



newSQL

Column Store, Document DB



Let's go back to Early-2000s...

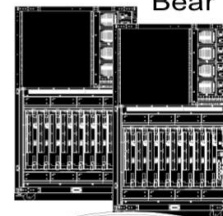
- All the big players were heavyweight and expensive.
 - Oracle, DB2, Sybase, SQL Server, etc.
- Open-source databases were missing important features.
 - Postgres, mSQL, and MySQL.

V2.5 *April 2001 – December 2002*

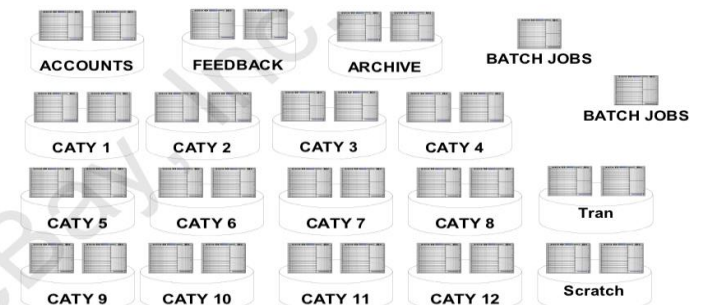
- Horizontal scalability through database splits
- Items split by category
- SPOF elimination



Bull Bear



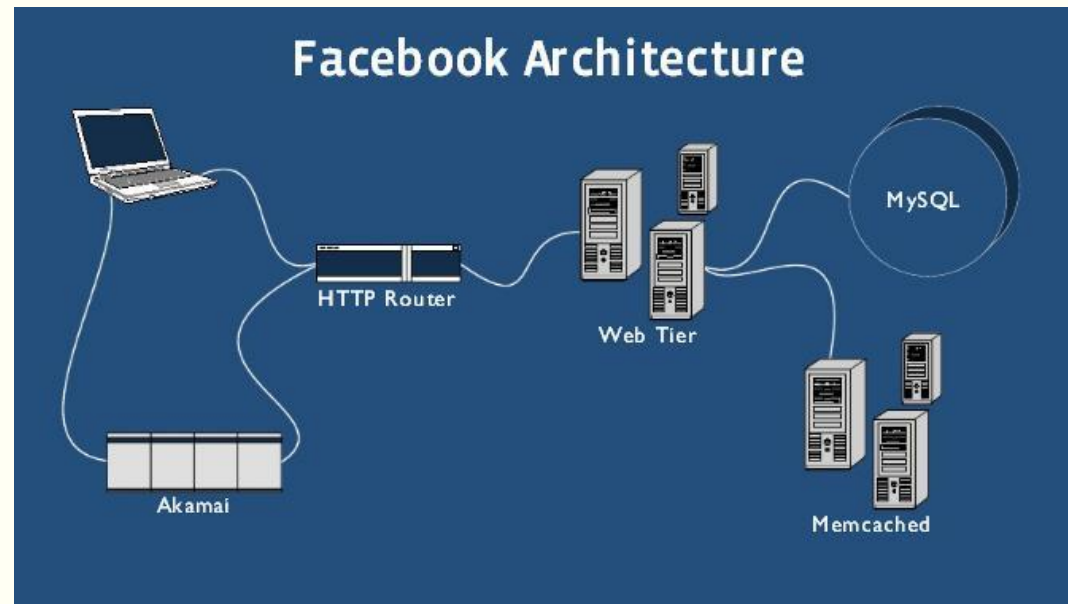
SUN
A3500



December, 2002

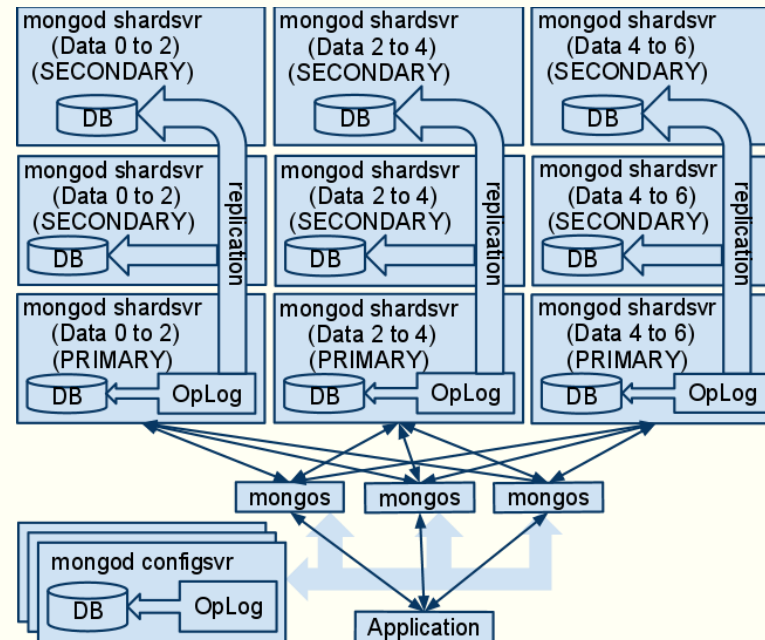
Mid 2000s

- MySQL + InnoDB is widely adopted by new web companies:
 - Supported transactions, replication, recovery.
 - Still must use custom middleware to scale out across multiple machines.
 - Memcache for caching queries.



Late-2000s

- NoSQL systems are able to scale horizontally:
 - Schemaless.
 - Using custom APIs instead of SQL.
 - Not ACID (i.e., eventual consistency)
 - Many are based on Google's BigTable or Amazon's Dynamo systems.

