

## 20. Making Virtual Memory Faster (TLBs)

---

EECS 370 – Introduction to Computer Organization – Winter 2009

**Prof. Todd Austin & Prof. Marios Papaefthymiou**

EECS Department  
University of Michigan in Ann Arbor, USA

© T. Austin & M. Papaefthymiou, 2009

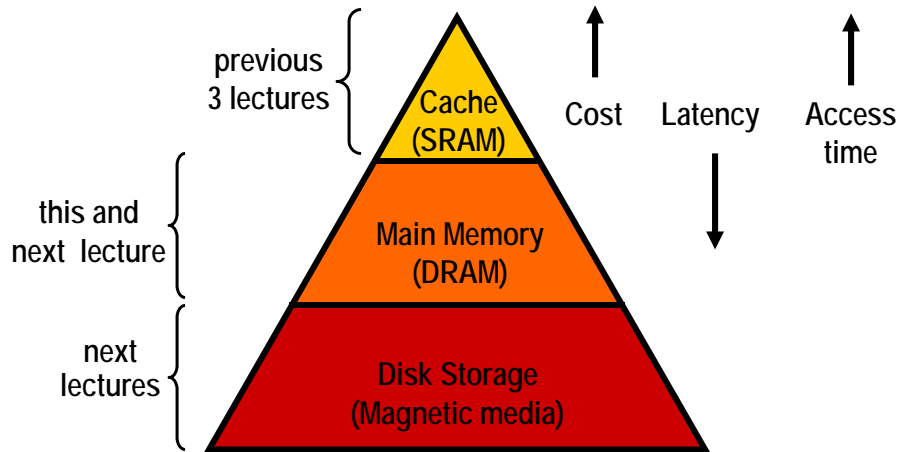
The material in this presentation cannot be  
copied in any form without our written permission

### This week in 370

---

- ❑ Today: TLBs
- ❑ Next lecture: Disk storage
- ❑ And, after that: Where are CPUs heading? GP-GPUs
- ❑ Early reminder  
FINAL EXAM – April 24<sup>th</sup>, 2009 – 7.00pm to 9.00pm  
In DOW 1013 and DOW 1014

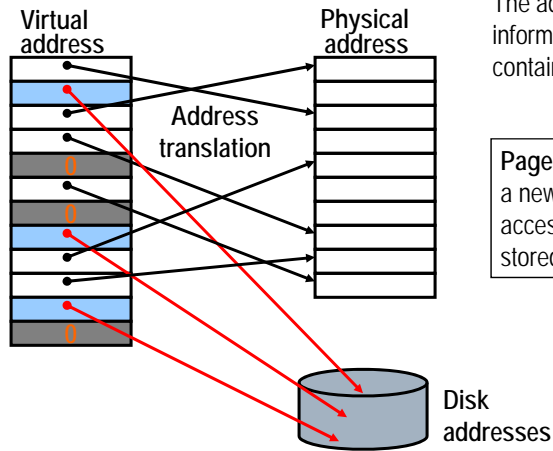
## Last lecture: Storage Hierarchy



## Last lecture: Virtual Memory

- ❑ Virtual memory lets the programmer “see” a memory array **larger** than the DRAM available on a particular computer system.
- ❑ Virtual memory enables multiple programs to share the physical memory without:
  - Knowing other programs exist (**transparency**).
  - Worrying about one program modifying the data contents of another (**protection**).

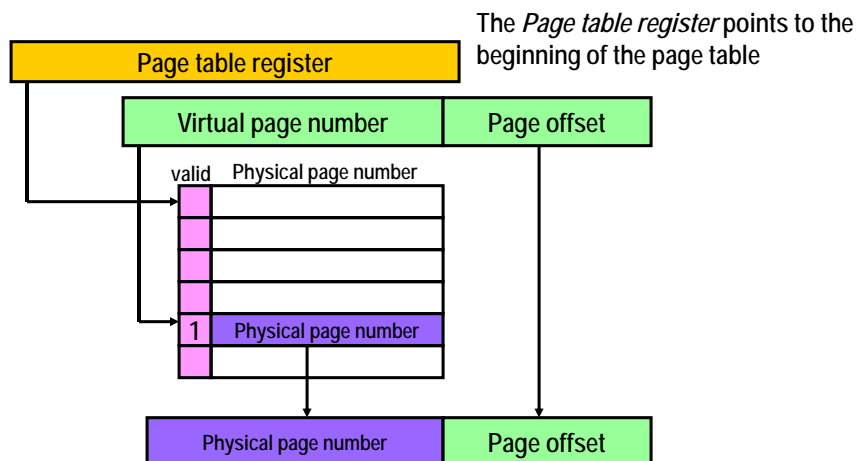
## Last lecture: Address Translation



The address translation information of the program is contained in the *Page Table*

