15-721 Database Management Systems

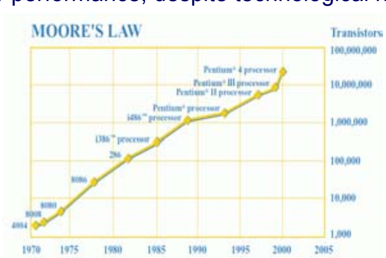
Databases and Micro-Architecture

Instructor: Anastassia Ailamaki

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

Trends in processor performance

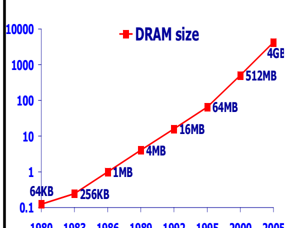
- Scaling # of transistors, innovative microarchitecture
- Higher performance, despite technological hurdles!

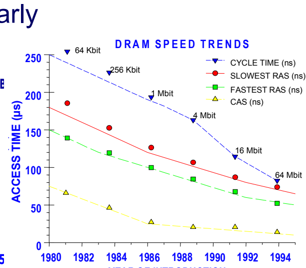


Processor speed doubles every 18 months

Trends in DRAM Performance

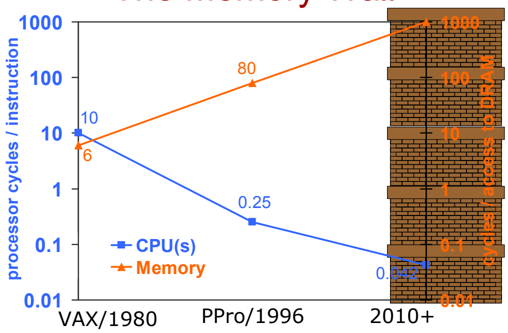
- Memory capacity increases exponentially
  - DRAM Fabrication primarily targets density
- Speed increases linearly





Larger but not as much faster memories

The Memory Wall



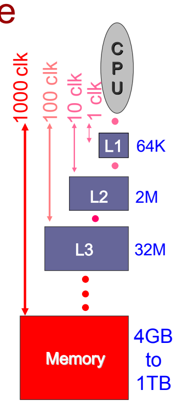
Trip to memory = thousands of instructions!

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???

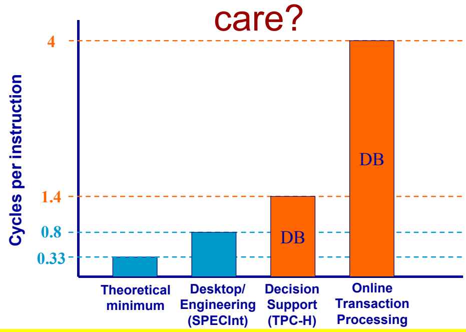


Modern storage managers

- 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

DB storage mgrs efficiently hide I/O data latencies

Why should we (databasers) care?

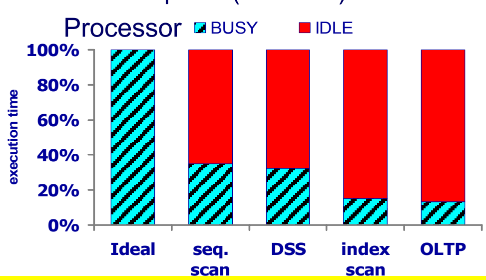


Database workloads under-utilize hardware
New bottleneck: Processor-memory delays

DB Hitting Memory Wall

[VLDB99]

On a modern computer (sans I/O)



DBMS can run MUCH faster if h/w resources are used efficiently

Outline

- Introduction
- Where does time go?
  - Background
  - Experimental setup & methodology
  - Results
  - Conclusions #1
- Weaving Relations for Cache Performance

H/W Performance Evaluation

- Benchmarks: SPEC, SPLASH, LINPACK
- Enterprise servers run commercial apps

How do database systems perform?

