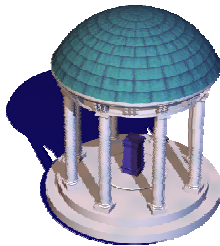# Query co-Processing on Commodity Processors
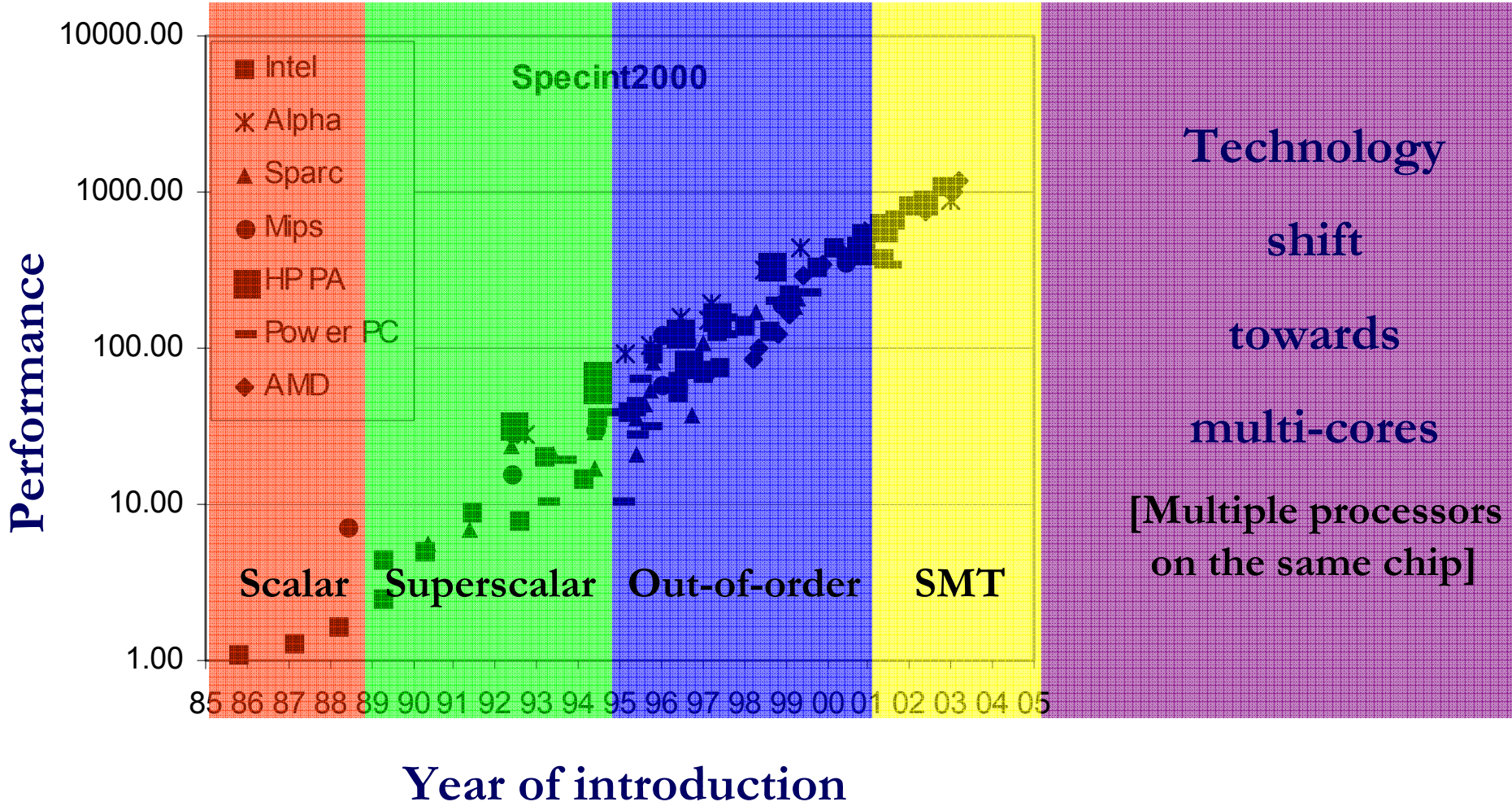
Anastassia Ailamaki
Carnegie Mellon University

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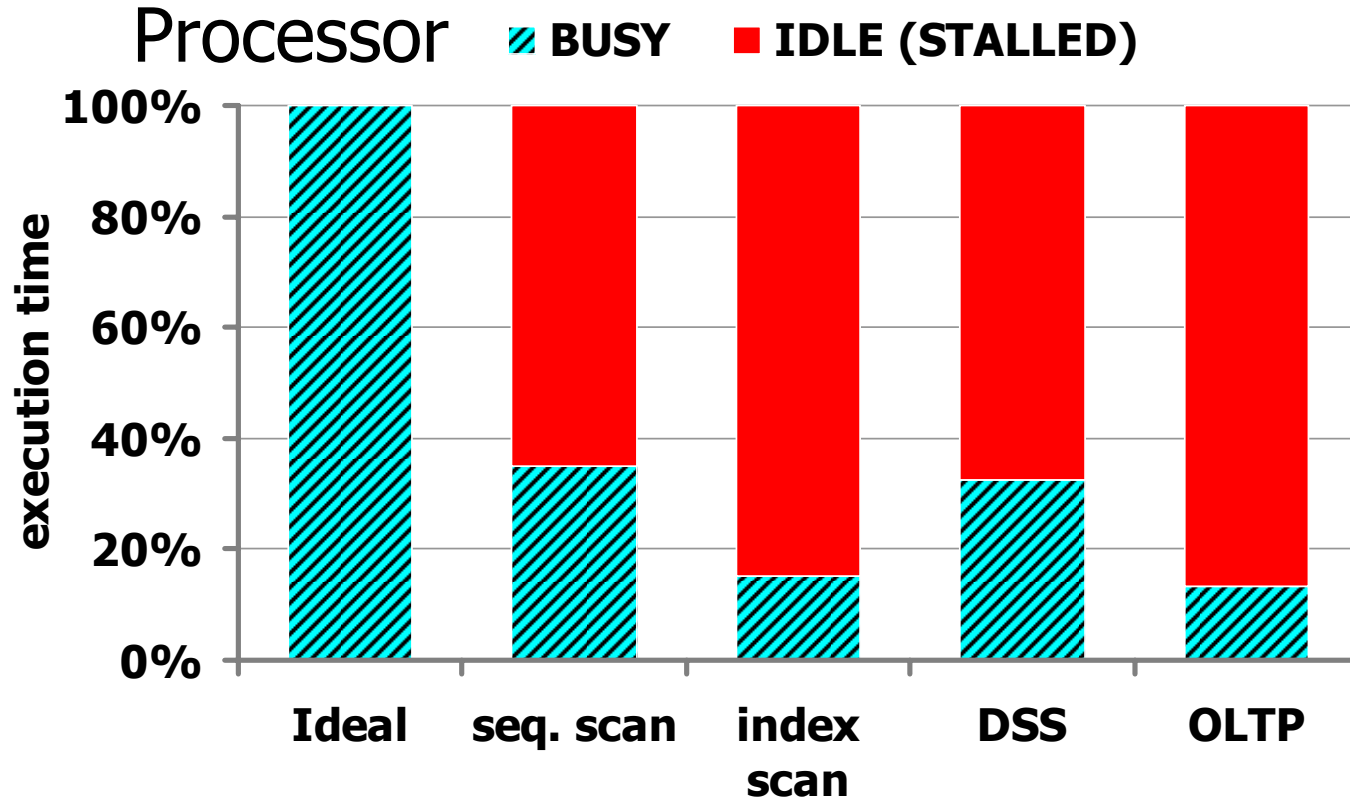*University of North Carolina at Chapel Hill*

Stavros Harizopoulos
MIT

Databases
@Carnegie Mellon

The UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Processor Performance over Time*

Focus of this Tutorial

- DB workload execution on a modern computer

Processor   ▨ **BUSY**   ■ **IDLE (STALLED)**

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

Detailed Tutorial Outline

- INTRODUCTION AND OVERVIEW
  - How computer architecture trends affect database workload behavior
  - CPUs, NPUs, and GPUs: opportunities 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 GPUs
  - 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

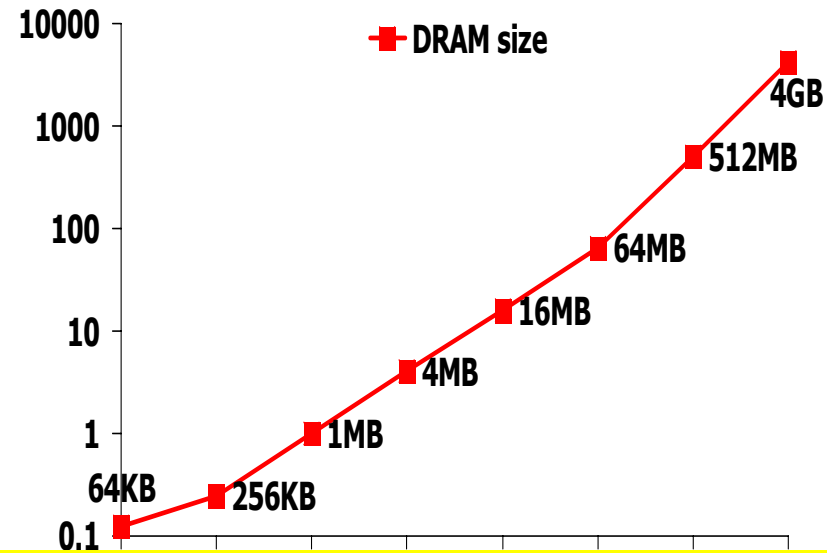- DBs on CONVENTIONAL PROCESSORS
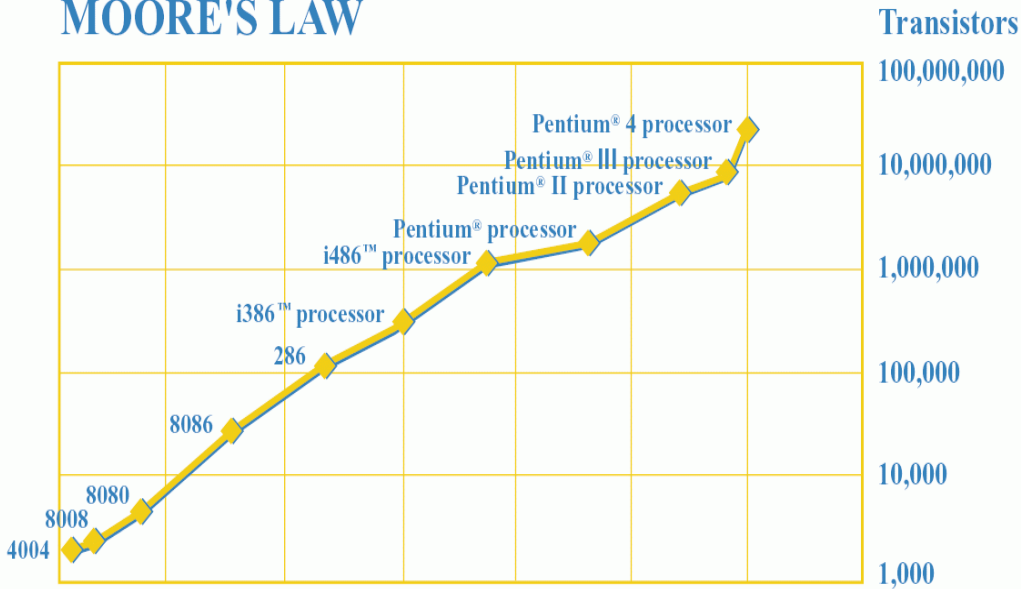
- QUERY co-PROCESSING: NETWORK PROCESSORS

- QUERY co-PROCESSING: GRAPHICS PROCESSORS

- CONCLUSIONS AND FUTURE DIRECTIONS
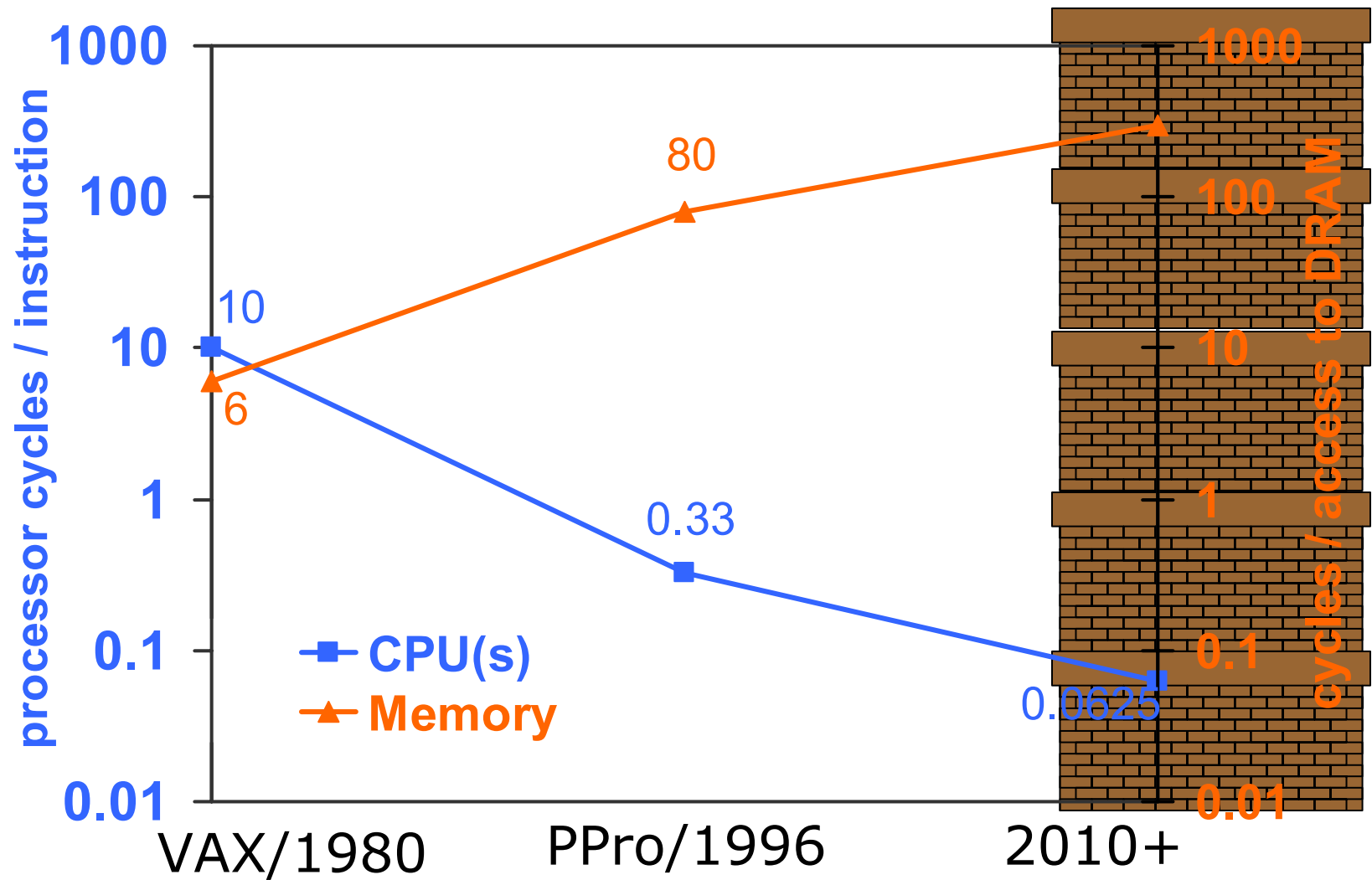
Processor/memory speed gap

- Moore's Law (despite technological hurdles!)
  - Innovative processor microarchitecture
  - Memory capacity increases exponentially
  - Speed increases linearly

MOORE'S LAW — Transistors: 100,000,000; 10,000,000; 1,000,000; 100,000; 10,000; 1,000. Points labeled: 4004, 8008, 8080, 8086, 286, i386™ processor, i486™ processor, Pentium® processor, Pentium® II processor, Pentium® III processor, Pentium® 4 processor.

DRAM size — 0.1, 1, 10, 100, 1000, 10000. Points labeled: 64KB, 256KB, 1MB, 4MB, 16MB, 64MB, 512MB, 4GB.

**2x processor speed every 18 months
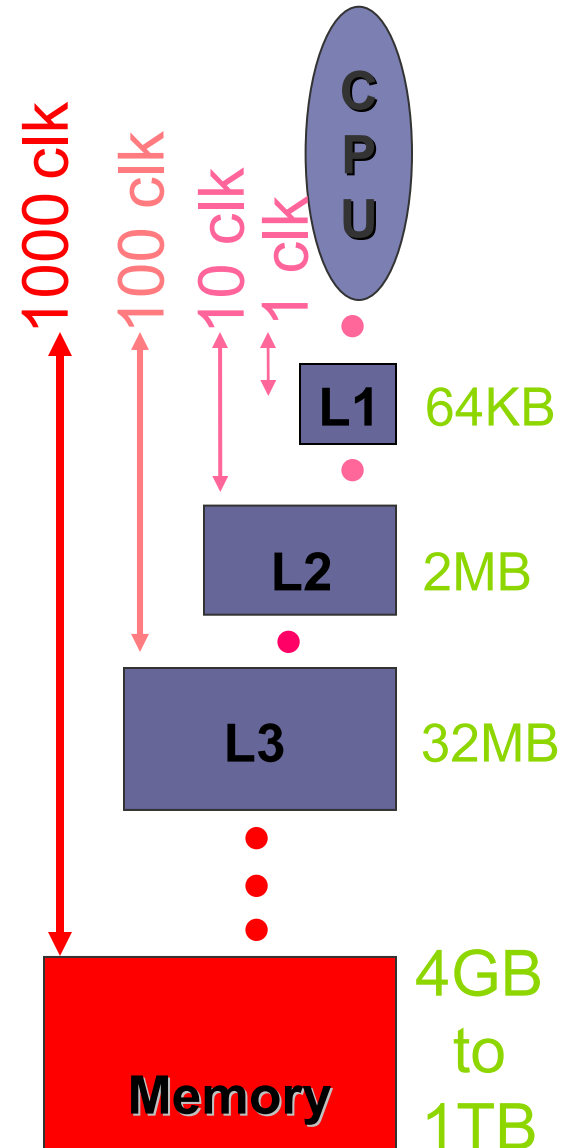Larger but not as much faster memories**

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**

C P U

1000 clk
100 clk
10 clk
1 clk

L1   64KB

L2   2MB

L3   32MB

Memory   4GB to 1TB

Efficient cache utilization is crucial!

