# CACHE-CONSCIOUS DATABASES

## E0 261

Jayant Haritsa

Computer Science and Automation

Indian Institute of Science

# The story until mid-1990s ...

- Architecture folks were infatuated by memory architectures for scientific applications
  - Proof: Look at their benchmarks (LL loops, Spec, ...) !
- Database folks were besotted with optimizing disk performance for business applications
  - Proof: Look at their benchmarks (TPC-x) !

# The Big Thaw post 1995

- Architects started looking into the performance of business applications
  - Proof: Four papers in ISCA 98 in the “Machine Measurement” session
  - Focus is on workload characterization

# ISCA 1998

- Memory System Characterization of Commercial Workloads.
  - Barroso, Gharachorloo, Bugnion
- Performance Characterization of a Quad Pentium Pro SMP using OLTP Workloads.
  - Keeton, Patterson, He, Raphael, Baker
- Execution Characteristics of Desktop Applications on Windows NT.
  - Lee, Crowley, Baer, Anderson, Bershad
- An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors.
  - Lo, Barroso, Eggers, Gharachorloo, Levy, Parekh

# The Big Thaw post 1995 (contd)

- Database folks started investigating memory-resident databases
  - Proof: papers in VLDB/SIGMOD 99, 2000, 2001, including best papers in VLDB 99 (today's paper), 2001
  - Focus is on how to alter DB algorithms to suit current architectures

# Why The Thaw?

- Architects:  because business is where the money is and supercomputer industry was going bankrupt
- DB: because DRAM is cheap, really cheap!

- While processor manufacturers routinely allow experimental evidence to be published, the same is not true of DBMS companies
  - the situation holds even today!

# The Big Move

- From Buffer Management to Cache Management
- Two aspects (as usual) :
  - data (query processing) [today's paper]
  - metadata (indexes) [next class]

# OLD WINE in NEW BOTTLE? (caches versus buffers)

- Cache hierarchy exists (Registers, L1, L2, Memory)
- Cache is entirely hardware managed, no user control
- Cache is not fully associative – in practice, either direct mapped or low set-associativity
- Cannot trade CPU cycles for improving cache performance
- Instruction/Data separation

# BACKGROUND

# BASIC CACHE FUNDAS

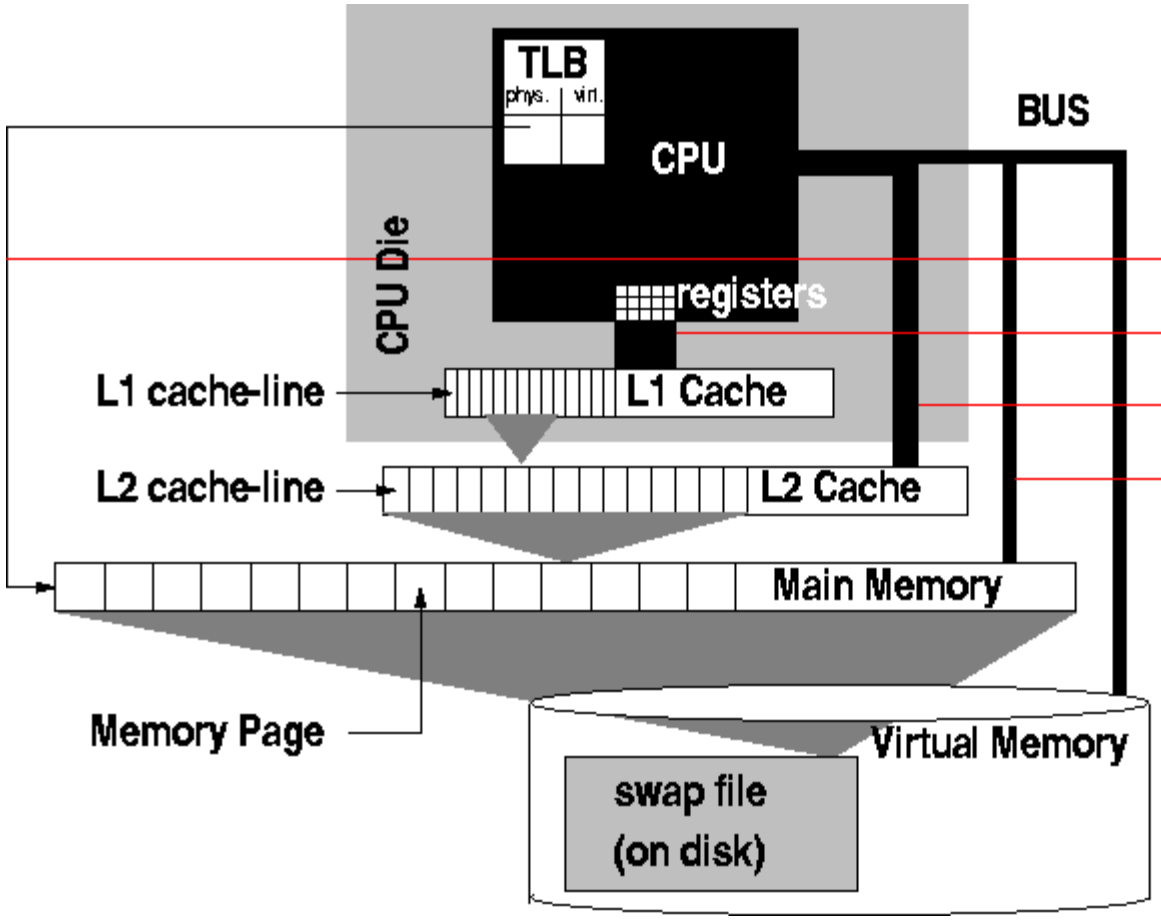
- Parameters
  - Capacity (number of bytes)
  - Block Size (fetch unit from memory to cache)
  - Associativity (DM/SA/FA)
- Misses
  - Compulsory (first access)
  - Capacity (Miss in FA cache)
  - Conflict (Hit in FA cache but miss in SA cache)

# DB Folk Wisdom

- Query processing touches so much data that locality in data accesses is inherently poor.
- Once data is in memory, it is accessed as fast as it could be.
- Hence, no need for cache-consciousness.

