Lecture 7:
Virtual Memory

ECE 411 COMPUTER ORGANIZATION AND DESIGN

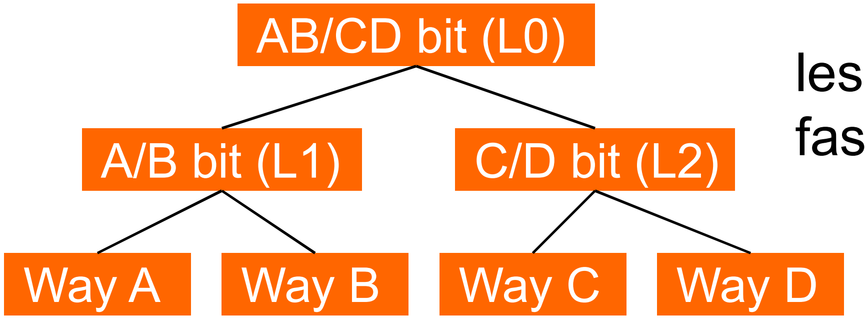
Review: Replacement Policies

- least-recently-used (LRU)
  - ✓ evict the line that has been least recently referenced
    - ○ need to keep track of order that lines in a set have been referenced
    - ○ overhead to do this gets worse as associativity increases

pseudo LRU

- random
  - ✓ just pick one at random
    - ○ easy to implement
    - ○ slightly lower hit rates than LRU on average
- not-most-recently-used
  - ✓ track which line in a set was referenced most recently, pick randomly from the others
    - ○ compromise in both hit rate and implementation difficulty
- virtual memories
  - ✓ use similar policies but spend more effort to improve hit rate

Review: Pseudo LRU Algorithm



less hardware than LRU & faster than LRU

L2L1L0 = 000,
there is a hit in Way B, what is the new updated L2L1L0?

L2L1L0 = 010,
a way needs to be replaced, which way would be chosen?

LRU update algorithm

	CD	AB	AB/CD
Way hit	**L2**	**L1**	**L0**
Way A	---	0	0
Way B	---	1	0
Way C	0	---	1
Way D	1	---	1

Replacement Decision

CD	AB	AB/CD	
L2	**L1**	**L0**	**Way to replace**
X	1	1	Way A
X	0	1	Way B
1	X	0	Way C
0	X	0	Way D

Virtual Memory

- virtual memory – separation of logical memory from physical memory
  - ✓ only a part of the program needs to be in memory for execution
    - ○ logical address space can be much larger than physical address space
  - ✓ allows address spaces to be shared by several processes (or threads)
  - ✓ allows for more efficient process creation

- virtual memory can be implemented via:
  - ✓ demand paging
  - ✓ demand segmentation

main memory is like a cache to the hard disc!

Virtual Address

- concept of a virtual (or logical) address space that is bound to a separate physical address space is central to memory management
  - ✓ virtual address – generated by CPU
  - ✓ physical address – seen by memory

- virtual and physical addresses
  - ✓ are the same in compile-time
  - ✓ differ in execution-time address-binding schemes

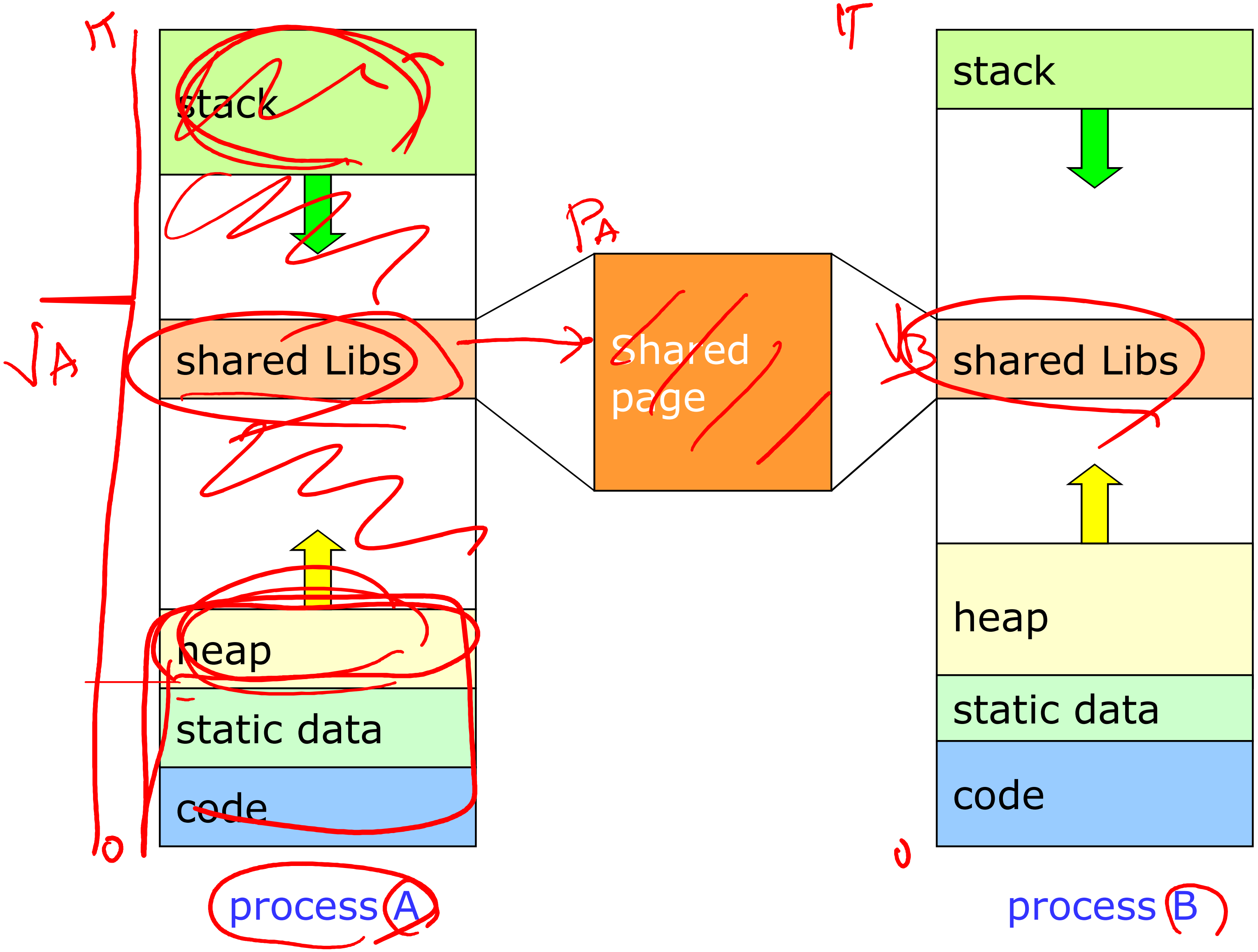
Advantages of Virtual Memory

1 ● translation
  - ✓ program can be given consistent view of memory, even though physical memory is scrambled
  - ✓ only the most important part of program ("working set") must be in physical memory
  - ✓ contiguous structures (like stacks) use only as much physical memory as necessary yet grow later

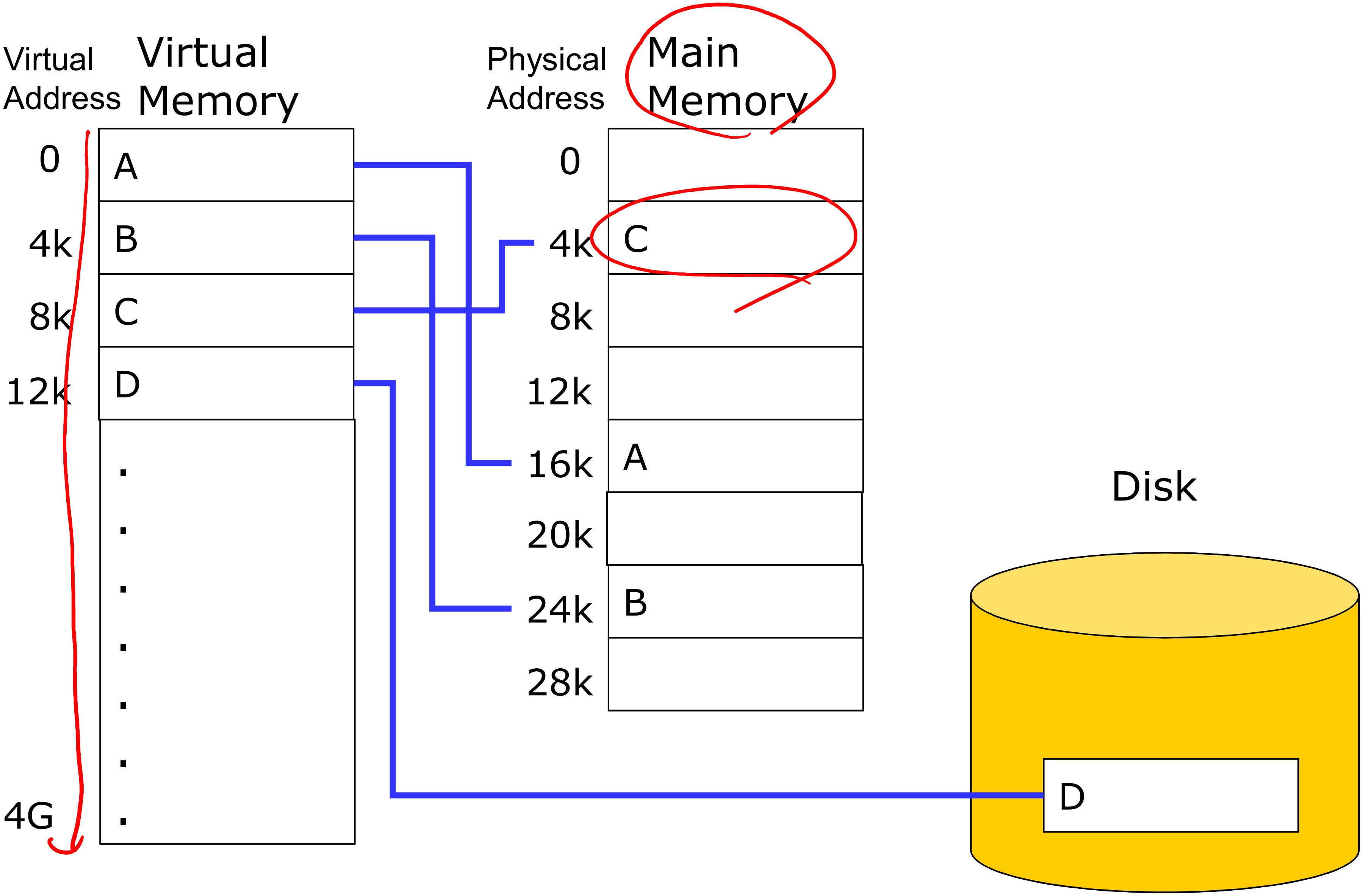
2 ● protection
  - ✓ different threads (or processes) protected from each other
  - ✓ different pages can be given special behavior
    - ○ (read only, invisible to user programs, etc.).
  - ✓ kernel data protected from user programs
  - ✓ very important for protection from malicious programs
    - ○ ~~far more "viruses" under Microsoft Windows~~

