NewSQL and Determinism

Shan-Hung Wu and DataLab

CS, NTHU

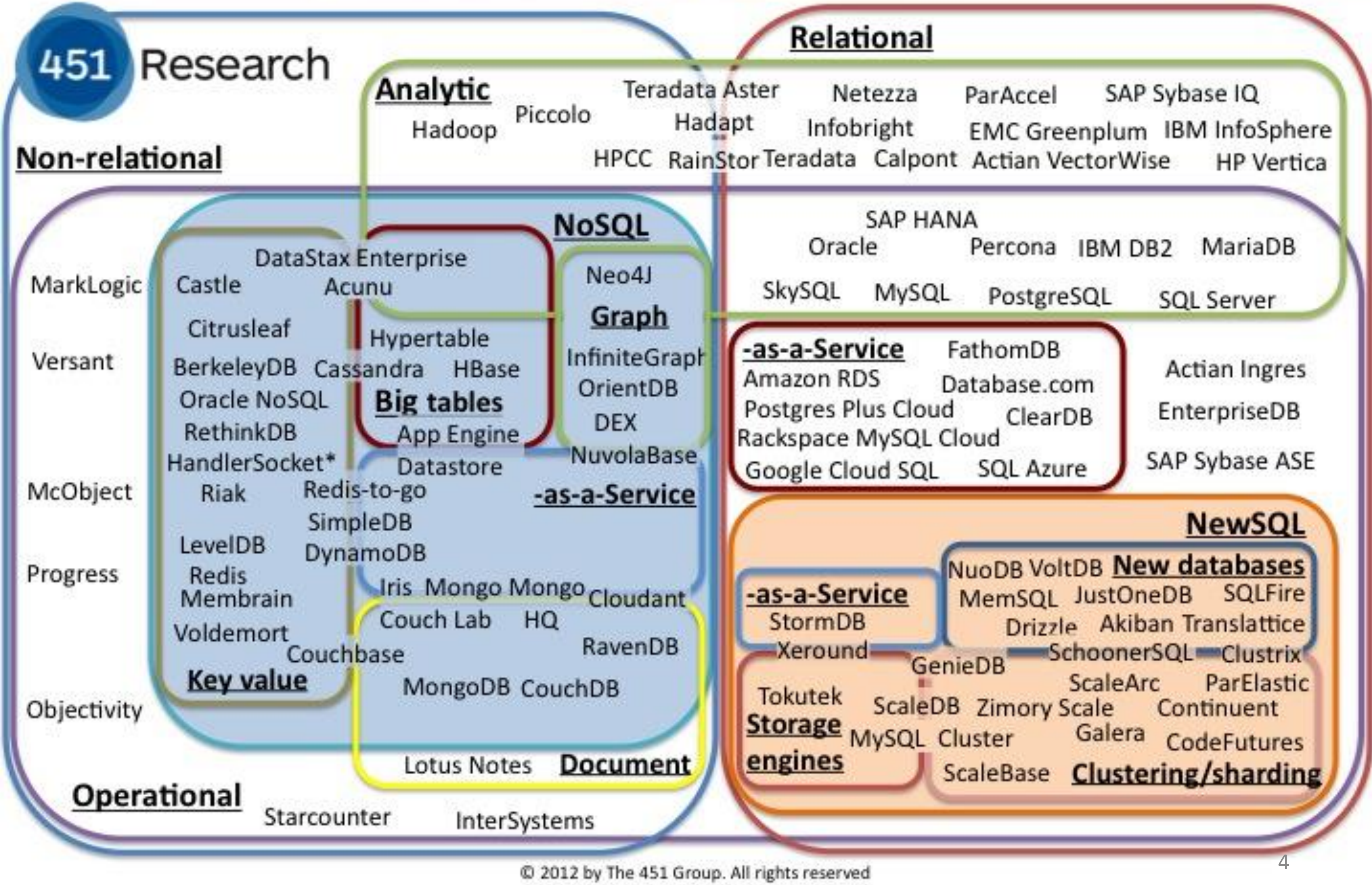
Outline

- The NewSQL landscape
- Determinism for SAE
- Implementing a deterministic DDBMS
  - Ordering requests
  - Ordered locking
  - Dependent transactions
- Performance study

Outline

- The NewSQL landscape
- Determinism for SAE
- Implementing a deterministic DDBMS
  - Ordering requests
  - Ordered locking
  - Dependent transactions
- Performance study

The evolving database landscape



451 Research

Non-relational

Relational

Analytic

Hadoop Piccolo Teradata Aster Hadapt HPCC RainStor Teradata Netezza Infobright Calpont ParAccel EMC Greenplum Actian VectorWise SAP Sybase IQ IBM InfoSphere HP Vertica

SAP HANA Oracle Percona IBM DB2 MariaDB

SkySQL MySQL PostgreSQL SQL Server

MarkLogic Versant McObject Progress Objectivity

NoSQL

DataStax Enterprise Castle Acunu

Citrusleaf BerkeleyDB Oracle NoSQL RethinkDB HandlerSocket* Riak LevelDB Redis Membrain Voldemort

Key value

Hypertable Cassandra HBase

Big tables

App Engine Datastore

Neo4J

Graph

InfiniteGraph OrientDB DEX NuvolaBase

Redis-to-go SimpleDB DynamoDB Iris Mongo Mongo Cloudant

-as-a-Service

Couch Lab HQ Couchbase RavenDB MongoDB CouchDB Lotus Notes

Document

Operational

Starcounter InterSystems

-as-a-Service

Amazon RDS FathomDB Database.com Postgres Plus Cloud ClearDB Rackspace MySQL Cloud Google Cloud SQL SQL Azure

Actian Ingres EnterpriseDB SAP Sybase ASE

NewSQL

-as-a-Service

StormDB Xeround

NuoDB VoltDB **New databases**

MemSQL JustOneDB SQLFire Drizzle Akiban Translattice

GenieDB SchoonerSQL Clustrix

Tokutek **Storage engines** ScaleDB MySQL Cluster

ScaleArc ParElastic Zimory Scale Continuent Galera CodeFutures ScaleBase **Clustering/sharding**

Shall we give up the relational model?

What To Lose?

- Expressive model
  - Normal forms to avoid inert/write/delete anomalies
- Flexible queries
  - Join, group-by, etc.
- Txs and ACID across tables

Workarounds

- No expressive model
  - Your application/data must fit
- No flexible queries
  - Issue multiple queries and post-process them
- No tx and ACID across key-value pairs/documents/entity groups
  - Code around to access data “smartly” to simulate cross-pair txs

“We believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions”

- an ironic to NoSQL when Google releases Spanner

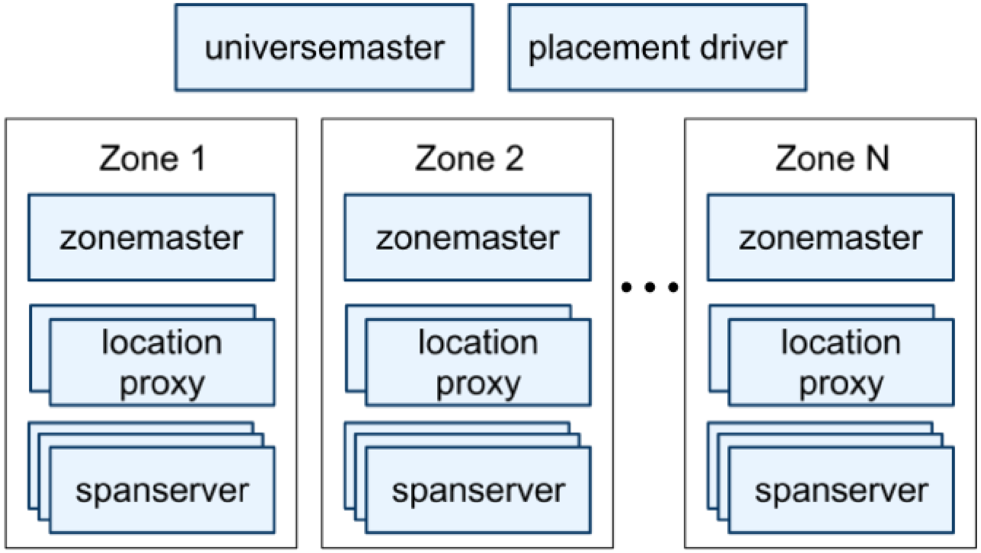
Is full DB functions + SAE possible?

