CS 143A: Operating Systems

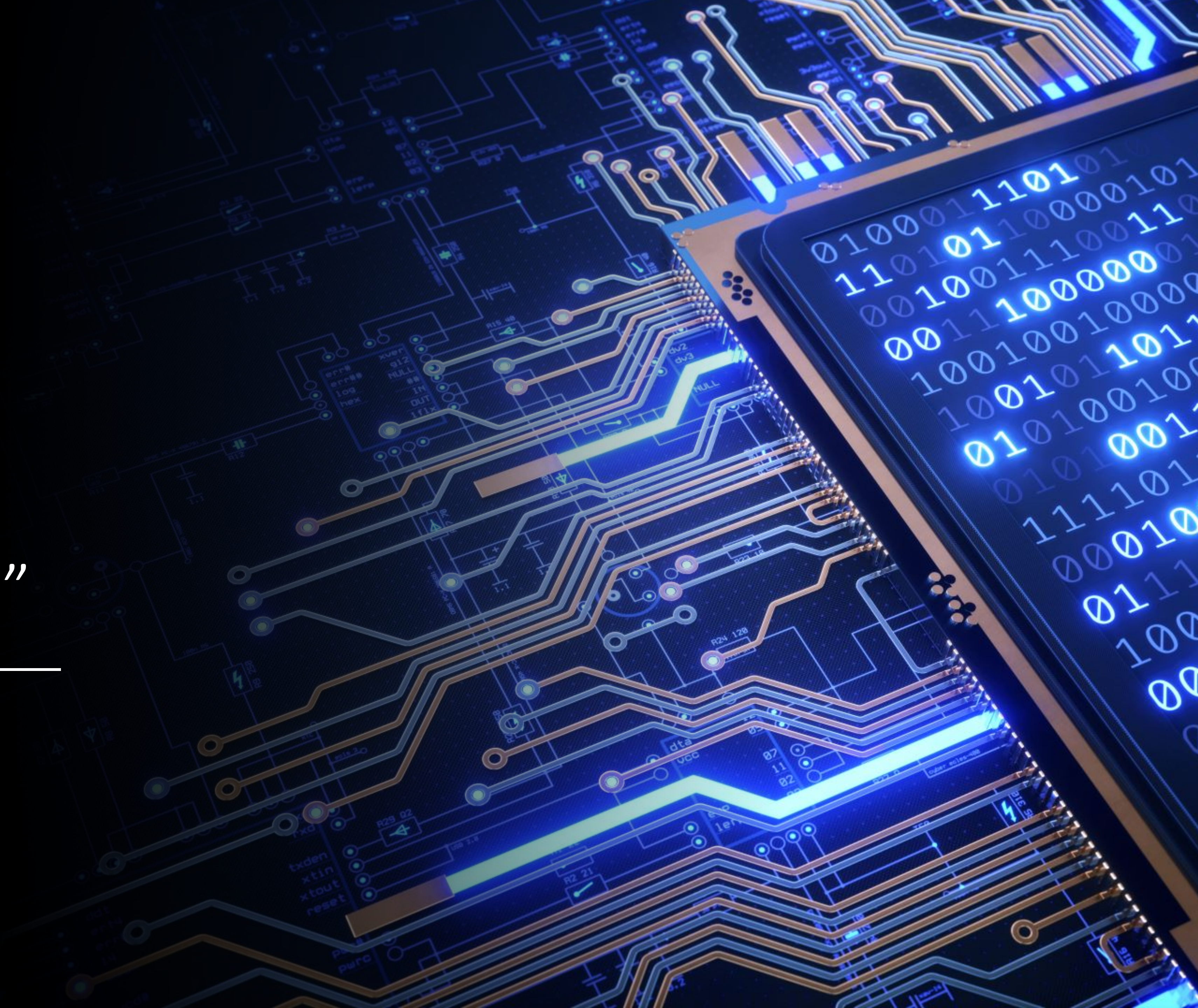
Discussion 5: Examples of segmentation and paging

Agenda

- Memory access quick recap
- Example of segmentation
- Example of paging
- Address translation overview

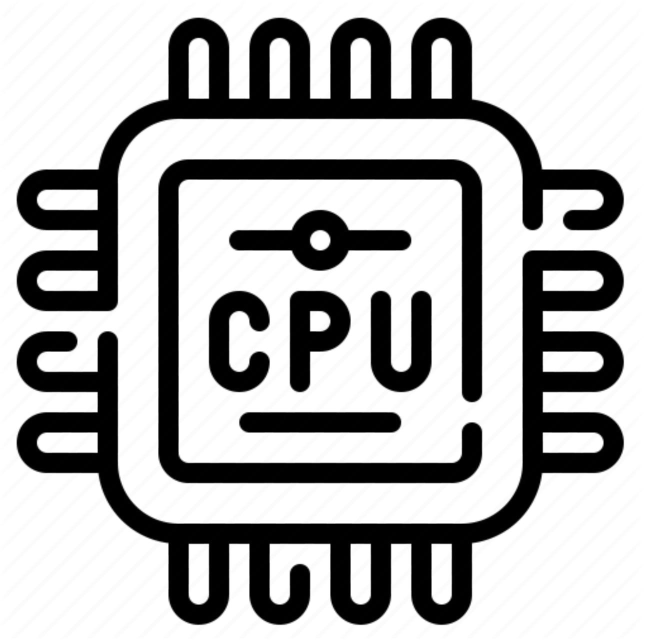
“All problems in computer science can be
solved by another level of indirection”

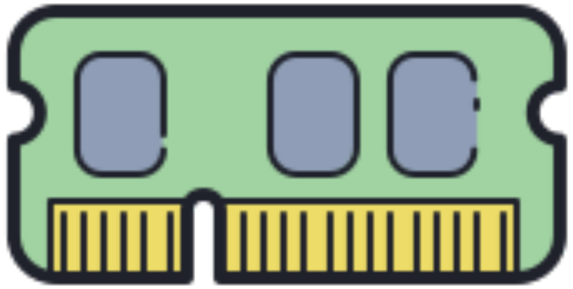
David Wheeler



Q1. How many address bits does a system have ?

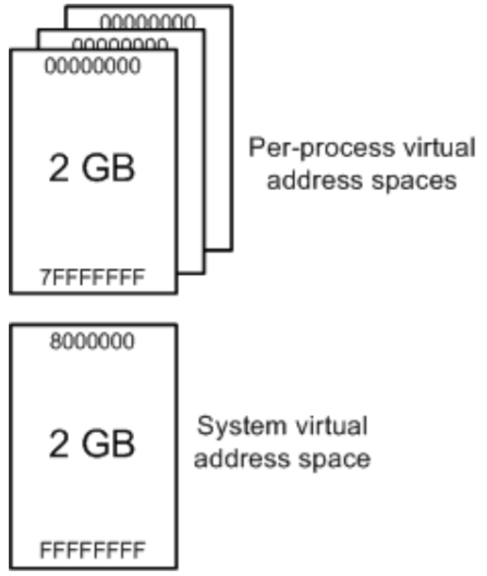
Depends on the CPU architecture





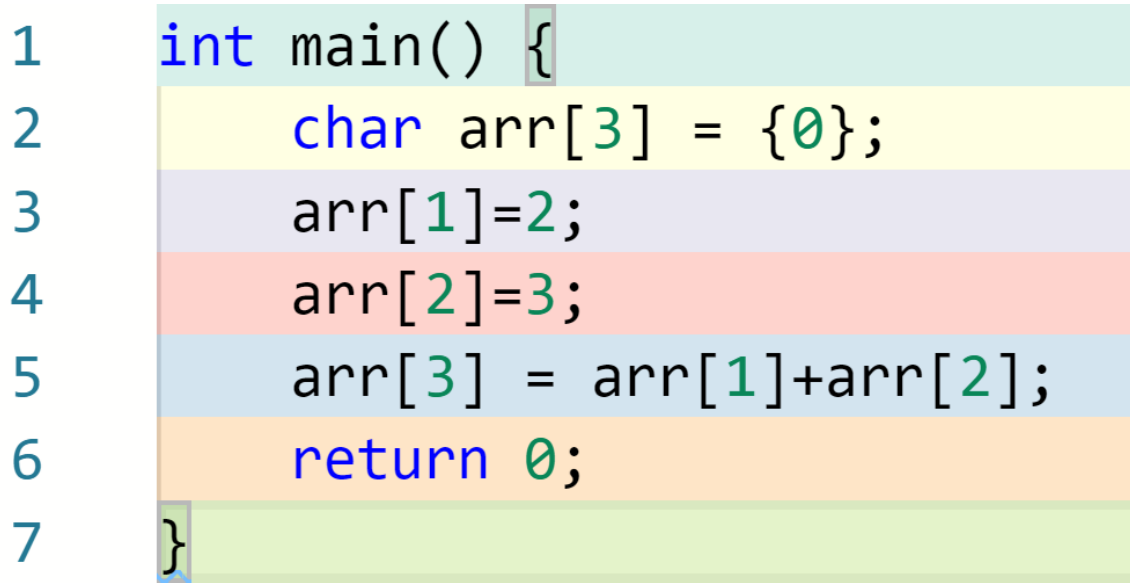
Examples

- X86 architecture – 32 bit
  - Page based 32-bit virtual memory system
  - 4GB (2^{32} = 4GB) of virtual memory.

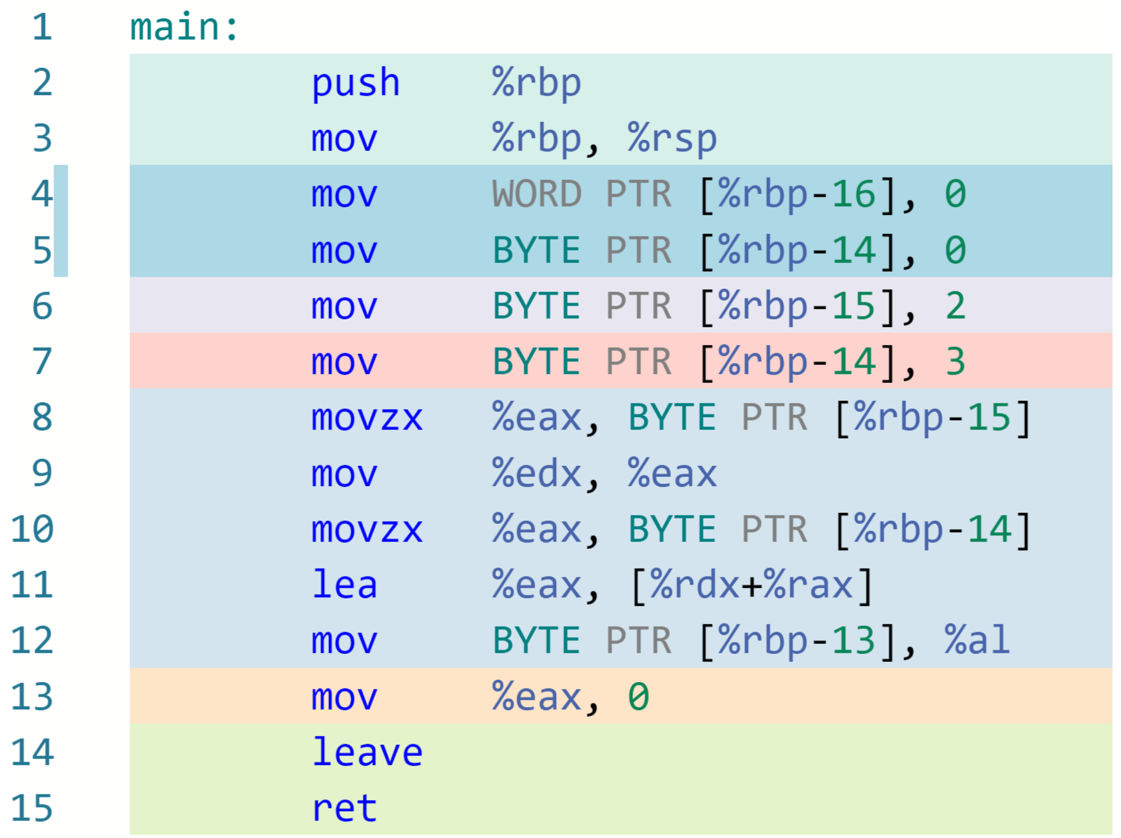

00000000
00000000
00000000
2 GB
7FFFFFFF
Per-process virtual address spaces
8000000
2 GB
FFFFFFFF
System virtual address space

- X86-64 architecture
  - Page based 64-bit virtual memory system

- Actual available physical memory might be quite less (e.g., 1GB RAM).

