Advanced Database Techniques 2010

Martin Kersten
Stefan Manegold
Fabian Groffen
Jenny Zhang

http://www.cwi.nl/~manegold/teaching/adt2010

Advanced Database Techniques

- Prerequisite knowledge
  - Linux, C
  - Relational data model
  - SQL
  - Relational algebra
  - Data structures (b-tree, hash)
  - Latex
- What is the practical experience?

Code bases

- Database management systems are BIG software systems
  - Oracle, SQL-server, DB2 >1 M lines
  - PostgreSQL 300K lines
  - MySQL 500 K lines
  - MonetDB 900K lines (300K *.mx)
  - SQLite 60K lines
- Programmer teams for DBMS kernels range from a few to a few hundred

Database sizes

- iPhone 7.000 records for audio
- SAP a catalog of > 12.000 relations
- Ebay simple click stream of 2.5 PetaByte
- Google ?? 1 M machines
- Physics (HEP) 1 petabyte/year
- Astronomy Skyserver

File Edit View Go Bookmarks Tools Help

http://cas.sdss.org/dr6/en/

Go

Getting Started Latest Headlines MonetDB, Query Pro... CWI

Sloan Digital Sky Survey / SkyServer

SDSS

Home | Tools | Schema | Projects | Astronomy | SDSS | Contact Us | Download | Site Search | Help

Welcome to the **DR6 site**!!
The Sixth Data Release is dedicated to **Jim Gray** for his fundamental contribution to the SDSS project and the extraordinary energy and passion he shared with everybody!

This website presents data from the Sloan Digital Sky Survey, a project to make a map of a large part of the universe. We would like to show you the beauty of the universe, and share with you our excitement as we build the largest map in the history of the world.

News

The site hosts data from **Data Release 6 (DR6)**. **What's new in DR6**, **what's new on this site**, and **known problems**.
More...

For Astronomers

A separate branch of this website for professional astronomers (English)

More...

SDSS is supported by

NSF NASA MEXT

Powered by Microsoft

Site Traffic
Privacy Policy

SkyServer Tools

- Famous places
- Get images
- Visual Tools
- Explore
- Search
- Object upload
- CasJobs

Science Projects

- Basic
- Advanced
- Challenges
- For Kids
- Games and Contests
- Teachers
- Links to other projects

Info Links

- About Astronomy
- About the SDSS
- About the SkyServer
- SDSS Data Release 6
- SDSS Project Website
- Open SkyQuery
- Images of RC3 Galaxies

Help

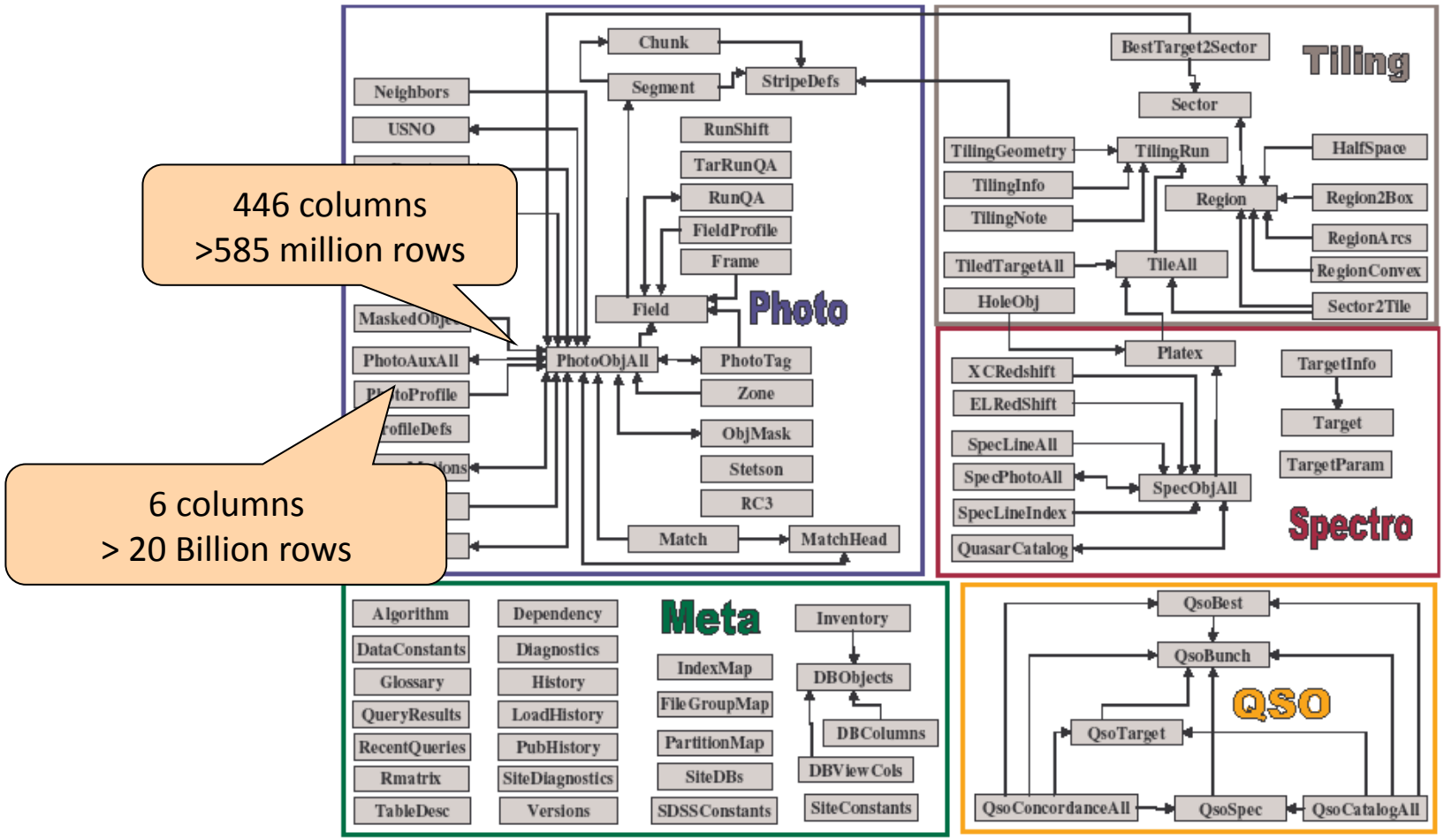
- Getting Started
- FAQ
- How To
- Glossary
- Schema Browser
- Sample SQL Queries
- Details of SDSS Data




