: Managing Multi-Data Center Data

Amr El Abbadi
Computer Science, UC Santa Barbara
amr@cs.ucsb.edu

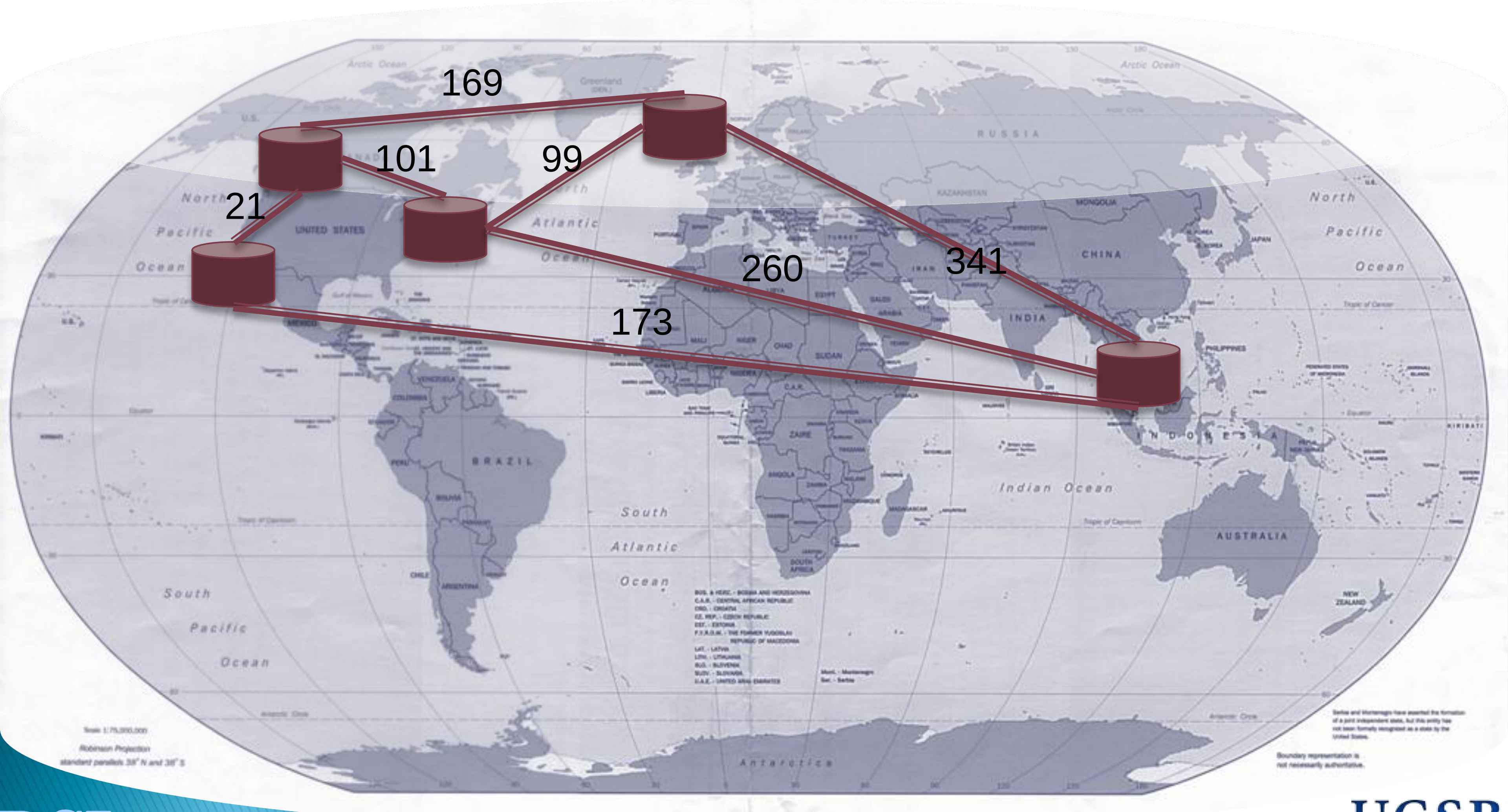
Collaborators: Divy Agrawal, Sudipto Das, Aaron Elmore, Hatem Mahmoud, Faisal Nawab, and Stacy Patterson.

DSL

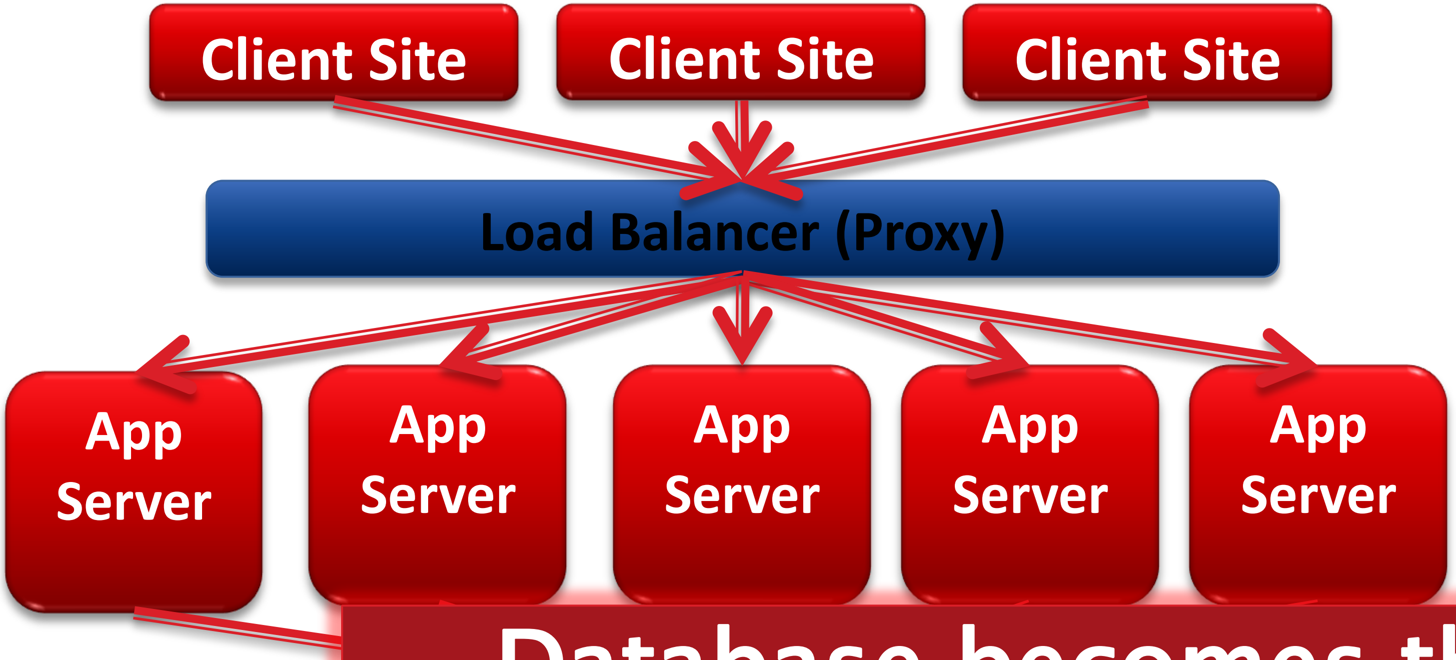
UCSB

Cloud Reality: The Data Centers

Round Trip Times (RTT)

