NewSQL
Principles, Systems and Current Trends

Patrick Valduriez

Ricardo Jimenez-Peris

Outline

- Motivations
- Principles and techniques
- Taxonomy of NewSQL systems
- Current trends

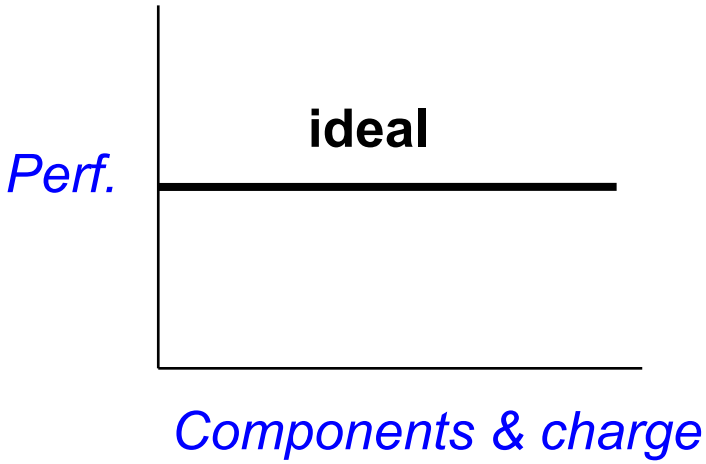
Motivations

Why NoSQL/NewSQL?

- Trends
  - Big data
    - Unstructured data
  - Data interconnection
    - Hyperlinks, tags, blogs, etc.
  - Very high scalability
    - Data size, data rates, concurrent users, etc.
- Limits of SQL systems (in fact RDBMSs)
  - Need for skilled DBA, tuning and well-defined schemas
  - Full SQL complex
  - Hard to make updates scalable
    - Parallel RDBMS use a shared-disk for OLTP, which is hard to scale

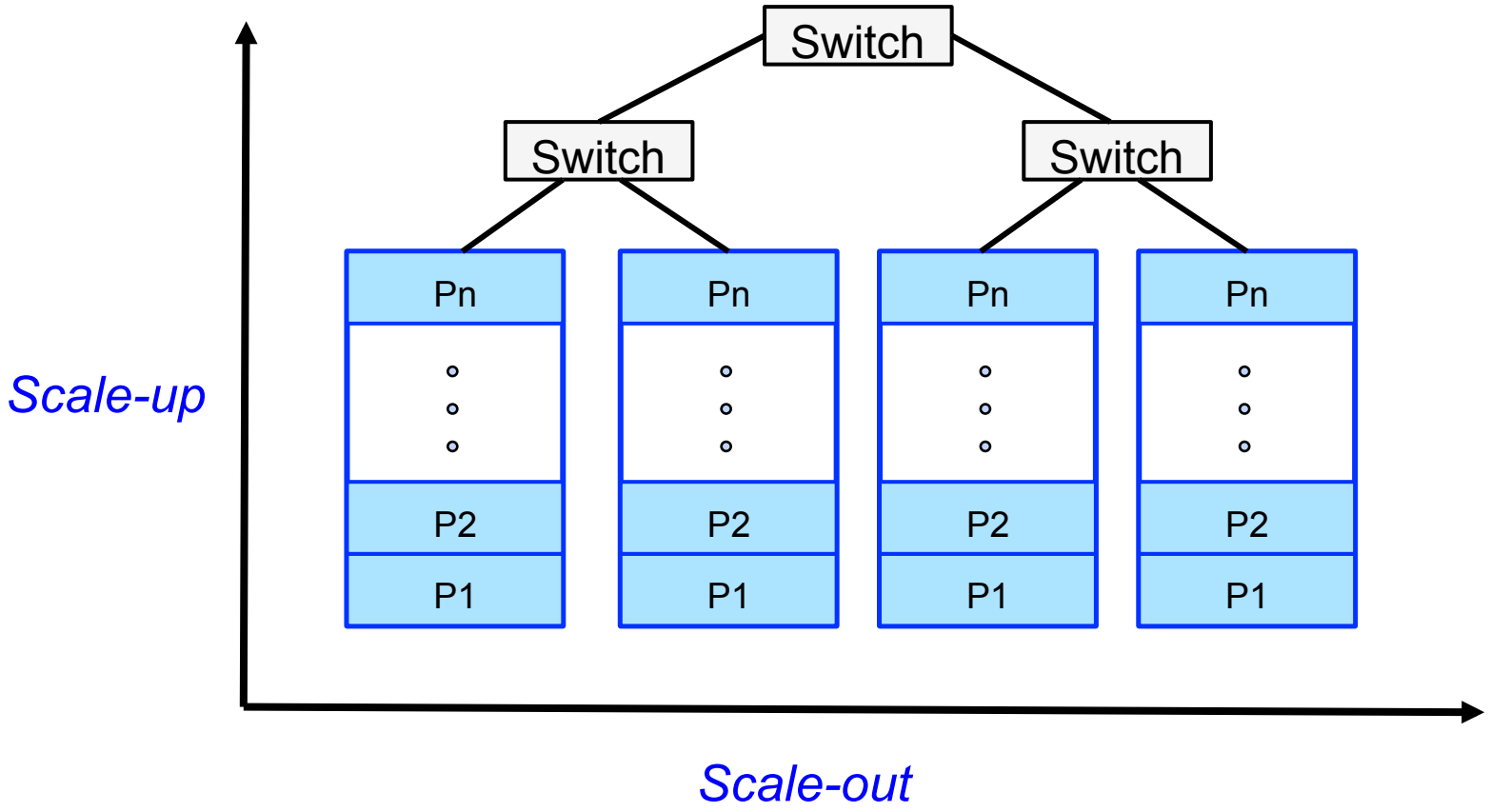
Scalability

- Ideal: linear scale-up
  - Sustained performance for a linear increase of database size and load
  - By proportional increase of components



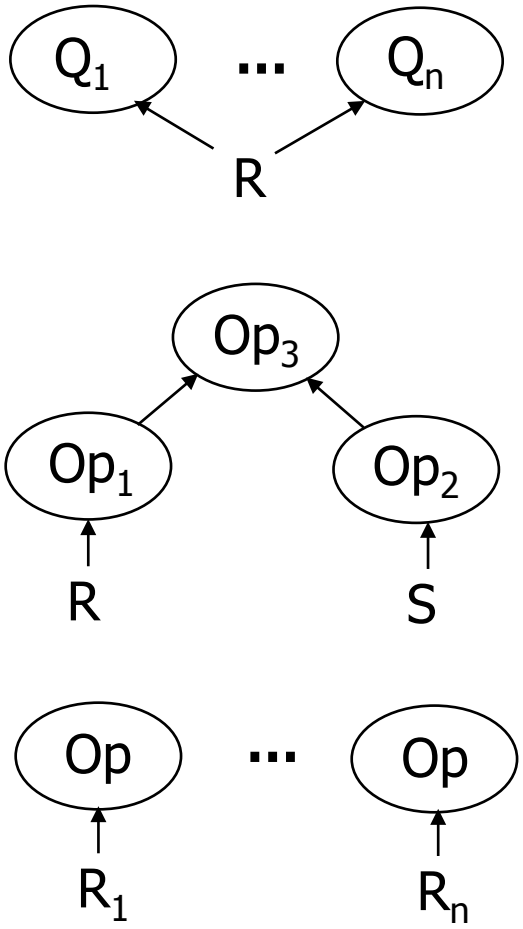
Vertical vs Horizontal Scaleup

- Typically in a shared-nothing computer cluster



Query Parallelism

- Inter-query
  - Different queries on the same data
  - For concurrent queries
- Inter-operation
  - Different operations of the same query on different data
  - For complex queries
- Intra-operation
  - The same operation on different data
  - For large queries



The CAP Theorem

- Polemical topic
  - "A database can't provide consistency AND availability during a network partition"
  - Argument used by NoSQL to justify their lack of ACID properties
  - Nothing to do with scalability
- Two different points of view
  - Relational databases
    - Consistency is essential
      - ACID transactions
  - Distributed systems
    - Service availability is essential
      - Inconsistency tolerated by the user, e.g. web cache

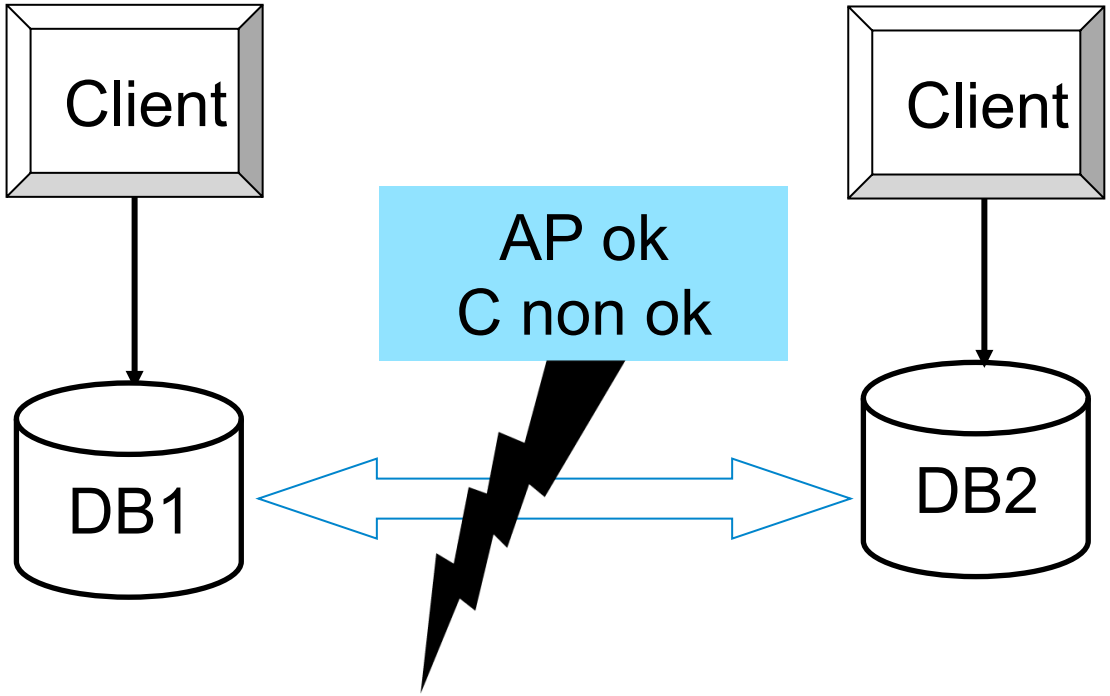
What is the CAP Theorem?

