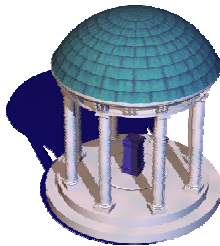


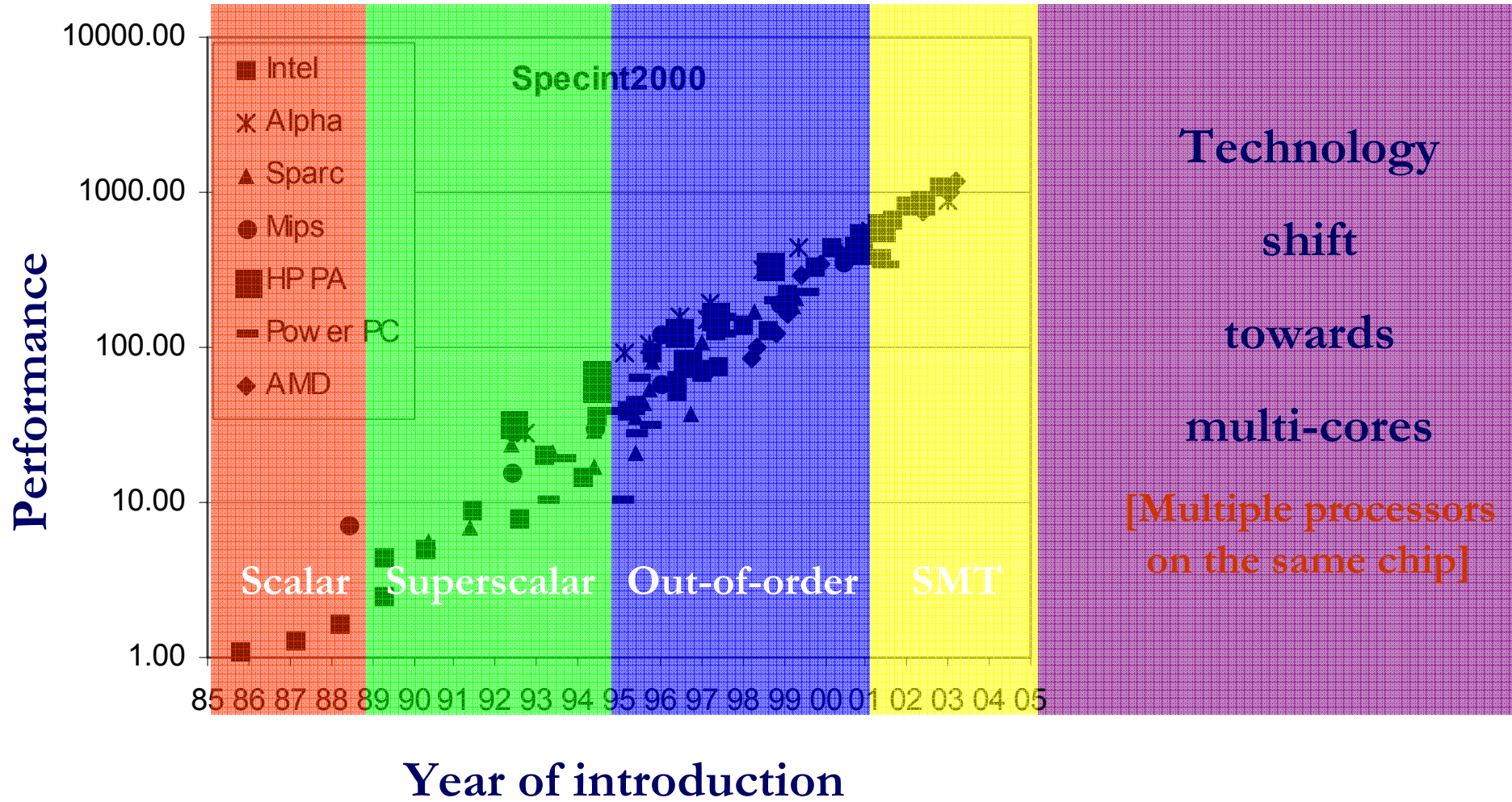
Query co-Processing on Commodity Processors

Anastassia Ailamaki
Carnegie Mellon University

Naga K. Govindaraju
Dinesh Manocha
University of North Carolina at Chapel Hill



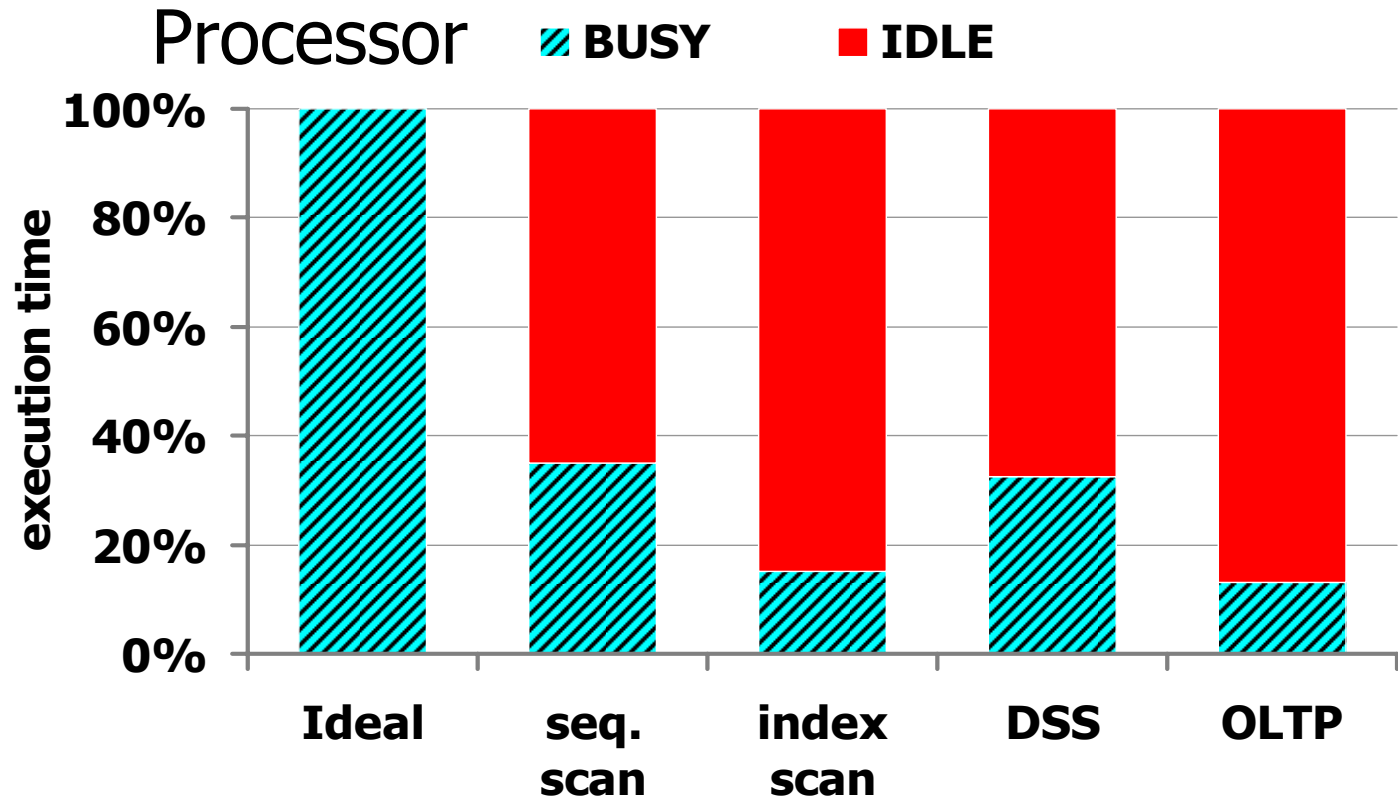
Processor Performance over Time*



*graph courtesy of Rakesh Kumar

Focus of this Seminar

- DB workload execution on a modern computer



How can we explore new hardware to run database workloads efficiently?

Detailed Seminar Outline

- INTRODUCTION AND OVERVIEW
 - How computer architecture trends affect database workload behavior
 - CPUs, NPU, and GPU: opportunity for architectural study
- DBs on CONVENTIONAL PROCESSORS
 - Query Processing: Time breakdowns, bottlenecks, and current directions
 - Architecture-conscious data management: limitations and opportunities
- QUERY co-PROCESSING: NETWORK PROCESSORS
 - TLP and network processors
 - Programming model
 - Methodology & Results
- QUERY co-PROCESSING: GRAPHICS PROCESSORS
 - Graphics Processor Overview
 - Mapping Computation to GPU
 - Database and data mining applications
- CONCLUSIONS AND FUTURE DIRECTIONS

Outline

- INTRODUCTION AND OVERVIEW



- **Computer architecture trends and DB workloads**

- Processor/memory speed gap
- Instruction-level parallelism (ILP)
- Chip multiprocessors and multithreading

- **CPUs, NPUs, and GPUs**

- DBs on CONVENTIONAL PROCESSORS

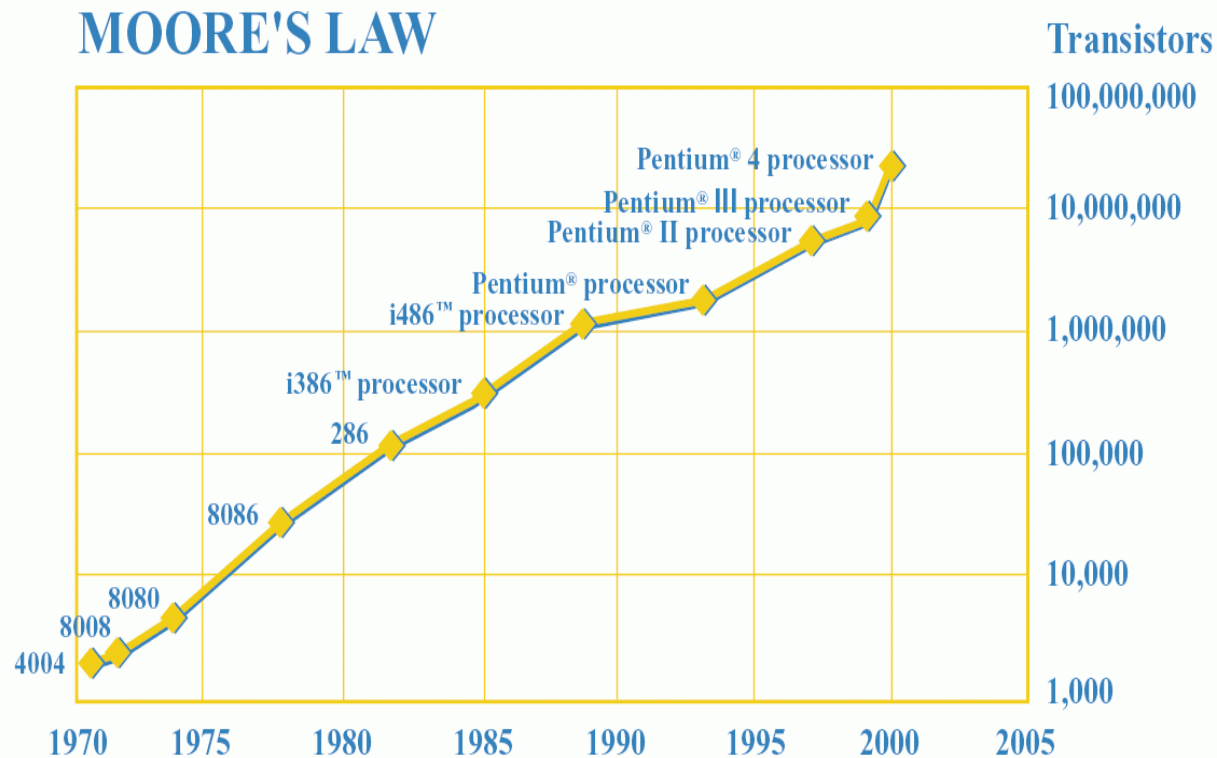
- QUERY co-PROCESSING: NETWORK PROCESSORS

- QUERY co-PROCESSING: GRAPHICS PROCESSORS

- CONCLUSIONS AND FUTURE DIRECTIONS

Processor speed

- Scale # of transistors, innovative microarchitecture
- Moore's Law (despite technological hurdles!)



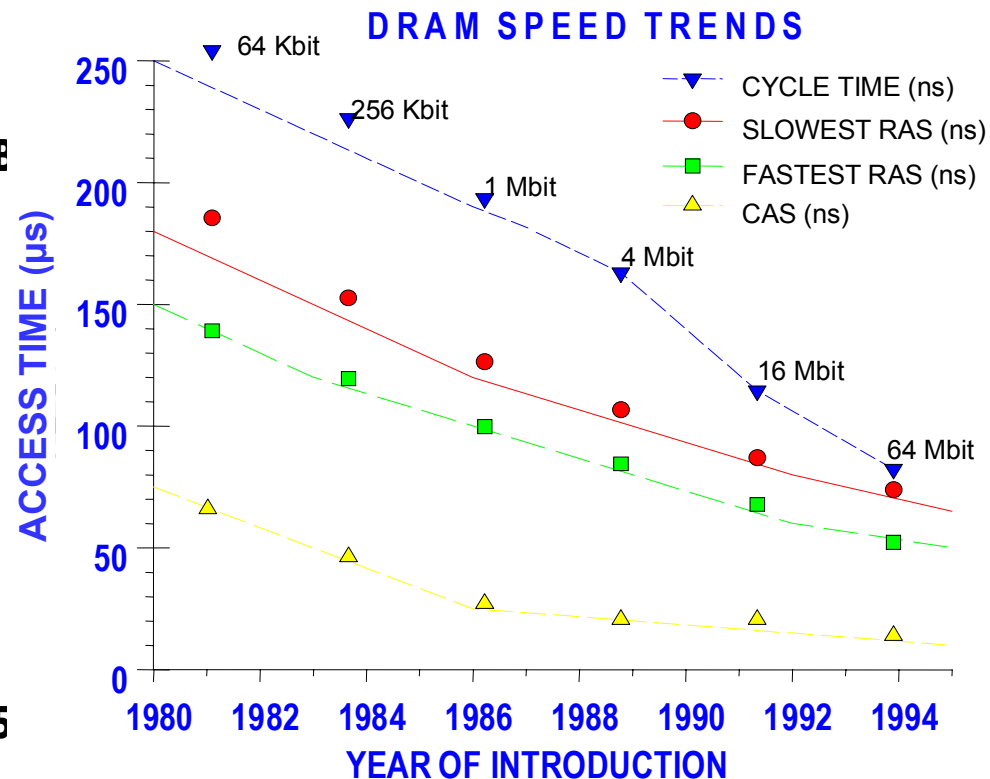
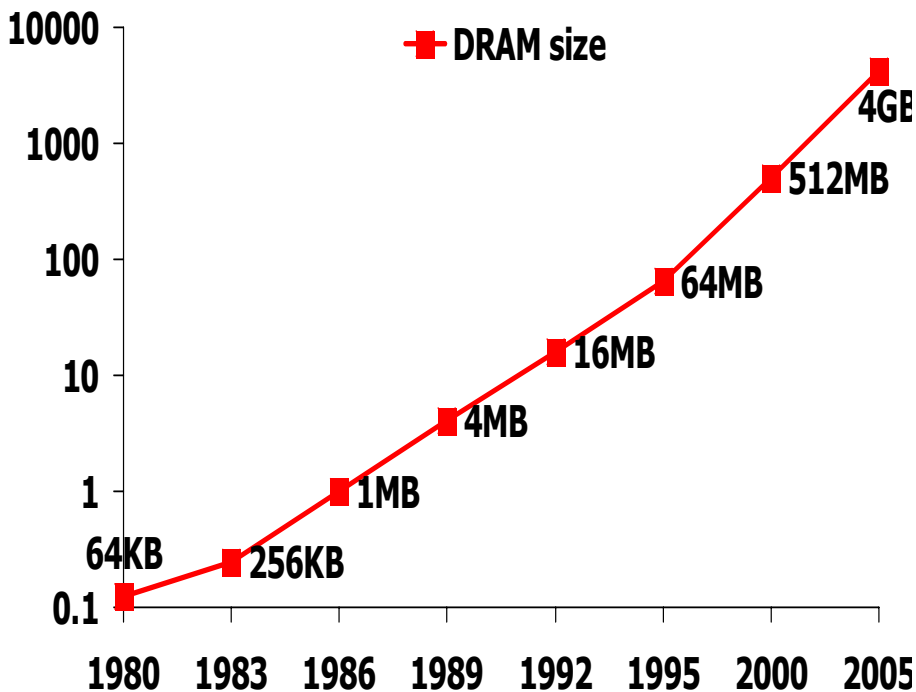
2x processor speed every 18 months

Memory speed and capacity

- Memory capacity increases exponentially

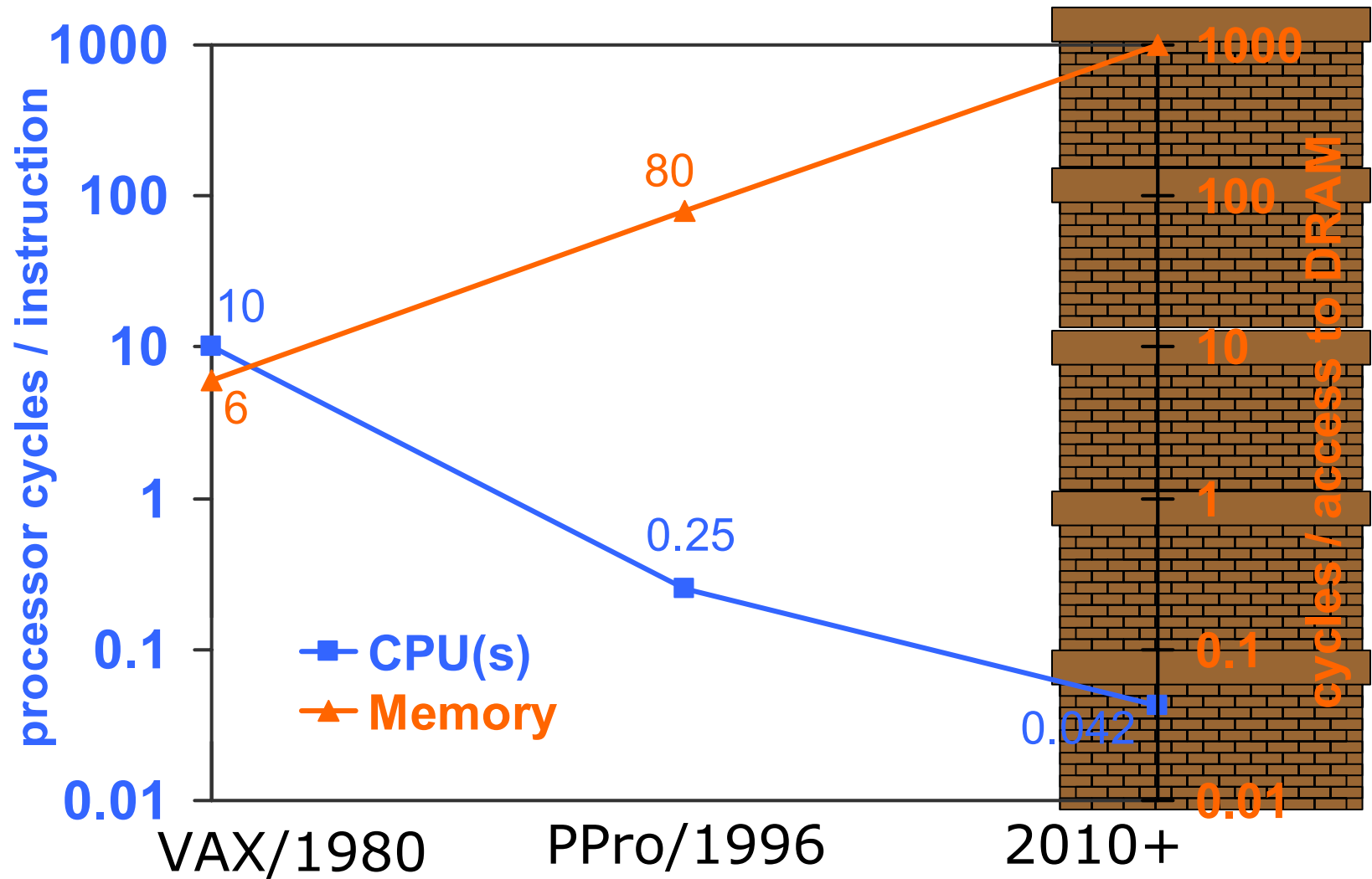
 - DRAM Fabrication primarily targets density

- Speed increases linearly



Larger but not as much faster memories

The Memory Wall



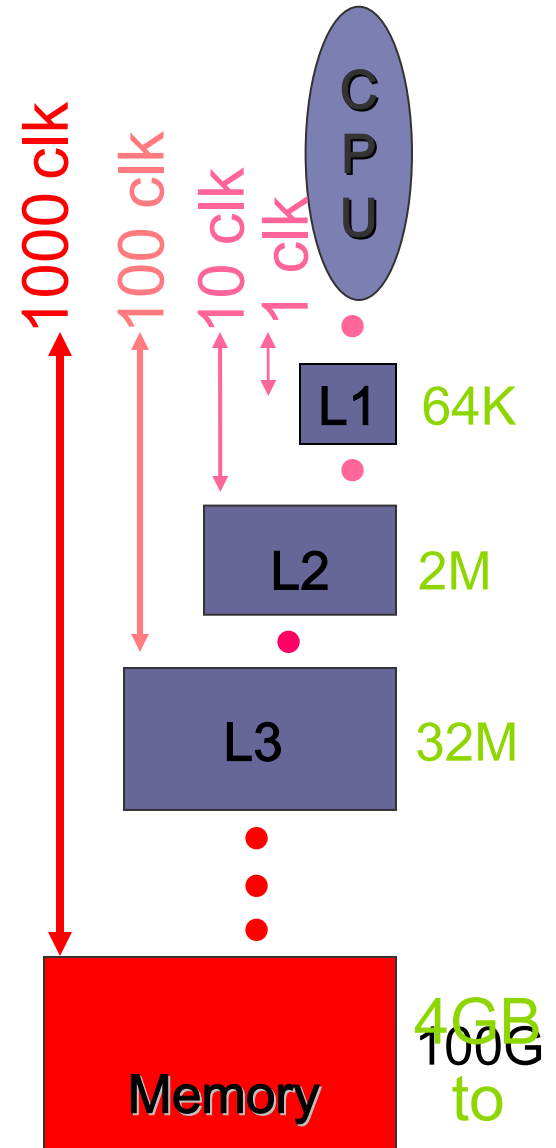
Trip to memory = 1000s of instructions!

Memory hierarchies

- 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**



Efficient cache utilization is crucial!