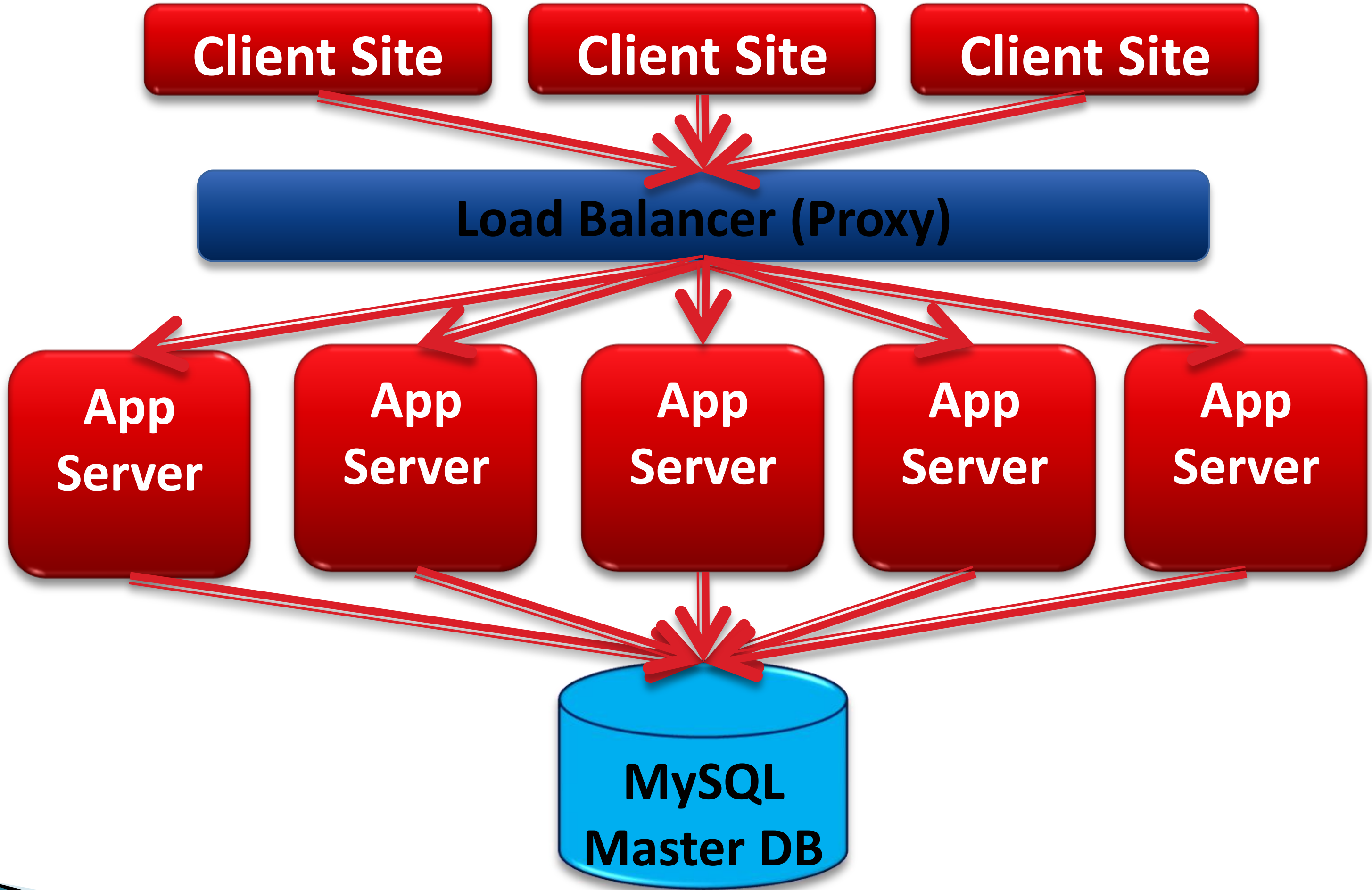


Scaling in the Cloud

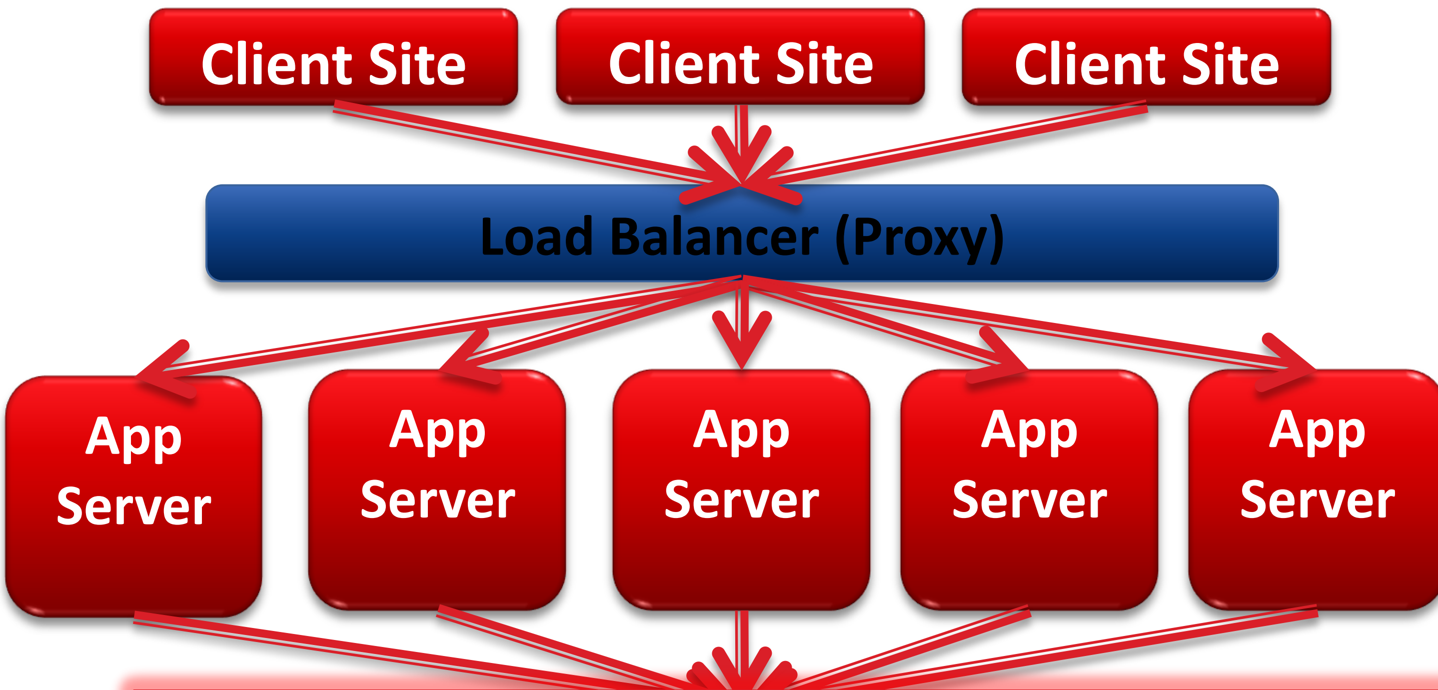


Database becomes the Scalability Bottleneck
Cannot leverage elasticity

Scaling in the Cloud



Scaling in the Cloud



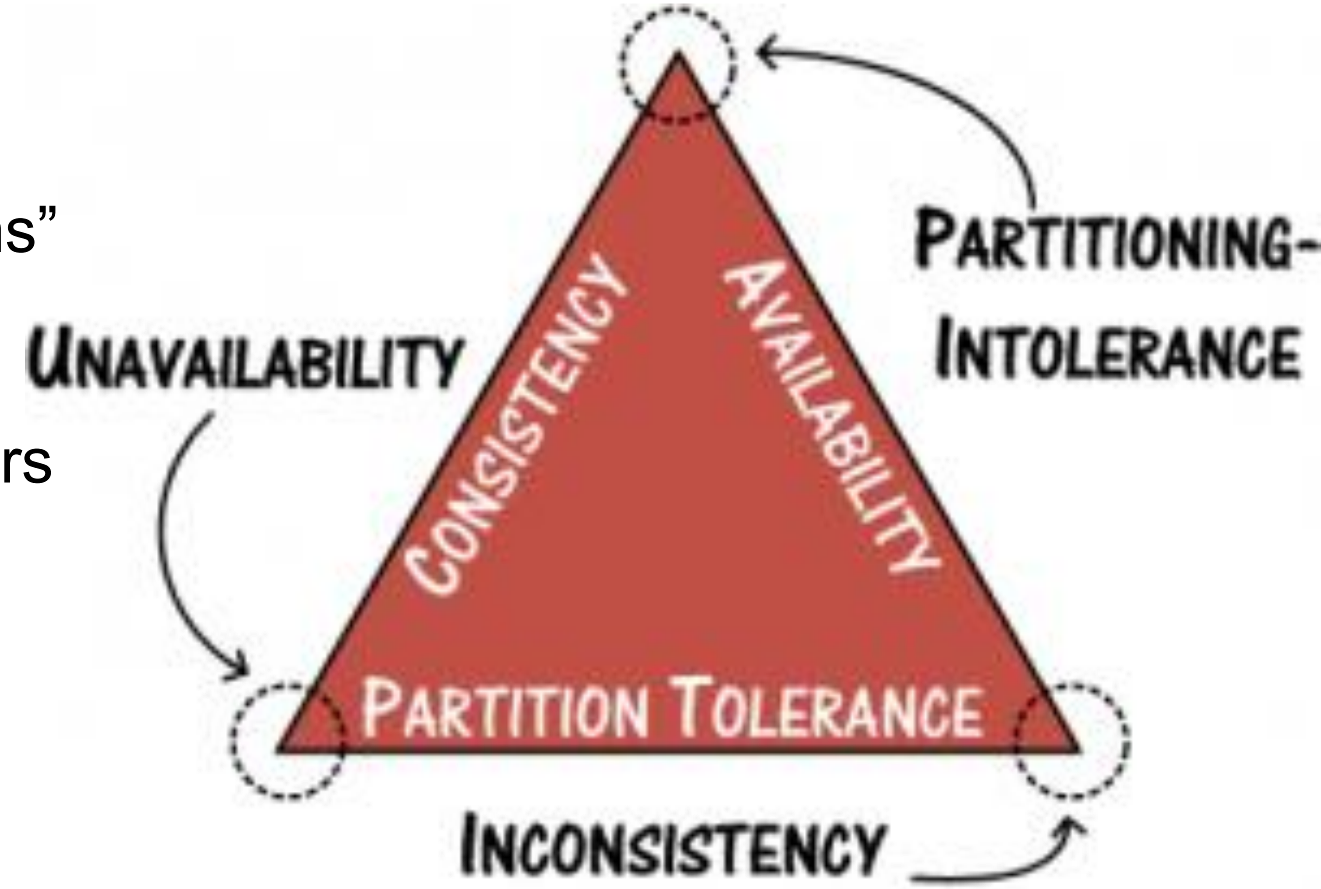
**Scalable and Elastic,
but limited consistency and
operational flexibility**

{name: "mongo", type:"DB"}

CAP Theorem (Eric Brewer)

- “Towards Robust Distributed Systems” PODC 2000.

- “CAP Twelve Years Later: How the "Rules" Have Changed” IEEE Computer 2012

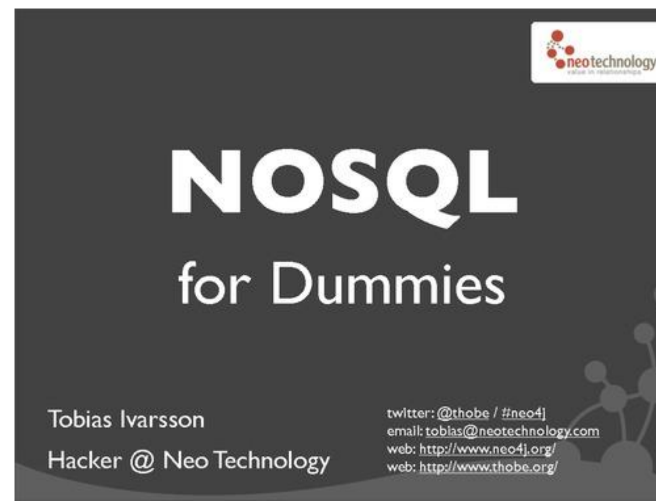



SQL
Just
say no


SQL


i★
NoSQL


SQL

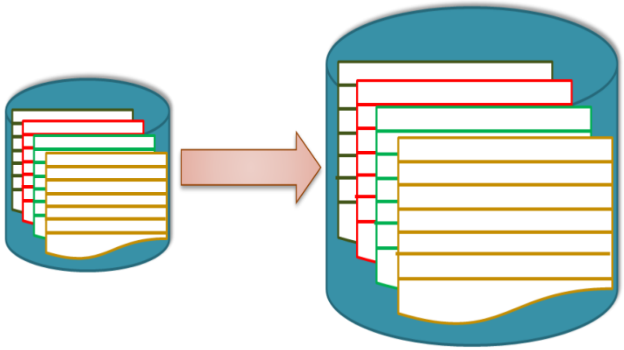

neotechnology
NOSQL
for Dummies
Tobias Ivarsson
Hacker @ Neo Technology
twitter: @thobe / #neo4j
email: tobias@neotechnology.com
web: http://www.neo4j.org/
web: http://www.thobe.org/

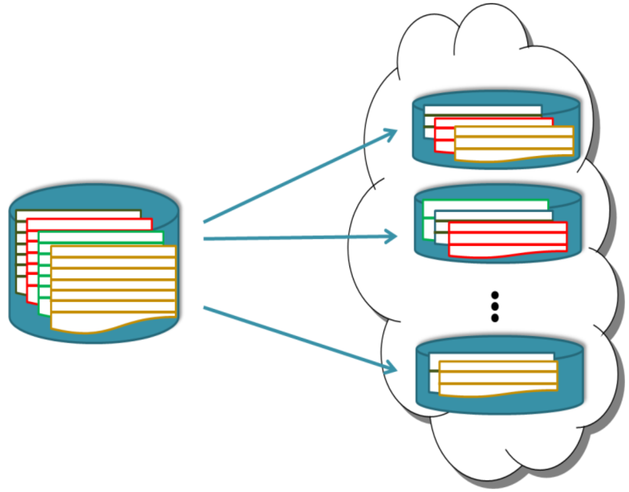
Atomicity in Key-Value Stores

- Operations on a single row are atomic.
- Objective: make read operations single-sited!
- Scalability and Elasticity: Data is partitioned across multiple servers.
- **Bigtable , PNUTS , Dynamo, Hypertable, Cassandra, Voldemort**

Practical approaches to scalability Circa Year 2000.

- Scale-up
  - Classical enterprise setting (RDBMS)
  - Flexible ACID transactions
  - Transactions in a single node
- Scale-out
  - Cloud friendly (Key value stores)
  - Execution at a single server
    - Limited functionality & guarantees
  - No multi-row or multi-step transactions





DSL UCSB

What about the Application Programmer?

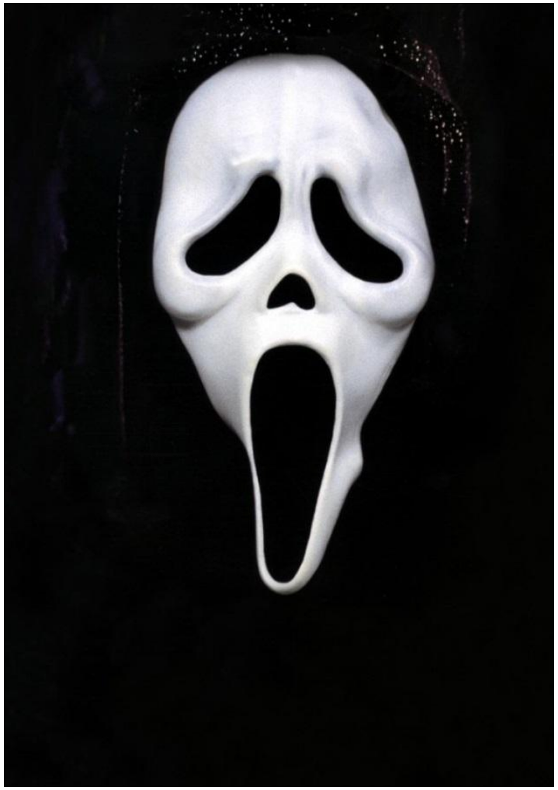
Distribution & Consistency

- Application developers need higher-level ***abstractions***:
  - MapReduce paradigm for Big Data analysis
  - Transaction Management in DBMSs

Outline

- NoSQL: Key-Value Stores
  - No Transactions.
  - Bigtable, Pnuts, Dynamo, Casandra,….
- SQL Take 1: Locality-based transactions
  - Limited Transactions
  - ElasTraS, G-Store, SQL-Azure, Relational Cloud
- SQL Take 2: Multi-data Centers
  - The Return of Transactions.
  - MegaStore
  - Paxos-CP
  - Spanner
  - Message-Futures
  - ……

NoSQL is apparently NOT going to deliver World Peace



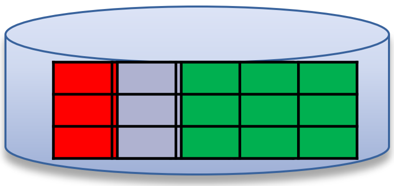
- It’s nice to have JOINs
- It’s nice to have transactions
- After 30 years of development, it seems that SQL Databases have some solid features, like the query analyzer.
- NoSQL is like the Wild West; SQL is civilization
- Gee, there sure are a lot of tools oriented toward SQL Databases.