3 ● sharing
  - ✓ map the same physical page to multiple users ("shared memory")

Use of Virtual Memory



Virtual vs. Physical Address Space



Paging

- frame
  - ✓ divide physical memory into fixed-size blocks (e.g., 4KB)
- pages
  - ✓ divide logical memory into blocks of same size (4KB)
  - ✓ to run a program of size *n* pages, need to find n free frames and load program
  - ✓ set up a page table to map page addresses to frame addresses
    - ○ operating system sets up the page table

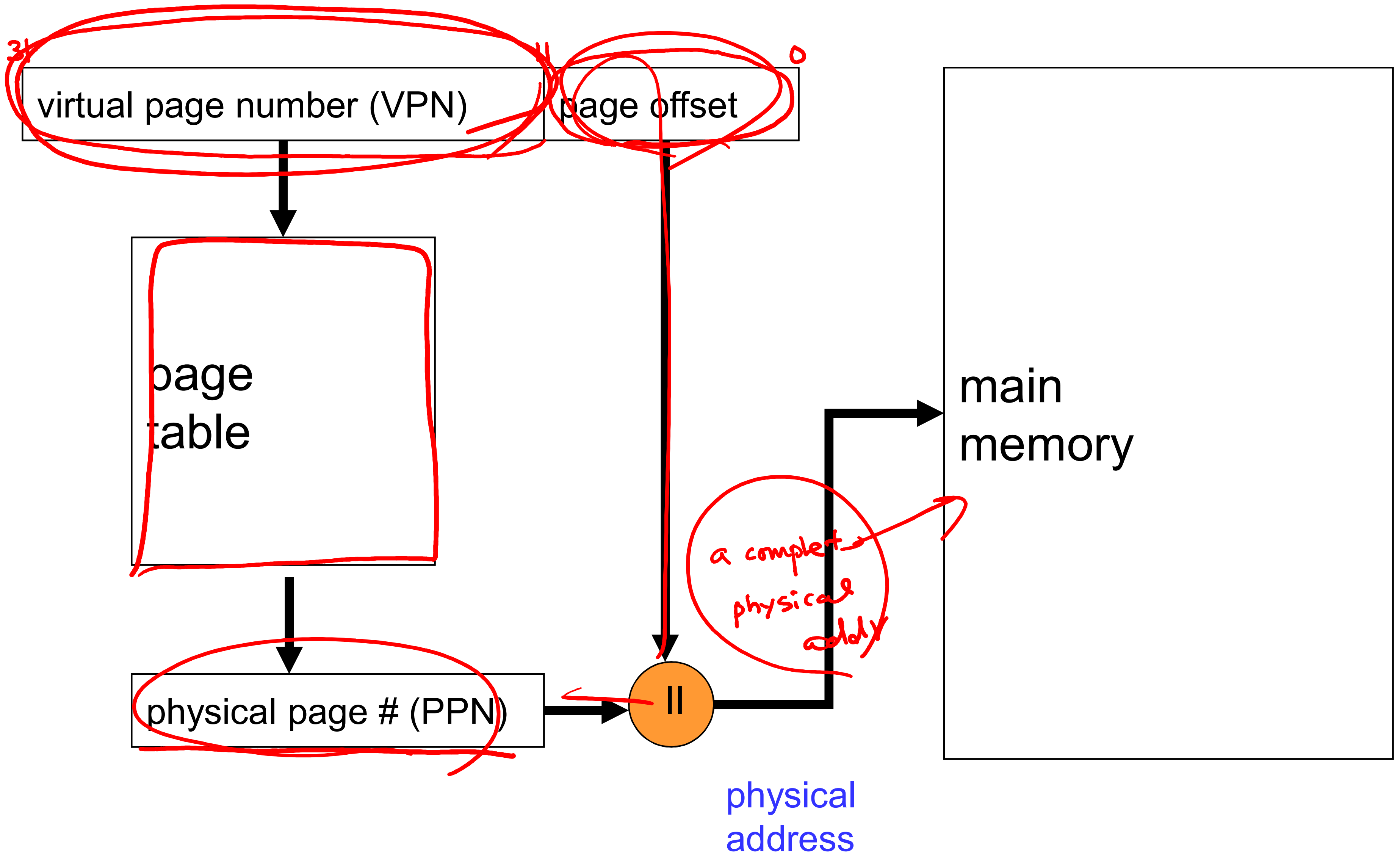
virtual addr

virtual PN

physical addr ✓

physical PN

Page Table and Address Translation



virtual page number (VPN)

page offset

page table

physical page # (PPN)

||

main memory

physical address

Page Table Structure Examples

- one-to-one mapping, space?
  - ✓ large pages
    - ○ Internal fragmentation (similar to having large line sizes in caches)
  - ✓ small pages
    - ○ page table size issues
- multi-level paging
- inverted page table

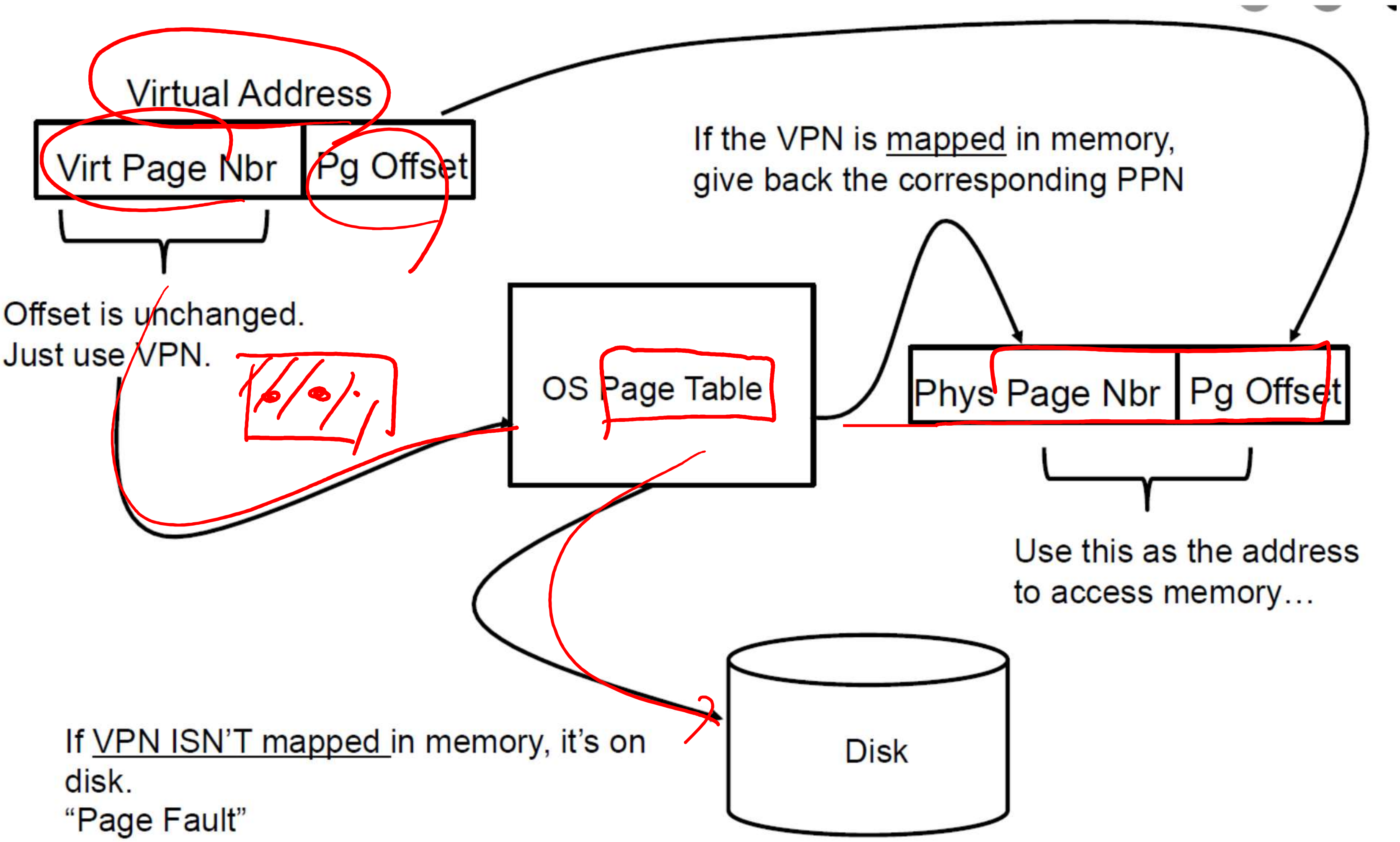
example:
64 bit address space, 4 KB pages (12 bits), 512 MB (29 bits) RAM

Number of pages = $2^{64}/2^{12} = 2^{52}$
(The page table has as many entrees)

Each entry is ~4 bytes, the size of the Page table is 2^{54} Bytes = 16 Petabytes!

