FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Datenbanksysteme Spezialisierung

Prof. Dr. Viktor Leis

Professur für Datenbanken und Informationssysteme

About the Course

- database courses generally teach the same textbook database architecture
- in the past 10-15 years many new systems have emerged that are very different from traditional ones
- the goal of the course is to survey the **architecture of modern high-performance relational database systems**
- along the way we will see many techniques (SIMD, synchronization, parallelization) that are useful for performance-critical systems in general

About the Course (2)

- each lecture I will present one or more research papers (typically SIGMOD, VLDB, ICDE, CIDR)
- links to the papers will be posted before the lecture
- you should read (or at least skim) the paper before the course and note some questions
- this should be a very interactive course
- at the end of the semester there will be an oral exam
- course is (also) a good preparation for a Bachelor's, Master's, and PhD thesis in databases

Some Topics

- query execution
  - through compilation
  - through vectorization
  - parallelization
- storage
  - column stores, compression
  - in-memory index structures, synchronization
- (multi-version) concurrency control
- query optimization
- buffer management
- ...

A Relational Model of Data for Large Shared Data Banks

E. F. CODD
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which

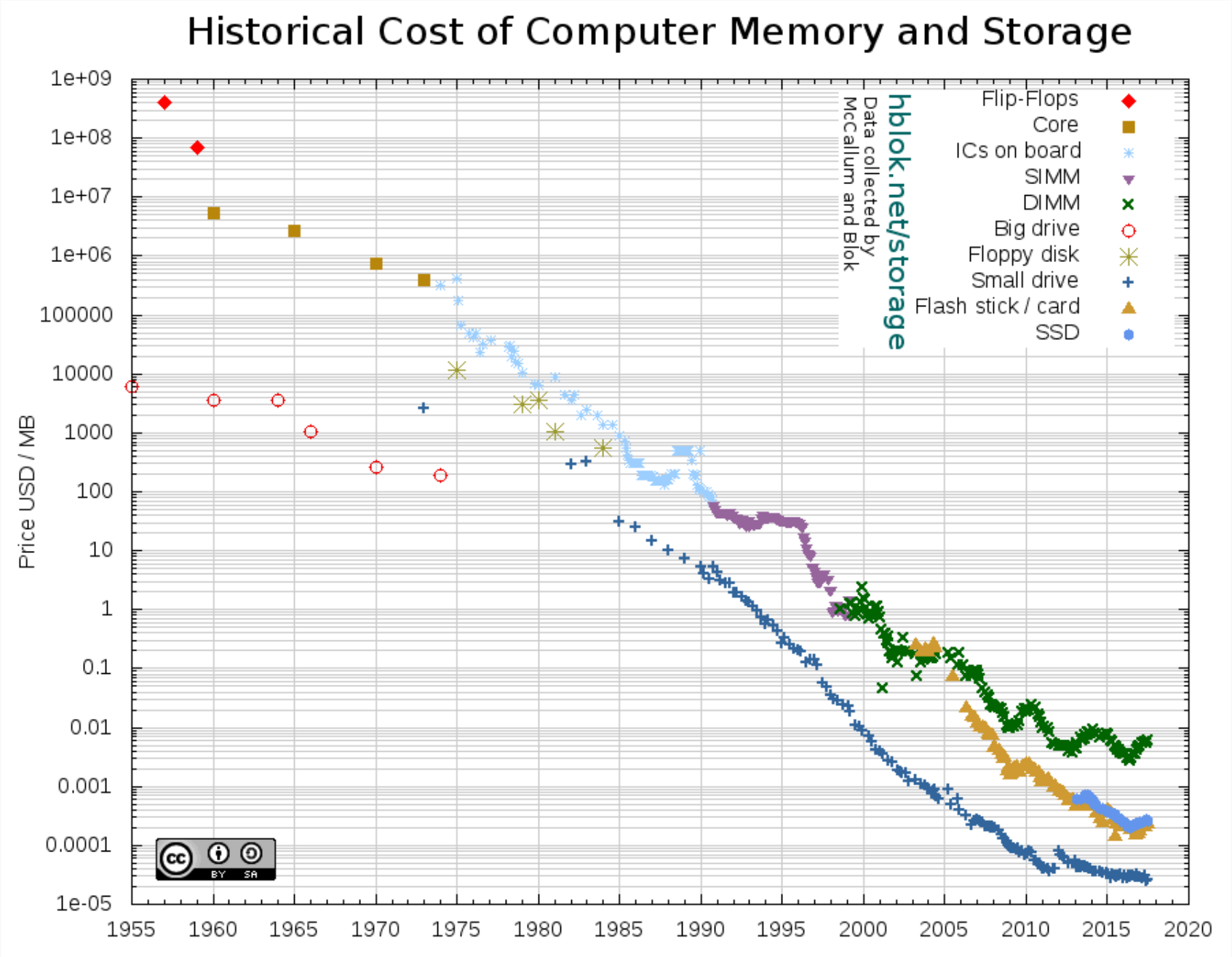
The “Dinosaurs”