Peter Wayner at InfoWorld “Seven Hard Truths” about NoSQL technologies July 2012.

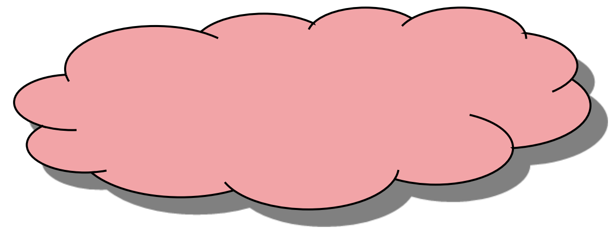

SQL


"THE RETURN"

Supporting SQL in the Cloud



RDBMS



Key Value Stores

Fission

Fusion

ElasTraS [HotCloud '09, TODS]
Cloud SQL Server [ICDE '11]
RelationalCloud [CIDR '11]

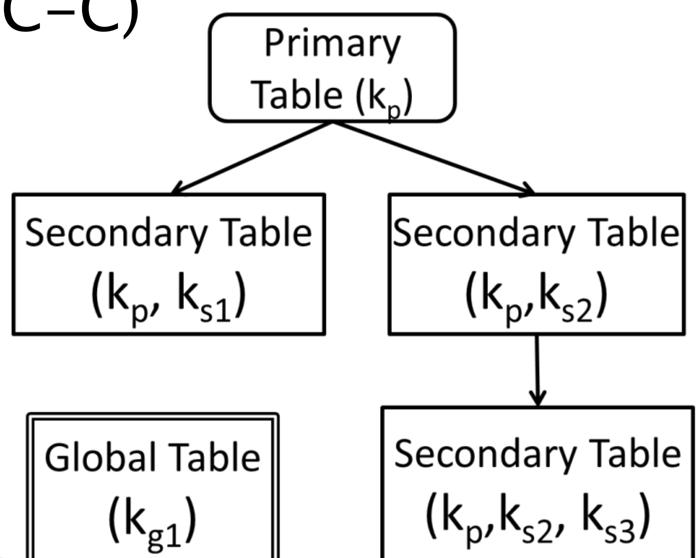
G-Store [SoCC '10]
MegaStore [CIDR '11]
ecStore [VLDB '10]
Walter [SOSP '11]

First Gen Data Center Systems

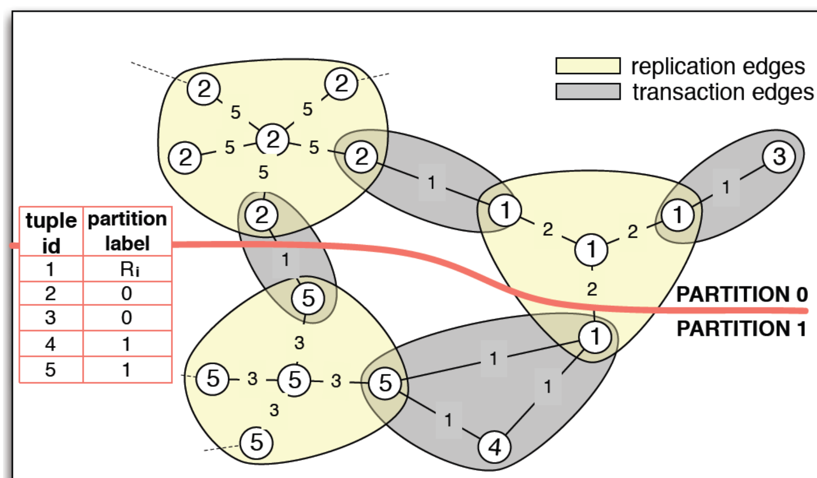
- These systems question the wisdom of abandoning the *proven* data management principles
- Gradual realization of the value of the concept of a “transaction” and other synchronization mechanisms
- Avoid distributed transactions by *co-locating* data items that are accessed *together*

Transactions using Data Partitioning (Statically)

- **Pre-defined** partitioning scheme
  - e.g.: Tree schema
  - ElasTras, SQLAzure
  - (TPC-C)

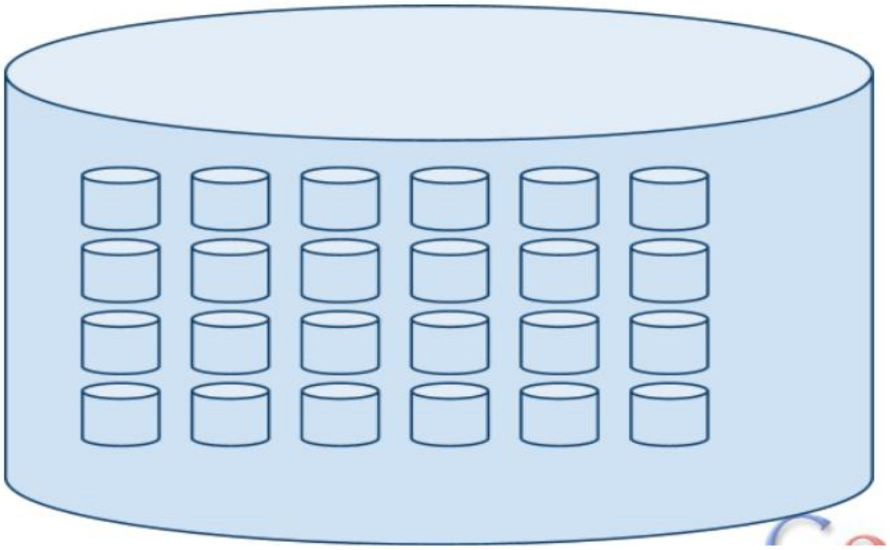


- **Workload driven** partitioning scheme
  - e.g.: Schism in RelationalCloud



Transactions using Data Partitioning (Statically)
Megastore (Google)-CIDR 2011

- Semantically pre-defined as Entity Groups
  - Blogs, email, maps
  - Cheap transactions in Entity groups (common)



Megastore Entity Groups

Semantically Predefined

- Email
  - Each email account forms a natural entity group
  - Operations within an account are transactional: user's send message is guaranteed to observe the change despite of fail-over to another replica
- Blogs
  - User's profile is entity group
  - Operations such as creating a new blog rely on asynchronous messaging with two-phase commit
- Maps
  - Dividing the globe into non-overlapping patches
  - Each patch can be an entity group

Dynamic Partitions

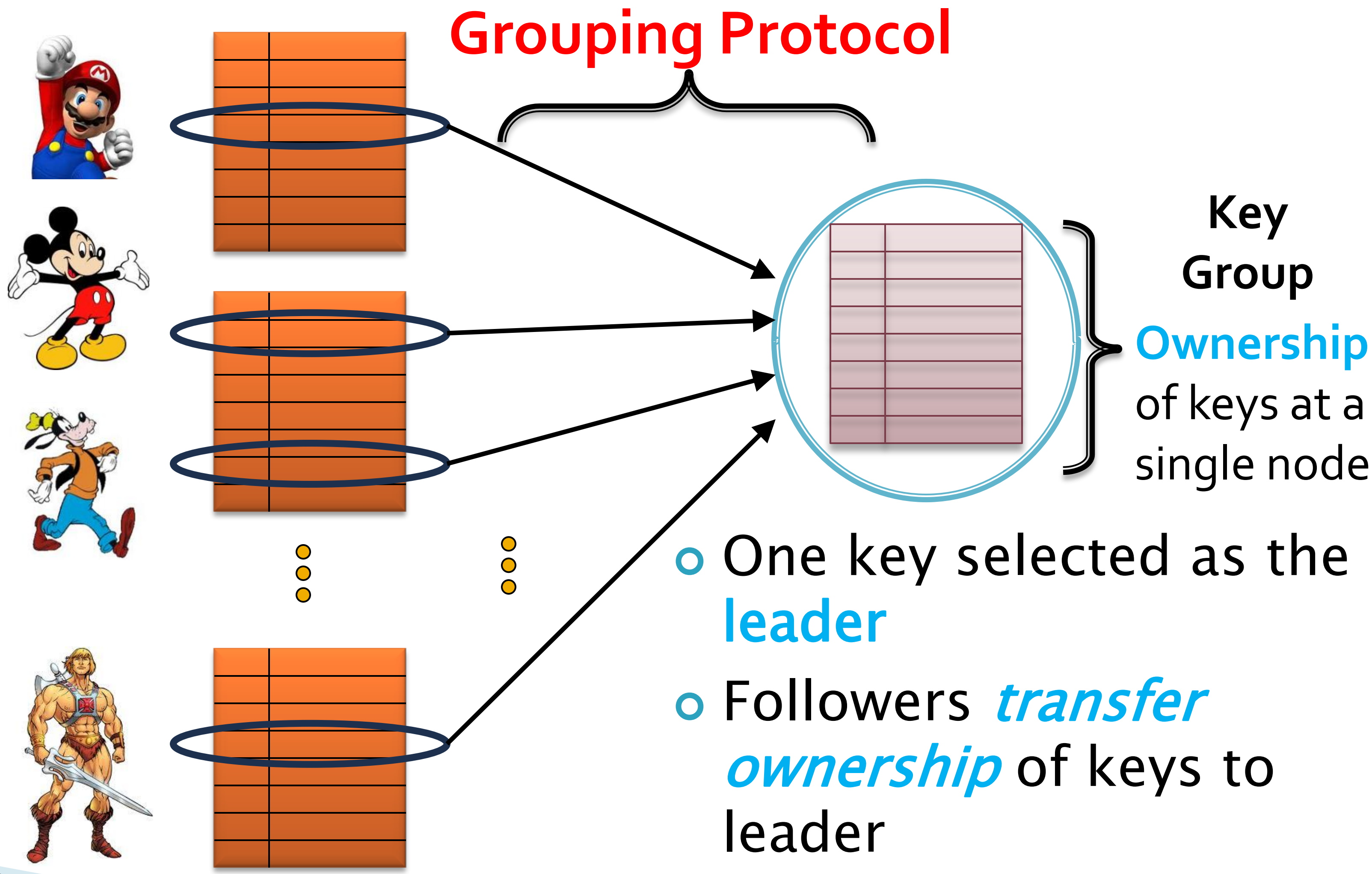
- Access patterns evolve, often rapidly
  - Online multi-player gaming applications
  - Collaboration based applications
  - Scientific computing applications
- Not amenable to static partitioning
  - Transactions access multiple partitions
  - Large numbers of distributed transactions
- How to efficiently execute transactions while avoiding distributed transactions?

G-Store (UCSB SoCC 2011)

- *Transactional* access to a *group of data items* formed *on-demand*
  - Dynamically formed database partitions
- *Challenge:* Avoid distributed transactions!
- *Key Group Abstraction*
  - Groups are *small*
  - Groups have *non-trivial lifetime*
  - Groups are *dynamic* and *on-demand*
- **Multitenancy**: Groups are dynamic tenant databases

Transactions on Groups

Without distributed transactions

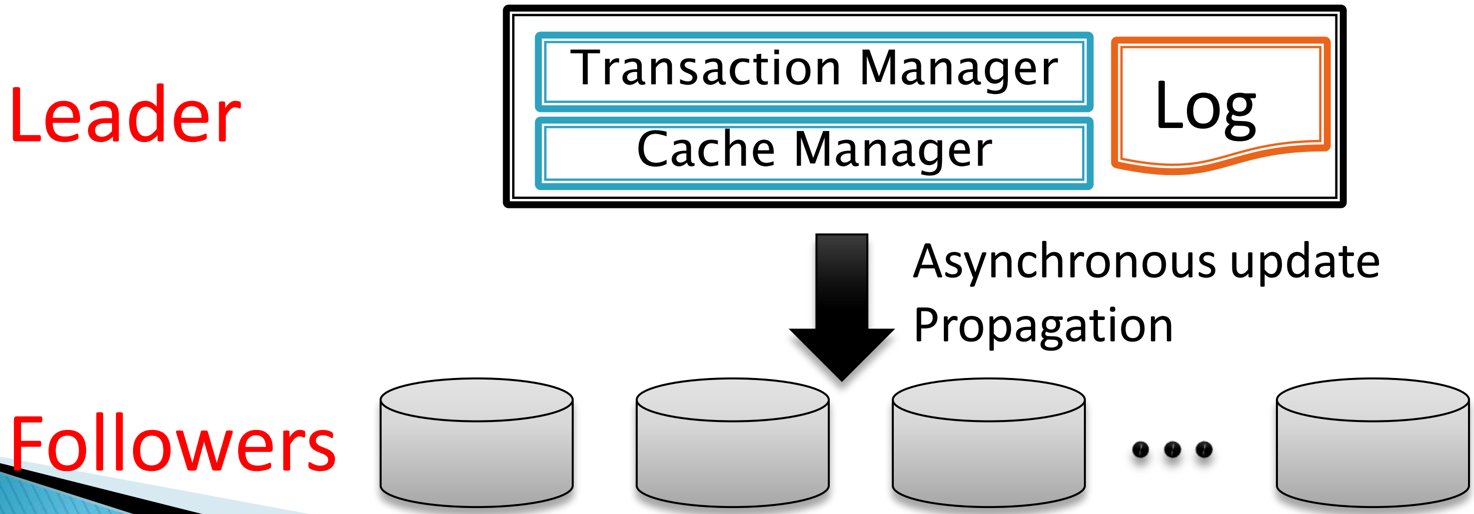


- One key selected as the **leader**
- Followers *transfer ownership* of keys to leader

