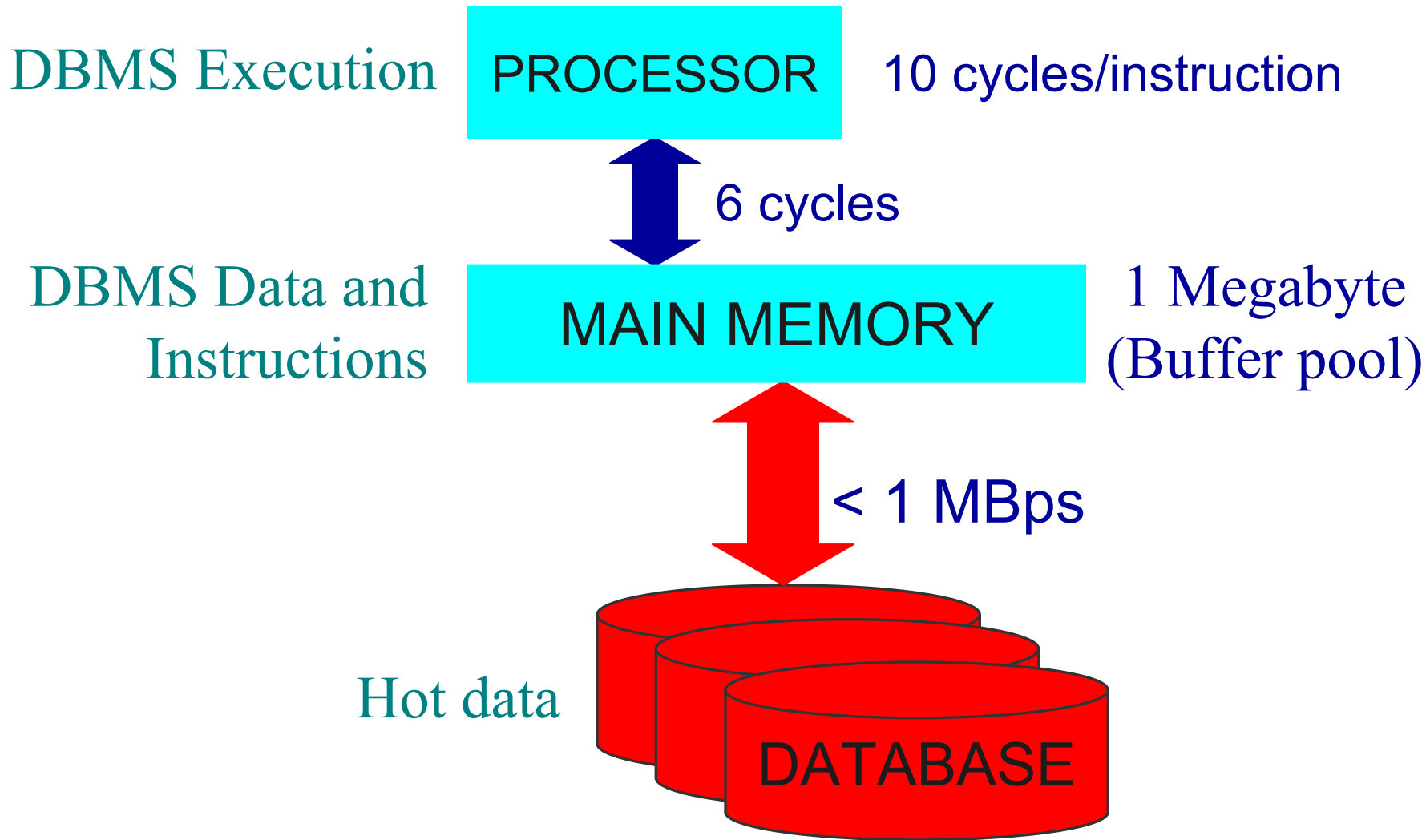


Architecture-Conscious Database Systems

Anastassia Ailamaki

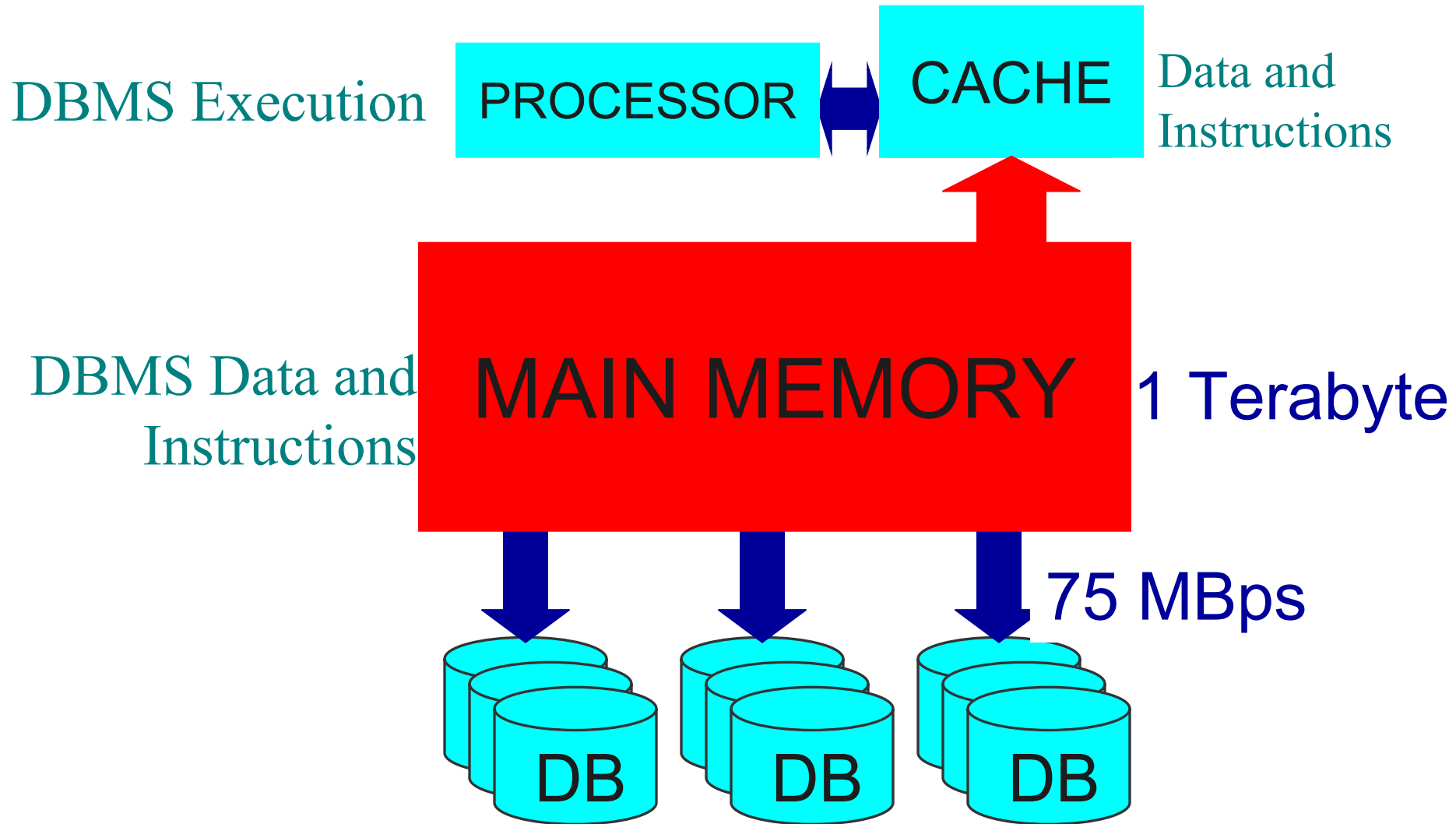
Ph.D. Examination
November 30, 2000

A DBMS on a 1980 Computer



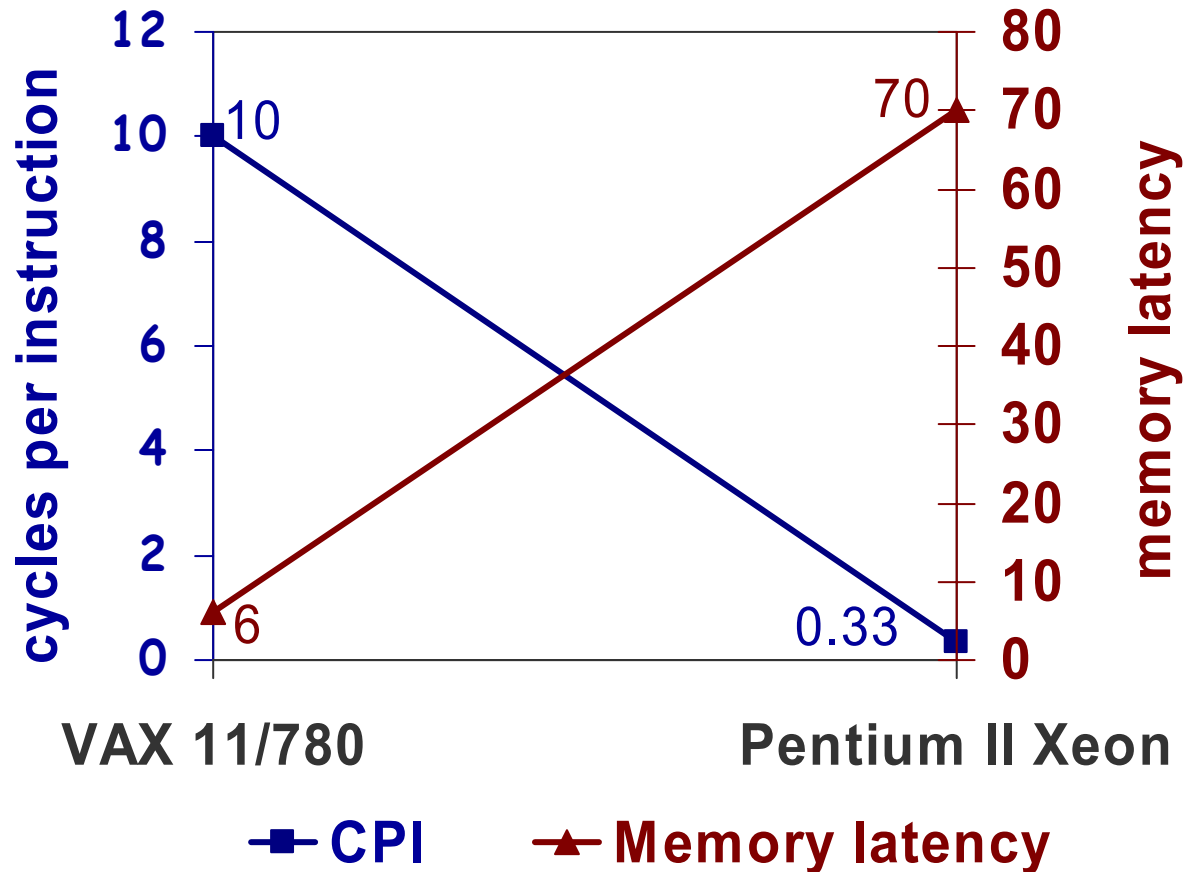
The main performance bottleneck was I/O latency

Present and Future Platforms



Hot data migrates to larger and slower main memory

Processor & Memory Speed Gap



One access to memory is 100's of instruction opportunities

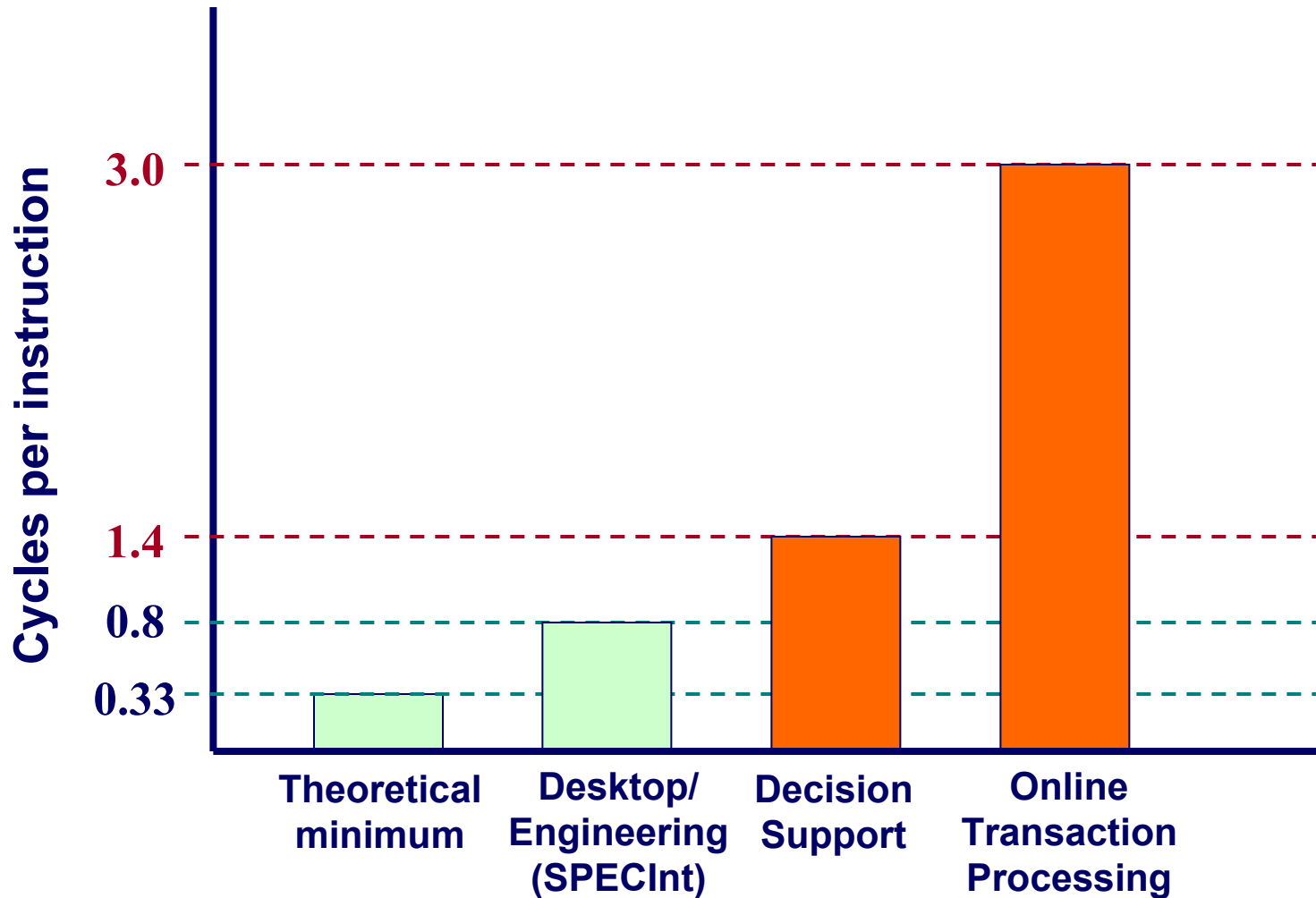
On Today's Computers

- “When you think about what today's machines do - they
- look at the instruction stream dynamically,
 - find parallelism on the fly,
 - execute instructions out of order, and
 - speculate on branch outcomes -
- it's amazing that they work.”

