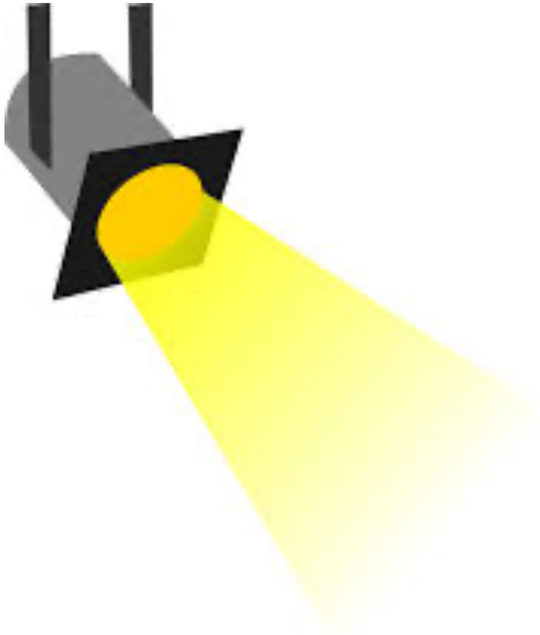
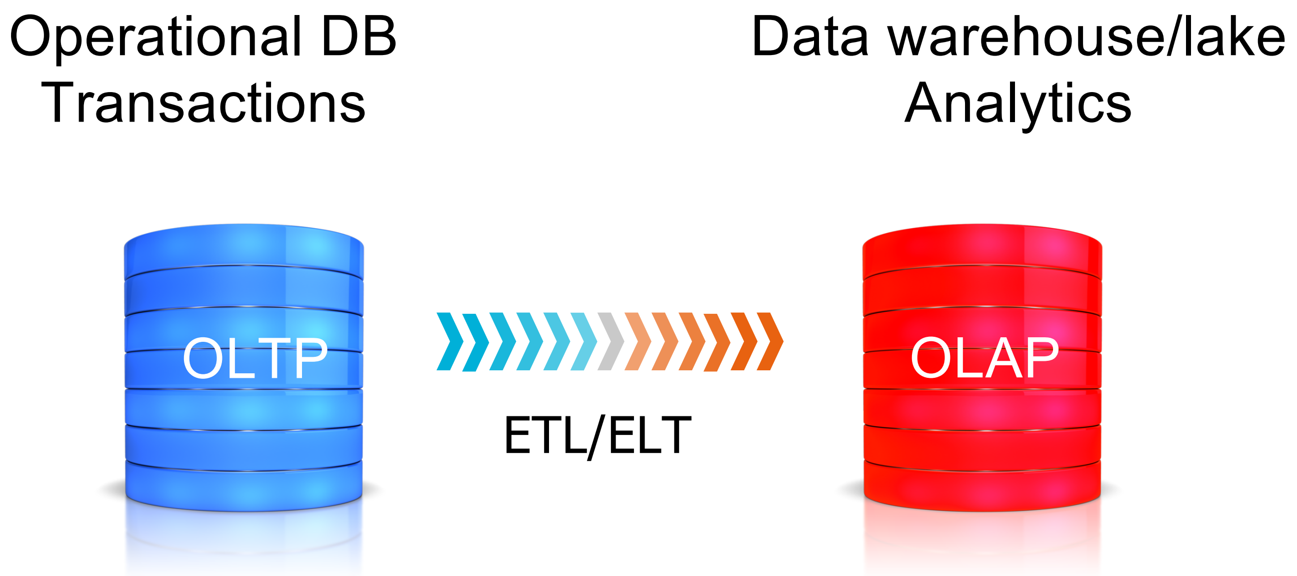
Spotlight on NewSQL



Why 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

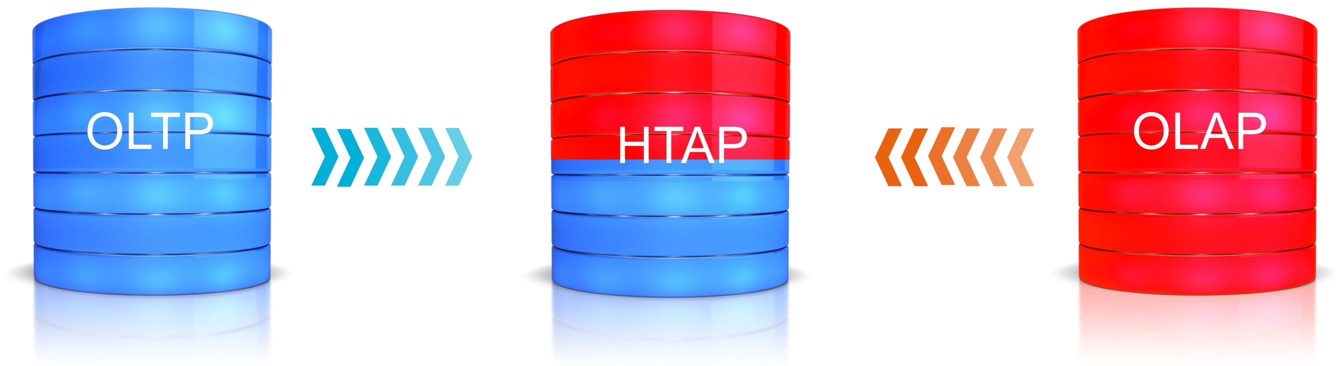
Transaction vs. Analytical Processing



- 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

Analytical queries
on operational data



- 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

Use Case: 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 Spanner
  - 30 billion search queries per month
  - 1 billion historical search events
  - Hundreds of Terabytes

Use Case: Network Monitoring

- NoSQL to store data at high rates
  - Data is put in a data store able to ingest data at very high rates
    - E.g. network performance monitoring information about packets sniffed in the network
- Problems
  - Because NoSQL is used to store the data, BI tools cannot be used for real time data
  - Data needs to be aggregated and exported periodically to an SQL database to query the data with BI tools

