

2026

CS 332

Subject 7

NJIT

A white curved line that starts under the 'N', goes under the 'J' and 'I', and ends under the 'T', creating a swoosh effect.

New Jersey's Science &
Technology University

THE EDGE IN KNOWLEDGE

Alex. Gerbessiotis

CS Department



The yellowish/brownish notes are copyright the textbook author.

The bluish notes are copyright the instructor and are add-ons and provide more explanation or additional details



Module 1

*Operating
Systems:
Internals
and Design
Principles*

Chapter 7 Memory Management

Ninth Edition
William Stallings

Frame	A fixed-length block of main memory.
Page	A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.
Segment	A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging).

Table 7.1

Memory Management Terms

First computers



0000

Program X

1500

Logical view of exe file

X: **Start:** 0000 **End:** 1500 **Length:** 1501

Physical view of exe file

X: **Start:** 0000 **Length:** 1501

All (memory) belongs to X !



1000

Program Z

2500

Logical view of exe file

Z: **Start:** 1000 **End:** 2500 **Length:** 1501

Physical view of exe file

Z: **Start:** 1000 **Length:** 1501

Need to accommodate MONITOR program
(aka Interrupt Handling plus may be more
aka rudimentary and primitive OS)

Assume OS + interrupt handling at addresses 0 to 999?

- Next step:
1. More advanced OS
 2. Two or more programs (processes) in MM at a time!



0000

Program X

1500

0000

Program Y

Logical view of exe files (View 1)

X: **Start:** 0000 **End:** 1500 **Length:** 1501
 Y: **Start:** 0000 **End:** 1499 **Length:** 1500

Logical view of exe files to be used: no END

X: **Start:** 0000 **Length:** 1501
 Y: **Start:** 0000 **Length:** 1500

X execution of an instruction

PC := 300
 M[300] = "LOAD AC, 1001"

Y execution of an instruction

PC := 800
 M[800] = "LOAD AC, 1001"

Two programs cannot co-exist in Main Memory if logical view is physical view as well (i.e. processes are stored in the same area of MM)
 They use the same UserAddressSpace (0 to 1499)
 Moreover, when an OS will be introduced the OS will be using address 0 (at least) for interrupt handling related activities

Next Page: A register known as SBR is introduced
 SBR: Segment Base Register
 All address arithmetic involves adding SBR's value
 SBR : Starting address of a process (in Main Memory)



0100

Process X

1600

01700

Process Y

Logical view of exe files

X:	Start: 0000	Length: 1501
Y:	Start: 0000	Length: 1500

Physical view (Kernel) of processes

X:	Start: 0100	Length: 1501
Y:	Start: 1700	Length: 1500

Execution: Logical vs Physical (Real)

X execution of an instruction

PC := 300
 M[300] = "LOAD AC, 1001"

using offset arithmetic

SBR = Start.X // 0100
 PC = SBR + 300
 M[SBR+300] = "LOAD AC, SBR+1001"

PC = 400
 M[400] = "LOAD AC, 1101"

Y execution of an instruction

PC := 800
 M[800] = "LOAD AC, 1001"

SBR = Start.Y // 1700
 PC = SBR + 800
 M[SBR+800] = "LOAD AC, SBR+1001"

physical (actual) locations: do additions

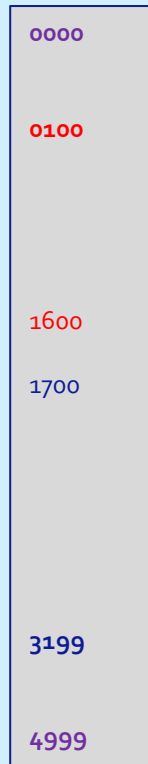
PC = 2500
 M[2500] = "LOAD AC, 2701"

Global Kernel Tables: next page

SBR = Segment Base Register : we resolve address L by adding SBR to L!



MT (Memory Table)



Process X

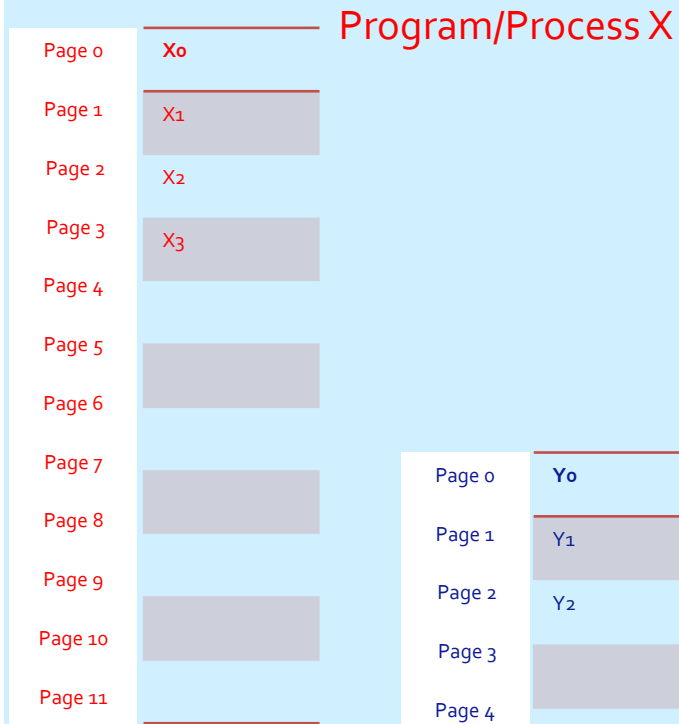
Process Y

What is that gap
Between 1601 and 1699?
We will call it external
fragmentation!
How to fix it? **NEXT PAGE**

That's the past ...



Moving Forward : Paging



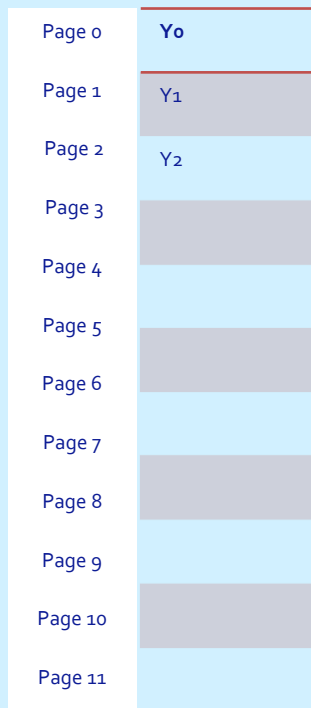
Logical Representation of exe files

	start	len		start	len
X	0000	1501	Y	0000	1500
PC := 300			PC := 800		
M[300] = "LOAD AC, 1001"			M[800] = "LOAD AC, 1001"		

Reminder from Subject 0:

#P (number of pages) = $\text{ceil}(\text{len}/\text{PageSize})$ **PageSize* = 500B given**
Pstart = $\text{start}/\text{PageSize}$ **Pstart=Starting Page Number**

Program/Process Y



Page view

#P: number of pages

X's new table

Pstart	#P	len
0	4	1501
(will use 0,1,2,3)		

Y's new table

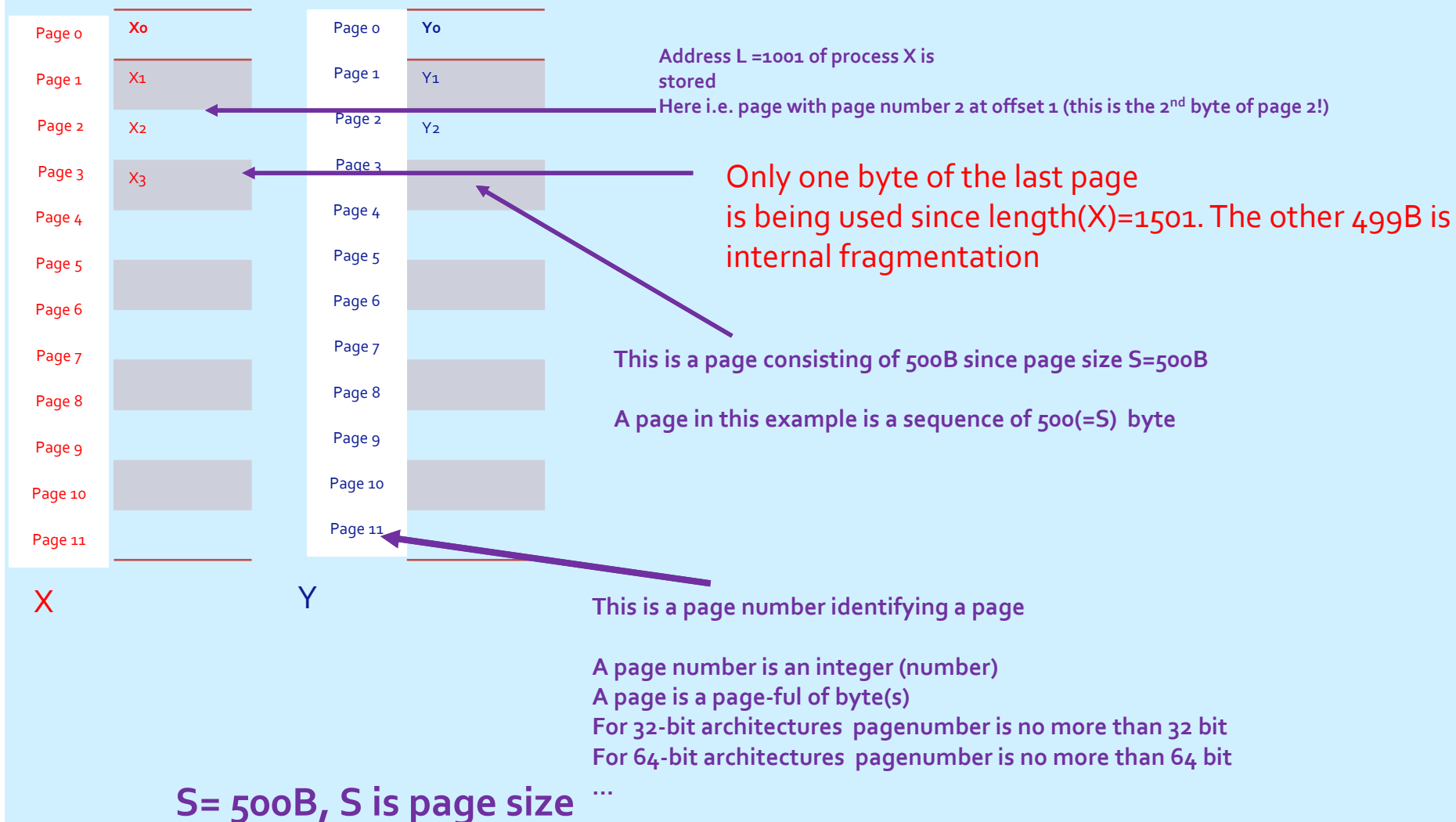
Pstart	#P	len
0	3	1500
(will use 0,1,2)		