John Hennessy, IEEE Computer, August 1999

New architectures are more sophisticated

Why Study Database Performance?



High average time per instruction for DB workloads

Contributions (I): Analysis

Problem: Where does query execution time go?

Proposed evaluation framework [VLDB'99]

Identified bottlenecks in hardware

- memory access
- hardware implementation details

Discovered two memory-related bottlenecks

- second-level cache data access
- first-level instruction cache access

Methodological discovery: micro-benchmarks

A systematic evaluation framework

Contributions (II): Software

Problem: Current data placement hurts caches

Proposed novel data placement [subm. SIGMOD'01]

- rearranges data records on disk page
- optimizes data cache performance

Evaluated it against the popular scheme

- 70% less data-related memory access delays
- does not affect I/O behavior
- especially beneficial for decision support workloads

A cache-conscious data placement

Contributions (III): Hardware

Problem: Hardware design affects DB behavior

Compared Shore on four different systems

- ❑ different processor architectures/ μ -architectures
- ❑ different memory subsystems

Found evidence that DBMSs would benefit from

- ❑ 2-4 way associative, larger L2, no inclusion
- ❑ large blocks, no sub-blocking
- ❑ high-accuracy branch prediction
- ❑ memory-aggressive execution engine

Step towards a DSS-centric machine

Outline

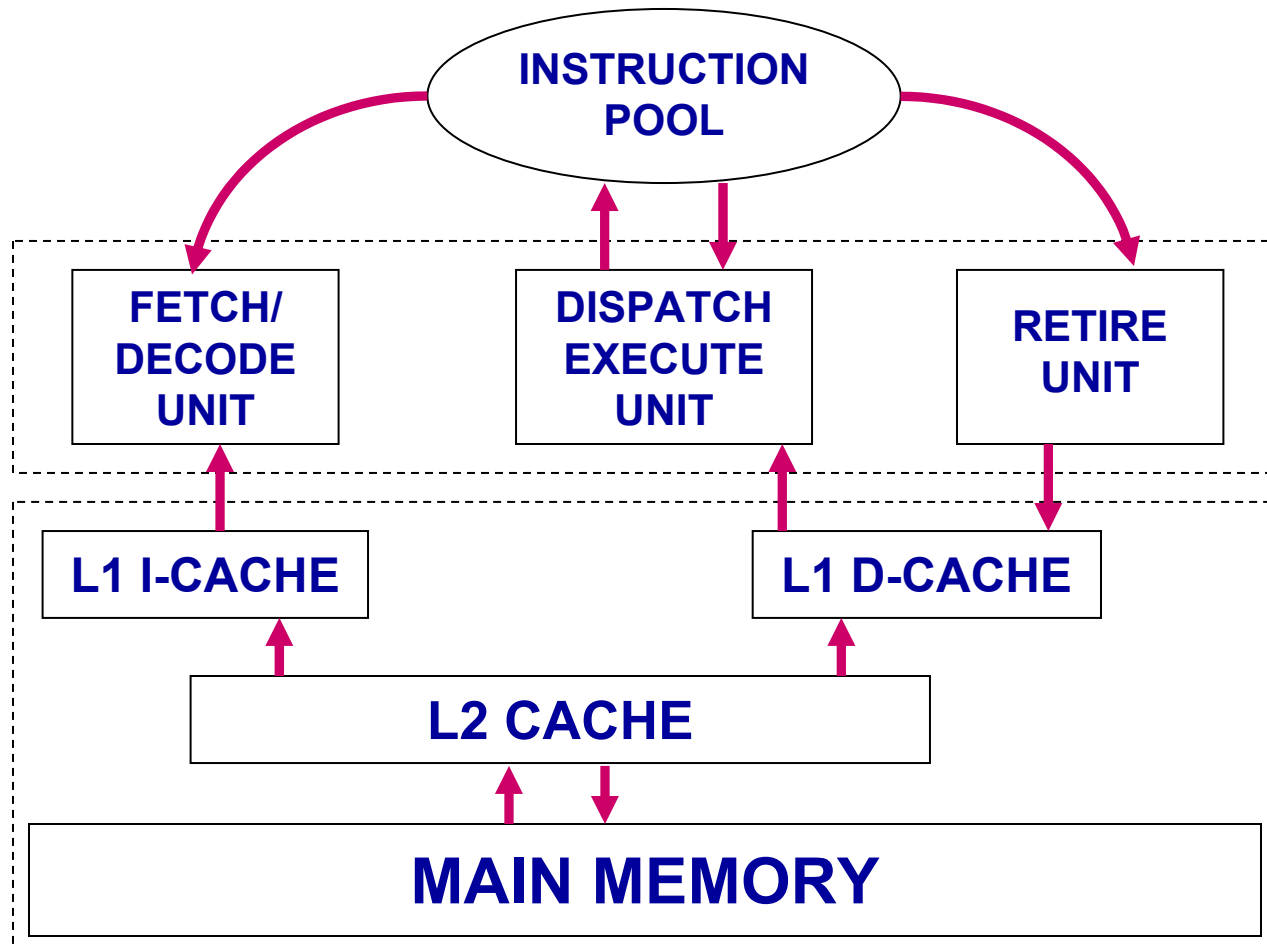
- Introduction
- **PART I: Analysis**
 - Background
 - Query execution time breakdown
 - Experimental results
 - Bottleneck assessment
- PART II: Partition Attributes Across (PAX)
- PART III: Towards a DSS-centric h/w design
- Conclusions

Previous Work

- Workload characterization studies, e.g., [Barroso 98], [Keeton 98]
 - Various platforms, mostly multiprocessor
 - One DBMS per platform
- Results:
 - Commercial different than scientific apps
 - OLTP different than DSS workloads
 - Memory is major bottleneck

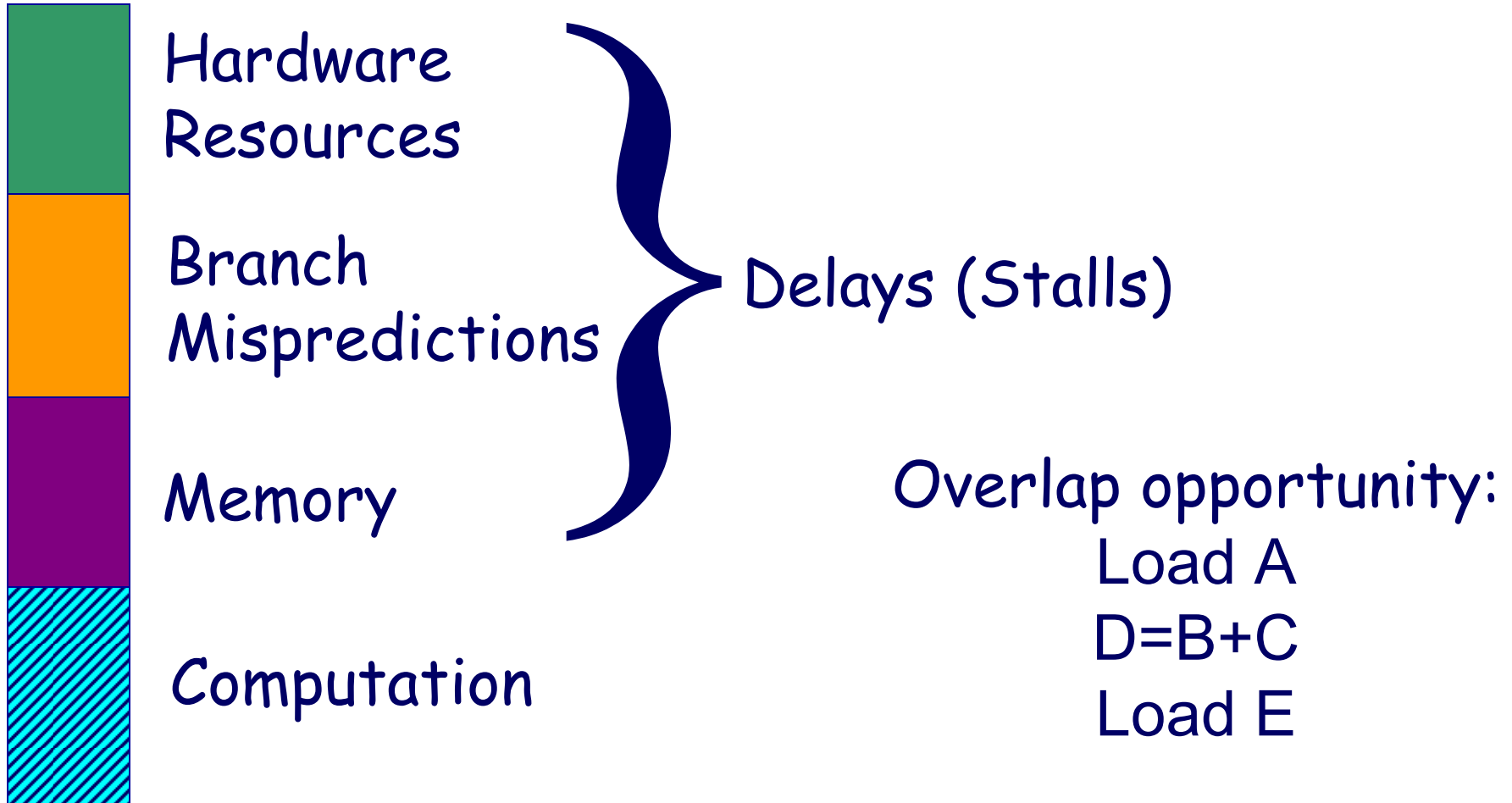
No coherent study across DBMSs and workloads

An Execution Pipeline



Branch prediction, non-blocking cache, out-of-order

Where Does Time Go?



$$\text{Execution Time} = \text{Computation} + \text{Stalls} - \text{Overlap}$$

Setup and Methodology

Range Selection

(sequential, indexed)

```
select avg (a3)
from R
where a2 > Lo and a2 < Hi
```

Equijoin

(sequential)

```
select avg (a3)
from R, S
where R.a2 = S.a1
```

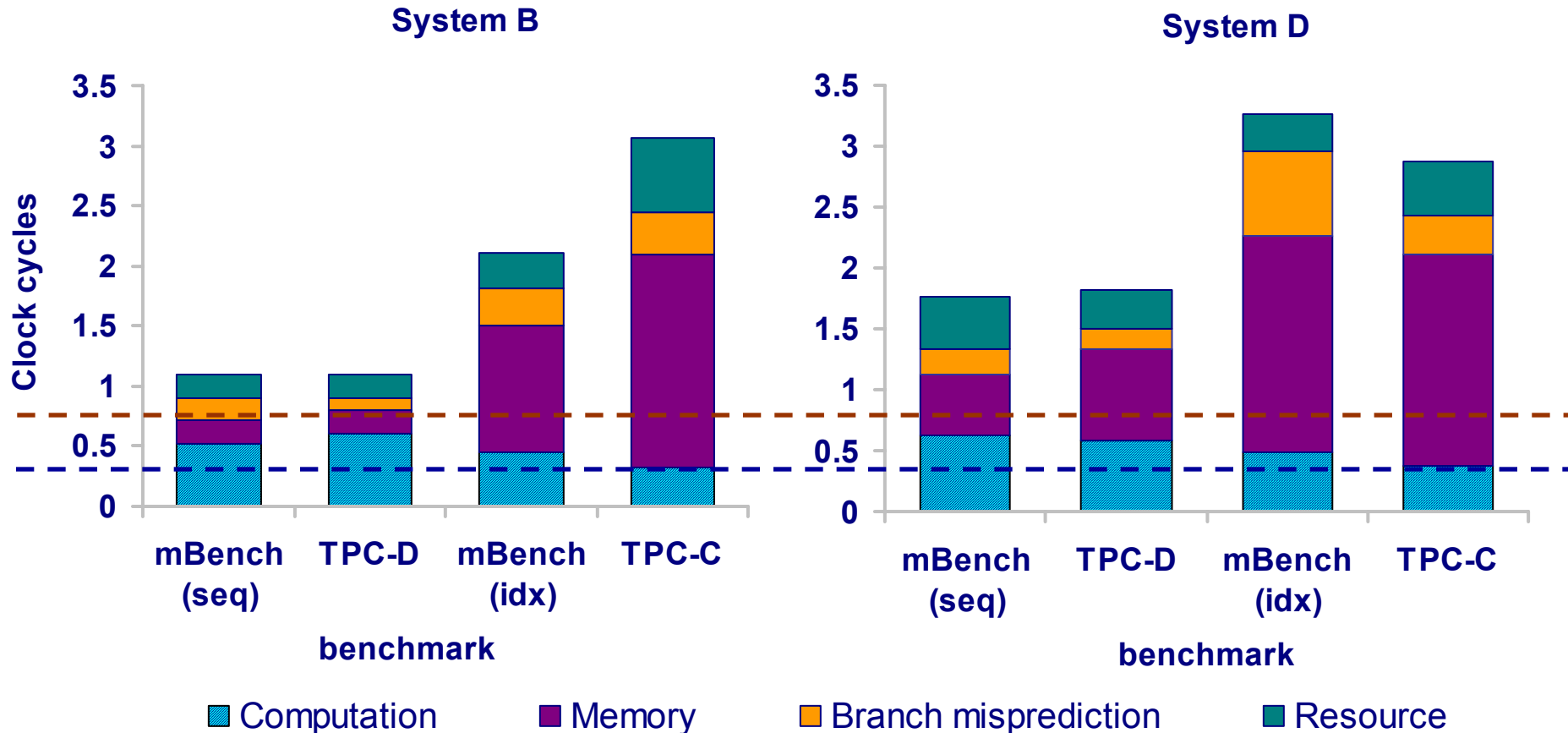
- Four commercial DBMSs: A, B, C, D
- 6400 PII Xeon/MT running Windows NT 4
- Used processor counters to measure/estimate