Google Spanner

- Descendant of Bigtable, successor to Megastore
- Properties
  - Globally distributed
  - Synchronous cross-datacenter replication (with Paxos)
  - Transparent sharding, data movement
  - General transactions
    - Multiple reads followed by a single atomic write
    - Local or cross-machine (using 2PC)
  - Snapshot reads

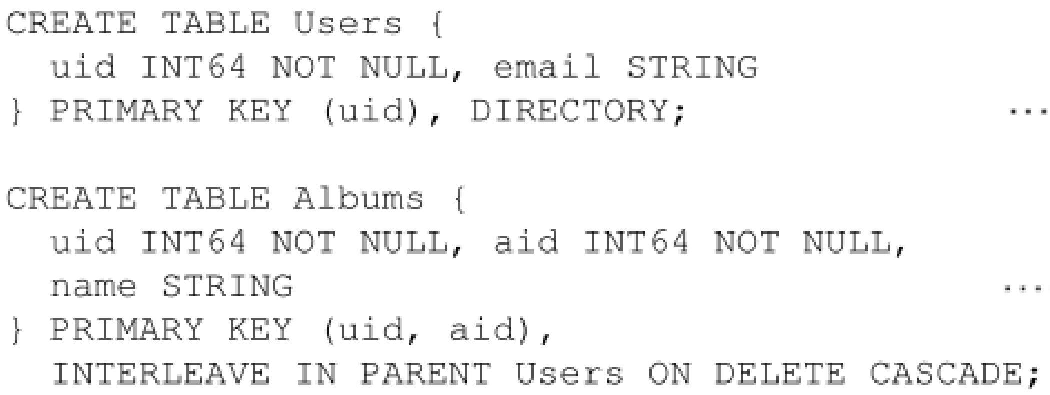
Spanner Server Organization

- Zones are the unit of physical isolation
- Zonemaster assigns data to spanservers
- Spanserver serves data (100~1000 tablets)



