



# VOLTDDB

## VoltDB

David Rolfe

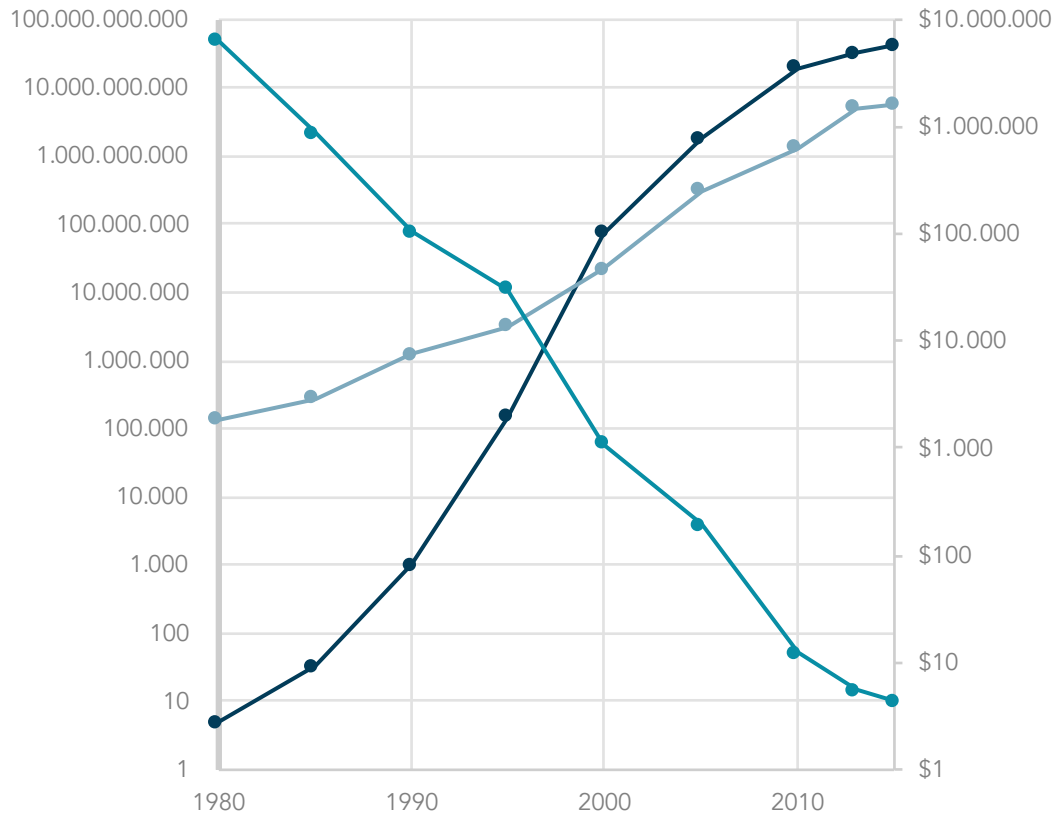
Director of Solution Architecture, EMEA

Doug Jauregui

Sr. Solutions Engineer

# Legacy database technology is obsolete

Internet Traffic (GB Month), Transistors per CPU and Cost of RAM over time



—●— Total Internet Bandwidth (GB/Mo)

—●— Transistors per CPU

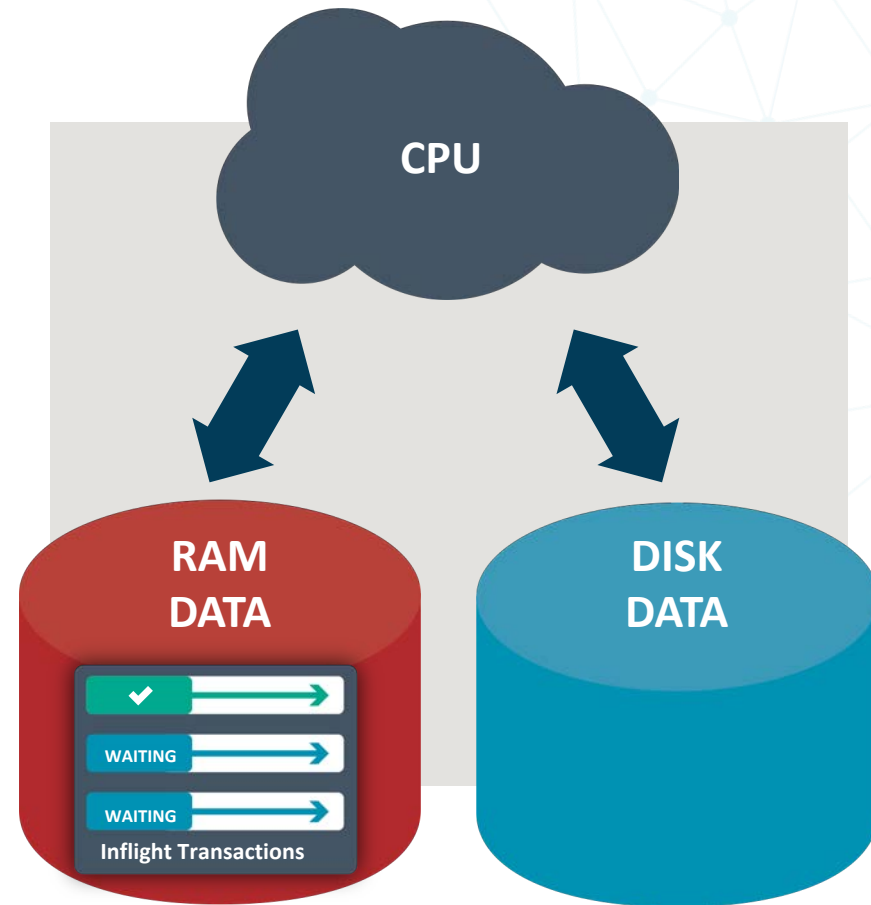
—●— Price of Ram (\$/GB)

