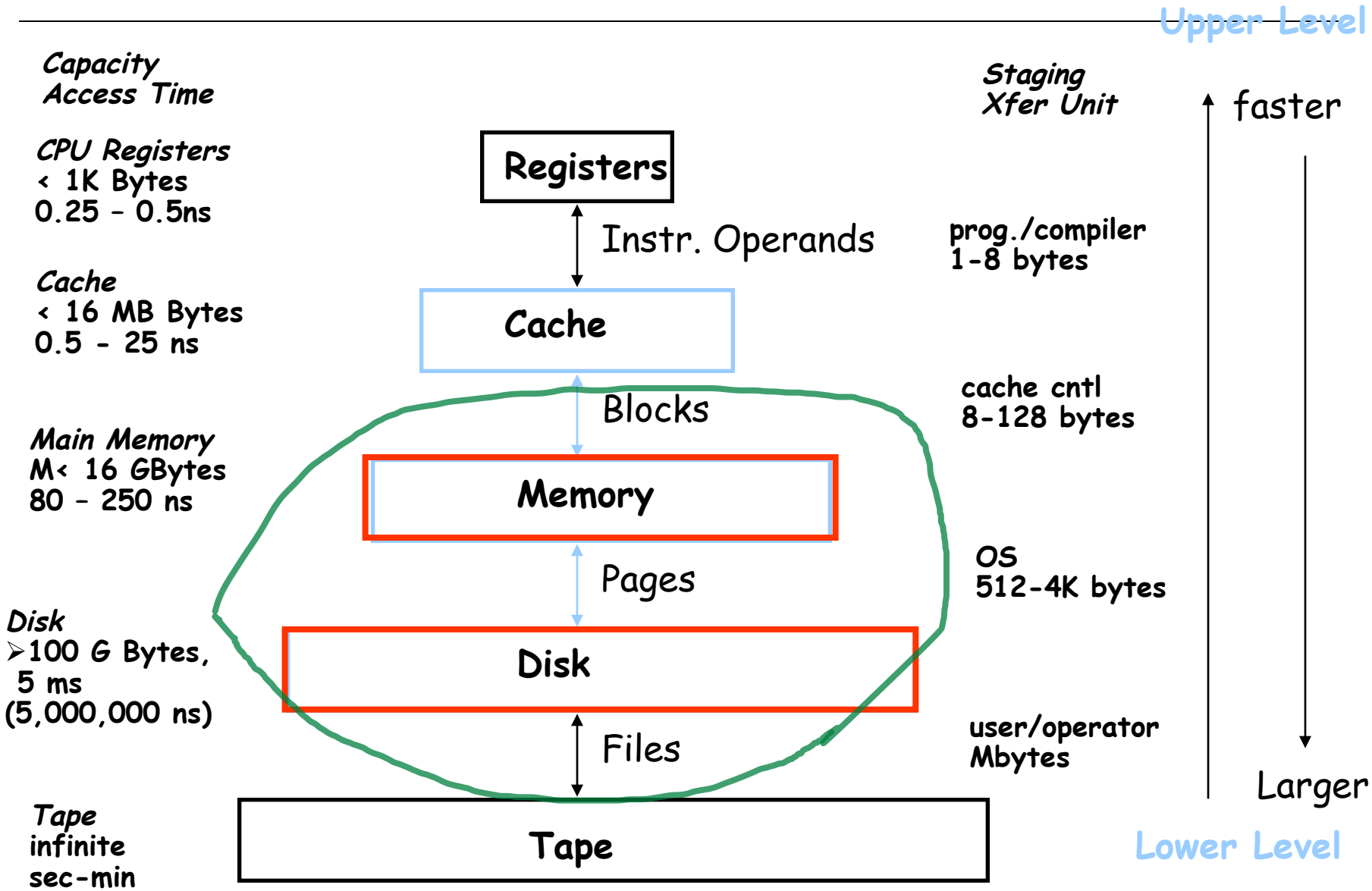
Lecture 9

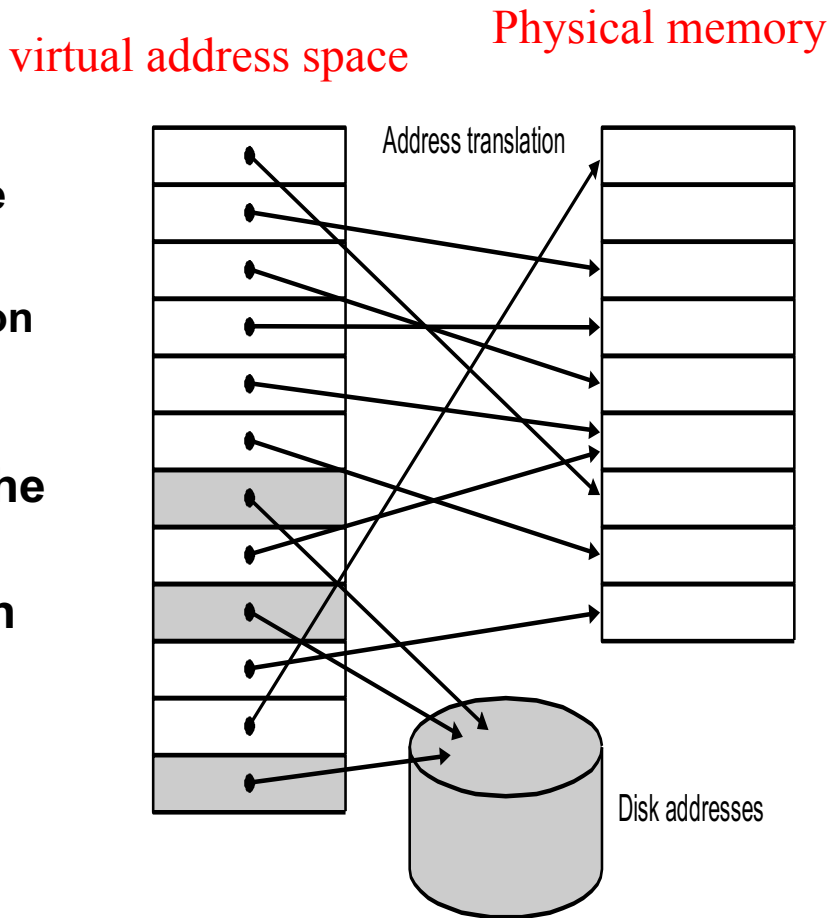
- Virtual Memory

Levels of the Memory Hierarchy



Virtual Memory: Motivation

- Virtual memory is a technique that uses main memory as a "cache" for secondary storage.
- **Allow safe sharing of memory among multiple programs**
  - **Each program has its own address space (Virtual address)**
  - **Virtual memory implements the translation from a program's address to physical address (provide protection)**
- **Allow a single user program to exceed the size of physical memory**
- **Simplify loading a program for execution by providing relocation**

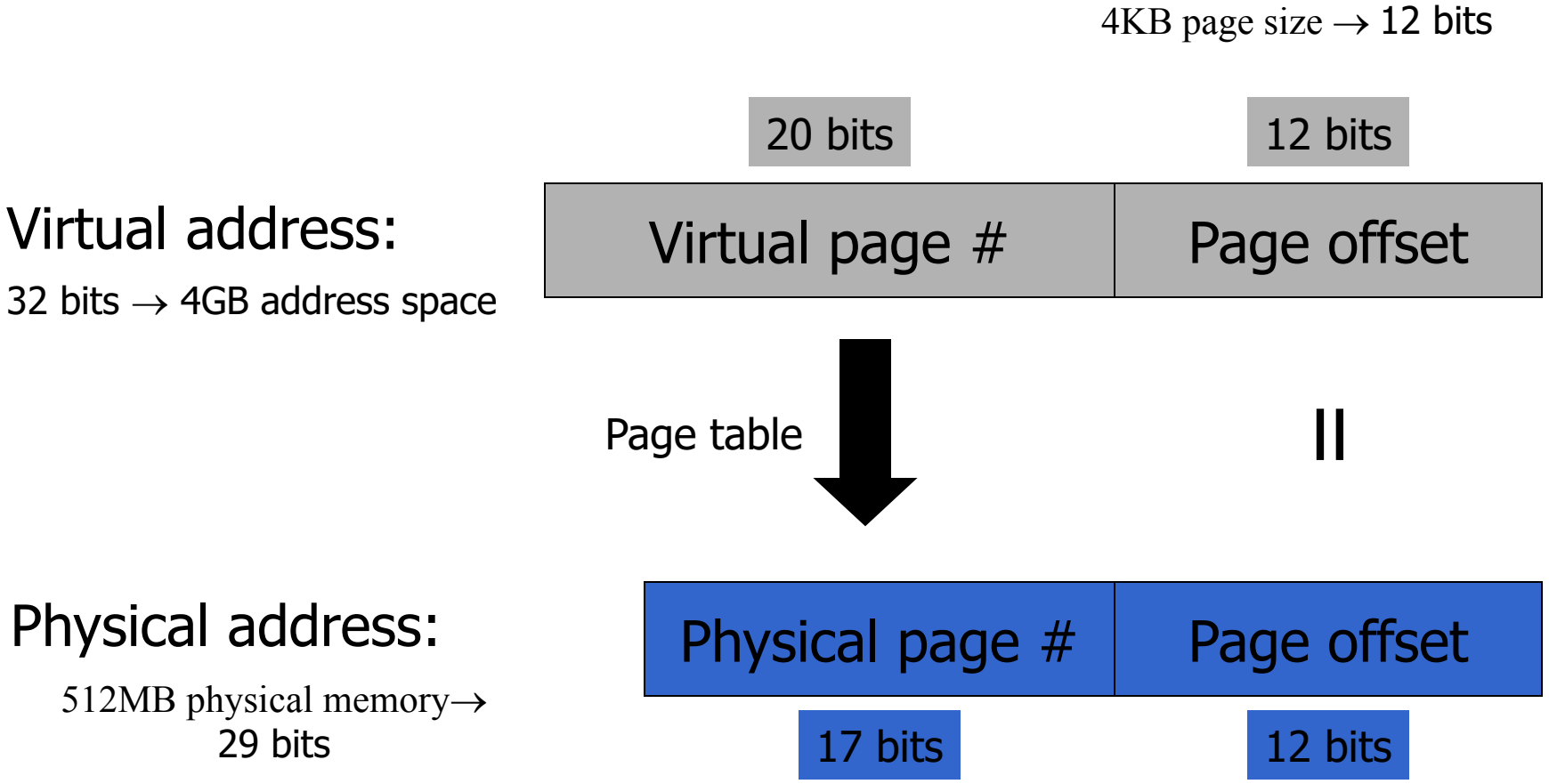


Virtual Memory Terminology

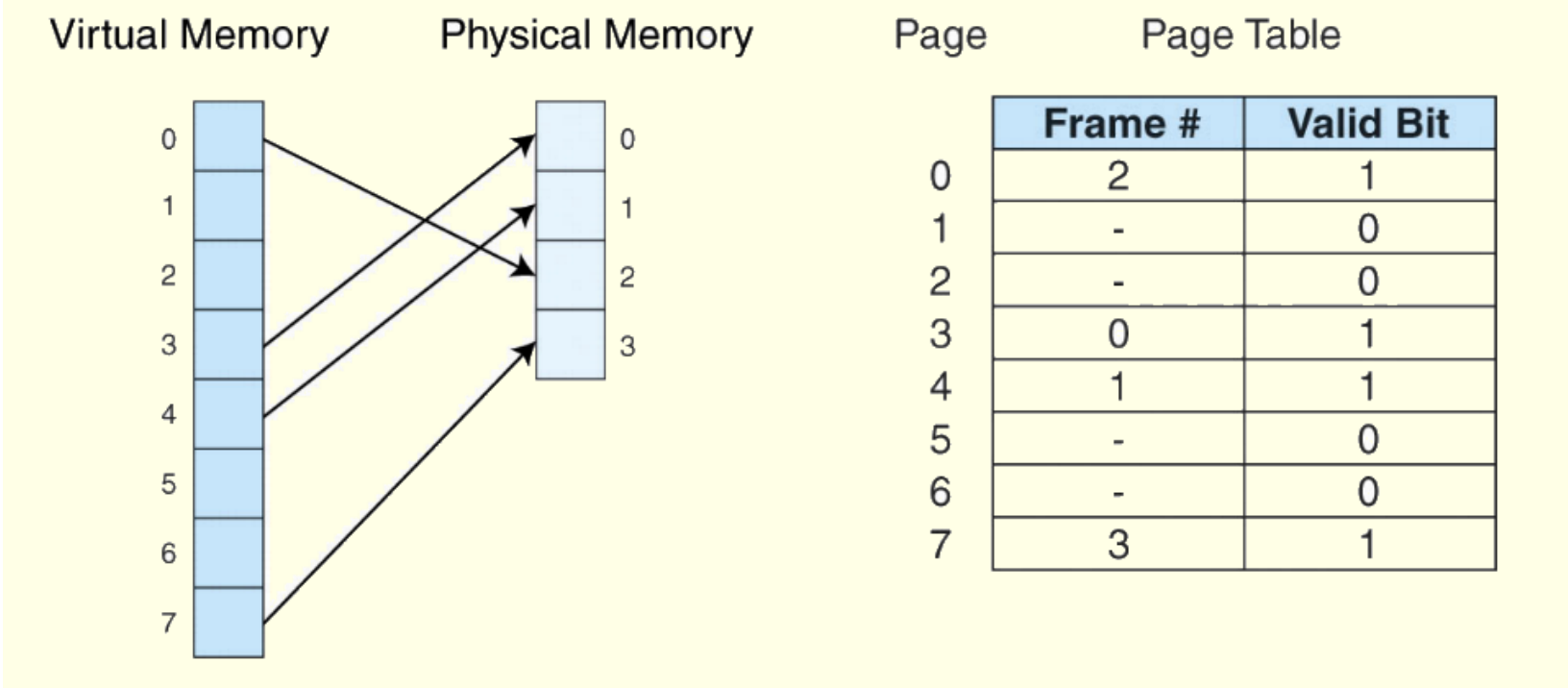
- Page or segment
  - A block transferred from the disk to main memory
    - Page: fixed size
    - Segment: variable size
- Page fault
  - The requested page is not in main memory (miss)
- Address translation or memory mapping
  - Virtual to physical address

