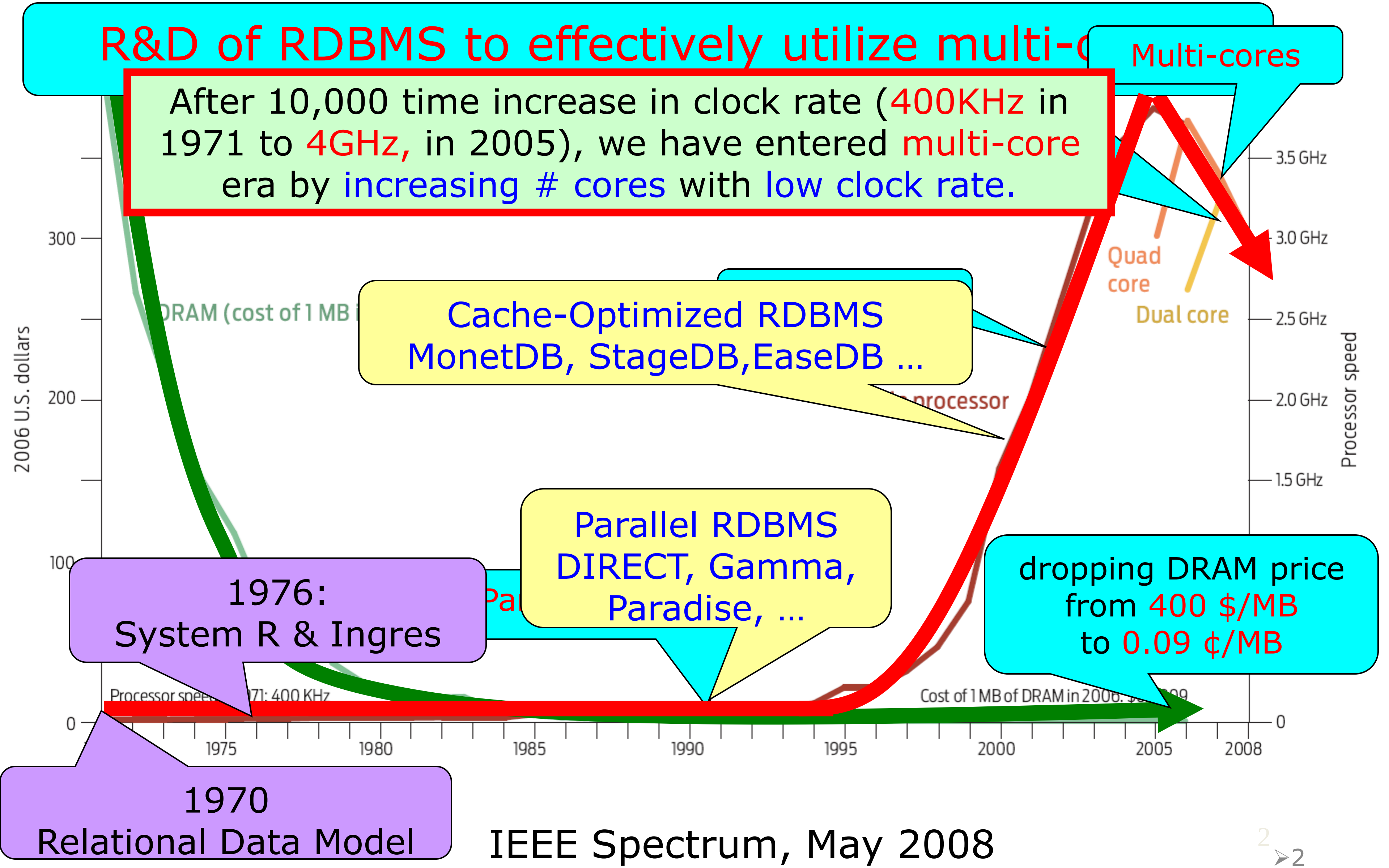
MCC-DB: Minimizing Cache Conflicts in Multi-core Processors for Databases

Rubao Lee[1,2] Xiaoning Ding[2] Feng Chen[2]
Qingda Lu[3] *Xiaodong Zhang*[2]

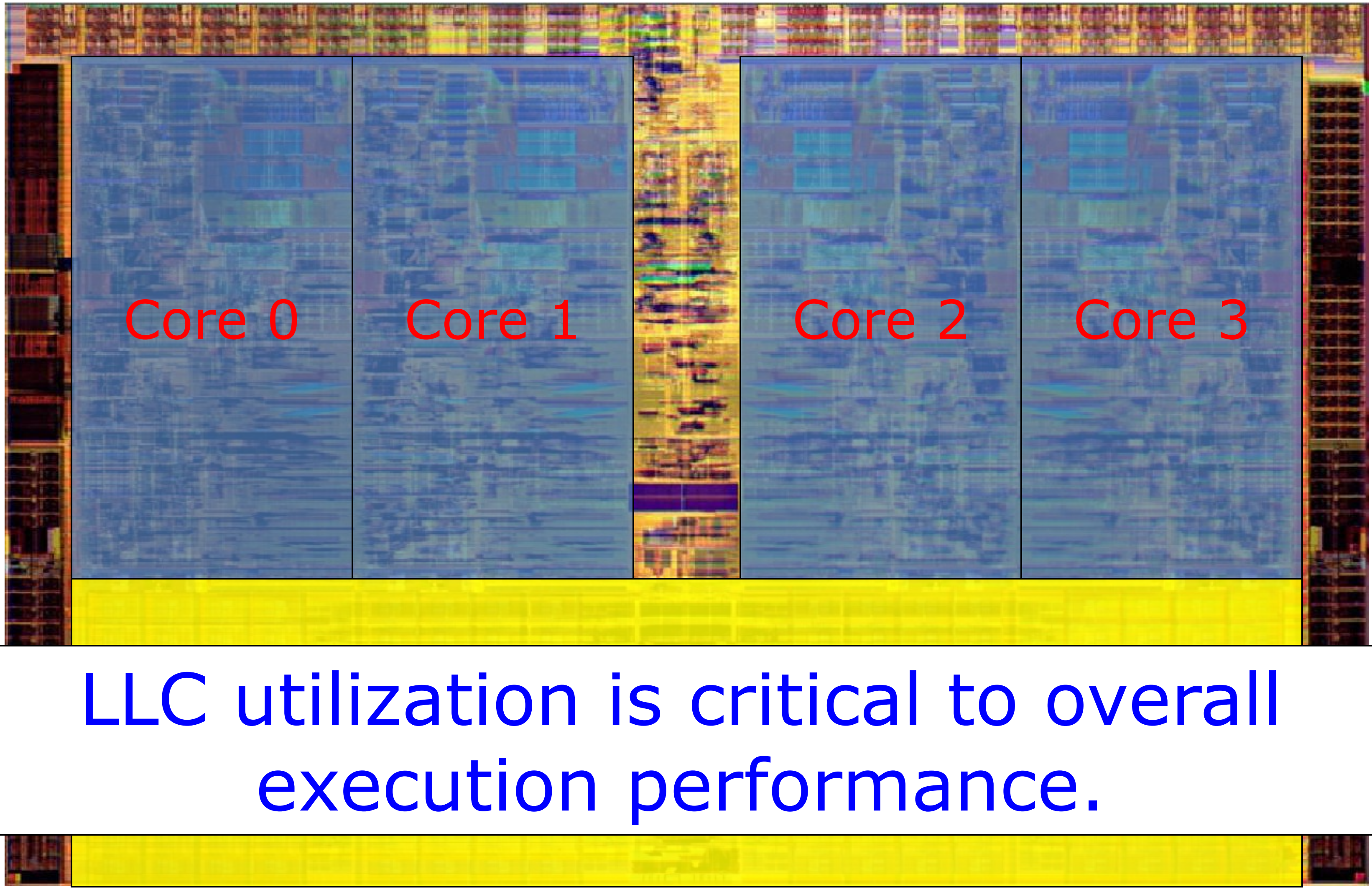
[1]Inst. of Computing Tech., Chinese Academy of Sciences
[2]Dept. of Computer Sci & Eng., The Ohio State University
[3]Systems Software Lab., Intel Cooperation

