DATABASE REPLICATION

A TALE OF RESEARCH ACROSS COMMUNITIES

Bettina Kemme
Dept. of Computer Science
McGill University
Montreal, Canada

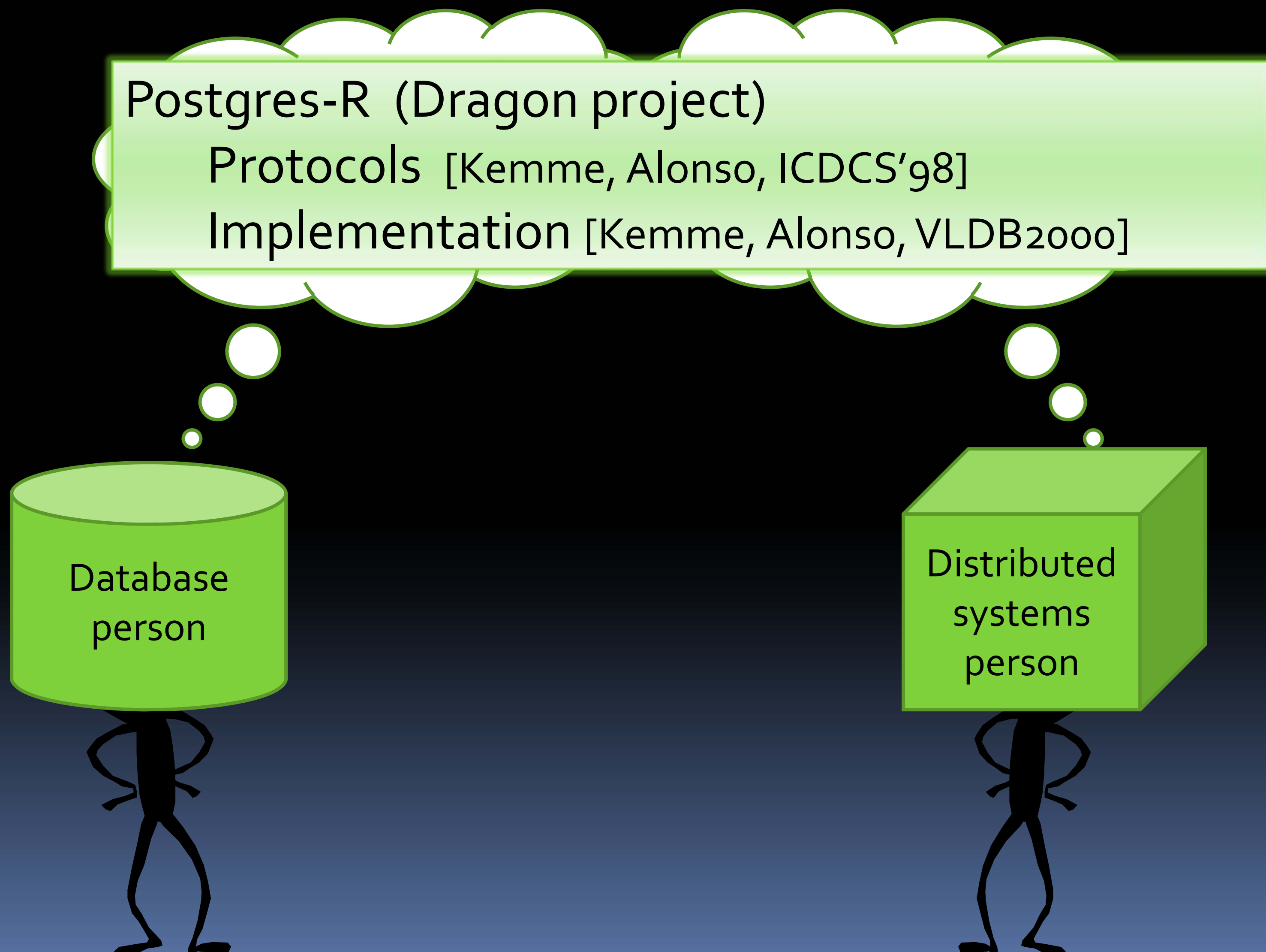
McGill
School of Computer Science

Gustavo Alonso
Systems Group
Dept. of Computer Science
ETH Zurich, Switzerland

Systems@ETH Zürich

VLDB 2010 - Singapore

Across communities



Postgres-R

- Intro to Replication
- Postgres-R
- In perspective
- Systems Today
- The next 10 years

Don't be lazy, be consistent: Postgres-R, A new way to implement Database Replication *

Bettina Kemme Gustavo Alonso
Information and Communication Systems Group, ETH Zürich, Switzerland
{kemme,alonso}@inf.ethz.ch

Abstract

Database designers often point out that eager, update everywhere replication suffers from high deadlock rates, message overhead and poor response times. In this paper, we show that these limitations can be circumvented by using a combination of known and novel techniques. Moreover, we show how the proposed solution can be incorporated into a real database system. The paper discusses the new protocols and their implementation in PostgreSQL. It also provides experimental results proving that many of the dangers and limitations of replication can be avoided by using the appropriate techniques.

1 Introduction

Existing replication protocols can be divided into eager and lazy schemes [GHOS96]. Eager protocols ensure that changes to copies happen within the transaction boundaries. That is, when a transaction commits, all copies have the same value. Lazy replication protocols propagate changes only after the transaction commits, thereby allowing copies to have different values. While eager replication emphasizes consistency, lazy replication pays more attention to efficiency. Among database designers, there is the widespread belief that eager replication is not practical. The "dangers" of eager replication have been analyzed by Gray et al. [GHOS96] and, since the publication of those results, the research focus has shifted towards lazy replication [CRR96, PMS99, ABKW98, BKR+99]. The drawback of lazy replication is that, if consistency is necessary, many non trivial problems arise. Namely, in the case of update everywhere (each copy can be updated), maintaining consistency is usually left to the user. If only a primary copy can be updated, consistency is achieved at the price of introducing a bottleneck and a single point of failure. In addition, recent results prove that consistency can only be guaranteed when the system configuration is severely restricted.

We see these as serious limitations. The thesis defended in this paper is that if the goal is to achieve consistency and the computing environment allows it, then eager replication should be used. In spite of what is commonly assumed, eager replication is perfectly feasible in many environments. A good example are the computer clusters which can be found behind many Internet sites. However, in order to circumvent the limitations of traditional solutions, it is necessary to rethink the way transaction and replica management is done. In this paper, we demonstrate how eager replication can be implemented in practice. Some of the techniques we use include executing the transaction first locally on shadow copies and postponing the propagation of updates to the end of the transaction, using group communication primitives for preordering transactions, and acquiring all locks a transaction needs in an atomic step. To prove the feasibility of these ideas, we have implemented them in Postgres-R, an extension of PostgreSQL [Pos98] and tested them extensively. The results prove that eager replication is feasible in clusters of computers and can scale to a relatively large number of nodes.

