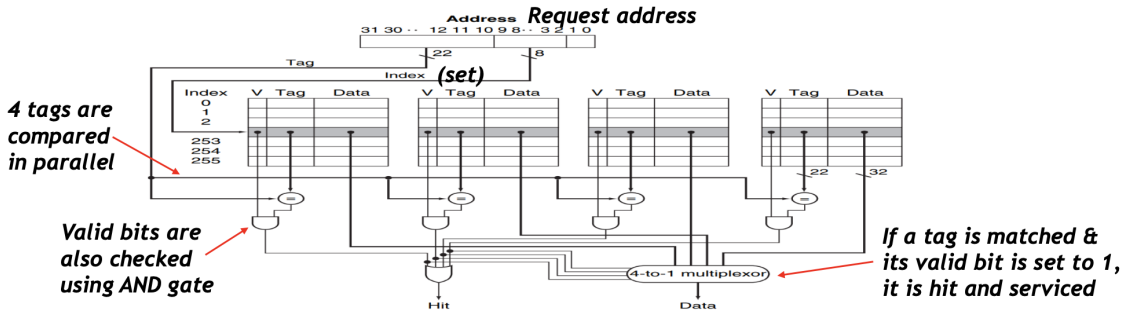
Computer Architecture (ENE1004)

▼ Lec 21

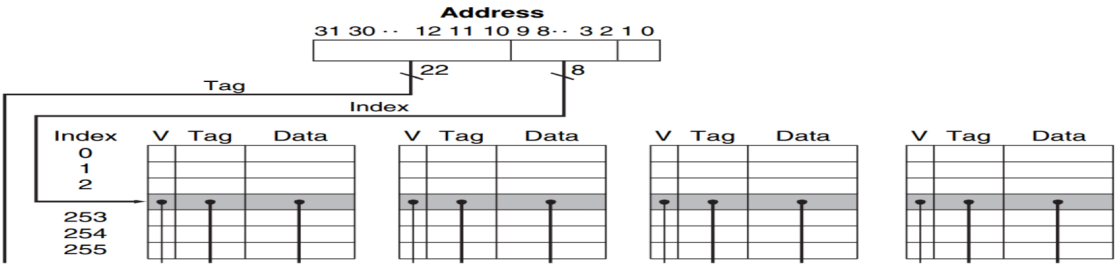
Lec 21: Large and Fast : Exploting Memory Hierarchy 5

How to Locate a Block in the Cache?

- How can a processor find a data block in a set-associative cache? (프로세서는 집합 연관 캐시에서 데이터 블록을 어떻게 찾을 수 있을까?)
  - In a direct-mapped cache, (i) find “index”, and (ii) check its “valid bit” & “tag” (직접 매핑된 캐시에서는 (i) “인덱스”를 찾고, (ii) “유효한 비트” 및 “태그”를 확인한다.)
  - In a set-associative cache, (i) find “set”, and (집합 연관 캐시에서, (i) “set(집합)”을 찾고, 그리고)
  - (ii) check “valid bit” & “tag” for all the blocks within the set ((ii) set(집합) 내의 모든 블록에 대해 “유효한 비트” 및 “태그”를 확인한다.)
- In a *n* way set-associative cache, *n* blocks in a set should be compared (n 방향 집합 연관 캐시에서는 한 set(집합)의 n개 블록을 비교해야한다.)
- An example: a 4-way set-associative cache with 1,024 1-word blocks (예: 1,024개의 1 word 블록이 있는 4방향 집합 연관 캐시)
  - 2 bits for 1-word block’s byte offset, 8 bits for 1024/4 sets, 22 (32-10-2) bits for tag (1 word 블록의 바이트 오프셋은 2비트, 1024/4 세트는 8비트, 태그는 22(32-8-2)비트이다)

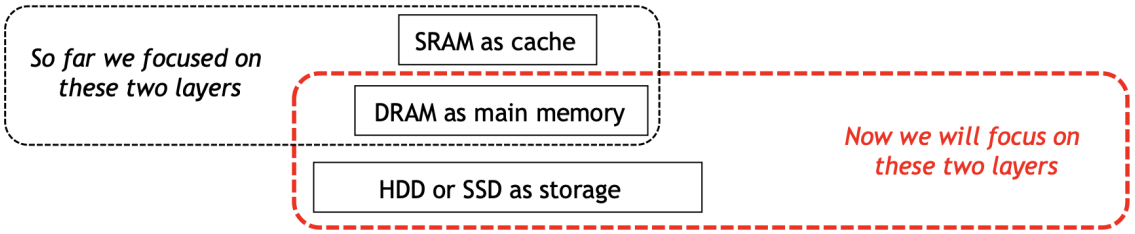


Choosing Which Block to Replace for a Miss?