- the earliest commercially available relational systems were Oracle, IBM System R, Ingres (end of 1970s)
- in 1990s and 2010s Oracle, MS SQL Server, IBM DB2 were the dominant players (in terms of mind- and market share)
- general-purpose systems: OLTP and OLAP
- behavior similar (all speak SQL), but different enough to make switching hard
- similar internal architecture
- stable, great functionality (ACID transactions, SQL)
- sometimes hard to use (database administrator needed), many tuning knobs to get better performance
- database systems research was kind of boring

The Anatomy of a Dinosaur

feature	technique
transaction isolation	2 Phase Locking
synchronization	latching (“lock coupling”)
large data sets	buffer management
durability	ARIES-style logging
indexing	B+tree
storage	slotted pages (row-wise)
SQL	iterator model (interpreter)
parallelization	Exchange operators
query optimization	cost-based DP

- the assumption is that the only thing that for performance is disk I/O operations


Historical Cost of Computer Memory and Storage
Price USD / MB
1e+09
1e+08
1e+07
1e+06
100000
10000
1000
100
10
1
0.1
0.01
0.001
0.0001
1e-05
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
hblok.net/storage
Data collected by
McCallum and Blok
Flip-Flops
Core
ICs on board
SIMM
DIMM
Big drive
Floppy disk
Small drive
Flash stick / card
SSD
BY SA

OLTP Through the Looking Glass, and What We Found There

Stavros Harizopoulos
HP Labs
Palo Alto, CA
stavros@hp.com

Daniel J. Abadi
Yale University
New Haven, CT
dna@cs.yale.edu

Samuel Madden Michael Stonebraker
Massachusetts Institute of Technology
Cambridge, MA
{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 1970's. 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 (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 raw 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, latching, locking, B-tree, and buffer management operations.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems — *transaction processing; concurrency.*

1. INTRODUCTION

Modern general purpose online transaction processing (OLTP) database systems include a standard suite of features: a collection of on-disk data structures for table storage, including heap files and B-trees, support for multiple concurrent queries via locking-based concurrency control, log-based recovery, and an efficient buffer manager. These features were developed to support transaction processing in the 1970's and 1980's, when an OLTP database was many times larger than the main memory, and when the computers that ran these databases cost hundreds of thousands to millions of dollars.

Today, the situation is quite different. First, modern processors are very fast, such that the computation time for many OLTP-style transactions is measured in microseconds. For a few thousand dollars, a system with gigabytes of main memory can be purchased. Furthermore, it is not uncommon for institutions to own networked clusters of many such workstations, with aggregate memory measured in hundreds of gigabytes — sufficient to keep many OLTP databases in RAM.

Second, the rise of the Internet, as well as the variety of data intensive applications in use in a number of domains, has led to a rising interest in database-like applications without the full suite of standard database features. Operating systems and networking conferences are now full of proposals for "database-like" storage systems with varying forms of consistency, reliability, concurrency, replication, and queryability [DG04, CDG+06, GBH+00, SMK+01].