**Page Table:**  
a new data structure accessed by the OS and HW stored in memory

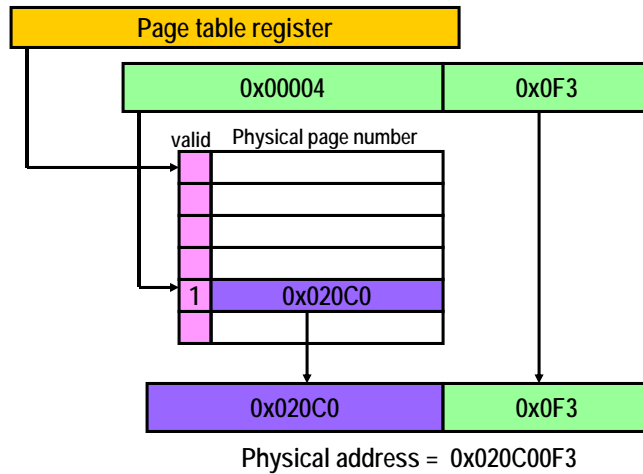
## Last lecture: How does it work?



The *Page table register* points to the beginning of the page table

## Last lecture: Page table components - Example

Virtual address = 0x000040F3

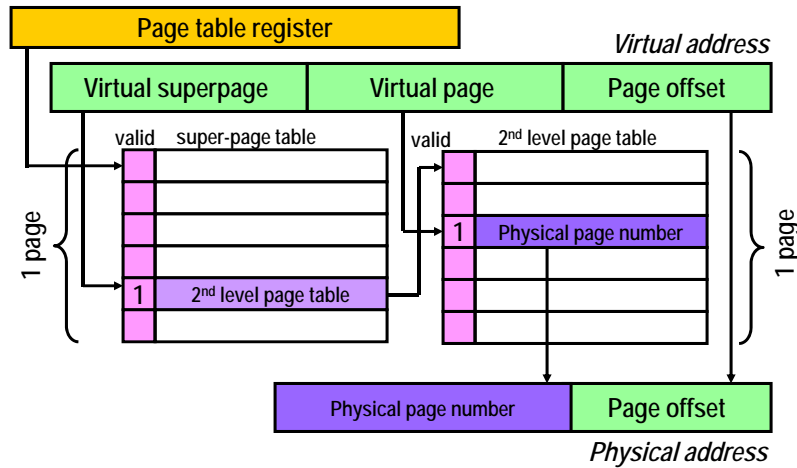


## Class problem – VM architecture

A new generation U-tanium processor has a 32-bit virtual address space and it can support up to 512 MB of byte addressable RAM. The machine has single level page tables and the page size is set to 4KB. Each entry of the page table contains an additional byte with control information (dirty, valid, etc.).

- What is the size of one page table entry in bytes? Remember that page table entries are byte aligned
- How many pages does the page table occupy?
- How many programs can run simultaneously on this machine? Each program has its own page table which is kept in memory at all times. Assume at least 128MB of RAM must be devoted to data (not page tables).
- How will increasing the page size to 32KB affect the page table? (same size, larger, smaller)

## Last lecture: Hierarchical page table



## Last lecture: Hierarchical page table

- |                          |                                                        |                      |
|--------------------------|--------------------------------------------------------|----------------------|
| <input type="checkbox"/> | How many bits in the virtual superpage field?          | <i>Example</i><br>10 |
| <input type="checkbox"/> | How many bits in the virtual page field?               | 10                   |
| <input type="checkbox"/> | How many bits in the page offset?                      | 12                   |
| <input type="checkbox"/> | How many pages in the super page table?                | 1024                 |
| <input type="checkbox"/> | How many bytes for each entry in the super page table? | 4                    |
| <input type="checkbox"/> | How many pages in the 2nd level of the page table?     | 1024                 |
| <input type="checkbox"/> | How many bytes for each VPN in a 2nd level table?      | 4                    |
| <input type="checkbox"/> | What is the total size of the page table?              | $4K+n*4K$            |

