NoSQL Databases 2

Marek Rychlý
rychly@fit.vut.cz

Brno University of Technology
Faculty of Information Technology
Department of Information Systems

Data Storage and Preparation (UPA)
30 September 2025

Outline

1. Quering and Aggregation in NoSQL
   - Quering in NoSQL
   - Aggregation in NoSQL
2. NewSQL Databases
   - MemSQL Database

Key-Value Storage in NoSQL

- NoSQL database are mostly distributed key-value stores.
  - Cassandra: PK and its partitioning and clustering components.
  - MongoDB: "_id" field in a collection of JSON documents.
  - Neo4J: ID of nodes and edges in a graph.
  - InfluxDB: Series key of a time-series data.
    (series key = measure name, tags and fields, timestamp)
- The key is utilized to find
  - correct data node storing the key-value pair in the cluster,
    (by a directory service or by a distributed hash table)
  - local disk block where the value is stored for the key.
    (by a AVL/B/B+ tree, by LSM tree, by a hash table, etc.)

Partitioning and Sharding for Scalability

- NoSQL implies performance, scalability, fault tolerance, etc.
  (with respect to the CAP theorem and the BASE approach)
- For these features, NoSQL databases partition data horizontally.
  - the vertical partitioning by database schema
    (by the common properties/type of data; e.g., to tables or collections)
  - the horizontal partitioning/sharding by grouped values
    (i.e., dividing and distribution of the data based on a shard key)
- Various (horizontal) partitioning strategies.
  (by range of values, by hash, by access frequency, by insert time, etc.)
- NoSQL databases usually partition data by hash of the key.
  (which is quite fast and data agnostic)

Pros and Cons of Data Sharding

+ handle more load using multiple nodes (horizontal scalability)
  (increased performance, both write and read operations)
+ nodes in the individual shards can replicate their data
  (all nodes in a shard are accessed by the same sharding key value)
– difficult to change a sharding key with existing data
  (both switching to different key and modifying a keyed value which would result into moving data to a different shard)
– difficult and expensive cross-shard operations
  (e.g., data shuffle required when aggregating or joining data bewteen shards)

Filtering by Key

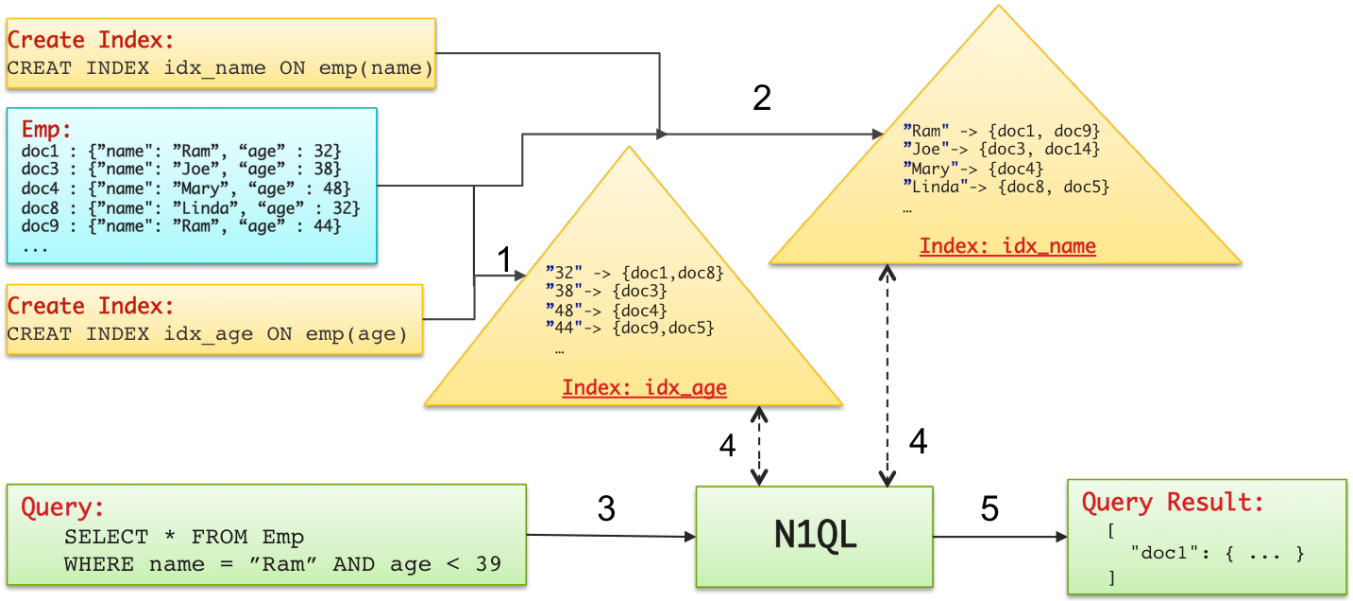
(key-scan or index-scan)

