CPT-S 415

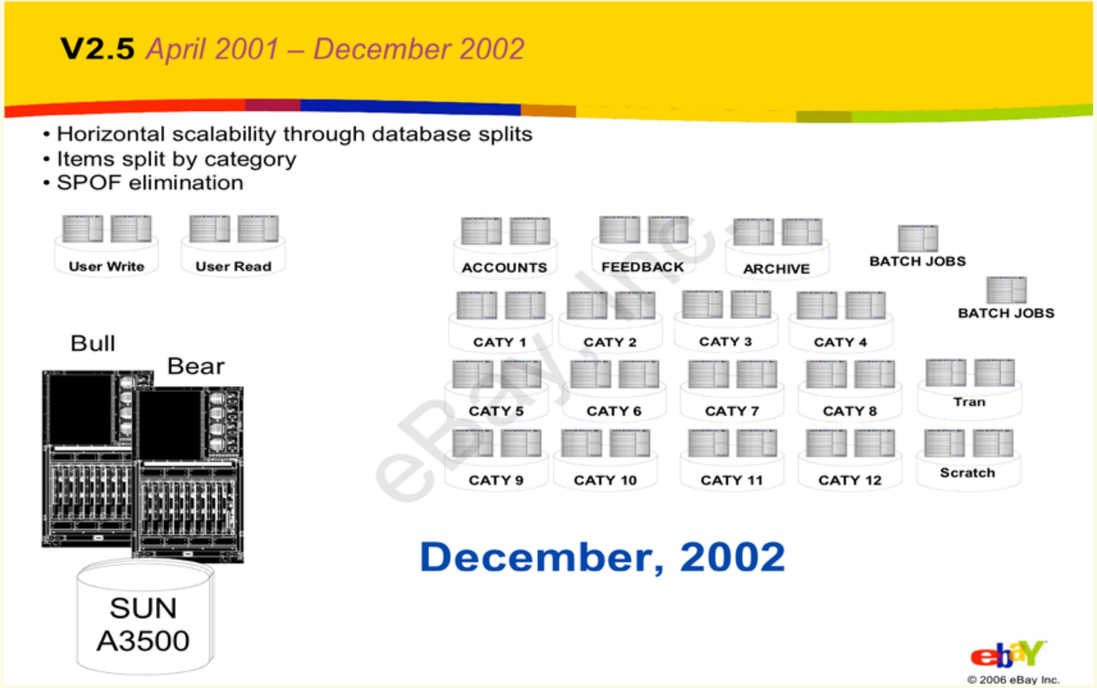
Big Data

Yinghui Wu

EME B45

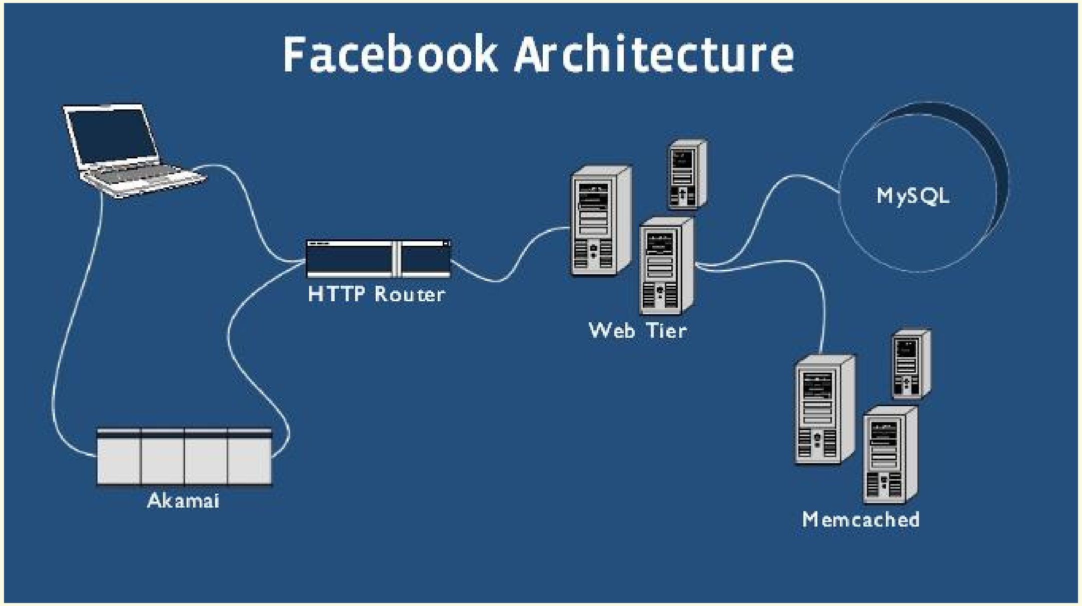
Let’s go back to Early-2000s…

- 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.



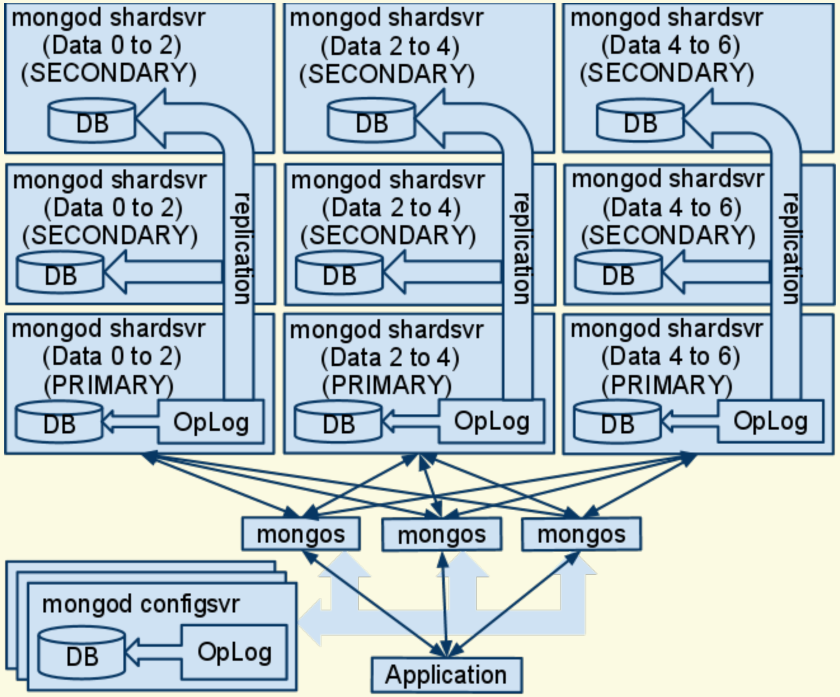
Mid-2000s

- ✓ 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.



