# Cache Management in Multicore Processors

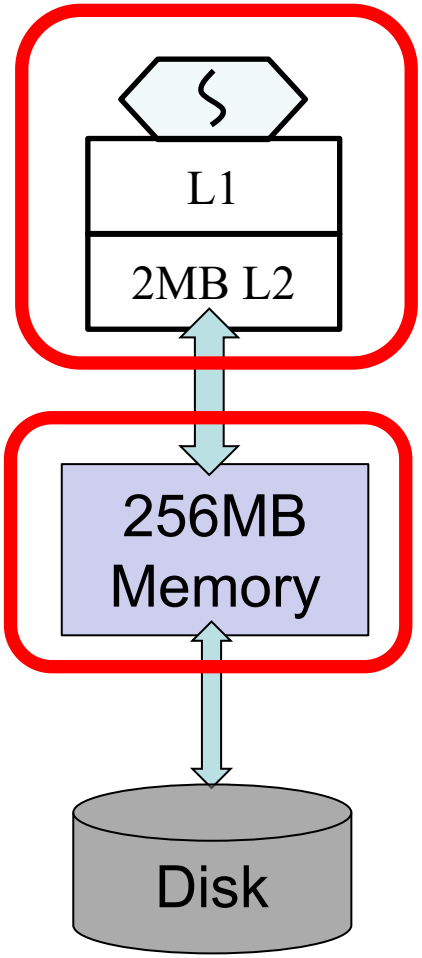
*Software and hardware Approaches*

*Xiaodong Zhang*

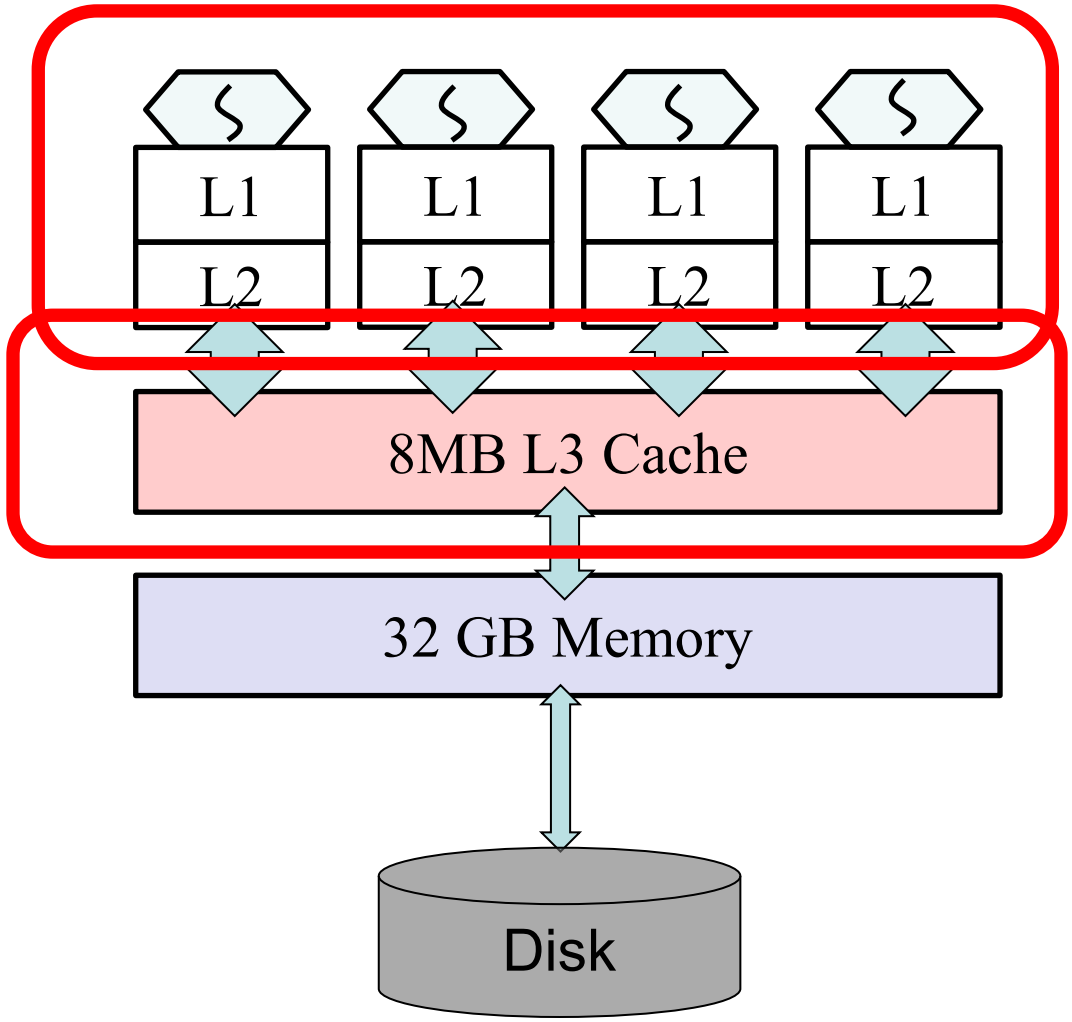
CSE 3421 additional notes for Memory Systems

# Impact of Multicore Procesors in Computer Systems

Dell Precision GX620
Purchased in 2004



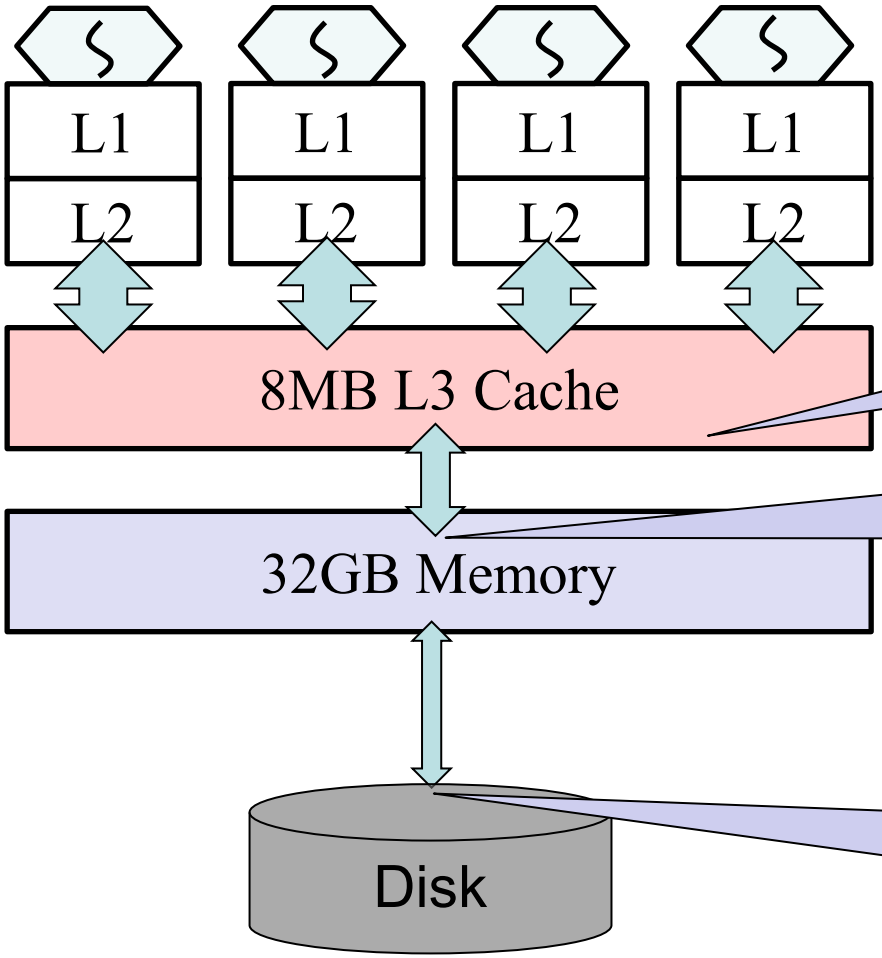

Dell Precision 7730
Purchased in 2018 with similar price



# Performance Issues w/ the Multicore Architecture

Slow data accesses to memory and disks continue to be major bottlenecks.