- Legacy RDBMS designs date from about 1985.
- Vendors are finding legacy databases increasingly uneconomic.
- Legacy databases struggle to scale beyond 2 nodes.
- But demand for transactions is increasing all the time.
- Meanwhile Moore's law means hardware and RAM keeps getting cheaper

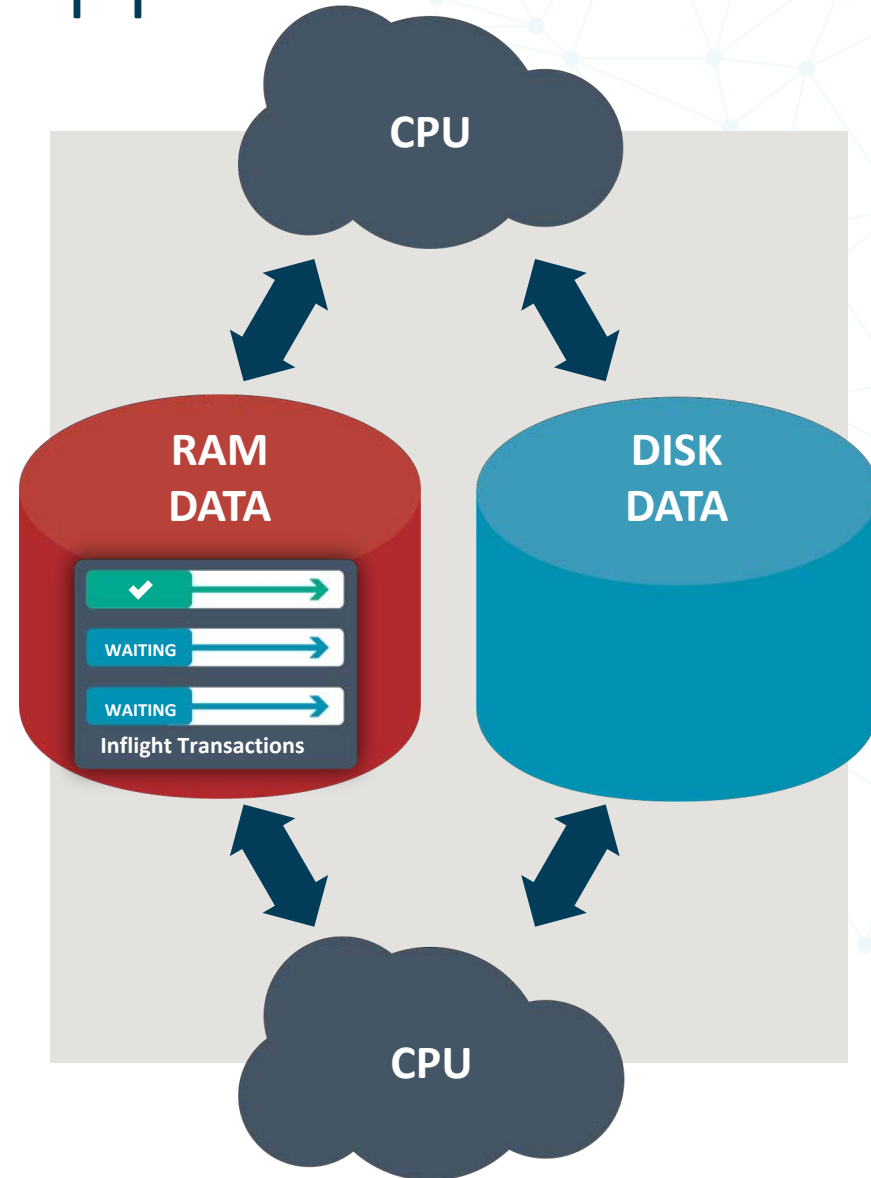
# 21<sup>st</sup> Century Requirements for transaction processing