Crafted microbenchmarks to isolate execution loops

Time Calculations

- Measured: Resource stalls, L1I stalls
- Estimated:
 - L1 data stalls: # misses * penalty
 - L2 stalls: # misses * measured memory latency
 - Branch misprediction stalls: # mispr. * penalty
- Overlap: measured CPI / expected CPI

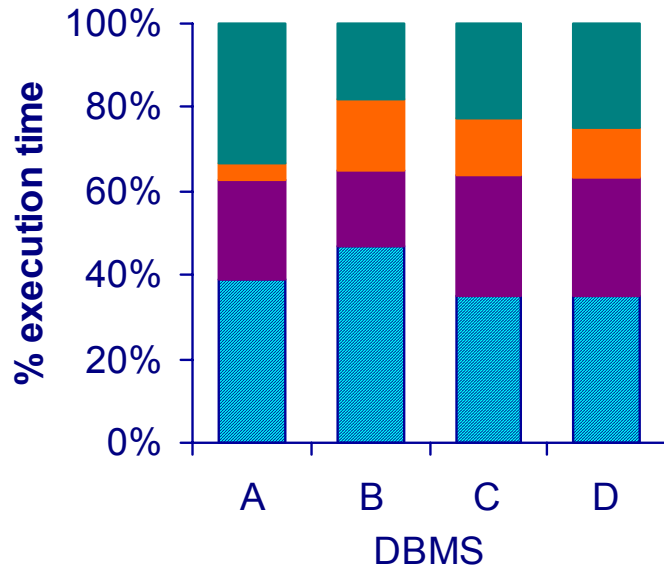
Microbenchmarks vs. TPC



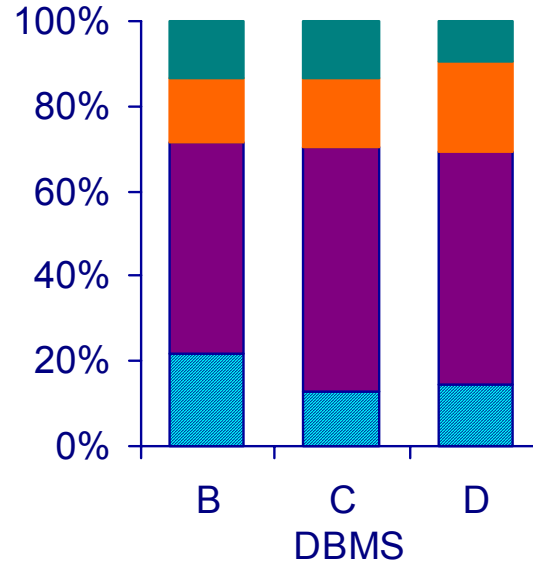
- High CPI compared to integer workloads
- Sequential scan / TPC-D, 2ary index / TPC-C

Execution Time Breakdown (%)

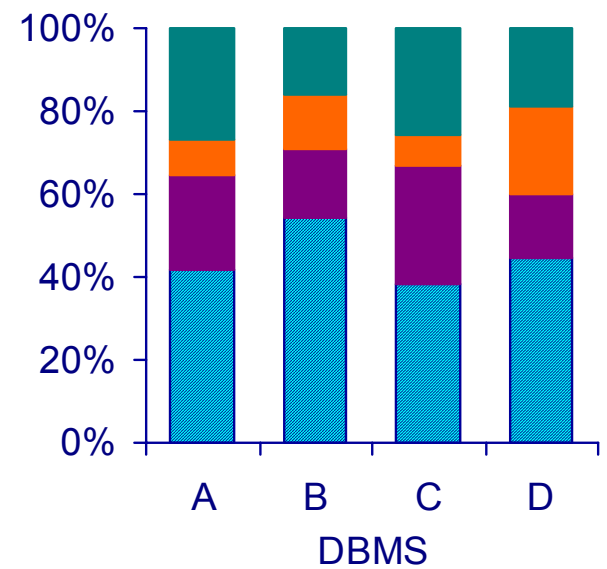
10% Sequential Scan



10% 2ary index selection



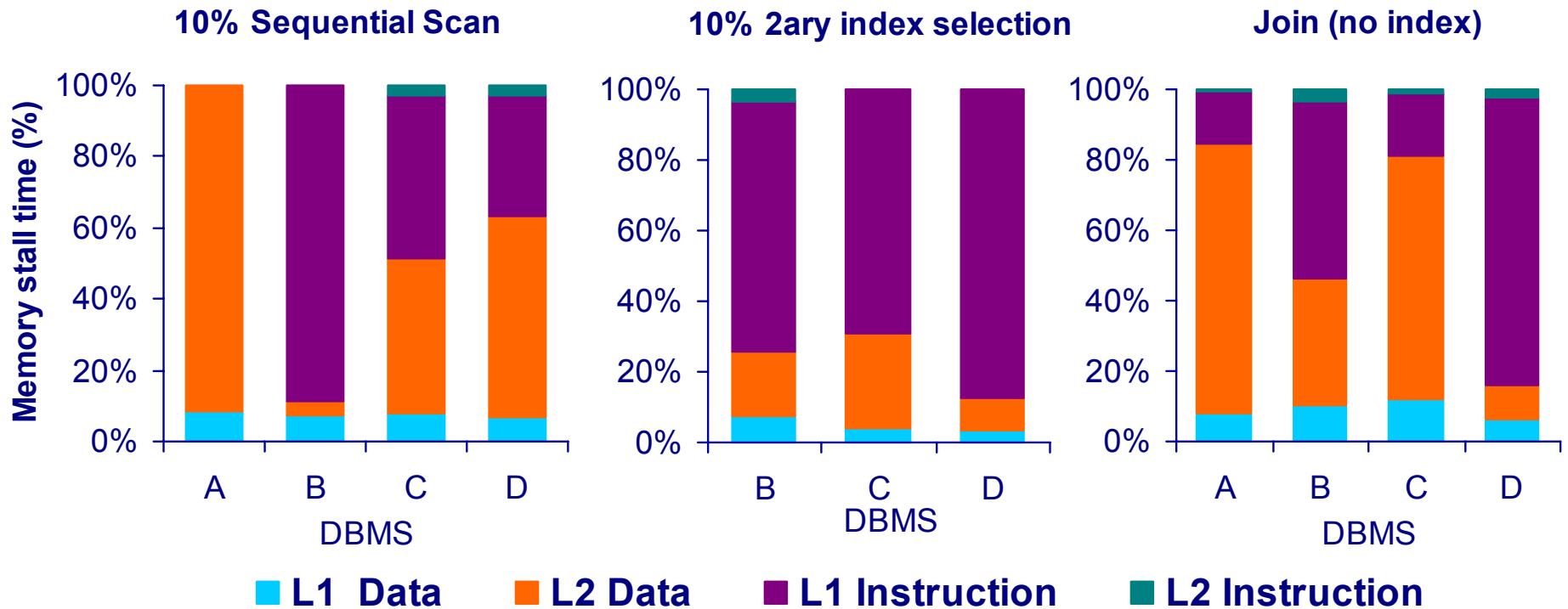
Join (no index)



■ Computation ■ Memory ■ Branch mispredictions ■ Resource

- ❑ Stalls at least 50% of time
- ❑ Memory stalls are major bottleneck

Memory Stalls Breakdown (%)



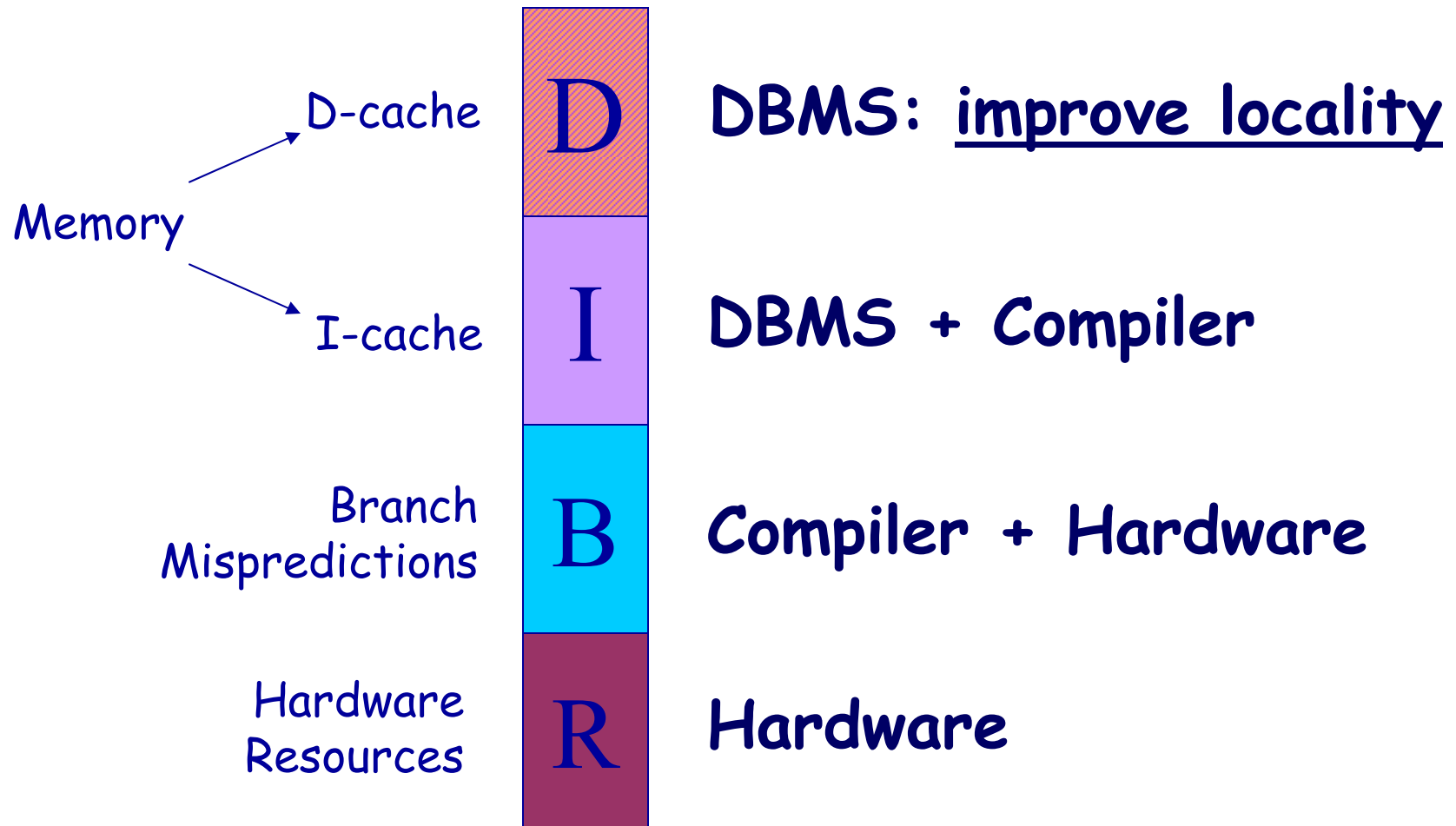
- L1 instruction and L2 data stalls dominate
- Different memory bottlenecks across DBMSs and queries

Summary of Analysis

- ❑ We can use microbenchmarks instead of TPC
- ❑ Execution time breakdown shows trends
- ❑ Memory access is a major bottleneck
 - ❑ Increasing memory-processor performance gap
 - ❑ Deeper memory hierarchies expected
 - ❑ L2 cache data misses
 - ❑ L2 grows (8MB), but will be slower
 - ❑ Stalls due to L1 I-cache misses
 - ❑ L1 I-cache not likely to grow as much as L2

We need to address every reason for stalls

Addressing Bottlenecks



Data cache: A clear responsibility of the DBMS

Outline

- Introduction
- PART I: Where Does Time Go?
- **PART II: Partition Attributes Across**
 - The current scheme: Slotted pages
 - Partition Attributes Across (PAX)
 - Performance Results
- PART III: Towards a DSS-centric h/w design
- Conclusions

The Data Placement Tradeoff

- ❑ *Slotted Pages: Used by all commercial DBMSs*
 - ❑ Store table records sequentially
 - ❑ Intra-record locality (attributes of record r together)
 - ❑ ...but pollutes cache
- ❑ *Inspiration: Vertical partitioning [Copeland'85]*
 - ❑ Store n -attribute table as n single-attribute tables
 - ❑ Problem: High record reconstruction cost
- ❑ *Partition Attributes Across (PAX)*
 - ❑ Have the cake and eat it, too!