Use Case: 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

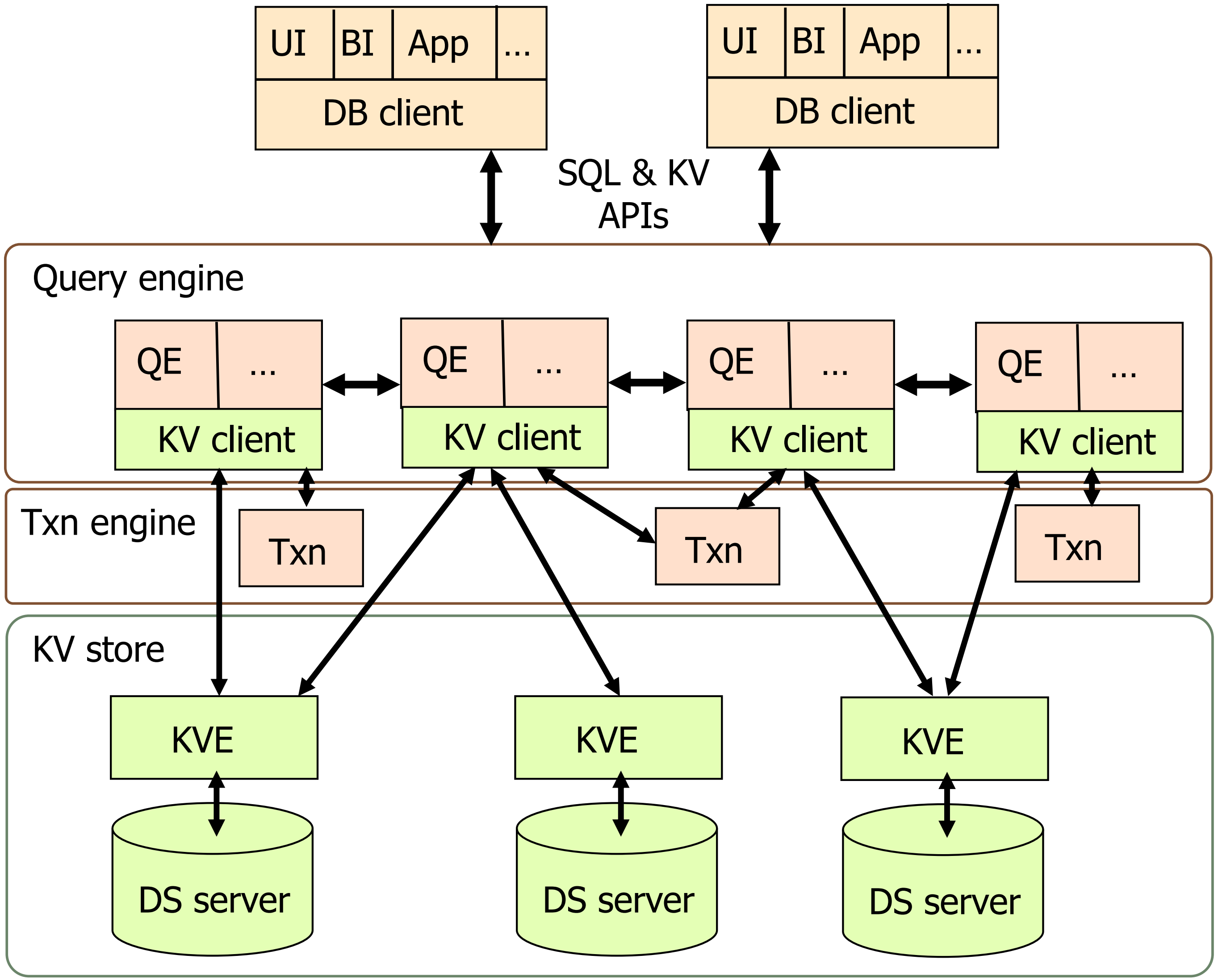
HTAP and Big Data

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

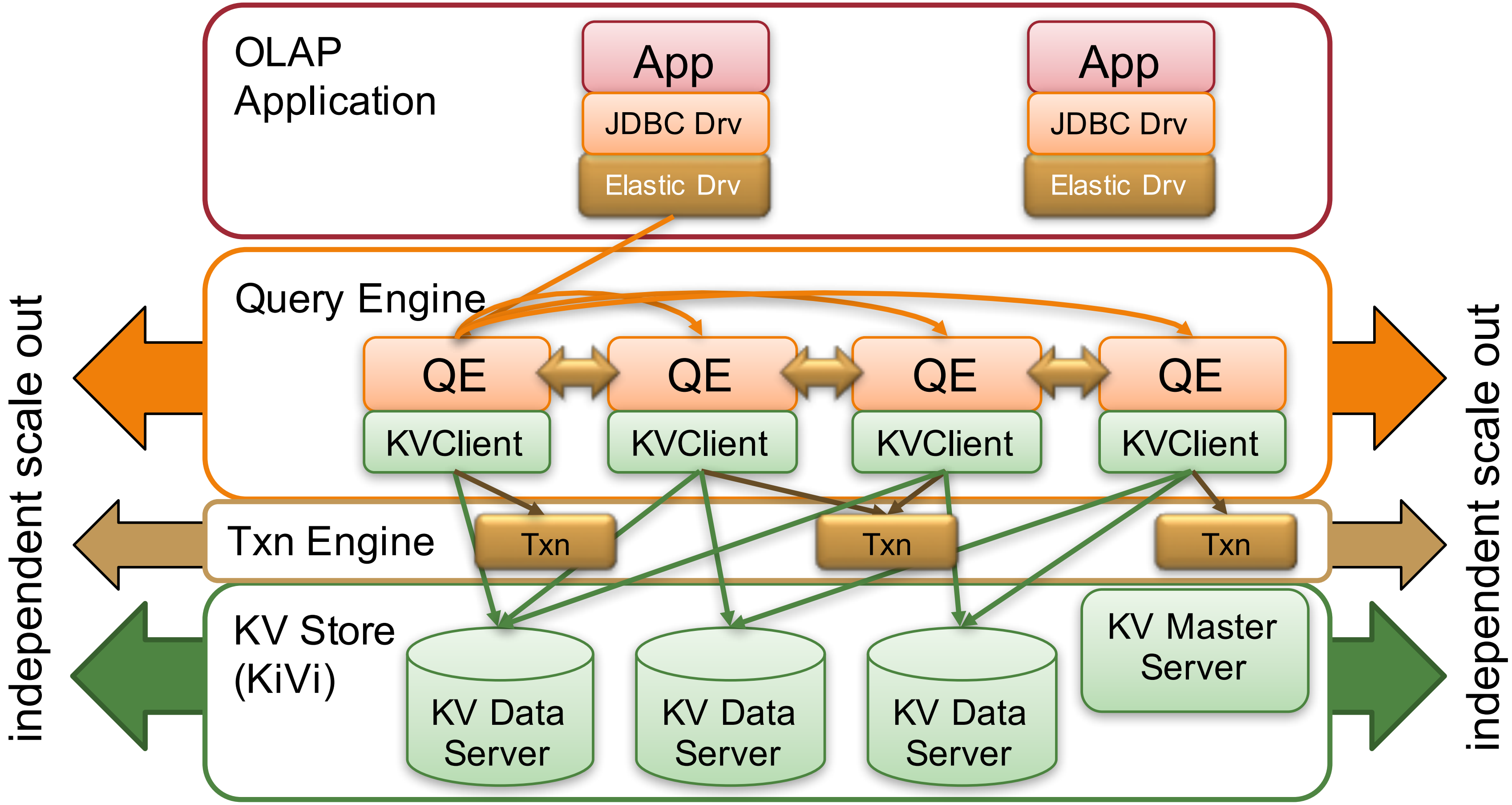
Main Techniques

- 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

NewSQL Distributed Architecture



