A plain english introduction to CAP Theorem

You'll often hear about the CAP theorem which specifies some kind of an upper limit when designing distributed systems. As with most of my other introduction tutorials, lets try understanding CAP by comparing it with a real world situation.

Chapter 1: "Remembrance Inc" Your new venture :

Last night when your spouse appreciated you on remembering her birthday and bringing her a gift, a strange Idea strikes you. People are so bad in remembering things. And you're sooo good at it. So why not start a venture that will put your talent to use? The more you think about it, the more you like it. In fact you even come up with a news paper ad which explains your idea

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 eg., 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 $0.1 per request

So, your typical phone conversation will look like this:

- Customer : Hey, Can you store my neighbor's birthday?
- You: Sure.. when is it?
- Customer : 2nd of jan
- You: (write it down against the customer's page in your paper note book) Stored. Call us any time for knowing your neighbor's birthday again!
- Customer : Thank you!
- You: No problem! We charged your credit card with $0.1

Chapter 2 : You scale up:

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

And there starts the problem. 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 tone. 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 up and bring in your wife to help you.

Your start with a simple plan:

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

Chapter 3 : You have your first “Bad Service” :

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

- Jhon: Hey
- You: Glad you called “Remembrance Inc!”. What can I do for you?
- Jhon: Can you tell me when is my flight to New Delhi?
- You: Sure.. 1 sec sir
  (You look up your notebook)
  (wow! there is no entry for “flight date” in Jhon’s page)!!!!!
- You: Sir, I think there is a mistake. You never told us about your flight to delhi
- Jhon: What! I just called you guys yesterday!(cuts the call!)

How did that happen? Could Jhon be lying? You think about it for a second and the reason hits you! Could Jhon’s call yesterday reached your wife? You go to your wife’s desk and check her notebook. Sure enough it’s 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!**

Chapter 4: You fix the Consistency problem:

Well, your competitors may ignore a bad service, but not you. You think all night in the bed when your wife is sleeping and come up with a beautiful plan in the morning. You wake up your wife and tell her:

” Darling this is what we are going to do from now”

- Whenever any one of us get a call for an update(when the customer wants us to remember something) before completing the call we tell the other person
- This way both of us note down any updates
- When there is call for search(When the customer wants information he has already stored) we don’t need to talk with the other person. Since both of us have the latest updated information in both of our note books we can just refer to it..

There is only one problem though, you say, and that is an “update” request has to involve both of us and we cannot work in parallel during that time. For eg. 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 **Availability problem , i.e, for eg., 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!”**

Chapter 5: You come up with the greatest solution Ever:

You being to realize a little bit on why 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”**? Could be difficult for others, but not for you!! Then next morning you come up with a solution that your competitors cannot think of in their dreams! You wake your wife up eagerly again..

” look” , you tell her.. “This is what we can do to be consistent and available” . The plan is mostly similar to what I told you yesterday:

- i) Whenever any one of us get a call for an update(when the customer wants us to remember something) before completing the call, if the other person is available we tell the other person. This way both of us note down any updates
- ii) But if the other person is not available(doesn't report to work) we send the other person an email about the update.
- iii) 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! You 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!**

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

Chapter 7: Conclusion :

So Let's look at CAP Theorem now. Its states that, when you are designing a distributed system you can get cannot achieve all three of Consistency, Availability and Partition tolerance. You can pick only two of:

- Consistency: You 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 until any one of you(you or your wife) report to work.
- Partition Tolerance: Remembrance Inc will work even if there is a communication loss between you and your wife!

Bonus : Eventual Consistency with a run around clerk :

Here is another food for thought. You can have a run around clerk, who will update other’s notebook when one of your’s or your wife’s note books is updated. The greatest benefit of this is that, he can work in 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 lose consistency of some time. For eg., 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.. But that said, this is not at all a bad idea if such cases are limited. For eg., assuming a customer won’t forget things so quickly that he calls back in 5 minutes.

That’s CAP and eventual consistency for you in simple english :)