Mapping from a Virtual to a Physical Address

4KB page size → 12 bits



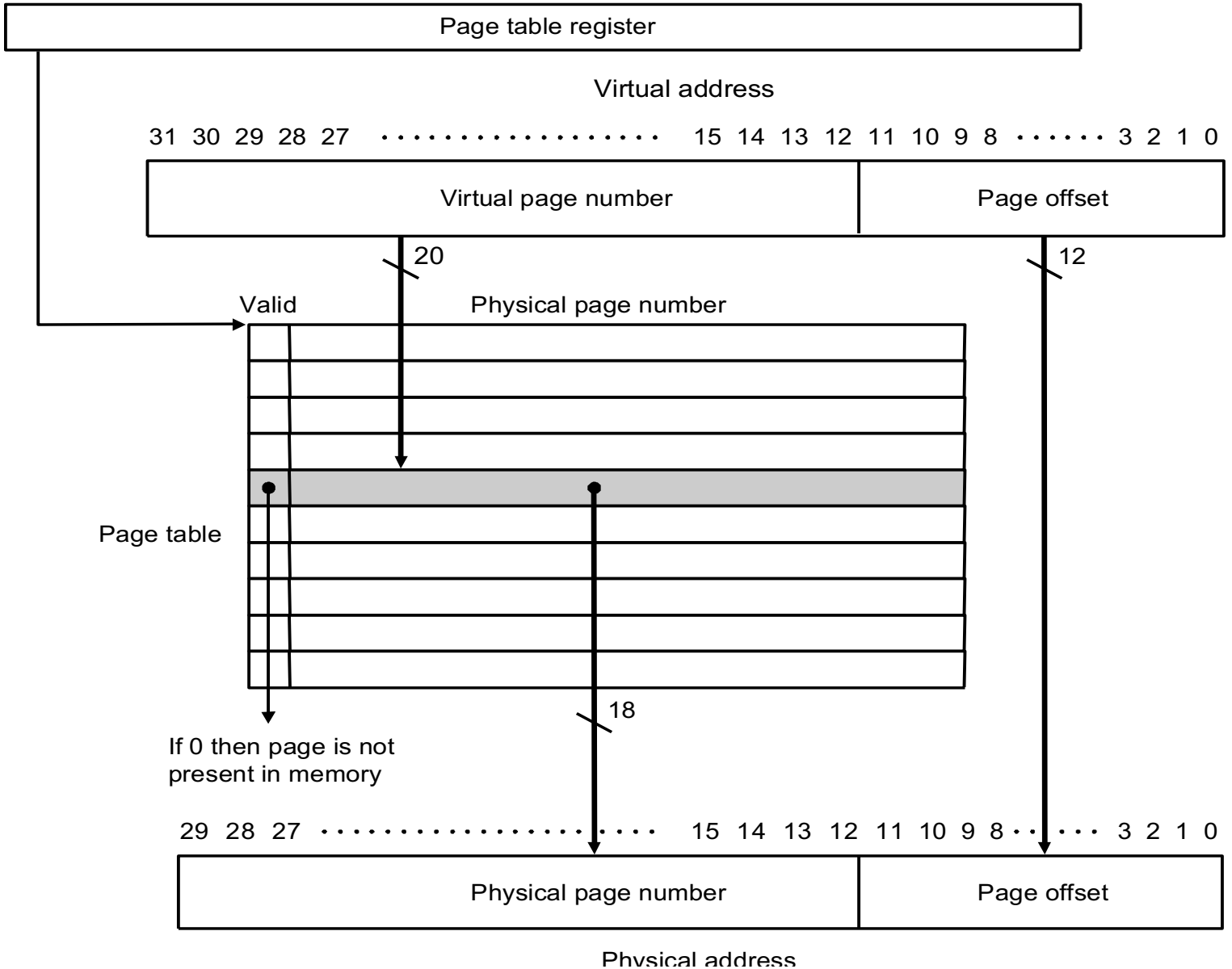
How to Find a Page: Page Table



Page Table

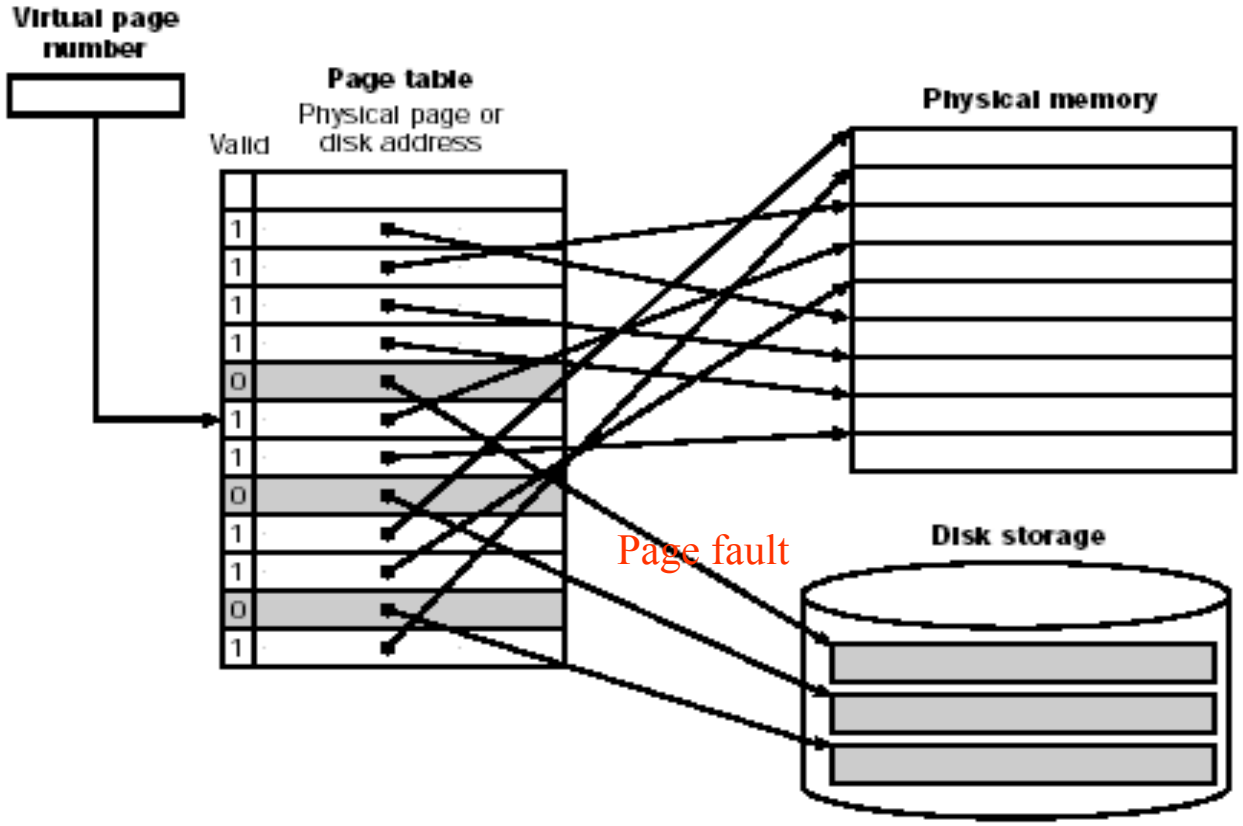
Page	Frame #	Valid Bit
0	2	1
1	-	0
2	-	0
3	0	1
4	1	1
5	-	0
6	-	0
7	3	1

Page Table (cont.)



Page Fault

- If the valid bit for a virtual page is off, a page fault occurs.



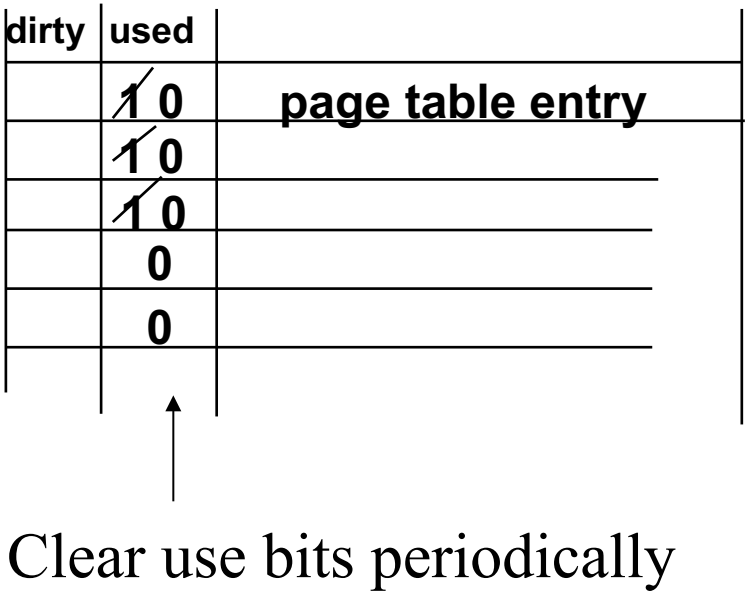
- An exception handler is invoked to handle the page fault
- Find the page in the disk, fetch it, replace a page in the main memory if there is no free page in the main memory

Key Decisions in Paging

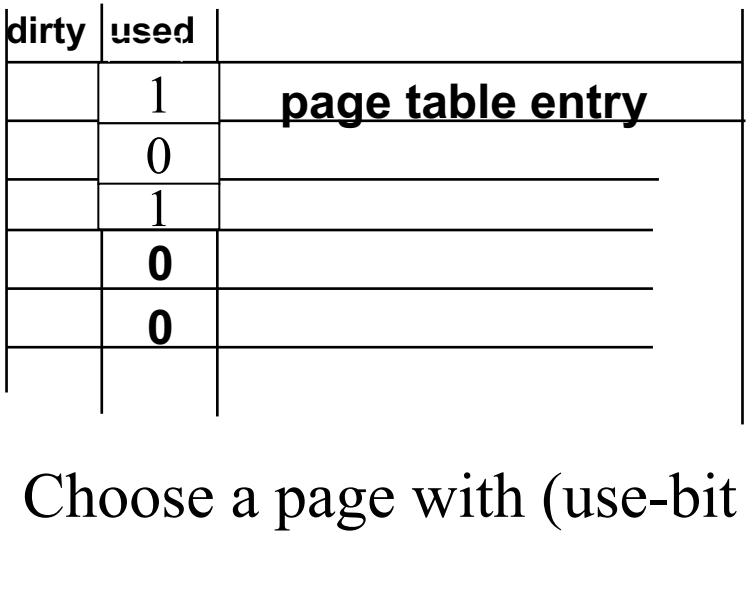
- Huge miss penalty: a page fault may take millions of cycles to process
  - Pages should be fairly large (e.g., 4KB) to amortize the high access time
  - Reducing page faults is important
    - LRU replacement is worth the price
    - fully associative placement
      => use page table (in memory) to locate pages
  - Handle the faults in software instead of hardware, because handling time is small compared to disk access
    - the software can be very smart or complex
    - the faulting process can be context-switched
  - Using write-through is too expensive, so we use write back

Page Replacement: 1-bit LRU

- Associated with each page is a *reference bit (use bit)*:
  - Ref. bit = 1 if page has been referenced in recent past
  - Ref. bit = 0 otherwise
- OS periodically clears the reference bit
- If replacement is necessary, choose any page frame such that its reference bit is 0. This is a page that has not been referenced in the recent past



Clear use bits periodically

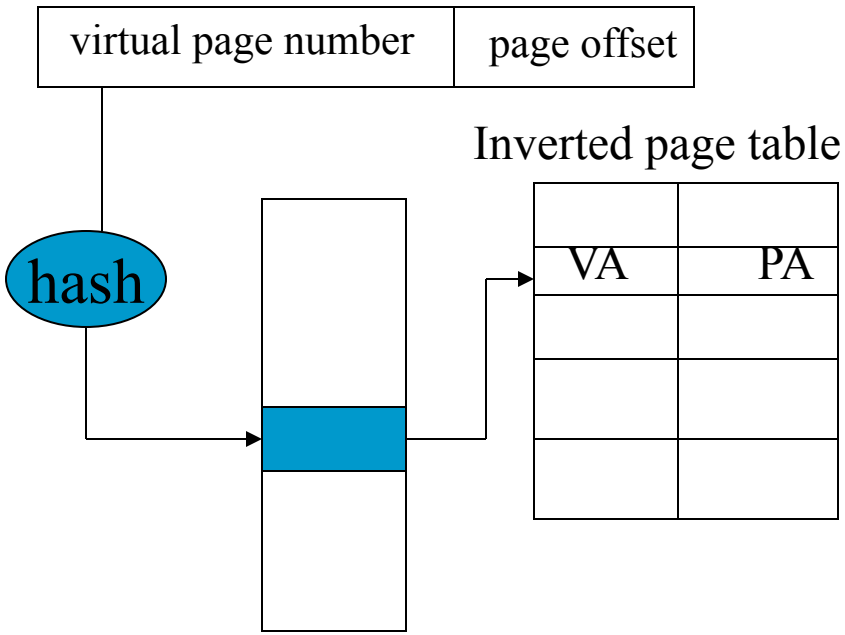


Choose a page with (use-bit = 0)

Problems with Address Translation

- Address translaction table can be very large!
  - **What is the size of the page table given a 32-bit virtual address, 4 KB pages, and 4 bytes per page table entry?**
  - **Solution: inverted page table &two-level**
- Every memory access may need to access the main memory twice
  - **TLB**

Inverted Page Table (HP, IBM)

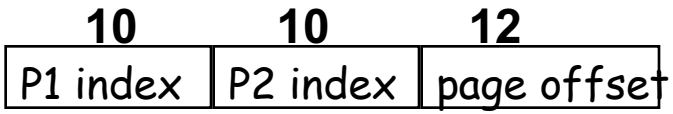


Hash Another Table (HAT)

