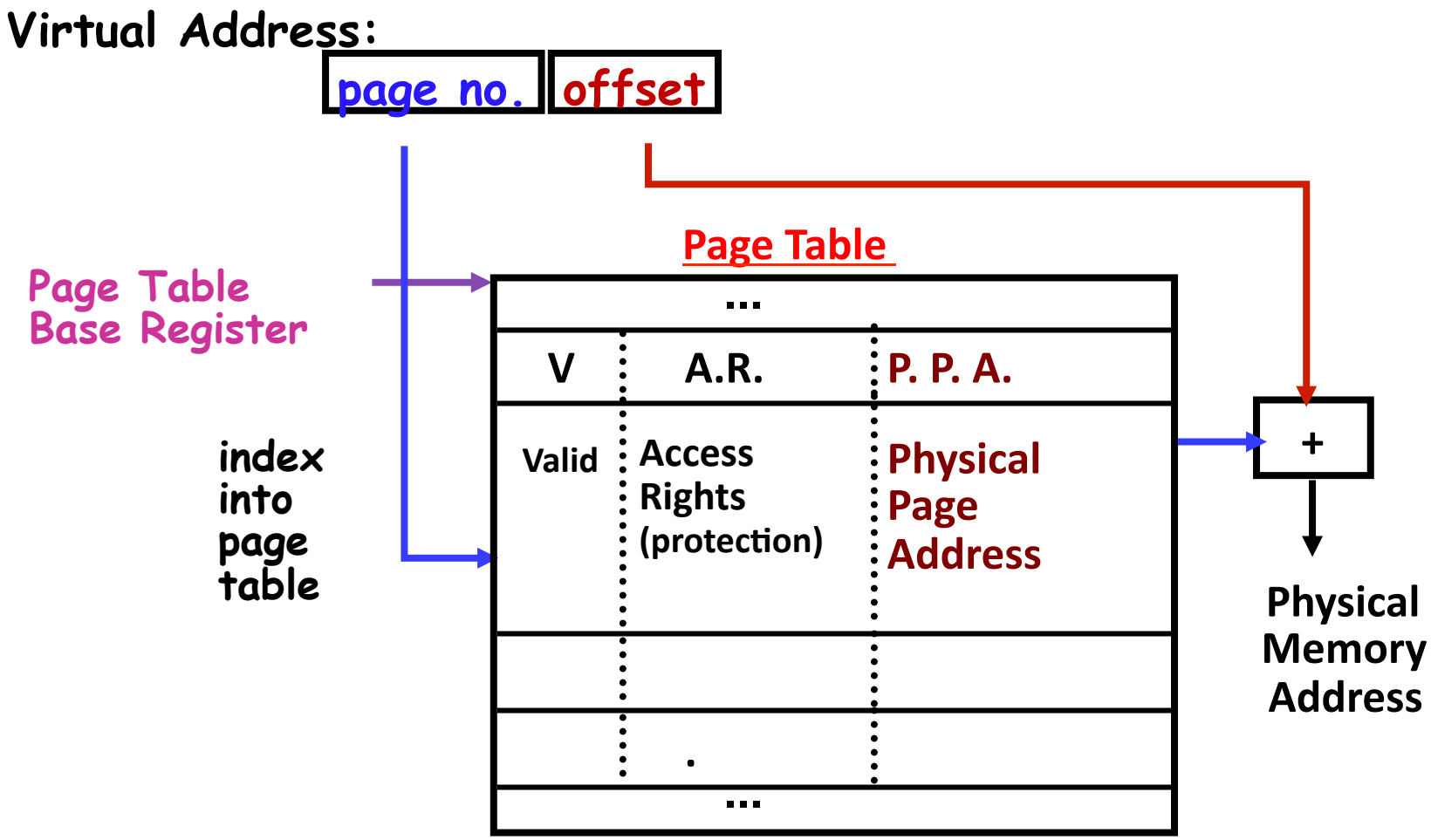
EECE 321: Computer Organization

Mohammad M. Mansour
Dept. of Electrical and Compute Engineering
American University of Beirut

