Chapter 7: Memory Management Techniques

Why do we need memory management?

- Improved resource utilization (only allocate resources based on demand)
- Independent resources for each process
- Make sure other processes don't access protected memory
- Using more resources than capacity physical memory

Simple Schemes for Memory Management

1. Fence Register to separate user and kernel

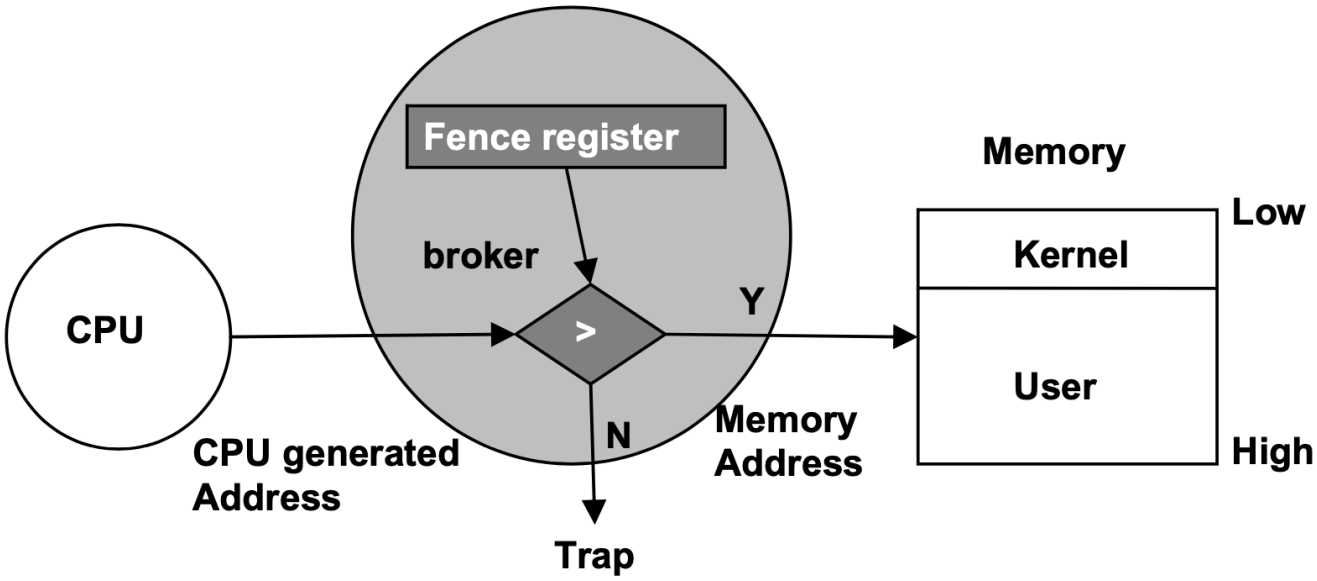


Figure 7.2: Fence Register