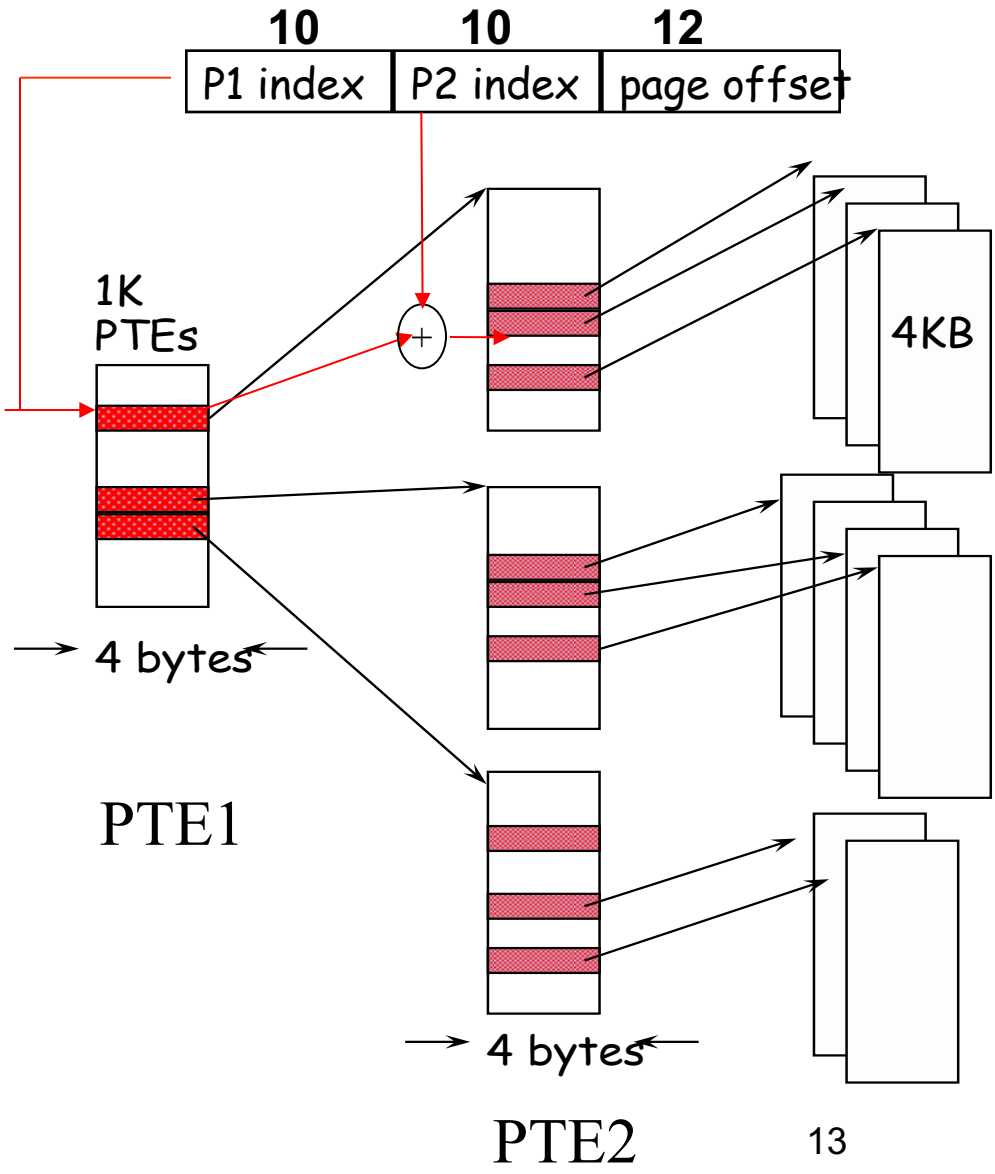
- One PTE per page frame
- Pros & Cons:
  - The size of table is the number of physical pages
  - Must search for virtual address (using hash)

Two-level Page Tables

32-bit address:



- **4GB virtual address space**
- **4 KB of PTE1**
- **4 MB of PTE2**

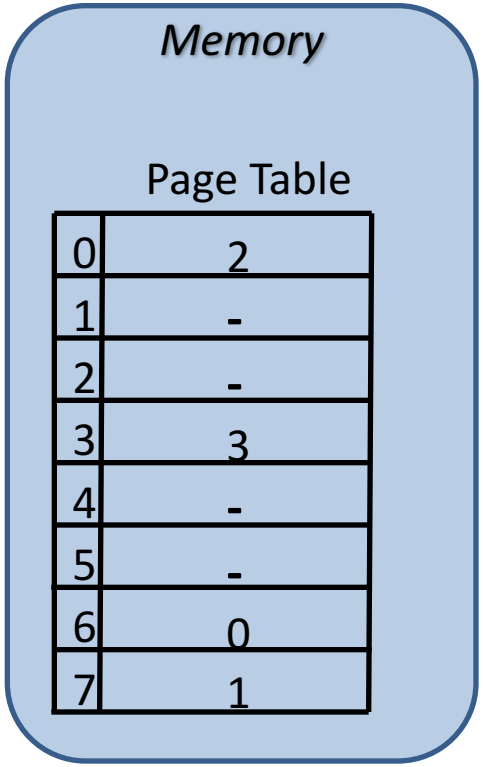


Fast Translation: Translation Look-aside Buffer (TLB)

TLB

tag	Data
0	2
3	3
6	0
7	1

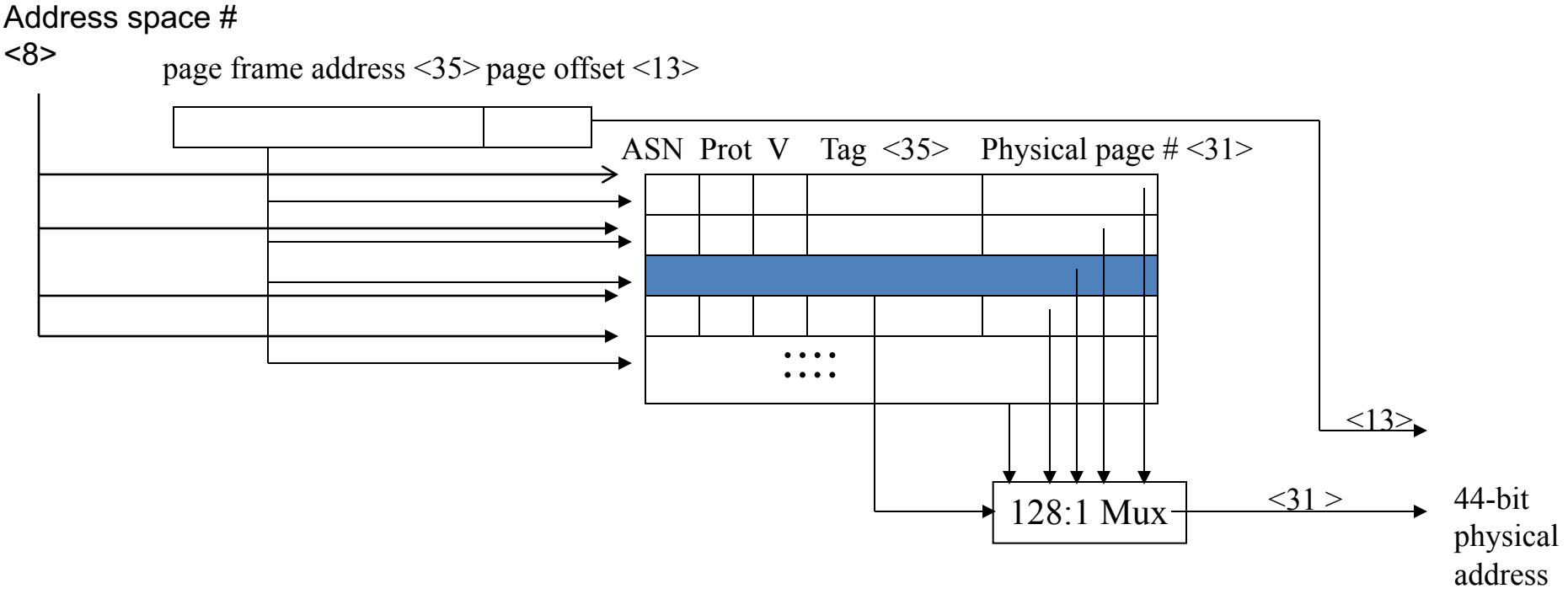
virtual page number **Physical page number**



