

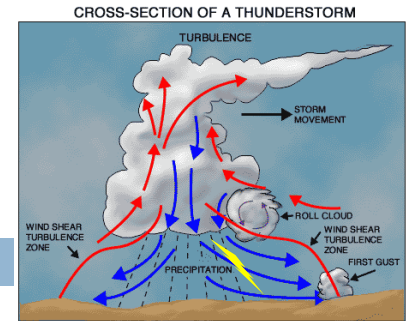
# CS5412: ANATOMY OF A CLOUD

Lecture VIII

Ken Birman

# How are cloud structured?

2

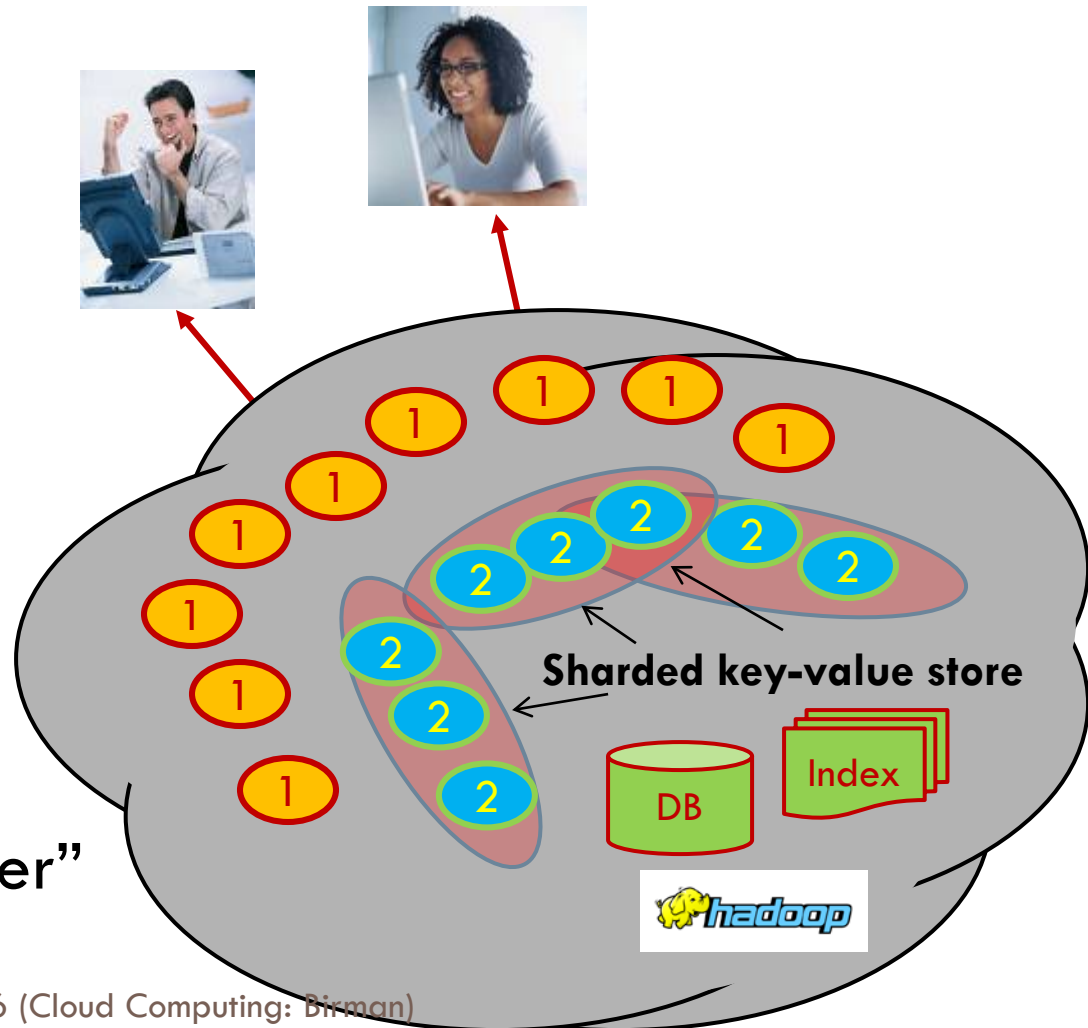


- Clients talk to clouds using web browsers or the web services standards
  - But this only gets us to the outer “skin” of the cloud data center, not the interior
  - Consider Amazon: it can host entire company web sites (like Target.com or Netflix.com), data (AC3), servers (EC2) and even user-provided virtual machines!