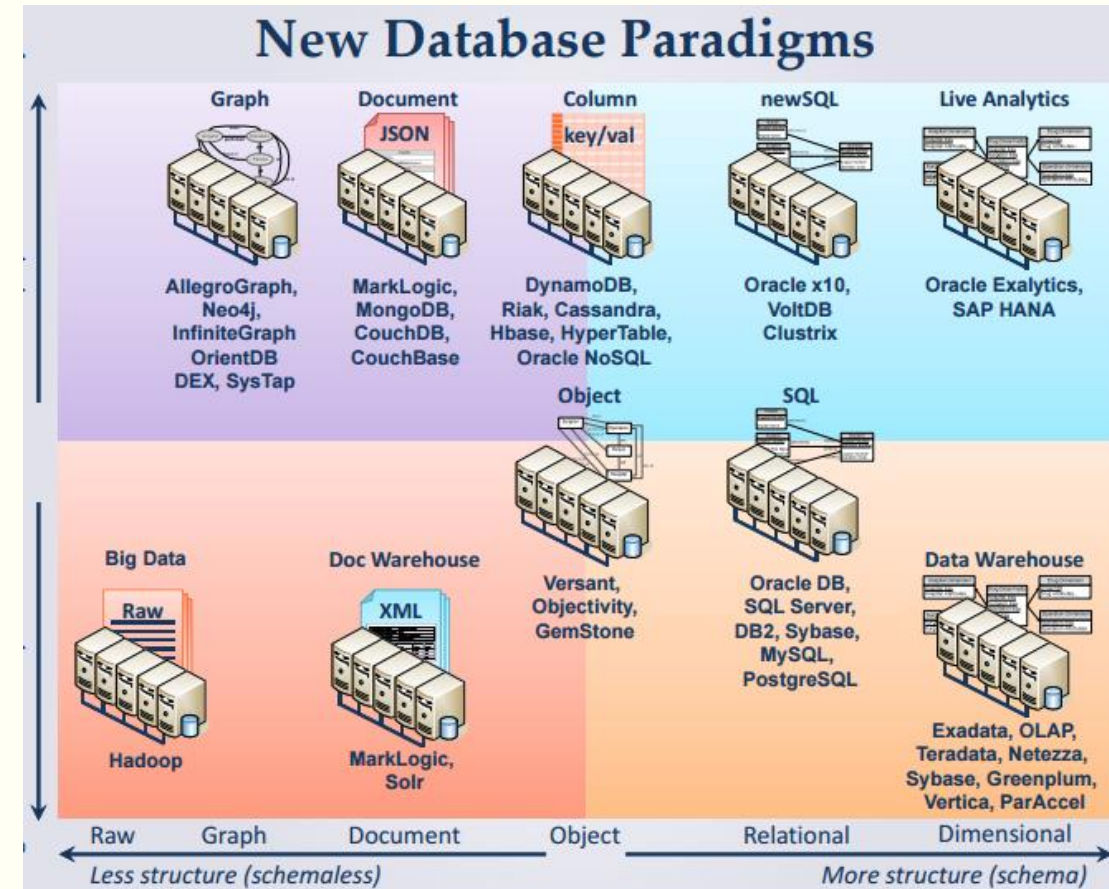


Early-2010s

- New DBMSs that can scale across multiple machines natively and provide ACID guarantees.
 - MySQL Middleware
 - Brand New Architectures
- “New SQL”



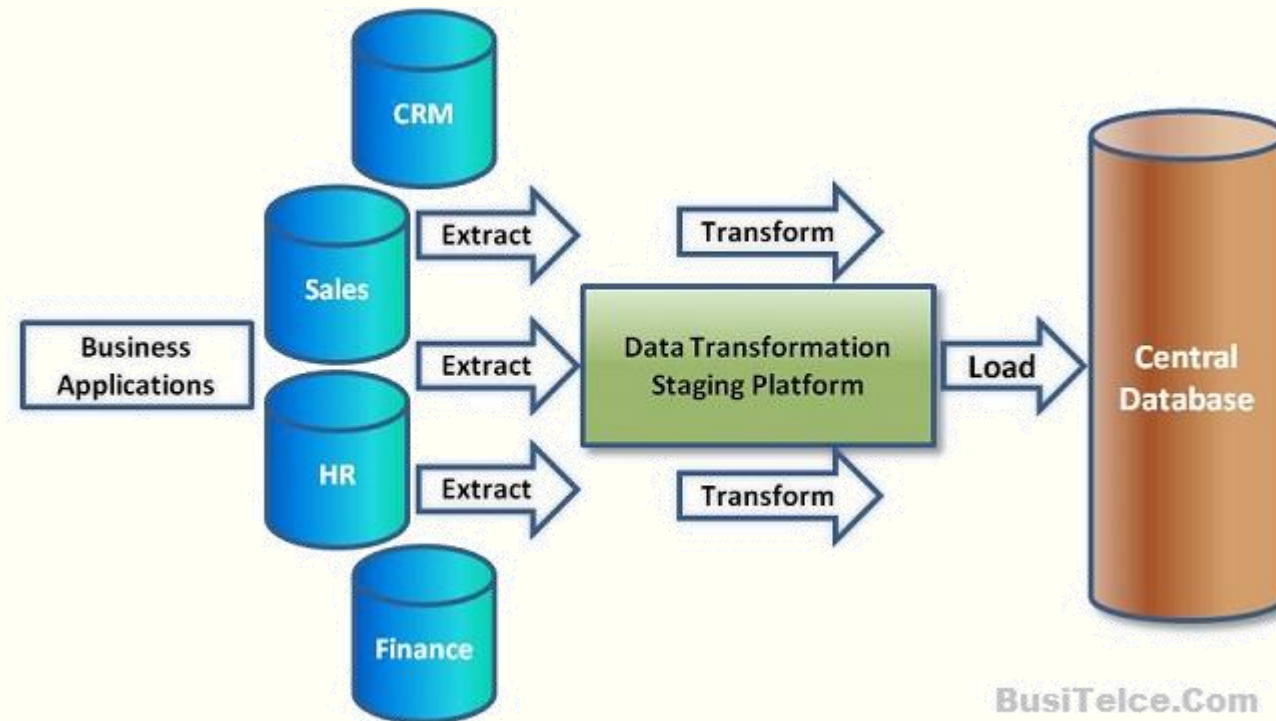
newSQL



old OLTP and old SQL

- An information system can be transactional (OLTP) or/and analytical (OLAP)
- OLTP (On-line Transaction Processing):
 - a large number of short on-line transactions (INSERT, UPDATE, DELETE)
 - very fast query processing
 - data integrity in multi-access environments
 - effectiveness measured by number of transactions per second.
 - schema is used to store transactional databases (usually 3NF).
- Old OLTP:
 - Collection of OLTP systems
 - Connected to ETL (Extract-Transform-and-Load)
 - Connected to one or more data warehouses
- Old SQL: techs, systems and vendors supporting old OLTP

ETL



New requirements (new OLTP)

- Web changes everything
- Large scale systems, with huge and growing data sets
 - 9M messages per hour in Facebook
 - 50M messages per day in Twitter
- Information is frequently generated by devices (cellphones, PDAs, sensors...)
 - “Online”
- High concurrency requirements, high-throughput ACID write
 - “Transaction”
- High Availability + Durability: core database requirements
- Need for high throughput
- Need for real-time analytics