Contact Us

Done

SkyServer Schema



Database Challenges

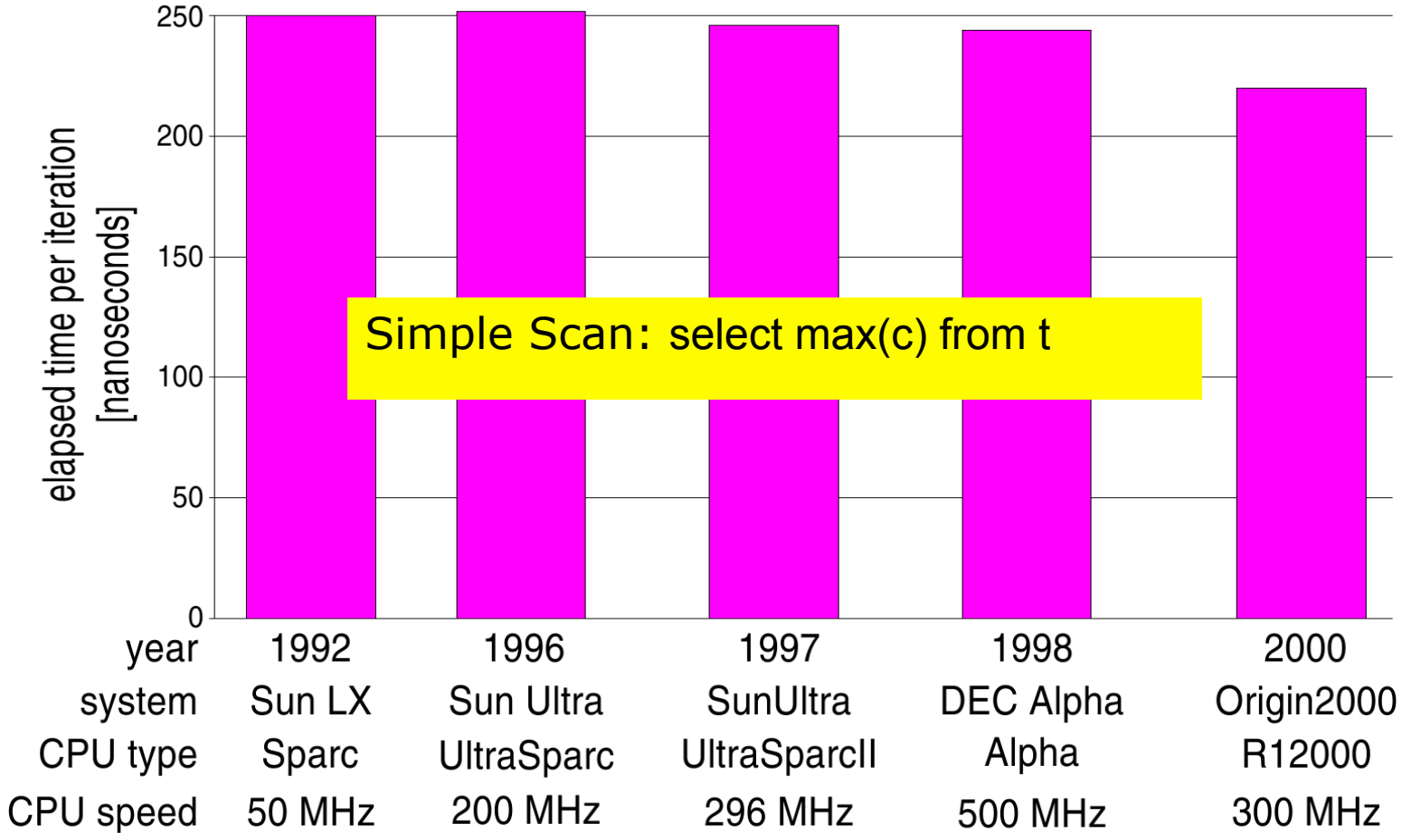
- Scale to very large database on very large distributed platforms

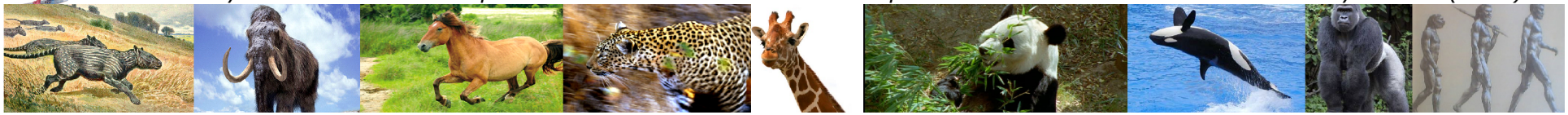
- Use the hardware in an optimal way

Mammals Flourished long before Dinosaurs became Extinct

S. Manegold, P. Boncz, M. Kersten

Evolution == Progress?