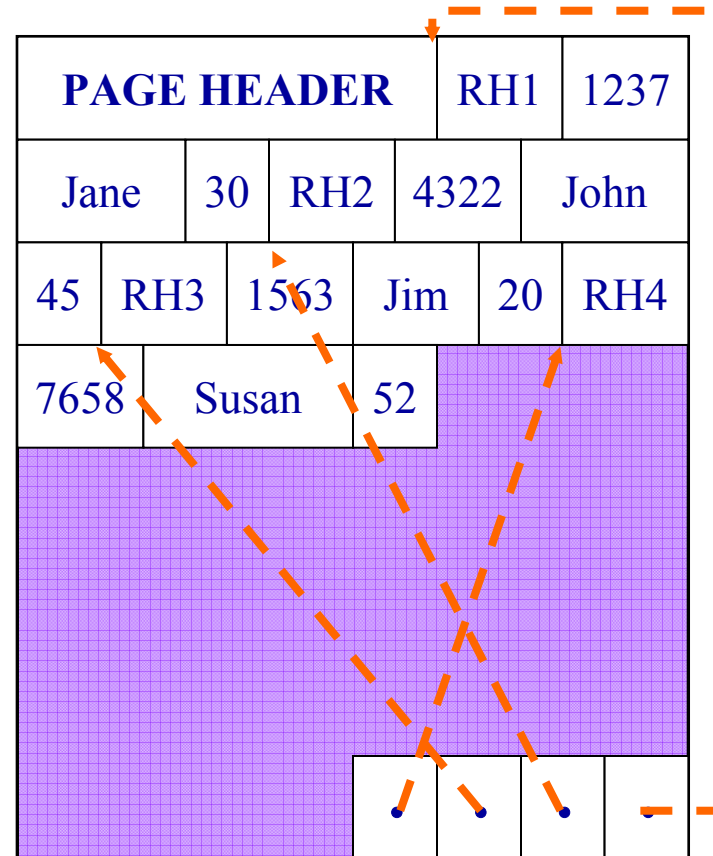
PAX: Inter-record locality, low reconstruction cost

Current Scheme: Slotted Pages

- Formal name: NSM (N-ary Storage Model)

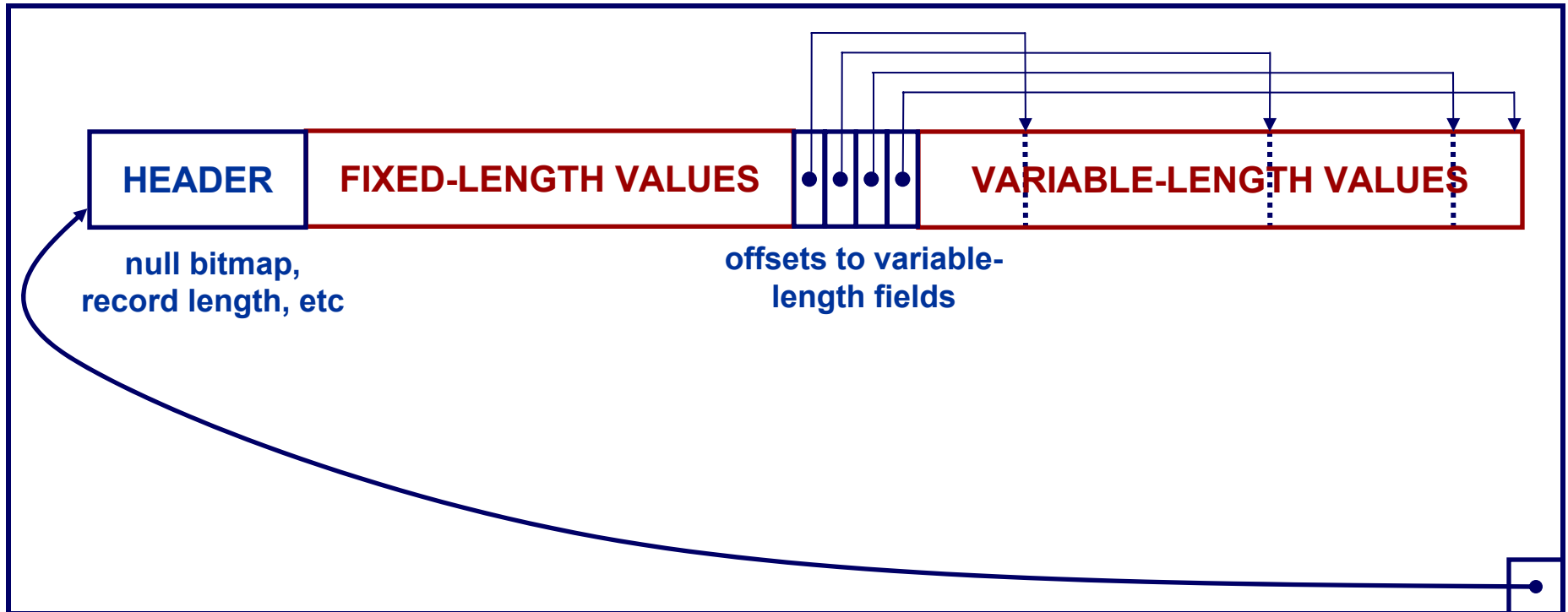
R

| RID | SSN | Name | Age |
|-----|------|-------|-----|
| 1 | 1237 | Jane | 30 |
| 2 | 4322 | John | 45 |
| 3 | 1563 | Jim | 20 |
| 4 | 7658 | Susan | 52 |
| 5 | 2534 | Leon | 43 |
| 6 | 8791 | Dan | 37 |



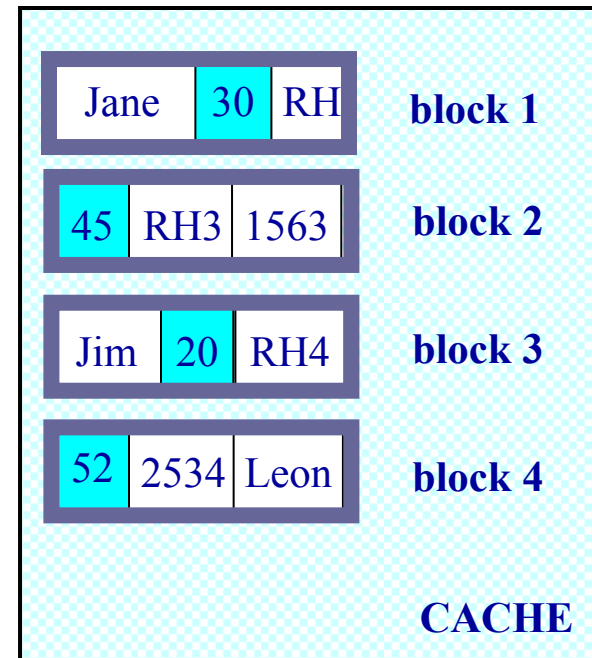
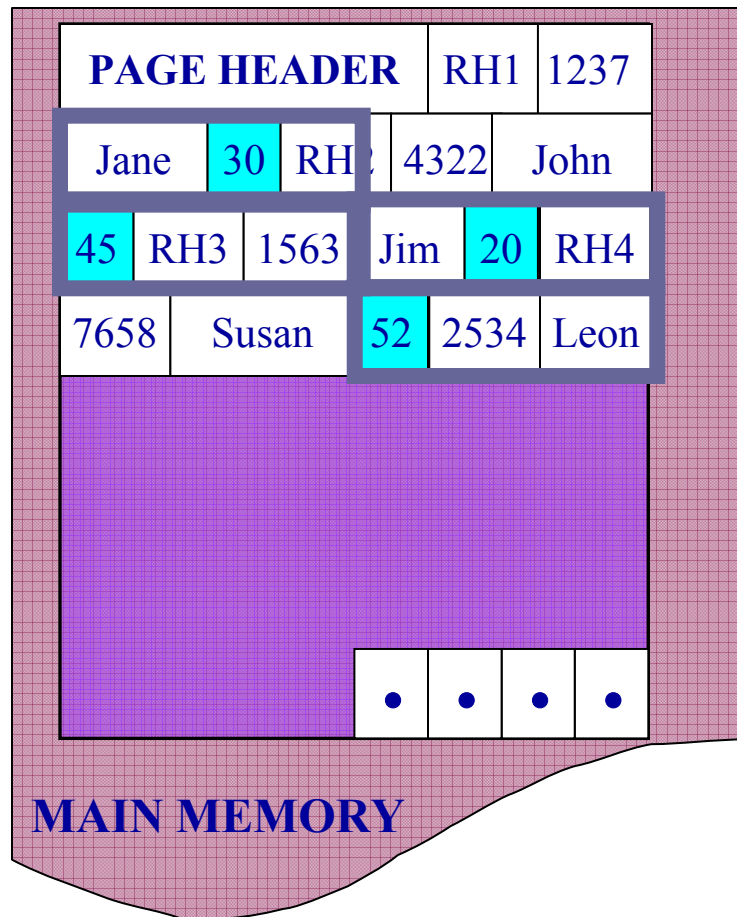
- Records are stored sequentially
- Offsets to start of each record at end of page

Current Scheme: Slotted Pages



All attributes of a record are stored together

NSM Cache Behavior

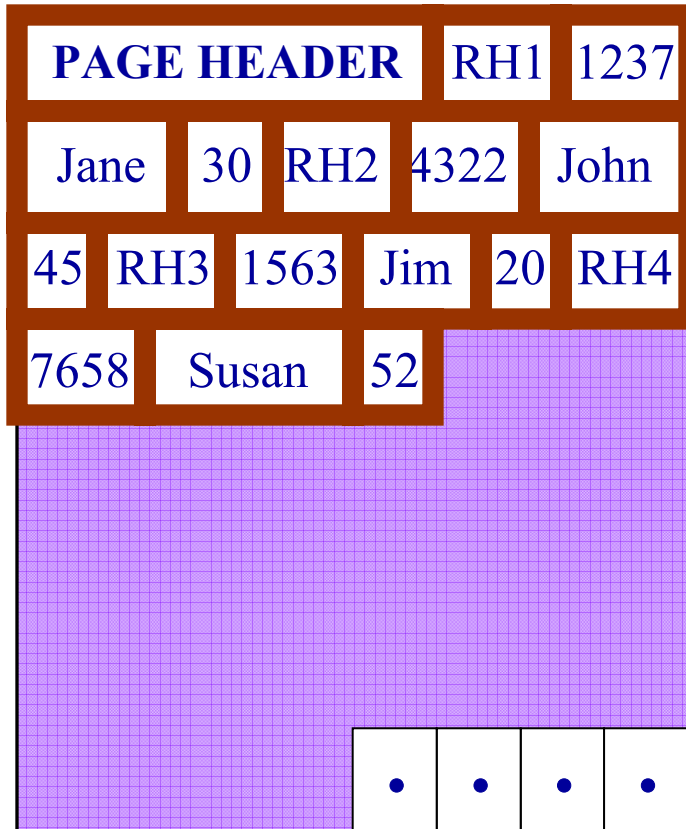


*select name
from R
where age > 40*

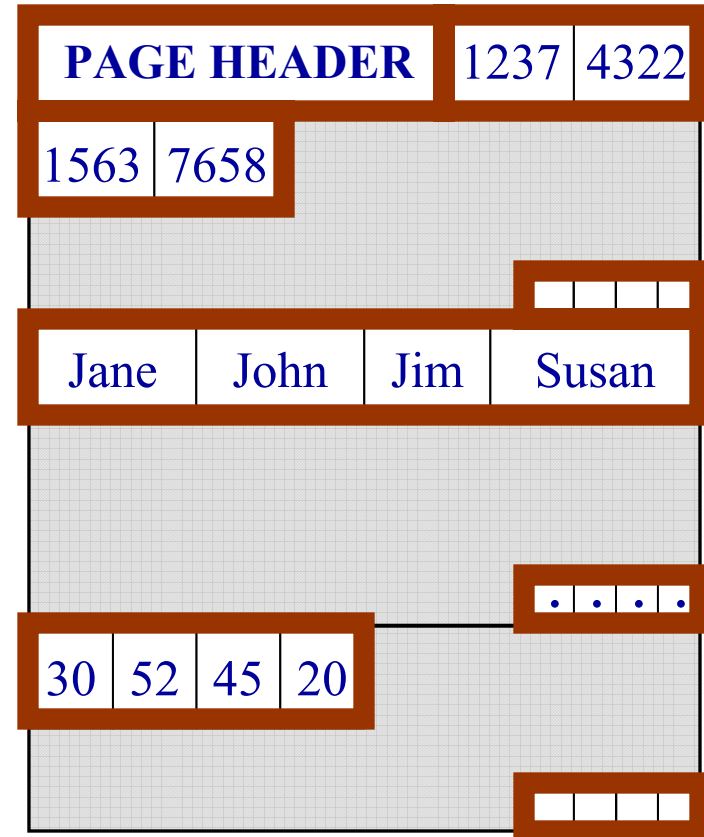
NSM pollutes the cache and wastes bandwidth

Partition Attributes Across (PAX)

NSM PAGE

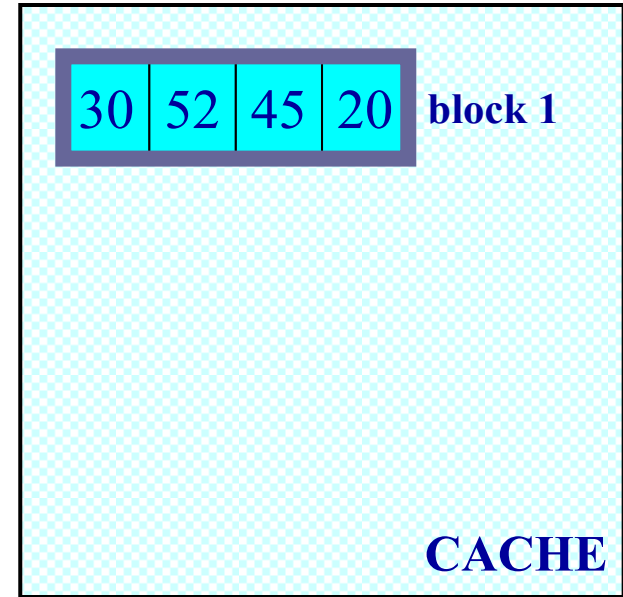
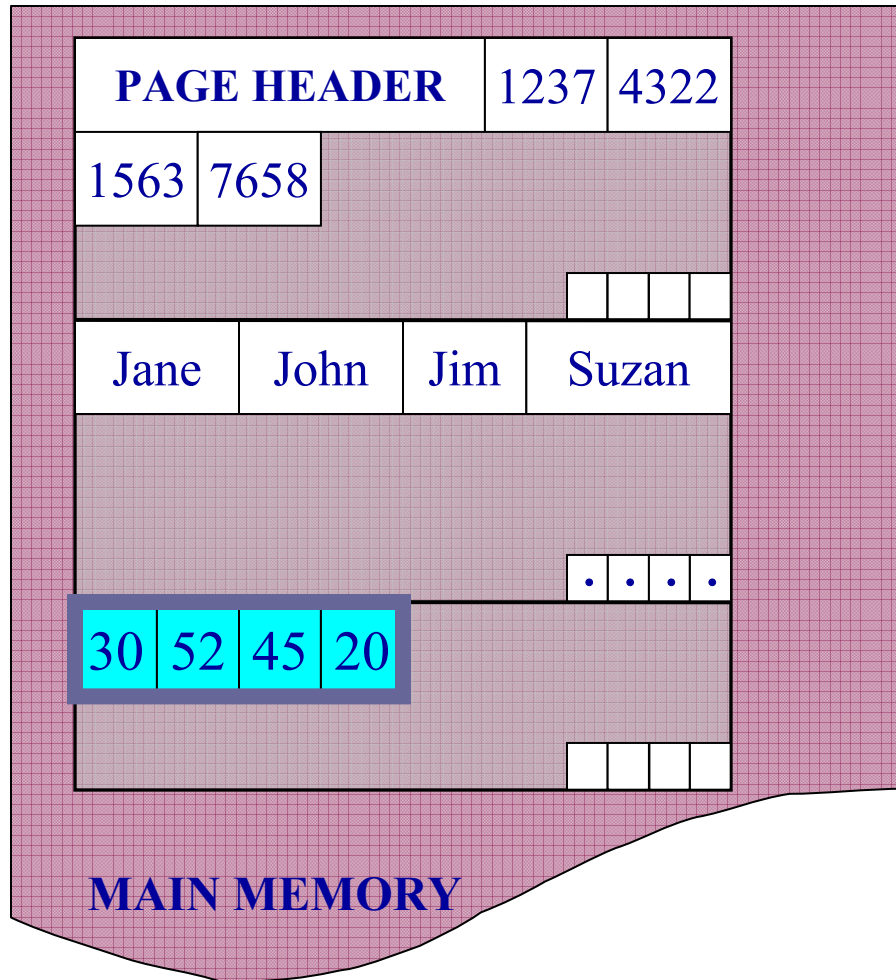


