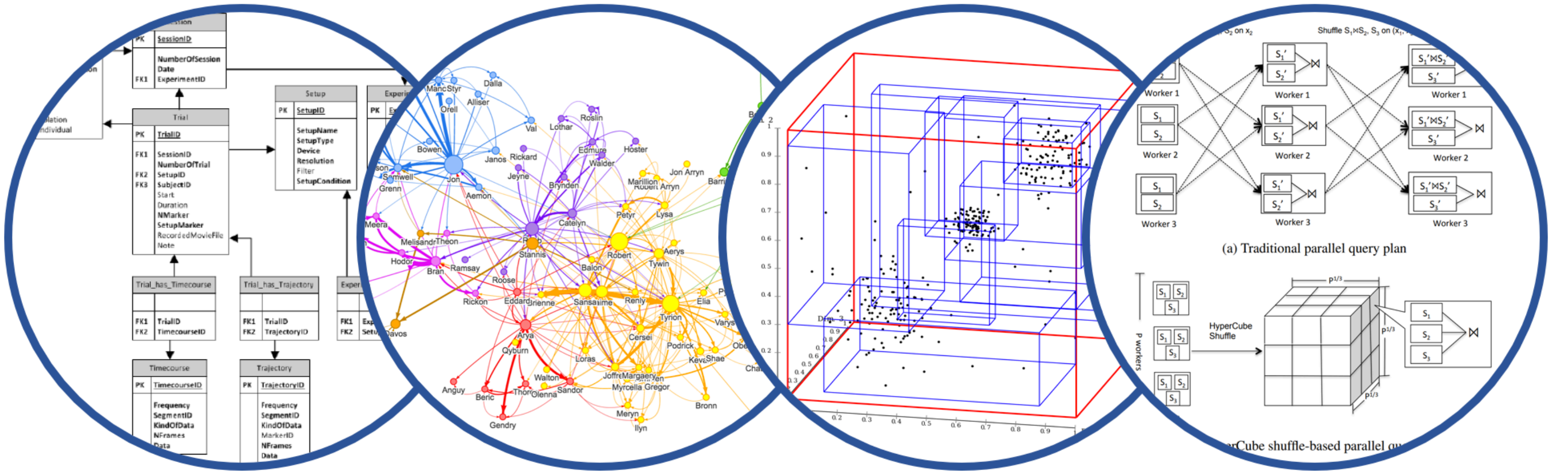


Database System Internals

Replication

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

- HW5 due tonight
- Lab4 due on Monday
- Lab5 due on June 5 / June 11. No late days
- No lab6 (we alternate with Lab5)

References

- Ullman Book Chapter 20.6

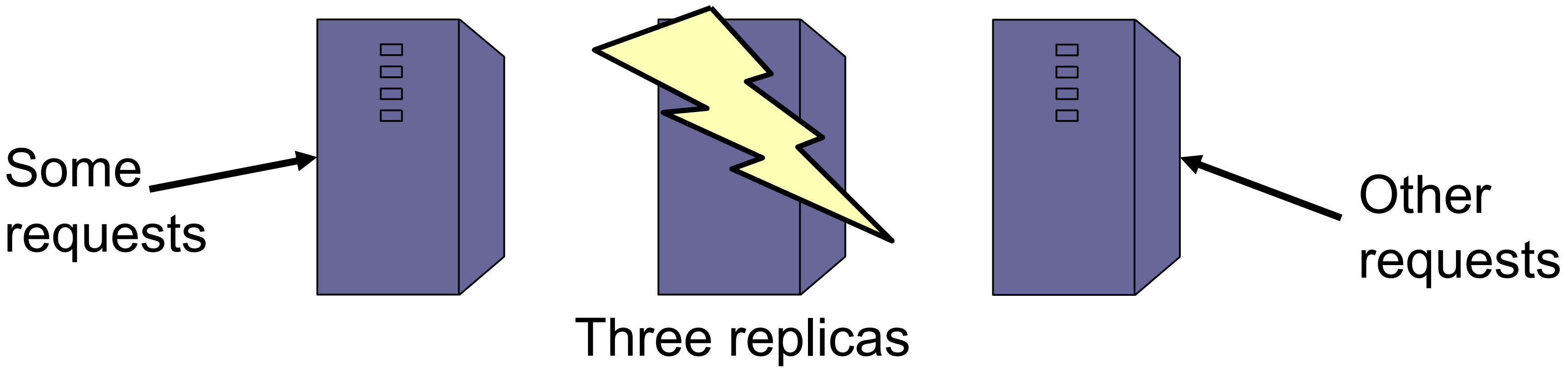
- Database management systems.
  Ramakrishnan and Gehrke.
  Third Ed. Chapter 22.11

Outline

- Goals of replication
- Three types of replication
  - Synchronous (aka eager) replication
  - Asynchronous (aka lazy) replication
  - Two-tier replication

Goals of Replication

- Goal 1: consistency. Always read latest update
- Goal 2: availability. Every request → a response
- Goal 3: performance. Fast read/writes



Discussion: NoSQL

New problem in the early 2000's

- Startup company launces Website backed up by MySQL, works fine with 50 users
- Suddenly, they are successful and have 1M users
- MySQL cannot keep up

Discussion: NoSQL

New problem in the early 2000's

- Startup company launces Website backed up by MySQL, works fine with 50 users
- Suddenly, they are successful and have 1M users
- MySQL cannot keep up

NoSQL:

- Distributed database (replication, partition)
- Give up strong consistency in favor of availability and performance (as we'll see discuss next)

Discussion: NoSQL

New problem in the early 2000's

- Startup company launces Website backed up by MySQL, works fine with 50 users
- Suddenly, they are successful and have 1M users
- MySQL cannot keep up

NoSQL:

- Distributed database (replication, partition)
- Give up strong consistency in favor of availability and performance (as we'll see discuss next)

