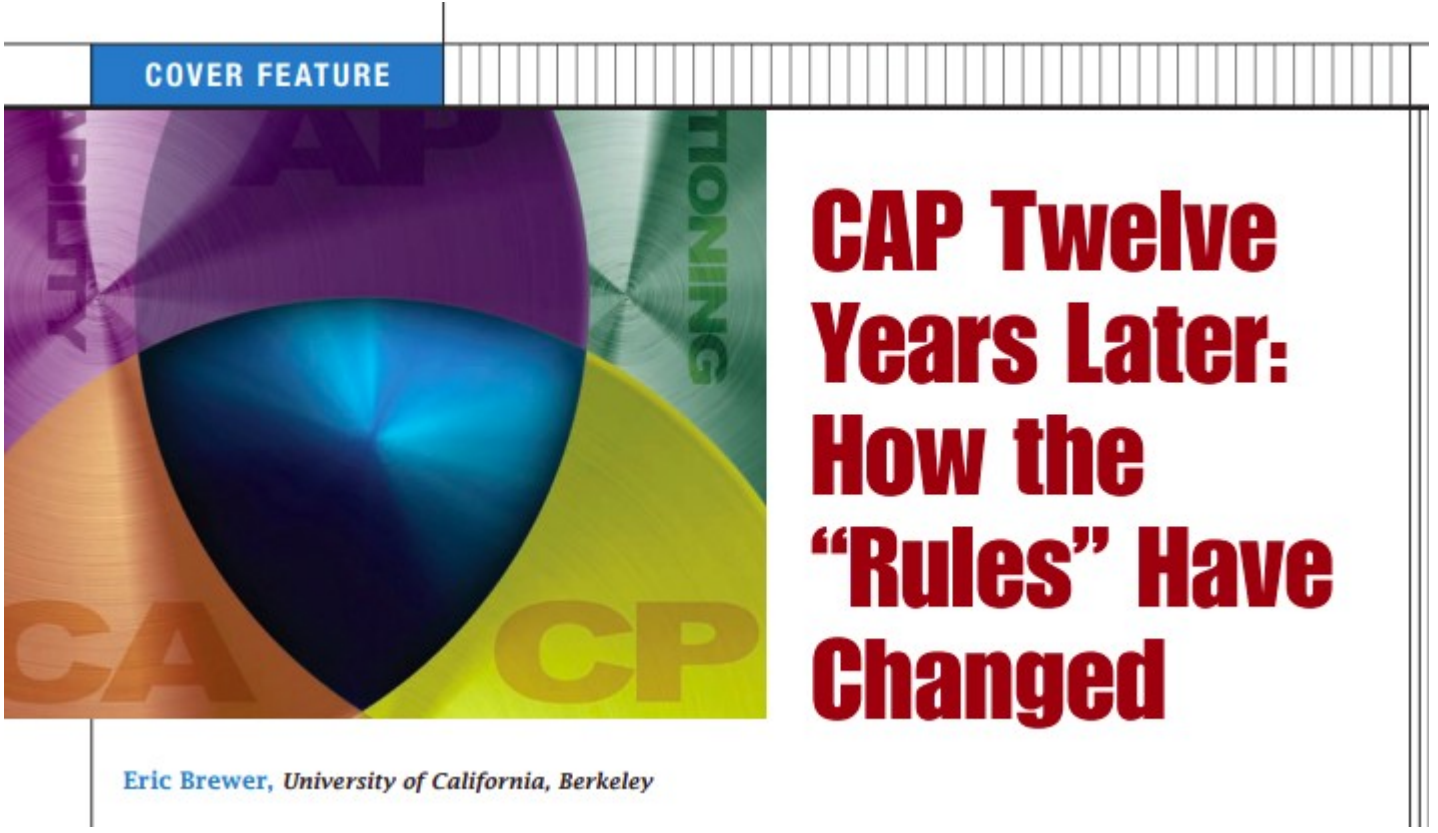
jedimaestro@asu.edu

Richard Feynman's core philosophy on testing ideas is that if a theory disagrees with experiment, it is wrong, regardless of its beauty or the author's intelligence. The key is to never fool yourself, as you are the easiest person to fool. Ideas must be validated by questioning, experimentation, and comparing results with nature. Speakola +5

