



VoltDB

Intelligent Ingestion

David Rolfe

Director of Solution Architecture, EMEA


17-Jul-18

VOLTDB

In 1984 the world was a very different place...

- Ronald Reagan was president
- For only US\$3700 (about US\$8700 now) you could buy an IBM PC with 256KB of RAM and a 1MB hard drive.
- Multi CPU computers were an exotic rarity.
- There was almost no computer to computer connectivity. Computers interacted with humans. Slowly.
- 1984 is roughly when the architecture of the major RDBMS products was designed.

The IBM Personal Computer



Name
The IBM Personal Computer

Manufacturer
International Business Machines Corporation
Entry Systems Division
POB 1328
Boca Raton, FL 33432
(800) 447-4700

Size
5/8 by 20 by 16 inches; 25 pounds

Components
Processor: Intel 8088, 4.77-MHz; socket for addition of the 8087 math coprocessor
Memory: 40K bytes ROM, 256K bytes RAM standard; 640K bytes RAM maximum with memory-expansion card
Keyboard: 83-key layout with 10 function keys and numeric/cursor keypad; detachable with 6-foot coil cable; adjustable typing angle
Mass Storage: 360K-byte double-sided 5 1/4-inch floppy-disk drive
Expansion: Five expansion slots (one used by floppy-disk drive controller)

Software
Diagnostics: Microsoft cassette BASIC interpreter in ROM

Optional Hardware

64/256K-byte Memory Expansion Card with 64K bytes	\$265
64K-byte Memory Module	\$100
Monochrome Display	\$275
Monochrome Display and Printer Adapter	\$250
Color Monitor	\$680
Color/Graphics Monitor Adapter	\$244
Disk drive (5 1/4-inch floppy disk)	\$425
Asynchronous Communications Adapter	\$100
8087 Math Coprocessor	\$230
Graphics Printer	\$449
Personal Computer Cluster Adapter	\$400
Personal Computer Cluster cable kit	\$110
Fixed Disk Drive Adapter	\$590
10-megabyte Fixed Disk	\$1395
PC Expansion Unit	\$2880

Optional Software

Personal Computer DOS 1.1 (with Advanced Disk BASIC)	\$40
Personal Computer DOS 2.1	\$65
Personal Computer Cluster Program	\$92

Documentation
Guide to Operations, BASIC Reference Manual

Prices

256K bytes of RAM and one floppy-disk drive	\$1995
256K bytes of RAM and two floppy-disk drives	\$2420

IBM has established itself as the nation's major micro-computer producer in the three years since it introduced the IBM Personal Computer in August, 1981. Drawing on a reputation for service and reliability with its larger computer systems, IBM has been able to quickly woo corporate America and even entice home users. Catering to over a million IBM PC users, software publishers have created for the computer one of the largest bases of microcomputer programs, thereby contributing further to the PC's proliferation. The 16-bit 8088 processor makes it possible for users to write larger, more complex programs that offer more

Then we had the era of the “One Big Database”...

- Organizations realized that data was *valuable*.
- Very bad practice for Use Cases to dictate data structures
 - Optimization regarded as a last resort. And a sign of failure.
- The database was a repository of corporate data.
- There was one centrally planned database schema that was ‘correct’.
- Everyone was supposed to use it. Or else.
- *With hindsight it was all a bit “Soviet”...*



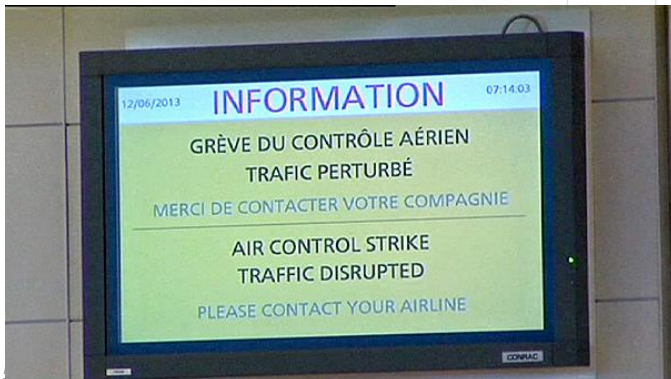
It may have been "Soviet" but...

- Databases were a new and radical concept that solved a real problem.
- They were inspired by the industrial scale chaos caused by every application 'owning' its own data
 - Some of that chaos has returned
- Regarding data as an asset that is more important than a single Use Case is 'common sense' now.
- The "One Big Database" was killed by:
 - Office politics/organizational complexity
 - Operational and design complexity.
 - Lack of suitably miraculous products.
 - The PC and virtualization.
- We now have gone to the other extreme – microservices and KV stores

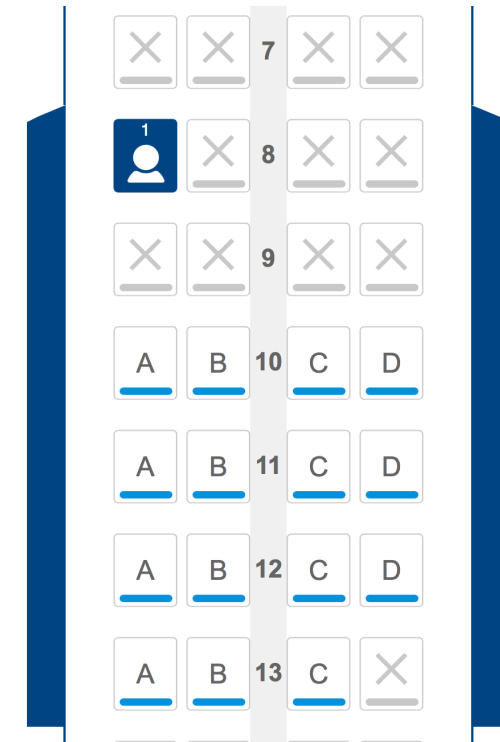
Modern applications are incredibly demanding

Goal: Predict flight delays.

from LON London		from NYC New York								
Flights				Fri 4	Sat 5	Sun 6	Mon 7	Tue 8	Wed 9	Thu 10
BA0117	operated by BA									
<u>LHR</u>	●————●	<u>JFK</u>		✈	✈	✈	✈	✈	✈	✈
08:30	0 stops	11:10		INFO	INFO	INFO	INFO	INFO	INFO	INFO
BA0175	operated by BA									
<u>LHR</u>	●————●	<u>JFK</u>		✈	✈	✈	✈	✈	✈	✈
09:35	0 stops	12:25		INFO	INFO	INFO	INFO	INFO	INFO	INFO
BA0001	operated by BA									
<u>LCY</u>	●—○—●	<u>JFK</u>					✈	✈	✈	✈
09:40	1 stop	14:05					INFO	INFO	INFO	INFO



"The Late Arrival Of The Incoming Aircraft"