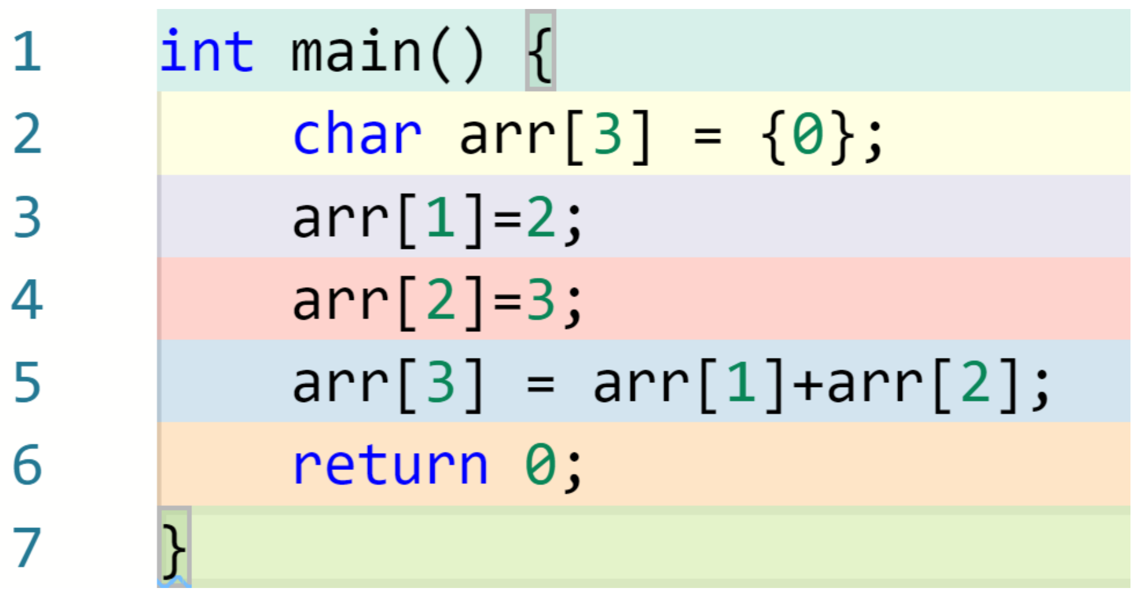


```c
int main() {
    char arr[3] = {0};
    arr[1]=2;
    arr[2]=3;
    arr[3] = arr[1]+arr[2];
    return 0;
}
```

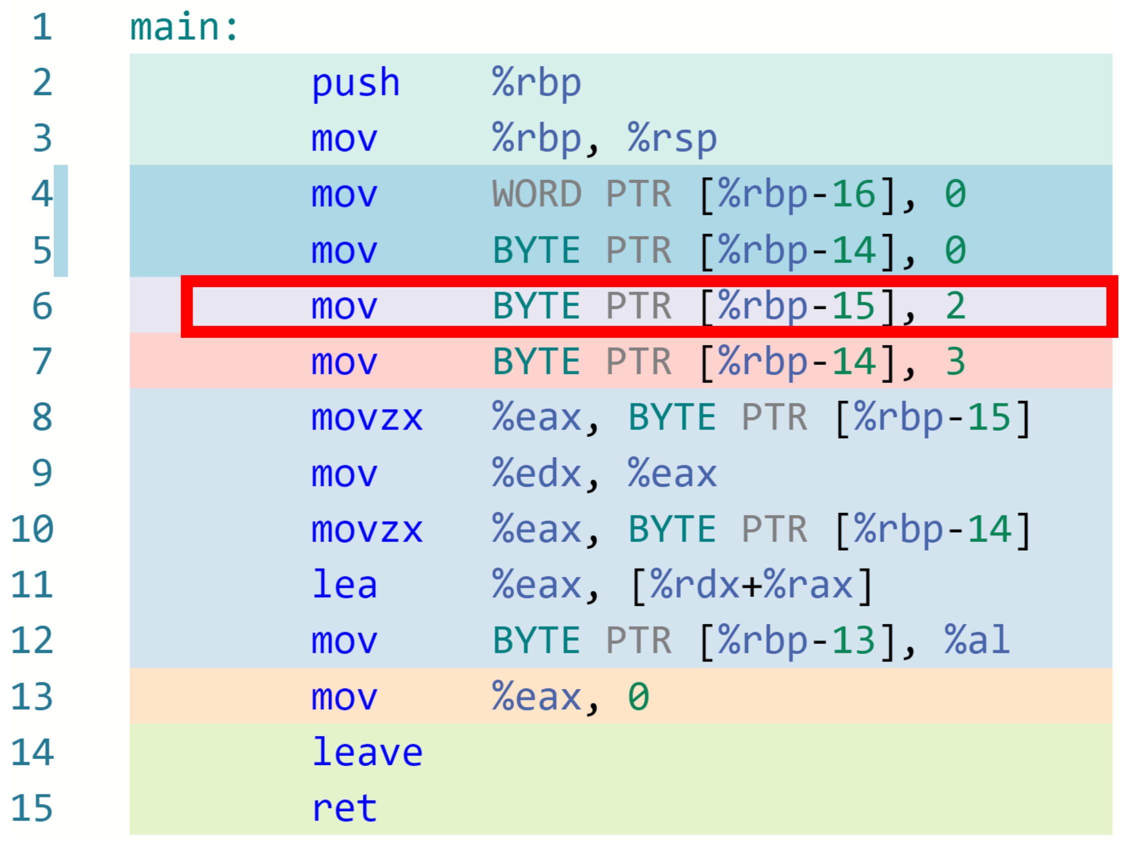


```
main:
        push    %rbp
        mov     %rbp, %rsp
        mov     WORD PTR [%rbp-16], 0
        mov     BYTE PTR [%rbp-14], 0
        mov     BYTE PTR [%rbp-15], 2
        mov     BYTE PTR [%rbp-14], 3
        movzx   %eax, BYTE PTR [%rbp-15]
        mov     %edx, %eax
        movzx   %eax, BYTE PTR [%rbp-14]
        lea     %eax, [%rdx+%rax]
        mov     BYTE PTR [%rbp-13], %al
        mov     %eax, 0
        leave
        ret
```

High level code -> Compiled code



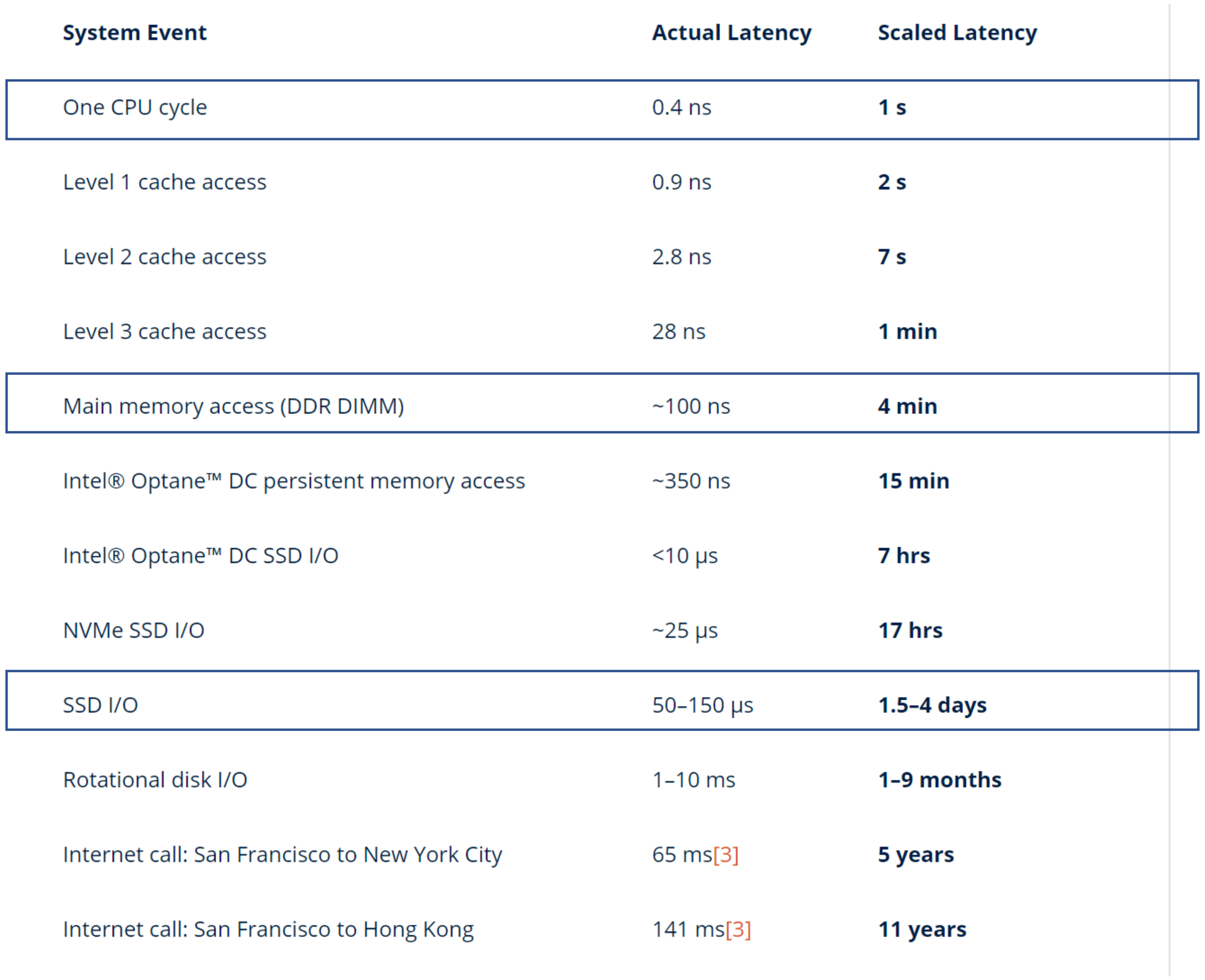
```c
int main() {
    char arr[3] = {0};
    arr[1]=2;
    arr[2]=3;
    arr[3] = arr[1]+arr[2];
    return 0;
}
```



```
main:
        push    %rbp
        mov     %rbp, %rsp
        mov     WORD PTR [%rbp-16], 0
        mov     BYTE PTR [%rbp-14], 0
        mov     BYTE PTR [%rbp-15], 2
        mov     BYTE PTR [%rbp-14], 3
        movzx   %eax, BYTE PTR [%rbp-15]
        mov     %edx, %eax
        movzx   %eax, BYTE PTR [%rbp-14]
        lea     %eax, [%rdx+%rax]
        mov     BYTE PTR [%rbp-13], %al
        mov     %eax, 0
        leave
        ret
```

High level code -> Compiled code

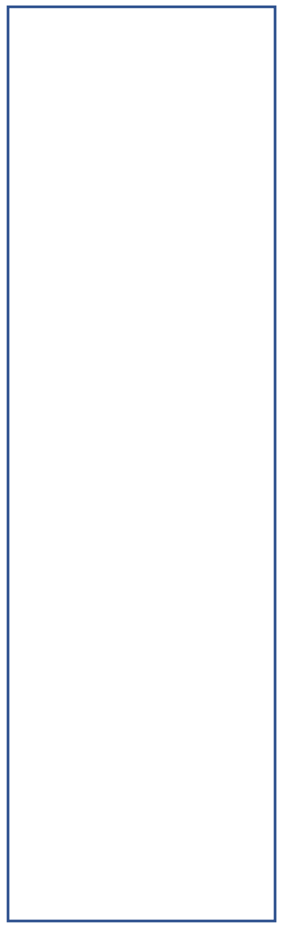
Latency examples



System Event	Actual Latency	Scaled Latency
One CPU cycle	0.4 ns	**1 s**
Level 1 cache access	0.9 ns	**2 s**
Level 2 cache access	2.8 ns	**7 s**
Level 3 cache access	28 ns	**1 min**
Main memory access (DDR DIMM)	~100 ns	**4 min**
Intel® Optane™ DC persistent memory access	~350 ns	**15 min**
Intel® Optane™ DC SSD I/O	<10 μs	**7 hrs**
NVMe SSD I/O	~25 μs	**17 hrs**
SSD I/O	50–150 μs	**1.5–4 days**
Rotational disk I/O	1–10 ms	**1–9 months**
Internet call: San Francisco to New York City	65 ms[3]	**5 years**
Internet call: San Francisco to Hong Kong	141 ms[3]	**11 years**

