

Database Architectures for New Hardware

a tutorial by

Anastassia Ailamaki

Database Group

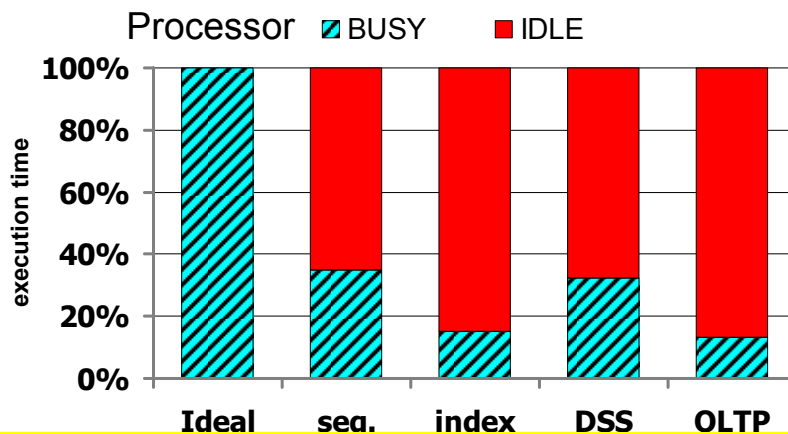
Carnegie Mellon University

<http://www.cs.cmu.edu/~natassa>

Focus of this tutorial

Databases
@Carnegie Mellon

- DB workload execution on a modern computer

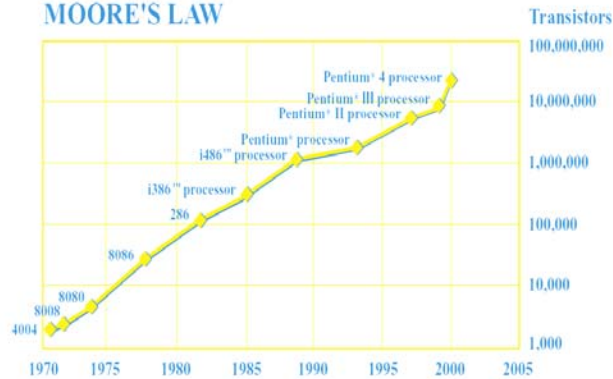


**DBMS can run MUCH faster if they use
new hardware efficiently**

Trends in processor performance

- ❑ Scaling # of transistors, innovative microarchitecture
- ❑ Higher performance, despite technological hurdles!

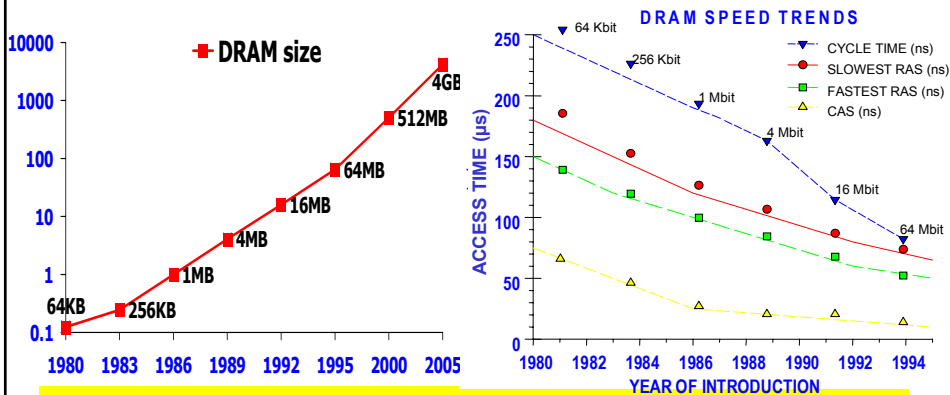
MOORE'S LAW



Processor speed doubles every 18 months

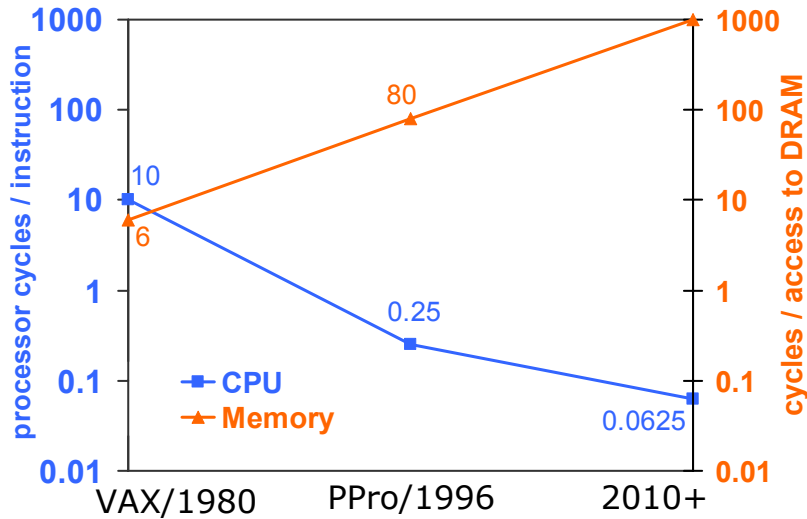
Trends in Memory (DRAM) Performance

- ❑ Memory capacity increases exponentially
 - DRAM Fabrication primarily targets density
- ❑ Speed increases linearly



Larger but not as much faster memories

The Memory/Processor Speed Gap



Trip to memory = thousands of instructions!

New Hardware

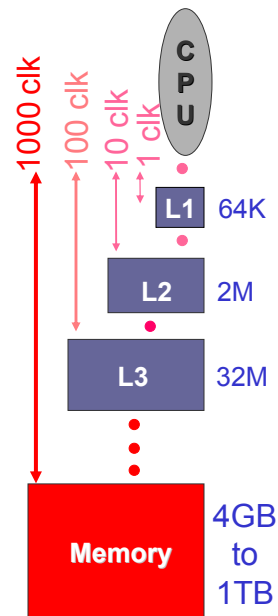
- ❑ Caches trade off capacity for speed
- ❑ Exploit instruction/data locality
- ❑ Demand fetch/wait for data

[ADH99]:

- ❑ Running top 4 database systems
- ❑ **At most 50% CPU utilization**

But wait a minute...

Isn't I/O the bottleneck???

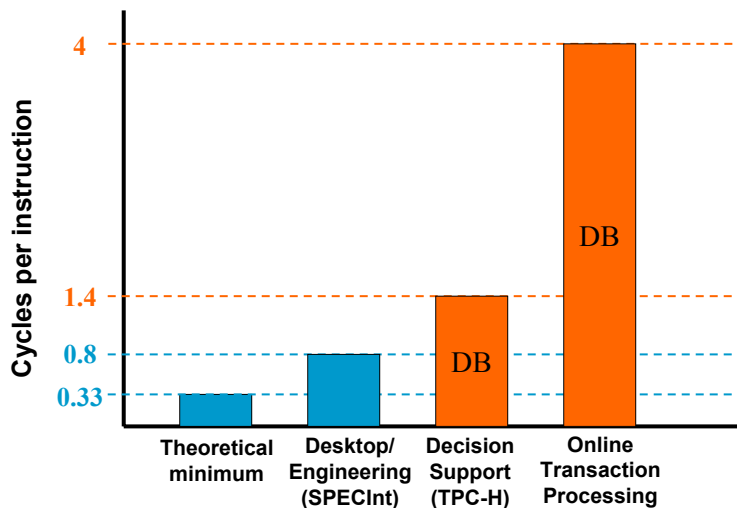


Modern storage managers

- ❑ Several decades work to hide I/O
 - ❑ Asynchronous I/O + Prefetch & Postwrite
 - Overlap I/O latency by useful computation
 - ❑ Parallel data access
 - Partition data on modern disk array [PAT88]
 - ❑ Smart data placement / clustering
 - Improve data locality
 - Maximize parallelism
 - Exploit hardware characteristics
- ...and larger main memories fit more data
- 1MB in the 80's, 10GB today, TBs coming soon

DB storage mgrs efficiently hide I/O latencies

Why should we (databasers) care?



Database workloads under-utilize hardware
New bottleneck: Processor-memory delays

Breaking the Memory Wall

Wish for a Database Architecture:

- ❑ that uses hardware intelligently
- ❑ that won't fall apart when new computers arrive
- ❑ that will adapt to alternate configurations

Efforts from multiple research communities

- ❑ Cache-conscious data placement and algorithms
- ❑ Instruction stream optimizations
- ❑ Novel database software architectures
- ❑ Novel hardware designs (covered briefly)

Detailed Outline

- ❑ Introduction and Overview
- ❑ New Hardware
 - Execution Pipelines
 - Cache memories
- ❑ Where Does Time Go?
 - Measuring Time (Tools and Benchmarks)
 - Analyzing DBs: Experimental Results
- ❑ Bridging the Processor/Memory Speed Gap
 - Data Placement
 - Access Methods
 - Query Processing Algorithms
 - Instruction Stream Optimizations
 - Staged Database Systems
- ❑ Newer Hardware
- ❑ Hip and Trendy
 - Query co-processing
 - Databases on MEMStore
- ❑ Directions for Future Research

Outline

- Introduction and Overview
- **New Hardware**
- Where Does Time Go?
- Bridging the Processor/Memory Speed Gap
- Hip and Trendy
- Directions for Future Research

This Section's Goals

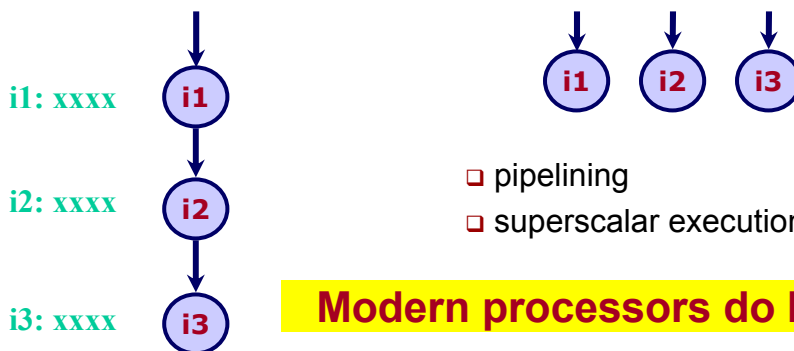
- Understand how a program is executed
 - How new hardware parallelizes execution
 - What are the pitfalls
- Understand why database programs do not take advantage of microarchitectural advances
- Understand memory hierarchies
 - How they work
 - What are the parameters that affect program behavior
 - Why they are important to database performance

Outline

- Introduction and Overview
- **New Hardware**
 - Execution Pipelines
 - Cache memories
- Where Does Time Go?
- Bridging the Processor/Memory Speed Gap
- Hip and Trendy
- Directions for Future Research

Sequential Program Execution

Sequential Code Instruction-level Parallelism (ILP)

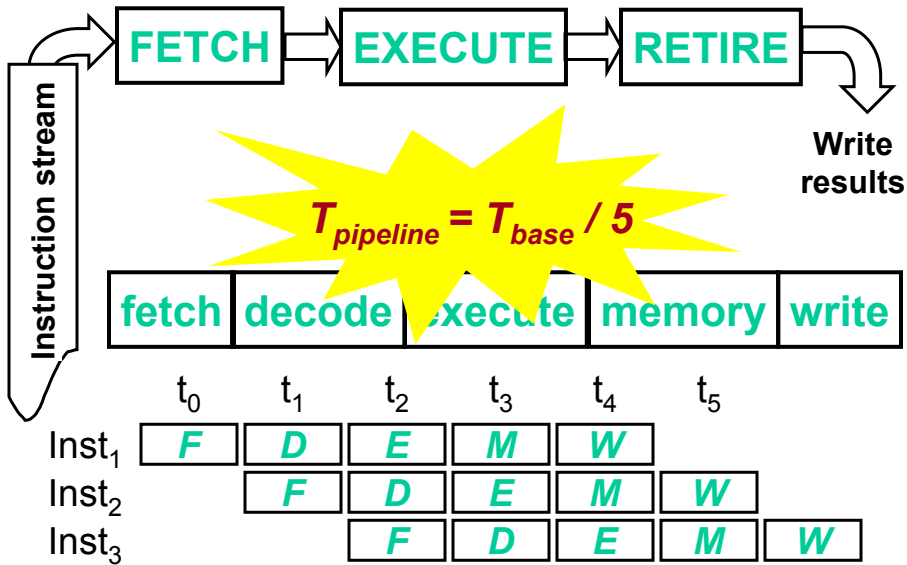


- pipelining
- superscalar execution

Modern processors do both!

- Precedences: overspecifications
- Sufficient, NOT necessary for correctness

Pipelined Program Execution



Pipeline Stalls (delays)

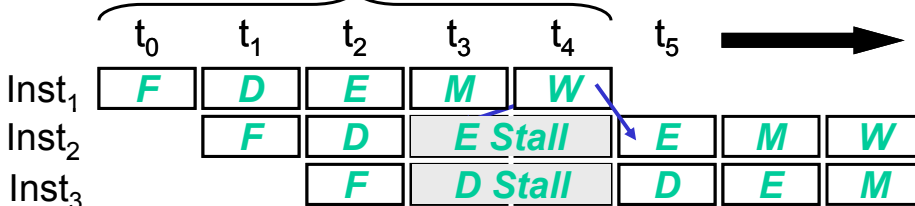
- Reason: dependencies between instructions
- E.g.,

Inst₁: $r1 \leftarrow r2 + r3$

Inst₂: $r4 \leftarrow r1 + r2$

Read-after-write (RAW)

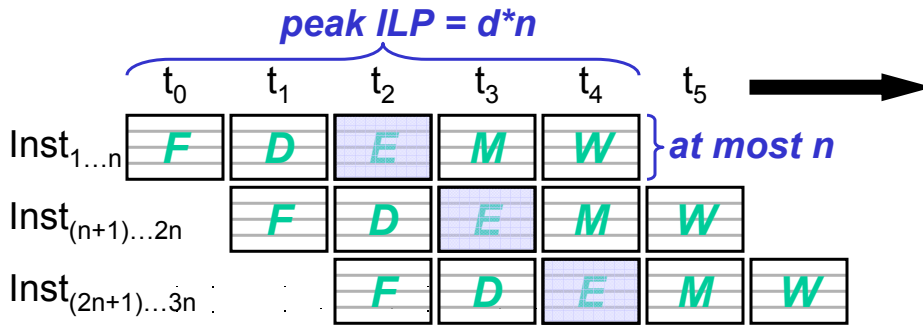
peak ILP = d



Peak instruction-per-cycle (IPC) = CPI = 1

DB programs: frequent data dependencies

Higher ILP: Superscalar Out-of-Order



Peak instruction-per-cycle (IPC)=n (CPI=1/n)

□ **Out-of-order (as opposed to “inorder”) execution:**

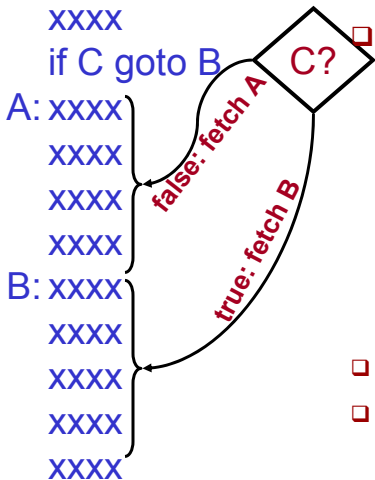
- Shuffle execution of independent instructions
- Retire instruction results using a *reorder buffer*

DB: 1.5x faster than inorder [KPH98,RGA98]
Limited ILP opportunity

Even Higher ILP: Branch Prediction

□ Which instruction block to fetch?