R&D of Database Systems Have Been Driven by Moore's Law



IEEE Spectrum, May 2008

A Multi-core Processor Provides Shared Hardware Resources

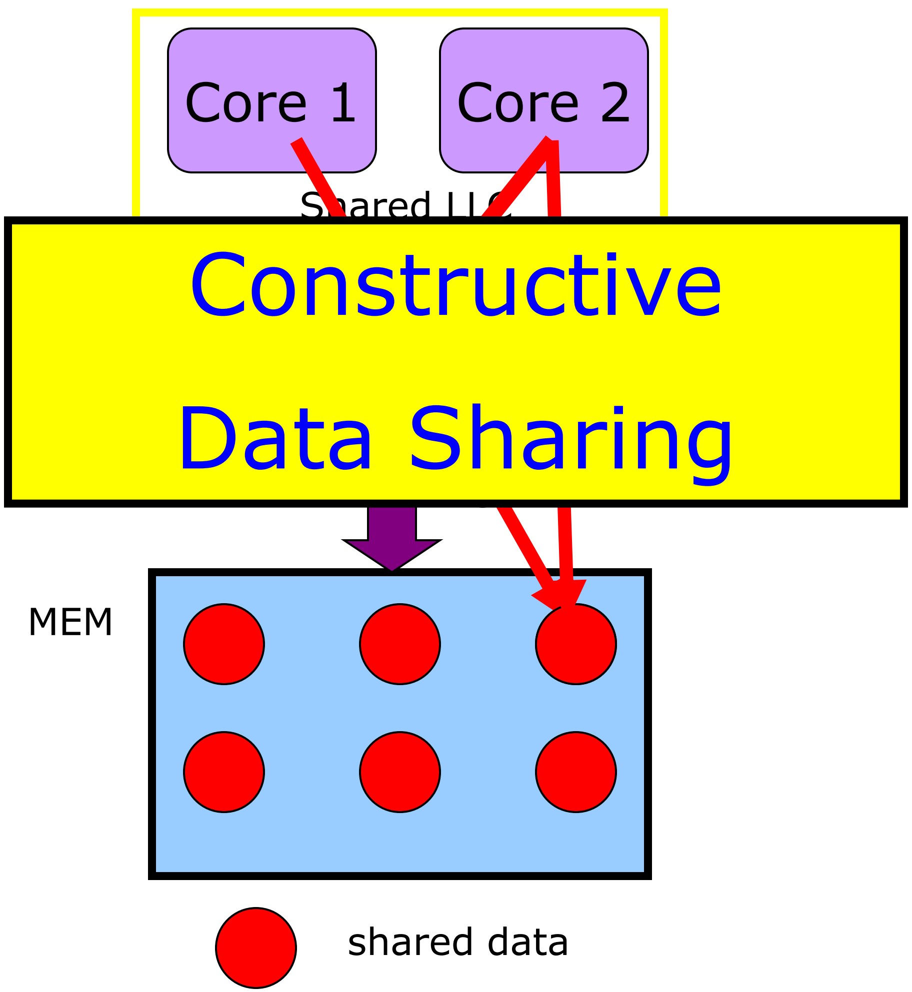


LLC utilization is critical to overall execution performance.

Intel Nehalem

The Shared LLC is a Double-edge Sword!

