

The End of an Architectural Era (It's time for a complete rewrite)

by

Michael Stonebraker

Who We Are

- ◆ **Dan Abadi, Stavros Harizopoulos**
 - ◆ **H-Store implementation**
- ◆ **Nabil Hachem**
 - ◆ **TPC-C benchmarking**
- ◆ **Mike Stonebraker, Sam Madden, Pat Helland**
 - ◆ **Kibitzers**

Outline

- ◆ **The current state of the world**
- ◆ **Why current architecture is “long in the tooth”**
- ◆ **How to beat it by a factor of 50 in every market I can think of**
- ◆ **Implications for the research community**

Current DBMS Gold Standard

- ◆ **Store fields in one record contiguously on disk**
- ◆ **Use B-tree indexing**
- ◆ **Use small (e.g. 4K) disk blocks**
- ◆ **Align fields on byte or word boundaries**
- ◆ **Conventional (row-oriented) query optimizer and executor**

Terminology -- “Row Store”

Record 1

Record 2

Record 3

Record 4

E.g. DB2, Oracle, Sybase, SQLServer, ...

Row Stores

- ◆ **Can insert and delete a record in one physical write**
- ◆ **Good for business data processing (the IMS market of the 1970s)**
- ◆ **And that was what System R and Ingres were gunning for**

Extensions to Row Stores Over the Years

- ◆ Architectural stuff (Shared nothing, shared disk)
- ◆ Object relational stuff (user-defined types and functions)
- ◆ XML stuff
- ◆ Warehouse stuff (materialized views, bit map indexes)
- ◆

At This Point, RDBMS is “long in the tooth”

- ◆ There are at least 4 (non trivial) markets where a row store can be clobbered by a specialized architecture (CIDR 07 paper)
 - ◆ Warehouses (Vertica, SybaseIQ, KX, ...)
 - ◆ Text (Google, Yahoo, ...)
 - ◆ Scientific data (MatLab, ASAP prototype)
 - ◆ Streaming data (StreamBase Coral8, ...)

At This Point, RDBMS is “long in the tooth”

- ◆ Leaving RDBMS with only the OLTP market
- ◆ But they are no good at that either!!!!!!

Alternate OLTP Proposal

- ◆ **First part**
 - ◆ **Main memory**
 - ◆ **Grid orientation**
 - ◆ **Threading**
 - ◆ **Redo Recovery**
- ◆ **Second part**
 - ◆ **Concurrency control**
 - ◆ **Undo**
 - ◆ **2 phase commit**

OLTP Has Changed

- ◆ 1970's: disk
- ◆ Now: main memory

TPC-C is 100 Mbytes per warehouse; 1000 warehouses is a HUGE operation;

i.e. 100 Gbytes;

i.e. main memory

OLTP Has Changed

- ◆ **1970's: terminal operator**
- ◆ **Now: unknown client over the web**

Cannot allow user stalls inside a transaction!!!!

Hence, there are no user stalls or disk stalls!!!!

Result: No Multi-threading!!!

- ◆ Heaviest TPC-C Xact reads/writes 200 records
 - ◆ Less than 1 msec!!
- ◆ Run all commands to completion; single threaded
- ◆ Dramatically simplifies DBMS
 - ◆ No B-tree latch crabbing
 - ◆ No pool of file handles, buffers, threads, ..

Multiple cores can be handled by multiple logical sites per physical site

Grid Computing

- ◆ Obviously cheaper
- ◆ Obvious wave of the foreseeable future (replacing shared disk)
- ◆ Horizontally partition data
 - ◆ Shared nothing query optimizer and executor
- ◆ Add/delete sites on the fly required

High end OLTP has to “scale out” not “scale up”

OLTP Has Changed

- ◆ **1970's: disaster recovery was “tape shipping”**
- ◆ **Now: 7 x 24 x 365 no matter what**

Tandem-style HA over a LAN and/or WAN is now required!!!

