FREIE UNIVERSITÄT BERLIN

Operating Systems & Computer Networks
5. Memory

Dr. Larissa Groth
Computer Systems & Telematics (CST)

Roadmap

1. Introduction and Motivation
2. Interrupts and System Calls
3. Processes
4. Scheduling
5. **Memory**
6. I/O and File System
7. Booting, Services, and Security

Lernziele I

- Sie nennen:
  - die wesentlichen Aufgaben eines general-purpose BS hinsichtlich Speicher

- Sie beschreiben:
  - den Zusammenhang zwischen physischen Adressen, virtuellen Adressen und relativen Adressen
  - den Unterschied zwischen Fixer und Dynamischer Partitionierung
  - den Zusammenhang zwischen Partitionierung und Fragmentierung
  - warum es beim Paging mit großer Wahrscheinlichkeit zu interner Fragmentierung in der letzten Page eines Prozesses kommt
  - die Replacement Verfahren FIFO, LIFO, LRU, LFU, LRD für fixe Partitionierung
  - warum Page Tables vom BS verwaltet werden und wie sie die Übersetzung aus virtuellen in physische Adressen unterstützen
  - wie das Problem gelöst werden kann, dass eine Page Table selbst nicht in eine Page passt
  - die Rolle des Translation Lookaside Buffer bei der Adressübersetzung

Lernziele II

- Sie wenden an:
  - die Dynamic Placement Verfahren First-Fit, Best Fit, Next Fit, Worst Fit für dynamische Partitionierung

- Sie untersuchen:
  - die Rolle der Page Size in Hinblick auf den Grad der internen Fragmentierung und auf die Größe der resultierenden Page Tables

- Sie bewerten:
  - die Replacement Verfahren FIFO, LIFO, LRU, LFU, LRD in Hinblick auf zeitliche und örtliche Lokalität

Motivation

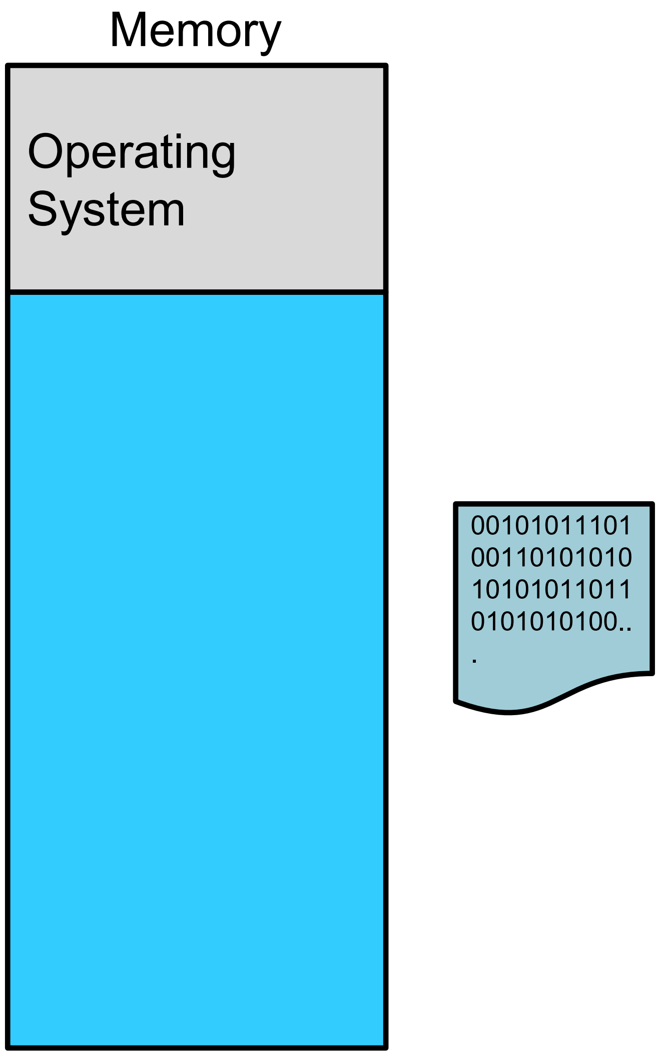
To which location in memory should the process image be loaded?

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

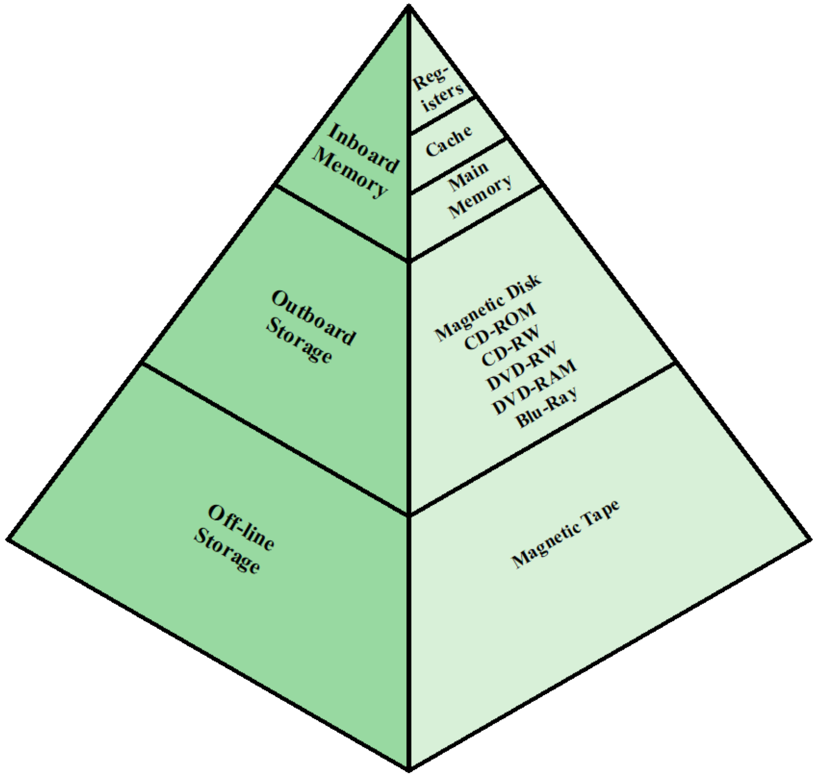


Figure 1.14 The Memory Hierarchy

See course Computer Architecture!

- Here: many pointers to this course

Memory Management

Closely related to processes

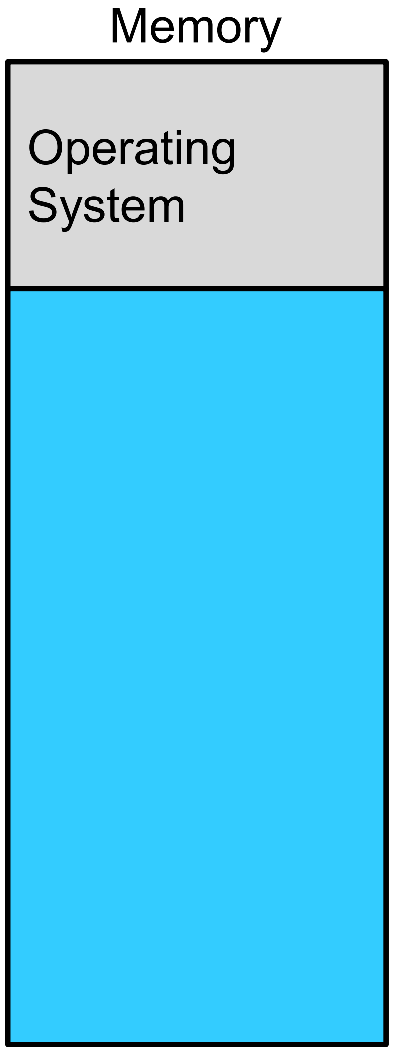
- Memory management isolates processes from each other

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 I

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)