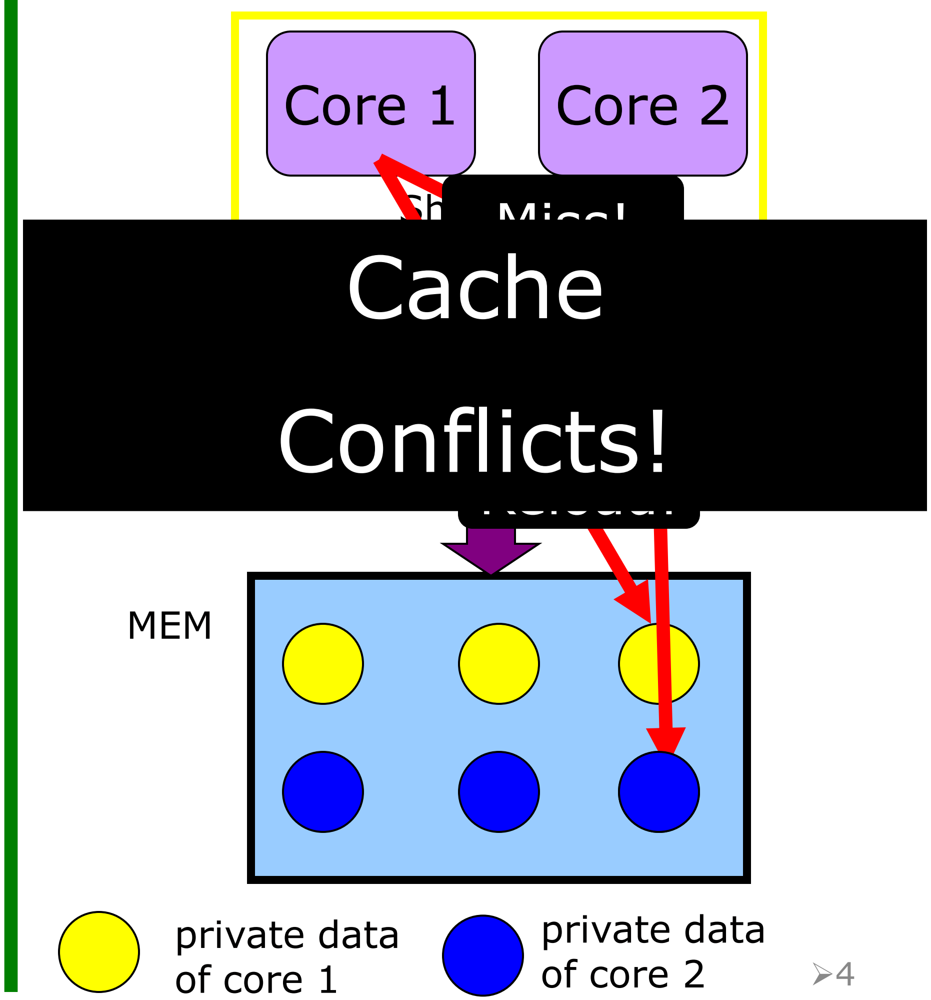
Good News



Constructive

Data Sharing

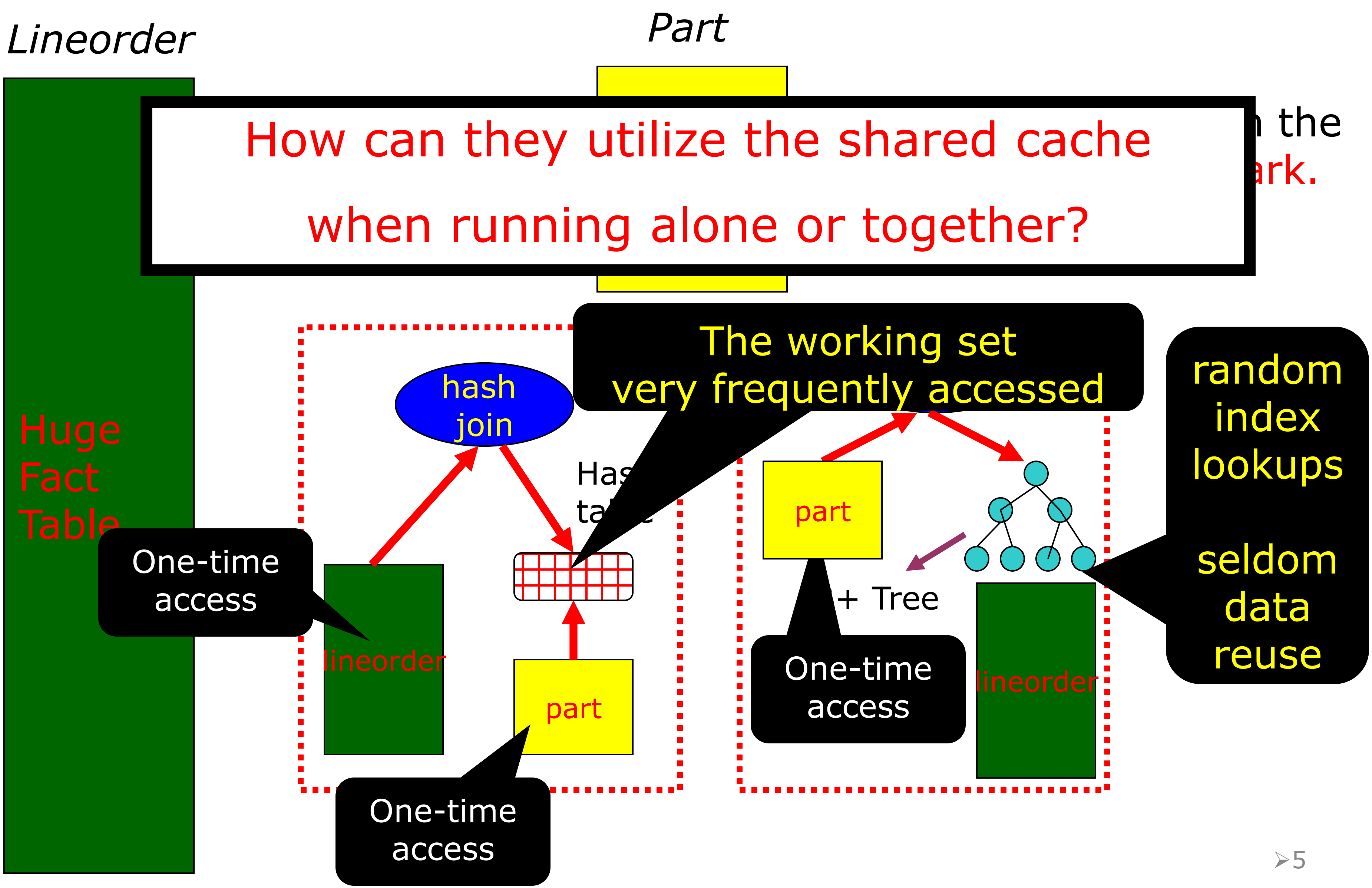
Bad News



Cache

Conflicts!

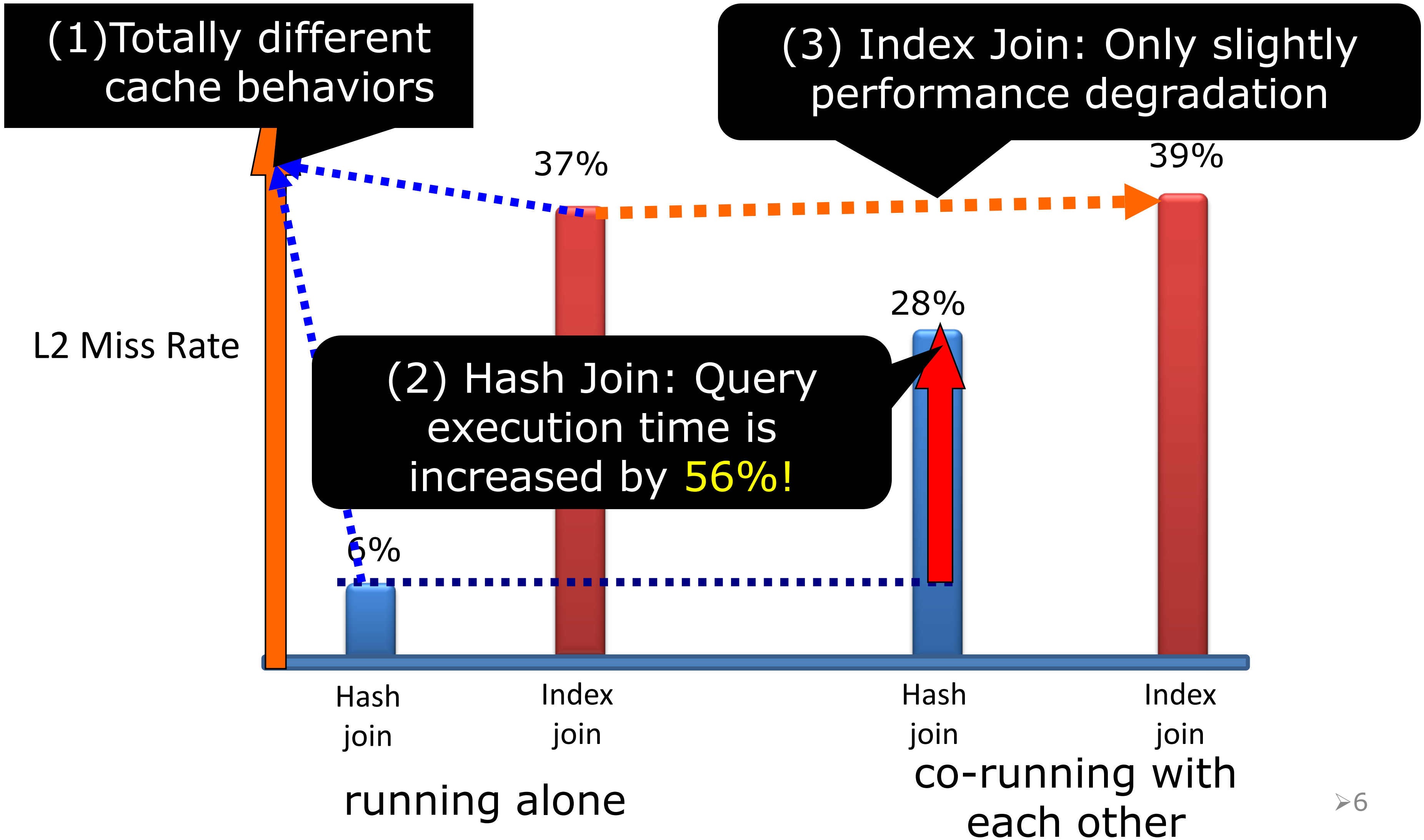
A Motivating Example: Hash Join and Index Join



L2 Miss Rates of Co-running Hash Join and Index Join

Hardware: Core2Quad Xeon X5355 (4MB L2$ for two cores)