Outline

- INTRODUCTION AND OVERVIEW
  - **Computer architecture trends and DB workloads**
    - Processor/memory speed gap
    - Instruction-level parallelism (ILP)
    - Chip multiprocessors and multithreading

- DBs on CONVENTIONAL PROCESSORS

- QUERY co-PROCESSING: NETWORK PROCESSORS

- QUERY co-PROCESSING: GRAPHICS PROCESSORS

- CONCLUSIONS AND FUTURE DIRECTIONS
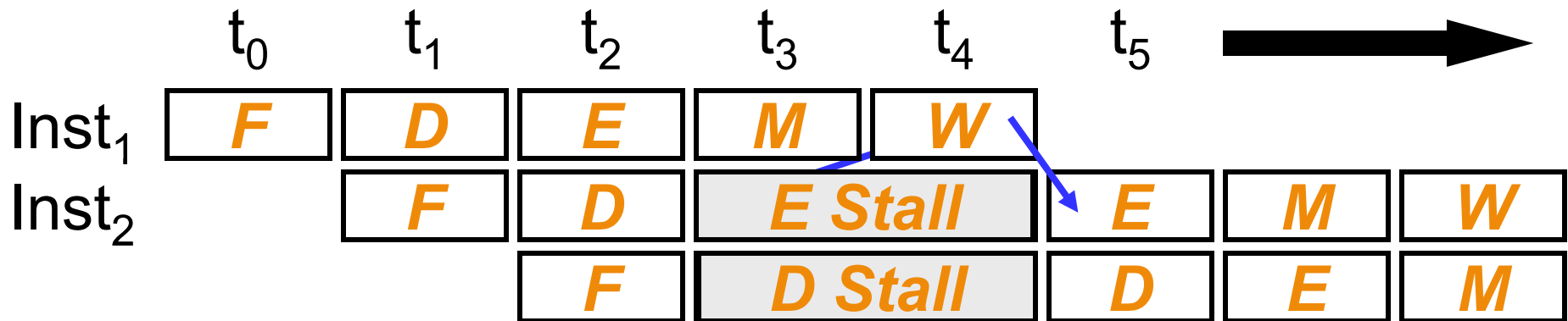
ILP: Processor pipelines

- Problem: dependences between instructions
- E.g.,
  - $Inst_1$: $r1 \leftarrow r2 + r3$
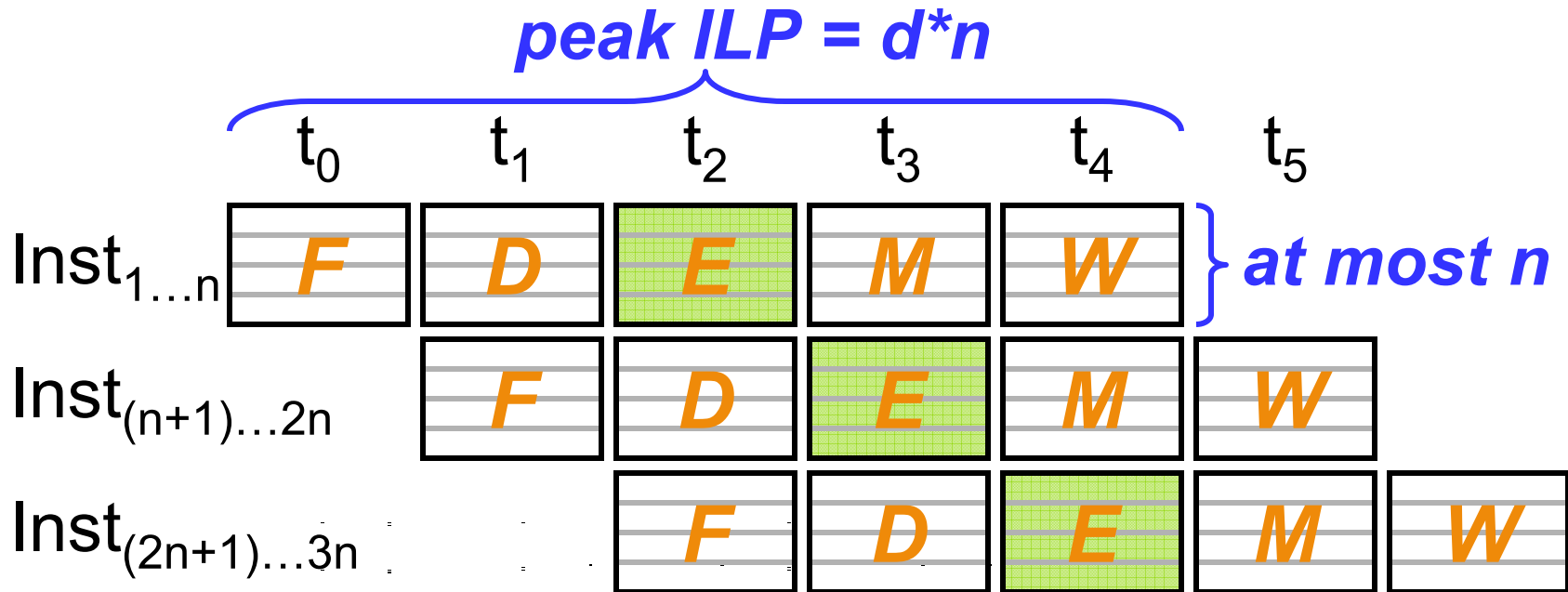 - $Inst_2$: $r4 \leftarrow r1 + r2$

    Read-after-write (RAW)

	t_0	t_1	t_2	t_3	t_4	t_5	
$Inst_1$	F	D	E	M	W		
$Inst_2$		F	D	E Stall	E	M	W
			F	D Stall	D	E	M

Real CPI > 1; Real ILP < 5

DB apps: frequent data dependences

> ILP: Superscalar Out-of-Order

peak ILP = d*n

	t_0	t_1	t_2	t_3	t_4	t_5

$Inst_{1...n}$ F   D   E   M   W   } *at most n*

$Inst_{(n+1)...2n}$ F   D   E   M   W

$Inst_{(2n+1)...3n}$ F   D   E   M   W

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: only 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
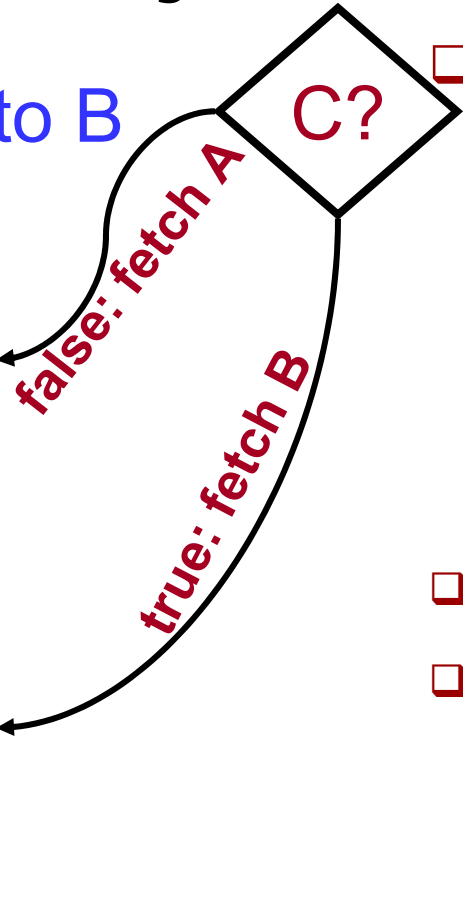
```
xxxx
if C goto B
A: aaaa
   aaaa
   aaaa
   aaaa
B: bbbb
   bbbb
   bbbb
   bbbb
   bbbb
```

C?

false: fetch A

true: fetch B

- ❑ IDEA: Speculate branch *while evaluating C*!
  - ■ Record history, predict A/B
  - ✓ If correct, saved a (long) delay!
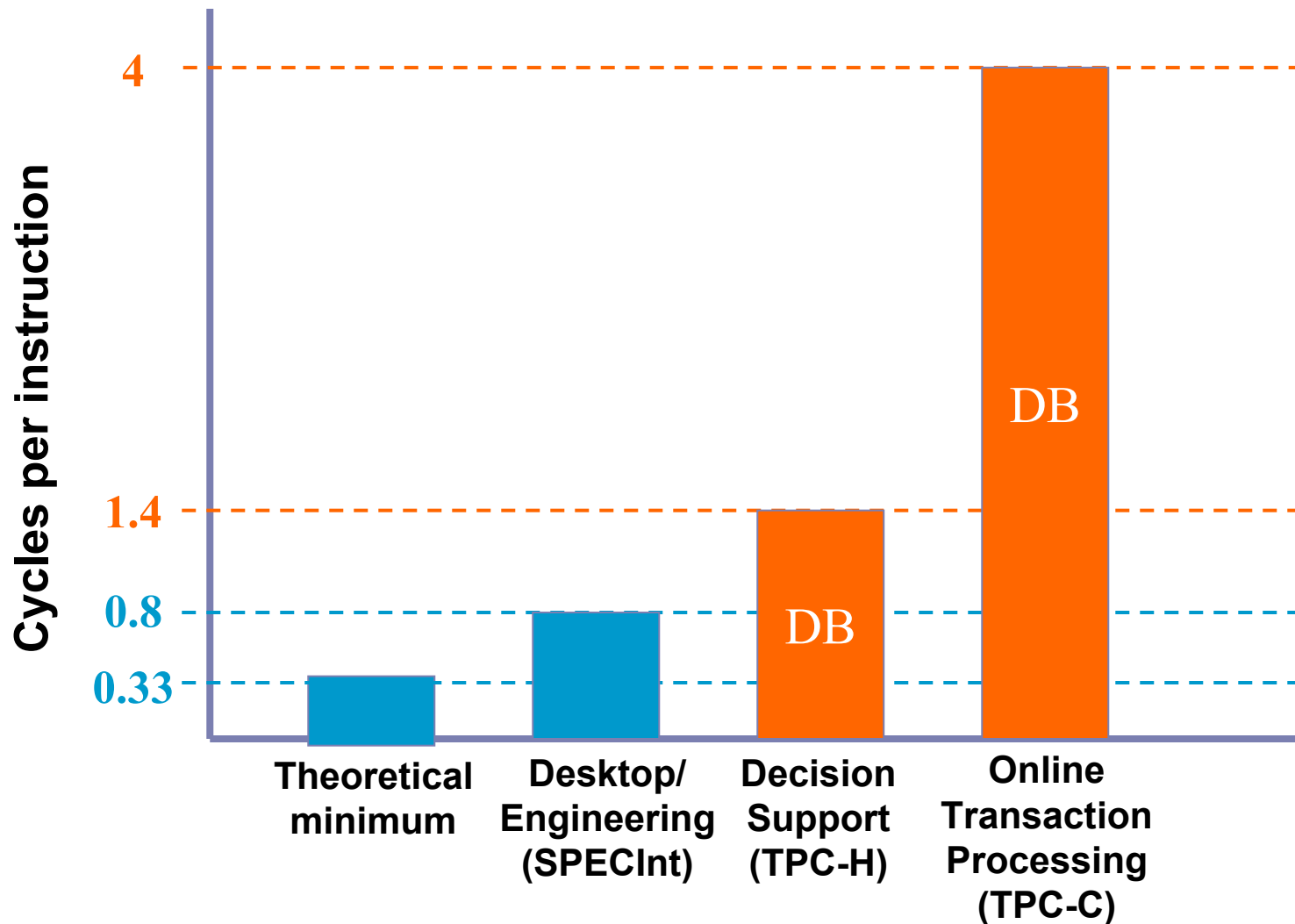 - 💣 If incorrect, misprediction penalty

- ❑ Excellent predictors (97% accuracy!)
- ❑ Mispredictions costlier in OOO
  - ❑ 1 lost cycle = >>1 missed instructions!

DB programs: long code paths => mispredictions

Database workloads on UPs

Cycles per instruction

Value	Workload
0.33	Theoretical minimum
0.8	Desktop/Engineering (SPECInt)
1.4	Decision Support (TPC-H) — DB
4	Online Transaction Processing (TPC-C) — DB

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

- DBs on CONVENTIONAL PROCESSORS

- QUERY co-PROCESSING: NETWORK PROCESSORS

- QUERY co-PROCESSING: GRAPHICS PROCESSORS

- CONCLUSIONS AND FUTURE DIRECTIONS

Coarse-grain parallelism

- ## Multithreading
  - Pursue multiple threads in parallel within a processor pipeline
  - Store multiple contexts in different register sets
  - Multiplex functional units between threads
  - Fast (hardware) context switching amongst threads
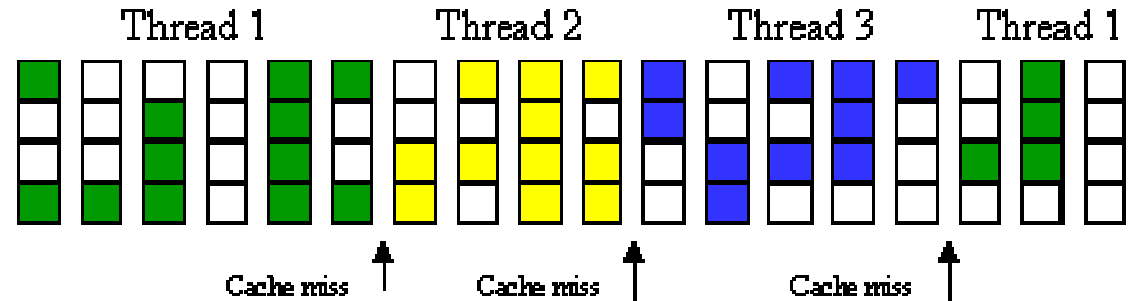
- ## Chip multiprocessors (CMPs)
  - >1 complete processors on a single chip
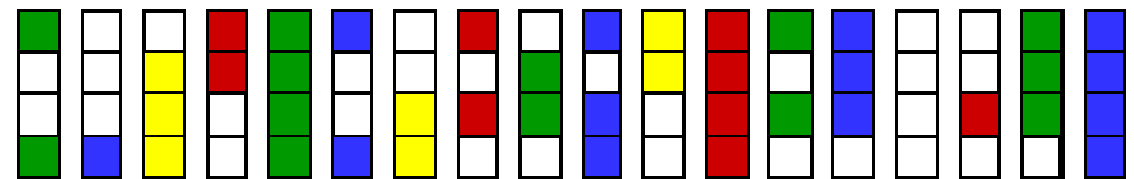 - Every functional unit of a processor is duplicated

A) Conventional Processor

Thread 1   OS context switch code   Thread 2

Interrupt, exception, or OS call   return from exception
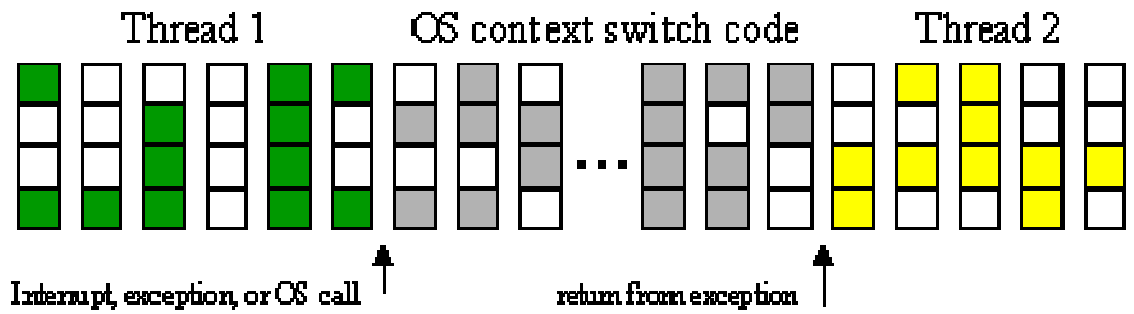
RS64-IV (IBM)

B) Coarse-grained Multithreaded (CMT)

Thread 1   Thread 2   Thread 3   Thread 1

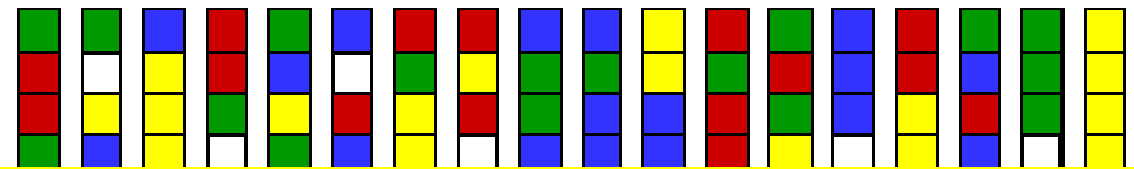Cache miss   Cache miss   Cache miss

Ultrasparc T1 (Sun)

C) Fine-grained Multithreaded (FMT)

POWER5 (IBM)

D) Simultaneous Multithreaded

Speedup: OLTP 3x, DSS 1.5x [LBE98]

The case for CMPs

- Getting diminishing returns
  - from a single core, although powerful
    (OoO, superscalar, multithreaded)
  - from a very large cache

- n-core CMP outperforms n-thread SMT
- CMPs offer productivity advantages

- Moore's law: 2x *transistors* every 18 months
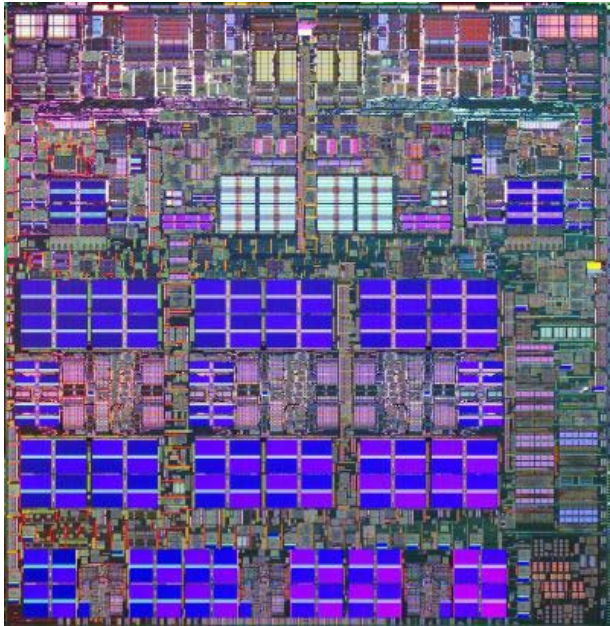 - More, not faster
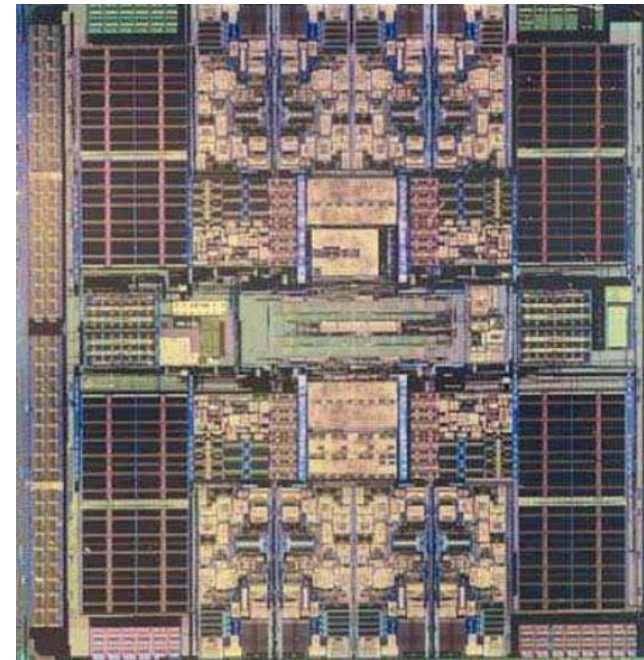
Expect exponentially more cores

A chip multiprocessor

CHIP 1

CHIP 2

L1I L1D

L2 CACHE

L3 CACHE

MAIN MEMORY
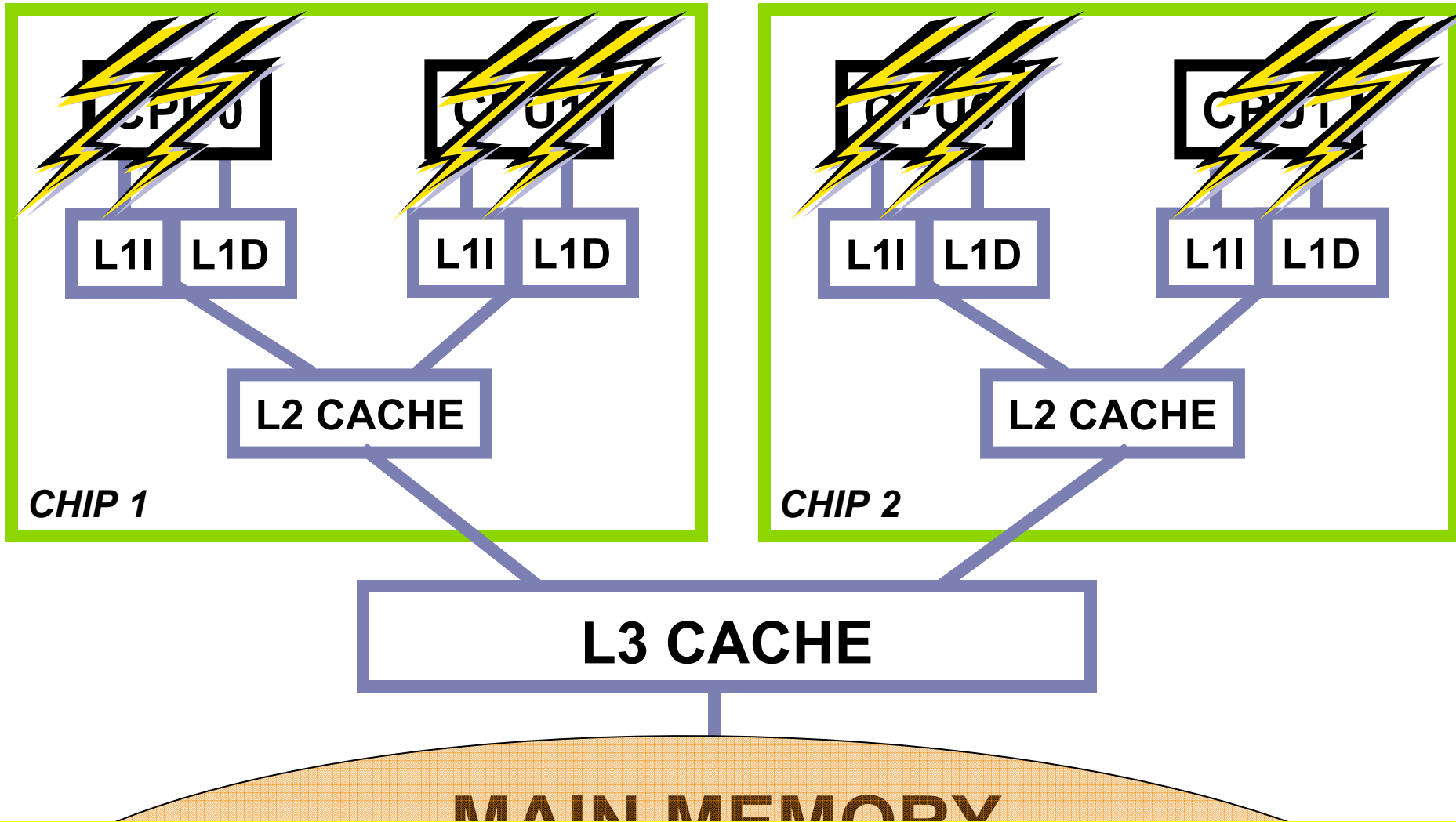
Highly variable memory latency
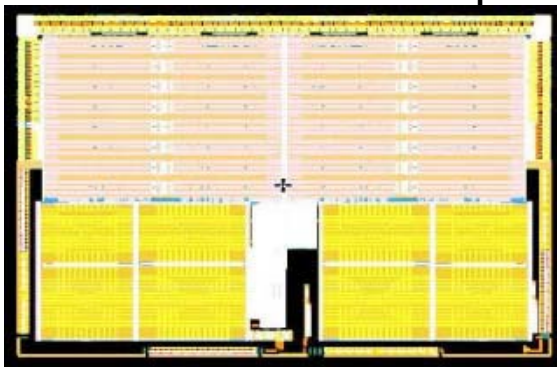**Speedup: OLTP 3x, DSS 2.3x on Piranha [BGM00]**
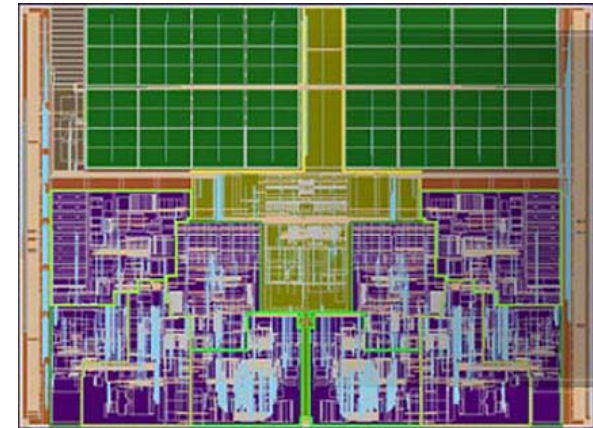
Current CMP technology

