The Case for HTAP

Hybrid Transaction Analytical Processing

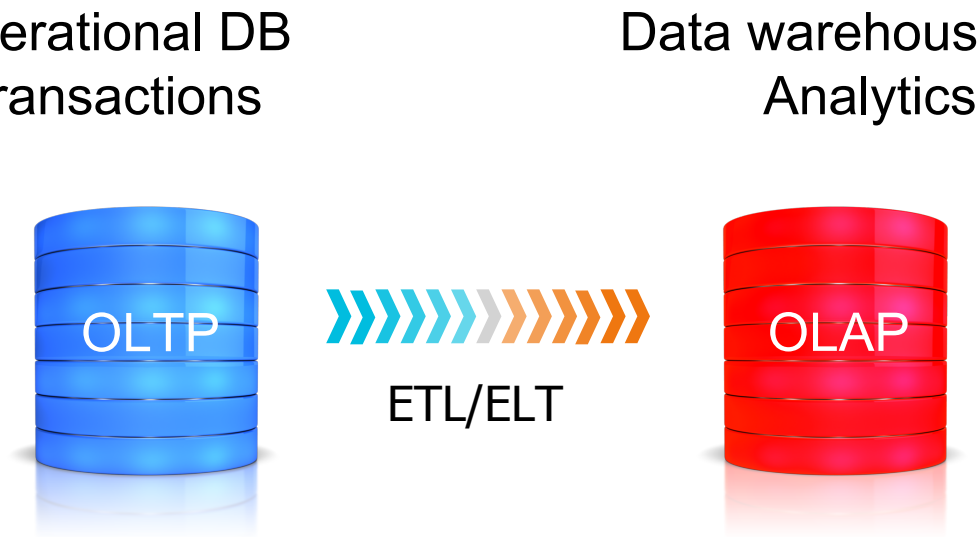
Patrick Valduriez

Outline

- Motivations
- HTAP
- LeanXcale
- Parallel polystore query processing
- Research directions

Motivations

Transaction vs. Analytical Processing



- Problems
  - ETL/ELT development cost up to 75% of analytics
  - Analytical queries on obsolete data
    - Leads to miss business opportunities, e.g., proximity marketing, real-time pricing, risk monitoring, etc.

Case Study: Banking

- Data lakes to store historical data
  - Data from mobile devices and the web coming with very high peaks
  - Use of ML to build predictive models over the historic data
  - Data copied from the data lake into GPU-based clusters to perform ML
- Problems
  - During data loading, ML processes must be paused to avoid observing inconsistent data and thus hurting the ML models that are being built
  - The ETL process may die without being noticed
    - Yields wrong ML models and a lot of effort to trace back what was the problem
  - Real-time analytics (e.g. real-time marketing) not possible

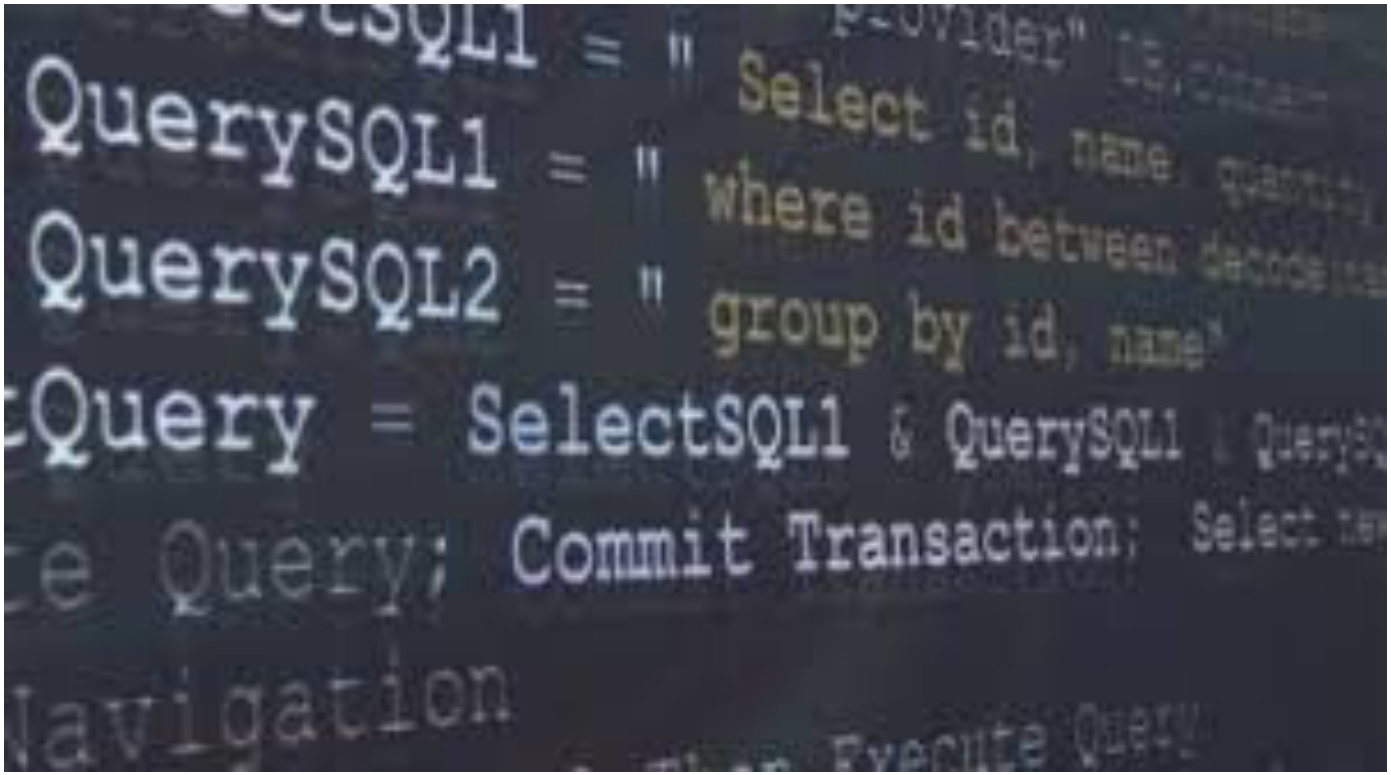
Case Study: IKEA

- Objective: proximity marketing
  - Real-time analysis of customer behavior in stores in order to provide targeted offers
- Requirements
  - Ingestion of real-time data on customer itineraries in store (through transactions)
    - Use of beacons (sensors) to identify and locate frequent customers from their smartphone
  - Analysis and segmentation of customers by similar behavior in other stores
- Problem
  - OLTP and OLAP at a very large scale in real time

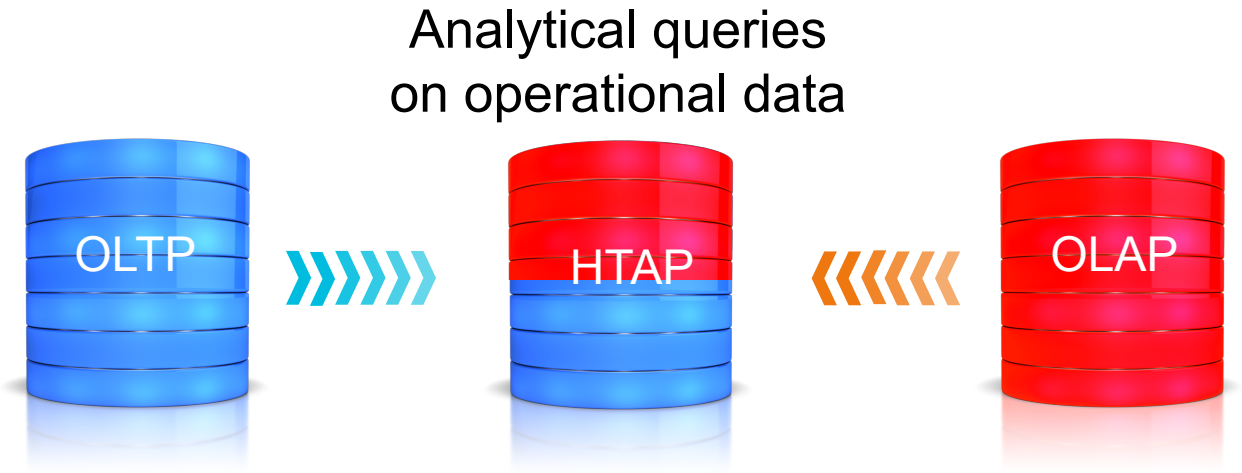
Case Study: Oil & Gas

- Context: drilling oil in a given location
- Objective: detect ASAP that the drilling prospection will fail
  - Save millions of $ by preventing useless drilling
- Requirements
  - Efficient ingestion of real-time data from drillers
    - With *transactions* to guarantee data consistency
  - Real time analytics of all the data produced by the drillers
- Problem
  - Transactions and real-time analytics on driller data