Challenge

- Give up SQL
- Give up ACID
 - Data accuracy
 - Funds transfer
 - Integrity constraints
 - Multi-record state
- noSQL fits
 - Non-transactional systems
 - Single record transactions that are commutative
- noSQL is not a good fit for
 - New OLTP: gaming, purchasing, order management, real-time analytical, etc

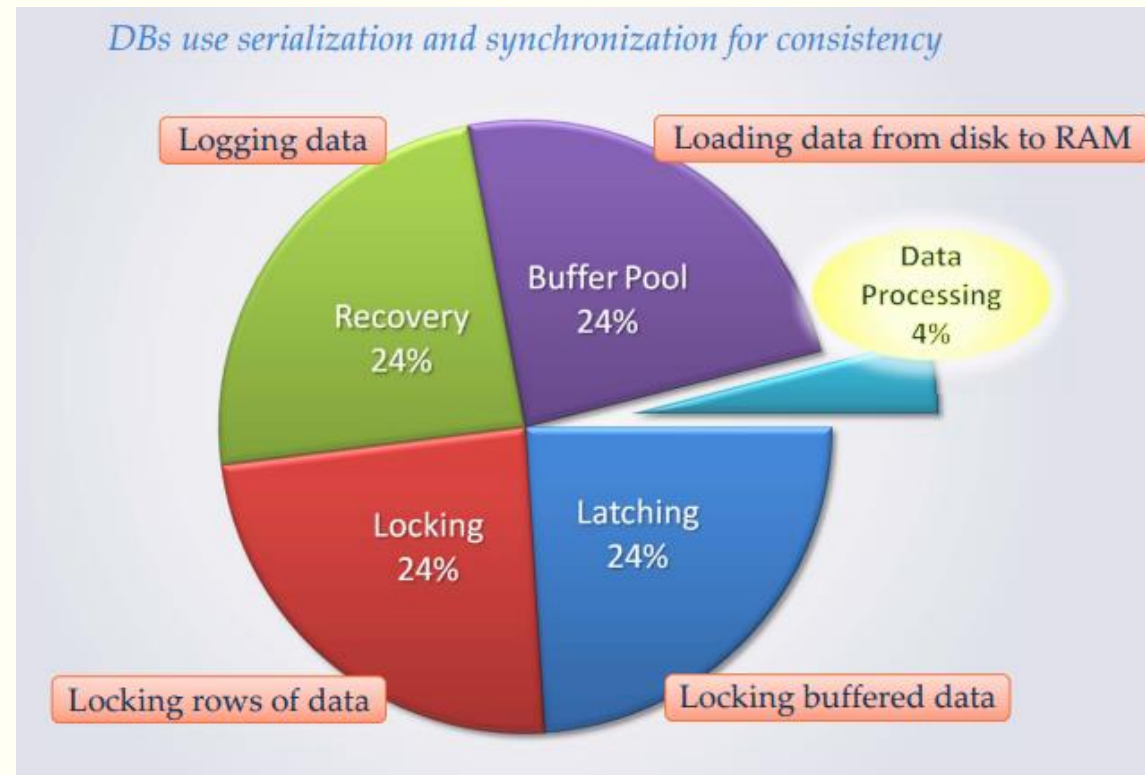
NewSQL: informal definition

- SQL is good
 - ACID is good
 - Figure out a way to make oldSQL perform
 - Make it scale like noSQL
 - Make it available
-
- “A DBMS that delivers the scalability and flexibility promised by NoSQL while retaining the support for SQL queries and/or ACID, or to improve performance for appropriate workloads.” -- *451 Group*

NewSQL: definition

- SQL as the primary mechanism for application interaction
- ACID support for transactions
- A non-locking concurrency control mechanism so real-time reads will not conflict with writes, and thereby cause them to stall.
- An architecture providing much higher per-node performance than available from the traditional "elephants"
- A scale-out, shared-nothing architecture, capable of running on a large number of nodes without bottlenecking
 - Michael Stonebraker

Traditional DBMS overheads



“Removing those overheads and running the database in main memory would yield orders of magnitude improvements in database performance”

NewSQL Design Principles

- SQL + ACID + performance and scalability through modern innovative software architecture
- Principle 1: minimizing or stay away from locking
- Principle 2: rely on main memory
- Principle 3: try to avoid latching
- Principle 4: cheaper solutions for HA

NewSQL Design Principles

- new solution other than low-level record level locking mechanism
 - Transaction processed in timestamp order with no locking (voltDB)
 - multisession concurrency control (nuoDB)
 - Optimistic concurrency control (Google)
 - *[Principle: minimizing or stay away from locking](#)*

NewSQL Design Principles

- new solution for buffer pool overhead
 - Main memory DBMS
 - Moderate case is tilted towards main memory
 - [*Principle 2: rely on main memory*](#)
- new solution to latching for shared data structures
 - new way to manage B-trees
 - Single-threading
 - [*Principle 3: try to avoid latching*](#)
- new solution for write-ahead logging
 - Built-in replication
 - [*Principle 4: cheaper solutions for HA*](#)

NewSQL: Categories

- New approaches: VoltDB, Clustrix, NuoDB
- New Storage engines: TokuDB, ScaleDB
- Transparent Clustering: ScaleBase, dbShards



VoltDB

- VoltDB is an in-memory, horizontally scalable, ACID compliant, fast RDBMS
- Backed and architected by Michael Stonebraker
- An open source project
- Java and C/C++
- Available in community and commercial editions

