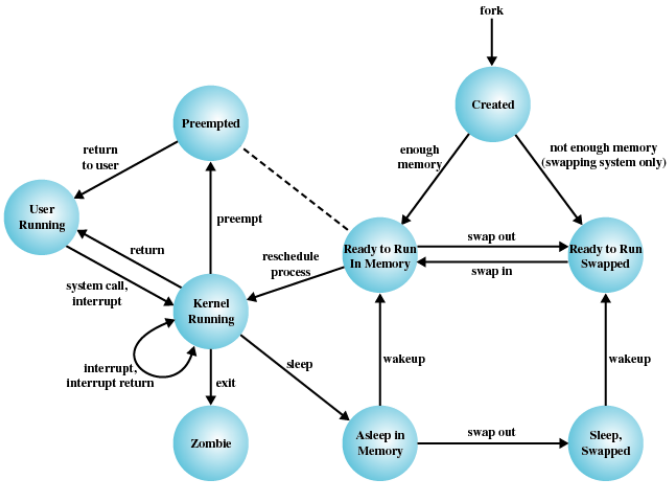
Operating Systems & Computer Networks

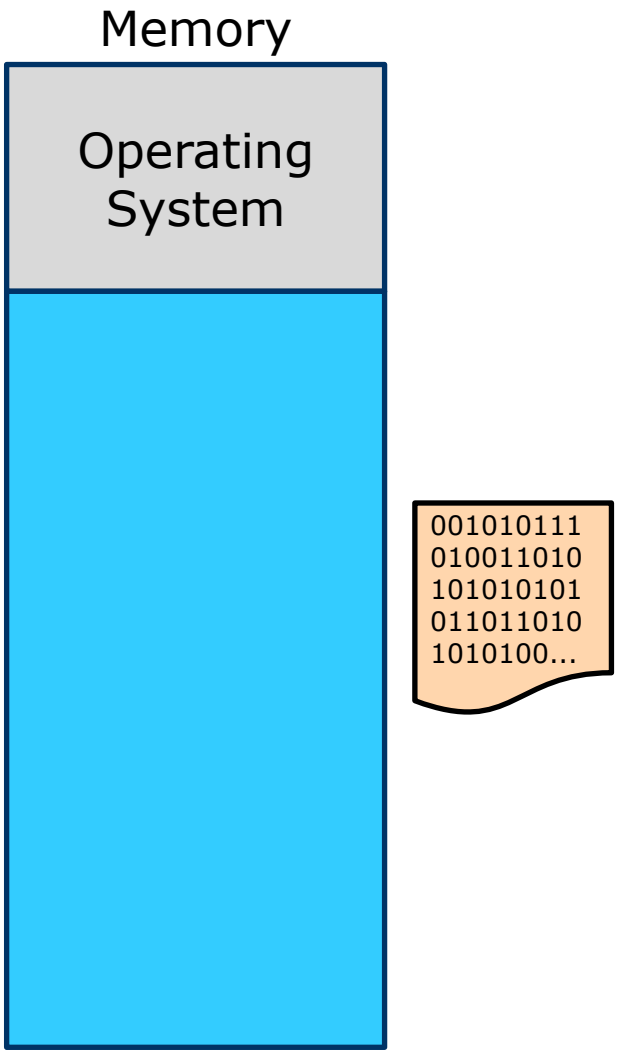
Memory



Content

Motivation

- To which location in memory should the process image be loaded?
- What happens to all the addresses contained in the process image?
- How does the OS know that no other process is using that memory?
- How can the OS prevent a process from accessing memory that it doesn't "own"?
- What's the best method to efficiently manage memory requests?



Motivation

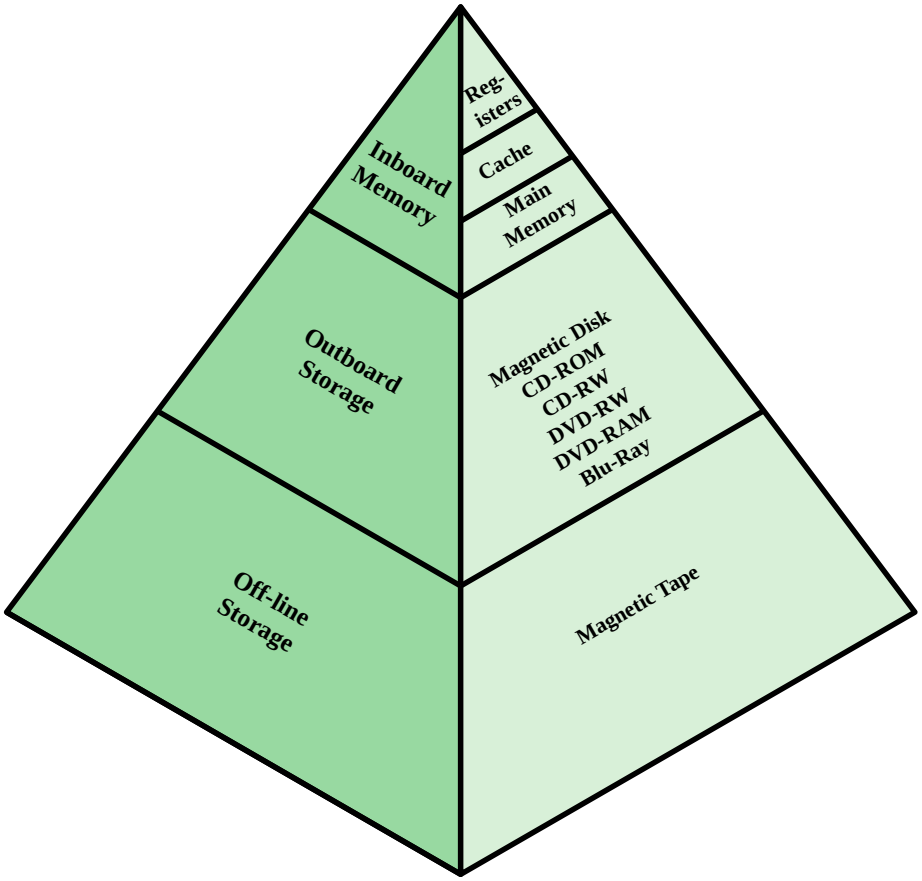
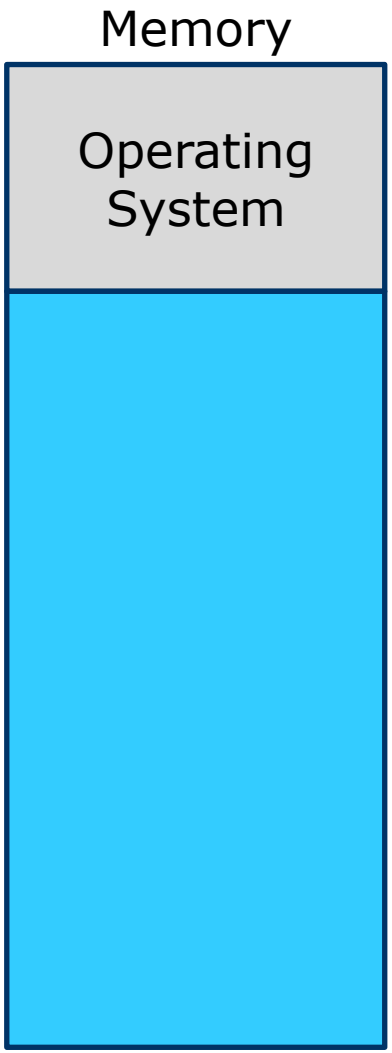


Figure 1.14 The Memory Hierarchy

Memory Management

- Closely related to processes
  - Memory management isolates processes from each other
- Goals
  - Subdividing memory to accommodate multiple processes
  - Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time
- Requirements
  - Relocation: Location in (physical) memory unknown or may change
  - Protection: Disallow access to memory of other processes
  - Sharing: Data for communication (IPC), program copy for memory reduction



Addressing

- Physical Address
  - The absolute address or actual location in main memory
  - ➢ Used by the kernel (to implement logical addresses)
- Relative Address
  - Address expressed as a location relative to some known point
  - Also commonly found in application programming (arrays)
- Logical/Virtual Address
  - Reference to memory location independent of current assignment of data to memory
  - Translation must be made to physical address
    - ➢ Requires hardware support
- Address space
  - Range of addresses that are (within the address space) unambiguously addressable

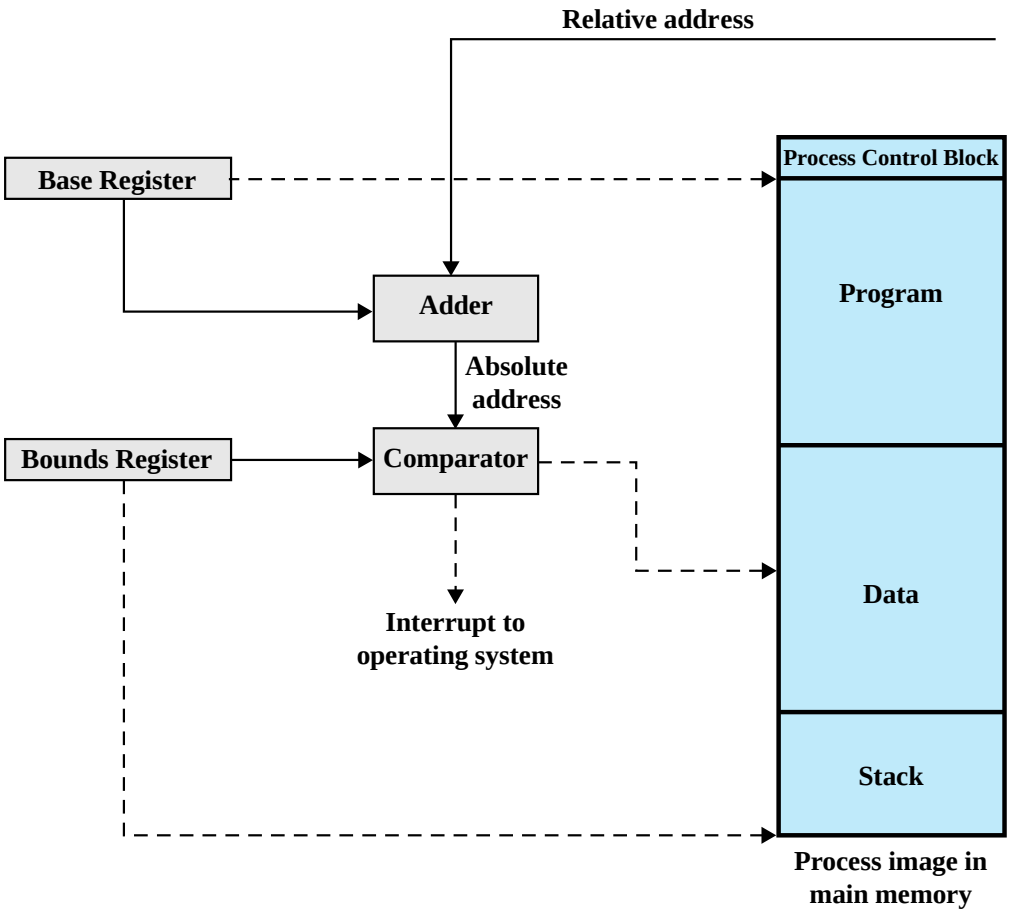
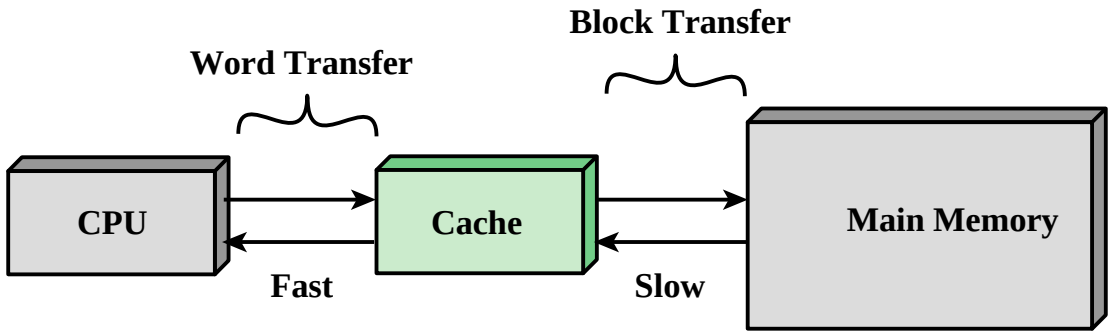
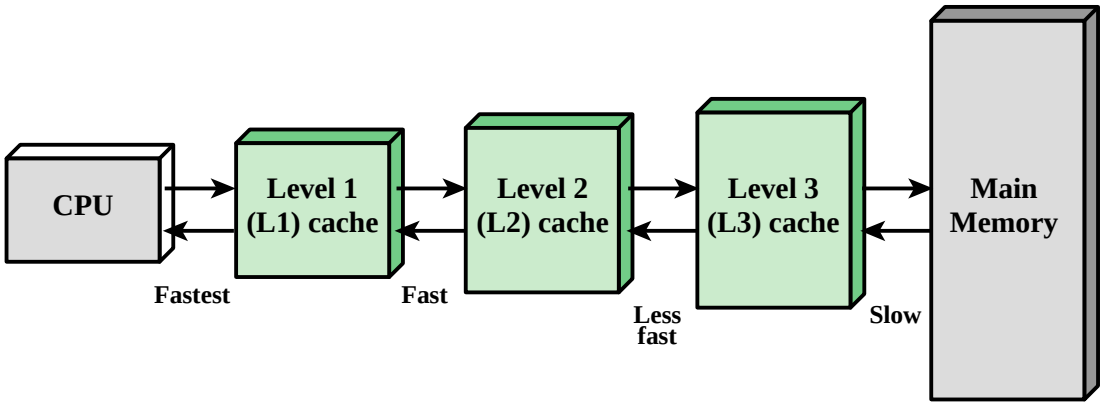