Key Feynman Quotes on Testing Ideas

- **The Ultimate Test:** "If it disagrees with experiment, it is wrong. In that simple statement is the key to science".
- **Avoiding Self-Deception:** "The first principle is that you must not fool yourself — and you are the easiest person to fool".
- **Questioning Answers:** "I'd rather have questions I can't answer than answers I can't question".
- **Intellectual Honesty:** "...if you're doing an experiment, you should report everything that you think might make it invalid — not only what you think is right about it".
- **On Understanding vs. Memorization:** "Without using the new word which you have just learned, try to rephrase what you have just learned in your own language".
- **The Role of Doubt:** "The question of doubt and uncertainty is what is necessary to begin".
- **Challenging Authority:** "Have no respect for authority". Speakola +6

https://sites.cs.ucsb.edu/~rich/class/cs293b-cloud/papers/brewer-cap.pdf



The CAP theorem states that any networked shared-data system can have at most two of three desirable properties:

- *consistency* (C) equivalent to having a single up-to-date copy of the data;
- *high availability* (A) of that data (for updates); and
- tolerance to network *partitions* (P).

WHY "2 OF 3" IS MISLEADING

The easiest way to understand CAP is to think of two nodes on opposite sides of a partition. Allowing at least one node to update state will cause the nodes to become inconsistent, thus forfeiting C. Likewise, if the choice is to preserve consistency, one side of the partition must act as if it is unavailable, thus forfeiting A. Only when nodes communicate is it possible to preserve both consistency and availability, thereby forfeiting P. The general belief

As the "CAP Confusion" sidebar explains, the "2 of 3" view is misleading on several fronts. First, because partitions are rare, there is little reason to forfeit C or A when the system is not partitioned. Second, the choice between C and A can occur many times within the same system at very fine granularity; not only can subsystems make different choices, but the choice can change according to the operation or even the specific data or user involved. Finally, all three properties are more continuous than binary. Availability is obviously continuous from 0 to 100 percent, but there are also many levels of consistency, and even partitions have nuances, including disagreement within the system about whether a partition exists.

CAP-LATENCY CONNECTION

In its classic interpretation, the CAP theorem ignores latency, although in practice, latency and partitions are deeply related. Operationally, the essence of CAP takes place during a timeout, a period when the program must make a fundamental decision—the *partition decision*:

- cancel the operation and thus decrease availability, *or*
- proceed with the operation and thus risk inconsistency.

Retrying communication to achieve consistency, for example, via Paxos or a two-phase commit, just delays the decision. At some point the program must make the decision; retrying communication indefinitely is in essence choosing C over A.

https://martinfowler.com/articles/patterns-of-distributed-systems/two-phase-commit.html

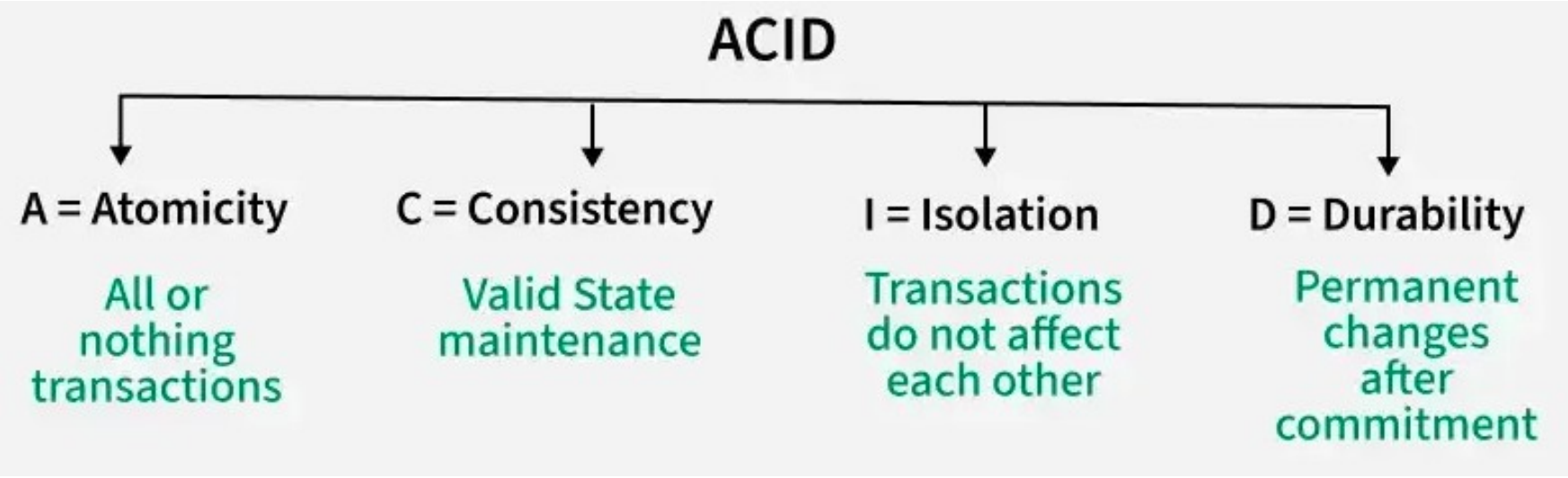
The essence of two-phase commit, unsurprisingly, is that it carries out an update in two phases:

1. The prepare phase asks each node if it can promise to carry out the update.
2. The commit phase actually carries it out.

Wikipedia: “The greatest disadvantage of the two-phase commit protocol is that it is a blocking protocol.”

Thus, pragmatically, a partition is a time bound on communication. Failing to achieve consistency within the time bound implies a partition and thus a choice between C and A for this operation. These concepts capture the core design issue with regard to latency: are two sides moving forward without communication?

https://www.geeksforgeeks.org/dbms/acid-properties-in-dbms/



Although both terms are more mnemonic than precise, the BASE acronym (being second) is a bit more awkward: Basically Available, Soft state, Eventually consistent. Soft state and eventual consistency are techniques that work well in the presence of partitions and thus promote availability.

REST *vs.* RPC

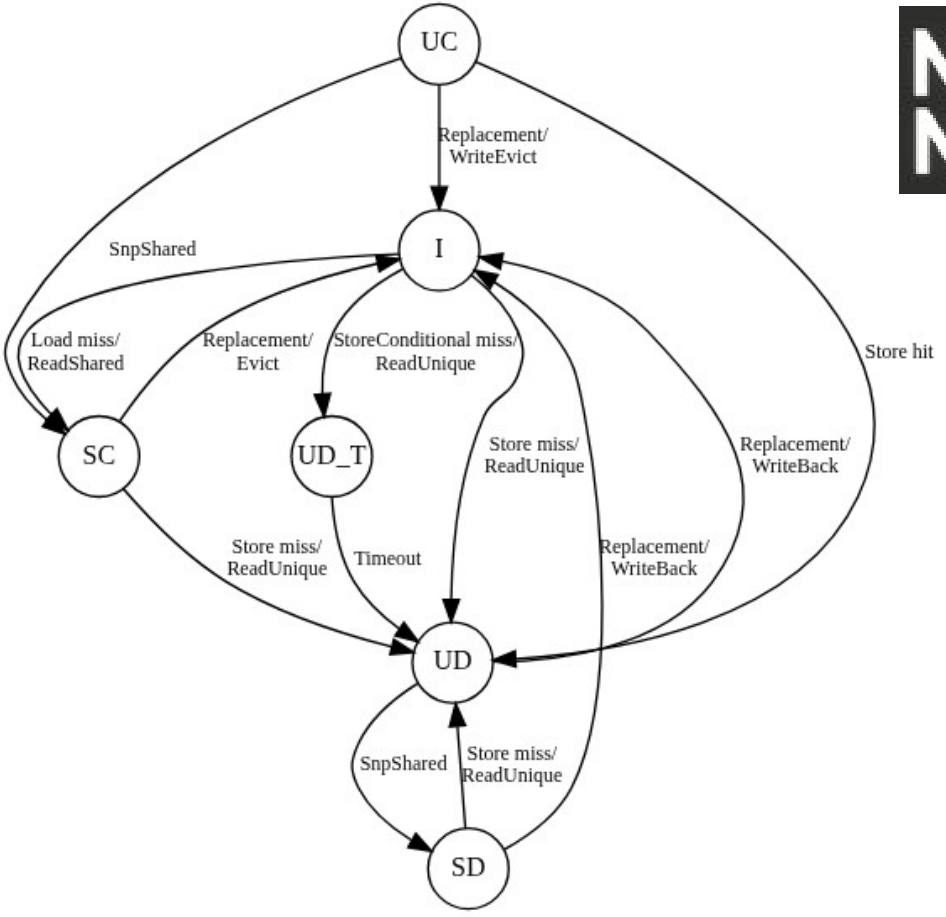
- REST is about nouns
  - GET /users/id/