ILP: Processor pipelines

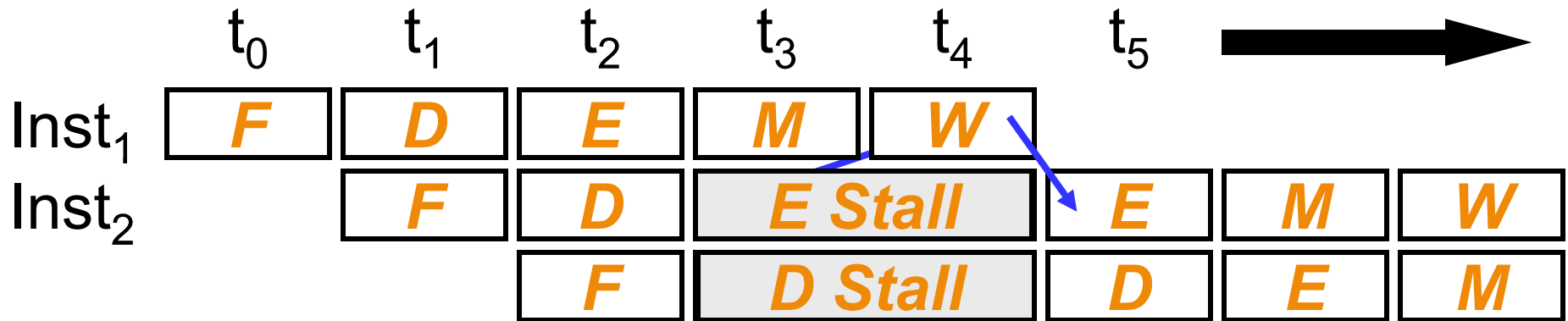
- Problem: dependences between instructions

- E.g.,

Inst₁: $r1 \leftarrow r2 + r3$

Inst₂: $r4 \leftarrow r1 + r2$

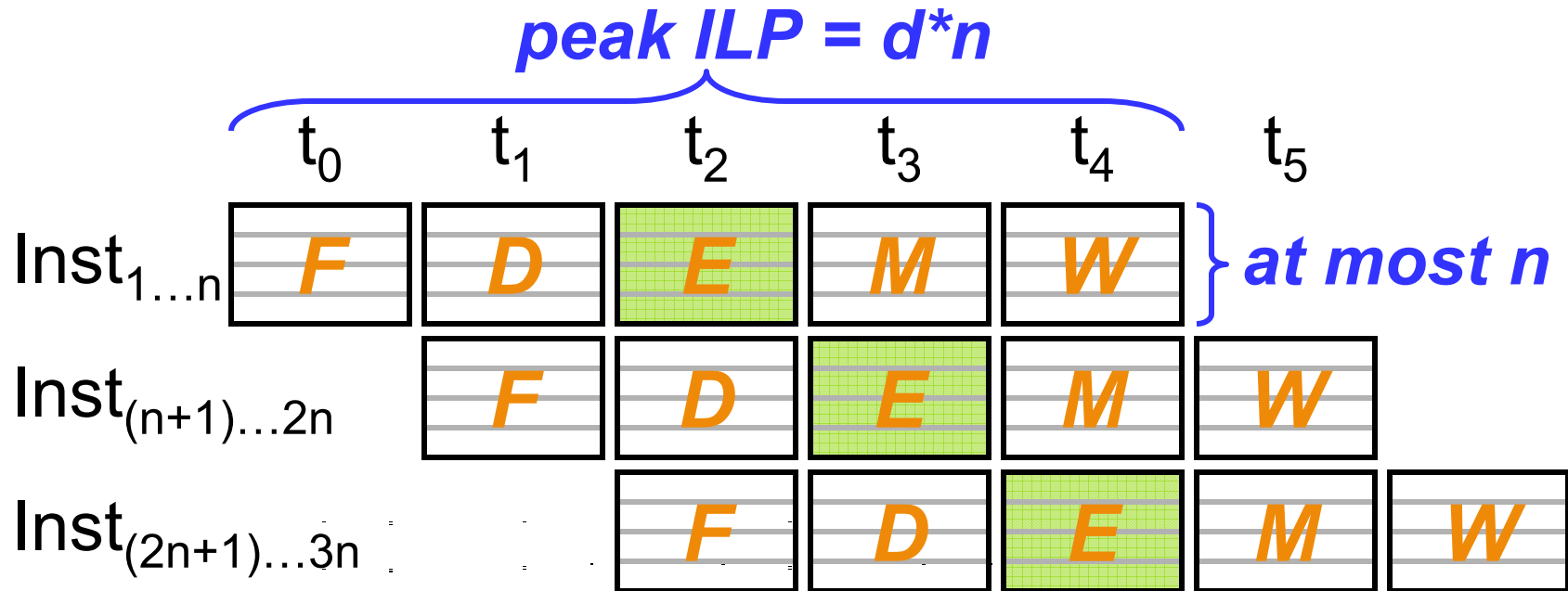
Read-after-write (RAW)



Real CPI < 1; Real ILP < 5

DB apps: frequent data dependences

> ILP: Superscalar Out-of-Order



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

● Out-of-order (vs. "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

>> ILP: Branch Prediction

● Which instruction block to fetch?

- Evaluating a branch condition causes pipeline stall

XXXX

if C goto B

A: aaaa

aaaa

aaaa

aaaa

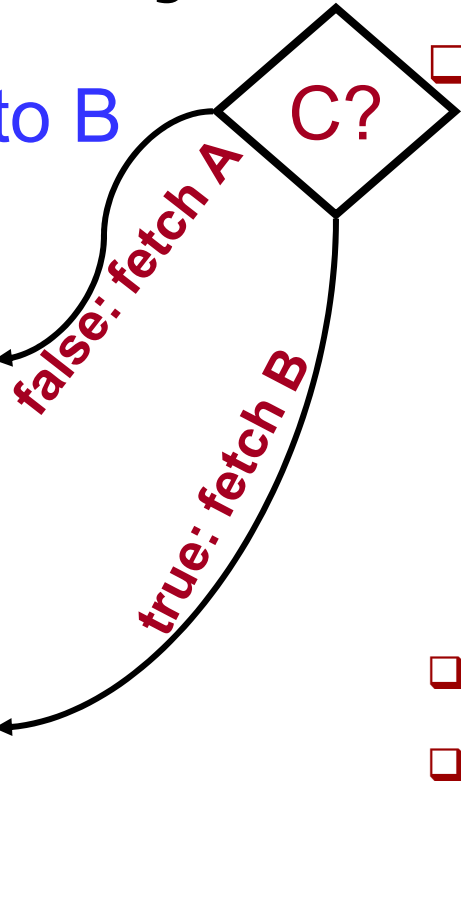
B: bbbb

bbbb

bbbb

bbbb

bbbb

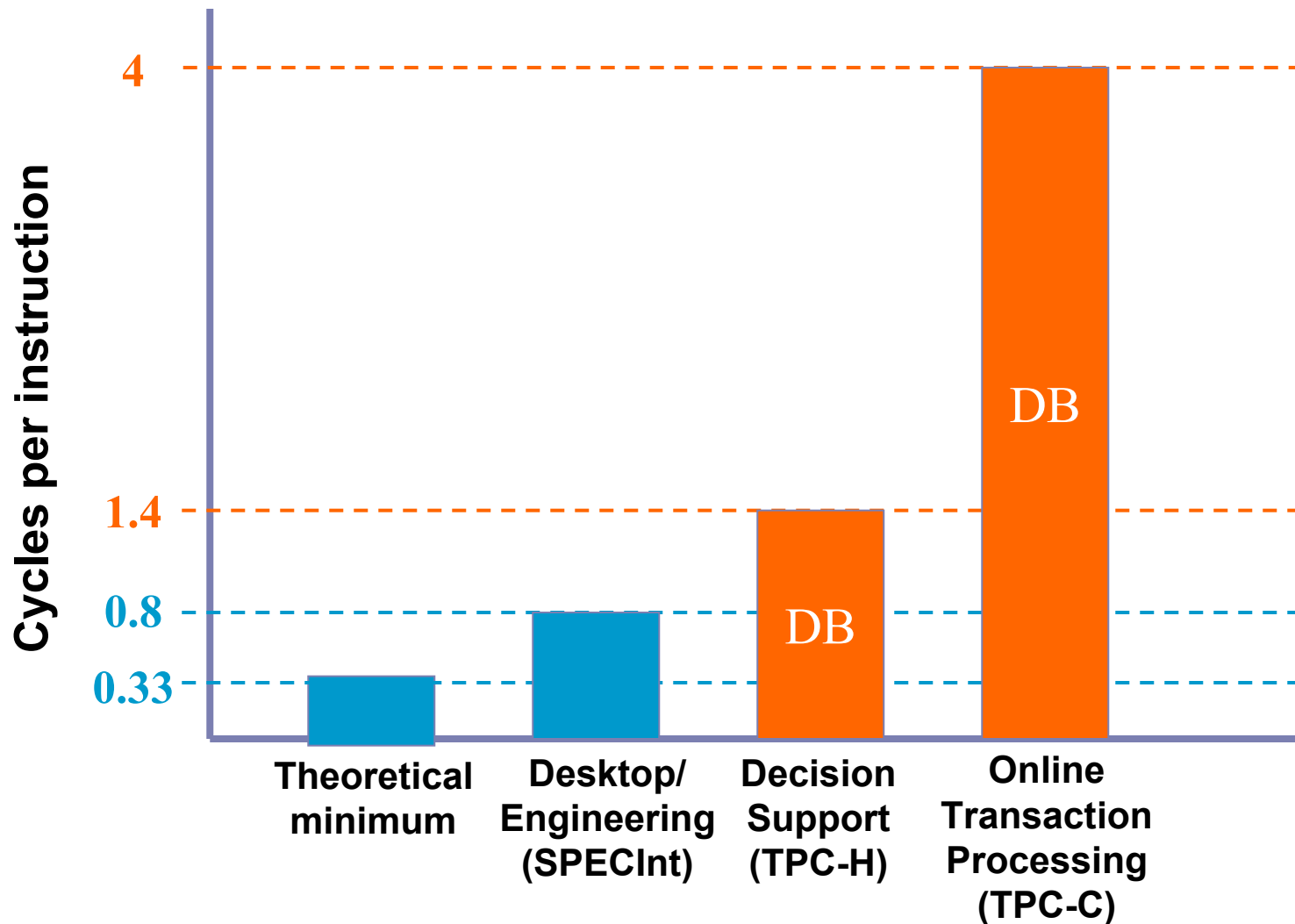


❑ **IDEA: Speculate branch *while evaluating C!***

- Record history, predict A/B
- ✓ If correct, saved a (long) delay!
- 💣 If incorrect, misprediction penalty:
 1. Flush pipeline
 2. Fetch correct instruction stream
- ❑ Excellent predictors (97% accuracy!)
- ❑ Mispredictions costlier in OOO
 - ❑ 1 lost cycle = >>1 missed instructions!

DB programs: long code paths => mispredictions

Database workloads on UPs



DB apps heavily under-utilize hardware

Outline

- INTRODUCTION AND OVERVIEW

- **Computer architecture trends and DB workloads**

- Processor/memory speed gap
 - Instruction-level parallelism (ILP)
 - Chip multiprocessors and multithreading

- **CPUs, NPUs, and GPUs**

- DBs on CONVENTIONAL PROCESSORS

- QUERY co-PROCESSING: NETWORK PROCESSORS

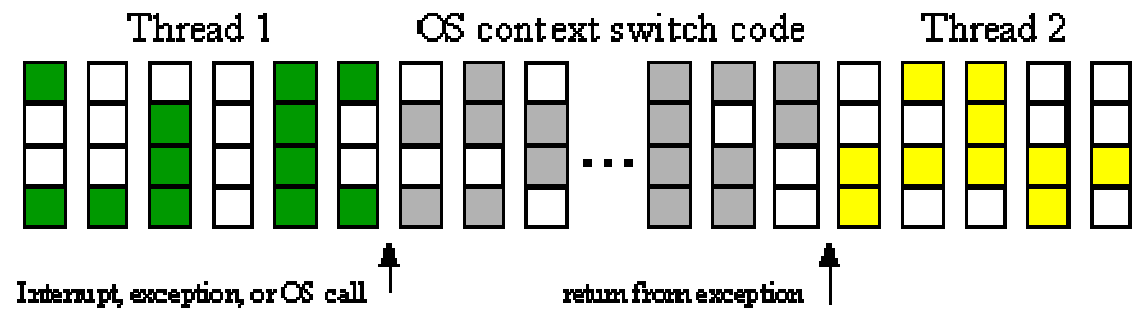
- QUERY co-PROCESSING: GRAPHICS PROCESSORS

- CONCLUSIONS AND FUTURE DIRECTIONS

Coarse-grain parallelism

- Simultaneous multithreading (SMT)
 - Store multiple contexts in different register sets
 - Multiplex functional units between threads
 - Simultaneously execute different contexts
 - Fast context switching amongst threads
- Chip multiprocessors (CMPs)
 - >1 complete processors on a single chip
 - Every functional unit of a processor is duplicated

A)
Conventional
Processor



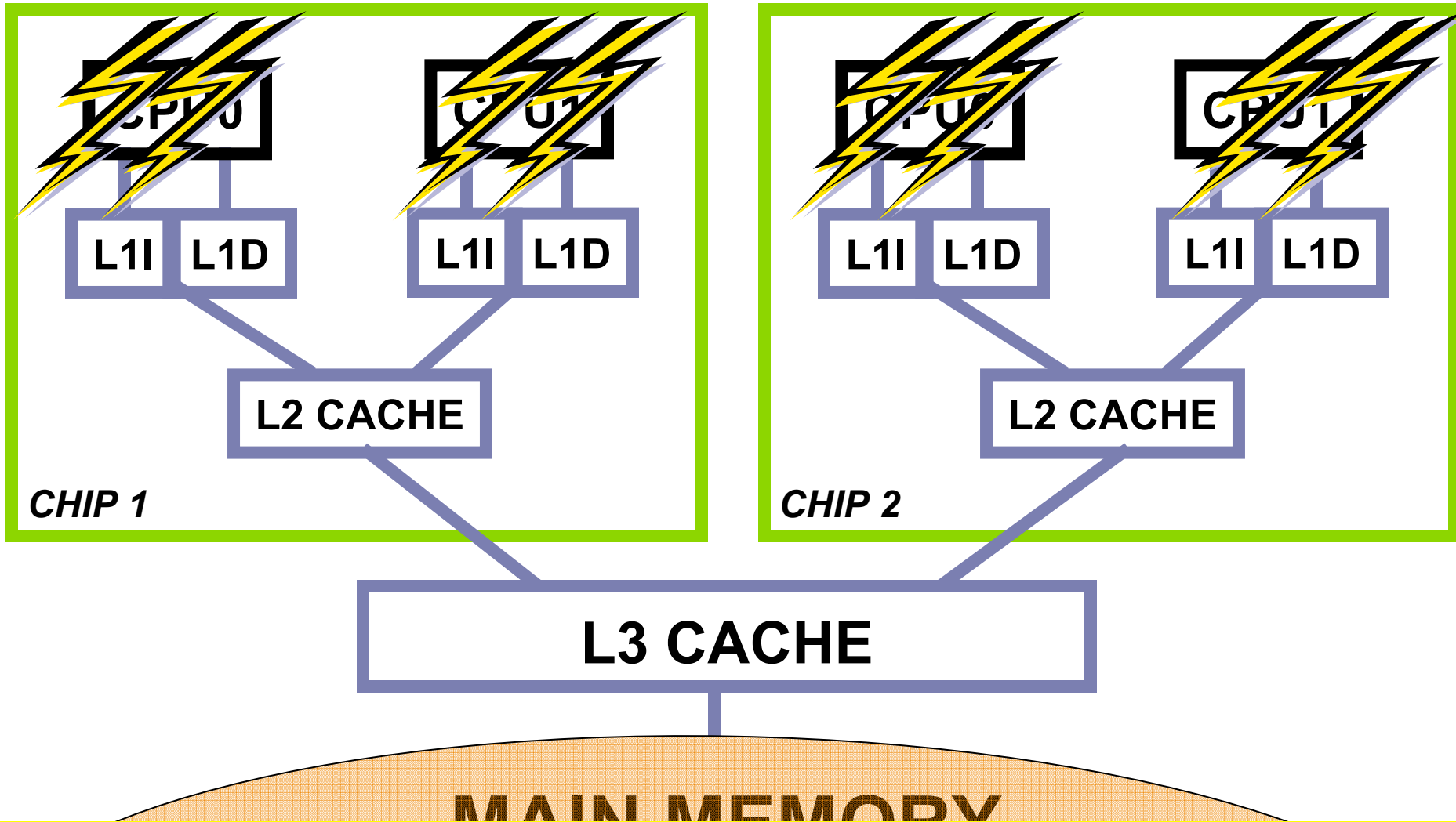
Speedup: OLTP 3x, DSS 0.5x [LBE98]

Why CMPs?

- Getting diminishing returns
 - from a single-threaded core, although powerful (OoO, superscalar, coffee-making)
 - from a very large cache
- n-core CMP outperforms n-thread SMT
- Smaller geometries: address within 1 cycle
- CMPs offer productivity advantages
- Moore's law: $2x$ *transistors* every 18 months
 - More, not faster

Expect exponentially more cores

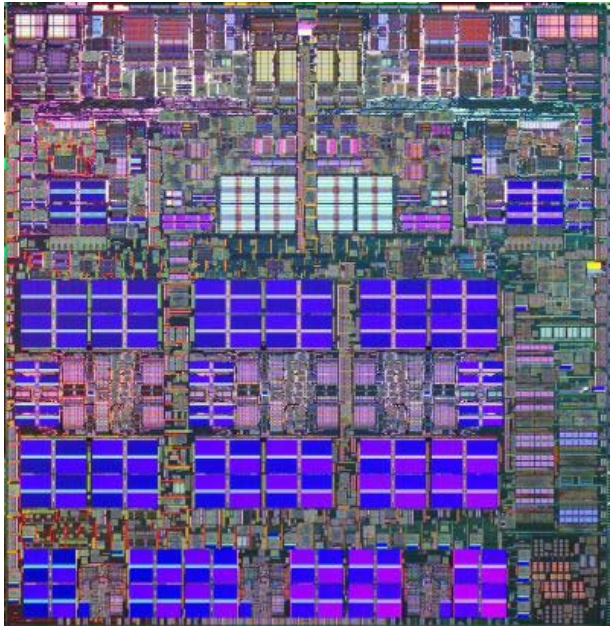
A chip multiprocessor



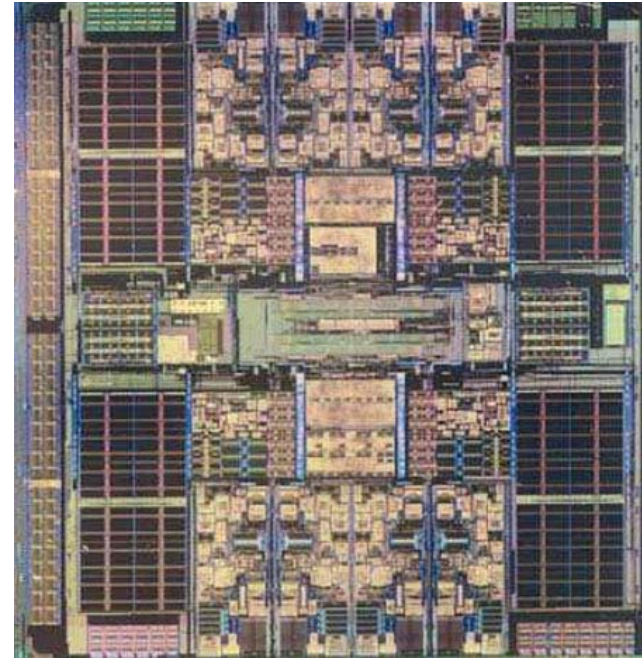
Highly variable memory latency

Speedup: OLTP 3x, DSS 2.3x on Piranha [BGM00]

Current CMP technology



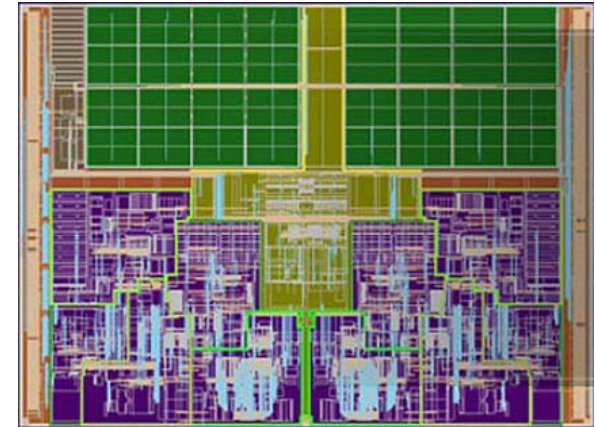
IBM Power 5



Sun Niagara



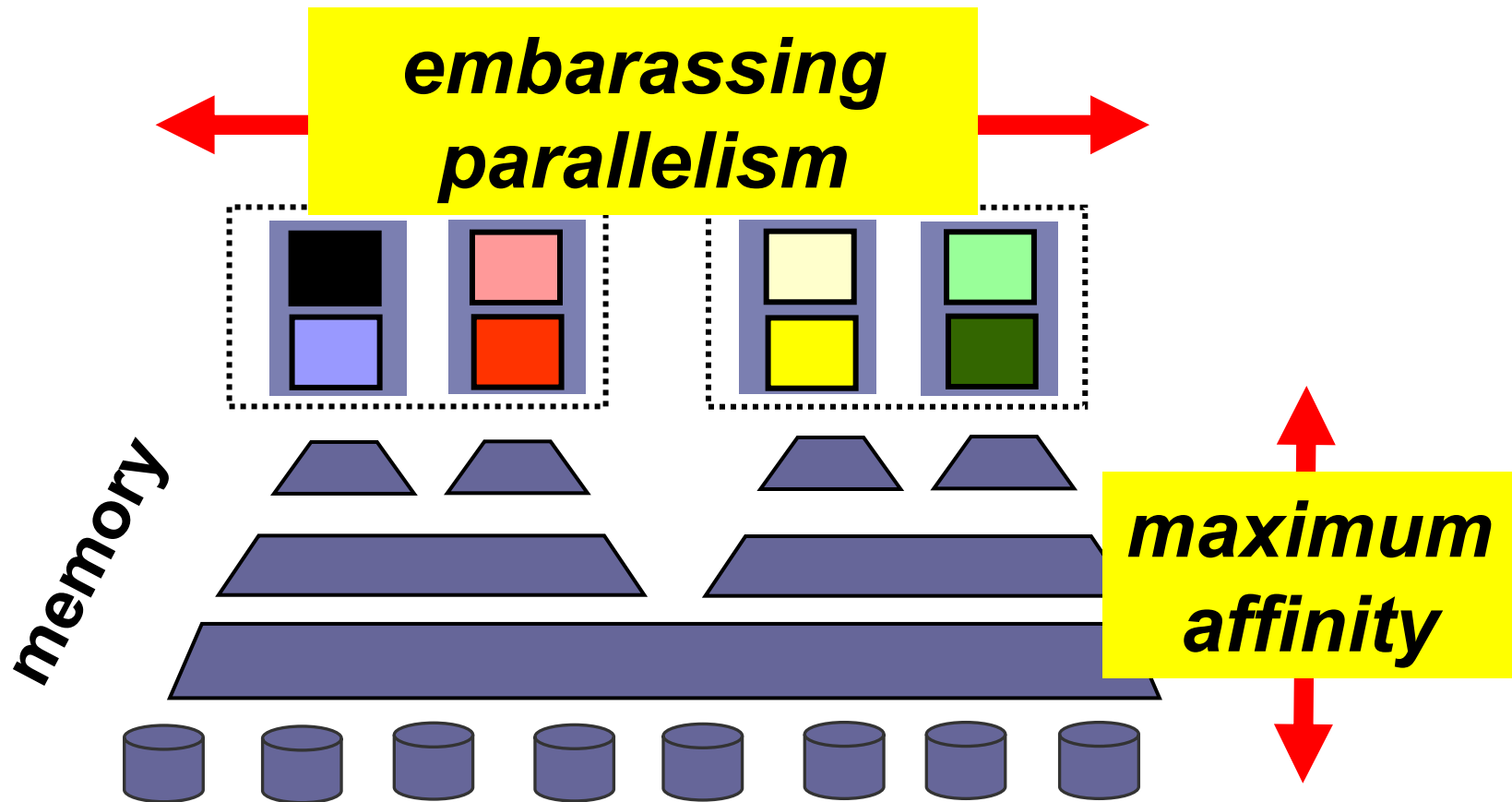
AMD Opteron



Intel Yonah

8x4=32 threads - how to best use them?