**All the CPUs in Top-500 Supercomputers are multicores.**

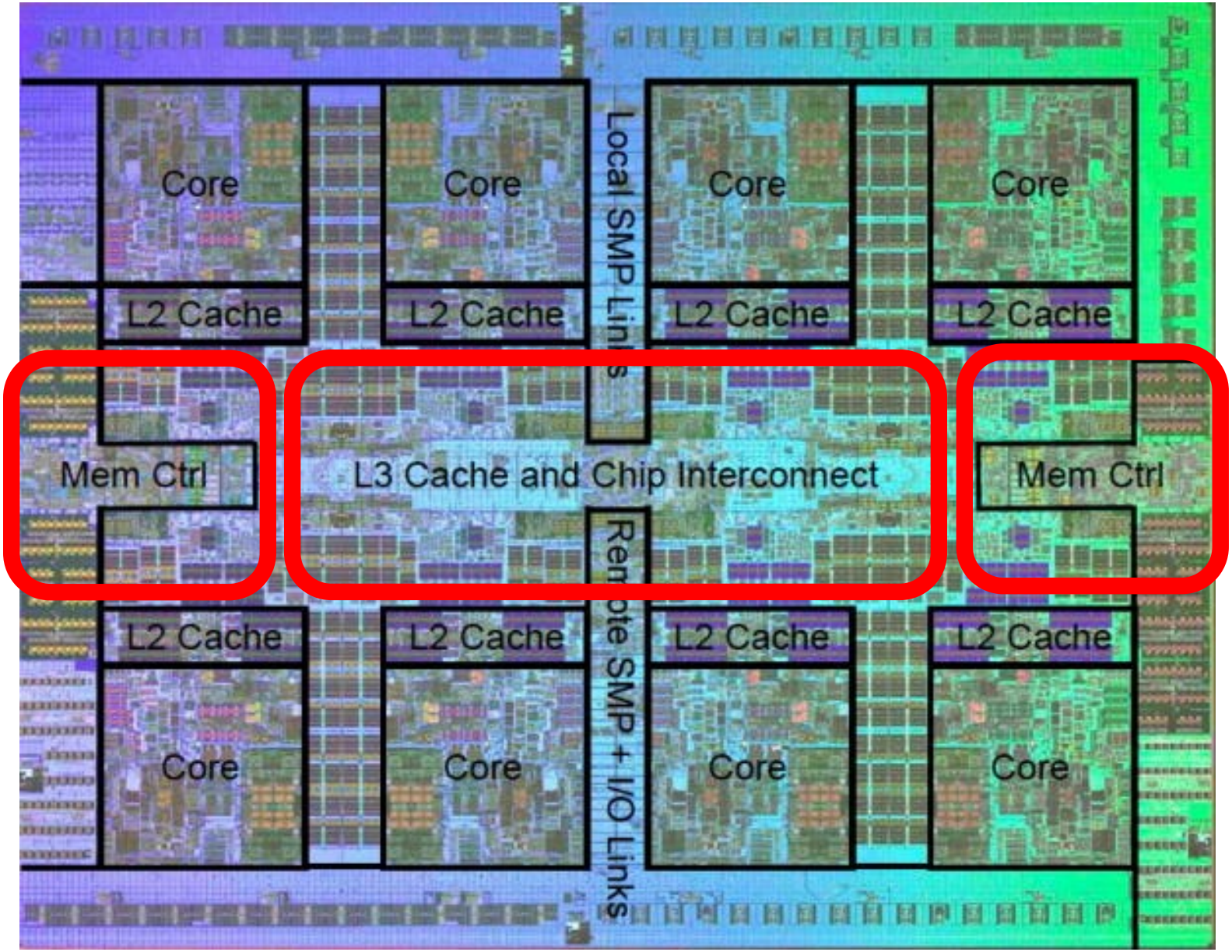



**Cache Contention and Pollution:**
Conflict cache misses among multi-threads can significantly degrade performance.

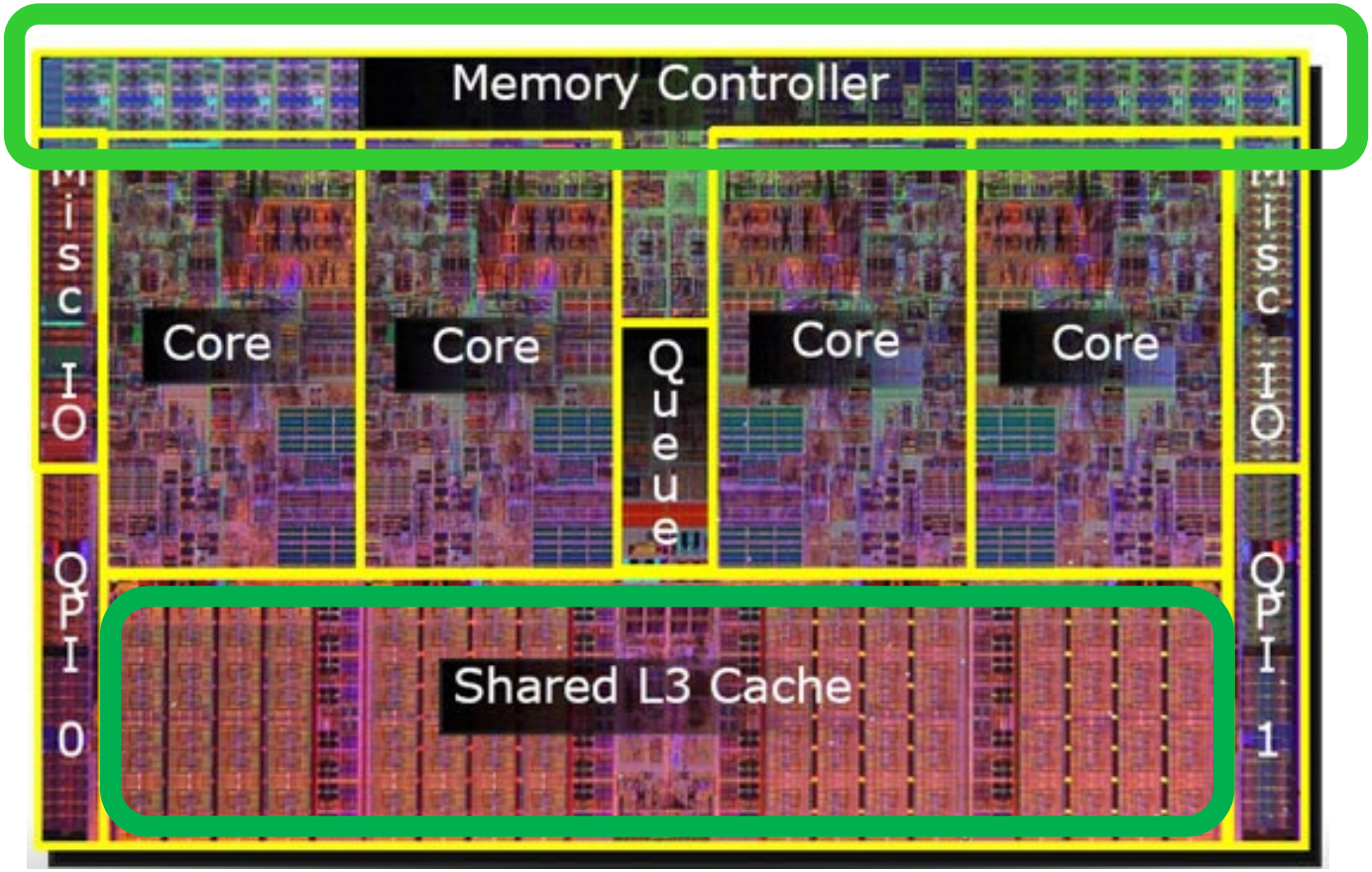
**Memory Bus Congestion:**
Bandwidth is limited to as the number of cores increases

**"Disk Wall":**
Multicores also demand high throughput from disks.

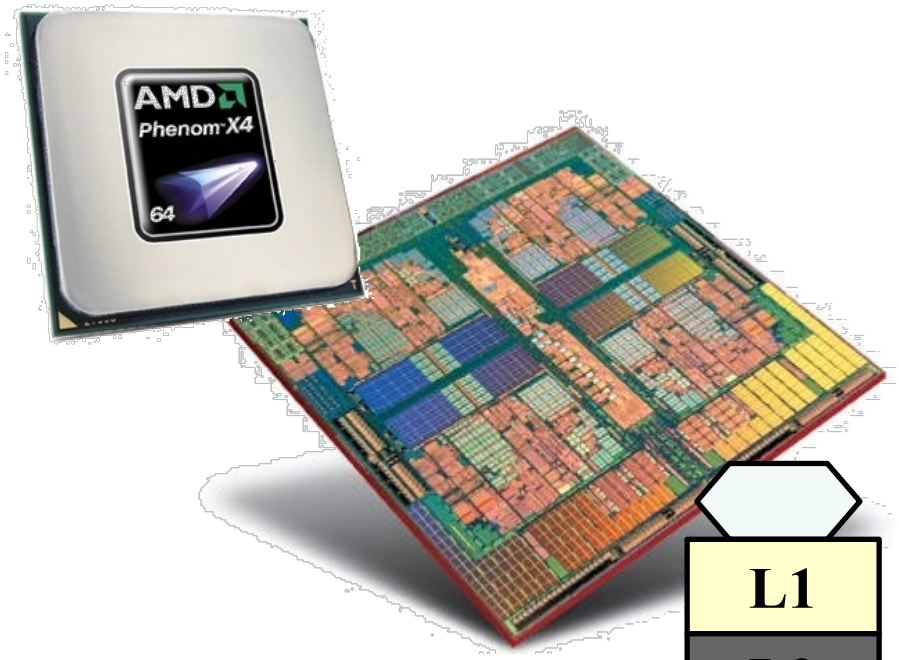
# IBM Power 7: Shared Last Level Cache (LLC)