Can’t fit the page table in the 512 MB RAM!

Virtual to Physical Page Translation



Offset is unchanged.
Just use VPN.

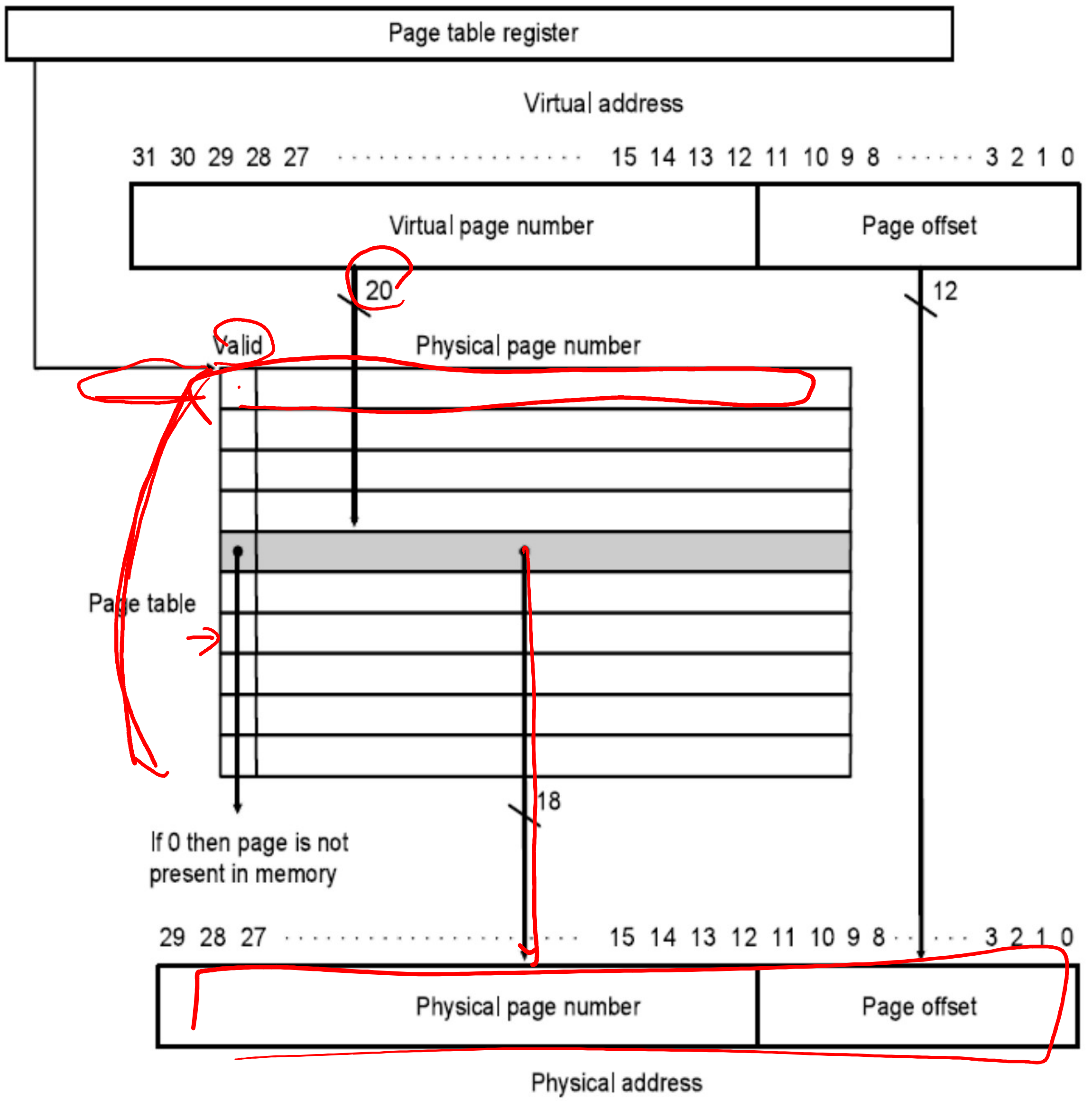
If the VPN is mapped in memory,
give back the corresponding PPN

Use this as the address
to access memory…

If VPN ISN'T mapped in memory, it's on disk.
"Page Fault"
Create a V-to-P map for it and fetch whole page
from disk into main memory

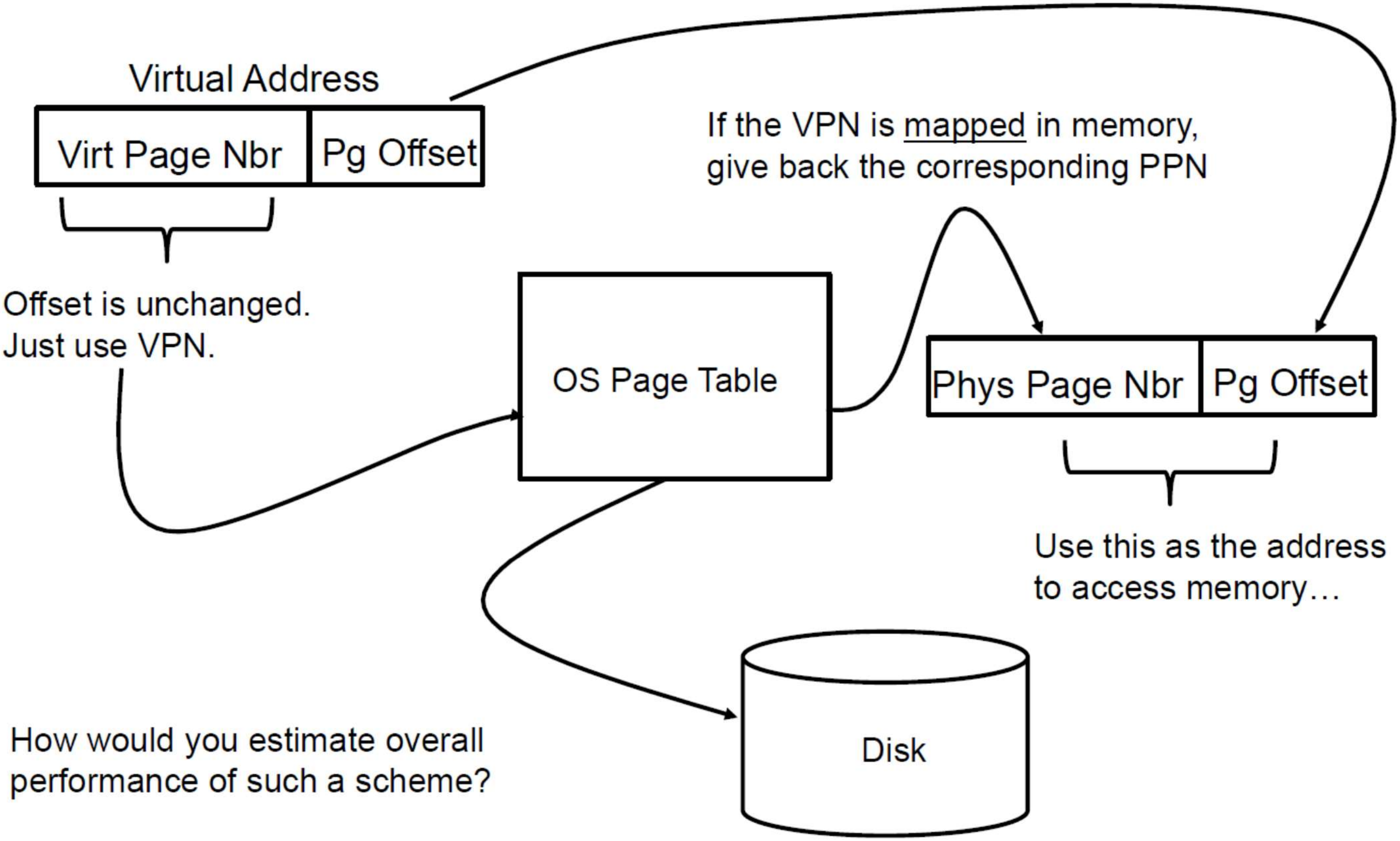
from COS 318 Operating Systems: Virtual Memory and Address Translation

(OS) Page Table



from COS 318 Operating Systems: Virtual Memory and Address Translation

Efficiency?



Virtual Address

Virt Page Nbr | Pg Offset

Offset is unchanged.
Just use VPN.