HTAP


QuerySQL1 = " Select id, name
QuerySQL1 = " where id between
QuerySQL2 = " group by id, name"
Query = SelectSQL1 & QuerySQL1
Commit Transaction;
Navigation

HTAP*: blending OLTP & OLAP



- Advantages
  - Cutting cost of business analytics by up to 75%
  - Simpler architecture: no more ETLs/ELTs
  - Real-time analytical queries on current data

*Gartner, 2015

HTAP and Big Data

- Challenges
  - Scaling out transactions
    - Millions of transactions per second
  - Mixed OLTP/OLAP workloads on big data
  - Big data ingestion from remote data sources
  - Polystore capabilities
    - To access HDFS, NoSQL and SQL data sources

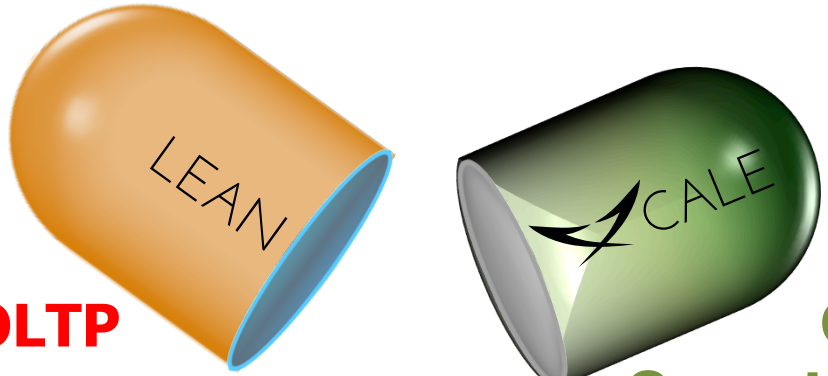
Related Work

- Parallel SQL DBMS
  - Can mix OLTP/OLAP through snapshot isolation and data versioning, e.g., Oracle Exadata
  - But hard to scale OLTP and expensive HW/SW
- In-memory SQL DBMS
  - Can support HTAP (e.g., HANA, MonetDB)
  - But hard to deal with big data
- NoSQL
  - Scalable key-value storage, data partitioning, fault-tolerance, …
  - But no ACID transactions

HTAP Top Systems

Vendor	Product	Comment
LeanXcale Inc.	LeanXcale	Ultra-scalable transactions, based on proprietary KV store (KiVi) and proprietary OLAP leveraging the Calcite optimizer
SAP	HANA	The HTAP pioneer. In-memory, column store
Google	Spanner	NewSQL service with ACID transactions and synchronous replication across data centers
MemSQL Inc.	MemSQL	In-memory, column and row store, MySQL compatible
Esgyn	Esgyn	Apache Trafodion for OLTP, Hadoop for OLAP
NuoDB	NuoDB	Cloud solution (Amazon)
Splice Machine	Splice Machine	HBase as storage engine, Derby as OLTP query engine and SparkQL as OLAP query engine. Custom centralized transactional manager
VoltDB Inc.	VoltDB	Open source and proprietary. In-memory

LEANXCALE



Ultra-Scalable OLTP

Full SQL Full ACID DB

Polyglot

Queries across SQL, HBase, Neo4J,
MongoDB, & Hadoop data lakes
Integration with Data Streaming

OLAP over Operational Data

Real-Time Big Data

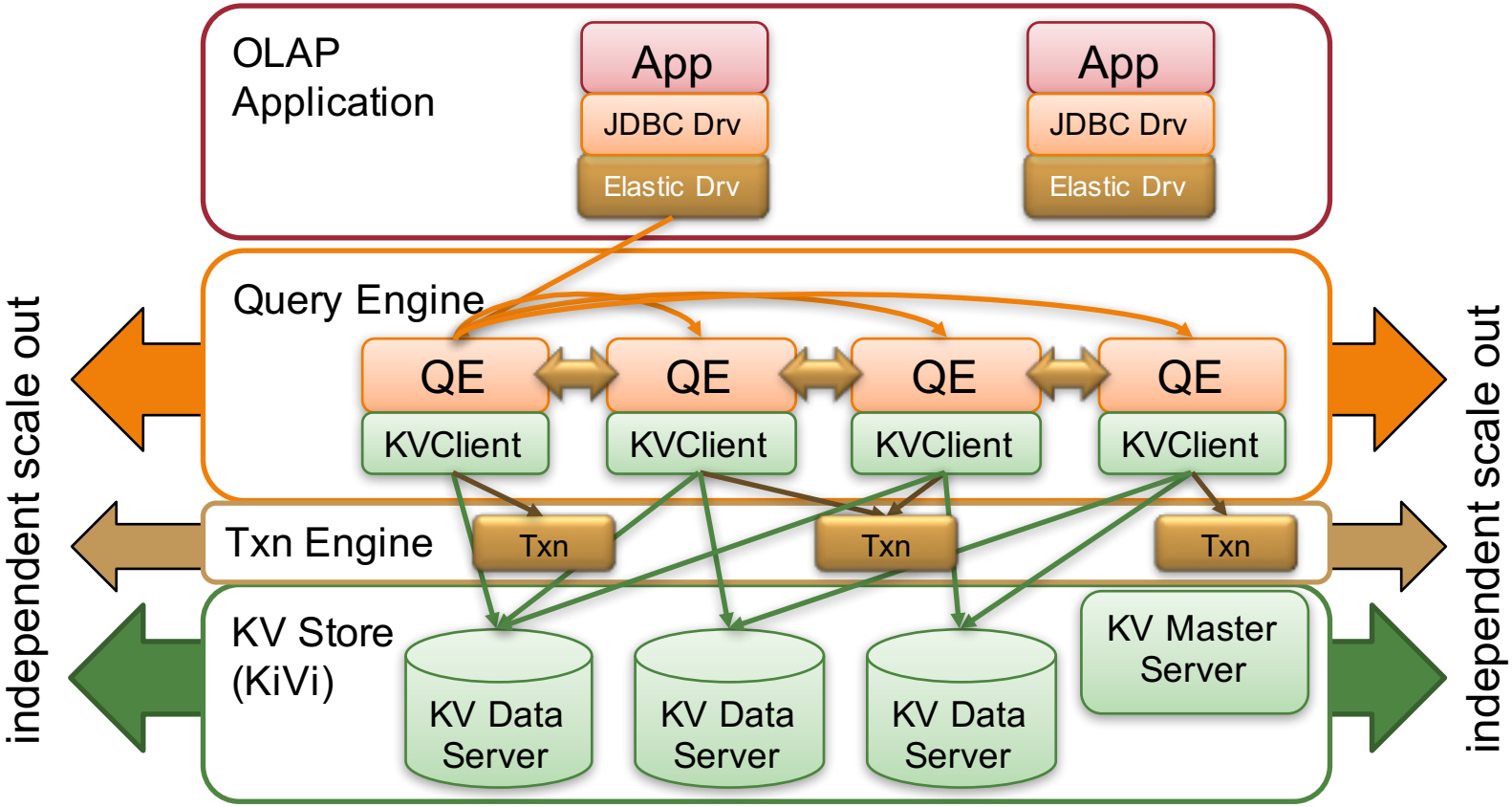
Elastic & Ultra-Efficient

Non-disruptive data migration,
continuous load balancing

LEANXCALE

- SQL/JSON DBMS
  - Access from a JDBC driver
- Key-value store (KiVi)
  - Dual SQL/KV interface over relational data with efficiency, elasticity, high availability, indexing, …
  - Fast, parallel data ingestion
  - Polystore access: HDFS, NoSQL, …
- OLAP parallel processing
  - Based on the Apache Calcite optimizer
  - Extensive push down of operators to KiVi
- Ultra-scalable transaction processing

LeanXcale Distributed Architecture



KiVi – Efficiency

- Multi-Workload
  - Efficient for both range queries and large data ingestion (updates/inserts)
  - Combines benefits of B+ and LSM trees thanks to a novel proprietary data structure
- NUMA aware architecture
  - Avoids cost of context switches, thread synchronization and remote NUMA accesses in multicore processors
- Vectorial
  - Uses vectorial registers and SIMD instructions, yielding 10-50x acceleration
- Columnar storage
  - Yields 10-100x acceleration for tables with large number of columns

KiVi – Elasticity

- Dynamic data migration
  - Able to move data partitions across servers without affecting the QoS of the applications updating those
- Dynamic load balancing
  - Balances the load across servers based on the current load using dynamic data migration
  - Takes into account all resource utilization: CPU, memory, IO, and network
- Fully elastic
  - Adds and removes nodes as needed to minimize resource usage