- Virtualization friendly .
- ACID transactions.
- Millisecond response times.
- No "Long Tail"
- Supports complicated logic
- Easily scalable beyond 2 nodes.
- HA "Just Happens"
- Geo replication
- "Translytics" / "HTAP"

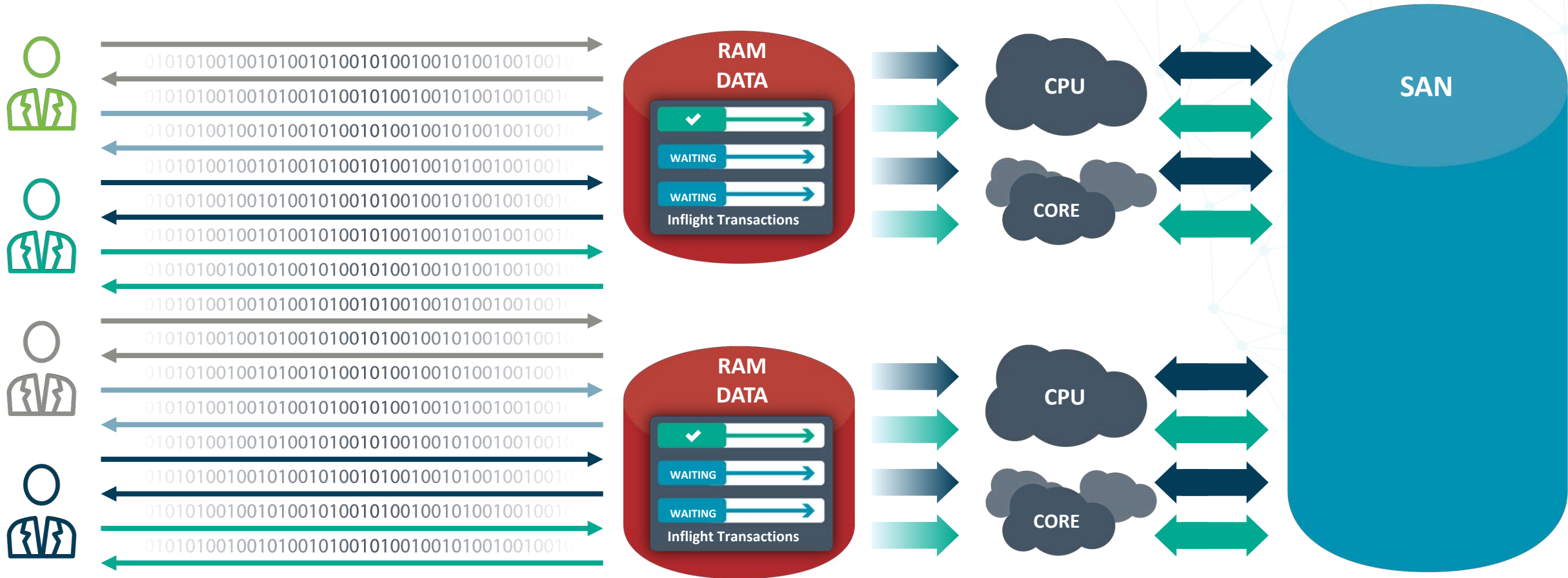
# RDBMS - How We Thought an RDBMS Worked



# RDBMS - What Actually Happens – Part 1...



# RDBMS - What Actually Happens – Part 2





# If we tried this in a supermarket...



# Dr. Michael Stonebraker found a solution..

### The End of an Architectural Era (It's Time for a Complete Rewrite)

Michael Stonebraker  
 Samuel Madden  
 Daniel J. Abadi  
 Stavros Harizopoulos  
 MIT CSAIL  
 {stonebraker, madden, dja, stavros}@csail.mit.edu

Nabil Hachem  
 AvenirCarte Consulting, LLC  
 nhachem@avc.com

Pat Hell  
 Microsoft Corp  
 phell@microsoft.com

### OLTP Through the Looking Glass, and What We Found There

Stavros Harizopoulos    Daniel J. Abadi    Samuel Madden    Michael Stonebraker  
 HP Labs    Yale University    Massachusetts Institute of Technology  
 Palo Alto, CA    New Haven, CT    Cambridge, MA  
 stavros@hp.com    dne@cs.yale.edu    {madden, stonebraker}@csail.mit.edu