The DBMS New Bottleneck

- Earlier bottleneck was I/O, now memory and compute intensive (e.g., data mining)
- Modern platforms:
  - sophisticated execution hardware
  - fast, non-blocking caches and memory

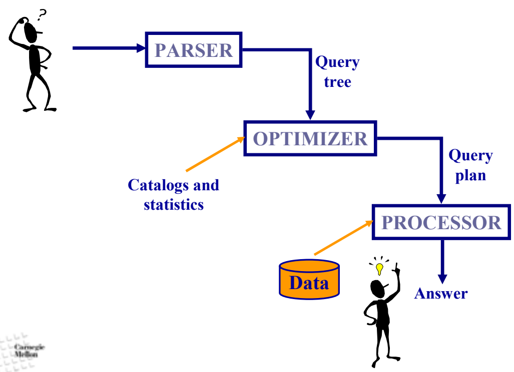
still...
DBMSs hardware behavior is suboptimal, compared to scientific workloads.

Prior Research

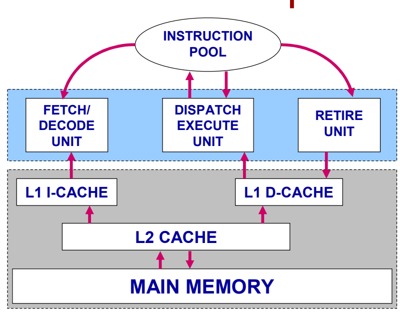
- Database research
  - smart use of cache for isolated tasks
- Architecture performance studies
  - analysis of hardware behavior shows problem

No coherent study across DBMSs and workloads

The Works of a DBMS



An Execution Pipeline



+ Branch prediction, non-blocking caches, out-of-order

Where Does Time Go?

- Computation
- Stalls
  - Cache misses
  - Branch mispredictions
  - Other execution pipeline stalls

- Stall time and computation overlap

$$\text{Time} = T_{Computation} + T_{Memory} + T_{Branch} + T_{Resource} - T_{Overlap}$$

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 PII counters

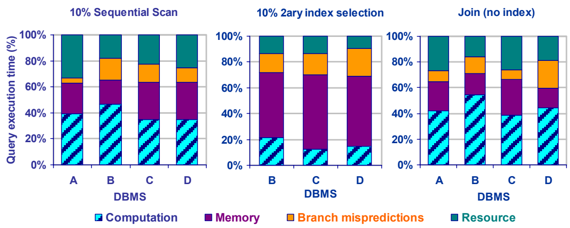
Why Simple Queries?

- Easy to setup and run
- Fully controllable parameters
- Enable iterative hypotheses
- Allow to isolate behavior of basic loops
- Building blocks for complex workloads?

Time Calculations

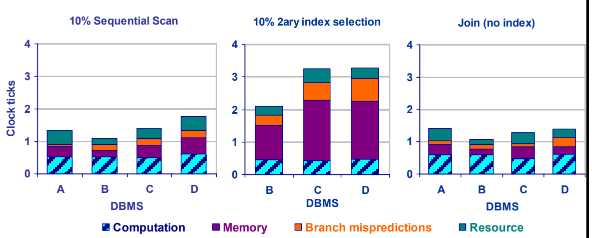
- Measured: Resource stalls, L1I stalls
- Estimated:
  - L1 data stalls: # misses * penalty
  - L2 stalls: # misses * measured memory latency
  - Branch misprediction stalls: # mispr. * penalty
- Overlap: measured CPI / expected CPI

Execution Time Breakdown (%)
Microbenchmarks



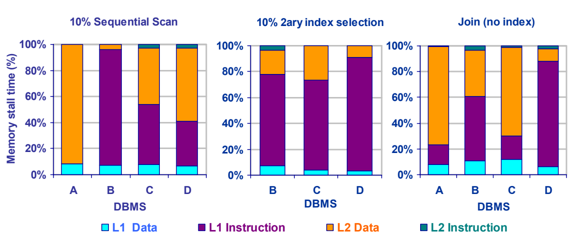
- Stalls at least 50% of time
- Memory stalls are major bottleneck