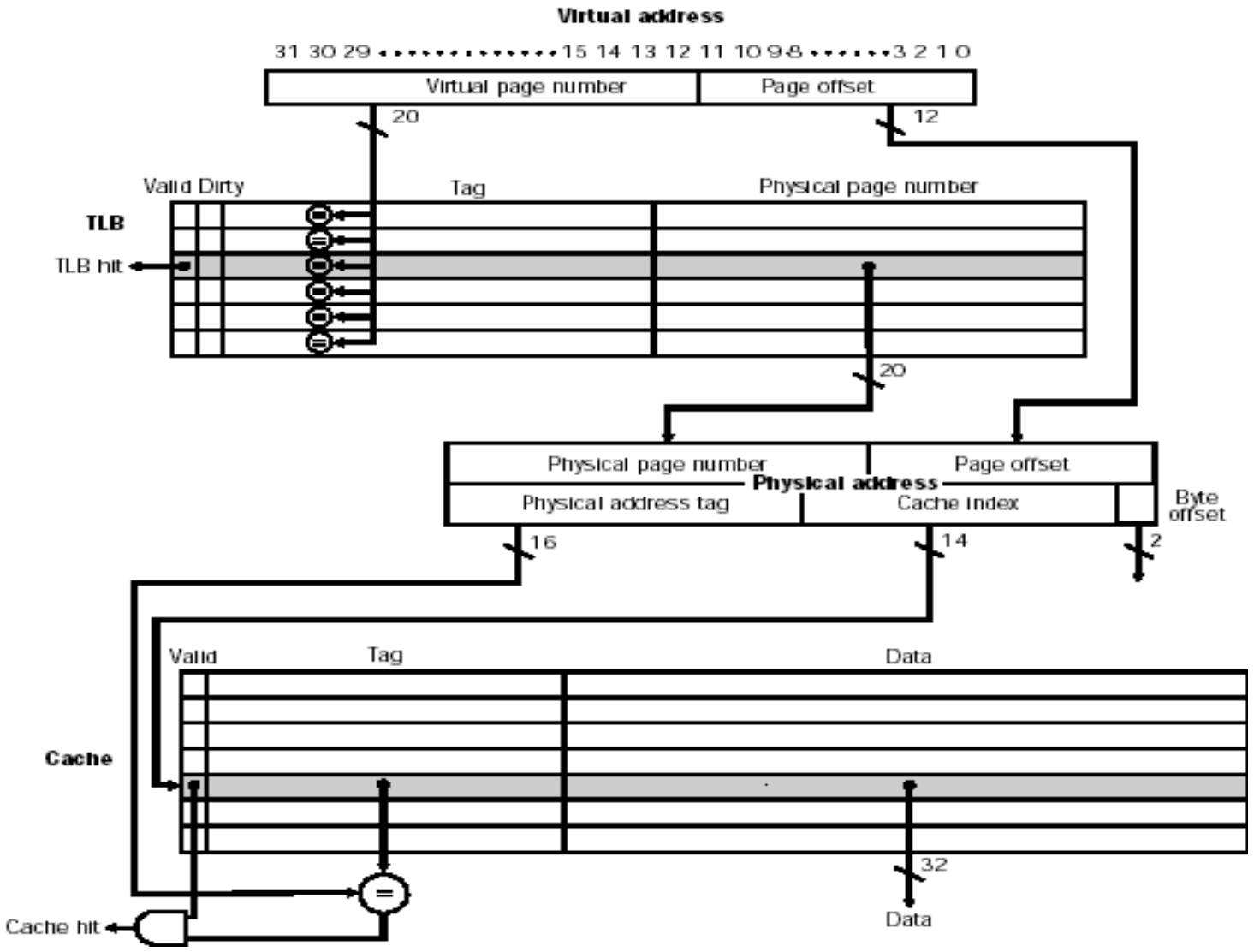
TLB: Cache of translated addresses

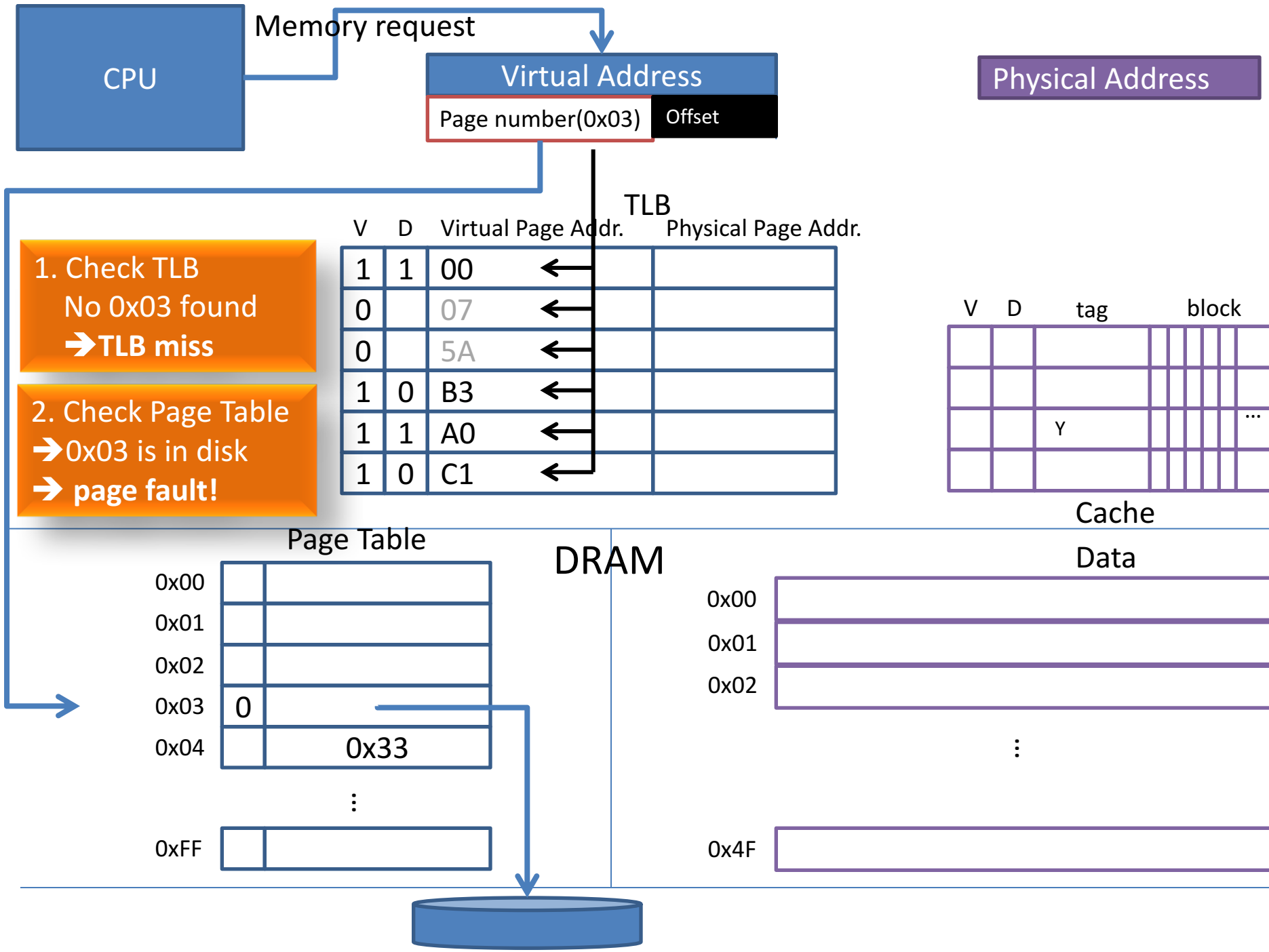
Fast Translation: Translation Look-aside Buffer (TLB)

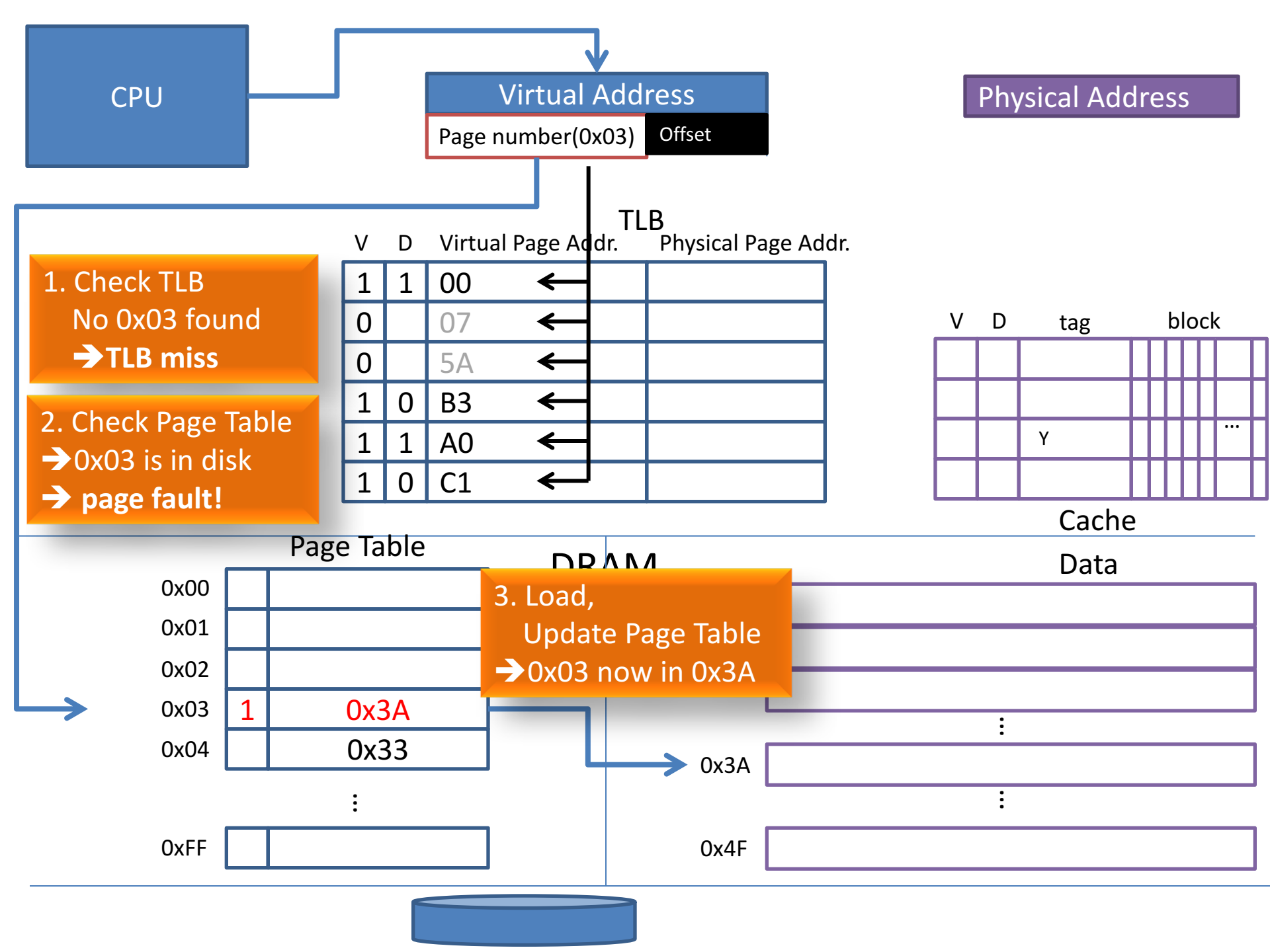
- Example: 48-bit virtual address, 44-bit physical address, 8K page, 128 entry fully associative

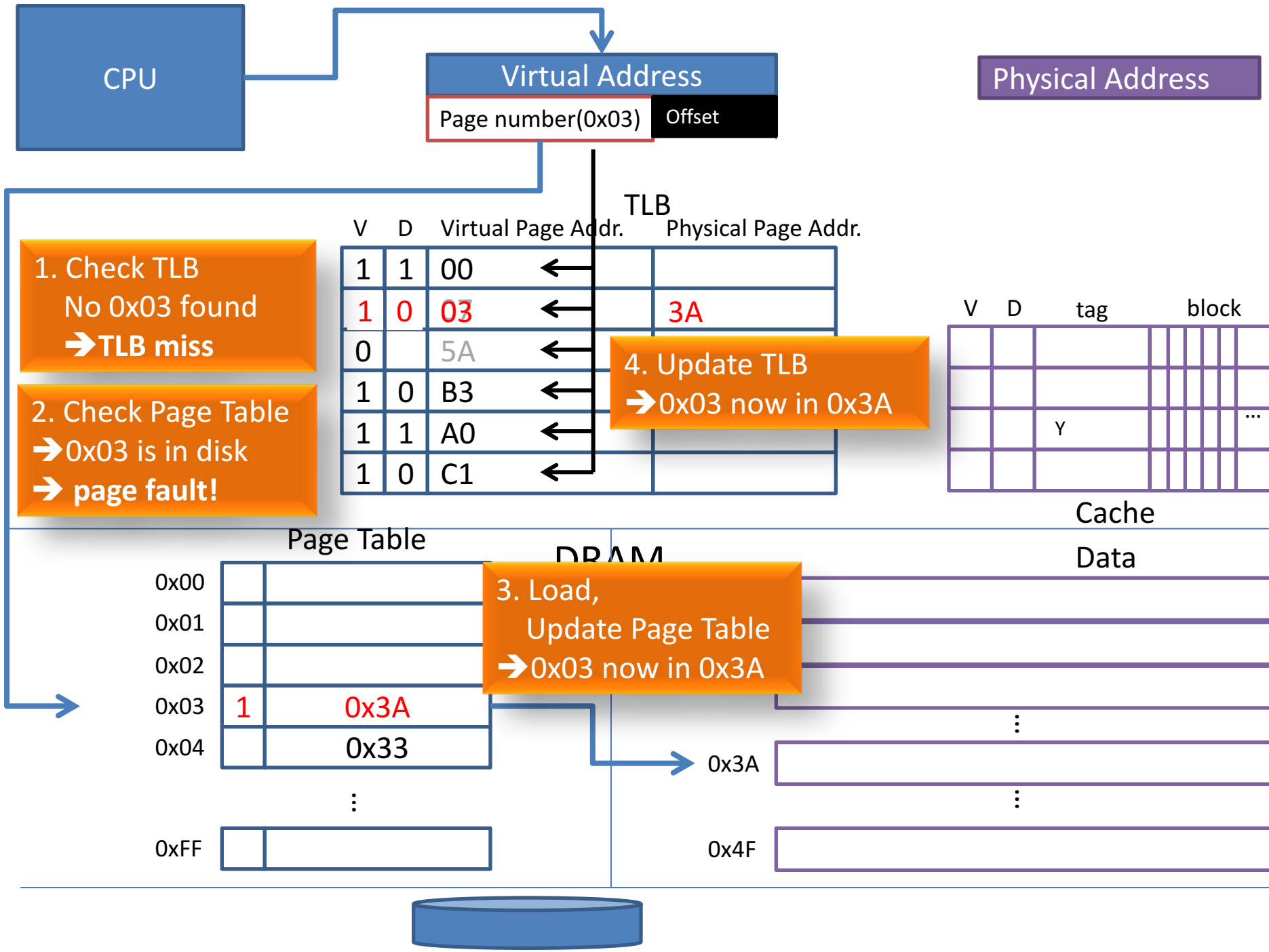


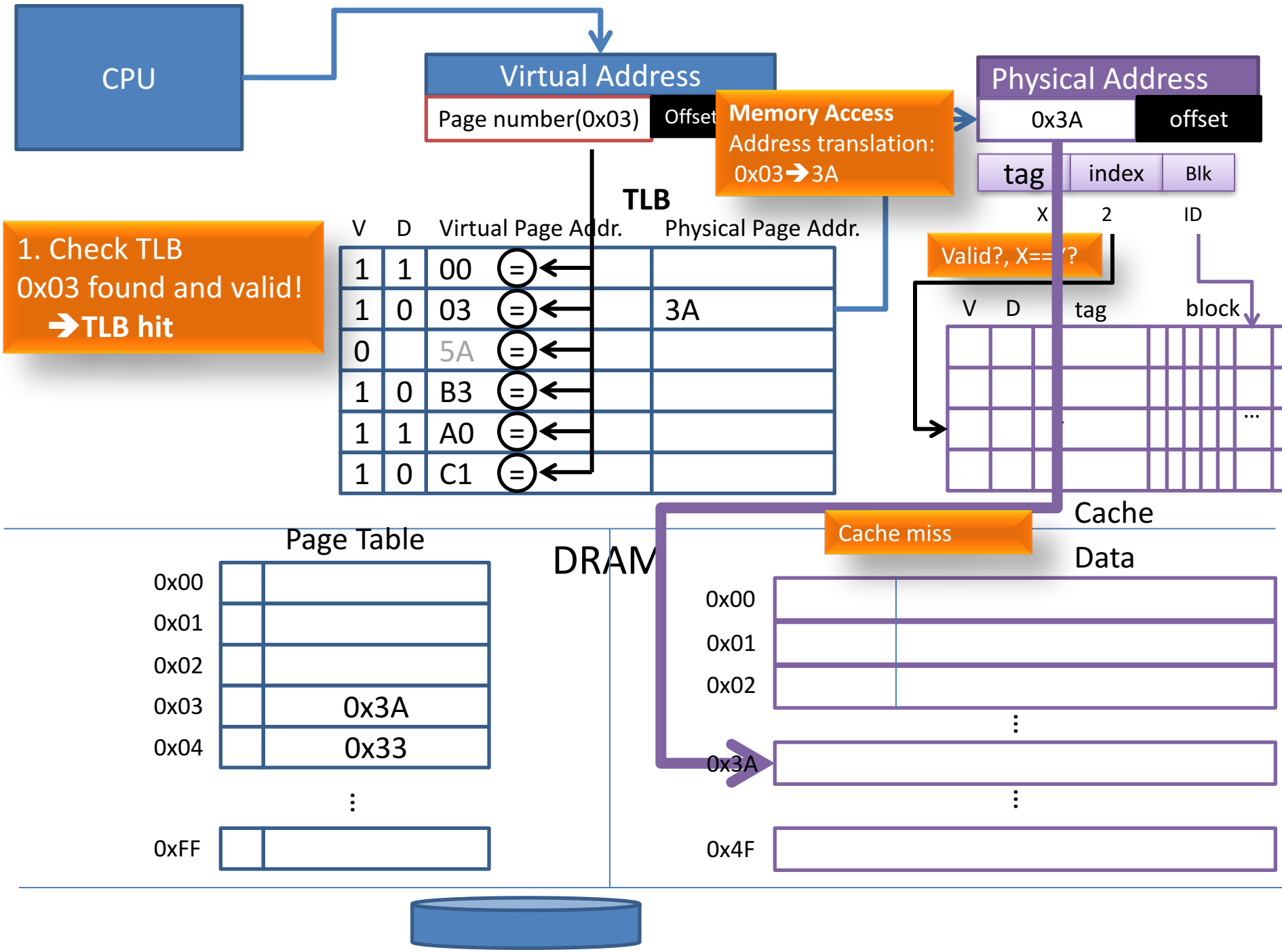
Integrating Virtual Memory, TLBs, and Caches- Intrinsity FastMATH