OS: Linux 2.6.20 DBMS: PostgreSQL 8.3.0 Tool: Perfmon2

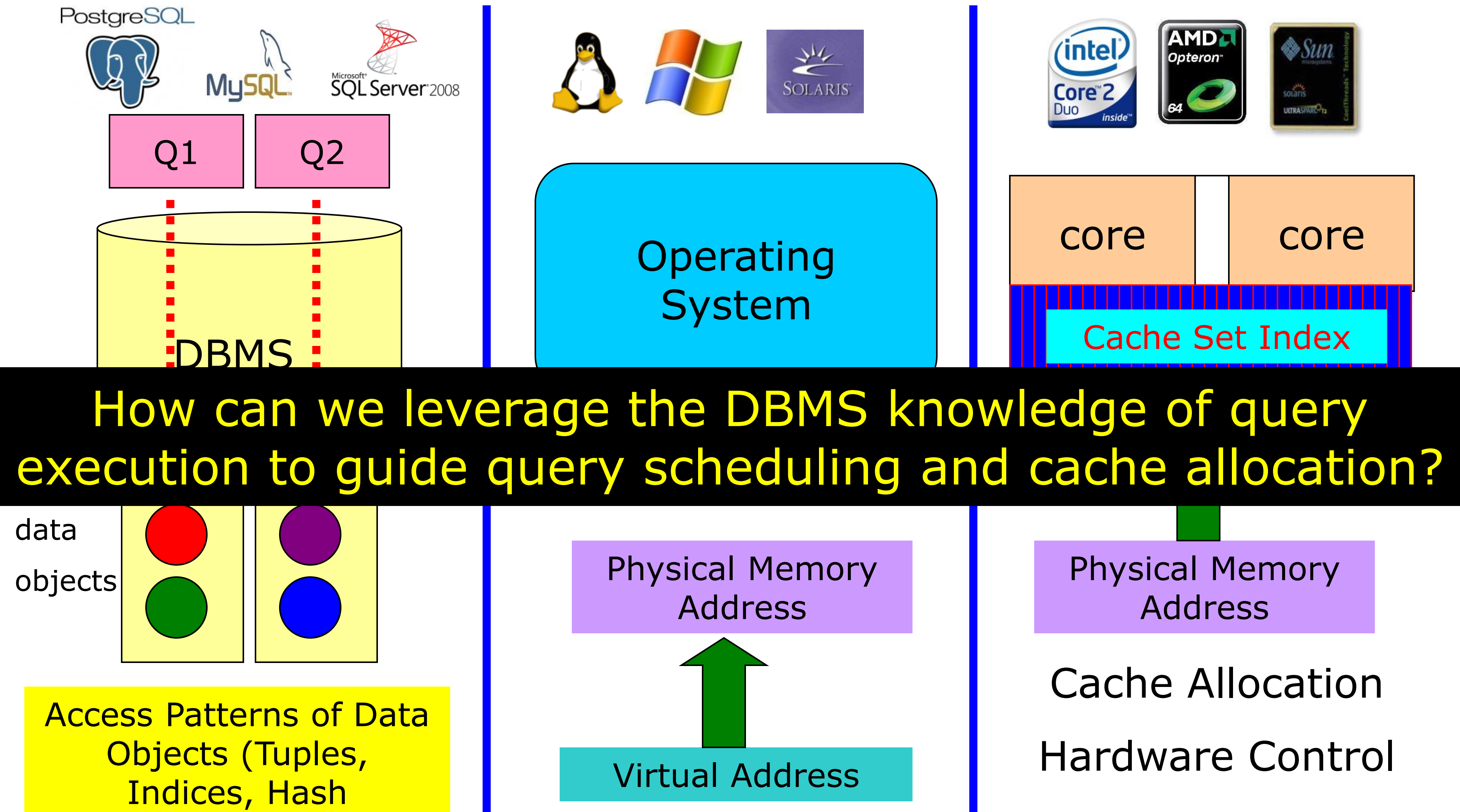


Challenges of DBMS running on Multi-core

- DBMSs have successfully been developed in an architecture/OS-independent mode
  - ✓ Buffer pool management (bypassing OS buffer cache)
  - ✓ Tablespace on raw device (bypassing OS file system)
  - ✓ DBMS threads scheduling and multiplexing

- DBMSs are not multicore-aware
  - ➢ Shared resources, e.g. LLC, are managed by CPU and OS.
  - ➢ Interferences of co-running queries are out of DBMS control
  - ➢ Locality-based scheduling is not automatically handled by OS

Who Knows What?

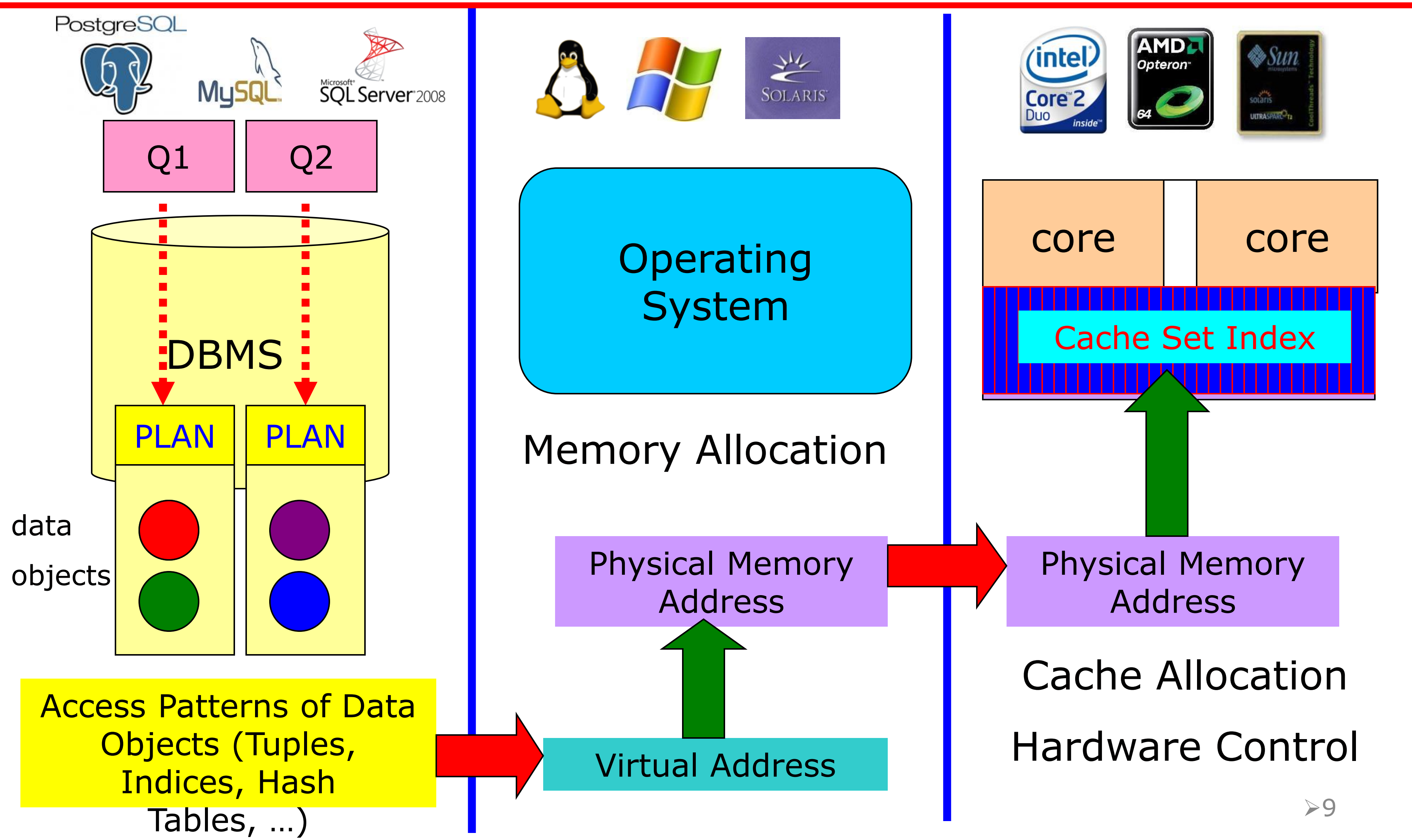


How can we leverage the DBMS knowledge of query execution to guide query scheduling and cache allocation?

The three parties are DISCONNECTED!
DBMS doesn't know cache allocation. OS and Chip don't know data access patterns.
The problem: cache conflicts due to lack of knowledge of query executions

Our Solution: MCC-DB

Objectives : Multicore-Aware DBMS with communication and cooperation among the three parties.