# Big picture overview

3

- Client requests are handled in the “first tier” by
  - ▣ PHP or ASP pages
  - ▣ Associated logic
- These lightweight services are fast and very nimble
- Much use of caching: a sharded key-value store in the “second tier”



# Many styles of system

4

- Near the edge of the cloud focus is on vast numbers of clients and rapid response
- Inside we find high volume services that operate in a pipelined manner, asynchronously
- Deep inside the cloud we see a world of virtual computer clusters that are scheduled to share resources and on which applications like MapReduce (Hadoop) are very popular

# Asynchronous pipeline model

5

- Outside layer of the cloud absorbs read accesses, but updates are tricky
- Popular model: Guess at the update outcome and respond using that, but send the “real” update asynchronously to a back-end server / database.
- Later, correct inconsistencies



# In the outer tiers replication is key

6

- We need to replicate
  - ▣ Processing: each client has what seems to be a private, dedicated server (for a little while)
  - ▣ Data: as much as possible, that server has copies of the data it needs to respond to client requests without any delay at all
  - ▣ Control information: the entire structure is managed in an agreed-upon way by a decentralized cloud management infrastructure

# Shared key-value store?

7



- The caching components running in tier two are central to the responsiveness of tier-one services
  - ▣ Basic idea is to always use cached data if at all possible, so the inner services (here, a database and a search index stored in a set of files) are shielded from “online” load
  - ▣ We need to replicate data within our cache to spread loads and provide fault-tolerance
  - ▣ But not everything needs to be “fully” replicated. Hence we often use “shards” with just a few replicas

# Sharding used in many ways

8

- The second tier could be any of a number of caching services:
  - ▣ Memcached: a sharable in-memory key-value store
  - ▣ Other kinds of DHTs that use key-value APIs
  - ▣ Dynamo: A service created by Amazon as a scalable way to represent the shopping cart and similar data
  - ▣ BigTable: A very elaborate key-value store created by Google and used not just in tier-two but throughout their “GooglePlex” for sharing information
- Notion of sharding is cross-cutting
  - ▣ Most of these systems replicate data to some degree
  - ▣ Achieves high availability but also increases performance



# Do we *always* need to shard data?

9

- Imagine a tier-one service running on 100k nodes
  - ▣ Can it ever make sense to replicate data on the entire set?
- Yes, if some kinds of information might be so valuable that almost every external request touches it.
  - ▣ Must think hard about patterns of data access and use
  - ▣ Some information needs to be heavily replicated to offer blindingly fast access on vast numbers of nodes
  - ▣ The principle is similar to the way Beehive operates.
    - Even if we don't make a dynamic decision about the level of replication required, the principle is similar
    - We want the level of replication to match level of load and the degree to which the data is needed on the critical path

