INFO 324 Operating Systems II

Ahmad FAOUR

Virtual Memory (2)

Multilevel Paging and Performance

Since each level is stored as a separate table in memory, converting a logical address to a physical one with a three-level page table may take four memory accesses. Why?

Descriptive Example

- Consider a system with:
  - 64 bits architecture
  - Page size 4 KB
  - Physical memory 512 MB
- What about multi-level page tables?
  - Q:How many levels is required?
  - A: 6 levels
  - Q: how much memory references for each address translation?
  - A: 6 memory accesses
- SO, system works slowly 6 times down in comparison with no-paging system!!!!

Descriptive Example

- Imagine if we can do the inverse. How?
  - Associate one PTE to each physical frame
  - On 512 MB physical memory with frame size 4KB
  - We need $2^{29}/2^{12} = 2^{17}$ entries
  - Each entry takes 16 bytes
  - The page table size is decreased to ~2MB
  - The page table is global for all processes in memory

- **=➔ Inverted Page Table**

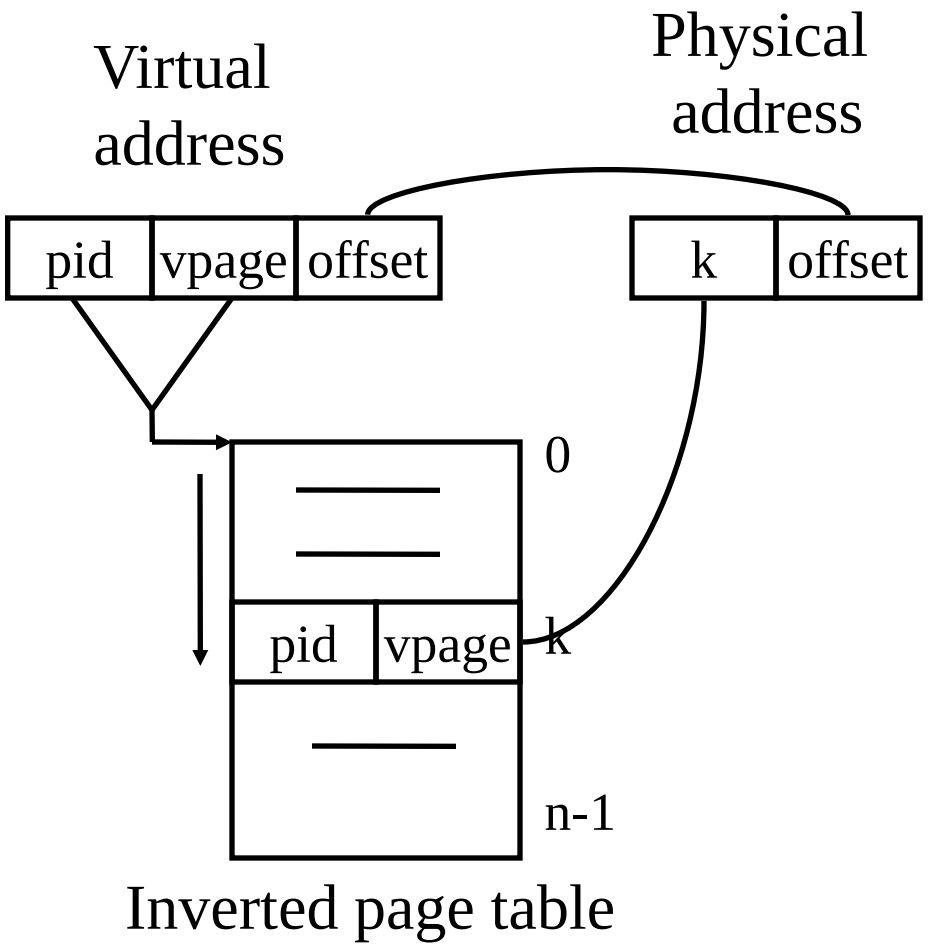
Inverted Page Tables

An inverted page table

- Has one entry for every resident memory page
- Roughly speaking, one for each frame of memory
- Records which page is in that frame
- Can not be indexed by page number

So how can we search an inverted page table on a TLB miss fault?

Inverted Page Tables



Main idea
- One PTE for each physical page frame
- Hash (Vpage, pid) to Ppage#
- Trade off space for time

Pros
- Small page table for large address space

Cons
- Lookup is difficult
- Overhead of managing hash chains, etc

Inverted Page Tables

- Given a page number (from a faulting address), do we exhaustively search all entries to find its mapping?

- How the lookup is made?
  - Linear Inverted Page Table
  - Hashed Inverted Page Table

Linear Inverted Page Table

- One entry for each physical frame indexed from 0 to number of frames -1
- Since the table is shared between all processes, each entry contains the pid of the process and and virtual page number
- For each VPN we must search the entire table
- 2^{17} possible search → Inefficient

Inverted Page Tables

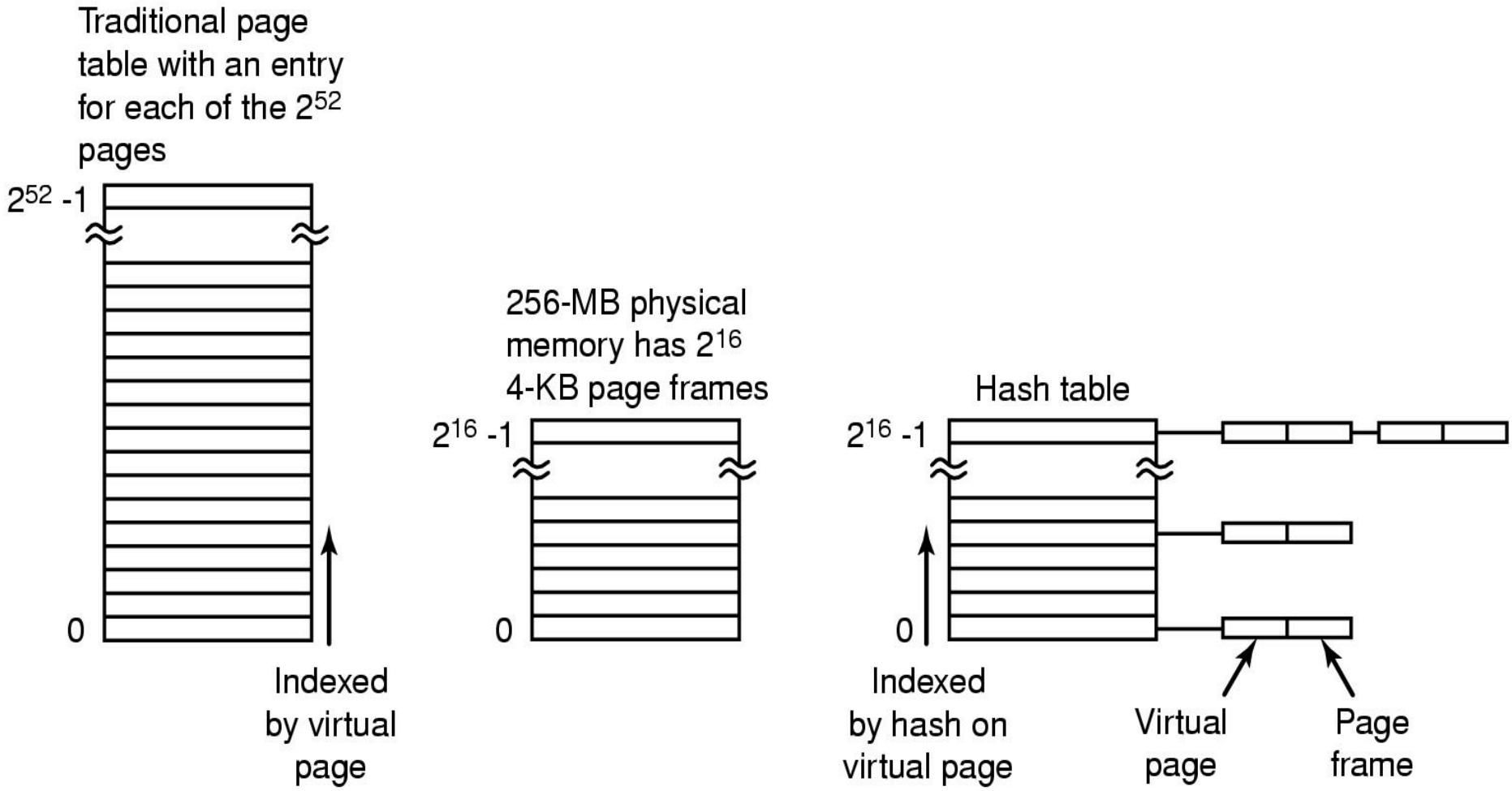
Given a page number (from a faulting address), do we exhaustively search all entries to find its mapping?

