D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Subject Name: Operating Systems

Unit: 5 Unit Name: Memory Management

Faculty Name: Ms. Puja Padiya

Index

Unit No: 5 Unit Name: Memory Management

Lecture: Segmentation

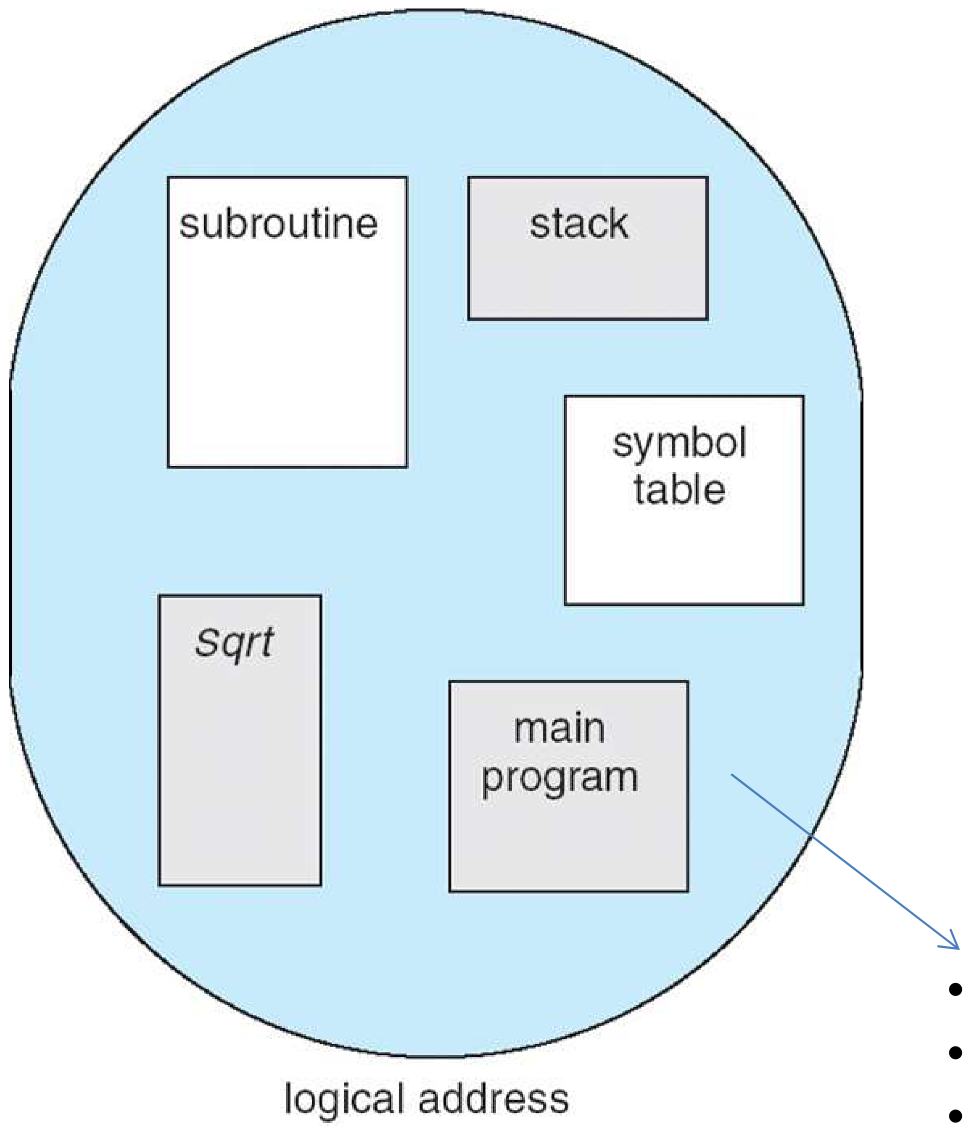
Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
  - A segment is a logical unit such as:

    main program
    procedure
    function
    method
    object
    local variables, global variables
    common block
    stack
    symbol table
    arrays