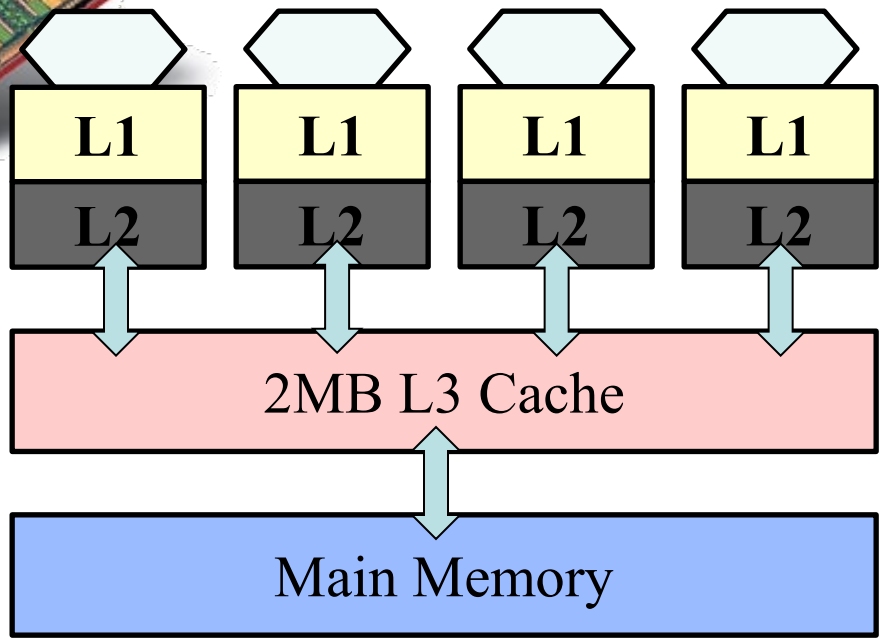


# Intel Core i7: Shared Last Level Cache (LLC)

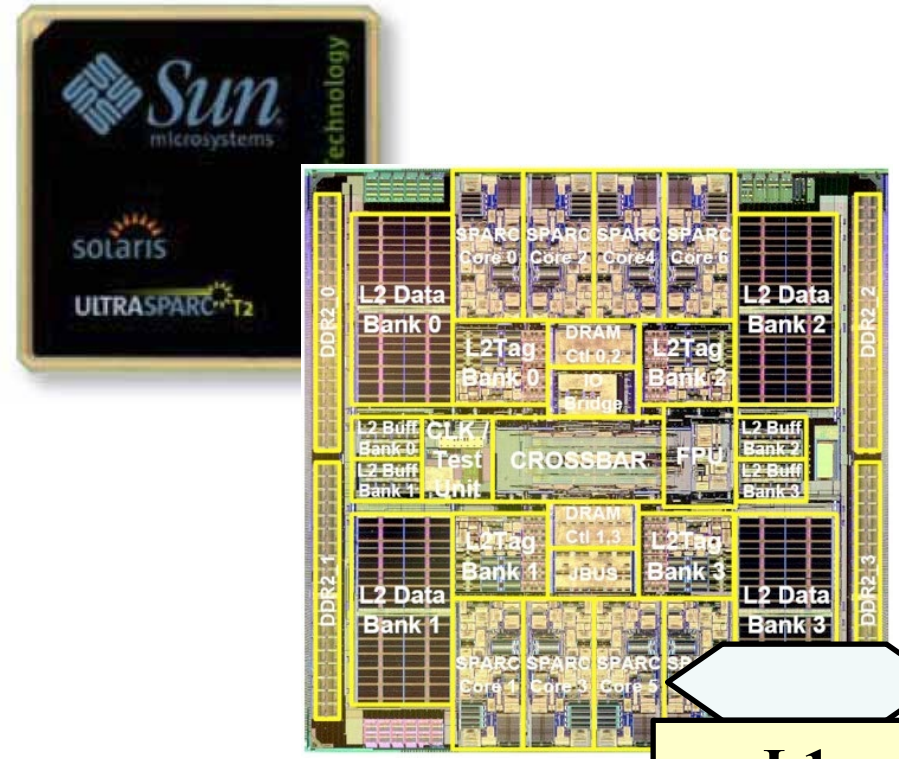
# AMD Phenom X4: Shared Last Level Cache (LLC)

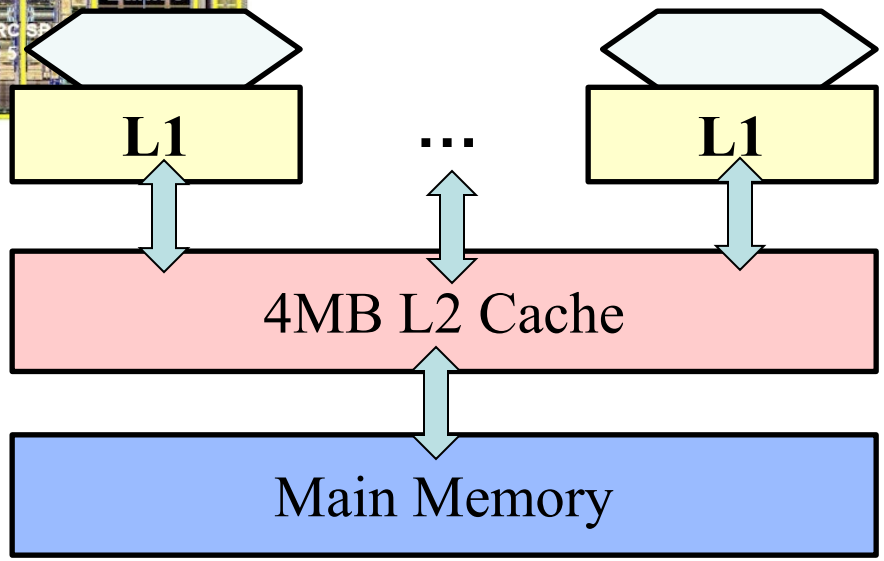



4 Cores share a 2MB last level cache (LLC) and data path

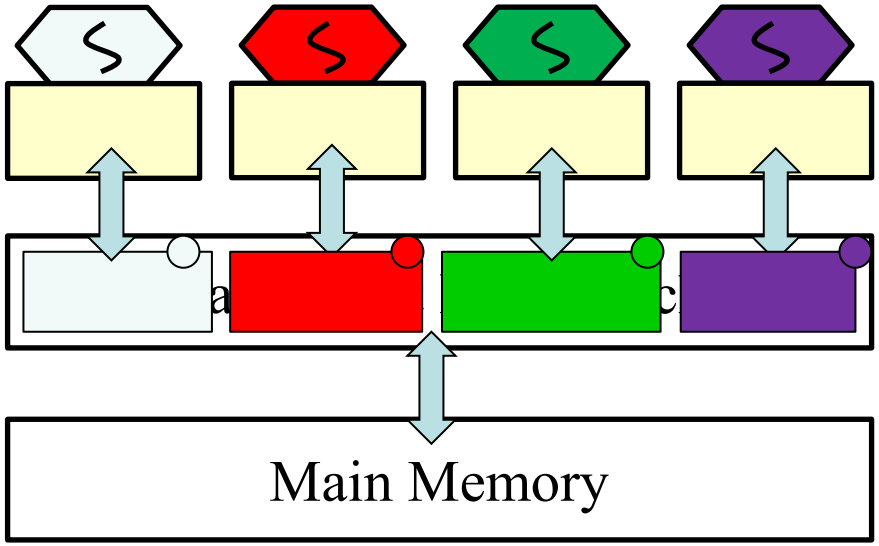
# Sun Niagara T2: Shared Last Level Cache (LLC)



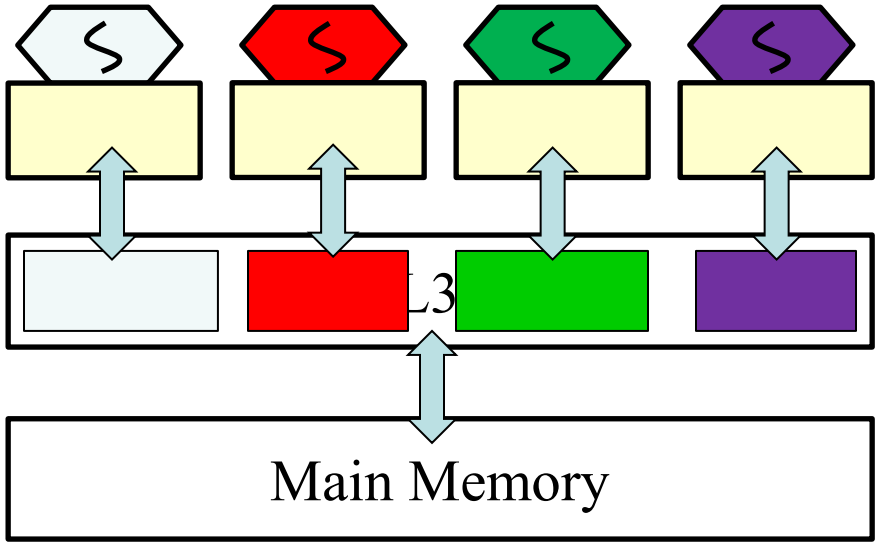

8 Cores share a 4MB last level cache (LLC) and data path

# Structure of General-Purpose Multicores



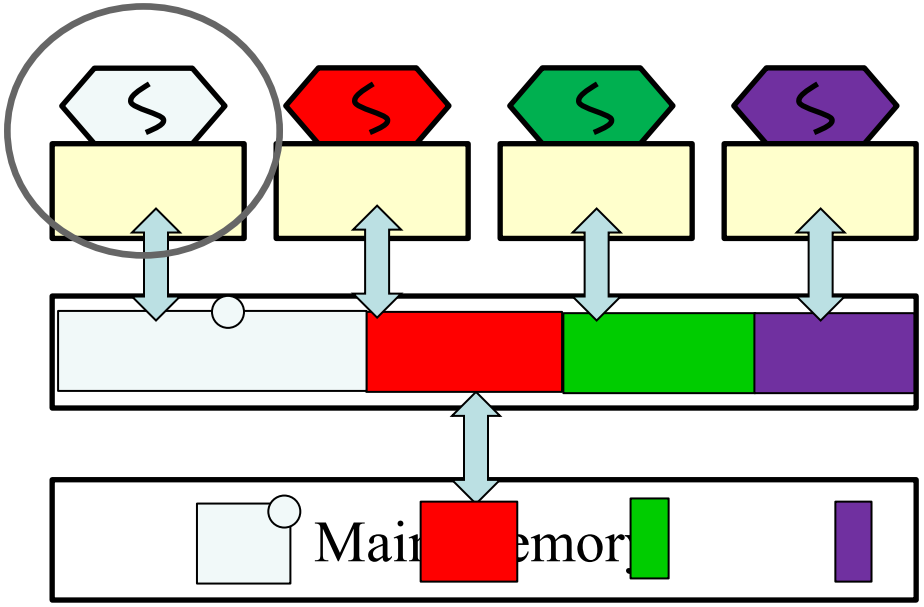

- Last level cache (LLC) and the connection to memory (e.g. memory bus and controller) are shared among multiple cores.
- Each thread needs a working set (a necessary size data set) to run
- Memory latency is order(s) of magnitude higher than cache access times
  - Accesses to LLC are from concurrent/parallel threads
  - Multiple working sets can be independent or dependent
  - High LLC hits if working sets do not conflict to each other

# Access Conflicts in LLC Happen without a Control



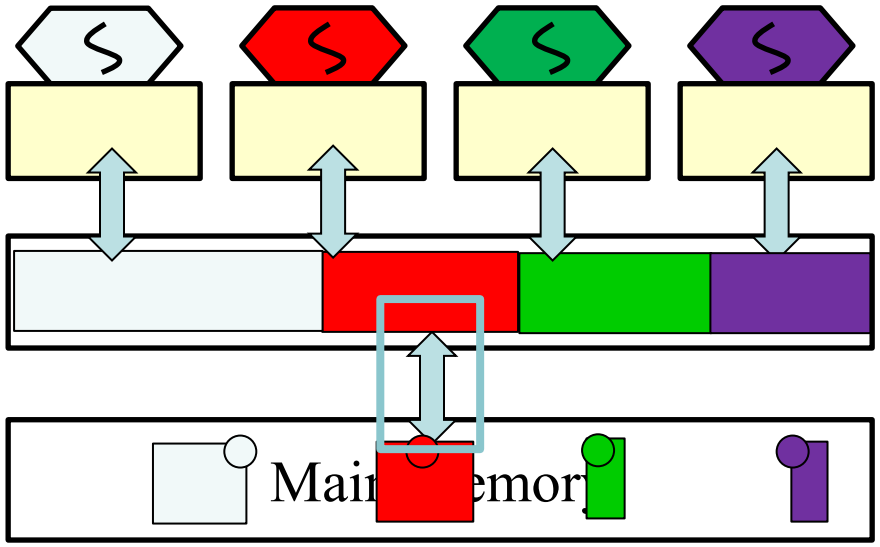

- LLC conflicts: accumulated working set size is larger than LLC.