PAX PAGE



Partition data *within* the page for spatial locality

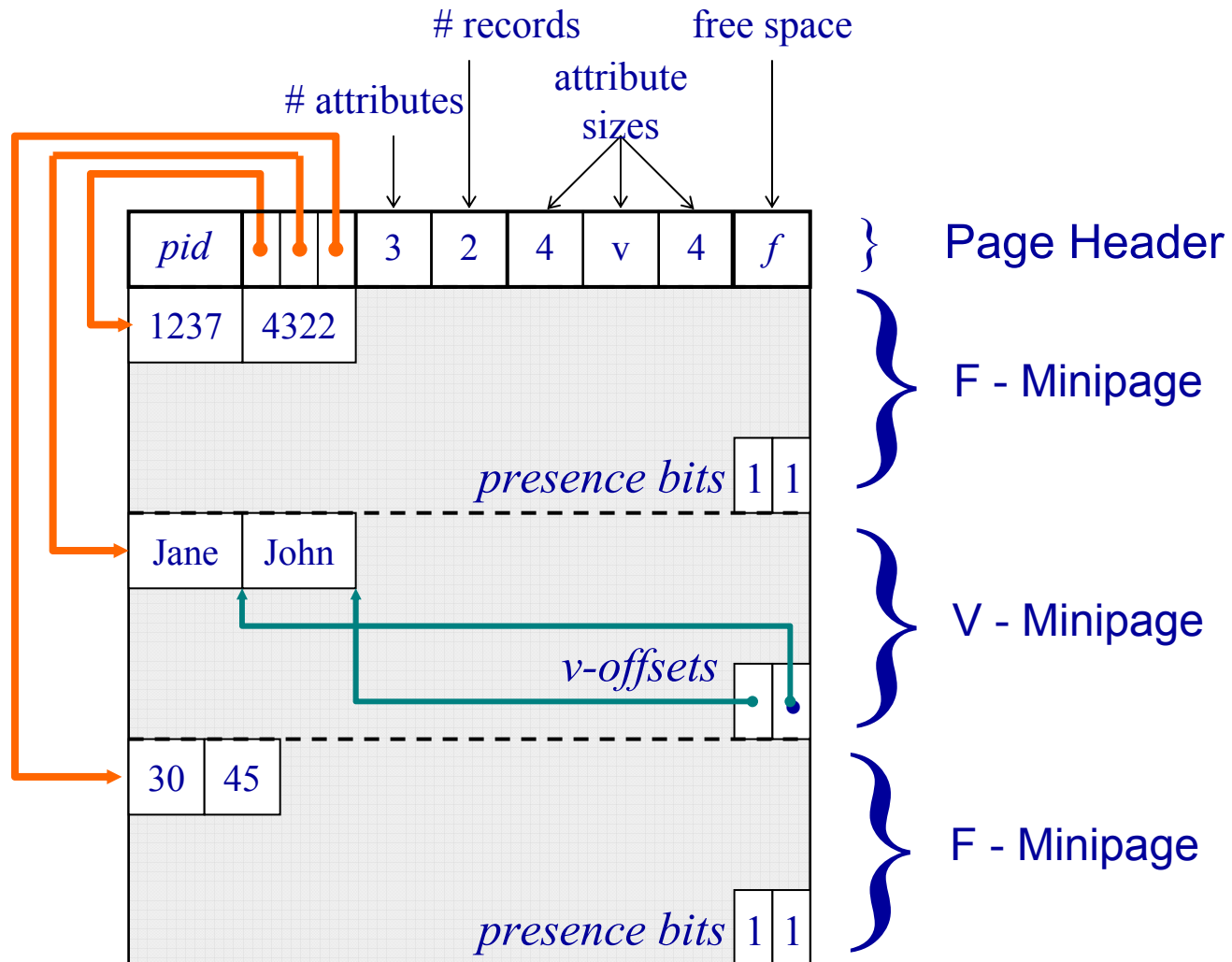
PAX: Mapping to Cache



*select name
from R
where age > 40*

Fewer cache misses, low reconstruction cost

PAX: Detailed Design



Basic Evaluation: Methodology

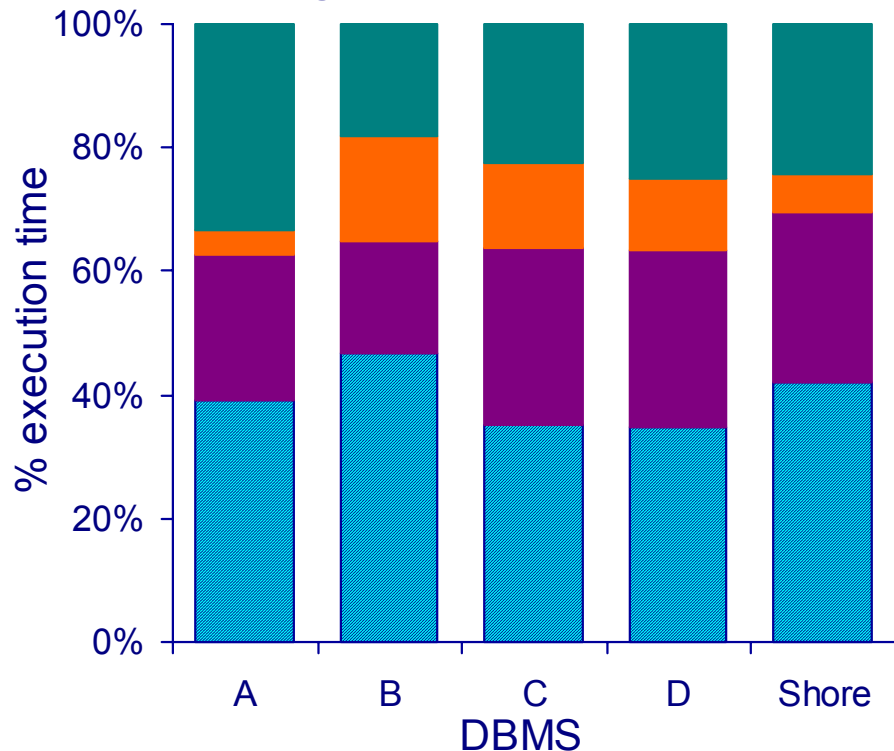
- Main-memory resident R
- Query:

```
select avg (ai)  
from R  
where aj >= Lo and aj <= Hi
```
- PII Xeon running Windows NT 4
- 16KB L1-I, 16KB L1-D, 512 KB L2, 512 MB RAM
- Used processor counters
- Implemented schemes on Shore Storage Manager
 - Similar behavior to commercial Database Systems

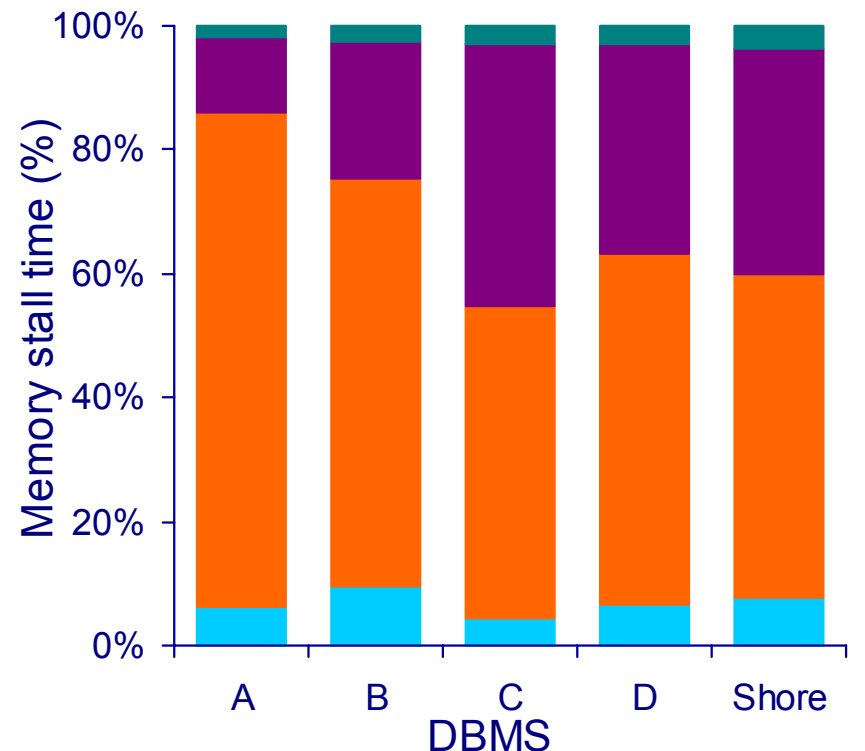
Why Use Shore?

- ❑ Range selection query on 4 commercial DBMSs + Shore
- ❑ Breakdown of execution & memory delays

Range Selection (no index)

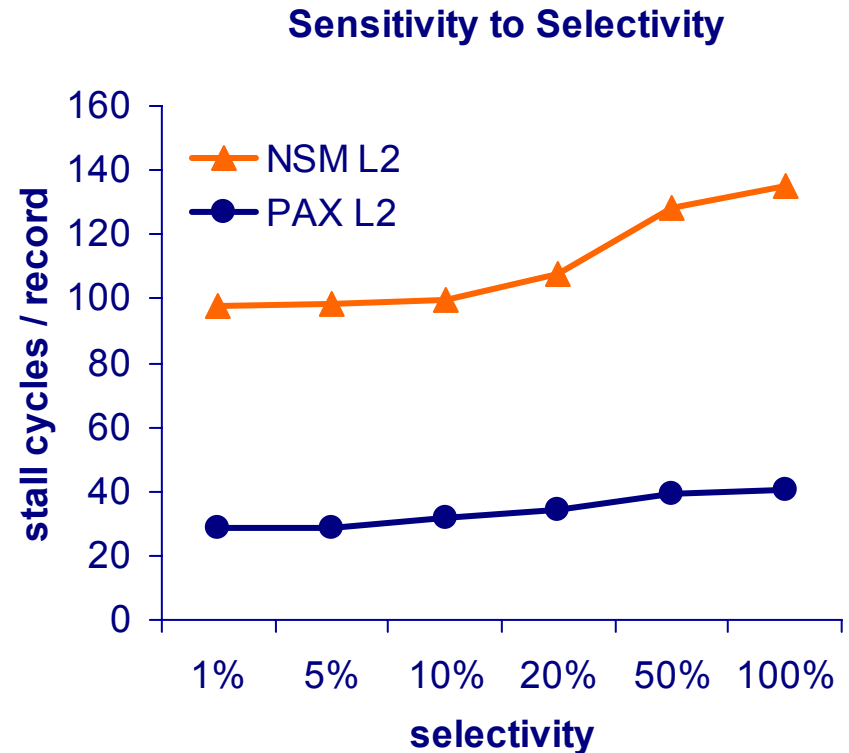
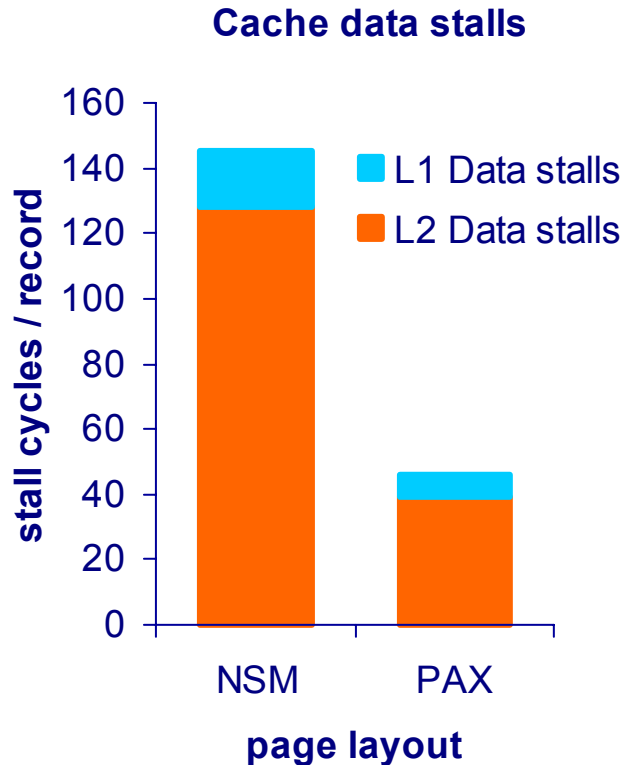


Range Selection (no index)