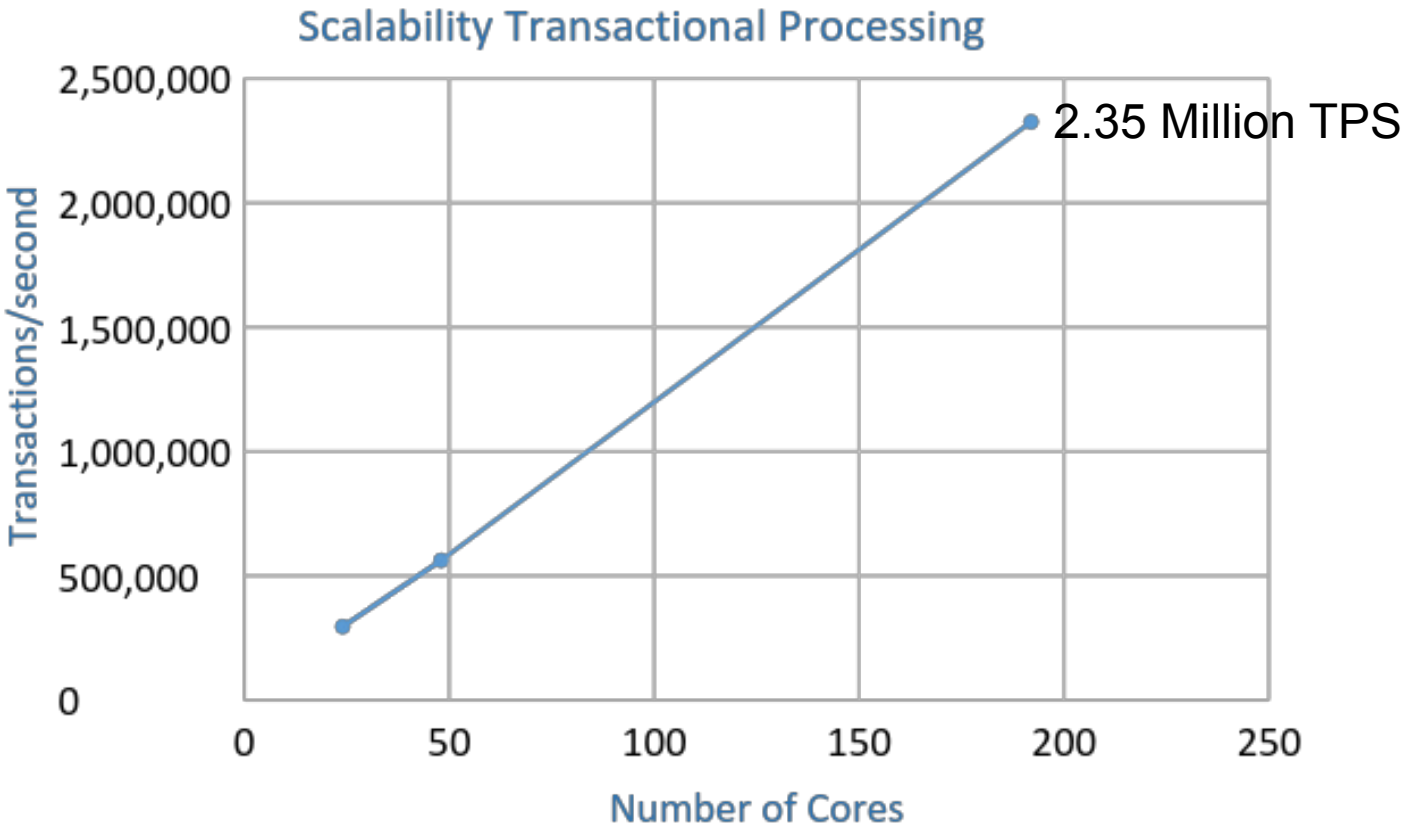
KiVi - Online Aggregation

- Commutative concurrency control
  - Enables to aggregate data (additions/subtractions) with high levels of concurrency without conflicts
- Online aggregation
  - Have an aggregate table
    - WebServer (server, size, nb_users, …)
  - A transaction can insert records and compute aggregations (SUM, COUNT, ... but not AVG) without experiencing conflicts
- Aggregation analytical queries become costless single row queries
  - Computing an aggregate simply requires reading the row from the aggregate table, thus removing the overhead of traditional aggregation analytical queries

KiVi – High Availability

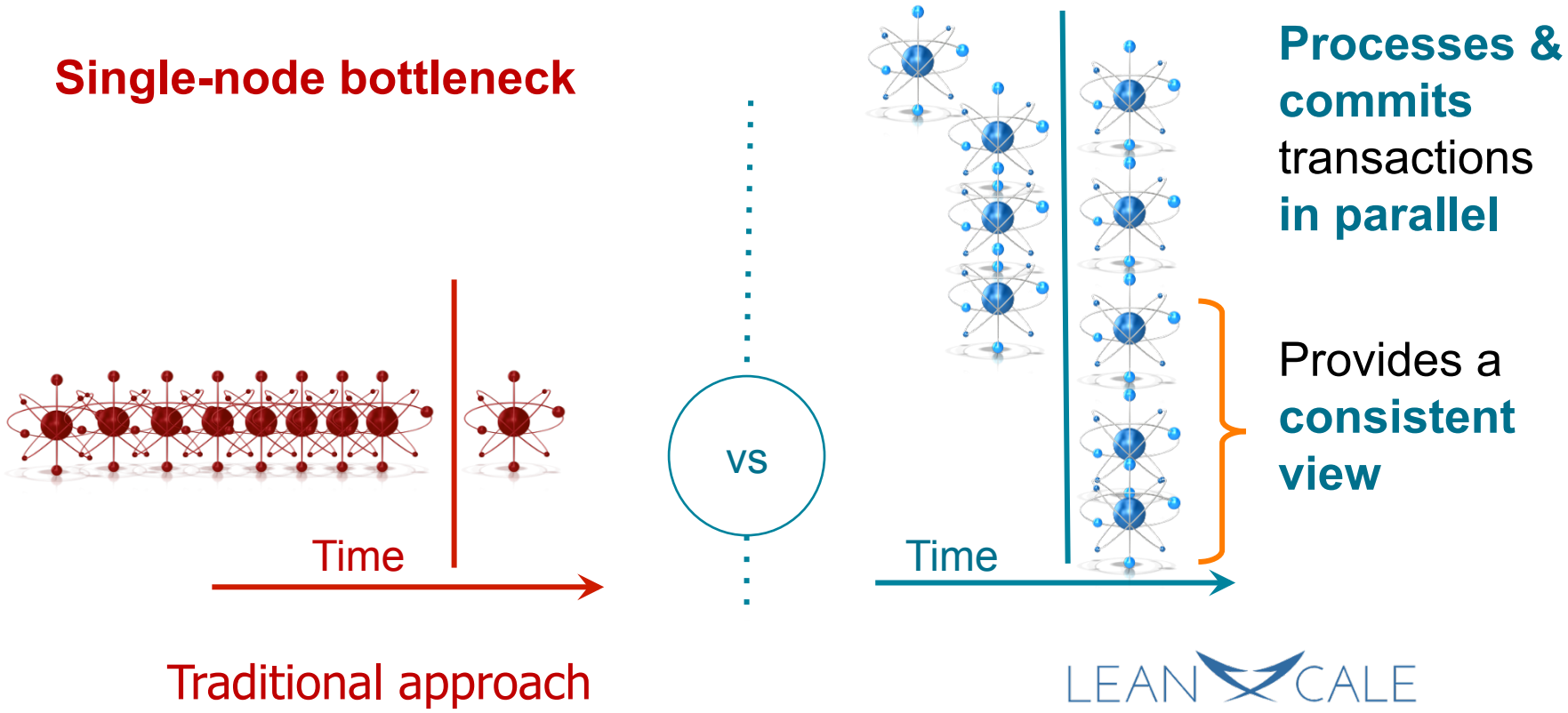
- Contention free, active-active replication
  - Takes advantage of transactional scalability
  - Fail-over: when a storage server fails, the other replicas take over and are already up-to-date, yielding zero-downtime
  - Novel replication algorithm that avoids expensive synchronization (2PC, Paxos) during commit across replicas