OS Page Table

If the VPN is mapped in memory,
give back the corresponding PPN

Phys Page Nbr | Pg Offset

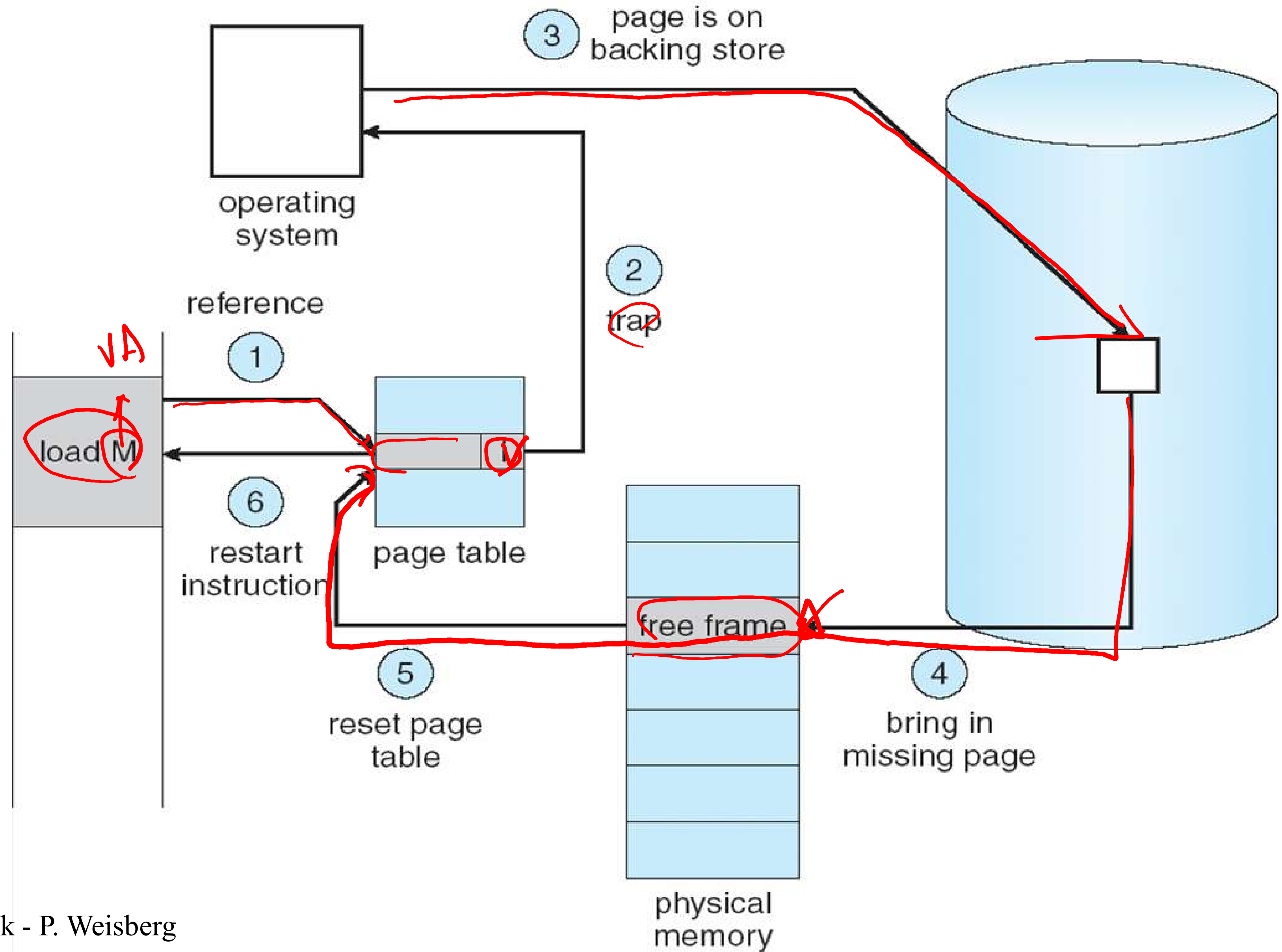
Use this as the address
to access memory…

Disk

How would you estimate overall
performance of such a scheme?

from COS 318 Operating Systems: Virtual Memory and Address Translation

Handling a Page Fault (1)



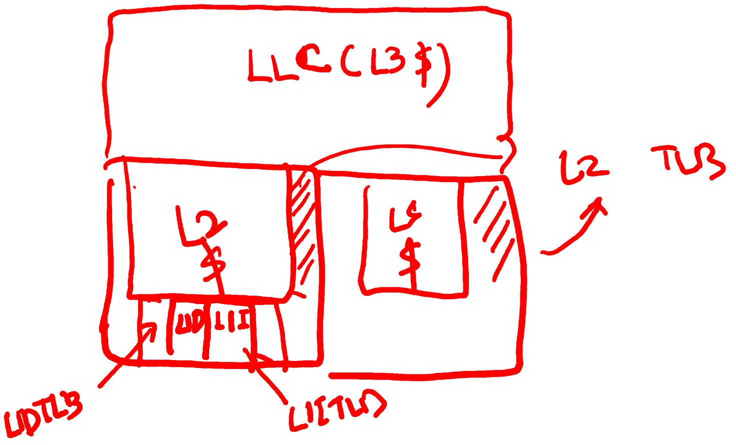
Handling a Page Fault (2)

1. if there is ever a reference to a page not in memory, first reference will cause page fault.
2. page fault is handled by the appropriate OS service routines.
3. locate needed page on disk (in file or in backing store).
4. swap page into free frame (assume available).
5. reset page tables – valid-invalid bit = v.
6. restart the instruction that caused the page fault.

explain the steps illustrated in the prev. slides

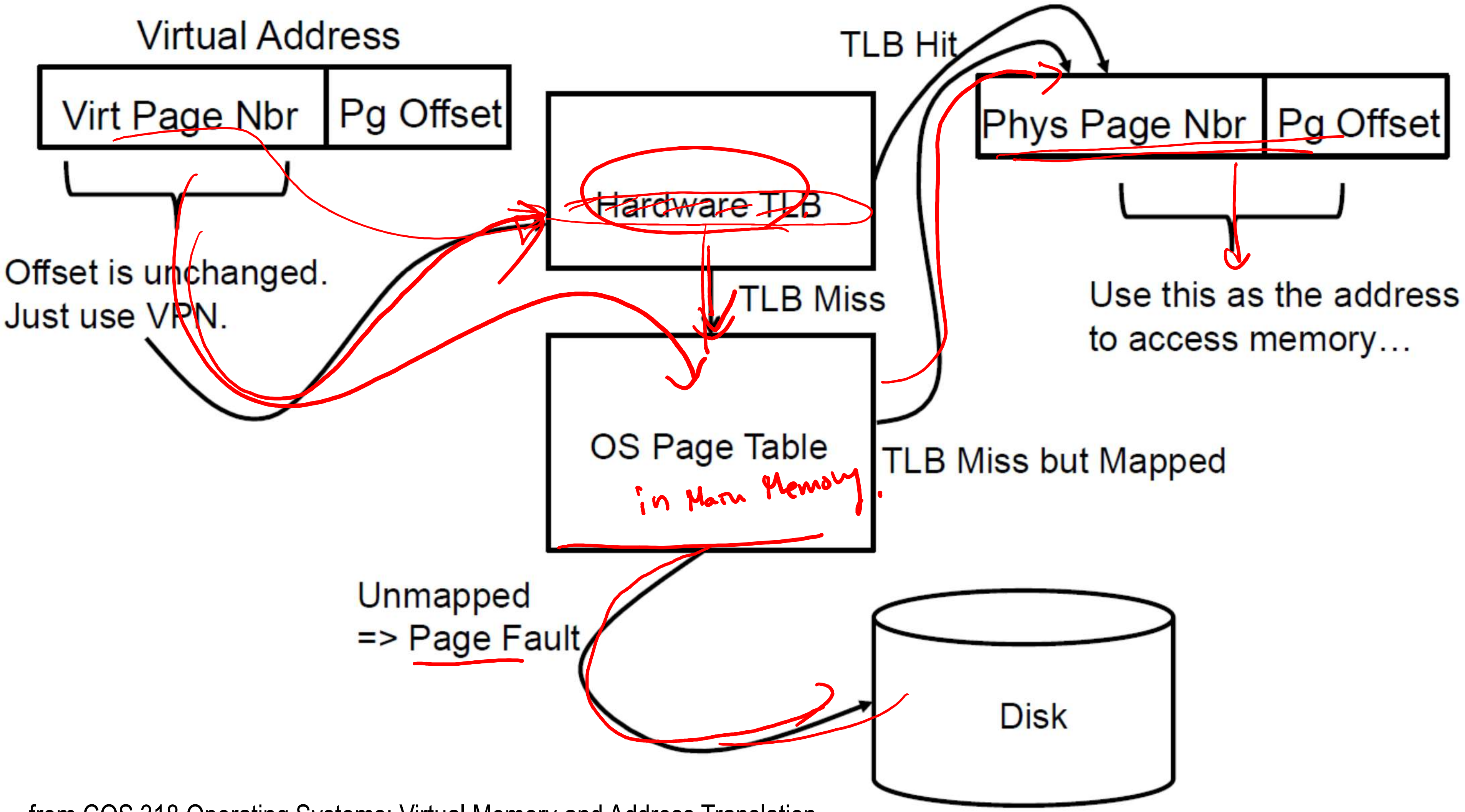
Fast Address Translation

- how often address translation occurs?
- where the page table is kept?
- keep translation in the hardware
- use Translation Lookaside Buffer (TLB)
  - ✓ instruction-TLB & data-TLB
  - ✓ essentially a cache (tag array = VPN, data array=PPN)
  - ✓ small (32 to 256 entries are typical)
  - ✓ typically fully associative (implemented as a content addressable memory, CAM) or highly associative to minimize conflicts



