Panel discussion

Peter Boncz (CWI)

Architecture-Conscious Databases: sub-optimization or the next big leap?

Sub-Optimization or Big Leap?

- The same stuff all over again..

s/Buffer Cache/L2/g

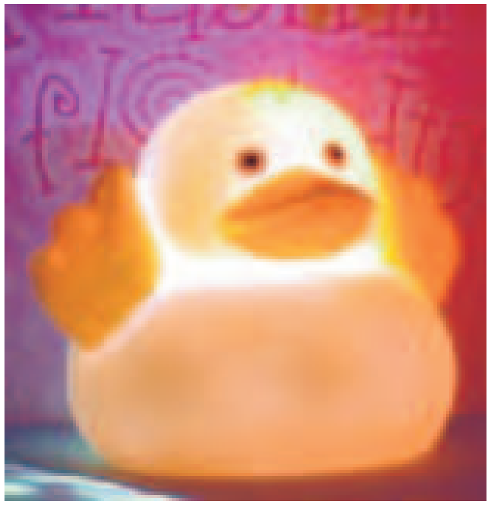
s/Page Fault/Cache Miss/g

s/Disk Block/Cache Line/g

- The final benefits are only a few percentage points

Sub-Optimization or Big Leap?

- Optimizing { cache use, IPC, .. } use can make a huge difference



- Future even more interesting:

  e.g. after hitting the memory wall, we now hit a CPU frequency wall.

"Computer architectures will fundamentally change""

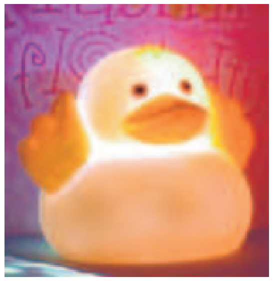
➔ need strong help from software (e.g. multicore)

Questions

- what will computer architectures look like in 5 years?
- do computer architecture trends/changes force us to re-think the classical DBMS architecture?
- to what extent are CPU manufacturers willing to listen to DBMS people?
- what architecture-conscious HYPEWORD data-management challenges/opportunities do you see in the next 5 years?

HYPEWORD in { XML, stream, mobile/ubiquitous, sensor, data mining, multimedia, biological }

The panelists



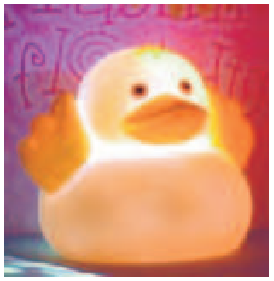
➔

?

- Doug Carmean (Intel)
- Bradley Kuszmaul (MIT)
- Jignesh Patel (Univ. Michigan)
- Babak Falsafi (CMU)
- Kenneth Ross (Columbia Univ.)

DAMON
Data Management On New Hardware

The panelists



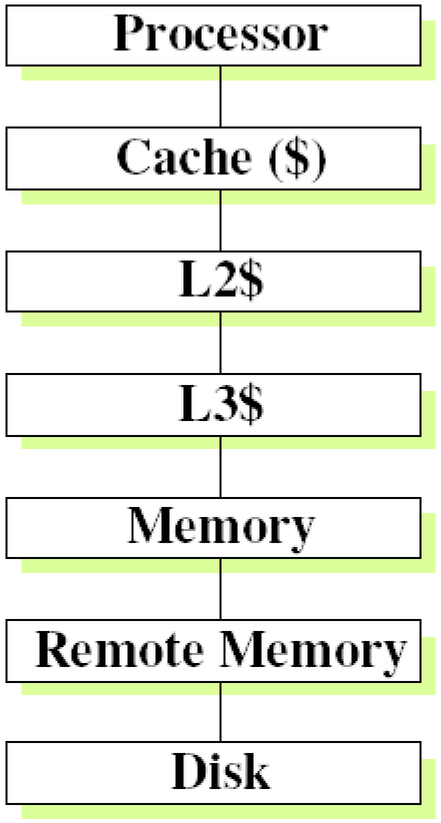
→

?

- Doug Carmean (Intel)
- **Bradley Kuszmaul (MIT)**
- Jignesh Patel (Univ. Michigan)
- Babak Falsafi (CMU)
- Kenneth Ross (Columbia Univ.)

The cache hierarchy is where the performance goes.