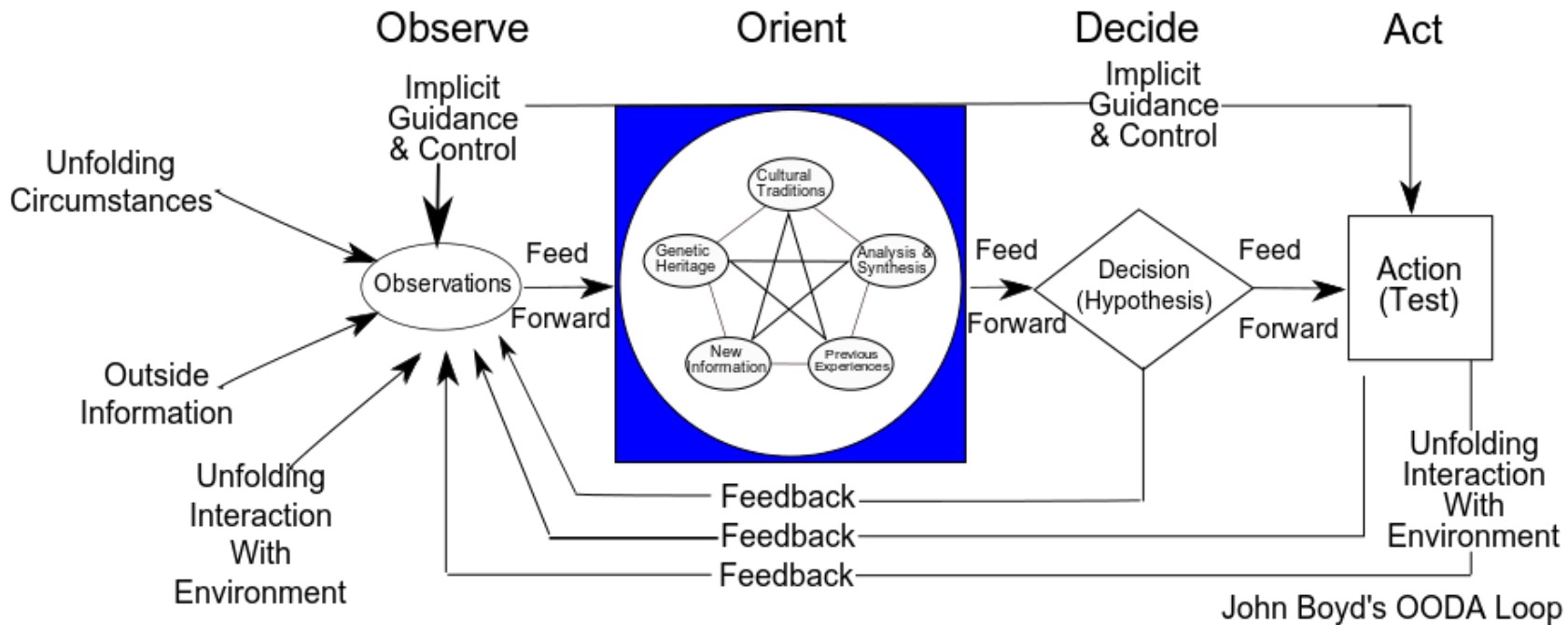
Raw TAF

KJFK 070809Z 0708/0812 36004KT P6SM SCT025 BKN040
 FM071400 04009KT P6SM SCT035 BKN050
 FM071800 15010G15KT P6SM SCT035 BKN050
 FM080100 09009KT P6SM SCT030 BKN100
 FM080900 05005KT P6SM SCT020 SCT100

Raw METAR

KJFK 070951Z 35006KT 10SM FEW060 BKN250 13/11 A3000 RMK AO2 SLP159 T01280106
 KJFK 070851Z 35005KT 10SM FEW060 BKN250 12/11 A2998 RMK AO2 SLP152 T01220106 53013
 KJFK 070751Z 36004KT 10SM FEW055 BKN250 13/11 A2996 RMK AO2 SLP146 T01330106
 KJFK 070651Z 36008KT 10SM SCT024 BKN055 14/11 A2996 RMK AO2 SLP144 T01440111

The OODA Loop, and why it's important...



John "Forty Second" Boyd

1. In order to 'win' your application's loop needs to run faster than 'reality'
2. 'Winning' against a web page means a loop of under 7 seconds.
3. 'Winning' in an IoT context means a loop of around 7 milliseconds.
4. 'Winning' now involves complex decision making at millisecond timescales.

Complexity, Latency and Volumes

Year	Internet Bandwidth (GB)	Required Response Time)	Application Complexity
1985	33	2 day batch turnaround	Row level locking being invented. Transactions of any sort rare.
1995	150,500	2 minute batch	Row level locking of 5-10 rows; web servers using databases.
2000	75,250,000	8000ms – Web Page Advertising	Web advertising: Cookie -> User -> Demographic -> Ad
2010	19,974,008,812	100ms - Video Game Analytics / Sophisticated web advertising	Decisions need to consider dozens/hundreds of elements
2015-2018	>42,423,169,029	10ms - IoT / Many Devices	Decisions may need to consider thousands of elements

Between 1984 and 2015 a lot changed...

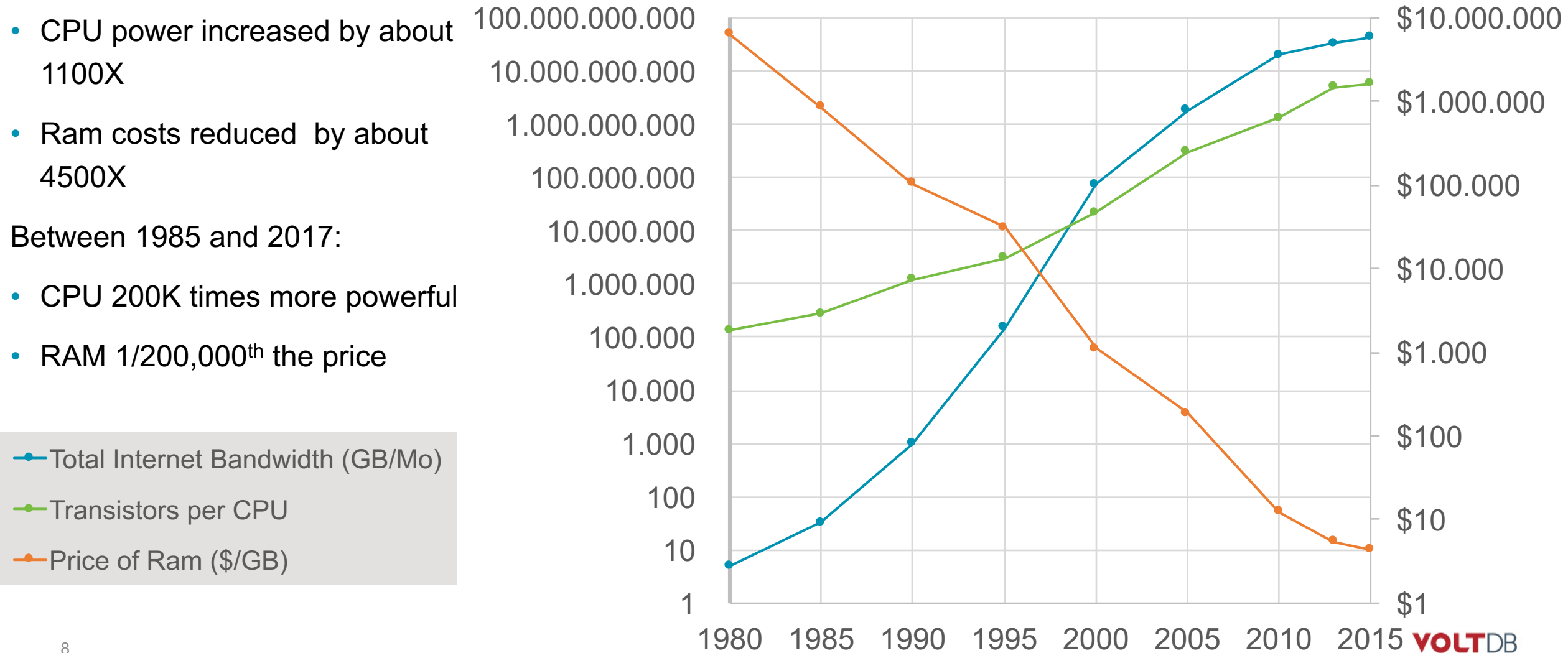
Between 1985 and 2005:

- CPU power increased by about 1100X
- Ram costs reduced by about 4500X

Between 1985 and 2017:

- CPU 200K times more powerful
- RAM 1/200,000th the price

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

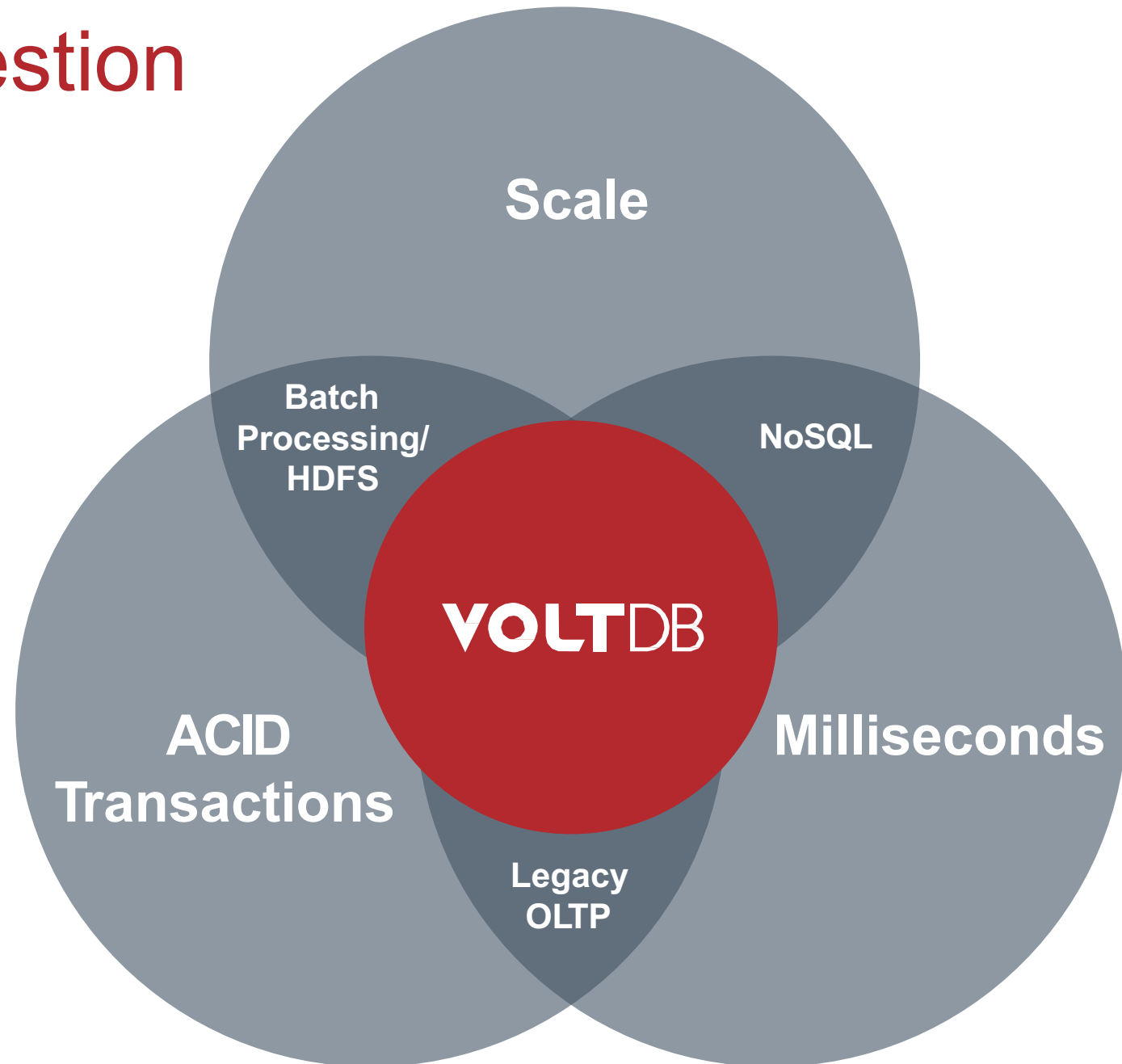


We now face real ‘pain’...

- Timescales (“OODA loop”), volumes and complexity all pose challenges.
- We need to take complex decisions, fast:
 - Complex Decisions usually involve many different data sources.
 - Key Value solutions imply a large number of network trips...
 - ...and ACID becomes an issue if your data changes while you are reading it
 - Legacy RDBMS implies a lack of scale.
 - Decisions involving shared resources usually implies locking or retries
 - Many decisions involve aggregate values (“HTAP/Translytics”)
 - Many decisions involve telling a third party something
- What we need is the ability to do all of this *as the data arrives*.

Intelligent Ingestion

1. Complex Decisions
2. Multiple Data Sources
3. ACID
4. Millisecond Timing
5. Massive Scale



A hand holding a pen writing on a surface with a binary code background. The background is a gradient of green and blue, with a pattern of binary code (0s and 1s) overlaid. The hand is in the foreground, holding a pen and writing on a surface. The overall image has a digital, data-driven aesthetic.

VOLTDB

Why use VoltDB for “Intelligent Ingestion”?

Intelligent Ingestion

VoltDB is optimized for Scale, ACID and Latency...