- No, that's too slow!
- A hash table could allow fast access given a page number
- O(1) lookup time with a good hash function

Hash Inverted Page Table

Hash Table:Data structure for associating a key with a value

- Apply hash function to key to produce a *hash*
- Hash is a number that is used as an array index
- Each element of the array can be a linked list of entries (to handle collisions)
- The list must be searched to find the required entry for the key (entry's key matches the search key)
- With a good hash function the average list length will be short


Traditional page table with an entry for each of the 2^{52} pages
2^{52} -1
0
Indexed by virtual page
256-MB physical memory has 2^{16} 4-KB page frames
2^{16} -1
0
Hash table
2^{16} -1
0
Indexed by hash on virtual page
Virtual page
Page frame

Which Page Table Design is Best?

The best choice depends on CPU architecture

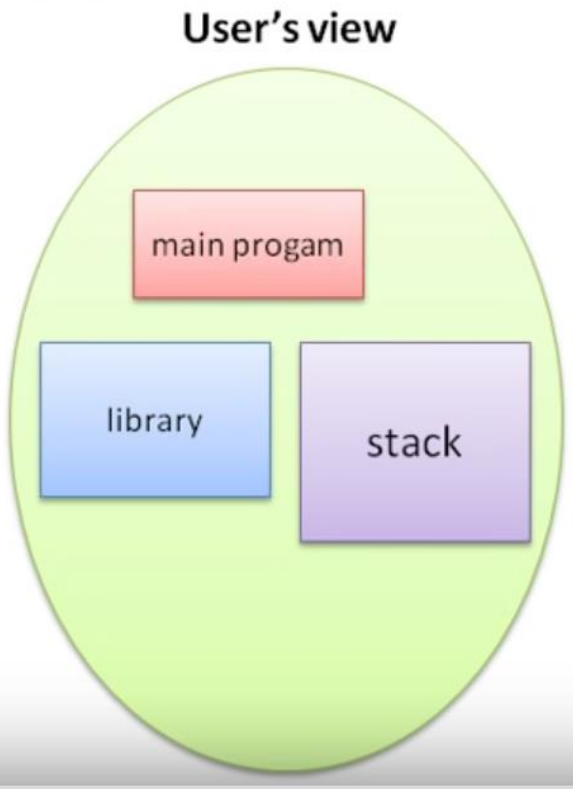
64 bit systems need inverted page tables

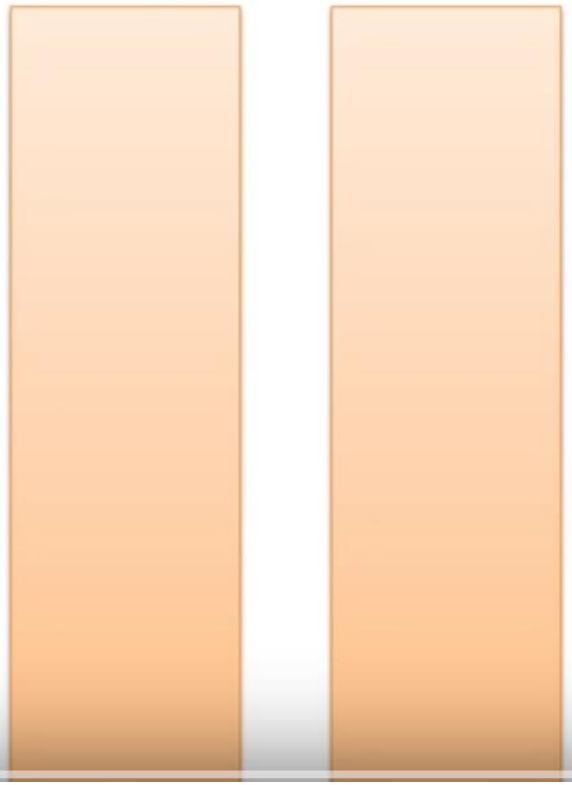
Some systems use a combination of regular page tables together with segmentation (later)

SEGMENTATION

Paging

User's view of memory is not same as actual memory.