- Evaluating a branch condition causes pipeline stall



□ **IDEA: Speculate branch *while evaluating C!***

- Record branch history in a buffer, predict A or B
- ✓ If correct, saved a (long) delay!
- If incorrect, misprediction penalty = Flush pipeline, fetch correct instruction stream
- Excellent predictors (97% accuracy!)
- Mispredictions costlier in OOO
 - 1 lost cycle = >1 missed instructions!

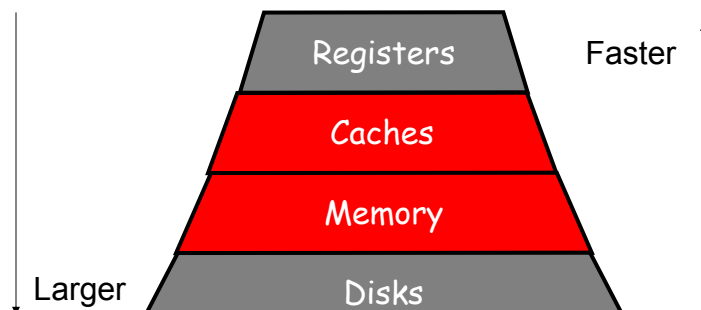
DB programs: long code paths => mispredictions

Outline

- Introduction and Overview
- **New Hardware**
 - Execution Pipelines
 - **Cache memories**
- Where Does Time Go?
- Bridging the Processor/Memory Speed Gap
- Hip and Trendy
- Directions for Future Research

Memory Hierarchy

- Make common case fast
 - common: temporal & spatial locality
 - fast: smaller, more expensive memory
- Keep recently accessed blocks (*temporal locality*)
- Group data into blocks (*spatial locality*)



DB programs: >50% load/store instructions

Cache Contents

- Keep recently accessed block in “cache line”

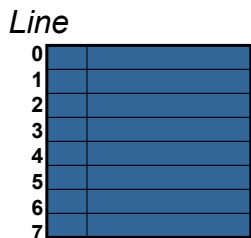


- On memory read
 - if incoming address = a stored address tag then
 - HIT: return data
 - else
 - MISS: choose & displace a line in use
 - fetch new (referenced) block from memory into line
 - return data

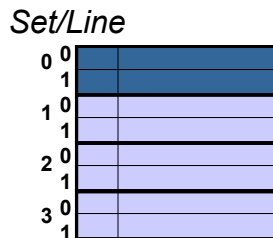
**Important parameters:
cache size, cache line size, cache associativity**

Cache Associativity

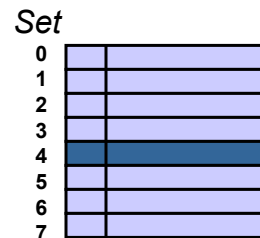
- means # of lines a block can be in (set size)
- Replacement: LRU or random, within set



Fully-associative
a block goes in
any frame



Set-associative
a block goes in
any frame in
exactly one set



Direct-mapped
a block goes in
exactly one
frame

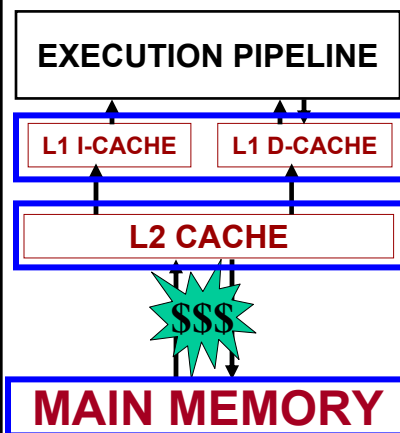
lower associativity ⇒ faster lookup

Miss Classification (3+1 C's)

- ❑ compulsory (cold)
 - “cold miss” on first access to a block
— *defined as: miss in infinite cache*
- ❑ capacity
 - misses occur because cache not large enough
— *defined as: miss in fully-associative cache*
- ❑ conflict
 - misses occur because of restrictive mapping strategy
 - only in set-associative or direct-mapped cache
— *defined as: not attributable to compulsory or capacity*
- ❑ coherence
 - misses occur because of sharing among multiprocessors

Cold misses are unavoidable
Capacity, conflict, and coherence misses can be reduced

Lookups in Memory Hierarchy



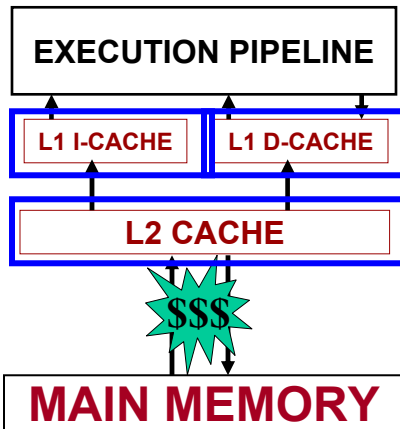
$$\text{miss rate} = \frac{\# \text{misses}}{\# \text{references}}$$

- ❑ L1: Split, 16-64K each.
As fast as processor (1 cycle)
- ❑ L2: Unified, 512K-8M
Order of magnitude slower than L1
(there may be more cache levels)
- ❑ Memory: Unified, 512M-8GB
~400 cycles (Pentium4)

Trips to memory are most expensive

Miss penalty

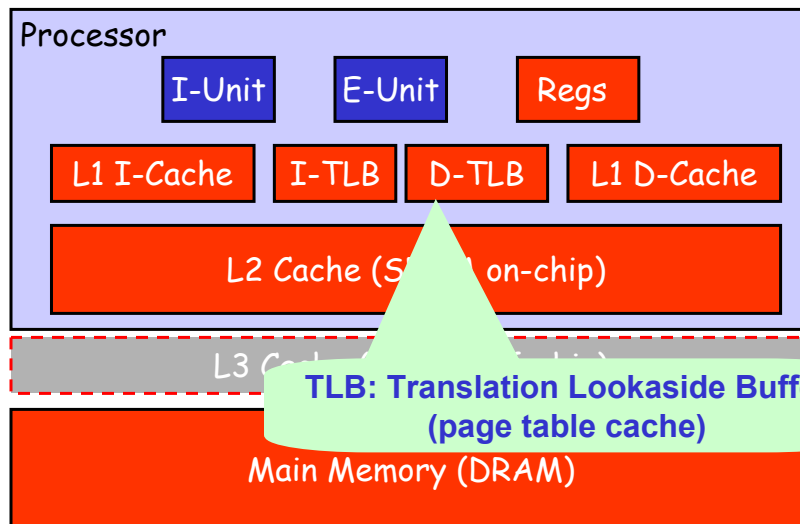
□ means the time to fetch and deliver block
 $avg(t_{access}) = t_{hit} + miss\ rate * avg(miss\ penalty)$



- Modern caches: *non-blocking*
- L1D: low miss penalty, if L2 hit (partly overlapped with OOO execution)
- L1I: In critical execution path. Cannot be overlapped with OOO execution.
- L2: High penalty (trip to memory)

DB: long code paths, large data footprints

Typical processor microarchitecture



Will assume a 2-level cache in this talk

Summary: New Hardware

- ❑ Fundamental goal in processor design: max ILP
 - Pipelined, superscalar, speculative execution
 - Out-of-order execution
 - Non-blocking caches
 - Dependencies in instruction stream lower ILP
- ❑ Deep memory hierarchies
 - Caches important for database performance
 - Level 1 instruction cache in critical execution path
 - Trips to memory most expensive
- ❑ DB workloads on new hardware
 - Too many load/store instructions
 - Tight dependencies in instruction stream
 - Algorithms not optimized for cache hierarchies
 - Long code paths
 - Large instruction and data footprints

Outline

- ❑ Introduction and Overview
- ❑ New Hardware
- ❑ **Where Does Time Go?**
- ❑ Bridging the Processor/Memory Speed Gap
- ❑ Hip and Trendy
- ❑ Directions for Future Research

This Section's Goals

- ❑ Hardware takes time: how do we measure time?
- ❑ Understand how to efficiently analyze microarchitectural behavior of database workloads
 - Should we use simulators? When? Why?
 - How do we use processor counters?
 - Which tools are available for analysis?
 - Which database systems/benchmarks to use?
- ❑ Survey experimental results on workload characterization
 - Discover what matters for database performance

Outline

- ❑ Introduction and Overview
- ❑ New Hardware
- ❑ **Where Does Time Go?**
 - **Measuring Time (Tools and Benchmarks)**
 - Analyzing DBs: Experimental Results
- ❑ Bridging the Processor/Memory Speed Gap
- ❑ Hip and Trendy
- ❑ Directions for Future Research

Simulator vs. Real Machine

Real machine

- ❑ Limited to available hardware counters/events
- ❑ Limited to (real) hardware configurations
- ❑ Fast (real-life) execution
 - Enables testing real: large & more realistic workloads
- ❑ Sometimes not repeatable

Simulator

- ❑ Can measure any event
 - ❑ Vary hardware configurations
 - ❑ (Too) Slow execution
 - Often forces use of scaled-down/simplified workloads
 - ❑ Always repeatable
- ❑ **Tool:** performance counters ❑ Virtutech Simics, SimOS, SimpleScalar, etc.

Real-machine experiments to locate problems
Simulation to evaluate solutions

Hardware Performance Counters

- ❑ What are they?
 - Special purpose registers that keep track of programmable events
 - Non-intrusive counts “accurately” measure processor events
 - Software API’s handle event programming/overflow
 - GUI interfaces built on top of API’s to provide higher-level analysis
- ❑ What can they count?
 - Instructions, branch mispredictions, cache misses, etc.
 - No standard set exists
- ❑ Issues that may complicate life
 - Provides only hard counts, analysis must be done by user or tools
 - Made specifically for each processor
 - even processor families may have different interfaces
 - Vendors don’t like to support because is not profit contributor

Evaluating Behavior using HW Counters

- ❑ Stall time (cycle) counters
 - very useful for time breakdowns
 - (e.g., instruction-related stall time)
- ❑ Event counters
 - useful to compute ratios
 - (e.g., # misses in L1-Data cache)
- ❑ Need to understand counters before using them
 - Often not easy from documentation
 - Best way: microbenchmark (run programs with pre-computed events)
 - E.g., strided accesses to an array

Example: Intel PPRO/PIII

Cycles	CPU_CLK_UNHALTED
Instructions	INST_RETIRED
L1 Data (L1D) accesses	DATA_MEM_REFS
L1 Data (L1D) misses	DCU_LINES_IN
L2 Misses	L2_LINES_IN
Instruction-related stalls	IFU_MEM_STALL
Branches	BR_INST_DECODED
Branch mispredictions	BR_MISS_PRED_RETIRED
TLB misses	ITLB_MISS
L1 Instruction misses	IFU_IFETCH_MISS
Dependence stalls	PARTIAL_RAT_STALLS
Resource stalls	RESOURCE_STALLS

“time”

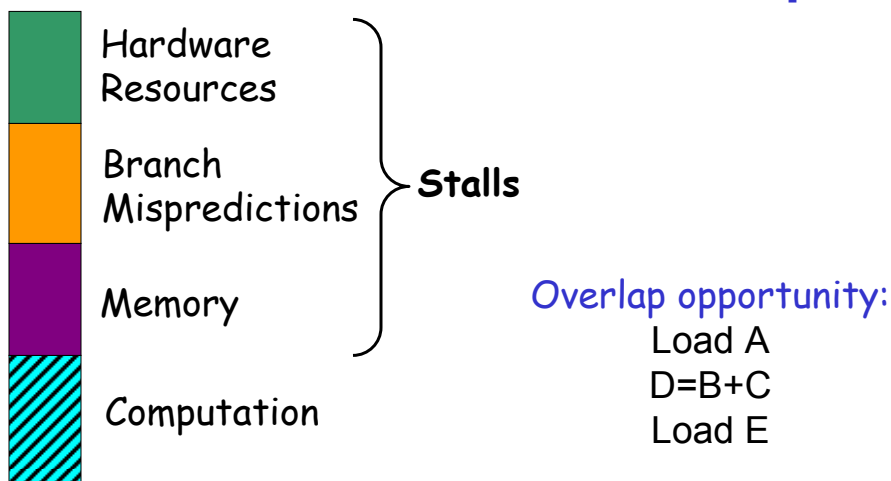
Lots more detail, measurable events, statistics
Often >1 ways to measure the same thing

Producing time breakdowns

- ❑ Determine benchmark/methodology (more later)
- ❑ Devise formulae to derive useful statistics
- ❑ Determine (and test!) software
 - E.g., Intel Vtune (GUI, sampling), or *emon*
 - Publicly available & universal (e.g., *PAPI* [DMM04])
- ❑ Determine time components $T_1 \dots T_n$
 - Determine how to measure each using the counters
 - Compute execution time as the sum
- ❑ Verify model correctness
 - Measure execution time (in #cycles)
 - **Ensure measured time = computed time (or almost)**
 - Validate computations using redundant formulae

Execution Time Breakdown Formula

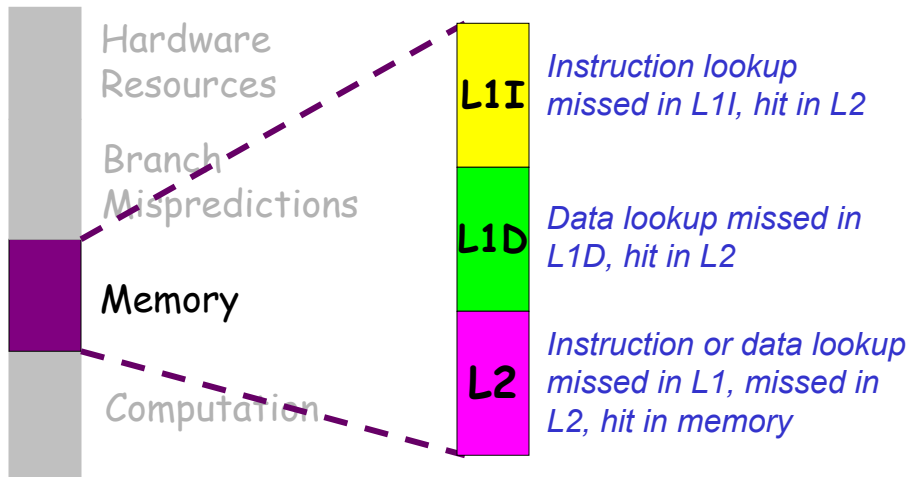
[ADH99]



Execution Time = Computation + Stalls - Overlap

Where Does Time Go (memory)?