IBM Power 5

Sun Niagara

AMD Opteron

Intel Yonah

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

Summary: Trends & DB workloads

- ● Hardware: continuously evolving
  - ● Superscalar → OoO → SMT → CMP
  - ● Processor/memory speed gap: growing
- ● Software: one processor does not fit all
  - ● At most 50% CPU utilization
  - ● 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: NPUs and GPUs

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
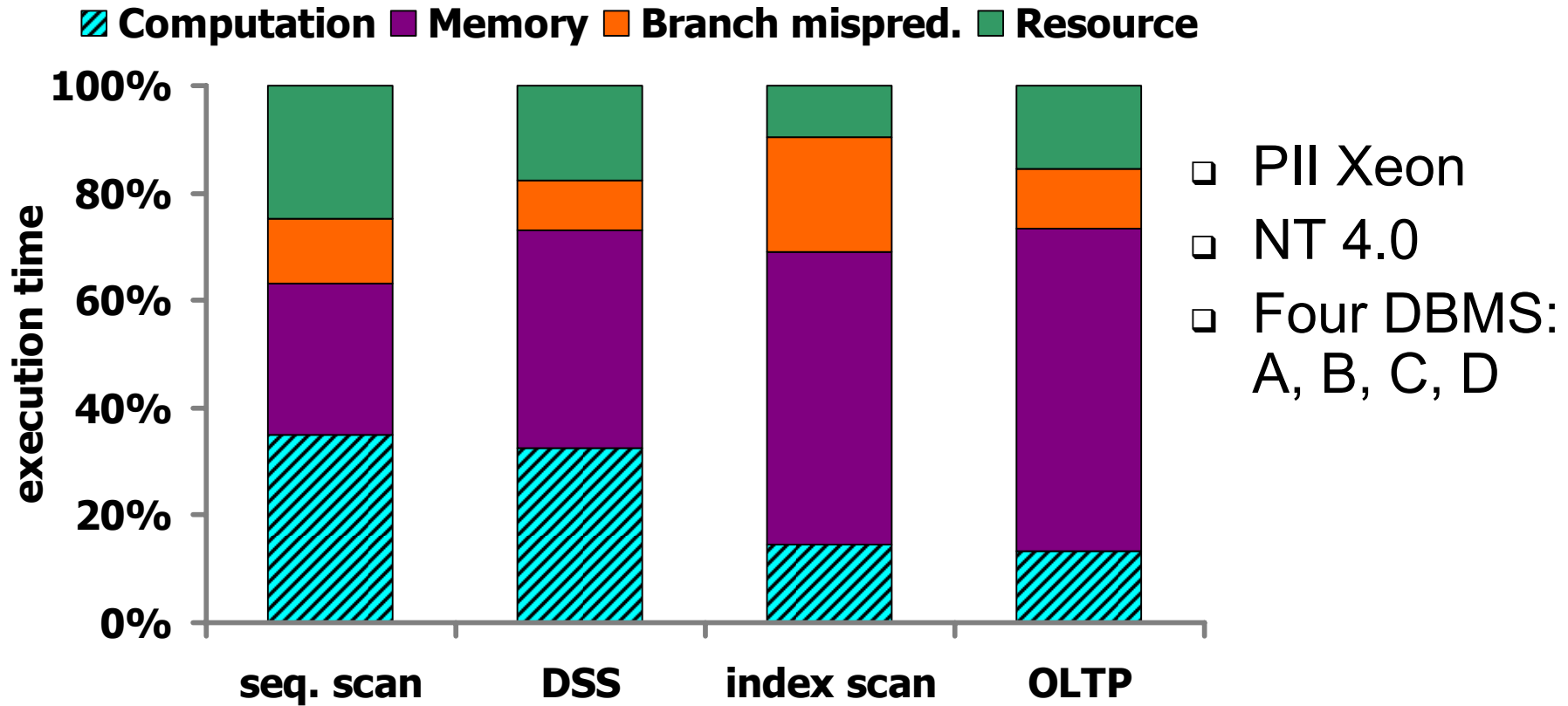 - Chip multiprocessor DB architectures

- QUERY co-PROCESSING: NETWORK PROCESSORS

- QUERY co-PROCESSING: GRAPHICS PROCESSORS

- CONCLUSIONS AND FUTURE DIRECTIONS

DB Execution Time Breakdown

[ADH99,BGB98,BGN00,KPH98,SAF04]

Legend: ▨ Computation ▪ Memory ▪ Branch mispred. ▪ Resource

y-axis: execution time (0% to 100%)
x-axis categories: seq. scan, DSS, index scan, OLTP

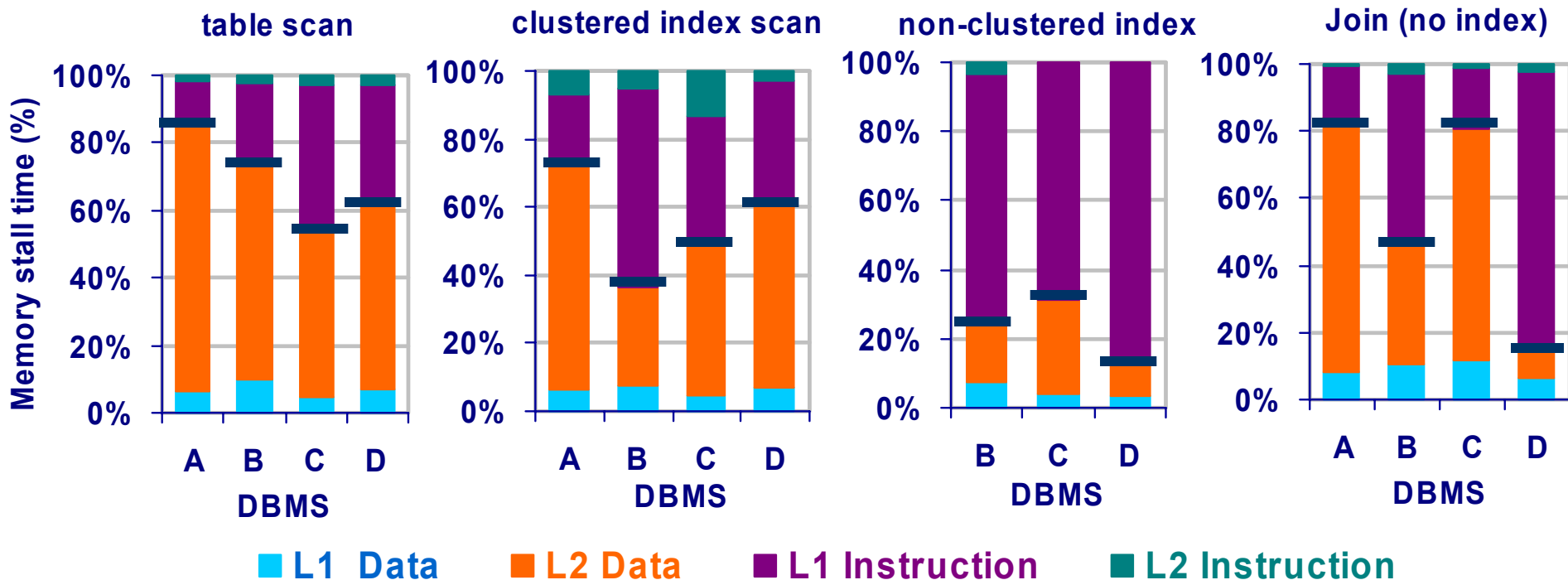- PII Xeon
- NT 4.0
- Four DBMS: A, B, C, D

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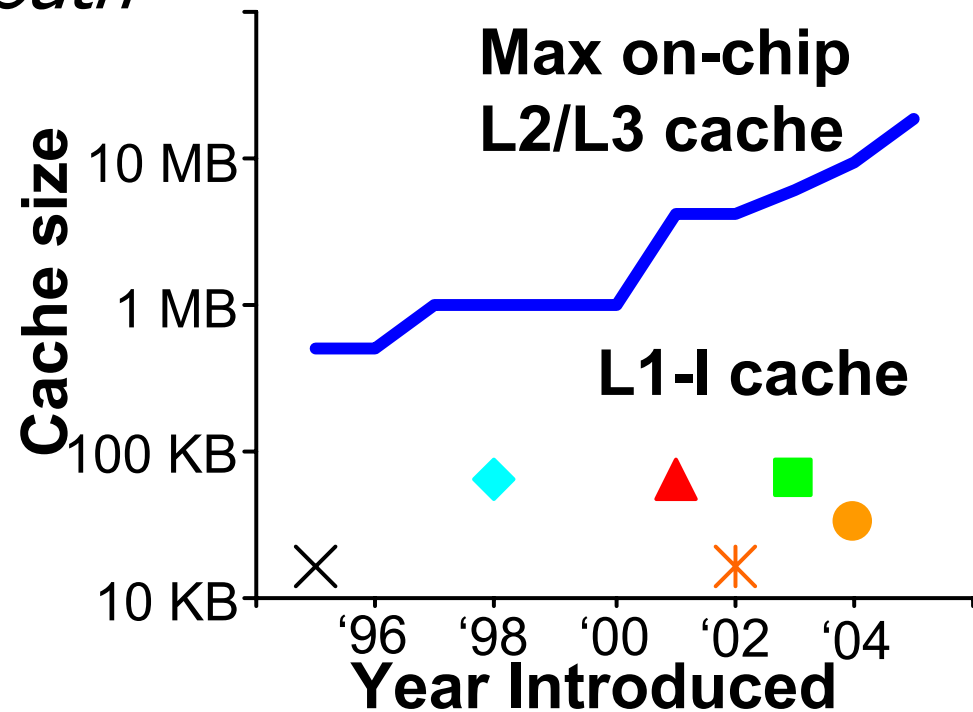Four commercial Database Systems: A, B, C, D

**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:

Max on-chip L2/L3 cache

Cache size

10 MB —

1 MB —

L1-I cache

100 KB —

10 KB —

'96 '98 '00 '02 '04

Year Introduced

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
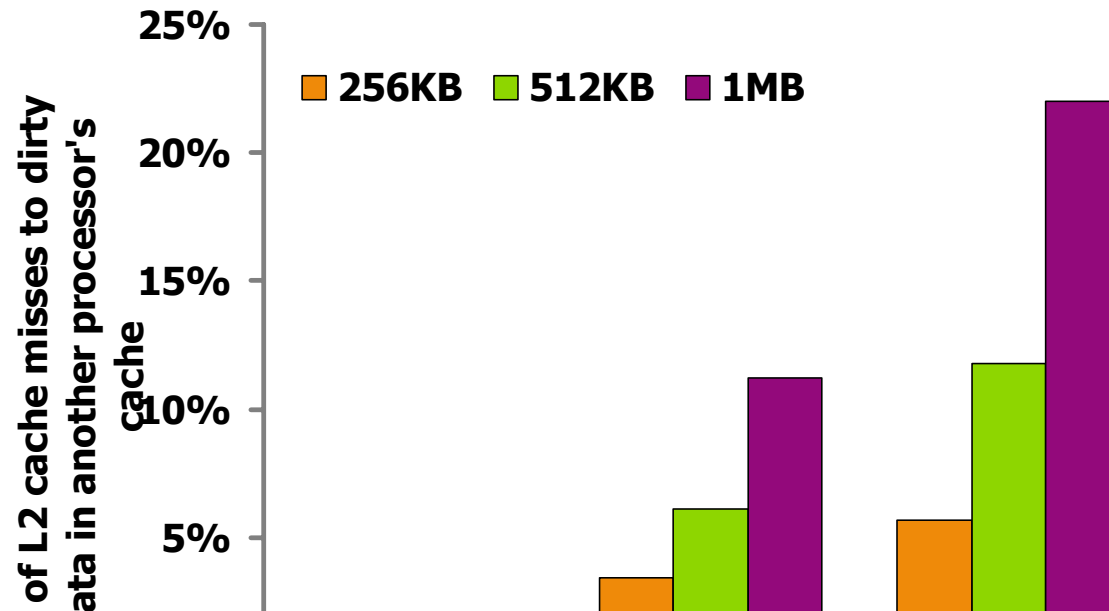
Larger L2: trade-off for OLTP Hardware needs help from software

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
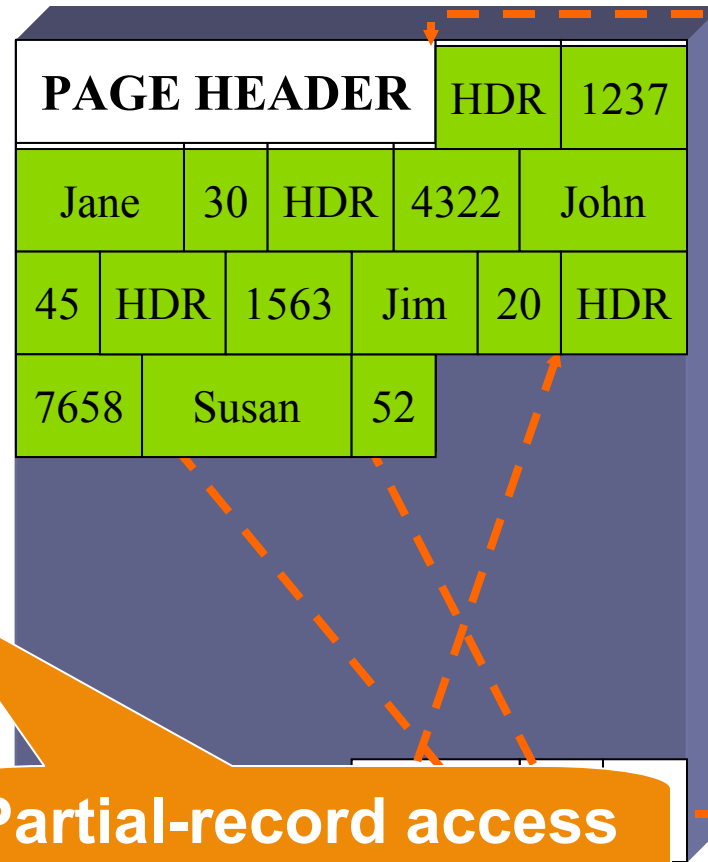
- QUERY co-PROCESSING: NETWORK PROCESSORS

- 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

PAGE HEADER		HDR	1237

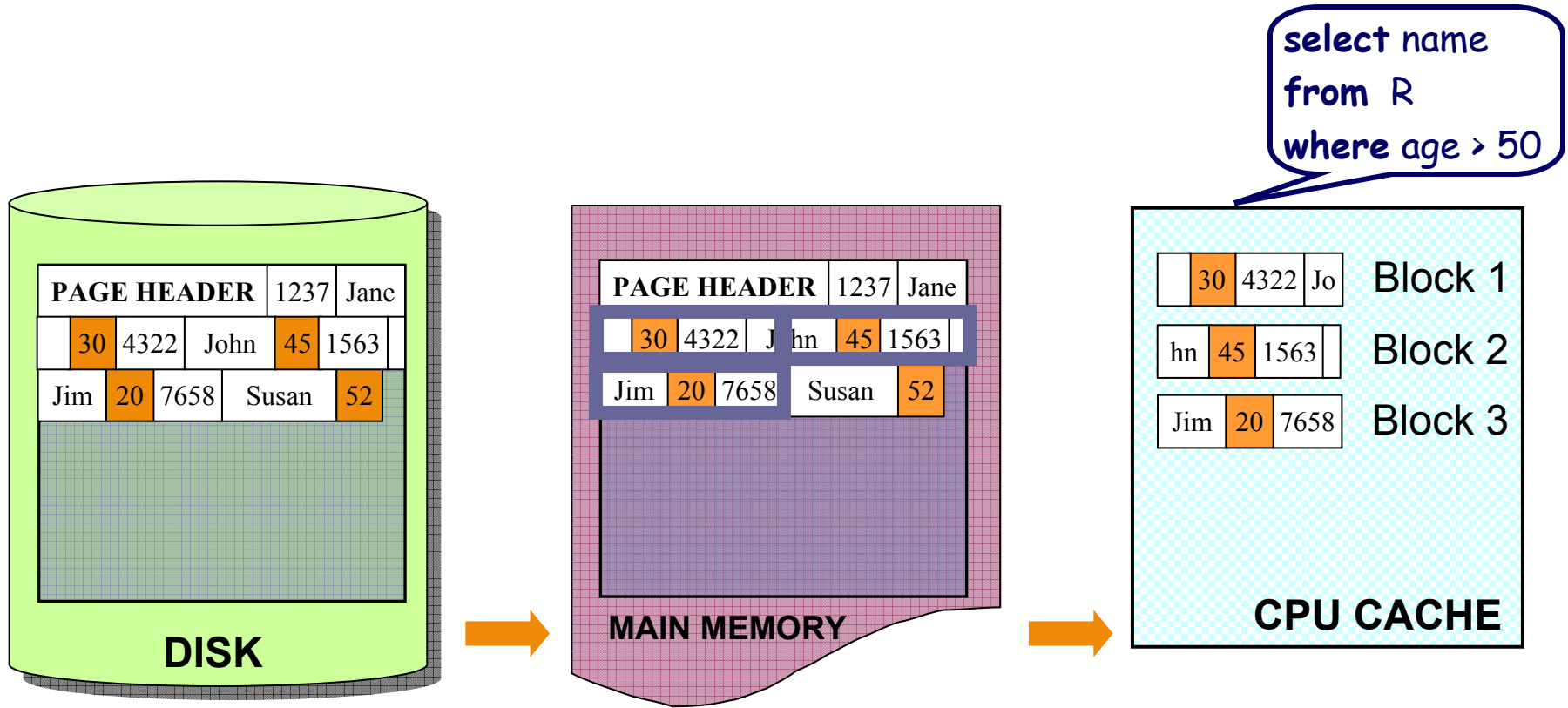| Jane	30	HDR	4322	John
45	HDR	1563	Jim	20	HDR
7658	Susan	52			

Full-record access

Partial-record access

rds stored sequentially
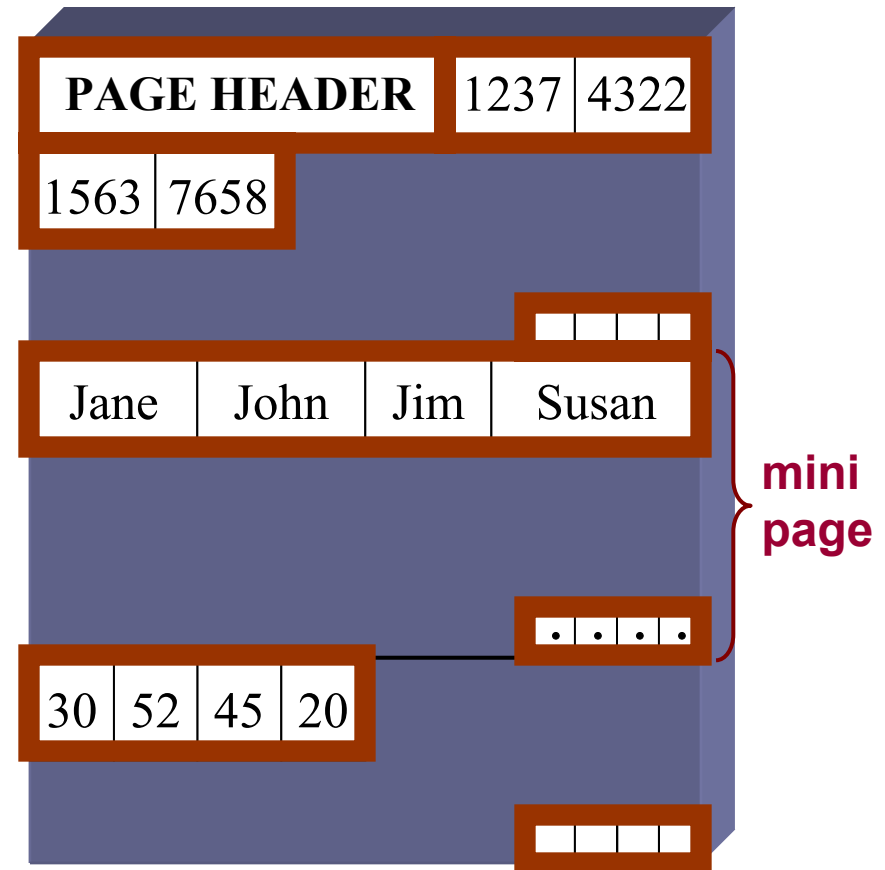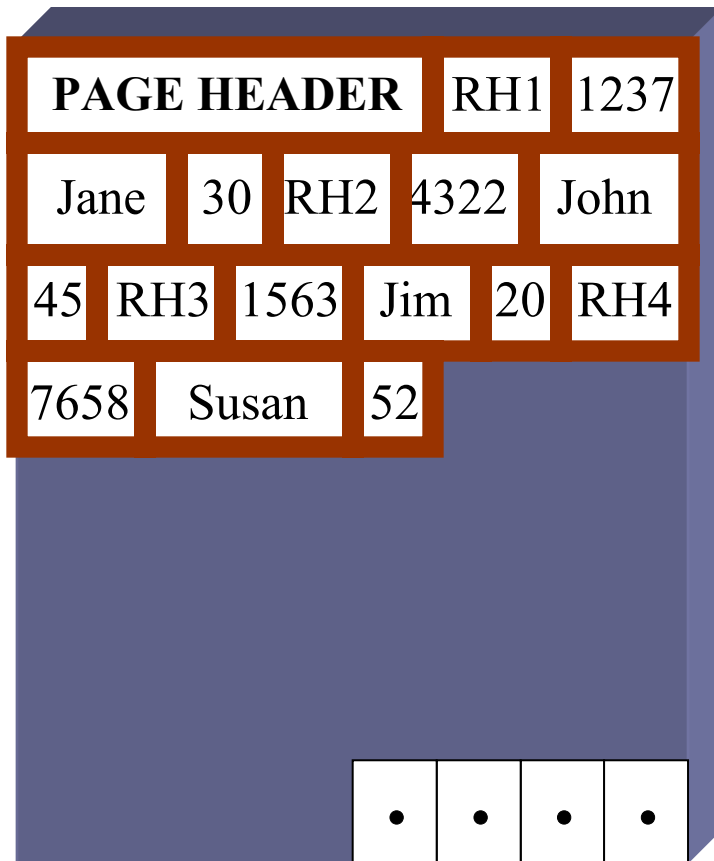
Attributes of a record stored together