Transactional Scalability



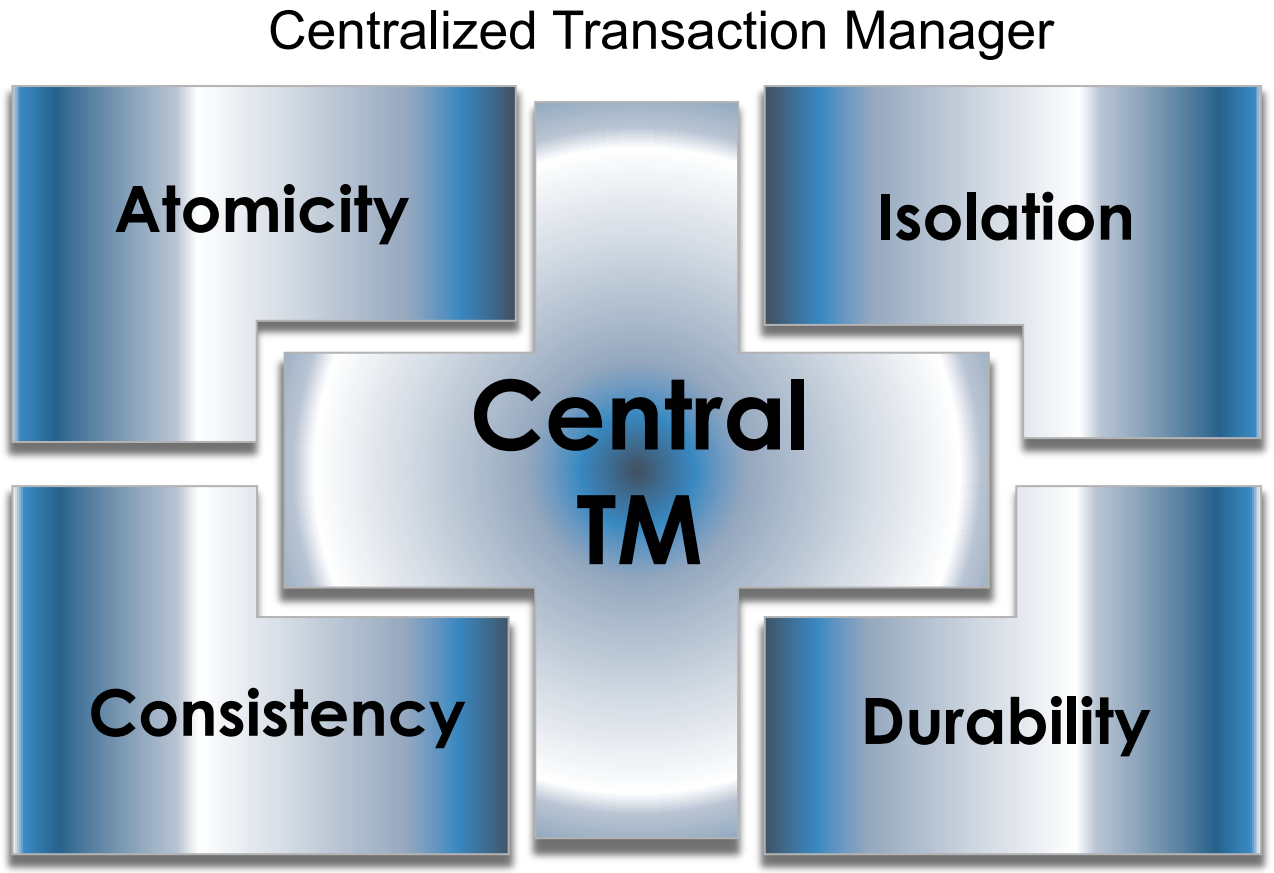
- Without data manager/logging to see how much TP throughput can be attained
- Based on a micro-benchmark to stress the TM

Highly Scalable Transaction Processing*



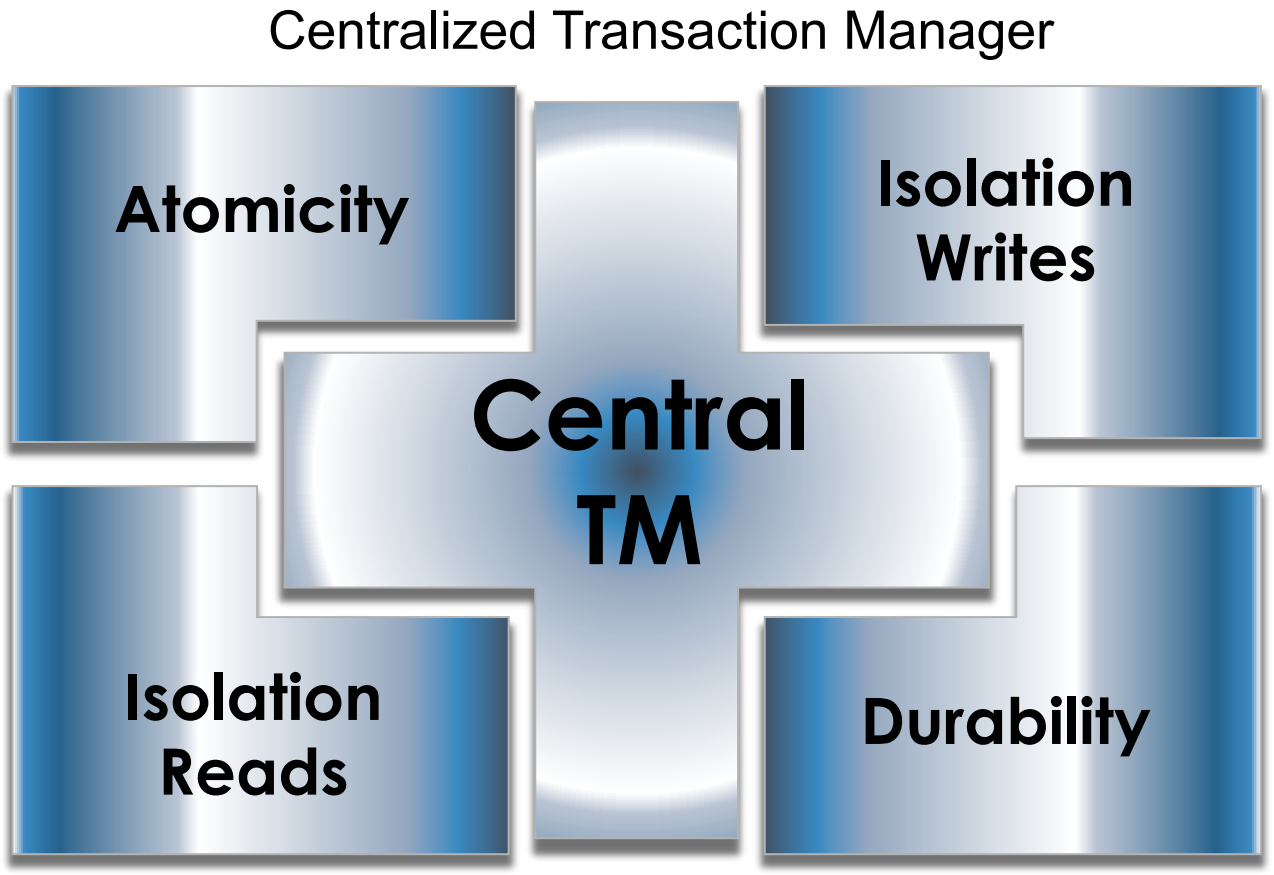
* R. Jimenez-Peris, M. Patiño-Martinez. System and method for highly scalable decentralized and low contention transactional processing. Priority date: 11th Nov. 2011. European Patent #EP2780832, US Patent #US9,760,597.