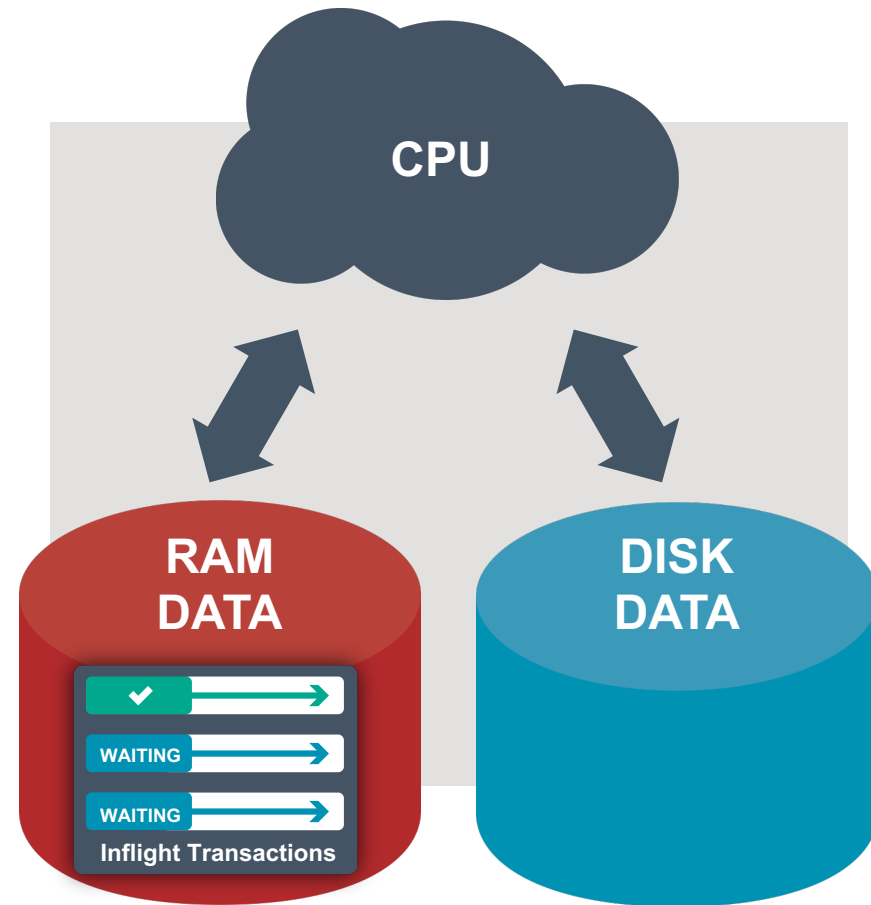
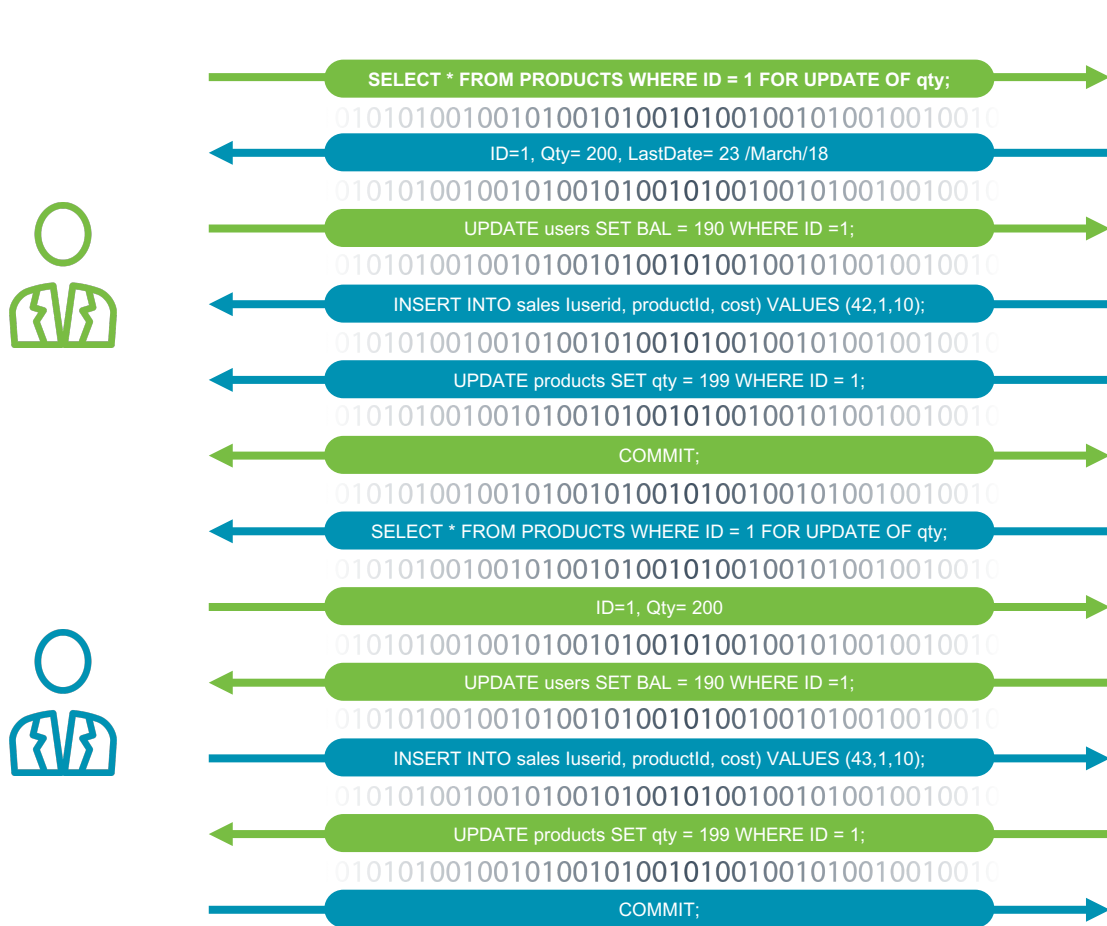
An **Ariel Atom**. Very good for going round racetracks very quickly in nice weather. Not so good for school runs, driving tests, shopping, off road activities, as an ambulance, polar exploration, amphibious assaults, carrying cargo....

VoltDB is optimized to solve one class of problems better than anything else that exists.

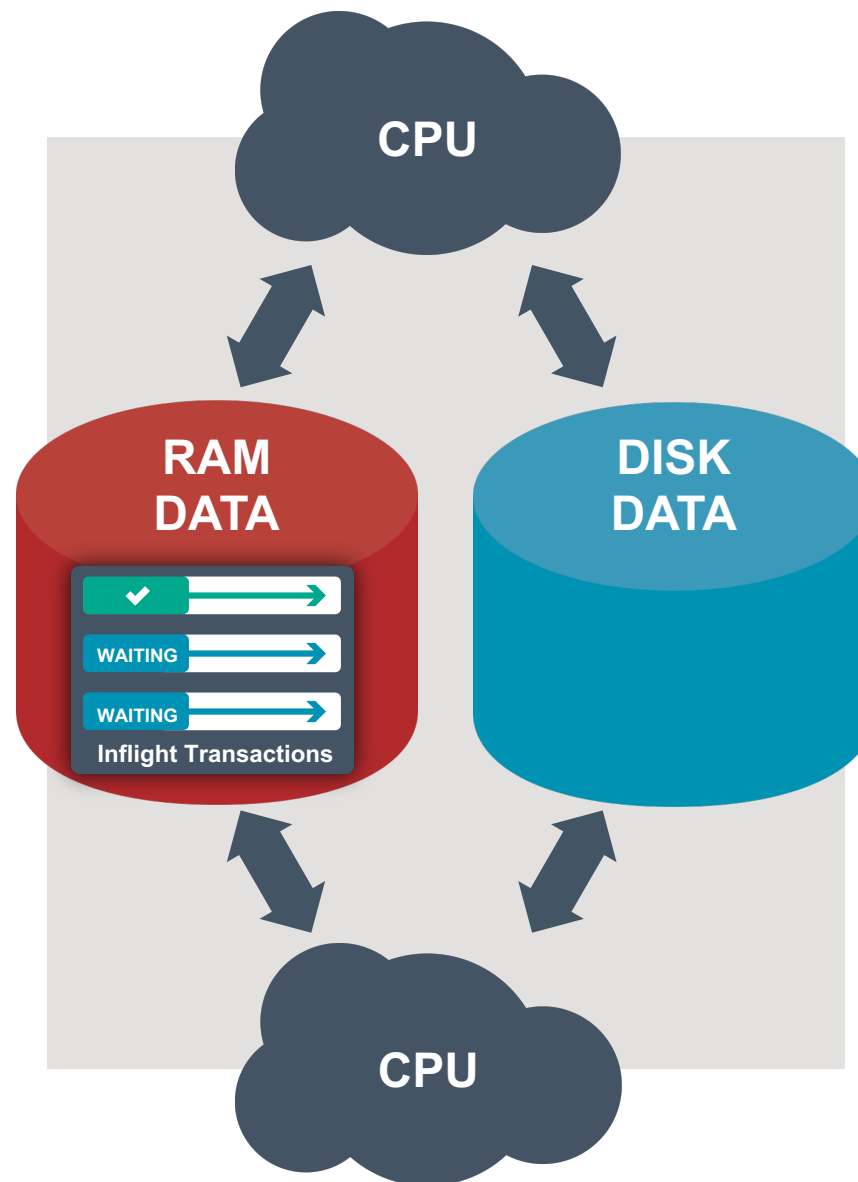
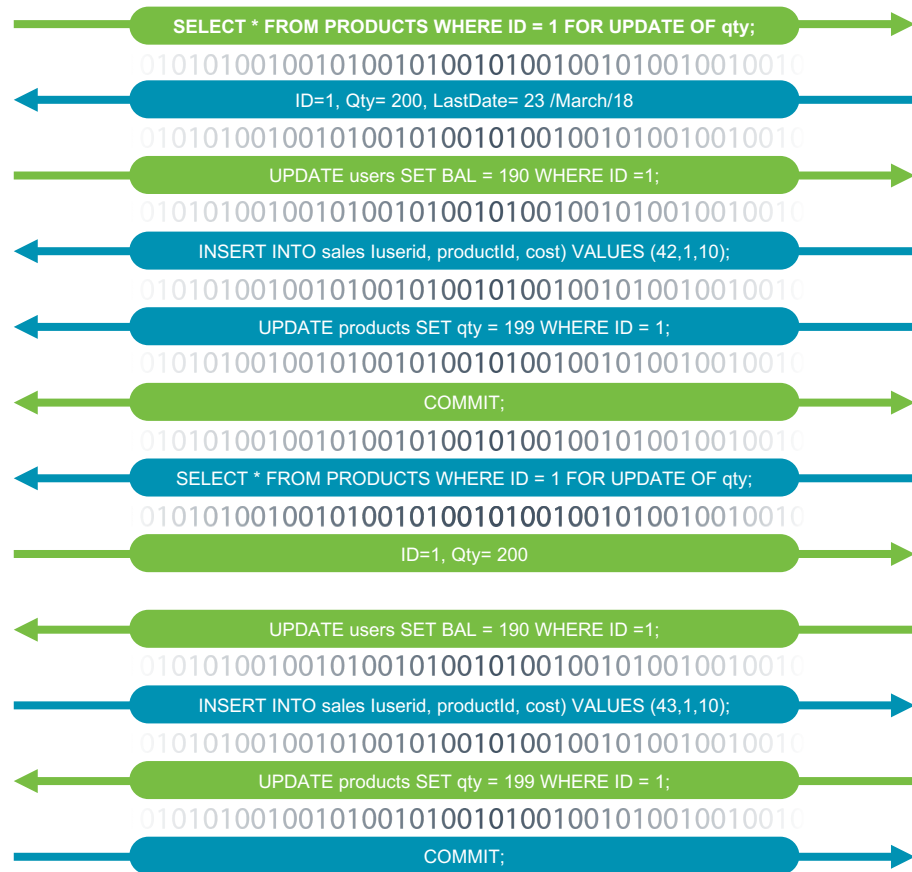
It is not a general purpose RDBMS, nor a replacement for one.

It's used to complement your existing stack rather than replacing it.

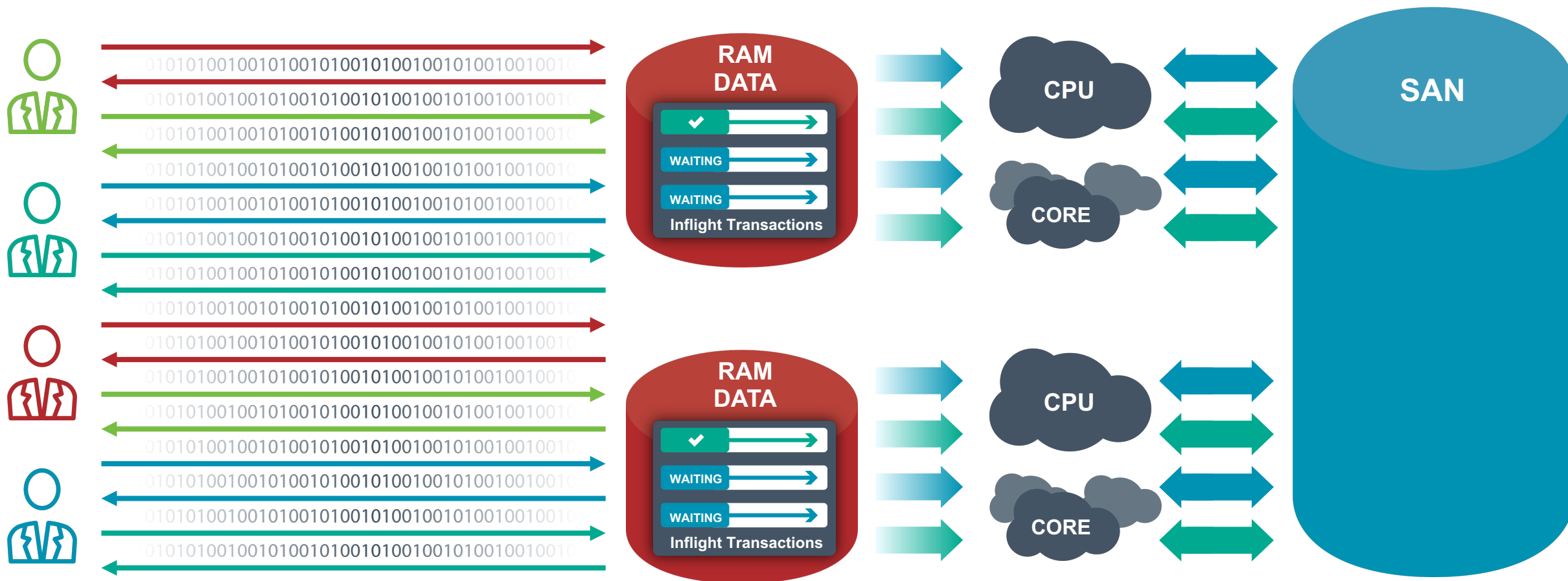
RDBMS - How We Thought an RDBMS Worked



RDBMS - What Actually Happens – Part 1...



RDBMS - What Actually Happens – Part 2



VoltDB was designed to solve this problem

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, harizopoulos}@csail.mit.edu

Nabil Hachem
 AvantiGarde Consulting, LLC
 nhachem@avgc.com

Pat Hell
 Microsoft Corp
 phell@microsoft.com

ABSTRACT

It is a truism that the current DBMS code lines will be all but obsolete in 10 to 20 years. This paper presents reasons and experimental evidence that show that the major DBMS vendors can be expected to fail to deliver systems that meet the needs of the data warehouse, stream processing, and scientific database markets.

Assessing that specialized engines designed for these markets over time, the current relational DBMS code lines will be all but obsolete for business data processing (OLTP) market and related markets where there are few alternatives in sight. In this paper we show that current DBMSs can be forced to reach the limits of what can be done by re-engineering a new ACFT prototype, H-Store, which we have built at MIT, to a parallel DBMS or to the social and transactional hardware, TPC-C.

We conclude that the current DBMS code lines, while attempting to do so, will not do so. It is time to start re-engineering. Hence, today we are 25 years old legacy code lines that should be retired in favor of a collection of "best-of-breed" specialized engines. The DBMS vendors need to re-evaluate their strategy and should start with a clean sheet of paper and design systems for tomorrow's requirements, not continue to patch code lines and architectures designed for yesterday's needs.

