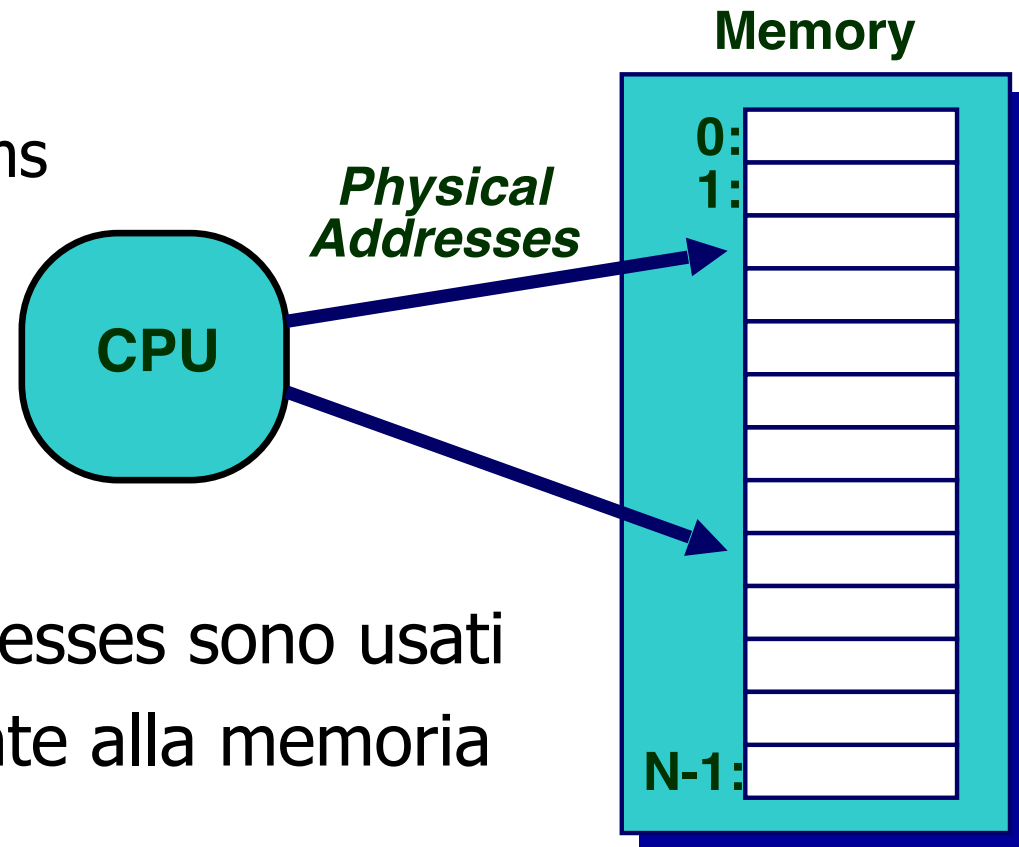


Memoria Virtuale

Andrea Bartolini – a.bartolini@unibo.it

A System with Physical Memory Only

- Examples:
 - ❑ most Cray machines
 - ❑ early PCs
 - ❑ many embedded systems



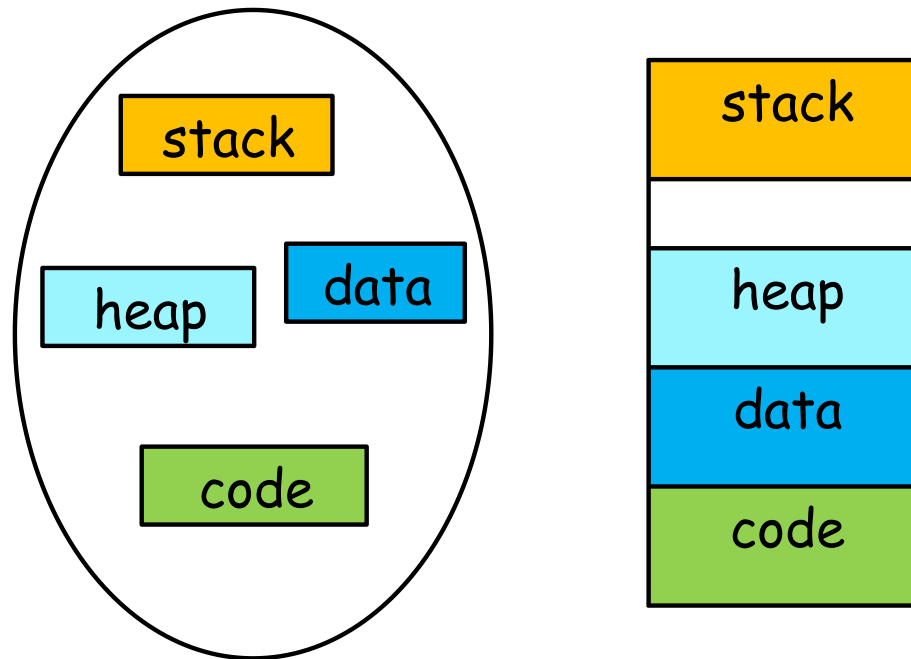
CPU's load or store addresses sono usati per accedere direttamente alla memoria

Isolation

- Applications' instructions and data are stored in main memory.
 - And cached when required
- While this is fine for a single application, it poses a security risk with multiple programs.
 - An untrusted program could read sensitive data.
 - Or corrupt the state of another application, including taking control of it
- We provide an operating system to prevent this happening.
- What hardware support can we provide to improve the performance of the OS?

Segmentazione

- Dividere lo spazio degli indirizzi virtuali in segmenti logici separati; ognuno fa parte del mem fisico



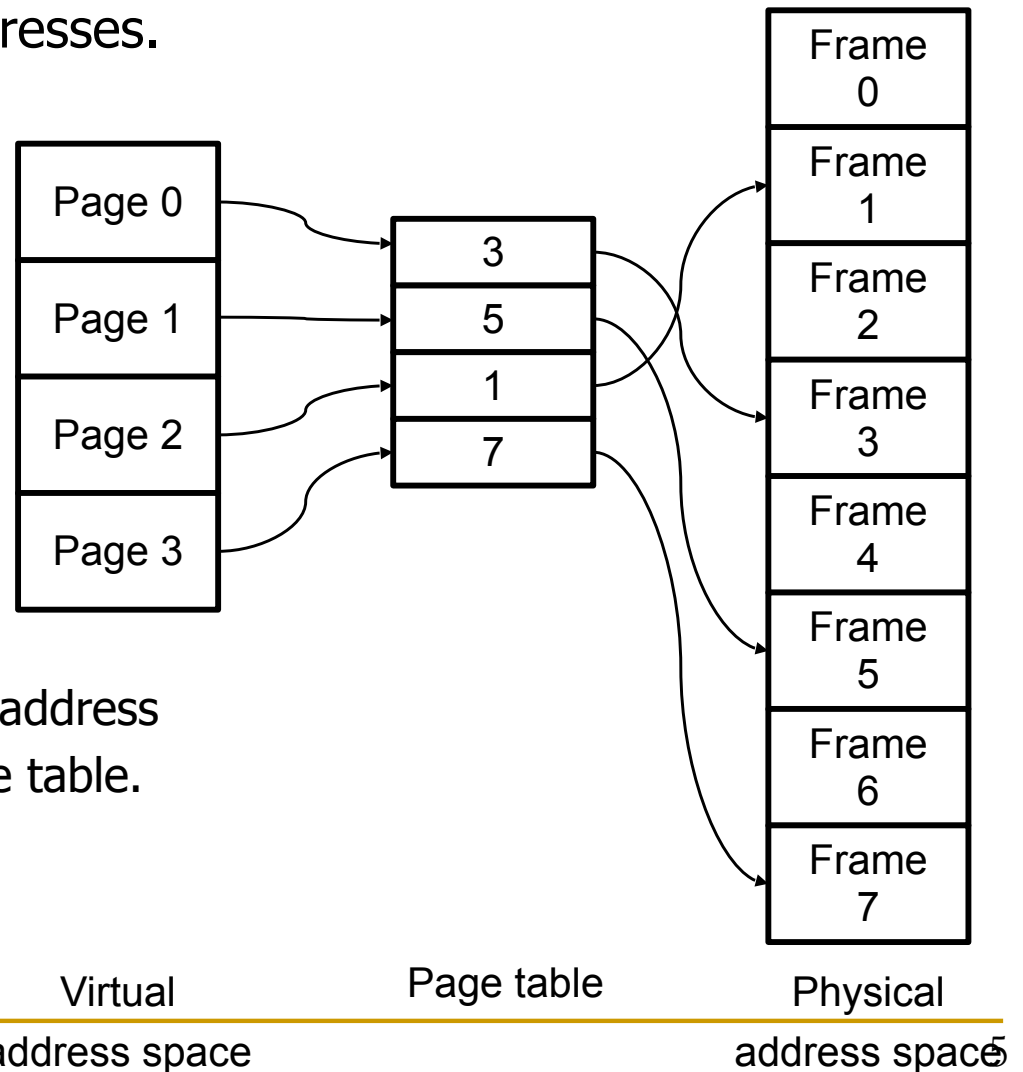
Problema:

- Segmenti di dimensione variabile e potenzialmente larghi
⇒ frammentazione.
 - Segmenti sono contigui nella memoria
 - Il programmatore deve preoccuparsi che i segmenti possano essere contenuti nella memoria fisica disponibile.
-

