21. Putting it all together: Virtual memory and Caches

EECS 370 – Introduction to Computer Organization - Winter 2016

Profs. Valeria Bertacco & Reetu Das

EECS Department
University of Michigan in Ann Arbor, USA

Announcements

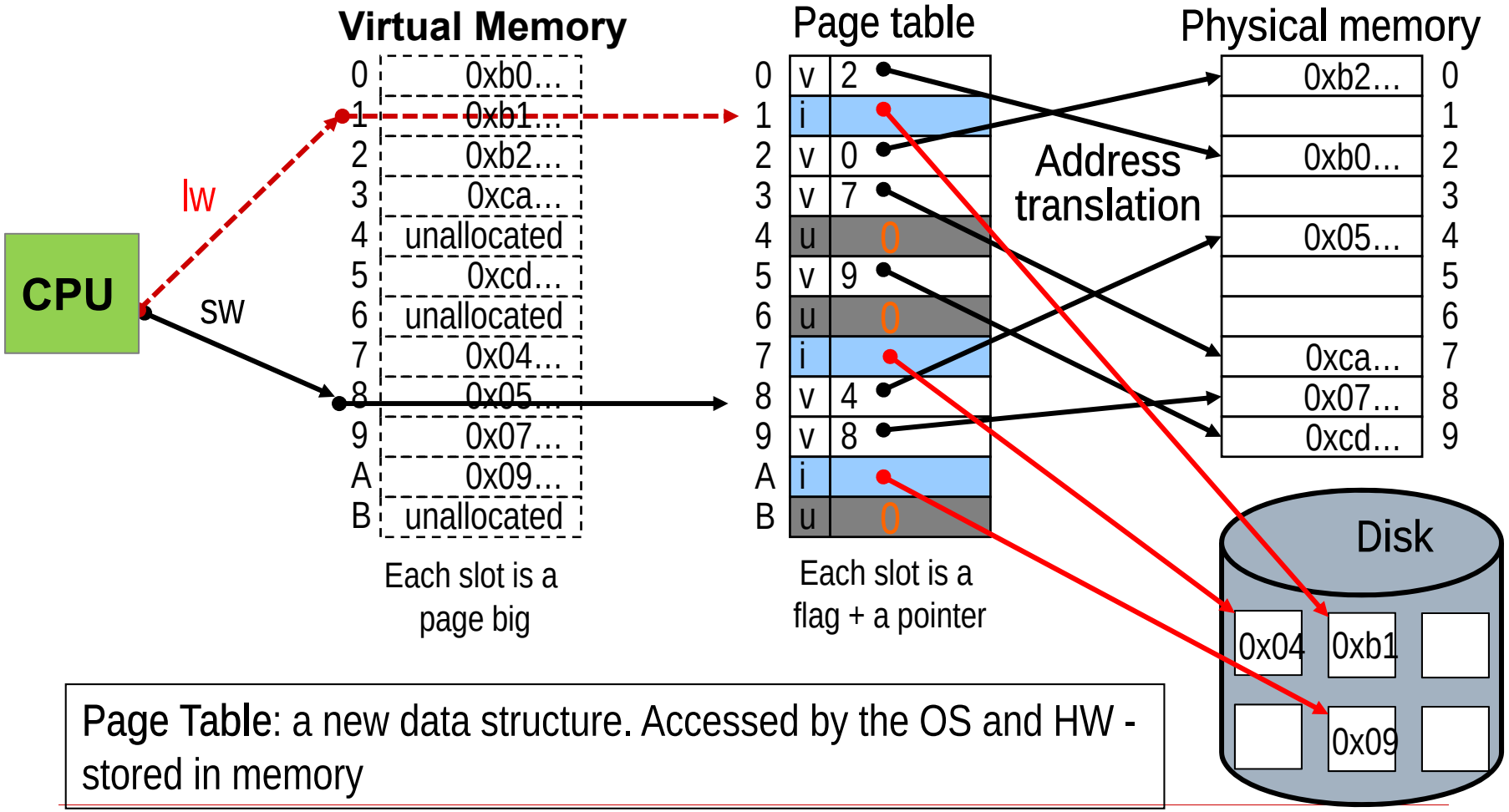
- Project 4 – due today
- Homework 7 – due Tuesday

- FINAL EXAM preview:
  - Cumulative
  - Cheat sheet – 3 sides (2 sheets)
  - 120 minutes long (longer than midterms!)
  - Make sure to not get sick

- Midterm2 grades will be republished this afternoon

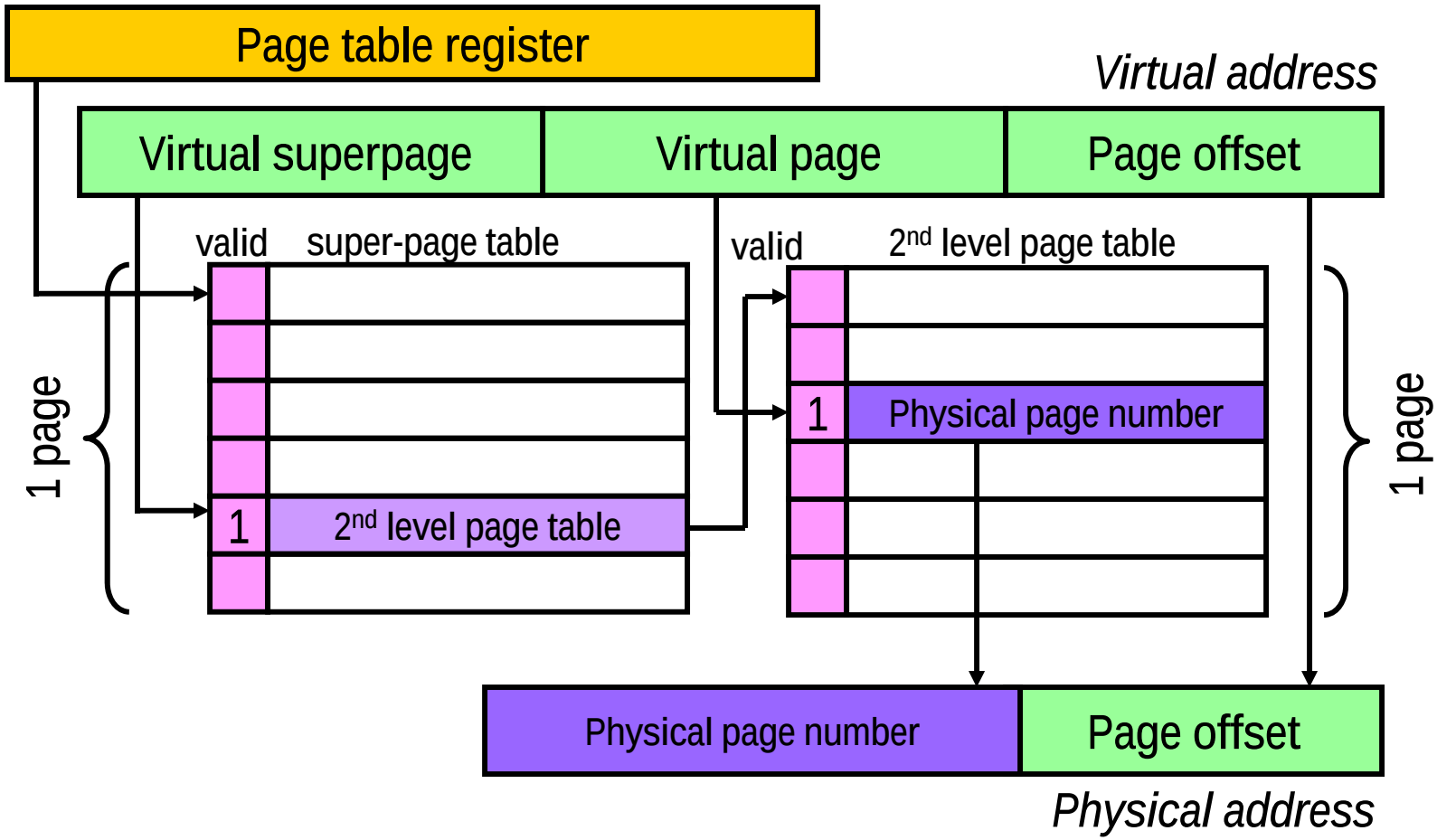
REVIEW: key pieces in the life of VM

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



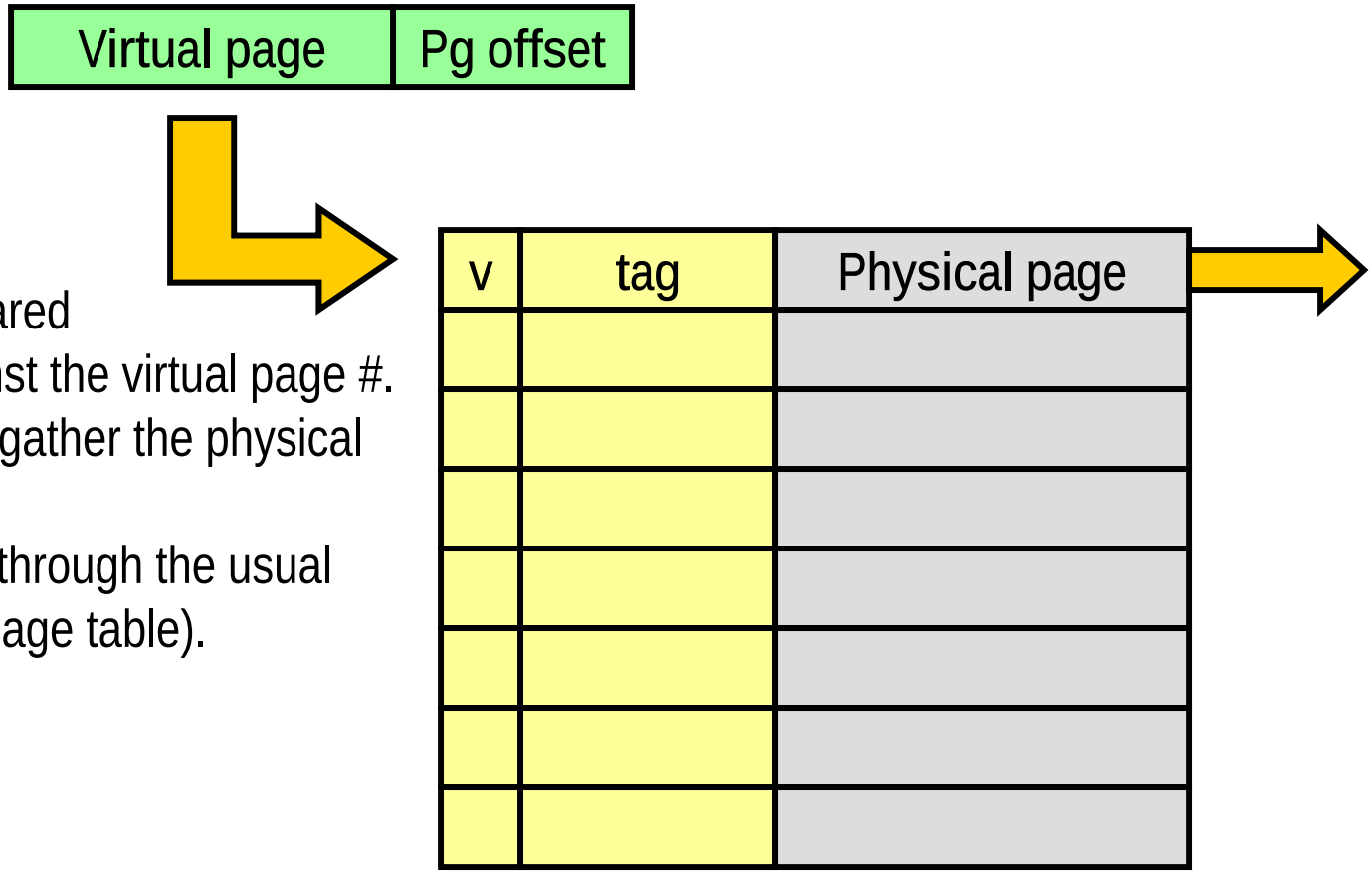
Page Table: a new data structure. Accessed by the OS and HW - stored in memory

REVIEW: Hierarchical page tables



REVIEW: Translation Look-aside Buffer

The TLB is a CAM = content-addressable memory



All tags are compared concurrently against the virtual page #.
If anyone hits, we gather the physical page #.
Otherwise, we go through the usual process (consult page table).