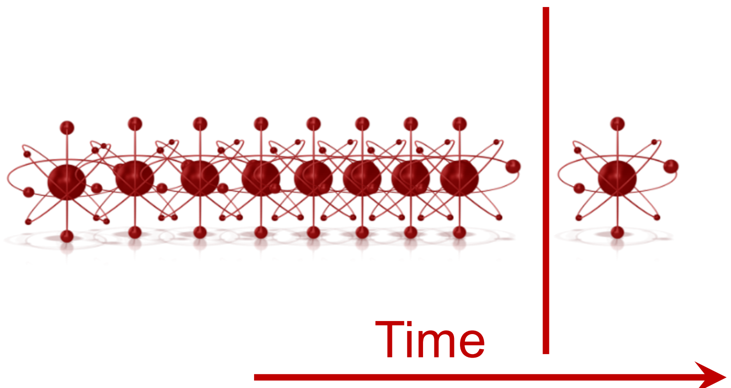
LeanXcale Architecture

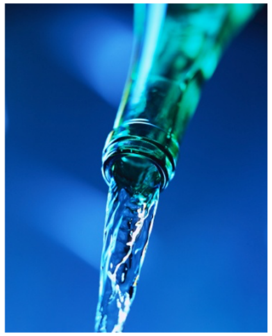


LeanXcale Scalable Transaction Processing*

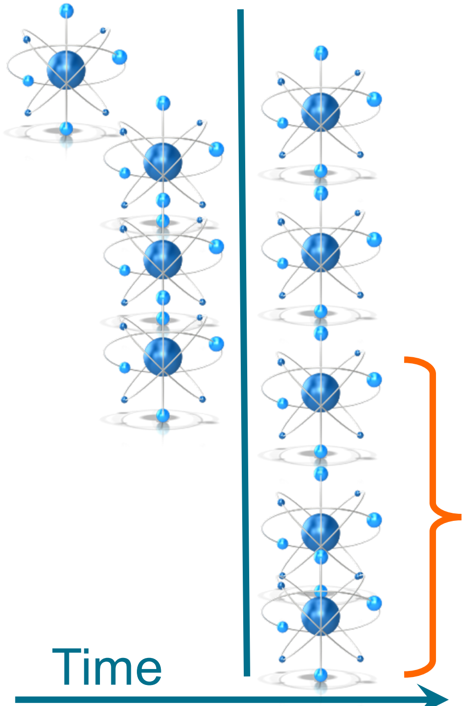
Traditional approach

Single-node bottleneck





vs



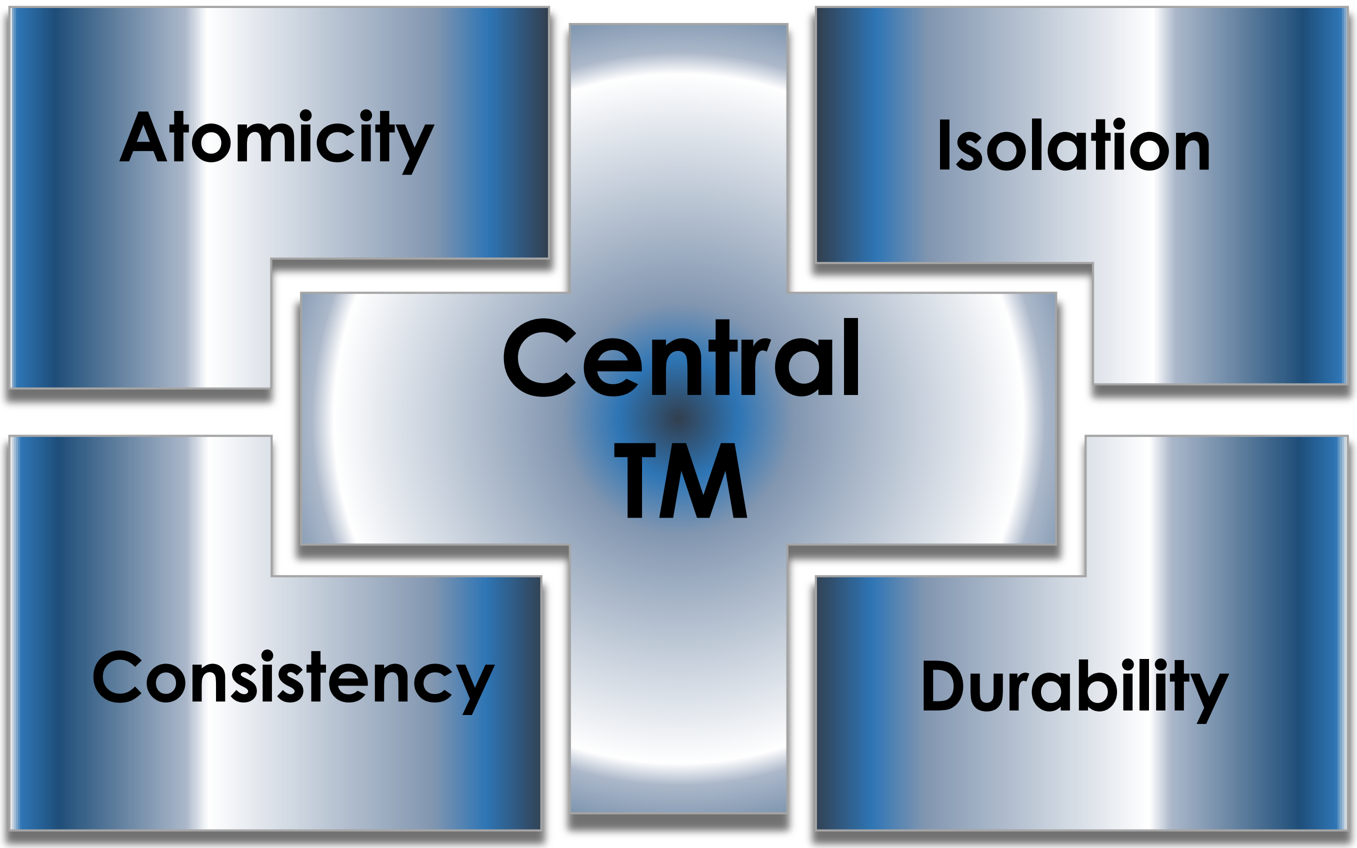
Processes & commits transactions **in parallel**