- Are these beliefs correct ?

# Memory Access Hierarchy




Latencies:

TLB miss: 5-60 cycles

L1 hit: 1-2 cycles

L1 miss: 6-20 cycles

L2 miss: 40-100 cycles

**Sizes:**

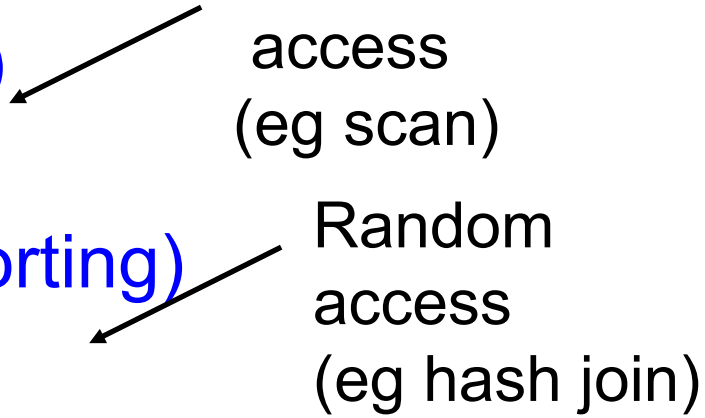
**Disk = 100 Mem**

**Mem = 100 L2**

**L2 = 100 L1**

# LOCALITY TECHNIQUES

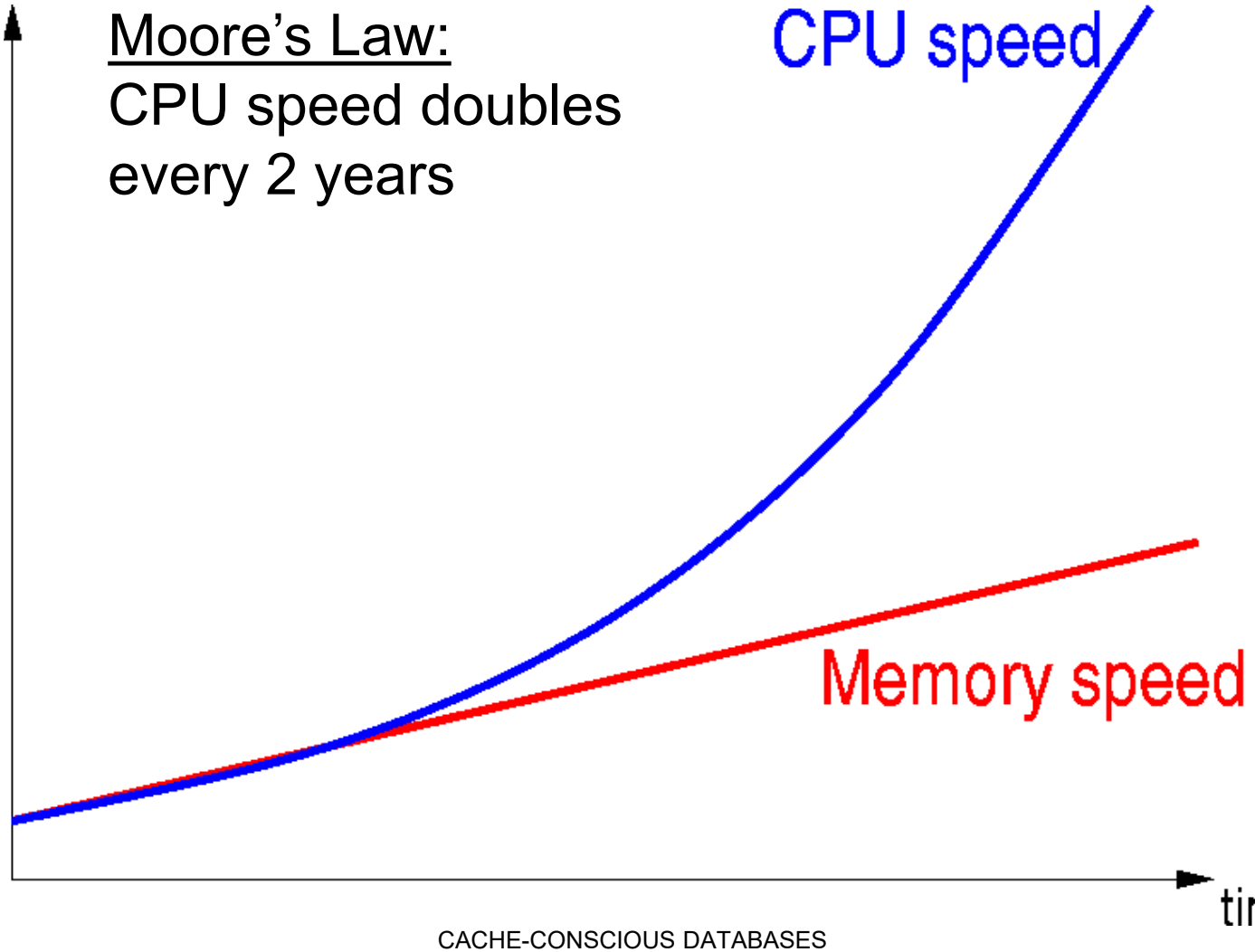
- For given cache organization, improve spatial and temporal locality
  - Blocking (like in nested loops)
    - change the algo
  - Partitioning (like in external sorting)
    - change the data layout
  - Data Filtering / Clustering
    - project out / cluster the relevant data
  - Loop Fusion
    - all operations on a given object done together