(this example is how 32bit x86 works)

## Class problem – multi-level VM

---

- ❑ Design the virtual memory of a byte addressable processor with 24-bits long addresses. No cache in the system. 256Kbytes of memory installed, and no additional memory can be added.
- ❑ Virtual memory page: 512 Bytes. Each page table entry must be an integer number of bytes, and must be the smallest size required to fit the physical page number + 1 bit to mark valid-entry
- ❑ **Design** a two-level virtual memory system. We want each second-level page table to fit exactly in one memory page, and superpage table entries are 3 bytes each (a memory address pointer to a 2L page table). Compute: Number of entries in each second-level page table; Number of virtual address bits used to index the second-level page table; Number of virtual address bits used to index the superpage table; Size of the superpage table.

## Performance of virtual memory

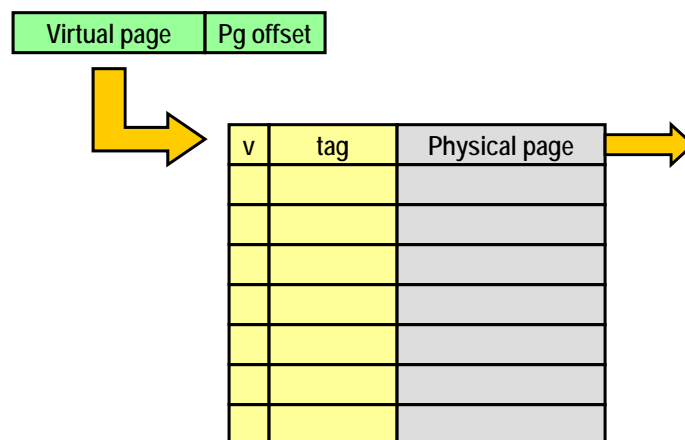
---

- ❑ To translate a virtual address into a physical address, we must first access the page table in physical memory.
- ❑ Then we access physical memory again to get (or store) the data
  - A load instruction performs at least 2 memory reads
  - A store instruction performs at least 1 read and then a write.
- ❑ Every memory access performs at least one slow access to main memory!

## Translation look-aside buffer

- ❑ We fix this performance problem by avoiding main memory in the translation from virtual to physical pages.
- ❑ We buffer the common translations in a **Translation Look-aside Buffer (TLB)**, a fast cache memory dedicated to storing a small subset of valid VtoP translations.

## TLB



## Where is the TLB lookup?

---

- ❑ We put the TLB lookup in the pipeline after the virtual address is calculated and before the memory reference is performed.
  - This may be before or during the data cache access.
  - Without a TLB hit we need to perform the translation during the memory stage of the pipeline.

## Placing caches in a VM system

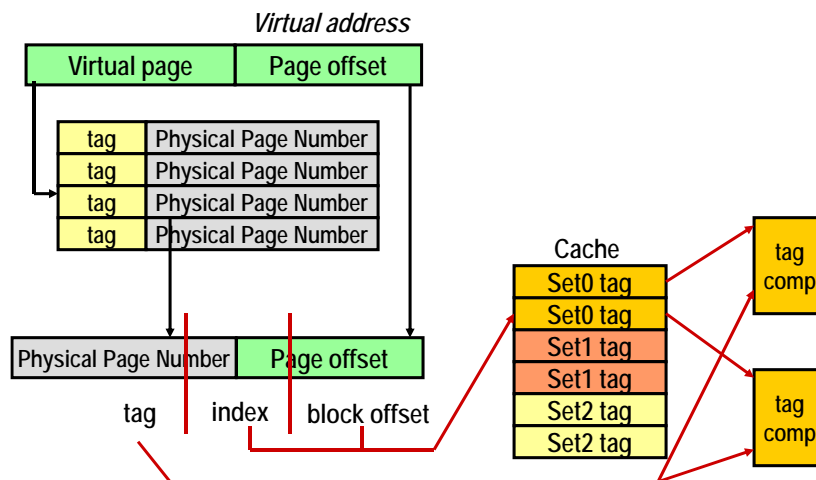
---

- ❑ VM systems give us two different addresses: virtual and physical
  
- ❑ Which address should we use to access the data cache?
  - Virtual address (before VM translation)
    - Faster access? More complex?
  - Physical address (after VM translations)
    - Delayed access?