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

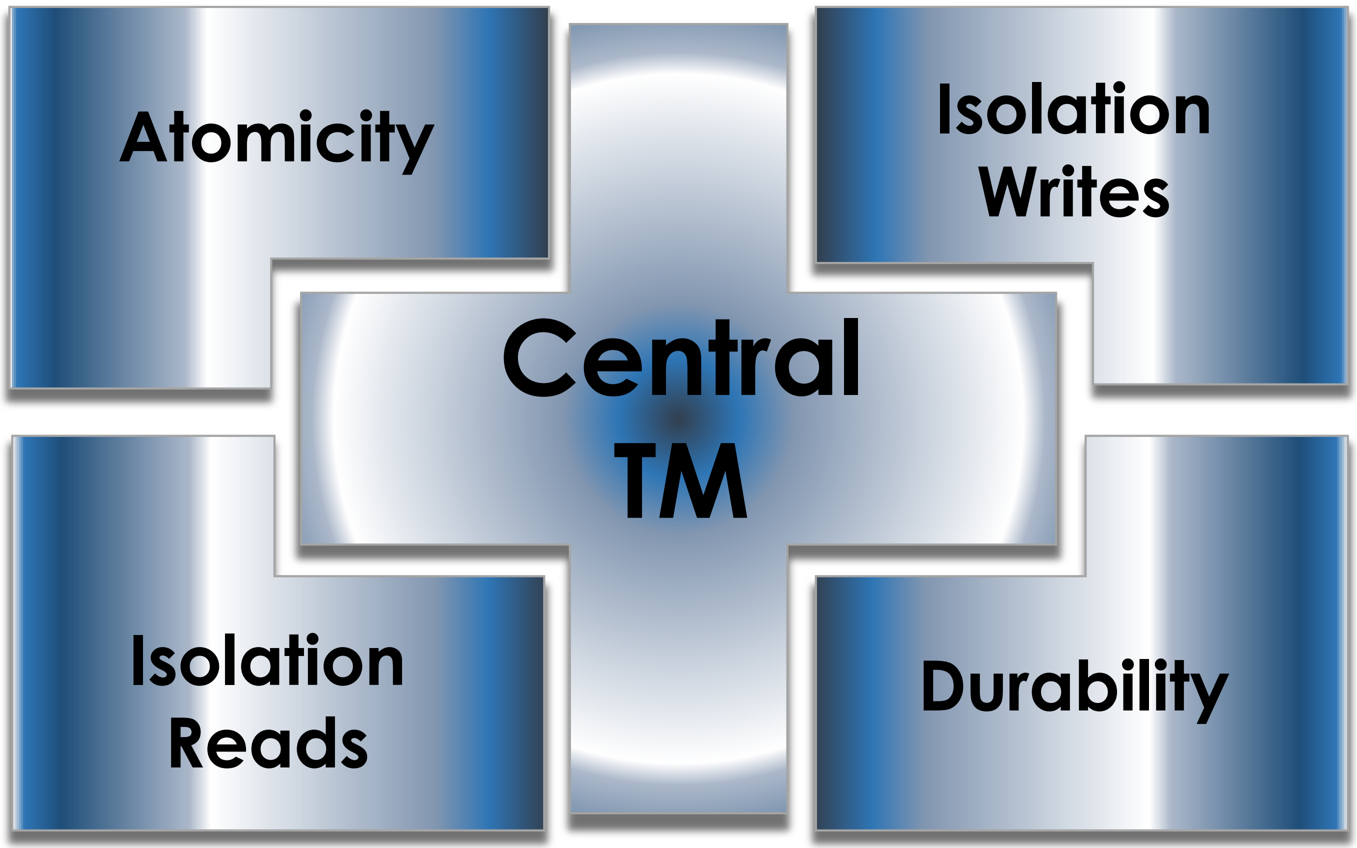
Centralized Transaction Manager



Single-node bottleneck

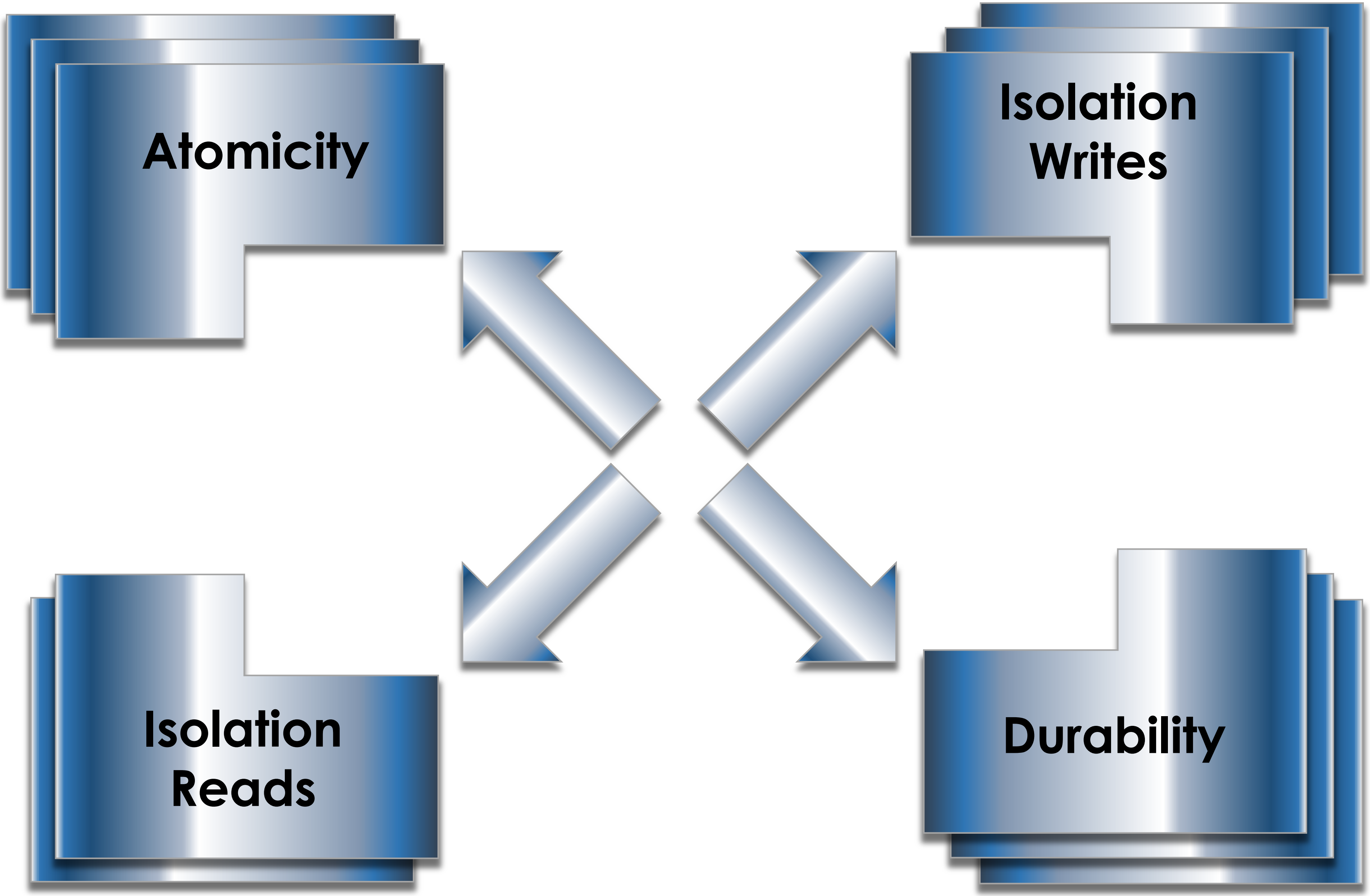
Traditional Approach

Centralized Transaction Manager

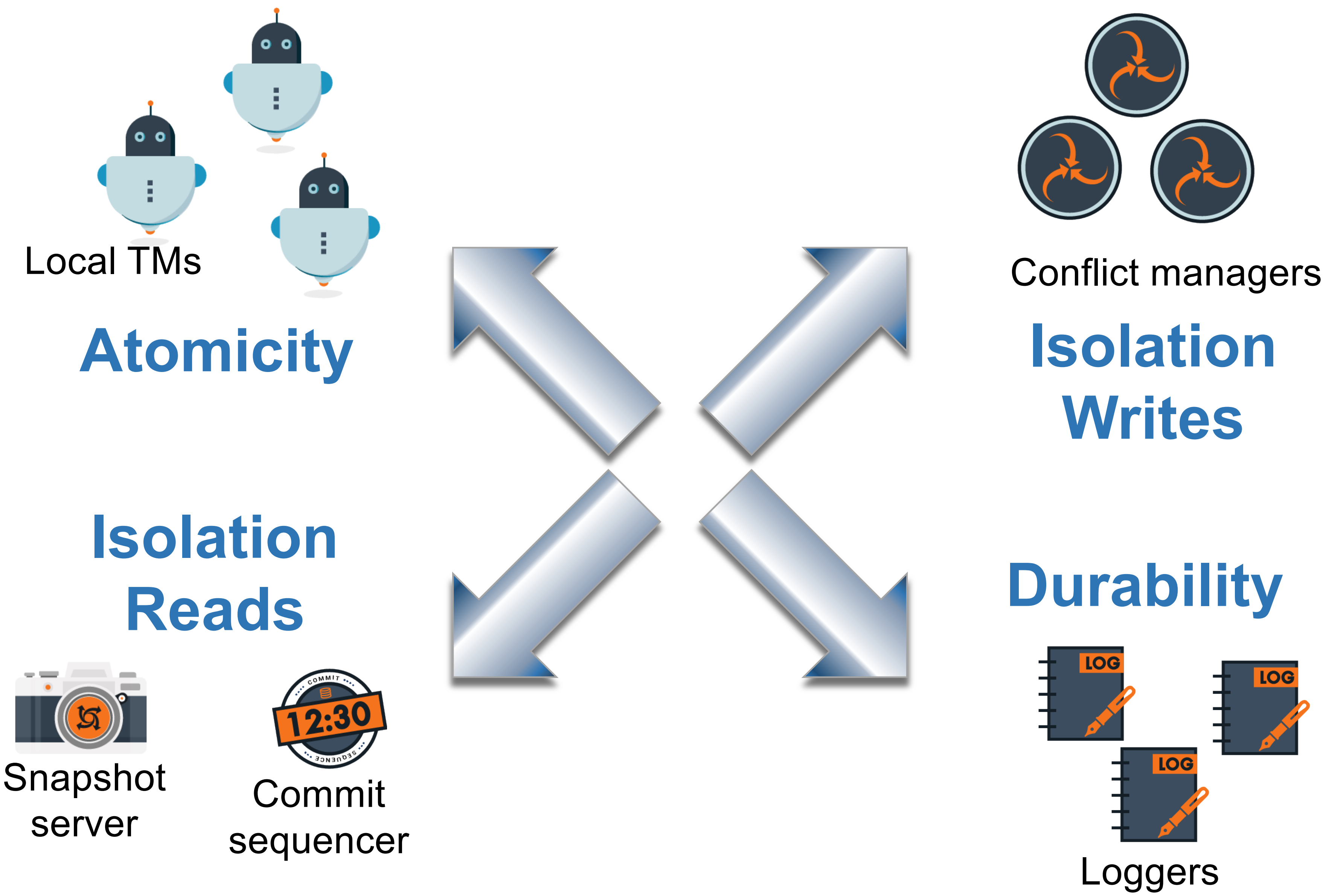