Outlines

- The MCC-DB Framework
  - Sources and types of cache conflicts
  - Three components of MCC-DB
  - System issues
  - Implementation in both PostgreSQL and Linux kernel
- Performance Evaluation
- Conclusion

Sources and Types of Cache Conflicts

1. Private data structures during query executions (cannot be shared by multi-cores)
2. Different cache sensitivities among various query plans
3. Inability to protect cache-sensitive plans by the LRU scheme
4. Limited cache space cannot hold working sets of co-running queries.

The locality strength of a query plan is determined by its data access pattern and working set size.

Strong locality

Small working set size (relative to cache size), which is frequently accessed

Moderate Locality

working set size comparable with cache size, which is moderately accessed

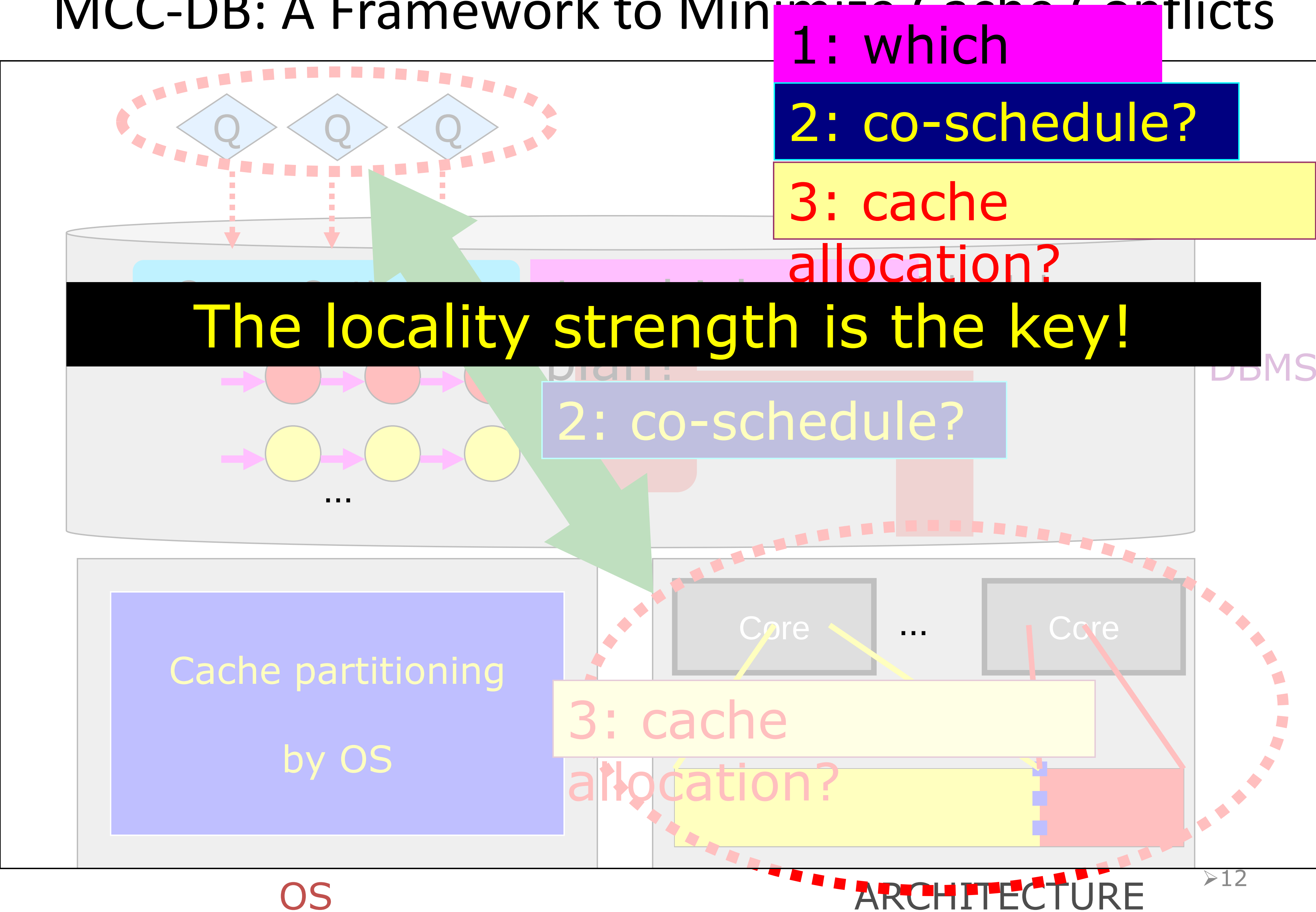
→ Capacity Contention

Weak Locality

seldom data reuse, one-time accesses, which has a large volume.

→ Cache Pollution

MCC-DB: A Framework to Minimize Cache Conflicts



Critical Issues

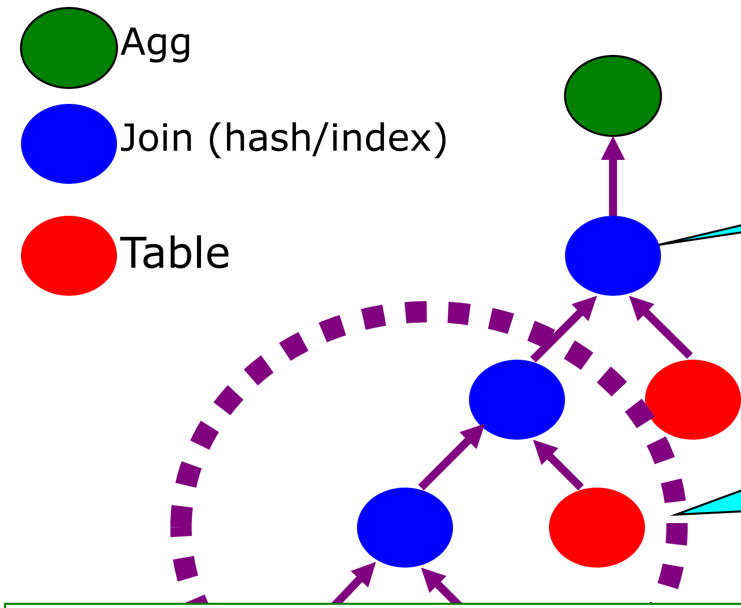
- How to estimate the locality strength of a query plan? (in DBMS domain)
- How to determine the policies for query execution co-scheduling and cache partitioning? (in DBMS domain and interfacing to OS)
- How to partition the shared LLC among multiple cores in an effective way? (in OS multicore processor domain)