1. INTRODUCTION

The biggest financial DBMSs all claim that their roots in System R date from the 1970s. For example, DB2 is a direct descendent of System R, being just the RDB portion of System R with a front-end screen. Similarly, SQL Server is a direct descendent of Subrel System S, which borrowed heavily from System R. Lastly, the first release of Oracle implemented the user interface from System R.

Permissions to copy without fee are granted by the copyright owner provided that the copies are not made for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale.

Copyright © 2008 ACM 978-1-60558-088-9/08/0006...\$5.00

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 dna@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 (Show) running a subset of TPC-C. Rather than simply profiling Show, we progressively modified it so that after every feature removal or optimization, we had a (faster) 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 new performance. We also show that there is no single "high pole in the tent" in modern (memory resident) database systems, but that substantial time is spent in logging, locking, B-tree, and buffer management operations.

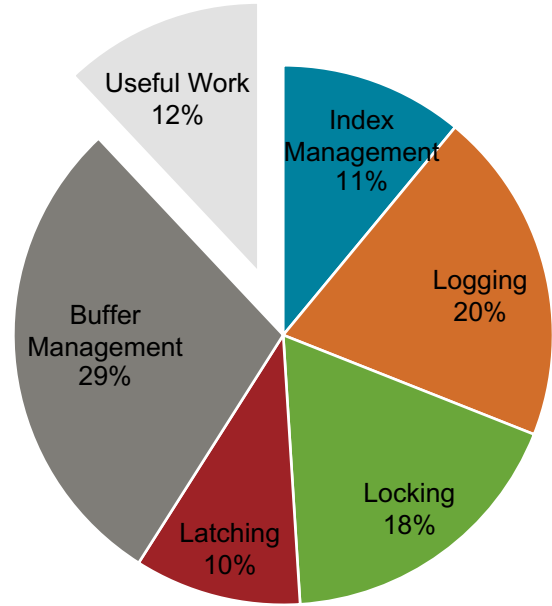
Categories and Subject Descriptors
 H.2.4 [Database Management]: Systems — transaction processing; concurrency.

General Terms
 Measurement, Performance, Experimentation.

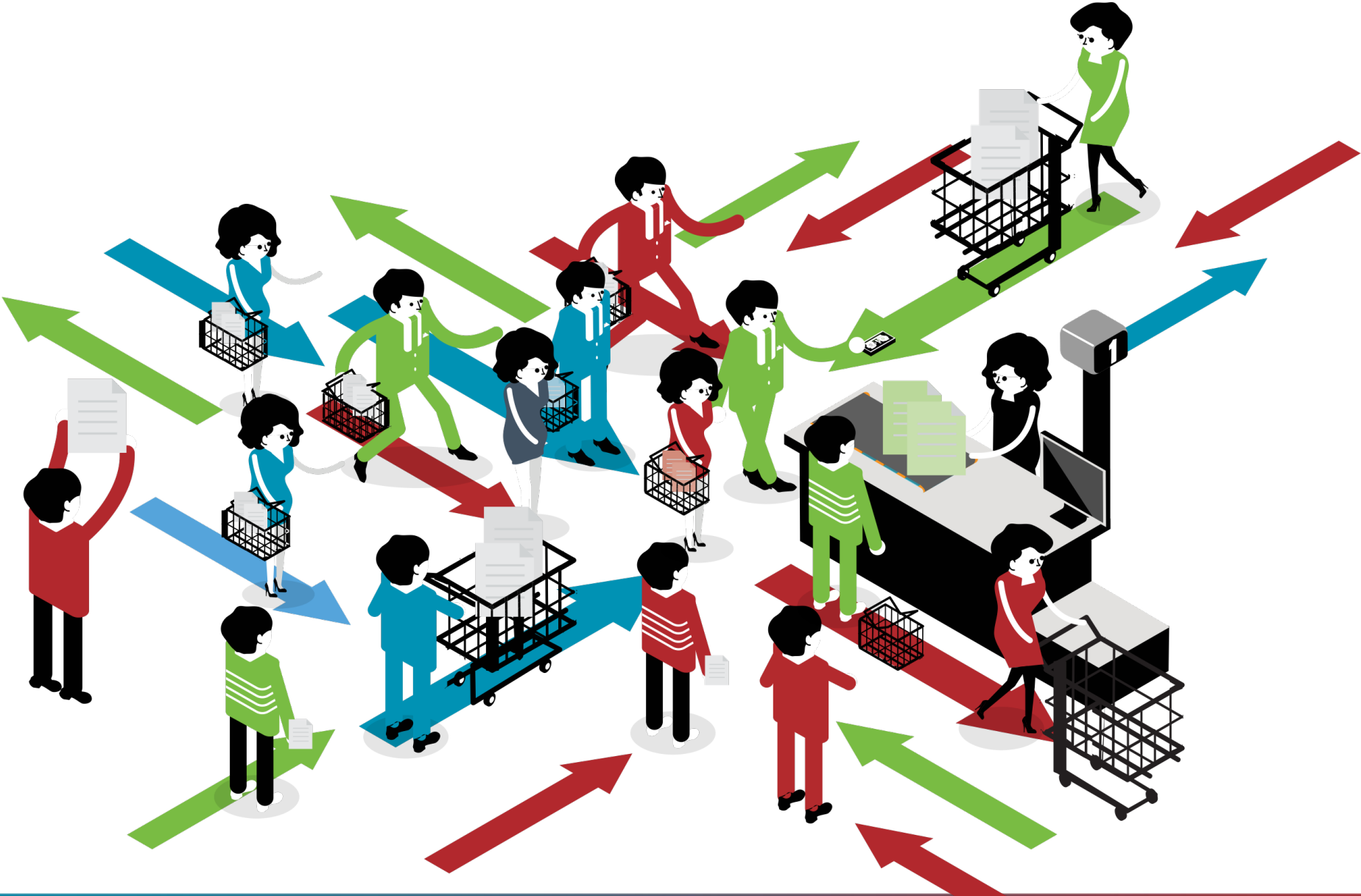
Keywords
 Online Transaction Processing, OLTP, main memory transaction processing, DBMS architecture.

Permissions 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 or distributed for profit or commercial advantage and that copies bear the notice and the full citation on the first page. In copy, otherwise, or to publish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