Technical Overview

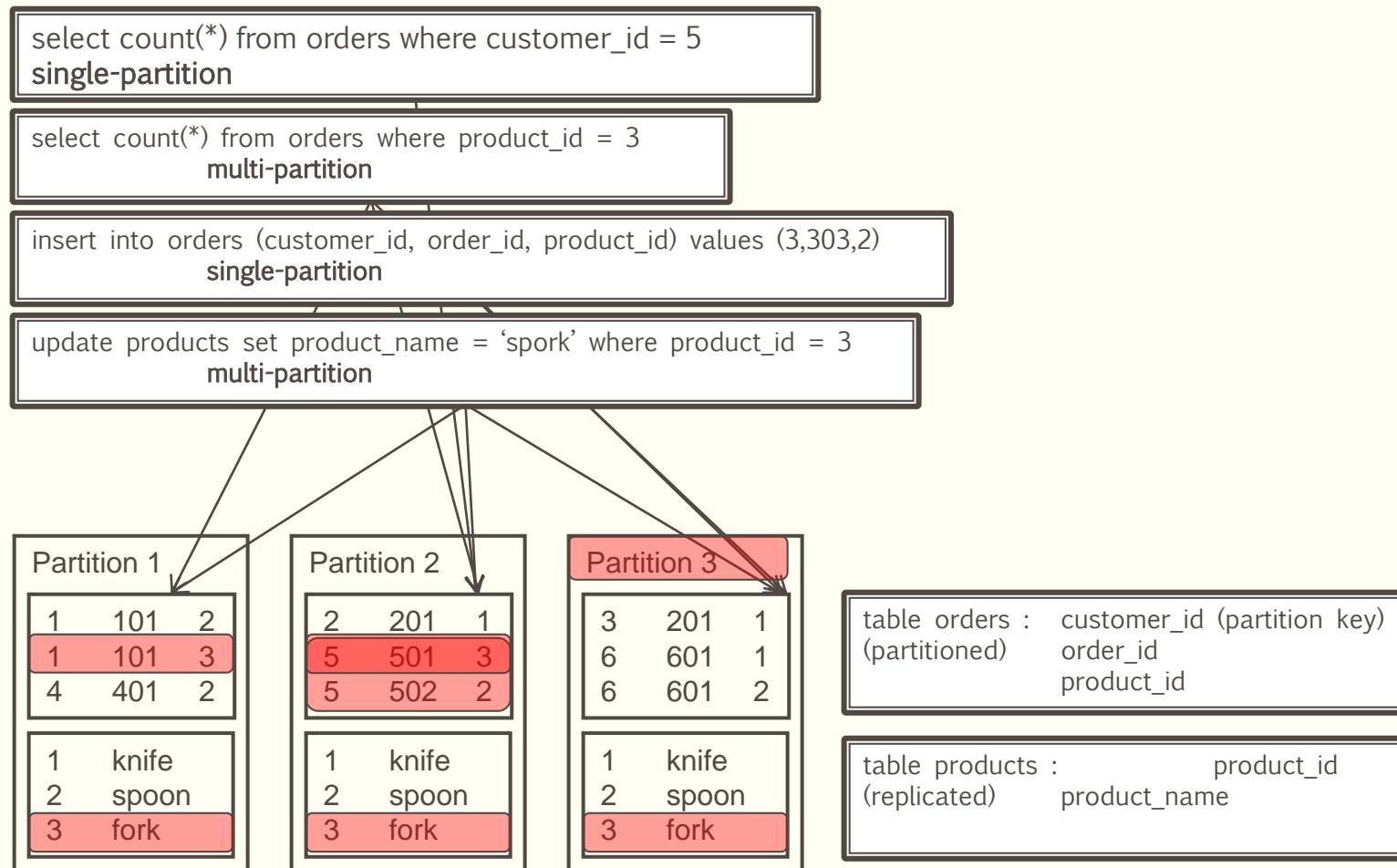
- VoltDB tries to avoid the overhead of traditional databases
- K-safety for fault tolerance
- In memory operation for maximum throughput
 - reduce buffer management
- Partitions operate autonomously and single-threaded
 - no latching or locking
- Built to horizontally scale

Technical Overview – Partitions (1/3)

- One **partition** per physical CPU core
 - Each physical server has multiple VoltDB partitions
- **Data** - Two types of tables
 - Partitioned
 - Single column serves as partitioning key
 - Rows are spread across all VoltDB partitions by partition column
 - Transactional data (high frequency of modification)
 - Replicated
 - All rows exist within all VoltDB partitions
 - Relatively static data (low frequency of modification)
- **Code** - Two types of **work** – both ACID
 - Single-Partition
 - All insert/update/delete operations within single partition
 - Majority of transactional workload
 - Multi-Partition
 - CRUD against partitioned tables across multiple partitions
 - Insert/update/delete on replicated tables

Technical Overview – Partitions (2/3)

■ Single-partition vs. Multi-partition



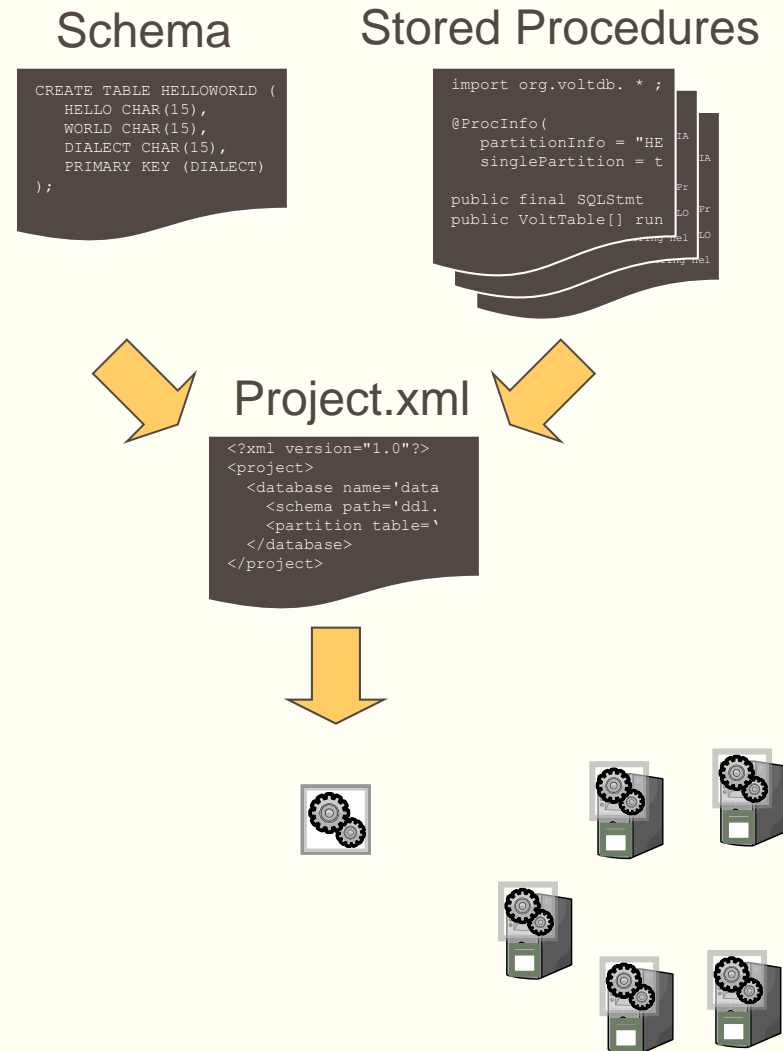
Technical Overview – Partitions (3/3)

- Looking inside a VoltDB partition...
 - Each partition contains data and an execution engine.
 - The execution engine contains a queue for transaction requests.
 - Requests are executed sequentially (single threaded).