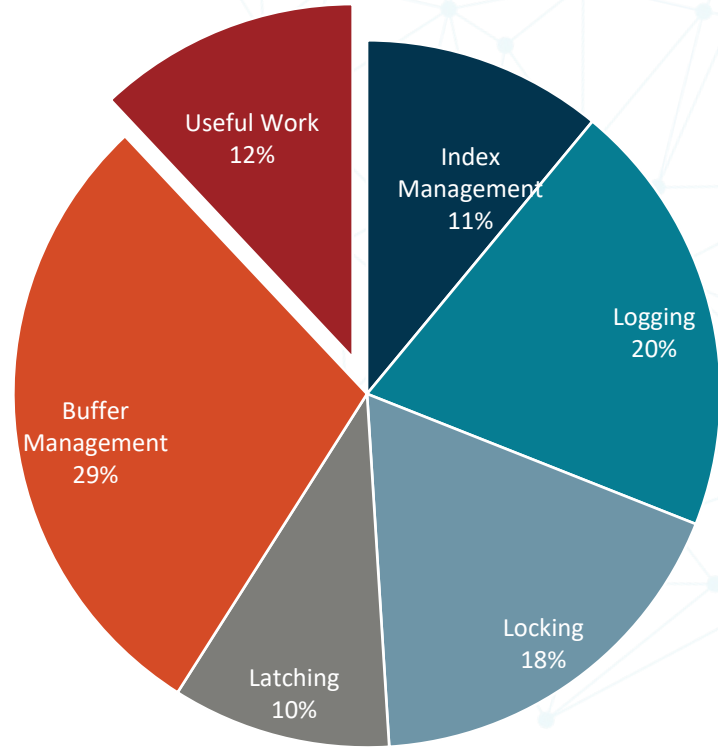
**ABSTRACT**  
 Online Transaction Processing (OLTP) databases include a suite of features — disk-resident B-trees and heap files, locking-based concurrency control, support for multi-threading — that were optimized for computer technology of the late 1970s. Advances in modern processors, memories, and networks mean that today's computers are vastly different from those of 30 years ago, such that many OLTP databases will now fit in main memory, and most OLTP transactions can be processed in milliseconds or less. Yet database architecture has changed little. Based on this observation, we look at some interesting variants of conventional database systems that one might build that exploit recent hardware trends, and speculate on their performance through a detailed instruction-level breakdown of the major components involved in a transaction processing database system (Shore) running a subset of TPC-C. Rather than simply profiling Shore, we progressively modified it so that after every feature removal or optimization, we had a (fairer) working system that fully ran our workload. Overall, we identify overheads and optimizations that explain a total difference of about a factor of 20x in raw performance. We also show that there is no single "high pole in the tent" in modern (memory resident) database systems, and that substantial time is spent in logging, locking, B-trees, and buffer management operations.

**Categories and Subject Descriptors:**  
 H.2.4 [Database Management]: Systems — transaction processing; concurrency.

**General Terms:**  
 Measurement, Performance, Experimentation.

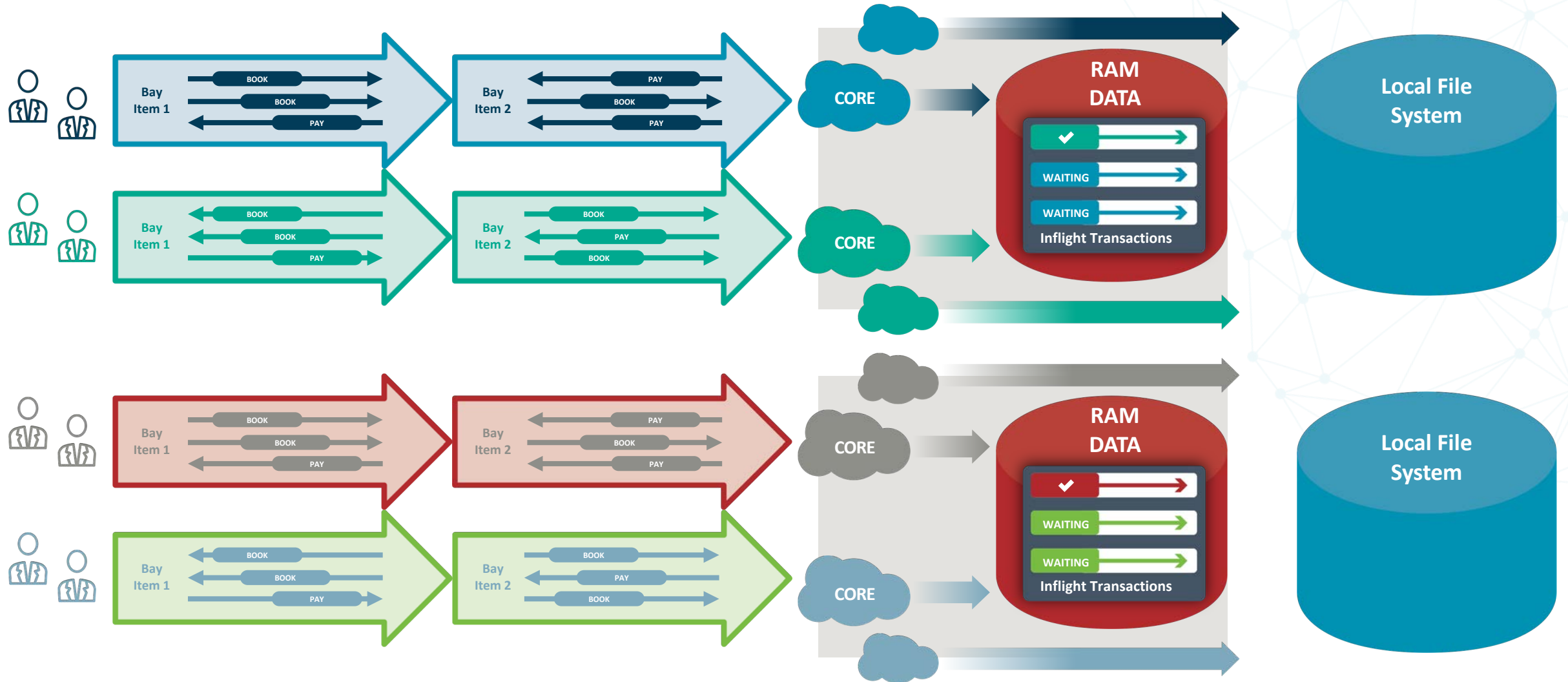
**Keywords:**  
 Online Transaction Processing, OLTP, main memory transaction processing, DBMS architecture.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made for profit or commercial advertising and that copies bear the notice and full citation on the first page. In very exceptional circumstances, permission to reproduce copies may be obtained from the ACM Publications Department. Copyright 2008 ACM 978-1-60558-100-9/08/0006...\$5.00