The Memory Hierarchy



1ns latency, 64-byte lines, 32KB

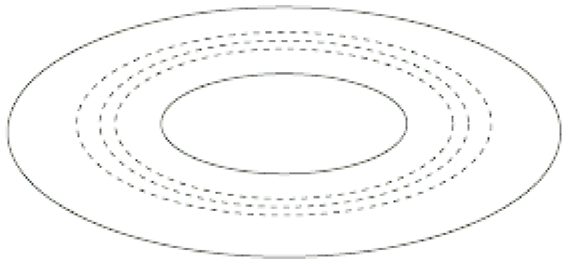
3ns latency, 256B lines, 4MB total

60ns latency, 4KB blocks, 16GB total

10,000ns latency, 1TB total

10^7ns latency, 100MB/s, Petabyte scale (complex latency)

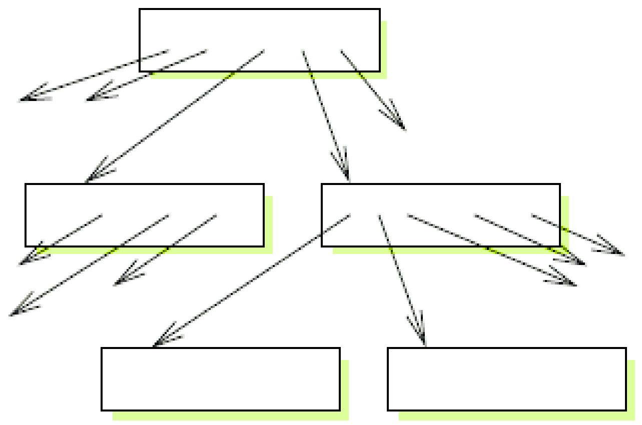
Disk Systems Are Complex



- Rotational latency 5ms,
- long seek 8ms,
- short seek 1ms.
- Outer tracks contain twice as much data as inner tracks, with the same transfer time (1 rotation).
- Caching in the disk controller.