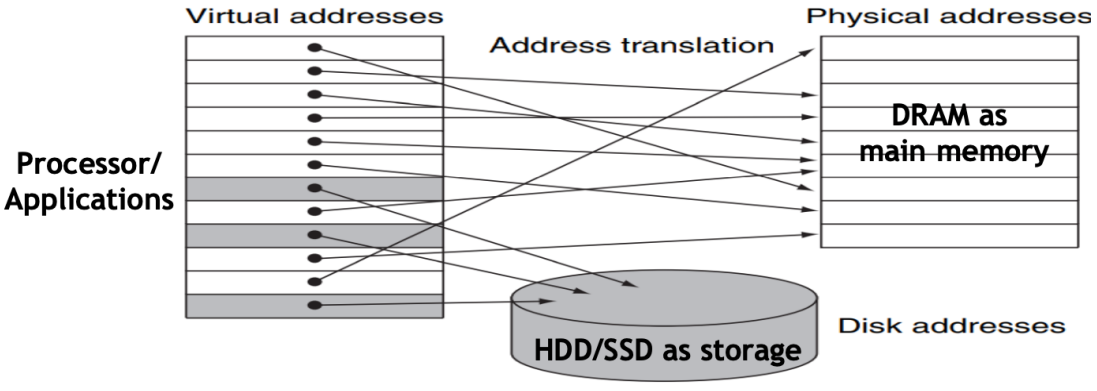
- Assume that, given a request, we found no tag match in the set (요청이 주어졌을때 세트(집합)에서 일치하는 태그를 찾지 못했다고 가정해보겠다)
  - This is a miss; the requested data block does not exist in the cache (이것은 miss이다; 요청된 데이터 블록이 캐시에 존재하지 않는다)
  - The request data block should be brought from the main memory and stored in the set (요청 데이터 블록을 메인 메모리에서 가져와서 세트에 저장해야 한다)
  - To store the requested block into the set, an existing block should be kicked out from the cache (요청된 블록을 세트에 저장하려면 기존 블록을 캐시에서 쫓아내야 한다)
  - Among the existing four blocks in the set, we must choose one block to replace (세트에 있는 기존 4개의 블록 중 대체할 블록 하나를 선택해야 한다)
- The most commonly used scheme is to pick least-recently-used (LRU) one (가장 일반적으로 사용되는 방식은 최근에 가장 적게 사용된 블록 (LRU)을 선택하는 것이다)
  - The block replaced is the one that has been unused for the longest time (대체되는 블록은 가장 오랫동안 사용되지 않은 블록이다)
  - It does make sense as it may be able to improve the temporal locality (이는 시간적 지역성을 개선할 수 있으므로 의미가 있다)
  - See the slide “Misses and Asso... (3) – 2-way Set Asso.”; Memory[8] is selected for Memory[6] (“Misses and Asso... (3) – 2-way Set Asso.” 슬라이드 참조; Memory[8]은 Memory[6]에 대해 선택된다)

Virtual Memory (가상 메모리)