Traditional Approach



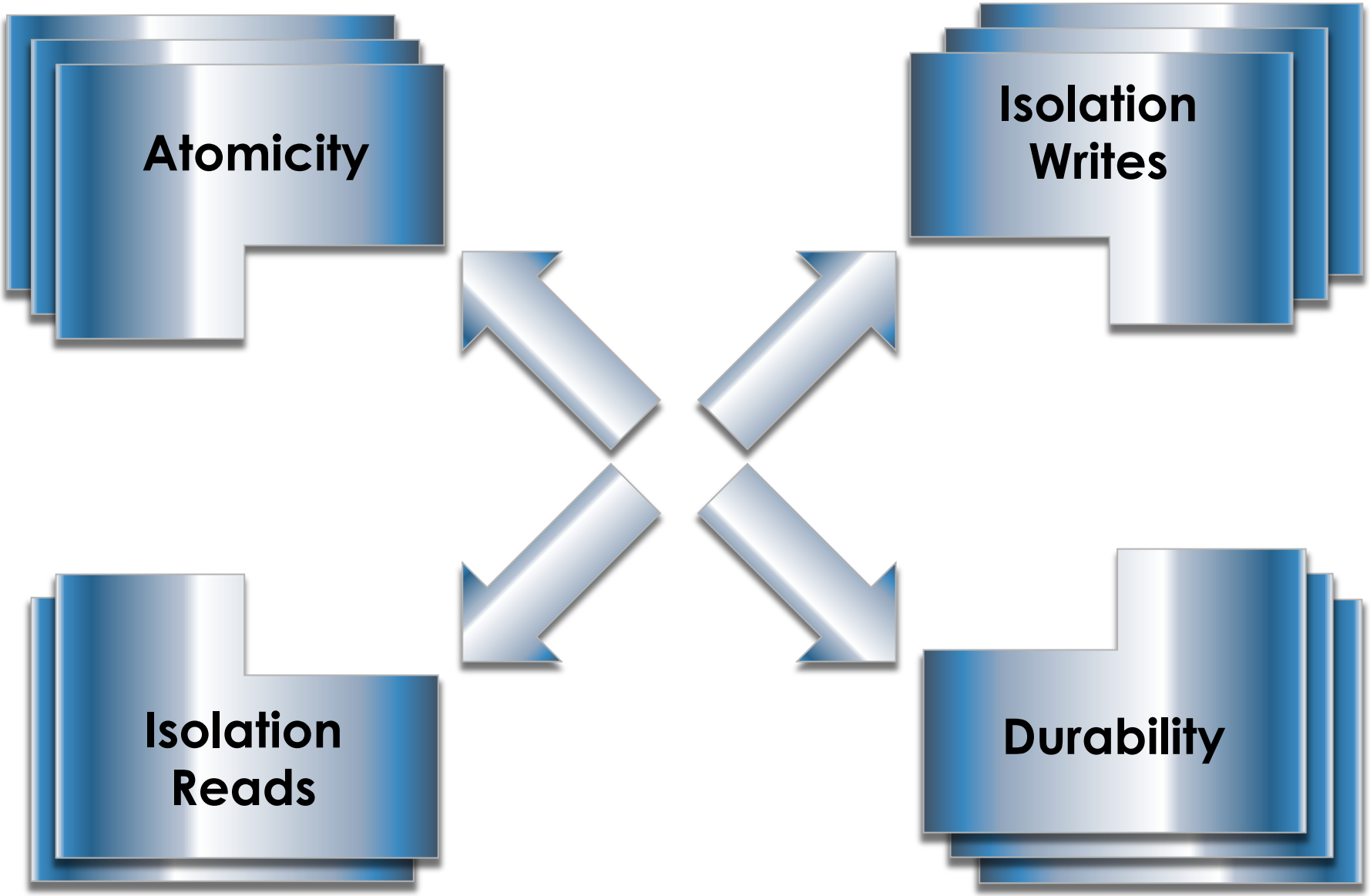
Single-node bottleneck

Traditional Approach



Single-node bottleneck

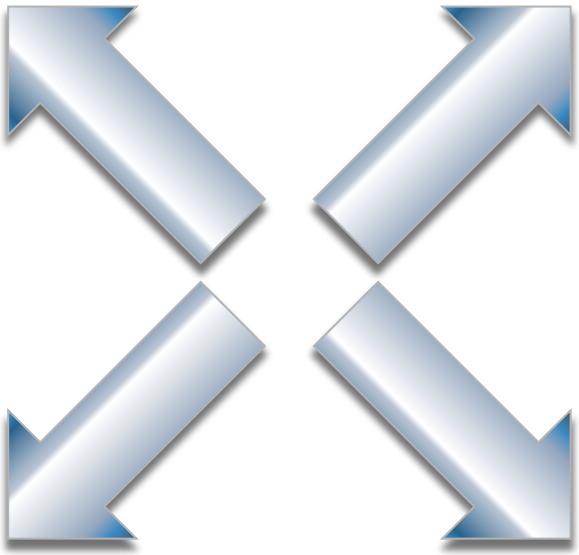
Scaling ACID Properties



Scaling ACID Properties

Local TMs

Atomicity



Conflict managers

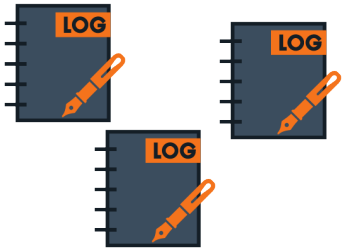
Isolation Writes

Isolation Reads

Snapshot server

Commit sequencer

Durability

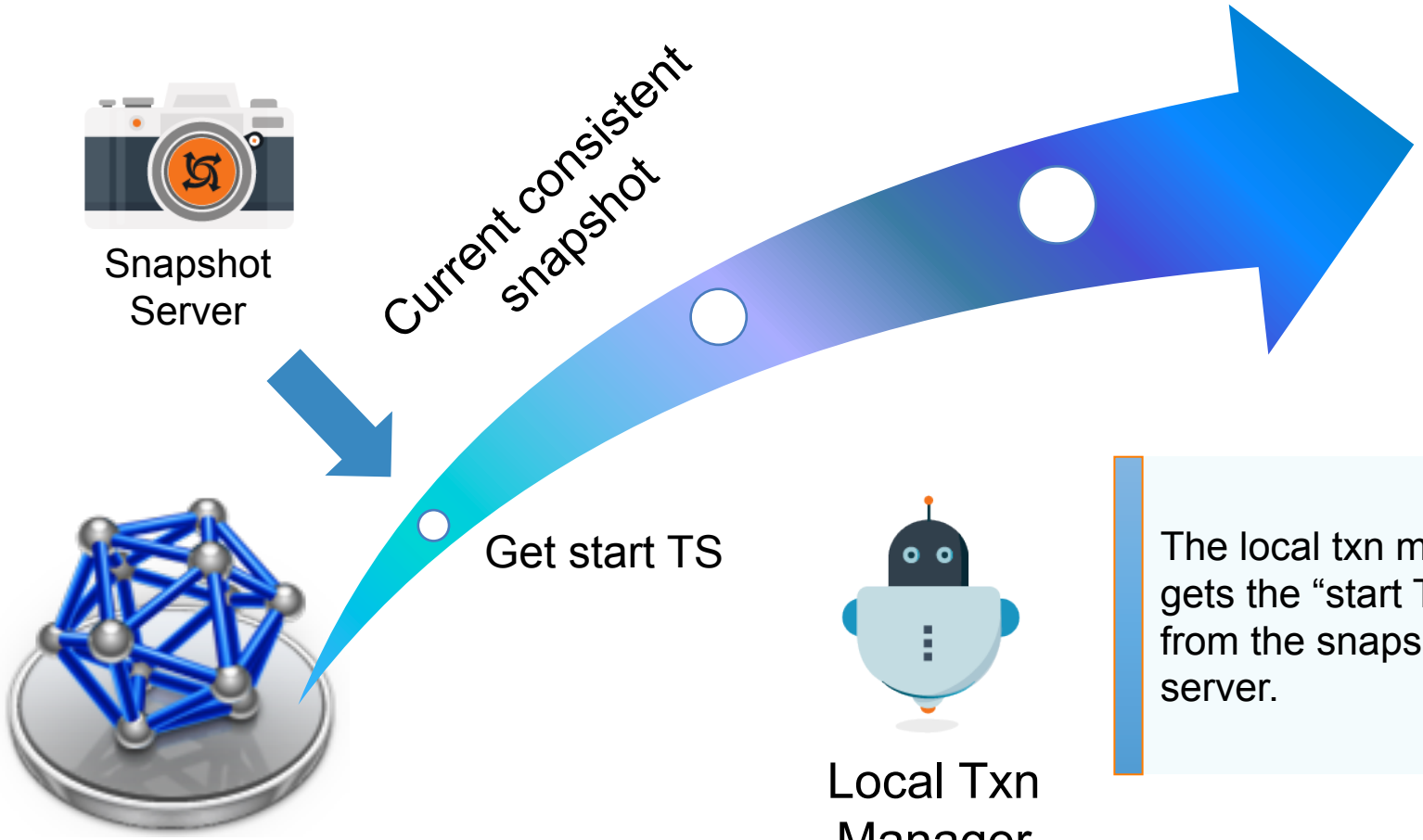


Loggers

Transaction Management Principles

- Separation of commit from the visibility of committed data
- Proactive pre-assignment of commit timestamps to committing transactions
- Detection and resolution of conflicts before commit
- Transactions can commit in parallel because:
  - They do not conflict
  - They have their commit timestamp already assigned that will determine their serialization order
  - Visibility is regulated separately to guarantee the reading of fully consistent states