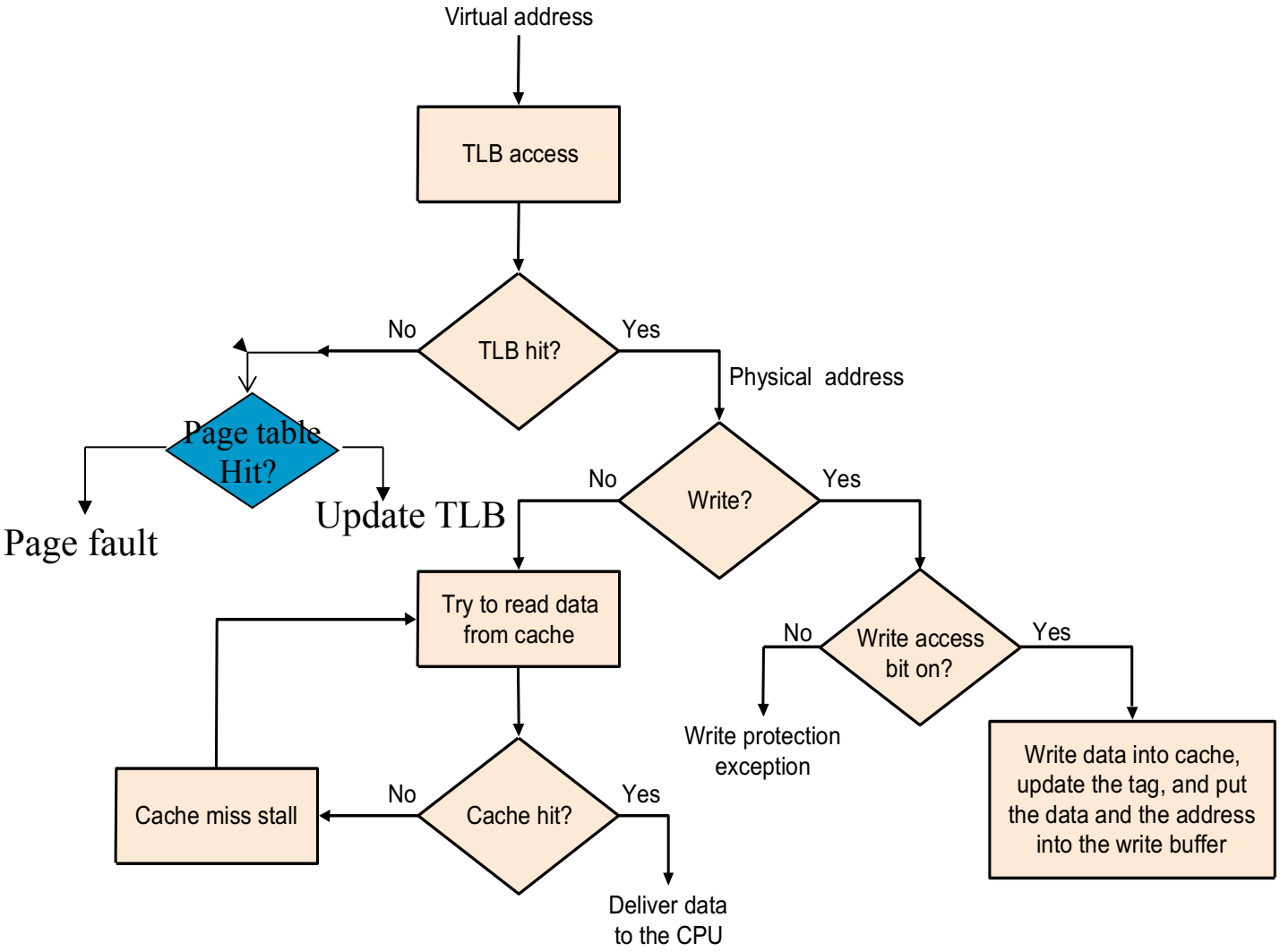

CPU
Memory request
Virtual Address
Page number(0x03)
Offset
Physical Address
TLB
V D Virtual Page Addr. Physical Page Addr.
1 1 00
0 07
0 5A
1 0 B3
1 1 A0
1 0 C1
1. Check TLB
No 0x03 found
➔TLB miss
2. Check Page Table
➔0x03 is in disk
➔ page fault!
V D tag block
Y
...
Cache
DRAM
Page Table
0x00
0x01
0x02
0x03
0
0x04
0x33
0xFF
Data
0x00
0x01
0x02
0x4F


CPU
Virtual Address
Page number(0x03)
Offset
Physical Address
TLB
V D Virtual Page Addr. Physical Page Addr.
1 1 00
0 07
0 5A
1 0 B3
1 1 A0
1 0 C1
1. Check TLB
No 0x03 found
➔TLB miss
2. Check Page Table
➔0x03 is in disk
➔ page fault!
V D tag block
Y
...
Cache
Page Table
DRAM
Data
0x00
0x01
0x02
0x03 1 0x3A
0x04 0x33
0xFF
3. Load,
Update Page Table
➔0x03 now in 0x3A
0x3A
0x4F


CPU
Virtual Address
Page number(0x03)
Offset
Physical Address
TLB
V
D
Virtual Page Addr.
Physical Page Addr.
1 1 00
1 0 03 3A
0 5A
1 0 B3
1 1 A0
1 0 C1
1. Check TLB
No 0x03 found
➔TLB miss
2. Check Page Table
➔0x03 is in disk
➔ page fault!
4. Update TLB
➔0x03 now in 0x3A
V
D
tag
block
Y
...
Cache
Page Table
DRAM
Data
0x00
0x01
0x02
0x03
0x04
0xFF
1
0x3A
0x33
3. Load,
Update Page Table
➔0x03 now in 0x3A
0x3A
0x4F


CPU
Virtual Address
Page number(0x03)
Offset
Memory Access
Address translation:
0x03➔3A
Physical Address
0x3A
offset
tag
index
Blk
X
2
ID
Valid?, X==1?
TLB
V D Virtual Page Addr. Physical Page Addr.
1 1 00
1 0 03 3A
0 5A
1 0 B3
1 1 A0
1 0 C1
1. Check TLB
0x03 found and valid!
➔TLB hit
V D tag block
Cache
Cache miss
DRAM
Page Table
0x00
0x01
0x02
0x03 0x3A
0x04 0x33
0xFF
Data
0x00
0x01
0x02
0x3A
0x4F

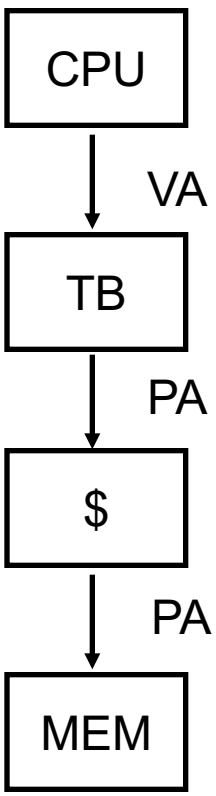
Processing in TLB+Cache



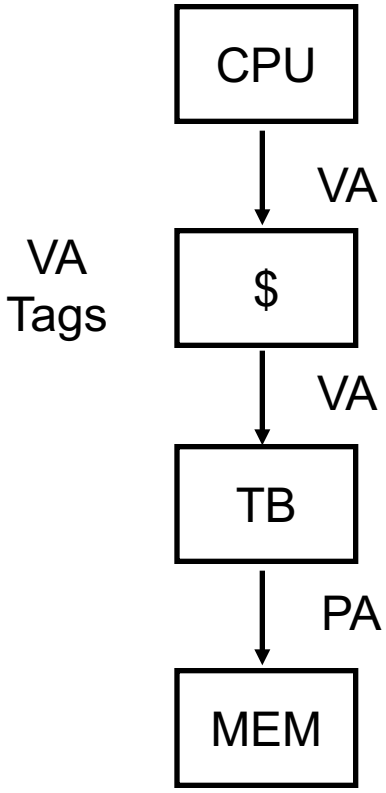
Possible Combinations of Events

TLB	Page table	Cache	Possible? If so, under what situations
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but retry found in page table; after retry, data found in cache.
Miss	Hit	Miss	TLB misses, but retry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data misses in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible; cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

TLB & Caches



Conventional Organization

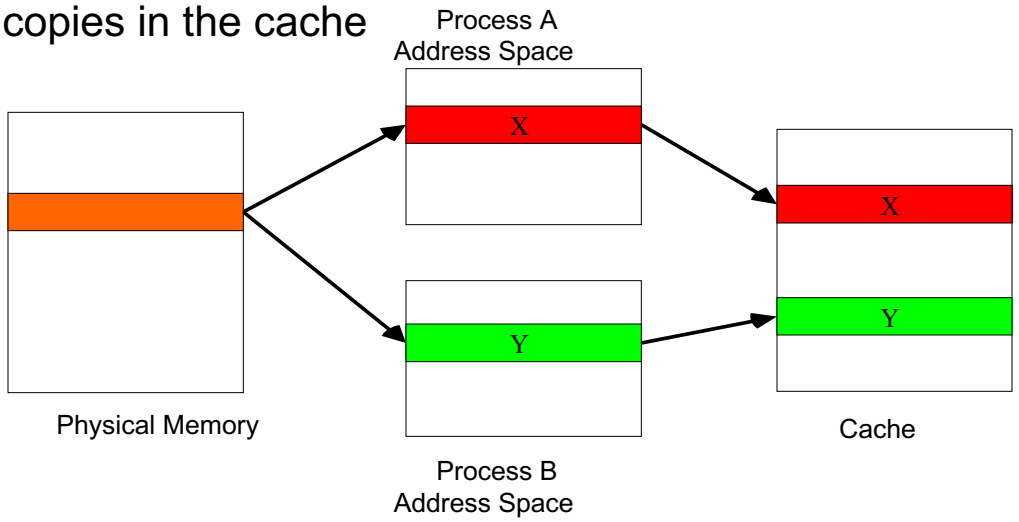


Virtually Addressed Cache
Translate only on miss

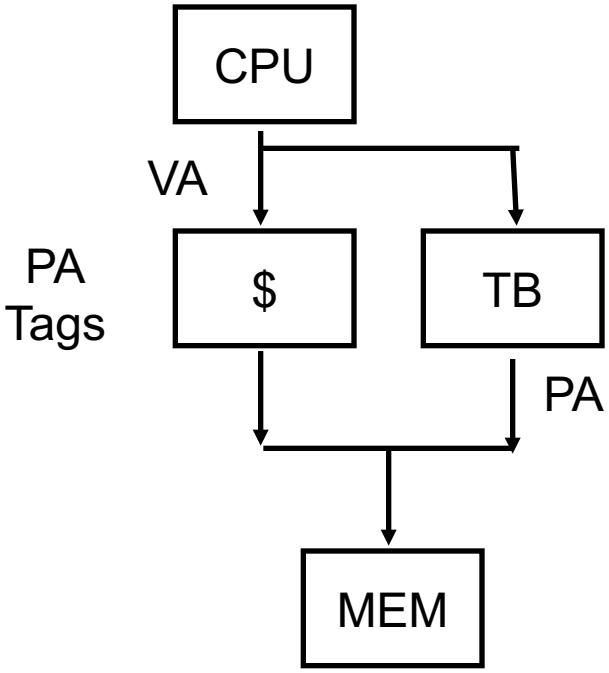
Problem: long cache access latency

Virtual Cache

- Advantage:
  - Avoid address translation before accessing cache
    - Faster hit time
- Problems:
  - How do we distinguish data of different processes?
    - Flush the cache during context switch
    - Add processor id (PID) to cache
  - I/O (physical address) must interact with cache
    - Physical -> virtual address translation
  - Aliases (Synonyms)
    - Two virtual addresses map to the same physical addresses
      - Two identical copies in the cache