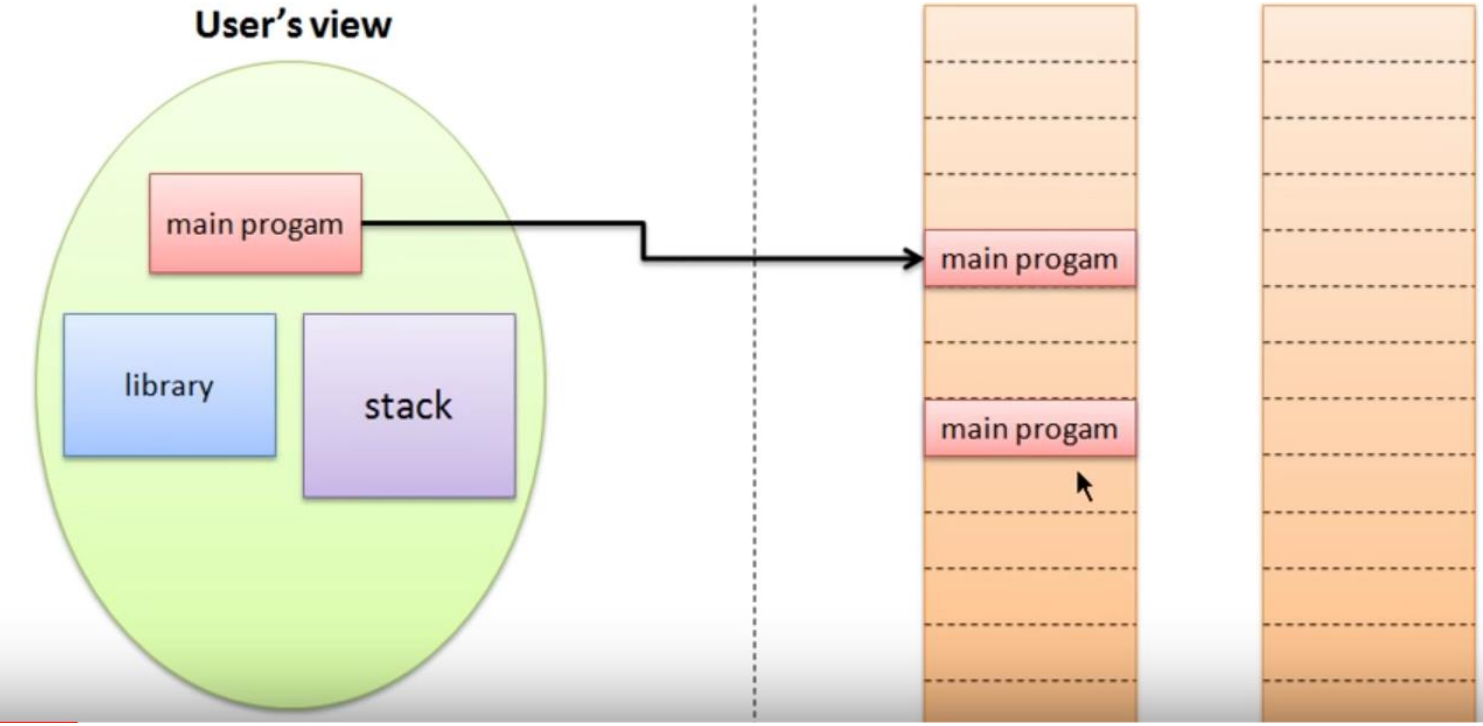




SEGMENTATION

Paging

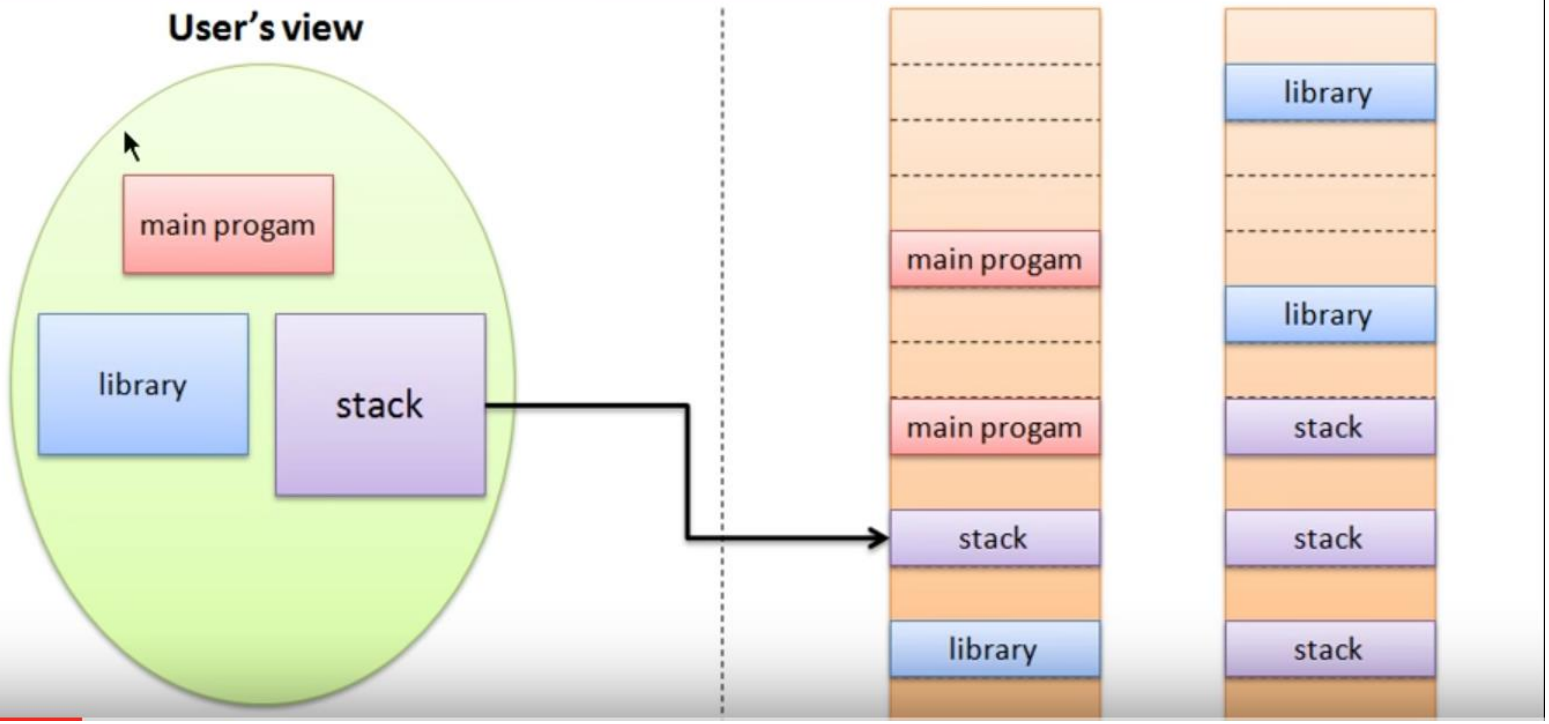
User's view of memory is not same as actual memory.



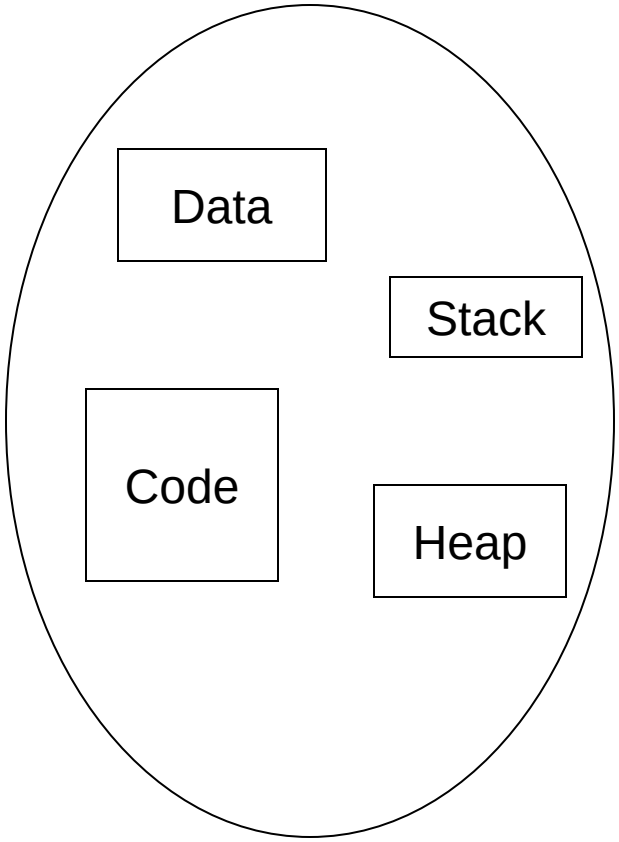
SEGMENTATION

Paging

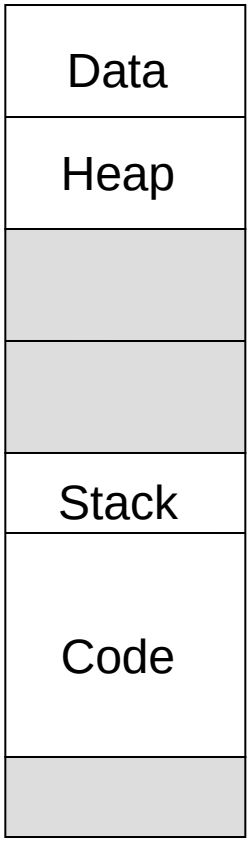
User's view of memory is not same as actual memory.



Segmentation



user view of memory



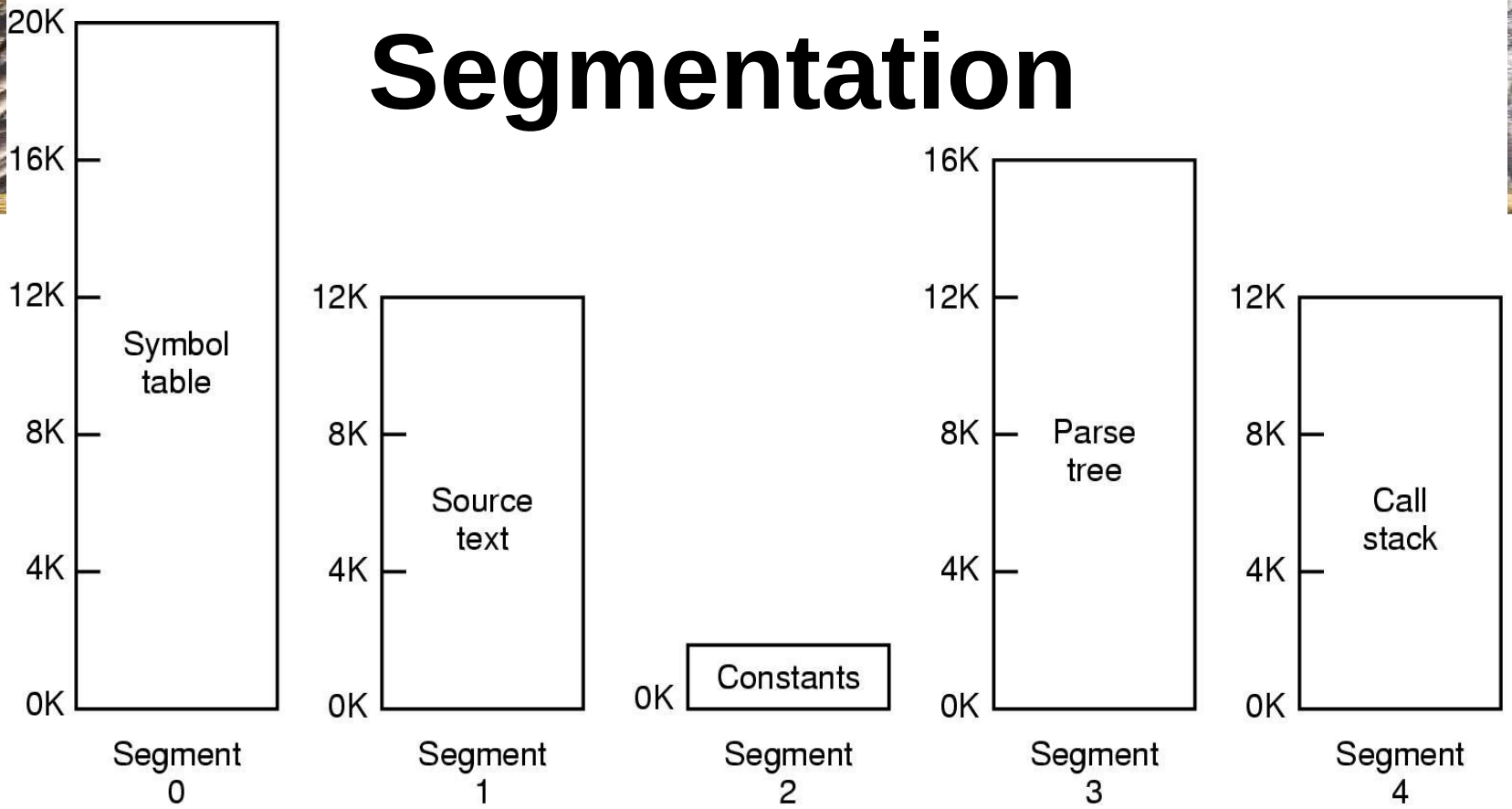
logical memory space

- Each page is only a piece of memory but has no meaning.
- A program is a collection of segments such as:
  - main program,
  - procedure,
  - function,
  - global variables,
  - common block,
  - stack,
  - symbol table

SEGMENTATION

- Where paging uses one continuous sequence of all virtual addresses from 0 to the maximum needed for the process.
- Segmentation is an alternate scheme that uses multiple separate address spaces for various segments of a program.
- A segment is a logical entity of which the programmer is aware. Examples include a procedure, an array, a stack, etc.
- Segmentation allows each segment to have different lengths and to change during execution.