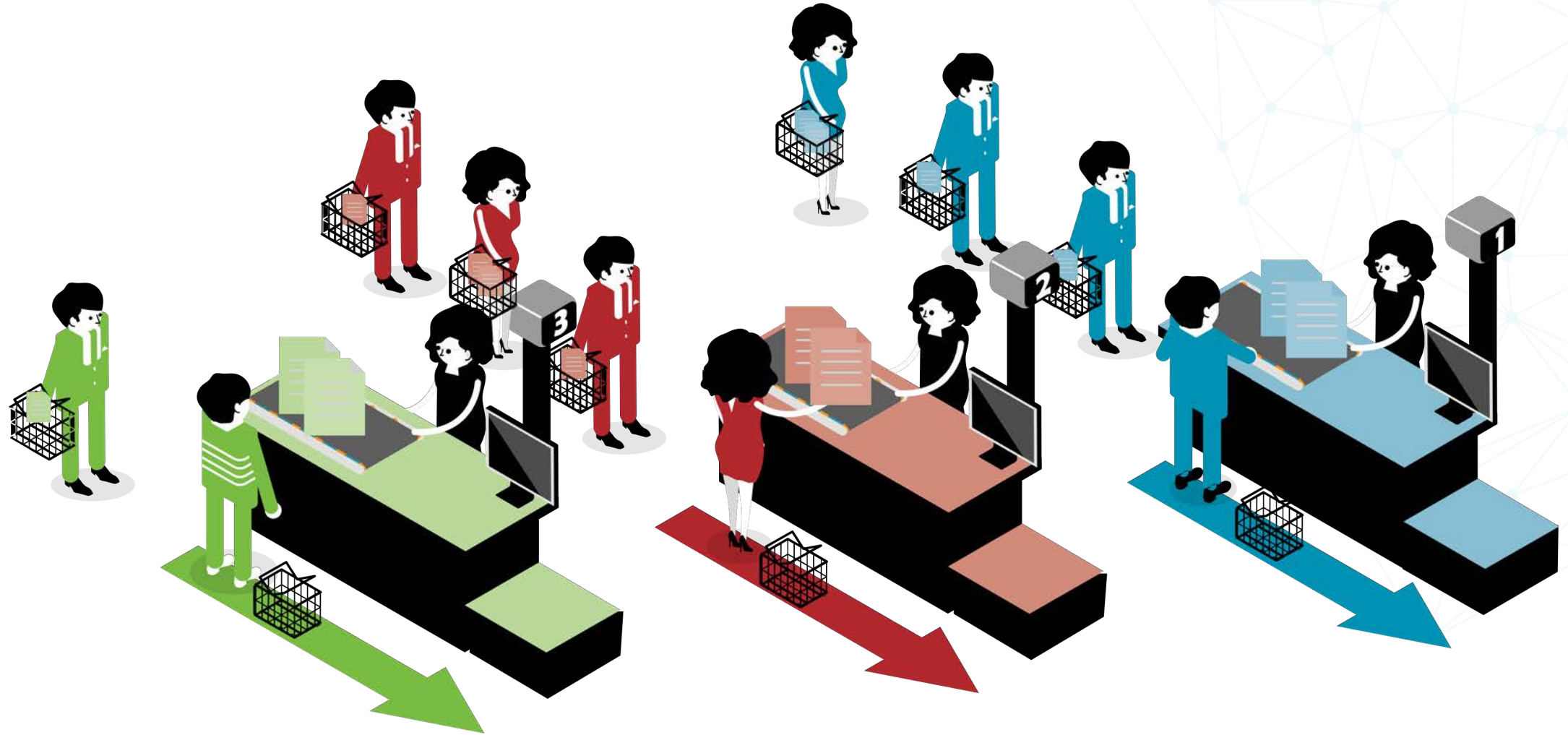




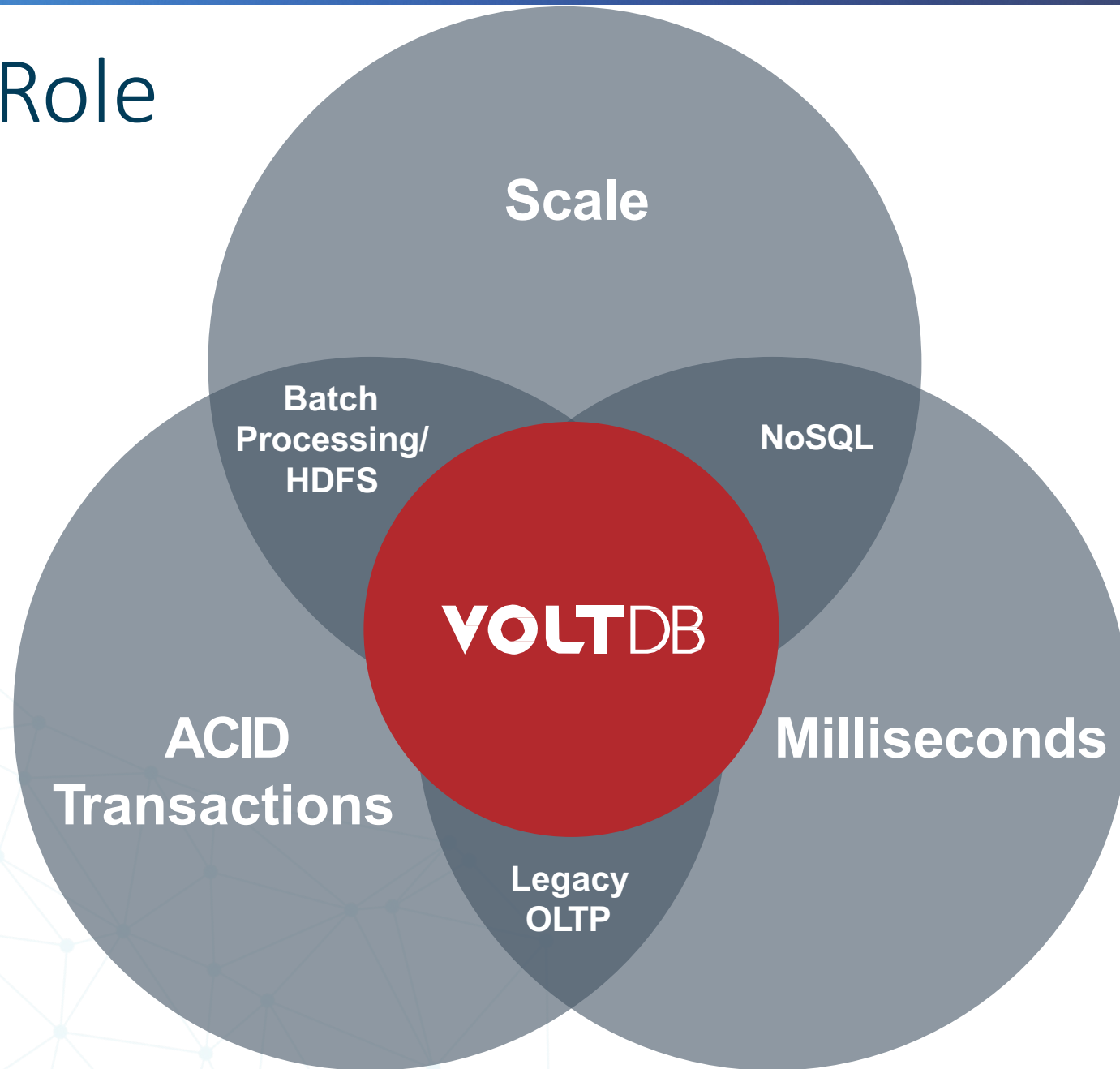
# How VoltDB works



# How a supermarket works...



# VoltDB's Role



# The only 3 ways to interact with any database

Approach	Examples	Strengths	Weaknesses
Many SQL Statements + Commit or Rollback	JDBC, ODBC,	Liked by developers, initial development is rapid	<ul style="list-style-type: none"><li>• Doesn't handle scaling OLTP loads well – DB spends its time figuring out who can see what instead of working</li><li>• Constant locking problems for shared, finite resources</li><li>• Failure of a client to Commit or Rollback causes a temporary resource leak</li></ul>
Move all the data to the client and back again	NoSQL, KV Stores	Very developer friendly	<ul style="list-style-type: none"><li>• Multiple updated copies of the data can arrive at the same time for scaling OLTP loads</li><li>• All of the data gets moved across the network, every time.</li></ul>
Stored Procedures	VoltDB, PL/SQL	Predictable speed and best possible scaling characteristics	<ul style="list-style-type: none"><li>• Not in fashion with developers.</li><li>• PL/SQL created perception of complexity.</li><li>• Other implementations of Java Stored Procedures really slow.</li></ul>

# A Proven and Reliable Partner

## Telco

Billing/rights management, subscriber data, etc.