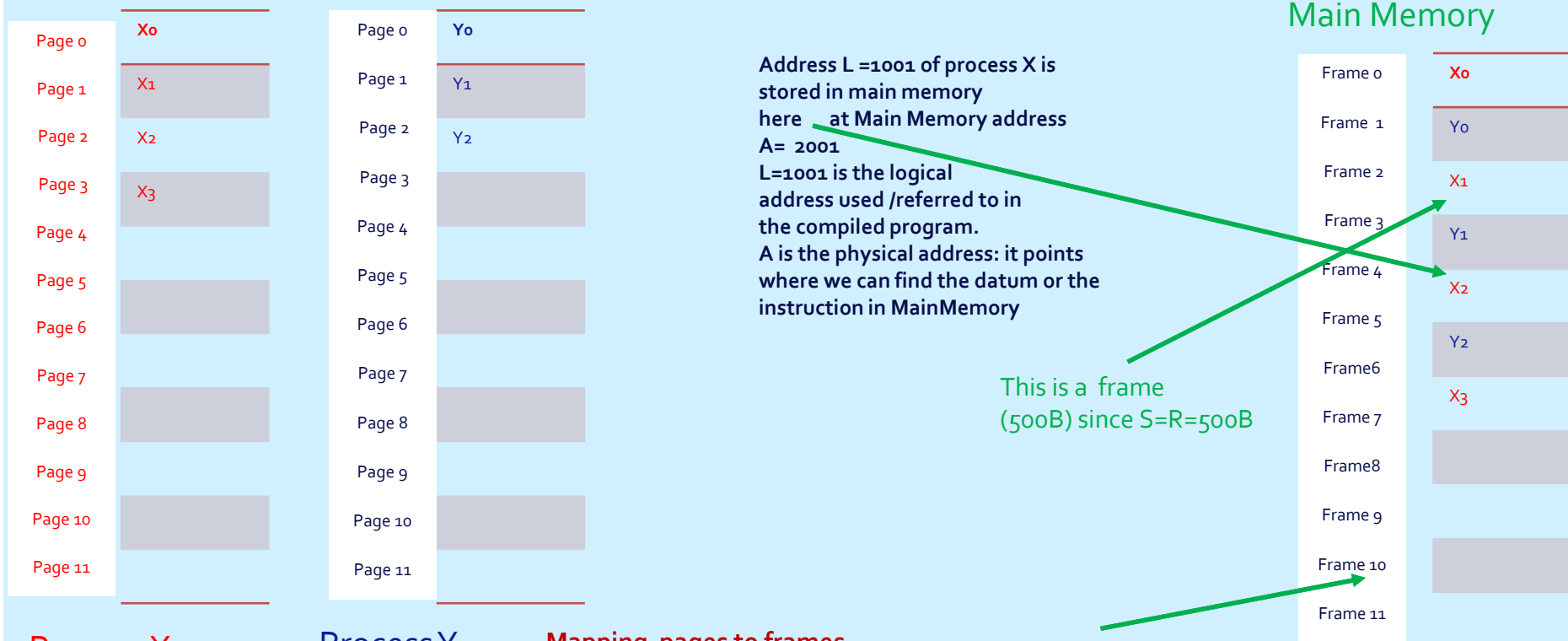
PC := 300 = (P#, offset) = (0, 300)
1001 = (P#, offset) = (2, 1)
M[(0, 300)] = "LOAD AC, (2, 1)"

PC := 800 = (P#, offset) = (1, 300)
1001 = (P#, offset) = (2, 1)
M[(1, 300)] = "LOAD AC, (2, 1)"

2001 ToBe Explained

*should be a power of 2; but this is an example: keep things simple





Process X

Process Y

Mapping pages to frames

page : logical view division of exe file

frame: physical division of main memory

we map a page onto a frame

a page is identified by a page number,

a frame is identified by a frame number

The mapping is available inside the

PAGETABLE of the process in question!

(i.e. one PAGETABLE per process)

R = 500B, R is frame size
R = S by default



Page 0	X ₀	Page 0	Y ₀
Page 1	X ₁	Page 1	Y ₁
Page 2	X ₂	Page 2	Y ₂
Page 3	X ₃	Page 3	
Page 4		Page 4	
Page 5		Page 5	
Page 6		Page 6	
Page 7		Page 7	
Page 8		Page 8	
Page 9		Page 9	
Page 10		Page 10	

Process X

Process Y

S = 500B

Every process has its own page table,
 A page table is stored in Main Memory

PageTable of Process **X**

Page Number is the index Frame Number is the 'contents' of PT

0	0
1	2
2	4
3	6

Frame number
PT[2] = 4
 Page number

PageTable of Process **Y**

Page Number Frame Number

0	1
1	3
2	5

Main Memory

Frame 0	X ₀
Frame 1	Y ₀
Frame 2	X ₁
Frame 3	Y ₁
Frame 4	X ₂
Frame 5	Y ₂
Frame 6	X ₃
Frame 7	
Frame 8	
Frame 9	
Frame 10	

R = 500B

A page table stores ... **frame numbers**
 A page table index is a **page number**
 A page table **DOES NOT STORE** pages!



1. In a process say **X** we use in its text a

LOAD AC, 1001

implies

Logical address

L

L = 1001

2. In paging with parameters S,R (page size S == frame size R), a byte of a logical address is stored in a page thus for L we need to find the page number P and offset T associated with that byte at (logical) address L i.e.

$P = L / S, T = L \% S$

P is the page number, T is page offset (Subject o)

$L = (P, T)$

3. A page is stored in MainMemory by mapping pages of an executable file to frames of MM. Thus we need to find where in MM we store the page with page number P and thus identify the frame containing the page by identifying its frame number F. We use the page table of process X to find the mapping P→F. This requires a Main Memory access since the page table of X is stored in main memory!

i.e. $F = PT[P] = M[PT + P * \text{sizeof}(\text{datatype}(P))]$

1st memory access

$F = PT[P]$

4. Instructions of the CPU need addresses in Main Memory neither logical addresses nor page numbers, nor frame numbers nor offsets. Thus from the step 3 computed F and and the step 2 computed T we need to construct a physical (or actual or absolute or real) address A of main memory to which L is mapped. The A is known as the

Actual or Physical address A from F and T

$A = (F, T)$

5. The actual address A is computed using F and T and frame size R as follows:

$A = F * R + T$

6. What do we do with the A?

A memory reference ! 2nd memory access

$M[A]$

This paging mechanism and translation steps are supported by the **Memory Management Unit (MMU) of the CPU.**

Question 1. For L=1001 of process X what is its A? Answer follows...

Question 2. For L=300 of process X what is its A?

Practice 1



THEORY

Page Size $S = 500B$, $R = 500B$

Step1	Logical Address	L
Step 2	Convert L into How? (From elementary school	$L = \langle PT \rangle$ $P = L/S$, $T = L \% S$ $L = P * S + T$
Step 3	Lookup PageTable of X and convert page number P into a frame number F (slide 14)	$F = PT[P]$
Step 4	Convert $\langle F, T \rangle$ into A How?	$\langle F, T \rangle$ $A = \langle F, T \rangle$ $A = F * R + T$

PRACTICE(EXAMPLE)

$L = 1001$ of Process X

$L = \langle 2, 1 \rangle$

$P = 1001 / 500 = 2$, $T = 1001 \% 500 = 1$

$F = PT[2] = 4$

$P = 2$ maps to $F = 4$

$\langle F, T \rangle = \langle 4, 1 \rangle$

$A = \langle 4, 1 \rangle$

$A = 4 * 500 + 1 = 2001$



2001 Finally Explained
 (Q1 of Slide 11)

Note: Page size (and also Frame size) is **NOT** a power of 2. $S = R = 500B$

Note: Mapping holds because $S = R$

Note: If S and R are a power of two, bit manipulation is possible and neither divisions are needed, nor multiplications, nor additions (see Subject 0 or Figure 7.12 of the textbook also included in these slides)

Note: If pagesize (and thus framesize) is a power of two (500 is NOT a power of two), we can use bit manipulations instead of divisions (/ and %).



Requires S to be a power of two: See Subject 6 or Fig 7.12 of textbook and these slides (later)

1. In a process say X we use a **Logical address in binary L**

2. If pagesize S and framesize R, $S=R$, are a power of two, then

2a. Writedown or convert L into binary,

2b. Compute $\lg(S)$

2c. Isolate $\lg(S)$ rightmost bit of binary L ; it is T

2d. Isolate remaining leftmost bit of binary L ; it is P

$$L = \langle P, T \rangle$$

3. The page table is next accessed all access in binary. An F in binary will be generated!

i.e. $F = PT[P] = M[PT + P * \text{sizeof}(\text{datatype}(P))]$

1st memory access

$$F = PT[P]$$

4. We converted the L into a P and T, and the P into an F. Memory access requires a binary address not (P,T) or (F,T) pairs. We bring the F and T both in binary close together: they will form the actual/physical/real/absolute address.

Actual, Physical, Real or Absolute address

$$A = (F, T)$$

5. A is formed as the concatenation of the bit of F and T into one binary sequence/number

$$A = \langle F, T \rangle$$

6. What do we do with the A ?

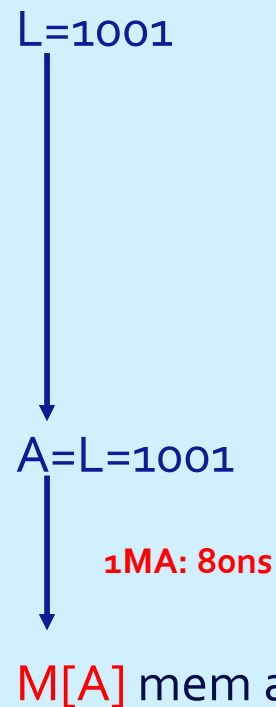
A memory reference ! 2nd memory access

$$M[A]$$

Starting with L we obtained M[A] in 2 memory accesses!

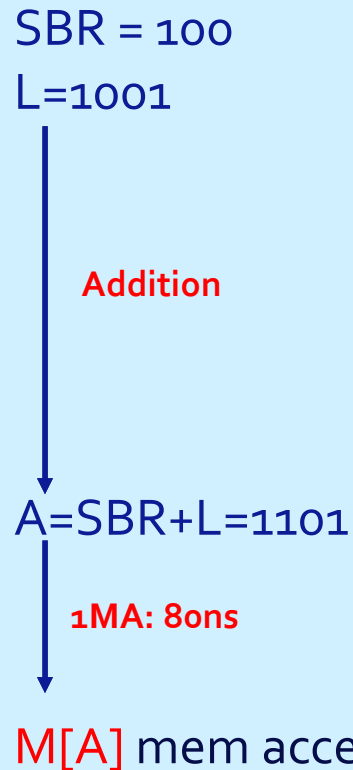


1940-1960s Simple Addressing (Subject 1)

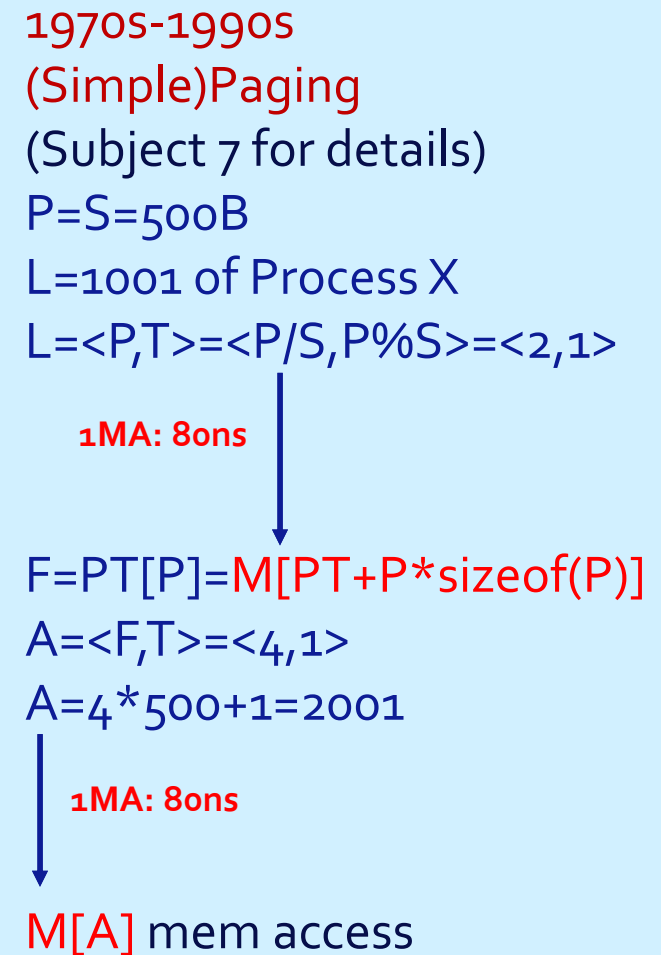


Time: $1 \times 8\text{ons} = 8\text{ons}$

1950s-1970s Segment Addressing (Subject 7)



Time: $1 \times 8\text{ons} + 1\text{ADD} \sim 8\text{ons}$



Time: $2 \times 8\text{ons} = 16\text{ons}$



Frame: a 'page' in main memory
Page: a 'page' in exec on HDD i.e.
a 'logical page' in program/process

Textbook Definition

Page
Frame

Alternative

Logical Page
Page

Instead of using confusingly
'logical page' and 'page' we prefer to use instead
page and frame respectively!