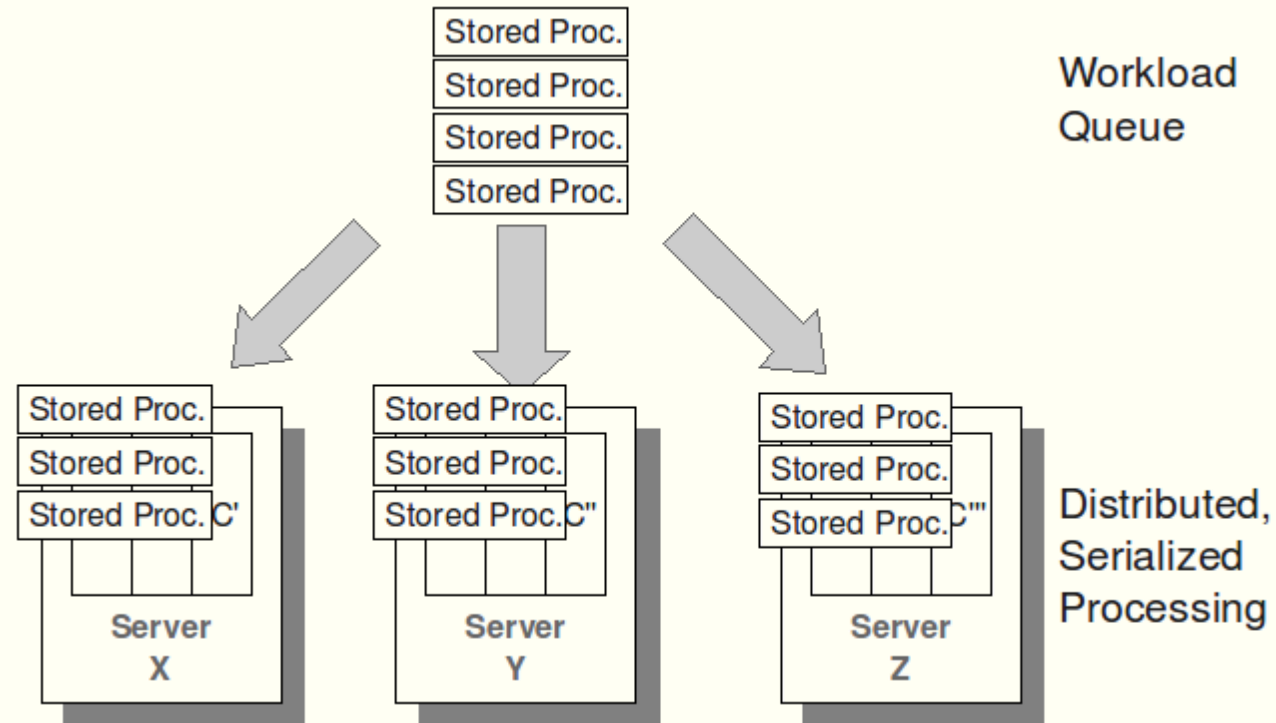


Technical Overview – Compiling

- The database is constructed from
 - The schema (DDL)
 - The work load (Java stored procedures)
 - The Project (users, groups, partitioning)
- VoltCompiler creates application catalog
 - Copy to servers along with 1 .jar and 1 .so
 - Start servers