- Quering by the paritioning key always supported.
  (it is fundamental in key-value databases)
- Often only available query model, available in API.
  (a NoSQL database is to store/retrieve, not to query data; moving query to the application layer)
- It is a primary index, usually implemented by DHT.
  (hashing, so the range scan is not available)
- Possible to introduce secondary indices.
  (however, those are not used for sharding, just for querying)

Filtering by Value

- Filtering by value supported in specialized NoSQL dbs.
  (with a structure value, e.g., in wide-column or document databases)

- The value-based querying possible
  - at the global level (iter-node)
    (by the index-scan; secondary indices required; eventual consistency)
  - at the local level (intra-node)
    (by local indices, e.g., B+ tree; full consistency with ACID)



(adopted from "Performance Ingredients for NoSQL: Intersect Scans in N1QL")

Query Optimization

- Not all query optimization steps from RDBMS are usable.
  (~~query rewrite~~, index selection, ~~join ordering~~ and ~~join type selection~~)
- NoSQL database usually do not support join operation.
  (usually not possible due to the data sharding and temporal inconsistency)
- The most of magic is done by the index selection.
  (by automatically/manually created secondary indices and their combinations)
- For complex queries, it may be necessary to process data in multiple iterations.
  (each iteration produce a key-value dataset)
- Nested data structures and data flattening may help.
  (instead of the join operation; prevent loading of external data)

Grouping Inside and Accross Partitions

- Grouping of data is restricted by partition boundaries.
  - It is easy to group/aggregate data inside a shard.
    (by local data processing on several shards in parallel)
  - Grouping accross the shards requires secondary indices.
    (i.e., previously created indicies, or ad-hoc temporary indices)
- A set of aggregation functions or patterns usually provided.
  (e.g., the aggregation operations and pipelines in MongoDB)
- Map-Reduce for complex or custom defined aggregation.
  (to perform an aggregation by aplying map and reduce functions)