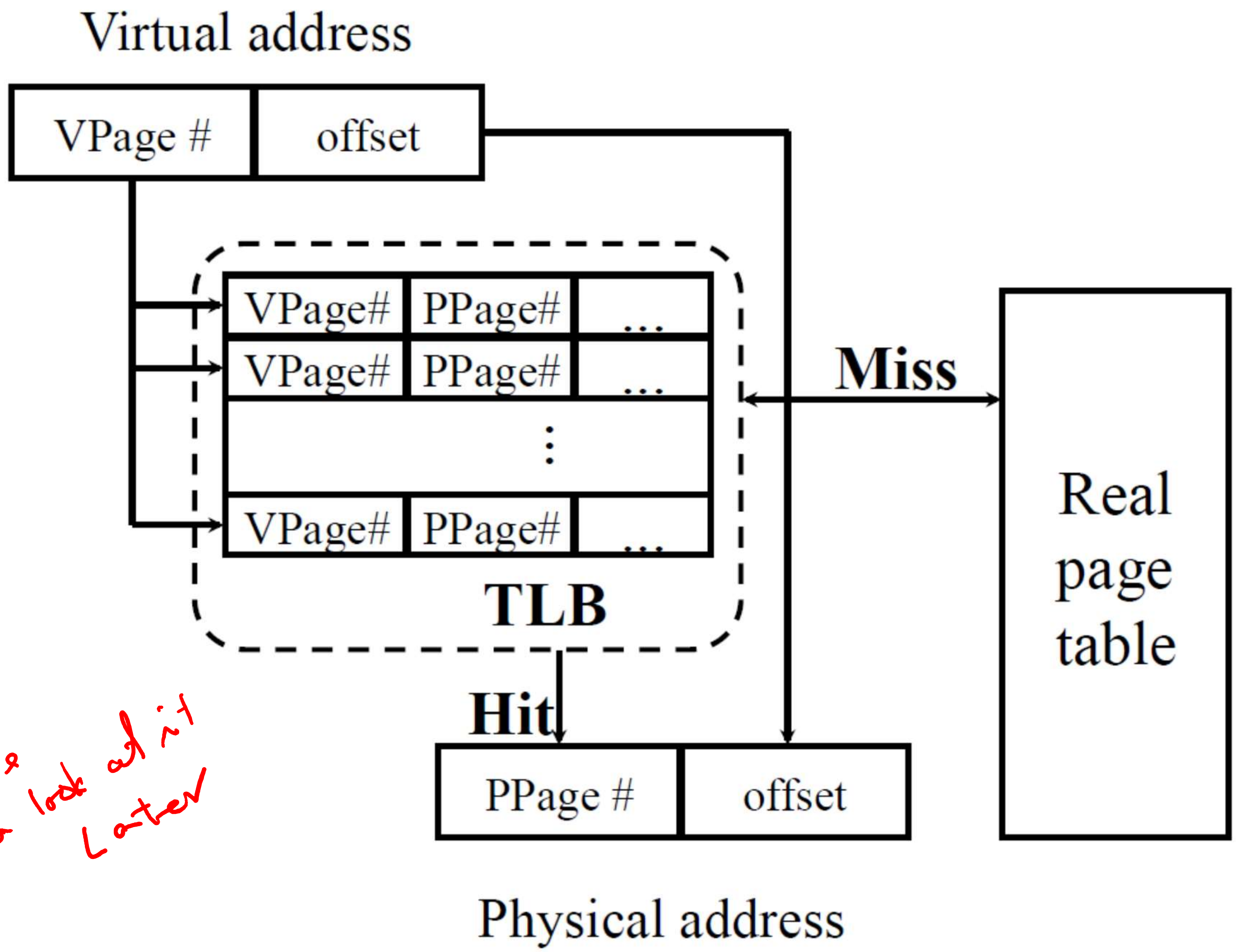
HW Translation Lookaside Buffer (TLB)

- store most common V-P mappings in hardware table



from COS 318 Operating Systems: Virtual Memory and Address Translation

TLB



have a look at it Later

What Happens on a Context Switch?

- each process has its own address space
- so, each process has its own page table
- so, page-table entries are only relevant for a particular process
- thus, the TLB must be flushed on a context switch
  - ✓ this is why context switches are so expensive

Alternative to Flushing

address space IDs

- we can avoid flushing the TLB if entries are associated w/ an address space

processes.

modify

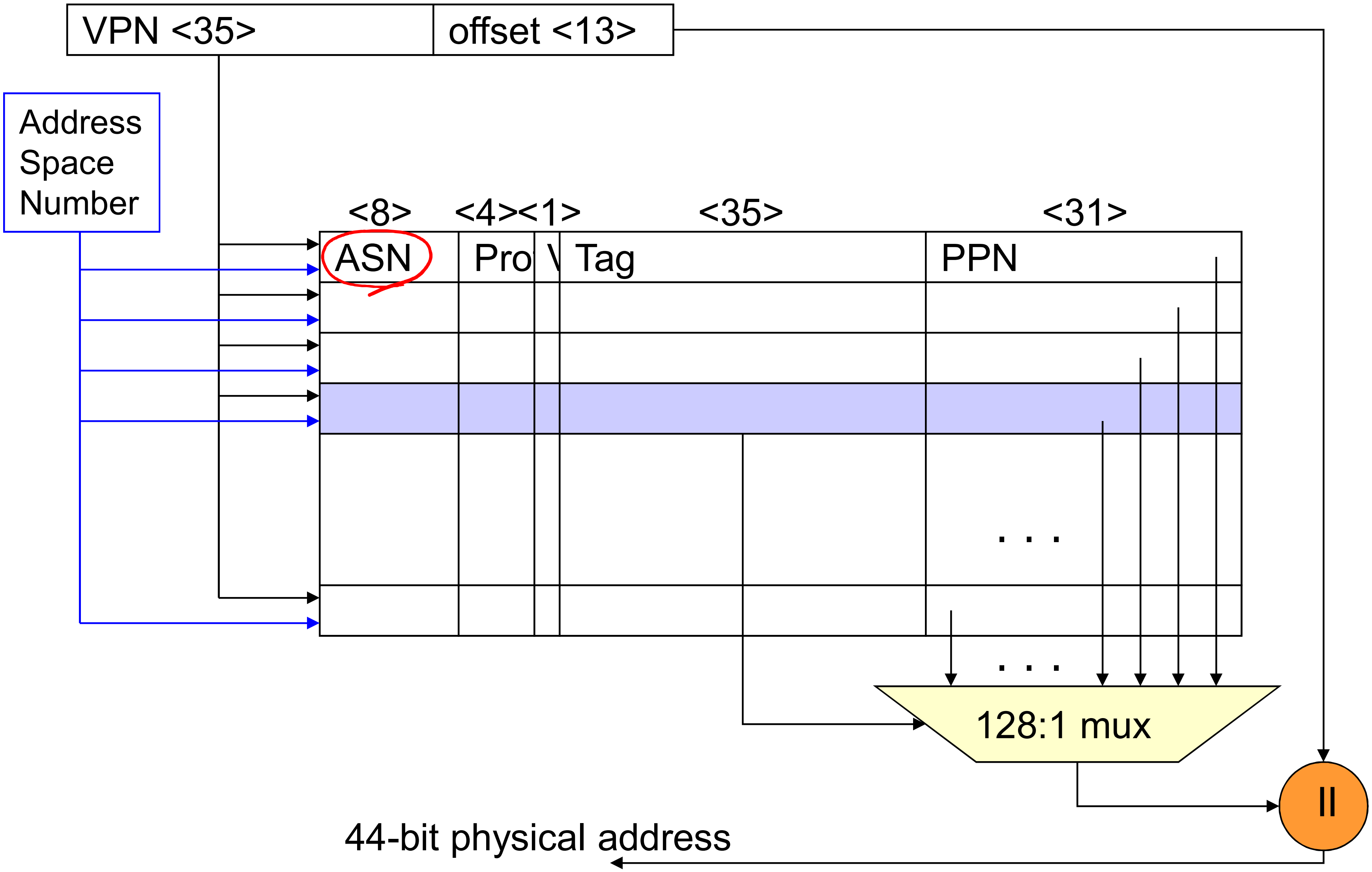
4	1	1	1	2	20
ASID	V	R	M	prot	page frame number

valid

read only

- when would this work well?
- when would this not work well?