Pages and Frames are of fixed length (not necessarily the same) but so far $S=R$

Memory Management Requirements

- Memory management is intended to satisfy the following requirements:
 - Relocation (swapping, or inside main me)
 - Protection
 - Sharing
 - Logical organization
 - Physical organization

Relocation

- Programmers typically do not know in advance which other programs will be resident in main memory at the time of execution of their program
- Active processes need to be able to be swapped in and out of main memory in order to maximize processor utilization
- Specifying that a process must be placed in the same memory region when it is swapped back in would be limiting
 - May need to *relocate* the process to a different area of memory

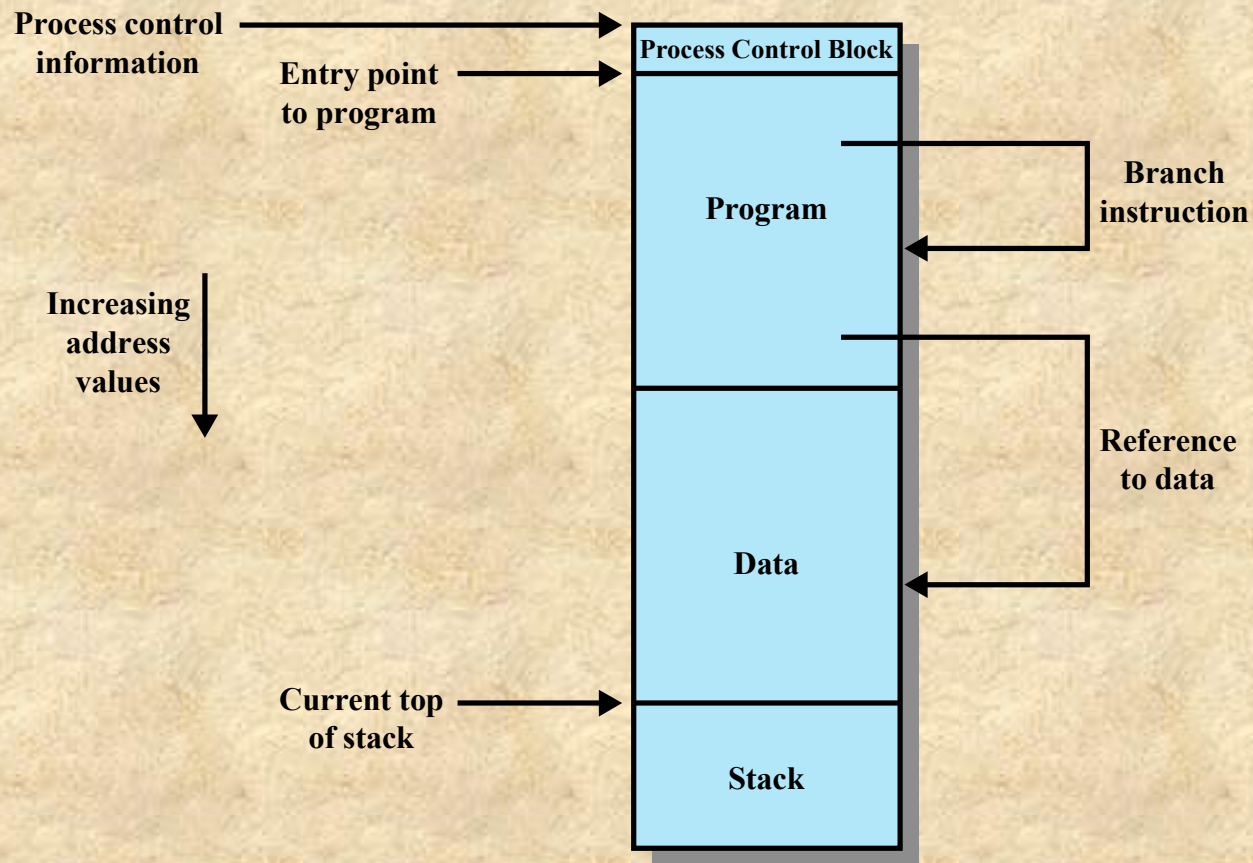


Figure 7.1 Addressing Requirements for a Process

Protection

- Processes need to acquire permission to reference memory locations for reading or writing purposes
- Location of a program in main memory is unpredictable
- Memory references generated by a process must be checked at run time
- Mechanisms that support relocation also support protection

Sharing

- Advantageous to allow each process access to the same copy of the program rather than have their own separate copy
- Memory management must allow controlled access to shared areas of memory without compromising protection
- Mechanisms used to support relocation support sharing capabilities

Logical Organization

- Memory is organized as linear

Programs are written in modules

- Modules can be written and compiled independently
 - Different degrees of protection given to modules (read-only, execute-only)
 - Sharing on a module level corresponds to the user's way of viewing the problem
- Segmentation is the tool that most readily satisfies requirements

Physical Organization

Cannot leave the programmer with the responsibility to manage memory

Memory available for a program plus its data may be insufficient

Programmer does not know how much space will be available

Overlaying allows various modules to be assigned the same region of memory but is time consuming to program

Memory Partitioning

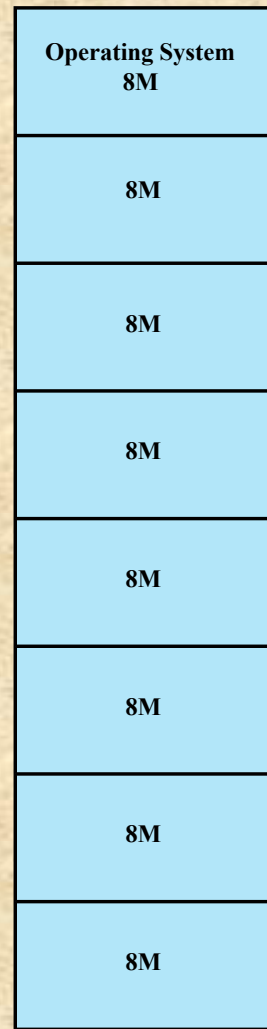
- Memory management brings processes into main memory for execution by the processor
 - Involves virtual memory
 - Based on segmentation and paging
- Partitioning
 - Used in several variations in some now-obsolete operating systems
 - Does not involve virtual memory

Table 7.2

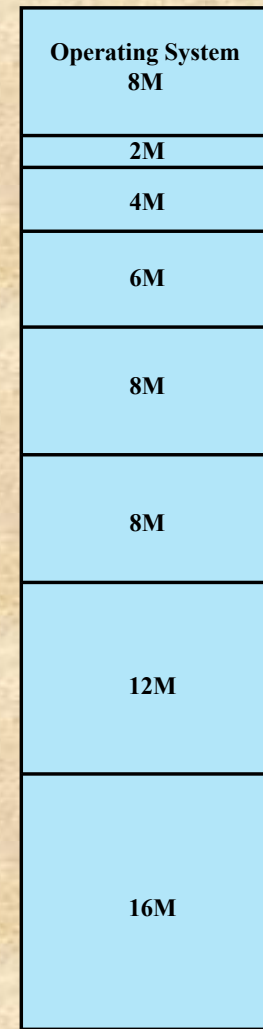
Memory Management Techniques

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
Virtual Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

(Table is on page 317 in textbook)



(a) Equal-size partitions



(b) Unequal-size partitions

Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Disadvantages

- A program may be too big to fit in a partition
 - Program needs to be designed with the use of overlays
- Main memory utilization is inefficient
 - Any program, regardless of size, occupies an entire partition
 - *Internal fragmentation*
 - Wasted space due to the block of data loaded being smaller than the partition

Disadvantages

- The number of partitions specified at system generation time limits the number of active processes in the system
- Small jobs will not utilize partition space efficiently

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as it requires
- This technique was used by IBM's mainframe operating system, OS/MVT

