The VoltDB Main Memory DBMS

Michael Stonebraker, Ariel Weisberg

3/19/19

Presented by: Sushant Raikar

UNIVERSITY OF WATERLOO

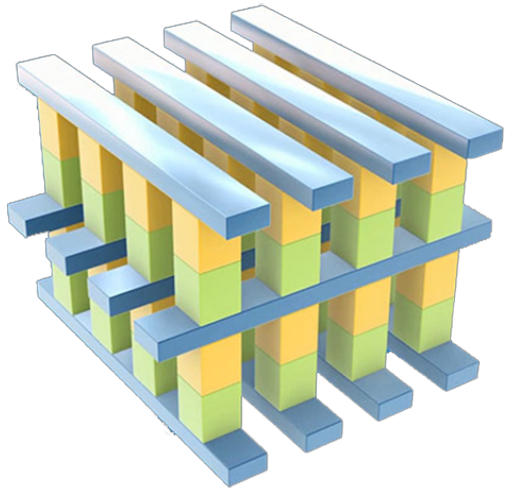
Motivation

1990s

- Expensive main memory
- Slow processors

2010s

- Main memory is cheaper
- Faster processors

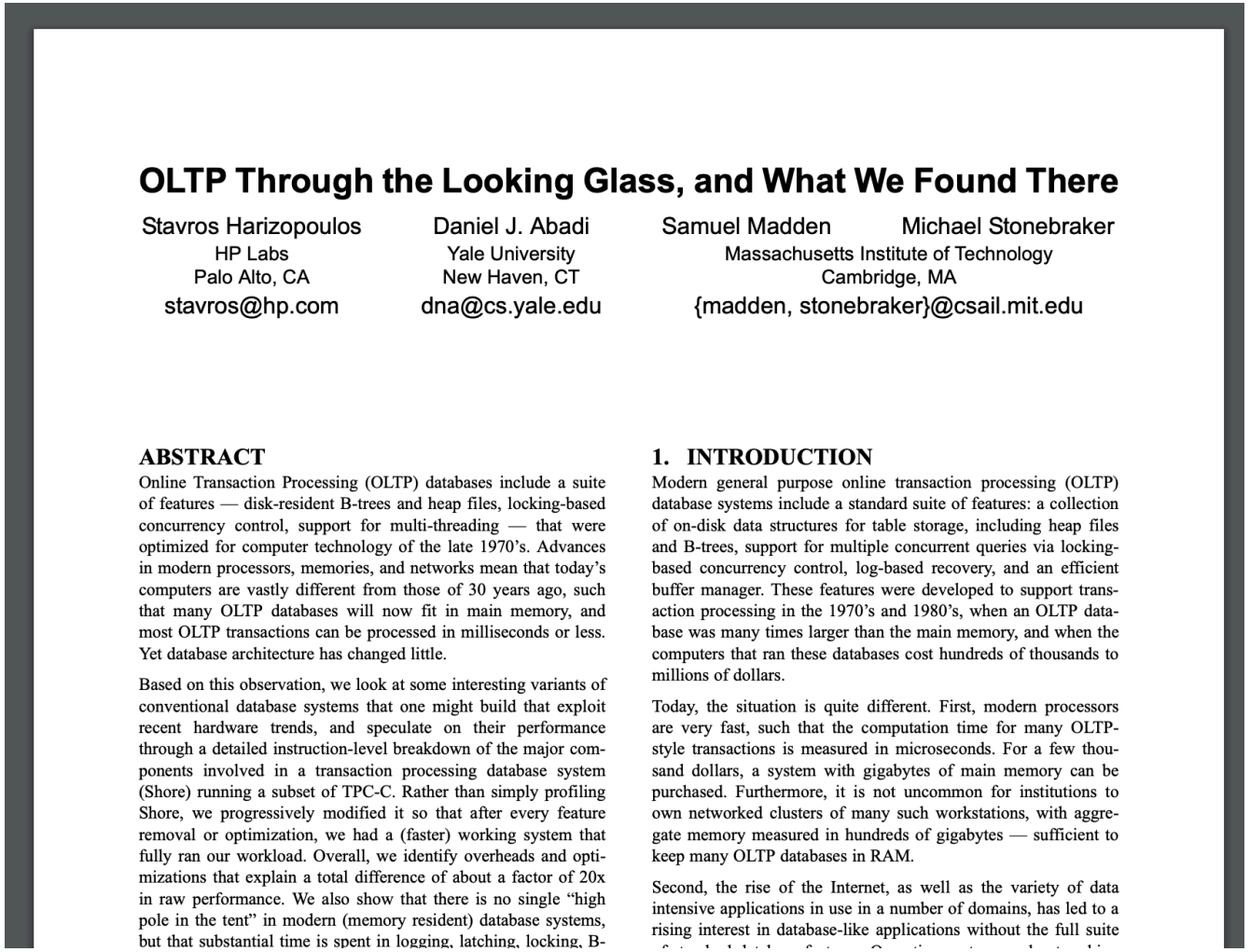


Future

- 100TBs can be deployed in memory
- PCM

Traditional DBMS

- 10% of time is spent on useful work
- 90% is just overhead
  - Buffer pool
  - Multi-threading
  - Locking
  - Logging



OLTP Through the Looking Glass, and What We Found There

Stavros Harizopoulos
HP Labs
Palo Alto, CA
stavros@hp.com

Daniel J. Abadi
Yale University
New Haven, CT
dna@cs.yale.edu

Samuel Madden Michael Stonebraker
Massachusetts Institute of Technology
Cambridge, MA
{madden, stonebraker}@csail.mit.edu

ABSTRACT

Online Transaction Processing (OLTP) databases include a suite of features — disk-resident B-trees and heap files, locking-based concurrency control, support for multi-threading — that were optimized for computer technology of the late 1970's. Advances in modern processors, memories, and networks mean that today's computers are vastly different from those of 30 years ago, such that many OLTP databases will now fit in main memory, and most OLTP transactions can be processed in milliseconds or less. Yet database architecture has changed little.

Based on this observation, we look at some interesting variants of conventional database systems that one might build that exploit recent hardware trends, and speculate on their performance through a detailed instruction-level breakdown of the major components involved in a transaction processing database system (Shore) running a subset of TPC-C. Rather than simply profiling Shore, we progressively modified it so that after every feature removal or optimization, we had a (faster) working system that fully ran our workload. Overall, we identify overheads and optimizations that explain a total difference of about a factor of 20x in raw performance. We also show that there is no single "high pole in the tent" in modern (memory resident) database systems, but that substantial time is spent in logging, latching, locking, B-

1. INTRODUCTION

Modern general purpose online transaction processing (OLTP) database systems include a standard suite of features: a collection of on-disk data structures for table storage, including heap files and B-trees, support for multiple concurrent queries via locking-based concurrency control, log-based recovery, and an efficient buffer manager. These features were developed to support transaction processing in the 1970's and 1980's, when an OLTP database was many times larger than the main memory, and when the computers that ran these databases cost hundreds of thousands to millions of dollars.

Today, the situation is quite different. First, modern processors are very fast, such that the computation time for many OLTP-style transactions is measured in microseconds. For a few thousand dollars, a system with gigabytes of main memory can be purchased. Furthermore, it is not uncommon for institutions to own networked clusters of many such workstations, with aggregate memory measured in hundreds of gigabytes — sufficient to keep many OLTP databases in RAM.

Second, the rise of the Internet, as well as the variety of data intensive applications in use in a number of domains, has led to a rising interest in database-like applications without the full suite

ASSUMPTIONS

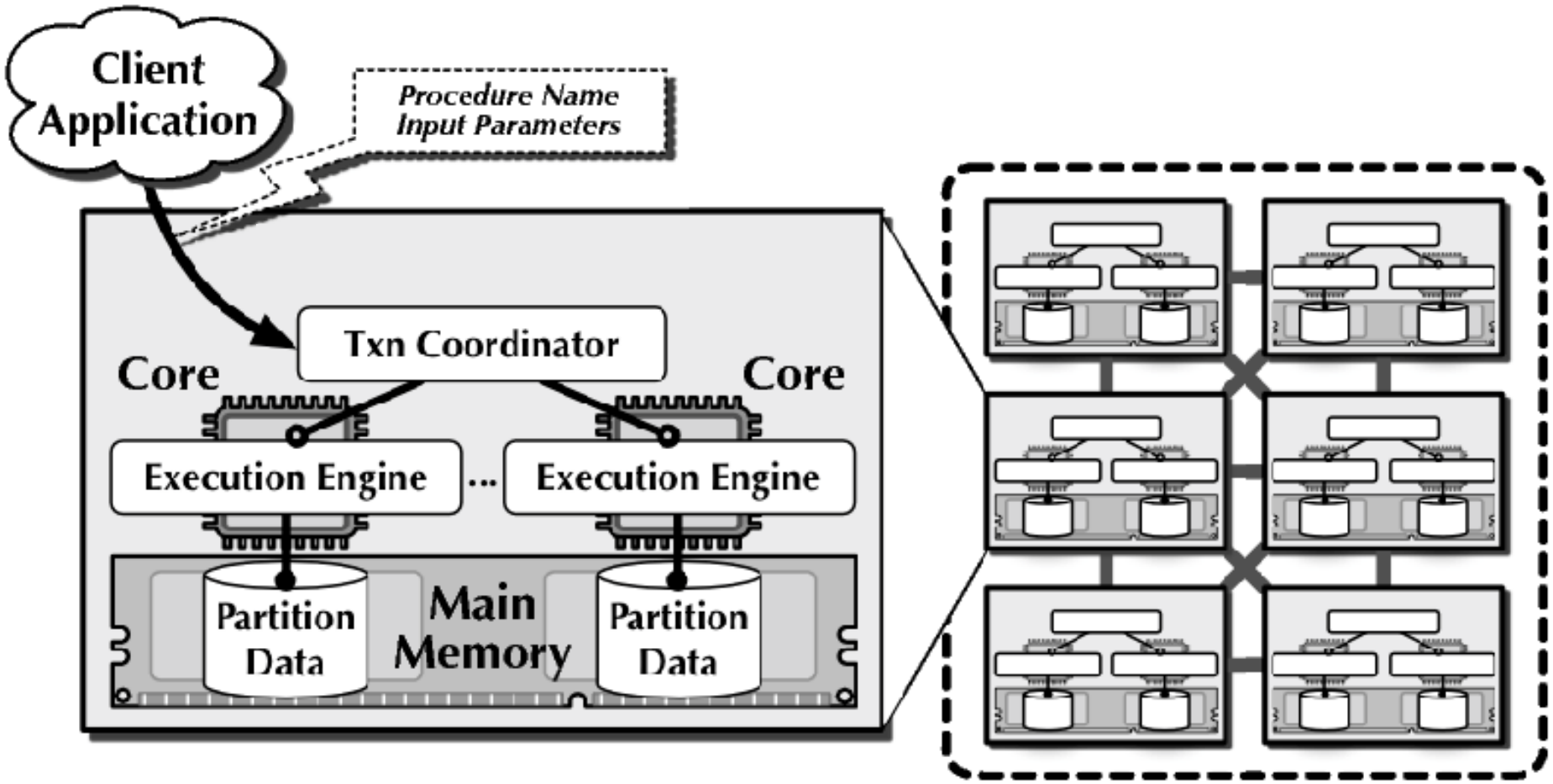
Assumptions

- High Availability is necessary
- ACID is necessary
- Distribution of transactions
  - Single-node
  - One-shot
  - General
- Transactions are deterministic

Assumption: Cluster

- Clusters are deployed on LAN
  - Network partitions are rare
  - Better latency
- Replication over WAN
  - Asynchronous

ARCHITECTURE


Client Application
Procedure Name
Input Parameters
Txn Coordinator
Core
Core
Execution Engine
...
Execution Engine
Partition Data
Main Memory
Partition Data

DESIGN CHOICES

Buffer-pool

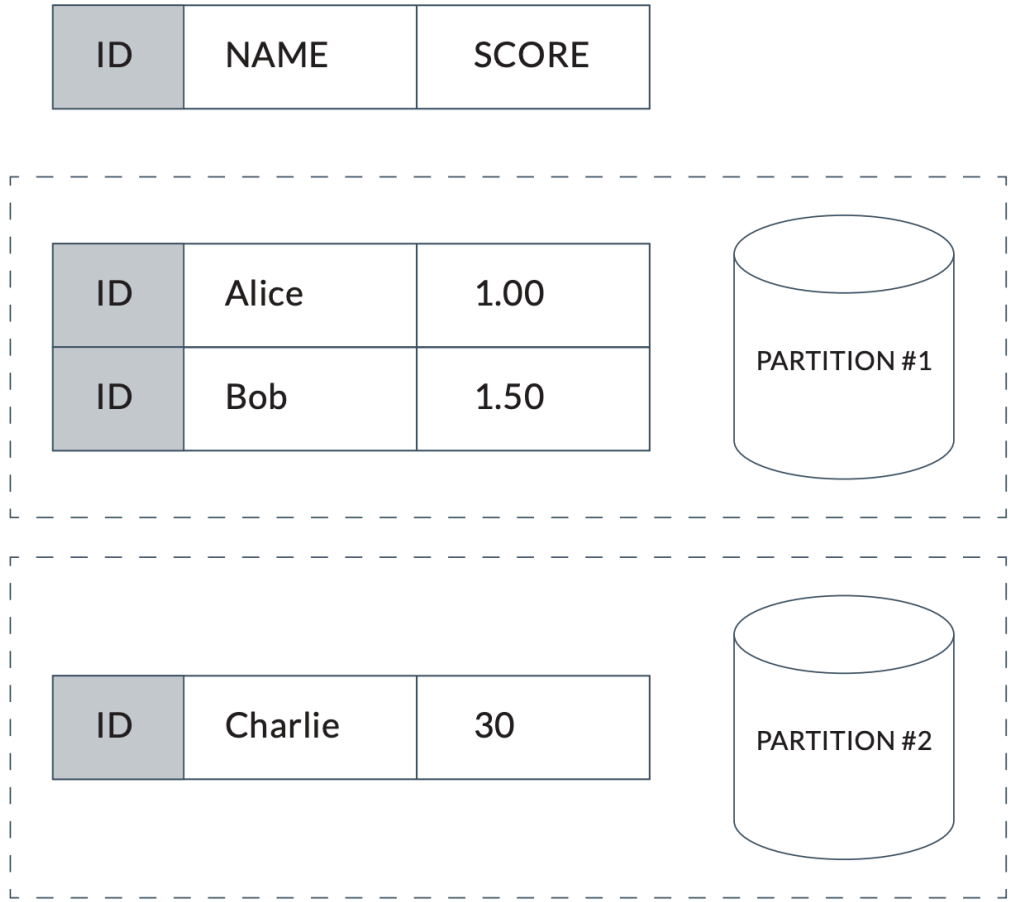
- Used for managing pages
- Tracks dirty or clean pages
- Not required for in-memory systems
  - Store data directly in memory
  - Use virtual memory pointers

Multi-threading & Locking

- Multi-threading causes race condition.
- VoltDB uses Share-nothing architecture
  - Memory chunk paired with single CPU

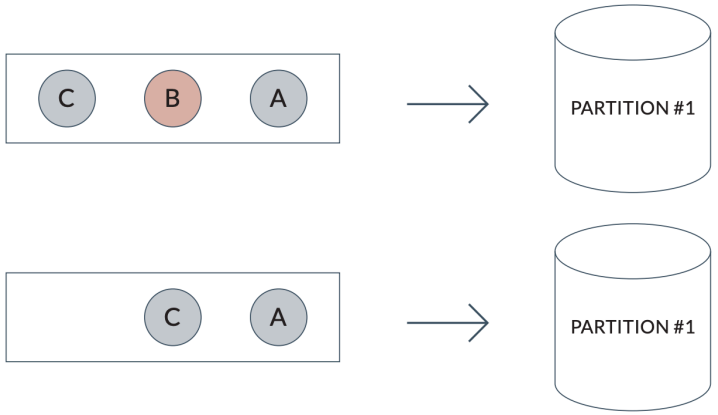
High Availability

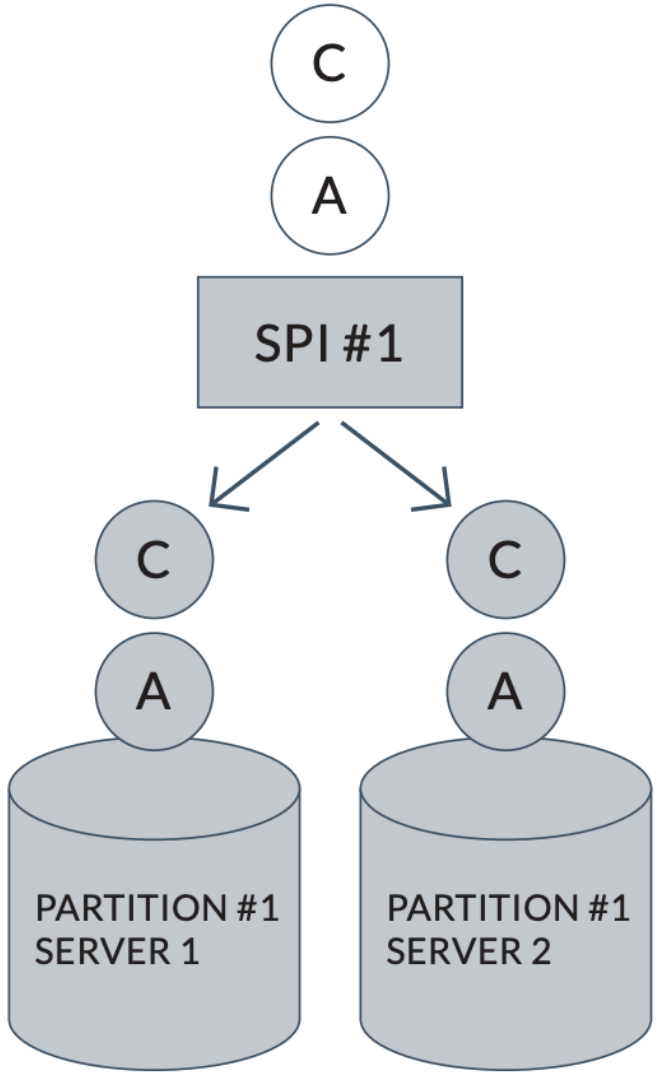
- Data is partitioned
  - Consistent Hashing
- Each partition is replicated
  - K-factor



Consistency

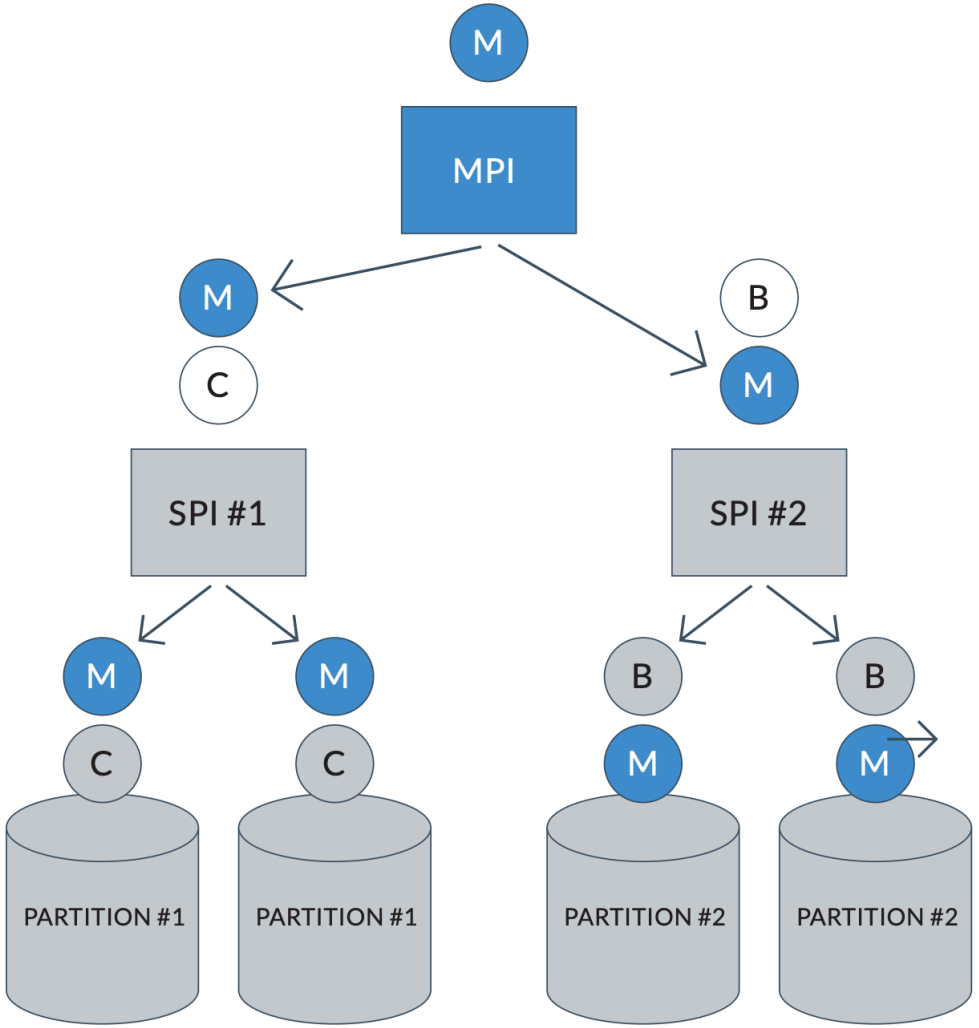
- Deterministic Transaction [Assumption]
- Uses “single partition initiator” to order transactions
- Reads are added without SPI





Consistency [Multi-node]

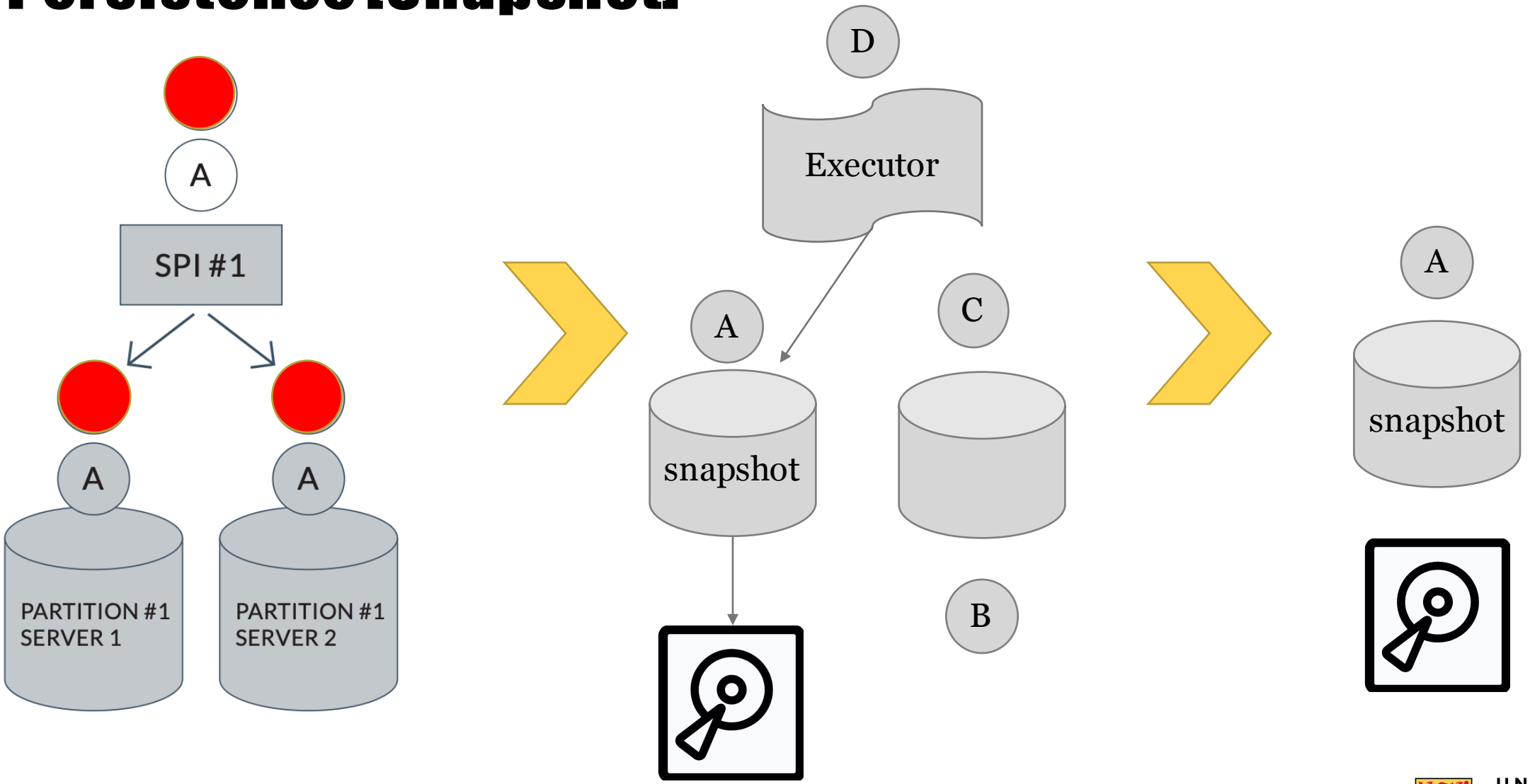
- Queries that span multiple partitions
  - Go through Multi-Partition initiator
  - C -> M -> B



Persistence

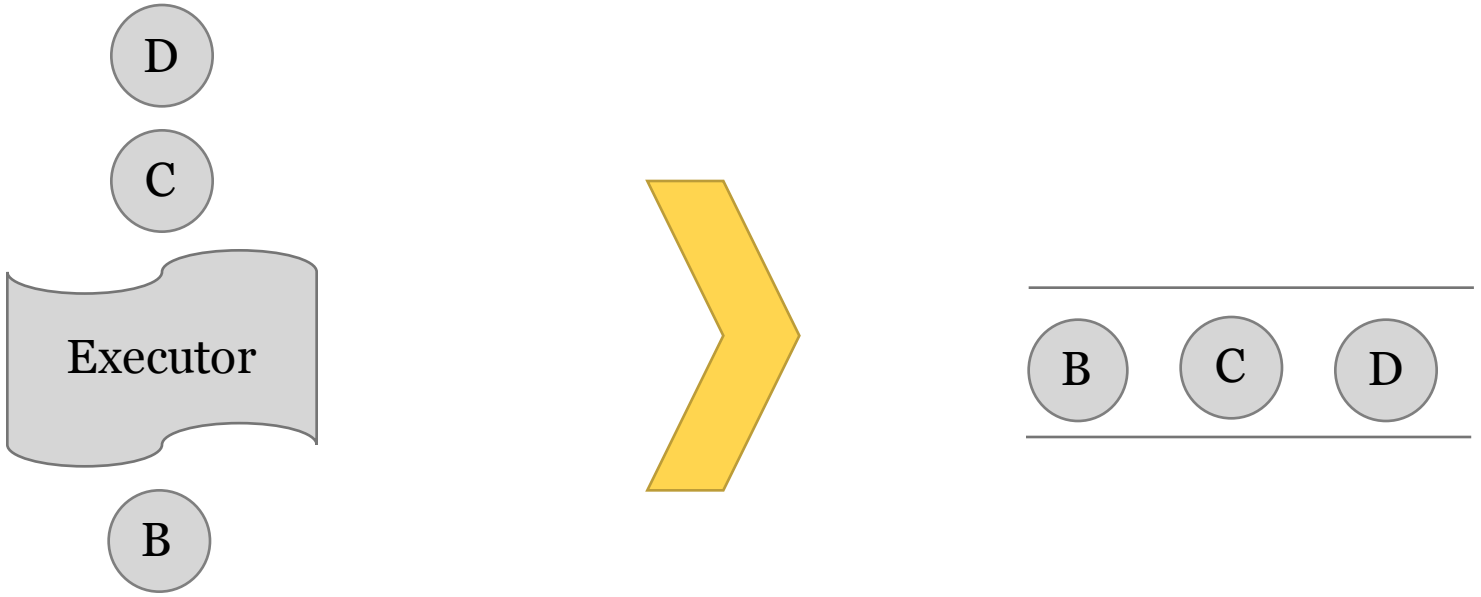
- Global Errors
- Solution:
  - Snapshot
  - Logging

Persistence [Snapshot]



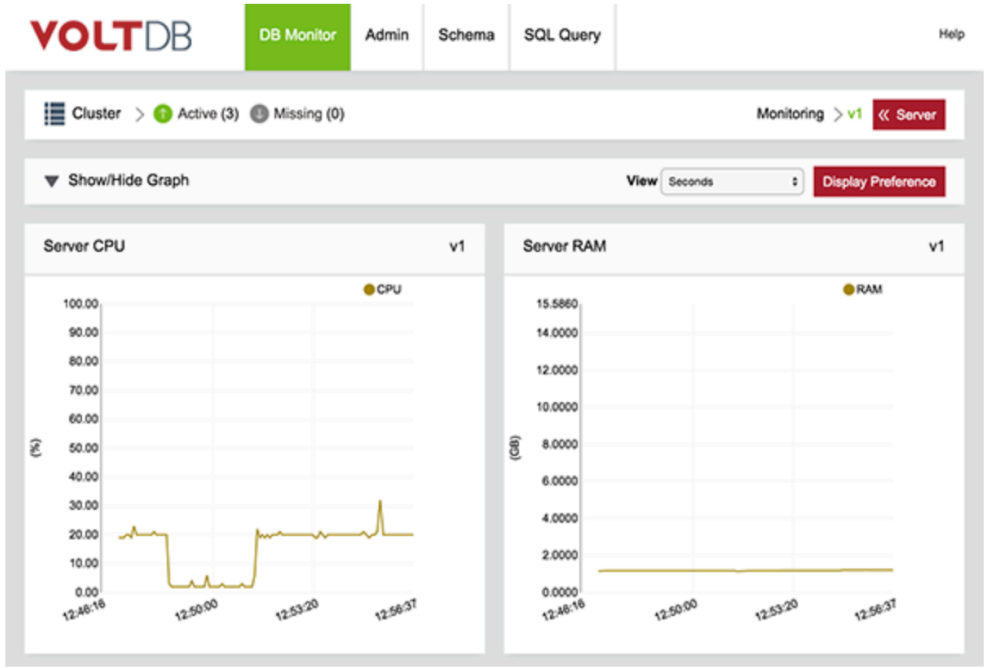
UNIVERSITY OF WATERLOO

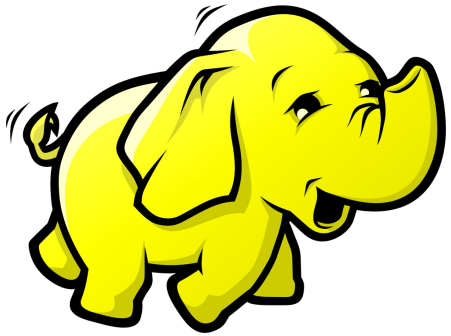
Persistence [Logging]



Development Focus in 2013

- SQL
- On-Line Reprovisioning
- Downstream Repositories





Application Areas

- High Speed updates
- Maintaining a state
  - Internet Games, Leaderboards
  - IOT
  - High Frequency Trading

Performance

- Voter Benchmark
  - Obeyed assumptions
  - Workload
    - Vote a candidate
    - Produce heat map for all states
    - Produce leaderboard once a second
  - Linear Scalability
    - 5 –node system -> 0.6 million transactions/second
    - 30 node system -> 3.4 million transactions/second

SUMMARY

Summary

- Traditional Design on main memory can do 10% of useful work
- No Buffer Pool
- No Multithreading, hence no locking
- Deterministic ordering
- Logging and Snapshot

TAKEAWAY

"One way to deal with a problem is to not have it."

VoltDB

THANK YOU

Summary

- TPC-Benchmark, indicates 10% of useful work
- No Buffer Pool
- No Multithreading, hence no locking
- Deterministic ordering
- Logging and Snapshot

Discussion Points

- How would you enforce deterministic transactions?
- What will be the impact of non-volatile memory on VoltDB?
- What could've been a better performance evaluation?
- Can group-commits cause inconsistency?
- Which point-of-view do you agree on, LeanStore (Buffer pool) VS VoltDB (No Buffer pool)?