Dealing with
The Distributed Systems
~~Foundations~~ of Managing Data in
Challenges the Cloud

Amr El Abbadi

University of California,

Santa Barbara

Evolution of computing history

- Main Frame with terminals
- Network of PCs & Workstations.
- Client-Server
- Now, moving forward to Large cloud.

Cloud Computing: Why Now?

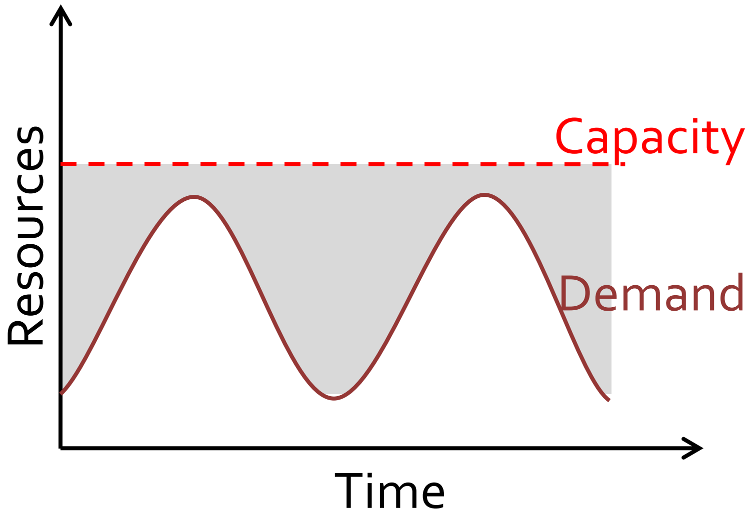
- Experience with very large datacenters
  - Unprecedented economies of scale
  - Transfer of risk

- Technology factors
  - Pervasive broadband Internet
  - Maturity in Virtualization Technology

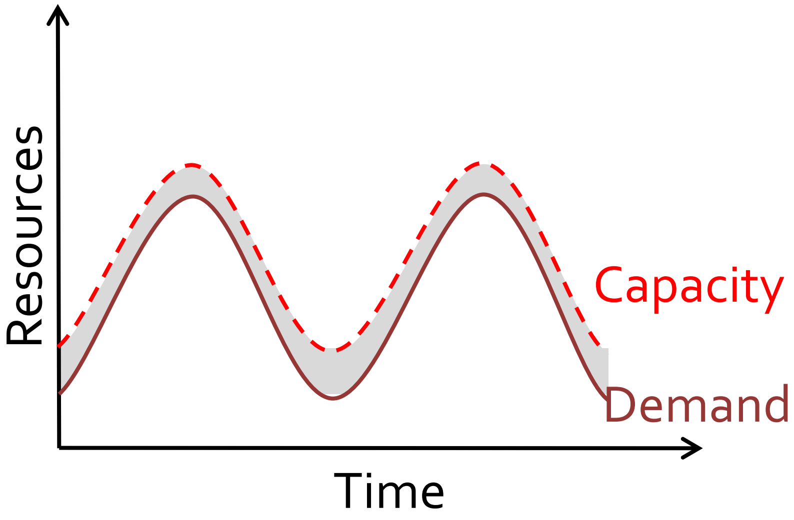
- Business factors
  - Economies of Scale
  - Pay-as-you-go billing model

Cloud's Promise: Elasticity

- Pay per use instead of provisioning for peak



Traditional Infrastructures

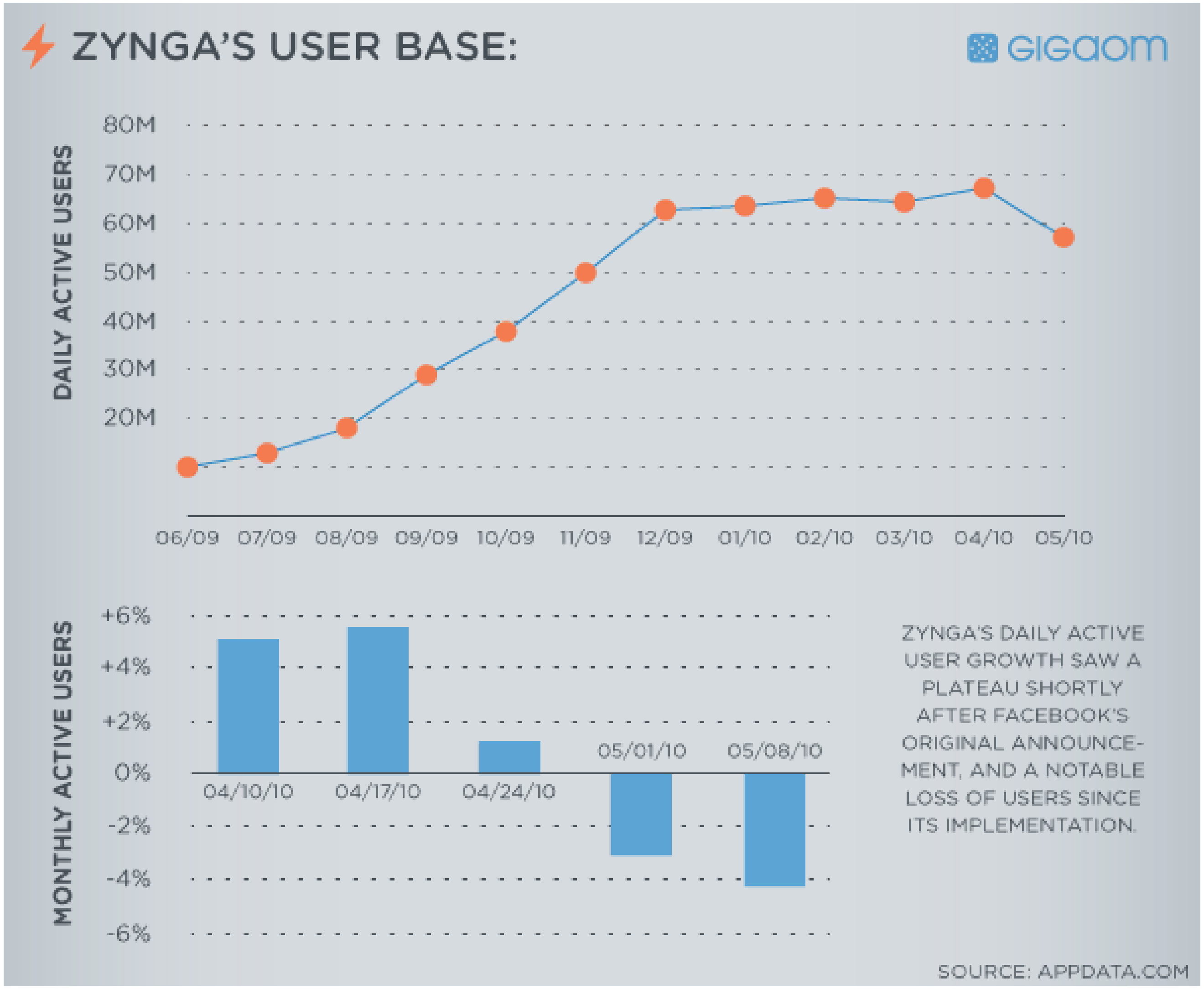


Deployment in the Cloud

▨ Unused resources

Slide Credits: Berkeley RAD Lab

Cloud Reality: Elasticity



ZYNGA'S DAILY ACTIVE USER GROWTH SAW A PLATEAU SHORTLY AFTER FACEBOOK'S ORIGINAL ANNOUNCEMENT, AND A NOTABLE LOSS OF USERS SINCE ITS IMPLEMENTATION.

SOURCE: APPDATA.COM

Explosive Data growth

- Wikipedia has over 3.5 million pages.
- Yahoo! 650M+ unique user, 11B page visits/month
- Flickr members uploaded over 5 billion photos

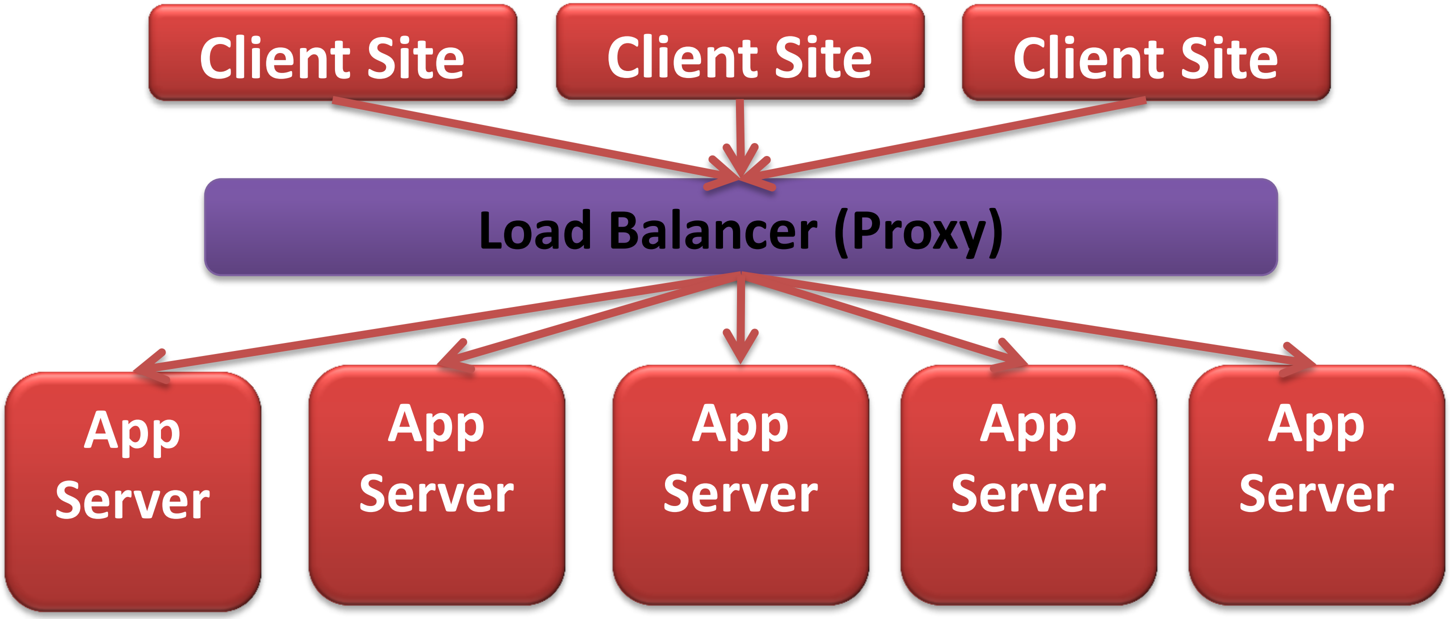
- Facebook:1Billion users,1.13 Trillion "likes", 219Billion photos and 140.3 Billion friendships.

- You Tube: 35 hours of videos uploaded each min.
- “more video uploaded to YouTube in the past two months than there would have been if ABC, CBS, and NBC had been airing 24/7 since 1948!”

Cloud Properties

- Commodity hardware
- Large Scale
- Elasticity

Elasticity in the Cloud



Client Site

Client Site

Client Site

Load Balancer (Proxy)

App Server

App Server

App Server

App Server

App Server

Why does this work?

- As long as requests are stateless, we can add more resources, thus providing:

- Scale
- Elasticity

But, most services need DATA!

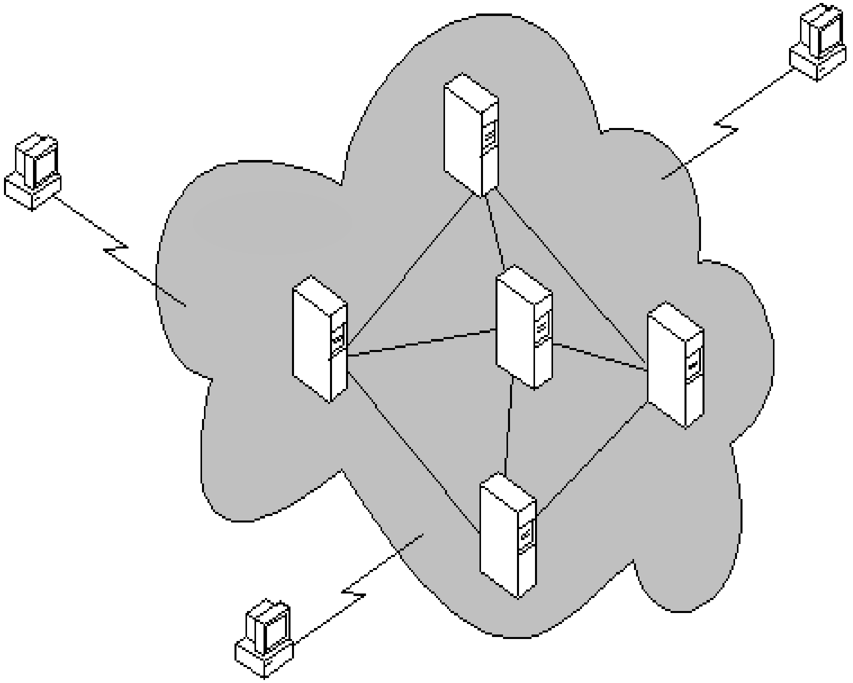
- Challenges:
  - How to scale with the increasing amounts of data
  - Where to store the data
  - Accessing data on multiple sites
  - Failures

Need

- Fault-tolerance:
  - Replication
- Large scale data:
  - Partition data across multiple servers
- Managing the system state.
- Must understand:
  - Database foundations
  - Distributed systems foundations.

Main Characteristics of Distributed Systems

- Independent processors, sites, processes
- Message passing
- No shared memory
- No shared clock
- Independent failure modes



Distributed System Models

- Synchronous System: Known bounds on times for message transmission, processing , bounds on local clock drifts, etc.
  - Can use timeouts
- Asynchronous System: No known bounds on times for message transmission, processing, bounds on local clock drifts, etc.
  - More realistic, practical, but no timeout.

CAUSALITY AND TIME

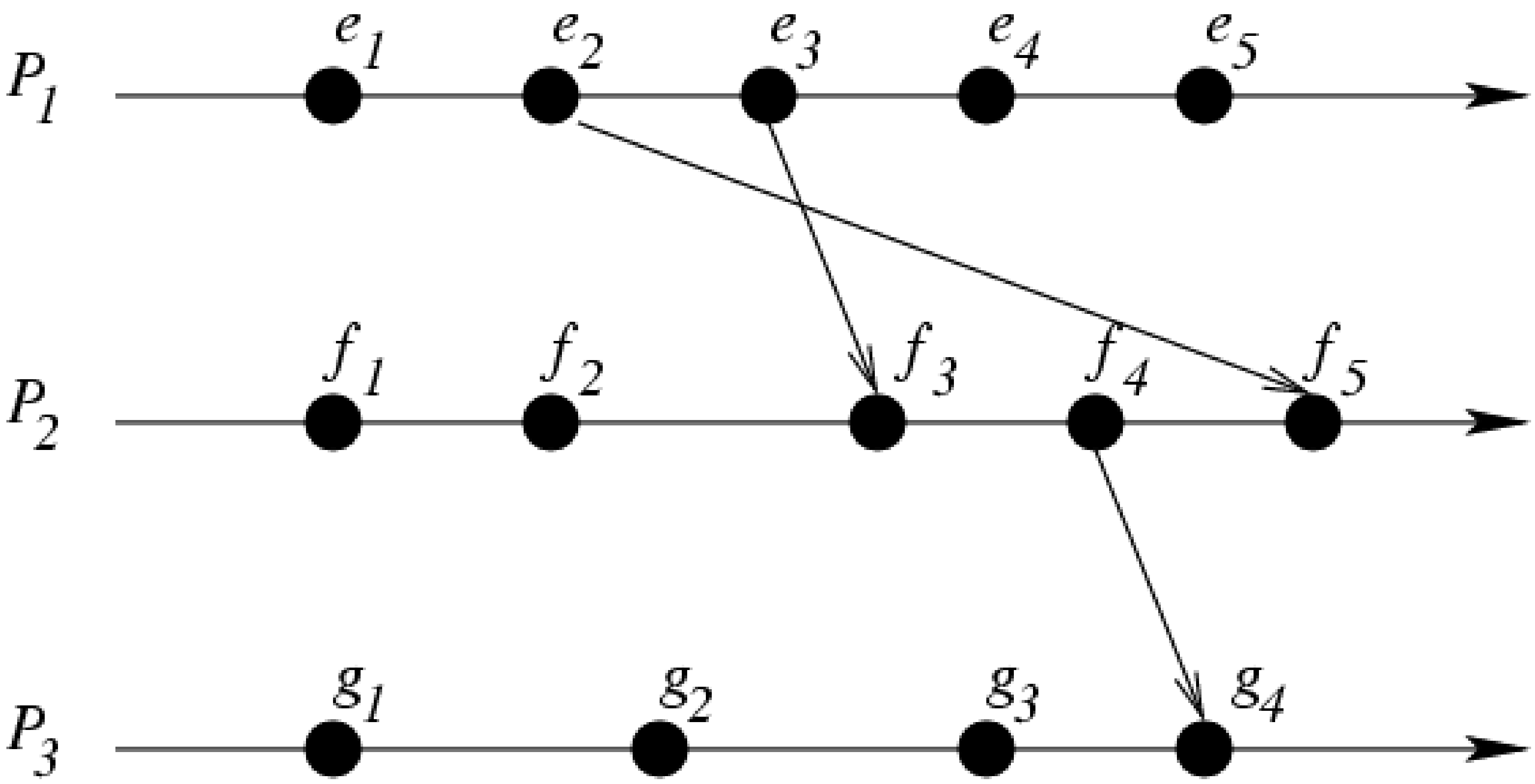
What is a Distributed System?

- A simple model of a distributed system proposed by Lamport in a landmark 1978 paper:
- “Time, Clocks and the Ordering of Events in a Distributed System” Communications of the ACM

What is a Distributed System?