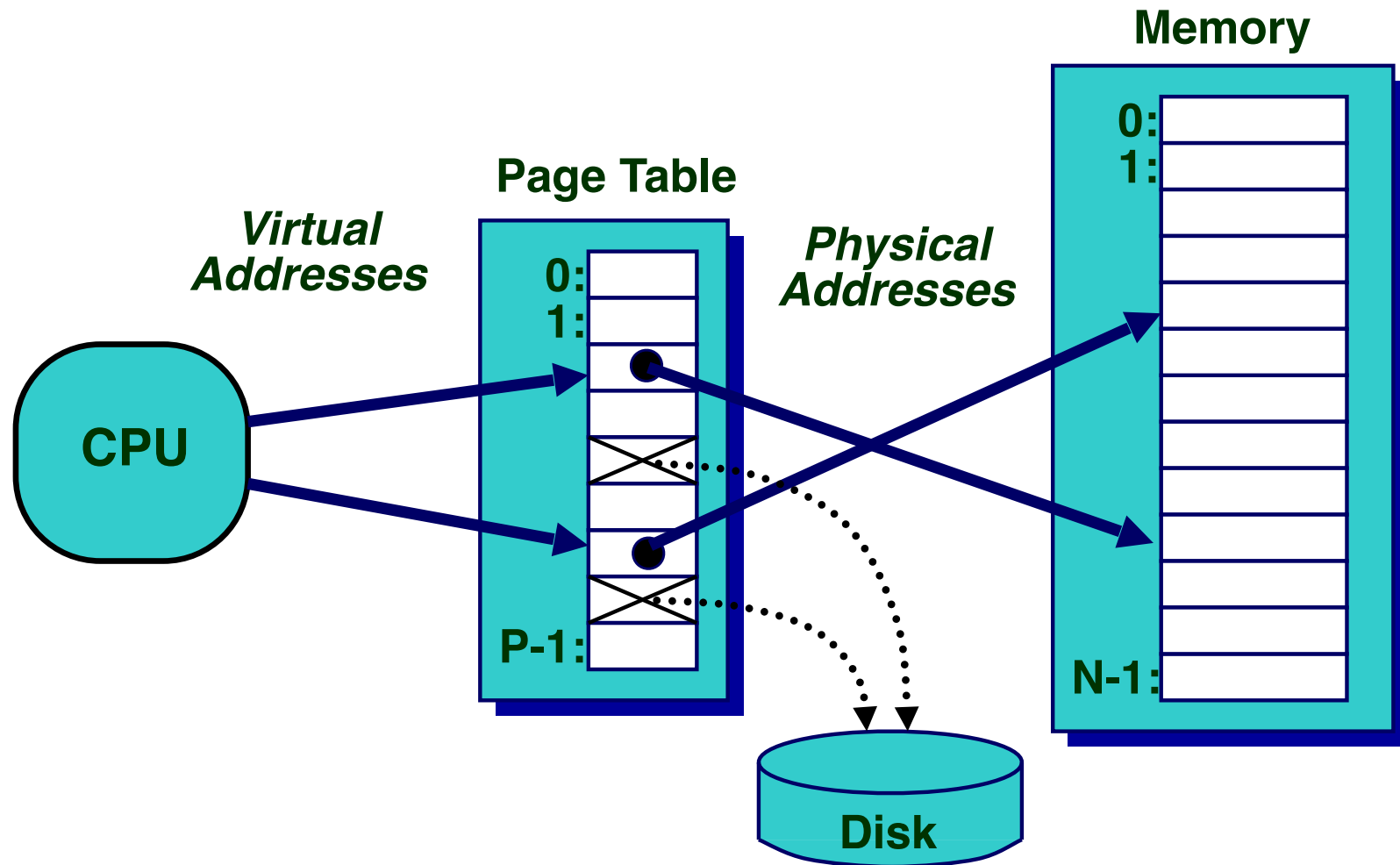
Paging

- The OS forbids direct access to memory.
- Instead, it introduces a layer of indirection.
 - ❑ Applications see memory as a range of virtual addresses.
 - ❑ The OS maps these to physical addresses.

- Paging is one method to achieve this.
 - ❑ Physical memory is split into fixed-sized frames.
 - ❑ Virtual memory is split into same-sized pages.
 - ❑ Each page is mapped to one frame.
 - Using the high-order bits from the address
 - This information is kept in the page table.



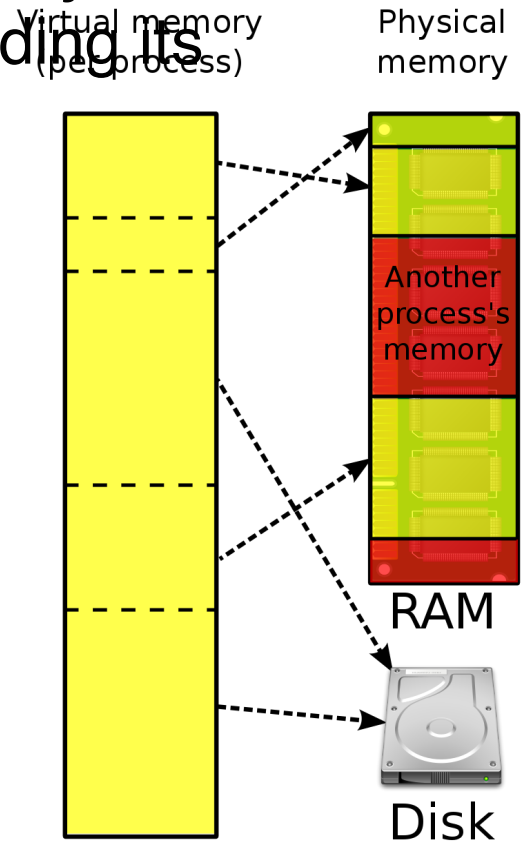
A System with Virtual Memory (Page based)



- **Address Translation:** hardware converte gli indirizzi virtuali in indirizzi fisici tramite una lookup table gestita dal sistema operativo (page table)

Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
 - ❑ Managed jointly by CPU hardware and the operating system (OS)
- Programs (OS processes) share main memory
 - ❑ Each gets a private virtual address space holding its frequently used code and data
 - ❑ Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
 - ❑ VM “block” is called a page
 - ❑ VM translation “miss” is called a *page fault*

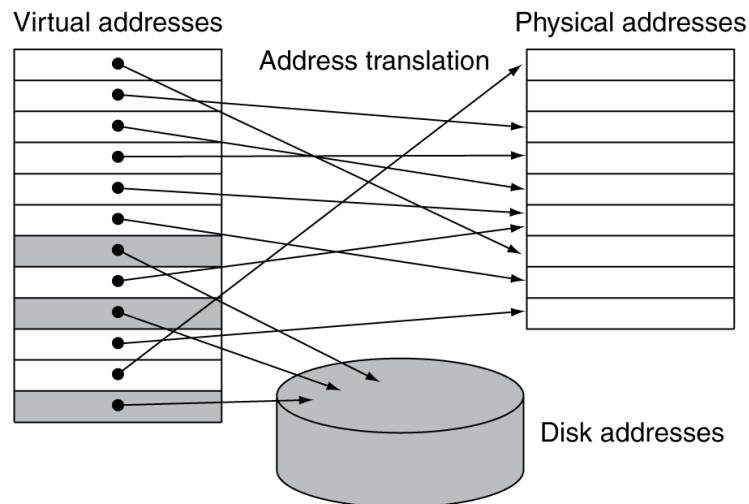


Recall: Virtual Memory Definitions

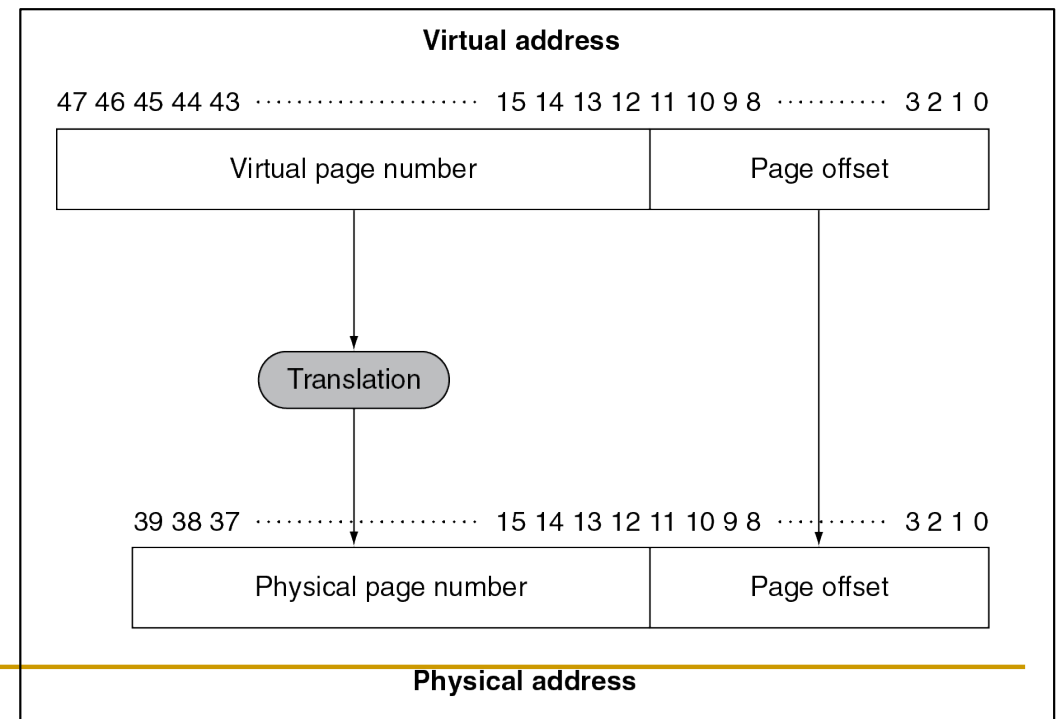
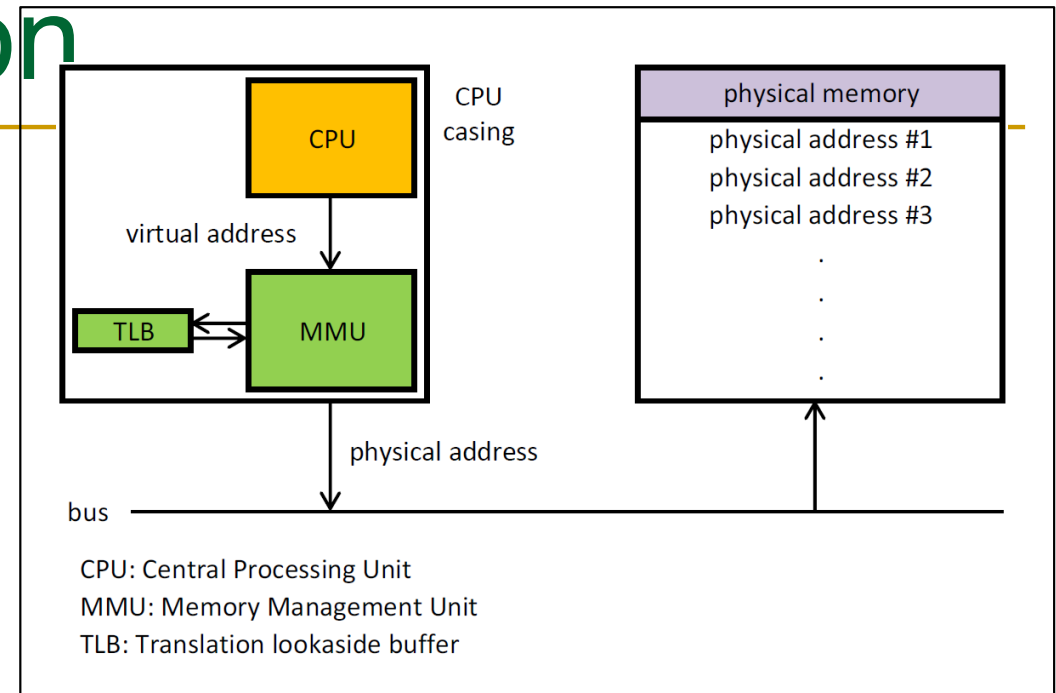
- **Page size**: quantità di memoria trasferita dal disco rigido alla DRAM contemporaneamente
- **Address translation**: Determinazione dell'indirizzo fisico dall'indirizzo virtuale
- **Page table**: lookup table utilizzata per tradurre gli indirizzi virtuali in indirizzi fisici (e trovare dove si trovano i dati associati)

Address Translation

- Fixed-size pages (e.g., 4K)



A *memory management unit (MMU)* is a computer hardware unit having all memory references passed through itself, primarily performing the translation of virtual memory addresses to physical addresses.



