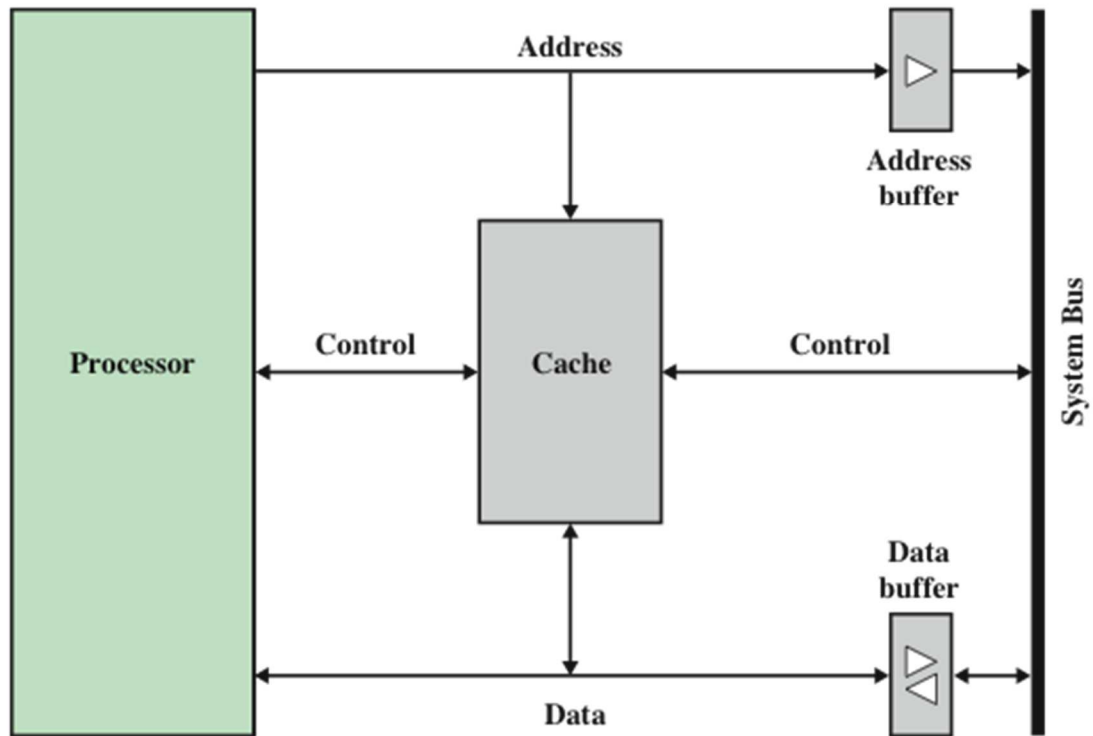
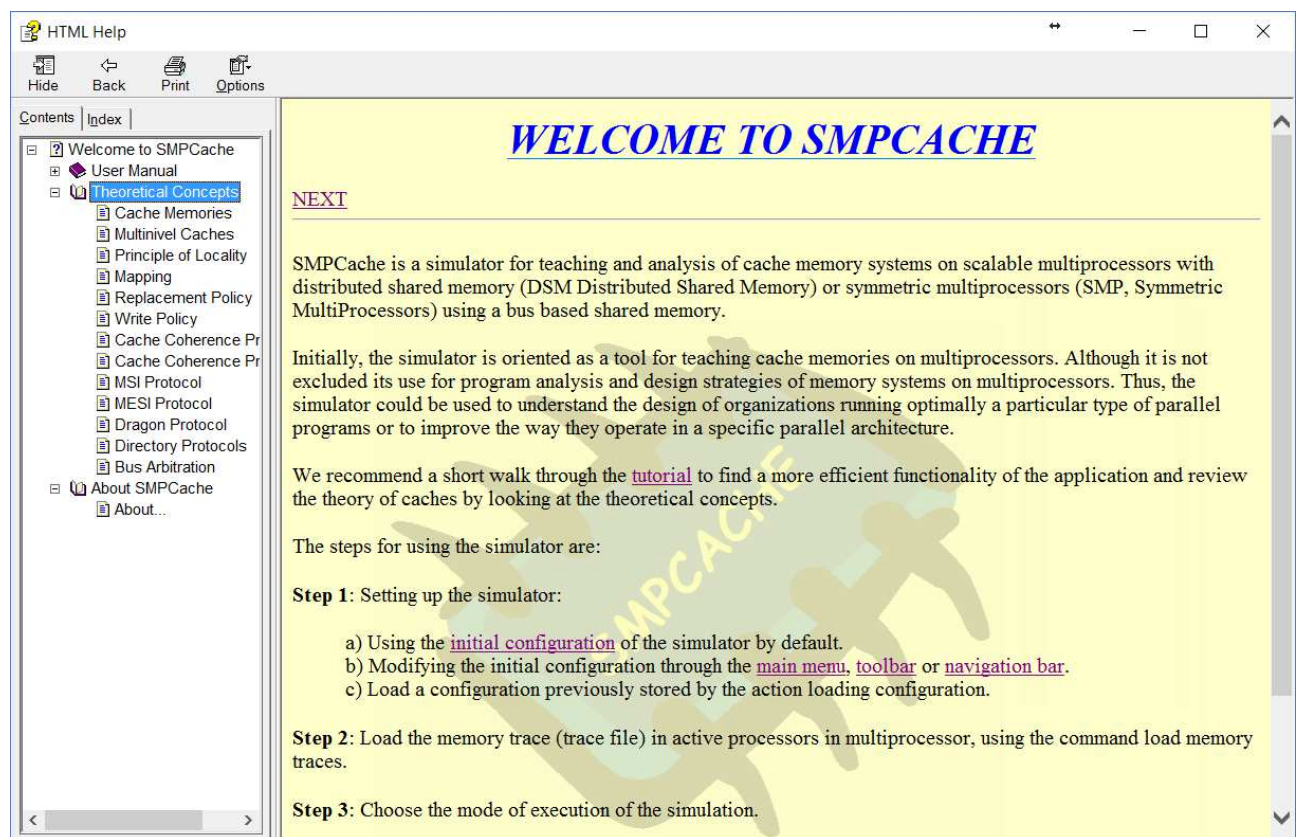


### Lab 3 – Cache Summary & Introduction to SMPCache Simulator



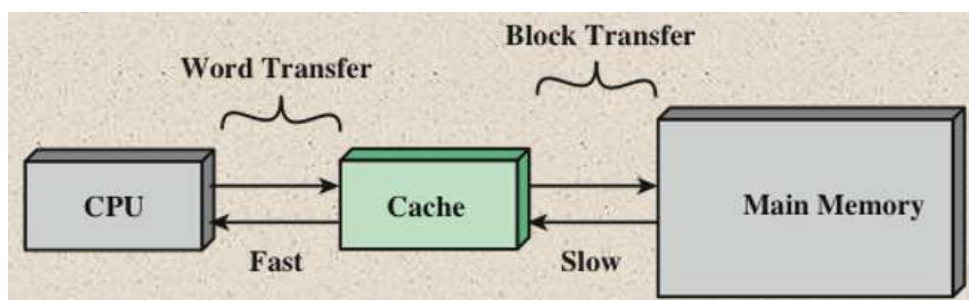
**Figure 4.6 Typical Cache Organization**

## I. Ôn tập lại các kiến thức cơ bản về Cache



# CACHE MEMORIES

- The cache is a device that sits between the central processing unit (CPU) and main memory (MP). Almost all current computers have cache, some more than once (multi-level hierarchy). It is a small and fast memory containing a copy of those MP positions that are being used by the CPU. It is based on the principle of locality: when the CPU calls reference an item, it will tend to be referenced again soon (temporary location, which leads us to have the cache as a small memory and fast), and is about some elements with high probability of being accessed (spatial locality, which leads us to consider the location of sets of words in the cache, or blocks).
  - We consider in our study an organization of the cache and main memory (MP) based on block structure. A block is the unit of information referenced in the memory hierarchy, composed of words of memory (read or written at the request of the CPU, but as part of a block). It is known as cache **line or partition** the **cache block** (k words) and the label (which identifies the address of the block structure corresponding MP) and the validity bit (bit that indicates whether the block contains data valid). We will call **set** a group of cache blocks whose labels are examined in parallel.
  - The memory address completely specifies the location of a word in main memory. It consists of the address of the block structure (which identifies a block in main memory) and the direction of displacement of block (that places the word within the block). This is the memory address or address provided by the CPU. To "convert" to address cache, be construed in accordance with the correspondence function correspondence function in question.
- 
- Cache là một thiết bị nằm giữa CPU và main memory (MP)
  - Các máy tính hiện nay thường có nhiều hơn 1 cache
  - Cache là bộ nhớ nhỏ, nhanh chứa 1 bản copy của MP, phần đang được CPU sử dụng.
  - Nó dựa trên nguyên lý cục bộ (principle of locality, khi CPU tham chiếu đến 1 word trong bộ nhớ, nó thường tham chiếu lại sau đó)
  - Trao đổi thông tin giữa MP và Cache theo đơn vị block (gồm nhiều word), giữa CPU và Cache theo đơn vị là word (tối thiểu 1 byte)
  - Các block trong MP được đưa vào các line trong Cache (do đó, kích thước line trong cache bằng kích thước block)
  - Các line trong Cache được nhóm thành các Set (ví dụ: k set), do đó phương pháp ánh xạ tương ứng sẽ là k-way associative.
  - Địa chỉ bộ nhớ được phát ra bởi CPU gồm 2 phần chính (phần đầu s bit để xác định block, phần thứ 2 là w bit sẽ xác định chính xác word trong block đã xác định). Phần đầu s bit sẽ được chia lại để xác định set/line trong cache, sau đó phần w cũng được dùng để xác định chính xác word nằm trong line đã xác định.