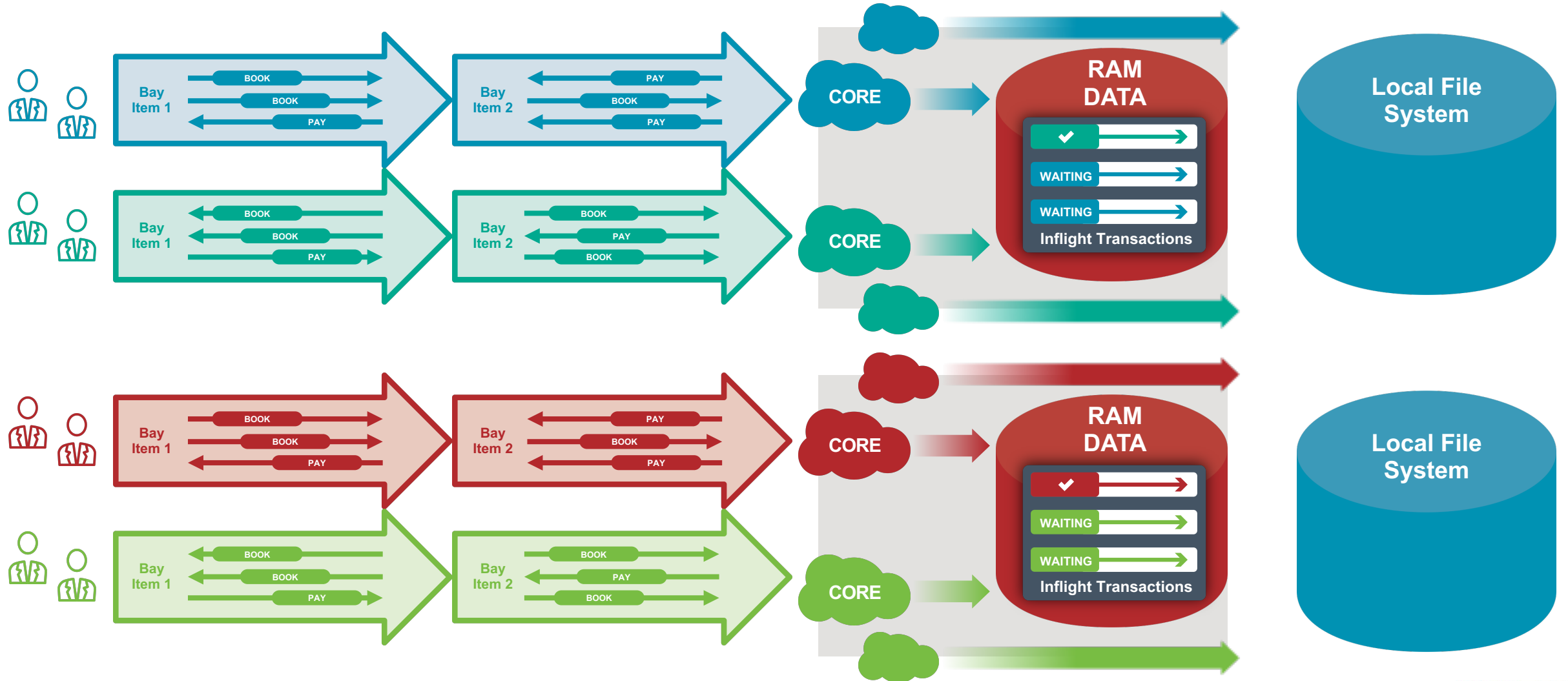
SIGMOD '08, June 9–12, 2008, Vancouver, BC, Canada.
 Copyright 2008 ACM 978-1-60558-088-9/08/0006...\$5.00.



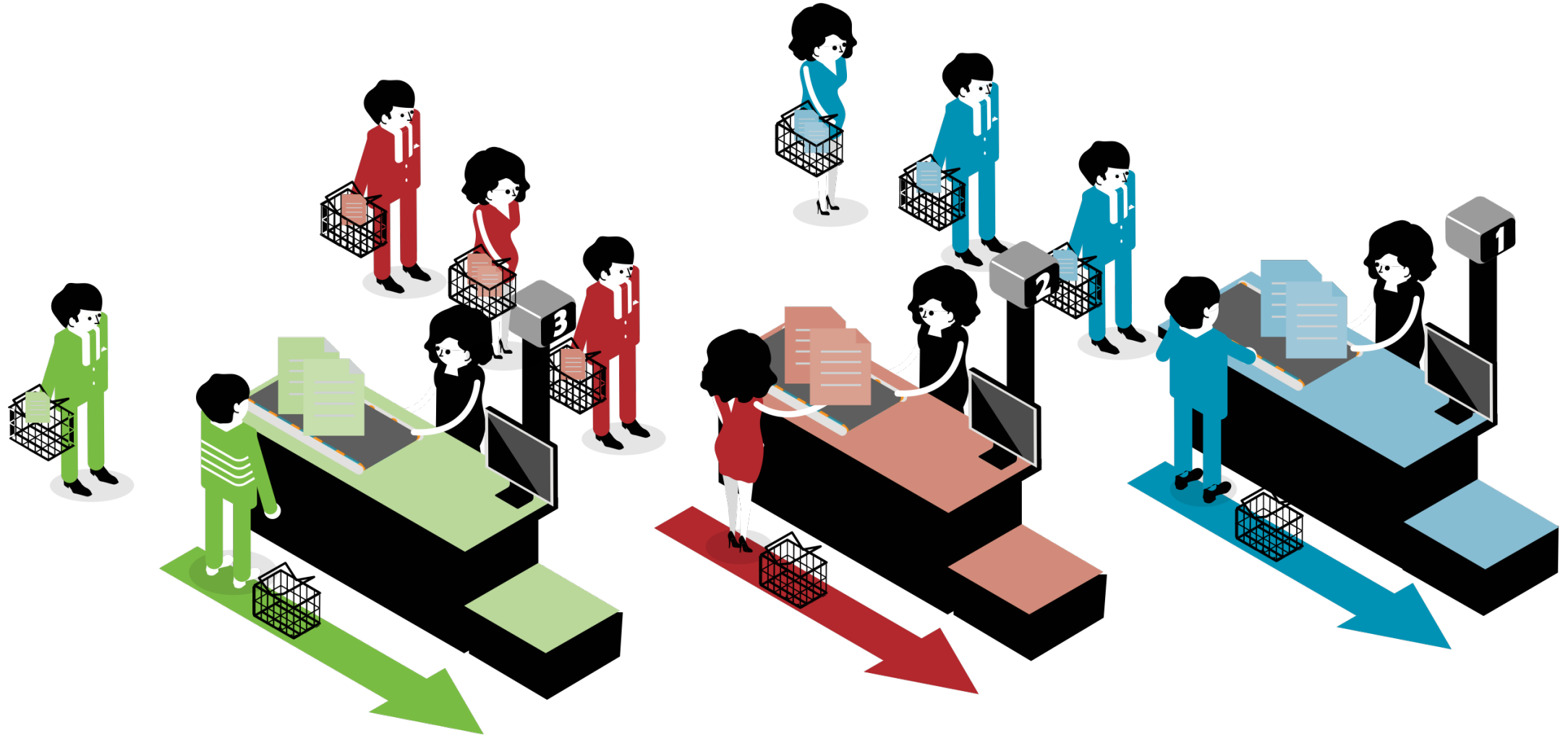
If we tried this in a supermarket...



How VoltDB works



How a supermarket works...



