Memory Management

CS3026 Operating Systems

Lecture 08

Memory Management

- Functional Requirements:
  - Relocation
    - Process images may be positioned at arbitrary locations in memory and may be relocated
    - Important for Virtual memory management
  - Partitioning
    - Create memory partitions and allocate them to processes
  - Security and Isolation
    - Protect segments of memory, isolate memory areas of processes
    - Needs hardware support
  - Sharing
    - Allow shared access to memory by different processes

Memory Management

- Non-Functional Requirements
  - Performance
    - Minimal overhead of memory management
    - Fast allocation
    - Avoid thrashing
  - Fairness
    - Avoid starvation of processes
    - Tune Working set according to process needs

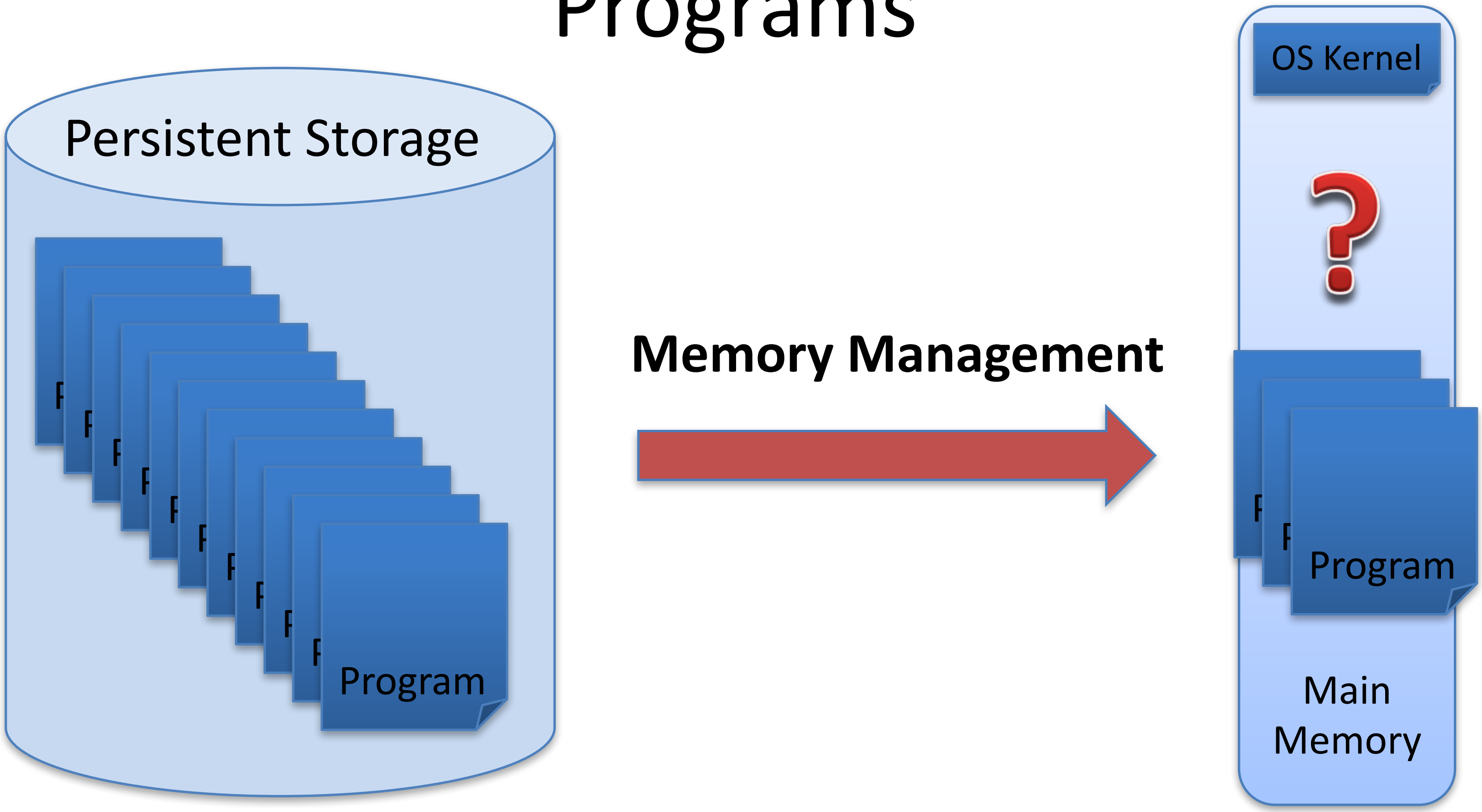
Meeting the Requirements

- Modern systems use virtual memory
  - Supported by hardware and software
  - Programs are not restricted in size by actual physical memory
  - Number of processes executing on system is not limited by physical memory
  - Based on paging and segmentation

Meeting the Requirements

- Logical Organisation
  - User programs operate within a virtual address space
  - User programs have a modular structure, each process is divided into segments (code, data, stack), mechanisms for protecting segments (read-only, execute-only) and sharing among processes
- Physical Organisation
  - Virtual memory based on a two-level hierarchy between physical memory and disk space
  - CPU can only access data in registers and physical memory
- Address Translation
  - MMU Memory Management Unit translates logical address into physical address

Multitasking – Concurrent Execution of Programs



- We need a memory management strategy to allow concurrent execution of programs

Core Concepts

Concurrency

Execution of programs concurrently

Virtualization of Processor
(processes, threads)

Principle: Context Switch
*"Allotment of **Time**"*

Virtualization

"Unlimited" resources and programs

Principle: Paging and Segmentation
*"Allotment of **Space**"*

Virtualization of Memory
(virtual address space)

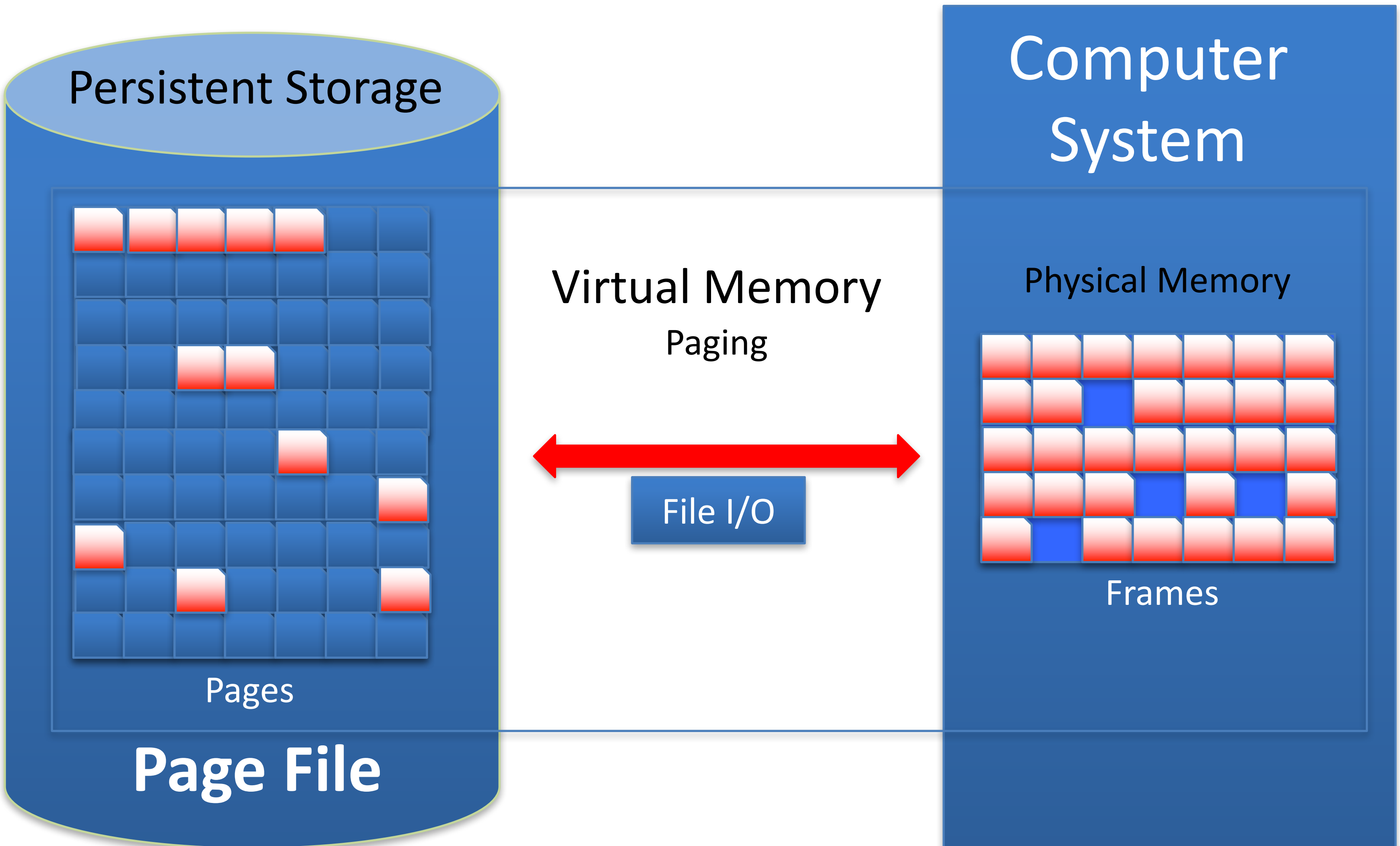
Persistence

Storage of Data

Virtual Memory Management

Virtualization

Memory Management

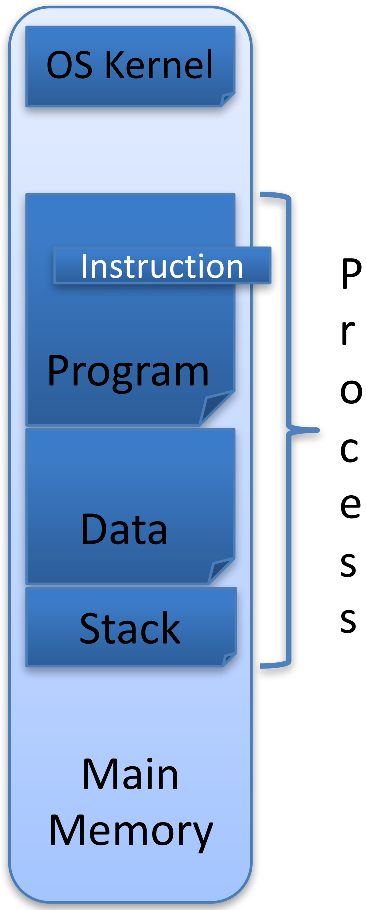


Paging

- Organise both virtual address space of process image and physical memory into fixed-sized blocks:
  - ***Frames***: Physical memory is divided into equal fixed-sized blocks, called “frames” or “page frames”
  - ***Pages***: Virtual address space of a process is divided into equal fixed-sized blocks called “pages”
  - Any page can be assigned to any free frame
- The operating system allocates a small amount of frames to a process
- Page table
  - Maps pages to frames

Virtual Memory Management

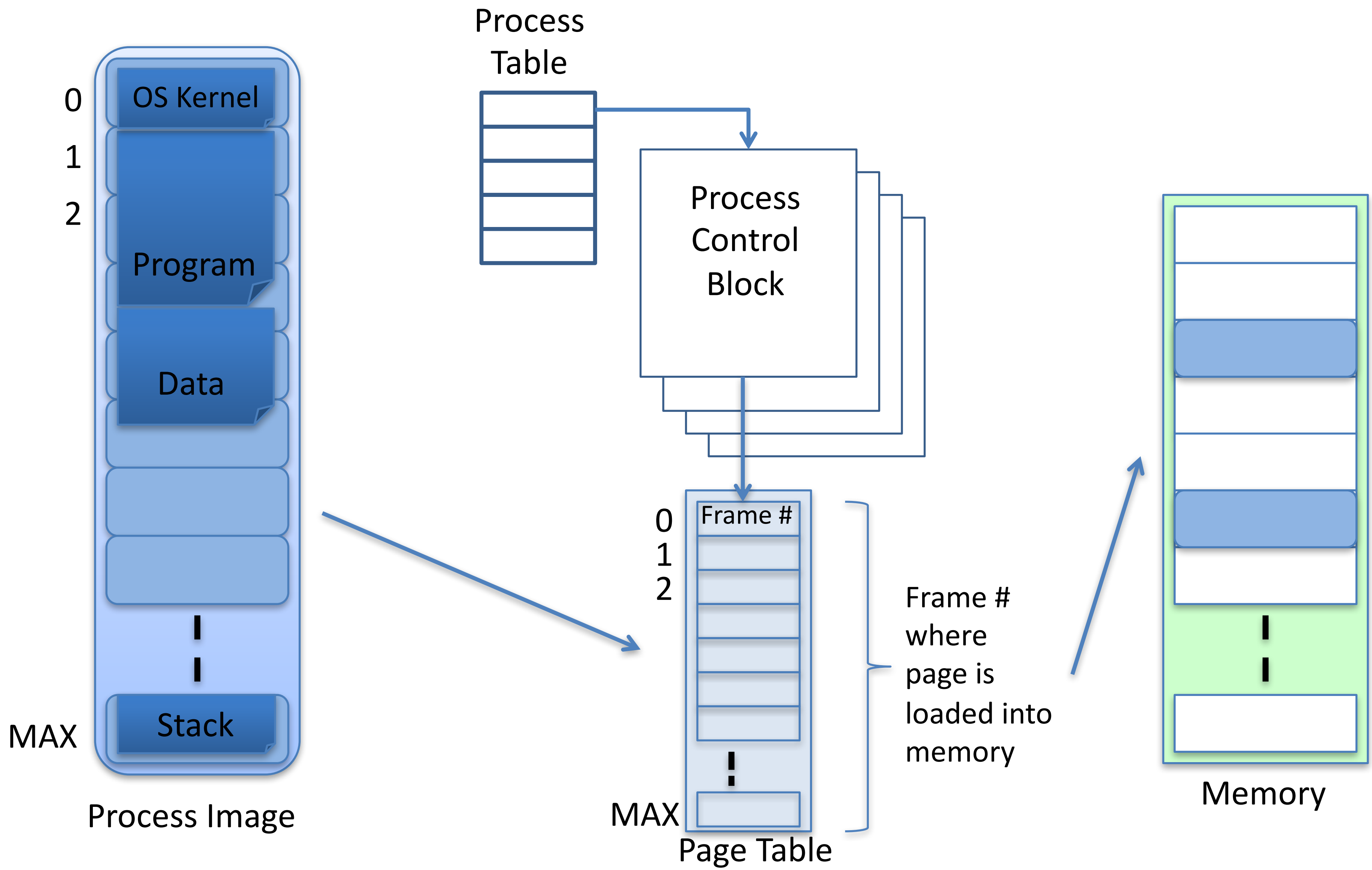
- Non-contiguous allocation of memory for processes
  - A process may be broken up into a number of pieces – pages that don’t need to occupy a contiguous area of main memory
  - Segments may start at different physical locations
- Processes only partially loaded into physical memory
  - Not all pages of a process image have to be in physical memory at the same time
  - Process is only allowed to load a limited set of pages – “working set”
- All memory references are logical addresses that are dynamically translated into physical addresses at run time



Paging and Virtual Memory

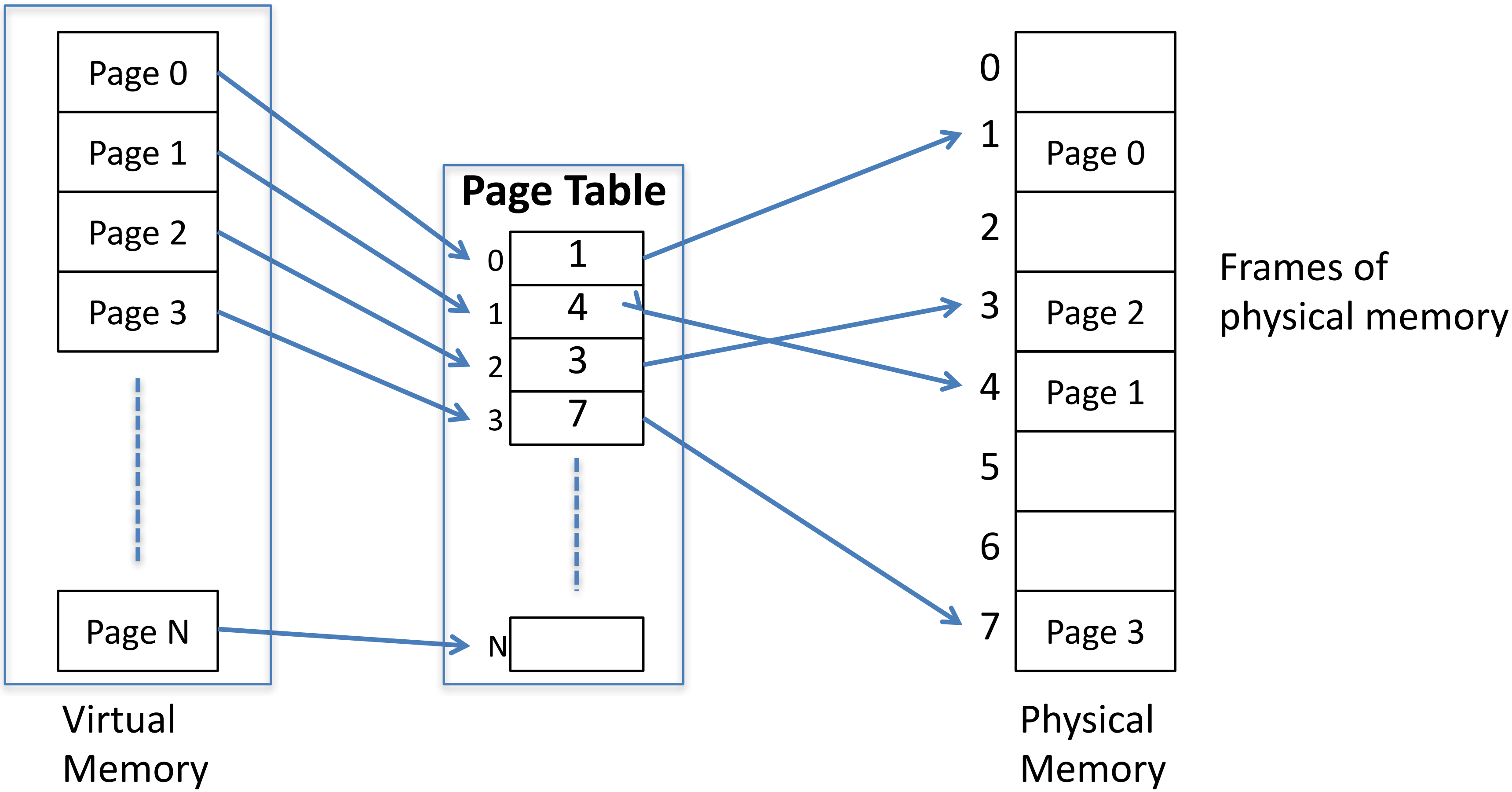
- Helps to implement virtual memory
  - Page file held on hard disk, is also organised in blocks of the same size as a page
  - Pages of a process can easily been swapped in and out of memory
  - We can implement a process image only being partially loaded, pages are loaded on demand
- No time-consuming search for free best-fit memory fragments
- No external memory fragmentation
- Paging is supported by modern hardware

Page Table of a process



Paging

- Allocation of pages to frames in physical memory can be arbitrary and non-contiguous



Page Table Management

- Operating system manages one page table per process
- A pointer to the page table is stored in the process control block
- Context switch (starts a new process)
  - the stored page table of the process has to be referenced correctly
  - We can load a “page table pointer” into one of the processor registers and use it for address translation

Paging: Address Translation

From logical to physical address

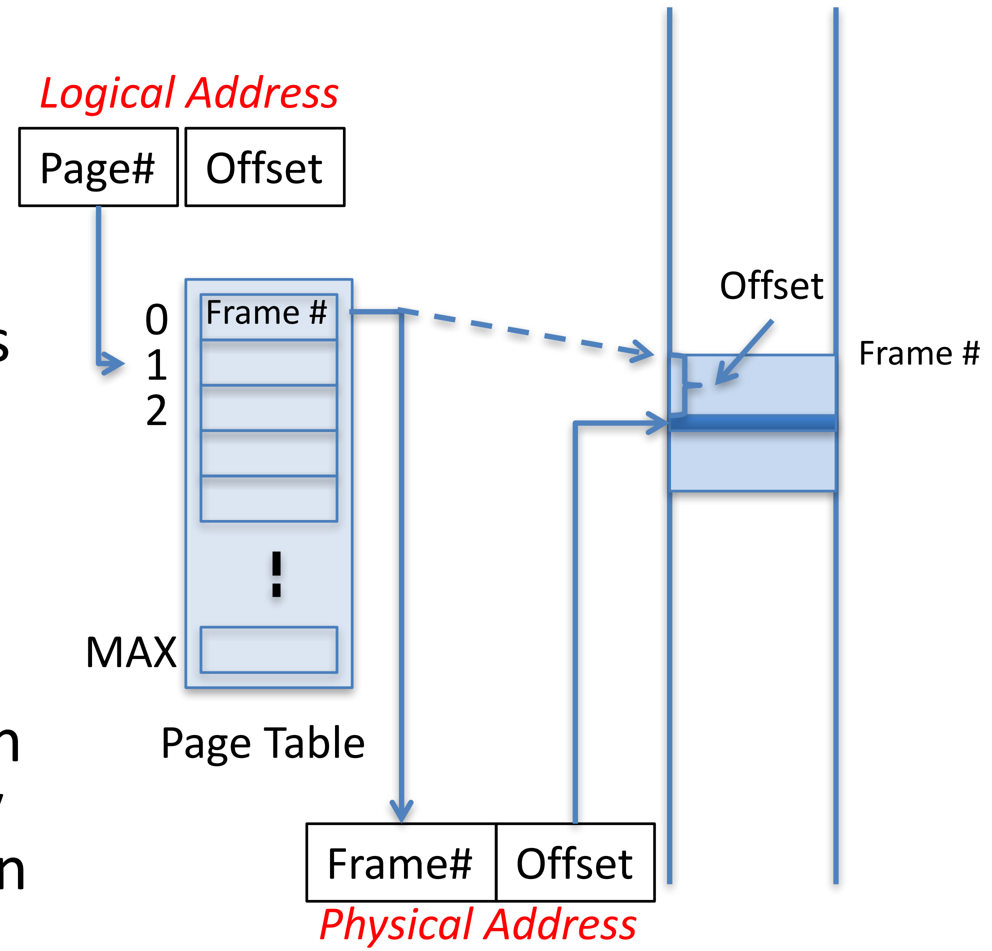
Addressing Virtual Memory

- Pages are typically of sizes between 512 bytes and 16MB
- Consider a 32-bit logical address
- Consider 4kB per page:
  - we need 12 bits ($2^{12} = 4096$) to address a location within the page
  - 20 bits for the page number: we can have 2^{20} = 1 Mio pages
    - **Page table must have 1 Mio entries !!**
- Address space: 4GB of virtual memory

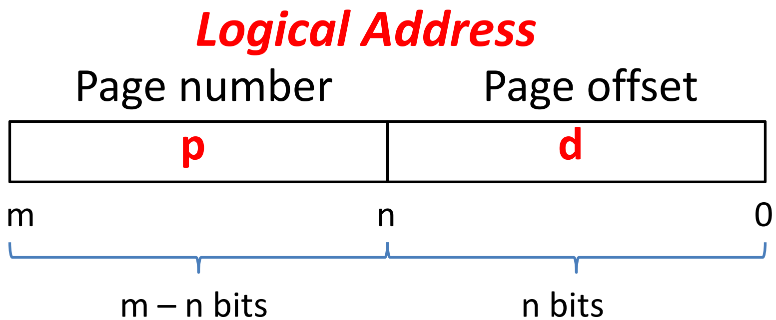
Bytes	Exponent			
1,024	2^{10}	**1kb**	**1024bytes**	
1,048,576	2^{20}	**1MB**	**1024kb**	**1024 x 1024**
1,073,741,824	2^{30}	**1GB**	**1024MB**	**1024 x 1024 x 1024**
4,294,967,296	2^{32}	**4GB**	**4 x 1024MB**	**4 x 1024 x 1024 x 1024**
1,099,511,627,776	2^{40}	**1TB**	**1024GB**	**1024 x 1024 x 1024 x 1024**
17,592,186,044,416	2^{44}	**16TB**	**16 x 1024GB**	**16 x 1024 x 1024 x 1024 x 1024**
1,125,899,906,842,620	2^{50}	**1PB**	**1024TB**	**1024 x 1024 x 1024 x 1024 x 1024**
1,152,921,504,606,850,000	2^{60}	**1EB**	**1024PB**	**1024 x 1024 x 1024 x 1024 x 1024 x 1024**
18,446,744,073,709,600,000	2^{64}	**16EB**		**16 x 1024 x 1024 x 1024 x 1024 x 1024 x 1024**

Paging: Address Translation

- A program in execution uses a logical address for addressing memory locations
- This logical address points to a particular location in virtual memory
  - Location within a virtual memory **page**
- Therefore, logical address has to contain information about the virtual memory ***page*** and the ***offset*** within this page



Paging: Address Translation



Logical address space: 2^m

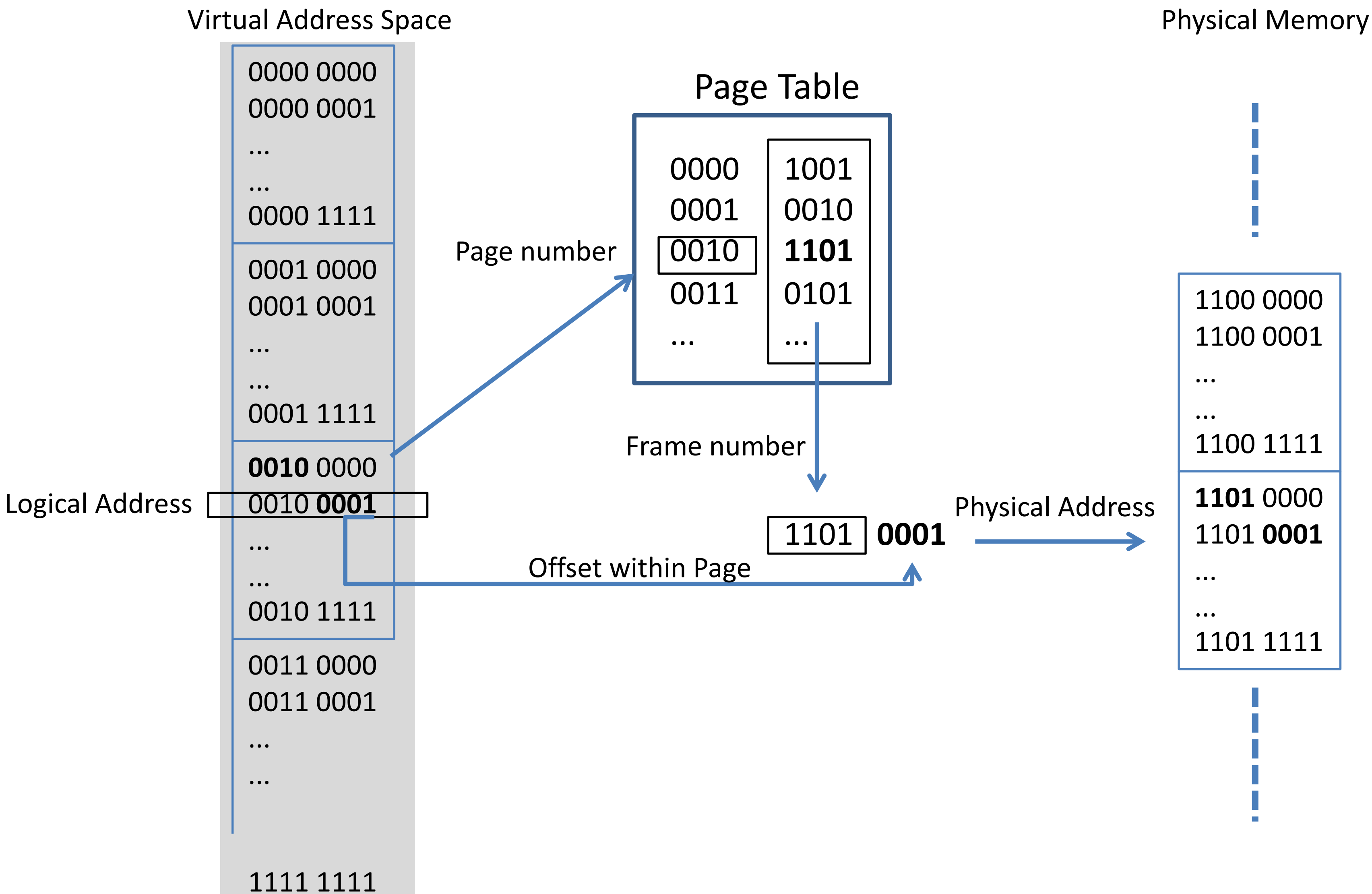
Size of a page: 2^n

Number of pages: 2^{m-n}

Size of page table

- Logical addresses have two parts
  - Page number (**p**)
    - Used as an index into the page table
    - An entry in this table contains the base address where a particular page is located in physical memory
  - Page offset (**d**)
    - Combined with the base address, defines the physical memory address within the space in memory occupied by the page
  - The high-order m-n bits encode the page number (index into page table), lower n bits encode the page offset

Paging: Address Translation



Paging: Address Translation

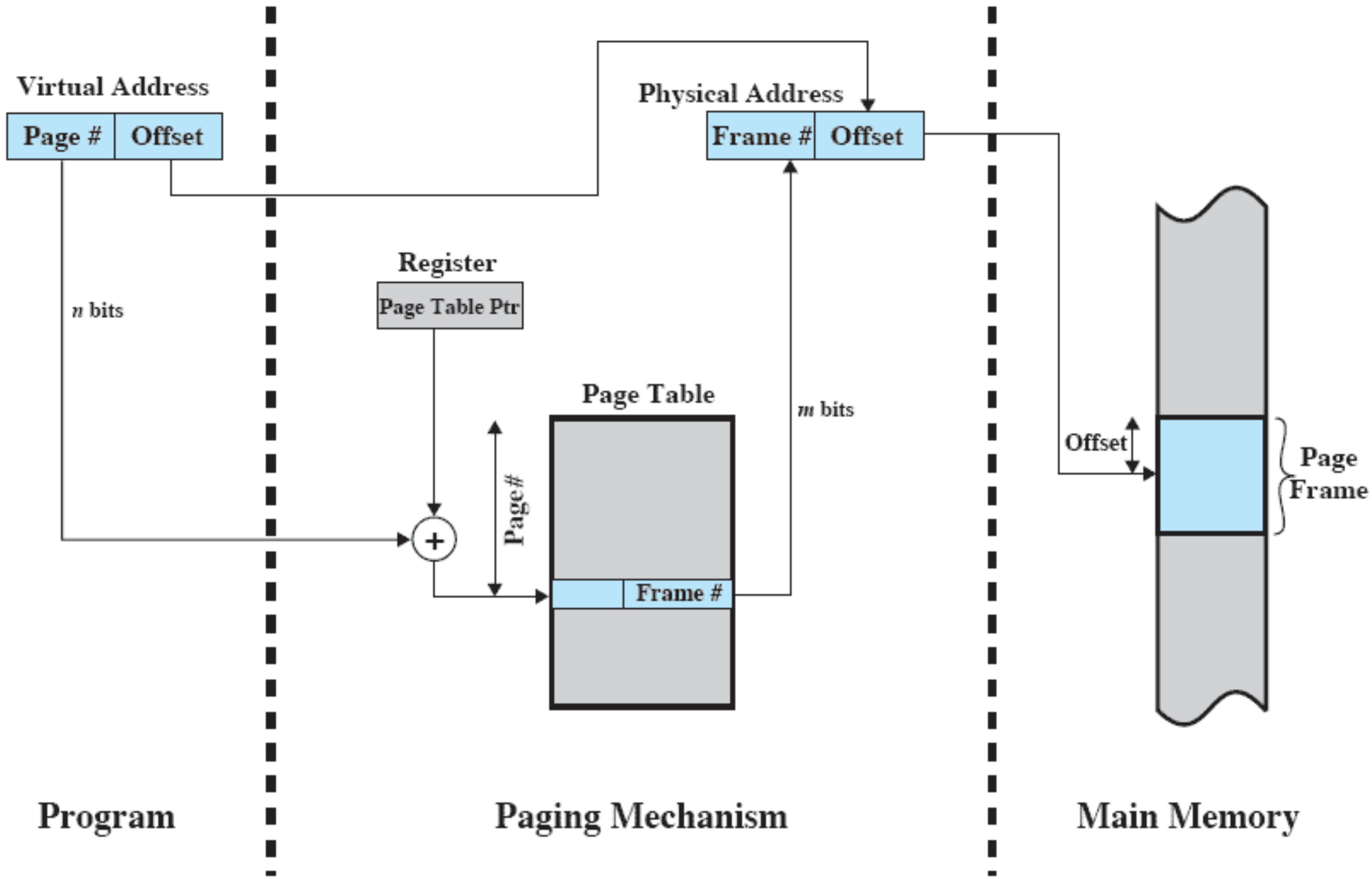
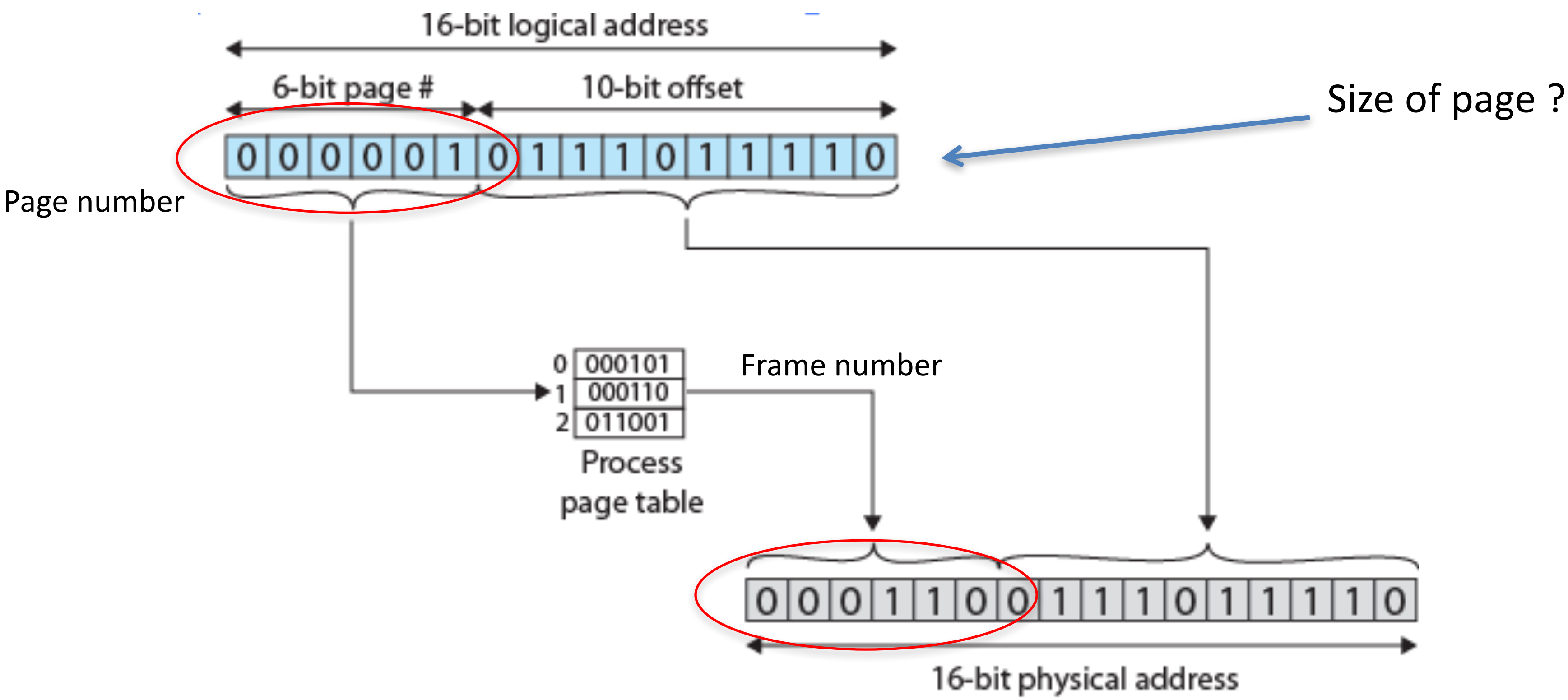


Figure 8.3 Address Translation in a Paging System

Address Translation

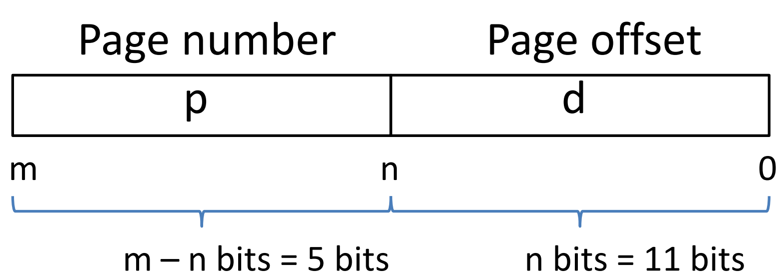


(a) Paging

- The physical address of memory location is generated by retrieving the physical base address of a page from the page table (index into page table contained in logical address) and combining it with the offset

Example

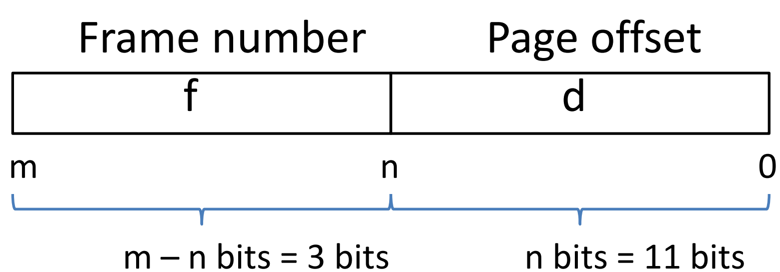
- Let's assume:
  - **Virtual memory consists of 32 pages**
  - A page is of size 2kB
  - Actual physical memory only consists of 8 frames
- How many bits needed for **logical address**?
  - Note: 1kB = 1024 bytes = 2^{10} bytes, 10 bits needed for addressing each byte within a page of size 1kB
- Addressing virtual memory (logical address space):
  - How many bits?
    - Page size: 2kB = 2048 bytes = 2 x 1024 bytes = 2 x 2^{10} = 2^{11} bytes
    - We need 11 bits for the offset within a page
    - We have 32 = 2^5 pages
    - We need 5 bits for the index in the page table (page number)



Logical Address: 16 bit

Example

- Let's assume:
  - Virtual memory consists of 32 pages
  - A page is of size 2kB
  - **Actual physical memory only consists of 8 frames**
- How many bits needed for **physical address**?
  - Note: 1kB = 1024 bytes = 2^{10} bytes, 10 bits needed for addressing each byte within a page of size 1kB
- Addressing physical memory:
  - How many bits for the physical address?
    - We need 11 bits for the offset within a frame (same size as page)
    - We have 8 = 2^3 frames
    - We need 3 bits for addressing a frame (content of page table)



Physical Address: 14 bit

Addressing Virtual Memory

- Pages are typically of sizes between 512 bytes and 16MB
- Consider a 32-bit logical address
- Consider 4kB per page:
  - we need 12 bits ($2^{12} = 4096$) to address a location within the page
  - 20 bits for the page number: we can have 2^{20} = 1 Mio pages
    - **Page table must have 1 Mio entries !!**
- Address space: 4GB of virtual memory

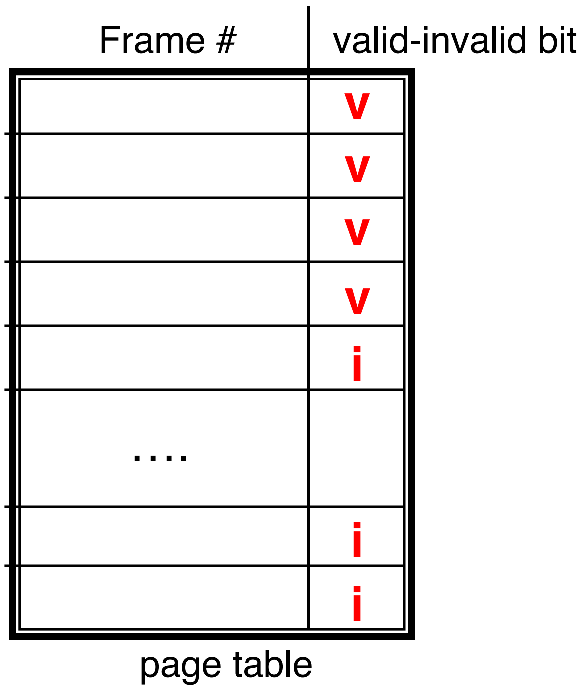
Bytes	Exponent			
1,024	2^{10}	1kb	1024bytes	
1,048,576	2^{20}	1MB	1024kb	1024 x 1024
1,073,741,824	2^{30}	1GB	1024MB	1024 x 1024 x 1024
4,294,967,296	2^{32}	4GB	4 x 1024MB	4 x 1024 x 1024 x 1024
1,099,511,627,776	2^{40}	1TB	1024GB	1024 x 1024 x 1024 x 1024
17,592,186,044,416	2^{44}	16TB	16 x 1024GB	16 x 1024 x 1024 x 1024 x 1024
1,125,899,906,842,620	2^{50}	1PB	1024TB	1024 x 1024 x 1024 x 1024 x 1024
1,152,921,504,606,850,000	2^{60}	1EB	1024PB	1024 x 1024 x 1024 x 1024 x 1024 x 1024
18,446,744,073,709,600,000	2^{64}	16EB		16 x 1024 x 1024 x 1024 x 1024 x 1024 x 1024

Fragmentation

- There is no external fragmentation
  - Any free frame can be allocated to a process that needs it
  - This is due to the fixed page size the correspondence between page size and frame size
- There is internal fragmentation
  - As page / frame sizes are fixed, there may be some waste
    - The last frame allocated to a process may not be needed in its completeness
  - Worst case: process may need n frames plus one additional byte! A complete additional frame has to be allocated just for this additional byte of required memory
  - Expected internal fragmentation: on average one-half page per process

Control Bits

- Page table entries are annotated with control bits:
  - Present or valid bit: indicates whether a page is loaded into memory or not
  - Modify bit: indicates whether a page has been modified since it was loaded
    - If not, no need to write it out when it is replaced
  - Other bits may indicate read/write protection, whether a page is shared etc.



Resident Set

- Is the number of pages currently loaded into physical memory
- Is restricted by the number of frames the operating system allocates to a process
- Allocation problem:
  - How many frames per process ??

Demand Paging

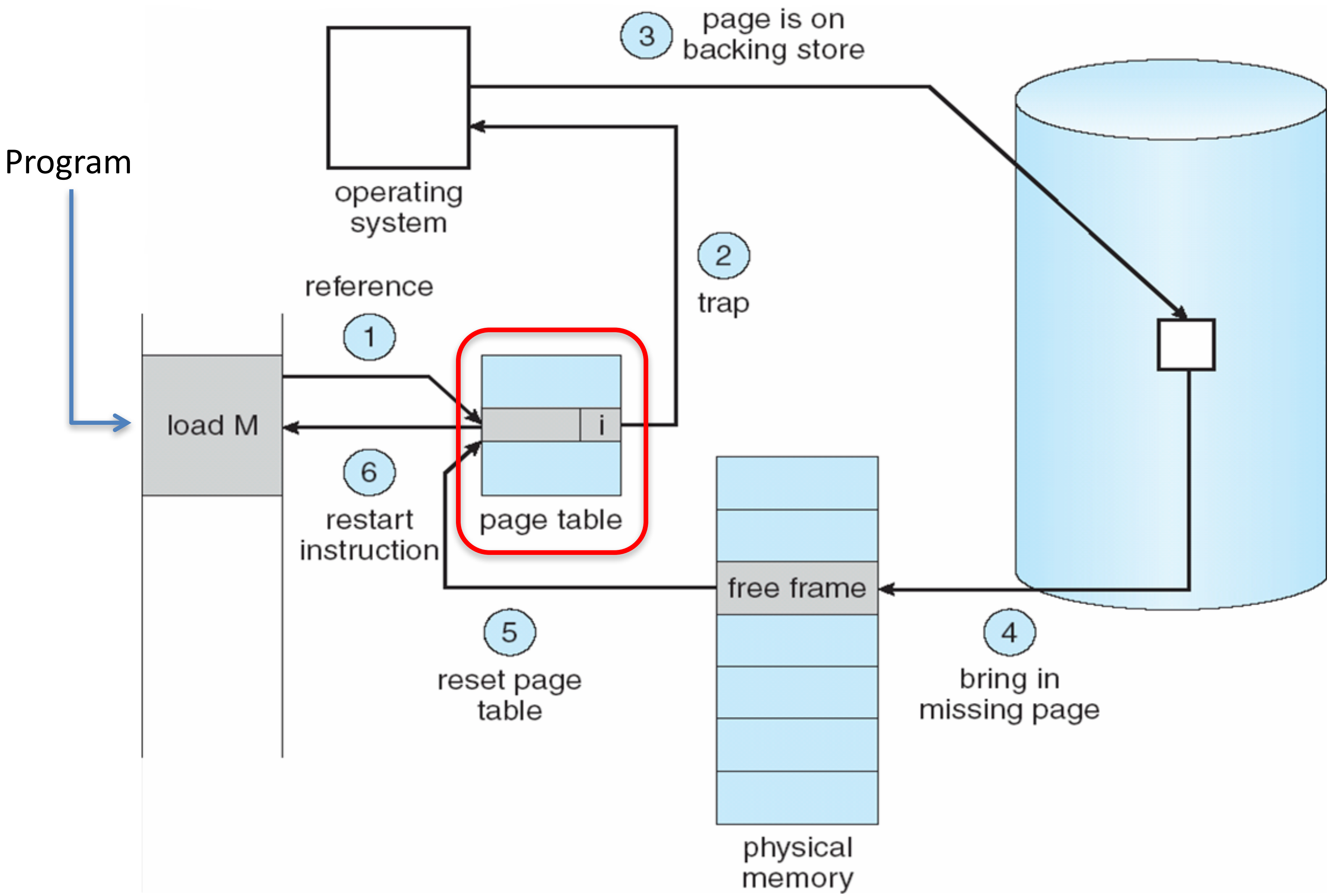
Demand Paging

- Virtualisation of memory management
  - Not all pages of a process must be in physical memory
    - A large part of the process image can be held on secondary storage
    - A page load mechanism is needed to load pages on demand
  - Only those pages that are actually needed, are loaded into memory
    - When program accesses memory, always check first whether page is in memory
  - If page is not loaded – “page fault”, leads to process interruption and I/O operations

Page Fault and Demand Paging

- If processor encounters a logical address outside the resident set of the currently executing process:
  - Process is interrupted, put in blocked state
  - I/O request is issued to read in that part of the process that contains referenced memory location
  - Other processes are scheduled for execution during this I/O activity (context switch)
  - I/O interrupt routine indicates end of read action to load needed parts of blocked process into physical memory
  - Process is put back into ready queue, waits for being rescheduled

Page Fault Handling



Principle of Locality

- Despite the elaborate and time-consuming procedure to handle page faults, virtual memory management is efficient
- Principle of Locality

  “**References to memory locations within a program tend to cluster**”

  - if there is an access to a memory location, the next access is, in all likelihood, very near to the previous one
  - Loading one page may satisfy subsequent memory access, as all required memory locations may be contained within this page, no loading of page necessary

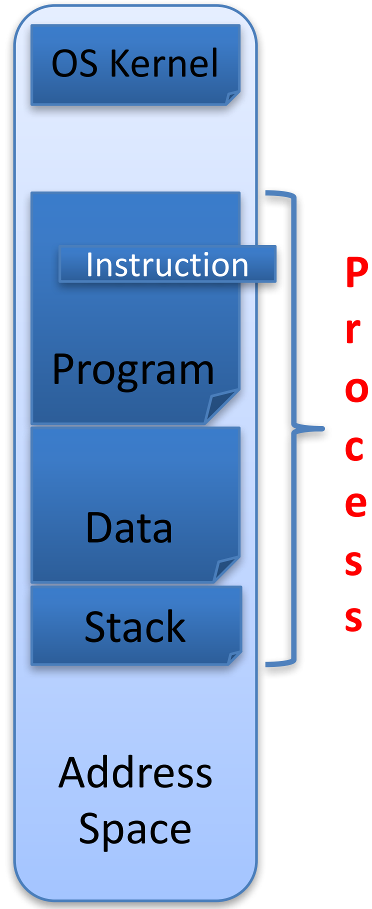
Principle of Locality

- We can observe that only a few pieces of a process image will be needed during short time periods
  - This is the current memory “locality” of a process
  - It is therefore possible to make informed guesses about which pieces will be needed in the near future
- This principle indicates that virtual memory management can be managed efficiently

Segmentation

Segmentation

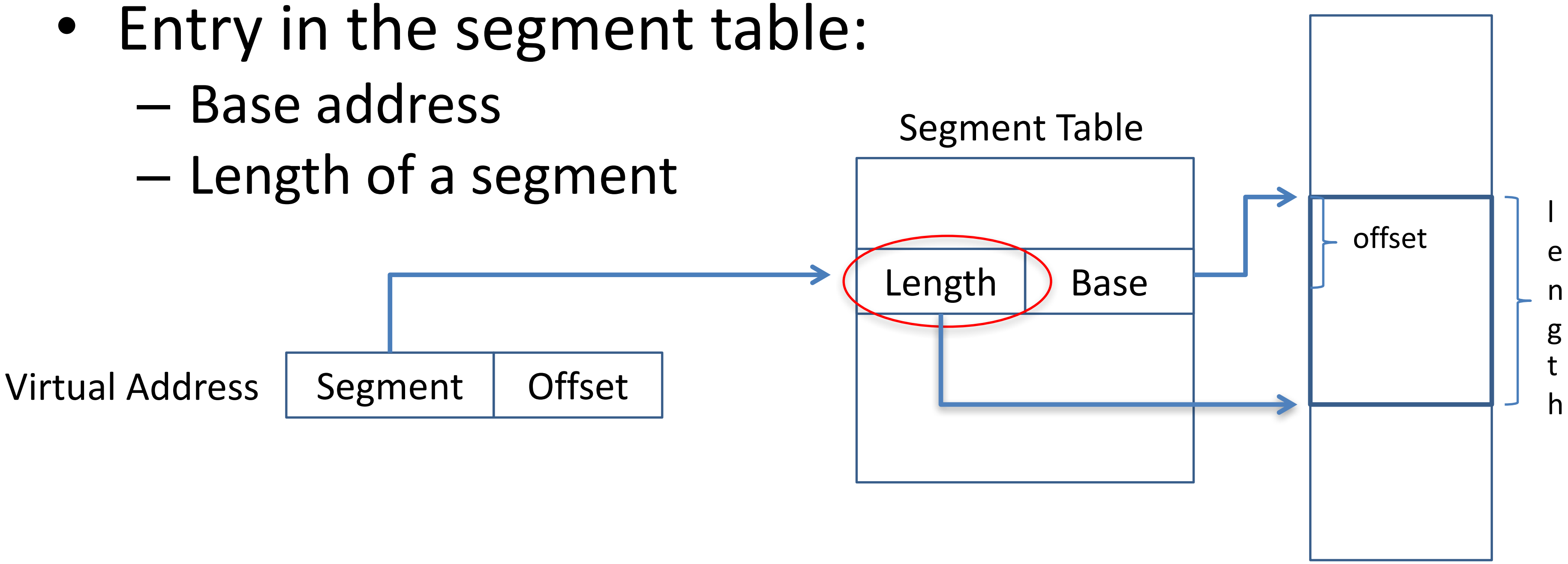
- **Each segment is a separate address space**
  - Has a base address where the segment starts, memory within a segment is addressed relative to this base address
  - Allows parts of a program code to be altered and recompiled independently
    - Shared libraries
- Segments allow Programs to be divided into logical units of variable length
  - Process image regarded as a collection of segments
    - Code segment
    - Data segment
    - Stack, Heap, shared libraries etc.
- Segments are visible to programmer
- In programs, a “virtual address” to a memory location consists of two parts: starting location of the segment and offset within the segment



Virtual Address	Segment ***i***	Offset

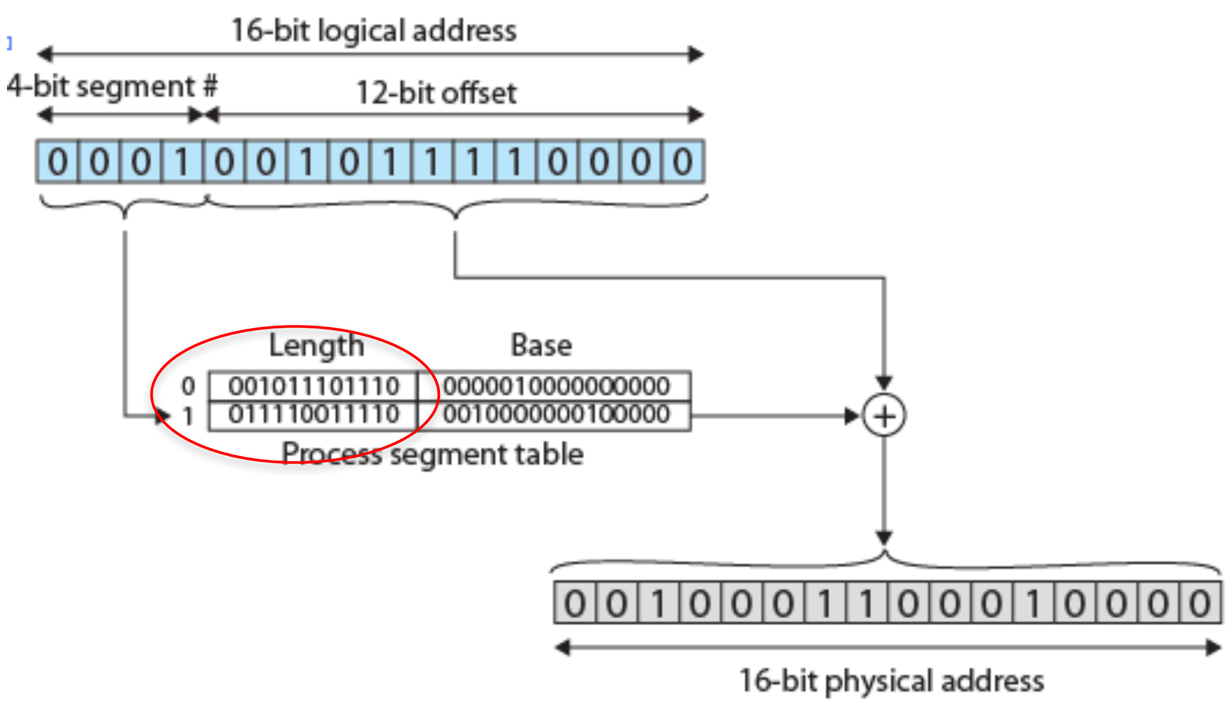
Segmentation

- Each process has a segment table
  - Each segment has a base address
  - Segments are of variable size
    - Segment table entry has to specify the **length** of a segment
  - Held in the process control block
- Entry in the segment table:
  - Base address
  - Length of a segment



Segmentation

- Each segment is a separate logical address space
  - Programmer may view memory as consisting of multiple address spaces
- Address translation via process segment table



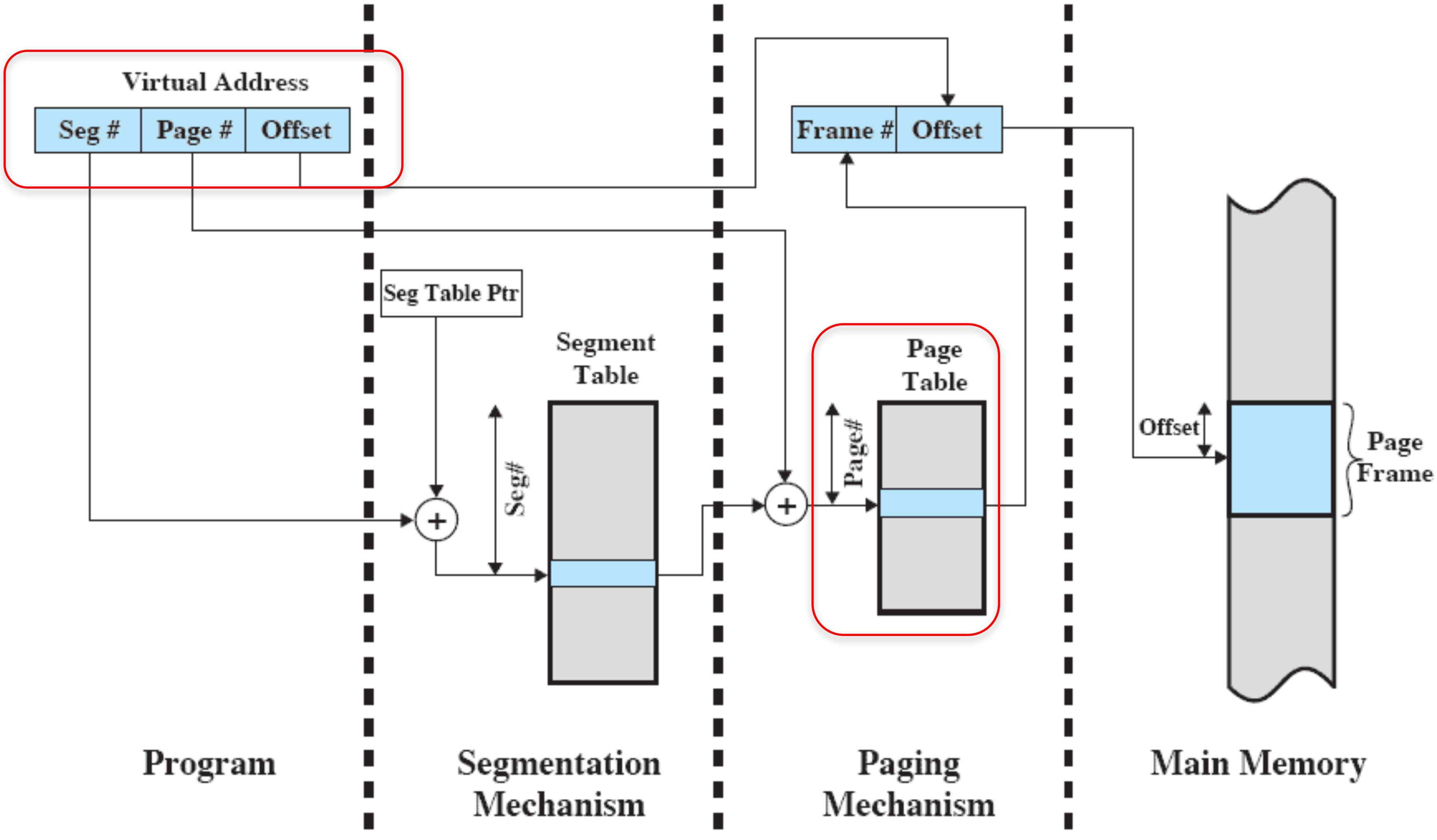
(b) Segmentation

Segmentation

- With segmentation, program may occupy non-contiguous memory
  - Each segment in separate memory area
  - Better utilisation of memory
    - Smaller contiguous segments of memory needed
- Segmentation is a dynamic partitioning scheme
  - No internal fragmentation, but external fragmentation

Combined Segmentation and Paging

- Each segment is broken into fixed-size pages



Segmentation and Paging

- Virtual address consists of at least 3 elements
  - Segment number
  - Page number
  - Offset within page
- Each segment is a separate virtual address space
  - Consists of a set of fixed-sized pages

Protection and Sharing

- Each segment can have special protection (read / write / execute policies)
  - Each segment is specified in physical memory by a base address and its length, each memory access has to be checked against these two parameters
  - E.g.: text segment is read-only
- Segments can be shared among processes