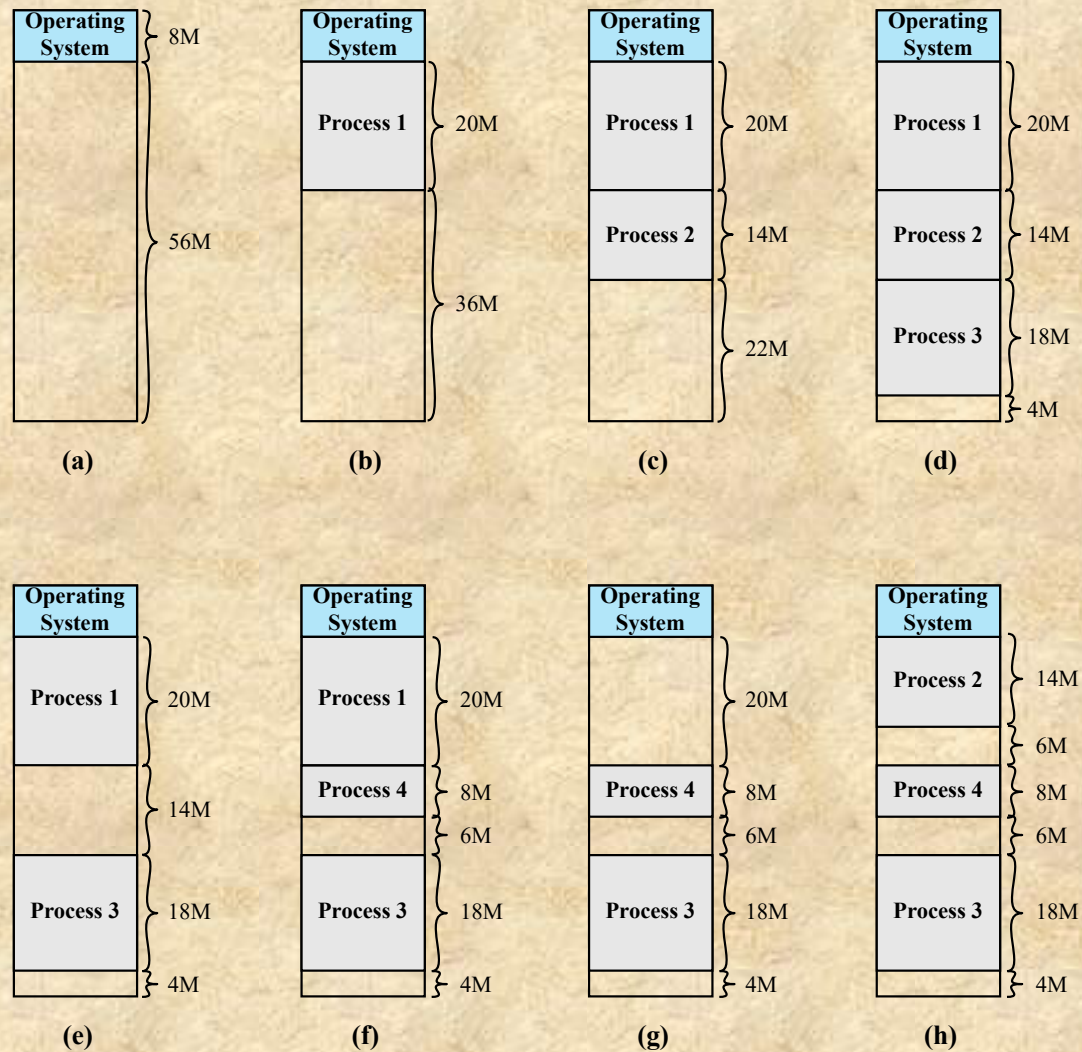


Figure 7.4 The Effect of Dynamic Partitioning

Dynamic Partitioning

External Fragmentation

- Memory becomes more and more fragmented
- Memory utilization declines

Compaction

- Technique for overcoming external fragmentation
- OS shifts processes so that they are contiguous
- Free memory is together in one block
- Time consuming and wastes CPU time



External Fragmentation: There is enough (fragmented) memory to satisfy a process request but it is not consecutively allocated

Internal Fragmentation: A process was allocated Y bytes yet it needs only X bytes where $X < Y$.

Eg a program request for 1000B but it is allocated 4096B=4KiB

Placement Algorithms

Best-fit

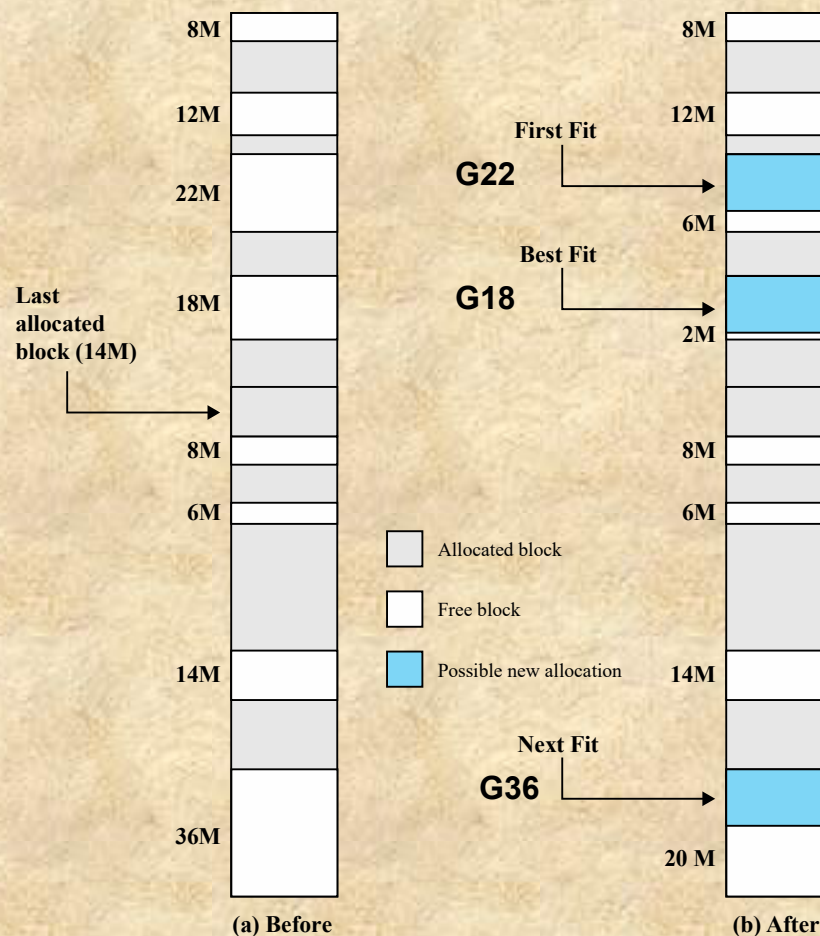
- Chooses the block that is closest in size to the request

First-fit

- Begins to scan memory from the beginning and chooses the first available block that is large enough

Next-fit

- Begins to scan memory from the location of the last placement and chooses the next available block that is large enough



**"Internal" to OS
Fragmentation is
known as external
Fragmentation!
(note the " "
quotation marks)**

**External
fragmentation:**

**Request for a
50MiB block**

One could describe (a)Before 'horizontally' as
G8A UA G12 UB G22 UC G18 UD UE G8B ...

Where U indicates an area of memory of unknown size that is in use and
G12 indicates a gap of unused memory (the 12 means now 12M). A G8A vs G8B distinguishes
the first from the second 'Gap' of the same size!

**Figure 7.5 Example Memory Configuration before
and after Allocation of 16-Mbyte Block**

Buddy System

- Comprised of fixed and dynamic partitioning schemes
- Space available for allocation is treated as a single block
- Memory blocks are available of size 2^K words, $L \leq K \leq U$, where
 - 2^L = smallest size block that is allocated
 - 2^U = largest size block that is allocated; generally 2^U is the size of the entire memory available for allocation