# Access Conflicts in LLC happen without a control



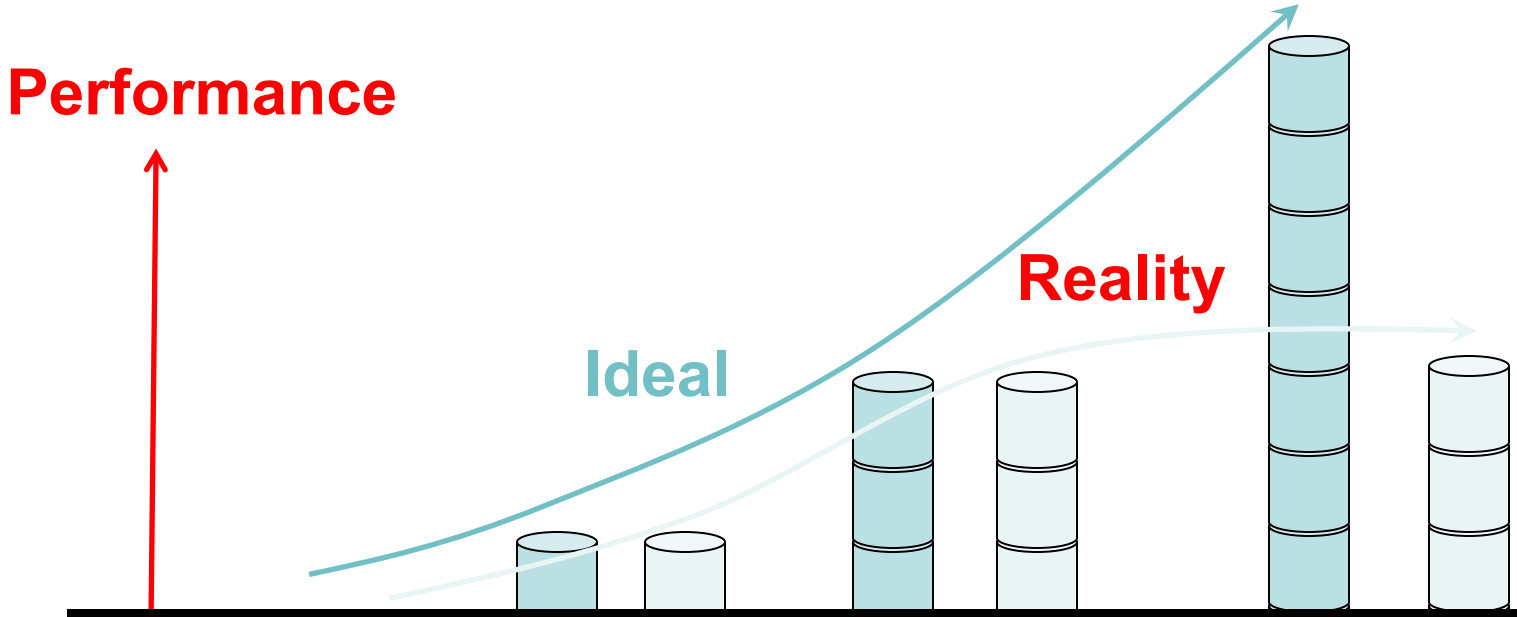

- LLC conflicts: accumulated working set size is larger than LLC.
- Evicting some frequently used data sets from LLC.
- Performance degradation due to increased memory accesses
  - Increasing access latency to victim threads

# Access Conflicts in LLC Happen without a control



- LLC conflicts: accumulated working set size is larger than LLC
- Evicting some frequently used data sets from LLC
- Performance degradation due to increased memory accesses
  - Increasing access latency to victim threads
  - Contention in memory bus further increases memory access latency

# Multicore Cannot Deliver Expected Performance as It Scales

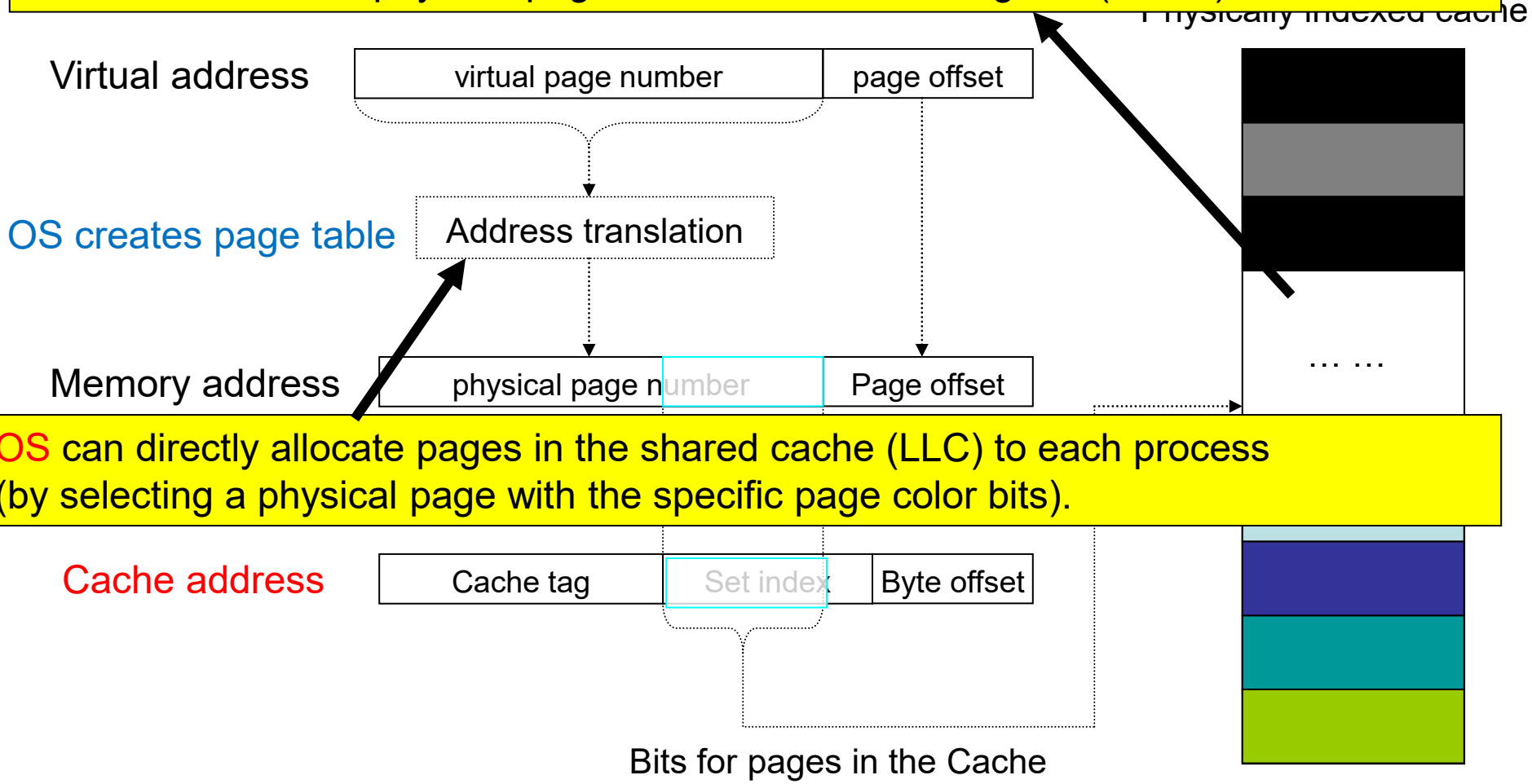



**Need effective mechanisms to control and handle inter-thread access contention in LLC**

"The Troubles with Multicores", David Patterson, IEEE Spectrum, July, 2010
"Finding the Door in the Memory Wall", Erik Hagersten, HPCwire, Mar, 2009
"Multicore Is Bad News For Supercomputers", Samuel K. Moore, IEEE Spectrum, Nov, 2008