- A set of processes that communicate using message passing.
- A process is a sequence of events
- 3 kinds of events:
  - Local events
  - Send events
  - Receive events
- Local events on a process for a total order.

Example of a Distributed System



Happens Before or Causal Order on Events

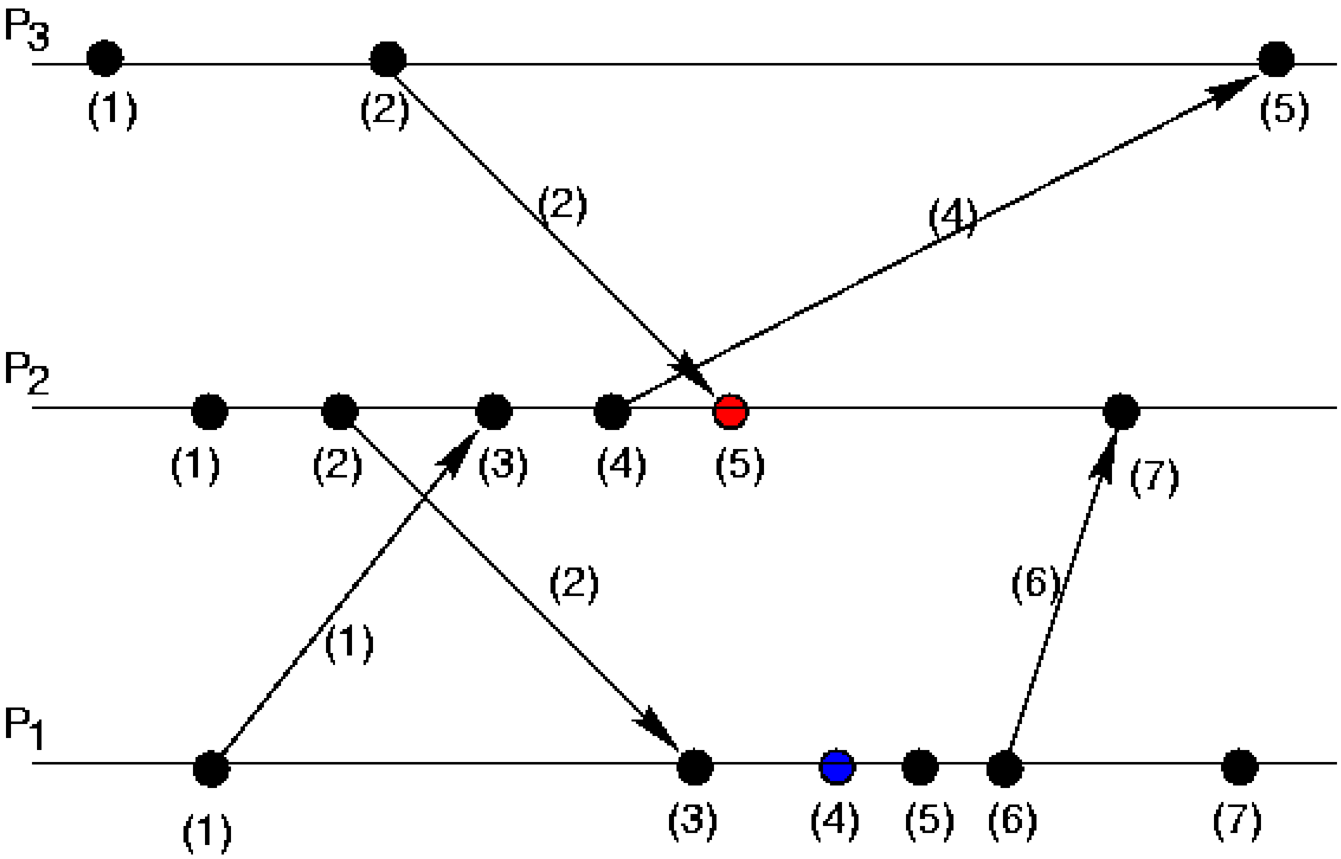
- Event *e* *happens before (causally precedes)* event *f*, denoted $e \rightarrow f$ if:
  1. The same process executes *e* before *f* ; or
  2. *e* is send(*m*) and *f* is receive(*m*); or
  3. Exists *h* so that $e \rightarrow h$ and $h \rightarrow f$
- We define *concurrent*, $e \mid\mid f$, as:

$$\neg(e \rightarrow f \vee f \rightarrow e)$$

Lamport Logical Clocks

- Assign “clock” value to each event
  - if *a*→*b* then **clock(*a*) < clock(*b*)**
- Assign each process a clock “counter”.
  - Clock must be incremented between any two events in the same process
  - Each message carries the sender’s clock value
- When a message arrives set local clock to:
  - max(local value, message timestamp + 1)

Example of a Logical Clock



Vector clocks

1. Vector initialized to 0 at each process

   $V_i[j] = 0$ for $i, j = 1, \ldots, N$

2. Process increments its element of the vector in local vector before event:

   $V_i[i] = V_i[i] + 1$

3. Piggyback V_i with every message sent from process P_i

4. When P_j receives message, compares vectors element by element and sets local vector to higher of two values

   $V_j[i] = \max(V_i[i], V_j[i])$ for $i=1, \ldots, N$

Comparing vector timestamps

Define

$V = V'$ iff $V[i] = V'[i]$ for $i = 1 \ldots N$

$V \leq V'$ iff $V[i] \leq V'[i]$ for $i = 1 \ldots N$

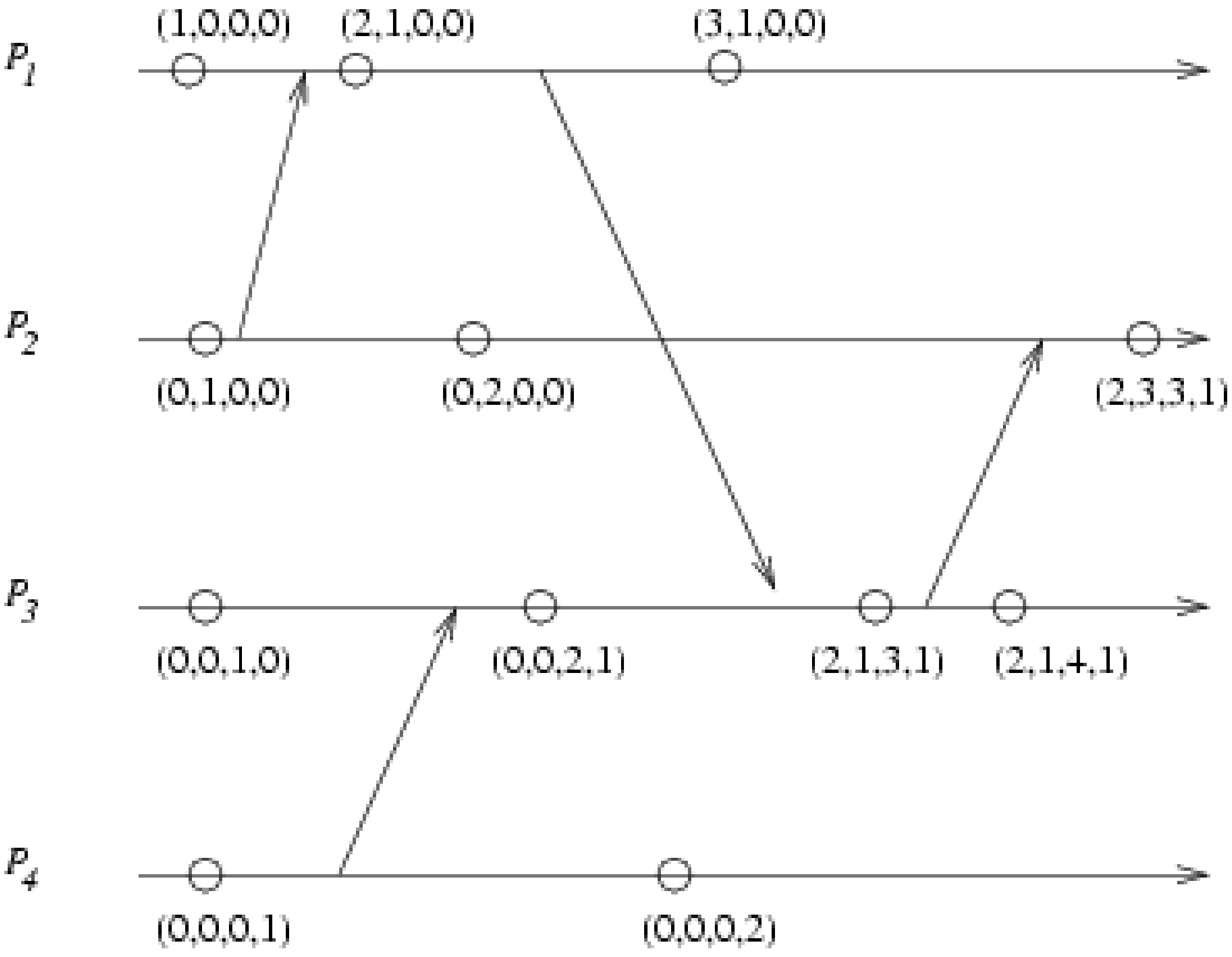
For any two events e, e'

$e \rightarrow e'$ if and only if $V(e) < V(e')$

Two events are **concurrent** if **neither**

$V(e) \leq V(e')$ nor $V(e') \leq V(e)$

Vector Clock Example



MUTUAL EXCLUSION AND QUORUMS

Distributed Mutual Exclusion

- Given a set of processes and a single resource, develop a protocol to ensure exclusive access to the resource by a single process at a time.

- This is a fundamental operation in operating systems, and is generalized to locking in databases.

Centralized Solution

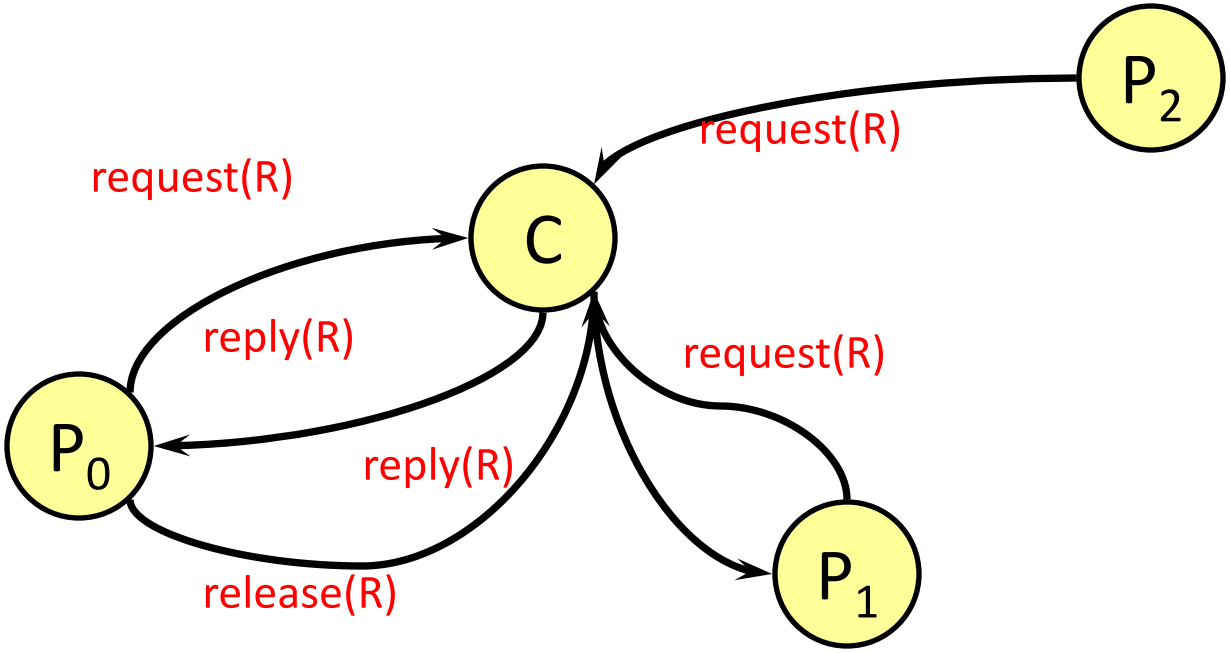
- Choose a special coordinator site, coord.
- Coord maintains a queue of pending requests.
- Protocol:
  - Process send request to coord.
  - If no other request, coord sends back reply.
    - Otherwise, put request in queue
  - On receipt of reply, process accesses resource.
  - Once done, process sends release to coord.
  - On receipt of release, coord checks queue for any pending requests.

Centralized Solution

Queue

P_1

P_2



thanks paul krzyzanowski rutgers

Distributed Solution (Lamport '78)

- Instead of a central coordinator, all processes collectively
- Use similar approach:
  - Process sends request to all processes and put request in local queue.
  - On receipt of request, process sends back reply.
  - Process accesses resource
    - On receipt of all replies
    - Own request at head of queue
  - Once done, process sends release to all processes.
  - On receipt of release, process removes request

Distributed Solution

- Does this work (Lamport original solution)?
- Need to order queues so they are identical:
  - Use logical Lamport time + proc id to break ties.
  - FIFO channels
- Requests are executed in causal order.

Quorums

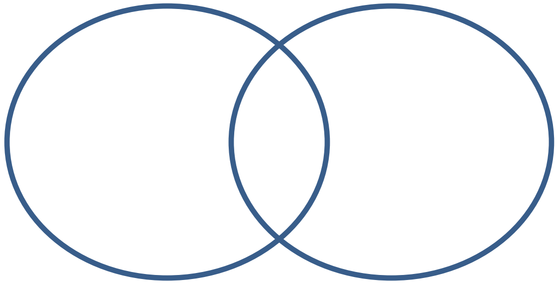
- What if there are failures?
- Do we need to communicate with ALL processes?
- Any two requests should have a common process to act as an arbitrator.
- Let process p_i (p_j)request permission from V_i (V_j), then

$$V_i \cap V_j \neq \phi.$$

- V_i is called a quorum.
- Basic protocol still works (basically think locking), but: Deadlock

Quorums

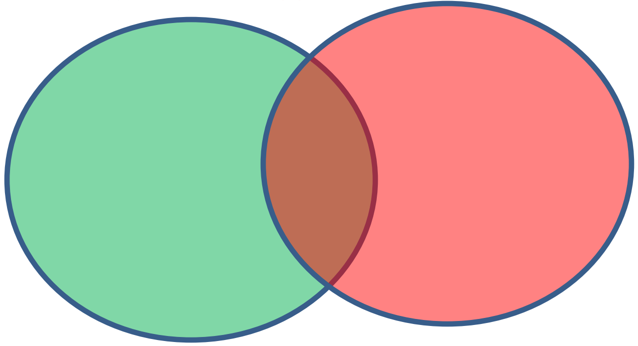
- Given n processes: $2|V_i| > n$, ie,

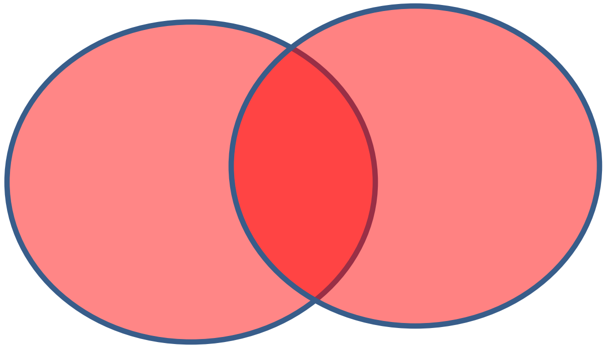


- In general, majority, ie ⌈(n/2)⌉. [Gifford 79]

General Quorums

- In a database context, we have read and write operations. Hence, read quorums, Q_r, and write quorums, Q_w.
- Simple generalization:
  - $Q_r \cap Q_w \neq \varphi$, $Q_w \cap Q_w \neq \varphi$
  - $Q_r + Q_w > n$ and $2 Q_w > n$





CONSENSUS AND BYZANTINE AGREEMENT

Consensus or Byzantine Agreement

- The Story (Lamport, Shostak, and Pease in 1982)
- Malicious Failures (byzantine failures)
- General sends an binary value to n-1 participants such that:

1. Agreement: All correct participants agree on same value
2. Validity: If general is correct, every participant agrees on the value general sends

General Impossibility Result

- In a synchronous distributed system:

No solution with fewer than 3f+1 processes can cope with f failures

Paxos

- Lamport the archeologist and the “Part-time Parliament” of Paxos:
  - The Part-time Parliament, TOCS 1998
  - Paxos Made Simple, *ACM SIGACT News 2001.*
  - Paxos Made Live, PODC 2007
  - Paxos Made Moderately Complex, (Cornell) 2011.
  - ……..