Lecture 35: Virtual Memory

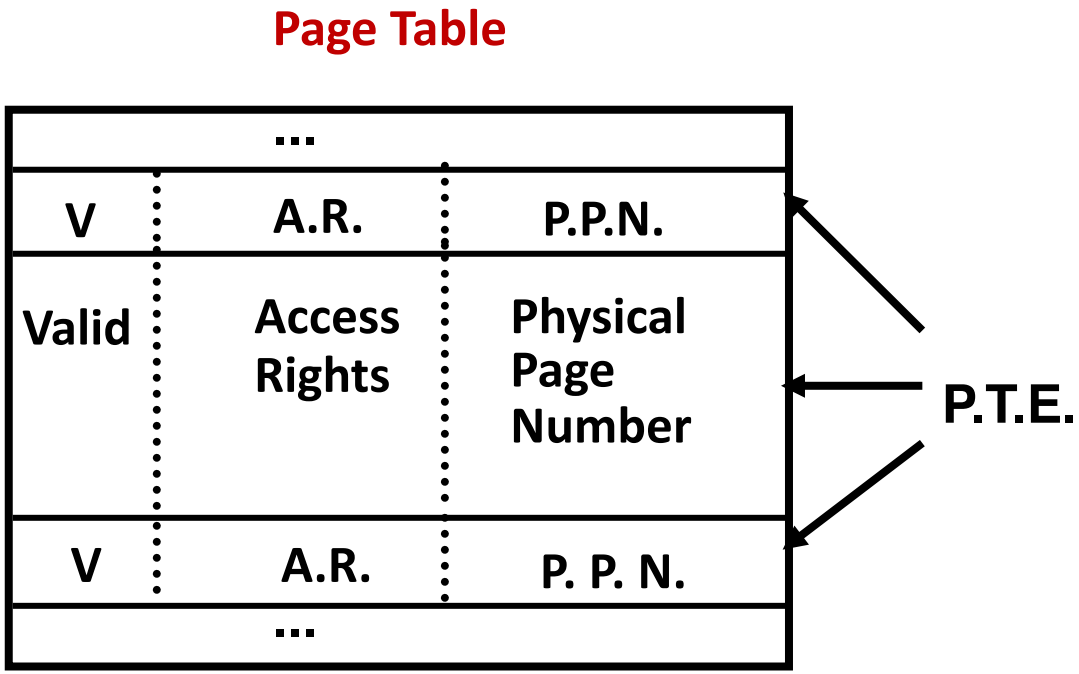
More Details about Page Table

- Page Table located in physical memory at address indicated by the Page Table Base Register



Page Table Entry (PTE) Format

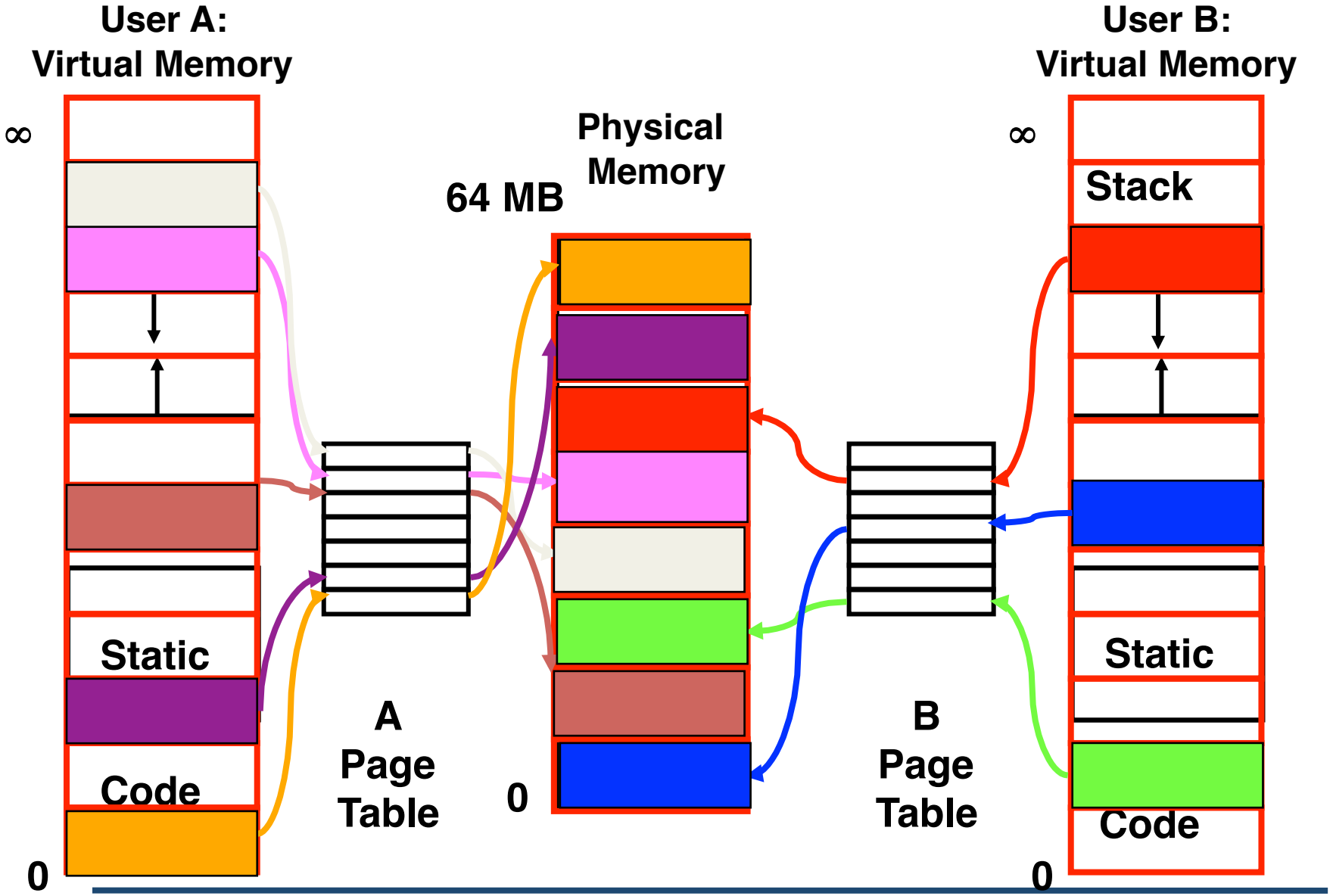
- Contains either Physical Page Number (PPN) or indication not in Main Memory
- OS maps to disk if Not Valid (V = 0)



