A Case for NUMA-aware Contention Management on Multicore Systems

Sergey Blagodurov et al, USENIX ATC 2011

A NUMA System

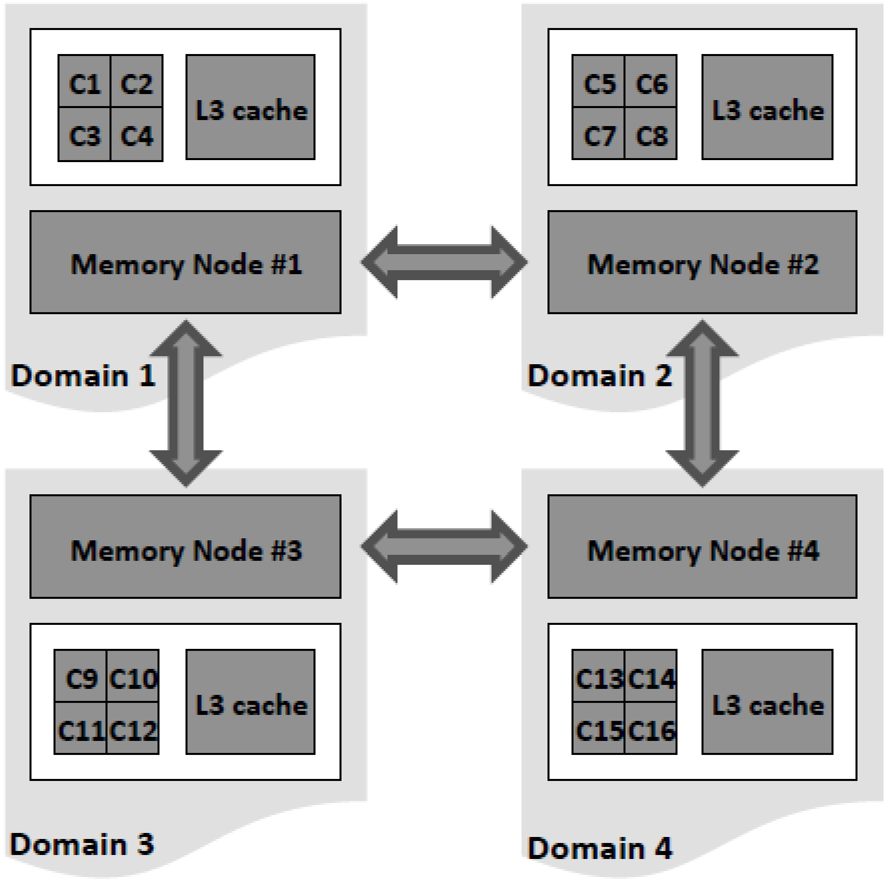


Figure 1: A schematic view of a system with four memory domains and four cores per domain. There are 16 cores in total, and a shared L3 cache per domain.

- There are multiple memory nodes, one per memory domain
- Local nodes can be accessed in less time than remote ones, and each node has its own memory controller.

Thread Migration Example

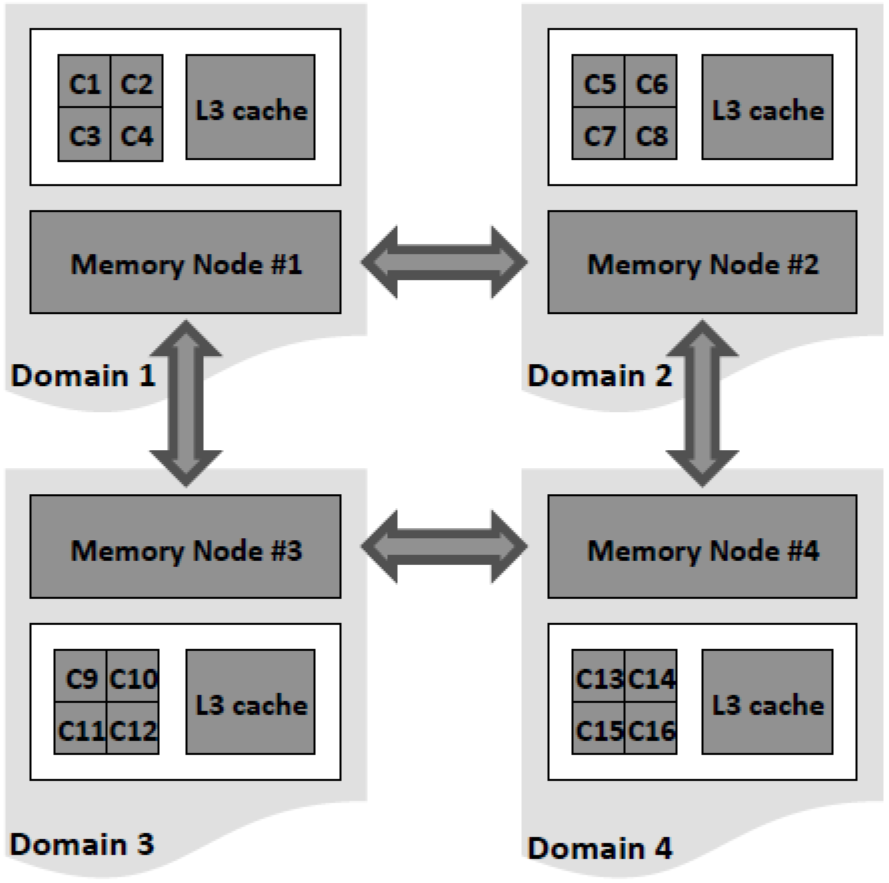
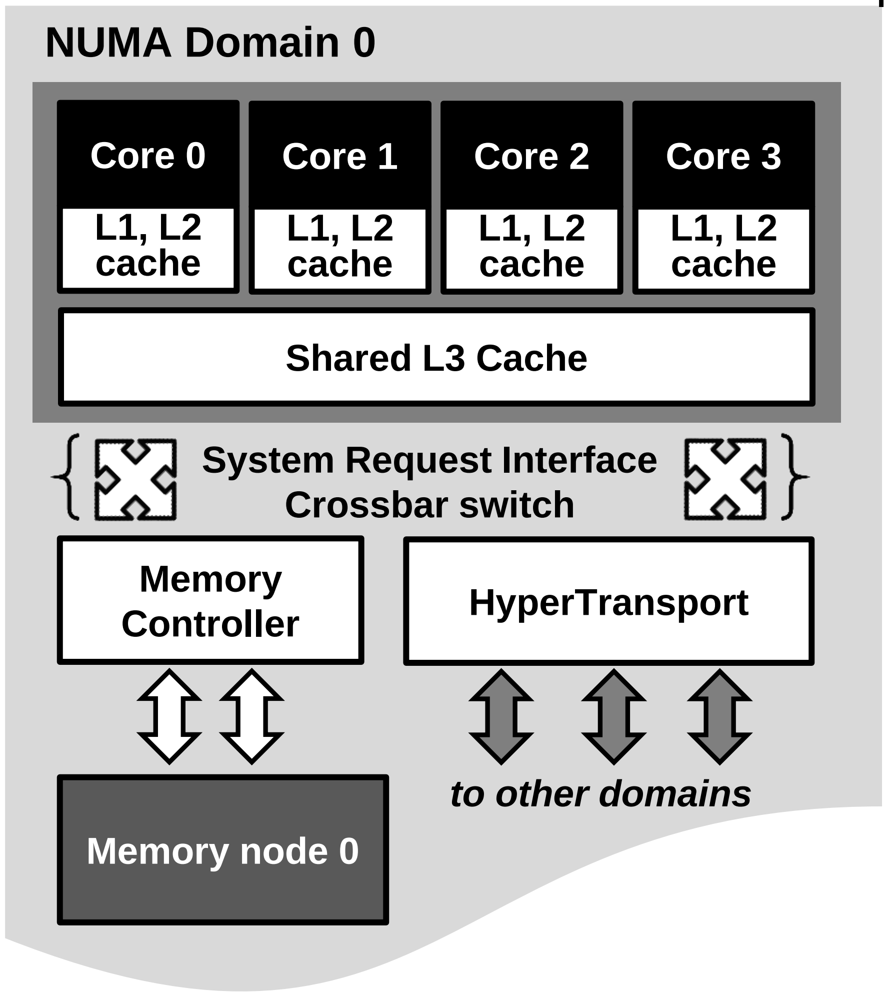