Compiler generates the segments
Loader assign the seg#

User’s View of a Program



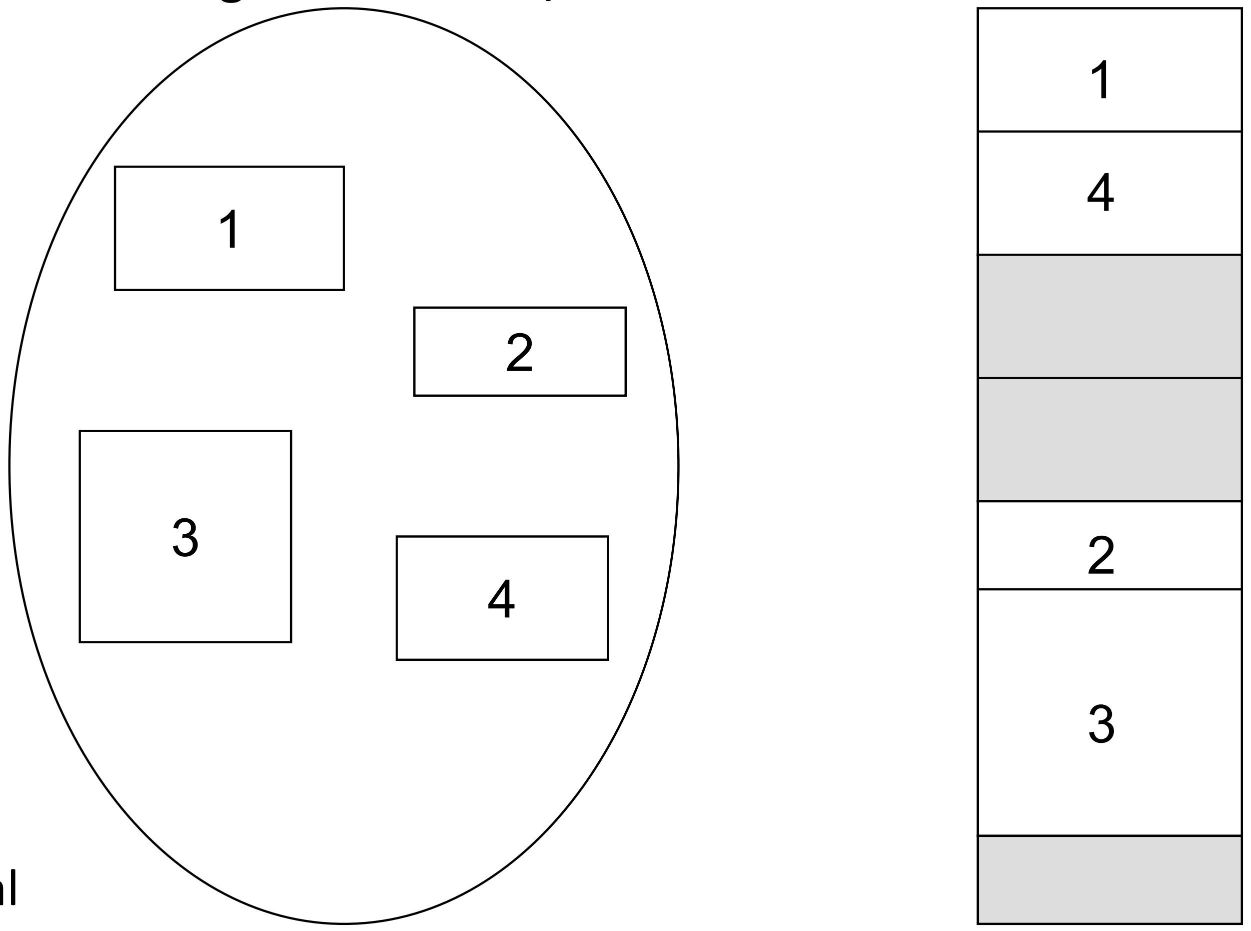
User specifies each address by two quantities
(a) Segment name
(b) Segment offset

Logical address contains the tuple
<segment#, offset>

- Variable size segments without order
- Length=> purpose of the program
- Elements are identified by offset