Summary: DB software needs



DBMS core design contradicts above goals

Outline

- INTRODUCTION AND OVERVIEW

- **Computer architecture trends and DB workloads**

- Processor/memory speed gap
 - Instruction-level parallelism (ILP)
 - Chip multiprocessors and multithreading



- CPUs, NPUs, and GPUs**

- DBs on CONVENTIONAL PROCESSORS

- QUERY co-PROCESSING: NETWORK PROCESSORS

- QUERY co-PROCESSING: GRAPHICS PROCESSORS

- CONCLUSIONS AND FUTURE DIRECTIONS

Exploiting heterogeneous hardware

- Software: one processor does not fit all
 - heavy reuse vs. sequential scan vs. random access loops
- Opportunities for architectural study
 - On real conventional processors
 - On simulators (hard to find/build, slow)
 - On co-processors
- Query co-processing on NPUs and GPUs
 - A promising approach
 - Indicative of future needs

Outline

● INTRODUCTION AND OVERVIEW

● DBs on CONVENTIONAL PROCESSORS

 Query Processing: Time breakdowns and bottlenecks

- Eliminating unnecessary misses: Data Placement
- Hiding Latencies
- Query processing algorithms and instruction cache misses
- Chip multiprocessor DB architectures

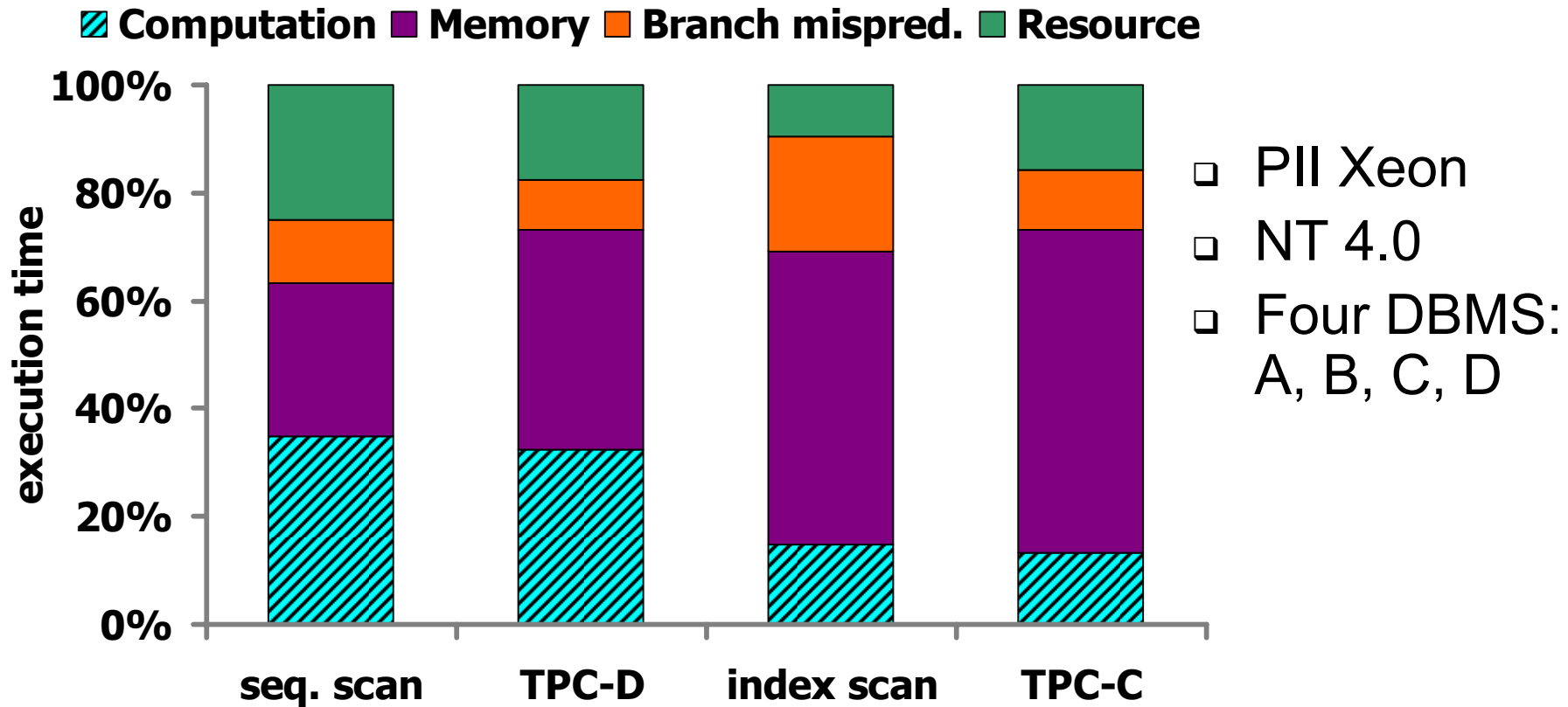
● QUERY co-PROCESSING: NETWORK PROCESSORS

● QUERY co-PROCESSING: GRAPHICS PROCESSORS

● CONCLUSIONS AND FUTURE DIRECTIONS

DB Execution Time Breakdown

[ADH99,BGB98,BGN00,KPH98,SAF04]



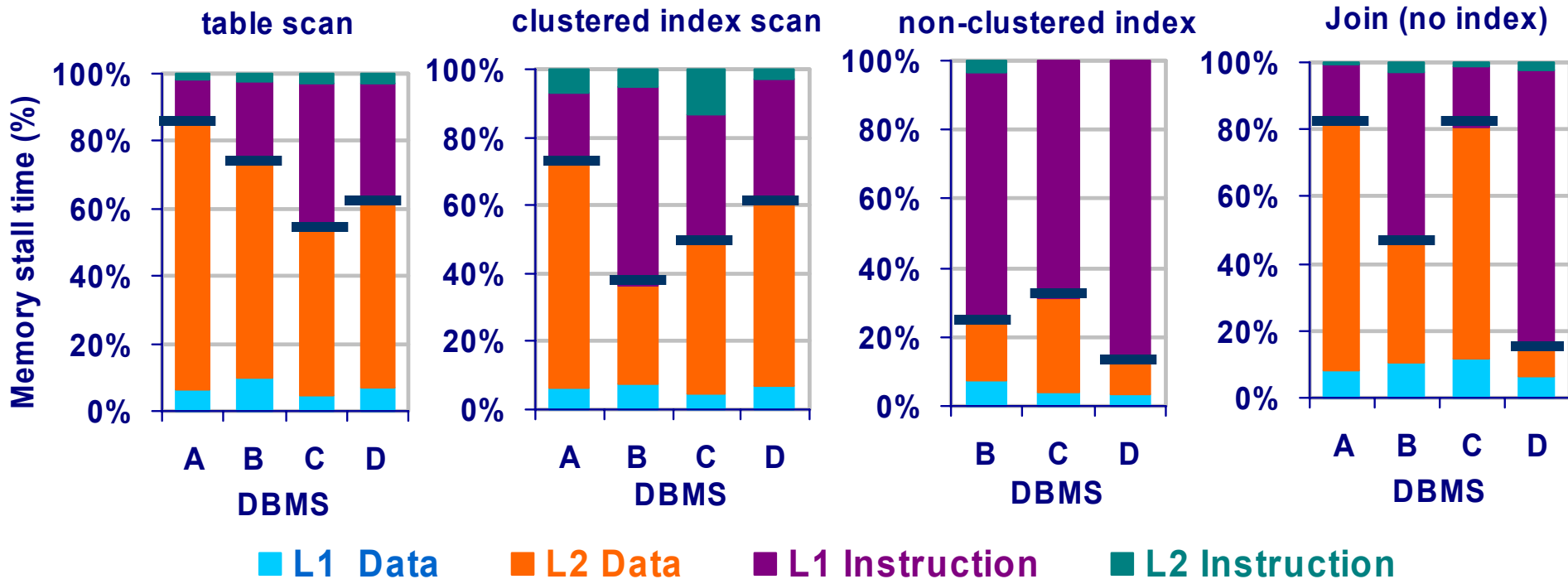
At least 50% cycles on stalls
Memory is major bottleneck
Branch mispredictions increase cache misses!

DSS/OLTP basics: Memory

[ADH99,ADH01]

PII Xeon running NT 4.0, used performance counters

Four commercial Database Systems: A, B, C, D



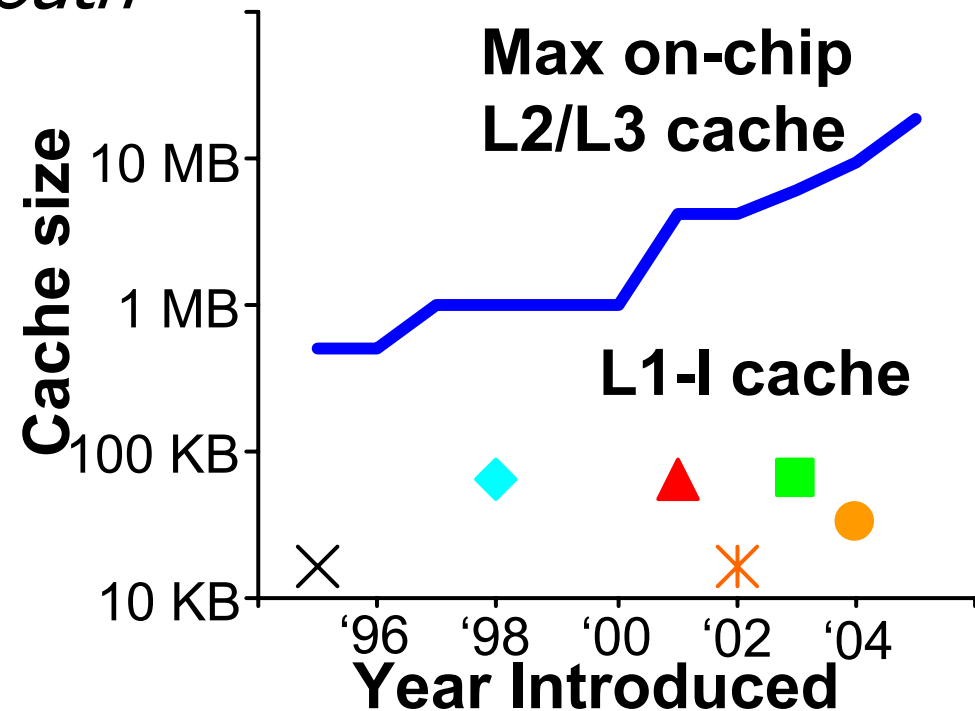
Bottlenecks: data in L2, instructions in L1
Random access (OLTP): L1I-bound

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:



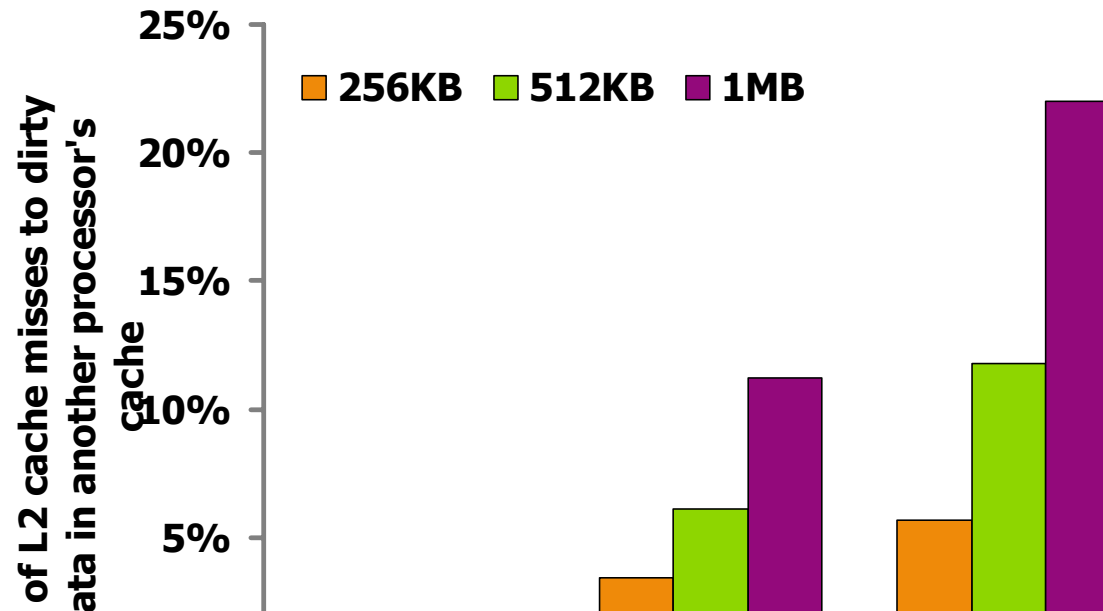
L1I size is stable

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
Hardware needs help from software

Summary: Time breakdowns

- 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)

Goal 1: Eliminate unnecessary misses

Goal 2: Hide latency of “cold” misses

Outline

● INTRODUCTION AND OVERVIEW

● DBs on CONVENTIONAL PROCESSORS

- Query Processing: Time breakdowns and bottlenecks
- Eliminating unnecessary misses: Data Placement
- Hiding Latencies
- Query processing algorithms and instruction cache misses
- Chip multiprocessor DB architectures

● QUERY co-PROCESSING: NETWORK PROCESSORS

● QUERY co-PROCESSING: GRAPHICS PROCESSORS

● CONCLUSIONS AND FUTURE DIRECTIONS

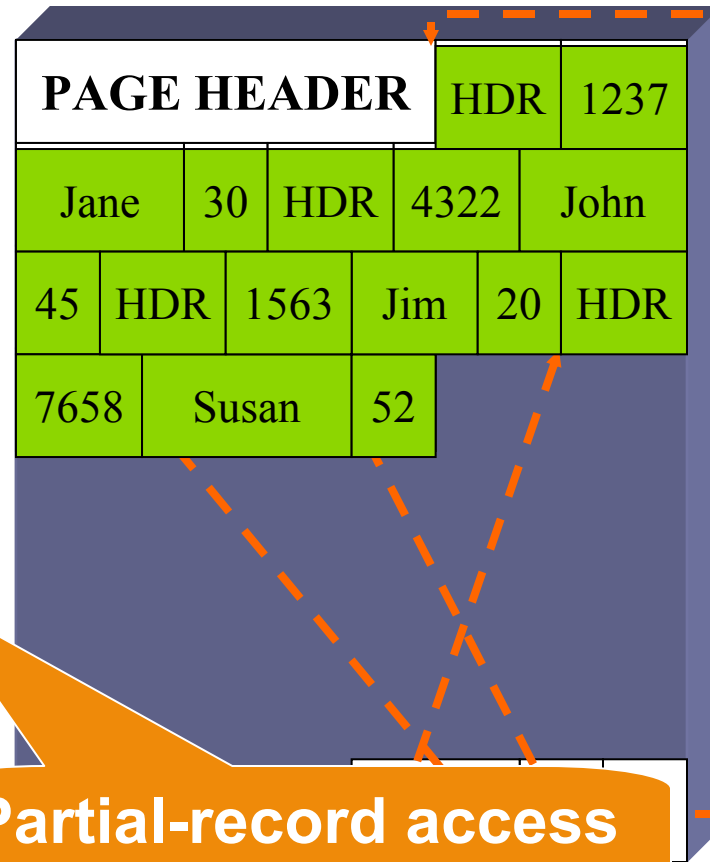
“Classic” Data Layout on Disk Pages

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

R

#	EID	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

Full-record access

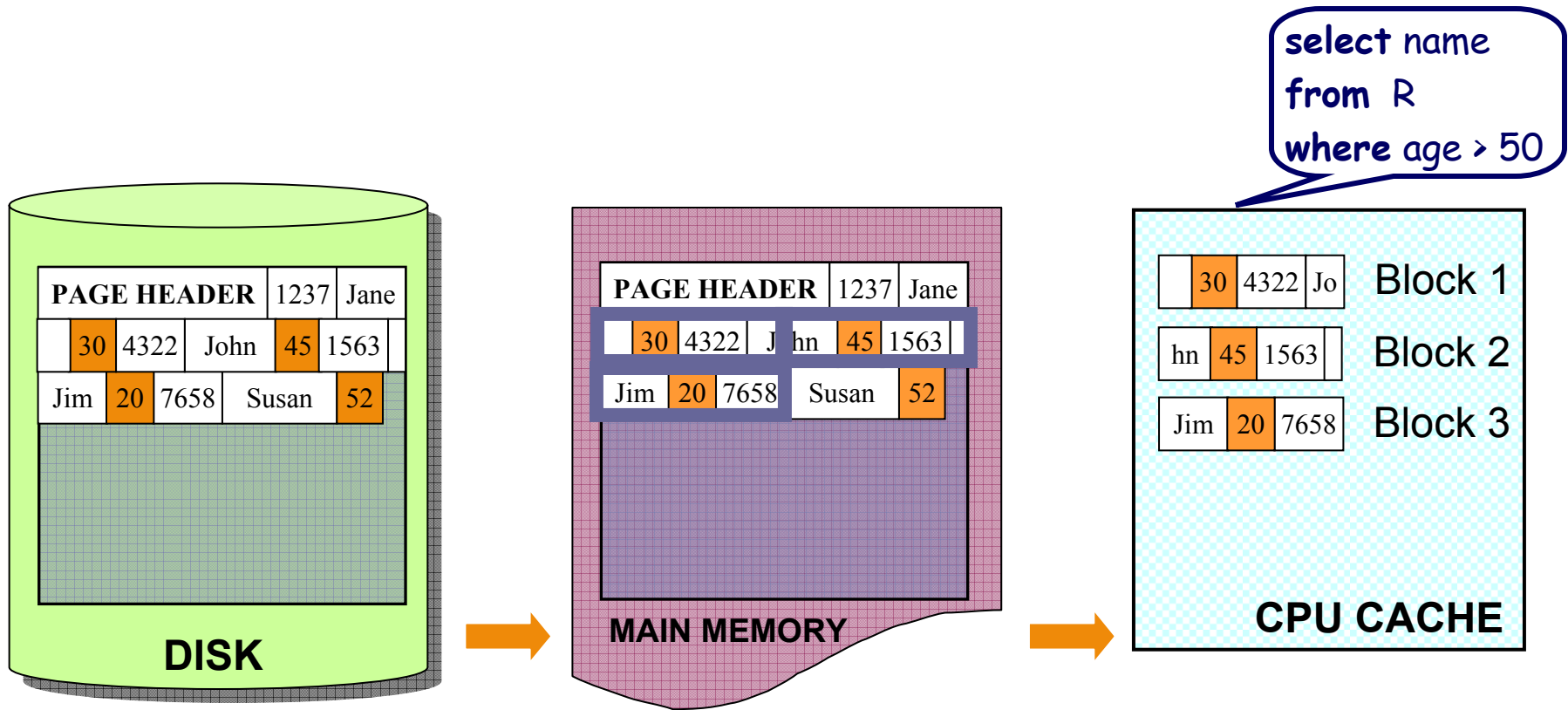


Partial-record access

Records stored sequentially

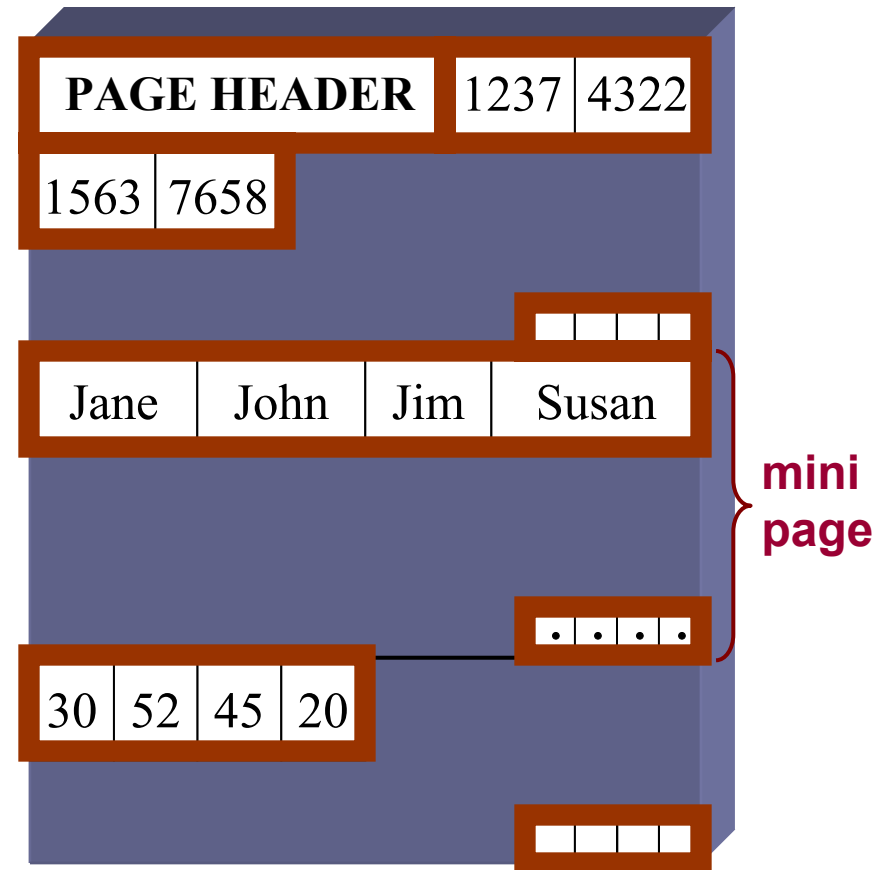
Attributes of a record stored together

NSM in Memory Hierarchy



NSM optimized for full-record access
Hurts partial-record access at all levels

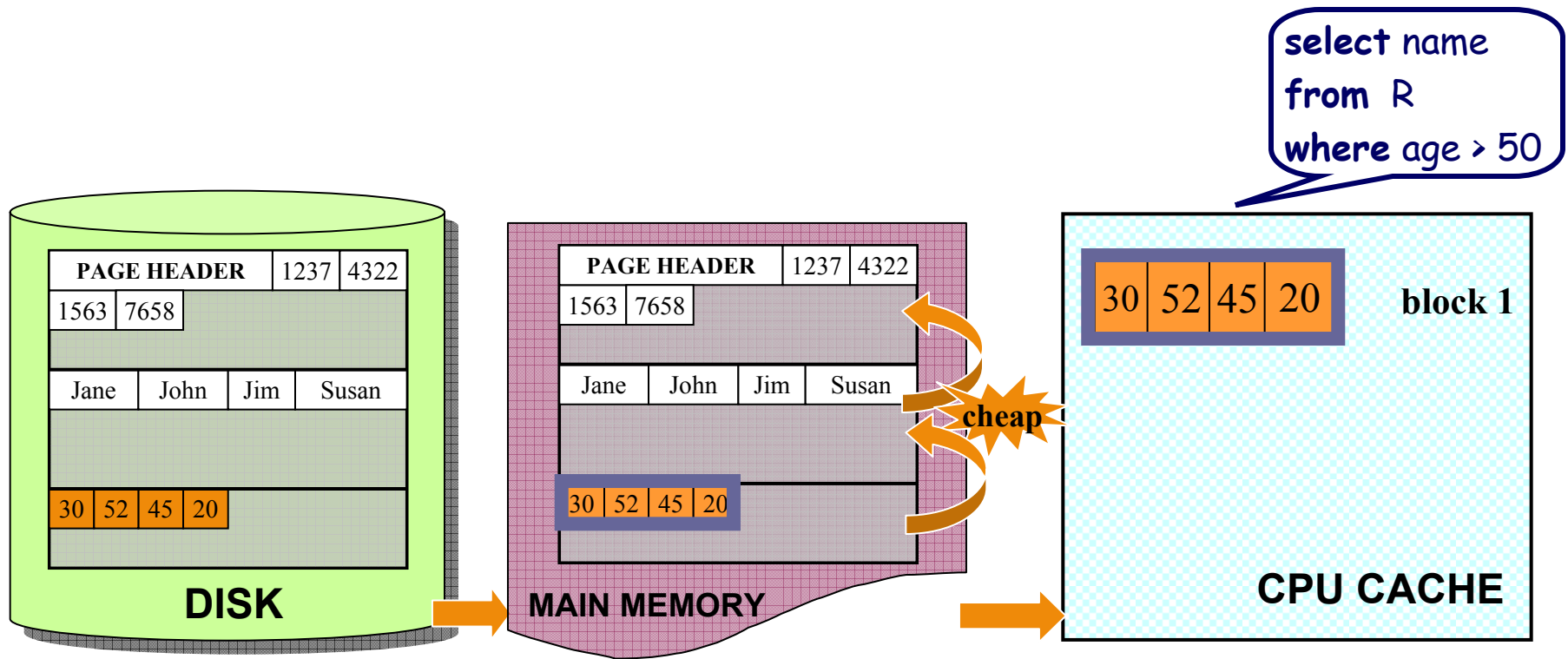
[ADH01]