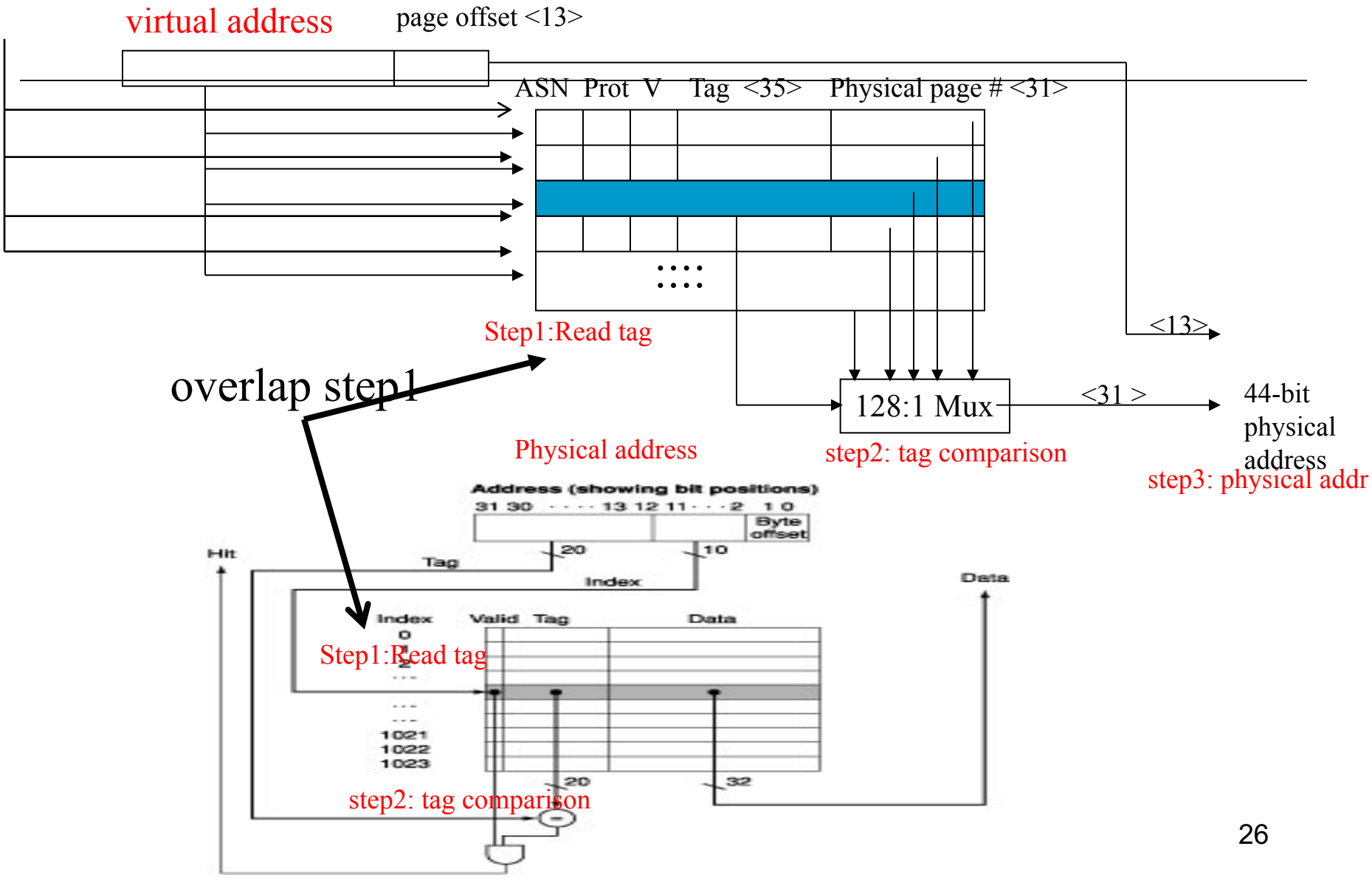


Virtually indexed & physically tagged cache



Overlap the time to
read the tags with address translation

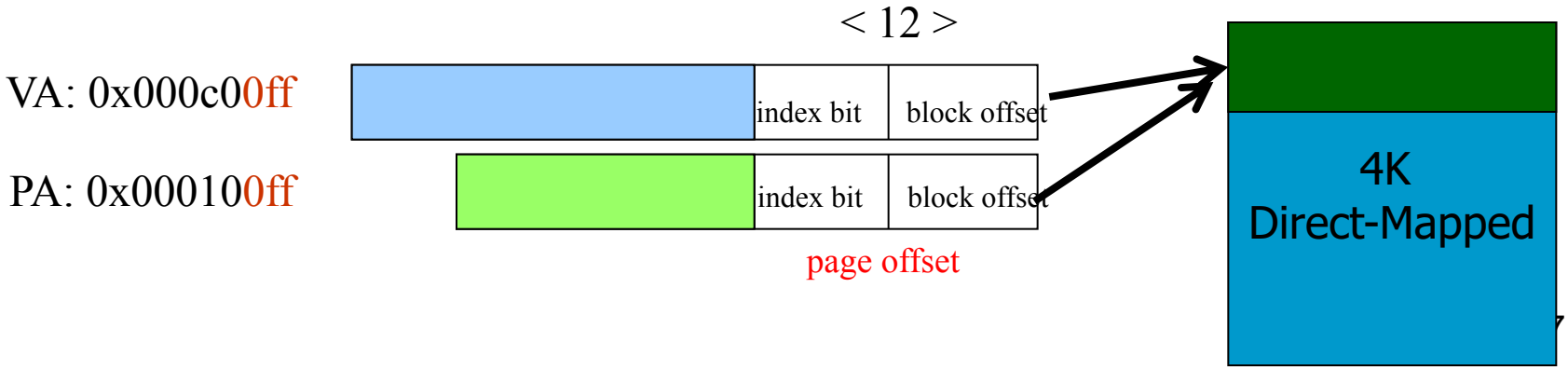
- It is actually a physical cache but we can find the set using a virtual address



- Use the part of addresses that is not affected by address translation to index a cache
  - Which part of an address can we use?

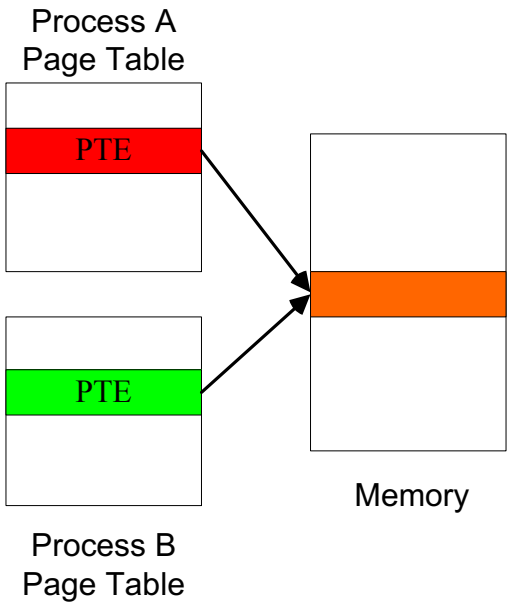
Virtually indexed & physically tagged cache (cont.)

1. What is the size of index & block-offset for a 4K direct-mapped cache?
2. Do two addresses with the same lower 12 bits map to the same set for a 4K direct-mapped cache?
3. Assume a 4K page. The lower 12 bits of a virtual address is the same as its physical address.
   -- We can find the cache set using the virtual address
4. If page size = 4K, what is the size limitation on a direct-mapped, virtually indexed, physically tagged cache?



Implementing Protection with Virtual Memory

- How does the Virtual Memory provide memory protection among processes?
  - Hardware support
    - Provide two modes – User vs. Kernel (Mode transition through System call and ERET)
    - Only processes in the Kernel mode can write to the processor states, such as page table pointer and tlb, through special instructions
    - Write access bit in the TLB to protect a page from being written
  - Put page tables in the addressing space of OS
    => user process cannot modify its own PT and can only use the storage given by OS
- Sharing: P2 asks OS to create a PTE for a virtual page in P1's space, pointing to page to be shared



TLB Misses & Page Faults

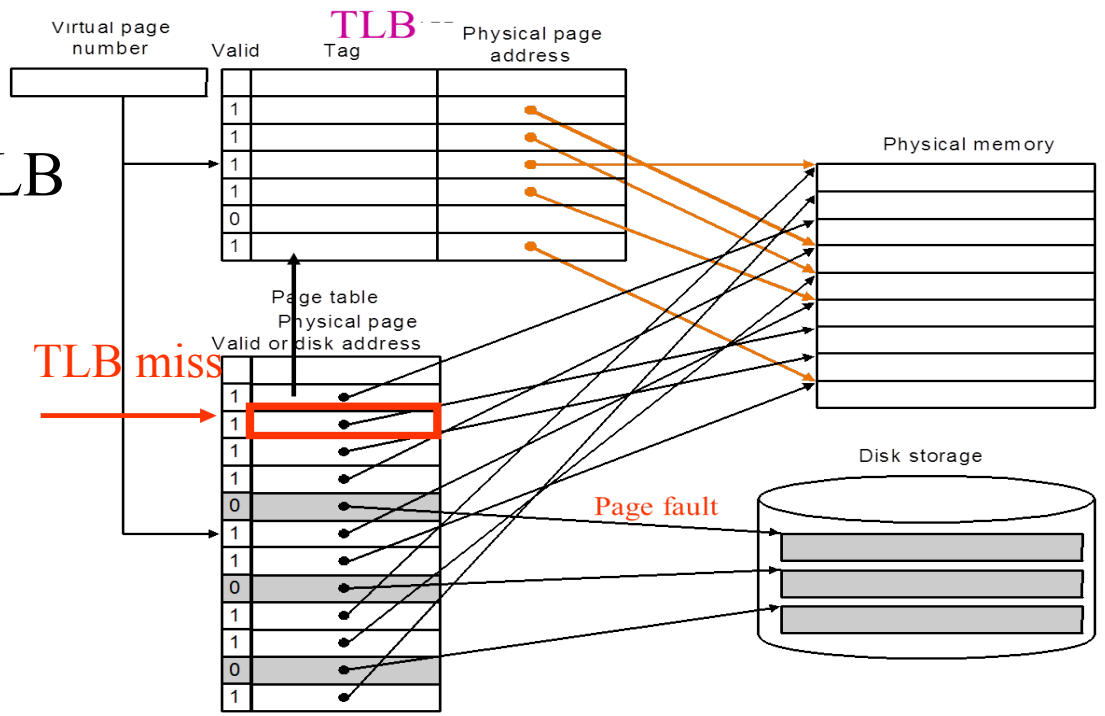
TLB miss

↓

is the page present in memory?

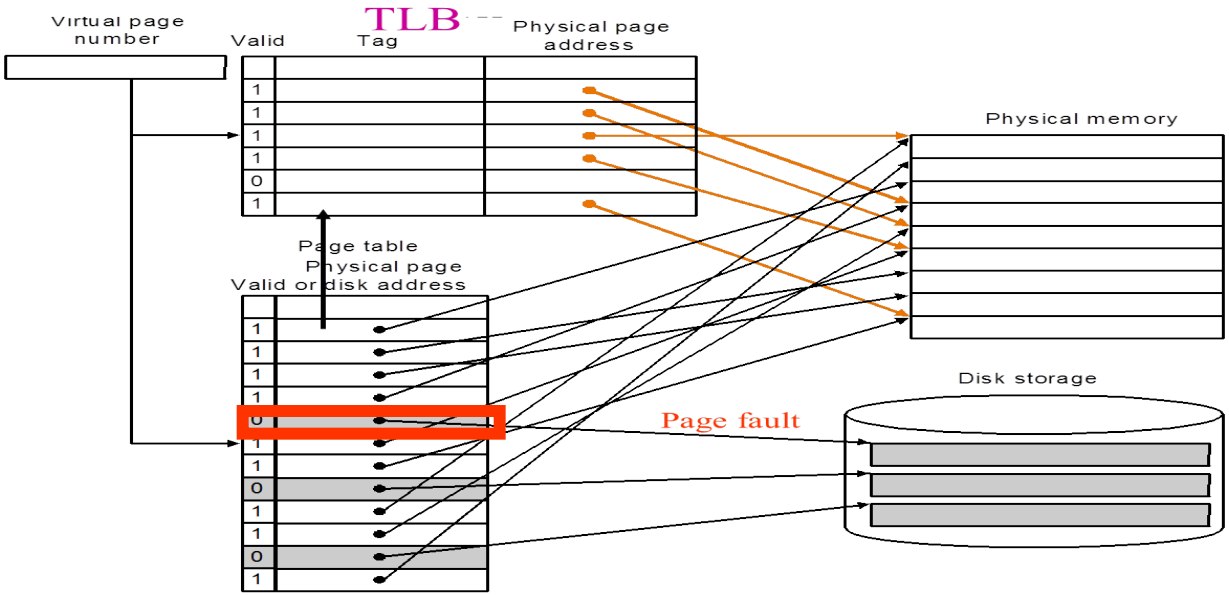
no → page fault → ↓

yes → update the TLB



15

Page Fault



Exception entry point – 0x80000180 & Cause register

Save process states

(a) Lock up the page table entry using the virtual address and find the location of the referenced page on disk

(b) Choose a physical page to replace (invalidate the corresponding entries in TLB and caches) if the page is dirty, it must be written out to disk first

(c) Fetch the page from disk into memory

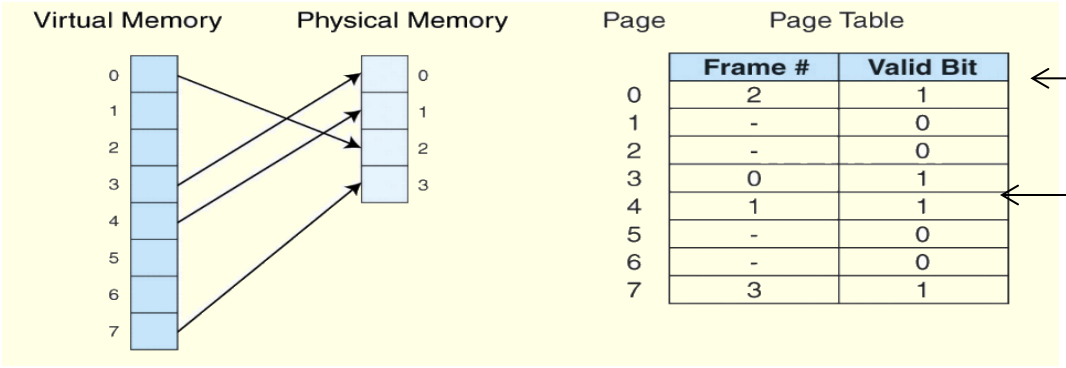
Restore States and Exception Return

Software TLB Miss Handler in MIPS

- Save the page # of the faulting reference in BadVAddr
- Invoke TLB miss handler at 0x80000000
  - HW puts the page table address and the virtual address into Context register
    - Upper 12 bits – page table address
    - Next 18 bits – virtual page number of the missing page
    - Size of PTE = 1 word

```
mfc0  $k1, Context  # copy address of PTE into temp $k1
lw    $k1, 0($k1)   # put PTE into temp $k1
mtc0  $k1, EntryLo # put PTE into special register EntryLo
tlbwr                # put EntryLo into TLB entry at Random
eret                  # return from TLB miss exception
```

- Performance optimization in handling TLB miss
  - Special exception entry point (0x800000000 instead of 0x80000180)
  - Do not check if the PTE is valid



Performance issue in Virtual Memory

- Thrashing
  - Frequent swapping pages between memory and disks
  - Solutions
    - Buy more memory
    - Reexamine the algorithm and data structures to reduce the working set of pages
- High TLB misses
  - Since TLB is small (32-64 entries), high TLB miss rate is expected.
  - Solutions
    - Variable page size
      - MIPS : 16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB, 256MB
  - Reexamine the algorithm and data structures to reduce the working set of pages.