Half-power point is size at which half your cost is latency,

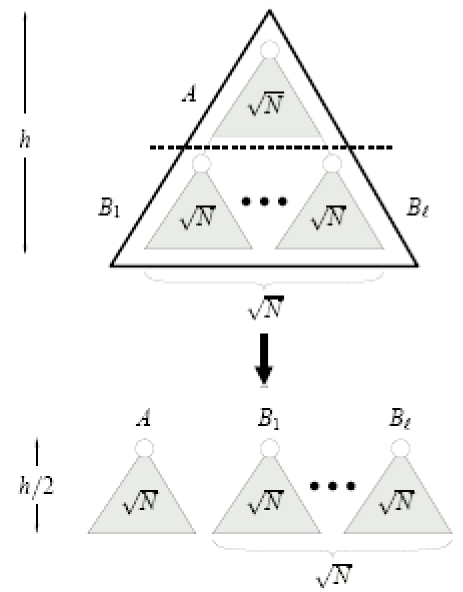
B-Tree Block Size

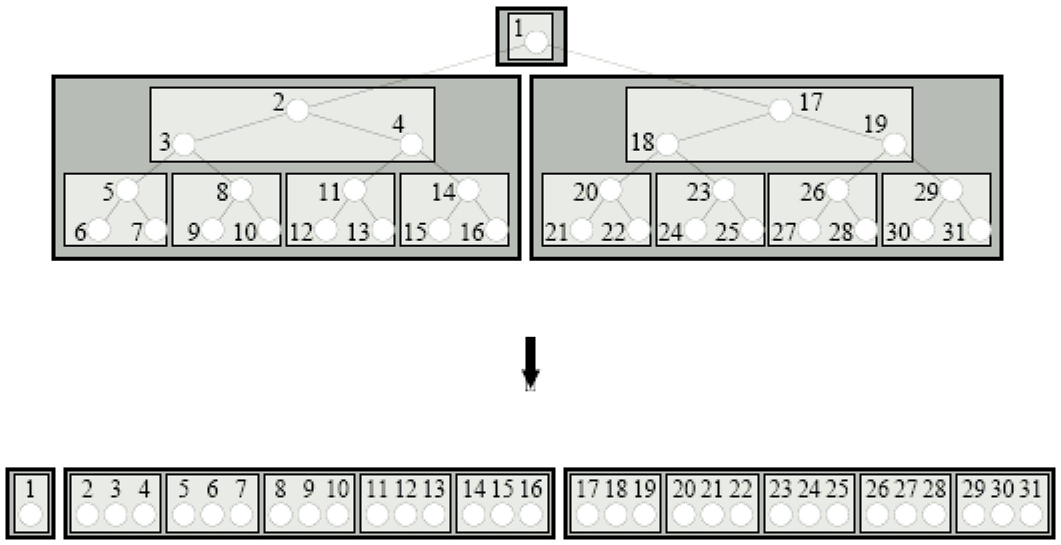


- B-trees choose the node degree so that each node fits in a block.
- What block size to use?
  - 64B (cache lines),
  - 4KB (page size),
  - $1/3$MB (random seek half-power point)

Small blocks make poor use of disk. Big blocks make poor use of cache (and may be expensive for insertions.)

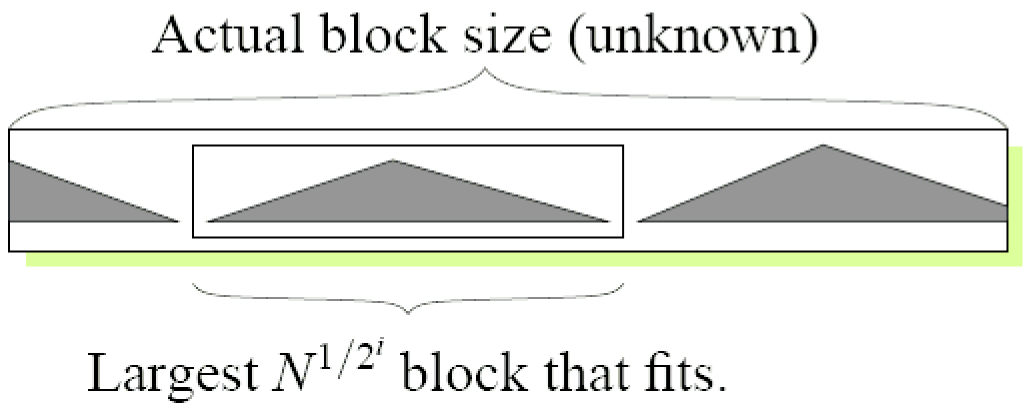
A Static Cache-Oblivious B-tree





First lay out the tree into blocks of size $\sqrt{N}$. Then lay out those blocks into blocks of size $N^{1/4}$. (And those into blocks of size $N^{1/8}$, ...)

Cache-Oblivious B-trees Optimize for All Block Sizes Simultaneously


Actual block size (unknown)
Largest $N^{1/2^i}$ block that fits.

Two blocks of size $\sqrt{B}$ give the same fanout as one block of size B. $\rightarrow$ At most 2X more block transfers than optimal.

(Potentially another 2X due to alignment. But small blocks, which pay the first factor of two, are unlikely to cross the actual block boundary.)

This analysis applies to *all* block sizes at the same time.

Some Performance measurements

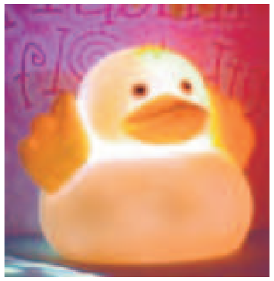
Random searches:

Data structure	Average time per search	
	small-machine	big-machine
CO Btree	12.3ms	13.8ms
Btree: 4KB Blocks:	17.2ms	22.4ms
16KB blocks:	13.9ms	22.1ms
32KB blocks:	11.9ms	17.4ms
64KB blocks:	12.9ms	17.6ms
128KB blocks:	13.2ms	16.5ms
256KB blocks:	18.5ms	14.4ms
512KB blocks:		16.7ms

Range queries run 100 times faster.

Conclusion: Must optimize for the cache heirarchy. Can do so in an architecturally independent way.

The panelists

 → ?

- Doug Carmean (Intel)
- Bradley Kuszmaul (MIT)
- **Jignesh Patel (Univ. Michigan)**
- Babak Falsafi (CMU)
- Kenneth Ross (Columbia Univ.)

Architecture-Conscious Databases: Sub-optimization or the Next Big Leap?