The Paxos Atomic Broadcast Algorithm

Thanks to Idit Keidar for slides

- Leader based: each process has an estimate of who is the current leader
- To order an operation, a process sends it to current leader
- The leader sequences the operations and launches a Consensus algorithm to ensure agreement

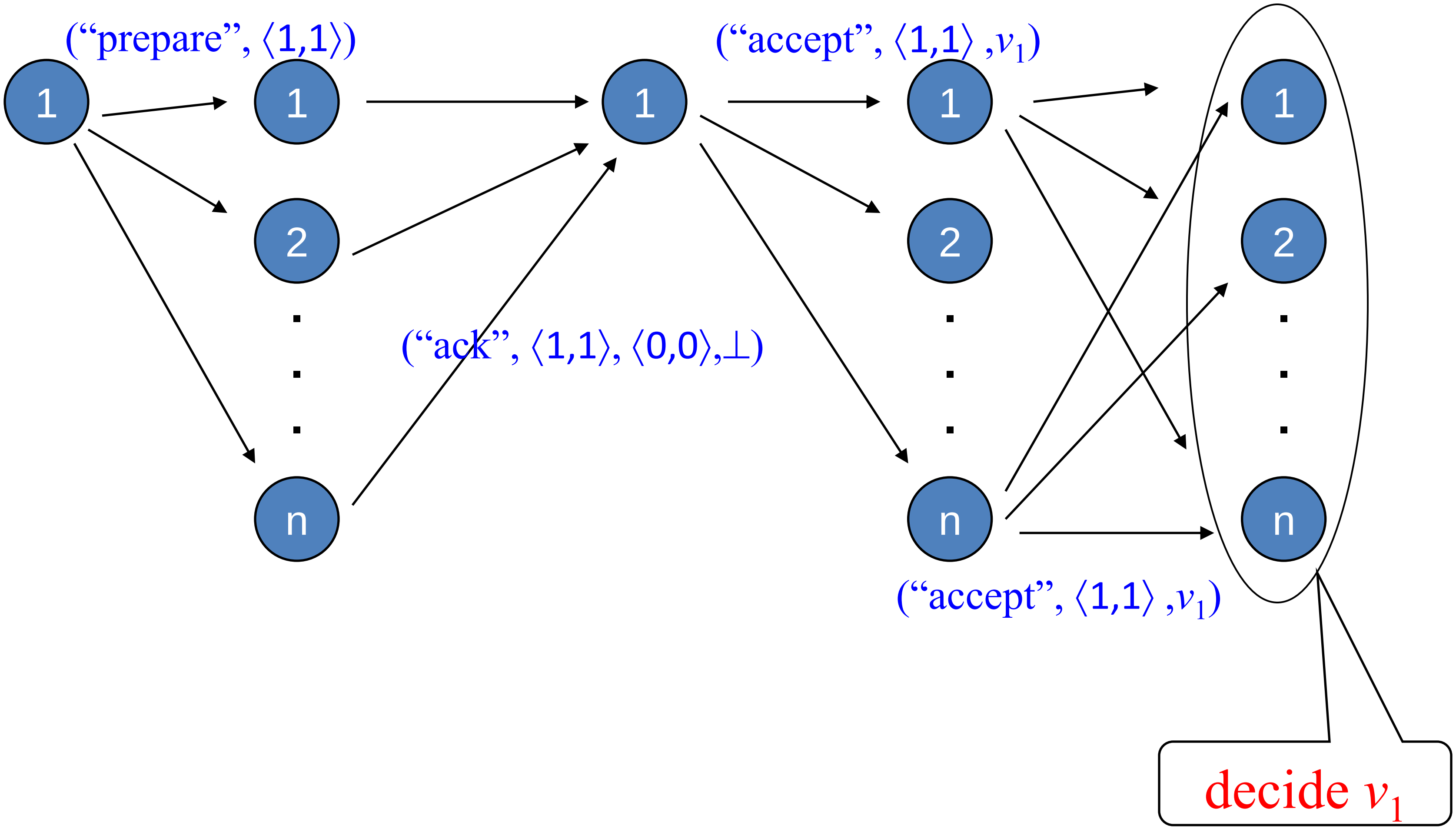
The Consensus Algorithm Structure

- Two phases
- Leader contacts a majority in each phase
- There may be multiple concurrent leaders
- *Ballots* distinguish among values proposed by different leaders
  - Unique, locally monotonically increasing
  - Processes respond only to leader with highest ballot seen so far

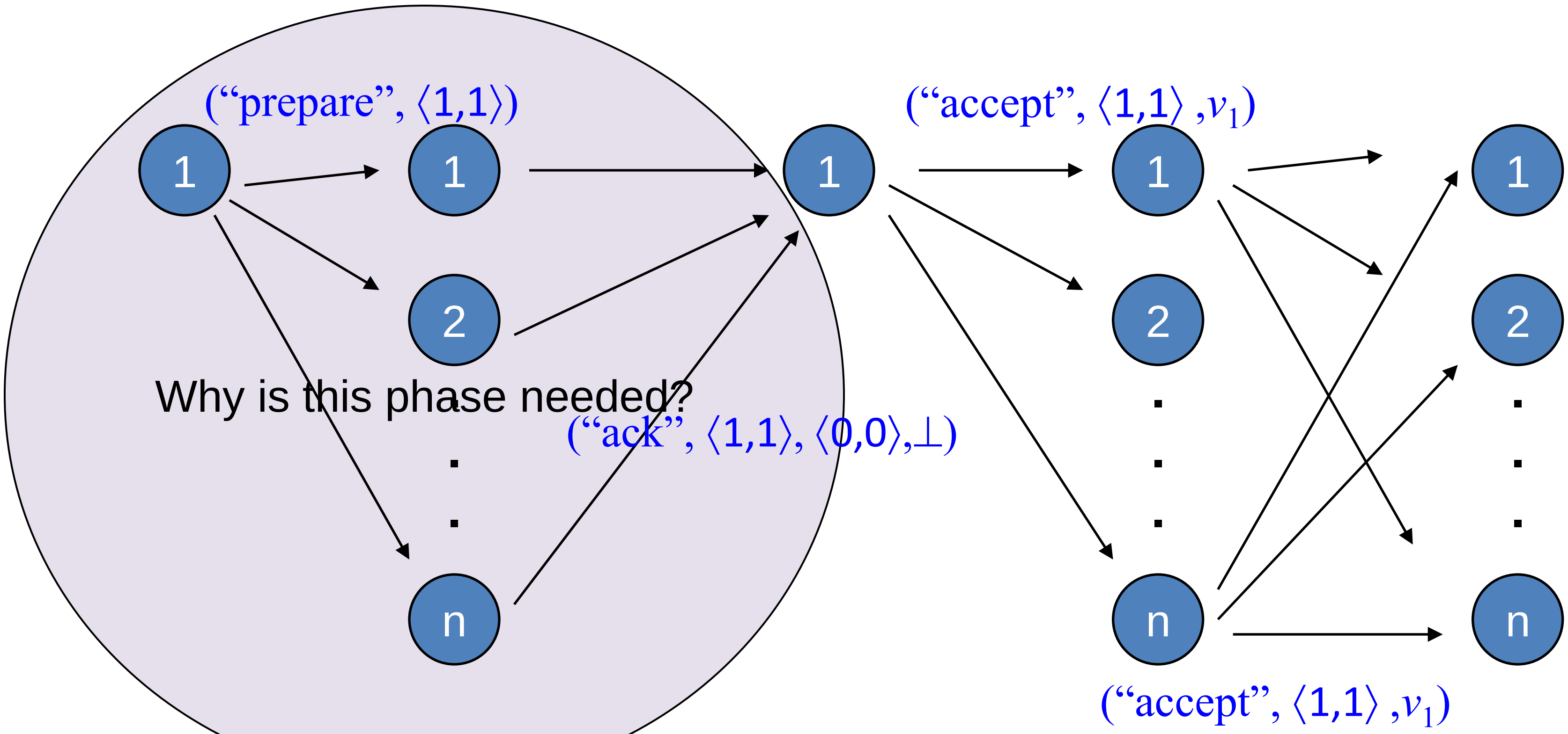
The Two Phases of Paxos

- Phase 1: prepare
  - If you *believe you are the leader*
    - Choose new unique ballot number
    - Learn outcome of all smaller ballots from majority
- Phase 2: accept
  - Leader *proposes* a value with its ballot number
  - Leader gets majority to *accept* its proposal
  - A value accepted by a majority can be decided

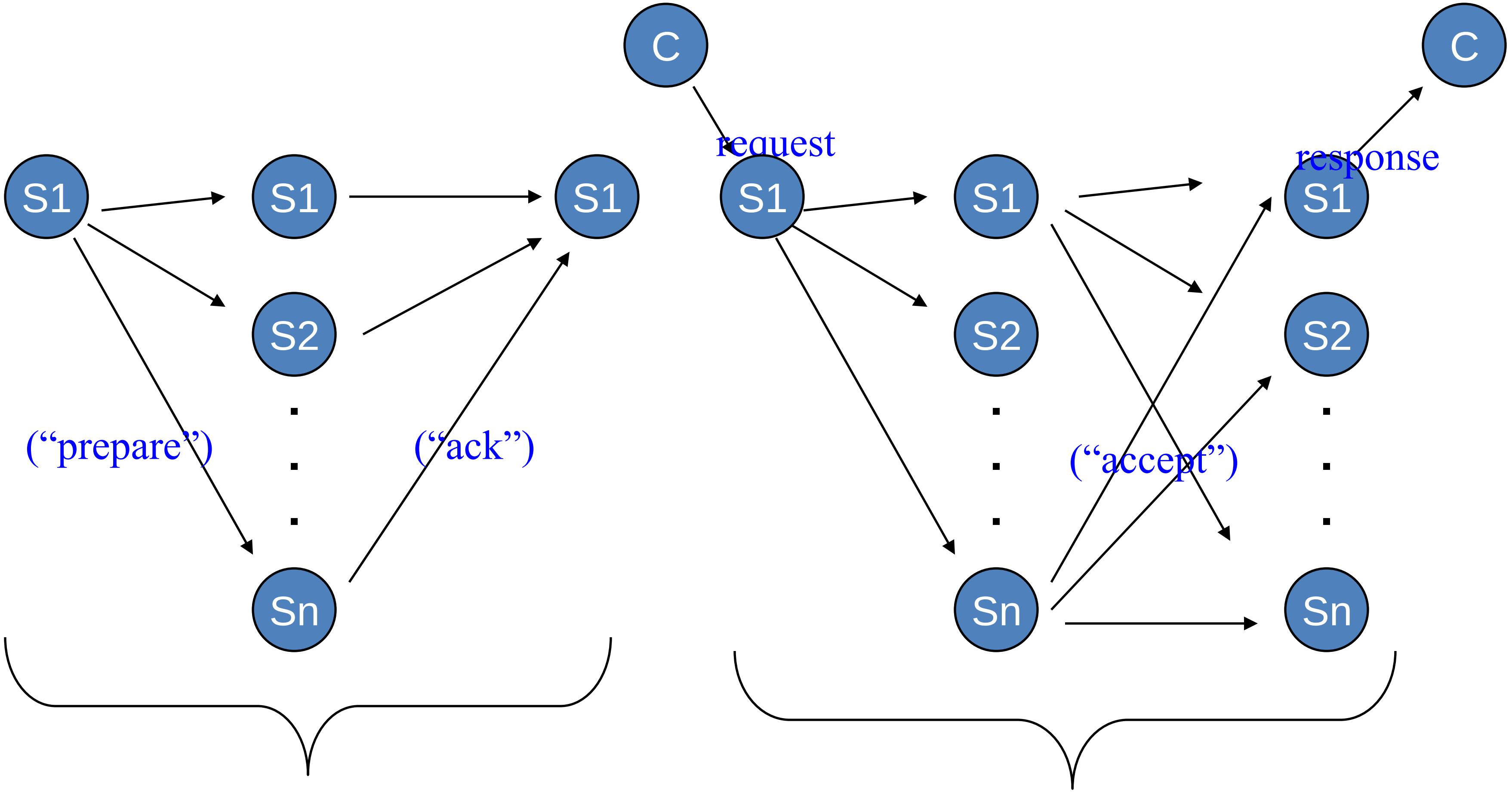
In Failure-Free Execution



Performance?



Failure free execution



Phase 1

Phase 2

Optimization

- Run Phase 1 only when the leader changes
  - Phase 1 is called “view change” or “recovery mode”
  - Phase 2 is the “normal mode”
- Each message includes BallotNum (from the last Phase 1) and ReqNum
- Respond only to messages with the “right” BallotNum

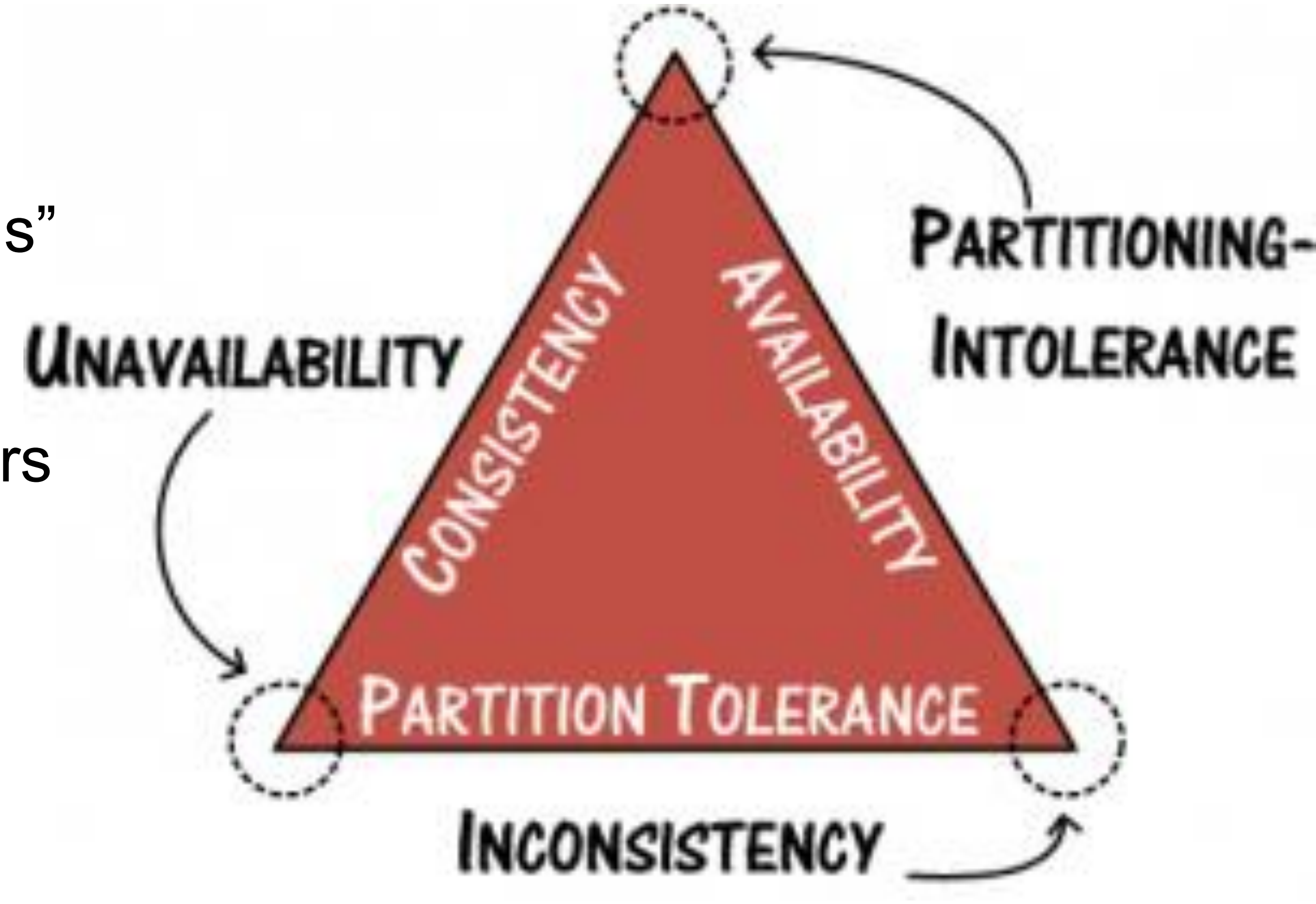
FLP Impossibility Theorem

- In an asynchronous system, consensus is impossible to solve if one process may crash and processes communicate by message passing.
- Proved by Fisher, Lynch and Paterson in PODS 1983 and who won the Dijkstra Prize for this result.

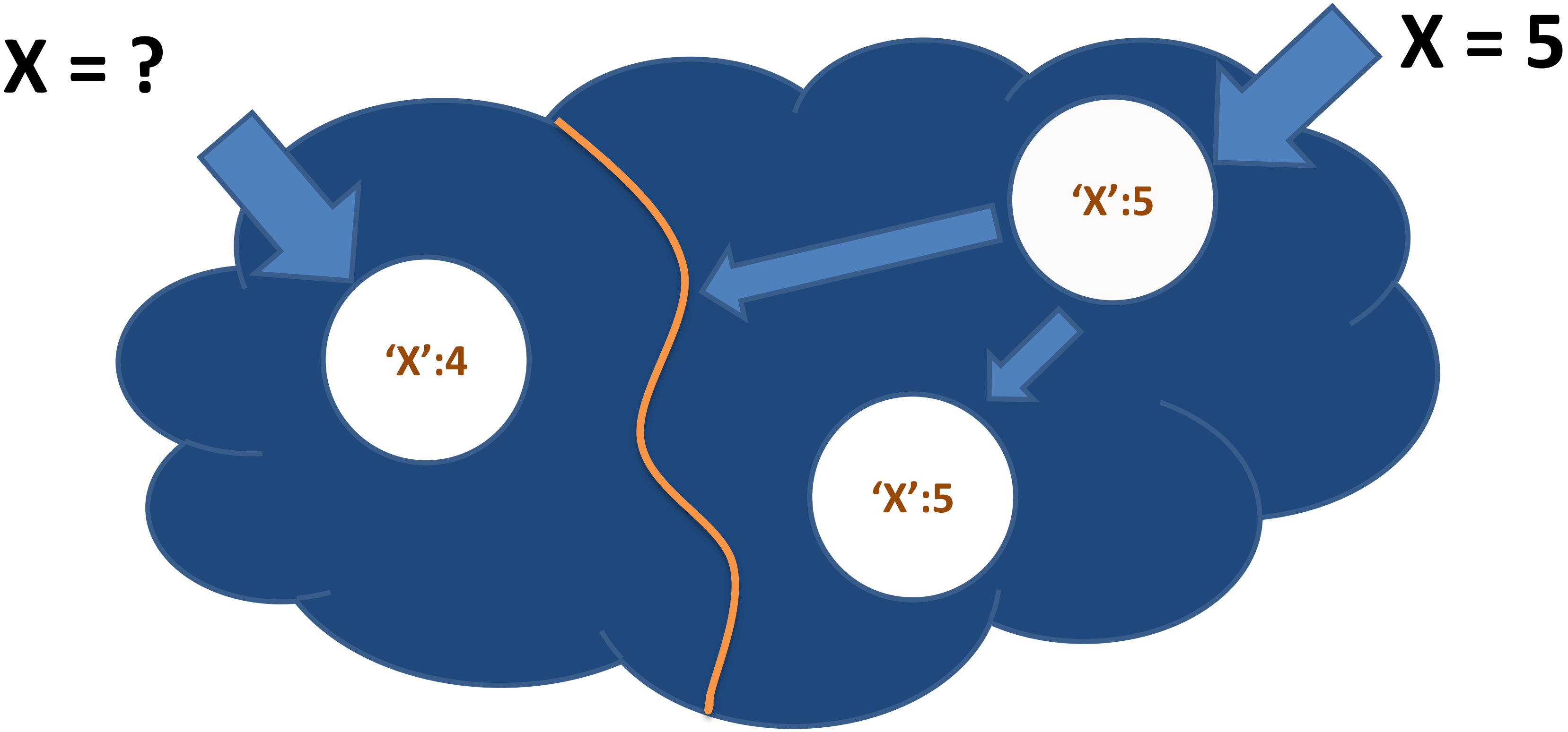
CAP Theorem (Eric Brewer)

- “Towards Robust Distributed Systems” PODC 2000.