- The desirable properties of a distributed system
  - ***C****onsistency*: all nodes see the same data values at the same time
  - ***A****vailability*: all requests get an answer
  - ***P****artition tolerance*: the system keeps functioning in case of network failure
- History
  - At the PODC 2000 conference, Brewer (UC Berkeley) conjectures that one can have only two properties at the same time
  - In 2002, Gilbert and Lynch (MIT) prove the conjecture, which becomes a theorem

Strong vs Eventual Consistency

- Strong consistency (ACID)
  - All nodes see the same data values at the same time
- Eventual consistency
  - Some nodes may see different data values at the same time
  - But if we stop injecting updates, the system reaches strong consistency

Illustration: Symmetric, Asynchronous Replication



But we have eventual consistency

- After reconnection (and resolution of update conflicts), consistency can be obtained

NoSQL (Not Only SQL) Definition

- Specific DBMS, typically for web-based data
  - Specialized data model
    - Key-value, table, document, graph
  - Trade relational DBMS properties
    - Full SQL, ACID transactions, data independence
  - For
    - Simplicity (schema, basic API)
    - Scalability and performance
    - Flexibility for the programmer (integration with programming language)

NoSQL Approaches

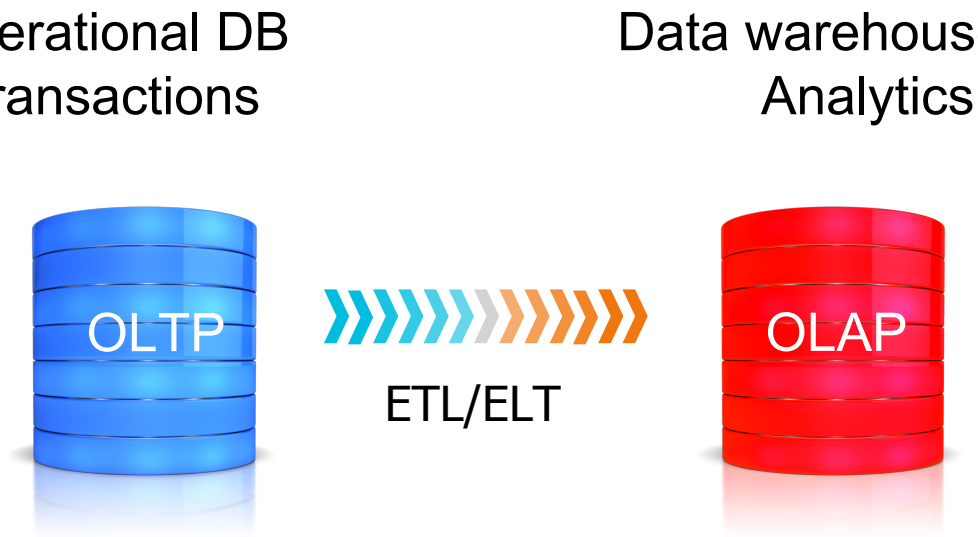
- Characterized by the data model, in increasing order of complexity:
  1. Key-value: DynamoDB, RockDB, Redis
  2. Tabular: Hbase, Bigtable, Cassandra
  3. Document: MongoDB, Coubase, CouchDB, Expresso
  4. Graph: Neo4J, AllegroGraph, MarkLogic, RedisGraph
  5. Multimodel: OrientDB, ArangoDB

- What about object DBMS or XML DBMS?
  - Were there much before NoSQL
  - Sometimes presented as NoSQL
  - But not really scalable

NewSQL

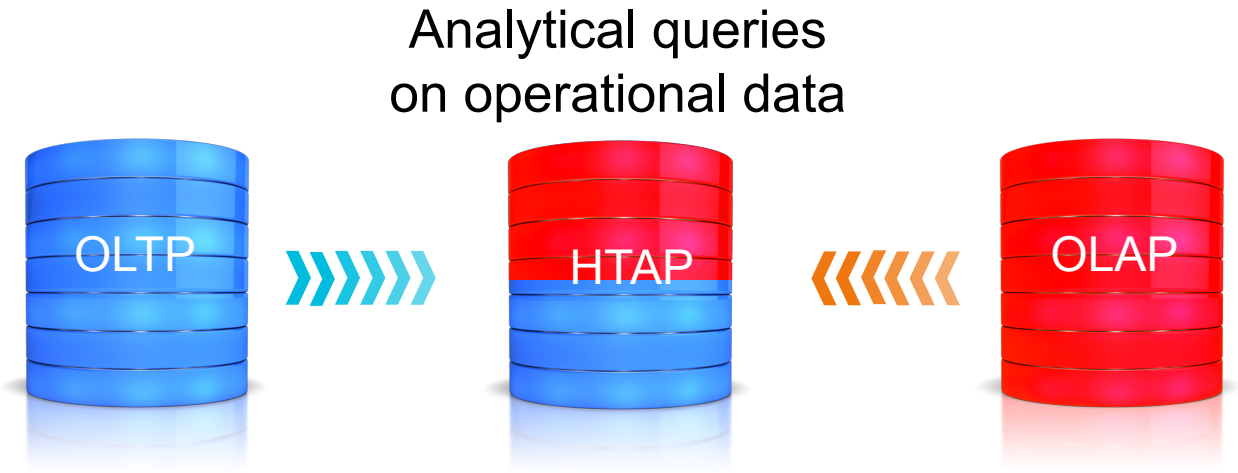
- Pros NoSQL
  - Scalability
    - Often by relaxing strong consistency
  - Performance
  - Practical APIs for programming
- Pros Relational
  - Strong consistency
  - Transactions
  - Standard SQL
    - Makes it easy for tool vendors (BI, analytics, …)
- NewSQL = NoSQL/relational hybrid

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.

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

Case Study: Google AdWords

- Application to produce sponsored links as results of search engine
  - Revenue: $50 billion/year
- Use of an auction system
  - Pure competition between suppliers to gain access to consumers, or consumer models (the probability of responding to the ad), and determine the right price offer (maximum cost-per-click (CPC) bid)
- The AdWords database with Google F1
  - 30 billion search queries per month
  - 1 billion historical search events
  - Hundreds of Terabytes