Logical View of Segmentation

Logical address <segment-number, offset>

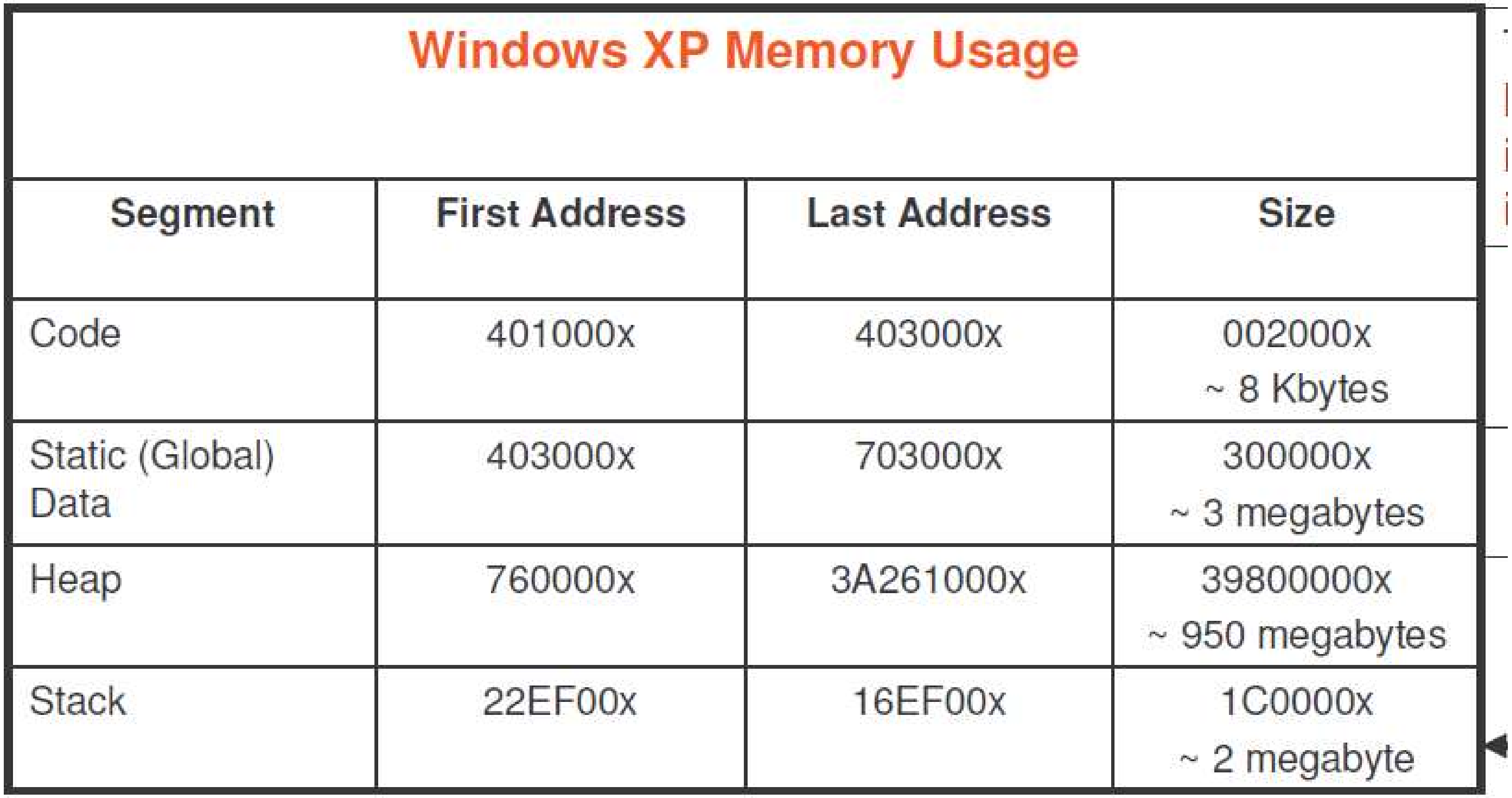


Logical address space

user space

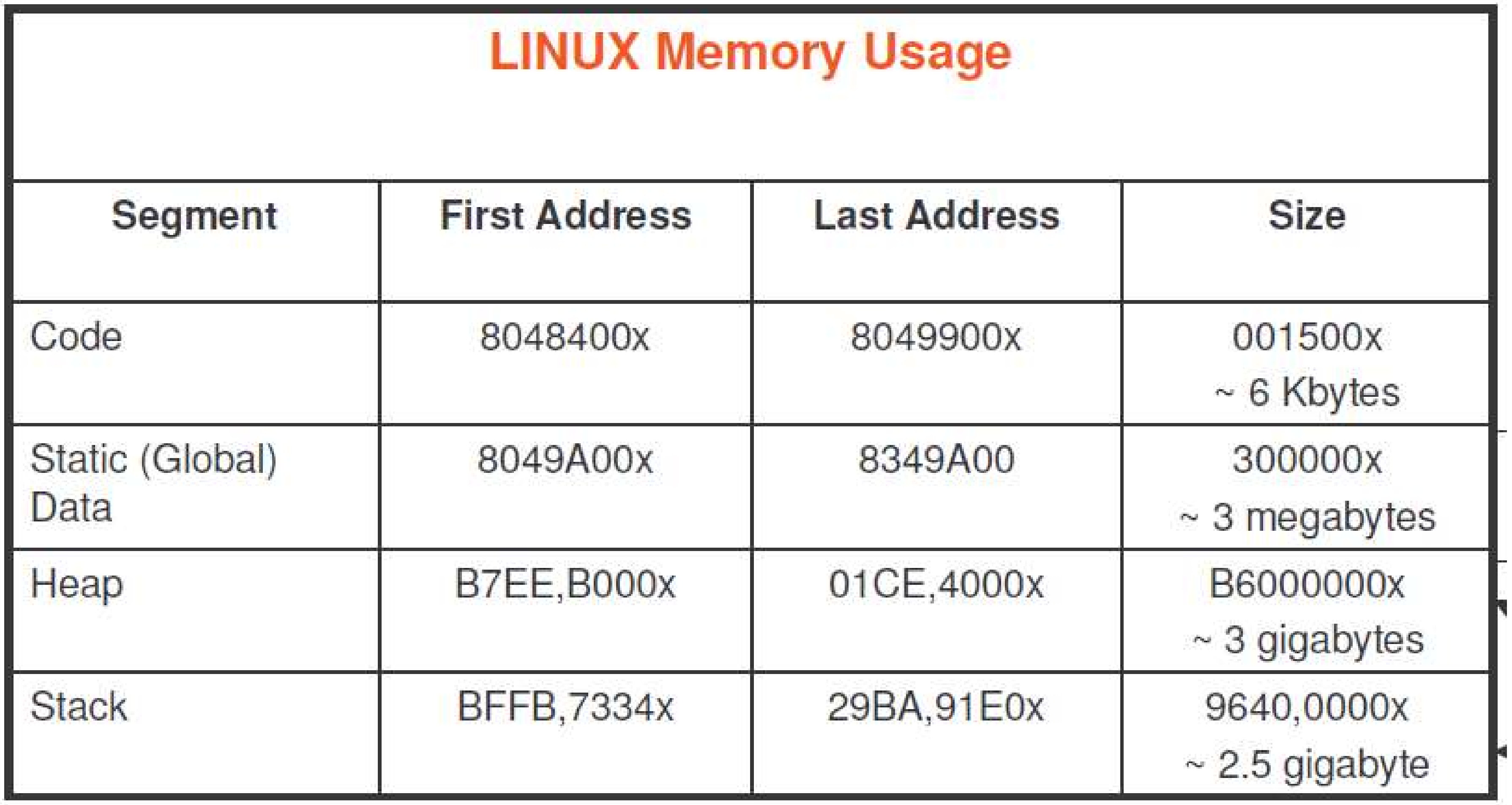
physical memory space

- Long term scheduler finds and allocates memory for all segments of a program
- Variable size partition scheme



Windows XP Memory Usage

Segment	First Address	Last Address	Size
Code	401000x	403000x	002000x ~ 8 Kbytes
Static (Global) Data	403000x	703000x	300000x ~ 3 megabytes
Heap	760000x	3A261000x	39800000x ~ 950 megabytes
Stack	22EF00x	16EF00x	1C0000x ~ 2 megabyte



LINUX Memory Usage

Segment	First Address	Last Address	Size
Code	8048400x	8049900x	001500x ~ 6 Kbytes
Static (Global) Data	8049A00x	8349A00	300000x ~ 3 megabytes
Heap	B7EE,B000x	01CE,4000x	B6000000x ~ 3 gigabytes
Stack	BFFB,7334x	29BA,91E0x	9640,0000x ~ 2.5 gigabyte

Memory image

```
0x08048368 <main+0>:    55                 push   %ebp
0x08048369 <main+1>:    89 e5              mov    %esp,%ebp
0x0804836b <main+3>:    83 ec 08           sub    $0x8,%esp
0x0804836e <main+6>:    83 e4 f0           and    $0xfffffff0,%esp
0x08048371 <main+9>:    b8 00 00 00 00     mov    $0x0,%eax
0x08048376 <main+14>:   83 c0 0f           add    $0xf,%eax
0x08048379 <main+17>:   83 c0 0f           add    $0xf,%eax
0x0804837c <main+20>:   c1 e8 04           shr    $0x4,%eax
0x0804837f <main+23>:   c1 e0 04           shl    $0x4,%eax
0x08048382 <main+26>:   29 c4              sub    %eax,%esp
0x08048384 <main+28>:   83 ec 0c           sub    $0xc,%esp
0x08048387 <main+31>:   68 c0 84 04 08     push   $0x80484c0
0x0804838c <main+36>:   e8 1f ff ff ff     call   0x80482b0
0x08048391 <main+41>:   83 c4 10           add    $0x10,%esp
0x08048394 <main+44>:   e8 02 00 00 00     call   0x804839b <b>
```

