Peer-to-Peer in the Datacenter: Amazon Dynamo

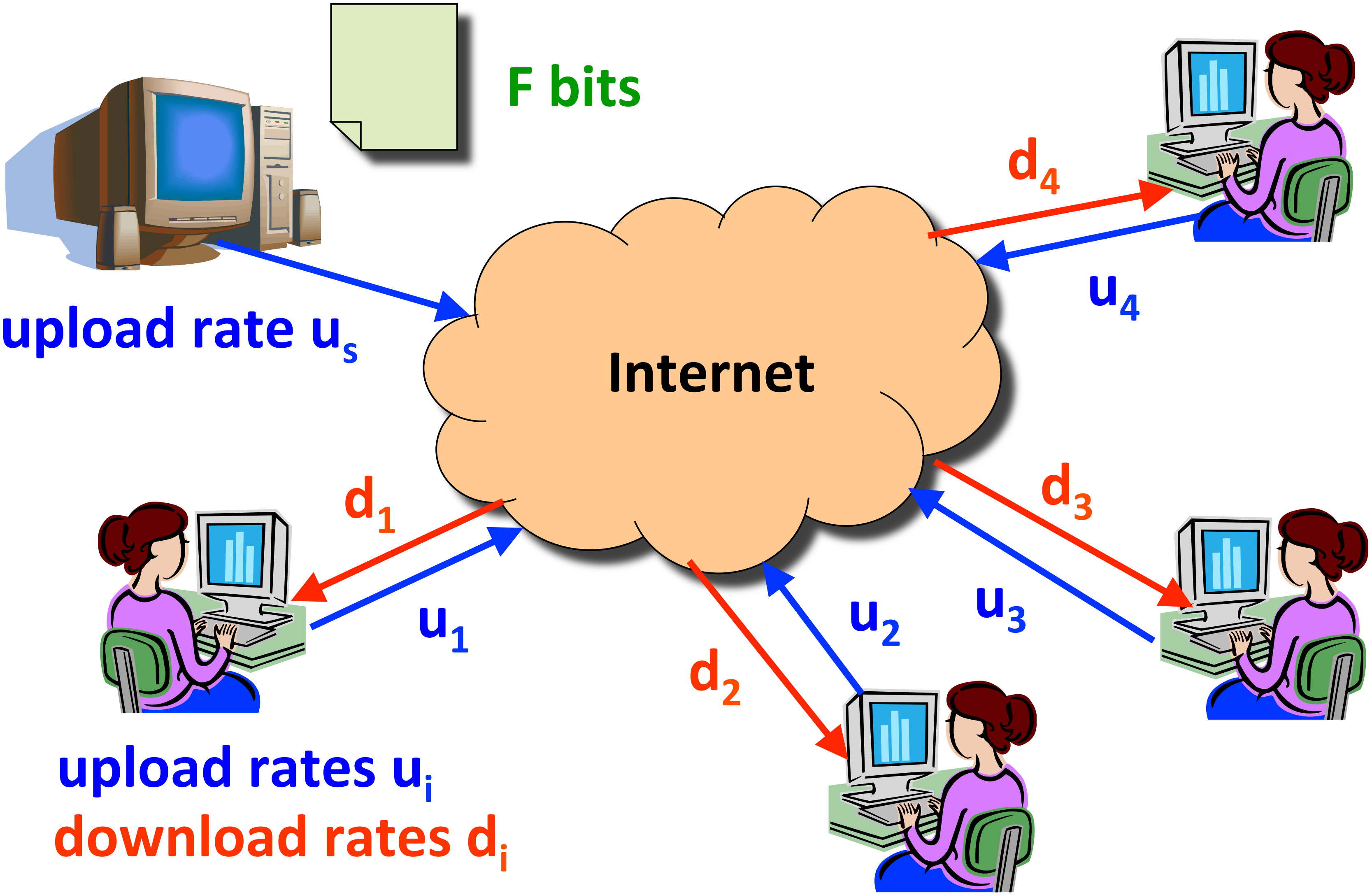
Aaron Blankstein

COS 461: Computer Networks

Lectures: MW 10-10:50am in Architecture N101

http://www.cs.princeton.edu/courses/archive/spr13/cos461/

Last Lecture…



upload rates u_i

download rates d_i

This Lecture…

Amazon’s “Big Data” Problem

- Too many (paying) users!
  - Lots of data
- Performance matters
  - Higher latency = lower “conversion rate”
- Scalability: retaining performance when large