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

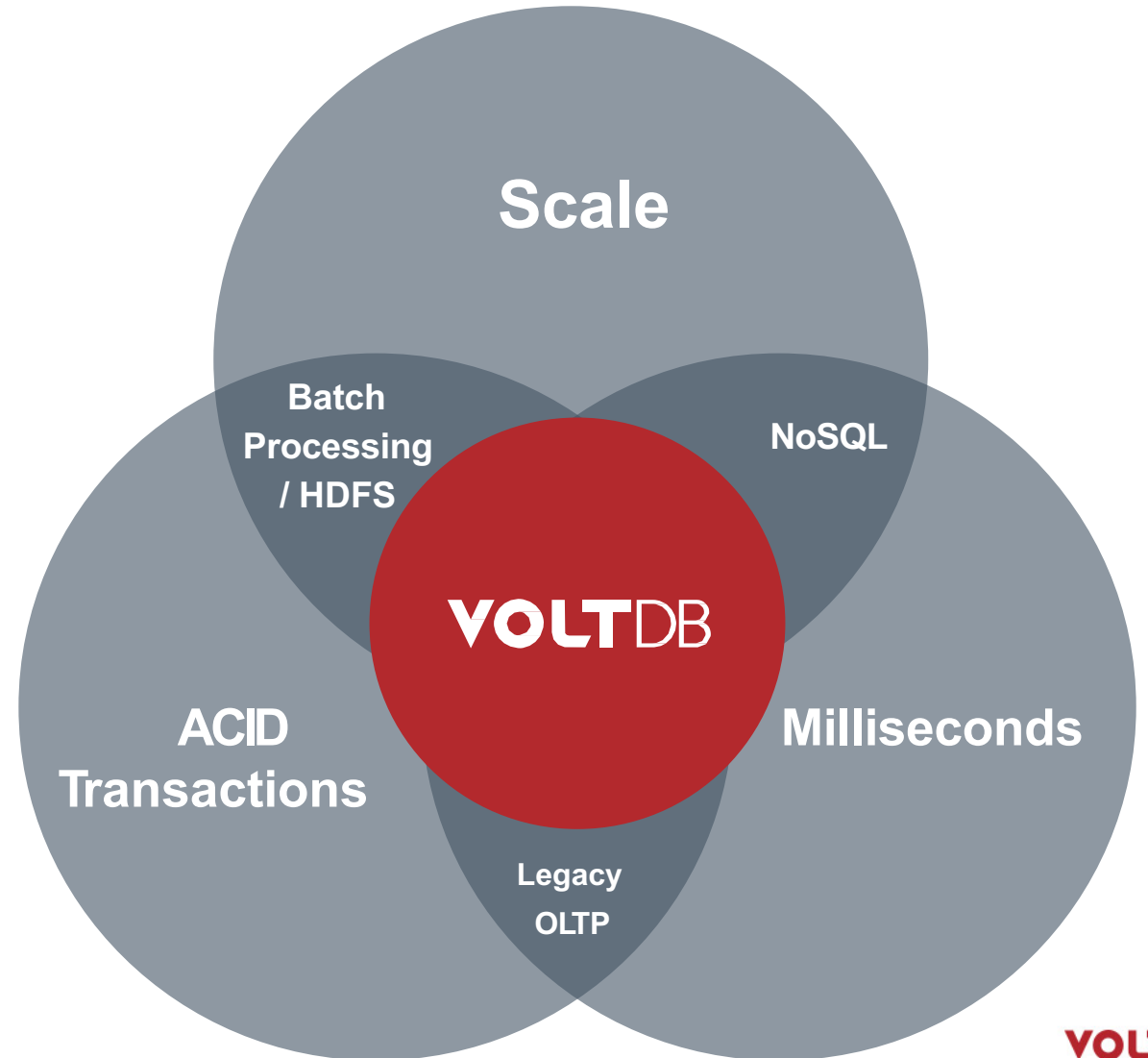
VOLTDDB

VoltDB and Machine Learning

ML: The search for value

ML adds *real* value when:

- Combined with state
 - State can be a composite of multiple data sources
- Is used to inform and influence decisions
- Happens in real time
- Happens at scale



VoltDB + ML

- The Good News

- VoltDB appears to work with any ML engine with a Java runtime
- VoltDB can be made to work with any ML engine with a C++ runtime
 - Requires JNI expertise

- Caveats

- Runtime engine can't have runtime dependencies/speak to another server
- Runtime engine must be deterministic
- Runtime engine must be fast (< 5ms)
 - >5ms undermines utility of using VoltDB
 - Be wary of instantiation costs

VoltDB + ML

- Limitations

- Non Deterministic code
- Other servers
- Not C++ or JVM compatible
 - Not a showstopper, but...
- Slow

- Example: Neural Nets

- But GDPR may make them unusable in EU anyway...

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 h20.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;
```

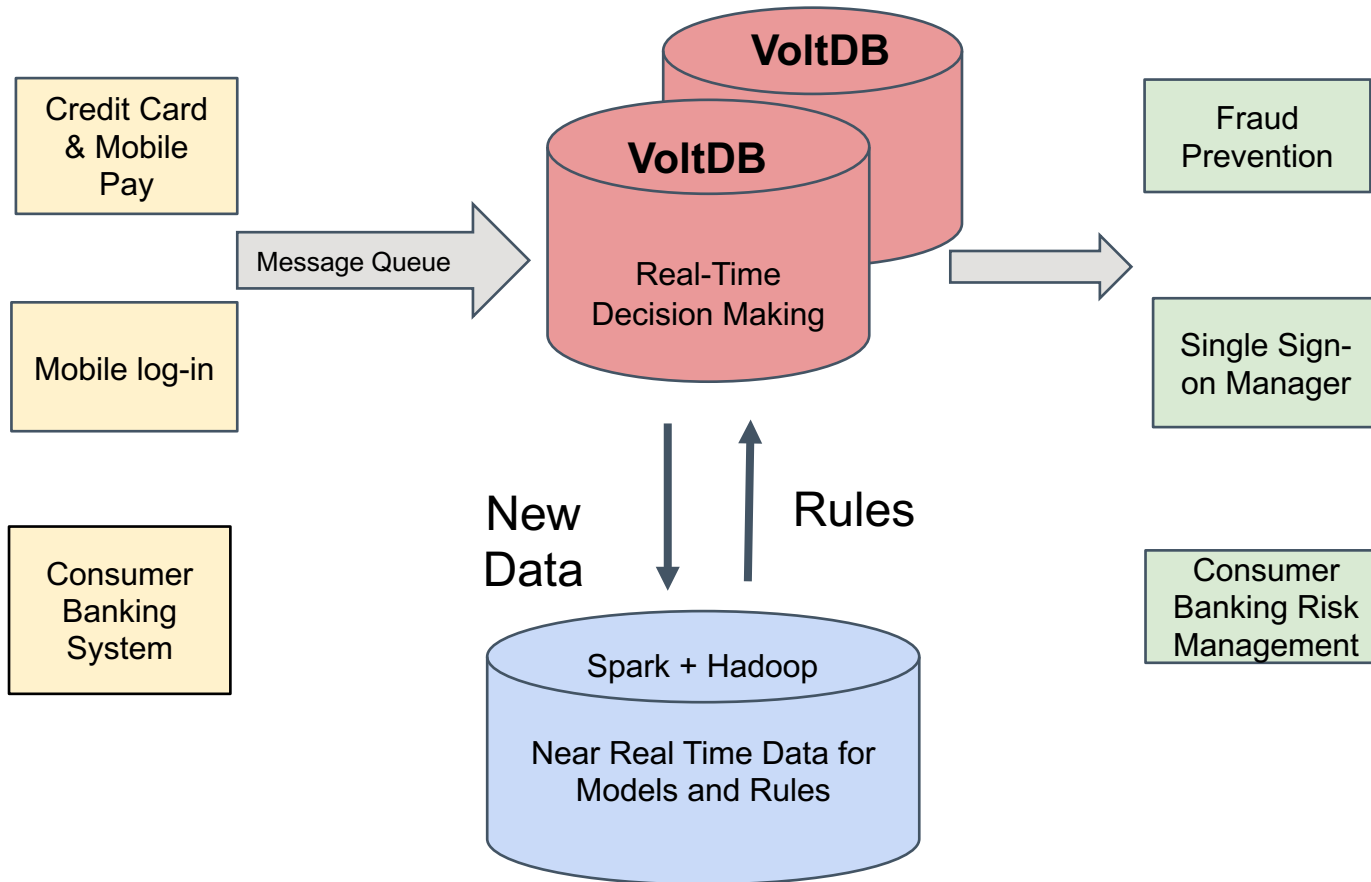
ML Example – Calling JPMML from a Procedure

```
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 (i.e. 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;  
    }  
}
```



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

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.