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



CAP – Why P (A or C)?



If we choose **A**, then Eventual Consistency…

Why sacrifice Consistency?

- It is a simple solution
  - nobody understands what sacrificing P really means
  - sacrificing A is unacceptable in the Web
  - possible to push the problem to app developer
- C not needed in many applications
  - Banks do not implement ACID (classic example wrong)
  - Airline reservation only transacts reads (Huh?)
  - MySQL et al. ship by default in lower isolation level
- Data is noisy and inconsistent anyway
  - making it, say, 1% worse does not matter ☺

[Vogels, VLDB 2007]

PEER TO PEER AND DISTRIBUTED HASH TABLES

Distributed Hash Tables

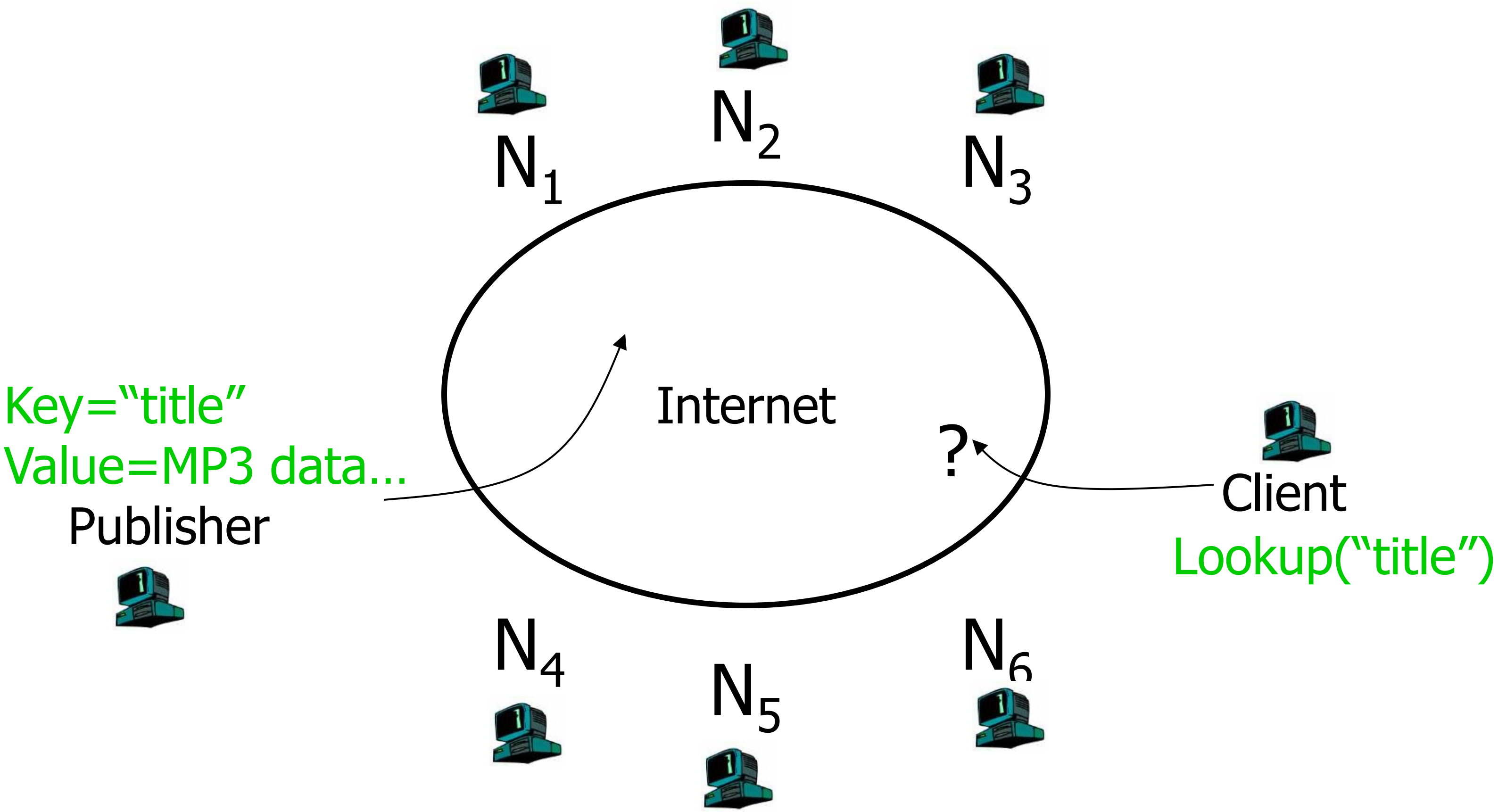
Challenge: To design and implement a robust and scalable distributed system composed of inexpensive, individually unreliable computers in unrelated administrative domains

Partial thanks Idit Keidar)

Searching for distributed data

- Goal: Make billions of objects available to millions of concurrent users
  - e.g., music files
- Need a distributed data structure to keep track of objects on different sires.
  - map object to locations
- Basic Operations:
  - Insert(key)
  - Lookup(key)

Searching



N_1

N_2

N_3

Internet

?

Key="title"
Value=MP3 data…
Publisher

Client
Lookup("title")

N_4

N_5

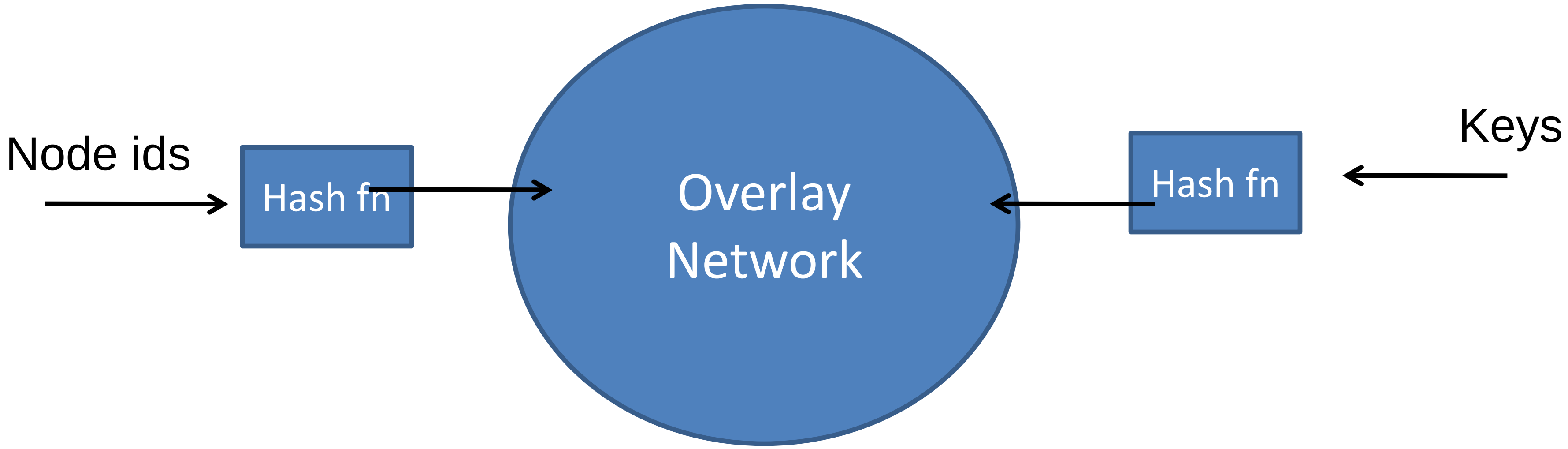
N_6

Simple Solution

- First There was Napster
  - Centralized server/database for lookup
  - Only file-sharing is peer-to-peer, lookup is not
- Launched in 1999, peaked at 1.5 million simultaneous users, and shut down in July 2001.

Overlay Networks

- A virtual structure imposed over the physical network (e.g., the Internet)
  - A graph, with hosts as nodes, and some edges



Node ids

Hash fn

Overlay Network

Hash fn

Keys

Unstructured Approach: Gnutella

- Build a decentralized unstructured overlay
  - Each node has several neighbors
  - Holds several keys in its local database
- When asked to find a key X
  - Check local database if X is known
  - If yes, return, if not, ask your neighbors
- Use a limiting threshold for propagation.

Structured vs. Unstructured

- The examples we described are *unstructured*
  - There is no systematic rule for how edges are chosen,
    each node "knows some" other nodes
  - Any node can store any data so a searched data might reside at any node
- *Structured overlay*:
  - The edges are chosen according to some rule
  - Data is stored at a pre-defined place
  - Tables define next-hop for lookup

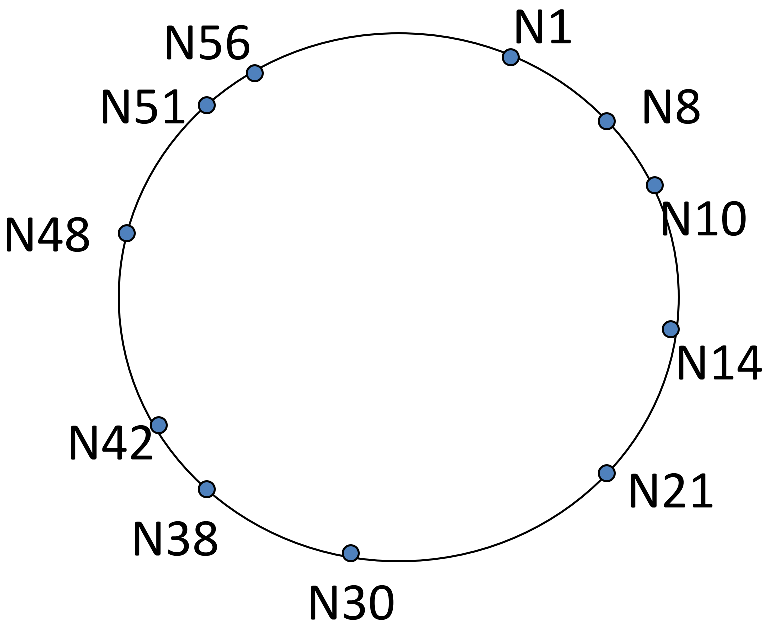
Distributed Hash Tables (DHTs)

- Nodes store table entries
- **lookup(key)** returns the location of the node currently responsible for this **key**

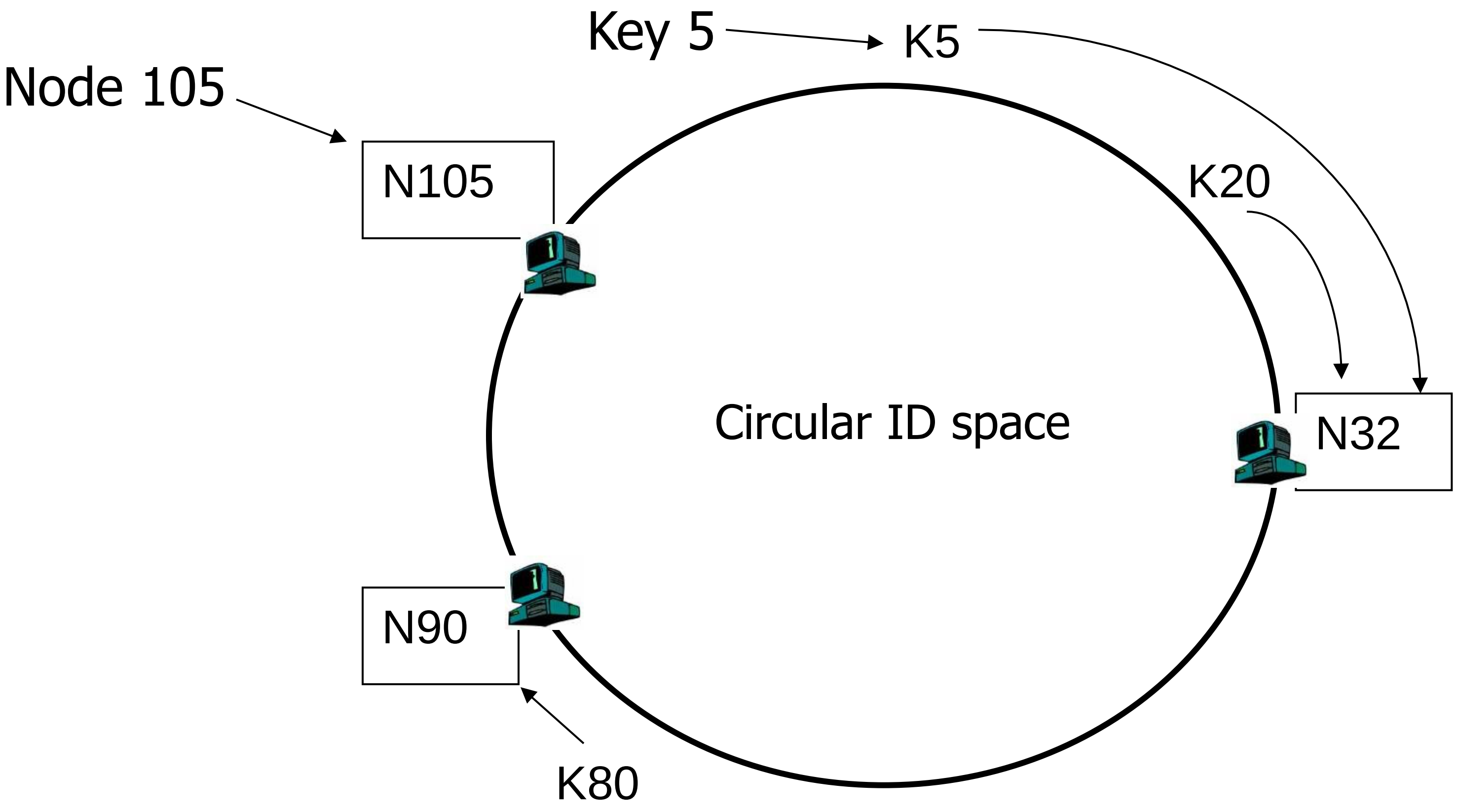
- We will discuss Chord, Stoica, Morris, Karger, Kaashoek, and Balakrishnan SIGCOMM 2001
- **Other examples:** CAN (Berkeley), Tapestry (Berkeley), Pastry (Microsoft Cambridge), etc.

Chord Logical Structure (MIT)

- *m*-bit ID space (2^m IDs), usually *m*=160.
- Nodes organized in a logical ring according to their IDs.