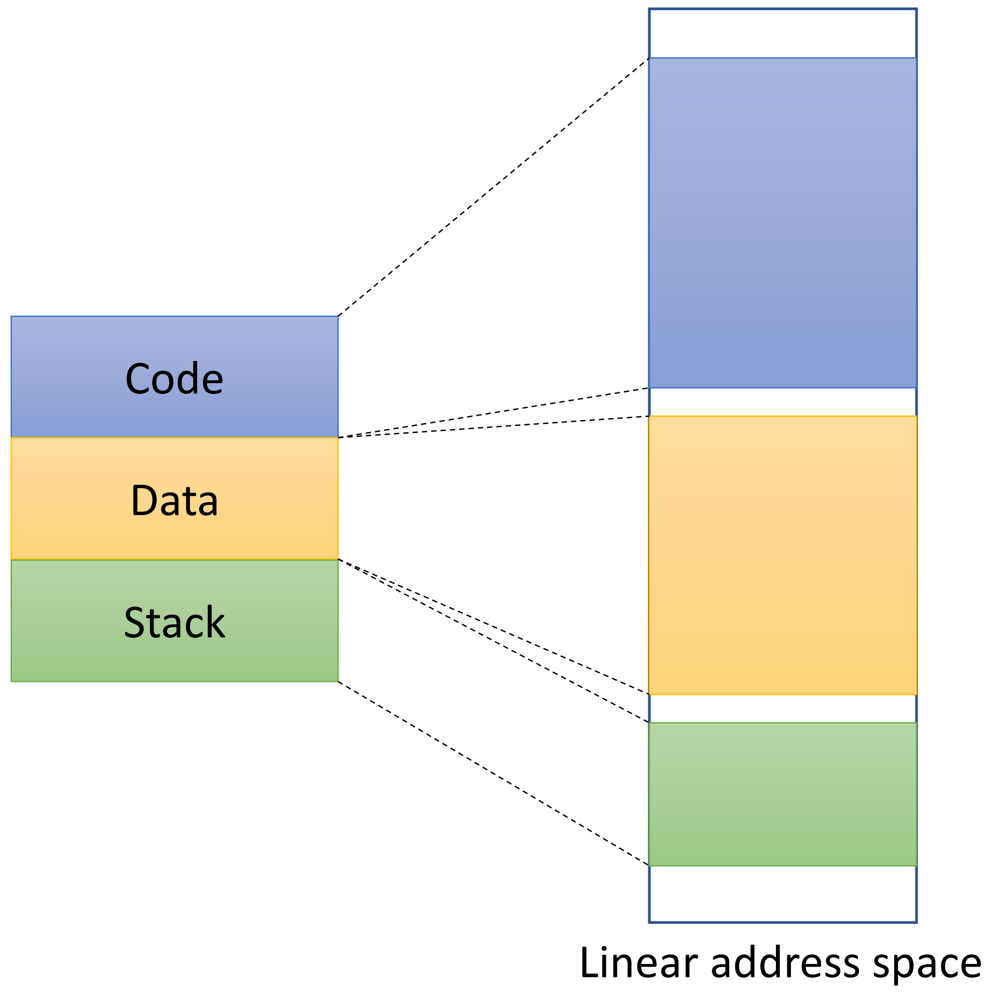
https://www.prowesscorp.com/computer-latency-at-a-human-scale/

Segmentation example


Linear address space

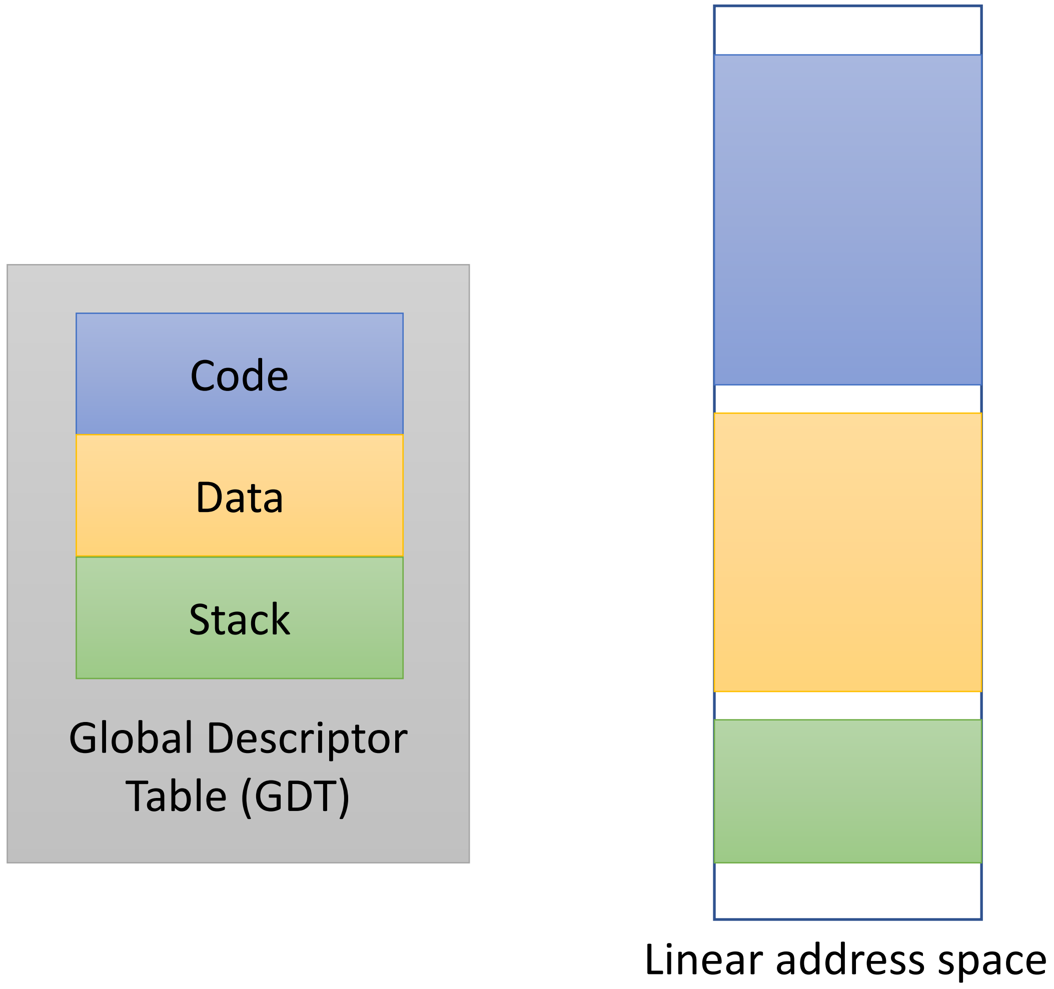
- A mechanism for dividing the processor’s addressable memory space (called the linear address space) into smaller protected address spaces called segments.

Segmentation example



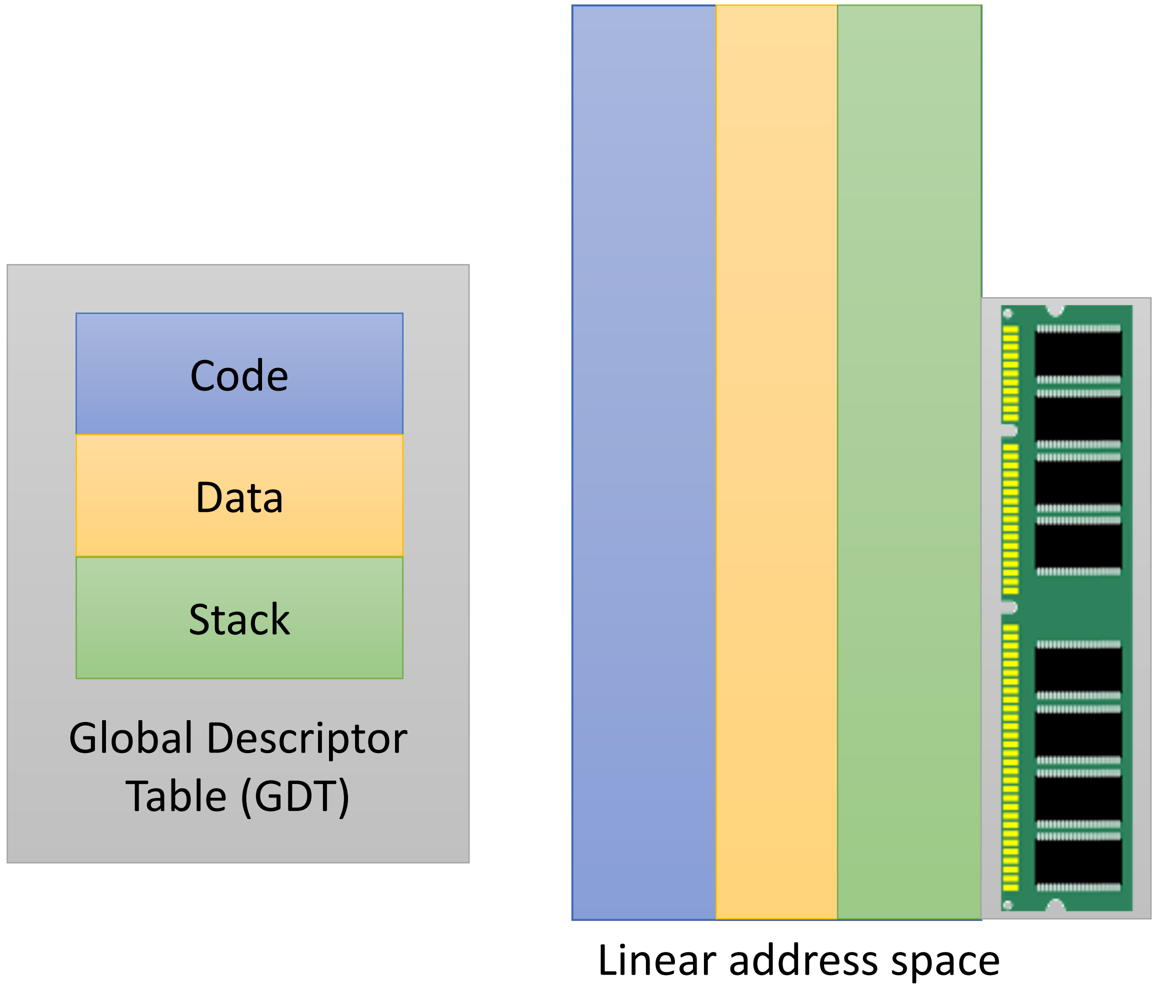
- A mechanism for dividing the processor’s addressable memory space (called the linear address space) into smaller protected address spaces called segments.
- Code, Data, and Stack for a program or System data structures

Segmentation example



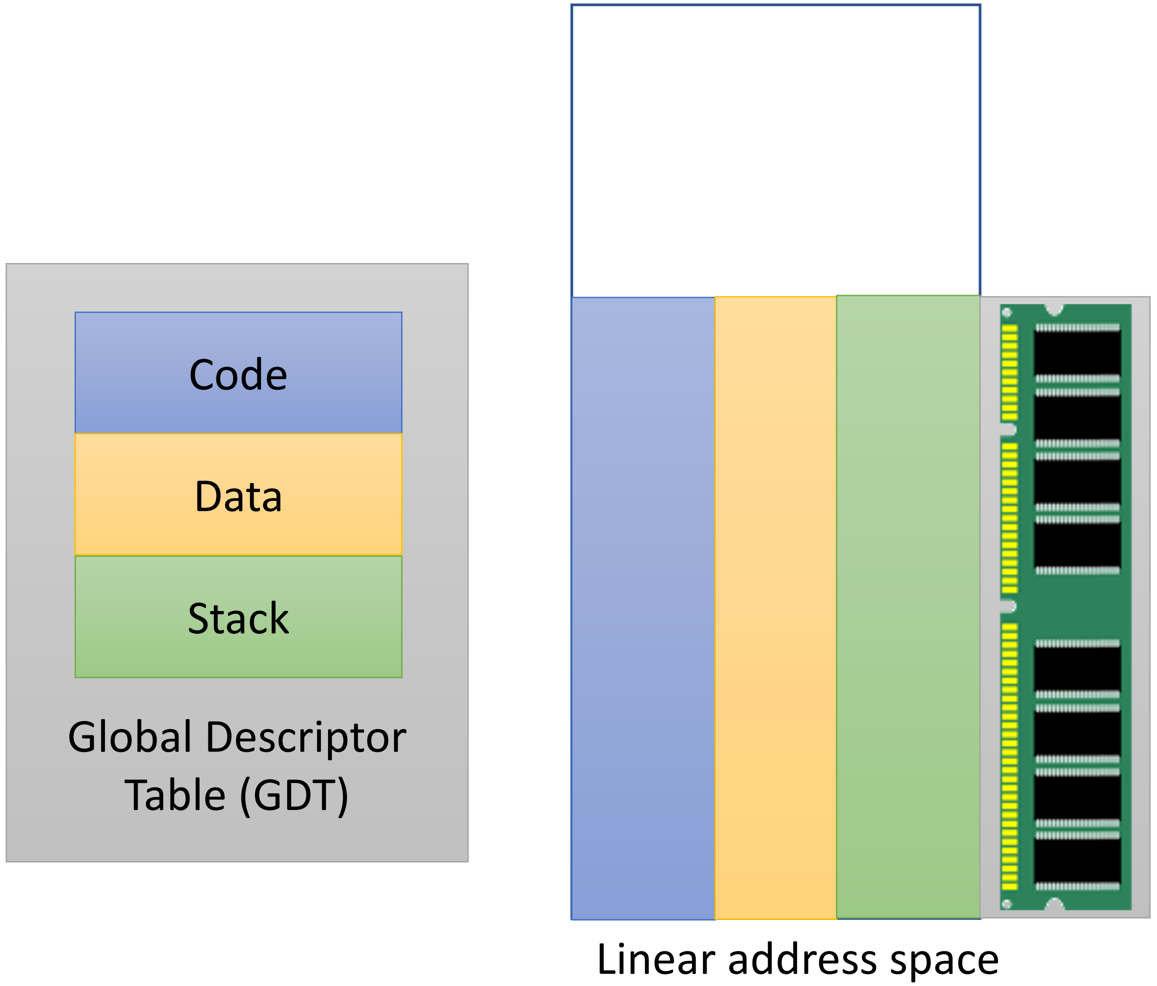
- A mechanism for dividing the processor’s addressable memory space (called the linear address space) into smaller protected address spaces called segments.
- Code, Data, and Stack for a program or System data structures