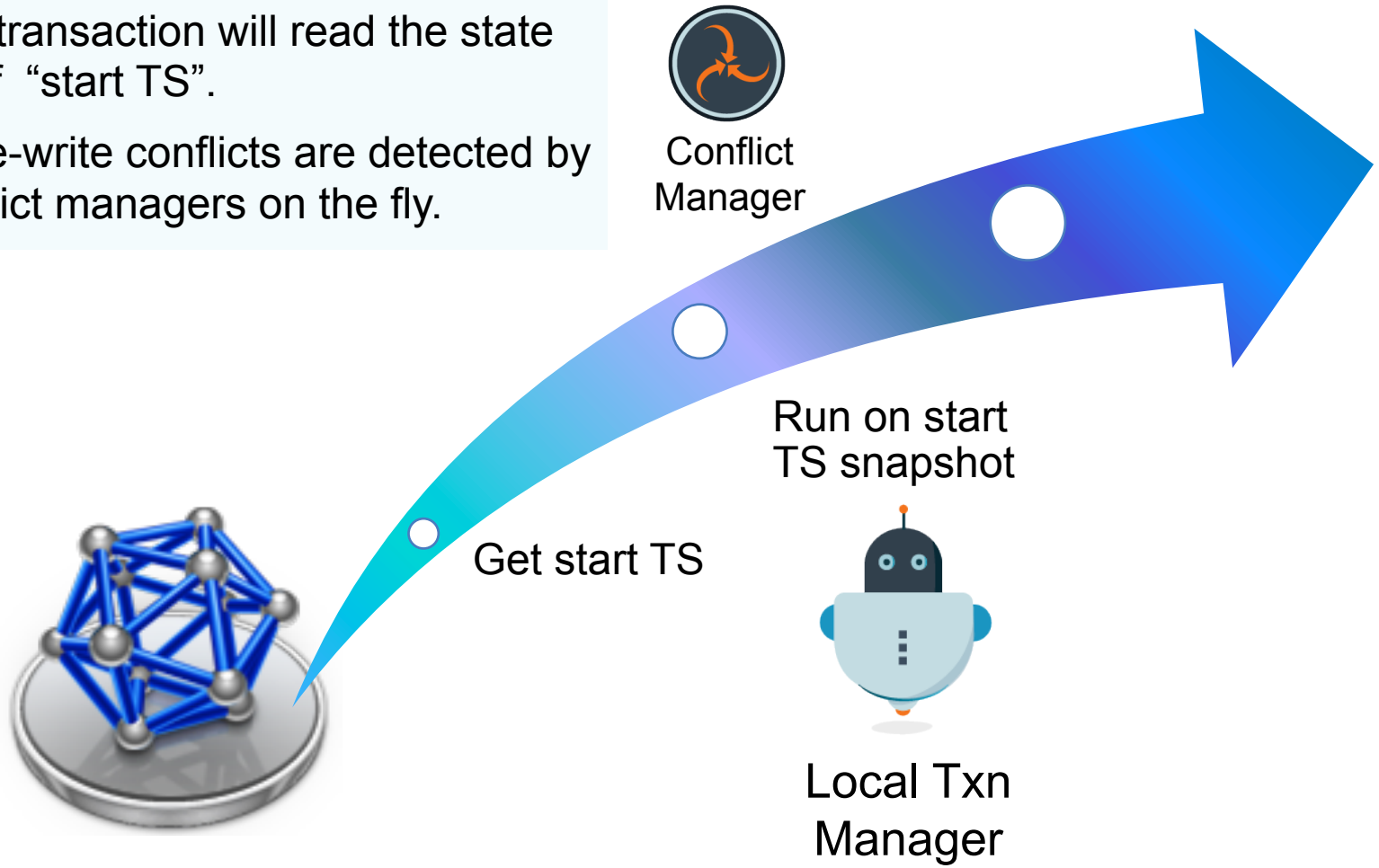
Transactional Life Cycle: start



Transactional Life Cycle: execution

The transaction will read the state as of "start TS".

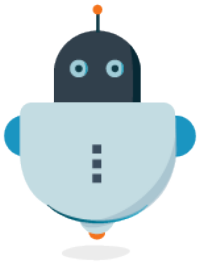
Write-write conflicts are detected by conflict managers on the fly.



Transaction Life Cycle: commit

The local transaction manager orchestrates the commit.

Transaction Life Cycle: commit



Local Txn Manager

Get Commit TS → Log → Public Updates → Report Snaps Serv

Commit TS

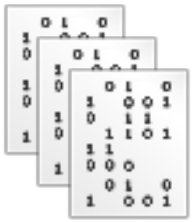
Writeset

Writeset

Commit TS

Commit Sequencer

Logger

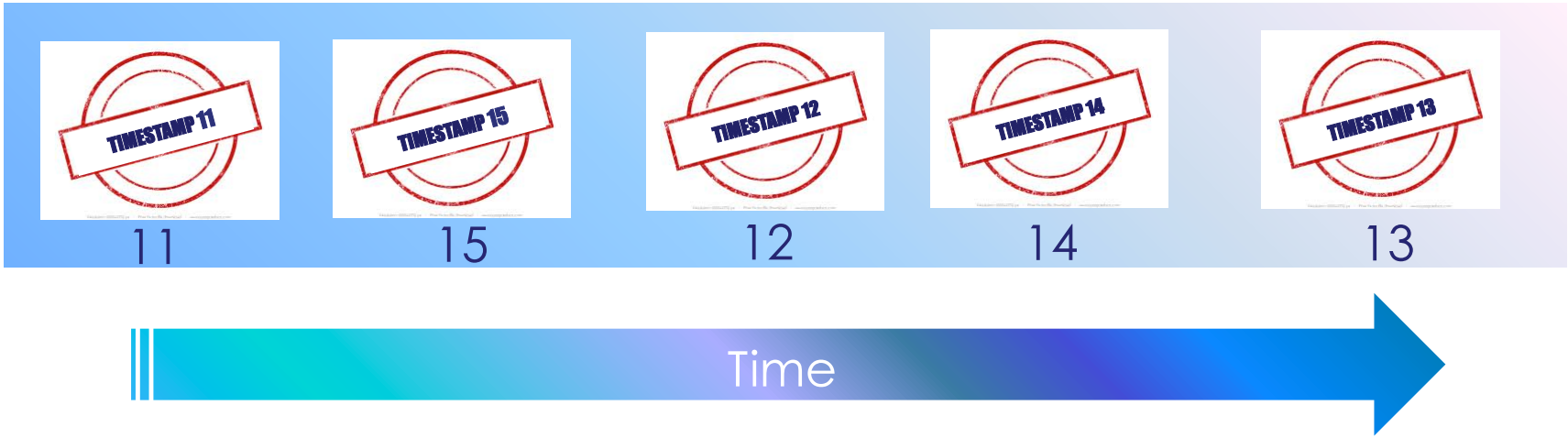


Data Store

Snapshot Server

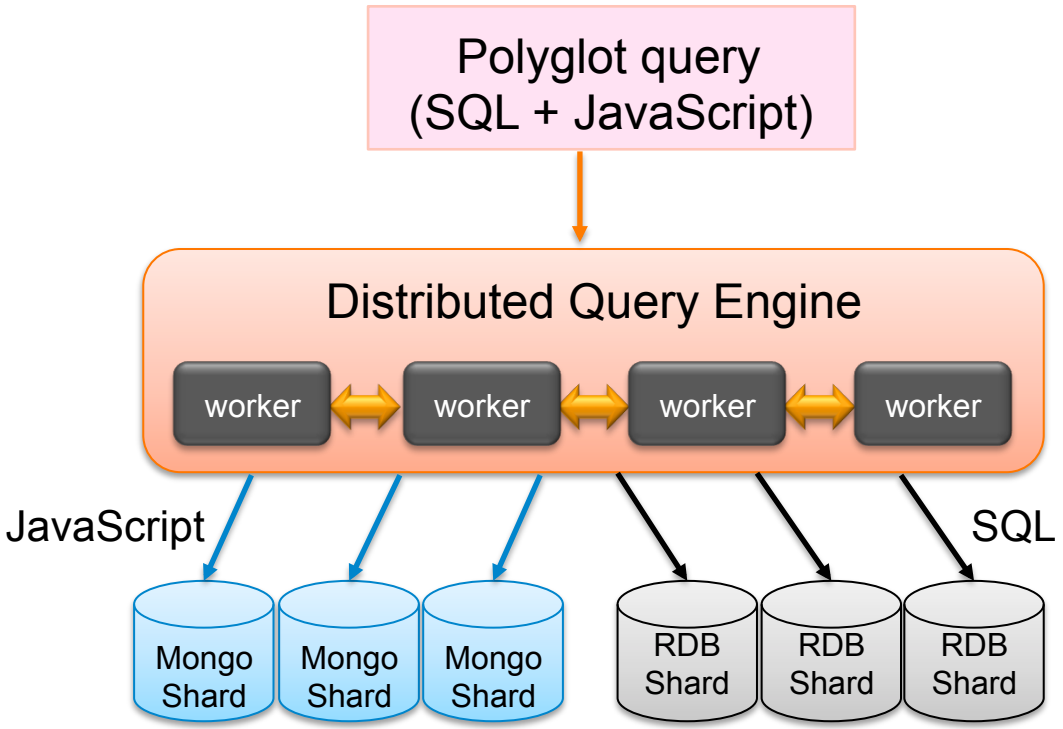
Transaction Life Cycle: commit

Sequence of commit timestamps received by the Snapshot Server



Evolution of the current snapshot at the Snapshot Server (staring at 10)

Parallel Polystore Query Processing with LeanXcale*

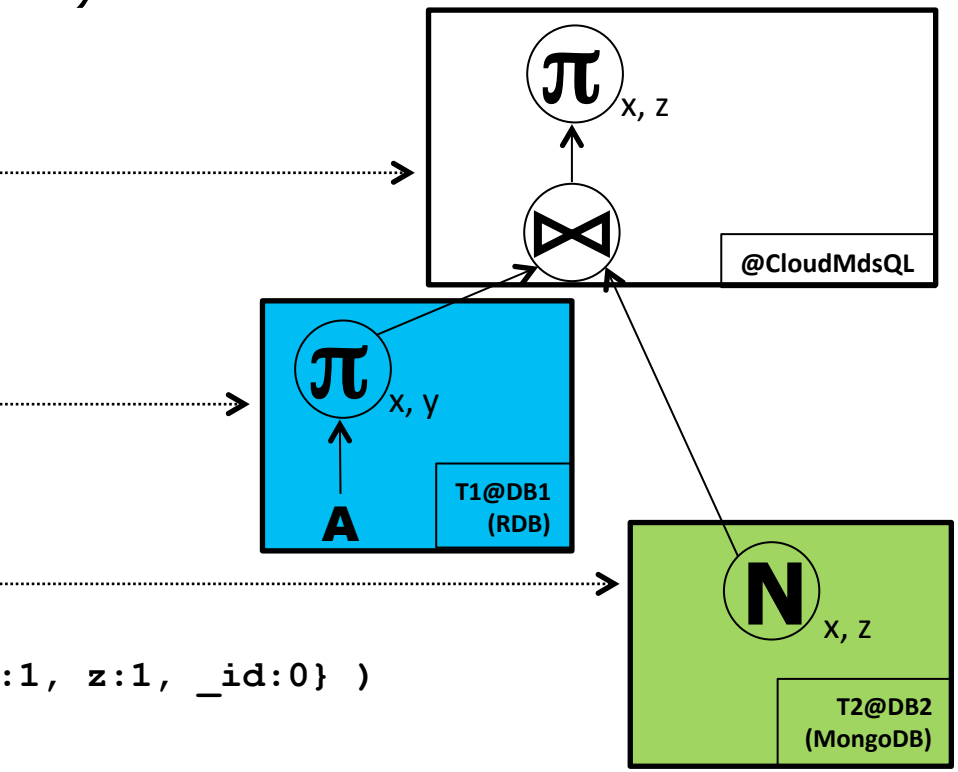


*B. Kolev, O. Levchenko, E. Pacitti, P. Valduriez, R. Vilaça, R. Gonçalves, R. Jiménez-Peris, P. Kranas. Parallel Polyglot Query Processing on Heterogeneous Cloud Data Stores with LeanXcale. IEEE Big Data, 2018.

Polyglot Query Example

- A query in CloudMdsQL* that integrates data from
  - DB1 – relational (RDB)
  - DB2 – document (MongoDB)

```
/* Integration */
SELECT T1.x, T2.z
FROM T1 JOIN T2
  ON T1.x = T2.x

/* SQL sub-query */
T1(x int, y int)@DB1 =
( SELECT x, y FROM A )

/* Native sub-query */
T2(x int, z string)@DB2 = {*
  db.B.find( {$lt: {x, 10}}, {x:1, z:1, _id:0} )
*}
```

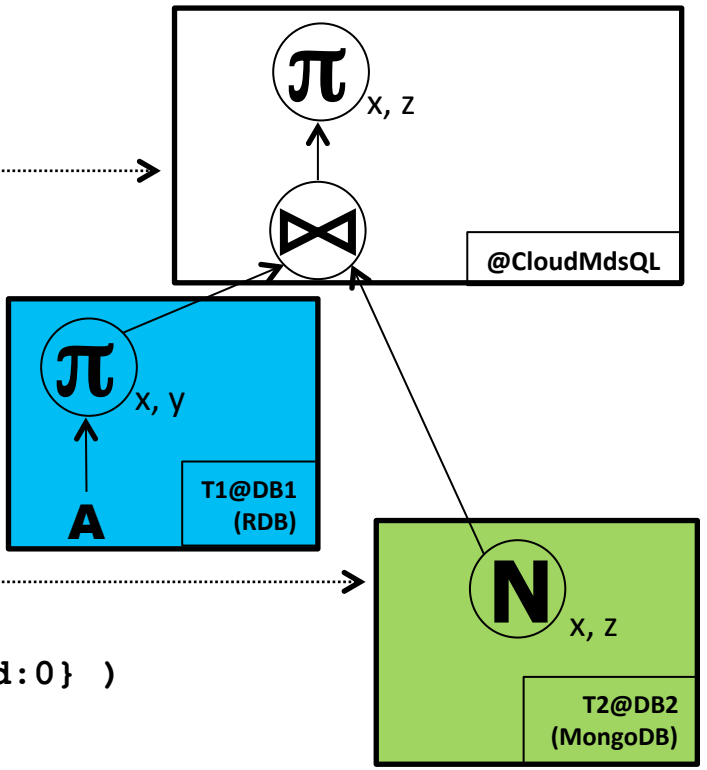


*B. Kolev, C. Bondiombouy, P.Valduriez, R. Jiménez-Peris, R. Pau, J. Pereira. The CloudMdsQL Multistore System. SIGMOD 2016.

Polyglot Query Example

- CloudMdSQL = SQL + subqueries
  - Expressed as named tables on ad-hoc schema
  - Compiled to query sub-plans

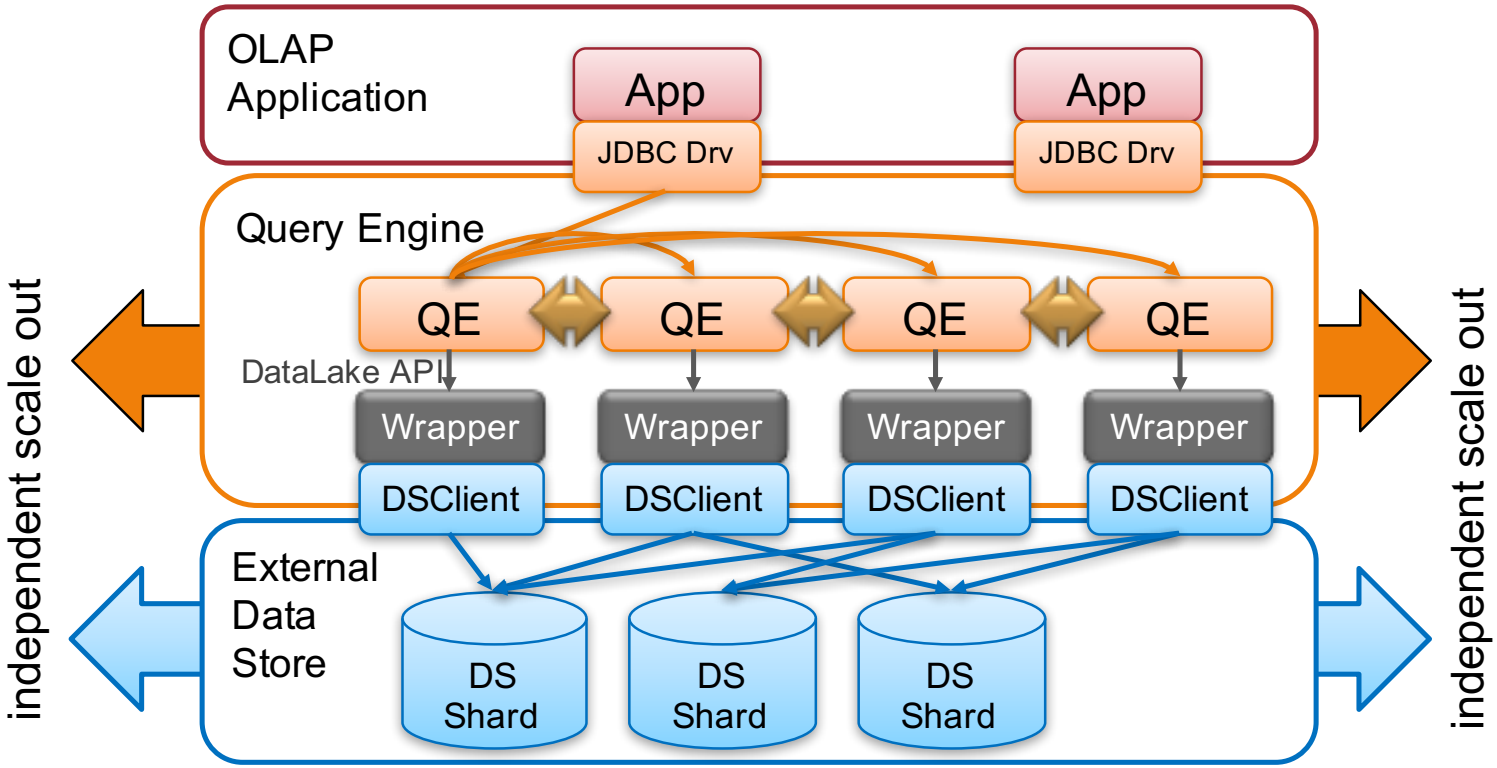
```
/* Integration */
SELECT T1.x, T2.z
FROM T1 JOIN T2
  ON T1.x = T2.x

/* SQL sub-query */
T1(x int, y int)@DB1 =
( SELECT x, y FROM A )

/* Native sub-query */
T2(x int, z string)@DB2 = {*
  db.B.find( {$lt: {x, 10}}, {x:1, z:1, _id:0} )
*}
```



Parallel Polystore Query Processing

- Objectives
  - Intra-operator parallelism
    - Apply parallel algorithms
  - Exploit data sharding in data stores
    - Access data shards (partitions) in parallel
  - Polyglot capabilities
  - Optimization
    - Select pushdown, bindjoin, etc.
- Solution
  - The LeanXcale Distributed Query Engine (DQE)
    - … with CloudMdsQL polyglot extensions