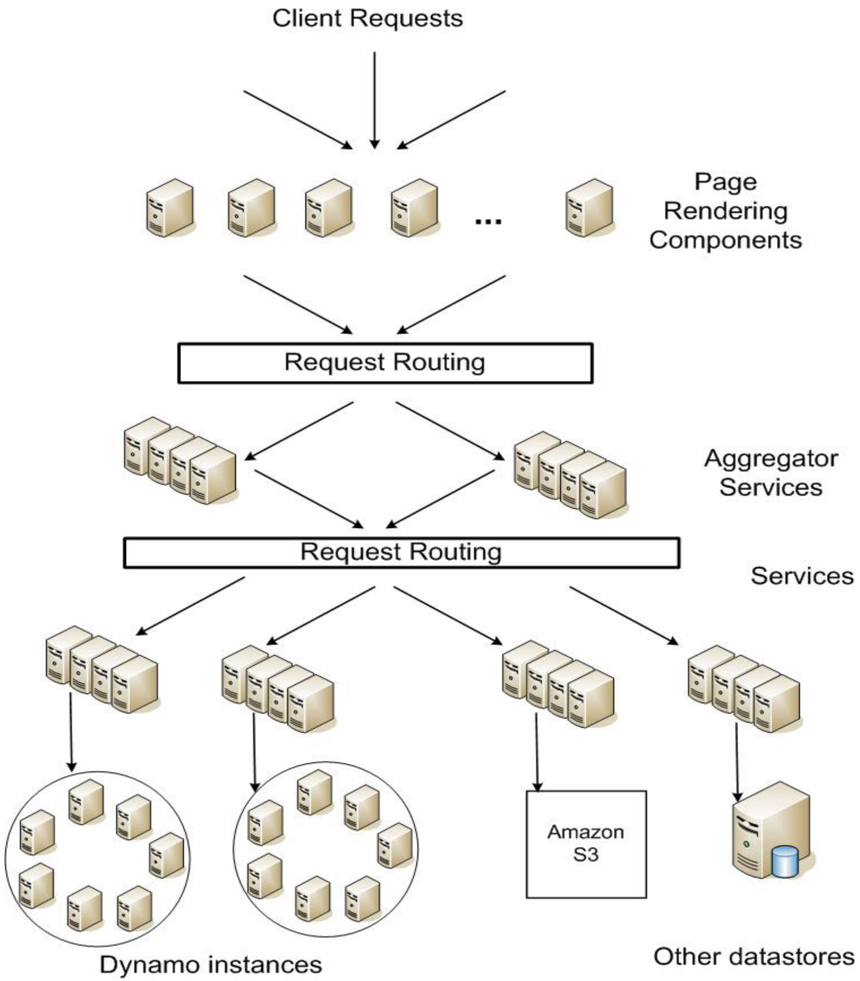
Tiered Service Structure

Stateless

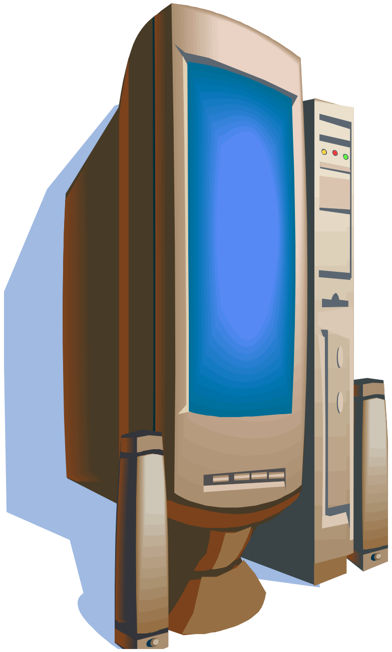
Stateless

Stateless

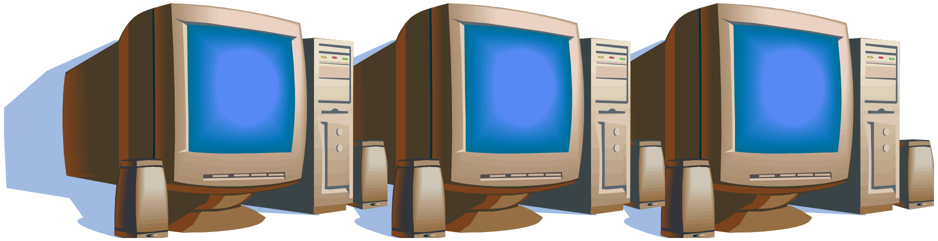
All of the State



Horizontal or Vertical Scalability?



Vertical Scaling



Horizontal Scaling

Horizontal Scaling Chaos

- Horizontal scaling is chaotic*
- Failure Rates:
  - – k = probability a machine fails in given period
  - – n = number of machines
  - – $1-(1-k)^n$ = probability of any failure in given period
  - – For 50K machines, with online time of 99.99966%:
    - 16% of the time, data center experiences failures
