-Megastore-
Providing Scalable, Highly Available, Storage for Interactive Services

Lin Jinting, Chen Dian, Zhang Weiming, Liang Jiahui

INDEX

01 INTRODUCTION

02 AVAILABILITY AND SCALE

03 MEGASTORE

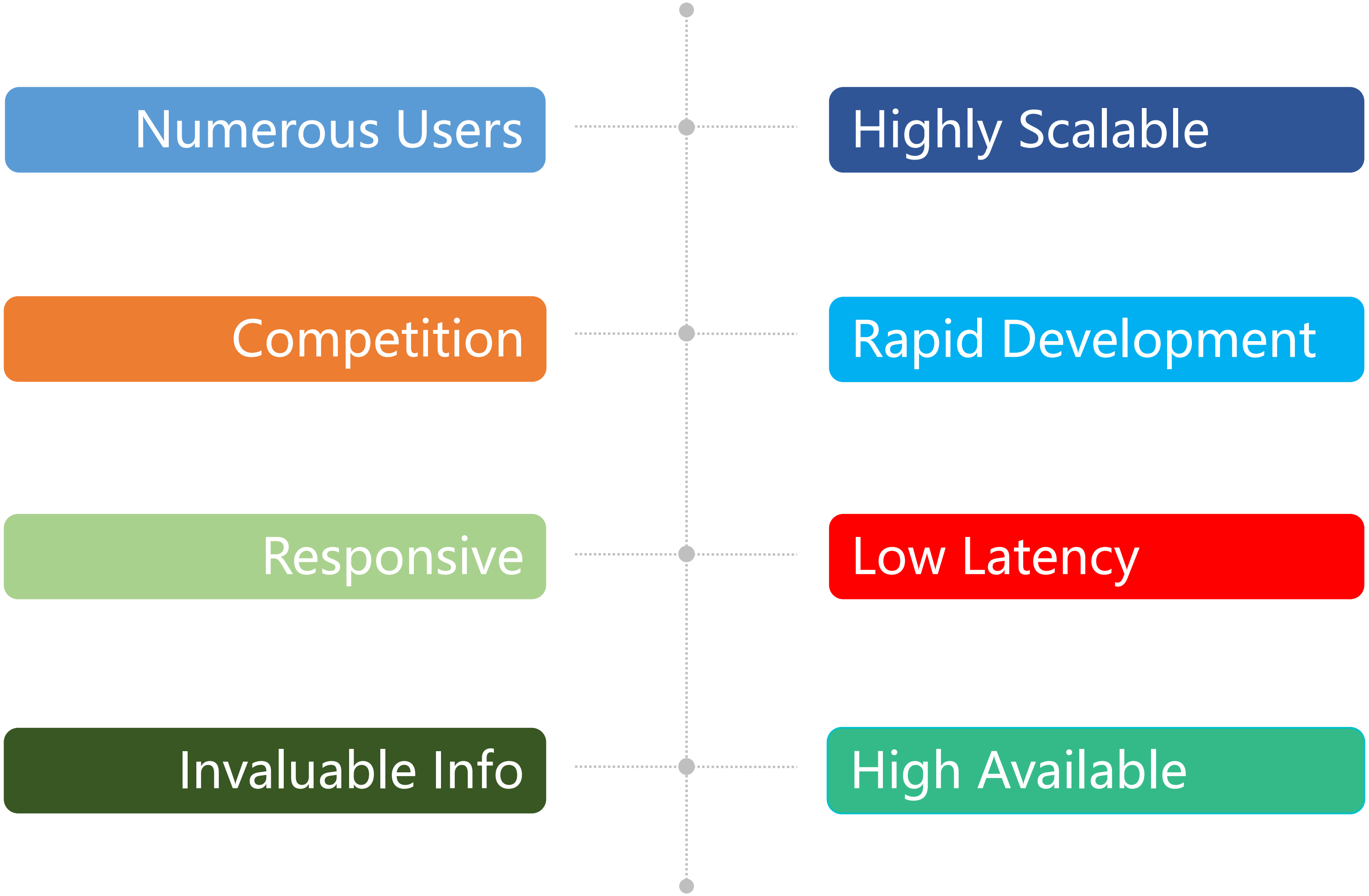
04 CORE: REPLICATION

05 CONCLUSION

1

INTRODUCTION

Today`s Interactive Online Services



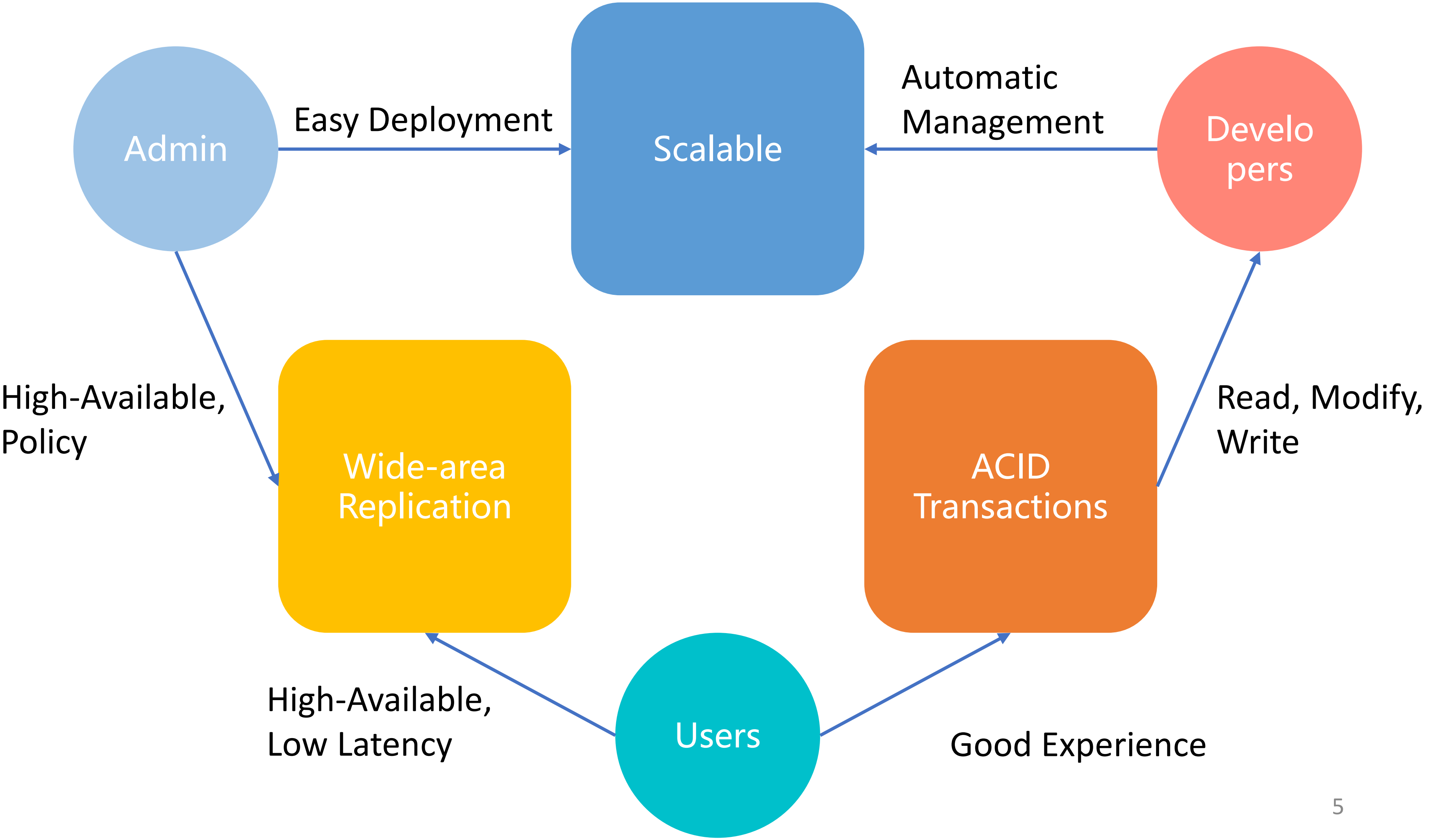
Numerous Users — Highly Scalable

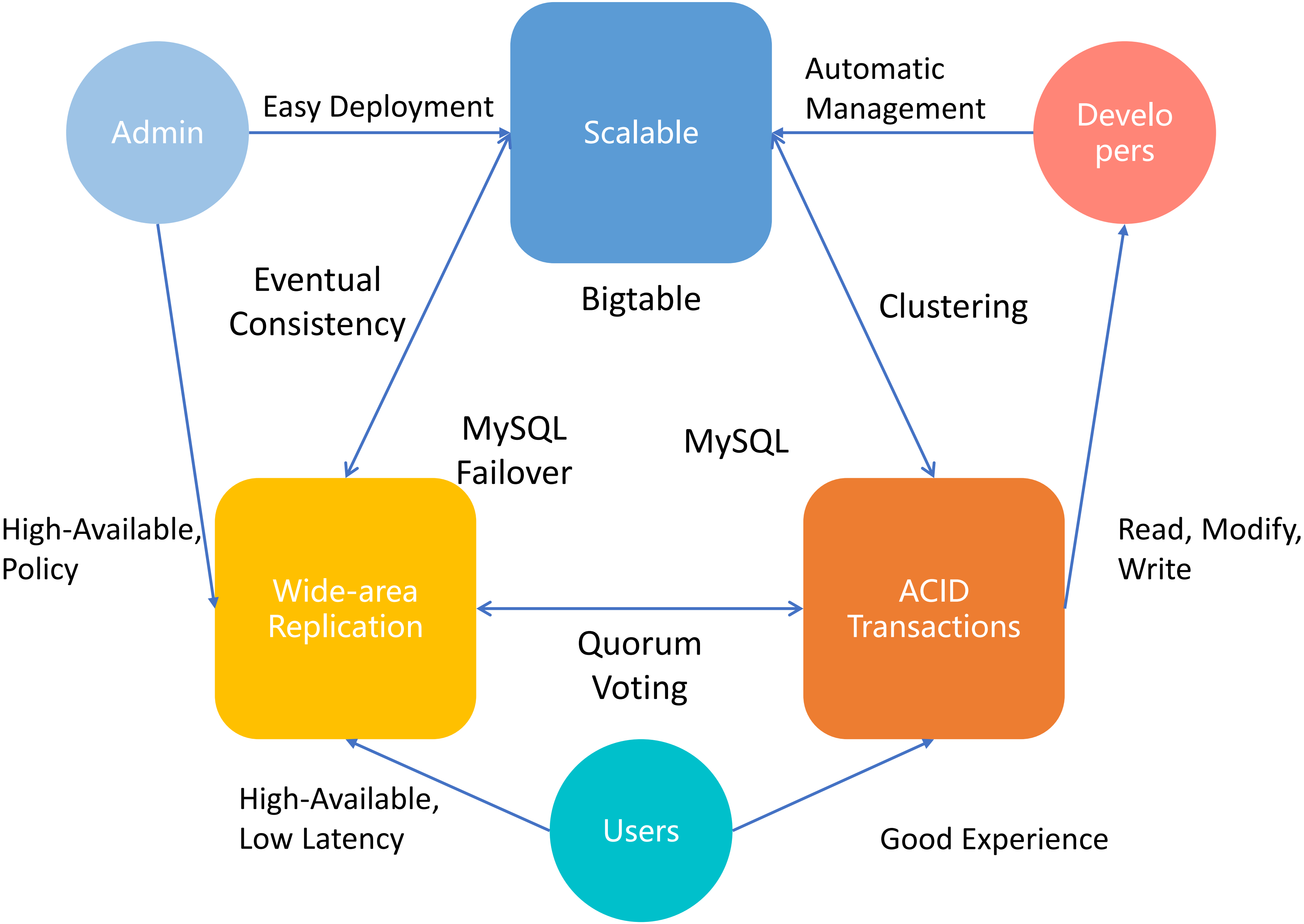
Competition — Rapid Development

Responsive — Low Latency

Invaluable Info — High Available

Everybody`s Requirement





Admin

Easy Deployment

Scalable

Automatic Management

Developers

Eventual Consistency

Bigtable

Clustering

MySQL Failover

MySQL

High-Available, Policy

Wide-area Replication

Quorum Voting

ACID Transactions

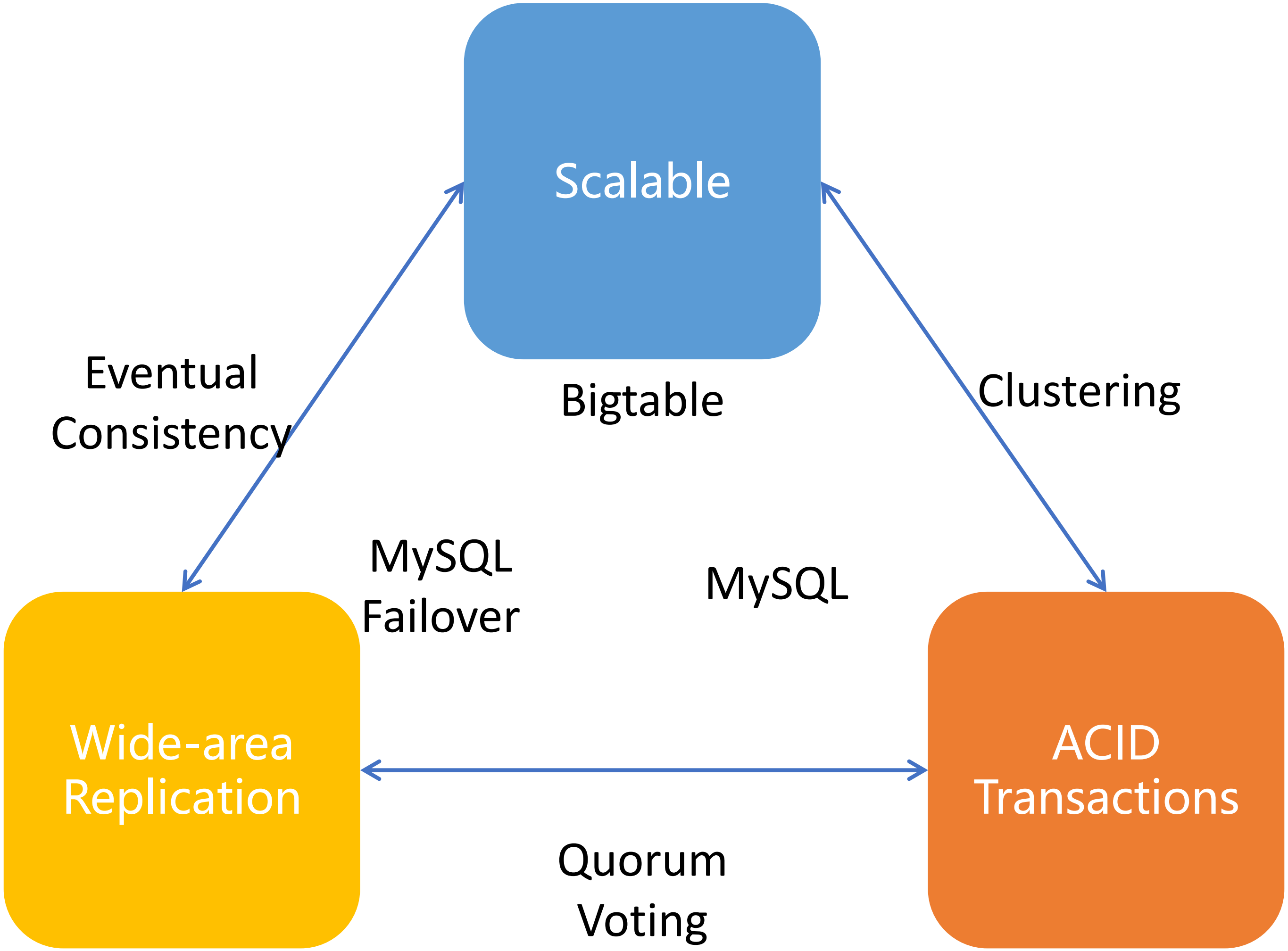
Read, Modify, Write

Users

High-Available, Low Latency

Good Experience

Our Solution



2

AVAILABILITY AND SCALE

Overview

1

For Availability

A Synchronous, Fault-tolerant log replicator. (i.e., by **Chubby**.)

2

For Scale

NoSQL data storage with its own replicated log. (i.e. by **Bigtable**.)