Figure 1: A schematic view of a system with four memory domains and four cores per domain. There are 16 cores in total, and a shared L3 cache per domain.

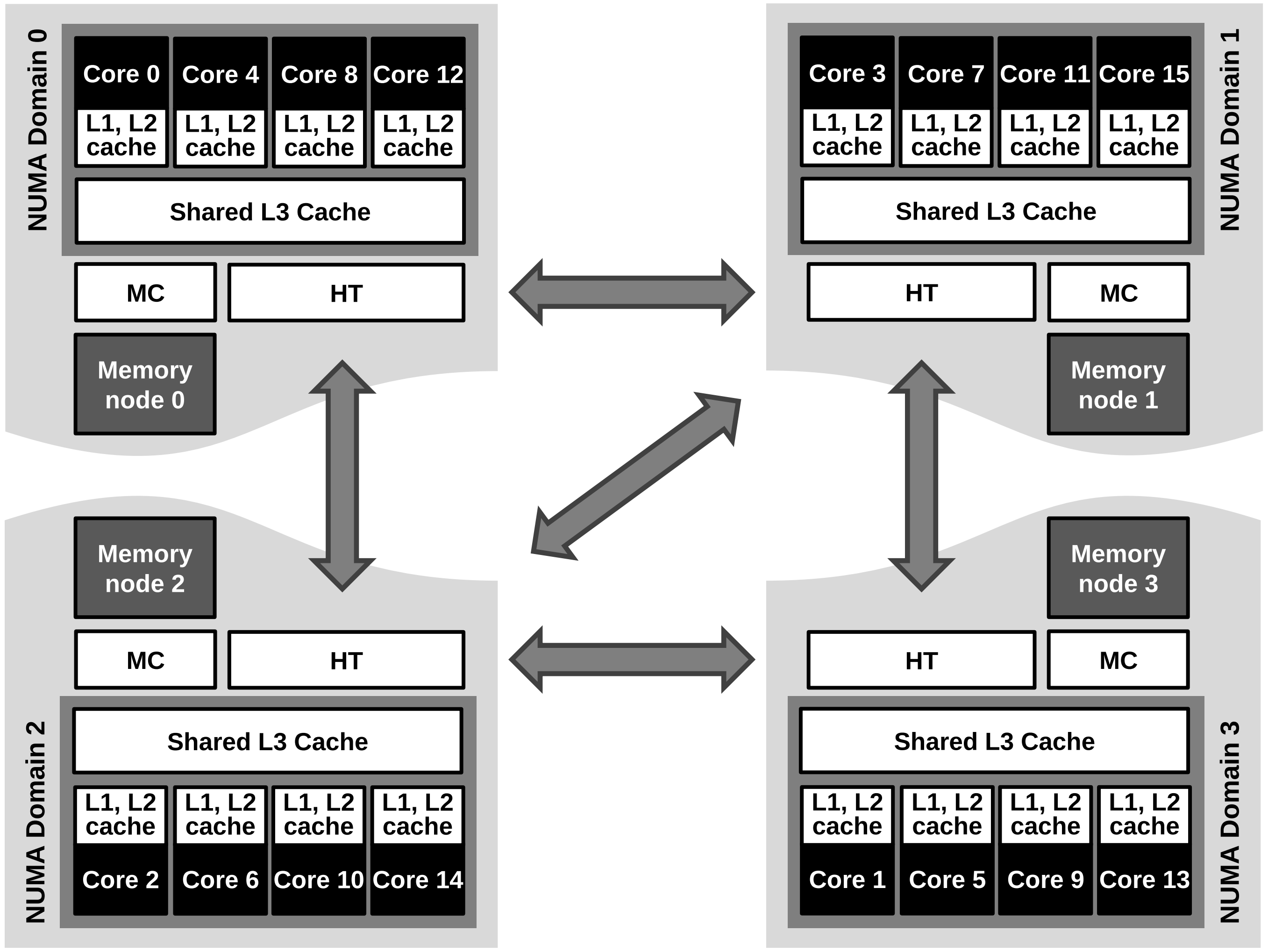
Suppose that two competing threads A and B run on cores C1 and C2 on a system shown in Figure 1. A contention-aware scheduler would detect that A and B compete and migrate one of the threads, for example thread B, to a core in a different memory domain, for example core C5. Now A and B are not competing for the last-level (L3) cache, and on UMA systems this would be sufficient to eliminate contention. But on a NUMA system shown in Figure 1, A and B are still competing for the memory controller at Memory Node #1 (MC in Figure 5), assuming that their memory is physically located in Node #1. So by simply migrating thread B to another memory domain, the scheduler does not eliminate one of the most significant sources of contention – contention for the memory controller