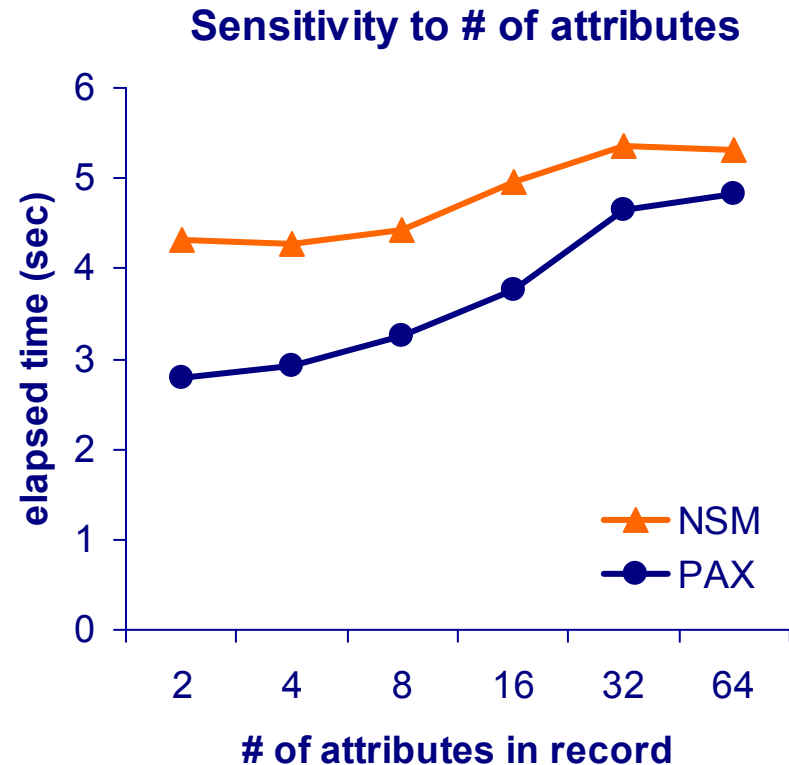
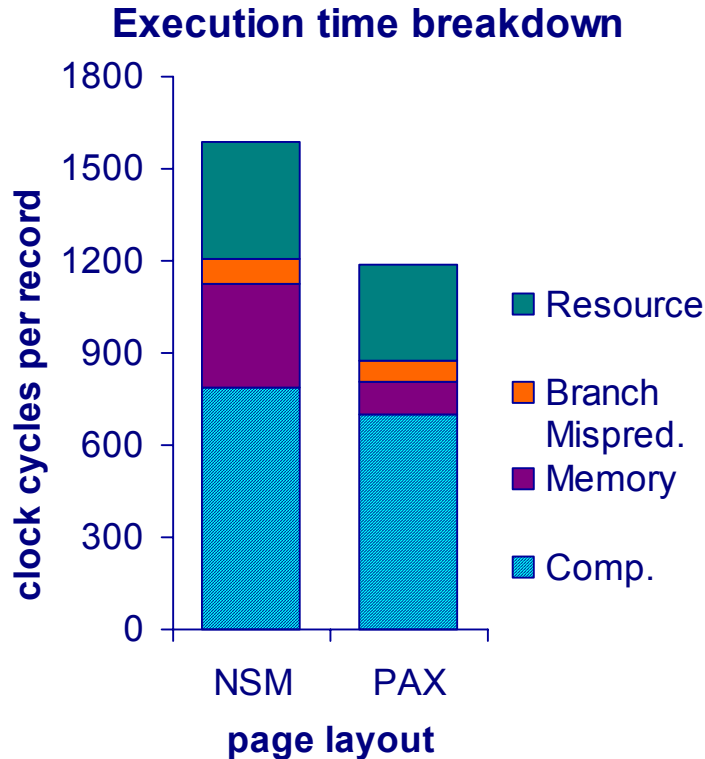
We can use Shore to evaluate DSS workload behavior

Effect on Accessing Cache Data



- ❑ PAX incurs **70%** less data cache penalty than NSM
- ❑ PAX reduces cache misses at both L1 and L2
- ❑ Selectivity doesn't matter for PAX data stalls

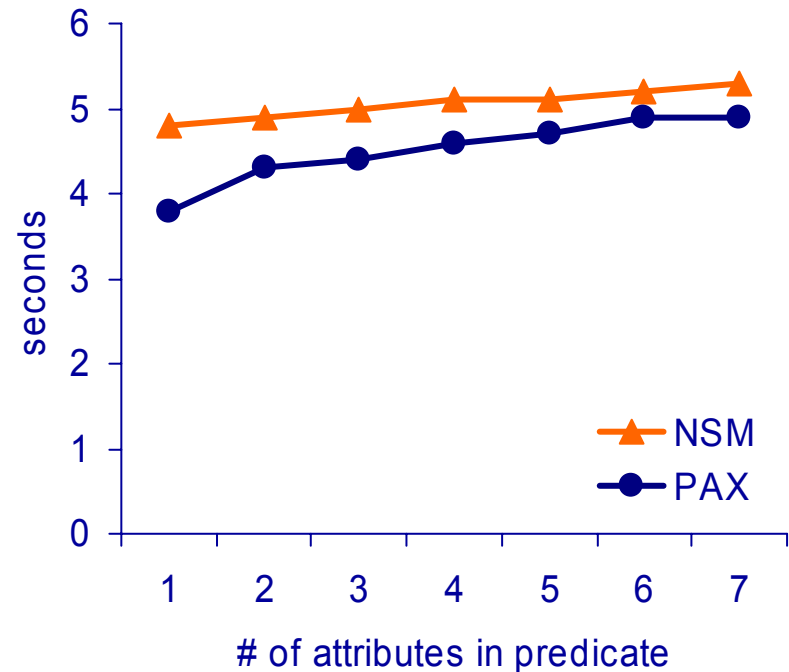
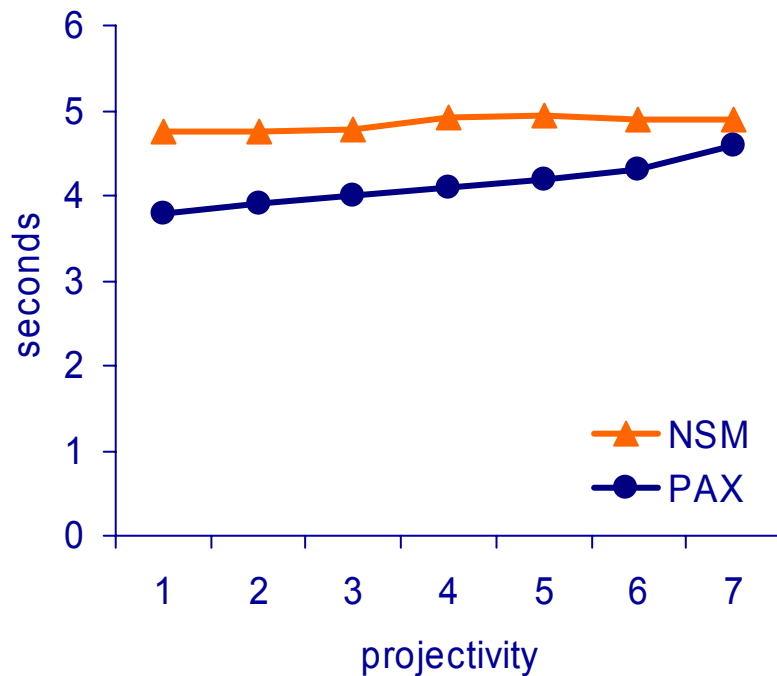
Time and Sensitivity Analysis



- PAX: 75% less memory penalty than NSM (10% of time)
- Execution times converge as number of attrs increases

Sensitivity Analysis (2)

- Elapsed time sensitivity to projectivity / # predicates
- Range selection queries, 1% selectivity



PAX and NSM times converge as query covers entire tuple

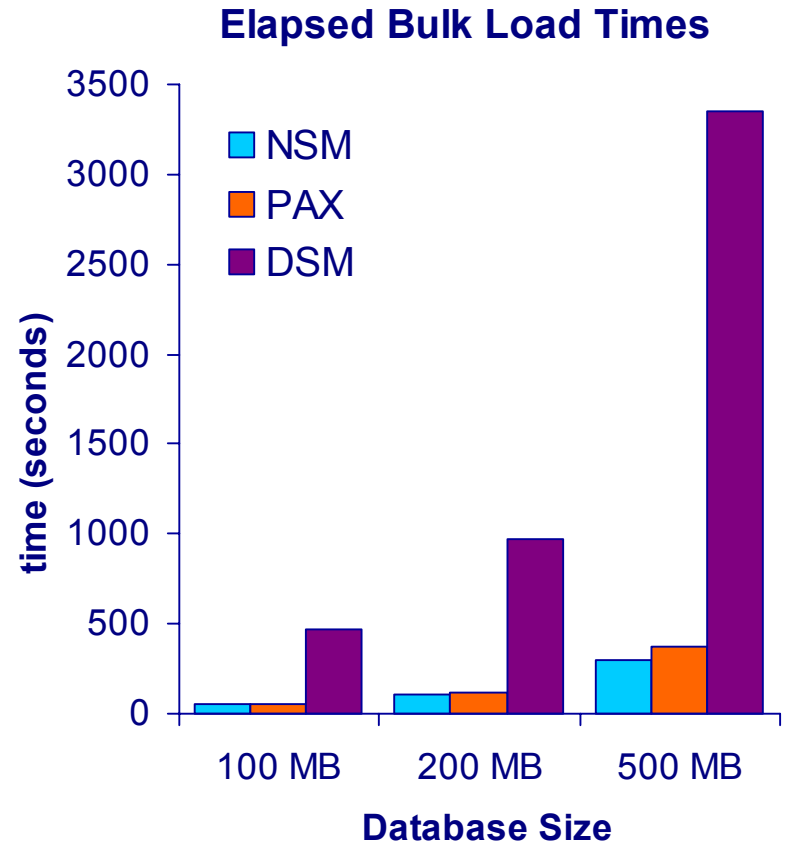
Evaluation Using a DSS Benchmark

- ❑ Loaded 100M, 200M, and 500M TPC-H DBs
- ❑ Ran Queries:
 - ❑ Range Selections w/ variable parameters (RS)
 - ❑ TPC-H Q1 and Q6
 - ❑ sequential scans
 - ❑ lots of aggregates (*sum, avg, count*)
 - ❑ grouping/ordering of results
 - ❑ TPC-H Q12 and Q14
 - ❑ (Adaptive Hybrid) Hash Join
 - ❑ complex 'where' clause, conditional aggregates
- ❑ PII Xeon running Windows NT 4
- ❑ Used processor counters

Insertions with PAX

- Estimate average field sizes
- Start inserting records
- If a record doesn't fit,
 - Reorganize page
 - (move minipage boundaries)
- Adjust average field sizes

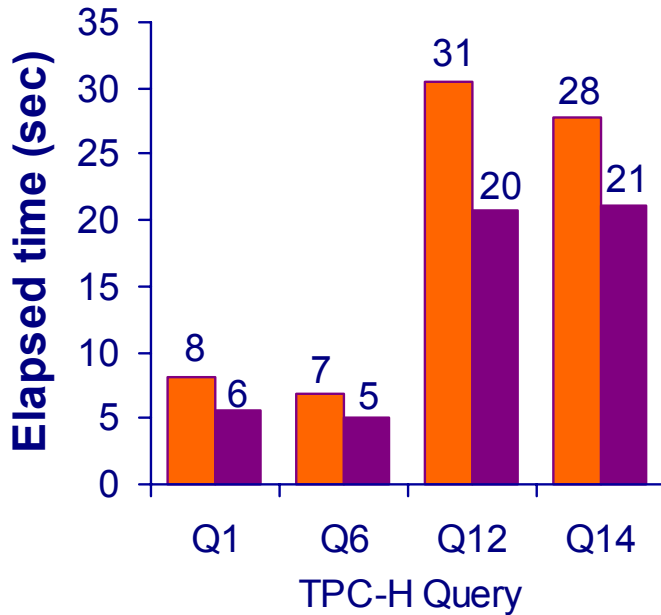
- 50% of reorganizations accommodate a single record



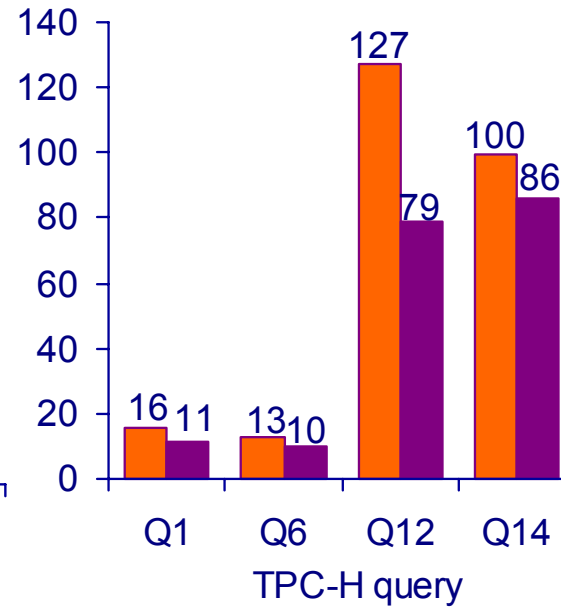
PAX loads a TPC-H database in 2-26% longer than NSM

Elapsed Execution Time

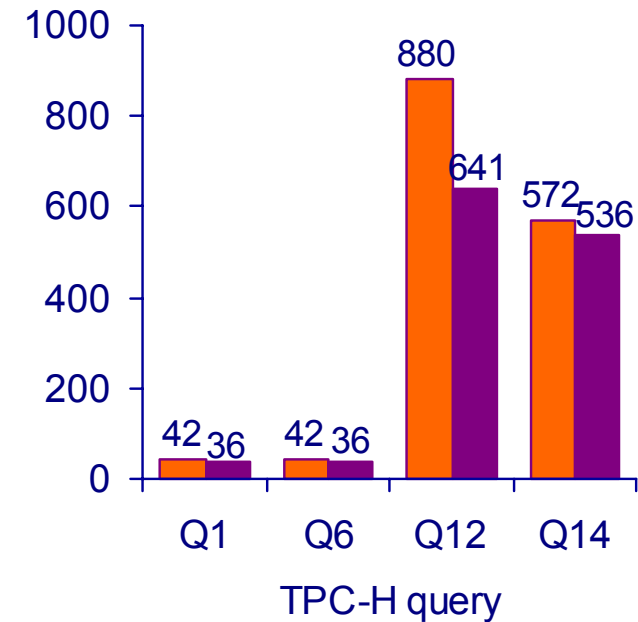
TPC-H 100M



TPC-H 200M



TPC-H 500M



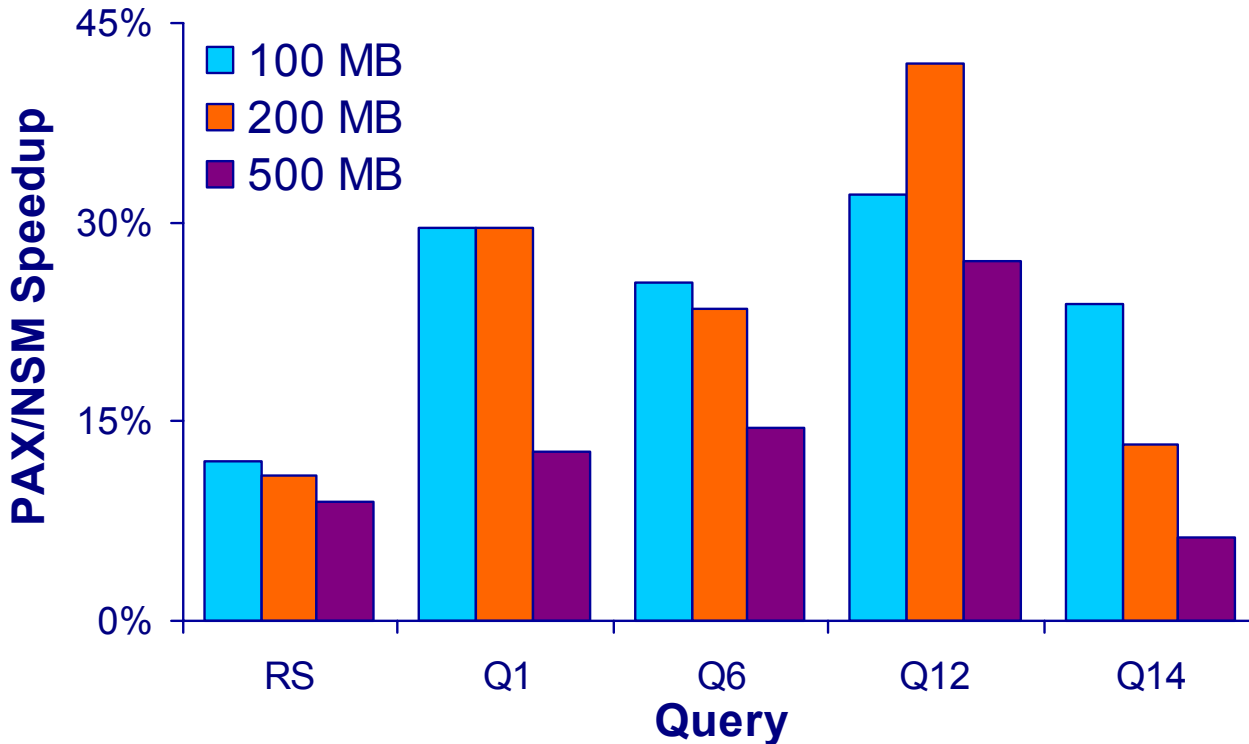
■ NSM

■ PAX

PAX improves performance up to 42% even with I/O

Speedup

PAX/NSM Speedup on PII/NT



- PAX improves performance up to 42% even with I/O
- Speedup differs across DB sizes

