CMSC 313 COMPUTER ORGANIZATION & ASSEMBLY LANGUAGE PROGRAMMING

LECTURE 27, FALL 2012

ANNOUNCEMENTS

Need student input on Lecturer Search

- **Max Morawski**
  - Lecture 2:30pm – 3:15pm, Fri 12/7, ITE 217
  - Meet with students 3:15pm – 3:45pm

- **Dr. Pedram Sadeghian**
  - Lecture 9:30am – 10:15am, Tues 12/11, ITE 217
  - Meet with students 10:15am – 10:45am

TOPICS TODAY

- Finish Caching
- Virtual Memory

RECAP CACHING

CACHING

- **Why: bridge speed difference between CPU and RAM**
- **Modern RAM allows blocks of memory to be read quickly**
- **Principle of locality: temporal and spatial**

During each memory access :

- **CPU checks if memory location is already in cache**
- **Found = cache hit:**
  - read from or write to cache
- **Not Found = cache miss:**
  - Fetch entire memory block of location into cache

CACHE MAPPING SCHEMES

Direct Mapping:

- **Each memory block mapped to 1 cache block**
- **Many memory blocks for each cache block**
- **Use *tag* to check if block in cache is the one needed**

Fully Associative Mapping:

- **Each memory block can be placed in any cache block**
- **Associative memory finds cache block with *tag***

Set Associative Mapping:

- **Hybrid of direct mapping and fully associative mapping**

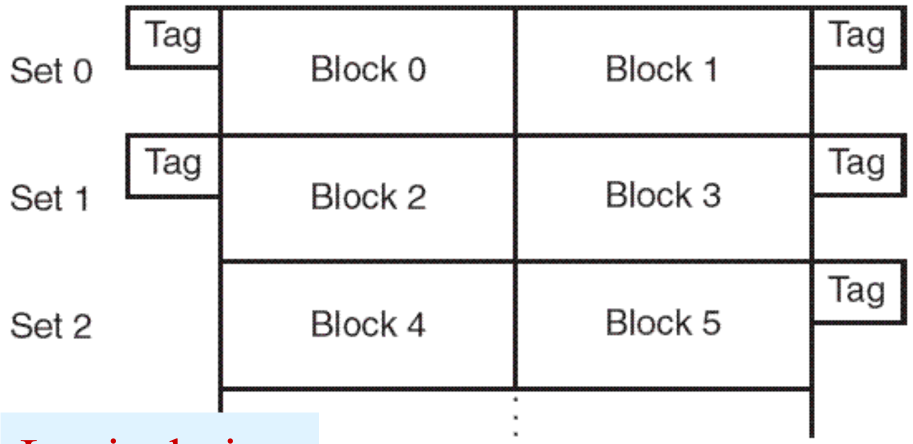
SET ASSOCIATIVE MAPPING

6.4 Cache Memory

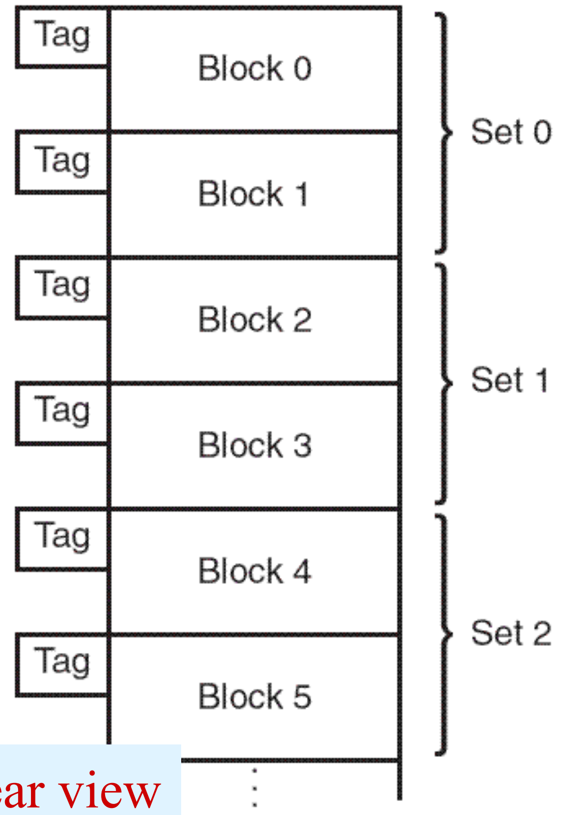
- Set associative cache combines the ideas of direct mapped cache and fully associative cache.
- An *N*-way set associative cache mapping is like direct mapped cache in that a memory reference maps to a particular location in cache.
- Unlike direct mapped cache, a memory reference maps to a set of several cache blocks, similar to the way in which fully associative cache works.
- Instead of mapping anywhere in the entire cache, a memory reference can map only to the subset of cache slots.

6.4 Cache Memory

- The number of cache blocks per set in set associative cache varies according to overall system design.
  - For example, a 2-way set associative cache can be conceptualized as shown in the schematic below.
  - Each set contains two different memory blocks.



Logical view



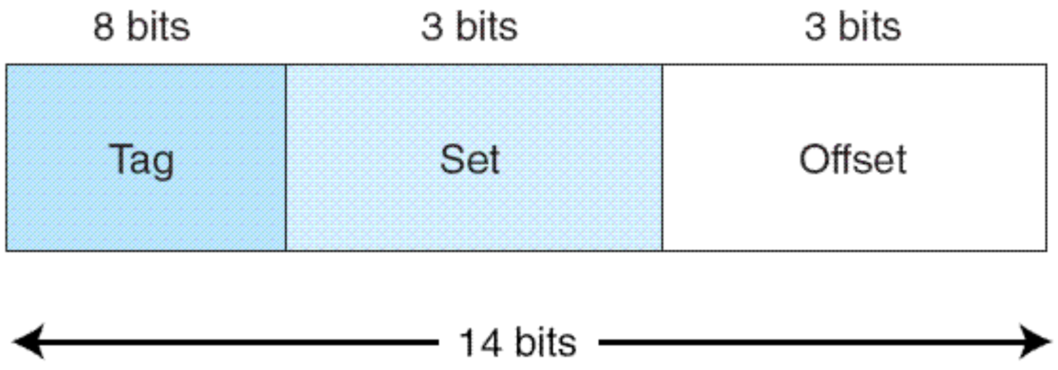
Linear view

6.4 Cache Memory

- In set associative cache mapping, a memory reference is divided into three fields: tag, set, and offset.
- As with direct-mapped cache, the offset field chooses the word within the cache block, and the tag field uniquely identifies the memory address.
- The set field determines the set to which the memory block maps.

6.4 Cache Memory

- EXAMPLE 6.5 Suppose we are using 2-way set associative mapping with a word-addressable main memory of 2^{14} words and a cache with 16 blocks, where each block contains 8 words.
  - Cache has a total of 16 blocks, and each set has 2 blocks, then there are 8 sets in cache.
  - Thus, the set field is 3 bits, the offset field is 3 bits, and the tag field is 8 bits.