Segmentation example



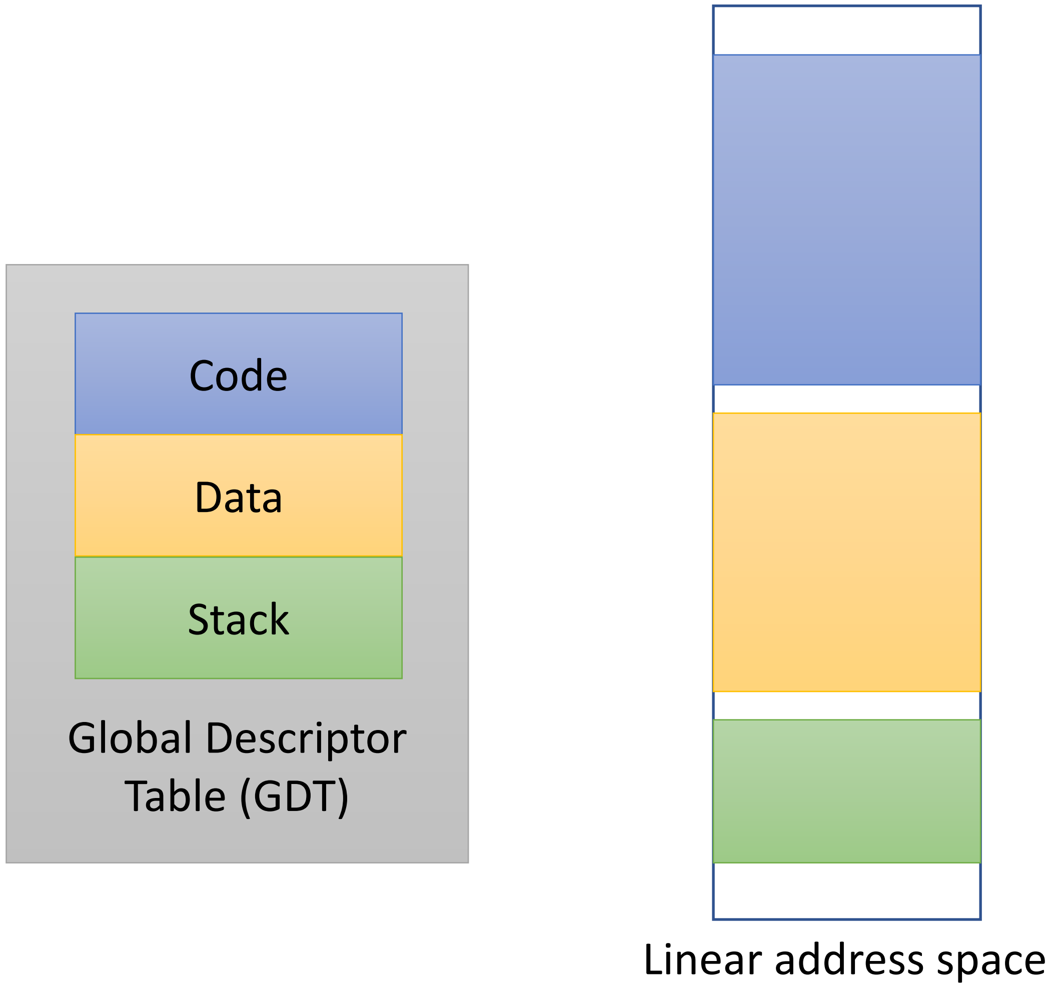
- A mechanism for dividing the processor's addressable memory space (called the linear address space) into smaller protected address spaces called segments.
- Code, Data, and Stack for a program or System data structures
- Different models of segmentation

Segmentation example



- A mechanism for dividing the processor’s addressable memory space (called the linear address space) into smaller protected address spaces called segments.
- Code, Data, and Stack for a program or System data structures
- Different models of segmentation

Segmentation example



- A mechanism for dividing the processor’s addressable memory space (called the linear address space) into smaller protected address spaces called segments.
- Code, Data, and Stack for a program or System data structures
- Different models of segmentation

Segmentation example

- Illustrate the memory organization of the x86 logical address translation through a simple example.

  Assume that the hardware translates the logical address ‘0xb00002005’.

  The GDT register value is ‘0x7095’ and base address of the segment involved in the translation of this logical address is 0xc00000.

  Draw a diagram representing the state of the GDT in physical memory and the process of translation.

- Assume: Padding left : 0 whenever applicable

Logical address (0xb00002005)

Logical address (0xb00002005)

Segment Selector				Offset							
0	**0**	**0**	**b**	**0**	**0**	**0**	**0**	**2**	**0**	**0**	**5**
0000	0000	0000	1011	0000	0000	0000	0000	0010	0000	0000	0101

GDT lookup

Segment Selector			
0	**0**	**0**	**b**
0000	**0000**	**0000**	**1** 0 11

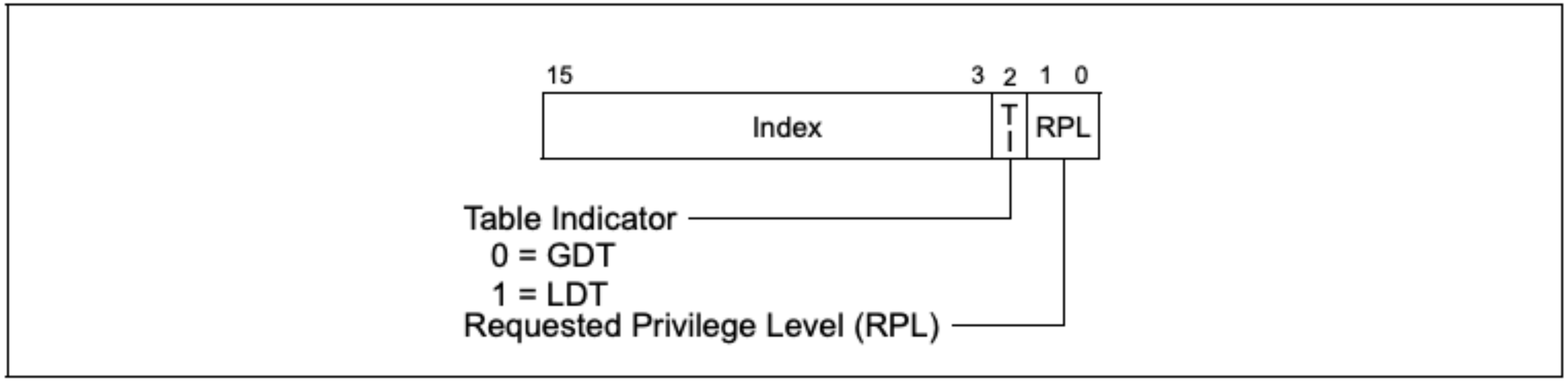
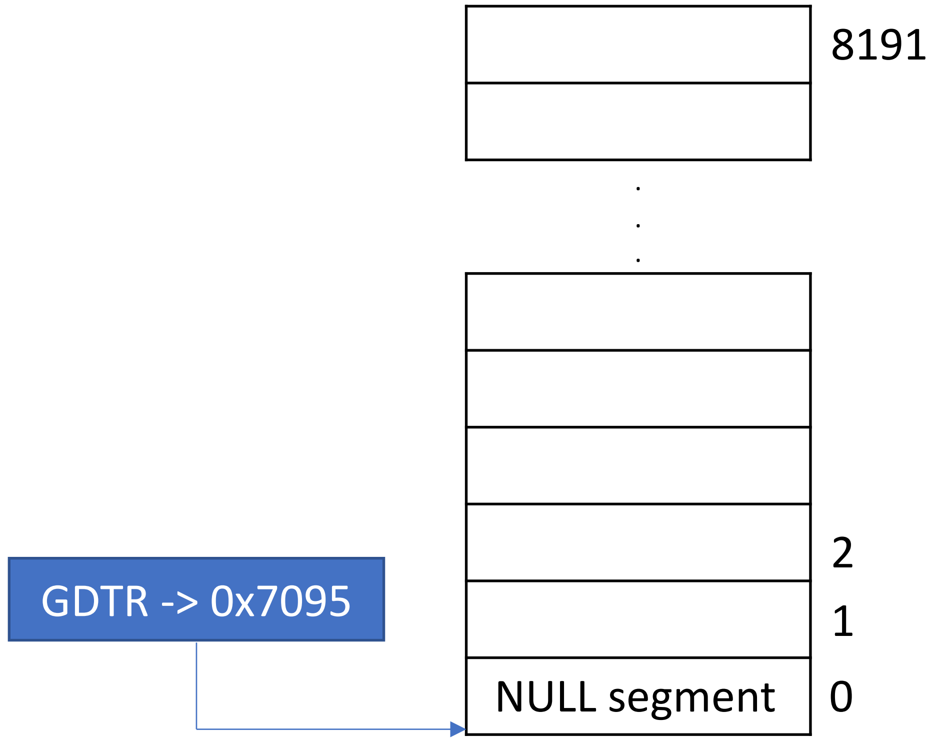