NSM in Memory Hierarchy

NSM optimized for full-record access
Hurts partial-record access at all levels
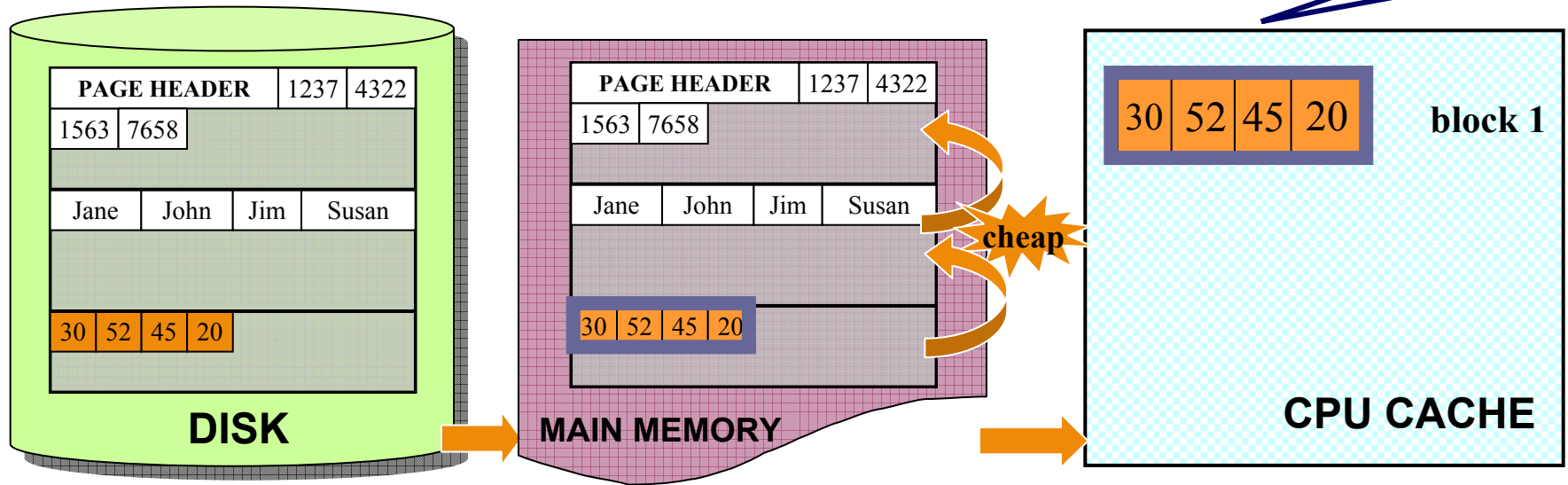
Partition Attributes Across (PAX)

[ADH01]

PAX partitions *within* page: cache locality

PAX in Memory Hierarchy

[ADH01]

select name
from R
where age > 50

PAGE HEADER 1237 4322
1563 7658

| Jane | John | Jim | Susan |

| 30 | 52 | 45 | 20 |

DISK

PAGE HEADER 1237 4322
1563 7658

| Jane | John | Jim | Susan |

| 30 | 52 | 45 | 20 |

cheap

MAIN MEMORY

| 30 | 52 | 45 | 20 | **block 1**

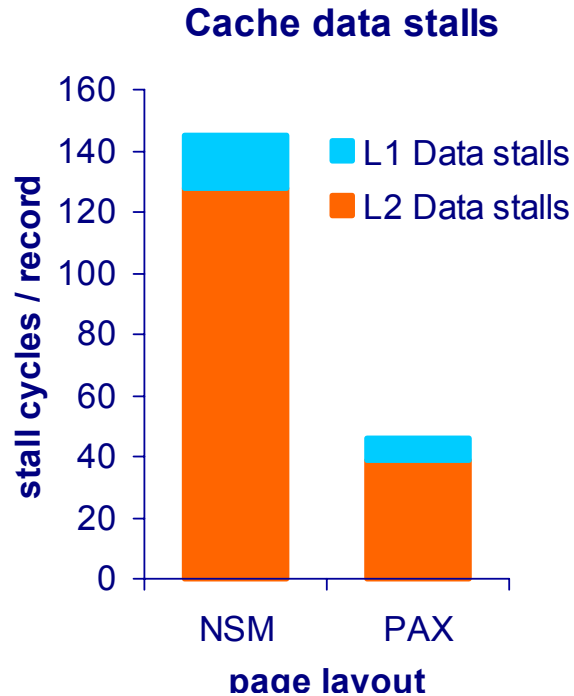**CPU CACHE**

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

PAX Performance Results (Shore)

[ADH01]

Cache data stalls

stall cycles / record — L1 Data stalls, L2 Data stalls — NSM, PAX — page layout

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

- 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]
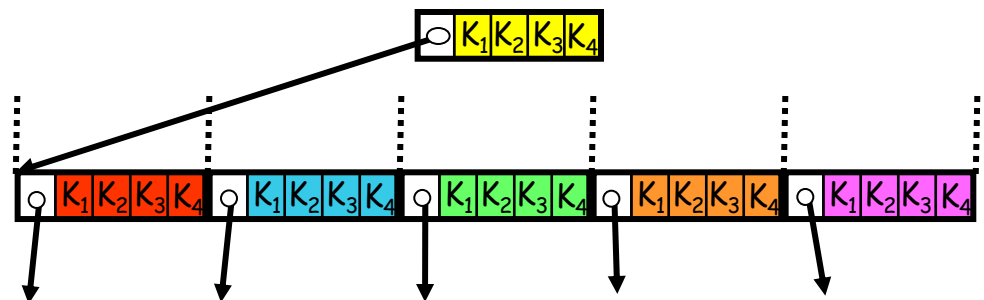
B-trees: < Pointers, > Fanout

[RR00]

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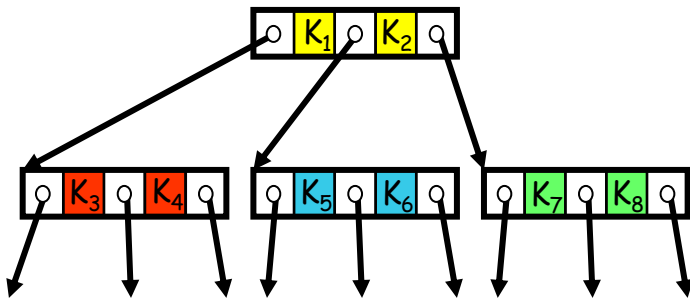
B+ Trees   CSB+ Trees

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

Data Placement: Summary

- Smart data placement increases spatial locality
- 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 the rest of misses?

- Idea: **hide latencies using** *prefetching*

- Prefetching enabled by

  - Non-blocking cache technology

  - Prefetch assembly instructions

    - **SGI R10000, Alpha 21264, Intel Pentium4**

```
pref 0(r2)
pref 4(r7)
pref 0(r3)
pref 8(r9)
```

CPU

L1 Cache

L2/L3 Cache

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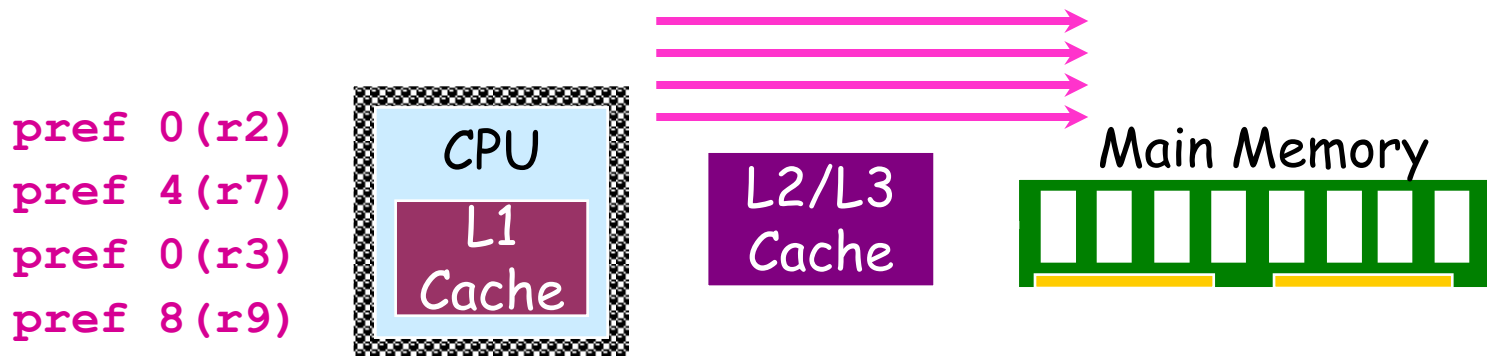
Prefetching hides cache miss latency
Efficiently used in pointer-chasing lookups!

> Prefetching B⁺-trees

[CGM01]

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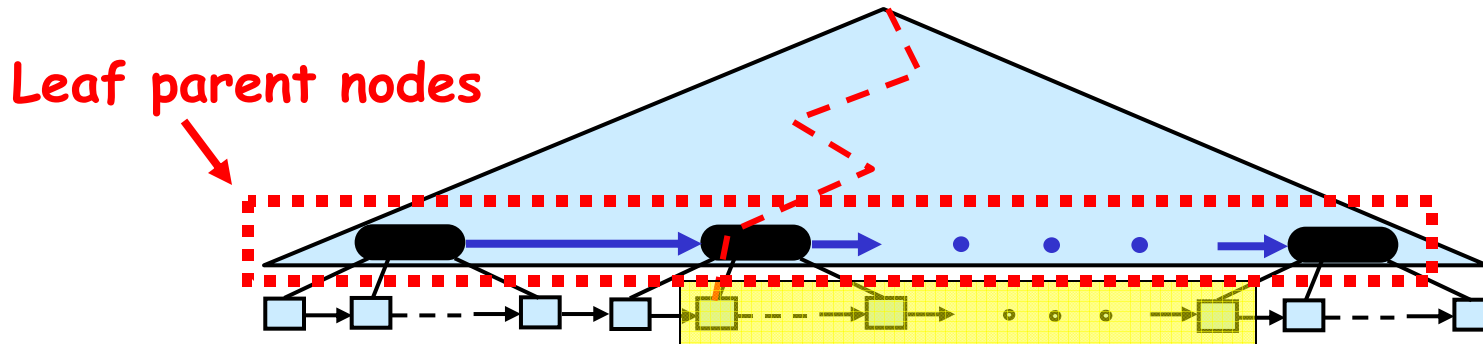- Prefetch all lines of a node before searching it

Cache miss

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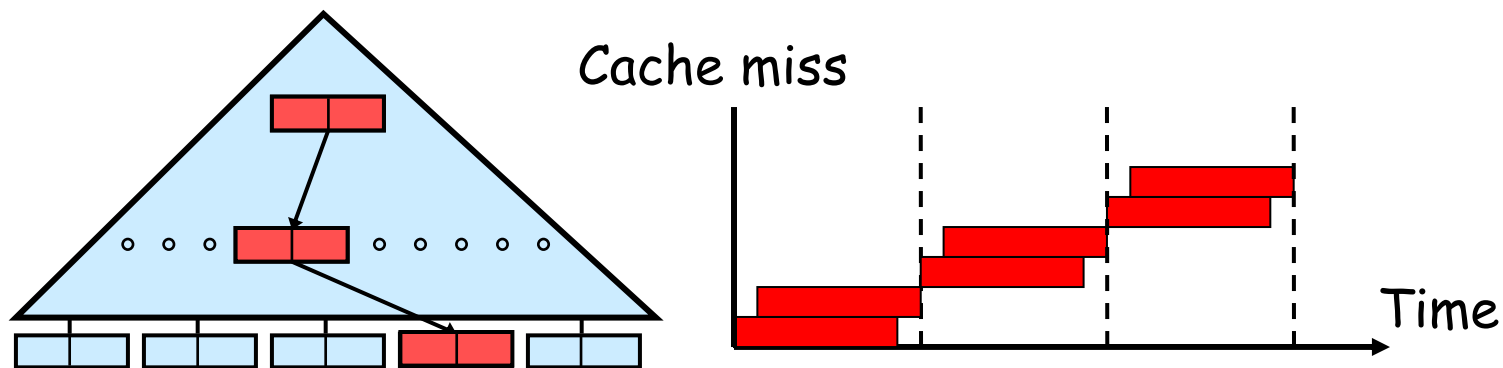
- **Cost to access a node only increases slightly**
- Much shallower trees, no changes required

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

>> Prefetching B+-trees

[CGM01,CGM02]

Leaf parent nodes

- Goal: faster range scan
  - Leaf parent nodes contain addresses of all leaves
  - Link leaf parent nodes together
  - Use this structure for prefetching leaf nodes

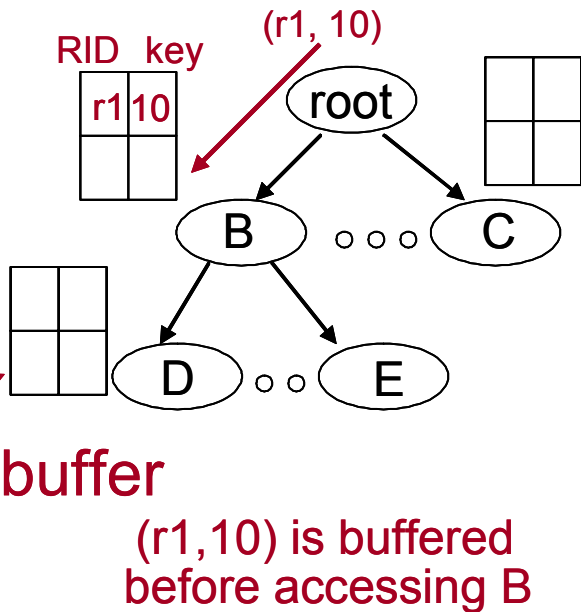- Fractal: Embed cache-aware trees in disk nodes

pB+-trees: 8X speedup over B+-trees
Fractal pB+-trees: 80% faster in-mem search

Bulk lookups

- 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)

RID   key

r1 | 10

(r1, 10)

root

B   ∘∘∘   C

D   ∘∘   E

buffer

(r1,10) is buffered
before accessing B

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 B-Tree cache performance [GL01]

Lots more to be done in the 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

Databases
@Carnegie Mellon   © A. Ailamaki 2004-06   *The* UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

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

The UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Query processing: directions

- 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]

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

X3	X2	X1	X0
	Y3		Y2		Y1		Y0

| OP | OP | OP | OP |

| X3 OP Y3 | X2 OP Y2 | X1 OP Y1 | X0 OP Y0 |
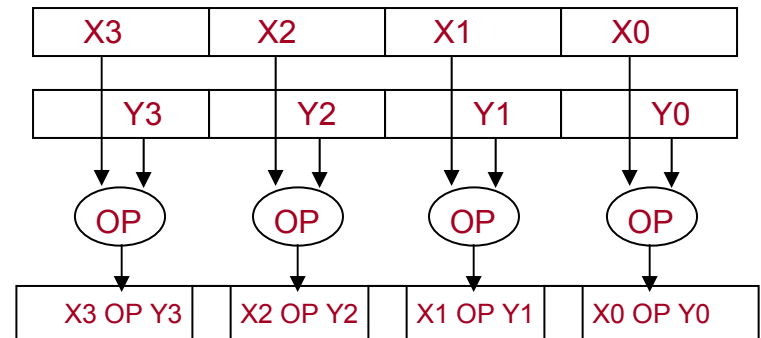
- Assume data stored columnwise as contiguous
  array of fixed-length numeric values (e.g., PAX)

- *Scan example:*

8	12	6	5
x[n+3]	x[n+2]	x[n+1]	x[n]
10	10	10	10

if x[n] > 10

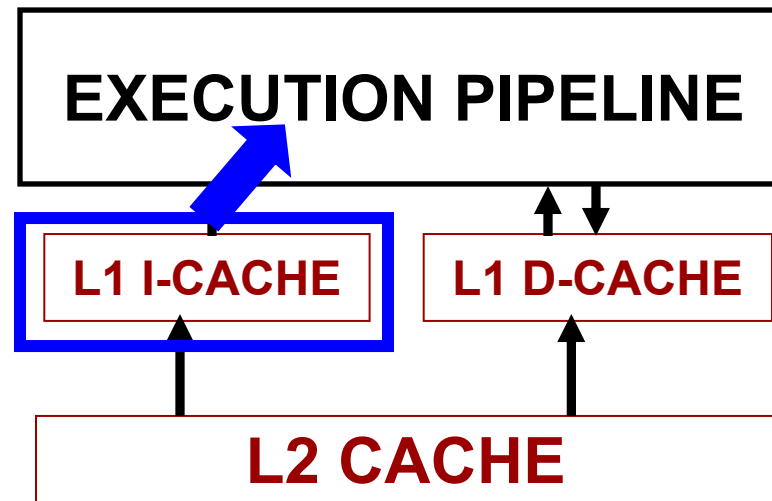result[pos++] = x[n]

produce bitmap
vector with 4
comparison results
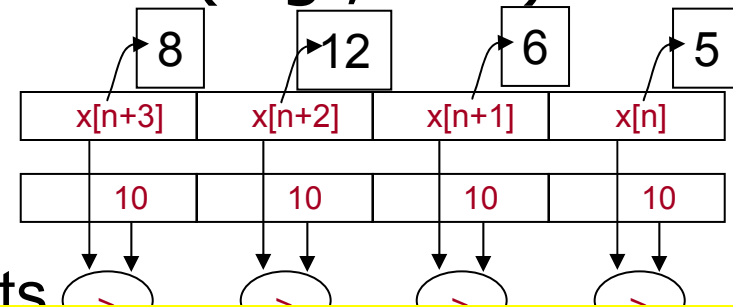
Superlinear speedup to # of parallelism
Need to rewrite code to use SIMD

Instruction-Related Stalls

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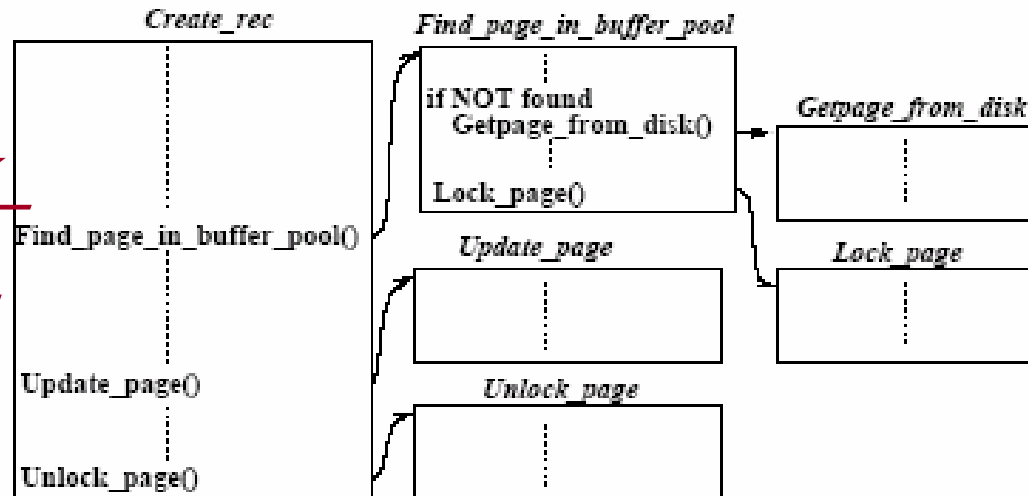- Importance of instruction cache: In critical path!