PAX partitions *within* page: cache locality

PAX in Memory Hierarchy

[ADH01]

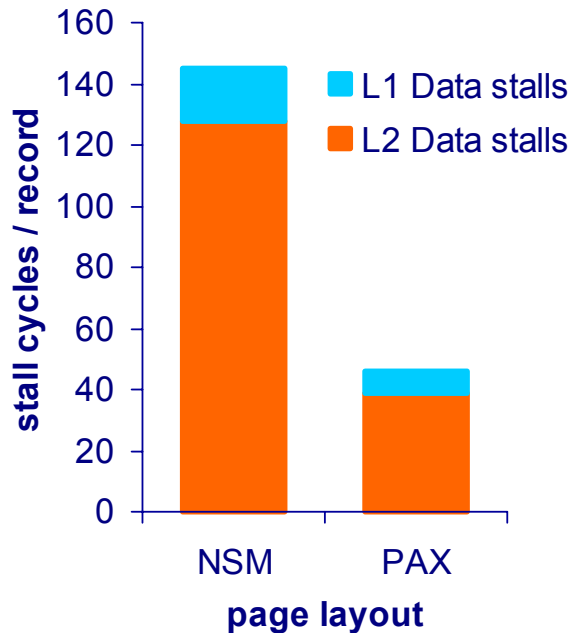


PAX optimizes cache-to-memory communication
Retains NSM's I/O (page contents do not change)

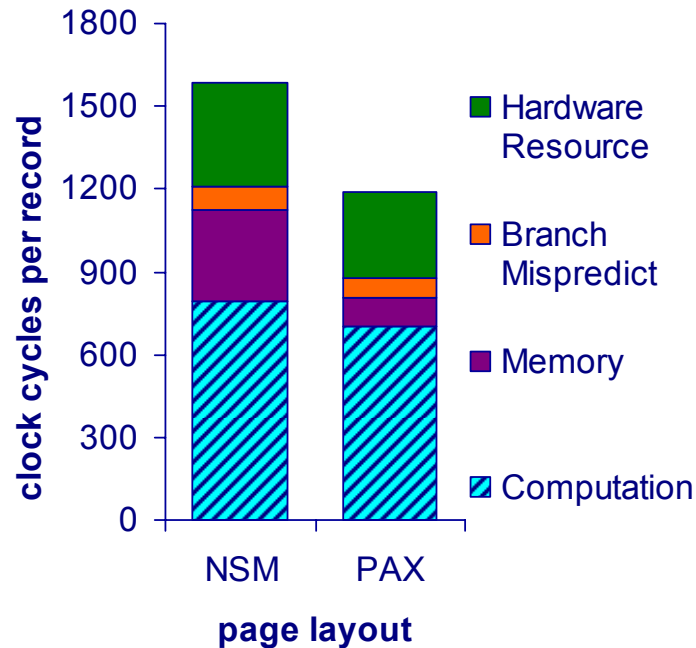
PAX Performance Results (Shore)

[ADH01]

Cache data stalls



Execution time breakdown



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
```

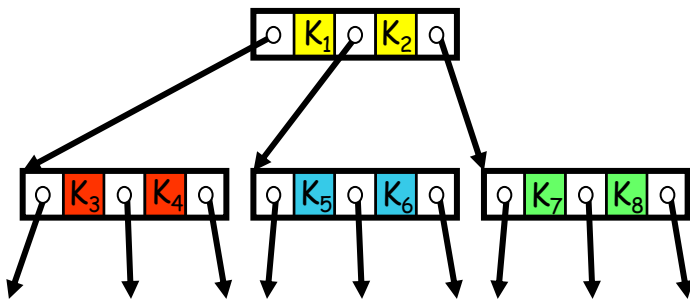
- ❑ 70% less data stall time (only cold misses left)
- ❑ Better use of processor's superscalar capability
- ❑ TPC-H queries: 15%-2x speedup
- ❑ Dynamic PAX: Data Morphing [HP03]
- ❑ CSM custom layout using scatter-gather I/O [SSS04]

B-trees: < Pointers, > Fanout

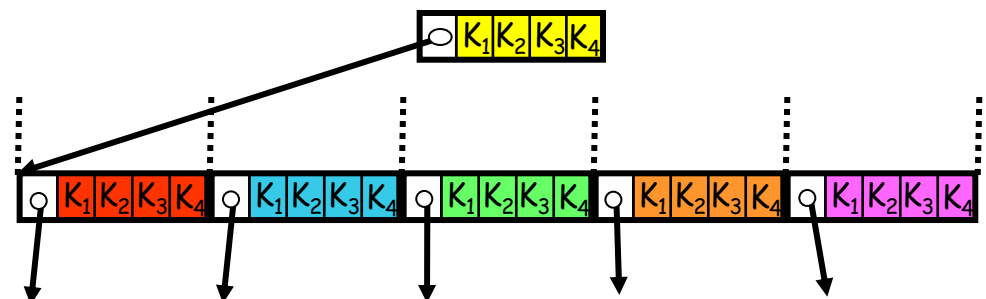
[RR00]

- Cache Sensitive B⁺ Trees (CSB⁺ Trees)
- Layout child nodes contiguously
- Eliminate all but one child pointers
 - *Integer keys double fanout of nonleaf nodes*

B⁺ Trees



CSB⁺ Trees



35% faster tree lookups

Update performance is 30% worse (splits)

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**
- **Fates Automatically-tuned DB Storage Manager**
- **CSB+-trees**
- **Also, Fractured Mirrors: *Cache-and-disk optimization [RDS02] with replication***

Outline

● INTRODUCTION AND OVERVIEW

● DBs on CONVENTIONAL PROCESSORS

- Query Processing: Time breakdowns and bottlenecks
- Eliminating unnecessary misses: Data Placement
- Hiding Latencies
- Query processing algorithms and instruction cache misses
- Chip multiprocessor DB architectures

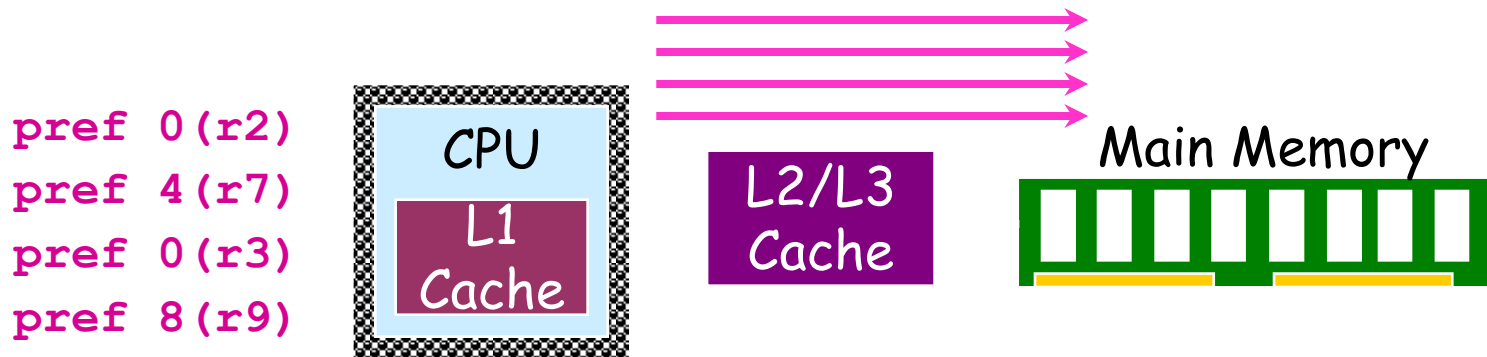
● QUERY co-PROCESSING: NETWORK PROCESSORS

● QUERY co-PROCESSING: GRAPHICS PROCESSORS

● CONCLUSIONS AND FUTURE DIRECTIONS

What about **cold** misses?

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

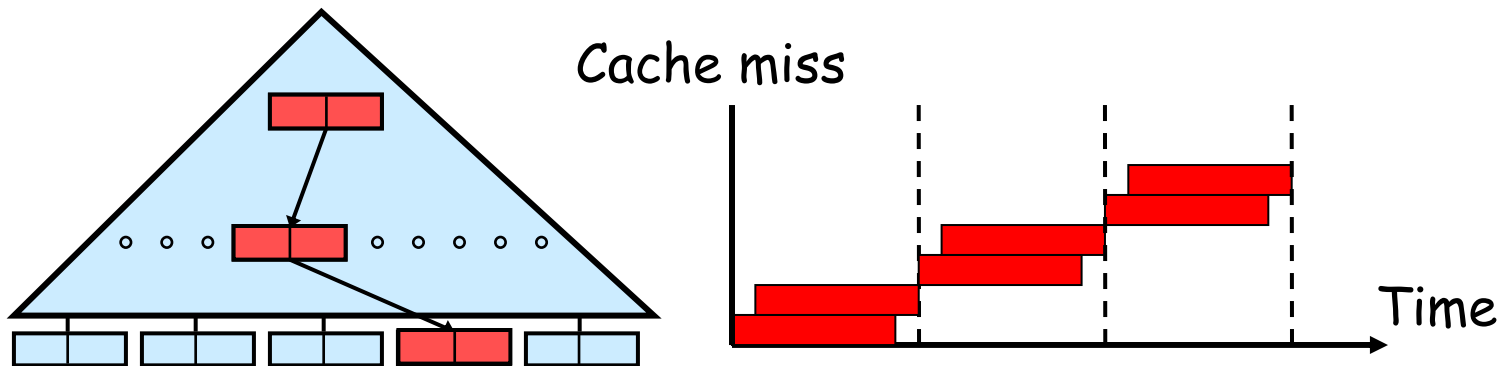


Prefetching hides cold cache miss latency
Efficiently used in pointer-chasing lookups!

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

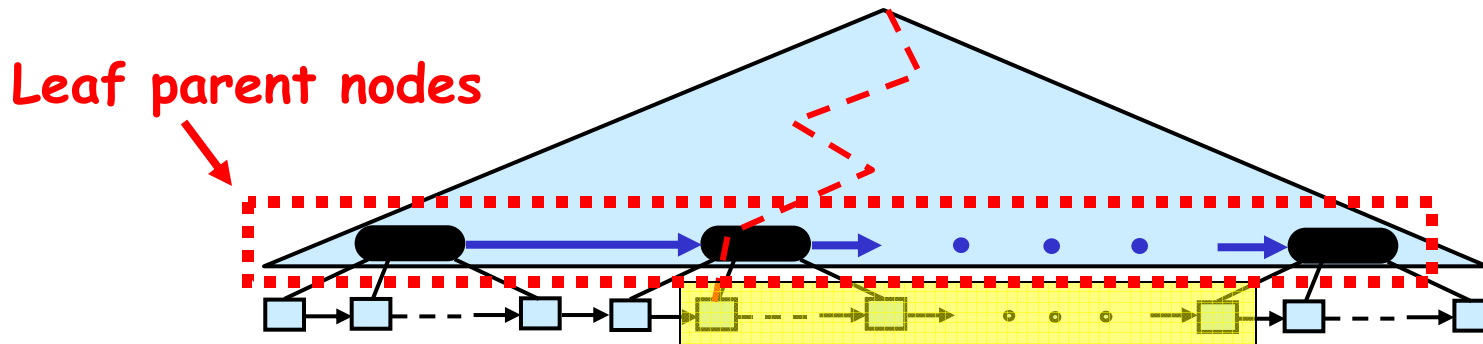


- **Cost to access a node only increases slightly**
- Much shallower trees, no changes required

>2x better search AND update performance
Approach complementary to CSB⁺-trees!

Fractal Prefetching B⁺-trees

[CGM01,CGM02]



- Goal: faster range scan

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

- Fractal: Embed cache-aware trees in disk nodes

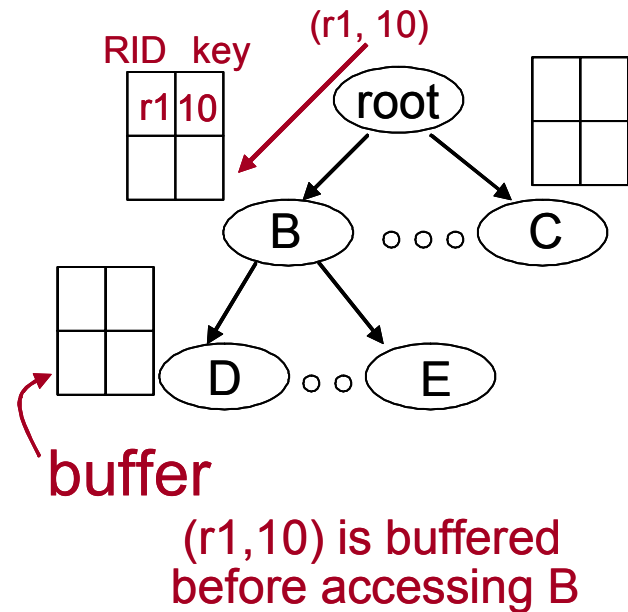
pB⁺-trees: 8X speedup over B⁺-trees

Fractal pB⁺-trees: 80% faster in-mem search

Bulk lookups

[ZR03a]

- Optimize data cache performance
 - Like computation regrouping
- 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

Hiding latencies: 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:

- CR-tree: Cache-conscious R-tree [KCK01]
 - Compresses MBR keys
- Cache-oblivious B-Trees [BDF00]
 - Optimal bound in number of memory transfers
 - Regardless of # of memory levels, block size, or level speed
- Survey of 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

● DBs on CONVENTIONAL PROCESSORS

- Query Processing: Time breakdowns and bottlenecks
- Eliminating unnecessary misses: Data Placement
- Hiding Latencies

 Query processing algorithms and instruction cache misses

- Chip multiprocessor DB architectures

● QUERY co-PROCESSING: NETWORK PROCESSORS

● QUERY co-PROCESSING: GRAPHICS PROCESSORS

● CONCLUSIONS AND FUTURE DIRECTIONS

Query Processing Algorithms

Adapt query processing algorithms to caches

Related work includes:

- Improving data cache performance
 - Sorting and hash-join
- Improving instruction cache performance
 - DSS and OLTP applications

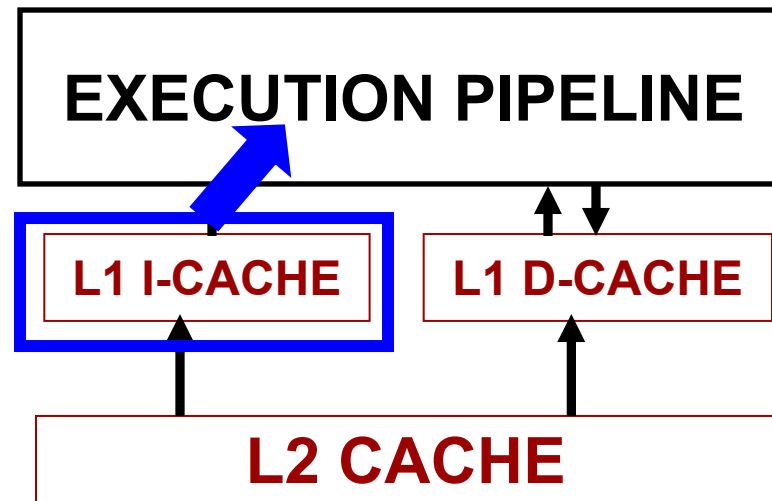
Query processing: directions

[see references]

- Alphasort: quicksort and key prefix-pointer [NBC94]
- Monet: MM-DBMS uses aggressive DSM [MBN04]++
 - Optimize partitioning with hierarchical radix-clustering
 - Optimize post-projection with radix-declustering
 - Many other optimizations
- Hash joins: aggressive prefetching [CAG04]++
 - Efficiently hides data cache misses
 - Robust performance with future long latencies
- Inspector Joins [CAG05]
- DSS I-misses: new “group” operator [ZR04]
- B-tree concurrency control: reduce readers’ latching [CHK01]

Instruction-Related Stalls

- 25-40% of OLTP execution time [KPH98, HA04]
- Importance of instruction cache: In critical path!



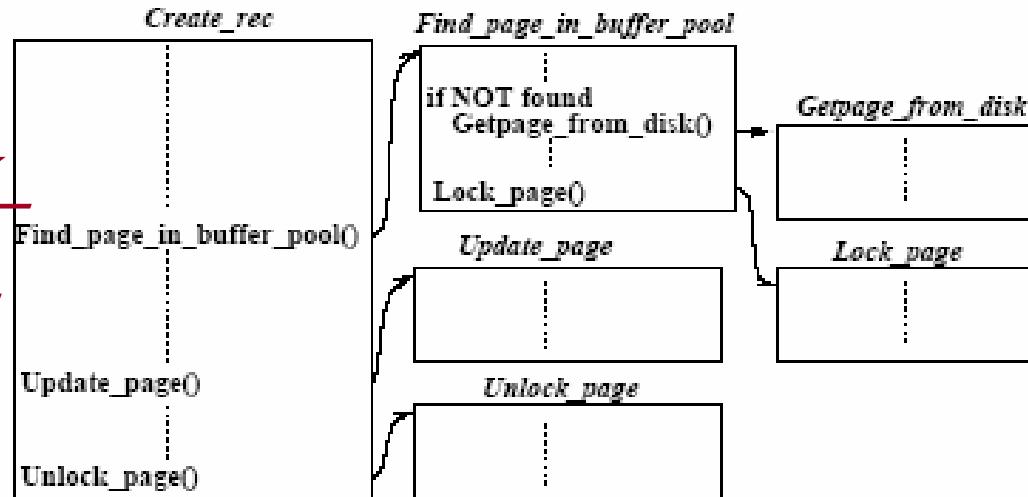
Impossible to overlap I-cache delays

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



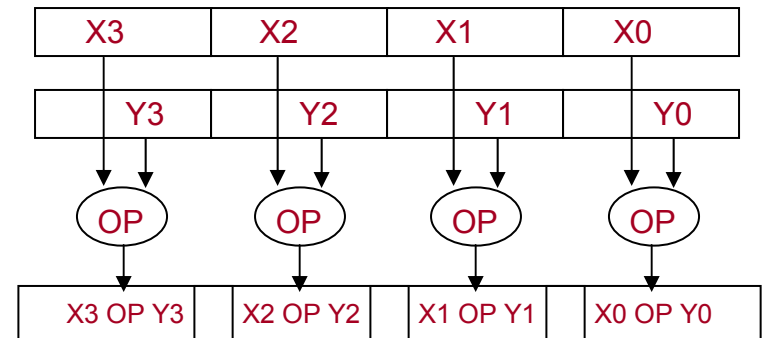
- Experiments: Shored on SimpleScalar Simulator
 - Running Wisconsin Benchmark

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

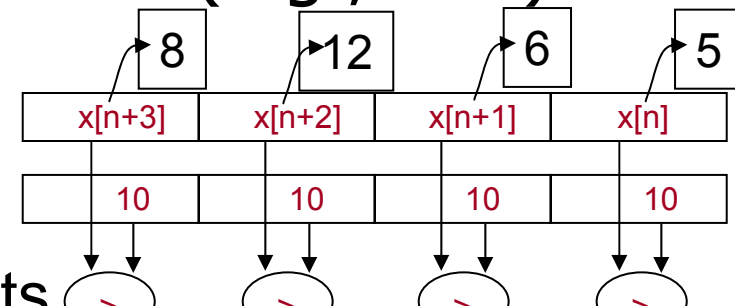


- Assume data stored columnwise as contiguous array of fixed-length numeric values (e.g., PAX)

- *Scan example:*

if $x[n] > 10$
produce bitmap
vector with 4
comparison results

$result[pos++] = x[n]$

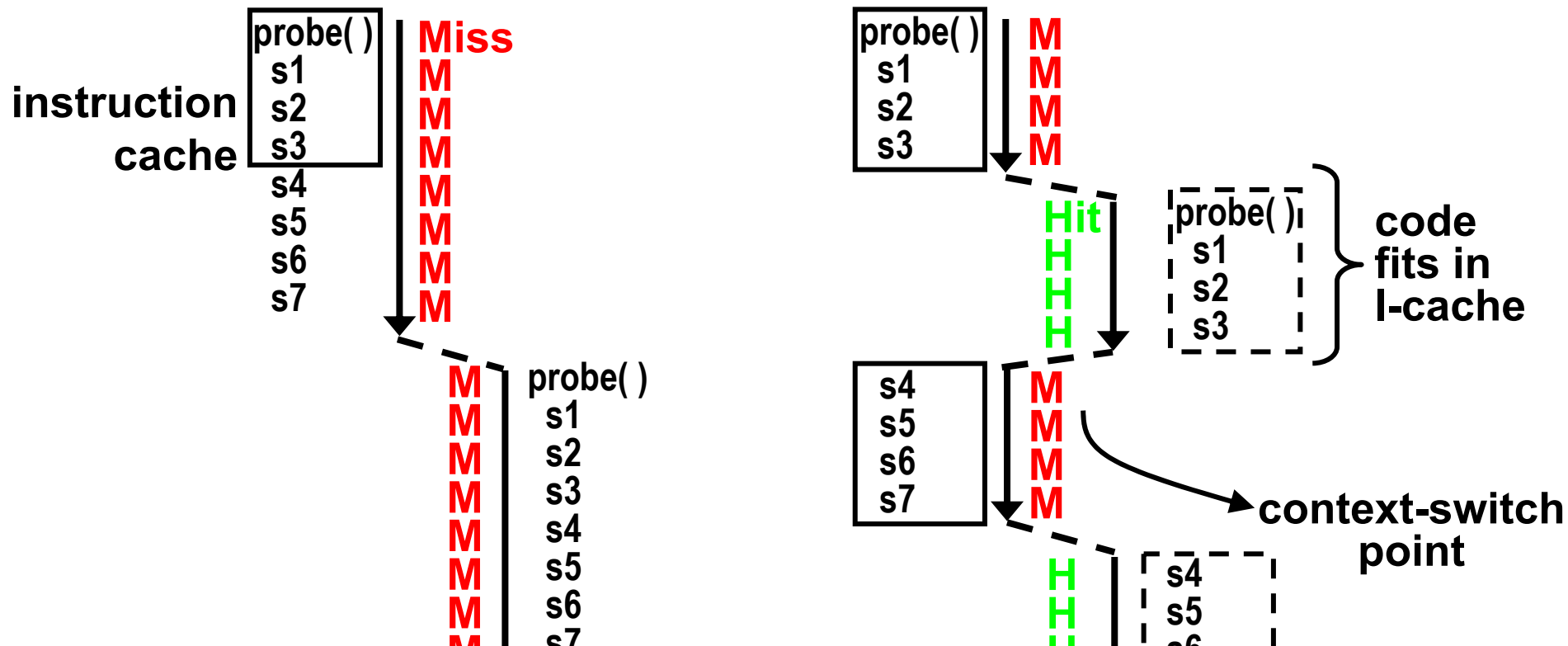


Superlinear speedup to # of parallelism
Need to rewrite code to use SIMD

STEPS: Cache-Resident OLTP

[HA04]

Synchronized **T**ransactions
through **E**xplicit **P**rocessor **S**cheduling



- **Index probe: 96% fewer L1-I misses**
- **TPC-C: we eliminate 2/3 of misses, 1.4 speedup**

Summary: Memory affinity

- 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 or grouping
 - Techniques for B-trees, hash joins
- Query processing algorithms
 - For sorting and hash-joins, addressing data and instruction stalls
- Low-level instruction cache optimizations
 - DSS: Call Graph Prefetching, SIMD
 - OLTP: STEPS (explicit transaction scheduling)

Outline

● INTRODUCTION AND OVERVIEW

● DBs on CONVENTIONAL PROCESSORS

- Query Processing: Time breakdowns and bottlenecks
- Eliminating unnecessary misses: Data Placement
- Hiding Latencies
- Query processing algorithms and instruction cache misses

→ Chip multiprocessor DB architectures

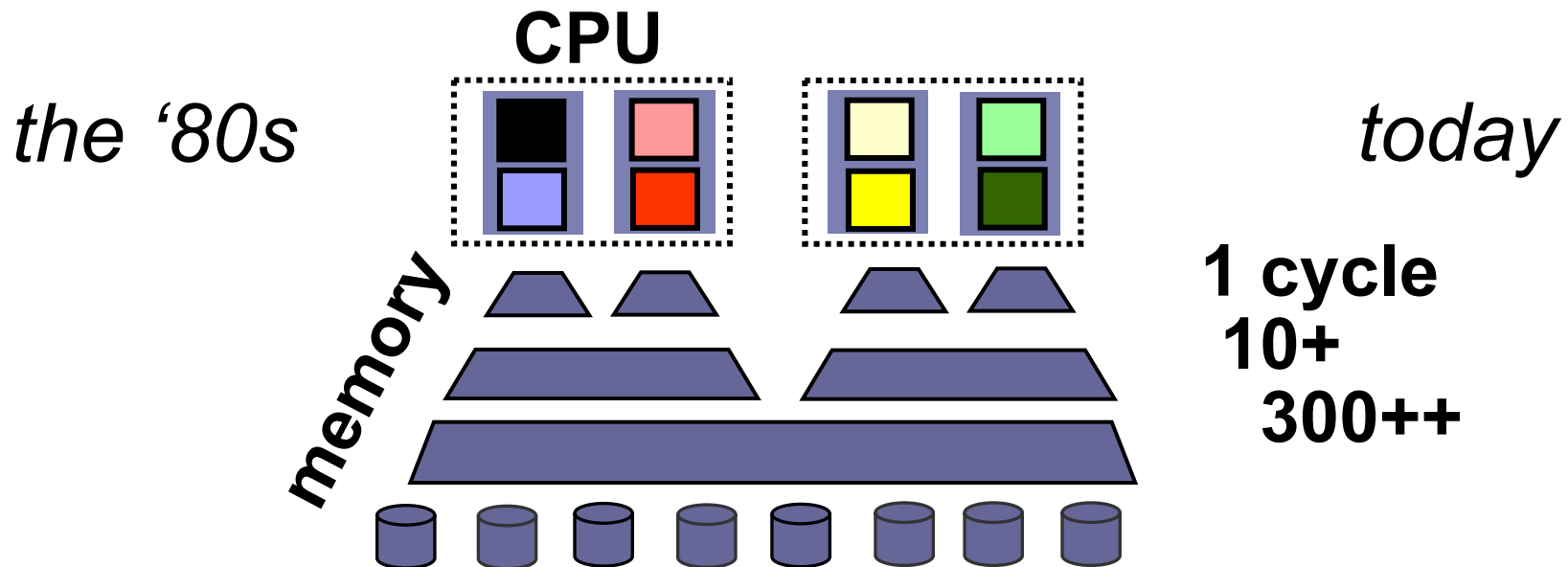
● QUERY co-PROCESSING: NETWORK PROCESSORS

● QUERY co-PROCESSING: GRAPHICS PROCESSORS

● CONCLUSIONS AND FUTURE DIRECTIONS

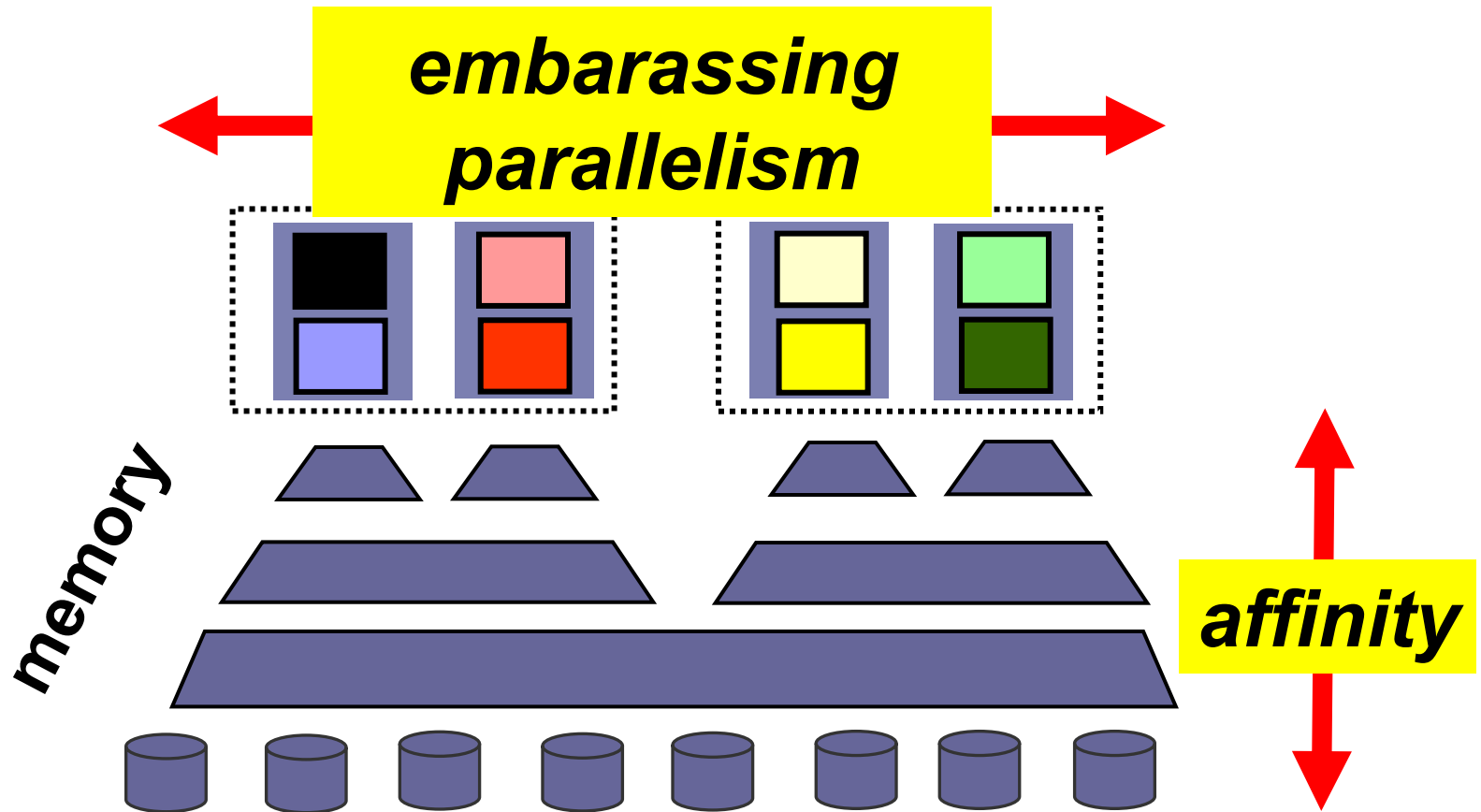
Evolution of hardware design

Past: HW = CPU+Memory+Disk



CPUs run faster than they access data

CMP, SMT, and memory



DBMS core design contradicts above goals

How SMTs can help DB Performance

[ZCR05]

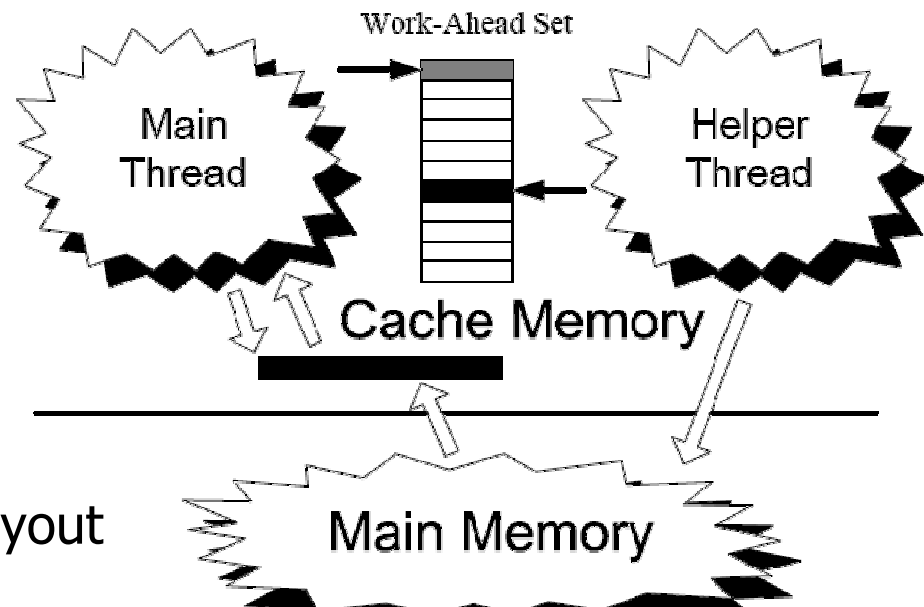
- Naïve parallel: treat SMT as multiprocessor
- Bi-threaded: partition input, cooperative threads
- Work-ahead-set: main thread + helper thread:
 - **Main thread posts “work-ahead set” to a queue**
 - **Helper thread issues load instructions for the requests**

● Experiments

- index operations and hash joins
- Pentium 4 with HT
- Memory-resident data

● Conclusions

- Bi-threaded: high throughput
- Work-ahead-set: high for row layout



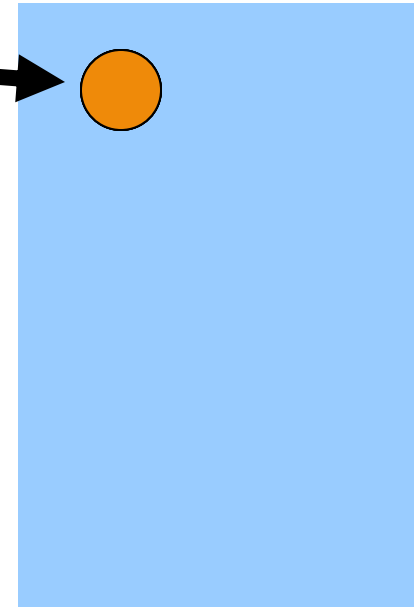
Work-ahead-set best with no data movement

Parallelizing transactions

[CAS05,CAS06]

DBMS