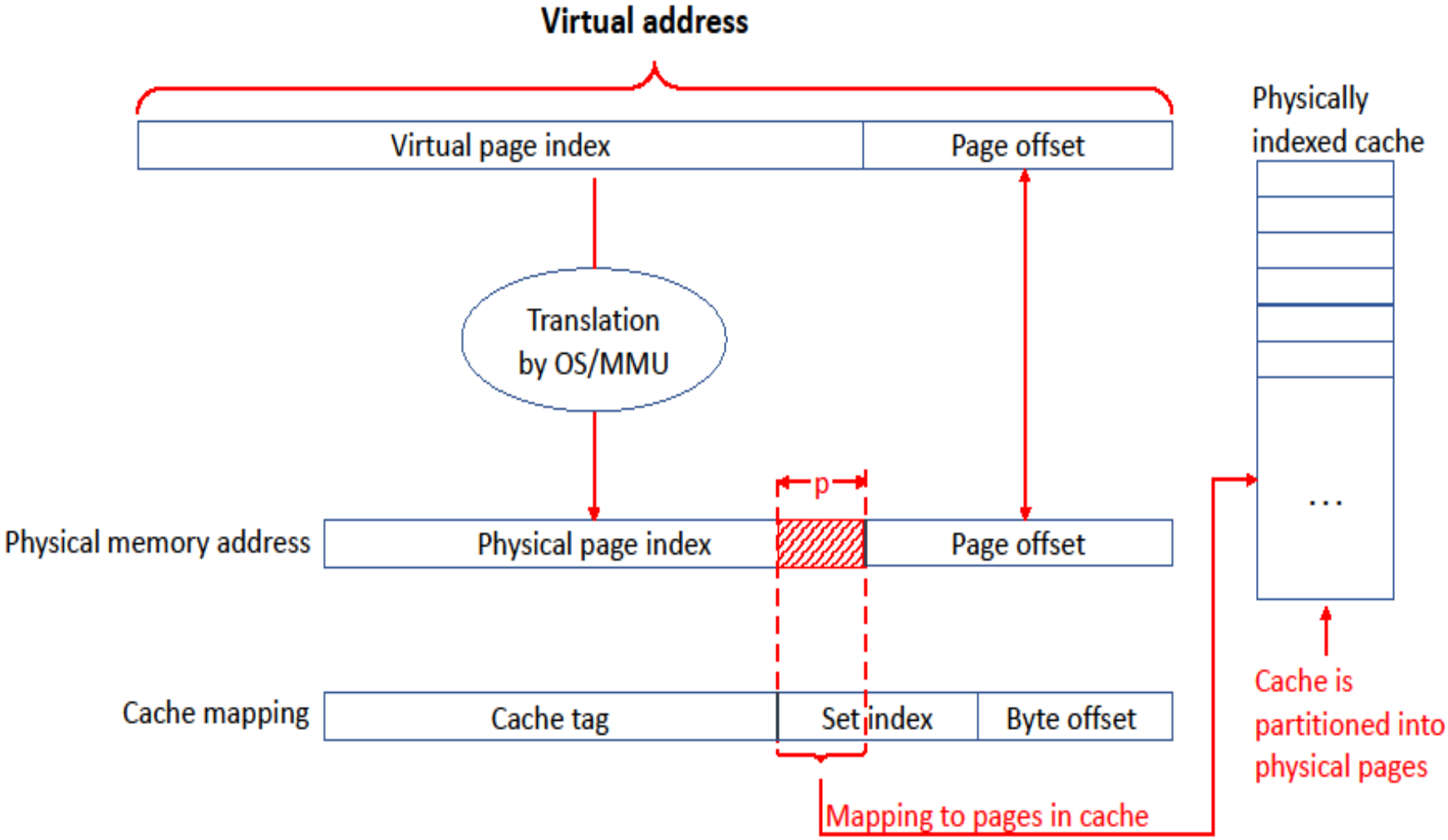
# OS Cache Partitioning in Multicores

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

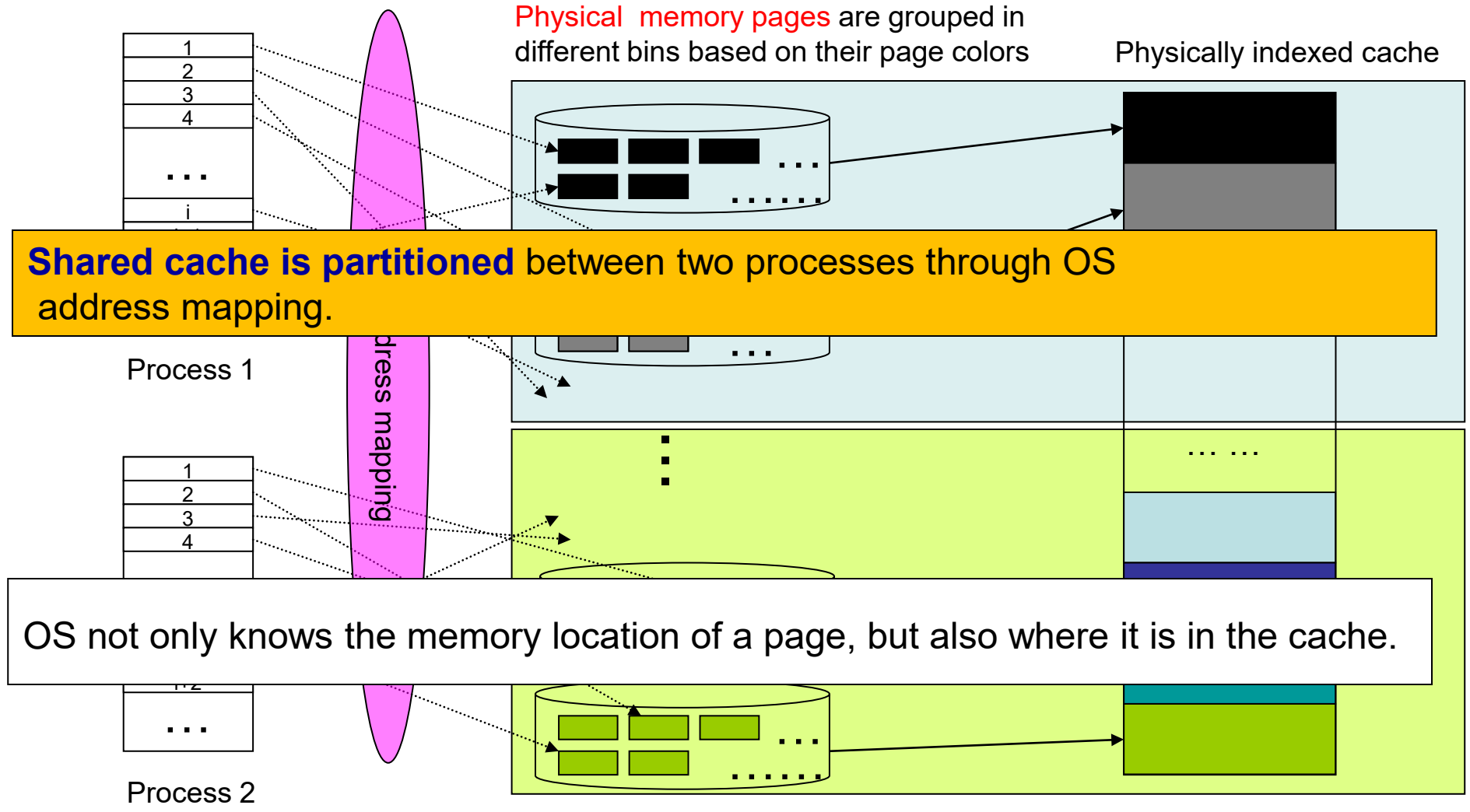



OS can directly allocate pages in the shared cache (LLC) to each process (by selecting a physical page with the specific page color bits).

# The key for OS to Directly Manage Hardware Cache



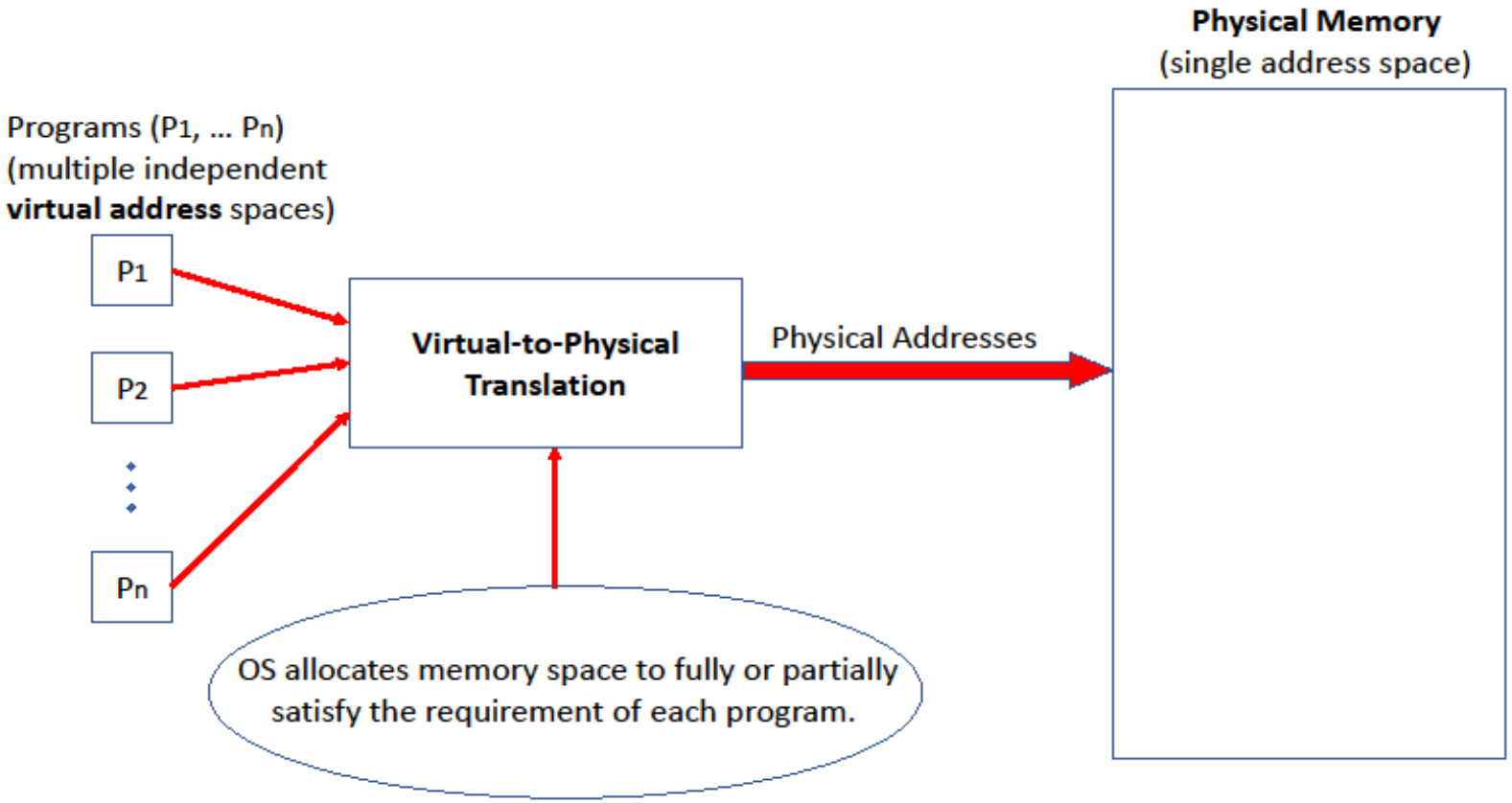
# OS can partition the cache into multiple regions