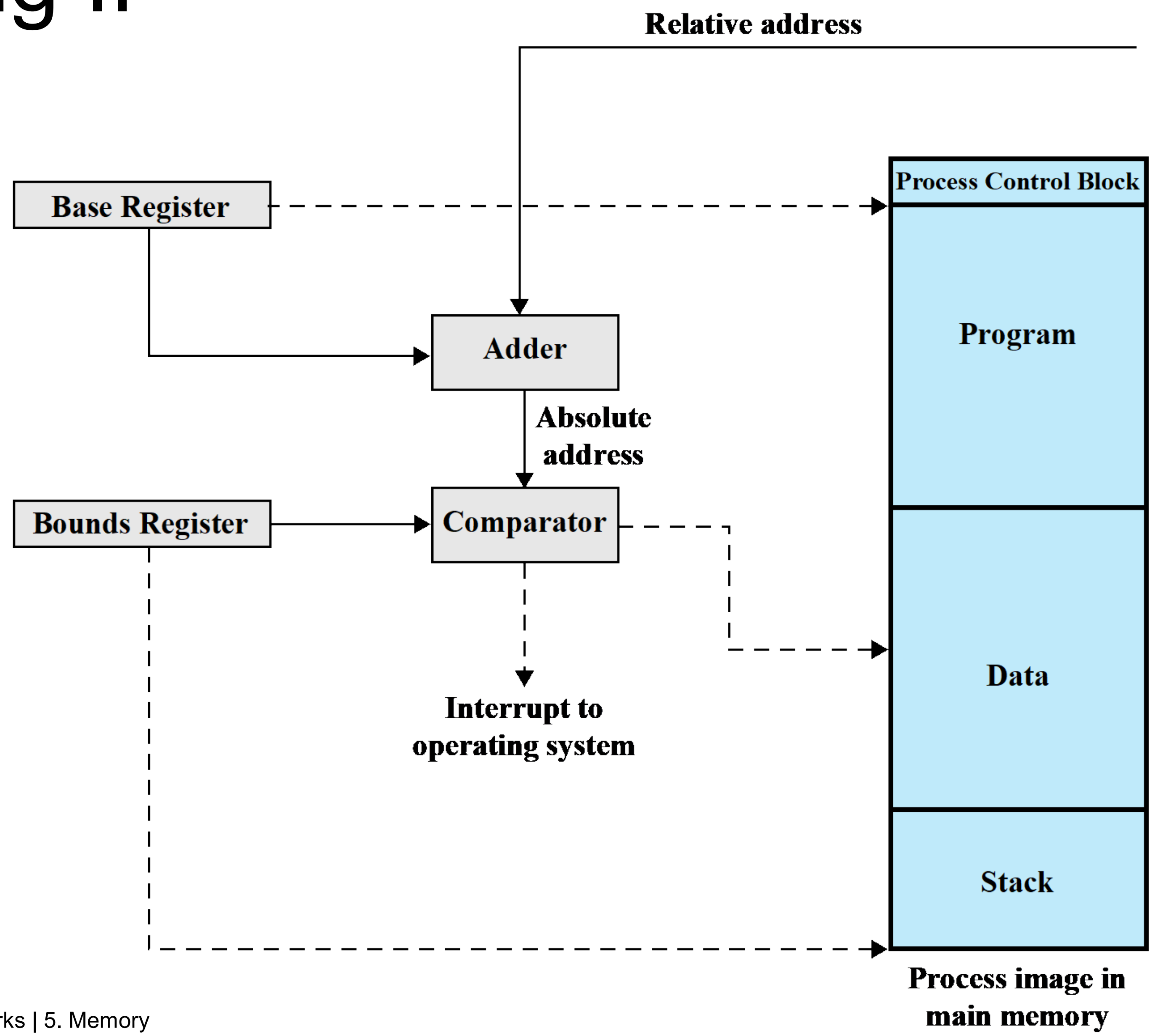
Logical/Virtual Address

- 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

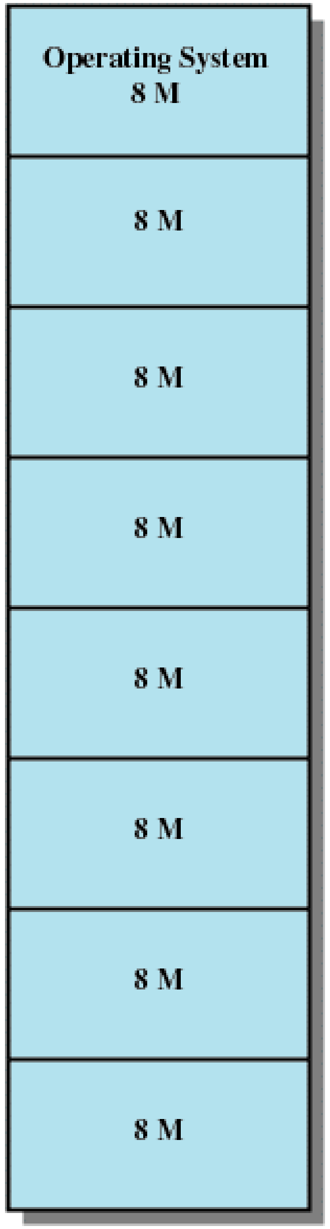
Addressing II



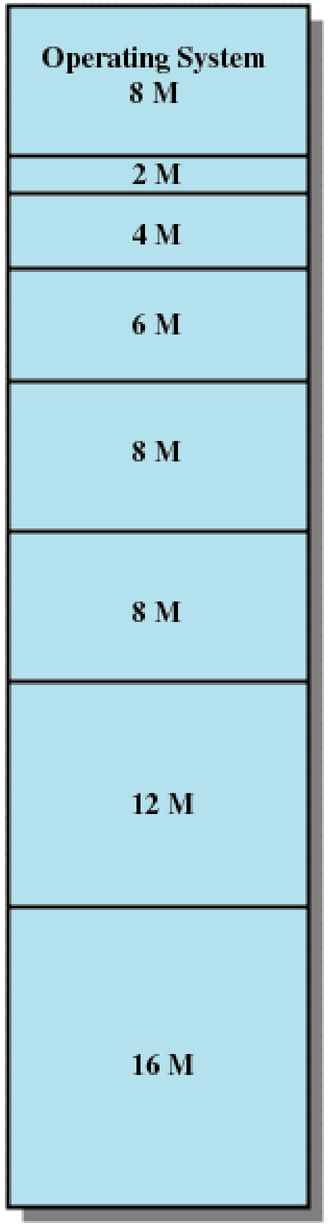
Fixed and Dynamic Partitioning

Partitioning

Fixed Partitioning
→ equal-size partitions



Dynamic Partitioning
→ unequal-size partitions

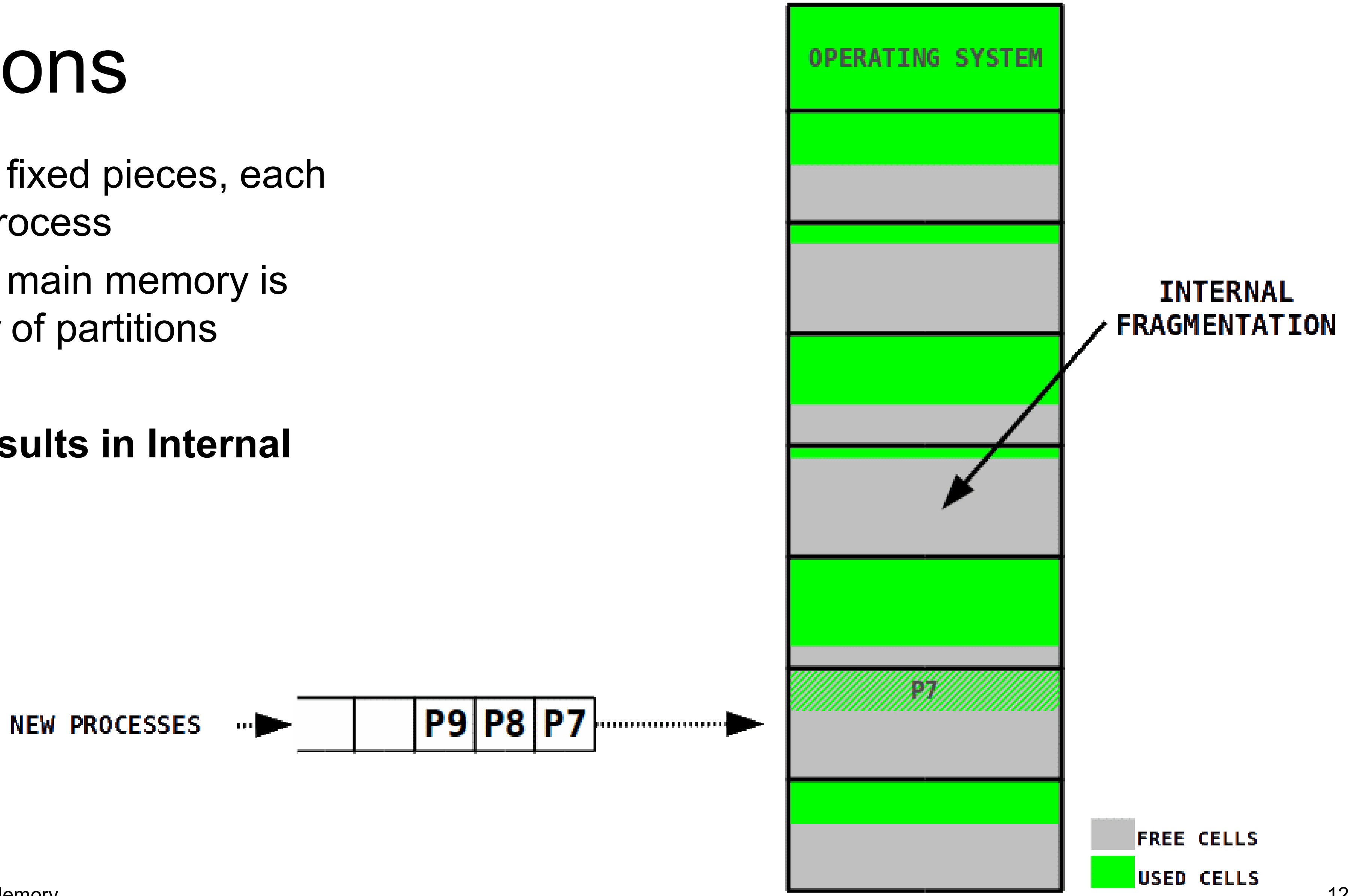


Fixed Partitions

Memory partitioned into fixed pieces, each partition can hold one process

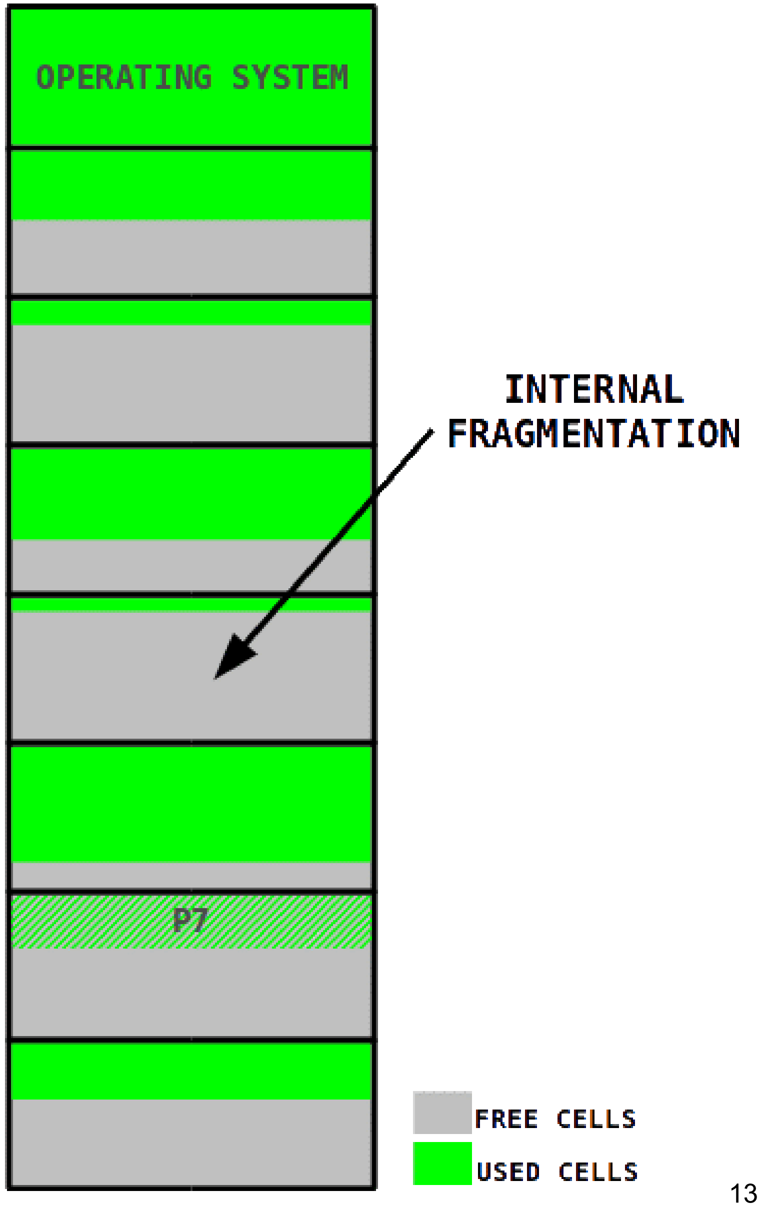
Amount of processes in main memory is bounded by the number of partitions