Built-in HA

- ◆ **Redundancy (at the table level) in the grid**
- ◆ **If grid has a WAN, then get disaster recovery**
- ◆ **Optimizer chooses which instance of a table to read, writes all instances (transactionally)**

Recovery in a K-safe Environment

- ◆ Restore dead site
- ◆ Query up sites for live data
- ◆ When up to speed, join the grid
- ◆ Stop if you lose $K+1$ sites
- ◆ No redo log!!!!
 - ◆ No slower than log recovery (Lau paper – SIGMOD 06)

Vertica has shown this to be perfectly workable – albeit sometimes outside customer's comfort zone....

Main Sources of Overhead in Main Memory DBMS

- ◆ **Disk I/O (gone)**
- ◆ **Resource control (gone)**
- ◆ **Synchronization (gone)**

- ◆ **Undo log (but in main memory and discard on commit)**
- ◆ **Concurrency control**
- ◆ **2 phase commit (for multi-site updates and copies)**

OLTP Has Changed

- ◆ **1970's: conversational transactions**
- ◆ **Now: stored procedures;**
 - ◆ **Can ask for all of them in advance**

Structure of H-Store

- ◆ **Get all transaction classes in advance**
 - ◆ **Instances differ by run-time parameters**
- ◆ **Construct a physical data base design (manually now; automatically in the future)**
 - ◆ **Table partitioning**
 - ◆ **Table-level replication**
- ◆ **Create a “gamma-style” query plan for each class**

Analyze Transaction Classes for Leverage Points

- ◆ **Whole bunch in the paper**
 - ◆ **Constrained tree applications, Single site transactions, one shots, ...**
- ◆ **Two allow leverage in TPC-C**
 - ◆ **Commutativity (Ants pioneered this)**
 - ◆ **Two-phase**

Two Phase

- ◆ In phase one, Xact can read and abort but not write
- ◆ In phase two, Xact can read and write but not abort

All TPC-C Xacts can be made two phase, with rearrangement of new_order logic

Commutativity

- ◆ **All pairs of Xacts produce the same final data base state**
 - ◆ **With any statement-level ordering at each site**

With this definition and a little trickery (in the paper),
all TPC-C transactions are commutative

Overhead Reduction

- ◆ **Commutativity and two-phase**
 - ◆ **No locking**
 - ◆ **No 2 phase commit**
 - ◆ **No undo log**

Tested configuration also used selective redundancy of read-only objects to improve site locality

TPC-C Performance on a Low-end Machine

- ◆ Elephant

- ◆ 850 TPS (1/2 the land speed record per processor)

- ◆ H-Store

- ◆ 70,416 TPS (1/2 the land speed record with \$2K of hardware)

Factor of 82!!!!!!

Open Research Problems

- ◆ **Teasing apart the factor of 82**
 - ◆ **In process**
- ◆ **Automatic data base designer**
 - ◆ **Create a physical data base design that is as fast as possible**

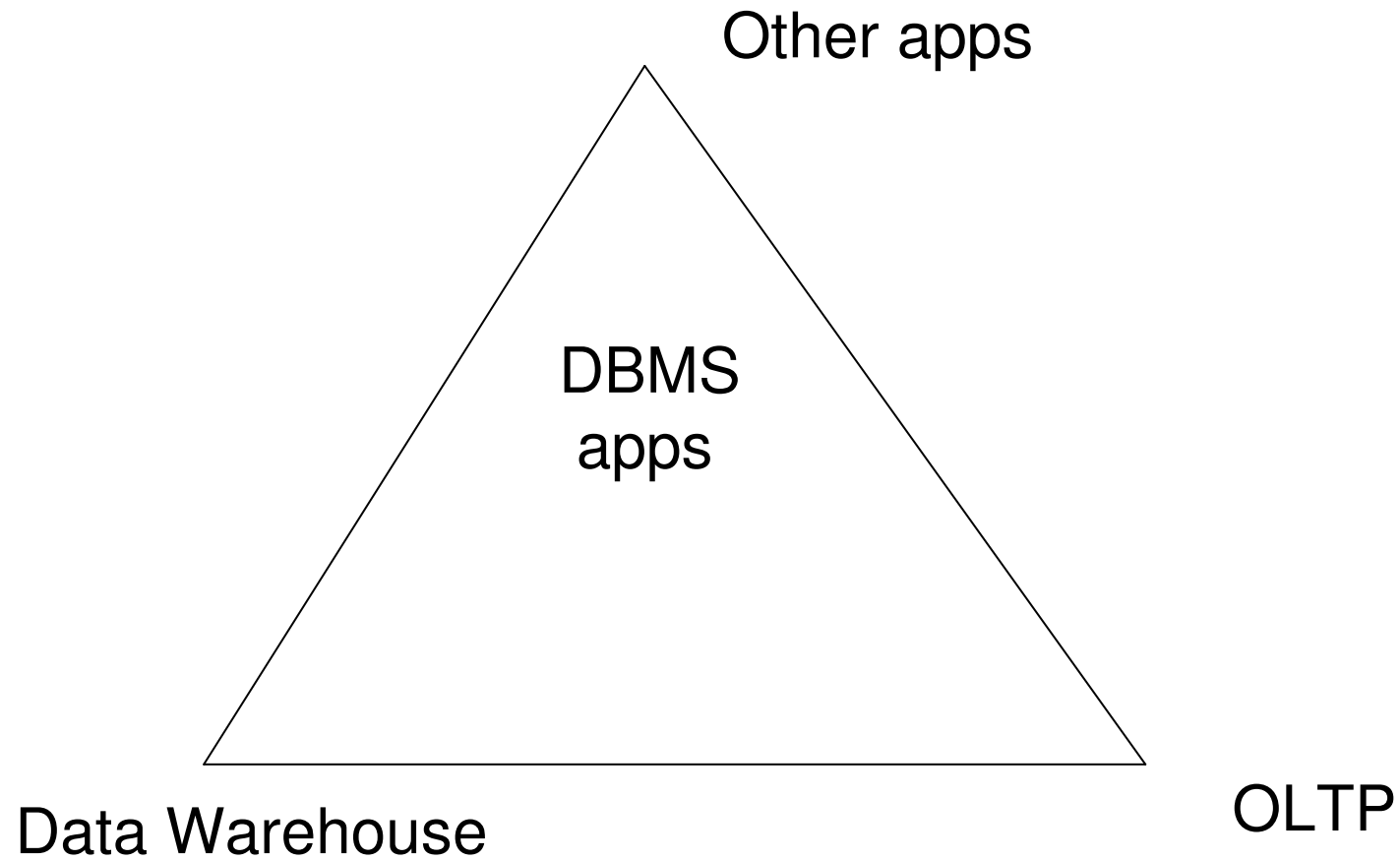
Open Research Problems

- ◆ **Concurrency control**
 - ◆ **Which variation on OCC to use when application is not “well behaved”**
- ◆ **Theory question**
 - ◆ **Characterize carefully the leverage points**

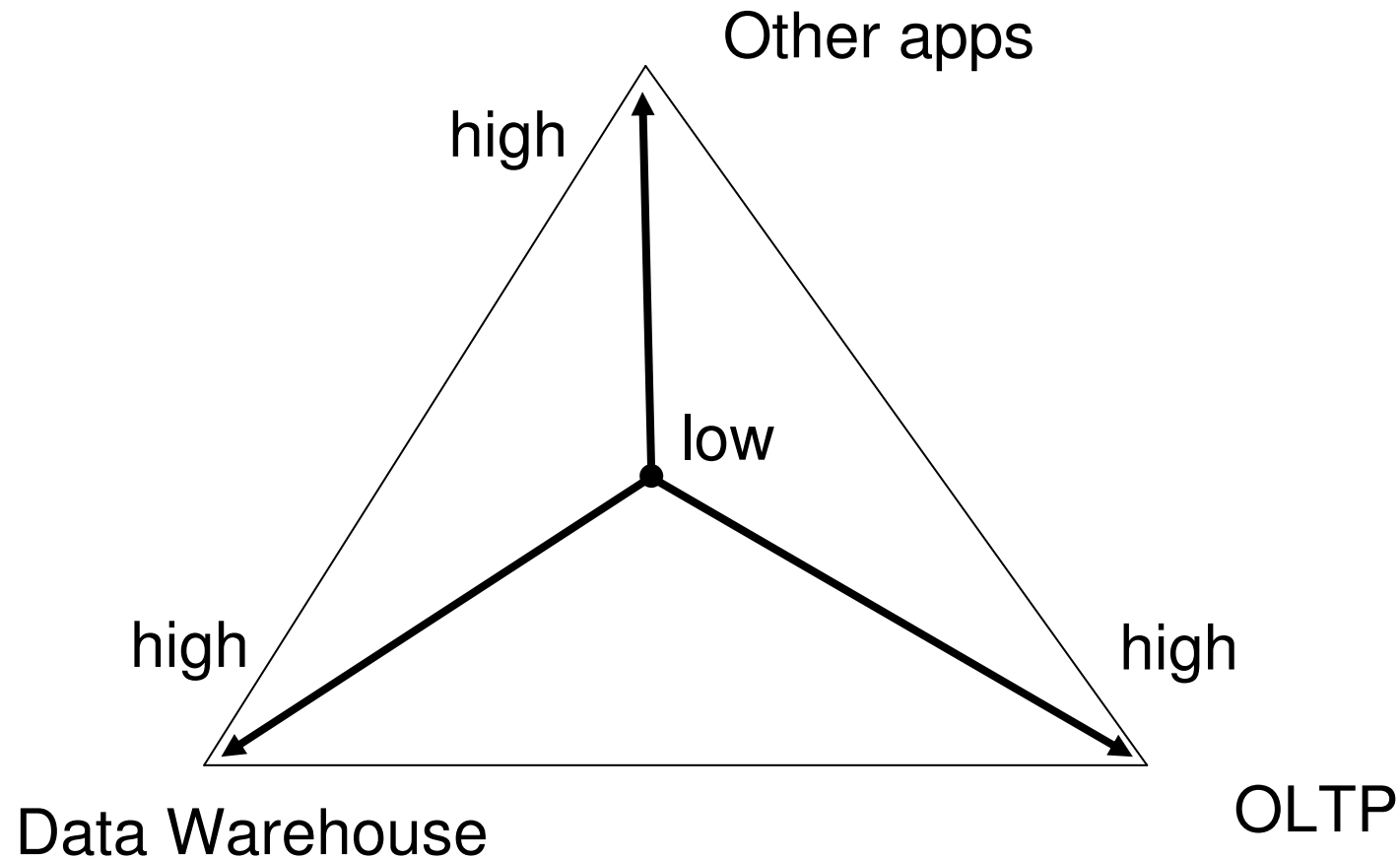
Implications for the Elephants

- ◆ They are selling “one size fits all”
- ◆ Which is 30 year old legacy technology that is good at nothing