Map and Reduce functions

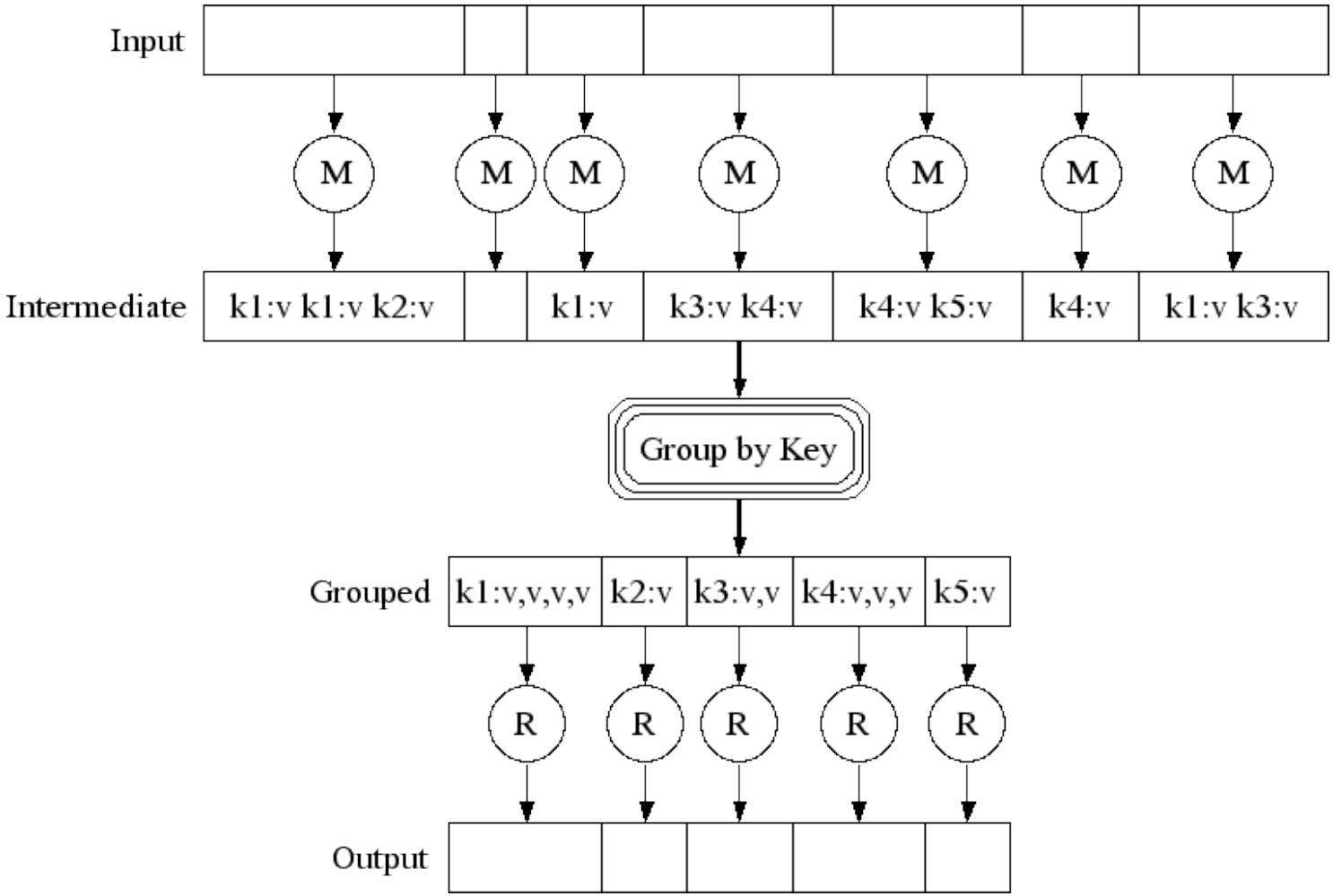
MapReduce applications are comprised of Map and Reduce functions defined on data represented by couples “key:value”.

$Map(k_1, v_1) \rightarrow list(k_2, v_2)$ Map is applied to each input data k_1:v_1 and creates a list of outputs k_2:v_2 for each input.

$Reduce(k_2, list(v_2)) \rightarrow list(k_3, v_3)$ Outputs of all Map applications are grouped according to a key and then Reduce is applied to each list for the given key and creates a list of output values from them.