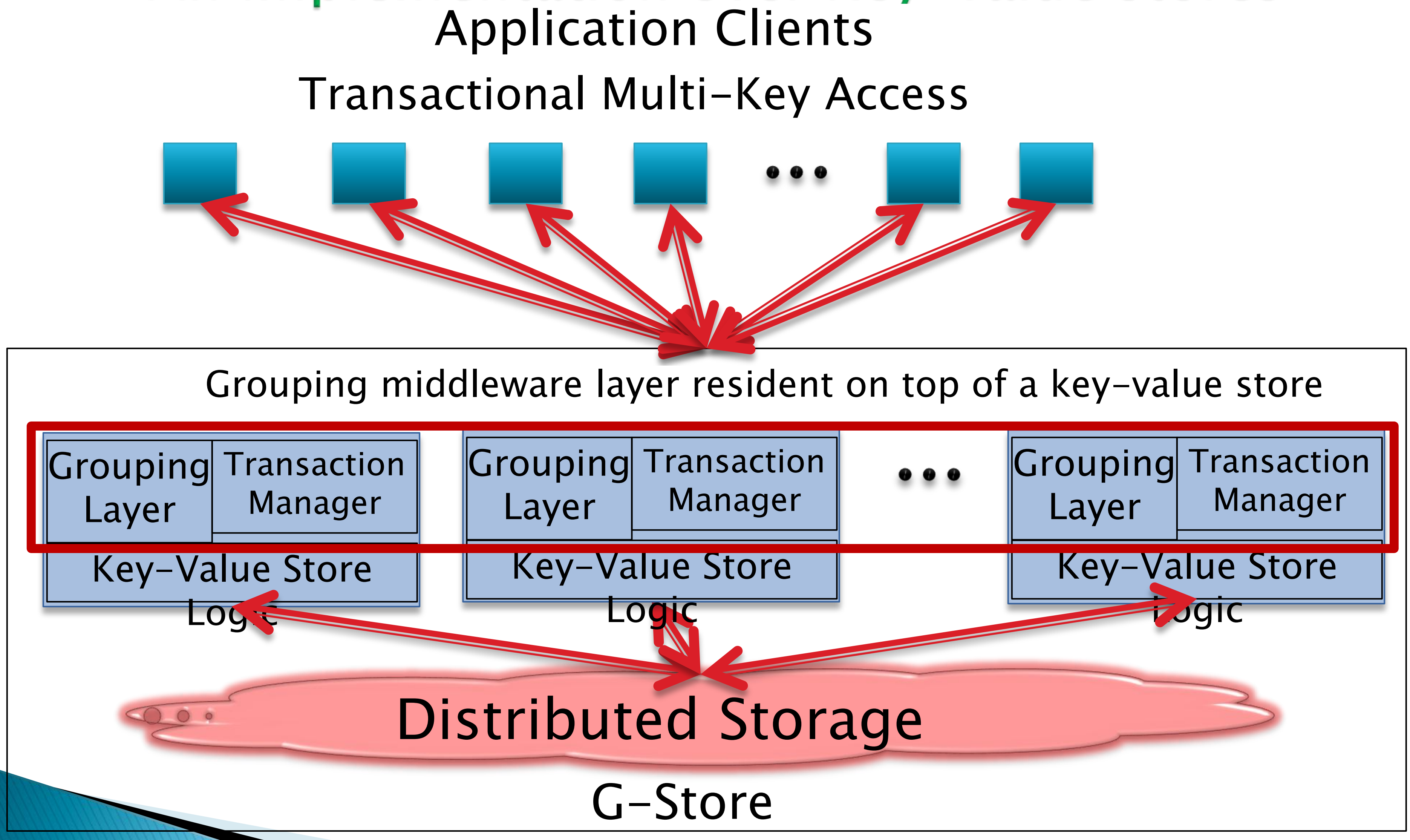
DSL UCSB

Efficient Transaction Processing

- How does the leader execute transactions?
  - Caches data for group members → underlying data store equivalent to a disk
  - Transaction logging for durability
  - Cache asynchronously flushed to propagate updates
  - Guaranteed update propagation



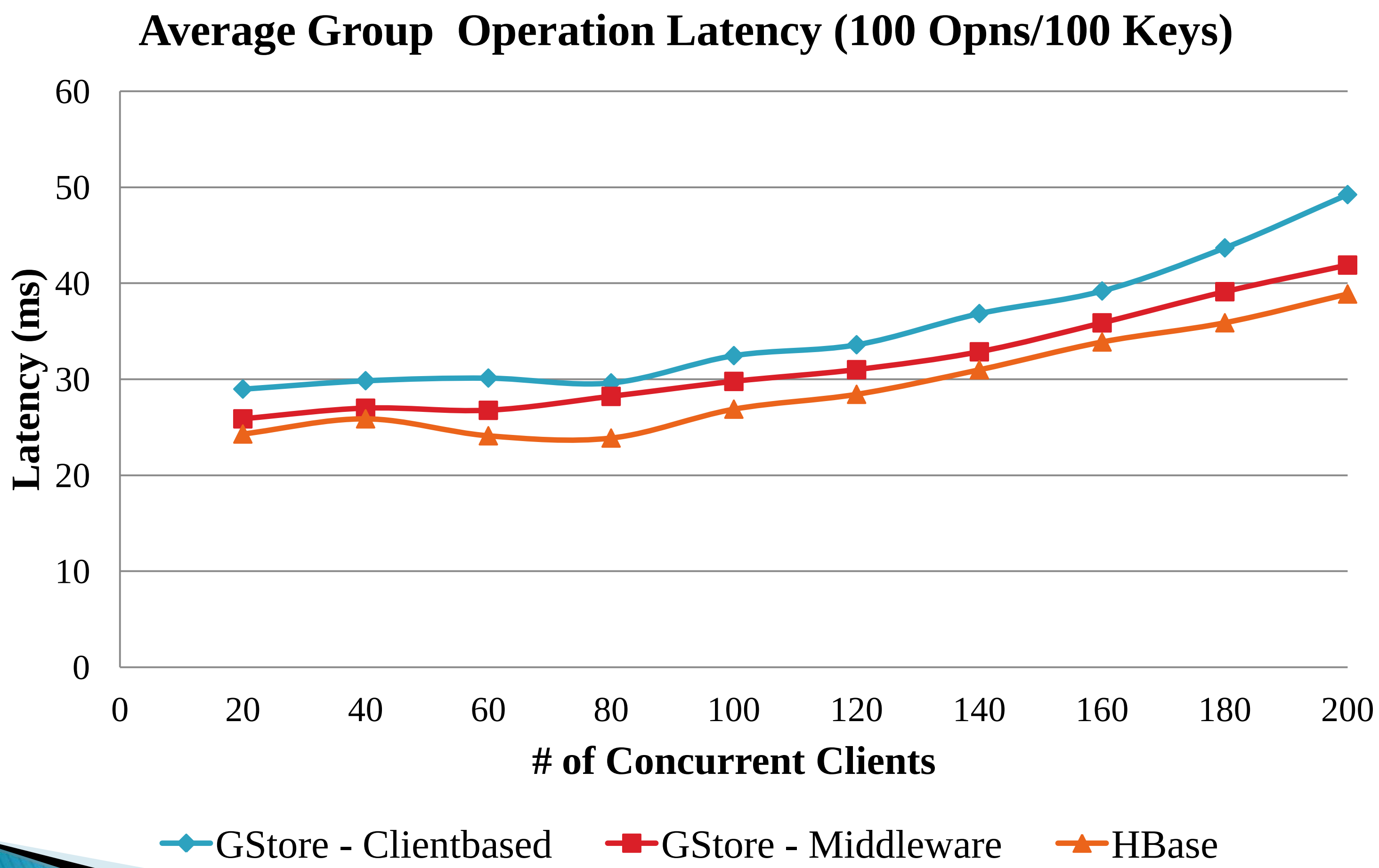
Prototype: G-Store
An implementation over Key-value stores



G-Store Evaluation

- Implemented using HBase
  - Added the middleware layer
  - ~15000 LOC
- Experiments in Amazon EC2
- Benchmark: An online multi-player game
- Cluster size: 10 nodes
- Data size: ~1 billion rows (>1 TB)
- For groups with 100 keys
  - Group creation latency: ~10 – 100ms
  - More than 10,000 groups concurrently created

Latency for Group Operations



Fault-tolerance in the Presence of Catastrophic Failures.

DSL

UCSB

Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' Data

Henry Blodget | April 28, 2011 | 87,084 | 75

Gmail Data Vanishes Into the Cloud

Monday, February 28, 2011

Contributed By:
Headlines

Google has been flooded with reports
complaining that their entire account h

INTERNET

GMail data loss attributed to software bug

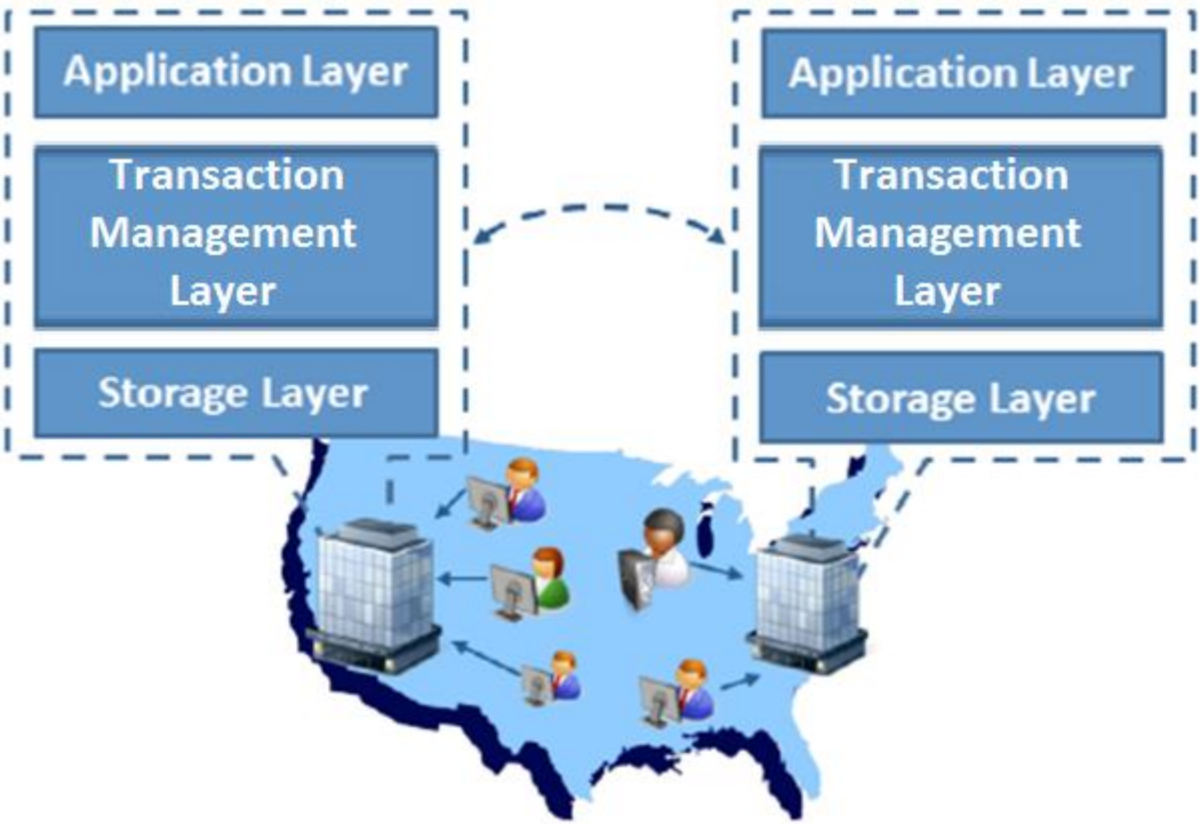
According to Google a bug in an updated version of their storage software was responsible for data loss in their servers affecting their redundant data stores. They have since reverted to an