Figure 7.8 Hardware Support for Relocation

Memory Access



(a) Single cache

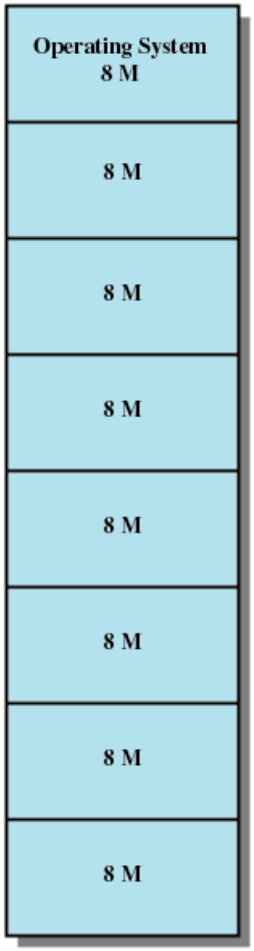


(b) Three-level cache organization

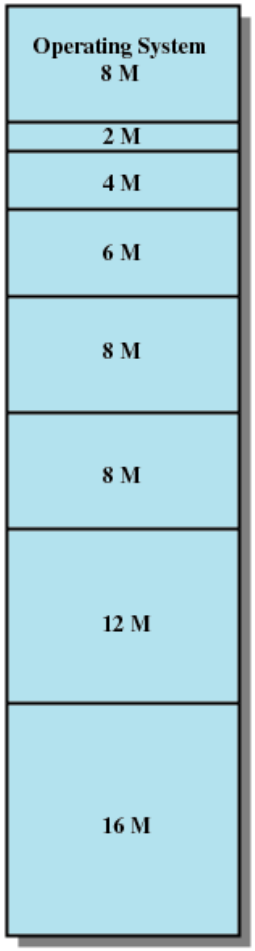
Figure 1.16 Cache and Main Memory

Fixed and Dynamic Partitioning

Fixed Partitioning



(a) Equal-size partitions

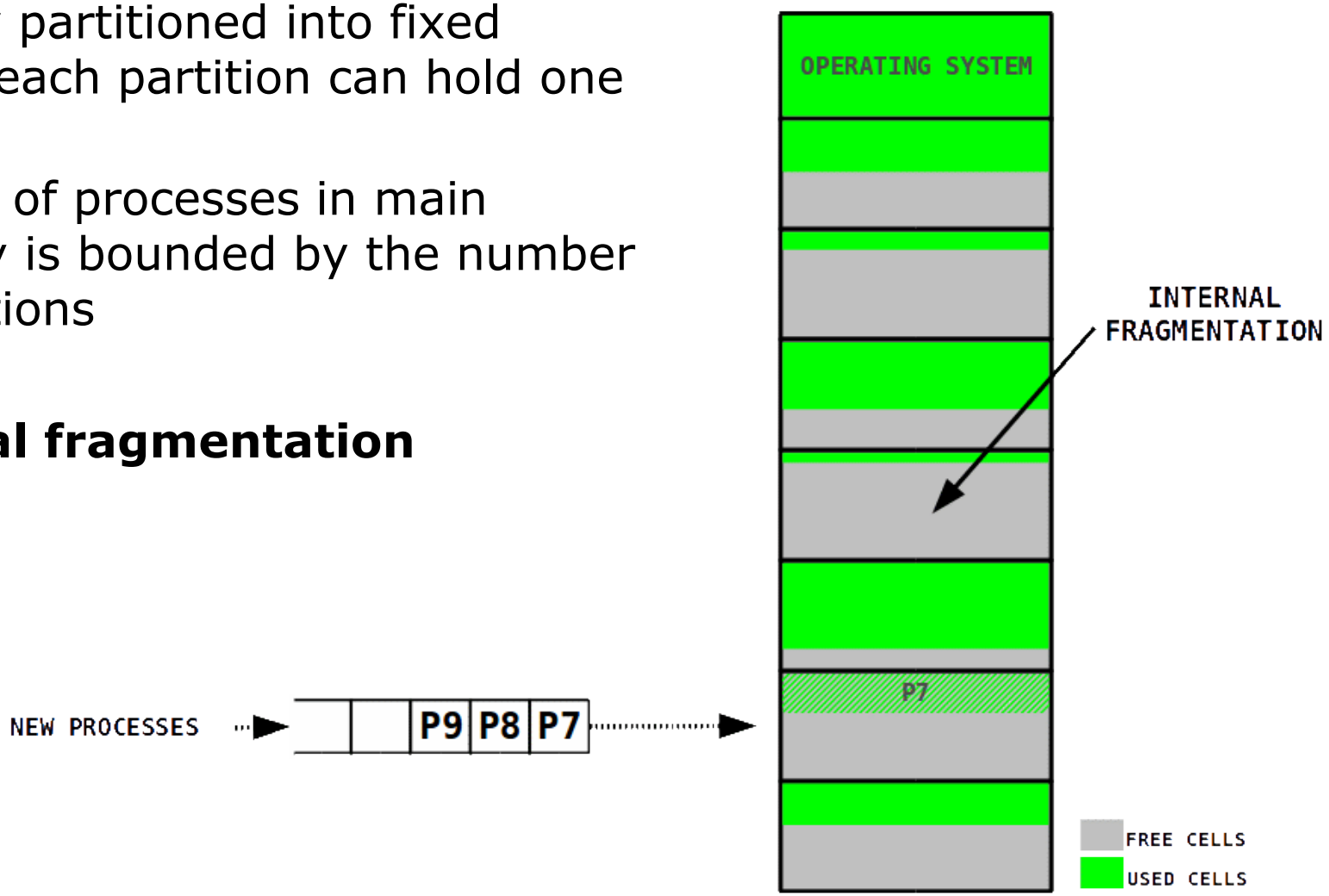


(b) Unequal-size partitions

Fixed Partitions

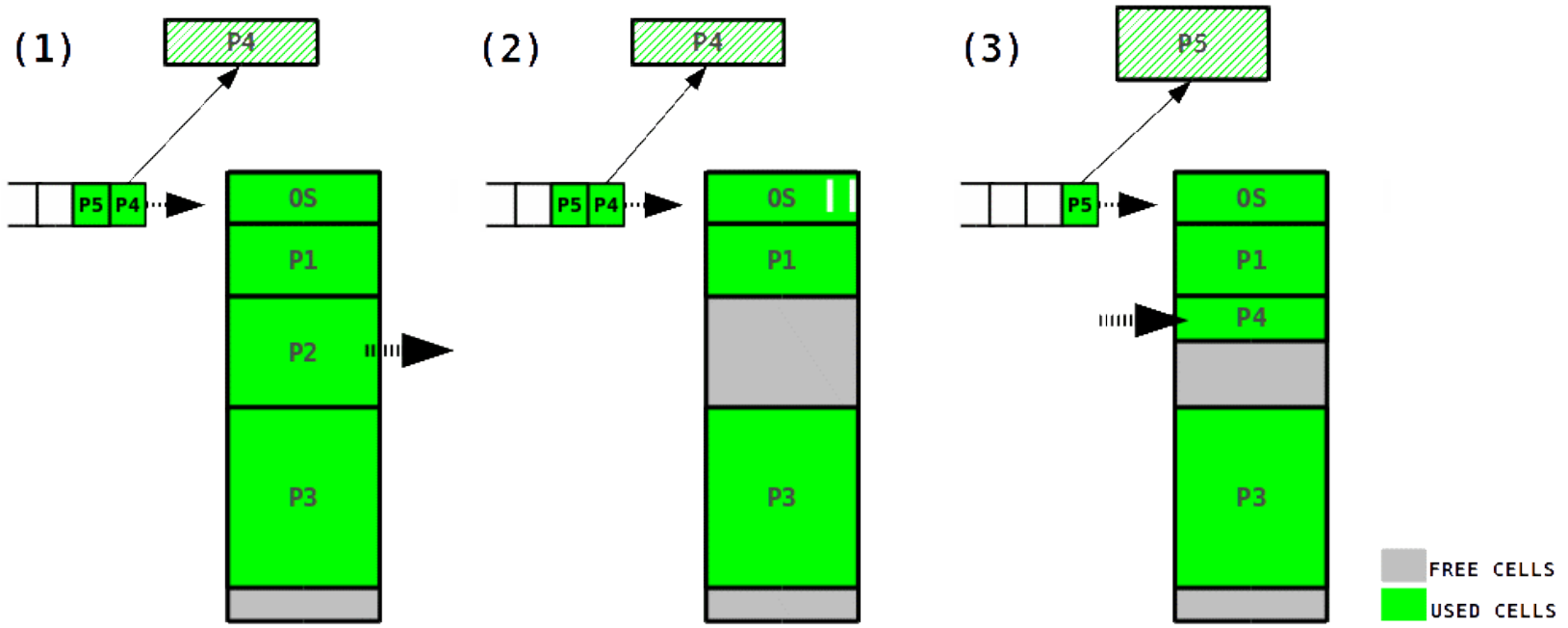
- Memory partitioned into fixed pieces, each partition can hold one process
- Amount of processes in main memory is bounded by the number of partitions

➢ **Internal fragmentation**



Dynamic Partitions

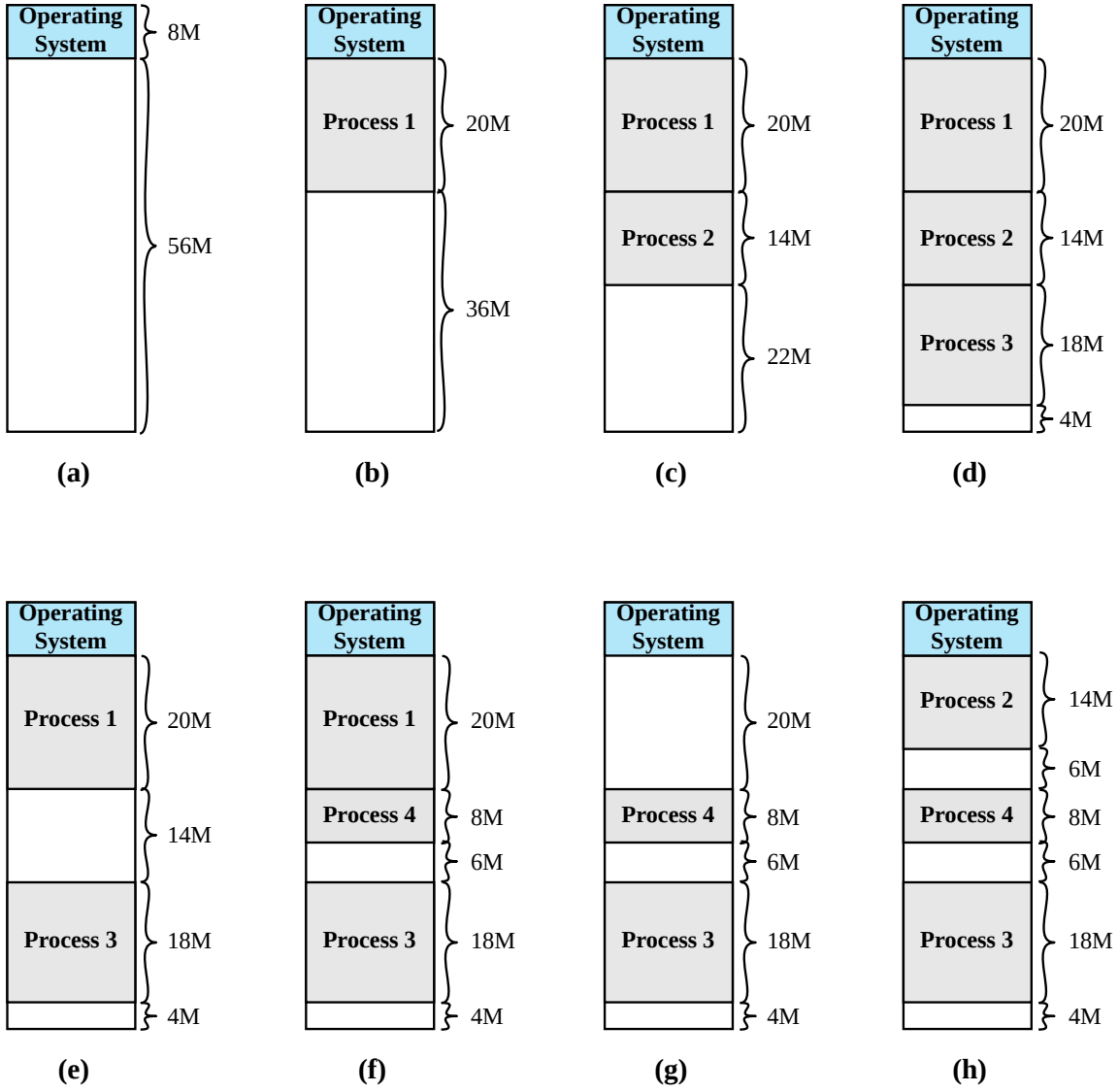
- Memory is divided into variable sized partitions on demand



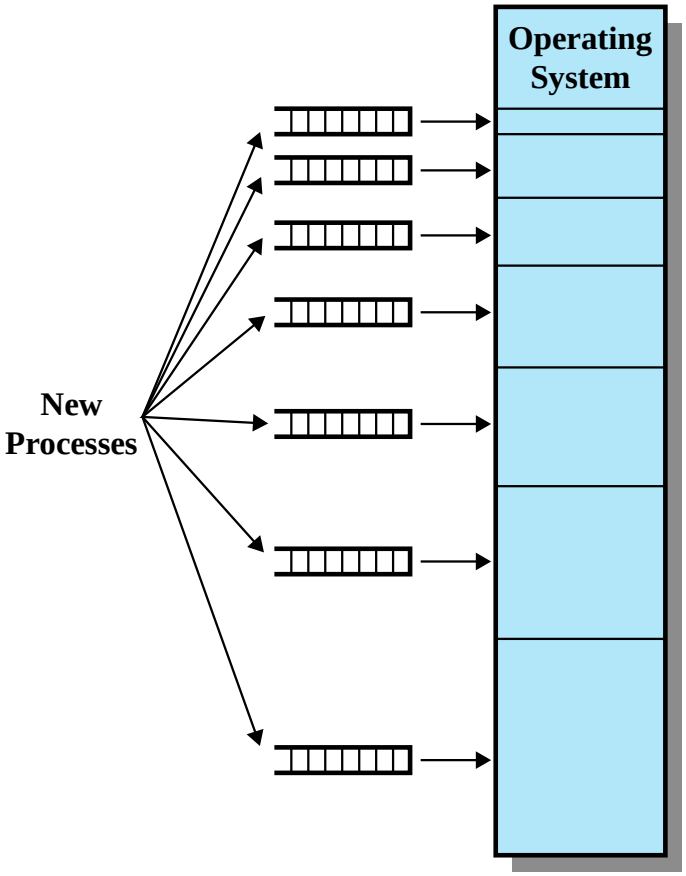
- Although there is enough space left for P5 it can not be allocated to the process because it is not continuous