This rising demand for database-like services, coupled with dra-

OLTP Through the Looking Glass [SIGMOD 2008]

- even a decade ago, the working set of many applications fit into main memory
- research question: Where does time go in OLTP?
- approach: disable/rip out components step by step (+ additional micro-optimizations)
- use Shore system
  - open source storage engine
  - developed at University of Wisconsin in early 1990s
  - architecturally similar to Dinosaurs
  - the assumption is that the results should be similar too

General Setup

- single-core Pentium 4, 3.2 GHz
- Linux
- measure instructions, cycles
- use TPC-C (standard OLTP benchmark)

TPC-C Schema

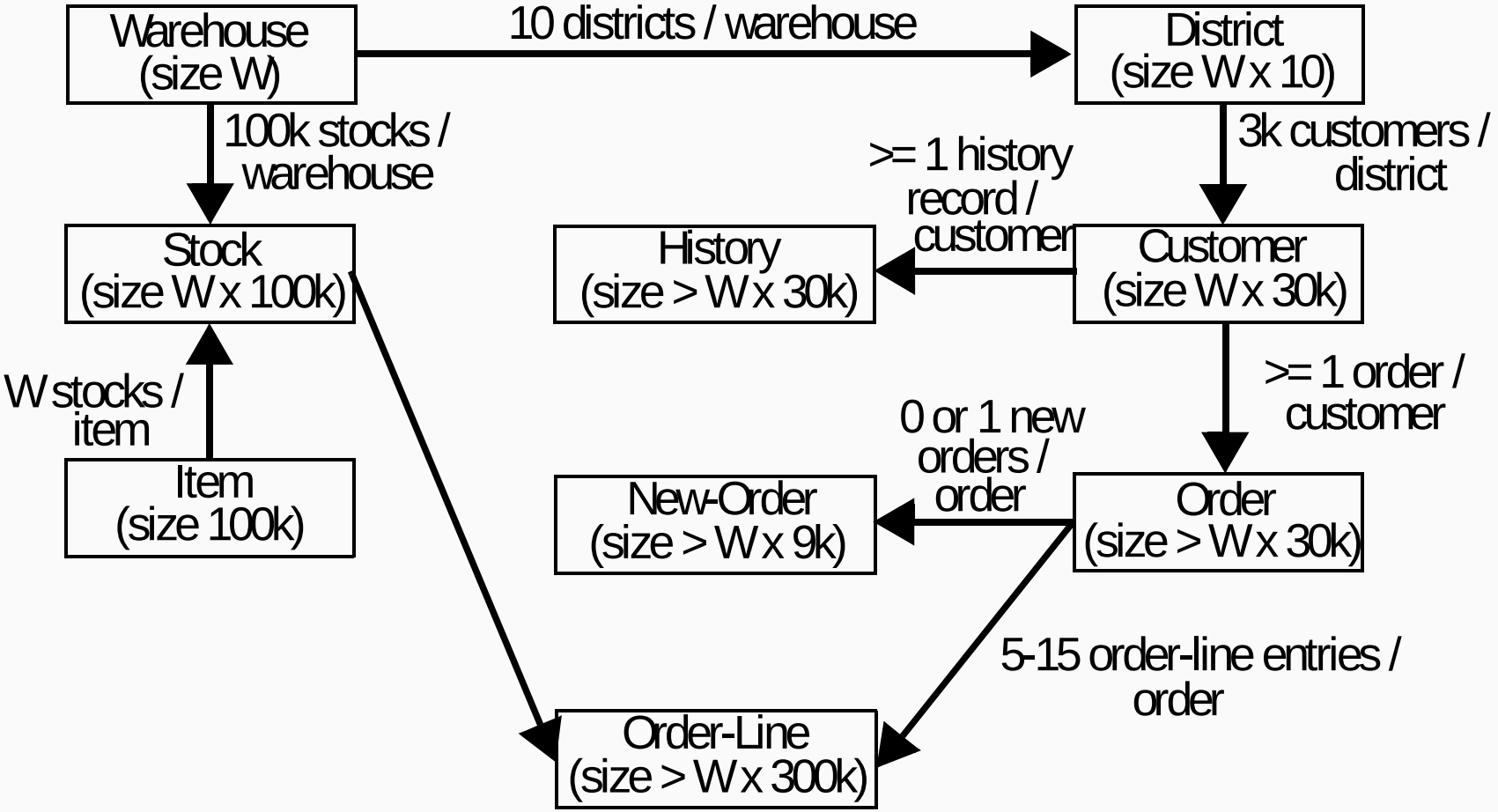


Figure 3. TPC-C Schema.

TPC-C Transactions

New Order	Payment
begin	begin
for loop(10)	Btree lookup(D), pin
.....Btree lookup(I), pin	Btree lookup (W), pin
Btree lookup(D), pin	Btree lookup (C), pin
Btree lookup (W), pin	update rec (C)
Btree lookup (C), pin	update rec (D)
update rec (D)	update rec (W)
for loop (10)	create rec (H)
.....Btree lookup(S), pin	commit
.....update rec (S)	
.....create rec (O-L)	
.....insert Btree (O-L)	
create rec (O)	
insert Btree (O)	
create rec (N-O)	
insert Btree (N-O)	
insert Btree 2ndary(N-O)	
commit	

Figure 4. Calls to Shore's methods for New Order and Payment transactions.

Instruction Breakdown

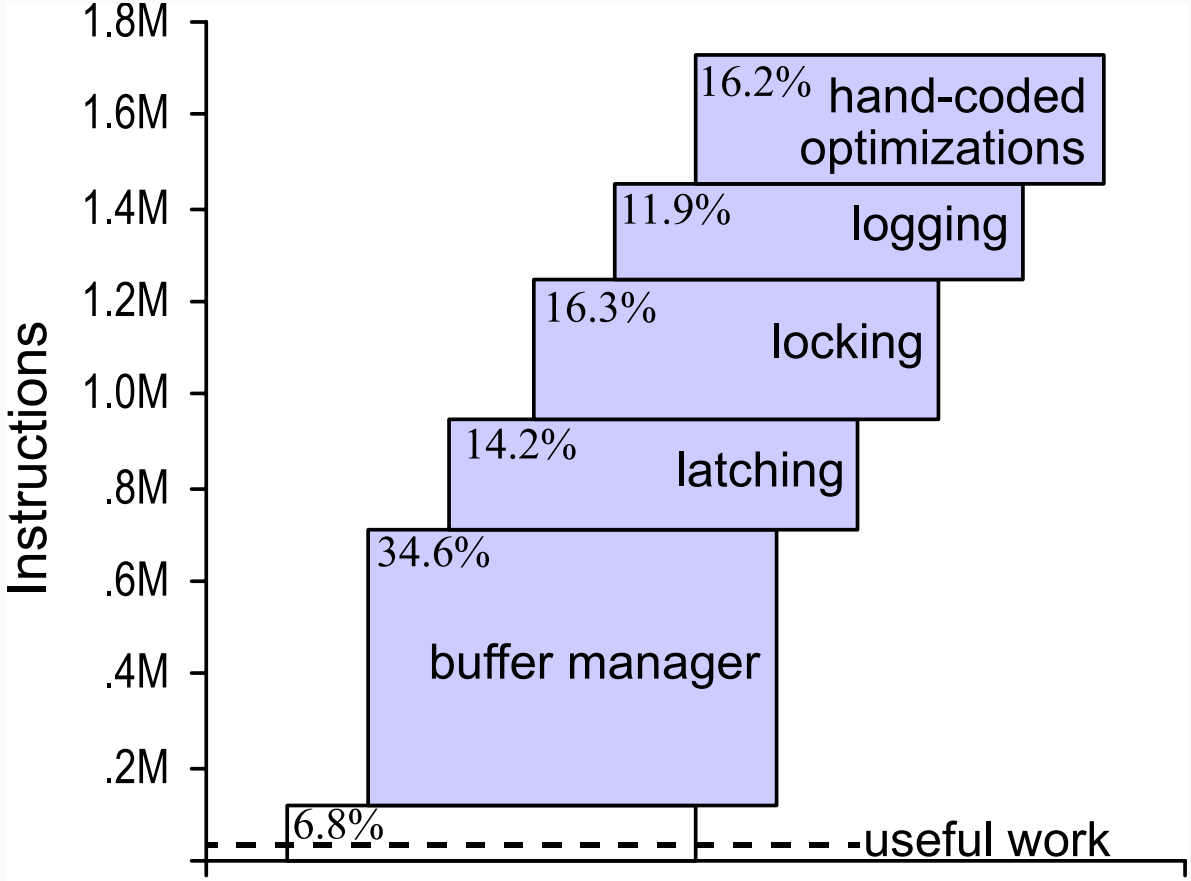


Figure 1. Breakdown of instruction count for various DBMS components for the New Order transaction from TPC-C. The top of the bar-graph is the original Shore performance with a main memory resident database and no thread contention. The bottom dashed line is the useful work, measured by executing the transaction on a no-overhead kernel.