# And it isn't just about updates

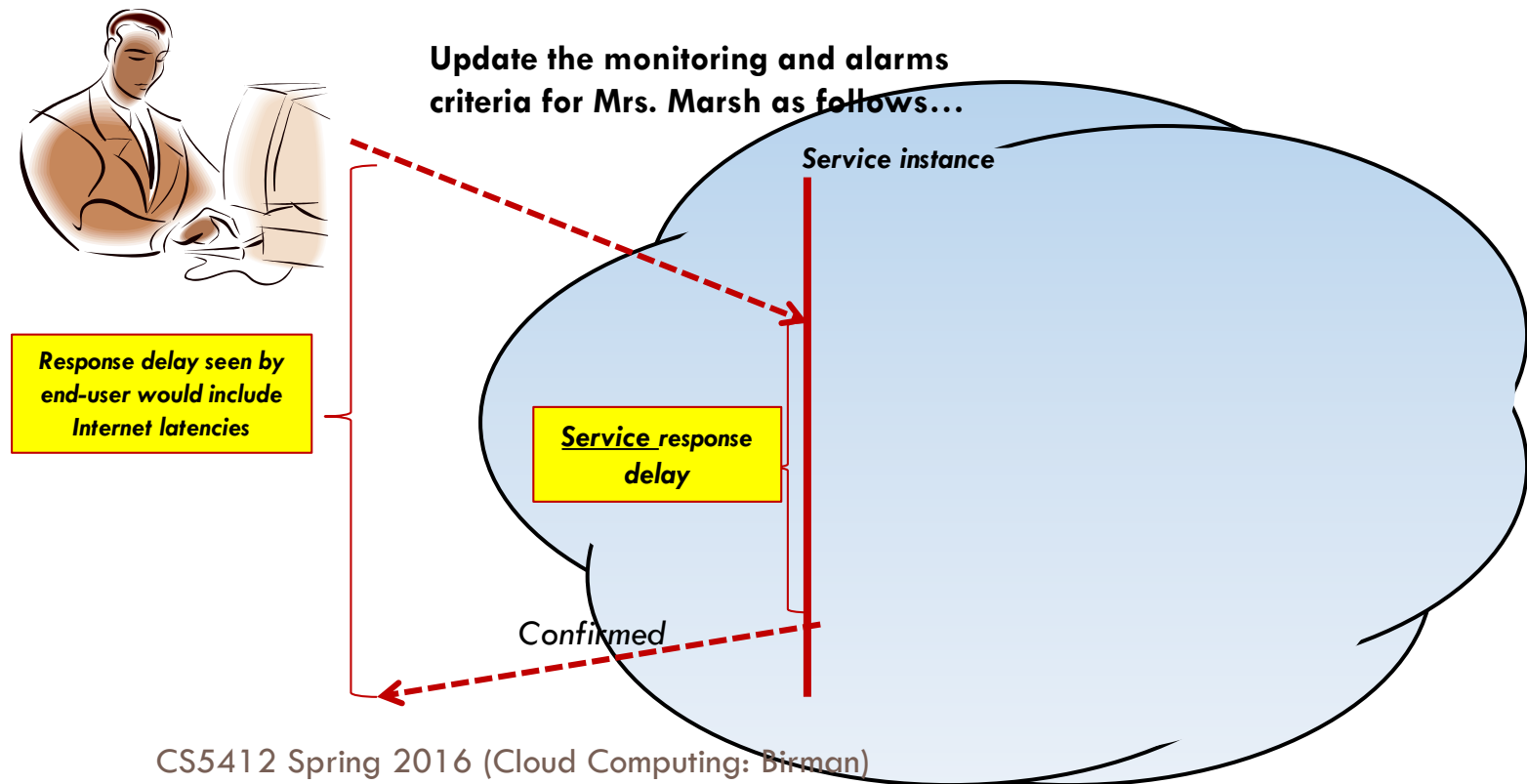
10

- Should also be thinking about patterns that arise when doing reads (“queries”)
  - ▣ Some can just be performed by a single representative of a service
  - ▣ But others might need the parallelism of having several (or even a huge number) of machines do parts of the work concurrently
- The term sharding is used for data, but here we might talk about “parallel computation on a shard”

# What does “critical path” mean?

11

- Focus on delay until a client receives a reply
- Critical path are actions that contribute to this delay



# What if a request triggers updates?

12

- If the updates are done “asynchronously” we might not experience much delay on the critical path
  - ▣ Cloud systems often work this way
  - ▣ Avoids waiting for slow services to process the updates but may force the tier-one service to “guess” the outcome
  - ▣ For example, could optimistically apply update to value from a cache and just hope this was the right answer
- Many cloud systems use these sorts of “tricks” to speed up response time