Segmentation



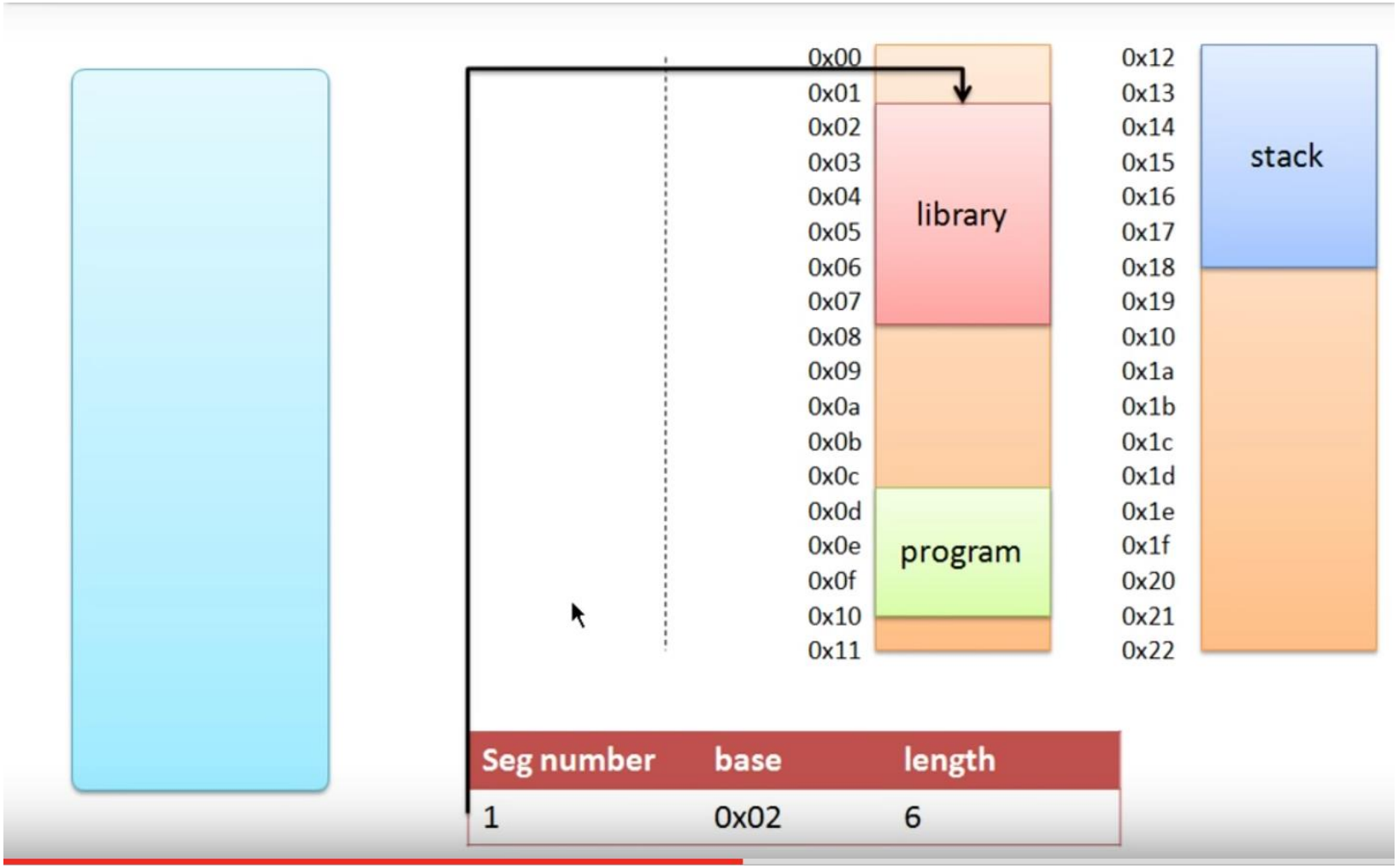
- Allows each table to grow or shrink, independently
- To specify an address in segmented memory, the program must supply a two-part address: (n,w) where n is the segment number and w is the address within the segment, starting at 0 in each segment.
- Changing the size of 1 procedure, does not require changing the starting address of any other procedure - a great time saver.

SEGMENTATION

Segmentation

- It is similar to dynamic partitioning as variable size segments
- The difference is that, the segments need not be contiguous
- It eliminates internal fragmentation
- But like dynamic , it suffers from external fragmentation

SEGMENTATION



SEGMENTATION

Segmentation

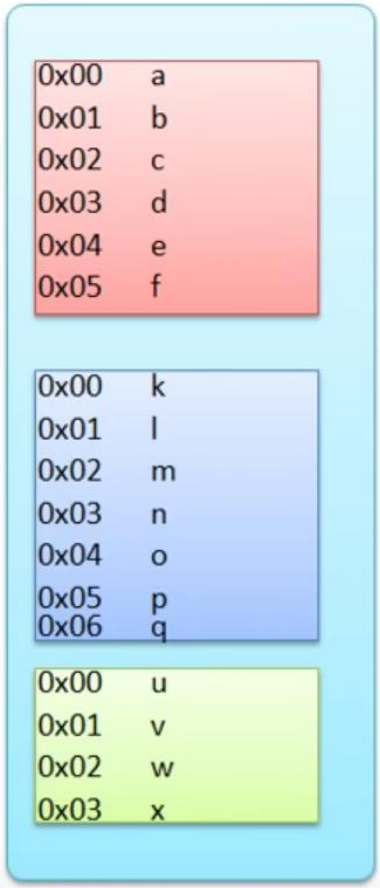
- There is no simple relationship between logical and physical address , which was pretty clear in paging
- Segment table keeps record of the segments as done by paging table before
- Logical address (segmentNo,Offset)

SEGMENTATION

Translation In segmentation

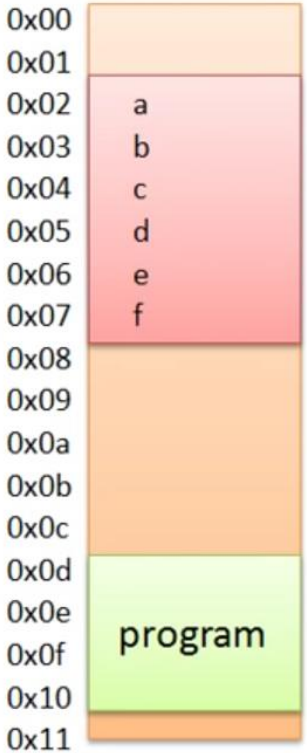
- If address is n+m bits
- Then n refers to segment no
- We use it to find the starting physical address by looking up in the segment table
- m refers to the offset
- Desired physical address is the sum of starting physical address and offset