CPI (Clocks Per Instruction)
Microbenchmarks



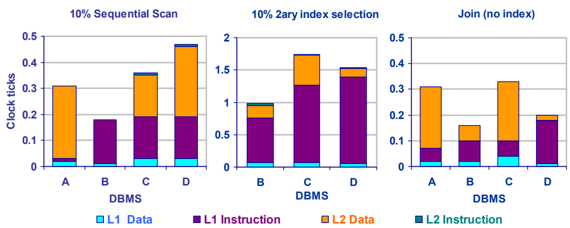
- CPI is high (compared to scientific workloads)
- Indexed access ⇦ more memory stalls per instruction

Memory Stalls Breakdown (%)
Microbenchmarks

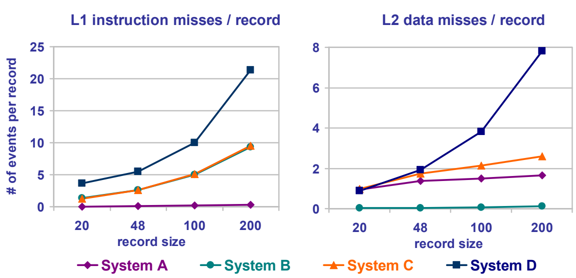


- Role of L1 data cache unimportant
- L1 instruction and L2 data stalls dominate
- Different memory bottlenecks across DBMSs and queries

Memory Stall CPI Breakdown Microbenchmarks



L1 Instruction / L2 Data Misses



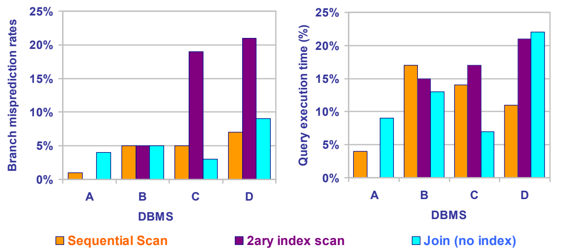
- L1I and L2D increase as a function of record size
- Why???

Memory Bottlenecks

- Stalls due to L2 cache data misses
  - Compulsory or repeated
  - L2 grows (8MB), but will be slower
- Stalls due to L1 I-cache misses
  - Possible causes: invalidations, OS, page code
  - L1 I-cache not likely to grow as much as L2

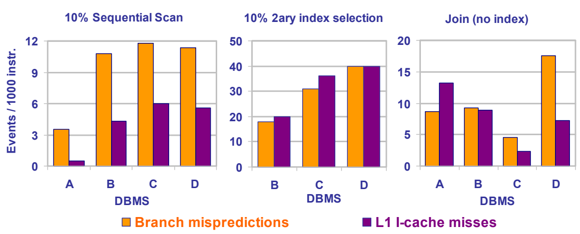
(lots of) further research needed in area

Branch Mispredictions



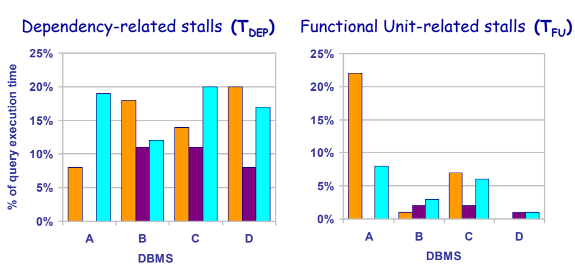
- Branch misprediction stall time always significant
- Larger BTB will reduce mispredictions

Branch Mispredictions Vs. L1 I-cache Misses



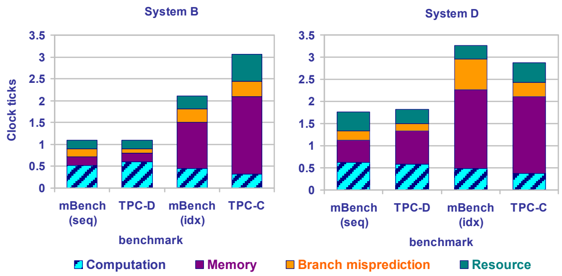
- More branch mispredictions incur more L1I misses
- Index code more complicated - needs optimization