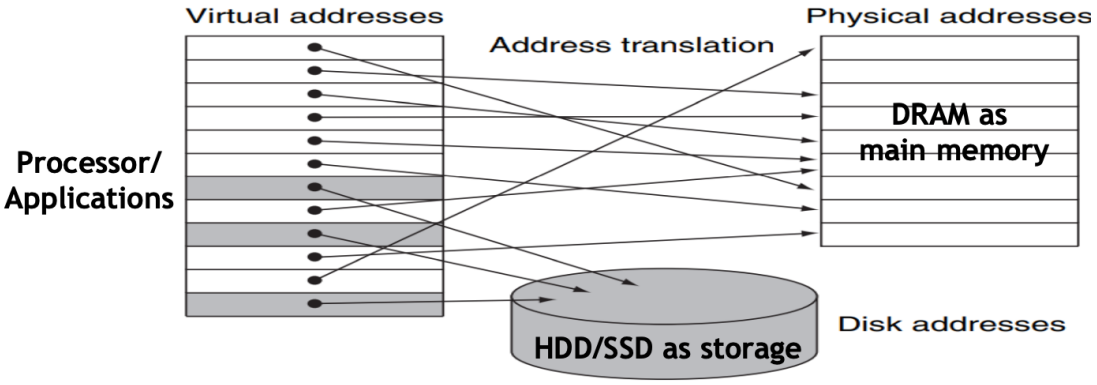
- Now, we will focus on another two-layers memory system: main memory + storage (이제 또 다른 2계층 메모리 시스템인 메인 메모리 + 스토리지에 집중하겠다)
  - All data sets of applications are stored in the storage (애플리케이션의 모든 데이트 세트는 스토리지에 저장된다)
  - The main memory hold part (subset) of all the data sets in the storage (메인 메모리는 스토리지에 있는 모든 데이터 세트의 일부(하위 집합)를 보유한다)
  - A CPU feels like the main memory holds all the data sets (CPU는 메인 메모리가 모든 데이터 세트를 보유하고 있는 것처럼 느껴진다)
  - So, particularly, we call this system “virtual memory” (따라서 특히 이 시스템을 “가상 메모리”라고 부른다)
- Page: The base unit of the virtual memory is called “page” (vs block/line in cache) (Page : 가상 메모리의 기본 단위를 “page”라고 한다 (캐시의 블록/라인과 비교))
- Page fault: A virtual memory miss is called “page fault” (Page 오류 : 가상 메모리 누락을 “page 오류”라고 한다)

Virtual Address and Physical Address



- DRAM can be accessed using its device addresses; we call them "physical addresses" (DRAM은 장치 주소를 사용하여 엑세스할 수 있으며, 이를 "물리적 주소"라고 부른다)
- However, a CPU does NOT use these physical addresses (그러나 CPU는 이러한 물리적 주소를 사용하지 않는다)
  - Instead, the processor uses "virtual addresses" (대신 프로세서는 "가상 주소"를 사용한다)
  - Memory addresses generated in load/store, branch, and jump instructions are virtual addresses (load/store, branch, and jump 명령어에서 생성된 메모리 주소는 가상 주소이다)
  - This is because all the data of applications cannot be accommodated by limited capacity of DRAM (이는 애플리케이션의 모든 데이터를 제한된 용량의 DRAM으로 수용할 수 없기 때문이다)
- Actually, an application running on the CPU has part of its data in the main memory
  while the remaining data in the storage; but, the CPU does NOT care about it and
  just uses the virtual addresses of the application (실제로 CPU에서 실행되는 애플리케이션은 데이터의 일부가 메인 메모리에 있고 나머지 데이터는 스토리지에 있다. 그러나 CPU는 그것에 대해 신경 쓰지 않고 애플리케이션의 가상 주소만 사용한다.)

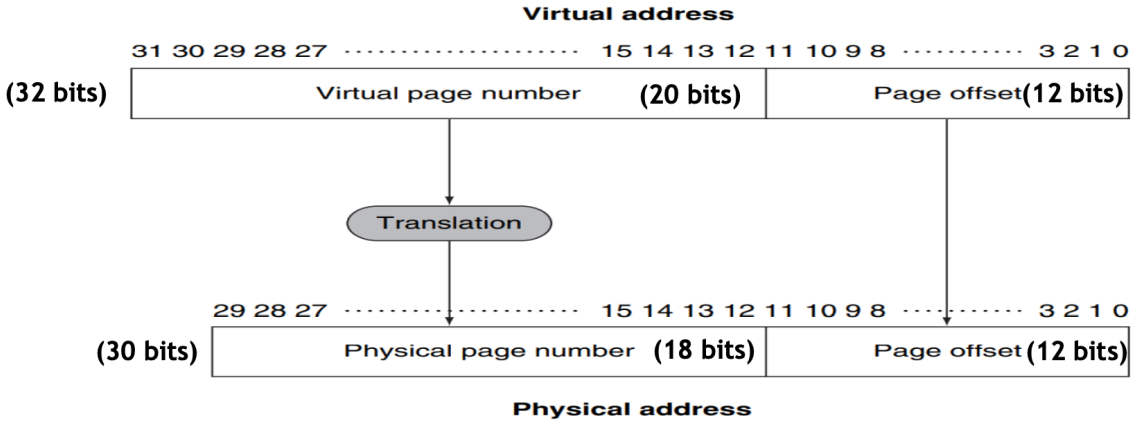
Address Translation: Virtual → Physical Address (1)



- To request a data in the main memory, an address translation is required (메인 메모리에 있는 데이터를 요청하려면 주소 변환이 필요하다)