CACHING POLICIES

- **Cache replacement policy**
    - For fully associative and set associative mapping
    - Which cache block gets kicked out?
    - Some schemes: first-in first-out, least recently used, ...

- **Cache write policy**
    - Write through: always write to main memory
    - Write back: write to main memory when replaced

CACHE PERFORMANCE

6.4 Cache Memory

- The performance of hierarchical memory is measured by its *effective access time* (EAT).
- EAT is a weighted average that takes into account the hit ratio and relative access times of successive levels of memory.
- The EAT for a two-level memory is given by:

$$\text{EAT} = H \times \text{Access}_C + (1-H) \times \text{Access}_{MM}.$$

where H is the cache hit rate and Access_C and Access_{MM} are the access times for cache and main memory, respectively.

6.4 Cache Memory

- For example, consider a system with a main memory access time of 200ns supported by a cache having a 10ns access time and a hit rate of 99%.
- Suppose access to cache and main memory occurs concurrently. (The accesses overlap.)
- The EAT is:

  0.99(10ns) + 0.01(200ns) = 9.9ns + 2ns = 11ns.

6.4 Cache Memory

- For example, consider a system with a main memory access time of 200ns supported by a cache having a 10ns access time and a hit rate of 99%.
- If the accesses do not overlap, the EAT is:

$$0.99(10ns) + 0.01(10ns + 200ns)$$

$$= 9.9ns + 2.01ns = 12ns.$$

- This equation for determining the effective access time can be extended to any number of memory levels, as we will see in later sections.

VIRTUAL MEMORY

MEMORY PROBLEMS

Not enough memory

- Many processes ran simultaneously
- Large applications, but most code is unused (MS Word)

Fragmentation

- Processes need contiguous blocks of memory
- Total amount of free memory is sufficient, but largest block of contiguous memory is too small

Unprotected memory

- Many processes ran simultaneously
- "Bad" processes can overwrite other processes' memory

6.5 Virtual Memory

- Cache memory enhances performance by providing faster memory access speed.
- Virtual memory enhances performance by providing greater memory capacity, without the expense of adding main memory.
- Instead, a portion of a disk drive serves as an extension of main memory.
- If a system uses paging, virtual memory partitions main memory into individually managed *page frames*, that are written *(or paged)* to disk when they are not immediately needed.

6.5 Virtual Memory

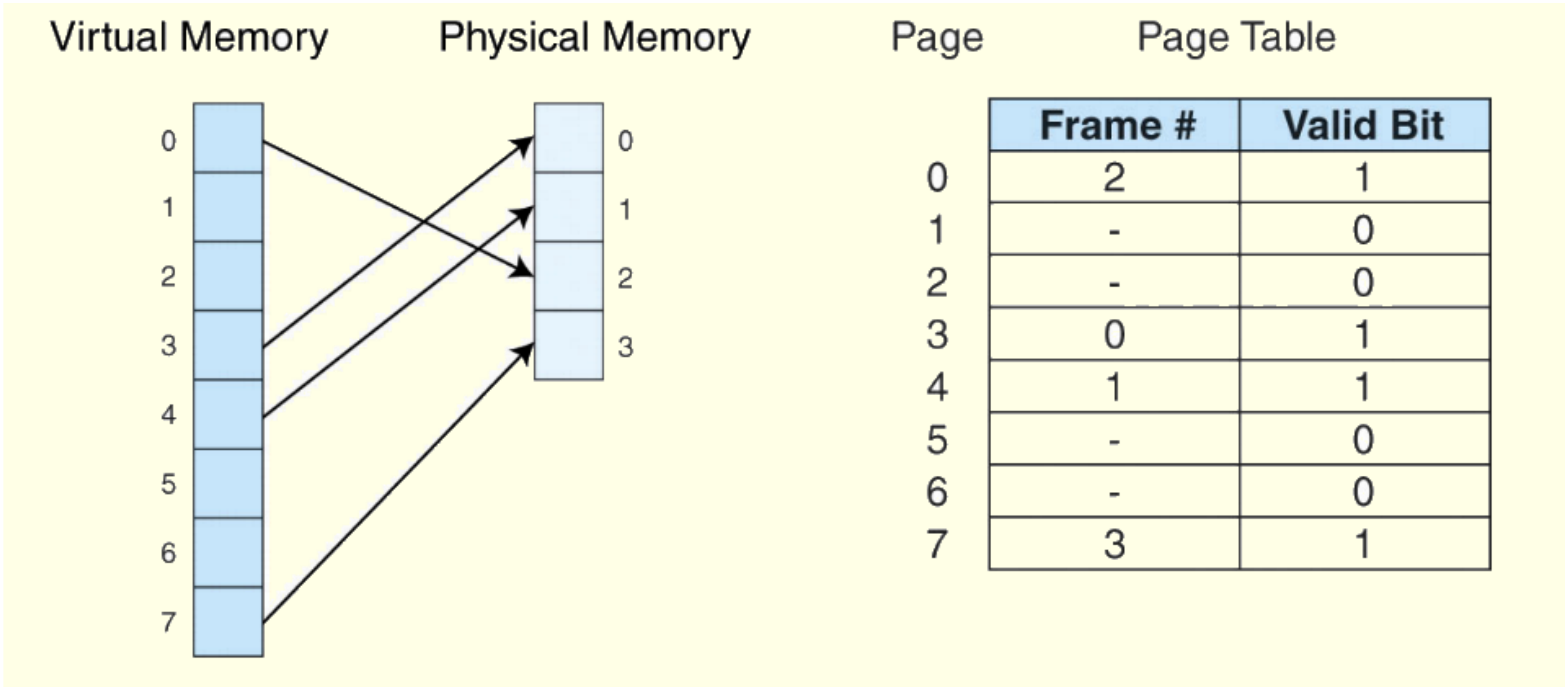
- A *physical address* is the actual memory address of physical memory.
- Programs create *virtual addresses* that are *mapped* to physical addresses by the memory manager.
- *Page faults* occur when a logical address requires that a page be brought in from disk.
- *Memory fragmentation* occurs when the paging process results in the creation of small, unusable clusters of memory addresses.

6.5 Virtual Memory

- Main memory and virtual memory are divided into equal sized pages.
- The entire address space required by a process need not be in memory at once. Some parts can be on disk, while others are in main memory.
- Further, the pages allocated to a process do not need to be stored contiguously-- either on disk or in memory.
- In this way, only the needed pages are in memory at any time, the unnecessary pages are in slower disk storage.

6.5 Virtual Memory

- Information concerning the location of each page, whether on disk or in memory, is maintained in a data structure called a *page table* (shown below).
- There is one page table for each active process.