2. Bounds Registers

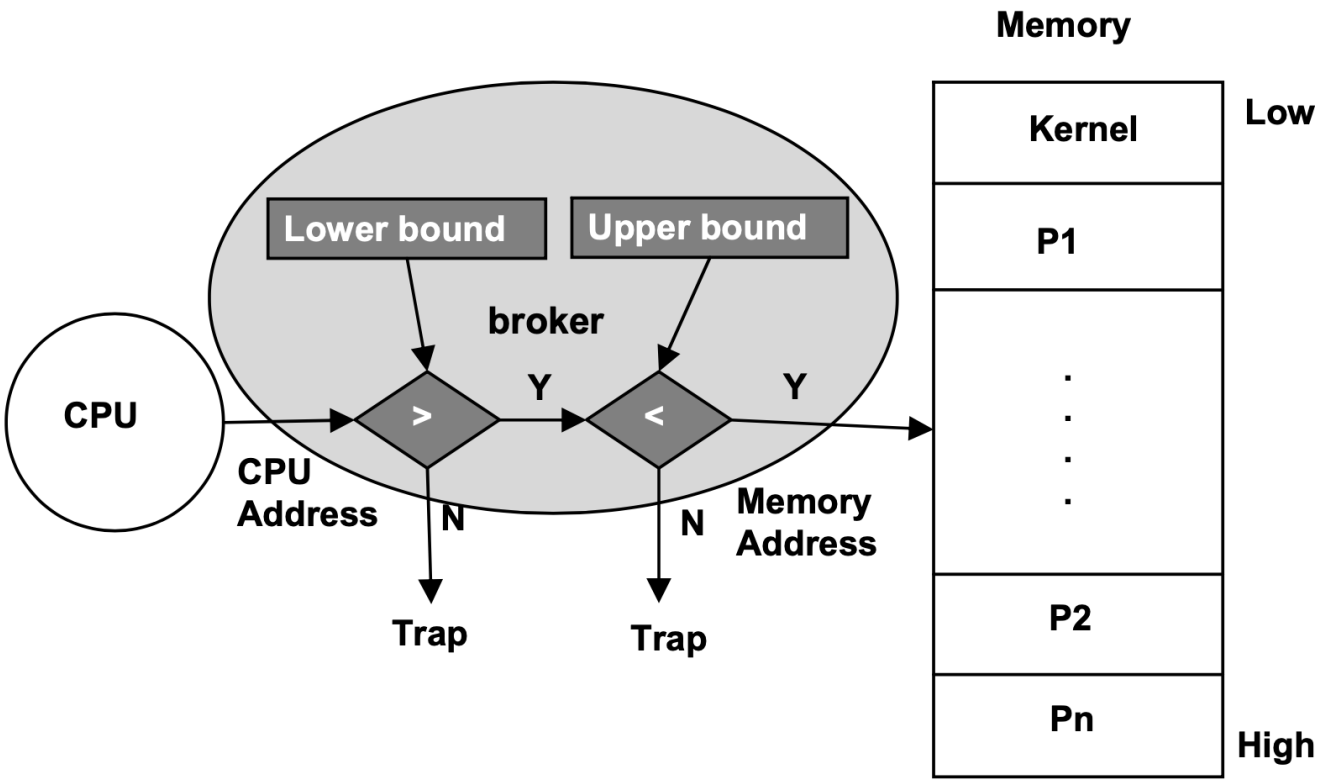


*Figure 7.3: Bounds Registers*Static Relocation

- Once an executable is created, memory addresses cannot be changed
- Bounds registers are part of the PCB

Dynamic Relocation

- Place an executable into any region of memory that can accommodate the memory needs of the process
- Memory address generated by a program can be changed during the execution of the program