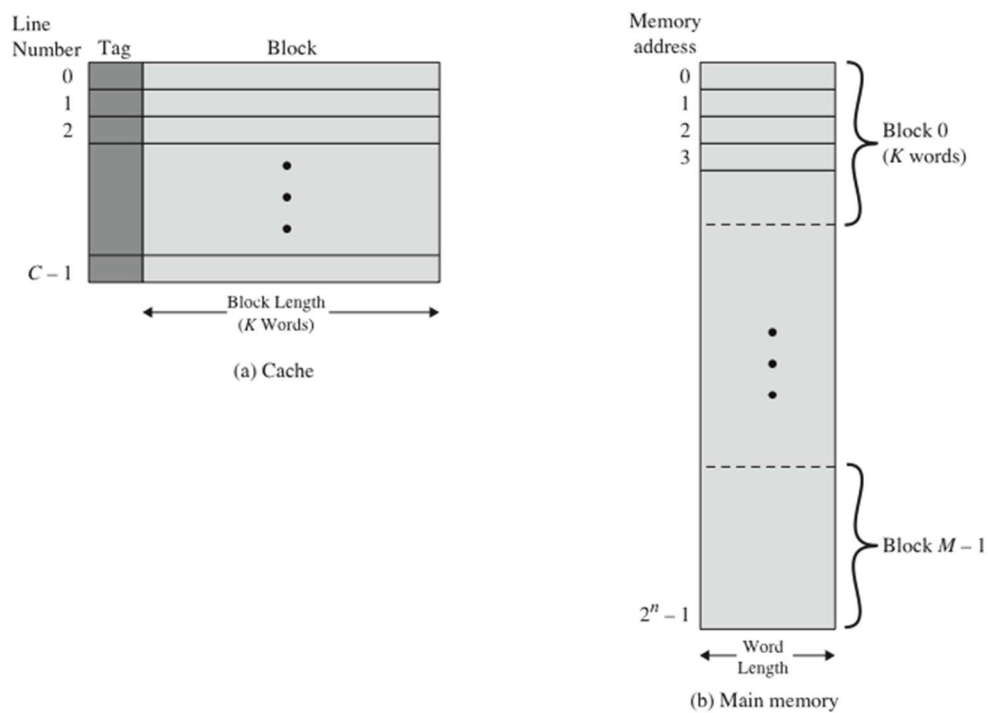


Figure 4.4 Cache/Main-Memory Structure

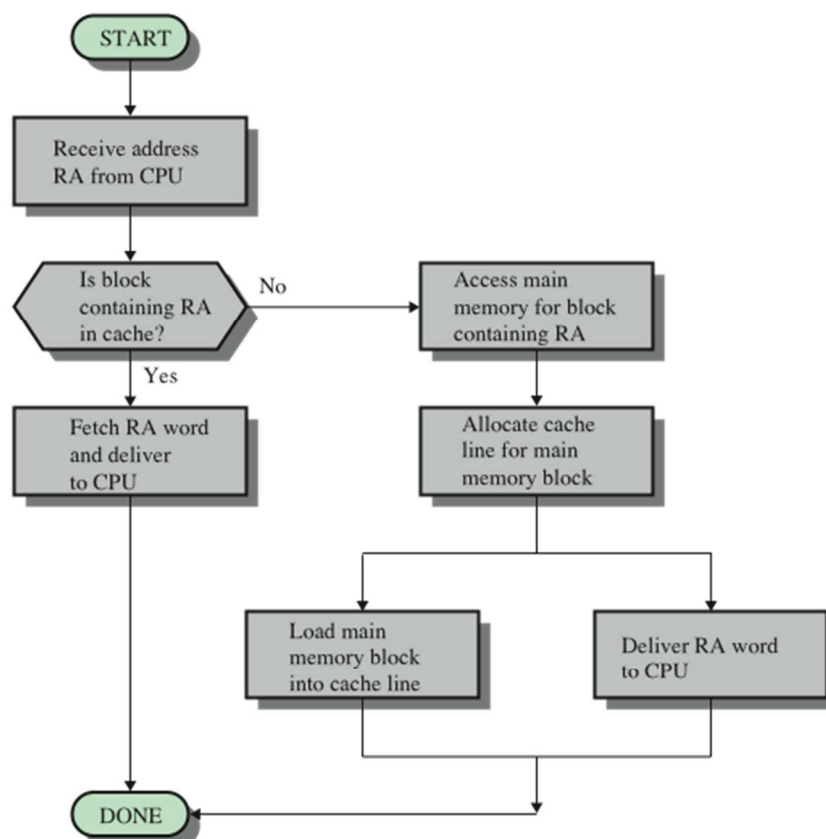


Figure 4.5 Cache Read Operation

# MULTILEVEL CACHES

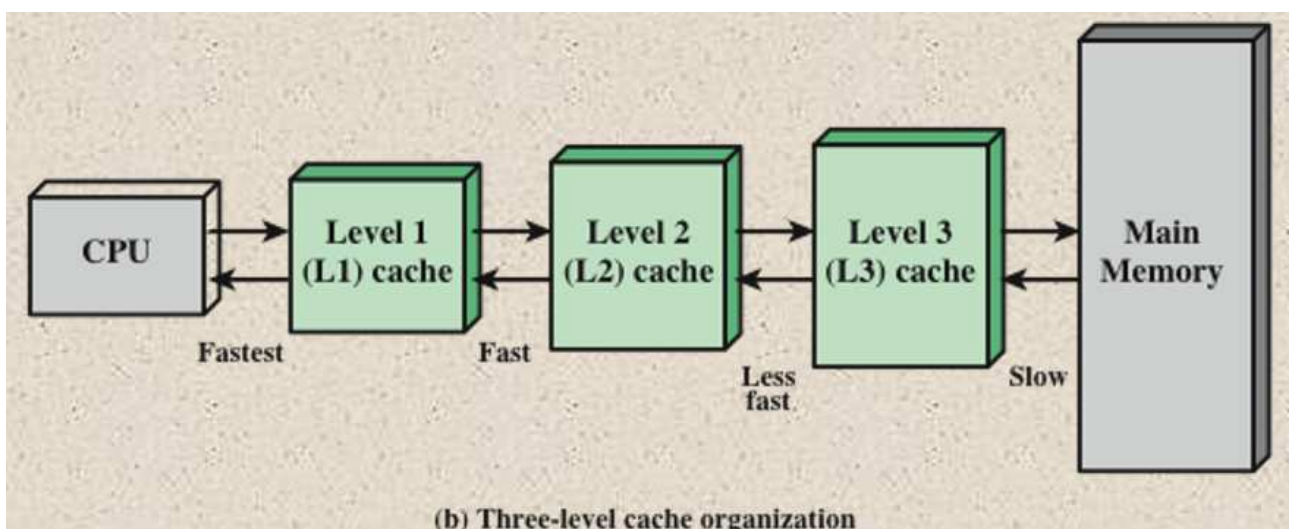
Current technology offers increased CPU speed, but there is not a parallel development with the memories, increasingly large, thus increasing the amount of time spent executing the accesses of memory. This problem is alleviated by a multilevel cache system, in which each cache is designed to meet different criteria. These hierarchies of cache improve the execution times, achieving an improved overall performance of the CPU, reducing the number of cycles and the cycle time.

Assume a hierarchy with two levels of cache, where:

- **L1.** Higher level or closer to the CPU. It is small, to match the clock cycle of the CPU. That is, the L1 size affects the frequency of the CPU clock.
- **L2.** Level immediately below (or secondary cache). It's great to capture many accesses that would otherwise go to main memory, and get a high hit rate. It has more design alternatives than the L1 cache.

We consider the multilevel inclusion property: all data that are in L1 are always in L2. It is desirable to maintain consistency among caches. The model shown is for different block sizes for L1 and L2, although the case is not supposed to SMP Cache.

- Công nghệ hiện nay đã cung cấp các CPU có tốc độ rất cao, nhưng không đồng thời tăng về tốc độ với bộ nhớ (thường tăng dung lượng lớn hơn), do đó làm tăng gian truy xuất bộ nhớ.
- ➔ Cần thiết kế một hệ thống Cache nhiều mức.
- Ví dụ: Một hệ thống cache gồm 2 mức:
  - L1: Mức cao hơn, gần CPU. Nhỏ, nhưng có tốc độ gần với tốc độ của CPU.
  - L2: Mức thứ cấp, xa CPU. Đảm bảo có thể đáp ứng hầu hết các truy xuất từ CPU mà không cần phải truy xuất ở main memory (tỉ lệ hit cao).
- Tất cả data có trong L1 luôn có ở L2



# **PRINCIPLE OF LOCALITY**

- The memory accesses (references) generated by a process (program in execution) are nonrandom but behave in a somewhat predictable manner. Such characteristics of programs are due to looping, sequential and block-formatted control structures inherent in the grouping of instructions, and data in programs. These properties are referred to as the **principle of locality**.
- There are two main components of the principle of locality, which coexist in an active process. These are temporal, and spatial localities. In temporal locality, there is a tendency for a process to reference in the near future the elements (memory blocks) referenced in the recent past. In spatial locality, there is a tendency for a process to make references to a portion of the address space in the neighbourhood of the last reference (memory access).
- Nguyên lý cục bộ (Principle of locality):
  - Các truy xuất bộ nhớ bởi chương trình không phải là ngẫu nhiên và có thể tiên đoán. Ví dụ: các chương trình thường gồm các vòng lặp, chuỗi/khối lệnh tuần tự. Các dữ liệu được nhóm theo mảng.
  - Locality liên quan đến cả
    - Thời gian (temporal): CPU có xu hướng truy xuất lại block vừa mới sử dụng
    - Không gian (spatial): CPU có xu hướng truy xuất đến các word gần với word vừa được truy xuất (thường trong cùng 1 block).