➢**Fixed Partitioning results in Internal fragmentation**



Internal Fragmentation

Fixed Partitioning results in **Internal Fragmentation**
→ free memory cells **within** partitions

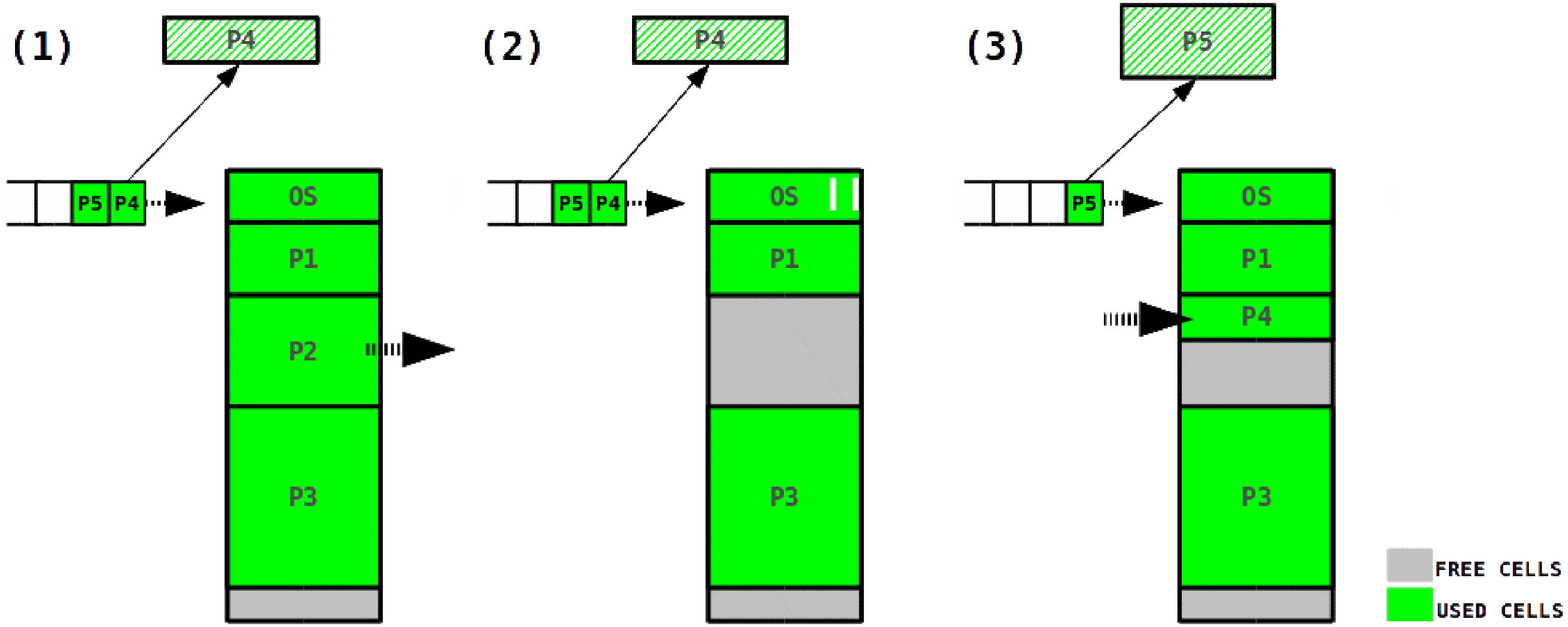


Dynamic Partitions

Memory is divided into variable sized partitions on demand

➢ **Dynamic Partitioning results in External Fragmentation**

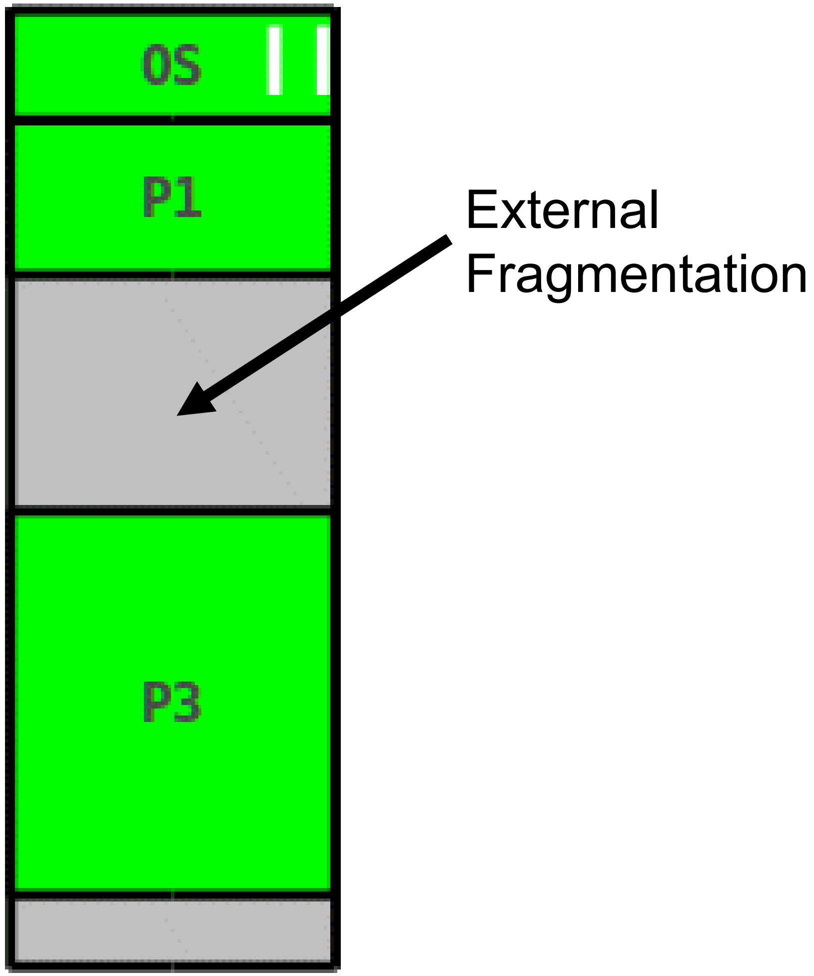
Example:



In this example: 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 results in **External Fragmentation**
→ free memory cells **outside** of partitions



Open Issues

Problems of virtual memory management

Problem areas when exchanging data between main memory and secondary memory

1. When to fetch data

- Best moment to load segments or pages into main memory

- Common approach:
  - Fetching on demand (*Demand Paging* for Paging)
    - Fetch data as soon as attempted access cannot be served from main memory
    - ➢ Called segment fault for segmentation or page fault for paging

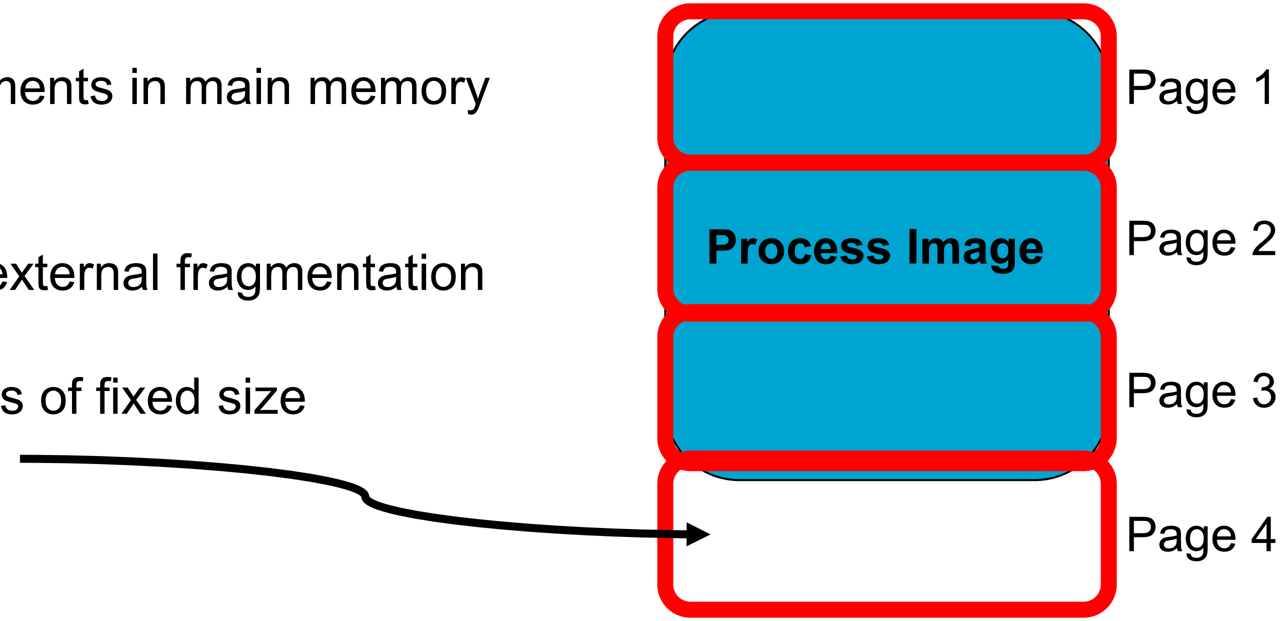
Problems of virtual memory management

2. Where to put data

- Find suitable location for pages or segments in main memory

- For paging
  - Every free partition is suitable → no external fragmentation
  - But internal fragmentation:
    - Process image is divided into pages of fixed size

    → unused free space in last page

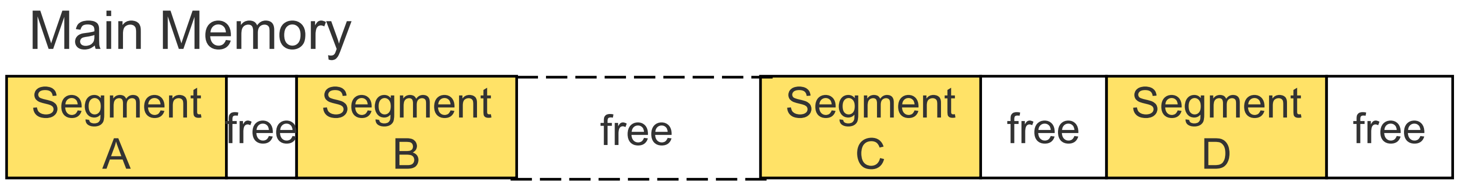


- For segmentation
  - Need to find free partition of appropriate size
  - Strategies (→ Dynamic Placement Algorithms)
    - first-fit: take first free partition that can hold new segment
    - best-fit: find smallest free partition that can hold new segment
    - worst-fit: find largest free partition that can hold new segment