PAX: Summary

Advantages

- ❑ High data cache performance
- ❑ Faster than NSM for DSS queries
- ❑ Orthogonal to other storage decisions
- ❑ Does not affect I/O performance

Current Disadvantages

- ❑ Complex free space mgmt with variable length attributes \Rightarrow Complicates update algorithm

PAX beneficial for read-mostly workloads (e.g., DSS)
(update-intensive workloads in future work)

Outline

- Introduction
- PART I: Where Does Time Go?
- PART II: Partition Attributes Across
- **PART III: Towards DSS-Centric H/W**
 - Memory subsystem
 - Branch prediction mechanism
 - Processor pipeline
- Conclusions

Platform Differences

- Architecture
 - RISC or CISC Instruction set
- Microarchitecture
 - Pipeline
 - Speculation (out-of-order, multiple issue)
 - Branch prediction
 - Memory subsystem
 - Cache size, associativity
 - Block size, subblocking
 - Inclusion

Which design looks beneficial for DSS workloads?

Experimental Setup

- Used four machines
 - Sun UltraSparc: US-II and US-III, Solaris 2.6/2.7
 - Intel P6: PII Xeon, Linux v2.2
 - DEC Alpha: 21164A, OSF1 v.4.0
- Architecture and Processor Microarchitecture

| Characteristic | UltraSparc | | PII Xeon | Alpha 21164 |
|------------------------|------------|---------|----------|-------------|
| | US-II | US-III | | |
| <i>speed</i> | 296 MHz | 300 MHz | 400 MHz | 532 MHz |
| <i>introduced in</i> | 1997 | 1997 | 1998 | 1996 |
| <i>out of order?</i> | no | no | yes | no |
| <i>instruction set</i> | RISC | RISC | CISC | RISC |

Cache Hierarchies

| Characteristic | | UltraSparc | | PII Xeon | Alpha 21164 |
|----------------|------------------------|-------------|-------------|--------------|-------------|
| | | US-II | US-III | | |
| L1 D | <i>size, assoc</i> | 16KB, DM | 16KB, DM | 16KB, 2-way | 8KB, DM |
| | <i>block/subblock</i> | 32/16 | 32/16 | 32/32 | 32/32 |
| | <i>inclusion by L2</i> | yes | yes | no | yes |
| L1 I | <i>size, assoc</i> | 16KB, 2-way | 16KB, 2-way | 16KB, 4-way | 8KB, DM |
| | <i>block/subblock</i> | 32/32 | 32/32 | 32/32 | 32/16 |
| | <i>inclusion by L2</i> | yes | yes | no | no |
| L2 | <i>size, assoc</i> | 2 MB, DM | 512KB, DM | 512KB, 4-way | 96KB, 3-way |
| | <i>block/subblock</i> | 64/64 | 64/64 | 32/32 | 64/32 |
| | <i>inclusion by L3</i> | N/A | N/A | N/A | yes |
| L3 | <i>size, assoc</i> | N/A | N/A | N/A | 4 MB / DM |
| | <i>block/subblock</i> | N/A | N/A | N/A | 64/64 |

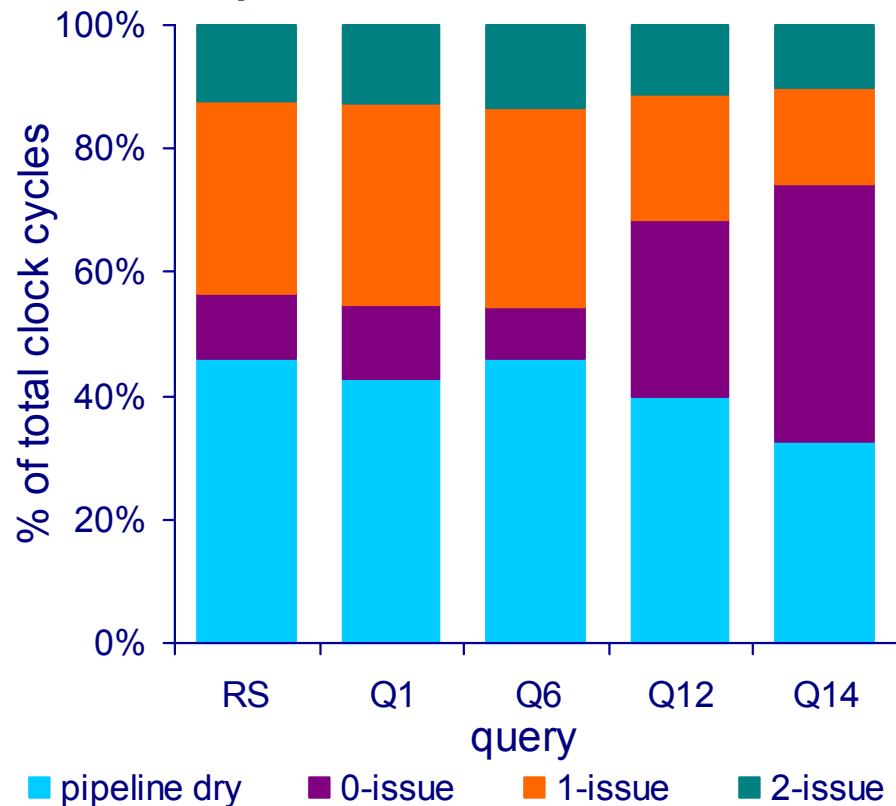
Methodology

- Compiled Shore with gcc 2.95.2
 - Alpha version not optimized
- Ran DSS workload
 - Range Selections w/ variable parameters (RS)
 - TPC-H 1, 6, 12, 14
- Used processors' counters
 - Sun: **run-pic** (by Glenn Ammons, modified)
 - PII: **PAPI** (public-domain counter library)
 - Alpha: **DCPI** (sampling software by Compaq)

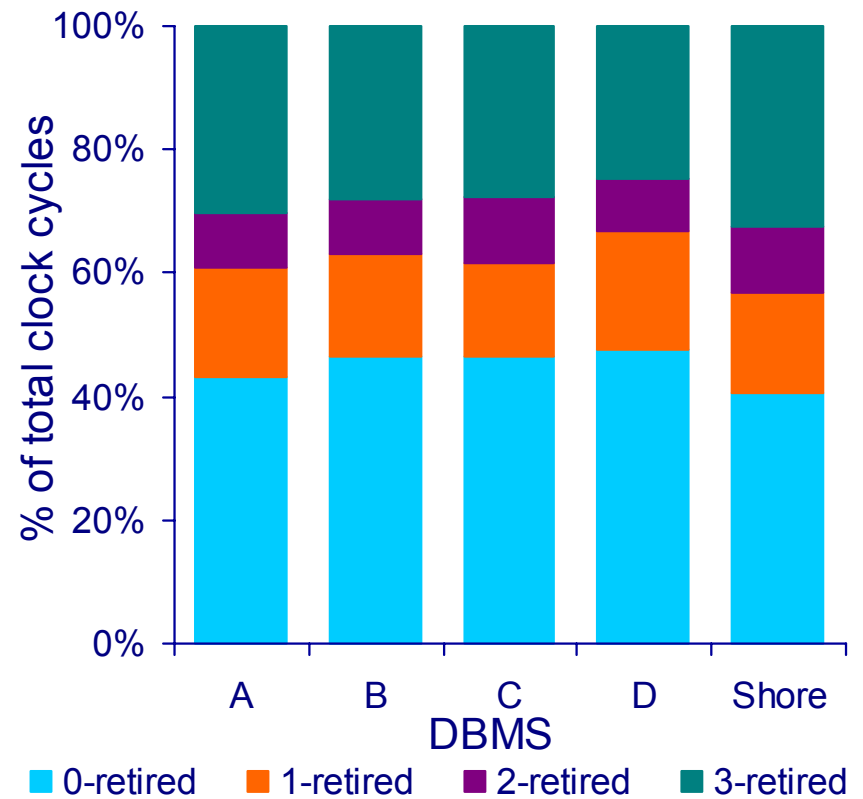
Superscalar Processor Capability

- Alpha issues at most 2 instructions / cycle (max=4)
- >60% of the time the Xeon retires 0/1 instruction (max=3)

Alpha 21164 Issue Breakdown

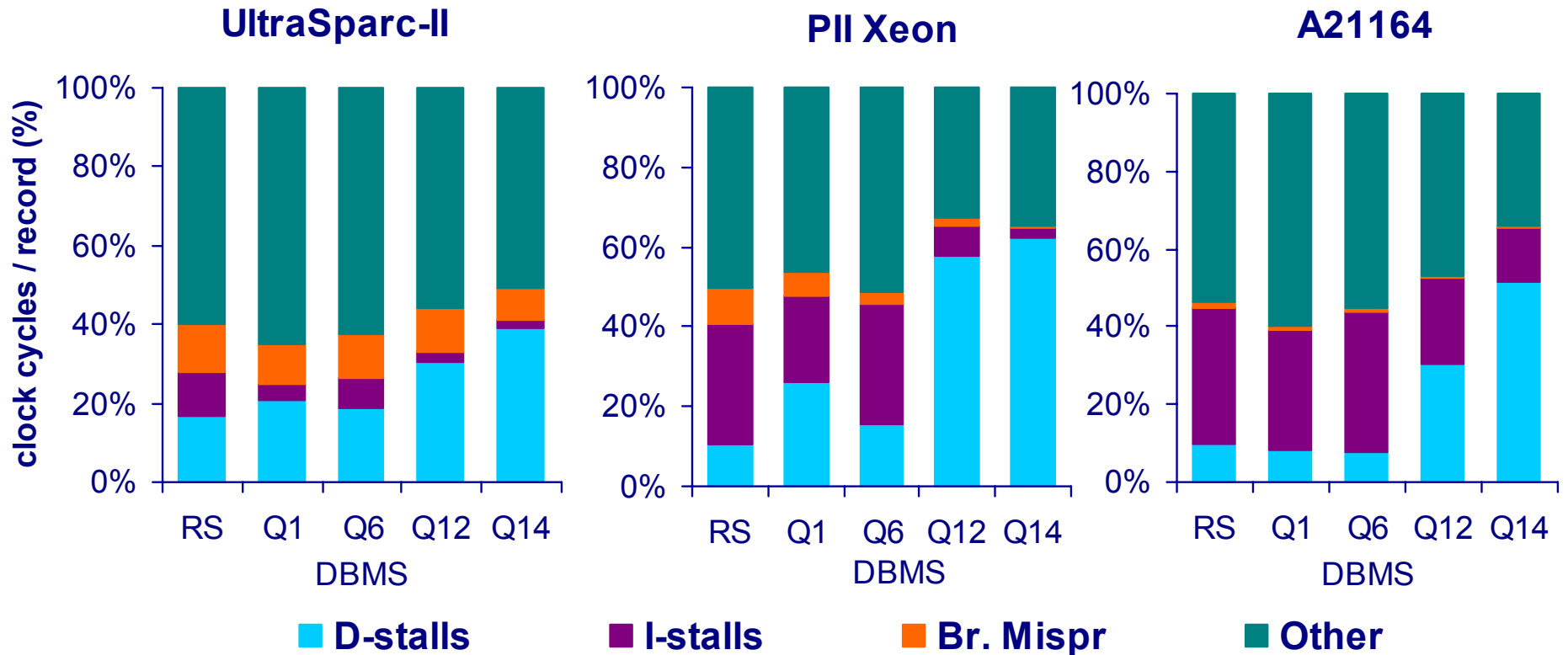


PII Xeon (NT) Retire Breakdown



The current issue/retire width remains unexploited

Clock-per-Record Breakdown



- Memory + branch misprediction stalls = 35-60% of time
- Data accesses: major memory bottleneck (esp. Q12, Q14)