Common Replication Strategies And Paxos

1. Asynchronous Master / Slave Mode

   Master maintains the writes ahead log. If there are appends to the replication log, master will be acknowledged in parallel with slave through Message Queue(MQ).

   Weakness: But if the target slave is down, data loss will occur. Therefore needs a consensus protocol.

2. Synchronous Master / Slave Mode

   Master inform its slaves after the changes are applied.

   Weakness: Needs a external system to keep the time.

3. Optimistic Replication

   Mutations are propagate through the gourp asynchronous.
   Because *the order of propagation is unpredictable*, it is impossible to implement transaction with such algorithm.

4. Paxos

   No distinguished master and slave. Use vote and catch up to keep data consensus.

Entity Groups

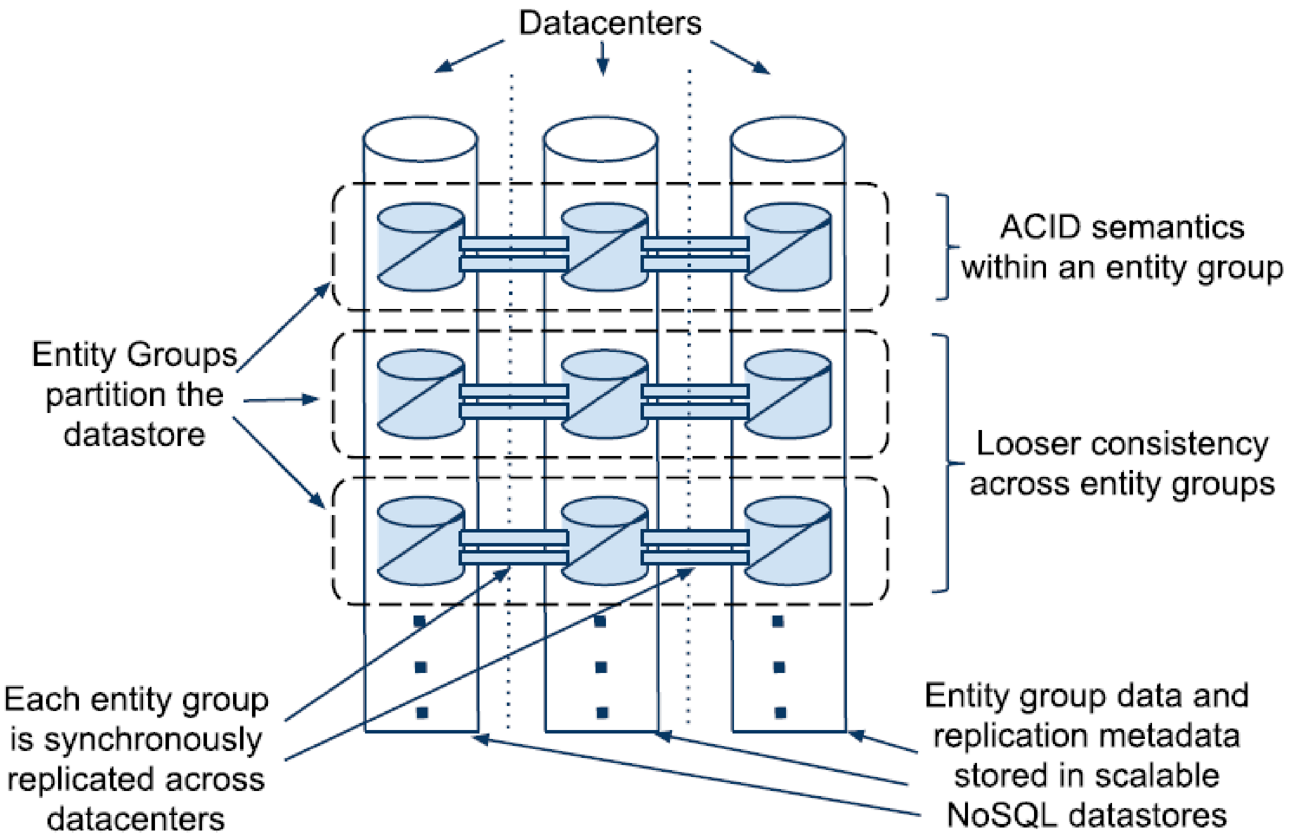

Datacenters
ACID semantics within an entity group
Entity Groups partition the datastore
Looser consistency across entity groups
Each entity group is synchronously replicated across datacenters
Entity group data and replication metadata stored in scalable NoSQL datastores