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

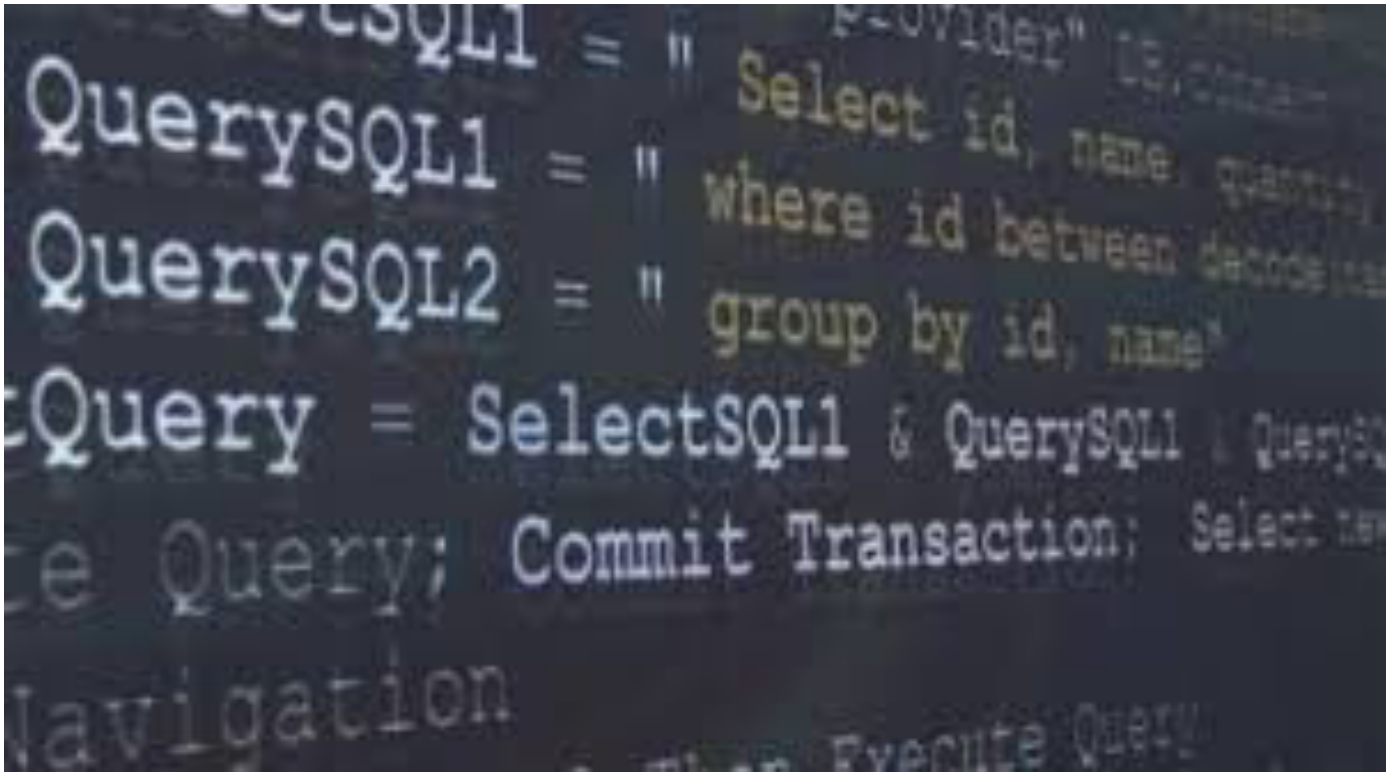
Case Study: Ikea

- 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

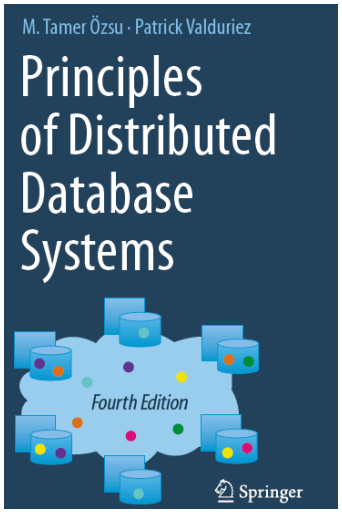
Case Study: Oil & Gas

- 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

Principles and Techniques of NewSQL



Principles of Distributed Database Systems

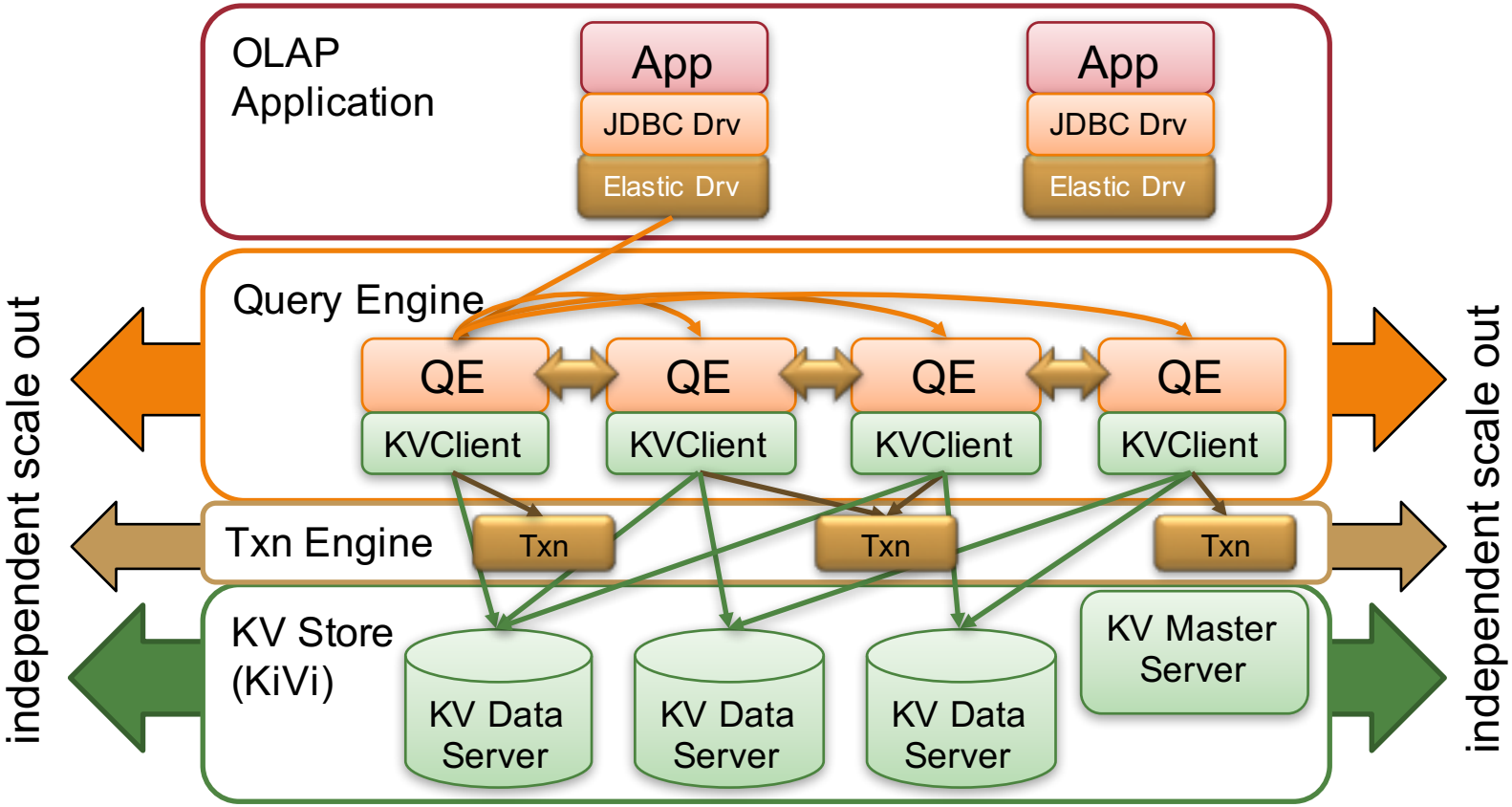


- Declarative languages
  - Optimization, caching, indexing
- Transactions
  - Strong consistency
- Shared-nothing cluster architecture
  - Scale out
  - Parallelism
- High availability in the cloud
  - Replication
- Data streaming
  - Online stream processing

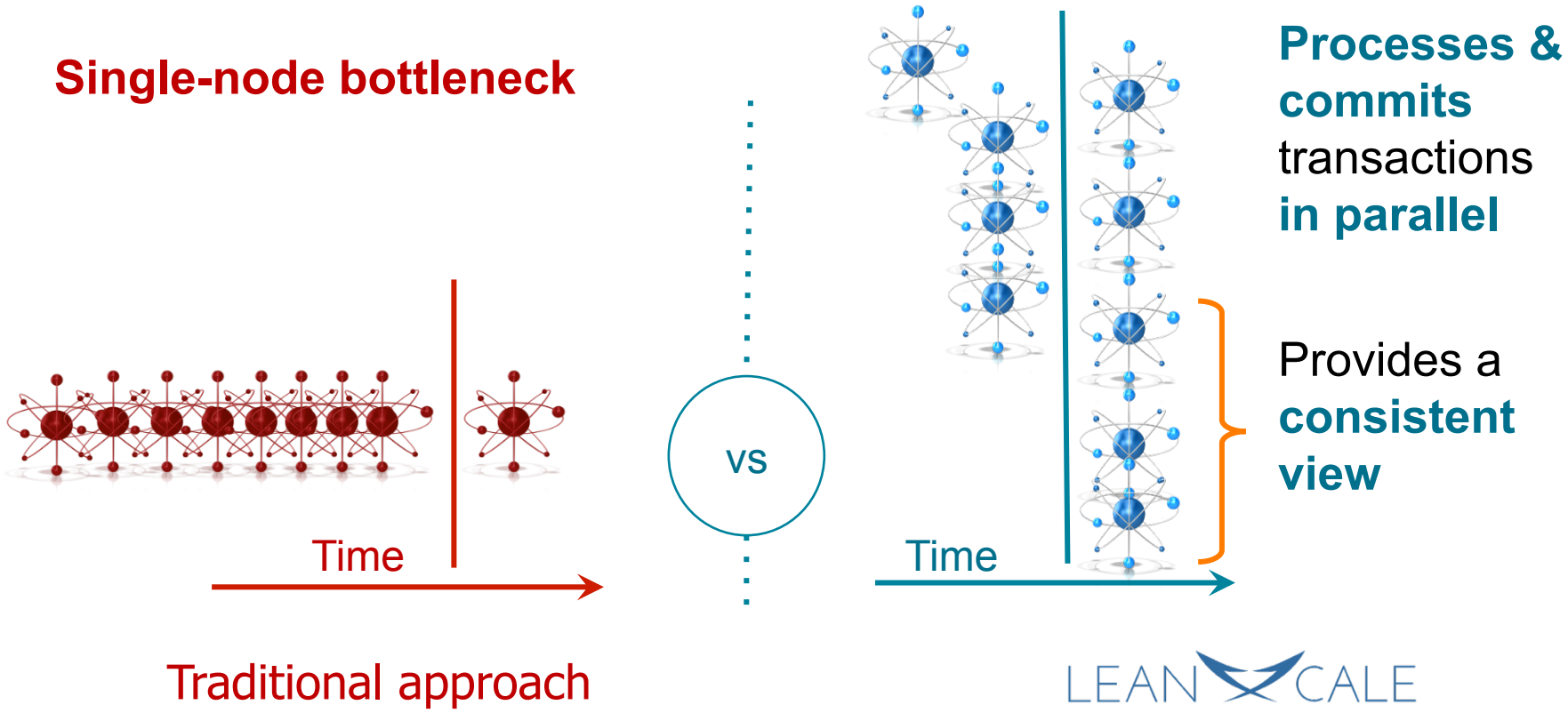
Main Techniques

- From SQL
  - Parallel, in-memory query processing
  - Fault-tolerance, failover and synchronous replication
  - Streaming
- From NoSQL
  - Key-value storage and access
  - JSON data support
  - Horizontal and vertical data partitioning (sharding)
- New
  - Scalable transaction management
  - Polyglot language and polystore
    - Access to SQL, NoSQL and HDFS data stores