Figure 3-6. Segment Selector

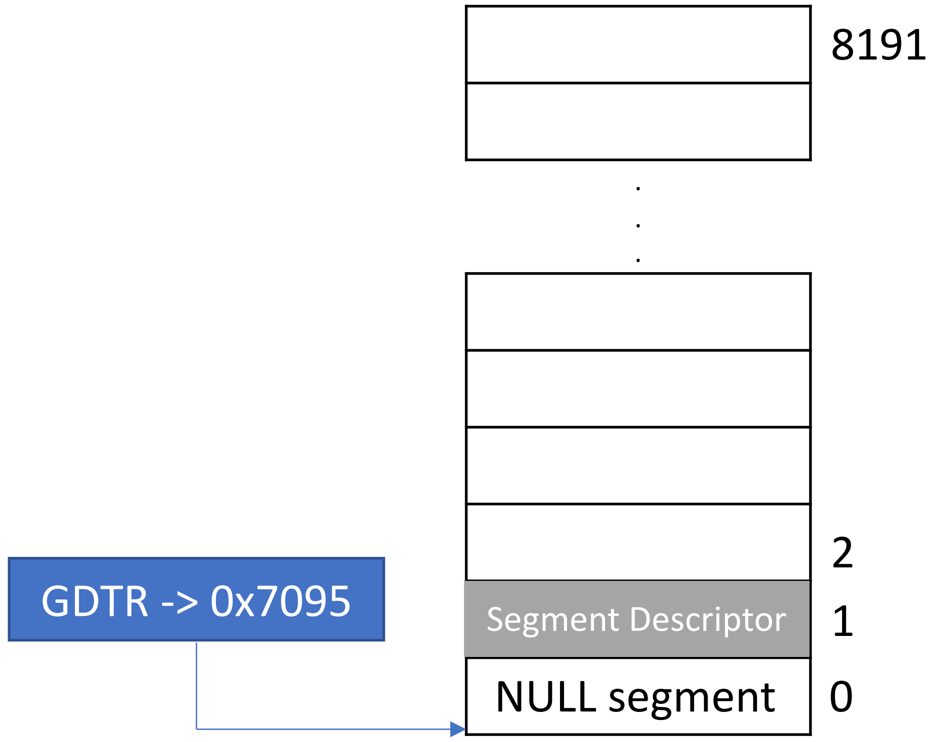
GDT lookup

Segment Selector			
0	**0**	**0**	**b**
0000	**0000**	**0000**	**1** 0 11



GDT lookup

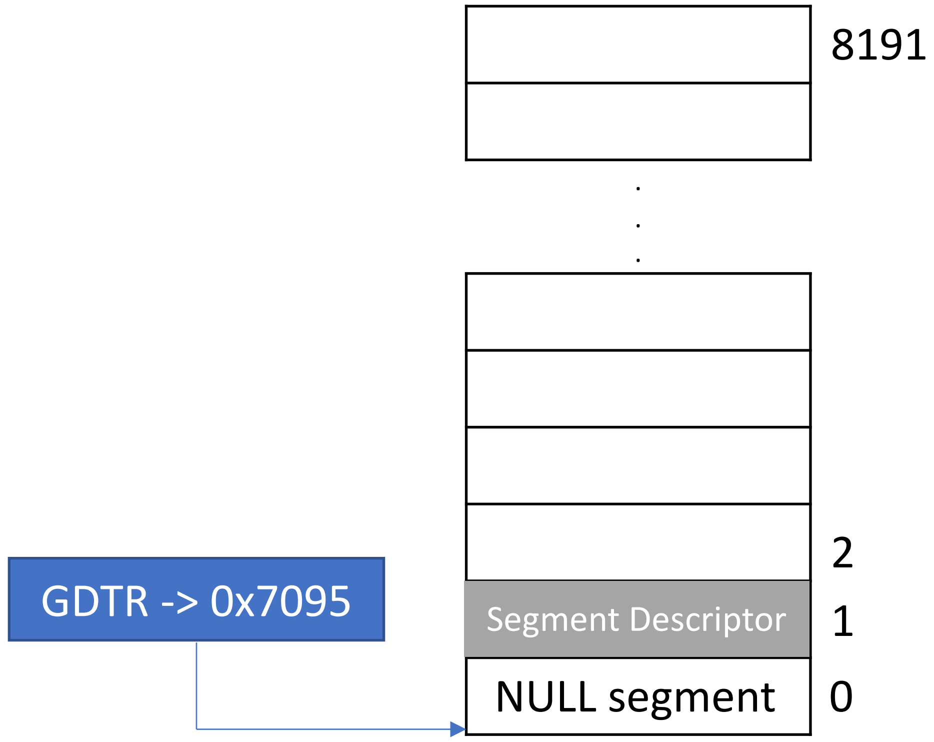
Segment Selector			
0	**0**	**0**	**b**
0000	**0000**	**0000**	**1** 0 11



Base	**Limit**	**Access Control**

GDT lookup

Segment Selector			
0	**0**	**0**	**b**
0000	**0000**	**0000**	**1** 0 11

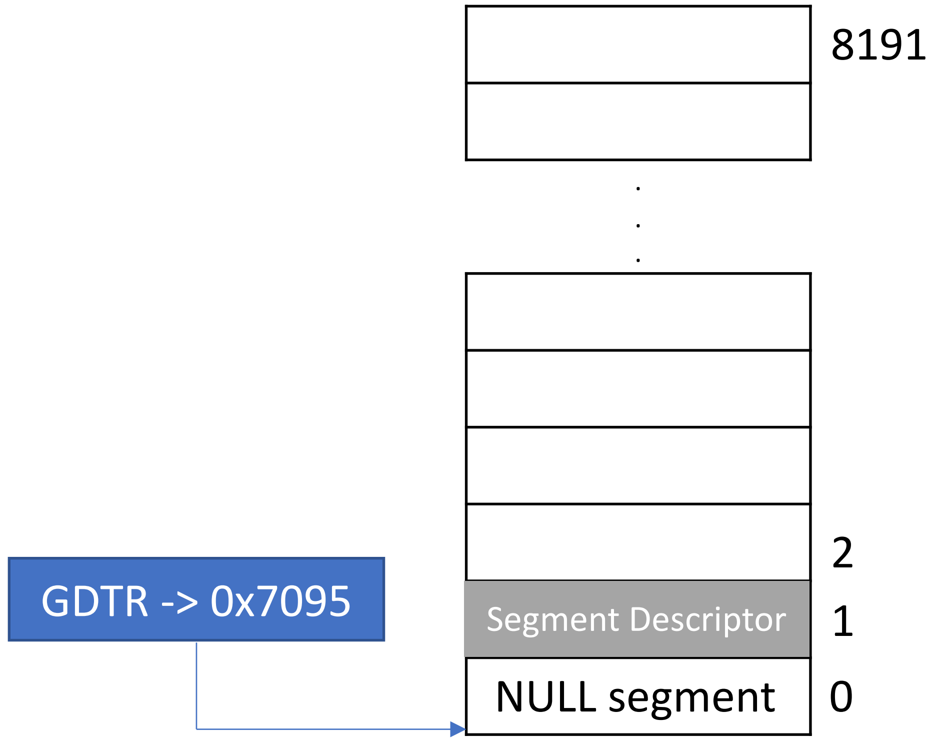


(Recall: base address of the segment involved in the translation of this logical address is 0xc00000.)

Base	Limit	Access Control
0x00c00000		

Linear address generation ?

Segment Selector				Offset							
0	**0**	**0**	**b**	**0**	**0**	**0**	**0**	**2**	**0**	**0**	**5**
0000	**0000**	**0000**	**1** 0 11	0000	0000	0000	0000	0010	0000	0000	0101



Linear address
= 0x00c00000 (Base)+ 0x00002005 (Offset)

= 0x00c02005

Base	Limit	Access Control
0x00c00000		

Linear address translation

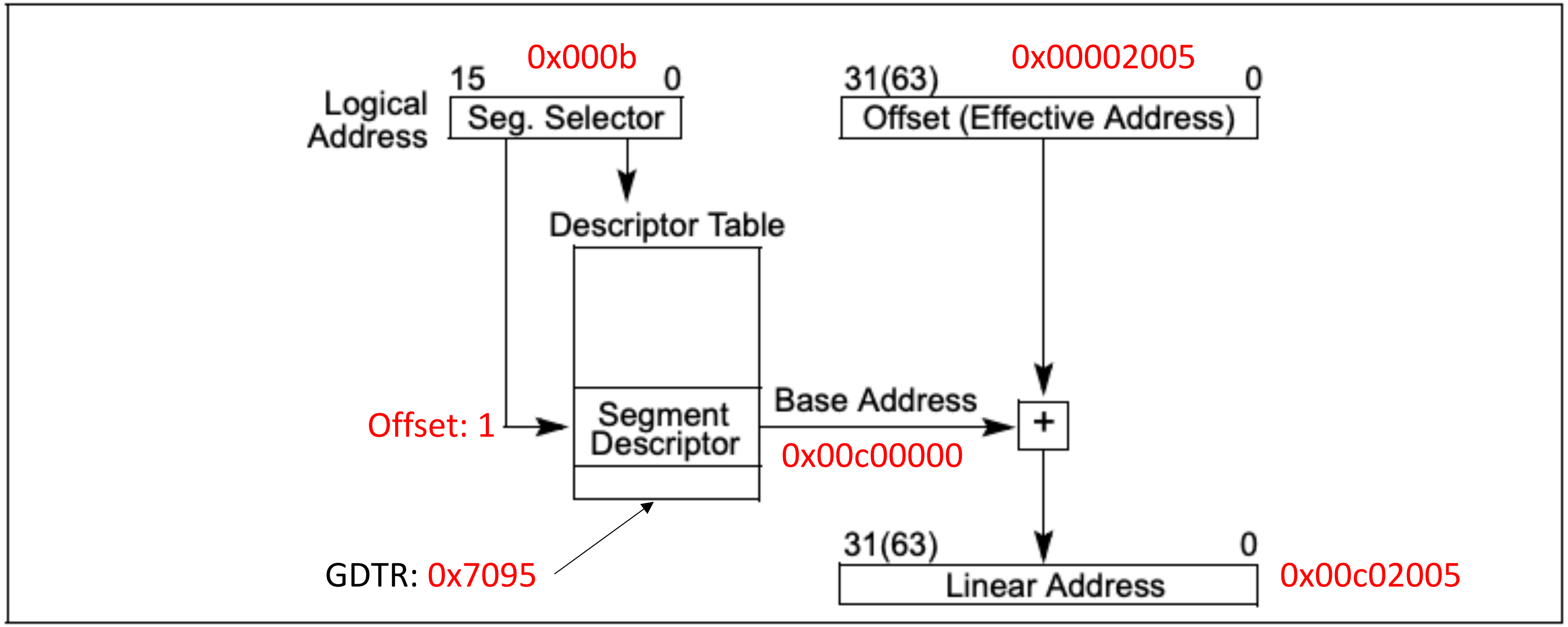


Figure 3-5. Logical Address to Linear Address Translation

Q: Where is the GDT ?

Q: Where is the GDT ?

In Main Memory

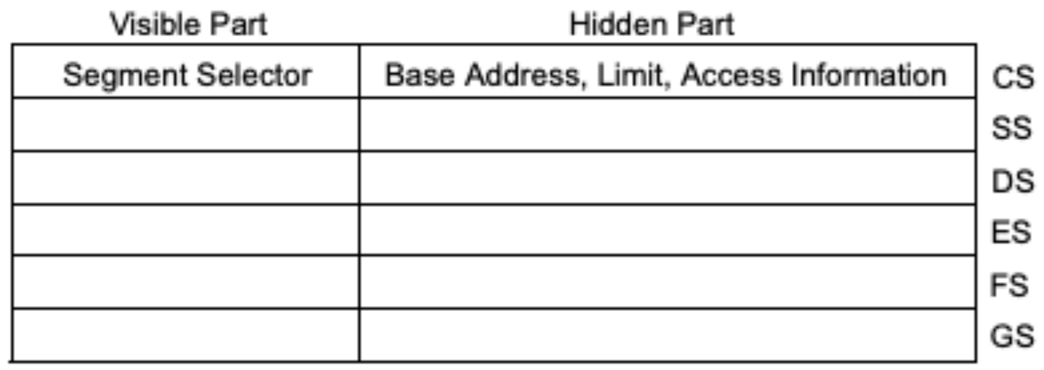
Q: Main memory access is slow. Can we do anything to make access faster ?

Q: Where is the GDT ?

In Main Memory

Q: Main memory access is slow. Can we do anything to make access faster ?

Yes, with registers to store 6 most recent translations
using Segment Registers



Visible Part	Hidden Part	
Segment Selector	Base Address, Limit, Access Information	CS
		SS
		DS
		ES
		FS
		GS

Figure 3-7. Segment Registers

Question

- Illustrate the memory organization of the x86, 4K, 32bit page table through a simple example.

  Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0. Draw a diagram representing the state of the page table in physical memory and the process of linear address space paging.

Virtual address : 0xc02005

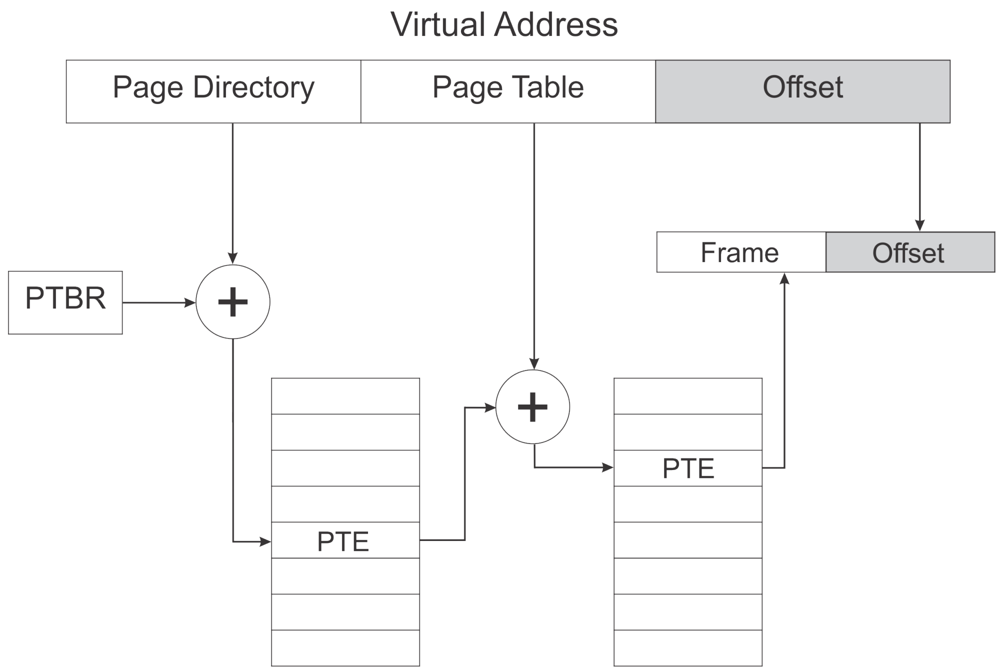
		c	0	2	0	0	5
0000	0000	1100	0000	0010	0000	0000	0101