Figure 1: Scalable Replication

Operations Across Entity Groups

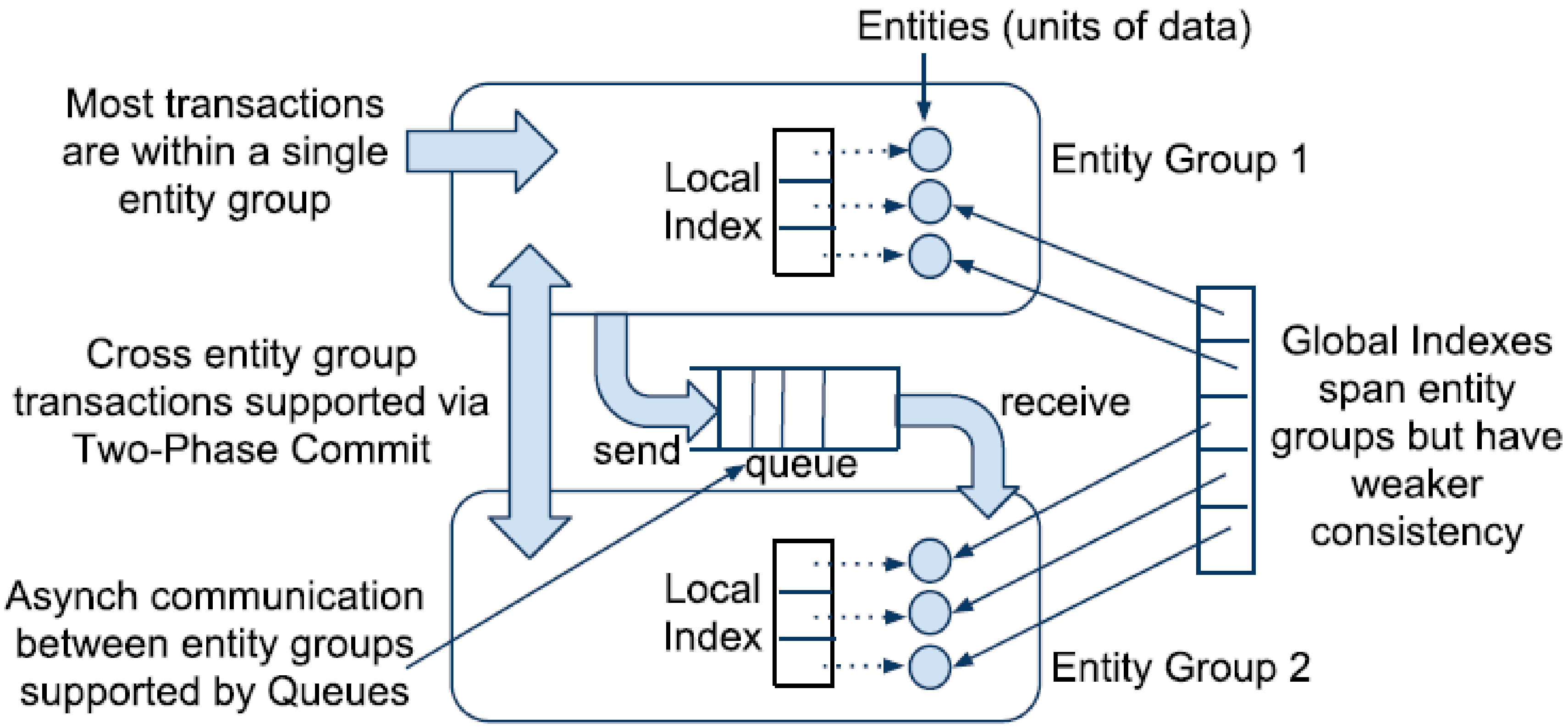
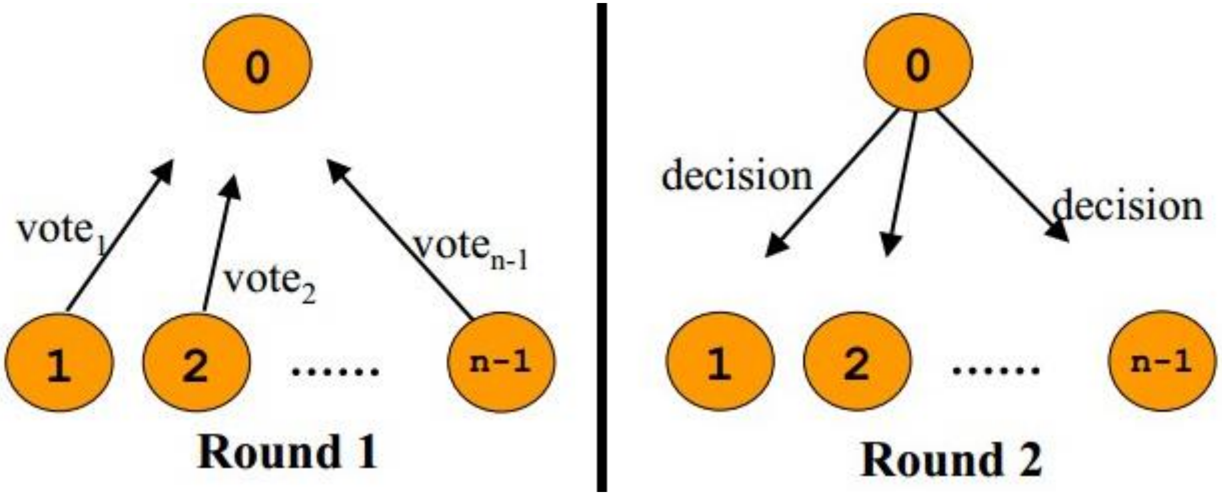

Entities (units of data)
Most transactions are within a single entity group
Entity Group 1
Local Index
Cross entity group transactions supported via Two-Phase Commit
send
queue
receive
Global Indexes span entity groups but have weaker consistency
Asynch communication between entity groups supported by Queues
Local Index
Entity Group 2

Figure 2: Operations Across Entity Groups

Two-Phase Commit(2PC)



1. Commit-request phase, so called voting phase

In this phase, coordinator(coord.) will inform all partcipants that whether they can cmmit or not, and all partcipants respond their status and make preparation if agree.

2. Commit phase

If all vote yes, than this transaction will be committed in all partcipants. If some of them vote no, this transaction will be abort and all partcipant will rollback previous actions.

How to define the boundary of a entity group?

1. Email

Each email account is able to form a entity group. And A sends an email to B will be regards as an across-group transaction.

2. Blogs

- The profile for each user will be formed a entity group naturally.
- Posts and metadata will be formed another entity group.
- A third entity group will be used to describes the blog titles.
- Transactions are rely on asynchronous message queue.

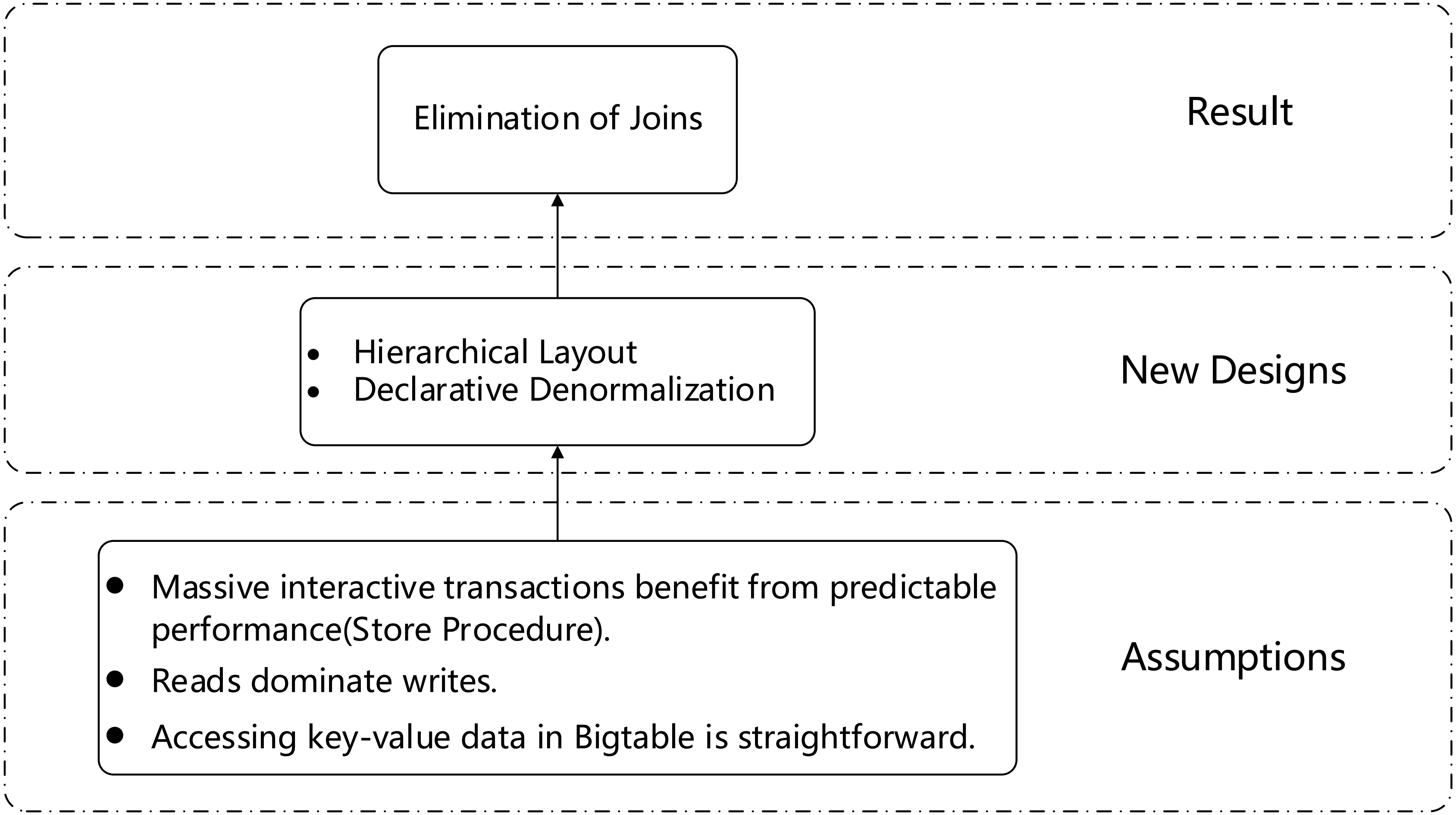
3. Maps

Not such modle is suitable for describing gergraphic data. We usually divide the map into several non-overlapping patches, in order to form a entity group for a single patch.