    - For 100K machines, 30% of the time!

Dynamo Requirements

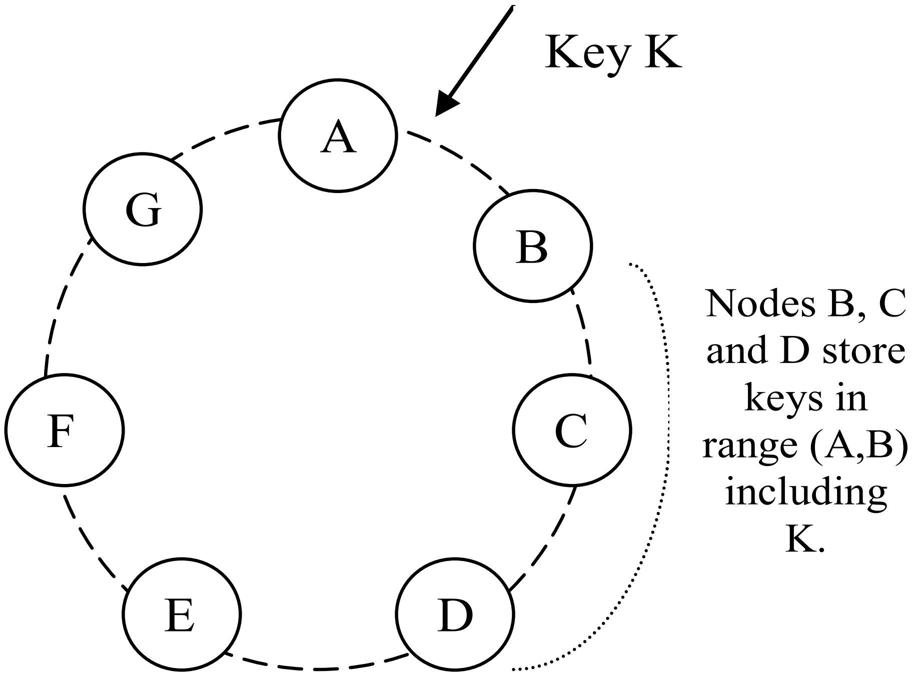
- High Availability
  - Always respond quickly, even during failures
  - *Replication!*
- Incremental Scalability
  - Adding "nodes" should be seamless
- Comprehensible Conflict Resolution
  - High availability in above sense implies conflicts

Dynamo Design

- Key-Value Store ➔ DHT over data nodes
  - get(k) and put(k, v)
- Questions:
  - Replication of Data
  - Handling Requests in Replicated System
  - Temporary and Permanent Failures
  - Membership Changes