Virtual address : 0xc02005

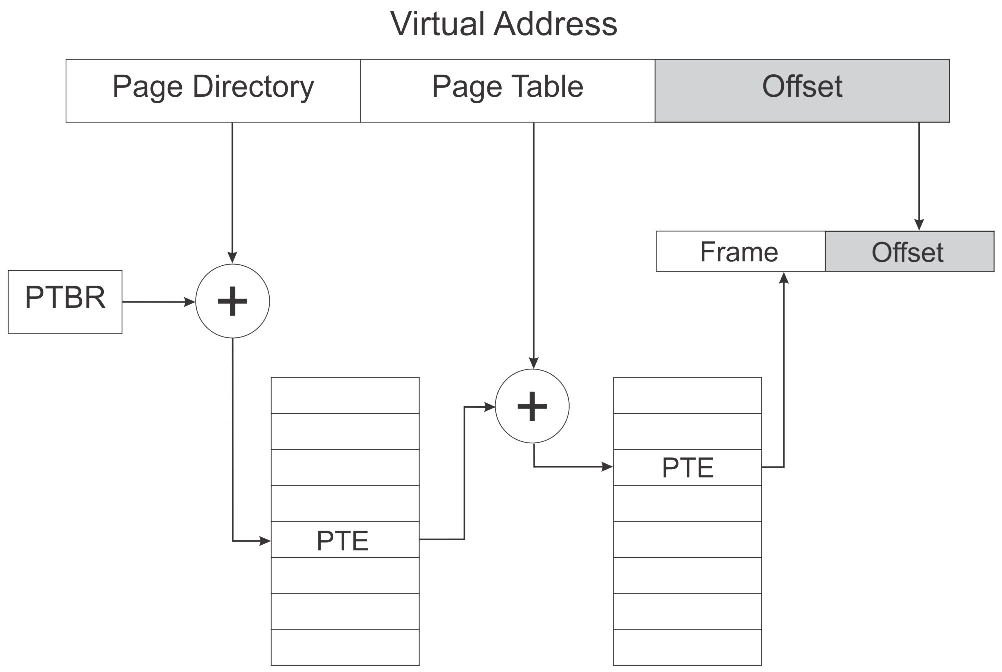
		c	0	2	0	0	5
0000	0000	1100	0000	0010	0000	0000	0101

0000 0000 1100 0000 0010 0000 0000 0101

Size of a page = 4K (specified already)



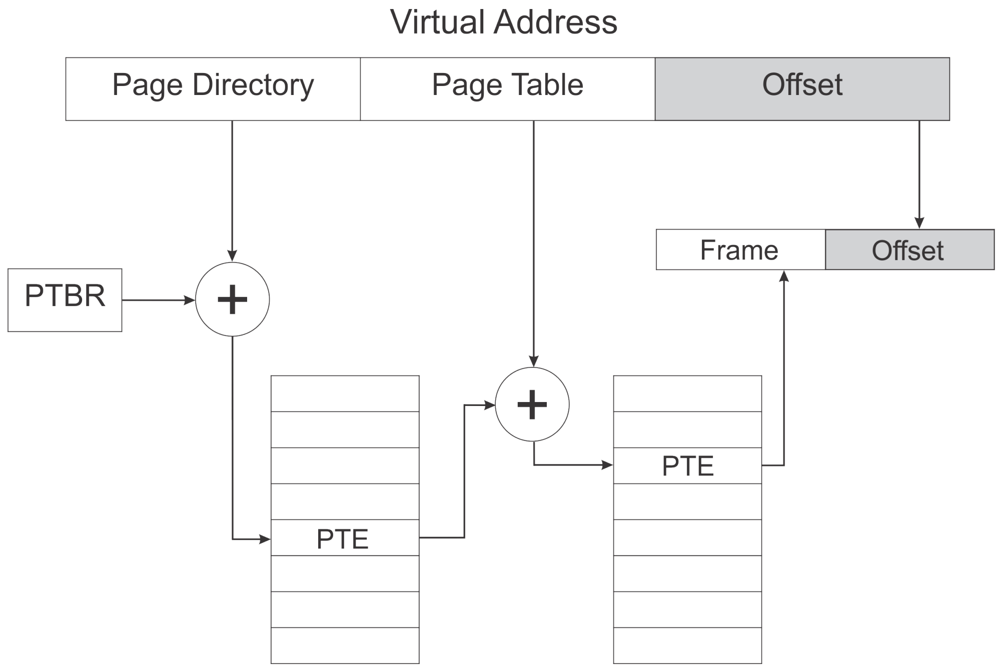
Virtual address : 0xc02005



2^Number of bits in offset = Size of page

0000 0000 1100 0000 0010 0000 0000 0101

Virtual address : 0xc02005

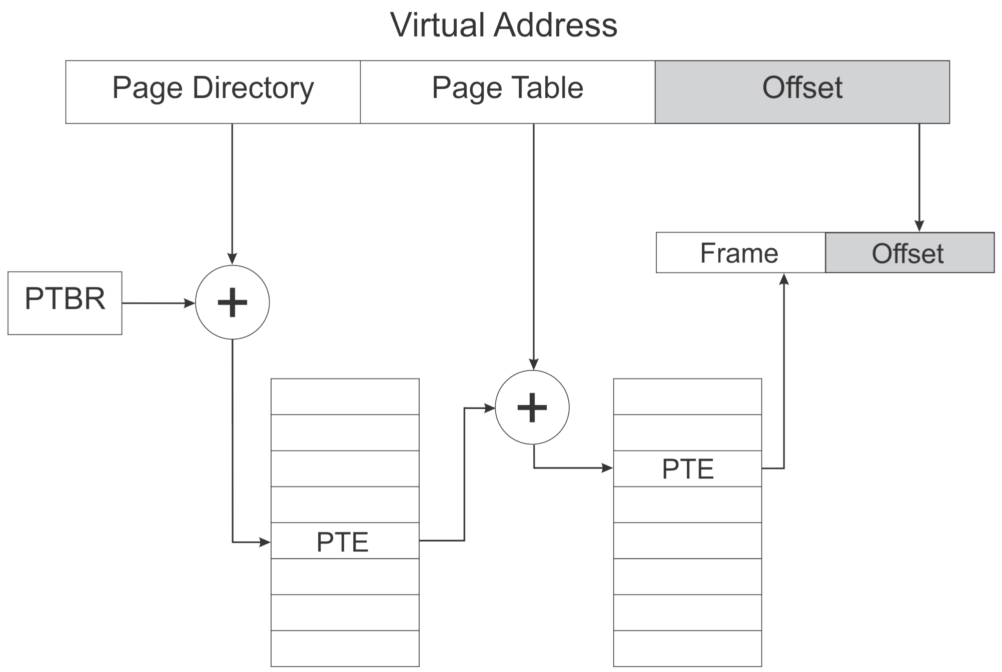


2^Number of bits in offset = Size of page

2^Number of bits in offset = 4kB

0000 0000 1100 0000 0010 0000 0000 0101

Virtual address : 0xc02005



0000 0000 1100 0000 0010 0000 0000 0101

2^Number of bits in offset = Size of page

2^Number of bits in offset = 4kB

Number of bits in offset = 12

What do the bits represent ?

- 0000 0000 1100 0000 0010 0000 0000 0101

What do the bits represent ?

- 0000 0000 1100 0000 0010 0000 0000 0101

0000 0000 11

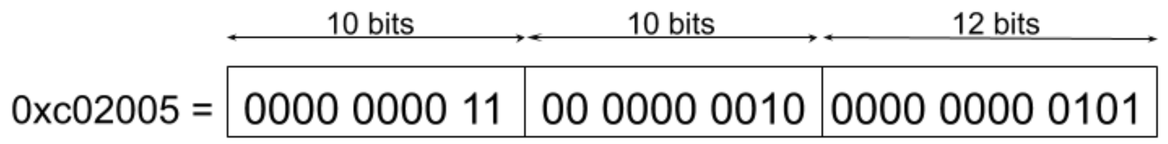
00 0000 0010

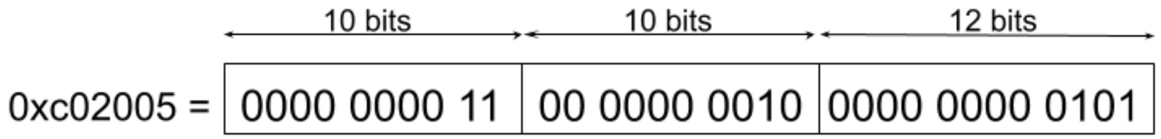
0000 0000 10

What do the bits represent ?

- 0000 0000 1100 0000 0010 0000 0000 0101

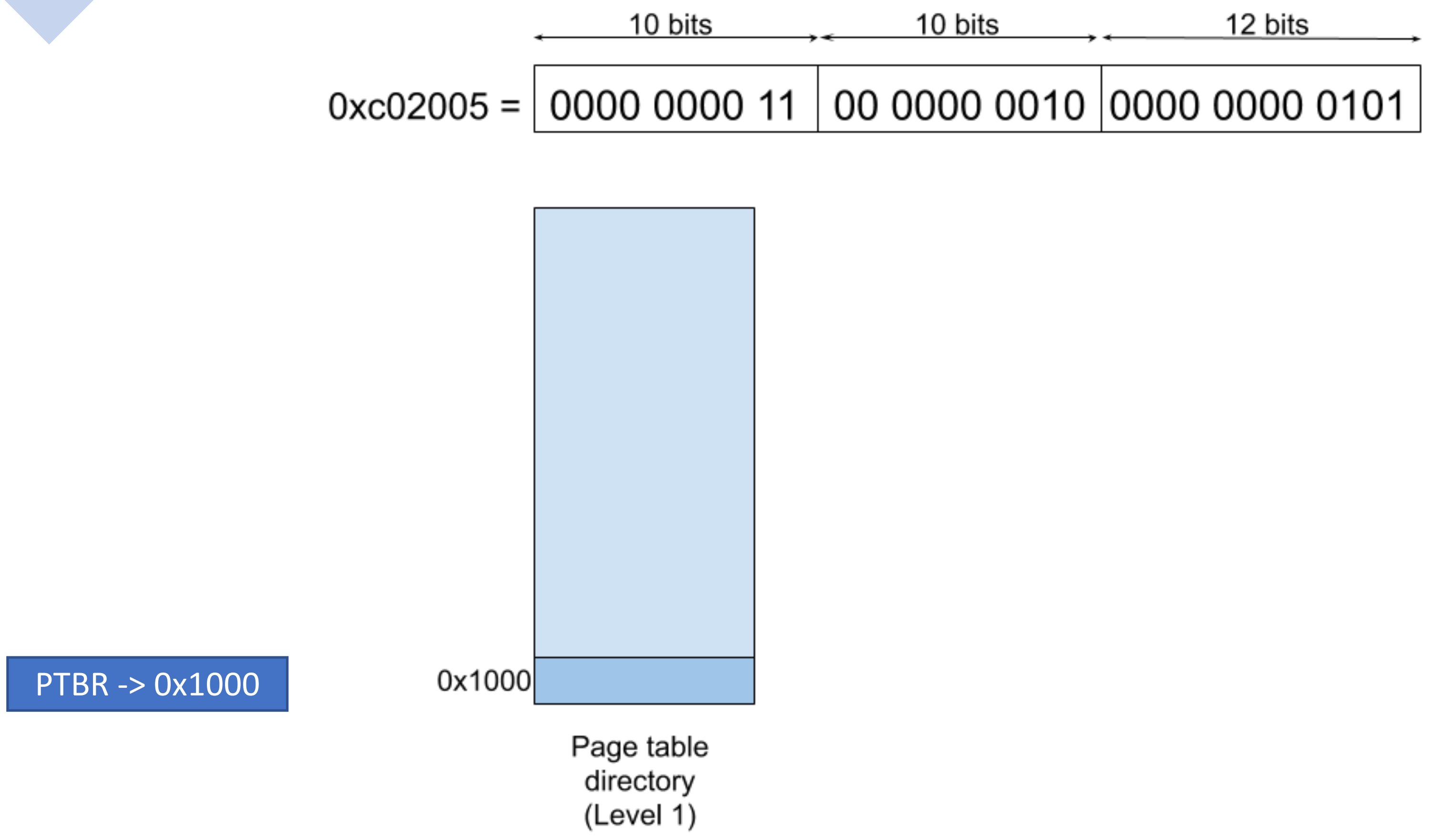
0000 0000 11	00 0000 0010	0000 0000 10
Offset for Page Table Directory (Level 1)	Offset for Page Table (Level 2)	Offset for a byte/word within a page.