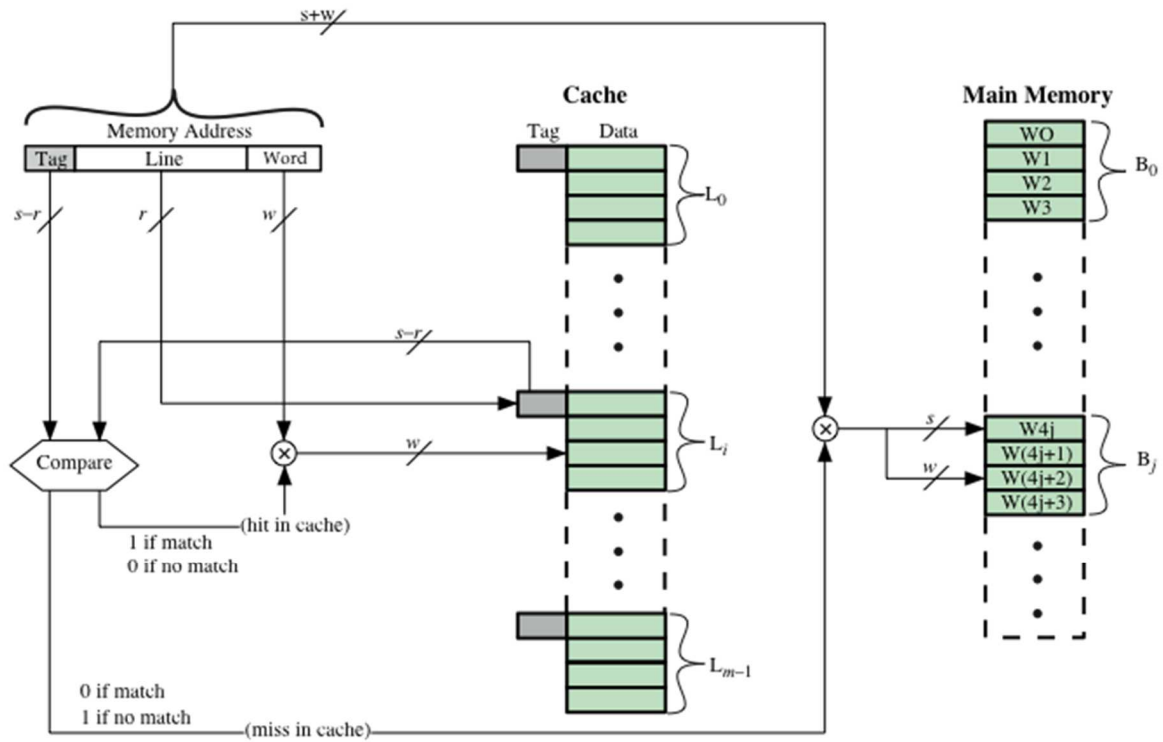
# MAPPING

- In order to locate a memory block in the cache, it is necessary to have some function which maps the main memory address into a cache location. For uniformity of reference, both main memory and cache are divided in equal-sized blocks. The mapping or placement policy determines the mapping function from the main memory address to the cache location. There are basically three placement policies: direct, fully associative, and set associative.
- The **direct mapping** is the simplest of all organizations. In this mapping, block  $i$  of the main memory maps into the block  $(i \text{ modulo } \text{blocks\_in\_cache})$  of the cache. This mapping does not need a special replacement policy. Of all the memory blocks that map into a cache block, only one can actually be in the cache at a time. Therefore, if a block caused a miss, we would simply determine the cache block this memory block maps onto and replace the block in that cache block. This occurs even when the cache is not full.
- In terms of performance, the **fully-associative mapping** is the best and most expensive cache organization. The mapping is such that any block in memory can be in any cache block. This mapping permits the development of a wide variety of [replacement policies](#).

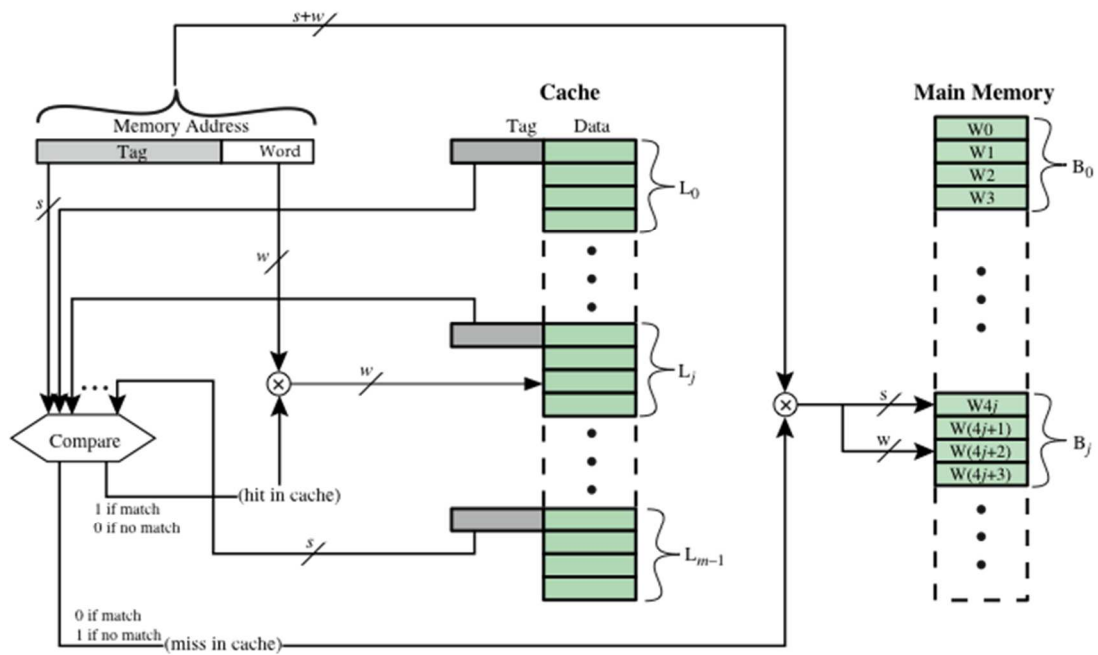
The **set-associative mapping** represents a compromise between direct and fully-associative mapping organizations. In this mapping, the cache is divided into  $S$  sets (with the same number of cache blocks, ways, per set). A block  $i$  in main memory can be in any cache block belonging to the set  $(i \text{ modulo } S)$  of the cache. This mapping also permits a wide variety of replacement policies. The set-associative mapping is the most commonly used replacement policy for cache memories.

- Có 3 chế độ ánh xạ (mapping)
  - o Direct (Trực tiếp):
    - Một block trong main memory chỉ được đưa vào 1 line nhất định trong cache. Do đó, không cần thuật toán thay thế (replacement algorithm). Nghĩa là, dù line khác trong cache có trống trong khi line mà block được quy định sẽ đưa vào đang chứa một block nào đó, thì block trong line đó cũng vẫn bị thay thế → Cứng nhắc trong việc thay thế
    - Nhưng, khi CPU tìm block trong cache thì nhanh, do chỉ cần so sánh với đúng line chứa block đó.
  - o Fully associative (Liên kết hoàn toàn):
    - Block trong main memory có thể được đưa vào bất kỳ line nào trong cache. Khi thay thế thì cần thực hiện theo thuật toán.
    - Nhưng khi CPU cần tìm 1 block trong cache thì phải so với tất cả các line.
  - o Set-associative (Liên kết tập hợp) – Thực tế trong các máy tính hiện nay
    - Dung hòa giữa 2 chế độ trên.
    - Cache được chia thành các set, mỗi set chứa  $k$  lines ( $k$ -way)
    - Một block trong main memory chỉ được đưa vào 1 set, nhưng có thể đưa vào bất kỳ line nào trong set đó → Cần thuật toán thay thế.



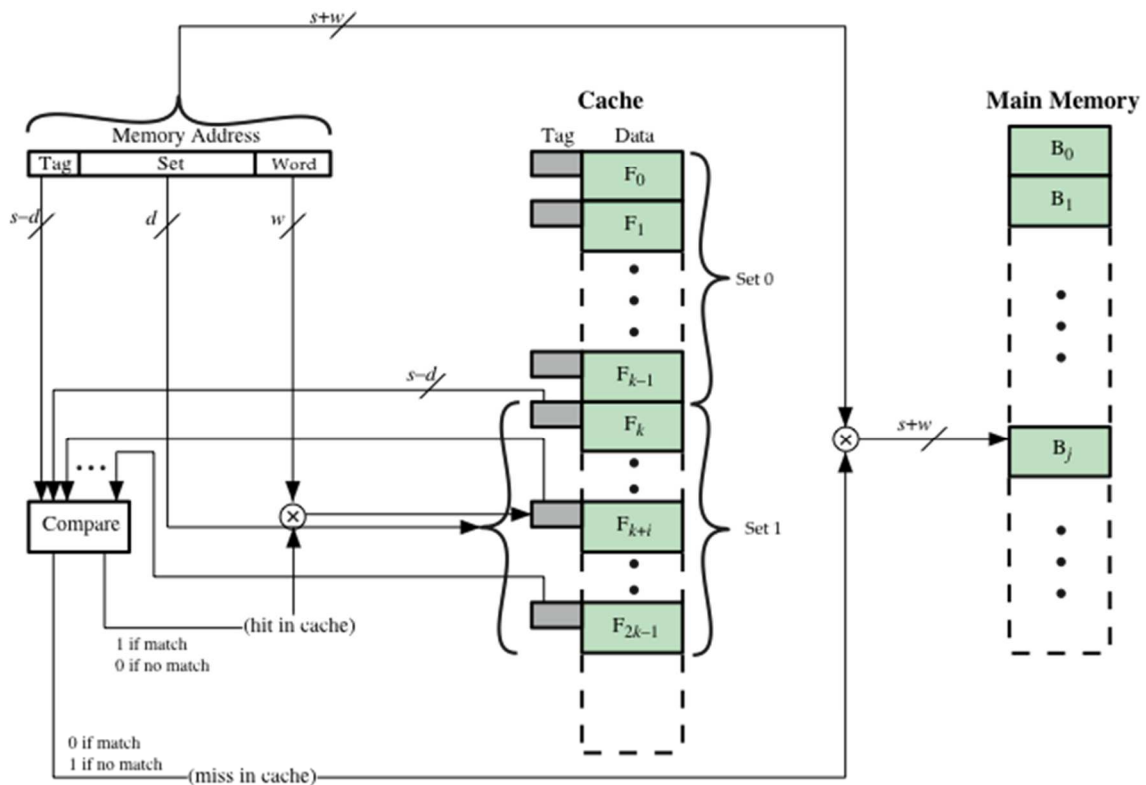


**Figure 4.9 Direct-Mapping Cache Organization**



**Figure 4.11 Fully Associative Cache Organization**





**Figure 4.14**  $k$ -Way Set Associative Cache Organization

## Set Associative Mapping Summary

- Address length =  $(s + w)$  bits
- Number of addressable units =  $2^{s+w}$  words or bytes
- Block size = line size =  $2^w$  words or bytes
- Number of blocks in main memory =  $2^{s+w}/2^w = 2^s$
- Number of lines in set =  $k$
- Number of sets =  $v = 2^d$
- Number of lines in cache =  $m = kv = k * 2^d$
- Size of cache =  $(k * 2^d) * 2^w = k * 2^{d+w}$  words or bytes
- Size of tag =  $(s - d)$  bits



# **REPLACEMENT POLICY**

- When the cache memory is full (or the corresponding set is full, in [set-associative caches](#)) and the fetched memory block is not in cache (a miss occurs), a replacement policy chooses which memory block to remove in order to create space for the fetched block. The commonly used replacement policies are: LRU (Least Recently Used), LFU (Least Frequently Used), FIFO (First-In, First-Out), and Random.
- In **LRU** policy, at a cache miss, the least recently referenced block of the resident set (all the cache in [fully-associative caches](#)) is replaced. The **LFU** policy replaces the block in the set/cache that has been referenced the least number of times. The **FIFO** policy replaces the block in the set/cache that has been in cache for the longest time. The **Random** policy chooses a block in the set/cache at random for replacement. The **LRU** policy is one of the most popular algorithms.