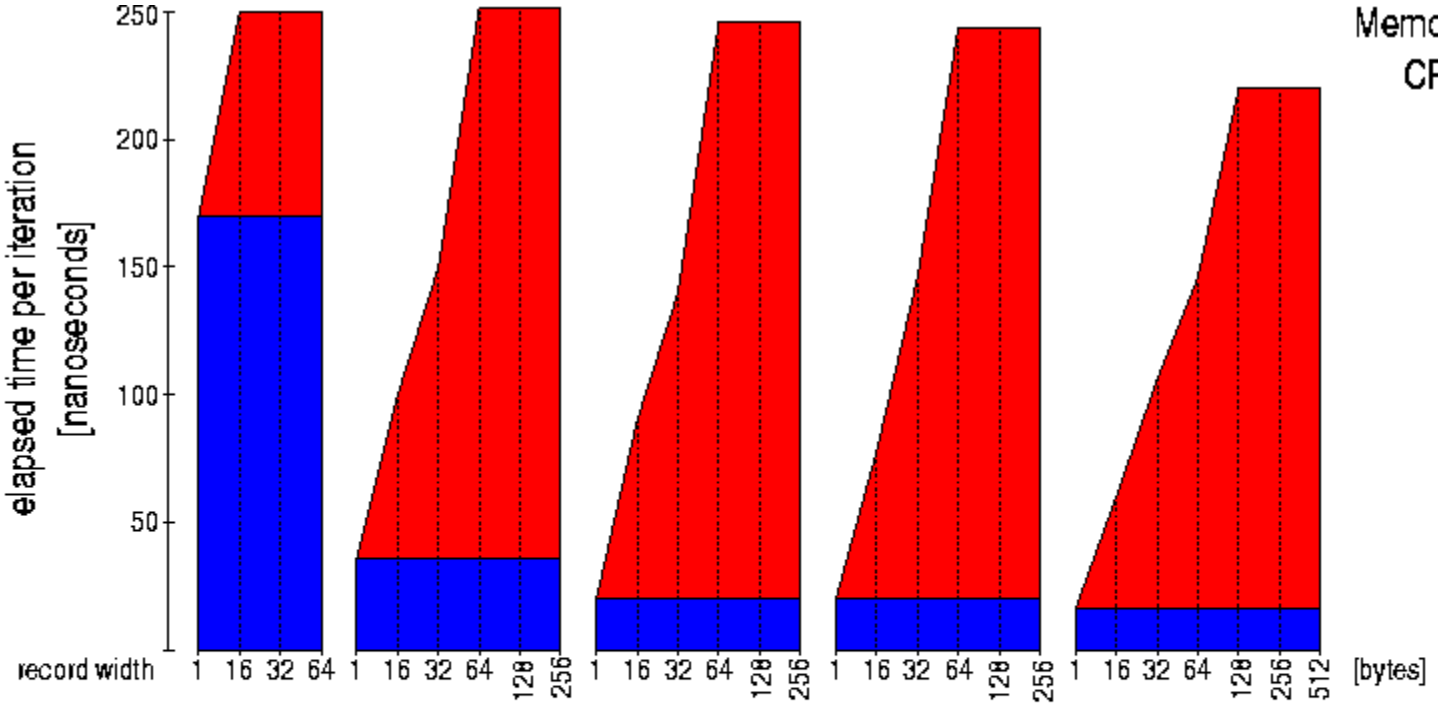
# GOING DUTCH

# Technology Trends

Moore's Law:
CPU speed doubles every 2 years



# Simple Table Scan




| | | | | | |
|---|---|---|---|---|---|
| year | 1992 | 1996 | 1997 | 1998 | 1998 |
| system | Sun LX | Sun Ultra | SunUltra | DEC Alpha | Origin2000 |
| CPU type | Sparc | UltraSparc | UltraSparcII | Alpha | R10000 |
| CPU speed | 50 MHz | 200 MHz | 296 MHz | 500 MHz | 250 MHz |
| cache line size (L1/L2) | 16 / - | 16 / 64 | 16 / 64 | 32 / 64 | 32 / 128 |

# Implications

- Memory access is a bottleneck
- Prevent cache and TLB misses
- Cache lines must be used fully