3

MEGASTORE

Assumptions And Philosophy For API Design



Elimination of Joins

Result

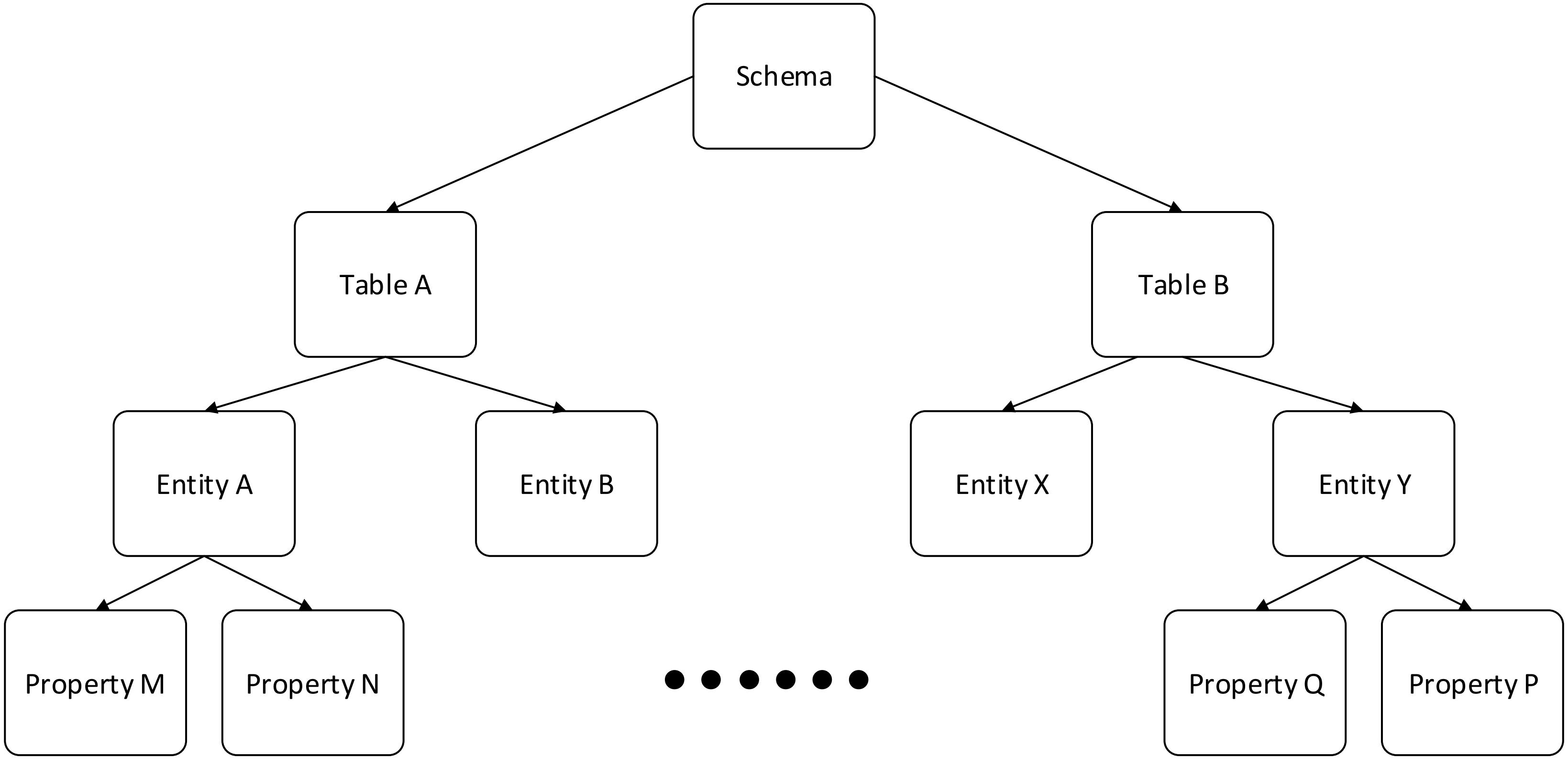
- Hierarchical Layout
- Declarative Denormalization

New Designs

- Massive interactive transactions benefit from predictable performance(Store Procedure).
- Reads dominate writes.
- Accessing key-value data in Bigtable is straightforward.

Assumptions

What Is Hierarchical Layout?



Sample Data Schema

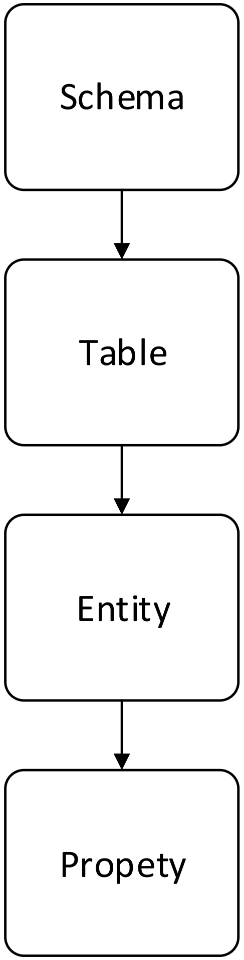
```
CREATE SCHEMA PhotoApp;
```

```
CREATE TABLE User {
   required int64 user_id;
   required string name;
}  PRIMARY KEY(user_id), ENTITY GROUP ROOT;
```

```
CREATE TABLE Photo {
   required int64 user_id;
   required int32 photo_id;
   required int64 time;
   required string full_url;
   optional string thumbnail_url;
   repeated string tag;
}  PRIMARY KEY(user_id, photo_id),
   IN TABLE User,
   ENTITY GROUP KEY(user_id) REFERENCES User;
```

```
CREATE LOCAL INDEX PhotosByTime ON Photo(user_id, time);
```

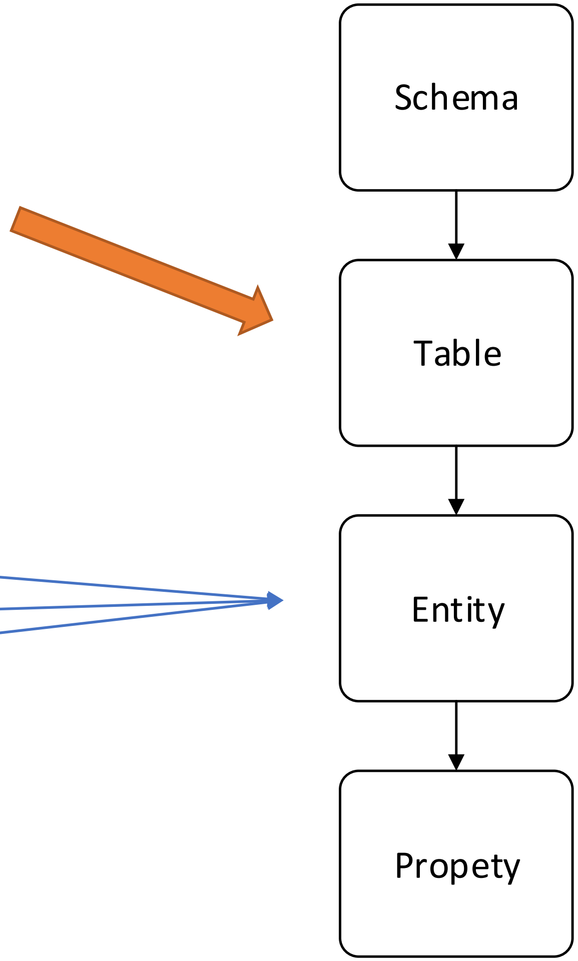
```
CREATE GLOBAL INDEX PhotosByTag ON Photo(tag) STORING (thumbnail_url);
```



Data Stored In Bigtable – User Table

```
CREATE TABLE User {
    required int64 user_id;
    required string name;
}    PRIMARY KEY(user_id), ENTITY
GROUP ROOT;
```

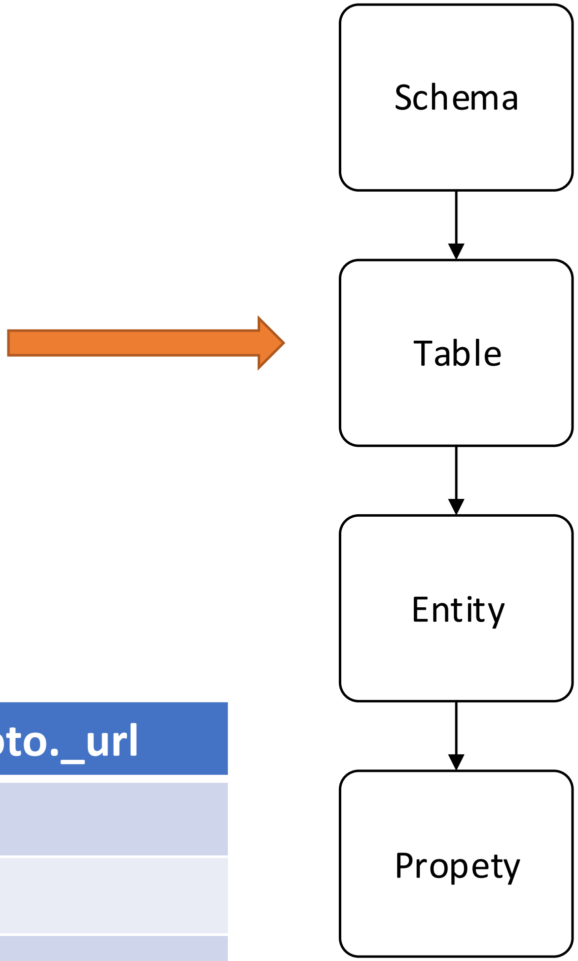
Row Key	User.name
101	John
102	Mary
103	Jane