Four sources of performance degradation that can occur on modern NUMA systems

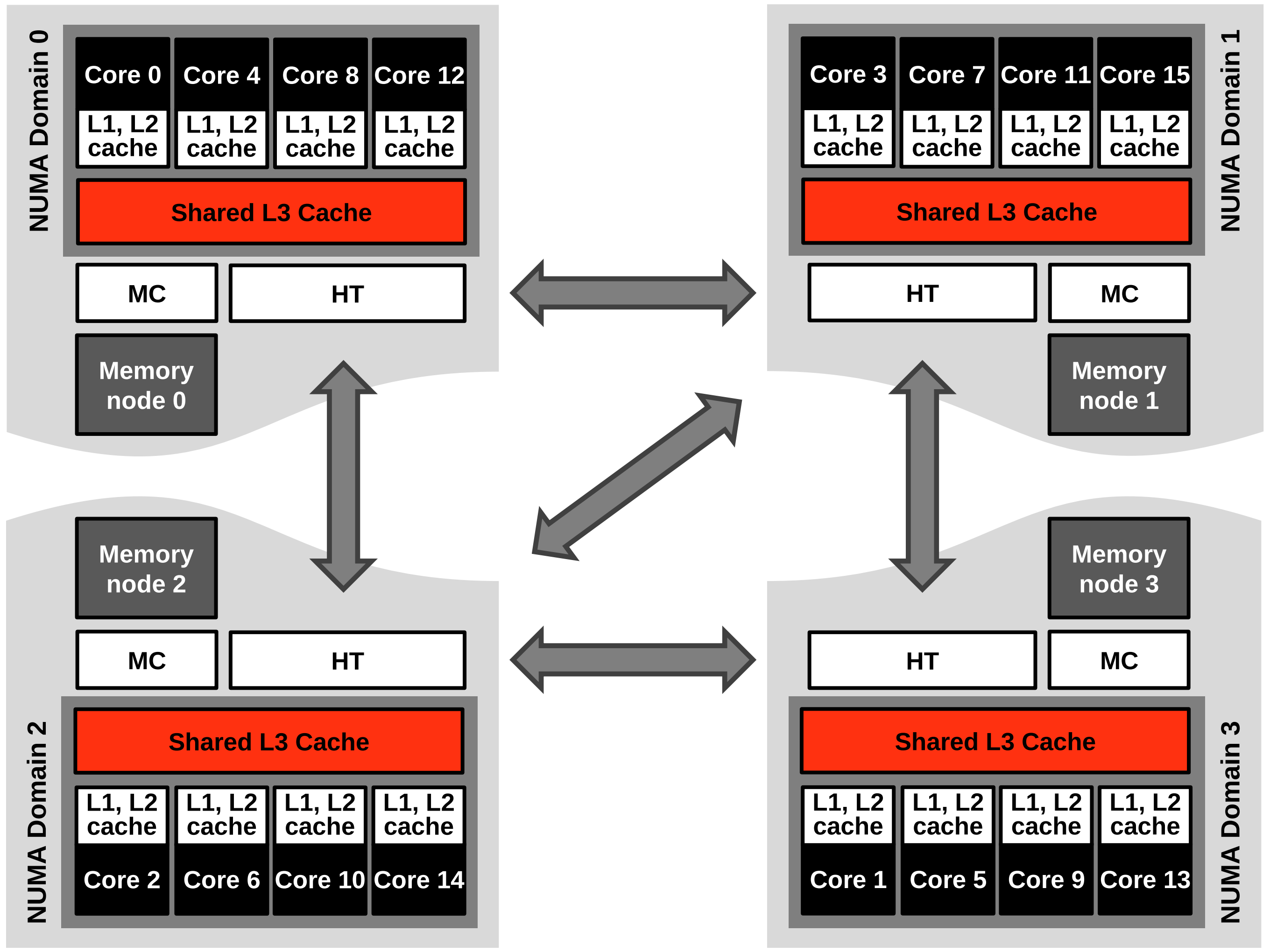
- Contention for the shared last-level cache (CA). This also includes contention for the system request queue and the crossbar.
- Contention for the memory controller (MC). This also includes contention for the DRAM prefetching unit.
- Contention for the inter-domain interconnect (IC).
- Remote access latency, occurring when a thread's memory is placed in a remote node (RL).