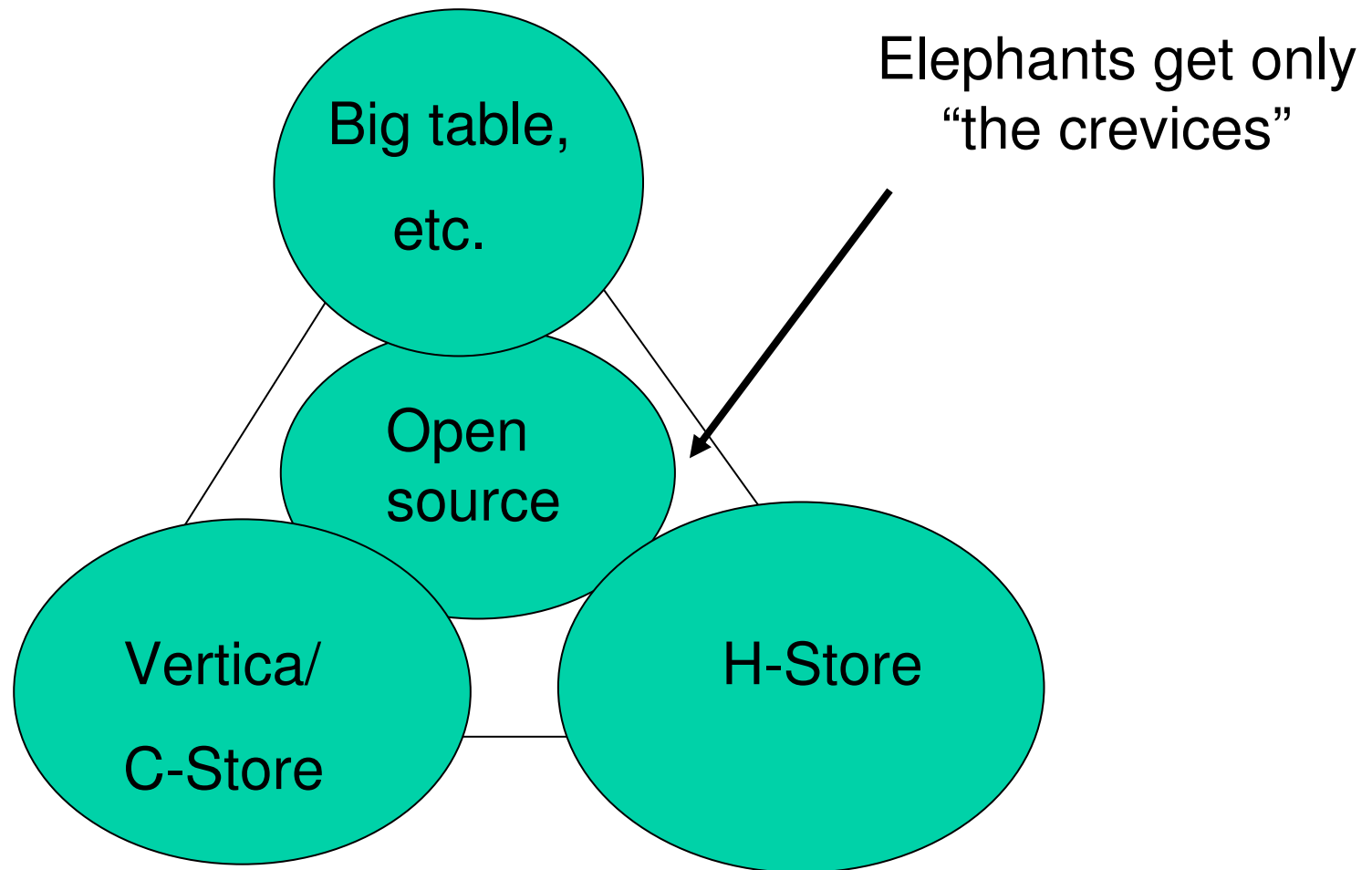
Pictorially:



The DBMS Landscape – Performance Needs



One Size Does Not Fit All -- Pictorially



Other Implications

- ◆ **Data model**
- ◆ **Query language**
- ◆ **Programming style**

Data Model -- Total Heresy....

- ◆ Relational model was the answer for OLTP in 1970s
- ◆ Time to rethink the “hallowed halls”
 - ◆ Warehouses are ER
 - ◆ Semi-structured data is RDF or XML
 - ◆ OLTP usually hierarchical (true for “one site” transactions)
- ◆ One size does not necessarily fit all!!!

Query Language

- ◆ **SQL is a “one-size-fits-all” language**
 - ◆ **OLTP can be a (possibly small) subset (e.g. no aggregates)**
 - ◆ **Warehouses do not require fancy consistency stuff**

Programming Style

- ◆ In the 1970's there were two proposals
 - ◆ Data sublanguage, e.g. SQL Quel, ... with ODBC/JDBC, ...
 - ◆ Extended programming language (Rigel, Pascal R, PL/1 extension)

Data sublanguage is 20x the lines of code

But won in the marketplace

Programming Style -- Today

- ◆ **ODBC/SQL is 20x Ruby on Rails**
- ◆ **High time to embed DBMS stuff cleanly in the PL**

Implications for the Research Community

- ◆ Find a problem area where there might be a factor of 50 and study it
- ◆ Lots of good choices
 - ◆ Web 2.0
 - ◆ Bio (RDF?)
 - ◆ Science in general
 - ◆ Integration of structured and unstructured data (Google meets DBMS)

Implications for the Research Community

- ◆ If you have a good idea -- prototype it
 - ◆ Ok to have a market-specific data model
 - ◆ And query language
- ◆ Could make use of existing systems in novel ways
 - ◆ RDF on a column store (Abadi paper)