Key Design Parameter in Memory Hierarchy

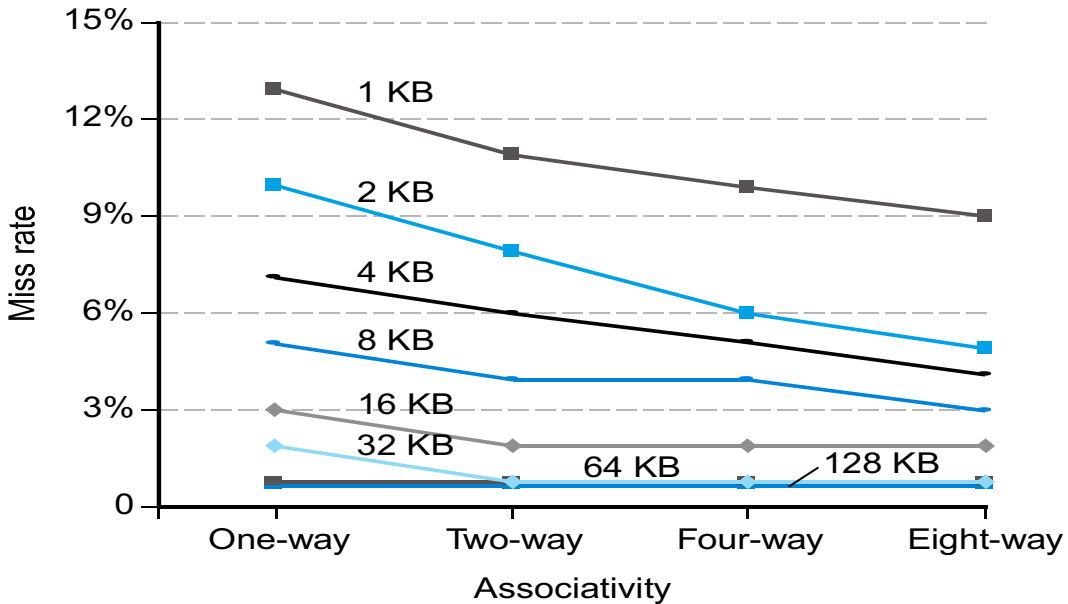
Feature	Typical values for L1 caches	Typical values for L2 caches	Typical values for paged memory	Typical values for a TLB
Total size in blocks	250-2000	4000-250,000	16,000-250,000	16-512
Total size in kilobytes	16-64	500-8000	250,000-1,000,000,000	0.25-16
Block size in bytes	32-64	32-128	4000-64,000	4-32
Miss penalty in clocks	10-25	100-1000	10,000,000-100,000,000	10-1000
Miss rate (global for L2)	2%-5%	0.1%-2%	0.00001%-0.0001%	0.01%-2%

A Common Framework for Memory Hierarchies

- Four questions for memory hierarchy:
  - Where can a block be placed in upper level?
    - Block placement: one place (direct mapped), a few places (set associative), or any place (fully associative)
  - How is a block found if it is in the upper level?
    - Block identification: indexing, limited search, full search, lookup table
  - Which block should be replaced on a miss?
    - Block replacement: LRU, random
  - What happens on a write?
    - Write strategy: write through or write back

Q1: Where can a Block be placed

Scheme name	No. of sets	Blocks/set
Direct mapped	No. of blocks in cache	1
Set associative	No. of blocks in cache ---------- Associativity	Associativity (2-16)
Fully associative	1	No. of blocks in cache



Q2: How is a Block Found?

Associativity	Location method	Comparisons required
Direct mapped	Index	1
Set associative	Index the set, search among elements	Degree of associativity
Full	Search all cache entries	Size of the cache
	Separate lookup table	0

- Virtual memory systems almost use fully associative placement.
- Set-associative placement is often used for caches and TLB.
- Direct-mapped caches have less access time and simplicity.

Q3: Which Block Should Be Replaced on a Cache Miss

- In a fully associative cache, all blocks are candidates for replacement.
- In a set associative cache, the blocks in the set are candidates for replacement.
- In a directed-mapped cache, there in only one candidate.
- Primary strategies for replacement in set-associative or fully associative caches:
  - Random
  - Least recently used (LRU)

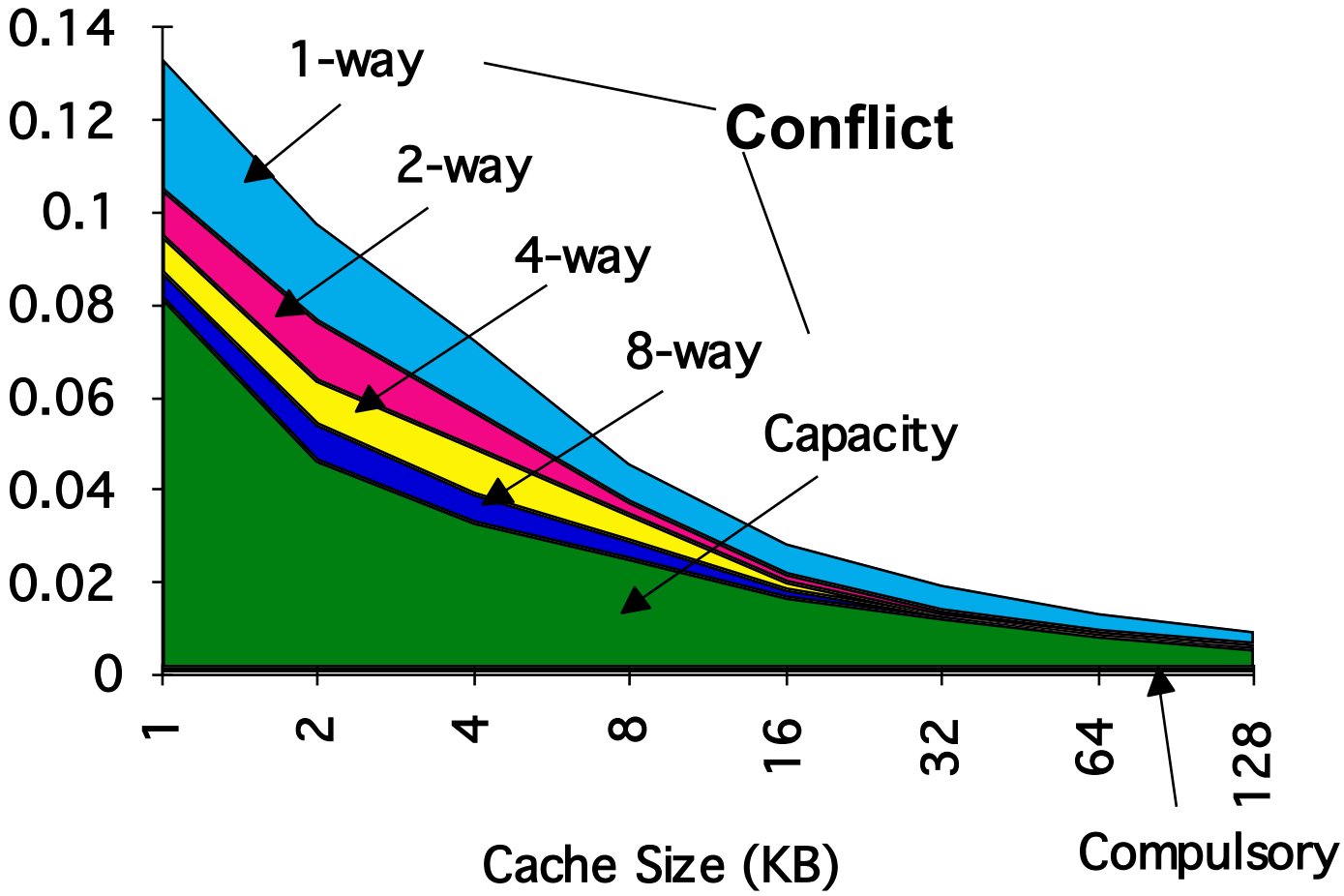
Q4: What Happens on a Write?

- Write-through: The information is written to both the block in the cache and to the block in the lower level of the memory hierarchy.
  - Misses are simpler and cheaper.
  - Easier to implement.
- Write-back: The information is written only to the block in the cache. The modified block is written to the lower level of the hierarchy only when it is replaced.
  - Individual words can be written by the processor at the rate of the cache.
  - Multiple writes within a block require only one write action.
  - The system can use high bandwidth transfer.

Three Cs

- Classifying Misses: 3 Cs
  - *Compulsory*—The first access to a block is not in the cache, so the block must be brought into the cache. Also called *cold start misses* or *first reference misses*.
    (Misses in even an Infinite Cache)
  - *Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
    (Misses in Fully Associative Size X Cache)
  - *Conflict*—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called *collision misses* or *interference misses*.
    (Misses in N-way Associative, Size X Cache)

3Cs Absolute Miss Rate



Challenge in Memory Hierarchy

- Every change that potentially improves miss rate can negatively affect overall performance

Design change	Effects on miss rate	Possible negative effects
size ↑	**capacity miss ↓**	**access time ↑**
associativity ↑	**conflict miss ↓**	**access time ↑**
block size ↑	**spatial locality ↑**	**miss penalty ↑**

- Trends:
  - Redesign DRAM chips to provide higher bandwidth or processing
  - Use prefetching & non-blocking cache (make cache visible to ISA)
  - Restructure code to increase locality

Techniques to reduce miss penalty

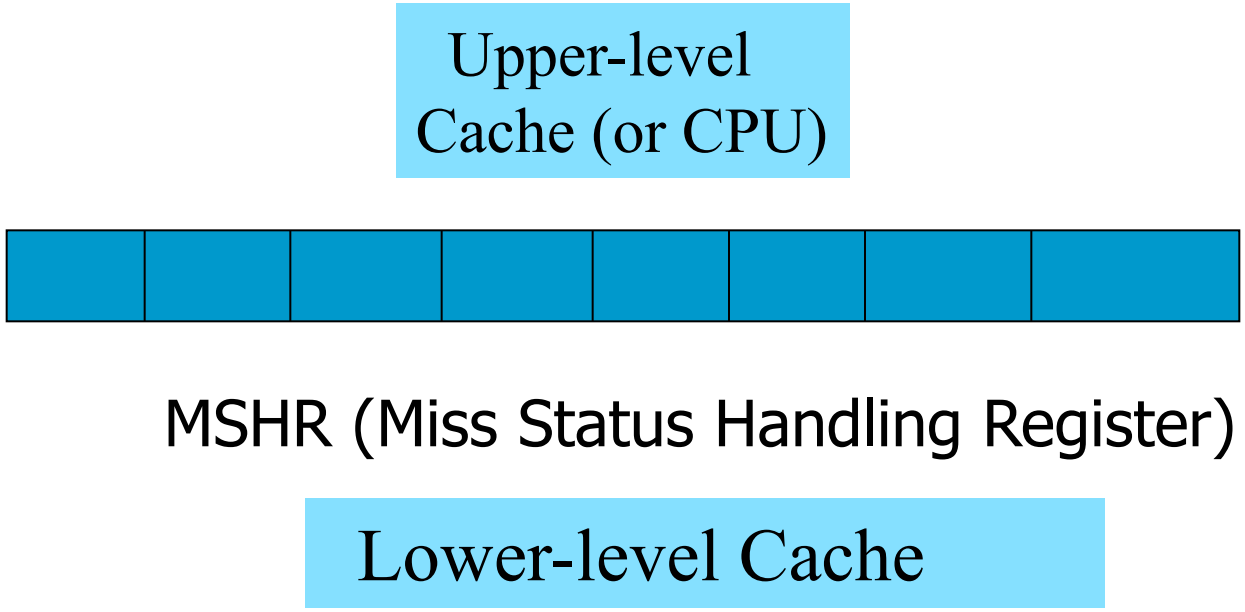
- Non-blocking caches
- Prefetching
- Third-level caches

Non-blocking Caches to reduce stalls on misses

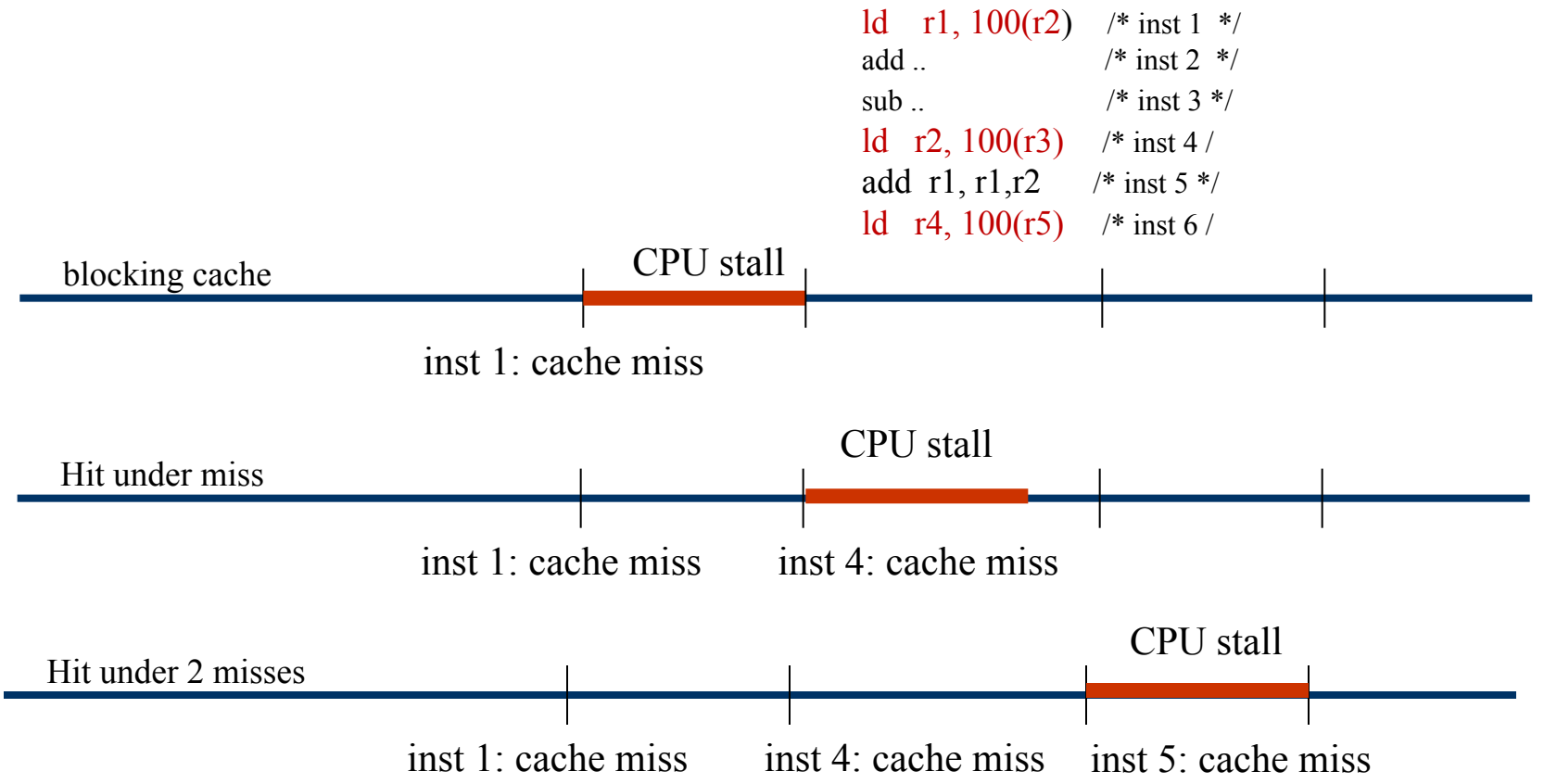
- *Non-blocking cache* or *lockup-free cache* allowing the data cache to continue to supply cache hits during a miss
- "*hit under miss*" reduces the effective miss penalty by being helpful during a miss instead of ignoring the requests of the CPU
- "*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses

Lock-up free Cache Architecture

- **LOCKUP-FREE INSTRUCTION FETCH/PREFETCH CACHE ORGANIZATION -** DAVID KROFT



Non-Blocking Cache



Cache performance:

A cache miss does not necessarily stall the CPU

Avoiding a cache miss that is completely hidden does not help performance

Pre-fetch

```
ld    r1, 100(r2)    /* inst 1 : address q = 100+[r2]  */
:
:
add  r1, r1,r2      /* inst 2 */
```

Pre-fetching = Requesting data early, so it's in cache when needed.