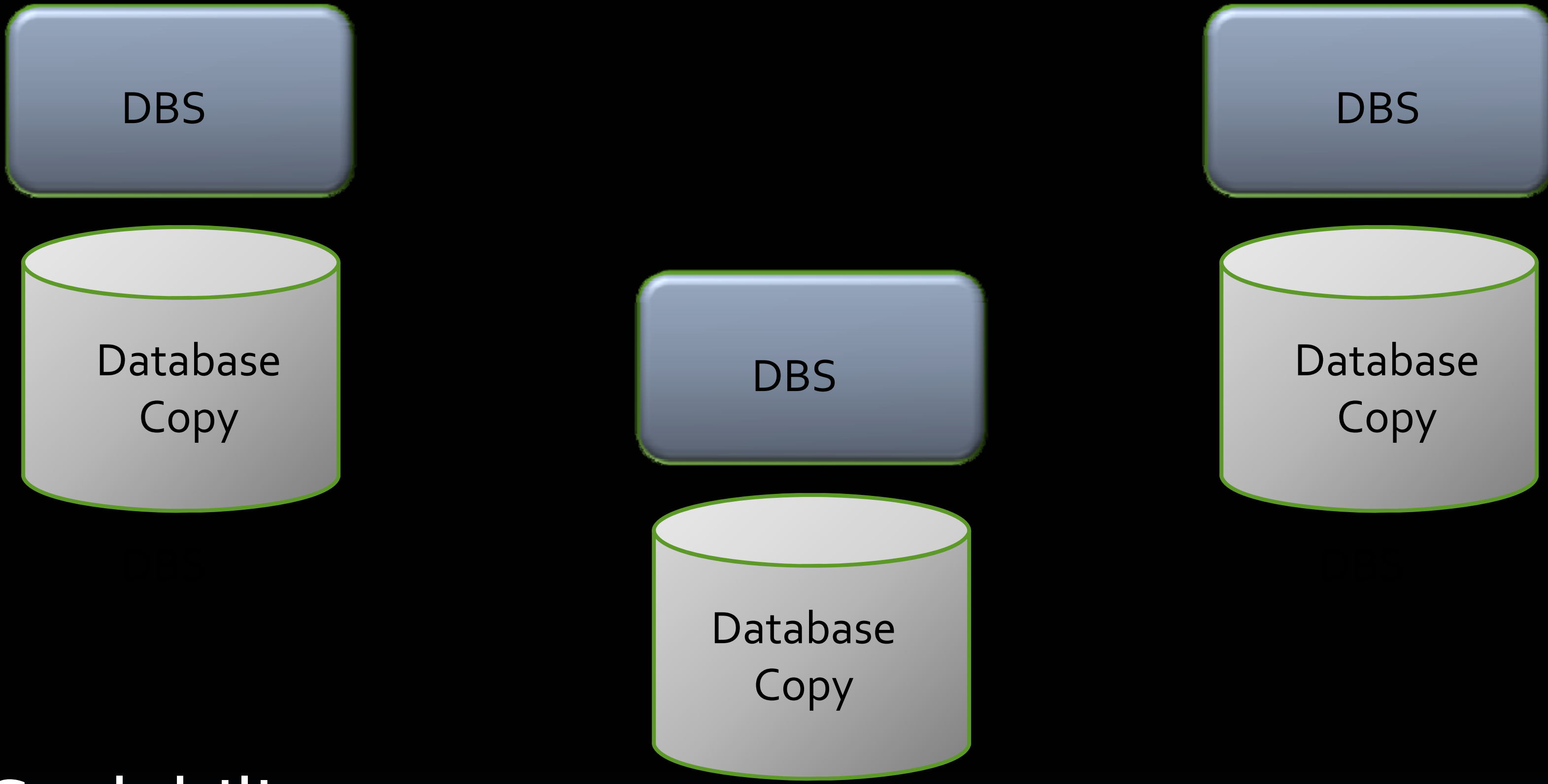
The paper is organized as follows. Section 2 discusses related work. Section 3 explains the principle techniques of our approach and presents a basic protocol. Section 4 discusses the architecture and implementation of Postgres-R. Section 5 presents performance results. Section 6 discusses configuration management and partial replication. Section 7 concludes the paper.

*Part of this work has been funded by ETH Zürich within the DRAGON Research Project (Reg-Nr. 41-2642.5)

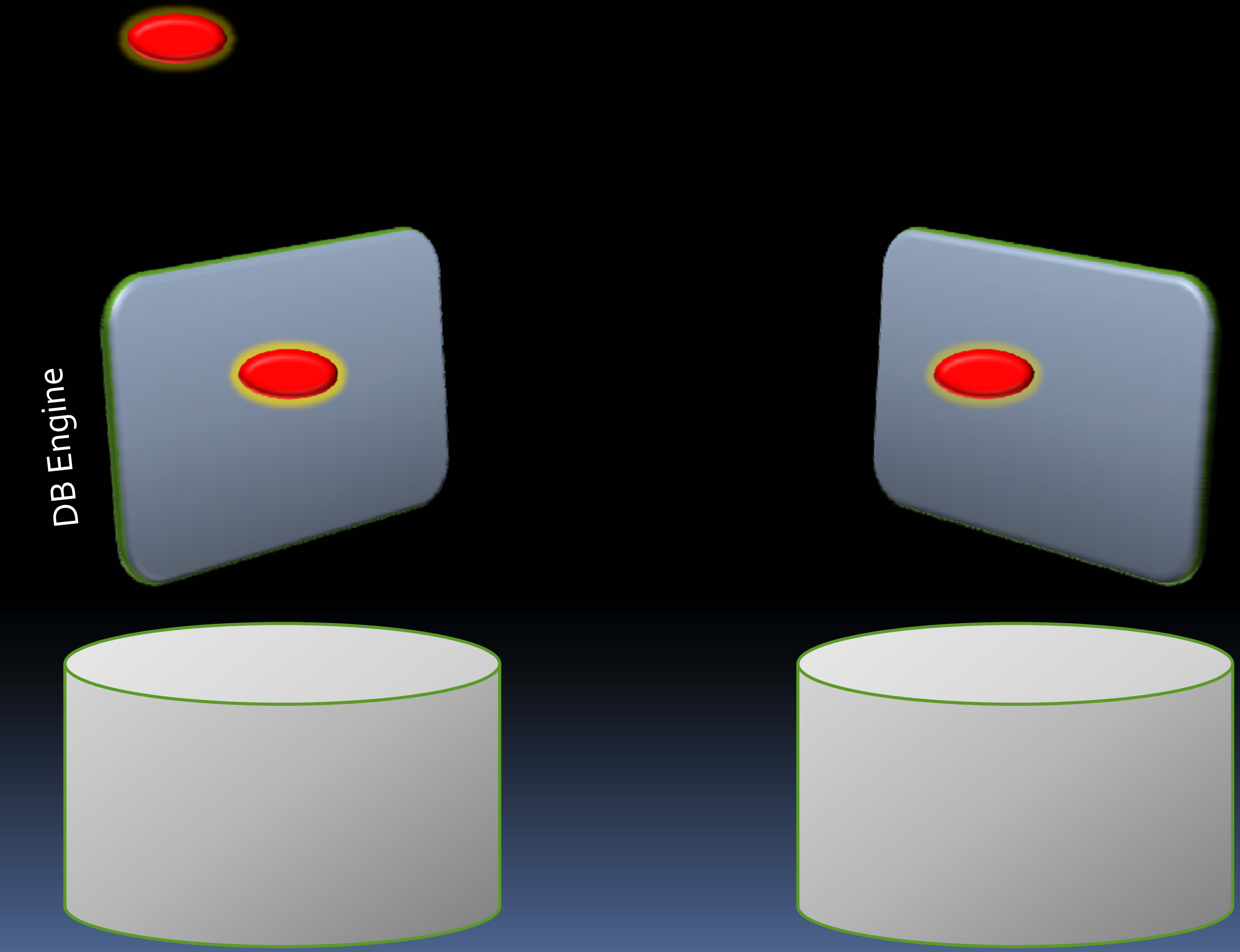
Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