# First-tier parallelism

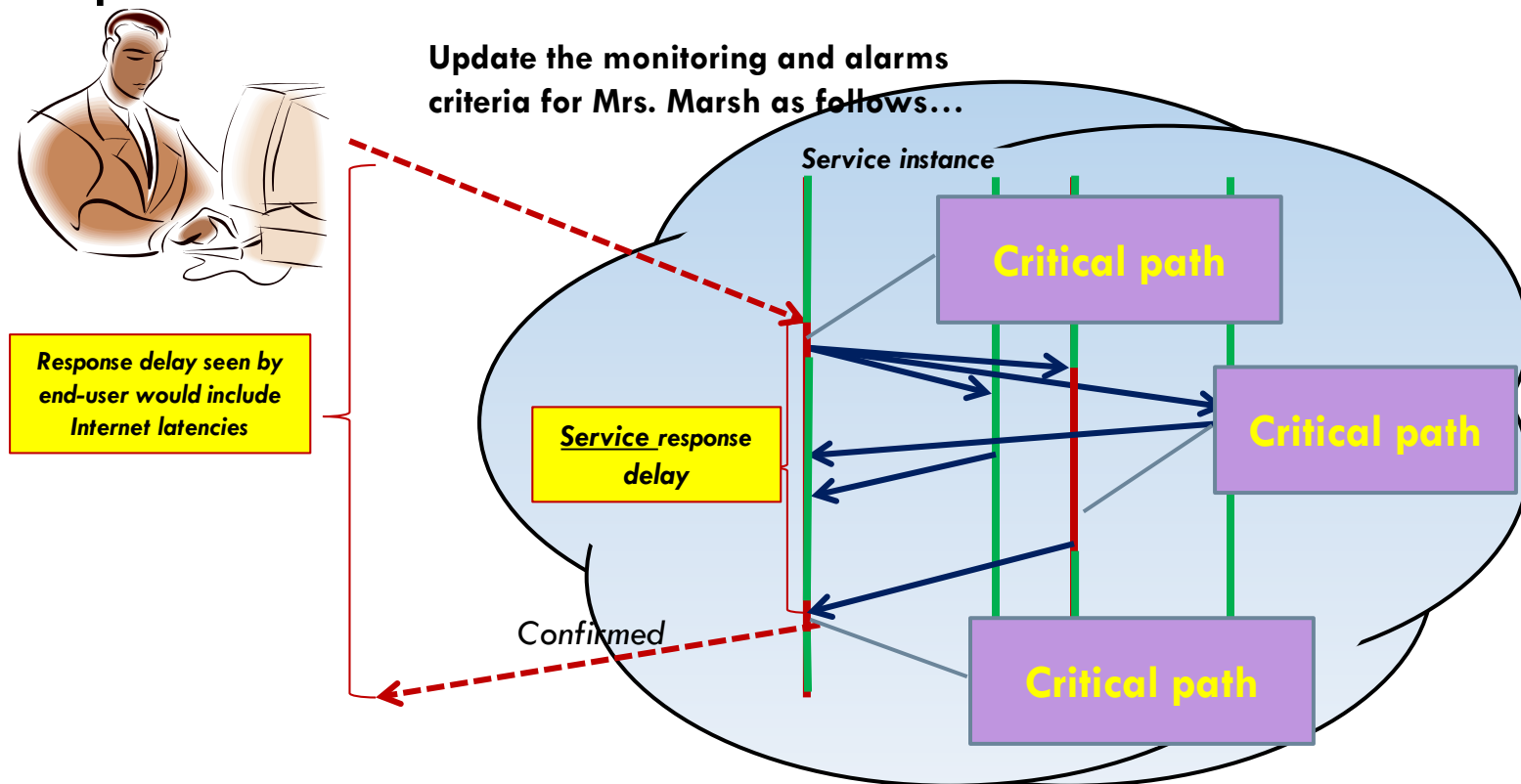
13

- Parallelism is vital to speeding up first-tier services
- Key question:
  - ▣ Request has reached some service instance X
  - ▣ Will it be faster...
    - ... For X to just compute the response
    - ... Or for X to subdivide the work by asking subservices to do parts of the job?
- Glimpse of an answer
  - ▣ Werner Vogels, CTO at Amazon, commented in one talk that many Amazon pages have content from 50 or more parallel subservices that ran, in real-time, on your request!

# What does “critical path” mean?

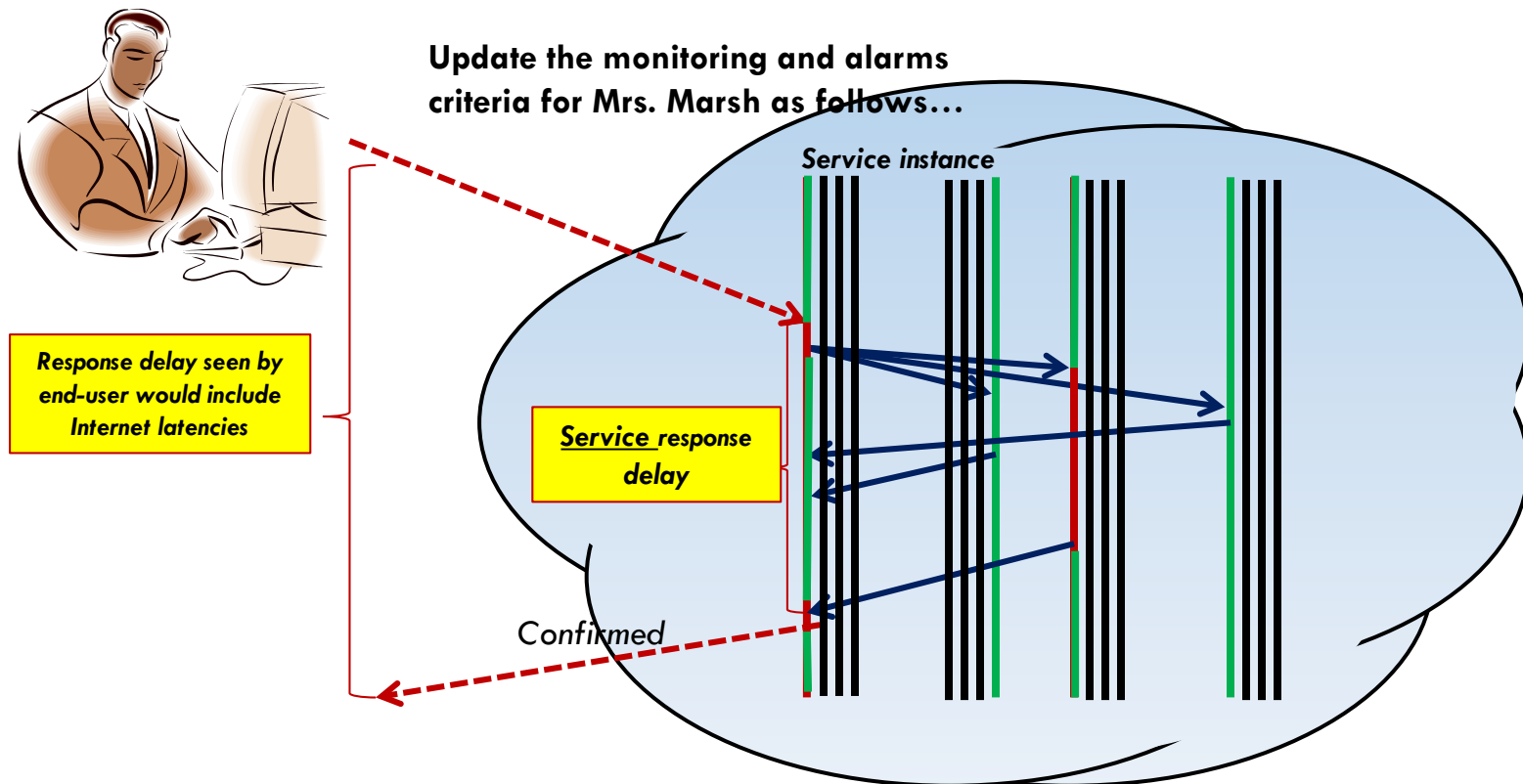
14

- In this example of a parallel read-only request, the critical path centers on the middle “subservice”