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

6.5 Virtual Memory

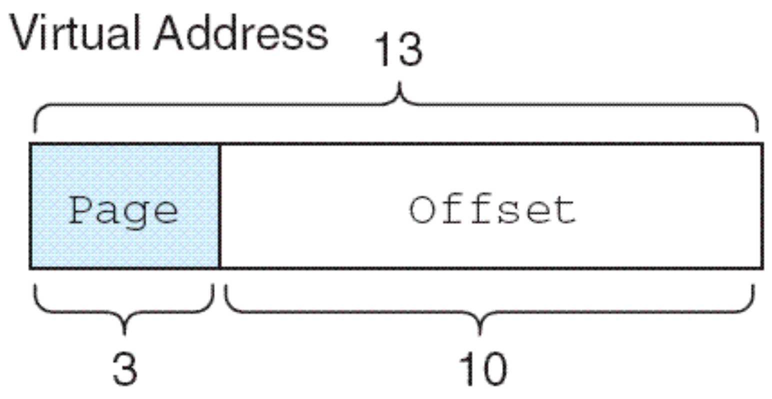
- When a process generates a virtual address, the operating system translates it into a physical memory address.
- To accomplish this, the virtual address is divided into two fields: A *page* field, and an *offset* field.
- The page field determines the page location of the address, and the offset indicates the location of the address within the page.
- The logical page number is translated into a physical page frame through a lookup in the page table.

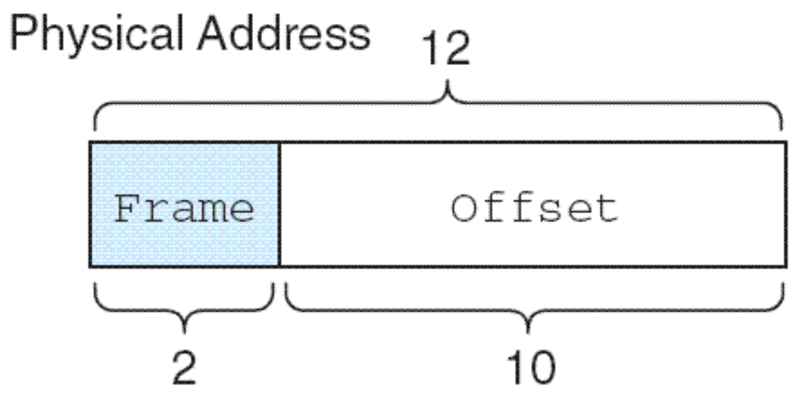
6.5 Virtual Memory

- If the valid bit is zero in the page table entry for the logical address, this means that the page is not in memory and must be fetched from disk.
  - This is a page fault.
  - If necessary, a page is evicted from memory and is replaced by the page retrieved from disk, and the valid bit is set to 1.
- If the valid bit is 1, the virtual page number is replaced by the physical frame number.
- The data is then accessed by adding the offset to the physical frame number.

6.5 Virtual Memory

- As an example, suppose a system has a virtual address space of 8K and a physical address space of 4K, and the system uses byte addressing.
  - We have $2^{13}/2^{10} = 2^3$ virtual pages.
- A virtual address has 13 bits (8K = 2^{13}) with 3 bits for the page field and 10 for the offset, because the page size is 1024.
- A physical memory address requires 12 bits, the first two bits for the page frame and the trailing 10 bits the offset.





6.5 Virtual Memory

- Suppose we have the page table shown below.
- What happens when CPU generates address 5459_{10} = 1010101010011_2 = 1553_{16}?

Page Table

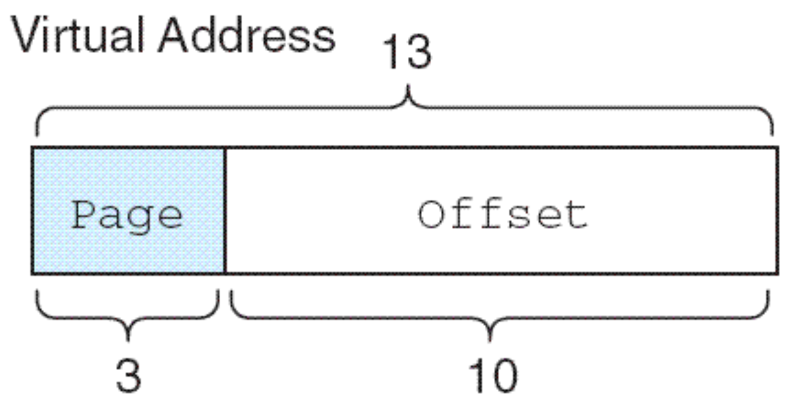
Page	Frame	Valid Bit
0	-	0
1	3	1
2	0	1
3	-	0
4	-	0
5	1	1
6	2	1
7	-	0

Addresses

Page	Base 10	Base 16
0 :	0 - 1023	0 - 3FF
1 :	1024 - 2047	400 - 7FF
2 :	2048 - 3071	800 - BFF
3 :	3072 - 4095	C00 - FFF
4 :	4096 - 5119	1000 - 13FF
5 :	5120 - 6143	1400 - 17FF
6 :	6144 - 7167	1800 - 1BFF
7 :	7168 - 8191	1C00 - 1FFF

6.5 Virtual Memory

- What happens when CPU generates address $5459_{10} = 1010101010011_2 = 1553_{16}$?



The high-order 3 bits of the virtual address, 101 (5_{10}), provide the page number in the page table.

6.5 Virtual Memory

- The address 1010101010011_2 is converted to physical address $010101010011_2 = 1363_{16}$ because the page field 101 is replaced by frame number 01 through a lookup in the page table.