➢ **External fragmentation**

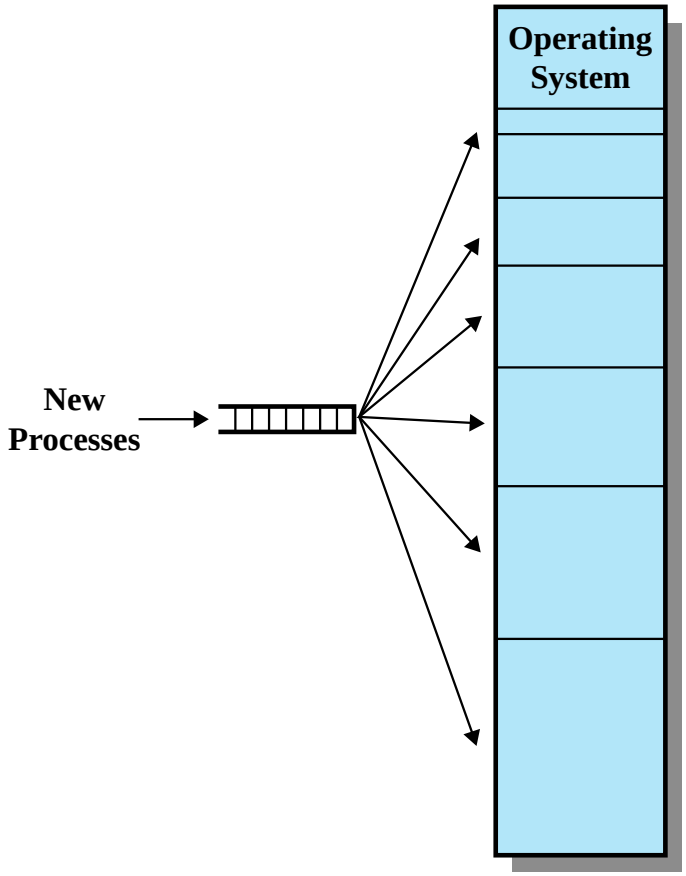
Dynamic Partitioning



Implementation



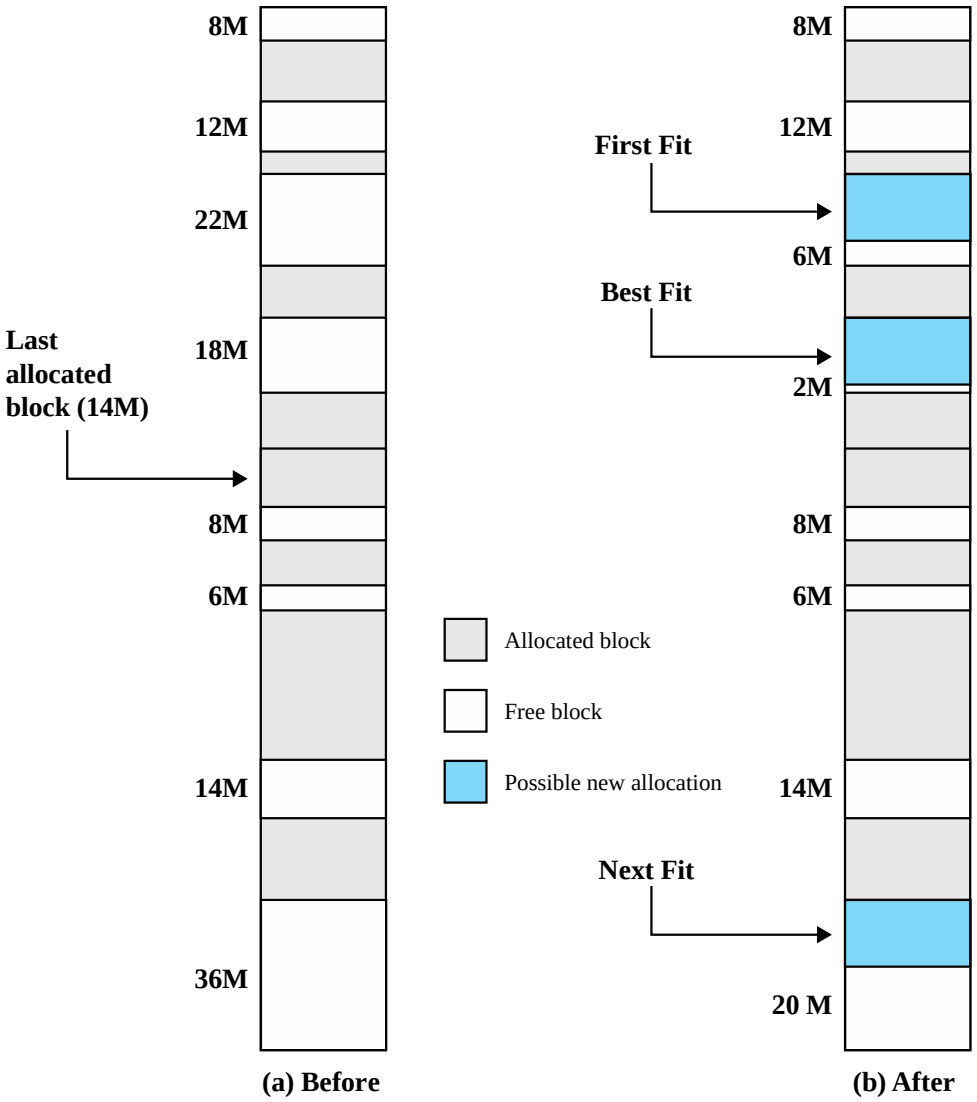
(a) One process queue per partition



(b) Single queue

Dynamic Placement Algorithms

- First-fit algorithm:
  - Scans memory from the beginning
  - ➢ Chooses first available block that is large enough
- Next-fit algorithm:
  - Scans memory from the location of the last placement
  - ➢ Tends to allocate block of memory at end of memory (where largest block is commonly found)



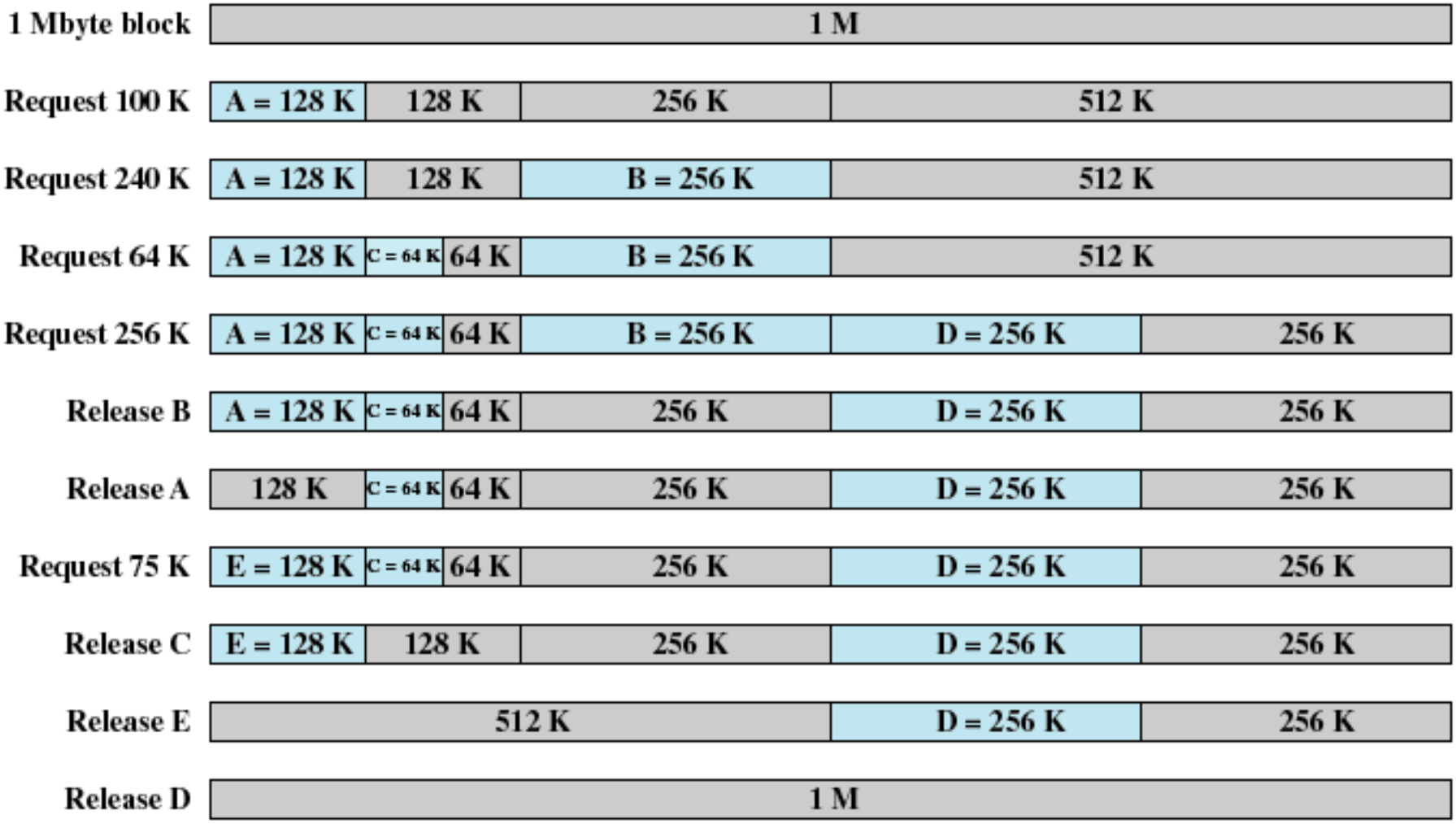
Buddy System

- Combines advantages of fixed and dynamic allocation
- Entire available space is treated as single block of size 2^U bytes
  - U := number of bits in address
- If memory of size s is requested ($2^{U-1} < s <= 2^U$), entire block is allocated
  - Otherwise block is split into two equal buddies
  - Process continues until smallest block greater than or equal to s is generated

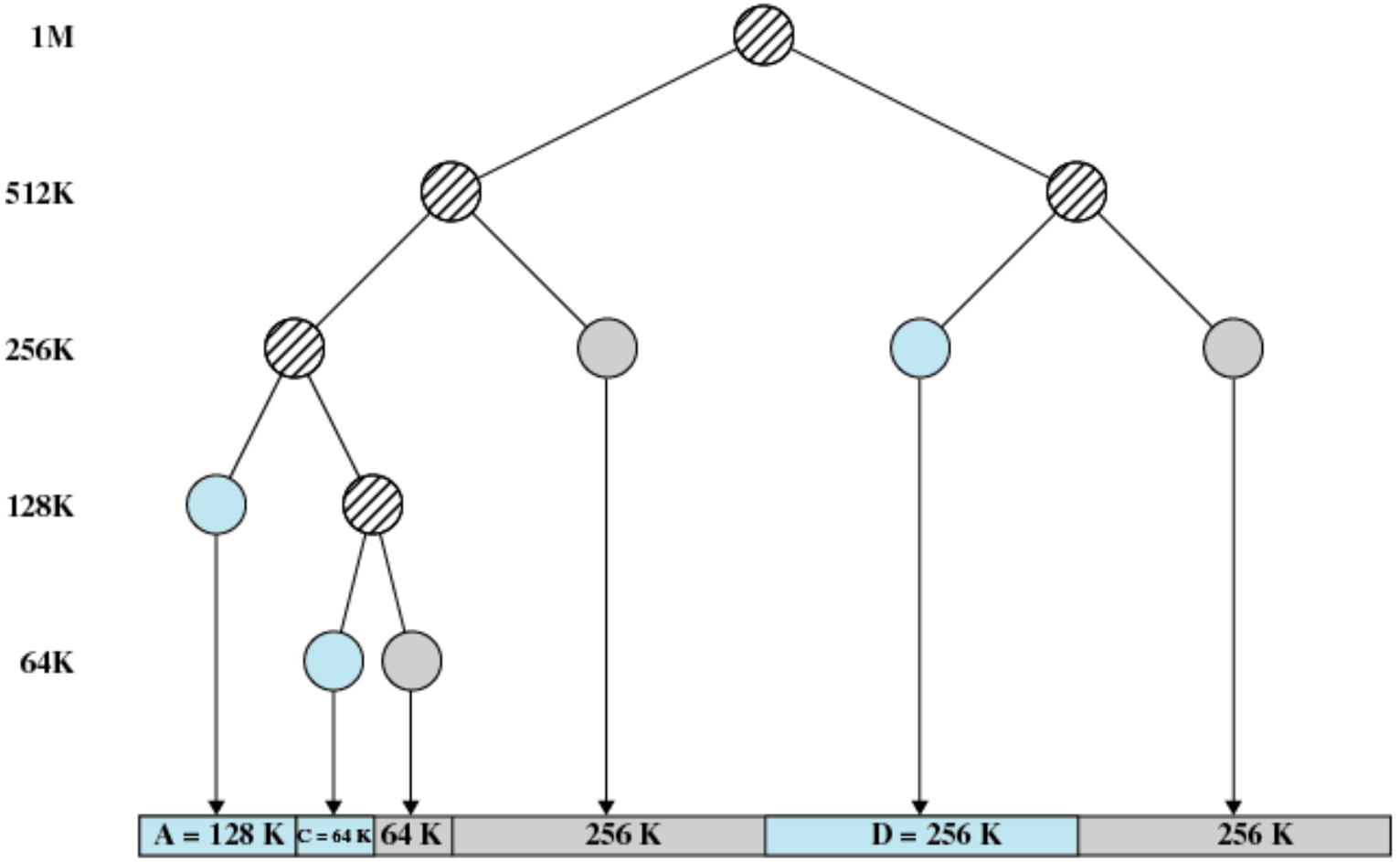
➢ Free blocks can easily be merged into bigger blocks