Single-node bottleneck

LeanXcale: Scaling ACID Properties



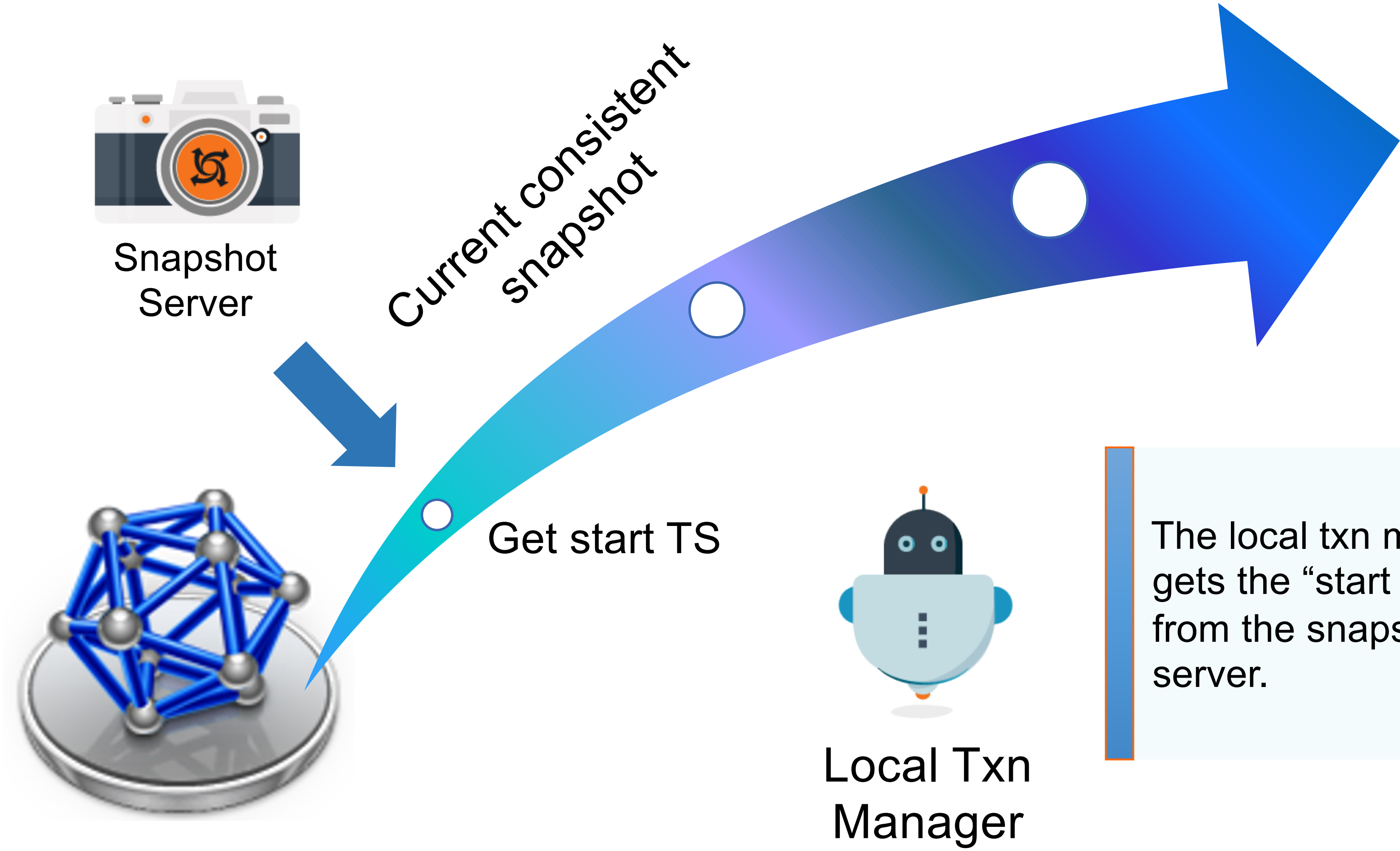
LeanXcale Scaling ACID Properties



LeanXcale Transaction Mgt 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

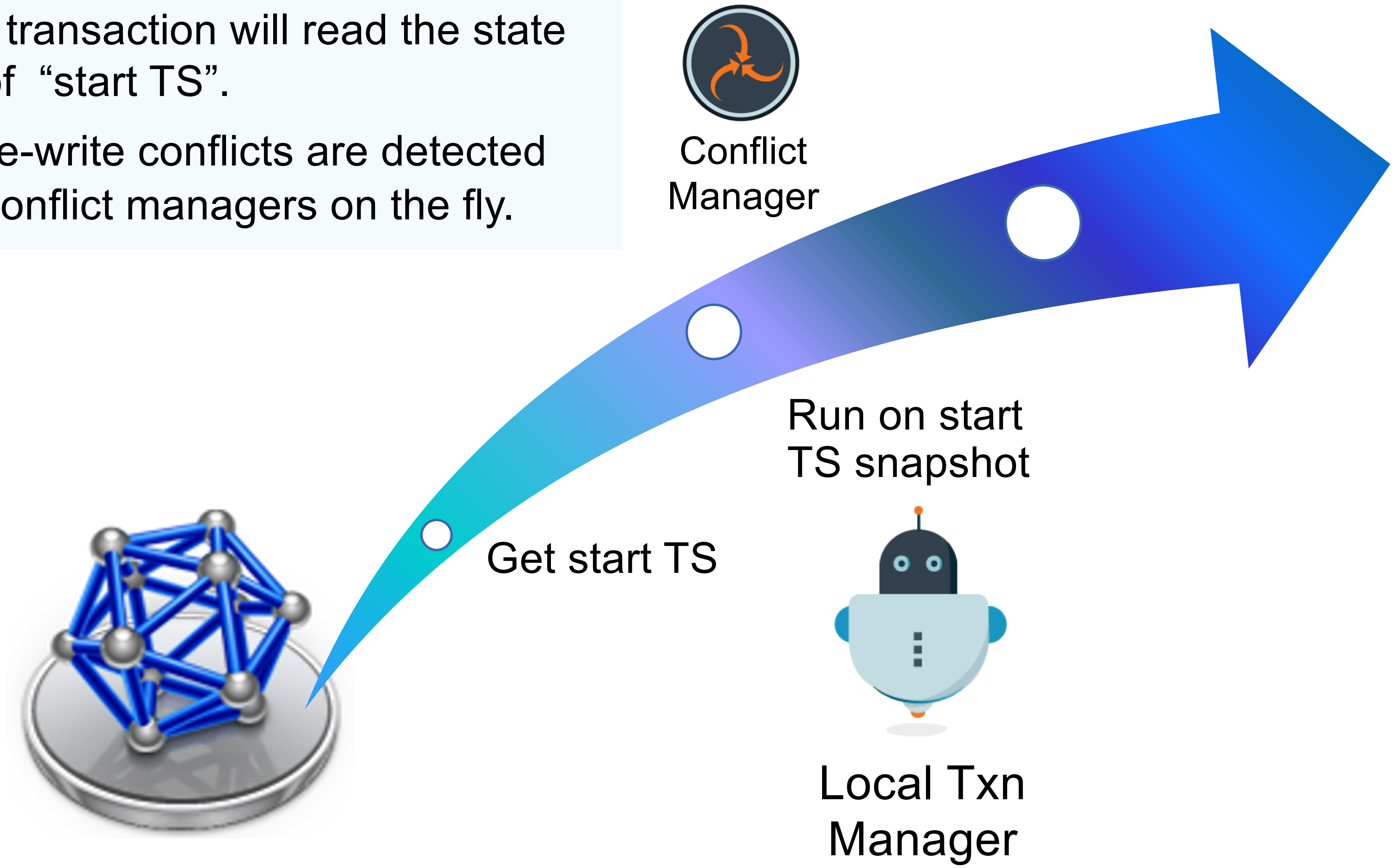