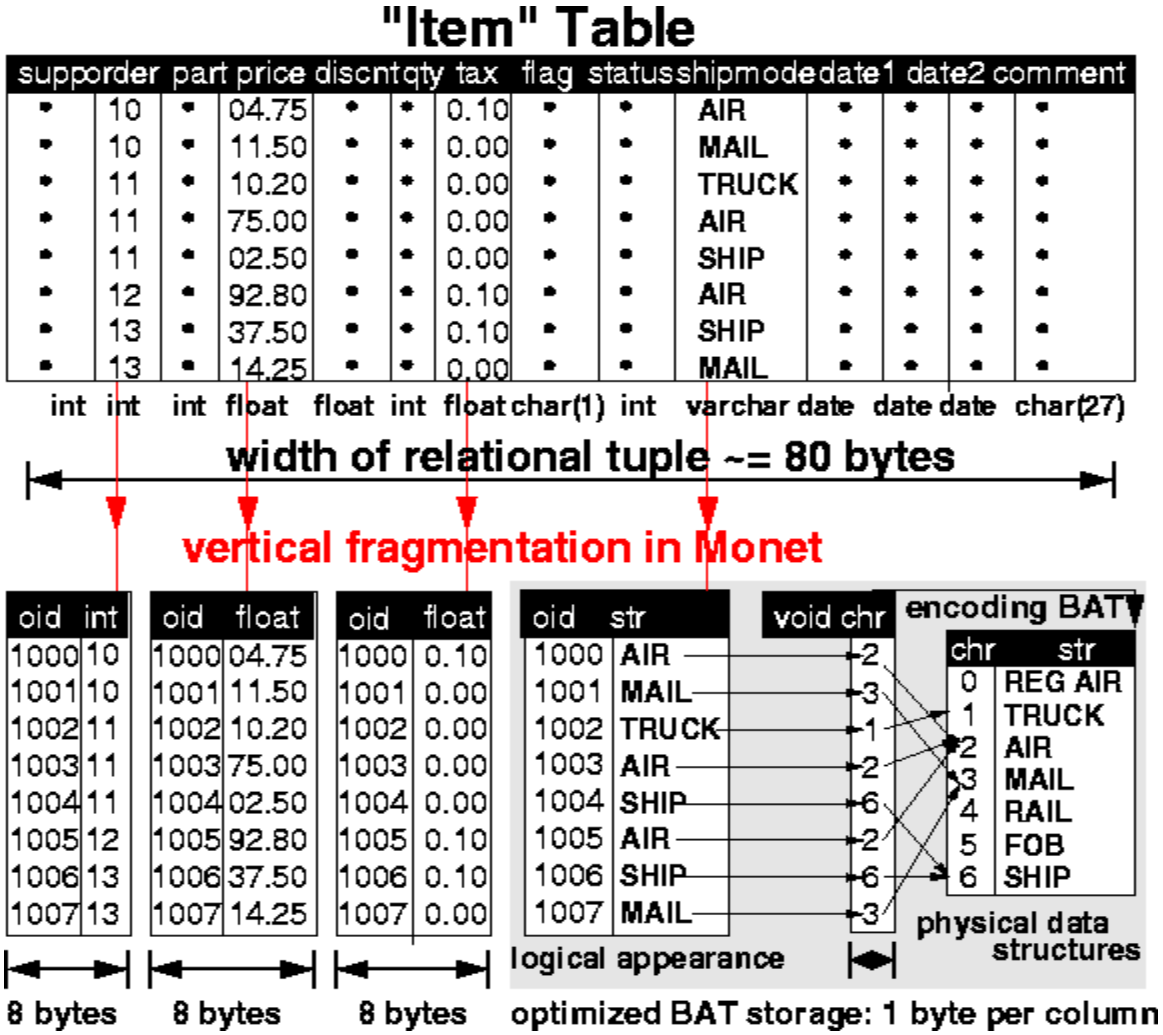
- DBMS must optimize
  - Data structures
  - Algorithms (focus: join)

# DATA STRUCTURES

# VERTICAL DECOMPOSITION

- Binary Association Table, one per column of original relation
- [OID, value] in each tuple of BAT
  - called a BUN
- OID can be implicit ⇒ VOID
- Small domain encodings

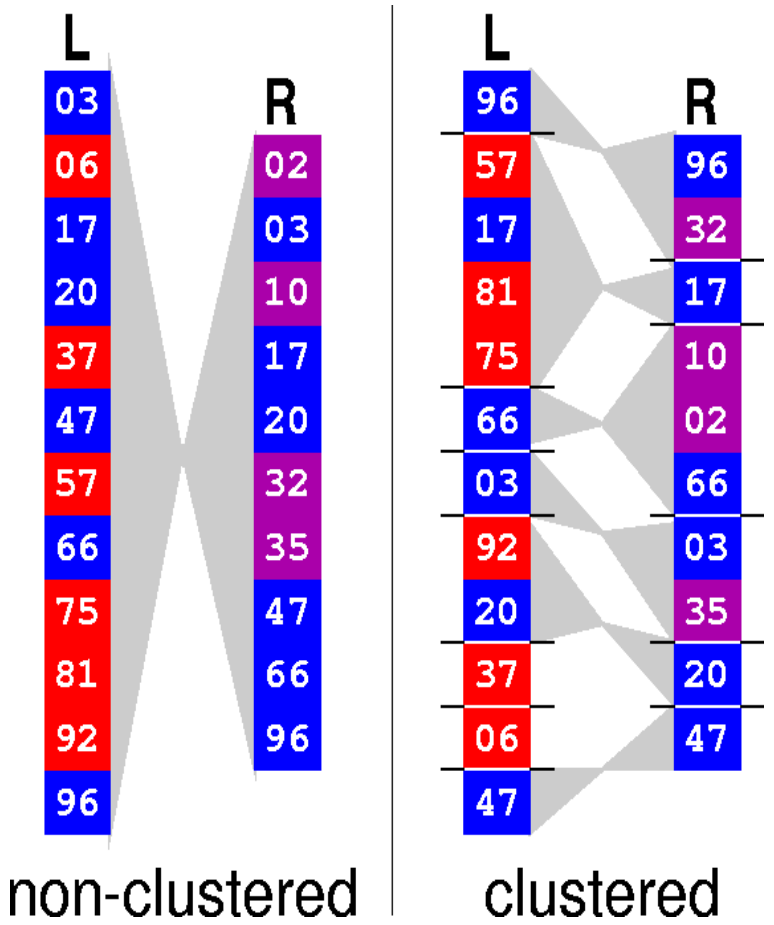
# V D Example


"Item" Table
supp order part price discnt qty tax flag status shipmode date1 date2 comment
AIR
MAIL
TRUCK
AIR
SHIP
AIR
SHIP
MAIL
int int int float float int float char(1) int varchar date date date char(27)
width of relational tuple ~= 80 bytes
vertical fragmentation in Monet
oid int
oid float
oid float
oid str
void chr
encoding BAT
chr str
0 REG AIR
1 TRUCK
2 AIR
3 MAIL
4 RAIL
5 FOB
6 SHIP
physical data structures
logical appearance
8 bytes
8 bytes
8 bytes
optimized BAT storage: 1 byte per column