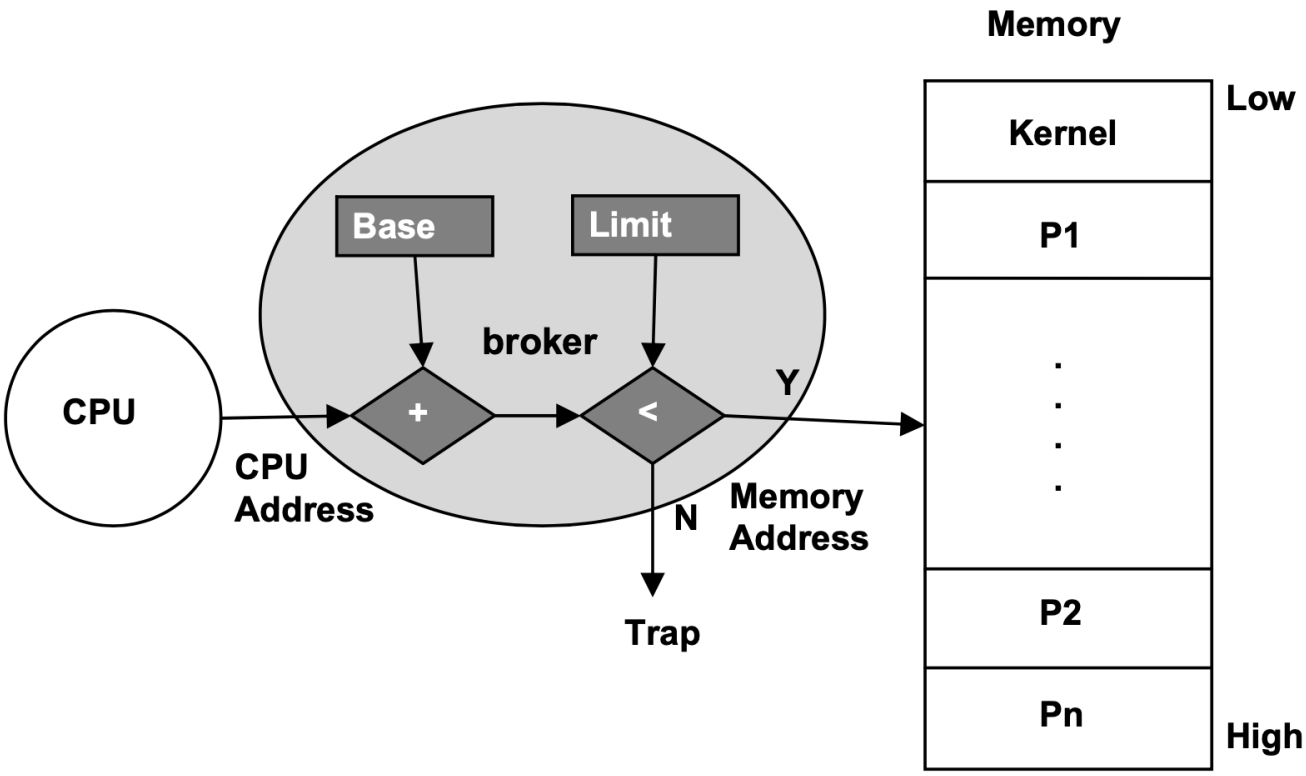


Figure 7.4: Base and Limit Registers

Fixed Size Partitions

- Leads to ***internal fragmentation***

Internal Fragmentation - wasted space inside each allocationAlso leads to another problem: ***external fragmentation***

- Memory chunks are non-consecutive and prevent allocations because of it

Variable Size Partitions

- Eliminates internal fragmentation
- Memory manager dynamically builds the table on the fly

Allocation table

Start address	Size	Process
0	2K	P1
2K	6K	P2
8K	3K	P3
11K	2K	FREE

Memory

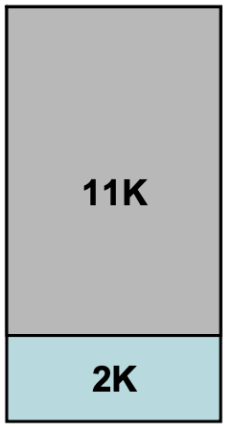


Figure 7.5: State of the Allocation Table After A Series of Memory Requests

Allocation table

Start address	Size	Process
0	2K	FREE
2K	6K	P2
8K	3K	P3
11K	2K	FREE

Memory

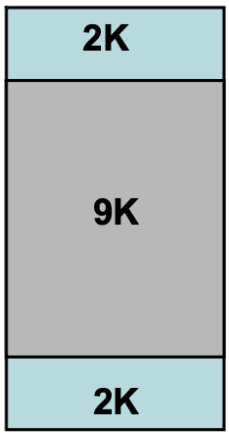


Figure 7.6: External Fragmentation After P1's Completion

1. Best Fit

   > The manager looks for the smallest space that can fit the allocation

- Better memory utilization
- Slower time tradeoff

2. First Fit

> The manager looks for the first space that can fit the allocation

- Time complexity is much better

Compaction

Allocation table

Start address	Size	Process
0	8K	FREE
8K	3K	P3
11K	2K	FREE

Memory

In the example above, the memory manager might choose to move P3 to start from address 0, creating a contiguous space of 10K

- Very expensive operation
- Virtually impossible in many architectures

Paged Virtual Memory

- Broker divides contiguous view into logical entities called pages
- Physical memory consists of *page frames* (a.k.a. *physical frames*)
    - Both logical frames and physical frames have the same fixed size, *pagesize*