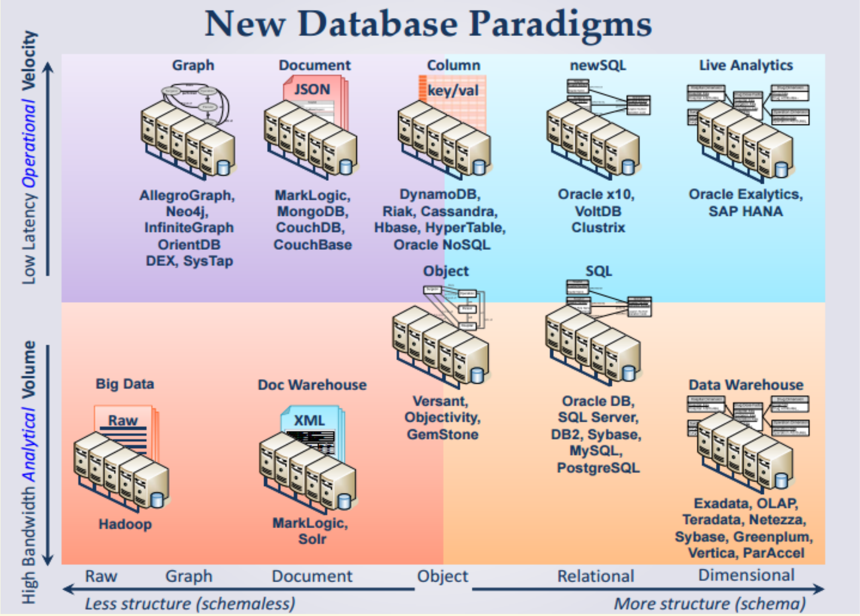
Early-2010s

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

- "New SQL"

NUODB
VoltDB
H-Store
SQLFire
SCALEBASE
JustOneDB
AKIBAN
Tokutek
xeround
GENIEDB
TRANSLATTICE
ScaleArc
Clustrix
dbShards
ScaleDB
SCHOONER

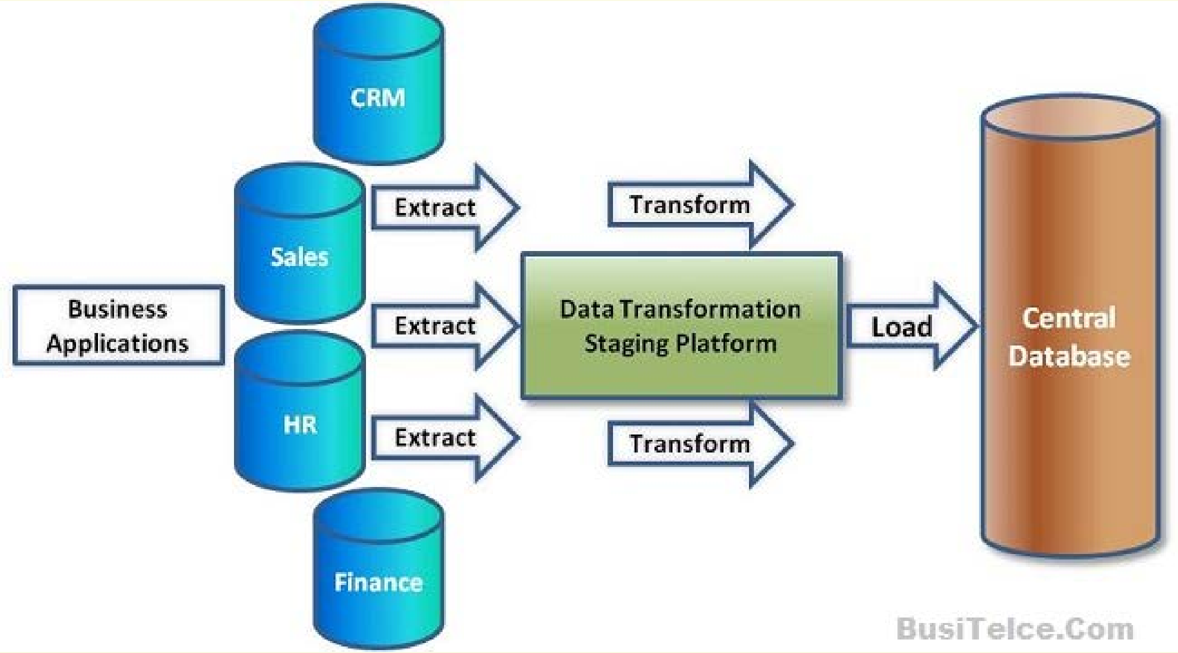
newSQL



New Database Paradigms
Low Latency Operational Velocity
High Bandwidth Analytical Volume
Graph
Document
JSON
Column
key/val
newSQL
Live Analytics
AllegroGraph, Neo4j, InfiniteGraph OrientDB DEX, SysTap
MarkLogic, MongoDB, CouchDB, CouchBase
DynamoDB, Riak, Cassandra, Hbase, HyperTable, Oracle NoSQL
Oracle x10, VoltDB Clustrix
Oracle Exalytics, SAP HANA
Object
SQL
Big Data
Raw
Hadoop
Doc Warehouse
XML
MarkLogic, Solr
Versant, Objectivity, GemStone
Oracle DB, SQL Server, DB2, Sybase, MySQL, PostgreSQL
Data Warehouse
Exadata, OLAP, Teradata, Netezza, Sybase, Greenplum, Vertica, ParAccel
Raw
Graph
Document
Object
Relational
Dimensional
Less structure (schemaless)
More structure (schema)

old OLTP and old SQL

- ✓ An information system can be transactional (OLTP) or/and analytical (OLAP)
- ✓ **OLTP** ...rt on-line transa...
  - ve...
  - da...
  - ef...
- ✓ schem...
- ✓ Old O...
  - Co...
  - Co...
  - Co...