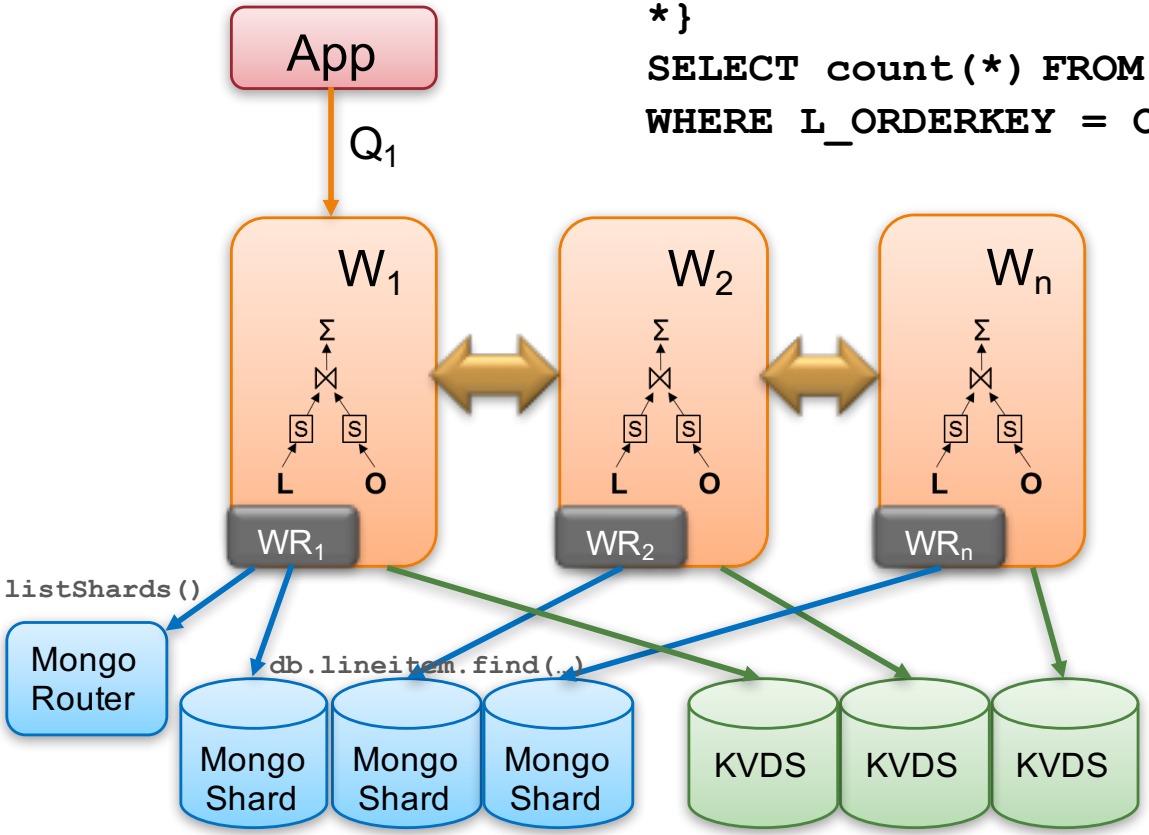
LeanXcale Polystore Architecture



- Workers access directly data shards through wrappers
  - DataLake API: get list of shards; assign shard to worker

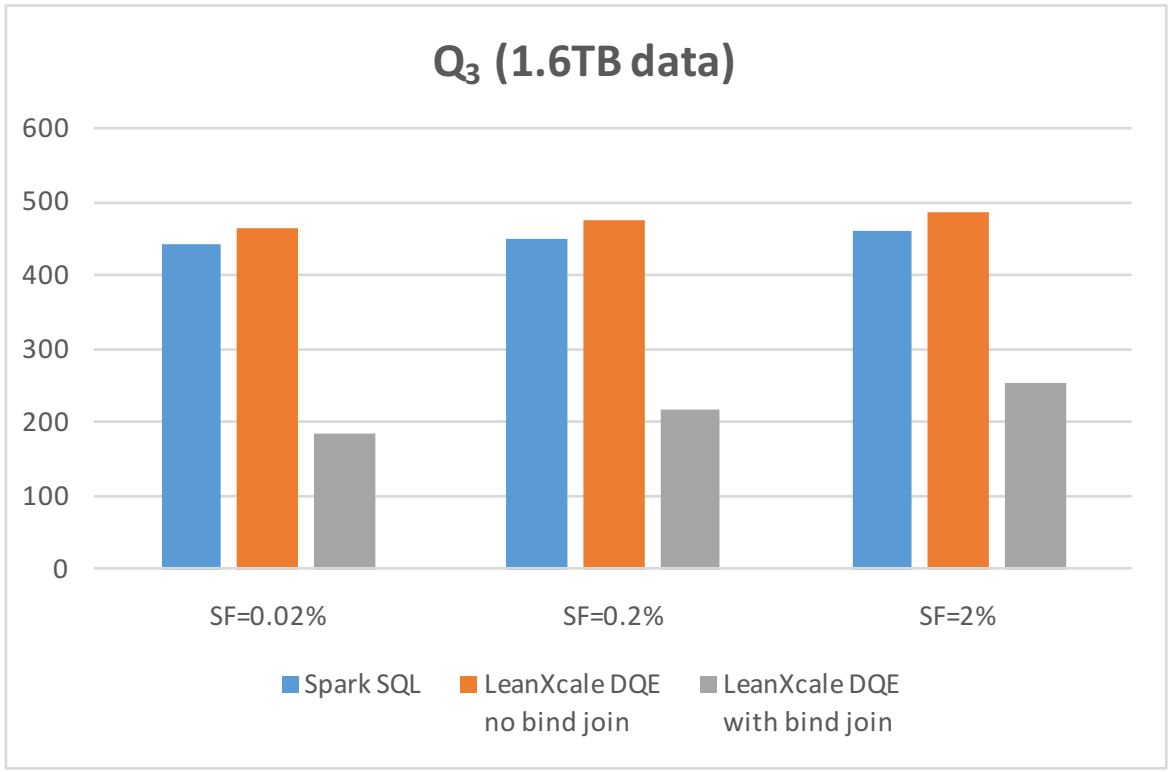
Query on LeanXcale and MongoDB

```
LineItem( L_ORDERKEY int, … )@mongo = {*
  return db.lineitem.findSharded(
     {l_quantity: {$lt: 5}} );
*}
SELECT count(*) FROM LineItem L, Orders O
WHERE L_ORDERKEY = O_ORDERKEY
```



Performance Evaluation

- Clicks: 1TB, 6 billion rows
- Orders_Items: 600GB, 3 billion items, 770 million docs
- 3 selectivity factors on the Clicks table*



* Experiments performed with the previous version of LeanXcale based on HBase

Research Directions in HTAP

Many Research Opportunities

- Polyglot SQL
  - SQL++ compatibility
  - JSON indexing within columns
- Polystore
  - Cost model, including histograms
  - Materialized views
- Streaming and CEP
  - Query language combining streaming and access to the database, e.g., through SQL or KiVi API
- Scientific applications
  - HTAP + scientific workflows
- Analytics and ML
  - Spark ML using updatable RDDs, instead of redoing RDDs periodically,
  - Incremental ML algorithms based on online aggregation, scalable updates and OLAP queries (as supported by LeanXcale)
- Benchmarking
  - Defining HTAP benchmarks and compare HTAP systems
  - Profiling HTAP (e.g., LeanXcale and KiVi) to find new optimizations

References

1. B. Kolev, O. Levchenko, E. Pacitti, P. Valduriez, R. Vilaça, R. Gonçalves, R. Jiménez-Peris, P. Kranas. Parallel Polyglot Query Processing on Heterogeneous Cloud Data Stores with LeanXcale. *IEEE Big Data*, 2018.

2. B. Kolev, P. Valduriez, C. Bondiombouy, R. Jiménez-Peris, R. Pau, J. Pereira. CloudMdsQL: Querying Heterogeneous Cloud Data Stores with a Common Language. *Distributed and Parallel Databases*, 34(4): 463-503, 2016.

3. B. Kolev, C. Bondiombouy, O. Levchenko, P.Valduriez, R. Jiménez-Peris, R. Pau, J. Pereira. Design and Implementation of the CloudMdsQL Multistore System. *CLOSER 2016.*

4. B. Kolev, C. Bondiombouy, P.Valduriez, R. Jiménez-Peris, R. Pau, J. Pereira. The CloudMdsQL Multistore System. *ACM SIGMOD 2016.*

5. B. Kolev, R. Pau, O. Levchenko, P. Valduriez, R. Jiménez-Peris, J. Pereira. Benchmarking polystores: The CloudMdsQL experience. Workshop on Methods to Manage Heterogeneous Big Data and Polystore Databases*, IEEE BigData*, 2016.

6. R. Jimenez-Peris, M. Patiño-Martinez. System and method for highly scalable decentralized and low contention transactional processing. Filed at USPTO: 2011. European Patent #EP2780832, US Patent #US9,760,597.