# JOIN PROCESSING

# Partitioned Joins

- Cluster both input relations
- Create clusters that fit in cache memory
- Join matching clusters
- Two algorithms:
  - Radix-Join
    (partitioned nested-loop)
  - Partitioned hash-join



# Straightforward Clustering

- Problem:

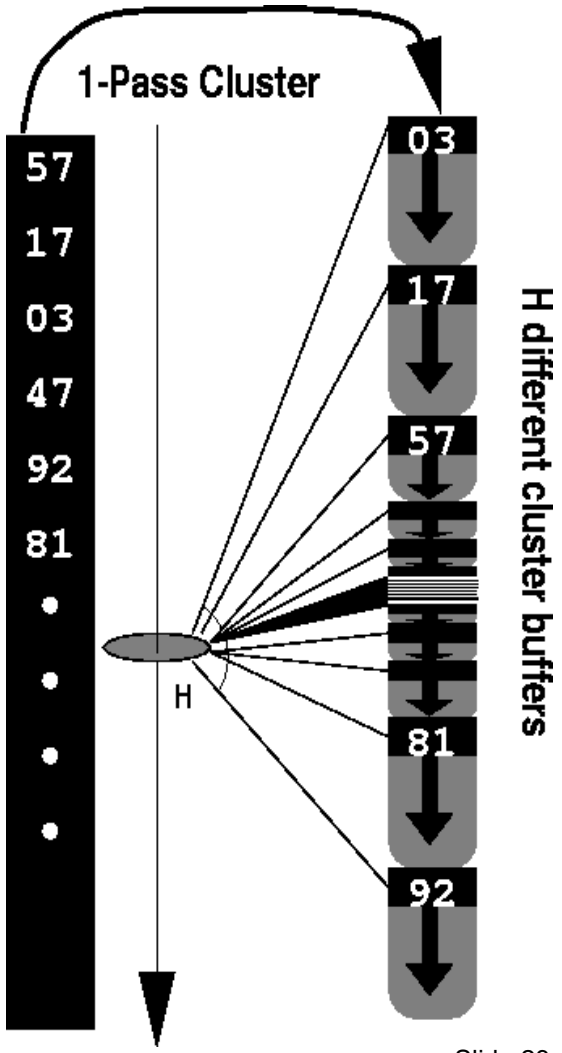
  Number of clusters exceeds number of

  – TLB entries ⇒ TLB trashing

  – Cache lines ⇒ cache trashing

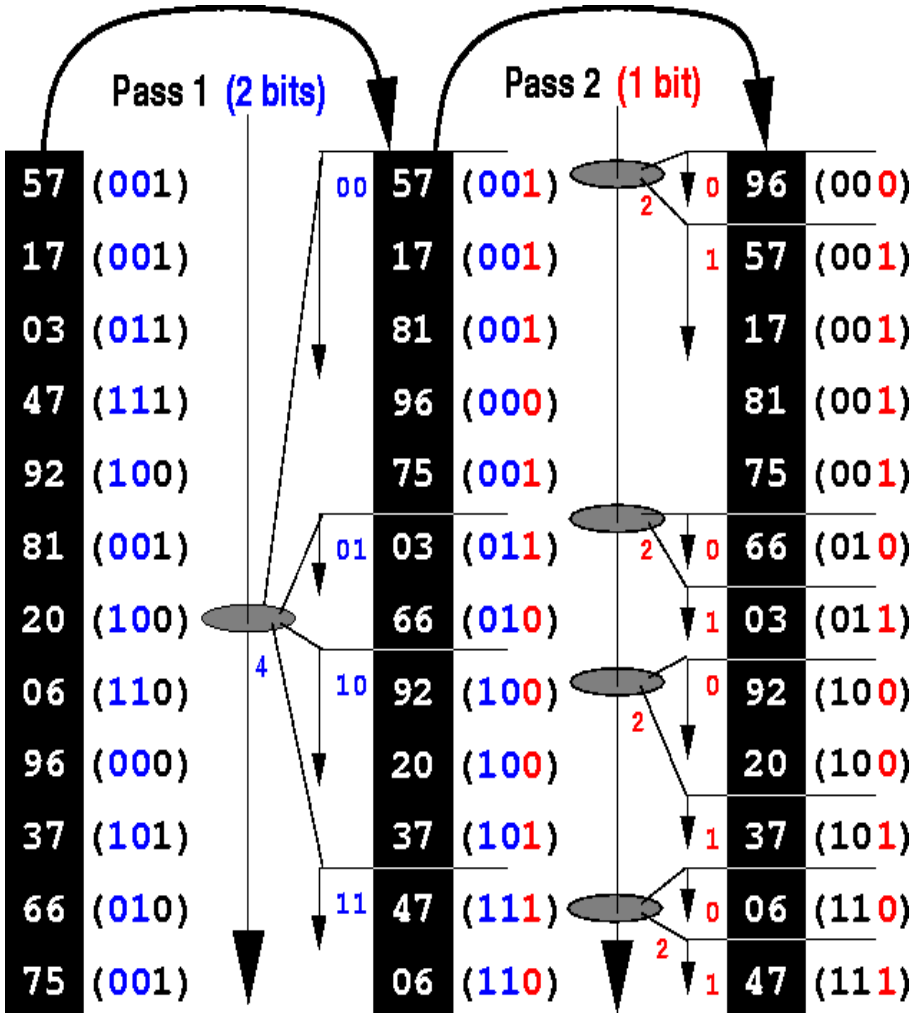
- Solution:

  Multi-pass radix-cluster



# Multi-Pass Radix-Cluster

- Multiple clustering passes
- Limit number of clusters per pass
- Avoid cache/TLB trashing
- Trade memory cost for CPU cost
- Any data type (radix-hash)



# Join

- Partitioned Radix-Join: Just do nested-loops because the cluster sizes are small.
- Partitioned Hash-Join: Effectively becomes merge-join of clusters because the radix-clustered relation is in fact ordered on radix-bits, and can then fit the inner relation cluster in cache, and do standard hash-join inside-cache (build hash-index on inner cluster and then probe with tuples from outside cluster).