## Các chính sách/thuật toán thay thế (replacement algorithm/policy)

- LRU (Least Recently Used)
  - o Khi miss, thì block trong cache gần đây ít được truy xuất nhất sẽ bị thay thế
- LFU (Least Frequently Used)
  - o Khi miss, thì block trong cache ít được truy xuất nhất sẽ bị thay thế
- FIFO (First-In, First-Out)
  - o Khi miss, thì block trong cache được đưa vào lâu nhất sẽ bị thay thế
- Random
  - o Chọn ngẫu nhiên một block trong cache để thay thế.

# WRITING STRATEGY

- The time when a word (actually, a block) in main memory is updated after a write depends on the writing strategy. There are two basic writing strategies: write-through, and write-back.
- In **write-through** strategy, the main memory copy of the data word (block) is updated directly after a write. Write-through strategy usually results in more memory traffic, which can be very detrimental to the performance of a multiprocessor system.
- The **write-back** strategy always allocates a cache block on a miss. When a write-hit occurs, only the block in the cache is modified. The main memory update takes place when the block is replaced and swapped back to memory. To improve performance, the replaced block is written back only if it has been modified

Chiến lược ghi (write policy): Khi nội dung block trong cache bị thay đổi (write-hit)

- Write-through: Ngay sau khi block trong cache được cập nhật (write-hit), thì cập nhật luôn lại block tương ứng trong main memory
  - o → Nhất quán về thông tin giữa Cache và main memory, nhưng chiếm nhiều băng thông của bus hệ thống và thời gian xử lý.
- Write-back: Chỉ cập nhật block trong cache (write-hit). Block tương ứng trong main memory chỉ được cập nhật khi block đó trong cache bị thay thế. Hoặc, chỉ cập nhật lại block trong main memory khi block đó bị thay đổi và thay thế trong cache.
  - o → Chiếm ít băng thông của bus hệ thống và thời gian xử lý. Nhưng khó khăn khi làm việc với hệ thống nhiều CPU/IO sử dụng chung main memory.

# *An Introduction to Multiprocessor Systems*

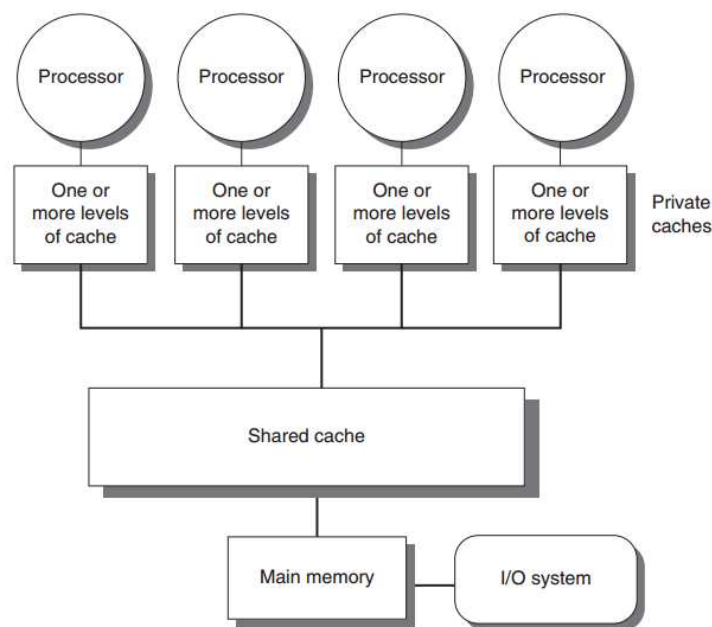
<http://www.realworldtech.com/coherency/>

## Memory Hierarchies

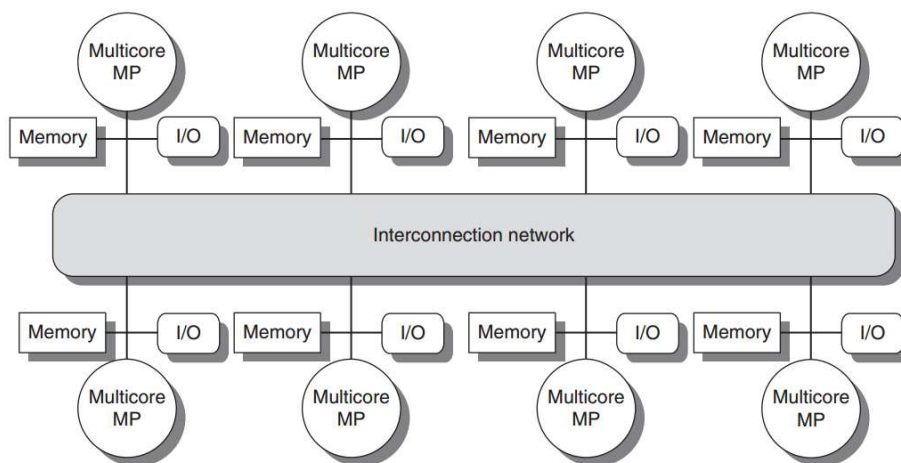
There are two different types of shared memory multi-processor systems: Symmetric Multi-Processors (SMPs) and Distributed Shared Memory (DSM). In both cases every processor can read and write to any portion of the system's memory. However, in an SMP system there is usually a single pool of memory, which is equally fast for all processors, (this is sometimes referred to as a Uniform Memory Access). In contrast, in a DSM system there are multiple pools of memory and the latency to access memory depends on the relative position of the processor and memory (this is also referred to as cache coherent Non-Uniform Memory Access). Typically, each processor has local memory (the lowest latency) while everything else is classified as remote memory and is slower to access.

SMP (Symmetric Multi-Processors) và DSM (Distributed Shared Memory) là hai kiểu tổ chức hệ thống nhớ chia sẻ trong các hệ thống đa vi xử lý (multi-processor system).

- Trong cả 2 kiểu tổ chức, mỗi bộ vi xử lý đều có thể đọc và ghi vào bất kỳ phần nào của bộ nhớ hệ thống.
- Tuy nhiên, hệ thống SMP thường có một vùng nhớ có tốc độ nhanh như nhau cho tất cả các bộ vi xử lý (còn được gọi là Truy xuất bộ nhớ đồng nhất (UMA: Uniform Memory Access)).
- Trái lại, hệ thống DSM có nhiều vùng nhớ và độ trễ trong các thao tác truy xuất bộ nhớ phụ thuộc vào vị trí tương đối của bộ vi xử lý và bộ nhớ (còn được gọi là Truy xuất bộ nhớ không đồng nhất liên kết cache (CC-NUMA: Cache Coherent Non-Uniform Memory Access)). Về cơ bản, mỗi bộ vi xử lý có một bộ nhớ cục bộ (local memory, có độ trễ thấp nhất), và các bộ nhớ còn lại được coi là bộ nhớ xa (remote memory, có tốc độ truy xuất chậm hơn).



**Figure 5.1** Basic structure of a centralized shared-memory multiprocessor based on a multicore chip. Multiple processor-cache subsystems share the same physical memory, typically with one level of shared cache, and one or more levels of private per-core cache. The key architectural property is the uniform access time to all of the memory from all of the processors. In a multichip version the shared cache would be omitted and the bus or interconnection network connecting the processors to memory would run between chips as opposed to within a single chip.



**Figure 5.2** The basic architecture of a distributed-memory multiprocessor in 2011 typically consists of a multi-core multiprocessor chip with memory and possibly I/O attached and an interface to an interconnection network that connects all the nodes. Each processor core shares the entire memory, although the access time to the local memory attached to the core's chip will be much faster than the access time to remote memories.

SMP systems nicely complement shared bus based architectures, since a shared bus provides a single point of arbitration, so cache coherency is relatively straight forward. When a processor needs to broadcast a cache coherency message, it simply sends that information over the bus, and all other processors can receive it. However, since memory is accessed at the speed of the slowest/most distant processor, physical limitations limit scalability to relatively few processors. This is the model that Intel has pursued since the advent of the Pentium Pro, and continues today with the entire Xeon line. DSM systems scale more effectively because local memory can be accessed rapidly. Point to point interconnects are a natural fit for DSM, because the bandwidth grows proportionally to the number of processors and amount of memory in the system. The most common example of a DSM system is any 2 or 4 processor Opteron. However, most high-end systems such as IBM's pSeries, the HP Superdome, SGI Altix, or larger Sun and Fujitsu SPARC systems use distributed memory.

- Các hệ thống SMP có ưu điểm trên các kiến trúc sử dụng chung bus (shared bus), vì các kiến trúc này chỉ cần sử dụng một trọng tài bus, do đó vấn đề liên kết cache đơn giản hơn. Khi một bộ vi xử lý cần quảng bá thông tin về liên kết cache, nó chỉ đơn giản gửi thông tin đó lên bus, và tất cả bộ vi xử lý đều có thể nhận. Tuy nhiên, do bộ nhớ được truy xuất ở các tốc độ bằng với tốc độ của bộ vi xử lý chậm nhất, do đó làm hạn chế khả năng mở rộng thêm bộ vi xử lý. Đây chính là mô hình bộ nhớ mà Intel theo đuổi từ khi phát minh ra Pentium Pro và tiếp tục sử dụng trong dòng máy Xeon.
- Trái lại, các hệ thống DSM có khả năng mở rộng linh hoạt, do các vùng nhớ cục bộ có thể được truy xuất rất nhanh. Với hệ thống này, kết nối điểm-điểm (point to point) là phù hợp nhất. Các hệ thống DSM tiêu biểu là bộ vi xử lý Opteron, cũng như hầu hết các hệ thống tốc độ cao như các dòng máy của IBM, HP. Các hệ thống DSM chỉ làm việc tốt khi hệ điều hành hỗ trợ kiểu truy xuất NUMA. Ngày nay, hầu hết các hệ điều hành đều hỗ trợ kiểu truy xuất này.