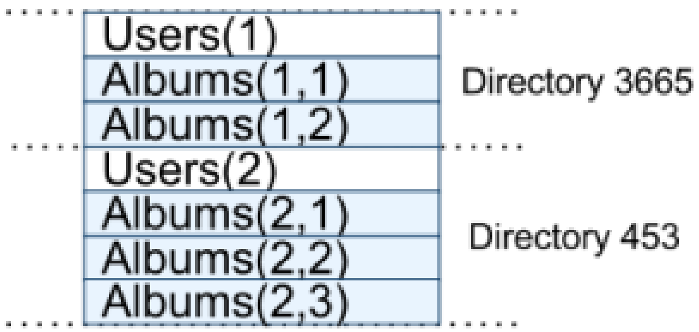
Data Model

- Spanner is semi-relational model
  - Every table has primary key (unique name of row)
- The ***hierarchy*** abstraction (assigned by client)
  - `INTERLEAVE IN` allows clients to describe the locality relationships that exist between multiple tables



```
CREATE TABLE Users {
  uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
  uid INT64 NOT NULL, aid INT64 NOT NULL,
  name STRING
} PRIMARY KEY (uid, aid),
  INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```



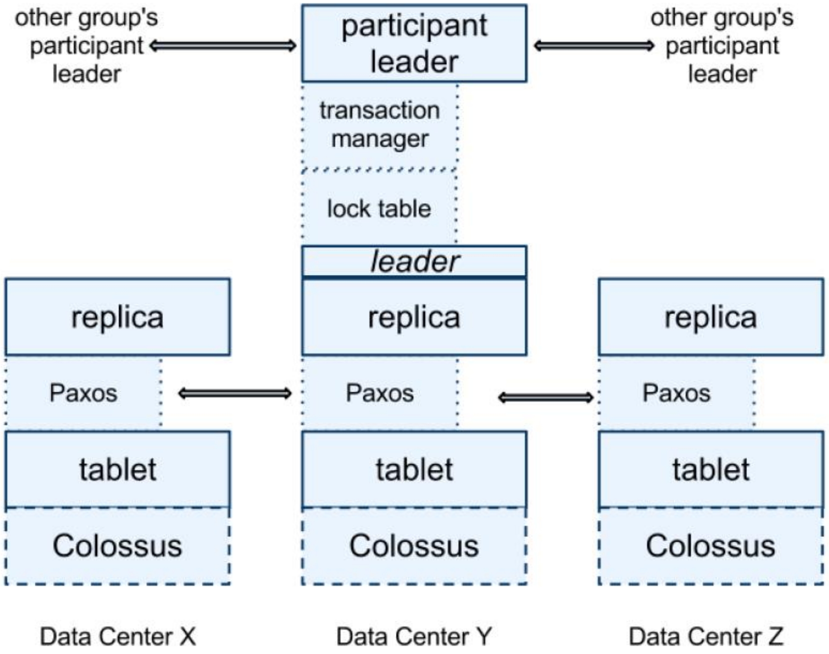
Users(1)
Albums(1,1)
Albums(1,2)
Directory 3665

Users(2)
Albums(2,1)
Albums(2,2)
Albums(2,3)
Directory 453

- Same hierarchy → same partition

Transaction Execution

- Writes are buffered at client until commit
- Commit: use Paxos to determine global serial order
- Multiple Paxos groups
  - Same hierarchy → same group
- Single Paxos group txn
  - bypass txn mgr
- Multi Paxos group txn
  - 2PC done by txn mgr



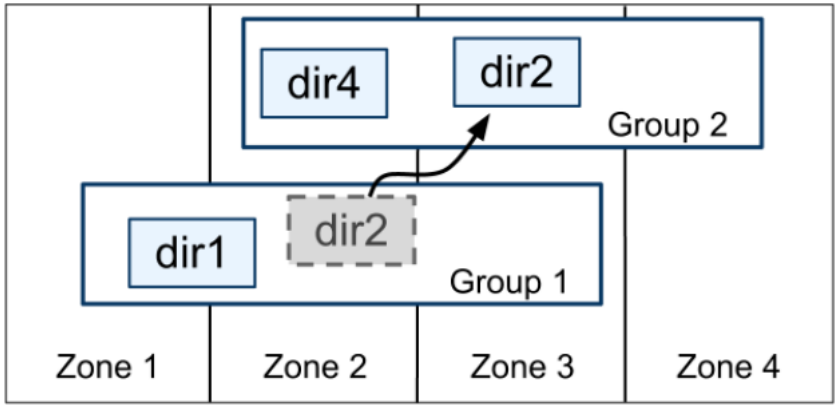
Directories

- Directories are the unit of data movement between Paxos groups
  - shed load
  - frequently accessed together
- Row's key in Spanner

  tablet may not

  contiguous

  (unlike Bigtable)

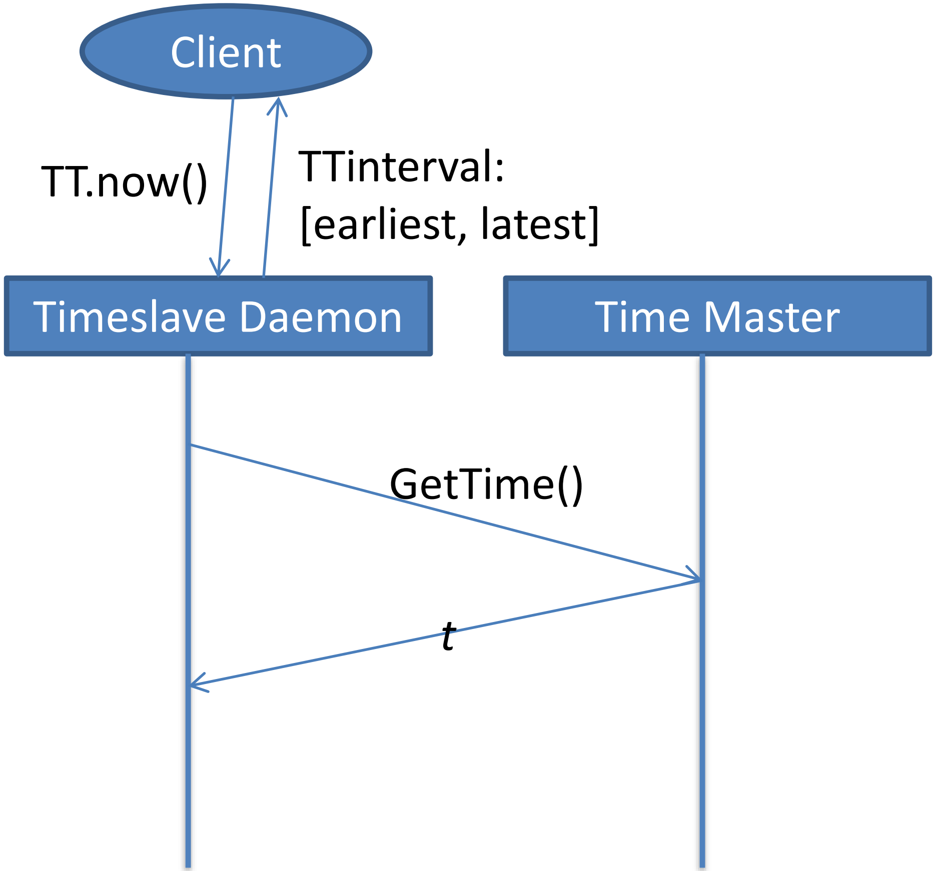


Isolation & Consistency

- Supported txns
  - Read-Write txn (need locks)
  - Read-Only txn (up-to-date timestamp)
  - Snapshot Read
- MVCC-based
- Consistent read/write at global scale
- Every r/w-txn has commit timestamp
  - Reflects the serialization order
- Snapshot reads are not blocked by writes (at the cost of snapshot isolation)

True Time

- GPS + atomic clock
- $t_{abs}(e)$: absolute time of an event *e*
- TrueTime guarantees that for an invocation $tt = TT.now(), \ \ tt.earlies \leq t_{abs}(e_{now}) \leq tt.latest,$ *where* e_{now} *is the invocation event*
- *Error bound* ϵ
  - half of the interval's width
  - *1~7 ms*

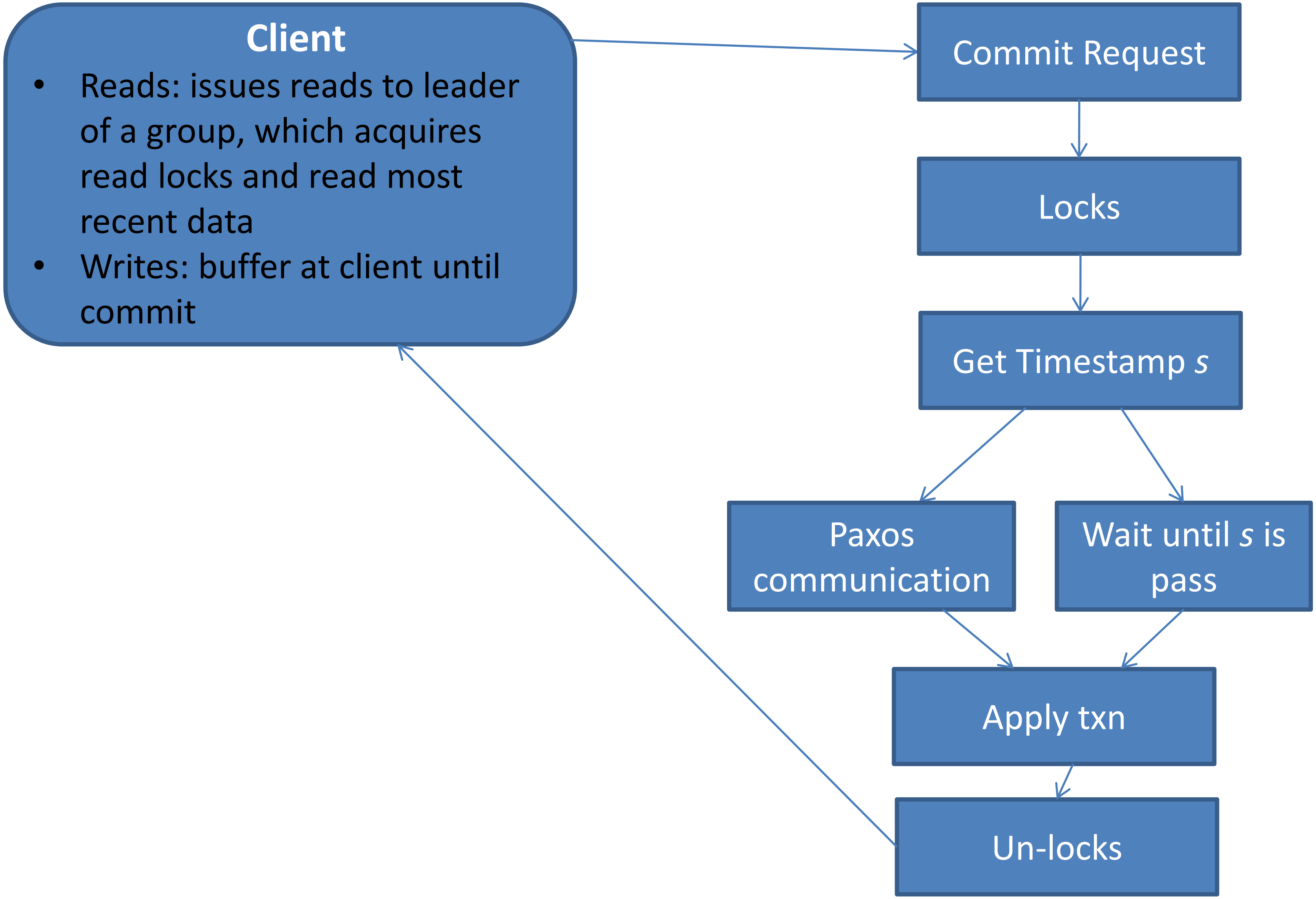


Timestamp Management

- Spanner enforces the following external consistency invariant
  - If the start of a transaction T2 occurs after the commit of a transaction T1 , then the commit timestamp of T2 must be greater than the commit timestamp of T1
  - Commit timestamp of a transaction Ti by s_i

$$t_{abs}(e_1^{commit}) < t_{abs}(e_2^{start}) \Rightarrow s_1 < s_2$$

Read-Write Txn



Client

- Reads: issues reads to leader of a group, which acquires read locks and read most recent data
- Writes: buffer at client until commit

Commit Request

Locks

Get Timestamp *s*

Paxos communication

Wait until *s* is pass

Apply txn

Un-locks

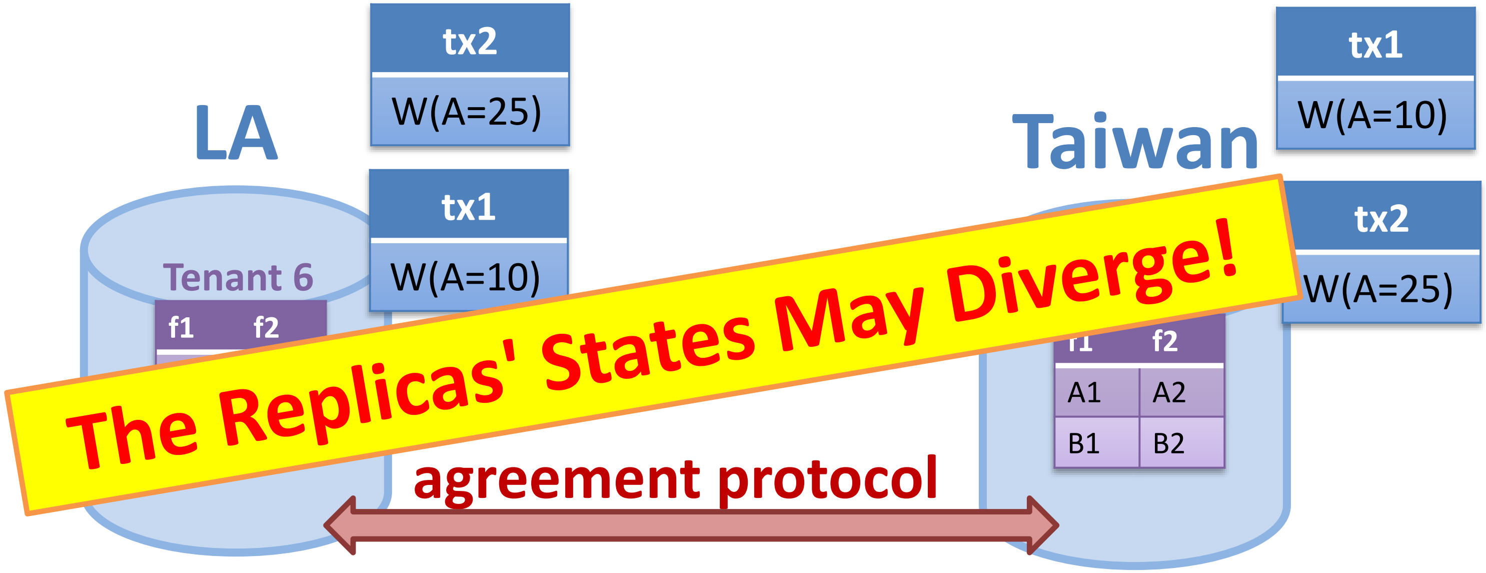
Outline

- The NewSQL landscape
- Determinism for SAE
- Implementing a deterministic DDBMS
  - Ordering requests
  - Ordered locking
  - Dependent transactions
- Performance study

What is a deterministic DDBMS?

What Is A Non-Deterministic DDBMS?

- Given Tx_1 and Tx_2, traditional DBMS guarantees ***some*** serial execution: Tx_1 -> Tx_2, or Tx_2 -> Tx_1
- Given a collection of requests/txs, DDBMS leaves the data (outcome) non-deterministically due to
  - Delayed requests, CC (lock granting), deadlock handling, buffer pinning, threading, etc.



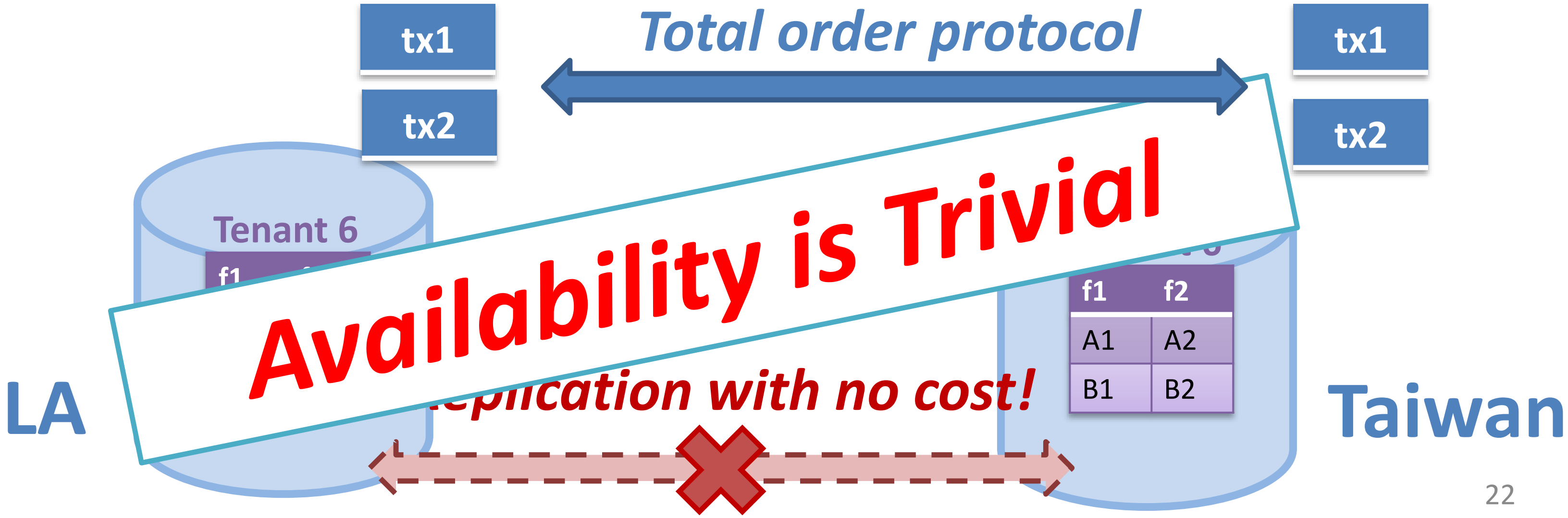
Deterministic DDBMS

- Given a collection of ***ordered*** requests/txs, if we can ensure that

  1) conflicting txs are executed on each replica following that order

  2) each tx is always run to completion (commit or abort by tx logic), even failure happens

- Then data in all replicas will be in same state, i.e., ***determinism***



f1	f2
A1	A2
B1	B2

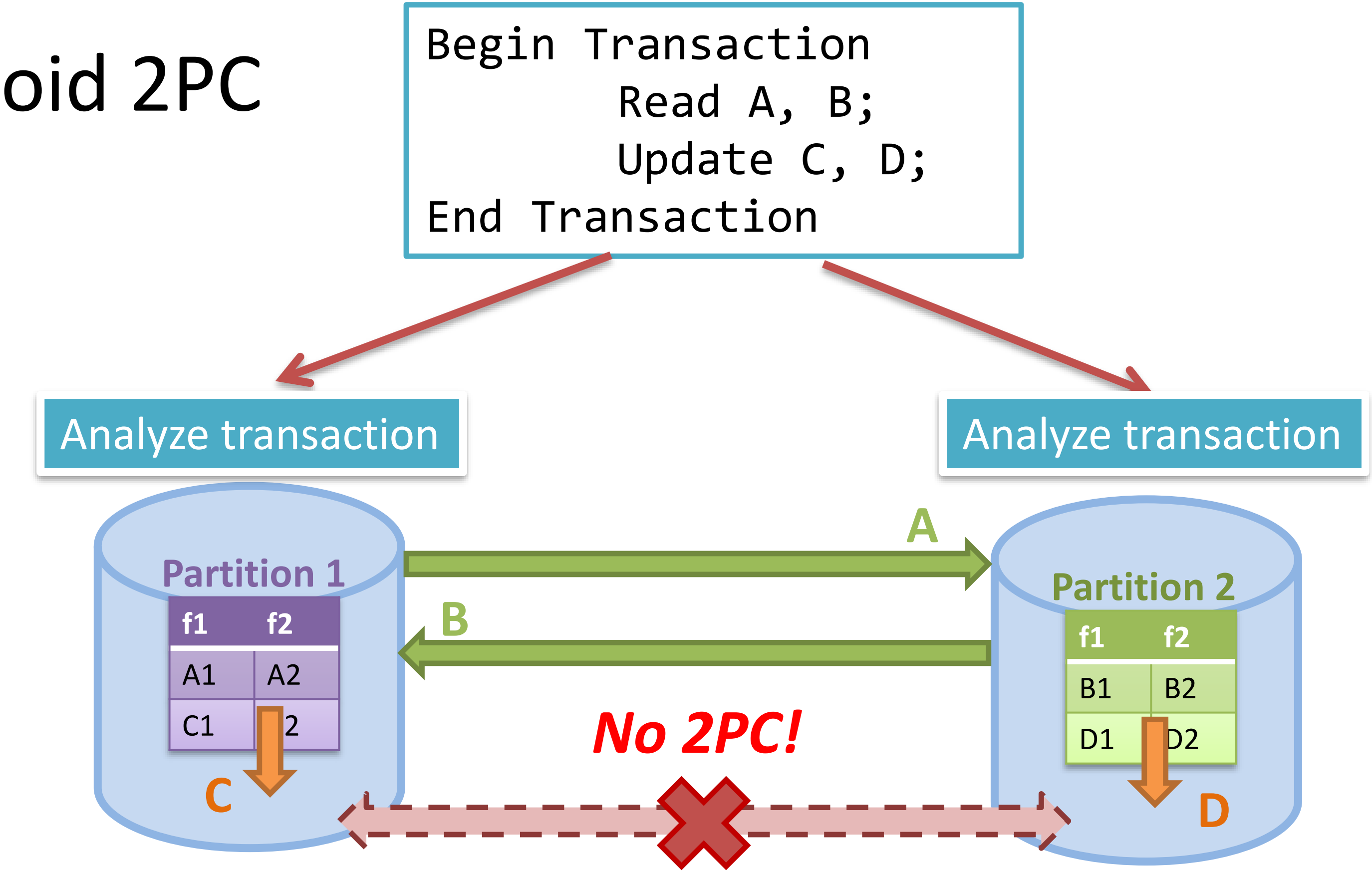
Wait, how about the delay in total-ordering?

Total-Ordering Cost

- Long tx latency
  - Hundreds of milliseconds
- However, the delay of total-ordering does ***not*** count into contention footprint
- No hurt on throughput!

Scalability via Push-based Dist. Txs.

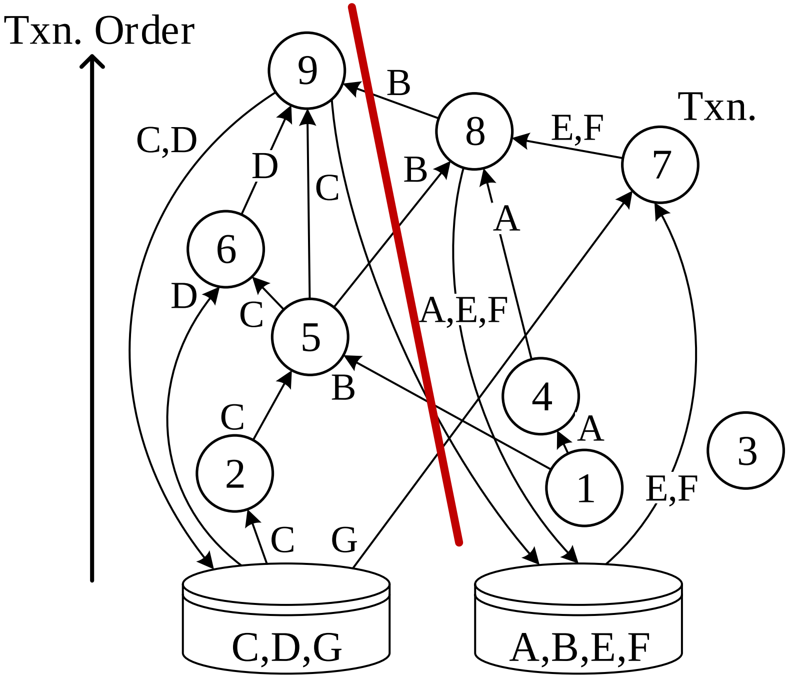
- Push readings proactively
- Parallel I/Os
- Avoid 2PC



```
Begin Transaction
       Read A, B;
       Update C, D;
End Transaction
```

T-Part

- Better tx dispatching → forward pushes

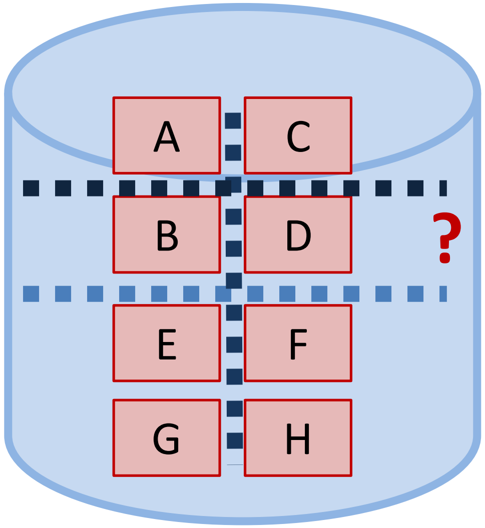


Elasticity

- Based on currently workload, we need to decide:
  1. How to re-partition data
  2. How to migrate data?

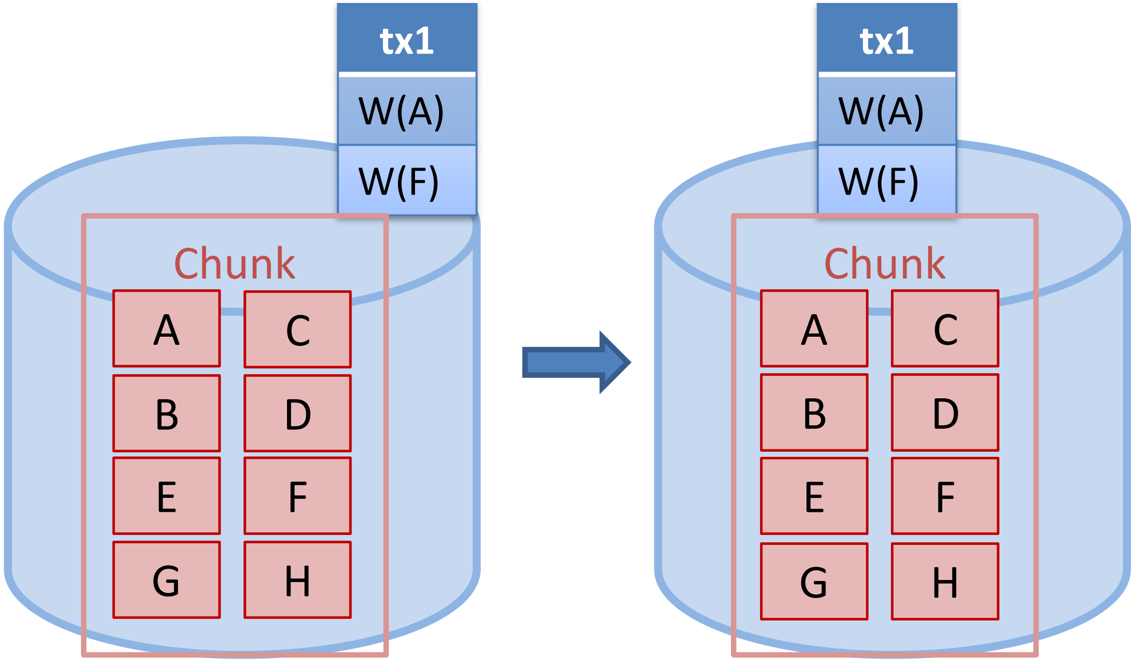
Re-Partitioning

- Data chunking
  - User-specified (e.g., Spanner hierarchy), or
  - System generated (e.g., consistent hashing)
- Master server for load monitoring
  - Practical for load balancing in a single datacenter



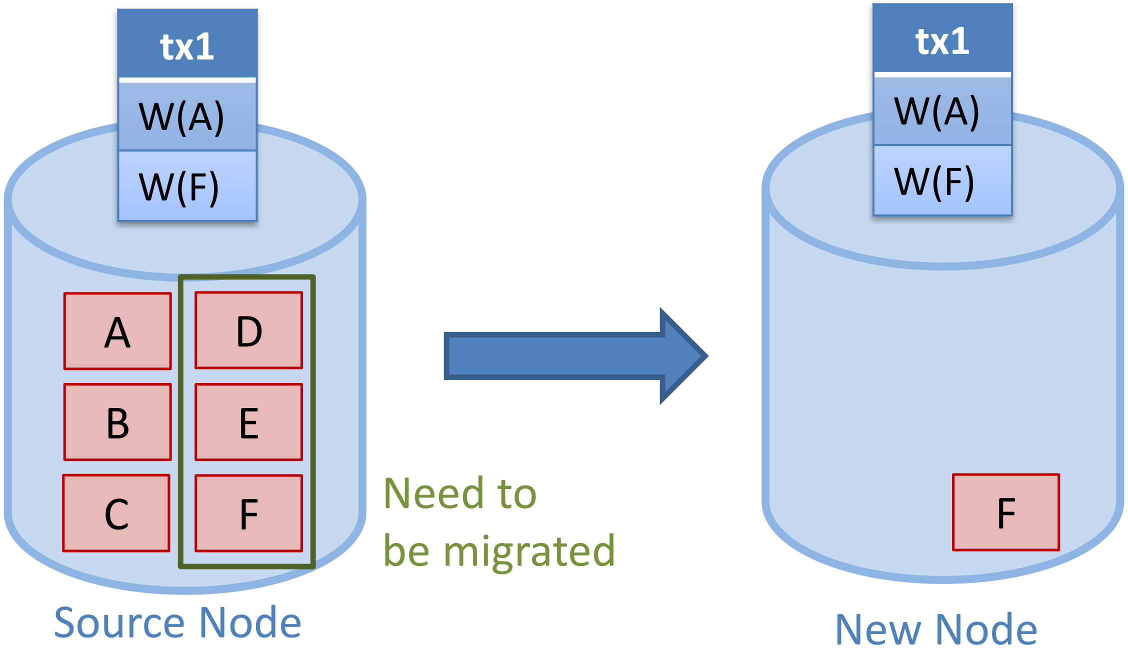
Migration ***without*** Determinism

- Txs that access data being migrated are blocked
- User-aware down time due to slow migration process



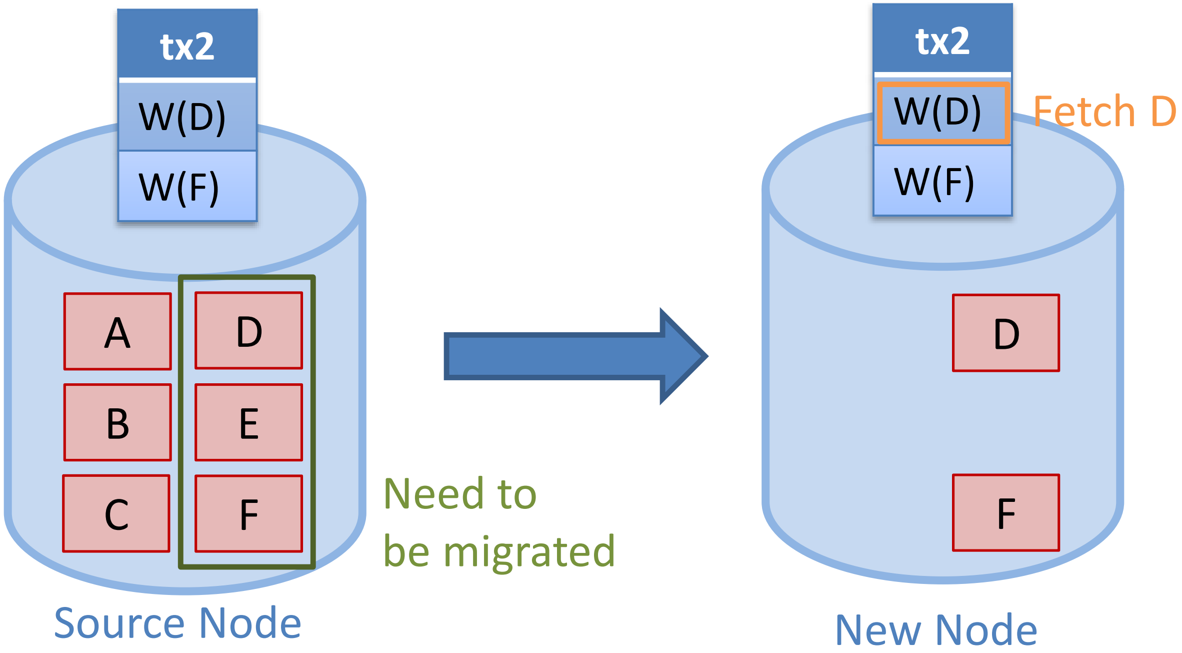
Migration ***with*** Determinism

- Replication is ***cheap*** in deterministic DBMSs
- To migrate data from a source node to a new node, the source node ***replicates*** data by executing txs in ***both*** nodes



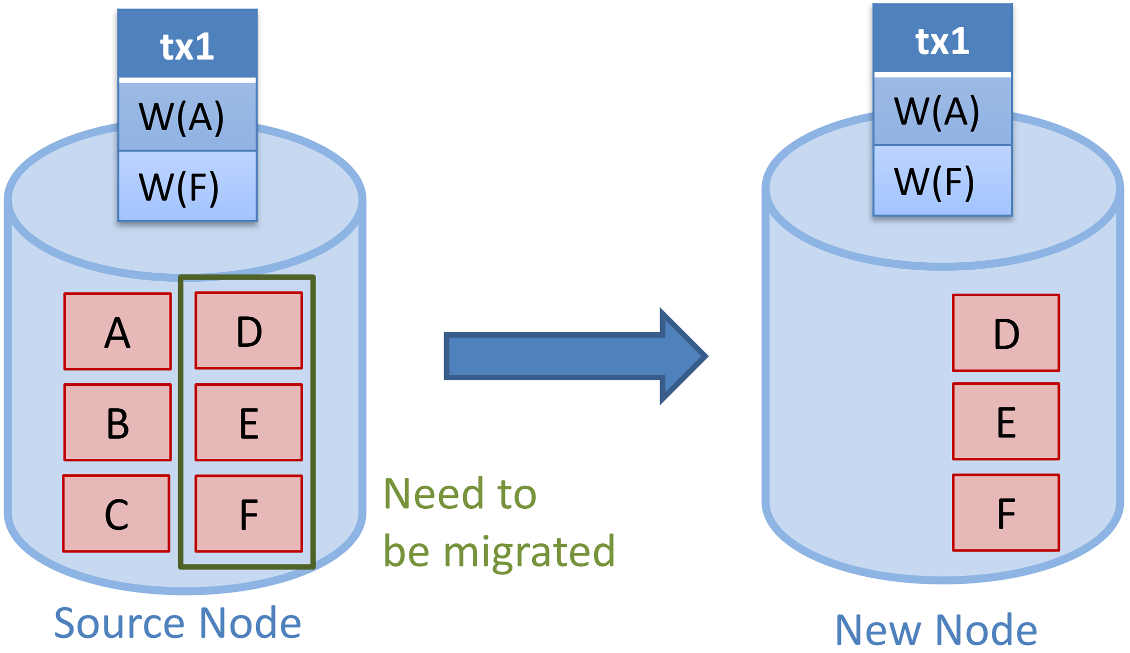
Foreground Pushing

- The source node ***pushes*** the data that txs need to access on the new node
  - If some of fetched records also need to be migrated, the new node will store those records



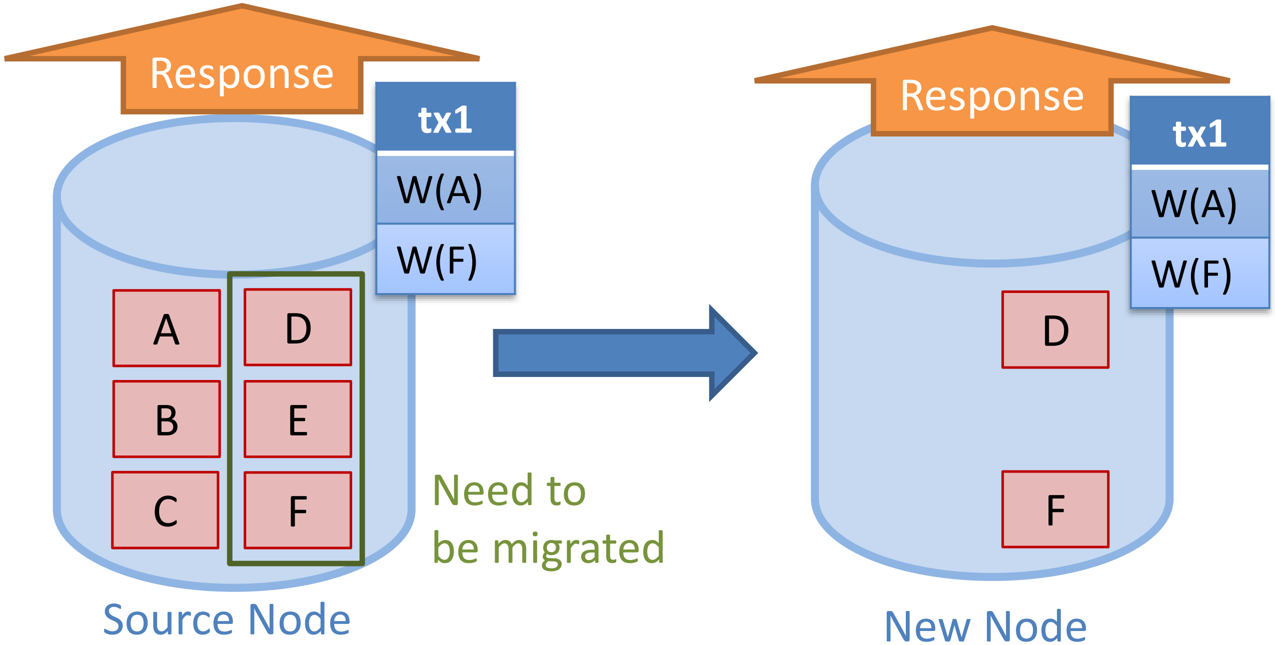
Background Pushing

- The rest of records which are not pushed by txs can be pushed by a background ***async-pushing*** thread
- Then, the source node deletes the migrated records



Migration vs. Crabbing

- Client served by ***any*** node running faster
  - – Source node in the beginning; dest. node at last
- Migration delay imperceptible



How to implement a deterministic DDBMS?

Outline

- The NewSQL landscape
- Determinism for SAE
- Implementing a deterministic DDBMS
  - Ordering requests
  - Ordered locking
  - Dependent transactions
- Performance study

Outline

- The NewSQL landscape
- Determinism for SAE
- Implementing a deterministic DDBMS
  - Ordering requests
  - Ordered locking
  - Dependent transactions
- Performance study

Implementing Deterministic DDBMS

- Requests need to be ordered
- A deterministic DBMS engine for each replica

Ordering Requests

- A ***total ordering protocol*** needed
  - Replicas have consensus on the order of requests
- A topic of ***group communication***
- To be discussed later

The Sources of Non-Determinism

- Thread and process scheduling
- Concurrency control
- Buffer and cache management
- Hardware failures
- Variable network latency
- Deadlock resolution schemes

How to make a DBMS deterministic (given the total-ordered requests)?

Outline

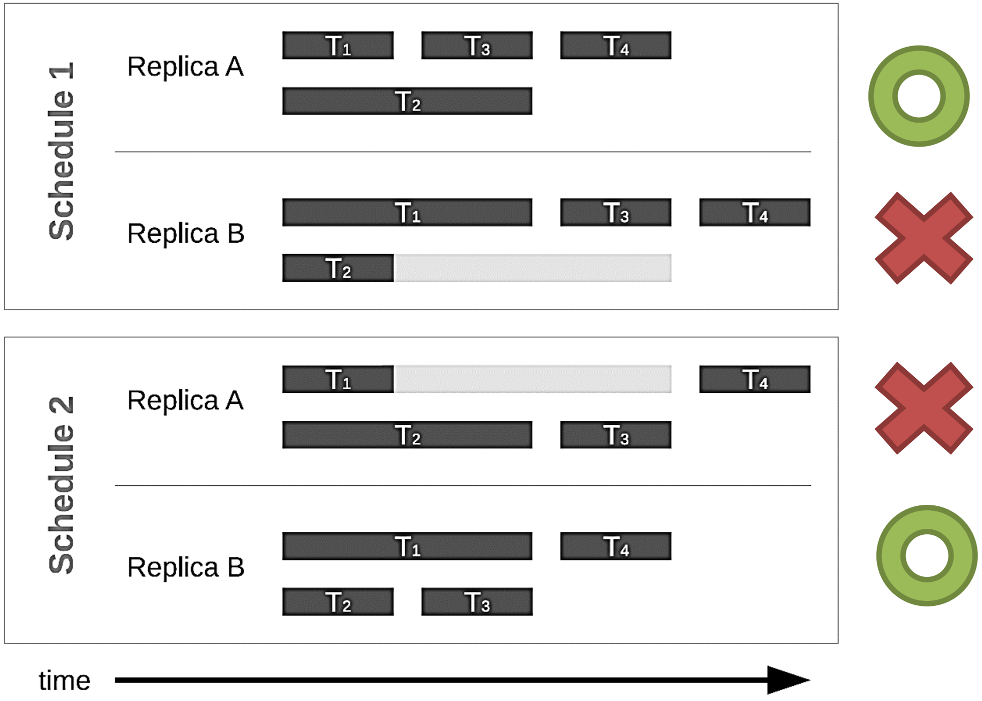
- The NewSQL landscape
- Determinism for SAE
- Implementing a deterministic DDBMS
  - Ordering requests
  - Ordered locking
  - Dependent transactions
- Performance study

Single-Threaded Execution

- Execute the ordered requests/txs one-by one
- Good idea?
- Poor performance?
  - No pipelining between CPU and I/O or networking
  - Does not take the advantage of multi-core
  - The order of non-conflicting txs doesn't matter
- Walk-arounds: H-store

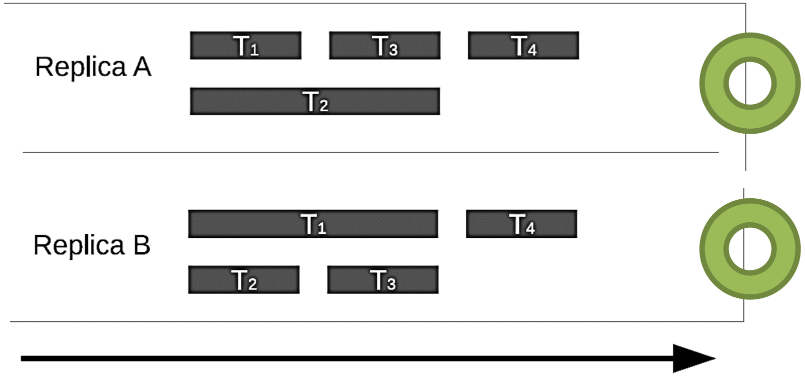
Fix-Ordered Multi-Thread Execution?

- T1, T2, and T3 do not conflict with each other
- T4 conflicts with all T1, T2, and T3



Dynamic Reordering is Better

- Schedule of conflicting txs are the same across all replicas
- Schedule of non-conflicting txs can be "***re-ordered***" automatically
  - Better pipelining to increase system utilization
  - Shorter latency



How? S2PL?

How to make S2PL deterministic?

A Case of Determinism

- Based on S2PL:
- ***Conservative***, ordered locking
- Each tx is always run to completion (commit or abort by tx logic), ***even failure happens***

Conservative, Ordered Locking

- Each tx has ID following the total order
- Requests ***all*** it locks ***atomically*** upon entering the system (conservative locking)
- For any pair of transactions T_i and T_j which both request locks on some record r, if $i \ < j$ then T_i must request its lock on r before T_j
- ***No deadlock!***

Execute to Completion

- Each node logs “requests”
  - Operations are idempotent if a tx is executed multiple times at the same place of a schedule
- No system-decided rollbacks during recovery
  - Recovery manager rollbacks all operations of incomplete txs
  - But loads request logs and ***replay*** those incomplete txs following the total order

Wait, how to know which object to lock ***before*** executing a tx?

Outline

- The NewSQL landscape
- Determinism for SAE
- Implementing a deterministic DDBMS
  - Ordering requests
  - Ordered locking
  - Dependent transactions
- Performance study

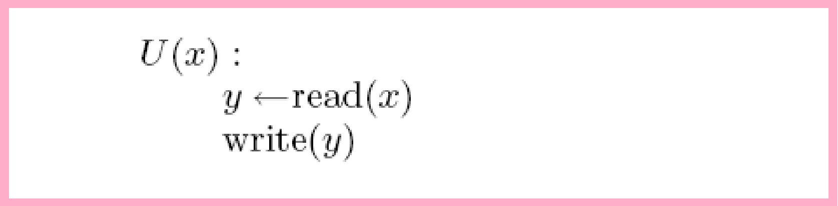
Dependent Transactions

- It isn’t necessarily possible for every tx to know which object to lock before execution
- Example: ***dependent txs***
  - SQL: UPDATE … SET … WHERE …
  - *x* is a field; *y* are values of another field
  - Which to lock for *y*?
  - Option1: the entire table where *y* belongs to

```
$U(x)$:
        $y \leftarrow read(x)$
        $write(y)$
```

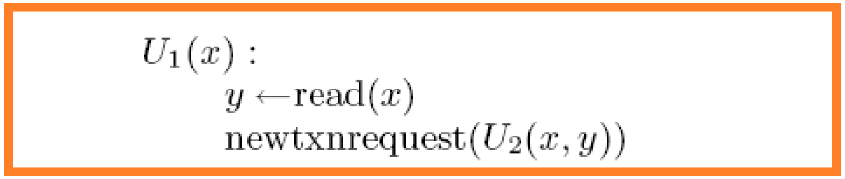
Rewriting Dependent Txs

- Option 2: rewrite tx!

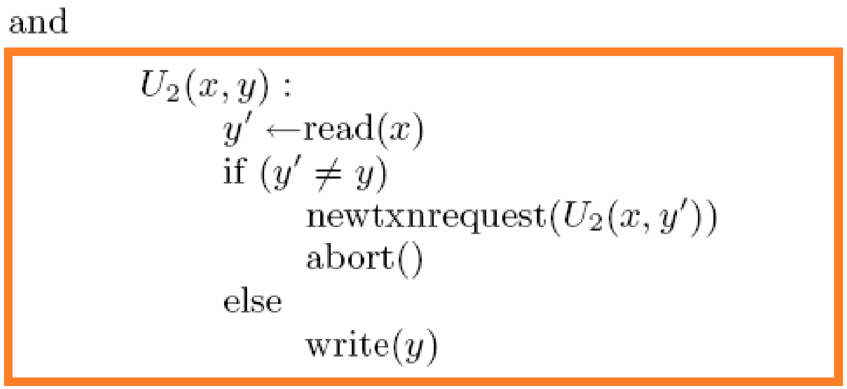


```
$U(x)$ :
    $y \leftarrow \mathrm{read}(x)$
    $\mathrm{write}(y)$
```

⇩ Reconnaissance tx

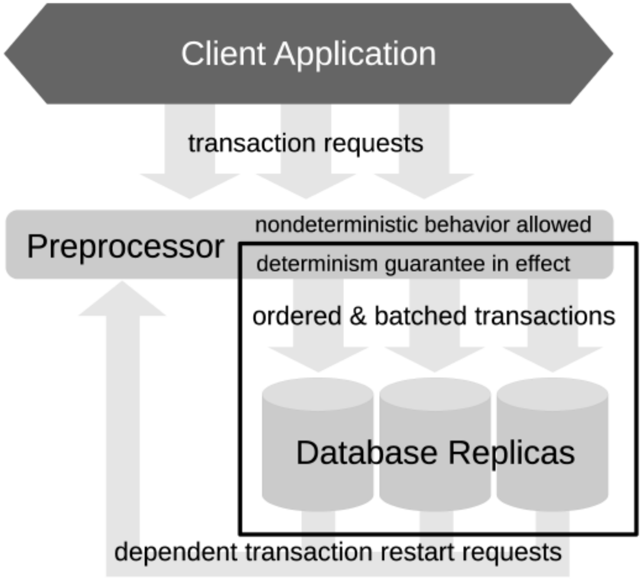


```
$U_1(x)$ :
    $y \leftarrow \mathrm{read}(x)$
    $\mathrm{newtxnrequest}(U_2(x,y))$
```



and

```
$U_2(x,y)$ :
    $y' \leftarrow \mathrm{read}(x)$
    if $(y' \neq y)$
        $\mathrm{newtxnrequest}(U_2(x,y'))$
        abort()
    else
        $\mathrm{write}(y)$
```


Client Application
transaction requests
Preprocessor
nondeterministic behavior allowed
determinism guarantee in effect
ordered & batched transactions
Database Replicas
dependent transaction restart requests

igure 2: Architecture of a deterministic DBMS.

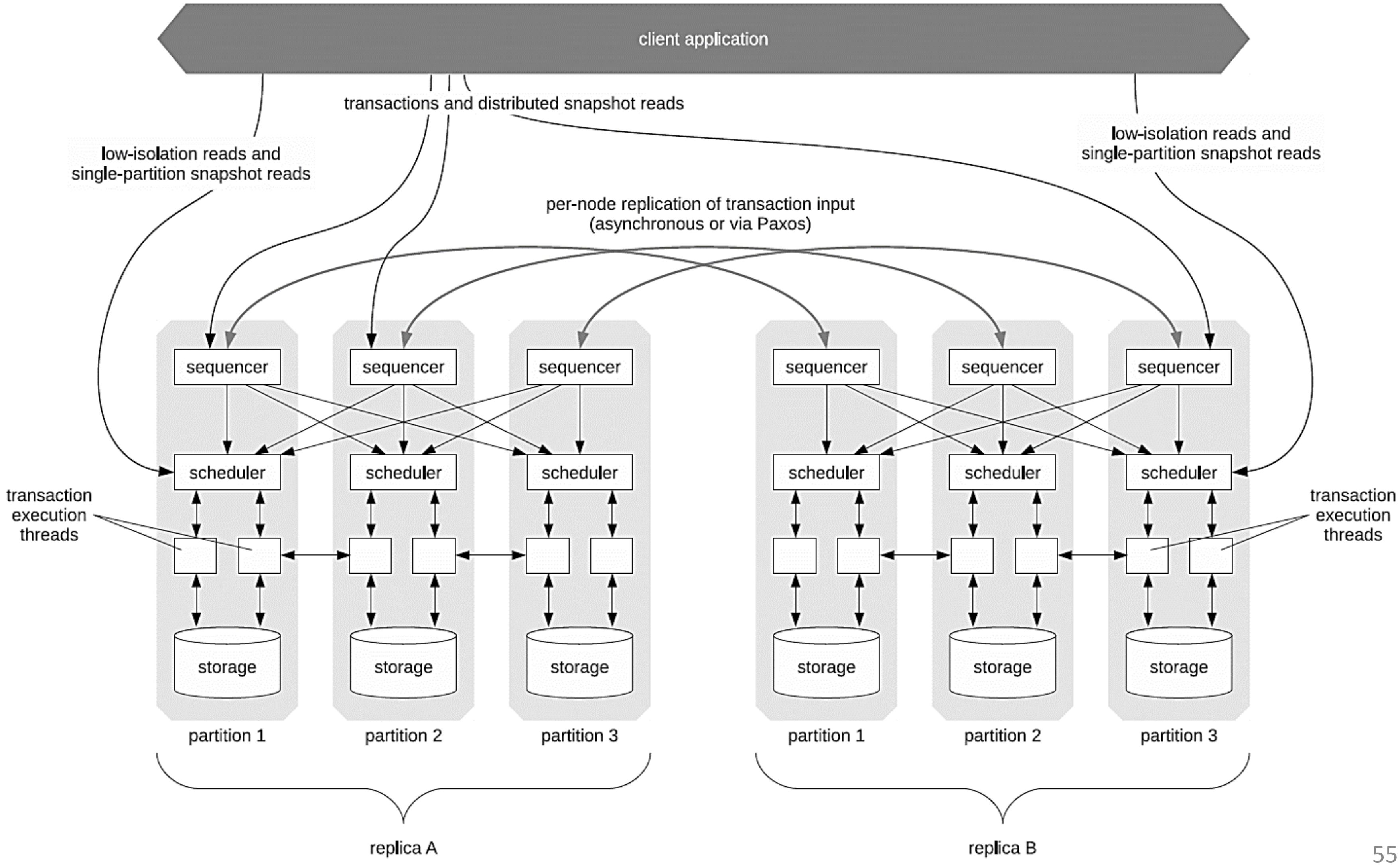
Rewriting Dependent Txs

- Pros?
  - Better concurrency for *y* (higher throughput)
- Overhead?
  - Increased latency
  - Higher abort rate if values in *x* are modified between U1 and U2
- Luckily, values in *x* may not be modified that often in practice
- Why?
  - Because otherwise you won’t use *x* as a search field
  - *x* is usually (secondarily) indexed

Outline

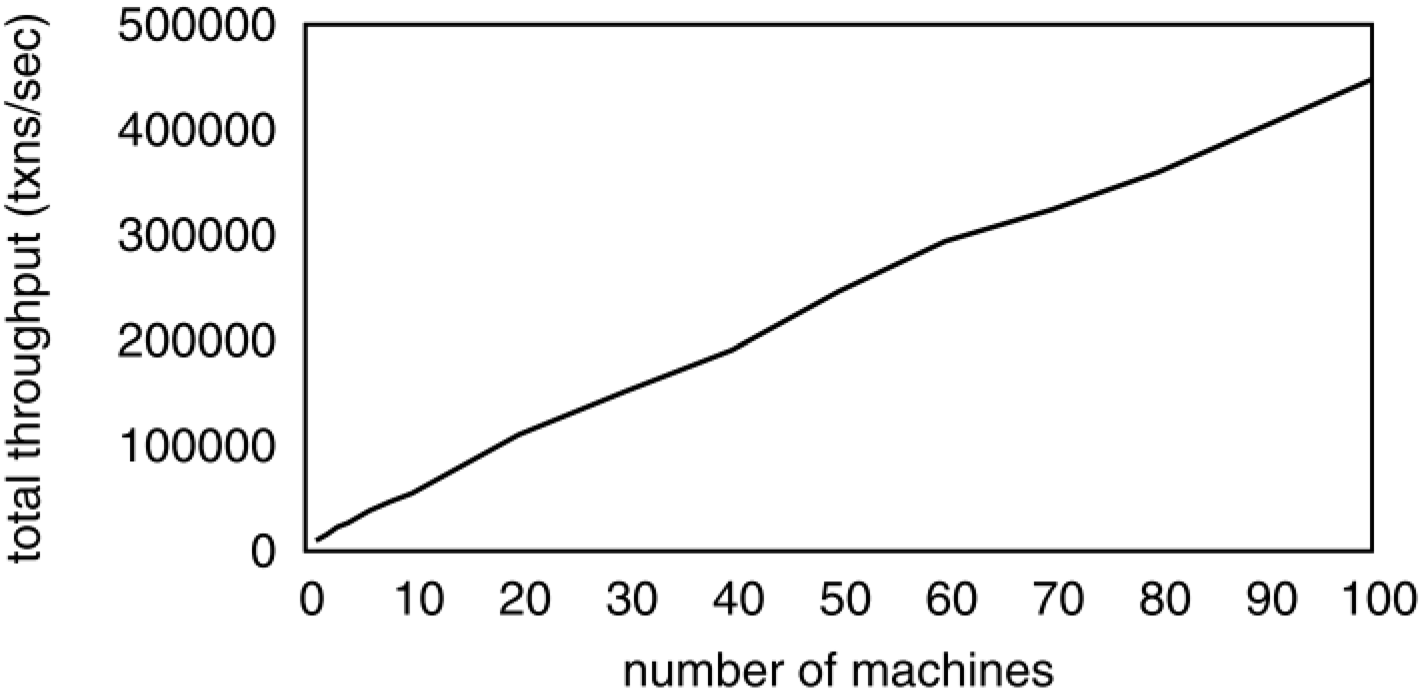
- The NewSQL landscape
- Determinism for SAE
- Implementing a deterministic DDBMS
  - Ordering requests
  - Ordered locking
  - Dependent transactions
- Performance study

A Deterministic DBMS Prototype


client application
transactions and distributed snapshot reads
low-isolation reads and single-partition snapshot reads
low-isolation reads and single-partition snapshot reads
per-node replication of transaction input (asynchronous or via Paxos)
sequencer
sequencer
sequencer
sequencer
sequencer
sequencer
scheduler
scheduler
scheduler
scheduler
scheduler
scheduler
transaction execution threads
transaction execution threads
storage
storage
storage
storage
storage
storage
partition 1
partition 2
partition 3
partition 1
partition 2
partition 3
replica A
replica B

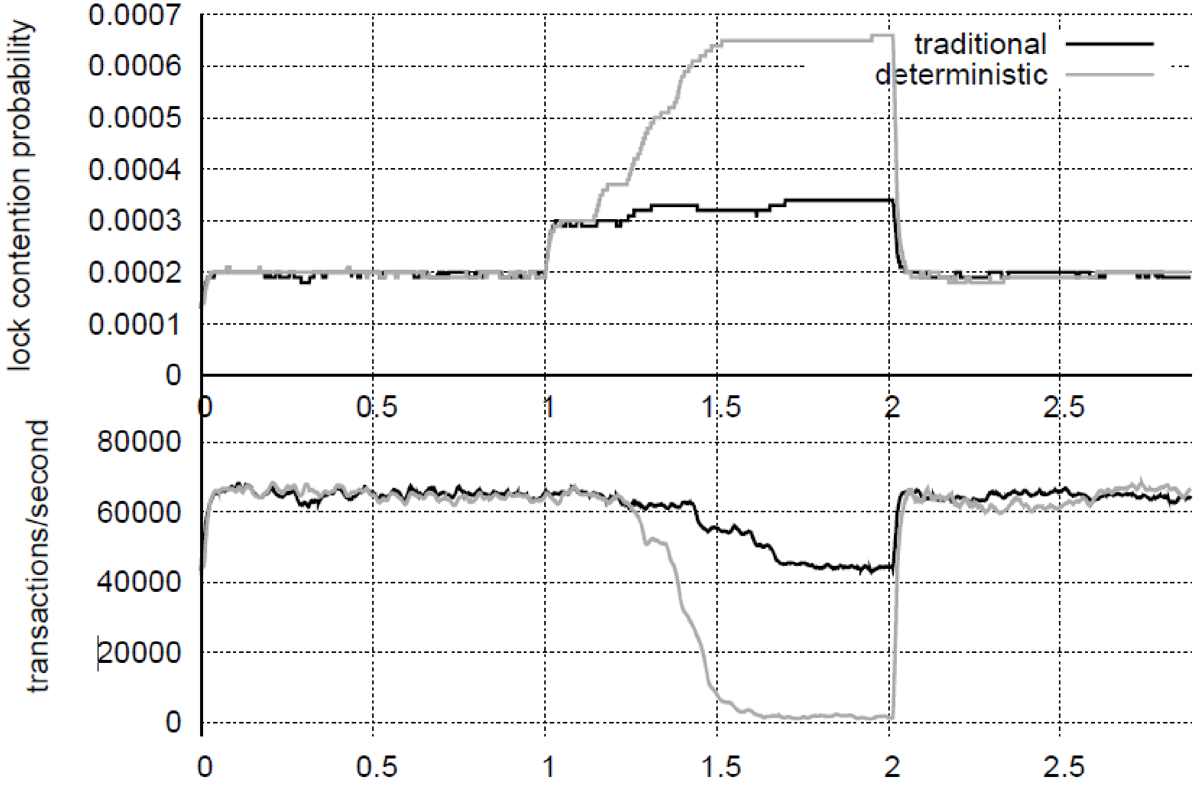
The Scalability of The Prototype

- Workload: TPC-C New Order Transactions



The Clogging Problem

- Workload: two txs conflict with probability 0.01%



A long tx arrives

Why?

Stalled Transaction and Clogging

- Clogging due to ***stalled txs***
  - Following up txs conflicting with the long tx
- Why not an issue in traditional S2PL?
  - On-the-fly “re-ordering” between conflicting txs
  - E.g., with wound-wait deadlock prevention
- Conservative CC is suitable only for ***workloads consisting of short txs***
  - Less-stalled txs
- In-memory DBMS (or alike) is a good fit
  - E.g., modern web deployments with Memcached between app server and DBMS

Review: Assumptions/Observations

- No ad-hoc txs
  - All txs are predefined (e.g., as stored procedures)
- No long txs
  - Minimize delay
  - Analytic jobs are moved to backend (e.g., GFS+MapReduce)
- Working set is small
- Locality

The Score Sheet

	DDBMS	Dynamo/Cassandra/Riak	Deterministic DDBMS
Data model	Relational	***Key-value***	Relational
Tx boundary	Unlimited	***Key-value***	Unlimited, ***but not ad-hoc***
Consistency	W strong, ***R eventual***	W strong, ***R eventual***	WR strong
Latency	Low (local partition only)	Low	***High*** (due to total ordering requests)
Throughput	High (***scale-up***)	High (scale-out)	High (scale-out, comm. delay doesn't impact CC)
Bottleneck/SPF	***Masters***	***Masters***	No
Consistency (F)	W strong, ***R eventual***	***WR eventual***	WR strong
Availability (F)	***Read-only***	Read-write	Read-write
Data loss (F)	***Some***	Some	No
Reconciliation	Not needed	***Needed (F)***	Not needed
Elasticity	***No***	***Manual, blocking migration***	Yes, automatic

- High latency can be "hidden" in some applications
  - E.g., Gmail (with the aid of AJAX)

Final Remarks

- Cloud databases: a un-unified world
  - Common goals: ***S***cale-out (not -up), high ***A***vailability, and ***E***lasticity
- It's not about how many DB products in the market you know
- It’s about knowing the trade-offs
  - Better architecture design if you are in DB field
  - To choose the right product(s) as a DBMS user
- Full DB functions + SAE for is still an active research problem

References

- Matthew Aslett. What we talk about when we talk about NewSQL. http://blogs.the451group.com/information_management/2011/04/06/what-we-talk-about-when-we-talk-about-newsql/
- Matthew Aslett. NoSQL, NewSQL and Beyond. http://blogs.the451group.com/information_management/2011/04/15/nosql-newsql-and-beyond/
- A Thomson et al. The Case for Determinism in Database Systems. 2010.
- A Thomson et al. Calvin: Fast Distributed Transactions for Partitioned Database Systems. 2012.
- Towards an Elastic and Autonomic Multitenant Database, A. Elmore et al. NetDB 2011.
- Workload-aware database monitoring and consolidation, C. Curino et al. SIGMOD 2011.