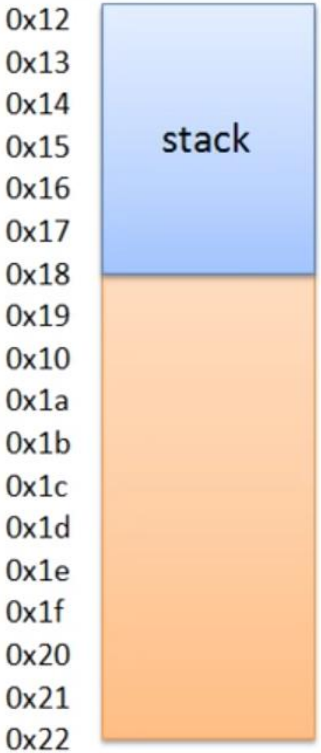
SEGMENTATION



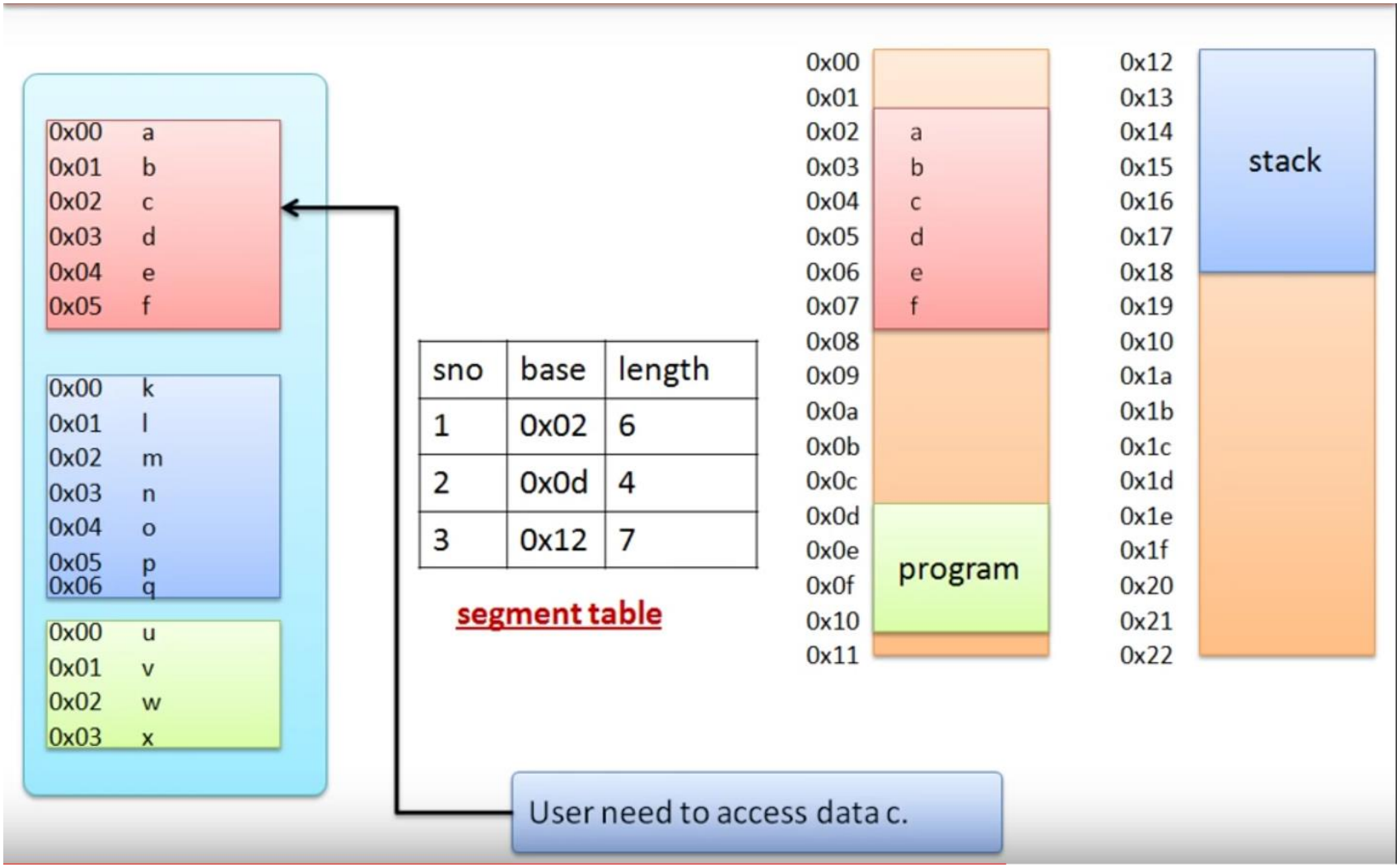
sno	base	length
1	0x02	6
2	0x0d	4
3	0x12	7