Recall: Virtual Memory Example

- **System:**
 - ❑ Virtual memory size: 2 GB = 2^x bytes
 - ❑ Physical memory size: 128 MB = 2^x bytes
 - ❑ Page size: 4 KB = 2^x bytes

Recall: Virtual Memory Example

- **System:**
 - ❑ Virtual memory size: 2 GB = 2^{31} bytes
 - ❑ Physical memory size: 128 MB = 2^{27} bytes
 - ❑ Page size: 4 KB = 2^{12} bytes

Recall: Virtual Memory Example

- **System:**

- ❑ Virtual memory size: 2 GB = 2^{31} bytes
- ❑ Physical memory size: 128 MB = 2^{27} bytes
- ❑ Page size: 4 KB = 2^{12} bytes

- **Organization:**

- ❑ Virtual address: x bits
- ❑ Physical address: x bits
- ❑ Page offset: x bits

Recall: Virtual Memory Example

- **System:**

- ❑ Virtual memory size: 2 GB = 2^{31} bytes
- ❑ Physical memory size: 128 MB = 2^{27} bytes
- ❑ Page size: 4 KB = 2^{12} bytes

- **Organization:**

- ❑ Virtual address: 31 bits
- ❑ Physical address: 27 bits
- ❑ Page offset: 12 bits

Recall: Virtual Memory Example

- **System:**

- ❑ Virtual memory size: 2 GB = 2^{31} bytes
- ❑ Physical memory size: 128 MB = 2^{27} bytes
- ❑ Page size: 4 KB = 2^{12} bytes

- **Organization:**

- ❑ Virtual address: 31 bits
- ❑ Physical address: 27 bits
- ❑ Page offset: 12 bits
- ❑ # Virtual pages = ?
- ❑ # Physical pages = ?

Recall: Virtual Memory Example

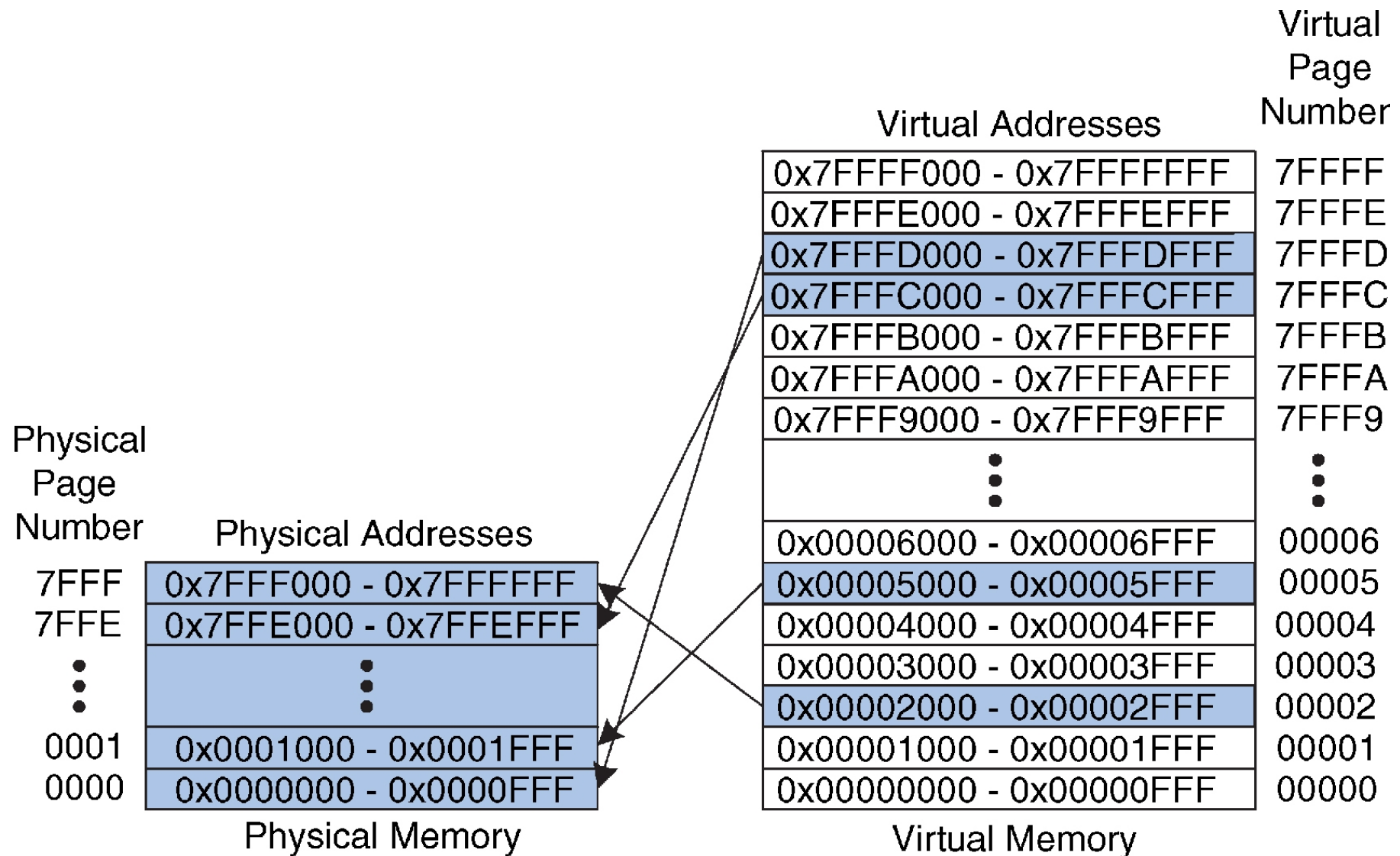
- **System:**

- ❑ Virtual memory size: 2 GB = 2^{31} bytes
- ❑ Physical memory size: 128 MB = 2^{27} bytes
- ❑ Page size: 4 KB = 2^{12} bytes

- **Organization:**

- ❑ Virtual address: **31** bits
- ❑ Physical address: **27** bits
- ❑ Page offset: **12** bits
- ❑ # Virtual pages = $2^{31}/2^{12} = 2^{19}$ (VPN = 19 bits)
- ❑ # Physical pages = $2^{27}/2^{12} = 2^{15}$ (PPN = 15 bits)

Recall: Virtual Memory Mapping Example



How Do We Translate Addresses?

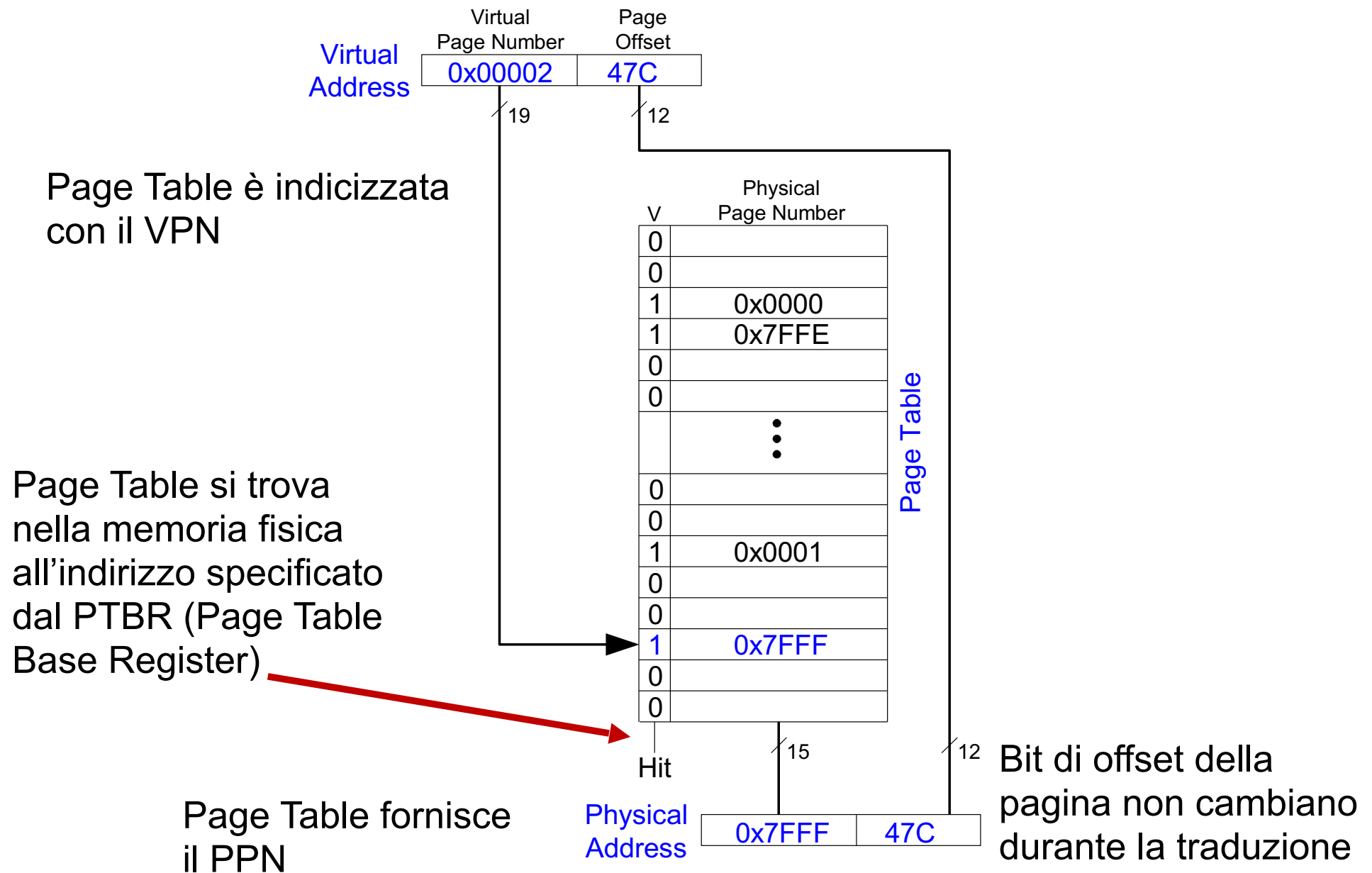
- **Page table**

- ❑ Una entry per ciascuna pagina virtuale (virtual page)

- Ciascuna **page table entry** deve possedere:

- ❑ **Valid bit**: dice se la pagina virtuale si trova in memoria fisica (in caso contrario, deve essere recuperata dal disco rigido)
 - ❑ **Physical page number**: dove si trova la pagina virtuale nella memoria fisica
 - ❑ (Replacement policy, dirty bits)

Page Table Address Translation Example



Page Table Address Translation Example 1

- Qual è l'indirizzo fisico associato all'indirizzo virtuale 0x5F20?
- Dobbiamo prima trovare la page table entry contenente la traduzione corrispondente al VPN
- Cercare la PTE all'indirizzo
 - $PTBR + VPN * PTE_size$