# With replicas we just load balance

15



# But when we add updates....

16

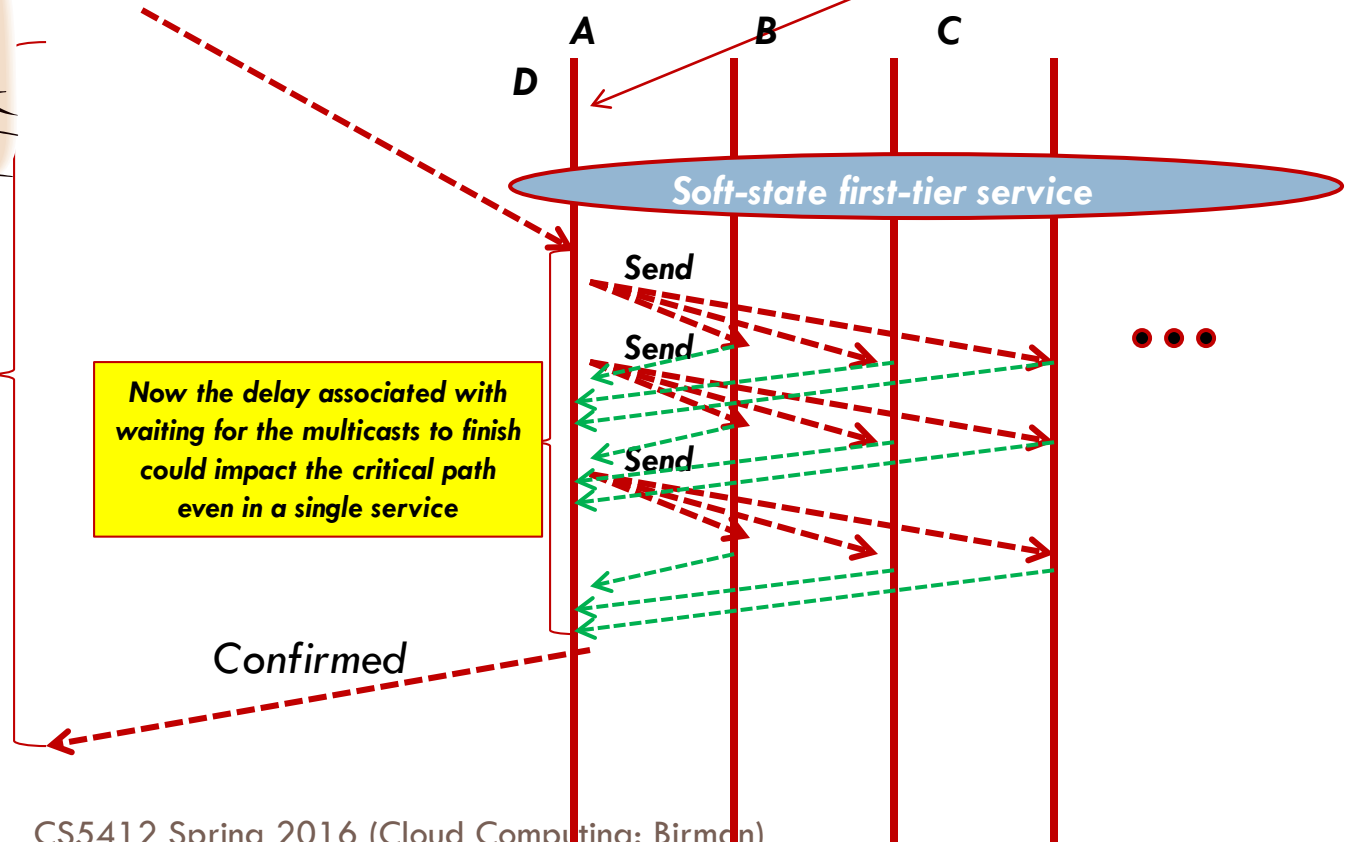
**Update the monitoring and alarms criteria for Mrs. Marsh as follows...**



Response delay seen by end-user would also include Internet latencies not measured in our work

Now the delay associated with waiting for the multicasts to finish could impact the critical path even in a single service

Execution timeline for an individual first-tier replica



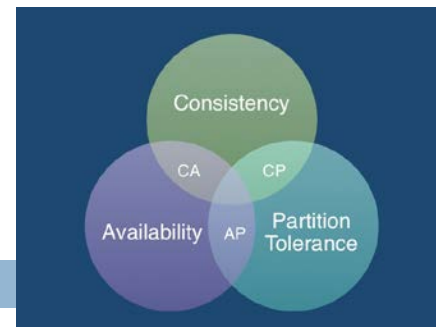


# What if we send updates without waiting?

17

- Several issues now arise
  - ▣ Are all the replicas applying updates in the same order?
    - Might not matter unless the same data item is being changed
    - But then clearly we do need some “agreement” on order
  - ▣ What if the leader replies to the end user but then crashes and it turns out that the updates were lost in the network?
    - Data center networks are surprisingly lossy at times
    - Also, bursts of updates can queue up
- Such issues result in *inconsistency*

# Eric Brewer's CAP theorem



18

- In a famous 2000 keynote talk at ACM PODC, Eric Brewer proposed that “you can have just two from 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

# CAP theorem

19

- A proof of CAP was later introduced by MIT's Seth Gilbert and Nancy Lynch
  - ▣ 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

# Is inconsistency a bad thing?

20

- How much consistency is really needed in the first tier 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?