The local txn mng gets the "start TS" from the snapshot server.

Transactional Life Cycle: execution

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

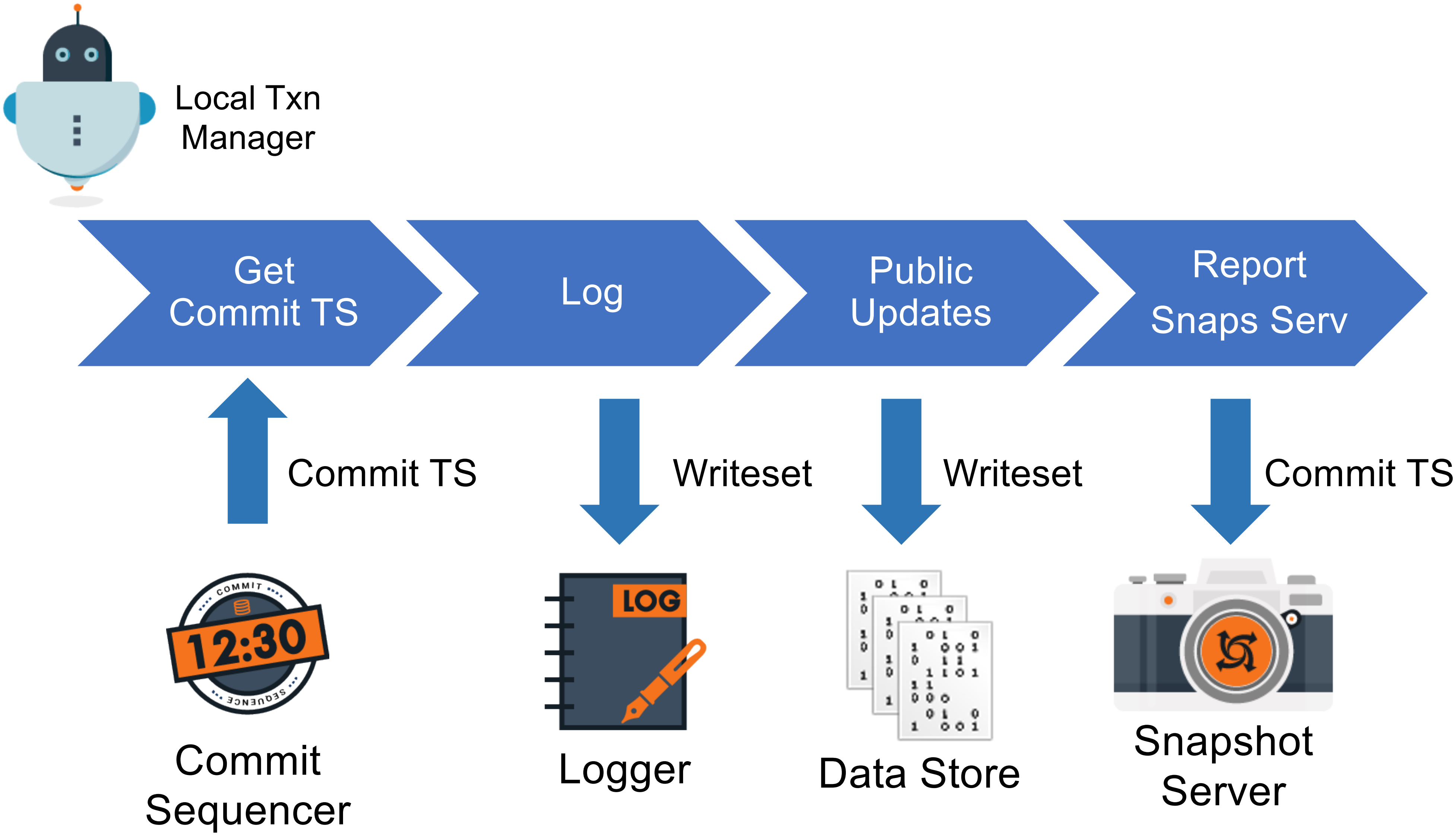
Write-write conflicts are detected by conflict managers on the fly.