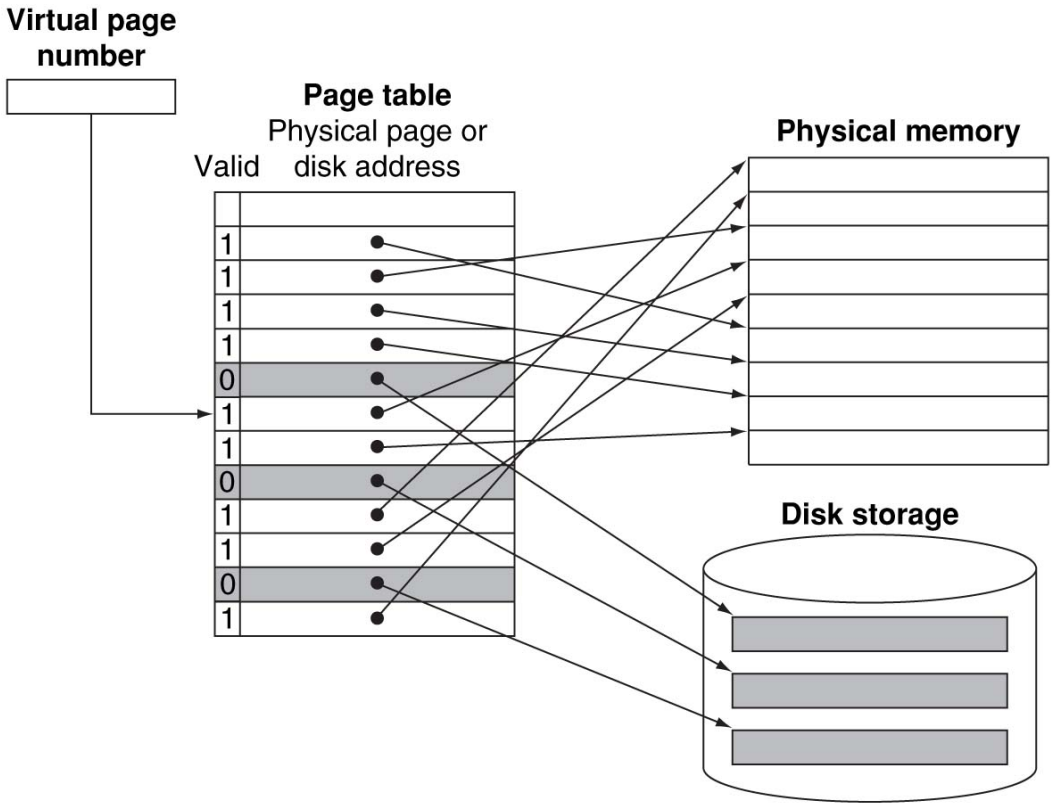
- **If valid, also check if have permission to use page:**

  Access Rights (A.R.) may be Read Only, Read/Write, Executable

Paging/Virtual Memory Multiple Processes



Page Table: Another View



Comparing the 2 Levels of Hierarchy: VM vs. Cache

Cache Version	Virtual Memory version
Block or Line	Page
Miss	Page Fault
Block Size: 32-64B	Page Size: 4K-64KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU)
Write Thru or Back	Write Back

Page Size

- On Unix, use the system function sysconf() to get page size: (or simply type pagesize command)

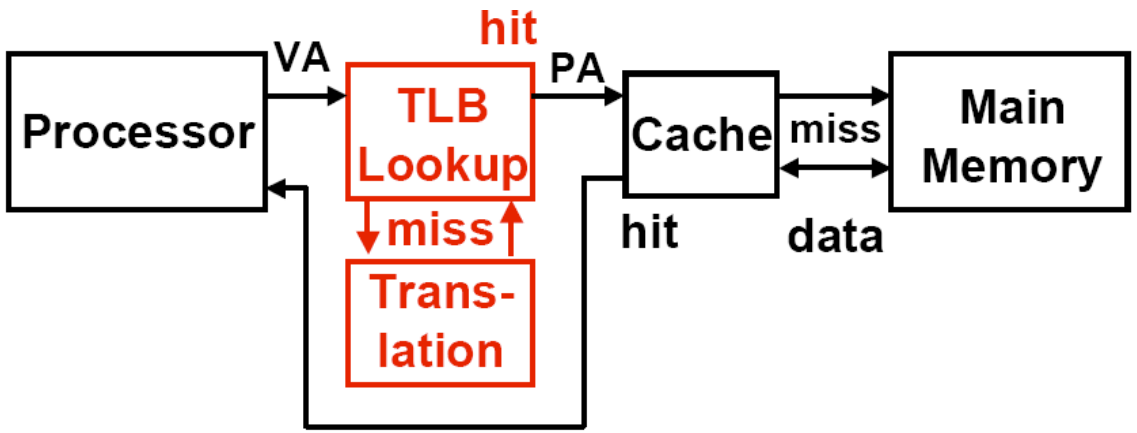
```
#include <stdio.h>
#include <unistd.h> // sysconf(3)
int main() {
   printf("The page size for this system is %ld bytes.\n",
          sysconf(_SC_PAGESIZE)); // _SC_PAGE_SIZE is OK too.

   return 0;
}
```

- **On Windows, use the system function** the system function GetSystemInfo () from kernel32.dll

```
#include <stdio.h>
#include <windows.h>
int main() {
   SYSTEM_INFO si;
   GetSystemInfo(&si);
   printf("The page size for this system is %u bytes.\n", si.dwPageSize);
   return 0;
}
```

Problems with Virtual Memory (Thus Far)

- (1) Slow: Every memory access requires:
  - 1 access to PT to get VPN->PPN translation
  - 1 access to MEM to get data at PA
- Observation: since locality in pages of data, there must be locality in virtual address translations of those pages.
- Since small is fast, why not use a small cache of virtual to physical address translations to make translation fast? (i.e., save the translations in a cache)
- For historical reasons, cache is called a Translation Lookaside Buffer, or TLB
  - **On TLB miss, get page table entry from main memory**



Problems with Virtual Memory (Thus Far)

- (2) Page Table too big!
  - 4GB Virtual Memory ÷ 1 KB page
    ⇒ ~ 4 million Page Table Entries
    ⇒ 16 MB just for Page Table for 1 process,
    8 processes ⇒ 256 MB for Page Tables!