Instructions vs. Cycles

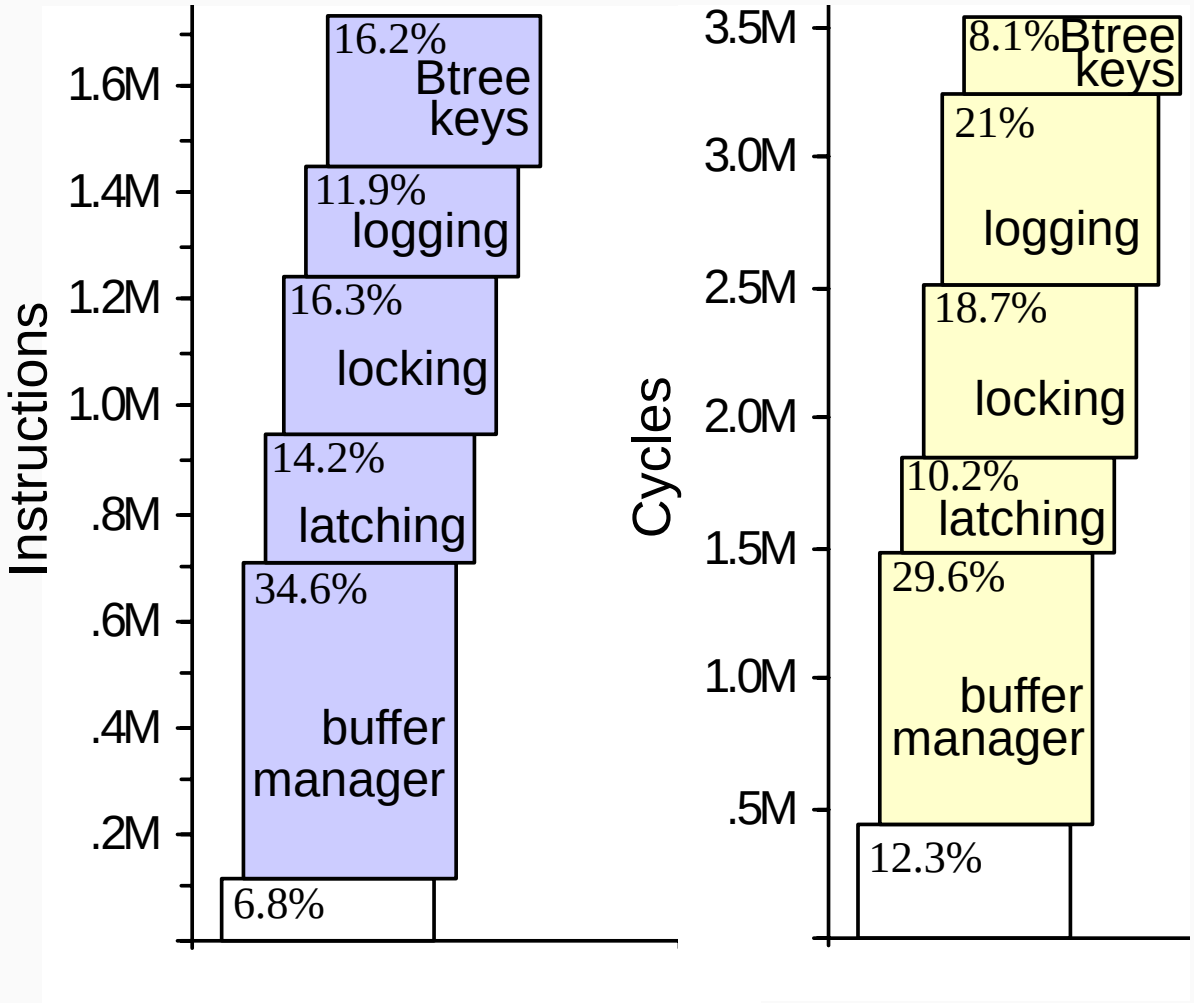


Figure 8. Instructions (left) vs. Cycles (right) for New Order.