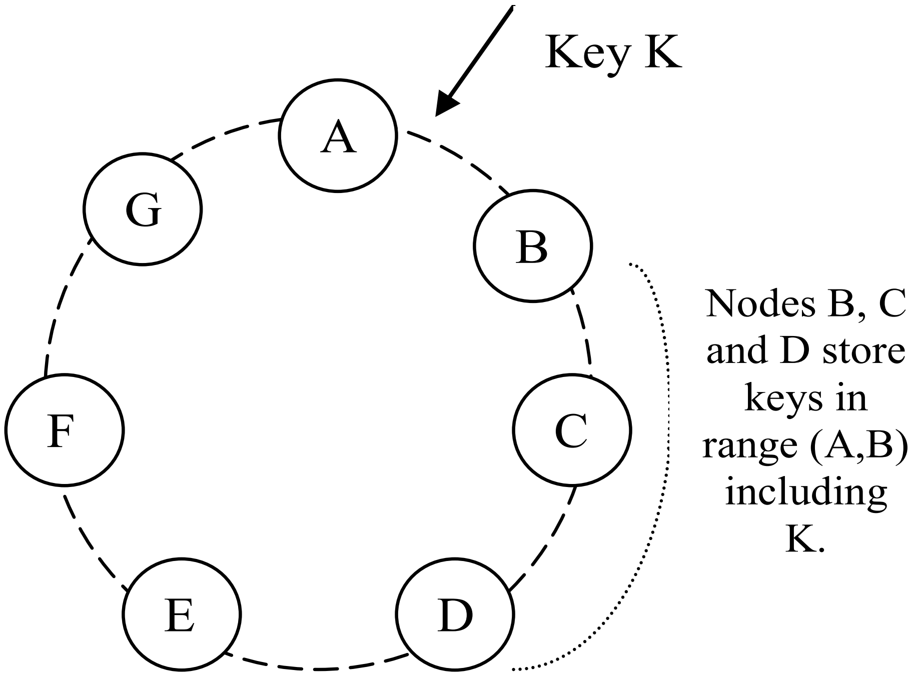
Data Partitioning and Data Replication

- Familiar?
- Nodes are virtual!
  - Heterogeneity
- Replication:
  - Coordinator Node
  - *N-1* successors also
  - Nodes keep preference list



Handling Requests

- Requests handled by coordinator
  - Consults replicas
- Forward request to *N* replicas from pref. list
  - *R* or *W* responses form a quorum
- For load balancing/ failures, any of the *top N* in the pref. list can handle request