Resource-related Stalls



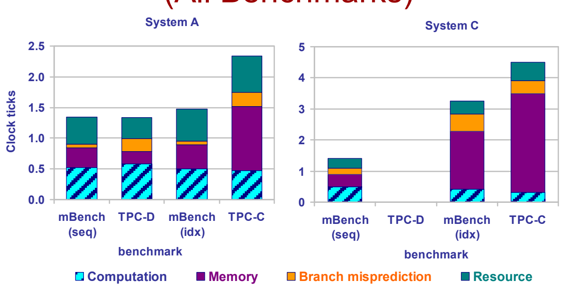
- High T_{DEP} for all systems : Low ILP opportunity
- A's sequential scan: Memory unit load buffers?

CPI Breakdown (B, D)
(All Benchmarks)



- Microbenchmark breakdown similar to TPC-D
- TPC-C: higher CPI, much higher memory stalls

CPI Breakdown (A, C)
(All Benchmarks)



- Microbenchmark breakdown similar to TPC-D
- TPC-C: higher CPI, much higher memory stalls

Summary of Results

- All stalls are significant
- Memory stalls dominate
  - L1 data stalls negligible
  - Instruction and L2 data stalls important
  - Relative contribution varies
- Indices break the caches
- Sequential scan & TPC-D, index & TPC-C
- TPC-C workloads incur more memory stalls

Conclusions #1

- First in-depth analysis across DBMSs
- Execution time breakdown shows trends
- Common bottleneck characterization:
  - Instruction misses on the first-level cache
  - Data misses on the second-level cache
- Focus on index access code
- TPC may not be necessary to locate bottlenecks

Outline

- Introduction
- Where Does Time Go?
- **Weaving Relations for Cache Performance**
  - **What's wrong with slotted pages?**
  - Partition Attributes Across (PAX)
  - Performance results
  - Conclusions #2

Data Placement on Disk Pages

- Commercial DBMSs use *Slotted pages*
  - ✓ Store table records sequentially
  - ☺ Intra-record locality (attributes of record *r* together)
  - ☹ Doesn't work well on today's memory hierarchies
- Alternative: *Vertical partitioning* [Copeland'85]
  - ✓ Store *n*-attribute table as *n* single-attribute tables
  - ☺ Inter-record locality, saves unnecessary I/O
  - ☹ Destroys intra-record locality => expensive to reconstruct record
- **Contribution: Partition Attributes Across**
  - ☺ **… have the cake and eat it, too**

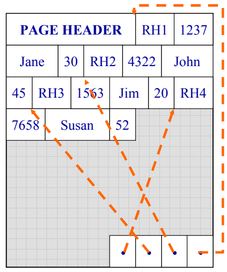
Inter-record locality + low reconstruction cost

Current Scheme: Slotted Pages

Formal name: NSM (N-ary Storage Model)