[ADH99]



$$\text{Memory Stalls} = \sum_n (\text{stalls at cache level } n)$$

What to measure?

- Decision Support System (DSS:TPC-H)
 - Complex queries, low-concurrency
 - Read-only (with rare batch updates)
 - Sequential access dominates
 - *Repeatable* (unit of work = query)
- On-Line Transaction Processing (OLTP:TPCC, ODB)
 - Transactions with simple queries, high-concurrency
 - Update-intensive
 - Random access frequent
 - *Not repeatable* (unit of work = 5s of execution after rampup)

Often too complex to provide useful insight

Microbenchmarks

[KPH98,ADH99,KP00,SAF04]

- ❑ What matters is basic execution loops
- ❑ Isolate three basic operations:
 - Sequential scan (no index)
 - Random access on records (non-clustered index)
 - Join (access on two tables)
- ❑ Vary parameters:
 - selectivity, projectivity, # of attributes in predicate
 - join algorithm, isolate phases
 - table size, record size, # of fields, type of fields
- ❑ Determine behavior and trends
 - Microbenchmarks can efficiently mimic TPC microarchitectural behavior!
 - Widely used to analyze query execution

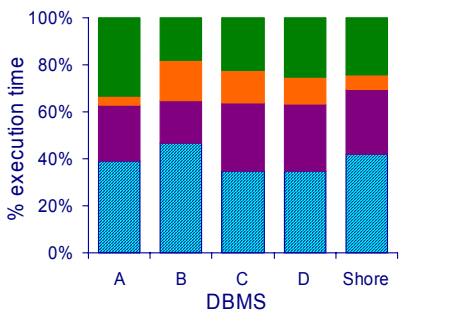
Excellent for microarchitectural analysis

On which DBMS to measure?

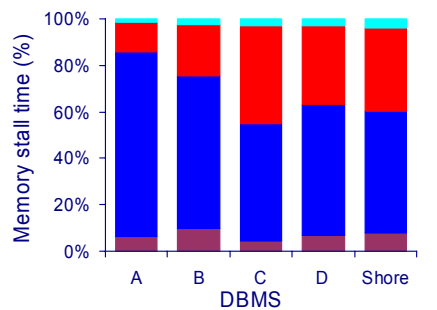
[ADH02]

- ❑ Commercial DBMS are most realistic
 - Difficult to setup, may need help from companies
- ❑ Prototypes can evaluate techniques
 - Shore [ADH01] (for PAX), PostgreSQL[TLZ97] (eval)
 - Tricky: similar behavior to commercial DBMS? **Shore: YES!**

Execution time breakdown



Memory stall time breakdown

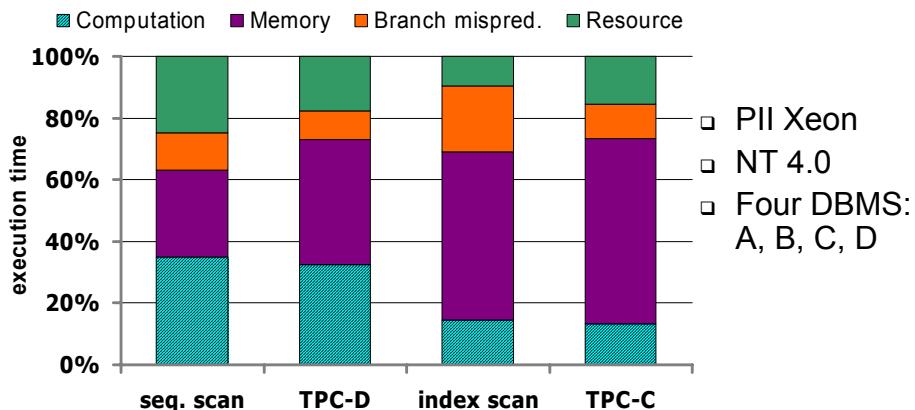


Outline

- Introduction and Overview
- New Hardware
- **Where Does Time Go?**
 - Measuring Time (Tools and Benchmarks)
 - **Analyzing DBs: Experimental Results**
- Bridging the Processor/Memory Speed Gap
- Hip and Trendy
- Directions for Future Research

DB Execution Time Breakdown

[ADH99,BGB98,BGN00,KPH98]

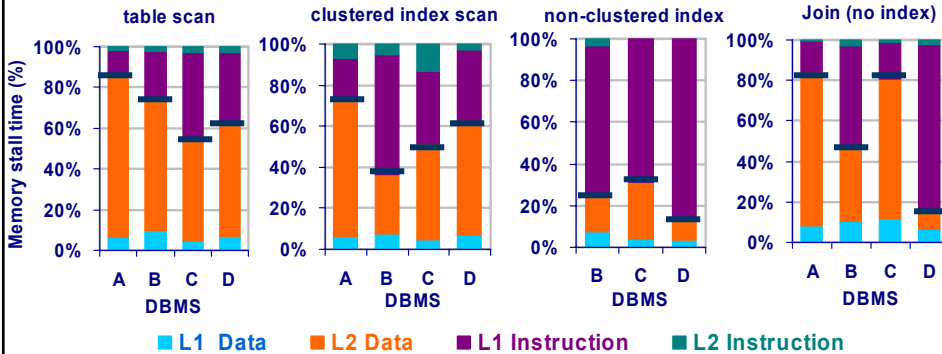


At least 50% cycles on stalls
Memory is major bottleneck
Branch mispredictions increase cache misses!

DSS/OLTP basics: Cache Behavior

[ADH99,ADH01]

- ❑ PII Xeon running NT 4.0, used performance counters
- ❑ Four commercial Database Systems: A, B, C, D



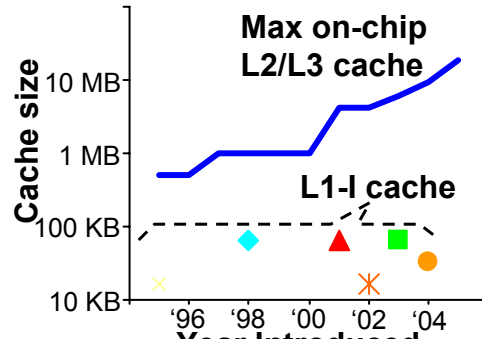
Bottlenecks: data in L2, instructions in L1
Random access (OLTP): L1I-bound

Why Not Increase L1I Size?

[HA04]

- ❑ Problem: a larger cache is typically a slower cache
- ❑ Not a big problem for L2
- ❑ L1I: in *critical execution path*
- ❑ slower L1I: slower clock

❑ Trends:

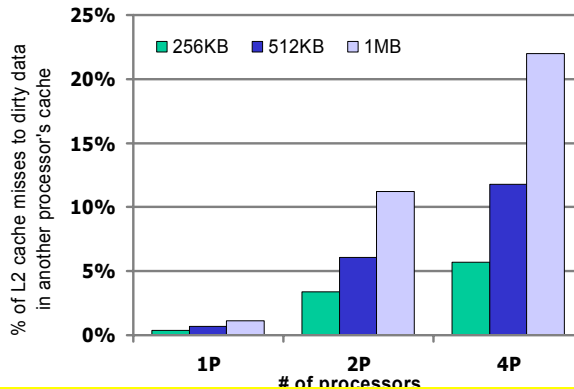


L1I size is stable
L2 size increase: Effect on performance?

Increasing L2 Cache Size

[BGB98,KPH98]

- DSS: Performance improves as L2 cache grows
- Not as clear a win for OLTP on multiprocessors
 - Reduce cache size \Rightarrow more capacity/conflict misses
 - Increase cache size \Rightarrow more coherence misses

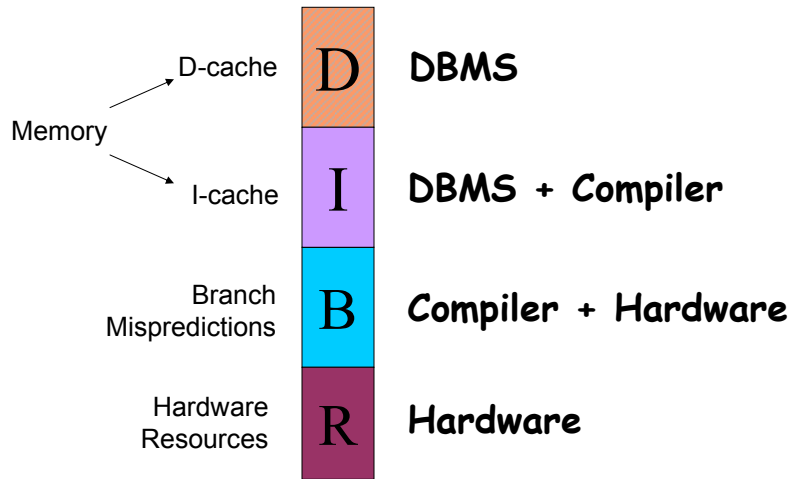


Larger L2: trade-off for OLTP

Summary: Where Does Time Go?

- Goal: discover bottlenecks
 - Hardware performance counters \Rightarrow time breakdown
 - Tools available for access and analysis (+simulators)
 - Run commercial DBMS and equivalent prototypes
 - Microbenchmarks offer valuable insight
- Database workloads: more than 50% stalls
 - Mostly due to memory delays
 - Cannot always reduce stalls by increasing cache size
- Crucial bottlenecks
 - Data accesses to L2 cache (esp. for DSS)
 - Instruction accesses to L1 cache (esp. for OLTP)

How to Address Bottlenecks



Next: Optimizing cache accesses

Outline

- Introduction and Overview
- New Hardware
- Where Does Time Go?
- **Bridging the Processor/Memory Speed Gap**
- Hip and Trendy
- Directions for Future Research

This Section's Goals

- ❑ Survey techniques to improve locality
 - Relational data
 - Access methods
- ❑ Survey new query processing algorithms
- ❑ Present a new database system architecture
- ❑ Briefly explain Instruction Stream Optimizations

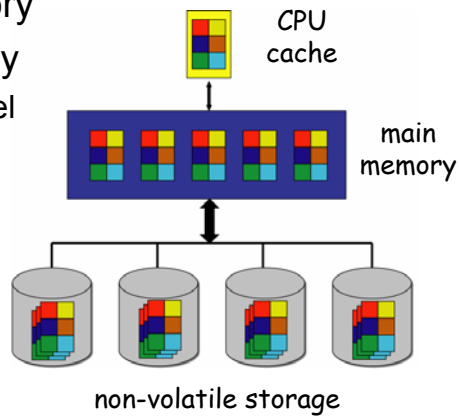
- ❑ Show how much good understanding of the platform can achieve

Outline

- ❑ Introduction and Overview
- ❑ New Hardware
- ❑ Where Does Time Go?
- ❑ **Bridging the Processor/Memory Speed Gap**
 - **Data Placement**
 - Access Methods
 - Query Processing
 - Instruction Stream Optimizations
 - Staged Database Systems
- ❑ Newer Hardware
- ❑ Hip and Trendy
- ❑ Directions for Future Research

Current Database Storage Managers

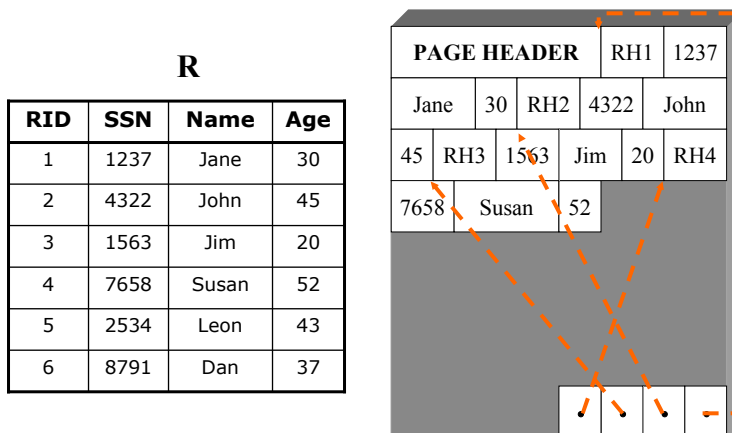
- ❑ Same layout on disk/memory
- ❑ Multi-level storage hierarchy
 - different devices at each level
 - different “optimal” access on each device
- ❑ Variable workloads and access patterns
 - OLTP: Full-record access
 - DSS: Partial-record access
 - no optimal “universal” layout