Jignesh M. Patel
University of Michigan

Those who cannot remember the past are condemned to repeat it.
- George Santayana

History (from a DB perspective)

- 1970s, 80s: Birth of RDBMS
  - Disk I/O are expensive: buffer management, prefer sequential IOs.
  - Customized hardware for DBMS?
    - Customized hardware is not cost-effective or portable, better to use smart software-based solutions for query processing.
- 1980s, 90s: High performance and scalable DBMS
  - Parallelism: RDBMS are very amenable to parallelism.
    - Shared-nothing - essentially application level-parallelism.
    - Need to worry about startup, interference, and skew.
  - Expanding DBMS functionality: Object*, external functions, rules, …
- 1990s, 21st century: Ubiquitous DBMS
  - Expanding DBMS applications: streams, scientific, semi-structured, personal data management, …
  - Architecture-conscious RDBMS:
    - Memory Wall: Design cache-aware methods, prefetching, …
    - Exploit new processor features: SIMD, co-processors (GPU), …

Future Computer Architecture and impact on DBMS

- SMP on a chip
  - Shared-nothing parallel DBMS will be the default installation.
- Memory hierarchy will continue
  - But processor clock speeds are not increasing rapidly

CPU extensions for DBMSs

- No compelling reason to add DBMS-specific extensions
- Recall the database machine's era
  - DBMS has smart and efficient software-based techniques
- What is the payoff for the hardware vendor?
  - Servers: Application servers (Java + DB), file servers, ...
  - Clients: Entertainment, Virus/spyware scanners, personal data management, ...
- DBMS part of a complex suite of software even on server machines
  - Lots of time is spent in external function and (Java) application code.

Rethink traditional DBMS Architecture?

- We already have shared-nothing parallelism.
- Memory hierarchy fundamentally remains the same: smaller and faster memory is closer to the processor
  - For shared-nothing parallelism, synchronization may be cheaper with shared on-chip memory
- Coprocessors for RDBMS operations
  - There will be less pressure to use coprocessors.
  - Not clear if this approach has a significant performance advantage when compared to the **best** current methods on regular processors.
  - End-to-end application performance?
    - RDBMS query processing will be a shrinking component.
  - Speed is only a small factor. Portability is important.

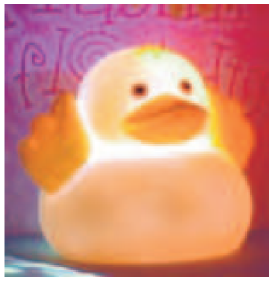
NO!

New Challenges

The role of DBMS is expanding!

- Efficient fuzzy matching algorithms on text, personal digital records, multimedia, graphs.
- Data mining will be more common as data volume continues to explode and complexity of analysis increases. Relatively little work on parallel algorithms here.
- Scientific workloads: Lots of very computationally expensive data analysis. Needs massively parallel methods. Data volume is exploding.
- But need clean programming interfaces.

The panelists



→

?

- Doug Carmean (Intel)
- Bradley Kuszmaul (MIT)
- Jignesh Patel (Univ. Michigan)
- **Babak Falsafi (CMU)**
- Kenneth Ross (Columbia Univ.)

DaMoN 2005 panel on

Architecture-Conscious Databases: sub-optimization or the next big leap?

Babak Falsafi
Computer Architecture Lab
Carnegie Mellon
http://www.ece.cmu.edu/CALCM