V	Physical Page Number
0	
0	
1	0x0000
1	0x7FFE
0	
0	
	⋮
0	
0	
1	0x0001
0	
0	
1	0x7FFF
0	
0	

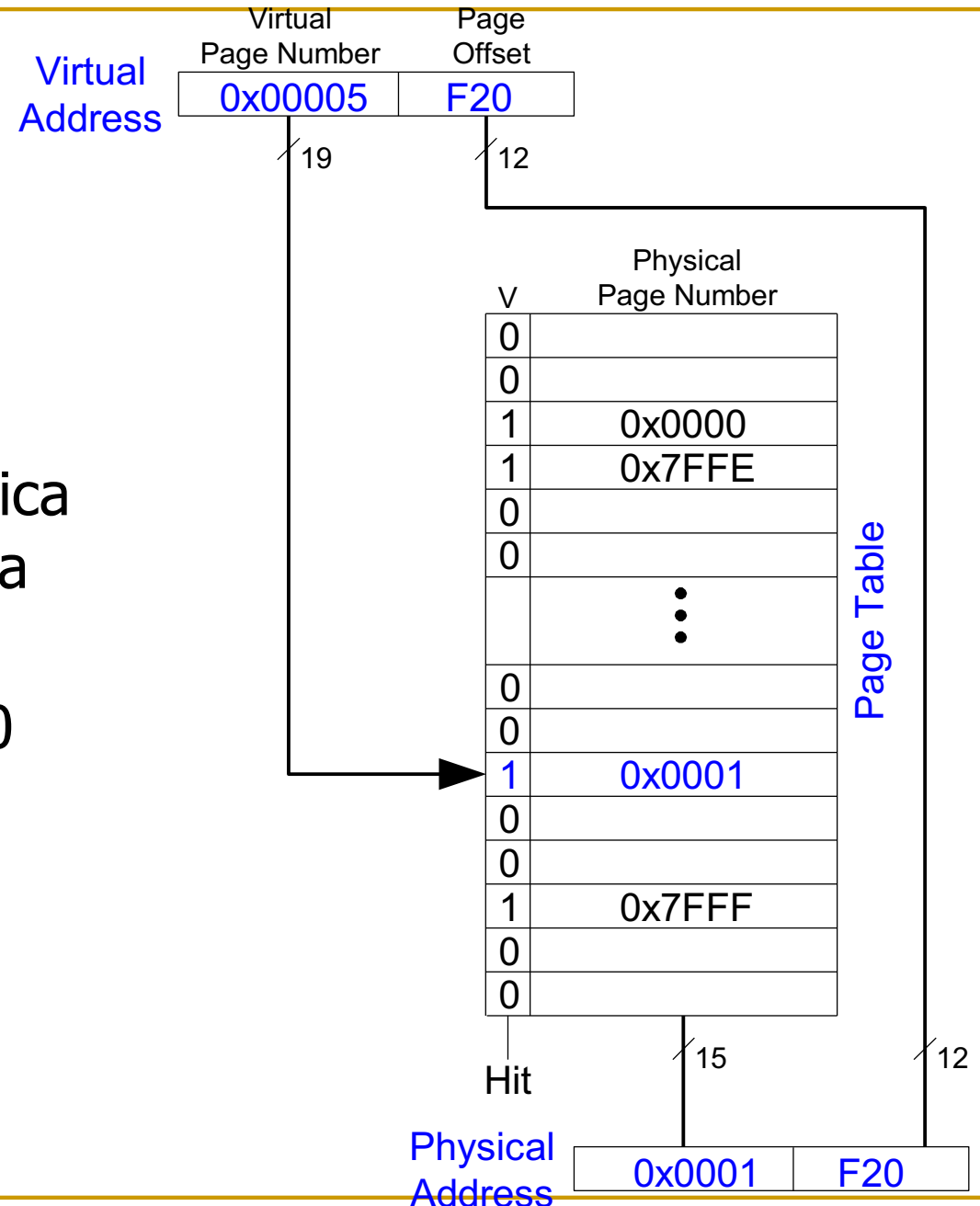
Page Table

Hit

15

Page Table Address Translation Example 1

- Qual è l'indirizzo fisico dell'indirizzo virtuale 0x5F20?
 - VPN = 5
 - Entry 5 in page table indica che VPN 5 è mappato alla pagina fisica 1
 - L'indirizzo fisico è 0x1F20



Page Table Address Translation Example 2

- Qual è l'indirizzo fisico dell'indirizzo virtuale 0x73E0?

V	Physical Page Number
0	
0	
1	0x0000
1	0x7FFE
0	
0	
	⋮
0	
0	
1	0x0001
0	
0	
1	0x7FFF
0	
0	