External Fragmentation for Dynamic Partitioning

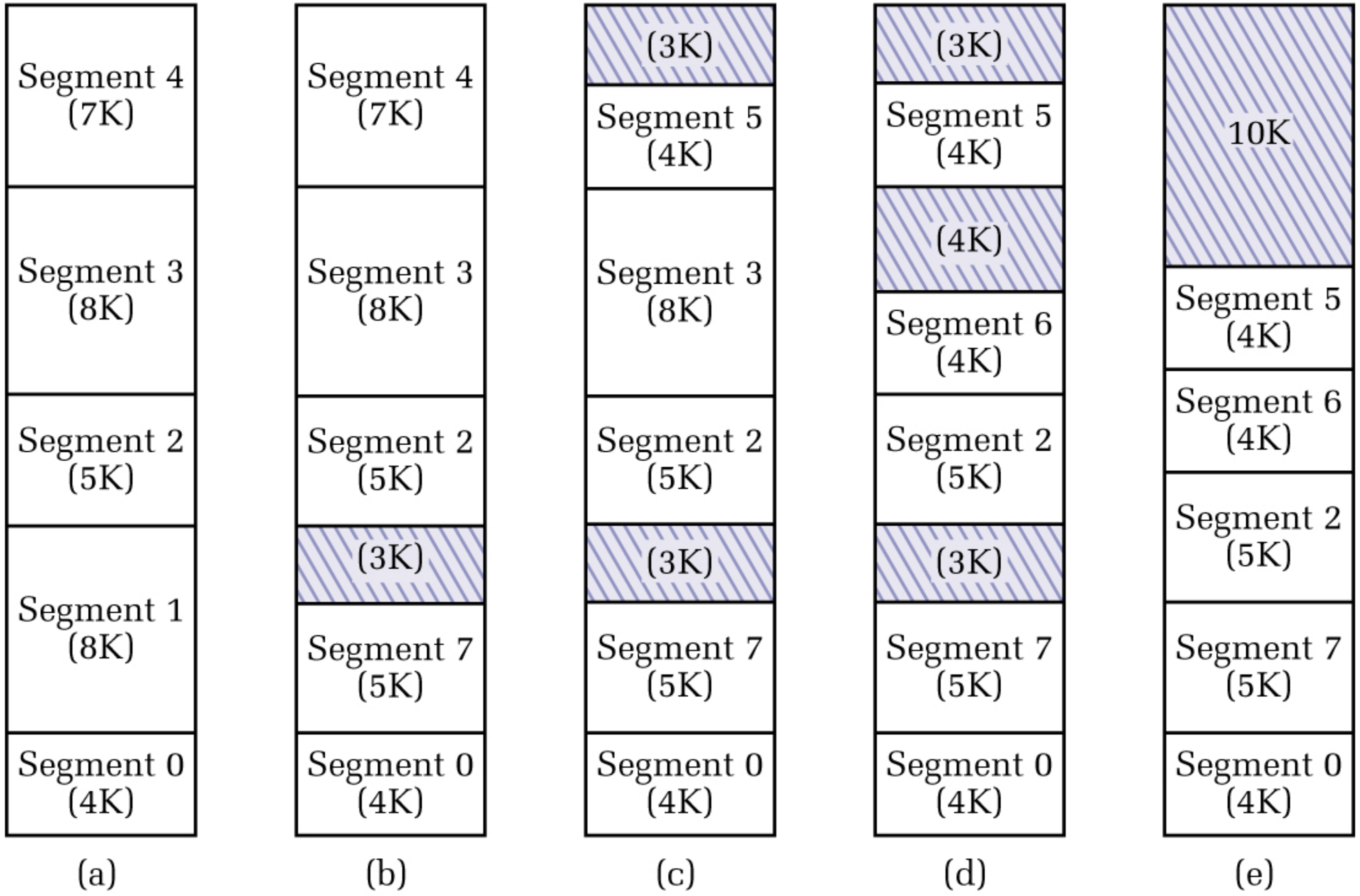
Problems for all three placement strategies for segmentation:

- Memory is divided into free and occupied areas → external fragmentation



- Free areas might be too small to store new segment

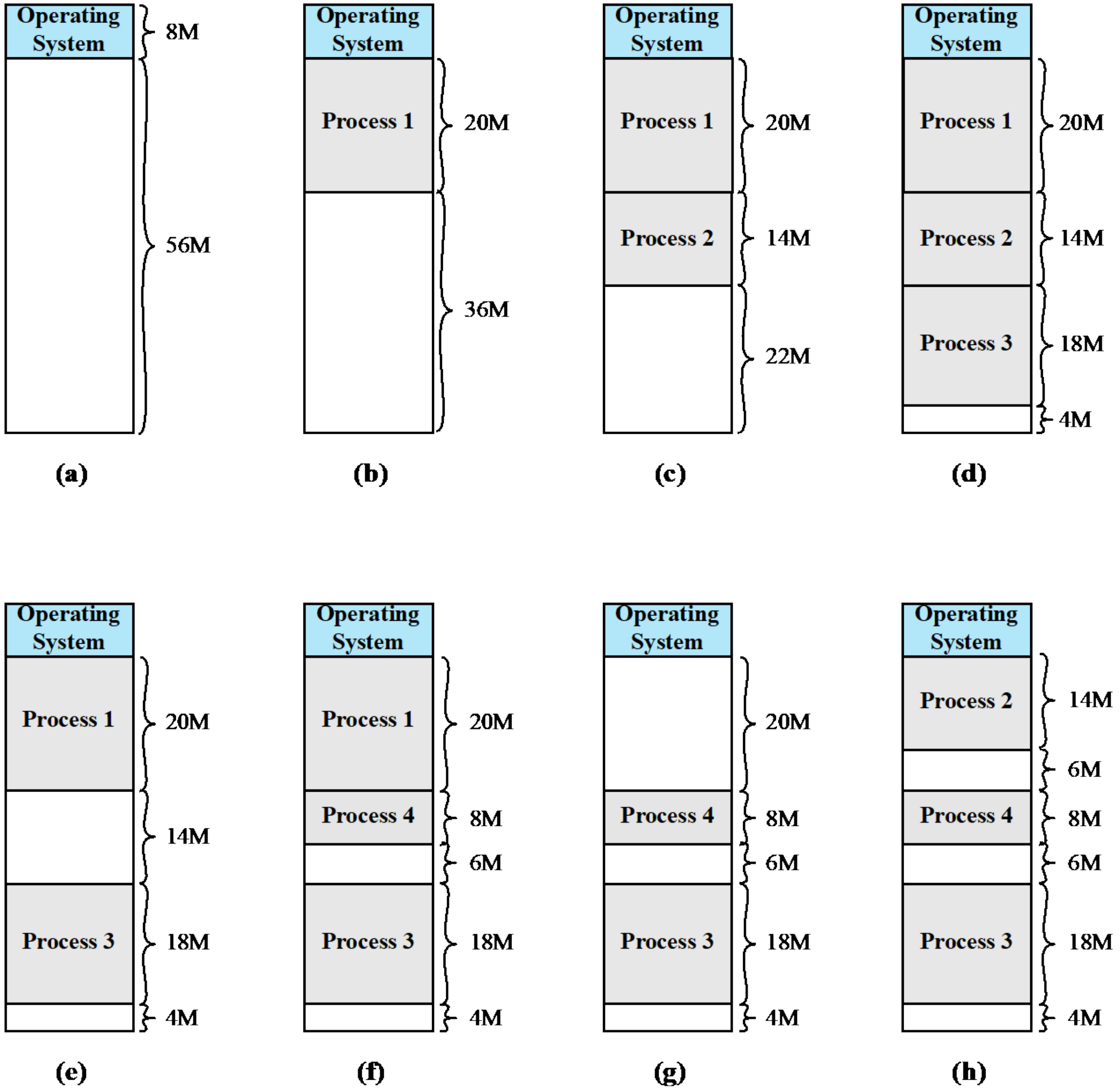
External Fragmentation and Compaction for Dynamic Partitioning



(a)-(d) Development of external fragmentation.

(e) Removal of the external fragmentation by compaction.