➢ Compactification eased by regularly sized blocks

Buddy System: Example



Buddy System: Example



Fragmentation of main memory

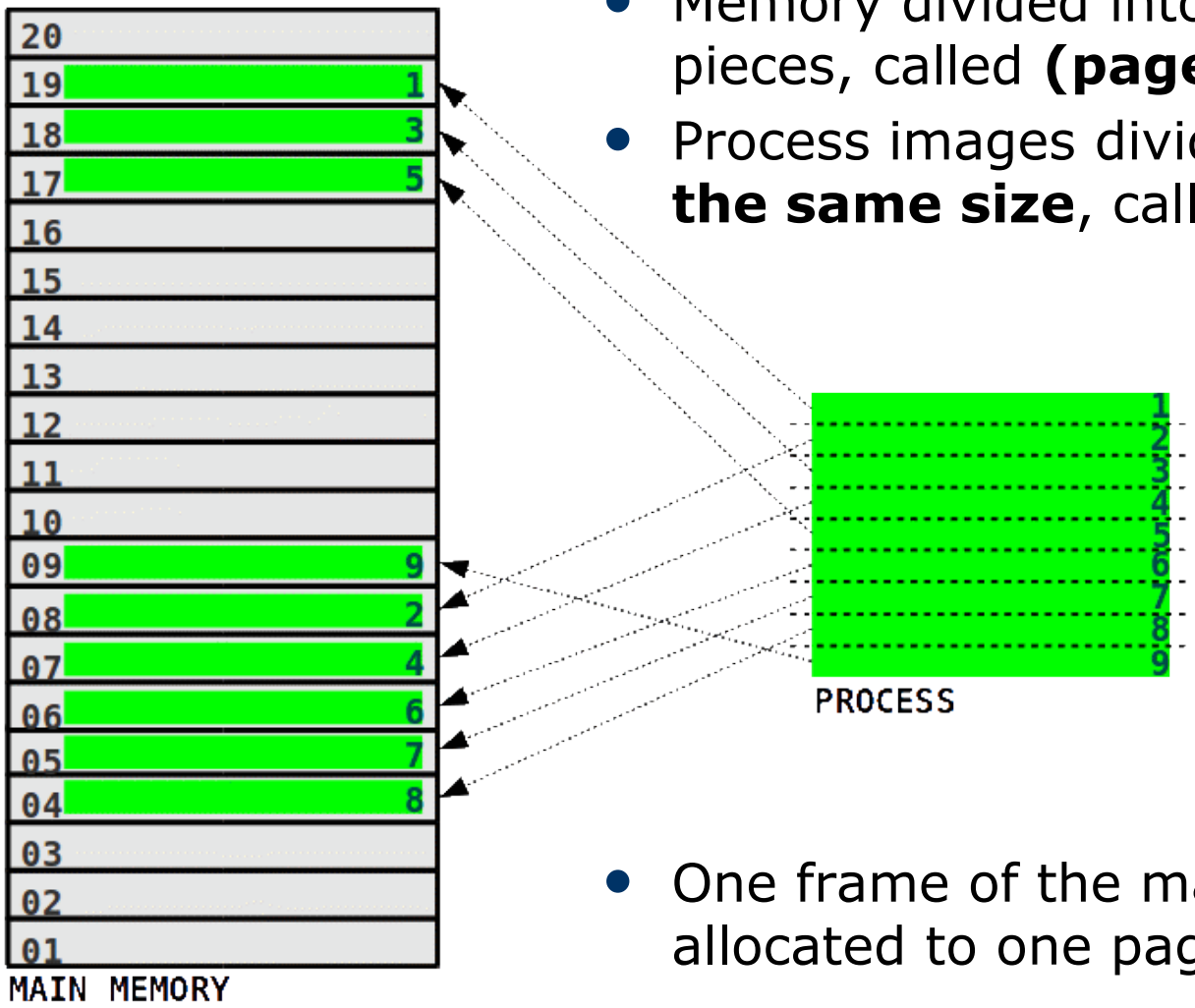
- Fragmentation: free cells in main memory are unusable because of the allocation scheme
  - memory space is wasted

- **Internal fragmentation**: the free memory cells are within the area allocated to a process
  - occurs using fixed partitions

- **External fragmentation**: the free memory cells are not in the area allocated to any process
  - occurs using dynamic partitions

Paging

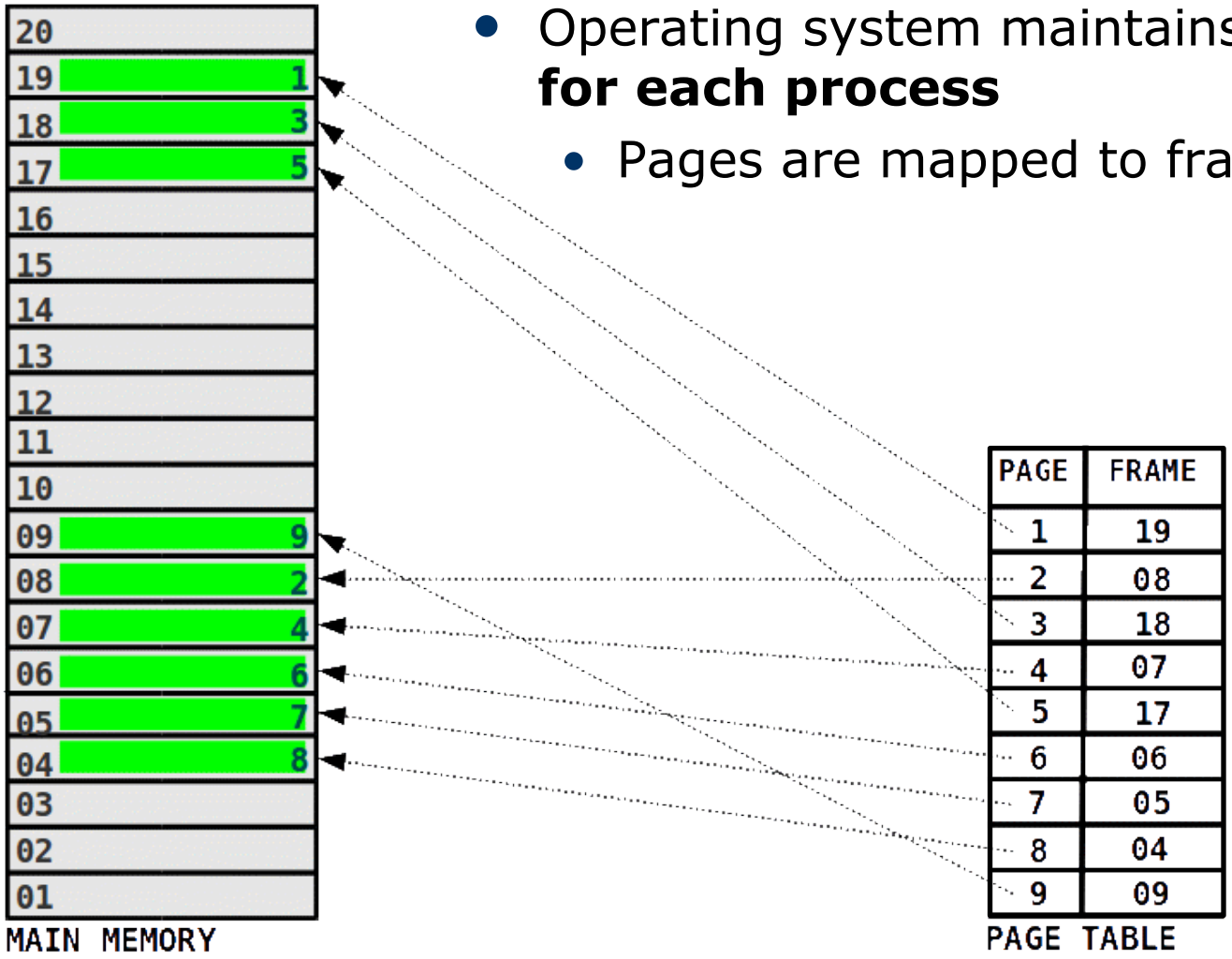
Paging



- Memory divided into small fixed-sized pieces, called **(page-)frames**
- Process images divided into pieces **of the same size**, called **pages**

- One frame of the main memory is allocated to one page of a process

Page Table



- Operating system maintains page table **for each process**
  - Pages are mapped to frames

Size of Frames/Pages

- Paging creates **no external fragmentation**
  - Since size of frames/pages is fixed

- Internal fragmentation depends on frame size
  - The smaller the frames the lower the internal fragmentation
  - BUT: the smaller the frames the bigger the page tables

Assignment of Pages to Frames

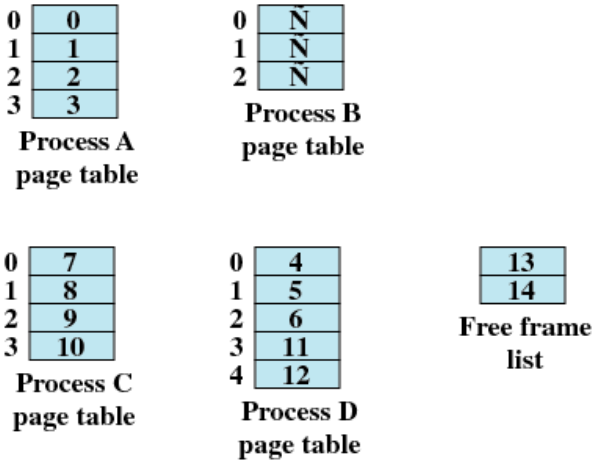
- Example:

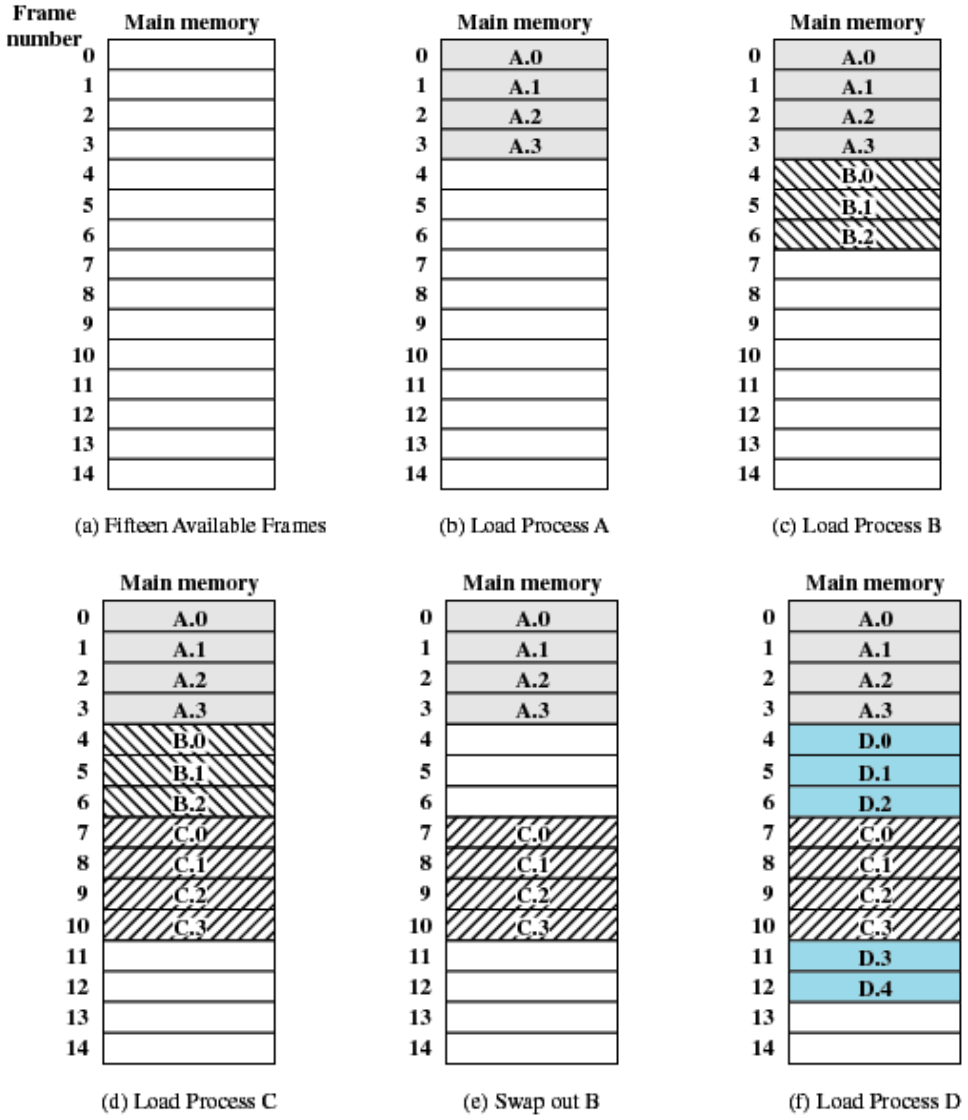
  (a) – (d) Load processes A, B, and C

  (e) Swap out process B

  (f) Load process D

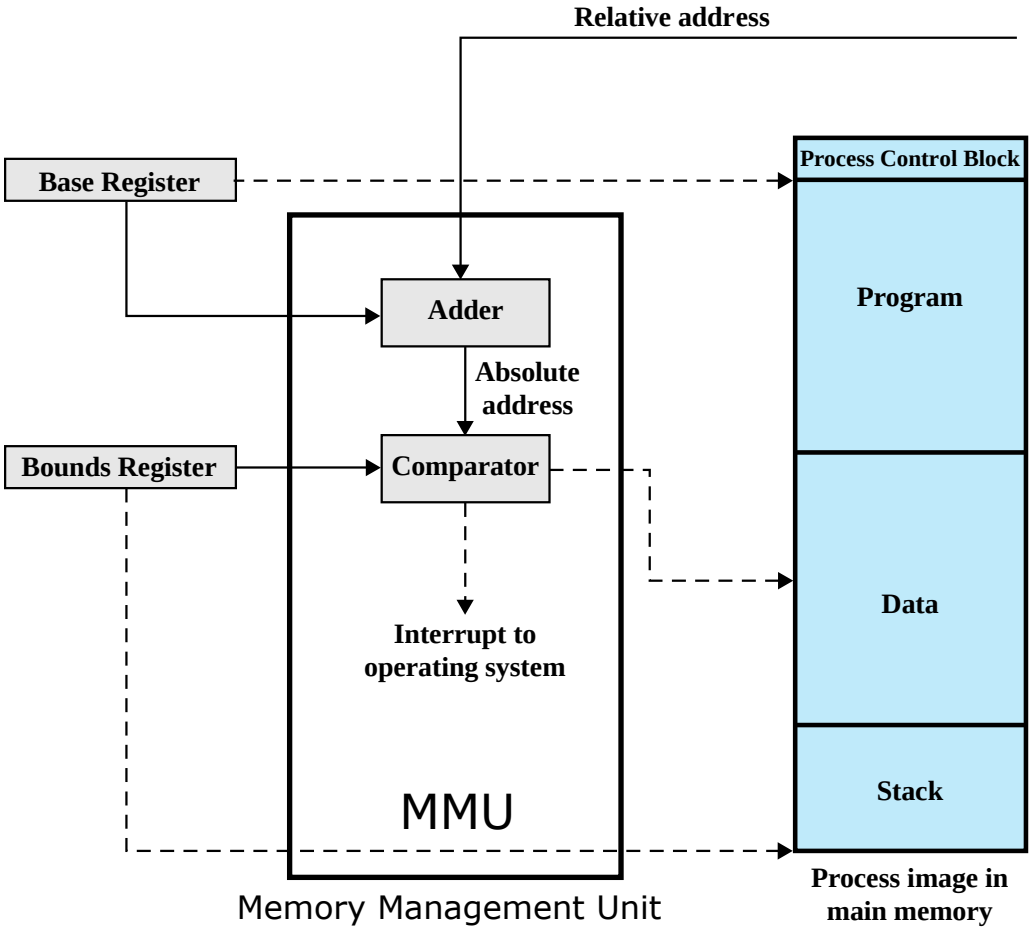
- Page Tables





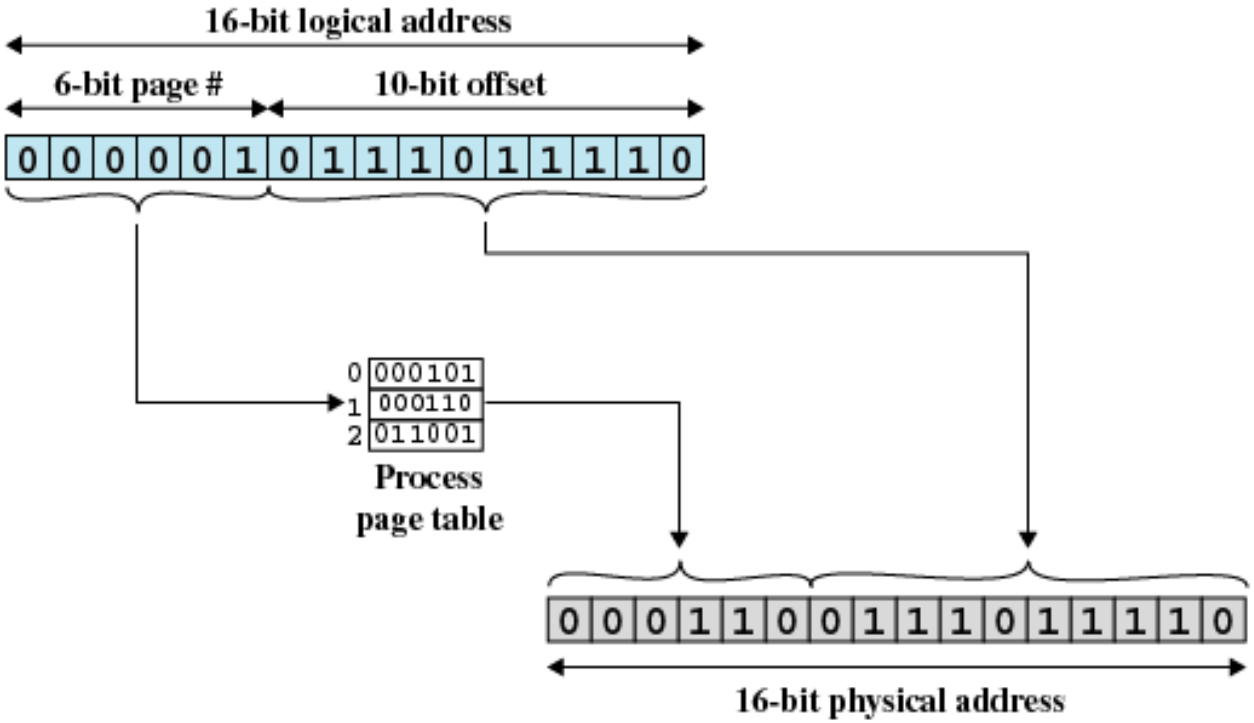
Hardware Support (MMU)

- Base register (starting address for the process)
- Bounds register (ending location of the process)
- Registers are set when process is loaded
- Bounds register is used for security purpose

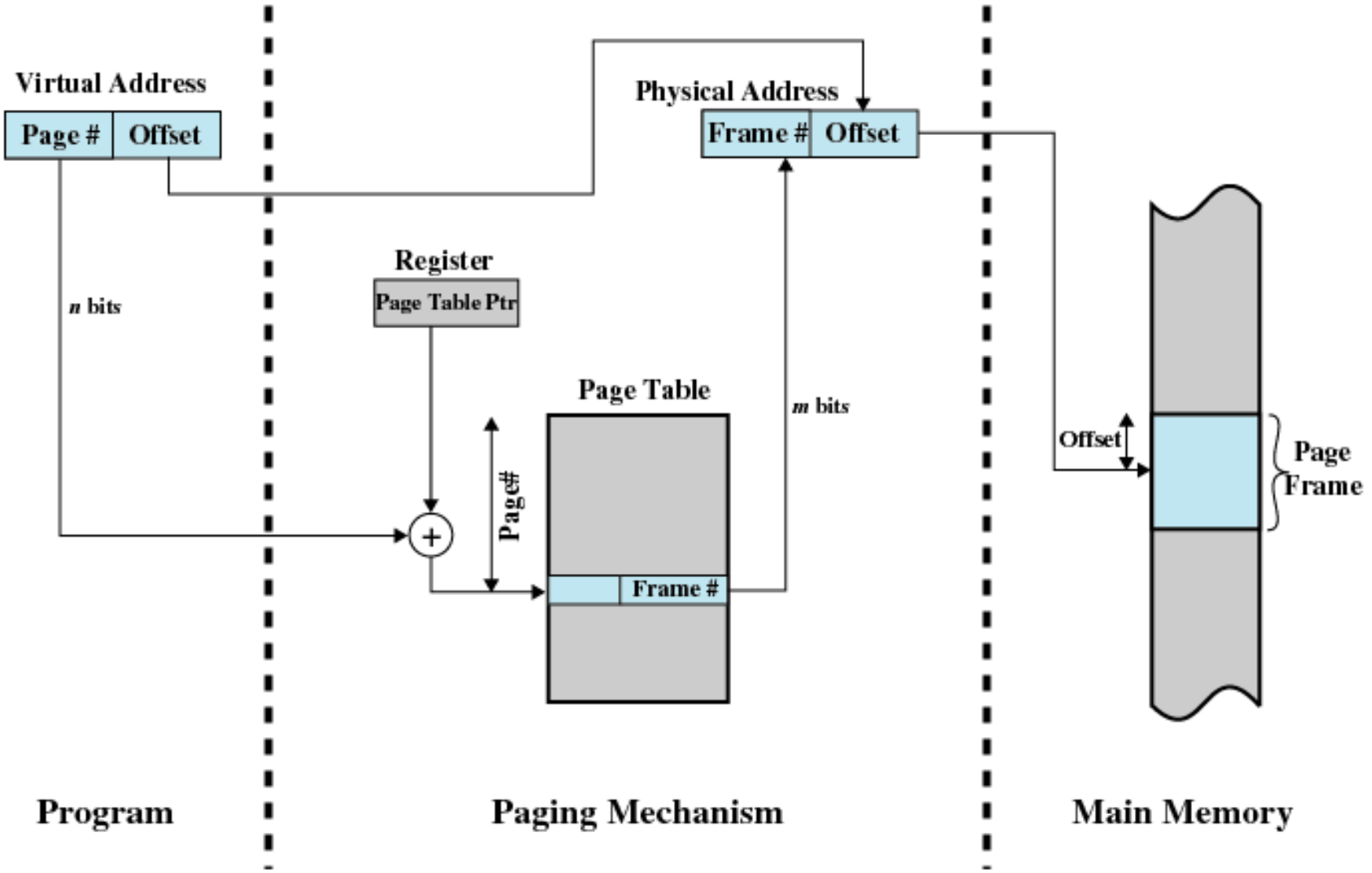


Addresses

- Memory address consists of a page number and offset within the page

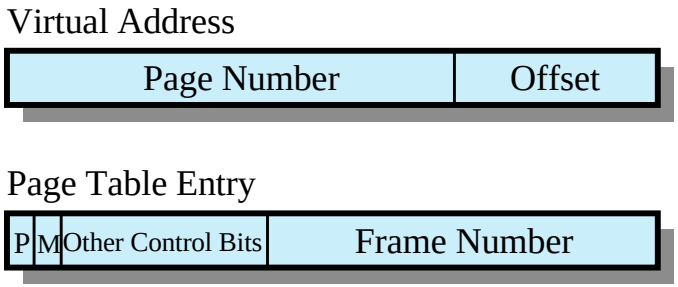


Paging Address Translation



Support Needed for Virtual Memory

- Hardware Support
  - Present bit: Page/segment is available in main memory
  - Modified bit: Content of page/segment has been modified
  - ➢ Implementation:
    - Paging:



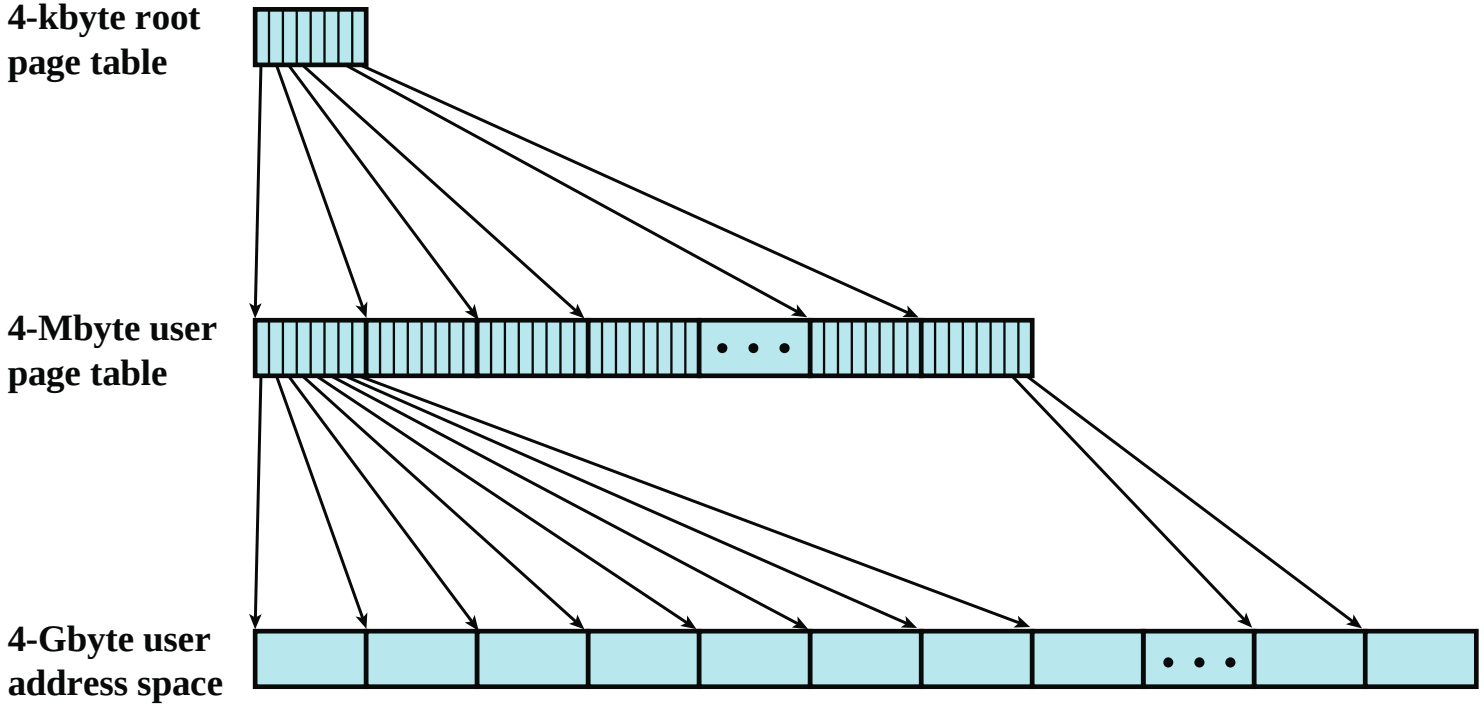
(a) Paging only

Other control bits:
- Write enabled
- Executable
- Shared between processes
- ...

- OS must be able to manage moving pages between primary and secondary memory

Hierarchical Page Table

- Page table *itself* may grow to considerable size



- Swap parts of page table to secondary storage
  - Problem: One virtual memory reference may cause two physical memory accesses (one to fetch page table, one to fetch data)
  - Performance penalty due to disk I/O delays

Hierarchical Page Table: Example

Translation Lookaside Buffer

- Problem: How to know which pages are loaded and up to date?

Translation Lookaside Buffer (TLB)

- Built into CPU
- Caches most recently used page table entries

Translation Lookaside Buffer

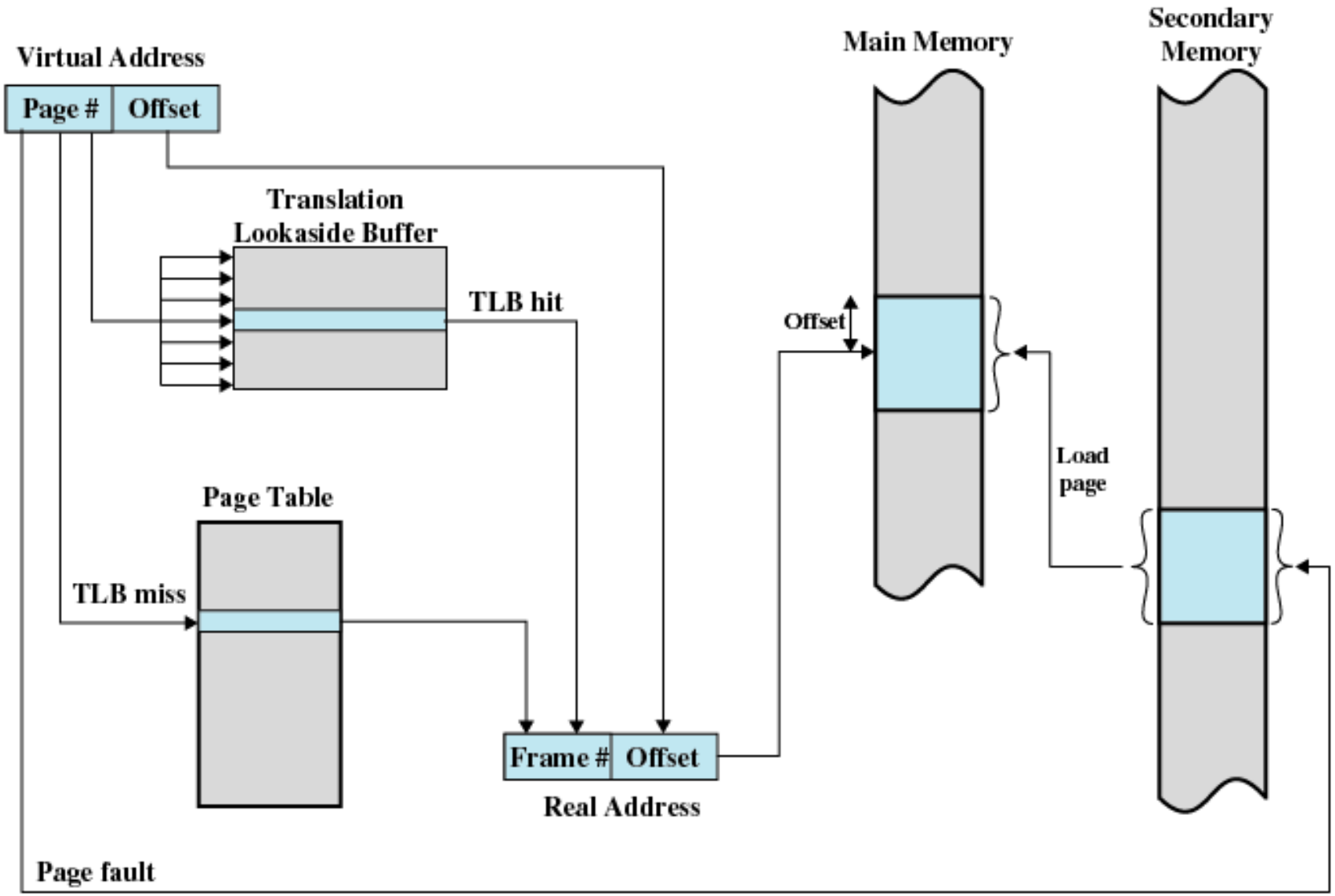
Basic steps:

1. Given a virtual address, processor examines TLB
   - If page table entry is present (TLB hit)
     - Retrieve frame number and form physical address
   - If page table entry is not found in TLB (TLB miss)
     - Fall back to process page table in main memory
       - For hierarchical page tables, possibly start recursion

2. OS checks if page is present in main memory
   - If not, issue page fault and fetch page from disk

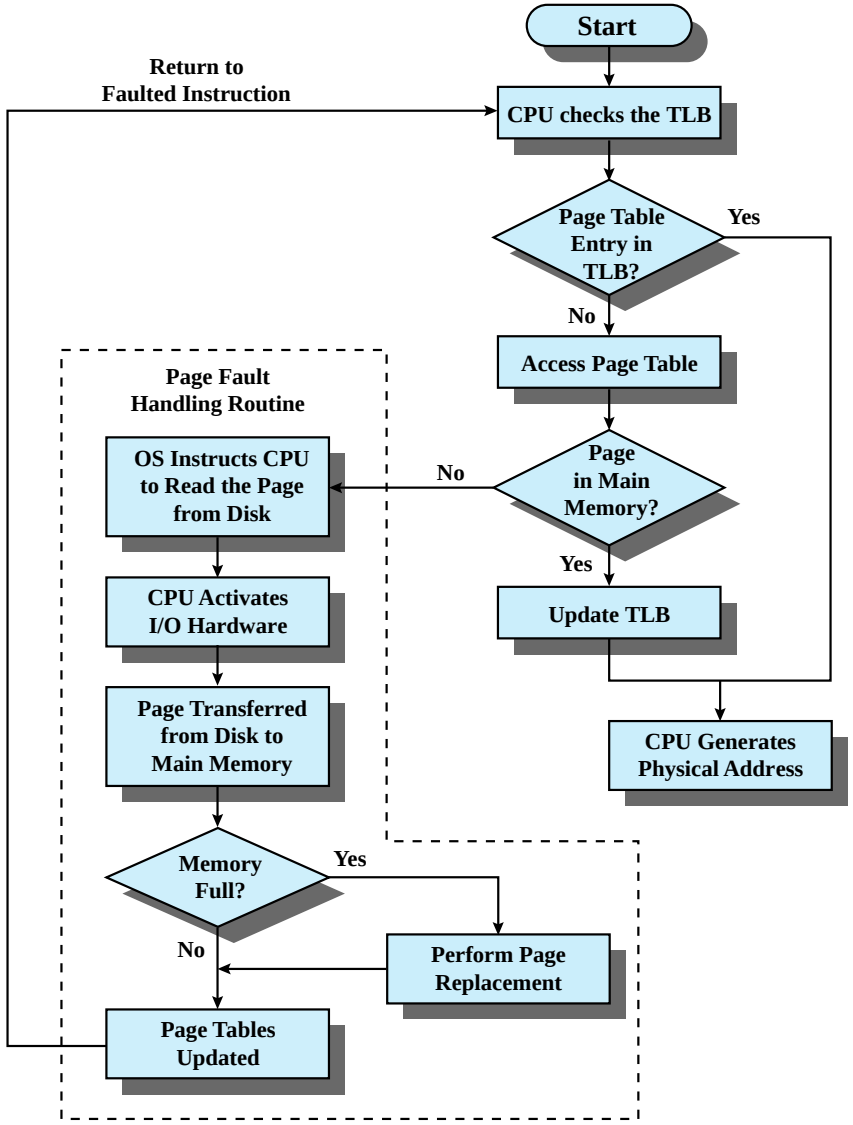
3. Update TLB to include new page entry

Translation Lookaside Buffer

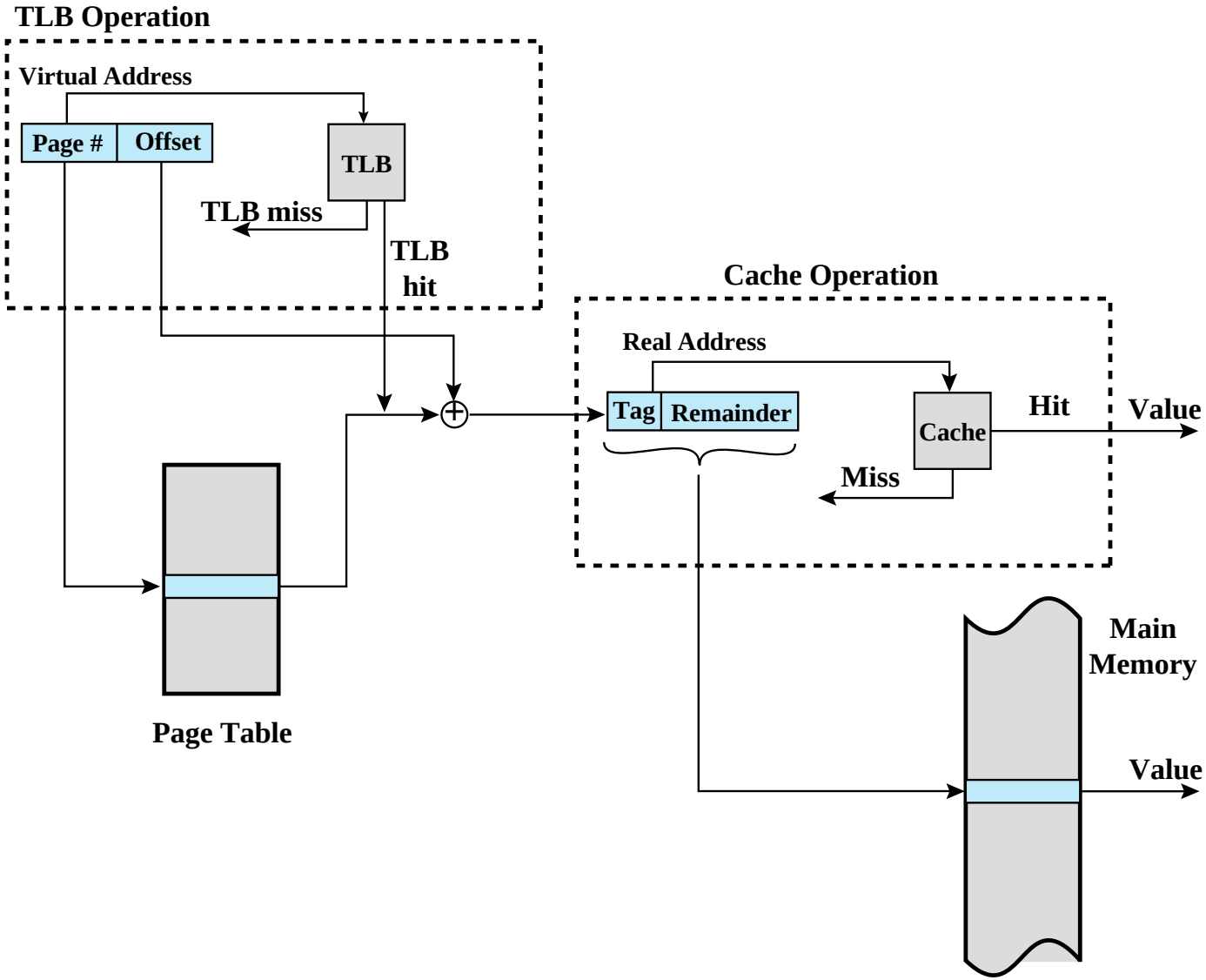


Translation Lookaside Buffer

Operation of Paging and Translation Lookaside Buffer



Translation Lookaside Buffer



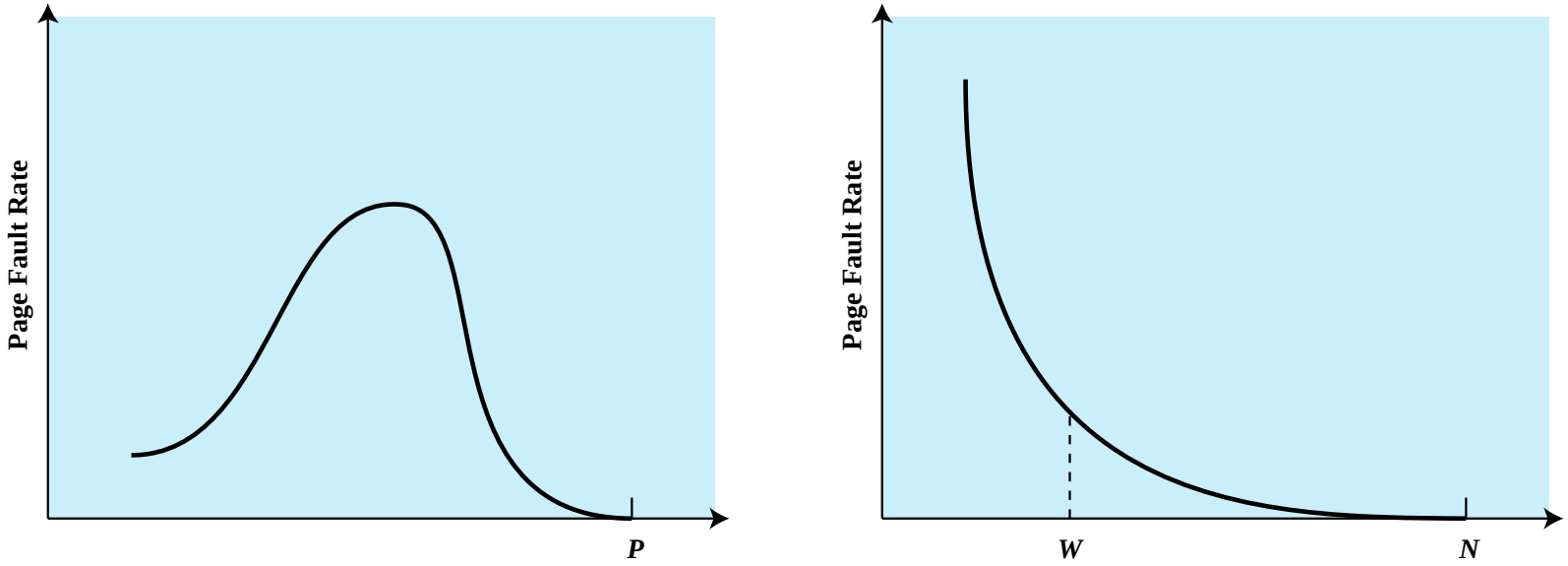
Paging

Page Size

Page Size

- Smaller page size …
  - less amount of internal fragmentation
  - more pages required per process
  - large number of pages will be found in main memory
- More pages per process means larger page tables
  - large portion of page tables in virtual memory
  - secondary memory is designed to efficiently transfer large blocks of data so a large page size is better
- With time pages in memory will contain portions of the process near recent references
  - Page faults low
- Increased page size causes pages to contain locations further from any recent reference
  - Page faults rise

Page Size



(a) Page Size

(b) Number of Page Frames Allocated

P **= size of entire process**
W **= working set size**
N **= total number of pages in process**

Figure 8.11 Typical Paging Behavior of a Program

Example Page Sizes

Table 8.2 Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbyes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Paging

Page Replacement

Problem: Thrashing

VM Thrashing:

- Page/segment of process is swapped out *just before* its needed
  - Happens under memory pressure, i.e., too many resource-hungry processes running on too little main memory
- Processor spends most of its time swapping pages/segments rather than executing user instructions
  - ➢ Computer stalls with heavy disk I/O

➢ Solution: "Good" page replacement policies

- Principle of Locality:
  - Program and data references within a process tend to cluster
  - Possible to make intelligent guesses about which pieces will be needed in the future

Algorithms / Policies

Fetch Policy

- Which page should be swapped in? When?

Alternatives

- Demand paging:
  only brings pages into main memory when reference is made to address on page
- Prepaging:
  brings in more pages than needed
  - Anticipates future requests

Replacement Policy

- Which page should be swapped out / replaced?

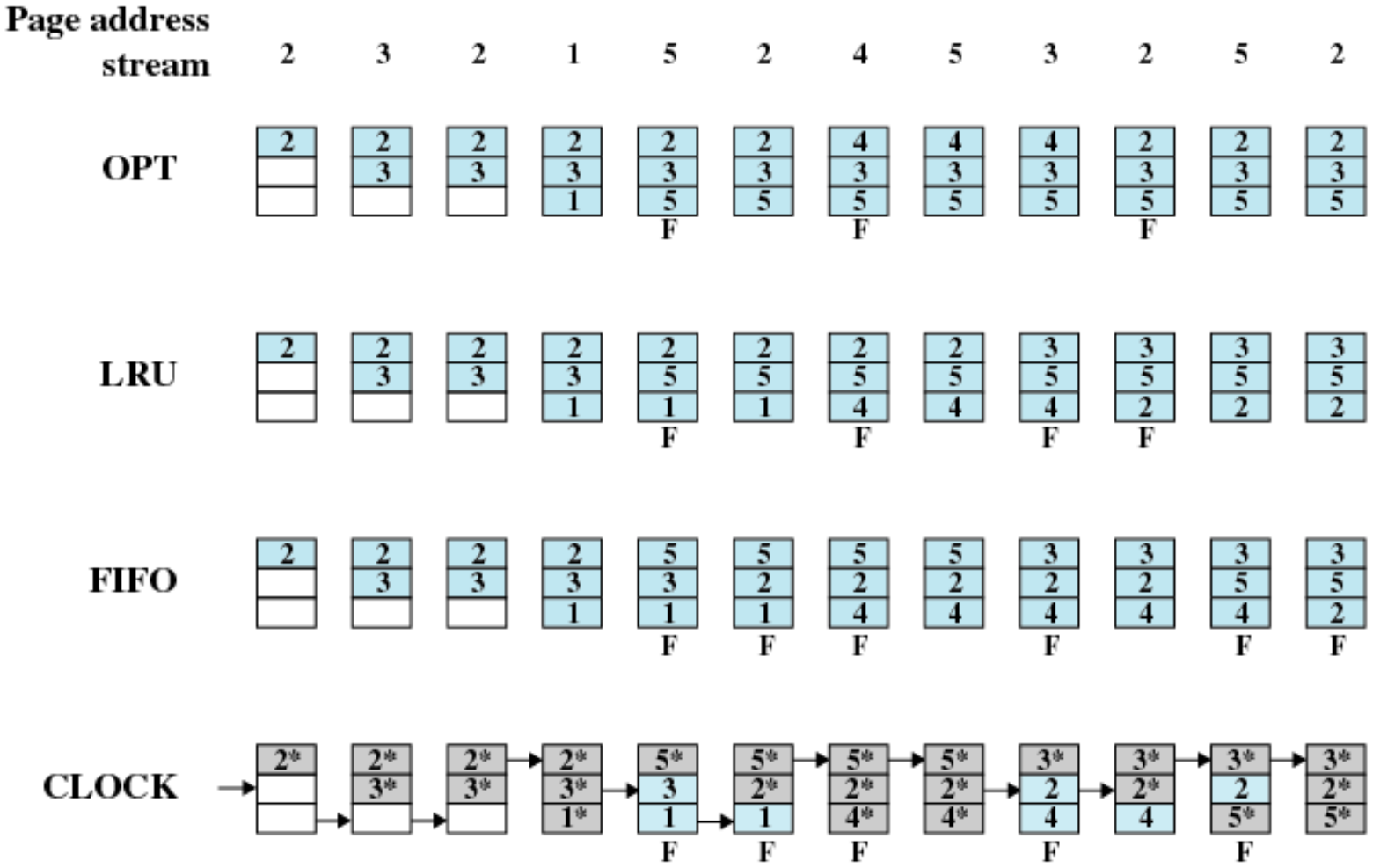
Approaches

- Remove page that is least likely to be referenced in near future
- Most policies predict future behavior on basis of past behavior, e.g.
  - First-In, First Out (FIFO)
  - Not Recently Used (NRU)
  - Least Recently Used (LRU)
  - ...