DHT: Consistent Hashing



Key 5

K5

Node 105

N105

K20

Circular ID space

N32

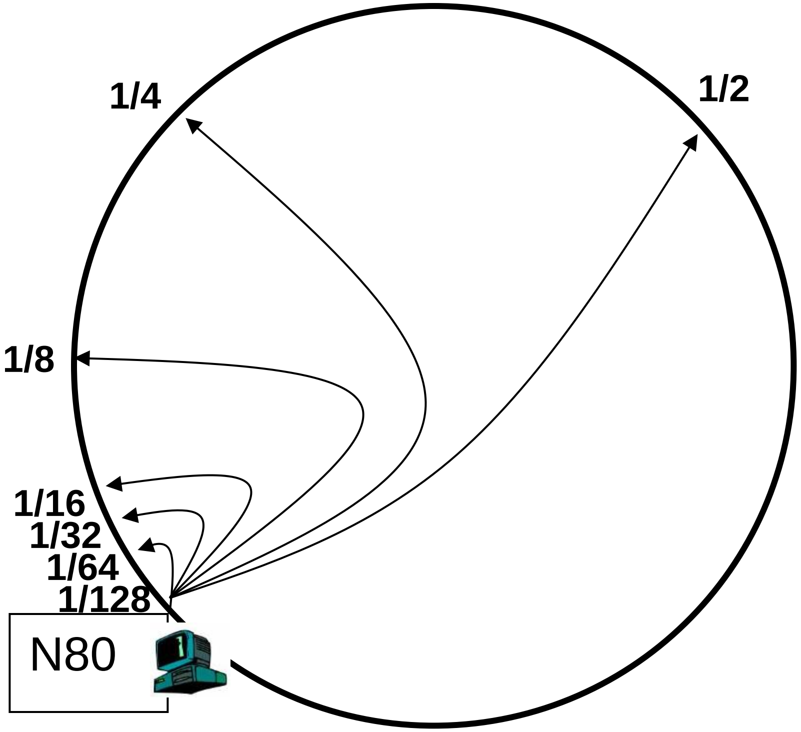
N90

K80

A key is stored at its successor: node with next higher ID

Thanks CMU for animation

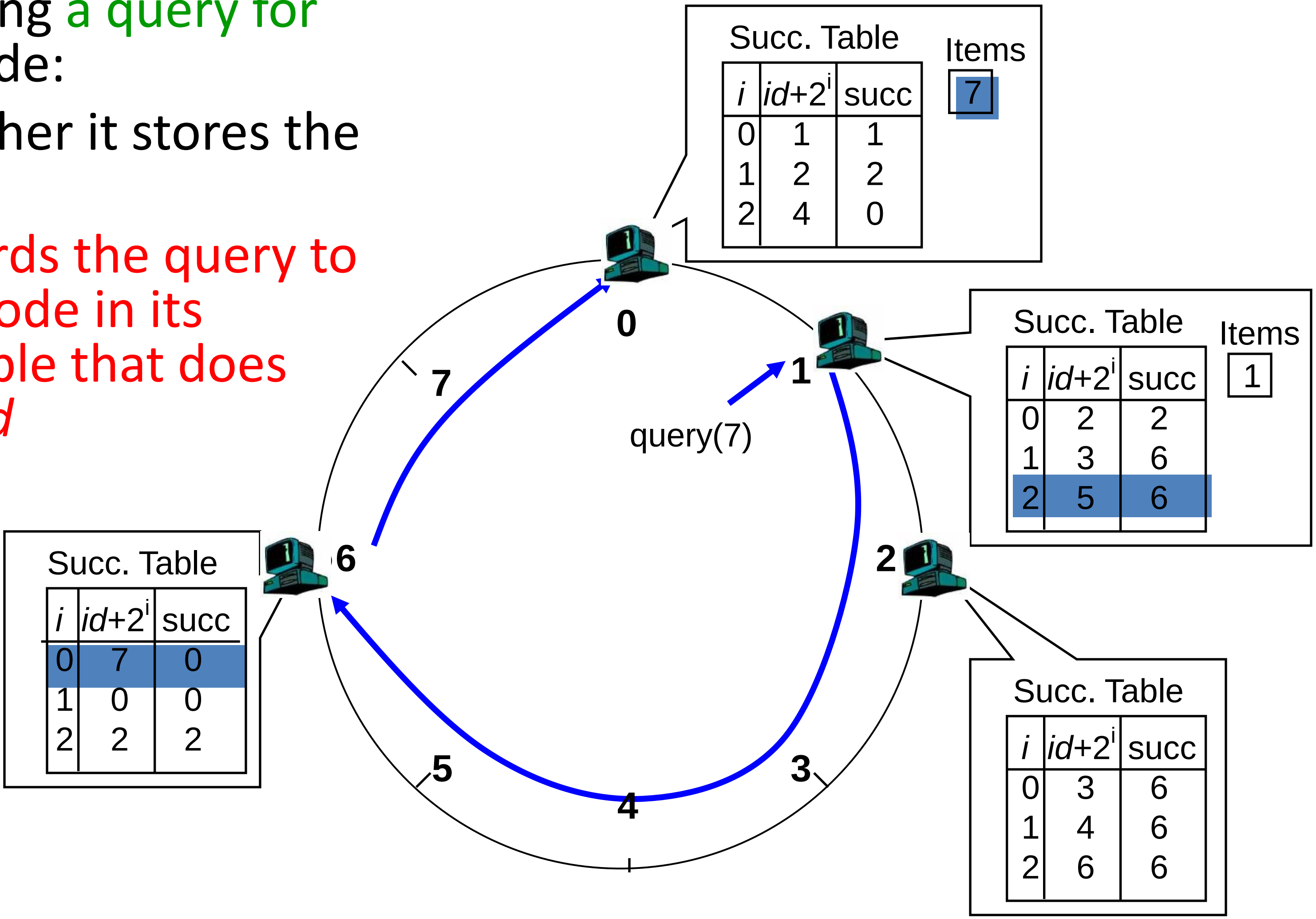
DHT: Chord "Finger Table"



- Entry *i* in the finger table of node *n* is the first node that succeeds or equals $n + 2^i$
- In other words, the ith finger points $1/2^{n-i}$ way around the ring

DHT: Chord Routing

- Upon receiving a query for item *id*, a node:
- Checks whether it stores the item locally?
- If not, forwards the query to the largest node in its successor table that does not exceed *id*



Node 0 — Succ. Table

i	*id*+2^i	succ
0	1	1
1	2	2
2	4	0

Items

7

Node 1 — Succ. Table

i	*id*+2^i	succ
0	2	2
1	3	6
2	5	6

Items

1

Node 2 — Succ. Table

i	*id*+2^i	succ
0	3	6
1	4	6
2	6	6

Node 6 — Succ. Table

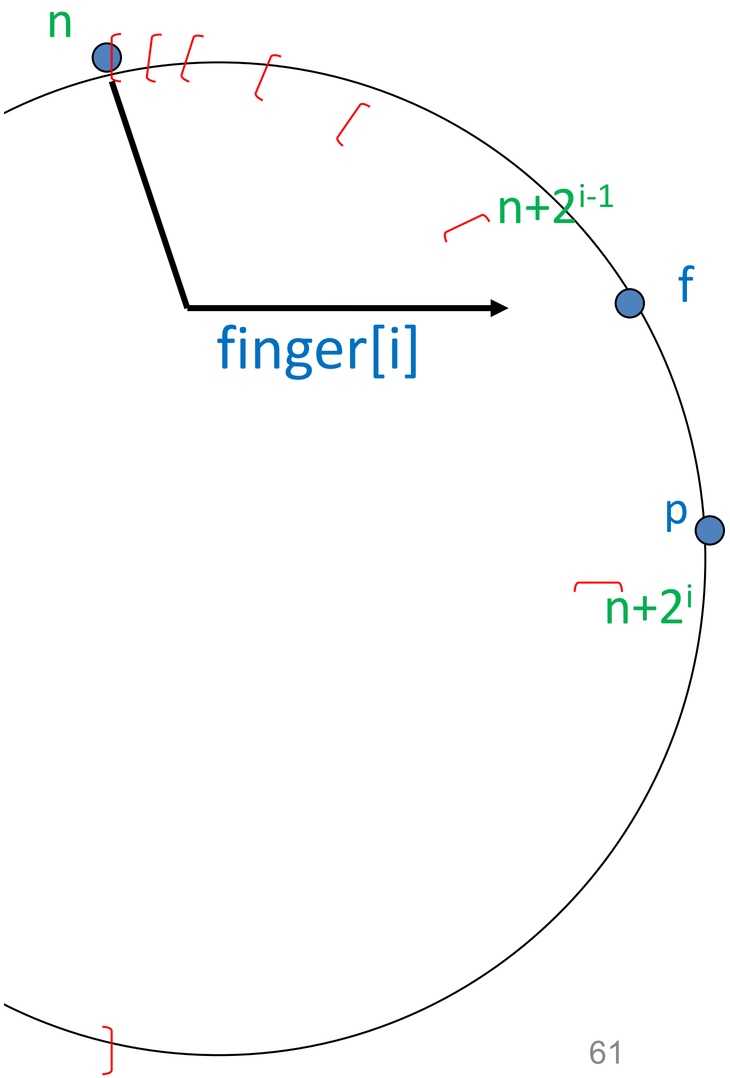
i	*id*+2^i	succ
0	7	0
1	0	0
2	2	2

Routing Time

- Node *n* looks up a key stored at node *p*
- *p* is in *n*'s *i*th interval:

  $p \in ((n+2^{i-1}) \bmod 2^m, (n+2^i) \bmod 2^m]$
- **n** contacts *f=finger[i]*
  - The interval is not empty so:

    $f \in ((n+2^{i-1}) \bmod 2^m, (n+2^i) \bmod 2^m]$
- *f* is at least 2^{i-1} away from *n*
- *p* is at most 2^{i-1} away from *f*
- The distance is halved at each hop.



The Transaction Concept

- Transactions were originally developed in the context of DBMS as a paradigm to deal with:
  - Concurrent access to shared data
  - Failures of different kinds/types.

- The key problem solved in an elegant manner:
  - Subtle and difficult issue of keeping data consistent in the presence of concurrency and failures

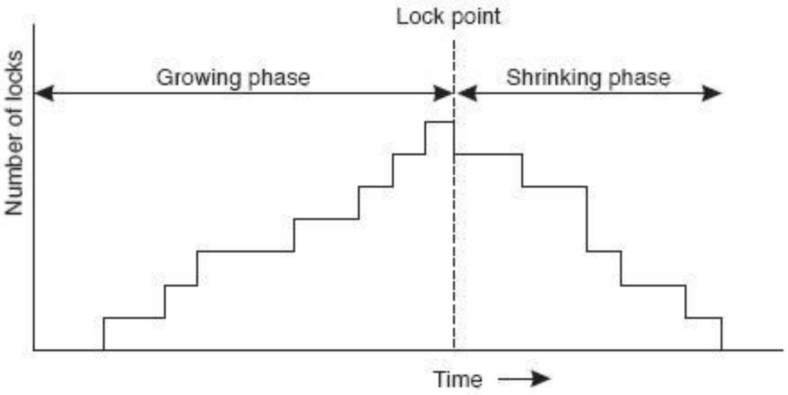
while ensuring performance, reliability, and availability.

Preliminaries: A database

- A **database** consists of a set of objects.
- A **transaction** is a set of operations (typically read and write) executed in some partial order.
- Transaction execution must be atomic:
  - no interference among transactions.
  - Either all its operations are executed or none.
- **Concurrency control protocol** ensures that concurrent transactions do not interfere with each other.
- **Recovery protocol** ensures the all or nothing property.

Concurrency Control

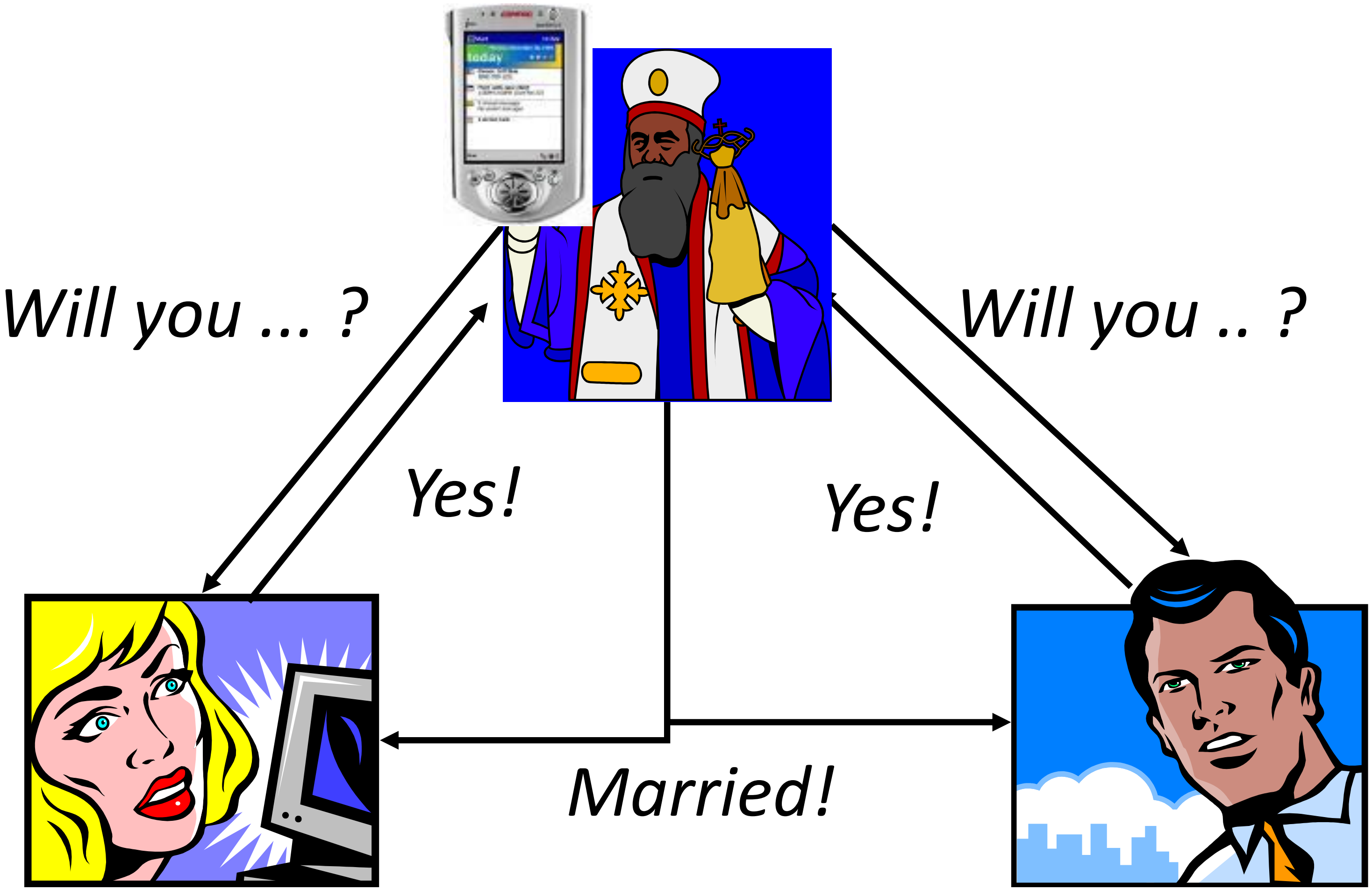
- A history is **serializable** if it is equivalent to a serial history over the same set of transactions.
- Different notions of serializability:
  - View Serializability: NP Complete ☹
  - Conflict Serializability: H is CSR iff SG(H) is acyclic
- Two Phase locking.
  - deadlock



Atomic Commitment

- Distributed handshake protocol known as two-phase commit (2PC):
  - A coordinator (the Transaction Manager) takes the responsibility of unanimous decision: COMMIT or ABORT
  - All database servers are the cohorts in this protocol and become dependent on the coordinator

Idea: Getting Married over the NW



Will you ... ?

Will you .. ?

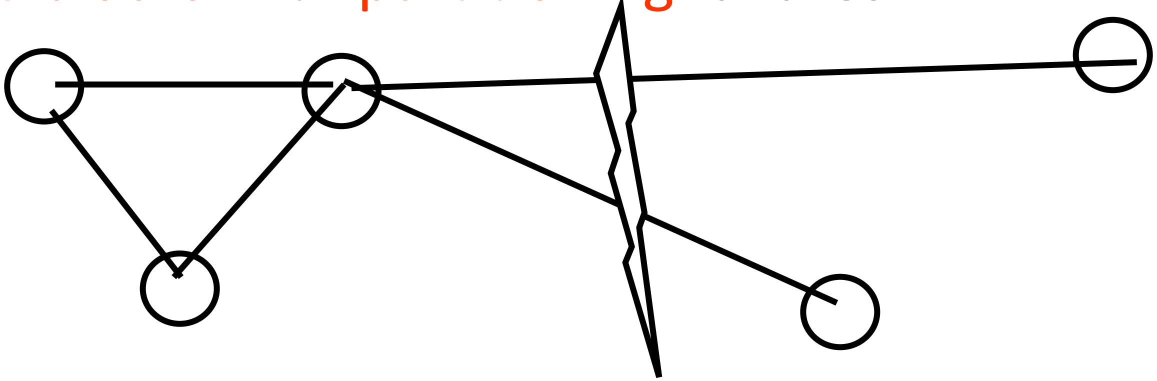
Yes!

Yes!

Married!

Commit Protocols

- What does a process do if it does not receive a message it is expecting? It BLOCKS.
- 2 PC blocks with failures
- 3PC is non-blocking with site failures only.
- 3PC blocks with partitioning failures.

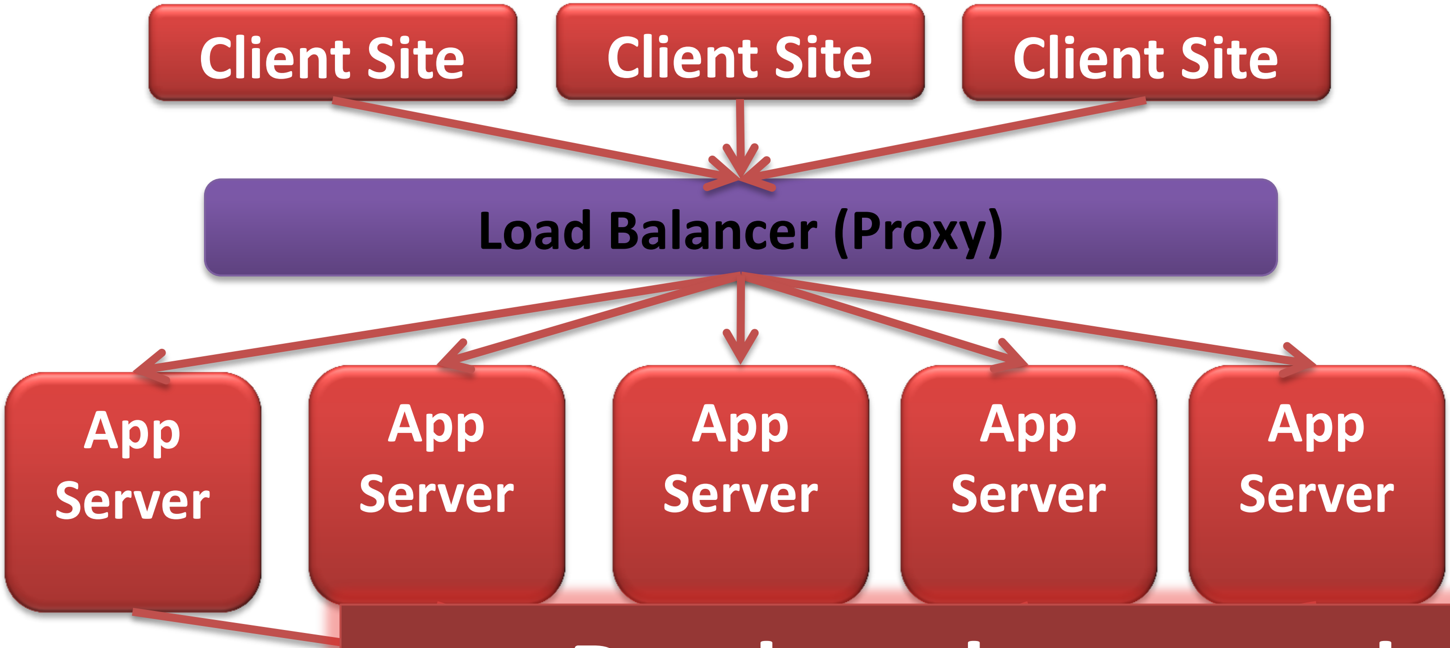


Partition 1 Partition 2

- Theorem [Skeen83]: There is no non-blocking atomic commit protocol in the presence of partitioning failures.