**Shared cache is partitioned** between two processes through OS address mapping.

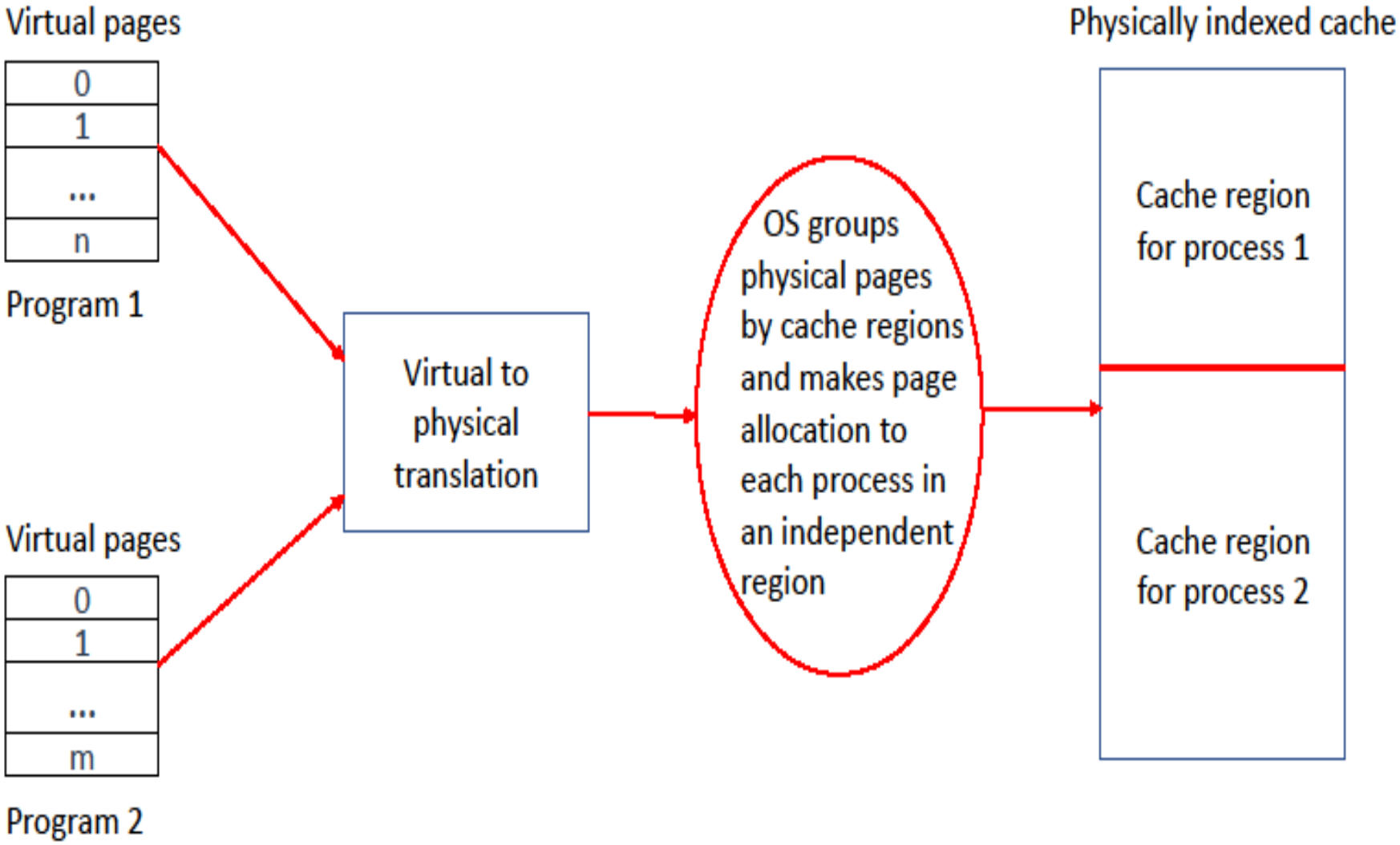
OS not only knows the memory location of a page, but also where it is in the cache.

# Virtual Memory for multiprogramming (review)




- Multiple programs want to run in a computer (competing in the same area)
- OS knows the program/data space and create a page table for each program (table assignment by reservation)
- Not all the data sets are moved to memory (most hungry guests first)
- Running the program (an usher leads the group to the table)

# The OS Capability for Hardware  Cache Management



# OS-Based Cache Partitioning (HPCA'08)

- Static cache partitioning
  - Predetermines the amount of cache blocks allocated to each process at the beginning of its execution
  - Divides shared cache to multiple regions and partition cache regions through OS page address mapping
- Dynamic cache partitioning
  - Adjusts cache quota among processes dynamically
  - Page re-coloring
  - Dynamically changes processes' cache usage through OS page address re-mapping
- Both are implemented in LINUX and open sourced

# The paper in High Performance Computer Architecture (HPCA'08)

## Gaining Insights into Multicore Cache Partitioning: Bridging the Gap between Simulation and Real Systems

Jiang Lin[1], Qingda Lu[2], Xiaoning Ding[2], Zhao Zhang[1], Xiaodong Zhang[2] and P. Sadayappan[2]
[1]Dept. of Electrical and Computer Engineering
Iowa State University
Ames, IA 50011
{linj,zzhang}@iastate.edu

[2] Dept. of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{luq,dingxn,zhang,saday}@cse.ohio-state.edu

### Abstract

*Cache partitioning and sharing is critical to the effective utilization of multicore processors. However, almost all existing studies have been evaluated by simulation that often has several limitations, such as excessive simulation time, absence of OS activities and proneness to simulation inaccuracy. To address these issues, we have taken an efficient software approach to supporting both static and dynamic cache partitioning in OS through memory address mapping. We have comprehensively evaluated several representative cache partitioning schemes with different optimization objectives, including performance, fairness, and quality of service (QoS). Our software approach makes it possible to run the SPEC CPU2006 benchmark suite to completion. Besides confirming important conclusions from previous work, we are able to gain several insights from whole-program executions, which are infeasible from simulation. For example, giving up some cache space in one program to help another one may improve the performance of both programs for certain workloads due to reduced contention for memory bandwidth. Our evaluation of previously proposed fairness metrics is also significantly different from a simulation-based study.*

*The contributions of this study are threefold. (1) To the best of our knowledge, this is a highly comprehensive execution- and measurement-based study on multicore cache partitioning. This paper not only confirms important conclusions from simulation-based studies, but also provides new insights into dynamic behaviors and interaction effects. (2) Our approach provides a unique and efficient option for evaluating multicore cache partitioning. The implemented software layer can be used as a tool in multicore performance evaluation and hardware design. (3) The proposed schemes can be further refined for OS kernels to improve performance.*

### 1. Introduction

Cache partitioning and sharing is critical to the effective utilization of multicore processors. Cache partitioning usually refers to the partitioning of shared L2 or L3 caches among a set of programming threads running simultaneously on different cores. Most commercial multicore processors today still use cache designs from uniprocessors, which do not consider the interference among multiple cores. Meanwhile, a number of cache partitioning methods have been proposed with different optimization objectives, including performance [17, 11, 5, 2], fairness [8, 2, 12], and QoS (Quality of Service) [6, 10, 12].

Most existing studies, including the above cited ones, were evaluated by simulation. Although simulation is flexible, it possesses several limitations in evaluating cache partitioning schemes. The most serious one is the slow simulation speed – it is infeasible to run large, complex and dynamic real-world programs to completion on a cycle-accurate simulator. A typical simulation-based study may only simulate a few billion instructions for a program, which is equivalent to about one second of execution on a real machine. The complex structure and dynamic behavior of concurrently running programs can hardly be represented by such a short execution. Furthermore, the effect of operating systems can hardly be evaluated in simulation-based studies because the full impact cannot be observed in a short simulation time. This limitation may not be the most serious concern for microprocessor design, but is becoming increasingly relevant to system architecture design. In addition, careful measurements on real machines are reliable, while evaluations on simulators are prone to inaccuracy and coding errors.

**Our Objectives and Approach** To address these limitations, we present an execution- and measurement-based study attempting to answer the following questions of concern: (1) Can we confirm the conclusions made by the simulation-based studies on cache partitioning and sharing

978-1-4244-2070-4/08/$25.00 ©2008 IEEE

# Impact of the Work

- Intel Software and Service Group (SSG) has adopted the OS-based cache partitioning methods (static and dynamic) as software solutions to manage the multi-core shared cache.

- The solution has been merged into a production system in a major automation industry (multicore-based motion controller)

- The software cache partitioning is becoming a standard method in any OS for multicores, such as Windows

# An Acknowledgment Letter from Intel

Jiang Lin, Ph.D
[redacted]

Dear Dr. Lin,

The software cache partitioning approach and a set of algorithms proposed by you and your collaborators helped our engineers implement a solution that provided 1.5x latency reduction in a custom Linux stack running on multi-core Intel platforms. The solution has been adopted by a major industrial automation vendor and facilitated the deployment on multi-core Intel platforms.

A paper documenting the research results titled as "Gaining Insights into Multicore Cache Partitioning: Bridging the Gap between Simulation and Real Systems" was presented and published in the 14th International Symposium on High-Performance Computer Architecture (HPCA'08). The authors of this paper are Jiang Lin, Qingda Lu, Xiaoning Ding, Zhao Zhang, Xiaodong Zhang and P. Sadayappan. This paper demonstrates how to effectively manage shared hardware caches in multi-core platforms with an advanced operating system technique, in order to improve system performance and provide better quality of service.

Thank you for your strong contribution, technical insights, and kind support!

Yours Sincerely,

Wolfgang Petersen
Director
EMEA SSG - Developer Relations Division

Intel GmbH
Dornacher Strasse 1
85622 Feldkirchen/München
Germany
Tel. +49 (0) 89/99143 0
Fax +49 (0) 89/9043948
Internet www.intel.de

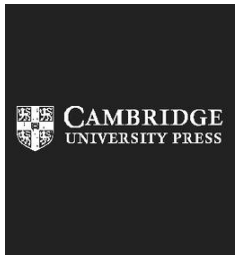
# Quotes from the Intel Letter

- The software cache partitioning approach and a set of algorithms helped our engineers implement a solution that provided 1.5 times latency reduction in a custom Linux stack running on multi-core Intel platforms.

- This solution has been adopted by a major industrial automation vendor and facilitated the deployment on multi-core platforms.

- Thanks for your strong contribution, technical insights, and kind support!