- ✓ Old SQL: techs, systems and vendors supporting old OLTP

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

- ✓ Ingest the firehose in real time
- ✓ Process, validate, enrich and respond in Real-time
- ✓ Real-time analytics

- ✓ Options:
  - Old SQL - Legacy RDBMS vendors
  - NoSQL: give up SQL and ACID
  - NewSQL: SQL + ACID + new architecture

noSQL

- ✓ 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

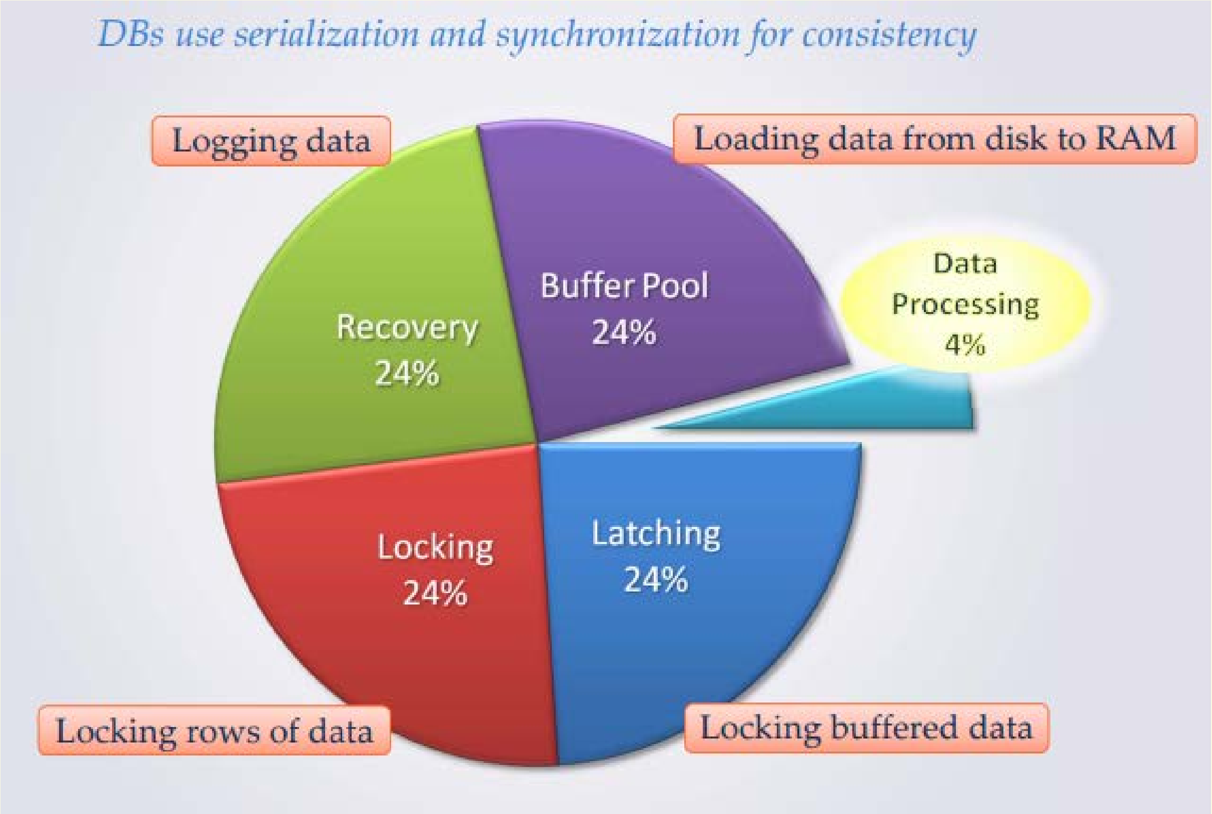
✓ "A DBMS that delivers the scalability and flexibility promised by NoSQL while retaining the support for SQL queries and/or ACID, or to improve performance for appropriate workloads."

-- ***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 databases

NewSQL: categories

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

voltDB

VoltDB

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

Technical Overview

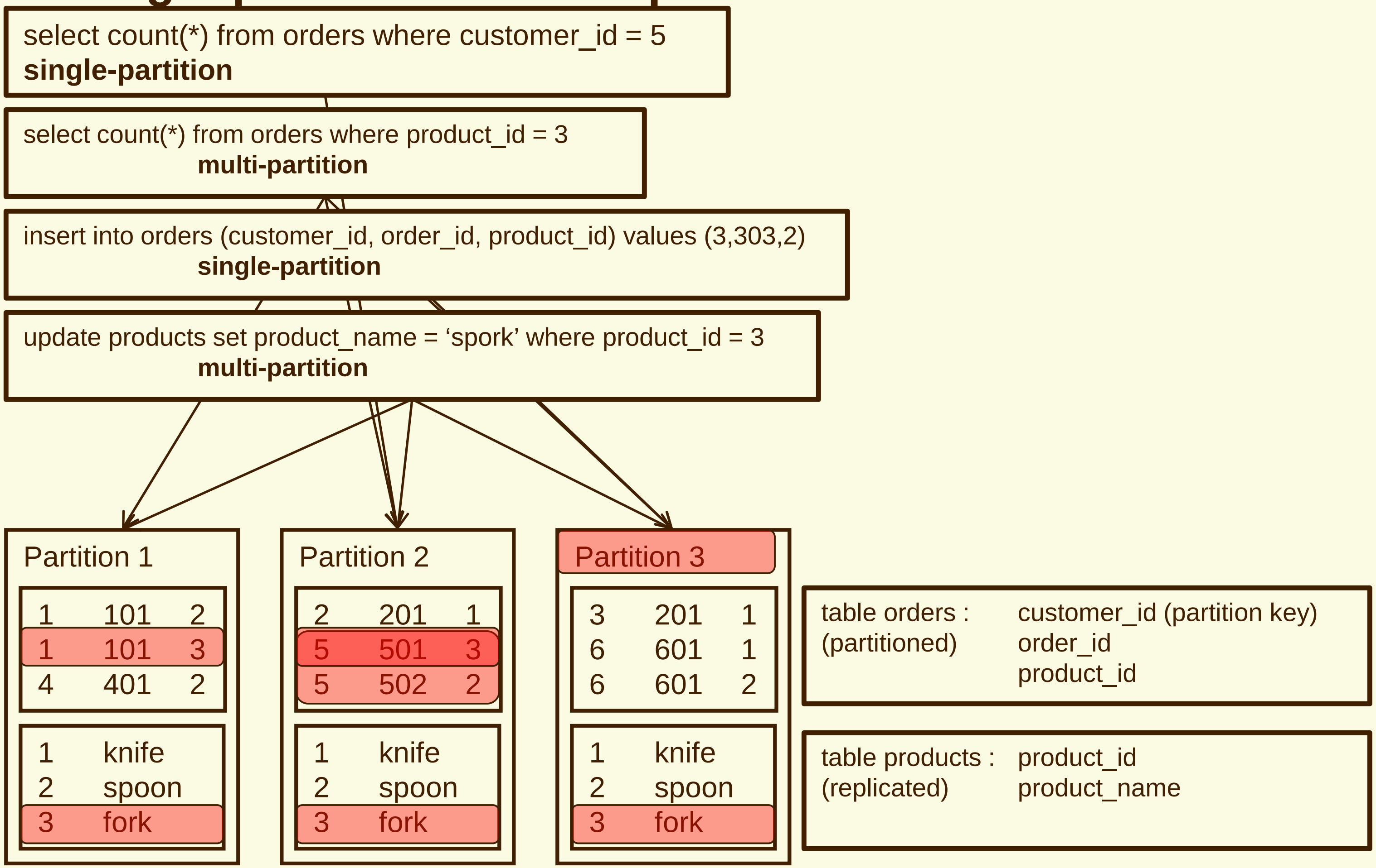
- ✓ VoltDB tries to avoid the overhead of traditional databases
  - ✓ K-safety for fault tolerance
  - ✓ In memory operation for maximum throughput
    - reduce buffer management
  - ✓ Partitions operate autonomously and single-threaded
    - no latching or locking
- ✓ Built to horizontally scale