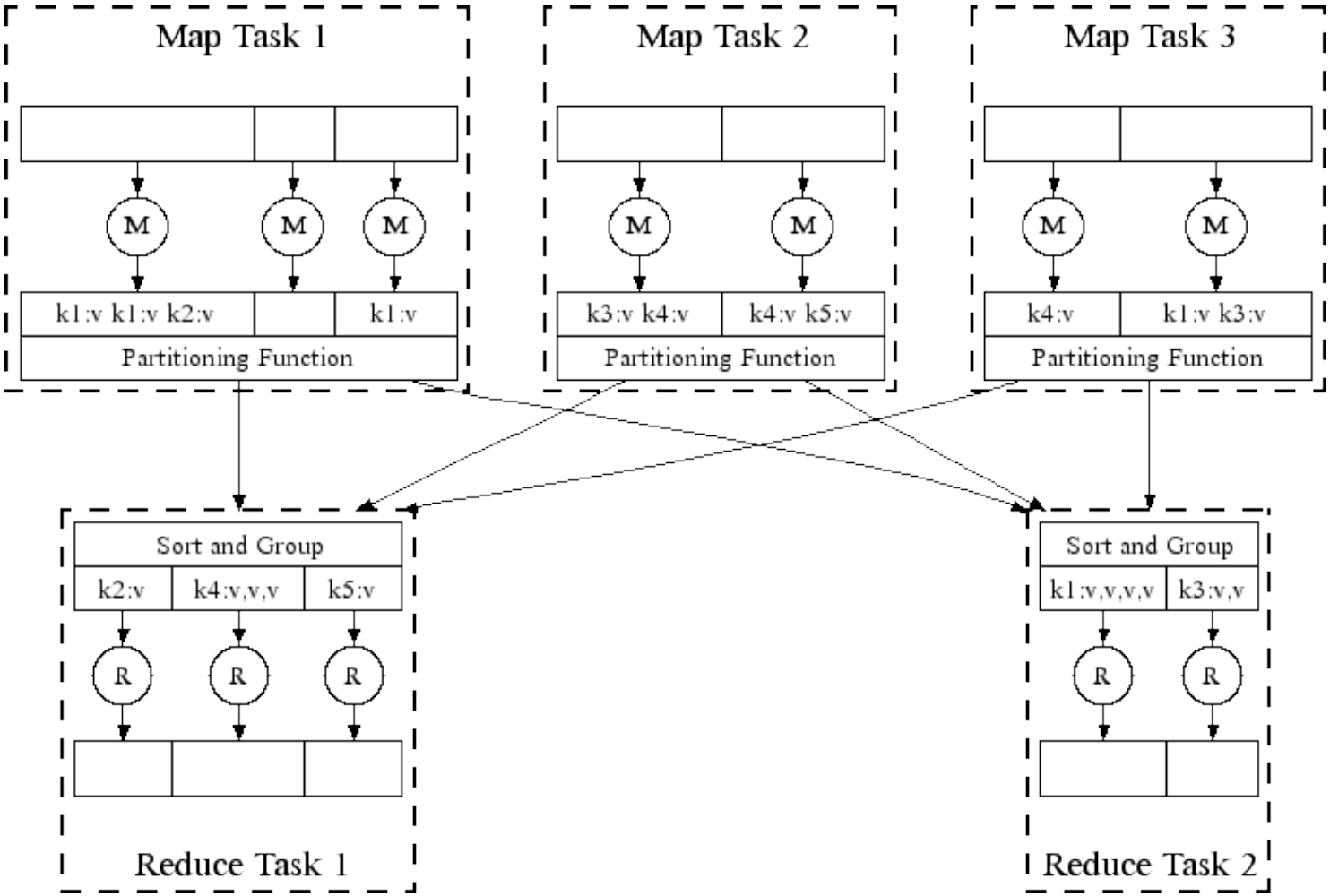
- k_1 key value of the input data item, for partitioning between nodes
- v_1 data of the input data item, i.e., item to be processed
- k_2, k_3 key value of the output data, e.g., name of the data processing result
- v_2 an intermediate result from data processing of individual inputs
- v_3 the result by aggregation of intermediate results for each key k_2

MapReduce process



(adopted from “MapReduce: Simplified Data Processing on Large Clusters”)