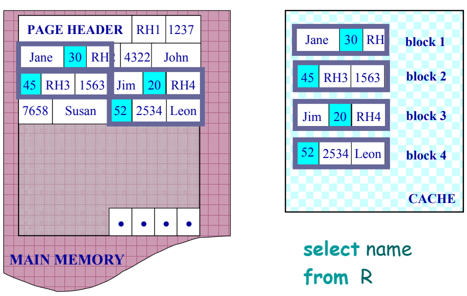
R

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

Predicate Evaluation using NSM



select name
from R
where age > 50

NSM pushes non-referenced data to the cache

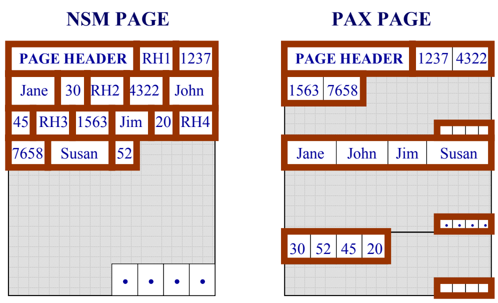
Need New Data Page Layout

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

and, most importantly, is…

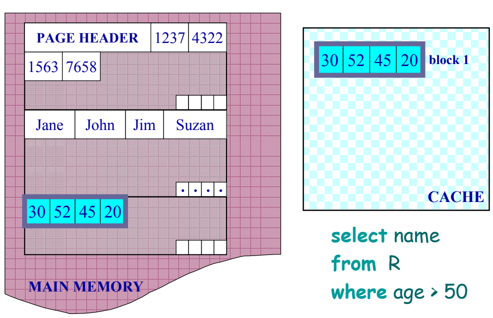
low-implementation-cost, high-impact

Partition Attributes Across (PAX)



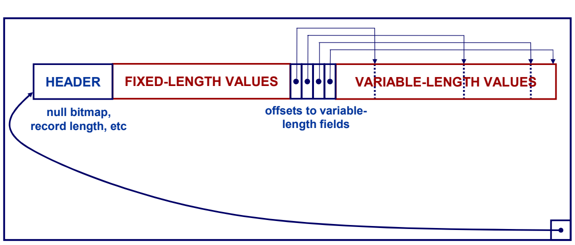
Partition data *within* the page for spatial locality

Predicate Evaluation using PAX



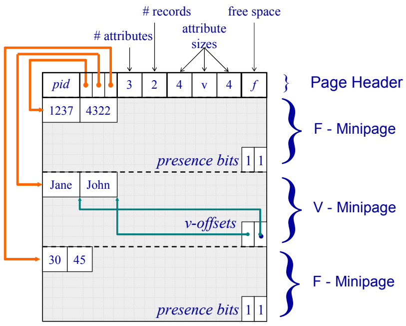
Fewer cache misses, low reconstruction cost

A Real NSM Record



NSM: All fields of record stored together + slots

PAX: Detailed Design



PAX: Group fields + amortizes record headers

Outline

- Introduction
- Where Does Time Go?
- **Weaving Relations for Cache Performance**
  - What's wrong with slotted pages?
  - Partition Attributes Across (PAX)
  - **Performance results**
  - Conclusions #2

Sanity Check: Basic Evaluation

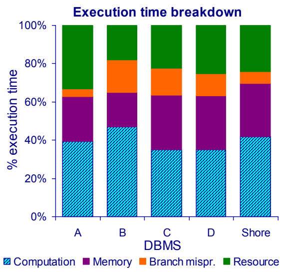
- Main-memory resident R, numeric fields
- Query:

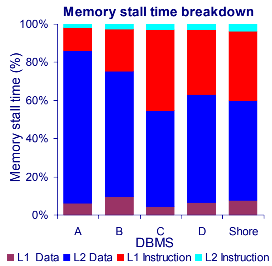
  select avg (a_i)
  from R
  where a_j >= Lo **and** a_j <= Hi

- 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?

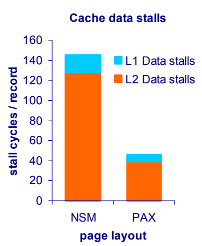
- Compare Shore query behavior with commercial DBMS
- Execution time & memory delays (range selection)

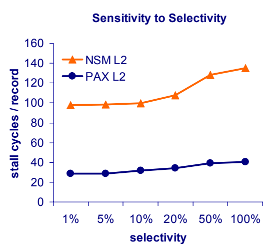




We can use Shore to evaluate workload behavior

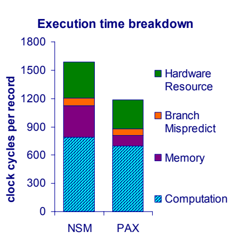
Effect on Accessing Cache Data

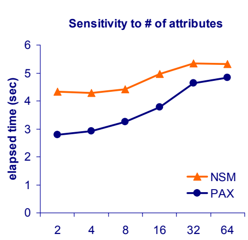




- **PAX saves 70% of NSM's data cache penalty**
- **PAX reduces cache misses at both L1 and L2**
- **Selectivity doesn't matter for PAX data stalls**