Data Stored In Bigtable – Photo Table

```
CREATE TABLE Photo {
    required int64 user_id;
    required int32 photo_id;
    required int64 time;
    required string full_url;
    optional string thumbnail_url;
    repeated string tag;
}   PRIMARY KEY(user_id, photo_id),
    IN TABLE User,
    ENTITY GROUP KEY(user_id)
REFERENCES User;
```

Row Key	Photo.name	Photo.tag	Photo._url
101, 500	12:30:01	Dinner, Paris	…
101, 502	12:15:22	Dinner, Paris	…
103, 19	08:32:11	Office	…



Data Stored In Bigtable – Foreign key

Entity Group *root*

```
CREATE TABLE User {
    required int64 user_id;
    required string name;
}   PRIMARY KEY(user_id), ENTITY GROUP ROOT;
```

1.Stored in User table

2.Child table refers to User

```
CREATE TABLE Photo {
    required int64 user_id;
    required int32 photo_id;
    required int64 time;
    required string full_url;
    optional string thumbnail_url;
    repeated string tag;
}   PRIMARY KEY(user_id, photo_id),
    IN TABLE User,
    ENTITY GROUP KEY(user_id) REFERENCES
User;
```

Data Stored In Bigtable – Actual Bigtable

Row key	User.name	Photo.time	Photo.tag	Photo._url	…
101	John				
101, 500		12:31:01	Dinner, Paris	…	…
101, 502		12:15:22	Betty, Paris	…	…
102	Mary				
103	Jane				
103, 19		08:32:11	Office	…	…

Primary Key Table **User** Table **Photo**

Data Stored In Bigtable – Entity Groups

	Row key	User.name	Photo.time	Photo.tag	Photo._url	•••
EG - John	101	John				
	101, 500		12:31:01	Dinner, Paris	···	···
	101, 502		12:15:22	Betty, Paris	···	···
EG - Mary	102	Mary				
EG – Jane	103	Jane				
	103, 19		08:32:11	Office	···	···

Primary Key   Table **User**   Table **Photo**

Entity Groups: Users and his Photos

1. **Root Entity: User**
2. **Child Entity: Photo**

Data Stored In Bigtable – Data Partitions

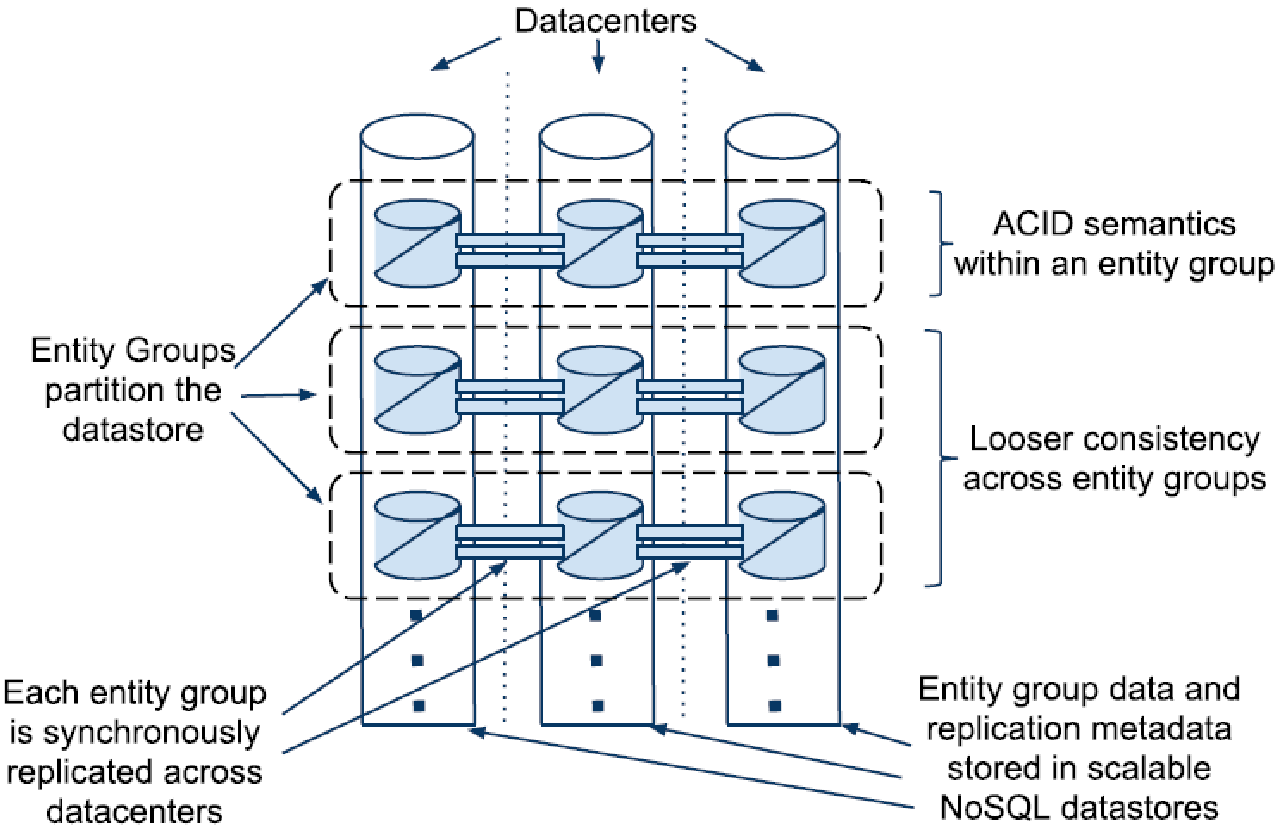
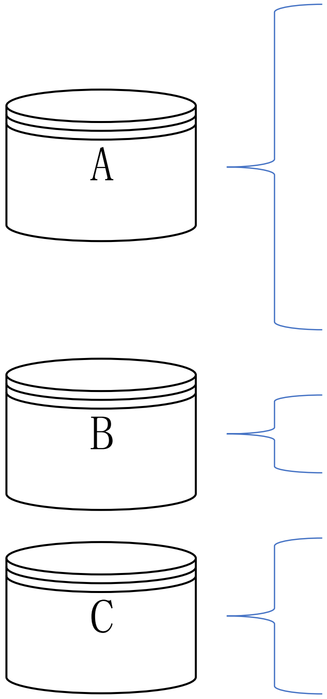

Datacenters
ACID semantics within an entity group
Entity Groups partition the datastore
Looser consistency across entity groups
Each entity group is synchronously replicated across datacenters
Entity group data and replication metadata stored in scalable NoSQL datastores

Figure 1: Scalable Replication

Data Stored In Bigtable – Data Partitions



Row key	User.name	Photo.time	Photo.tag	Photo._url	•••
101	John				
101, 500		12:31:01	Dinner, Paris	…	…
101, 502		12:15:22	Betty, Paris	…	…
102	Mary				
103	Jane				
103, 19		08:32:11	Office	…	…

Data Stored In Bigtable – Index Features

1. Storing Clause

Storing additional info for retrieval. It will make search for specific entity spend less communication rounds.

```
CREATE GLOBAL INDEX PhotosByTag ON Photo(tag) STORING (thumbnail_url);
```

If you want to get the thumbnail_url without this index, it will take at least 2 rounds communications. But with this, it takes 1 round minimum.

2. Repeated Indexed

```
CREATE GLOBAL INDEX PhotosByTag ON Photo(tag) STORING (thumbnail_url);
```

Each repeated tag has its own index entry.

3. Inline Indexes

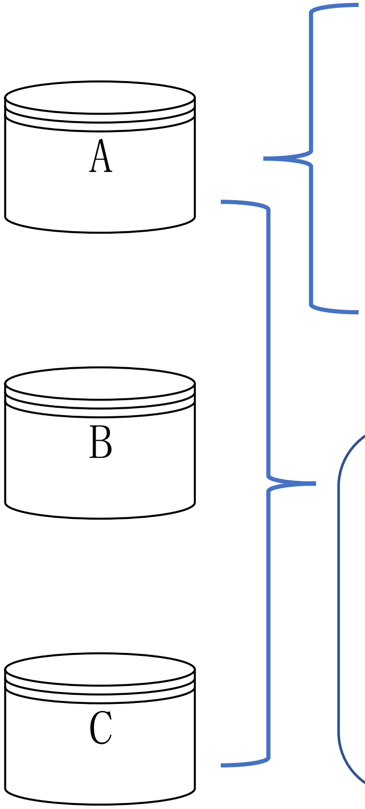
```
CREATE LOCAL INDEX PhotosByTime ON Photo(user_id, time);
```

Extracting slices of info from child entities and storing it in the parent for fast access.

Data Stored In Bigtable – Local And Global Index

```
CREATE LOCAL INDEX PhotosByTime ON Photo(user_id, time);
```

```
CREATE GLOBAL INDEX PhotosByTag ON Photo(tag) STORING (thumbnail_url);
```



User._id	time	Row key
John	12:31:01	101, 500
John	12:15:22	101, 502
James	12:12:00	108, 501

Local Index:
Used to find entities within an EG.

Photo.tag	Thumbnail_url	Row key
Dinner	…	101, 500
Paris	…	101, 500
Betty	…	108, 501

Global Index:
Used to find entities across an EG.

Within EG - ACID Semantics And MVCC

1. Write Ahead Log

Write it *before* apply the changes. It can be used for fail recovery or transaction rollback.

2. MultiVersion Concurrency Control (MVCC)

Different values can be stored in a single Bigtable cell, with their *timestamps* attached.

Reader uses *timestamps* to identify the latest value for target property in a fully updated transaction.