## Financial Services

Risk, market data management, customer mgt.



## Personalize, Customize, Target

Ad optimization, audience segmenting, customer service



## IoT Platforms, Energy, Sensor

Smart grid/meters, asset tracing & management



## Infrastructure, Dashboards, KPIs

Data pipeline, system performance, streaming ETL.

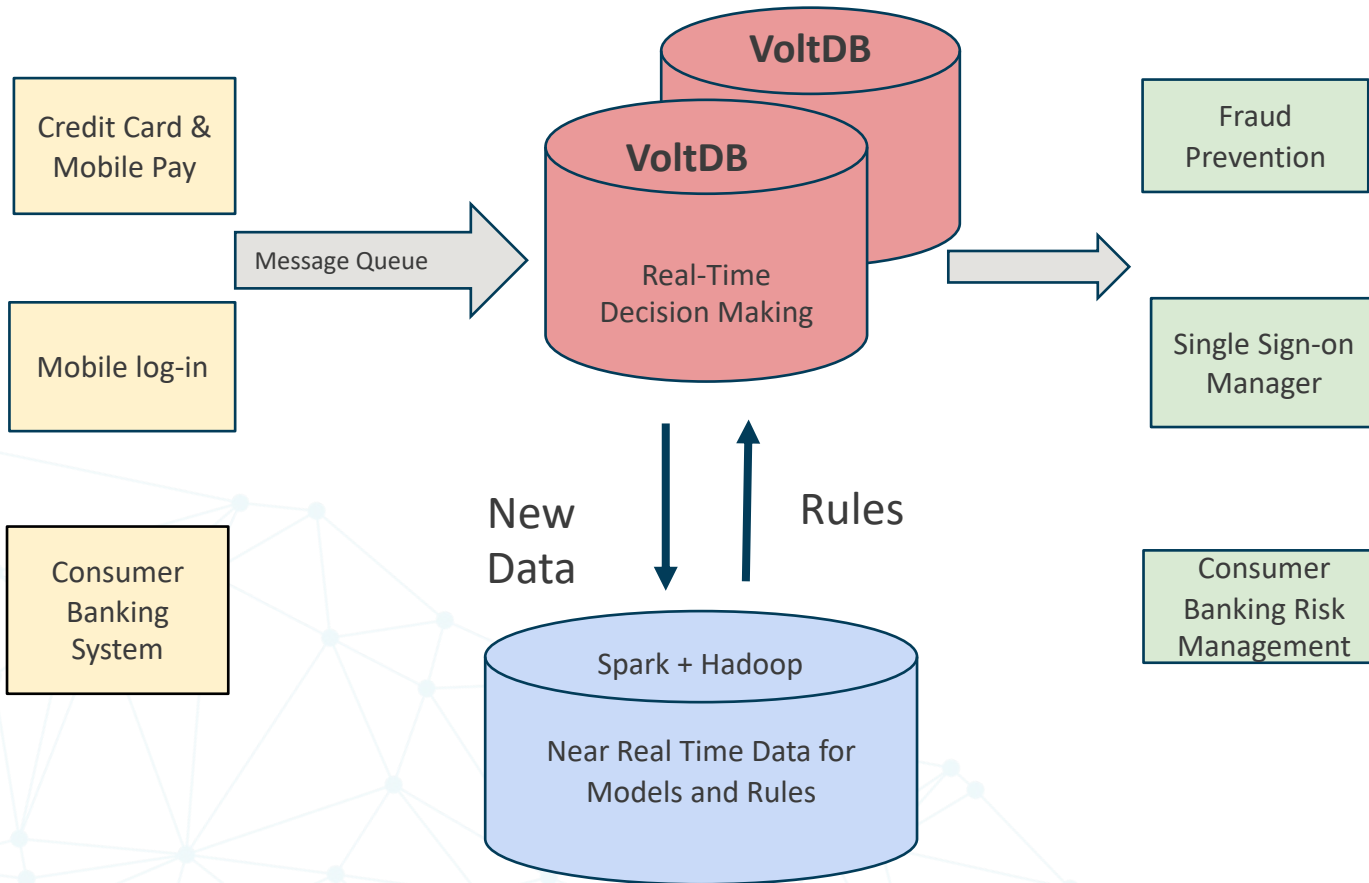




# VoltDB & Machine Learning



# HUAWEI



## Application/Use Case

- Fraud Prevention
- Single sign-in of all Huawei phones
- Consumer banking risk management

## Why VoltDB?

- > 50% reduction in fraud cases
- > \$15M/year saved from fraud loss
- 10k complex Transactions Per Second
- 99.99% transactions finish < 50ms
- 10x better performance than traditional fraud detection