Analogy

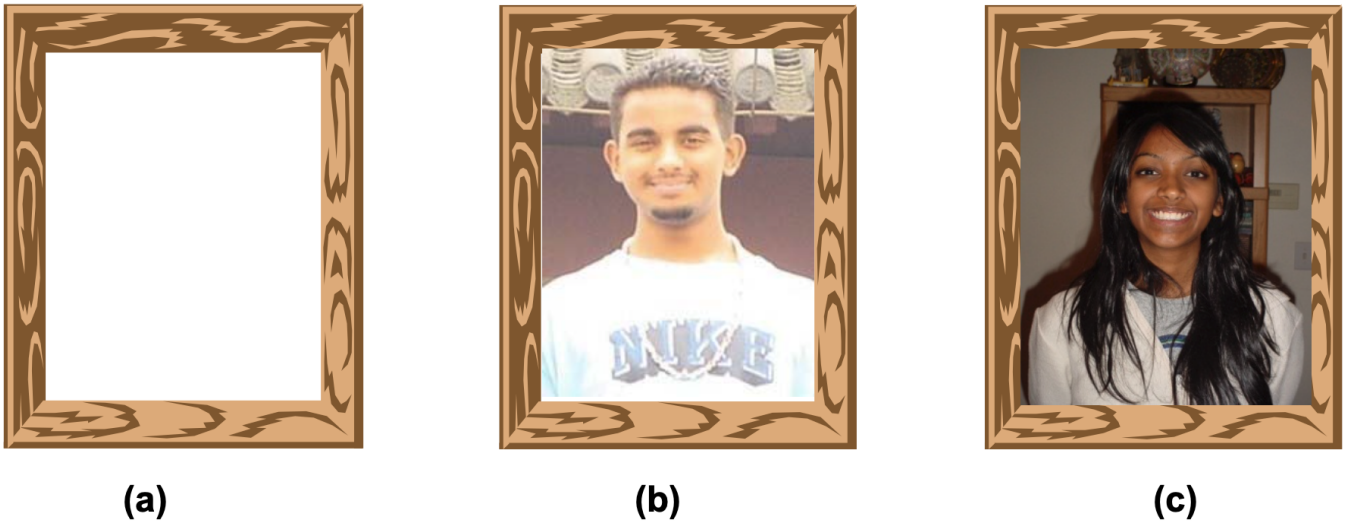


*Figure 7.7: Picture Frame Analogy*Important Notes

- Physical frame is reused for different students
- Does not need a unique for each student since he only sees one at a time

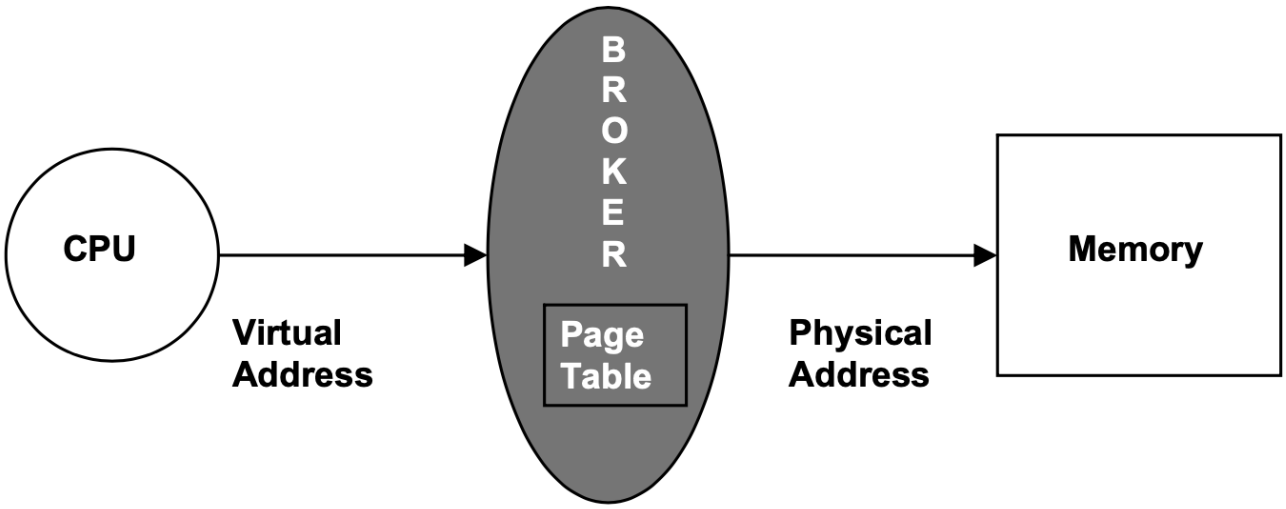


*Figure 7.8: Page Table*The user's view is *virtual memory* and the logical pages are *virtual pages*

- CPU generates the virtual addresses
- Page table translates virutal address to physical address

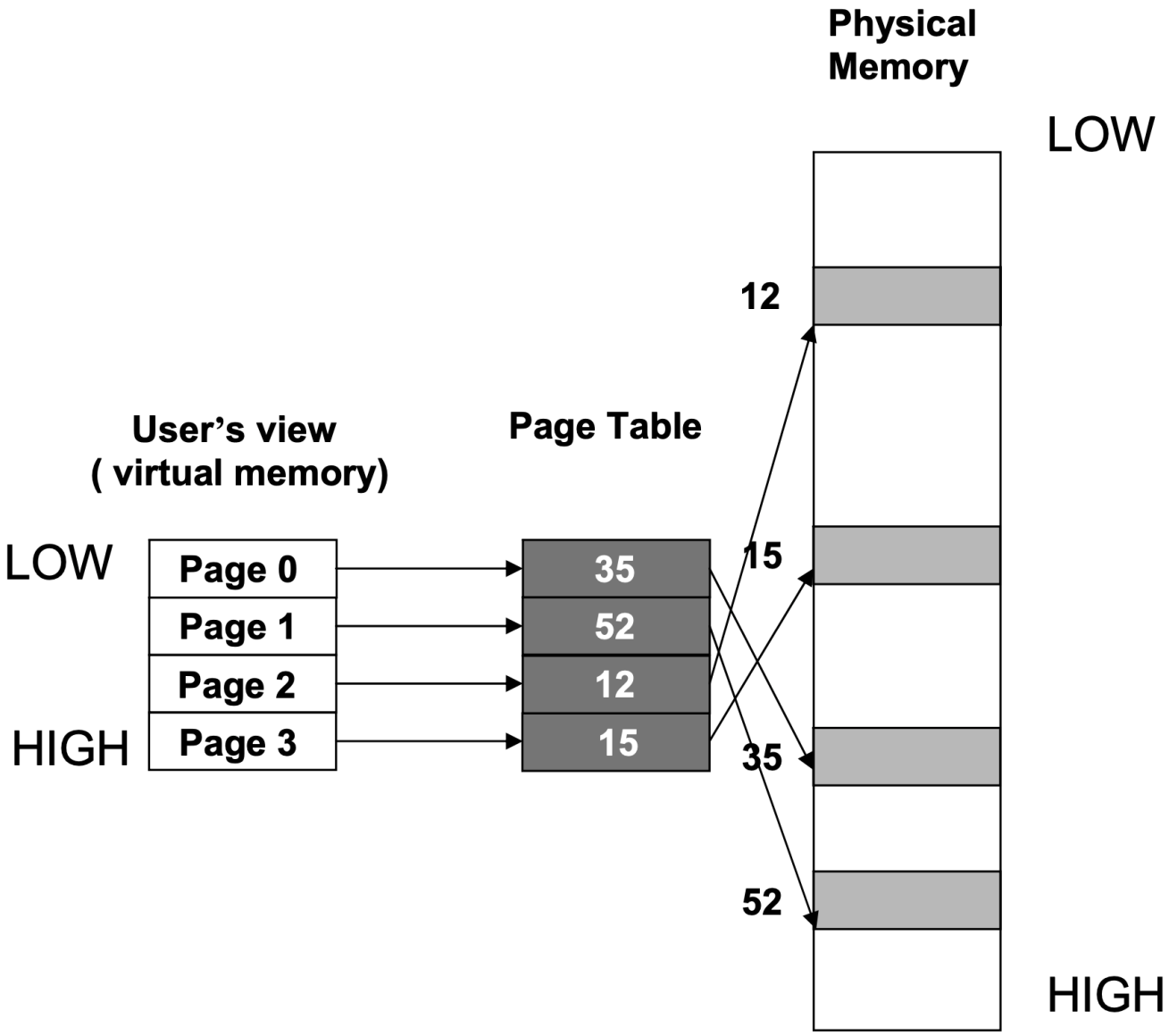
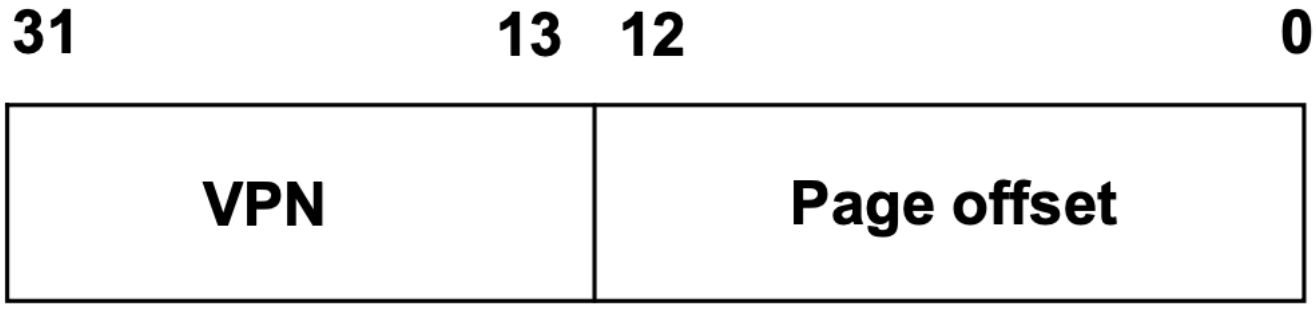


Figure 7.9: Decoupling User's Contiguous View of Memory

Page Table

If ***page size*** is N, then ***page offset*** is found from the lower $log_2 N$ bits of the virtual address



*32-bit virtual address with 8K Pages*In the above example, we need 2^{13} bits to address all 8K, and the remaining bits are used for the VPN**Converting virtual address to physical address**

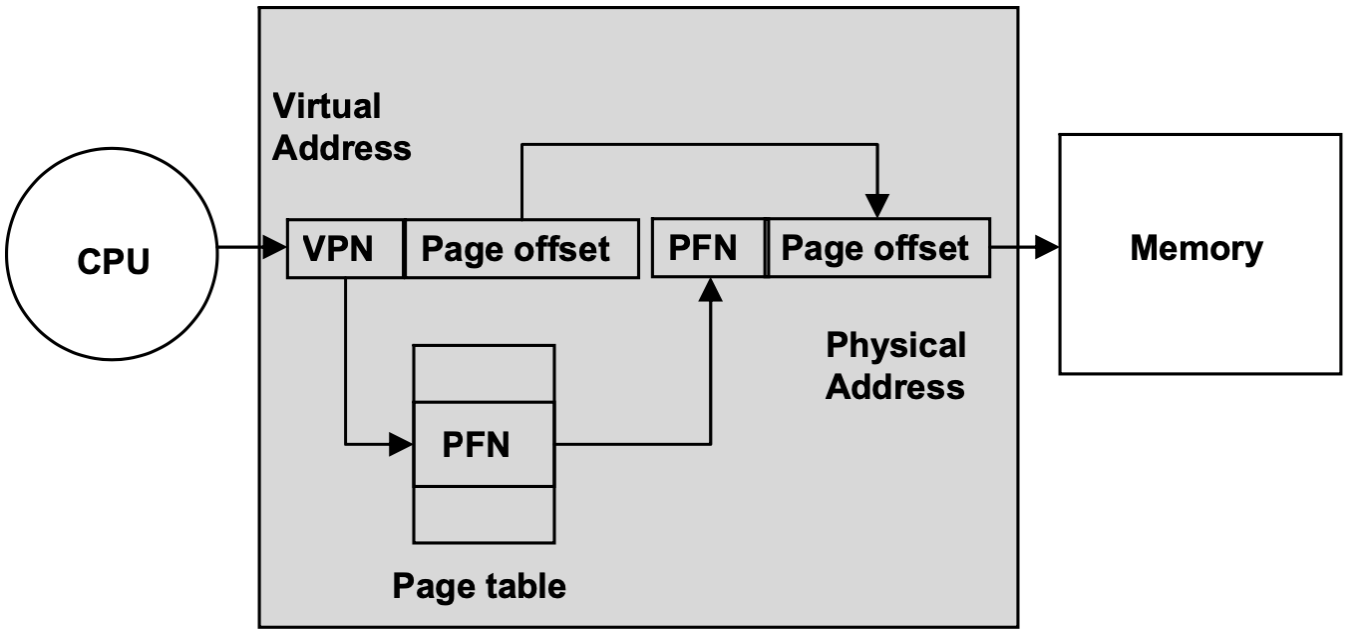


*Figure 7.10: Address Translation*Since hardware has to look this up every memory access, the page table resides in memory, ***one per process*** to provide memory protection

- Page table takes VPN and gives PFN
- Page offset is the same as page sizes will be the same
- Introduce the **Page Table Base Register (PTBR)** which contains the base address of the page table for the currently running process

Hardware for Paging

- Computes address of page table entry (*PTE*) that corresopnds to the VPN using the contents of the PTBR
- PFN fetched from this entry concatenated with page offset gives physical address

Page Table Set Up

```
typedef struct control_block_type {
    enum state_type state;
    address PC;
    int reg_file[NUMREGS];
```

```c
    struct control_block *next_pcb;
    int priority;
    address PTBR;
}
```

Segmented Virtual Memory

- unique segment number
- size of segment

Figure 7.11: Segmented Address

Segmentation Hardware Support

Each entry in the segment table is called a *segment descriptor*

- Gives start address for a segment and the size
- Each process has its own segment table
- Requires a *Segment Table Base Register*

Paging vs Segmentation

Attribute	Paging	Segmentation
Address spaces per process	One	Several
Visibility to user	User is unaware of paging, looks linear	User aware of multiple address spaces
Software Engineering	No benefit	Enables modular design, increases maintainability
Size of page/segment	Fixed by architecture	Variable chosen by the user for each individual segment

Attribute	Paging	Segmentation
Internal Fragmentation	Possible	None
External Fragmentation	None	Possible

Paged Segmentation

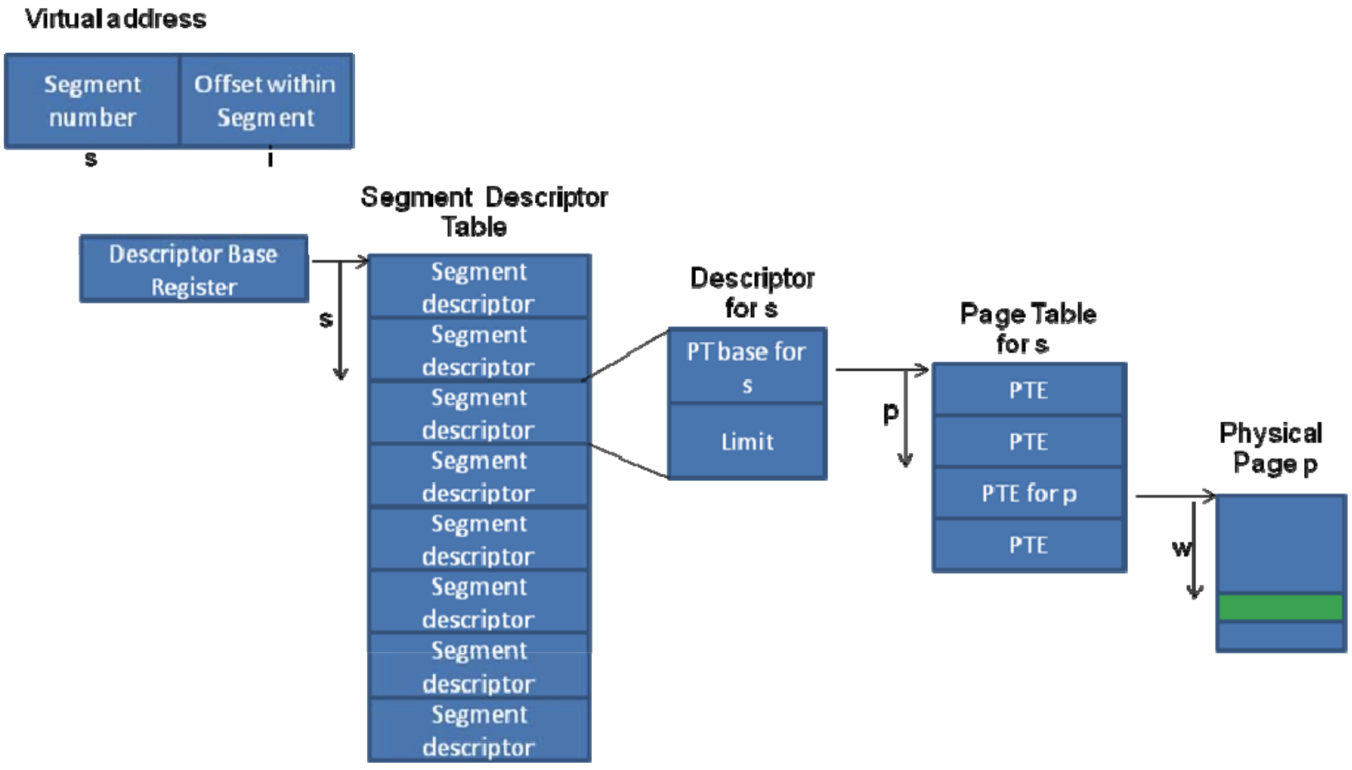


Figure 7.12: Paged Segmentation