## Physically addressed caches

- ❑ Perform TLB lookup *before* cache tag comparison.
  - Use bits from physical address to index set
  - Use bits from physical address to compare tag
  
- ❑ Slower access?
  - Tag lookup takes place *after* the TLB lookup.
  
- ❑ Simplifies some VM management
  - When switching processes, TLB must be invalidated, but cache OK to stay as is.

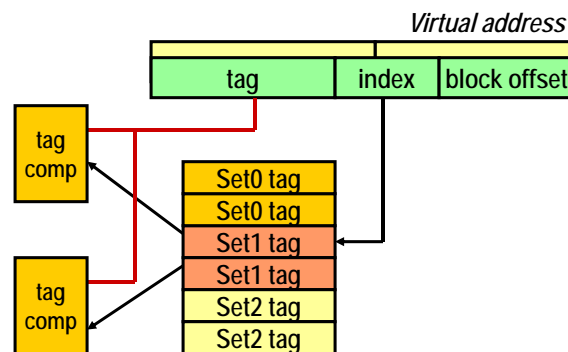
## Physical caches



## Virtually addressed caches

- ❑ Perform the TLB lookup at the same time as the cache tag compare.
  - Uses bits from the virtual address to index the cache set
  - Uses bits from the virtual address for tag match.
- ❑ Problems:
  - Aliasing: Two processes may refer to the same physical location with different virtual addresses.
  - When switching processes, TLB must be invalidated, dirty cache blocks must be written back to memory, and cache must be invalidated.

## Virtually address caches

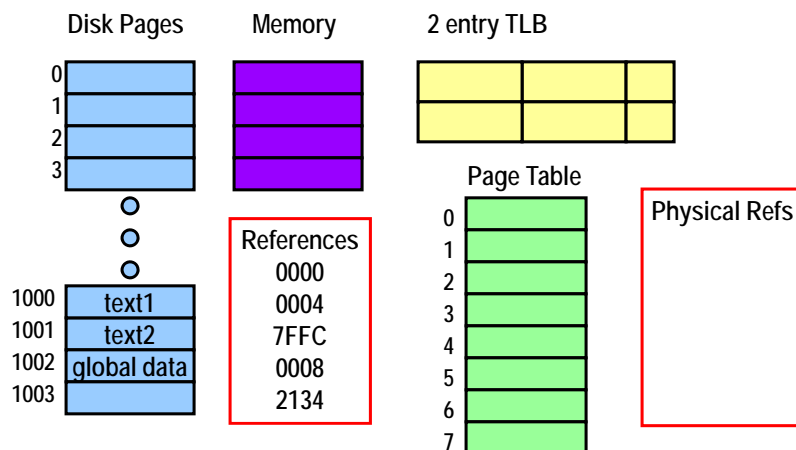


- TLB is accessed in parallel with cache lookup
- Physical address is used to access main memory in case of a cache miss

## OS support for Virtual Memory

- It must be able to modify the page table register, update page table values, etc.
  - To enable the OS to do this, **BUT** not the user program, we have different execution modes for a process:
    - one which has **executive** (or **supervisor** or **kernel** level) permissions and
    - one that has **user level** permissions

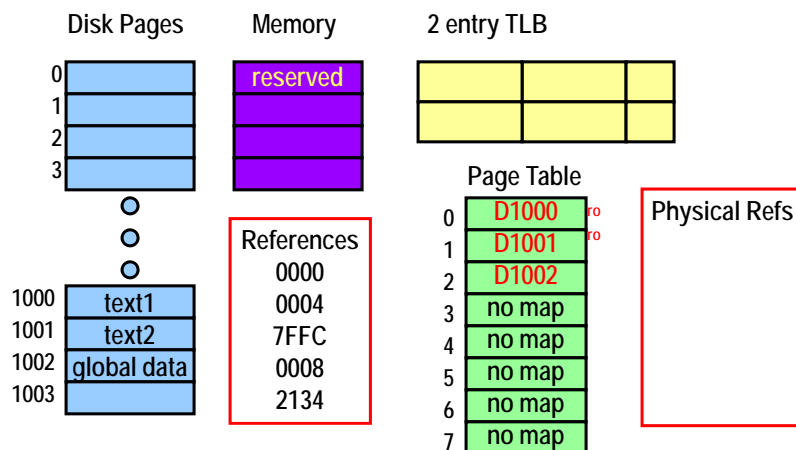
## Extended Example: Loading a program into memory



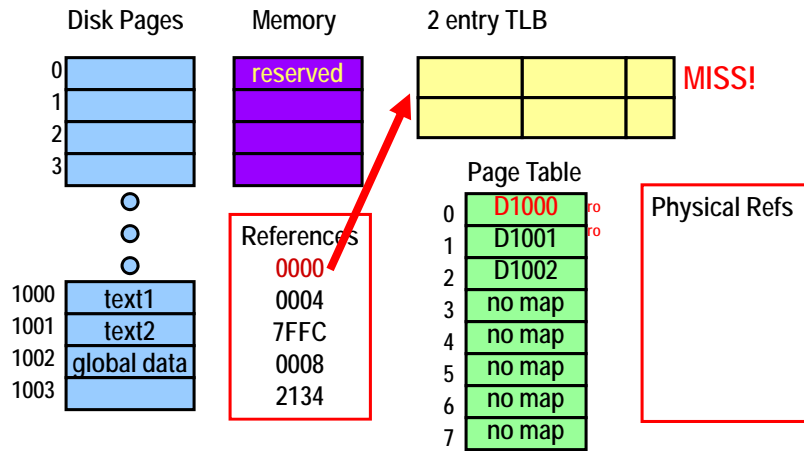
## Additional Information

- ❑ Page size = 4KB
- ❑ Page entry table size = 4B
- ❑ Page table register points to physical address 0000

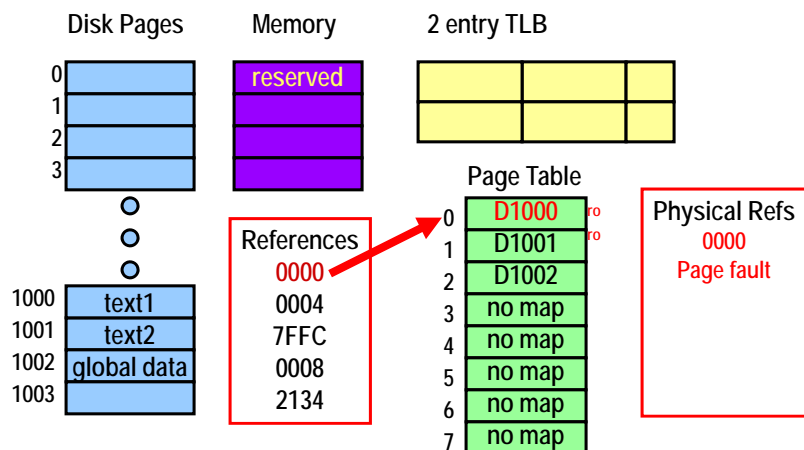
## Step 1: read executable header and initialize page table



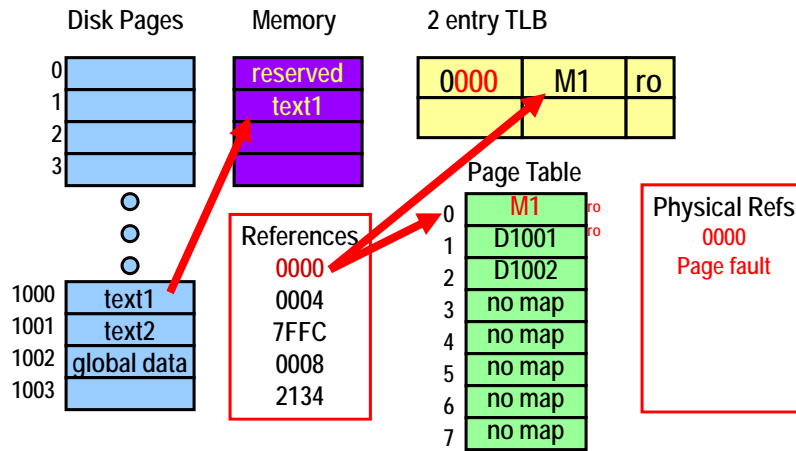
## Step 2: load PC from header and start execution