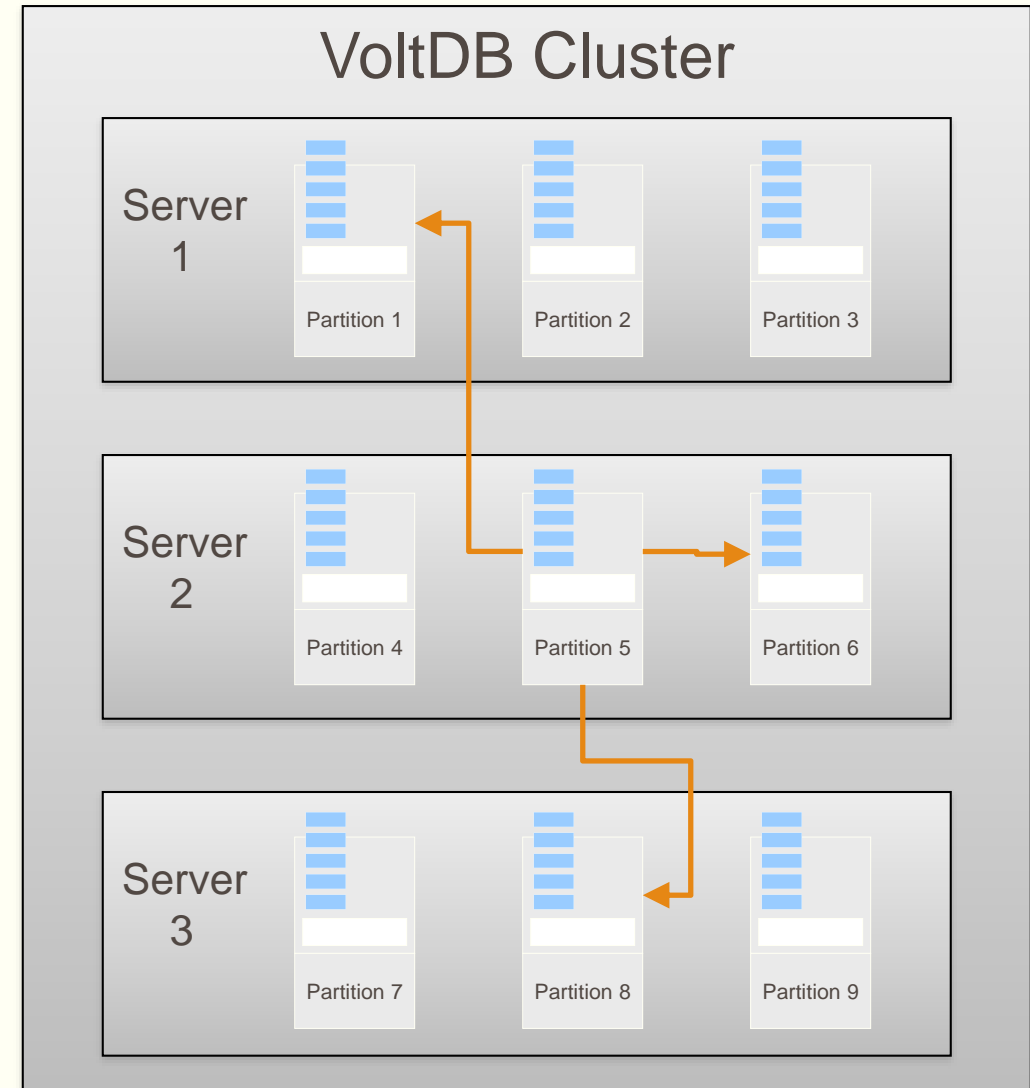
Transaction Model



Procedures **routed** to, **ordered** and **run** at partitions

Transaction Execution

- Single partition transactions
 - All data is in one partition
 - Each partition operates autonomously
- Multi-partition transactions
 - One partition distributes and coordinates work plans

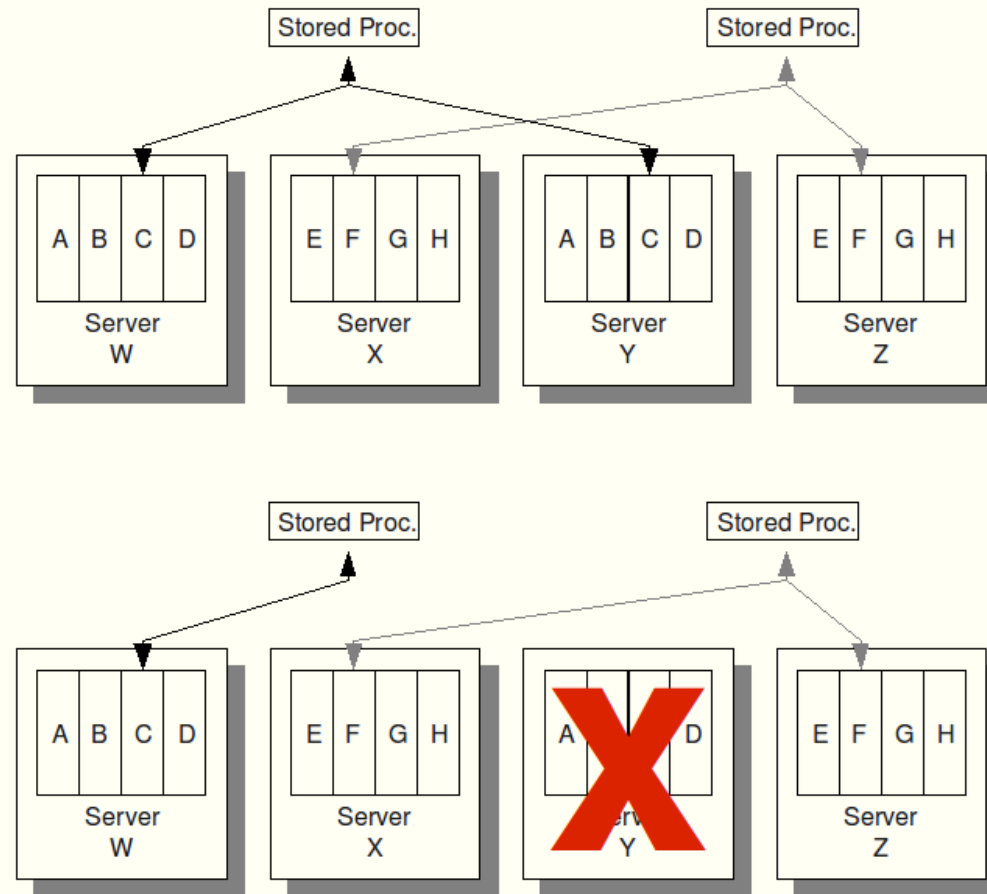


Data Availability and Durability

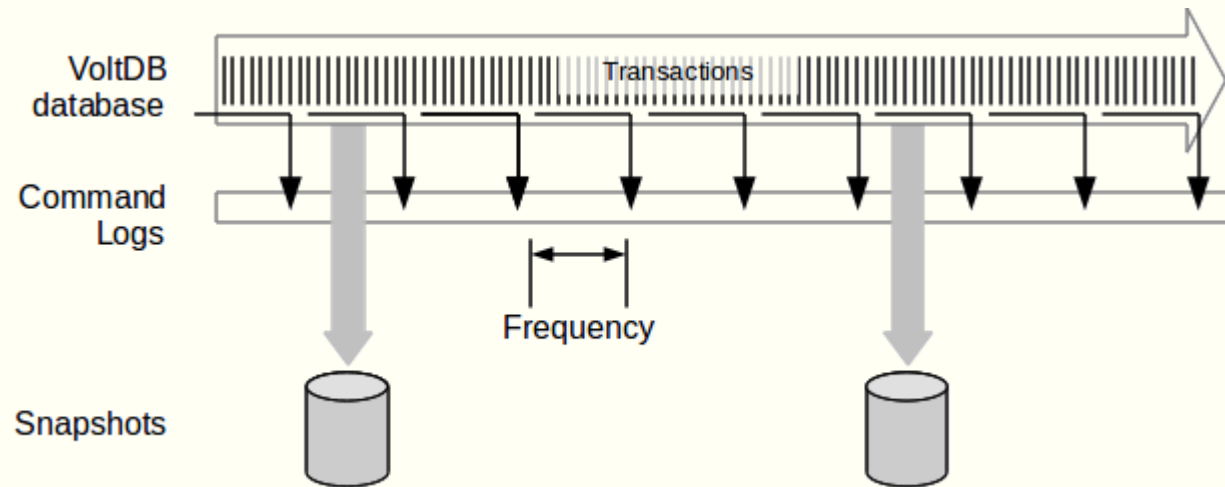
- High Availability
 - Data stored on server replicas (user configurable)
 - Failover data redundancy
 - No single point of failure
- Database Snapshots
 - Simplifies backup/restore
 - Scheduled, continuous, on demand
 - Cluster-wide consistent copy of all data
- Command Logging
 - Between Snapshots, every transaction is durable to disk

K-safety

- Duplicate database partitions for fault tolerance. K: # of replicas



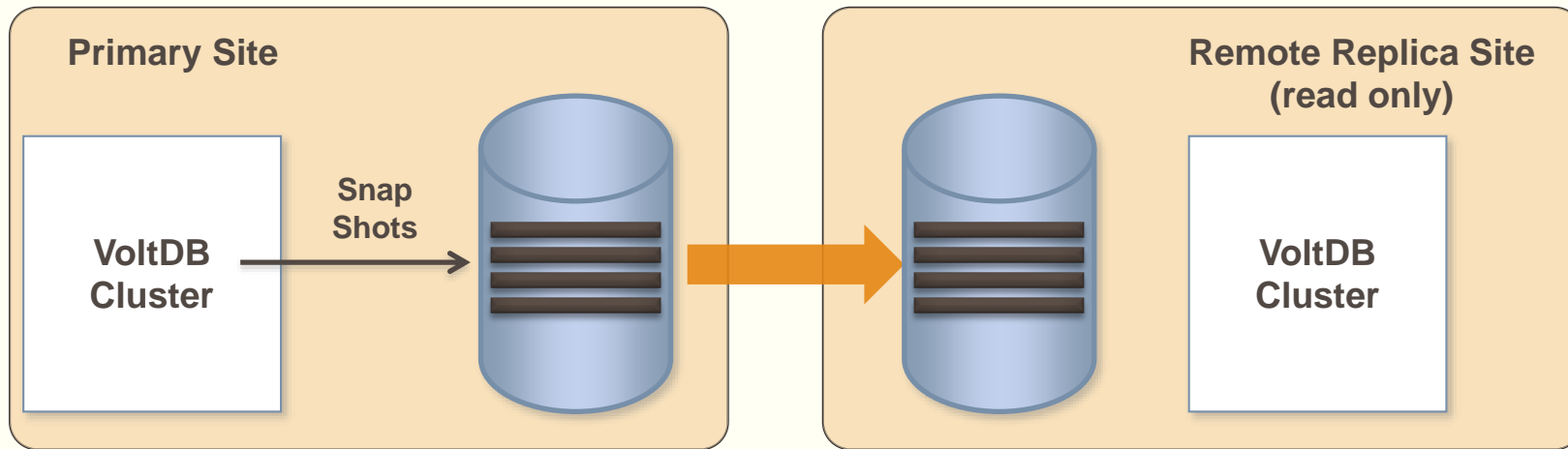
Command Logging



- Synchronous logging provides highest durability at reduced performance
- Asynchronous logging best performance at reduced durability

Disaster Recovery

- Disk snapshots replicated via storage system
- Stream command logs from Primary to Replica
- Run from Replica on DR event, reverse on recovery



Lack of Concurrency

- Single-threaded execution within partitions (single-partition) or across partitions (multi-partition)
- No concern about locking/dead-locks
 - great for “inventory” type applications
 - checking inventory levels
 - creating line items for customers
- Because of this, transactions execute in microseconds.
- However, single-threaded comes at a price
 - Other transactions wait for running transaction to complete
 - Useful for OLTP, not OLAP

nuoDB

- nuoDB is an elastically scalable, ACID compliant, newSQL Database
- Backed and architected by Jim Starkley
- Runs on JVM
- Proprietary source project