At the same time, reads and writes are *isolated*, as there are multiply versions. If a writer is appending the latest value to bigtable, reads will fetch the one version older value. *But still the latest value until the writer finishes.*

Photo.tag
[(Dinner, Paris), 12:30:01], [(Father, Mother), 12:31:01]
[(Betty, Paris), 12:15:22], [(Betty), 12:16:22]

Read – Current, Snapshot And Inconsistent Read

1. Current Read

   Always done within a single EG. Read info before *confirming all previous transactions are applied.*

2. Snapshot Read

   Picks up the *latest known* fully applied version, though there may be some transactions waiting for applied, for example, transactions delayed in Asynchronous Message Queue for network problems.

3. Inconsistent Read

   Reads the value in Bigtable directly regards the log status.

See the differences between them from table below:

Type	Data Consensus	Latency
Current Read	High	High
Snapshot Read	Medium	Medium
Inconsistent Read	Low	Low

Write – Complete Transaction Lifecycle

1. Current Read

   Uses a current read to determine the next available log position.

2. Application Logic

   Prepare the data to be written together, and designate it a latest log position. Also, batching writes to a front-end server can reduce the possibility of contention.

3. Commit

   Client submit mutations and the server will use Paxos to vote a consensus value across all replicas.

4. Apply

   Write mutations that win the Paxos procedure into the Bigtable, with its own timesamp attached.

5. Clean UP

   Clean all unnecessary values. For example, older version of a updated value.

4

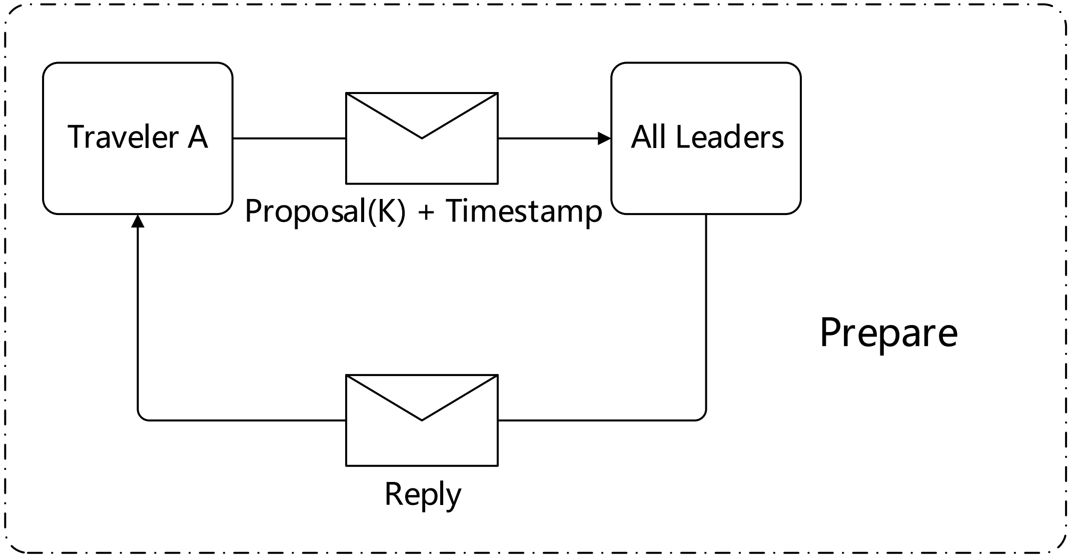
CORE: REPLICATION

Original Paxos With Sample I

A way to reach consensus among a distributed system on a given value by winning more than half votes. There are 2 phases: (1)Prepare and (2)Accept.

But it is ill-suited for high letency network.

Suppose 25 travellers need to reach a consensus about where to go with 5 additional leaders.



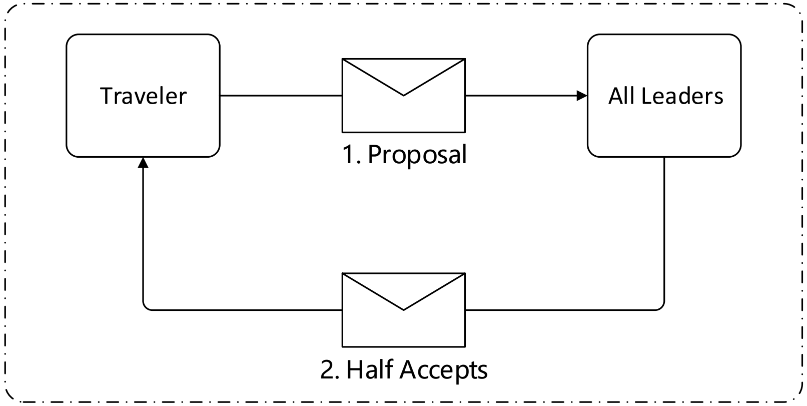
Leaders will only reply the latest message he receives.

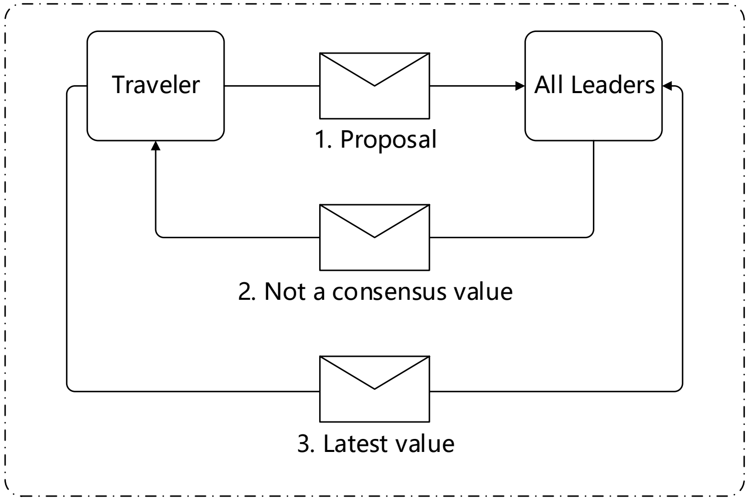
With at least half of the leaders` accept(i.e., 3 leaders), this traveler can enter the Accept phase.

Original Paxos With Sample II

There are 2 possible situations:

1. If none of the leaders had made decision. This traveller will send message to all leaders with his proposal.
   1. If more half leaders reach a consensus, this will be the decision.
   2. If it is the other situation, he has to retry from Prepare phase.
2. At least 1 leader had made decision.
   1. If more half leaders reach a consensus, this will be the decision.
   2. If all leaders had not reached consensus, he will supports the latest decison.





Megastore`s Approach – Fast Read And Write

1. Fast Read
    1. Local reads

       "Write usually succeed on all replicas." Therefore, allowing local reads with lower latercies and better utilization is reasonable.
    2. Coordinator

       "A coordinator is a server tracks a set of entity groups for withc its replica has observed all Paxos writes."

2. Fast Write
    1. Implied prepare message

       Each successful write implied a prepare message for next log position it needs to perform next write. So, 1 round for each subsequent writes is saved.
    2. "Use the closest replica"

       Select the replica with most submitting in this region.

Megastore`s Approach – Replica Types

There are 3 different types of replicas in Megastore:

1. Full Replica
2. Witness Replica
3. Read-only Rreplica

See the differences from table below.

	Full Replica	Witness Replica	Read-only replica
Current Read	Yes	No	No
Log and Data Storage	All	Not-applied log, No data and indexes	Full data snapshot
Vote	Yes	Yes	No
Usage	All	Tie breakers, Voting	Dissemination - CDN