Stateless, cacheable, *etc.*

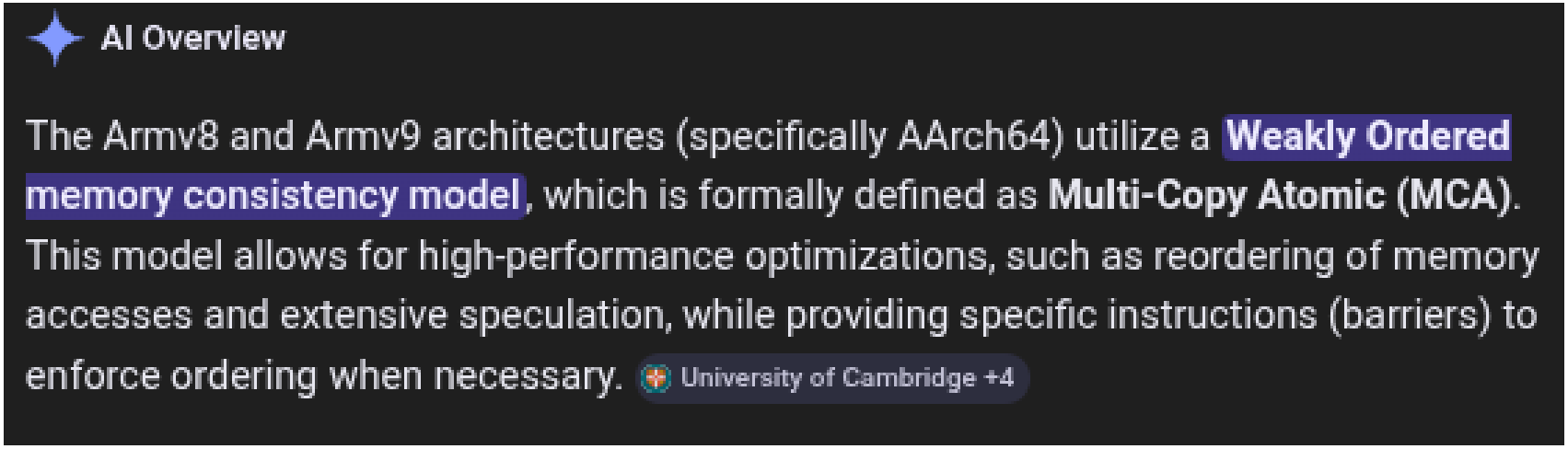
- RPC is about verbs
  - GetUserId()@…

Marshaling, stubs

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

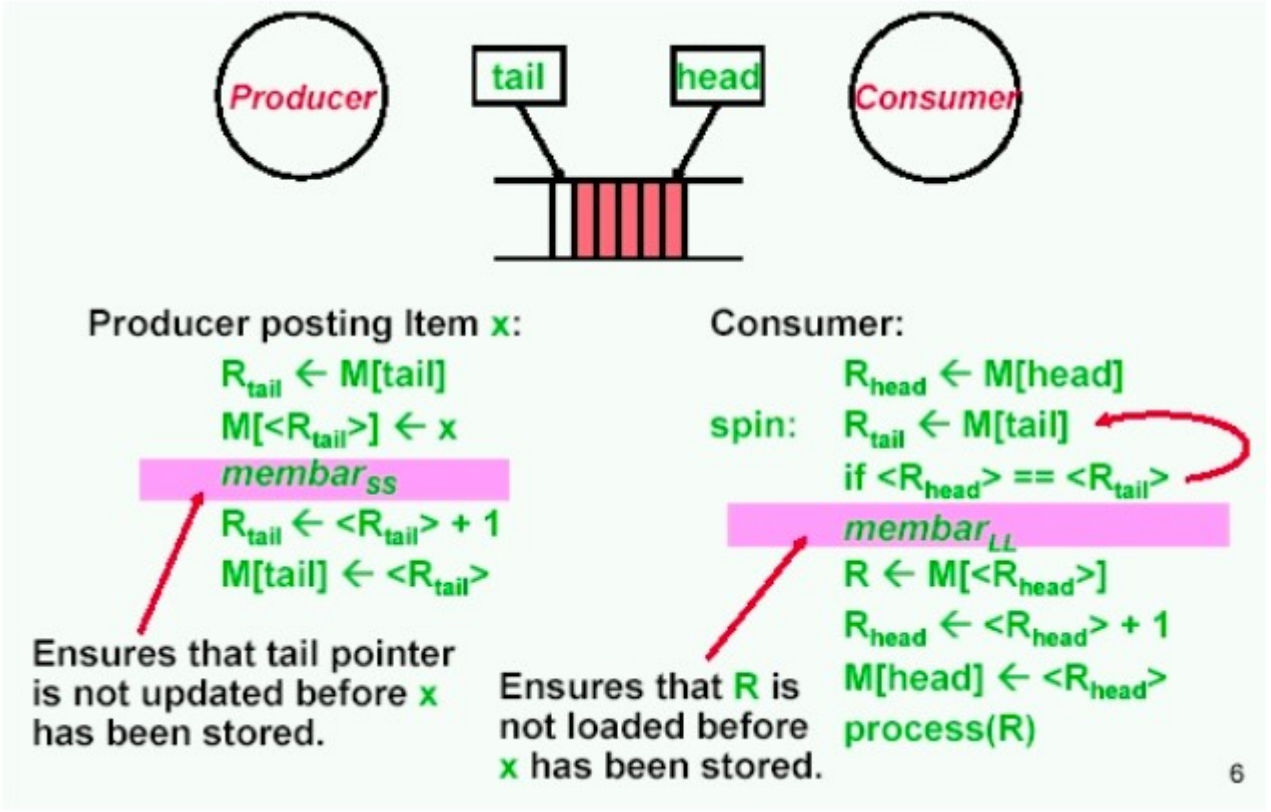


https://www.gem5.org/documentation/general_docs/ruby/CHI/



https://www.cl.cam.ac.uk/~pes20/armv8-mca/armv8-mca-draft.pdf

https://www.cl.cam.ac.uk/teaching/0506/CompArch/parallel/weak.html



nfs network file system consistency model

The **Network File System (NFS)** primarily employs a **close-to-open** **consistency** model. This model ensures that once a client finishes writing to a file and closes it, any subsequent open of that same file by another client will see those updates. UCLA Computer Science Department +3

Key Consistency Principles

- **Close-to-Open Consistency**: When a client closes a file, all pending writes are flushed to the server. Conversely, when a client opens a file, it checks with the server (often via a `GETATTR` request) to see if the file has been modified. If the server's version is newer, the client invalidates its local cache to ensure it reads the latest data.
- **Unlock-to-Lock Consistency**: Similar to close-to-open, when a client releases a file lock, it flushes updates to the server. The next client to acquire a lock on that file is guaranteed to see the most recent data.
- **Attribute Caching (GETATTR)**: To reduce server load, clients cache file attributes (like modification time) for a short period (typically 3 to 60 seconds). This introduces a "window of vulnerability" where a client might use stale data because it hasn't yet re-validated its cache with the server.
- **Weak Concurrency Guarantees**: NFS does **not** provide strong "write-to-read" consistency. If one client is writing and another is reading the same file simultaneously without closing/opening or locking, the reader may see an outdated or partially updated version of the file. UCLA Computer Science Department +7

What does all this mean about your semester project?

Asynchronous I/O

- What makes this a solution to distributed system communication? After all, you implement it in a single process.
- Why not just use threads?

RPC

- What semantics does your RPC library of choice use? (Can also ask this about message passing and others).
- Does your application care about network partitions? Consistency? Availability? Broadcast? Multicast?

Message passing

- What kind of marshaling are you using? SOAP? (Can also ask this about RPC or others).
- How does your approach scale *vs.* your groupmates? How simple is your code *vs.* your groupmates?