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
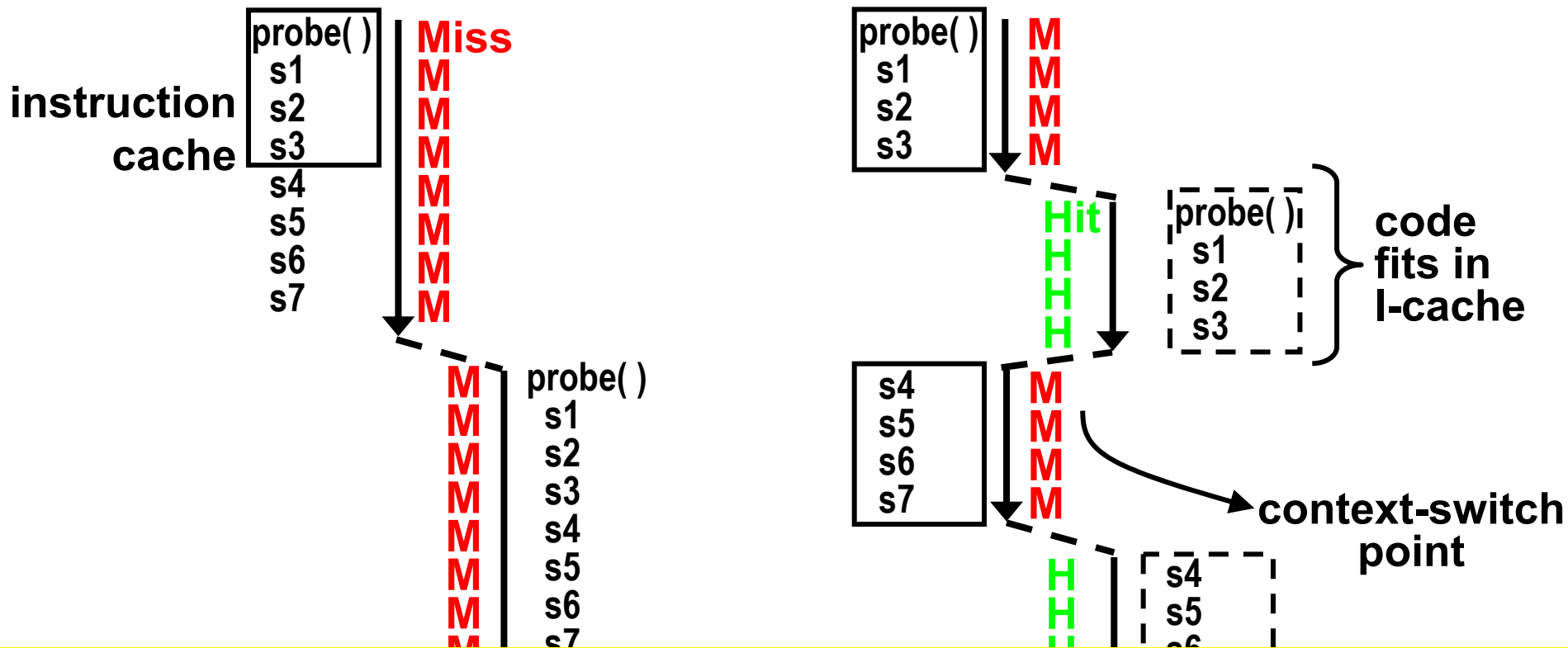

- Experiments: Shore on SimpleScalar Simulator
  - Running Wisconsin Benchmark

Beneficial for predictable DSS streams

STEPS: Cache-Resident OLTP

[HA04]

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

instruction cache

```
probe( )
 s1
 s2
 s3
 s4
 s5
 s6
 s7
```
Miss
M
M
M
M
M
M
M
M

M
M
M
M
M
M
M
M
```
probe( )
 s1
 s2
 s3
 s4
 s5
 s6
 s7
```

```
probe( )
 s1
 s2
 s3
```
M
M
M
M

Hit
H
H
H
H
```
probe( )
 s1
 s2
 s3
```
code fits in I-cache

```
s4
s5
s6
s7
```
M
M
M
M
M

context-switch point

H
H
```
s4
s5
s6
```

- **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
- 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
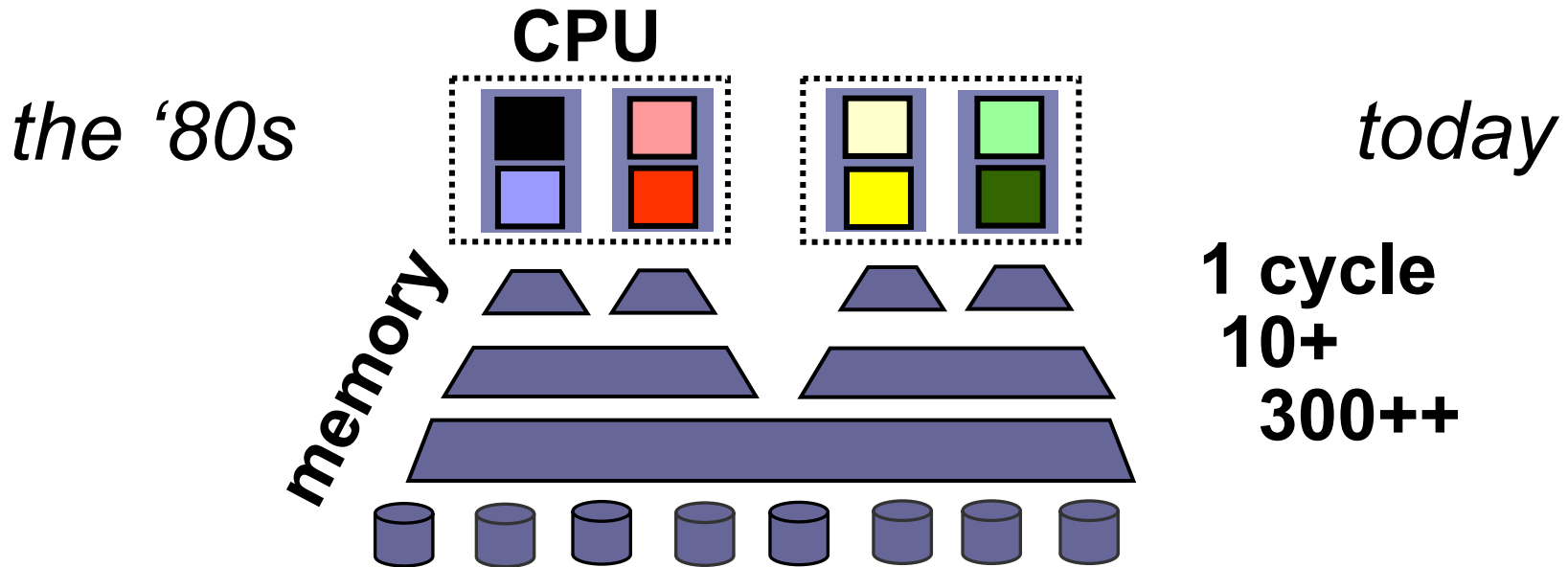 - Chip multiprocessor DB architectures

- QUERY co-PROCESSING: NETWORK PROCESSORS

- QUERY co-PROCESSING: GRAPHICS PROCESSORS
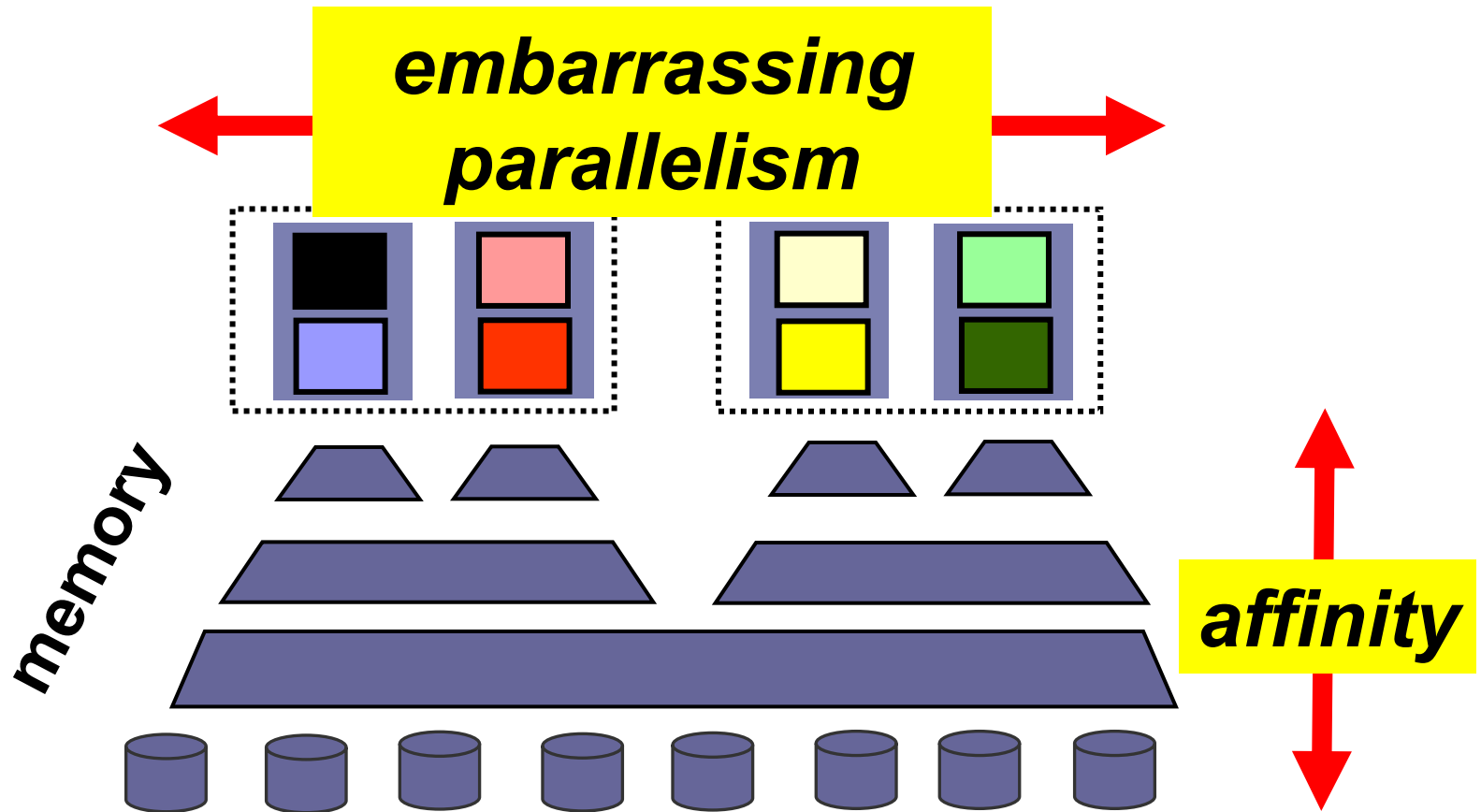
- CONCLUSIONS AND FUTURE DIRECTIONS

Evolution of hardware design

Past: HW = CPU+Memory+Disk

CPU

the '80s

today

memory

1 cycle
10+
300++

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

Main Thread

Helper Thread

Cache Memory

Main Memory
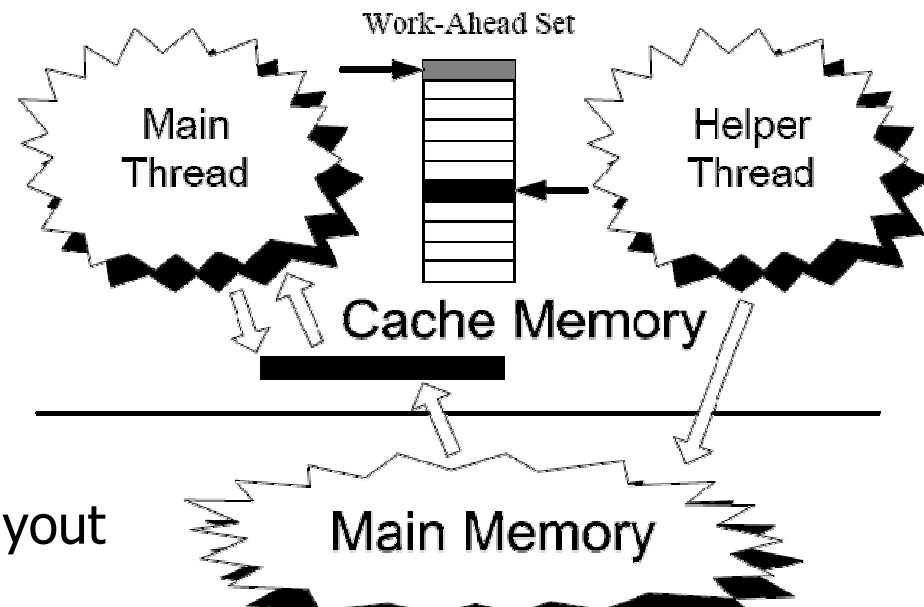
Work-ahead-set best with no data movement
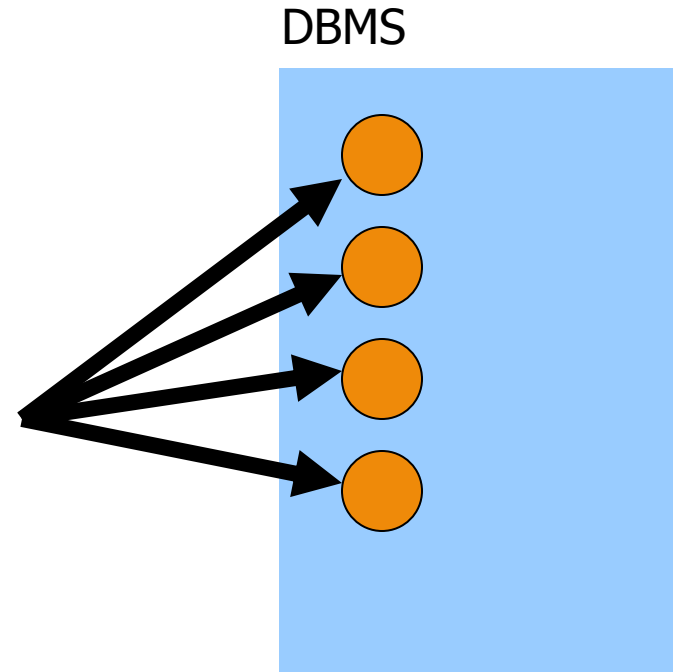
Parallelizing transactions

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;
}
```
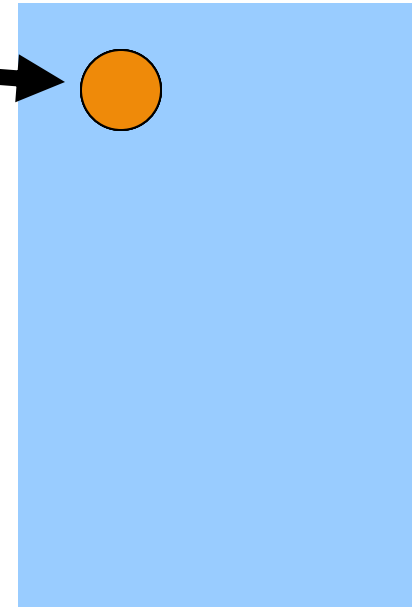
- **Intra-query parallelism**
  - Used for long-running queries (decision support)
  - Does not work for short queries
- **Short queries dominate in OLTP workloads**

Parallelizing transactions

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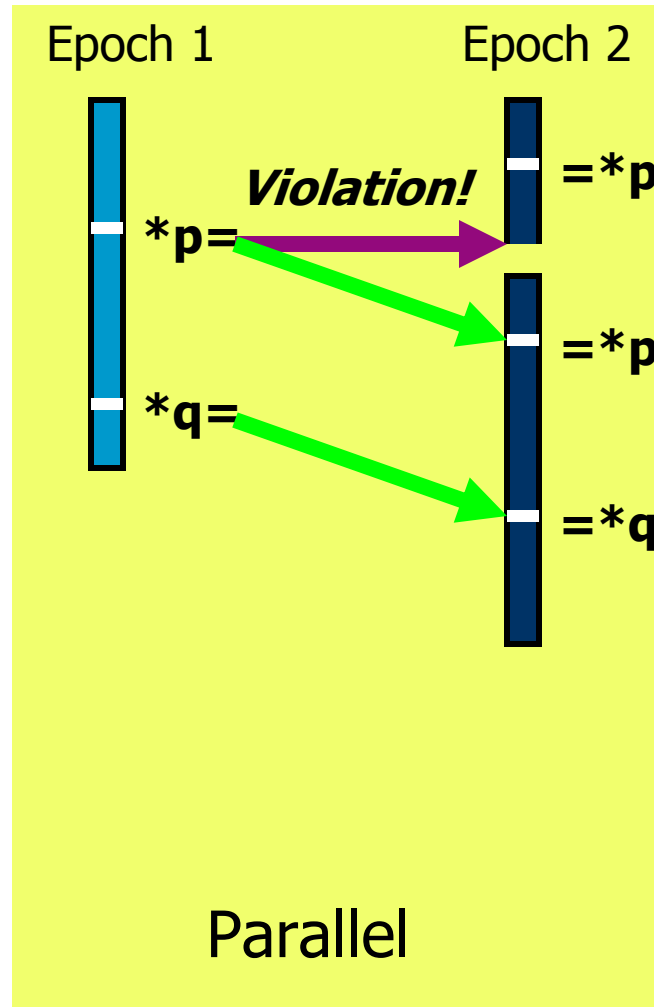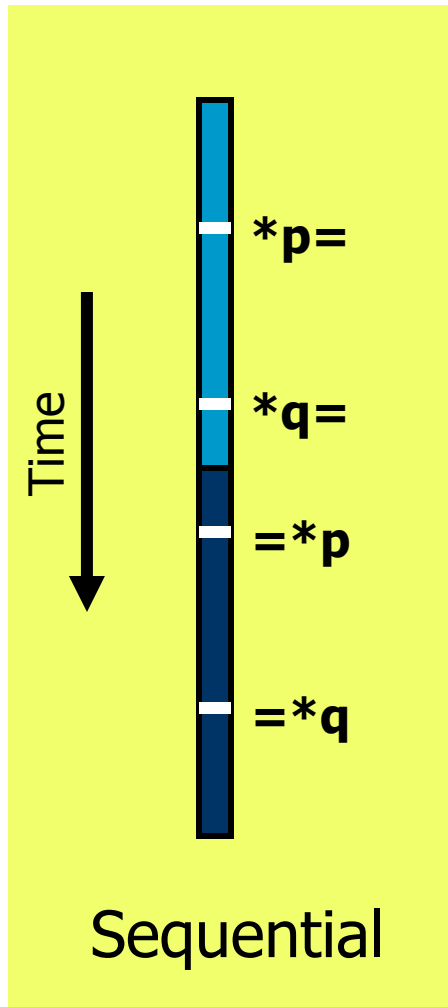
- Intra-transaction parallelism
  - Each thread spans multiple queries
- Hard to add to existing systems!

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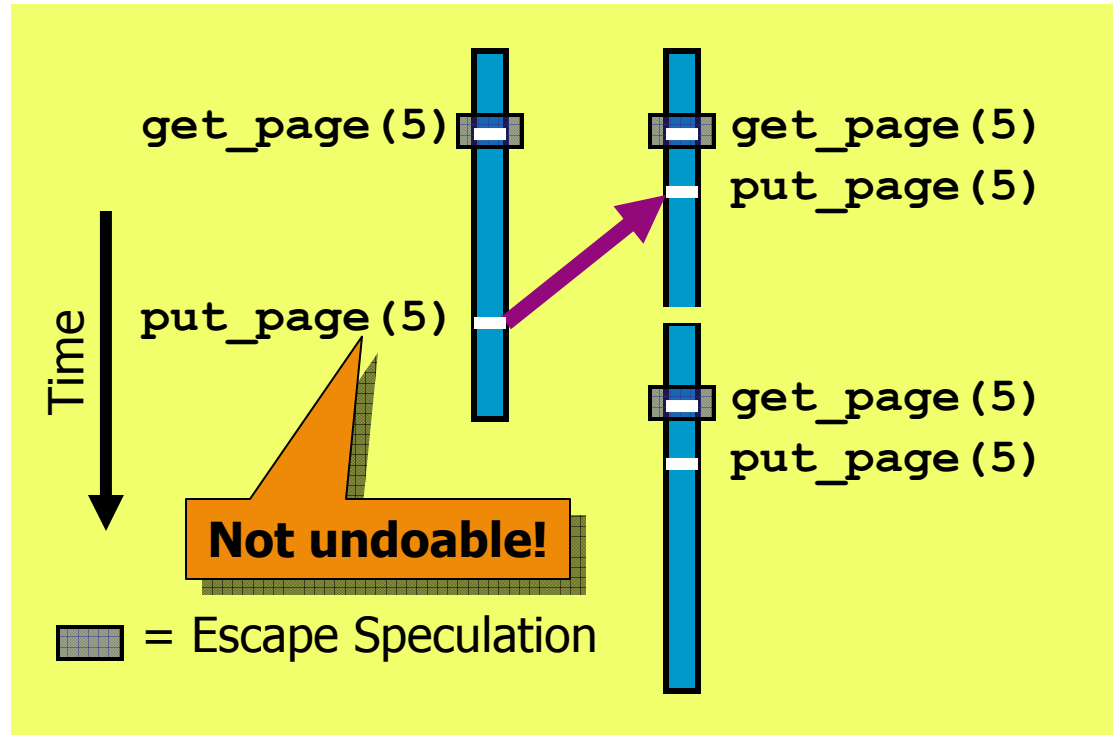
Thread Level Speculation (TLS)
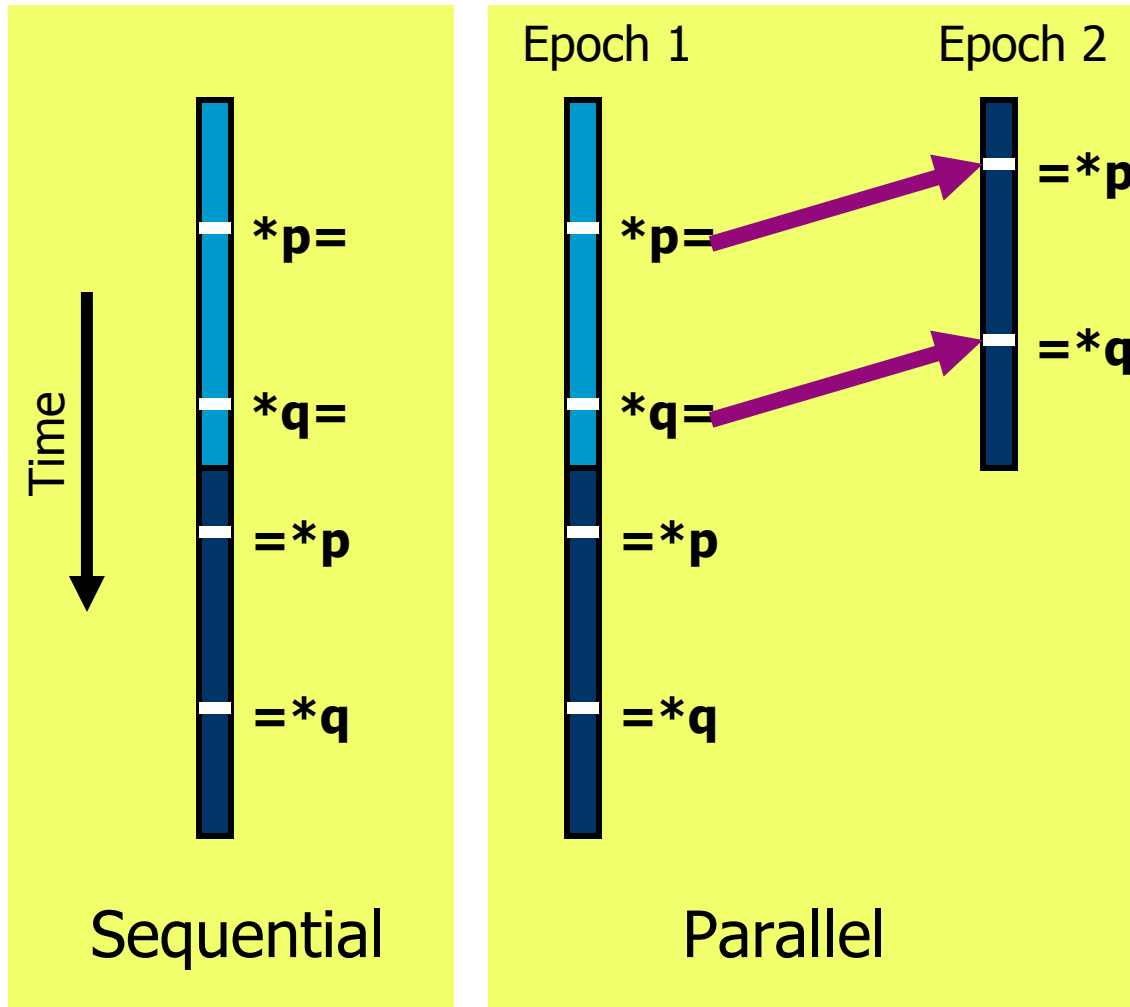
Thread Level Speculation (TLS)

- Use *epochs*

- *Detect* violations
- *Restart* to recover
- *Buffer* state

- Worst case:
  - Sequential
- Best case:
  - Fully parallel

Data dependences limit performance

Example: Buffer Pool Management

CPU