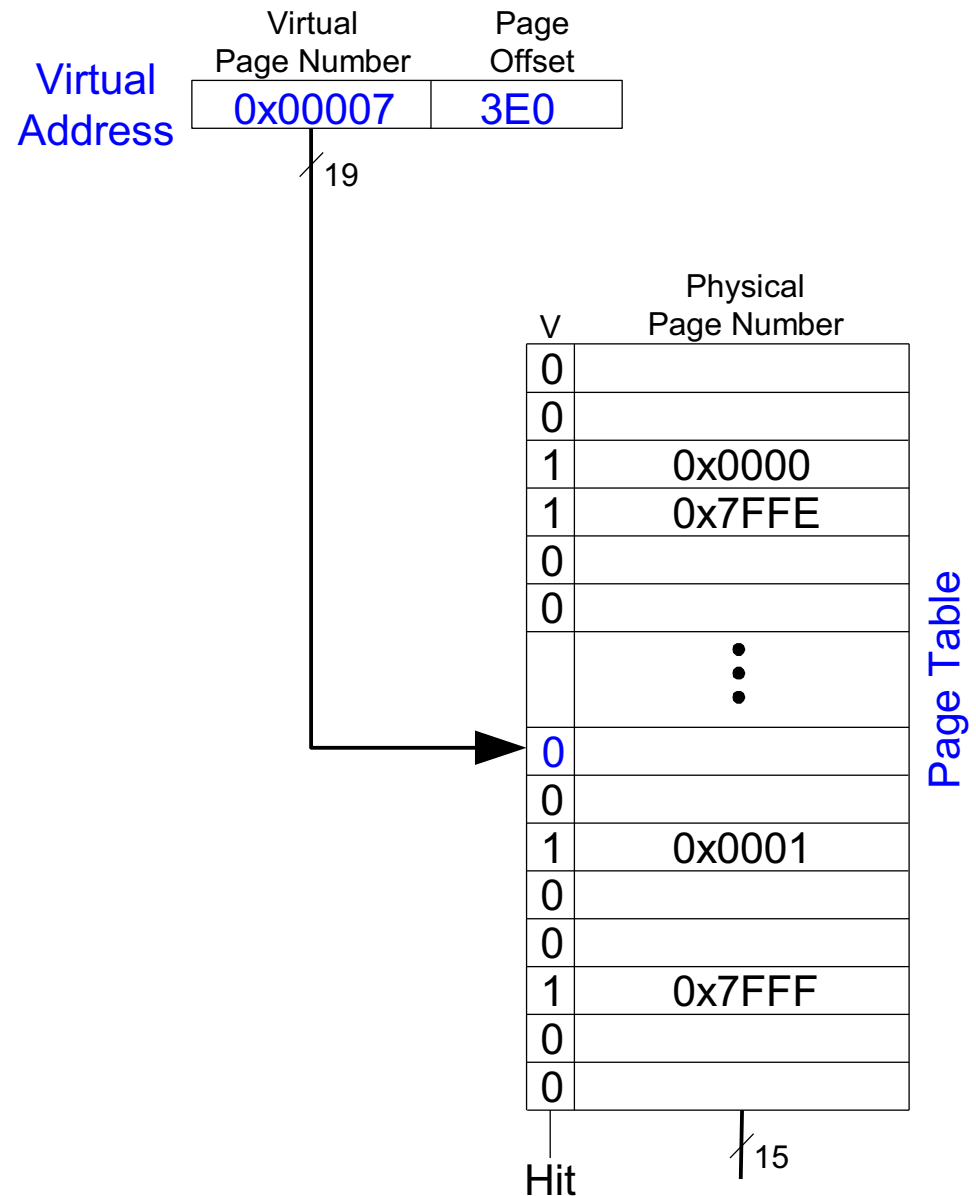
Page Table

Hit

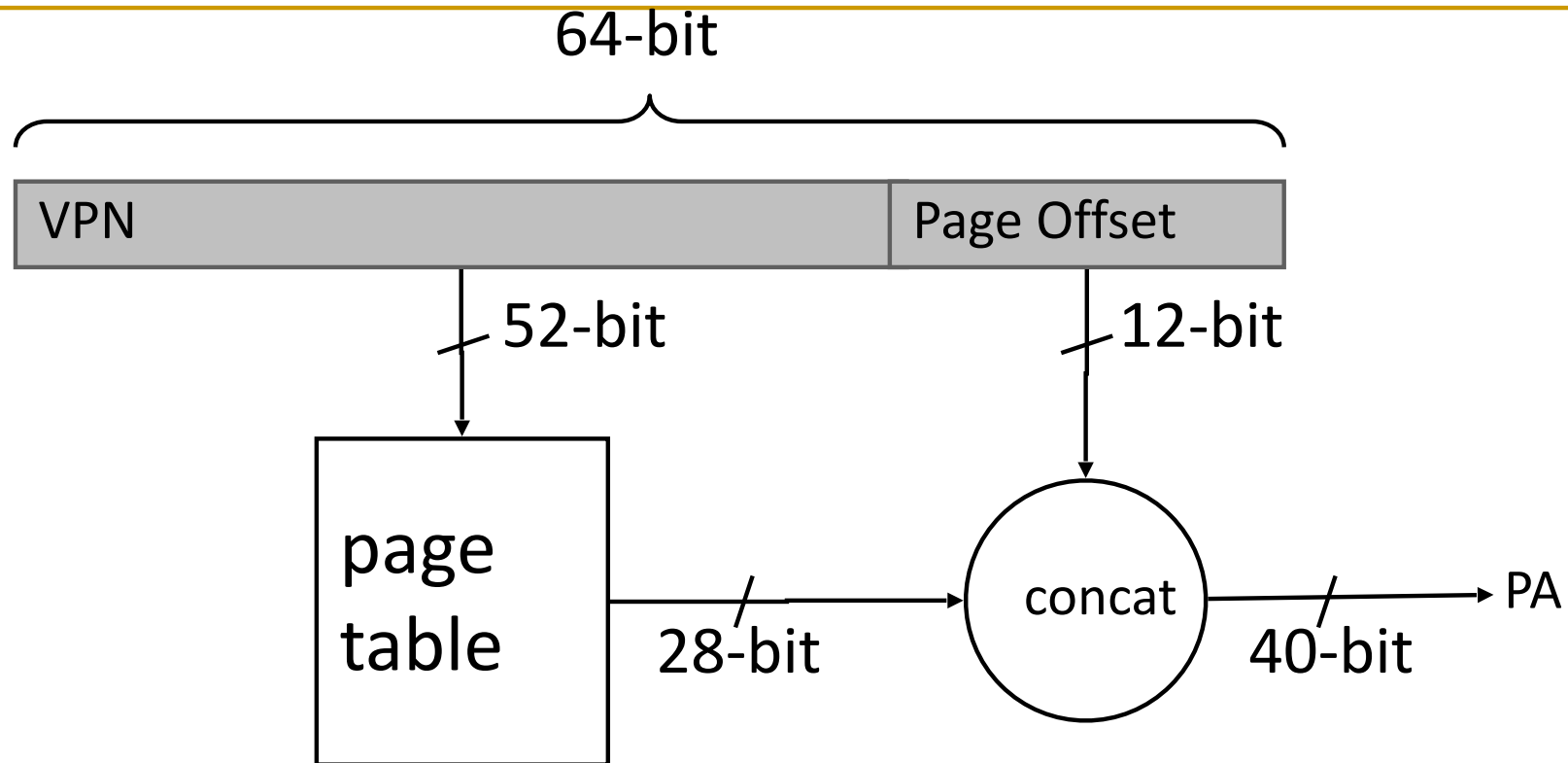
15

Page Table Address Translation Example 2

- Qual è l'indirizzo fisico dell'indirizzo virtuale 0x73E0?
 - VPN = 7
 - Entry 7 nella page table non è valido, la pagina non è in memoria fisica
 - La pagina virtuale deve essere scambiata in memoria fisica dal disco (page swapping)

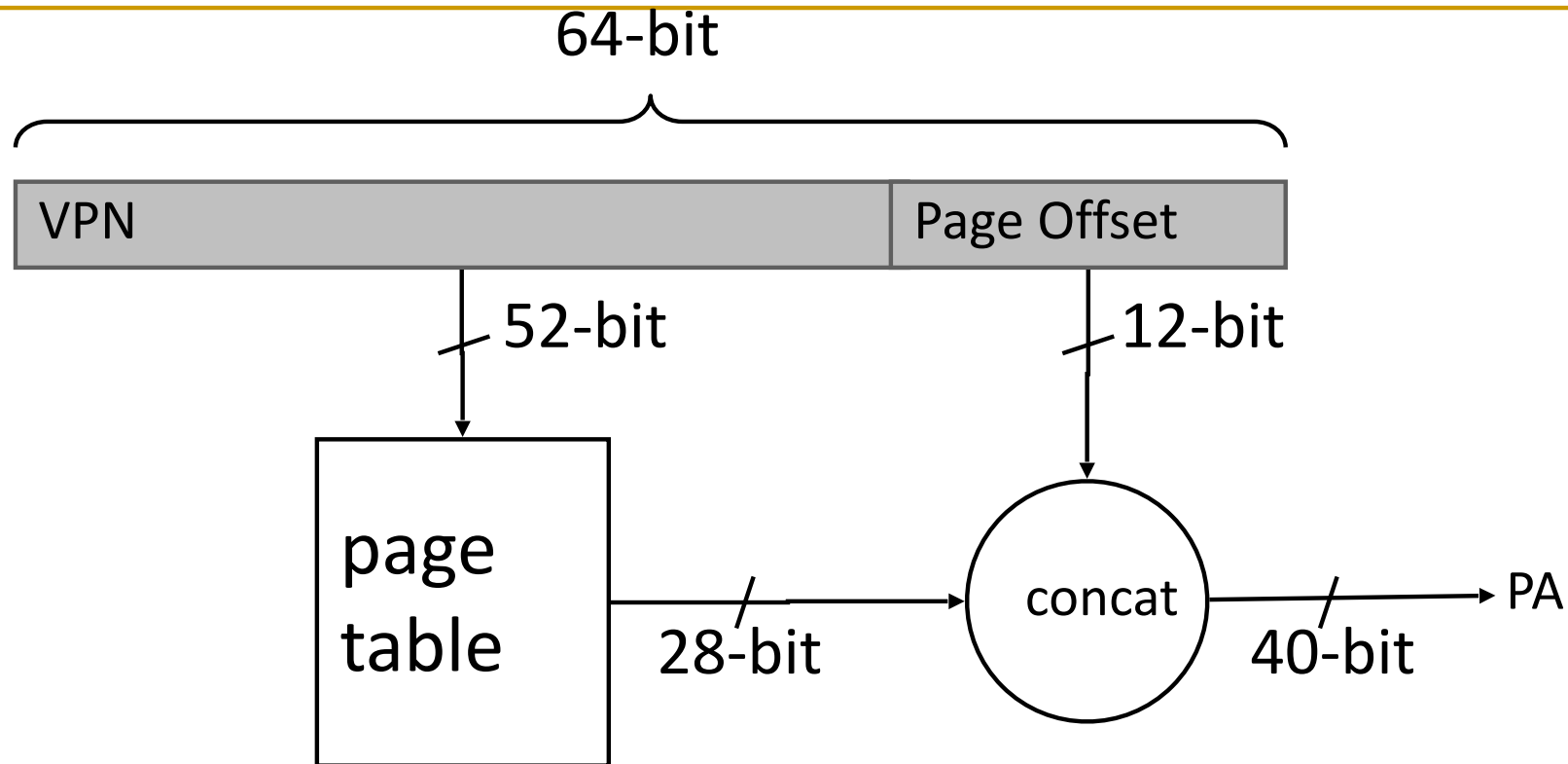


Issue: Page Table Size



- 64-bit VA e 40-bit PA, quanto è grande la tabella delle pagine?

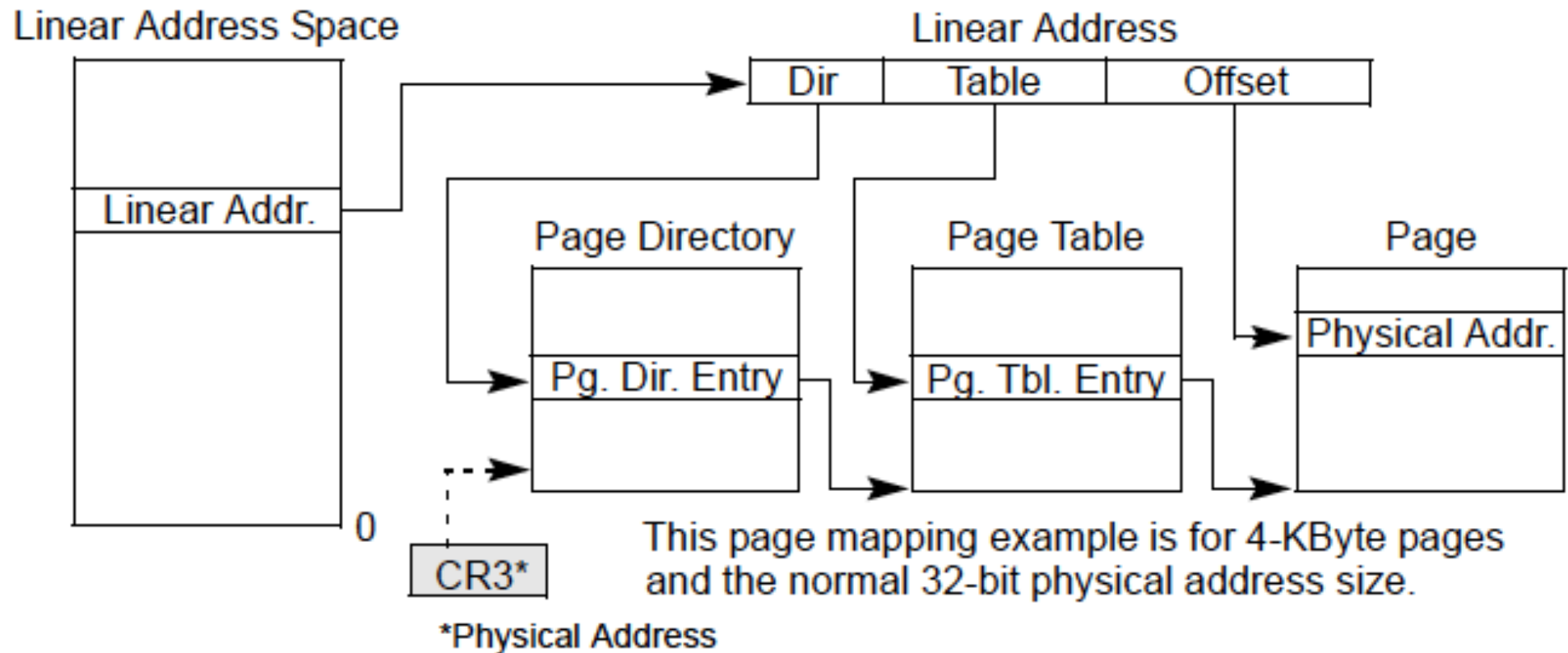
Issue: Page Table Size



- 64-bit VA e 40-bit PA, quanto è grande la tabella delle pagine?
 - 2^{52} entries x ~ 4 bytes $\approx 2^{54}$ bytes**
 - e questo è per un solo processo!
 - e il processo potrebbe non utilizzare l'intero spazio della memoria virtuale!