Class Problem 1 – 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)

Caches in systems with VM

- VM systems give us two different addresses: virtual and physical.

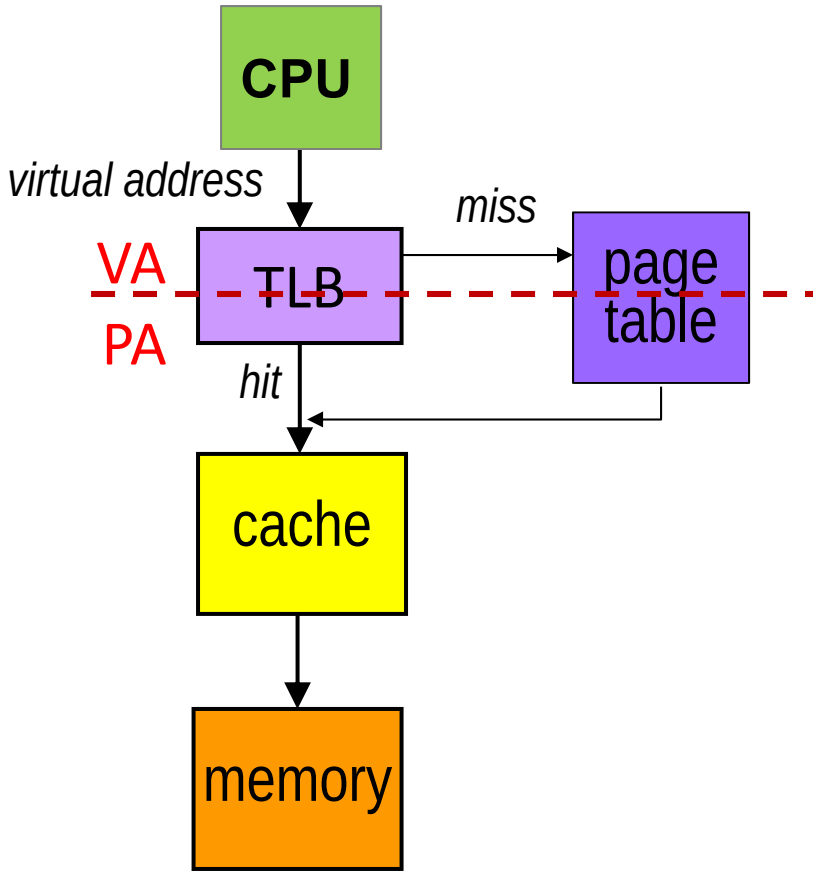
- Which address should we use to access the data cache?
  - Physical address (after VM translations).
    - Delayed access.
  - Virtual address (before VM translation).
    - Faster access.
    - More complex.

Cache & VM Organization: option 1

Physically-addressed $


Slower

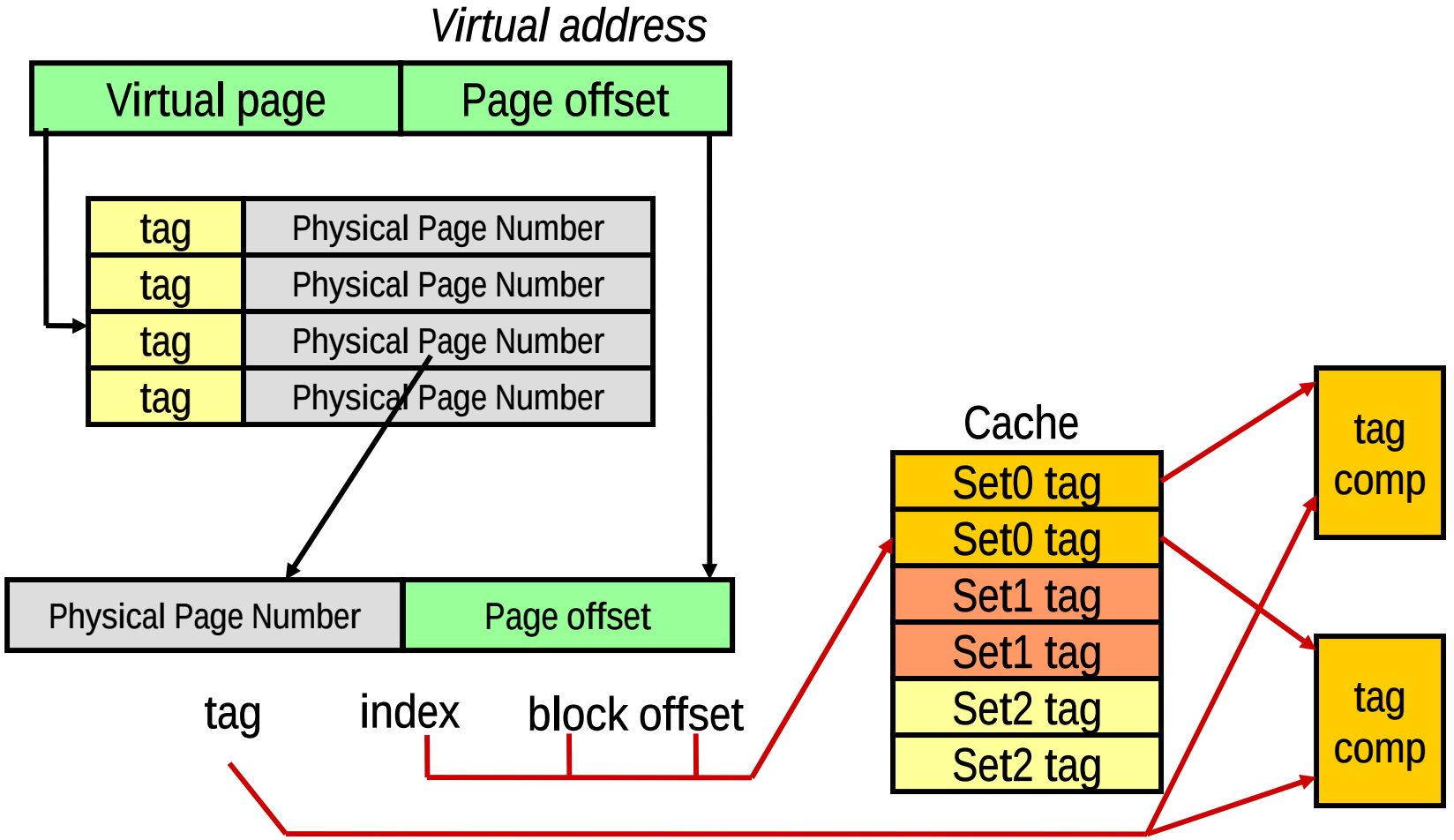

Low complexity



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.
  - Implications? Might result in fewer cache misses if context switches very common (but they generally are not).

Physically addressed caches

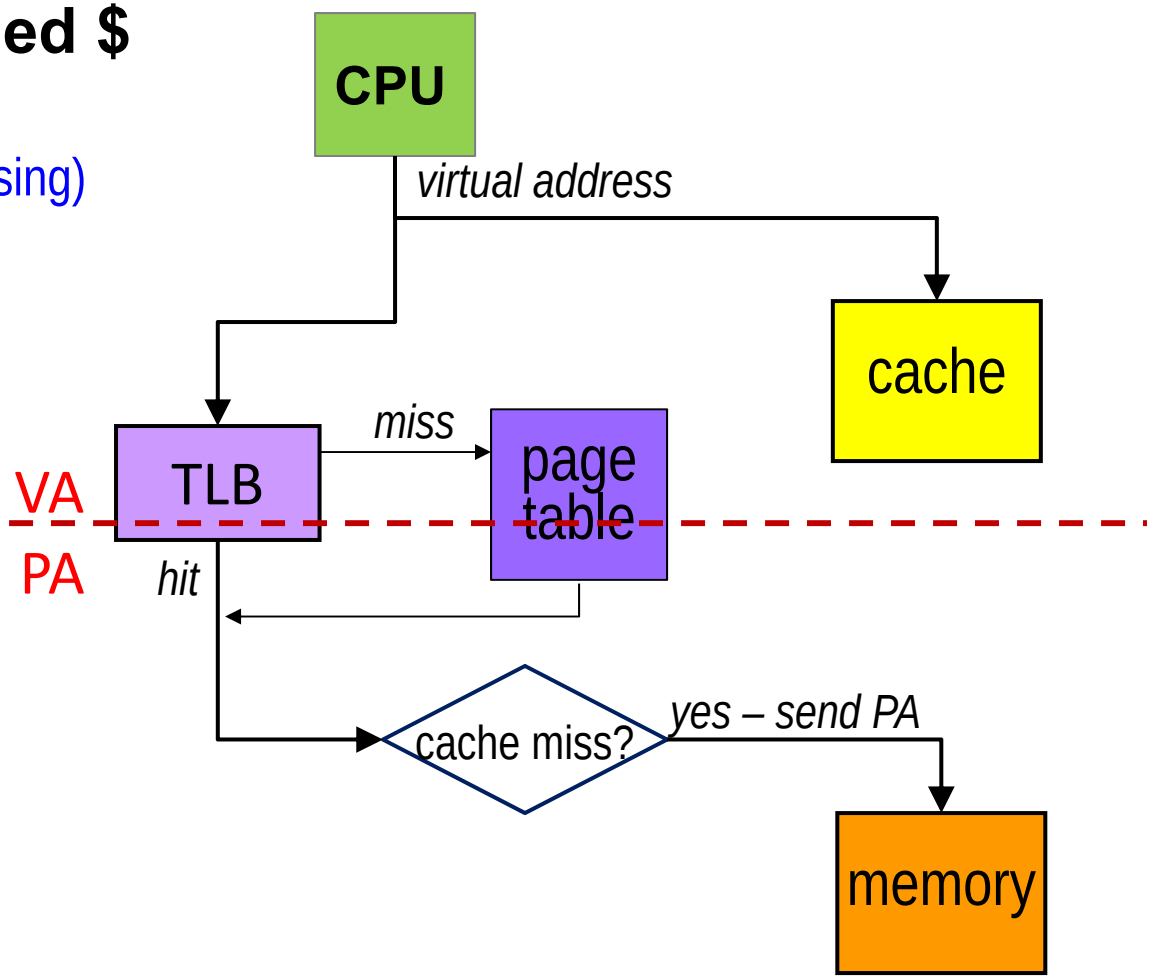


Cache & VM Organization: option 2

Virtually-addressed $

High complexity (aliasing)