segment table

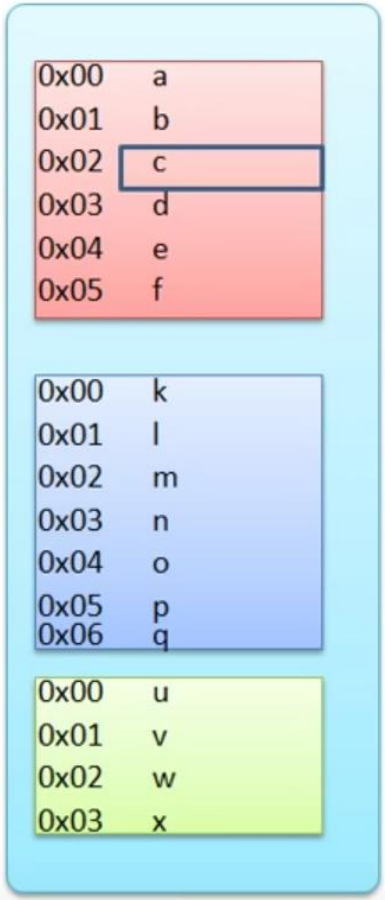




SEGMENTATION

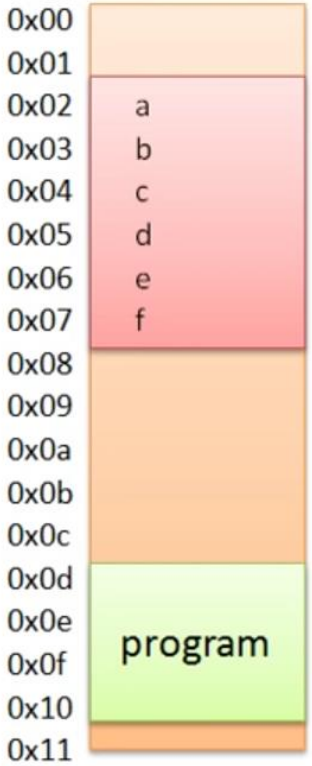


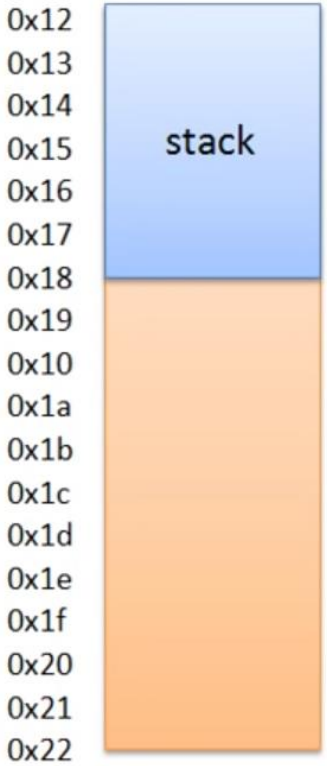
SEGMENTATION



sno	base	length
1	0x02	6
2	0x0d	4
3	0x12	7

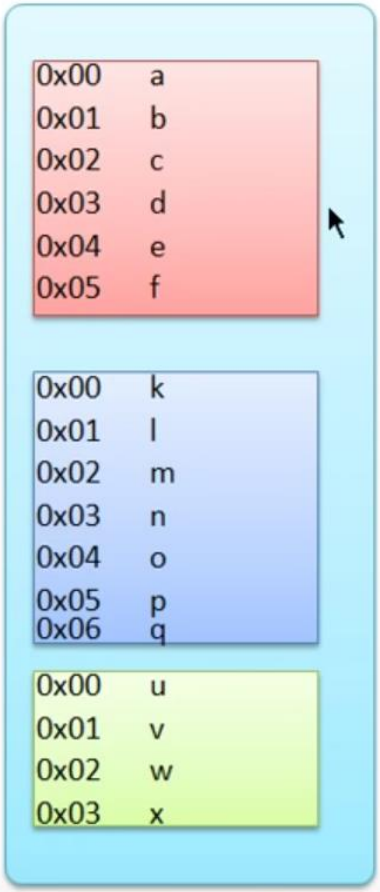
segment table





<Segment-number, Offset>

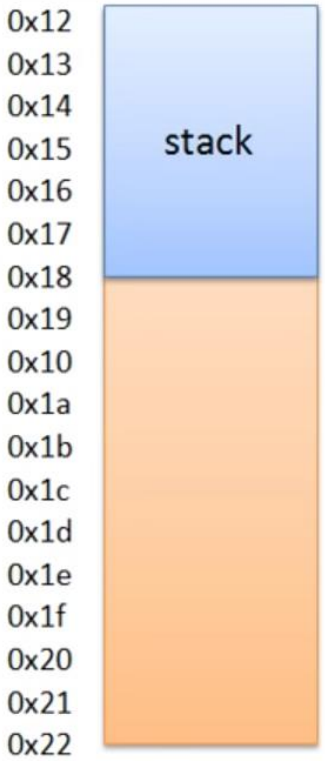
SEGMENTATION



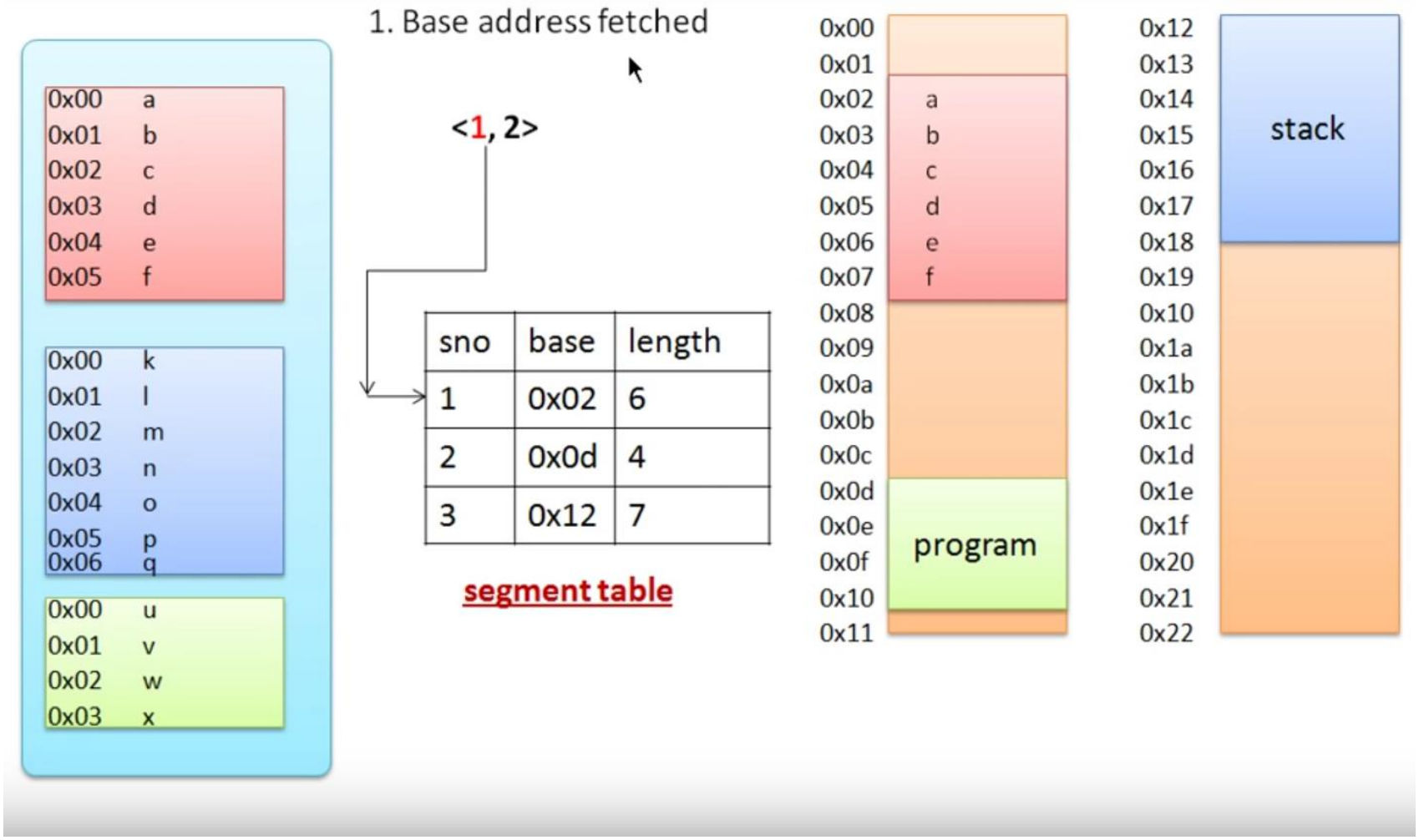
sno	base	length
1	0x02	6
2	0x0d	4
3	0x12	7

segment table

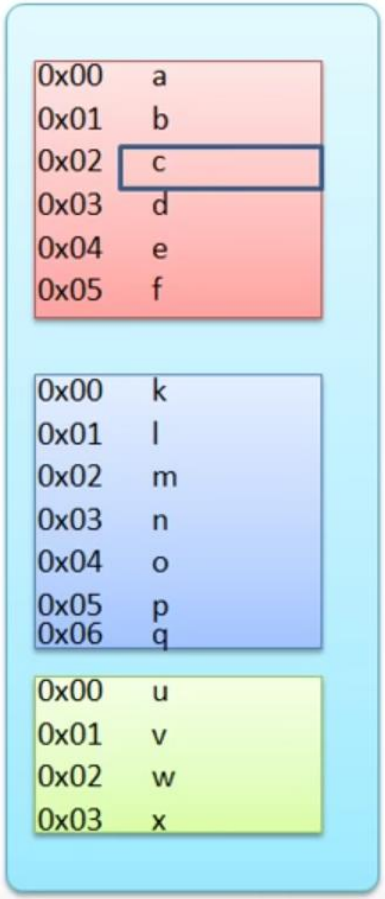
<1, 2>



SEGMENTATION



SEGMENTATION



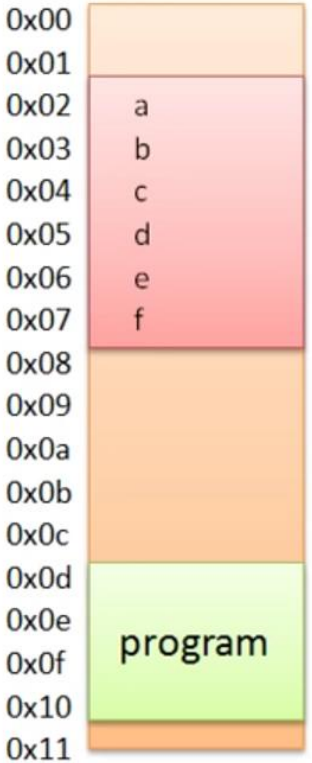
1. Base address fetched

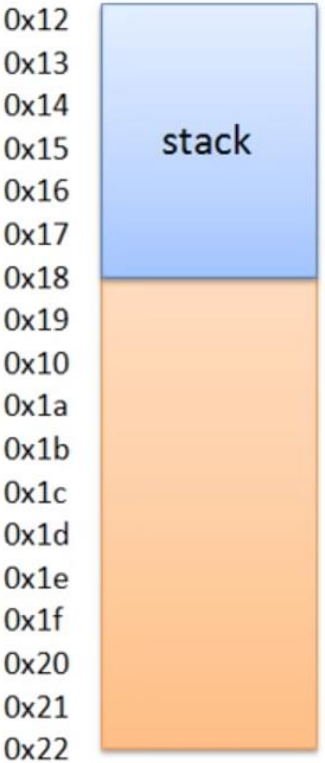
2. offset must be between 0 and limit of that segment.