Example: Alpha 21264 data TLB

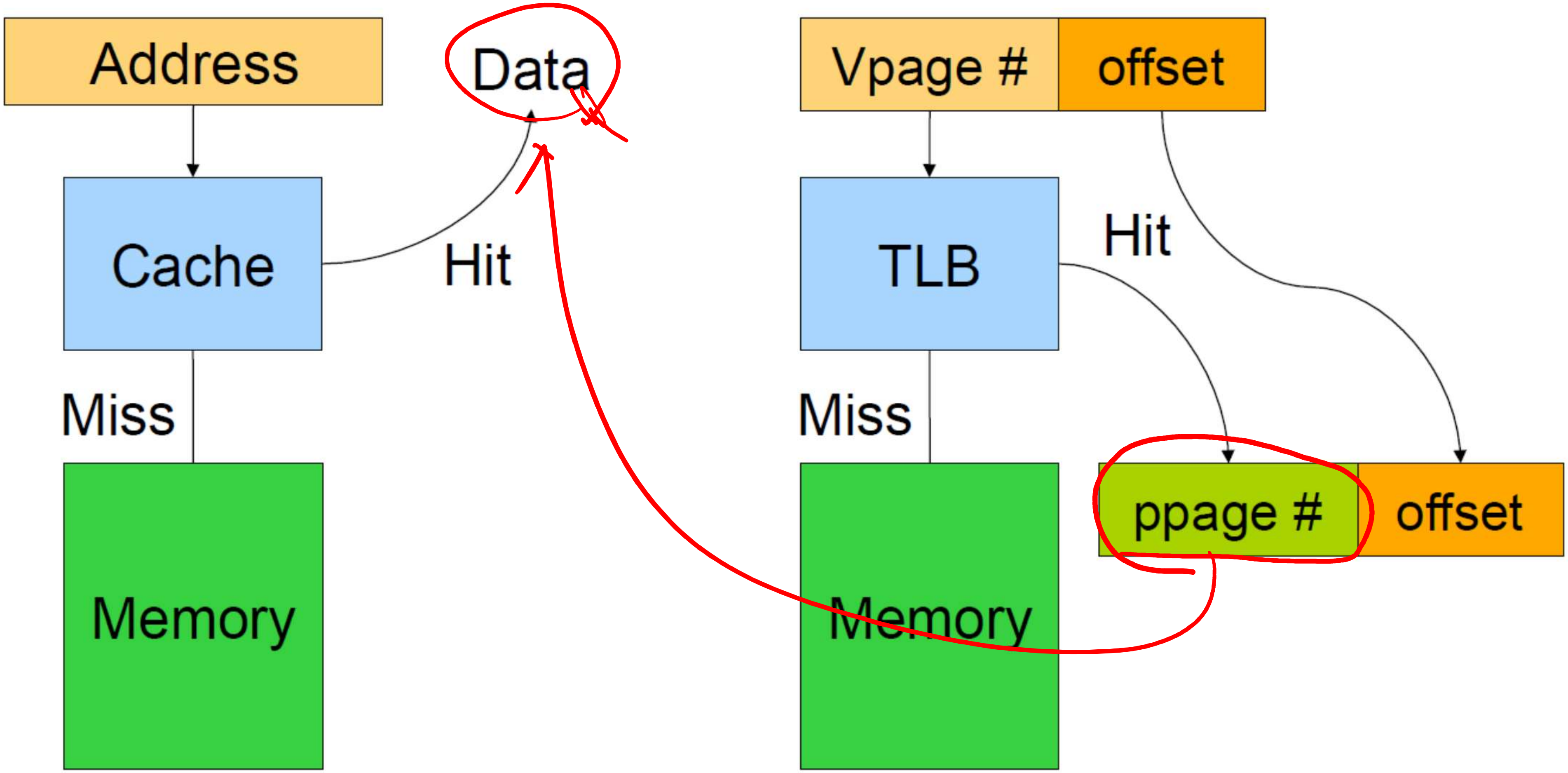


Hard versus Soft Page Faults

- hard page faults
  - ✓ those page faults that require issuing a read from secondary storage.
- soft page faults
  - ✓ those page faults where the page is already in main memory but the TLB and/or the PTE has marked the page as invalid.
  - ✓ soft faults are used when hardware support is not available to handle TLB misses

Cache vs TLB

- similarities
  - ✓ cache a portion of memory
  - ✓ access (lower level) memory on a miss



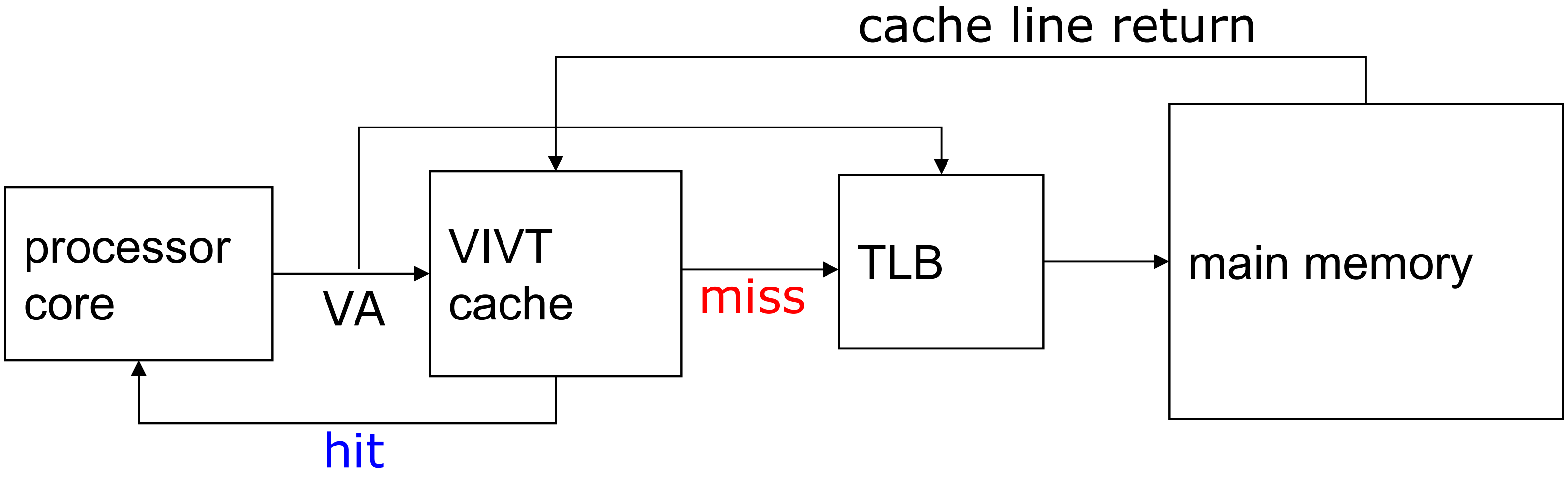
Caches and Virtual Memory

- do we send virtual or physical addresses to the cache?
  - ✓ virtual → faster, because don't have to translate
    - ○ issue: different programs can reference the same virtual address, either creates security/correctness hole or requires flushing the cache every time you context switch
  - ✓ physical → slower, but no security issue

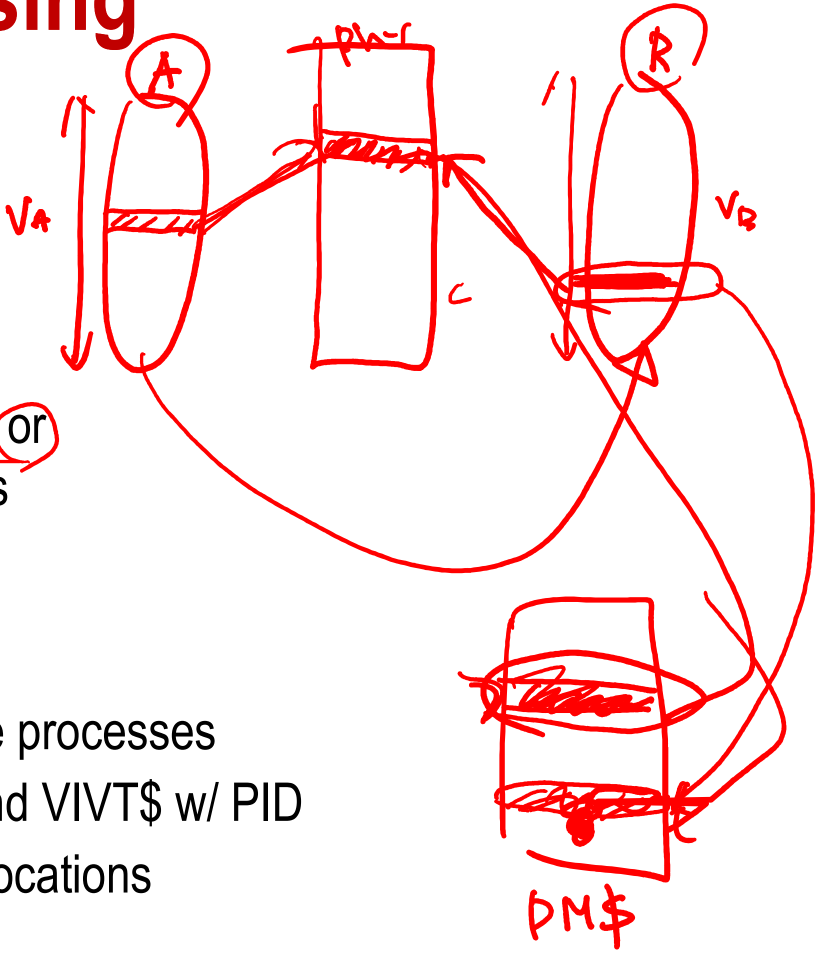
- actually, there are four possibilities
  - ✓ VIVT: Virtually-indexed Virtually-tagged Cache
  - ✓ PIPT: Physically-indexed Physically-tagged Cache
  - ✓ VIPT: Virtually-indexed Physically-tagged Cache
  - ✓ ~~PIVT: Physically-indexed Virtually-tagged Cache~~

Virtually-Indexed Virtually-Tagged

- fast cache access
- only require address translation when going to memory (miss)
- issues?



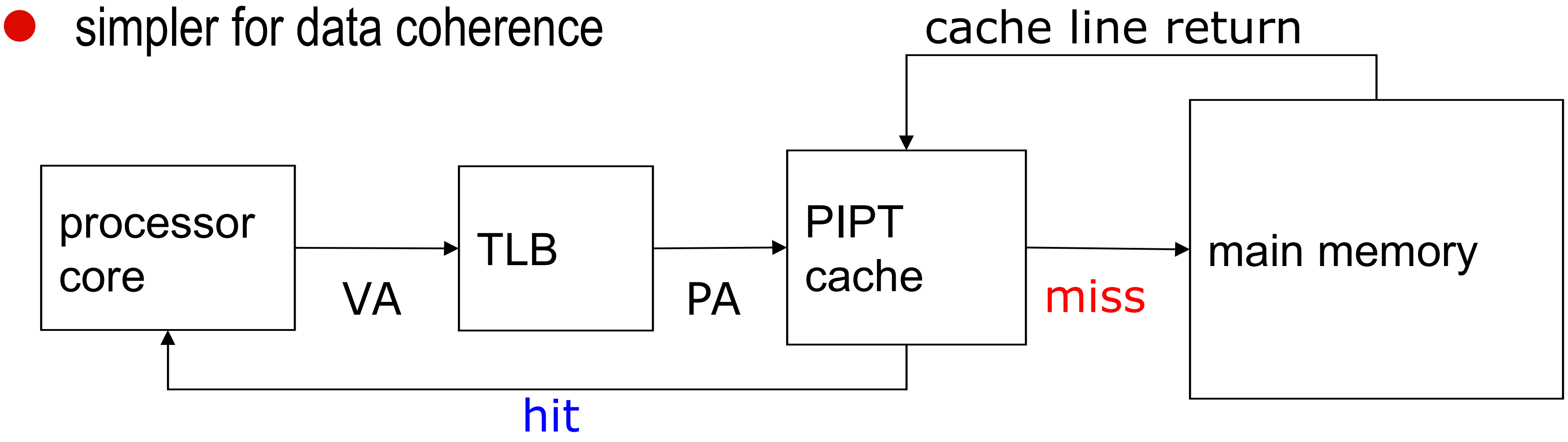
VIVT Cache Issues - Aliasing



- homonym
  - ✓ same VA maps to different PAs
  - ✓ occurs when there is a context switch
  - ✓ solutions
    - ○ include process ID (PID) in cache or
    - ○ flush cache upon context switches
- synonym (also a problem in VIPT)
  - ✓ different VAs map to the same PA
  - ✓ occurs when data is shared by multiple processes
  - ✓ duplicated cache line in VIPT cache and VIVT$ w/ PID
  - ✓ data is inconsistent due to duplicated locations
  - ✓ solution
    - ○ can write-through solve the problem?
    - ○ flush cache upon context switch
    - ○ if (index+offset) < page offset, can the problem be solved? (discussed later in VIPT)