Solution: Multi-Level Page Tables

Example from the x86 architecture



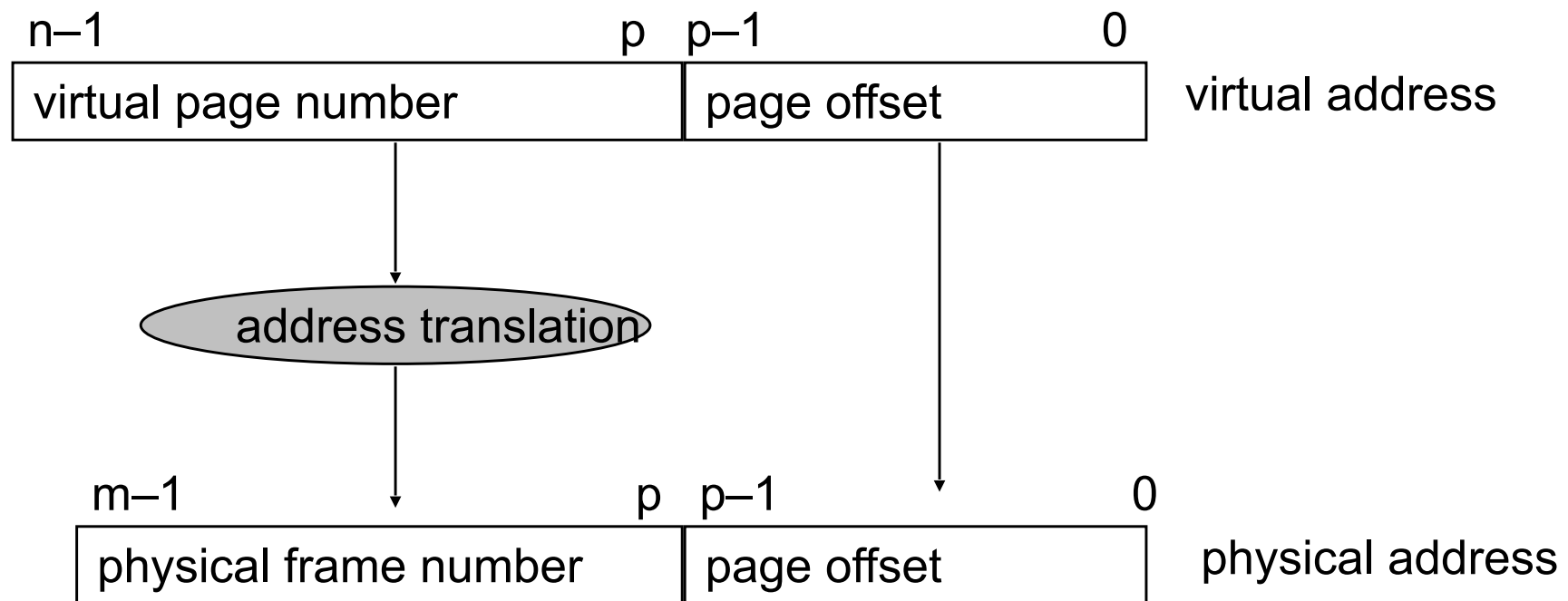
Address Translation

- Come ottenere l'indirizzo fisico da un indirizzo virtuale?
- Page size specified by the ISA
 - Today: 4KB, 8KB, 2GB, ... (small and large pages mixed together)
 - Trade-offs? (ricorda le lezioni sulle cache)
- Page Table contiene una entry per ogni pagina virtuale
- Called Page Table Entry (PTE)
 - What is in a PTE?

Address Translation (I)

- Parameters

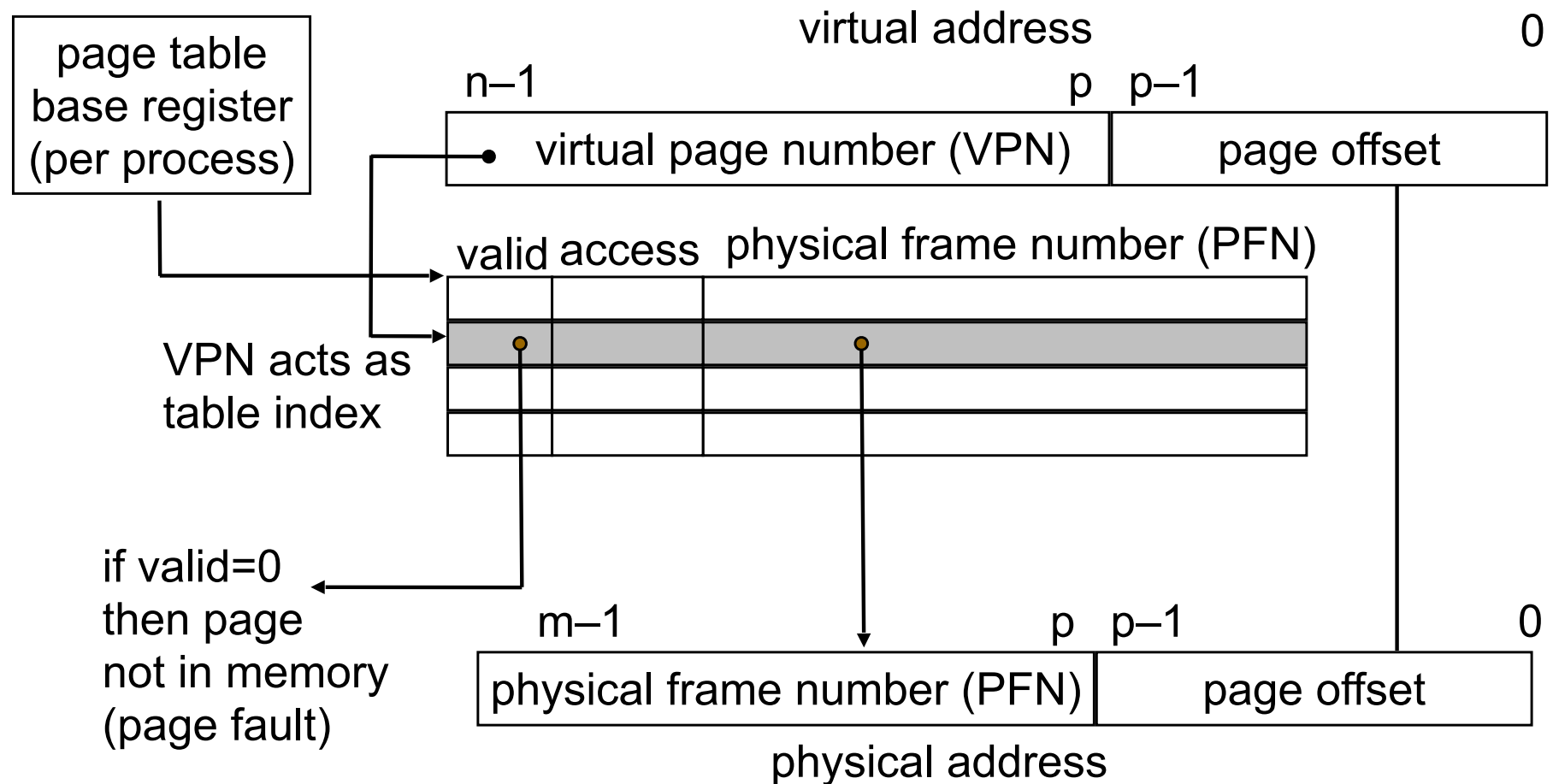
- $P = 2^p =$ page size (bytes).
- $N = 2^n =$ Virtual-address limit
- $M = 2^m =$ Physical-address limit



Page offset bits don't change as a result of translation

Address Translation (II)

- Separate (set of) page table(s) per process
- VPN forms index into page table (points to a page table entry)
- Page Table Entry (PTE) provides information about page



Page Table Challenges

- Challenge 1: **Page table è larga**
 - ▣ almeno una parte di essa deve essere nella memoria fisica
 - ▣ soluzione: Page Table a più livelli (gerarchiche)

Challenge 2: **Ogni istruzione di fetch o load/store richiede almeno due accessi alla memoria:**

1. uno per la traduzione degli indirizzi (lettura nella page table)
2. uno per accedere ai dati con l'indirizzo fisico (dopo la traduzione)

- Due accessi alla memoria per instruction fetch o load/store riduce notevolmente i tempi di esecuzione

A meno che non siamo intelligenti...

- ▣ → velocizzare la traduzione...

Translation Lookaside Buffer (TLB)

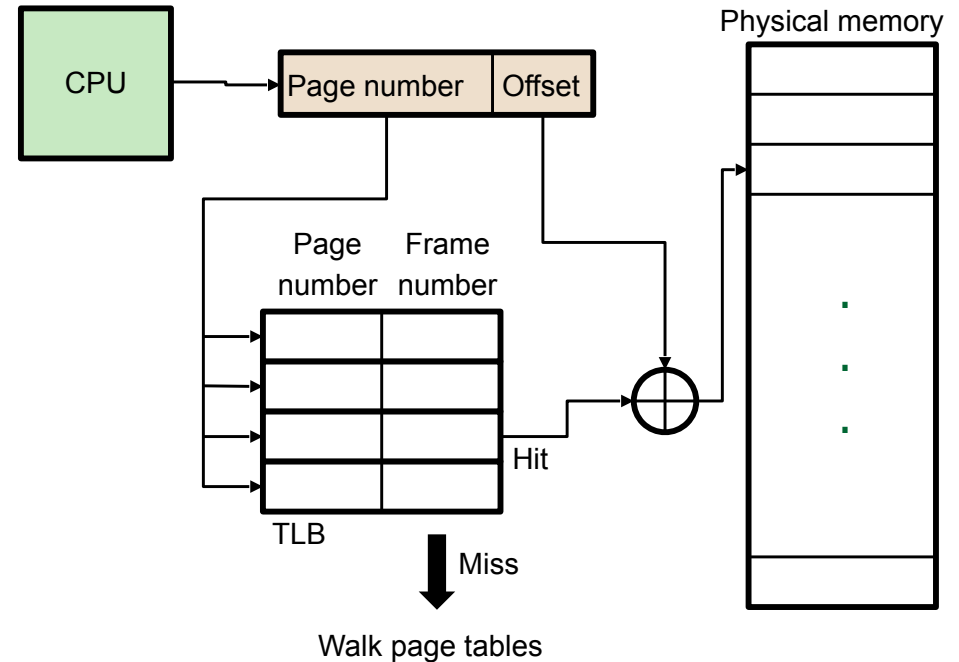
- Idea: Cache the page table entries (PTEs) in a hardware structure in the processor to speed up address translation
- Translation lookaside buffer (TLB)
 - Piccola cache delle traduzioni utilizzate più di recente (PTEs)
 - Riduce il numero di accessi alla memoria necessari per (soprattutto) instruction fetches e loads/stores ad uno.

Translation Lookaside Buffer (TLB)

- Accessi alla page table hanno un sacco di località temporale
 - ❑ Gli accessi ai dati hanno località temporale e spaziale
 - ❑ Large page size (say 4KB, 8KB, or even 1-2GB)
 - ❑ È probabile che istruzioni e load/store consecutive accedono alla stessa pagina
- TLB
 - ❑ Piccola: accessibile in ~ 1 ciclo
 - ❑ Normalmente 16 - 512 entries
 - ❑ Alta associatività
 - ❑ $> 95-99\%$ hit rates tipica (dipende dal workload)
 - ❑ Riduce il numero di accessi alla memoria per la maggior parte dei fetch a istruzioni e loads/stores