Architecture Example

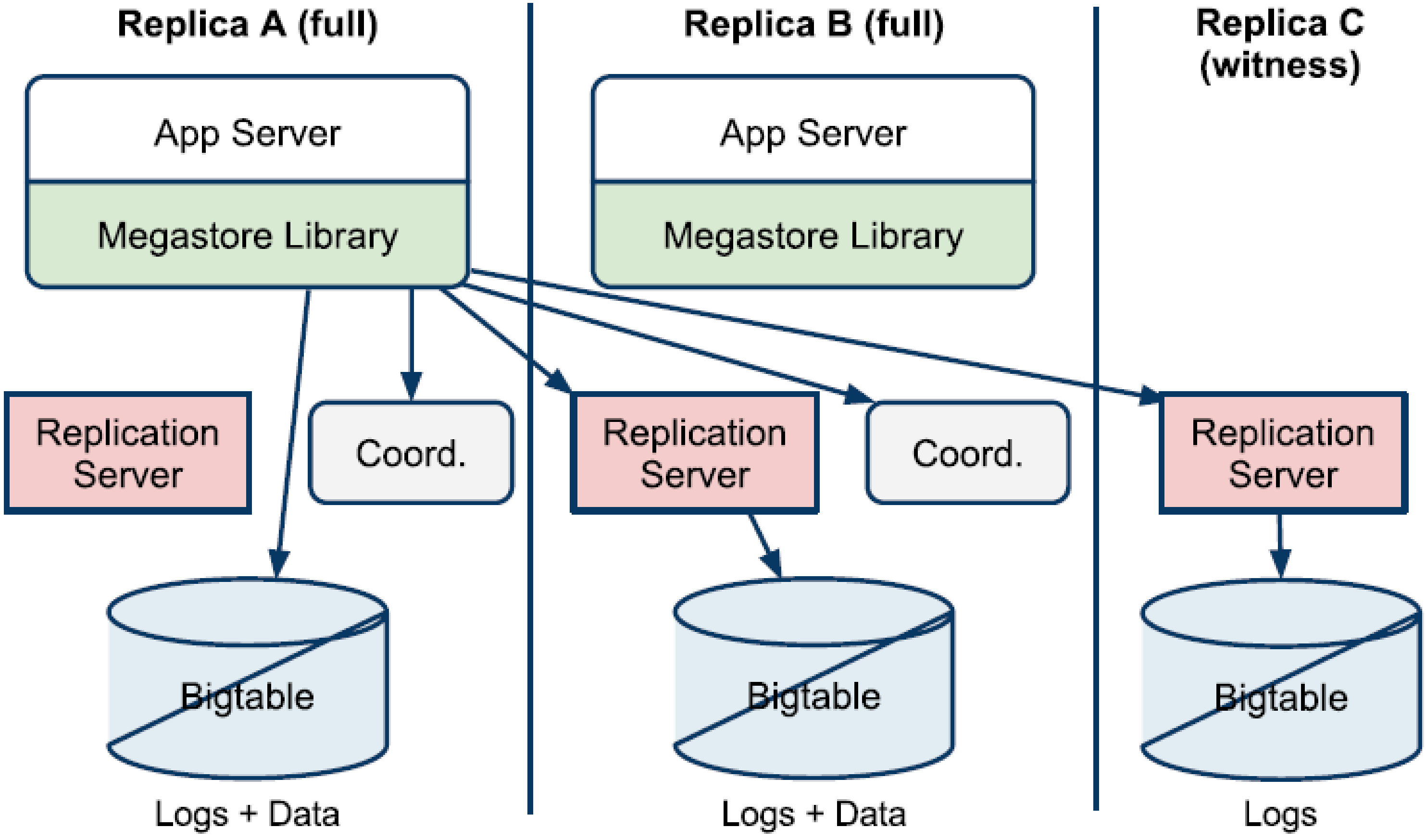
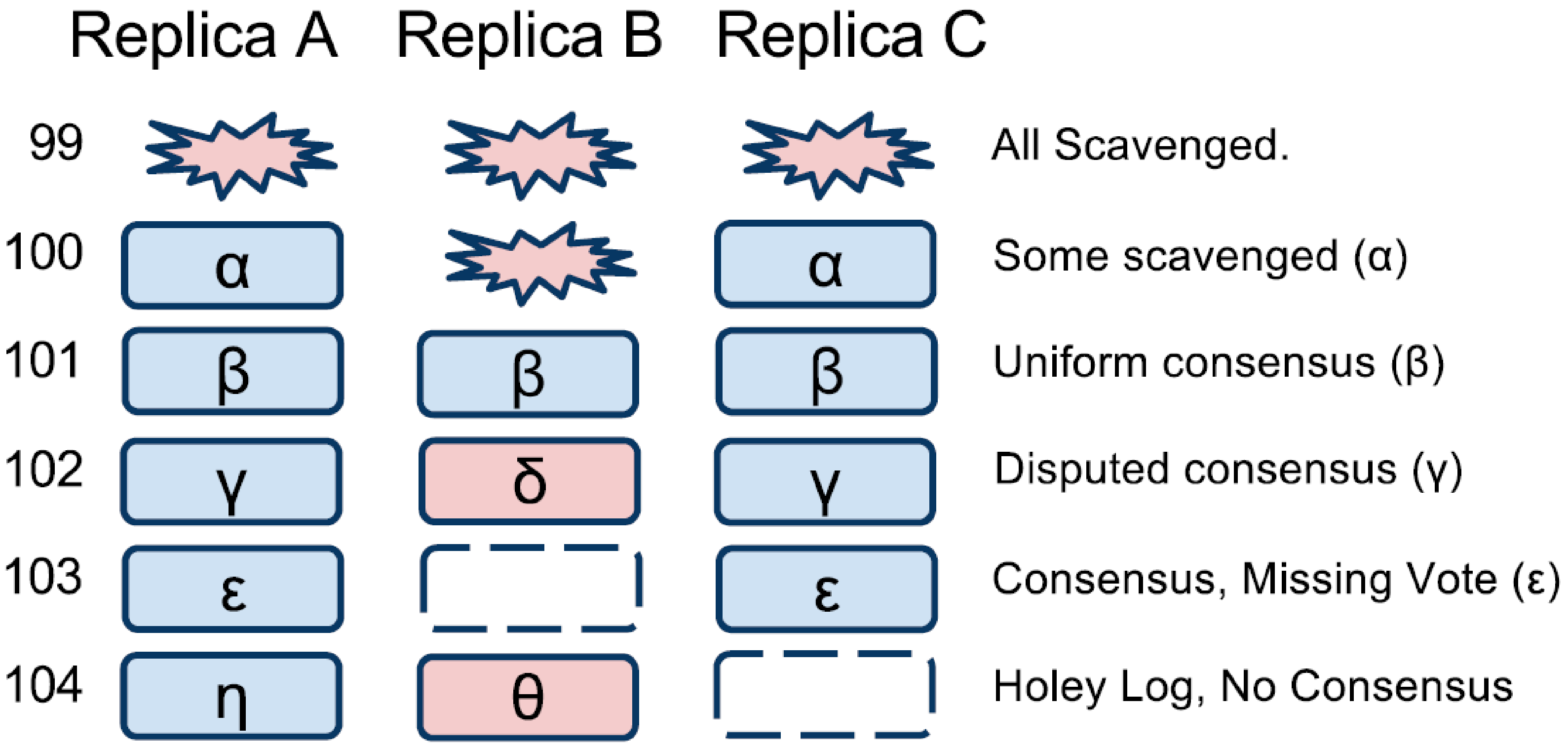


Figure 5: Megastore Architecture Example

Replicated Logs



	Replica A	Replica B	Replica C	
99	✸	✸	✸	All Scavenged.
100	α	✸	α	Some scavenged (α)
101	β	β	β	Uniform consensus (β)
102	γ	δ	γ	Disputed consensus (γ)
103	ε	⬚	ε	Consensus, Missing Vote (ε)
104	η	θ	⬚	Holey Log, No Consensus

Figure 6: Write Ahead Log

Reads

There are 5 steps for read algorithm in Megastore:

1. Query Local

   "Determine if the entity group is up-to-date locally."

2. Find Position

   Get the highest log position, and select the corresponding replica.

   1. Local read. Get the log position and timestamp locally.
   2. Majority read.

3. Catchup

   1. Get unknow value from other replica; run Paxos for any unconsensus.
   2. Apply all consensus value, and push the state up-to-date.

4. Validate

   Send its coordinator a message asserting itself up-to-date.

5. Query Data

Read Example Timeline

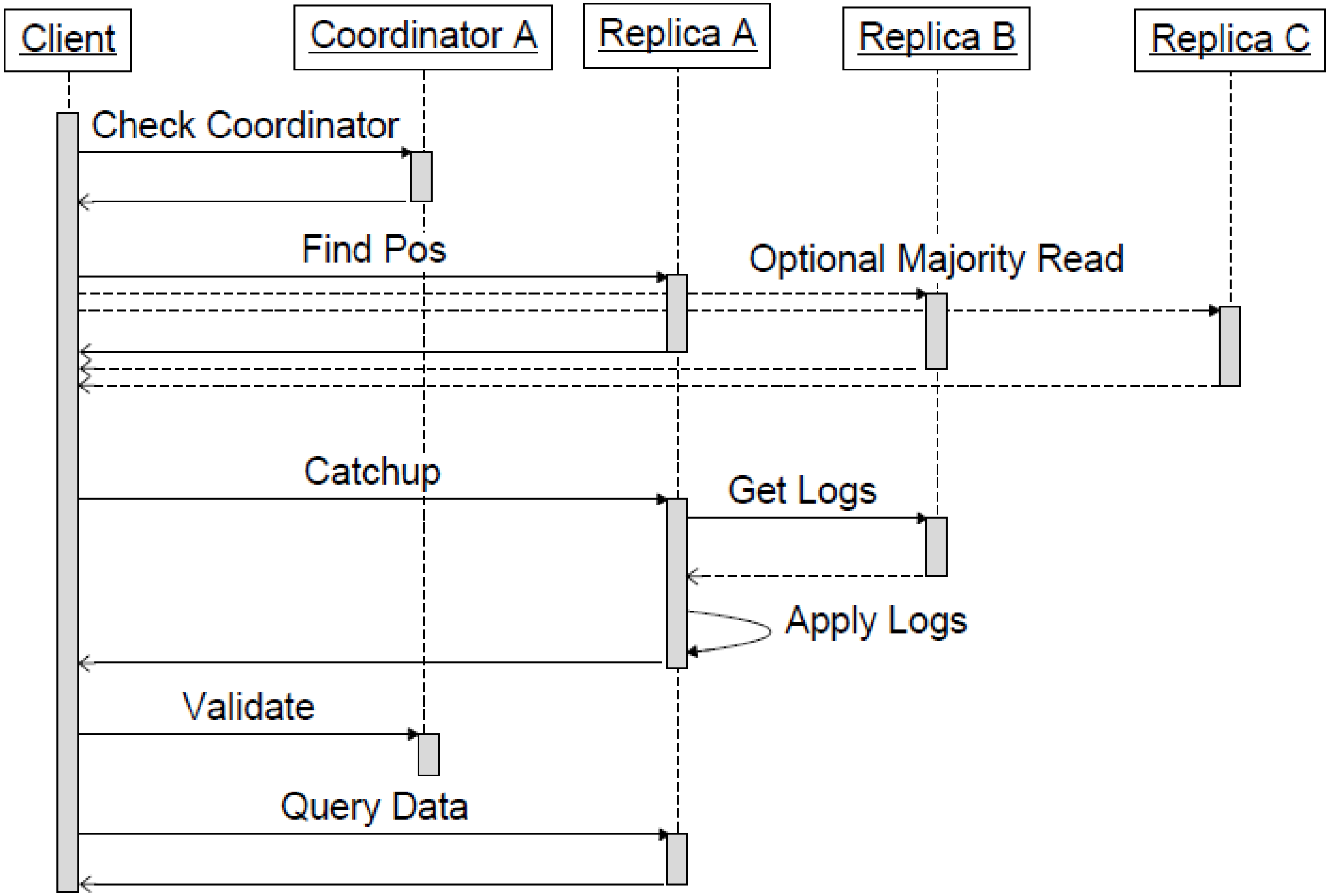


Figure 7: Timeline for reads with local replica A

Writes

There are 5 steps for write algorithm in Megastore:

1. Accept Leader

   "Ask the leader to accept the value as proposal number zero."

2. Prepare

   "Run Paxos Prepare phase at all replicas."

3. Accept

   Ask the rest of replicas to accept the proposal, as the Accept Phase in Paxos.

4. Invalidate

   If a full replica does`n accept this value, invalidate its coordinator.