- Hello,
- A few days ago we sent you an email letting you know that we were working on recovering an inconsistent data snapshot of one or more of your Amazon EBS volumes. We are very sorry, but ultimately our efforts to manually recover your volume were unsuccessful...
- What we were able to recover has been made available via a snapshot, although the data is in such a state that it may have little to no utility...
- If you have no need for this snapshot, please delete it to avoid incurring storage

Fault-tolerance in the Cloud

- Need to tolerate catastrophic failures
  - Geographic Replication
- How to support ACID transactions over data replicated at multiple datacenters
  - One-copy serializablity: Gives Consistency and Replication. Clients can access data in any datacenter, appears as single copy with atomic access
- Major challenges:
  - Latency bottleneck (cross data center communication)
  - Concurrent Consistency
  - Replica Consistency

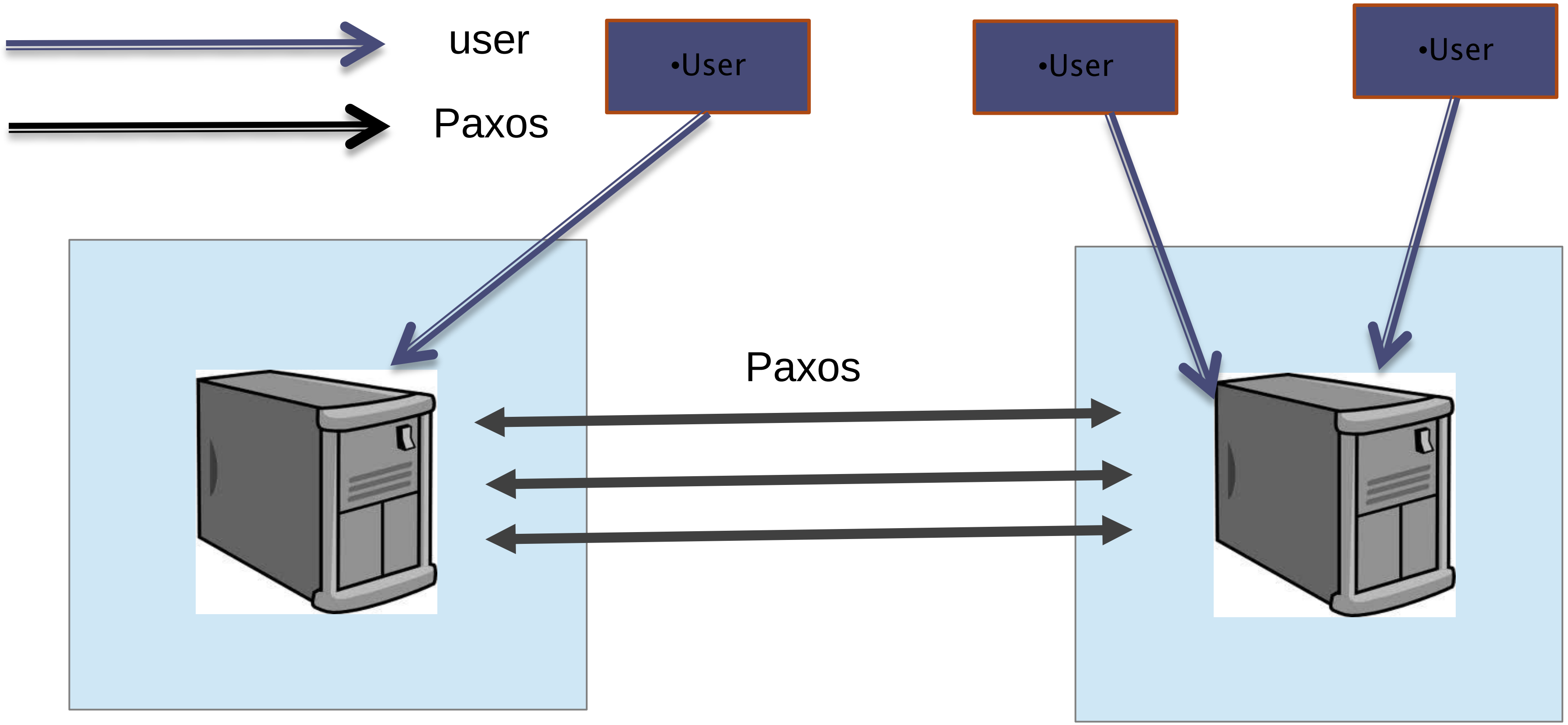
Cross-datacenter Replication



Consistency and Replication hand in hand

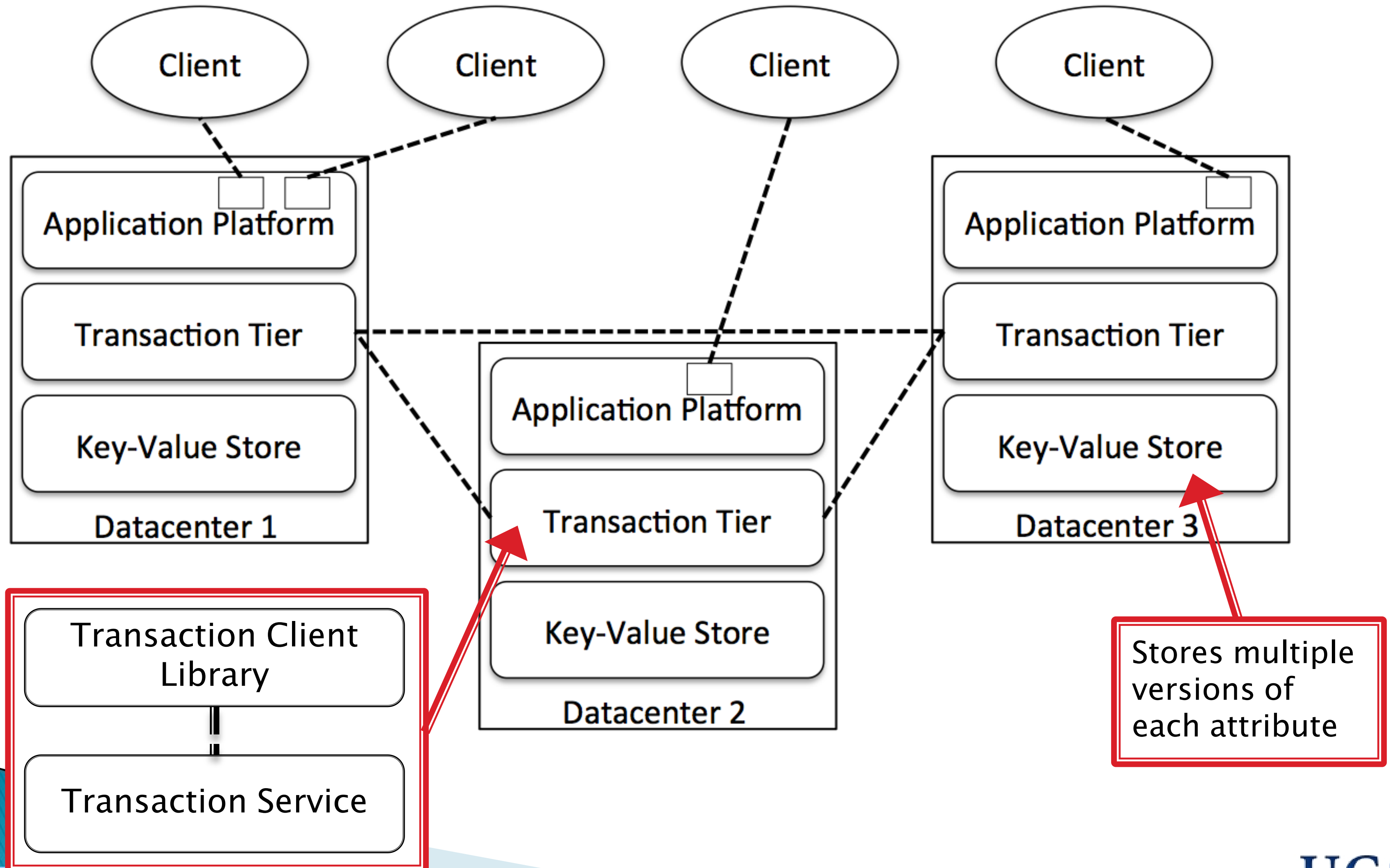
- The Paxos Approach

Megastore–Google (CIDR11) PaxosCP–UCSB (VLDB12)



Log position	1	2	3	4	5
Transaction	α	β	γ		

System Architecture (Mega Store)



Data Model & Write-Ahead Log

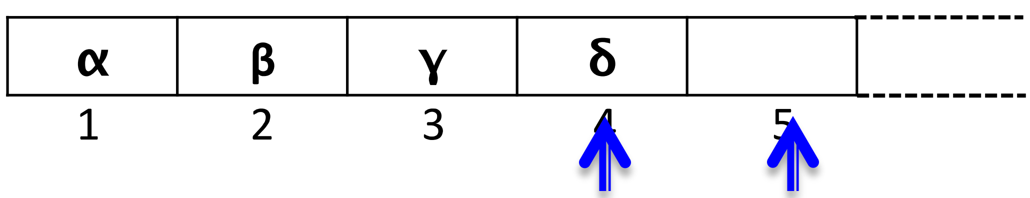
- Data divided into entity groups.
- Each group has write-ahead log.
- Data and log replicated at every datacenter.
- Optimistic concurrency control:
  - Read from datastore.
  - Write to local copy.
  - On commit, write to log.
  - Log entry: (*txn_id*, *read set*, *write set*)
- Log entries applied to data as needed.

DSL

UCSB

Optimistic Concurrency Control with Write-Ahead Log

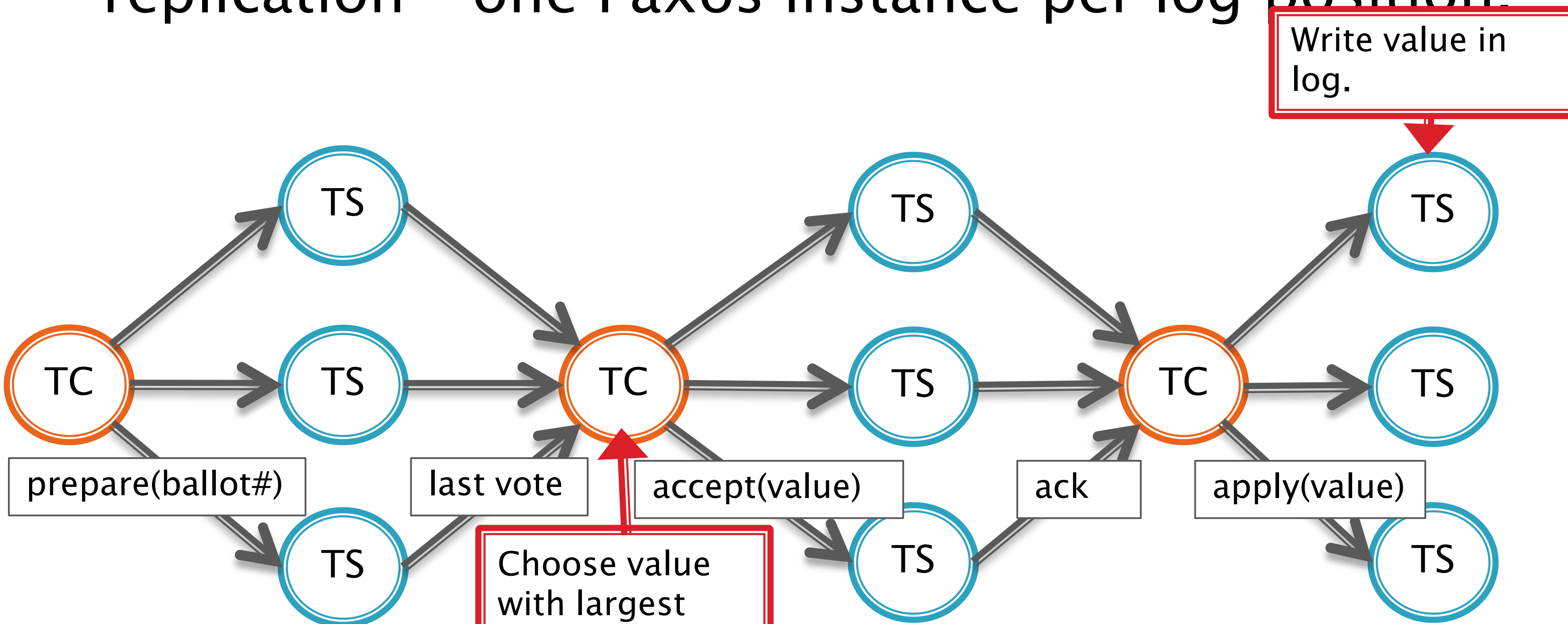
- Every tenant has a write-ahead log, replicated at every datacenter.