An AMD Opteron 8356 Barcelona domain



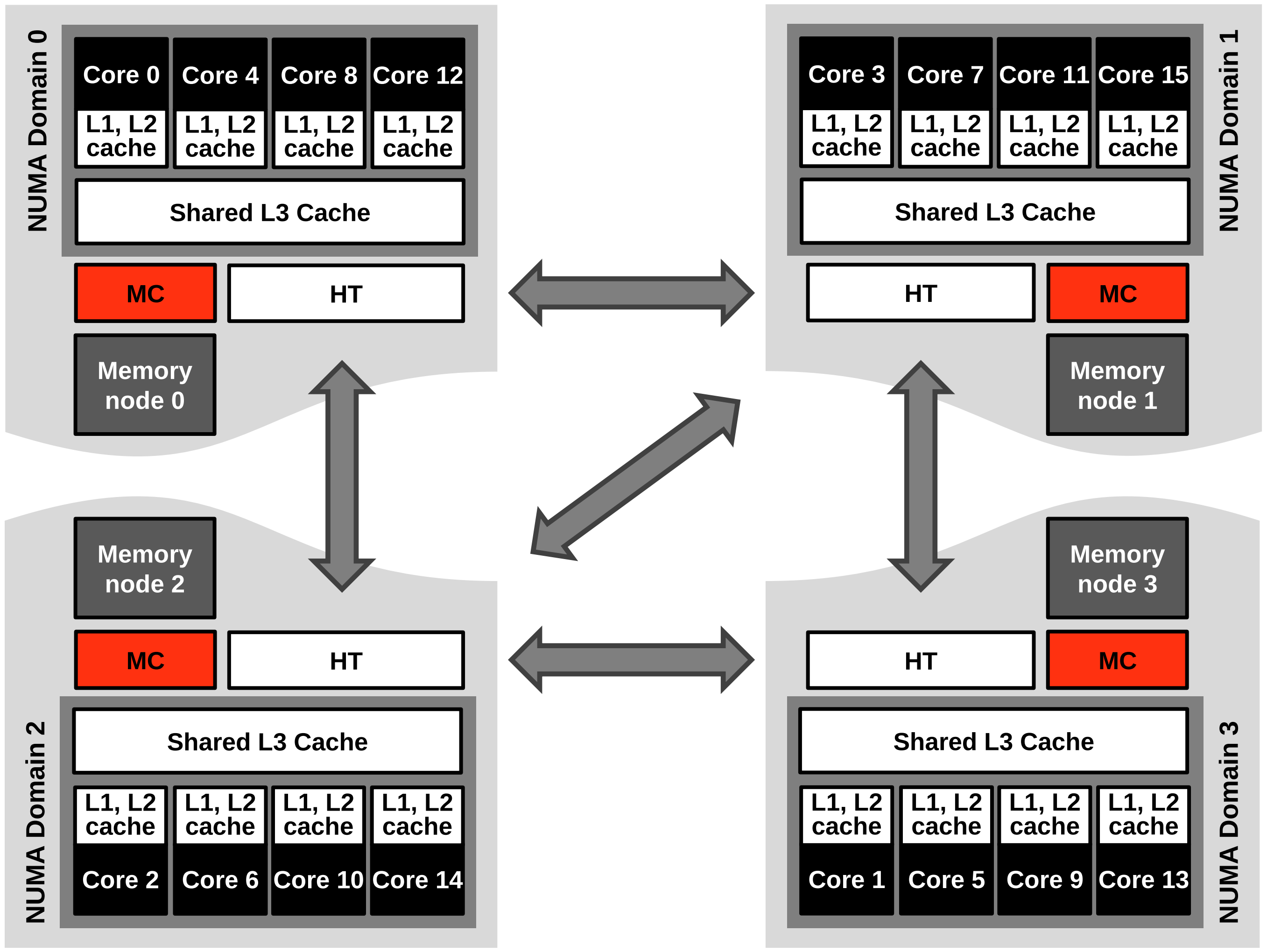
An AMD Opteron system with 4 domains

Talk by Sergey Blagodurov
Stony Brook University

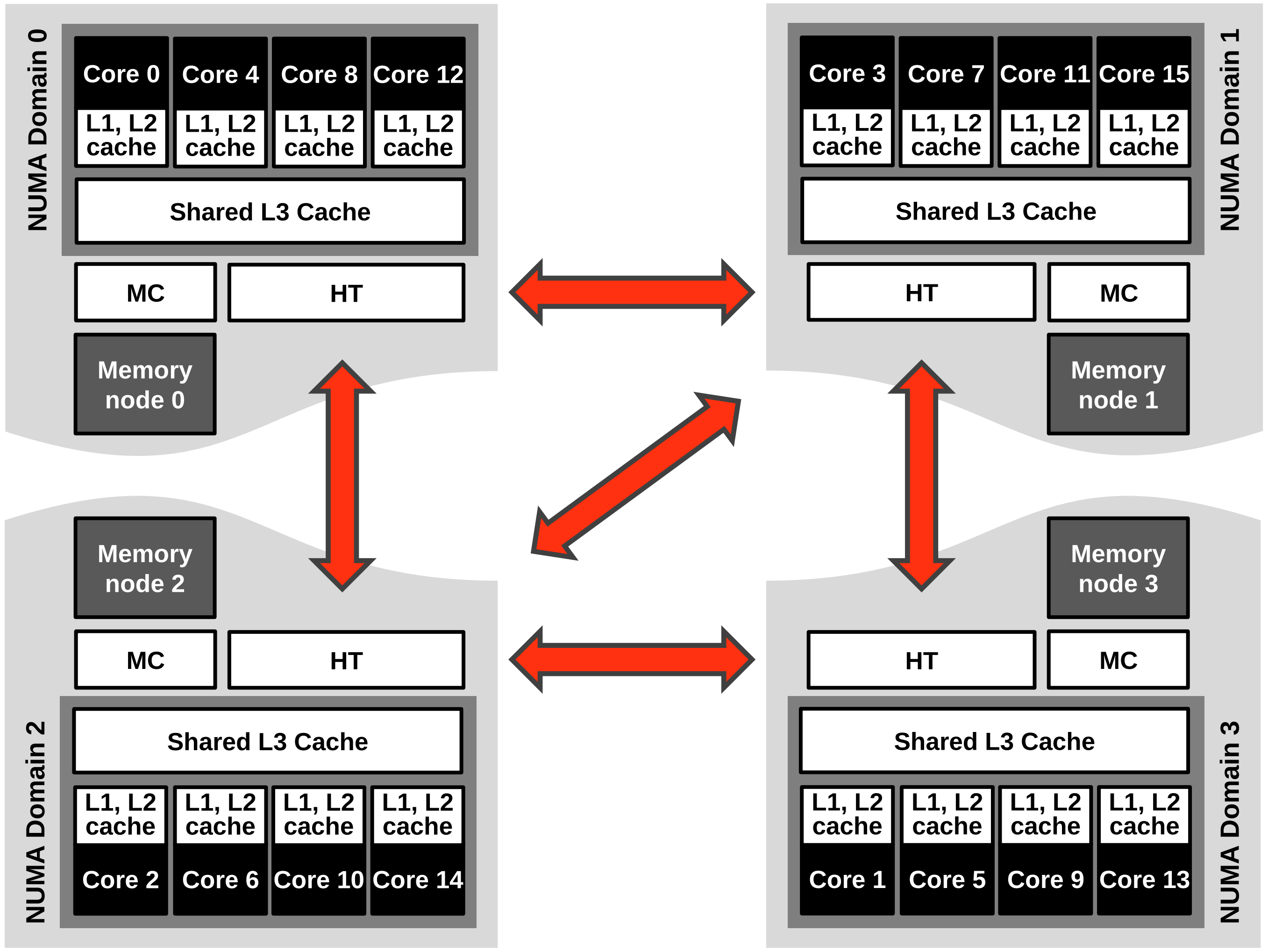


Contention for the shared last-level cache (CA)