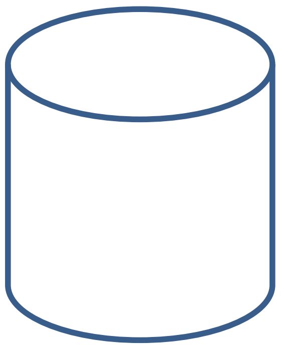
```
void  b();
void  c();
int   main( )
{
   printf( "Hello from main\n");
   b();
}
// This routine reads the opcodes from memory and prints them out.
void   b()
 {
    char  *moving;

    for ( moving = (char *)(&main); moving < (char *)(&c); moving++ )
       printf( "Addr = 0x%x, Value = %2x\n", (int)(moving), 255 & (int)*moving );
 }
 void   c()
 {
 }
```

```
0x0804839b <b+0>:     55                    push   %ebp
0x0804839c <b+1>:     89 e5                 mov    %esp,%ebp
0x0804839e <b+3>:     83 ec 08              sub    $0x8,%esp
0x080483a1 <b+6>:     c7 45 fc 68 83 04 08  movl   $0x8048368,0xfffffffc(%ebp)
0x080483a8 <b+13>:    81 7d fc d9 83 04 08  cmpl   $0x80483d9,0xfffffffc(%ebp)
0x080483af <b+20>:    73 26                 jae    0x80483d7 <b+60>
0x080483b1 <b+22>:    83 ec 04              sub    $0x4,%esp
0x080483b4 <b+25>:    8b 45 fc              mov    0xfffffffc(%ebp),%eax
0x080483b7 <b+28>:    0f be 00              movsbl (%eax),%eax
0x080483ba <b+31>:    25 ff 00 00 00        and    $0xff,%eax
```

Executable file and virtual address

a.out



Symbol table	
Name	address
SQR	0
SUM	4

Virtual address space

Paging view

```
0          Load     0
4          ADD      4
```

Segmentation view

```
<CODE, 0>          Load      <ST,0>
<CODE, 2>          ADD       <ST,4>
```

Segmentation Architecture

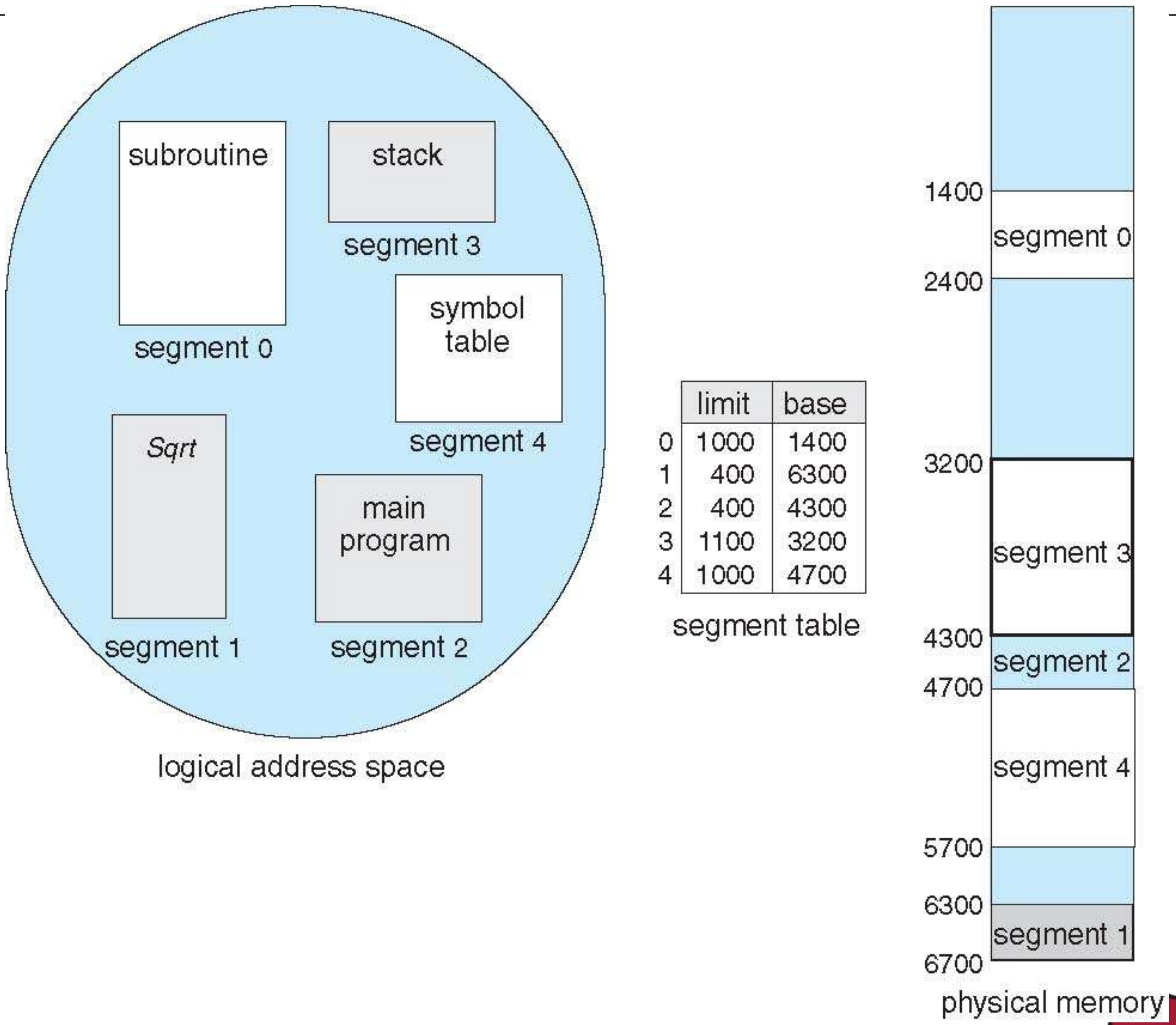
- Logical address consists of a two tuple:

  <segment-number, offset>

- **Segment table** – maps two-dimensional logical address to physical address;
- Each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < **STLR**

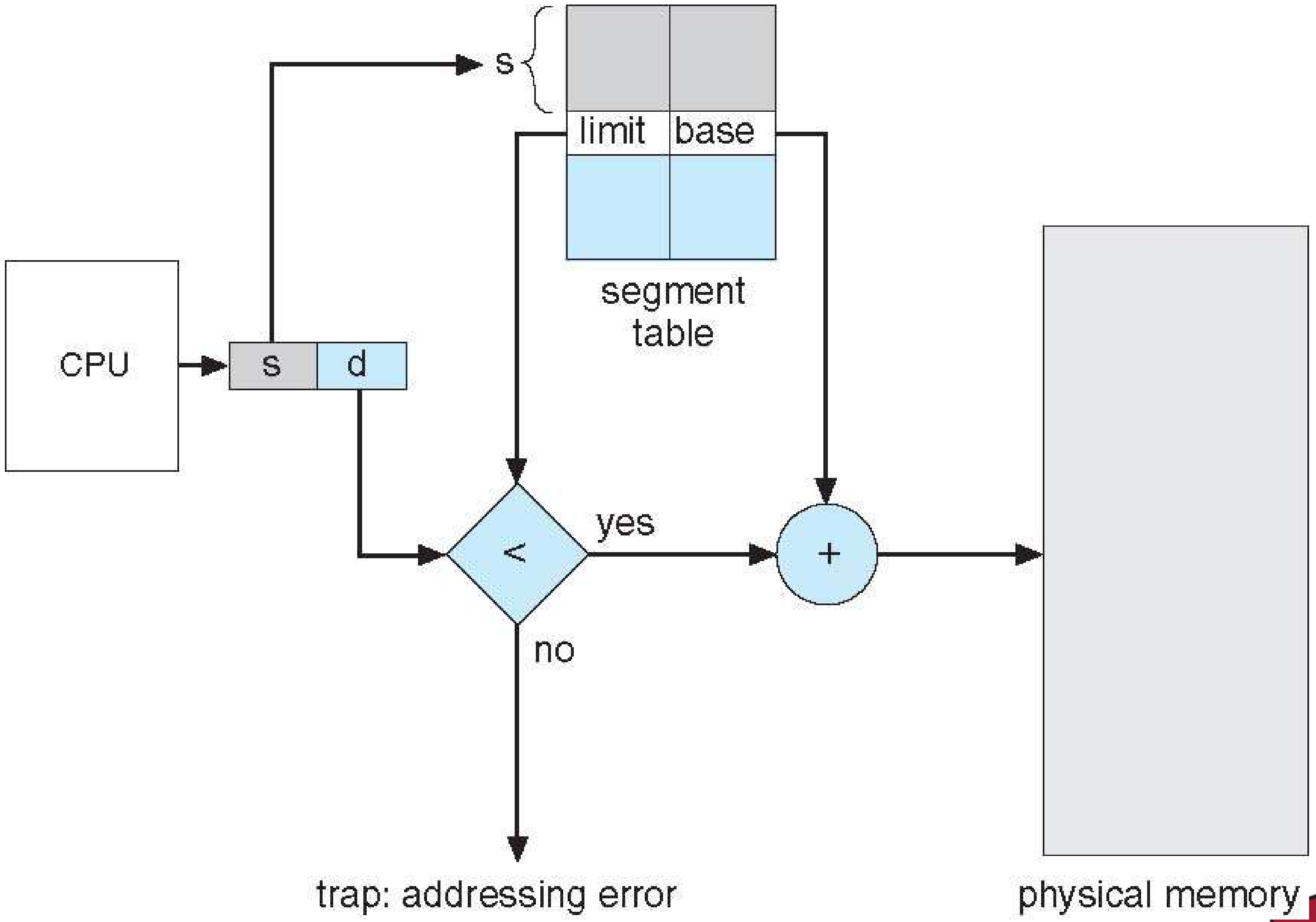
Example of Segmentation



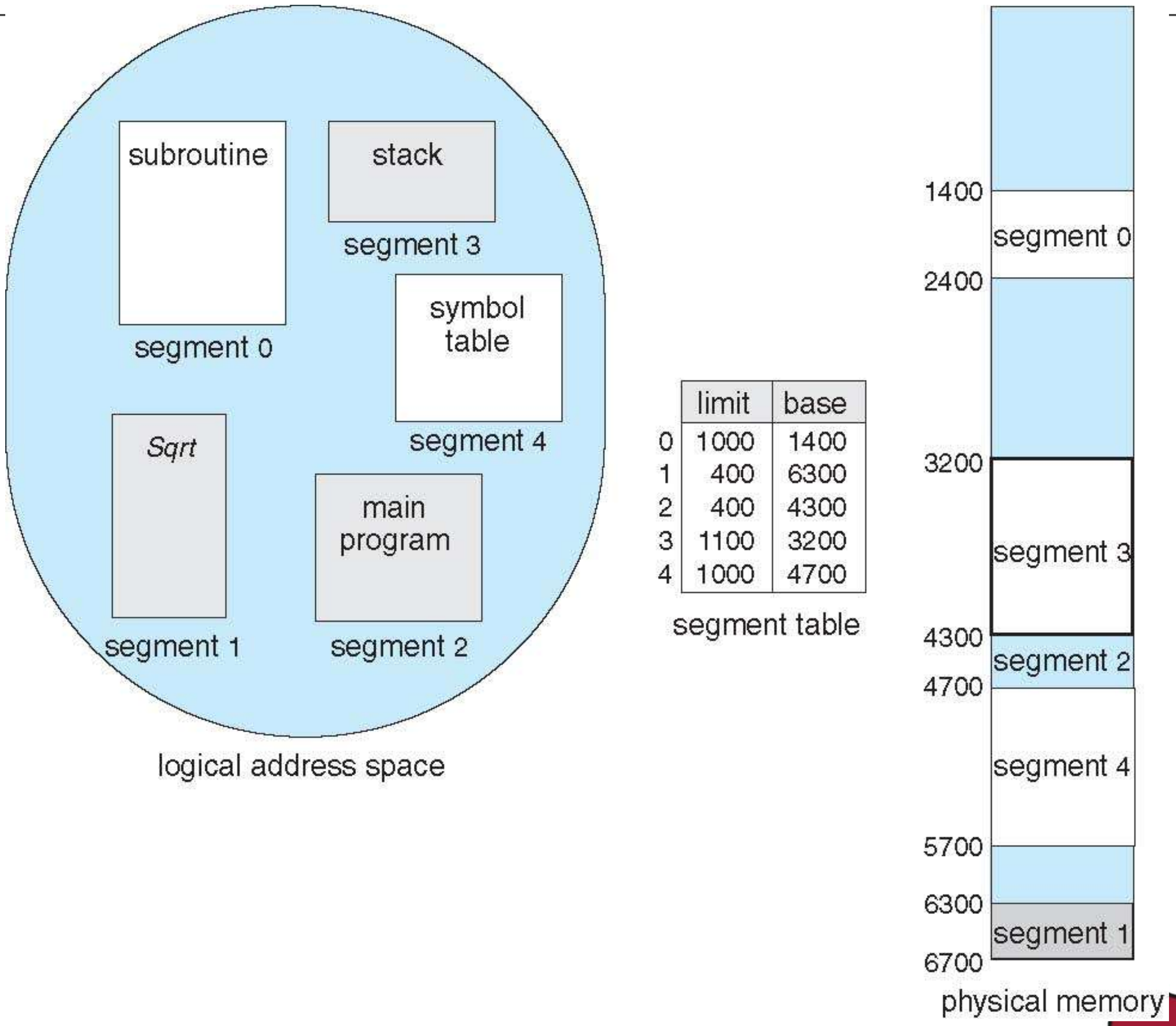
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table

Segmentation Hardware



Example of Segmentation



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table

Segmentation Architecture

- Protection
- Protection bits associated with segments
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges
- Code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
  - Long term scheduler
  - First fit, best fit etc
- Fragmentation

Segmentation with Paging

Key idea:

Segments are splitted into multiple pages