```
get_page(5)
put_page(5)
```

Buffer Pool

ref: 0

Time

```
get_page(5)
put_page(5)
```

```
get_page(5)
put_page(5)
```

```
get_page(5)
put_page(5)
```

Not undoable!

= Escape Speculation

Example: Buffer Pool Management

CPU

`get_page(5)`
`put_page(5)`

Buffer Pool

ref: 0

Time

`get_page(5)`   `get_page(5)`

`put_page(5)`

`put_page(5)`

▦ = Escape Speculation
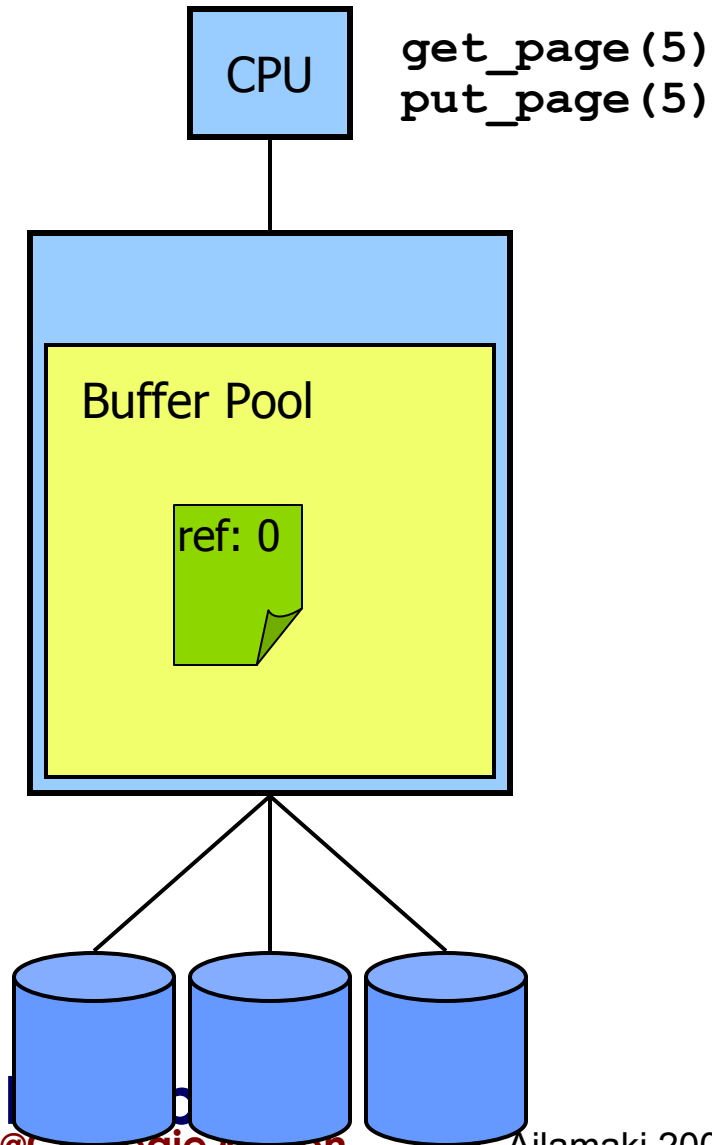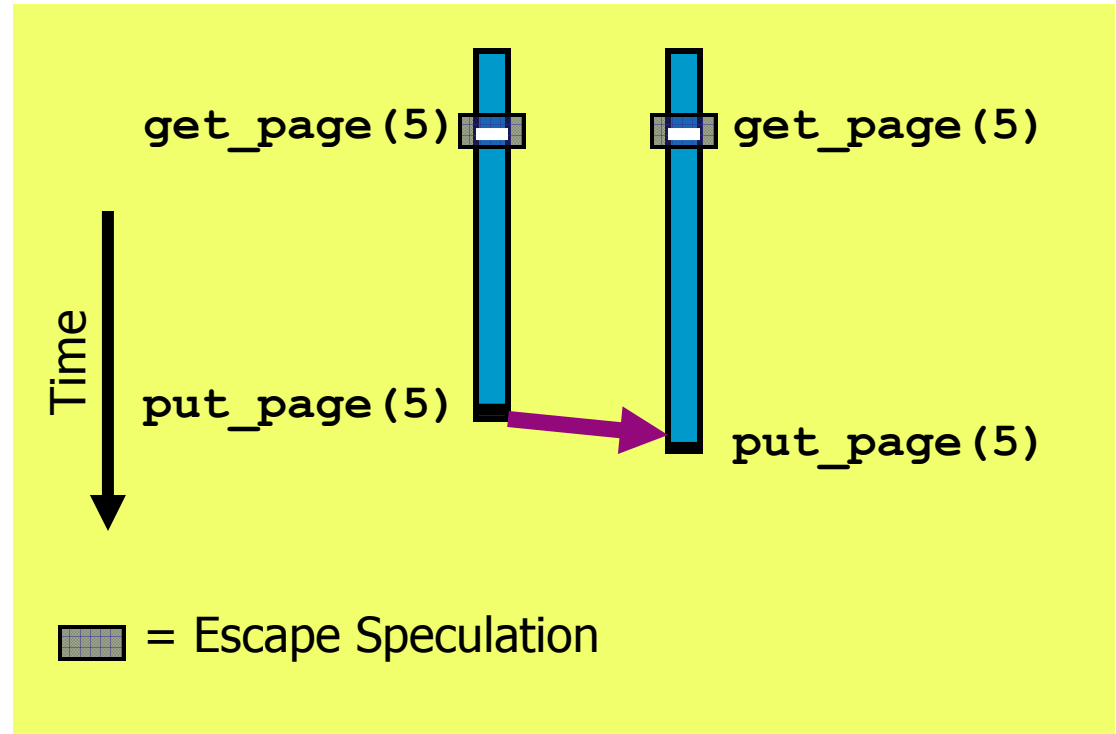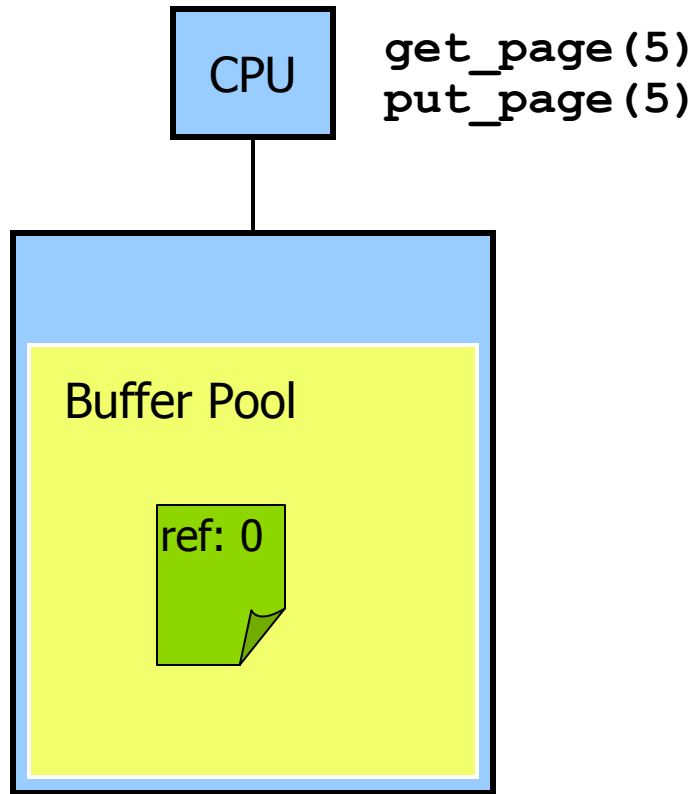
- *Delay* `put_page` until end of epoch
  - *Avoid dependence*

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**
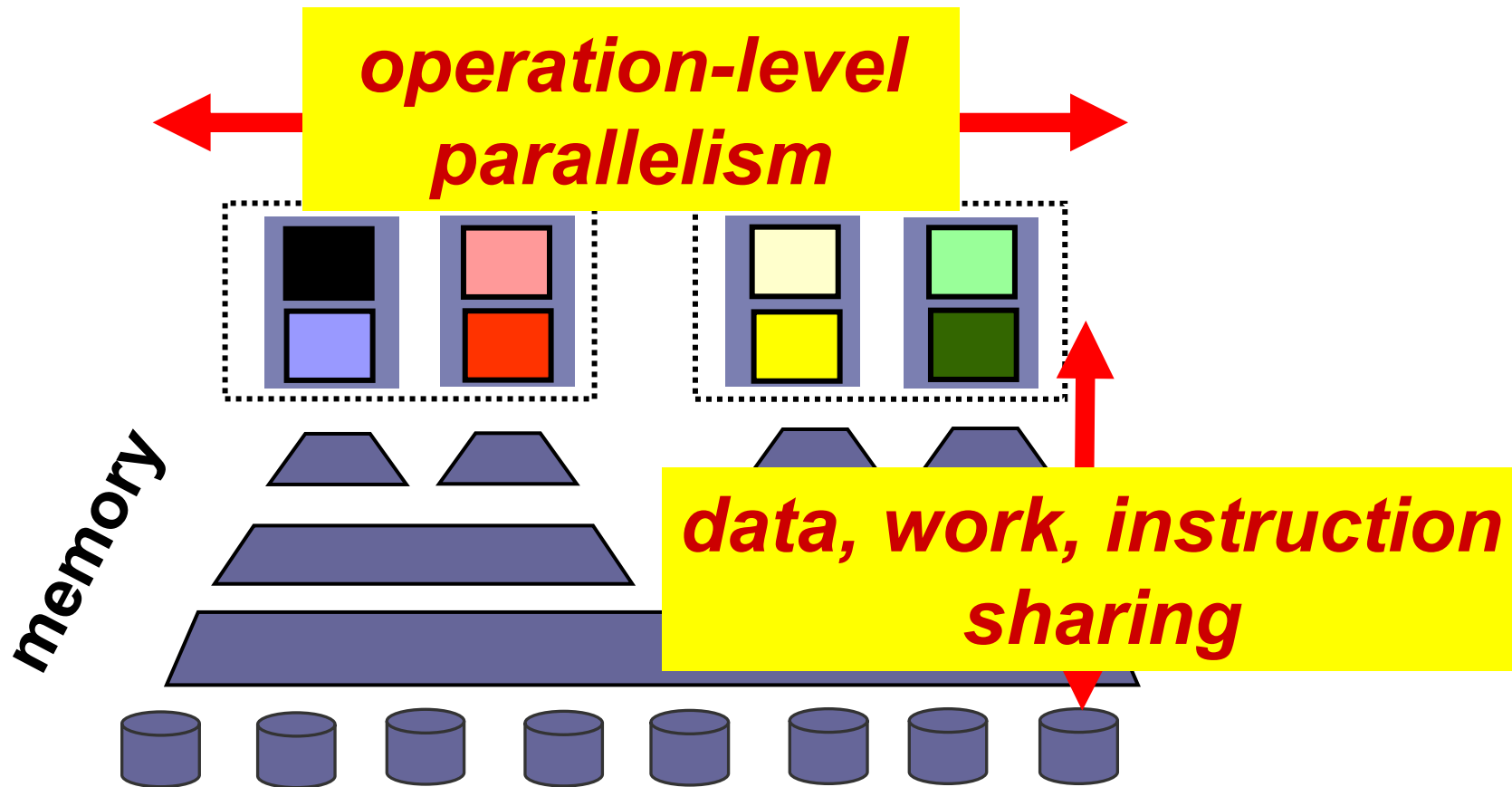 - Buffer pool, memory, and cursor *allocation*
- **Traditional parallelization**
  - Memory allocation, cursor pool, error checks, false sharing

2x lower latency with 4 CPUs
Useful for non-TLS parallelism as well

DBMS parallelism and memory affinity

operation-level parallelism

memory

data, work, instruction sharing

StagedDB design addresses shortcomings

StagedDB software design

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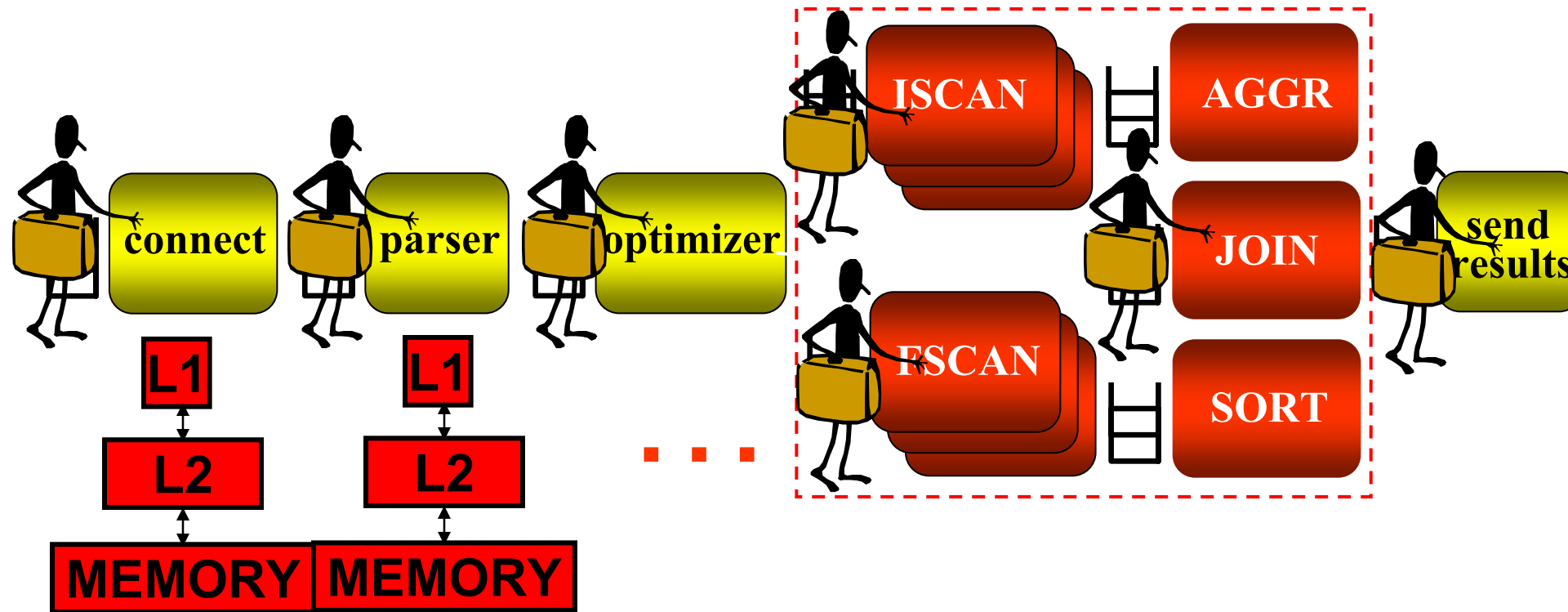
Optimize instruction/data cache locality
Naturally enable multi-query processing
Highly scalable, fault-tolerant, trustworthy

QPipe: operation-level parallelism

[HSA05]

packet dispatcher

Q Q Q Q

μEngine-A

conventional design

query plans

A

J

S S

thread pool

μEngine-S

μEngine-J

read

read

write
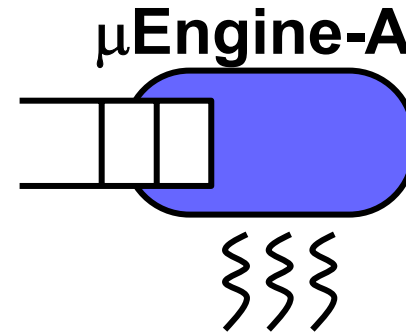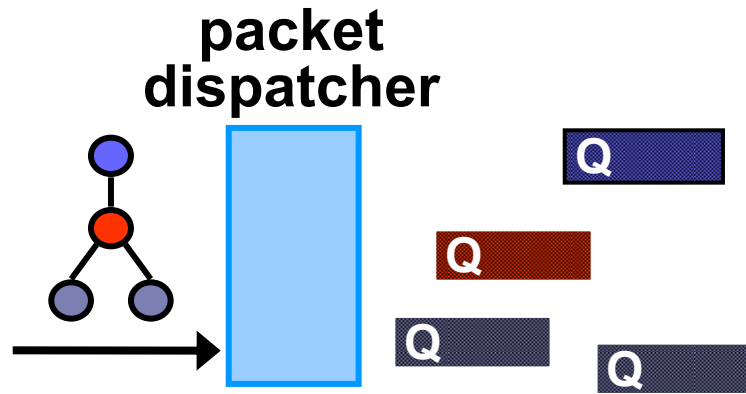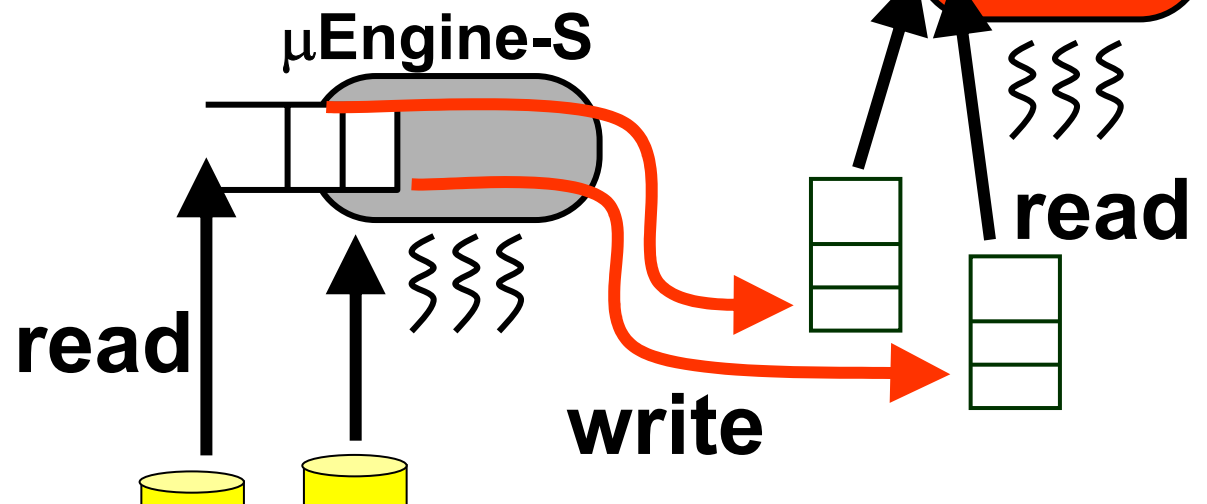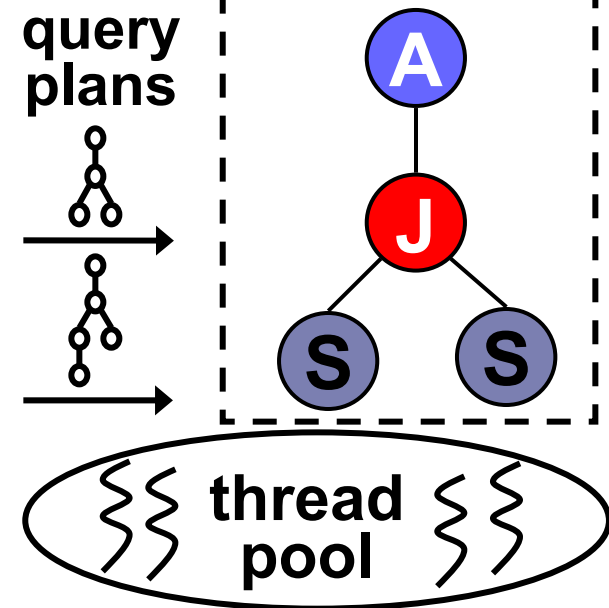
Stable throughput as #users increases

Future: NUCA hierarchy abstraction

P   P

P

Private
caches

P

Large shared
on-chip cache

P

P   P
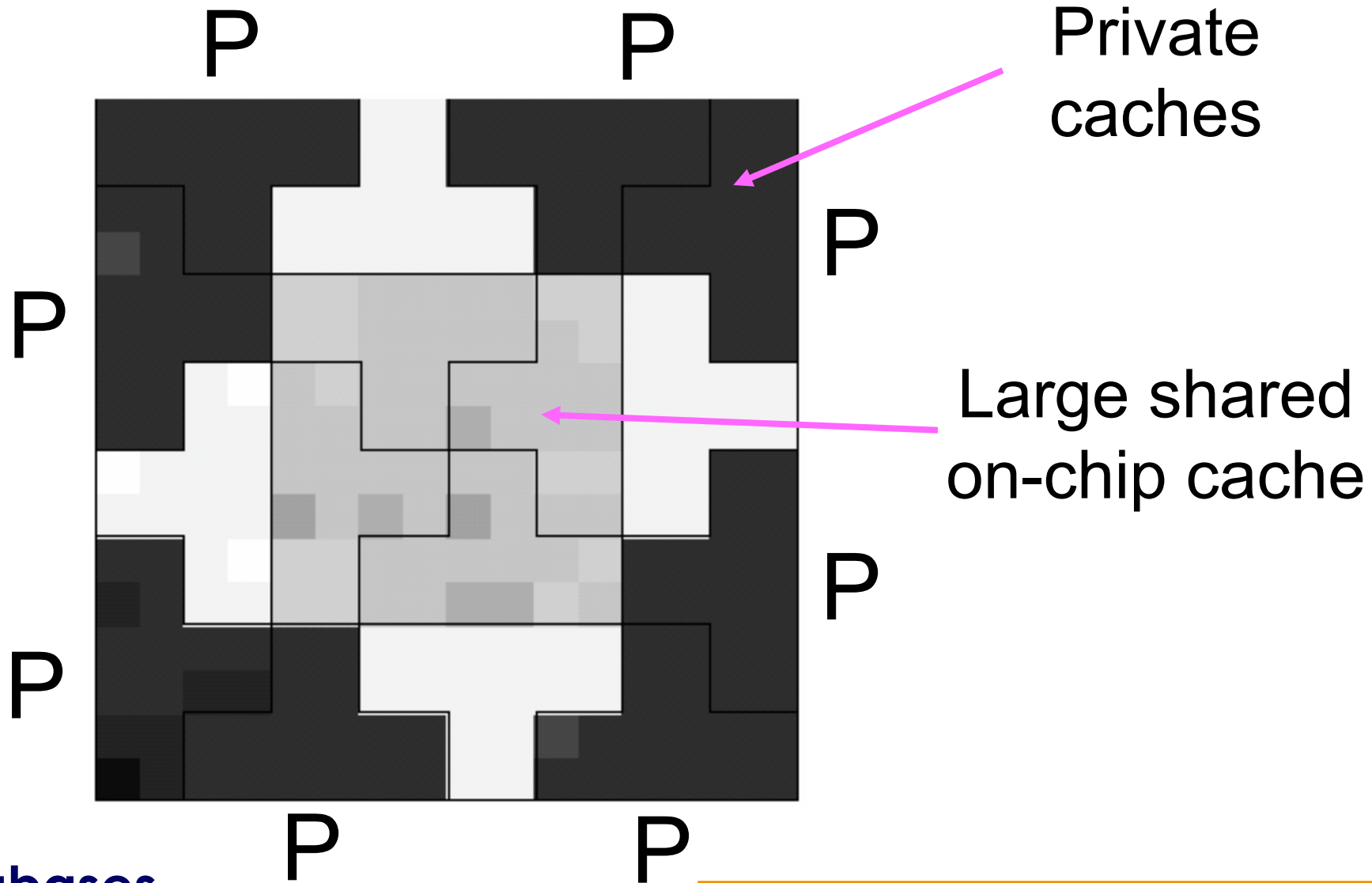
Data movement on CMP hierarchy*