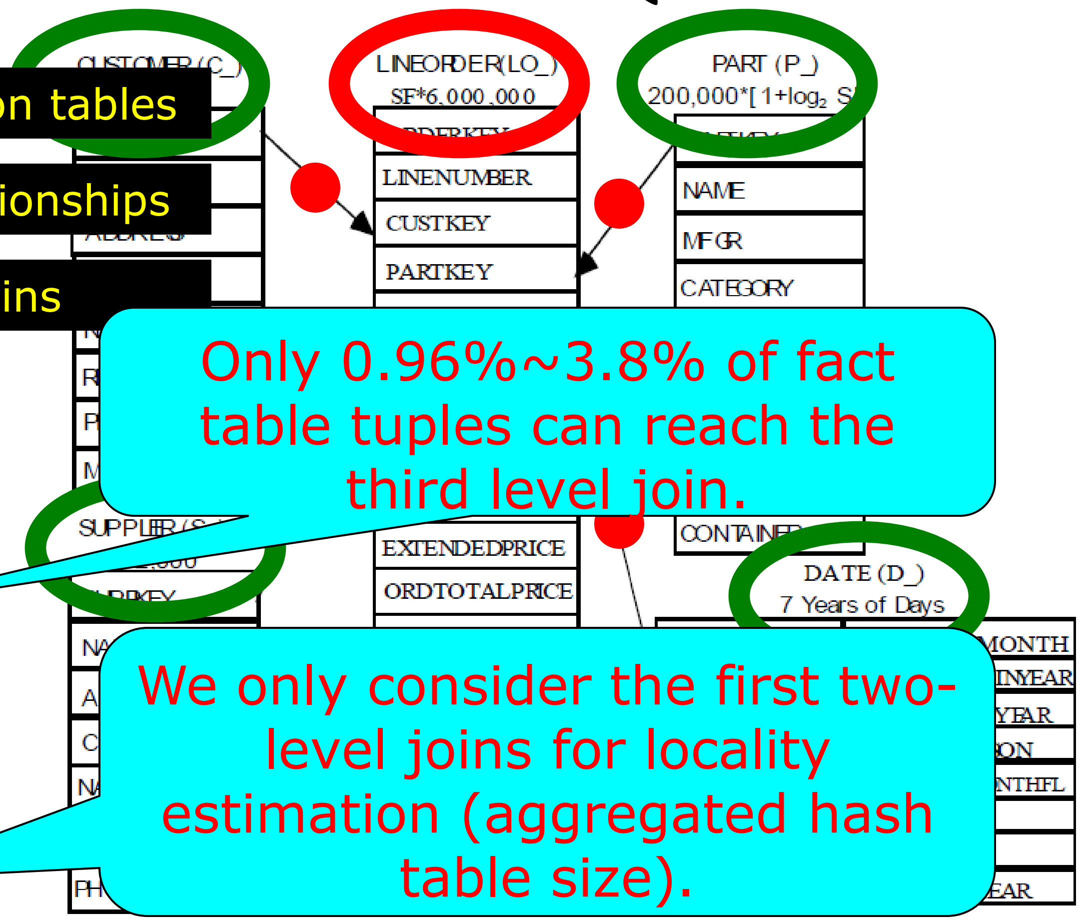
Locality Estimation for Warehouse Queries

1: Huge fact table and small dimension tables

2: equal-join on key-foreign key relationships

3: aggregations and grouping after joins





Only 0.96%~3.8% of fact table tuples can reach the third level join.

We only consider the first two-level joins for locality estimation (aggregated hash table size).

The figure is from *Star Schema*

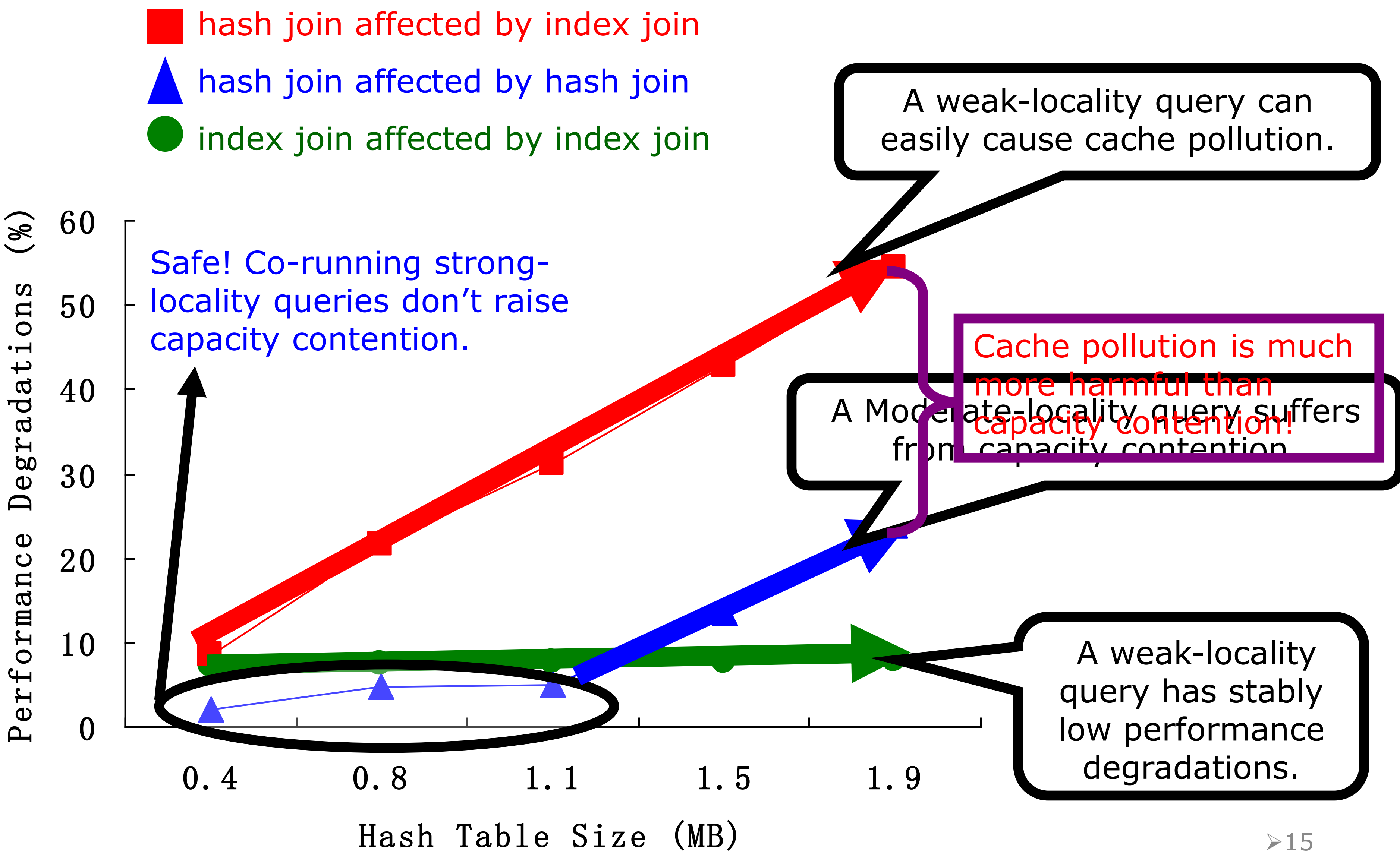
Locality Estimation of Join Operators

◆**Index join** is estimated to have weak locality due to random index lookups on the huge fact table.

◆**Hash join** is estimated to have strong, moderate, or weak locality, according to its hash table size (see papers for the details).

Interference between Queries with Different Localities

The figure is only a part of the experimental result for measuring performance degradations when co-running various hash joins and index joins (see papers!).



Cache Conflicts

Cache pollution:

A weak-locality plan (a large volume of one-time accessed data sets) can easily pollute the shared cache space.

Capacity contention:

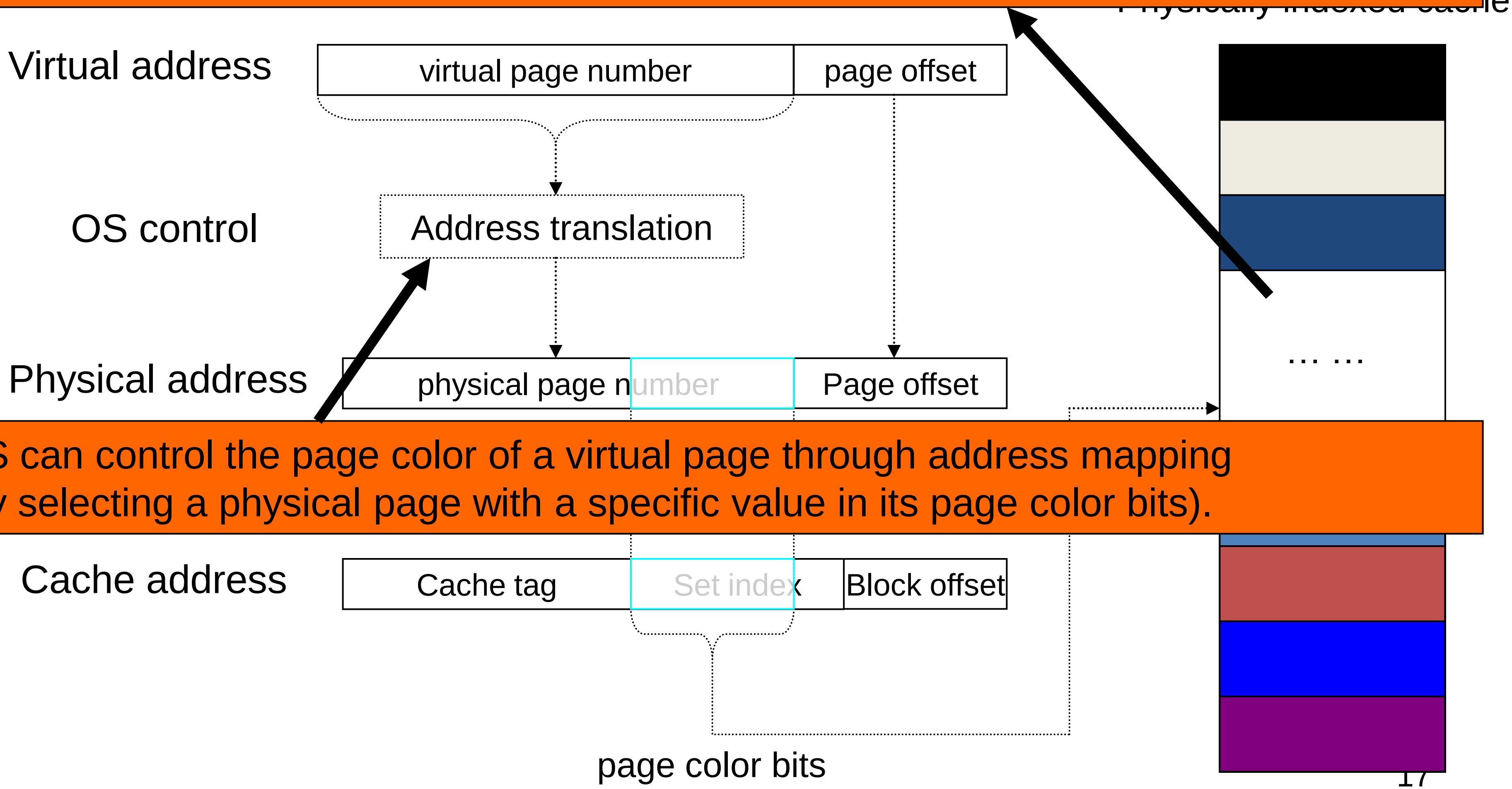
Co-running moderate-locality plans compete for the shared cache space, and misses are due to limited space.

Cache pollution is more damaging than capacity contention! (useful data objects are replaced by one-time accessed ones)

Capacity contention cannot be removed (limited cache space), but cache pollution can be (cache partitioning)!

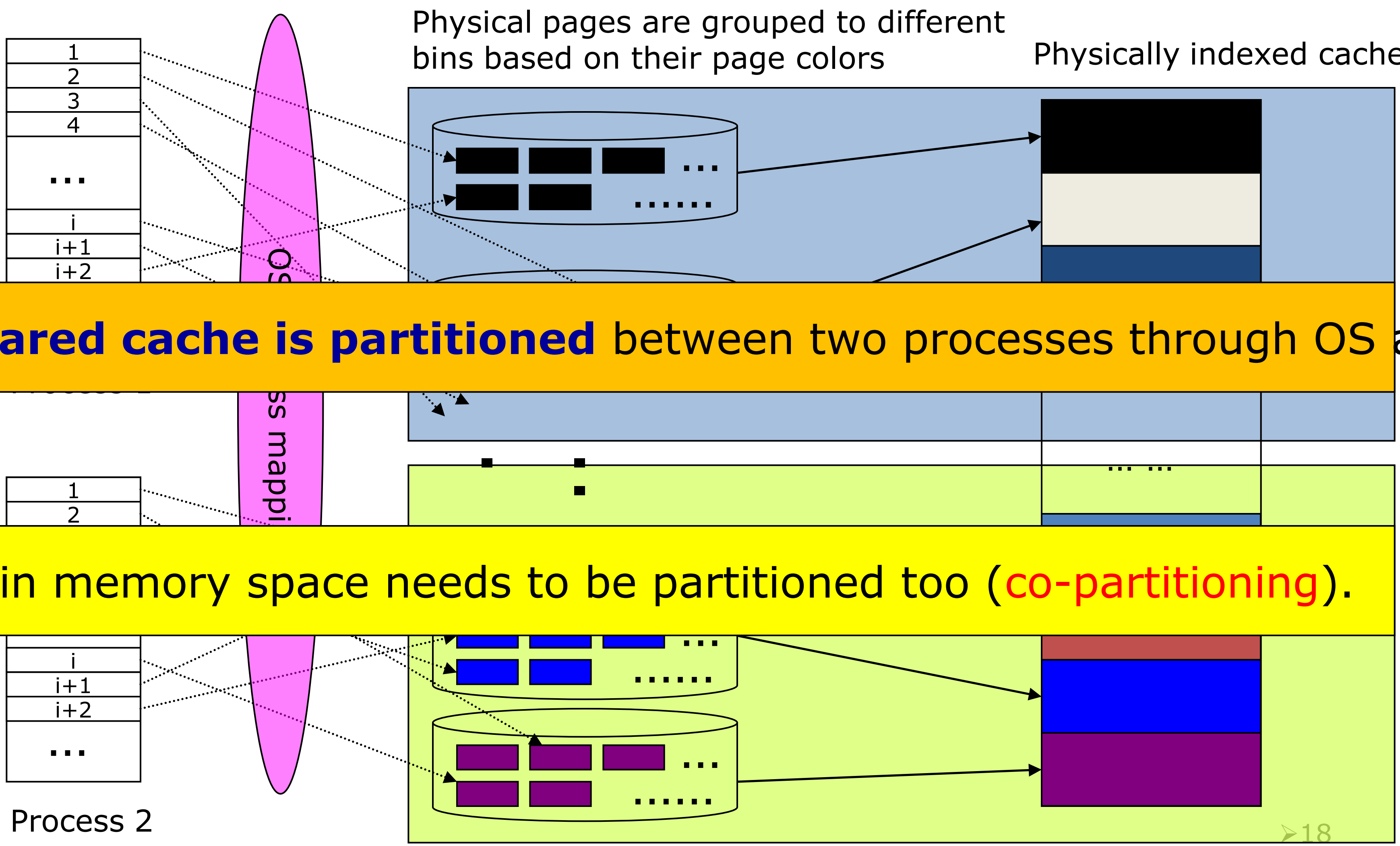
Page Coloring for Cache Partitioning

- Physically indexed caches are divided into multiple regions (colors).
- All cache lines in a physical page are cached in one of those regions (colors).



OS can control the page color of a virtual page through address mapping
(by selecting a physical page with a specific value in its page color bits).

Shared LLC can be partitioned into multiple regions



Shared cache is partitioned between two processes through OS ad

Main memory space needs to be partitioned too (co-partitioning).

Scheduling with/without cache partitioning

Scheduling without cache partitioning: a DBMS-only effort

SLS: co-scheduling query plans with the same locality strength.

(1) Low interference between weak-locality plans

(2) Avoid cache pollution by not co-running them together

(3) Cache allocation is not applied: performance is sub-optimal

---

Scheduling with cache partitioning: DBMS + OS efforts.

MLS: co-scheduling query plans with mixed locality strengths.

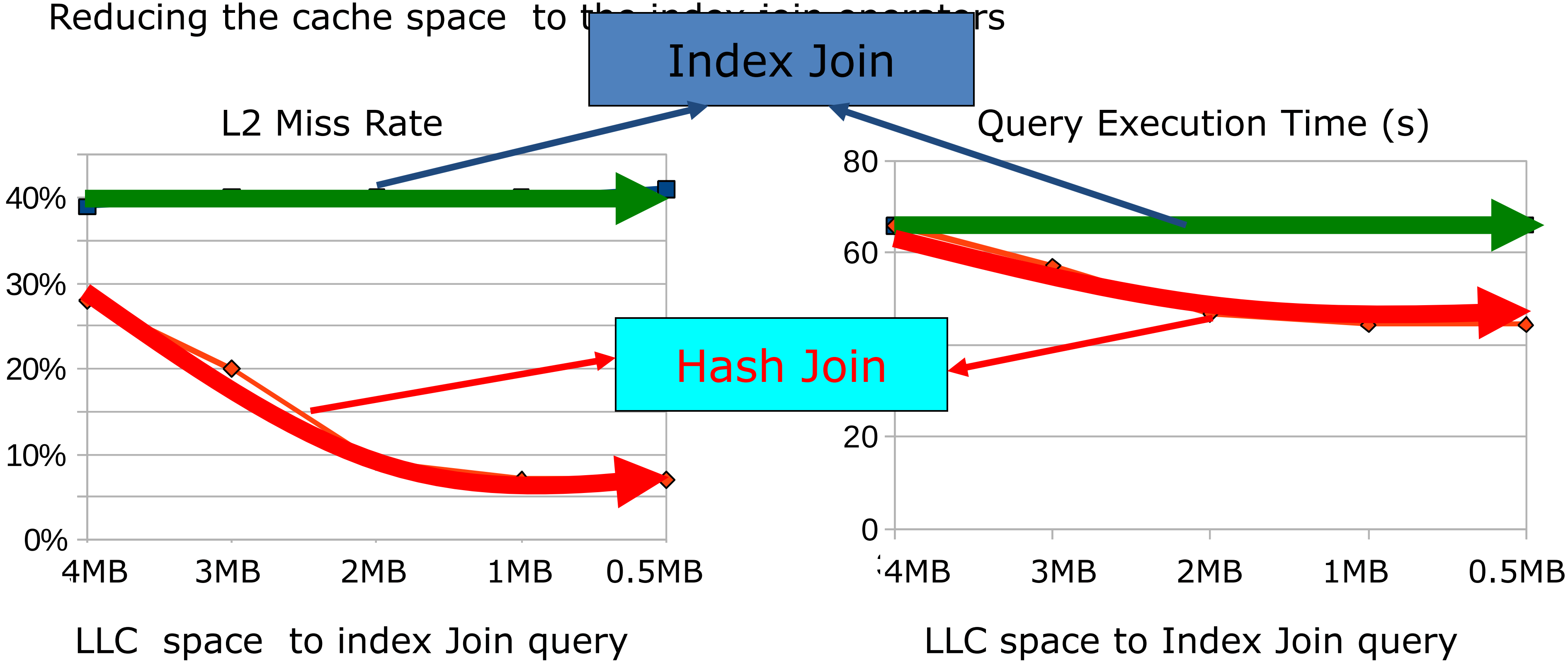
(1) Eliminate cache pollution by limiting the cache space for weak-locality queries

(2) Avoid capacity contention by allocating space to each query according to their need.

The Effectiveness of Cache Partitioning

co-running a hash join (strong/moderate locality) and an index join (weak locality)

Reducing the cache space to the index join operators

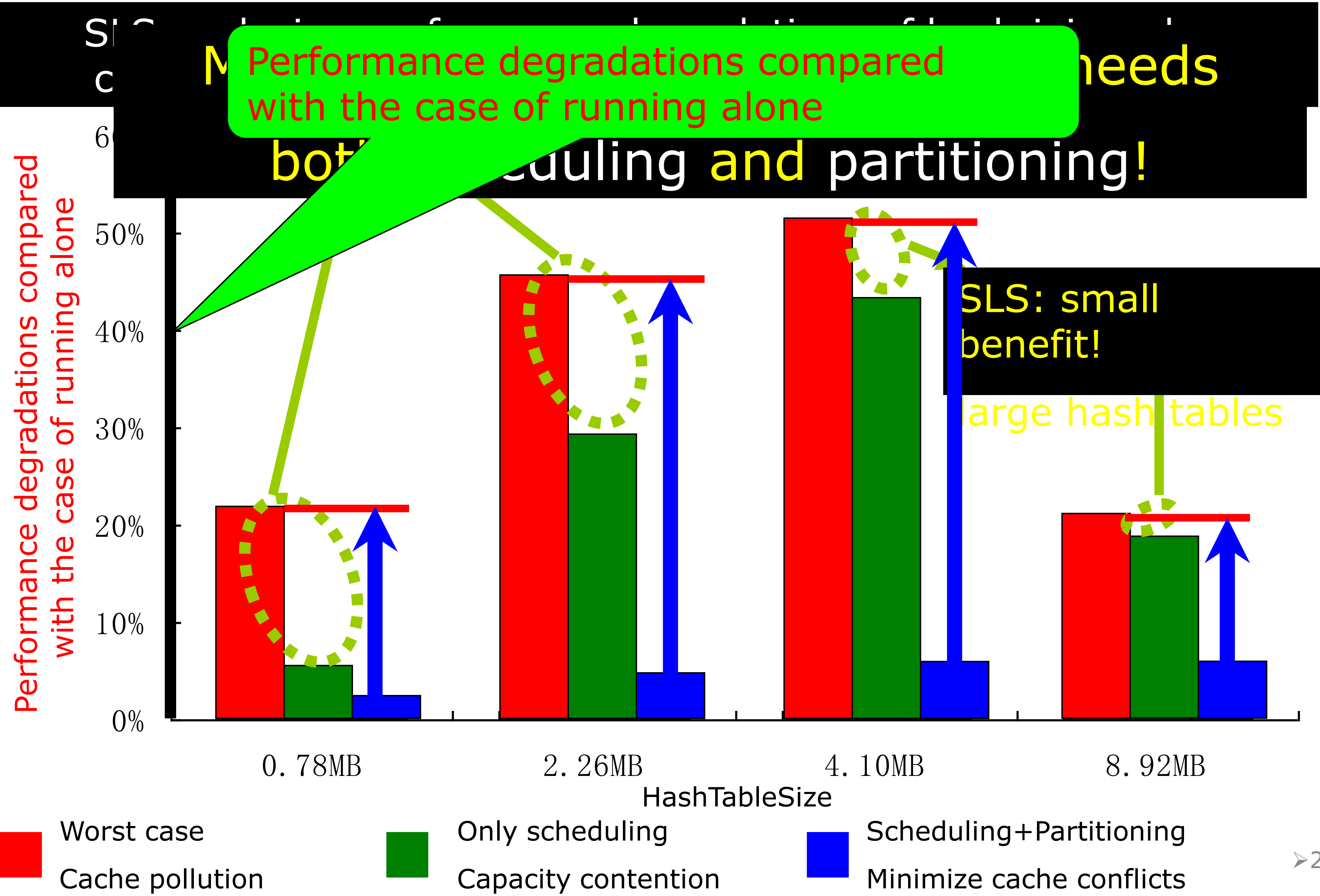


Cache Partitioning

Maxmizing performance of strong/moderate-locality query without slowing down of the weak-locality query

SLS (DB scheduling only) vs MLS (DB scheduling OS partitioning)

co-running hash joins with different hash table sizes and index joins



Summary

- Multicore Shared LLC is out of the DBMS management
  - Causing cache pollution related conflicts
  - Under- or over-utilizing cache space
  - Significantly degrading overall DBMS execution performance
- MCC-DB makes collaborative efforts between DBMS & OS
  - Make query plans with mixed locality strengths
  - Schedule co-running queries to avoid capacity and conflict misses
  - Allocating cache space according to demands
  - All decisions are locality centric
- Effectiveness is shown by experiments on warehouse DBMS

- MCC-DB methodology and principle can be applied to a large scope of data-intensive applications

Thank You !