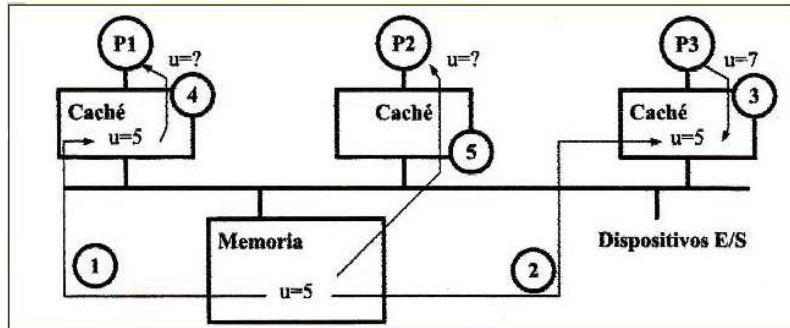
### Summary

- Symmetric multiprocessors (SMP)
  - Small number of cores
  - Share single memory with uniform memory latency
- Distributed shared memory (DSM)
  - Memory distributed among processors
  - Non-uniform memory access/latency (NUMA)
  - Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks



# CACHE COHERENCE PROBLEM

The presence of private caches in a multiprocessor necessarily introduces **cache coherence problems**, which can result in data inconsistency. That is, multiple copies of the same data can exist in different caches at a given time, and this is a potential problem.



**Figure 2.** Example of cache coherence problem.

The figure shows three processors with caches connected by a bus to a shared main memory. It is performed by the processor accesses a series of position  $u$ . First, the processor **P1** reads  $u$ , by bringing a copy to his cache. After the processor **P3** reads  $u$ , which also puts a copy in his cache. After this, the processor **P3** writes to the position or changing its value from 5 to 7. With a direct-write cache this will cause the main memory location update, but when the processor **P1** reread the position  $u$  (action 4), unfortunately reads the value obsolete (5) from its own cache instead of correct value (7) of main memory. The situation is even worse if the caches are writeback, because writing of **P3** could simply set the modified bit associated with the cache block that holds the position  $u$ , and update the main memory directly. Only when this cache block is subsequently replaced from **P3**'s cache, its contents would be written into main memory. Not only **P1** read the value obsolete, but when the processor **P2** reads the position  $u$  (action 5) do not find it in your cache and read the obsolete value of 5 from main memory, instead of 7. Finally, if multiple processors write different values for the position  $u$  in their writeback caches, the final value that will remain in the main memory is determined by the order in which the cache block containing the position or are replaced, and will have nothing to do with the order in which the writings of the position or occurred.

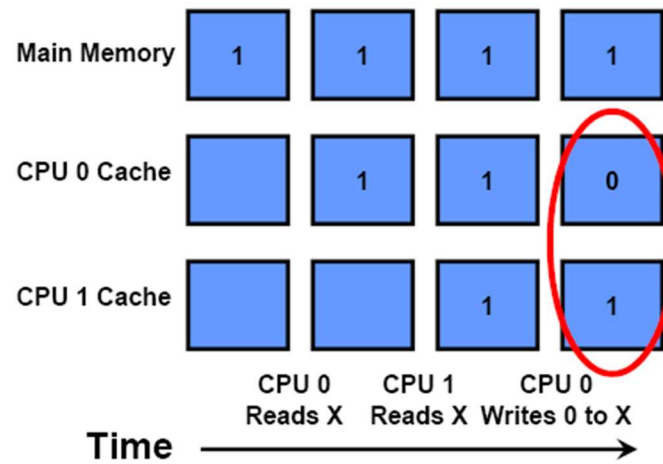
To resolve this issue we focus on cache coherence schemes based on hardware; the cache coherence needs to be treated as a basic issue in the design of hardware. For example, caching obsolete copies of a shared location must be removed when that location is changed, either invalidated or updated with the new value. Within the schemes, hardware-based cache coherence, we can differentiate between **network architectures with coherent cache** and **cache coherence protocols**. The network architectures propons multiprocessors with a hierarchy of buses, where the network traffic is reduced by hierarchical cache coherence protocols. There are two kinds of protocols to maintain cache coherence in multiprocessors:

- **Based on directory:** information about a physical memory block is held in a unique position.
- **Snoopy:** every cache has a copy of data from a physical memory block also has a copy of the information on it. These caches are commonly used in shared memory systems with a common bus.

Vấn đề về liên kết cache (Cache coherence): Khi các CPU đều có cache riêng, có thể dẫn đến sự không nhất quán trong dữ liệu (dữ liệu có nhiều bản copy khác nhau) → Cần các thủ tục (protocol) để giải quyết vấn đề liên kết các cache.

Ví dụ:

- CPU 0 đọc giá trị của X là 1 đưa vào Cache của mình
  - CPU 1 đọc giá trị của X là 1 đưa vào Cache của mình
  - CPU 0 ghi giá trị mới của X là 0 vào Cache của mình, trong khi giá trị của X trong Cache của CPU 1 vẫn là 1.
- ➔ Dữ liệu của X không nhất quán trong Cache của CPU 0 và CPU 1





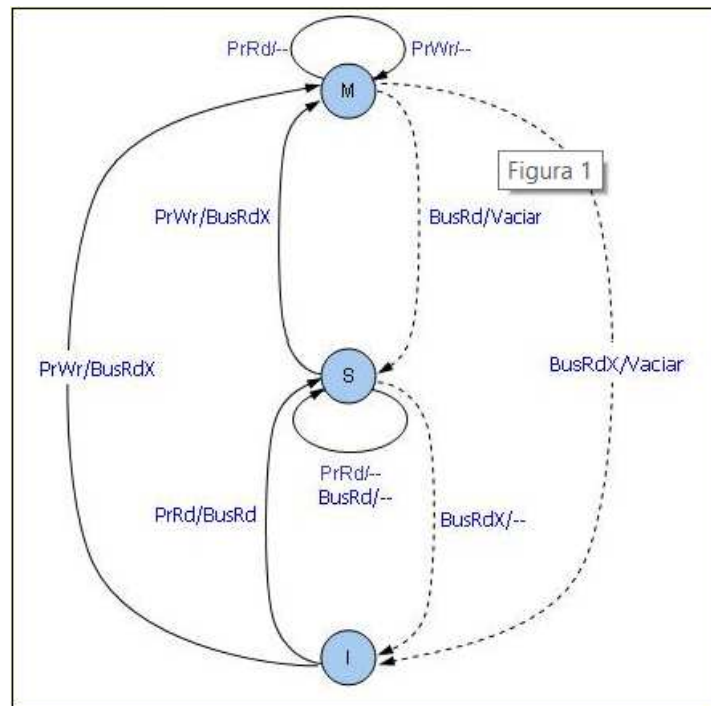
# COHERENCE CACHE PROTOCOLS

- In these protocols, the basic idea is for the cache controllers to monitor all transactions that occur on the shared bus, which serves naturally as a broadcast medium for the transactions, and ensure that no processor ever uses stale data. That is, the snooping cache controller takes action if a bus transaction is relevant to it, i.e. involves a memory block of which it has a copy in its cache, and updates its state suitably to keep its local cache coherent. A snoopy cache coherence protocol ties together two basic facets of computer architecture: bus transactions and the state transition diagram associated with a cache block.
  - Each block has a state associated with it, along with the tag and data, indicating the disposition of the block, e.g., invalid, valid, or dirty. State transitions for a block occur upon access to an address that maps to the block. In a snoopy cache coherence protocol, each cache controller receives two sets of inputs: the processor issues memory requests, and the bus snooper informs about transactions from other caches. In response to either, the controller updates the state of the appropriate block in the cache. All these updates, and other actions, are represented in the state transition diagram.
  - Many coherence protocols have been devised for write-back caches. In this simulator, you can choose among three very usual alternatives: [MSI](#) and [MESI](#) in a DSM system and , and [MSI](#), [MESI](#) and [DRAGON](#) in a SMP system
- 
- Ý tưởng cơ bản của các bộ điều khiển cache là giám sát mọi giao dịch xảy ra trên bus chung (là phương tiện truyền giao dịch) để đảm bảo rằng không có bộ vi xử lý nào sử dụng dữ liệu cũ (stale data). Bộ điều khiển cache snoopy sẽ hoạt động nếu có một giao dịch liên quan đến nó, đó là nó sẽ quản lý block trong main memory (block có 1 bản copy trong cache) và cập nhật trạng thái phù hợp để đảm bảo cache cục bộ nhất quán.
    - o Mỗi block ngoài thông tin về tag và data, thì có thêm thông tin về trạng thái (state): invalid, valid hoặc dirty.
    - o Giao dịch xảy ra với 1 block khi có một truy xuất đến một địa chỉ ứng với block đó.
    - o Trong thủ tục điều khiển cache snoopy, mỗi bộ điều khiển cache nhận hai tập hợp đầu vào:
      - Bộ vi xử lý phát tra yêu cầu bộ nhớ
      - Nhận thông báo về giao dịch từ các cache khác.
    - o Khi đáp ứng, bộ điều khiển cache cập nhật trạng thái tương ứng của cache (Tất cả các cập nhật và hoạt động khác được biểu diễn trong lược đồ chuyển trạng thái (state transition diagram))
  - Đa phần các thủ tục liên kết cache thiết kế cho các cache theo chính sách write-back.
  - Trong bộ mô phỏng này:
    - o Với hệ thống DSM, có các thủ tục: [MSI](#) và [MESI](#)
    - o Với hệ thống SMP, có các thủ tục: [MSI](#), [MESI](#) và [DRAGON](#)