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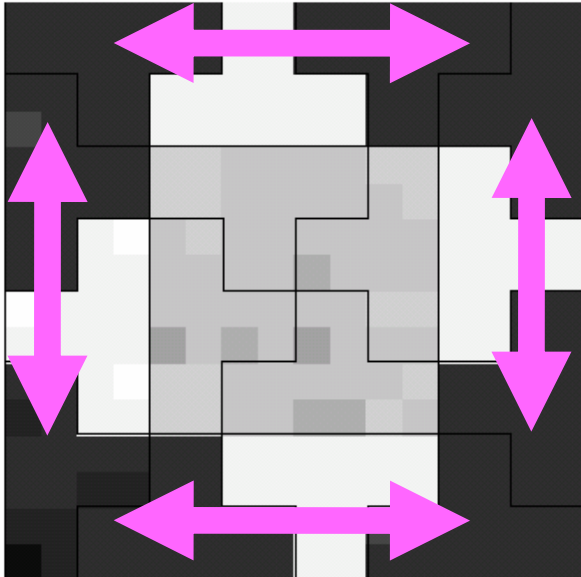
OLTP
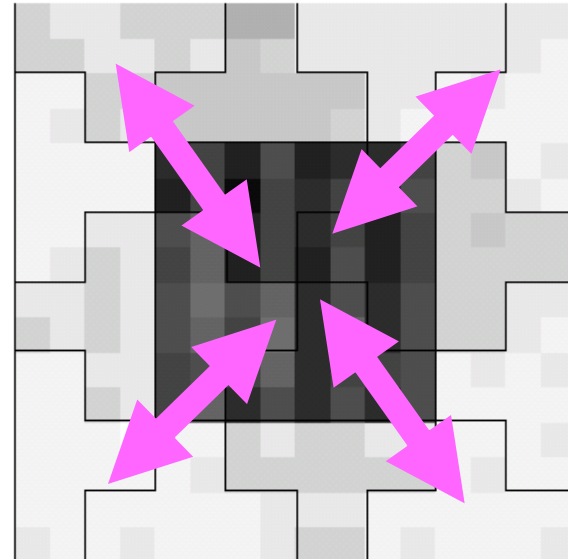
- Traditional DBMS: shared information in middle
- StagedDB: exposed data movement

data from Beckmann&Wood, Micro04

StagedCMP: StagedDB on Multicore

- μEngines run independently on cores
- Dispatcher routes incoming tasks to cores

Tradeoff:
- Work sharing at each uEngine
- Load of each uEngine
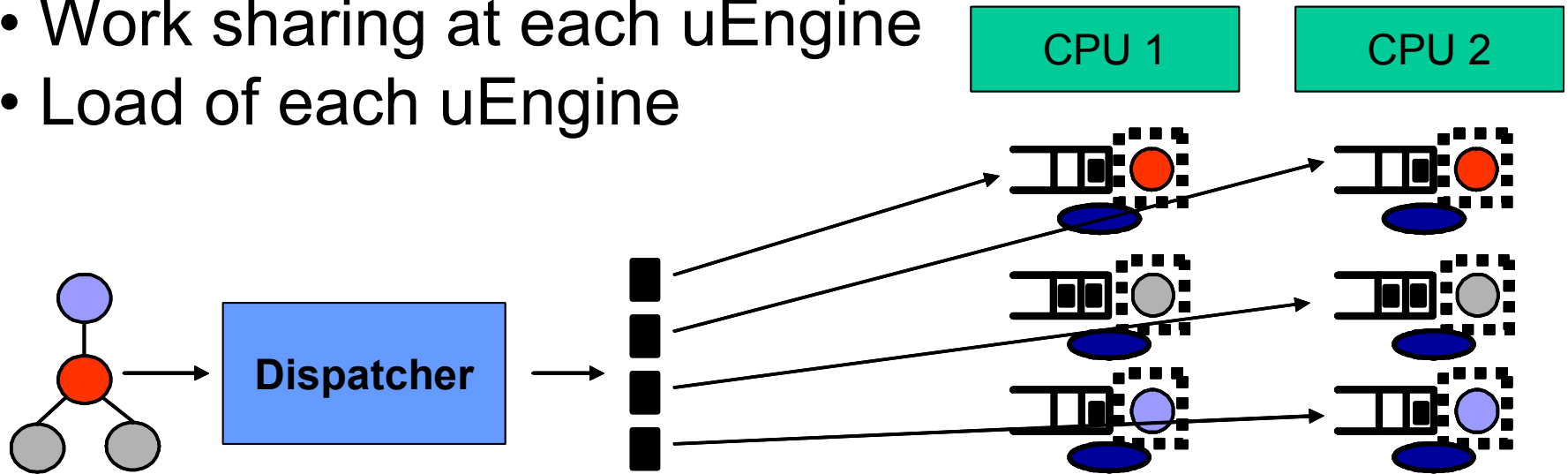
Potential: better work sharing, load balance on CMP

Summary: data mgmt on SMT/CMP

- Work-ahead sets using helper threads
  - Use threads for prefetching

- 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 processors

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

CPU
(3 GHz)

System Memory
(2 GB)

PCI-X Bus
(528 MB/s)

Network Processor
(600 MHz)

NP Memory
(1 GB)

Conventional

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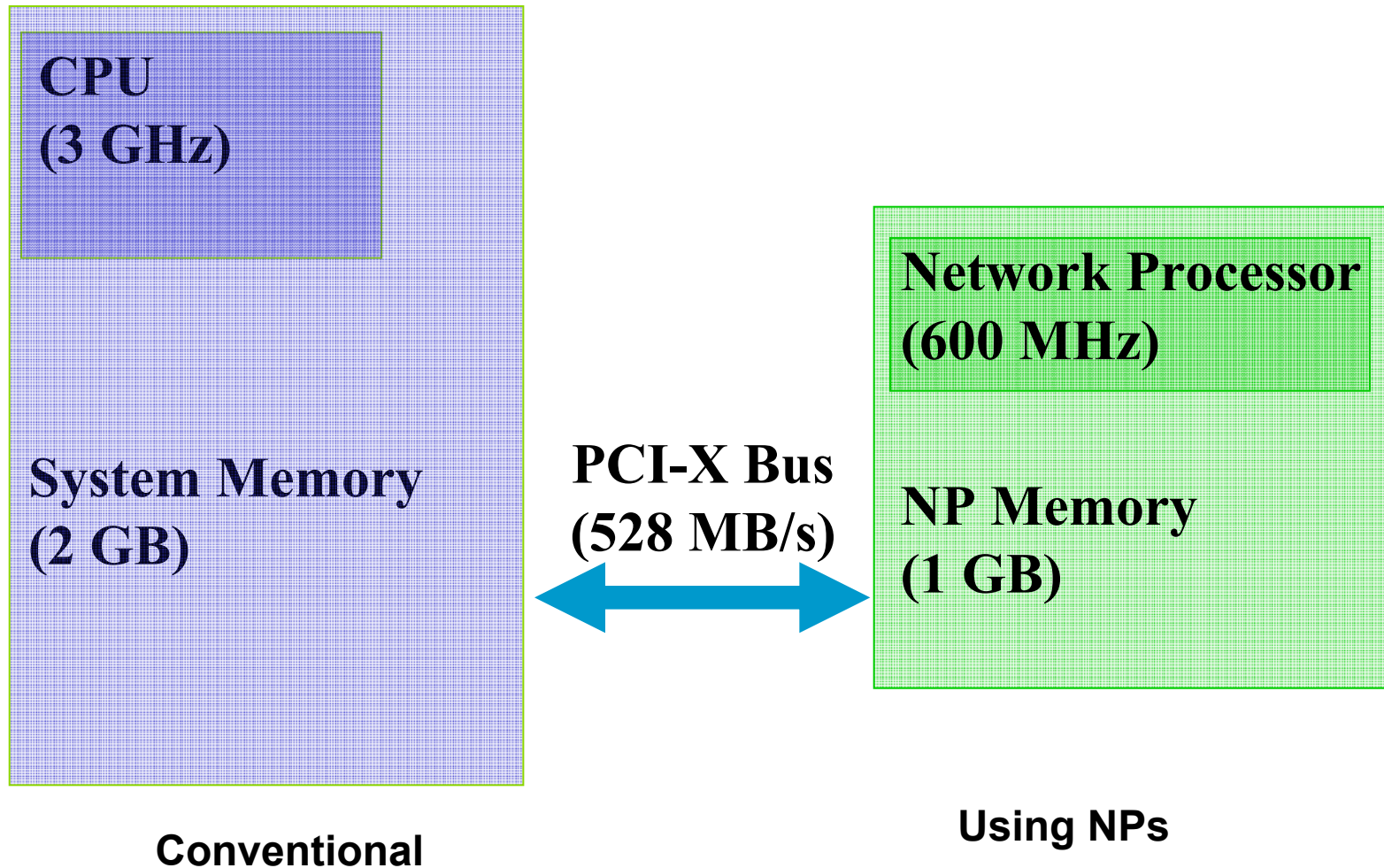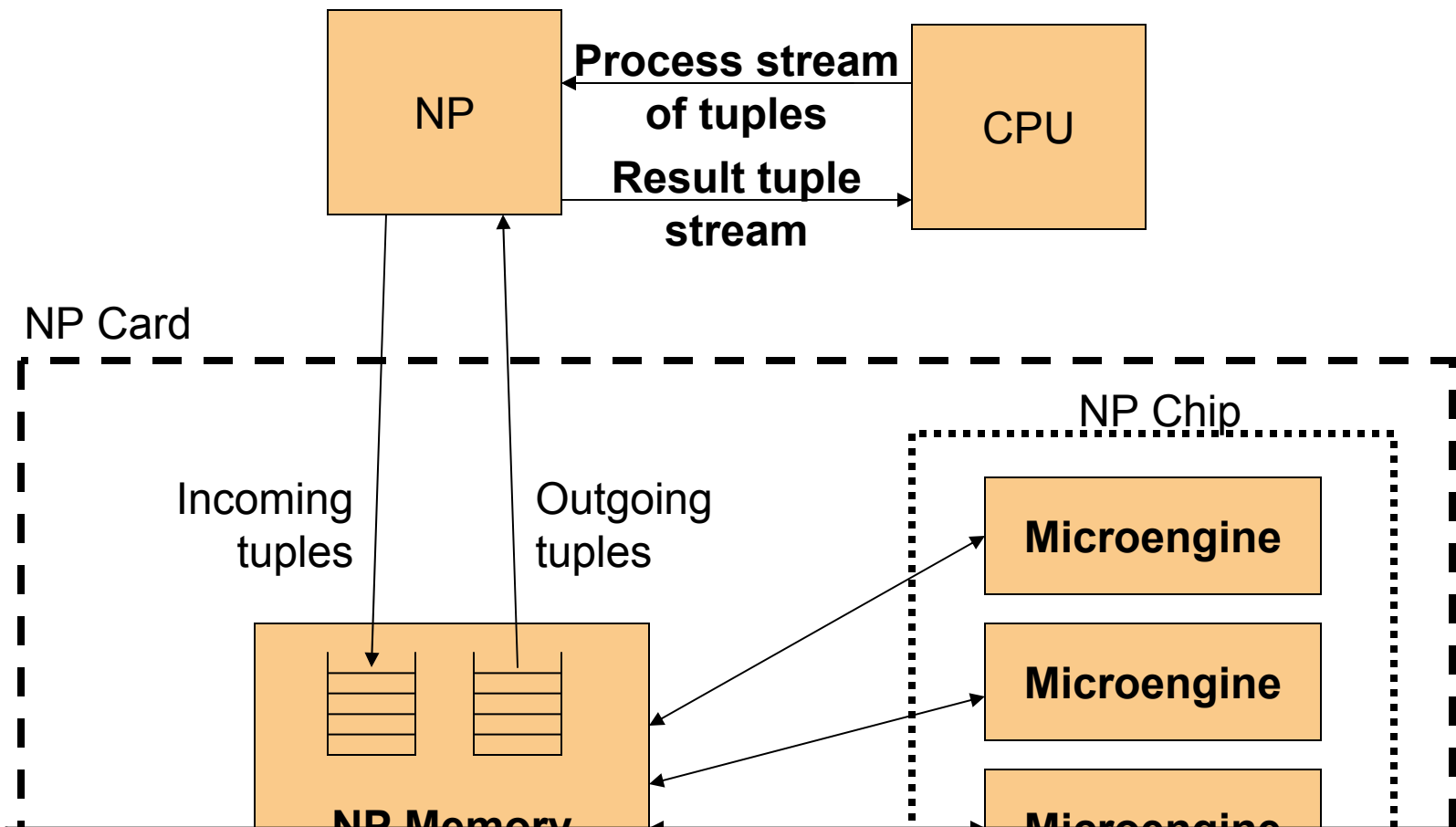
	Intel IXP2400	Intel IXP2805
Microengines	8	16
Thread contexts	64	128
Clock rate	600 MHz	1500 MHz
Transistor count	~60 M	~110M
Power	< 16 W	< 30 W
Memory (DRAM)	1x DDR	3x Rambus

TLP opportunity in DB operators

- Sequential or index scan
  - Fetch tuples in parallel

Name	ID	Grade
Smith	0	B
Chen	1	A
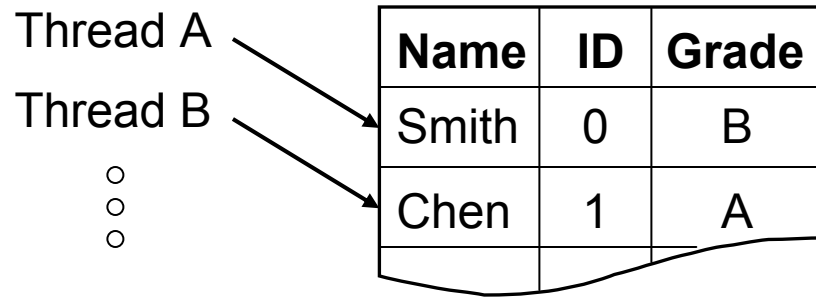
Thread A

Thread B

- Hash join
  - Probe tuples in parallel

Name	ID	Grade
Smith	0	B
Chen	1	A
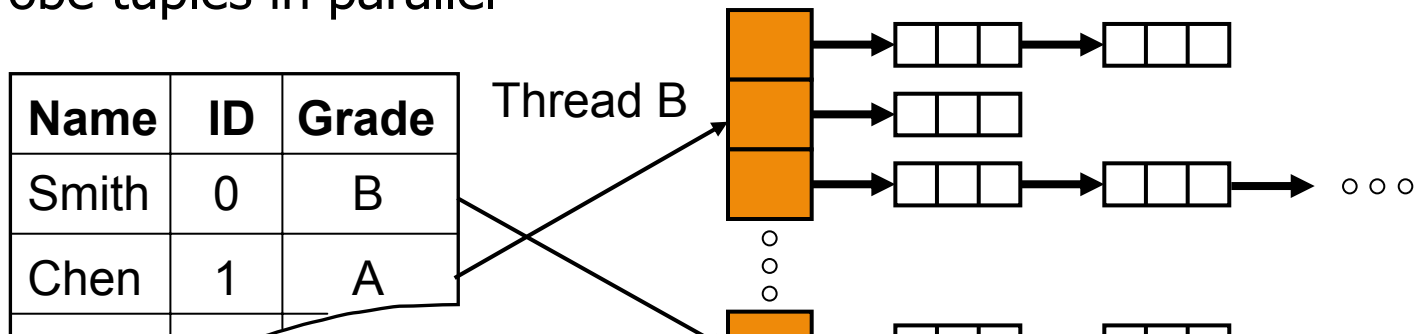
Thread B

Hardware thread support helps expose parallelism without significant overhead

Multi-threaded Core
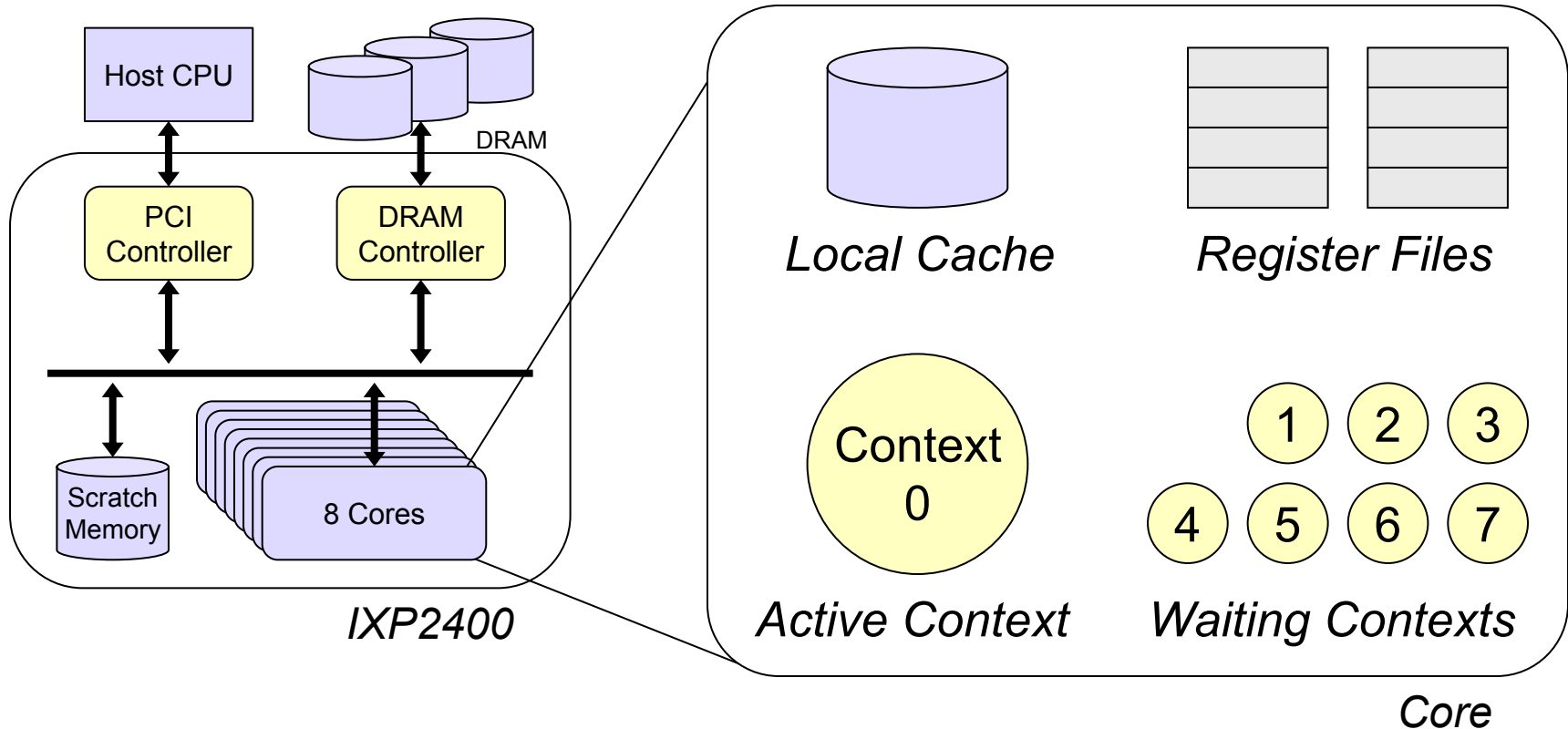
- Simple processing core
  - 5-stage, single-issue pipeline @ 600MHz, 2.5KB local cache
  - Switch contexts at programmer's discretion

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

Multithreading in Action
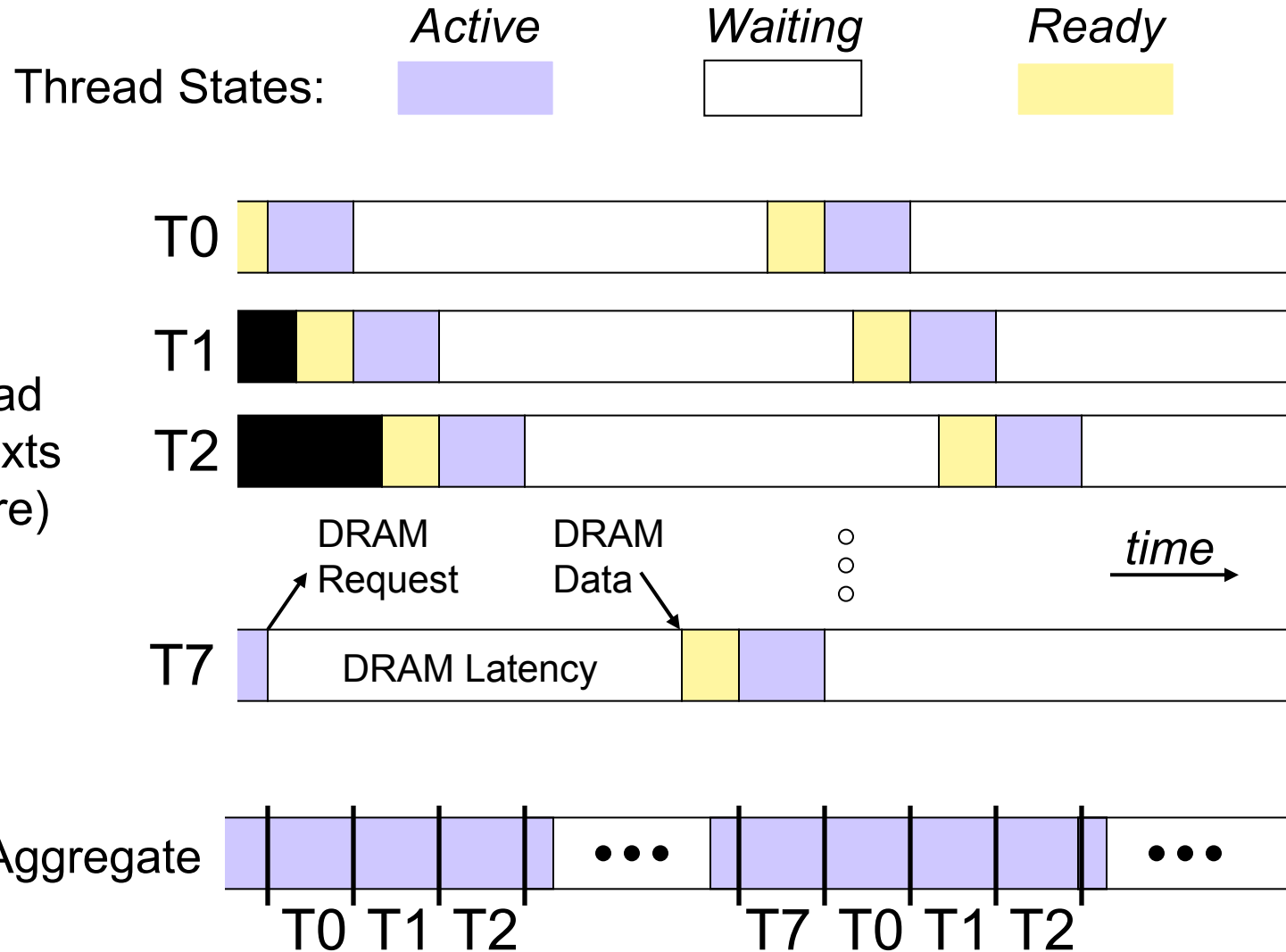
Sequential Scan Setup [GAH05]

