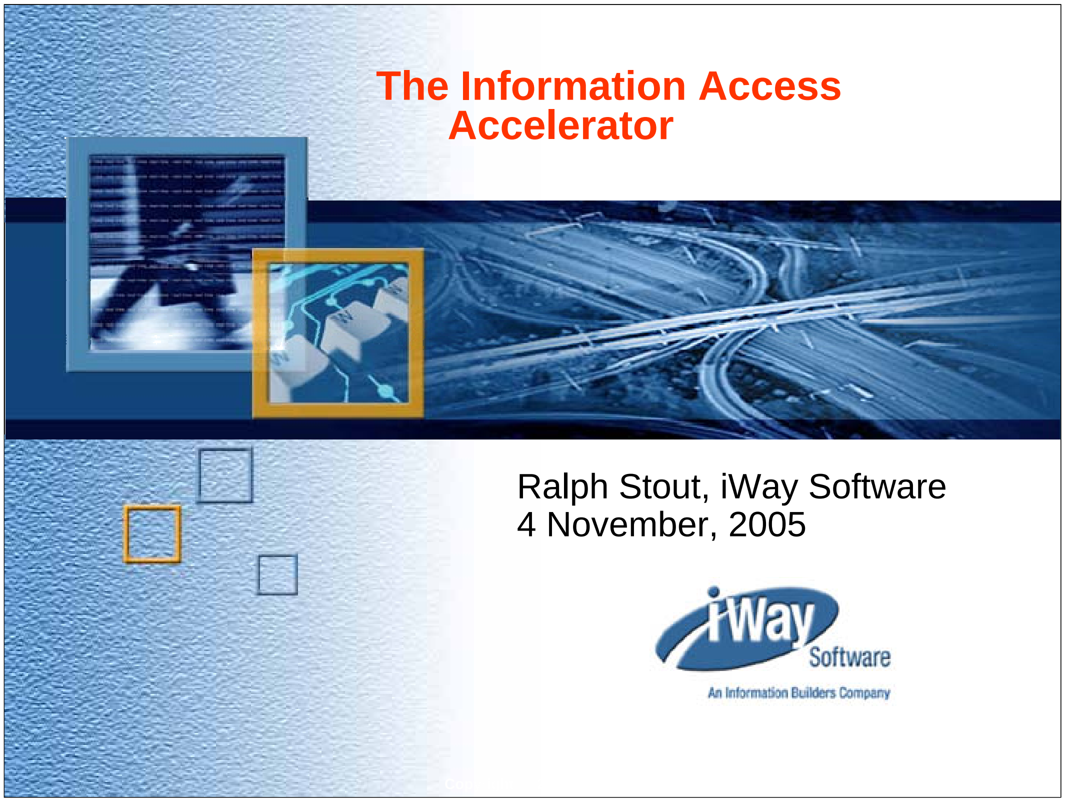
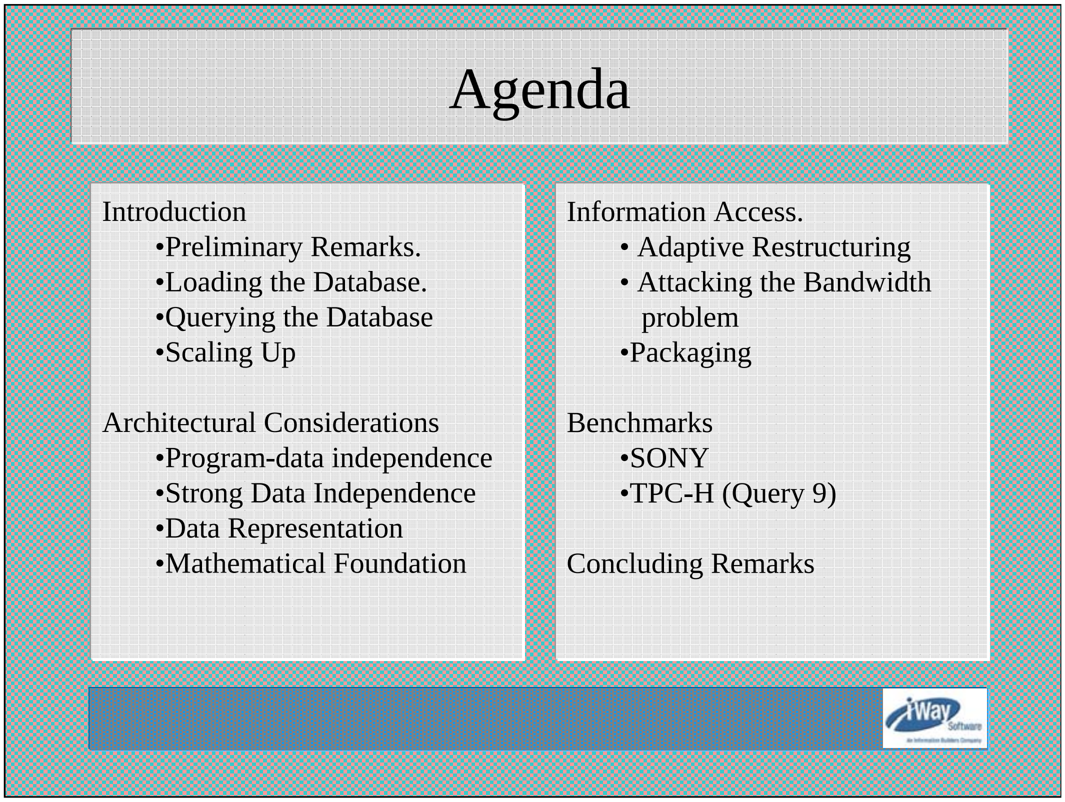


The Information Access Accelerator

Ralph Stout, iWay Software
4 November, 2005

iWay Software
An Information Builders Company



Agenda

Introduction

- Preliminary Remarks.
- Loading the Database.
- Querying the Database
- Scaling Up

Architectural Considerations

- Program-data independence
- Strong Data Independence
- Data Representation
- Mathematical Foundation

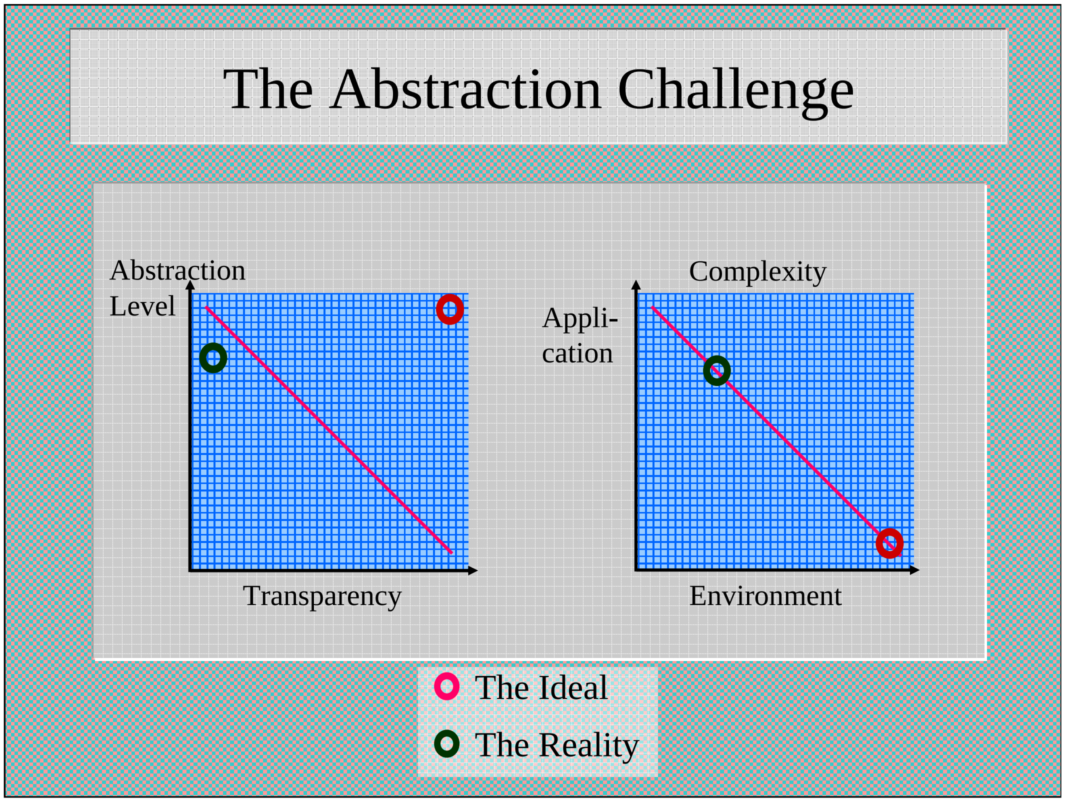
Information Access.

- Adaptive Restructuring
- Attacking the Bandwidth problem
- Packaging

Benchmarks

- SONY
- TPC-H (Query 9)

Concluding Remarks



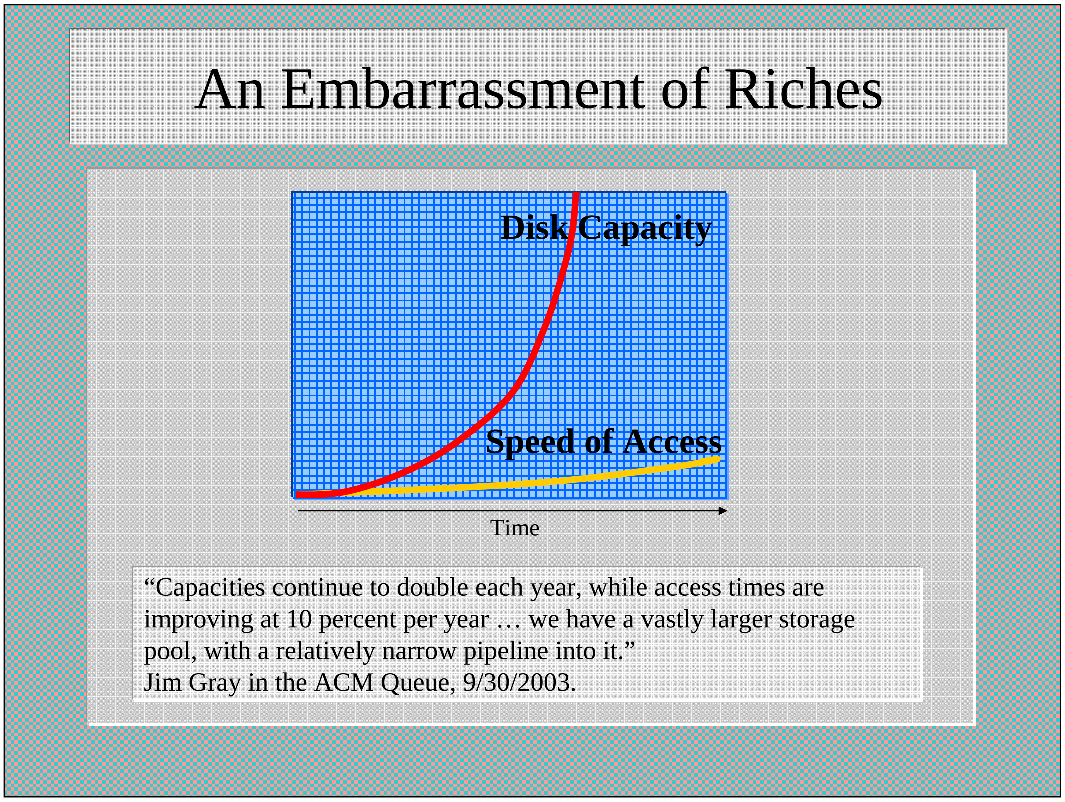
The Abstraction Challenge

○ The Ideal

○ The Reality

Left diagram: As the abstraction level they support rises, software systems tend to become increasingly opaque, making it impossible for application designers to see (or control) the factors that govern performance. We seem to be unable to achieve a high level of abstraction without interfering with our ability to view and manipulate performance-related software layers. Moore's Law has helped mask the performance penalties associated with high abstraction and low transparency levels but it can not do so indefinitely.

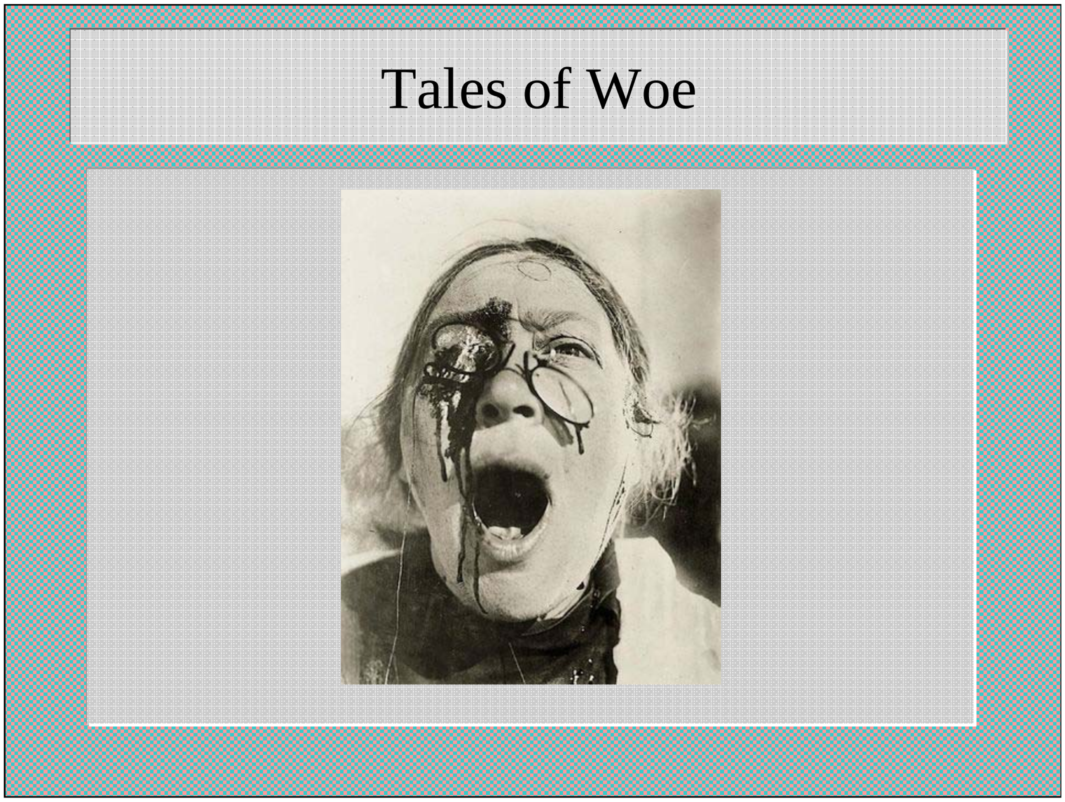
Right diagram: It goes without saying that trivial applications running in trivial environments can't do anything interesting. Complexity must be rooted either in the software infrastructure, the application system, or both. The more complexity the infrastructure absorbs, the better developers like it. To attain their goal (which is to write no code at all) IT organizations force software vendors to keep boosting what is already a dangerously high level of abstraction.