# EXPERIMENTS

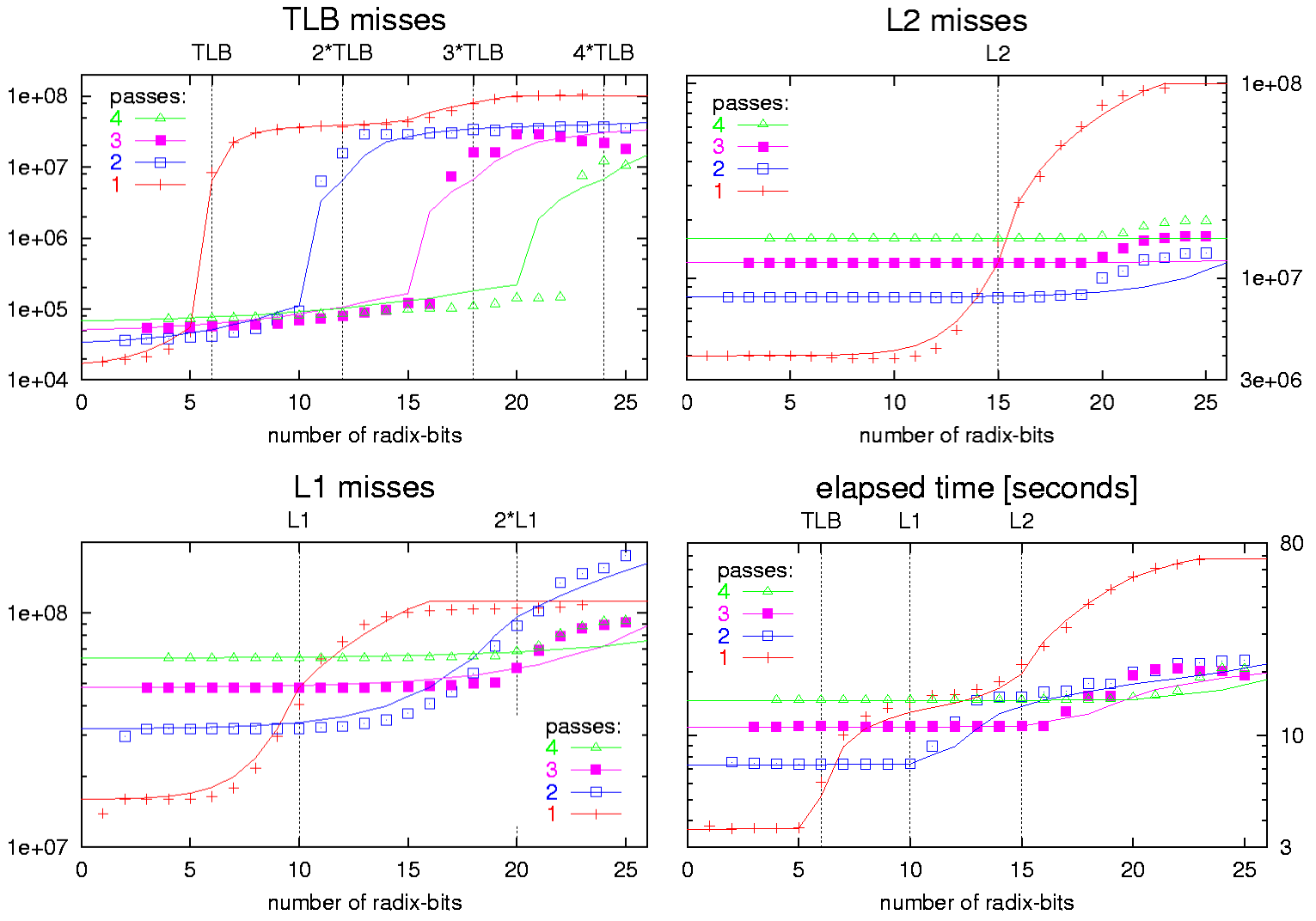
# Experimental Setup

- Platform:
  - SGI Origin2000 (MIPS R10000, 250 MHz)
  - 32 KB L1 cache (1024 lines * 32 bytes)
  - 4 MB L2 cache (32768 lines * 128 bytes)
  - 64 entries TLB with Page-size 16 KB $\Rightarrow$ 1M of directly addressable memory
- System:
  - Monet DBMS
- Hardware event counters
  - to analyze L1/L2 cache and TLB misses