<1, 2>

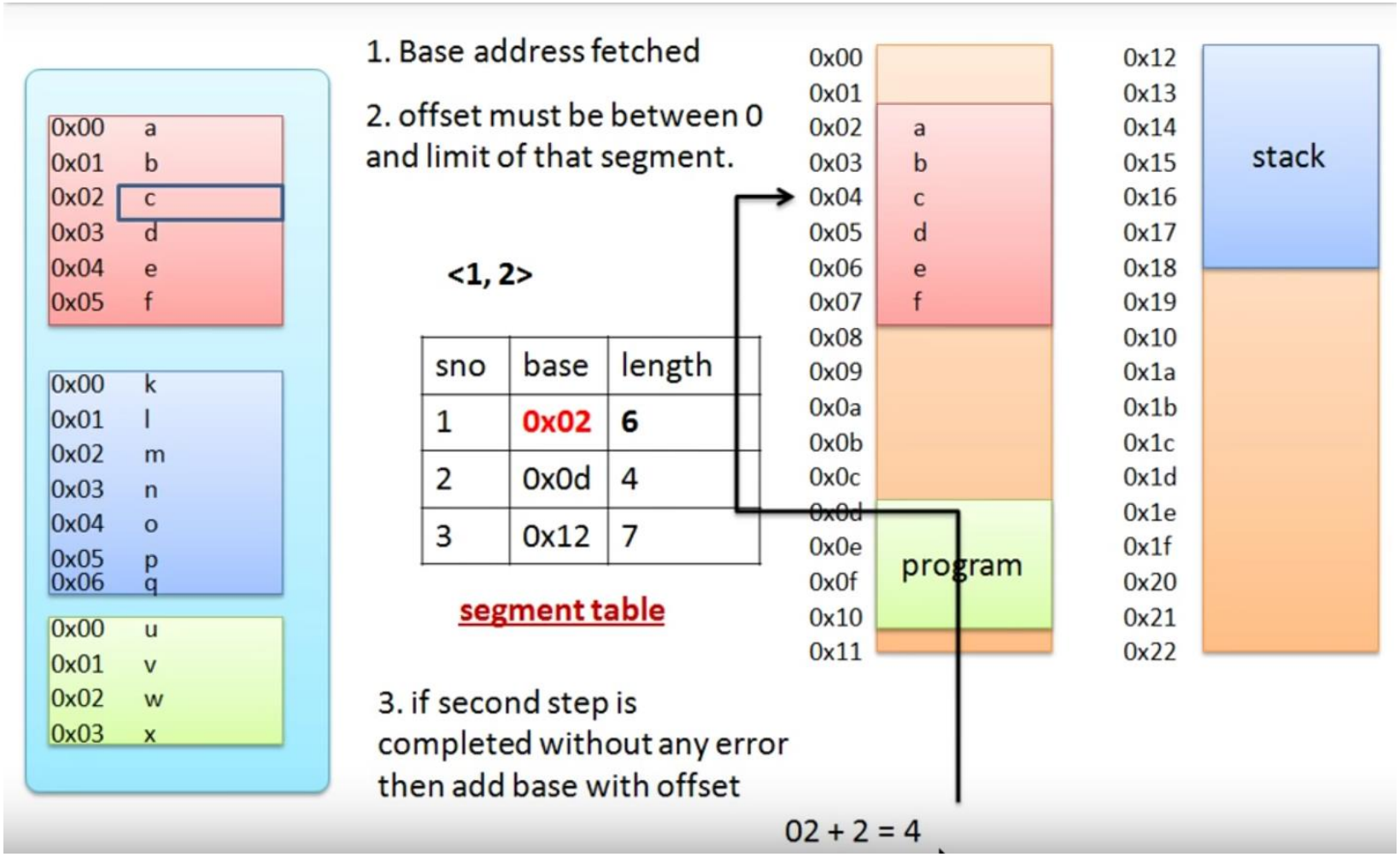
sno	base	length
1	0x02	6
2	0x0d	4
3	0x12	7

segment table





SEGMENTATION



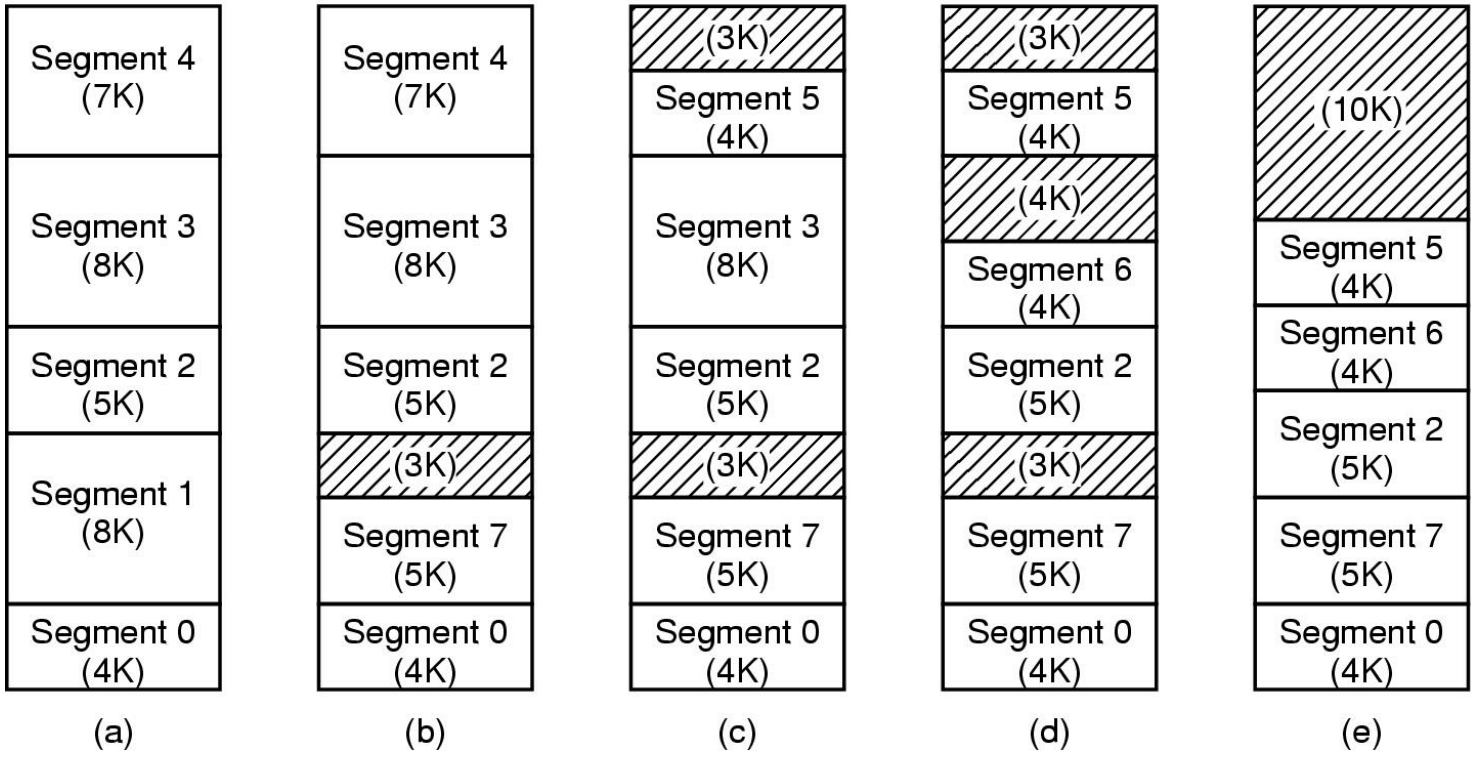
Segmentation Permits Sharing Procedures or Data between Several Processes

- A common example is the **shared library**, such as a large graphical library compiled into nearly every program on today's modern workstations.
- With segmentation, the library can be put in a segment and shared by multiple processes, avoiding the need to have the entire library in every process's address space.
- Since each segment contains a specific logical entity, the user can protect each appropriately (without concern for where boundaries are in the paging system): a procedure segment can be set execute but not read or write; an array can be specified read/write but not execute; etc. This is a great help in debugging.

Comparison of paging and segmentation

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Pure Segmentation

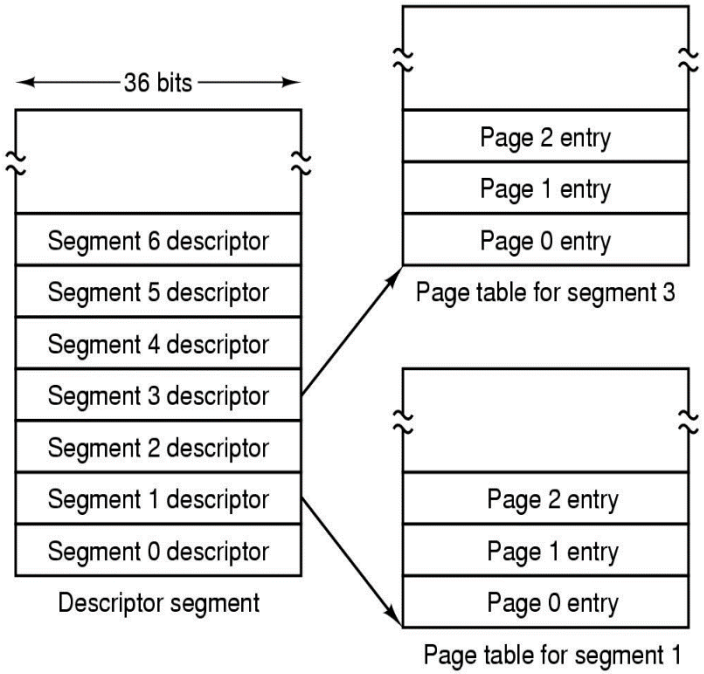


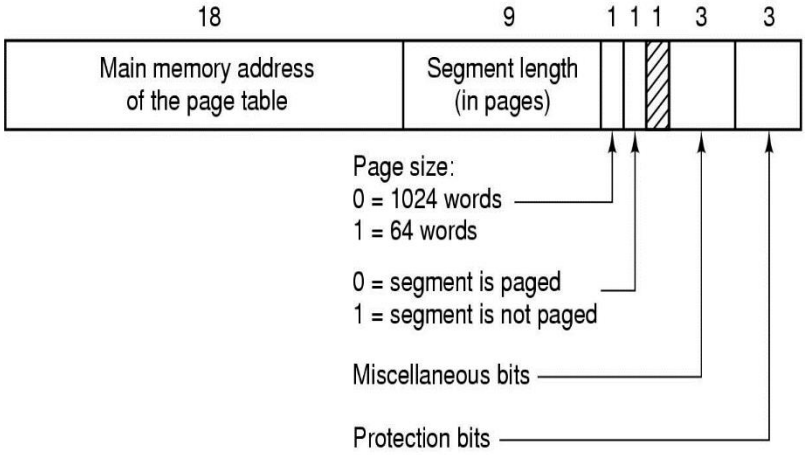
(a) Memory initially containing 5 segments of various sizes.