Transaction Life Cycle: commit

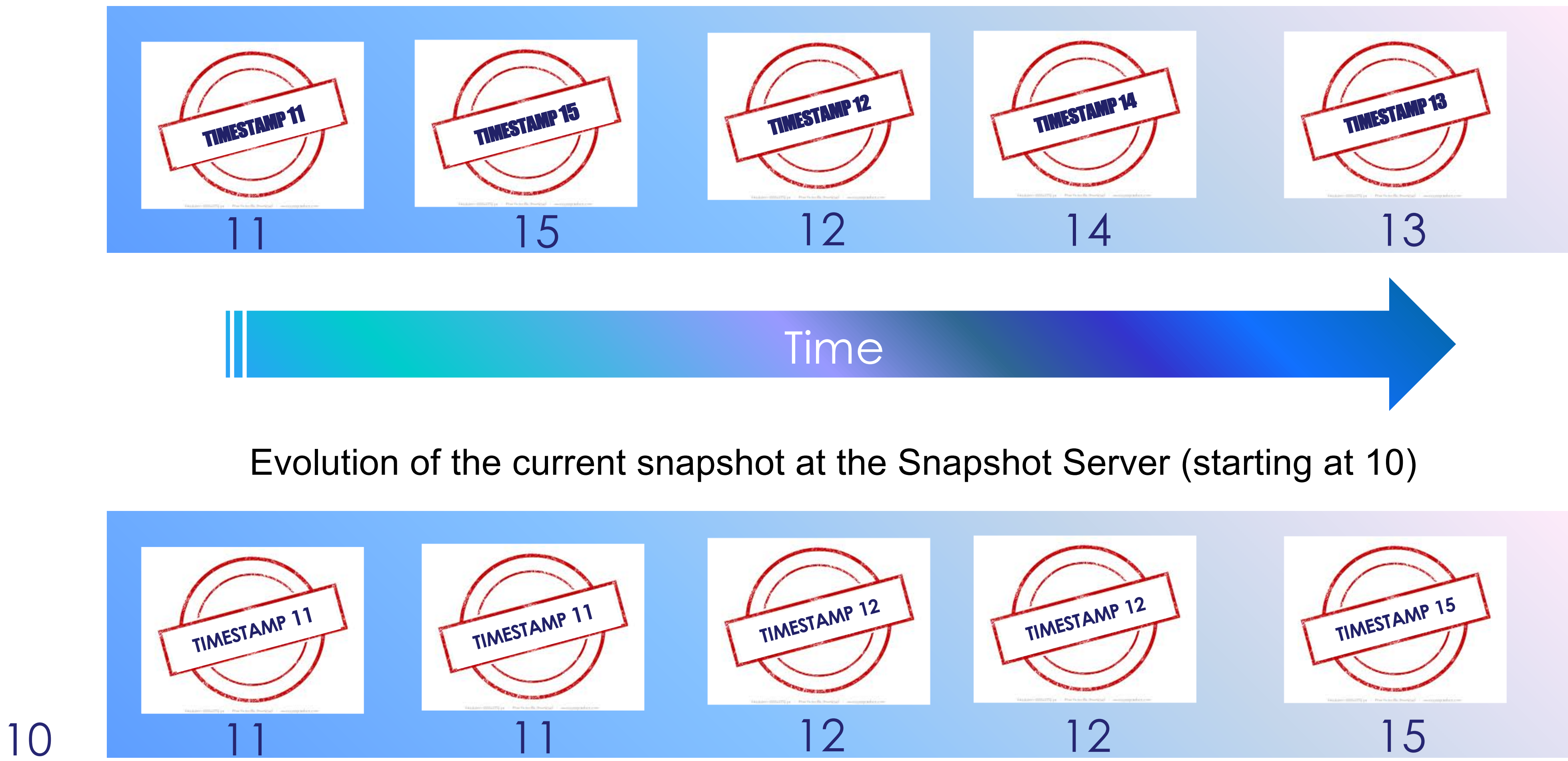
The local transaction manager orchestrates the commit.