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
    - <u>All</u> rows exist within <u>all</u> 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

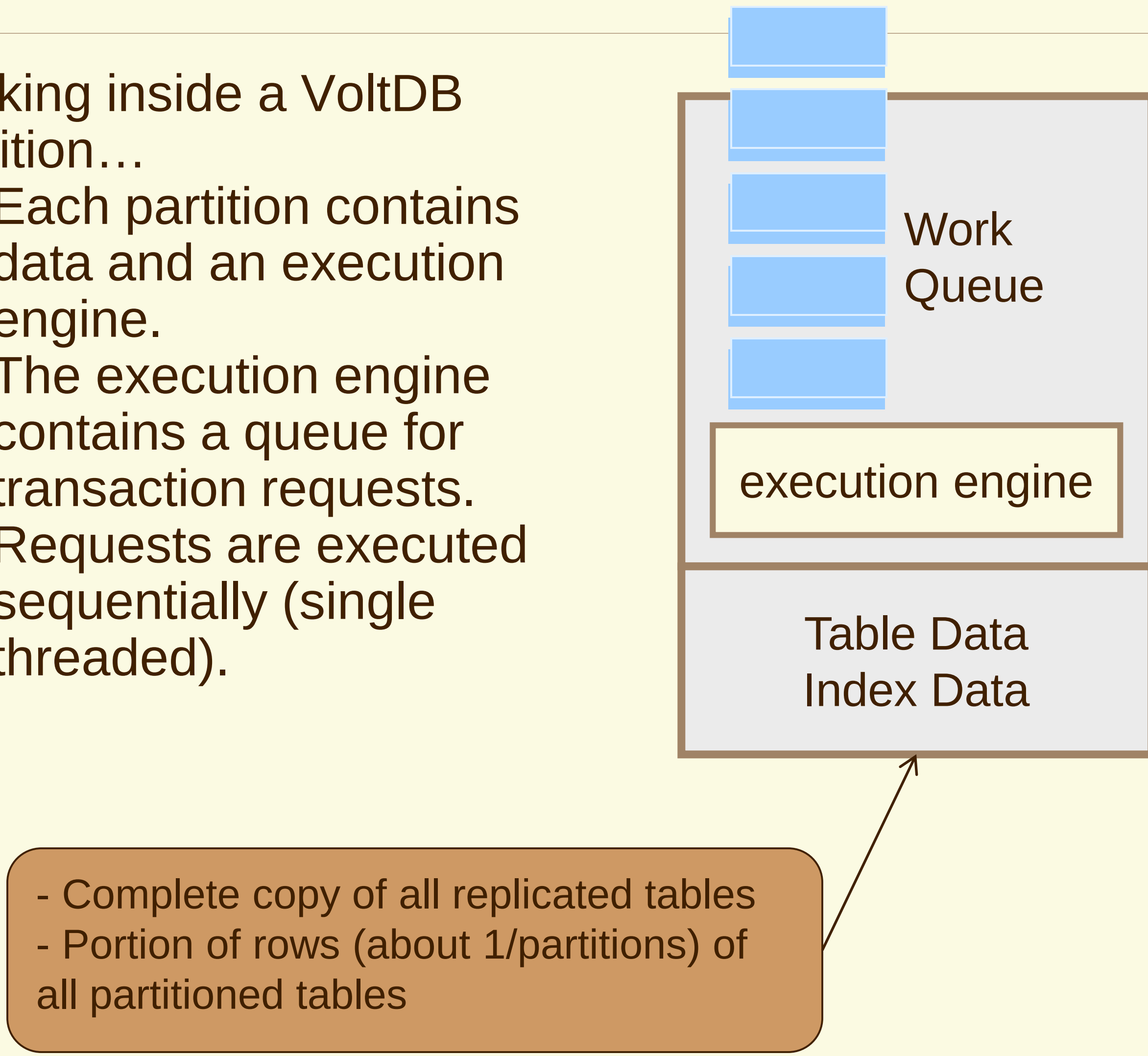
Technical Overview – Partitions (2/3)

✓ Single-partition vs. Multi-partition



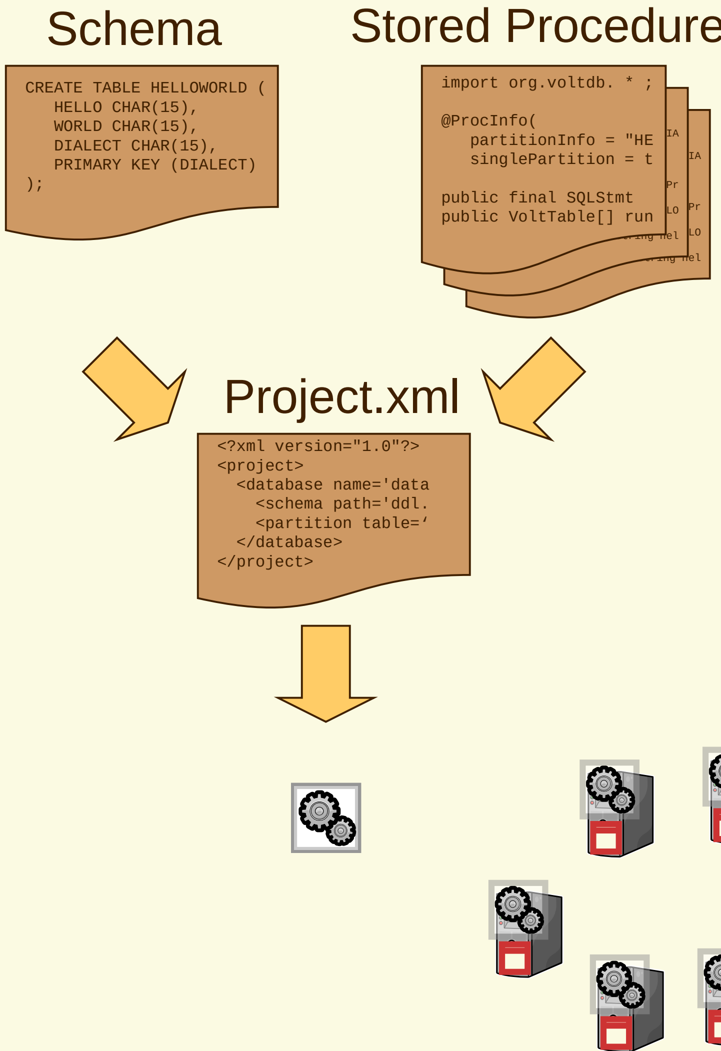
Technical Overview – Partitions (3/3)

- 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).