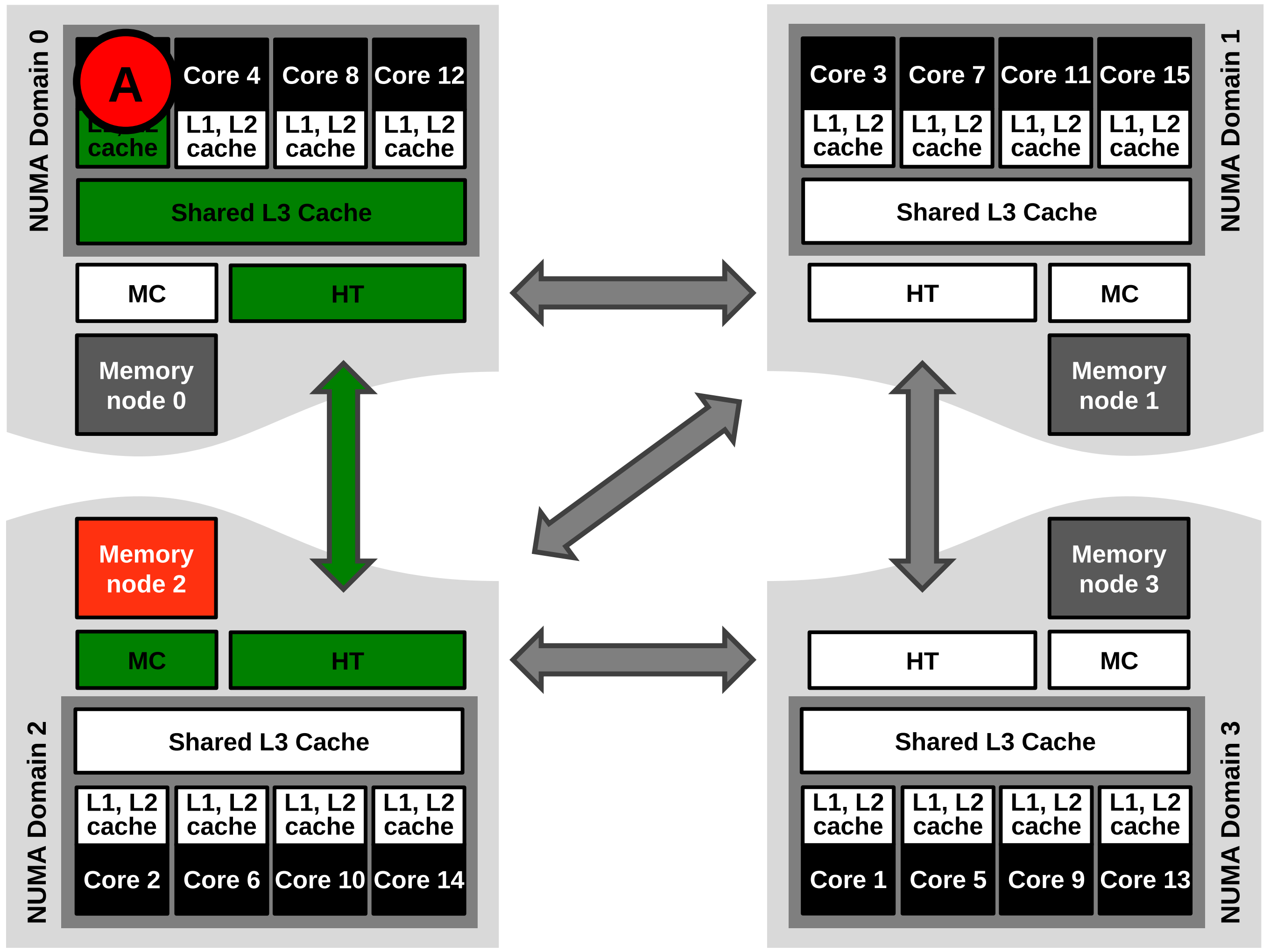
Talk by Sergey Blagodurov
Stony Brook University



Contention for the memory controller (MC)



Contention for the inter-domain interconnect (IC)



Remote access latency (RL)