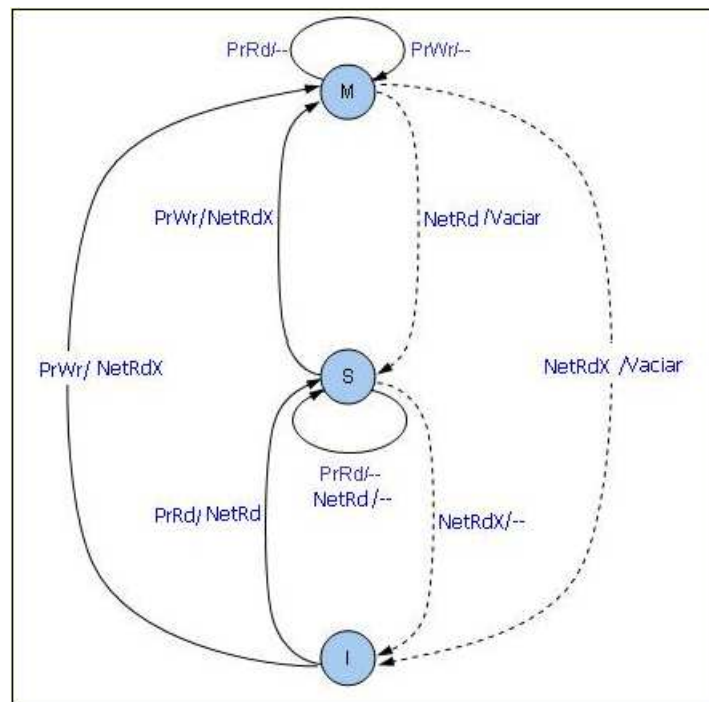
*(Xem từng thủ tục (protocol) ở các phần sau. Không đi chi tiết, chỉ để tham khảo)*

# **MSI PROTOCOL**

- **MSI** is a cache coherence protocol based on invalidations to write-back caches. In invalidation-based protocols, when a processor writes to a shared position, the state of the cache memory block in the caches of other processors, which contain the block, is changed to invalid. This protocol uses the three states that require any writeback cache: invalid (I), shared (S), modified (M). I has the obvious meaning. The S means that the block is present without change in one or more processor caches, main memory copies and these caches are all updated. M (or dirty) means that only one processor has a valid copy of the block in its cache, main memory copy is obsolete and no other cache can have the block in a valid state (S or M).
- In a DSM system, the processor issues two types of requests, reads (PrRd) and writings (PrWr). In addition, the communications assistant allows the following transactions: Net Read (NetRd), Net Read-Exclusive (NetRdX) and NetWriteback (NetWb). In a NetRd, the cache controller puts the address on the network and requests a copy of the block that tries to read (PrRd miss). In a NetRdX, the cache controller puts the address on the network and requests an exclusive copy of the block that tries to modify (PrWr of a block that has modified state in the cache). In NetWb, the cache controller sets the direction and content of the memory block in the network, and updates the main memory.
- In an SMP system, the processor issues the same request in a DSM system. On the bus are allowed the following transactions: Bus Read (BusRd), Bus Read-Exclusive (BusRdX) and Bus Writeback (BusWb). In a BusRd, the cache controller puts the address on the bus and asks for a copy of the block that tries to read (PrRd miss). In a BusRdX, the cache controller puts the address on the bus and requests an exclusive copy of the block to try to modify (PrWr of a block that has modified state in the cache). In a BusWb, the cache controller sets the direction and content of the memory block on the bus, and updates the main memory.
- In addition to changing the status of blocks in the cache, the cache controller may be involved in the transactions of the network and "empty" block referenced content to the requesting node, instead of allowing the main memory to supply the data.
- The state transition diagram for the MSI protocol is shown in Figures 1 and 2. This chart takes as input the current state of the request block and the transaction processor or informed by the assistant, and outputs the next state for the cache block. In this diagram, the notation A / B means when viewed from the standpoint of the processor or the wizard the event A, in addition to the change of state, generate the transaction or action B. If a transition is not shown, means that interest does not need or should not take any action accordingly.



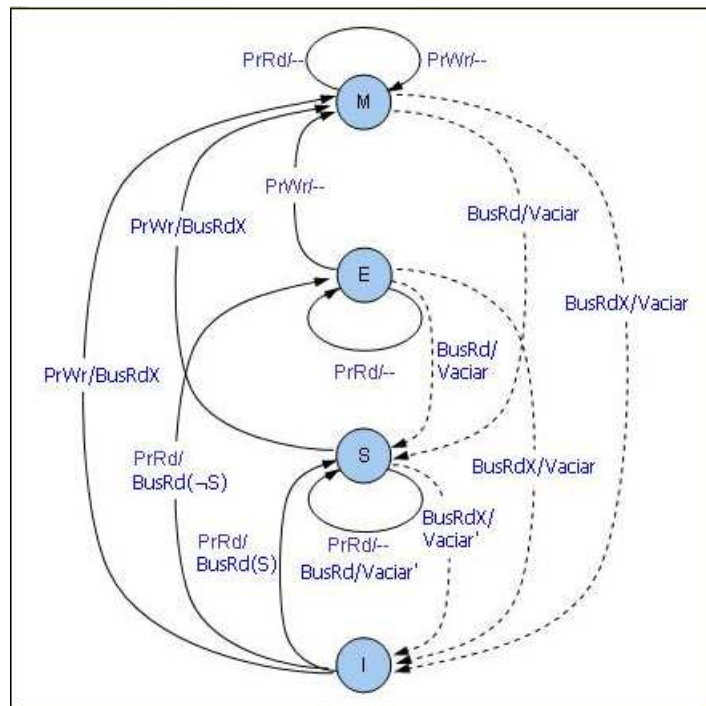
**Figure 3.** MSI Invalidation protocol in an SMP system.



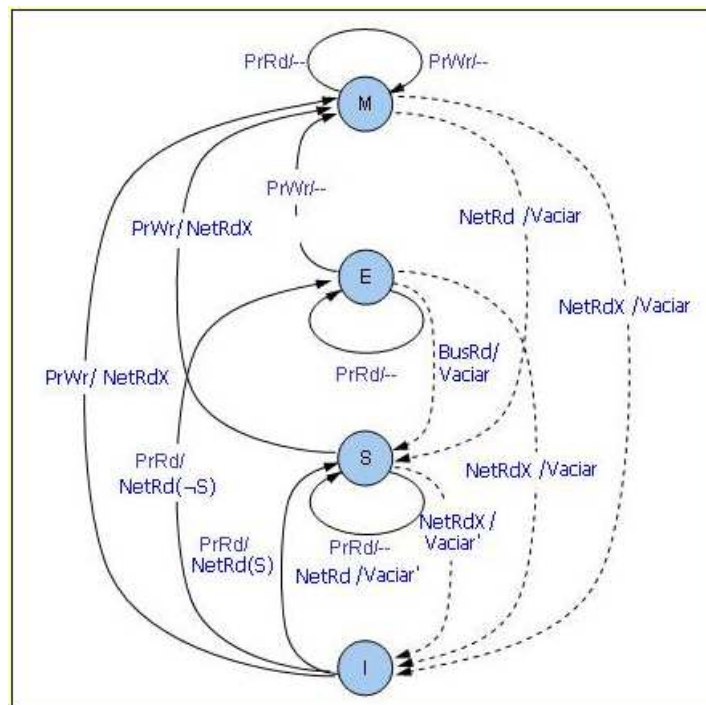
**Figure 4.** MSI Invalidation protocol in an DSM system.

# **MESI PROTOCOL**

- **MESI** protocol (or Illinois) is a cache coherence protocol based on invalidations to write-back caches. In invalidation-based protocols, when a processor writes to a shared position, the state of the cache memory block in the caches of other processors, which contain the block, is changed to invalid. The MESI protocol is only an improvement in the [MSI](#) protocol (adds a status of "exclusive") in the sense that it reduces the number of network transactions that take place due to the coherence protocol.
- This protocol uses four block states: invalid (I), shared (S), exclusive (E), modified (M). I and M have the same meaning as in the [MSI](#) protocol. E (clean or not shared) means that only one cache (the cache) has a copy of the block, and has not been modified (the main memory is updated). The S means that potentially two or more processors (including the cache) have this block in its cache in an unmodified state. In conclusion, this protocol requirement places a new network messages or bus transactions. To distinguish between two types of transactions, shared, represented by the notation (S), and exclusive, symbolized by the notation (-S) available for cache controllers determined in a ruling read if any other cache contains the data (block), and to decide whether to load the block depending on state E or S. Consequently, the notation NetRd (S) or BusRd (S) means that when the transaction occurred reading the transaction was received indicating sharing and exclusivity. A simple BusRd or NetRd means that do not care about the distinction between shared and unique transactions for that transition.
- As in the [MSI](#) protocol, there are two requests from the processor (PrWr and PrRd), three transactions on the net for a DSM, NetRd, NetRdX, and NetWb, and three transactions on the bus for an SMP system, BusRd, BusWb and BusRdX. In addition to the Flush action, now, there is also Flush action. "Where possible, the protocol also makes caches instead of main memory, provide data for transactions NetRd, NetRdX, BusRd and BusRdX. Since multiple processors can have a copy of memory block in its cache, we need only choose one to provide data on the network. Flush' is true only for the processor, the remaining processors perform no action.
- The state transition diagram for the MESI protocol is shown in Figure 3 and 4. The notation is equivalent to the same as the [MSI](#) protocol.



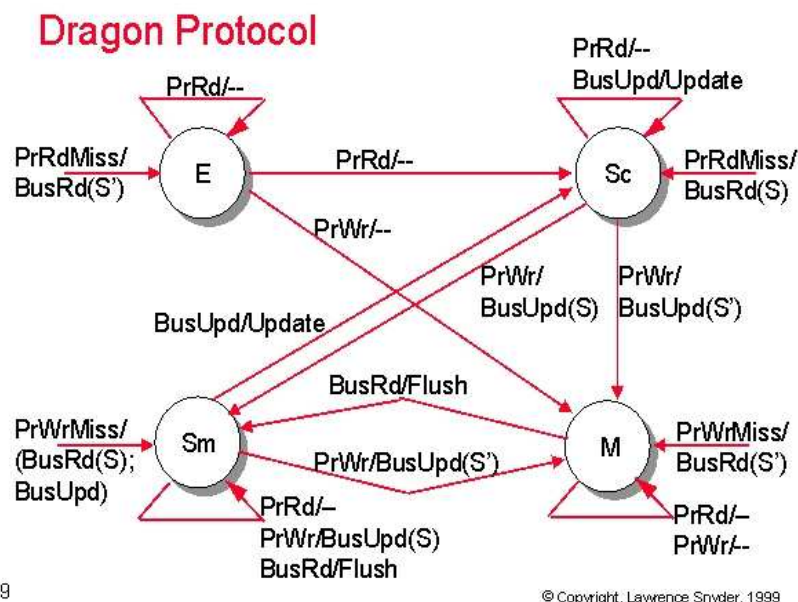
**Figure 5.** MESI Invalidation protocol in an SMP system.