Faster



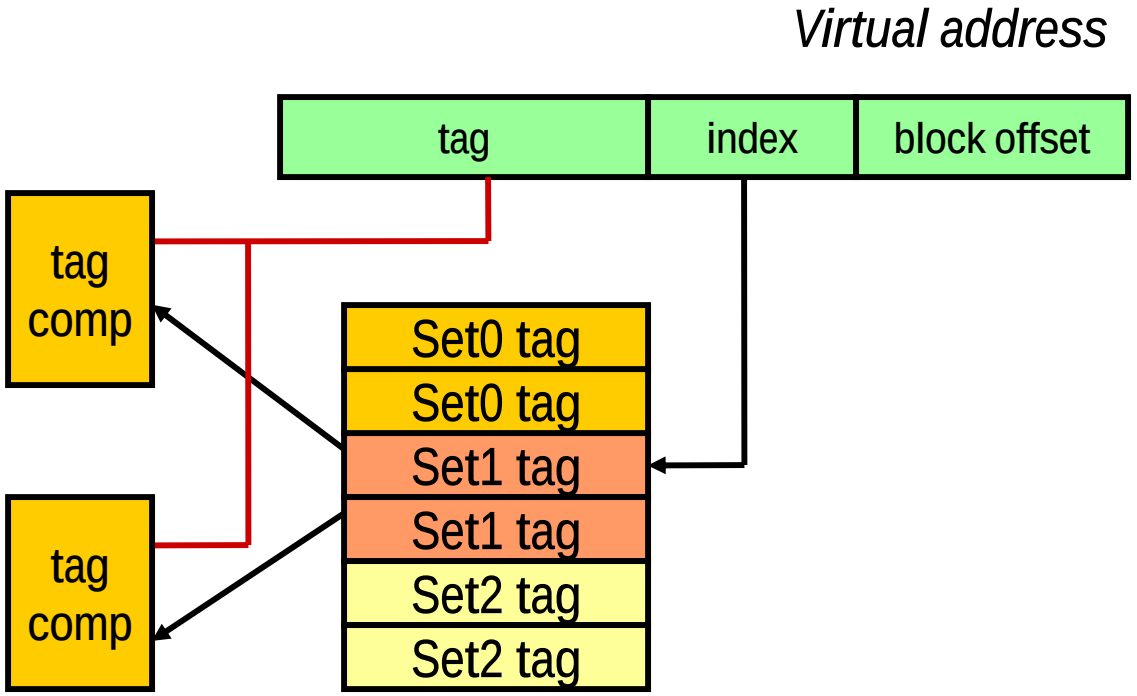
Virtually addressed caches

- 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.

Multitasking with VM

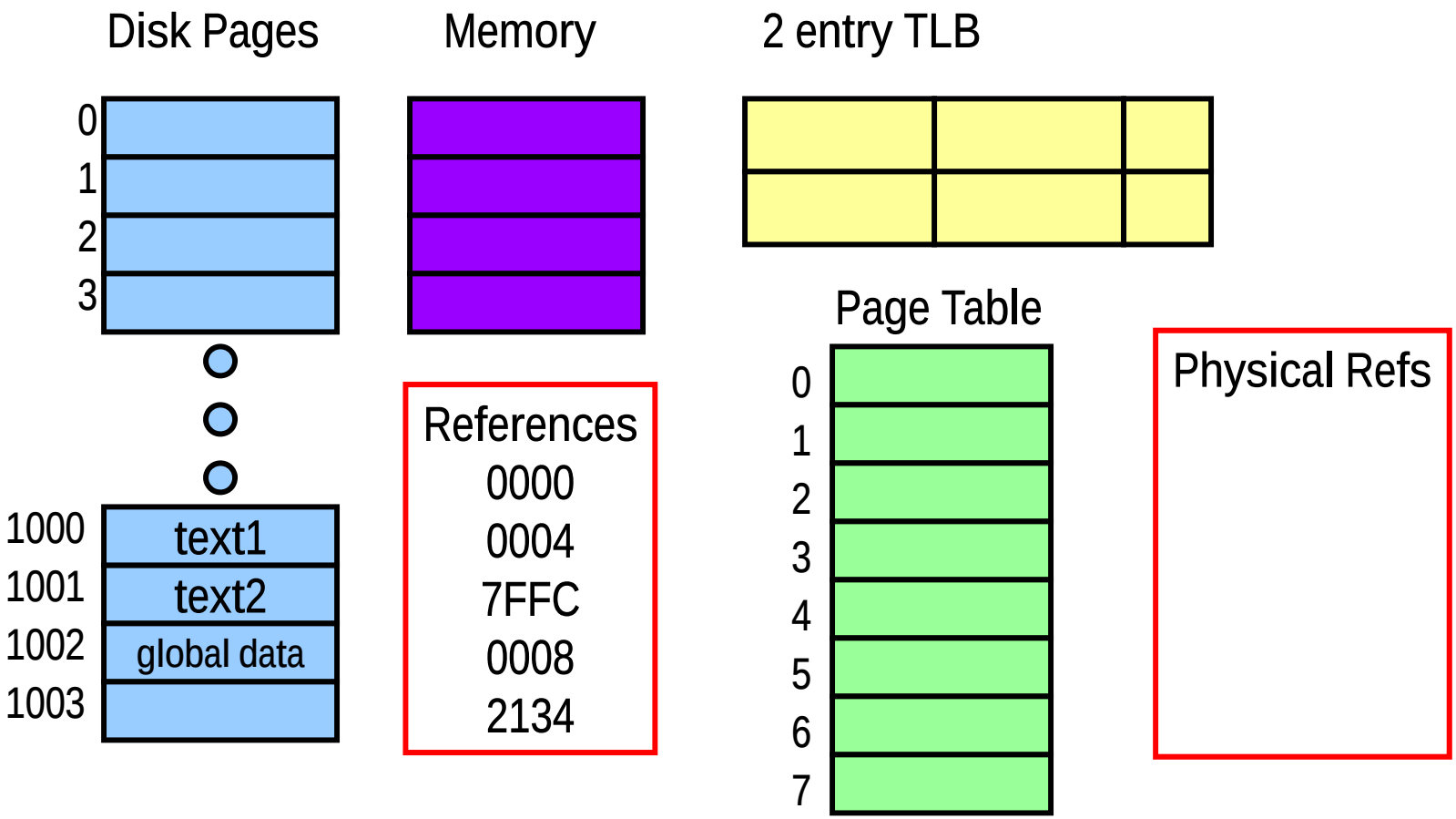
- If virtually-addressed caches:
  - Flush the cache between each context switch.
    OR
  - Use processID (a unique number for each processes given by the operating system) as part of the tag