# A New Textbook Merging Architecture, Data and Systems

# Some quotes from the Preface

## Preface

### The Data-Centric Computing Time

Dramatic changes and fast advancement have been made in the computing field. The first stage of computing was computation focused (or for number crunching), which started from the first two independent digital computers in the world in 1930s: the ABC machine [17] built by John Atanasoff and Clifford Berry at the Iowa State University in USA and the Zuse machines [21] built by Konrad Zuse in Berlin, Germany. Several key design ideas were used in ENIAC that was the first operational computer in the world in 1943 [16]. After then, computer hardware design has been unified under the von Neumann model [72]. With the advancement of the semiconductors and VLSI technologies, the growth of CPU chip performance had followed Moore's Law [15] until 2015. During this period of time, operating systems, databases, scientific computing libraries became mature to support important applications in the society. In early 1970s, the development of the second stage of computing began, which is the networking. The tagline of "The Network is the Computer" by Sun Microsystem in the mid-1980s reflected the ambition and the goal in the second stage of computing. Today, almost every digital device is connected in the Internet directly or indirectly for various types computing applications. After the great accomplishments in computer hardware, system software, application software and networks in both wired and wireless connections, we have inevitably entered the third stage of computing, which is data-centric computing. The title of a Google Book "The Datacenter as a Computer" [6] reflects the major roles of computing in this stage.

### The Requirement of Understanding the Interactions

After the R&D of computing for more than 80 years, any computer science subject, such as algorithms and data structures, computer architecture, operating systems, databases, and networking, are hardly independent and self-contained. This is because Moore's Law improved performance of computing for all applications at the circuit level. In contrast, in the post-Moore's Law era, highly efficient execution for any applications, such as databases and machine learning, is no longer machine independent; but comes from a balance among three

# Some quotes from the Preface (continued)



critical factors: low algorithm complexity, low data movement, and high parallelism. In short, as computer scientists, a key issue we address today is how to tailor our applications to best utilize rich resources provided by systems software and hardware architecture. To accomplish our goal, we must first understand the anatomies of each of the three entities: applications, computer architectures, and software systems, and then seek for "sweet spots" to obtain optimal and desirable performance effects subject to acceptable overhead at a low cost. The performance improvement in the post Moore's Law era demands significant efforts for computer scientists to reconstruct the existing "one-size fit all" ecosystem by an interactive approach cross multiple areas in applications, systems software and hardware architecture. However, our computer science classroom teaching is still subject-focused. For example, we teach our students in an Algorithm class to reduce the algorithm complexity with the big-O notation by counting the number of computing operations, assuming that data movement involved in the algorithm is cost-free. In practice, a proficient programmer may NOT select an algorithm with the lowest complexity, but select an algorithm that has high parallelism and high locality for them to best utilize powerful and diverse architecture resources to achieve highest performance at a low cost. Most textbooks in different subjects are written from "domain perspectives". For example, database textbooks are mainly focused on the design and implementation at a logical level. Operating system and computer architecture textbooks are mainly focused on describing how to design the hardware and how to implement systems software to gain the critical path execution efficiency and to gain the effectiveness of resource management, which reflect the "infrastructure builder perspectives".

## The Goal of This Book

The three key words in the title of this textbook: **data**, **architecture**, and **systems**, reflect the unique aim of my writing, namely to knock down the walls that separate the three closely related subjects in classrooms. Having taught and conducted research in computer science for many years, I have always been interested in looking into data movement behavior of various applications at runtime in computer architecture, software systems, distributed systems, and in databases. With my students and collaborators, I have also proposed methods to maximize parallelism and to minimize the data movement based on the identified critical issues. Several of the basic research results of mine have been merged into the production systems in both hardware and software. In fact, the background knowledge for these research investigations is largely in the scope of undergraduate classes of computer architecture, operating systems and databases. However, the interactive thinking among the three subjects are intensive. I have realized that we do not teach our students in this way of thinking. For example, the virtual memory management studied in the operating system class is somehow independent of the cache structure in a computer architecture class although

# Some quotes from the Preface (continued)

… In fact, the background knowledge for these research investigations is largely in the scope of undergraduate classes of computer architecture, operating systems and databases. However, the interactive thinking among the three subjects are intensive. I have realized that we do not teach our students in this way of thinking.

# Some quotes from the Preface (continued)

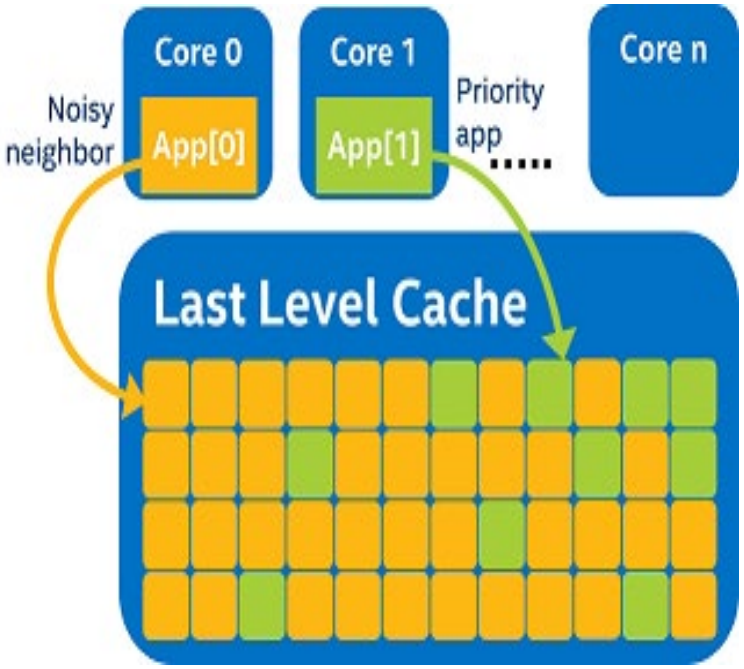
For example, the virtual memory management studied in the operating system class is somehow independent of the cache structure in a computer architecture class although they are closely related to each other in the same memory space. If we had bridged the two subjects of cache management in hardware architecture and memory management in OS, students would have learned that OS is also able to determine the space allocation in the hardware cache for each process in a page unit, avoiding conflicts in LLC in multicores.

# Some quotes from the Preface (continued)

My teaching experiences and research activities have motivated me to write an unconventional book to cover these three areas in a linked and interactive way. I hope I have some "latecomer advantage" for writing this book due to rich experiences and lessons I have had in the three fields.

# Cache Allocation Technology (CAT) by Intel

•Intel introduced CAT to allow users to partition the shared cache in multicore in 2016



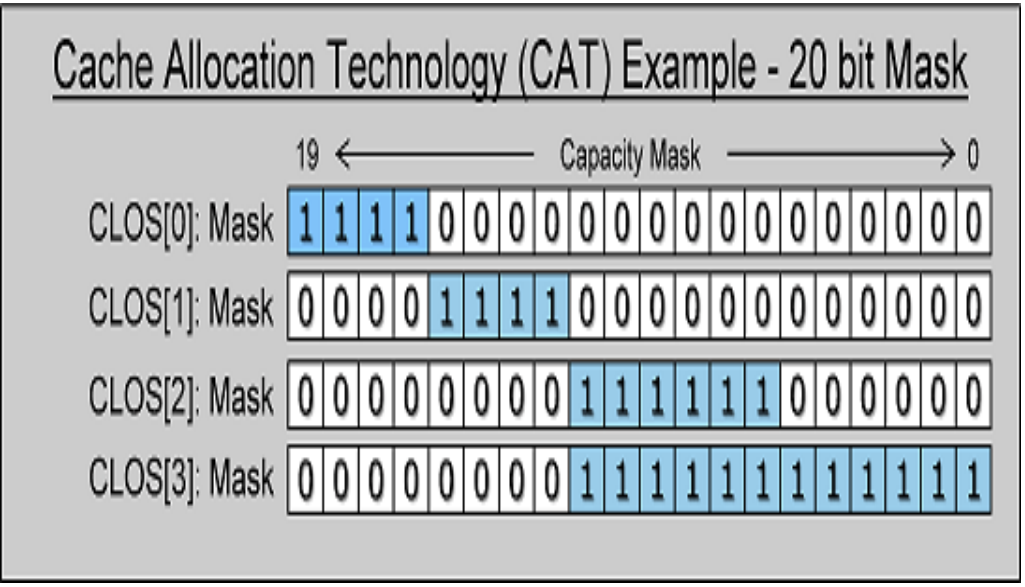

•The last level cache is a high set-associative cache with many ways.

•Process App[0] in core 0 over utilizes the shared resources

•Process App[1] in core 1 can be negatively affected

# Cache Allocation Technology (continued)

- CAT has a bit mask representing all the ways in the cache to partition the cache vertically (OS partitions cache horizontally)
- Each process manages a bit mask. bit=1 means this cache way is accessed by the process.
  - The bit mask represents a Class of Service (CLOS) for each process
- The default setting means the LLC is shared by all processes without ownership




- CLOS[0] exclusively owns 4 ways of cache
- CLOS[1] exclusively owns another 4 ways
- CLOS[2] and CLOS[3] share 6 ways of cache
- CLOS[3] exclusively owns 6 ways
- For a 32 ways of cache, certain ways are reserved for system

# Memory Address: the source of the three innovations

- A memory address points a Byte (byte addressable). Last significant bits (byte or page offset) determine the unit of accesses: a word (4-bytes), multiple words, a page, ….