# THE WISDOM OF THE SAGES

# eBay's Five Commandments

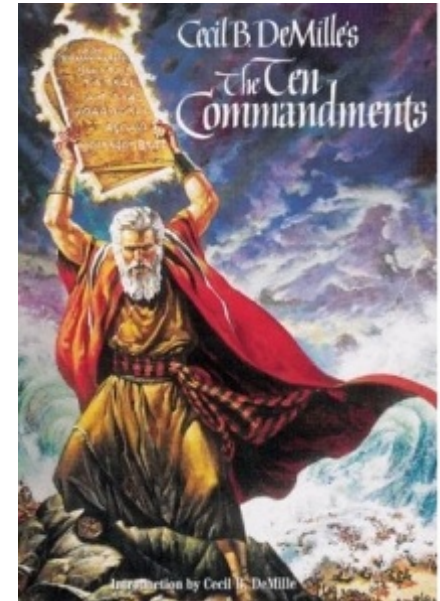


22

- 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

23



- 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



24

- 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

25



**Consistency technologies  
just don't scale!**



# But inconsistency brings risks too!

26

My rent check bounced?  
That can't be right!

- Inconsistency causes bugs
  - ▣ Clients would never be able to trust servers... a free-for-all



- Weak or “best effort” consistency?
  - ▣ Strong security guarantees demand consistency
  - ▣ Would you trust a medical electronic-health records system or a bank that used “weak consistency” for better scalability?

# Puzzle: Is CAP valid in the cloud?

27

- Facts: data center networks don't normally experience partitioning failures
  - ▣ Wide-area links do fail
  - ▣ But most services are designed to do updates in a single place and mirror read-only data at others
  - ▣ So the CAP scenario used in the proof can't arise
- Brewer's argument about not waiting for a slow service to respond does make sense
  - ▣ Argues for using any single replica you can find
  - ▣ But does this preclude that replica being consistent?

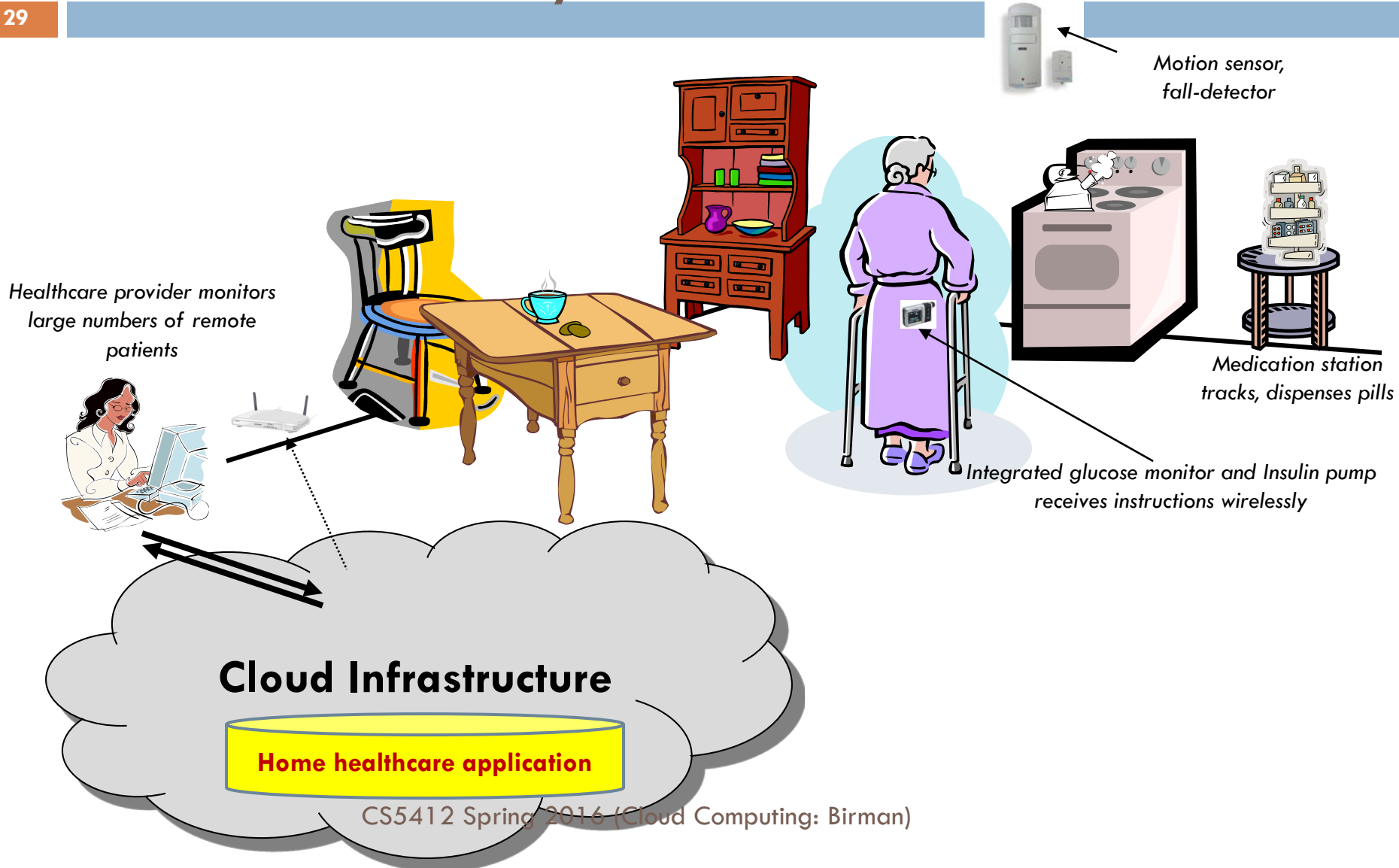
# What does “consistency” mean?