Virtually addressed caches



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

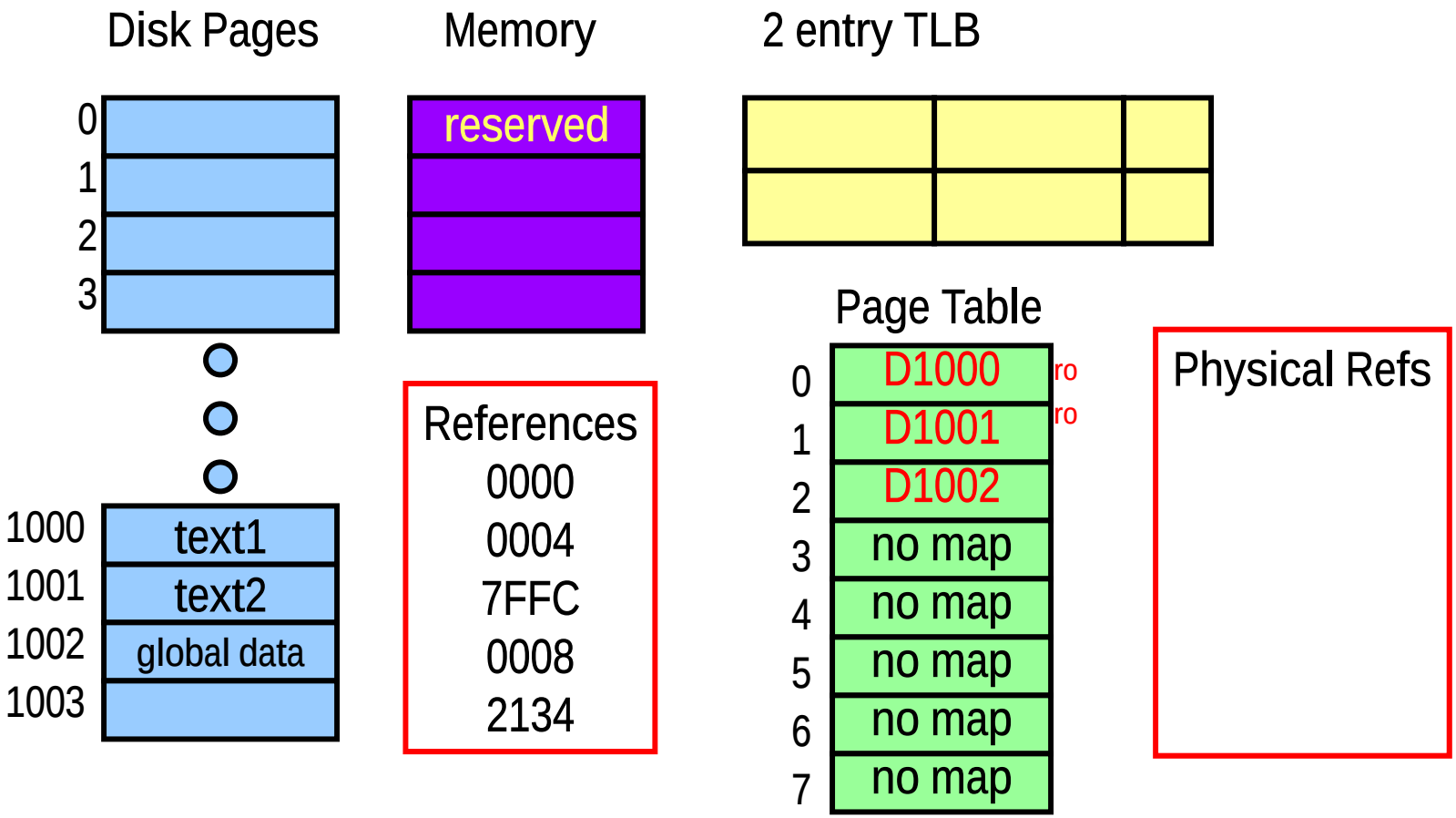
Loading a program into memory



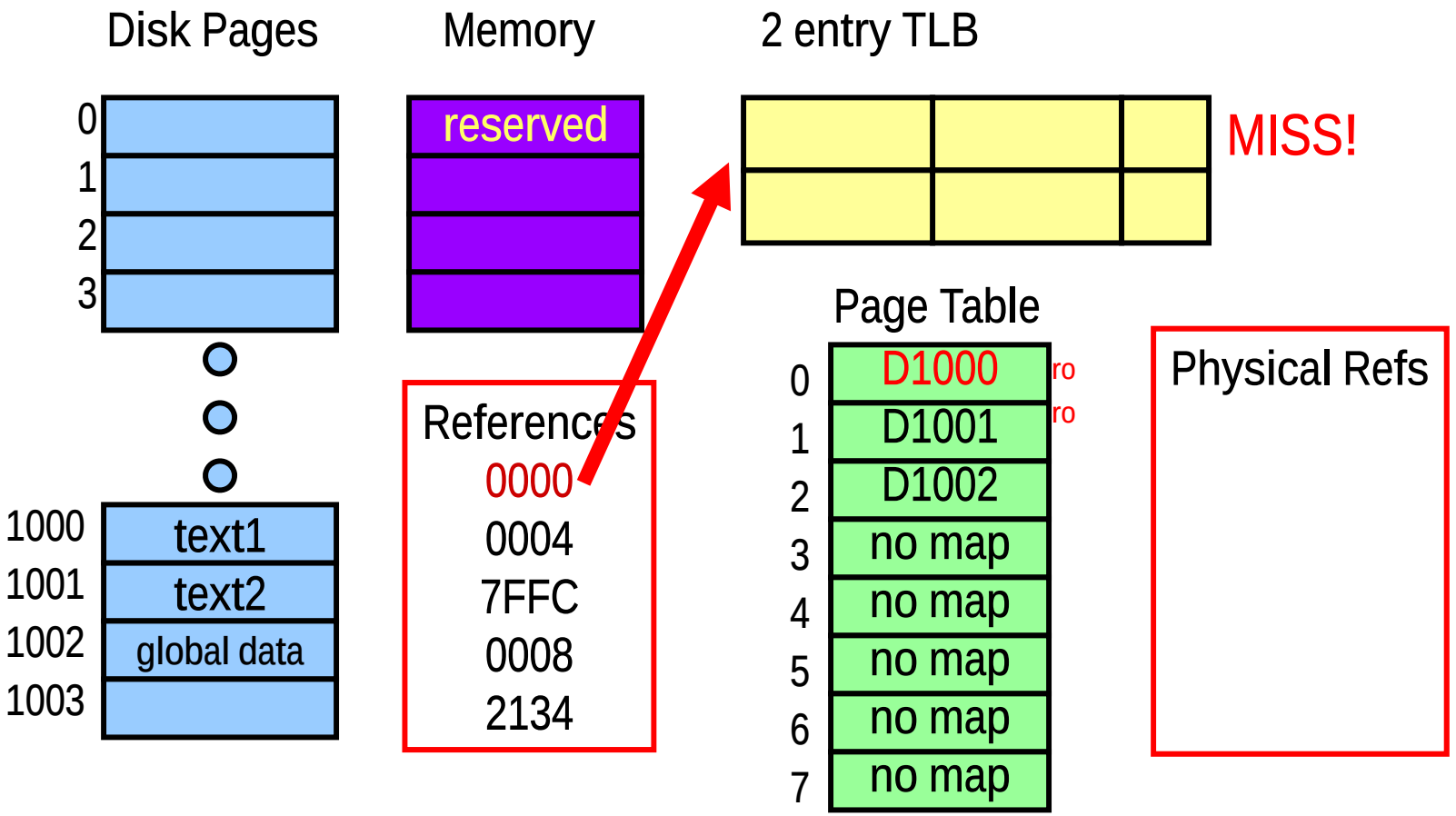
Additional information

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

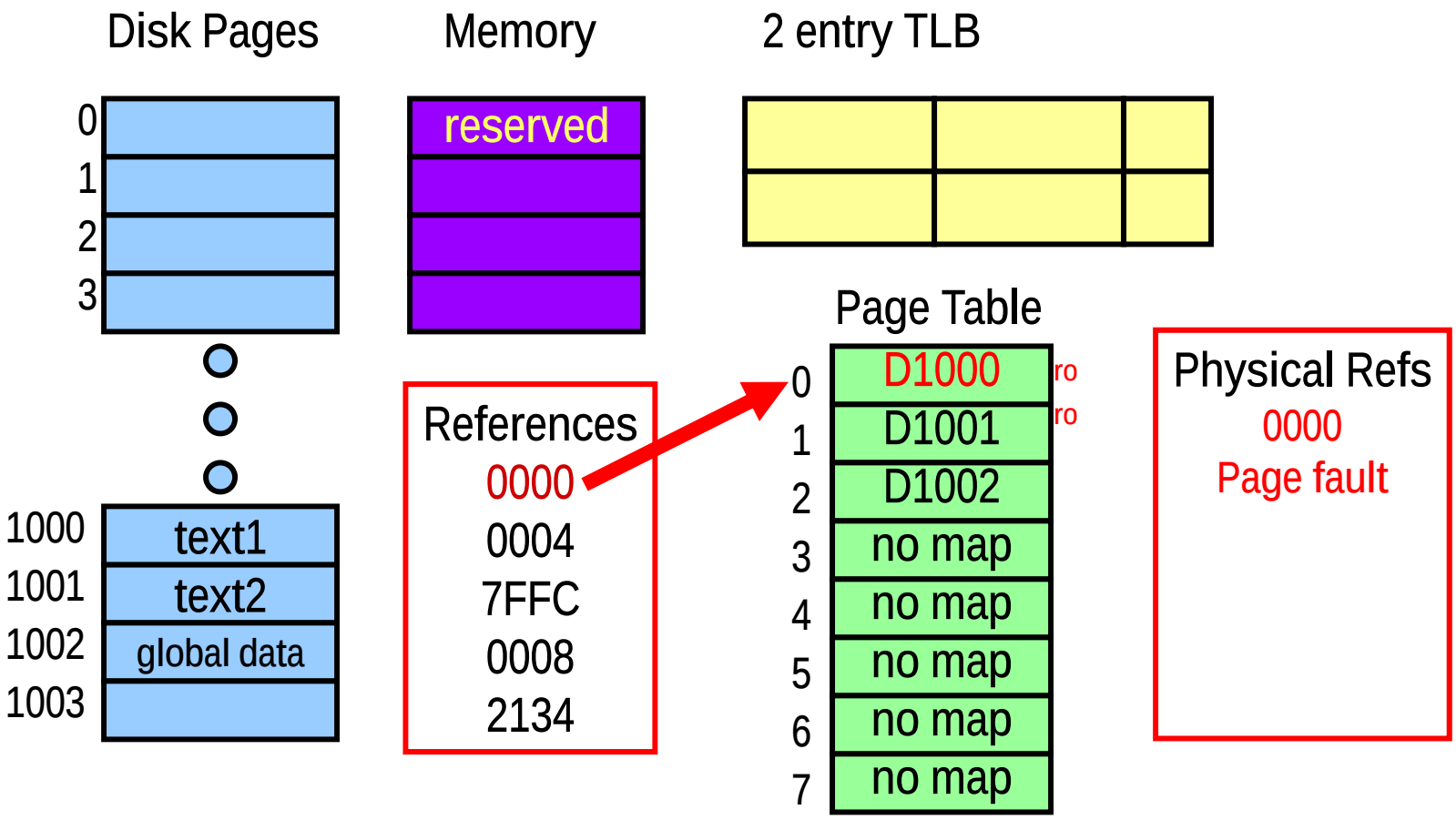
Step 1: read executable header and initialize page table



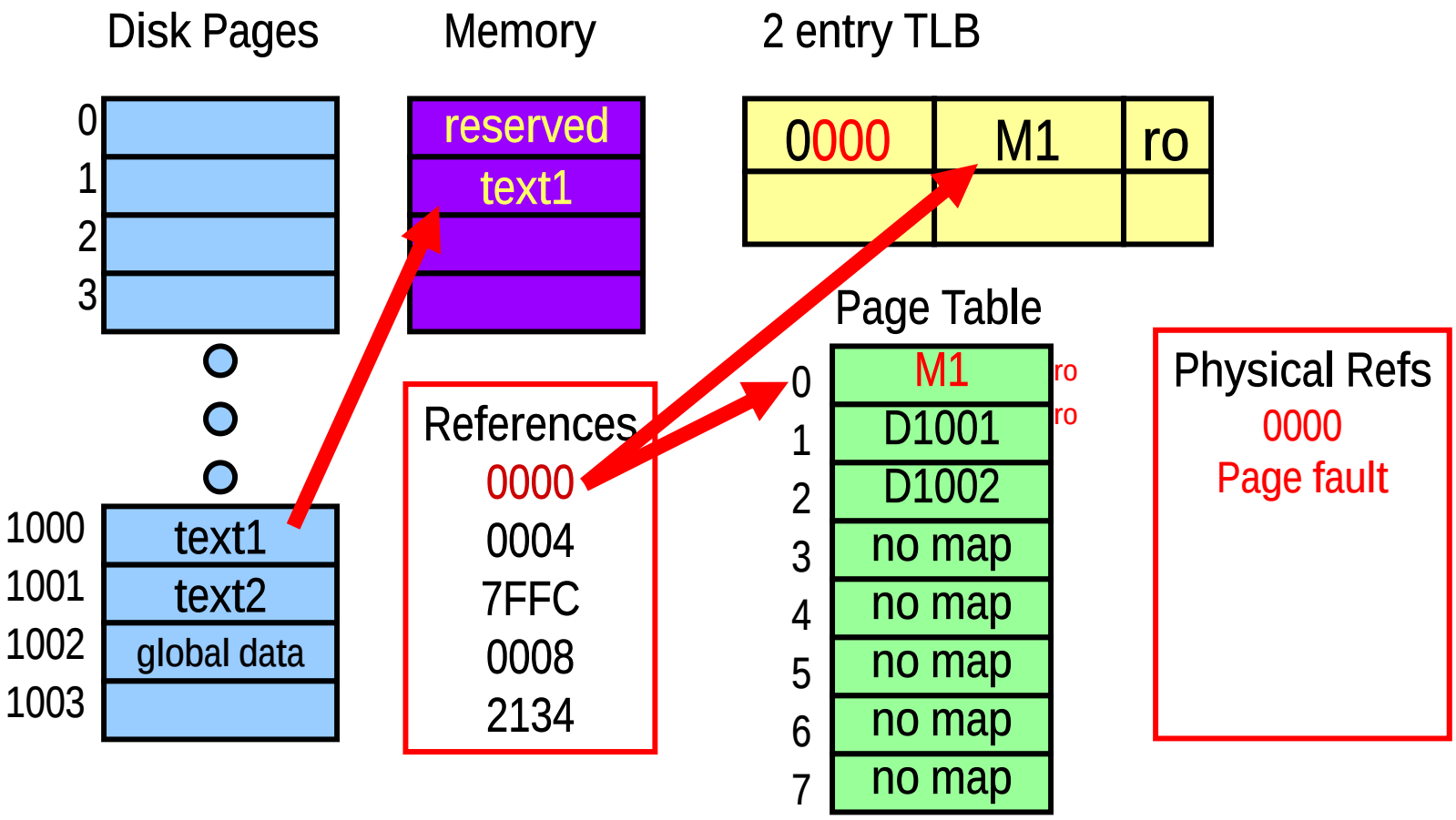
Step 2: load PC from header and start execution