10 bits
10 bits
12 bits
0xc02005 =
0000 0000 11
00 0000 0010
0000 0000 0101

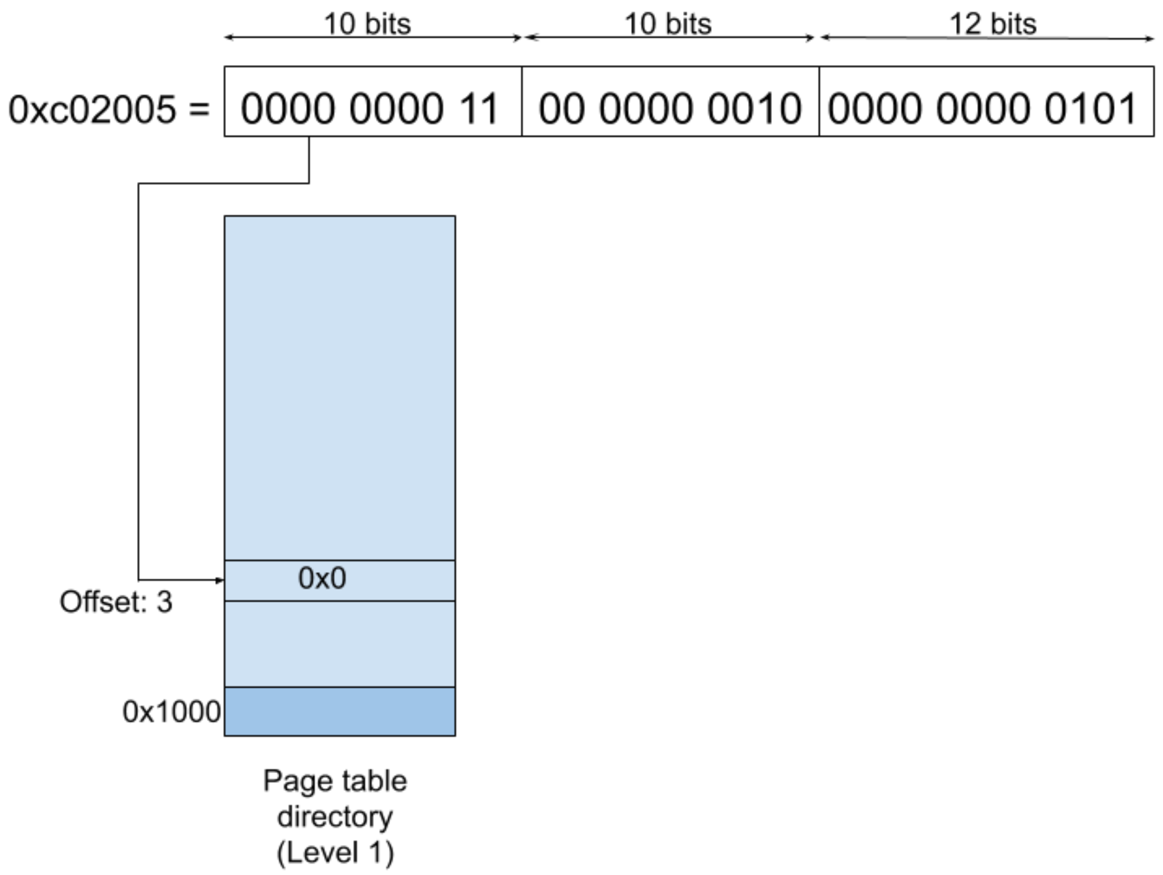


PTBR -> 0x1000

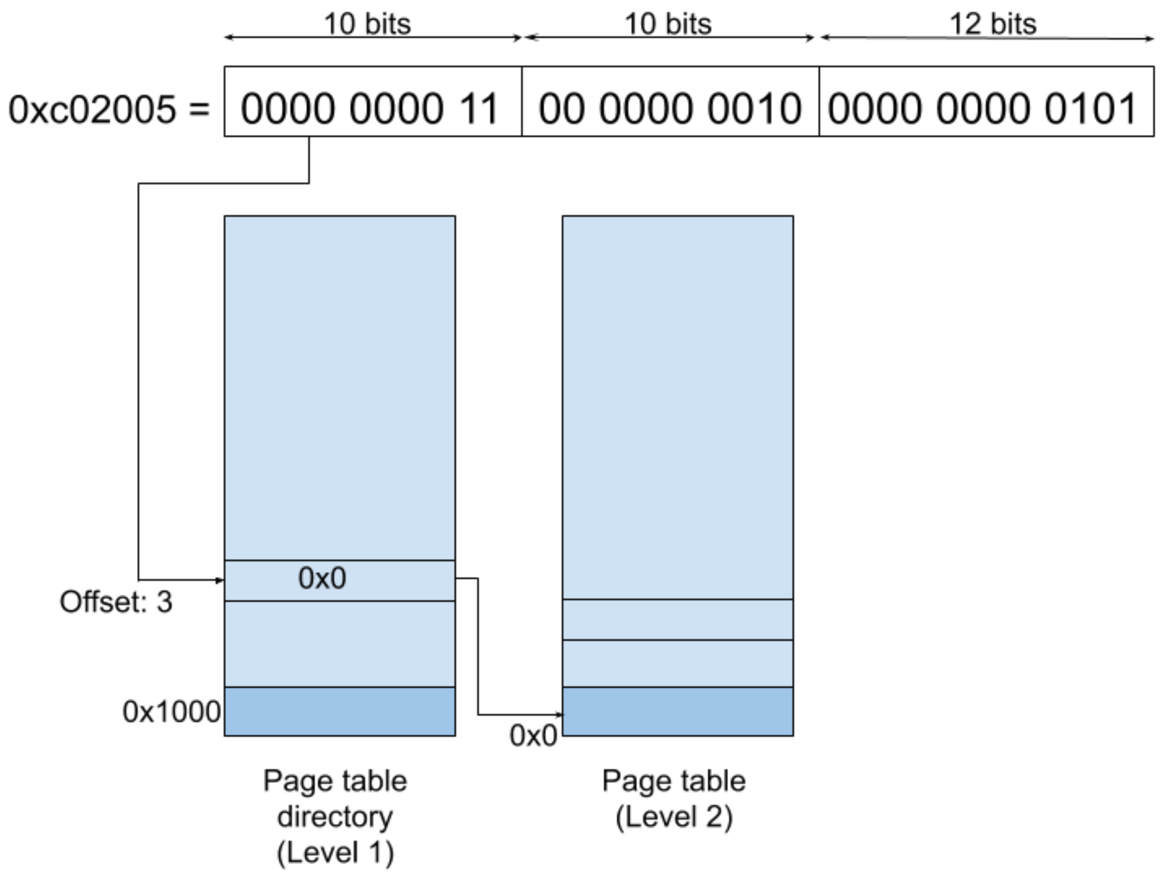
Looking back : Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.



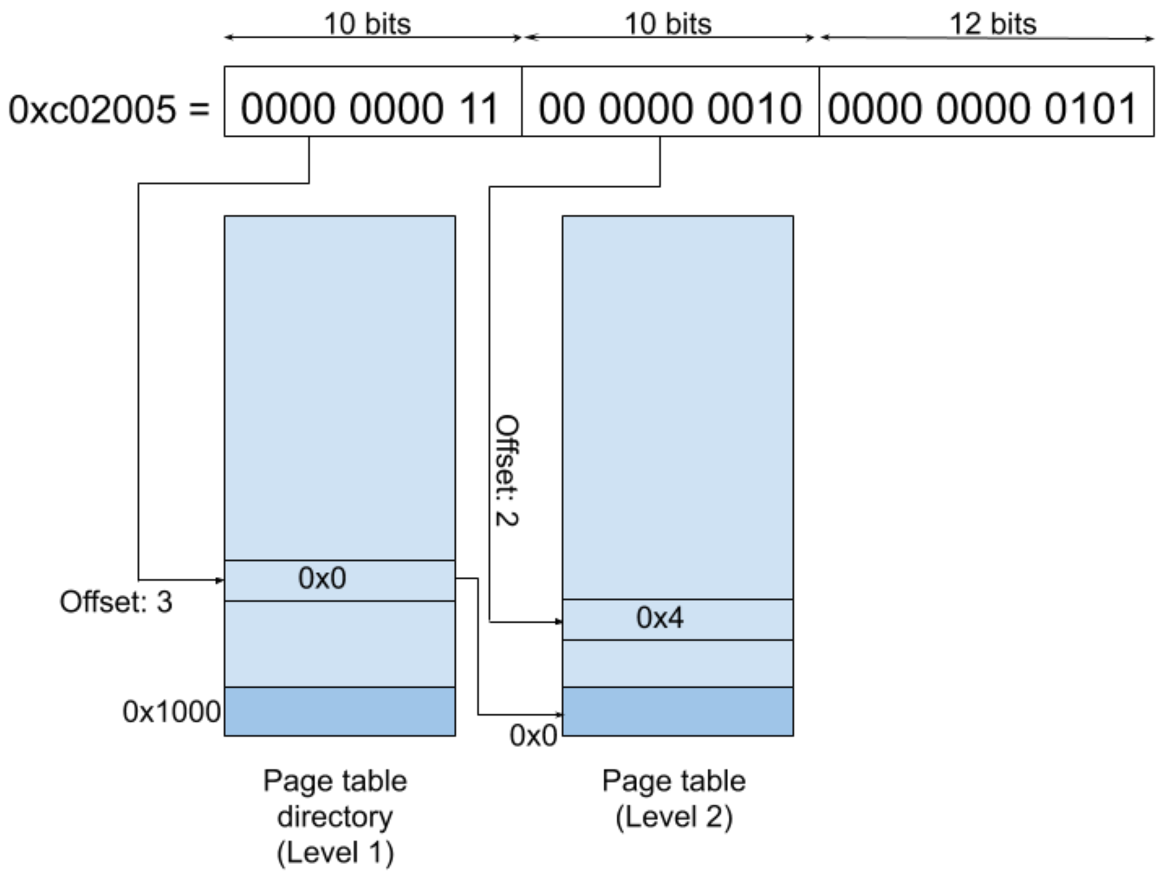
Looking back : Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.



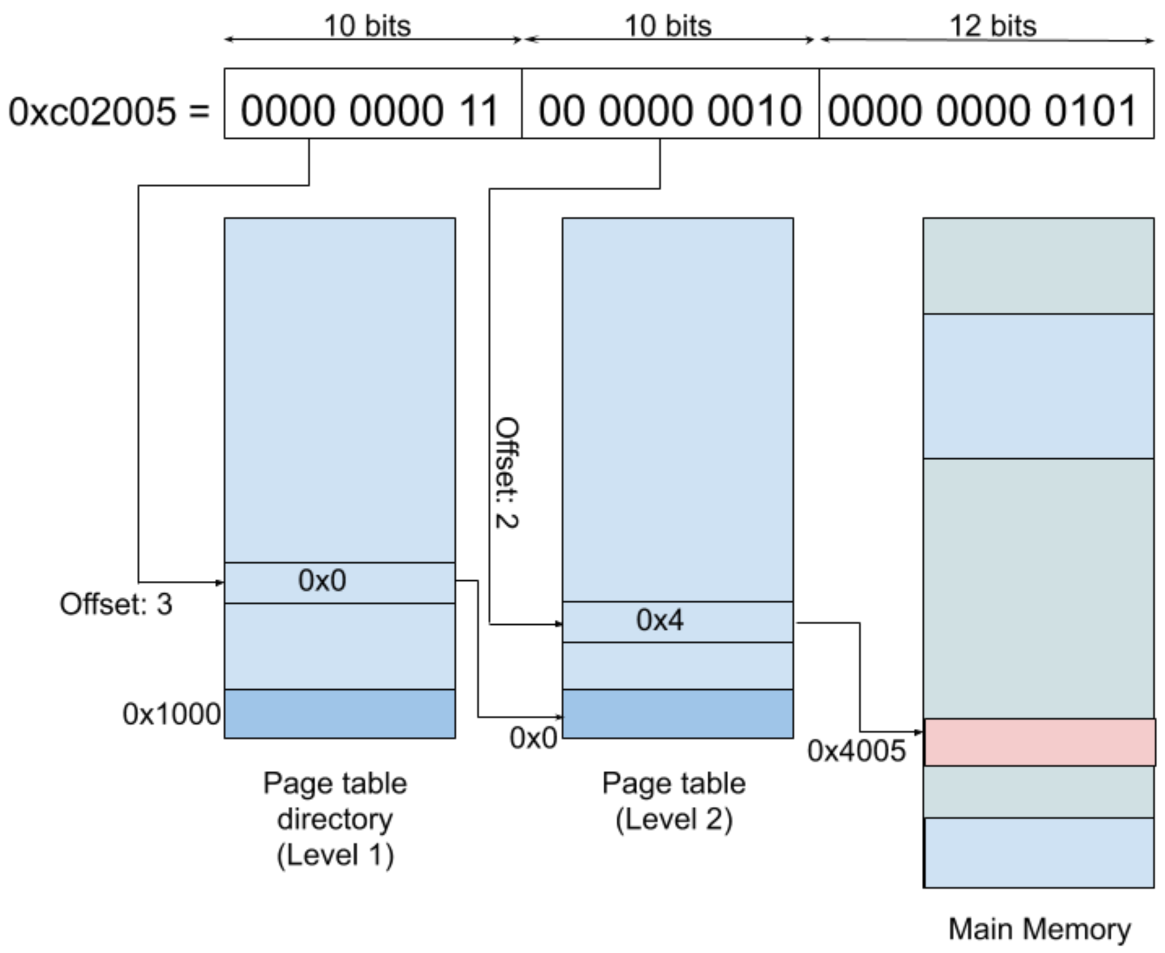
Looking back : Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.