Parallel MapReduce process

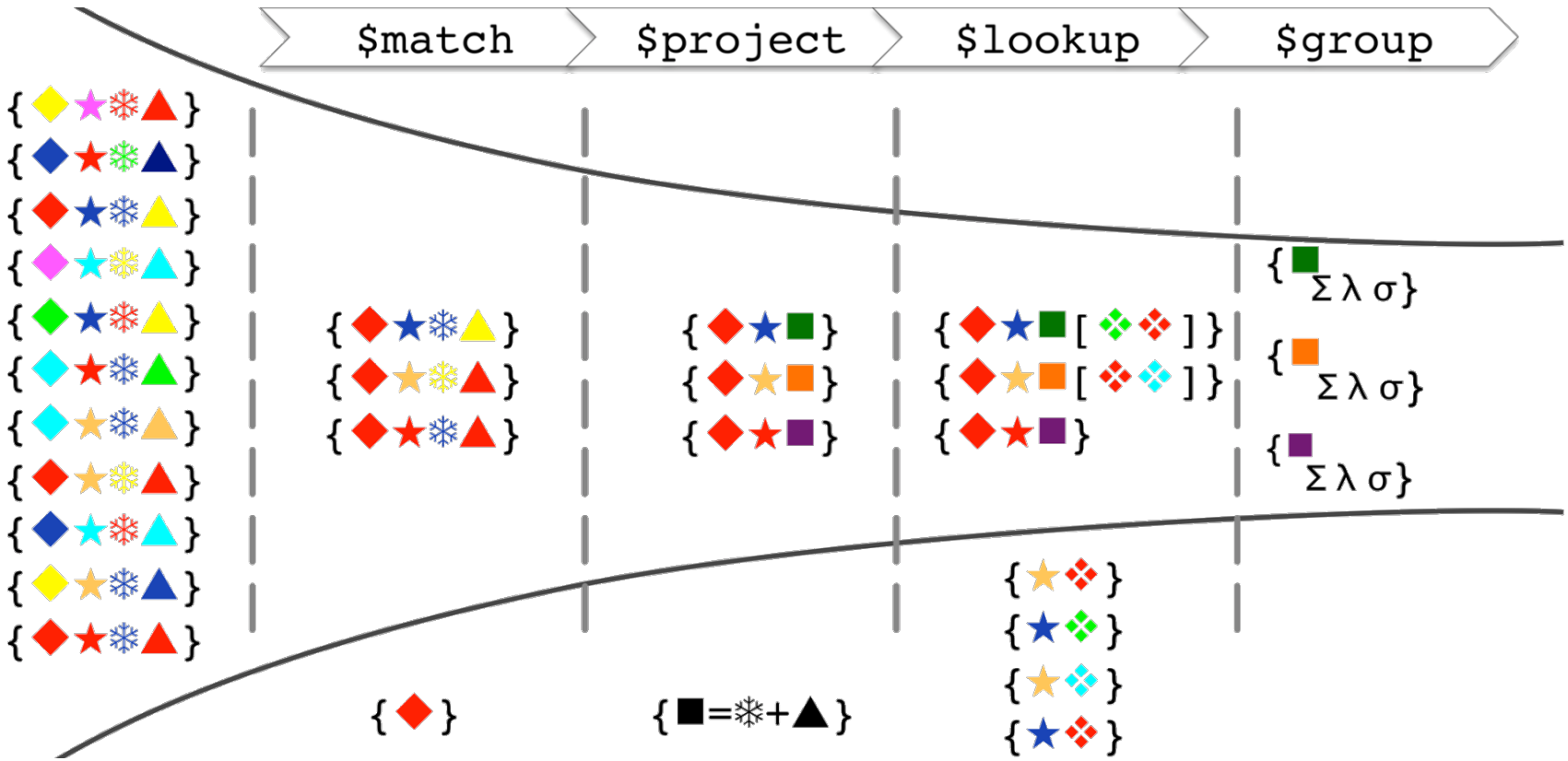


(adopted from “MapReduce: Simplified Data Processing on Large Clusters”)

Aggregation Pipline in MongoDB

- Data processing pipelines for (mainly) aggregation.
  (documents enter a multi-stage pipeline that transforms them into results)
- Each stage transforms the documents as they pass through by
  - $match to filter documents at the beginning of a pipeline,
  - $project to process data inside the documents,
  - $lookup to enhance data by external document,
  - $group to group documents by an index,
  - $sort to sort documents by an index.
- All results (both intermediate and final) are document collections.
  (in each collection, the documents are identified by they “_id” fields)

Aggregation Pipline in MongoDB: An Example



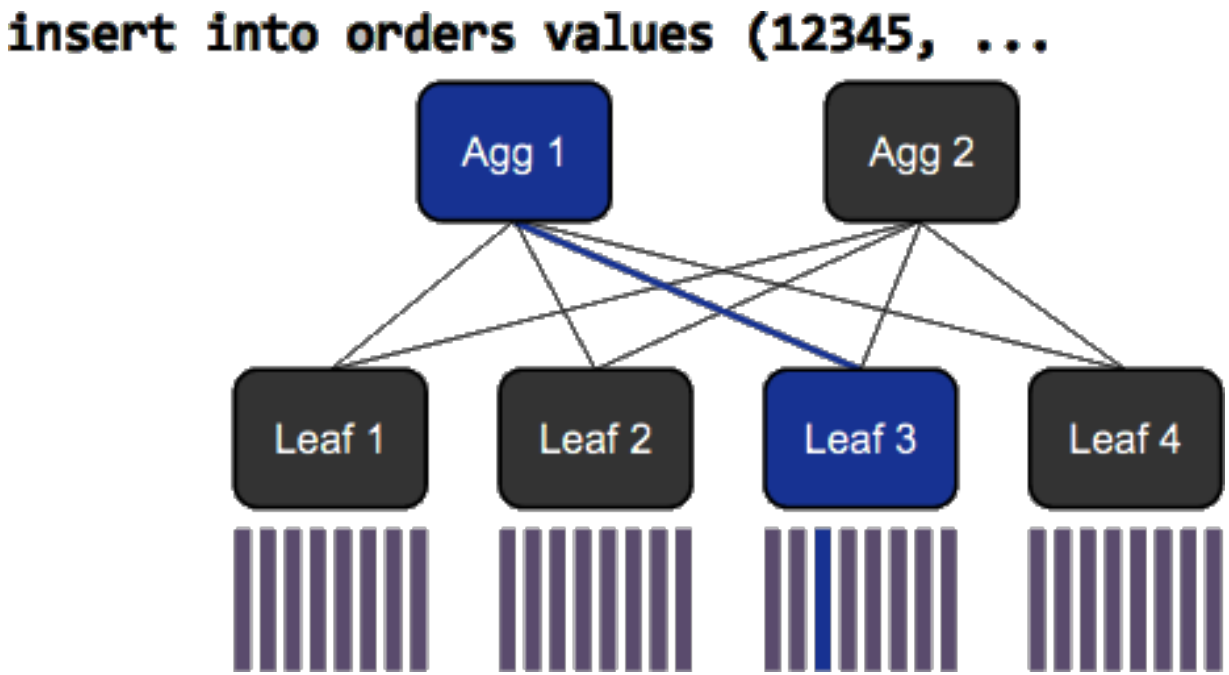
(adopted from “Joins and Other Aggregation Enhancements Coming in MongoDB 3.2”)