28

- We need to pin this basic issue down!
- As used in CAP, consistency is about two things
  - ▣ First, that updates to the same data item are applied in some agreed-upon order
  - ▣ Second, that once an update is acknowledged to an external user, it won't be forgotten
- Not all systems need both properties

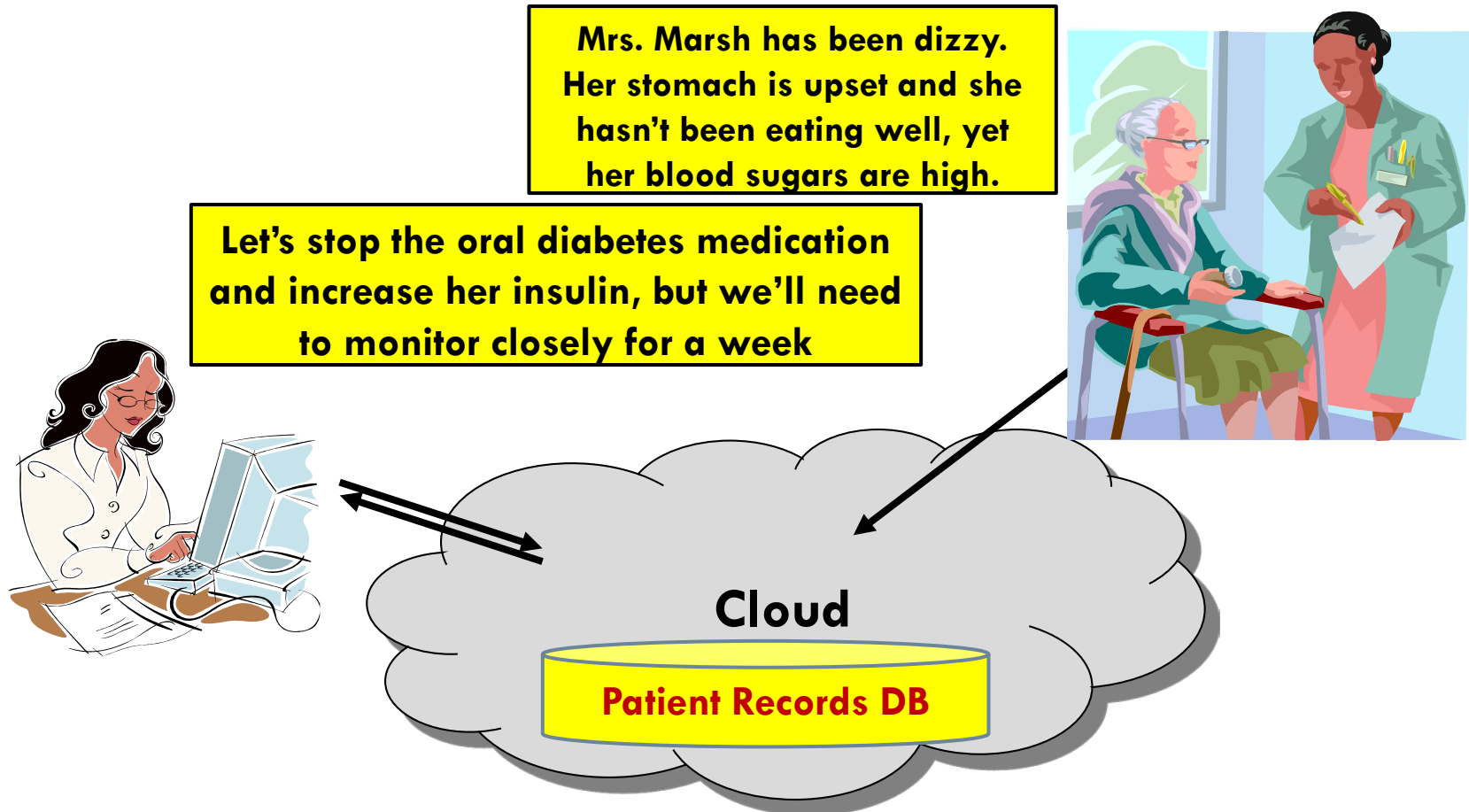
# What properties are needed in remote medical care systems?