# VoltDB & Machine Learning

- VoltDB has a C++ core with a Java layer on top for running stored procedures
- VoltDB implements High Availability by running the same code in two places at once.
- Any Java class can be used in a stored procedure call provided:
  - It's deterministic (all copies of the code have to act the same way...)
  - It doesn't access network resources (which would make it non-deterministic)
- Examples: H2O.AI and (J)PMML

# ML Example – User Defined Function in H2O

```
public class AirlineDemoUDF {  
  
    private static String modelClassName = "gbm_pojo_test";  
  
    public String ademo(String cRSDepTime, String year, String month, String dayOfMonth, String dayOfWeek,  
        String uniqueCarrier, String origin, String dest) {  
  
        try {  
  
            hex.genmodel.GenModel rawModel;  
            rawModel = (hex.genmodel.GenModel) Class.forName(modelClassName).newInstance();  
            EasyPredictModelWrapper model = new EasyPredictModelWrapper(rawModel);  
  
            RowData row = new RowData();  
            row.put("Year", year);  
            row.put("Month", month);  
            row.put("DayofMonth", dayOfMonth);  
            row.put("DayOfWeek", dayOfWeek);  
            row.put("CRSDepTime", cRSDepTime);  
            row.put("UniqueCarrier", uniqueCarrier);  
            row.put("Origin", origin);  
            row.put("Dest", dest);  
            BinomialModelPrediction p = model.predictBinomial(row);  
  
            return (p.label);  
  
        } catch (Exception e) {  
  
            System.err.println(e.getMessage());  
            return null;  
  
        }  
  
    }  
}
```

```
CREATE FUNCTION ademo FROM METHOD h2o.AirlineDemoUDF.ademo;  
  
CREATE PROCEDURE flight_hist  
PARTITION ON TABLE flights COLUMN f_FlightNum AS  
SELECT f_cRSDepTime, f_year, f_month, f_dayOfMonth,  
f_dayOfWeek, f_uniqueCarrier, f_origin, f_dest  
,ademo(f_cRSDepTime, f_year, f_month, f_dayOfMonth,  
f_dayOfWeek, f_uniqueCarrier, f_origin, f_dest ) ademo  
from flights  
where f_FlightNum = ?  
order by f_year, f_month, f_dayOfMonth,f_cRSDepTime;
```

# ML Example – Calling JPMML from a Procedure

```
public VoltTable[] runModel(String pmmlFileName, VoltTable inputParams) throws Exception {
    Evaluator evaluator = pmmlEvaluators.get(pmmlFileName);

    if (evaluator == null) {
        throw new Exception("Model " + pmmlFileName + " not found");
    }

    List<InputField> inputFields = evaluator.getInputFields();
    Map<FieldName, FieldValue> arguments = new LinkedHashMap<FieldName, FieldValue>();

    // Sanity check input params

    if (inputParams == null) {
        throw new Exception("VoltTable inputParams can't be null");
    }

    if (inputParams.getRowCount() != 1) {
        throw new Exception("VoltTable inputParams must have one row");
    }

    if (inputParams.getColumnCount() != inputFields.size()) {
        throw new Exception("VoltTable inputParams must match length of inputFields. inputParams
            + inputParams.getColumnCount() + " columns, expect " + inputFields.size());
    }

    inputParams.advanceRow();
    for (InputField inputField : inputFields) {
        mapVoltParamToPmmlParam(inputParams, arguments, inputField);
    }

    Map<FieldName, ?> result = evaluator.evaluate(arguments);

    // Processing results
    // Retrieving the values of target fields (ie. primary results):
    List<TargetField> targetFields = evaluator.getTargetFields();
    VoltTable resultTable = mapPmmlTargetFieldsToVoltTable(result, targetFields);

    // other fields
    List<OutputField> outputFields = evaluator.getOutputFields();
    VoltTable otherTable = mapPmmlOutputFieldsToVoltTable(result, outputFields);

    VoltTable[] outputParams = { resultTable, otherTable };

    return outputParams;
}
```

```
public class GolfDemo extends VoltProcedure {

    public VoltTable[] run(double temperature, double humidity,
        String windy, String outlook) throws VoltAbortException {

        VoltTable[] pmmlOut;

        try {

            JPMMLImpl i = JPMMLImpl.getInstance();
            VoltDBJPMMLWrangler w = i.getPool().borrowObject();
            final String modelName = "tree.model";
            VoltTable paramtable = w.getEmptyTable(modelName);
            paramtable.addRow(temperature, humidity, windy, outlook);
            pmmlOut = w.runModel(modelName, paramtable);

        } catch (Exception e) {

            System.err.println(e.getMessage());
            throw new VoltAbortException(e);

        }

        voltExecuteSQL(true);
        return pmmlOut;
    }
}
```



For more information:

[www.voltodb.com](http://www.voltodb.com)

[drolfe@voltodb.com](mailto:drolfe@voltodb.com)