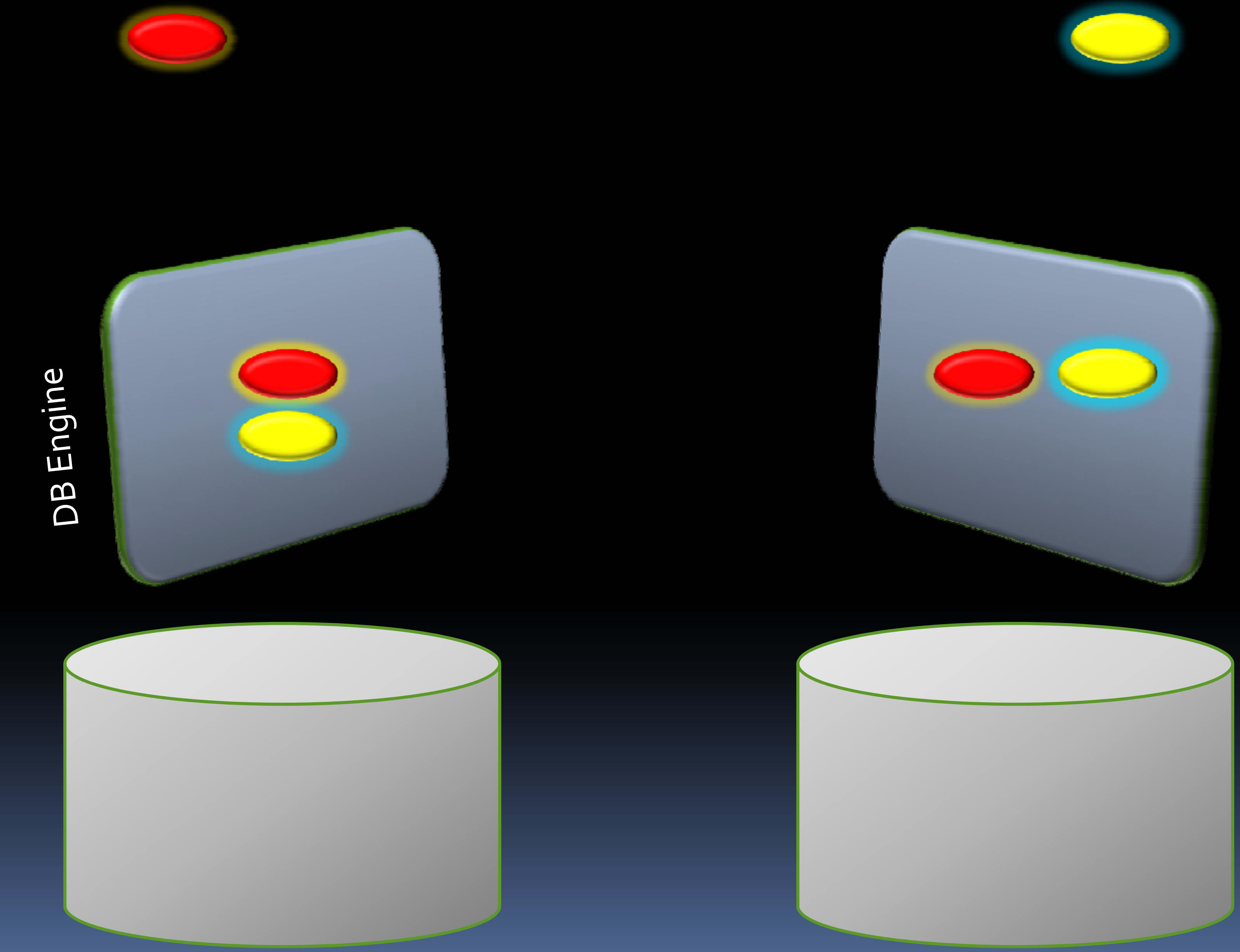
Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.

A brief introduction to database replication

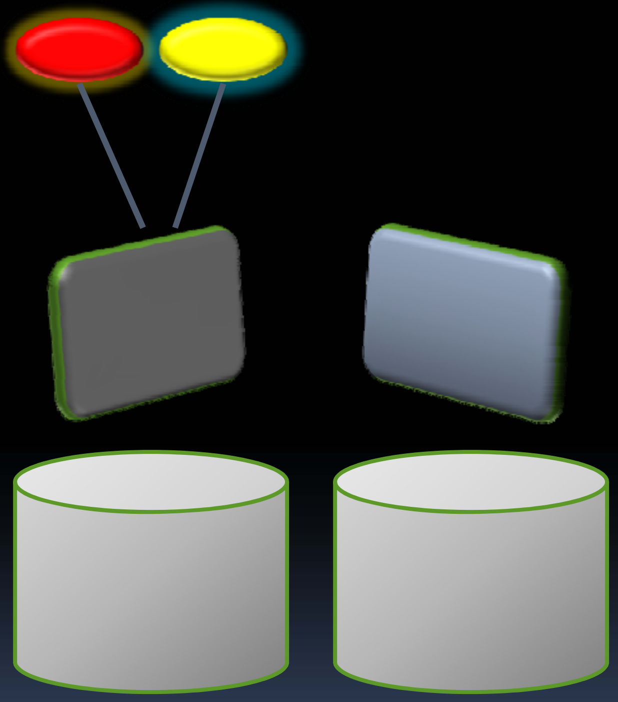


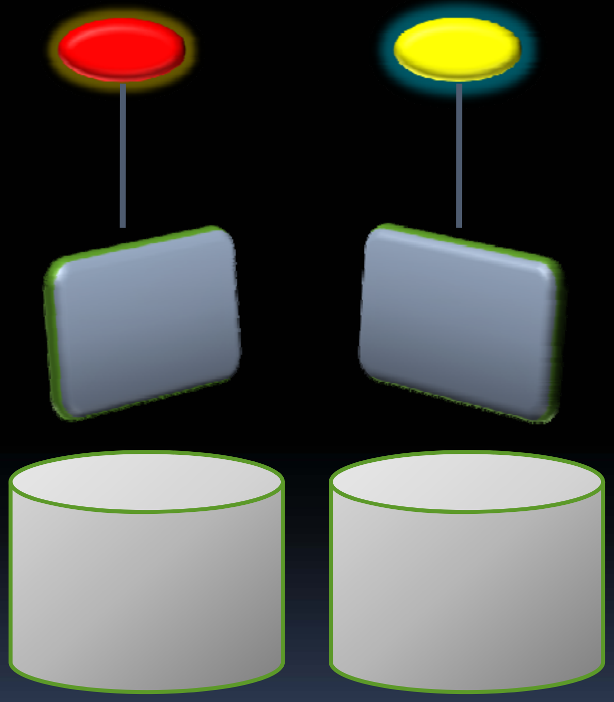
- Scalability
- Fault-tolerance
- Fast access
- Special purpose copies