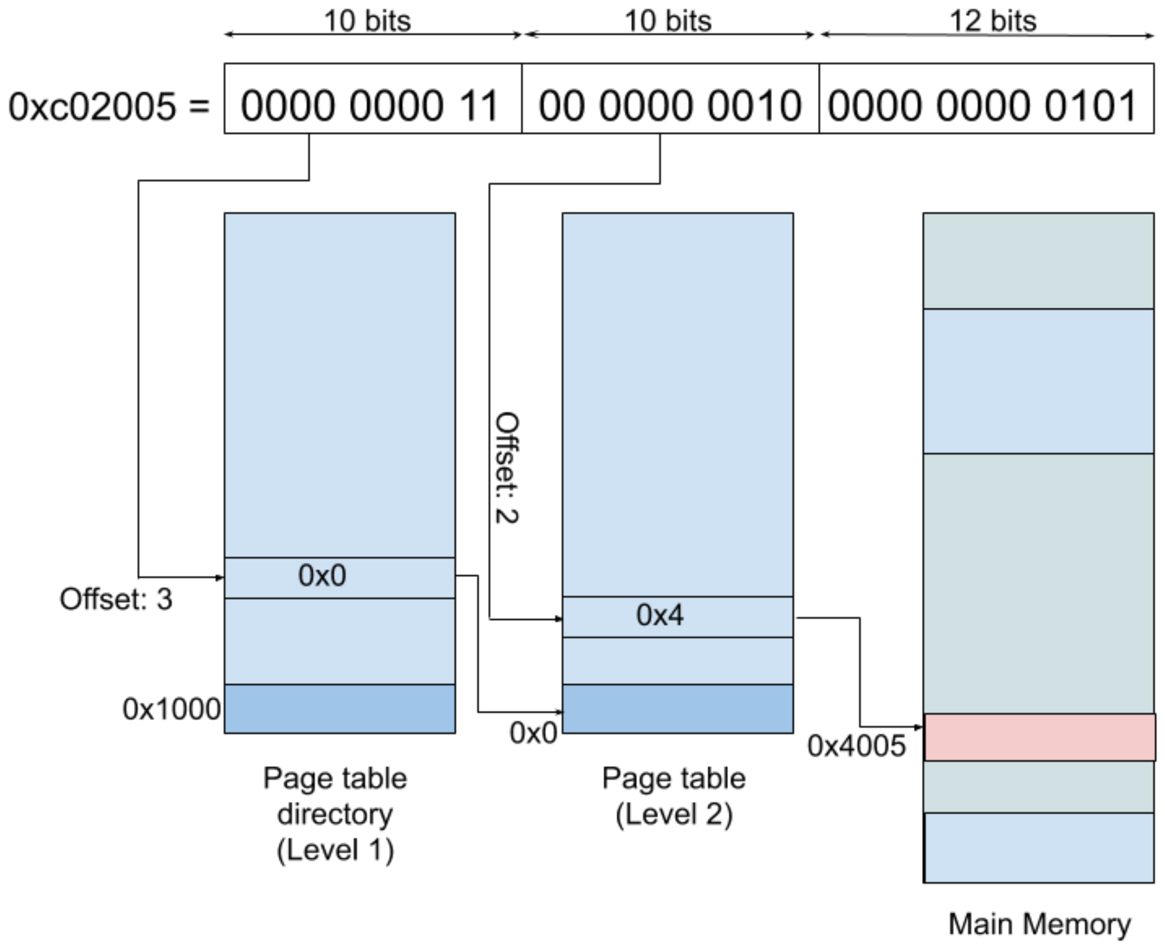
Looking back : Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.



Looking back : Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.



Looking back : Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.



Looking back : Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.

Q: Where are the Page Tables ?

Q: Where are the Page Tables ?

In Main Memory

Q: Main memory access is slow. Can we do anything to make access faster ?

Q: Where are the Page Tables ?

In Main Memory

Q: Main memory access is slow. Can we do anything to make access faster ?

Yes, with Translation Lookaside Buffers

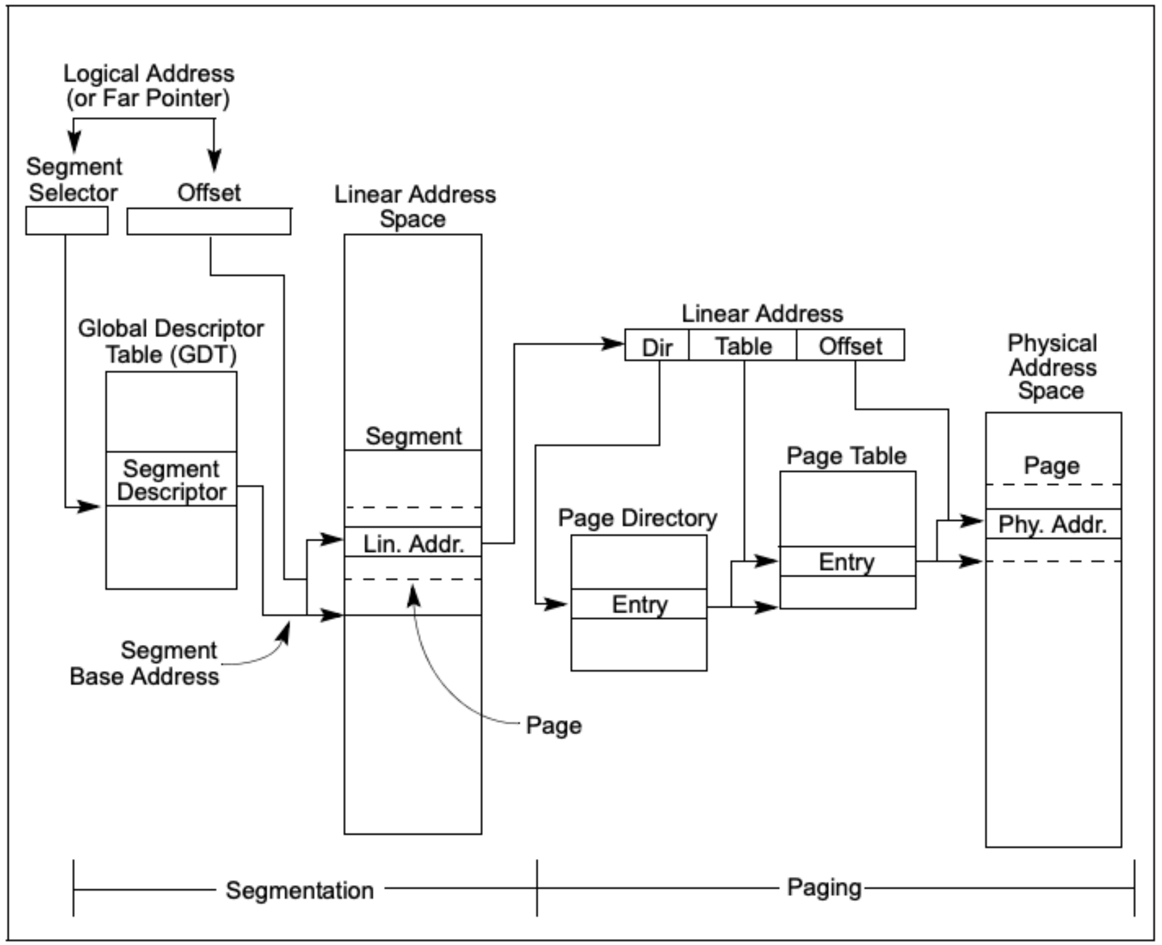
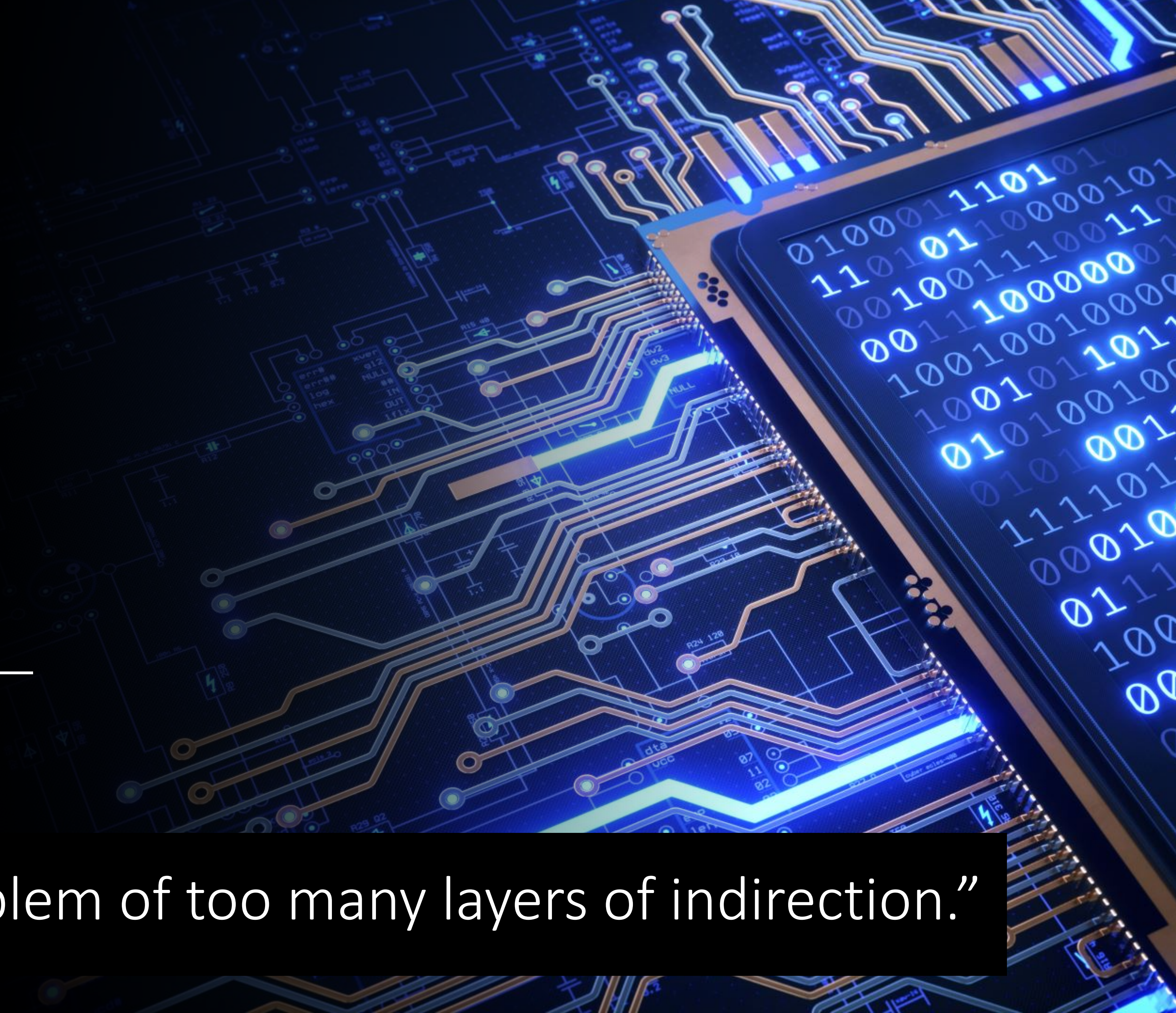


Figure 3-1. Segmentation and Paging

Address translation overview:

Putting everything together



“All problems in computer science can be solved by another level of indirection”

David Wheeler

“...except for the problem of too many layers of indirection.”

Questions