- Memory Bank Offset
  - Pages (determined by the page offset) are interleaved among memory banks

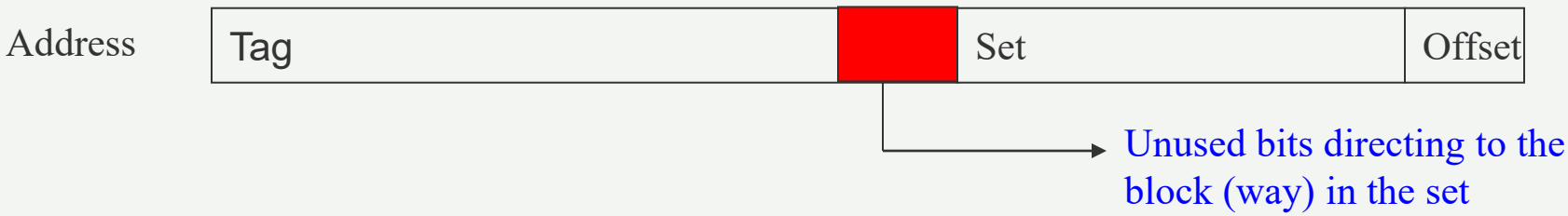
# Memory address:
# a bridge between **cache and DRAM**
# a bridge between **OS and architecture**

- Memory address is subdivided to form different types of caches
  - Direct-map cache, set associative cache, and fully associative cache
  - Multicolumn cache creates a hybrid direct/set-associative cache

- Memory address determines page allocations among M banks
  - Correlation between cache conflicts and DRAM row-buffer conflicts
  - Page interleaving permutation breaks the correlation for high hits in RB

- OS maps virtual pages to physical pages (memory addresses)
  - Relationship between cache blocks and memory pages is defined
  - OS can directly manage the cache allocation for each process to avoid the last-level-cache in multicores

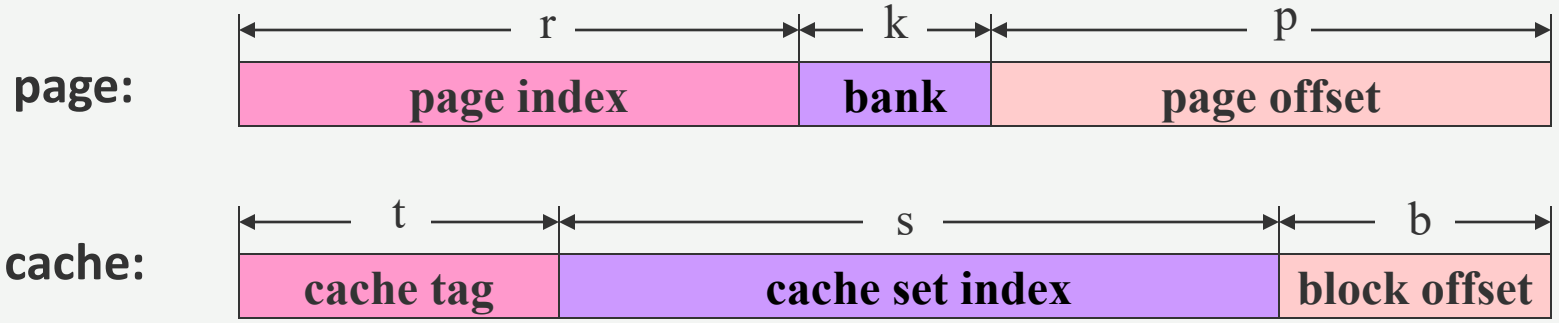
# Memory address Insight (1): multicolumn cache




- The bits given to tag and ``set bits'' generate a direct-mapped location: Major Location.
- A major location mapping = direct-mapping.

- Directing mapping within an set associative cache

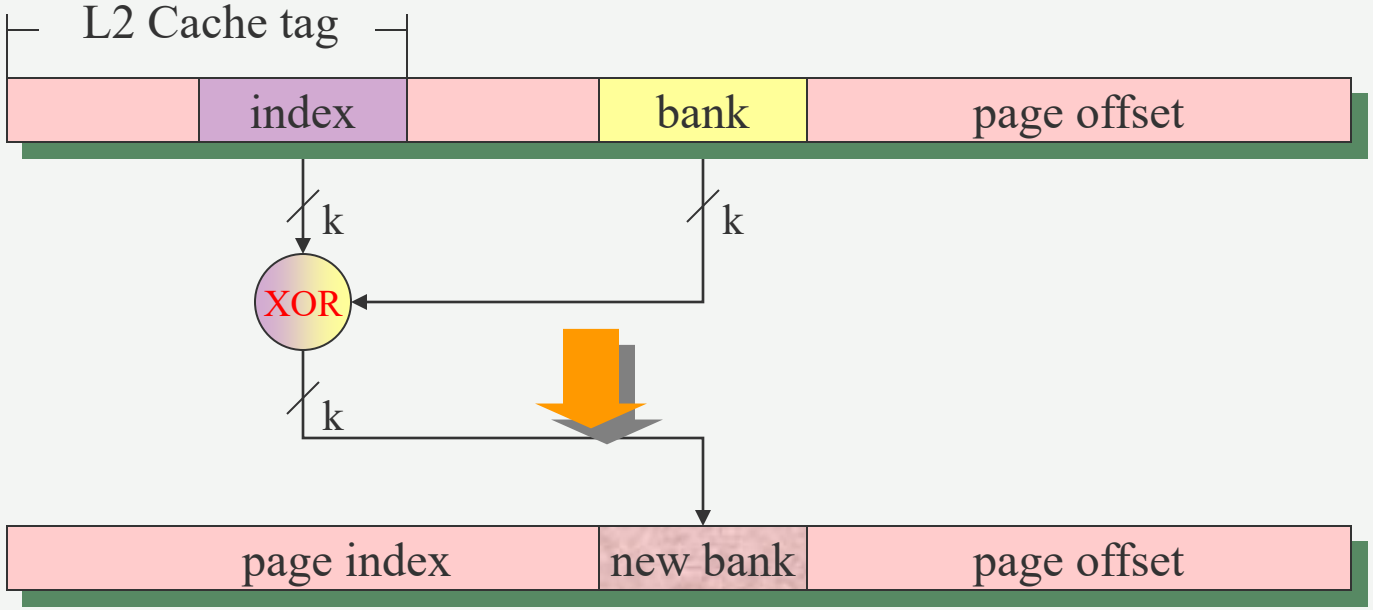
# Memory address insights (2): Conflicting misses in cache are also DRAM row-buffer conflicting misses




- cache-conflict: same cache index, different tags.
- row-buffer conflict: same bank index, different pages.
- address mapping: bank index $\subseteq$ cache set index
- Considering two addresses, x and y, and conventional mapping
- Property: $\forall x \forall y$, x and y conflict on cache $\Rightarrow$ also on row buffer.
- **There is one-to-one correspondence between cache and row buffer hits/conflicts.**

**Note:** $\forall x \forall y$ => for all x and for all y

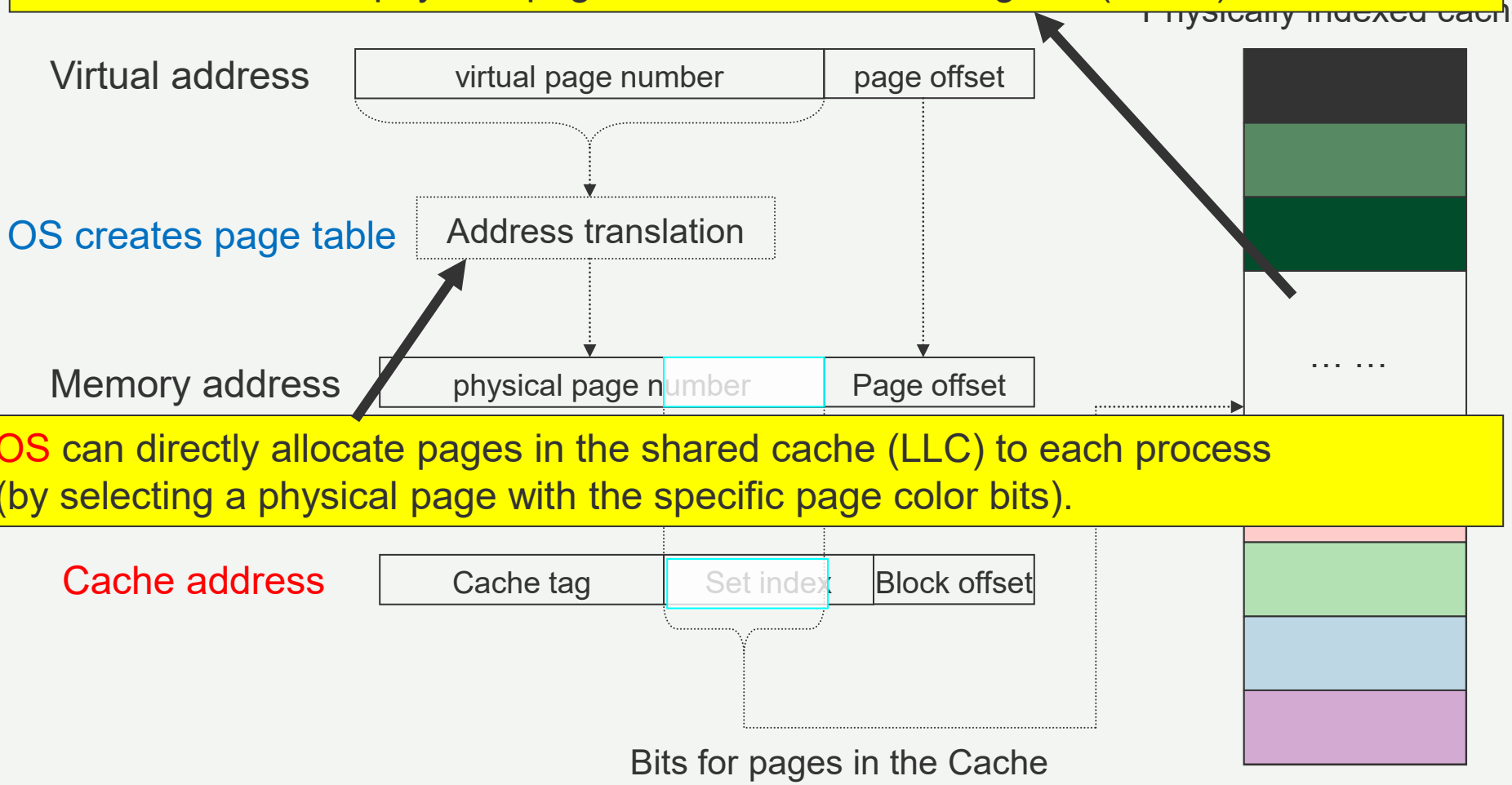
# Breaking the Symmetry by Permutation-based Page Interleaving




- The k bits of the bank index xor with last order k bits in the cache tag field
  - starting from the least significant bit
- The k bit xor result forms a new bank index
- Note: memory addresses are not changed, only new bank index is generated

# Memory address insights (3): OS partitions LLC for each process

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




OS can directly allocate pages in the shared cache (LLC) to each process
(by selecting a physical page with the specific page color bits).