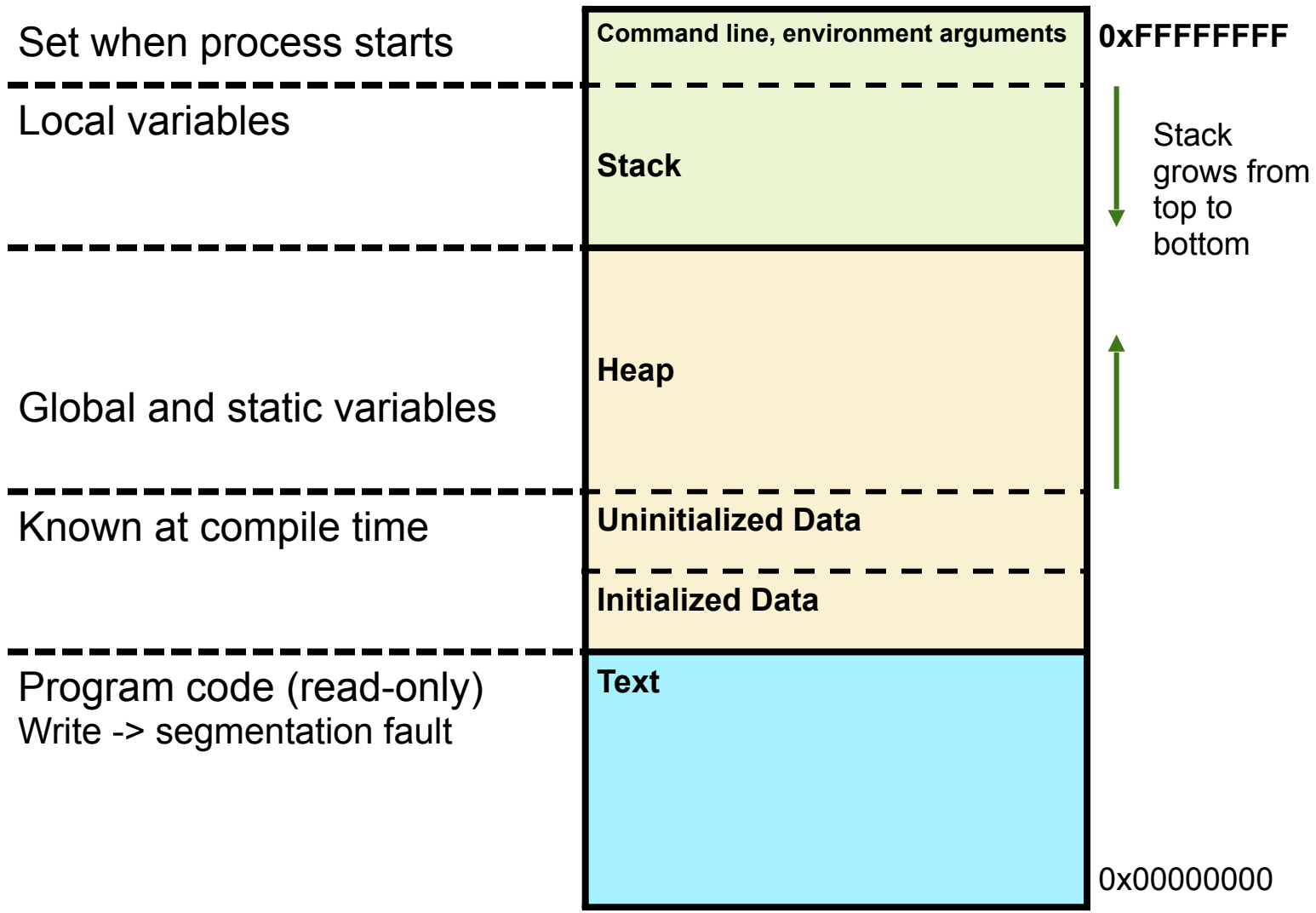
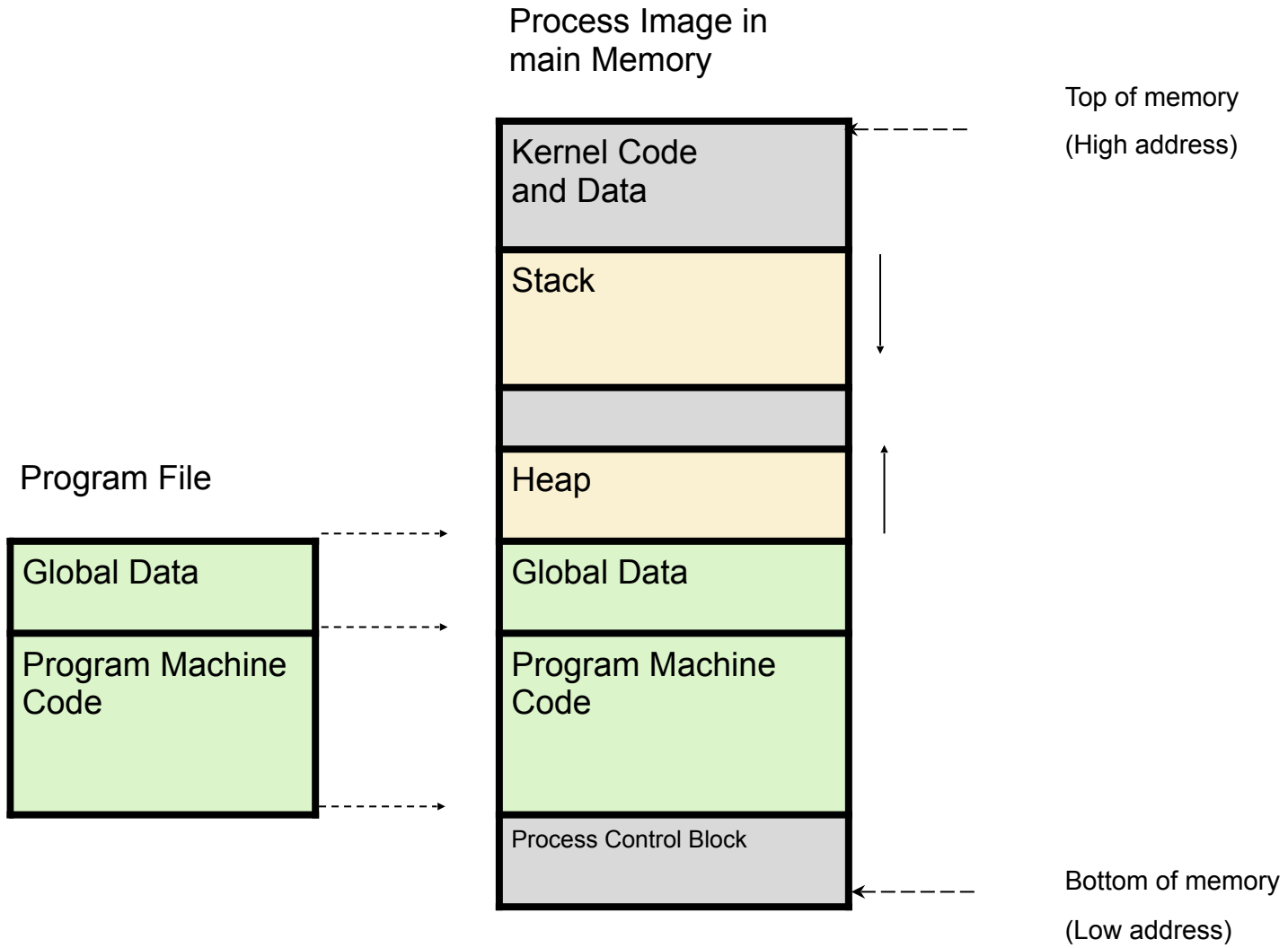
Memory Management - II

Segmentation and Paging

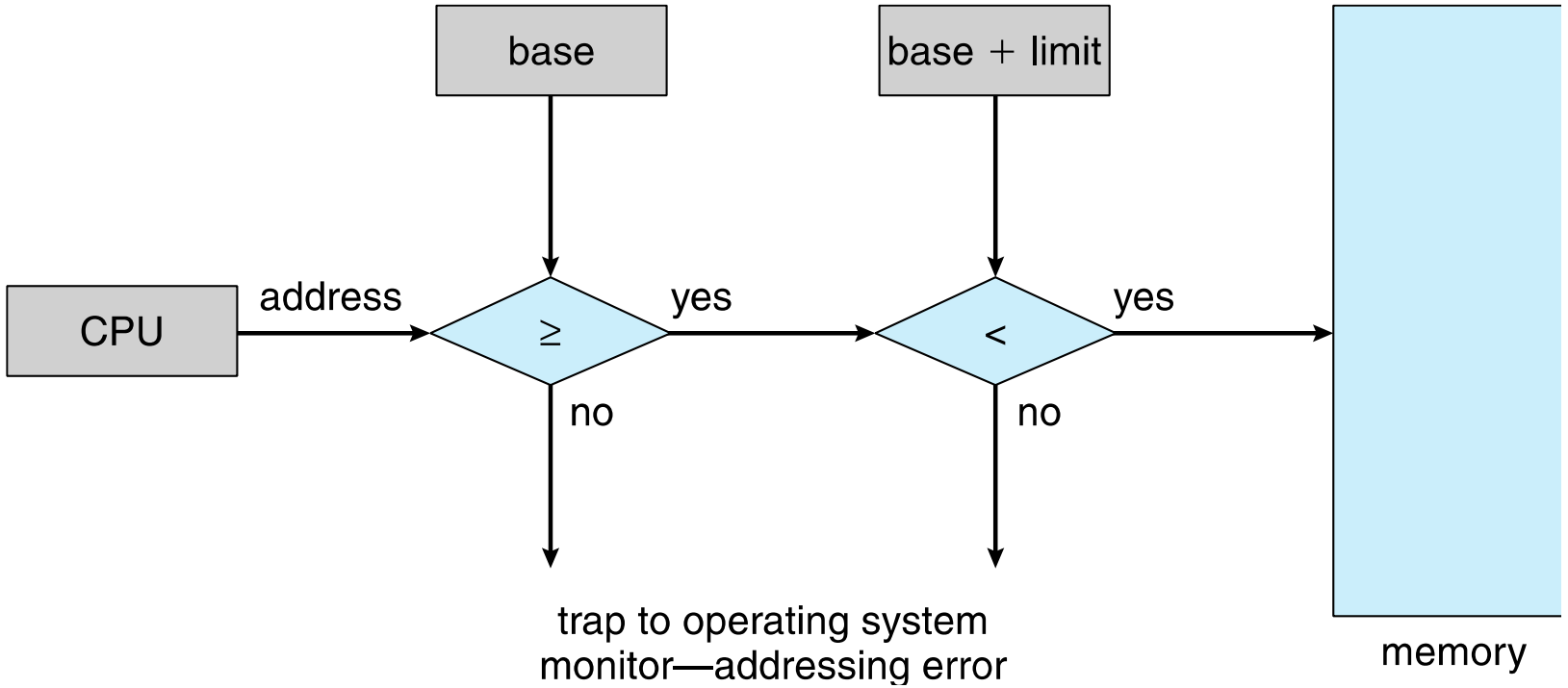
Process Virtual Address Space



Address Space: Processes



Memory Protection



Process Address Space

- Virtual address space:
  - Starting address is 0000H.
  - Includes space for heap and stack
- Physical address space: RAM region actually allocated
- Memory Management Unit
  - Translates virtual address to physical address
  - Checks that memory violation does not occur
- Memory management
  - Paging
  - Segmentation

Memory Virtualization

- Space Sharing
- Hardware Support
  - Bound registers – one pair (base and limit) per CPU core
  - Segments and CPU registers (x86)
    - **Code Segment**
    - **Stack Segment**
    - **Data Segment**
    - **Extra Data Segment**
    - **Context switch: registers saved (outgoing) and restored (incoming)**

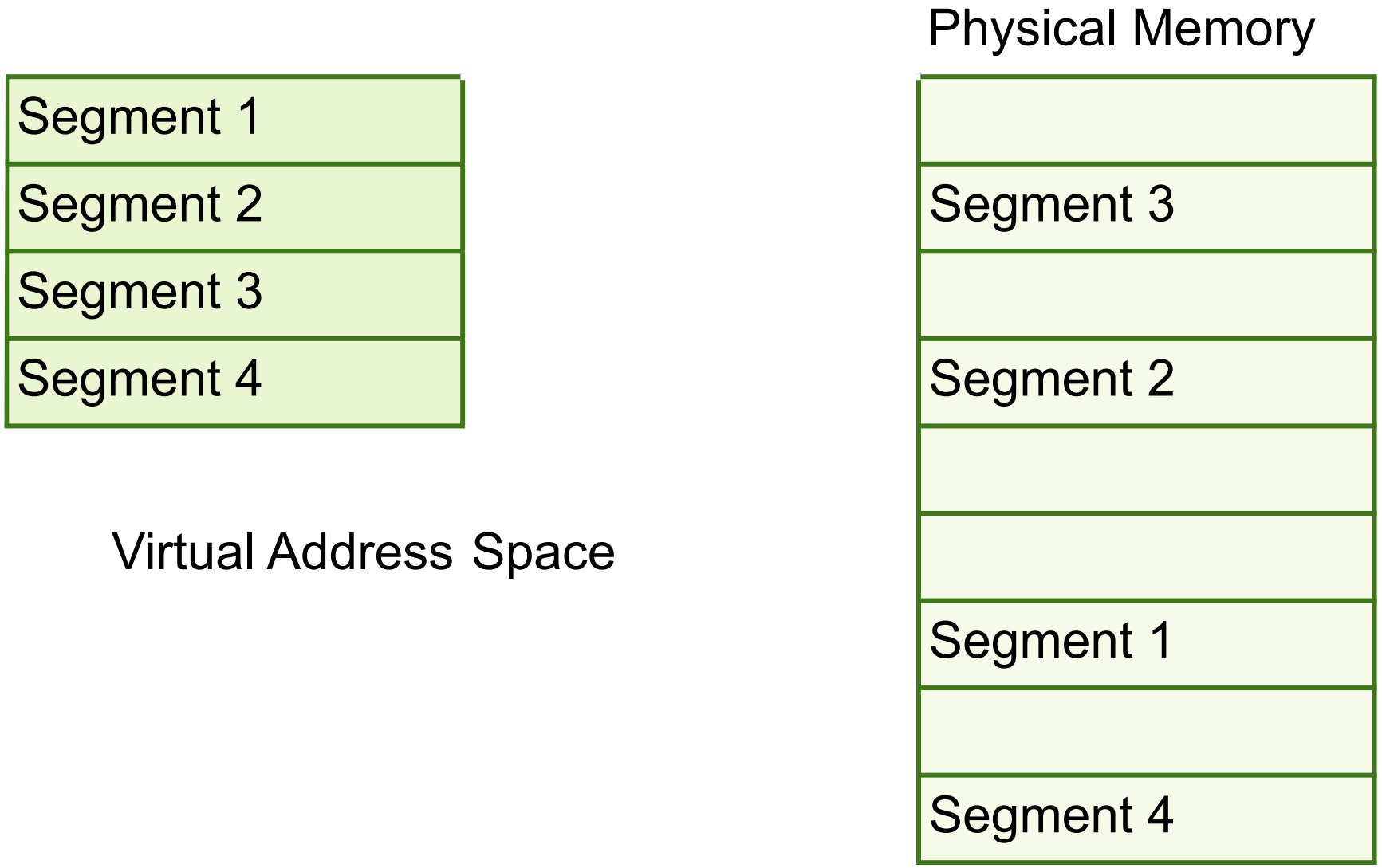
Segmentation

- Three logical segments
  - Code, stack and heap
  - Two bits needed to represent each segment type
- Virtual address: $< Segment\ type,\ Offset >$
- Segment register: 16 bits
  - 2 bits reserved for type [00: code, 01: stack, 10: heap]
  - 14 bits for offset
- Segment size = 2^{14} = 16 KB
- Code of program limited to 16 KB
- Size of program limited to 48 KB

Segmentation

- One base, limit register pair per logical segment
  - code, stack, heap
- Allocation:
  - Contiguous memory allocated to each segment
  - Segments can be allocated independently
- Address represented as base and offset
  - Base address of segment stored in register
  - Offset: number of bits (say m)
  - Offset determined by Segment Size

Segmentation : Memory Allocation



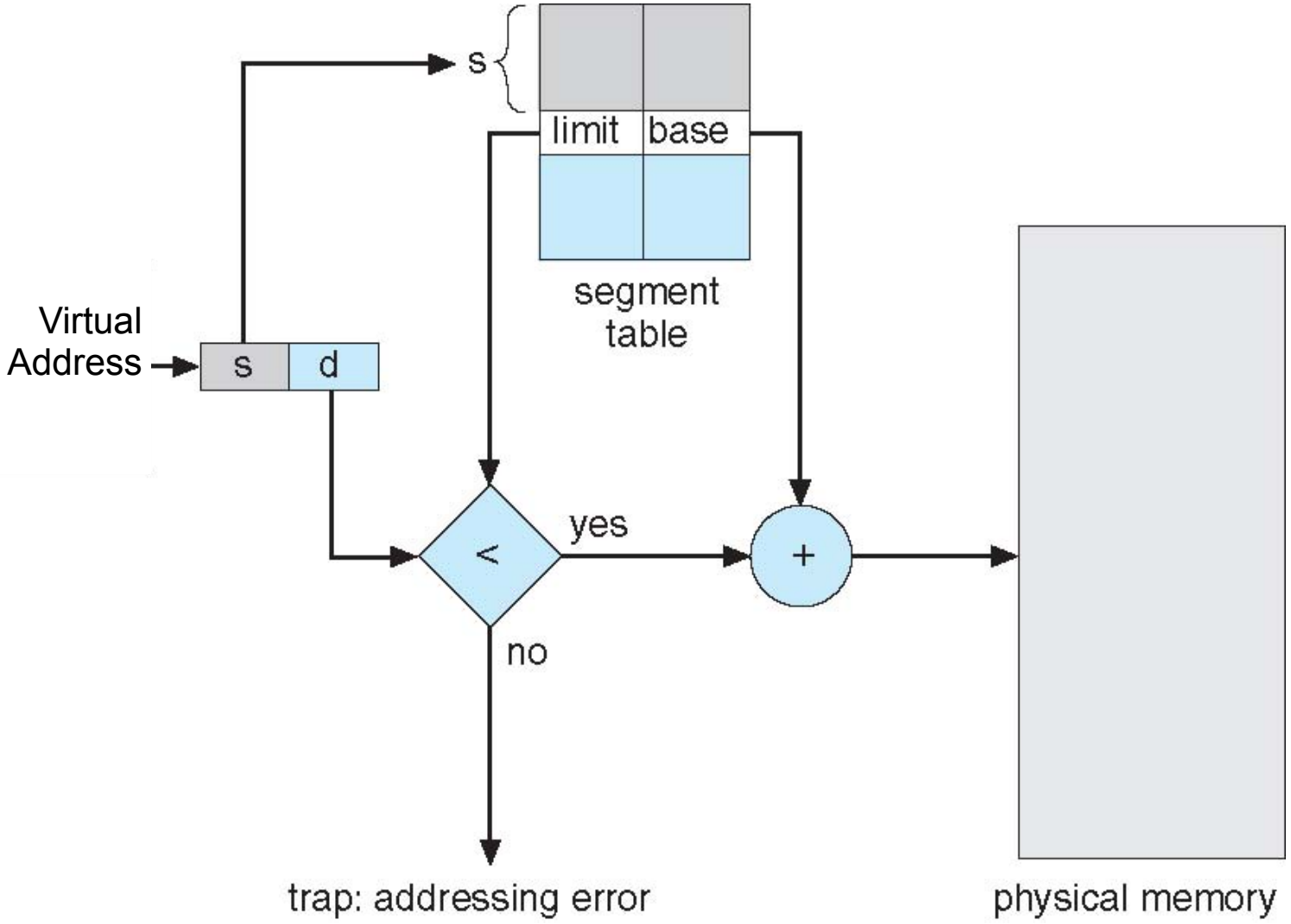
Segmentation: Hardware Support

- Logical/Virtual address consists <segment-number, offset>,
- Segment table – maps virtual address to physical address
  - base – starting physical address of segment
  - limit –length of the segment
- Segment-table base register (STBR): location of segment table
- Segment-table length register (STLR): number of segments
- For valid address
  - Segment number < STLR
  - Offset < limit

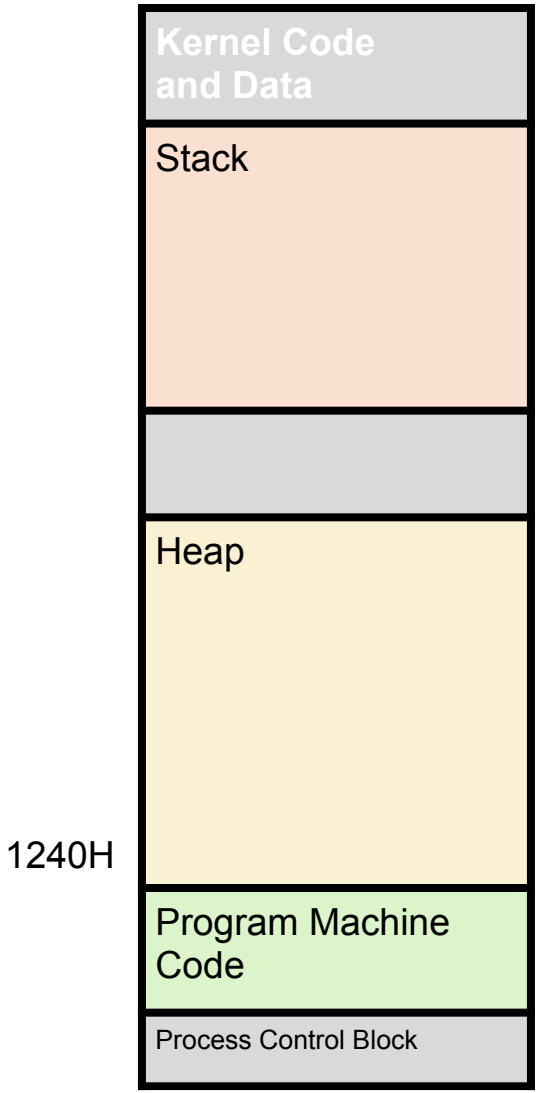
Segmentation: Address Translation

- For valid address
  - Segment number < STLR
  - Offset < limit
- Address translation:
  - Use segment number as index into Segment Table
  - Base address = ST[segment number]
  - If (offset < limit)
    - **Physical address = Base Address + offset**
  - Else
    - **Segmentation Fault**
  - Endif

Segmentation: Address Translation

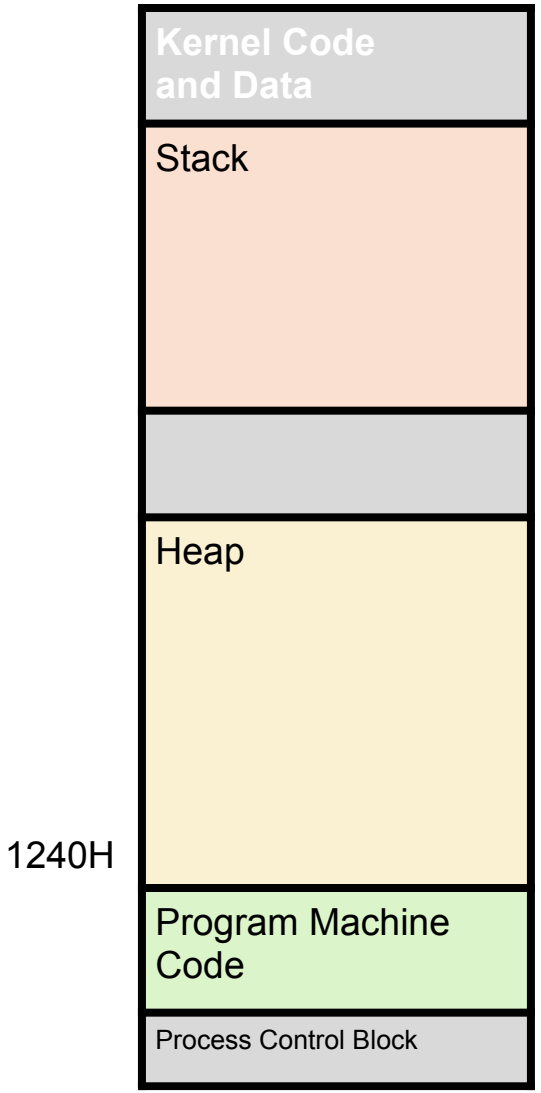


Segmentation Address Translation: Example



- Virtual address: 16 bits
- Segment size: 1 KB = 0400 H
- Base address: 4000 H
- Limit register: 43FF H
- Virtual address: 1356 H
- Offset: (1356 – 1240) H = 0116 H
- Physical address: (4000+0116) = 4116 H
- (4116 > 4000) Yes
- (4116 < 43FF) Yes
- Access allowed: Yes

Segmentation Address Translation: Example

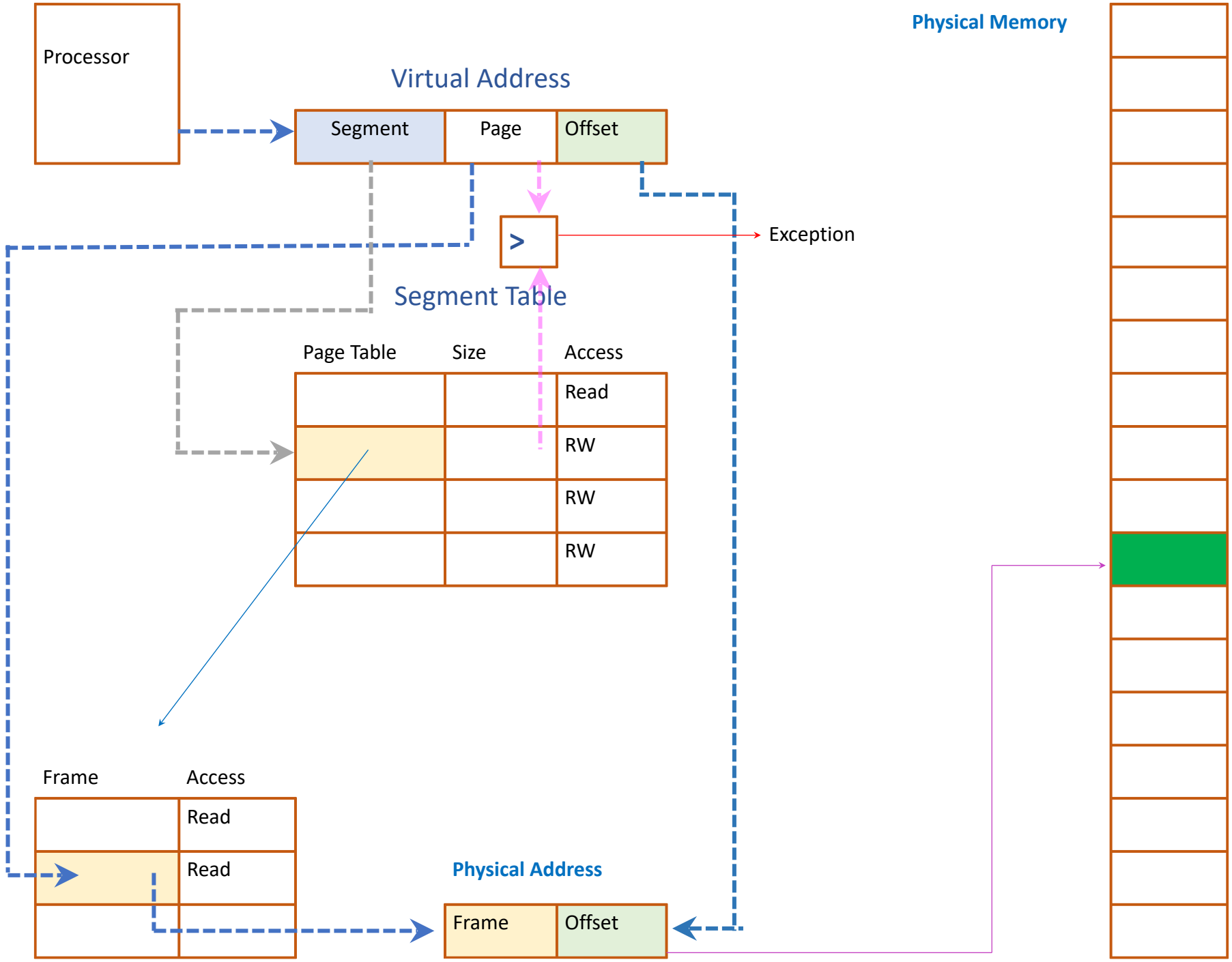


- Virtual address: 16 bits
- Segment size: 1 KB = 0400 H
- Base address: 4000 H
- Limit register: 43FF H
- Virtual address: 1736 H
- Offset: (1736 – 1240) H = 04F6 H
- Physical address: (4000+0456) = 4456 H
- (4456 > 4000) : Yes
- (4456 < 43FF) : No
- Access allowed : No

Challenges with Segmentation

- Logical entities: Variable size
- Segment size needs to be fixed; limited by size of base and limit registers
- Otherwise, segmentation needs to be paged
  - Increases complexity
  - Increases access time

Paged Segmentation

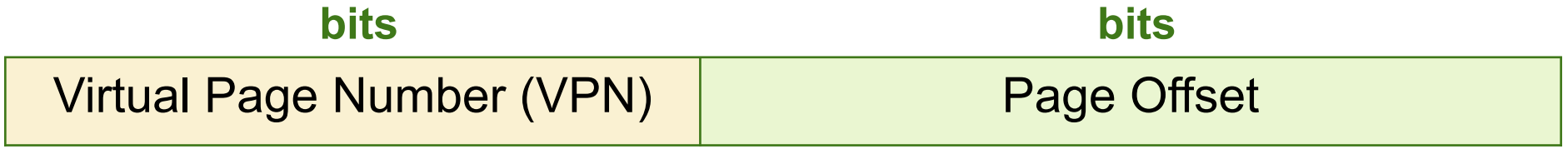


Paging

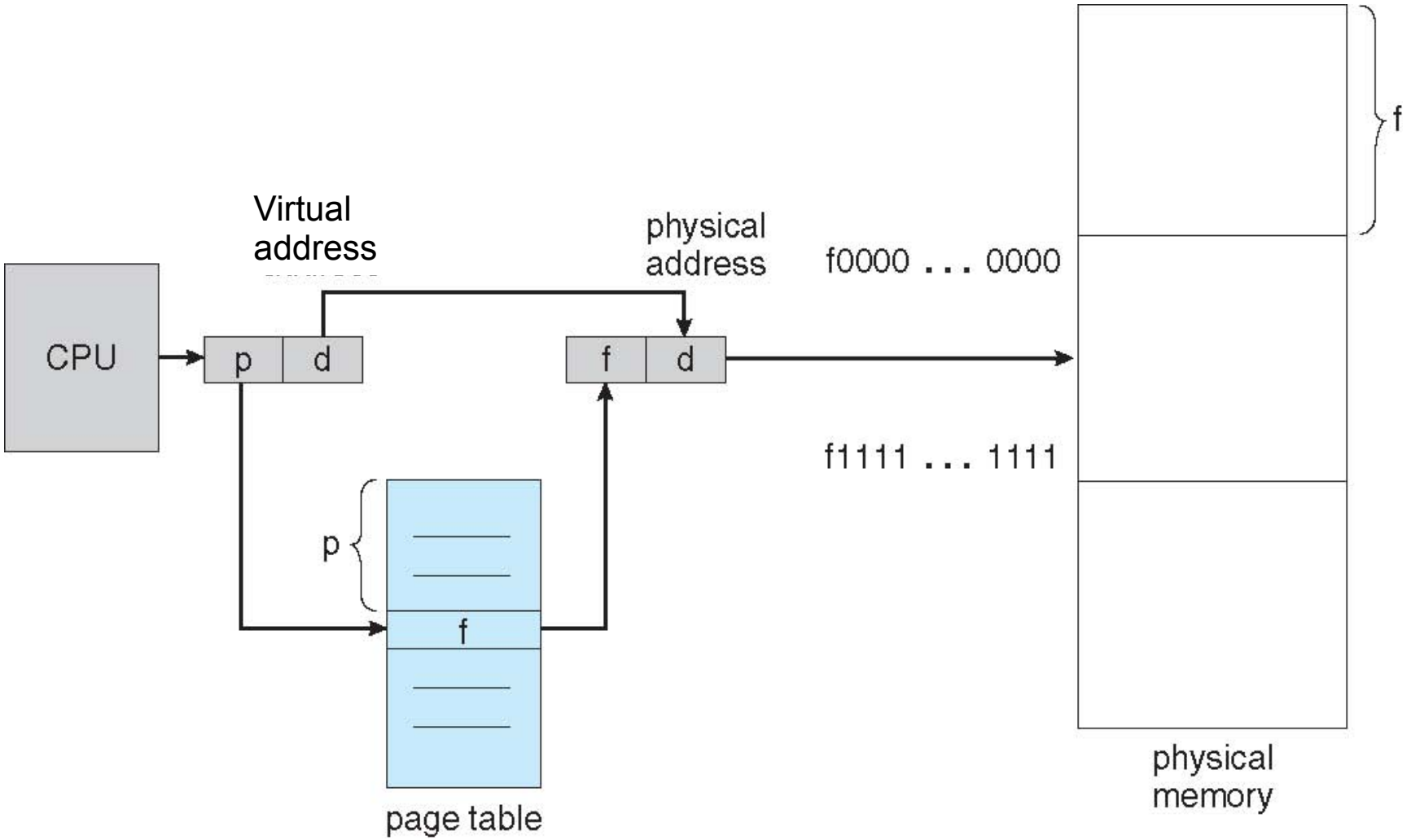
- Virtual address space divided into equal sized pages (non-overlapping but contiguous)
- Physical memory also divided into same sized blocks (called frames)
- Page size = 2^m bytes
- OS loads pages to physical frames
  - creates page table per process to store mapping
  - PTBR (page table base register): address of page table
  - Page table entries (PTE) changed as pages are loaded to/removed from main memory
  - At a given time, only a subset of process pages may be loaded
- Internal fragmentation (average) = half a page size

Paging: Address Translation

- Virtual address is a 2-tuple $< vp,\ d >$
  - Virtual page number (vp) – index into page table that contains base address of each page in physical memory
  - Page offset (d) – added to base address of the page to obtain the physical memory address
- Virtual address of m-bits (Virtual space = 2^m bytes)
- Page size = 2^n bytes (n-bits)
- Physical address is a 2-tuple $< p,\ d >$
- Address translation: replacing vp by $p\ =$ PT[vp]



Paging: Address Translation



Linux: Process Memory Map

- How a process is mapped to memory

  cat /proc/pid/maps

- List of all the memory mappings for the process

  cat /proc/pid/pagemap

- Details: https://kernel.org/doc/html//latest/admin-guide/mm/pagemap.html

Windows: Process Memory Map

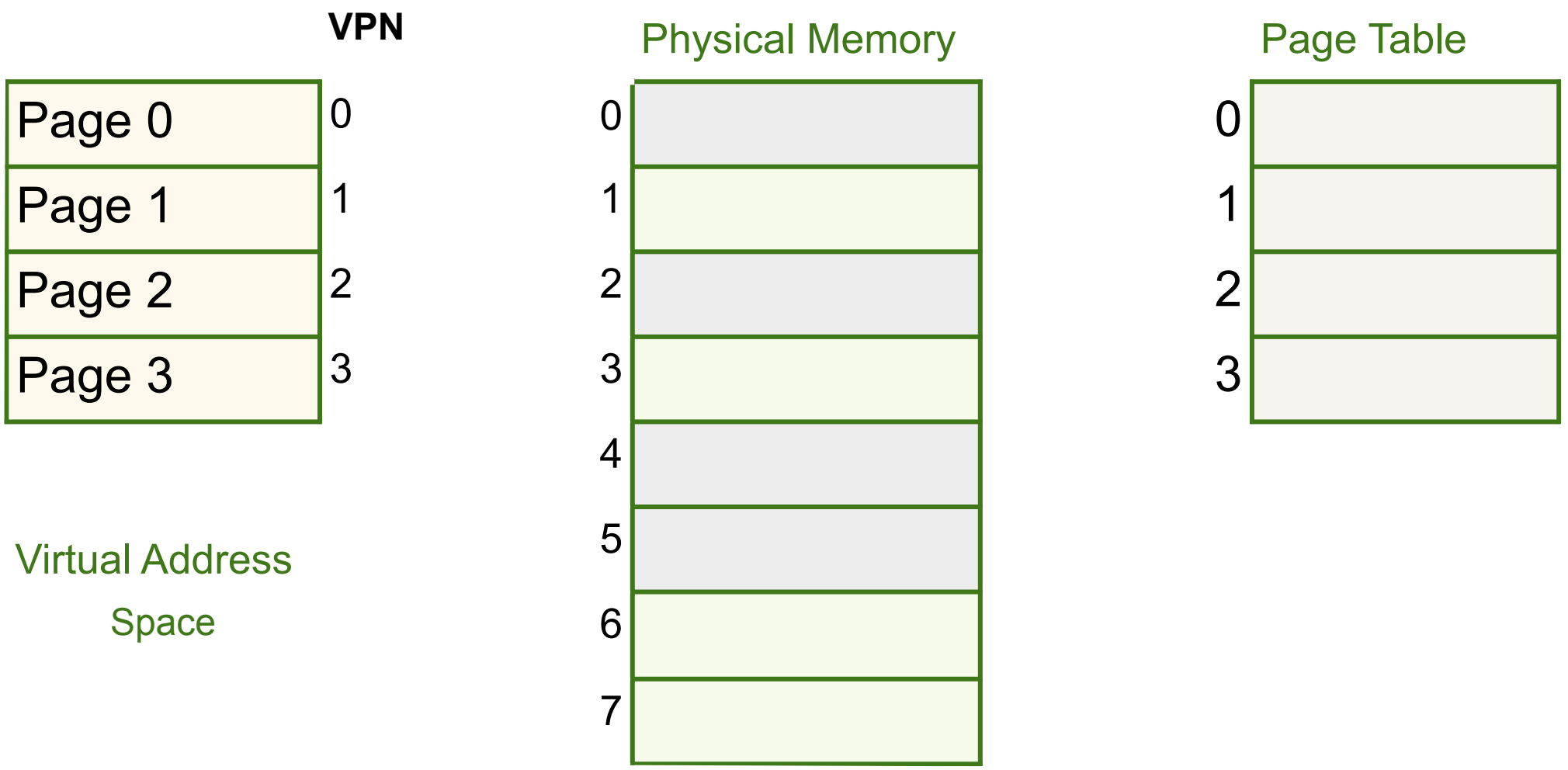
- List of all the memory mappings for the process with the specified PID.

  tasklist /v /pid <pid>

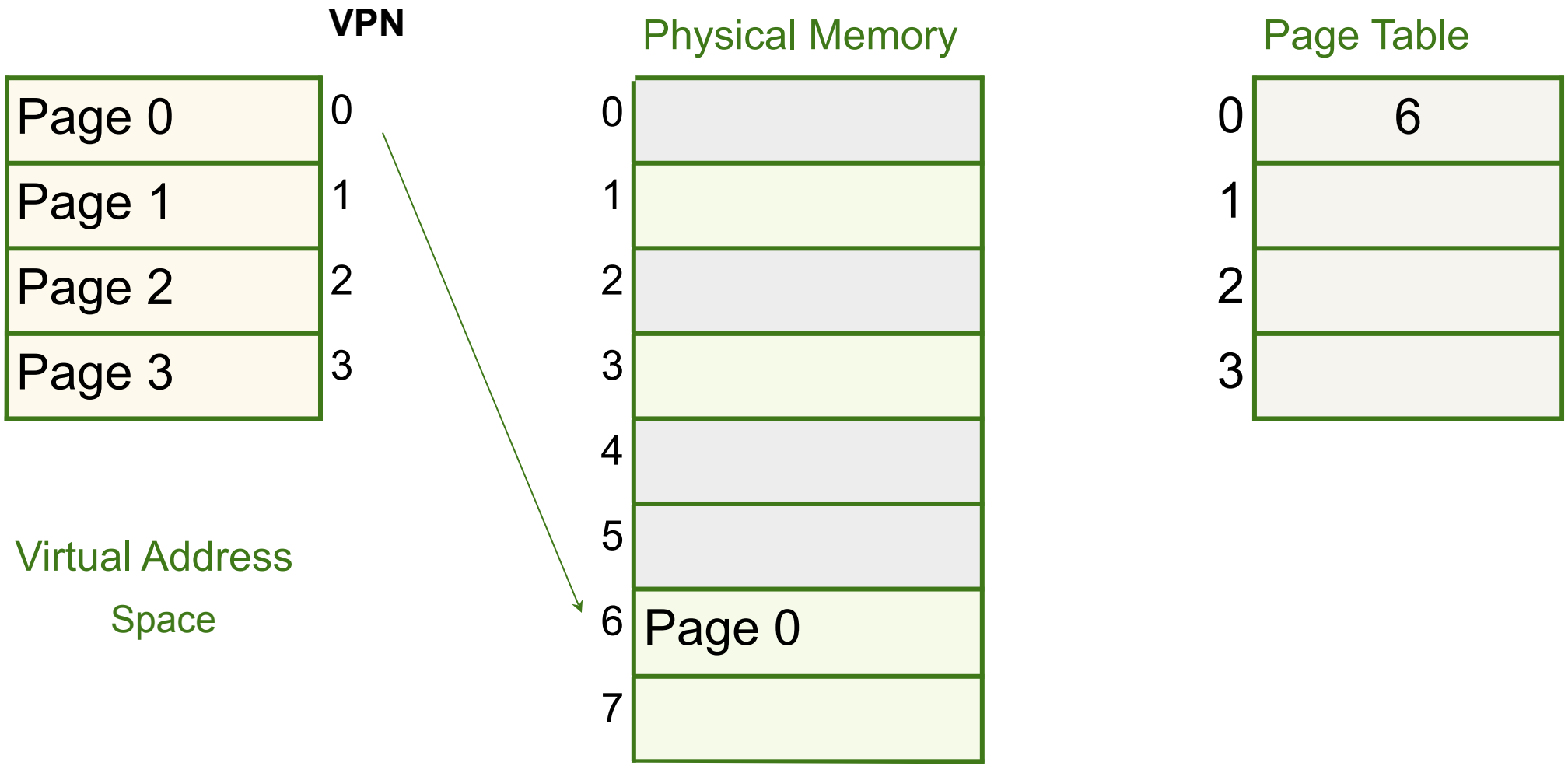
- This command will open a debugger session for the process with the specified PID. You can then use the dt nt!_MM_PAGE_TABLE command to dump the page table for the process.

  windbg -i <pid>

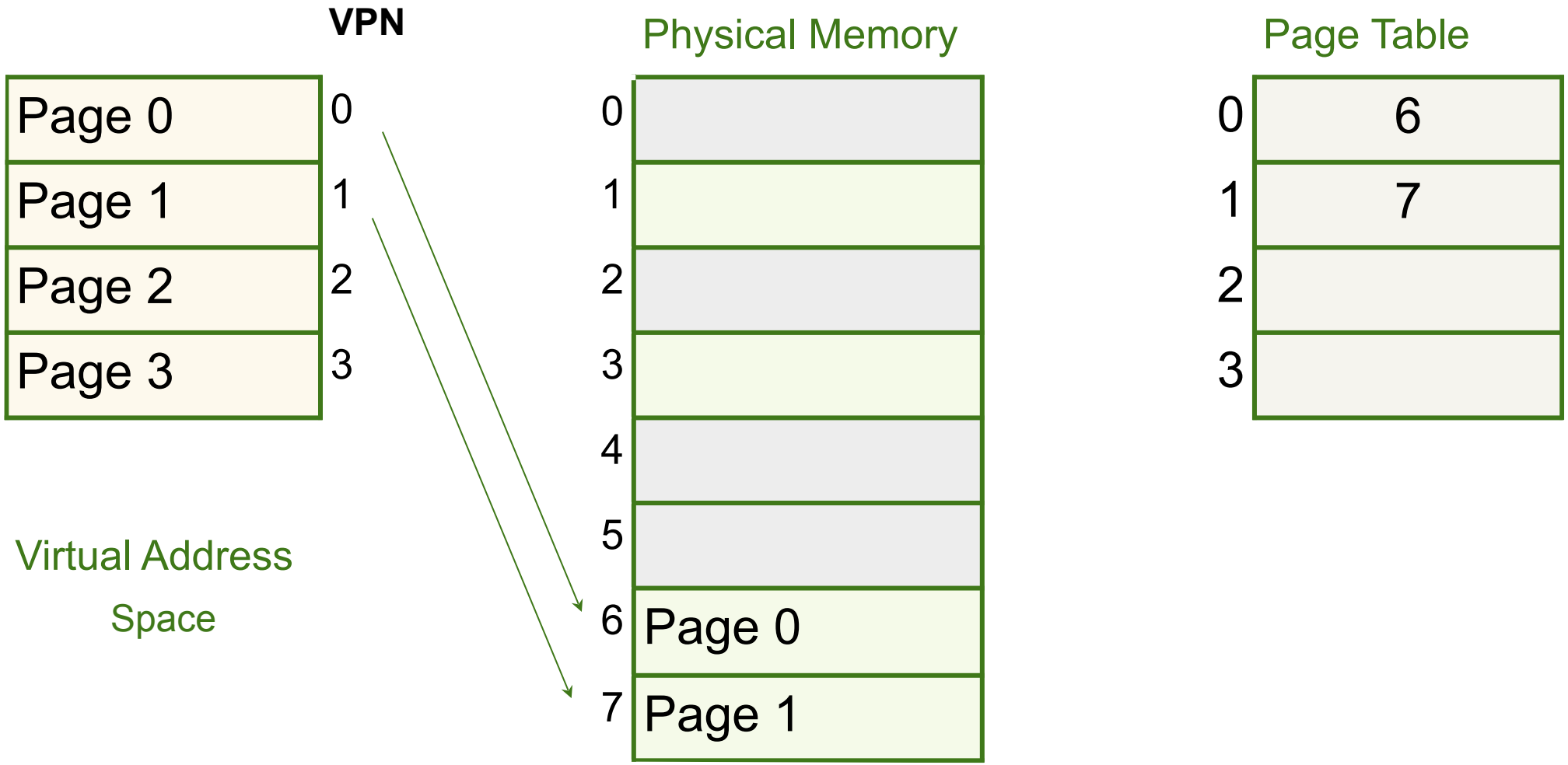
Paging : Memory Allocation



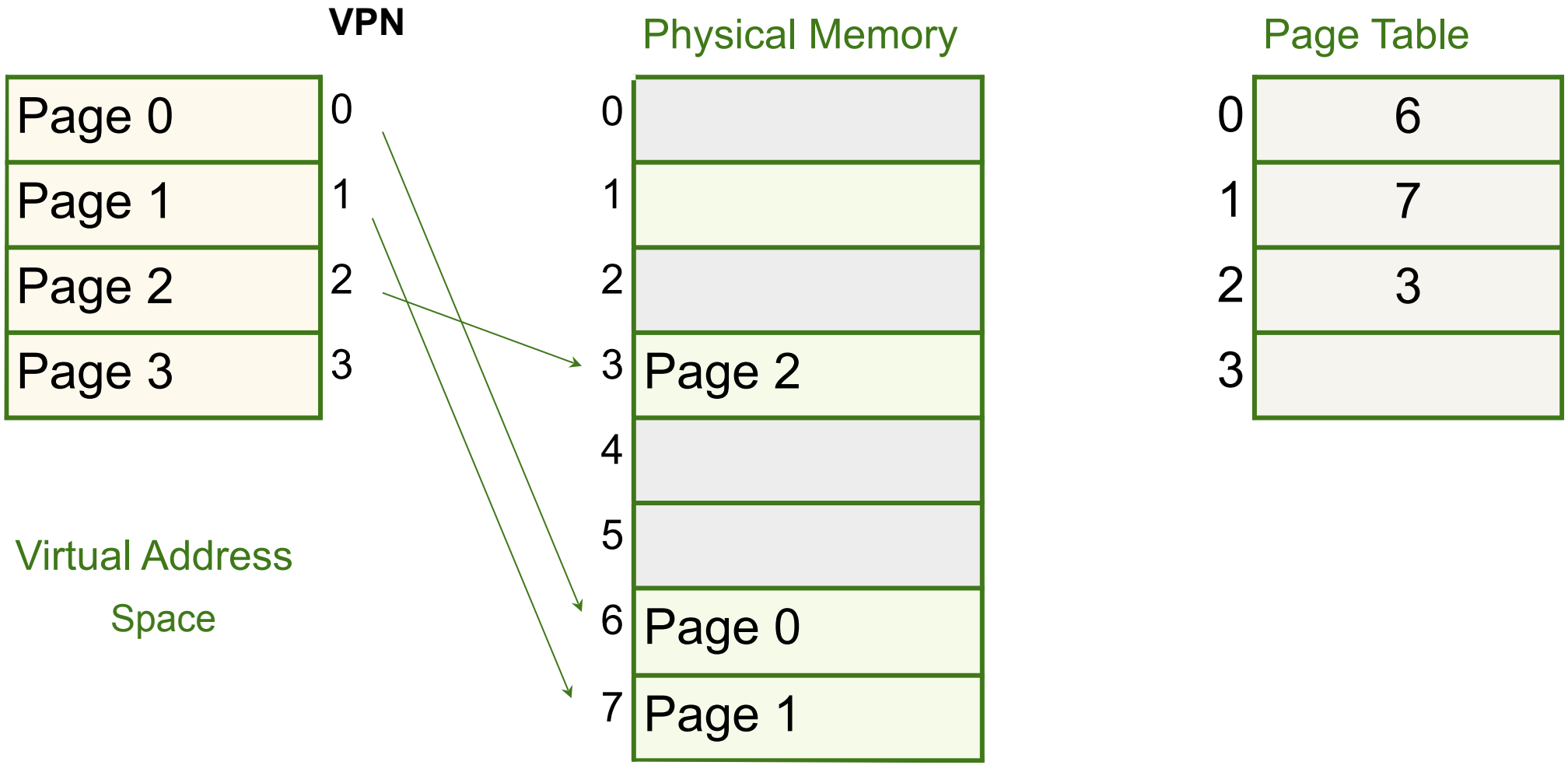
Paging : Memory Allocation



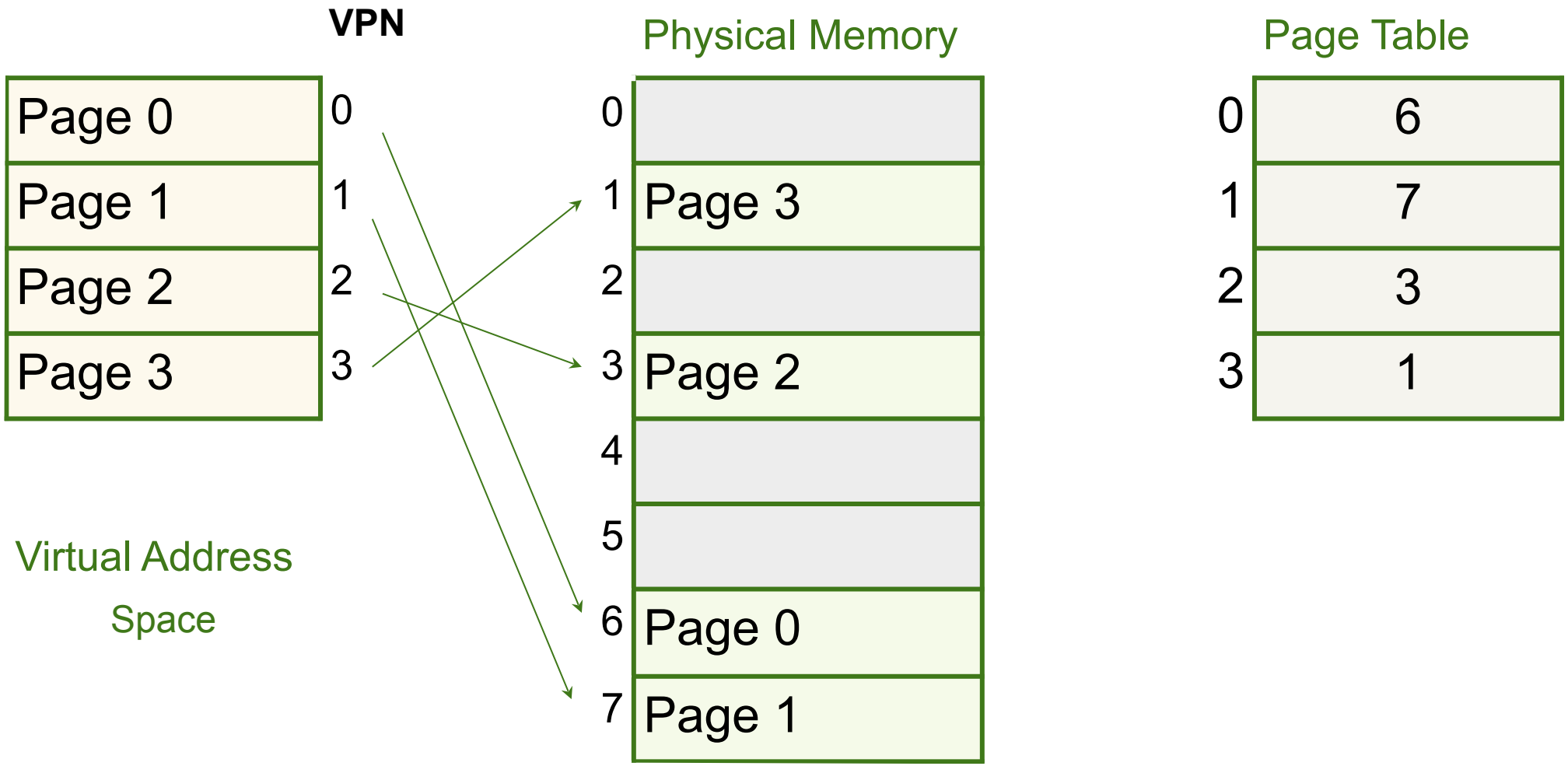
Paging : Memory Allocation



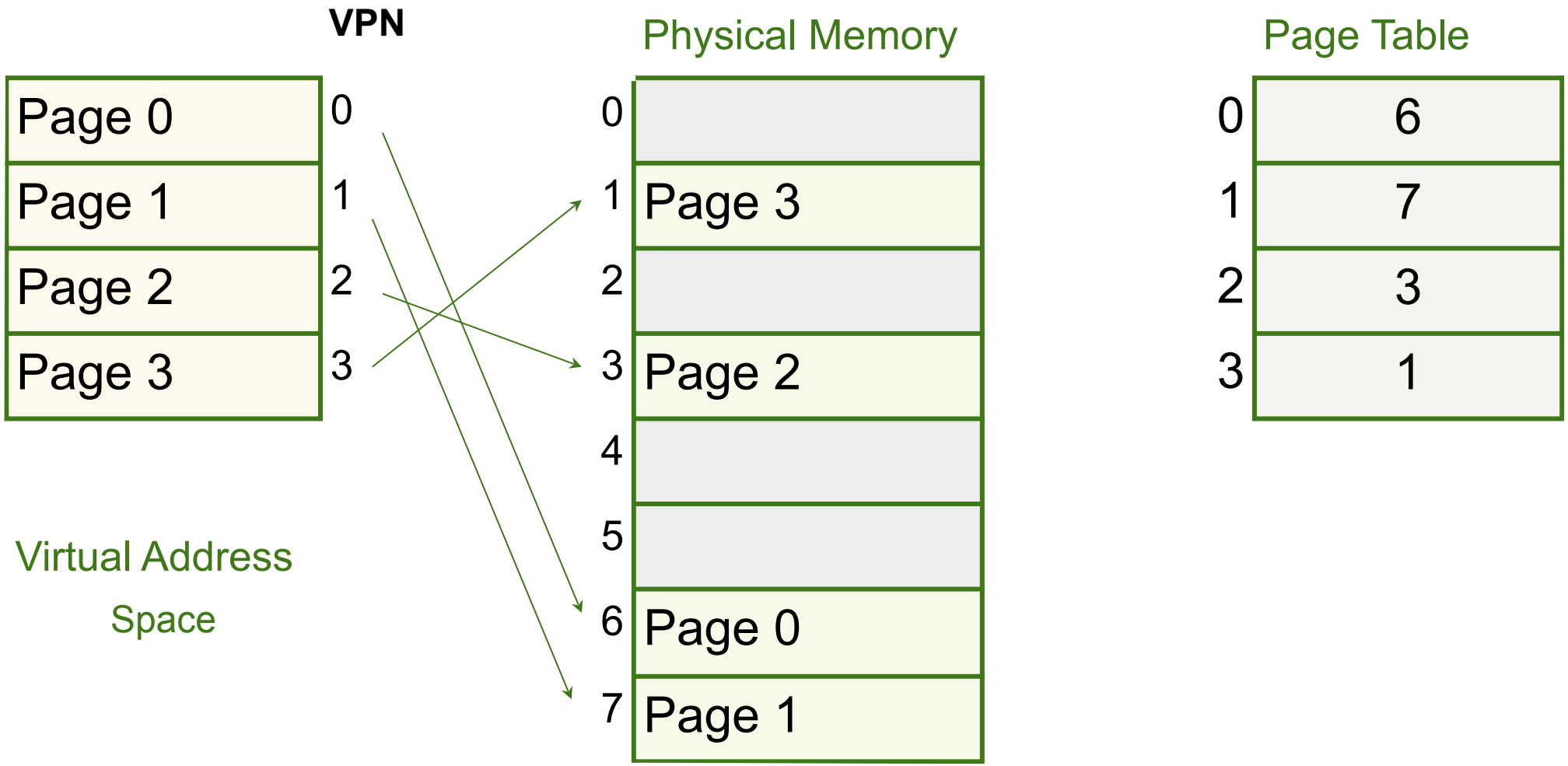
Paging : Memory Allocation

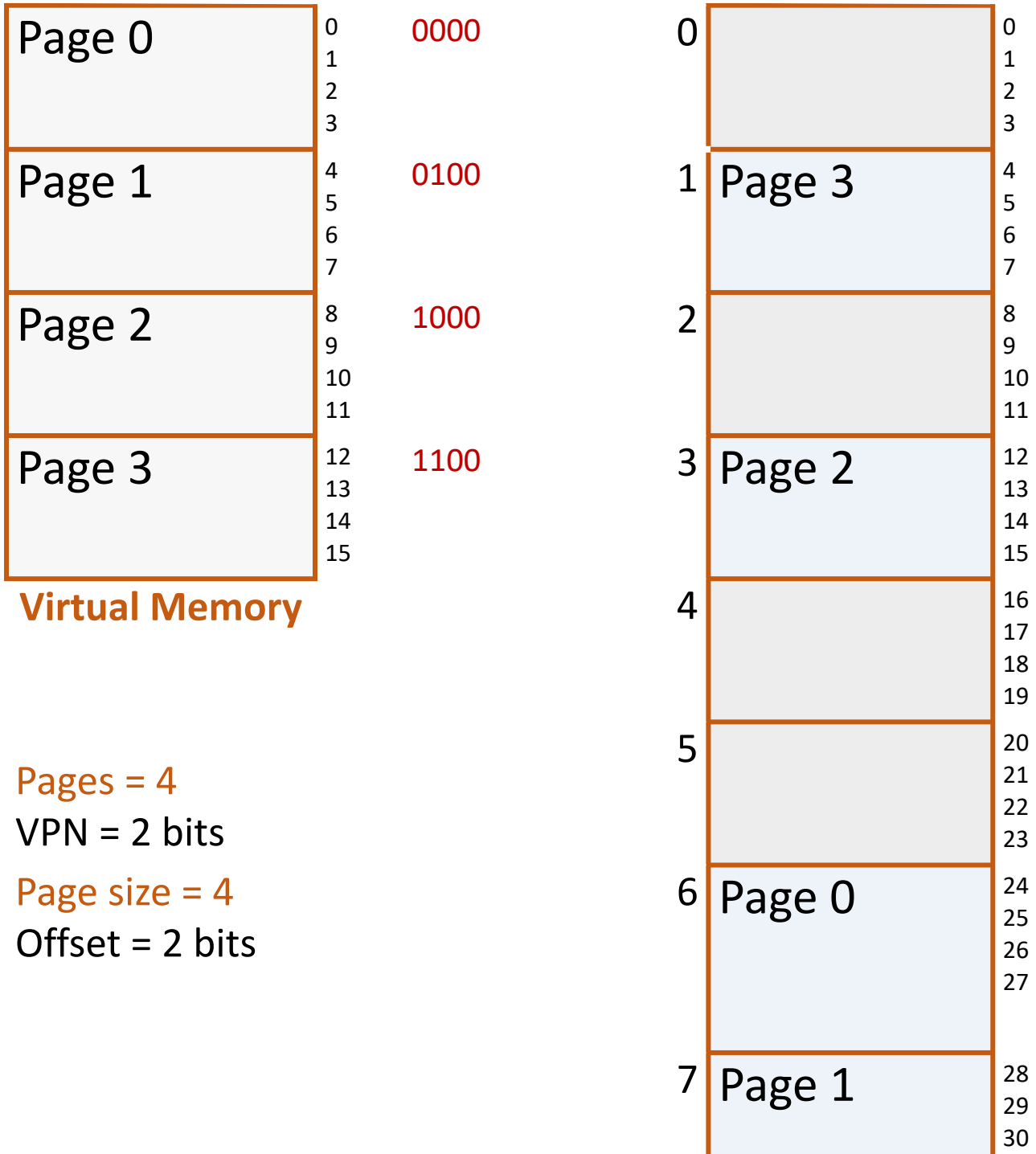


Paging : Memory Allocation



Paging : Memory Allocation





0	6
1	7
2	3
3	1

Page Table

Virtual address = 13 = 1101_2

VPN = 11_2 = 3

Offset = 01_2

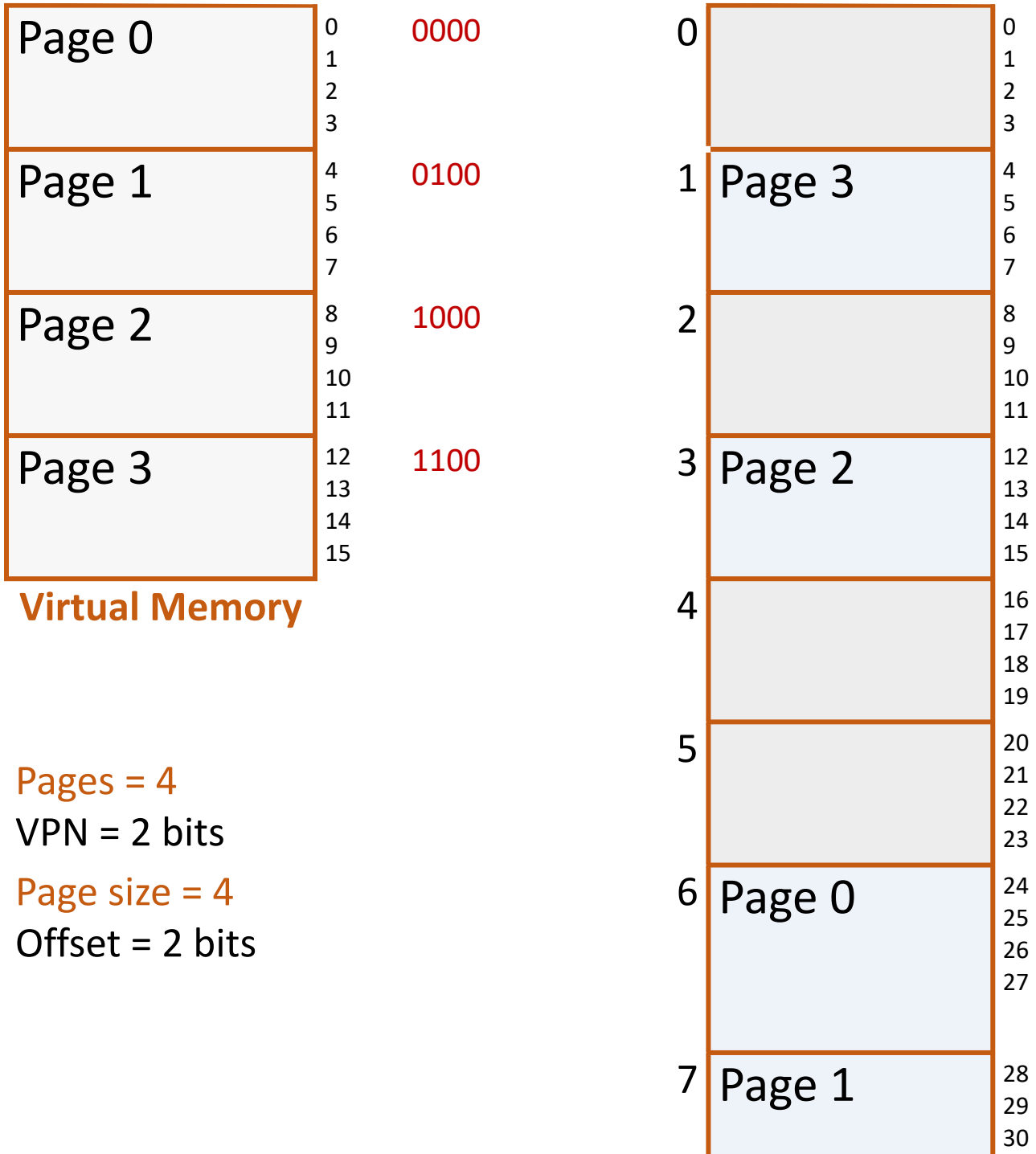
No. of Physical Pages = 8

Physical page no = 3 bits

Physical page = 1 = 001_2

Physical address = 00101_2

= 5



0	6
1	7
2	3
3	1

Page Table

Virtual address = 7 = 0111_2

VPN = 01_2 = 1

Offset = 11_2

No. of Physical Pages = 8

Physical page no = 3 bits

Physical page = 7 = 111_2

Physical address = 111111_2

= 31

Page Table Entry

- Page Table Entry consists of
  - Valid bit: indicates if PTE is valid
  - Physical page number
  - Protection bits: read, write, or execute
  - Present bit: page present in memory or swapped to disk
  - Dirty bit: if page contents have changed
  - Reference bits: used in page replacement policy

Internal Fragmentation: Calculation

- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes =35 pages + 1,086 bytes
  - Internal fragmentation = 2,048 - 1,086 = 962 bytes
- Page size = 4KB, Process size = 23MB
- Page size = 1024 bytes, Process size = 35,428 bytes
- Page size = 4096 bytes, Process size = 4,27,854 bytes

Page Table Size Computation

- Virtual address = 22 bits
- Page size = 4 KB
- Physical Memory Size = 16 MB
- Size of page table?

Page Table Size Computation

- Virtual address = 22 bits
- Page size = 4 KB = $2^2 2^{10} = 2^{12}$
- Page offset = 12 bits
- No. of bits for VPN = 22 – 12 = 10 bits
- Number of virtual pages = 2^{10}
- Number of entries in page table = 2^{10}
- Physical Memory Size = 16 MB = $2^4 2^{20} = 2^{24}$
- Bits for physical page number = 24 – 10 = 14 bits
- PTE requires 2 bytes
- Size of page table = 2 x 2^{10} bytes = 2KB

Page Table Size Computation

- Virtual address space = 950 MB
- Page size = 16 KB
- Physical Memory Size = 4 GB
- Size of page table?

Page Table Size Computation

- Virtual address = 950 MB = requires 30 bits
- Page size = 16 KB = 2^{14} (requires 14 bits)
- Physical Memory Size = 4 GB = 2^{32} B
- Page offset = 14 bits
- No. of bits for VPN = 30 – 14 = 16 bits
- Number of virtual pages = 2^{16}
- Number of entries in page table = 2^{16}
- Bits for physical page number = 32 – 14 = 18 bits
- PTE requires 3 bytes
- Size of page table = 3 x 2^{16} bytes = 192 KB

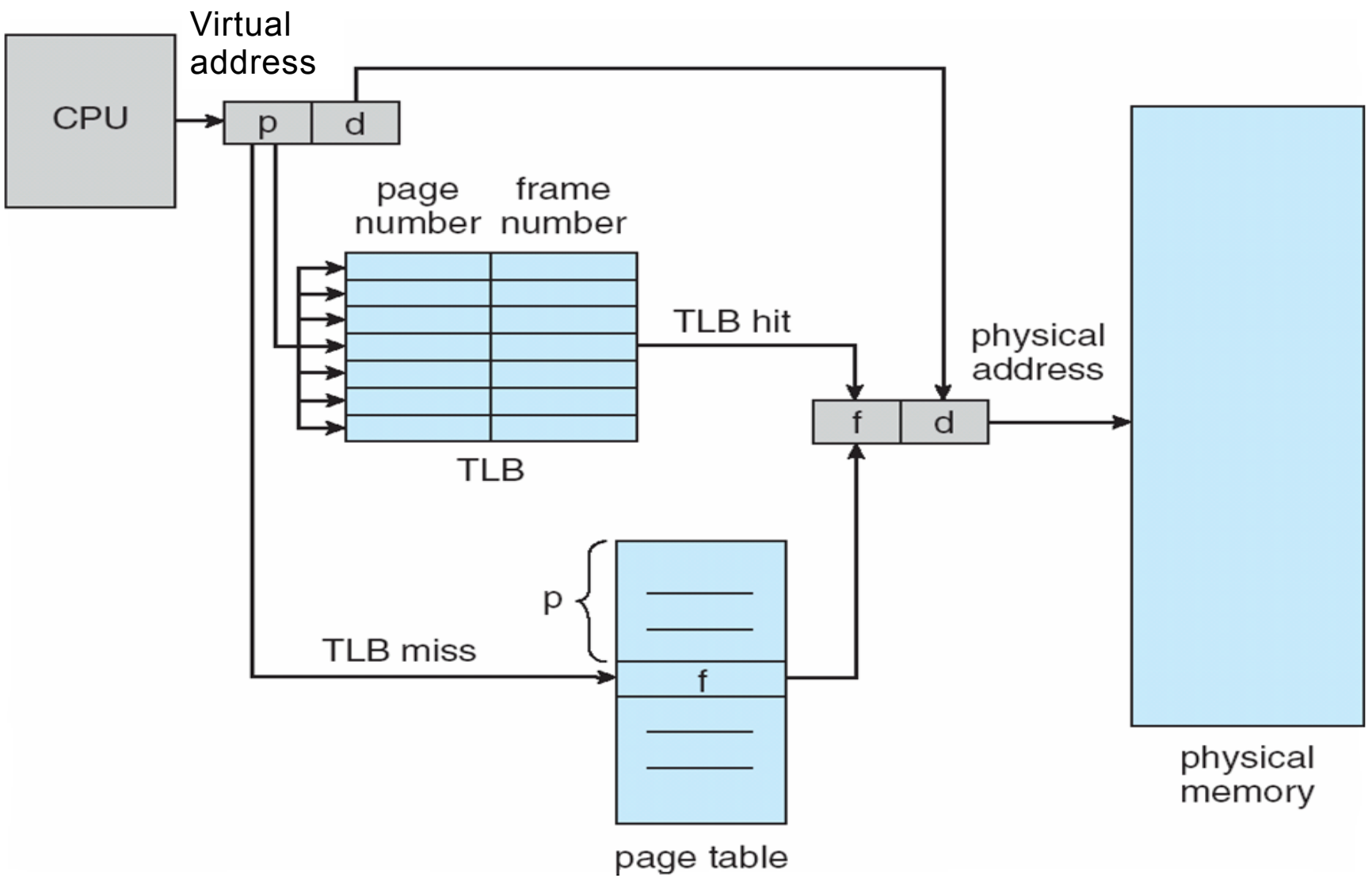
Paging Numericals

- For an embedded small computer supporting upto 1k bytes physical memory, where its virtual address is 12 bits. Suppose that the size of page/frame is 128 bytes. Use a figure to illustrate the physical and virtual address that shows the number of bits for offset and frame/page number for one level page table.
- For an embedded small computer supporting upto 1k bytes physical memory, where its virtual address is 12 bits. Suppose that the size of page/frame is 128 bytes. Use a figure to illustrate the physical and virtual address that shows the number of bits for offset and frame/page number for one level page table.
  - What is the number of virtual pages for each process?
  - How many physical frames in total?
  - How many entries in page table for each process?

Page Table Implementation

- Page table is kept in main memory
  - Page-table base register (PTBR) : starting address
  - Page-table length register (PTLR) : size of page table
- Two memory accesses required for every data/instruction
  - First access page table and
  - Second access for the data / instruction
- Faster access: special fast-lookup hardware cache called translation look-aside buffers (TLBs) used for page table
  - TLB: associative memory
  - CPU accesses PTE from TLB; on failure page table is accessed

Paging: Address Translation with TLB



TLB: Effective Page Access Time

- Hit ratio = α (percentage of times a valid PTE is found in TLB)
- Consider α = 80%,
- Access Time t_A = 20 ns for TLB,
- Access Time t_M =100 ns for memory
- Effective Access Time (EAT)

  $= (t_A + t_M)\,\alpha + (t_A + 2t_M)(1 - \alpha)$

  $= 0.80 \times 120 + 0.20 \times 220$

  $= 96 + 44 = 140$ns

TLB: Effective Page Access Time

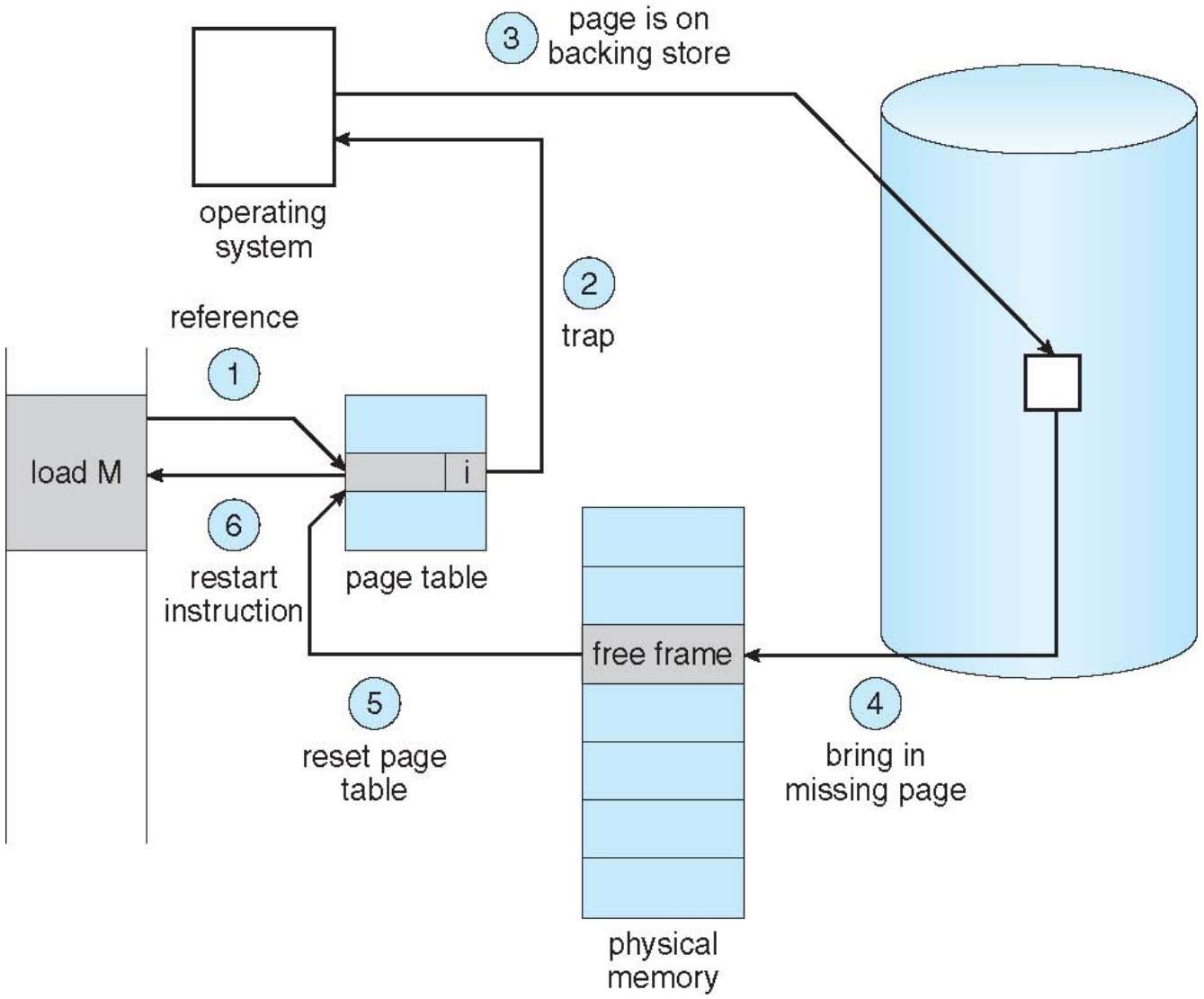
- Hit ratio = α (percentage of times a valid PTE is found in TLB)
- Consider α = 99%,
- Access Time t_A = 20 ns for TLB,
- Access Time t_M =100 ns for memory
- Effective Access Time (EAT)

$$= (t_A + t_M)\,\alpha + (t_A + 2t_M)(1 - \alpha)$$

$$= 0.99 \times 120 + 0.01 \times 220$$

$$= 118.8 + 2.2 = 121 \text{ ns}$$

Page Fault


3
page is on
backing store
operating
system
2
trap
reference
1
load M
i
6
restart
instruction
page table
free frame
5
reset page
table
4
bring in
missing page
physical
memory

Page Table Size

- Internal fragmentation (average) = half a page size
- Small page size
  - Page table size shall increase
  - Page faults may increase as pages need to be loaded into memory
  - Internal fragmentation shall decrease
- Large page size
  - Small page table
  - Memory wasted in internal fragmentation
  - Page fault shall reduce
  - More efficient to transfer to/from disk

Thank you.