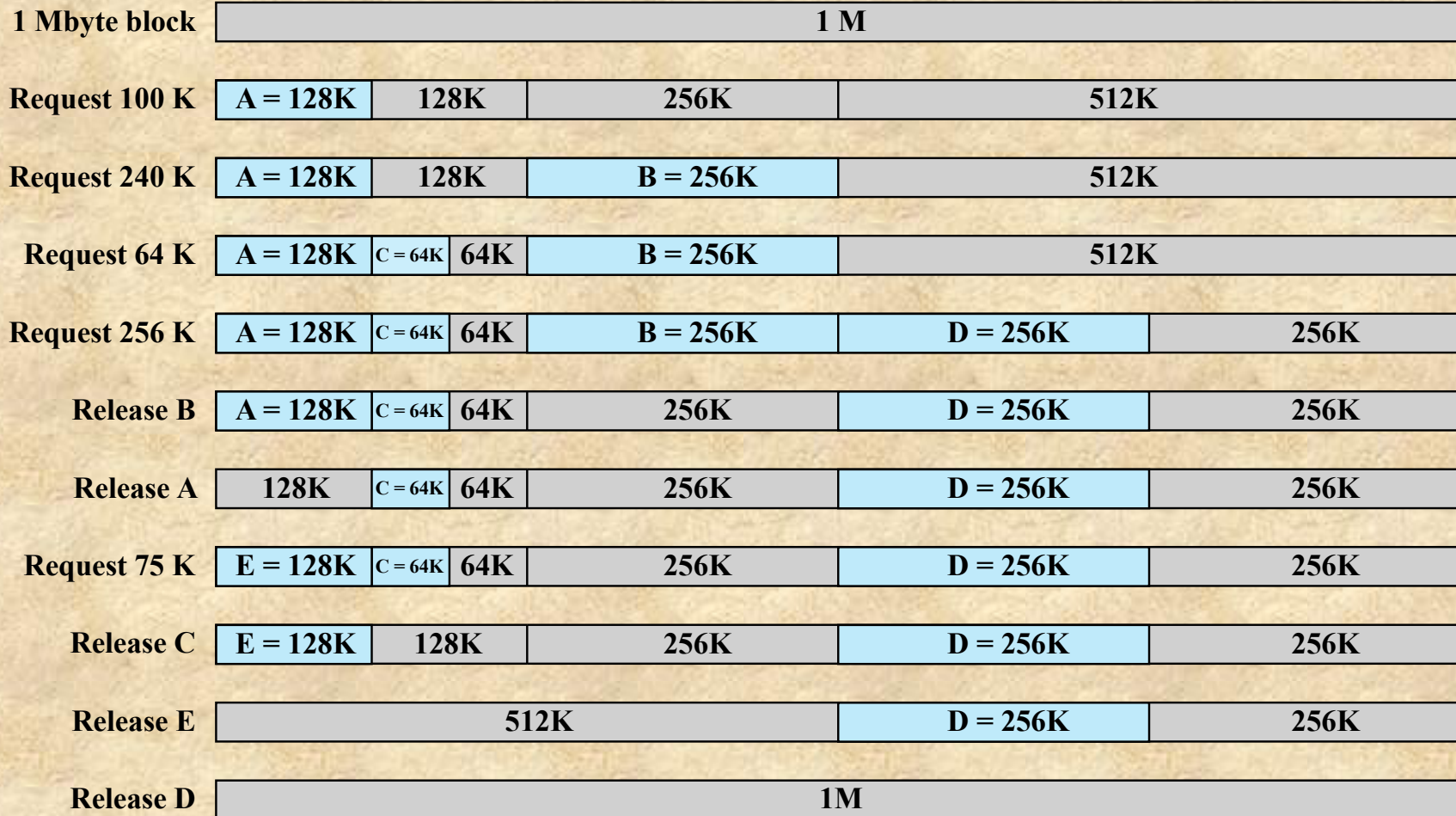


Figure 7.6 Example of Buddy System

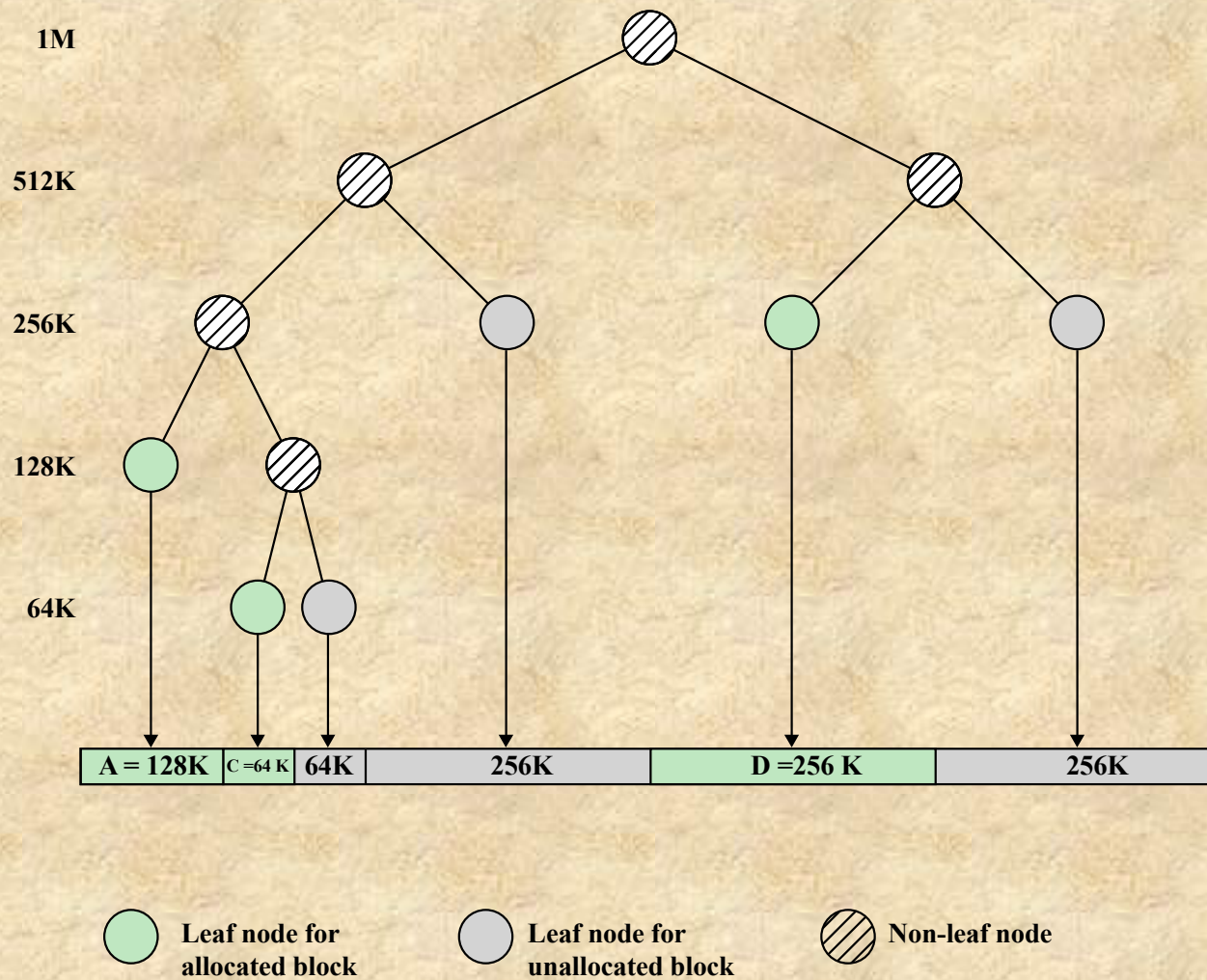


Figure 7.7 Tree Representation of Buddy System

Relocation

- When the fixed partition scheme is used, we can expect a process will always be assigned to the same partition
 - Whichever partition is selected when a new process is loaded will always be used to swap that process back into memory after it has been swapped out
 - In that case, a simple relocating loader can be used
 - When the process is first loaded, all relative memory references in the code are replaced by absolute main memory addresses, determined by the base address of the loaded process
- In the case of equal-size partitions and in the case of a single process queue for unequal-size partitions, a process may occupy different partitions during the course of its life
 - When a process image is first created, it is loaded into some partition in main memory; Later, the process may be swapped out
 - When it is subsequently swapped back in, it may be assigned to a different partition than the last time
 - The same is true for dynamic partitioning
- When compaction is used, processes are shifted while they are in main memory
 - Thus, the locations referenced by a process are not fixed
 - They will change each time a process is swapped in or shifted

Addresses

Logical Intels calls it **LINEAR**

- Reference to a memory location independent of the current assignment of data to memory

Relative

- A particular example of logical address, in which the address is expressed as a location relative to some known point

Physical or Absolute

- Actual location in main memory

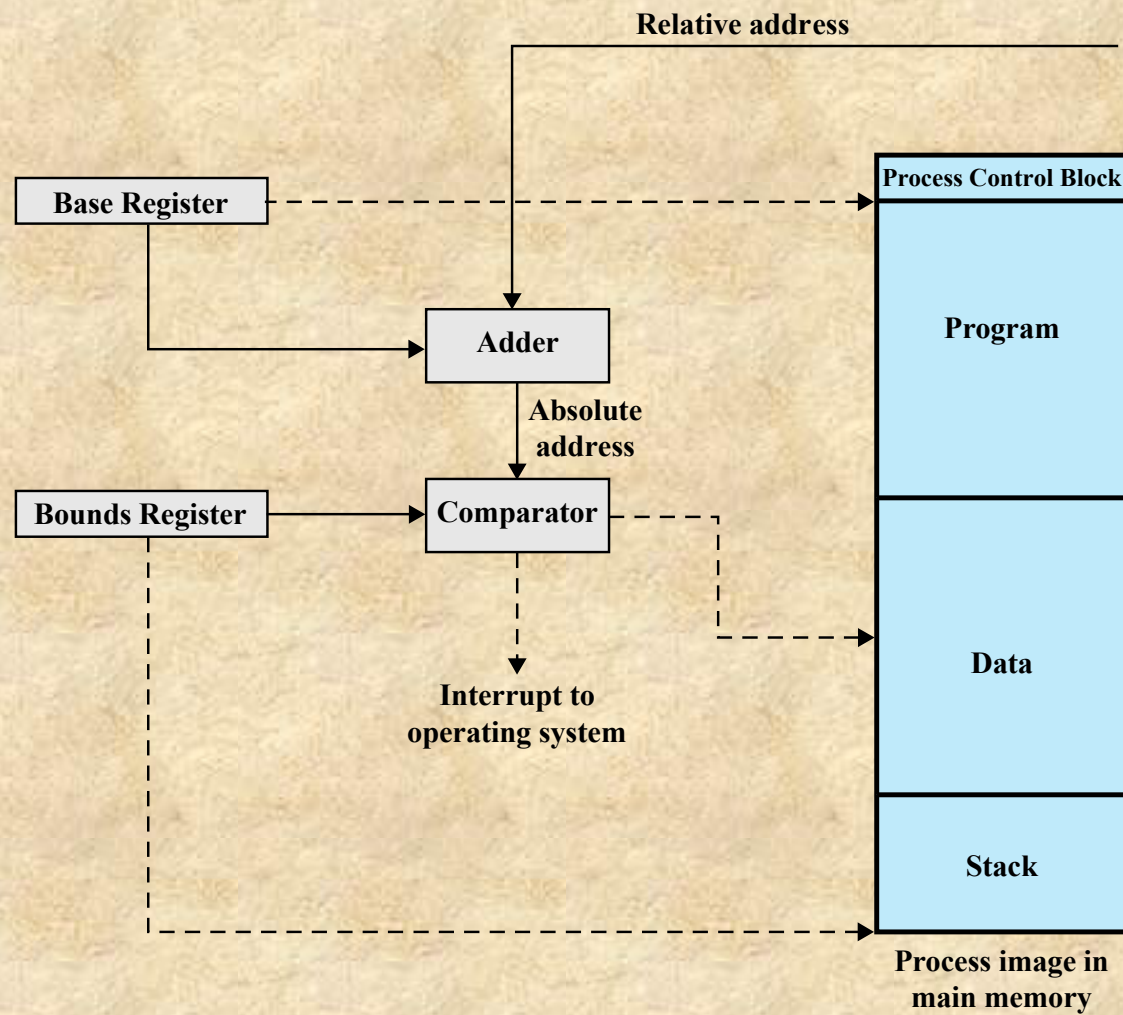


Figure 7.8 Hardware Support for Relocation



- 2.1. In fixed partitioning (equal size partitions) what kind of fragmentation do we have?
- 2.2. In fixed partitioning (unequal size partitions) what kind of fragmentation do we have?
- 2.3. Same question re dynamic partitioning?
- 2.4 We have a paging scheme where page size $S=256$ and address are 23 bit (yes! it's not a typo). How many bit for a page number, how many bit for an offset, how many pages do we have in the logical address space, how many bit for a page number?
- 2.5 Practice with placement algorithms for dynamic partitioning!



Module 2



(Simple) Paging

Exe of Process is split into (Logical) Pages

Main Memory (MM) is split into Frames

A page of a process is mapped to a frame of MM

Page Table maps Page Numbers onto Frame Numbers

One Page Table per process

All of a process in MM unless it is swapped out or exiting

All of a process in MM or All of a process out of MM

Program Size $\leq$ MainMemorySize or

ProgramSize $\leq$ Available MainMemorySize

Paging

- Partition memory into equal fixed-size chunks that are relatively small
- Process is also divided into small fixed-size chunks of the same size

Pages

- Chunks of a process

Frames

- Available chunks of memory

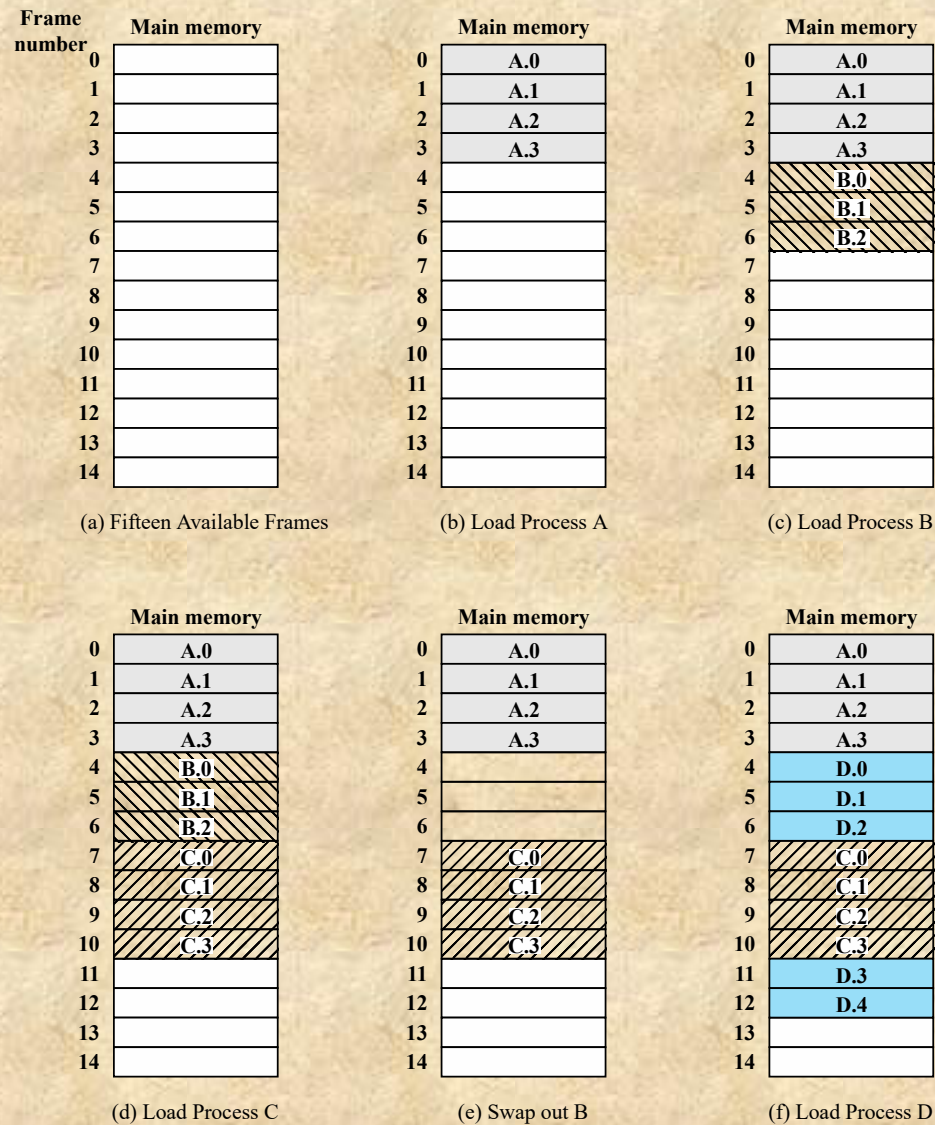


Figure 7.9 Assignment of Process Pages to Free Frames

Page Table

- Maintained by operating system for each process
- Contains the frame location for each page in the process
- Processor must know how to access for the current process
- Used by processor to produce a physical address