Goal: Reduce data traffic in memory hierarchy

“Classic” Data Layout on Disk Pages

- ❑ NSM (n-ary Storage Model, or Slotted Pages)



Records are stored sequentially
Attributes of a record are stored together

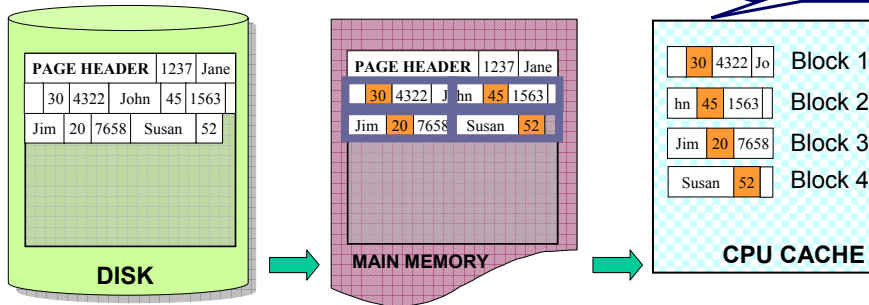
NSM in Memory Hierarchy



Query accesses all attributes (full-record access)

Query evaluates attribute "age" (partial-record access)

`select name
from R
where age > 50`



- ❑ Optimized for full-record access
- ❑ Slow partial-record access
 - ❑ Wastes I/O bandwidth (fixed page layout)
 - ❑ Low spatial locality at CPU cache

Decomposition Storage Model (DSM)

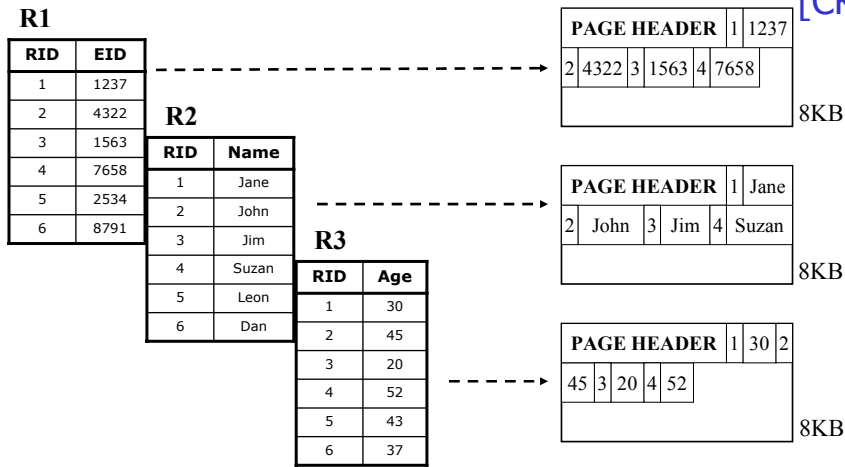
[CK85]

EID	Name	Age
1237	Jane	30
4322	John	45
1563	Jim	20
7658	Susan	52
2534	Leon	43
8791	Dan	37

Partition original table into n 1-attribute sub-tables

Decomposition Storage Model (DSM)

[CK85]



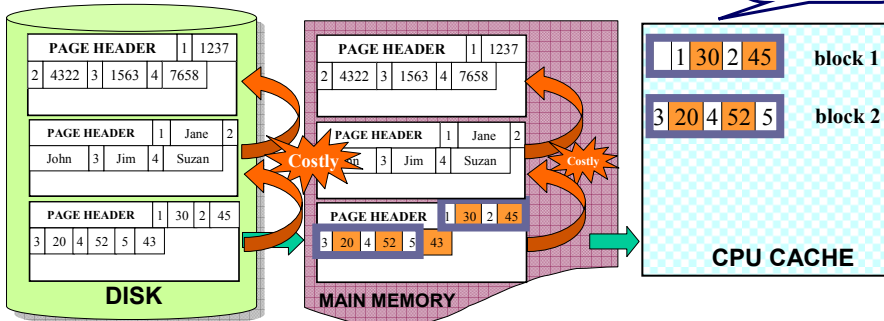
Partition original table into n 1-attribute sub-tables
Each sub-table stored separately in NSM pages

DSM in Memory Hierarchy

Query accesses all attributes (full-record access)

BEST Query accesses attribute "age" (partial-record access)

select name
from R
where age > 50



- ❑ Optimized for partial-record access
- ❑ Slow full-record access
 - ❑ Reconstructing full record may incur random I/O

Repairing NSM's cache performance

We need a data placement that...

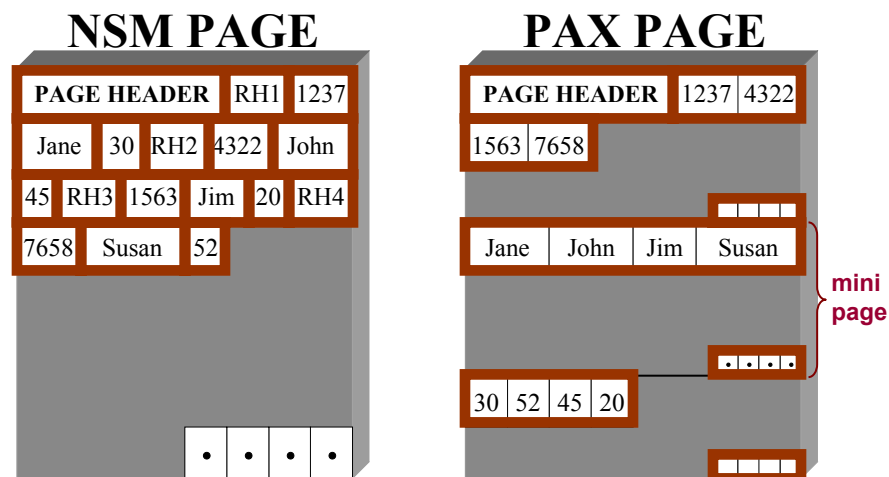
- ❑ Eliminates unnecessary memory accesses
- ❑ Improves inter-record locality
- ❑ Keeps a record's fields together
- ❑ Does not affect NSM's I/O performance

and, most importantly, is...

low-implementation-cost, high-impact

Partition Attributes Across (PAX)

[ADH01]



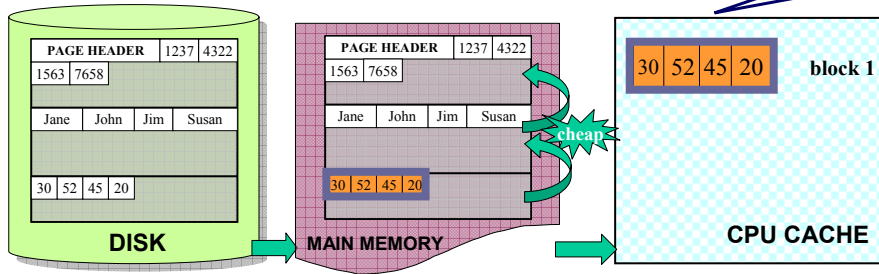
Idea: Partition data *within* page for spatial locality

PAX in Memory Hierarchy

[ADH01]

- BEST** Partial-record access in memory
- BEST** Full-record access on disk

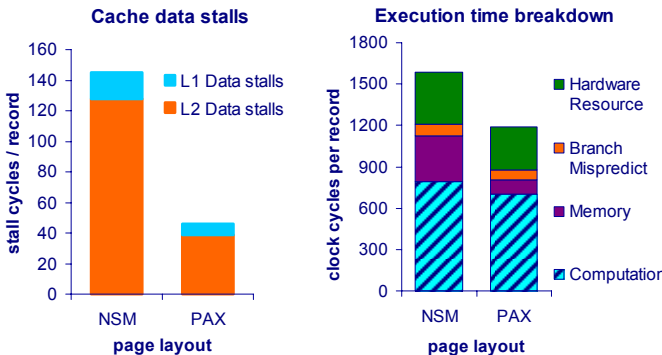
```
select name
from R
where age > 50
```



- Optimizes CPU cache-to-memory communication
- Retains NSM's I/O (page contents do not change)

PAX Performance Results (Shore)

[ADH01]



PII Xeon
Windows NT4
16KB L1-I&D,
512 KB L2,
512 MB RAM

```
Query:
select avg (ai)
from R
where aj >= Lo
and aj <= Hi
```

- Validation with microbenchmarks:
 - 70% less data stall time (only compulsory misses left)
 - Better use of processor's superscalar capability
- TPC-H performance: 15%-2x speedup in queries
 - Experiments with/without I/O, on three different processors

PAX eliminates unnecessary trips to memory

Dynamic PAX: Data Morphing

[HP03]

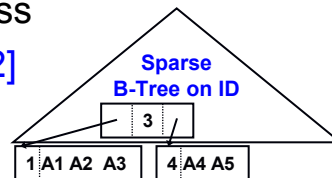
- ❑ PAX random access: more cache misses in record
- ❑ Store attributes accessed together contiguously
- ❑ Dynamic partition updates with changing workloads
 - Optimize total cost based on cache misses
 - Partition algorithms: naïve & hill-climbing algorithms
- ❑ Fewer cache misses
 - Better projectivity and scalability for index scan queries
 - Up to 45% faster than NSM & 25% faster than PAX
- ❑ Same I/O performance as PAX and NSM
- ❑ Future work: how to handle conflicts?

Alternatively: Repair DSM's I/O behavior

- ❑ We like DSM for partial record access
- ❑ We like NSM for full-record access

Solution: Fractured Mirrors [RDS02]

1. Get data placement right

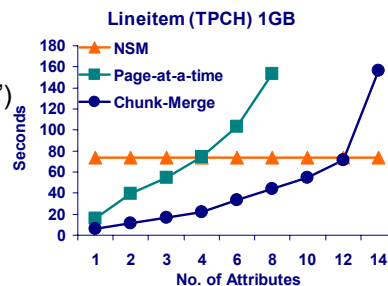


2. Faster record reconstruction

Instead of record- or page-at-a-time...

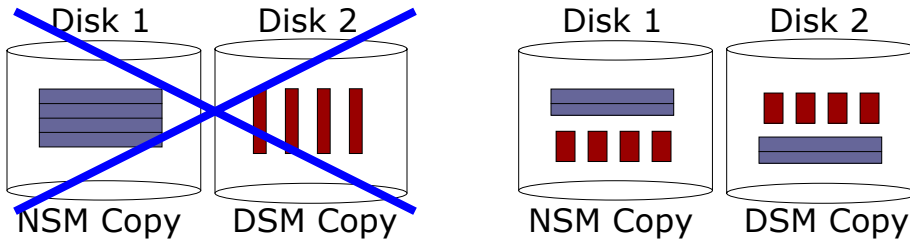
Chunk-based merge algorithm!

1. Read in segments of M pages (a “chunk”)
2. Merge segments in memory
3. Requires $(N \cdot K) / M$ disk seeks
4. For a memory budget of B pages, each partition gets B / N pages in a chunk



Fractured Mirrors

3. Smart mirroring



- ❑ Achieves 2-3x speedups on TPC-H
- ❑ Needs 2 copies of the database
- ❑ Future work:
 - A new optimizer
 - Smart buffer pool management
 - Updates

Summary (no replication)

Page layout	Cache-memory Performance		Memory-disk Performance	
	full-record access	partial record access	full-record access	partial record access
NSM	😊	😞	😊	😞
DSM	😞	😊	😞	😊
PAX	😊	😊	😊	😞

Need new placement method:

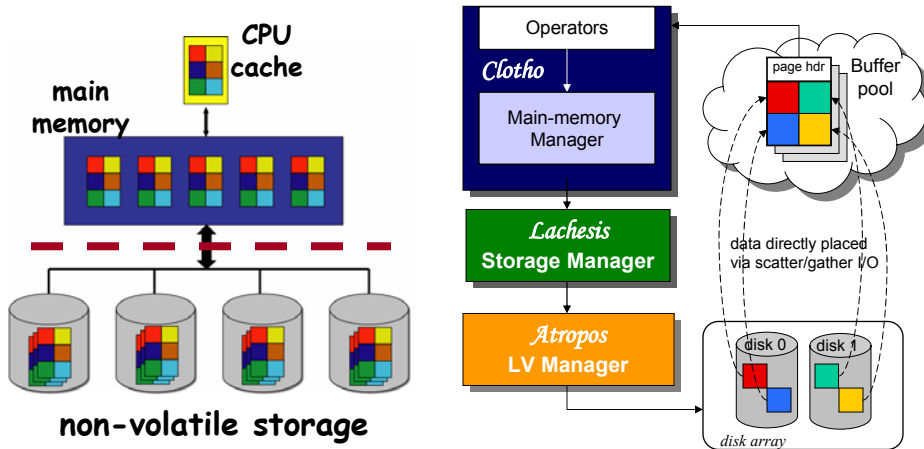
- ❑ Efficient full- and partial-record accesses
- ❑ Maximize utilization at all levels of memory hierarchy

**Difficult!!! Different devices/access methods
Different workloads on the same database**

The Fates Storage Manager

[SAG03,SSS04,SSS04a]

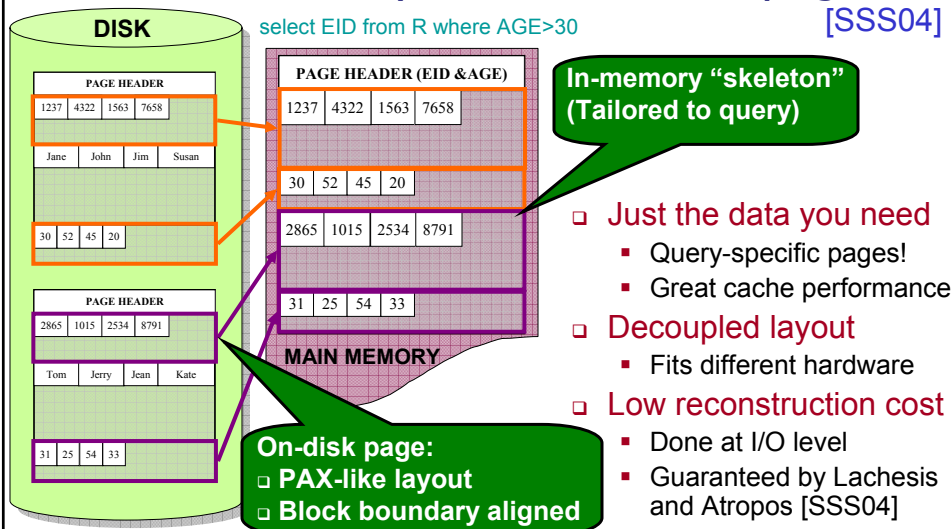
□ IDEA: Decouple layout!