Cloud Reality: The Data Centers

Scaling in the Cloud



Client Site

Client Site

Client Site

Load Balancer (Proxy)

App Server

App Server

App Server

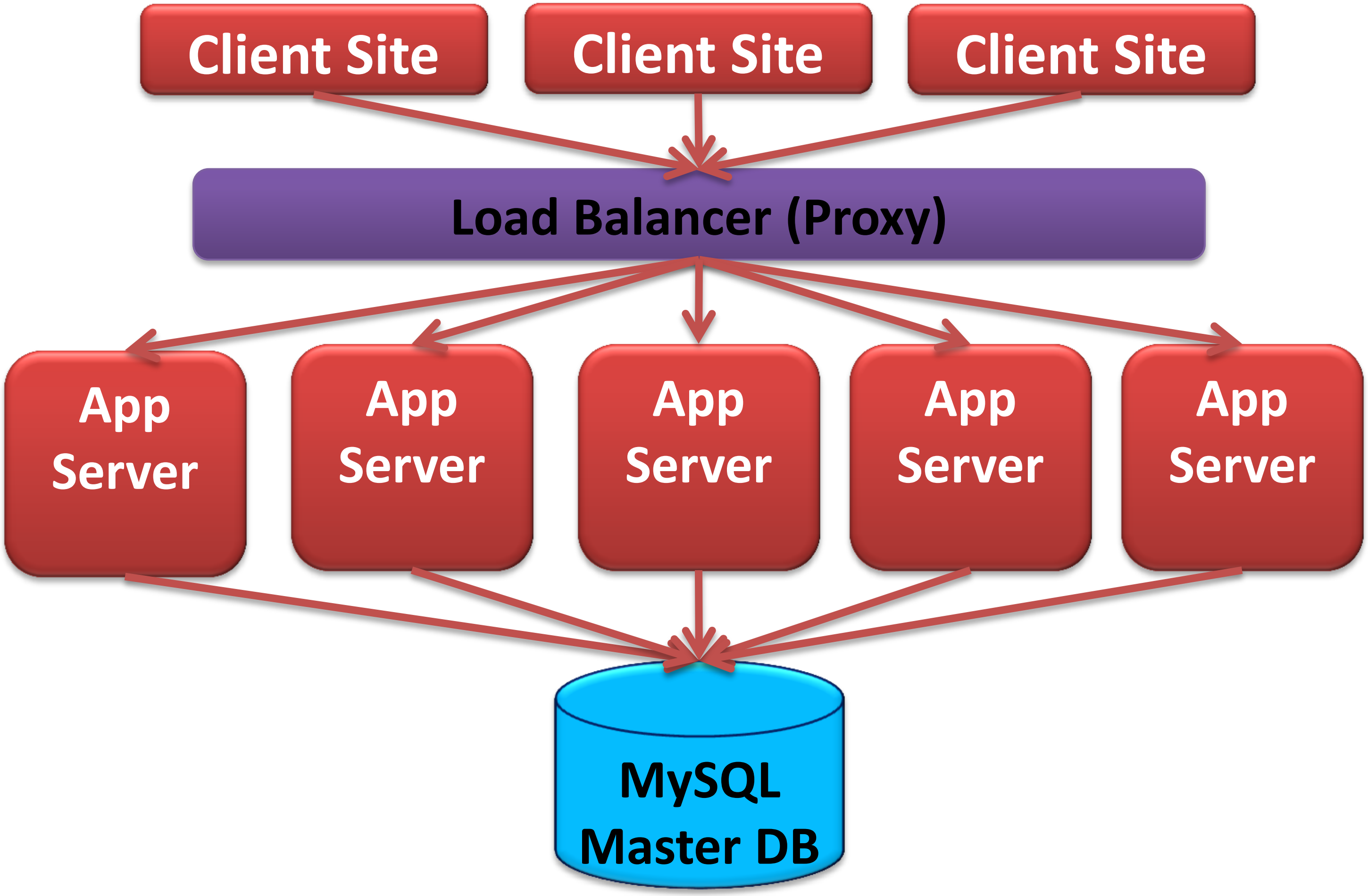
App Server

App Server

Database becomes the Scalability Bottleneck

Cannot leverage elasticity

Scaling in the Cloud



Client Site

Client Site

Client Site

Load Balancer (Proxy)

App Server

App Server

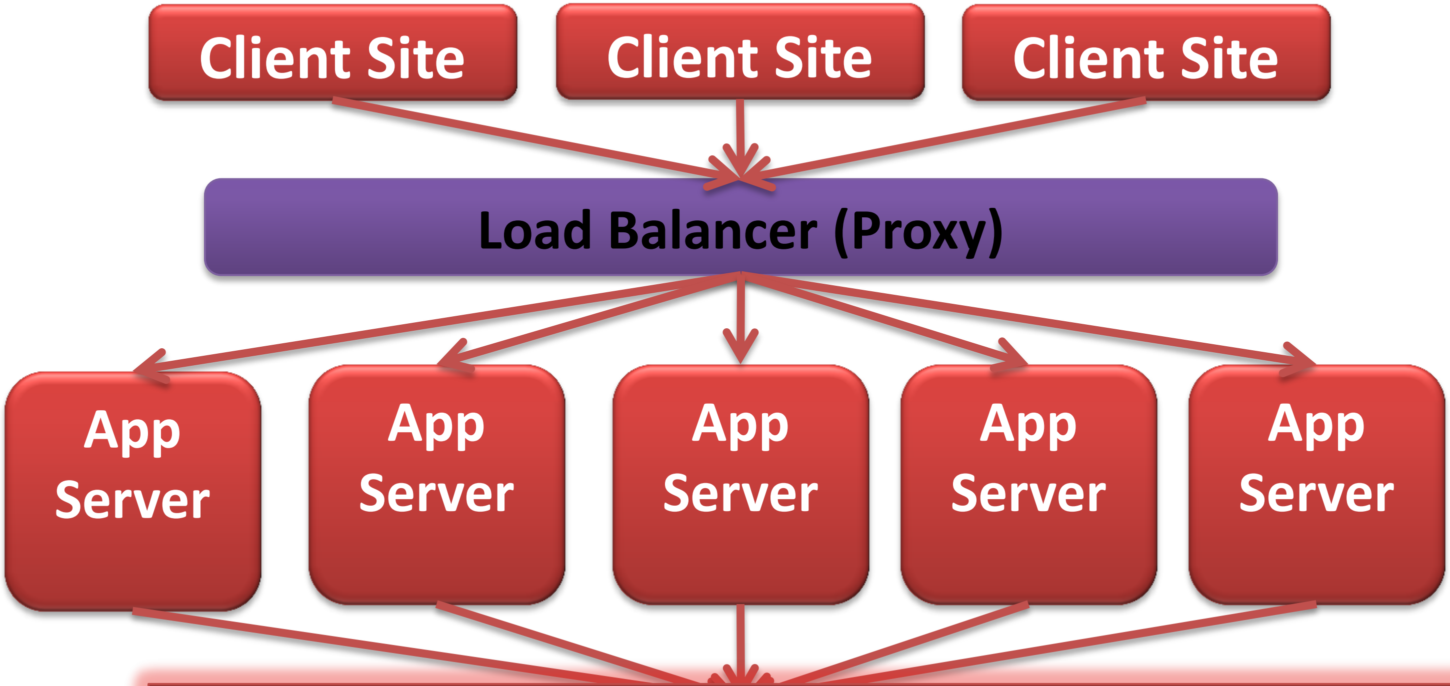
App Server

App Server

App Server

MySQL Master DB

Scaling in the Cloud



Client Site

Client Site

Client Site

Load Balancer (Proxy)

App Server

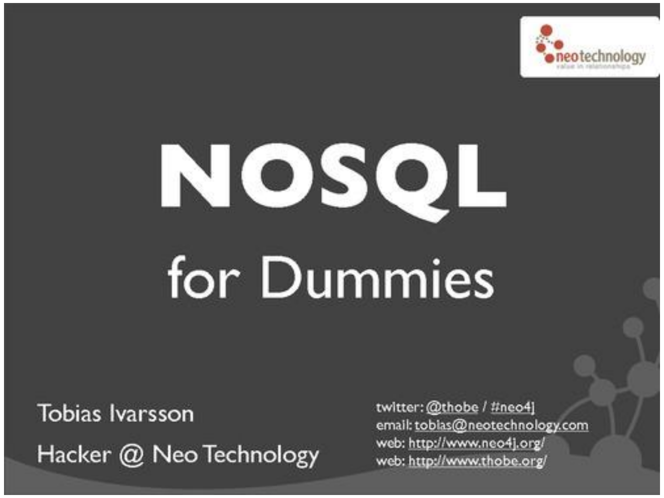
App Server

App Server

App Server

App Server

Scalable and Elastic, but limited consistency and operational flexibility



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/

Key Value Stores

- Key-Valued data model
  - Key is the unique identifier
  - Key is the granularity for consistent access
  - Value can be structured or unstructured
- Gained widespread popularity
  - In house: **Bigtable** (Google), **PNUTS** (Yahoo!), **Dynamo** (Amazon)
  - Open source: **HBase, Hypertable, Cassandra, Voldemort**
- Popular choice for the modern breed of web-applications

Big Table (Google)

- Data model.
  - Sparse, persistent, multi-dimensional sorted map indexed by a row key, column key, and a timestamp.
  - (row: byte[ ], column: byte[ ], time: int64) → byte[ ]
- Scalability and Elasticity: Data is partitioned across multiple servers.

Atomicity Guarantees in Key-Value Stores

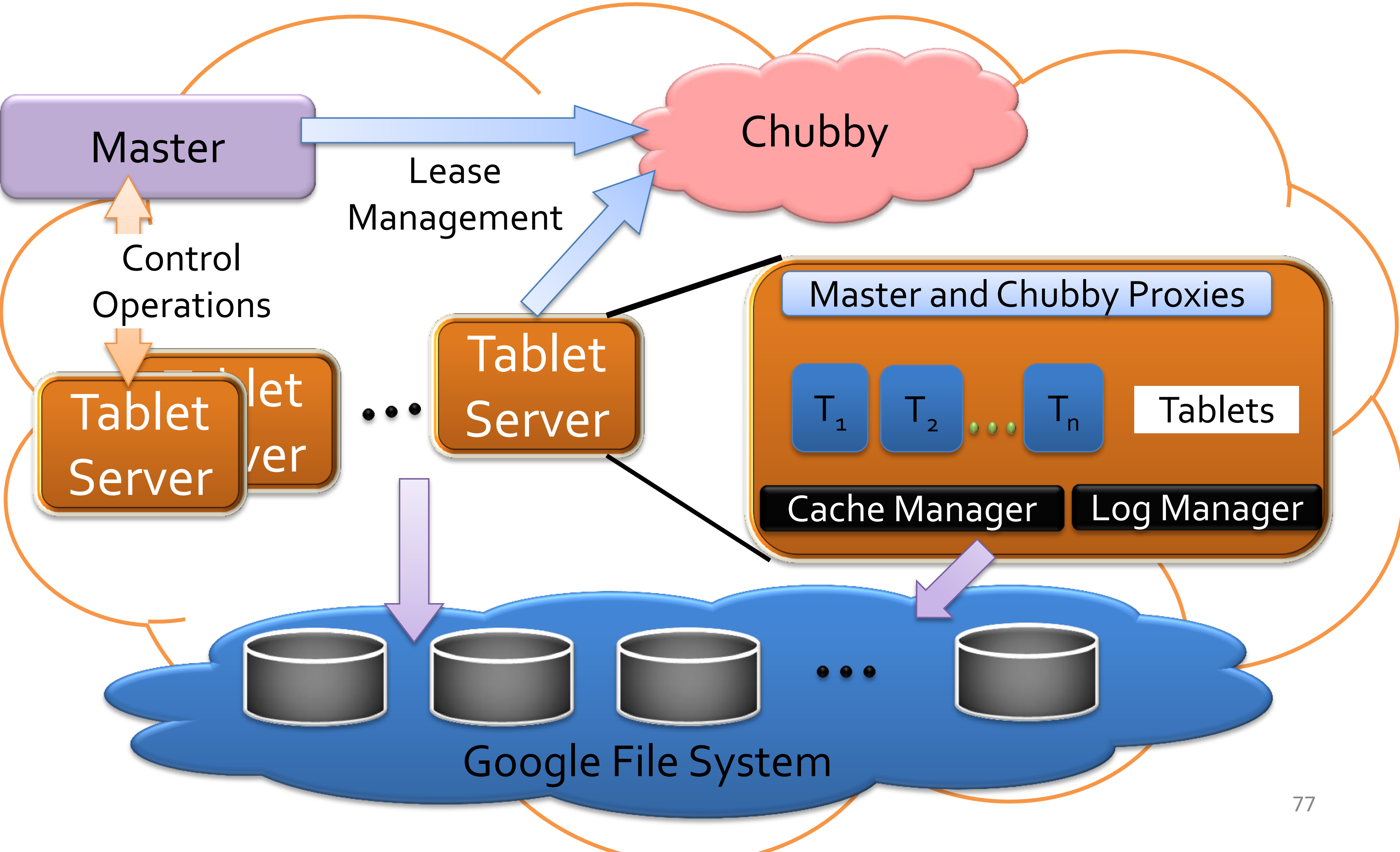
- Every read or write of data under a single row is atomic.

- **Objective**: make read operations single-sited!

Big Table's Building Blocks

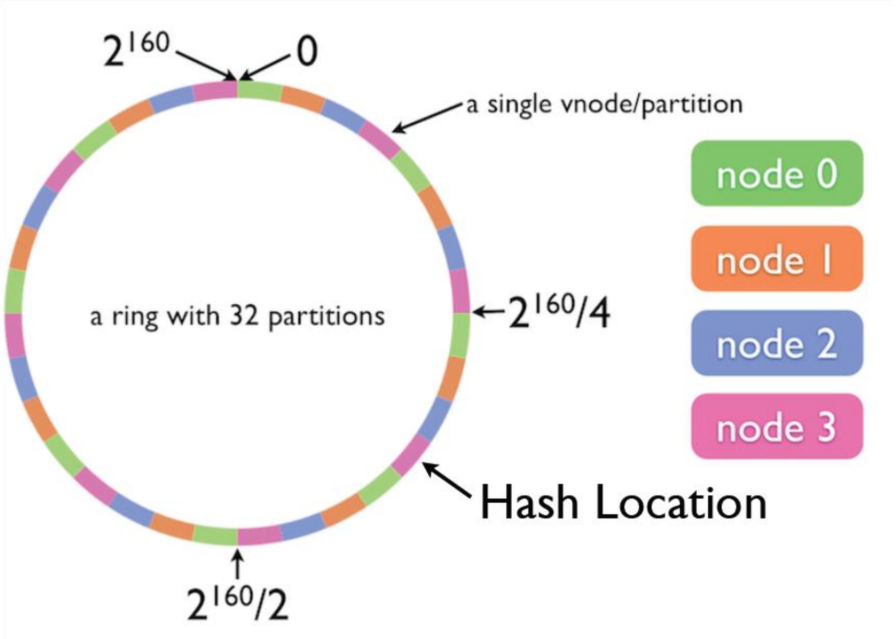
- Tablet servers
  - Handles read and writes to its tablet and splits tablets
  - Each tablet is typically 100-200 MB in size
- Master Server
  - Assigns tablets to tablet servers
  - Detects the addition and deletion of tablet servers
  - Balances tablet-server load
- Google File System (GFS)
  - Highly available distributed file system that stores log and data files
- Chubby
  - Manage meta-data
  - Highly available persistent distributed lock manager

Overview of Bigtable Architecture



Dynamo (Amazon) and Cassandra (Facebook)

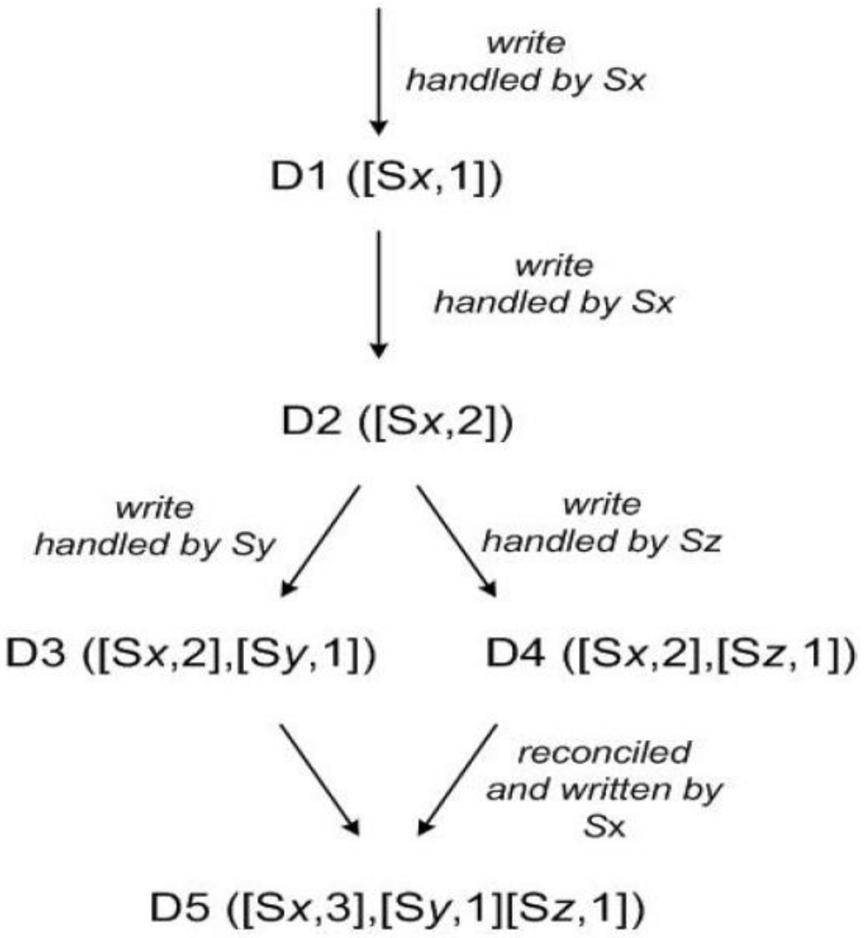
- Consistent hashing: the output range of a hash function is treated as a fixed circular space or "ring" a la Chord.
- "Virtual Nodes": Each node can be responsible for more than one virtual node (to deal with non-uniform data and load distribution)



Sloppy Quorum

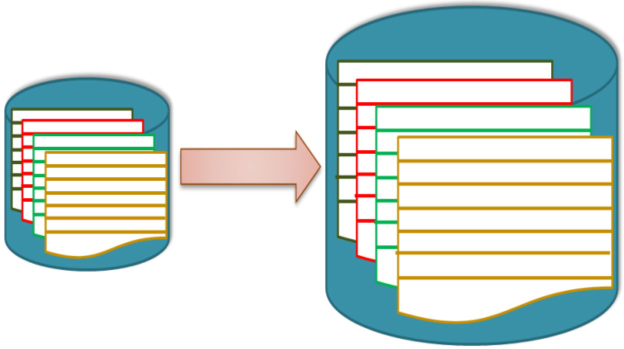
- R and W is the minimum number of nodes that must participate in a successful read/write operation.
- Setting R + W > N yields a quorum-like system.
- Operation latency dictated by the slowest of t replicas. For this reason, R and W are usually configured to be less than N, to provide better latency and availability.
- Use vector clocks in order to capture causality between different versions of same object
- Application reconciles divergent versions and collapses into a single new version.