5. Apply

Write Example Timeline

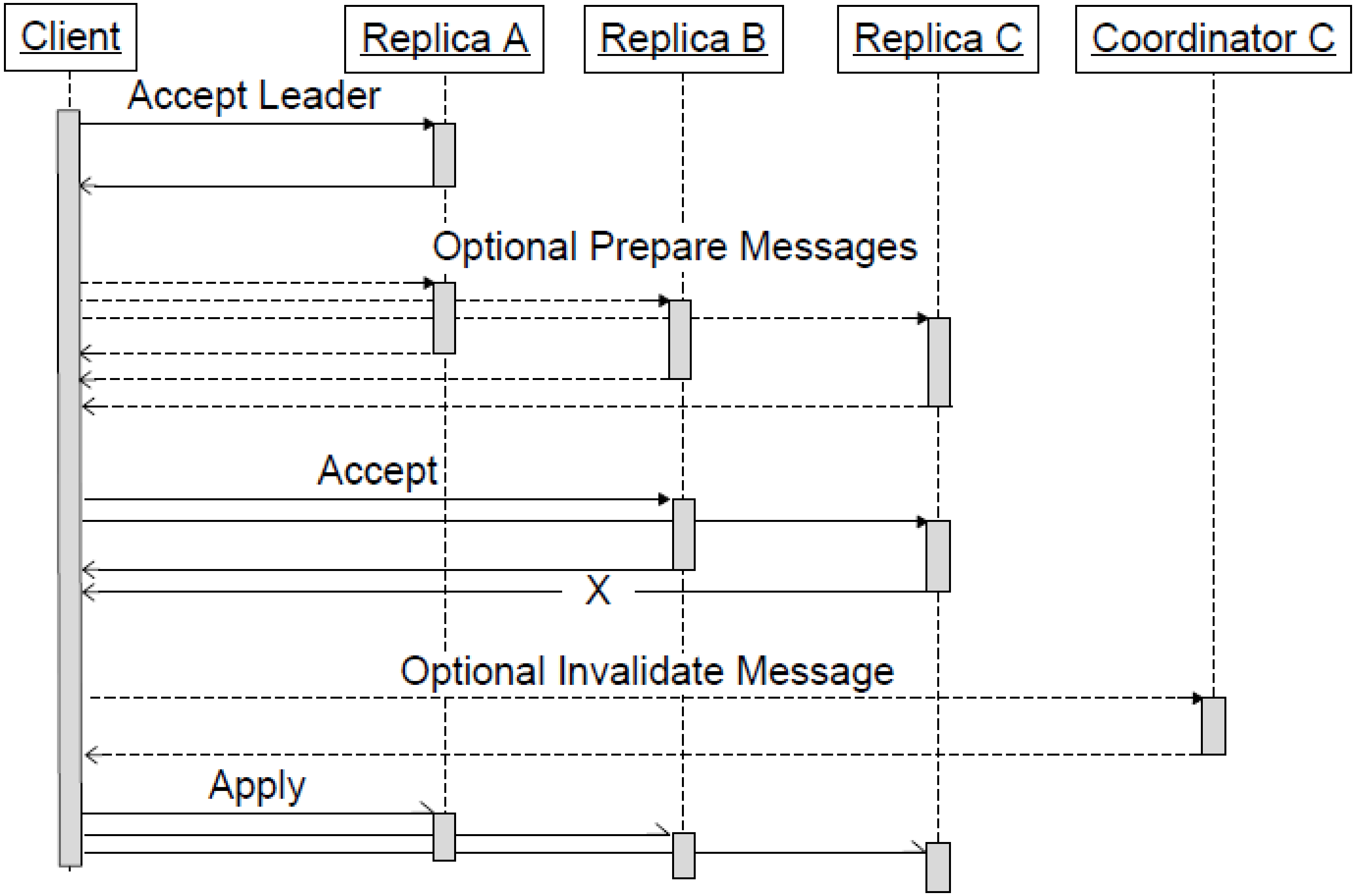


Figure 8: Timeline for writes

Coordinator Availability

Coordinator is a simpler process than Bigtable with much more stability. But it still has the risk to crash or other situations that cause its unabaliable.

1. Reader

   "To precess a request, a coordinator must hold a majority of its locks."

2. Writer

   Test the coordinator whether it still has locks.

If a live coordinator suddenly lost network connection, writers has to wait for the lock expired or a maunal repair.

Distribution of Availability

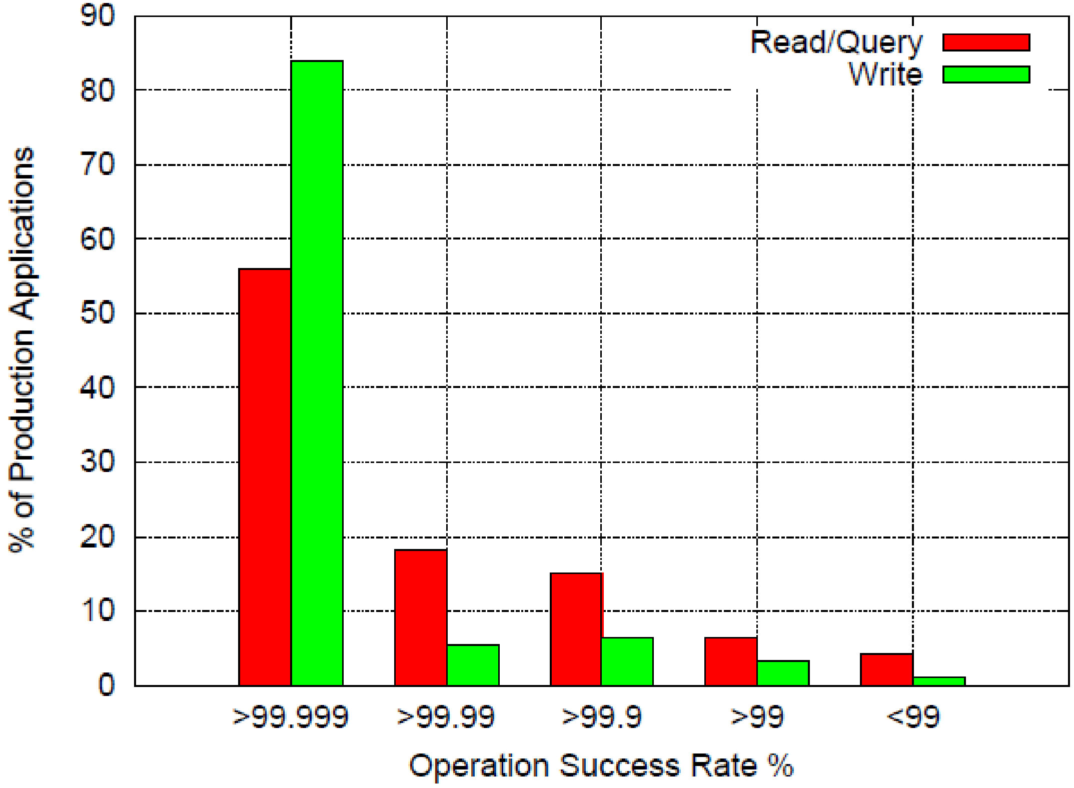


Figure 9: Distribution of Availability

Latency

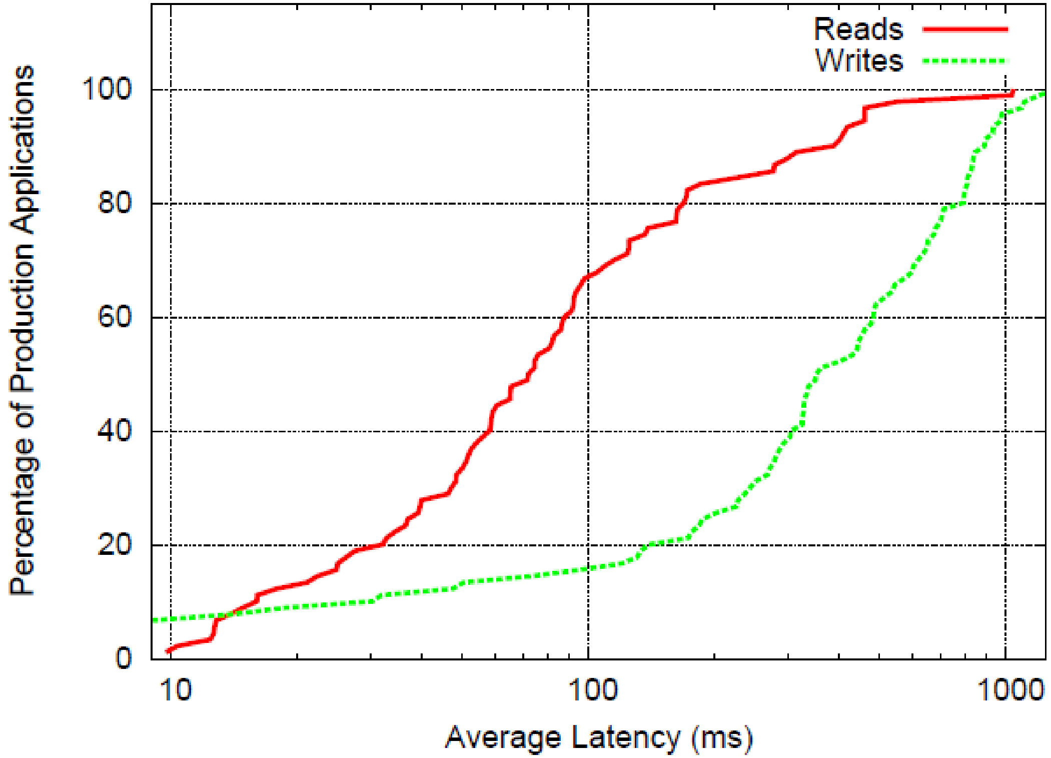


Figure 10: Distribution of Average Latencies

5

CONCLUSION

Conclusion

In this presentation, we make introduction for the following concepts:

1. Entity Groups;
2. Data Model and MVCC;
3. Paxos in Megastore;
4. Replication algorithm.

Since Megastore is the origin of many distributed systems, we truely hope our audience can fully understand the revolutional concept and theory that Megastore brings to us.

One step further, keep the motivation for learing and innovating is the fundamental motive power towards truth.

「Stay Hungry, Stay Foolish.」
By Steve Jobs