**Figure 6.** MESI Invalidation protocol in an DSM system.

# **DRAGON PROTOCOL**

- **DRAGON** is a cache coherence protocol based on updates for write-back cache. In update-based protocols, when a processor writes a memory block, it is updated in the caches of other processors that have a copy of that block.
- This protocol uses four states: exclusive (E), shared-no-modified (SC), shared-modified (SM) and modified (M). E and M have the same meaning as in the [MESI](#) protocol. SC means that potentially two or more processors (including this cache) have this block in its cache. The main memory may or may not be current. SM means that potentially two or more processors have this block in its cache, main memory is out of date, and is the responsibility of this process the main memory upgrade while this block is replaced in the cache. As in the MESI protocol is necessary shared signal (S).
- The processor requests are similar to those of protocol [MESI](#). Besides PrRd and PrWr, there are two requests more when a block of memory is accessed for the first time: Read-miss (PrRdMiss) and Write-miss (PrWrMiss).
- As for bus transactions, the DRAGON protocol maintains BusRd and BusWb and adds a new request for update (BusUpd). This transaction takes the specific word written by the processor and spread on the bus to all other caches can be updated.
- Finally, with regard to actions, the only new capacity is needed for the cache controller updates a block of memory in the local cache with the content being broadcast in the bus for a transaction BusUpd relevant.
- Figure 7 shows the state diagram for this protocol.

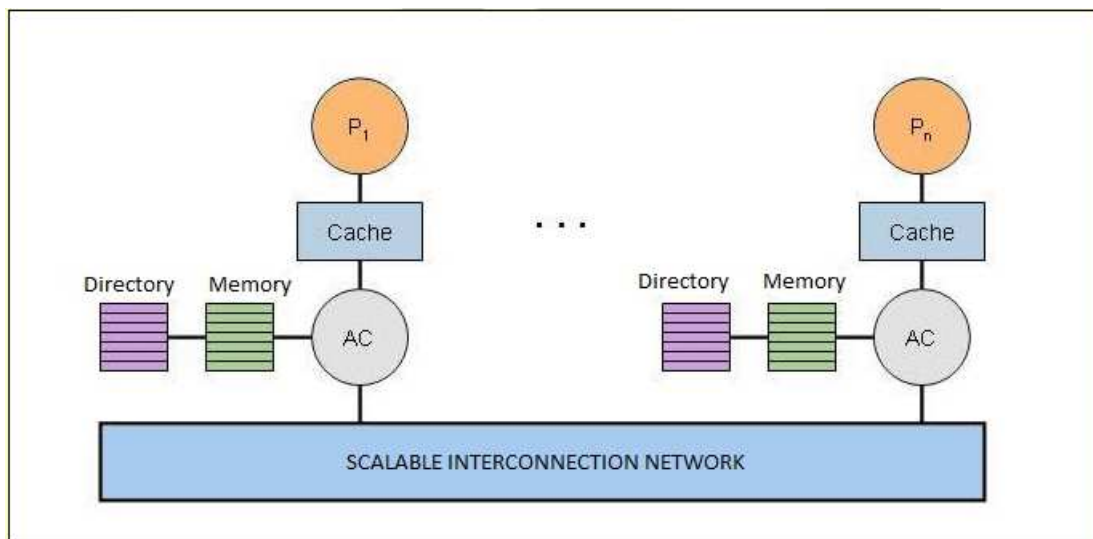


**Figure 7. DRAGON protocol**

# **DIRECTORY PROTOCOLS**

Scalable [cache coherence](#) is typically based on the concept of directory. Since the state of a block in the [caches](#) can not be determined implicitly by a request on a shared bus and having watching through cache controllers, the idea is to maintain this state explicitly in a place -called directory- where requests can go and search the block state.

Imagine that each block of main memory is associated with a record that contains information about the caches that contain a copy of the block and the state in those caches. This record is called **directory entry** for that block.



In a **scalable multiprocessor with directories** for each block of main memory, the same size as a cache block, has a directory entry that holds data copies in the caches and their corresponding states.

Like bus-based systems, there may be many caches with a clean block, that is readable, but if the block is modifiable (can write) in a cache, then only the cache has a valid copy of the block. When a node incurs a cache miss, first communicating with the directory entry for the block using a peer-to-peer transaction network. Because the directory entry is in main memory for the block, its position can be determined from the direction of the block. From the directory, the node determines where the copies are valid in the caches (if any) and what further action should be performed. Then it communicates with the copies in the caches, if necessary using additional network transactions. For example, you can get a block of another node or modified during a write operation, send invalidations to other nodes and get their respective acknowledgments. Also communicate the changes resulting from the states of the blocks in the cache directory entry by network transactions, so that the directory is updated

Trong các bộ đa vi xử lý (multiprocessor) có thể mở rộng (scalable), đều có một mục thư mục (directory entry)/bản ghi (record) gắn với mỗi cache chứa các bản copy dữ liệu trong cache và trạng thái tương ứng của nó.



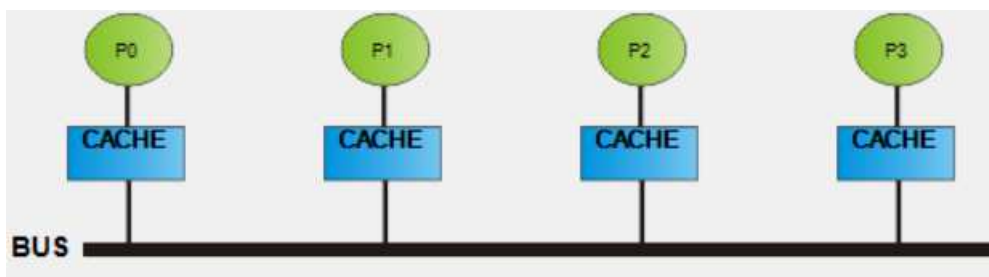
# **BUS ARBITRATION**

Chỉ có ở tổ chức bộ nhớ kiểu SMP (Symmetric Multi-Processors)

Since the bus is a shared resource, a mechanism must be provided to resolve contentions. There are a lot of conflict-resolution methods. These methods include static or fixed priorities, FIFO queues, etc.; or even, the use of schemes like: LRU, LFU, or Random. These schemes have a explanation similar to the replacement policies with the same name. For example, the LRU scheme is a dynamic priority algorithm which gives the highest priority to the requesting processor that has not used the bus for the longest interval.

Do bus là nguồn tài nguyên chung, do đó sẽ xảy ra các xung đột khi sử dụng chung bởi các thành phần khác nhau → Cần các chính xác phân xử/trọng tài (arbitration) buss. Gồm:

- LRU: Gán mức ưu tiên cao nhất cho bộ vi xử lý (có yêu cầu) chưa được sử dụng bus trong một thời gian dài.
- LFU: Gán mức ưu tiên cao nhất cho bộ vi xử lý (có yêu cầu) với với tần xuất sử dụng bus ít nhất.
- Random: Gán ưu tiên sử dụng bus một cách ngẫu nhiên



## II. Giới thiệu SMPCache

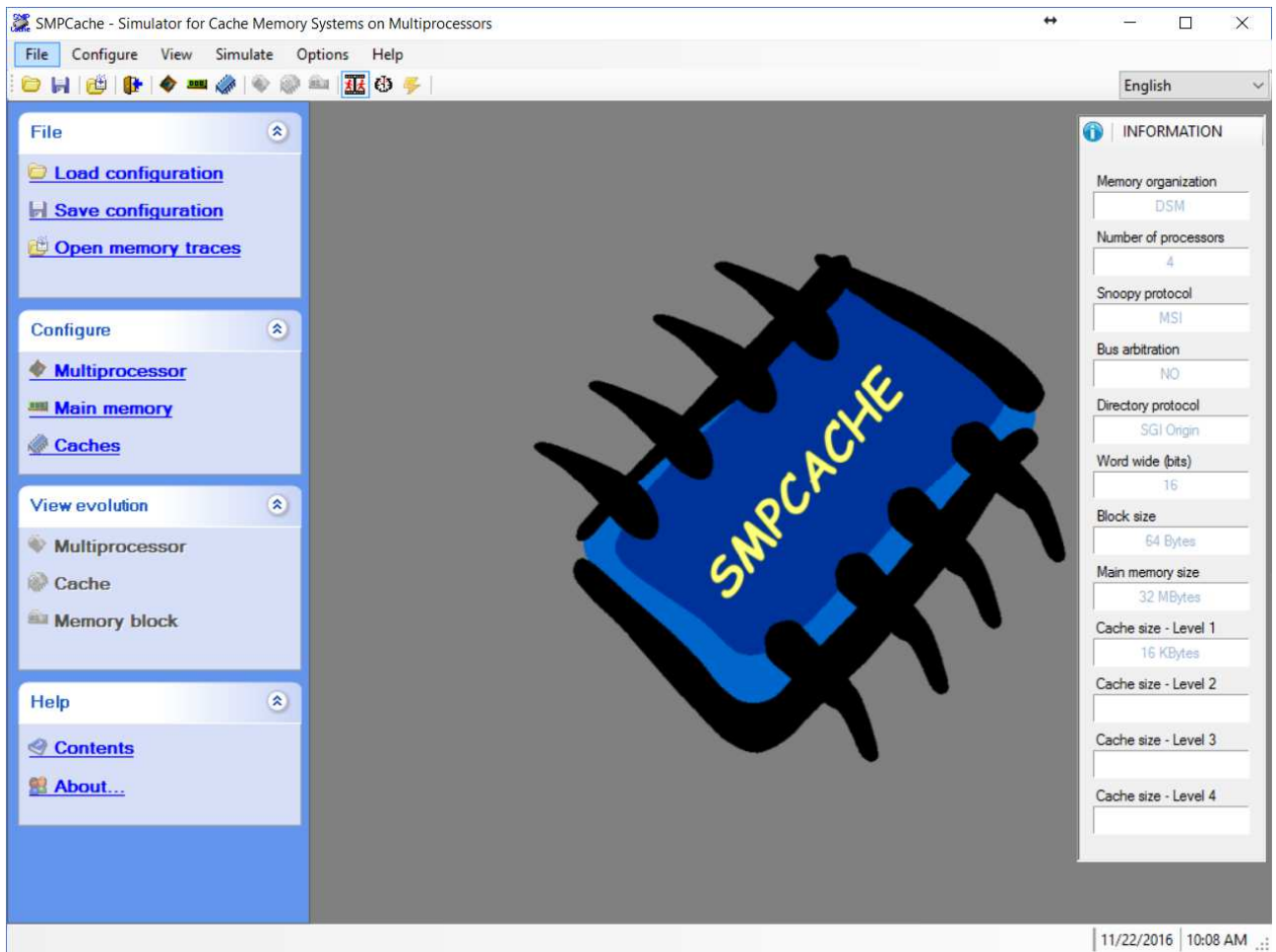


Chạy SMPCache



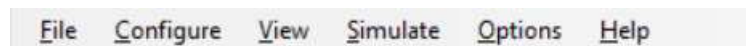
## 1. Giao diện

- Sau khi khởi động. Giao diện chính của SMPCache như sau



Gồm các phần sau:

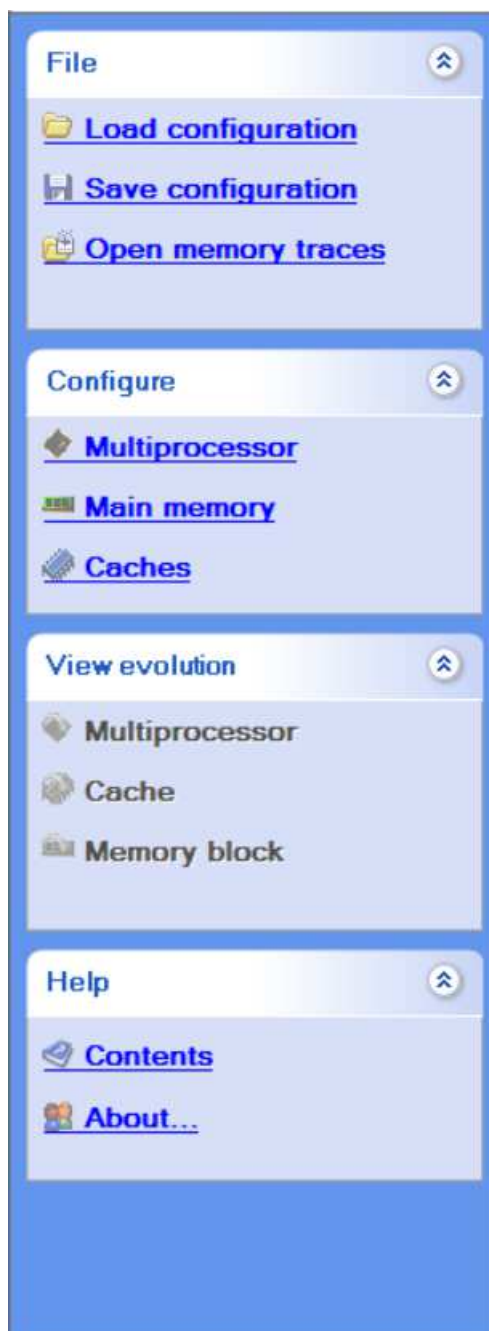
- Menu



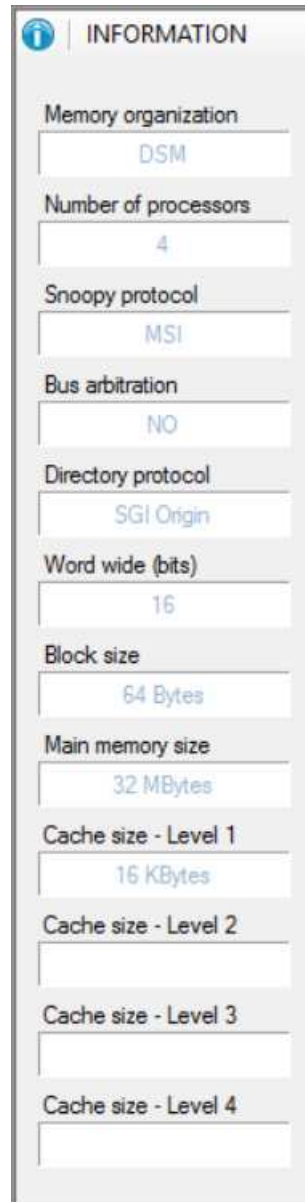
- Toolbar



- Những chức năng chính của SIMCache (Cửa sổ bên trái)



- Cấu hình chính của Machine (Processor, Main memory, Cache) (Cửa sổ bên phải)



The 'INFORMATION' window displays the following configuration parameters:





- Memory organization: DSM
- Number of processors: 4
- Snoopy protocol: MSI
- Bus arbitration: NO
- Directory protocol: SGI Origin
- Word wide (bits): 16
- Block size: 64 Bytes
- Main memory size: 32 MBytes
- Cache size - Level 1: 16 KBytes
- Cache size - Level 2: (empty)
- Cache size - Level 3: (empty)
- Cache size - Level 4: (empty)

## 2. Quy trình

- Gồm 3 bước:
  - o Bước 1:
    - Thiết lập mô phỏng thông qua các cấu hình (configuration). Cấu hình gồm có của processor, cache và main memory







- Có thể Tạo ra một cấu hình mới rồi lưu lại Hoặc nạp một cấu hình đã lưu/có ra sửa dụng

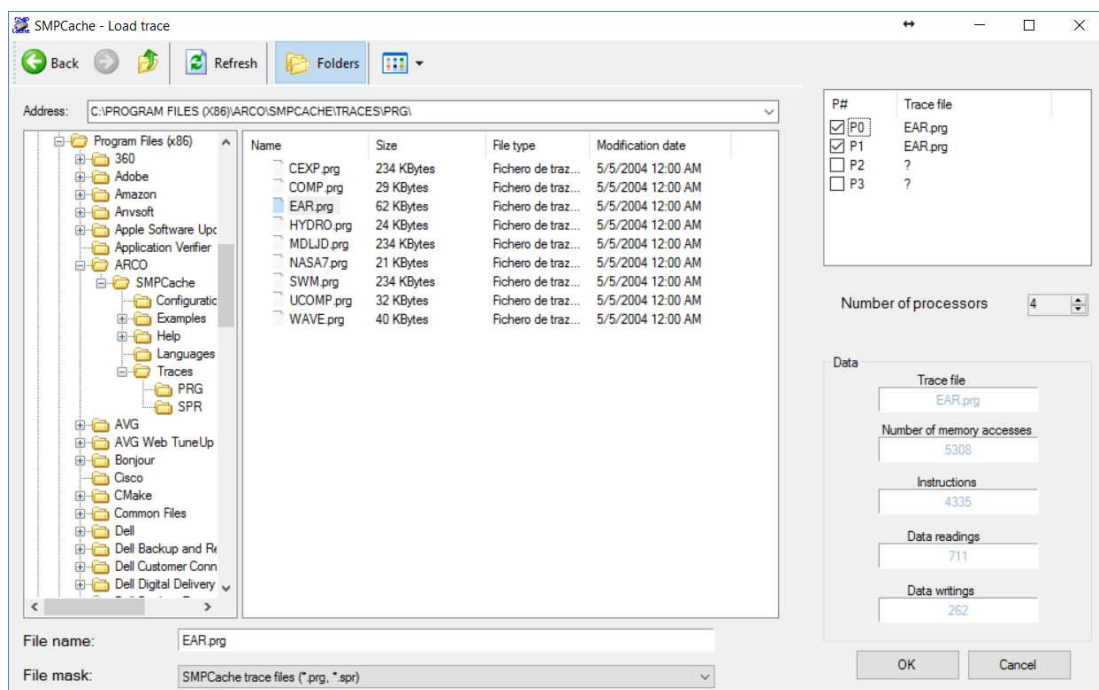
	Load configuration	F2
	Save configuration	F3
	Open memory traces	Alt+F2
	Exit	F3

- Bước 2:

- Mở một file memory traces (lưu các truy xuất bộ nhớ) vào processor, để xem quá trình thực hiện của processor, cache và main memory

	Load configuration	F2
	Save configuration	F3
	Open memory traces	Alt+F2
	Exit	F3

- Ví dụ: Chú ý lựa chọn các processor

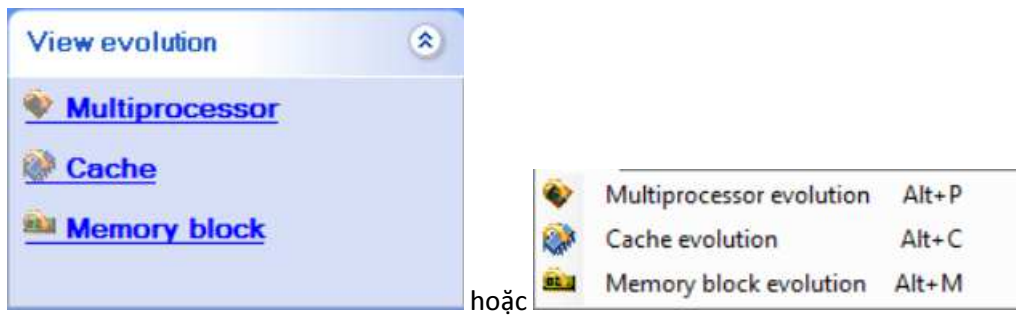


- Bước 3: Thường lựa chọn Complete execution

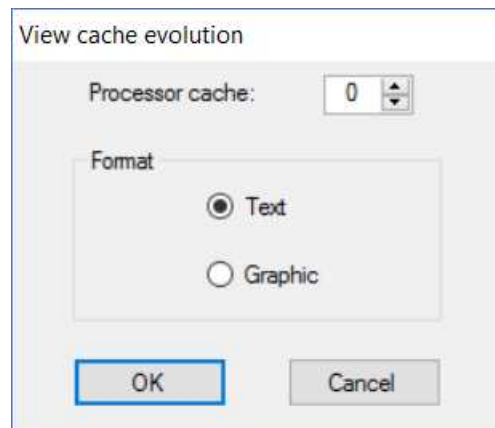
- Lựa chọn chế độ thực hiện mô phỏng

<input checked="" type="checkbox"/>	Step by step	F7
<input type="checkbox"/>	Breakpoint	F8
<input type="checkbox"/>	Complete execution	F9

- Bước 4:
  - o Lựa chọn chế độ hiển thị và thực hiện mô phỏng



- o Chế độ hiển thị



- Chế độ Text sẽ hiển thị các thông tin chi tiết dạng bảng thống kê
- Chế độ Graphic sẽ hiển thị thông tin dạng biểu đồ.