Vector clock example

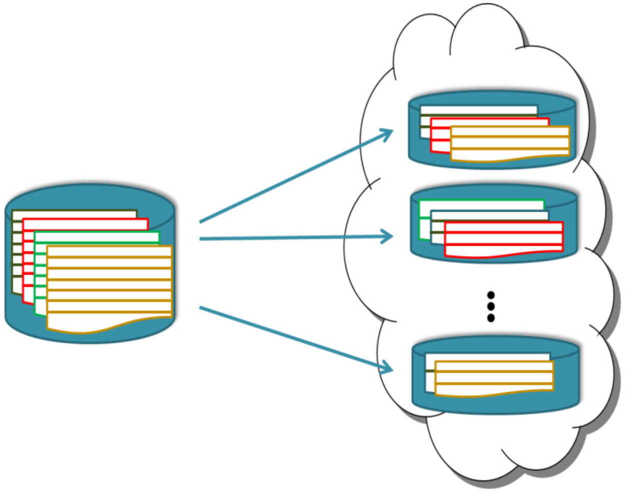


Practical approaches to scalability Circa Year 2000.

- **Scale-up**
  - **Classical enterprise** setting (RDBMS)
  - Flexible **ACID transactions**
  - Transactions in a single node



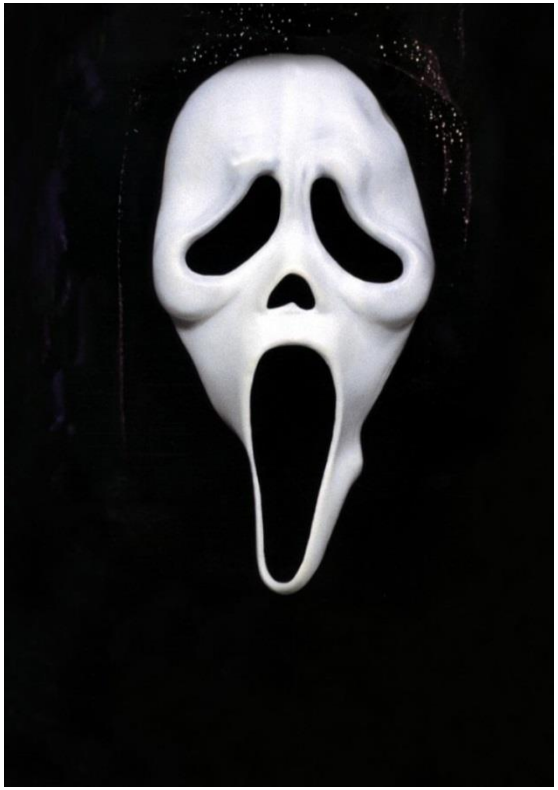
- **Scale-out**
  - **Cloud friendly** (Key value stores)
  - Execution at a single server
    - Limited functionality & guarantees
  - No **multi-row** or **multi-step** transactions



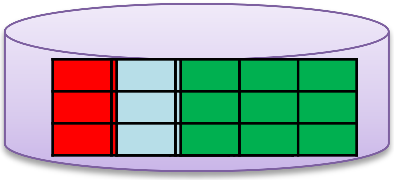
Distribution & Consistency

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

NoSQL is apparently NOT going to deliver World Peace



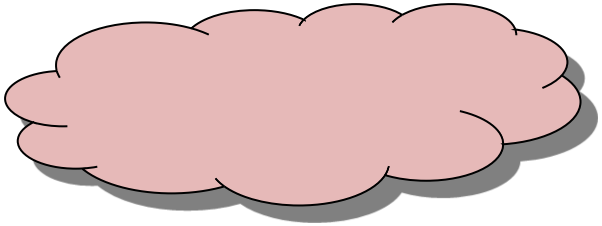
Supporting SQL in the Cloud



RDBMS

Fission

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



Key Value Stores

Fusion

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

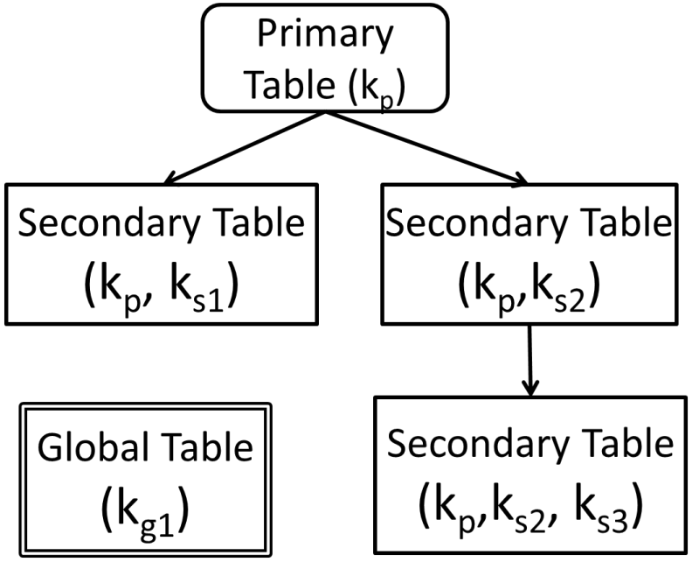
First Gen Data Center Systems

- ➔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



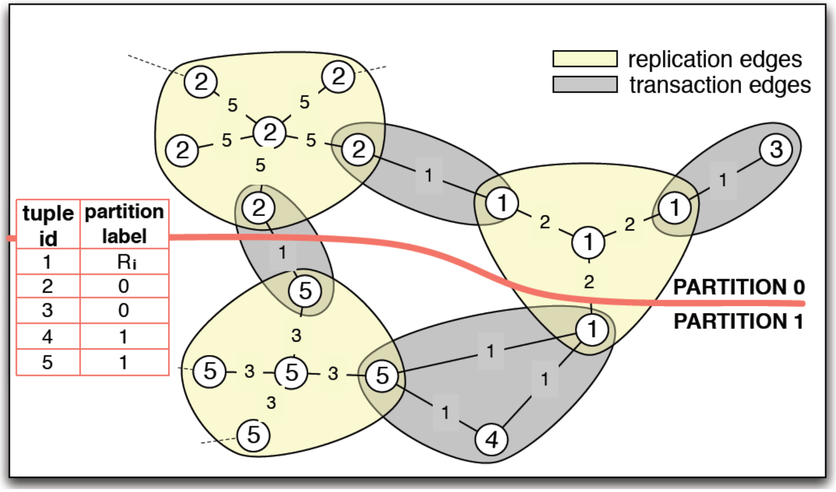
Primary Table (k_p)

Secondary Table (k_p, k_{s1})

Secondary Table (k_p,k_{s2})

Global Table (k_{g1})

Secondary Table (k_p,k_{s2}, k_{s3})