Placement of Threads and Memory

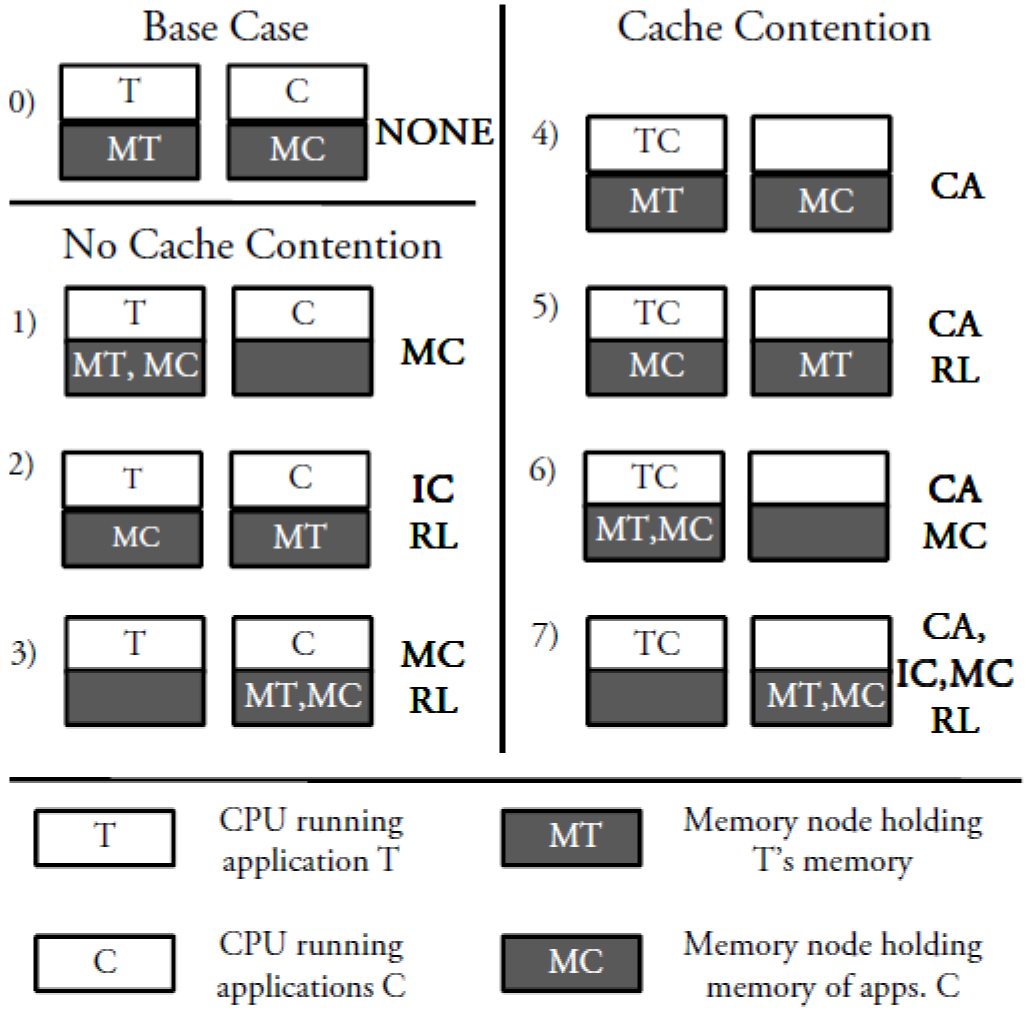
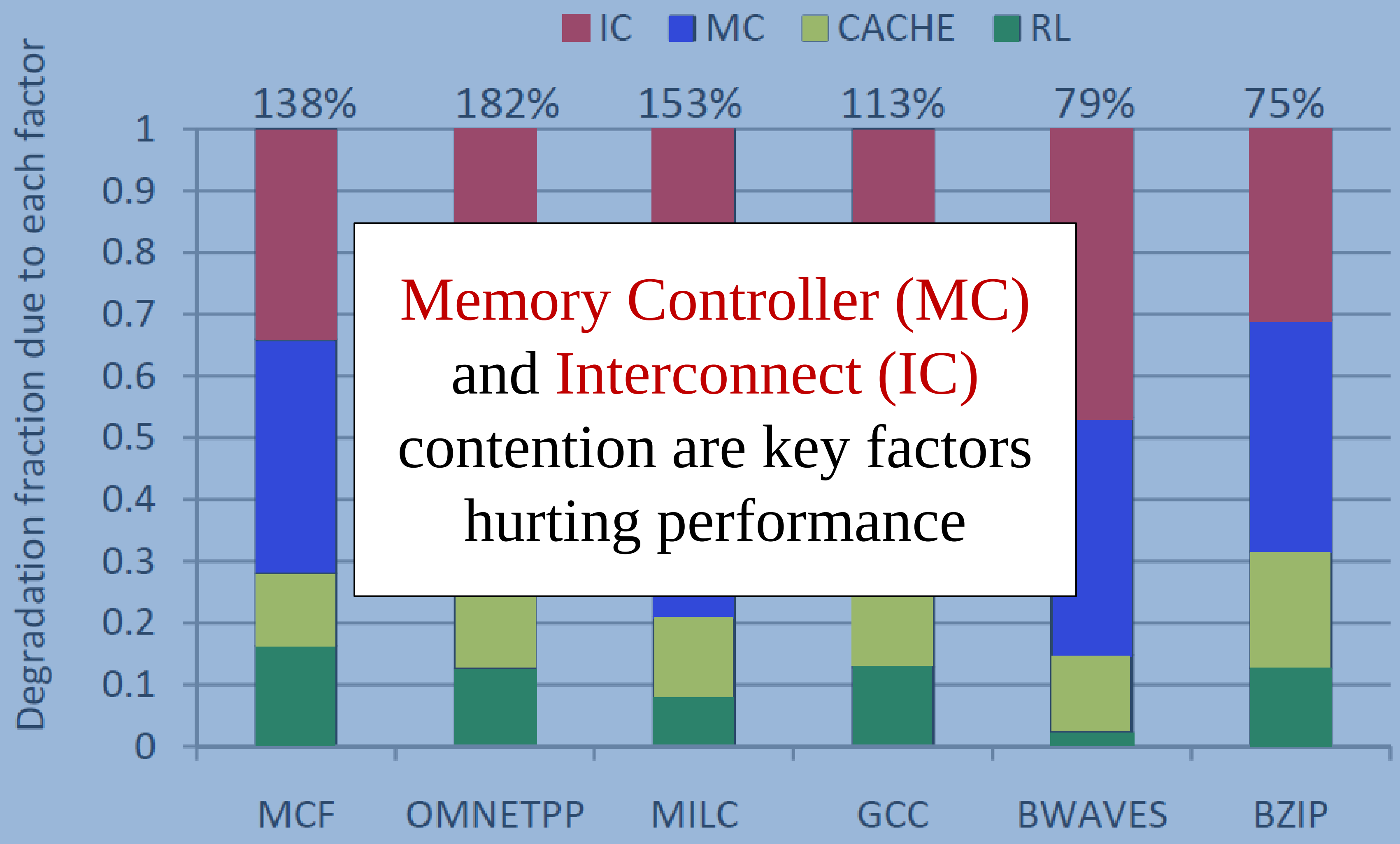


Figure 3: Placement of threads and memory in all experimental configurations.