- Transaction operations:
  - **Read** version based on read log position.
  - **Write** in to local copy.
  - **Commit**
    - If read-only, automatic commit.
    - Else, try to commit to **commit log position**. Transaction Services coordinate using PAXOS to decide whether to commit or abort.

Paxos Commit Protocol (Megastore, CIDR 2011)

- Paxos for state machine replication (Lamport98).
- Here used for concurrency control and log replication – one Paxos instance per log position.

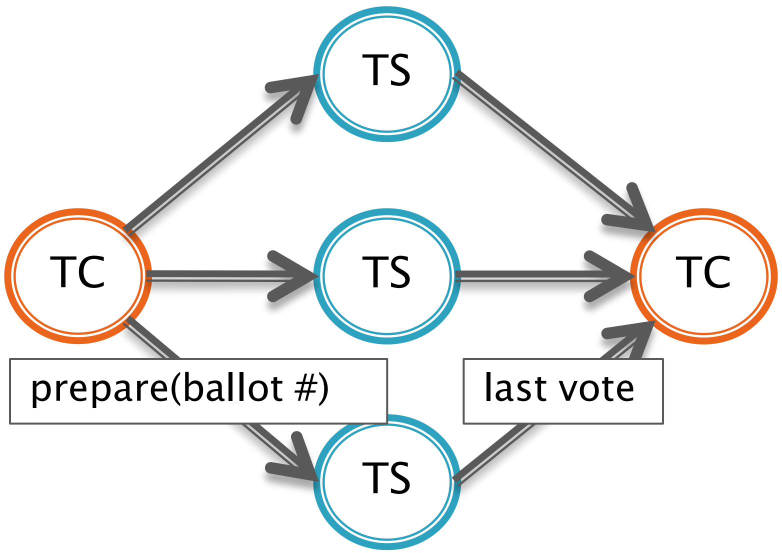


Only one ... s each log position.
Others are aborted ➔ Concurrency Prevention not Concurrency Control!

Write transactions are serialized!

Paxos with Combination and Promotion

UCSB: Patterson et al. VLDB 2012



- If no majority value in "last vote" messages
  combine nonconflicting values,
  send accept for combined values.
- Else if majority respond and no conflicts with winning transaction
  promote to next log position (repeatedly).
- Else continue basic Paxos

Paxos-CP only aborts a transaction if commit would violate one-copy serializability, ie, a conflict with a preceding write

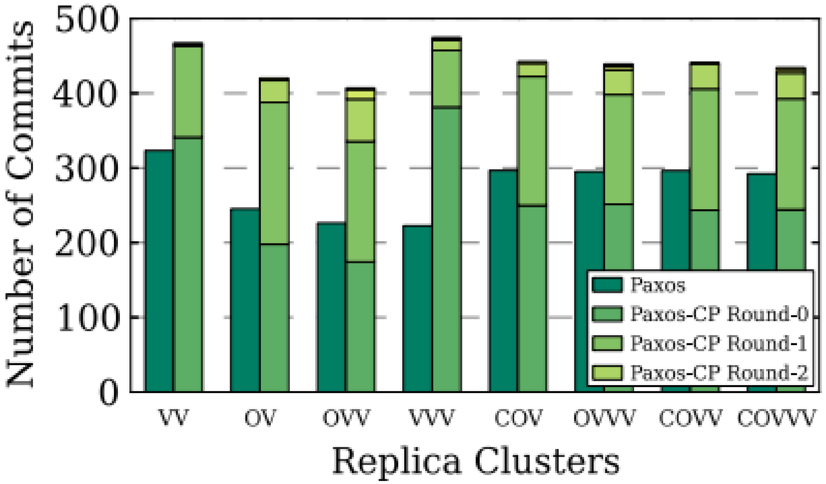
➔ true Concurrency Control.

Evaluation Setup

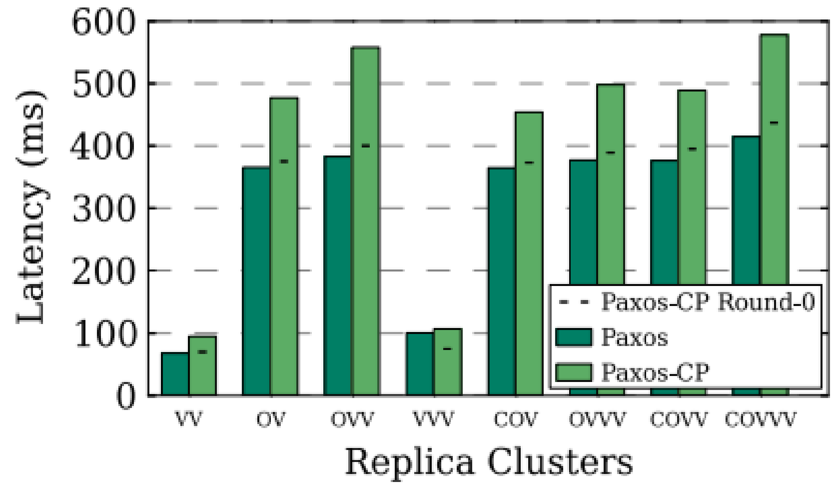
- Prototype implementation:
  - Basic Paxos and Paxos-CP, in Java
  - Hbase for key-value store
  - Modified YCSB benchmark (Cooper SOCC'10, Das VLDB'11)
- Evaluation setting:
  - Run on Amazon's public cloud
  - Using medium Hi-CUP instances with Elastic Block Storage
  - 3 nodes in Virginia, 1 in Oregon, 1 in California
- Benchmark workload:
  - 500 transactions
  - Each transaction access 10 attributes, 50% reads, 50% writes

Paxos-CP Evaluation

- Multi-data center experiments on EC2
  - Virginia – Oregon – California



(a) Number of successful transaction commits, out of 500 transactions.

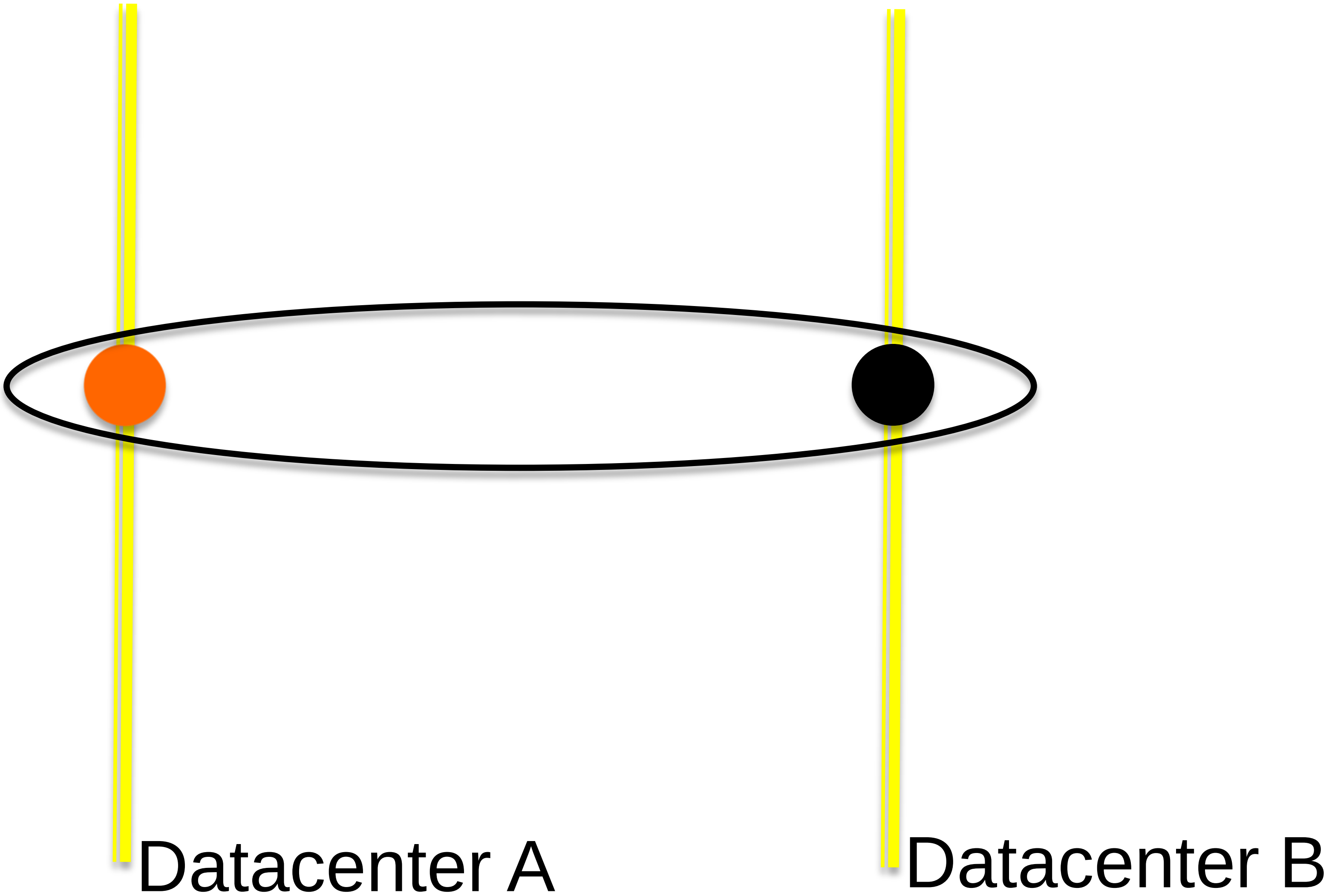


(b) Average latency for all transactions.