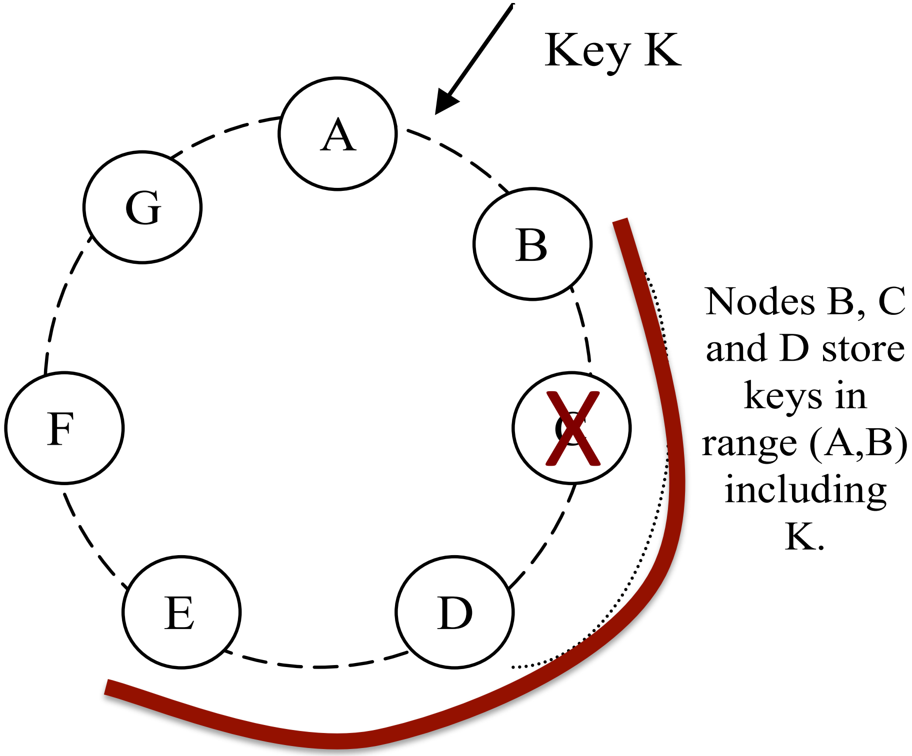


Detecting Failures

- *Purely* Local Decision
    - – Node A may decide independently that B has failed
    - – In response, requests go further in the pref. list
- A request hits an unsuspecting node
    - – "temporary failure" handling occur

Handling Temporary Failures

- E is in replica set
  - Needs to receive the replica
  - Hinted Handoff: replica contains “original” node
- When C comes back
  - E forwards the replica back to C



Add E to the replica set!

Managing Membership

- Peers randomly tell another their known membership history – "gossiping"
- Also called epidemic algorithm
  - Knowledge spreads like a disease through system
  - Great for ad hoc systems, self-configuration, etc.
  - Does this make sense in Amazon's environment?

Gossip could partition the ring

- Possible Logical Partitions
  - A and B choose to join ring at about the same time: unaware of one another, may take long time to converge to one another
- Solution:
  - Use *seed* nodes to reconcile membership views: well-known peers which are contacted more frequently

Why is Dynamo Different?

- So far, looks a lot like normal p2p
- Amazon wants to use this for application data!
- Lots of potential synchronization problems
- Dynamo uses versioning to provide *eventual consistency.*

Consistency Problems

- Shopping Cart Example:
  - Object is a history of “adds” and “removes”
  - *All adds* are important (trying to make money)

Client:	Expected Data at Server:
Put(k, [+1 Banana])	[+1 Banana]
Z = get(k)	[+1 Banana, +1 Banana]
Put(k, Z + [+1 Banana])	[+1 Banana, +1 Banana, -1 Banana]
Z = get(k)	
Put(k, Z + [-1 Banana])	

What if a failure occurs?

Client:	Data on Dynamo:	
Put(k, [+1 Banana])	[+1 Banana]	at A
Z = get(k)	*A Crashes*	
Put(k, Z + [+1 Banana])	*B* ***not*** *in first Put's quorum*	
Z = get(k)	[+1 Banana]	at B
Put(k, Z + [-1 Banana])	[+1 Banana, -1 Banana]	at B
	Node A Comes Online	

At this point, Node A and B disagree about the current state of the object – how is that resolved? Can we even tell that there is a conflict?

"Time" is largely a human construct

- What about time-stamping objects?
  - We could authoritatively say whether an object is newer or older…
  - *all events are not necessarily witnessed*
- If our system's notion of time corresponds to "real-time"…
  - A new object always blasts away older versions, even though those versions may have important updates (as in bananas example).
- Requires a new notion of time (causal in nature)
- Anyhow, real-time is impossible in any case

Causality

- Objects are causally related if the value of one object depends on (or witnessed) the previous
- Conflicts can be detected when replicas contain causally independent objects for a given key.
- Can we have a notion of time which captures causality?