- We run a target application, denoted as T with a set of three competing applications, denoted as C. The memory of the target application is denoted MT, and the memory of the competing applications is denoted MC.
  - CA: Cache contention
  - MC: Memory Controller
  - IC: Interconnect
  - RL: Remote Latency

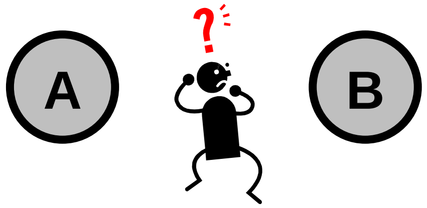


Memory Controller (MC) and Interconnect (IC) contention are key factors hurting performance

Dominant degradation factors

Characterization method

- Given two threads, decide if they will hurt each other's performance if co-scheduled



Scheduling algorithm

- Separate threads that are expected to interfere

Limited observability

- We do not know for sure if threads compete and how severely!

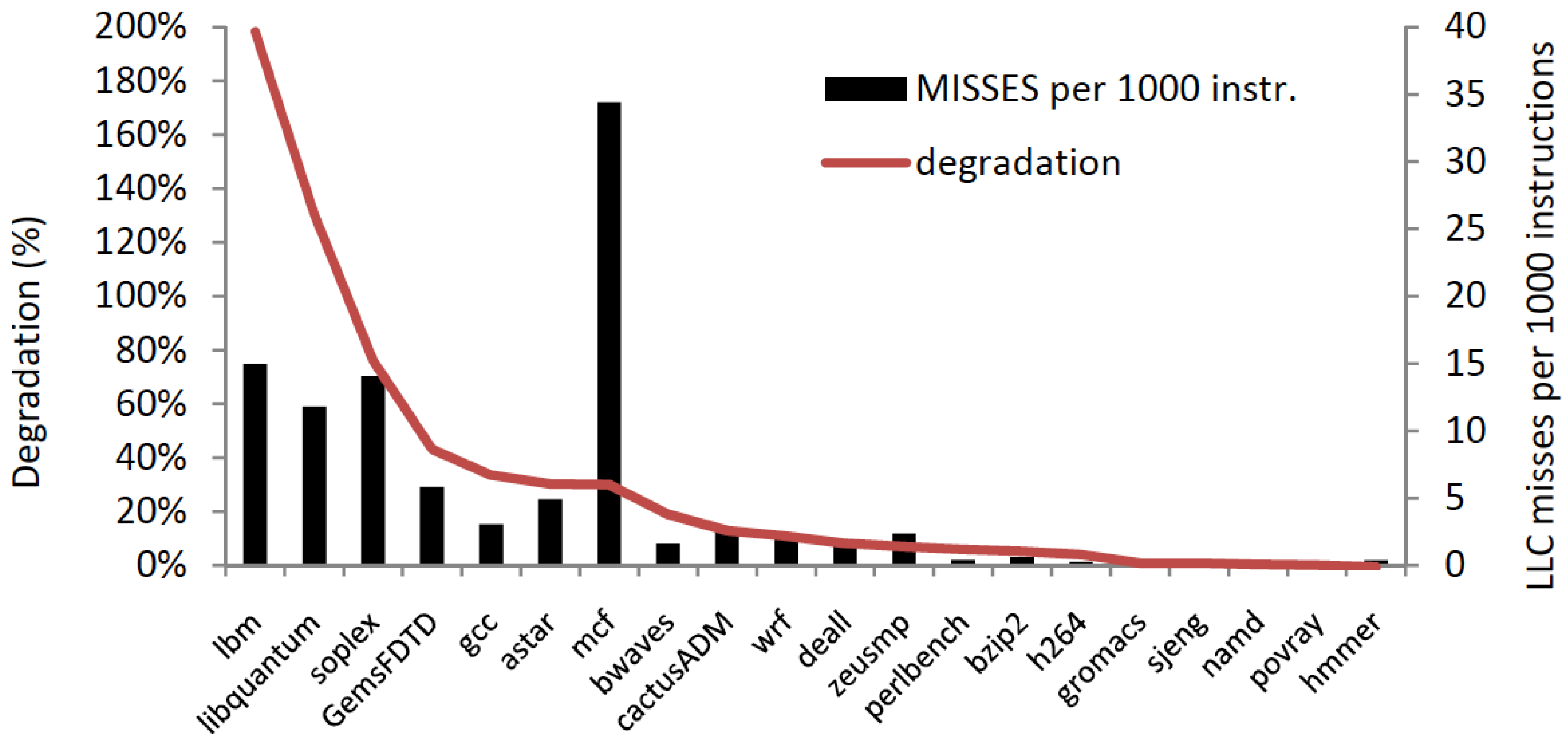
Trial and error infeasible on large systems

- Can’t try all possible combinations
- Even sampling becomes difficult

A good trade-off: measure LLC Miss rate!