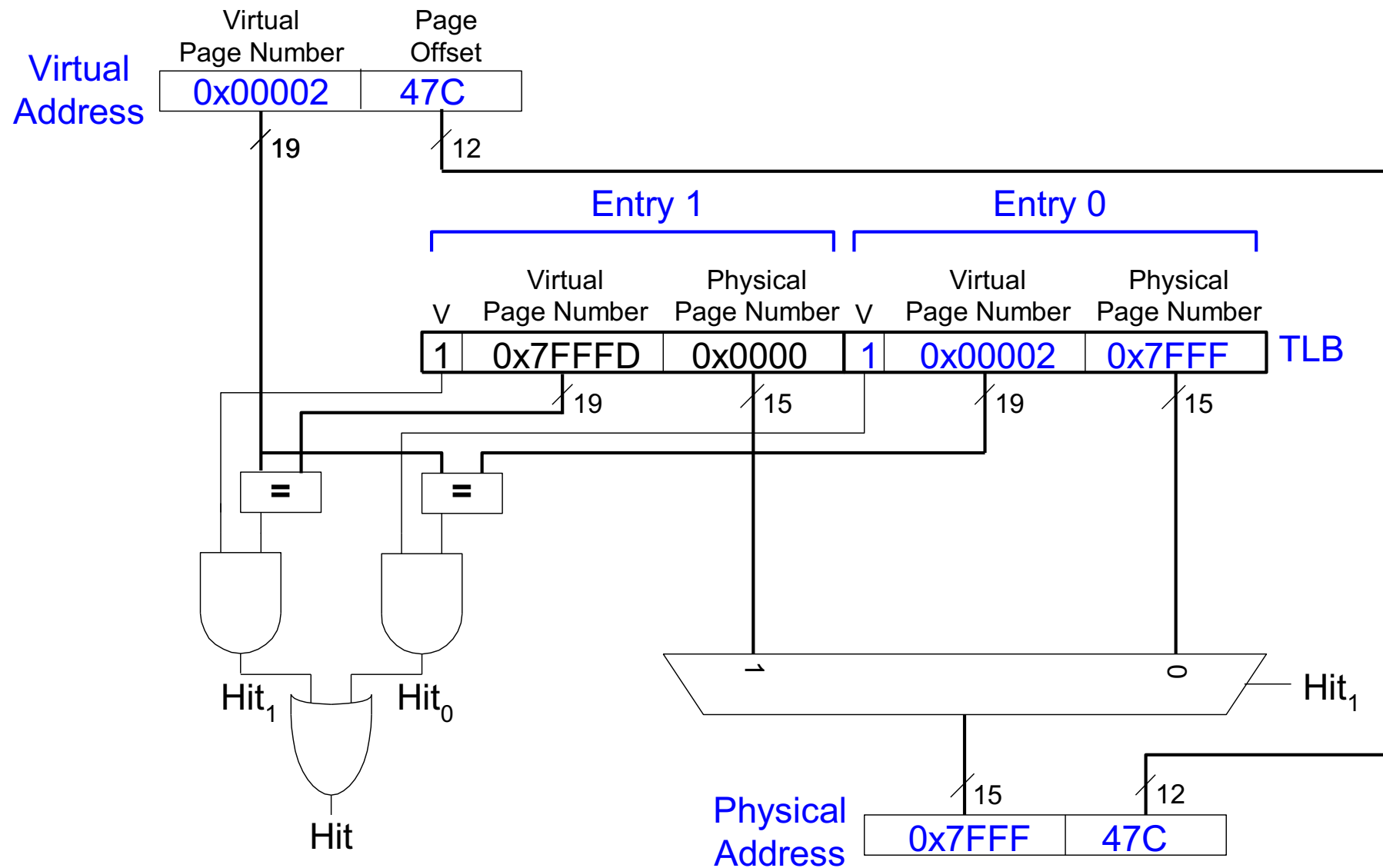
TLB

Translation lookaside buffer



- The TLB is a cache of page translations.
 - ▣ Provides access to recent virtual to physical address mappings quickly
- Each block is one or two page-table entries.
- TLBs are usually fully associative.
- On a hit, forward the physical address to the L1 cache.
- On a miss, the MMU will walk the page tables to find the translation.

Example Two-Entry TLB

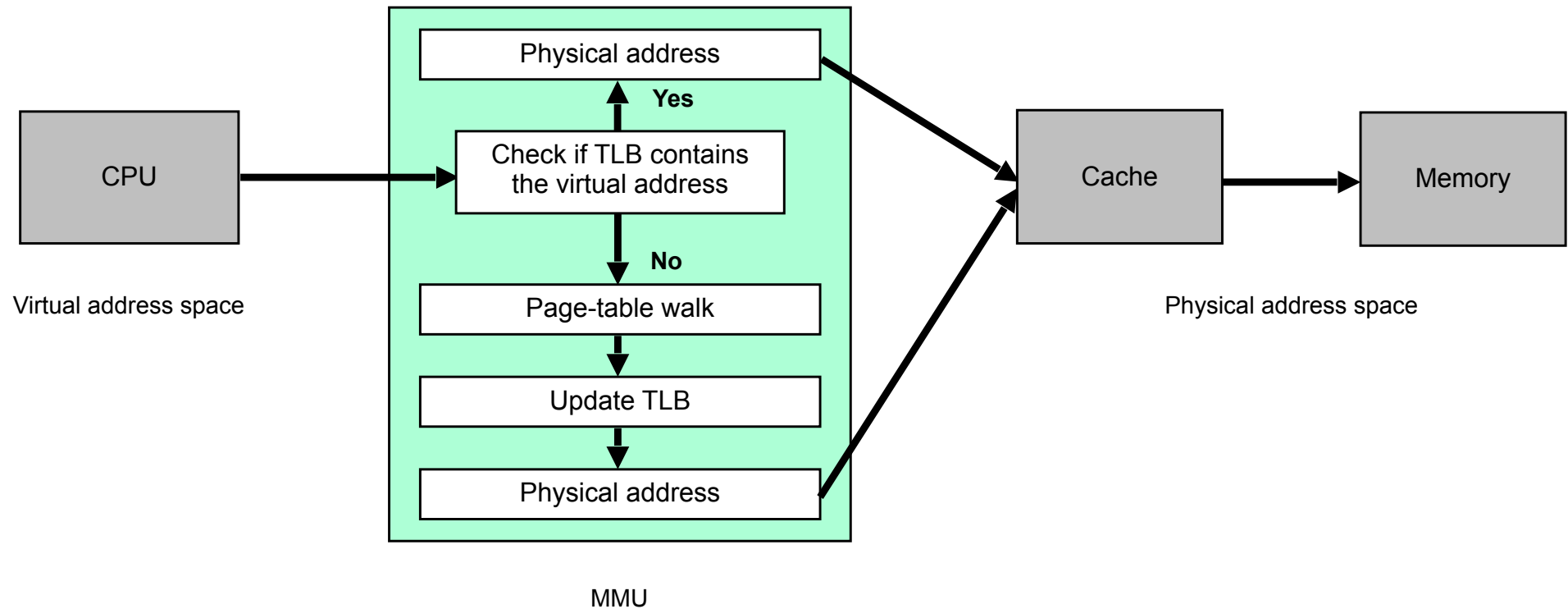


Virtual Memory Support and Examples

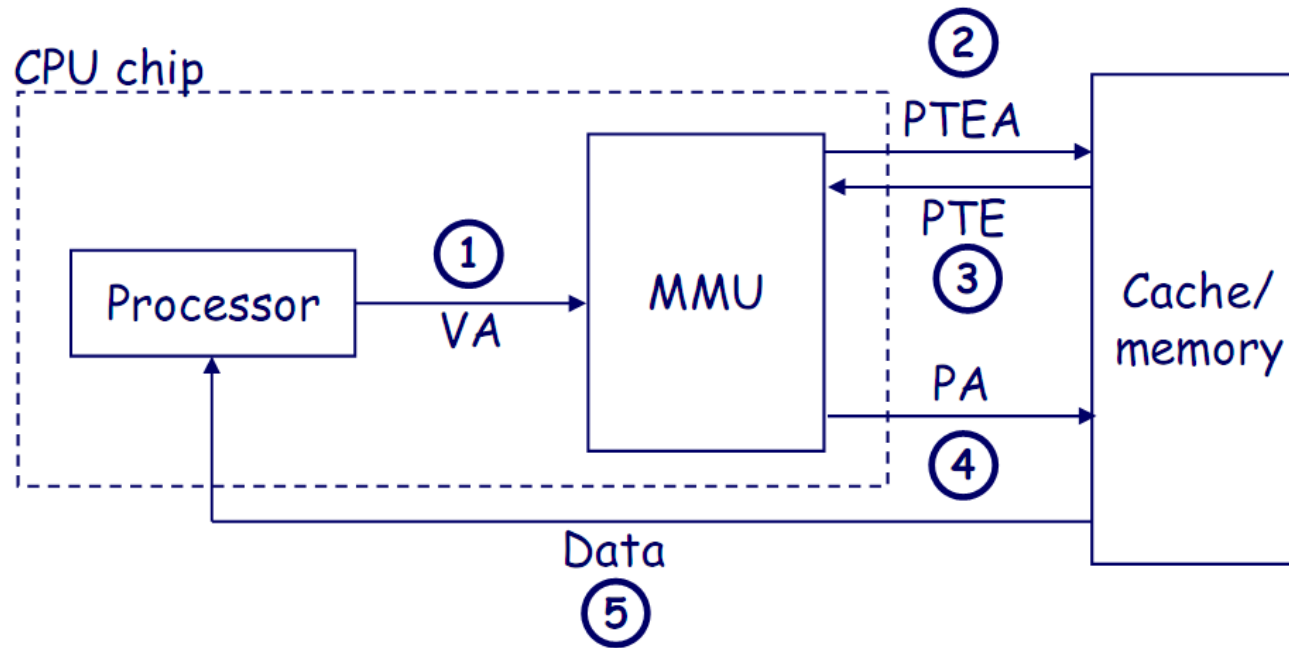
Supporting Virtual Memory

- Virtual memory **richiede supporto sia HW+SW**
 - Page Table è in memoria
 - Può essere memorizzata tramite una cache in speciali strutture hardware chiamate Translation Lookaside Buffers (TLBs)
- Il componente hardware è denominato **MMU** (memory management unit)
 - Includes Page Table Base Register(s), TLBs, page walkers
- **E' lavoro del software** sfruttare la MMU per:
 - Popolare la page tables, decider cosa rimuovere in physical memory
 - Modificare il Page Table Register durante context switch (per utilizzare la page table relativa al thread in esecuzione)
 - Gestire i page faults ed assicurare il mapping corretto.