- Spatial Locality to the rescue
  - Ex: if each page is 4 KB, lots of nearby references
  - No matter how big program is, at any time only accessing a few pages
  - "Working Set": recently used pages

Translation Look-aside Buffer (TLB)

- To avoid accessing memory twice to obtain a physical address, a special cache that keeps track of recent translations is used.
  - This cache is called a translation look-aside buffer (TLB).
  - TLB just a cache on the page table mappings
- TLB access time comparable to cache (much less than main memory access time)
- The TLB cache entries include:
  - A tag entry that holds a portion of the virtual page number
  - A data entry that holds the physical page number.
  - Other bookkeeping bits: Ex:
    - **Dirty** bit if page is written, upon replacement need to know whether to write back or not.
    - **Reference** bit if page was accessed recently (to implement an LRU policy).

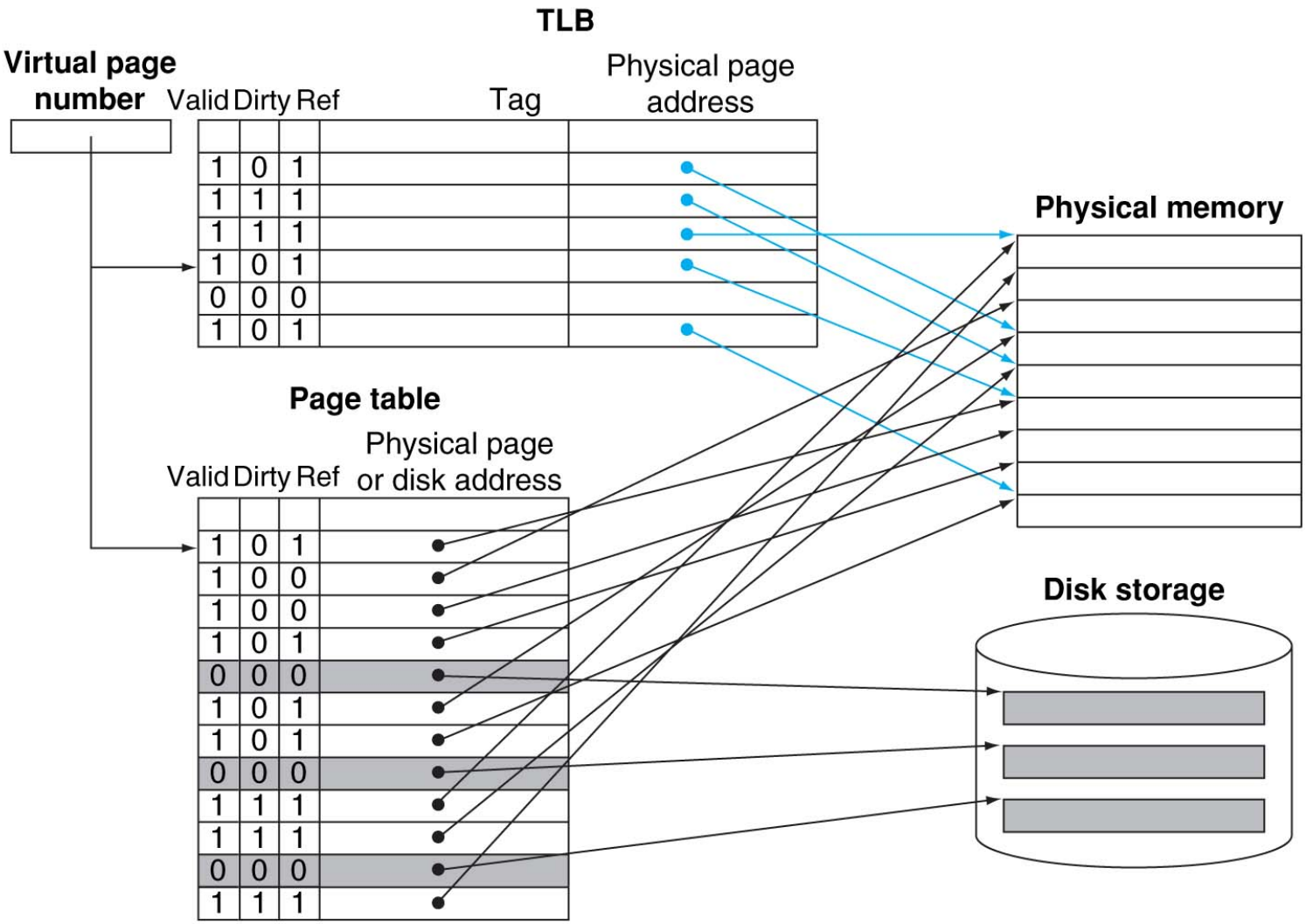
Tag page #	Physical	Dirty	Ref	Valid	Access Rights

Translation Look-aside Buffer (TLB)

- TLBs are usually small, typically up to 4K entries
  - Block size = 1 or 2 page table entries
  - Miss penalty = 10-30 clock cycles; Miss rate = 0.01%- 1%

- Like any other cache, the TLB can be
  - direct mapped,
  - set associative, or
  - fully associative

Address Mapping Using TLB



Translation Look-aside Buffer (TLB)

- On every memory reference, the virtual page number is looked up in the TLB.
  1. If there is hit, the physical page number is used to form the physical address, and the reference bit is set. That address is then sent to cache.
     - If the processor is performing a write, the dirty bit is also turned on.