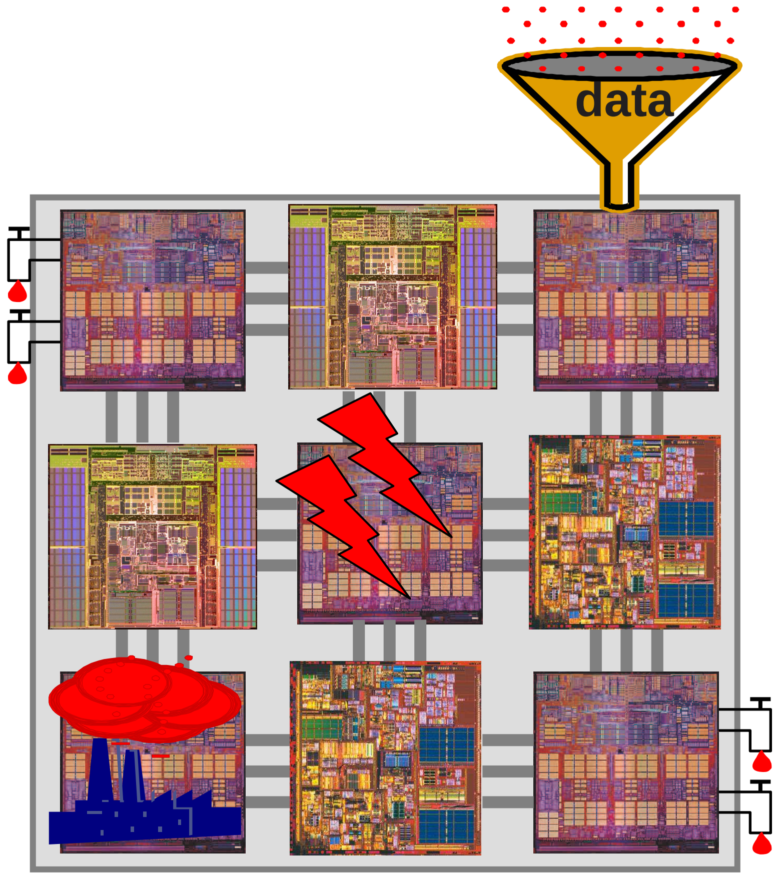
Technology Scaling Trends

Good news:

- 100B trans/chip by 2015
- Tens of cores

Bad news:

- Heterogeneous, smoking and unreliable cores
- Faster but constrained clocks
- Off-chip latency/bw bottlenecks continue



2015 Multi-core Chip

Scaling Implications for DB Servers

Lots of cores & on-chip memory:

- Parallelism galore: intra-transaction
- Sea of memory, not hierarchies

Heterogeneous resources:

- ILP, TLP, and DLP (vector)?
- Both HW-/SW-managed caches

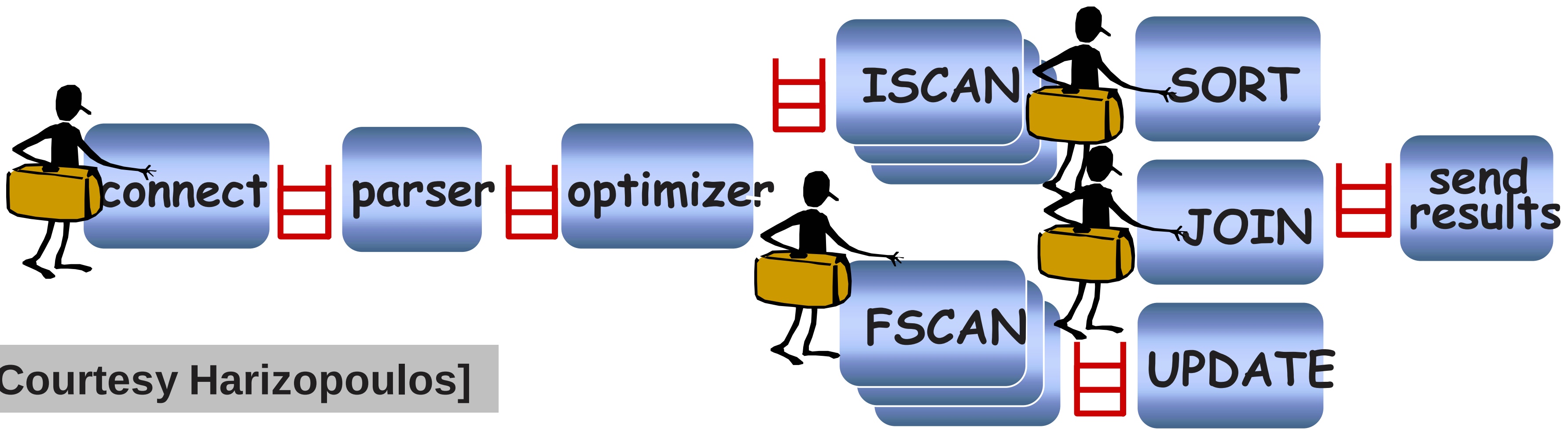
Unreliable HW:

- Fine-grain transaction semantics

Need tight HW/SW collaboration!