DB Engine
DB Engine

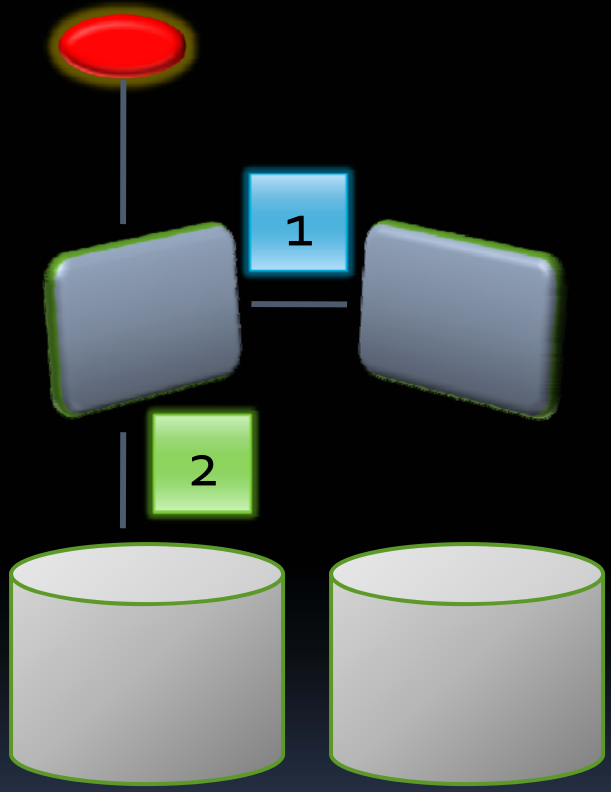

DB Engine
DB Engine

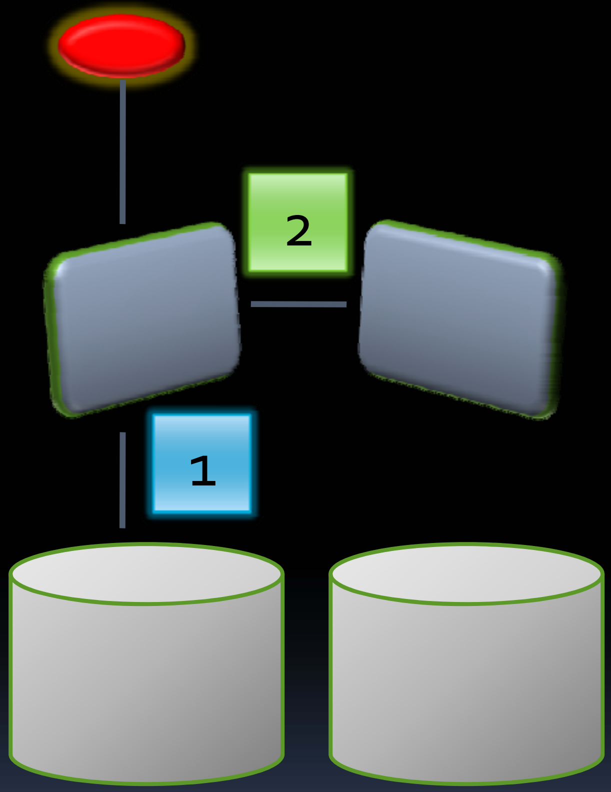
Primary Copy vs. Update Everywhere





Eager (synchr.) vs. Lazy (asynchr.)





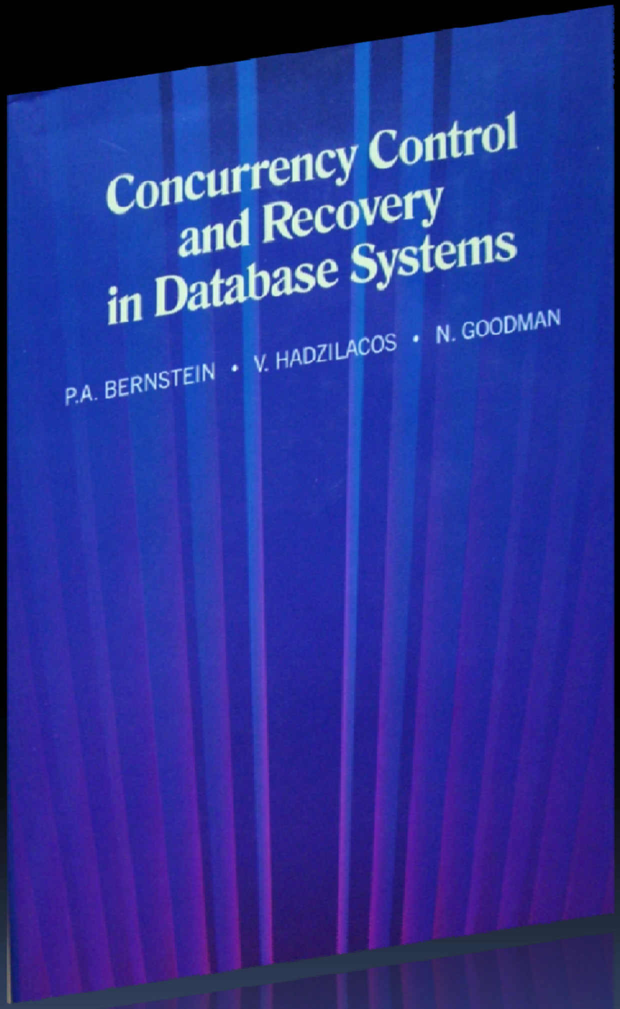
Theory of replication 10 years ago

	PRIMARY COPY	UPDATE EVERYWHERE
EAGER		
LAZY		

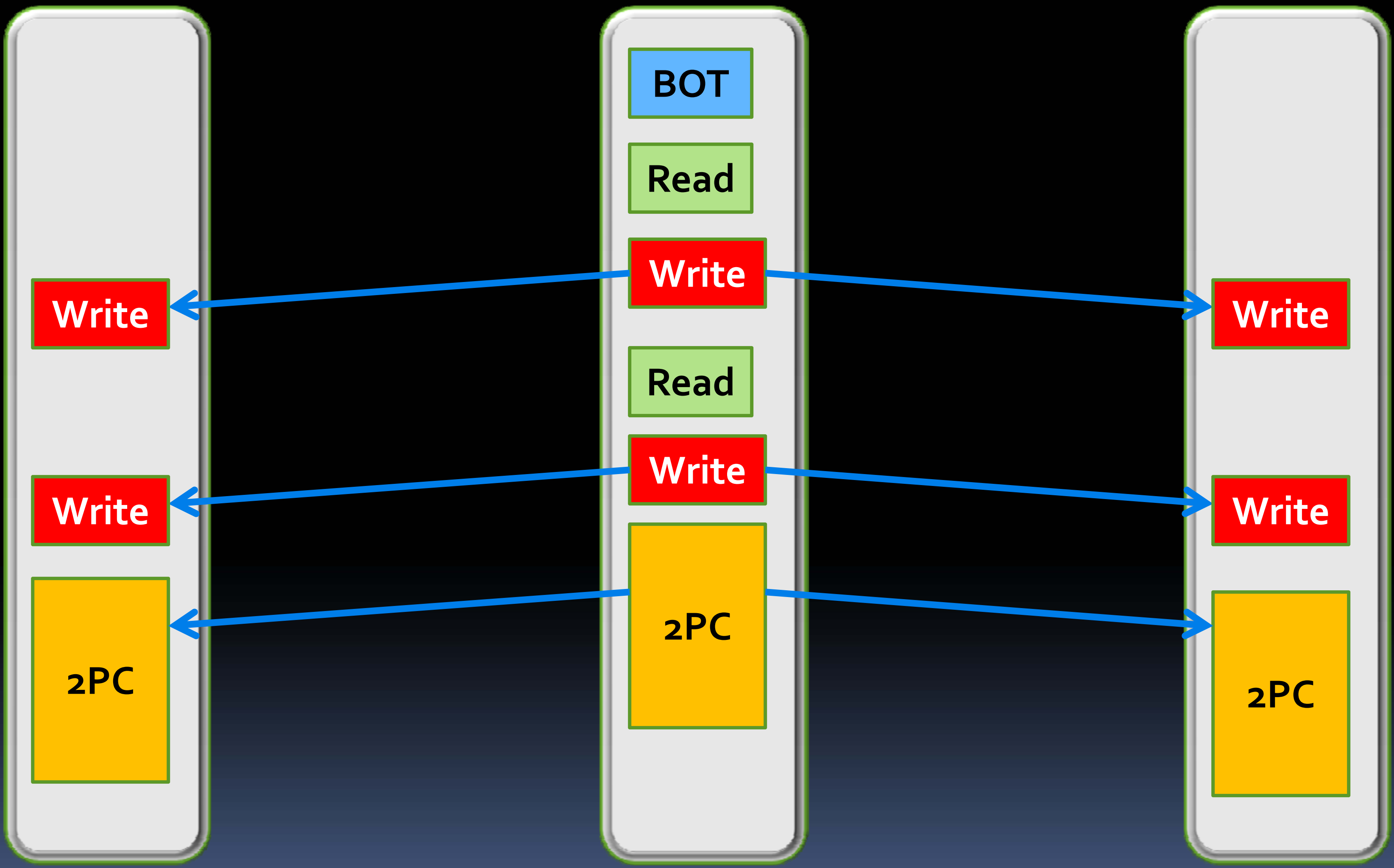
Replication in practice 10 years ago

	PRIMARY COPY	UPDATE EVERYWHERE
EAGER		
LAZY		

Replication in 2000



- Read-one / Write All
- Distributed locking (writes)
- 2 Phase Commit


BOT
Read
Write
Write
Write
Read
Write
Write
Write
2PC
2PC
2PC

The Dangers of Replication

Gray et al. SIGMOD 1996

The Dangers of Replication and a Solution

Jim Gray (Gray@Microsoft.com)
Pat Helland (PHelland@Microsoft.com)
Patrick O'Neil (POneil@cs.UMB.edu)
Dennis Shasha (Shasha@cs.NYU.edu)

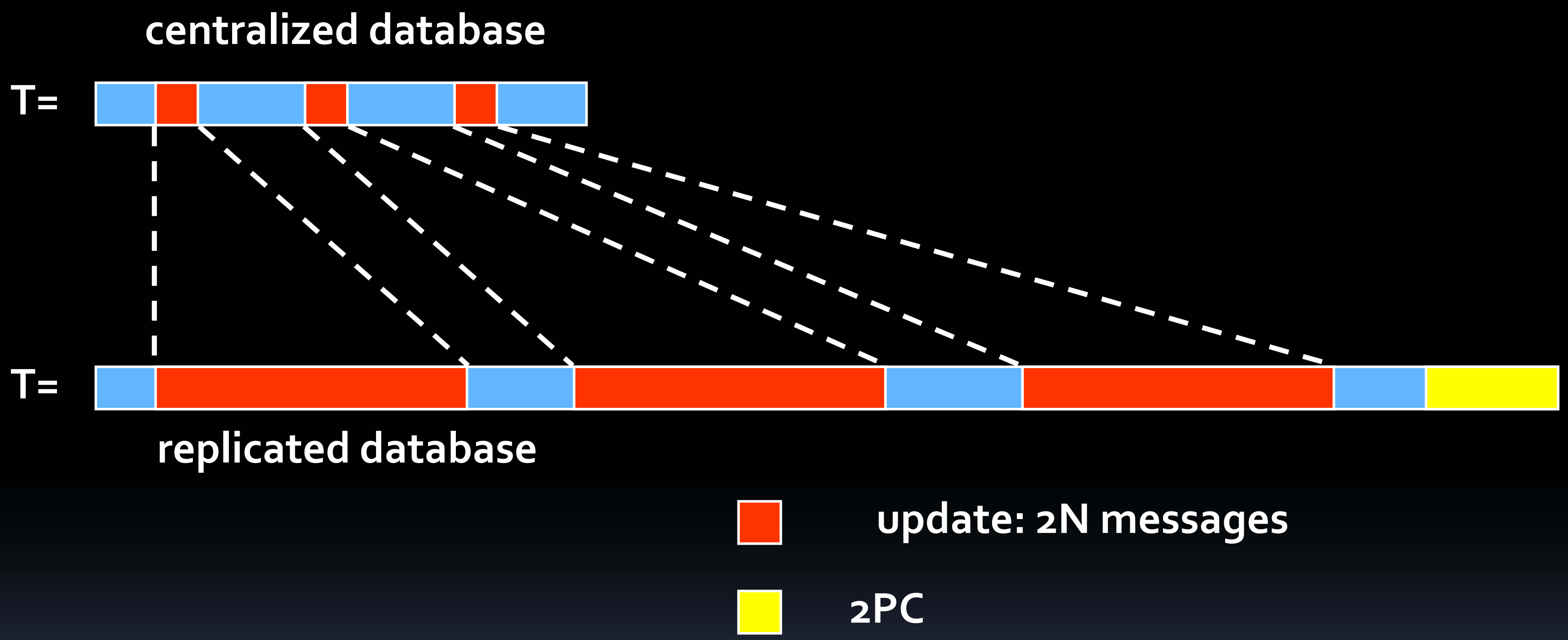
Abstract: *Update anywhere-anytime-anyway transactional replication has unstable behavior as the workload scales up: a ten-fold increase in nodes and traffic gives a thousand fold increase in deadlocks or reconciliations. Master copy replication (primary copy) schemes reduce this problem. A simple analytic model demonstrates these results. A new two-tier replication algorithm is proposed that allows mobile (disconnected) applications to propose tentative update transactions that are later applied to a master copy. Commutative update transactions avoid the instability of other replication schemes.*

1. Introduction

Eager replication delays or aborts an uncommitted transaction if committing it would violate serialization. Lazy replication has a more difficult task because some replica updates have already been committed when the serialization problem is first detected. There is usually no automatic way to reverse the committed replica updates, rather a program or person must ***reconcile*** conflicting transactions.

To make this tangible, consider a joint checking account you share with your spouse. Suppose it has $1,000 in it. This account is replicated in three places: your checkbook, your spouse's checkbook, and the bank's ledger.

Response Time and Messages



… and that's not all

- Network becomes an issue
  - Messages = copies x write operations
- Quorums?
  - Reads must be local for complex SQL operations
  - Different in key value stores (e.g., Cassandra)

Our goal

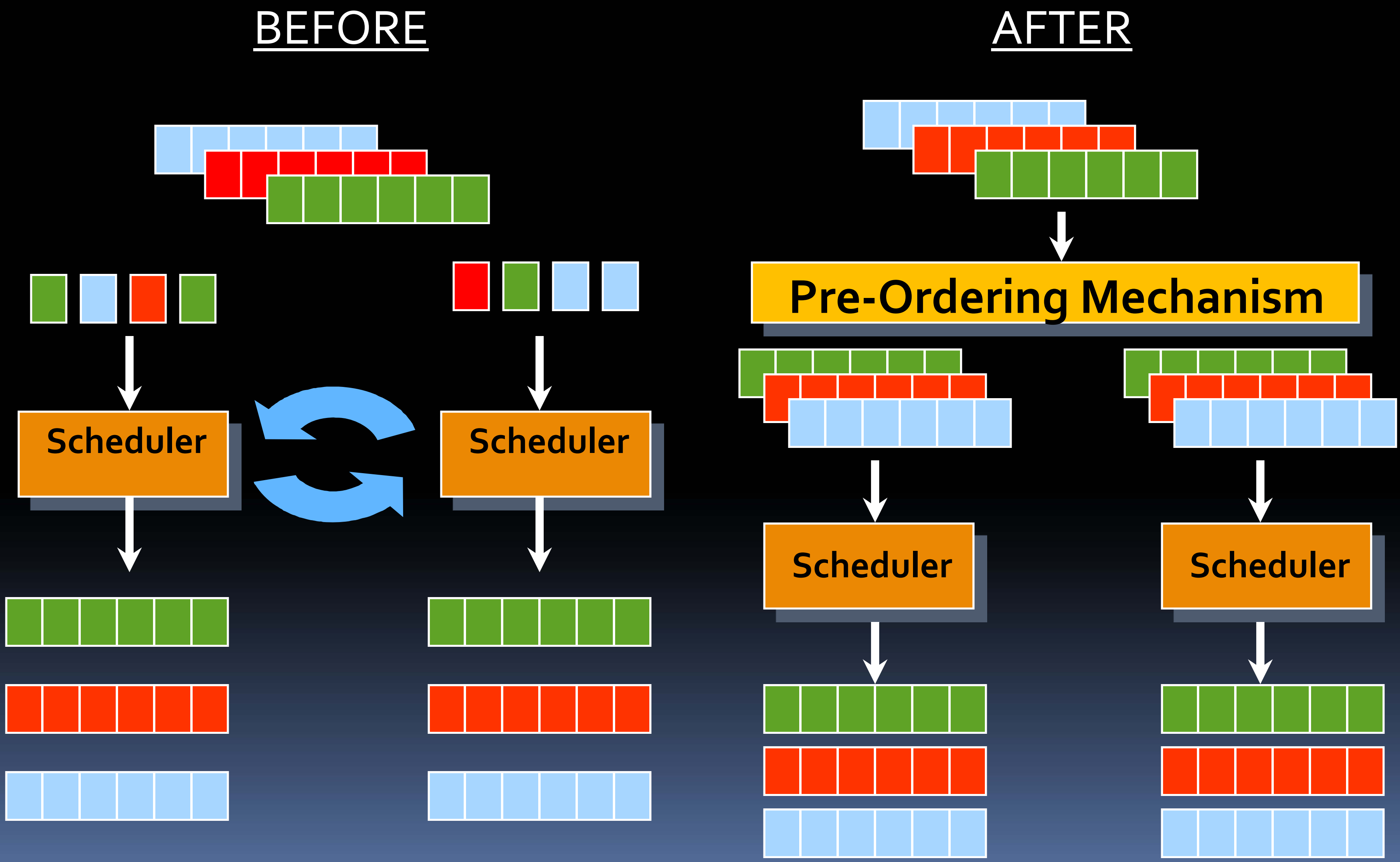
Can we get scalability and consistency when replicating a database?

Postgres-R in detail

Fundamentals

- Exploitation of group communication systems
  - Ordering semantics
    - Affect isolation / concurrency control
  - Delivery semantics
    - Affect atomicity

Key insight in Postgres-R

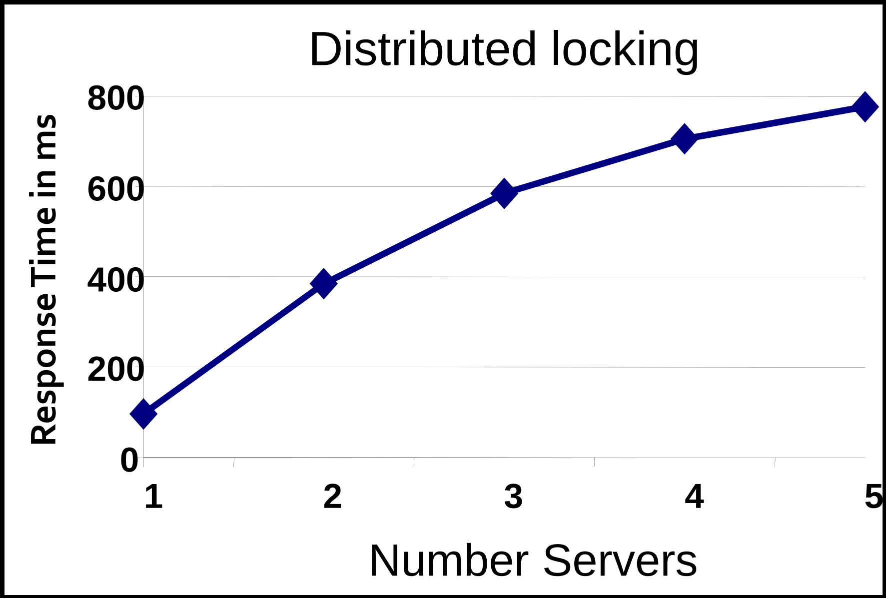


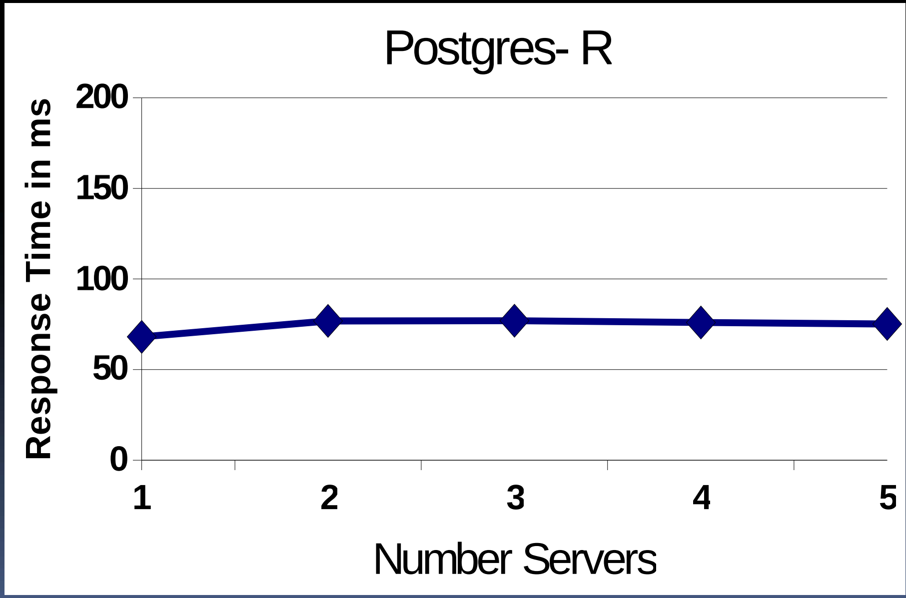
The devil is in the details

- Total order to serialization order
- Provide various levels of isolation / atomicity degrees
- Read operations always local
- Propagate changes on transaction basis
- No 2PC
  - Rely on delivery guarantees
  - Return to user once local replica commits
- Determinism
- Propagate changes vs. SQL statements

Distributed locking

Did we really avoided the dangers of replication?





Postrges-R removed a lot of the overhead of replication, providing scalability while maintaining strong consistency

In perspective

What worked

- Ordered and guaranteed propagation of changes through an agreement protocol external to the engine
- The implementation was crucial to prove the point
- Thinking through the optimizations / real system issues
- Levels of consistency

What did not work

- Modify the engine
  - Today middleware based solutions
- Enforce serializability
  - Today SI and session consistency
  - Data warehousing less demanding
  - Cloud computing has lowered the bar

Systems today

Very rich design space

- Applications (OLAP vs OLTP)
- Data layer (DB vs. others)
- Throughput/Response time
- Staleness
- Availability guarantees
- Partial vs. full replication
- Granularity of changes, operations

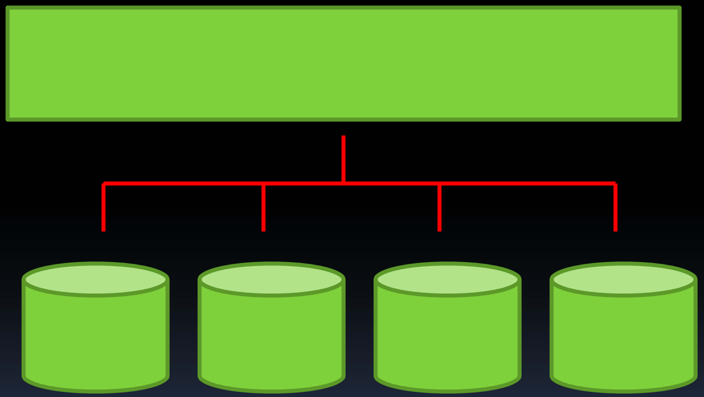
A Suite of Replication Protocols

	Correctness	Local decisions	Mssgs/txn	Problems
Serializability	1-copy serializability	write locks abort conflicting local read locks	2 mssgs write set confirm commit	very high abort rate
Cursor Stability	no lost updates no dirty reads	write locks abort conflicting local read locks	2 mssgs write set confirm commit	inconsistent reads
Snapshot isolation	allows write skew	first writer to commit wins (deterministic)	1 mssg write set	high abort rate for update hot-spots
Hybrid	1-copy serializability	as serializability for update transactions	2 mssgs write set confirm commit	requires to identify the queries

6 versions of each protocol depending on delivery guarantees and whether deferred updates or shadow copies are used (22 protocols)
[Kemme and Alonso, ICDCS'98]

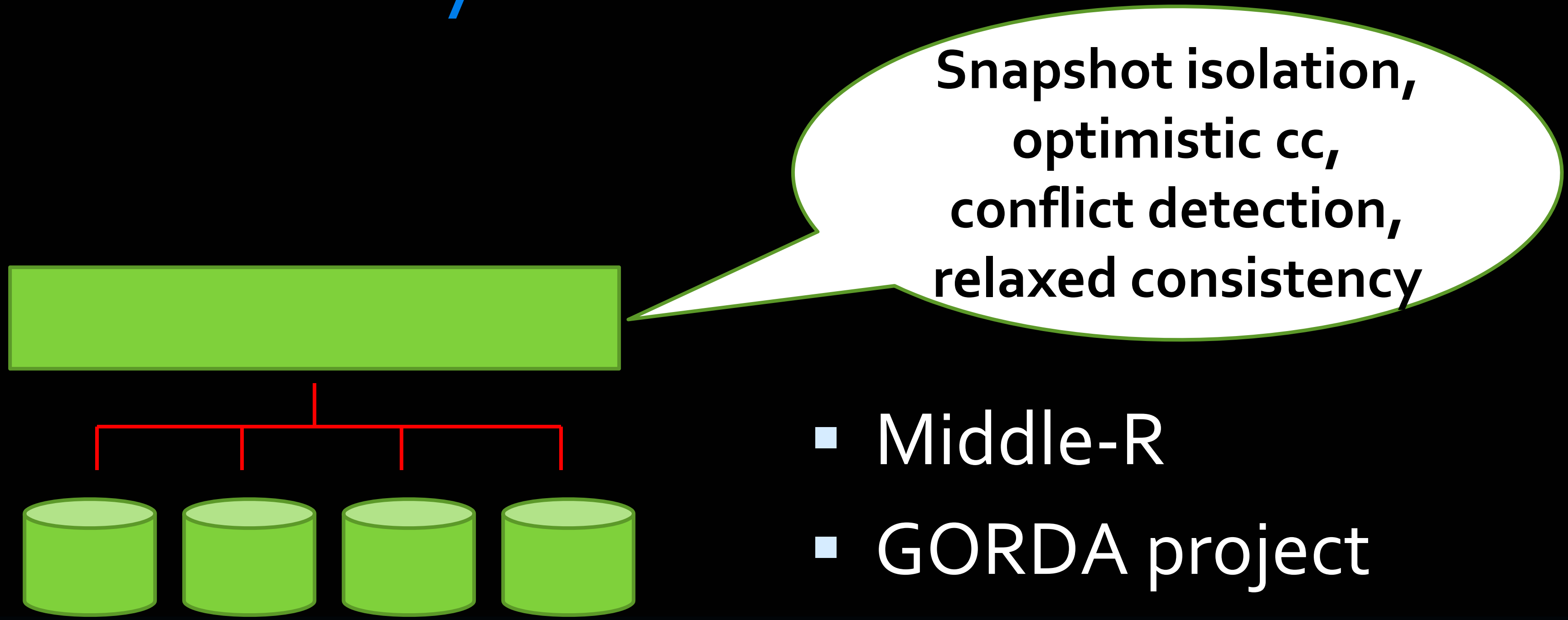
Architecture

- Eventually all this moved to a middleware layer above the database



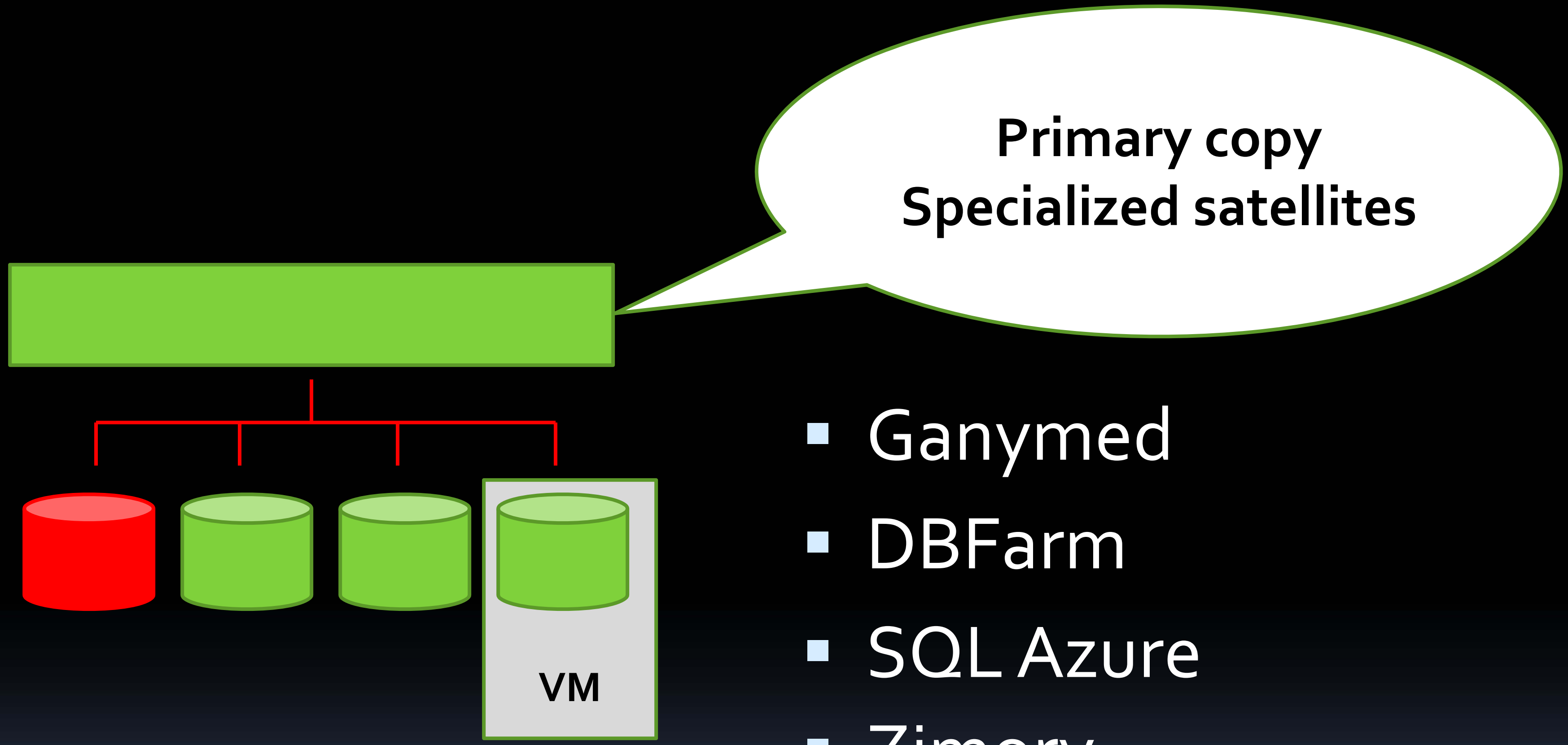
- Middle-R
- All systems out there today

Consistency variations



- Middle-R
- GORDA project
- Tashkent
- Pronto/Sprint
- C-JDBC
- C. Amza et al.

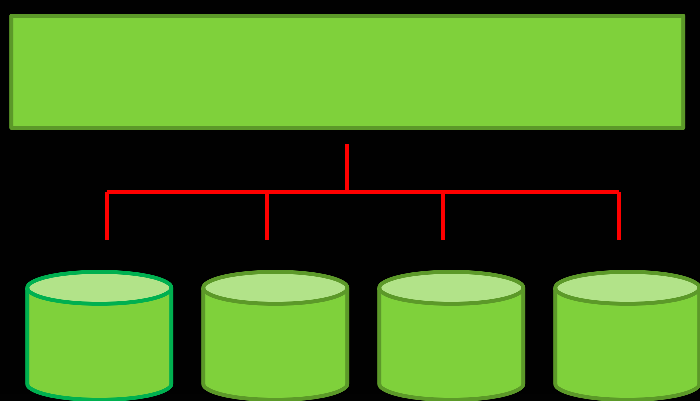
Architectural variations



- Ganymed
- DBFarm
- SQL Azure
- Zimory (virtualized satellites)

Engine variations

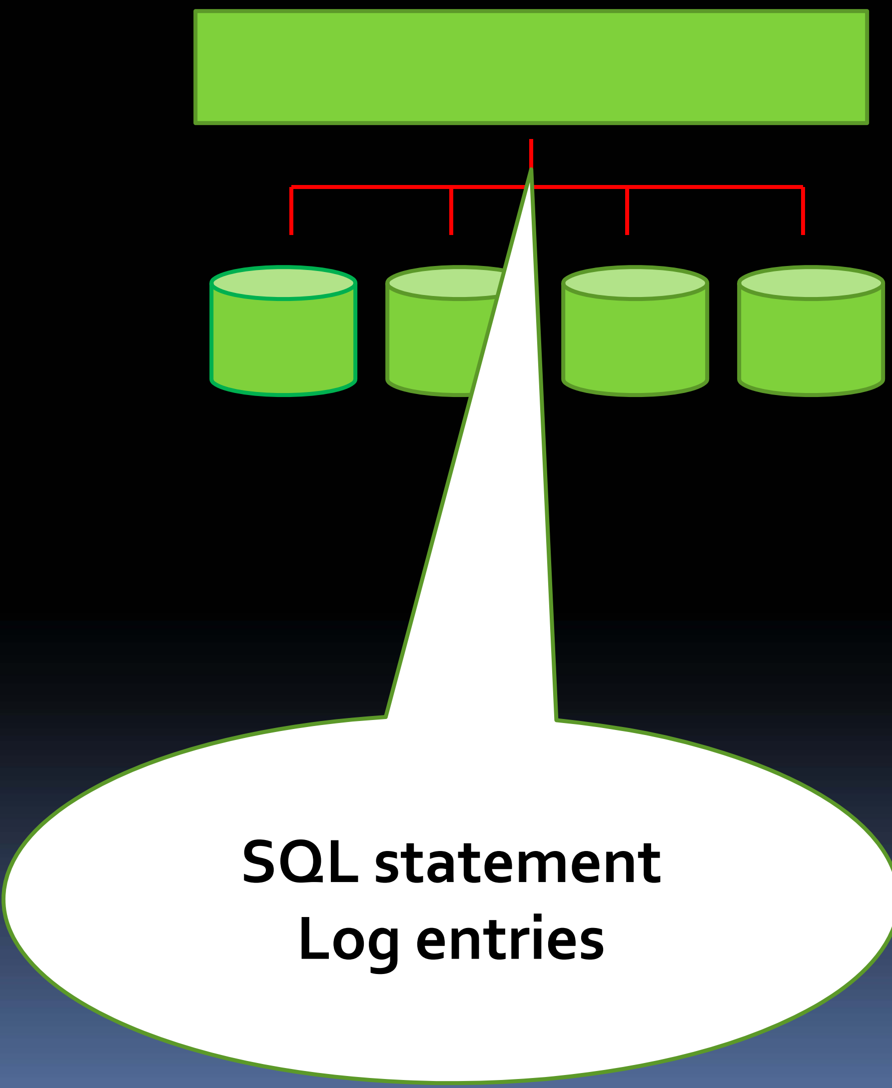
- Xkoto (Teradata)
- Galera
- Continuent
- SQL Azure
- Ganymed



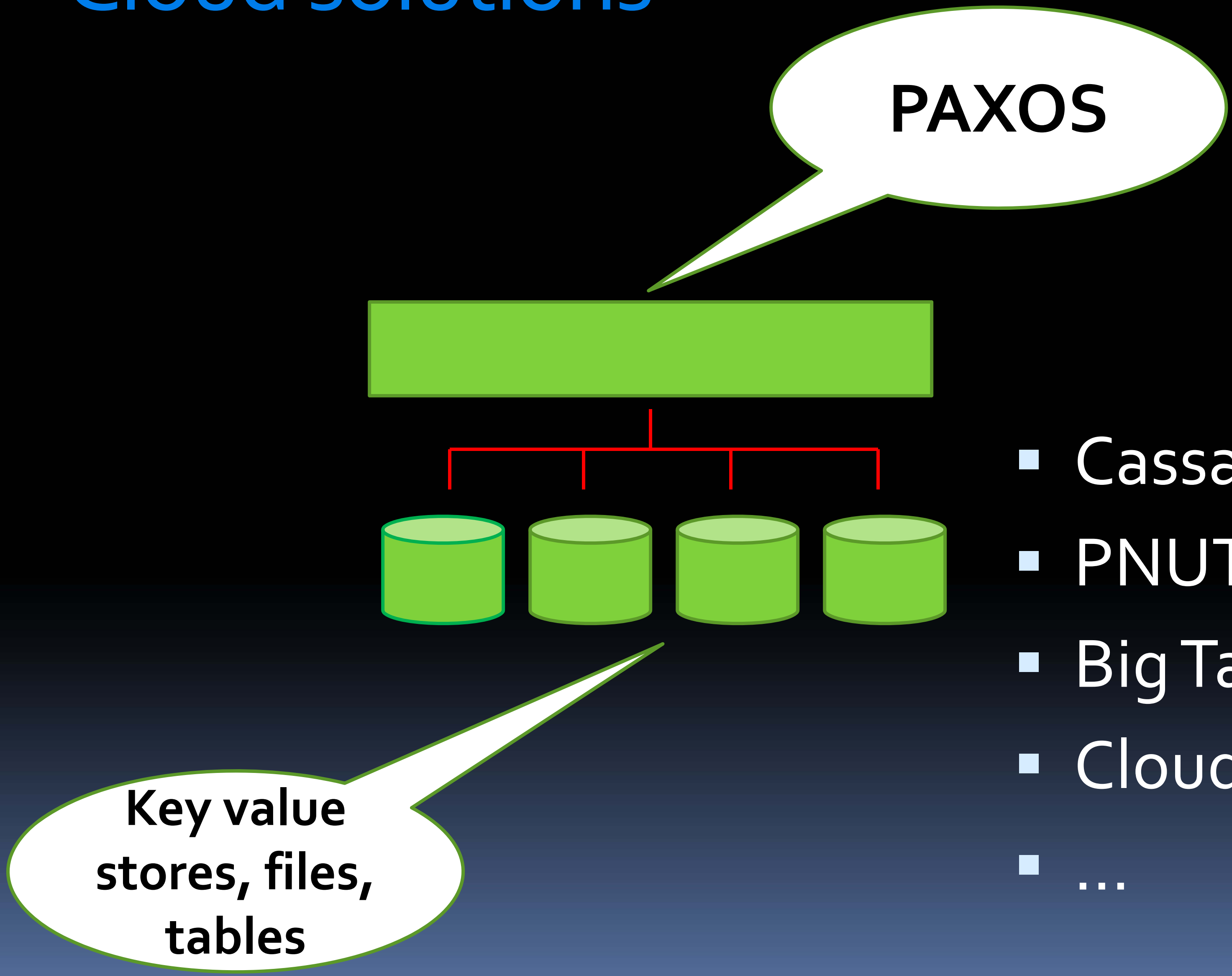
MySQL, DB2,
SQL Server, Oracle,
Heterogeneous

Change unit variations

- Xkoto (Teradata)
  - SQL statement propagation
- Continuent
  - Log capture
- Determinism!



Cloud solutions



- Cassandra
- PNUTS
- Big Table
- Cloudy
- …

With a lot of related topics

- Consistency (Khuzaima et al., Cahill et al.)
- Applications (Vandiver et al.)
- Agreement protocols (Lamport et al.)
- Determinism (Thomson et al.)
- Recovery (Kemme et al., Jimenez et al.)

…

The next 10 years

Understanding the full picture

- Paxos- Group communication protocols differ in the exact properties they provide
  - Often difficult to understand for outsiders
    - Subtleties in implementation and efficiency
  - Complex implementations
- Adjusting the agreement protocols to the needs of databases
  - Properties that suffice
  - Efficient implementation
- Plenty of use cases (one size does not fit all)

Database and distributed systems

- Databases and distributed systems have converged in practice
  - Many similar concepts
- Research
  - Work still done in separate communities
- Teaching
  - Dire need for joint courses (thanks to Amr El Abbadi and Divy Agrawal from UCSB!)

Thanks

- Andre Schiper, Fernando Pedone, Matthias Wiesmann (EPFL)
- Marta Patiño, Ricardo Jiménez (UPM)
- The PostgreSQL community
- PhD and master students at ETH and McGill who have worked and are working on related ideas
- Many colleagues and friends …