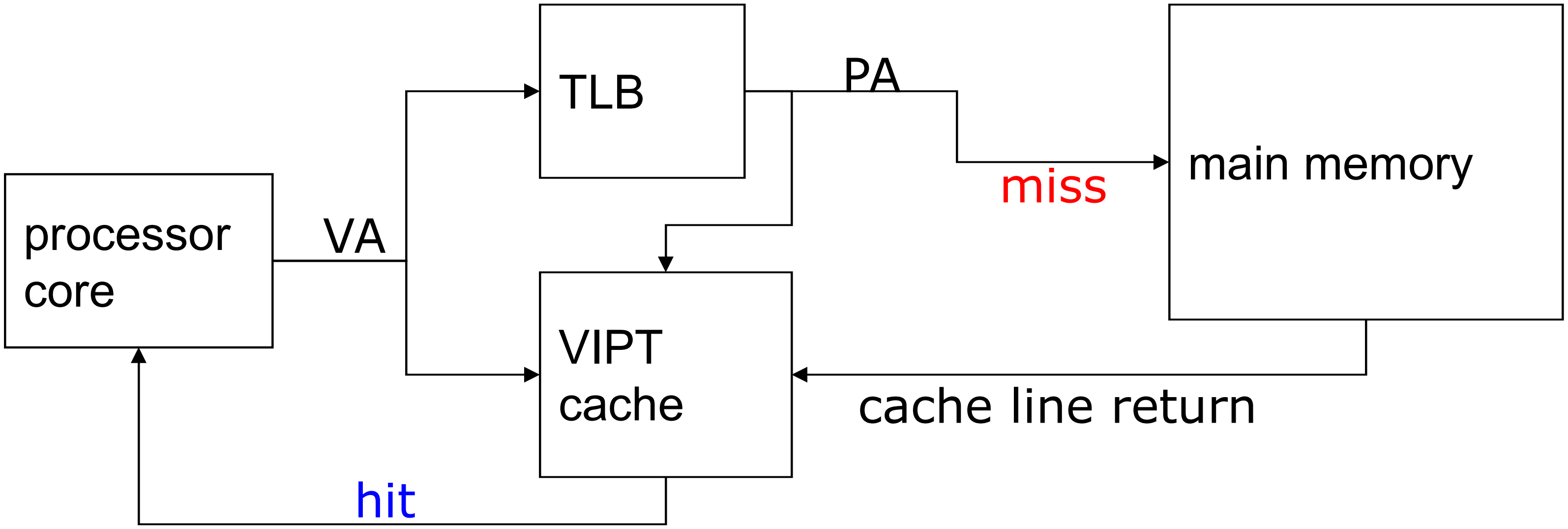
Physically-Indexed Physically-Tagged

- slower, always translate address before accessing memory
- simpler for data coherence

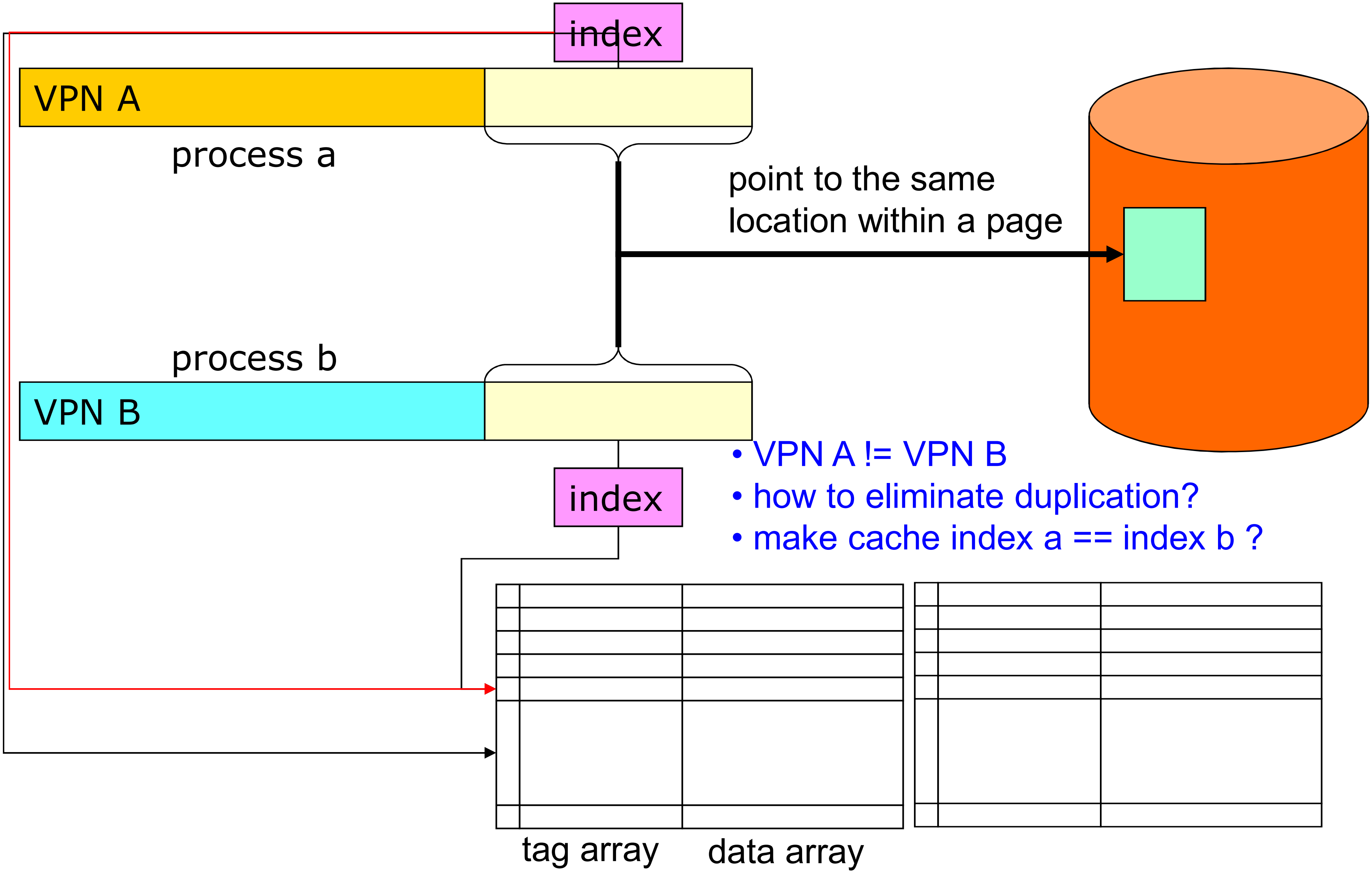


Virtually-Indexed Physically-Tagged

- gain benefit of a VIVT and PIPT
- parallel access to TLB and VIPT cache
- no homonym
  - ✓ how about synonym?



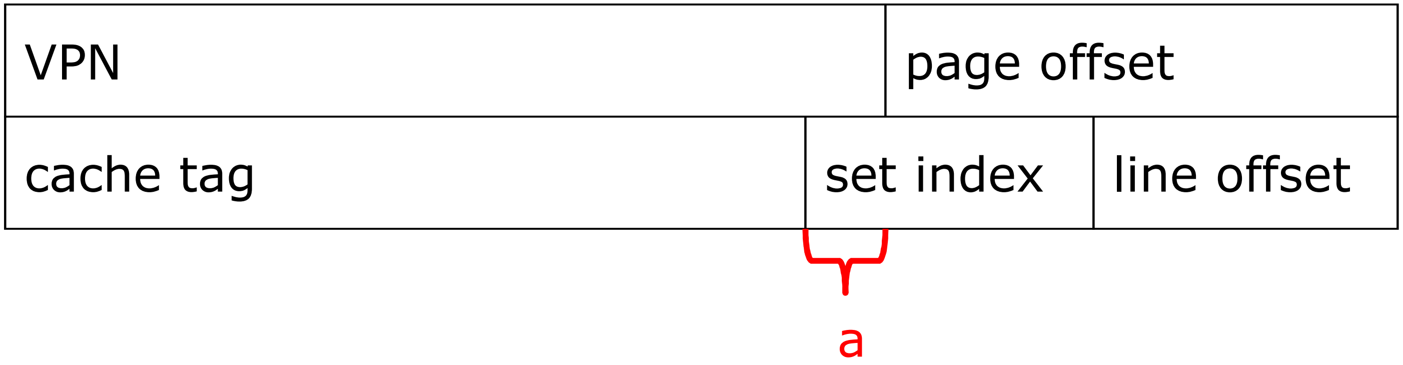
Deal w/ Synonym in VIPT Cache



- VPN A != VPN B
- how to eliminate duplication?
- make cache index a == index b ?