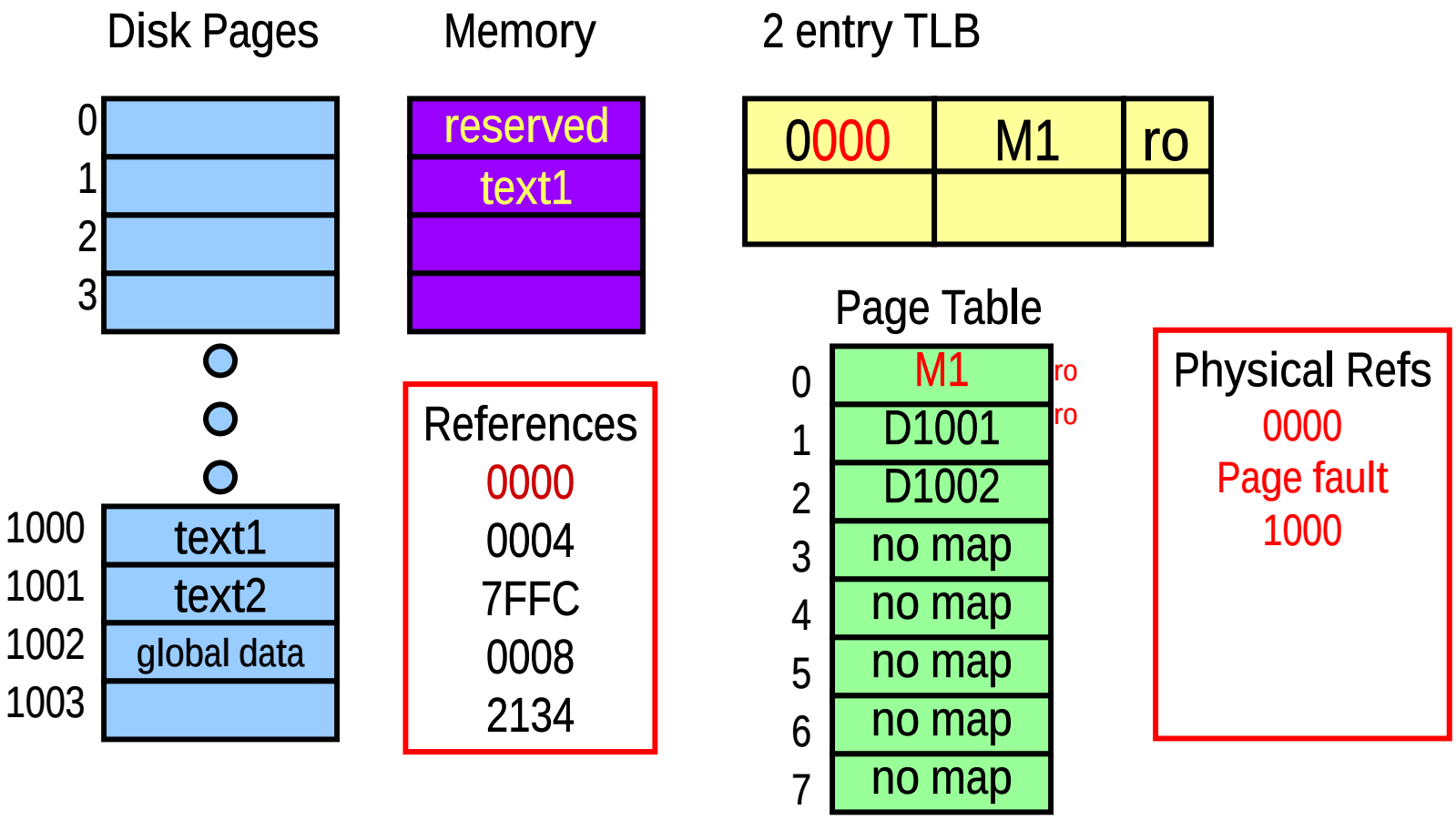
Fetching instruction 0000



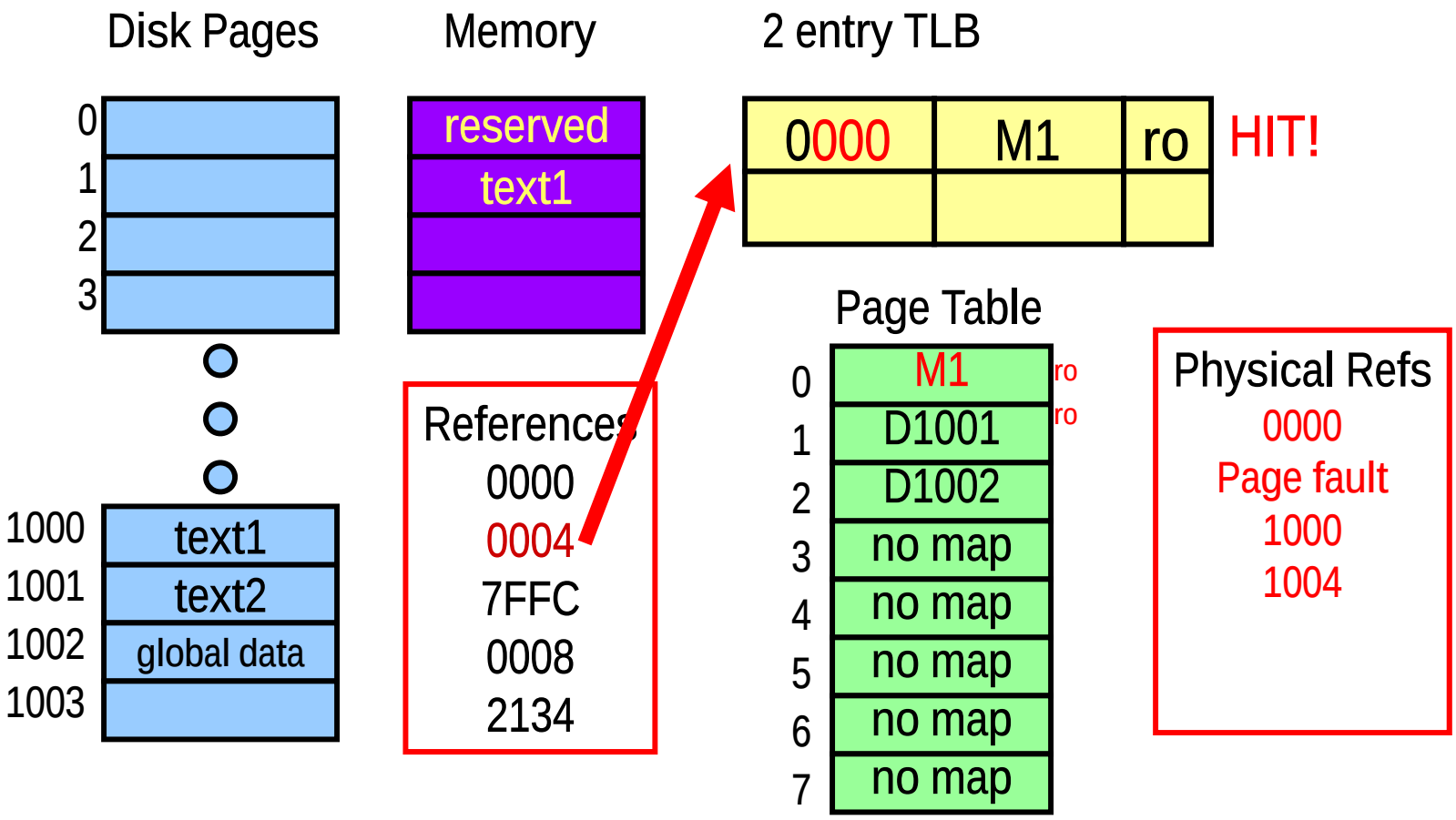
Fetching instruction 0000



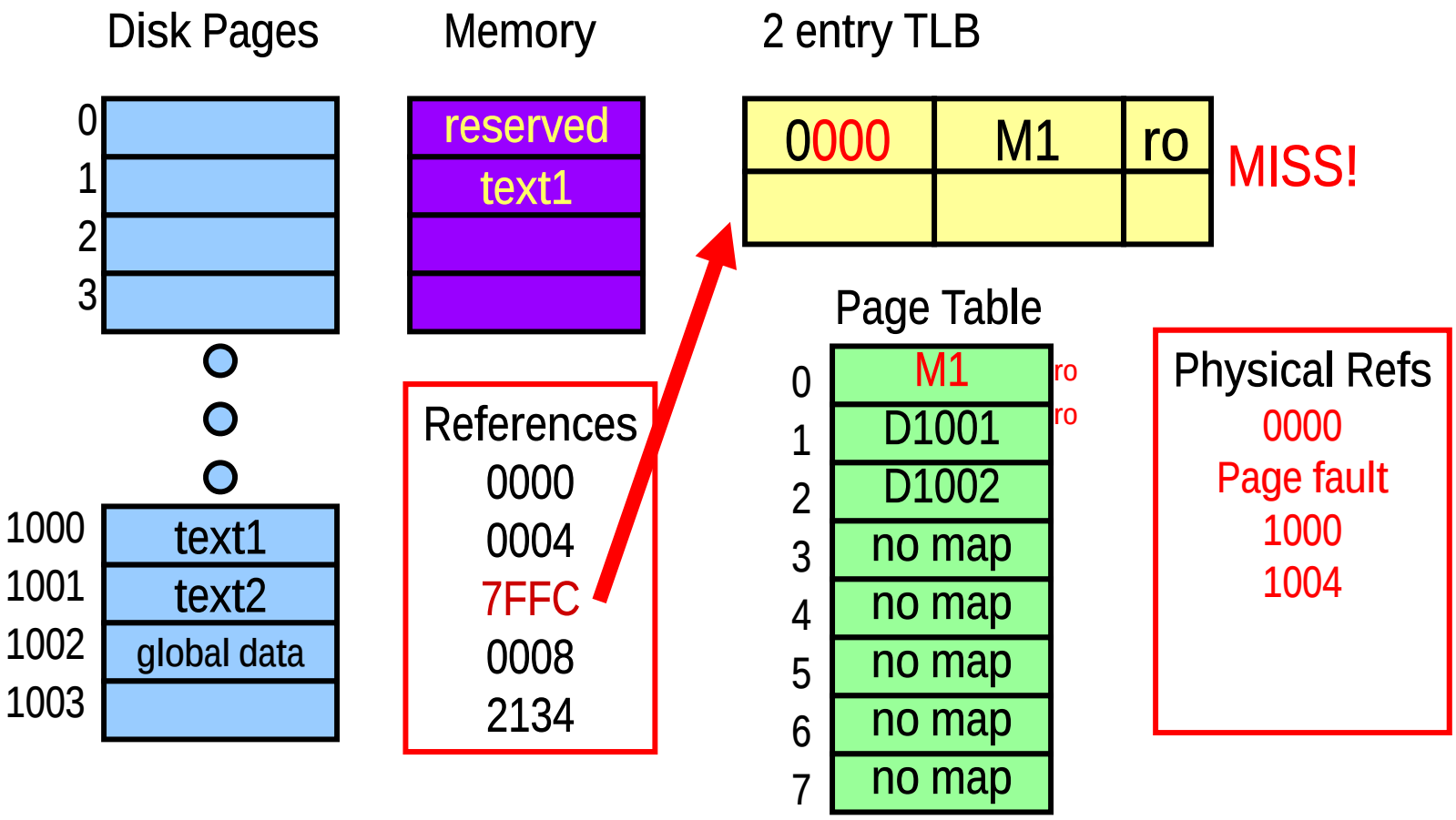
Fetching instruction 0000



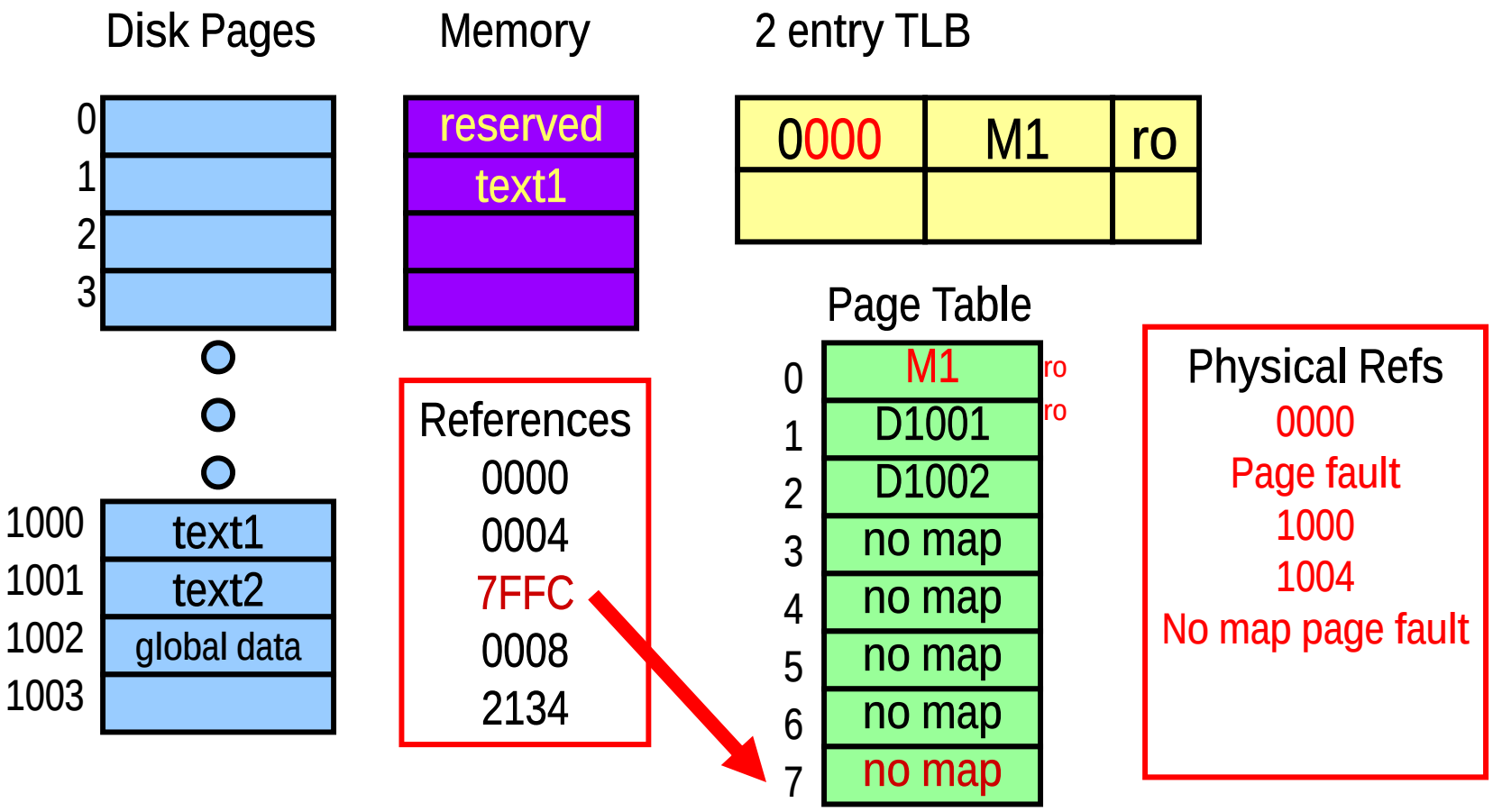
Fetching instruction 0004



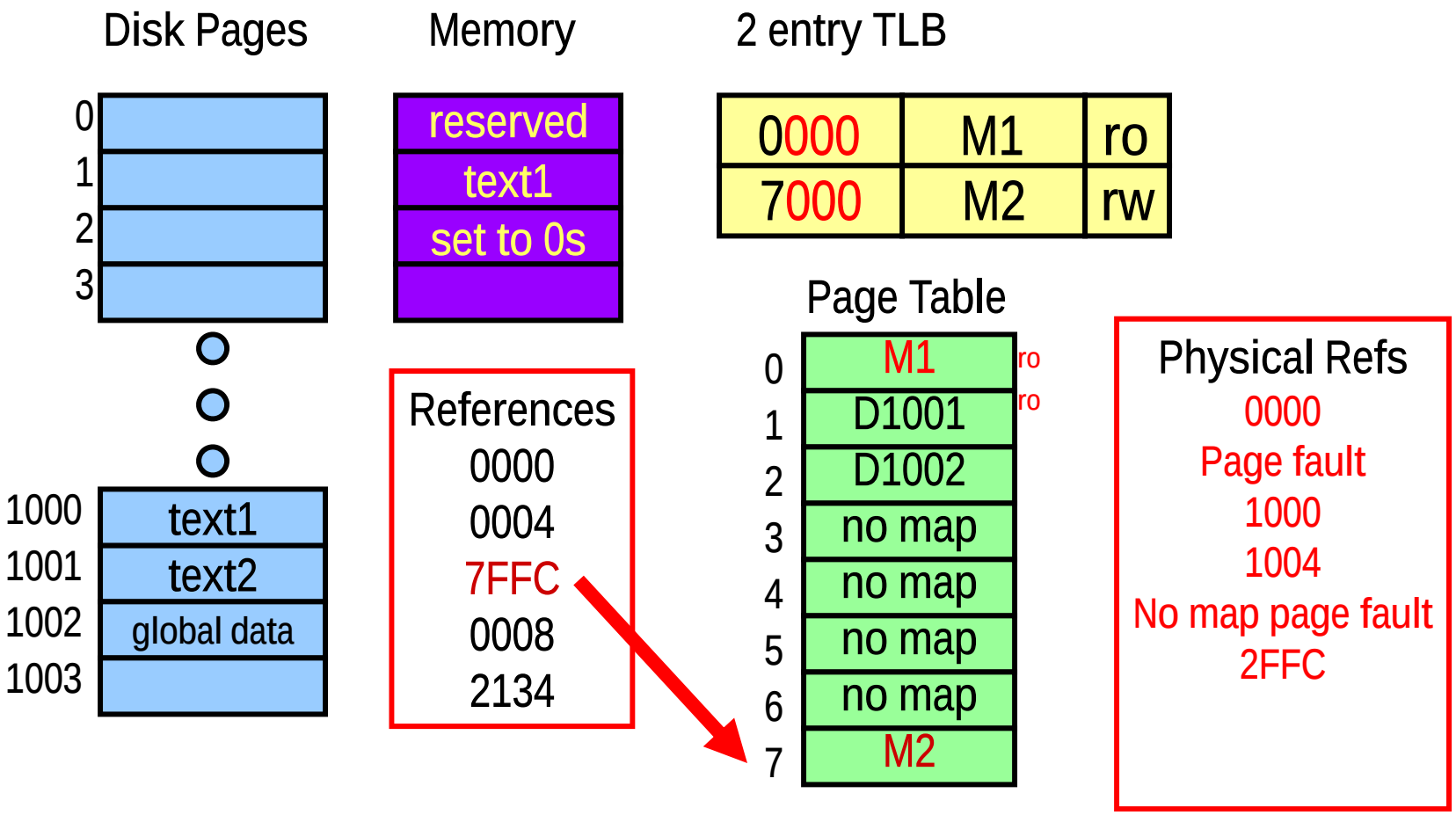
Reference 7FFC



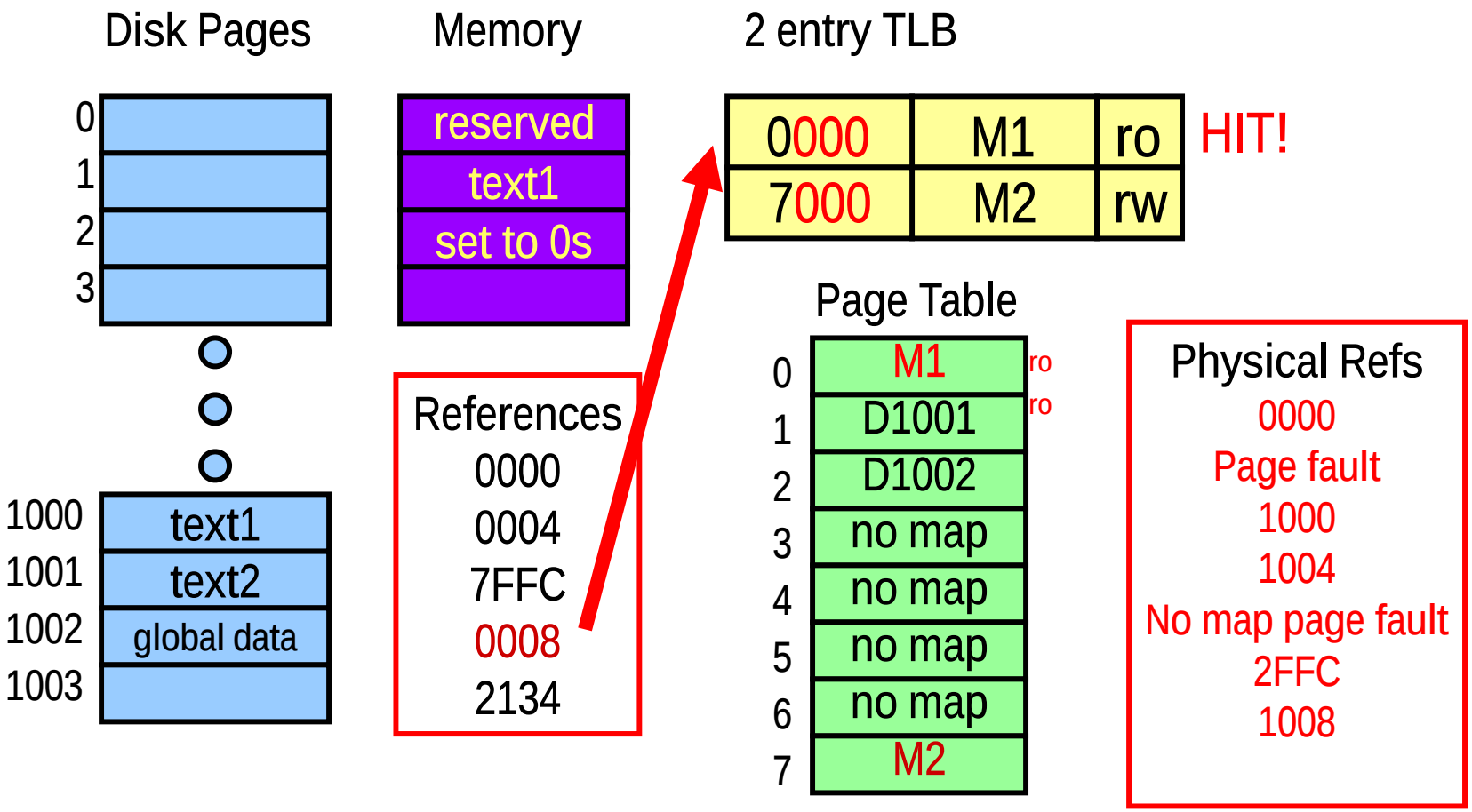
Reference 7FFC



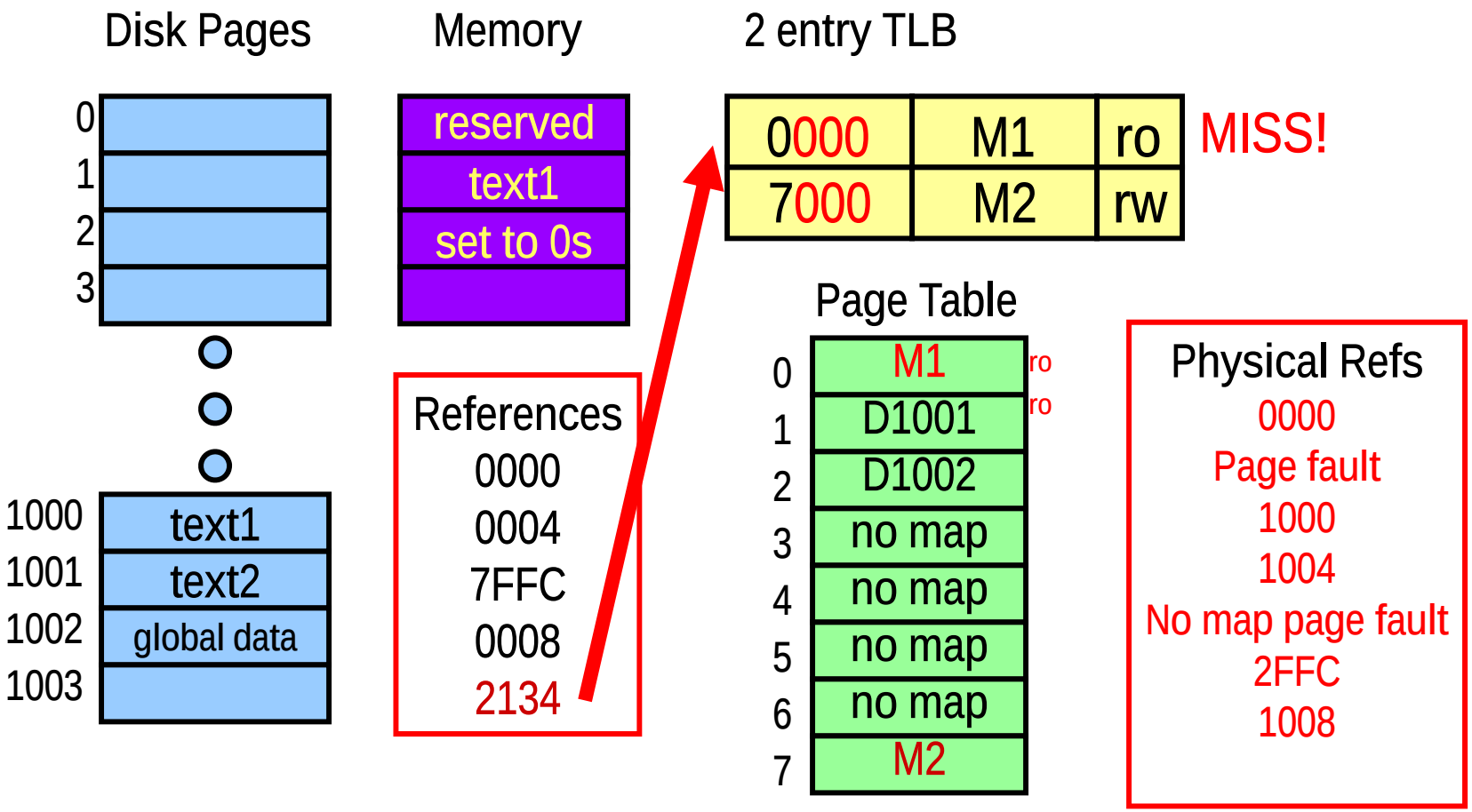
Reference 7FFC



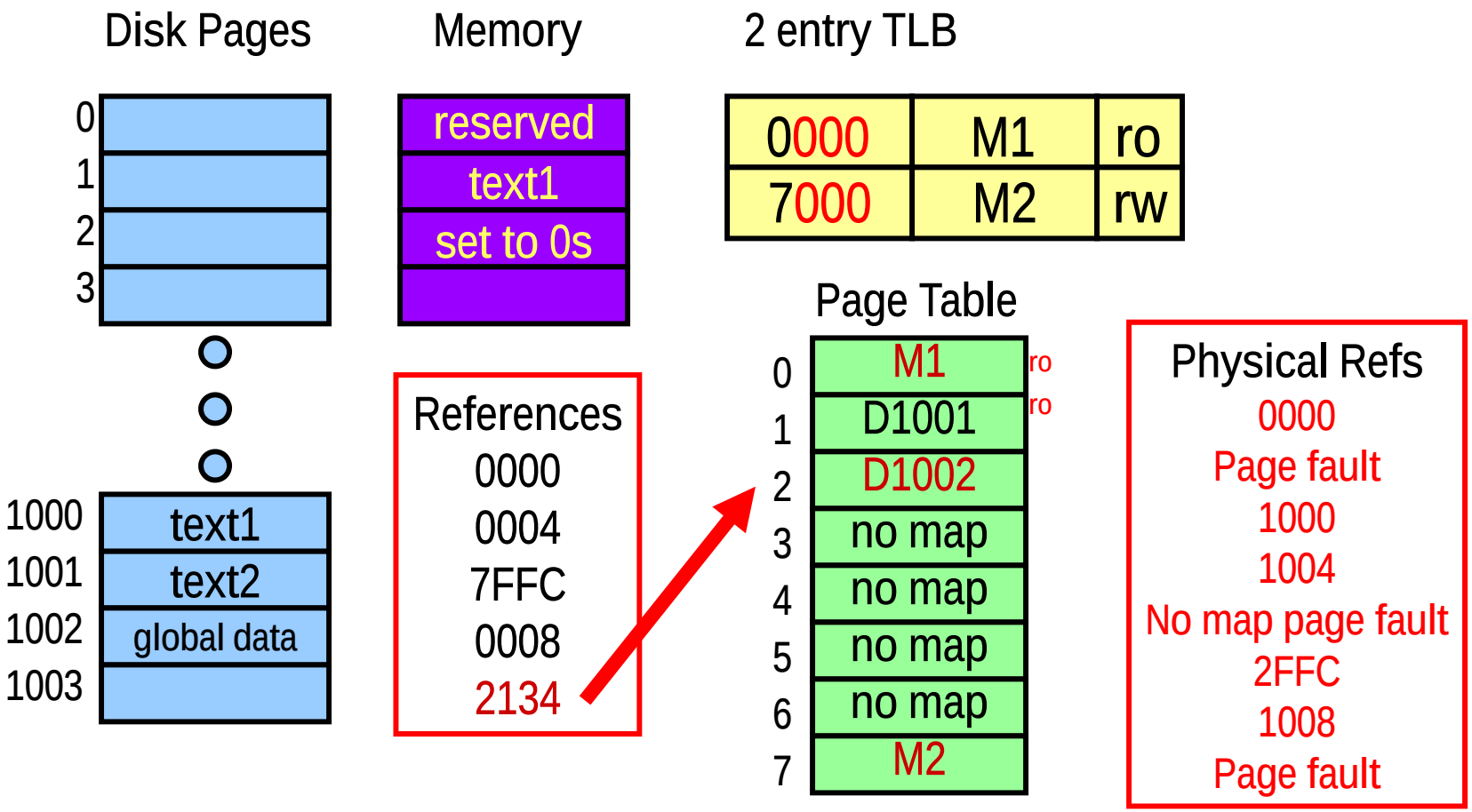
Fetching instruction 0008



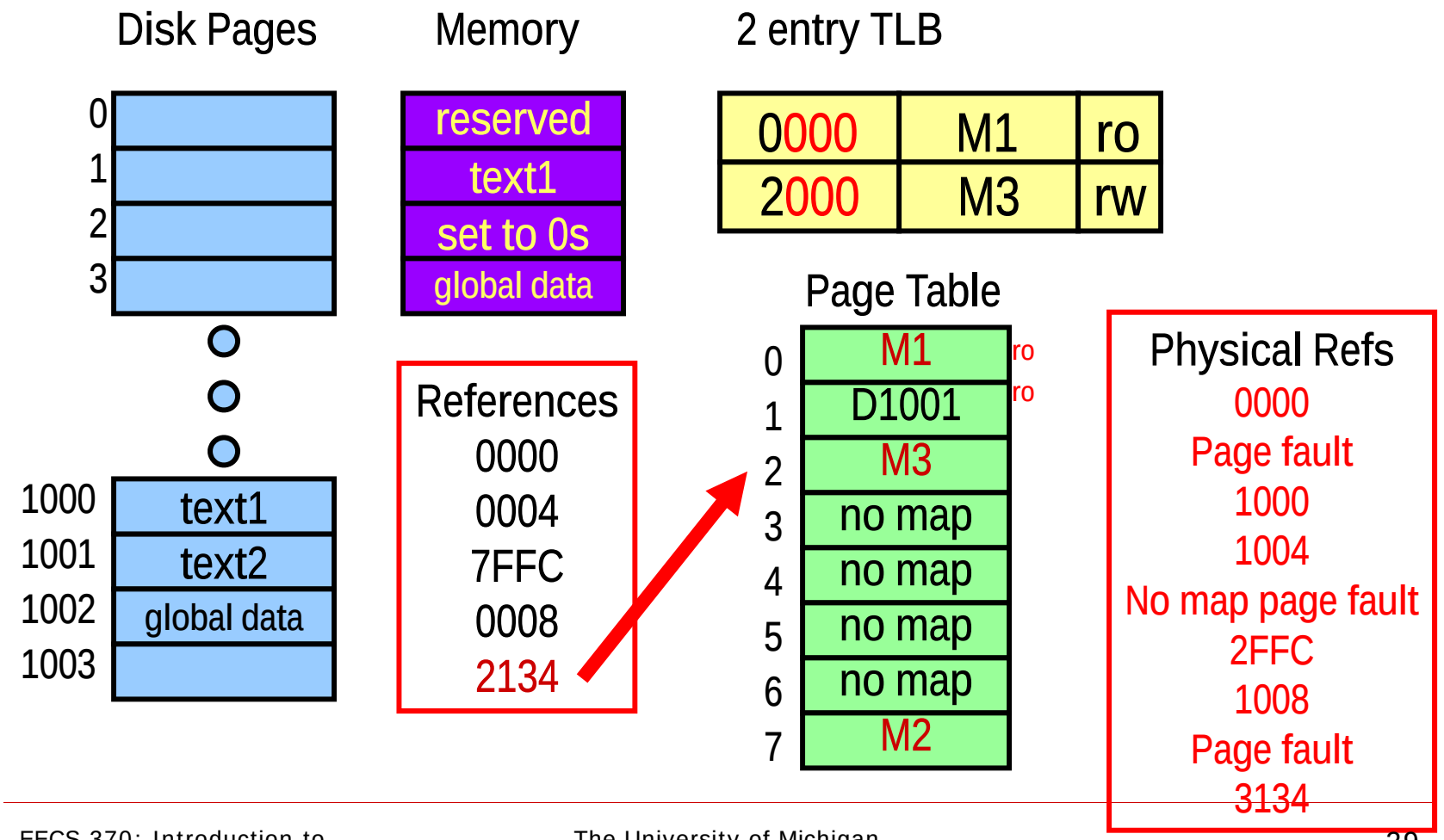
Reference 2134



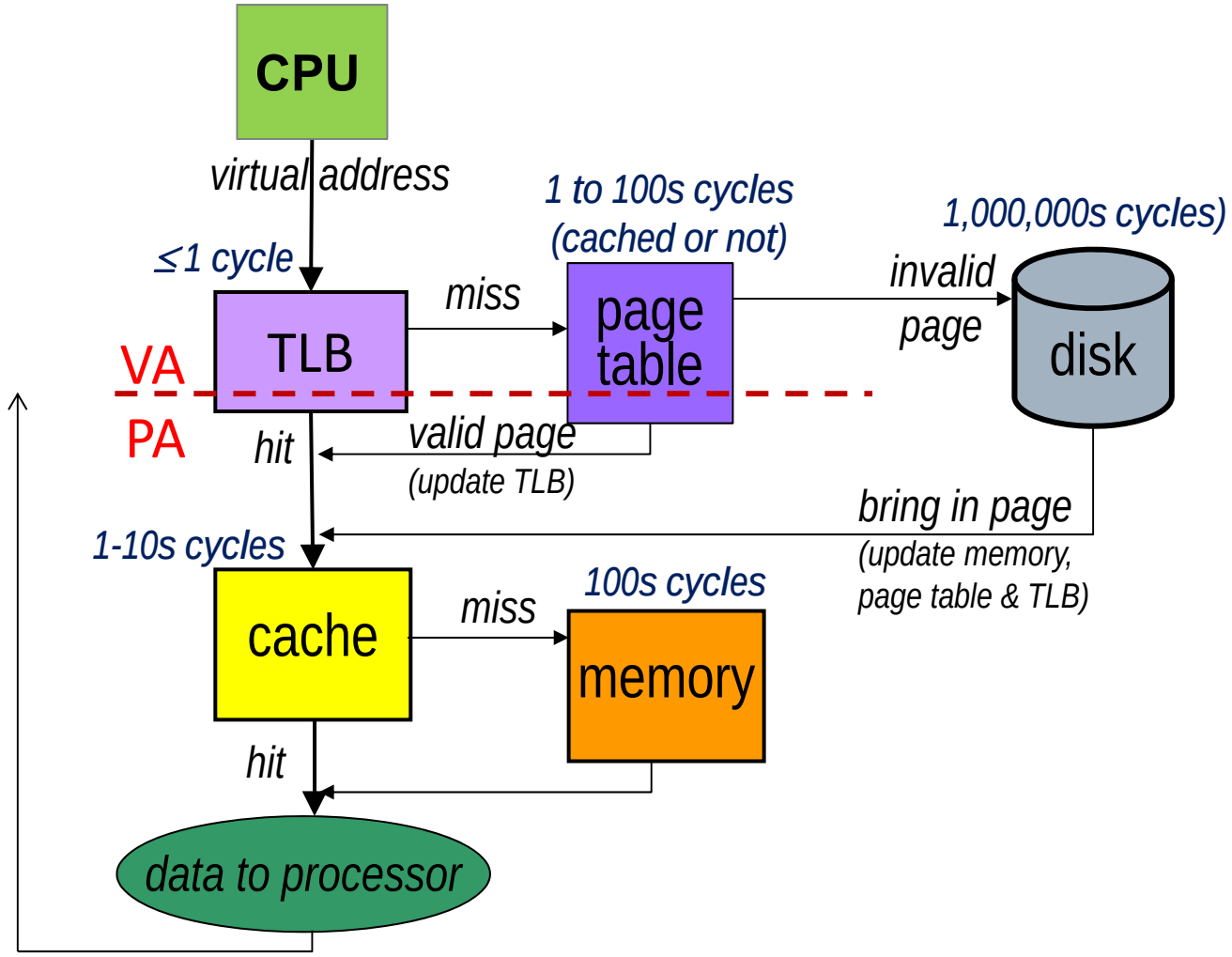
Reference 2134