Time and Sensitivity Analysis

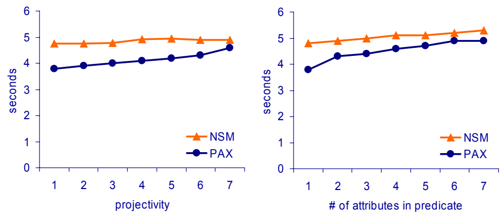




- **PAX: 75% less memory penalty than NSM (10% of time)**
- **Execution times converge as number of attrs increases**

Sensitivity Analysis (2)

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

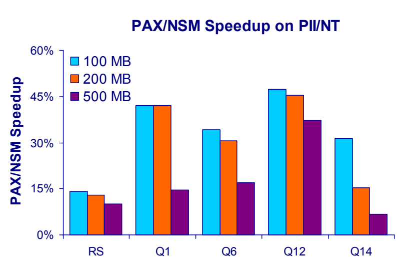


PAX,NSM times converge as query covers entire tuple

Evaluation Using DSS

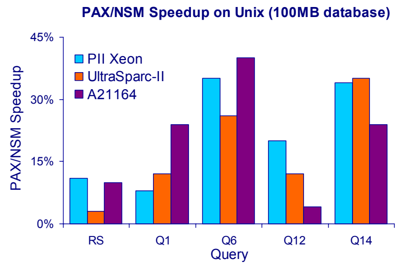
- 100M, 200M, and 500M TPC-H DBs
- Queries:
  1. Range Selections w/ variable parameters (RS)
  2. TPC-H Q1 and Q6
     - sequential scans
     - lots of aggregates (*sum*, *avg*, *count*)
     - grouping/ordering of results
  3. TPC-H Q12 and Q14
     - (Adaptive Hybrid) Hash Join
     - complex 'where' clause, conditional aggregates
- 128MB buffer pool

TPC-H Queries: Speedup



- **PAX improves performance even with I/O**
- **Speedup differs across DB sizes**

PAX vs. NSM across platforms



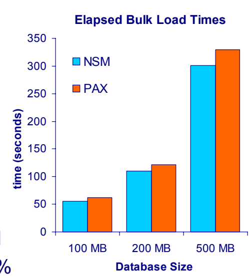
PAX improves performance across platforms

49

Insertions

- 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 to accommodate a single record
- Threshold 10%: penalty =0.8%



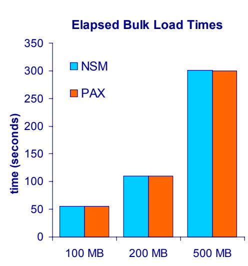
Initial load penalty: 2-10% for a TPC-H DB

50

Insertions (UPDATED Results)

- Follow described algorithm
- **Use Histograms to Allocate Optimal Page (as w/ NSM)**

- 50% of reorganizations to accommodate a single record
- **Reorganizations do not incur a measurable cost**



PAX does not incur a penalty on insertions

51

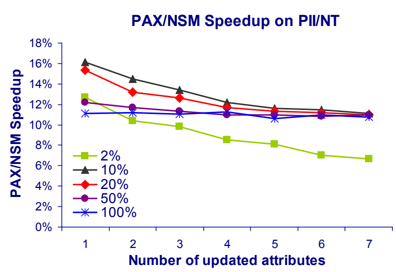
Updates

- Policy: Update in-place
- Variable-length: Shift when needed
- PAX only needs shift minipage data

- Update statement:

  update R
  set $a_p = a_p + b$
  where $a_q >$ Lo and $a_q <$ Hi

Updates: Speedup



- **PAX always speeds queries up (7-17%)**
- **Lower selectivity => reads dominate speedup**
- **High selectivity => write-backs dominate speedup**

Conclusions #2

- PAX: a *low-cost, high-impact* DP technique

- Performance
  - Eliminates unnecessary memory references
  - High utilization of cache space/bandwidth
  - Faster than NSM (does not affect I/O)

- Usability
  - Orthogonal to other storage decisions
  - “Easy” to implement in large existing DBMSs