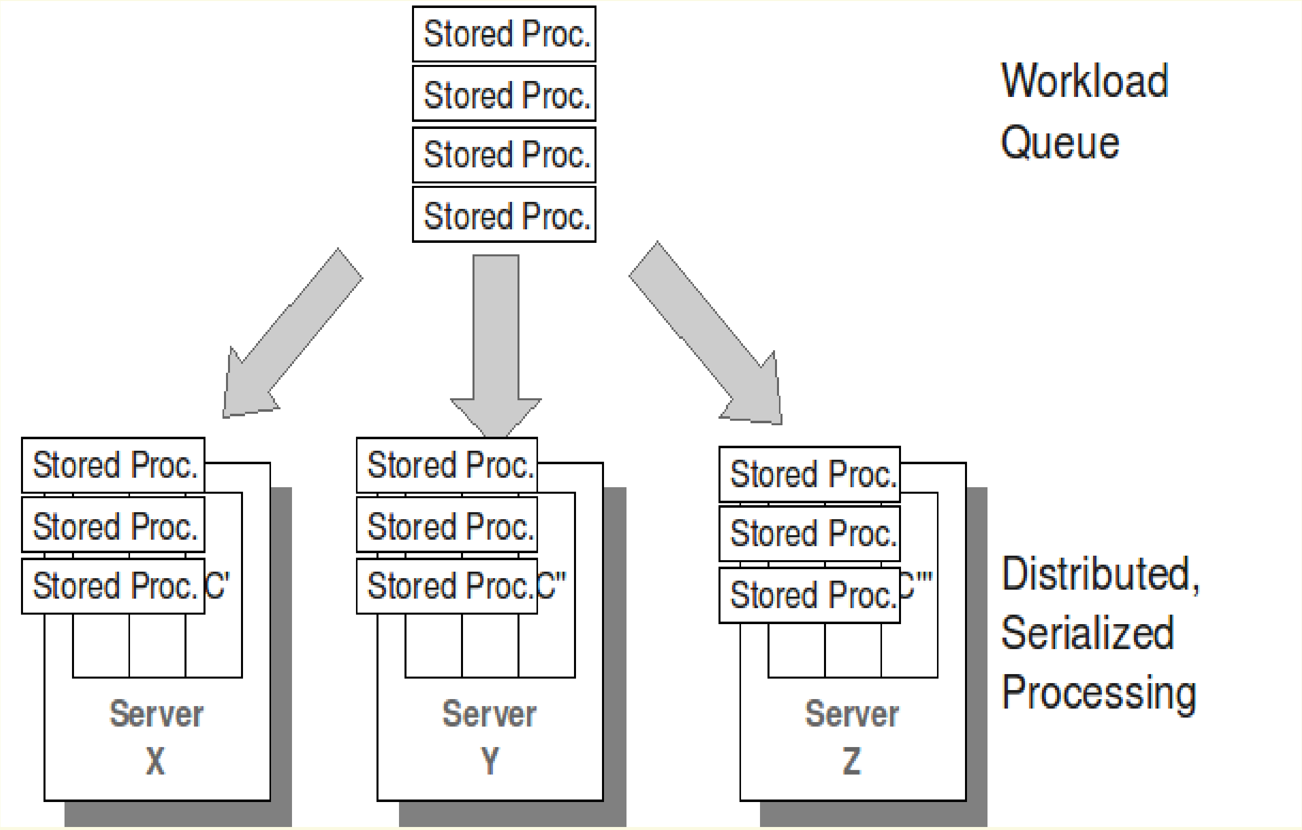


Technical Overview – Compiling

- ✓ 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



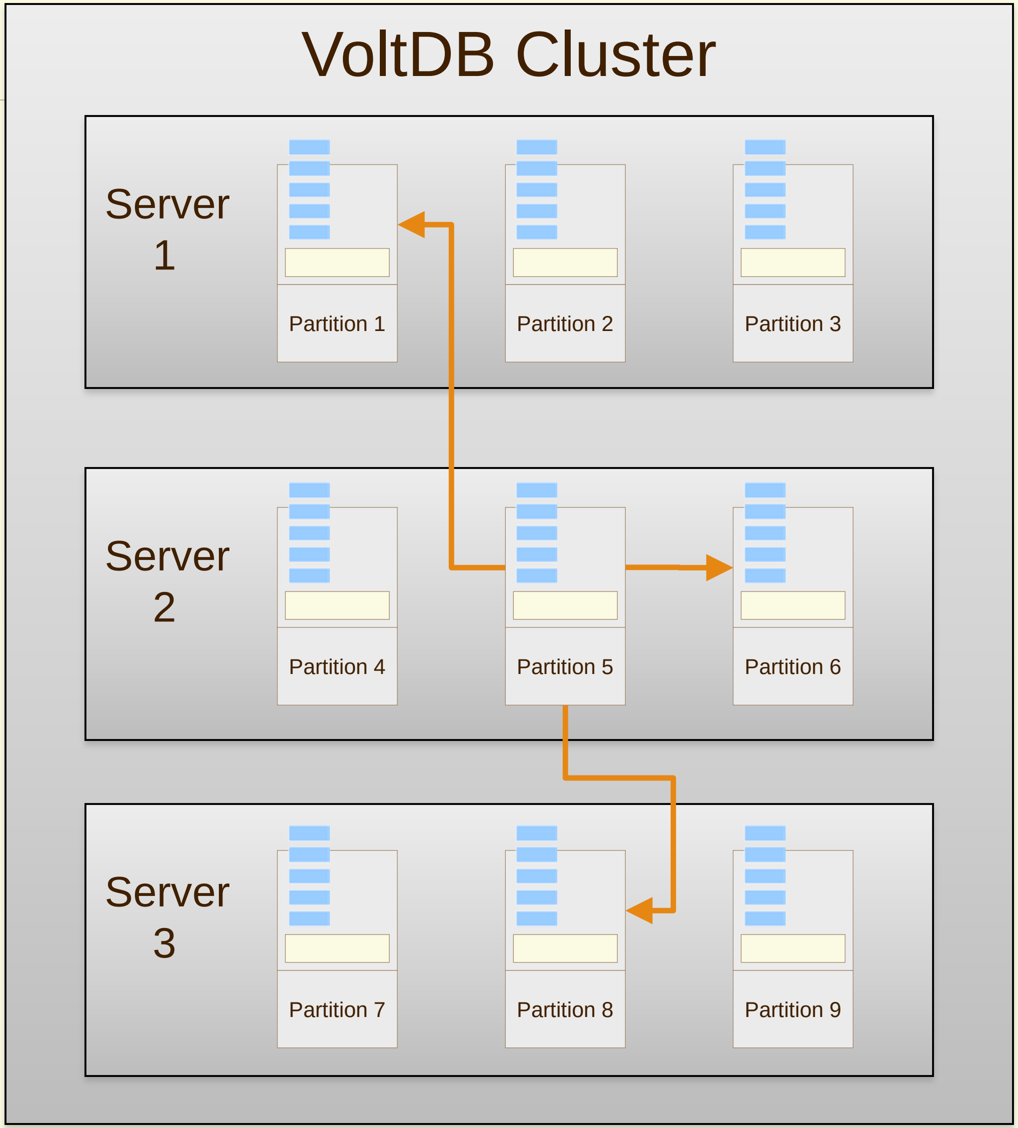
Transaction Model



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

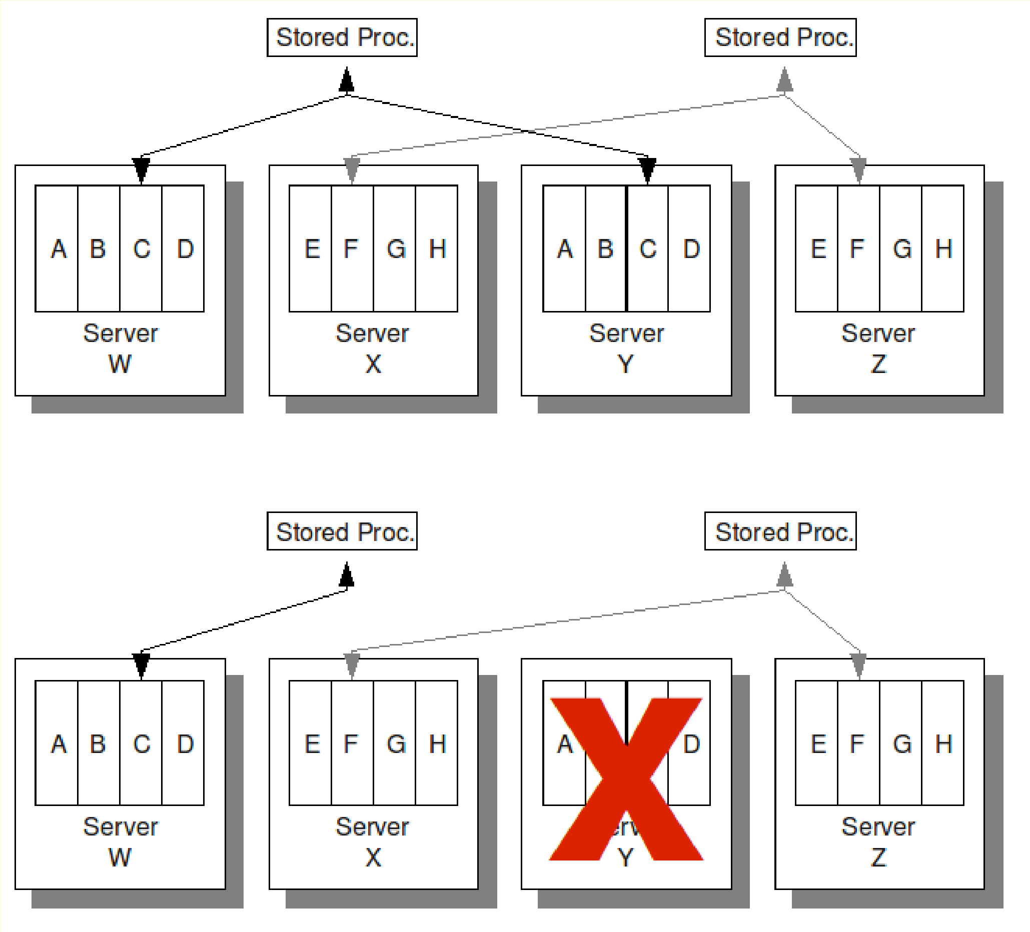


Data Availability and Durability

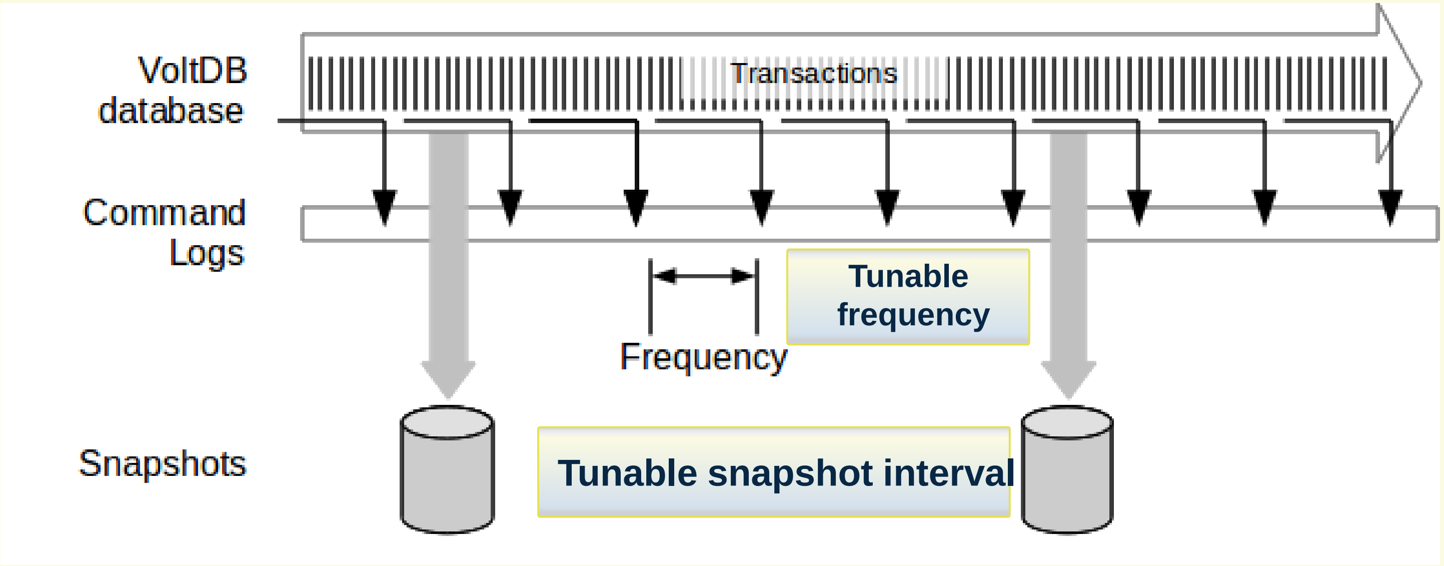
- ✓ High Availability
  - Data stored on server replicas (user configurable)
  - Failover data redundancy
  - No single point of failure
- ✓ Database Snapshots
  - Simplifies backup/restore
  - Scheduled, continuous, on demand
  - Cluster-wide consistent copy of all data
- ✓ Command Logging
  - Between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