  2. If there is a TLB miss, we must determine if there is a page fault or a TLB miss (just the translation is missing, but the actual page is in memory).

     - Case 1: If the page is in memory and only translation is missing, the CPU can handle it by loading the translation from memory into TLB and then trying the reference again.

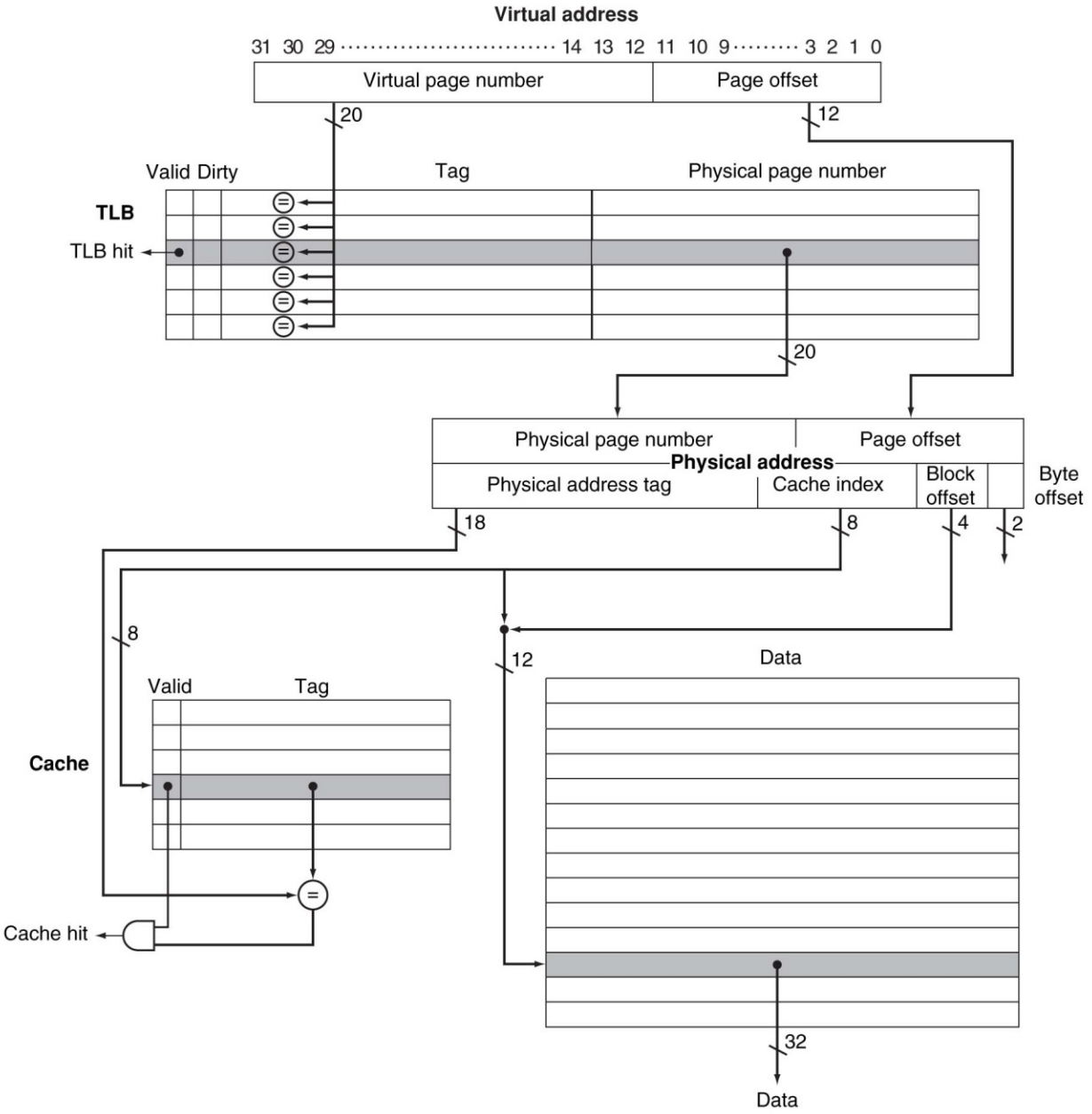
     - Case 2: If the page is not present in memory, then the TLB miss is a real page fault. In this case the **CPU invokes the OS using an exception**. The exception handling routine will bring the missing page into memory and update both the page table and the TLB.

- Because TLB size is small, TLB misses will be much more frequent than true page faults.