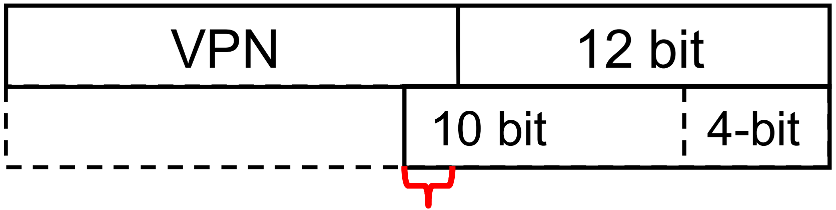
Synonym in VIPT Cache

- if two VPNs do not differ in a then there is no synonym problem, since they will be indexed to the same set of a VIPT cache
- imply # of sets cannot be too big
- max number of sets = page size / cache line size
  - ✓ ex: 4KB page, 32B line, max set = 128
- a complicated solution in MIPS R10000



R10000's Solution to Synonym

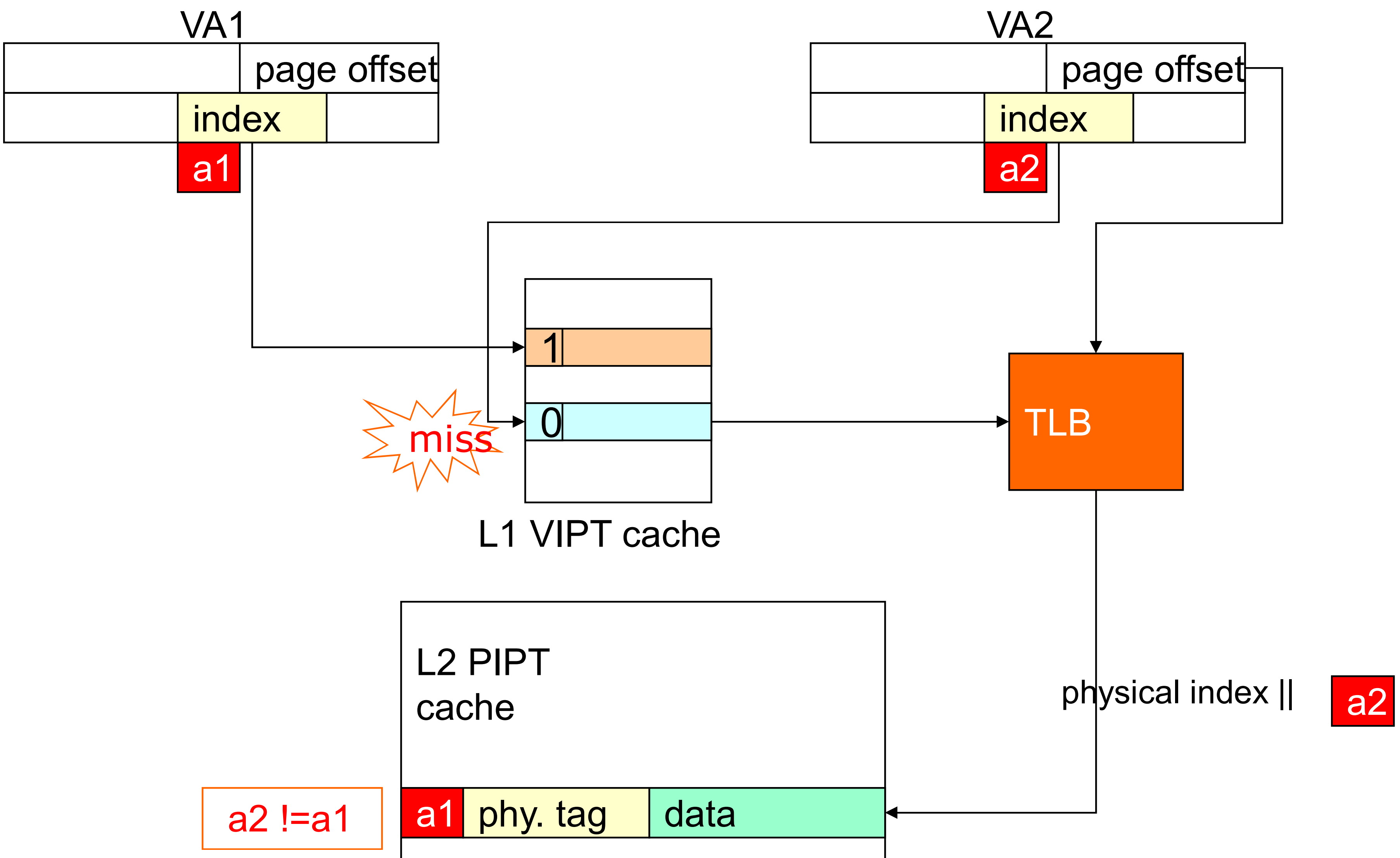
- 32KB 2-Way virtually-indexed L1



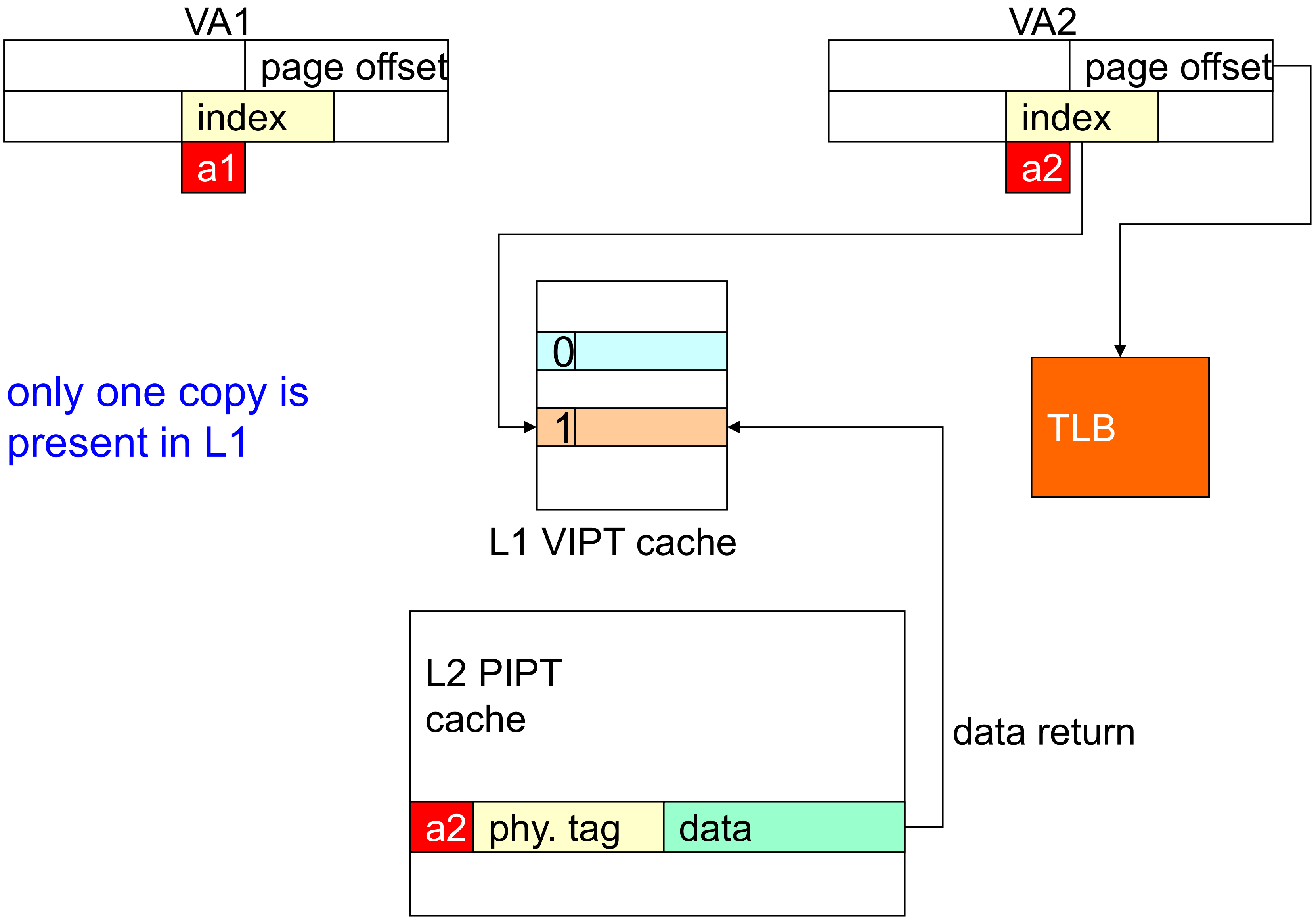
a= VPN[1:0] stored as part of L2 cache Tag

- direct-mapped physical L2
  - L2 is *Inclusive* of L1
  - VPN[1:0] is appended to the "tag" of L2
- given two virtual addresses *VA1* and *VA2* that differs in *VPN[1:0]* and both map to the same physical address *PA*
  - suppose *VA1* is accessed first so blocks are allocated in L1&L2
  - what happens when *VA2* is referenced?
    1 *VA2* indexes to a different block in L1 and misses
    2 *VA2* translates to *PA* and goes to the same block as *VA1* in L2
    3. tag comparison fails (since *VA1*[1:0]≠*VA2*[1:0])
    4. treated just like as a L2 conflict miss ⇒ *VA1*'s entry in L1 is ejected (or dirty-written back if needed) due to inclusion policy

Deal w/ Synonym in MIPS R10000



Deal w/ Synonym in MIPS R10000



only one copy is present in L1

Announcement

- today's lecture: virtual memory
  - ✓ Ch. 5.6 – 5.7 (HP1)
  - ✓ http://csapp.cs.cmu.edu/2e/ch9-preview.pdf
- next lecture: more virtual memory
  - ✓ Ch. 5.6 – 5.7 (HP1)
  - ✓ http://csapp.cs.cmu.edu/2e/ch9-preview.pdf
- MP assignment
  - ✓ MP2 check-point 1 due on 2/11 5pm
  - ✓ HW1 due on 2/13 5pm