    - Processor requests a data item with its virtual address (프로세서가 가상 주소로 데이터 항목을 요청한다)
    - DRAM can be accessed using its physical address (DRAM은 물리적 주소를 사용하여 액세스 할 수 있다)
- Note that a data of an application can be in either the main memory or the storage (애플리케이션의 데이터는 메인 메모리 또는 스토리지에 있을 수 있다)
    - If the data is in the main memory, the virtual address → the physical address (데이터가 메인 메모리에 있는 경우, 가상 주소 → 물리적 주소)
    - If the data is in the storage, the virtual address → the disk address (데이터가 스토리지에 있는 경우, 가상 주소 → 디스크 주소)

Address Translation: Virtual → Physical Address (2)

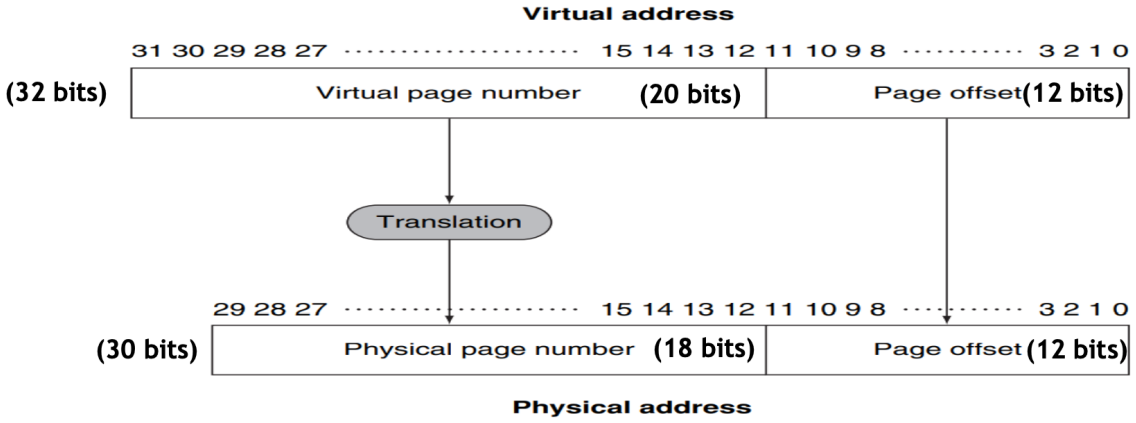


- Recall that the base unit in the virtual memory is “page” (가상 메모리의 기본 단위는 “page”라는 것을 기억해라)
    - The virtual address for a page can be divided into “virtual page number” + “page offset” (page의 가상 주소는 “가상 page 번호” + “page 오프셋”으로 나눌 수 있다)
    - The page offset indicates a specific byte within the page (recall “byte offset” in cache) (페이지 오프셋은 페이지 내의 특정 바이트를 나타낸다 (캐시의 “바이트 오프셋”을 떠올려라))
    - So, we can infer the size of a page from the number of bits for the page offset (따라서 페이지 오프셋의 비트 수에서 페이지의 크기를 유추할 수 있다)

- Here, the lower 12 bits of a 32-bit address is used as page offset; the page size is 2^12 bytes (4 KiB) (여기서는 32비트 주소의 하단 12비트를 페이지 오프셋으로 사용하며, 페이지 크기는 2^12 바이트 (4 KiB)이다)

- The upper portion of the virtual address indicates “virtual page number” (가상 주소의 위쪽 부분은 “가상 페이지 번호”를 나타낸다)

Address Translation: Virtual → Physical Address (3)



- Address translation: virtual page number → physical page number (주소 변환 : 가상 페이지 번호 → 실제 페이지 번호)
  - The page offset does NOT change; it is for specifying bytes within a page (페이지 오프셋은 변경되지 않는다; 이것은 페이지 내 바이트 지정을 위한 것이다)
  - In general, # of virtual pages is much larger than # of physical pages (sizes of apps >>> DRAM) (일반적으로 가상 페이지의 개수는 물리적 페이지의 개수보다 훨씬 크다 (app의 크기 >>> DRAM))
- Here, we can infer the DRAM size from the number of bits in the physical address (여기서 물리적 주소의 비트 수에서 DRAM 크기를 유추할 수 있다)
  - The physical address is 30-bit (2^30 bytes = 1 GB) (물리적 주소는 30비트 (2^30바이트 = 1GB)이다)
  - There are 2^18 pages, each of which is 2^12 bytes, in this 1 GB DRAM (이 1GB DRAM에는 각각 2^12바이트인 2^18개의 페이지가 있다)