```
SELECT cust_info FROM customer;  
UPDATE district WITH order_id;  
INSERT order_id INTO new_order;  
foreach(item) {  
    GET quantity FROM stock;  
    quantity--;  
    UPDATE stock WITH quantity;  
    INSERT item INTO order_line;  
}
```



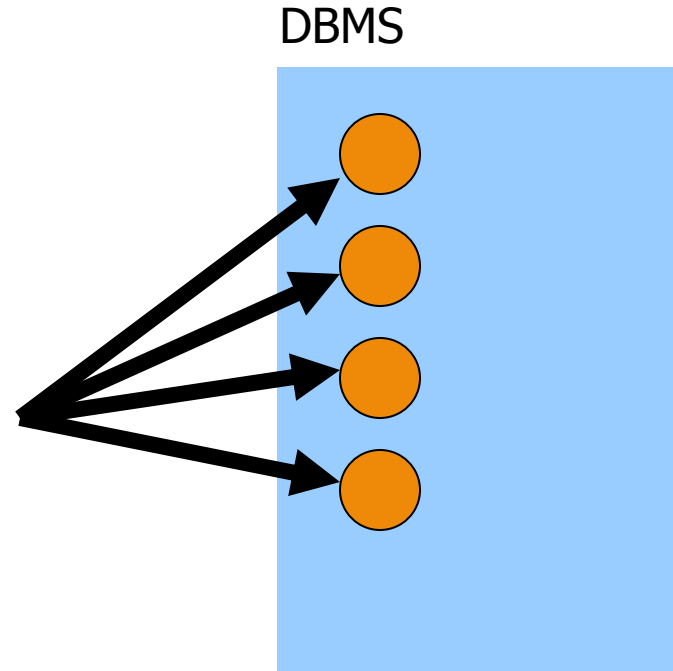
● Intra-query parallelism

- Used for long-running queries (decision support)
- Does not work for short queries

● Short queries dominate in OLTP workloads

Parallelizing transactions

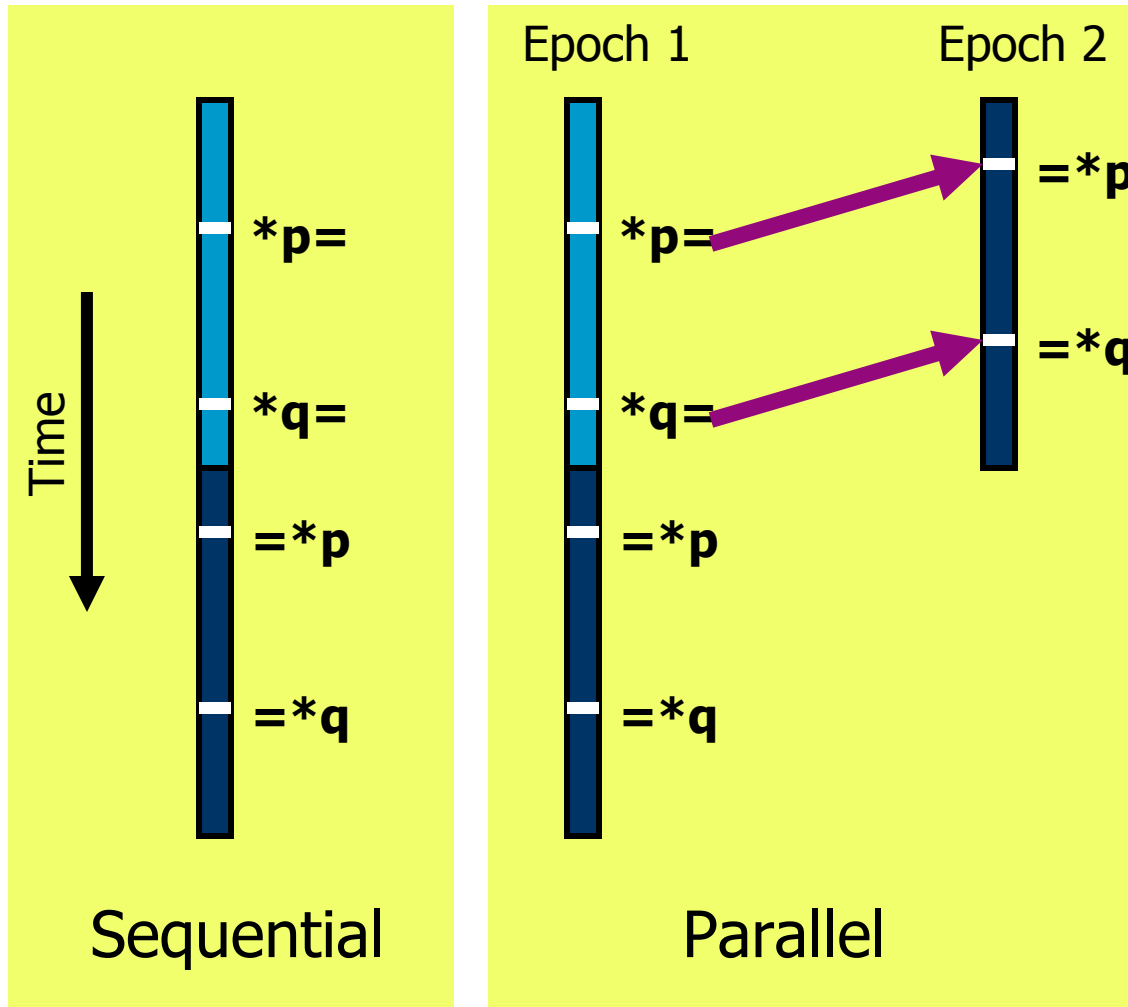
```
SELECT cust_info FROM customer;  
UPDATE district WITH order_id;  
INSERT order_id INTO new_order;  
foreach(item) {  
    GET quantity FROM stock;  
    quantity--;  
    UPDATE stock WITH quantity;  
    INSERT item INTO order_line;  
}
```



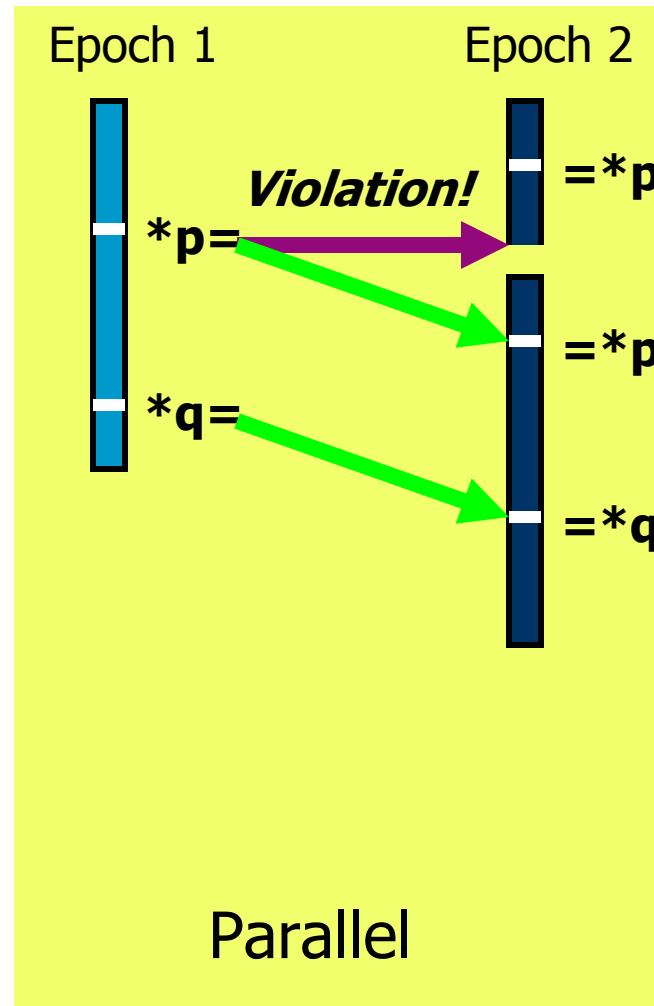
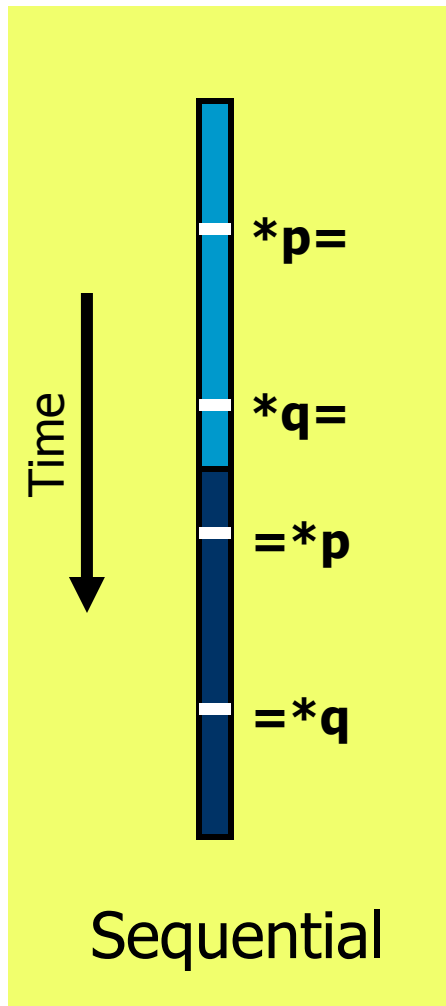
- Intra-transaction parallelism
 - Each thread spans multiple queries
- Hard to add to existing systems!

**Thread Level Speculation (TLS)
makes parallelization easier.**

Thread Level Speculation (TLS)



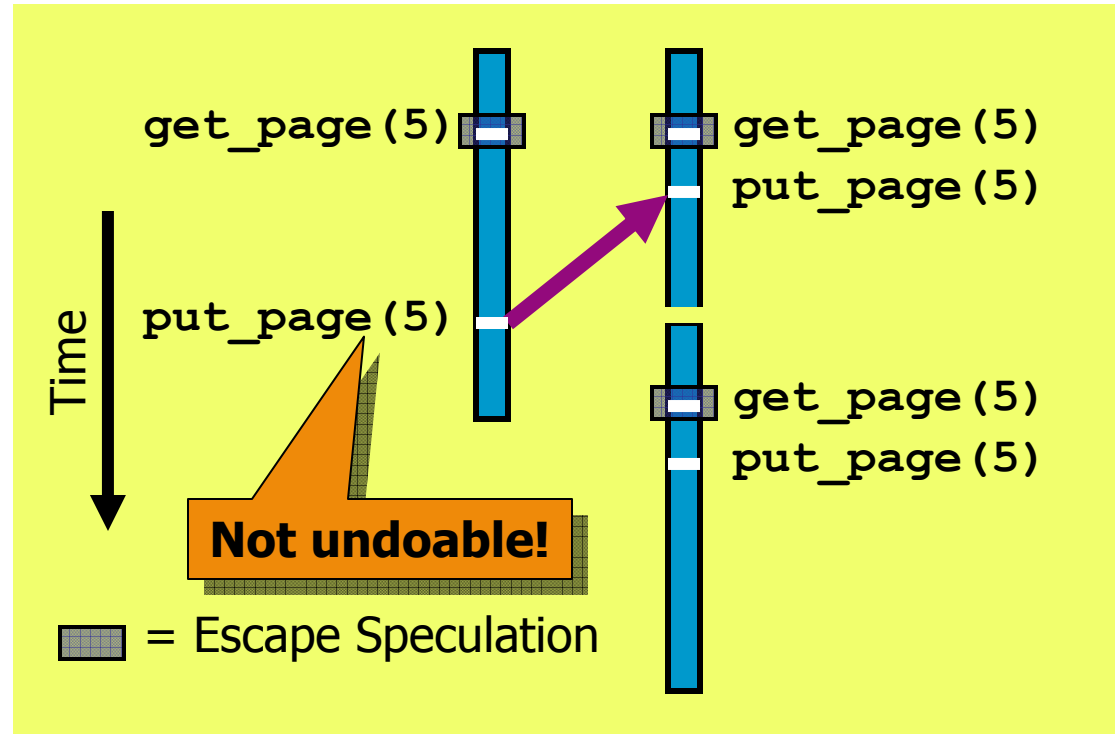
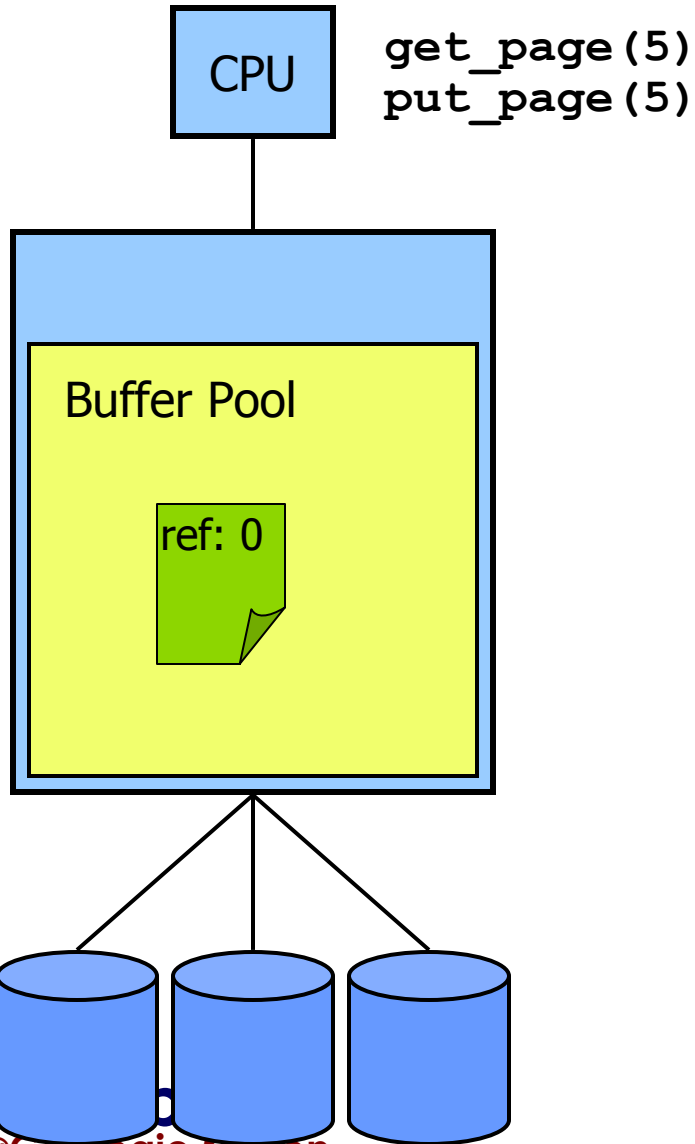
Thread Level Speculation (TLS)



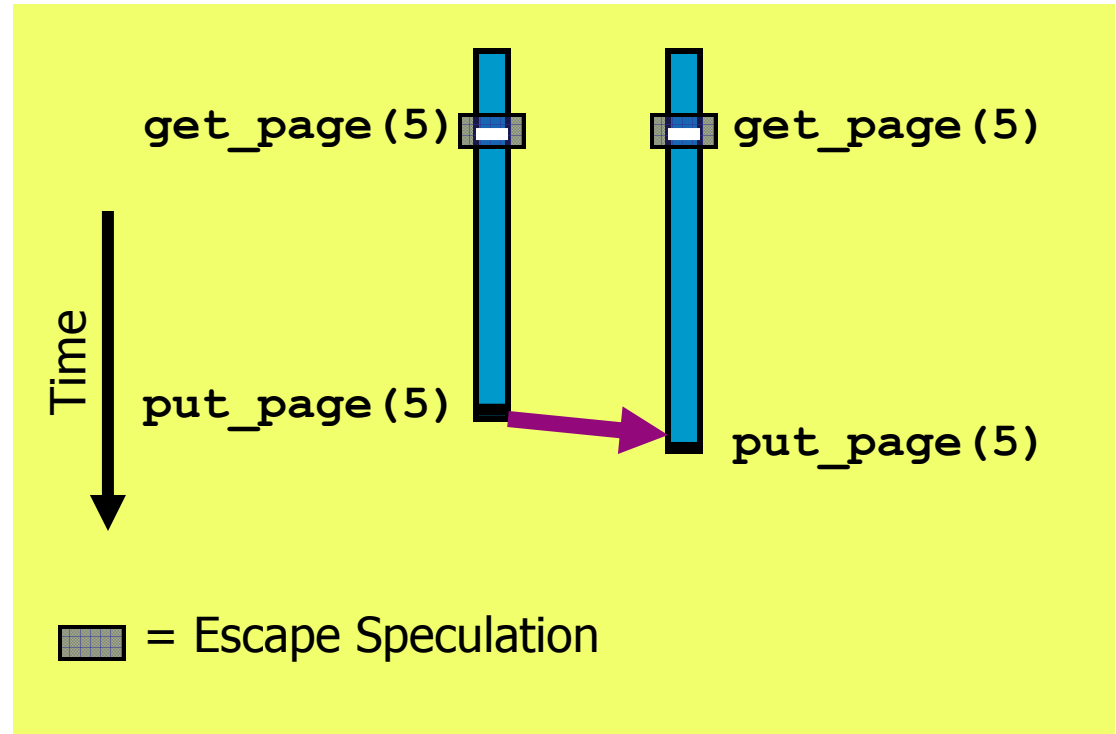
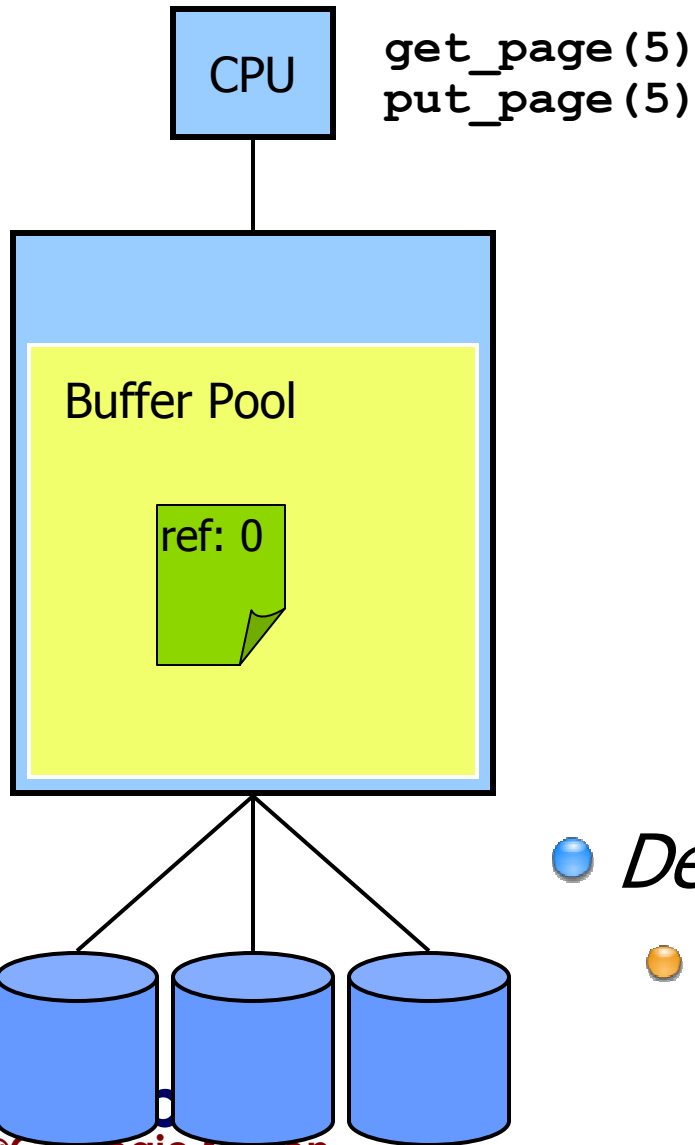
- Use *epochs*
- Detect violations
- Restart to recover
- Buffer state
- Worst case:
 - Sequential
- Best case:
 - Fully parallel

Data dependences limit performance

Example: Buffer Pool Management



Example: Buffer Pool Management



- *Delay* `put_page` until end of epoch
- *Avoid dependence*

Removing Bottleneck Dependences

Introducing three techniques:

- **Delay operations** until non-speculative

- Mutex and lock *acquire* and *release*
- Buffer pool, memory, and cursor *release*
- Log sequence number assignment

- **Escape speculation**

- Buffer pool, memory, and cursor *allocation*

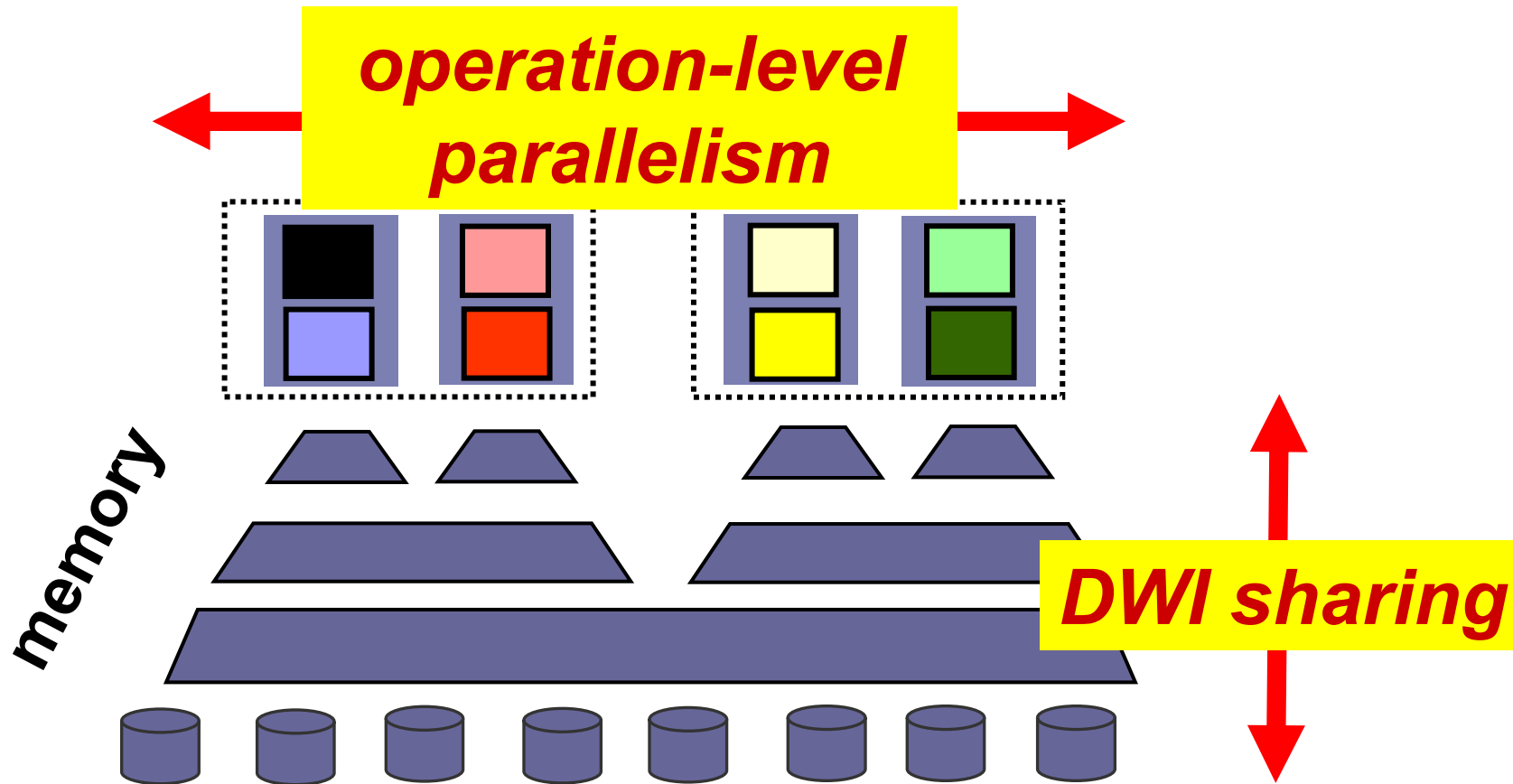
- **Traditional parallelization**

- Memory allocation, cursor pool, error checks, false sharing

2x lower latency with 4 CPUs

Useful for non-TLS parallelism as well

DBMS parallelism and memory affinity



StagedDB design addresses shortcomings

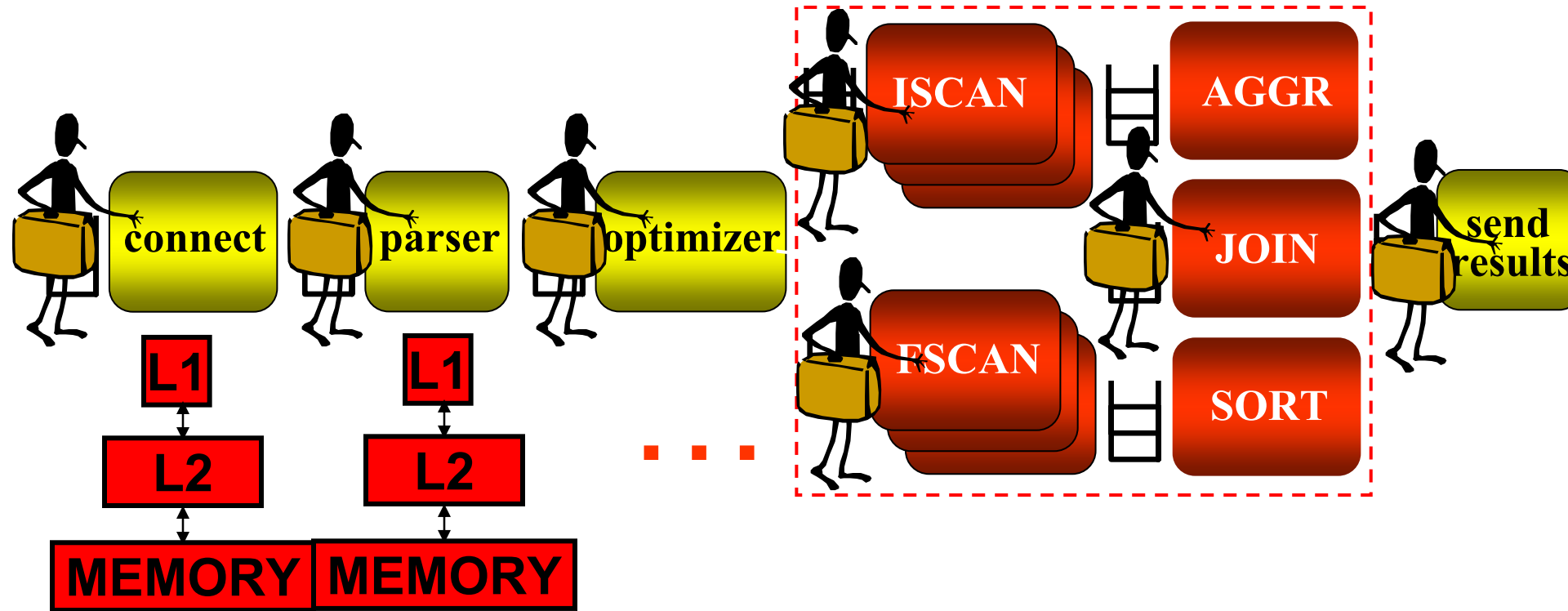
StagedDB software design

[HA03]

- Cohort query scheduling: amortize loading time
- Suspend at module boundaries: maintain context
- Break DBMS into stages
- Stages act as independent servers
- Queries pick services they need
- Proposed query scheduling algorithms to address locality/wait time tradeoffs [HA02]

Prototype design

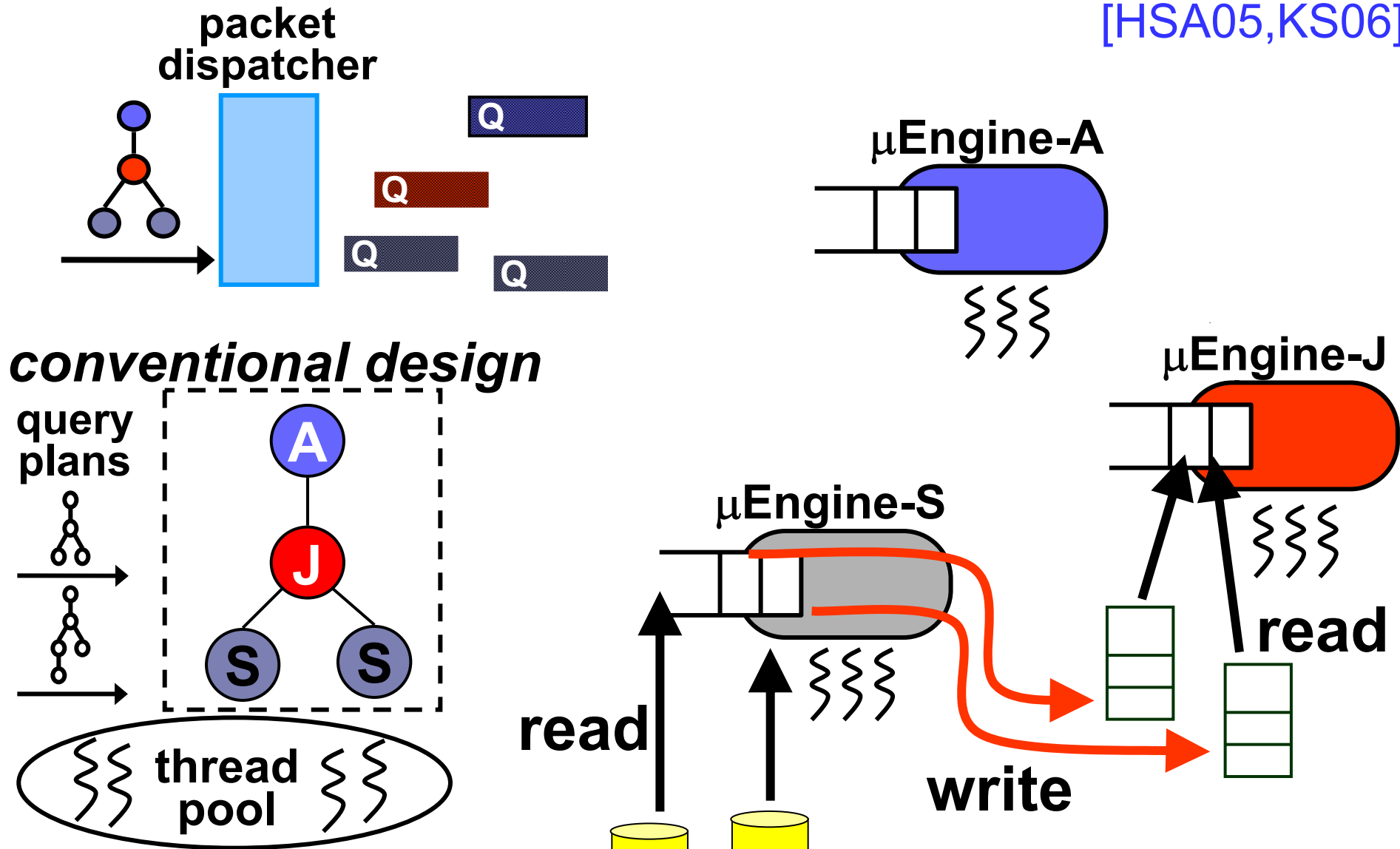
[HA03,HSA05]



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

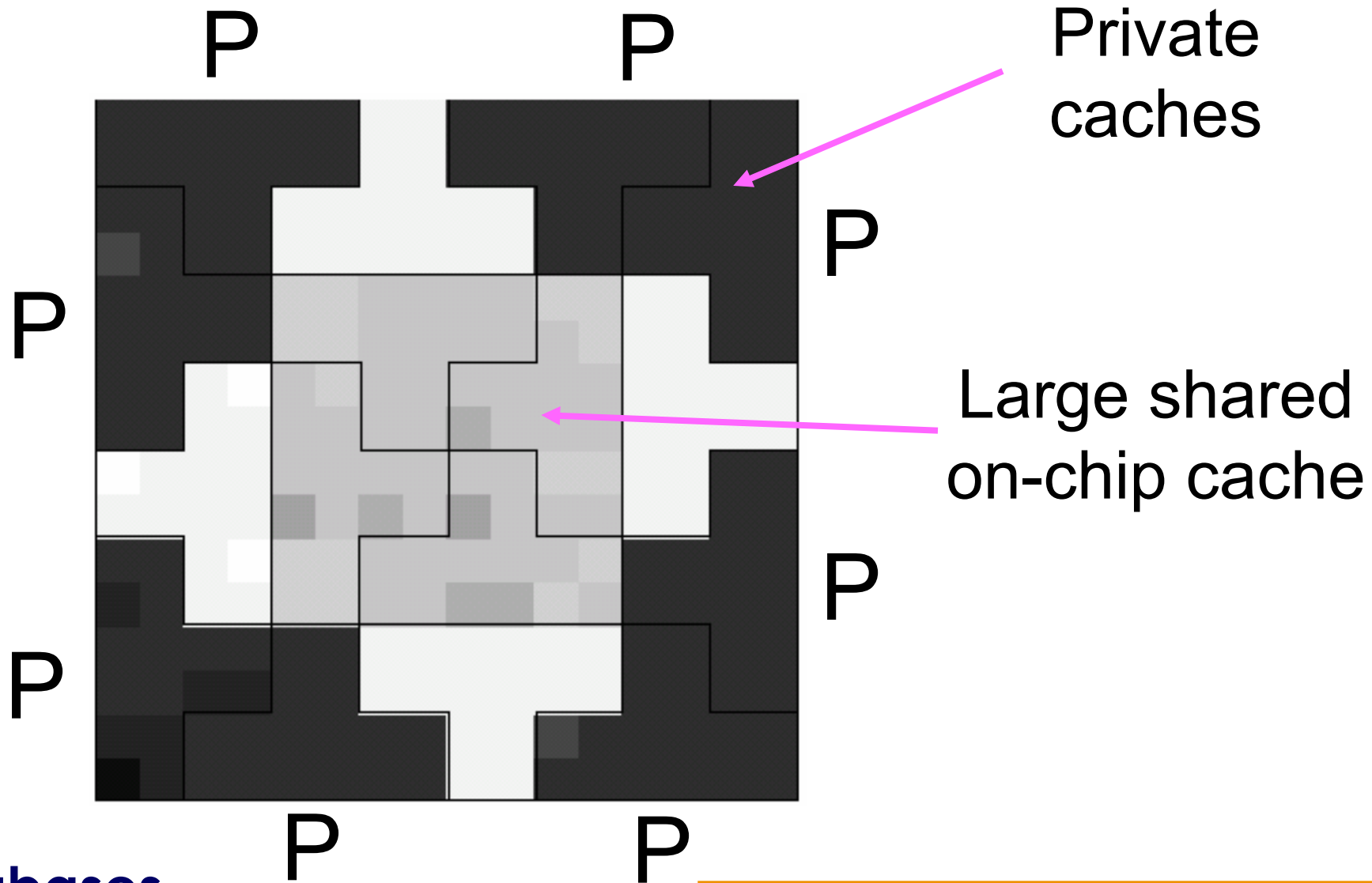
QPipe: operation-level parallelism

[HSA05,KS06]



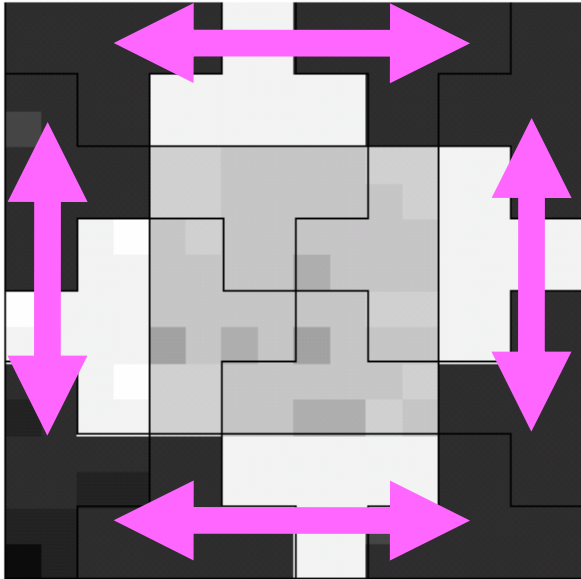
Stable throughput as #users increases

Future: NUCA hierarchy abstraction

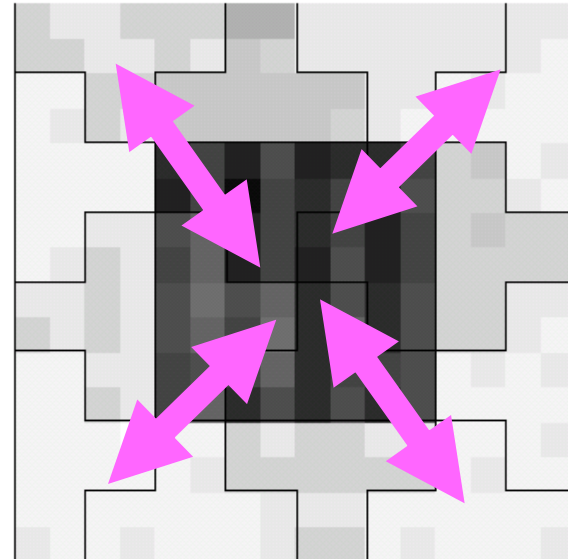


Data movement on CMP hierarchy*

Stencil computation



OLTP



- Traditional DBMS: shared information in middle
- StagedDB: exposed data movement

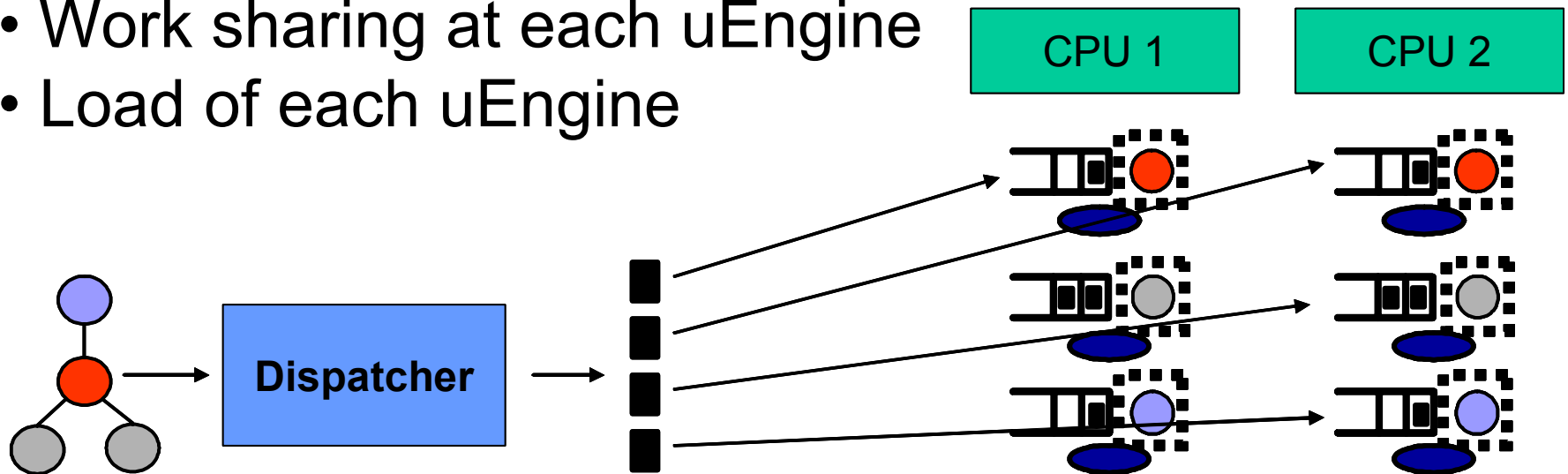
**data from Beckmann&Wood, Micro04*

StagedCMP: StagedDB on Multicore

- μ Engines run independently on cores
- Dispatcher routes incoming tasks to cores

Tradeoff:

- Work sharing at each uEngine
- Load of each uEngine



Potential: better work sharing, load balance on CMP

Summary: data mgmt on SMT/CMP

- Work-ahead sets using helper threads
 - Use threads for prefetching
 - Predecessor to lifeguards: a generalized notion of a helper thread
- Intra-transaction parallelism using TLS
 - Thread-level speculation necessary for transactional memories
 - Techniques proposed applicable on today's hardware too
- Staged Database System Architectures
 - Addressing both memory affinity and unlimited parallelism
 - Opportunity for data movement prediction amongst procesors

Outline

- INTRODUCTION AND OVERVIEW
- DBs on CONVENTIONAL PROCESSORS
- **QUERY co-PROCESSING: NETWORK PROCESSORS**
 - TLP and network processors
 - Programming model
 - Methodology & Results
- QUERY co-PROCESSING: GRAPHICS PROCESSORS
- CONCLUSIONS AND FUTURE DIRECTIONS

Modern Architectures & DBMS

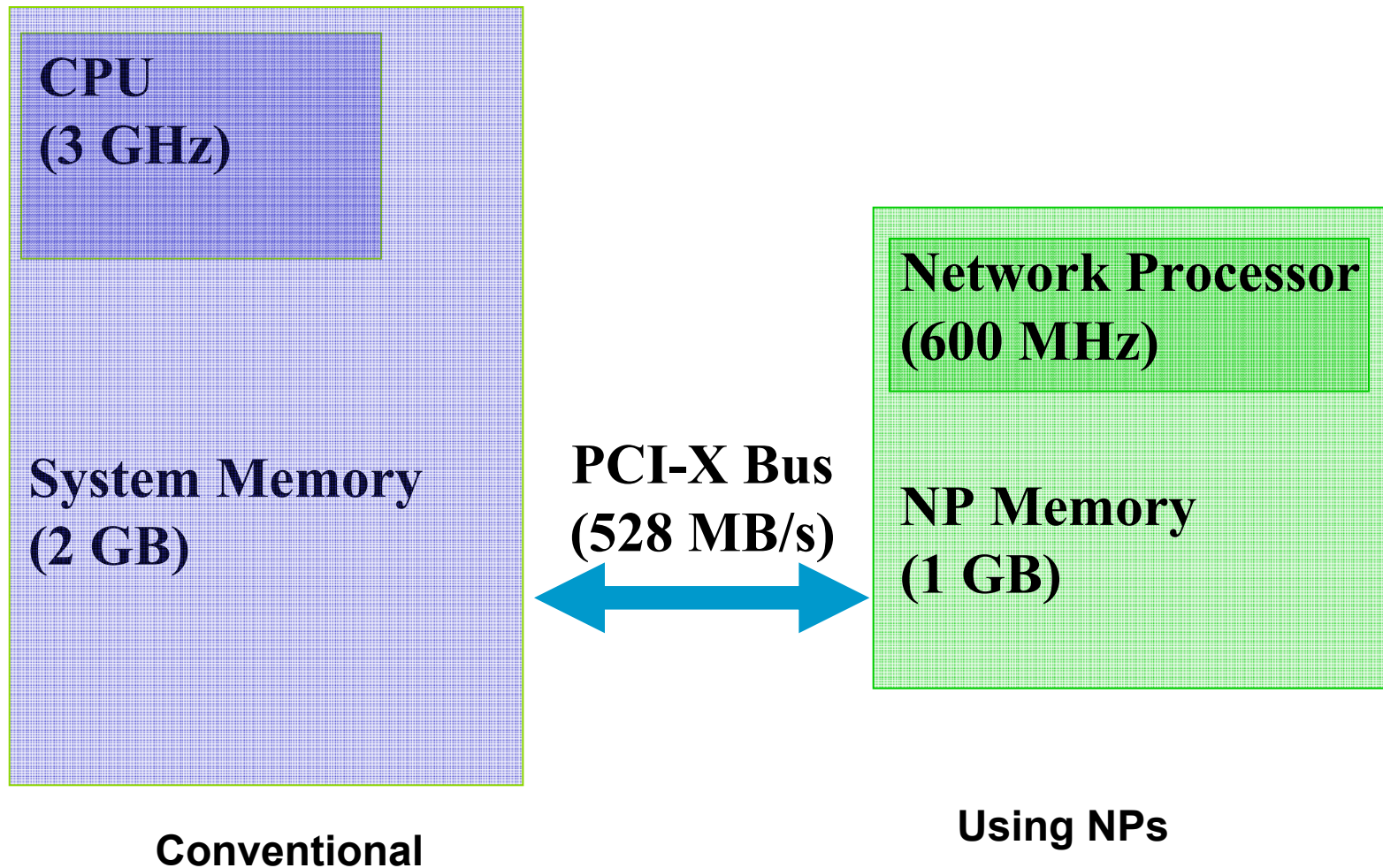