0	0
1	1
2	2
3	3

**Process A
page table**

0	—
1	—
2	—

**Process B
page table**

0	7
1	8
2	9
3	10

**Process C
page table**

0	4
1	5
2	6
3	11
4	12

**Process D
page table**

13
14

**Free frame
list**

Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

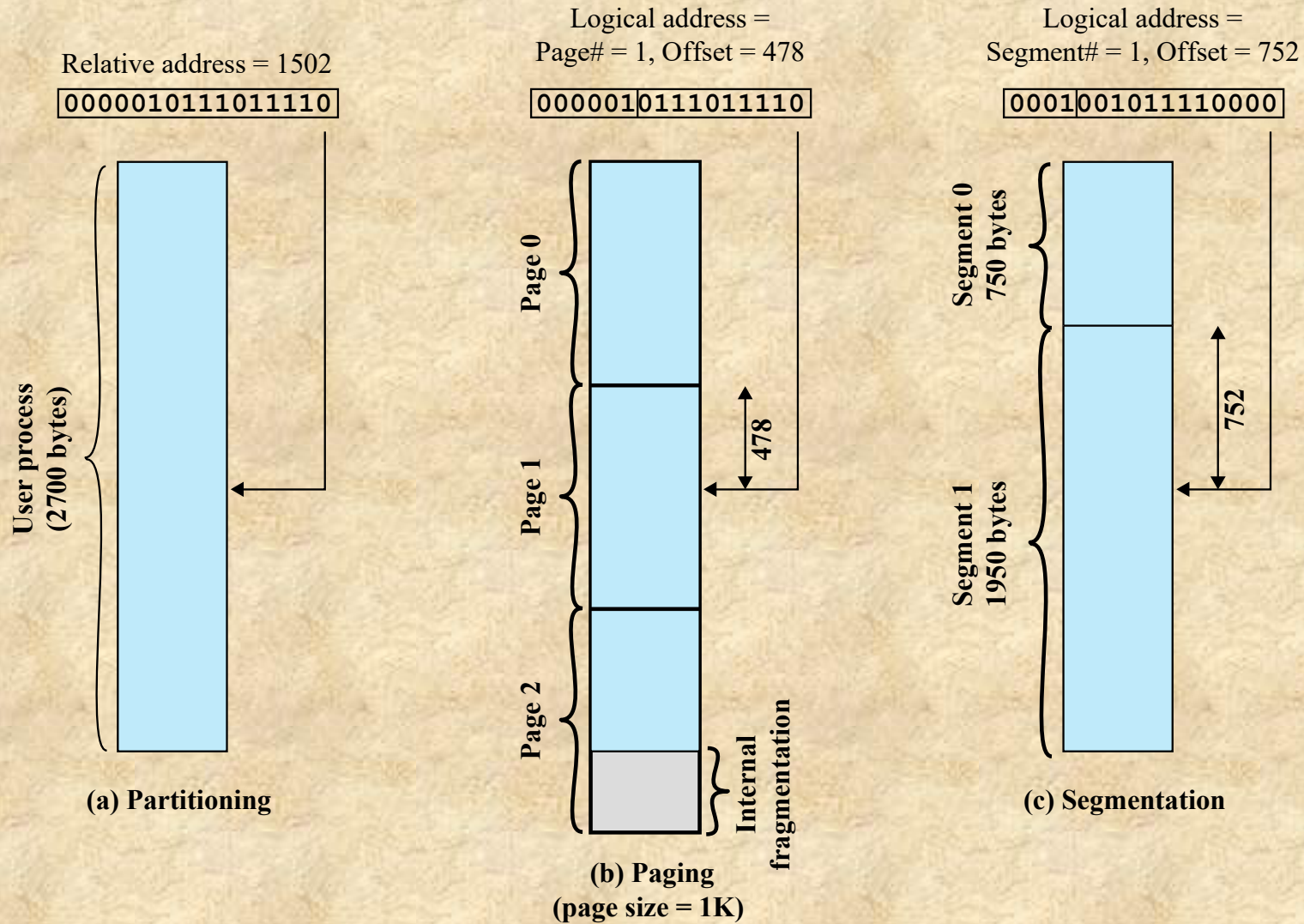
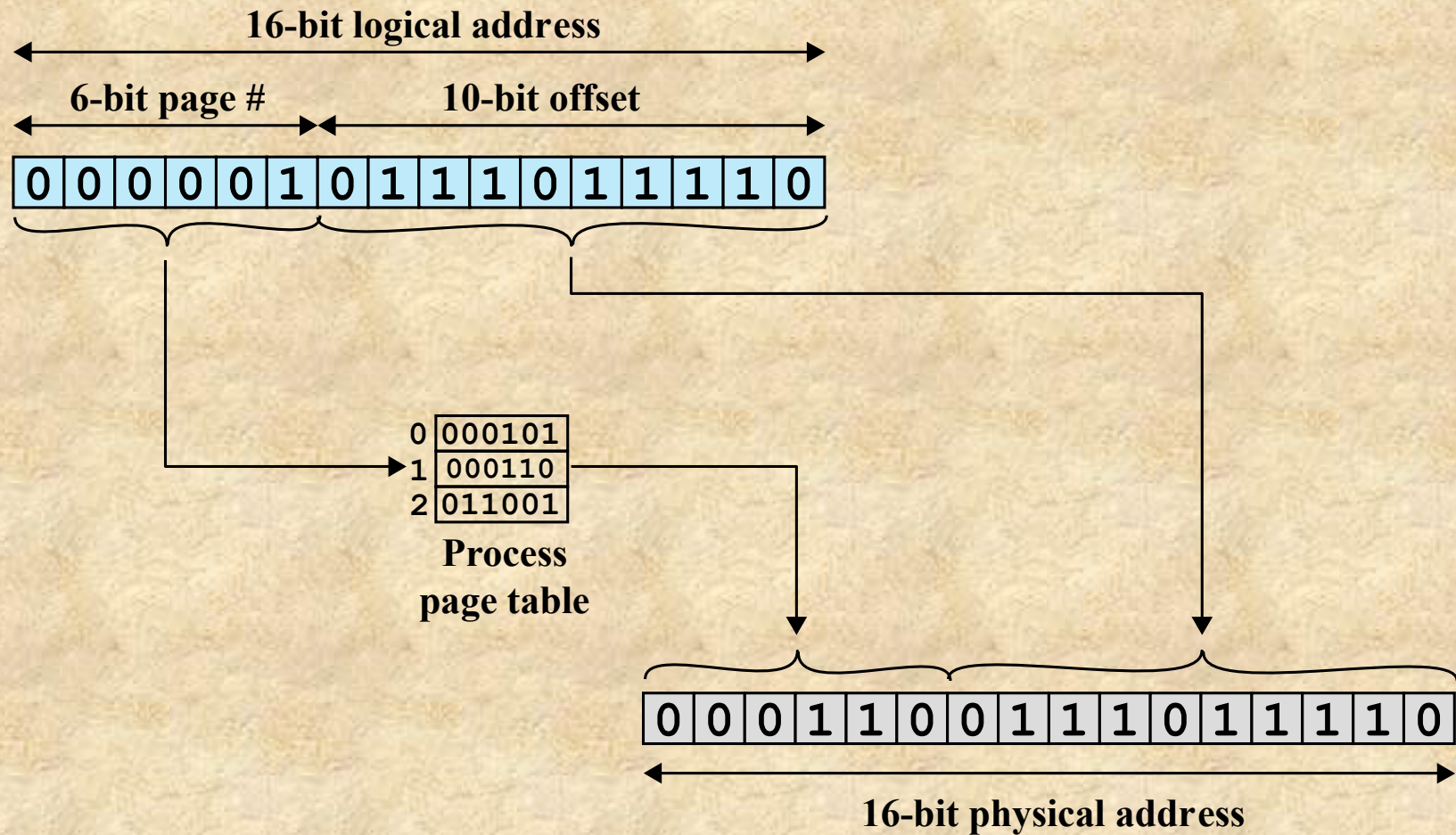
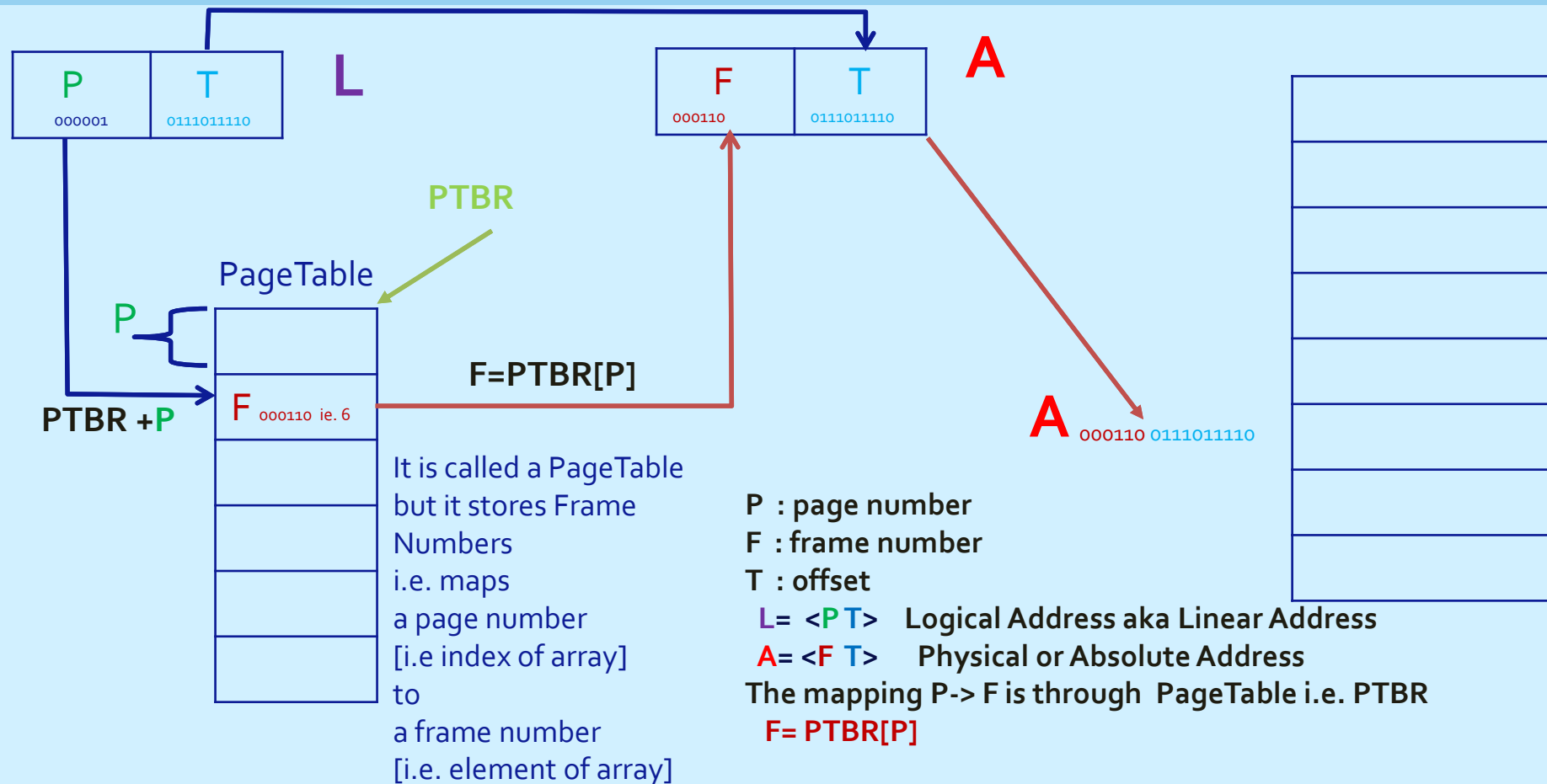


