

[Skip to content](#)

**Martin Kleppmann**

› [About/Contact](#)

---

# Please stop calling databases CP or AP

In his excellent blog post [Notes on Distributed Systems for Young Bloods](#), Jeff Hodges Tweet recommends that you use the [CAP theorem](#) to critique systems. A lot of people have taken that advice to heart, describing their systems as “CP” (consistent but not available under network partitions), “AP” (available but not consistent under network partitions), or sometimes “CA” (meaning “I still haven’t read [Coda’s post from almost 5 years ago](#)”).

I agree with all of Jeff’s other points, but with regard to the CAP theorem, I must disagree. The CAP theorem is too simplistic and too widely misunderstood to be of much use for characterizing systems. Therefore I ask that we retire all references to the CAP theorem, stop talking about the CAP theorem, and put the poor thing to rest. Instead, we should use more precise terminology to reason about our trade-offs.

(Yes, I realize the irony of writing a blog post about the very topic that I am asking people to stop writing about. But at least it gives me a URL that I can give to people when they ask why I don’t like them talking about the CAP theorem. Also, apologies if this is a bit of a rant, but at least it’s a rant with lots of literature references.)

## CAP uses very narrow definitions

If you want to refer to CAP as a *theorem* (as opposed to a vague hand-wavy concept in your database’s marketing materials), you have to be precise. Mathematics requires precision. The proof only holds if you use the words with the same meaning as they are used in [the proof](#). And the proof uses very particular definitions:

- › *Consistency* in CAP actually means [linearizability](#), which is a very specific (and very strong) notion of consistency. In particular it has got nothing to do with the C in ACID, even though that C also stands for “consistency”. I explain the meaning of linearizability below.
- › *Availability* in CAP is defined as “every request received by a non-failing [database] node in the system must result in a [non-error] response”. It’s not sufficient for *some* node to be able to handle the request: *any* non-failing node needs to be able to handle it. Many so-called “highly available” (i.e. low downtime) systems actually do not meet this definition of availability.
- › *Partition Tolerance* (terribly mis-named) basically means that you’re communicating over an [asynchronous network](#) that may delay or drop messages. The internet and all our datacenters [have this property](#), so you don’t really have any choice in this matter.

Also note that the CAP theorem doesn’t just describe any old system, but a very specific model of a system:

- › The CAP system model is a single, read-write register – that’s all. For example, the CAP theorem says nothing about transactions that touch multiple objects: they are simply out of scope of the theorem, unless you can somehow reduce them down to a single register.
- › The only fault considered by the CAP theorem is a network partition (i.e. nodes remain up, but the network between some of them is not working). That kind of fault absolutely **does happen**, but it’s not the only kind of thing that can go wrong: nodes can crash or be rebooted, you can run out of disk space, you can hit a bug in the software, etc. In building distributed systems, you need to consider a much wider range of trade-offs, and focussing too much on the CAP theorem leads to ignoring other important issues.
- › Also, the CAP theorem says nothing about latency, which people **tend to care about more** than availability. In fact, CAP-available systems are allowed to be arbitrarily slow to respond, and can still be called “available”. Going out on a limb, I’d guess that your users wouldn’t call your system “available” if it takes 2 minutes to load a page.

If your use of words matches the precise definitions of the proof, then the CAP theorem applies to you. But if you’re using some other notion of consistency or availability, you can’t expect the CAP theorem to still apply. Of course, that doesn’t mean you can suddenly do impossible things, just by redefining some words! It just means that you can’t turn to the CAP theorem for guidance, and you cannot use the CAP theorem to justify your point of view.