Page Table

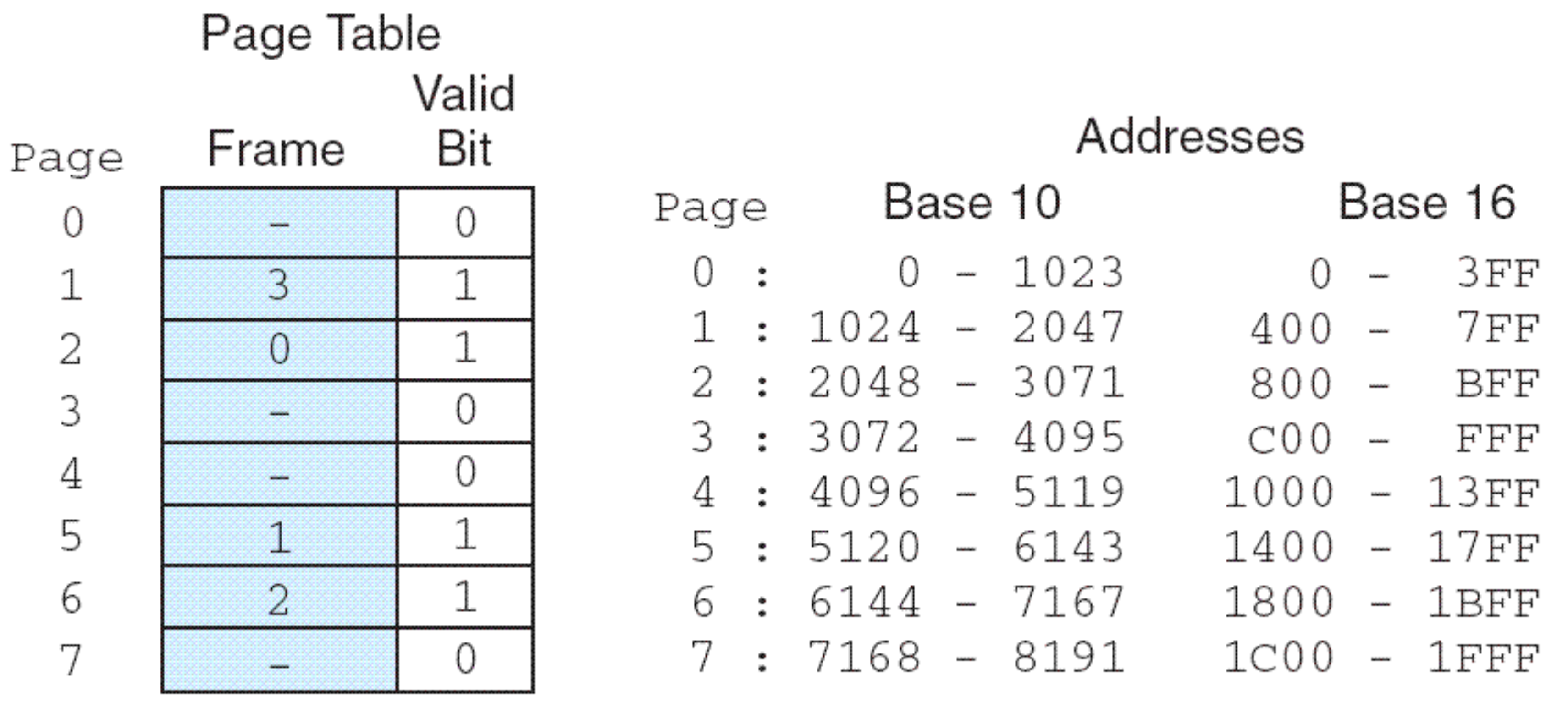
Page	Frame	Valid Bit
0	-	0
1	3	1
2	0	1
3	-	0
4	-	0
5	1	1
6	2	1
7	-	0

Addresses

Page	Base 10	Base 16
0 :	0 - 1023	0 - 3FF
1 :	1024 - 2047	400 - 7FF
2 :	2048 - 3071	800 - BFF
3 :	3072 - 4095	C00 - FFF
4 :	4096 - 5119	1000 - 13FF
5 :	5120 - 6143	1400 - 17FF
6 :	6144 - 7167	1800 - 1BFF
7 :	7168 - 8191	1C00 - 1FFF

6.5 Virtual Memory

- What happens when the CPU generates address 1000000000100_2?



Page Table

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

Addresses

Page	Base 10	Base 16
0 :	0 - 1023	0 - 3FF
1 :	1024 - 2047	400 - 7FF
2 :	2048 - 3071	800 - BFF
3 :	3072 - 4095	C00 - FFF
4 :	4096 - 5119	1000 - 13FF
5 :	5120 - 6143	1400 - 17FF
6 :	6144 - 7167	1800 - 1BFF
7 :	7168 - 8191	1C00 - 1FFF

6.5 Virtual Memory

- We said earlier that effective access time (EAT) takes all levels of memory into consideration.
- Thus, virtual memory is also a factor in the calculation, and we also have to consider page table access time.
- Suppose a main memory access takes 200ns, the page fault rate is 1%, and it takes 10ms to load a page from disk. We have:

  EAT = 0.99(200ns + 200ns) 0.01(10ms) = 100, 396ns.

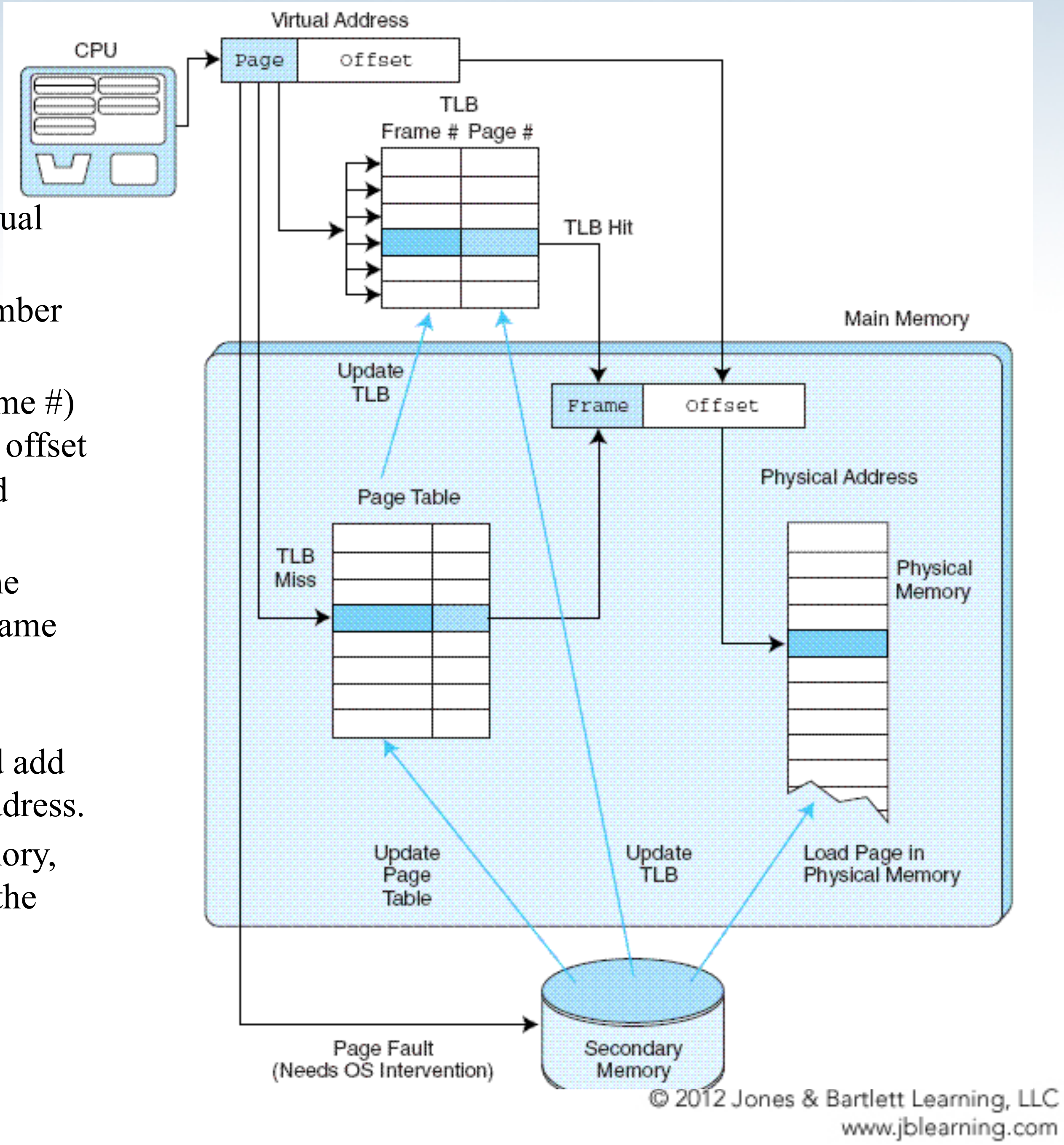
6.5 Virtual Memory

- Even if we had no page faults, the EAT would be 400ns because memory is always read twice: First to access the page table, and second to load the page from memory.
- Because page tables are read constantly, it makes sense to keep them in a special cache called a *translation look-aside buffer* (TLB).
- TLBs are a special associative cache that stores the mapping of virtual pages to physical pages.