Transaction Life Cycle: commit



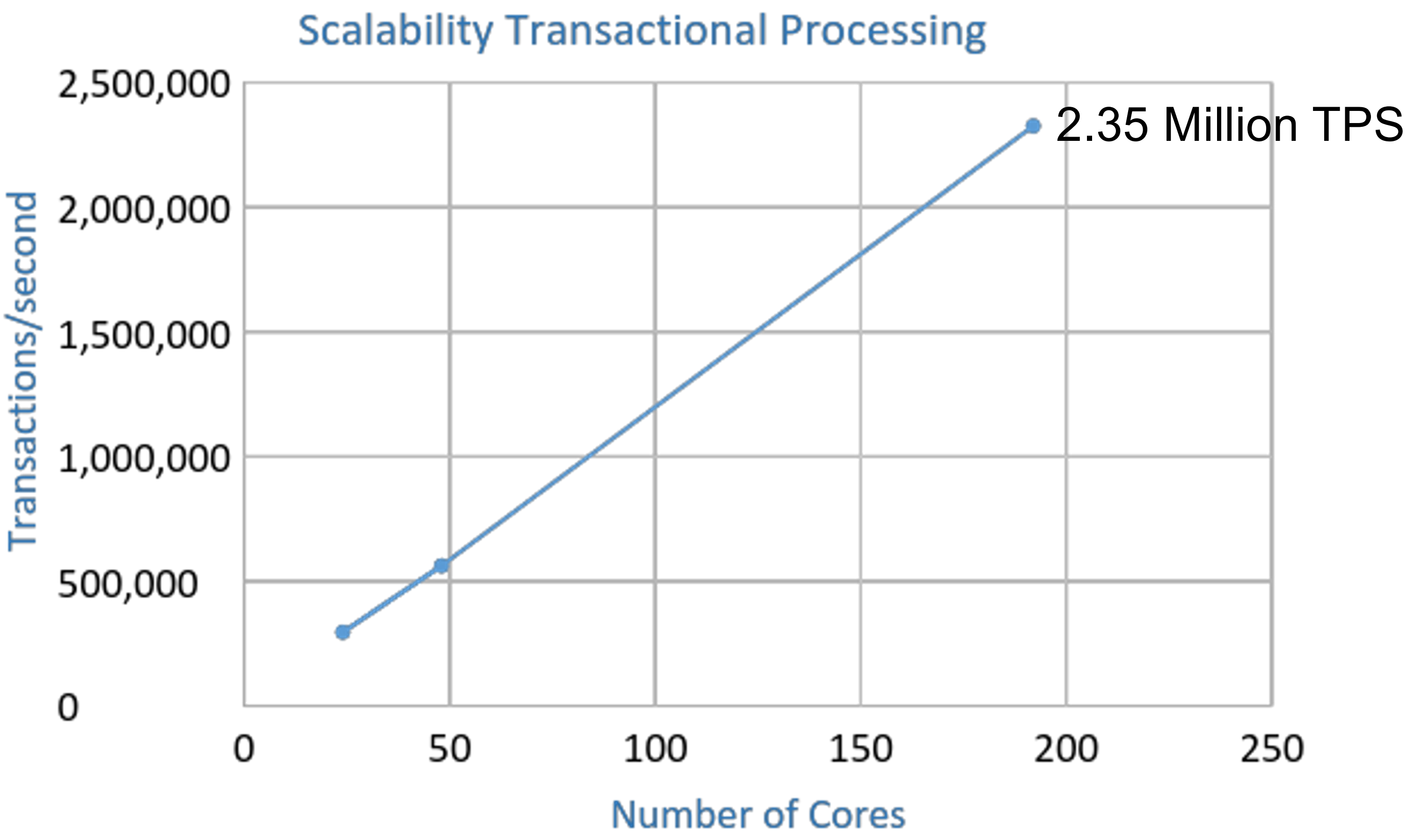
Transaction Life Cycle: commit

Sequence of commit timestamps received by the Snapshot Server



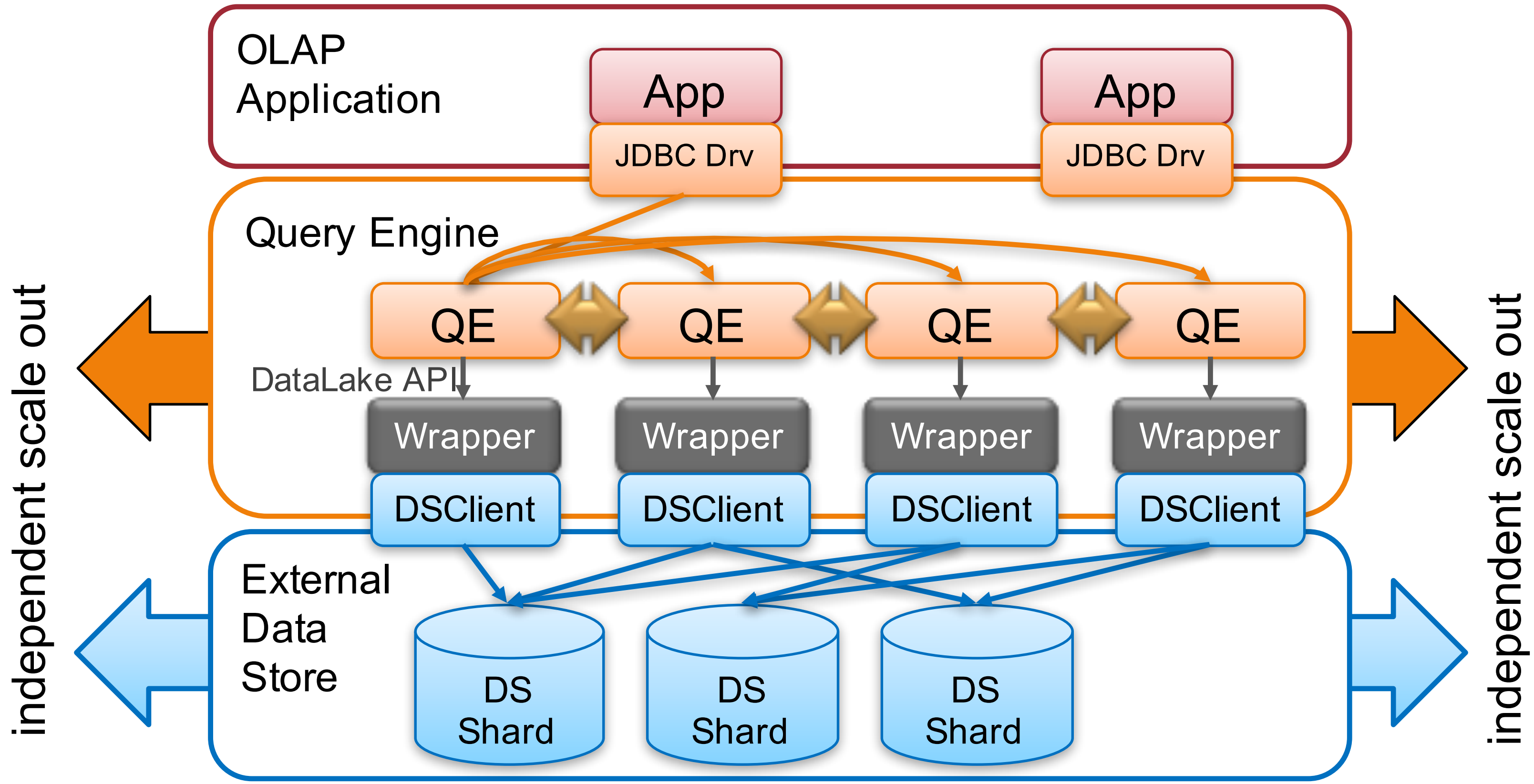
Evolution of the current snapshot at the Snapshot Server (starting at 10)

LeanXcale Transactional Scalability



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

LeanXcale Polystore Architecture



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

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.

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
```