Transactions Per Second

- out-of-the-box Shore: 640
- disable log flushing: 1,700
- disable components: 12,700
- standalone C implementation: 46,500

OLTP Through the Looking Glass: Conclusions

- traditional database implementation and architecture are extremely inefficient
- 10× to 100× performance gains are achievable
- all components are slow (“no high pole in the tent”)

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, dna, stavros}@csail.mit.edu

Nabil Hachem
AvantGarde Consulting, LLC
nhachem@agdba.com

Pat Helland
Microsoft Corporation
phelland@microsoft.com

ABSTRACT

In previous papers [SC05, SBC+07], some of us predicted the end of "one size fits all" as a commercial relational DBMS paradigm. These papers presented reasons and experimental evidence that showed that the major RDBMS vendors can be outperformed by 1-2 orders of magnitude by specialized engines in the data warehouse, stream processing, text, and scientific database markets.

Assuming that specialized engines dominate these markets over time, the current relational DBMS code lines will be left with the business data processing (OLTP) market and hybrid markets where more than one kind of capability is required. In this paper we show that current RDBMSs can be beaten by nearly two orders of magnitude in the OLTP market as well. The experimental evidence comes from comparing a new OLTP prototype, H-Store, which we have built at M.I.T., to a popular RDBMS on the standard transactional benchmark, TPC-C.

We conclude that the current RDBMS code lines, while attempting to be a "one size fits all" solution, in fact, excel at nothing. Hence, they are 25 year old legacy code lines that should be retired in favor of a collection of "from scratch" specialized engines. The DBMS vendors (and the research community) should start with a clean sheet of paper and design systems for tomorrow's requirements, not continue to push code lines and architectures designed for yesterday's needs.

1. INTRODUCTION

All three systems were architected more than 25 years ago, when hardware characteristics were much different than today. Processors are thousands of times faster and memories are thousands of times larger. Disk volumes have increased enormously, making it possible to keep essentially everything, if one chooses to. However, the bandwidth between disk and main memory has increased much more slowly. One would expect this relentless pace of technology to have changed the architecture of database systems dramatically over the last quarter of a century, but surprisingly the architecture of most DBMSs is essentially identical to that of System R.

Moreover, at the time relational DBMSs were conceived, there was only a single DBMS market, business data processing. In the last 25 years, a number of other markets have evolved, including data warehouses, text management, and stream processing. These markets have very different requirements than business data processing.

Lastly, the main user interface device at the time RDBMSs were architected was the dumb terminal, and vendors imagined operators inputting queries through an interactive terminal prompt. Now it is a powerful personal computer connected to the World Wide Web. Web sites that use OLTP DBMSs rarely run interactive transactions or present users with direct SQL interfaces.

In summary, the current RDBMSs were architected for the business data processing market in a time of different user interfaces and different hardware characteristics. Hence, they all

The End of an Architectural Era (It's Time for a Complete Rewrite) [ICDE 2007]

- Stonebraker's lesson:
  - existing code bases are hopeless, rewrite needed
  - specialized, simplified systems for OLTP, OLAP, text, etc.
  - let's start lots of startups building specialized systems (OLTP: H-Store, OLAP: C-Store/Vertica, Data Integration/Cleaning: Tamr)
- German lesson:
  - general-purpose relational systems are kind of nice
  - DRAM is cheap (1 TB RAM for less than 50K EUR), let's go in-memory only
  - SAP HANA, TUM HyPer

Next Week: In-Memory Indexing

- The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases
  Viktor Leis, Alfons Kemper, Thomas Neumann, ICDE 2013
- HOT: A Height Optimized Trie Index for Main-Memory Database Systems
  Robert Binna, Eva Zangerle, Martin Pichl, Günther Specht, Viktor Leis, SIGMOD 2018