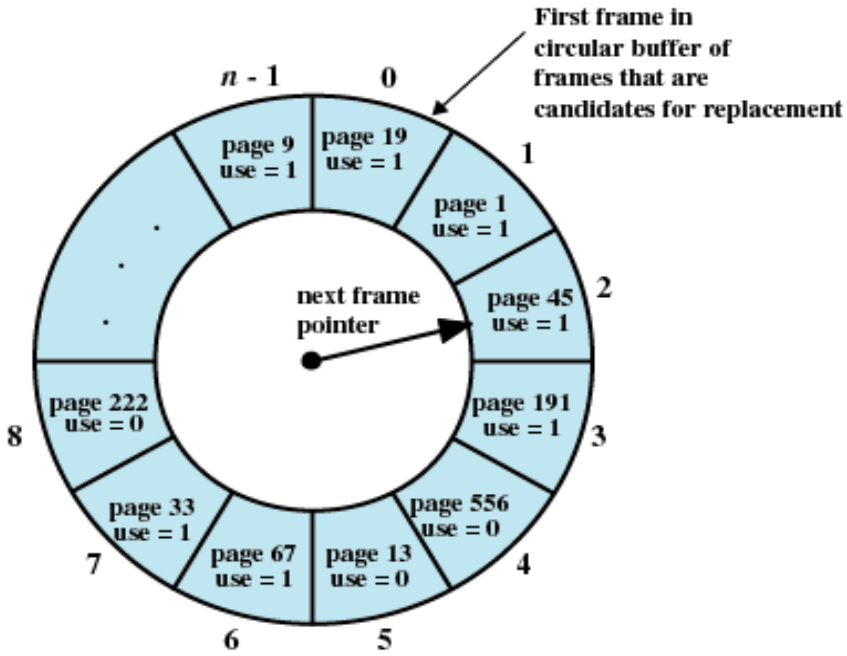
Some Basic Replacement Algorithms

- Optimal policy (for reference *only*)
  - Selects page for which time to next reference is longest
  - ➢ *Impossible* to have perfect knowledge of future events
- Least Recently Used (LRU)
  - Replaces page that has not been referenced for longest time
  - By principle of locality, least likely to be referenced in near future
- First-in, First-out (FIFO)
  - Treats page frames allocated to a process as circular buffer
  - Pages are removed in round-robin style
  - Page that has been in memory the longest is replaced (but may be needed soon)
- Clock Policy
  - When a page is first loaded in memory, *use bit* is set to 1
  - When page is referenced, use bit is set to 1
  - During search for replacement, each use bit is changed to 0
  - When replacing pages, first frame with use bit set to 0 is replaced

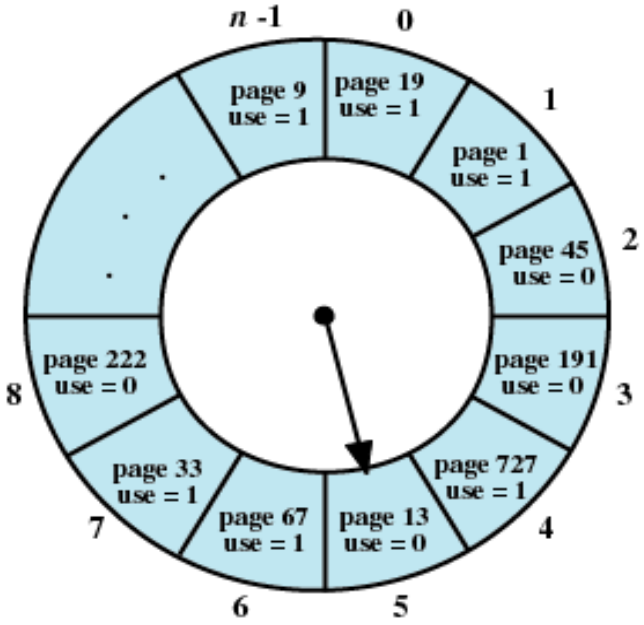
Page Replacement Example



Page Replacement Example

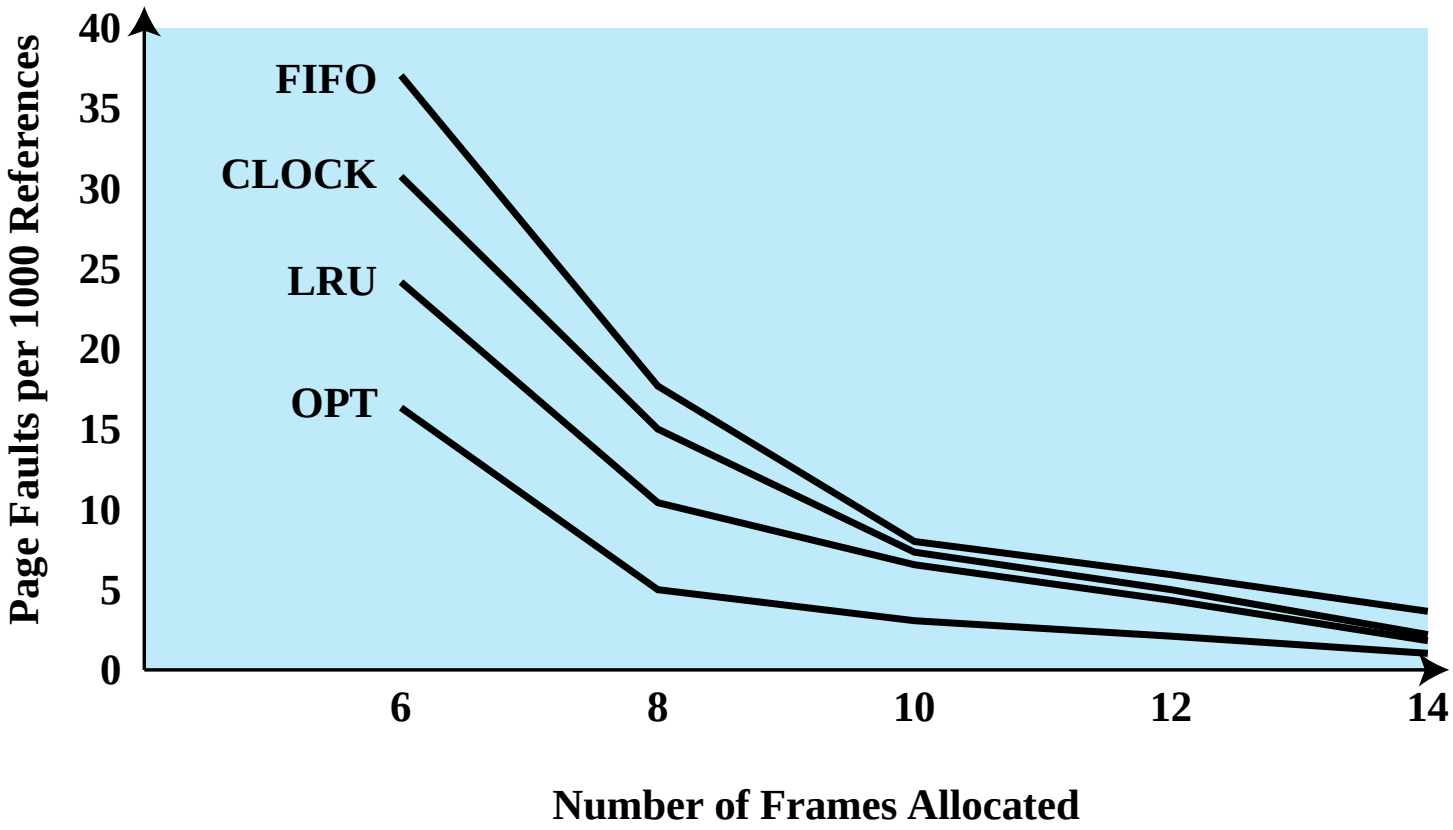


(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Comparison of Placement Algorithms

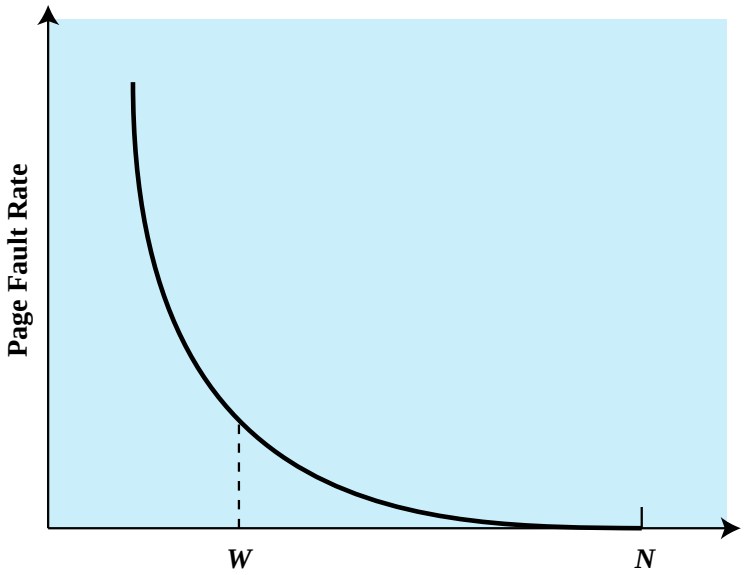


Paging

Resident Size

Resident Set Size

- Fixed-allocation
  - Gives a process a fixed number of pages
  - When a page fault occurs, one of the pages of that process must be replaced
- Variable-allocation
  - Number of pages varies over the lifetime of the process



(b) Number of Page Frames Allocated

Resident Set Size

- Decide ahead of time the amount of allocation to give a process
- If allocation is too small, there will be a high page fault rate
- If allocation is too large there will be too few programs in main memory

Working Set

- Working Set of a process

$W(t, \Delta)$ = set of pages of the process that have been referenced in the last Δ time units

- Property

$$W(t, \Delta+1) \supseteq W(t, \Delta)$$

Resident Set Size

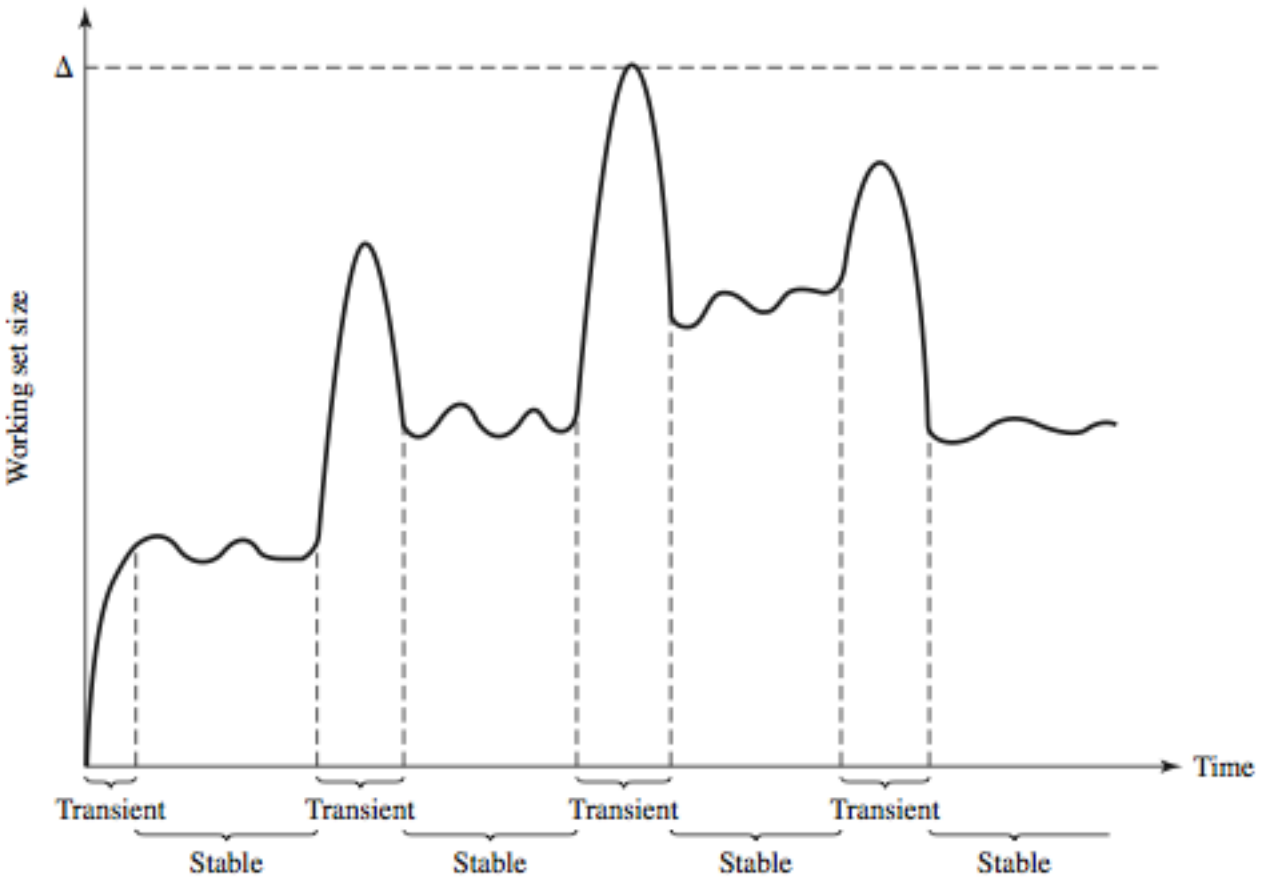


Figure 8.20 **Typical Graph of Working Set Size [MAEK87]**

Working Set

Page References	Window Size, D			
	2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	18 24	•	24 17 18
17	18 17	24 18 17	•	•
17	17	18 17	•	•
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	•
17	24 17	•	•	17 15 24
24	•	24 17	•	•
18	24 18	17 24 18	17 24 18	15 17 24 18