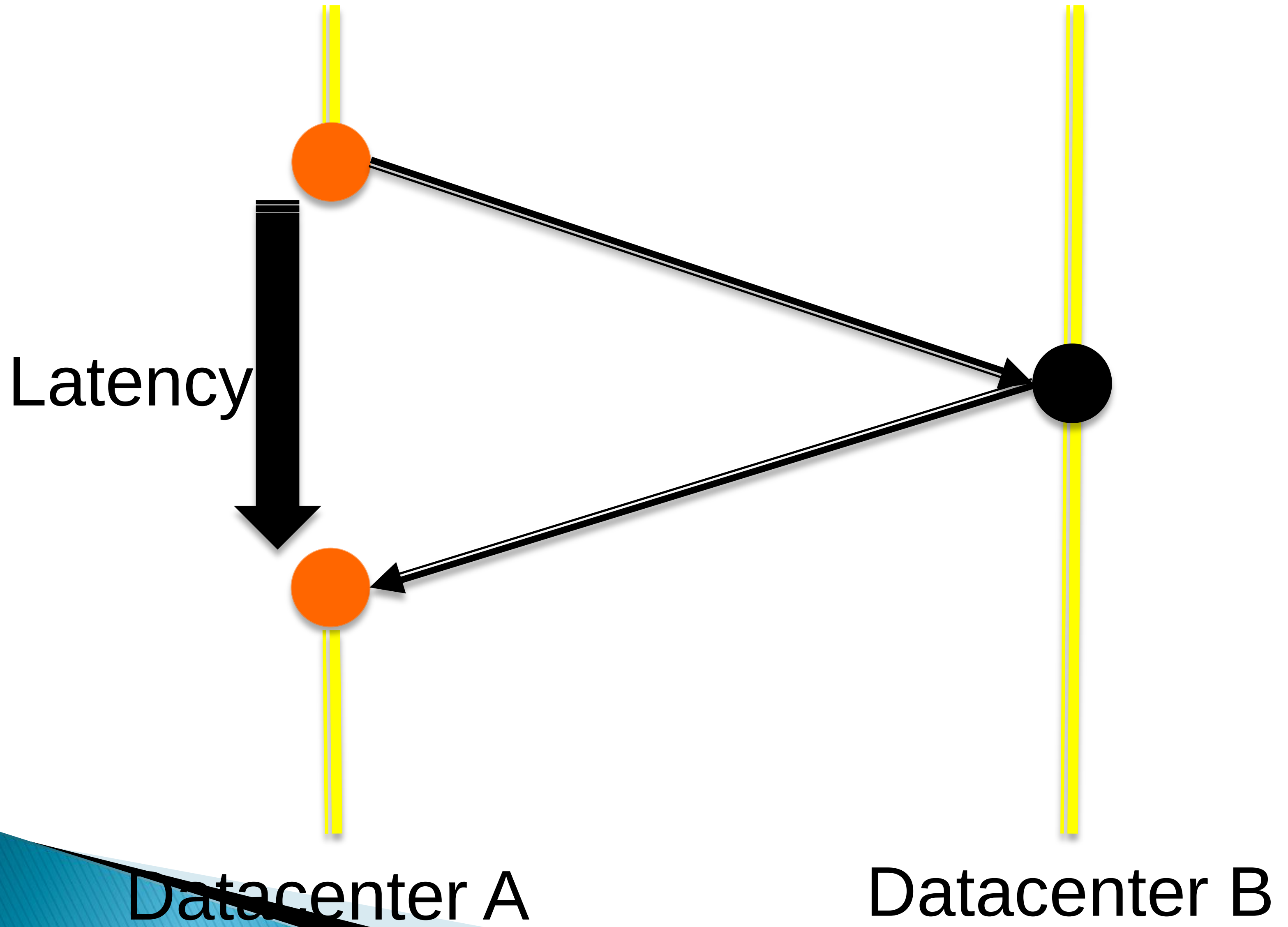
Calculated from averages for all combinations of replica locations.
1 transaction per second. 100 total attributes.

Consistency and Replication hand in hand

- Asynchronous coordination



Datacenter A
Datacenter B

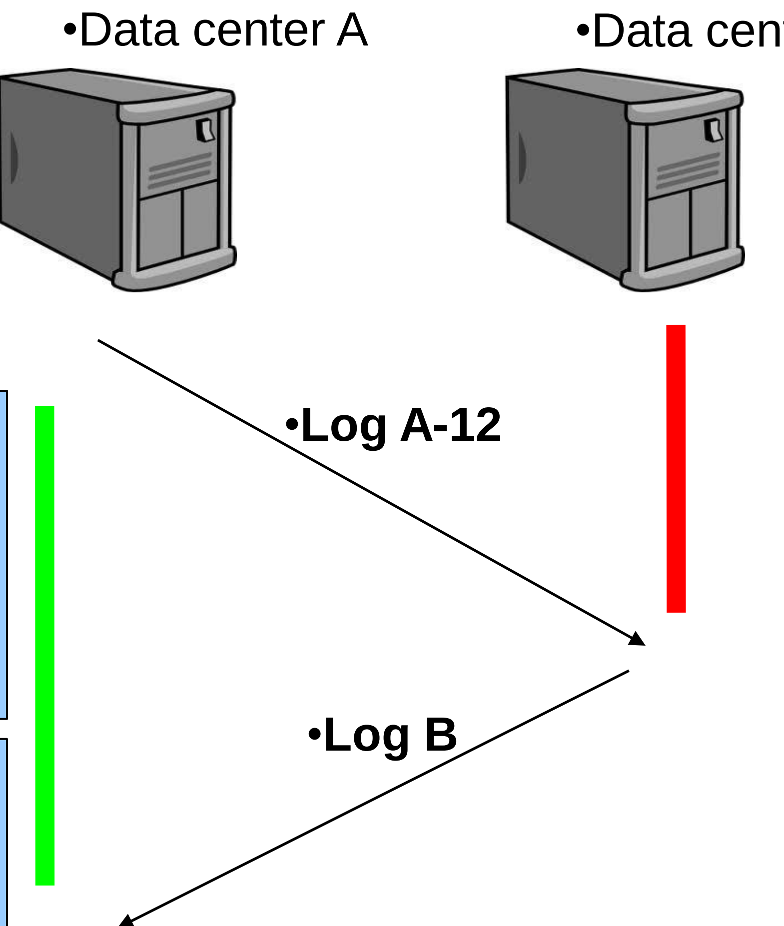

Latency
Datacenter A
Datacenter B

Message Futures – UCSB [CIDR 13]

- Log A-12 carries a
- Reservation with
- A value 12

- Transactions requesting to commit in the green
- area are assigned
- reservation number 12

- Green area transactions
- Commit at point 13 if no Conflicts are observed with Log B

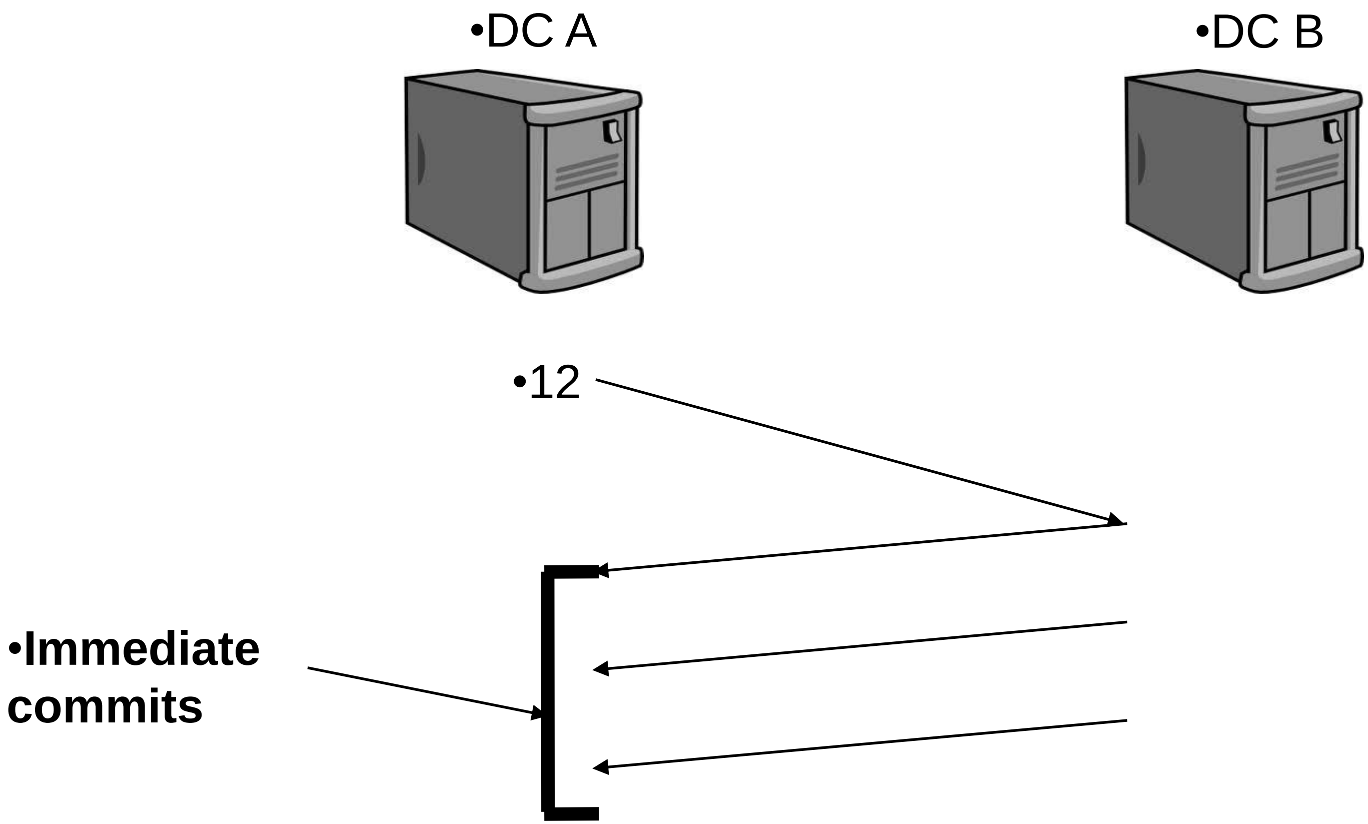


- Transactions in the red
- region and earlier are
- included in Log B

- Log B acknowledges
- The receipt of
- Reservation 12

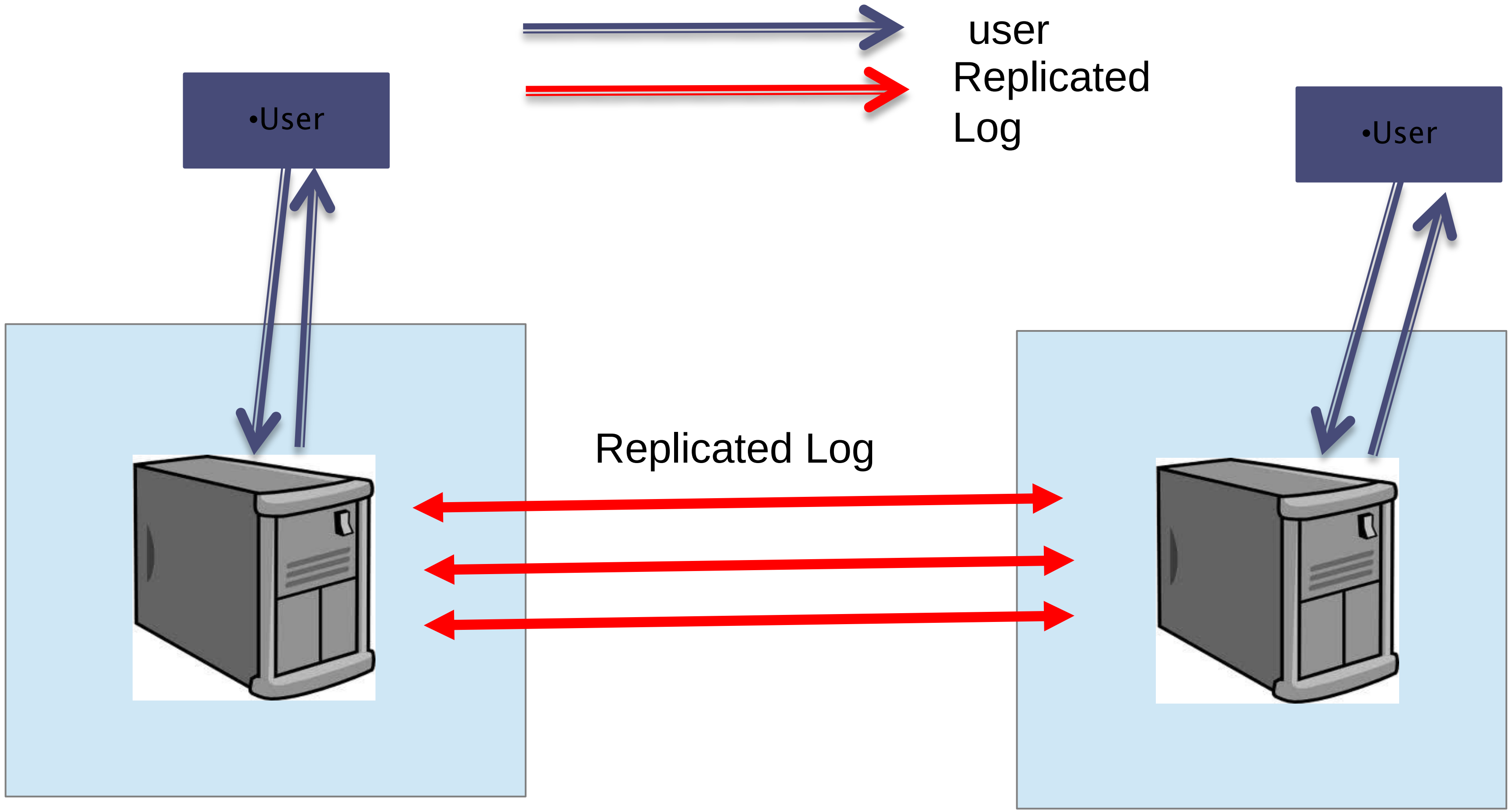
Message Futures cases

•Data center B sends Logs at a higher rate.