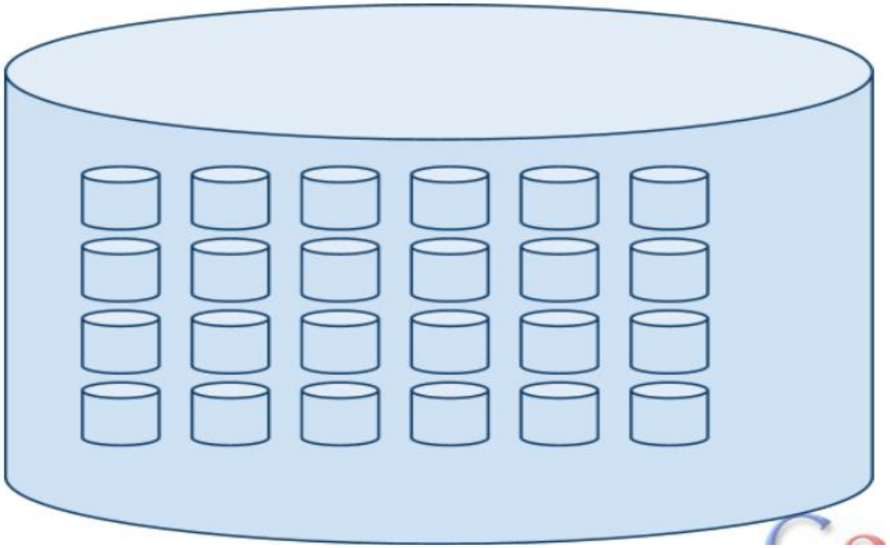


tuple id	partition label
1	R_i
2	0
3	0
4	1
5	1

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

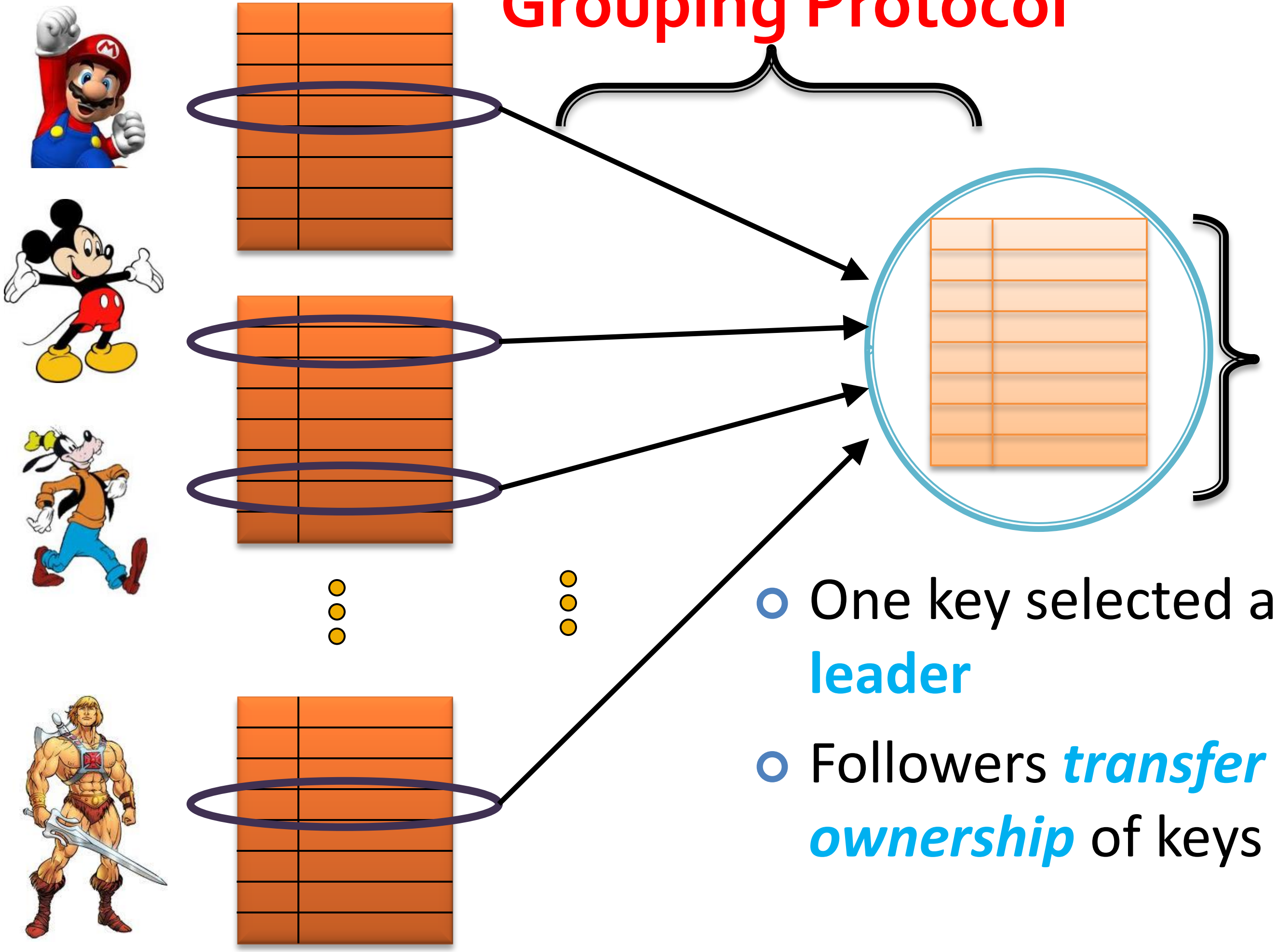
G-Store

UCSB Das et al. ACM SoCC'2010

- ***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***

Transactions on Groups

Without distributed transactions



Grouping Protocol

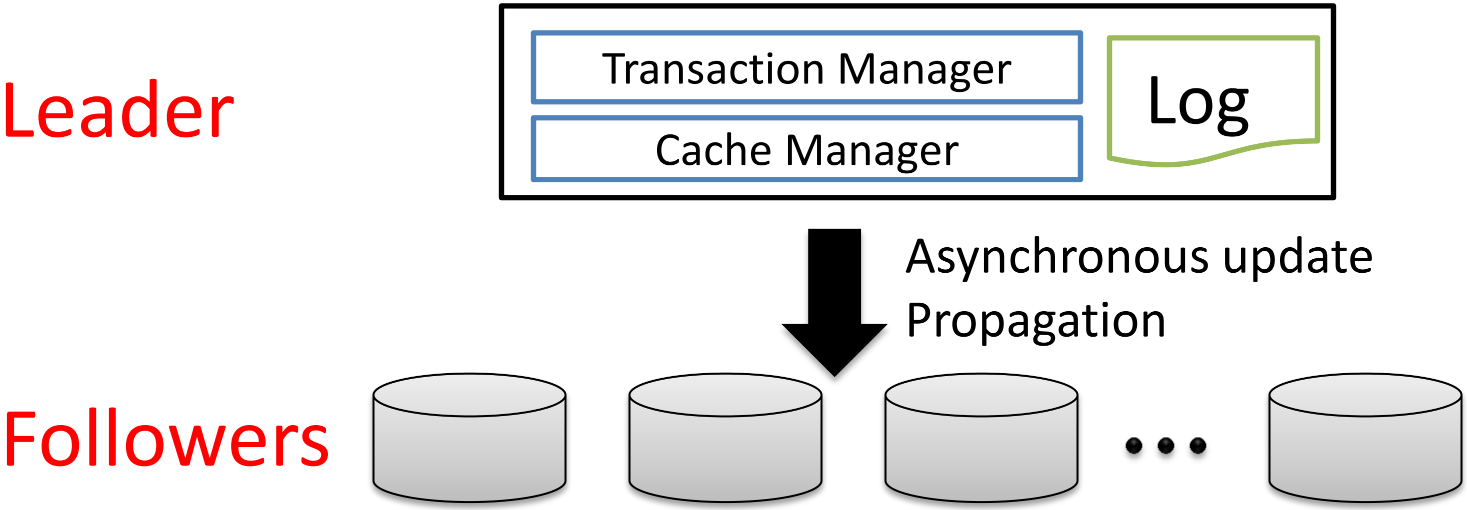
Key Group

Ownership of keys at a single node

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

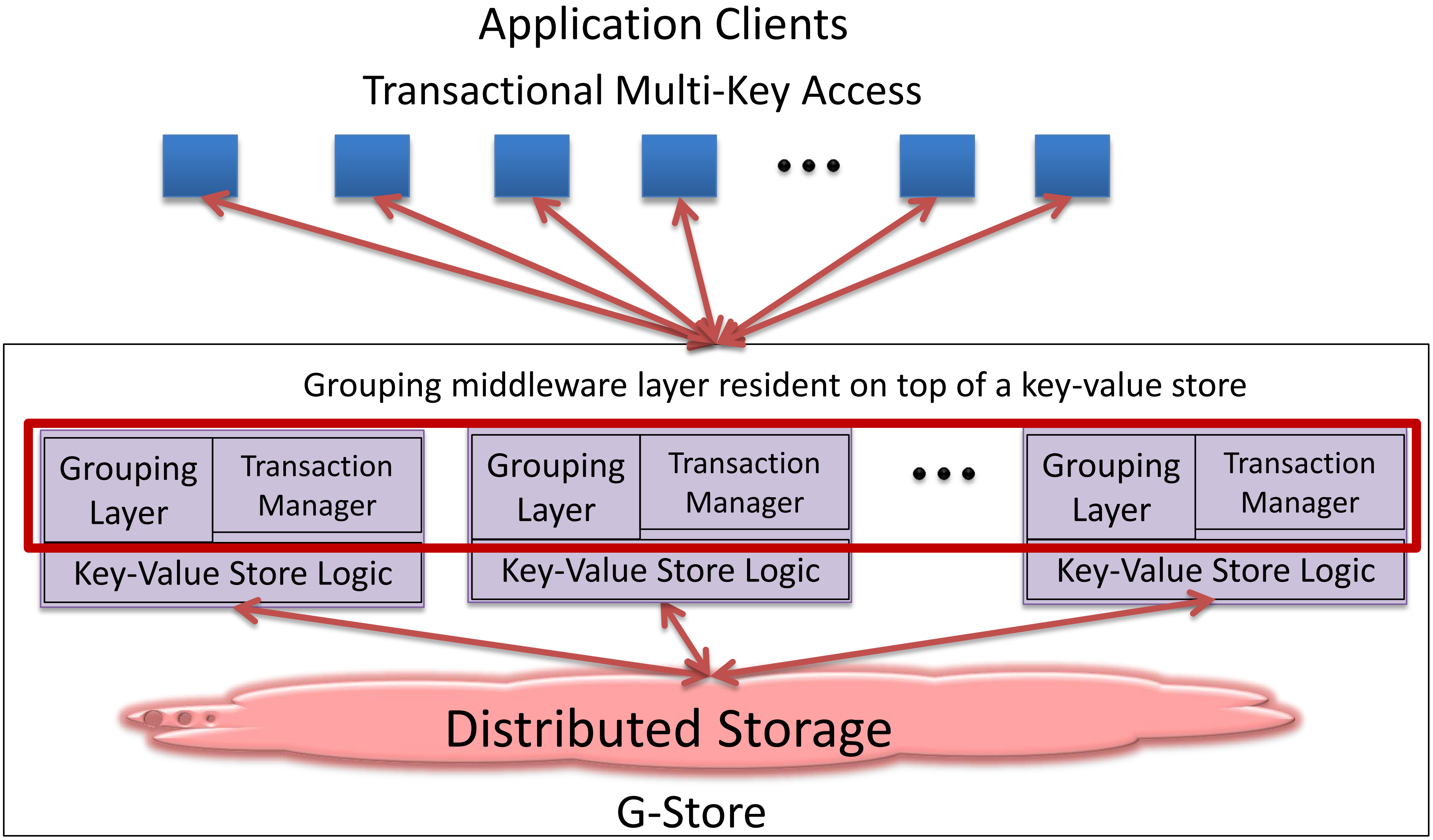
Efficient Transaction Processing

- 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



Application Clients

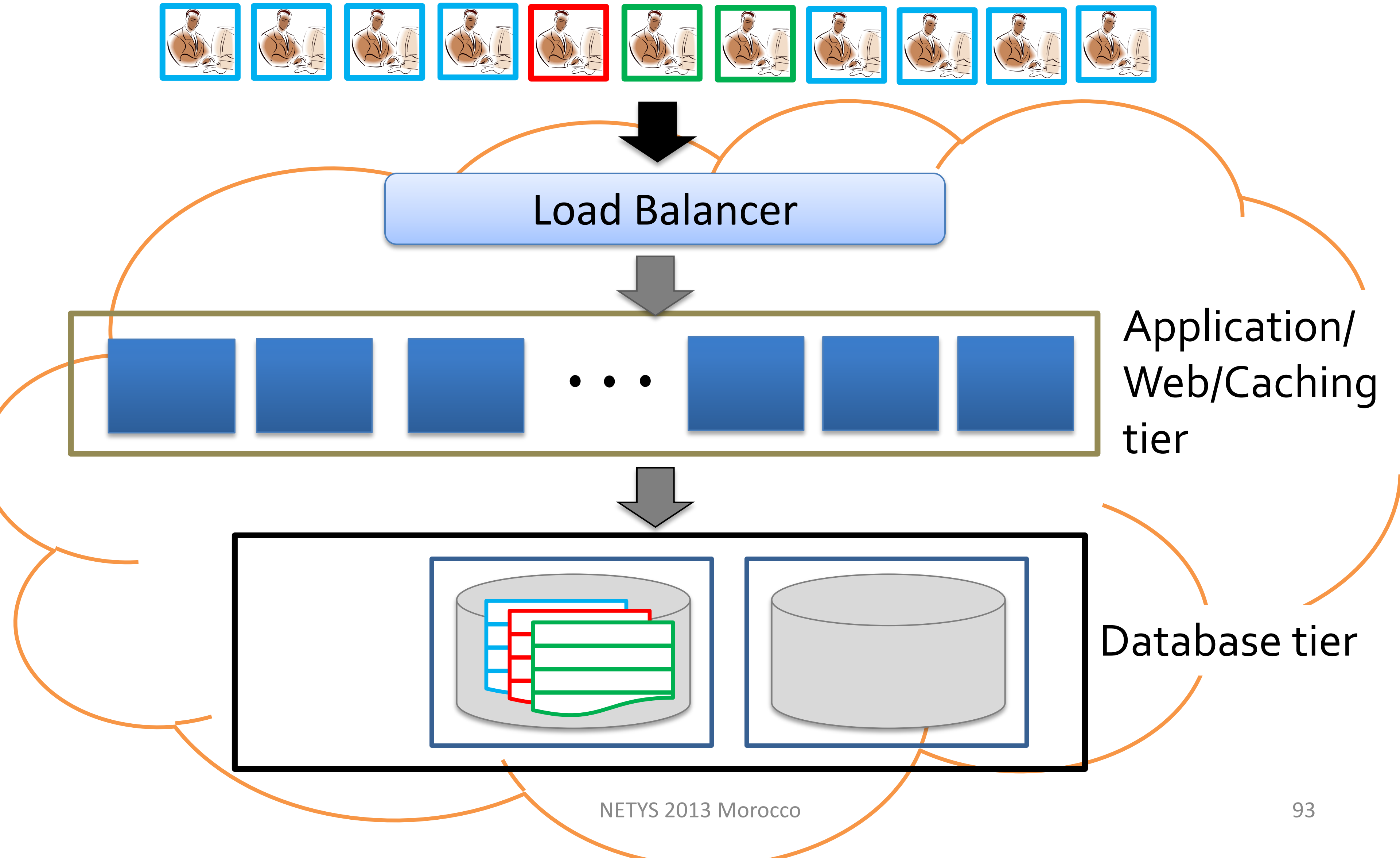
Transactional Multi-Key Access

Grouping middleware layer resident on top of a key-value store

Distributed Storage

G-Store

Challenge: Elasticity in Database tier



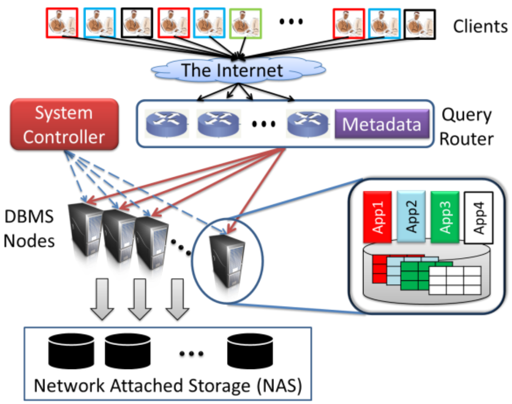
Load Balancer

Application/
Web/Caching
tier

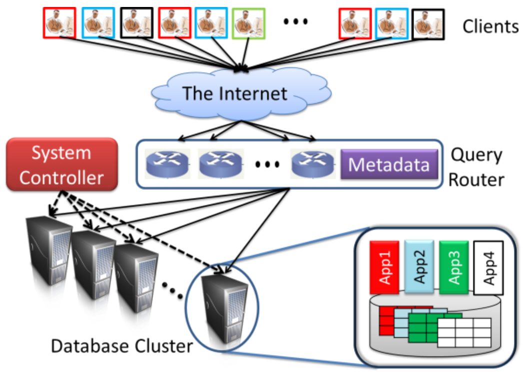
Database tier

Two common DBMS architectures

- **Decoupled storage architectures**
  - ElasTraS, G-Store, Deuteronomy, MegaStore
  - Persistent data is not migrated
  - **Albatross [VLDB 2011]**



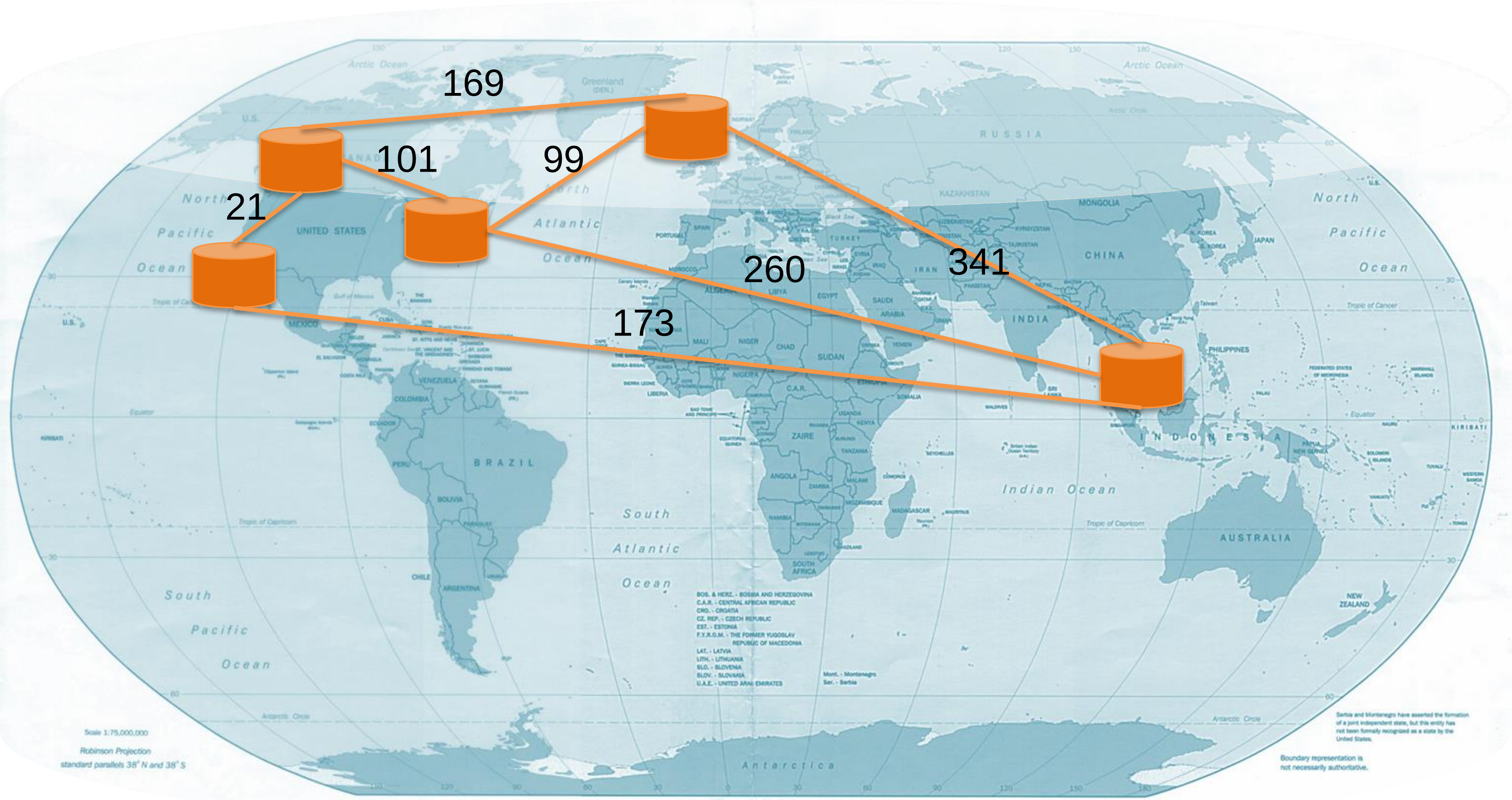
- **Shared nothing architectures**
  - SQL Azure, Relational Cloud, MySQL Cluster
  - Migrate persistent data
  - **Zephyr [SIGMOD 2011]**



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)
  - Distributed synchronization
  - Atomic commitment

Round Trip Times (RTT)



Fault-tolerance in the Cloud

- Megastore Google (CIDR 2011)
- Paxos-CP UCSB (VLDB 2012)
- Message Futures UCSB (CIDR 2013)
- MDCC Berkeley (EuroSys 2013)
- Spanner Google (OSDI 2012)
- Replicated Commits UCSB (On-going)

Next Steps

- 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.

Distributed Systems References

- Leslie Lamport: Time, Clocks, and the Ordering of Events in a Distributed System. Commun. ACM 21(7): 558-565 (1978)
- Mani Chandy, Leslie Lamport: Distributed Snapshots: Determining Global States of Distributed Systems ACM Trans. Comput. Syst. 3(1): 63-75 (1985)
- Gene T. J. Wuu, Arthur J. Bernstein: Efficient Solutions to the Replicated Log and Dictionart Problems. PODC 1984: 233-242
- Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in P2P systems. In Communications of the ACM, February 2003
- Reliable Distributed Computing with the Isis Toolkit. K. Birman and R. van Renesse, eds. IEEE Computer Society Press, 1994.

Distributed Systems References

- Leslie Lamport, Robert E. Shostak, Marshall C. Pease: The Byzantine Generals Problem. ACM Trans. Program. Lang. Syst. 4(3): 382-401 (1982)
- Leslie Lamport: The Part-Time Parliament. ACM Trans. Comput. Syst. 16(2): 133-169 (1998)
- Michael J. Fischer, Nancy A. Lynch, Mike Paterson: Impossibility of Distributed Consensus with One Faulty Process. PODS 1983: 1-7
- Eric A. Brewer. Towards robust distributed systems. (Invited Talk)Principles of Distributed Computing, July 2000.

Database References

- Concurrency Control and Recovery in Database Systems Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman . 1987 (http://research.microsoft.com/en-us/people/philbe/ccontrol.aspx)
- Gerhard Weikum, Gottfried Vossen: Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery Morgan Kaufmann 2002
- Transaction Processing: Concepts and Techniques, Jim Gray and Andreas Reuter. Morgan Kaufmann Publishers 1992

Key-Value Store References

- Chang, Dean, Ghemawat, Hsieh, Wallach, Burrows, Chandra, Fikes, Gruber: **Bigtable: A Distributed Storage System for Structured Data**. OSDI 2006
- **The Google File System**: Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. Symp on Operating Systems Princ 2003.
- **GFS: Evolution on Fast-Forward:** Kirk McKusick, Sean Quinlan Communications of the ACM 2010.
- Cooper, Ramakrishnan, Srivastava, Silberstein, Bohannon, Jacobsen, Puz, Weaver, Yerneni: **PNUTS: Yahoo!'s hosted data serving platform**. VLDB 2008.
- DeCandia,Hastorun,Jampani, Kakulapati, Lakshman, Pilchin, Sivasubramanian, Vosshall, Vogels: **Dynamo: amazon's highly available key-value store**. SOSP 2007
- Cooper, Silberstein, Tam, Ramakrishnan, Sears: Benchmarking cloud serving systems with YCSB. SoCC 2010

First Gen Cloud db References

- Das, Agrawal, El Abbadi, ***"G-Store: A Scalable Data Store for Transactional Multi key Access in the Cloud"*** Symposium on Cloud Computing (SOCC) 2010
- Agrawal, El Abbadi, Antony, Das**, *"Data Management Challenges in Cloud Computing Infrastructures"*****, International Workshop on Databases in Networked Information Systems (DNIS 2010), March 2010
- Das**,** Agrawal, El Abbadi, ***"ElasTraS: An Elastic Transactional Data Store in the Cloud"***, HotCloud '09
- Curino, Jones,Popa,Malviya,Wu,Madden,Balakrishnan, Zeldovich: **Relational Cloud: a Database Service for the cloud**. CIDR 2011
- Bernstein, Cseri,Dani, Ellis, Kalhan, Kakivaya,Lomet,Manne,Novik,Talius: **Adapting microsoft SQL server for cloud computing**. ICDE 2011.
- Levandoski, Lomet, Mokbel,Zhao: **Deuteronomy: Transaction Support for Cloud Data.** CIDR 2011.
- Brantner, Iorescu, Graf, Kossmann, Kraska: **Building a database on S3**. SIGMOD 2008.
- Kraska, Hentschel, Alonso, Kossmann: **Consistency Rationing in the Cloud: Pay only when it matters.** PVLDB 2009
- Kossmann, Kraska, Loesing: An evaluation of alternative architectures for transaction processing in the cloud. SIGMOD 2010

2nd Gen Cloud DB References

- Baker, Bond,Corbett, Furman, Khorlin, Larson, Leon, Li, Lloyd, Yushprakh: **Megastore: Providing Scalable, Highly Available Storage for Interactive Services**. CIDR 2011.
- Patterson, Elmore, Nawab, Agrawal, El Abbadi: **Serializability, not Serial: Concurrency Control and Availability in Multi-Datacenter Datastores**. VLDB 2012
- Nawab, Agrawal, El Abbadi, "Message Futures: Fast Commitment of **Transactions in Multi-datacenter Environments**", CIDR, 2013
- *Kraska, Pang, Franklin, Madden, Fekete:* **MDCC: Multi-Data Center Consistency.** EuroSys 2013
- Corbett et al.: **Spanner: Google's Globally-Distributed Database.** OSDI 2012.