29



# Which matters more: fast response, or durability of the data being updated?

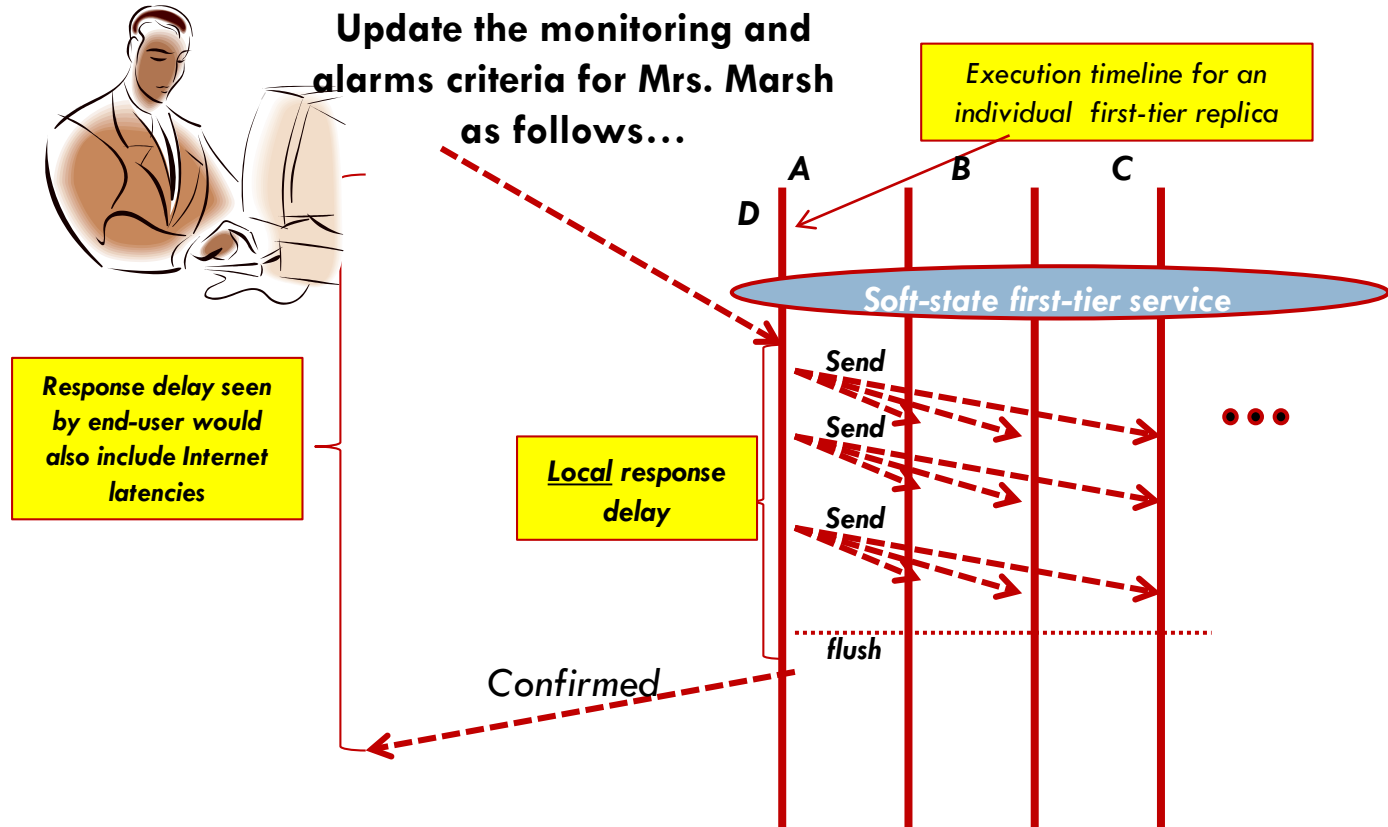
30



□ Need: Strong consistency and durability for data

# What if we were doing online monitoring?

31



- An online monitoring system might need 1 ms response times. It would value consistency, yet be less concerned with durability

# Cloud services and their properties

32

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



# Core problem?



33

- Our challenge is a form of “role-driven” service in which the properties of the service are shaped by the way we plan to use it
  - ▣ One application might have multiple such services in it, like the medical example, which has a monitoring service with real-time roles and a durable database.
  - ▣ We don’t want to sweep important properties away, but we sometimes have to make tradeoffs

# Where does this all lead us?

34

- High assurance cloud computing is feasible!
  - ▣ Experts already doing it in a plethora of services
  - ▣ The main obstacle is that typical application developers tend to use one-size-fits-all packages with one-size-fits-all properties and guarantees.
- As we develop better tools and migrate them to the cloud platforms developers use, options will improve
- We'll see that these really are solvable problems!