Dynamic Partitioning Example



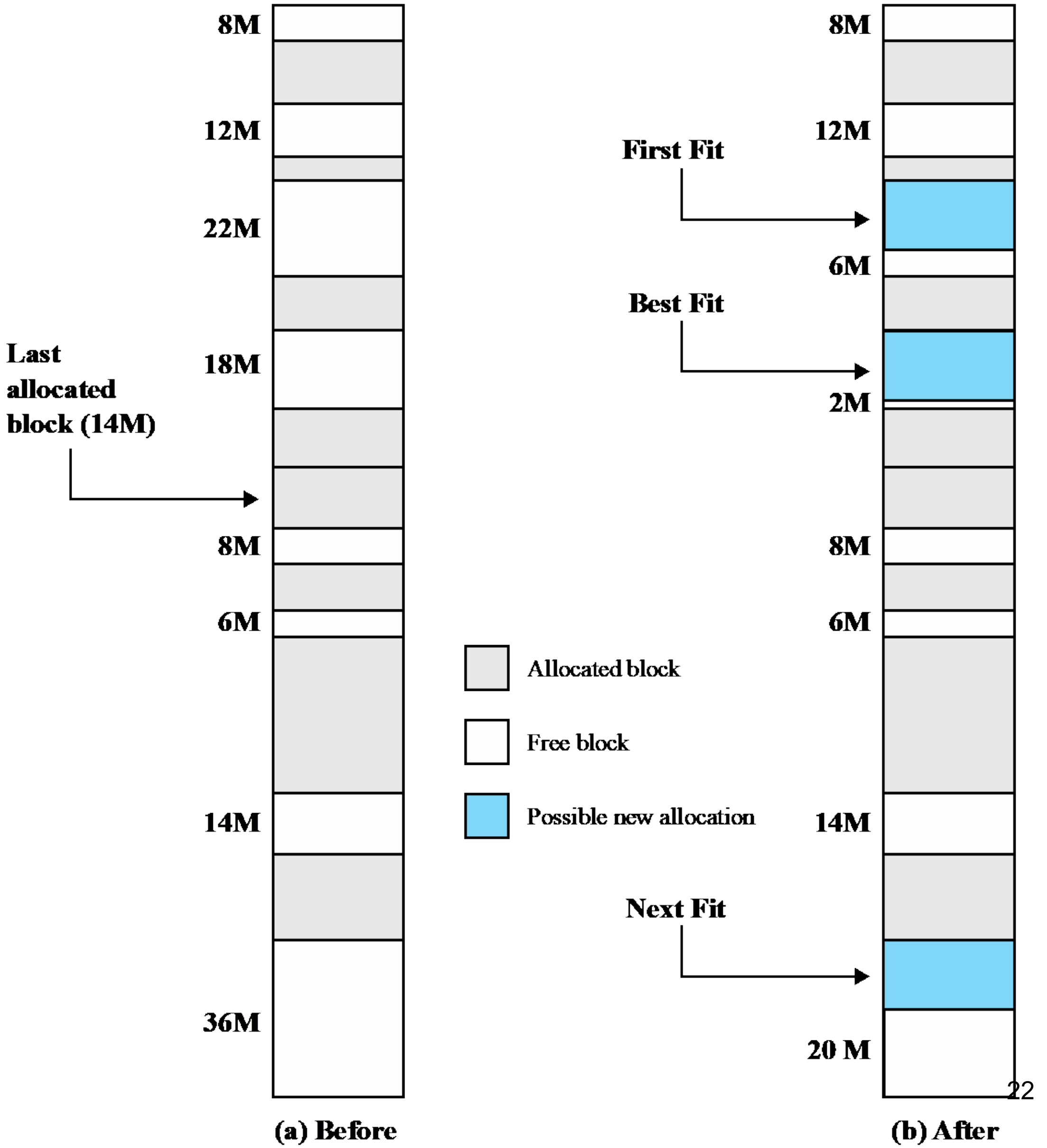
Dynamic Placement Algorithms

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)



Problems of virtual memory management

3. How to replace data

- Which segment or page is replaced to create free space for new data?

- For paging:
  - See replacement algorithms on following slide

- For segmentation:
  - E.g. limit number of simultaneously used segments for each process
  - → when loading new segment, one of the other segments of this process is replaced
  - Also generally possible: one of the replacement algorithms discussed for paging

Replacement Algorithms for Fixed Partitioning

Replacement for paging – 5 possible strategies for replacing pages

- FIFO (first-in-first-out)
  - Replace oldest page
- LIFO (last-in-first-out)
  - Replace youngest page
- LRU (least recently used)
  - Replace page that has not been accessed for longest time
- LFU (least frequently used)
  - Replace page that has smallest number of accesses
- LRD (least reference density)
  - Mix of LRU and LFU: replace page with least number of accesses in relation to fetch time

Additionally: prefer to replace pages that have not been changed
➡ no write back necessary

Remarks on Page/Segment faults

For both segment-oriented and page-oriented memory management:

- If segment or page is not available in main memory, interrupt is issued to load page or segment from secondary memory (segment fault or page fault)

How to detect such faults?

- For segmentation:
  - Bit in segment descriptor indicates if segment is available in main memory or not
- For paging:
  - Page number maintained in page table
  - Bit in page table indicates if page is available in main memory or not

Fragmentation of main memory

Fragmentation: free cells in main memory may be unusable because of the allocation scheme

- memory space might get wasted

Internal fragmentation: the free memory cells are within the area allocated to a process

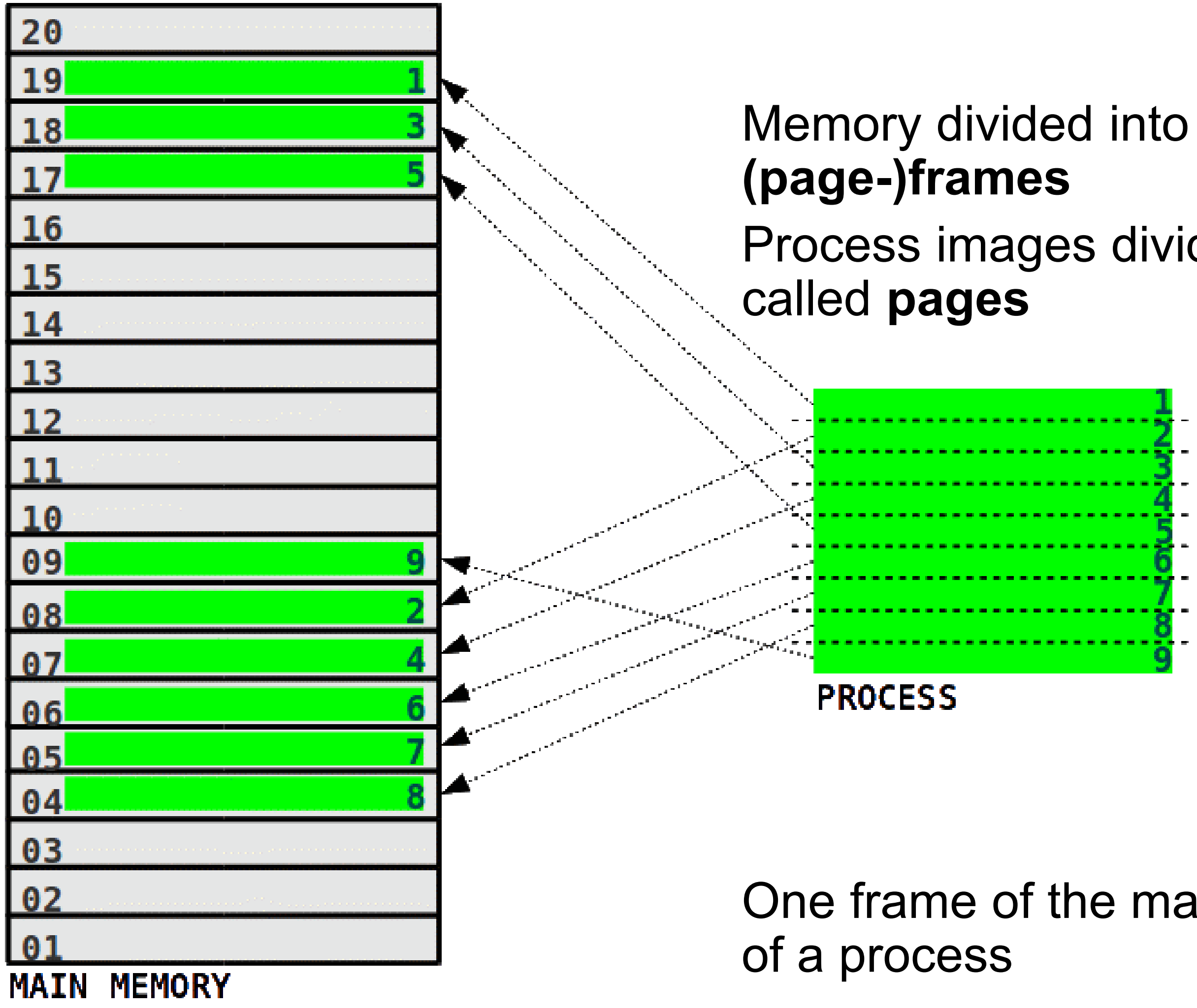
- occurs using fixed partitions

External fragmentation: the free memory cells are not in the area allocated to any process

- occurs using dynamic partitions

Paging

Paging

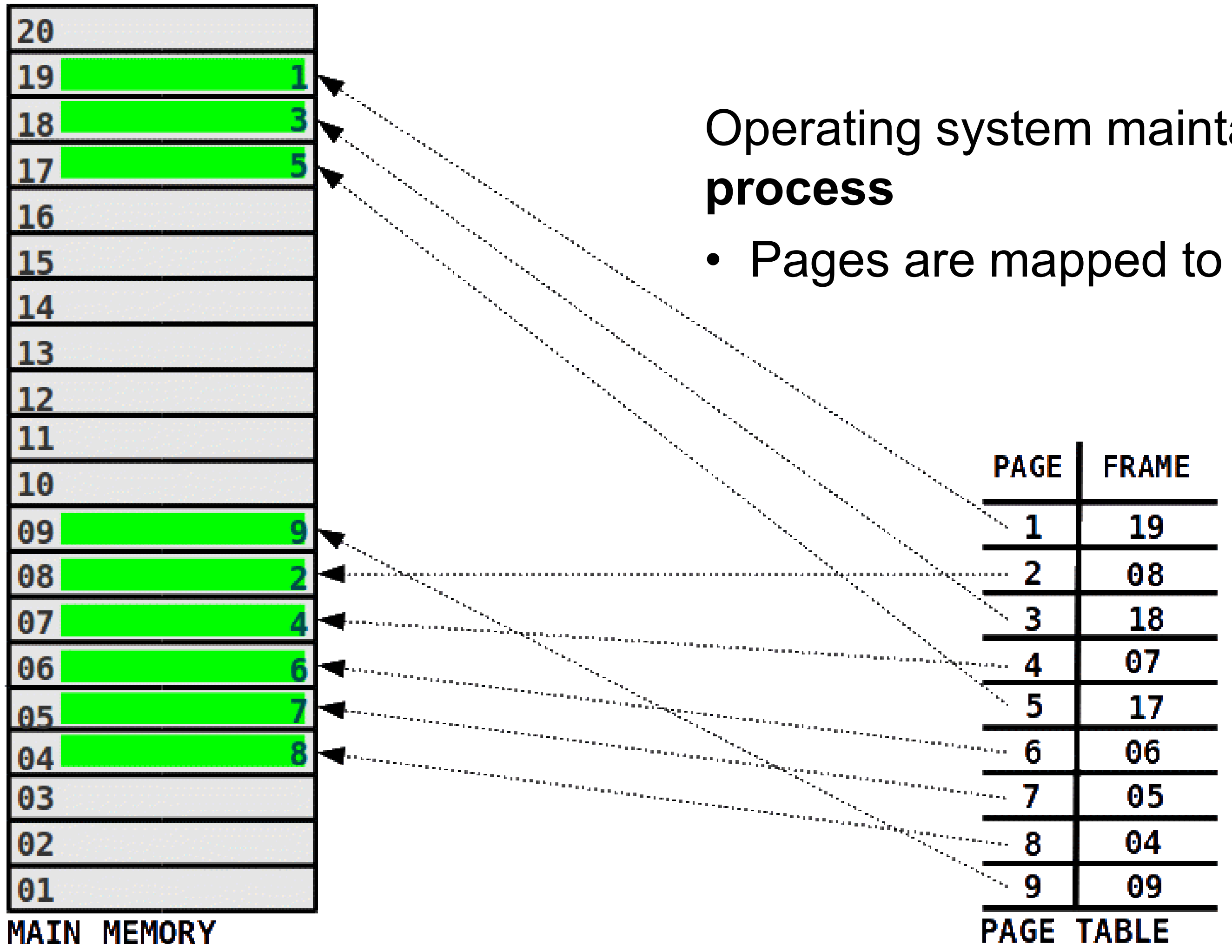


Memory divided into small fixed-sized pieces, called **(page-)frames**

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

Paging Example

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

(a)