Today: strong consistency is standard requirement

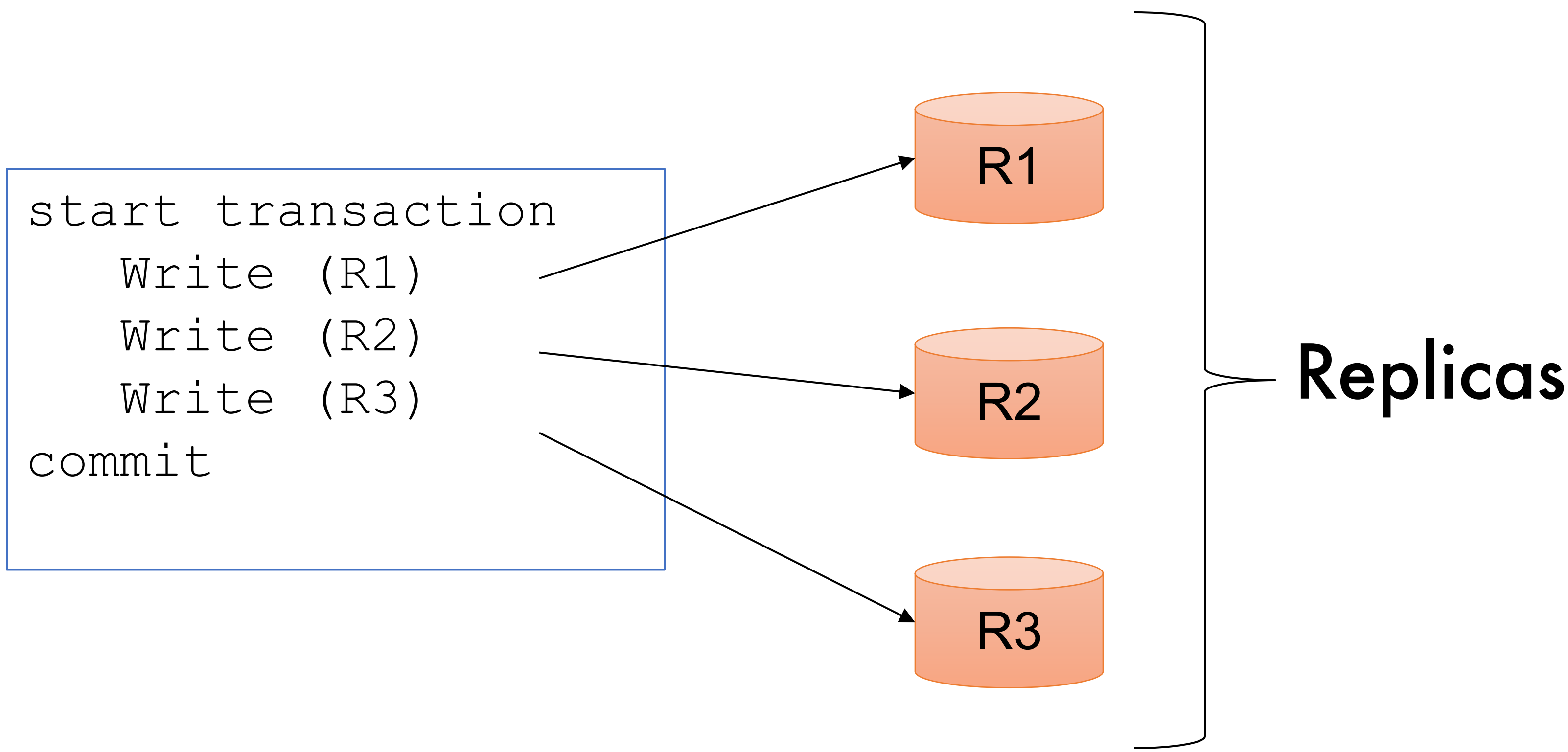
Types of Replication

	Master	Group
Synchronous	✓	
Asynchronous		

Synchronous Replication

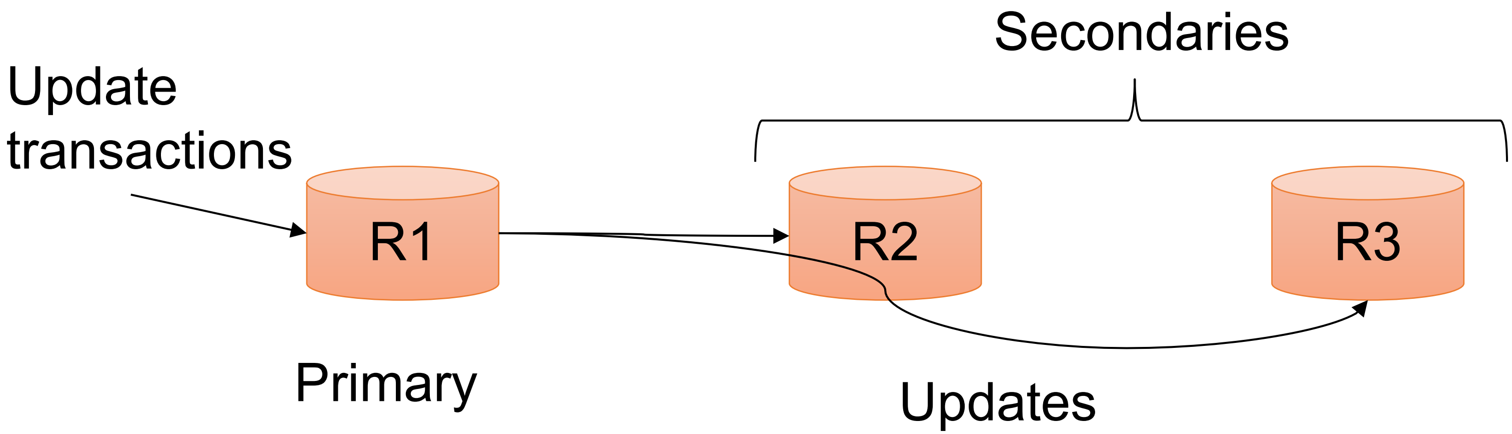
- Also called eager replication
- All updates are applied to all replicas (or to a majority) as part of a single transaction (need two phase commit)
- Transactions must acquire global locks
  - Nobody can read while we synchronize the replicas
- Main goal: as if there was only one copy
  - Maintain **consistency**
  - Maintain **one-copy serializability**
  - I.e., execution of transactions has same effect as an execution on a non-replicated db

Synchronous Replication



Synchronous Master Replication

- One master for each object holds primary copy
  - The "Master" is also called "Primary"
  - To update object, transaction must acquire a lock at the master
  - Lock at the master is global lock
- Master propagates updates to replicas synchronously
  - Updates propagate as part of the same distributed transaction
  - Need to run 2PC at the end



Crash Failures

- What happens when a secondary crashes?

Crash Failures

- What happens when a secondary crashes?
  - Nothing happens
  - When secondary recovers, it catches up

Crash Failures

- What happens when a secondary crashes?
  - Nothing happens
  - When secondary recovers, it catches up
- What happens when the master/primary fails?

Crash Failures

- What happens when a secondary crashes?
  - Nothing happens
  - When secondary recovers, it catches up
- What happens when the master/primary fails?
  - Blocking would hurt availability
  - Must chose a new primary: run election

Network Failures

- Network failures can cause trouble...
  - Secondaries think that primary failed
  - Secondaries elect a new primary
  - But primary can still be running
  - Now have two primaries!

Majority Consensus

- To avoid problem, only majority partition can continue processing at any time

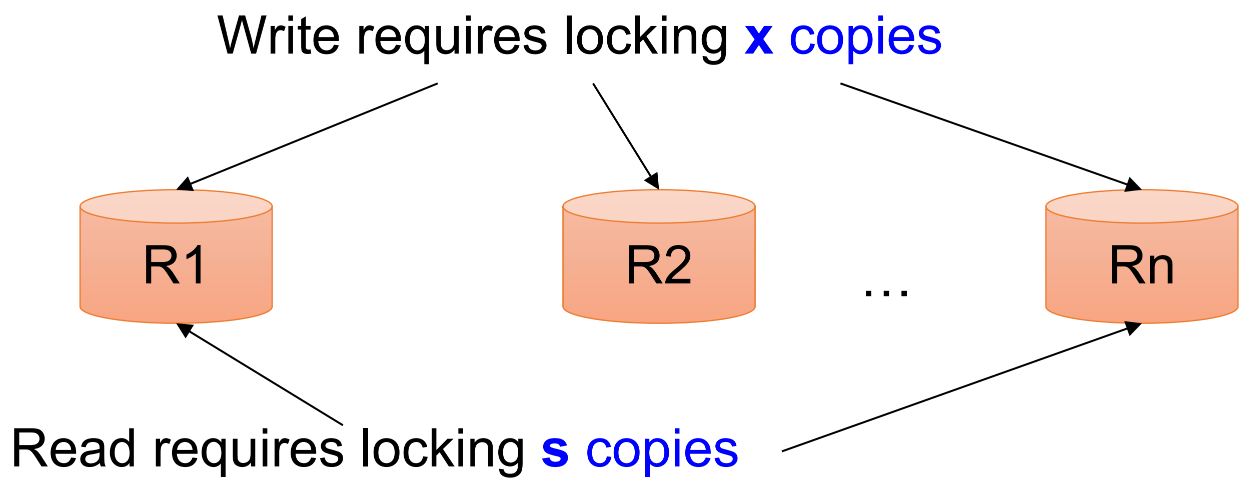
- In general,
  - Whenever a replica fails or recovers...
  - ...a set of communicating replicas must determine...
  - ...whether they have a majority before they can continue

Types of Replication

	Master	Group
Synchronous	✓	✓
Asynchronous		

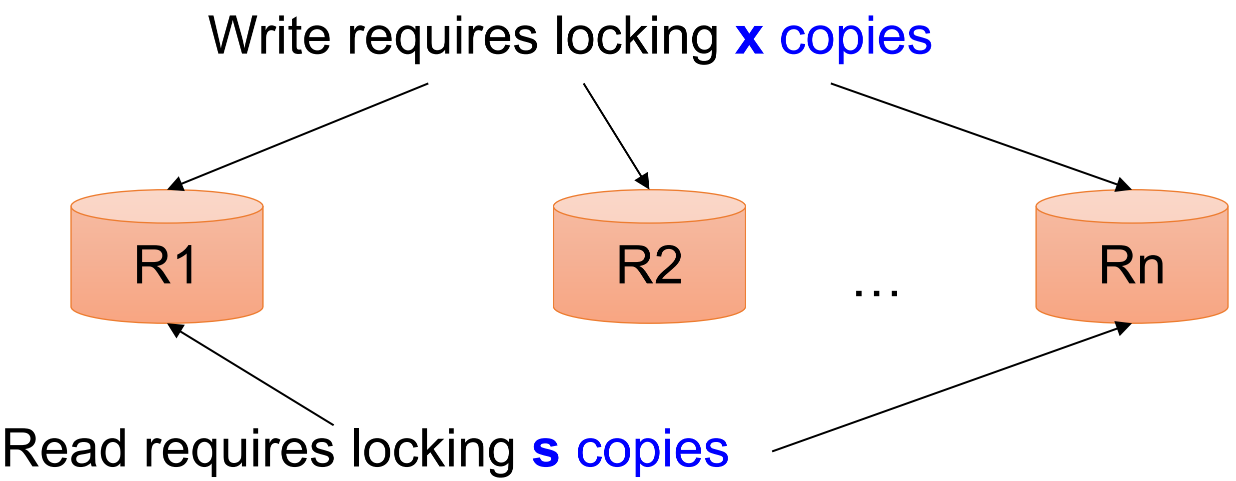
Synchronous Group Replication

- **Master-less**
  - Any node can initiate a transaction!
  - Need to gather a number of nodes that agree on a particular transaction
  - Each copy has its own lock



Synchronous Group Replication

- With n copies
  - Exclusive lock on x copies is global exclusive lock
  - Shared lock on s copies is global shared lock
  - Must have: $2x > n$ and $s + x > n$
  - Version numbers serve to identify current copy



Synchronous Group Replication

- Majority locking
  - $s = x = \lceil (n+1)/2 \rceil$ eg: 11 nodes: need 6 locked
  - Usually not attractive because reads are slowed down

- Read-locks-one, write-locks-all
  - s=1 and x = n, high read performance
  - Reads are very fast

Synchronous Replication Properties

- Favours consistency over availability
  - Only majority partition can process requests
  - There appears to be a single copy of the db
- High runtime overhead
  - Must lock and update at least majority of replicas
  - Two-phase commit
  - Runs at pace of slowest replica in quorum
  - So overall system is now slower
  - Higher deadlock rate (transactions take longer)

Types of Replication

	Master	Group
Synchronous	✓	✓
Asynchronous	✓	

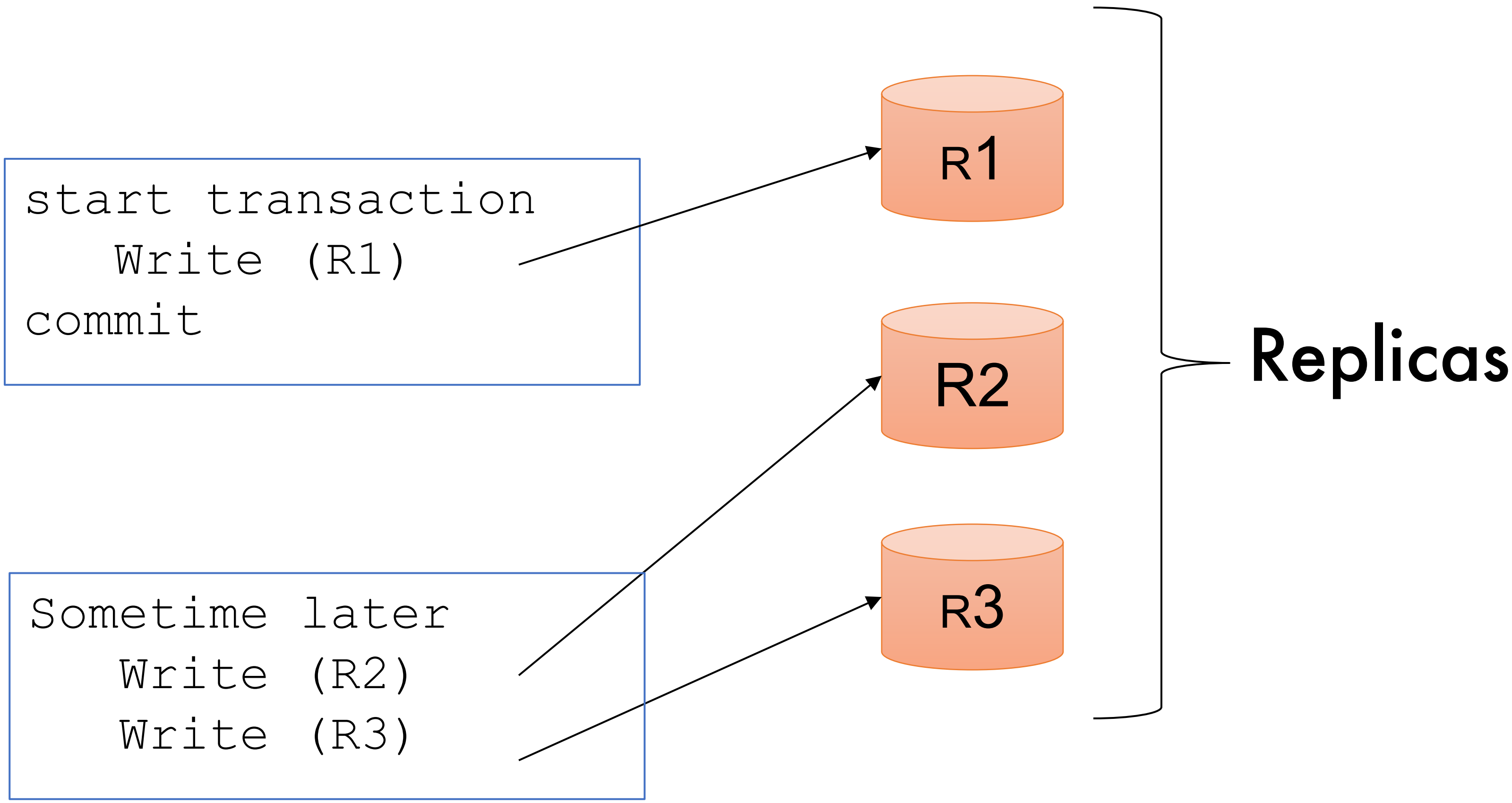
Asynchronous Replication

- Also called lazy replication
- Also called optimistic replication

- Main goals: availability and performance

- Approach
  - One replica updated by original transaction
  - Updates propagate asynchronously to other replicas

Asynchronous Replication



Asynchronous Master Replication

One master holds primary copy

- Transactions update primary copy
- Master asynchronously propagates updates to replicas, which process them in same order
  E.g. through log shipping
- Ensures single-copy serializability

What happens when master/primary fails?

- Can lose most recent transactions when primary fails!
- After electing a new primary, secondaries must agree who is most up-to-date

Discussion: Log Shipping

A general problem:

- A master operates on a database
- The DB needs to be replicated to one or several replicas (e.g. hot stand-by databases)

Discussion: Log Shipping

A general problem:

- A master operates on a database
- The DB needs to be replicated to one or several replicas (e.g. hot stand-by databases)
- Log Shipping Technique

Discussion: Log Shipping

A general problem:

- A master operates on a database
- The DB needs to be replicated to one or several replicas (e.g. hot stand-by databases)
- Log Shipping Technique:
  - Master node ships the tail of the log to the replicas
    E.g. when it flushes the log tail to disk
  - Replicas REDO the log; this is very efficient
  - Need very little systems development: we create the log anyway, and we have the REDO function anyway

Discussion: Log Shipping

A general problem:

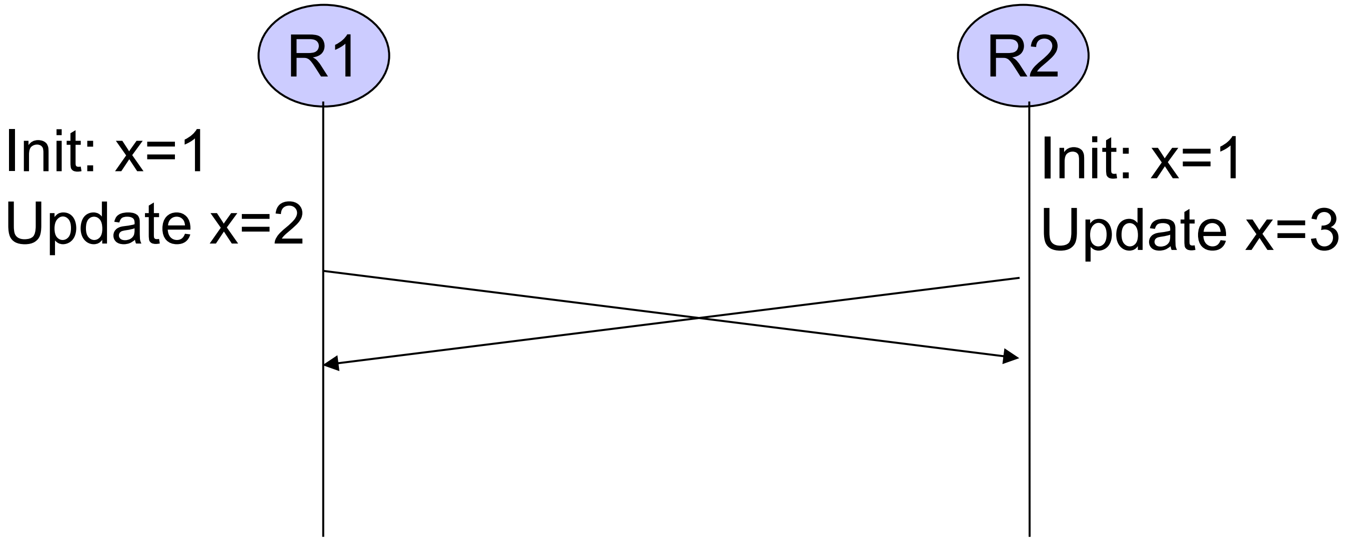
- A master operates on a database
- The DB needs to be replicated to one or several replicas (e.g. hot stand-by databases)
- Log Shipping Technique:
  - Master node ships the tail of the log to the replicas E.g. when it flushes the log tail to disk
  - Replicas REDO the log; this is very efficient
  - Need very little systems development: we create the log anyway, and we have the REDO function anyway
  - Complications due to the need to "remove" updates of active transactions (they may later abort)

Types of Replication

	Master	Group
Synchronous	✓	✓
Asynchronous	✓	✓

Asynchronous Group Replication

- Also called multi-master
- Best scheme for availability
- Cannot guarantee one-copy serializability!



Asynchronous Group Replication

- Cannot guarantee one-copy serializability!
- Instead guarantee convergence
  - Db state does not reflect any serial execution
  - But all replicas have the same state
- Called "Eventual Consistency" = if the DB stops operations, then eventually all copies are equal

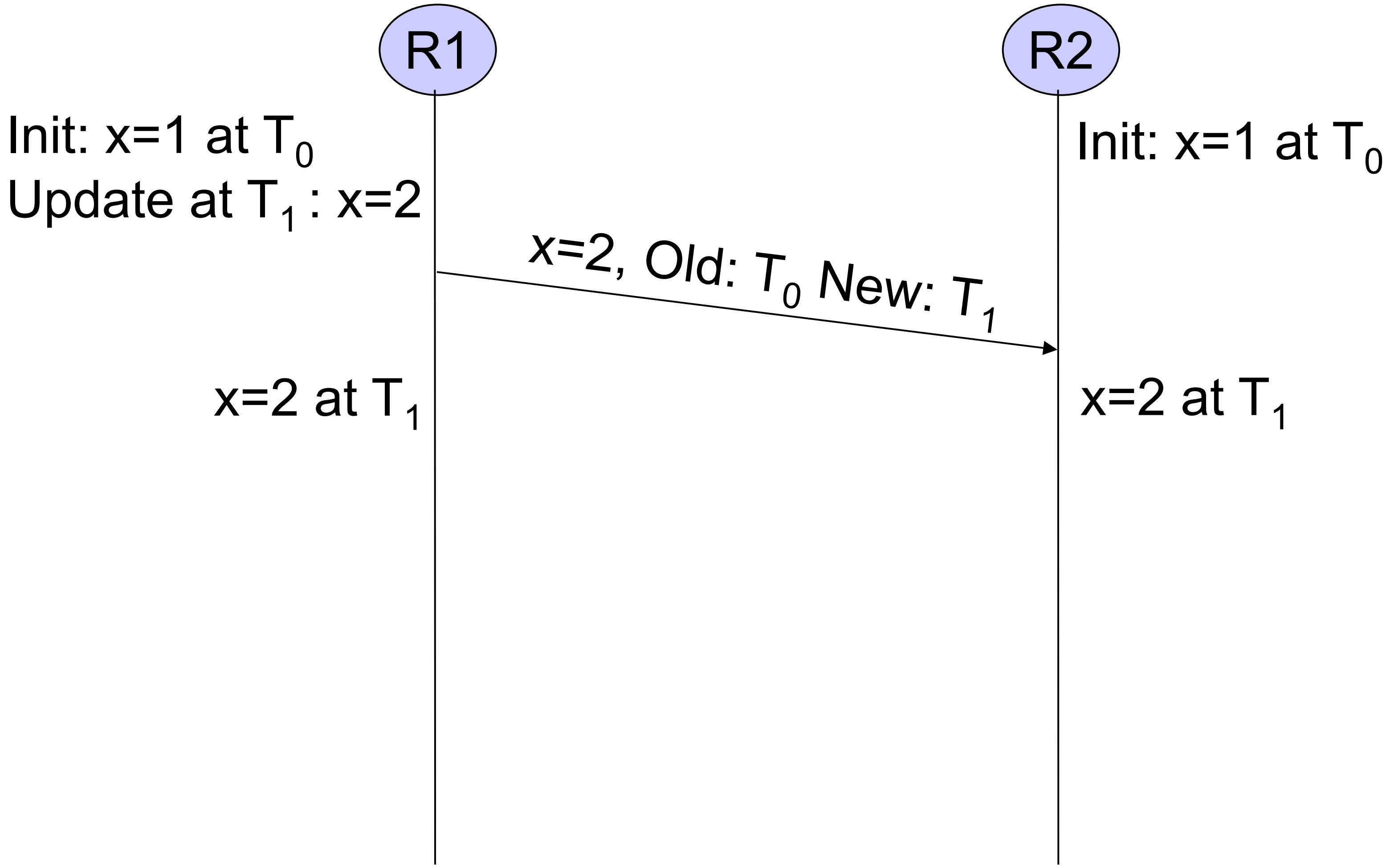
Asynchronous Group Replication

- Cannot guarantee one-copy serializability!
- Instead guarantee convergence
  - Db state does not reflect any serial execution
  - But all replicas have the same state
- Called "Eventual Consistency" = if the DB stops operations, then eventually all copies are equal
- Detect conflicts and reconcile replica states

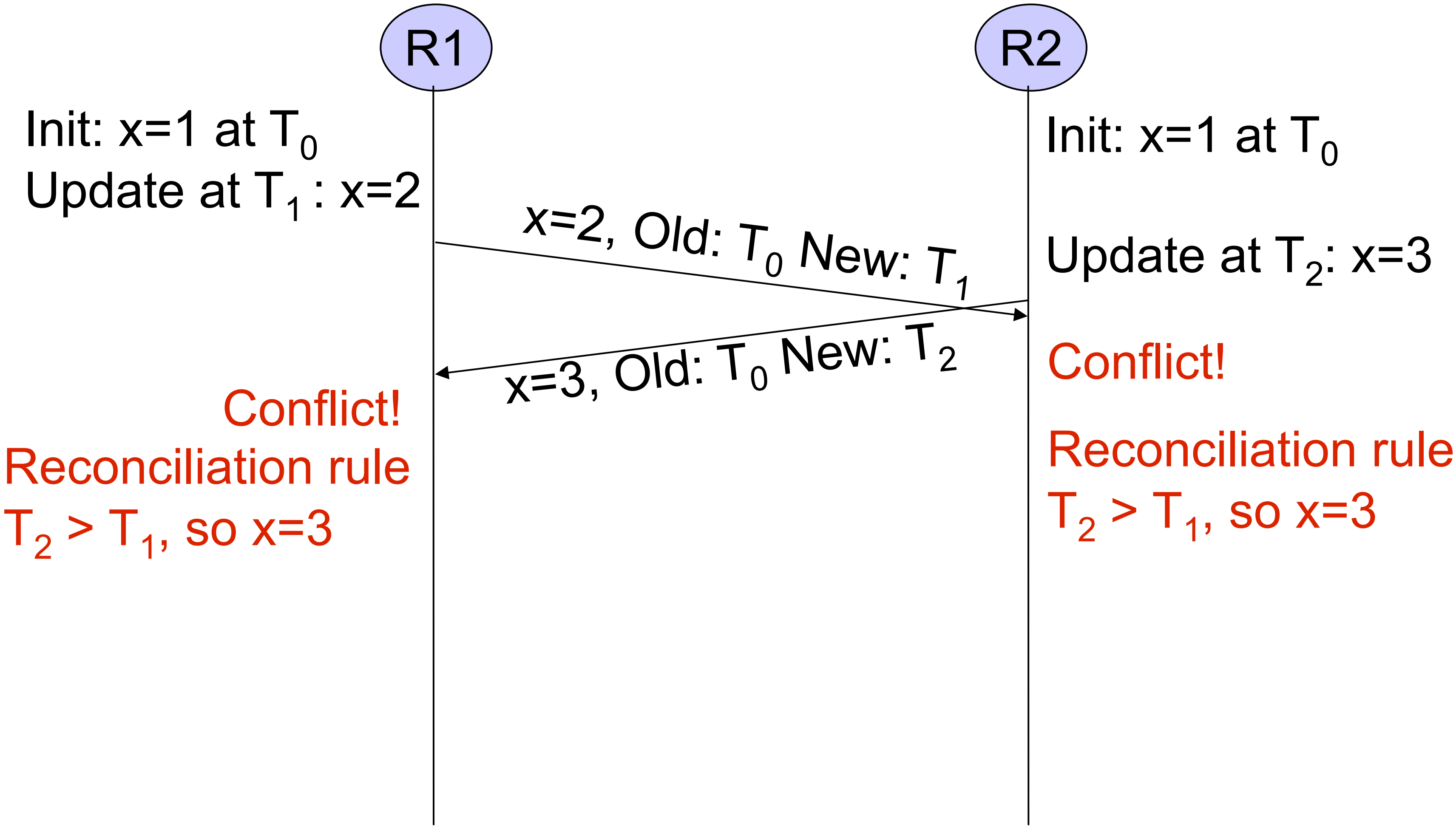
Asynchronous Group Replication

- Cannot guarantee one-copy serializability!
- Instead guarantee convergence
  - Db state does not reflect any serial execution
  - But all replicas have the same state
- Called "Eventual Consistency" = if the DB stops operations, then eventually all copies are equal
- Detect conflicts and reconcile replica states
- Reconciliation techniques:
  - Most recent timestamp wins
  - Site A wins over site B
  - But also: user-defined rules, or even manual

Detecting Conflicts Using Timestamps



Detecting Conflicts Using Timestamps



Vector Clocks

- An extension of Multiversion Concurrency Control (MVCC) to multiple servers

- Standard MVCC:
  each data item X has a timestamp t:
  $X_4, X_9, X_{10}, X_{14}, \ldots, X_t$

- Vector Clocks:
  X has set of [server, timestamp] pairs
  X([s1,t1], [s2,t2],...)

Vector Clocks

Dynamo:2007

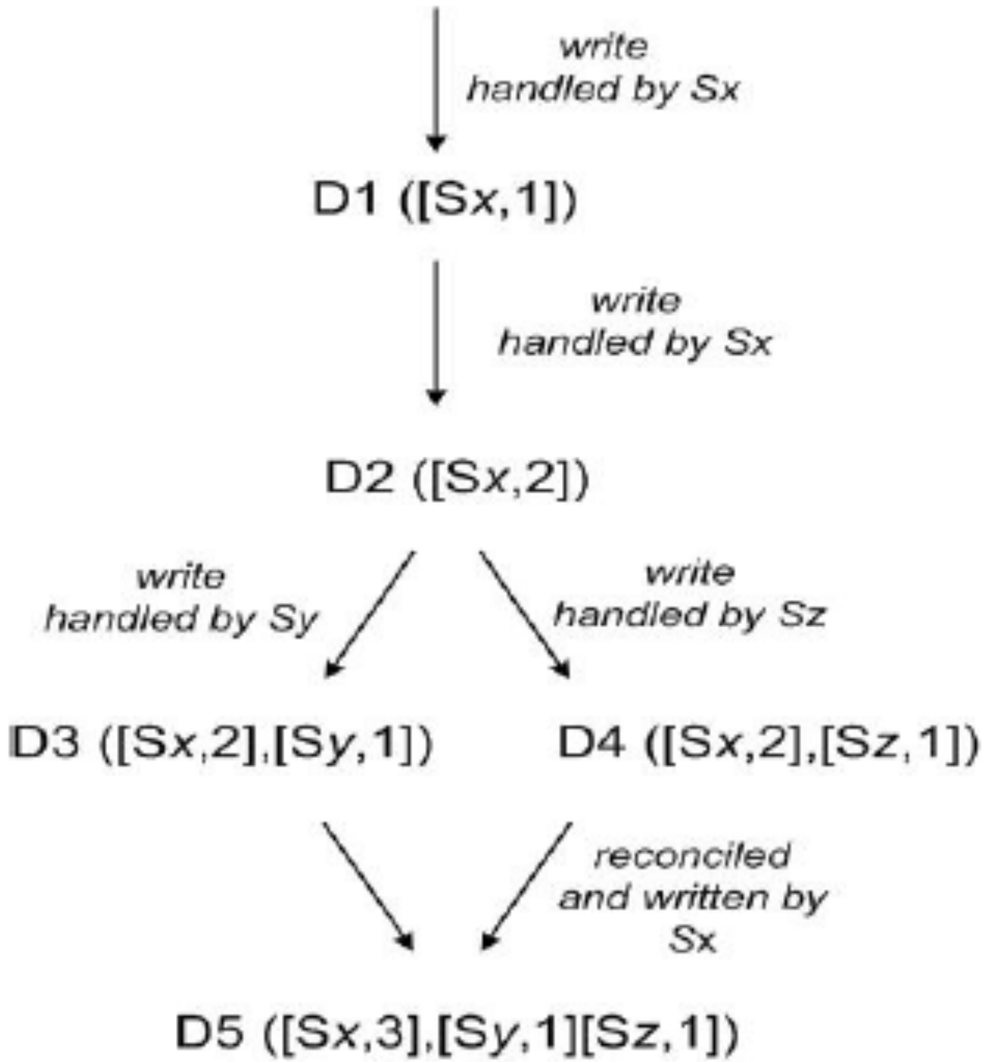


Figure 3: Version evolution of an object over time.

Basic Operations

TXN Reads an element X:

- Request is handled by a site s_i…
- …which returns X and its vector clock: $VC = [s_1,t_1],[s_2,t_2],\ldots,[s_n,t_n]$

Basic Operations

TXN Reads an element X:

- Request is handled by a site s_i...
- ...which returns X and its vector clock:
  VC = $[s_1,t_1],[s_2,t_2],\ldots,[s_n,t_n]$

Process and update it locally X = ...[something]...

Basic Operations

TXN Reads an element X:

- Request is handled by a site s_i...
- ...which returns X and its vector clock: $VC = [s_1,t_1],[s_2,t_2],\ldots,[s_n,t_n]$

Process and update it locally X = ...[something]...

TXN Writes the element X:

- Request is handled by a site s'

Basic Operations

TXN Reads an element X:

- Request is handled by a site s_i...
- ...which returns X and its vector clock: $VC = [s_1,t_1],[s_2,t_2],...,[s_n,t_n]$

Process and update it locally X = ...[something]...

TXN Writes the element X:

- Request is handled by a site s'
- If s' already has X with vector clock VC', then first reconcile VC and VC'

Basic Operations

TXN Reads an element X:

- Request is handled by a site s_i...
- ...which returns X and its vector clock:
  VC = $[s_1,t_1],[s_2,t_2],\ldots,[s_n,t_n]$

Process and update it locally X = ...[something]...

TXN Writes the element X:

- Request is handled by a site s'
- If s' already has X with vector clock VC',
  then first reconcile VC and VC'
- If s' is not in VC, then add [s,1] to VC

Basic Operations

TXN Reads an element X:

- Request is handled by a site s_i...
- ...which returns X and its vector clock:
  VC = $[s_1,t_1],[s_2,t_2],\ldots,[s_n,t_n]$

Process and update it locally X = ...[something]...

TXN Writes the element X:

- Request is handled by a site s'
- If s' already has X with vector clock VC', then first reconcile VC and VC'
- If s' is not in VC, then add [s,1] to VC
- If [s',t] is in VC, then replace with [s',t+1]

Conflicts and Reconciliation

A site has two version of X

- X1 with vector clock VC1 and
- X2 with vector clock VC2

If VC1, VC2 have a conflict, then use application specific reconciliation to compute (X,VC)

If there is no conflict, then:

- X = latest of X1, X2
- VC = VC1 ∪ VC2

Vector Clocks: Conflict or not?

Reconcile (X1,VC1), (X2,VC2) to get (X,VC)

VC1	VC2	Conflict ?	X	VC
$[S_1,3]$	$[S_1,4]$			

Vector Clocks: Conflict or not?

Reconcile (X1,VC1), (X2,VC2) to get (X,VC)

VC1	VC2	Conflict ?	X	VC
[S_1,3]	[S_1,4]	No	X2	[S_1,4]

Vector Clocks: Conflict or not?

Reconcile (X1,VC1), (X2,VC2) to get (X,VC)

VC1	VC2	Conflict ?	X	VC
$[S_1,3]$	$[S_1,4]$	No	X2	$[S_1,4]$
$[S_1,3],[S_2,6]$	$[S_1,4],[S_3,2]$			

Vector Clocks: Conflict or not?

Reconcile (X1,VC1), (X2,VC2) to get (X,VC)

VC1	VC2	Conflict ?	X	VC
[S_1,3]	[S_1,4]	No	X2	[S_1,4]
[S_1,3],[S_2,6]	[S_1,4],[S_3,2]	Yes	-	-

Vector Clocks: Conflict or not?

Reconcile (X1,VC1), (X2,VC2) to get (X,VC)

VC1	VC2	Conflict ?	X	VC
[S_1,3]	[S_1,4]	No	X2	[S_1,4]
[S_1,3],[S_2,6]	[S_1,4],[S_3,2]	Yes	-	-
[S_1,3],[S_2,6]	[S_1,4],[S_2,6],[S_3,2]			

Vector Clocks: Conflict or not?

Reconcile (X1,VC1), (X2,VC2) to get (X,VC)

VC1	VC2	Conflict ?	X	VC
[S_1,3]	[S_1,4]	No	X2	[S_1,4]
[S_1,3],[S_2,6]	[S_1,4],[S_3,2]	Yes	-	-
[S_1,3],[S_2,6]	[S_1,4],[S_2,6],[S_3,2]	No	X2	[S_1,4],[S_2,6],[S_3,2]

Vector Clocks: Conflict or not?

Reconcile (X1,VC1), (X2,VC2) to get (X,VC)

VC1	VC2	Conflict ?	X	VC
[S_1,3]	[S_1,4]	No	X2	[S_1,4]
[S_1,3],[S_2,6]	[S_1,4],[S_3,2]	Yes	-	-
[S_1,3],[S_2,6]	[S_1,4],[S_2,6],[S_3,2]	No	X2	[S_1,4],[S_2,6],[S_3,2]
[S_1,3],[S_2,10]	[S_1,4],[S_2,6],[S_3,2]			

Vector Clocks: Conflict or not?

Reconcile (X1,VC1), (X2,VC2) to get (X,VC)

VC1	VC2	Conflict ?	X	VC
[S_1,3]	[S_1,4]	No	X2	[S_1,4]
[S_1,3],[S_2,6]	[S_1,4],[S_3,2]	Yes	-	-
[S_1,3],[S_2,6]	[S_1,4],[S_2,6],[S_3,2]	No	X2	[S_1,4],[S_2,6],[S_3,2]
[S_1,3],[S_2,10]	[S_1,4],[S_2,6],[S_3,2]	Yes	-	-

Vector Clocks: Conflict or not?

Reconcile (X1,VC1), (X2,VC2) to get (X,VC)

VC1	VC2	Conflict ?	X	VC
$[S_1,3]$	$[S_1,4]$	No	X2	$[S_1,4]$
$[S_1,3],[S_2,6]$	$[S_1,4],[S_3,2]$	Yes	-	-
$[S_1,3],[S_2,6]$	$[S_1,4],[S_2,6],[S_3,2]$	No	X2	$[S_1,4],[S_2,6],[S_3,2]$
$[S_1,3],[S_2,10]$	$[S_1,4],[S_2,6],[S_3,2]$	Yes	-	-
$[S_1,3],[S_2,10]$	$[S_1,4],[S_2,20],[S_3,2]$			

Vector Clocks: Conflict or not?

Reconcile (X1,VC1), (X2,VC2) to get (X,VC)

VC1	VC2	Conflict ?	X	VC
$[S_1,3]$	$[S_1,4]$	No	X2	$[S_1,4]$
$[S_1,3],[S_2,6]$	$[S_1,4],[S_3,2]$	Yes	-	-
$[S_1,3],[S_2,6]$	$[S_1,4],[S_2,6],[S_3,2]$	No	X2	$[S_1,4],[S_2,6],[S_3,2]$
$[S_1,3],[S_2,10]$	$[S_1,4],[S_2,6],[S_3,2]$	Yes	-	-
$[S_1,3],[S_2,10]$	$[S_1,4],[S_2,20],[S_3,2]$	No	X2	$[S_1,4],[S_2,20],[S_3,2]$

Vector Clocks: Conflict or not?

General rule:

- VC1 precedes VC2 if for all [s,t] in VC1 there exists [s,t'] in VC2 with $t \leq t'$
- VC2 precedes VC2 if ... [symmetric rule]
- Otherwise, VC1 and VC2 are in conflict

Asynchronous Group Replication Properties

- Favours availability over consistency
  - Can read and update any replica
  - High runtime performance
- Weak consistency
  - Conflicts and reconciliation

Outline

- Goals of replication

- Three types of replication
  - Synchronous (aka eager) replication
  - Asynchronous (aka lazy) replication
  - Two-tier replication

Two-Tier Replication

- Benefits of lazy master and lazy group
- Each object has a master with primary copy
- When disconnected from master
  - Secondary can only run tentative transactions
- When reconnects to master
  - Master reprocesses all tentative transactions
  - Checks an acceptance criterion
  - If passes, we now have final commit order
  - Secondary undoes tentative and redoes committed

Conclusion

- Replication is a very important problem
  - Fault-tolerance (various forms of replication)
  - Caching (lazy master)
  - Warehousing (lazy master)
  - Mobility (two-tier techniques)
- Replication is complex, but basic techniques and trade-offs are **very well known**
  - Synchronous or asynchronous replication
  - Master or quorum

Slide from Andy Pavlo @ CMU

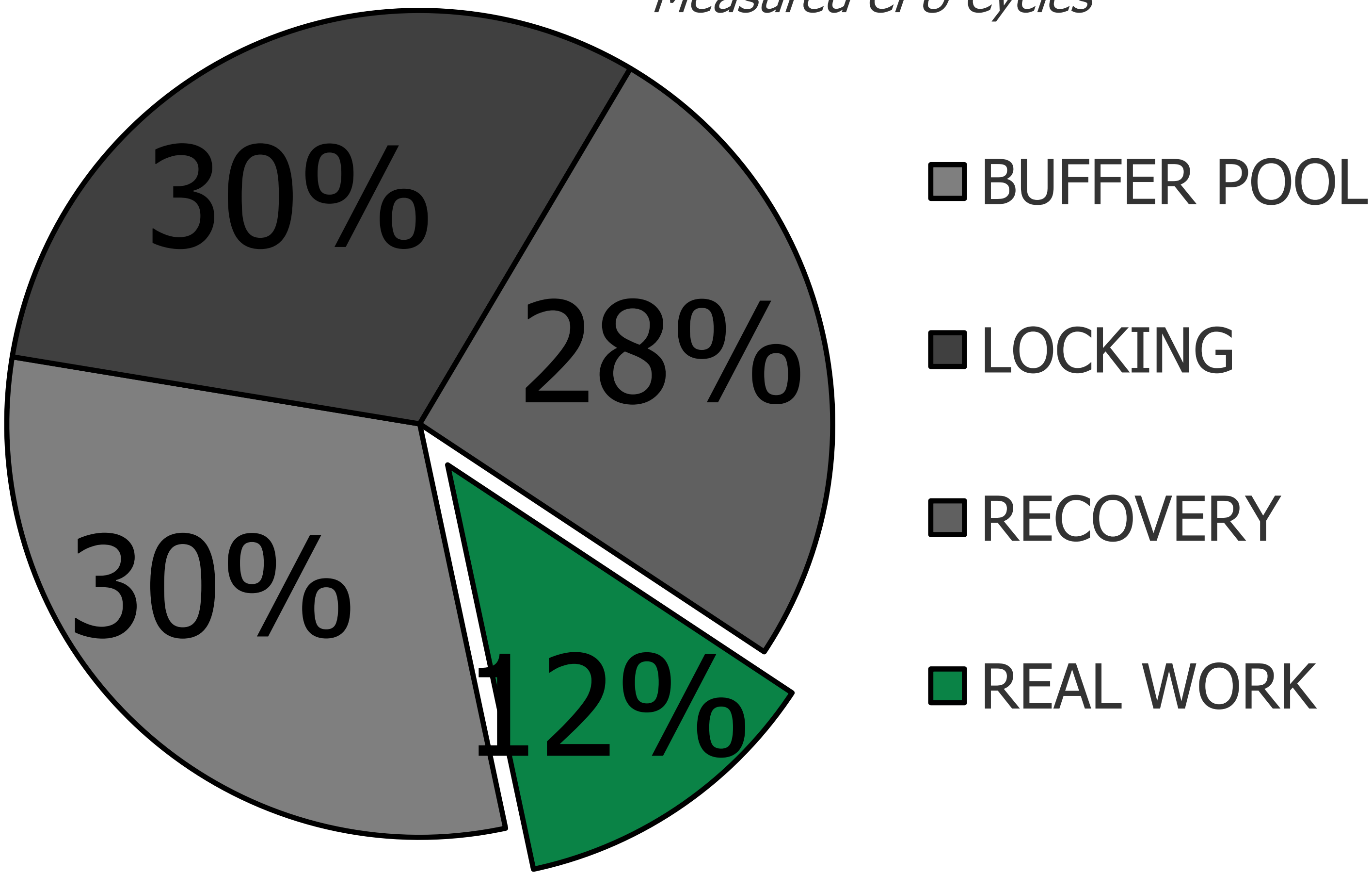
Some Popular NewSQL Systems

- H-Store
  - Research system from Brown U., MIT, CMU, and Yale
  - Commercialized as VoltDB
- Hekaton
  - Microsoft
  - Fully integrated into SQL Server
- Hyper
  - Hybrid OLTP/OLAP
  - Research system from TU Munich. Bought by Tableau
- Spanner
  - Google

H-Store Insight

TRADITIONAL DBMS:

Measured CPU Cycles



OLTP THROUGH THE LOOKING GLASS,
AND WHAT WE FOUND THERE
SIGMOD, pp. 981-992, 2008.

Slide from Andy Pavlo @ CMU

H-Store Key Ideas

- Main-memory storage
  - Avoids disk IO costs / buffer pool costs
  - Durability through snapshots + cmd log
  - Replication
- Serial execution
  - One database partition per thread on one core
  - Avoid overheads related to locking
- All transactions are stored procedures
  - Command logging avoids heavy recovery overheads
- Avoid distributed transactions
  - But when needed, run 2PC

STORED PROCEDURE

VoteCount:

```
SELECT COUNT(*)
 FROM votes
WHERE phone_num = ?;
```

InsertVote:

```
INSERT INTO votes
 VALUES (?, ?, ?);
```

```
run(phoneNum, contestantId, currentTime) {
      result = execute(VoteCount, phoneNum);
      if (result > MAX_VOTES) {
            return (ERROR);
      }
      execute(InsertVote, phoneNum,
                        contestantId,
                        currentTime);
      return (SUCCESS);
}
```

Slide from Andy Pavlo @ CMU

Some Details

At one node:

- Data is partitioned
- One database partition per thread on one core
- TXN receives a time stamp TS = serialization order
- TXN is assigned to a "base partition"; if data is need for other partitions, it sends requests there
- Partition managers order the requests based on TS. If conflict: abort, then restart (since stored procedure) with larger TS
- When a TXN has been granted locks at all partitions that it needs, then it can execute
- If more partitions are needed, then abort/restart

Some Details

Stored procedure

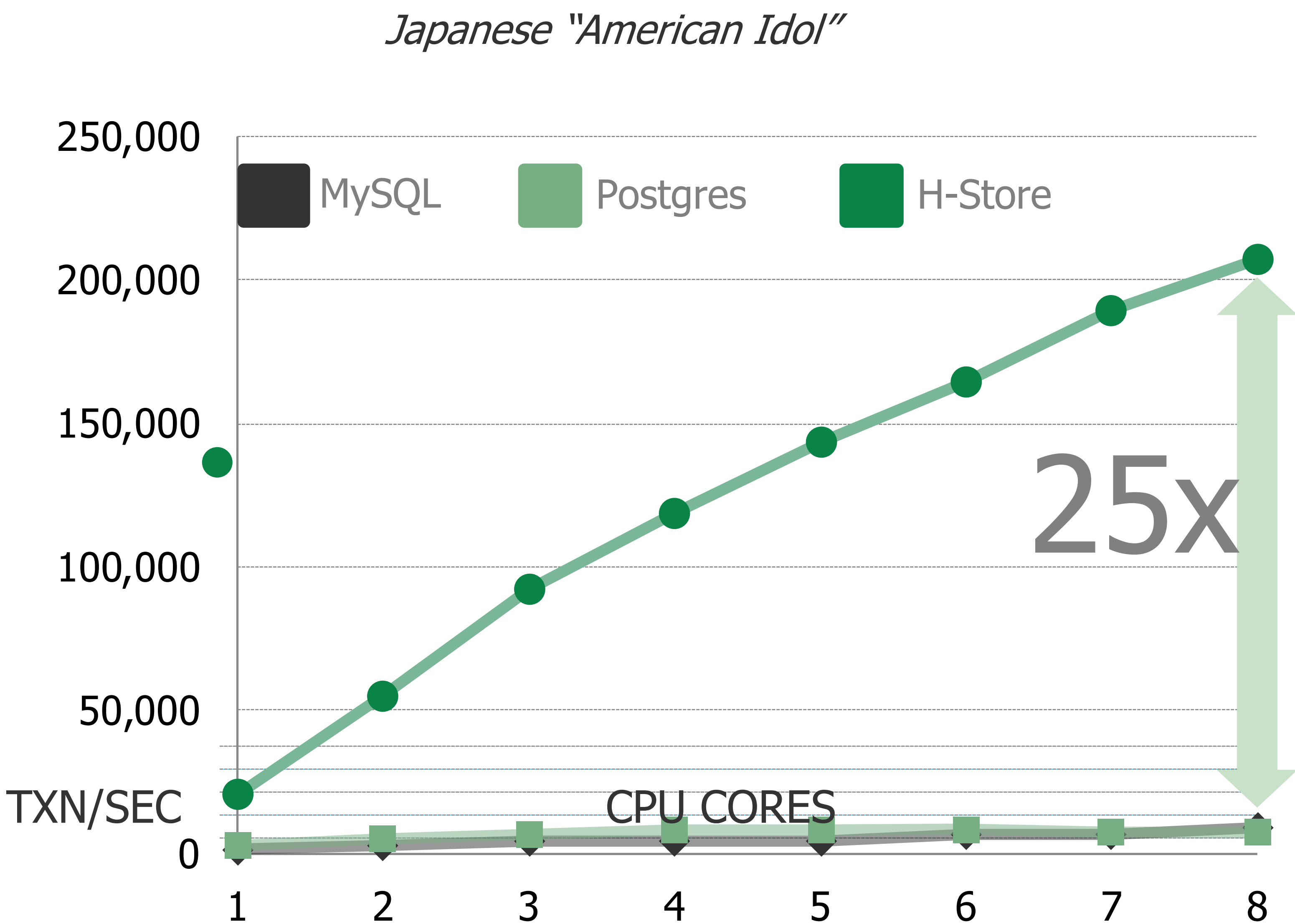
- TXN = One stored procedure
- Arbitrary Java code, BUT must be deterministic! No: call to the systems clock, random number generators, messages to other threads
- Have several parameterized queries, i.e. with '?'
- Several invocations of these queries are collected in a *batch*, then sent to the engine for execution
- If the batch requests data from a partition where the TXN does not have the lock: ABORT/RESTART
- Commit across multiple partitions: 2PC
- Command log: write just the procedure name plus parameters; only for committed TXN

Some Details

Replication

- Recovery is slow → H-Store uses replication
- Initially, run Paxos to choose a master node
- During normal operation: TXN's are executed on the master node, who sends identical commands to the replica nodes; results are checked, and validated if majority, otherwise abort; minority nodes are considered failed
- When the master fails, run Paxos to elect new master.

Voter Benchmark



Slide from Andy Pavlo @ CMU

Hekaton

- Focus: DBMS with large main memories and many core CPUs

- Integrated with SQL Server

- Key user-visible features
  - Simply declare a table "memory resident"
  - Hekaton tables are fully durable and transactional, though non-durable tables are also supported
  - Query can touch both Hekaton and regular tables

Hekaton Key Details

- Idea: To increase transaction throughput must decrease number of instructions / transaction
- Main-memory DBMS
  - Optimize indexes for memory-resident data
  - Durability by logging and checkpointing records to external storage
- No partitioning
  - Any thread can touch any row of any table
- No locking
  - Uses a new MVCC method for isolation

Hekaton More Details

- Optimized stored procedures
  - Compile statements and stored procedures into customized, highly efficient machine code

Hyper

- Hybrid OLTP and OLAP
- In-memory data management
  - Including optimized indexes for memory-resident data
  - Data compression for cold data
- Data-centric code generation
  - SQL translated to LLVM
- OLAP separated from OLTP using MVCC
- Exploits hardware transactional memory
- Data shuffling and distribution optimizations

Conclusion

- Many innovations recently in
  - Big data analytics
  - Transaction processing at very large scale
- Many more problems remain open
- This course teaches foundations
- Innovate with an open mind!