(b)-(d) Memory after various replacements: external fragmentation

(e) Removal of external fragmentation by compaction eliminates the wasted memory in holes.

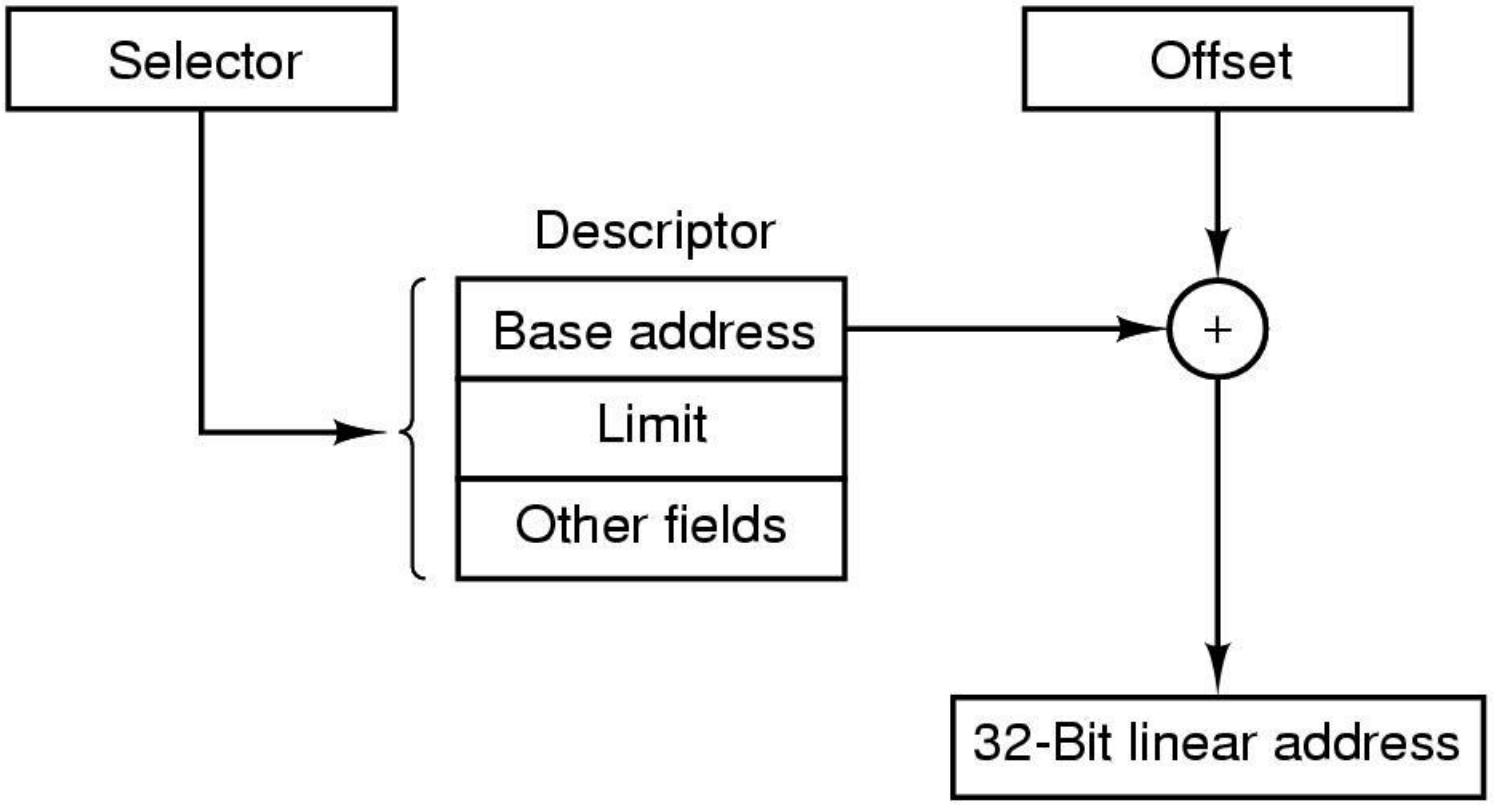
Segmentation with Paging: MULTICS (1)





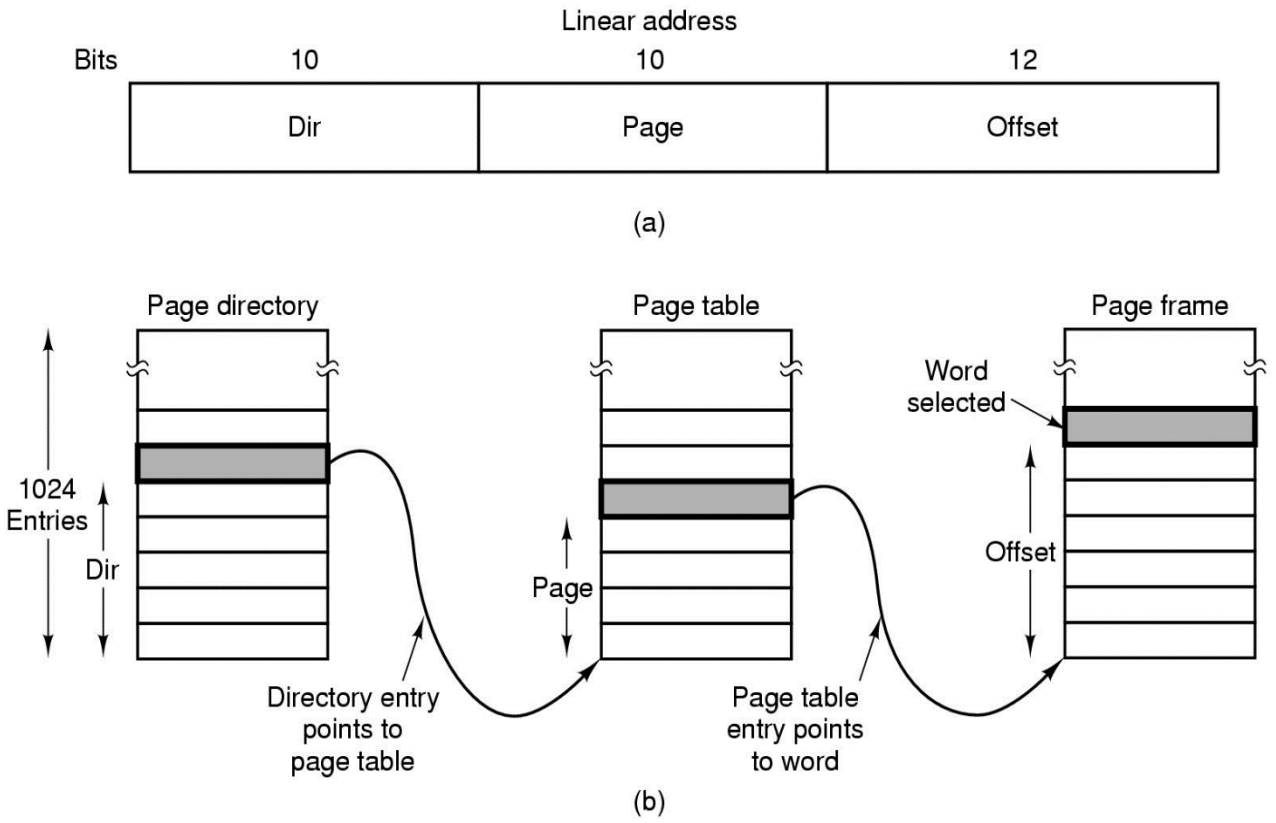
- Descriptor segment points to page tables
- Segment descriptor – numbers are field lengths

Segmentation with Paging: Pentium (3)



Conversion of a (selector, offset) pair to a linear address

Segmentation with Paging: Pentium (4)



Mapping of a linear address onto a physical address

Segmentation vs. Paging

Do we need to choose one or the other?

Why not use both together

- Paged segments
- Paging for memory allocation
- Segmentation for maintaining protection information at a coarse granularity
- Segmentation and paging in combination for translation