Pre-fetching (cont.)

```
For (i = 0; i< 100; i++)
        prefetch (a[i+4]);
        a[i] = a[i]+ 8
```

Two basic approaches:

Static: Compiler inserts load instructions into code.

Dynamic: Hardware looks ahead in code for memory accesses.

- Miss penalty determines how early to pre-fetch.
- Processor may be able to load into cache w/o loading into register.
- Safe to pre-fetch speculatively.

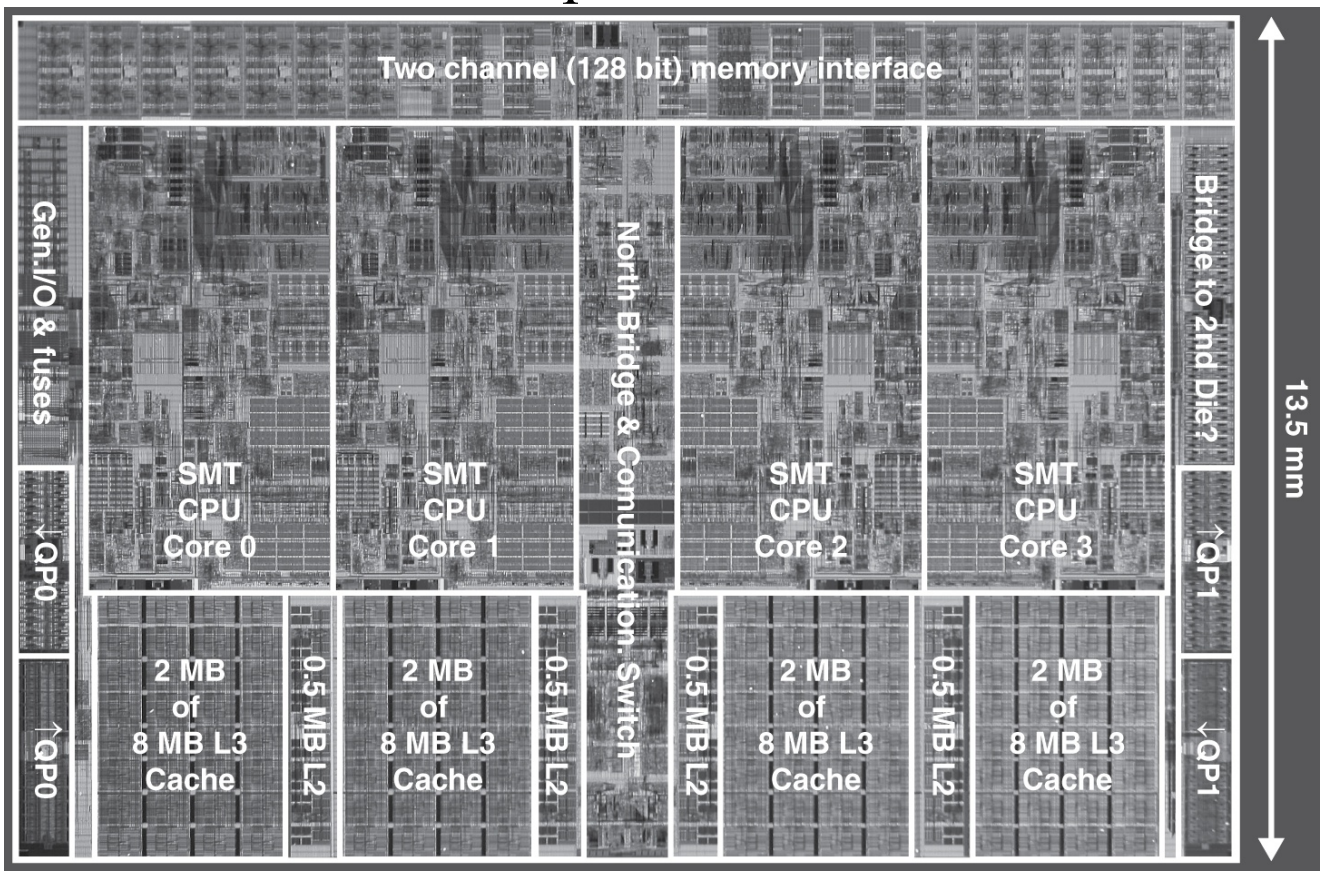
Problem: May replace data in cache that is still needed.

Third-level Caches

- Intel Pentium Xeon
  - 1 MB, L3 cache
- Intel Pentium 4 Extream Edition
  - 2MB, L3 cache

Multilevel On-Chip Caches

Intel Nehalem 4-core processor



Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache

2-Level TLB Organization

	Intel Nehalem	AMD Opteron X4
Virtual addr	48 bits	48 bits
Physical addr	44 bits	48 bits
Page size	4KB, 2/4MB	4KB, 2/4MB
L1 TLB (per core)	L1 I-TLB: 128 entries for small pages, 7 per thread (2×) for large pages L1 D-TLB: 64 entries for small pages, 32 for large pages Both 4-way, LRU replacement	L1 I-TLB: 48 entries L1 D-TLB: 48 entries Both fully associative, LRU replacement
L2 TLB (per core)	Single L2 TLB: 512 entries 4-way, LRU replacement	L2 I-TLB: 512 entries L2 D-TLB: 512 entries Both 4-way, round-robin LRU
TLB misses	Handled in hardware	Handled in hardware

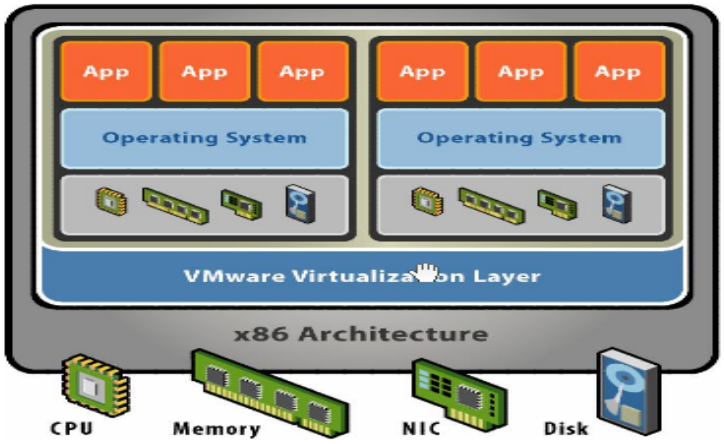
3-Level Cache Organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

n/a: data not available

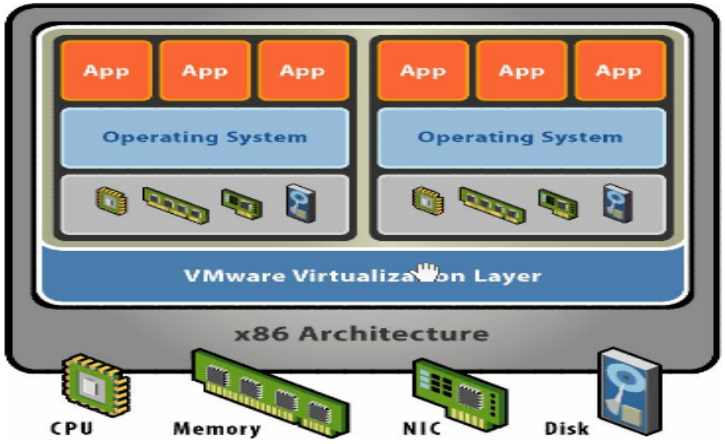
Virtual Machines

- Host computer emulates guest operating system and machine resources
  - Improved isolation of multiple guests
  - Avoids security and reliability problems
  - Aids sharing of resources
- Virtualization has some performance impact
  - Feasible with modern high-performance computers
- Examples
  - IBM VM/370 (1970s technology!)
  - VMWare
  - Microsoft Virtual PC



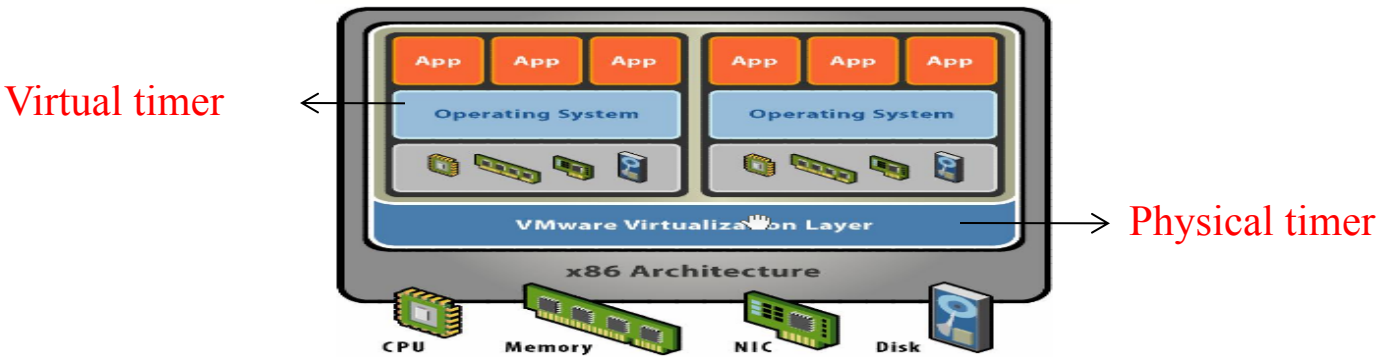
Virtual Machine Monitor

- Maps virtual resources to physical resources
  - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
  - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
  - Emulates generic virtual I/O devices for guest



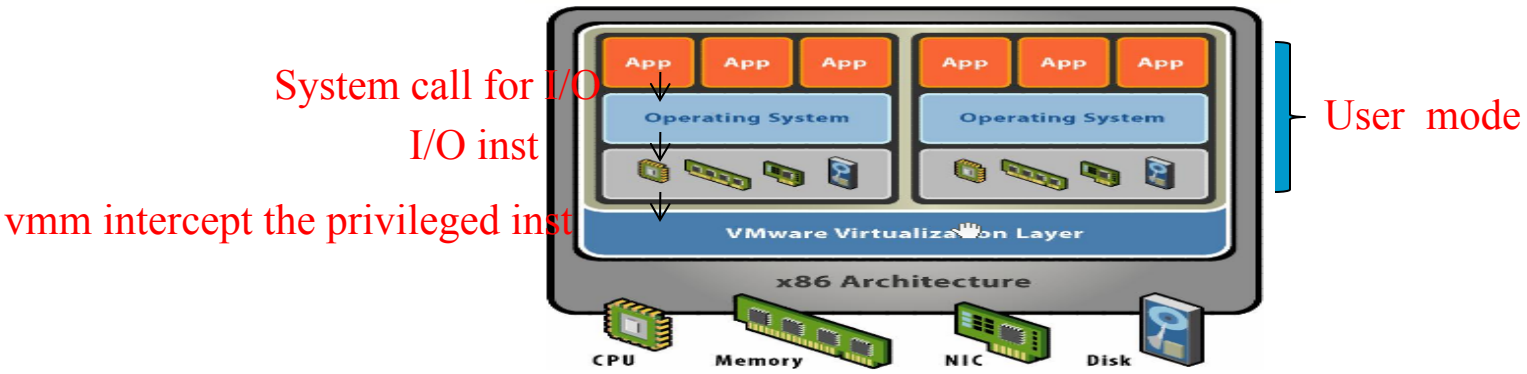
Example: Timer Virtualization

- In native machine, on timer interrupt
  - OS suspends current process, handles interrupt, selects and resumes next process
- With Virtual Machine Monitor
  - VMM suspends current VM, handles interrupt, selects and resumes next VM
- If a VM requires timer interrupts
  - VMM emulates a virtual timer



Instruction Set Support

- User and System modes
- Privileged instructions only available in system mode
  - Trap to system if executed in user mode
- All physical resources only accessible using privileged instructions
  - Including page tables, interrupt controls, I/O registers
- Renaissance of virtualization support
  - E.g., Intel VT (Virtualization Technology)
  - VMM must ensure that guest system only interacts with virtual resources ⇒ conventional guest OS runs as user mode program on top of VMM



Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk