VoltDB: an SQL Developer's Perspective

Tim Callaghan, VoltDB Field Engineer
tcallaghan@voltdb.com

Scaling Traditional OLTP Databases

- Sharding improves performance but introduces…
  - Management complexity
    - + disjointed backup/recovery and replication
    - + manual effort to re-partition data
  - Application complexity
    - + shard awareness
    - + cross partition joins
    - + cross partition transactions
  - And, each shard still suffers from traditional OLTP performance limitations
- If you can shard, your application is probably great in VoltDB.

Technical Overview

- "OLTP Through the Looking Glass"

  http://cs-www.cs.yale.edu/homes/dna/papers/oltpperf-sigmod08.pdf

- VoltDB avoids the overhead of traditional databases
  - K-safety for fault tolerance

  - no logging

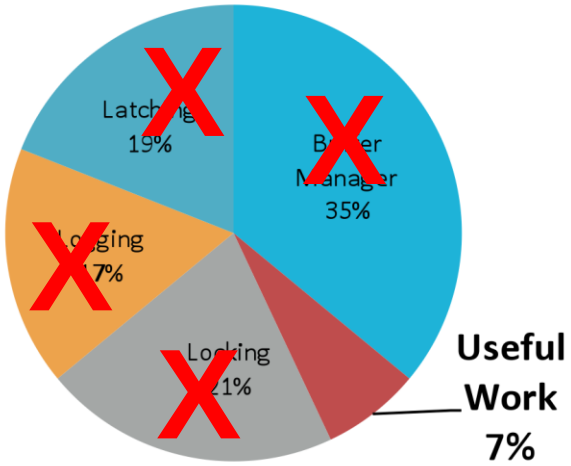
  - In memory operation for maximum throughput

  - no buffer management

  - Partitions operate autonomously and single-threaded

  - no latching or locking

- Built to horizontally scale

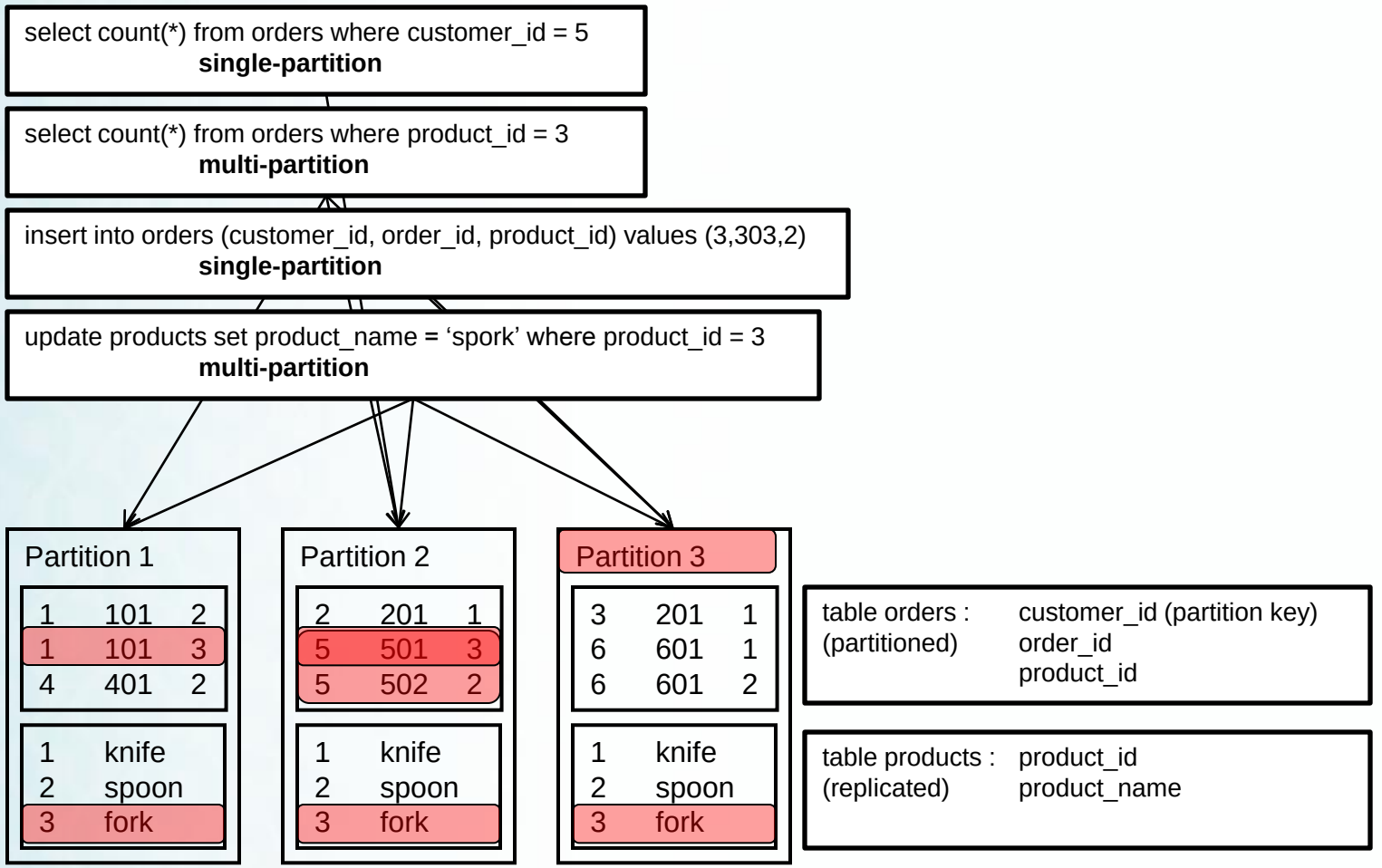


Technical Overview – Partitions (1/3)

- 1 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

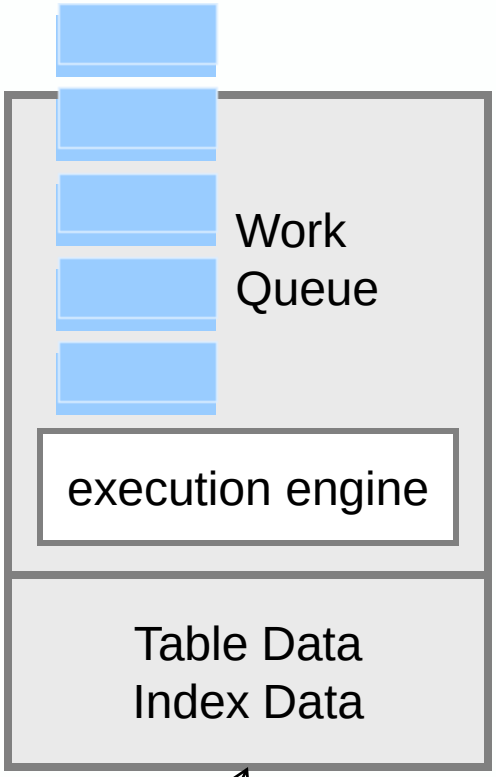
Technical Overview – Partitions (2/3)

- **Single-partition vs. Multi-partition**



Technical Overview – Partitions (3/3)

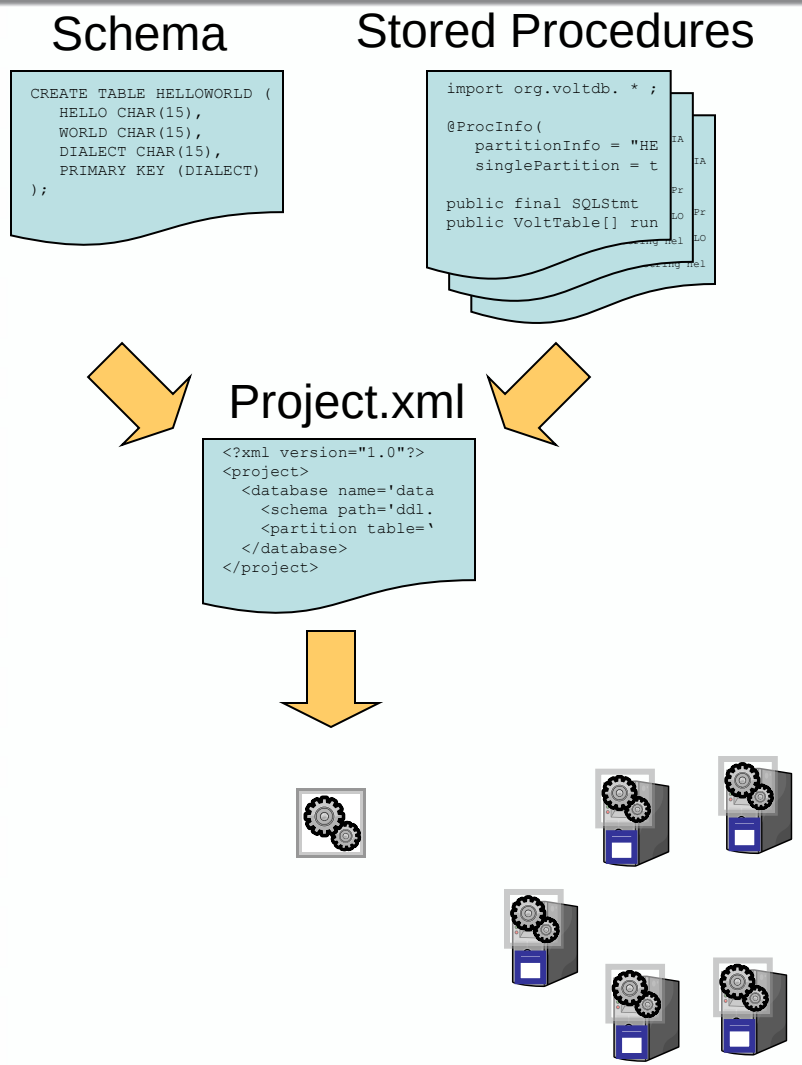
- Looking inside a VoltDB partition…
  - Each partition contains data and an execution engine.
  - The execution engine contains a queue for transaction requests.
  - Requests are executed sequentially (single threaded).



- Complete copy of all replicated tables
- Portion of rows (about 1/partitions) of all partitioned tables

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



Technical Overview - Transactions

- All access to VoltDB is via Java stored procedures (Java + SQL)
- A single invocation of a stored procedure is a transaction (committed on success)
- Limits round trips between DBMS and application
- High performance client applications communicate asynchronously with VoltDB

Technical Overview – Clusters/Durability

- Scalability
  - Increase RAM in servers to add capacity
  - Add servers to increase performance / capacity
  - Consistently measuring 90% of single-node performance increase per additional node
- High availability
  - K-safety for redundancy
- Snapshots
  - Scheduled, continuous, on demand
- Spooling to data warehouse
- Disaster Recovery/WAN replication (Future)
  - Asynchronous replication

Table/Index Storage

- VoltDB is entirely in-memory
- Cluster must collectively have enough RAM to hold all tables/indexes (k + 1 copies)
- Even data distribution is important