- Instruction-Level Parallelism (ILP)
 - Out-of-order (OoO) execution window
 - Cache hierarchies - spatial / temporal locality
- DBMS' memory system characteristics
 - Limited locality (e.g., sequential scan)
 - Random access patterns (e.g., hash join)
 - Pointer-chasing (e.g., index scan, hash join)
- DBMS needs Memory-Level Parallelism (MLP)

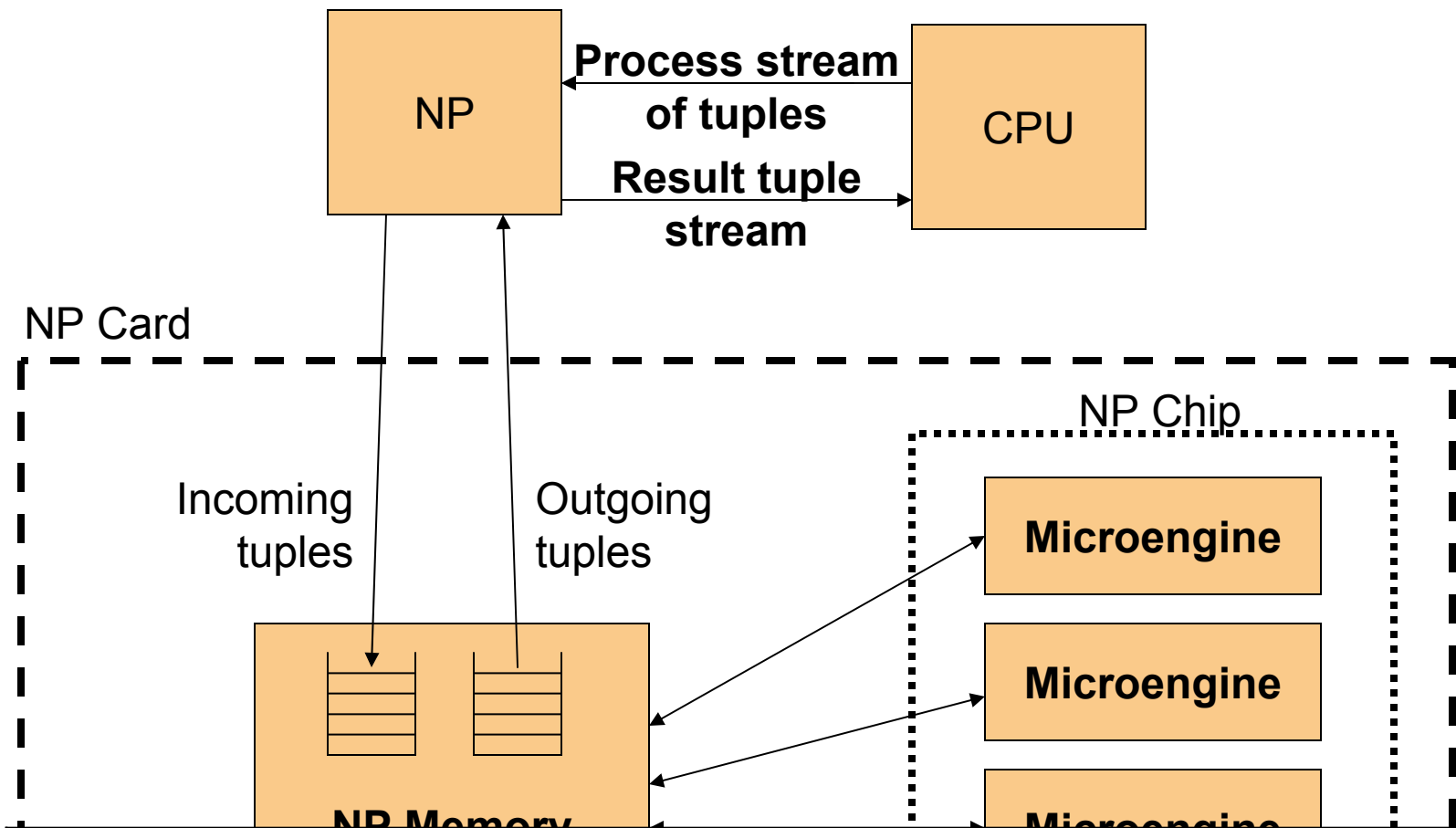
Opportunities on NPUs

- DB operators on thread-parallel architectures
 - Expose parallel misses to memory
 - Leverage intra-operator parallelism
- Evaluation using network processors
 - Designed for packet processing
 - Abundant thread-level parallelism (64+)
 - Speedups of 2.0X-2.5X on common operators

Early insight on *heterogeneous* architectures and DBMS execution

Query Processing Architectures





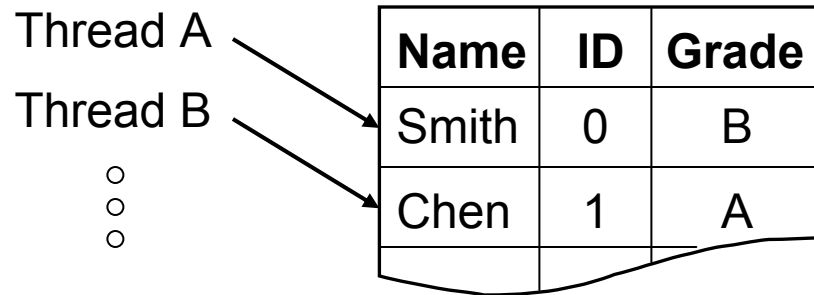
	<u>Intel IXP2400</u>	<u>Intel IXP2805</u>
<u>Microengines</u>	8	16
<u>Thread contexts</u>	64	128
<u>Clock rate</u>	600 MHz	1500 MHz
<u>Transistor count</u>	~60 M	~110M
<u>Power</u>	< 16 W	< 30 W
<u>Memory (DRAM)</u>	1x DDR	3x Rambus

TLP opportunity in DB operators

[GAH05]

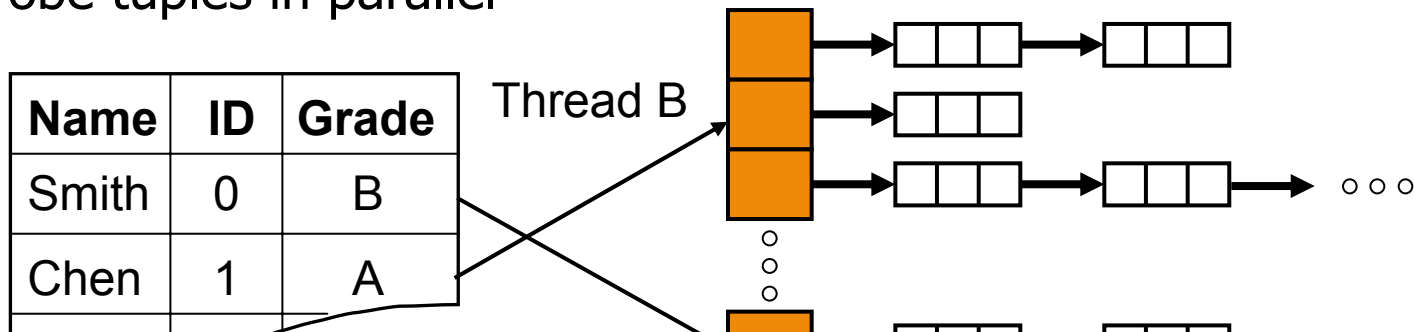
Sequential or index scan

- Fetch tuples in parallel



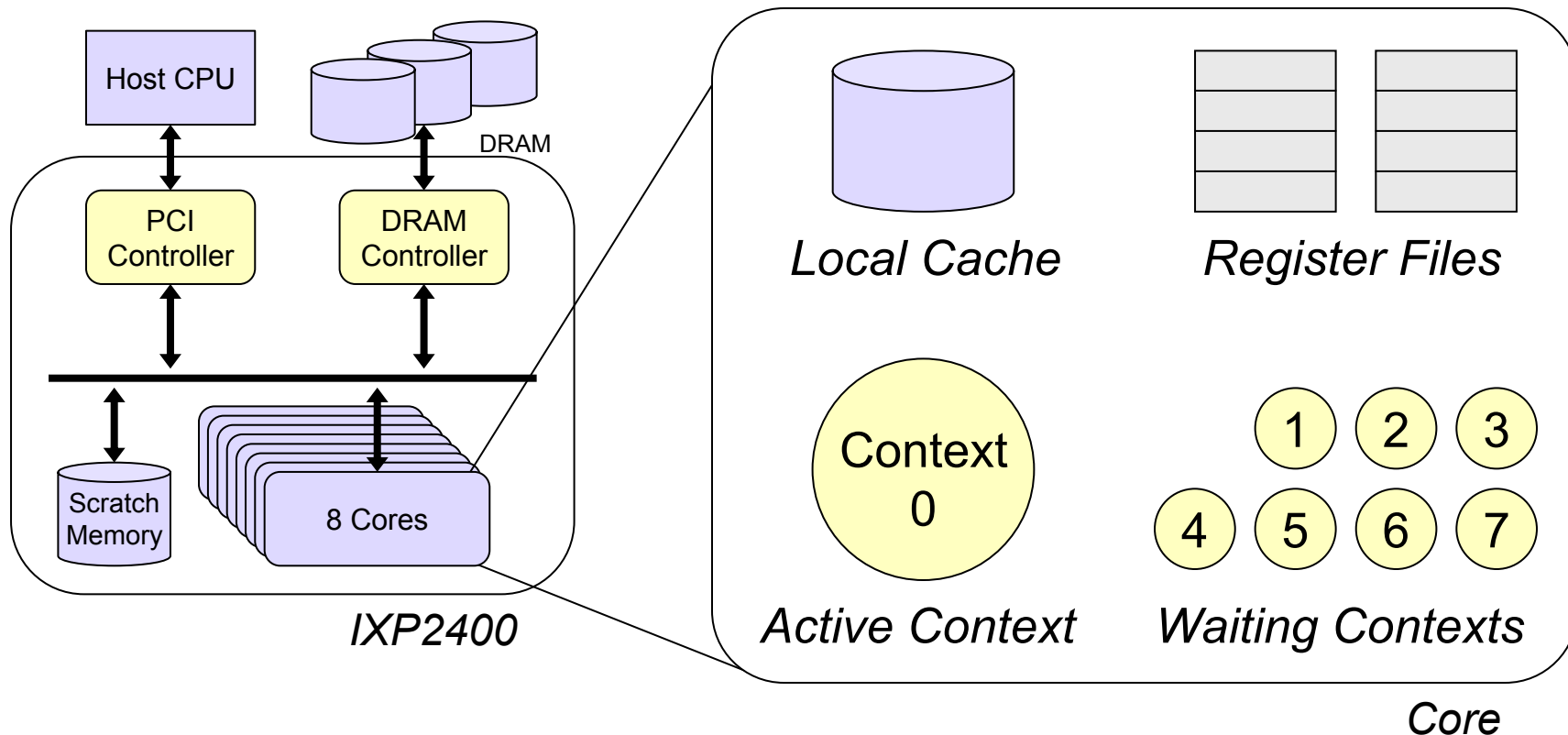
Hash join

- Probe tuples in parallel



Hardware thread support helps expose parallelism without significant overhead

Multi-threaded Core



● Simple processing core

- 5-stage, single-issue pipeline @ 600MHz, 2.5KB local cache
- Switch contexts at programmer's discretion

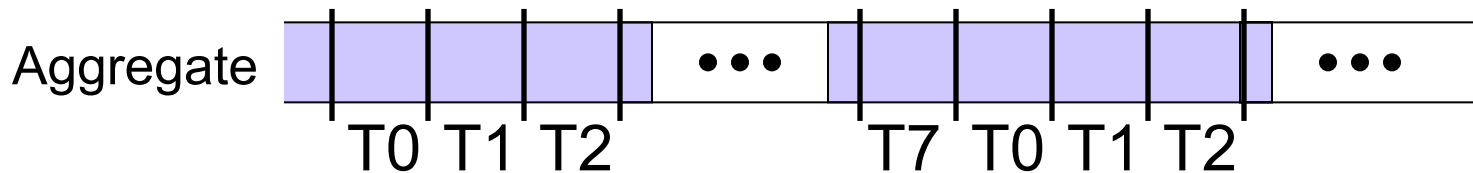
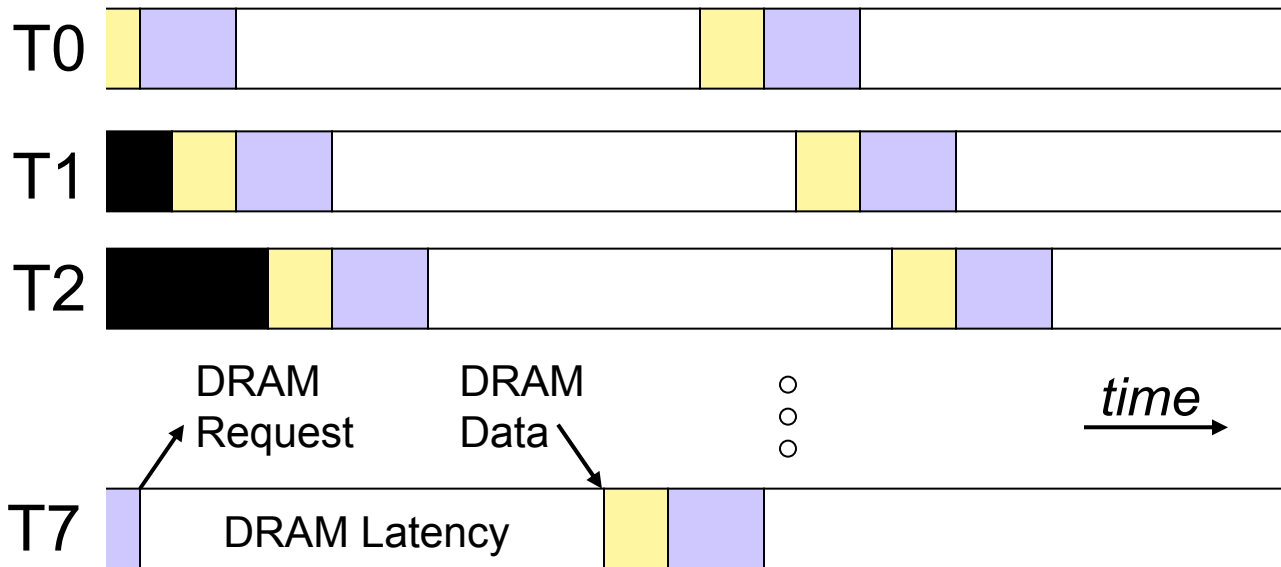
Sensible for simple, long-running code
Throughput, rather than single-thread performance

Multithreading in Action

Thread States: *Active* *Waiting* *Ready*

Thread
Contexts
(1 core)

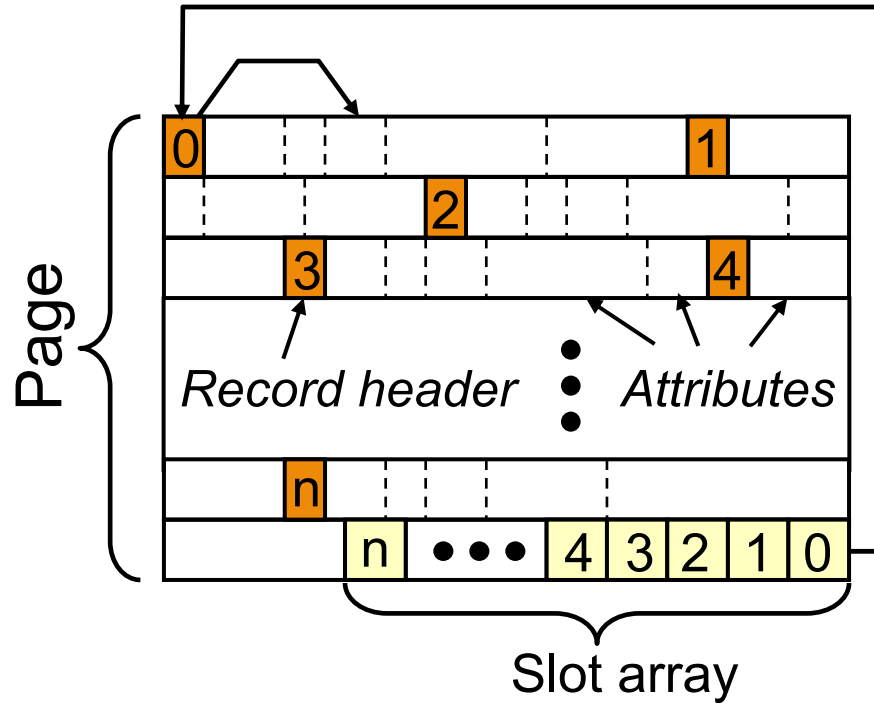


Methodology

- IXP2400 on prototype PCI card
 - 256MB PC2100 SDRAM
 - Separated from host CPU
- Pentium 4 Xeon 2.8GHz
 - 8KB L1D, 512KB L2, 4 pending misses
 - 3GB PC2100 SDRAM
- Workloads
 - TPC-H *orders* and *lineitems* tables (250MB)
 - Sequential scan, hash join

Sequential Scan Setup

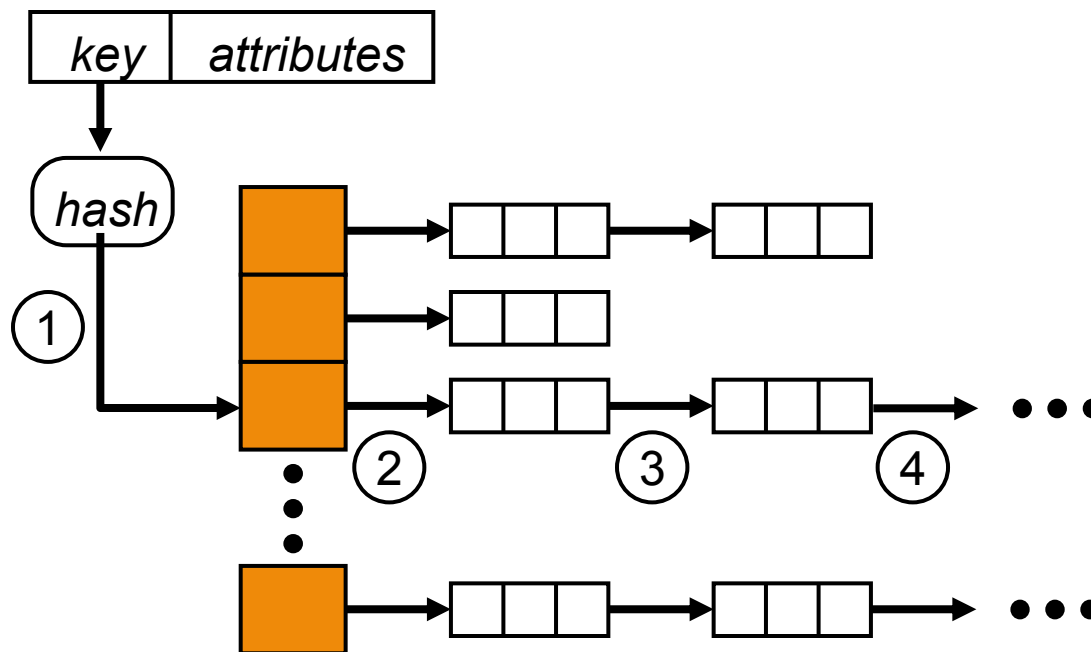
- Use slotted page layout (8KB)



- Network processor implementation
 - Each page scanned by threads on one core
 - Overlap individual record access within core

Hash Join Setup

- Model 'probe' phase



- Assign pages of outer relation to one core
 - Each thread context issues one probe
 - Overlap dependent accesses within core

Performance

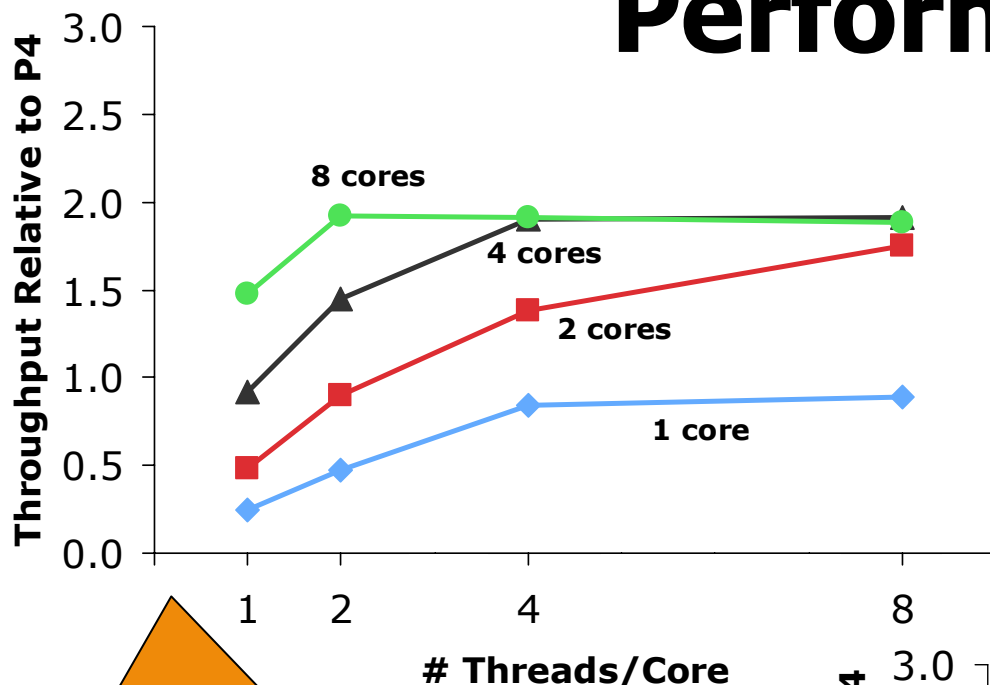
[GAH05]

IXP2400 prototype card

- 256MB PC2100 SDRAM
- Separated from host CPU

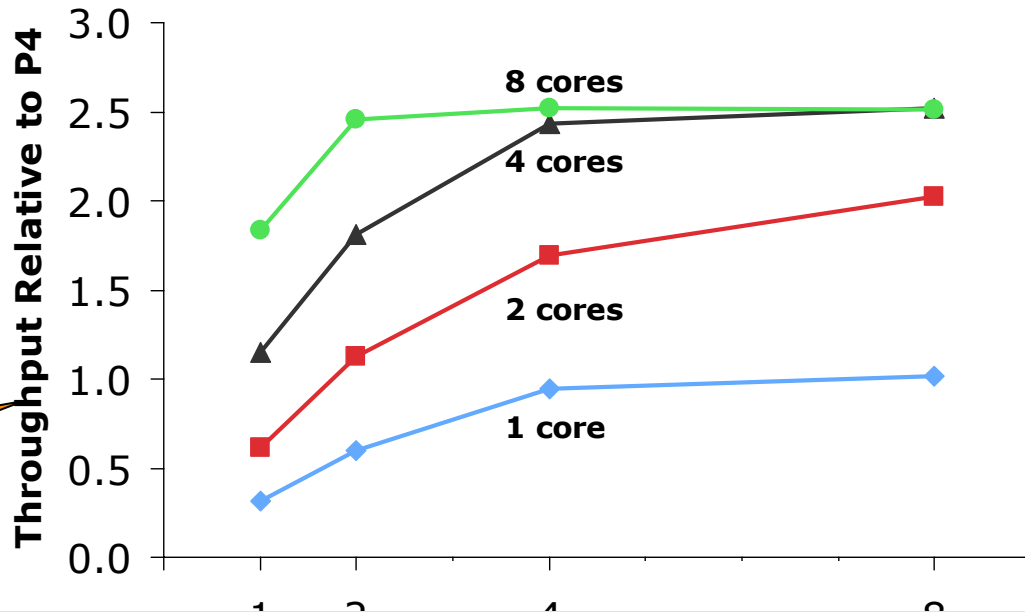
Pentium 4 Xeon 2.8GHz

- 8KB L1D, 512KB L2
- 3GB PC2100 SDRAM



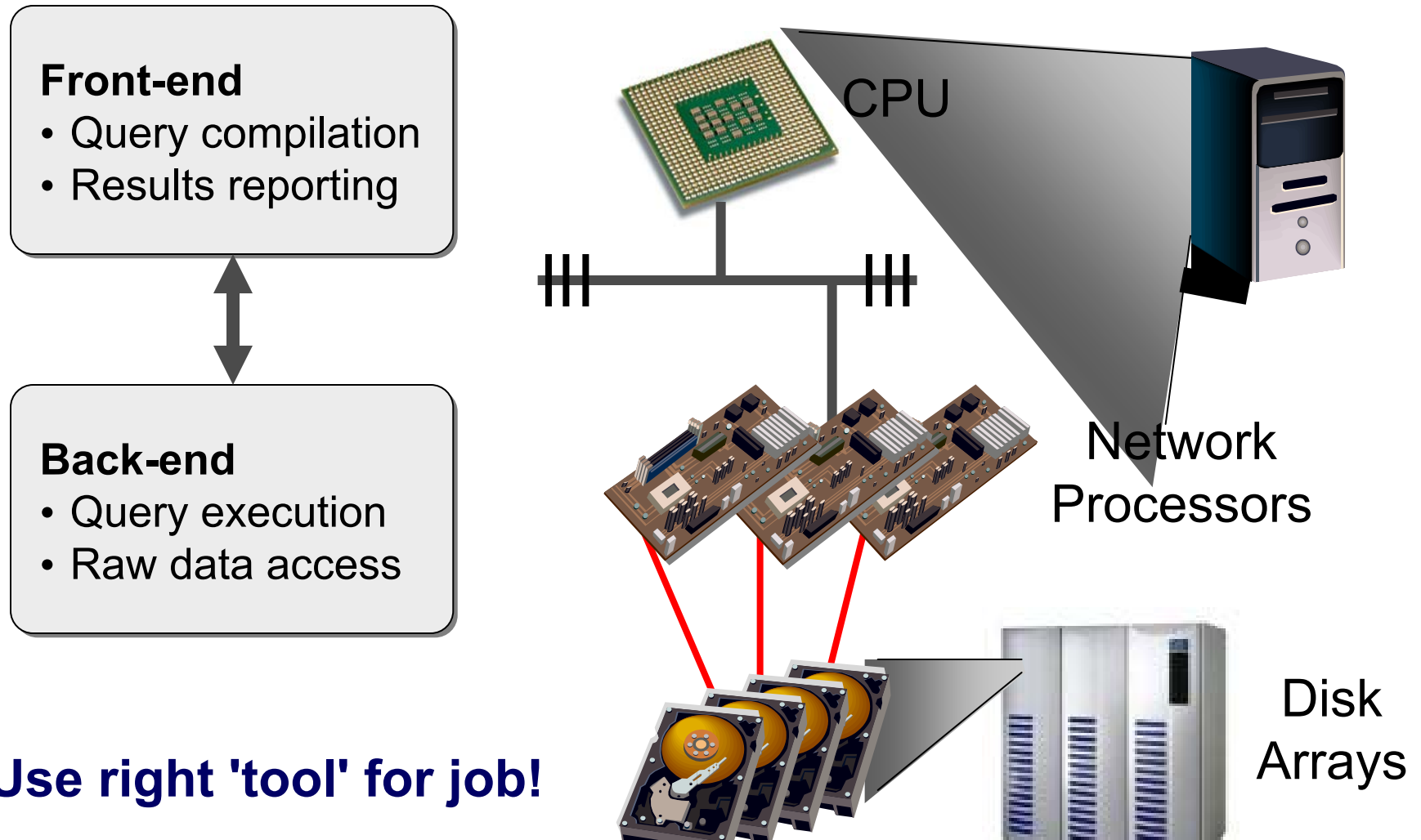
Sequential scan 250MB
(*Lineitem*)

Hash join (*Lineitem*, *Orders*)



Performance limited by DRAM controller

Query Coprocessing with NPs



Substantial intra-operator parallelism opportunity

Conclusions

- Uniprocessor architectures

- Main problem: memory access latency - still not resolved


- Multiprocessors: SMP, SMT, CMP

- Memory bandwidth a scarce resource
- Programmer/software to tolerate uneven memory accesses
- Lots of parallelism available in hardware
 - “will you still need me, will you still feed me, when I’m 64?”
- Immense data management research opportunities

- Query co-processing

- NPUs: Simple hardware, lots of threads, highly programmable
- Beat Pentium 4 by 2X-2.5X on DB operators
- Indication of need for heterogeneous processors?

Outline

- INTRODUCTION AND OVERVIEW
- DBs on CONVENTIONAL PROCESSORS
- QUERY co-PROCESSING: NETWORK PROCESSORS
-  ● **QUERY co-PROCESSING: GRAPHICS PROCESSORS**
- CONCLUSIONS AND FUTURE DIRECTIONS

References...



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)

- [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*.
- [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: Where Does Time Go?** 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*.

References

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.
- [YAA04] **Declustering Two-Dimensional Datasets over MEMS-based Storage.** H. Yu, D. Agrawal, and A.E. Abbadi. *International Conference on Extending DataBase Technology (EDBT)*, Heraklion-Crete, Greece, March 2004.