0	N
1	N
2	N
3	N

Page Table Process A

0	N
1	N
2	N

Page Table Process B

0	N
1	N
2	N
3	N

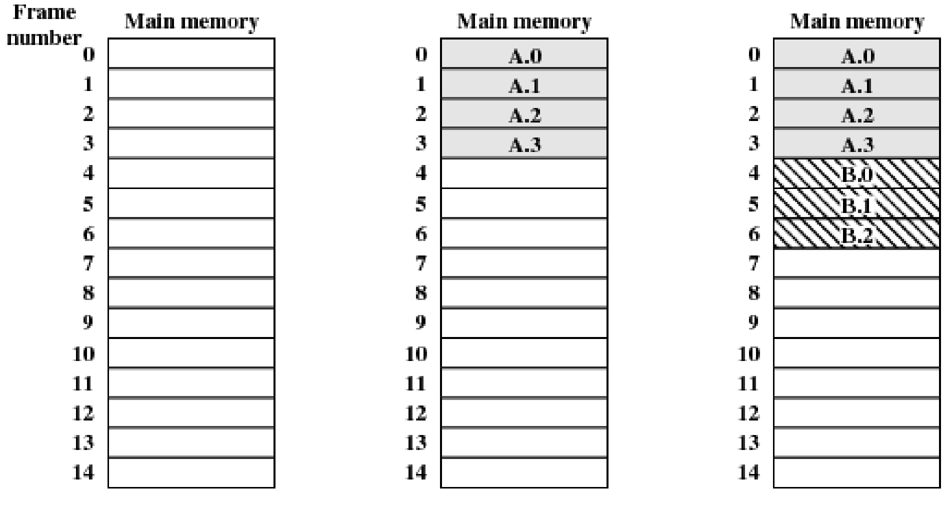
Page Table Process C

0	N
1	N
2	N
3	N
4	N

Page Table Process D

Free Frame List:

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14



(a) Fifteen Available Frames (b) Load Process A (c) Load Process B

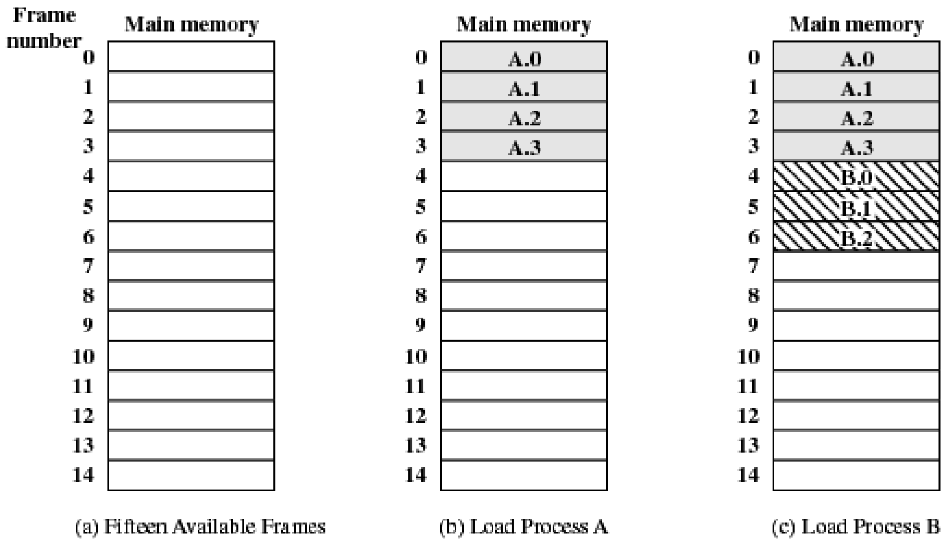
Assignment of Pages to Frames

Example:

(a) – (d) Load processes A, B, and C

(e) Swap out process B

(f) Load process D



Page Tables

(b)

	Page Table Process A
0	0
1	1
2	2
3	3

	Page Table Process B
0	N
1	N
2	N

	Page Table Process C
0	N
1	N
2	N
3	N

	Page Table Process D
0	N
1	N
2	N
3	N
4	N

Free Frame List:

4
5
6
7
8
9
10
11
12
13
14

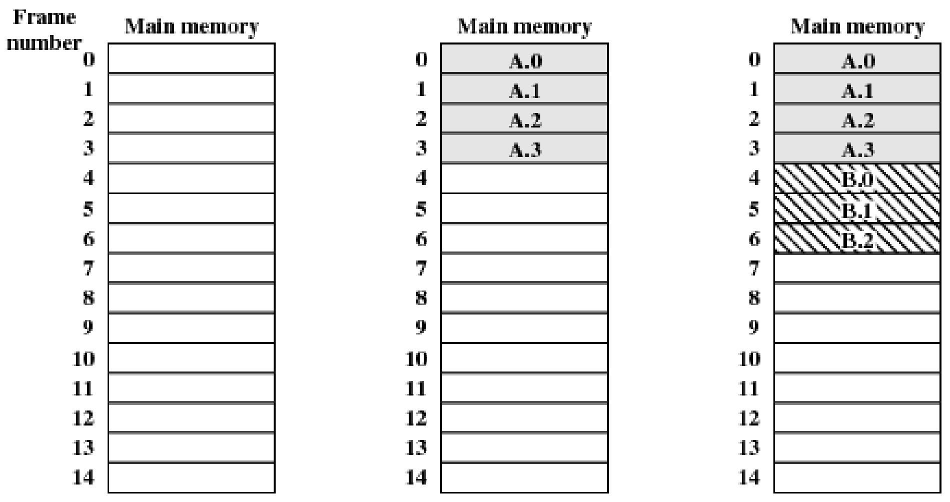
Assignment of Pages to Frames

Example:

(a) – (d) Load processes A, B, and C

(e) Swap out process B

(f) Load process D



(a) Fifteen Available Frames (b) Load Process A (c) Load Process B

Page Tables

(c)

0	0
1	1
2	2
3	3

Page Table Process A

0	4
1	5
2	6

Page Table Process B

0	N
1	N
2	N
3	N

Page Table Process C

0	N
1	N
2	N
3	N
4	N

Page Table Process D

Free Frame List:

7
8
9
10
11
12
13
14

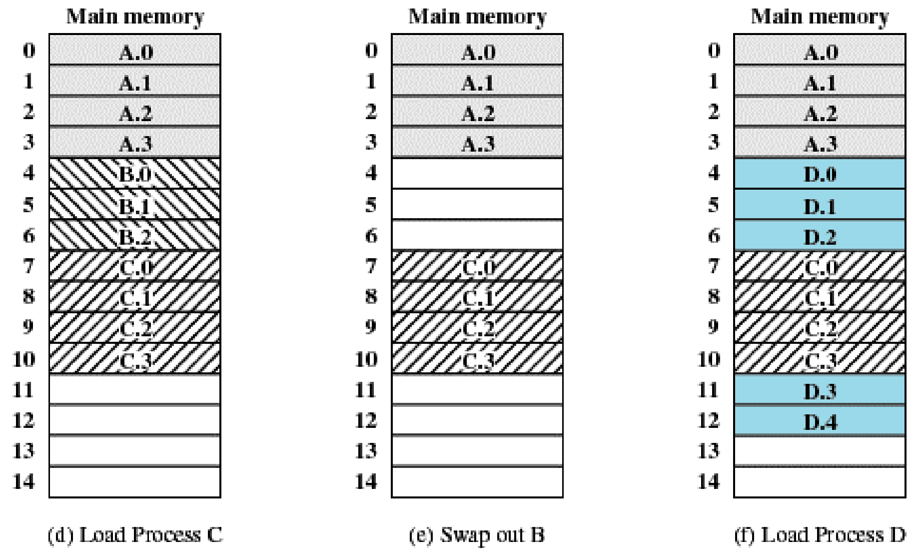
Assignment of Pages to Frames

Example:

(a) – (d) Load processes A, B, and C

(e) Swap out process B

(f) Load process D



Page Tables

(d)

Page Table Process A

0	0
1	1
2	2
3	3

Page Table Process B

0	4
1	5
2	6

Page Table Process C

0	7
1	8
2	9
3	10

Page Table Process D

0	N
1	N
2	N
3	N
4	N

Free Frame List:

11
12
13
14

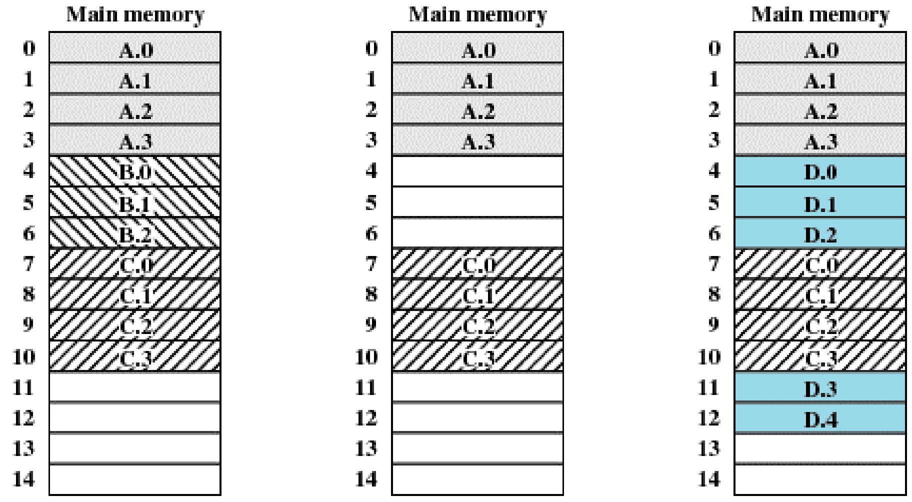
Assignment of Pages to Frames

Example:

(a) – (d) Load processes A, B, and C

(e) Swap out process B

(f) Load process D



(d) Load Process C (e) Swap out B (f) Load Process D

Page Tables

(e)

0	0
1	1
2	2
3	3

Page Table Process A

0	N
1	N
2	N

Page Table Process B

0	7
1	8
2	9
3	10

Page Table Process C

0	N
1	N
2	N
3	N
4	N

Page Table Process D

Free Frame List:

4
5
6
11
12
13
14

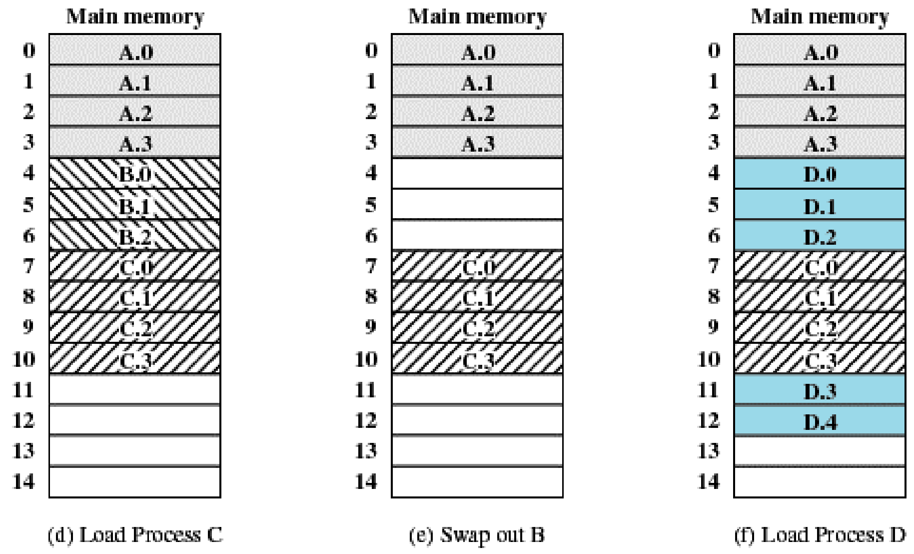
Assignment of Pages to Frames

Example:

(a) – (d) Load processes A, B, and C

(e) Swap out process B

(f) Load process D



Page Tables

(f)

Page	Page Table Process A	Page Table Process B	Page Table Process C	Page Table Process D
0	0	N	7	4
1	1	N	8	5
2	2	N	9	6
3	3		10	11
4				12

Free Frame List:

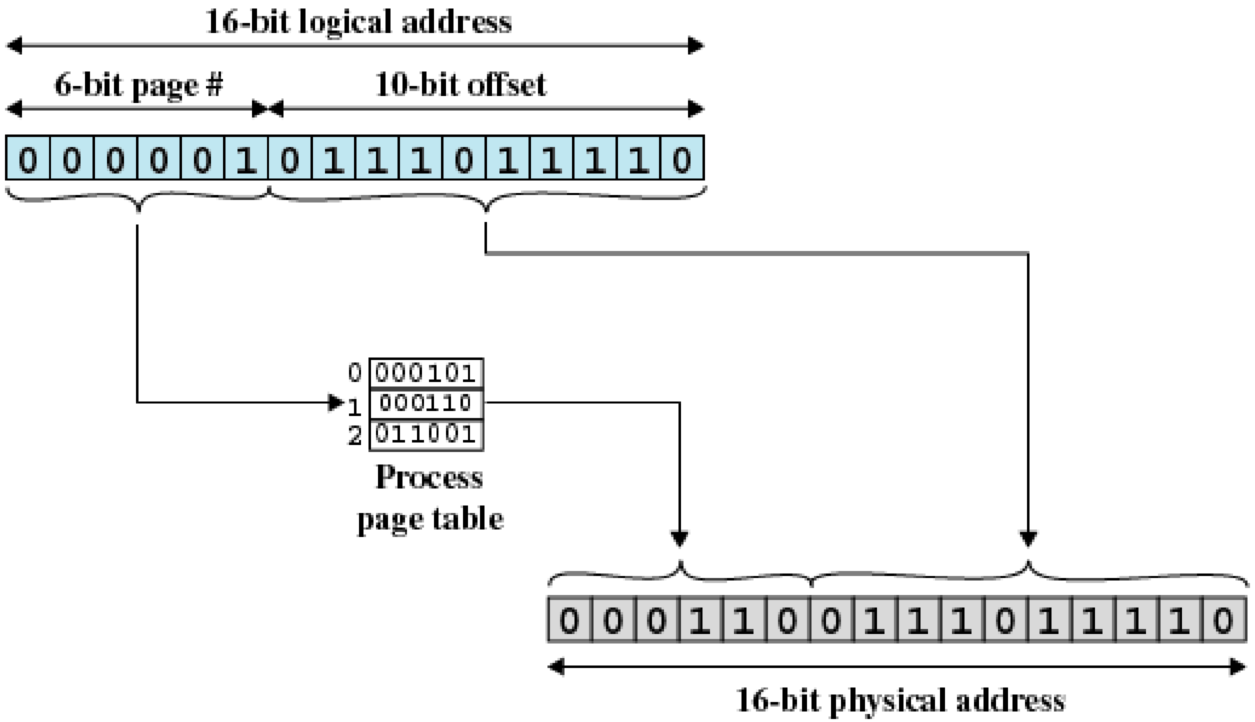
13
14

Address Translation

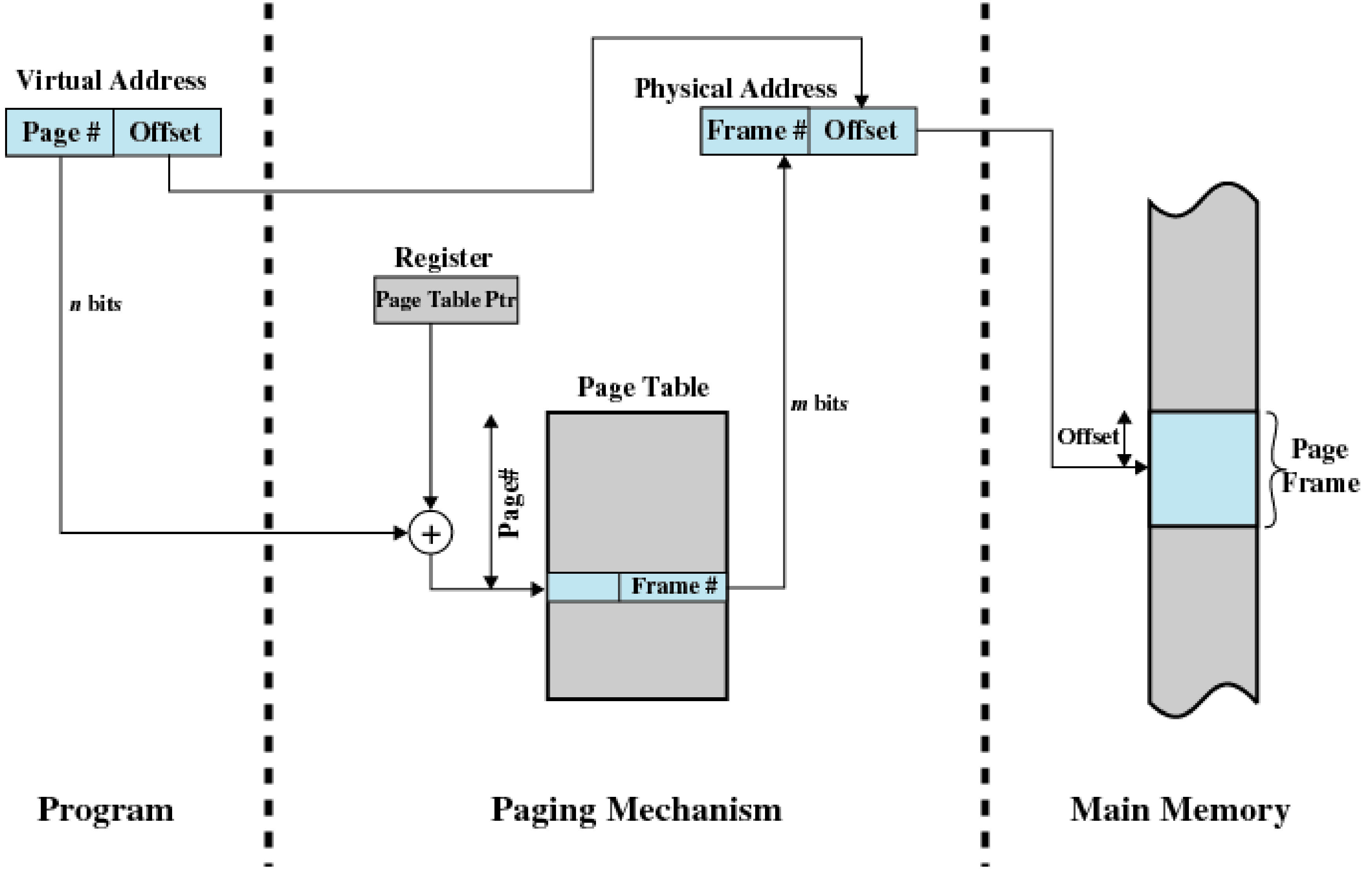
Addresses

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

Example for Address Translation from virtual to physical address:



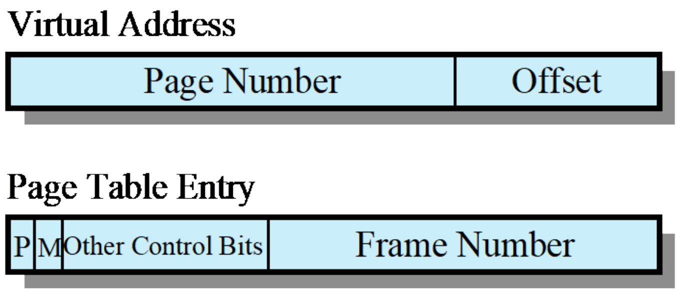
Paging Address Translation



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

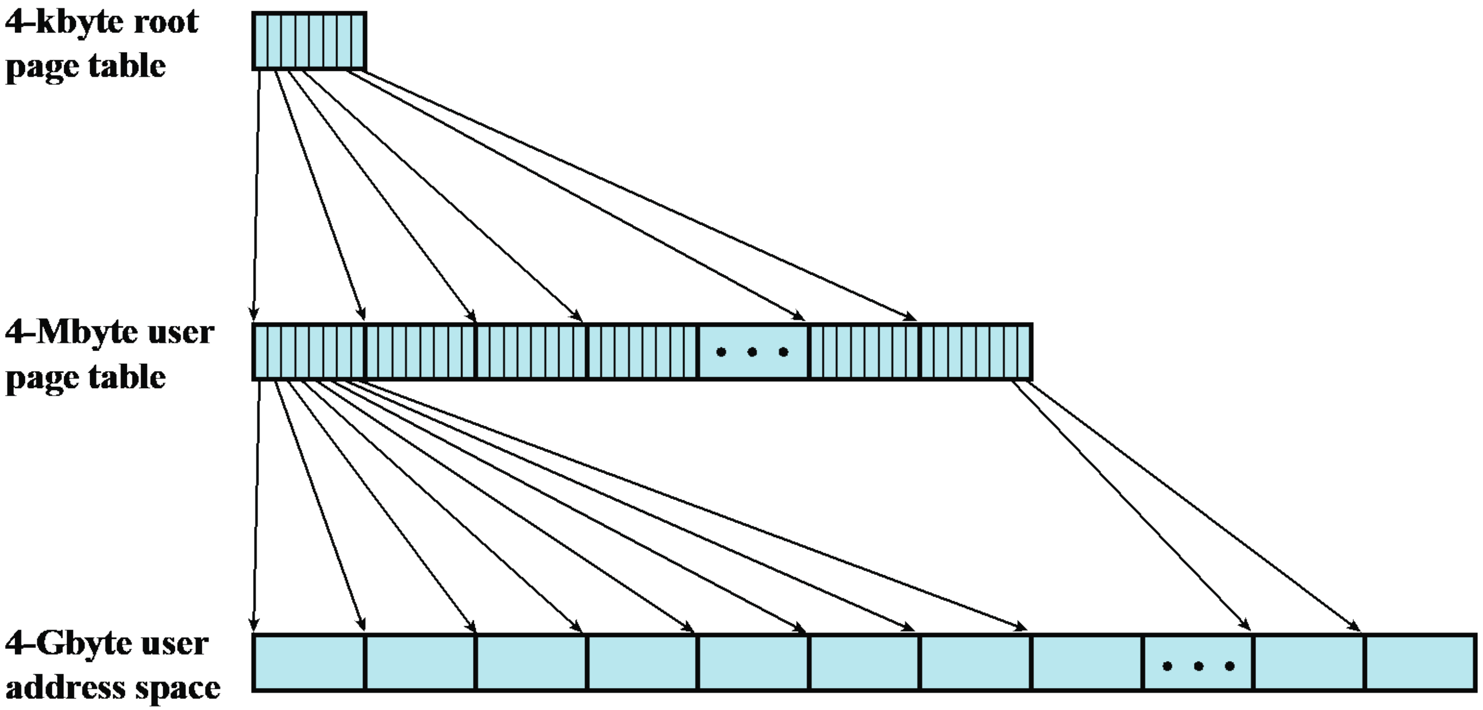
Other control bits:

- 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

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

Increased page size causes pages to contain locations further from any recent reference

- Page faults rise

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

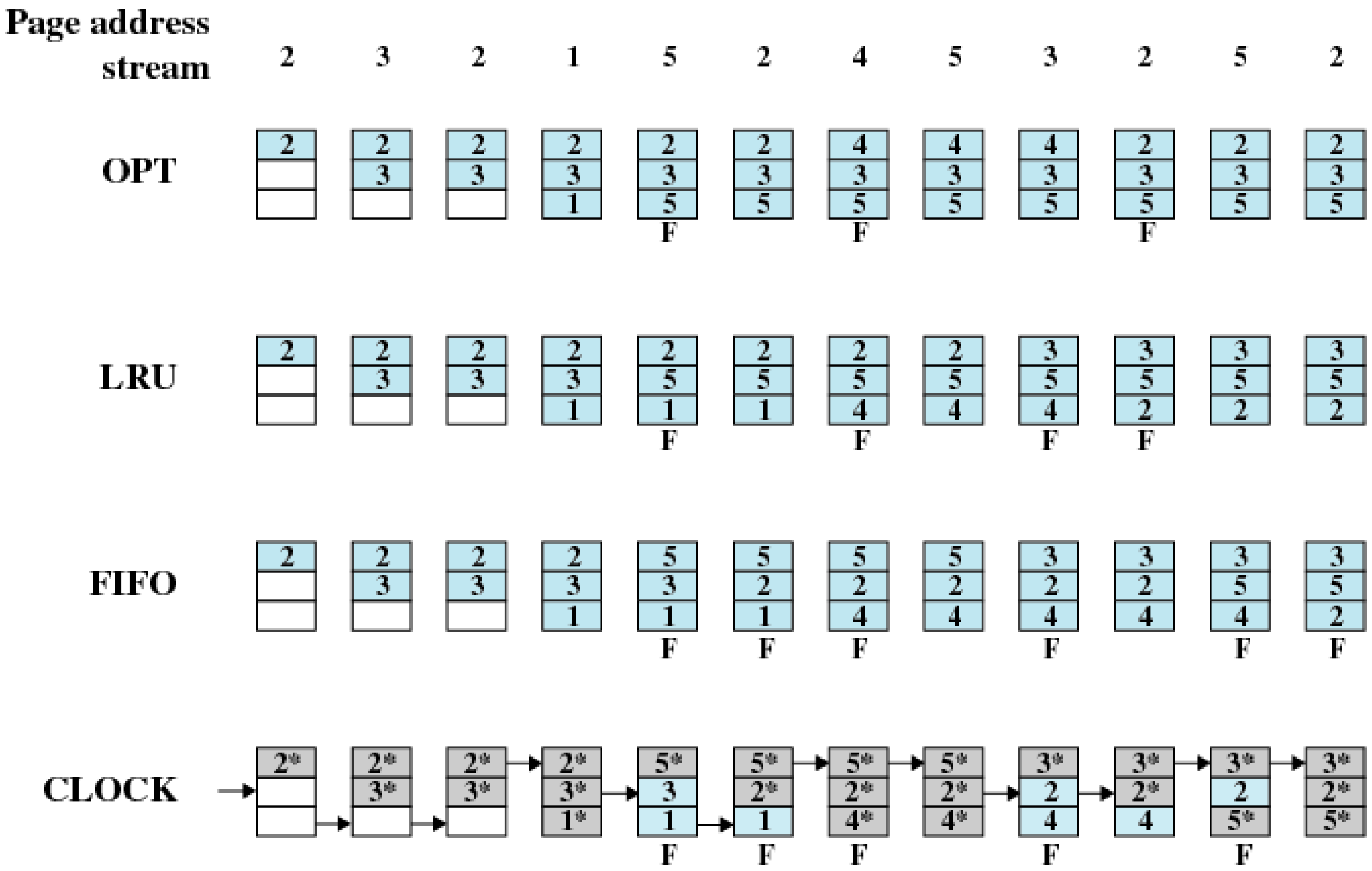
➢ *Impossible* to have perfect knowledge of future events

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)

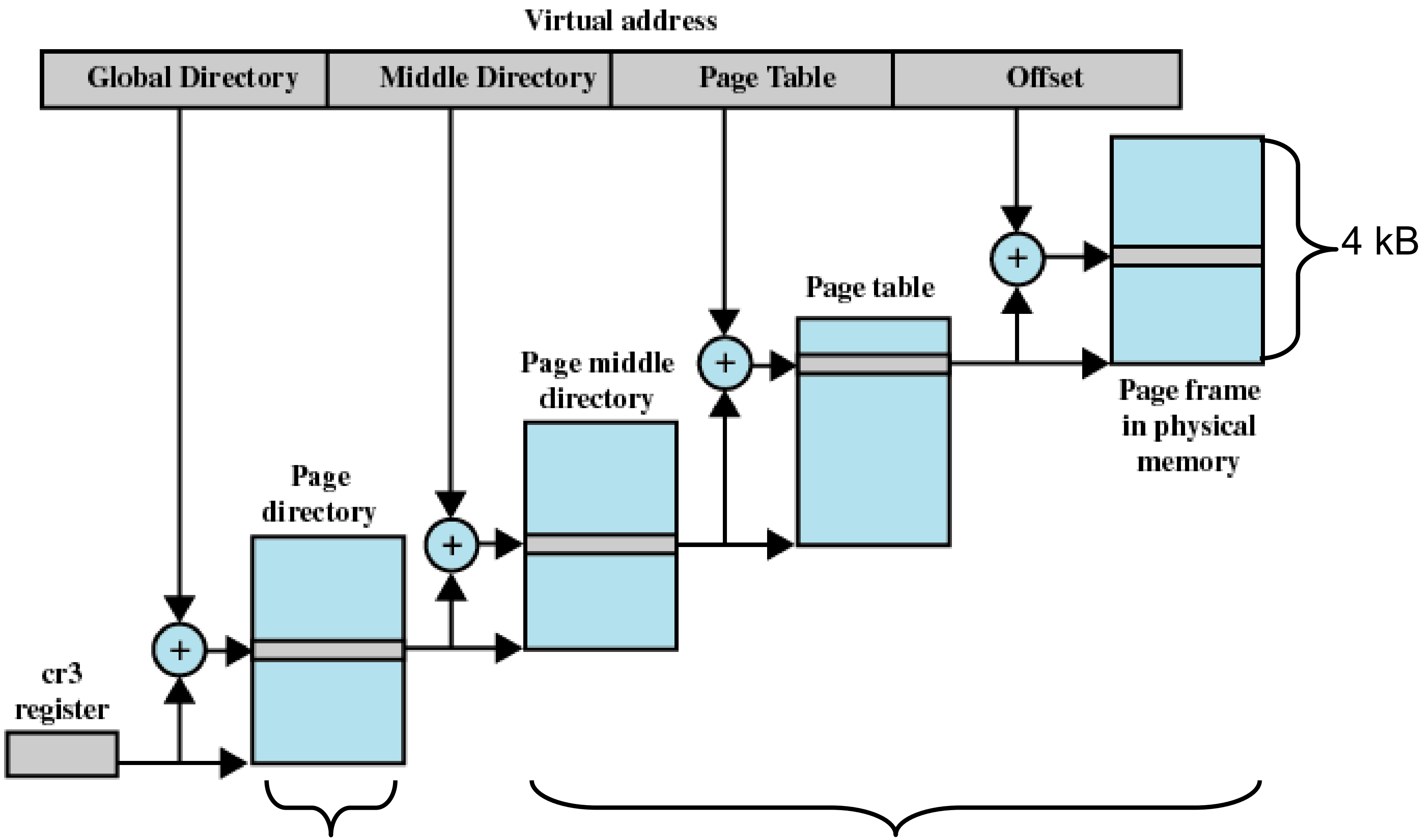
Clock Policy
- When a page is first loaded in memory, *use bit* is set to 1
- When page is referenced, use bit is set to 1
- During search for replacement, each use bit is changed to 0
- When replacing pages, first frame with use bit set to 0 is replaced

Page Replacement Example



F = page fault occurring after the frame allocation is initially filled

Example: Linux VM Implementation



Roadmap

1. Introduction and Motivation
2. Interrupts and System Calls
3. Processes
4. Scheduling
5. Memory
6. **I/O and File System**
7. Booting, Services, and Security