Memory does not need to store full NSM pages

Clotho: memory stores PAX minipages

[SSS04]



In-memory "skeleton" (Tailored to query)

- Just the data you need
 - Query-specific pages!
 - Great cache performance
- Decoupled layout
 - Fits different hardware
- Low reconstruction cost
 - Done at I/O level
 - Guaranteed by Lachesis and Atropos [SSS04]

New buffer pool manager handles sharing

CSM: best-case performance of DSM and NSM

[SSS04]

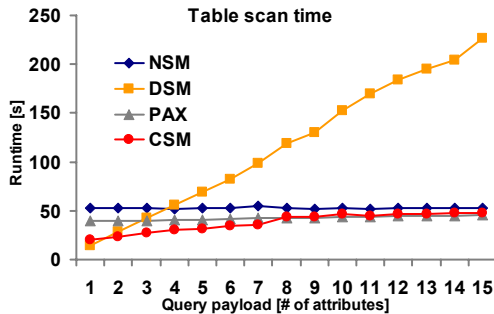


Table: a1 ... a15 (float)

Query: select a1, ...
from R
where a1 < Hi

Page layout	Cache-memory Performance		Memory-disk Performance	
	full-record access	partial record access	full-record access	partial record access
CSM	😊	😊	😊	😊

TPC-H: Outperform DSM by 20% to 2x
TPC-C: Comparable to NSM (6% lower throughput)

Data Placement: Summary

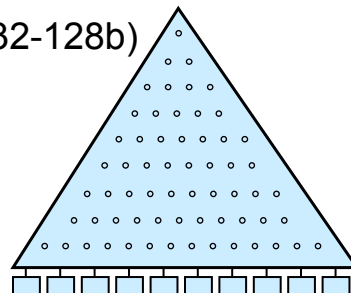
- ❑ Smart data placement increases spatial locality
 - Research targets table (relation) data
 - Goal: Reduce number of *non-cold cache misses*
- ❑ Techniques focus grouping attributes into cache lines for quick access
- ❑ **PAX, Data morphing: Cache optimization techniques**
- ❑ **Fractured Mirrors: Cache-and-disk optimization**
- ❑ **Fates DB Storage Manager: Independent data layout support across the entire memory hierarchy**

Outline

- Introduction and Overview
- New Hardware
- Where Does Time Go?
- **Bridging the Processor/Memory Speed Gap**
 - Data Placement
 - **Access Methods**
 - Query Processing
 - Instruction Stream Optimizations
 - Staged Database Systems
- Newer Hardware
- Hip and Trendy
- Directions for Future Research

Main-Memory Tree Indexes

- T Trees: proposed in mid-80s for MMDBs [LC86]
 - Aim: balance space overhead with searching time
 - Uniform memory access assumption (no caches)
- Main-memory B⁺ Trees: better cache performance [RR99]
- Node width = cache line size (32-128b)
 - Minimize number of cache misses for search
 - **Much higher** than traditional disk-based B-Trees
- So now trees are too deep

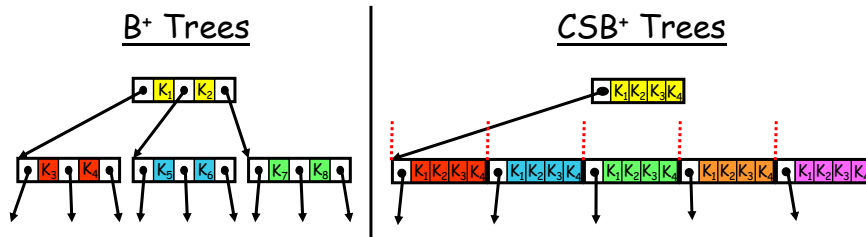


How to make trees shallower?

Reducing Pointers for Larger Fanout

[RR00]

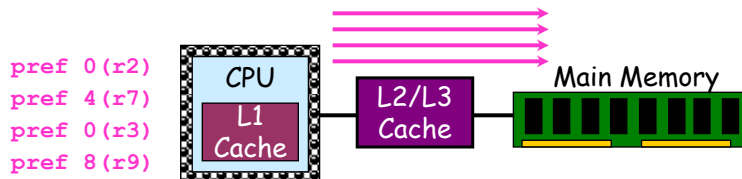
- ❑ Cache Sensitive B+ Trees (CSB+ Trees)
- ❑ Layout child nodes contiguously
- ❑ Eliminate all but one child pointers
 - *Double fanout of nonleaf node*



35% faster tree lookups
Update performance is 30% worse (splits)

What do we do with cold misses?

- ❑ Answer: **hide latencies using prefetching**
- ❑ Prefetching enabled by
 - Non-blocking cache technology
 - Prefetch assembly instructions
 - SGI R10000, Alpha 21264, Intel Pentium4

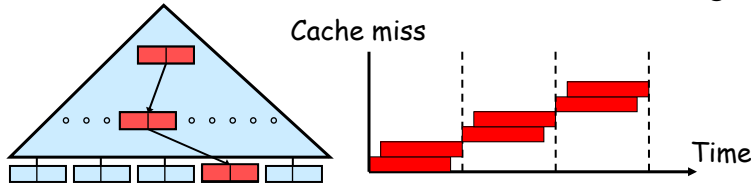


Prefetching hides cold cache miss latency
Efficiently used in pointer-chasing lookups!

Prefetching B+ Trees

[CGM01]

- ❑ (pB+ Trees) Idea: Larger nodes
- ❑ Node size = multiple cache lines (e.g. 8 lines)
 - Later corroborated by [HP03a]
- ❑ Prefetch all lines of a node before searching it



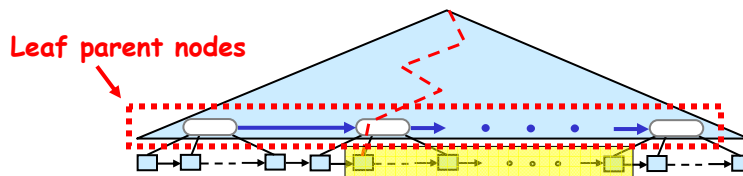
- ❑ **Cost to access a node only increases slightly**
- ❑ Much shallower trees, no changes required

**>2x better search AND update performance
Approach complementary to CSB+ Trees!**

Prefetching B+ Trees

[CGM01]

- ❑ Goal: faster range scan



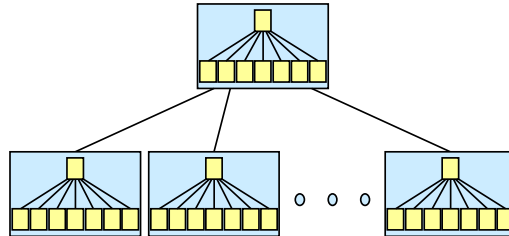
- ❑ Leaf parent nodes contain addresses of all leaves
- ❑ Link leaf parent nodes together
- ❑ Use this structure for prefetching leaf nodes

pB+ Trees: 8X speedup over B+ Trees

Fractal Prefetching B+ Trees

[CGM02]

- ❑ What if B+-tree does not fit in memory?
- ❑ (fpB+ Trees) Idea: Combine memory & disk trees



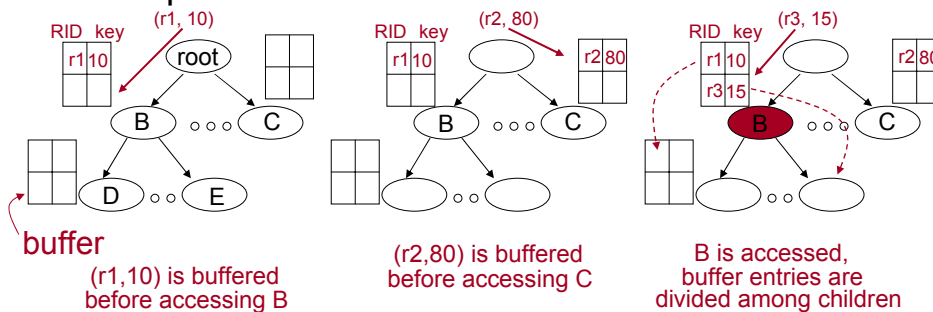
- ❑ Embed cache-optimized trees in disk tree nodes
- ❑ **fpB+ Trees optimize both cache AND disk**
- ❑ Key compression to increase fanout [BMR01]

Compared to disk-based B+ Trees, 80% faster in-memory searches with similar disk performance

Bulk lookups: Buffer Index Accesses

[ZR03a]

- ❑ Optimize data cache performance
- ❑ Similar technique in [PMH02]
- ❑ Idea: increase *temporal* locality by delaying (buffering) node probes until a group is formed
- ❑ Example: NLJ probe stream: (r1, 10) (r2, 80) (r3, 15)



3x speedup with enough concurrency

Access Methods: Summary

- ❑ Optimize B+ Tree pointer-chasing cache behavior
 - Reduce node size to few cache lines
 - Reduce pointers for larger fanout (CSB+)
 - “Next” pointers to lowest non-leaf level for easy prefetching (pB+)
 - Simultaneously optimize cache *and* disk (fpB+)
 - Bulk searches: Buffer index accesses

Additional work:

- ❑ Cache-oblivious B-Trees [BDF00]
 - Optimal bound in number of memory transfers
 - Regardless of # of memory levels, block size, or level speed
- ❑ Survey of techniques for B-Tree cache performance [GL01]
 - Existing heretofore-folkloric knowledge
 - Key normalization/compression, alignment, separating keys/pointers

**Lots more to be done in area – consider
interference and scarce resources**

Outline

- ❑ Introduction and Overview
- ❑ New Hardware
- ❑ Where Does Time Go?
- ❑ **Bridging the Processor/Memory Speed Gap**
 - Data Placement
 - Access Methods
 - **Query Processing**
 - Staged Database Systems
 - Instruction Stream Optimizations
- ❑ Newer Hardware
- ❑ Hip and Trendy
- ❑ Directions for Future Research

Query Processing Algorithms

Idea: Adapt query processing algorithms to caches

Related work includes:

- ❑ Improving data cache performance
 - Sorting
 - Join
- ❑ Improving instruction cache performance
 - DSS applications

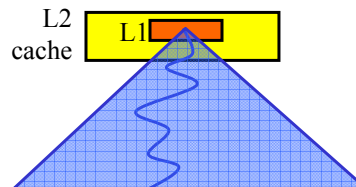
Sorting

[NBC94]

- ❑ In-memory sorting / generating runs
- ❑ AlphaSort



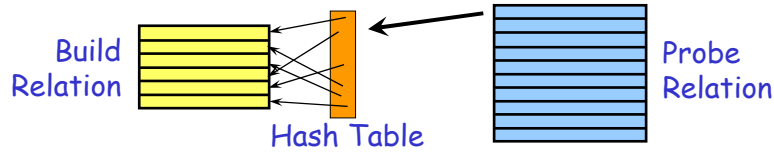
Quick Sort



Replacement-selection

- ❑ Use quick sort rather than replacement selection
 - Sequential vs. random access
 - No cache misses after sub-arrays fit in cache
- ❑ Sort (key-prefix, pointer) pairs rather than records
 - 3x cpu speedup for the Datamation benchmark

Hash Join

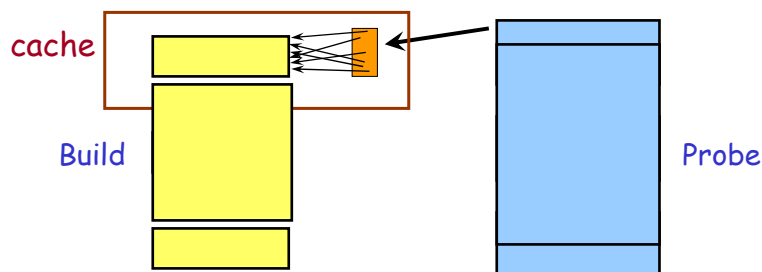


- ❑ Random accesses to hash table
 - Both when building AND when probing!!!
- ❑ Poor cache performance
 - $\geq 73\%$ of user time is CPU cache stalls [CAG04]
- ☞ Approaches to improving cache performance
 - Cache partitioning – maximizes locality
 - Prefetching – hides latencies

Reducing non-cold misses

[SKN94]

- ❑ Idea: *Cache partitioning* (similar to I/O partitioning)
 - Divide relations into cache-sized partitions
 - Fit build partition and hash table into cache
 - Avoid cache misses for hash table visits

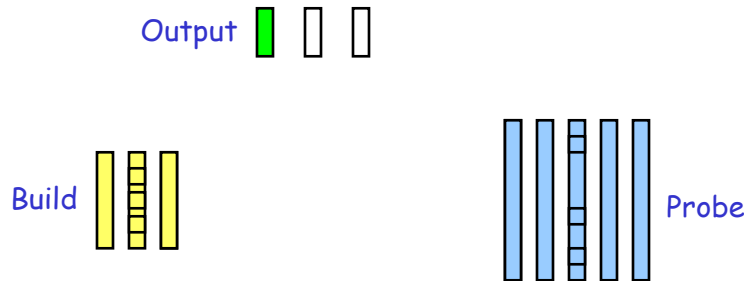


1/3 fewer cache misses, 9.3% speedup
>50% misses due to partitioning overhead

Hash Joins in Monet

[B02]

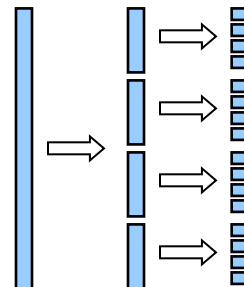
- ❑ Monet main-memory database system [B02]
 - Vertically partitioned tuples (DSM)
- ❑ Join two vertically partitioned relations
 - Join two join-attribute arrays [BMK99,MBK00]
 - Extract other fields for output relation [MBN04]