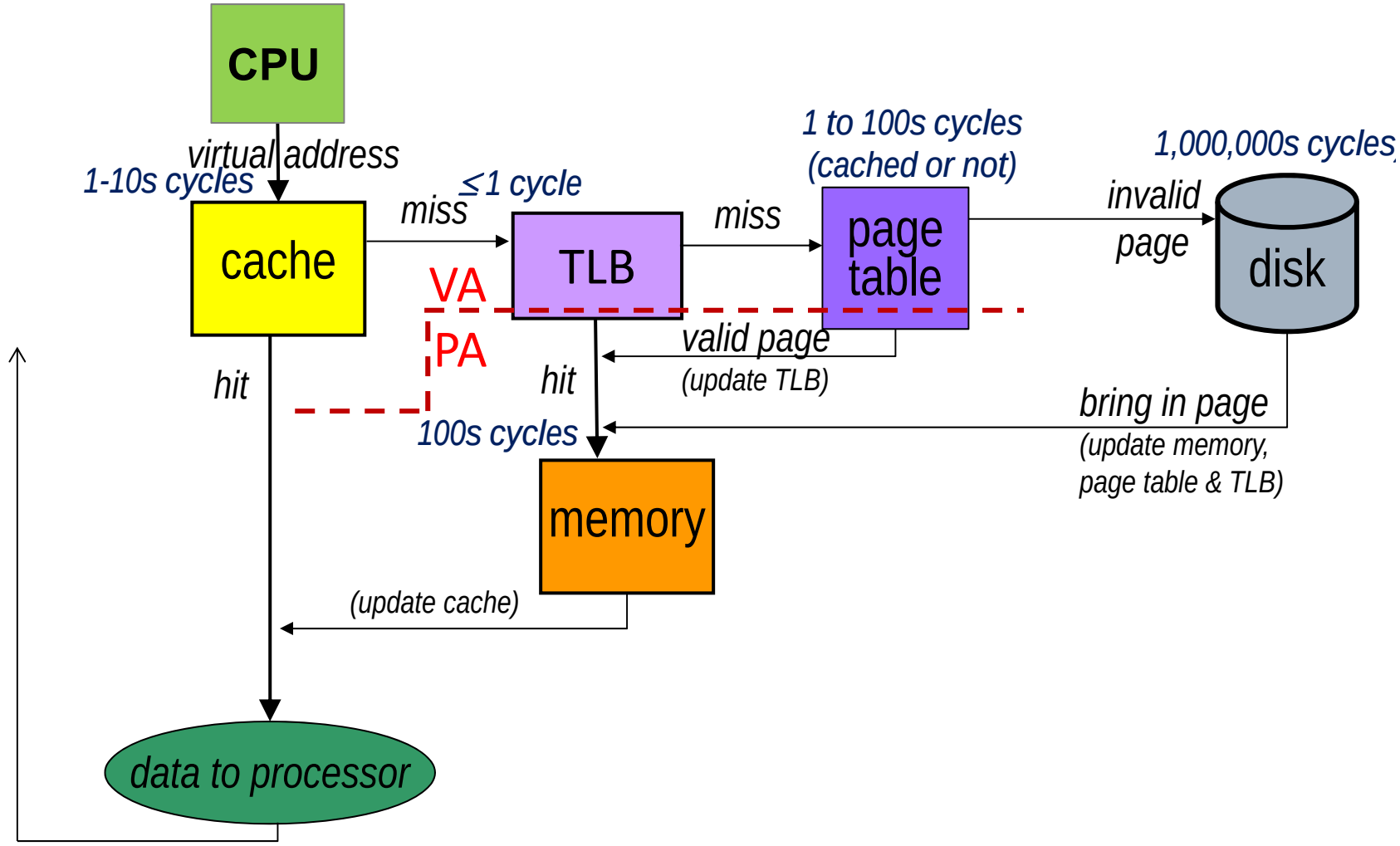
Reference 2134



Physically addressed caches: detailed flow



Virtually addressed caches: detailed flow



Class problem 2 – 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 memory accesses)
  - Accesses to main memory require 30 cycles and hard drive require 100,000 cycles.

Notes:

- The TLB access and cache access are sequential.
- If data is swapped out to hard drive, the TLB will always miss.
- TLB hits will not cause a page fault.
- The page table is uncacheable and always available in main memory.
- Upon retrieval from cache, main memory or hard drive, the data is sent immediately to the CPU, while other updates occur in parallel.