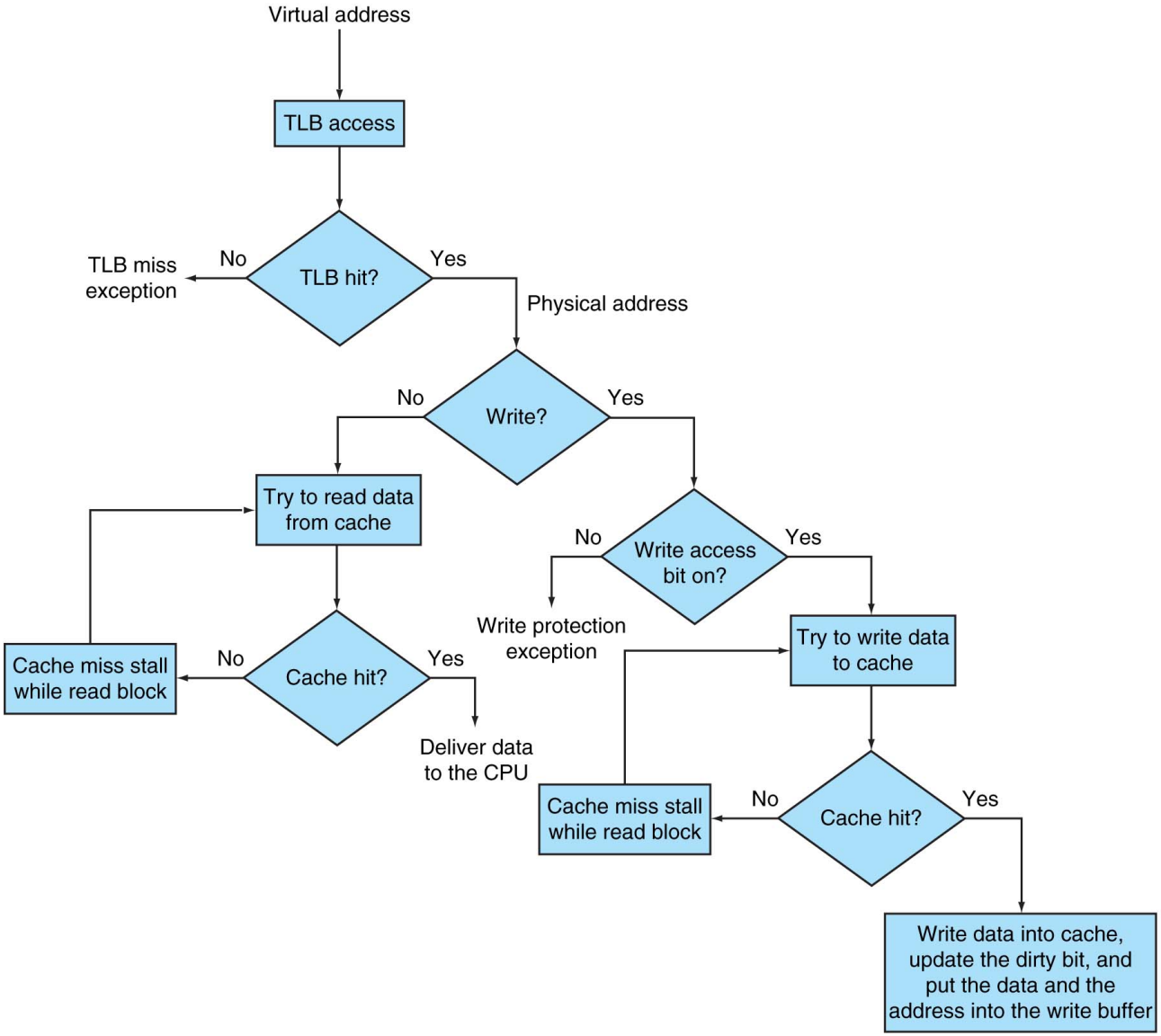
MIPS R2000 TLB

Memory system:

4K pages
32-bit address
64-entry TLB
VA = PA



Example: Intrinsity FastMATH TLB



Possible Combinations of Events in TLB, VM System, & Cache

TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry 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 must miss 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.

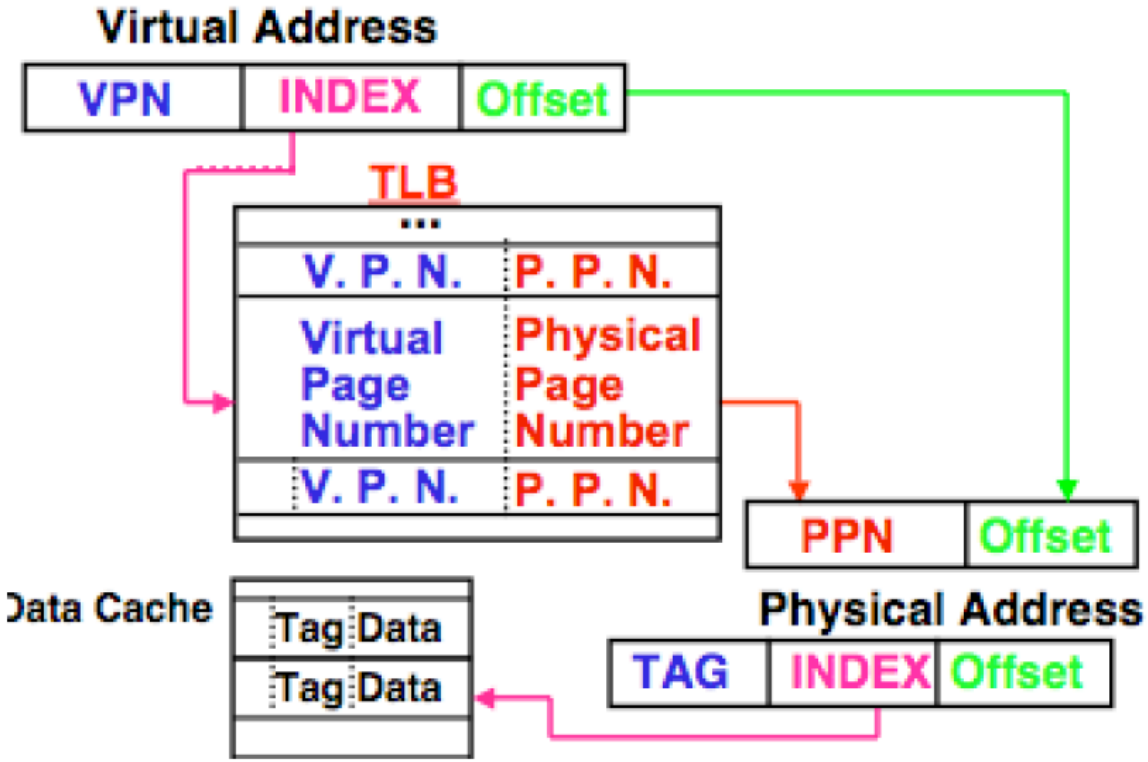
Typical Key Design Parameters

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	15,000–50,000	16,000–250,000	40–1024
Total size in kilobytes	16–64	500–4000	1,000,000–1,000,000,000	0.25–16
Block size in bytes	16–64	64–128	4000–64,000	4–32
Miss penalty in clocks	10–25	100–1000	10,000,000–100,000,000	10–1000
Miss rates (global for L2)	2%–5%	0.1%–2%	0.00001%–0.0001%	0.01%–2%

Recap

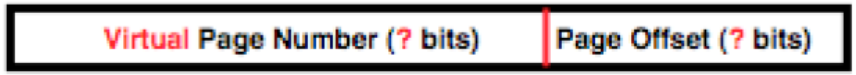
- What if we miss in TLB?
  - **Option 1: Hardware checks page table and loads new Page Table Entry into TLB**
  - **Option 2: Hardware traps to OS, up to OS to decide what to do**
    - **MIPS follows Option 2: Hardware knows nothing about page table**
- What if the data is on disk? (i.e., true page fault)
  - **We load the page off the disk into a free block of memory, using a DMA (Direct Memory Access – very fast!) transfer**
    - **In the meantime, OS switches to some other process waiting to be run**
  - **When the DMA is complete, we get an interrupt and update the process's page table**
    - **So when we switch back to the task, the desired data will be in memory**
- What if we don't have enough memory?
  - **We choose some other page belonging to a program and transfer it onto the disk if it is dirty**
    - **If clean (disk copy is up-to-date), just overwrite that data in memory**
    - **We choose the page to evict based on replacement policy (e.g., LRU)**
  - **And update that program's page table to reflect the fact that its memory moved somewhere else**
  - **If pages are continuously swapped between disk and memory, then this process is called Thrashing.**

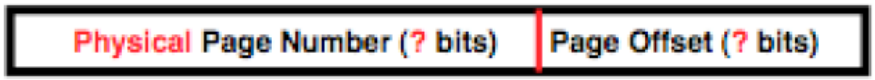
Address Translation



Question 1

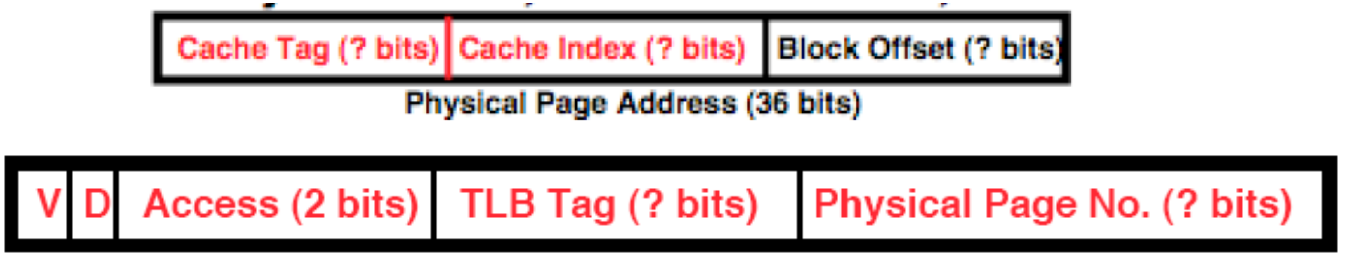
- Assume
  - 40-bit virtual address, 16 KB page
  - 36-bit physical address
- What is the number of bits in the following fields:
  - Virtual Page Number
  - Virtual Page offset
  - Physical Page Number
  - Physical Page offset?
- Solution:
  1. **22/18 (VPN/PO), 22/14 (PPN/PO)**
  2. **24/16, 20/16**
  3. **26/14, 22/14**
  4. **26/14, 26/10**
  5. **28/12, 24/12**





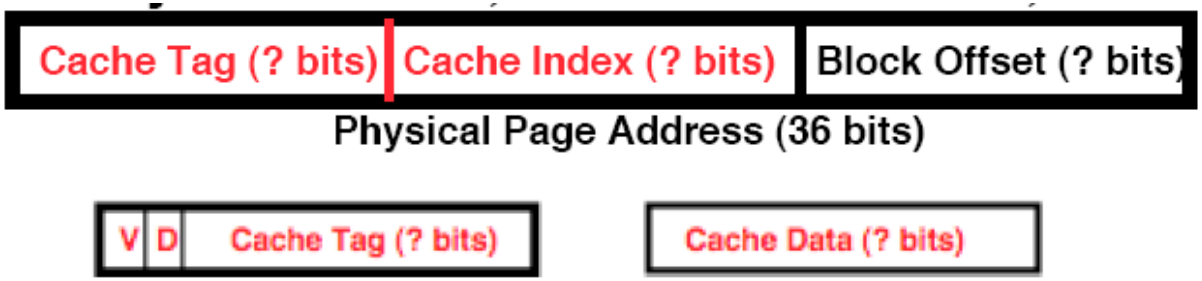
Question 2

- Assume
  - **2-way set-assoc. TLB, 512 entries, 40b VA, 36b PA, 16KB pages**
  - **TLB Entry: Valid bit, Dirty bit, Access Control (say 2 bits), Virtual Page Number, Physical Page Number**
- What is the number of bits in TLB Tag / Index / Entry?
  - **1: 12 / 14 / 38 (TLB Tag / Index / Entry)**
  - **2: 14 / 12 / 40**
  - **3: 18 / 8 / 44**
  - **4: 18 / 8 / 58**



Question 3

- Assume
  - **2-way set-assoc, 64KB data cache, 64B block**
  - **Data Cache Entry: Valid bit, Dirty bit, Cache tag + ? bits of Data**
- What is the number of bits in Data cache Tag / Index / Offset / Entry?
  - **1: 12 / 9 / 14 / 87 (Tag/Index/Offset/Entry)**
  - **2: 20 / 10 / 6 / 86**
  - **3: 20 / 10 / 6 / 534**
  - **4: 21 / 9 / 6 / 87**
  - **5: 21 / 9 / 6 / 535**



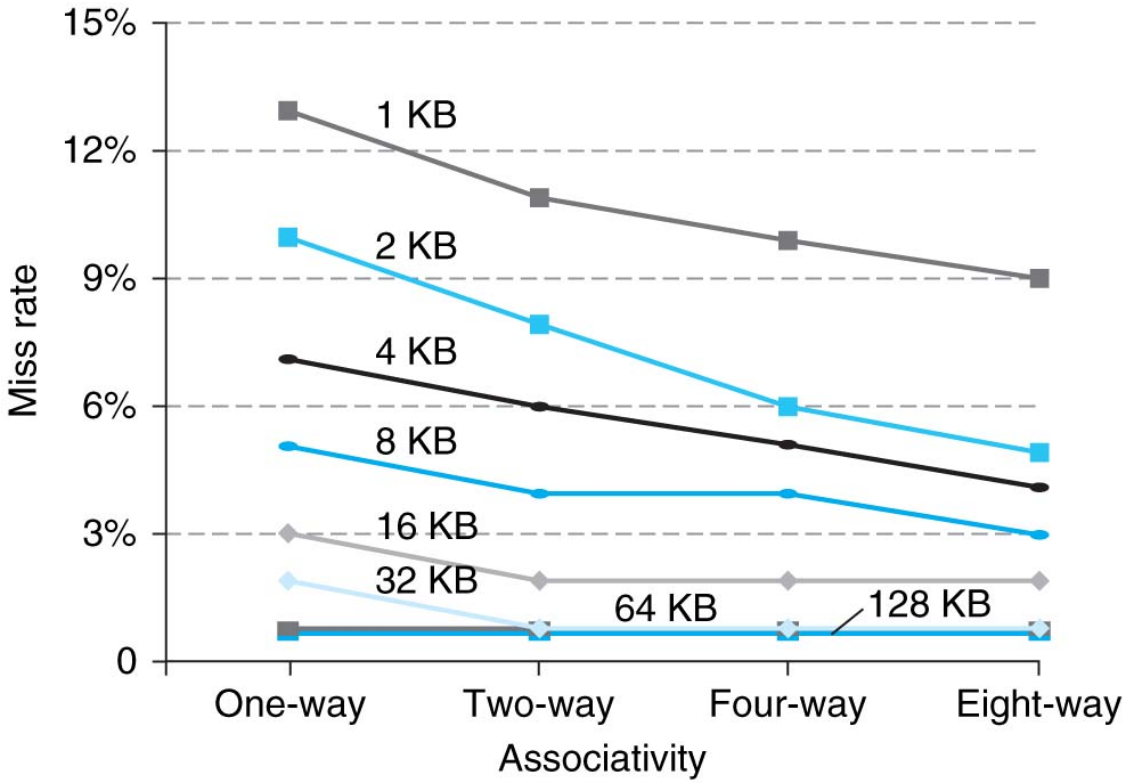
Questions for Any Memory Hierarchy

- Q1: Where can a block be placed?
  - One place (direct mapped)
  - A few places (set associative)
  - Any place (fully associative)
- Q2: How is a block found?
  - Indexing (as in a direct-mapped cache)
  - Limited search (as in a set-associative cache)
  - Full search (as in a fully associative cache)
  - Separate lookup table (as in a page table)
- Q3: Which block is replaced on a miss?
  - Least recently used (LRU)
  - Random
- Q4: How are writes handled?
  - Write through (Level never inconsistent w/lower)
  - Write back (Could be “dirty”, must have dirty bit)

Three Advantages of Virtual Memory

- Translation:
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multiple processes reasonable
  - 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 still grow later
- Protection:
  - Different 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
  - Special Mode in processor ("Kernel mode") allows processor to change page table/TLB
- Sharing:
  - Can map same physical page to multiple users("Shared memory")

Impact of Associativity on Miss Rate



Memory Hierarchy Design Challenges

Design change	Effect on miss rate	Possible negative performance effect
Increase cache size	Decreases capacity misses	May increase access time
Increase associativity	Decreases miss rate due to conflict misses	May increase access time
Increase block size	Decreases miss rate for a wide range of block sizes due to spatial locality	Increases miss penalty. Very large block could increase miss rate