Figure 8.19 Working Set of Process as Defined by Window Size

Load Control

- Determines the number of processes that will be resident in main memory
- Too few processes, many occasions when all processes will be blocked and much time will be spent in swapping
- Too many processes will lead to thrashing

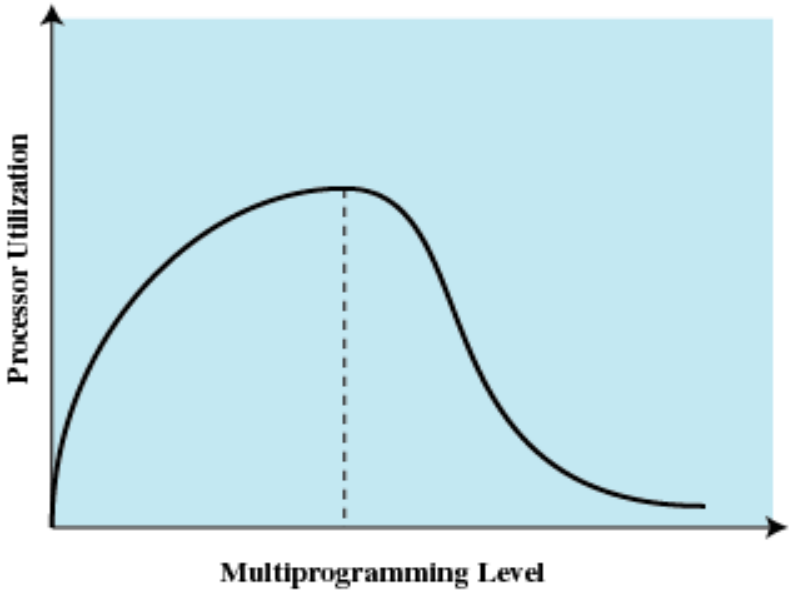


Figure 8.21 Multiprogramming Effects

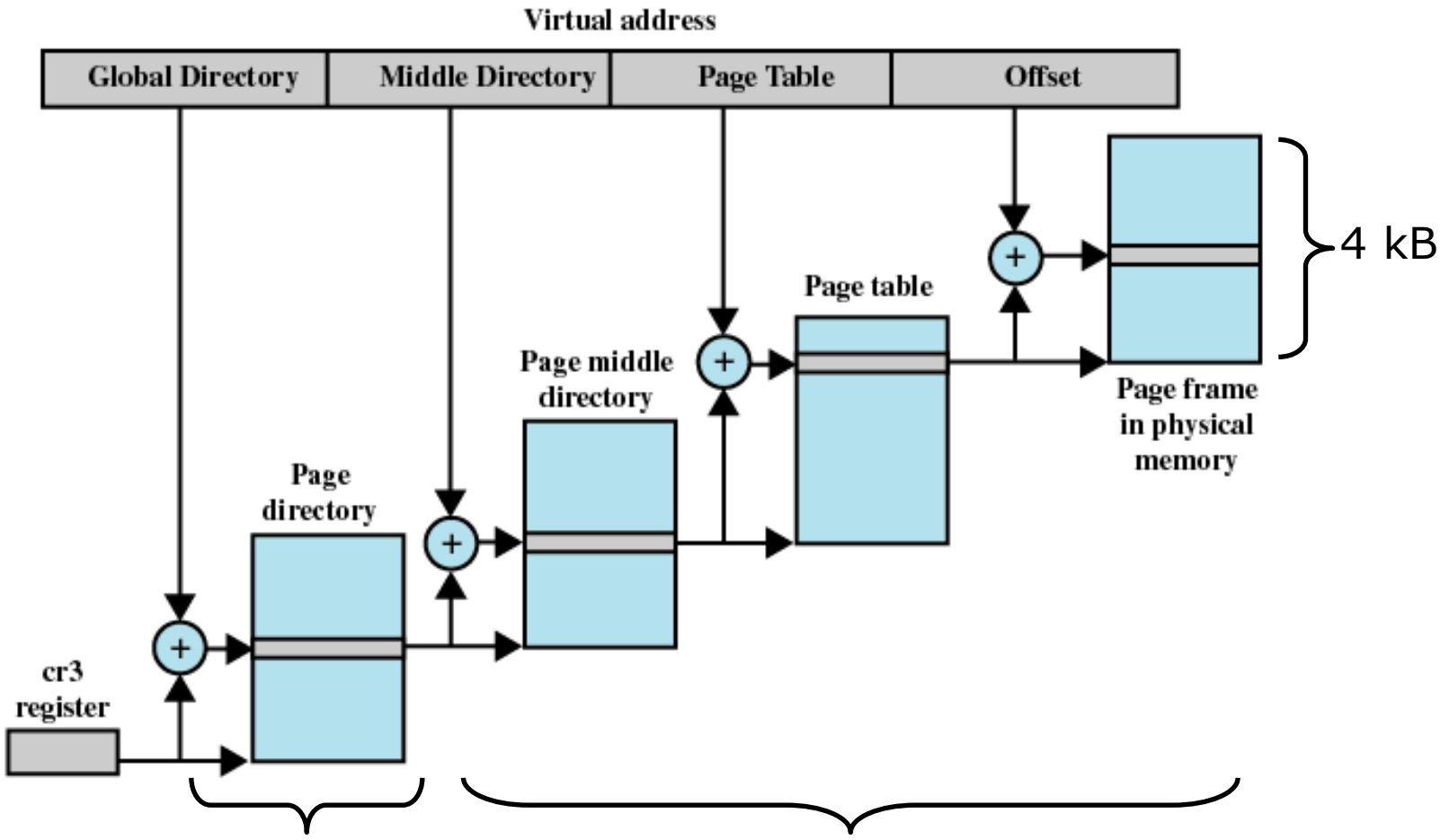
Segmentation

Segmentation

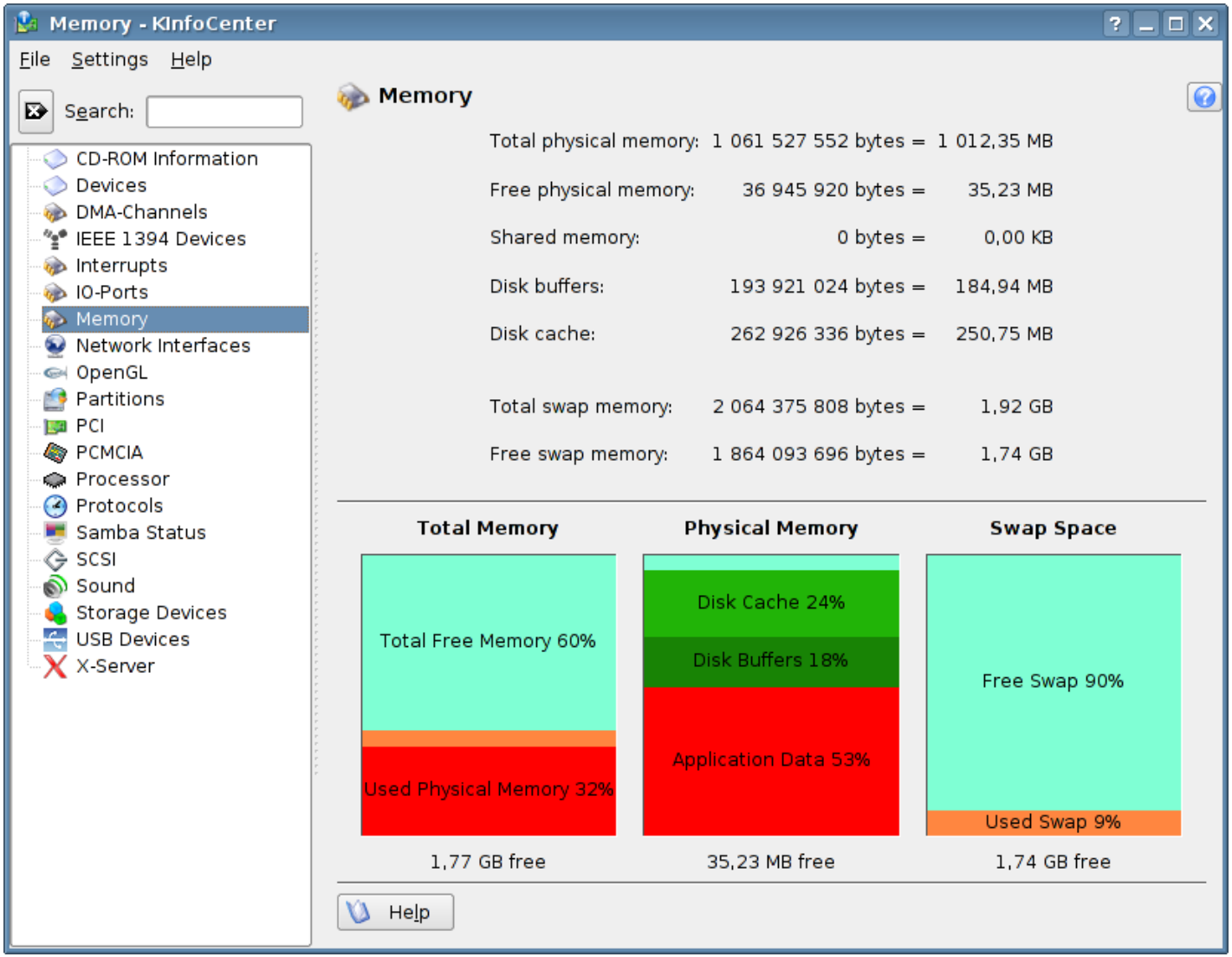
- All segments of all programs do not have to be of the same length
- There is a maximum segment length
- Addressing consist of two parts - a segment number and an offset
- Since segments are not equal, segmentation is similar to dynamic partitioning

Example Linux

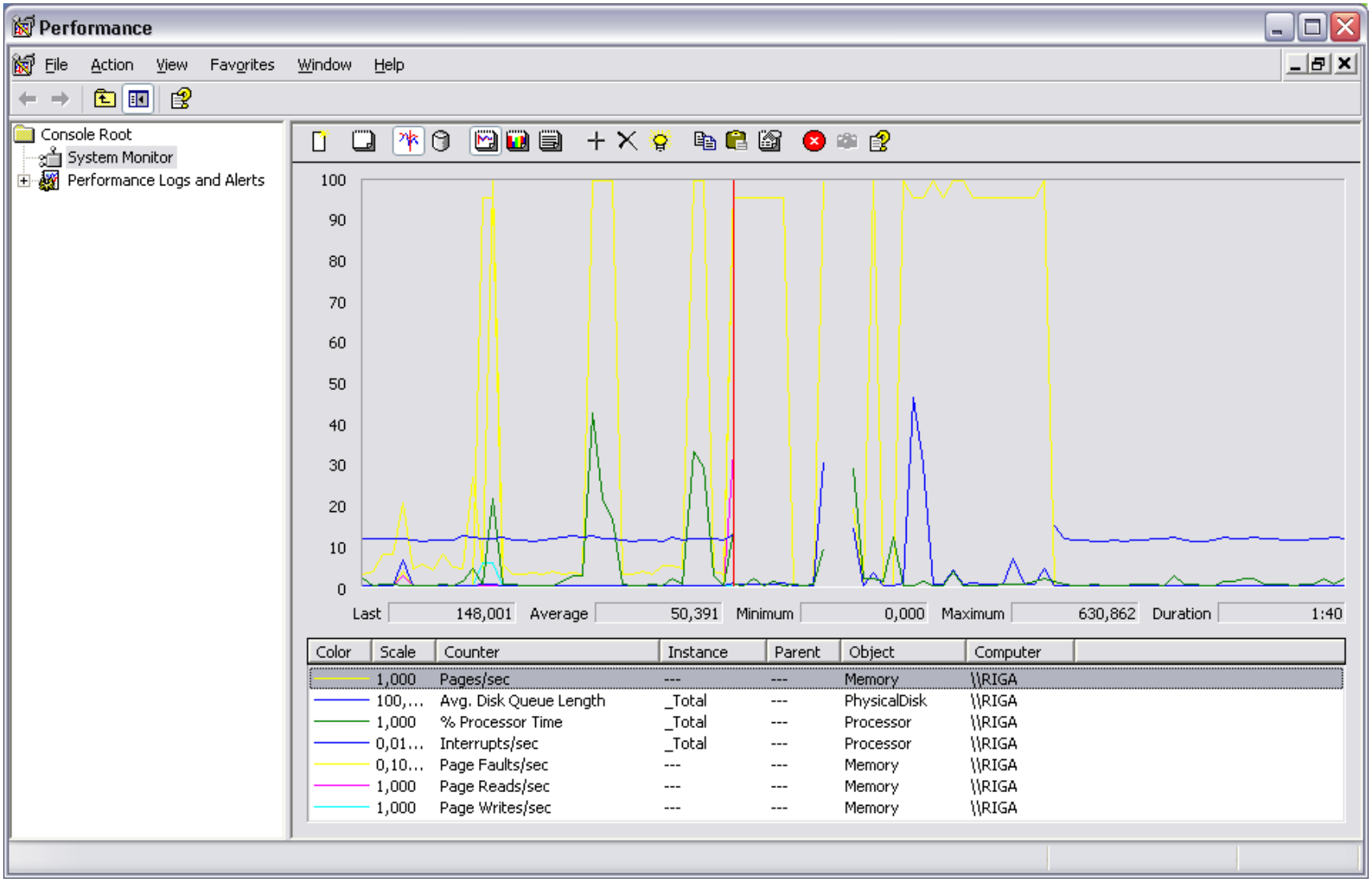
Example: Linux VM Implementation



Example: Linux Memory Utilization



Example: Windows Paging (perfmon)



System Calls Related to Memory Management

Related System Calls (Linux)

- **`int brk(void *end_data_segment)`**
  - Sets end of data segment of process to **`end_data_segment`**
- **`void *sbrk(intptr_t increment)`**
  - Increments the program's data space by **`increment`** bytes
- **`void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)`**
  - Maps **`length`** bytes of file descriptor **`fd`** to address **`start`**
  - With flag **`MAP_ANONYMOUS`** no actual file is needed
- **`int munmap(void *start, size_t length)`**
  - Deletes mapping to specified address

Related Library Wrappers

- **`void *malloc(size_t size)`**
  - Allocates **`size`** bytes and returns pointer
  - Returns NULL if no memory is available
- **`void free(void *ptr)`**
  - Frees memory pointed to by **`ptr`**
- **`void *calloc(size_t nmemb, size_t size)`**
  - Allocates and zeroes memory for **`nmemb`** elements of size **`size`** bytes
- **`void *realloc(void *ptr, size_t size)`**
  - Changes size of previously allocated memory at **`ptr`** to **`size`** bytes