Distributed Architecture



Scalable Transaction Processing*

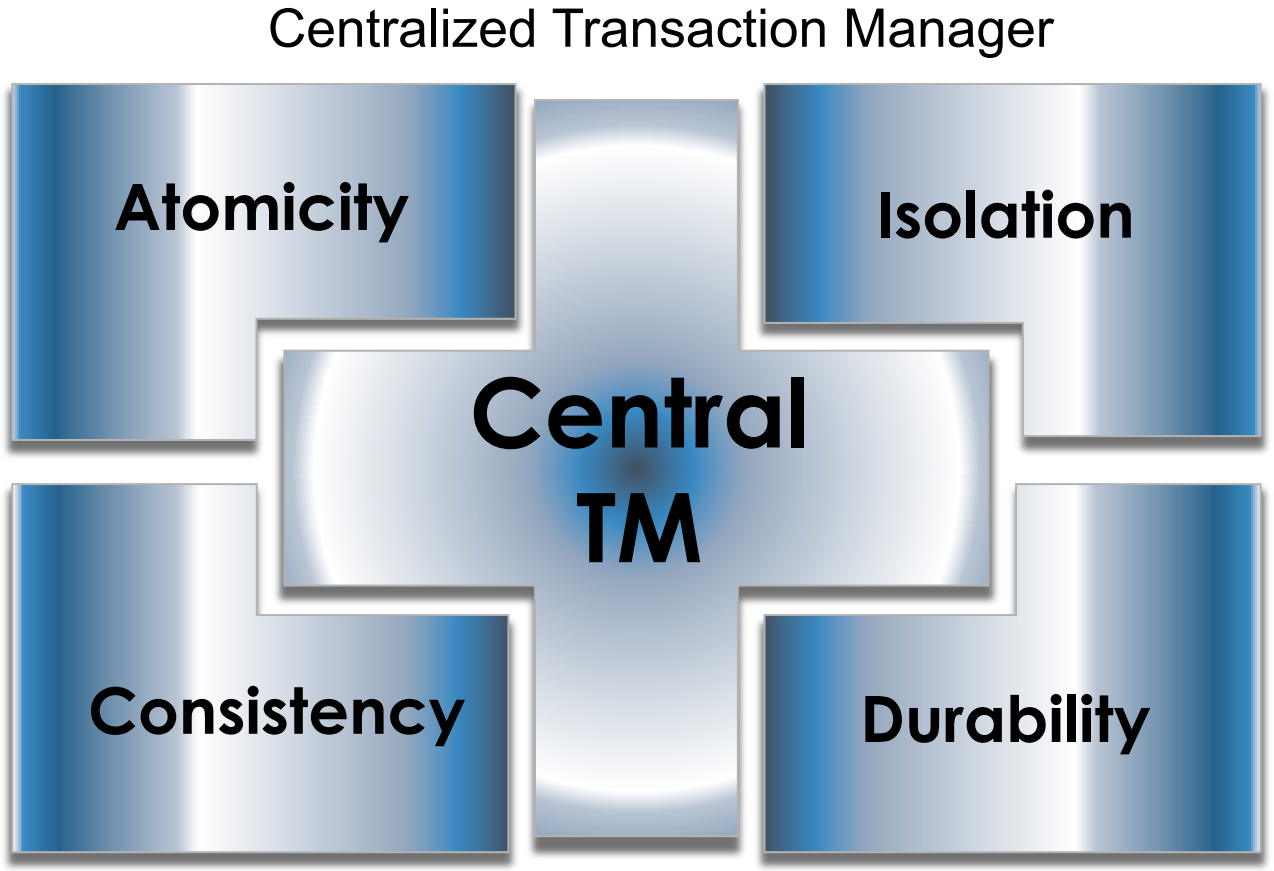


Processes & commits transactions **in parallel**

Provides a **consistent view**

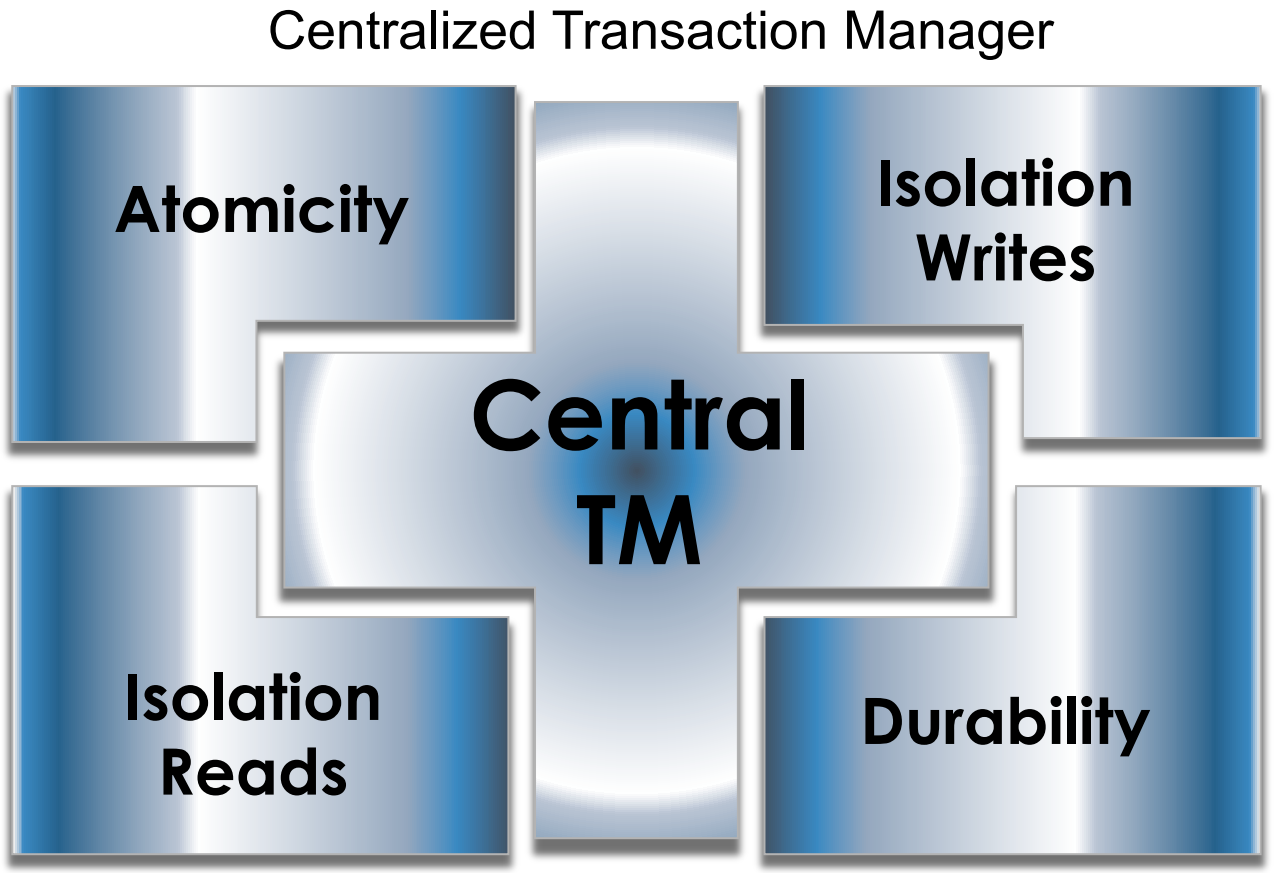
* R. Jimenez-Peris, M. Patiño-Martinez. System and method for highly scalable decentralized and low contention transactional processing. Priority date: 11th Nov. 2011. European Patent #EP2780832, US Patent #US9,760,597.