An Embarrassment of Riches

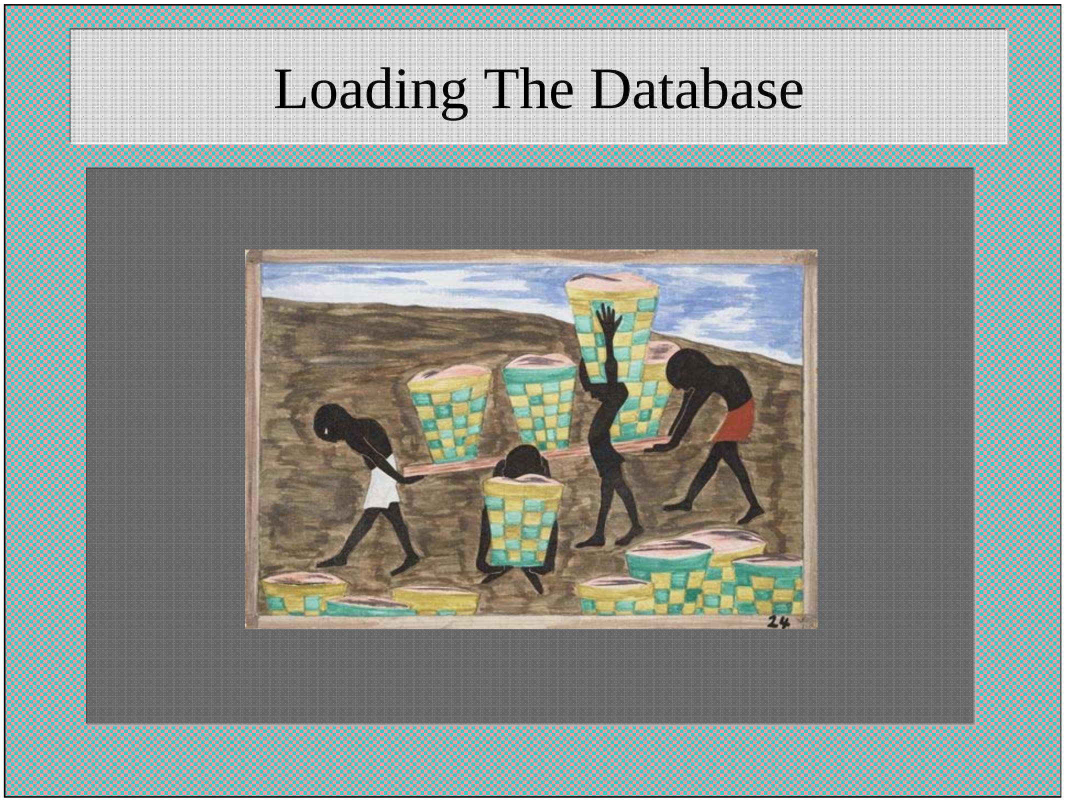
"Capacities continue to double each year, while access times are improving at 10 percent per year … we have a vastly larger storage pool, with a relatively narrow pipeline into it."
Jim Gray in the ACM Queue, 9/30/2003.

Storage capacity is increasing much faster than our ability to access information. If commercial RDBMS continue to read only a few kilobytes of data at a time, they will not achieve the bandwidth needed to perform operations on data sets that will soon be two orders of magnitude larger than they are today. RDBMSs vendors are reluctant to abandon the random access strategies they have used since the 1970s. But they must – and the transition is likely to be painful. It means shedding access mechanisms that many developers regard as sacrosanct. Who would have thought, for example, that B-Trees and other venerable data structures would eventually become a barrier to performance? Our experiments with XSP, an access engine that eschews pointers of any kind, have shown pretty conclusively that conventional, pointer-based, storage structures already represent more of a problem than a solution. As you will see, XSP regularly outperforms DB2 and Oracle. Why? Because it is not constrained by small page sizes and because it can exert a greater degree of performance control over its I/O access mechanism than any commercial DBMS.



Tales of Woe

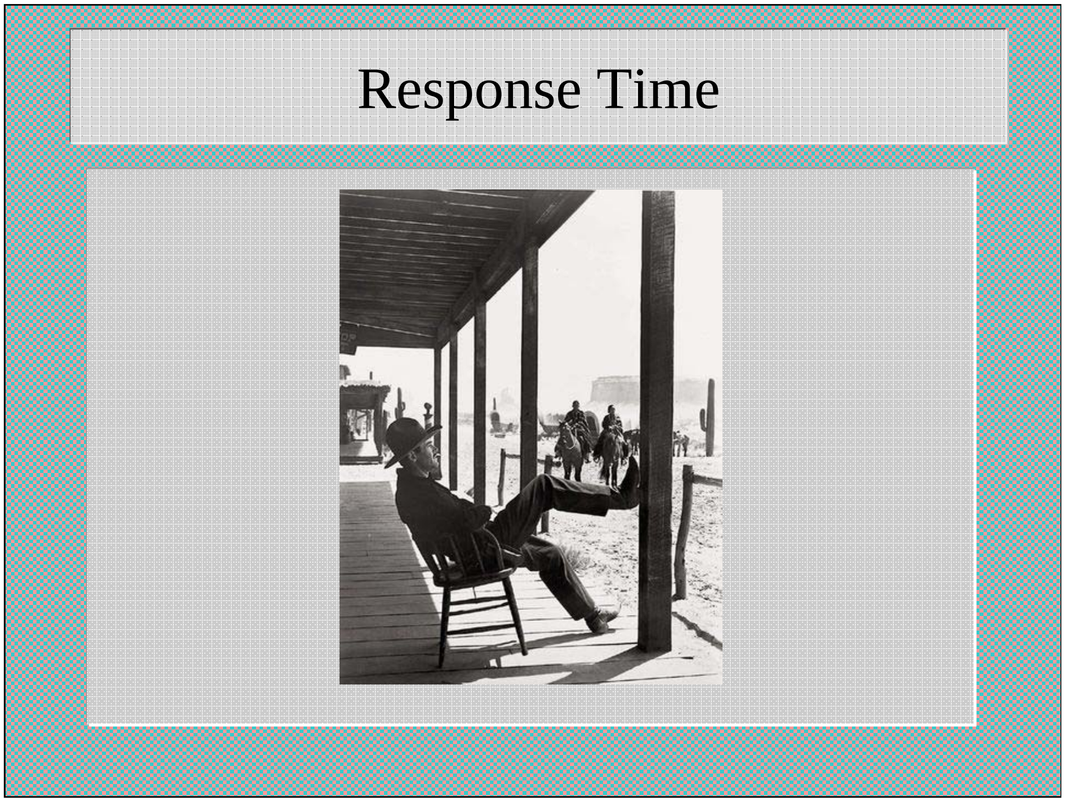
People are not satisfied with their DBMSs. They take too long to load data; they bog down when the database gets too large and when the queries get too complex.



Loading The Database

Consider the curious story of Lewis Cunningham, who, in a blog entitled “The Load to Hell …” recounts his experiences loading 800 million records into an Oracle database. After processing only 35 million rows in the first ten days (ten days!), Cunningham, who was under time pressure, decided to start over from scratch. Plan B, which entailed discarding some indices and foreign keys, came to grief because of a persistent Oracle bug. Plan C entailed discarding *all* primary keys, sacrificing even more integrity constraints and indices and bypassing the Oracle loader. Cunningham’s blog speaks volumes about the state of the database art circa 2005. Only by writing an application-specific loader and emasculating the index structure of the database, was he able to meet his deadline. Looking back over the statistics posted by the native Oracle loader, we see that it was achieving a single-processor insertion rate of only 41 rows per second. Cunningham’s hand-tailored solution ran much faster but, then, it postponed most of the heavy lifting until later on. Index building, as any veteran of the ETL wars will tell you, sets you back.

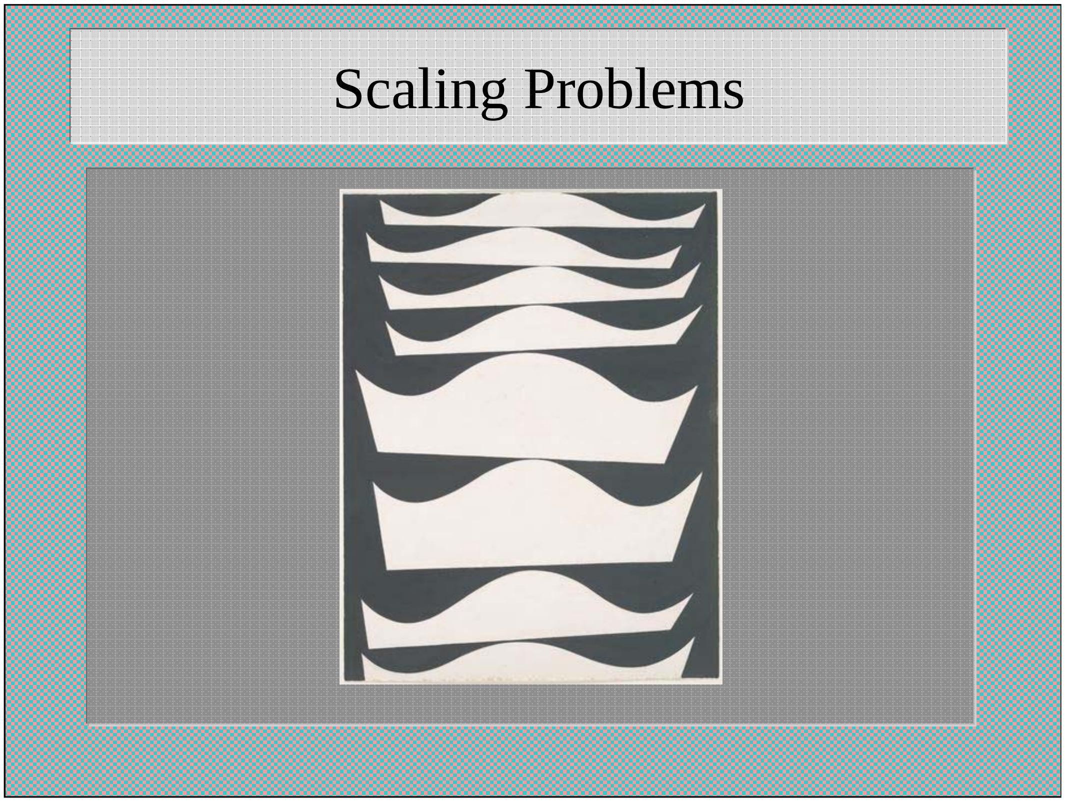
Reference: “The Load To Hell …”, ITtoolbox Blogs, Entry posted on 7/8/2005 by Lewis R. Cunningham (http://blogs.ittoolbox.com/oracle/guide/archives/004888.asp)



Response Time

Indexing schemes make sense only if their benefits more than compensate for the price they exact at load time. The results of a recent experiment[1] pitting Oracle’s sophisticated random access engine against a set of sequential access procedures written in SETL DB, call the wisdom of the B-tree indexing tradeoff into question. Oracle completed the suite of 21 TPC-H decision support queries against a one gigabyte database in 2266 seconds; SETL DB executed the same suite in 219 seconds. Why? In the course of processing the test script Oracle read innumerable, small, sparsely populated pages and SETL DB a relative handful of large, densely populated, pages. Oracle hopped from track to track, repositioning the read/write head and waiting for the platter to rotate. But SETL DB, having repositioned the read/write head, always pumped as much data as it possibly could through the I/O pipeline. This simple strategy kept disk arm movement and rotational delays in check; Oracle’s highly optimized indexing scheme did not.

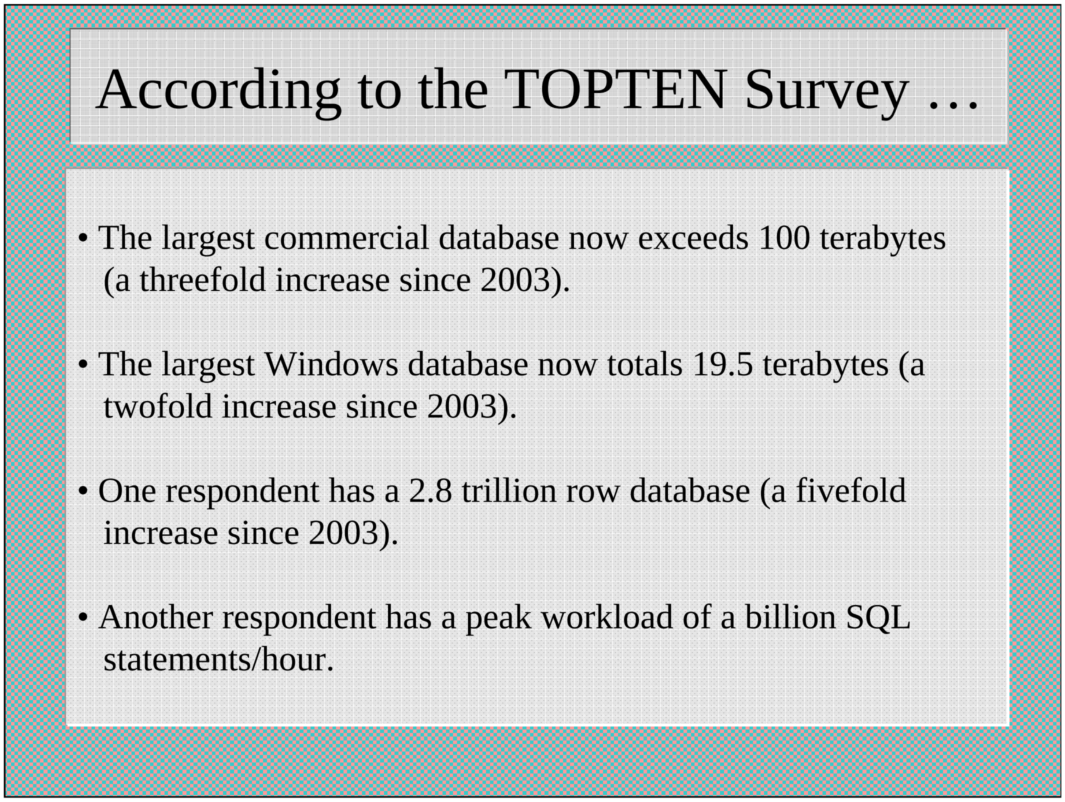
[1] Study conducted by Professor Jacob T. Schwartz and Dr. Toto Paxia of NYU; Ralph Stout and Yakov Kushnirsky of iWay Software.



Scaling Problems

Disk capacity is increasing at an unprecedented rate, much faster than that of the slender pipeline that links it to the CPU, and the resulting imbalance is generating waves of technological change that will have a profound impact on all of us. According to Jim Gray, the head of Microsoft’s Bay Area Research Center, “capacities continue to double each year, while access times are improving at 10 percent per year.”[1] If this trend continues, IT will be unable to cope with database growth.

[1] ACM Queue, June, 2003, “A conversation with Jim Gray,” a June, 2003 (interview conducted by Dave Patterson).

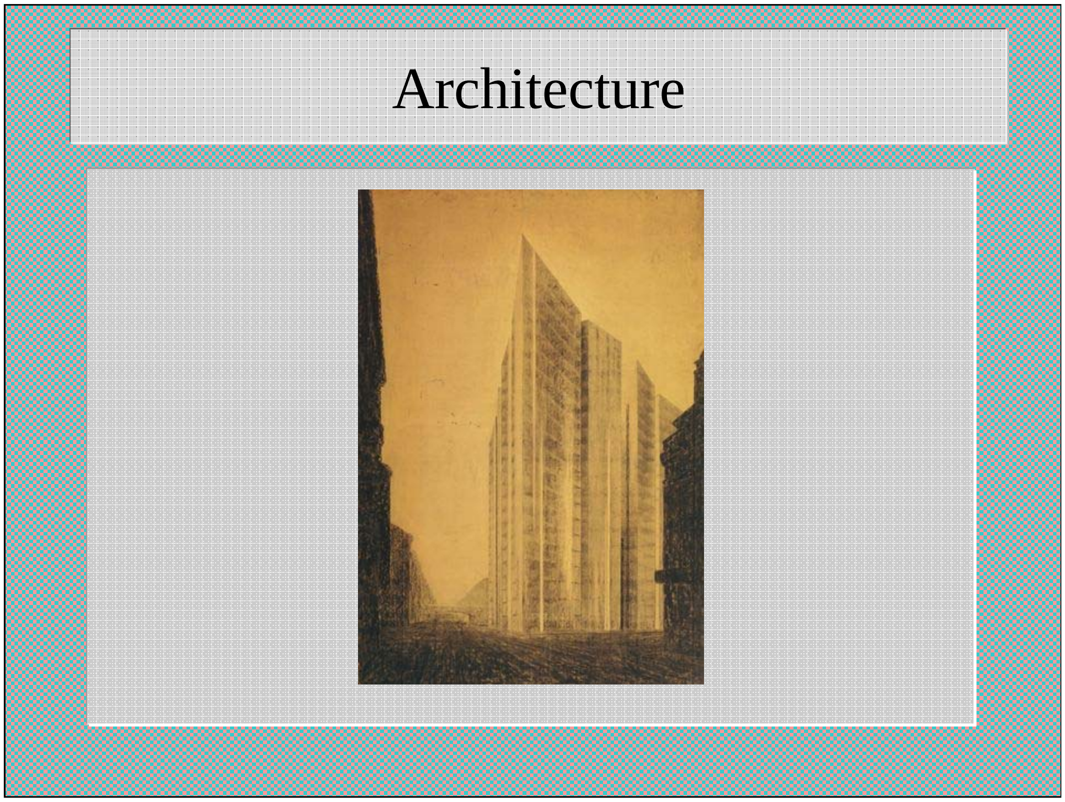


According to the TOPTEN Survey …

- The largest commercial database now exceeds 100 terabytes (a threefold increase since 2003).
- The largest Windows database now totals 19.5 terabytes (a twofold increase since 2003).
- One respondent has a 2.8 trillion row database (a fivefold increase since 2003).
- Another respondent has a peak workload of a billion SQL statements/hour.

According to the most recent TOPTEN Survey[1], the largest commercial database already exceeds 100 terabytes (a three-fold increase since 2003) and the largest database running within the Windows environment totals 19.5 terabytes. One respondent claims to have a 2.8 trillion row database (a five-fold increase since 2003) and another a peak workload of no less than a billion SQL statements per hour. These are impressive numbers. Now that the hardware constraints that have historically held database growth in check have been removed, rapid database growth is likely if not inevitable.

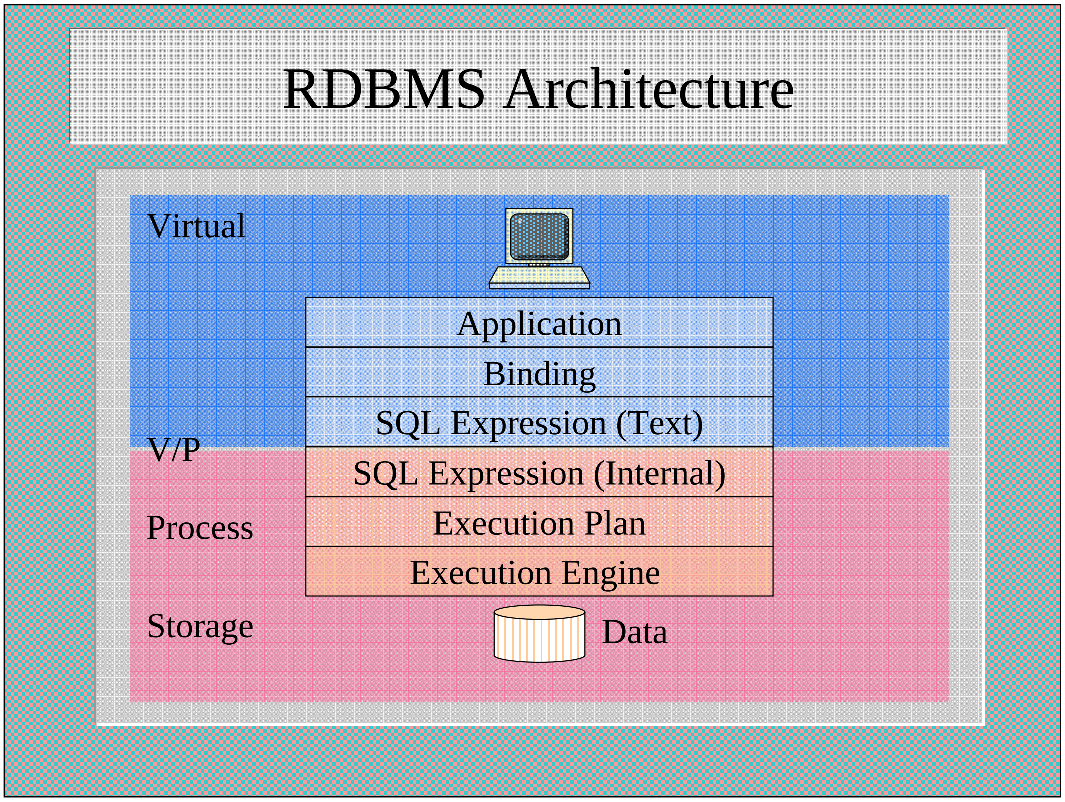
TOPTEN: A survey of the world’s largest and most heavily used databases conducted yearly by the Winter Corporation (www.wintercorp.com). HP, IBM Corporation , Microsoft Corporation, Oracle Corporation, Sun Microsystems, Inc. and Sybase Inc. sponsor this program.



Architecture

History: Early mainframe computers came with less memory and external storage than many of the prosaic consumer items on sale today at Wal-Mart. It is no accident that the designers of the pioneering SEQUEL-XRM and SYSTEM R prototypes that, in the 1970s, ushered in the relational era were especially conservative in their use of hardware resources. By 1981, when Oracle and SQL/DS, the first commercial implementations of the SQL language, hit the street, computers had improved — Moore's law was at work even then — but not nearly enough to justify a serious reexamination of the basic design decisions that contributed so much to the success of the first SQL prototypes. In consequence, Oracle and SQL/DS inherited the physical data access architecture of SYSTEM R and so, in time, did DB2, DG/SQL, SYBASE and INGRES.

SYSTEM R layered SQL language capabilities on top of the B-tree mechanism invented in 1971 by Rudolph Bayer, then a research scientist at Boeing. Over the years, B+-trees, B*-trees, 2-3 B-trees, B-trees with prefix compression and many other variations on Bayer's original theme have evolved but for the purposes of this discussion, it is sufficient to know that all B-tree engines — and by extension, all leading relational database management systems — partition the disk into units of modest size called nodes or pages and that they treat the disk as a random access device. This low bandwidth solution to low-level data access is the main reason why complex queries take so long to process.



RDBMS Architecture

Virtual

Application

Binding

SQL Expression (Text)

V/P

SQL Expression (Internal)

Process

Execution Plan

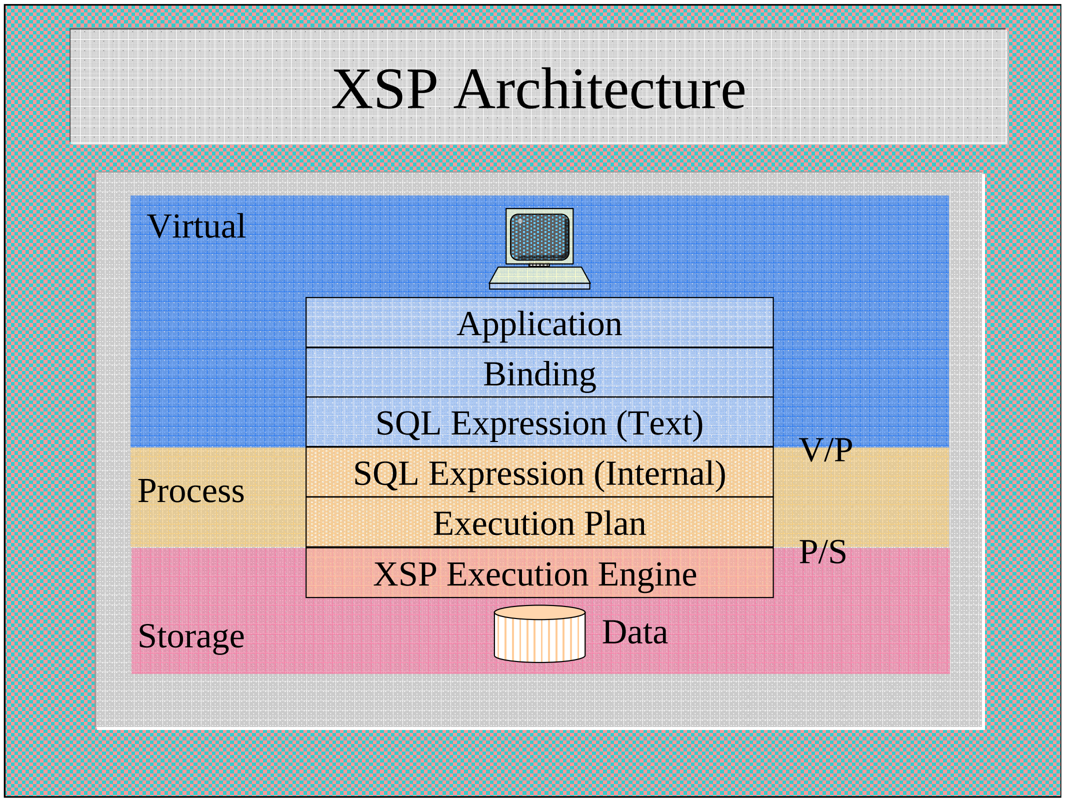
Execution Engine

Storage

Data

The strength of the relational data model lies in its ability both to uncouple application logic and data structures and to raise the degree of programming abstraction to a comfortable level. While the model says nothing about how to realize an RDBMS, the most popular implementations (from IBM, Oracle and Informix) are virtually identical. None offers much in the way of transparency, so if you are experiencing performance problems there isn’t much you can do much to remedy the situation.

Thanks to a well-drawn interface (V/P in the above diagram) between its virtual and process layers, the relational model delivers program/data independence. Unfortunately, because the process and storage layers lack a similarly rigorous interface, it is impossible to inject a better storage solution into the mix.



XSP Architecture

Virtual

Application

Binding

SQL Expression (Text)

V/P

Process

SQL Expression (Internal)

Execution Plan

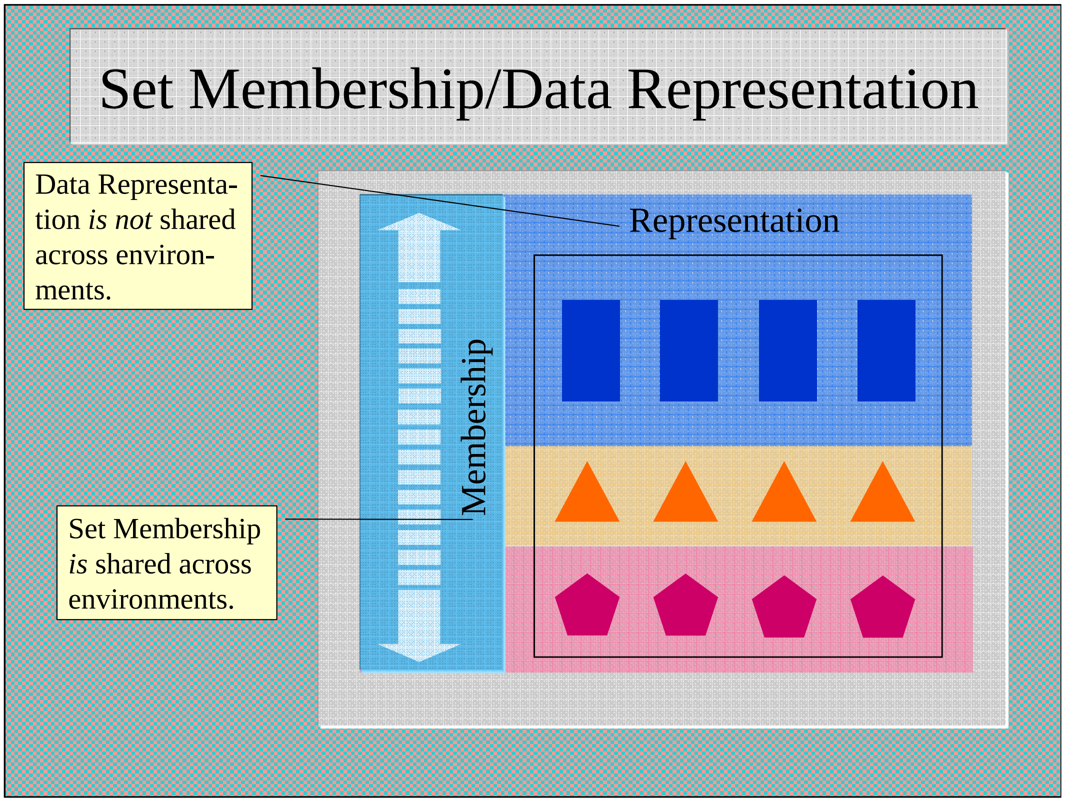
P/S

XSP Execution Engine

Storage

Data

In an XSP-based system the process and the storage layers operate independently. Here, instead of a single well-defined interface, there are two: V/P and P/S. With this architecture, it is possible to exchange an inefficient storage structure (e.g., B-Trees) for something better without having to re-implement the process subsystem. Conventional RDBMSs lack this degree of transparency.



Set Membership/Data Representation

Data Representation *is not* shared across environments.

Set Membership *is* shared across environments.

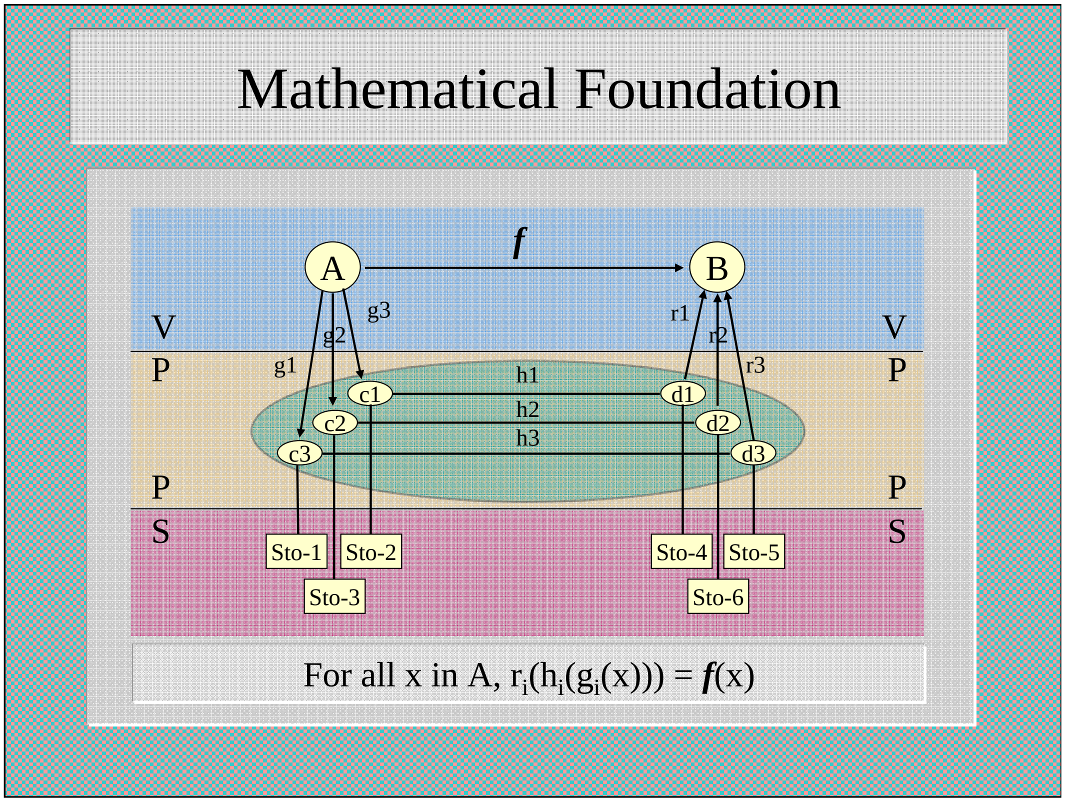
The XST architecture preserves membership across environments while making it possible for the data representation to vary from one environment to the next. Thus, performance at the process level can remain independent of how relationships are expressed at the conceptual level and how data is represented at the storage level.

This is the case for the XST approach in a nutshell:

(1) Set membership determines functionality.

(2) Data representation at the storage level determines performance.

(3) Strong data independence separates functionality concerns from performance concerns.



Mathematical Foundation

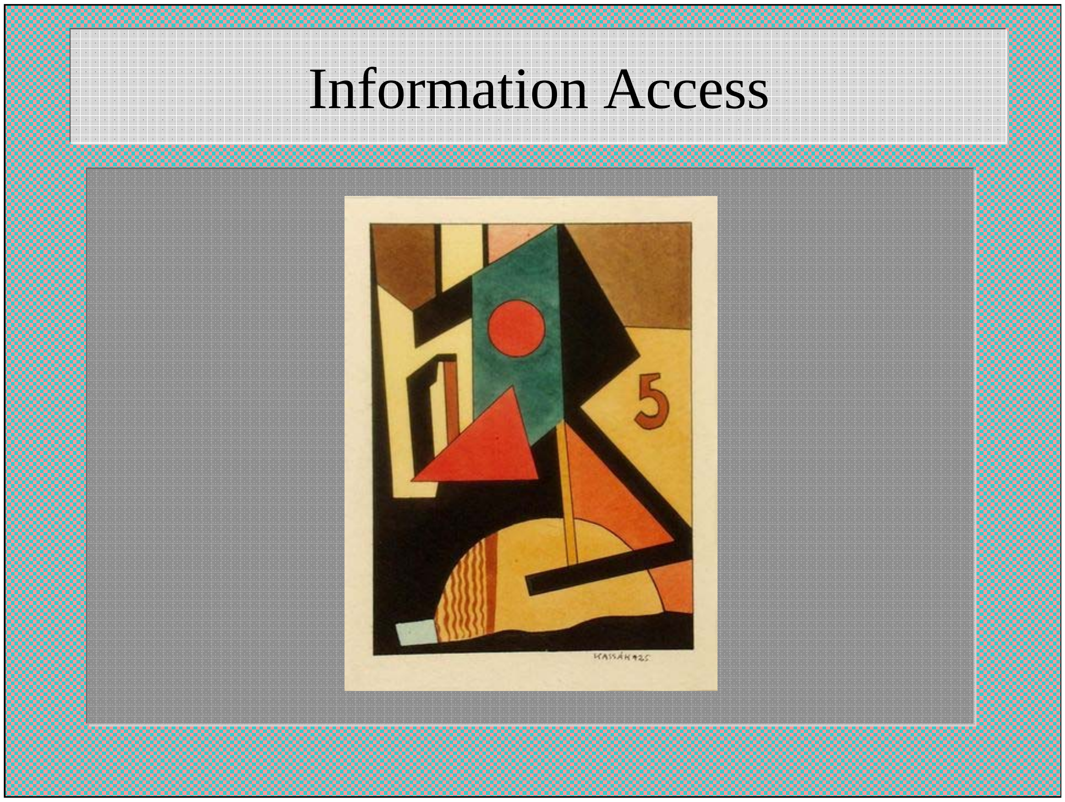
For all x in A, $r_i(h_i(g_i(x))) = \boldsymbol{f}(x)$

This is what mathematicians would call a commutative diagram.

Think of a system in terms of five components: three architectural layers (V, P and S) connected by two interfaces (V/P and P/S). V represents the application view level, P the database processing level and S the secondary storage level. From a mathematical point of view, the three levels are structurally distinct, disjoint spaces comprising operations and operands. The user view and processing levels are linked by the V/P interface (commonly called the query language interface) and the processing and storage levels are linked by the P/S interface (commonly called the I/O interface).

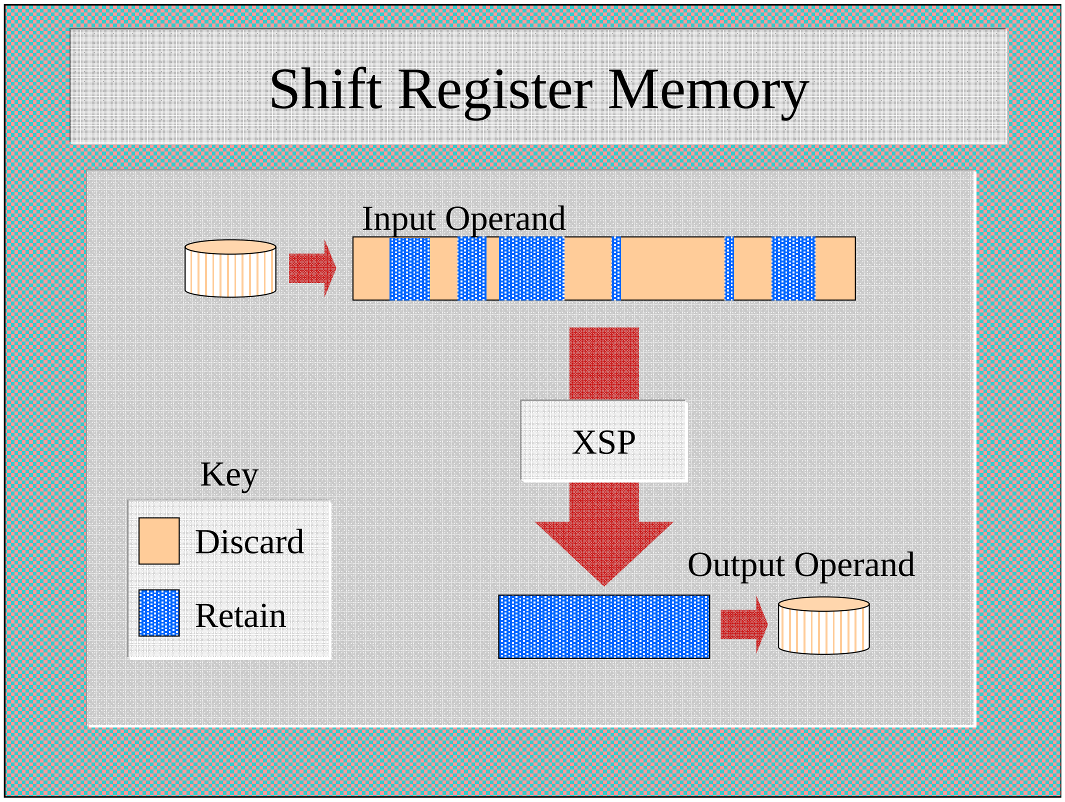
Let A and B represent any two collections of relational tables visible to application programs. We model the transformation of A into an implementation-friendly representation in P by g_i to c_i and the transformation of B into an implementation-friendly representation in P by r_i from d_i. The process, **f** , which transforms A to B, has meaning within V, but not within P. The problem: To find h_i that map c_i to d_i such that for all x in A, ri(hi(gi(x))) = f(x).

Systems, like XSP, which are based on the mapping of mathematical identities between spaces are said to be strongly data independent because no space in the mix knows how any other space organizes data. This level of independence goes beyond that supported by any RDBMS. RDBMSs insulate application logic from external data structures; XSP goes one step farther. It makes it possible to exert adaptive control over system performance (Imagine being able to replace a poor performing data structure with something better).



Information Access

Disk technology is moving into uncharted territory. Data access — positioning the disk arm and rotating the platter — is 10 times faster than it was in 1989 and I/O transfer rates are 40 times faster, but storage capacity has increased by a factor of no less than 10,000. According to Jim Gray of Microsoft, “if you read a few kilobytes in each access … it [would] take a year to read a 20-terabyte disk.” That’s low bandwidth — and since B-tree pages are rarely more than 75% full (when a page fills up, it splits into a pair of half-empty pages) DBMSs that rely on B-tree access make matters even worse. XSP attacks the bandwidth problem by (1) reading large blocks of data whenever it repositions the read/write head and (2) by optimizing I/O traffic with informationally dense data transfers.



Shift Register Memory

Input Operand

XSP

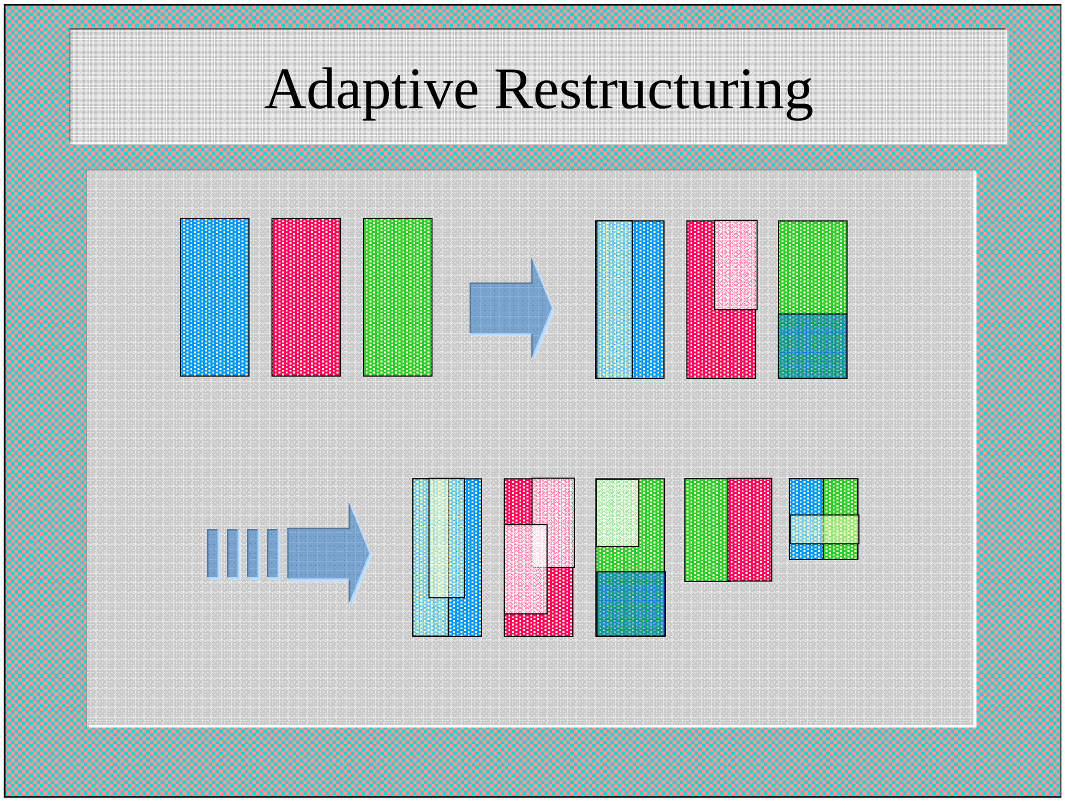
Key

🟧 Discard

🟦 Retain

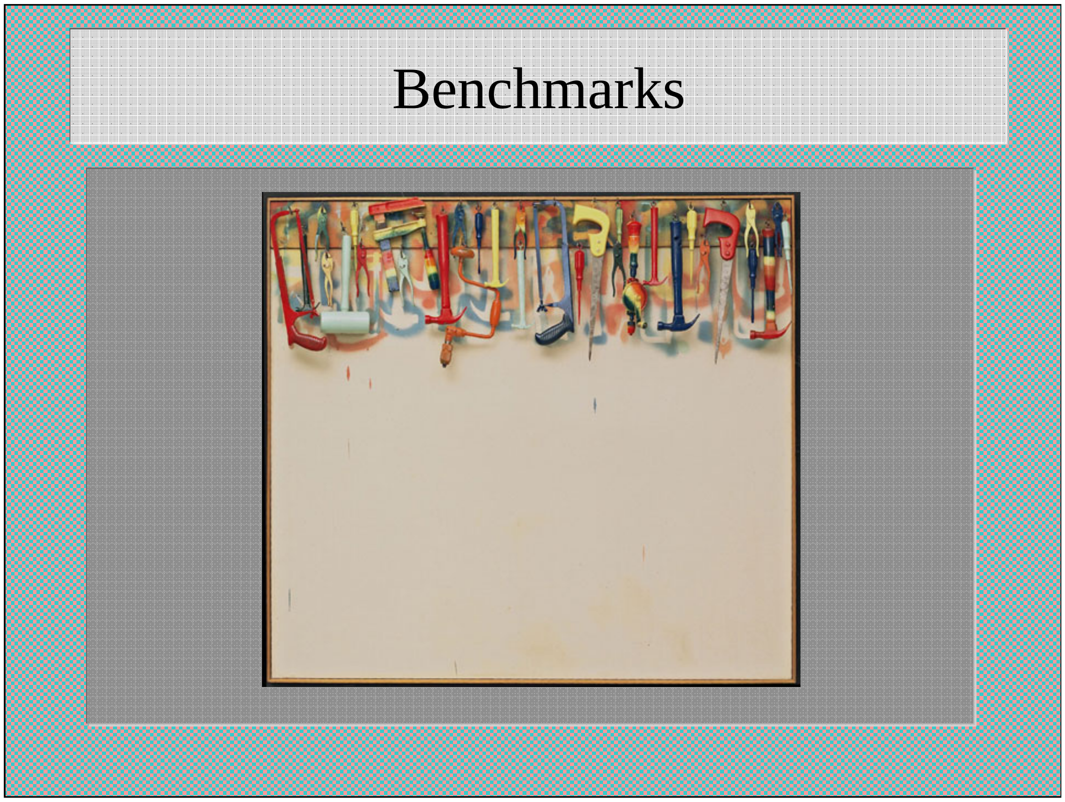
Output Operand

XSP works like a shift register memory, converting large data sets into smaller ones. Here, we see it reading an operand, which for the sake of argument we will assume is peppered with data relevant to the current query. The result: a compact operand containing nothing but relevant data. A complex analytical query would typically trigger many such operations. Each would reduce the amount of data in play until, finally, no more reduction is necessary. At that point, XSP would assemble its response and make itself available to process another request.



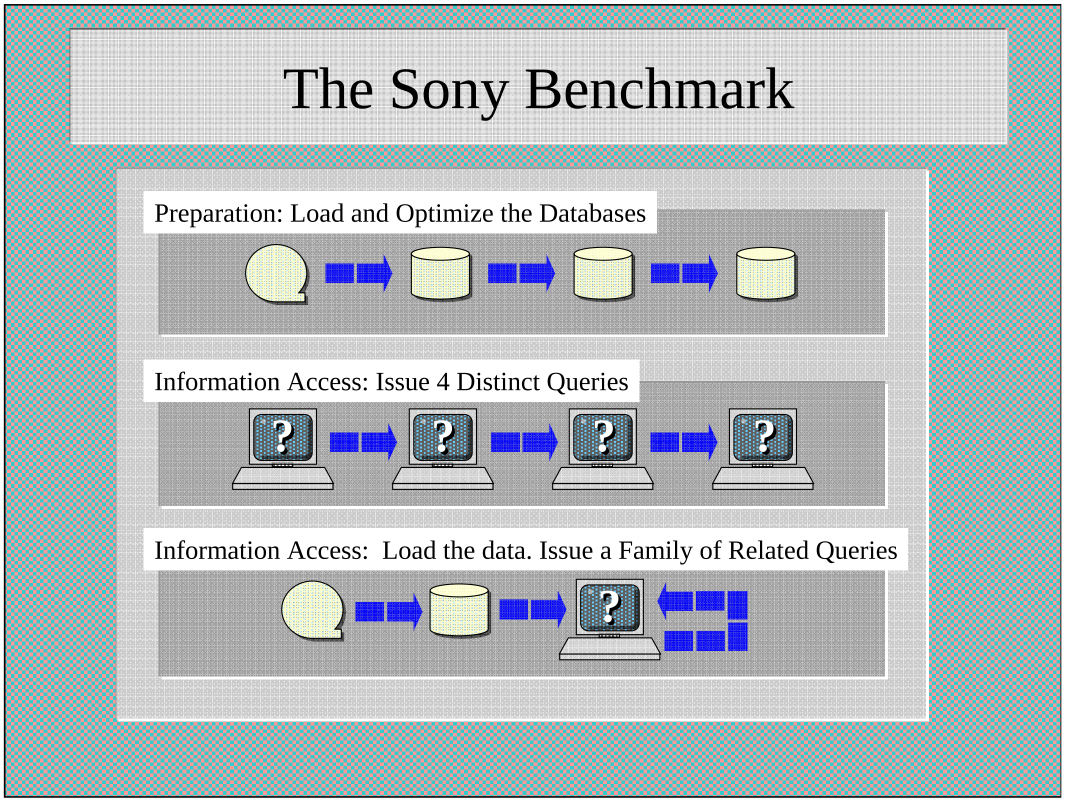
Adaptive Restructuring

Strong Data Independence makes adaptive restructuring for performance purposes possible. While, in theory, the composition of the database could evolve constantly, we plan to adopt a less volatile approach. In our scheme, the system would accumulate a knowledge base reflecting usage patterns and, when it is time to restructure, consult the knowledge base and act accordingly.



Benchmarks

iWay Software has been experimenting with XSP for some time. The results of two of our benchmark studies follow.



The Sony Benchmark

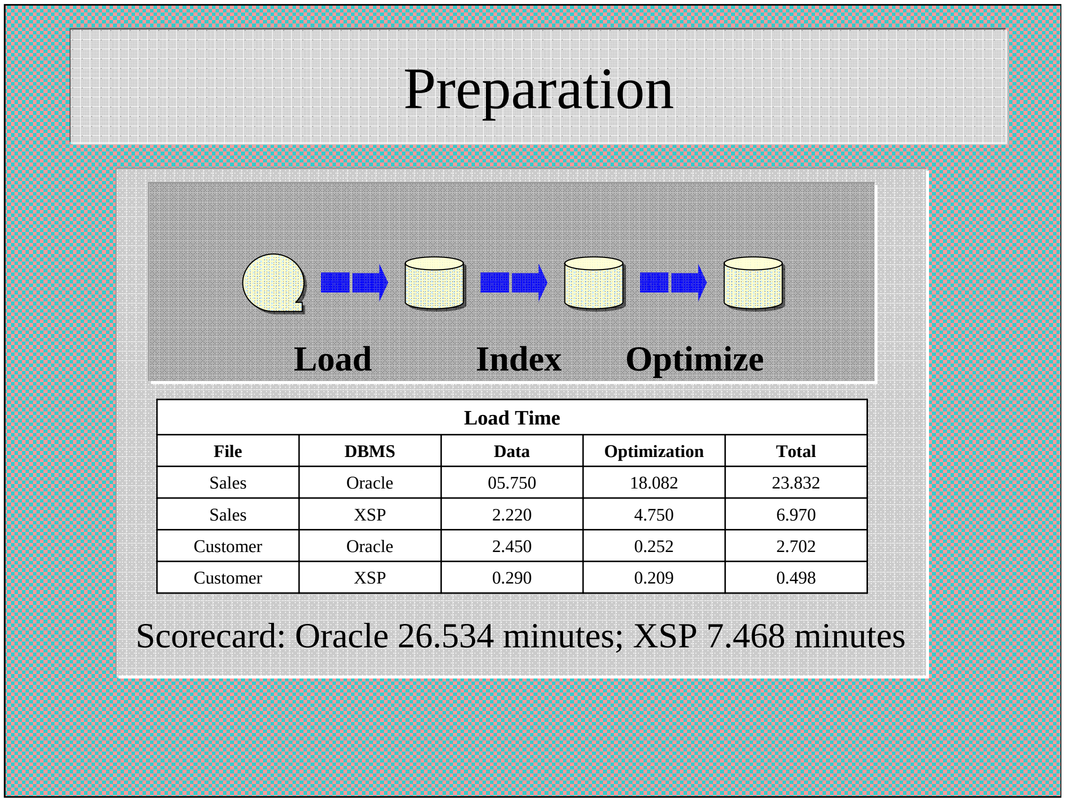
Preparation: Load and Optimize the Databases

Information Access: Issue 4 Distinct Queries

Information Access: Load the data. Issue a Family of Related Queries

The Sony series of benchmark tests measured database load time and query response time. The first test (top) entailed loading one gigabyte of tab-delimited raw data into a structured database and organizing it for optimum performance. The second test (center) entailed processing four problem queries. The third combined data loading and iterative query execution.

The test were conducted on a 2.6 MHz Pentium 4 PC with one Gigabyte of RAM and an 80 Gigabyte, 7200 rpm parallel ATA hard drive



Preparation

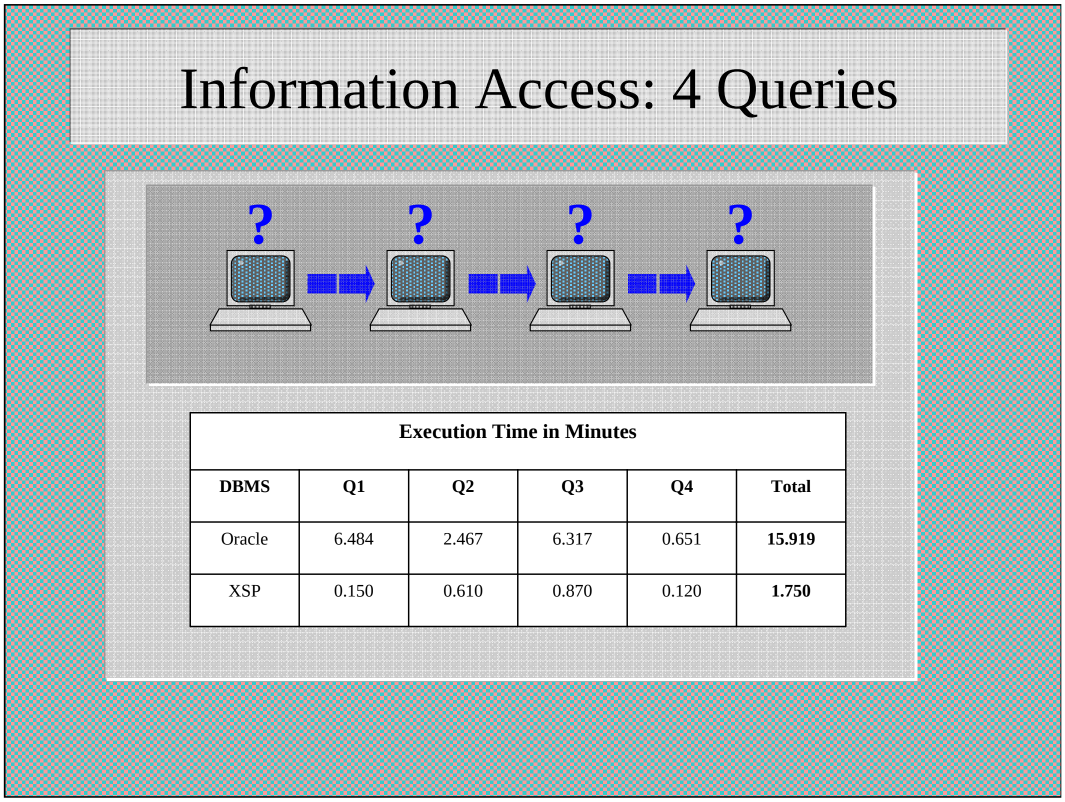
Load Index Optimize

Load Time				
File	**DBMS**	**Data**	**Optimization**	**Total**
Sales	Oracle	05.750	18.082	23.832
Sales	XSP	2.220	4.750	6.970
Customer	Oracle	2.450	0.252	2.702
Customer	XSP	0.290	0.209	0.498

Scorecard: Oracle 26.534 minutes; XSP 7.468 minutes

Given a set of tab-delimited raw data files and some questions to answer, RDBMSs cannot respond at once. They must first *load* the database and, assuming performance is a concern, build auxiliary index files and accumulate statistics as well. XSP is more flexible in this regard. XSP can either load the data up front or load it on the fly. In order to facilitate a comparison between Oracle and XSP, we directed XSP to load the data up front. This table summarizes the amount of time Oracle and XSP required to load and optimize the Sony Sales database. All timings are quoted in minutes and fractions of minutes.

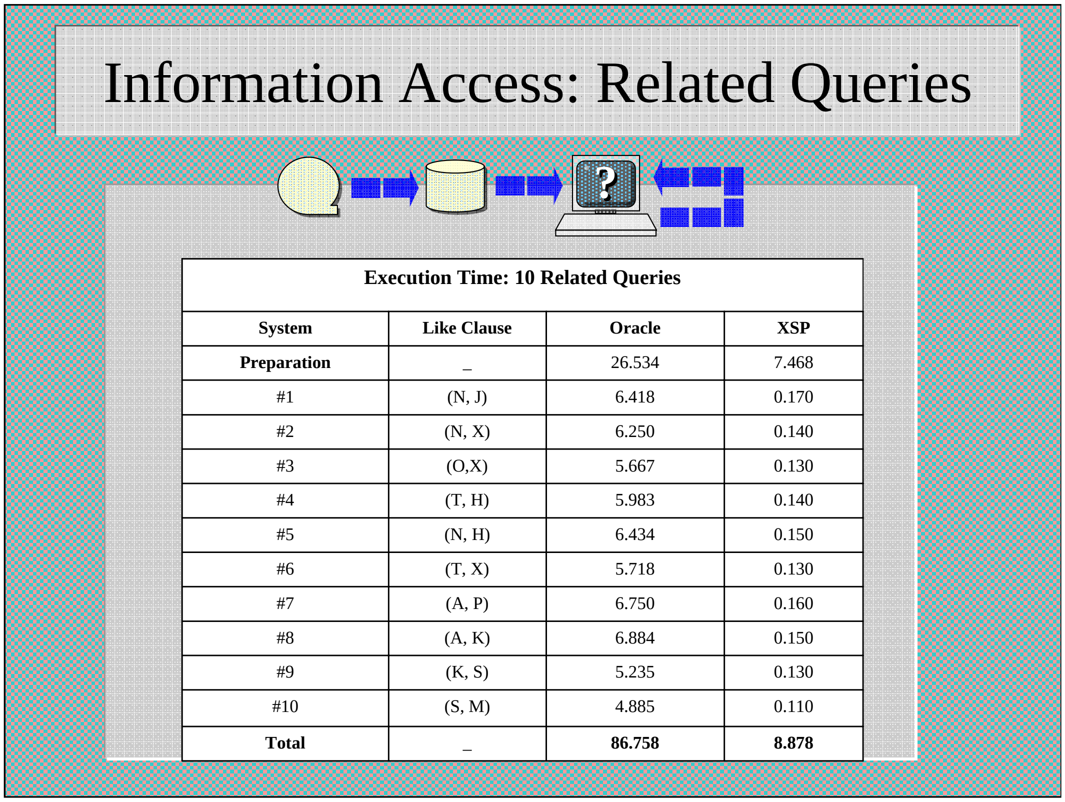
Oracle took a total of 26.534 minutes and XSP a total of 7.468 minutes to prepare the database. For Oracle, the optimization step entailed building indices and computing statistics. For XSP it entailed extracting some useful subsets from the raw data, manipulating the subsets and saving them for future use.



Information Access: 4 Queries

Execution Time in Minutes					
DBMS	**Q1**	**Q2**	**Q3**	**Q4**	**Total**
Oracle	6.484	2.467	6.317	0.651	**15.919**
XSP	0.150	0.610	0.870	0.120	**1.750**

Sony supplied four sample SQL queries, which for convenience we will call Q1, Q2, Q3 and Q4. We combined the queries into scripts and executed the scripts, taking care to isolate the systems being tested, both from the network, which could have a destabilizing effect, and from competing application systems. This table summarizes our findings. Although the Sony database is small (1 gigabyte), the queries are complex. Q1, for example, contains two sub-queries, both calling for an outer join. The sub-queries generate intermediate answer sets, which then are (inner) joined to produce a final result.

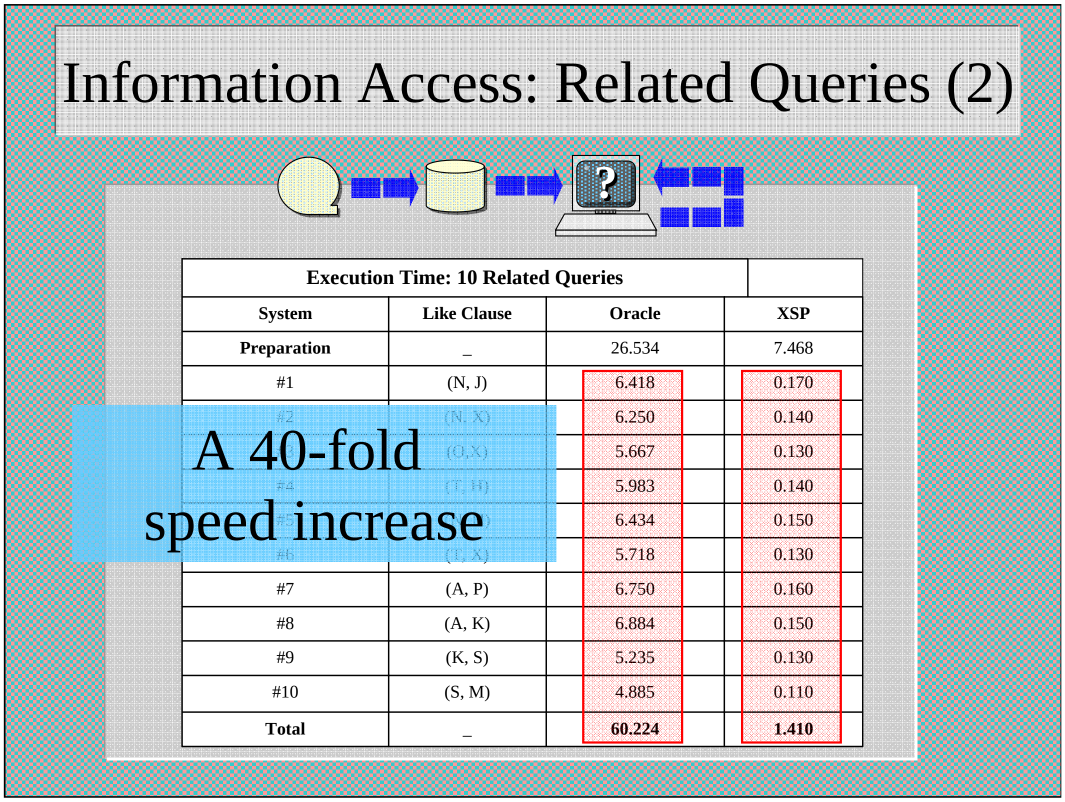


Information Access: Related Queries

Execution Time: 10 Related Queries			
System	**Like Clause**	**Oracle**	**XSP**
Preparation	_	26.534	7.468
#1	(N, J)	6.418	0.170
#2	(N, X)	6.250	0.140
#3	(O,X)	5.667	0.130
#4	(T, H)	5.983	0.140
#5	(N, H)	6.434	0.150
#6	(T, X)	5.718	0.130
#7	(A, P)	6.750	0.160
#8	(A, K)	6.884	0.150
#9	(K, S)	5.235	0.130
#10	(S, M)	4.885	0.110
Total	_	**86.758**	**8.878**

The test script in this case calls for loading the database and executing 10 related requests for information. The first such request is simply Q1 from the previous test. The remaining nine are identical to Q1 in all respects except for parameter substitutions. Column two of the table documents the parameter values that make each query in the suite unique.

Remark: The slight timing discrepancies between row #1 in the above table and column Q1 in the previous table are to be expected.

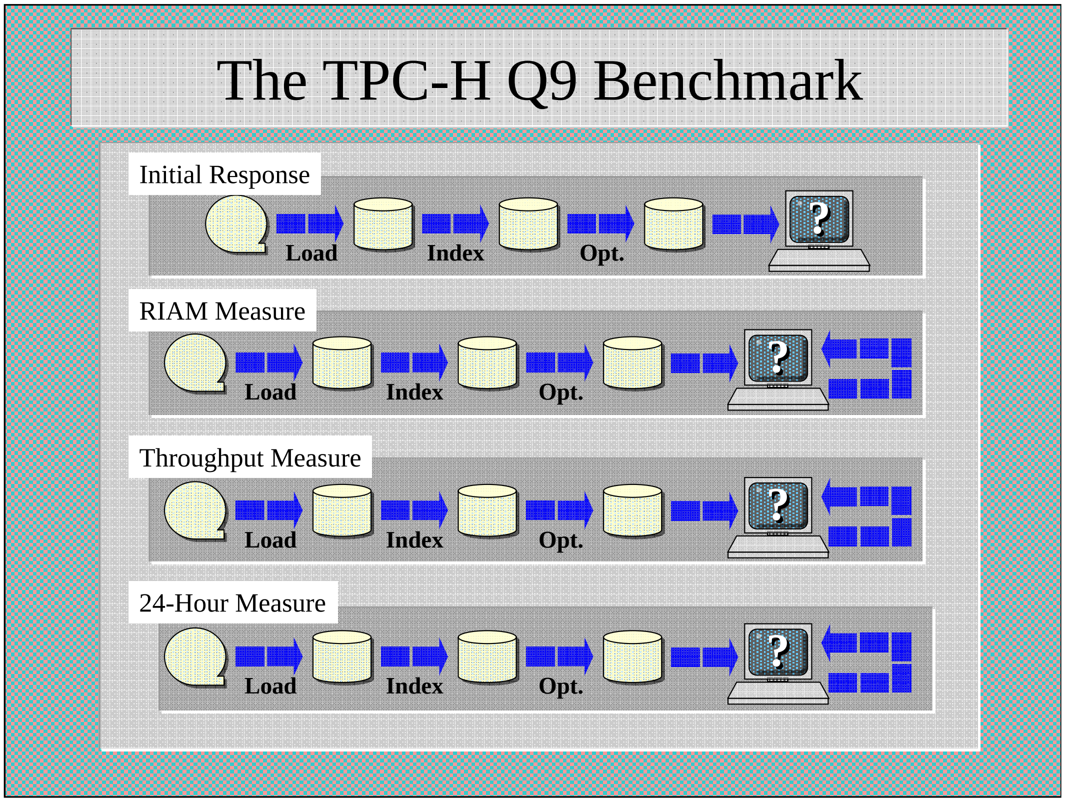


Information Access: Related Queries (2)

Execution Time: 10 Related Queries			
System	**Like Clause**	**Oracle**	**XSP**
Preparation	_	26.534	7.468
#1	(N, J)	6.418	0.170
#2	(N, X)	6.250	0.140
#3	(O,X)	5.667	0.130
#4	(T, H)	5.983	0.140
#5	[illegible]	6.434	0.150
#6	(T, X)	5.718	0.130
#7	(A, P)	6.750	0.160
#8	(A, K)	6.884	0.150
#9	(K, S)	5.235	0.130
#10	(S, M)	4.885	0.110
Total	_	**60.224**	**1.410**

A 40-fold speed increase

Oracle processed the 10 queries in 60.224 minutes. XSP, at 1.410 minutes, processed the suite more than 40 times faster.



The TPC-H Q9 Benchmark

Initial Response

Load Index Opt.

RIAM Measure

Load Index Opt.

Throughput Measure

Load Index Opt.

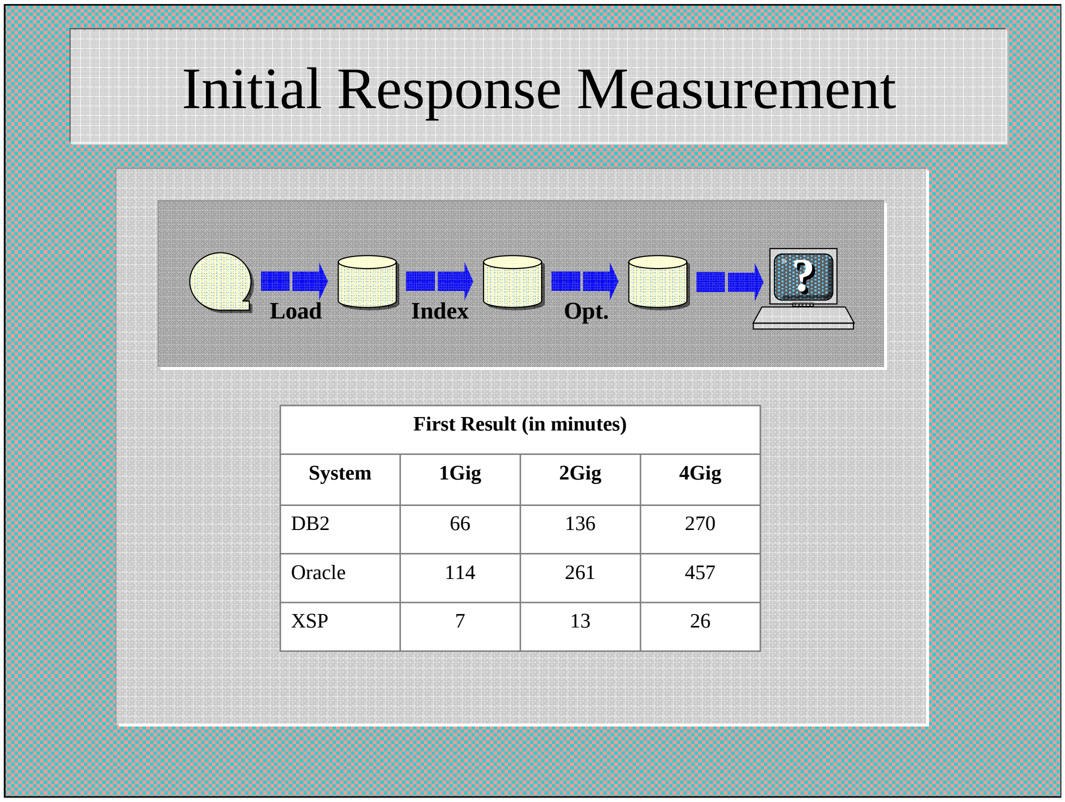
24-Hour Measure

Load Index Opt.

The TPC-H benchmark measures the ability of decision support systems to deal with large quantities of data, process complex SQL queries and respond to realistic business questions. Its Composite query-per-hour performance measure has become the preferred metric for comparing the relative merits of relational DBMSs. In our experiments we applied a slightly different metric. Instead of simply taking query response time into account, we factored database load time in as well.

The TPC consortium provides a program (DBGEN) that generates character-delimited interrelated sequential files. Of the 22 queries designed to exploit these relationships, the Product Type Profit Measure Query (Q9) is among the most difficult. It is not possible to respond to Q9, a six-way join, without performing a full-table scan of a very large table.

We conducted the tests on a 440 MHz Pentium with 256 Megabytes of RAM and 30 Gigabytes of external storage (Type: Western Digital WD300BB-32CCB0). The transfer rate of this unit, which is in the 16-18 Mb/sec range, is slow by modern standards. Today, high-end hard drives often deliver data at rates that exceed 100Mb/sec. The slow speed of the WD device is worth noting here because it penalizes XSP much more than it does either DB2 or Oracle.

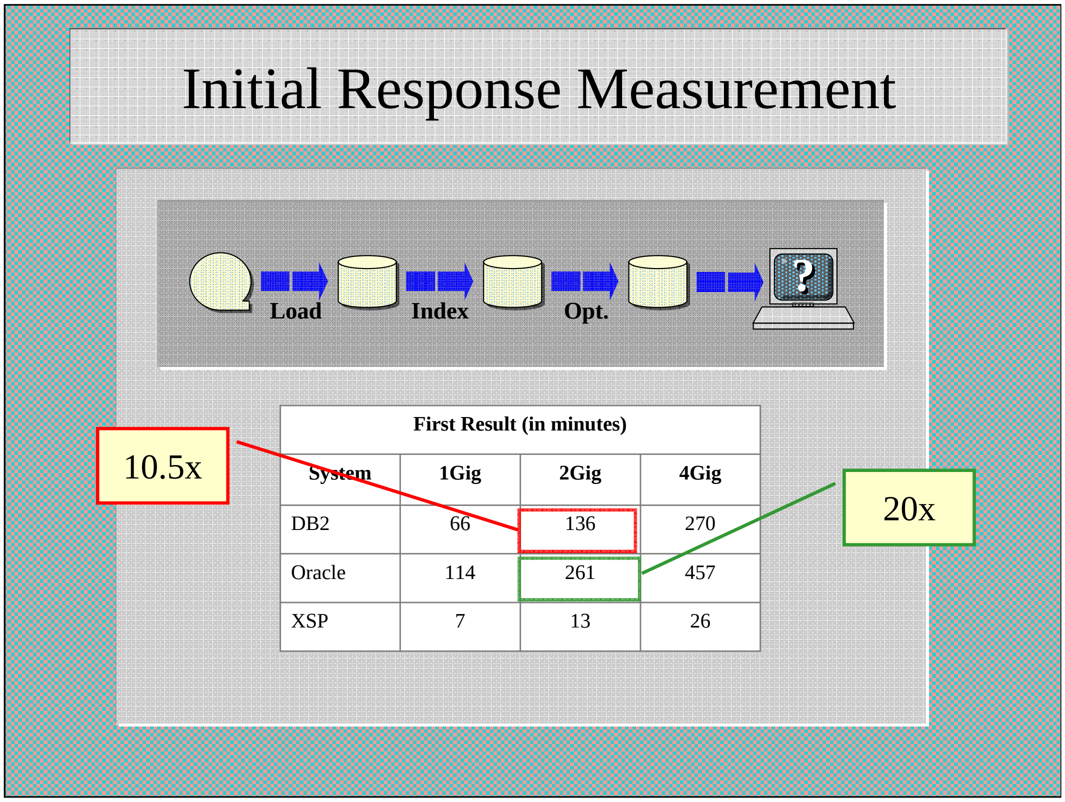


Initial Response Measurement

Load
Index
Opt.
?

First Result (in minutes)			
System	**1Gig**	**2Gig**	**4Gig**
DB2	66	136	270
Oracle	114	261	457
XSP	7	13	26

This table documents how long it took each system to load the database and produce an initial response to Q9.The raw data file ranged in size from 1 gigabyte to 4 gigabytes. Oracle was the slowest to respond because it took a very long time to load and optimize the data. It is worth noting that Oracle processed Q9 faster than DB2 (but not nearly as fast as XSP).



Initial Response Measurement

Load

Index

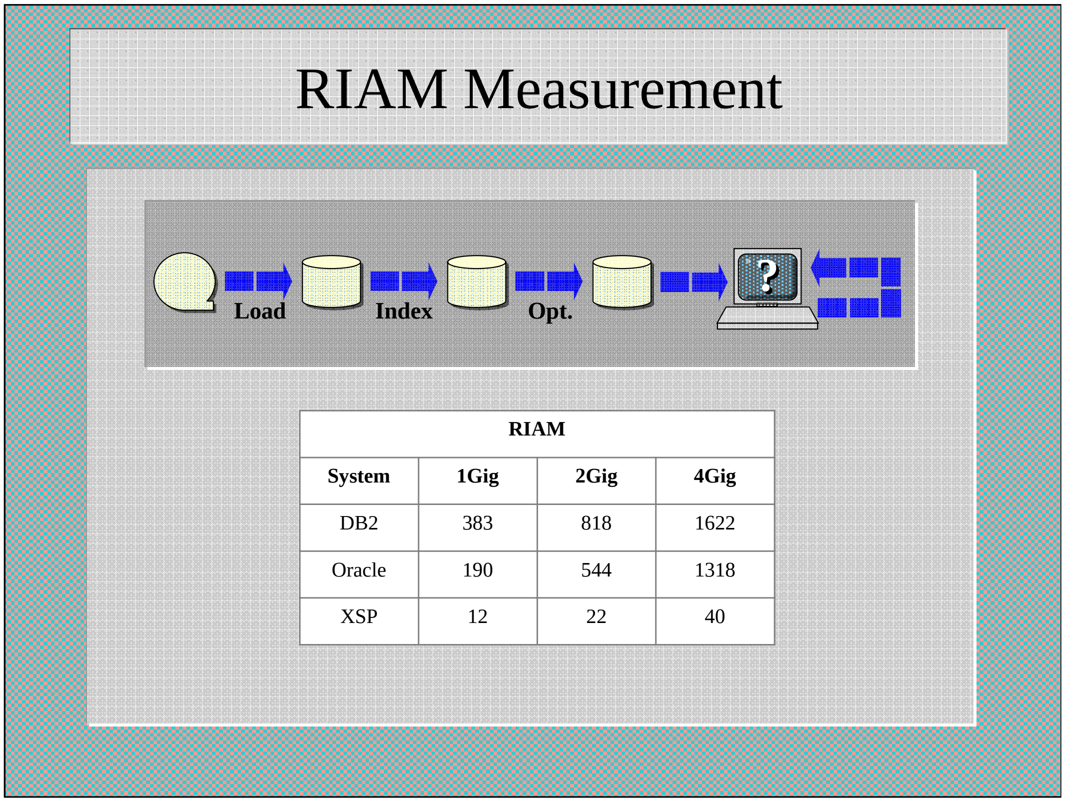
Opt.

First Result (in minutes)			
System	1Gig	2Gig	4Gig
DB2	66	136	270
Oracle	114	261	457
XSP	7	13	26

10.5x

20x

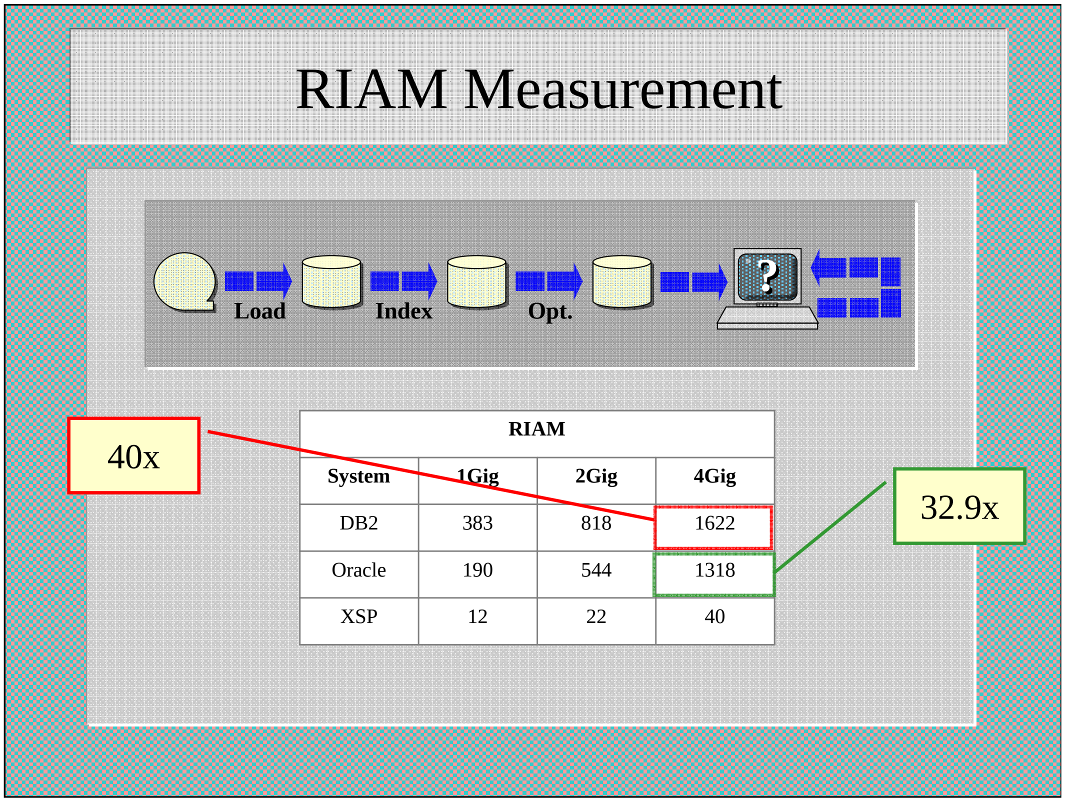
Note that XSP outperforms DB2 and Oracle in every case. In the two gigabyte test, for example, it generated a first response 10.5 faster than DB2 and 20 times faster than Oracle.



RIAM Measurement

RIAM			
System	**1Gig**	**2Gig**	**4Gig**
DB2	383	818	1622
Oracle	190	544	1318
XSP	12	22	40

RIAM, for Rapid Information Access Measure, tells how long it takes to (1) load and optimize a collection of raw data, (2) respond to a query for the first time and (3) respond to ten additional (parameterized) variations of that query. Here, Oracle's superior, but costly, optimization procedure enabled it to outperform DB2. Note that XSP consistently outperformed both Oracle and DB2.



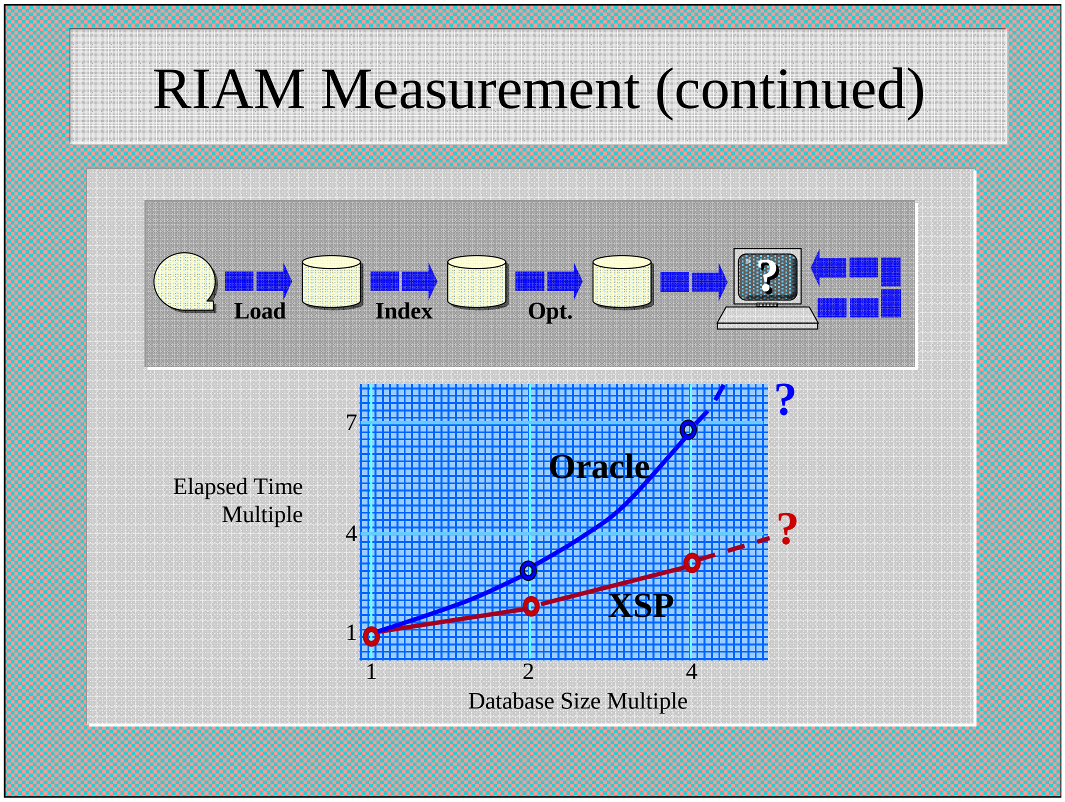
RIAM Measurement

RIAM			
System	1Gig	2Gig	4Gig
DB2	383	818	1622
Oracle	190	544	1318
XSP	12	22	40

40x

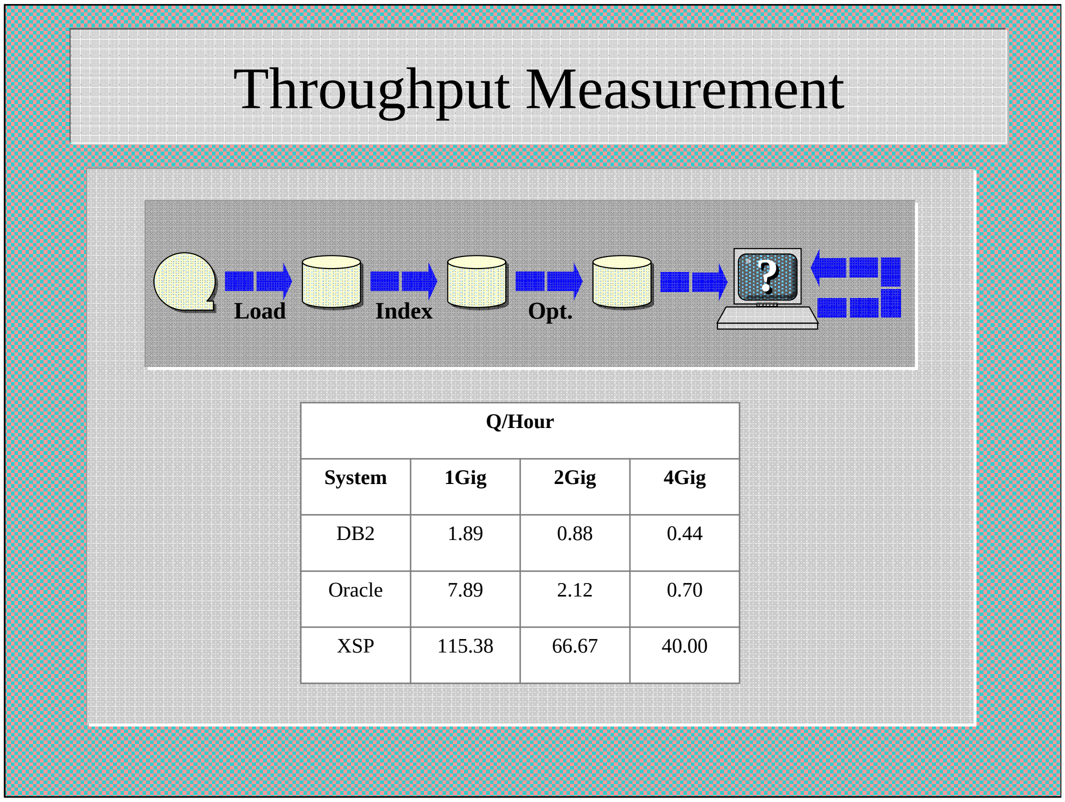
32.9x

XSP completed the 4 gigabyte RIAM test 40 times faster than DB2 and 32.9 times faster than Oracle.



RIAM Measurement (continued)

Note also that XSP is more scalable than either DB2 or Oracle. Oracle, for example, took nearly seven times longer to complete the test given a four-fold increase in file size. XSP took four times longer given a four-fold increase.

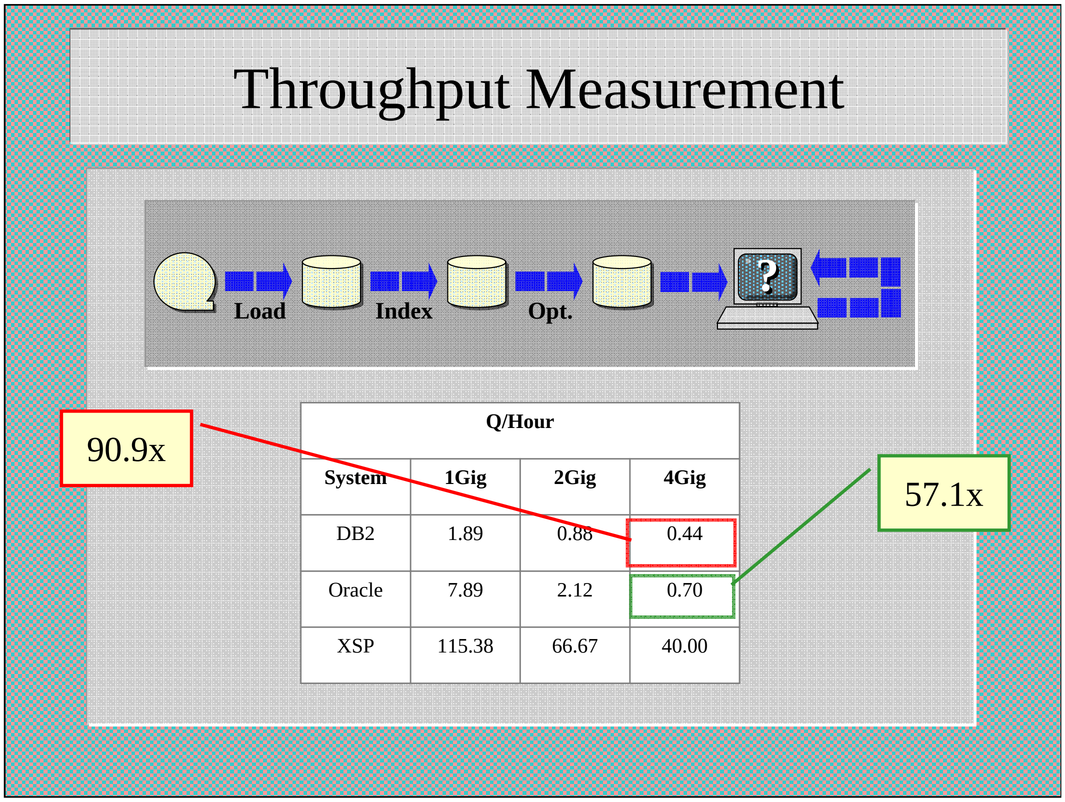


Throughput Measurement

Load
Index
Opt.
?

Q/Hour			
System	**1Gig**	**2Gig**	**4Gig**
DB2	1.89	0.88	0.44
Oracle	7.89	2.12	0.70
XSP	115.38	66.67	40.00

This table measures system throughput as a function of database size. Each cell represents the number of responses a system can deliver per processing hour under ideal conditions. The table does not take database load time and optimization overhead into account.



Throughput Measurement

Load

Index

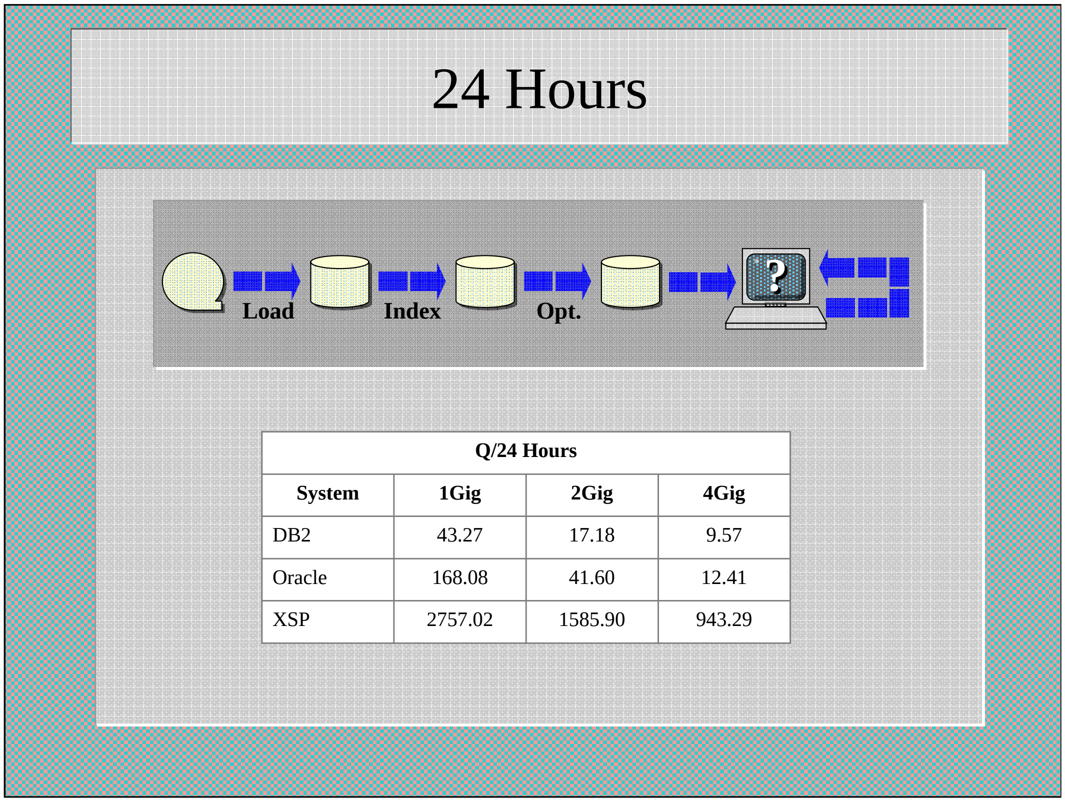
Opt.

90.9x

57.1x

Q/Hour			
System	1Gig	2Gig	4Gig
DB2	1.89	0.88	0.44
Oracle	7.89	2.12	0.70
XSP	115.38	66.67	40.00

XSP throughput exceeds that of DB2 by a factor of 90.9 and Oracle by a factor of 57.1 in the 4 gigabyte test.

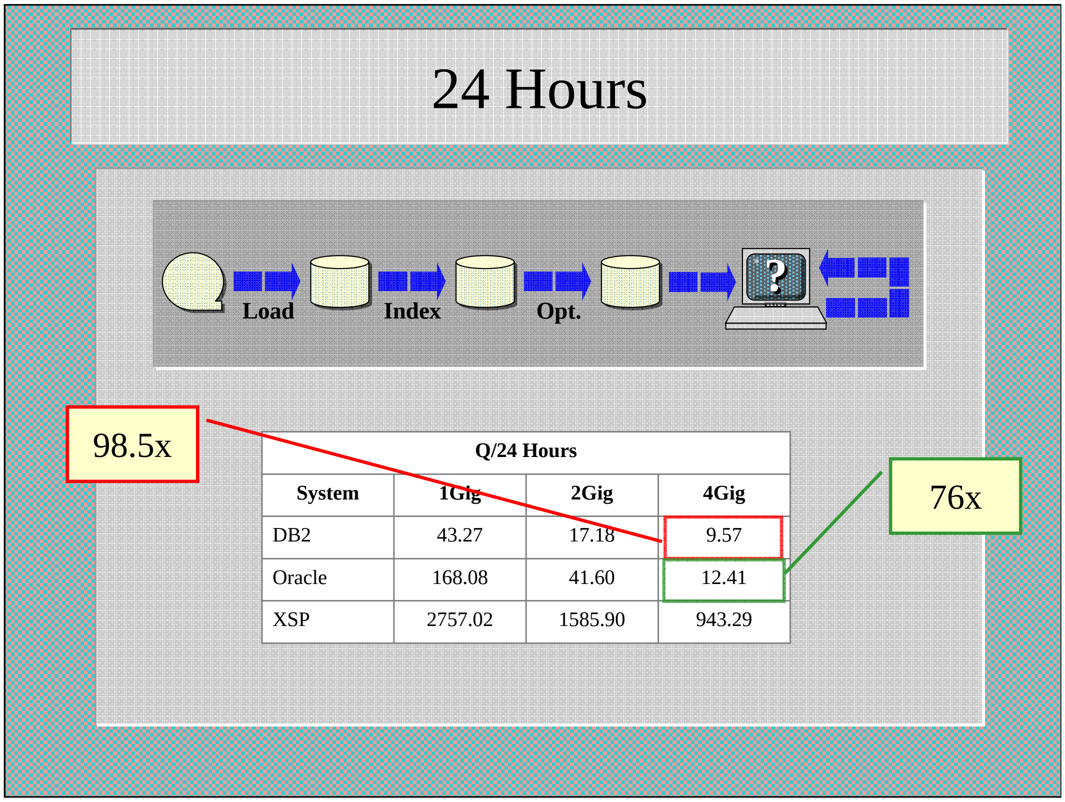


24 Hours

Load
Index
Opt.
?

Q/24 Hours			
System	**1Gig**	**2Gig**	**4Gig**
DB2	43.27	17.18	9.57
Oracle	168.08	41.60	12.41
XSP	2757.02	1585.90	943.29

This table projects the number of Q9 result sets the systems would be able to produce in a 24-hour period.



24 Hours

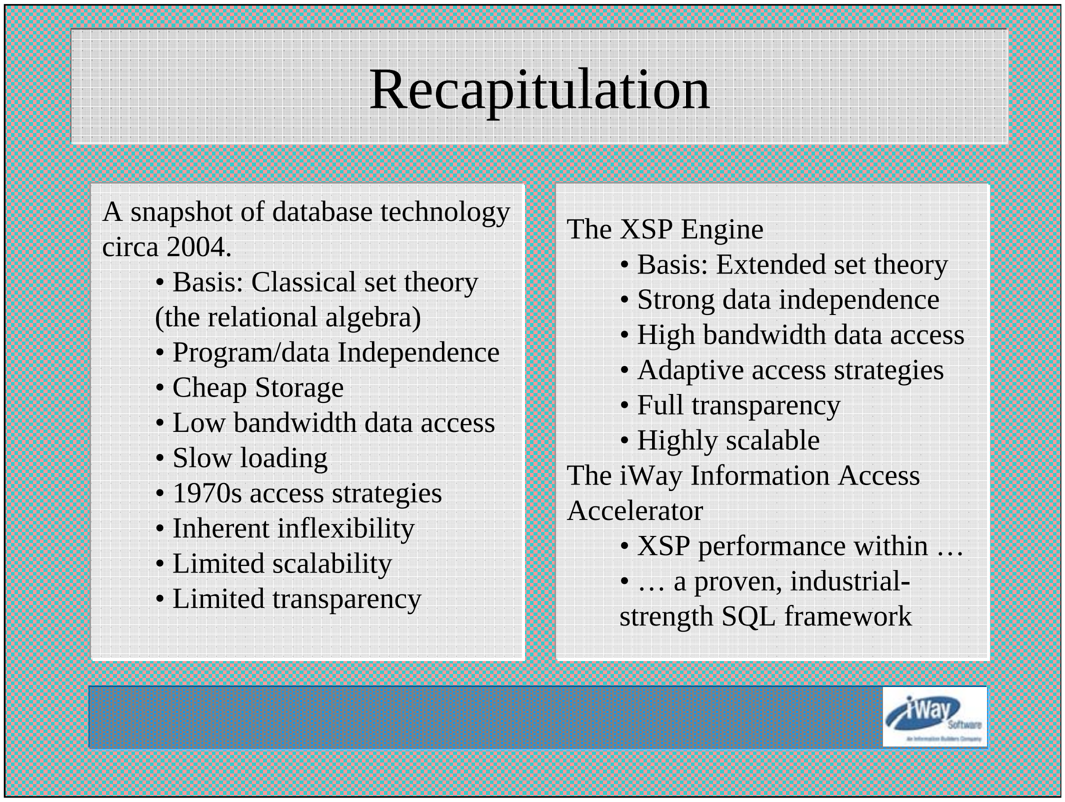
Load
Index
Opt.
?

98.5x

Q/24 Hours			
System	1Gig	2Gig	4Gig
DB2	43.27	17.18	9.57
Oracle	168.08	41.60	12.41
XSP	2757.02	1585.90	943.29

76x

Over a 24-hour period in the 4 gigabyte test, XSP generated more than 98 times as many responses as DB2 and 76 times as many responses as Oracle.



Recapitulation

A snapshot of database technology circa 2004.

- Basis: Classical set theory (the relational algebra)
- Program/data Independence
- Cheap Storage
- Low bandwidth data access
- Slow loading
- 1970s access strategies
- Inherent inflexibility
- Limited scalability
- Limited transparency

The XSP Engine

- Basis: Extended set theory
- Strong data independence
- High bandwidth data access
- Adaptive access strategies
- Full transparency
- Highly scalable

The iWay Information Access Accelerator

- XSP performance within …
- … a proven, industrial-strength SQL framework