NewSQL Database

- Provide NoSQL features on ACID-complaint SQL databases.
  (high-throughput and high-availability systems with transactional consistency)
- Offer strong consistency by sacrificing some availability
  (favoring consistency over availability, i.e., CP from CAP)
  - by using consensus protocols such as Paxos or Raft,
  - by forcing durability or synchronous replications as in MemSQL.
- Support SQL interfaces and backward compatibility to RDBMS.
  (however, not all SQL queries may be optimal)
- Some of NewSQL databases:
  ClustrixDB, NuoDB, CockroachDB, MemSQL, VoltDB, Percona TokuDB, ActorDB

MemSQL Aggregators: Insert

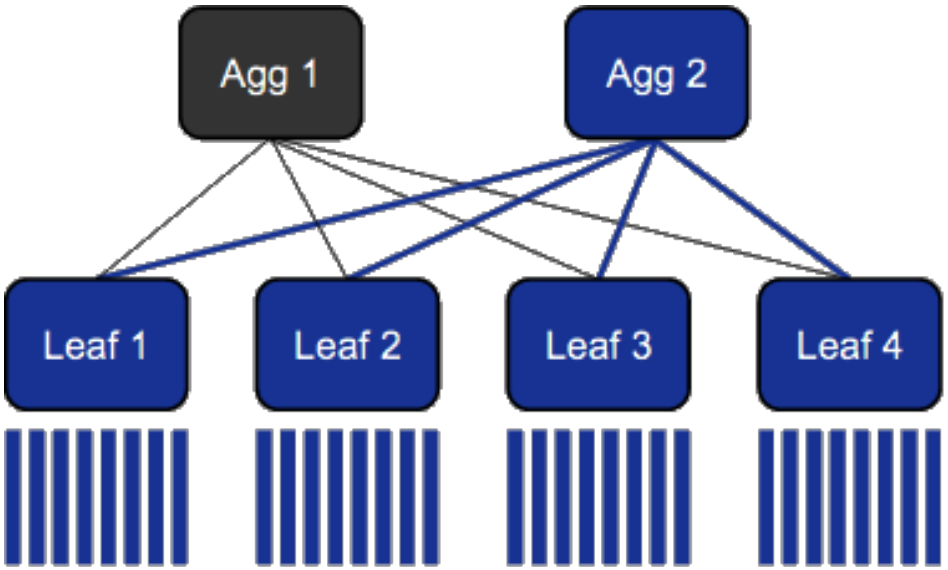
(an aggregator sends row to the leave based on a shard-key hash value)



(adopted from “Overview of MemSQL”)

MemSQL Aggregators: Select

(an aggregator sends request to the leaves and combines results)