Each page is loaded into frames in the memory

Segmentation with Paging

- Supports segmentation with paging
  - Each segment can be 4 GB
  - Up to 16 K segments per process
  - <selector(16), offset (32)>
  - Divided into two partitions
    - First partition of up to 8 K segments are private to process (kept in **local descriptor table LDT**)
    - Second partition of up to 8K segments shared among all processes (kept in **global descriptor table GDT**)

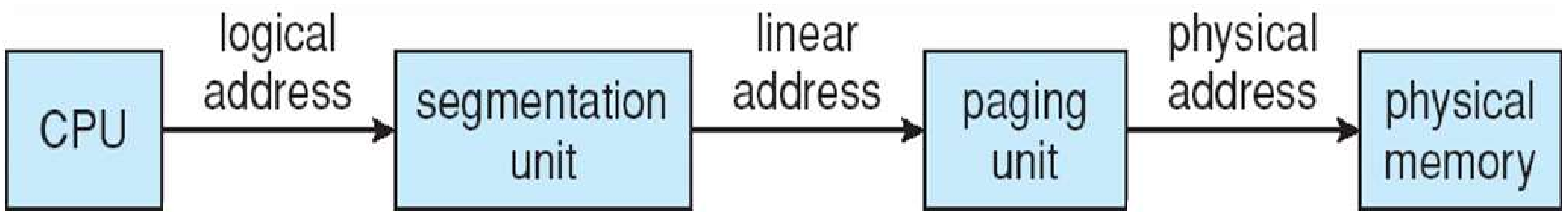
S(13)	G(1)	P(2)

- CPU generates logical address (six Segment Reg.)
  - Given to segmentation unit
    - Which produces linear addresses
  - Physical address 32 bits
  - Linear address given to paging unit
    - Which generates physical address in main memory
    - Paging units form equivalent of MMU
    - Pages sizes can be 4 KB