Monet: Reducing Partition Cost

[BMK99
MBK00]

- ❑ Join two arrays of simple fields (8 byte tuples)
 - Original cache partitioning is single pass
 - TLB thrashing if # partitions > # TLB entries
 - Cache thrashing if # partitions > # lines in cache
- ❑ Solution: multiple passes
 - # partitions per pass is small
 - Radix-cluster [BMK99,MBK00]
 - Use different bits of hashed keys for different passes
 - E.g. In figure, use 2 bits of hashed keys for each pass
- ❑ Plus CPU optimizations
 - XOR instead of %



2-pass partition

Up to 2.7X speedup on an Origin 2000
Results most significant for small tuples

Monet: Extracting Payload

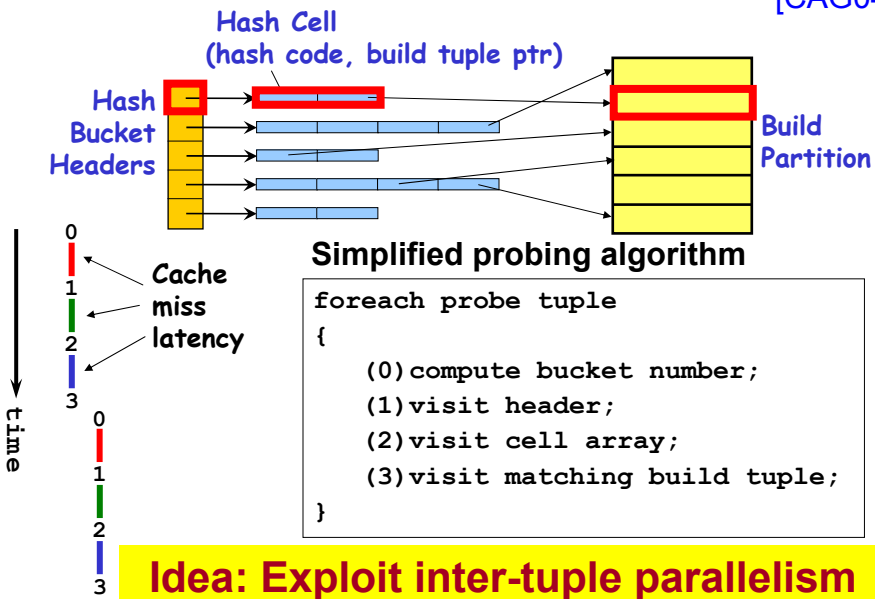
[MBN04]

- ❑ Two ways to extract payload:
 - Pre-projection: copy fields during cache partitioning
 - Post-projection: generate join index, then extract fields
- ❑ Monet: post-projection
 - Radix-decluster algorithm for good cache performance
- ❑ Post-projection good for DSM
 - Up to 2X speedup compared to pre-projection
- ❑ Post-projection is not recommended for NSM
 - Copying fields during cache partitioning is better

Paper presented in this conference!

Optimizing non-DSM hash joins

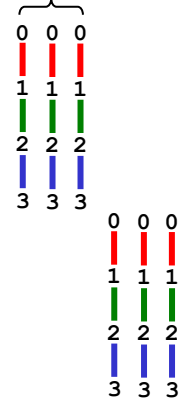
[CAG04]



Group Prefetching

[CAG04]

a group



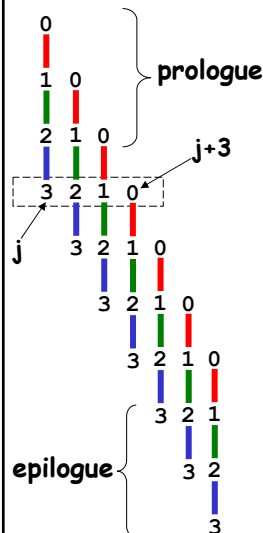
```

foreach group of probe tuples {
  foreach tuple in group {
    (0) compute bucket number;
    prefetch header;
  }
  foreach tuple in group {
    (1) visit header;
    prefetch cell array;
  }
  foreach tuple in group {
    (2) visit cell array;
    prefetch build tuple;
  }
  foreach tuple in group {
    (3) visit matching build tuple;
  }
}

```

Software Pipelining

[CAG04]



```

Prologue;
for j=0 to N-4 do {
  tuple j+3:
    (0) compute bucket number;
    prefetch header;

  tuple j+2:
    (1) visit header;
    prefetch cell array;

  tuple j+1:
    (2) visit cell array;
    prefetch build tuple;

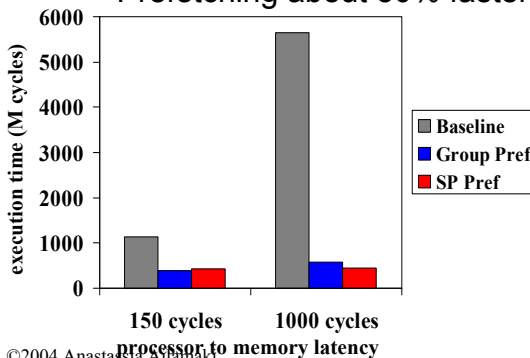
  tuple j:
    (3) visit matching build tuple;
}
Epilogue;

```

Prefetching: Performance Results

[CAG04]

- ❑ Techniques exhibit similar performance
- ❑ Group prefetching easier to implement
- ❑ Compared to cache partitioning:
 - Cache partitioning costly when tuples are large (>20b)
 - Prefetching about 50% faster than cache partitioning



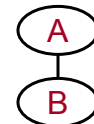
❑ **9X speedups over baseline at 1000 cycles**

❑ **Absolute numbers do not change!**

DSS: Reducing I-misses

[PMA01,ZR04]

- ❑ Demand-pull execution model: one tuple at a time
 - ABABABABABABABABAB...
 - If $A + B > L1$ instruction cache size
 - **Poor instruction cache utilization!**



Query Plan

- ❑ Solution: multiple tuples at an operator
 - ABBBBBAAAAABBBBBB...

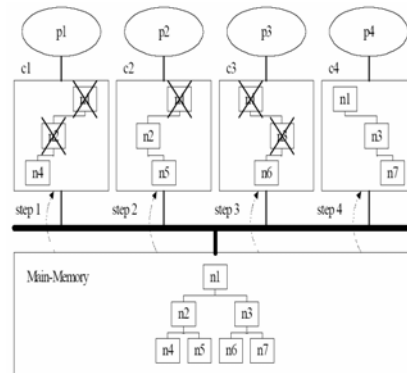
- ❑ Modify operators to support block of tuples [PMA01]
- ❑ Insert “buffer” operators between A and B [ZR04]
 - “buffer” calls B multiple times
 - Stores intermediate tuple pointers to serve A’s request
 - No need to change original operators

12% speedup for simple TPC-H queries

Concurrency Control

[CHK01]

- ❑ Multiple CPUs share a tree
- ❑ Lock coupling: too much cost
 - Latching a node means writing
 - True even for readers !!!
 - Coherence cache misses due to writes from different CPUs



- ❑ Solution:
 - Optimistic approach for readers
 - Updaters still latch nodes
 - Updaters also set node versions
 - Readers check version to ensure correctness

**Search throughput: 5x (=no locking case)
Update throughput: 4x**

Query processing: summary

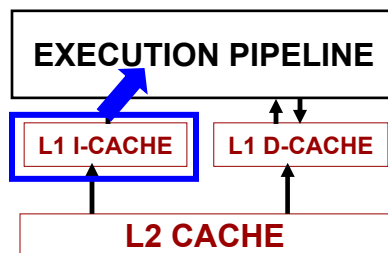
- ❑ Alphasort: use quicksort and key prefix-pointer
- ❑ Monet: MM-DBMS uses aggressive DSM
 - Optimize partitioning with hierarchical radix-clustering
 - Optimize post-projection with radix-declustering
 - Many other optimizations
- ❑ Traditional hash joins: aggressive prefetching
 - Efficiently hides data cache misses
 - Robust performance with future long latencies
- ❑ DSS I-misses: group computation (new operator)
- ❑ B-tree concurrency control: reduce readers' latching

Outline

- Introduction and Overview
- New Hardware
- Where Does Time Go?
- **Bridging the Processor/Memory Speed Gap**
 - Data Placement
 - Access Methods
 - Query Processing
 - **Instruction Stream Optimizations**
 - Staged Database Systems
- Newer Hardware
- Hip and Trendy
- Directions for Future Research

Instruction-Related Stalls

- 25-40% of execution time [KPH98, HA04]
- Recall importance of instruction cache: In the critical execution path!



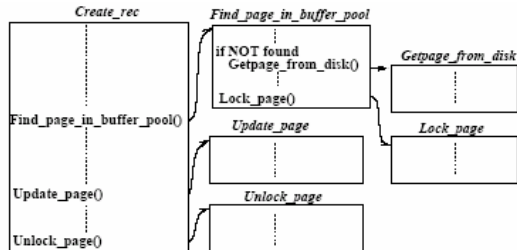
Impossible to overlap I-cache delays

Call graph prefetching for DB apps

[APD03]

- Goal: improve DSS I-cache performance
- Idea: Predict next function call using small cache

□ Example: *create_rec* always calls *find_*, *lock_*, *update_*, and *unlock_page* in same order



- Experiments: Shore on SimpleScalar Simulator
 - Running Wisconsin Benchmark

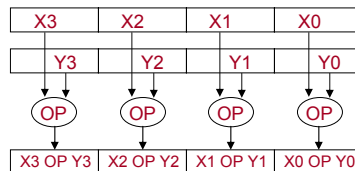
Beneficial for predictable DSS streams

DB operators using SIMD

[ZR02]

- SIMD: Single – Instruction – Multiple – Data
In modern CPUs, target multimedia apps

□ Example: Pentium 4, 128-bit SIMD register holds four 32-bit values

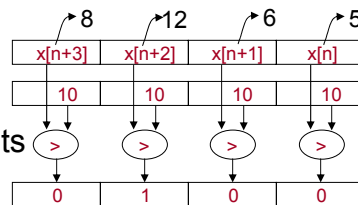


- Assume data stored columnwise as contiguous array of fixed-length numeric values (e.g., PAX)

□ Scan example:

if $x[n] > 10$
 $\text{result}[\text{pos}++] = x[n]$
 original scan code

SIMD 1st phase:
 produce bitmap vector with 4 comparison results in parallel



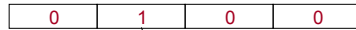
DB operators using SIMD

- Scan example (cont'd)

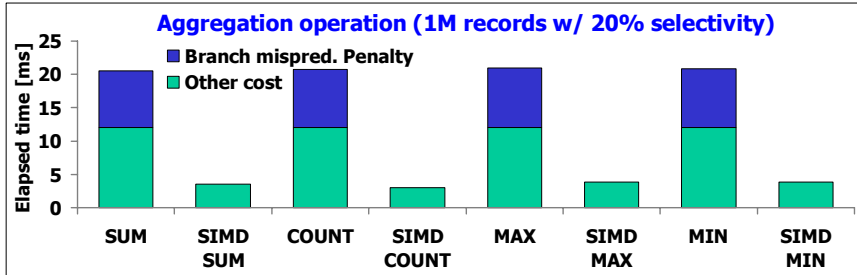
SIMD 2nd phase:

if bit_vector == 0, continue

else copy all 4 results, increase pos when bit==1



keep this result

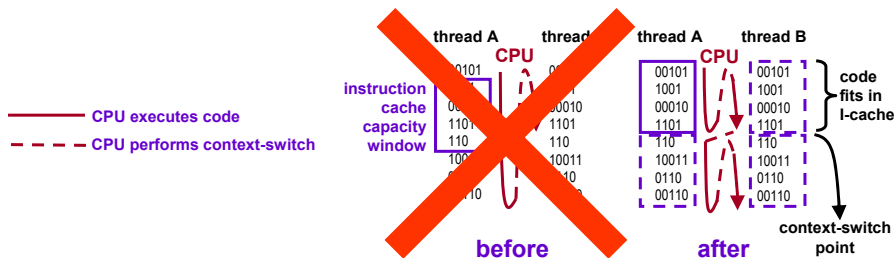


- Parallel comparisons, fewer branches ⇒ fewer mispredictions

Superlinear speedup to # of parallelism
Need to rewrite code to use SIMD

STEPS: Cache-Resident OLTP

- Targets instruction-cache performance for OLTP
- Exploits high transaction concurrency
- Synchronized Transactions through Explicit Processor Scheduling: Multiplex concurrent transactions to exploit common code paths



All capacity/conflict I-cache misses gone!