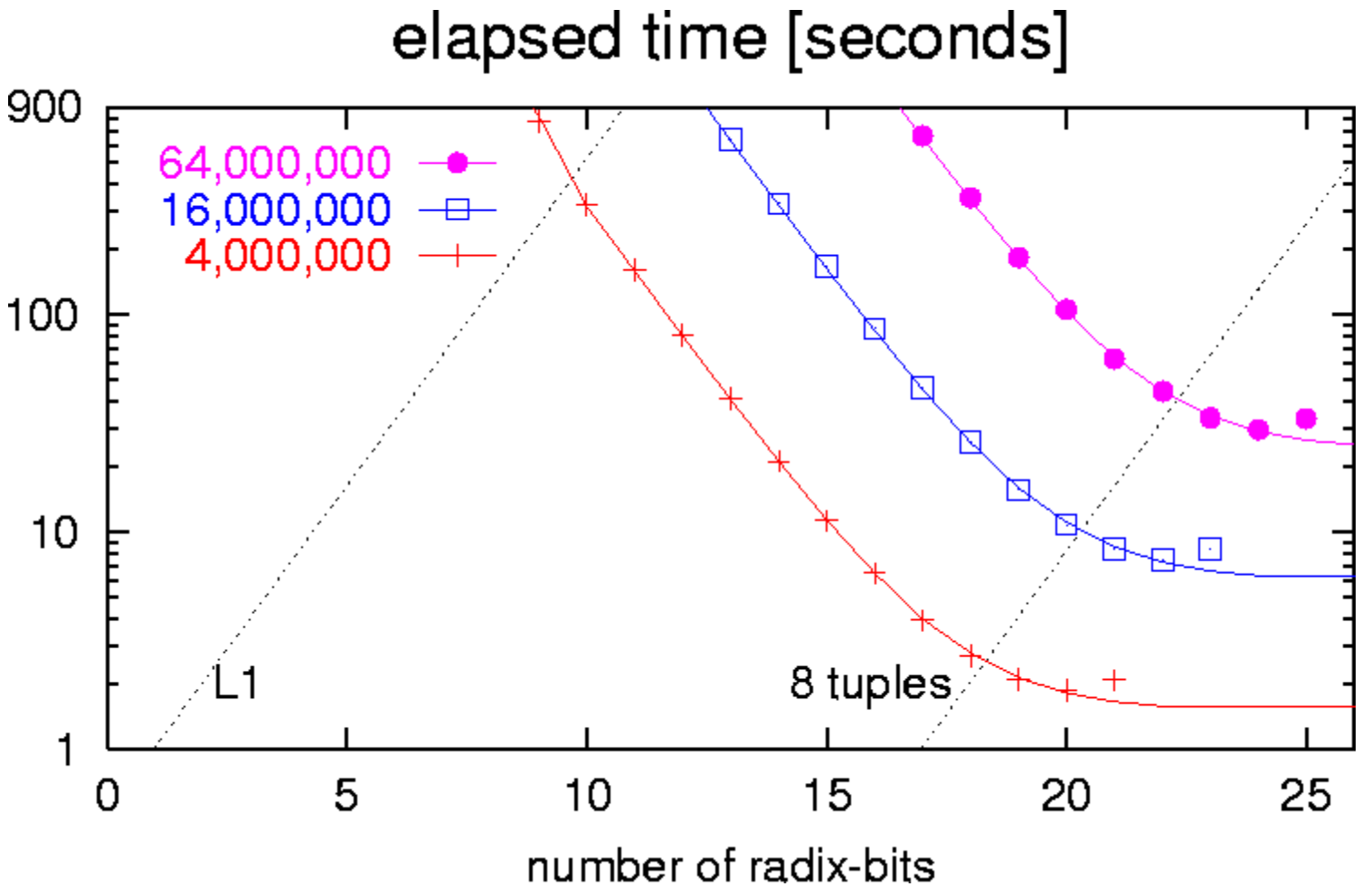
# Experimental Setup (contd)

- Data sets:
  - Uniformly distributed unique Integer join columns
  - Join hit-rate of 1
  - Cardinalities: 15K — 64M
  - Output: (OID, OID) join index
- Clustering Parameters:
  - B: number of clustering bits
  - P: number of clustering passes
  - $B_{p:}$ number of clustering bits/pass
  - $_1\sum^P B_p = B$

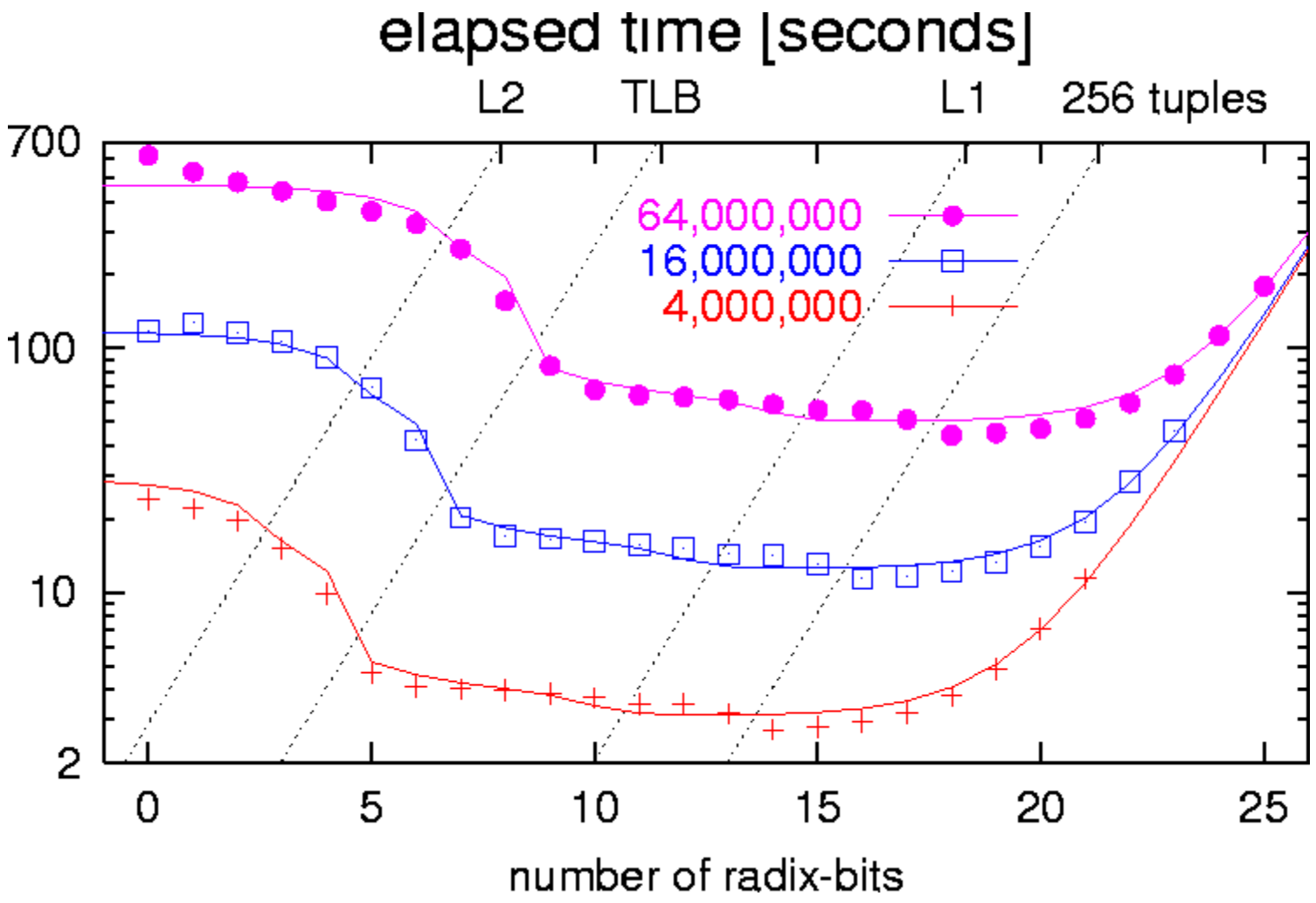
# Radix-Clustering [Fig 9]