Branch Prediction

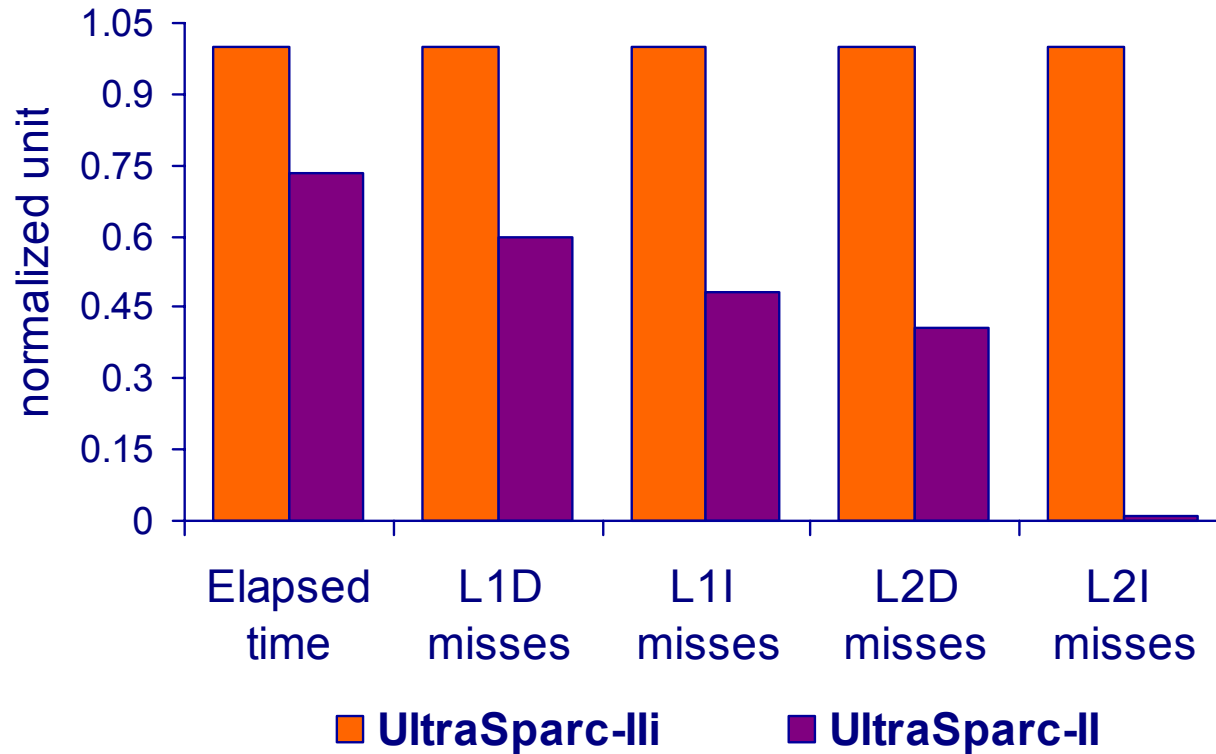
- ❑ Branch penalty = frequency*misprediction rate*penalty
- ❑ Frequency is typically 20-25%
- ❑ In-order processors => lower penalty
- ❑ Low misprediction accuracy may break it (e.g., UltraSparc)

| Characteristic | | PII Xeon | Alpha 21164 |
|---------------------------|------------|----------|-------------|
| Branch frequency | RS, Q1, Q6 | 18% | 7% |
| | Q12, Q14 | 22% | 9% |
| Branch misprediction rate | RS, Q1, Q6 | 3.5% | 15% |
| | Q12, Q14 | 1% | 6% |
| Branch penalty (cycles) | | 15 | 5 |

High-accuracy predictors

Cache Inclusion

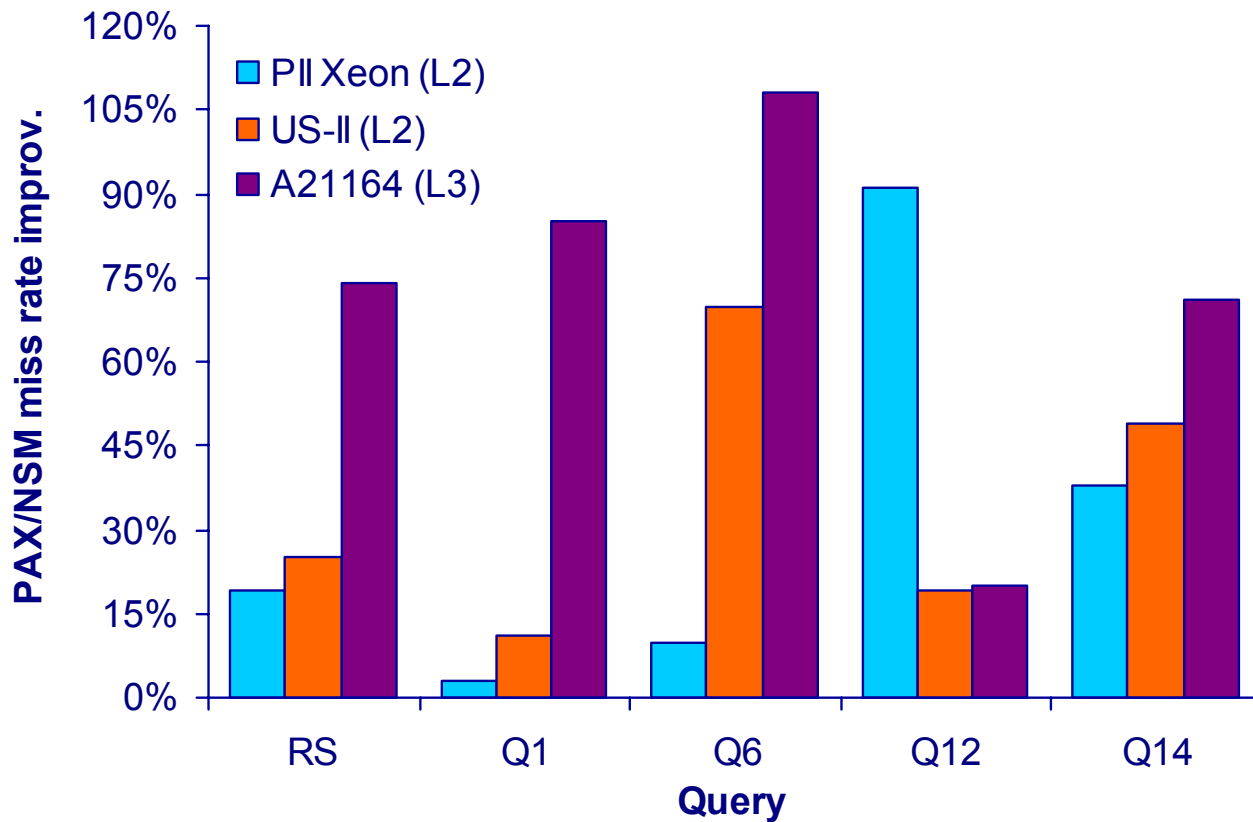
UltraSparc II/III cache comparison (RS)



Small caches should not maintain inclusion

Cache Block Size

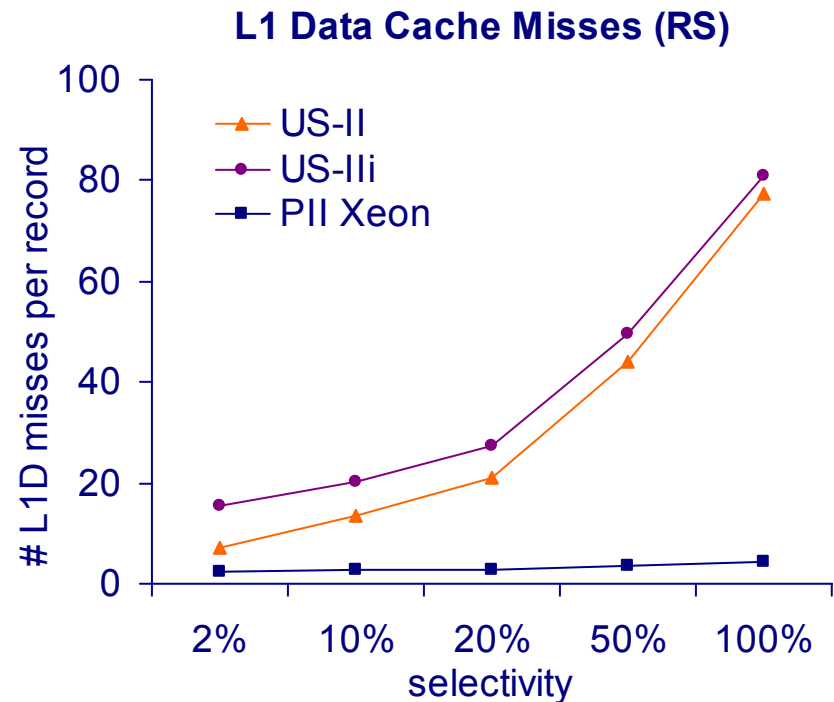
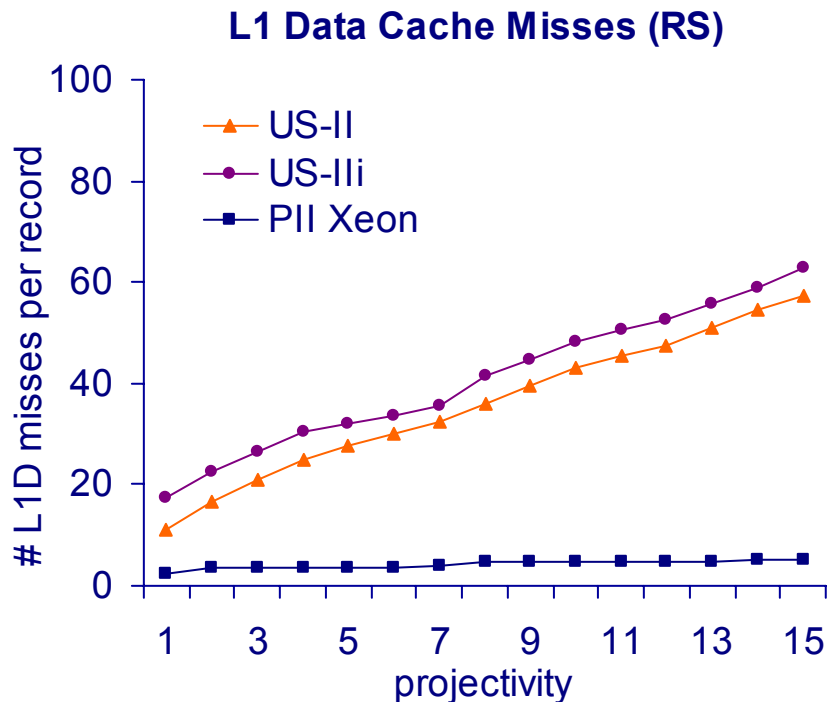
PAX savings on L3 data miss rates



Larger cache line = lower miss rates

Sub-Blocking / Associativity

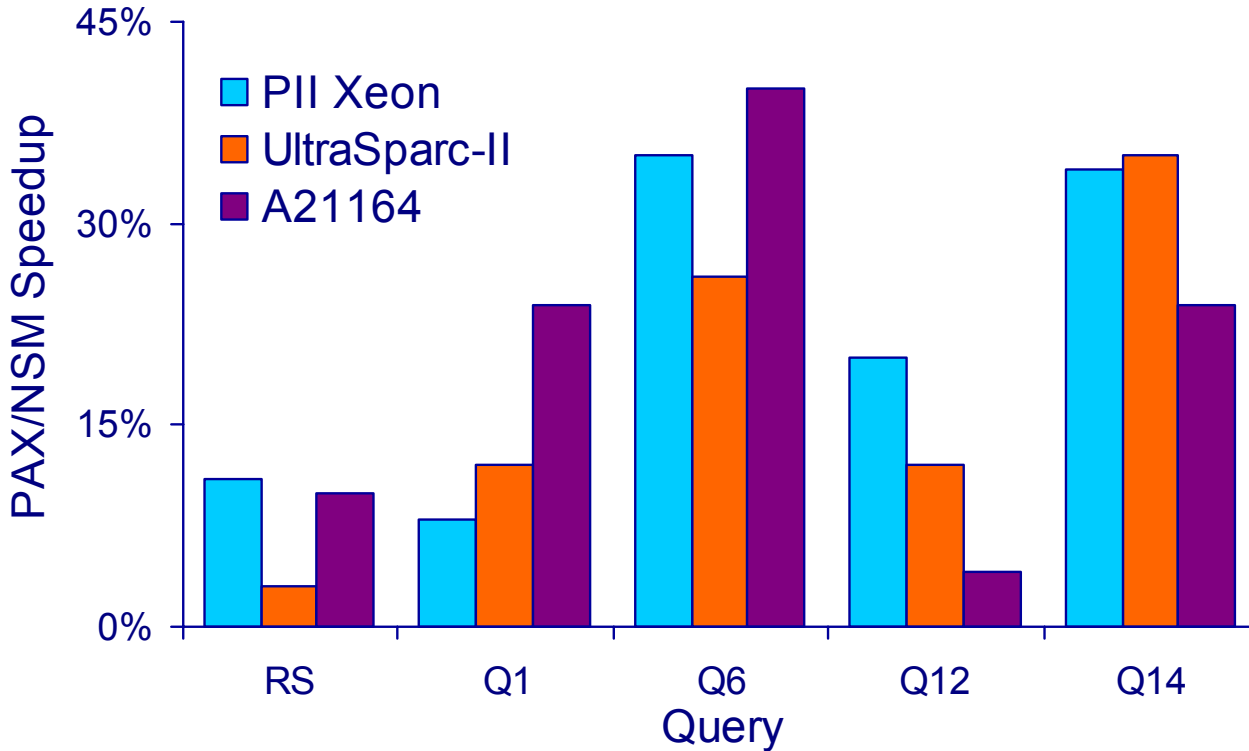
- ❑ UltraSparc: direct-mapped, subblocking (32/16)
- ❑ Xeon: 2-way, no subblocking (32/32)



High associativity, no sub-blocking

PAX vs. NSM across platforms

PAX/NSM Speedup on Unix (100MB database)



PAX improves all queries

Summary

- ❑ Memory Hierarchy
 - ❑ Non-blocking caches
 - ❑ >64-byte block, no sub-blocking
 - ❑ Generous-sized L1-I (128K) and L2 (> 2MB)
 - ❑ A tiny, fast L1/2 with a large, slow L3 won't add much
 - ❑ High associativity (2-4)
 - ❑ No inclusion (at least for instructions)
- ❑ Processor pipeline
 - ❑ Issue width is fine, out-of-order overlaps stall time
 - ❑ Execution engine to sustain >1 load/store instr.
 - ❑ High-accuracy branch prediction

...provided that implementation cost remains stable.

Conclusions

- Found trends in behavior of commercial DBMSs
 - using an analytic framework to model execution time
- Identified bottlenecks among HW components
 - Main memory access is the new DB bottleneck
 - Major showstoppers: L1 Instruction + L2 Data
- Proposed new design for cache performance
 - Increase spatial locality using novel data placement
 - 70% less data-related memory access delays
 - Significant improvement on sequential scans
- Evaluated several hardware parameters
 - Suggested DSS-centric processor and memory design