STEPS: Cache-Resident OLTP

- ❑ STEPS implementation runs full OLTP workloads (TPC-C)
- ❑ Groups threads per DB operator, then uses fast context-switch to reuse instructions in the cache

- ❑ **Full-system TPC-C implementation:**
- ❑ **65% fewer L1-I misses, 40% speedup**

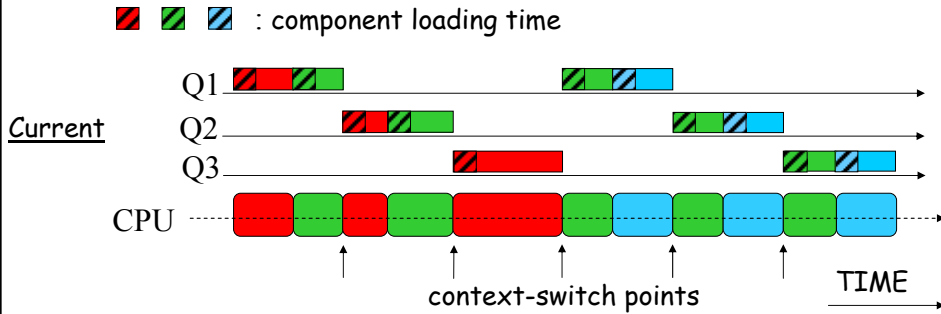
**STEPS minimizes L1-I cache misses
without increasing cache size**

Outline

- ❑ Introduction and Overview
- ❑ New Hardware
- ❑ Where Does Time Go?
- ❑ **Bridging the Processor/Memory Speed Gap**
 - Data Placement
 - Access Methods
 - Query Processing
 - Instruction Stream Optimizations
 - **Staged Database Systems**
- ❑ Newer Hardware
- ❑ Hip and Trendy
- ❑ Directions for Future Research

Thread-based concurrency pitfalls

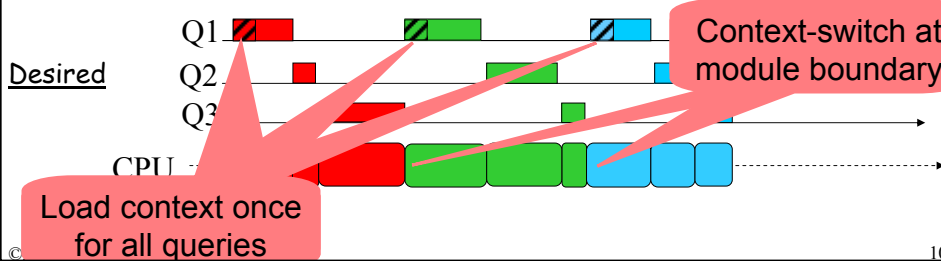
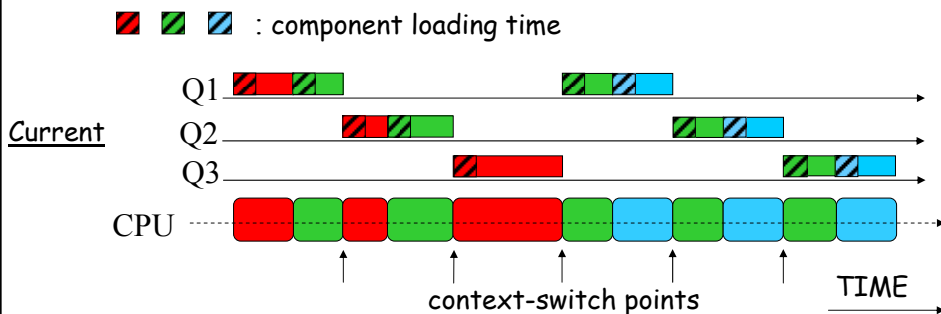
[HA03]



**Context loaded multiple times for each query
No means to exploit overlapping work**

Thread-based concurrency pitfalls

[HA03]



Staged Database Systems

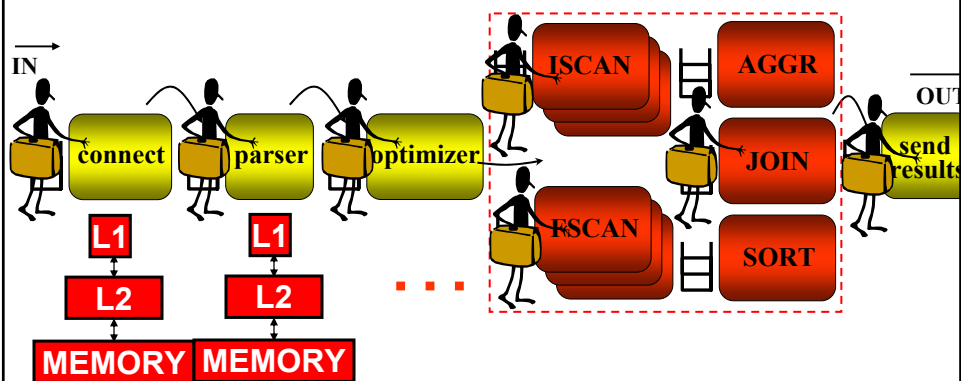
[HA03]

- ❑ Staged software design allows for
 - Cohort scheduling of queries to amortize loading time
 - Suspend at module boundaries to maintain context
- ❑ Break DBMS into stages
- ❑ Stages act as independent servers
- ❑ Queries exist in the form of “packets”

- ❑ Proposed query scheduling algorithms to address locality/wait time tradeoffs [HA02]

Staged Database Systems

[HA03]



Optimize instruction/data cache locality
Naturally enable multi-query processing
Highly scalable, fault-tolerant, trustworthy

Summary: Bridging the Gap

- ❑ Cache-aware data placement
 - Eliminates unnecessary trips to memory
 - Minimizes conflict/capacity misses
 - Fates: decouple memory from storage layout
- ❑ What about compulsory (cold) misses?
 - Can't avoid, but can hide latency with prefetching
 - Techniques for B-trees, hash joins
- ❑ Staged Database Systems: a scalable future
- ❑ Addressing instruction stalls
 - DSS: Call Graph Prefetching, SIMD, *group* operator
 - OLTP: STEPS, a promising direction for any platform

Outline

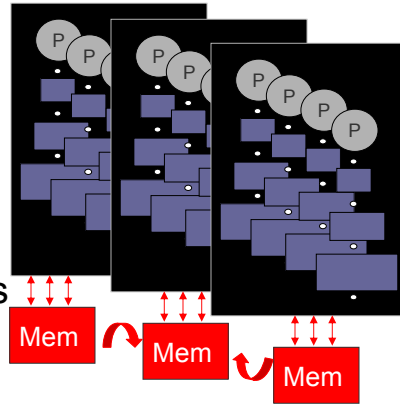
- ❑ Introduction and Overview
- ❑ New Hardware
- ❑ Where Does Time Go?
- ❑ Bridging the Processor/Memory Speed Gap
- ❑ **Newer Hardware**
- ❑ Hip and Trendy
- ❑ Directions for Future Research

Current/Near-future Multiprocessors

Typical platforms:

1. Chips with multiple cores
2. Servers with multiple chips
3. Memory shared across

Multiprocessor Server



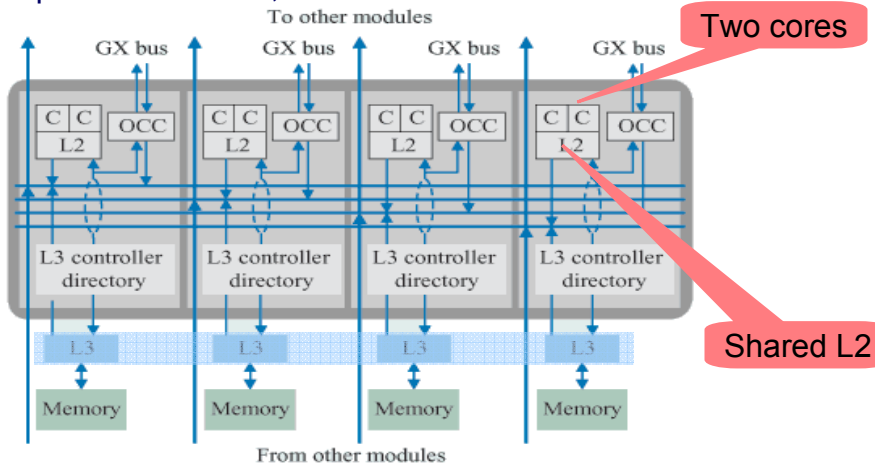
Memory access:

- ❑ Traverse multiple hierarchies
- ❑ Large non-uniform latencies

Programmer/Software must Hide/Tolerate Latency

Chip Multi-Processors (CMP)

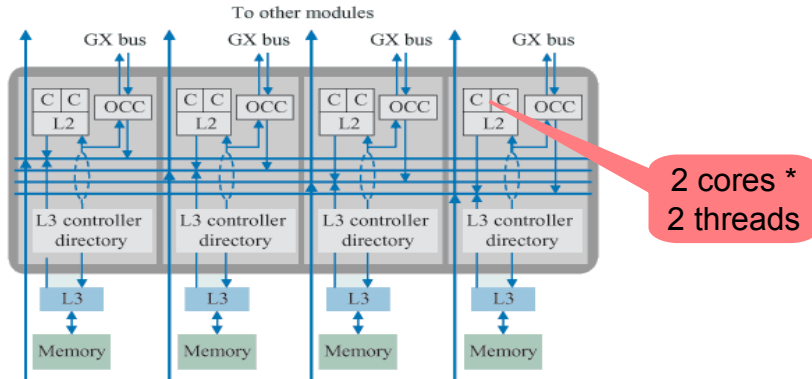
Example: IBM Power4, Power5



Highly variable memory latency
Speedup: OLTP 3x, DSS 2.3x on Piranha [BGM00]

Simultaneous Multi-Threading (SMT)

- ❑ Implements threads in a superscalar processor
- ❑ Keeps hardware state for multiple threads
- ❑ E.g.: Intel Pentium 4 (SMT), IBM Power5 (SMT&CMP)



Speedup: OLTP 3x, DSS 0.5x (simulated) [LBE98]

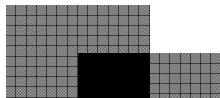
Outline

- ❑ Introduction and Overview
- ❑ New Hardware
- ❑ Where Does Time Go?
- ❑ Bridging the Processor/Memory Speed Gap
- ❑ **Hip and Trendy**
 - **Query co-processing**
 - Databases on MEMStore
- ❑ Directions for Future Research

Optimizing Spatial Operations

[SAA03]

- ❑ Spatial operation is computation intensive
 - Intersection, distance computation
 - Number of vertices per object \uparrow , cost \uparrow
- ❑ Use graphics card to increase speed
- ❑ Idea: use color blending to detect intersection
 - Draw each polygon with gray
 - Intersected area is black because of color mixing effect
 - Algorithms cleverly use hardware features



Intersection selection: up to 64% improvement using graphics card

Fast Computation of DB Operations Using Graphics Processors

[GLW04]

- ❑ Exploit graphics features for database operations
 - Predicate, Boolean operations, Aggregates
- ❑ Examples:
 - Predicate: attribute > constant
 - Graphics: test a set of pixels against a reference value
 - pixel = attribute value, reference value = constant
 - Aggregations: COUNT
 - Graphics: count number of pixels passing a test
- ❑ Good performance: e.g. over 2X improvement for predicate evaluations

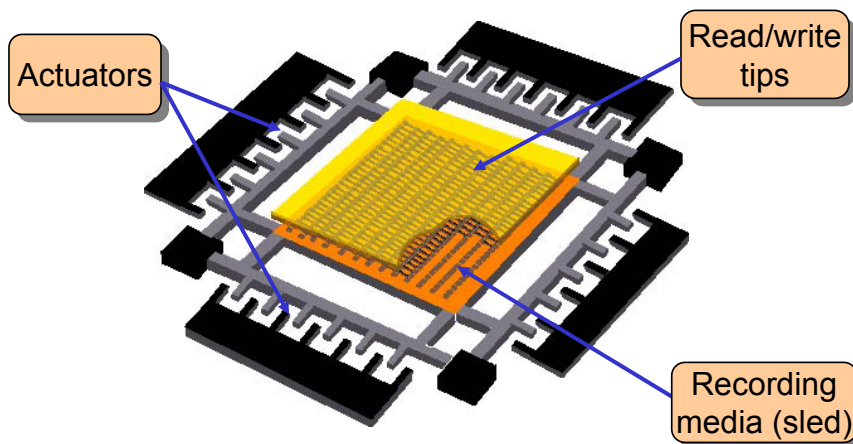
Promising! Peak performance of graphics processor increases 2.5-3 times a year

Outline

- Introduction and Overview
- New Hardware
- Where Does Time Go?
- Bridging the Processor/Memory Speed Gap
- **Hip and Trendy**
 - Query co-processing
 - **Databases on MEMStore**
- Directions for Future Research

MEMStore (MEMS*-based storage)

- On-chip mechanical storage - using MEMS for media positioning



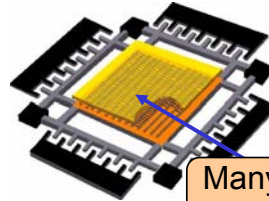
* microelectromechanical systems

MEMStore (MEMS*-based storage)

* microelectromechanical systems



Single read/write head



Many parallel heads

- ❑ 60 - 200 GB capacity
 - 4 - 40 GB portable
- ❑ 100 cm³ volume
- ❑ 10's MB/s bandwidth
- ❑ < 10 ms latency
 - 10 - 15 ms portable
- ❑ 2 - 10 GB capacity
- ❑ < 1 cm³ volume
- ❑ ~100 MB/s bandwidth
- ❑ < 1 ms latency

So how can MEMS help improve DB performance?

Two-dimensional database access

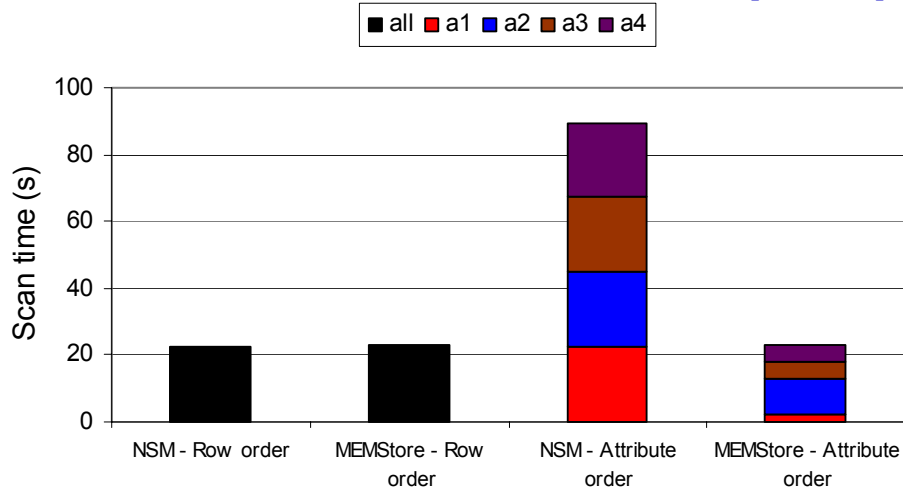
[SSA03, YAA03, YAA04]

		Attributes								
Records		0	33	54	1	34	55	2	35	56
		3	30	57	4	31	58	5	32	59
		6	27	60	7	28	61	8	29	62
		15	36	69	16	37	70	17	38	71
		12	39	66	13	40	67	14	41	68
		9	42	63	10	43	64	11	44	65
		18	51	72	19	52	73	20	53	74
		21	48	75	22	49	76	23	50	77
		24	45	78	25	46	79	26	47	80

Exploit inherent parallelism

Two-dimensional database access

[SSA03]



Peak performance along both dimensions

Outline

- Introduction and Overview
- New Hardware
- Where Does Time Go?
- Bridging the Processor/Memory Speed Gap
- Hip and Trendy
- **Directions for Future Research**

Future research directions

- ❑ Rethink Query Optimization – with increasing complexity, cost-based optimization not ideal
- ❑ Multiprocessors and really new modular software architectures to fit new computers
 - Current research in DB workloads only scratches surface
 - Optimize execution on multiple-core chips
 - Exploit multithreaded processors
- ❑ Power-aware database systems
 - On embedded processors, laptops, etc.
- ❑ Automatic data placement and memory layer optimization – one level should not need to know what others do
 - Auto-everything
- ❑ Aggressive use of hybrid processors

ACKNOWLEDGEMENTS

Special thanks go to...