The next slide shows address lookup steps when a TLB is involved.

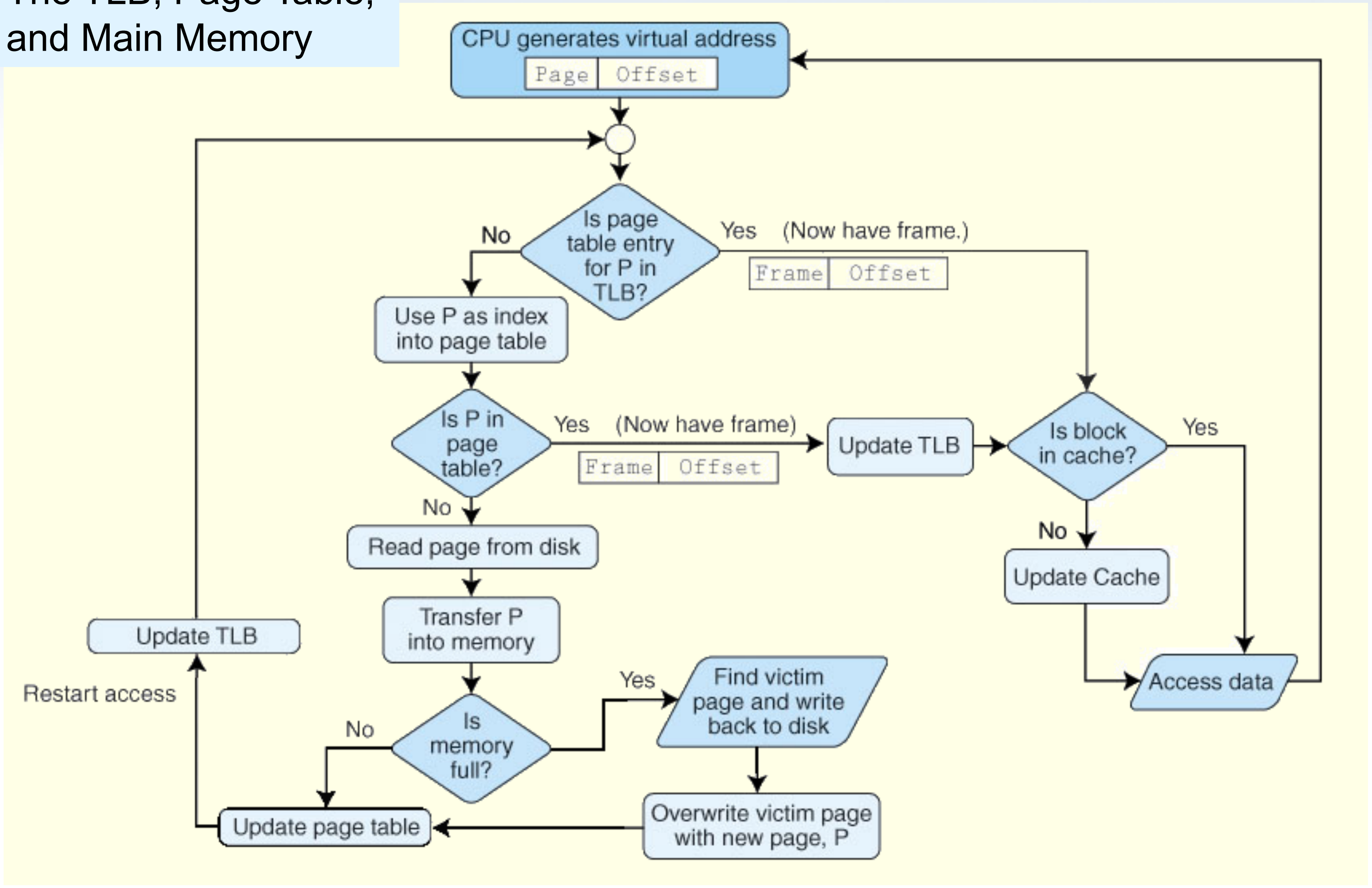
TLB lookup process

1. Extract the page number from the virtual address.
2. Extract the offset from the virtual address.
3. Search for the virtual page number in the TLB.
4. If the (virtual page #, page frame #) pair is found in the TLB, add the offset to the physical frame number and access the memory location.
5. If there is a TLB miss, go to the page table to get the necessary frame number.
If the page is in memory, use the corresponding frame number and add the offset to yield the physical address.
6. If the page is not in main memory, generate a page fault and restart the access when the page fault is complete.



6.5 Virtual Memory

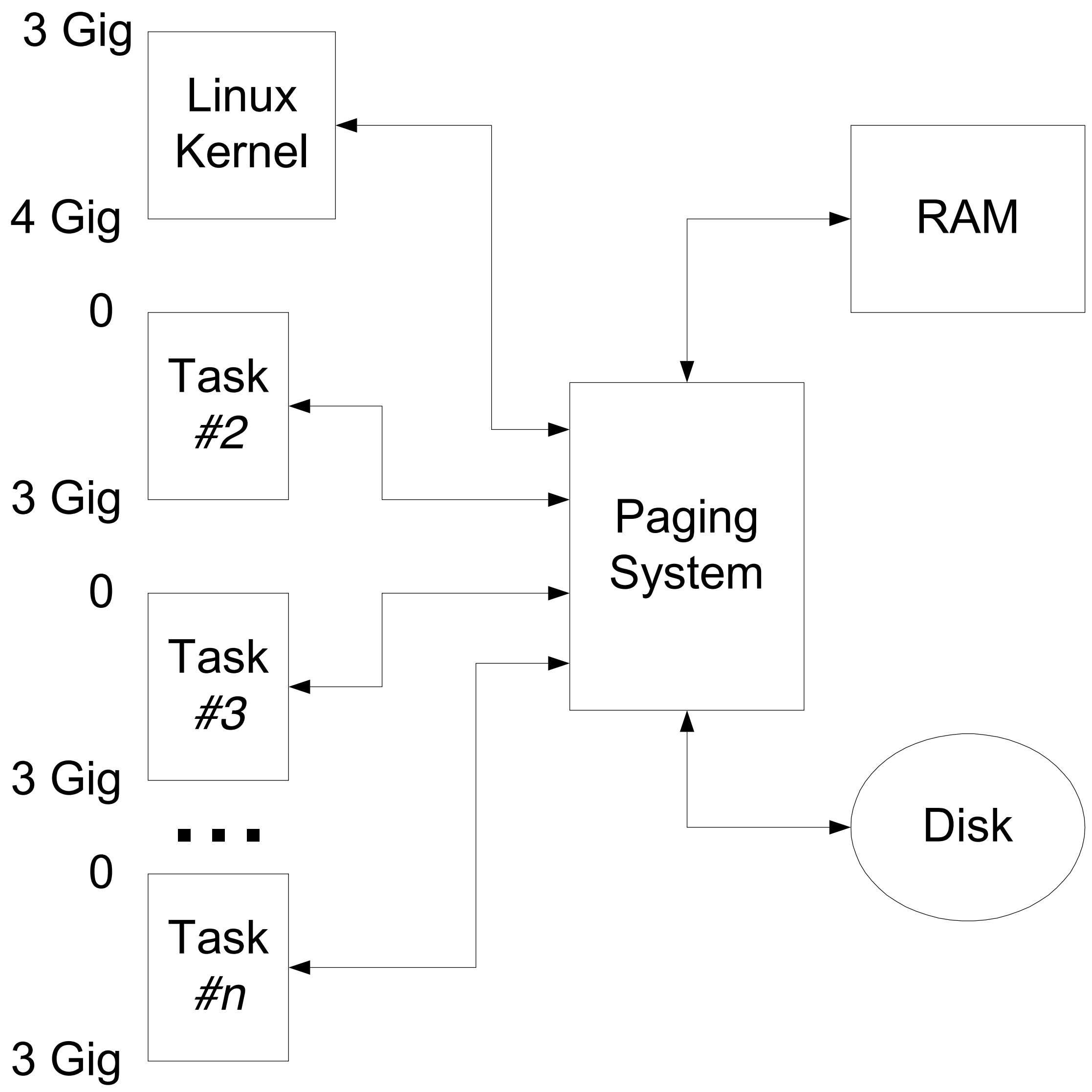
Putting it all together: The TLB, Page Table, and Main Memory