K-safety

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



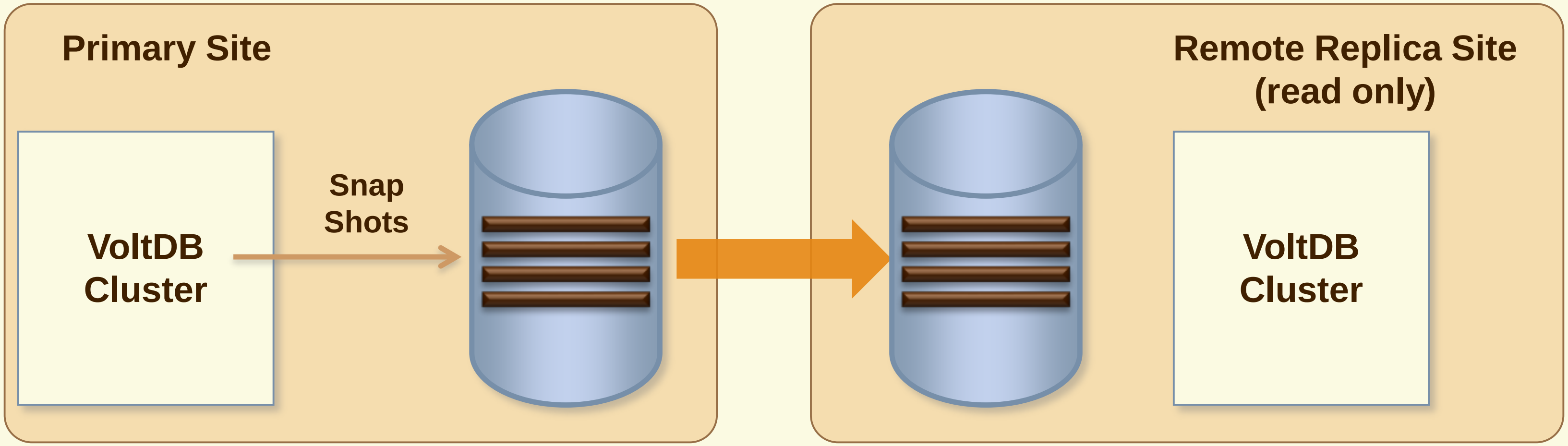
Command Logging



- 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

Summary: 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*

nuoDB

nuoDB

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

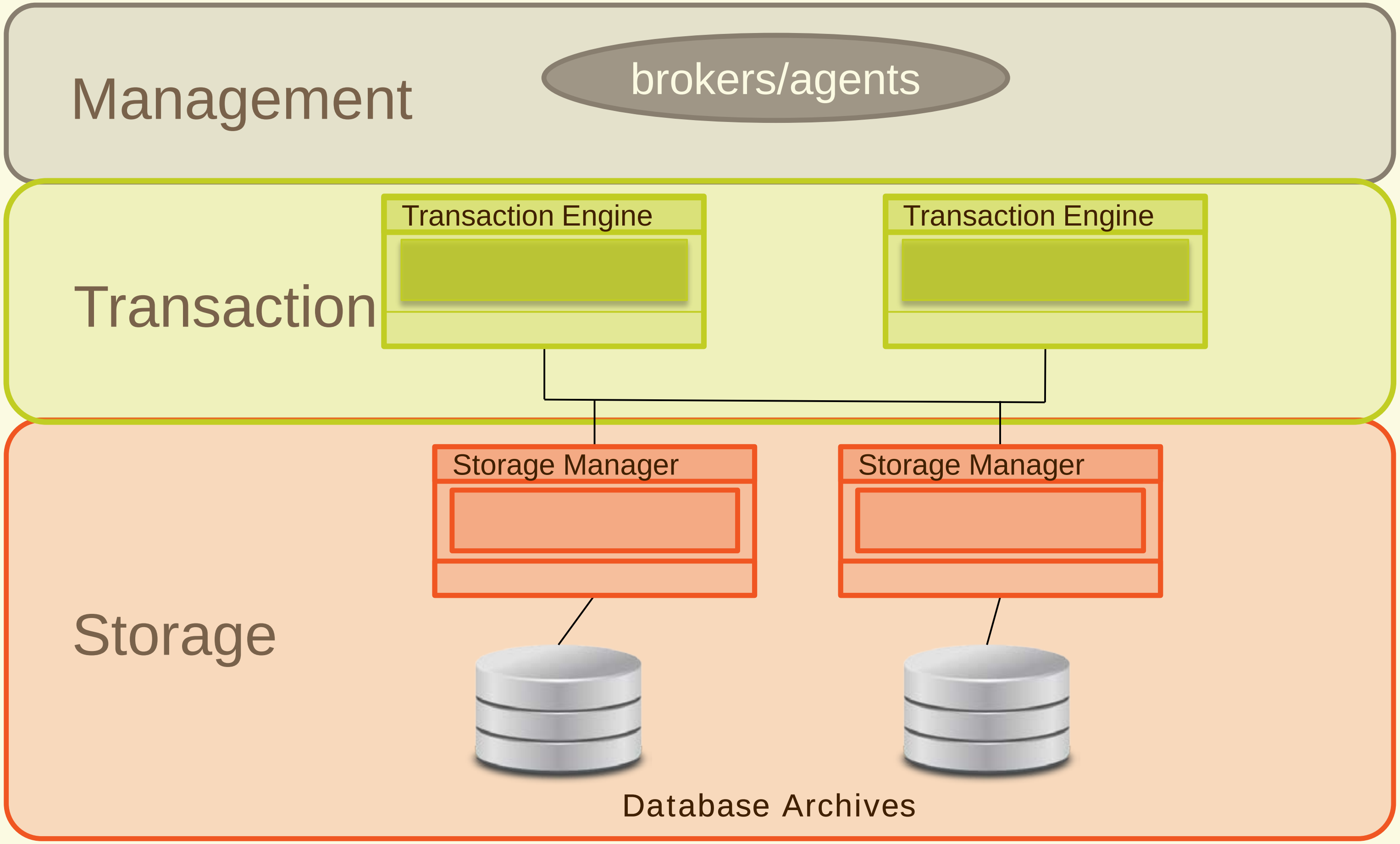
NuoDB: Architecture

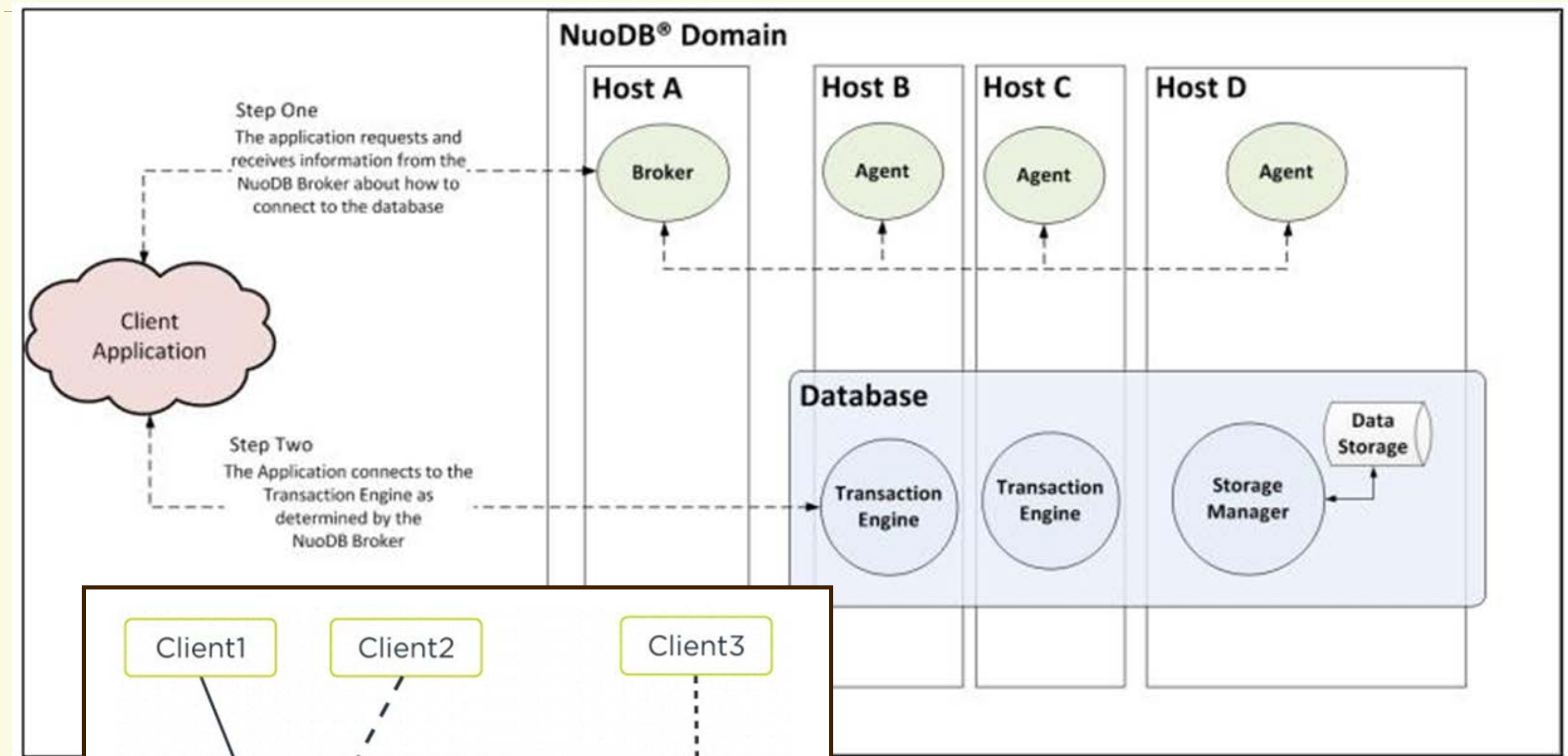
- ✓ Multi-tier Architecture
  - – Transaction tier
  - – Storage tier
  - – Management tier
- ✓ Multi-Tenant
- ✓ Heavy use of memory
  - – hot data stays in memory
  - – Cold data in persistent store
- ✓ Object Oriented
  - – Objects are atoms
- ✓ Asynchronous Messaging
- ✓ Partial, On-Demand replication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

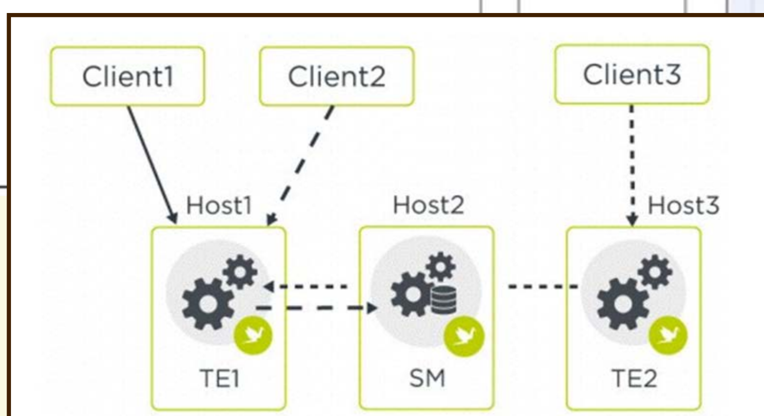
Technical Overview – Tiered Architecture

- ✓ 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




NuoDB® Domain
Host A
Host B
Host C
Host D
Step One
The application requests and receives information from the NuoDB Broker about how to connect to the database
Broker
Agent
Agent
Agent
Client Application
Database
Data Storage
Transaction Engine
Transaction Engine
Storage Manager
Step Two
The Application connects to the Transaction Engine as determined by the NuoDB Broker


Client1
Client2
Client3
Host1
Host2
Host3
TE1
SM
TE2

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 t
  - +: rich clie
  - +: establis
  - -: not a sca
  - -: complex
- NoSQL
  - +: higher a
  - +: support
  - -: fundame
  - -: require a
- NewSQL
  - +: stronge
  - +: familiar
  - +: richer a
  - +: leverage
  - -: 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