Versioning

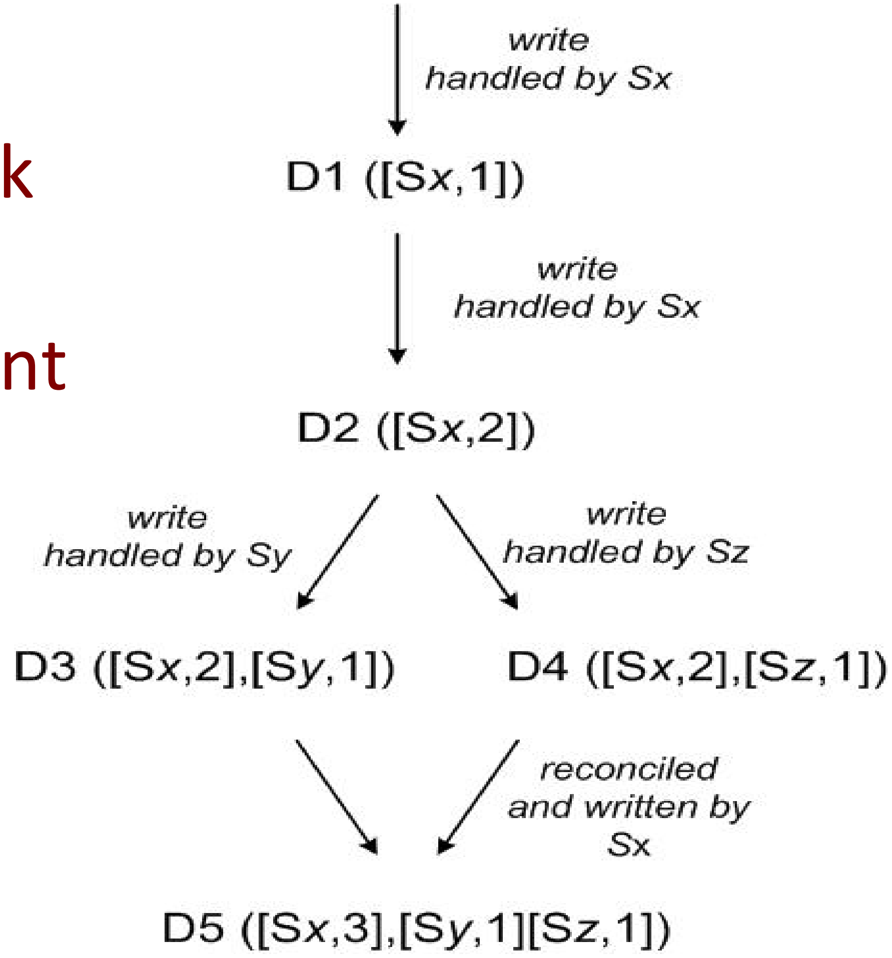
- Key Idea: every PUT includes a version, indicating the most recently witnessed version of the object being updated
- Problem: replicas may have diverged
  - No single authoritative version number (or “clock” number)
  - Notion of time must use a *partial ordering* of events

Vector Clocks

- Every replica has its own logical clock
  - Incremented before it sends a message
- Every message attached with *vector* version
  - Includes originator's clock
  - Highest seen logical clocks for each replica
- If M_1 is causally dependent on M_0:
  - Replica sending M_1 will have seen M_0
  - Replica will have seen clocks ≥ all clocks in M_0

Vector Clocks in Dynamo

- Vector clock per object
- Gets() return vector clock of object
- Puts() contain most recent vector clock
  - – Coordinator treated as “originator”
- Serious conflicts are resolved by the application / client



Vector Clocks in Banana Example

Client:

Put(k, [+1 Banana])
Z = get(k)
Put(k, Z + [+1 Banana])
Z = get(k)
Put(k, Z + [-1 Banana])

Data on Dynamo:

[+1] v=[(A,1)] at A
A Crashes
B ***not*** *in first Put's quorum*
[+1] v=[(B,1)] at B
[+1,-1] v=[(B,2)] at B
Node A Comes Online

[(A,1)] and [(B,2)] are a conflict!

Eventual Consistency

- Versioning, by itself, does not guarantee consistency
  - If you don't require a majority quorum, you need to periodically check that peers aren't in conflict
  - How often do you check that events are not in conflict?
- In Dynamo
  - Nodes consult with one another using a tree hashing (Merkel tree) scheme
  - Allows them to quickly identify whether they hold different versions of particular objects and enter conflict resolution mode

NoSQL

- Notice that Eventual Consistency, Partial Ordering do not give you ACID!
- Rise of NoSQL (outside of academia)
  - Memcache
  - Cassandra
  - Redis
  - Big Table
  - Neo4J
  - MongoDB