Overview of Memory Access Using an MMU

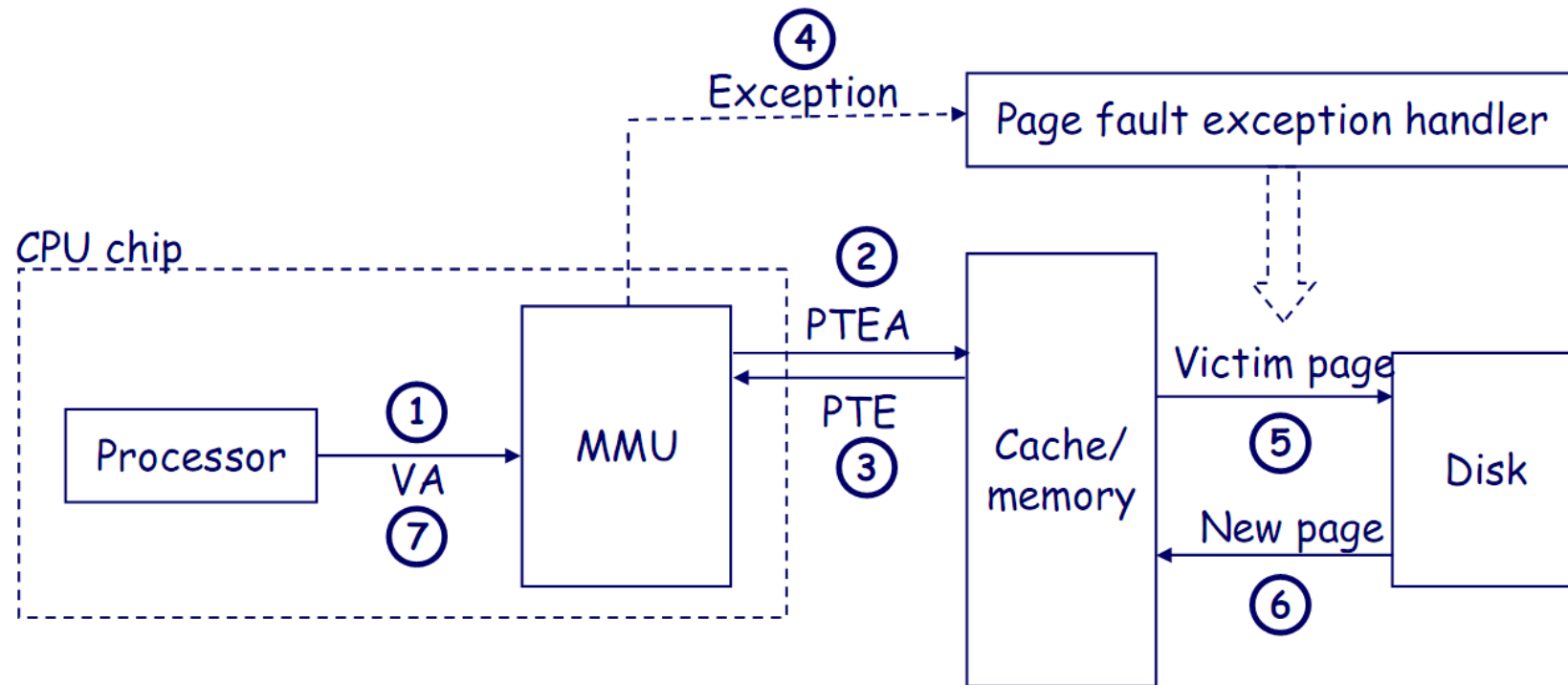


Address Translation: Page Hit



- 1) Processor sends virtual address to MMU
- 2-3) MMU fetches PTE from page table in memory
- 4) MMU sends physical address to L1 cache
- 5) L1 cache sends data word to processor

Address Translation: Page Fault

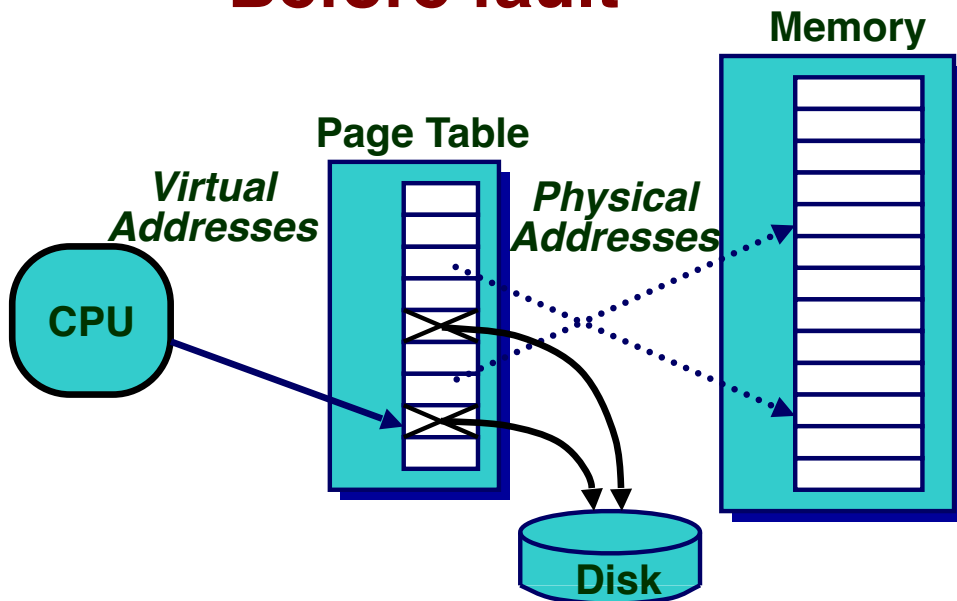


- 1) Processor sends virtual address to MMU
- 2-3) MMU fetches PTE from page table in memory
- 4) Valid bit is zero, so MMU triggers page fault exception
- 5) Handler identifies victim, and if dirty pages it out to disk
- 6) Handler pages in new page and updates PTE in memory
- 7) Handler returns to original process, restarting faulting instruction.

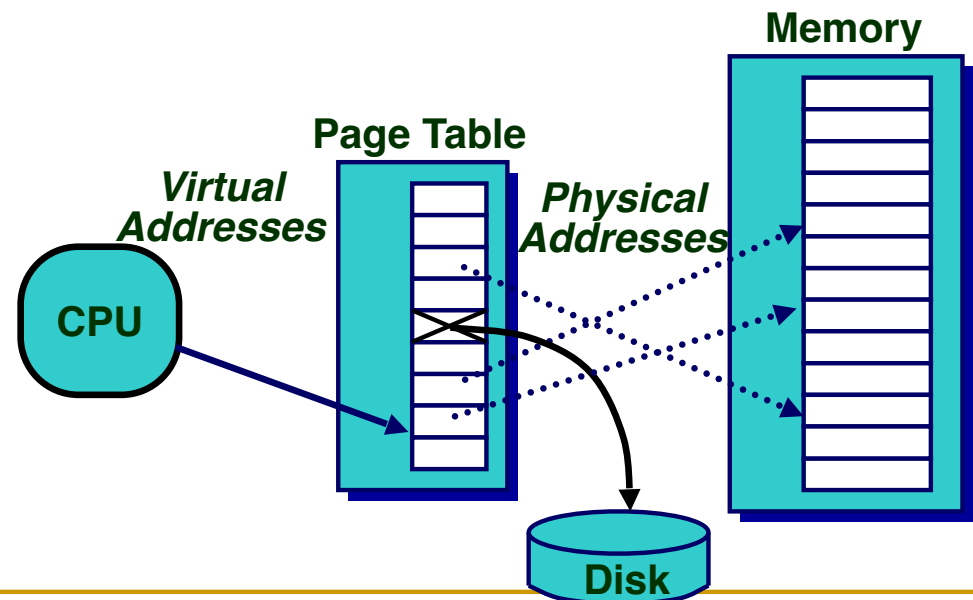
Page Fault (“A Miss in Physical Memory”)

- Se una pagina non è in memoria fisica ma disco
 - Page table entry indica che la pagina virtuale non è in memoria
 - L'accesso alla pagina genera una page fault exception
 - Il trap handler dell'OS copia i dati dal disk alla memoria
 - Altri processi possono continuare l'esecuzione
 - Il sistema operativo ha il pieno controllo sul posizionamento

Before fault

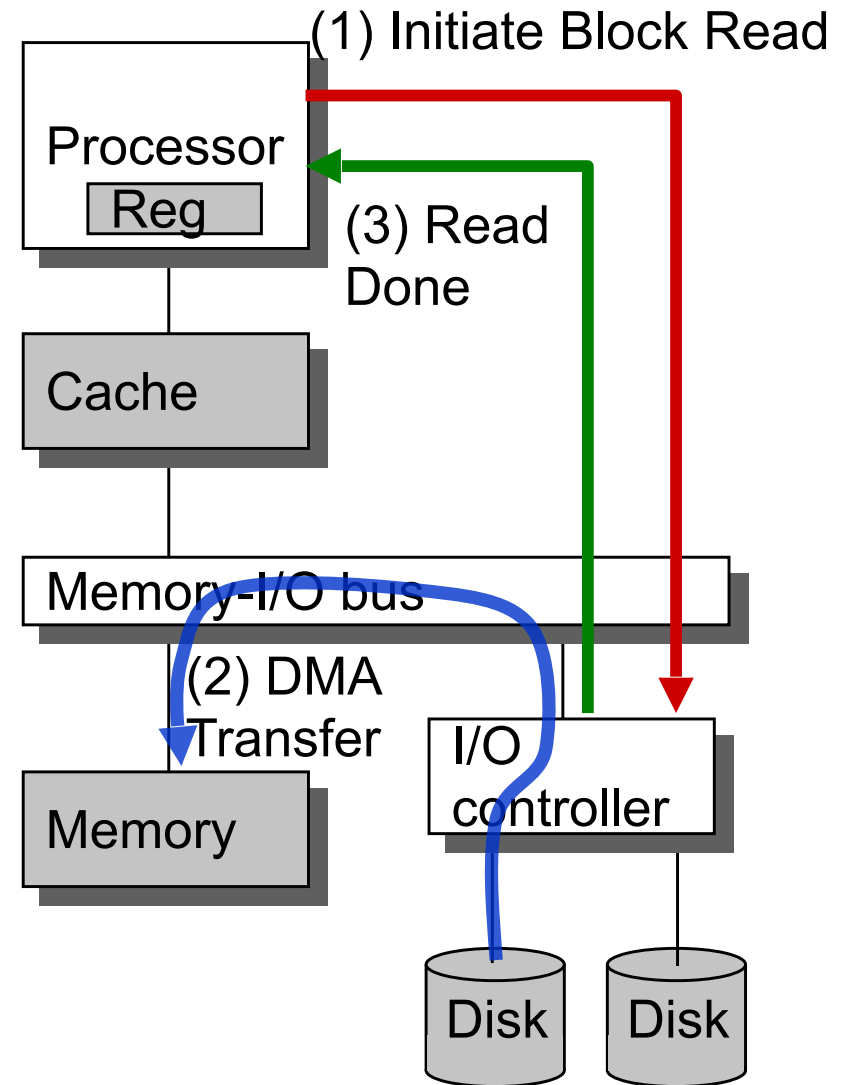


After fault



Servicing a Page Fault

- (1) Processor signals controller
 - ❑ Read block of length P starting at disk address X and store starting at memory address Y
- (2) Read occurs
 - ❑ Direct Memory Access (DMA)
 - ❑ Under control of I/O controller
- (3) Controller signals completion
 - ❑ Interrupt processor
 - ❑ OS resumes suspended process



Page Replacement Algorithms

- Se la memoria fisica è piena(i.e., list of free physical pages is empty), quale physical frame deve essere rimpiazzato on a page fault?
- Is True LRU feasible?
 - 4GB memory, 4KB pages, how many possibilities of ordering?
- Modern systems use approximations of LRU
 - E.g., reference bit

Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
 - ❑ *Reference bit* (aka *use bit*) in PTE set to 1 on access to page
 - ❑ Periodically cleared to 0 by OS
 - ❑ A page with reference bit = 0 has not been used recently
 - Disk writes take millions of cycles
 - ❑ Block at once, not individual locations
 - ❑ Write through is impractical
 - ❑ Use write-back
 - ❑ Dirty bit in PTE set when page is written
-