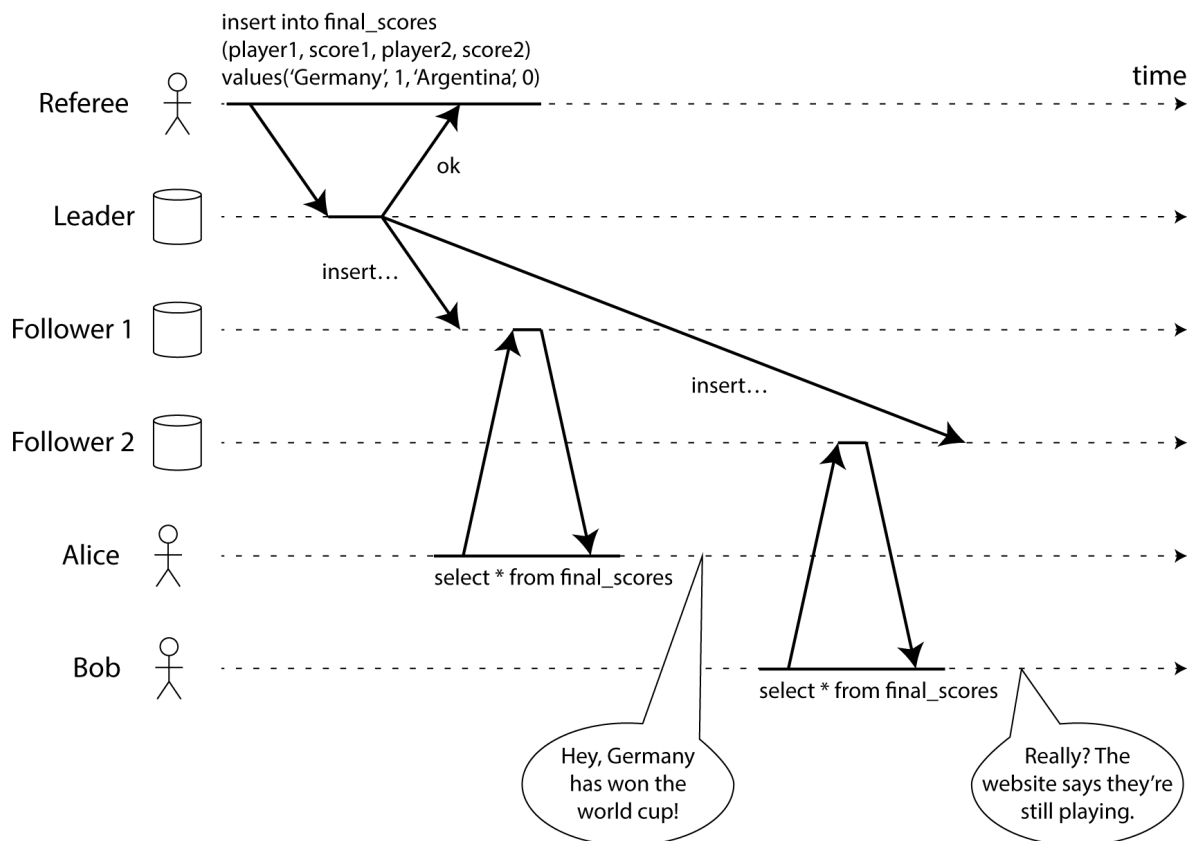
If the CAP theorem doesn’t apply, that means you have to think through the trade-offs yourself. You can reason about consistency and availability using your own definitions of those words, and you’re welcome to prove your own theorem. But please don’t call it CAP theorem, because that name is already taken.

## Linearizability

In case you’re not familiar with linearizability (i.e. “consistency” in the CAP sense), let me explain it briefly. The **formal definition** is not entirely straightforward, but the key idea, stated informally, is this:

If operation B started after operation A successfully completed, then operation B must see the the system in the same state as it was on completion of operation A, or a newer state.

To make this more tangible, consider an example of a system that is *not* linearizable. See the following diagram (sneak preview from an unreleased chapter of **my book**):



This diagram shows Alice and Bob, who are in the same room, both checking their phones to see the outcome of the **2014 football world cup final**. Just after the final score is announced, Alice refreshes the page, sees the winner announced, and excitedly tells Bob about it. Bob incredulously hits *reload* on his own phone, but his request goes to a database replica that is lagging, and so his phone shows that the game is still ongoing.

If Alice and Bob had hit reload at the same time, it wouldn't have been surprising if they had got two different query results, because they don't know at exactly what time their respective requests were processed by the server. However, Bob knows that he hit the reload button (initiated his query) *after* he heard Alice exclaim the final score, and therefore he expects his query result to be at least as recent as Alice's. The fact that he got a stale query result is a violation of linearizability.

Knowing that Bob's request happened strictly after Alice's request (i.e. that they were not concurrent) depends on the fact that Bob heard about Alice's query result through a separate communication channel (in this case, IRL audio). If Bob hadn't heard from Alice that the game was over, he wouldn't have known that the result of his query was stale.

If you're building a database, you don't know what kinds of backchannel your clients may have. Thus, if you want to provide linearizable semantics (CAP-consistency) in your database, you need to make it appear as though there is only a single copy of the data, even though there may be copies (replicas, caches) of the data in multiple places.

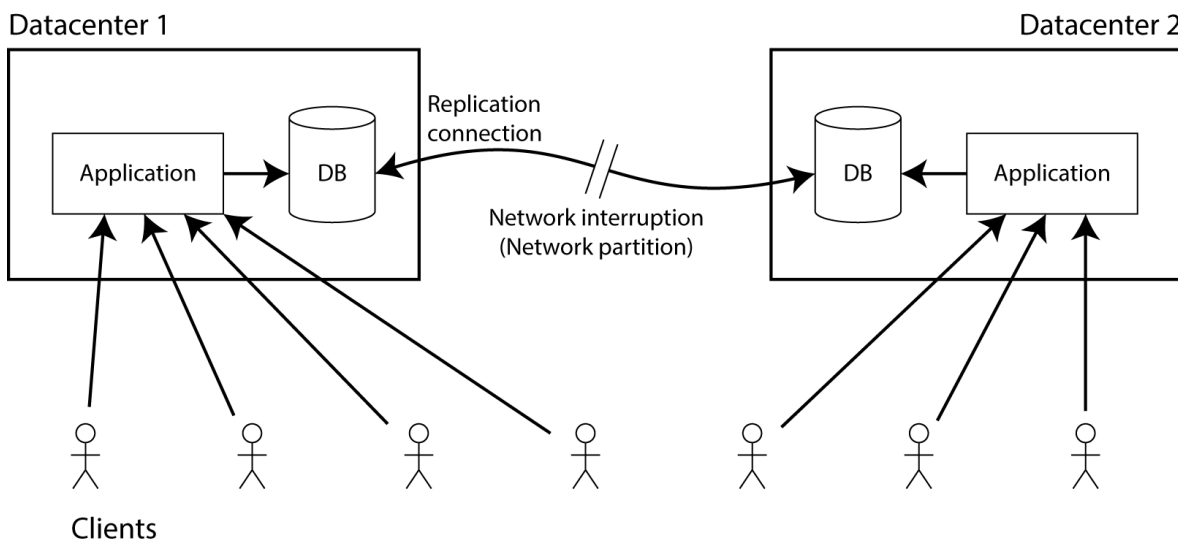
This is a fairly expensive guarantee to provide, because it requires a lot of coordination. Even the CPU in your computer **doesn't provide linearizable access to your local RAM!** On modern CPUs, you need to use an explicit **memory barrier instruction** in order to get linearizability. And even testing whether a system provides linearizability is **tricky**.

# CAP-Availability

Let's talk briefly about the need to give up either linearizability or availability in the case of a network partition.

Let's say you have replicas of your database in two different datacenters. The exact method of replication doesn't matter for now – it may be single-leader (master/slave), multi-leader (master/master) or quorum-based replication (**Dynamo-style**). The requirement of replication is that whenever data is written to data in one datacenter, it also has to be written to the replica in the other datacenter. Assuming that clients only connect to one datacenter, there must be a network link between the two datacenters over which the replication happens.

Now assume that network link is interrupted – that's what we mean with a *network partition*. What happens?



Clearly you can choose one of two things:

1. The application continues to be allowed to write to the database, so it remains fully available in both datacenters. However, as long as the replication link is interrupted, any changes that are written in one datacenter will not appear in the other datacenter. This violates linearizability (in terms of the previous example, Alice could be connected to DC 1 and Bob could be connected to DC 2).
2. If you don't want to lose linearizability, you have to make sure you do all your reads and writes in one datacenter, which you may call the *leader*. In the other datacenter (which cannot be up-to-date, due to the failed replication link), the database must stop accepting reads and writes until the network partition is healed and the database is in sync again. Thus, although the non-leader database has not failed, it cannot process requests, so it is not CAP-available.

(And this, by the way, is essentially the proof of the CAP theorem. That's all there is to it. This example uses two datacenters, but it applies equally to network problems within a single datacenter. I just find it easier to think about when I imagine it as two datacenters.)

Note that in our notionally "unavailable" situation in option 2, we're still happily processing requests in one of the datacenters. So if a system chooses linearizability (i.e. it is not CAP-available), that doesn't necessarily

mean that a network partition automatically leads to an outage of the application. If you can shift all clients to using the leader datacenter, the clients will in fact see no downtime at all.

Availability in practice **does not quite correspond** to CAP-availability. Your application's availability is probably measured with some SLA (e.g. 99.9% of well-formed requests must return a successful response within 1 second), but such an SLA can be met both with CAP-available and CAP-unavailable systems.

In practice, multi-datacenter systems *are* often designed with asynchronous replication, and thus non-linearizable. However, the reason for that choice is often the latency of wide-area networks, not just wanting to tolerate datacenter and network failures.

## Many systems are neither linearizable nor CAP-available

Under the CAP theorem's strict definitions of consistency (linearizability) and availability, how do systems fare?

For example, take any replicated database with a single leader, which is the standard way of setting up replication in most relational databases. In this configuration, if a client is partitioned from the leader, it cannot write to the database. Even though it may be able to read from a follower (a read-only replica), the fact that it cannot write means any single-leader setup is not CAP-available. Never mind that such configurations are often marketed as "high availability".

If single-leader replication is not CAP-available, does that make it "CP"? Wait, not so fast. If you allow the application to make reads from a follower, and the replication is asynchronous (the default in most databases), then a follower may be a little behind the leader when you read from it. In this case, your reads will not be linearizable, i.e. not CAP-consistent.

Moreover, databases with **snapshot isolation**/MVCC are intentionally non-linearizable, because enforcing linearizability would reduce the level of concurrency that the database can offer. For example, **PostgreSQL's SSI** provides *serializability* but not *linearizability*, and **Oracle provides neither**. Just because a database is branded "ACID" doesn't mean it meets the CAP theorem's definition of consistency.

So these systems are neither CAP-consistent nor CAP-available. They are neither "CP" nor "AP", they are just "P", whatever that means. (Yes, the "two out of three" formulation *does* allow you to pick only one out of three, or even none out of three!)

What about "NoSQL"? Take MongoDB, for example: it has a single leader per shard (or at least it's supposed to, if it's not in split-brain mode), so it's not CAP-available by the argument above. And Kyle **recently showed** that it allows non-linearizable reads even at the highest consistency setting, so it's not CAP-consistent either.

And the **Dynamo** derivatives like Riak, Cassandra and Voldemort, which are often called "AP" since they optimize for high availability? It depends on your settings. If you accept a single replica for reads and writes ( $R=W=1$ ), they are indeed CAP-available. However, if you require quorum reads and writes ( $R+W>N$ ), and you have a network partition, clients on the minority side of the partition cannot reach a quorum, so quorum operations are not CAP-available (at least temporarily, until the database sets up additional replicas on the minority side).

You sometimes see people people claiming that quorum reads and writes guarantee linearizability, but I think it would be unwise to rely on it – subtle combinations of features such as sloppy quorums and read repair can lead to **tricky edge cases** in which deleted data is resurrected, or the number of replicas of a value falls below the original  $W$  (violating the quorum condition), or the number of replica nodes increases above the original  $N$  (again violating the quorum condition). All of these lead to non-linearizable outcomes.

These are not bad systems: people successfully use them in production all the time. However, so far we haven't been able to rigorously classify them as "AP" or "CP", either because it depends on the particular operation or configuration, or because the system meets neither of the CAP theorem's strict definitions of consistency or availability.

## Case study: ZooKeeper

What about ZooKeeper? It uses a **consensus algorithm**, so people generally regard it as a **clear-cut case of choosing consistency over availability** (i.e. a "CP system").

However, if you look at the **ZooKeeper docs**, they make quite clear that ZooKeeper by default *does not* provide linearizable reads. Each client is connected to one of the server nodes, and when you make a read, you see only the data on that node, even if there are more up-to-date writes on another node. This makes reads much faster than if you had to assemble a quorum or contact the leader for every read, but it also means that ZooKeeper by default *does not* meet the CAP theorem's definition of consistency.

It is possible to make linearizable reads in ZooKeeper by **preceding a read with a sync command**. That isn't the default though, because it comes with a performance penalty. People do use sync, but usually not all the time.

What about ZooKeeper availability? Well, ZK requires a **majority quorum** in order to reach consensus, i.e. in order to process writes. If you have a partition with a majority of the nodes on one side and a minority on the other, then the majority side continues to function, but the nodes on the minority side can't process writes, even though the nodes are up. Thus, writes in ZK are not CAP-available under a partition (even though the majority side can continue to process writes).

To add to the fun, ZooKeeper 3.4.0 added a **read-only mode**, in which nodes on the minority side of a partition can continue serving read requests – no quorum needed! This read-only mode *is* CAP-available. Thus, ZooKeeper by default is neither CAP-consistent (CP) nor CAP-available (AP) – it's really just "P". However, you can optionally make it CP by calling sync if you want, and for reads (but not for writes) it's actually AP, if you turn on the right option.

But this is irritating. Calling ZooKeeper "not consistent", just because it's not linearizable by default, really badly misrepresents its features. It actually provides an excellent level of consistency! It provides **atomic broadcast** (which is **reducible to consensus**) combined with the session guarantee of **causal consistency** – which is **stronger** than **read your writes**, **monotonic reads** and **consistent prefix reads** combined. The documentation says that it provides **sequential consistency**, but it's under-selling itself, because ZooKeeper's guarantees are in fact much stronger than sequential consistency.

As ZooKeeper demonstrates, it is quite reasonable to have a system that is neither CAP-consistent nor CAP-available in the presence of partitions, and by default isn't even linearizable in the *absence* of



partitions. (I guess that would be PC/EL in **Abadi's PACELC framework**, but I don't find that any more enlightening than CAP.)

## CP/AP: a false dichotomy

The fact that we haven't been able to classify even one datastore as unambiguously "AP" or "CP" should be telling us something: those are simply not the right labels to describe systems.

I believe that we should stop putting datastores into the "AP" or "CP" buckets, because:

- › Within one piece of software, you may well have various operations with **different consistency characteristics**.
- › Many systems are neither consistent nor available under the CAP theorem's definitions. However, I've never heard anyone call their system just "P", presumably because it looks bad. But it's not bad – it may be a perfectly reasonable design, it just doesn't fit one of the two CP/AP buckets.
- › Even though most software doesn't neatly fit one of those two buckets, people try to shoehorn software into one of the two buckets anyway, thereby inevitably changing the meaning of "consistency" or "availability" to whatever definition suits them. Unfortunately, if the meaning of the words is changed, the CAP theorem no longer applies, and thus the CP/AP distinction is rendered completely meaningless.
- › A huge amount of subtlety is lost by putting a system in one of two buckets. There are many considerations of fault-tolerance, latency, simplicity of programming model, operability, etc. that feed into the design of a distributed systems. It is simply not possible to encode this subtlety in one bit of information. For example, even though ZooKeeper has an "AP" read-only mode, this mode still provides a total ordering of historical writes, which is a vastly stronger guarantee than the "AP" in a system like Riak or Cassandra – so it's ridiculous to throw them into the same bucket.
- › Even Eric Brewer **admits** that CAP is misleading and oversimplified. In 2000, it was meant to start a discussion about trade-offs in distributed data systems, and it did that very well. It wasn't intended to be a breakthrough formal result, nor was it meant to be a rigorous classification scheme for data systems. 15 years later, we now have a much greater range of tools with different consistency and fault-tolerance models to choose from. CAP has served its purpose, and now it's time to move on.

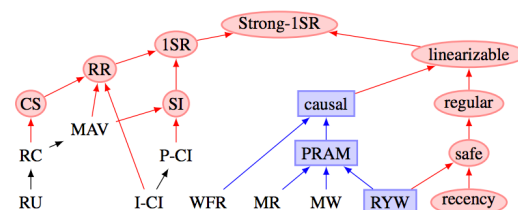
## Learning to think for yourself

If CP and AP are unsuitable to describe and critique systems, what should you use instead? I don't think there is one right answer. Many people have thought hard about these problems, and proposed terminology and models to help us understand problems. To learn about those ideas, you'll have to go deeper into the literature.

- › A good starting point is Doug Terry's paper in which he **explains various different levels of eventual consistency using Baseball examples**. It's very readable and clear, even if (like me) you're not American and have no clue about Baseball.
- › If you're interested in transaction isolation models (which is not the same as consistency of distributed

replicas, but somewhat related), my little project [Hermitage](#) may be relevant.

- › The connections between replica consistency, transaction isolation and availability are explored by [Peter Bailis et al.](#) (That paper also explains the meaning of that hierarchy of consistency levels which Kyle Kingsbury [likes to show](#).)



- › When you've read those, you should be ready to dive deeper into the literature. I've scattered a ton of links to papers throughout this post. Do take a look at them: a number of experts have already figured out a lot of stuff for you.
- › As a last resort, if you can't face reading the original papers, I suggest you take a look at [my book](#), which summarizes the most important ideas in an approachable manner. (See, I tried *very hard* not to make this post a sales pitch.)
- › If you want to know more specifically about using ZooKeeper correctly, [Flavio Junqueira and Benjamin Reed's book](#) is good.

Whatever way you choose to learn, I encourage you to be curious and patient – this stuff doesn't come easy. But it's rewarding, because you learn to reason about trade-offs, and thus figure out what kind of architecture works best for your particular application. But whatever you do, please stop talking about CP and AP, because they just don't make any sense.

Thank you to [Kyle Kingsbury](#) and [Camille Fournier](#) for comments on a draft of this post. Any errors or unpalatable opinions are mine, of course.

Join the discussion about this article [on Hacker News](#).

19 Comments

Martin Kleppmann's Blog

1 Login ▾

♥ Recommend 19 Share

Sort by Best ▾



Join the discussion...



**Coda Hale** · a year ago

"However, if you require quorum reads and writes ( $R+W>N$ ), and you have a network partition, clients on the minority side of the partition cannot reach a quorum, so quorum operations are not CAP-available (at least temporarily, until the database sets up additional replicas on the minority side)."

Cassandra, Riak, and Voldemort all implement what they call hinted handoff and what the Dynamo paper refers to as "sloppy quorum". If they can't achieve a quorate write, they will write to the coordinator node (or a random reachable node) which will then propagate the write when the cluster is no longer partitioned. Depending on the system and configuration, this may reduce to  $W=1$



and may reduce to write.

3 ^ | v • Reply • Share ›



**Martin Kleppmann** Mod → Coda Hale • a year ago

Hey Coda, thanks for chiming in. I agree about sloppy quorum for writes (I allude to it above, but didn't want to make the post even more complicated by going into a long digression about it).

However, what about reads? My understanding is that if a client can't reach R (non-sloppy) replicas, it won't be able to serve reads. Assuming  $R=W=2$ ,  $N=3$ , that would make reads CAP-unavailable.

Please correct me if I'm wrong, as I'm not a deep expert on Dynamo-style datastores.

^ | v • Reply • Share ›



**devnulldude** → Martin Kleppmann • a year ago

Cassandra clients have downgrade policies that would allow you to read at a different consistency level to trade off your C/A. E.g. try your read at quorum, if that fails, downgrade to CL:ONE

It won't be consistent because you'd be at the mercy of whatever that node's last known data was. For immutable data storage then you just have to worry about getting a NULL value which means the write wrote to A and B but not yet C

1 ^ | v • Reply • Share ›



**Luccas Maso** → devnulldude • a year ago

Nice point about sloppy Quorums, this is confused me too. But now, there is some scenario in Cassandra that makes it not Available in CAP? Maybe with Linearizable consistency setting?

^ | v • Reply • Share ›



**Will Holley** → Martin Kleppmann • a year ago

In CouchDB (2.0), which also uses Dynamo-style clustering, quorum is sloppy on reads as well. If  $R=2$  is specified but only one replica is available in a minority partition, it will respond successfully using that.

^ | v • Reply • Share ›



**Gaëtan Voyer-Perrault** • a year ago

I think there is still a basic value in the CP / AP dynamic.

That value is a simple shorthand for "what is the default behaviour when a partition happens"?

Yes, it's a shorthand. Yes, it would be great if people could read your 400-page book before making a decision about which database to use. But that's also asking a lot.

It's especially asking a lot from users who often don't actually need any form of sharding but

it's especially asking a lot from users who often don't actually need any form of sharding, but only kind of think they do. Often because of vague scalability concerns and a complete misunderstanding of the effort required to keep a 100 server DB cluster running.

2 ^ | v • Reply • Share ›



**Ram Krishnan** • 2 months ago

I would think it should be "Oracle provides both" instead of "neither" in the statement "For example, PostgreSQL's SSI provides serializability but not linearizability, and Oracle provides neither".

Going by your definition of "linearizability" in the "soccer score" graphic, you are guaranteed to get the same result, even though you hit different nodes of the same cluster. It is not guaranteed only in replicated systems, which anyway is not meant to cater to this scenario.

The same can be extended to serializability as well, as long as transaction is set to that mode.

^ | v • Reply • Share ›



**jabirfatah91** • 3 months ago

Nice article. Is there any NoSQL who directly focus only on Quorum? I mean to maintain consistency (whether its poor or strong), which Non-relational database implements Quorum?

^ | v • Reply • Share ›



**sreeni** • 5 months ago

great writeup !!! which set of editing tools do you use ?

^ | v • Reply • Share ›



**gleb Chermennov** • a year ago

great article, Martin - it helped a lot to clear up some terms that Kyle uses in his posts :)

^ | v • Reply • Share ›



**Konstantin Urysov** • a year ago

Hi, thank you for the article. I read very carefully your explanation of linearizability, but still did not catch the difference between linearizability and consistency, at least in your example. To my mind, the system follows "AP" rules, as it allows to Follower 1 and Follower 2 respond immediately without any delay for checking the actual state. Why than you called the system "not linearizable", but not "eventually consistent"?

When it comes to ZooKeeper, that's correct it is hard to say is the system "AP" or "CP", however it becomes much easier if we would look on the system in perspective of each request. Thus, when we are trying to read the data it will be "AP", and when we are making sync operation before reading it will be "CP". How do you think in case when we are discussing systems where we are able to change consistency level, can we use CAP theorem in case of particular requests to the systems?

^ | v • Reply • Share ›



**anomalizer** • a year ago



I had a similar gripe with people trying to classify systems as "two of three from CAP" but at a different level of abstraction. This a topic that I wrote about in an informal manner a couple of months ago <http://technology.inmobi.com/b...>

The bottom line is pretty much the same: acknowledging the various types of consistency opens up a far more wider choice of possibilities that requires us to think about the system at hand rather than hide behind the foregone conclusions of an CP / AP model.

^ | v • Reply • Share ›



**Imad Qureshi** • a year ago

This is actually a great post. I was hoping that you would talk about HBase also but you didn't but you talk about Zookeeper which HBase uses as it's coordination service. Zookeeper is not a Database. HBase and similarly Google's big table are strongly consistent. The user gets an answer or no answer at all. Would love to hear your thoughts.

^ | v • Reply • Share ›



**Martin Kleppmann** Mod ➔ Imad Qureshi • a year ago

What do you mean with "strongly consistent"? Is it a formally defined term? If you mean "linearizable", why not call it that?

About HBase, I don't know exactly. I believe it uses a single-leader design, which means it potentially could be linearizable — but as Kyle's MongoDB test shows, a single-leader architecture doesn't per se guarantee linearizability. Please test for yourself, and post your results!

^ | v • Reply • Share ›



**Imad Qureshi** ➔ Martin Kleppmann • a year ago

Well, both HBase and Google's Big Table guarantee linearizability by making sure that request for a particular row key always goes to the same server. Two clients asking a question will always get the same answer or no answer at all. Data is distributed across the cluster based on row key. In case of node failure, the data served by the failed server is then served by other nodes who have a copy (Hadoop makes sure three copies of data). By strongly consistent, I meant it is consistent as in CP.

Master in HBase is of less significance than other systems. On first call, client needs to connect to HMaster to find out which server is hosting the key, but then it can client directly goes to the server hosting the key. The only time this client will need to go to Master again is, if the server has failed or there is a split causing the key to move to a different server.

^ | v • Reply • Share ›



**Lucas Maso** • a year ago

Nice article man! You helped me clarify a lot of confusions, because I was trying to understand why some database were on the CP camp, but it does not allow linearizability consistency as proven in the CAP theorem. Same for MongoDB, it is available, but not in the means for A of CAP once a partition with non-failing nodes can proceed with read but not

write requests.

^ | v • Reply • Share ›



**PAUL Miller** • a year ago

It looks like the only one calling databases CP and AP are you, and no one else. Lets make a blog arguing some moot point that no one cares about, get people to read it, profit?!?!?!?

^ | v • Reply • Share ›



**Martin Kleppmann** Mod ➔ PAUL Miller • a year ago

I envy you for apparently not having endured any time on Hacker News or in the company of the roflscale crowd.

9 ^ | v • Reply • Share ›



**Tzach Livyatan** ➔ PAUL Miller • a year ago

Here is a good post, categorizing NoSQL to AP,CP and CA

<http://blog.nahurst.com/visual...>

I still find this categorizing useful, although, as demonstrated, not accurate.

1 ^ | v • Reply • Share ›



Subscribe



Add Disqus to your site Add Disqus Add



Privacy

## Subscribe

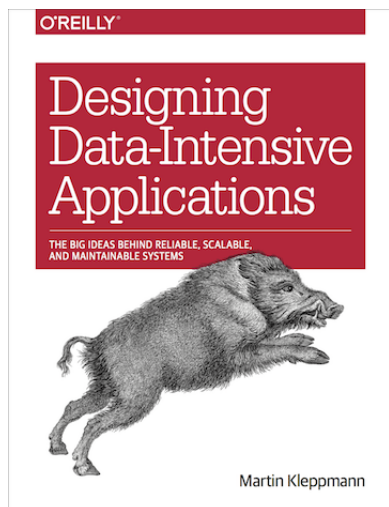
Site RSS feed

To get notified when I write something new, [follow me](#) on Twitter, [subscribe](#) to the RSS feed, or enter your email address:

Email address:

I won't give your address to anyone else, won't send you any spam, and you can unsubscribe at any time.

## My book



My book, [Designing Data-Intensive Applications](#), is available in early release from O'Reilly.

I am a researcher at the [University of Cambridge Computer Laboratory](#), working at the intersection of databases, distributed systems, and information security.

## Tweets by @martinkl



**Martin Kleppmann** @martinkl

Letters for Black Lives: [lettersforblacklives.com/dear-mom-dad-u...](https://lettersforblacklives.com/dear-mom-dad-u...)

I helped translate it into German / Deutsche Übersetzung:  
[lettersforblacklives.com/liebe-eltern-l...](https://lettersforblacklives.com/liebe-eltern-l...)

**Liebe Eltern, liebe Verwandte, das Leben der Schwarzen ist...**  
This is the German version of the open letter created by Letters...  
[lettersforblacklives.com](https://lettersforblacklives.com)

5h

Martin Kleppmann Retweeted



**Emin Gün Sirer** @el33th4xor

Nothing infuriates me as much as trying to capture a complex dist sys trade-off as a dumbass triangle that isn't even correct.

10 Jul

Martin Kleppmann Retweeted



**Raul C Fernandez** @raulcfernandez

Cool, we showed similar results in our ATC14 paper. It's the consequence of using batch proc. to do streaming  
[twitter.com/martinkl/statu...](https://twitter.com/martinkl/statu...)

07 Jul

[Embed](#)

[View on Twitter](#)

## Recent posts

- › 15 Apr 2016: [Announcing TRVE DATA: Placing a bit less trust in the cloud](#)
- › 30 Mar 2016: [Device security and the FBI](#)
- › 18 Feb 2016: [Should law enforcement services have a backdoor into smartphones?](#)
- › 08 Feb 2016: [How to do distributed locking](#)
- › 28 Dec 2015: [My year 2015 in review](#)
- › [Full archive](#)

## Conference talks

- › 07 Mar 2016 at QCon
- › 04 Mar 2016 at University of Cambridge
- › 29 Jan 2016 at Domain Driven Design Europe
- › 13 Nov 2015 at All Your Base
- › 06 Nov 2015 at Øredev
- › [Full archive](#)



Unless otherwise specified, all content on this site is licensed under a [Creative Commons Attribution 3.0 Unported License](#). Theme borrowed from [Carrington](#), ported to [Jekyll](#) by Martin Kleppmann.