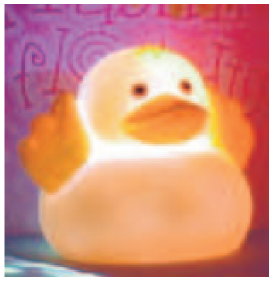
One Solution: Staged Server Architecture

- Componentize server into operators
- Operator-level pipelining
  - Map operators to cores
  - HW/SW data streaming



[Courtesy Harizopoulos]

The panelists

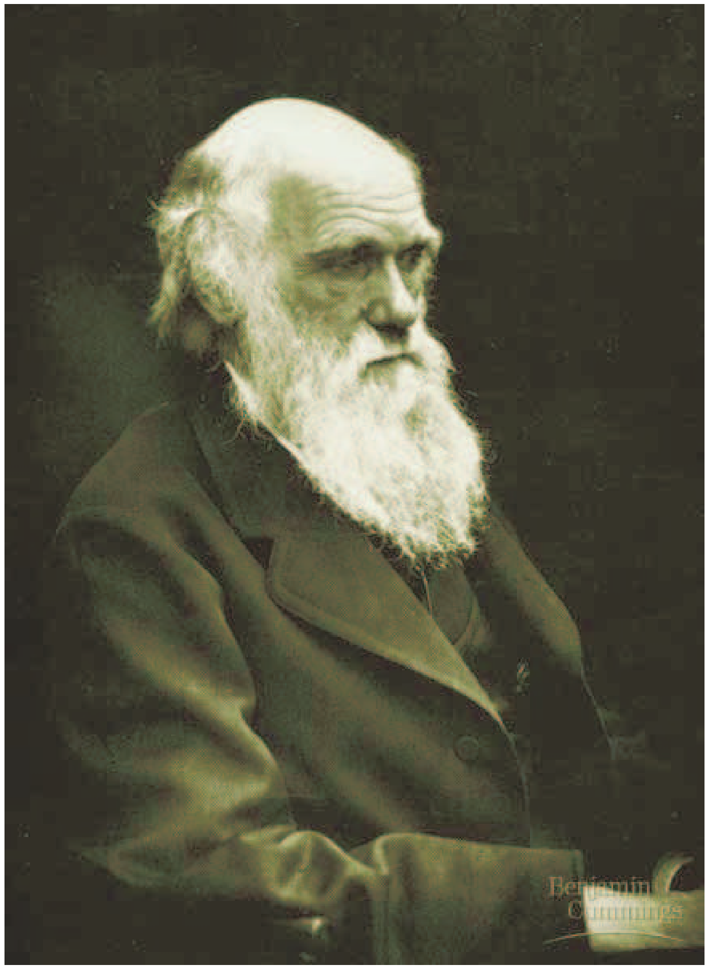
 ➔ ?

- Doug Carmean (Intel)
- Bradley Kuszmaul (MIT)
- Jignesh Patel (Univ. Michigan)
- Babak Falsafi (CMU)
- **Kenneth Ross (Columbia Univ.)**

Architecture-Conscious Databases Panel

DaMoN 2005

Ken Ross



Charles Darwin

Survival of the Fittest

- Small genetic changes make the entity fitter, and are selected by the environment → Evolution
- Small ≠ Unimportant
- Most changes not visible
- Changes to the environment create new optimization criteria for natural selection.

Survival of the Fittest DB

- Small implementation changes make the DB faster, and are selected by the marketplace → Evolution
- Small ≠ Unimportant
- Most changes not visible to users or developers (and shouldn't be!)
- Changes to the technology environment create new optimization criteria for DB implementation.

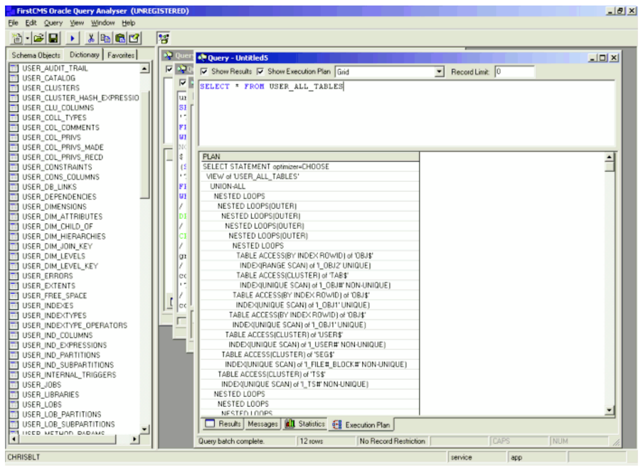
“Sub-Optimization” or “The Next Big Thing”?