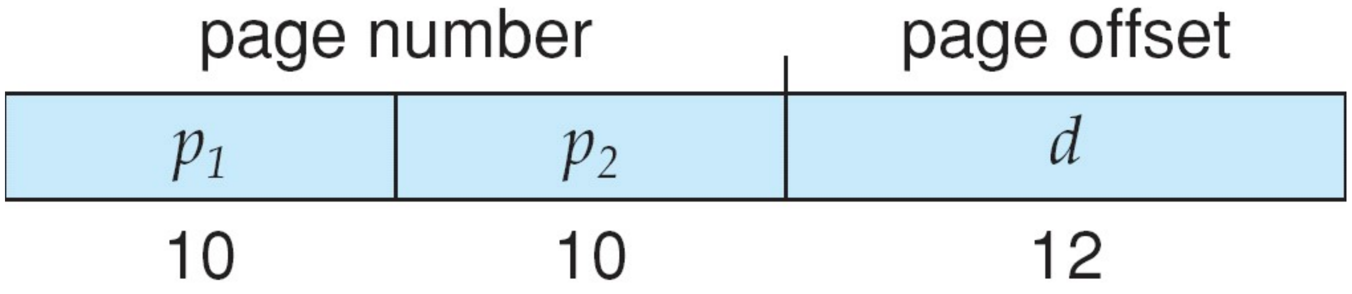
Intel 80386

IBM OS/2

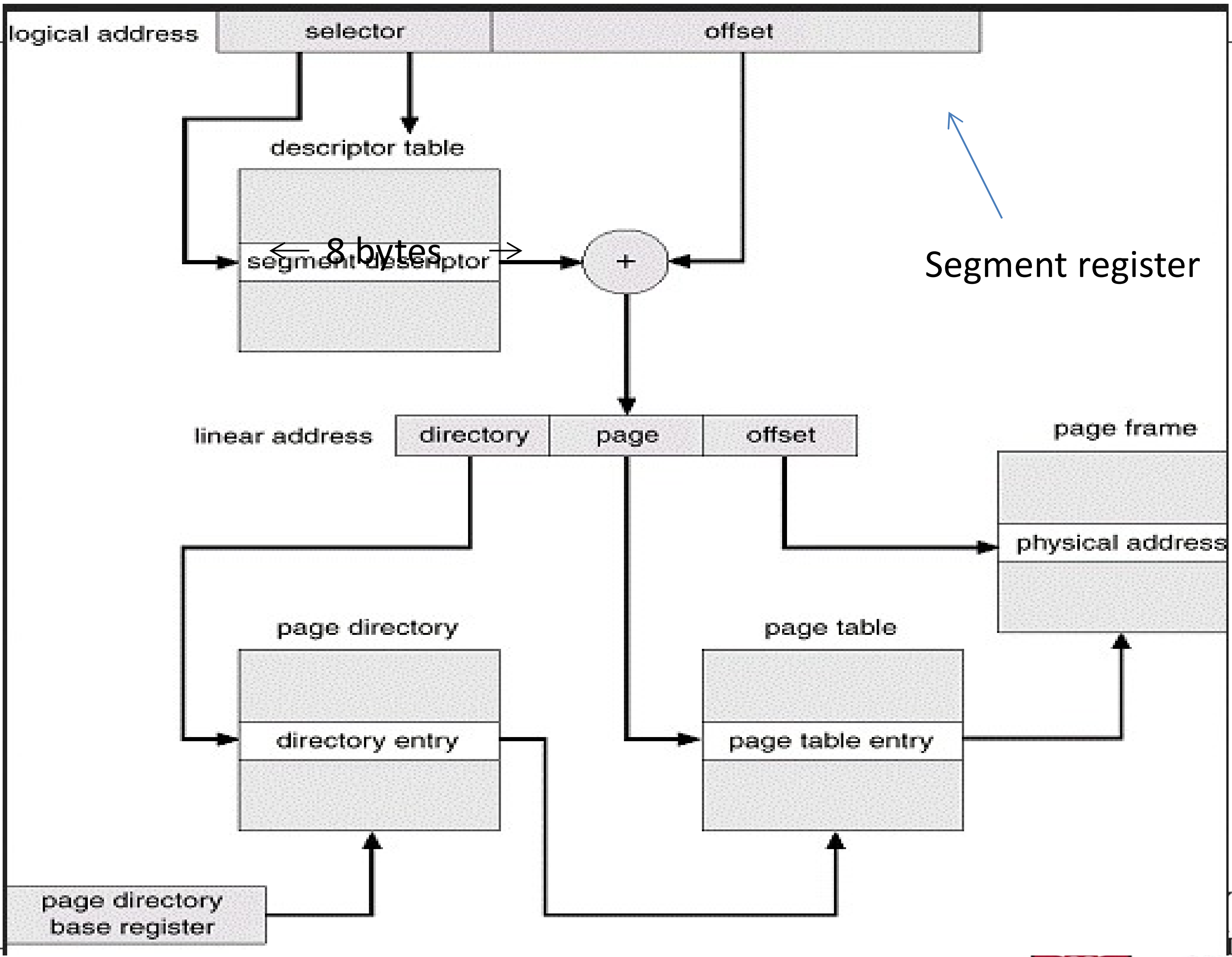
Logical to Physical Address Translation in Pentium



Page table=2^{20} entries

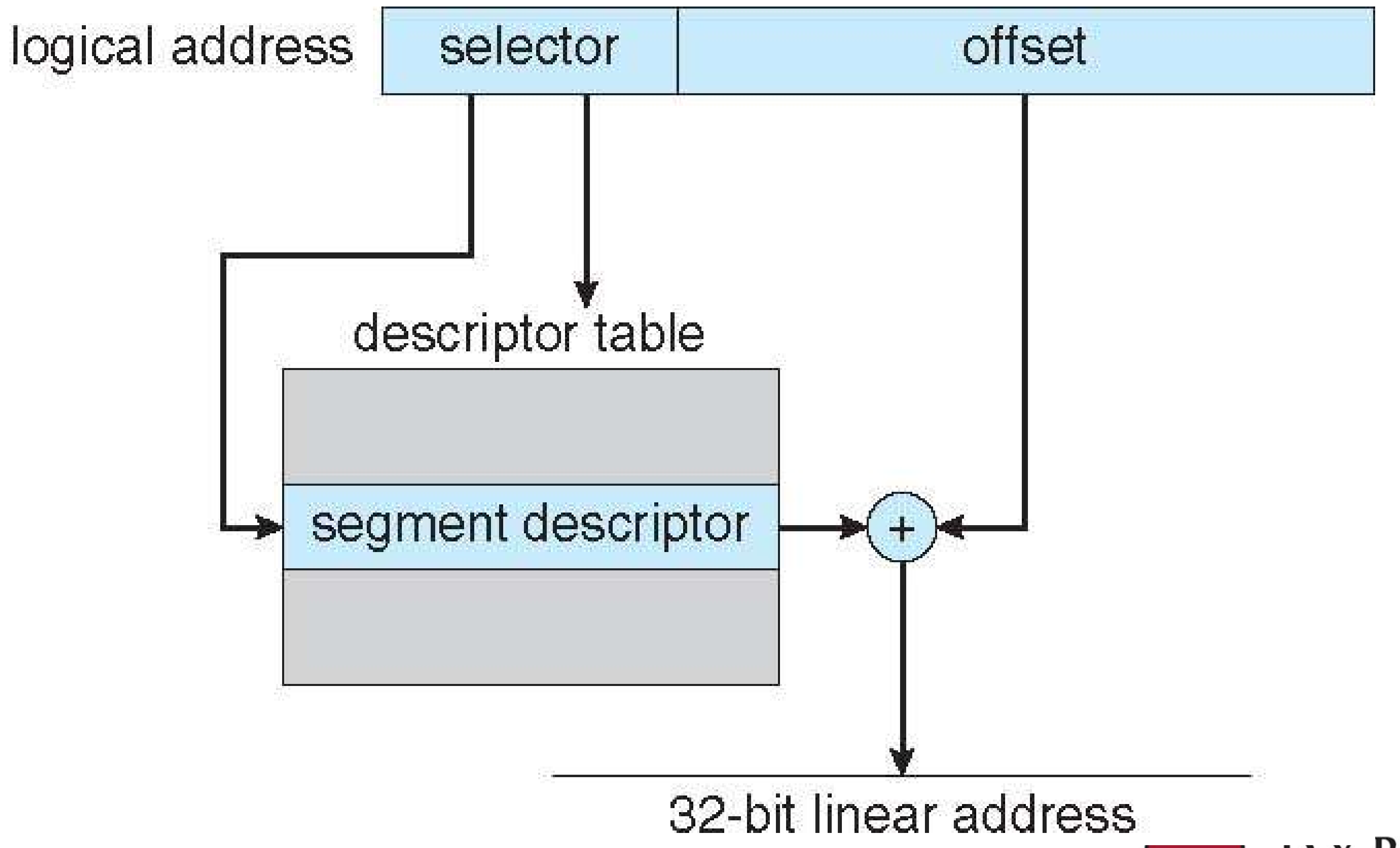


Example: The Intel Pentium

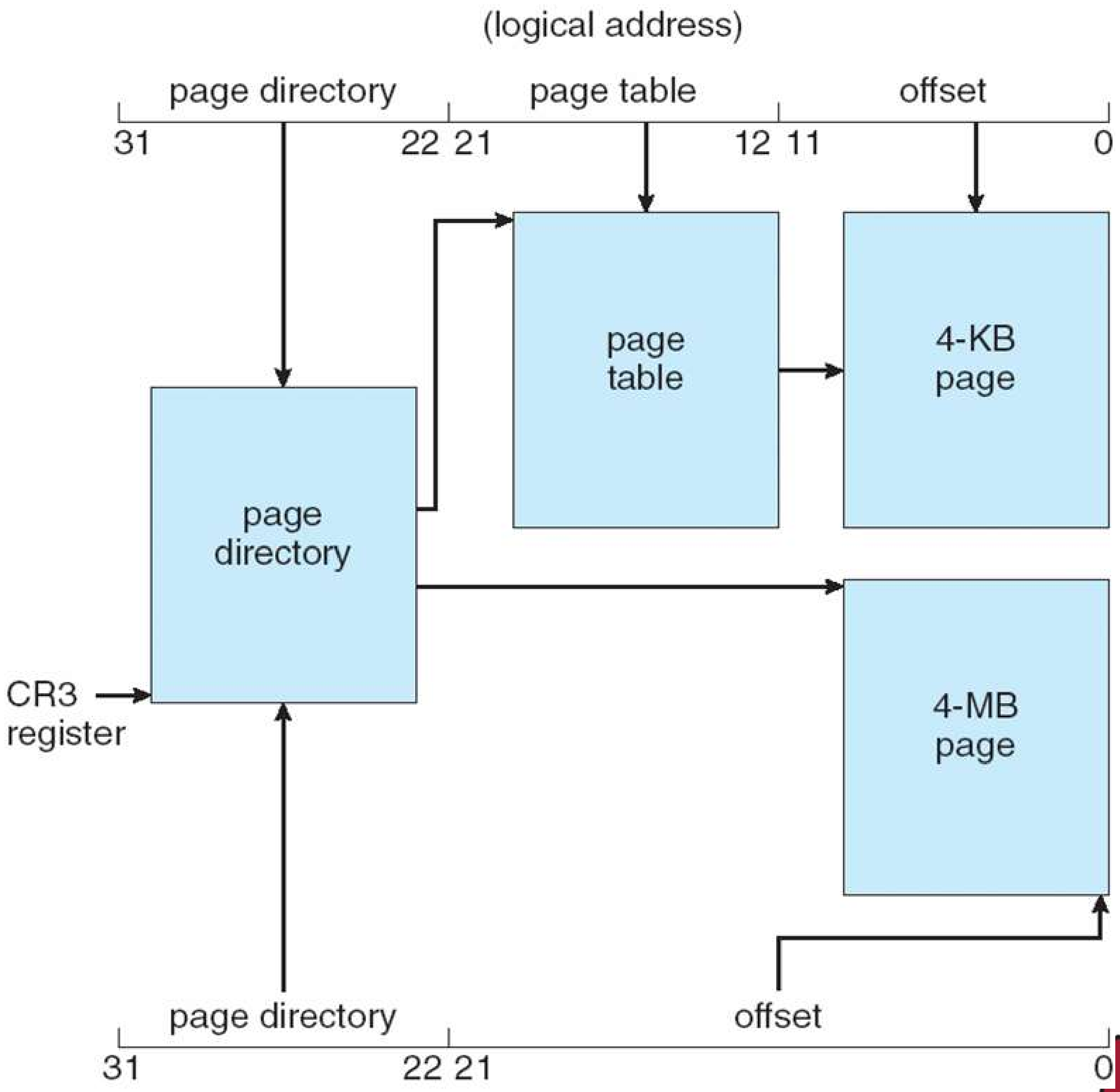


Segment register

Intel Pentium Segmentation



Pentium Paging Architecture



Thank You

D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI