Studying state-of-the-art HTAP systems

- Satya Sai Bharath Vemula
- Rwitam Bandyopadhyay
- Shubham Pandey

PURDUE UNIVERSITY

Table of Contents

PURDUE UNIVERSITY®

SnappyData

Disagg1

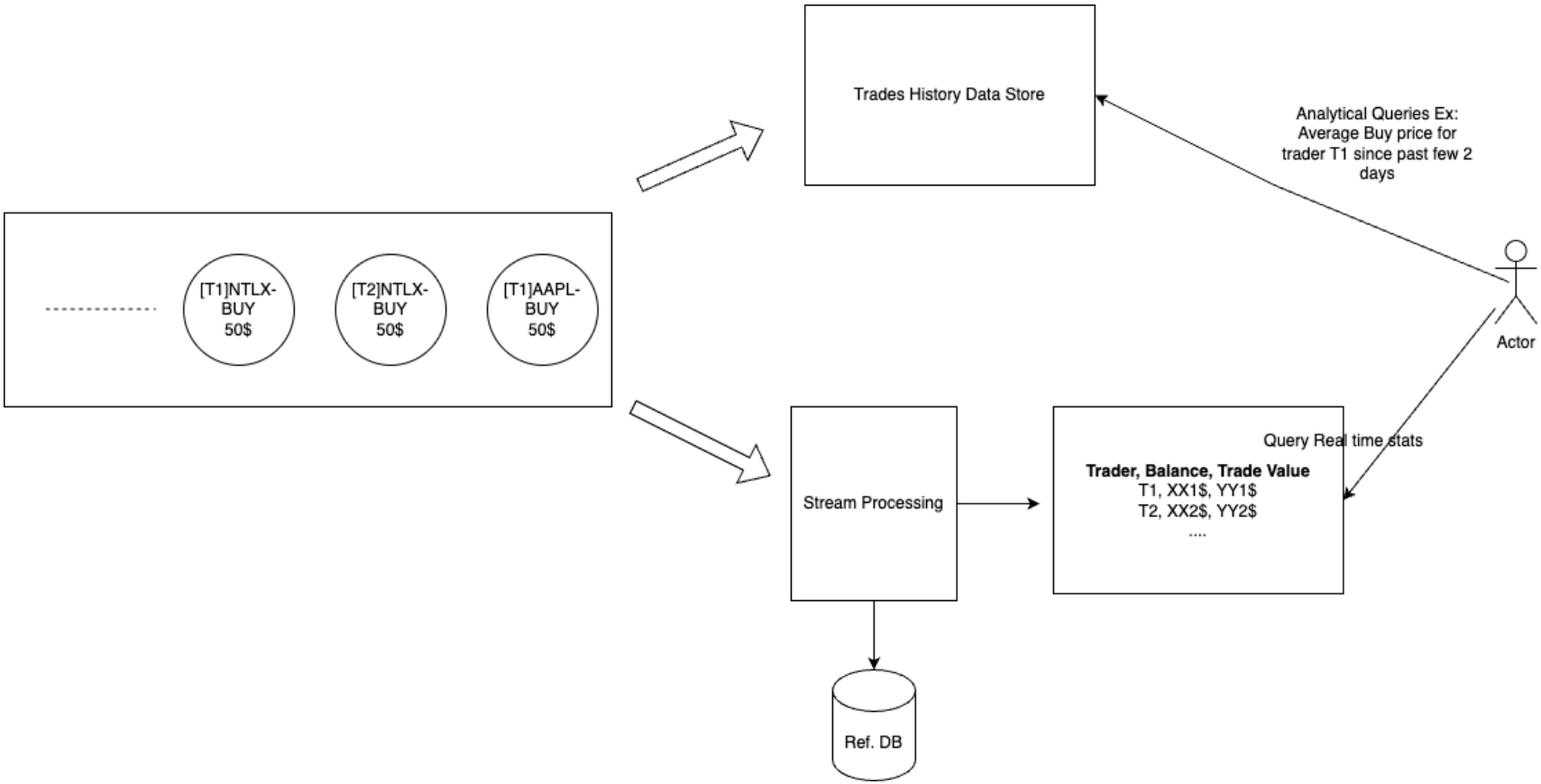
Presented by Satya Sai Bharath Vemula

PURDUE UNIVERSITY

Introduction

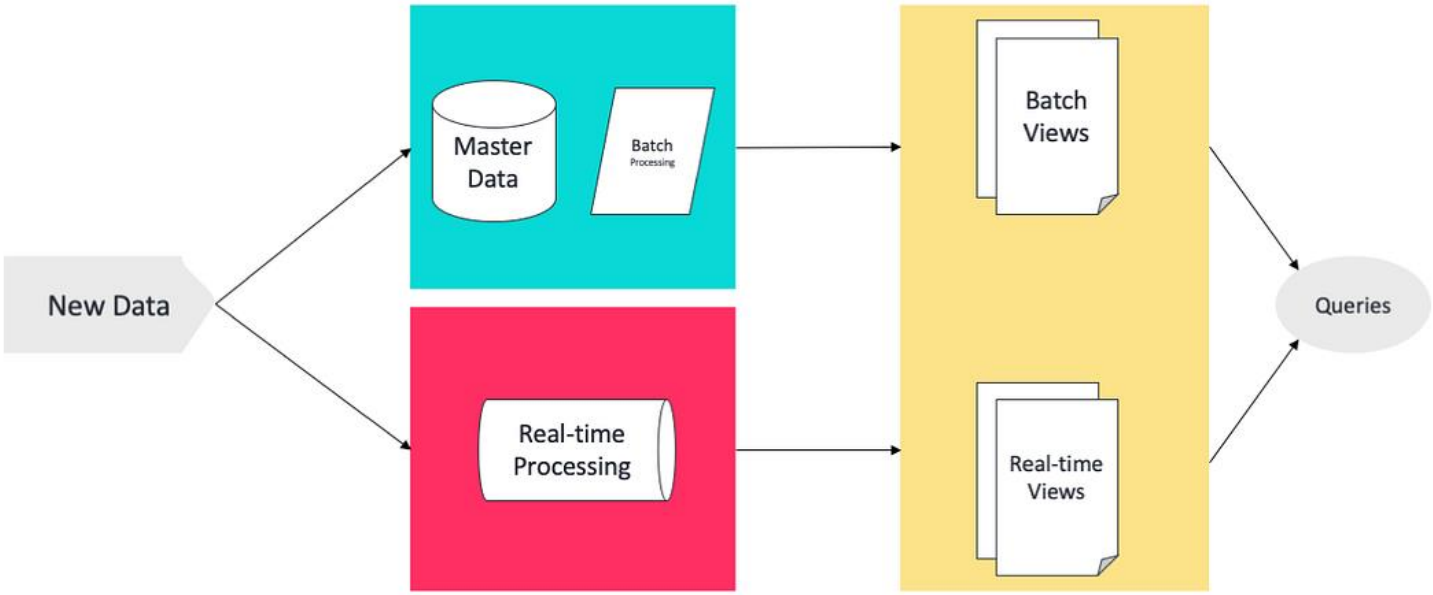
- Recently acquired by TIBCO and re-named as TIBCO Compute DB
- A Unified Cluster for OLTP, OLAP and Streaming
- This is a HTAP database which was primarily envisioned to solve problems with Lambda Architecture

Trading System



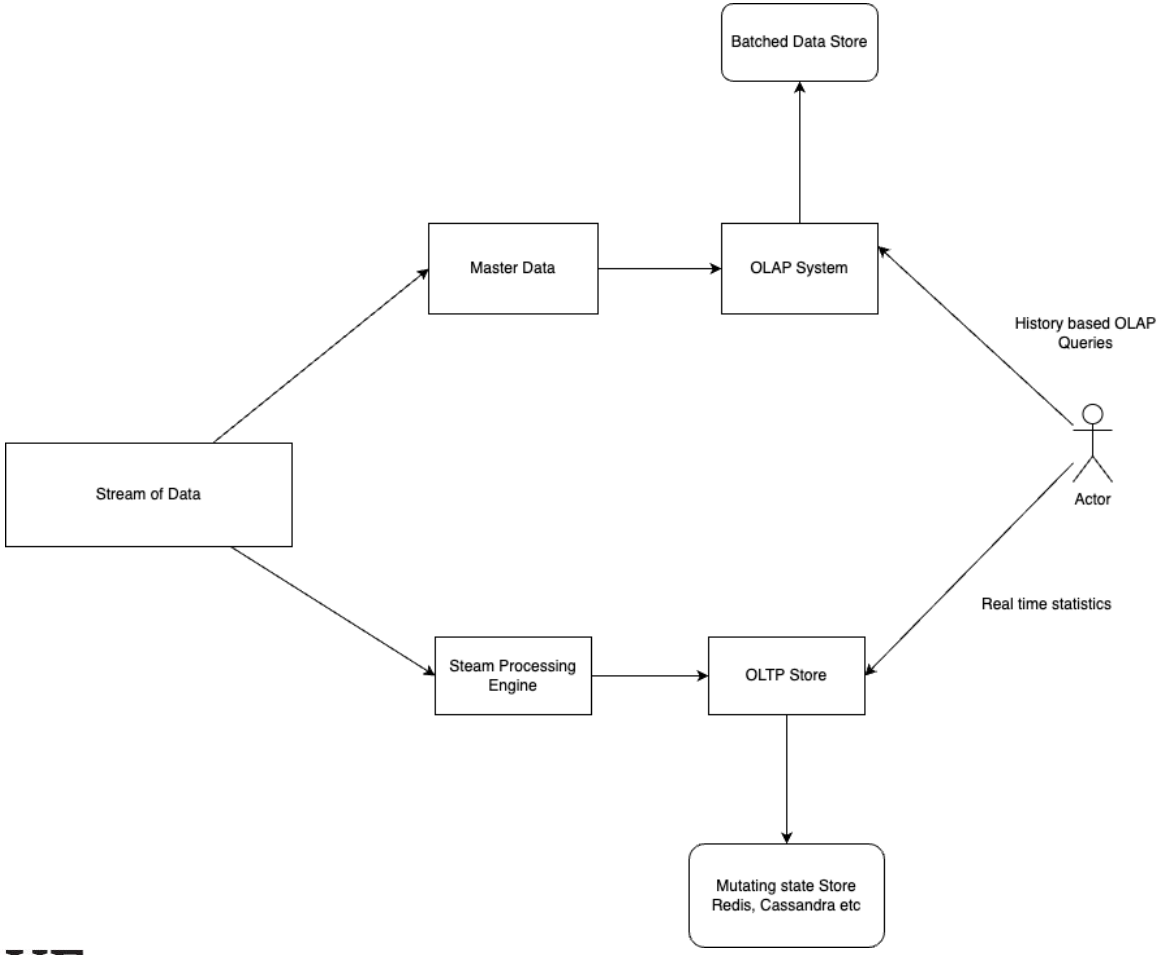
PURDUE UNIVERSITY

Lambda Architecture

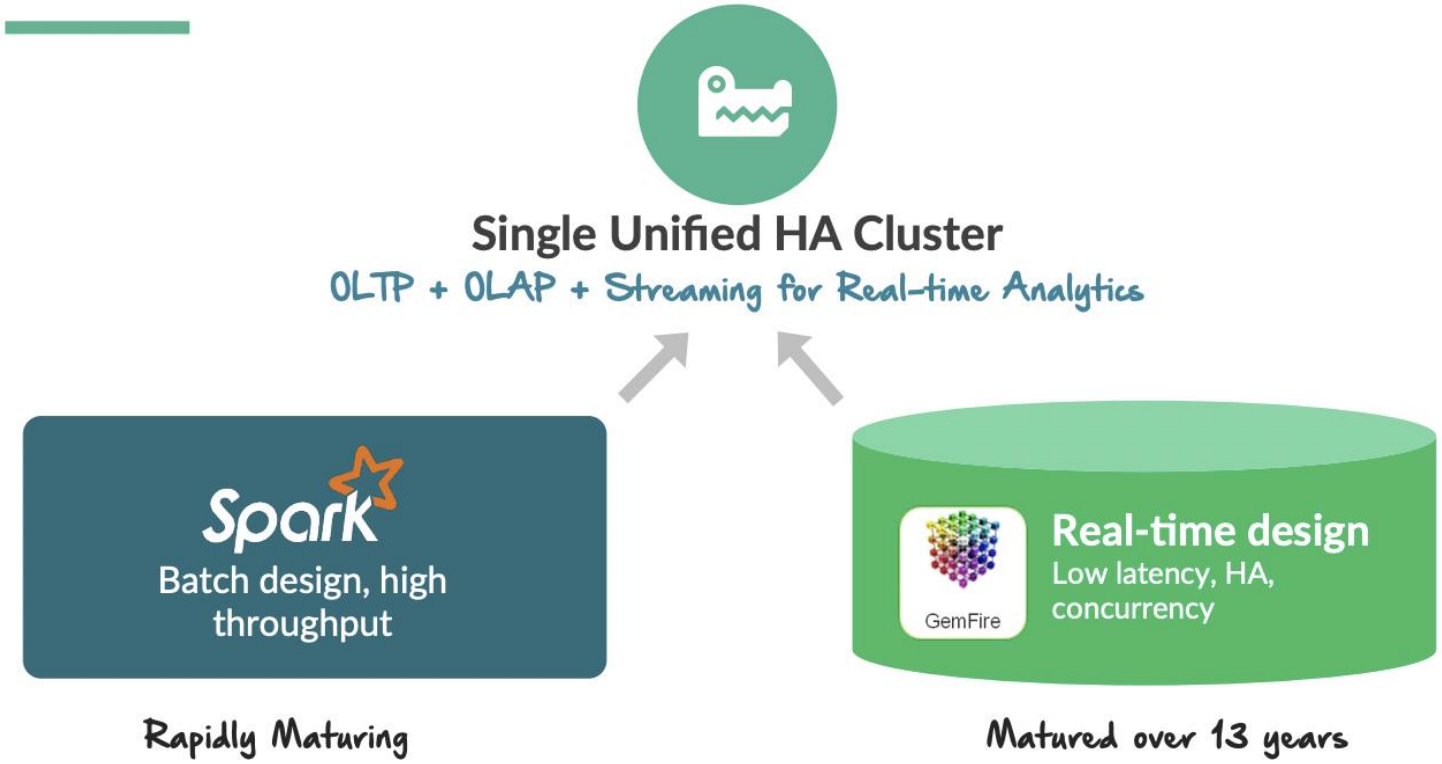


Source: https://www.cidrdb.org/cidr2017/slides/p28-mozafari-cidr17-slides.pdf

Lambda Architecture Contd.

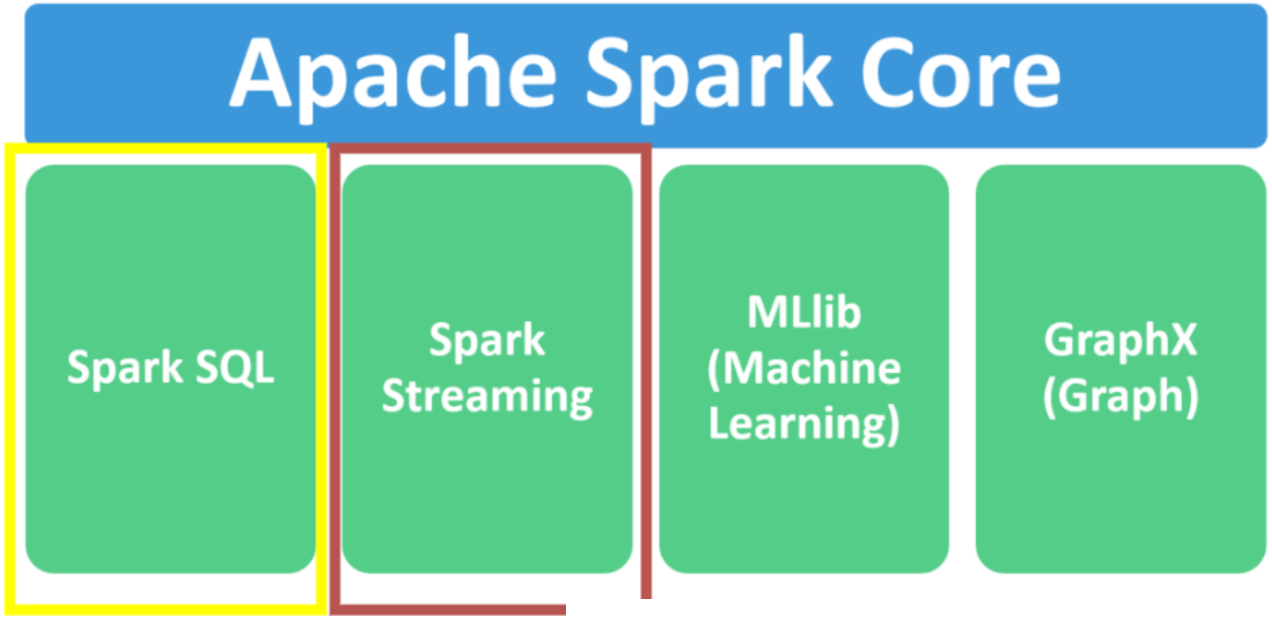


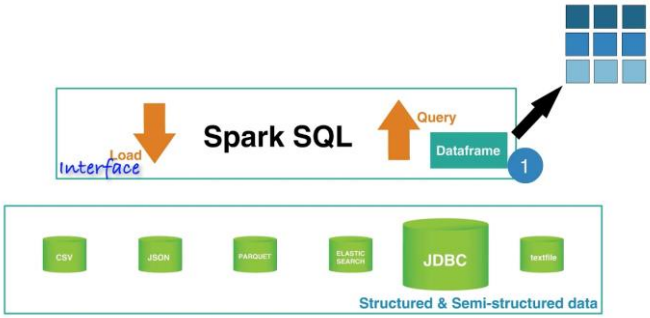
SnappyData Architecture



Source: https://www.cidrdb.org/cidr2017/slides/p28-mozafari-cidr17-slides.pdf

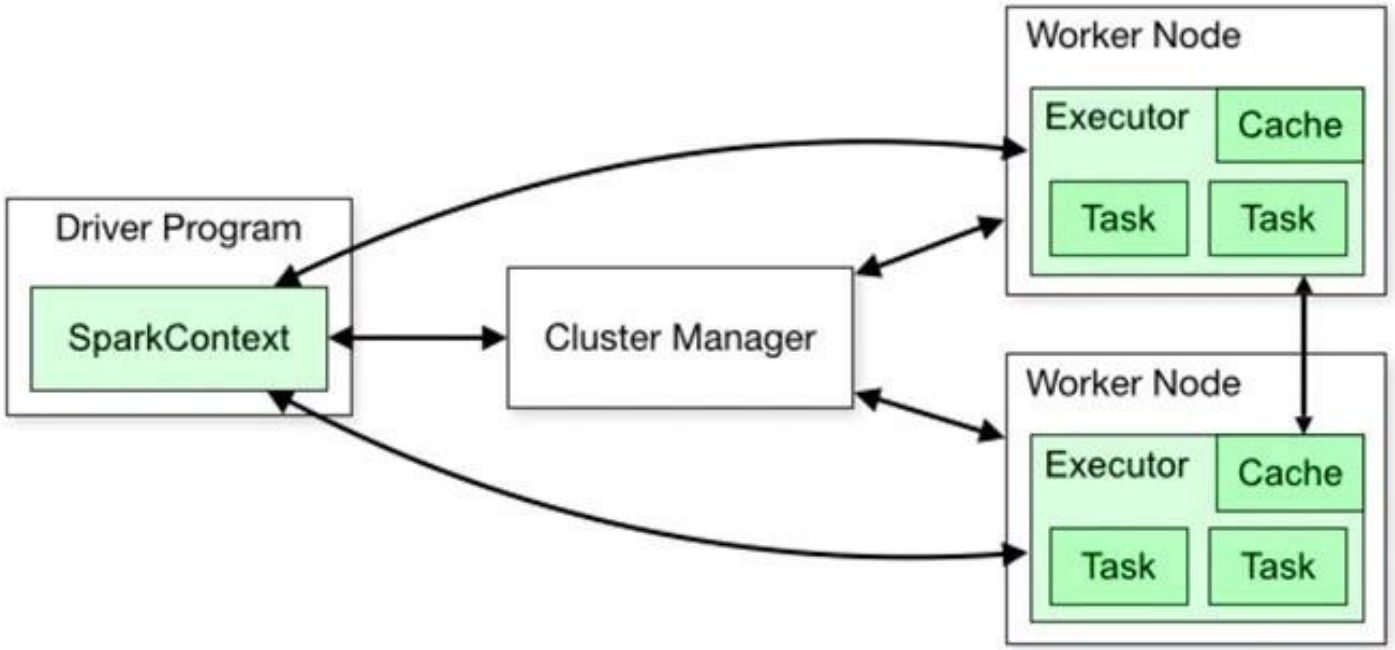
Spark





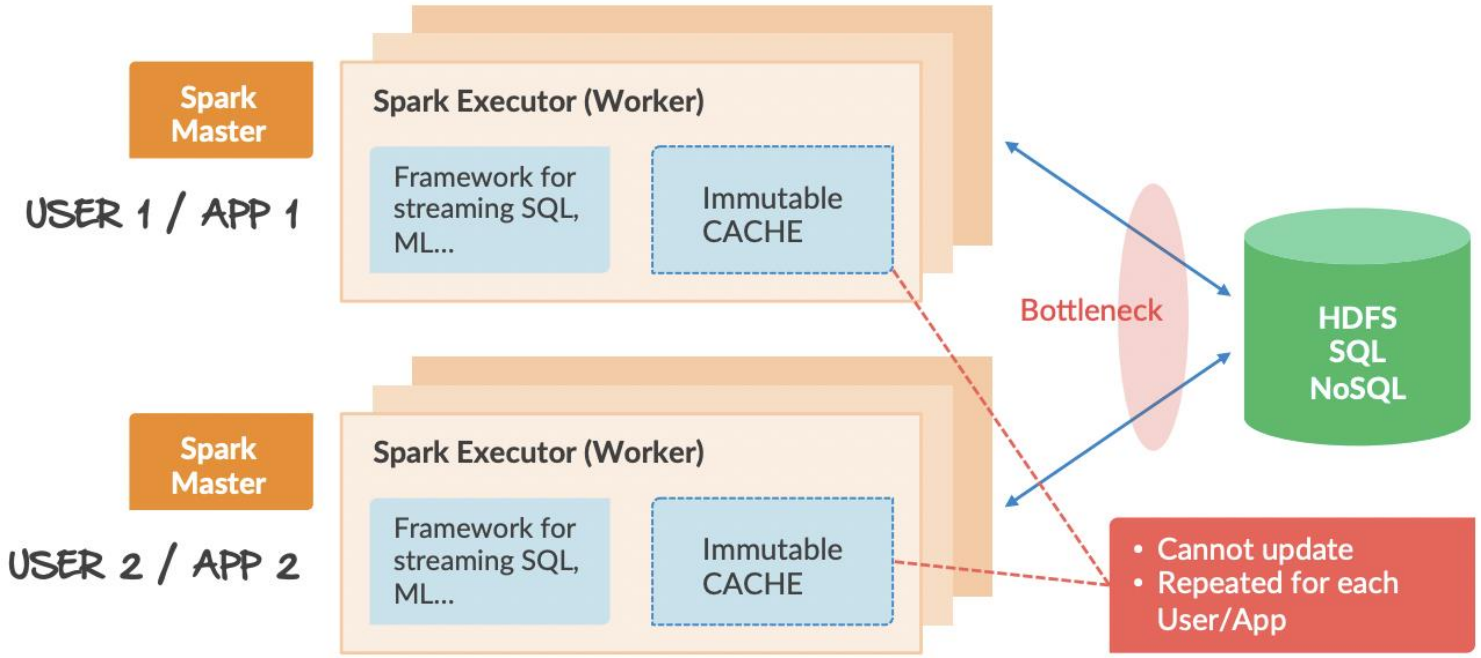
Source: https://www.youtube.com/watch?v=M40WQzjGePw&ab_channel=BigDataElearning

Spark Architecture



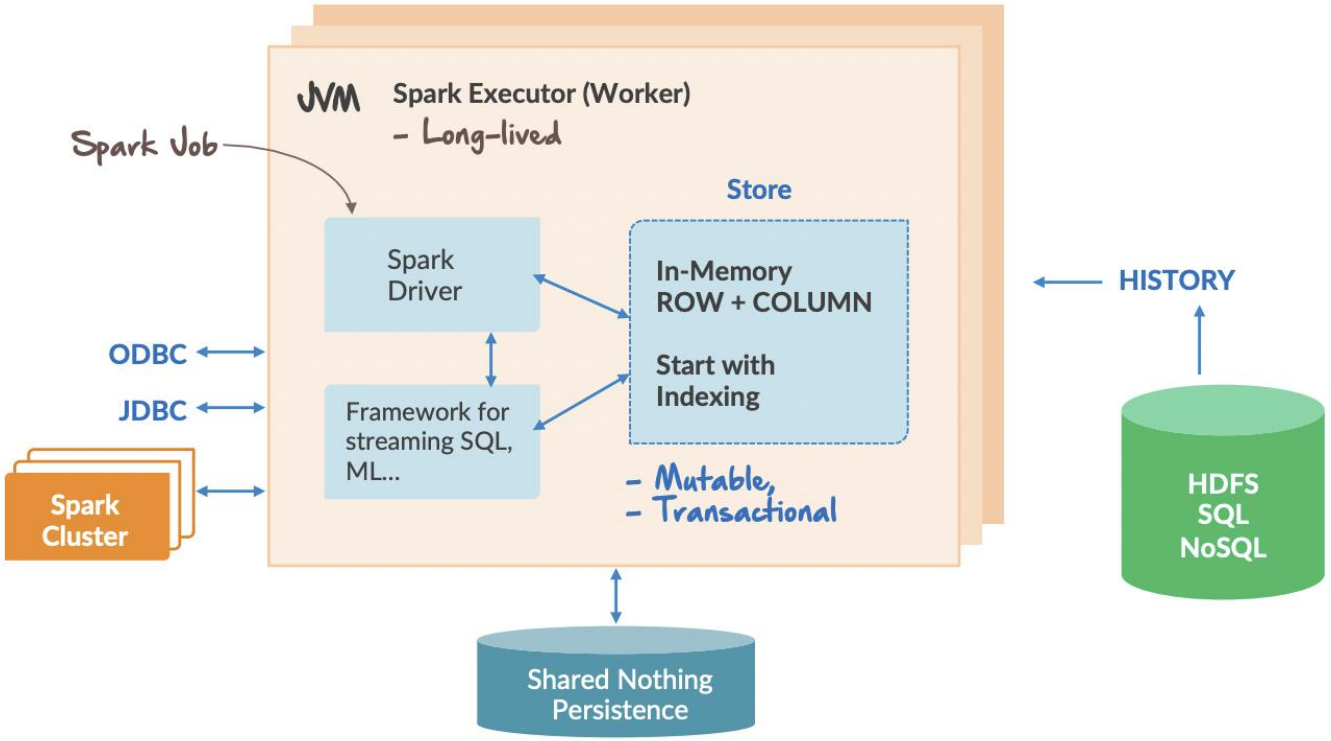
Source: https://www.analyticsvidhya.com/blog/2020/11/data-engineering-for-beginners-get-acquainted-with-the-spark-architecture/

Traditional Spark



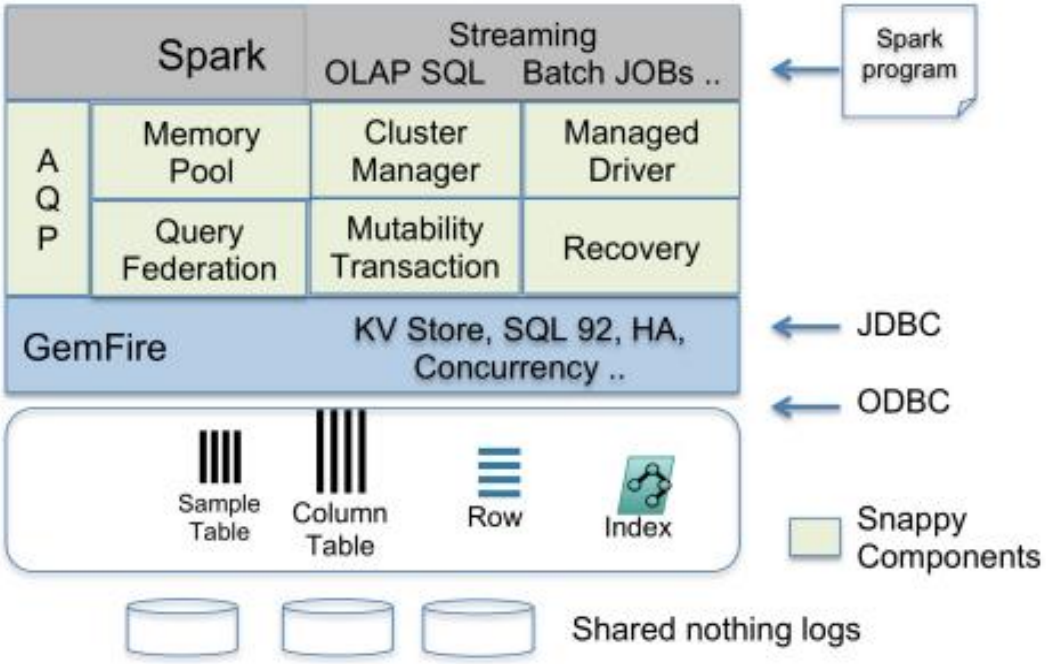
Source: https://www.cidrdb.org/cidr2017/slides/p28-mozafari-cidr17-slides.pdf

Spark Architecture



Source: https://www.cidrdb.org/cidr2017/slides/p28-mozafari-cidr17-slides.pdf

SnappyData



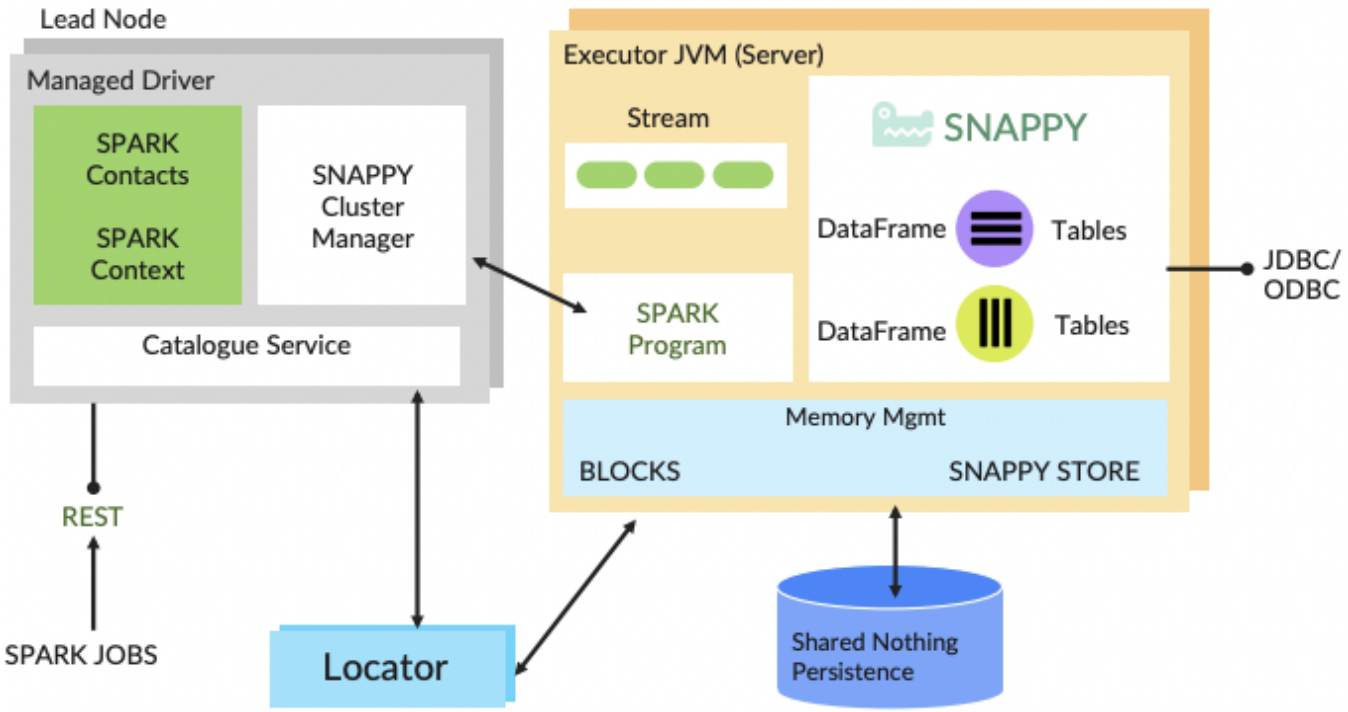

```
1  // Create a SnappyContext from a SparkContext
2  val spContext = new org.apache.spark.SparkContext(conf)
3  val snpContext = org.apache.spark.sql. SnappyContext (
       spContext)
4
5  // Create a column table using SQL
6  snpContext.sql("CREATE TABLE MyTable (id int, data string)
       using column")
7
8  // Append contents of a DataFrame into the table
9  someDataDF.write.insertInto("MyTable");
10
11 // Access the table as a DataFrame
12 val myDataFrame: DataFrame = snpContext.table("MyTable")
13 println(s"Number of rows in MyTable = ${myDataFrame.count()
       }")
```

Listing 1: Working with DataFrames in SnappyData

PURDUE UNIVERSITY

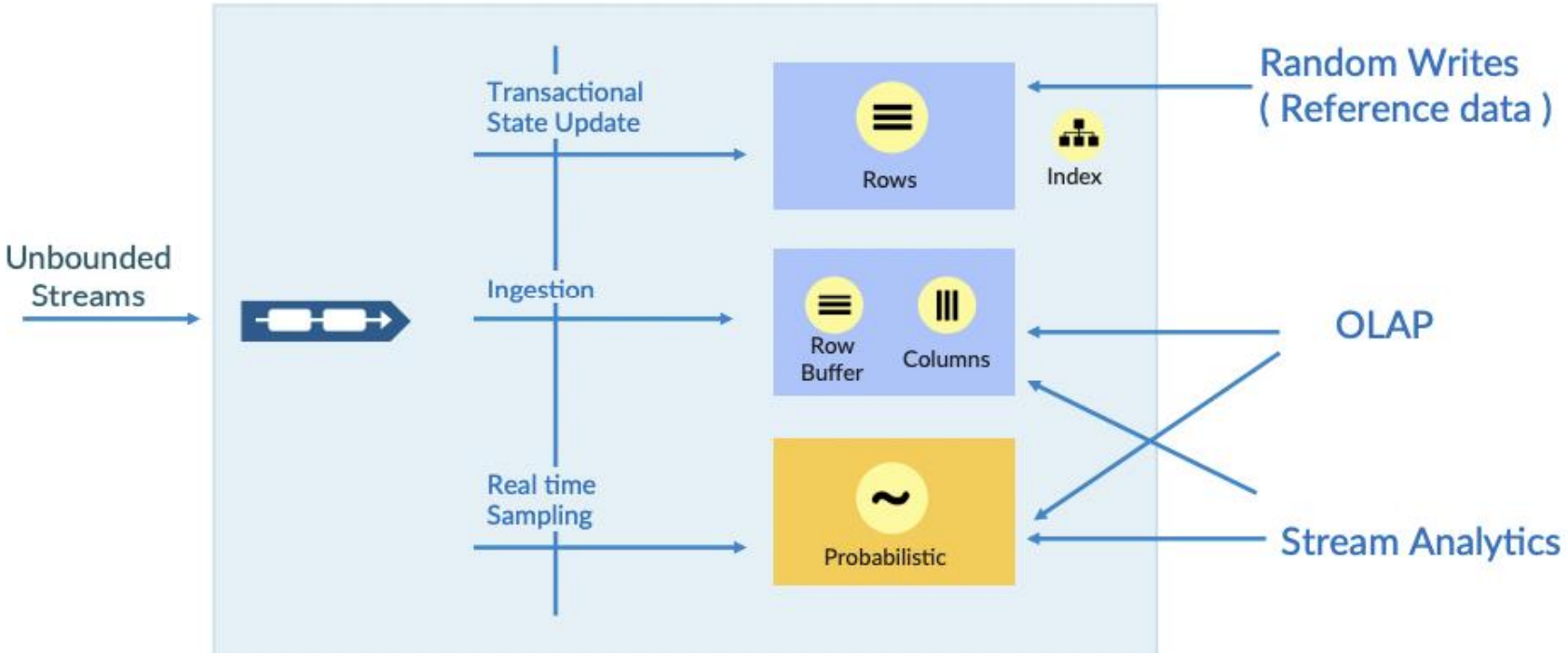
Source: https://www.cidrdb.org/cidr2017/slides/p28-mozafari-cidr17-slides.pdf

Catalogue Service



Source: https://www.cidrdb.org/cidr2017/slides/p28-mozafari-cidr17-slides.pdf

Data Models



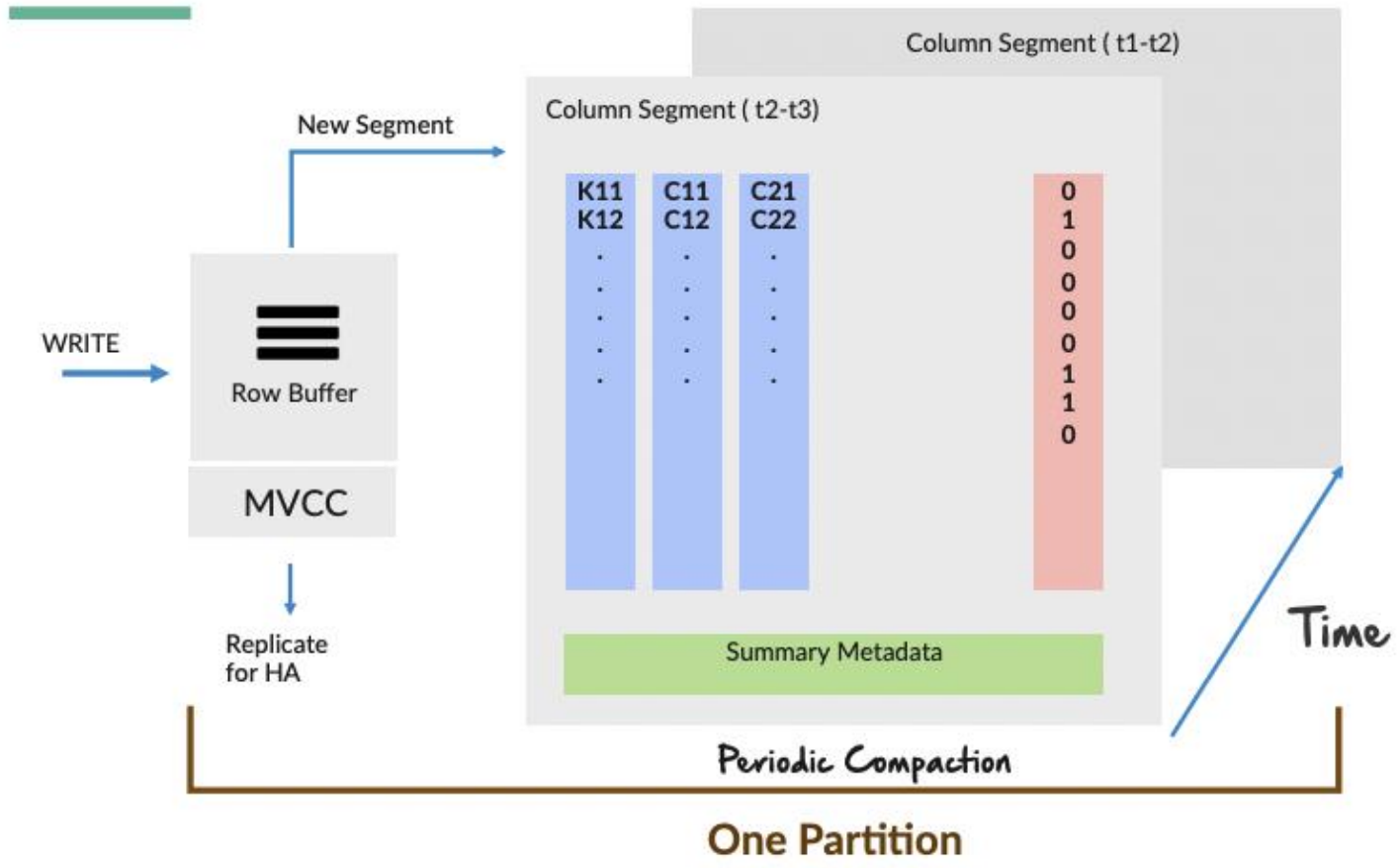
Source: https://www.cidrdb.org/cidr2017/slides/p28-mozafari-cidr17-slides.pdf

Query Processing

- SQL queries are federated between Spark's Catalyst and GemFire's OLTP engine
- An initial query plan determines if the query is a low latency operation (e.g., a key-based lookup) or a high latency one (scans/aggregations).
- SnappyData avoids scheduling overheads for OLTP operations by immediately routing them to appropriate data partitions
- Transactions follow a 2-phase commit protocol using GemFire's Paxos implementation to ensure consensus and view consistency across the cluster.

Source: https://www.cidrdb.org/cidr2017/papers/p28-mozafari-cidr17.pdf

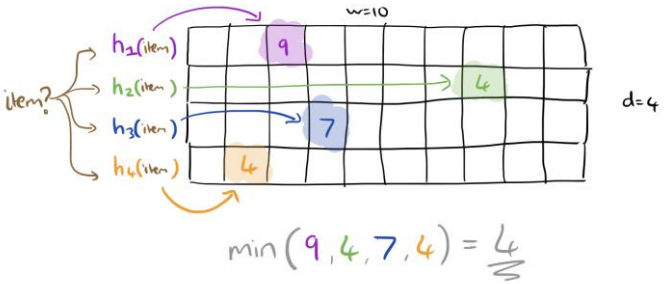
Column Store Updates

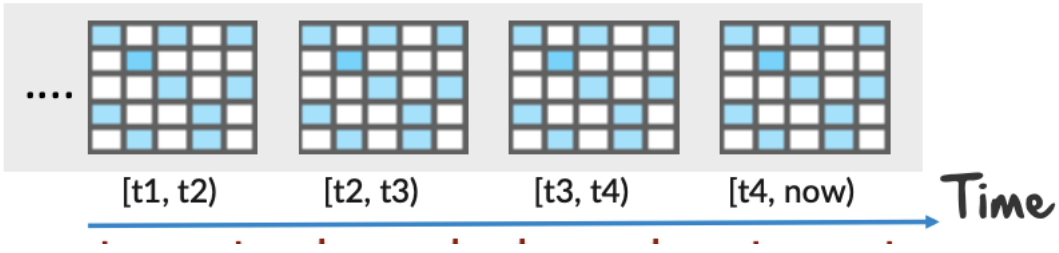


Source: https://www.cidrdb.org/cidr2017/slides/p28-mozafari-cidr17-slides.pdf

Probabilistic Data Store

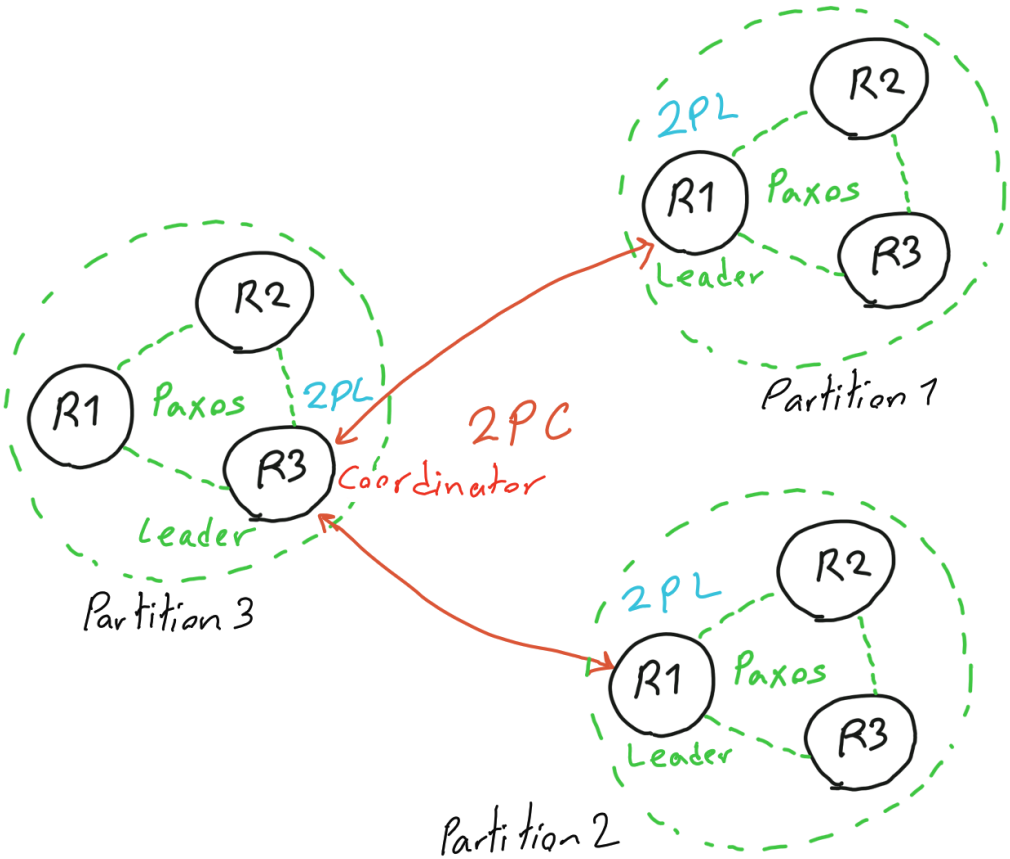
- SnappyData uses count min sketch data structure to empower AQP
- The columns on which sampling can be enabled is either Join attributes(foreign keys) or they can determined at the run time by a tool **WorkloadMiner**
- Snappy Data maintains samples separately for every t units of time





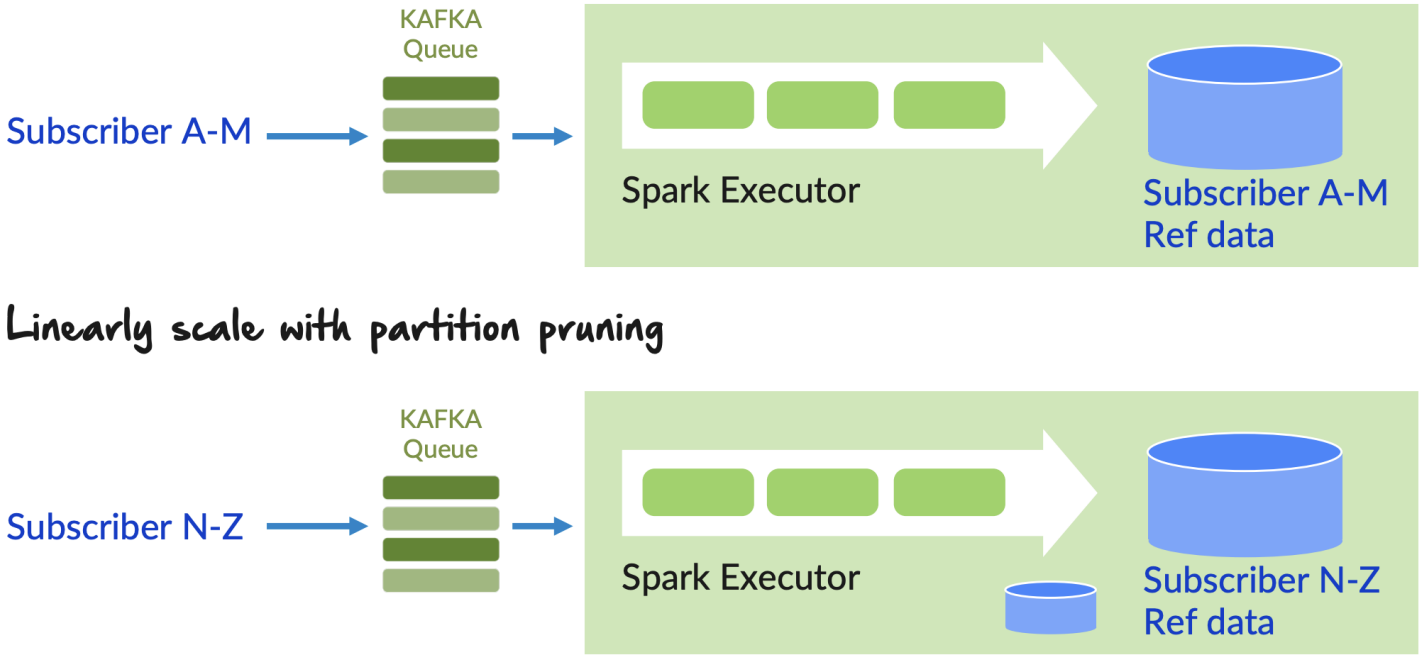
Source: https://www.cidrdb.org/cidr2017/slides/p28-mozafari-cidr17-slides.pdf

Scalability



Source: https://www.mydistributed.systems/2020/08/google-spanner.html

Co-Partition and Co-Location



Source: https://www.cidrdb.org/cidr2017/slides/p28-mozafari-cidr17-slides.pdf

Availability and Fault Tolerance

- Driver Node has a secondary as a backup, whenever the primary fails secondary takes over.
  - This ensures the Client will always be able to send requests
- For the Executor Nodes, the data is replicated through Paxos at Gemfire level.
  - This ensure n node failure tolerance semantics (Given that 2*n + 1 nodes for the Paxos group)

VOLTDB

Disagg1

Presented by Rwitam Bandyopadhyay

PURDUE UNIVERSITY

Interesting DB that challenges traditional RDBMS

Iterative development over multiple papers

- OLTP Through the Looking Glass, and What We Found There
- The End of an Architectural Era (It's Time for a Complete Rewrite)
- The VoltDB Main Memory DBMS

All these papers are by Michael Stonebraker et. al. who is also one of the founders of VoltDB.

Shift towards main memory databases

Historically declining prices of DRAM

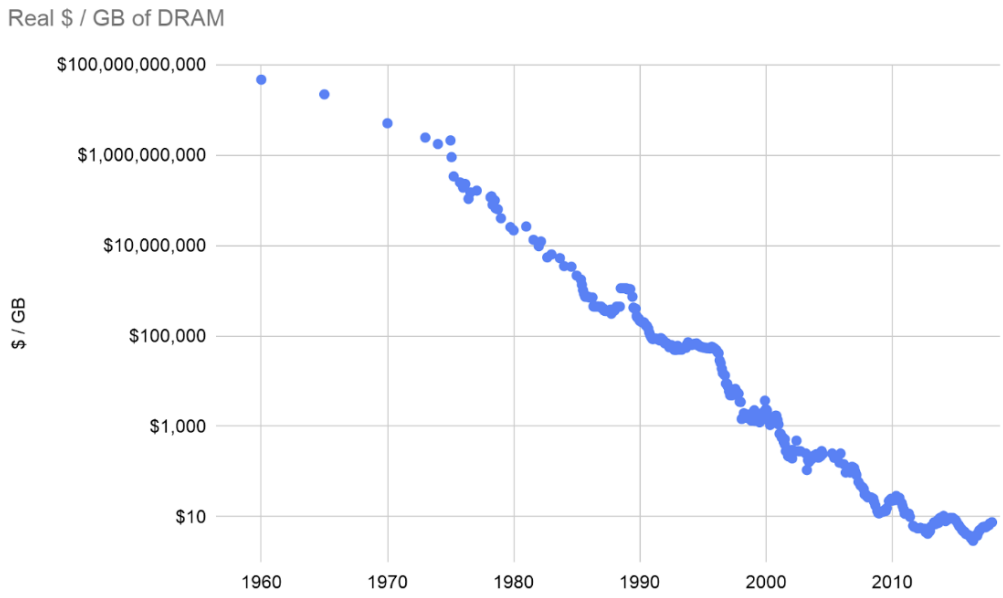
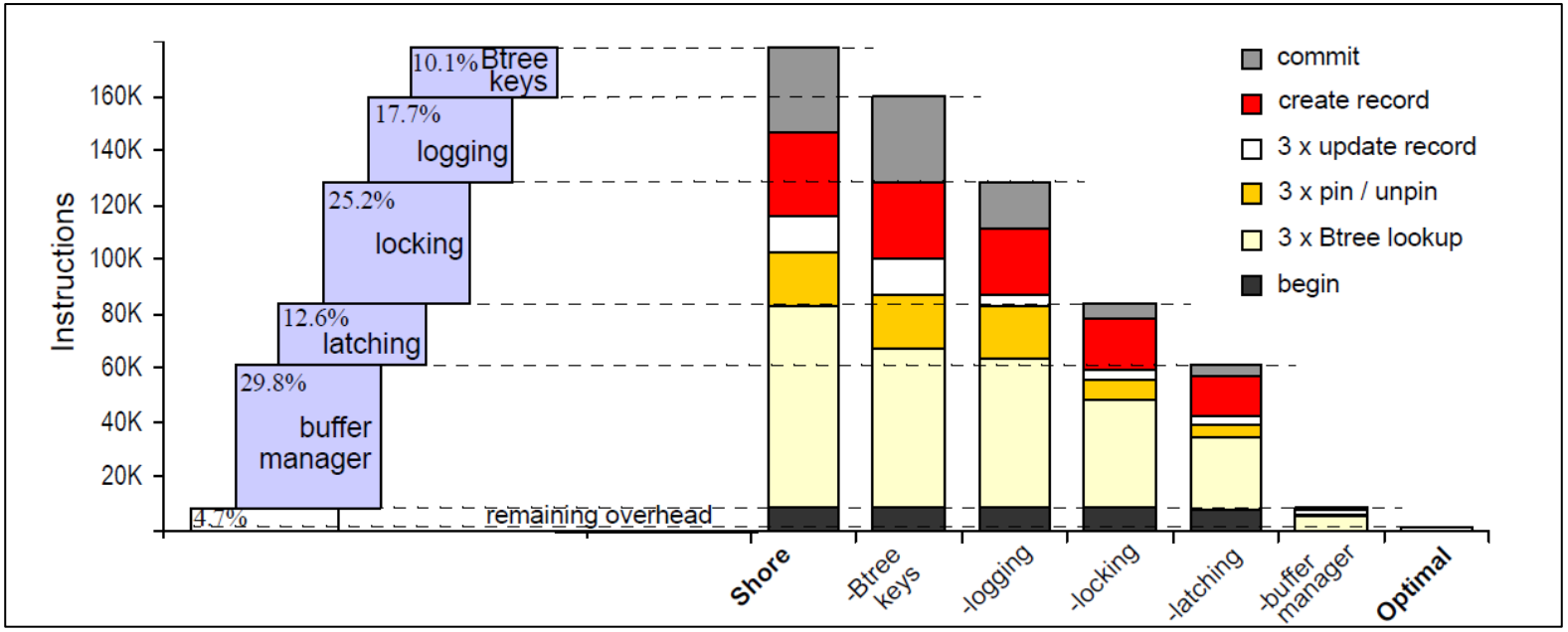


Figure 1: Price per gigabyte of DRAM from 1957 to 2018 from John McCallum's dataset, which we converted to 2020 dollars using the Consumer Price Index.[7]

Source: AI Impacts - Trends in DRAM price per gigabyte

OLTP Overheads

Dissecting where time goes inside of a modern DBMS



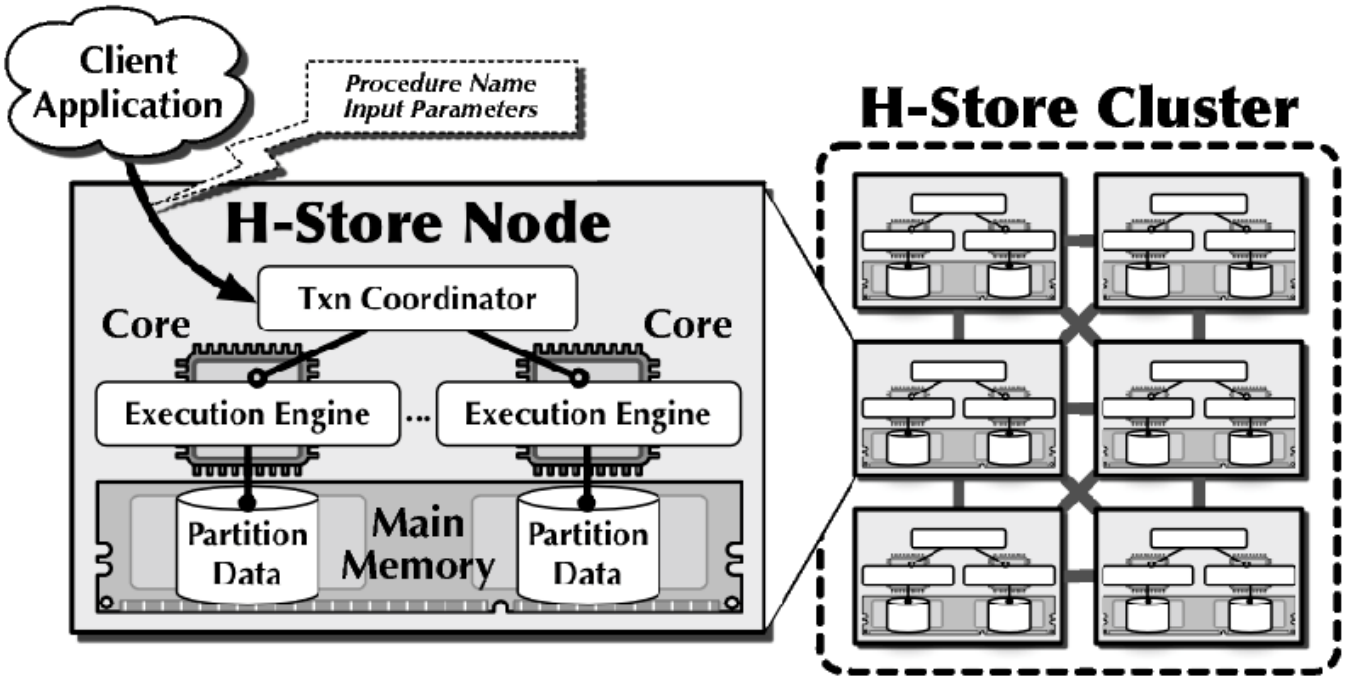
Source: Harizopoulos et al. OLTP through the looking glass, and what we found there, SIGMOD '08: Proceedings of the 2008 International Conference on Management of Data.

VoltDB Eliminates All of these Legacy DBMS Overheads

- Data/ processing are partitioned together and distributed across the CPU cores (“virtual nodes”) in a shared-nothing hardware cluster.
- Data is held in memory for maximum throughput and to eliminate the need for buffer management.
- Each single-threaded partition operates autonomously, eliminating the need for locking and latching.
- Data is automatically replicated for intra- and inter-cluster high availability

VoltDB Architecture

Borrowed majorly from H-Store (Shared Nothing)

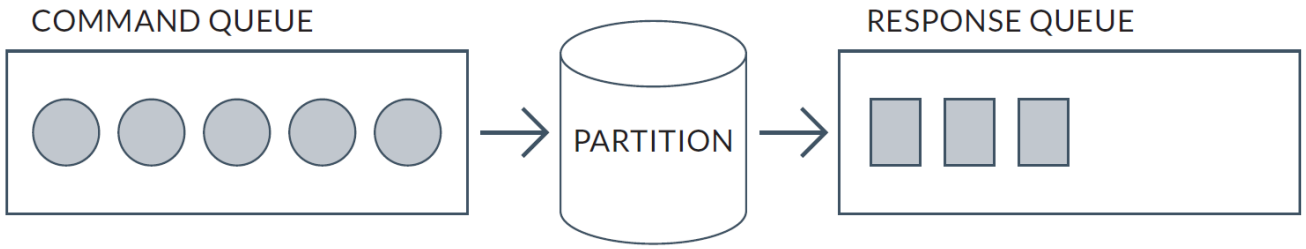


Source: https://hstore.cs.brown.edu/documentation/architecture-overview/

VoltDB Transactions

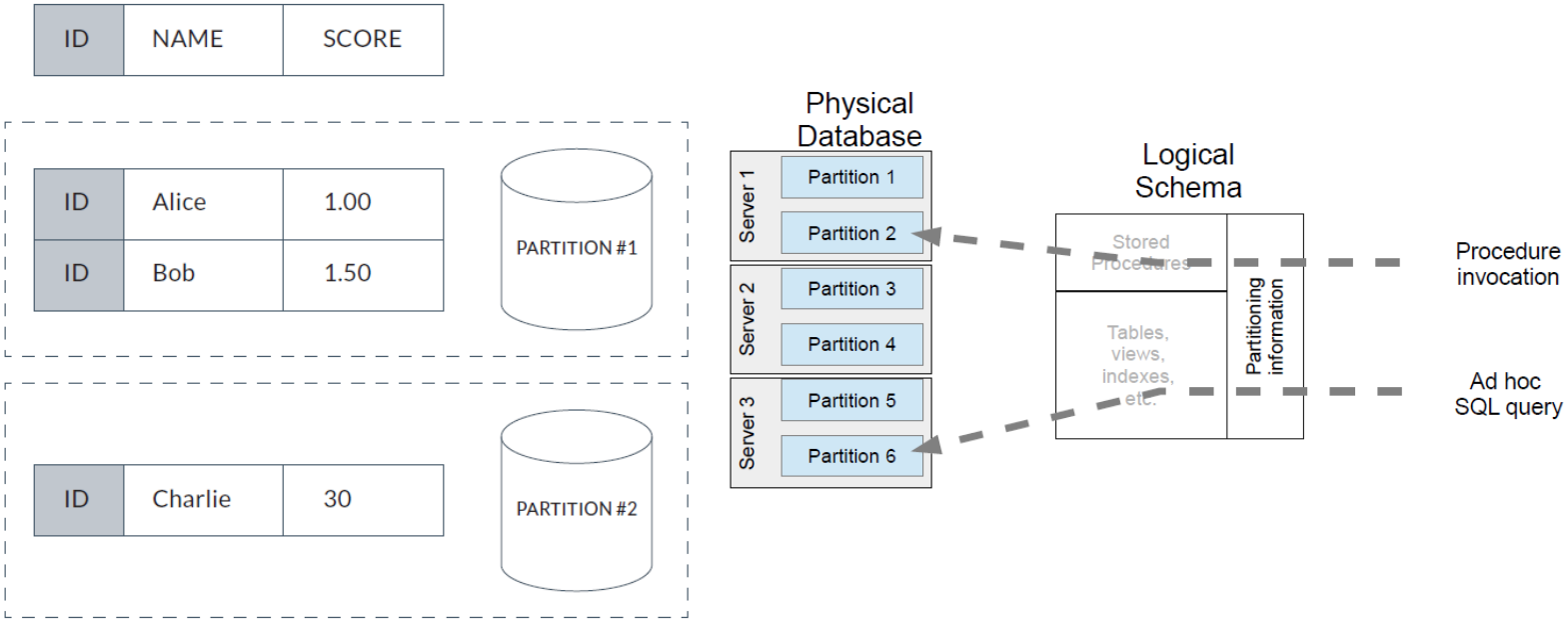
How can we achieve maximum throughput?

- VoltDB ditches parallelism and goes towards sequential operations.
- A command queue is filled up, and the CPU continuously loops through the commands and processes them, leading to 100% utilization.



Source: VoltDB Technical Note - How VoltDB does Transactions

Scaling across multiple CPUs

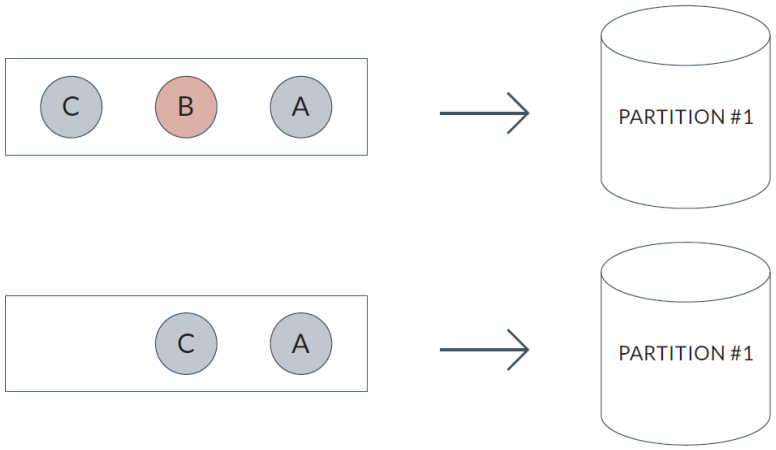


Source: VoltDB Technical Note - How VoltDB does Transactions

VoltDB Transaction Partitioning

Single Partition Transactions

- Commands are replicated to all partition replicas, so they are consistent
- Read commands can be directed to only one partition replica, since they do not change the data

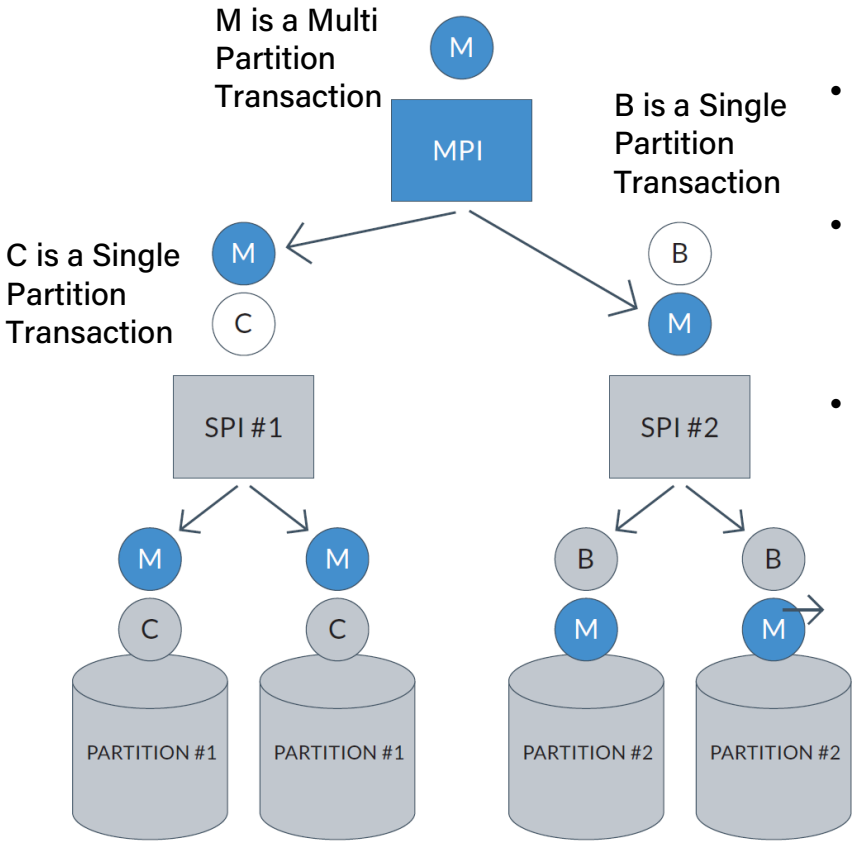


Resulting serializable timeline is A -> C

Source: VoltDB Technical Note - How VoltDB does Transactions

VoltDB Transaction Partitioning

Multiple Partition Transactions



- Multiple partition transactions have to be sent across multiple hosts/ virtual nodes.
- Every node needs to "commit" successfully for the atomicity of the transaction to be maintained.
- A 2-phase commit protocol is invoked between the partitions to ensure overall transactional consistency.

Resulting serializable timeline is C -> M -> B

Source: VoltDB Technical Note - How VoltDB does Transactions

What causes deadlocks/ delays/ waits?

- Administrative
  Waits resulting from DBA commands that cause users to wait (for example, an index rebuild)
- Application
  Waits resulting from user application code (for example, lock waits caused by row level locking or explicit lock commands)
- Commit
  This wait class only comprises one wait event - wait for redo log write confirmation after a commit (that is, 'log file sync')
- Concurrency
  Waits for internal database resources (for example, latches)
- Configuration
  Waits caused by inadequate configuration of database or instance resources (for example, undersized log file sizes, shared pool size)
- Network
  Waits related to network messaging (for example, 'SQL*Net more data to dblink')
- System I/O
  Waits for background process I/O (for example, DBWR wait for 'db file parallel write')
- User I/O
  Waits for user I/O (for example 'db file sequential read')

Source: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/classes-of-wait-events.html

ACID Guarantees (Continued)

VoltDB circumvents most of the “waits”

- We have no buffer/ log waits.
- We have no need for locks because there are no concurrent transactions operating on a single row.
- Traditionally, concurrent transactions were implemented to utilize the CPU more during IO waits.
- Since VoltDB operates on main memory, there is almost no IO wait, hence we can give up on parallelism and do purely serial operations.
- There are developer good practices to utilize VoltDB to it’s true potential. (<10ms transactions, stored procedures transactions only etc.)

Source: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/classes-of-wait-events.html

ACID Guarantees (Continued)

Types of OLTP Operations

- (**Vast majority**) Single-node transactions - In this case, there is a database design that allocates the shards of each table to nodes in such a way that most transactions are local to a single node. For eg. order placements, votes etc.

- (**Lesser number**) One-shot transactions - For example, moving $X from account Y to account Z (without checking for an overdrawn condition) is a one-shot that will be executed at two nodes, one holding the shard of Y and the other the shard containing Z.

- (**Even less**) General transactions - These are transactions that consist of multiple phases where each phase must be completed before the next phase begins.

Source: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/classes-of-wait-events.html

VoltDB Transaction Summary

- VoltDB executes commands to query and change data serially at a partition.
- Commands can be ad hoc SQL, stored procedures, or SQL plan fragments.
- Each command is run with strict ACID semantics.
- Table data is distributed across partitions.
- Execution is scaled across CPU cores (and servers) by adding partitions.
- Performs really well if your business use-case is aligned to the adapted case of VoltDB.
- Performs as good, if not better, when compared to traditional RDBMS if you have general purpose, multi-partition, cross-network transactions.

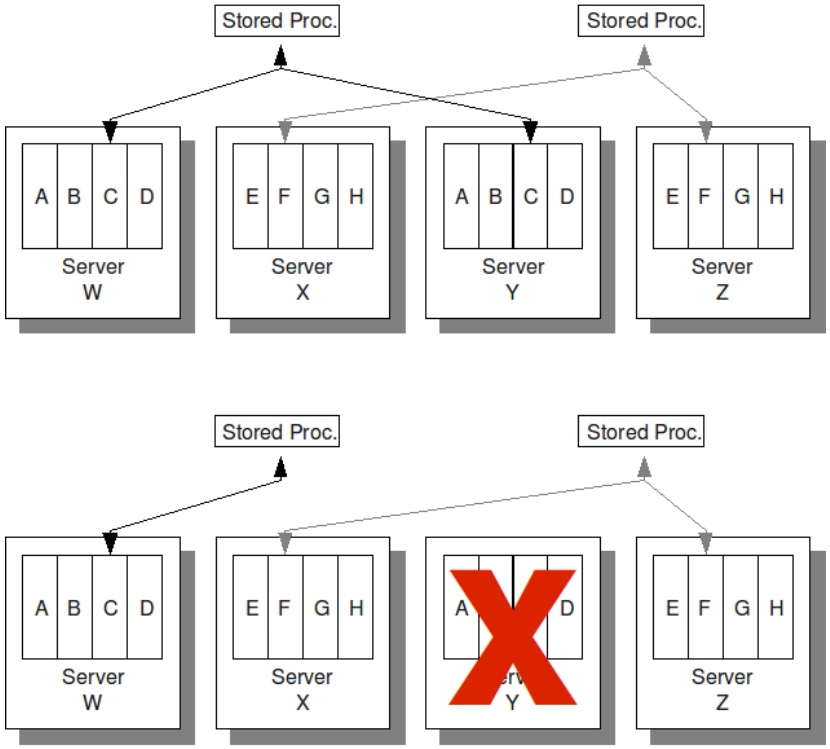
Source: VoltDB Technical Note - How VoltDB does Transactions

VoltDB Availability

Data durability?

- Snapshots
- Command Logging (~WAL)
- K-Safety
- DB Replication

K-Safety in Action



Source: Using VoltDB - https://docs.voltdb.com/UsingVoltDB/

Conclusion

VoltDB – HTAP?

- VoltDB provides a scalable, fault-tolerant, serializable, fully durable, high throughput transaction-processing model.
- As a side-effect of being a complete in-memory database, we get good OLAP performance too. The only real latency would be network latency resulting from multiple hosts.
- Clusters/grids in datacenters theoretically have a very high overall main-memory capacity, so OLAP performance would be bottlenecked by total available memory.
- To support aggregations in very selective columns, we can add an additional column store for user-defined columns.

DB2 BLU

Disagg1

Presented by Shubham Pandey

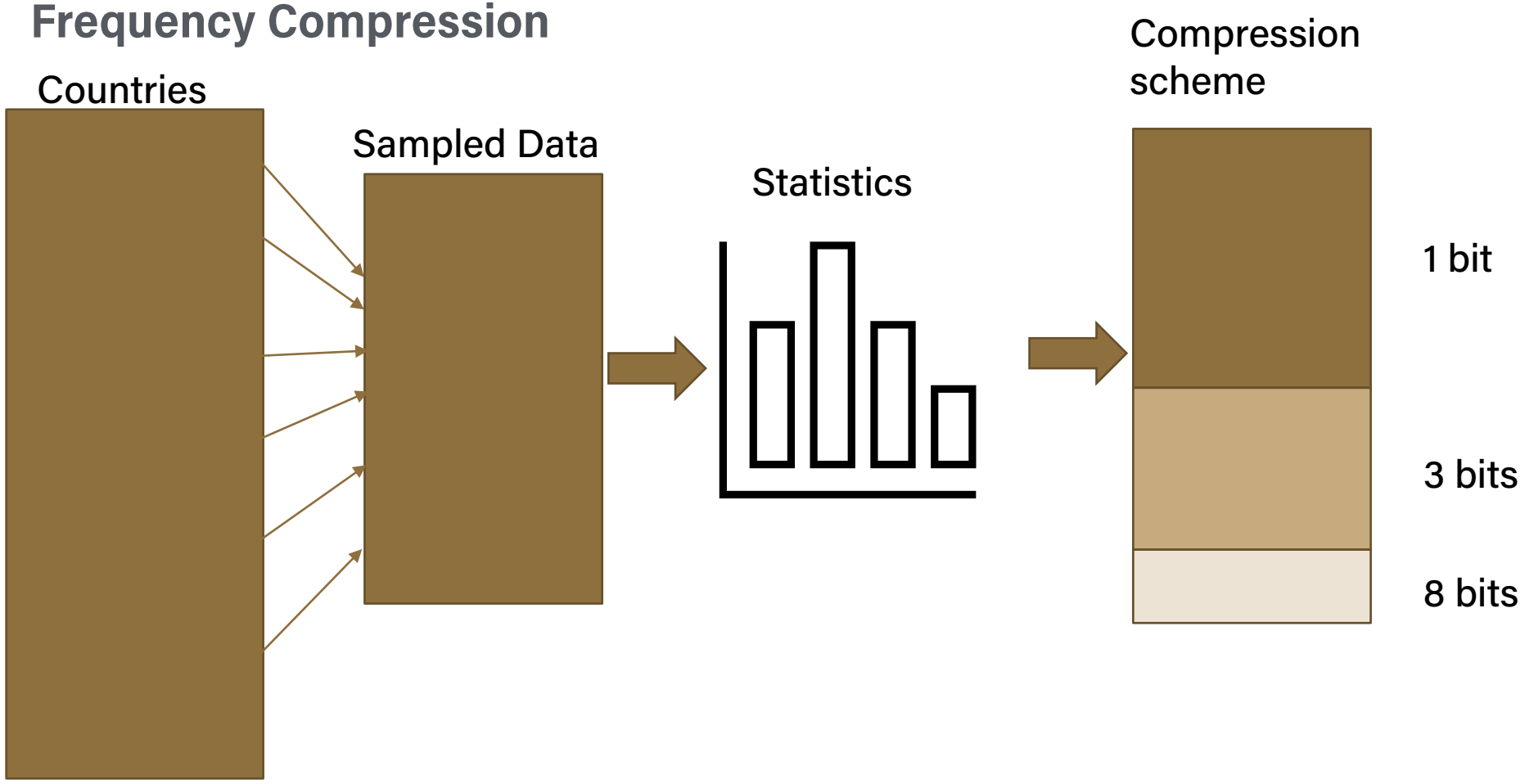
PURDUE UNIVERSITY

Introduction

- Accelerate Business Intelligence queries in columnar storage (10-50x)
- Improve compression (3-10x)
- No reliance on indexes and materialized views
- Perform predicate evaluation, join and grouping on compressed values
- Process multiple records using SIMD instructions

Source: https://virtual-dba.com/platforms/ibm-db2-luw-old/db2-blu/
https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

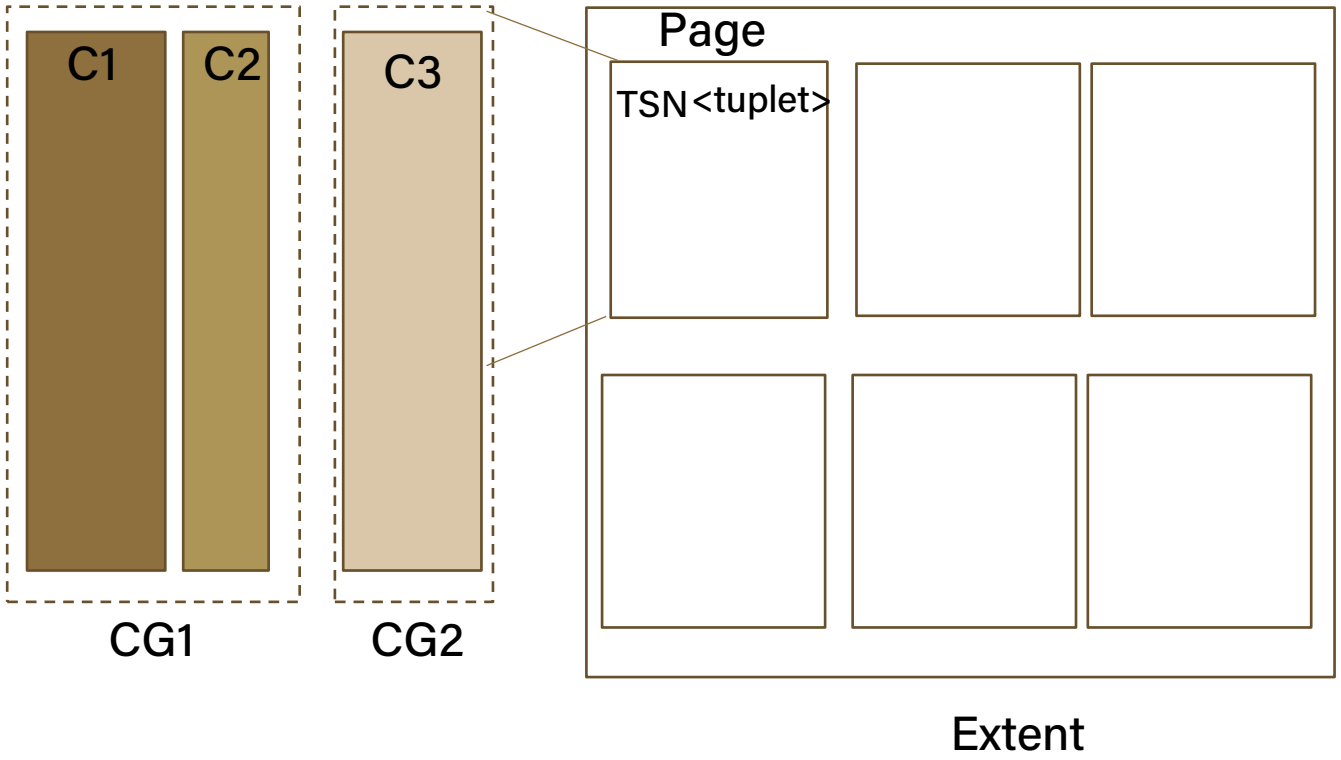
Frequency Compression



Source: https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

Data Layout and Compression

Column Group Storage



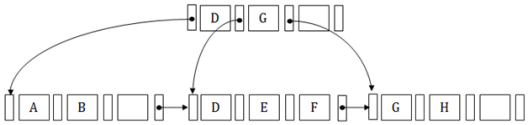
Page Header

Start TSN

Tuplet Count

Page Map

Page ID -> Start TSN, CG

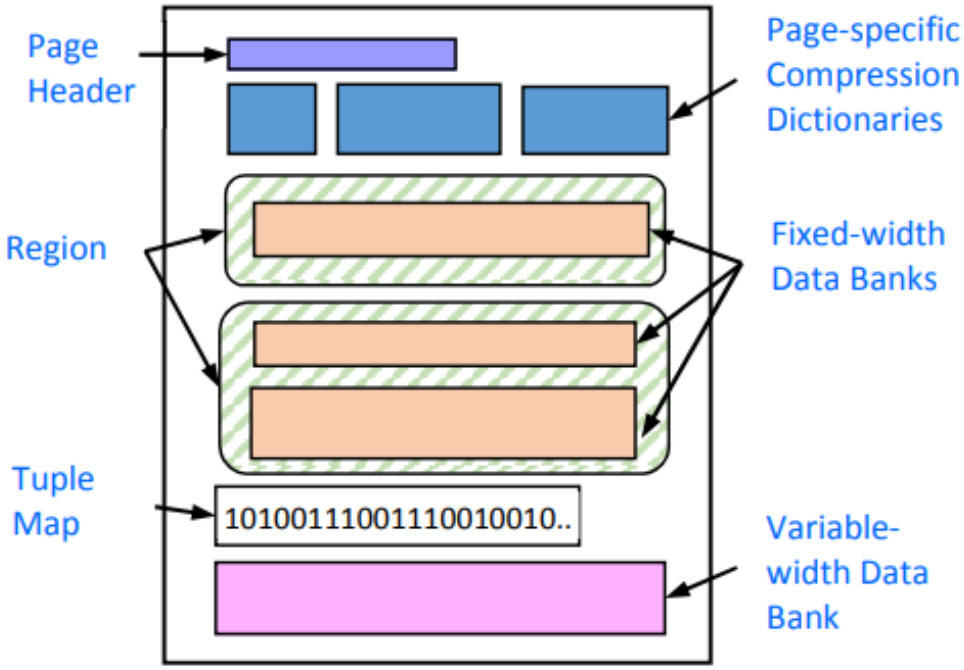


B+ Tree

Source: https://www.javatpoint.com/dbms-b-plus-tree

Data Layout and Compression

Page Format



Source: https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

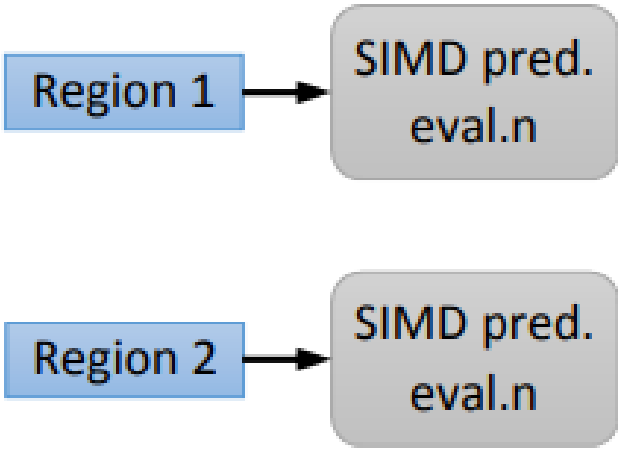
Scans and Predicate Evaluation

- Leaf Predicate Evaluator (LEAF)
- Load Column Evaluator (LCOL)

Source: https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

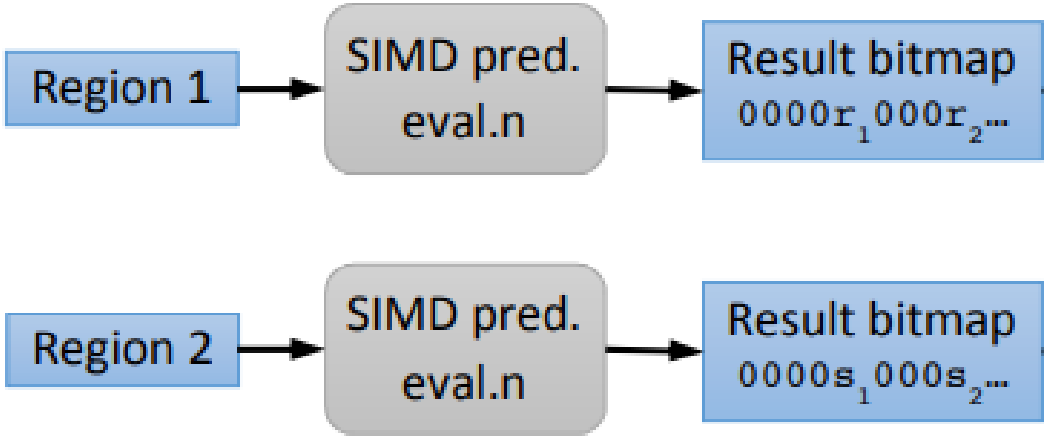
Scans and Predicate Evaluation

Leaf Predicate Evaluator (LEAF)



Source: https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

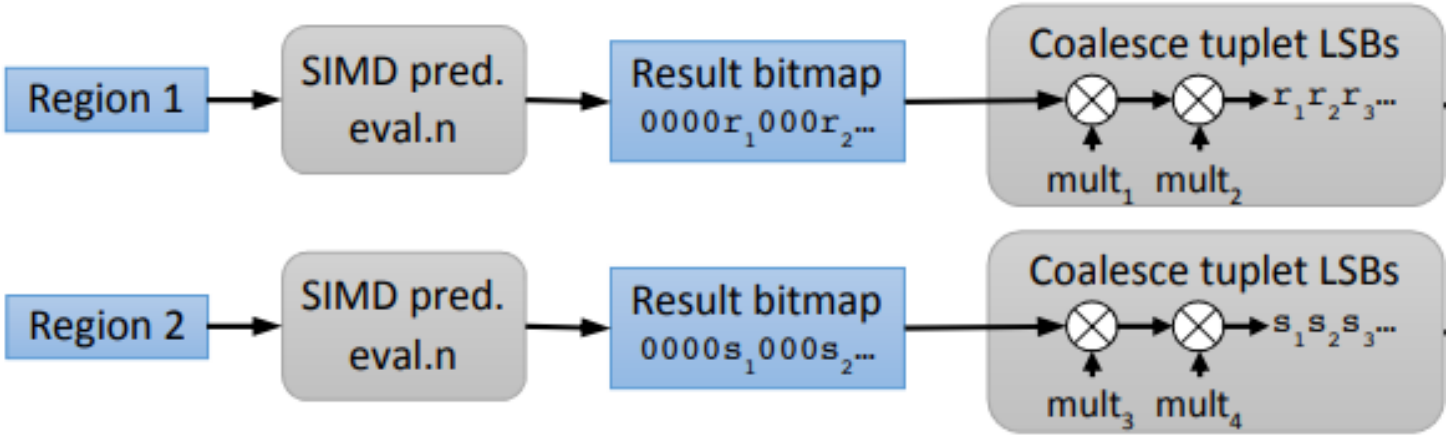
Leaf Predicate Evaluator (LEAF)



Source: https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

Scans and Predicate Evaluation

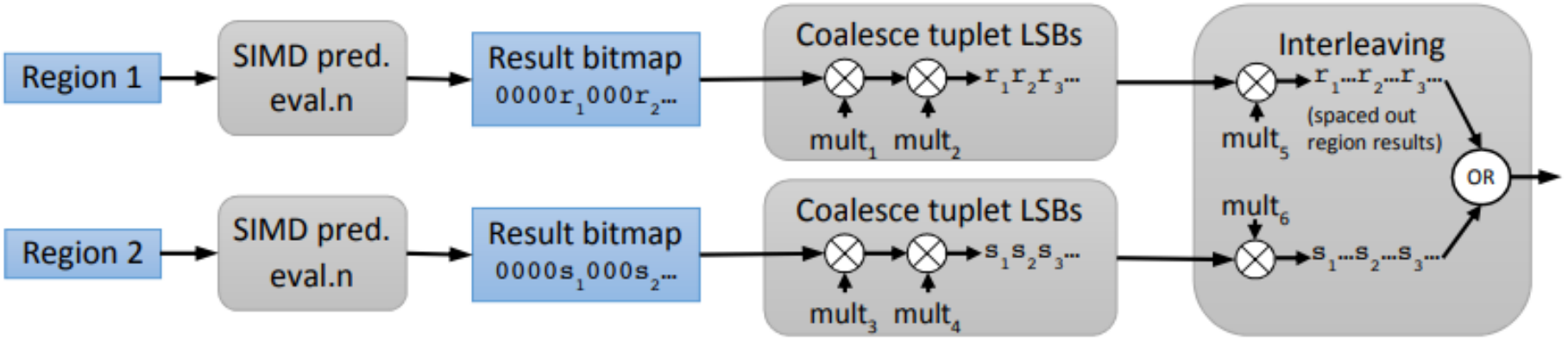
Leaf Predicate Evaluator (LEAF)



Source: https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

Scans and Predicate Evaluation

Leaf Predicate Evaluator (LEAF)



Source: https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

Scans and Predicate Evaluation

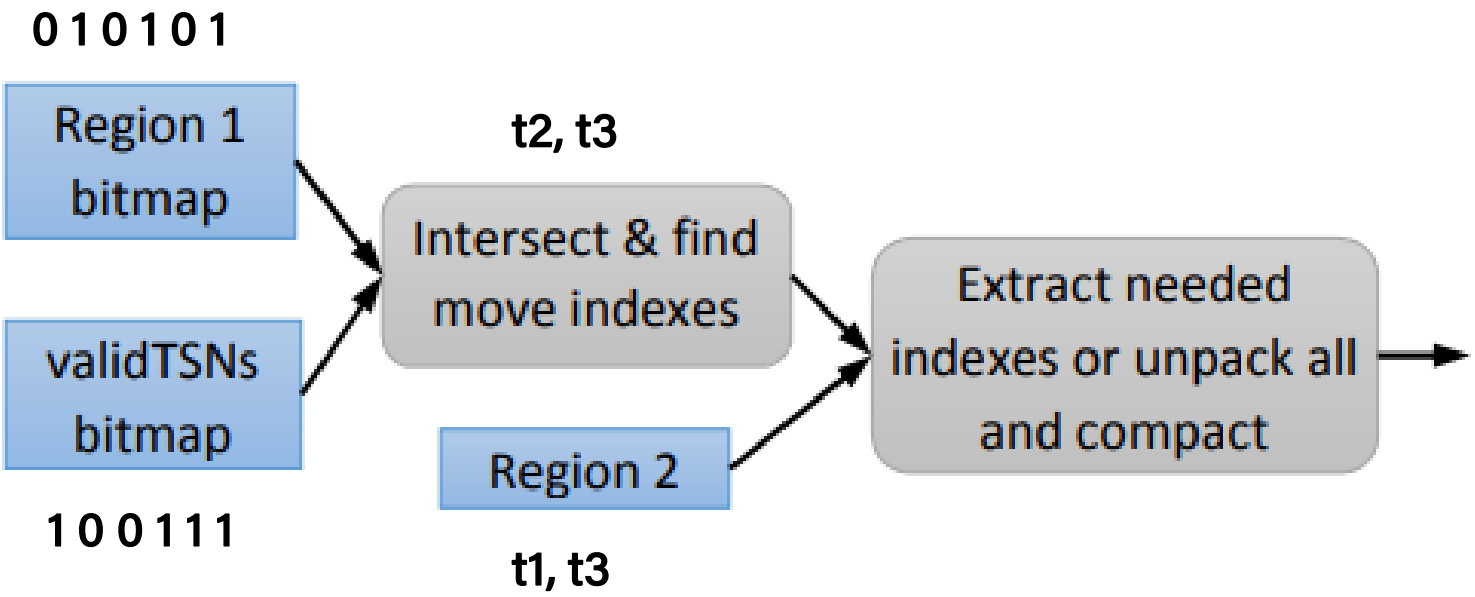
Load Column Evaluator (LCOL)

- Data Access Operator
- Output is Compacted
- Works on 1 region at a time

Source: https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

Scans and Predicate Evaluation

Load Column Evaluator (LCOL)



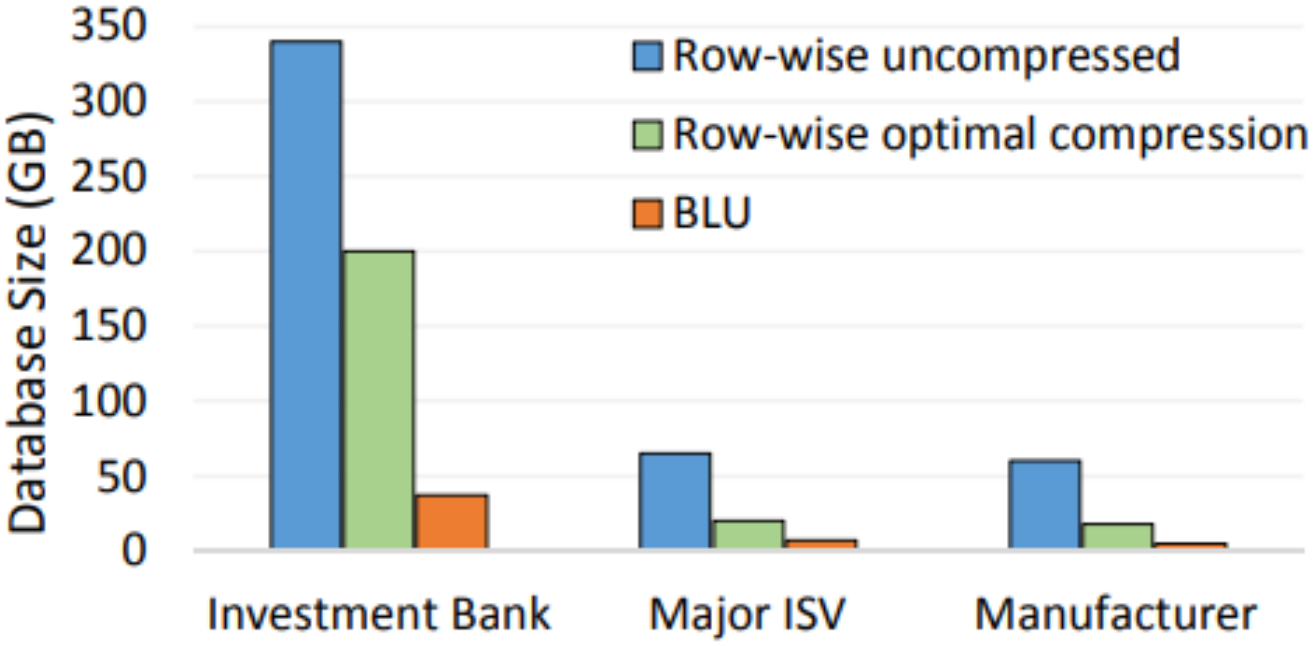
Source: https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

Performance Improvements

Workload	Speedup with DB2 BLU
Analytic ISV	37.4x
Large European Bank	21.8x
BI Vendor (reporting)	124x
BI Vendor (aggregate)	6.1x
Food manufacturer	9.2x
Investment Bank	36.9x

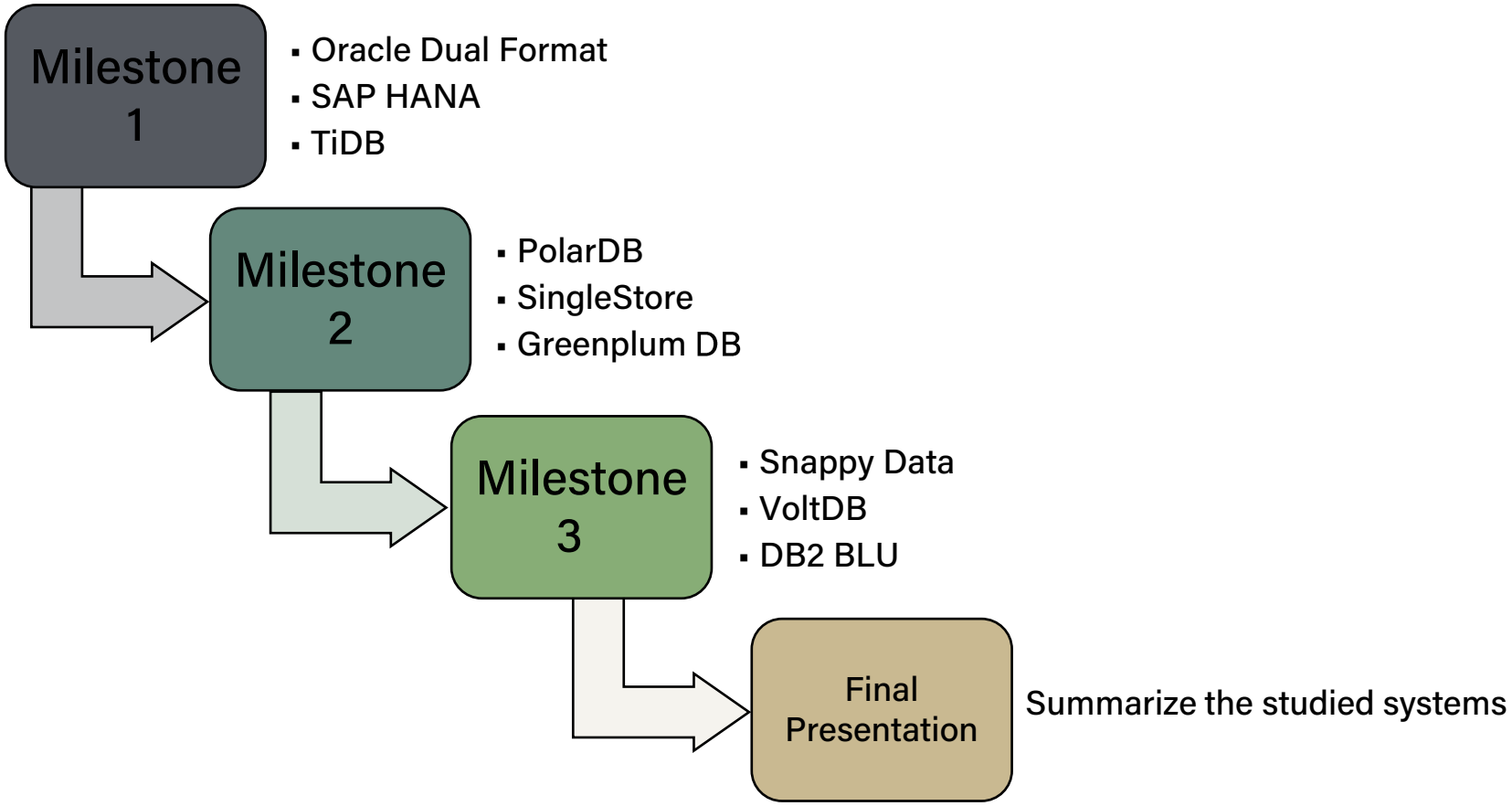
Source: https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

Storage Costs



Source: https://www.vldb.org/pvldb/vol6/p1080-barber.pdf

Project Summary



THANK YOU

PURDUE UNIVERSITY®