Traditional Approach



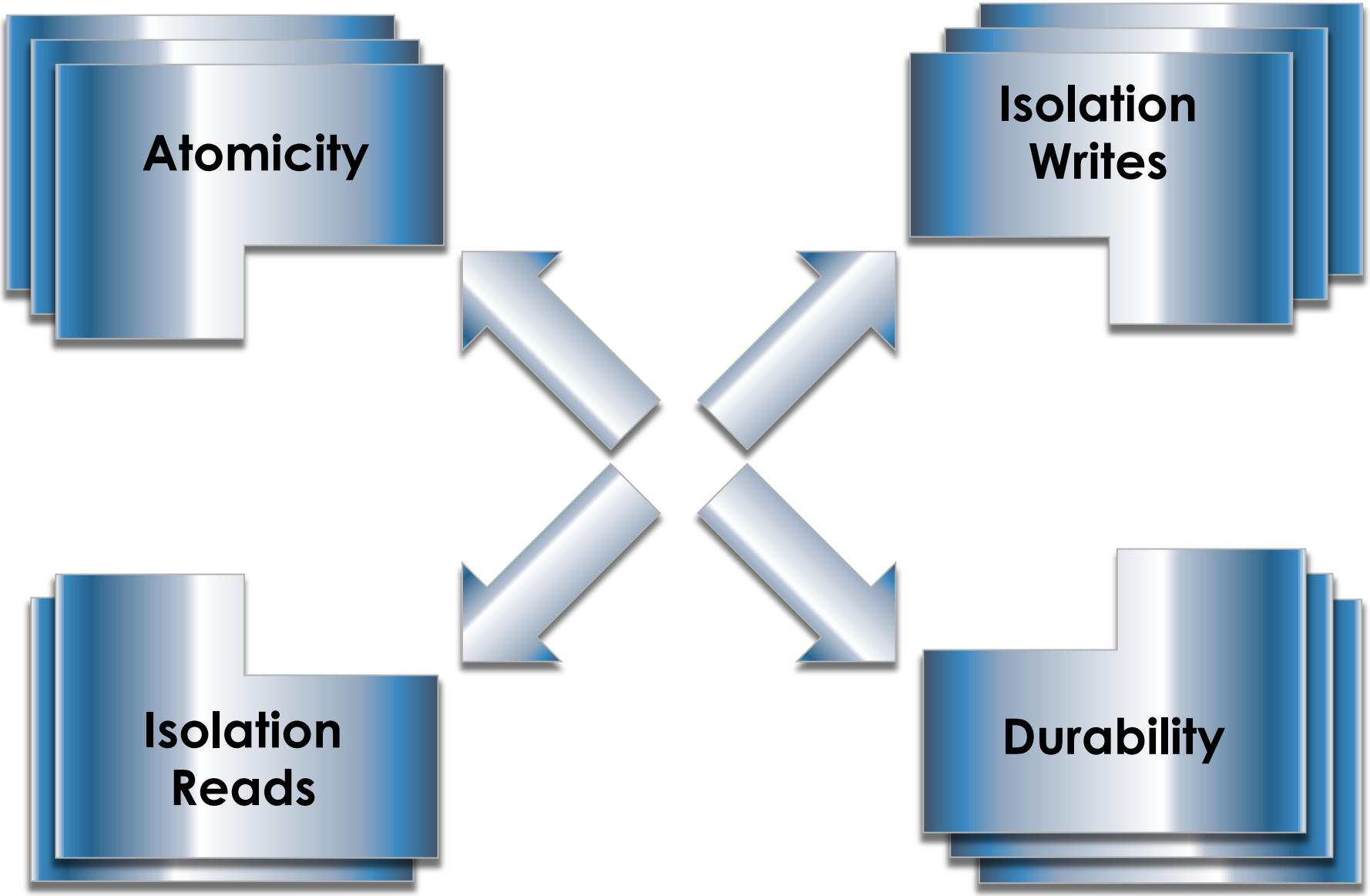
Single-node bottleneck

Traditional Approach

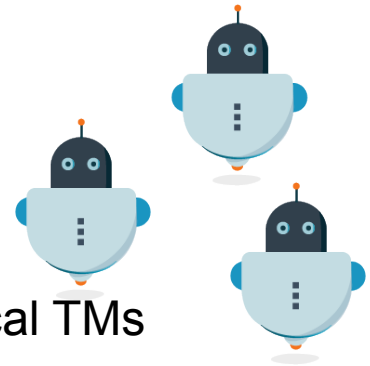


Single-node bottleneck

Scaling ACID Properties

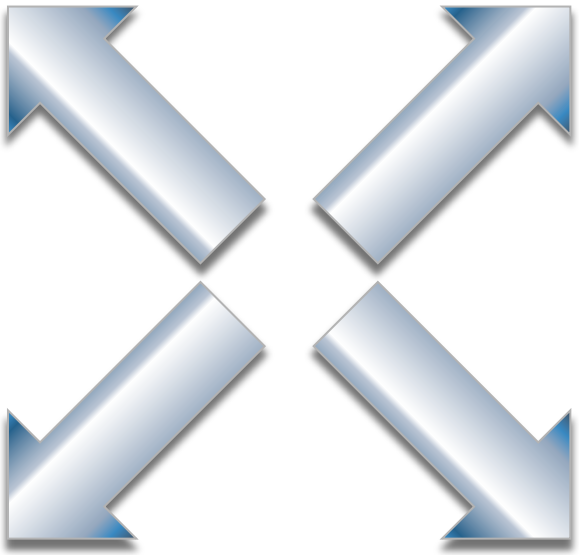


Scaling ACID Properties



Local TMs

Atomicity



Conflict managers

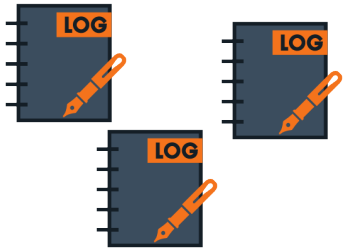
Isolation
Writes

Isolation
Reads

Snapshot server

Commit sequencer

Durability

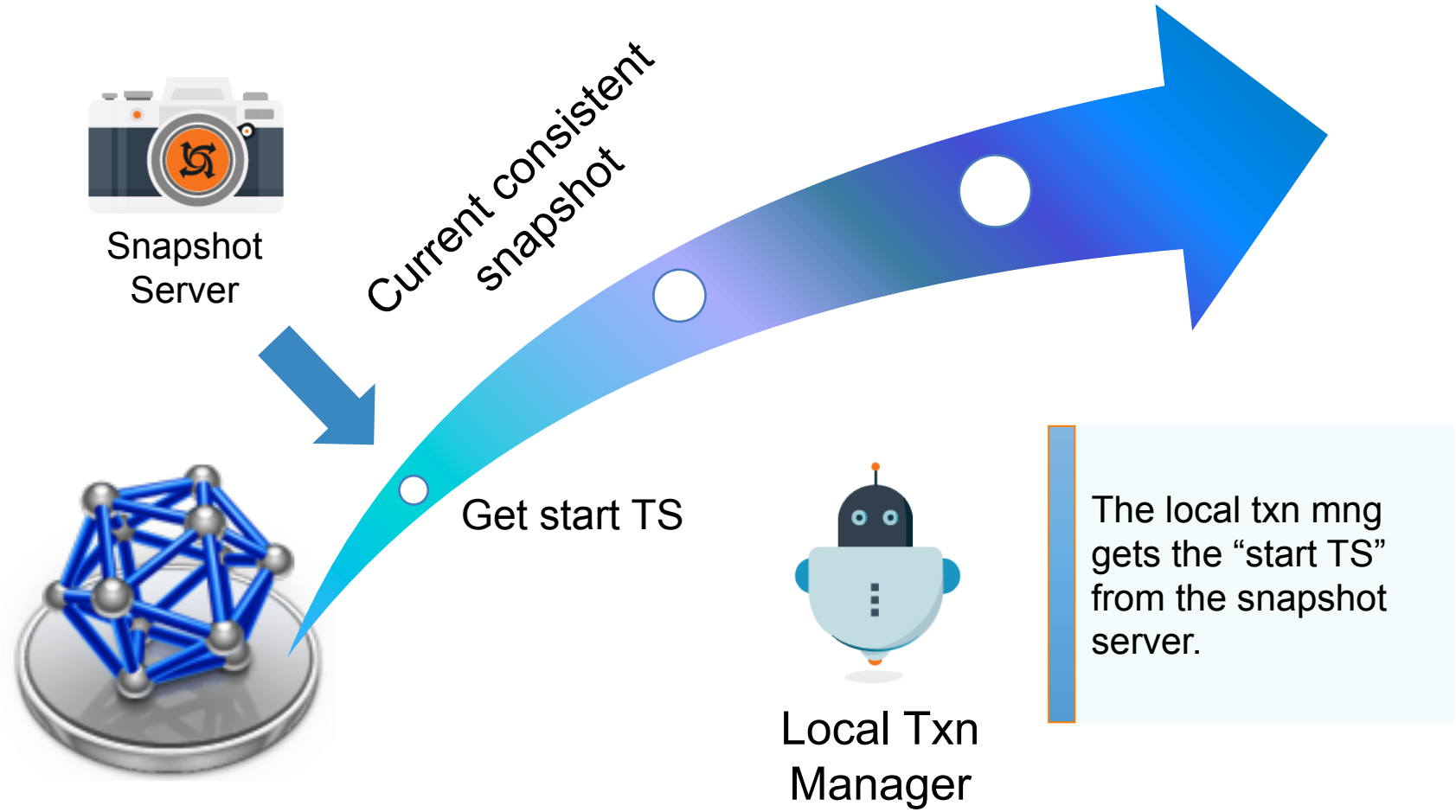


Loggers

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

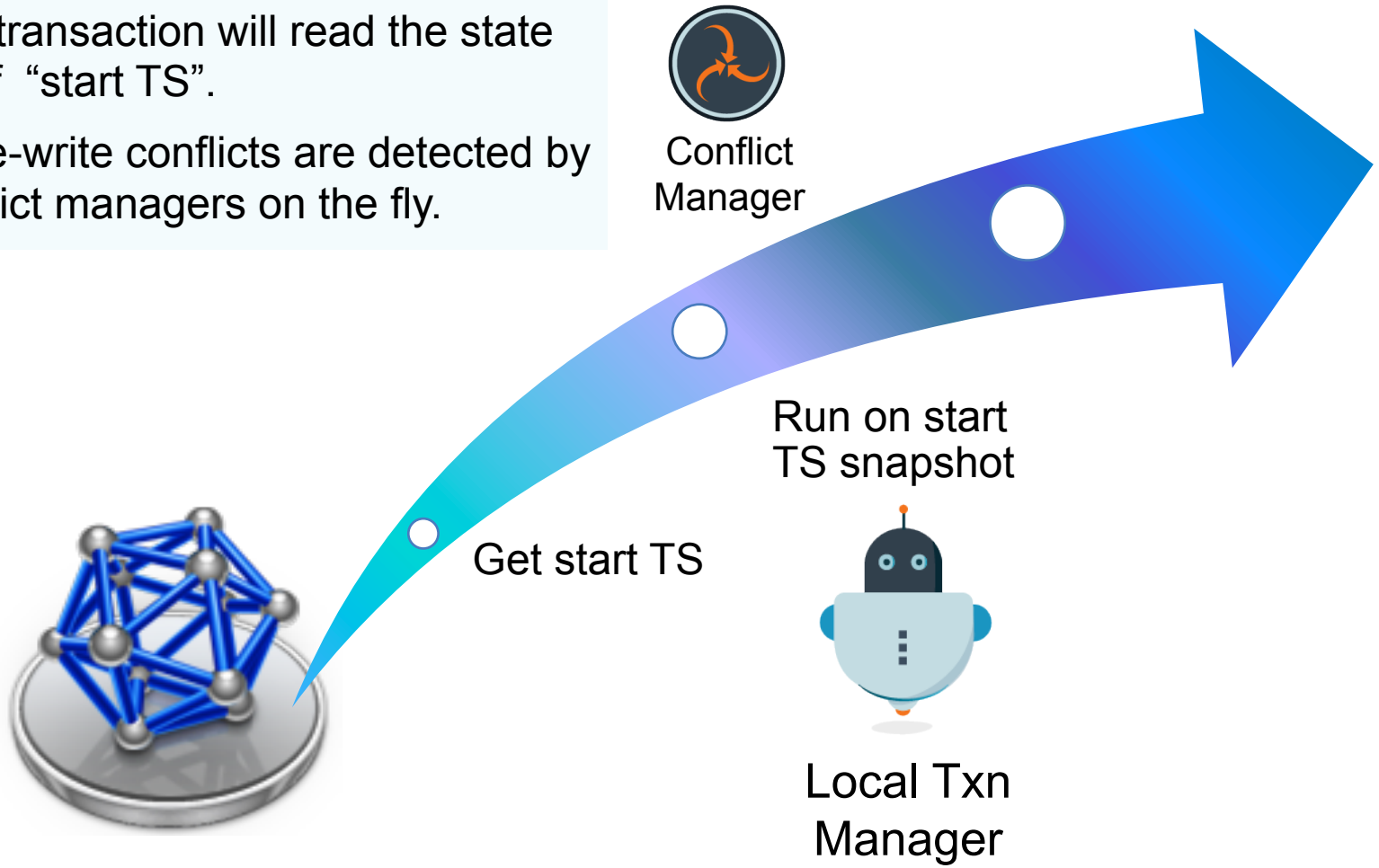
Transactional Life Cycle: start



Transactional Life Cycle: execution

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

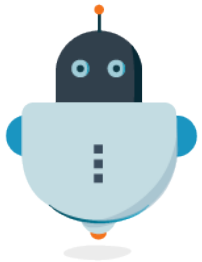
Write-write conflicts are detected by conflict managers on the fly.



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

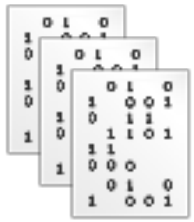
Writeset

Writeset

Commit TS

Commit Sequencer

Logger

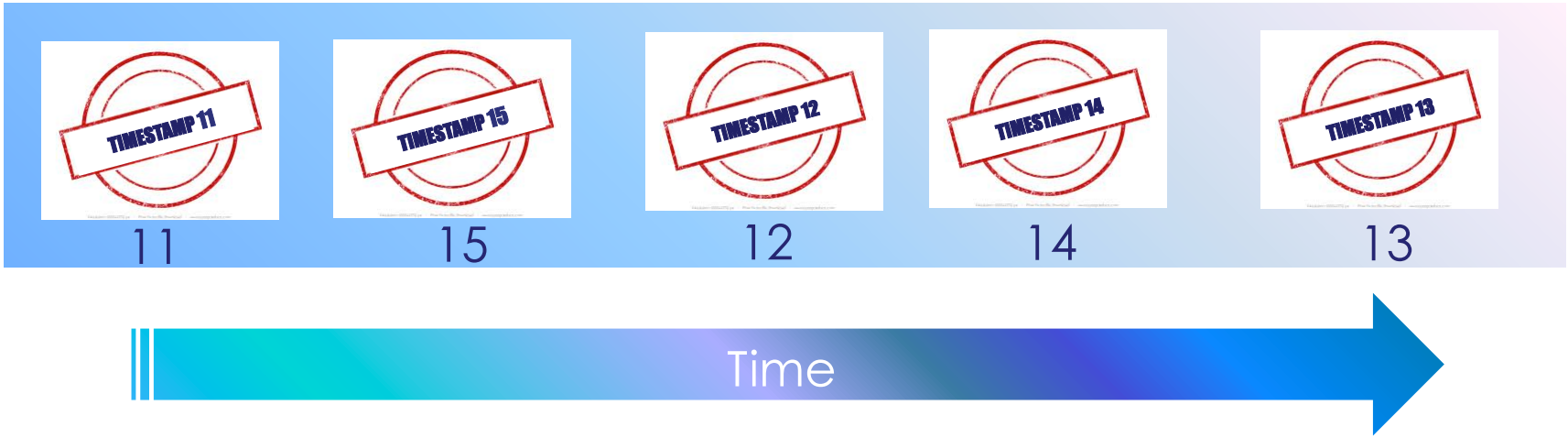


Data Store

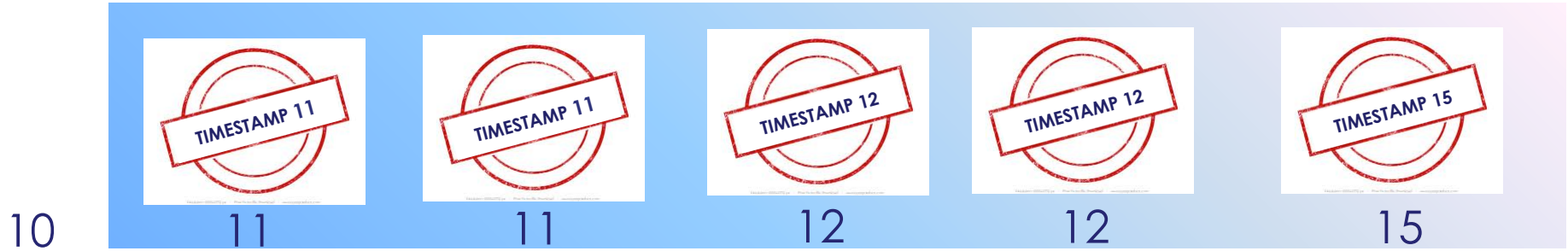
Snapshot Server

Transaction Life Cycle: commit

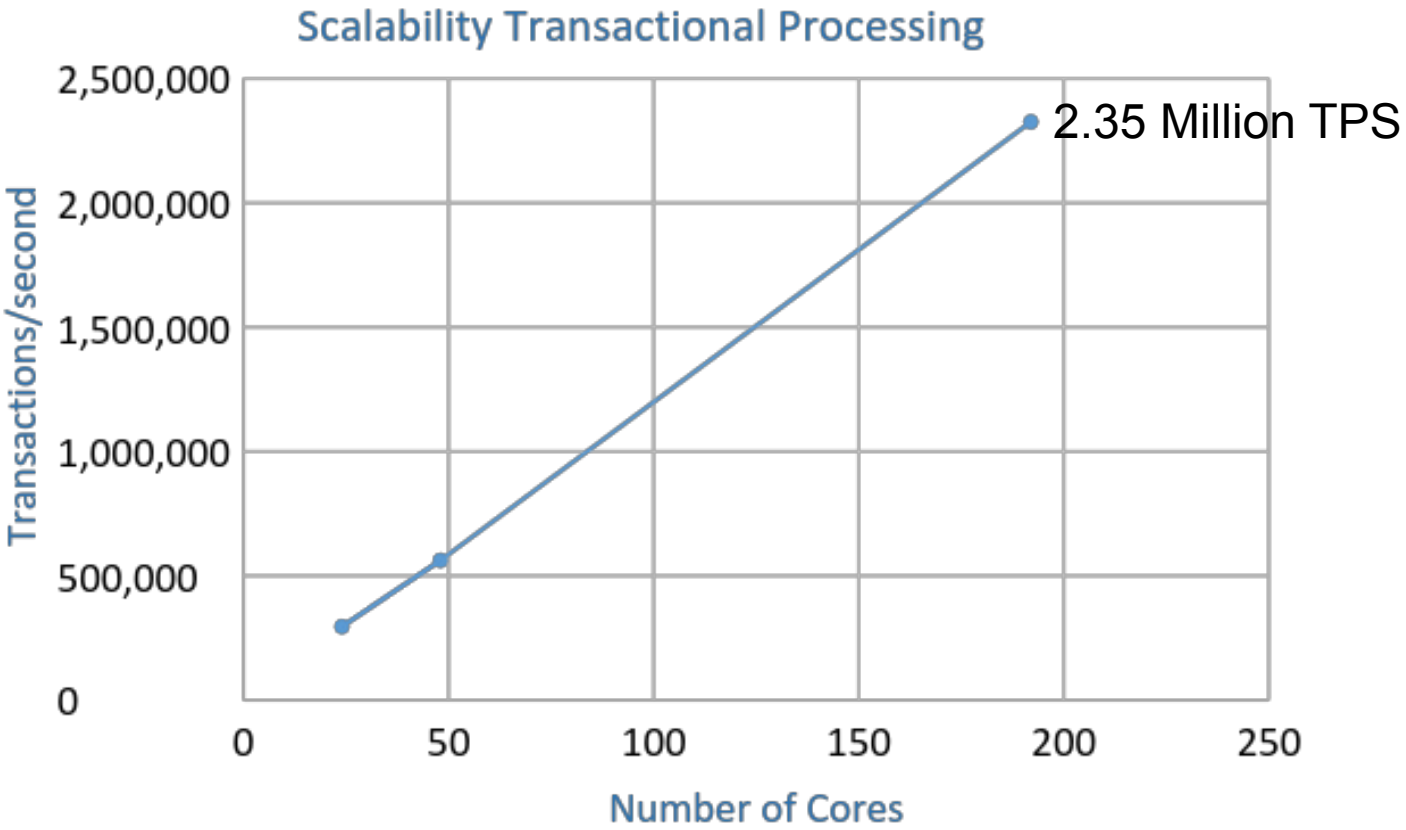
Sequence of commit timestamps received by the Snapshot Server



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



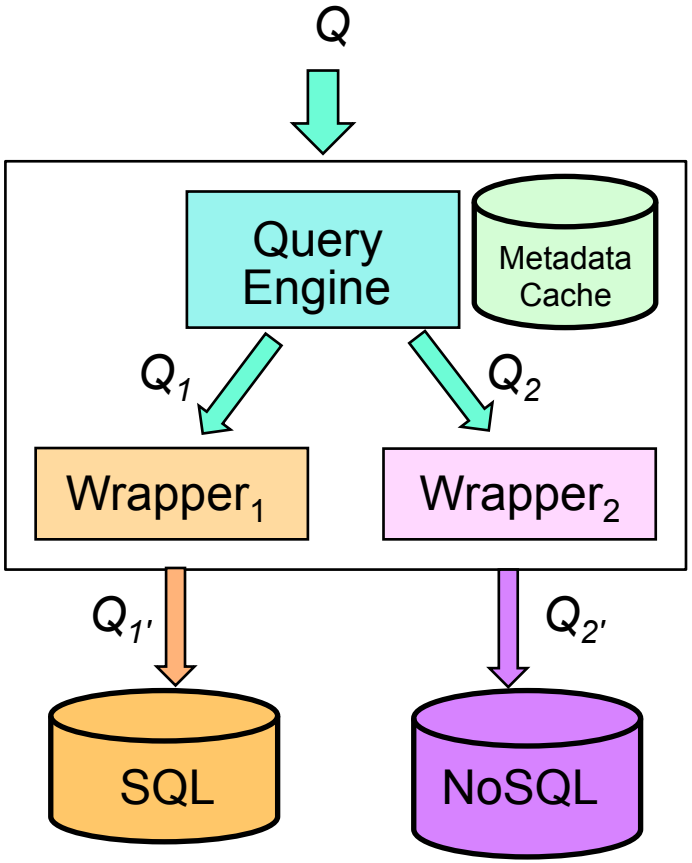
Transactional Scalability



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

Polystore

- Goal
  - Integrated access to heterogeneous data stores such as SQL, NoSQL, HDFS, and CEP
- Query engine
  - Transforms queries into sub-queries for wrappers
  - Integrates (computes) the results of sub-queries
- Wrapper
  - Transforms sub-queries into sources' languages
  - Transforms the results in the QE format



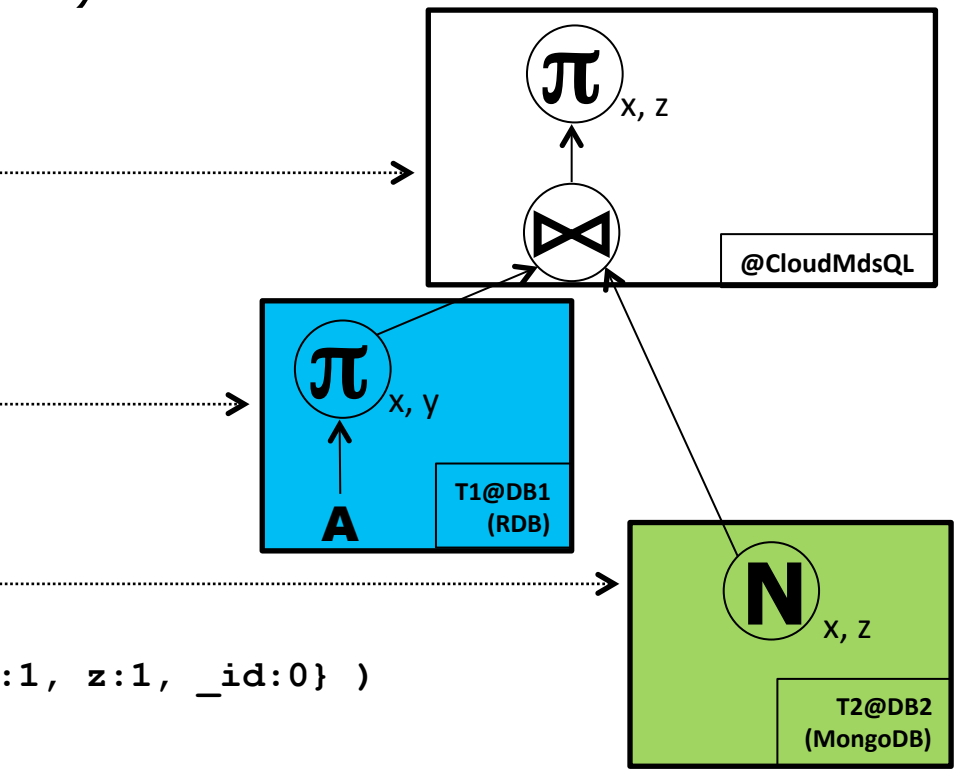
Polyglot Query Example

- 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.

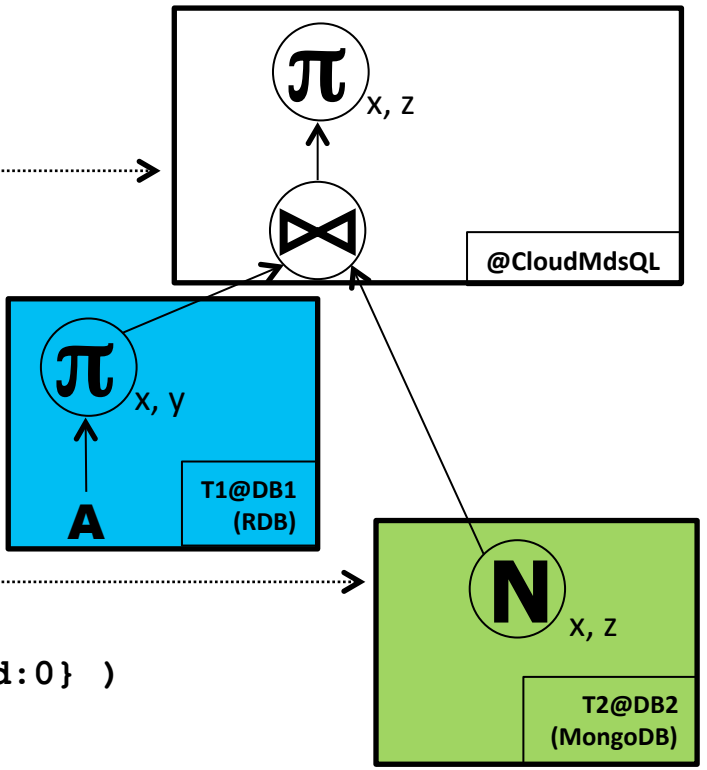
Polyglot Query Example

- CloudMdSQL = SQL + (native) 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} )
*}
```

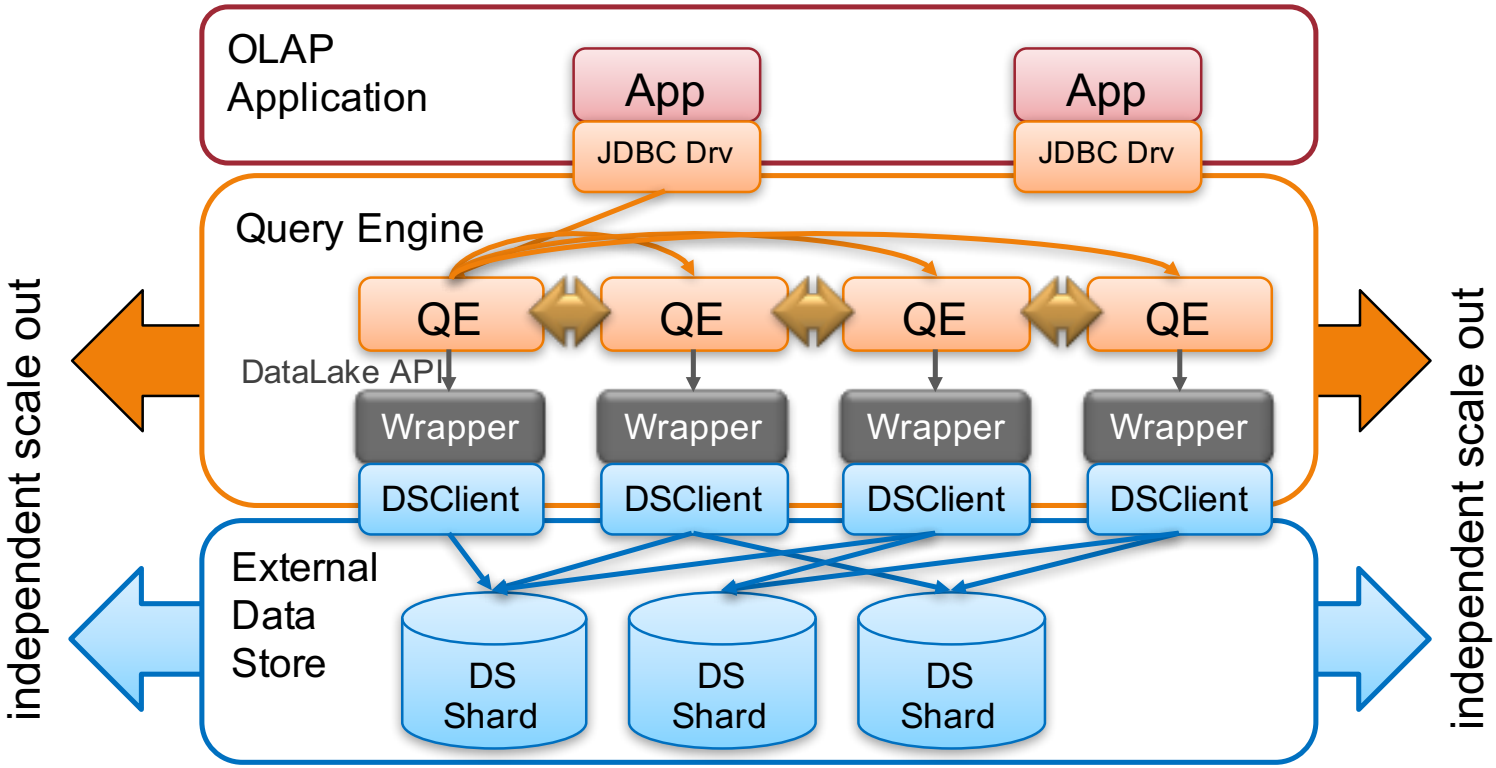


Parallel Polystore Query Processing

- 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

- *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.

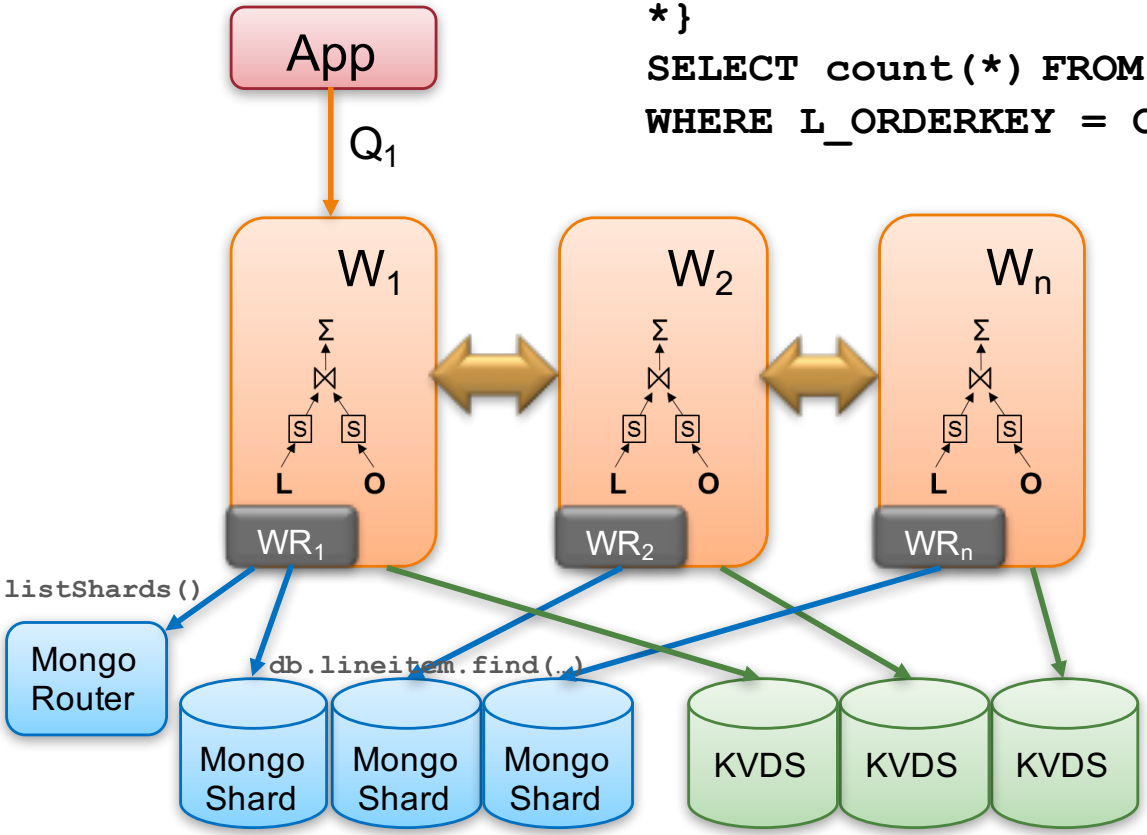
LeanXcale Polystore Architecture



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

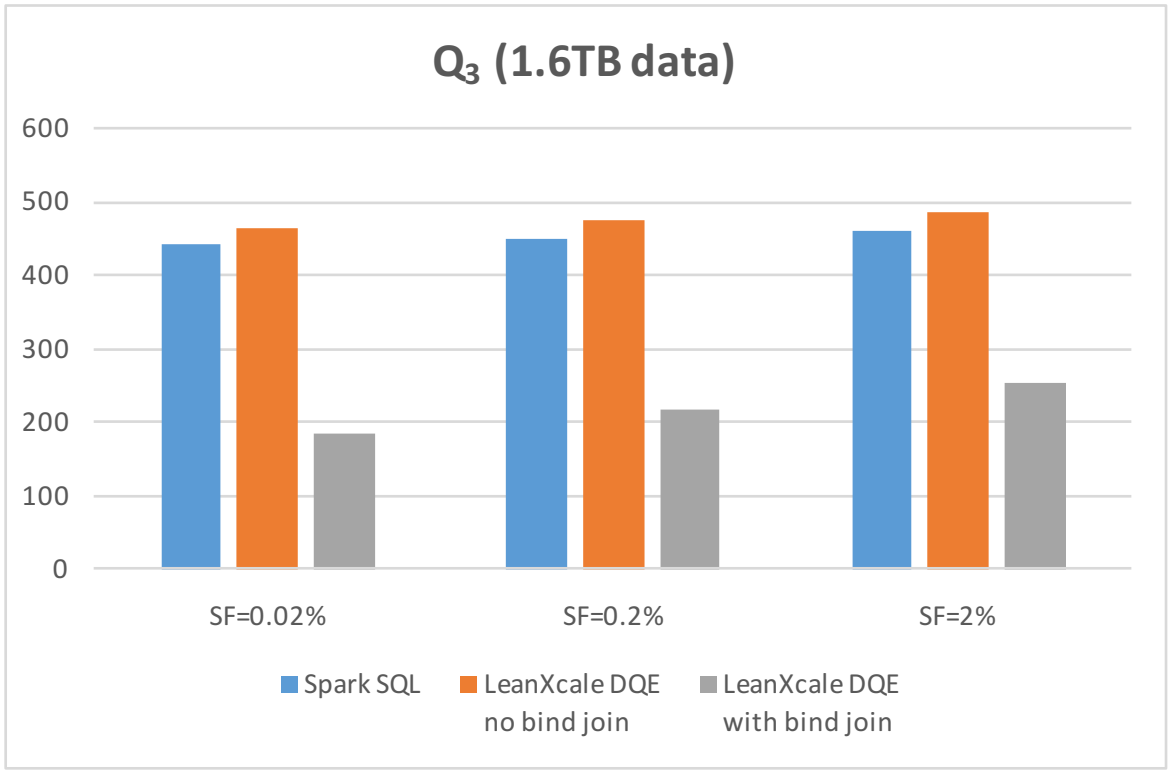
Query on LeanXcale and MongoDB

```
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
```



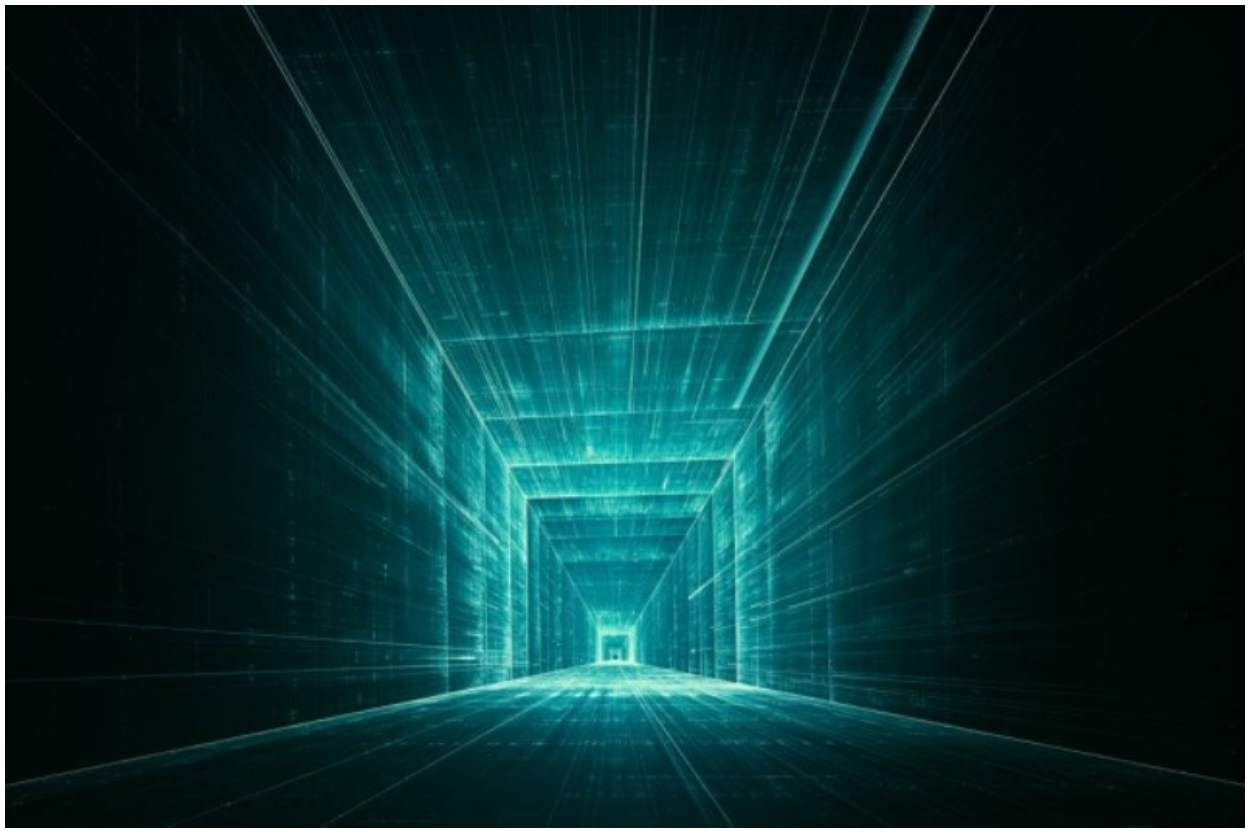
Performance Evaluation

- 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

Taxonomy of NewSQL Systems



SQL versus NoSQL versus NewSQL

	SQL	NoSQL	NewSQL
Data model	Relational	Specialized: KV, document, graph	Relational
Query language	SQL	Key-value API New query language	SQL Key-value API
Scalability	Only for high-end (Teradata, Exadata)	By design (SN cluster)	By design (SN cluster)
Consistency	Strong	Limited	Strong
Big data ecosystem	External tables (HDFS)	Integration within Hadoop	Integration within Hadoop
Workload	OLAP XOR OLTP	OLTP	OLAP, OLTP, HTAP
Polystore	SQL or HDFS data sources	SQL or HDFS data sources	SQL, NoSQL, HDFS data sources

Taxonomy Definition

- Why a taxonomy?
  - Different flavors of NewSQL systems, for different workloads
- Main dimensions
  - Transaction model
  - Scalability
  - Storage engine
  - Query parallelism
  - Polystore

Transaction Model

- Ad hoc
  - ACID properties partially provided
  - Isolation and atomicity not guaranteed
- A priori-knowledge
  - Requires to know which rows will be read and written before executing the transaction
- ACID
  - ACID properties fully provided
  - Isolation levels well such as serializability or snapshot isolation

Scalability

- Bounded scalability
  - Centralized transaction manager: can scale out as far as it does not get overloaded
- Logarithmic scalability
  - ROWA replication: can scale out the read workload
  - Cache consistency: requires synchronization of the updated blocks
  - 2PC: to deal with multi-node transactions
- Linear scalability
  - Linear scale-out in shared-nothing cluster

Storage Engine

- Relational
  - Support of algebraic operators, such as predicate filtering, aggregation, grouping and sorting
- Read/write, key-value
  - Capability of reading and writing individual data items
  - May support range queries

Query Parallelism

- Inter-query parallelism
  - Multiple queries can be processed in parallel
- Intra-query parallelism
  - Inter-operator parallelism: different operators in the query plan can be processed on different nodes, but a single operator runs on a single node.
  - Intra-operator (SIMD) parallelism: the same operator can run across multiple nodes

Different Flavors of NewSQL Systems

- SQL+key-value
  - Use a key-value data store to scale data management, e.g. Splicemachine, EsgynDB and LeanXcale.
- HTAP
  - Systems that combine OLTP and OLAP, e.g. SAP Hana, EsgynDB and LeanXcale.
- In memory
  - Optimized for processing the workload fully in main memory, e.g. SAP Hana and MemSQL
- New transaction managers
  - New, scalable approach to transaction management, e.g. Spanner and LeanXcale

Some NewSQL Systems

Vendor	Product	Objective	Comment
Google	Spanner	OLTP	Google cloud distributed database service. Used by F1 for the AdWords app.
LeanXcale	LeanXcale	HTAP	HTAP DBMS with fast insertion, fast aggregation over real-time data and polystore capability
SAP	Hana	HTAP	The HTAP pioneer, based on in-memory, column store
MemSQL Inc.	MemSQL	HTAP	In-memory, column and row store, MySQL compatible
Esgyn	EsgynDB	HTAP	Apache Trafodion for OLTP, Hadoop for OLAP
NuoDB	NuoDB	OLTP	Distributed SQL DBMS with P2P architecture
Splice Machine	Splice Machine	HTAP	HBase as storage engine, Derby as OLTP query engine and SparkQL as OLAP query engine

Google Spanner

- Globally distributed database service in Google Cloud
  Synchronous replication between data centers with Paxos
  - Load balancing between Spanner servers
    - Favor the geographical zone of the client
- Different levels of consistency
  - ACID transactions
  - *Snapshot* (read only) transactions
    - Based on data versioning
  - Optimistic transactions (read without locking, then write)
    - Validation phase to detect conflicts and abort conflicting transactions
- Two interfaces
  - SQL
  - NoSQL key-value interface
- Hierarchical relational storage
  - Precomputed joins

LeanXcale

- SQL DBMS
  - Access from a JDBC driver
  - Polyglot language with JSON support (pending)
- Key-value store (KiVi)
  - Fast, parallel data ingestion
  - Multistore access: HDFS, MongoDB, Hbase, …
- OLAP parallel processing (Query Engine)
  - Based on Apache Calcite
- Ultra-scalable transaction processing (patented)
  - SQL isolation level: snapshot
  - Timestamp-based ordering and conflict detection just before commit
  - Parallel commits of transactions

Comparisons

System	Trans. model	Scalability	Storage	Query parallelism	Polystore
Spanner	ACID	Linear	Key-value	Inter-query	
LeanXcale	ACID	Linear	Relational key-value	Intra-query Intra-operator	SQL, NoSQL, HDFS
Hana	ACID	2PC	Relational Columnar	Intra-query Intra-operator	HDFS
MemSQL	Ad hoc	Log	Relational	Inter-query	HDFS (Spark connector)
EsgynDB	ACID	2PC	Key-value	Intra-query Intra-operator	HDFS
NuoDB	ACID	Log	Read/write	Inter-query	HDFS
Splice Machine	ACID	Centralized TM	Key-value	Intra-query Intra-operator	HBase, HDFS

Conclusion

- NewSQL is the hottest trend in database management:
- Scales out but without renouncing to SQL and ACID transactions.
- NewSQL has different roots leading to different flavours
- The taxonomy enables comparing NewSQL systems as well as SQL systems

Current Trends

Business Perspectives

- NewSQL is the database kind growing faster in the market
  - 33% CAGR, according to 451 Research market analyst in their Total Data Report
  - It will become soon an important player in the database landscape
- Early adopters of NewSQL will become leaders in database management
  - Faster database development with lower costs in engineering and lower needs in talent

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
  - NewSQL/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
- Benchmarking
  - Defining NewSQL/HTAP benchmarks and compare systems
  - Profiling to find new optimizations

References

1. T. Özsu, P. Valduriez. *Principles of Distributed Database Systems*. Fourth Edition. Springer, 2019.

2. 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.

3. 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 BigData*, 2018.

4. 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.

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