- Sub-optimization, of course!
- But many small changes can radically alter the overall structure

The Old Days

DB Implementor

DB software



```
USER_AUDIT_TRAIL
USER_CATALOG
USER_CLUSTERS
USER_CLUSTER_HASH_EXPRESSIO
USER_CLU_COLUMNS
USER_COLL_TYPES
USER_COL_COMMENTS
USER_COL_PRIVS
USER_COL_PRIVS_MADE
USER_COL_PRIVS_RECD
USER_CONSTRAINTS
USER_CONS_COLUMNS
USER_DB_LINKS
USER_DEPENDENCIES
USER_DIMENSIONS
USER_DIM_ATTRIBUTES
USER_DIM_CHILD_OF
USER_DIM_HIERARCHIES
USER_DIM_JOIN_KEY
USER_DIM_LEVELS
USER_DIM_LEVEL_KEY
USER_ERRORS
USER_EXTENTS
USER_FREE_SPACE
USER_INDEXES
USER_INDEXTYPES
USER_INDEXTYPE_OPERATORS
USER_IND_COLUMNS
USER_IND_EXPRESSIONS
USER_IND_PARTITIONS
USER_IND_SUBPARTITIONS
USER_INTERNAL_TRIGGERS
USER_JOBS
USER_LIBRARIES
USER_LOBS
USER_LOB_PARTITIONS
USER_LOB_SUBPARTITIONS
```

```
SELECT * FROM USER_ALL_TABLES
```

```
PLAN
SELECT STATEMENT optimizer=CHOOSE
 VIEW of 'USER_ALL_TABLES'
  UNION-ALL
   NESTED LOOPS
    NESTED LOOPS(OUTER)
     NESTED LOOPS(OUTER)
      NESTED LOOPS(OUTER)
       NESTED LOOPS(OUTER)
        NESTED LOOPS
         TABLE ACCESS(BY INDEX ROWID) of 'OBJ$'
          INDEX(RANGE SCAN) of 'I_OBJ2' UNIQUE)
         TABLE ACCESS(CLUSTER) of 'TAB$'
          INDEX(UNIQUE SCAN) of 'I_OBJ#' NON-UNIQUE)
        TABLE ACCESS(BY INDEX ROWID) of 'OBJ$'
         INDEX(UNIQUE SCAN) of 'I_OBJ1' UNIQUE)
       TABLE ACCESS(BY INDEX ROWID) of 'OBJ$'
        INDEX(UNIQUE SCAN) of 'I_OBJ1' UNIQUE)
      TABLE ACCESS(CLUSTER) of 'USER$'
       INDEX(UNIQUE SCAN) of 'I_USER#' NON-UNIQUE)
     TABLE ACCESS(CLUSTER) of 'SEG$'
      INDEX(UNIQUE SCAN) of 'I_FILE#_BLOCK#' NON-UNIQUE)
    TABLE ACCESS(CLUSTER) of 'TS$'
     INDEX(UNIQUE SCAN) of 'I_TS#' NON-UNIQUE)
   NESTED LOOPS
    NESTED LOOPS
```

```
Query batch complete.   12 rows   No Record Restriction
CHRISBLT   service   app
```