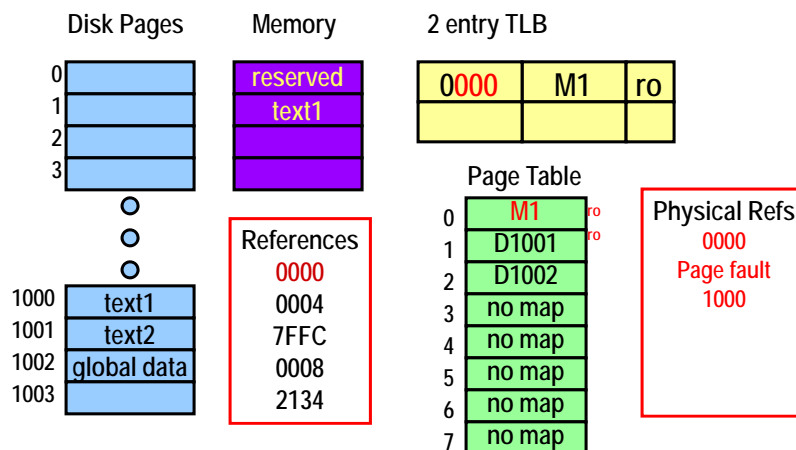
## Fetching instruction 0000



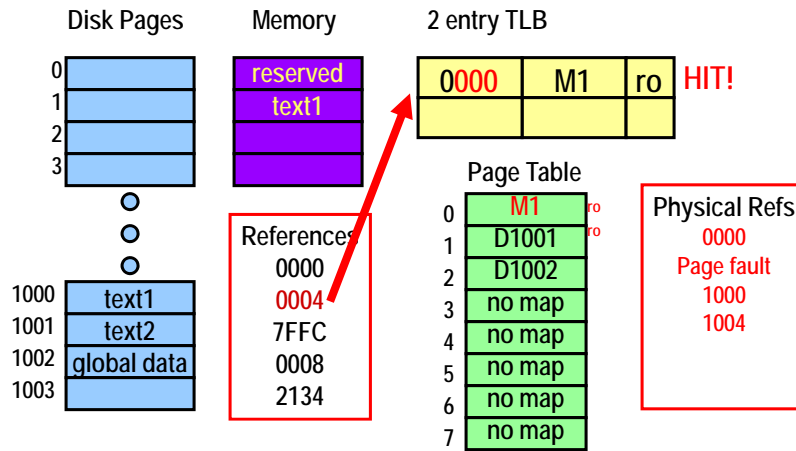
## Fetching instruction 0000



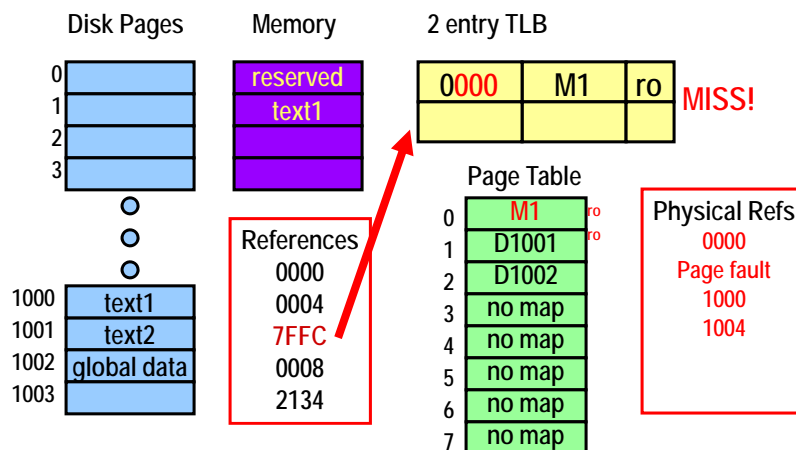
## Fetching instruction 0000



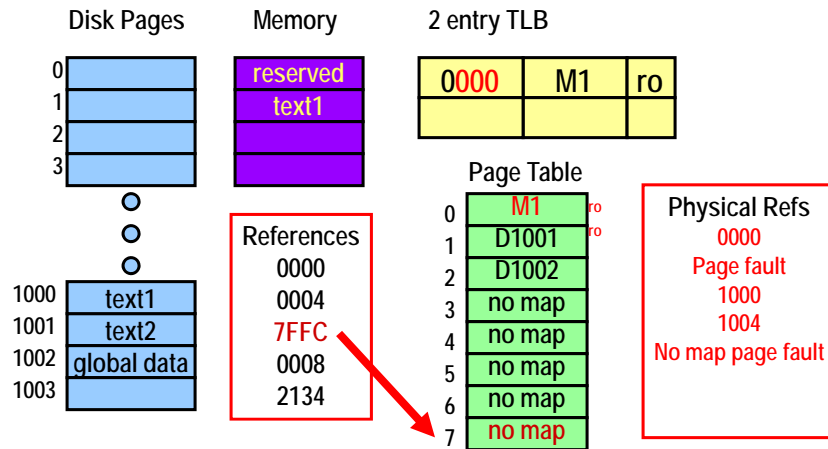
## Fetching instruction 0004



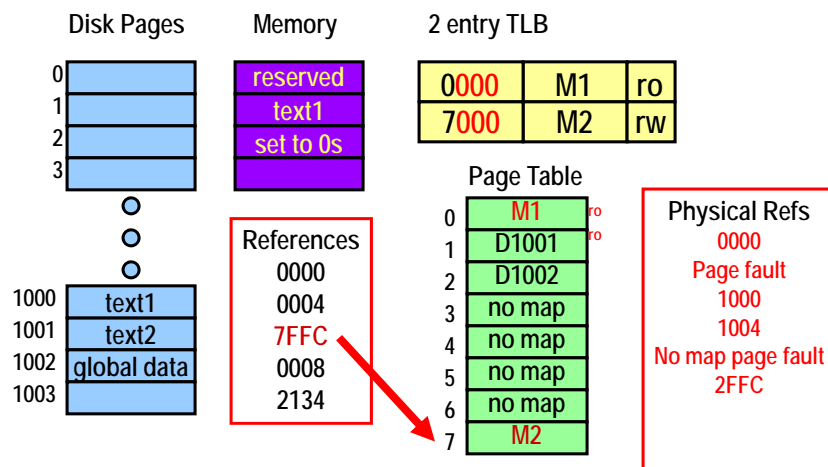
## Reference 7FFC



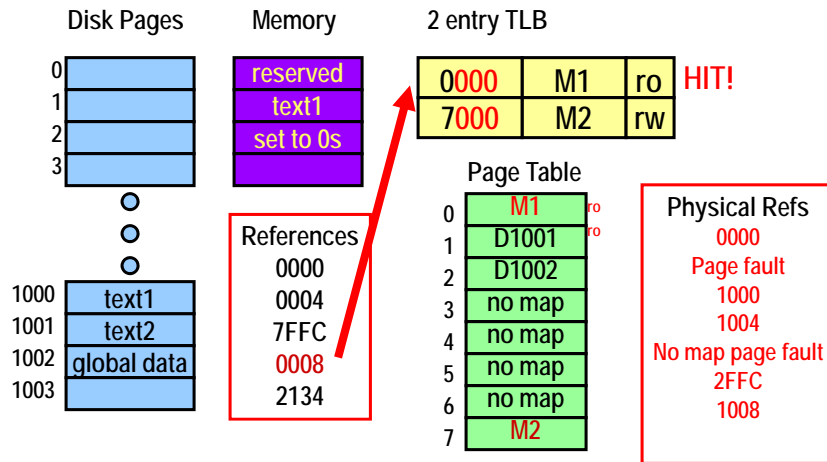
## Reference 7FFC



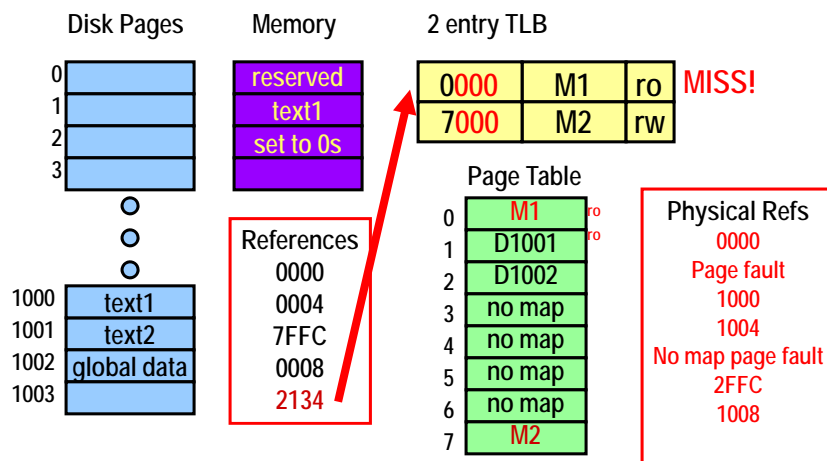
## Reference 7FFC



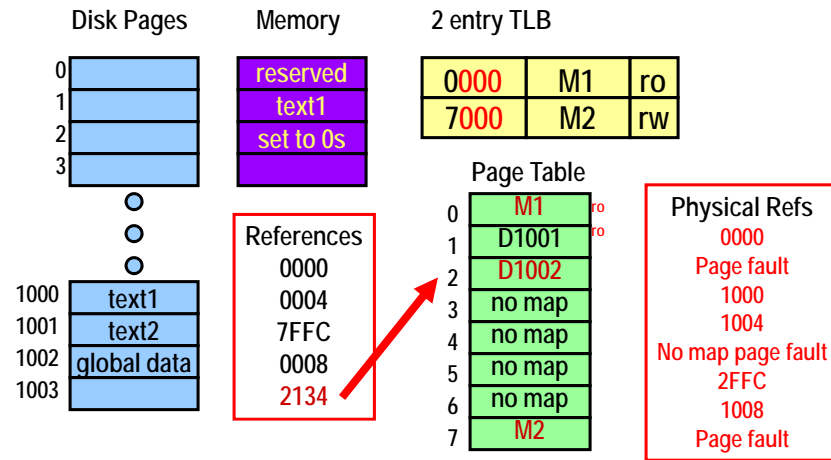
## Fetching instruction 0008



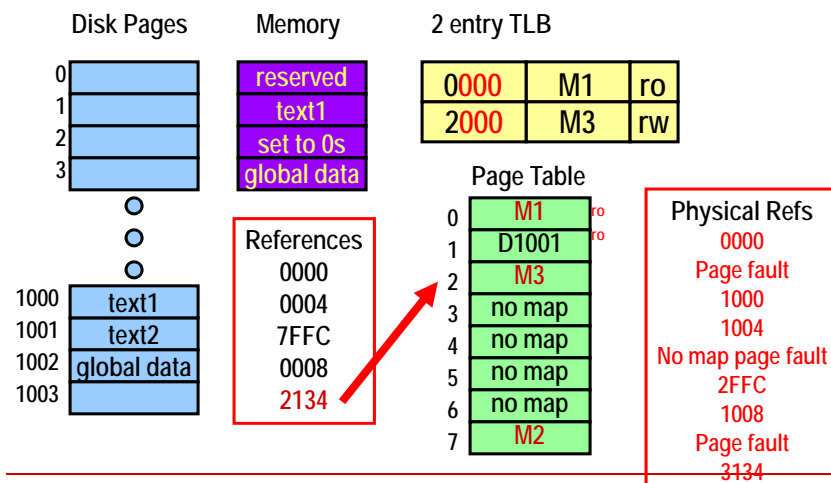
## Reference 2134



## Reference 2134



## Reference 2134



## Multiple processes

---

- ❑ *Virtual cache* support for multiple processes:
  - Flush the cache between each **context switch**.
  - Use **processID** (a unique number for each processes given by the operating system) as part of the tag

## Class problem – VM performance

---

- ❑ Consider a system with unified data and instruction cache. The virtual memory system is a one-level page table system.
  - TLB hit rate: 99 %, TLB access time is 1 cycle
  - Cache hit rate: 95 %, cache access time is 1 cycle
  - 'Page not loaded in memory' rate: 0.1 % (that's 1 in one thousand)
  - The TLB access and cache access are sequential.
  - If a data is swapped out to hard drive, the TLB will always miss. Also TLB hits will not cause a page fault.
  - The page table is uncacheable and always available in main memory.
  - Accesses to main memory require 30 cycles
  - Accesses to the hard drive require 100,000 cycles.
  - Upon retrieval from cache, main memory or hard drive, the data is sent immediately to the CPU, while other updates occur in parallel.

Compute the average latency of a memory access for this machine.