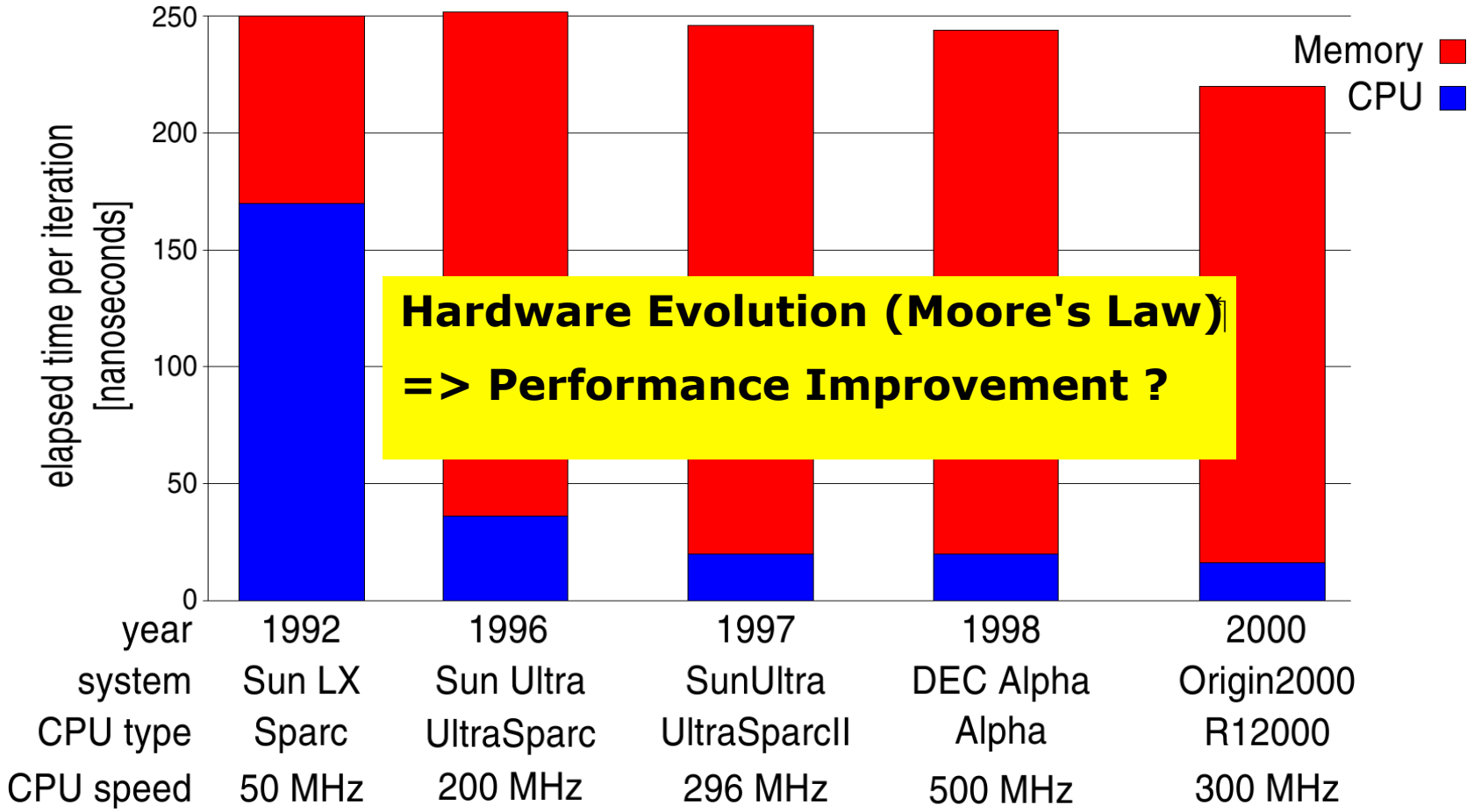


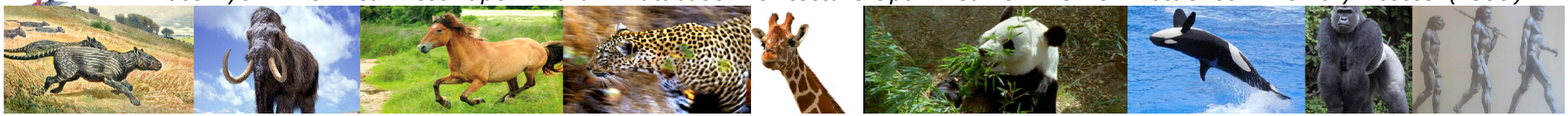


Mammals Flourished long before Dinosaurs became Extinct

S. Manegold, P. Boncz, M. Kersten

Evolution == Progress?





Mammals Flourished long before Dinosaurs became Extinct

S. Manegold, P. Boncz, M. Kersten

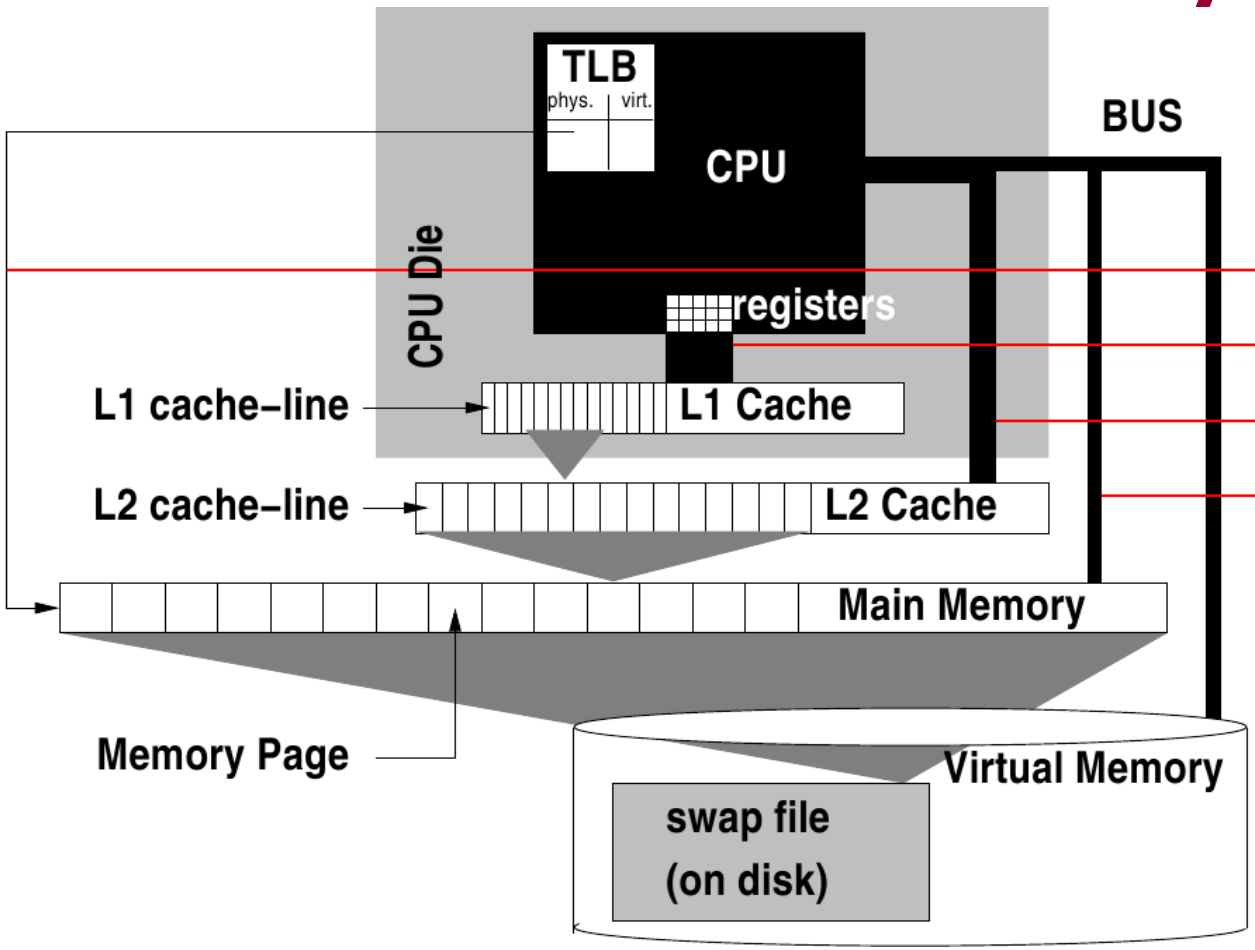
Databases hit The Memory Wall

- Detailed and exhaustive analysis for different workloads using 4 RDBMSs by Anastassia Ailamaki et al. "DBMSsOn A Modern Processor: Where Does Time Go?" (VLDB 1999)
- CPU is 50%-90% idle, waiting for memory:
  - L1 data stalls
  - L1 instruction stalls
  - L2 data stalls
  - TLB stalls
  - Branch mispredictions
  - Resource stalls

Mammals Flourished long before Dinosaurs became Extinct

S. Manegold, P. Boncz, M. Kersten

CPU & Hierarchical Memory System (1999)



Latencies:

TLB miss: 5–60 cycles

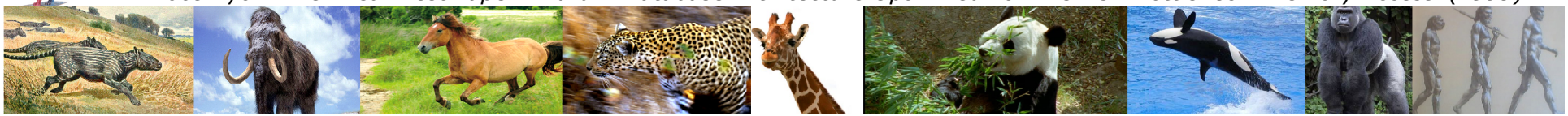
L1 hit: 1–2 cycles

L1 miss: 6–20 cycles

L2 miss:40–100 cycles

2009:

- **L2 (+L3) on CPU die**
- **Memory access: up to 1000 cycles**



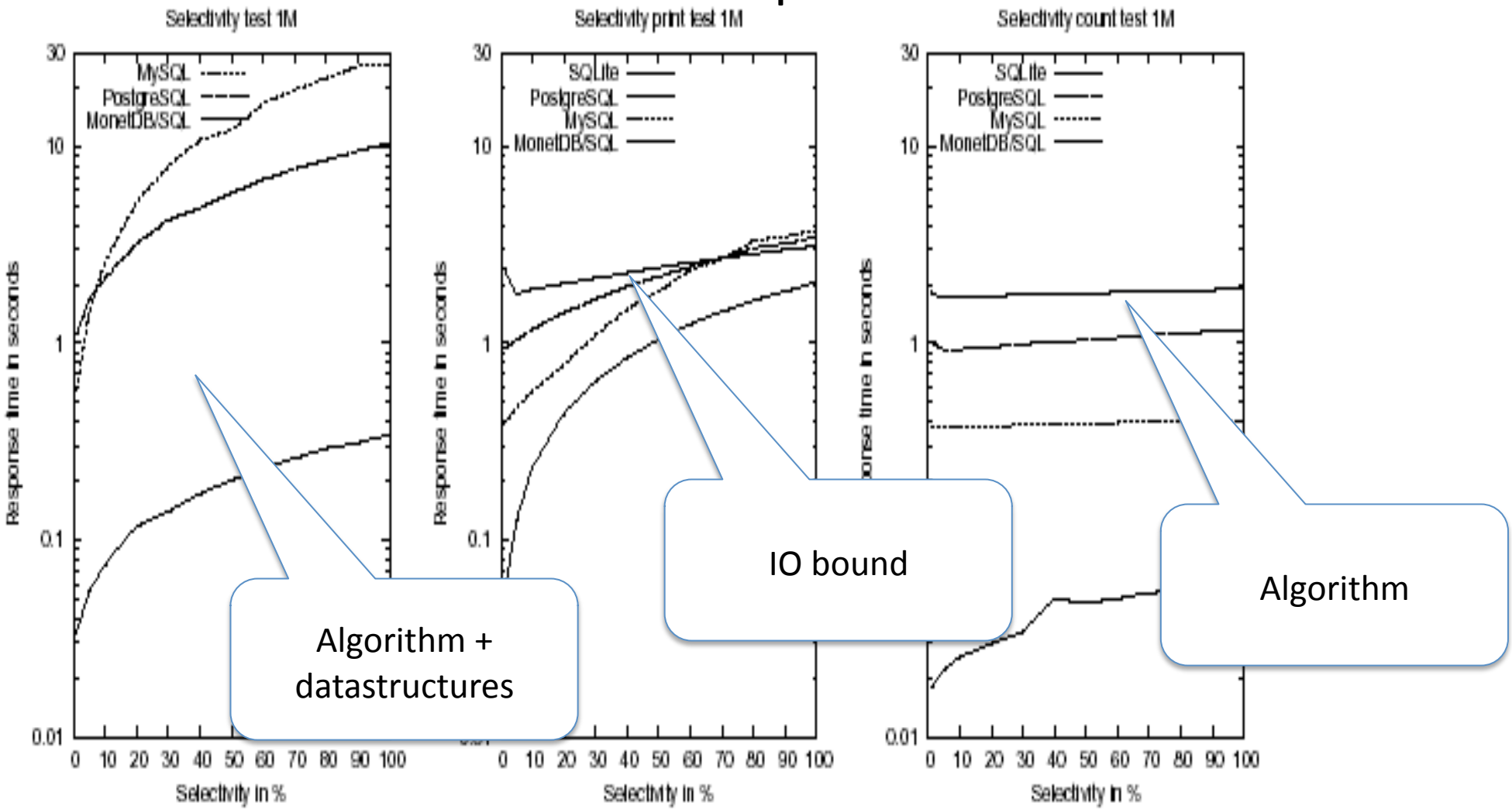
Performance components

- Hardware platform
- Data structures
- Algebraic optimizer
- SQL parser
- Application code
  - What is the total cost of execution ?
  - How many tasks can be performed/minute ?
  - How good is the optimizer?
  - What is the overhead of the datastructures ?

Not all are equal

	MonetDB	PostgreSQL	MySQL	SQLite	SQLlite nosync
1000 inserts transactions	0.27	4.30	0.15	13.06	0.22
25000 inserts 1 transaction	6.71	4.91	2.18	0.94	1.42
100 range selects	0.18	3.62	2.76	2.49	2.52
100 string range selects	2.15	13.40	4.64	3.36	3.37
5000 range index selects	5.22	4.61	1.27	1.12	1.16
1000 updates	0.43	1.73	8.41	0.63	0.63
25000 updates with index	8.33	18.79	8.13	3.52	3.10
25000 updates on text	10.32	48.13	6.98	2.40	1.72
Insert from select	0.65	61.36	1.53	2.78	1.59
Delete on text index	0.32	1.50	0.97	4.00	0.56
Delete with index	0.22	1.31	2.26	2.06	0.75
Big insert after delete	0.36	13.16	1.81	3.21	1.48
Big delete and small insert	0.93	4.55	1.70	0.61	0.40

Not all are equal



MonetDB experiment

- A single 20K length session interaction with V4.3.13 and V5 using the same Perl DBI application

clients	V4	V4/throughput	V5	V5/throughput
1	19.1	1047/sec	18.3	1092/sec
2	29.9	1339/sec	24.8	1612/sec
4	59.1	1353/sec	55.9	1431/sec
8	120	1333/sec	101	1584/sec

Monet experiment

- To remove the overhead of the Perl application experiment was also ran with a C -mapi implementation.

clients	V4	V4/throughput	V5	V5/throughput
1	3.2	6250/sec	3.0	6666/sec
2	5.1	7843/sec	4.2	9523/sec
4	17.1	4678/sec	8.1	9876/sec
8	35	4[illegible]/sec	16.2	9876/sec

CONSIDER ALL COMPONENTS IN YOUR EVALUATION

Gaining insight

- Study the code base (inspection+profiling)
  - Often not accessible outside development lab
- Study individual techniques (data structures+simulation)
  - Focus of most PhD research in DBMS
    - Detailed knowledge becomes available, but ignores the total cost of execution.
- Study as a black box
  - Develop a representative application framework
    - Benchmarks !

Performance Benchmarks

- Suites of tasks used to quantify the performance of software systems
- Important in comparing database systems, especially as systems become more standards compliant.

Performance Benchmarks

- Commonly used performance measures:
  - **Response time** (delay from submission of transaction to return of result)
  - **Throughput** (transactions per second, or tps)
  - **Availability** or mean time to failure

  - **Speedup** (linear->twice as much resources reduces time half)
  - **Scaleup** (response time remains constant with increasing load and resources)
  - **Sizeup** (doubling the size does not double required resources)

Benchmark design

- Benchmark suite structures
  - Simple, one shot experiment
    - time to set-up a connection with a db server
    - Selection processing for multiple selectivity factors
  - Complex, multi-target experiments
    - Geared at supporting a particular domain
    - To study the behavior of the DBMS software

Benchmark Design

- Multi-target benchmark components:
  - An application context
  - A database schema
  - A database content
  - A query workload

- These components can be fixed upfront or be generated dynamically

Benchmark design

- The key question for any benchmark
  - The ratio for its design
  - Its ability to differentiate systems
  - Its ability to highlight problematic areas
  - Its repeatability on multiple platforms

Wisconsin benchmark

- Designed in 1981 to study the query performance of database algorithms on a single user system
- Used extensively over the last 20 years to assess maturity of a kernel
- The results published caused legal problems for the authors

Wisconsin Benchmark

- Wisconsin Benchmark components
  - Single schema
  - Relations: ONEKTUP, TENKTUP1,TENKTUP2
  - Workload: 32 simple SQL queries
  - Metric: response time

- Key design issue is to be able to predict the outcome of a query
  - DBMS testing; optimizer cost-model design

Wisconsin Benchmark

```
CREATE TABLE TENKTUP1
(  unique1 integer NOT NULL,
   unique2 integer NOT NULL PRIMARY KEY,
   two integer NOT NULL,
   four integer NOT NULL,
   ten integer NOT NULL,
   twenty integer NOT NULL,
   hundred integer NOT NULL,
   thousand integer NOT NULL,
   twothous integer NOT NULL,
   fivethous integer NOT NULL,
   tenthous integer NOT NULL,
   odd100 integer NOT NULL,
   even100 integer NOT NULL,
   stringu1 char(52) NOT NULL,
   stringu2 char(52) NOT NULL,
   string4 char(52) NOT NULL
)
```

Sort order, clustering

unique1 0-9999 random candidate key

unique2 0-9999 random declared key

Secondary index

Wisconsin Benchmark

```
CREATE TABLE TENKTUP1
(  unique1 integer NOT NULL,
   unique2 integer NOT NULL PRIMARY KEY,
   two integer NOT NULL,
   four integer NOT NULL,
   ten integer NOT NULL,
   twenty integer NOT NULL,
   hundred integer NOT NULL,
   thousand integer NOT NULL,
   twothous integer NOT NULL,
   fivethous integer NOT NULL,
   tenthous integer NOT NULL,
   odd100 integer NOT NULL,
   even100 integer NOT NULL,
   stringu1 char(52) NOT NULL,
   stringu2 char(52) NOT NULL,
   string4 char(52) NOT NULL
)
```

Cyclic numbers, e.g.
0,1,2,3,4,0,1,2,4,0….

Selectivity control
Aggregation
Non-clustered index

Wisconsin Benchmark

```
CREATE TABLE TENKTUP1
(  unique1 integer NOT NULL,
   unique2 integer NOT NULL PRIMARY KEY,
   two integer NOT NULL,
   four integer NOT NULL,
   ten integer NOT NULL,
   twenty integer NOT NULL,
   hundred integer NOT NULL,
   thousand integer NOT NULL,
   twothous integer NOT NULL,
   fivethous integer NOT NULL,
   tenthous integer NOT NULL,
   odd100 integer NOT NULL,
   even100 integer NOT NULL,
   stringu1 char(52) NOT NULL,
   stringu2 char(52) NOT NULL,
   string4 char(52) NOT NULL
)
```

50 groups, 2% each
Cyclic assigned

Wisconsin Benchmark

```
CREATE TABLE TENKTUP1
(  unique1 integer NOT NULL,
   unique2 integer NOT NULL PRIMARY KEY,
   two integer NOT NULL,
   four integer NOT NULL,
   ten integer NOT NULL,
   twenty integer NOT NULL,
   hundred integer NOT NULL,
   thousand integer NOT NULL,
   twothous integer NOT NULL,
   fivethous integer NOT NULL,
   tenthous integer NOT NULL,
   odd100 integer NOT NULL,
   even100 integer NOT NULL,
   stringu1 char(52) NOT NULL,
   stringu2 char(52) NOT NULL,
   string4 char(52) NOT NULL
)
```

Strings 52 chars long
$xxx..25..xxx$xxx..25..xxx$

$ is replaced by A-Z
Stringu1, stringu2 are keys
String4 contains 4 different

Wisconsin Benchmark

- Comments on old database structure
  - Tuple size (203 bytes) dictated by the page size
  - Relation size dictated by low memory, e.g. a 2 megabyte database was almost a complete disk

- Redesign and scaling up
  - Relation size increased to 100K and beyond
  - Cyclic values -> random to generate more realistic distribution
  - Strings start with 7 different char from A-Z

Wisconsin Benchmark

- Query benchmark suite aimed at performance of
  - Selection with different selectivity values
  - Projection with different percentage of duplicates
  - Single and multiple joins
  - Simple aggregates and aggregate functions
  - Append, delete, modify

- Queries may use (clustered) index

Wisconsin Benchmark

The speed at which a database system can process a selection operation depends on a number of factors including:

1) The storage organization of the relation.
2) The selectivity factor of the predicate.
3) The hardware speed and the quality of the software.
4) The output mode of the query.

Wisconsin Benchmark

The selection queries in the Wisconsin benchmark explore the effect of each and the impact of three different storage organizations :

1) Sequential (heap) organization.

2) Primary clustered index on the unique2 attribute. (Relation is sorted on unique2 attribute)

3) Secondary, dense, non-clustered indices on the unique1 and *onePercent* attributes.

Wisconsin Benchmark

Query 1 (no index) & **Query 3** (clustered index) - 1% selection

```
INSERT INTO TMP
SELECT * FROM TENKTUP1
WHERE unique2 BETWEEN 0 AND 99
```

Query 2 (no index) & **Query 4** (clustered index) - 10% selection

```
INSERT INTO TMP
SELECT * FROM TENKTUP1
WHERE unique2 BETWEEN 792 AND 1791
```

Wisconsin Benchmark

Query 5 - 1% selection via a non-clustered index

```
INSERT INTO TMP
SELECT * FROM TENKTUP1
WHERE unique1 BETWEEN 0 AND 99
```

Query 6 - 10% selection via a non-clustered index

```
INSERT INTO TMP
SELECT * FROM TENKTUP1
WHERE unique1 BETWEEN 792 AND 1791
```

Wisconsin Benchmark

The join queries in the benchmark were designed to study the effect of three different factors:

1) The impact of the complexity of a query on the relative performance of the different database systems.

2) The performance of the join algorithms used by the different systems.

3) The effectiveness of the query optimizers on complex queries.

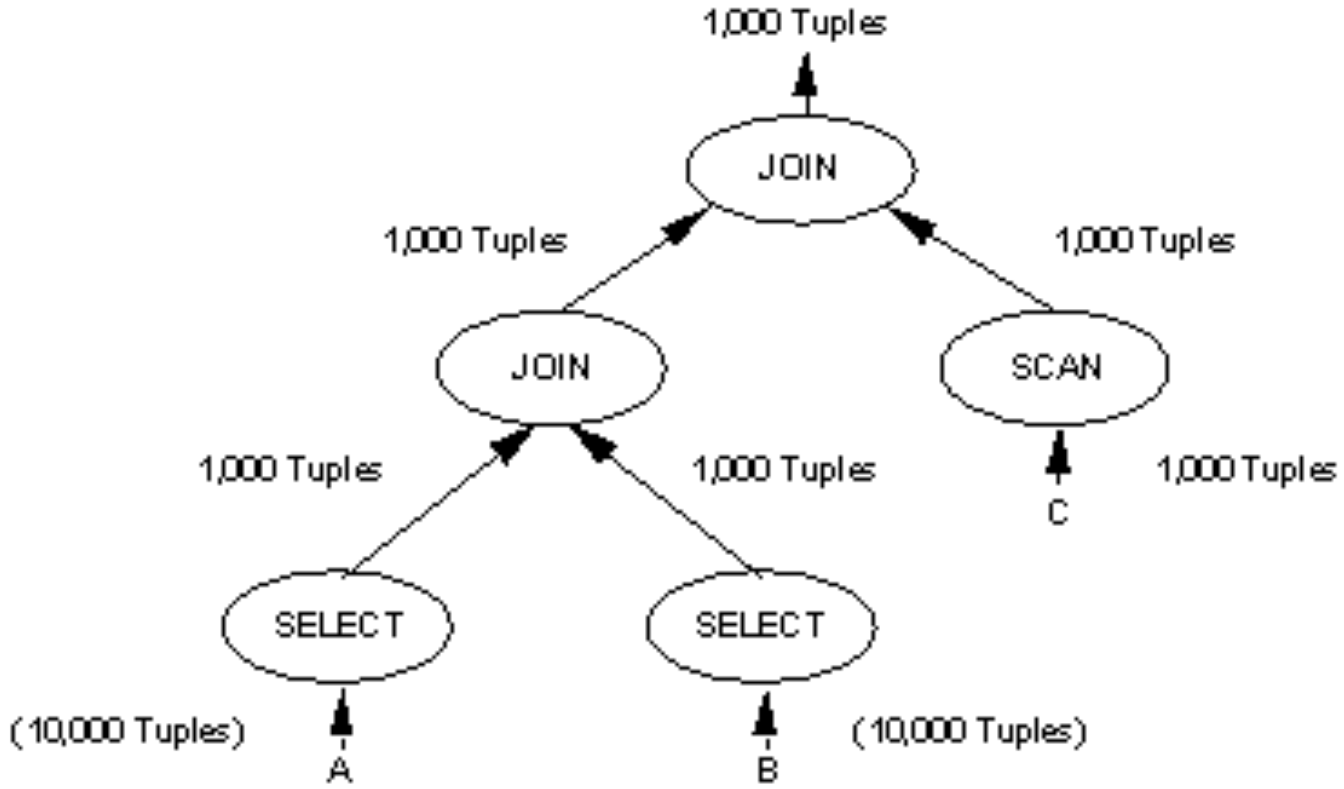
Wisconsin Benchmark

JoinABprime - a simple join of relations A and Bprime where the cardinality of the Bprime relation is 10% that of the A relation.

JoinASelB - this query is composed of one join and one selection. A and B have the same number of tuples. The selection on B has a 10% selectivity factor, reducing B to the size of the Bprime relation in the JoinABprime query. The result relation for this query has the same number of tuples as the corresponding JoinABprime query.

Wisconsin Benchmark

JoinCselASelB



Wisconsin Benchmark

Query 9 (no index) and **Query 12** (clustered index) - JoinAselB

```
INSERT INTO TMP
SELECT * FROM TENKTUP1, TENKTUP2
WHERE (TENKTUP1.unique2 = TENKTUP2.unique2)
AND (TENKTUP2.unique2 < 1000)
```

Query to make Bprime relation

```
INSERT INTO BPRIME
SELECT * FROM TENKTUP2
WHERE TENKTUP2.unique2 < 1000
```

Wisconsin Benchmark

Query 16 (non-clustered index) - JoinABprime

```
INSERT INTO TMP
SELECT * FROM TENKTUP1, BPRIME
WHERE (TENKTUP1.unique1 = BPRIME.unique1)
```

Query 17 (non-clustered index) - JoinCselAselB

```
INSERT INTO TMP
SELECT * FROM ONEKTUP, TENKTUP1
WHERE (ONEKTUP.unique1 = TENKTUP1.unique1)
AND (TENKTUP1.unique1 = TENKTUP2.unique1)
AND (TENKTUP1.unique1 < 1000)
```

Wisconsin benchmark

Implementation of the projection operation is normally done in two phases in the general case.

- First a pass is made through the source relation to discard unwanted attributes.
- A second phase is necessary in to eliminate any duplicate tuples that may have been introduced as a side effect of the first phase (i.e. elimination of an attribute which is the key or some part of the key).

Wisconsin Benchmark

Query 18 - Projection with 1% Projection

```
INSERT INTO TMP
SELECT DISTINCT two, four, ten, twenty, onePercent, string4
FROM TENKTUP1
```

Query 19 - Projection with 100% Projection

```
INSERT INTO TMP
SELECT DISTINCT two, four, ten, twenty, onePercent, tenPercent,
    twentyPercent, fiftyPercent, unique3, evenOnePercent,
    oddOnePercent, stringu1, stringu2, string4
FROM TENKTUP1
```

Performance Benchmarks (Cont.)

- Beware when computing average throughput of different transaction types
  - E.g., suppose a system runs transaction type A at 99 tps and transaction type B at 1 tps.
  - Given an equal mixture of types A and B, throughput is **not** (99+1)/2 = 50 tps.
  - Running one transaction of each type takes time 1+.01 seconds, giving a throughput of 1.98 tps (= 2/1.01).
  - **Interference** (e.g. lock contention) makes even this incorrect if different transaction types run concurrently

Metric Selections

- Criteria
  - Easy to explain in mathematical terms to users
  - Non-hypersensitivity
  - Scalability translates to easy change of metric
  - Balance to capture delta changes in outlier positions
  - Easy translation to operational decisions

Metric Selections

- Arithmetic mean $\bar{x} = \frac{x_1 + x_2 + \ldots + x_n}{n}$
- Geometric mean $g = \sqrt[n]{x_1 x_2 \ldots x_n}$
- Harmonic mean $\frac{1}{H} \equiv \frac{1}{n} \sum_{i=1}^{n} \frac{1}{x_i}.$

Metric Selections

- Arithmetic mean

$$\bar{x} = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

$$g = \sqrt[n]{x_1 x_2 \ldots x_n}$$

- Geometric mean

$$\log g = \frac{\log x_1 + \log x_2 + \ldots + \log x_n}{n}$$

Metric Selections

- Geometric mean $g = \sqrt[n]{x_1 x_2 \ldots x_n}$

$$\log g = \frac{\log x_1 + \log x_2 + \ldots + \log x_n}{n}$$

$$\log (g_f + f) = \frac{\log (x_1 + f) + \log (x_2 + f) + \ldots + \log (x_n + f)}{n}$$

Database Application Classes

- **Online transaction processing (OLTP)**
  - requires high concurrency and clever techniques to speed up commit processing, to support a high rate of update transactions.
- **Decision support applications**
  - including online analytical processing, or OLAP applications, require good query evaluation algorithms and query optimization.
- **Embedded applications**
  - Requires small footprint, small database storage

Benchmarks Suites

- The Transaction Processing Council (www.tpc.org) benchmark suites are widely used.
  - **TPC-A** and **TPC-B**: simple OLTP application modeling a bank teller application with and without communication
    - Not used anymore
  - **TPC-C**: complex OLTP application modeling an inventory system
    - Current standard for OLTP benchmarking

Benchmarks Suites (Cont.)

- TPC benchmarks (cont.)
  - **TPC-D**: complex decision support application
    - Superceded by TPC-H and TPC-R
  - **TPC-H:** (H for ad hoc)
    - Models ad hoc queries which are not known beforehand
      - Total of 22 queries with emphasis on aggregation
    - prohibits materialized views
    - permits indices only on primary and foreign keys
  - **TPC-R:** (R for reporting) same as TPC-H, but without any restrictions on materialized views and indices
  - **TPC-W**: (W for Web) End-to-end Web service benchmark modeling a Web bookstore, with combination of static and dynamically generated pages

TPC Performance Measures

- TPC performance measures
  - **transactions-per-second** with specified constraints on response time
  - **transactions-per-second-per-dollar** accounts for cost of owning system
- TPC benchmark requires database sizes to be scaled up with increasing transactions-per-second
  - reflects real world applications where more customers means more database size and more transactions-per-second
- External audit of TPC performance numbers mandatory
  - TPC performance claims can be trusted

TPC Performance Measures

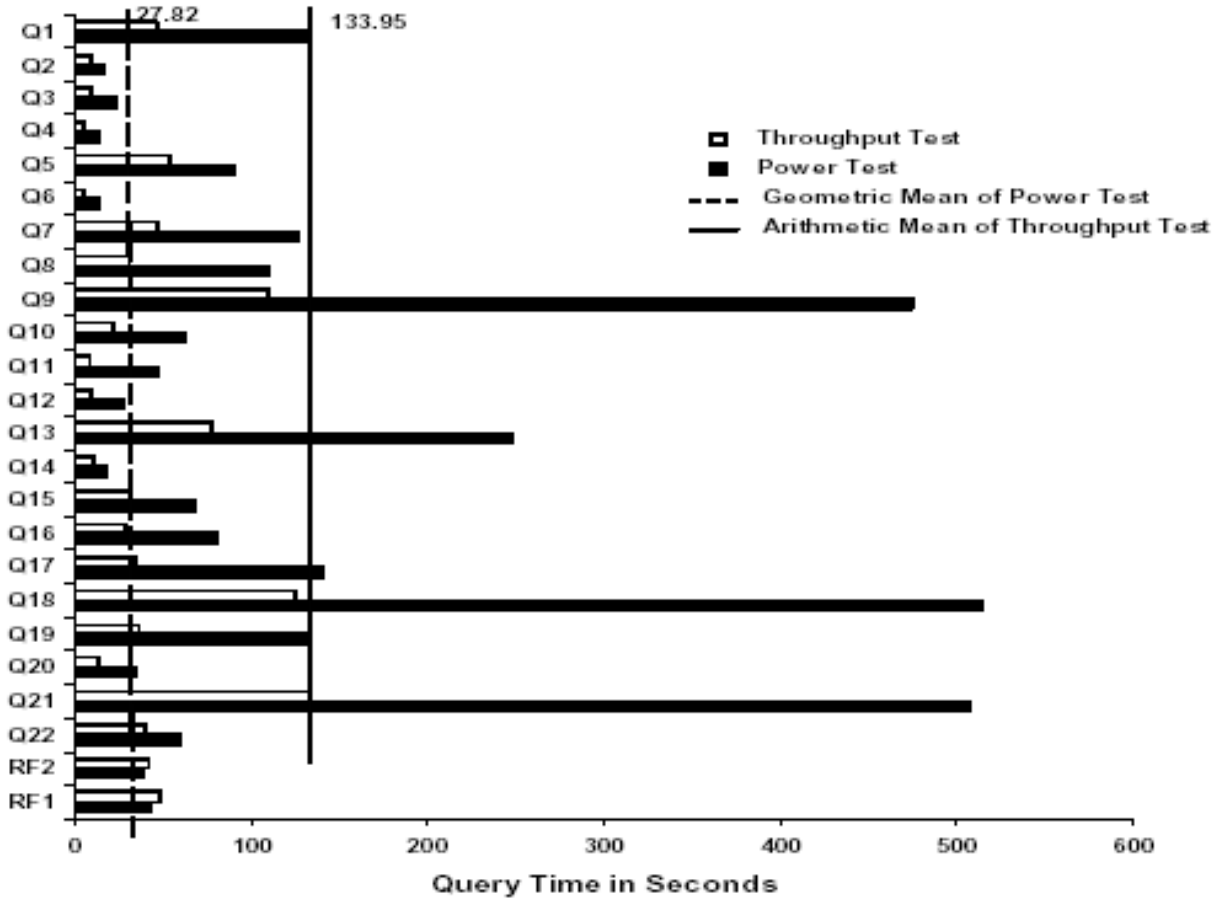
- Two types of tests for TPC-H and TPC-R
  - **Power test**: runs queries and updates sequentially, then takes mean to find queries per hour
  - **Throughput test**: runs queries and updates concurrently
    - multiple streams running in parallel each generates queries, with one parallel update stream
  - **Composite query per hour metric**: square root of product of power and throughput metrics
  - **Composite price/performance metric**

100 GB Results

Rank	Company	System	QphH	Price/QphH	System Availability	Database	Operating System	Date Submitted	Cluster
1	IBM	IBM eServer 325	12,216	71 US $	11/08/03	IBM DB2 UDB 8.1	Suse Linux Enterprise Server 8	07/29/03	Y
2	IBM	IBM eServer xSeries 445 8P	5,602	73 US $	12/31/03	IBM DB2 UDB 8.1	Microsoft Windows Server 2003 Enterprise Edition	06/30/03	N
3	MAXDATA	MAXDATA Platinum 9000-4R	4,307	70 Euros	11/08/03	IBM DB2 UDB 8.1	Microsoft Windows Server 2003 Enterprise Edition	07/03/03	N
4	hp invent	HP ProLiant DL 760 G2 8P	4,224	43 US $	08/15/03	Microsoft SQL Server 2000 Enterprise Edition	Microsoft Windows Server 2003 Enterprise Edition	08/15/03	N
5	IBM	IBM eServer xSeries 440	3,861	102 US $	07/15/03	IBM DB2 UDB 8.1	Microsoft Windows Server 2003 Enterprise Edition	04/22/03	N
6	hp invent	HP Proliant DL760 G2 8P	3,346	65 US $	04/24/03	Microsoft SQL Server 2000 Enterprise Edition	Microsoft Windows Server 2003 Enterprise Server	02/28/03	N
7	IBM	IBM eServer xSeries 440	3,342	131 US $	05/15/03	IBM DB2 UDB 8.1	Microsoft Windows Server 2003 Enterprise Edition	12/16/02	N
8	hp invent	HP ProLiant DL580 G2 4P	2,605	38 US $	08/15/03	Microsoft SQL Server 2000 Enterprise Edition	Microsoft Windows Server 2003 Enterprise Edition	08/15/03	N
9	Sun microsystems	SunFire V480	2,140	44 US $	11/26/03	Sybase IQ 12.5	Sun Solaris 9	07/28/03	N
10	hp invent	HP ProLiant DL580G2 4P	2,106	48 US $	05/28/03	Microsoft SQL Server 2000 Enterprise Edition	Microsoft Windows Server 2003 Enterprise Server	05/28/03	N

Total System Cost	Composite Query-per-hour Metric	Price/Performance
863,410	**12,216.1 QphH@100GB**	**$ 71 per QphH@100GB**

Database Size	Database Manager	Operating System	Availability Date
100GB	**IBM DB2 UDB 8.1**	**SuSE Linux Enterprise Server 8**	**November 8, 2003**



Database Load Time: 00:37:30	Load Included Backup: N	Total Data Storage / Database Size: 58.31

Database Load Time: 00:37:30	Load Included Backup: N	Total Data Storage / Database Size: 58.31
RAID (Base Table): Y	RAID (Base Tables and Auxiliary Data Structures): Y	RAID (ALL): Y

System Configuration:	8 nodes
Processors:	2 x 2GHz AMD 64-bit Opteron Model 246 processor
Memory:	6 x DIMMS 1GB ECC 2700 2.5 CL
Disk Controllers:	1 x QLA2342 HBA (2 ports x 2G bps FC) 1 x IBM FC2-133 HBA (1 ports x 2G bps FC)
Disk Drives:	3 (IBM TotalStorage FAStT EXP700 cases) × 14 (15K FC Hot-Swap Drive) ×18.2 GB 1 × 18.2GB 15K Ultra160 SCSI Hot-Swap Drive
Total Disk Storage:	5831.06 GB

IBM FC2-133 Host Bus Adapter	24P0960	1	1,485	8	11,880	0
Qlogic QLA 2342 Host Bus Adapter	408082	3	2,291	8	18,328	0
IBM USB Keyboard w/2 port USB Hub	22P5185	1	45	1	45	
IBM USB Optical Wheel Mouse	06P4069	1	20	1	20	
APC Smart-UPS Model 1400	32P1020	1	855	1	855	0
E54 15" (13.8" Viewable) Color Monitor	633147N	1	129	1	129	90
NetBAY42S Enterprise Rack	9306421	1	1,439	4	5,756	300
				Subtotal	112,774	8581
Server Storage						
IBM TOTALSTORAGE FASTT EXP700	1740-1RU	1	3,000	24	72,000	17,280
18.2GB 15K FC Hot-Swap Drive	5211	1	681	336	228,816	
Short Wave SFP	2210	1	499	24	11,976	
1M LC-LC FIBRE OPTIC CABLE	5601	1	79	24	1,896	
				Subtotal	300,816	17,280
Server Software						
DB2 UDB ESE V8.1 License/1-yr. Maint.	D518GLL	1	20,451	16	327,216	
DB2 UDB ESE V8.1 Support – 1Yr/Proc	E00BILL	1	974	32		31,168
DB2 UDB ESE V8.1 DPF License/1-yr. Maint.	D518JLL	1	6,143	16	98,288	
DB2 UDB ESE V8.1 DPF Support – 1Yr/Proc	E00BJLL	1	293	32		9,376
SuSE SLES8 8 proc + 1 yr maint.	2119-3INT-8	2	2,585	1	2,585	
SuSE SLES 8 Main. 1 yr/8proc	2119-MFJ-8	2	2,456	4		9,824
				Subtotal	428,089	50,368
Large volume discount on IBM hardware of 14%; prices vary if purchased separately.14%				**Discount**	54,498	
				Total	787,181	76,229
				Three-Year Cost of Ownership:		863,410

Pricing: 1 - IBM Business Partner; 2 - SuSE, 3 - CDW shop online Warranty and Maintenance: The standard warranty has been upgraded to 3 years of 24x7x4 coverage.	**QphH:**	12216.1
	$/QphH:	$71

Prices used in TPC benchmarks reflect the actual prices a customer would pay for a one-time purchase of the stated components. Individually negotiated discounts are not permitted. Special prices based on assumptions about past or future purchases are not permitted. All discounts reflect standard pricing policies for the listed components. For complete details, see the pricing sections of the TPC benchmark specifications. If you find that stated prices are not available according to these terms, please inform the TPC at pricing@tpc.org. Thank you.

Take aways

- Databases from small to extreme large
- DBMS are highly complex systems
- Performance is a user benefit, but also a cost reduction method
- Proper (micro) benchmarks are the yardsticks