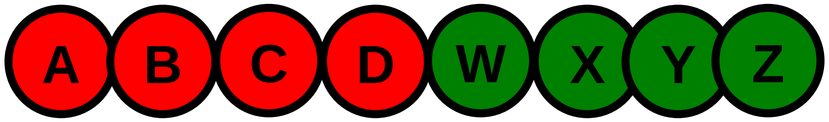
- Threads interfere if they have high miss rates
- No account for cache contention impact

Characterization Method

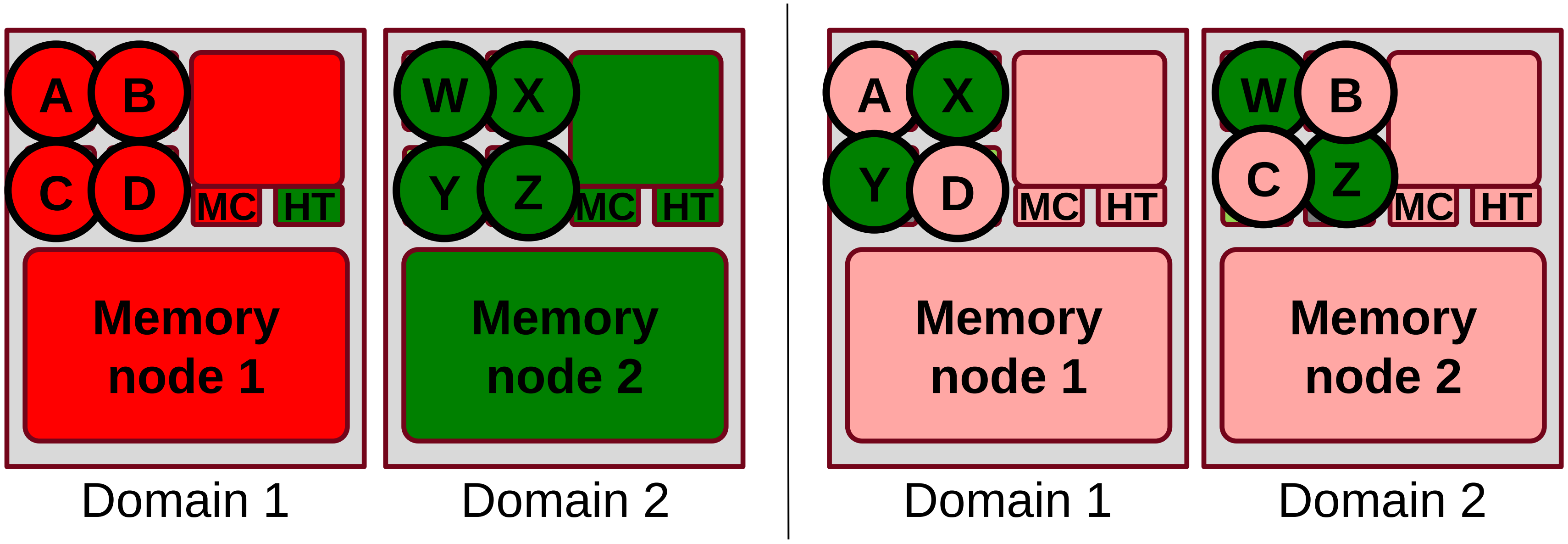


Miss rate as a predictor for contention penalty

Sort threads by LLC missrate:

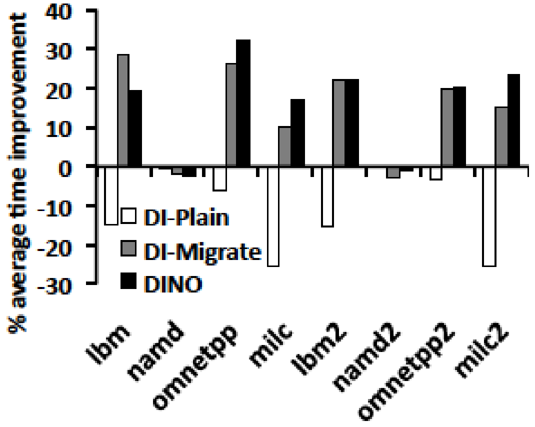


Goal: isolate threads that compete for shared resources *and pull the memory to the local node upon migration*

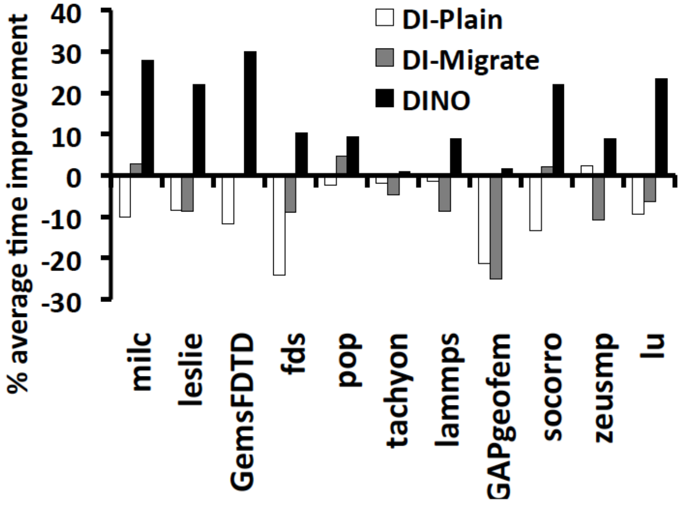


Migrate competing threads along with memory to different domains

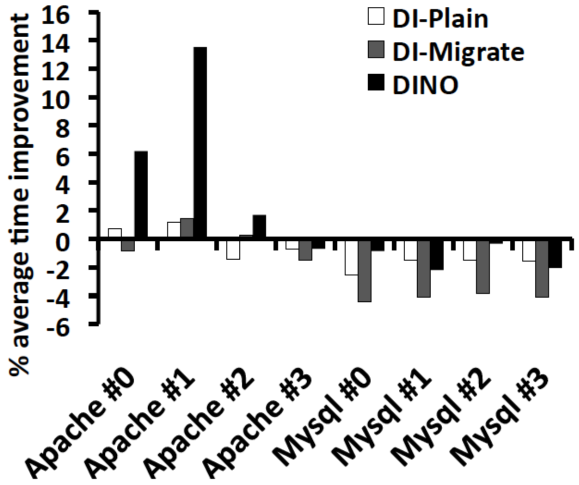
Server-level scheduling



SPEC CPU 2006



SPEC MPI 2007



LAMP

Server-level results

Talk by Sergey Blagodurov
Stony Brook University

SFU SIMON FRASER UNIVERSITY THINKING OF THE WORLD