Disk block size,

Physical layout, …

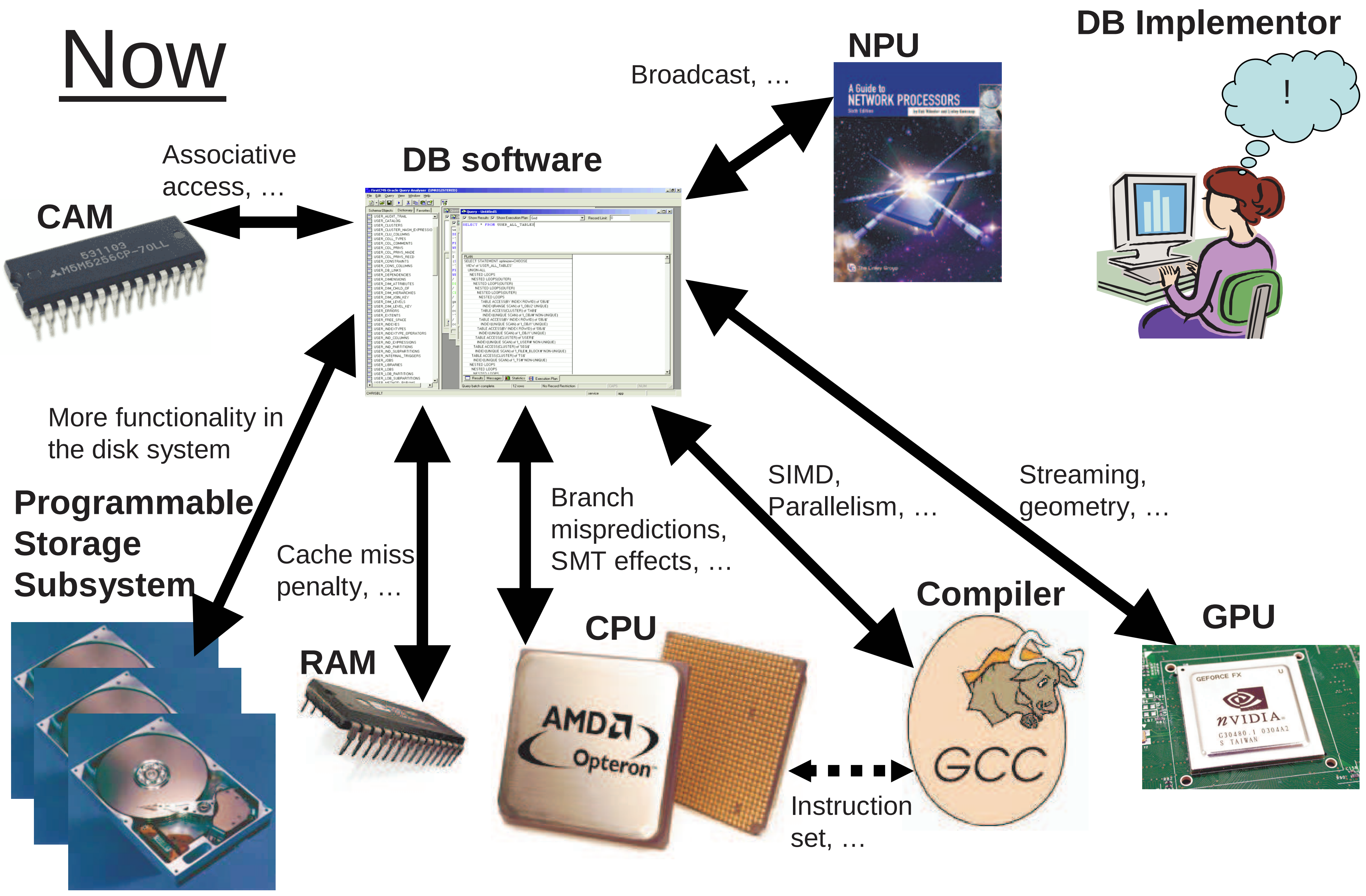
Sequential access,

…

Disk

As disks got better, there was little need for fundamental change.

Now


DB Implementor
!
NPU
Broadcast, …
DB software
CAM
Associative access, …
More functionality in the disk system
Programmable Storage Subsystem
Cache miss penalty, …
RAM
Branch mispredictions, SMT effects, …
CPU
SIMD, Parallelism, …
Compiler
Streaming, geometry, …
GPU
Instruction set, …

Challenges

- Too much is visible to the DB implementor!
- Abstractions, modularity
- Interactions: many competing “small” changes
- Management

Summary

- [carmean] CPU future = Cell done right
- [carmean] Hardware companies are willing to listen, also on APIs
- [kuszmaul] Not cache-conscious ➔cache-oblivious (will also do disk!)
- [patel] Focus on extensibility, RDBMS query processing a shrinking component
  optimize/parallelize: fuzzy matching, graph algorithms, data mining, scientific
- [patel] RDBMS architecture is fine; will handle massive multiprocessors well
- [patel] limitations of coprocessors will prevent adoption
- [falsafi] heterogeneous CPUs, sea of programmable memory (no hierarch. cache)
- [falsafi] bottomline: need for tight hardware/software collaboration
- [falsafi] solution: staged servers architecture (operator-level pipelining)
- [ross] survival of the fittest database: small changes lead evolution
- [ross] database implementor is confused (from disk alone to many arch. factors)
  challenge=> abstraction