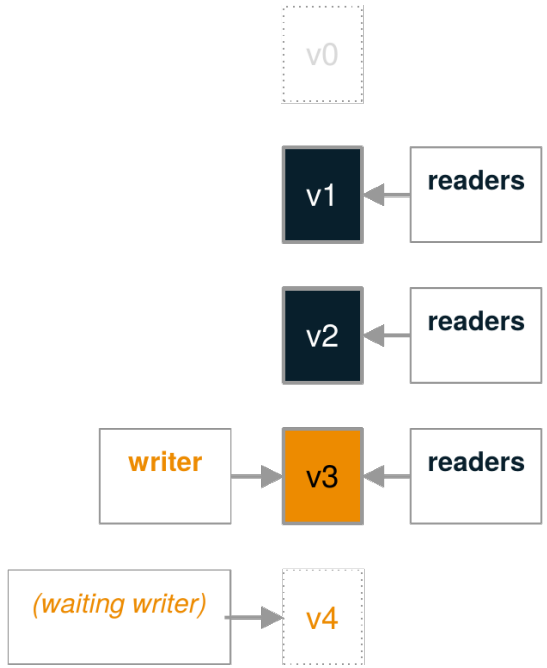

```
select count(1) from orders;
```

```
leaf1> using memsql_demo_0
select count(1) from orders;

leaf2> using memsql_demo_1
select count(1) from orders;

leaf3> using memsql_demo_2
select count(1) from orders;
...
```

(adopted from “Overview of MemSQL”)

MemSQL Multiversion Concurrency Control

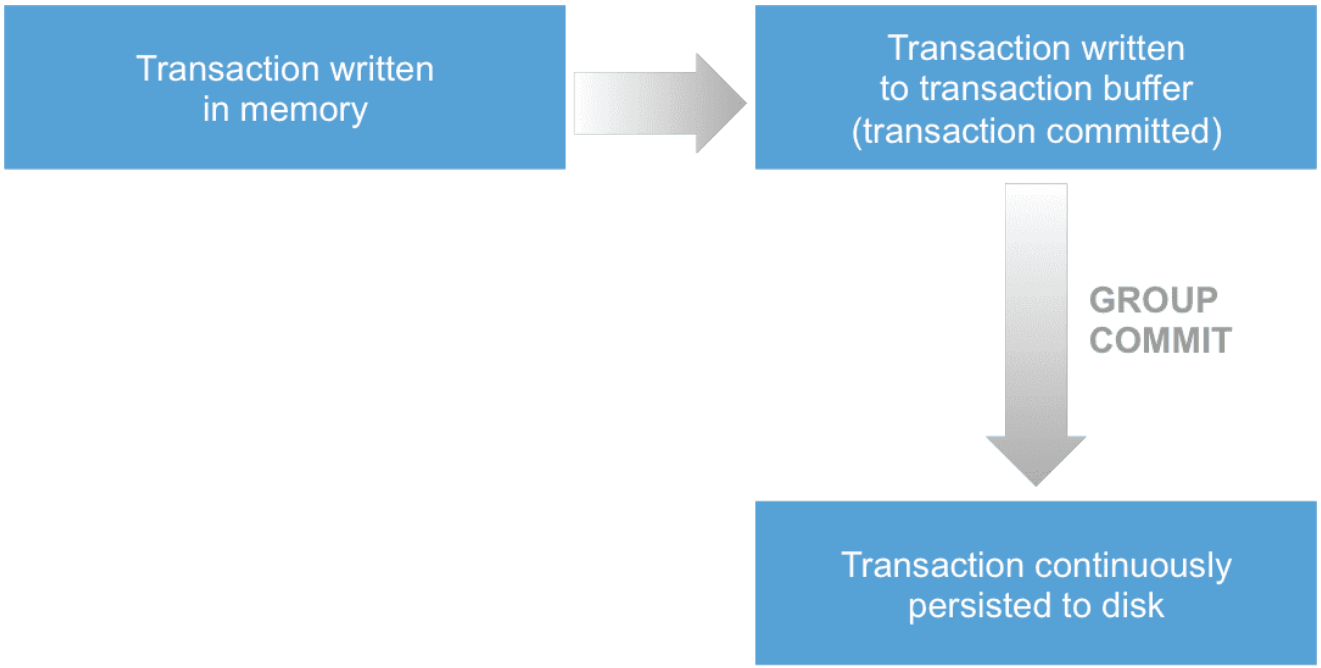
- Every write creates a new version of row.
- Old versions get garbage-collected.
- Reads are never blocked.
- Row-level locking for writes.
- Allows online ALTER TABLE!



(adopted from “In-Memory Database Performance on AWS M4 Instances”)

MemSQL Durability

(a transaction buffer to increase performance at the expense of reliability)



(adopted from “Overview of MemSQL”)

Thank you for your attention!

Marek Rychlý
<rychly@fit.vut.cz>