•A new transactions at the immediate commit zone
•will have its reservation (12) already acknowledged

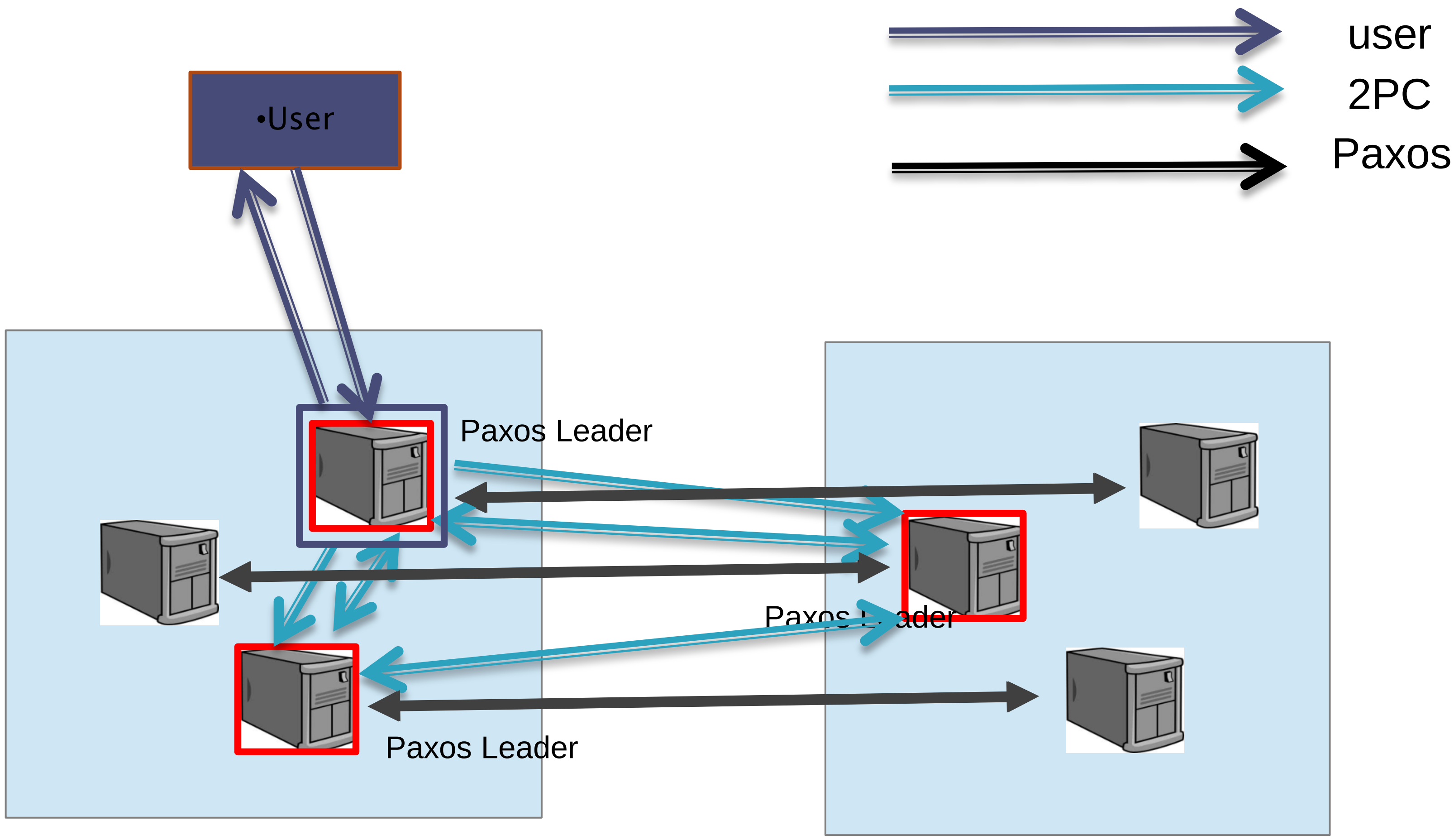
Message Futures



Consistency over Replication

- Transaction execution ON fault-tolerant replicated storage

Spanner—Google [OSDI 12]

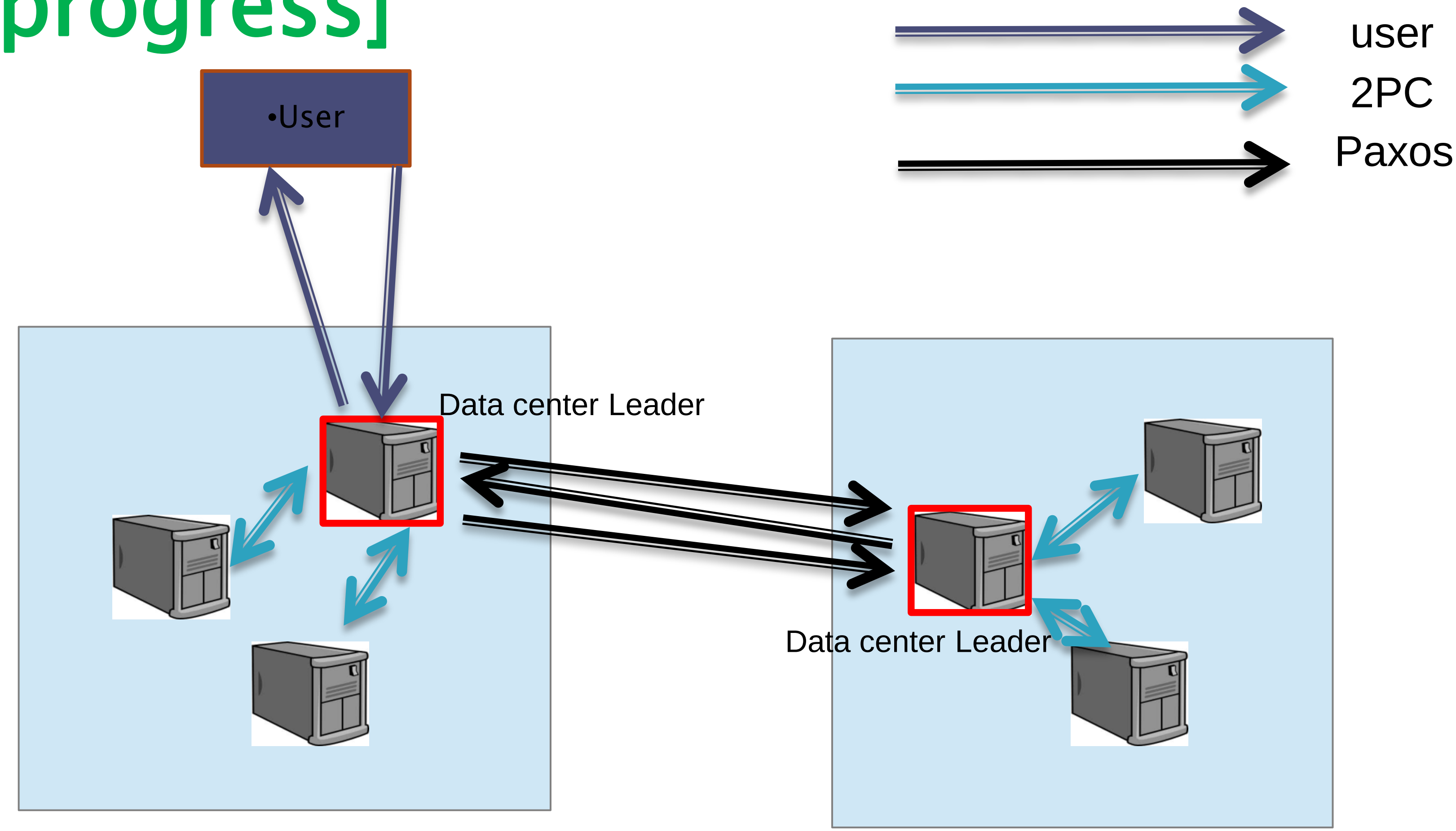


Number of wide-area messages: 1 3 4 6

Replication over Consistency

- Replication on Consistent ACID Data Centers

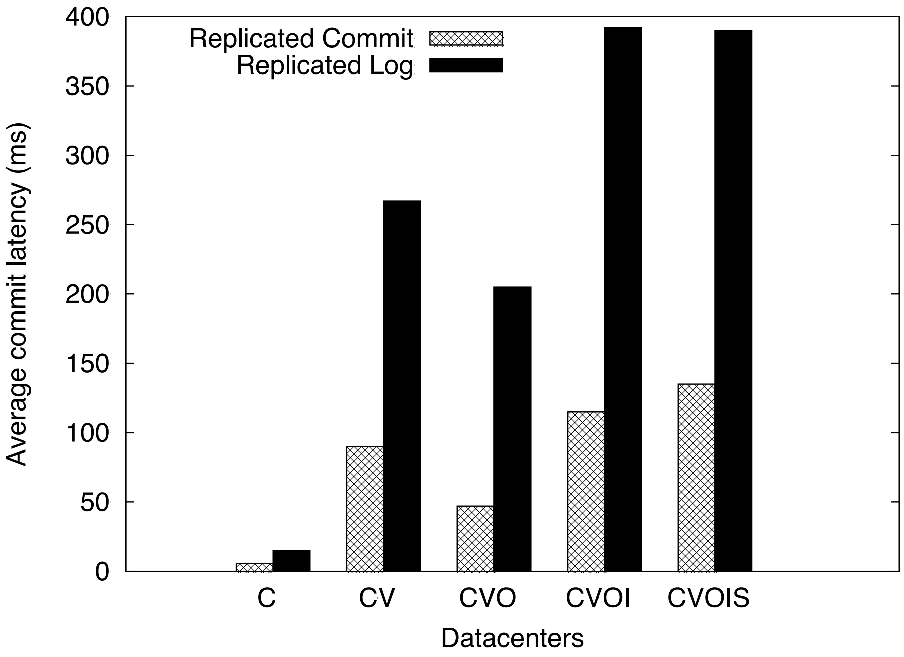
Replicated Commit --UCSB [in-progress]

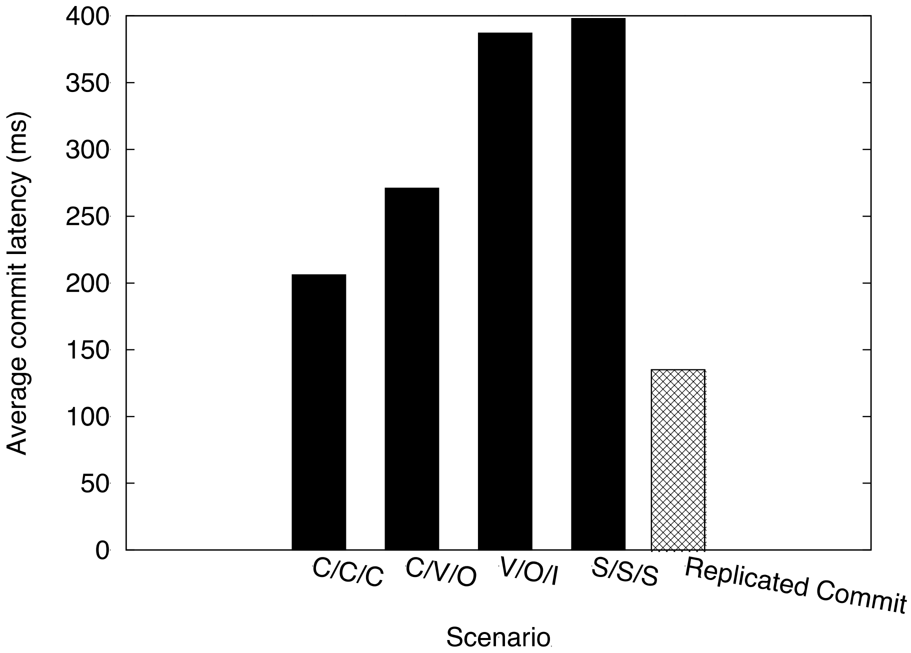


Number of wide-area messages: 1 2

Spanner/Replicated Commit Evaluation

- Multi-data center experiments on EC2
  - Virginia – Oregon – California – Ireland – Singapore





Replicated Log is the class of protocols containing Google's Spanner. CVO is a 3-data center scenario and C/V/O is a replicated log scenario with 3 replica leaders at C, V, and O.

Concluding Remarks

- Better understand the various paradigms and alternatives.
- Develop a general framework to explain the pros and cons of these approaches.
- Automatically configure systems for better performance.

- We are in the era of Globalization

Round Trip Times (RTT)