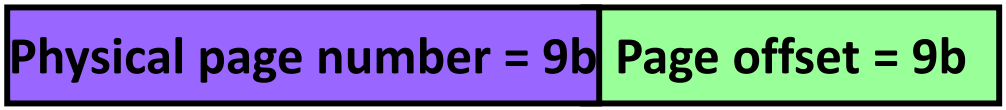
Compute the average latency of a memory access for this machine.

Problems solutions

Lecture 20 – problem 2, part 2

Virtual Address	Virtual Super Page	Virtual Page	Page offset	Page fault?	Physical page num.	Physical Address
0x000F0C	0x00	0x07	0x10C	Y	0x000	0x0010C
0x001F0C	0x00	0x0F	0x10C	Y	0x001	0x0030C
0x020F0C	0x01	0x07	0x10C	Y	0x002	0x0050C

Virtual address = 24b

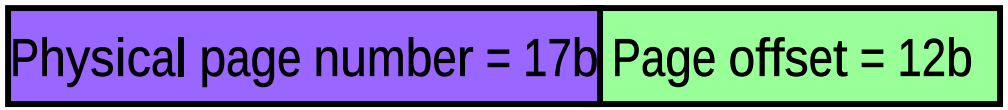


Physical address = 18b

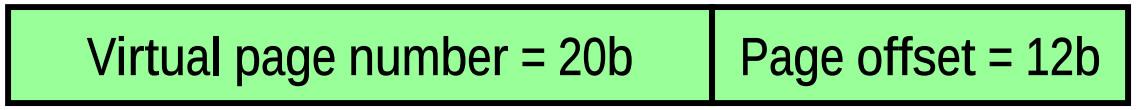
Assume memory for pages is "somewhere else" in memory

Class Problem 1 – VM architecture

Page Offset: 12 bits (4KB page size)



Physical address = 29b (512MB Mem size)



Virtual address = 32b

Page table entry = 17b + 8b ~ 4B
Page table size = 2^20 x 4B = 4MB
How many pages does the page table occupy? 4MB / 4KB = 1K pages

Memory left for page tables = 512MB – 128MB (Data) = 384MB
Max # processes = 384MB / 4MB (1 page table per process) = 96

smaller

Class problem 2 – VM performance

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

Physically tagged:

Address translation: 1 + 0.01*30

Data access: 1 + 0.05*(30 + 0.001*100k)

Virtually tagged:

Address translation: 0.05*(1 + 0.01*30)

Data access: 1 + 0.05*(30 + 0.001*100k)

Avg Mem Access = Addr translation + Data access