- Use slotted page layout (8KB)

Page

0 1
2
3 4

Record header   • *Attributes*
                  •
                  •

n

n • • • 4 3 2 1 0

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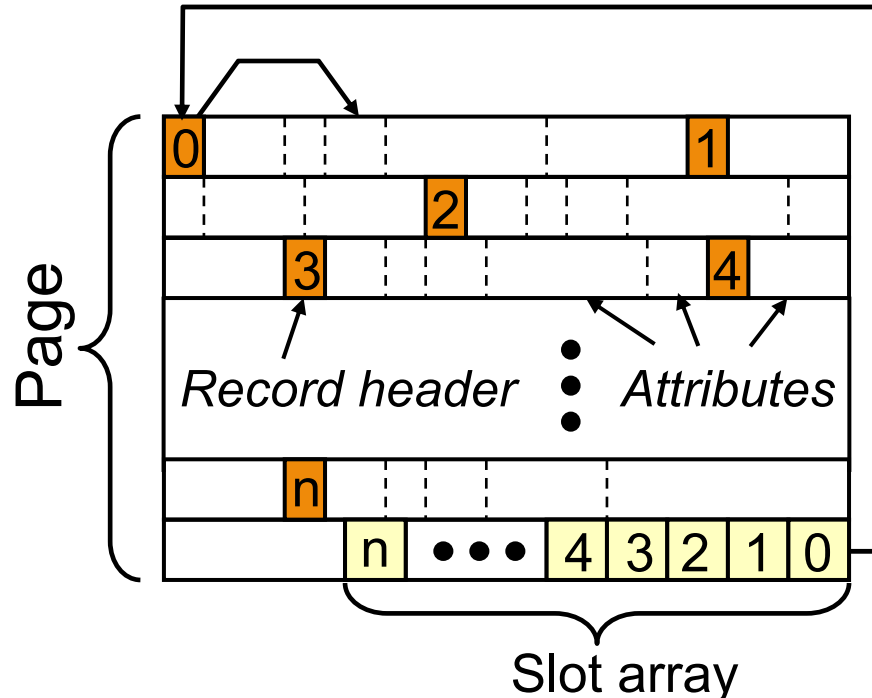
- Network processor implementation
  - Each page scanned by threads on one core
  - Overlap individual record access within core

Hash Join Setup

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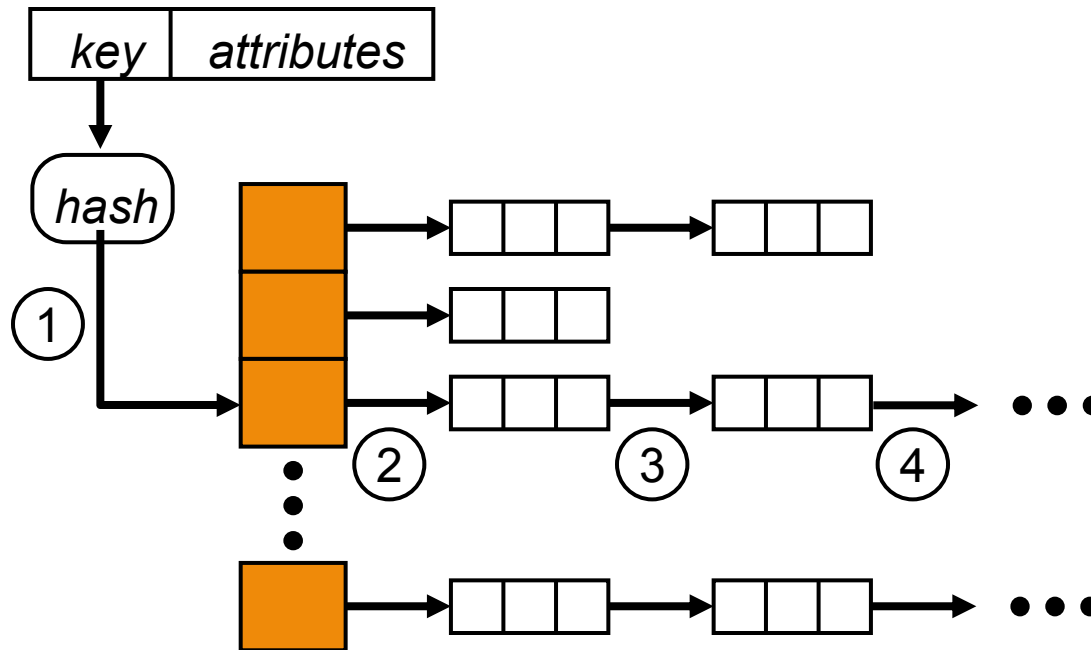
- Assign pages of outer relation to one core
  - Each thread context issues one probe
  - Overlap dependent accesses within core

Performance

- IXP2400 prototype card
  - 256MB PC2100 SDRAM
  - Separated from host CPU

- Pentium 4 Xeon 2.8GHz
  - 8KB L1D, 512KB L2
  - 3GB PC2100 SDRAM
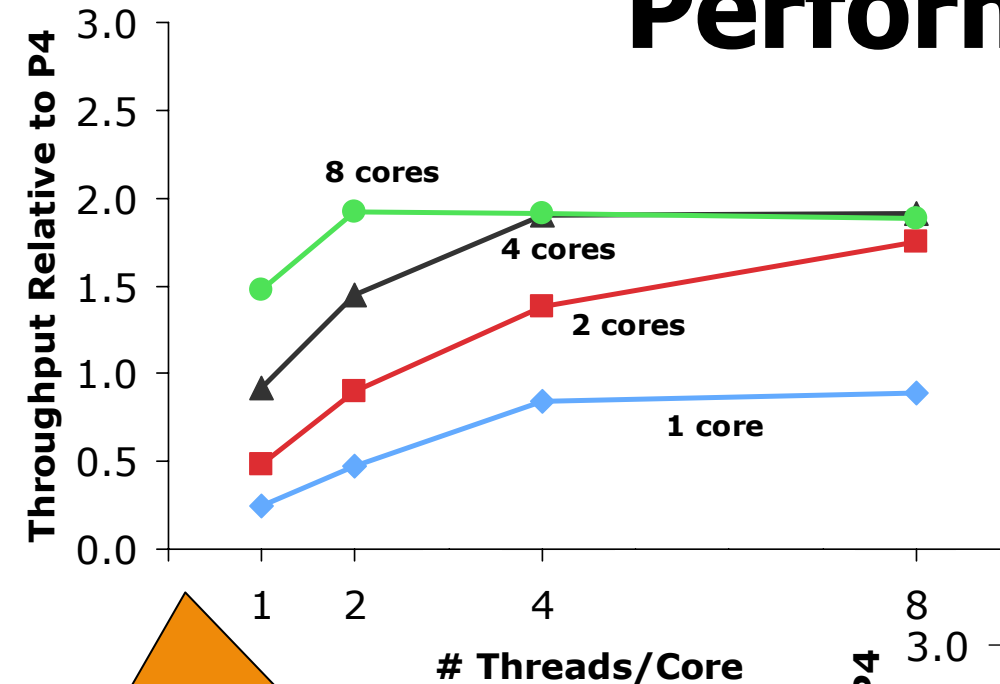
Sequential scan 250MB (*Lineitem*)

Hash join (*Lineitem, Orders*)
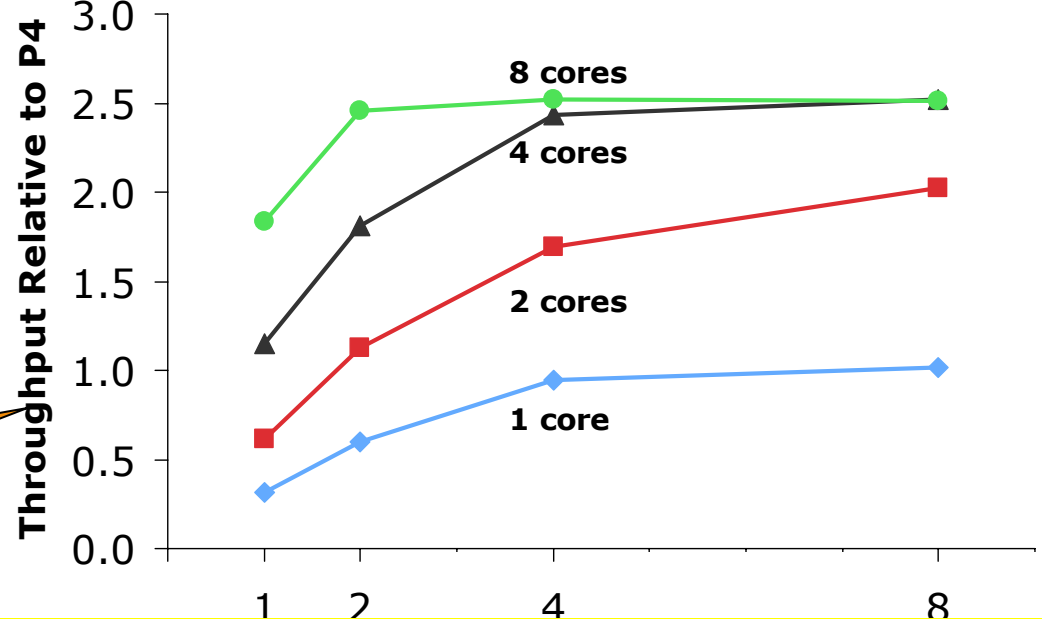
Performance limited by DRAM controller

Query Coprocessing with NPs

Front-end
- Query compilation
- Results reporting

Back-end
- Query execution
- Raw data access

CPU

Network Processors

Disk Arrays
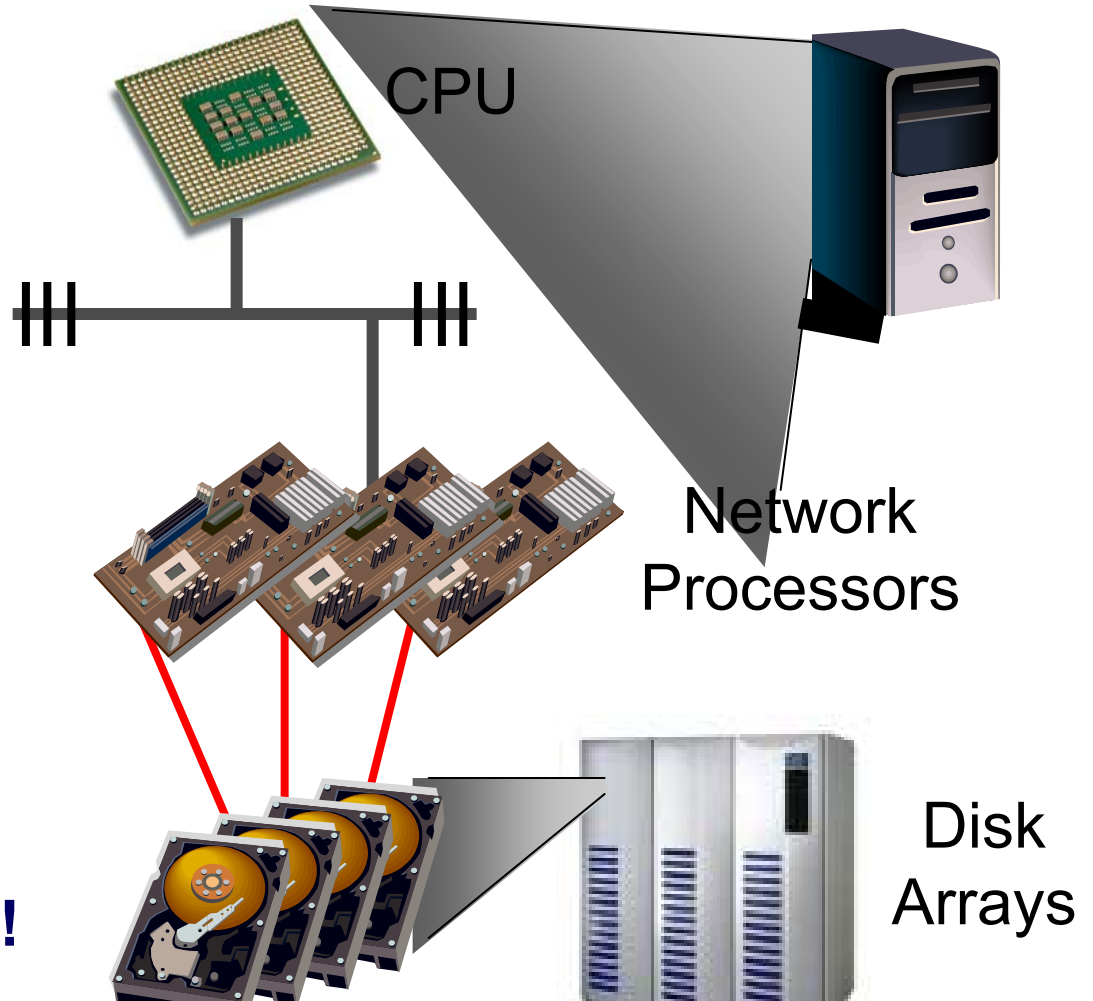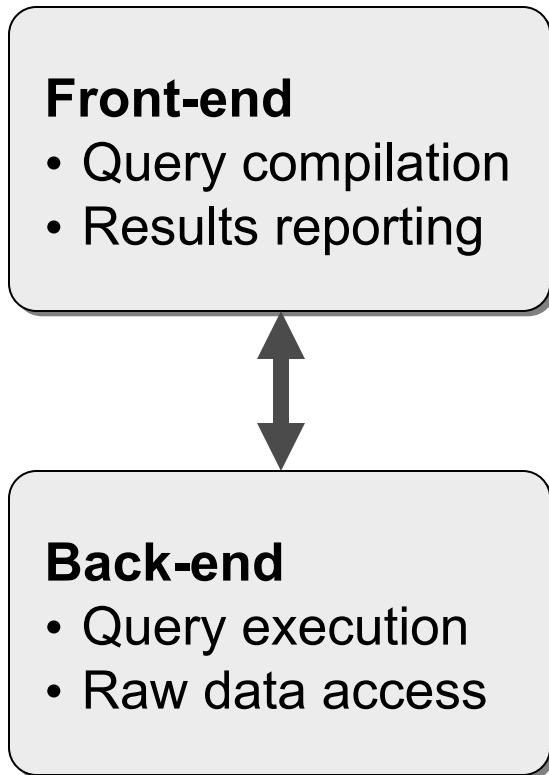
Use right 'tool' for job!

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

NEXT: QUERY co-PROCESSING: GRAPHICS PROCESSORS

- CONCLUSIONS AND FUTURE DIRECTIONS

References…
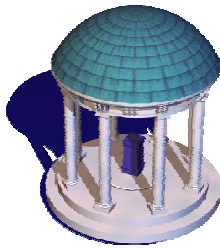
References
Where Does Time Go? (simulation only)

[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. Nowatzyk, 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
Instrustion 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. Hseih, 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. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. *International Symposium on Computer Architecture (ISCA). Vancouver, Canada, June 2000.*