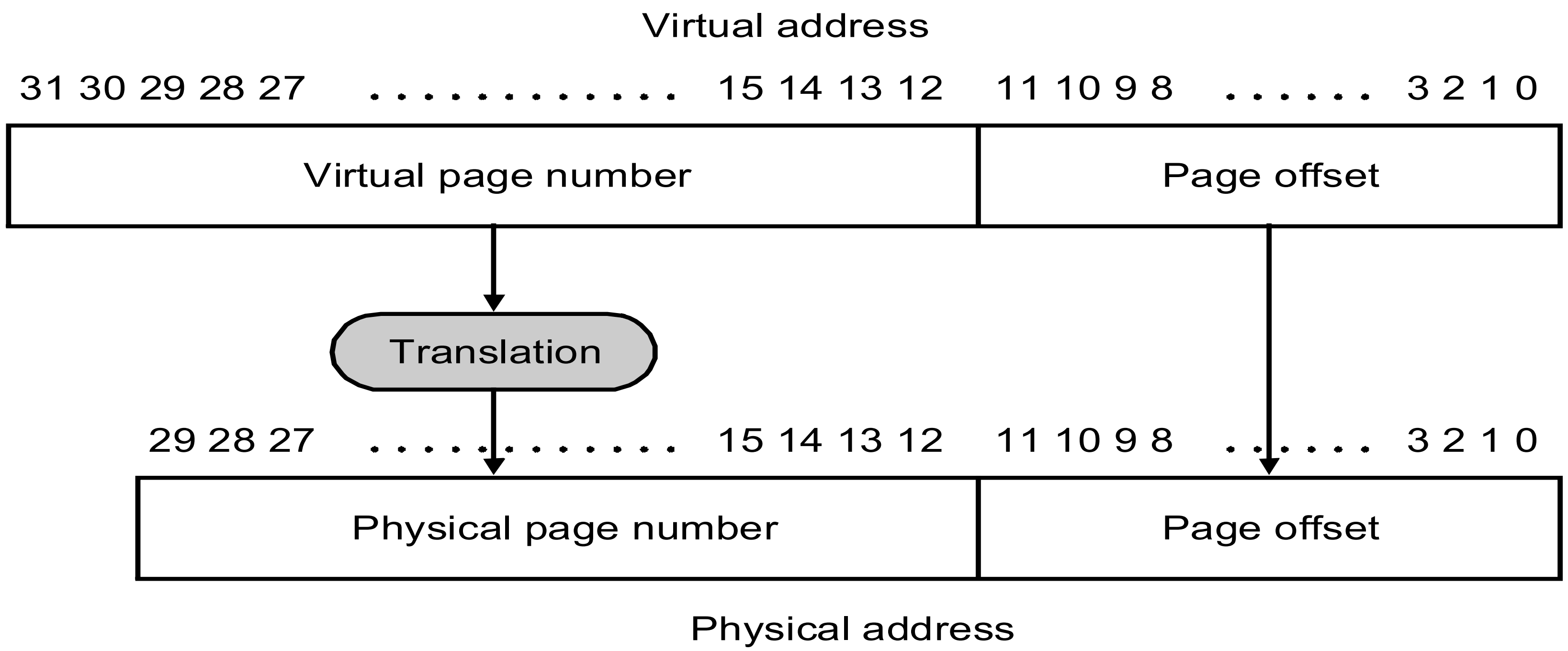
VIRTUAL MEMORY IN LINUX

Linux Virtual Memory Space

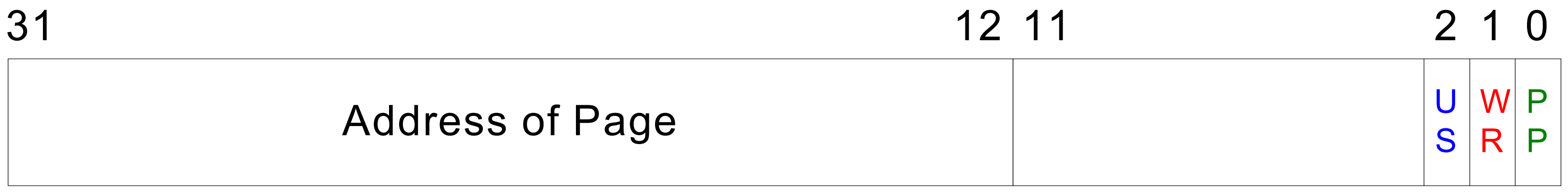
- Linux reserves 1 Gig memory in the virtual address space
- The size of the Linux kernel significantly affects its performance (swapping is expensive)
- Linux kernel can be customized by including only relevant modules
- Designating kernel space facilitates protection of
- The portion of disk used for paging is called the swap space



Virtual Addressing



- Page faults are costly and take millions of cycles to process (disks are slow)
- 80386 Page attributes:
  - RW: read and write permission
  - US: User mode or kernel mode only access
  - PP: present bit to indicate where the page is



Page Table

Hardware supported

Page table:

★ Resides in main memory

★ One entry per virtual page

★ No tag is requires since it covers all virtual pages

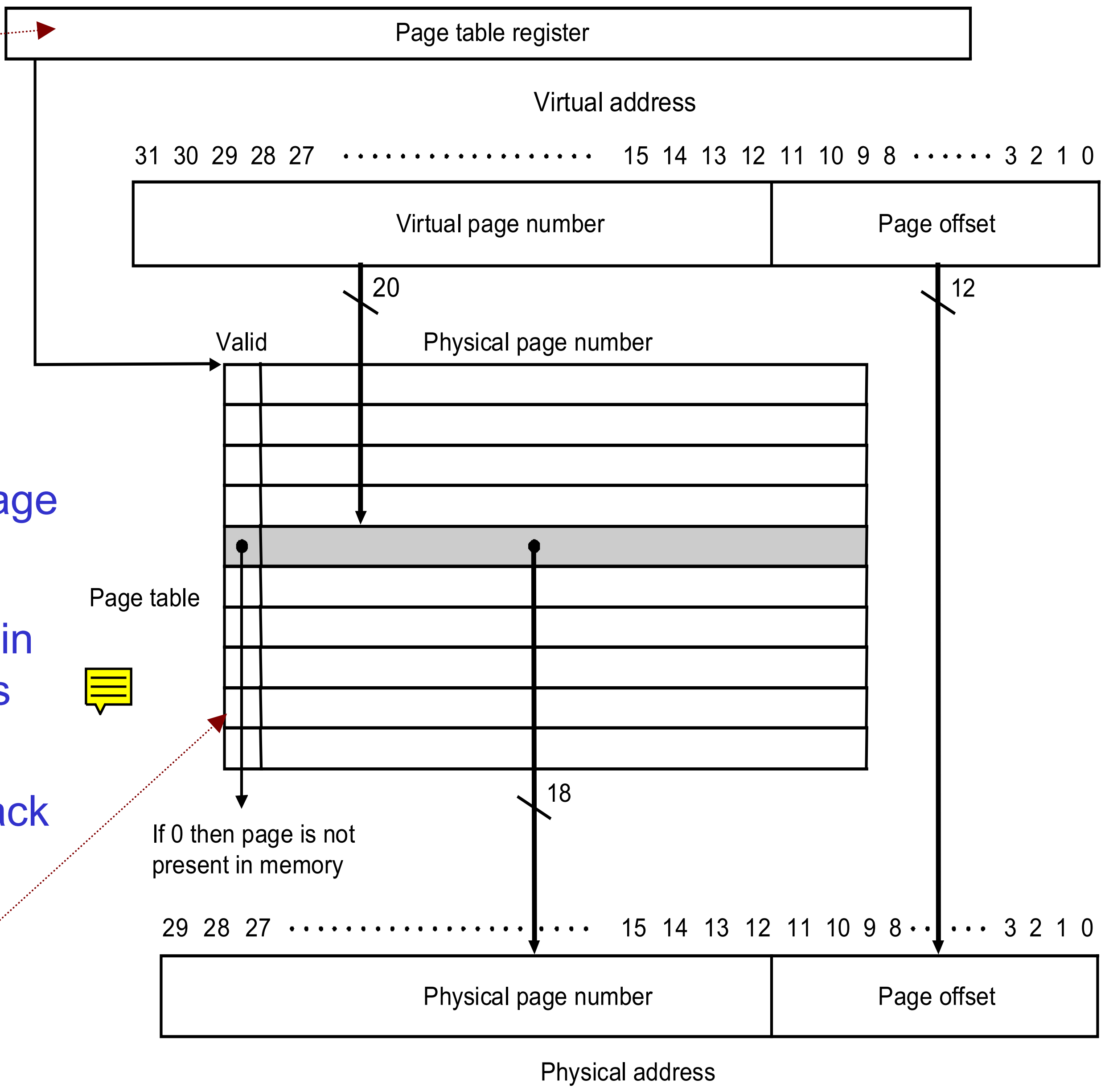
★ Point directly to physical page

★ Table can be very large

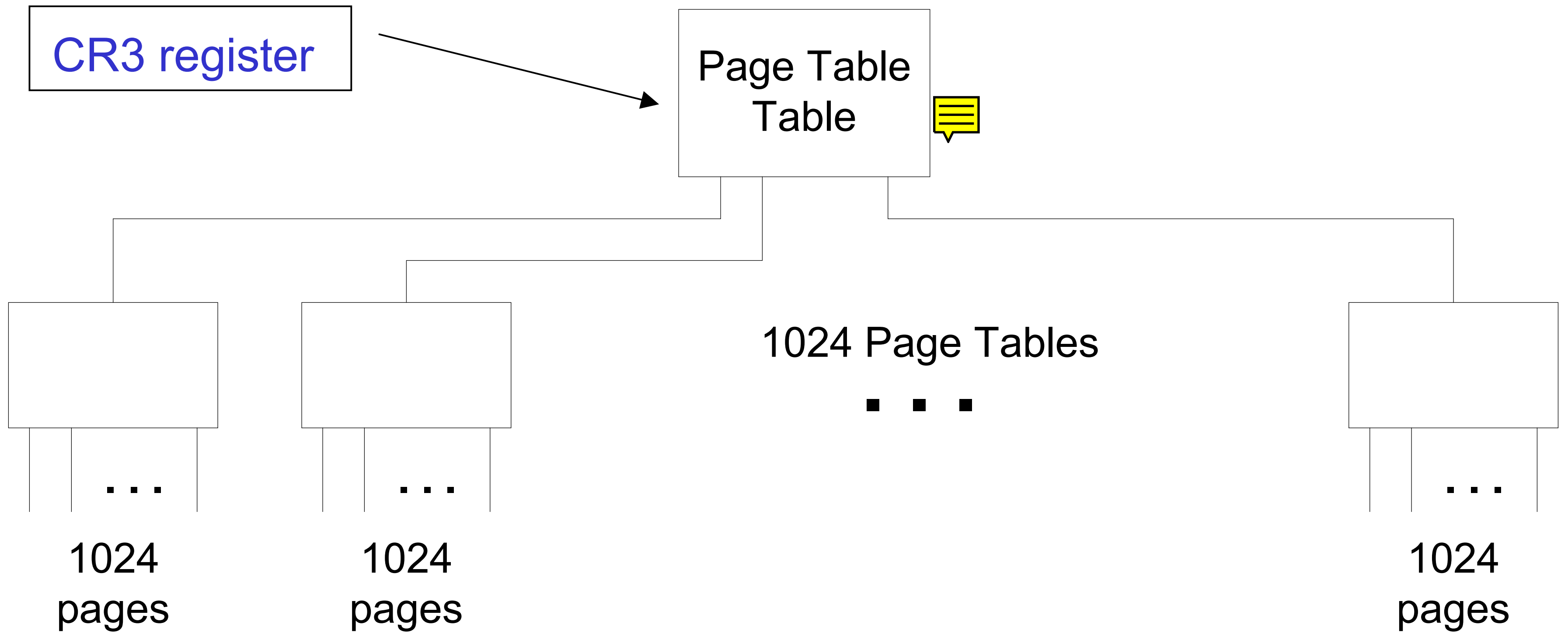
★ Operating sys. may maintain one page table per process

★ A dirty bit is used to track modified pages for copy back

Indicates whether the virtual page is in main memory or not



Linux 2-Level Page Table



- The CR3 register is designated for pointing to the first level page table
- The CR3 is part of the task state that needs to be saved at preemption

31 — 22	21 — 12	11 — 0
Index into Page Table Table	Index into Page Table	Index into Page

3.7.1. Linear Address Translation (4-KByte Pages)

Figure 3-12 shows the page directory and page-table hierarchy when mapping linear addresses to 4-KByte pages. The entries in the page directory point to page tables, and the entries in a page table point to pages in physical memory. This paging method can be used to address up to 2^{20} pages, which spans a linear address space of 2^{32} bytes (4 GBytes).

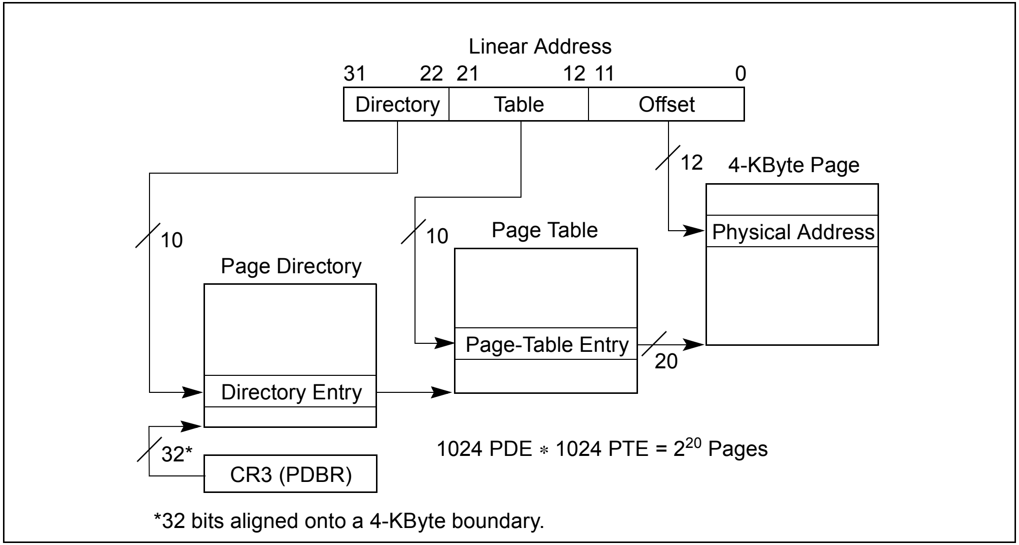


Figure 3-12. Linear Address Translation (4-KByte Pages)

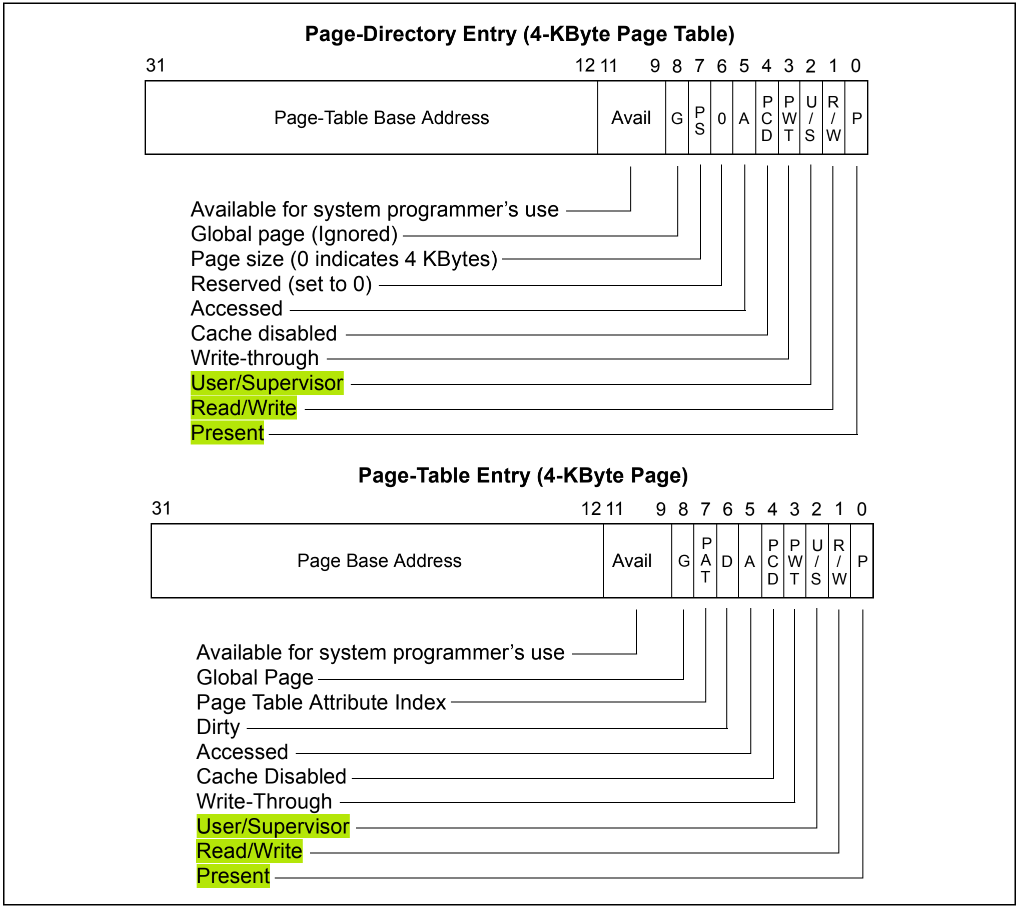


Figure 3-14. Format of Page-Directory and Page-Table Entries for 4-KByte Pages and 32-Bit Physical Addresses

Virtual Memory: Problems Solved

- **Not enough physical memory**
  - ◇ Uses disk space to simulate extra memory
  - ◇ Pages not being used can be swapped out (how and when you'll learn in CMSC 421 Operating Systems)
  - ◇ Thrashing: pages constantly written to and retrieved from disk (time to buy more RAM)
- **Fragmentation**
  - ◇ Contiguous blocks of virtual memory do not have to map to contiguous sections of real memory
- **Memory protection**
  - ◇ Each process has its own page table
  - ◇ Shared pages are read-only
  - ◇ User processes cannot alter the page table (must be supervisor)

Memory Protection

- Prevents one process from reading from or writing to memory used by another process
- Privacy in a multiple user environments
- Operating system stability
  - ◇ Prevents user processes (applications) from altering memory used by the operating system
  - ◇ One application crashing does not cause the entire OS to crash

Virtual Memory: too slow?

- **Address translation is done in hardware**

  In the middle of the fetch execute cycle for:

  ```
  MOV     EAX, [buffer]
  ```

  the physical address of buffer is computed in hardware.

- **Recently computed page locations are cached in the translation lookaside buffer (TLB)**
- **Page faults *are* very expensive (millions of cycles)**
- **Operating systems for personal computers have ~~only recently~~ added memory protection**

NEXT TIME

- Review