- **Shimin Chen, Minglong Shao, Stavros Harizopoulos, and Nikos Hardavellas** for invaluable contributions to this talk
- **Steve Schlosser** (MEMStore)
- **Ravi Ramamurthy** (fractured mirrors)
- **Babak Falsafi** and **Chris Colohan** (h/w architecture)



REFERENCES

(used in presentation)

References

Where Does Time Go? (simulation only)

- [ADS02] **Branch Behavior of a Commercial OLTP Workload on Intel IA32 Processors.** M. Annavaram, T. Diep, J. Shen. *International Conference on Computer Design: VLSI in Computers and Processors (ICCD)*, Freiburg, Germany, September 2002.
- [SBG02] **A Detailed Comparison of Two Transaction Processing Workloads.** R. Stets, L.A. Barroso, and K. Gharachorloo. *IEEE Annual Workshop on Workload Characterization (WWC)*, Austin, Texas, November 2002.
- [BGN00] **Impact of Chip-Level Integration on Performance of OLTP Workloads.** L.A. Barroso, K. Gharachorloo, A. Nowatzky, and B. Verghese. *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Toulouse, France, January 2000.
- [RGA98] **Performance of Database Workloads on Shared Memory Systems with Out-of-Order Processors.** P. Ranganathan, K. Gharachorloo, S. Adve, and L.A. Barroso. *International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, California, October 1998.
- [LBE98] **An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors.** J. Lo, L.A. Barroso, S. Eggers, K. Gharachorloo, H. Levy, and S. Parekh. *ACM International Symposium on Computer Architecture (ISCA)*, Barcelona, Spain, June 1998.
- [EJL96] **Evaluation of Multithreaded Uniprocessors for Commercial Application Environments.** R.J. Eickemeyer, R.E. Johnson, S.R. Kunkel, M.S. Squillante, and S. Liu. *ACM International Symposium on Computer Architecture (ISCA)*, Philadelphia, Pennsylvania, May 1996.

References

Where Does Time Go? (real-machine/simulation)

- [RAD02] **Comparing and Contrasting a Commercial OLTP Workload with CPU2000.** J. Rupley II, M. Annavaram, J. DeVale, T. Diep and B. Black (Intel). *IEEE Annual Workshop on Workload Characterization (WWC)*, Austin, Texas, November 2002.
- [CTT99] **Detailed Characterization of a Quad Pentium Pro Server Running TPC-D.** Q. Cao, J. Torrellas, P. Trancoso, J. Larriba-Pey, B. Knighten, Y. Won. *International Conference on Computer Design (ICCD)*, Austin, Texas, October 1999.
- [ADH99] **DBMSs on a Modern Processor: Experimental Results** A. Ailamaki, D. J. DeWitt, M. D. Hill, D.A. Wood. *International Conference on Very Large Data Bases (VLDB)*, Edinburgh, UK, September 1999.
- [KPH98] **Performance Characterization of a Quad Pentium Pro SMP using OLTP Workloads.** K. Keeton, D.A. Patterson, Y.Q. He, R.C. Raphael, W.E. Baker. *ACM International Symposium on Computer Architecture (ISCA)*, Barcelona, Spain, June 1998.
- [BGB98] **Memory System Characterization of Commercial Workloads.** L.A. Barroso, K. Gharachorloo, and E. Bugnion. *ACM International Symposium on Computer Architecture (ISCA)*, Barcelona, Spain, June 1998.
- [TLZ97] **The Memory Performance of DSS Commercial Workloads in Shared-Memory Multiprocessors.** P. Trancoso, J. Larriba-Pey, Z. Zhang, J. Torrellas. *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, San Antonio, Texas, February 1997.

Architecture-Conscious Data Placement

- [SSS04] **Clotho: Decoupling memory page layout from storage organization.** M. Shao, J. Schindler, S.W. Schlosser, A. Ailamaki, G.R. Ganger. *International Conference on Very Large Data Bases (VLDB), Toronto, Canada, September 2004.*
- [SSS04a] **Atropos: A Disk Array Volume Manager for Orchestrated Use of Disks.** J. Schindler, S.W. Schlosser, M. Shao, A. Ailamaki, G.R. Ganger. *USENIX Conference on File and Storage Technologies (FAST), San Francisco, California, March 2004.*

Architecture-Conscious Access Methods

- [ZR03a] **Buffering Accesses to Memory-Resident Index Structures.** J. Zhou and K.A. Ross. *International Conference on Very Large Data Bases (VLDB), Berlin, Germany, September 2003.*
- [HP03a] **Effect of node size on the performance of cache-conscious B+ Trees.** R.A. Hankins and J.M. Patel. *ACM International conference on Measurement and Modeling of Computer Systems (SIGMETRICS), San Diego, California, June 2003.*
- [CGM02] **Fractal Prefetching B+ Trees: Optimizing Both Cache and Disk Performance.** S. Chen, P.B. Gibbons, T.C. Mowry, and G. Valentin. *ACM International Conference on Management of Data (SIGMOD), Madison, Wisconsin, June 2002.*
- [GL01] **B-Tree Indexes and CPU Caches.** G. Graefe and P. Larson. *International Conference on Data Engineering (ICDE), Heidelberg, Germany, April 2001.*
- [CGM01] **Improving Index Performance through Prefetching.** S. Chen, P.B. Gibbons, and T.C. Mowry. *ACM International Conference on Management of Data (SIGMOD), Santa Barbara, California, May 2001.*
- [BMR01] **Main-memory index structures with fixed-size partial keys.** P. Bohannon, P. McIlroy, and R. Rastogi. *ACM International Conference on Management of Data (SIGMOD), Santa Barbara, California, May 2001.*
- [BDF00] **Cache-Oblivious B-Trees.** M.A. Bender, E.D. Demaine, and M. Farach-Colton. *Symposium on Foundations of Computer Science (FOCS), Redondo Beach, California, November 2000.*
- [RR00] **Making B+ Trees Cache Conscious in Main Memory.** J. Rao and K.A. Ross. *ACM International Conference on Management of Data (SIGMOD), Dallas, Texas, May 2000.*
- [RR99] **Cache Conscious Indexing for Decision-Support in Main Memory.** J. Rao and K.A. Ross. *International Conference on Very Large Data Bases (VLDB), Edinburgh, the United Kingdom, September 1999.*
- [LC86] **Query Processing in main-memory database management systems.** T. J. Lehman and M. J. Carey. *ACM International Conference on Management of Data (SIGMOD), 1986.*

Architecture-Conscious Query Processing

- [MBN04] **Cache-Conscious Radix-Decluster Projections.** Stefan Manegold, Peter A. Boncz, Niels Nes, Martin L. Kersten. *In Proceedings of the International Conference on Very Large Data Bases (VLDB), Toronto, Canada, September 2004.*
- [GLW04] **Fast Computation of Database Operations using Graphics Processors.** N.K. Govindaraju, B. Lloyd, W. Wang, M. Lin, D. Manocha. *ACM International Conference on Management of Data (SIGMOD), Paris, France, June 2004.*
- [CAG04] **Improving Hash Join Performance through Prefetching.** S. Chen, A. Ailamaki, P. B. Gibbons, and T.C. Mowry. *International Conference on Data Engineering (ICDE), Boston, Massachusetts, March 2004.*
- [ZR04] **Buffering Database Operations for Enhanced Instruction Cache Performance.** J. Zhou, K. A. Ross. *ACM International Conference on Management of Data (SIGMOD), Paris, France, June 2004.*
- [SAA03] **Hardware Acceleration for Spatial Selections and Joins.** C. Sun, D. Agrawal, A.E. Abbadi. *ACM International conference on Management of Data (SIGMOD), San Diego, California, June, 2003.*
- [CHK01] **Cache-Conscious Concurrency Control of Main-Memory Indexes on Shared-Memory Multiprocessor Systems.** S. K. Cha, S. Hwang, K. Kim, and K. Kwon. *International Conference on Very Large Data Bases (VLDB), Rome, Italy, September 2001.*
- [PMA01] **Block Oriented Processing of Relational Database Operations in Modern Computer Architectures.** S. Padmanabhan, T. Malkemus, R.C. Agarwal, A. Jhingran. *International Conference on Data Engineering (ICDE), Heidelberg, Germany, April 2001.*
- [MBK00] **What Happens During a Join? Dissecting CPU and Memory Optimization Effects.** S. Manegold, P.A. Boncz, and M.L. Kersten. *International Conference on Very Large Data Bases (VLDB), Cairo, Egypt, September 2000.*
- [SKN94] **Cache Conscious Algorithms for Relational Query Processing.** A. Shatdal, C. Kant, and J.F. Naughton. *International Conference on Very Large Data Bases (VLDB), Santiago de Chile, Chile, September 1994.*
- [NBC94] **AlphaSort: A RISC Machine Sort.** C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D.B. Lomet. *ACM International Conference on Management of Data (SIGMOD), Minneapolis, Minnesota, May 1994.*

Instrution Stream Optimizations and
DBMS Architectures

- [HA04] **STEPS towards Cache-resident Transaction Processing.** S. Harizopoulos and A. Ailamaki. *International Conference on Very Large Data Bases (VLDB), Toronto, Canada, September 2004.*
- [APD03] **Call Graph Prefetching for Database Applications.** M. Annavaram, J.M. Patel, and E.S. Davidson. *ACM Transactions on Computer Systems, 21(4):412-444, November 2003.*
- [SAG03] **Lachesis: Robust Database Storage Management Based on Device-specific Performance Characteristics.** J. Schindler, A. Ailamaki, and G. R. Ganger. *International Conference on Very Large Data Bases (VLDB), Berlin, Germany, September 2003.*
- [HA02] **Affinity Scheduling in Staged Server Architectures.** S. Harizopoulos and A. Ailamaki. *Carnegie Mellon University, Technical Report CMU-CS-02-113, March, 2002.*
- [HA03] **A Case for Staged Database Systems.** S. Harizopoulos and A. Ailamaki. *Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, January 2003.*
- [B02] **Monet: A Next-Generation DBMS Kernel For Query-intensive Applications.** P. A. Boncz. *Ph.D. Thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, May 2002.*
- [PMH02] **Computation Regrouping: Restructuring Programs for Temporal Data Cache Locality.** V.K. Pingali, S.A. McKee, W.C. Hsieh, and J.B. Carter. *International Conference on Supercomputing (ICS), New York, New York, June 2002.*
- [ZR02] **Implementing Database Operations Using SIMD Instructions.** J. Zhou and K.A. Ross. *ACM International Conference on Management of Data (SIGMOD), Madison, Wisconsin, June 2002.*

References

Newer Hardware

- [BWS03] **Improving the Performance of OLTP Workloads on SMP Computer Systems by Limiting Modified Cache Lines.** J.E. Black, D.F. Wright, and E.M. Salgueiro. *IEEE Annual Workshop on Workload Characterization (WWC)*, Austin, Texas, October 2003.
- [GH03] **Technological impact of magnetic hard disk drives on storage systems.** E. Grochowski and R. D. Halem *IBM Systems Journal* 42(2), 2003.
- [DJN02] **Shared Cache Architectures for Decision Support Systems.** M. Dubois, J. Jeong , A. Nanda, *Performance Evaluation* 49(1), September 2002 .
- [G02] **Put Everything in Future (Disk) Controllers.** Jim Gray, *talk at the USENIX Conference on File and Storage Technologies (FAST)*, Monterey, California, January 2002.
- [BGM00] **Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing.** L.A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzky, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. *International Symposium on Computer Architecture (ISCA)*. Vancouver, Canada, June 2000.
- [AUS98] **Active disks: Programming model, algorithms and evaluation.** A. Acharya, M. Uysal, and J. Saltz. *International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, California, October 1998.
- [KPH98] **A Case for Intelligent Disks (IDISks).** K. Keeton, D. A. Patterson, J. Hellerstein. *SIGMOD Record*, 27(3):42--52, September 1998.
- [PGK88] **A Case for Redundant Arrays of Inexpensive Disks (RAID).** D. A. Patterson, G. A. Gibson, and R. H. Katz. *ACM International Conference on Management of Data (SIGMOD)*, June 1988.

References

Methodologies and Benchmarks

- [DMM04] **Accurate Cache and TLB Characterization Using hardware Counters.** J. Dongarra, S. Moore, P. Mucci, K. Seymour, H. You. *International Conference on Computational Science (ICCS)*, Krakow, Poland, June 2004.
- [SAF04] **DBmbench: Fast and Accurate Database Workload Representation on Modern Microarchitecture.** M. Shao, A. Ailamaki, and B. Falsafi. *Carnegie Mellon University Technical Report CMU-CS-03-161*, 2004 .
- [KP00] **Towards a Simplified Database Workload for Computer Architecture Evaluations.** K. Keeton and D. Patterson. *IEEE Annual Workshop on Workload Characterization*, Austin, Texas, October 1999.

Useful Links

- ❑ Info on Intel Pentium4 Performance Counters:
<ftp://download.intel.com/design/Pentium4/manuals/25366814.pdf>
- ❑ AMD hardware performance counters
<http://www.amd.com/us-en/Processors/DevelopWithAMD/>
- ❑ PAPI Performance Library
<http://icl.cs.utk.edu/papi/>
- ❑ Intel® VTune™ Performance Analyzers
<http://developer.intel.com/software/products/vtune/>