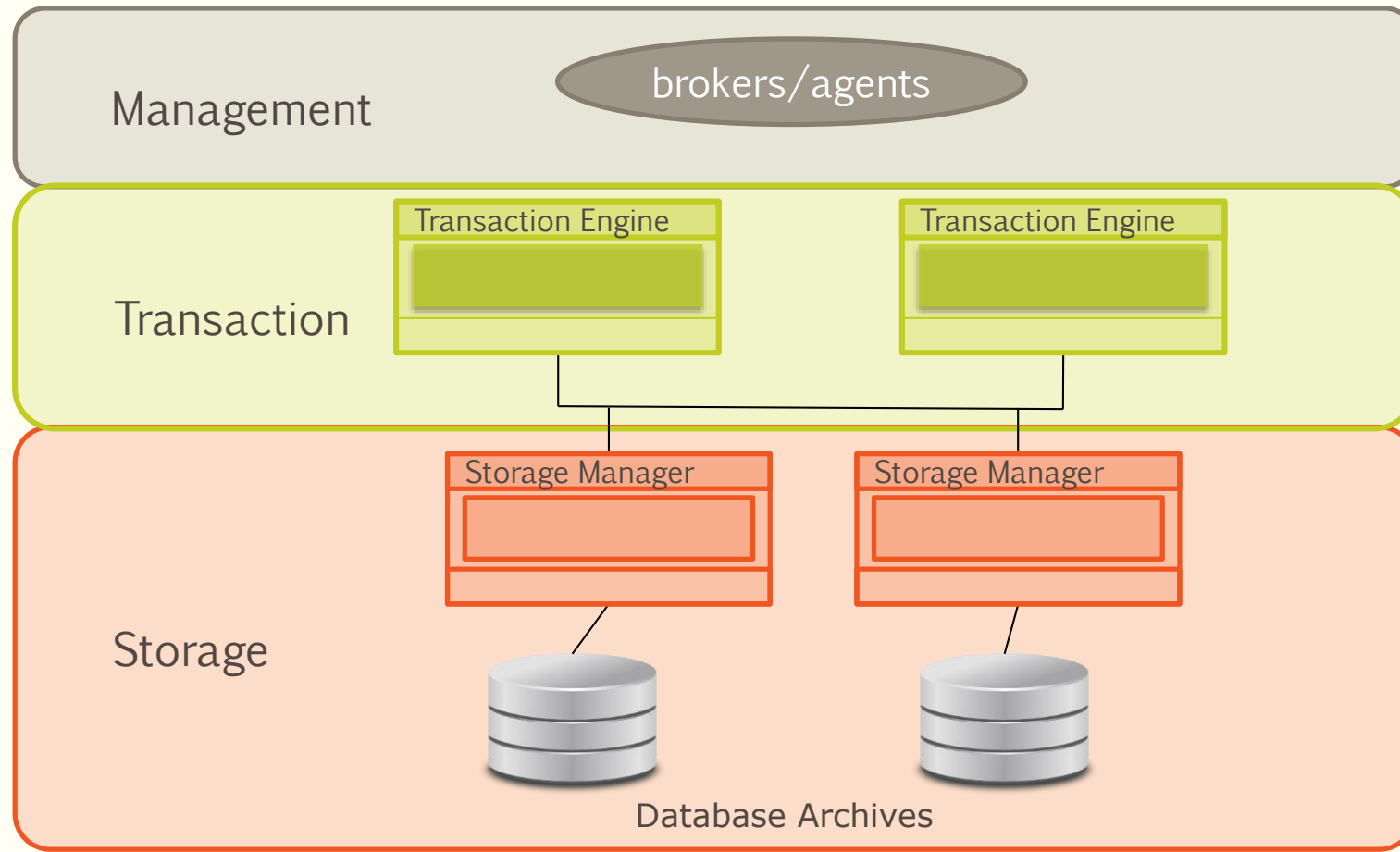
NuoDB: Architecture

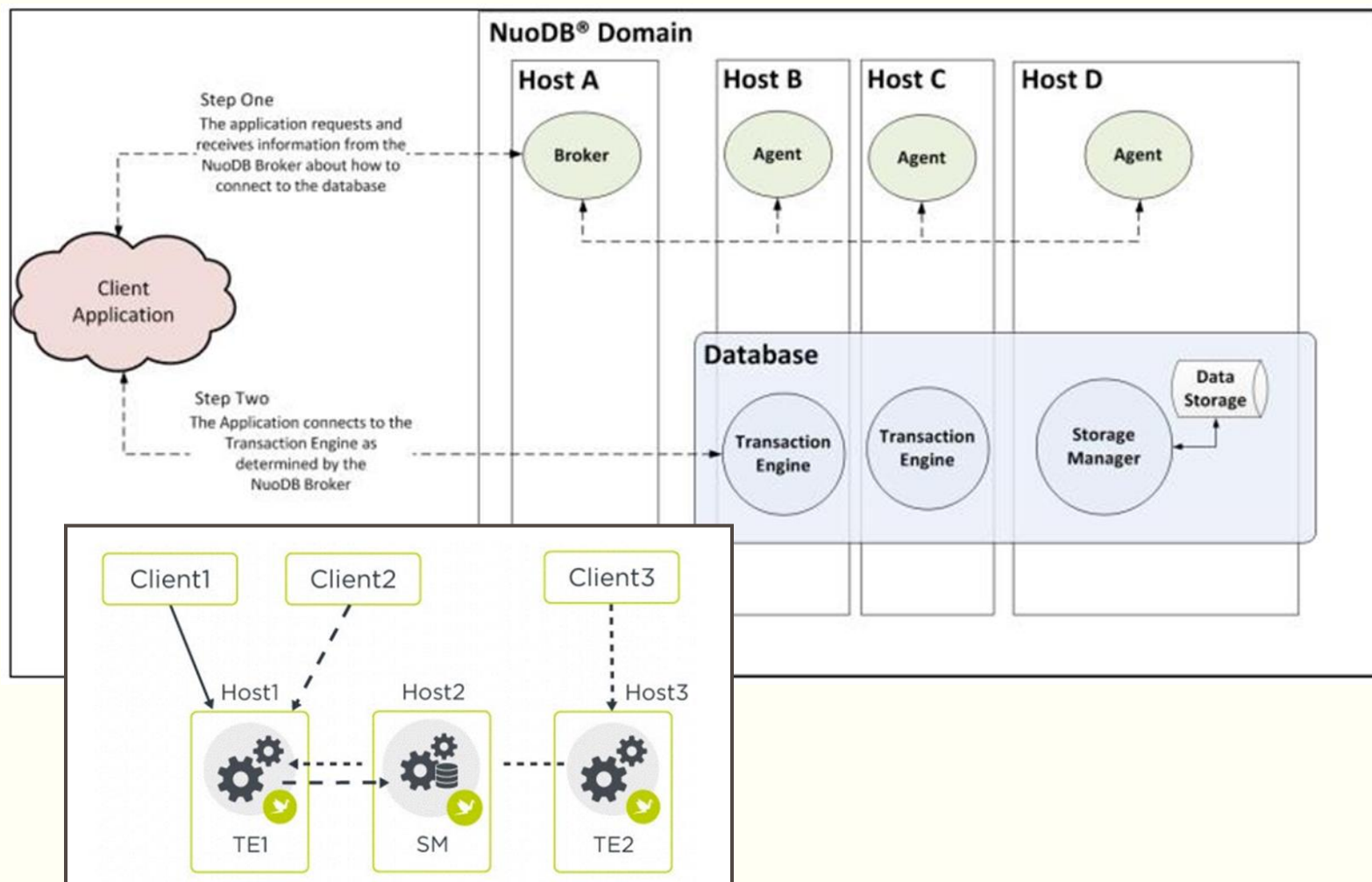
- Multi-tier Architecture
 - Transaction tier
 - Storage tier
 - Management tier
- Multi-Tenant
- Heavy use of memory
 - hot data stays in memory
 - Cold data in persistent store
- Object Oriented
 - Objects are atoms
- Asynchronous Messaging
- Partial, On-Demand replication
- MVCC - Concurrency

Technical Overview – Tiered Architecture

- Tiered Architecture
 - Transactions: Transaction Engine
 - Parse, compile, optimize and execute SQL commands
 - Stores some information in memory locally
 - Map to locate the information
 - Any transaction engine can service any piece of information regardless of where it resides
 - Adding transaction engines -> More throughput
 - Storage: Storage manager
 - Can run on HDFS or Amazon S3
 - Adding storage manager -> more resistance to failure
 - Management: Agents and Brokers
 - (Re)starting transaction engines and storage managers
 - Collect statistics from them
 - Clients connect to TE via Brokers

Multi-tier Architecture





Conclusions

- NewSQL is an established trend with a number of options
- Hard to pick one because they're not on a common scale No silver bullet
- Growing data volume requires ever more efficient ways to store and process it

oldSQL, NoSQL and NewSQL: pros/cons

- OldSQL
 - +: proven techs and standard SQL, ACID
 - +: rich client-side transaction
 - +: established market
 - -: not a scale-out architecture: single machine bottleneck
 - -: complex management, tuning for performance
- NoSQL
 - +: higher availability and scalability
 - +: support non-relational data, and OLAP
 - -: fundamentally no ACID
 - -: require application code; lack support for declarative queries
- NewSQL
 - +: stronger consistency and often full transaction support
 - +: familiar SQL and standard tooling
 - +: richer analytics using SQL and extensions
 - +: leverages noSQL-style clustering
 - -: no NewSQL is as general-purpose as traditional SQL systems
 - -: in-memory architecture may hinder the application for large dataset
 - -: partial access to the rich tooling of traditional SQL