# Radix-NestedLoops-Join [Fig 10d]

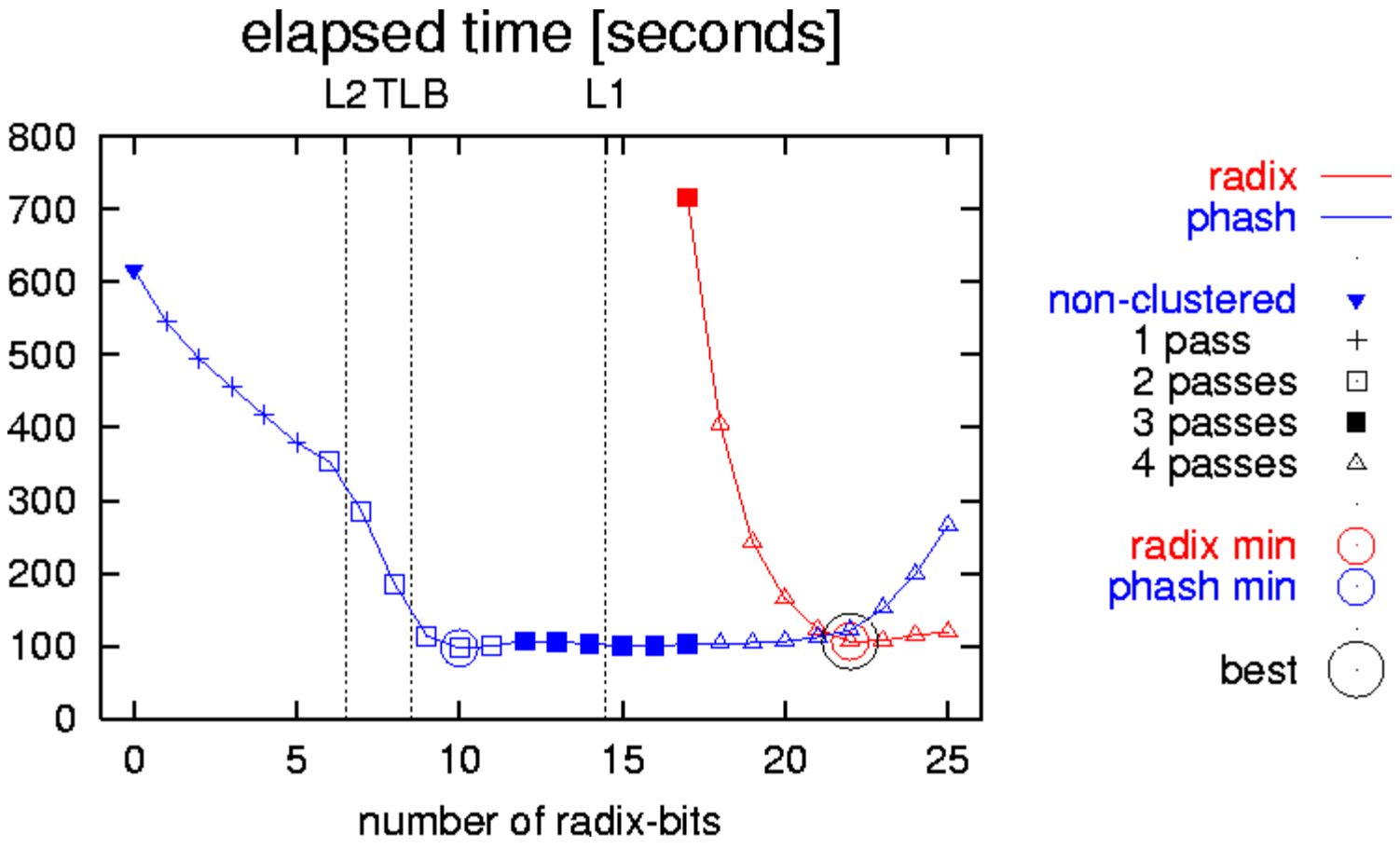


# Radix-Partitioned-Hash-Join [Fig 11d]



# Overall Performance

[Fig 12]



# Effect of Cache-Consciousness [Fig 13]

- Log-scale: Orders of magnitude improvement over sort-merge and standard hash-join

# TAKE-AWAY

- Problem:
  - Memory access is the new performance bottleneck
- Solutions:
  - Vertical decomposition for sequential access
  - Radix-cluster-based algorithms for random access (eg hash join)

# END CC DATABASES

## E0 261