- [YAA03] **Tabular Placement of Relational Data on MEMS-based Storage Devices.** H. Yu, D. Agrawal, A.E. Abbadi. *International Conference on Very Large Data Bases (VLDB)*, Berlin, Germany, September 2003.
- [ZR03] **A Multi-Resolution Block Storage Model for Database Design.** J. Zhou and K.A. Ross. *International Database Engineering & Applications Symposium (IDEAS)*, Hong Kong, China, July 2003.
- [SSA03] **Exposing and Exploiting Internal Parallelism in MEMS-based Storage.** S.W. Schlosser, J. Schindler, A. Ailamaki, and G.R. Ganger. *Carnegie Mellon University, Technical Report CMU-CS-03-125*, March 2003
- [HP03] **Data Morphing: An Adaptive, Cache-Conscious Storage Technique.** R.A. Hankins and J.M. Patel. *International Conference on Very Large Data Bases (VLDB)*, Berlin, Germany, September 2003.
- [RDS02] **A Case for Fractured Mirrors.** R. Ramamurthy, D.J. DeWitt, and Q. Su. *International Conference on Very Large Data Bases (VLDB)*, Hong Kong, China, August 2002.
- [ADH02] **Data Page Layouts for Relational Databases on Deep Memory Hierarchies.** A. Ailamaki, D. J. DeWitt, and M. D. Hill. *The VLDB Journal*, 11(3), 2002.
- [ADH01] **Weaving Relations for Cache Performance.** A. Ailamaki, D.J. DeWitt, M.D. Hill, and M. Skounakis. *International Conference on Very Large Data Bases (VLDB)*, Rome, Italy, September 2001.
- [BMK99] **Database Architecture Optimized for the New Bottleneck: Memory Access.** P.A. Boncz, S. Manegold, and M.L. Kersten. *International Conference on Very Large Data Bases (VLDB)*, Edinburgh, the United Kingdom, September 1999.

References

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.
- [KCK01] **Optimizing Multidimensional Index Trees for Main Memory Access.** K. Kim, S.K. Cha, and K. Kwon. *ACM International Conference on Management of Data (SIGMOD)*, Santa Barbara, California, May 2001.
- [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.

References

Architecture-Conscious Query Processing

- [CAG05] **Inspector Joins.** Shimin Chen, Anastassia Ailamaki, Phillip B. Gibbons, and Todd C. Mowry. *International Conference on Very Large Data Bases (VLDB)*, Trondheim, Norway, September 2005.
- [MBN04] **Cache-Conscious Radix-Decluster Projections.** Stefan Manegold, Peter A. Boncz, Niels Nes, Martin L. Kersten. *International Conference on Very Large Data Bases (VLDB)*, Toronto, Canada, September 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.
- [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.

References

Instruction Stream Optimizations and DBMS Architectures

- [HSA05] **QPipe: A Simultaneously Pipelined Relational Query Engine.** S. Harizopoulos, V. Shkapenyuk and A. Ailamaki. *ACM International Conference on Management of Data (SIGMOD)*, Baltimore, MD, June 2005.
- [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

Data management on SMT, CMP, and SMP

- [CAS06] **Tolerating Dependences Between Large Speculative Threads Via Sub-Threads.** Christopher B. Colohan, Anastassia Ailamaki, J. Gregory Steffan and Todd C. Mowry. *International Symposium on Computer Architecture (ISCA)*. Boston, MA, June 2006.
- [CAS05] **Improving Database Performance on Simultaneous Multithreading Processors.** J. Zhou, J. Cieslewicz, K. A. Ross and M. Shah. *International Conference on Very Large Data Bases (VLDB)*, Trondheim, Norway, September 2005.
- [ZCR05] **Optimistic Intra-Transaction Parallelism on Chip Multiprocessors.** Christopher B. Colohan, Anastassia Ailamaki, J. Gregory Steffan and Todd C. Mowry. *International Conference on Very Large Data Bases (VLDB)*, Trondheim, Norway, September 2005.
- [GAH05] **Accelerating Database Operations Using a Network Processor.** Brian T. Gold, Anastassia Ailamaki, Larry Huston, Babak Falsafi. *Workshop on Data Management on New Hardware (DaMoN)*, Baltimore, Maryland, USA, 2005.
- [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.
- [DJN02] **Shared Cache Architectures for Decision Support Systems.** M. Dubois, J. Jeong, A. Nanda, *Performance Evaluation* 49(1), September 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.