Figure 7.11 Logical Addresses



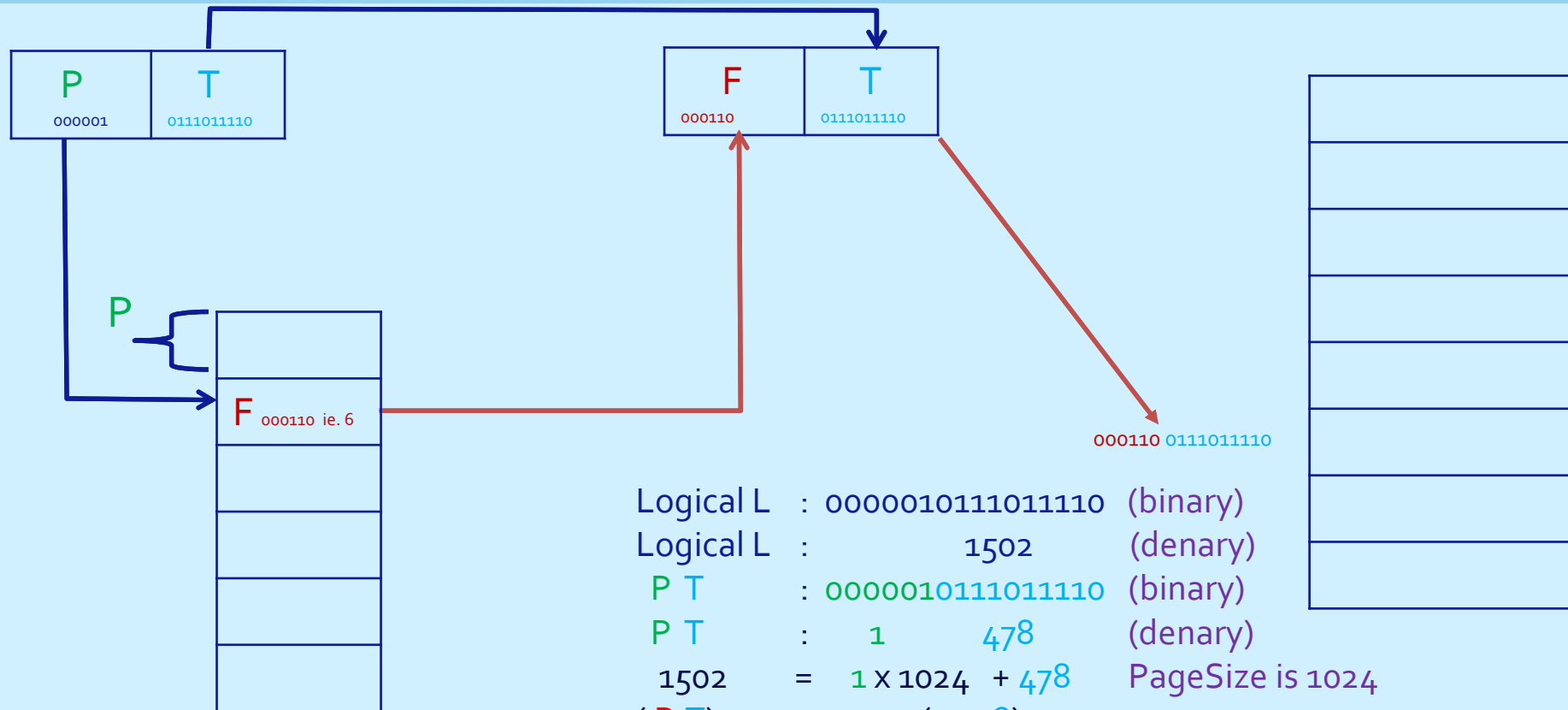
(a) Paging

Figure 7.12 Examples of Logical-to-Physical Address Translation



User Program uses **L** (logical address)

CPU uses **A** (Absolute/Physical) address



Logical L : 0000010111011110 (binary)
 Logical L : 1502 (denary)
 P T : 0000010111011110 (binary)
 P T : 1 478 (denary)
 $1502 = 1 \times 1024 + 478$ PageSize is 1024
 (P, T) : (1, 478)
 (F, T) : (6, 478) PTBR[1] is a 6
 $6622 = 6 \times 1024 + 478$
 (in denary): 6 478
 F T : 0001100111011110
 Physical A : 6622 (denary)
 Physical A : 0001100111011110 (binary)

P : page number (6bit) range : $0-63 = 2^{6-1}$
 F : frame number (unless stated, assume P's)
 T : offset (10bit) range: $0 \dots 1023$
 (if offset is 10bit, and its range is $0 \dots 1023$
 page size is 1024 . Also $2^{10} = 1024$)



Input is L in logical address space. What is A?

Intel Architectures:

1. Retrieve PTBR

(done once at load time)

2. Logical Address:

3. Access Page Table

(map P to an F)

4. Form Physical Address:

5. Memory Access

CR₃ stores PTBR location in MM

i.e. $PTBR = M[CR_3]$

S and R are known in advance

$PTBR = M[CR_3]$

(note that PTBR value gets stored in a register)

$L = \langle P T \rangle$ (generate P,T from L)

$F = PTBR[P]$ i.e. $F = M[PTBR + P]$

(Subject o: $F = *(PTBR + P)$; PTRB is available in a register)

$A = \langle F T \rangle$

$M[A]$

From **L** to **$M[A]$**

we need 2 memory accesses i.e.

$M[PTBR + P]$ and $M[A]$

($M[CR_3]$ is done once when process is (re) started; not for every memory reference!)



Logical and Physical address space are 32-bit. $S=R=4096B$. Process 20 has the page table of length 16 as shown below left. We use hex for logical addresses and physical addresses. Contents of PTBR in hexadecimal.

(a) For $L=0X0000AABC$. What is A ? Give answer in hex and in binary.

(b) What is the A for $L=0X00000000$? Give answer in hex only.

PTBR

0X0000F
 0X00001
 0X00002
 0X00003
 0X00004
 0X000F1
 0X000A1
 0X00AF2
 0X00FA1
 0X0FAFA
 0XA0001
 0XABCDE
 0XABBBB
 0XBAAAA
 0XBAAAB
 0X0000A

(a) $L=0X0000AABC$. $S=4096$ i.e. $\lg(S)=12$. Offset T is thus 12 bit / 3hex digit and therefore pagenumber P is $32-12=20$ bit / 5hex digit

$L=<0000AABC> = <0000A ABC> \rightarrow P=0x0000A, T=0XABC$

$F=PTBR[P]=PTBR[0x0000A]=PTBR[10]=0XA0001 \quad T=0XABC$

$A=(F,T)=(0XA0001,0XABC) = (\text{spaces below for readability only})$

$<A0001 ABC> = <1010 0000 0000 0000 0001 1010 1011 1100>=$

$0XA0001ABC=0b101000000000000000001101010111100$

(b) $L=0X00000000 \Rightarrow L=<00000000>=<00000 000> \rightarrow P=0X00000, T=0X000$

$F=PTBR[P]=PTBR[0x00000]=PTBR[0] = 0X0000F$

$A=(F,T)=(0X0000F,0X000) = <0000F 000> = 0X0000F000$

Segmentation

- A program can be subdivided into segments
 - May vary in length
 - There is a maximum length
- Addressing consists of two parts:
 - Segment number
 - An offset
- Similar to dynamic partitioning
- Eliminates internal fragmentation

Segmentation

- Usually visible
- Provided as a convenience for organizing programs and data
- Typically the programmer will assign programs and data to different segments
- For purposes of modular programming the program or data may be further broken down into multiple segments
 - The principal inconvenience of this service is that the programmer must be aware of the maximum segment size limitation

Address Translation

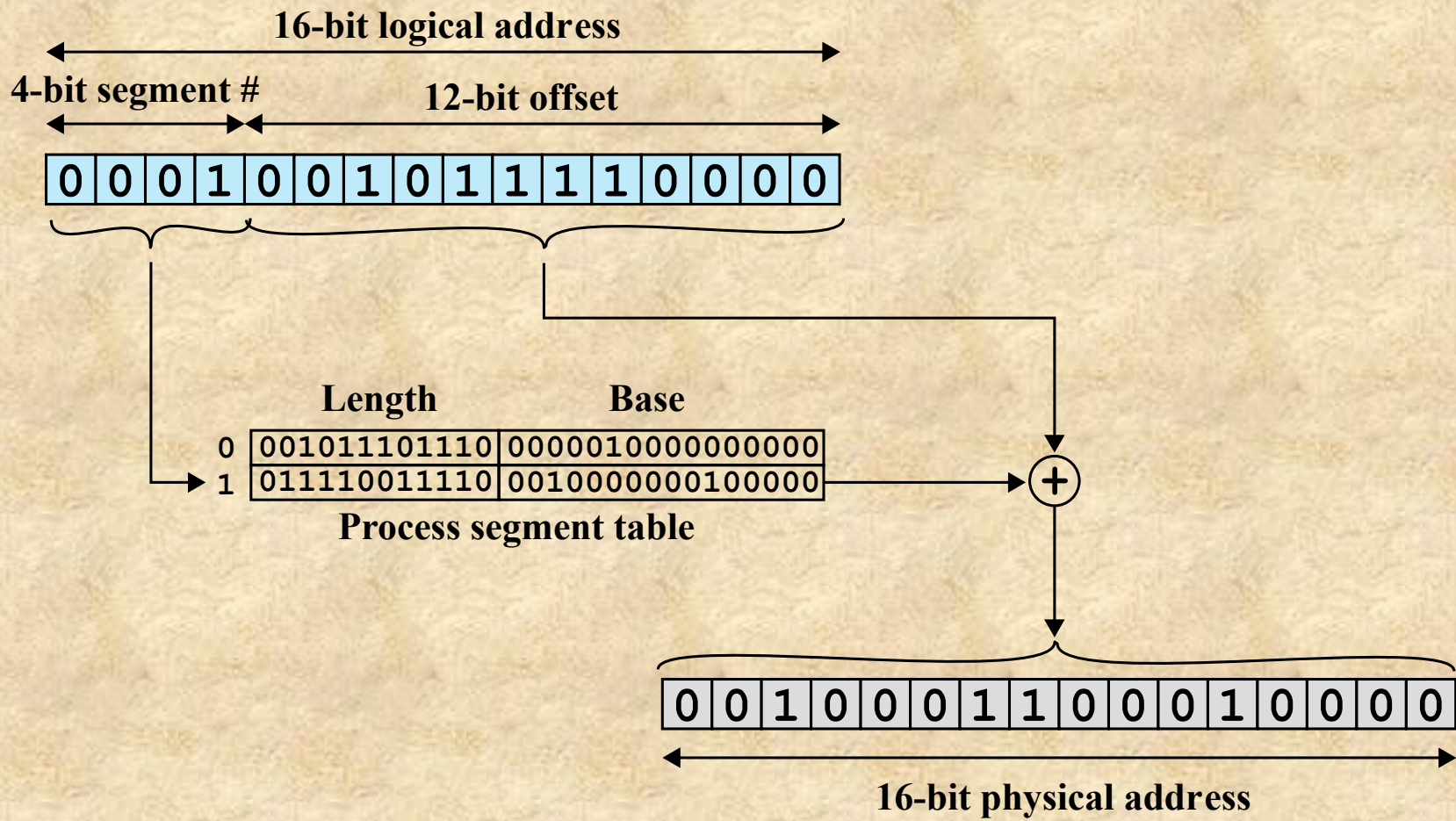
- Another consequence of unequal size segments is that there is no simple relationship between logical addresses and physical addresses
- The following steps are needed for address translation:

Extract the segment number as the leftmost n bits of the logical address

Use the segment number as an index into the process segment table to find the starting physical address of the segment

Compare the offset, expressed in the rightmost m bits, to the length of the segment. If the offset is greater than or equal to the length, the address is invalid

The desired physical address is the sum of the starting physical address of the segment plus the offset



(b) Segmentation

Figure 7.12 Examples of Logical-to-Physical Address Translation



We have a Logical Address $L=23456$ and page size is $S=10000$ (Frame size $R=S$)
 A page number x is mapped to frame number $x+3$ (i.e. $F=P+3$)
 Find Physical Address A

Page size $S=10000$ implies Offset T range $T: 0 \dots 9999$ i.e. $0 \leq T < S$

From previous page: $L = P \times S + T$ L :dividend, S divisor
 P, T are quotient and remainder

From elementary school $23456 = P \times 10000 + T \Rightarrow P = 23456 / 10000 = 2, T = 23456 - 2 \times 10000 = 3456$

$$F = P + 3 = 2 + 3 = 5$$

$$A = F \times R + T, \quad F=5, R=S=10000, T=3456 \Rightarrow$$

$$A = 5 \times 10000 + 3456 = 53456 \Rightarrow$$

P maps to F through page table and $F=P+3$ as given

If S is not a power of two

you need divisions and multiplications

If S is a power of two

you need bit manipulations only!

THE POWER ... of ... a POWER of TWO!

Practice 4



4.1. Using the previous page information map $L=45678$ into an A.

4.2 With paging does a process suffer from internal or external fragmentation or both? Explain.

Summary

- Memory management requirements
 - Relocation
 - Protection
 - Sharing
 - Logical organization
 - Physical organization
- Paging
- Memory partitioning
 - Fixed partitioning
 - Dynamic partitioning
 - Buddy system
 - Relocation
- Segmentation

Definitions



1. **Compile:** from .c into .s (assembly)

2. **Assemble** from .s into .o (object code)

3. **Link** from (multiple) .o into .exe (executable)

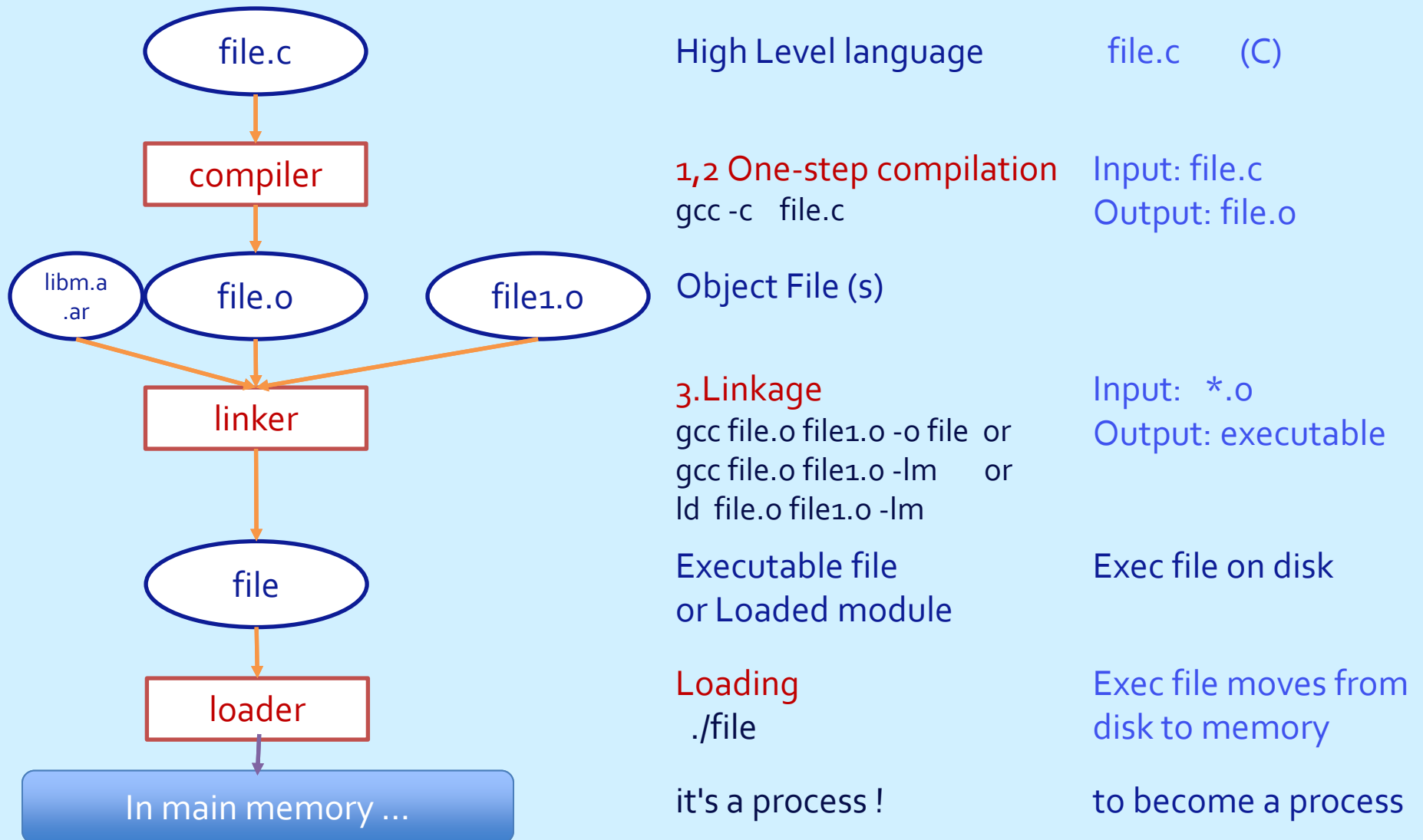
4. **Load** from executable into running in memory

Most people when they say
they expect **'compilation'**
compile+assemble+link

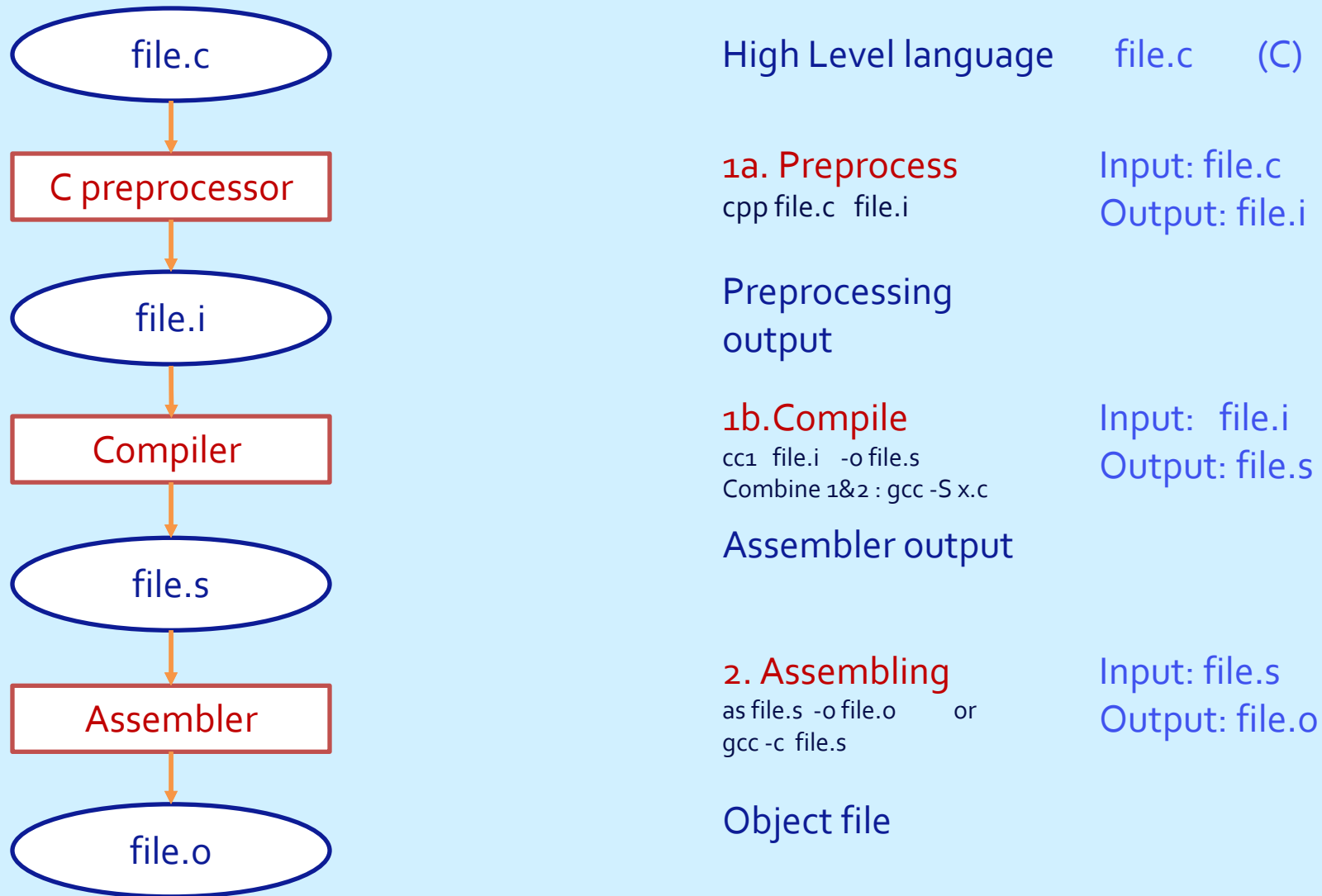
Note: MS uses .exe form executables; Linux executable do not use a suffix: they can be arbitrary file names.

Note: **Compilation hides a preprocessing step.**

Compilation + memory



Compilation(+hidden steps)



NJIT

A white curved line that starts under the 'N' and sweeps upwards and to the right, ending under the 'T'.

New Jersey's Science &
Technology University

THE EDGE IN KNOWLEDGE

Alex. Gerbessiotis

CS Department