Main Memory

COMP 3361: Operating Systems I

Winter 2015

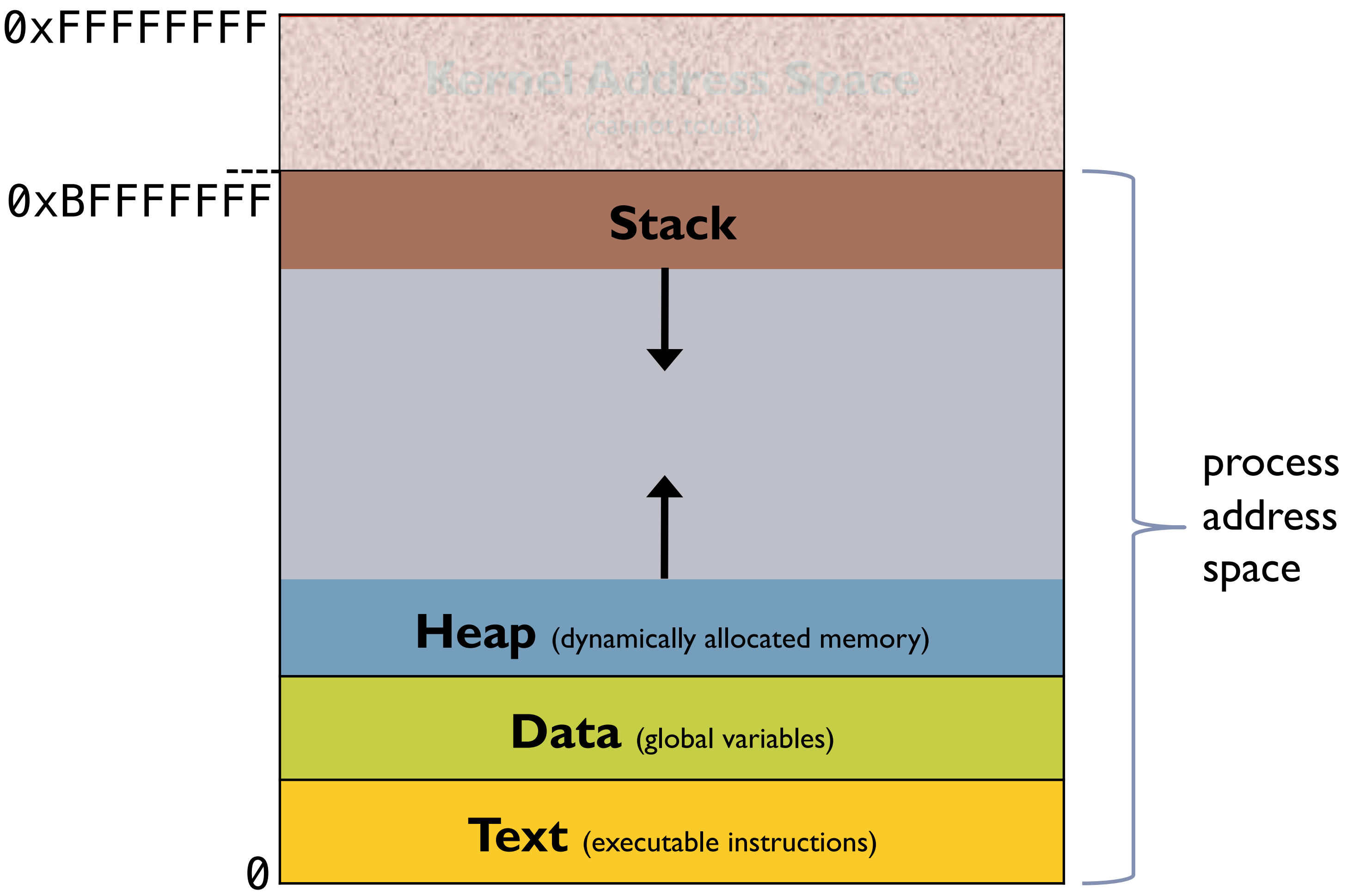
http://www.cs.du.edu/3361

Logical (Virtual) Address Space

- One possible logical address space
  - begins at address zero and ends at (process size - 1)
  - process here includes program code/data, stack, heap, etc.
  - the size of everything must be known in advance so that the size of a process can be calculated

- Another possibility
  - begin at address zero and end at maximum possible address
    - with a 32 bit logical address space, that is 2^{32}-1 = 0xFFFFFFFF
  - design a layout for the process in this space
  - there is ample room for parts to grow

32-bit Logical (Virtual) Address Space



Scanario

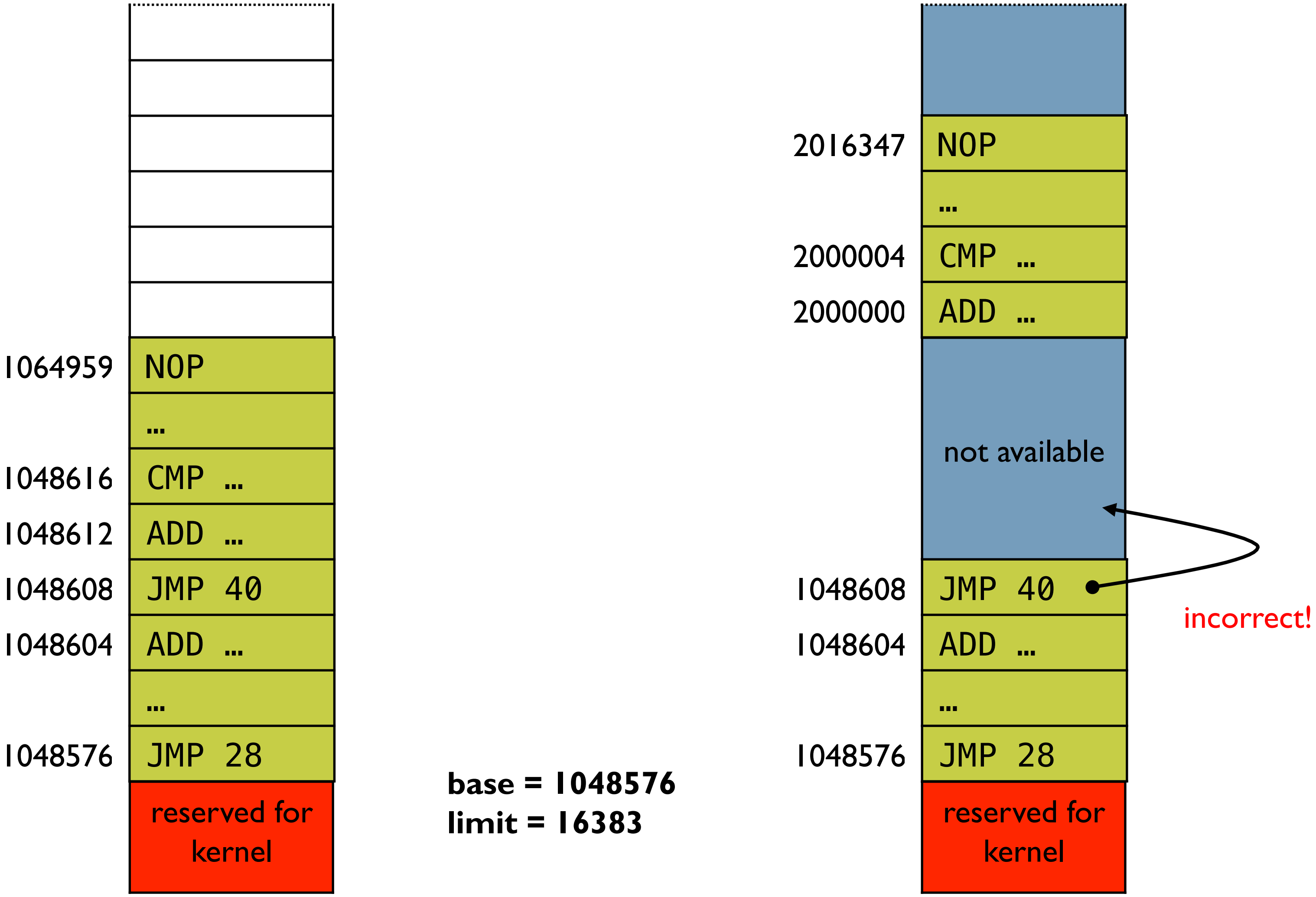
- Process A needs n frames of memory
- There are more than n frames of free memory
- All free areas are less than n frames in size!
- What to do?

Another Program

address	instruction
16383	NOP
	…
40	CMP …
36	ADD …
32	JMP 40
28	ADD …
	…
0	JMP 28

Program C (16KB)

5 Fragmented Process in Memory



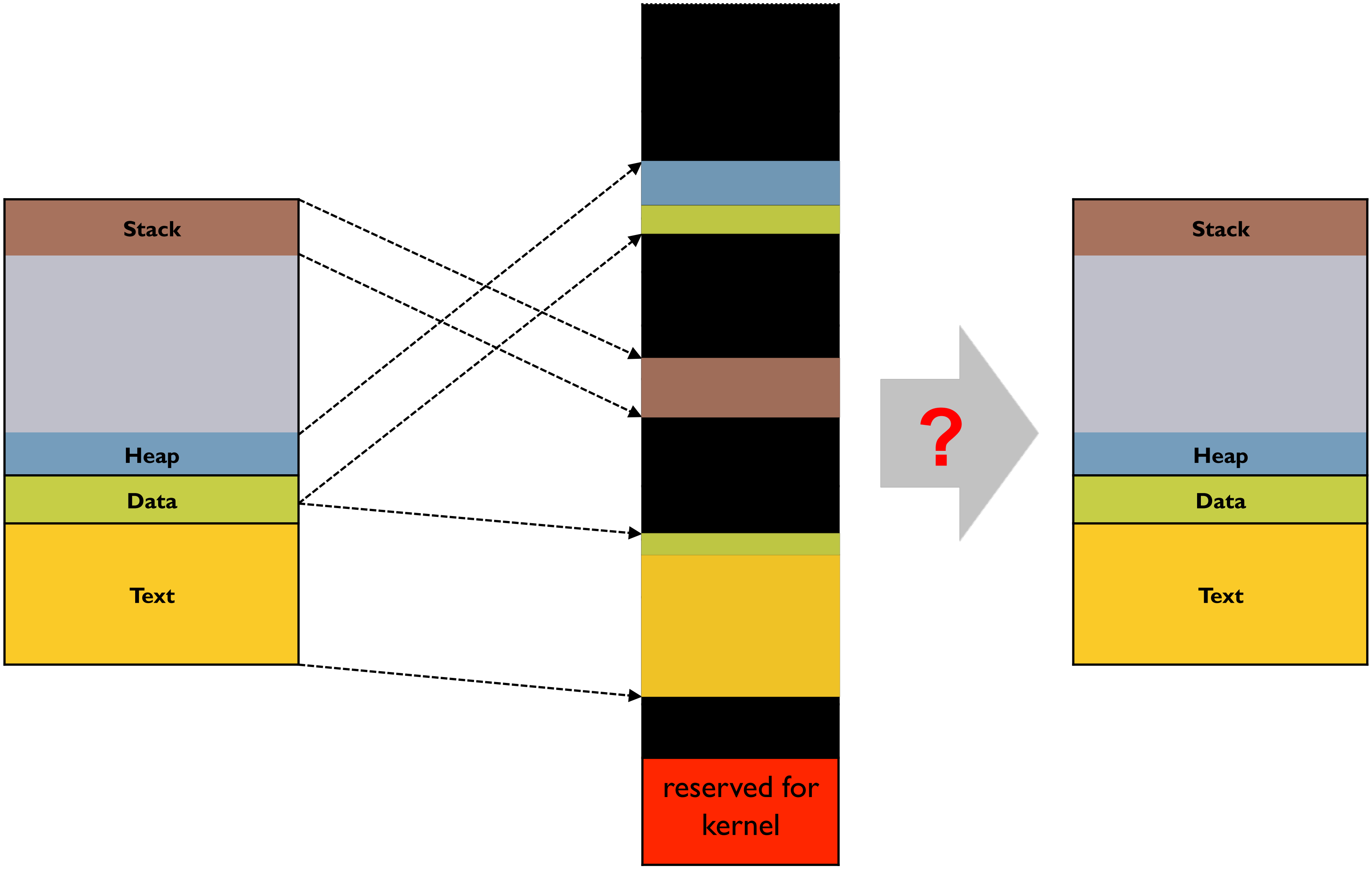
1064959	NOP
	…
1048616	CMP …
1048612	ADD …
1048608	JMP 40
1048604	ADD …
	…
1048576	JMP 28
	reserved for kernel

base = 1048576
limit = 16383

2016347	NOP
	…
2000004	CMP …
2000000	ADD …
	not available
1048608	JMP 40
1048604	ADD …
	…
1048576	JMP 28
	reserved for kernel

incorrect!

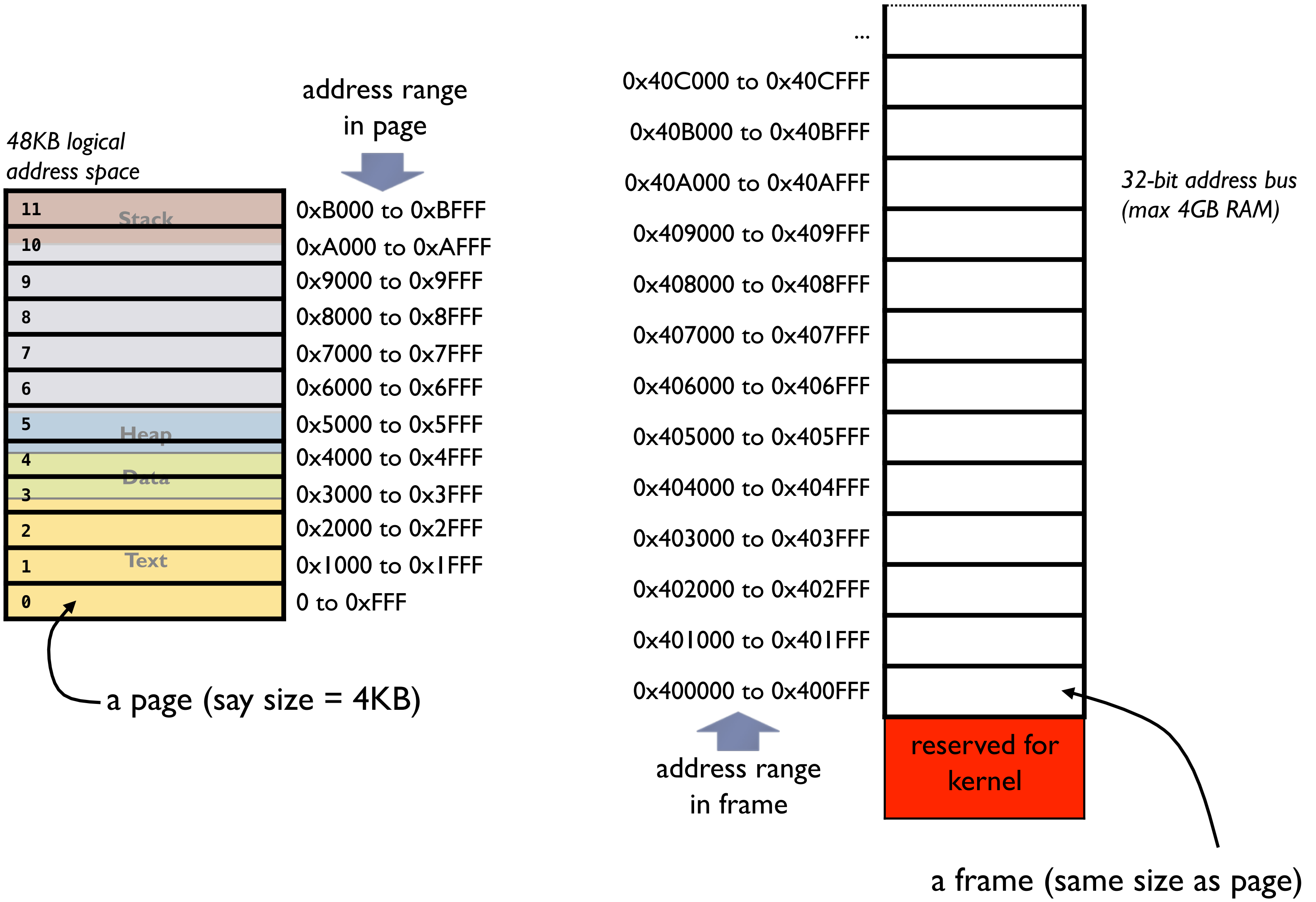
Stitching Process Fragments



Paging

- Divide physical memory into fixed-size allocation units, called **frames**
  - typically 4 KB; 2 MB, 4 MB and 1GB also possible
- Divide logical memory into **pages** of same size as a frame
- Keep track of all free frames
- For a process of *n* pages, find *n* free frames and load process in those frames
  - set up a **page table** to translate logical to physical addresses

Pages and Frames



48KB logical address space

address range in page

11	Stack	0xB000 to 0xBFFF
10		0xA000 to 0xAFFF
9		0x9000 to 0x9FFF
8		0x8000 to 0x8FFF
7		0x7000 to 0x7FFF
6		0x6000 to 0x6FFF
5	Heap	0x5000 to 0x5FFF
4	Data	0x4000 to 0x4FFF
3		0x3000 to 0x3FFF
2		0x2000 to 0x2FFF
1	Text	0x1000 to 0x1FFF
0		0 to 0xFFF

a page (say size = 4KB)

...

0x40C000 to 0x40CFFF

0x40B000 to 0x40BFFF

0x40A000 to 0x40AFFF

0x409000 to 0x409FFF

0x408000 to 0x408FFF

0x407000 to 0x407FFF

0x406000 to 0x406FFF

0x405000 to 0x405FFF

0x404000 to 0x404FFF

0x403000 to 0x403FFF

0x402000 to 0x402FFF

0x401000 to 0x401FFF

0x400000 to 0x400FFF

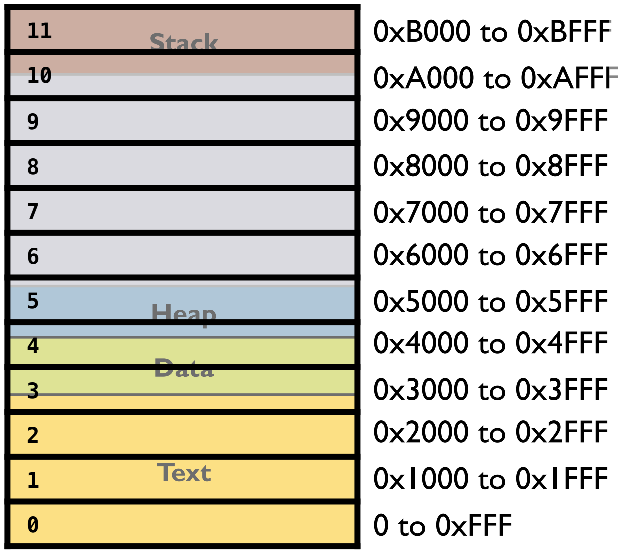
reserved for kernel

address range in frame

32-bit address bus (max 4GB RAM)

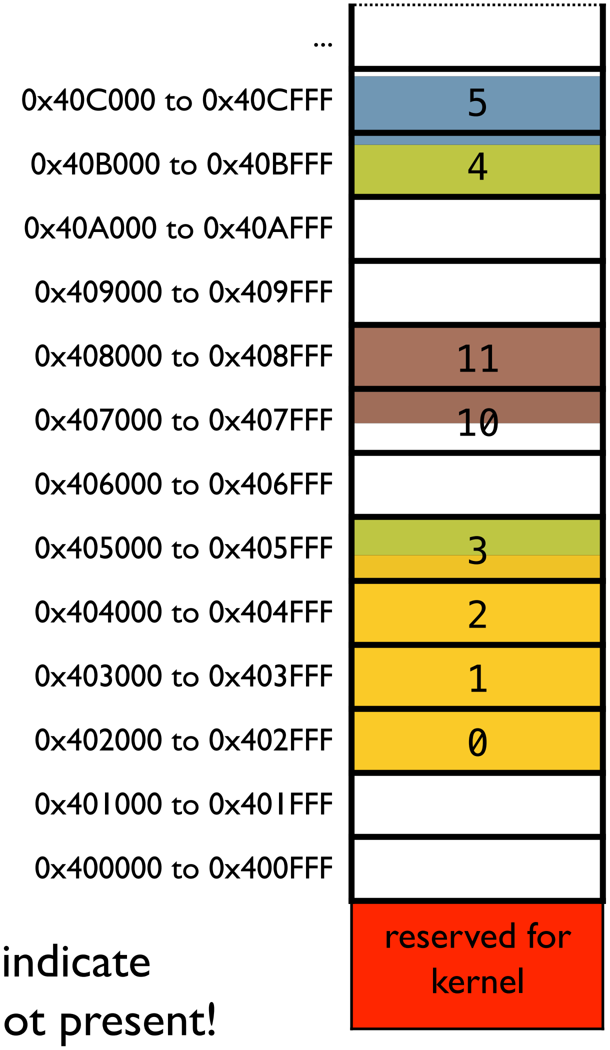
a frame (same size as page)

Page Table



page table

0	0x00402000
1	0x00403000
2	0x00404000
3	0x00405000
4	0x0040B000
5	0x0040C000
6	0x00000000
7	0x00000000
8	0x00000000
9	0x00000000
10	0x00407000
11	0x00408000



logical address	physical address
0x0000	0x402000
0x5010	0x40C010
0x7000	**0x0 (Wrong!)**

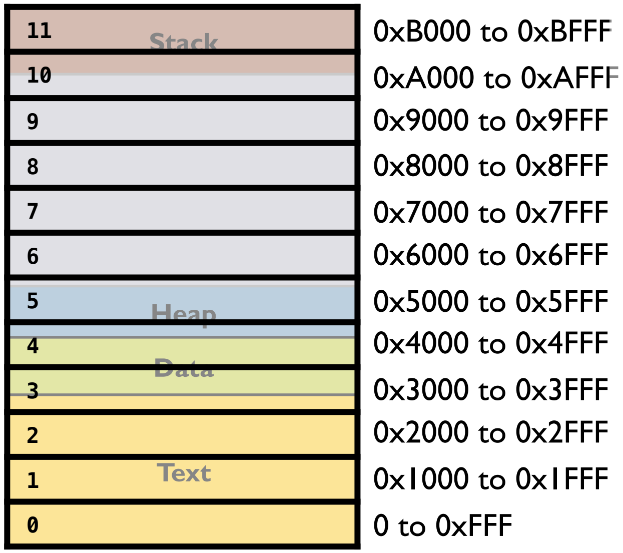
need method to indicate that mapping is not present!

Main Memory

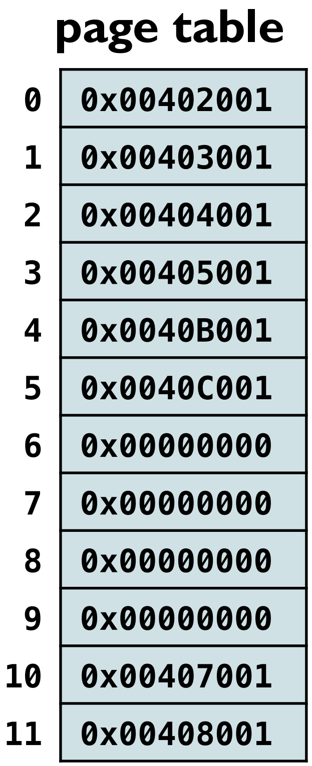
Present Bit

- Observe: if page/frame size is 4KB (0x1000 bytes), then start address of page/frame will always have lower 12 bits as zero
  - 0x0, 0x1000, 0x2000, 0x3000, …
- We can store additional information in those 12 bits
- When reading it as an address, we logically AND the page table value with 0xFFFFF000
- Lets use bit 0 to signify if mapping is present or not
  - 0: not present; 1: present

Page Table with Present Bit

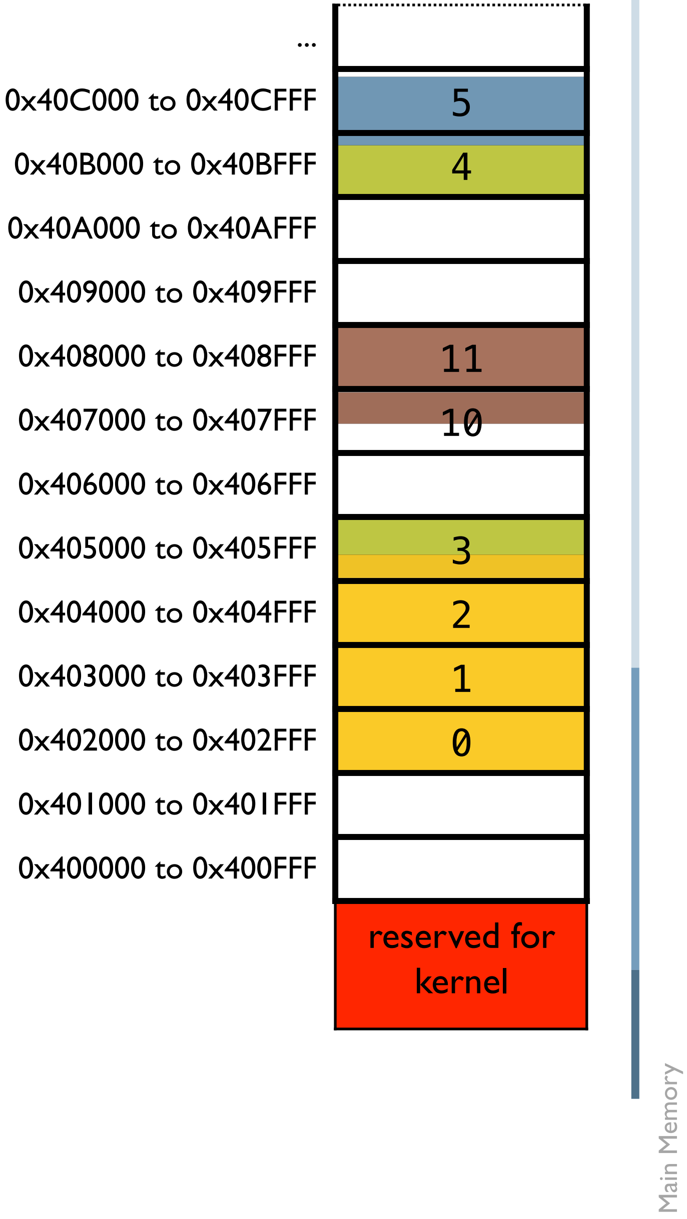


11	0xB000 to 0xBFFF
10	0xA000 to 0xAFFF
9	0x9000 to 0x9FFF
8	0x8000 to 0x8FFF
7	0x7000 to 0x7FFF
6	0x6000 to 0x6FFF
5	0x5000 to 0x5FFF
4	0x4000 to 0x4FFF
3	0x3000 to 0x3FFF
2	0x2000 to 0x2FFF
1	0x1000 to 0x1FFF
0	0 to 0xFFF



page table

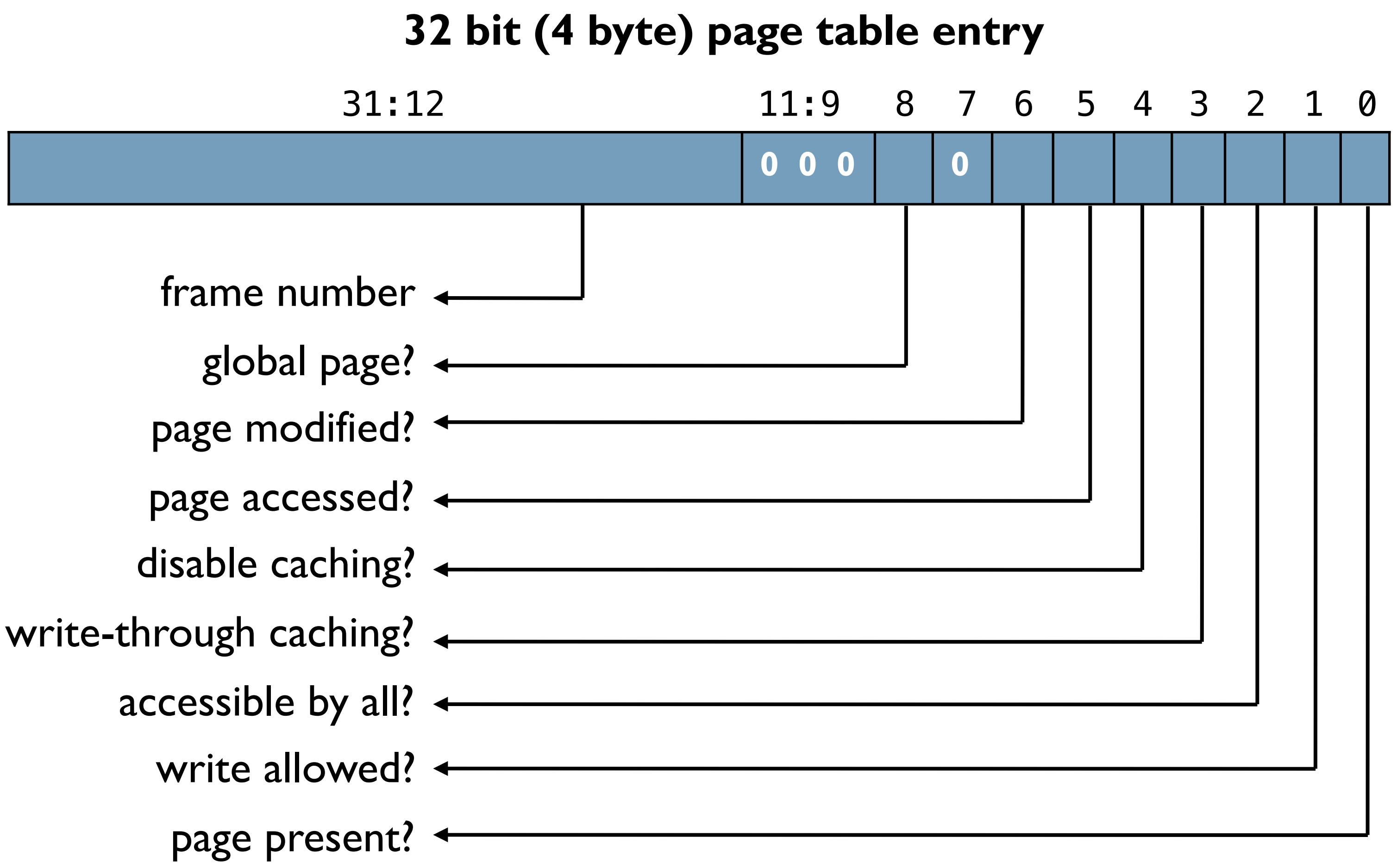
0	0x00402001
1	0x00403001
2	0x00404001
3	0x00405001
4	0x0040B001
5	0x0040C001
6	0x00000000
7	0x00000000
8	0x00000000
9	0x00000000
10	0x00407001
11	0x00408001



...	
0x40C000 to 0x40CFFF	5
0x40B000 to 0x40BFFF	4
0x40A000 to 0x40AFFF	
0x409000 to 0x409FFF	
0x408000 to 0x408FFF	11
0x407000 to 0x407FFF	10
0x406000 to 0x406FFF	
0x405000 to 0x405FFF	3
0x404000 to 0x404FFF	2
0x403000 to 0x403FFF	1
0x402000 to 0x402FFF	0
0x401000 to 0x401FFF	
0x400000 to 0x400FFF	
	reserved for kernel

logical address	physical address
0x0000	0x402000
0x5010	0x40C010
0x7000	no mapping

12 Other Information in Page Table Entry



set (1) if yes, otherwise clear (0)

Dissecting a Logical Address

Page size = 4KB
32-bit logical address: **0x00001A21**

page table

0	...
1	0x12ABB107
2	...
3	...
4	...
...	...
1024^2–1	...

Page table entry to look at: **0x00001A21 >> 12 = 0x1**

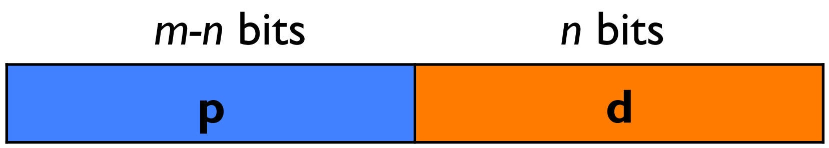
Offset within frame: **0x00001A21 & 0x00000FFF = 0xA21**

Physical address: **(0x12ABB107 & 0xFFFFF000) + 0xA21 = 0x12ABBA21**

global
accessible by all
write allowed
present

Address Translation Scheme

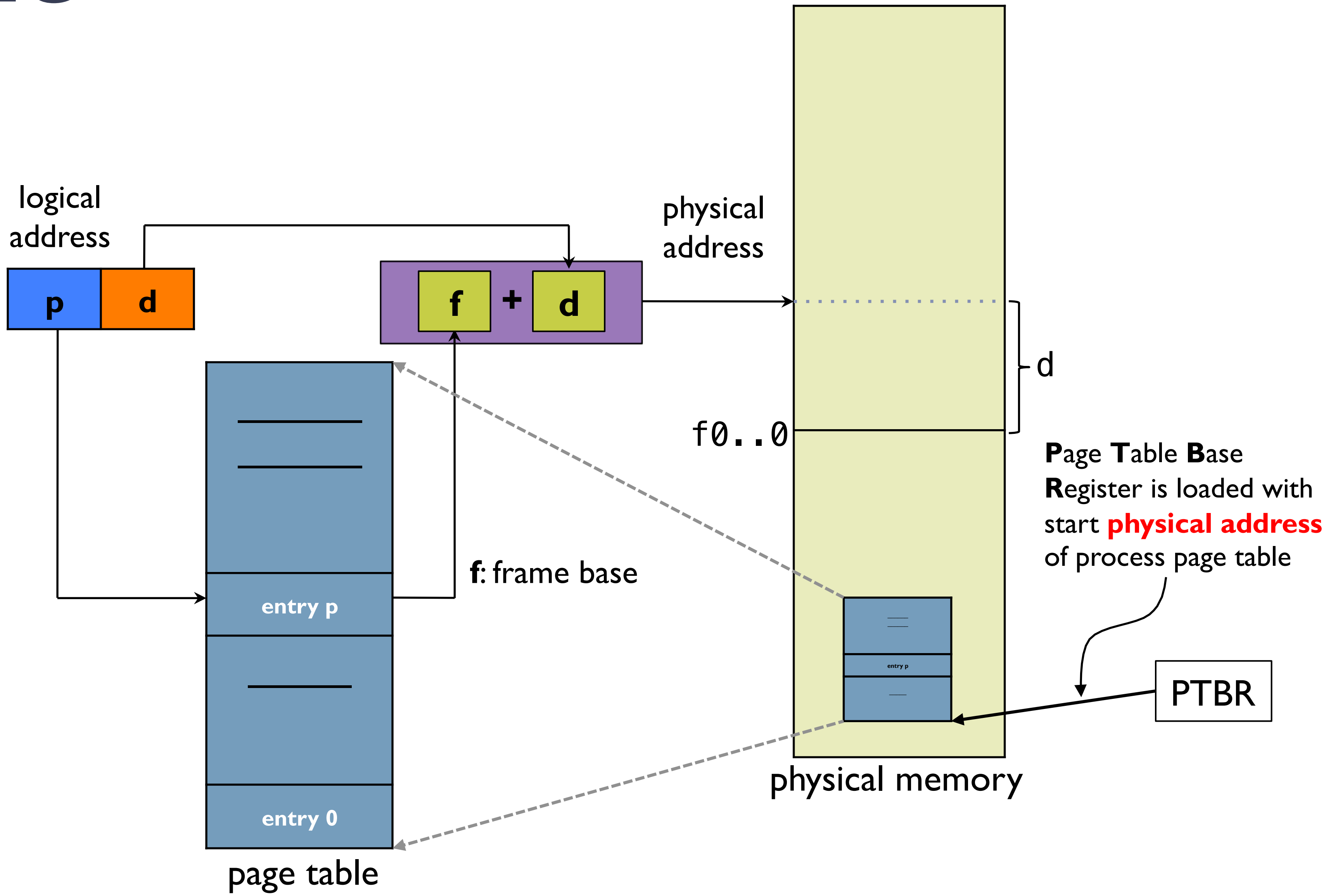
- Logical address is divided into:



  - **page number (*p*)**: use as an index into a page table and get base address of frame corresponding to page
  - **page offset (*d*)**: add to base address to determine the physical memory address

- For a given logical address space of 2^m and page size 2^n
  - higher *(m-n)* bits is page number; lower *n* bits is page offset

Paging in Hardware



Page **T**able **B**ase **R**egister is loaded with start **physical address** of process page table

16

Making Lookups Faster

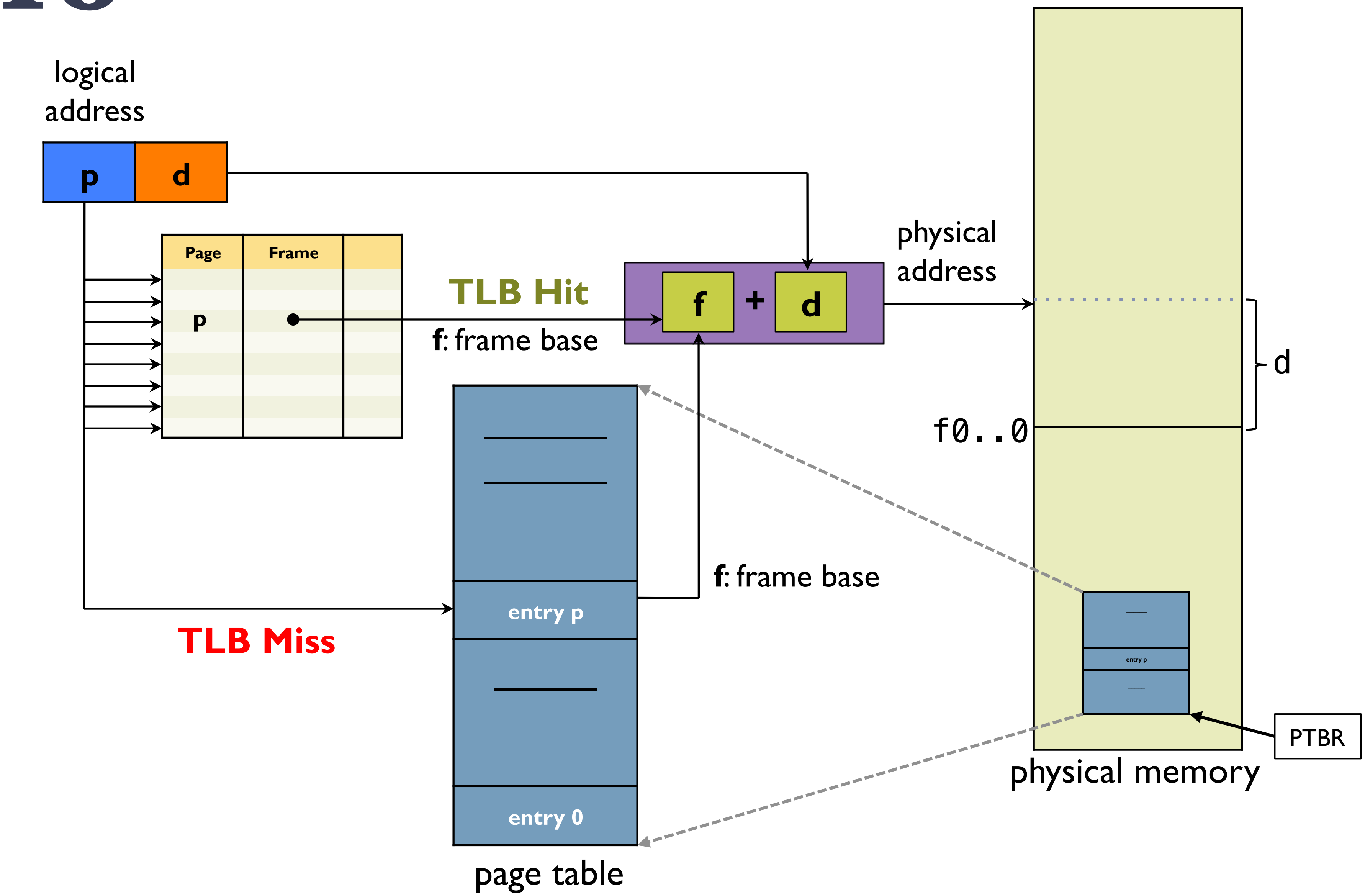
- Page table is kept in main memory
  - page-table base register (PTBR) points to the page table
- Every data/instruction access requires two memory accesses
  - one for the page table and one for the data/instruction
- To make it faster, use a special fast-lookup hardware cache called **Translation Look-aside Buffer (TLB)**
  - associative, high speed memory

Associative Memory

Page	Frame	other configuration

- High speed parallel search
- Input: page number
- Lookup: parallel lookup of the associative array for the page number
- Output: frame number
  - or a ***TLB miss***: frame reference must be obtained from page table and the TLB must be updated
- TLB flush/update necessary during process switch

18 Paging with TLB

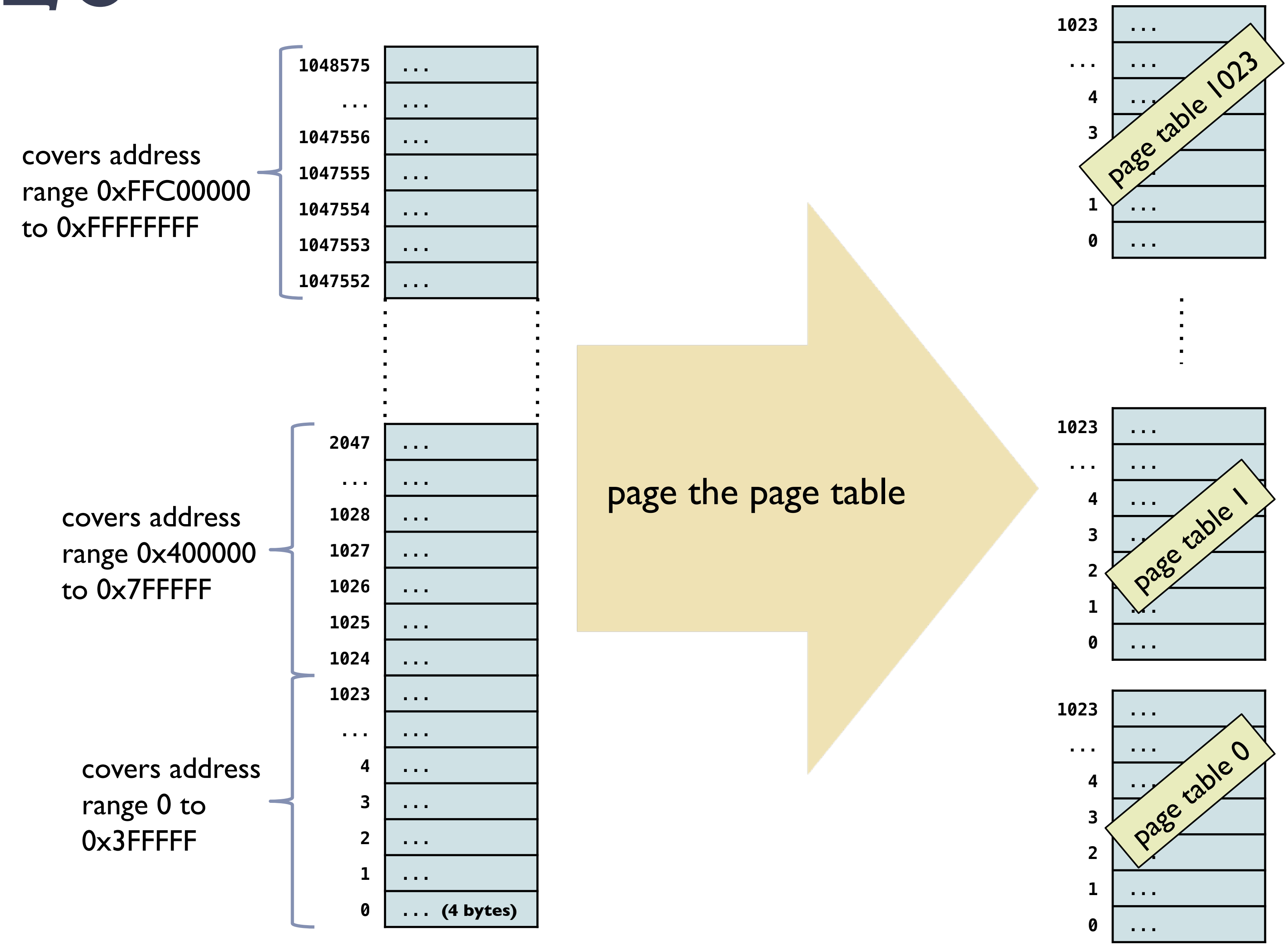


19

Large Page Tables

- For a 32 bit logical address space and 4 KB page size, there can be 1,048,576 pages
- Each page needs a page table entry
- Each entry is 4 bytes
- A page table needs 1048576 x 4 bytes = 4 MB of memory
- **That's 4 MB for each process!**

Paging the Page Table



covers address range 0xFFC00000 to 0xFFFFFFFF

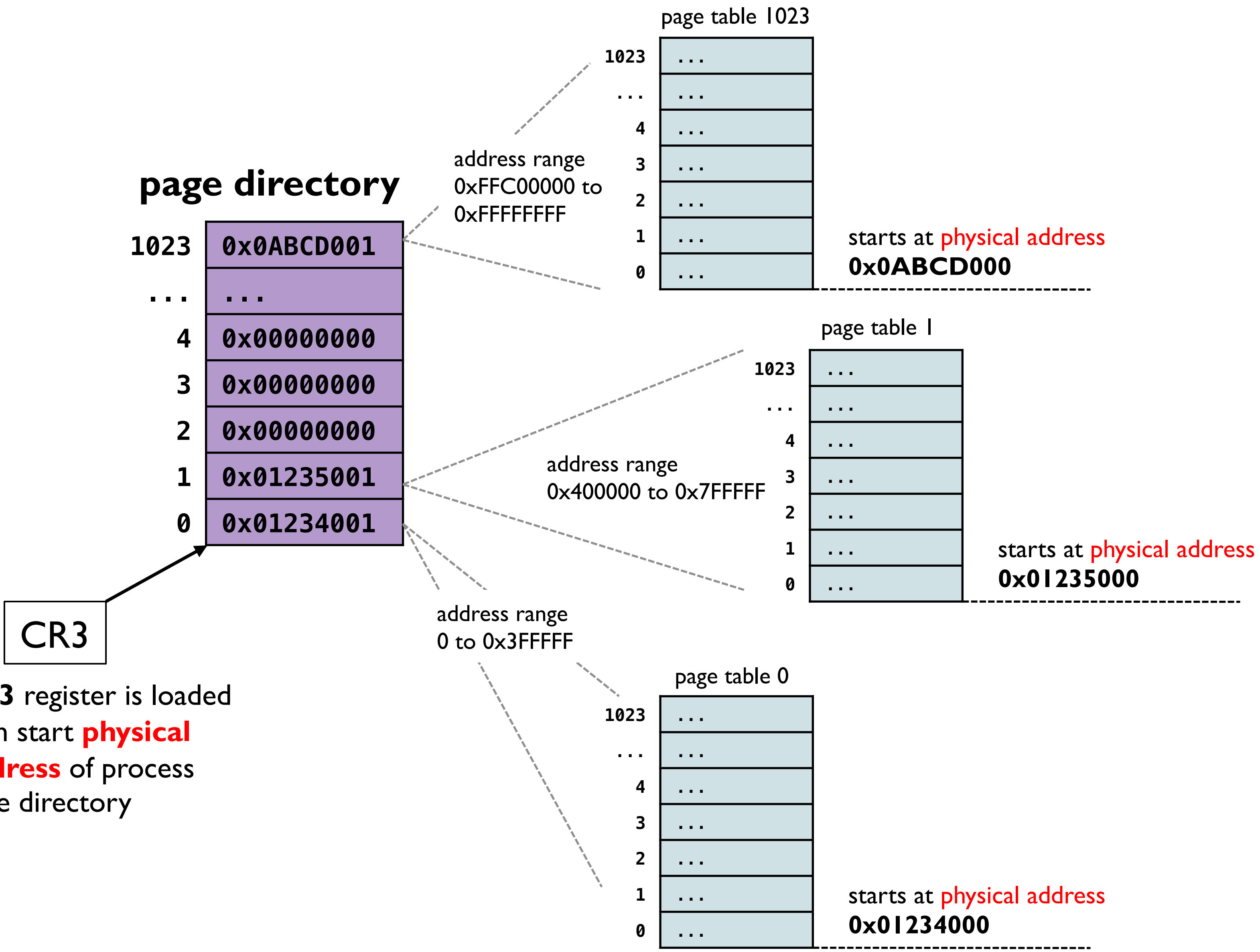
covers address range 0x400000 to 0x7FFFFF

covers address range 0 to 0x3FFFFF

page the page table

21

Two-Level Paging



page directory

1023	0x0ABCD001
...	...
4	0x00000000
3	0x00000000
2	0x00000000
1	0x01235001
0	0x01234001

CR3

CR3 register is loaded with start **physical address** of process page directory

address range 0xFFC00000 to 0xFFFFFFFF

page table 1023

1023	...
...	...
4	...
3	...
2	...
1	...
0	...

starts at physical address **0x0ABCD000**

address range 0x400000 to 0x7FFFFF

page table 1

1023	...
...	...
4	...
3	...
2	...
1	...
0	...

starts at physical address **0x01235000**

address range 0 to 0x3FFFFF

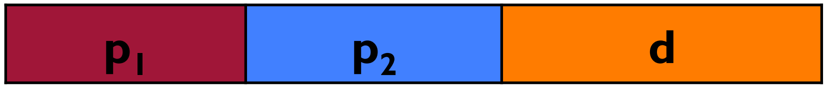
page table 0

1023	...
...	...
4	...
3	...
2	...
1	...
0	...

starts at physical address **0x01234000**

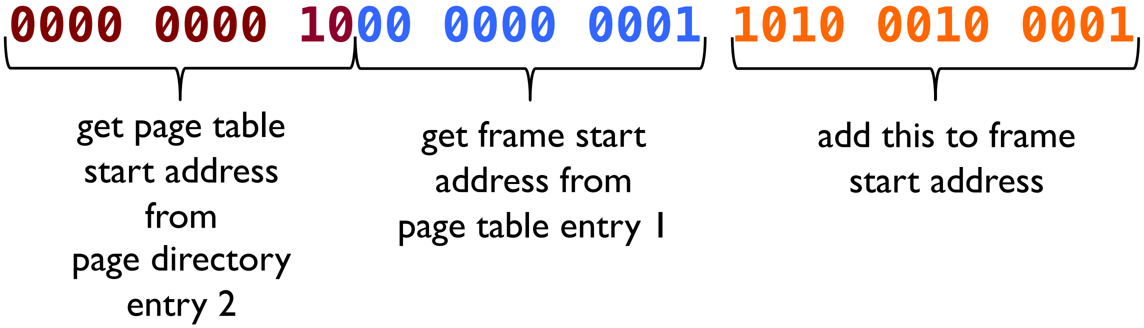
Two-Level Paging (contd.)

- A logical address is broken down as follows:



where p_2 is the index into the page table pointed by the the page directory entry p_1

32-bit logical address: **`0x00801A21`**



`0000 0000 1000 0000 0001 1010 0010 0001`

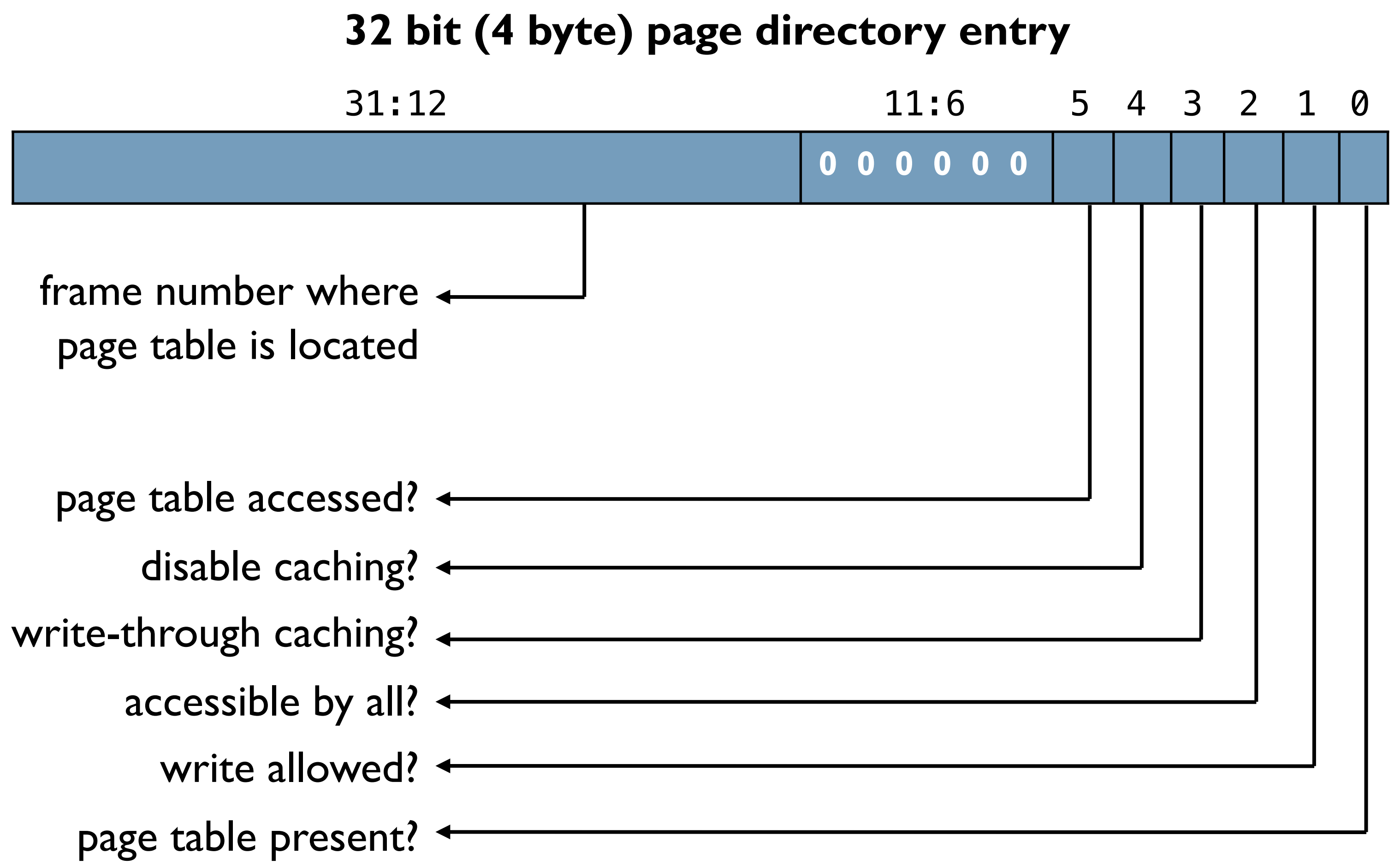
get page table start address from page directory entry 2

get frame start address from page table entry 1

add this to frame start address

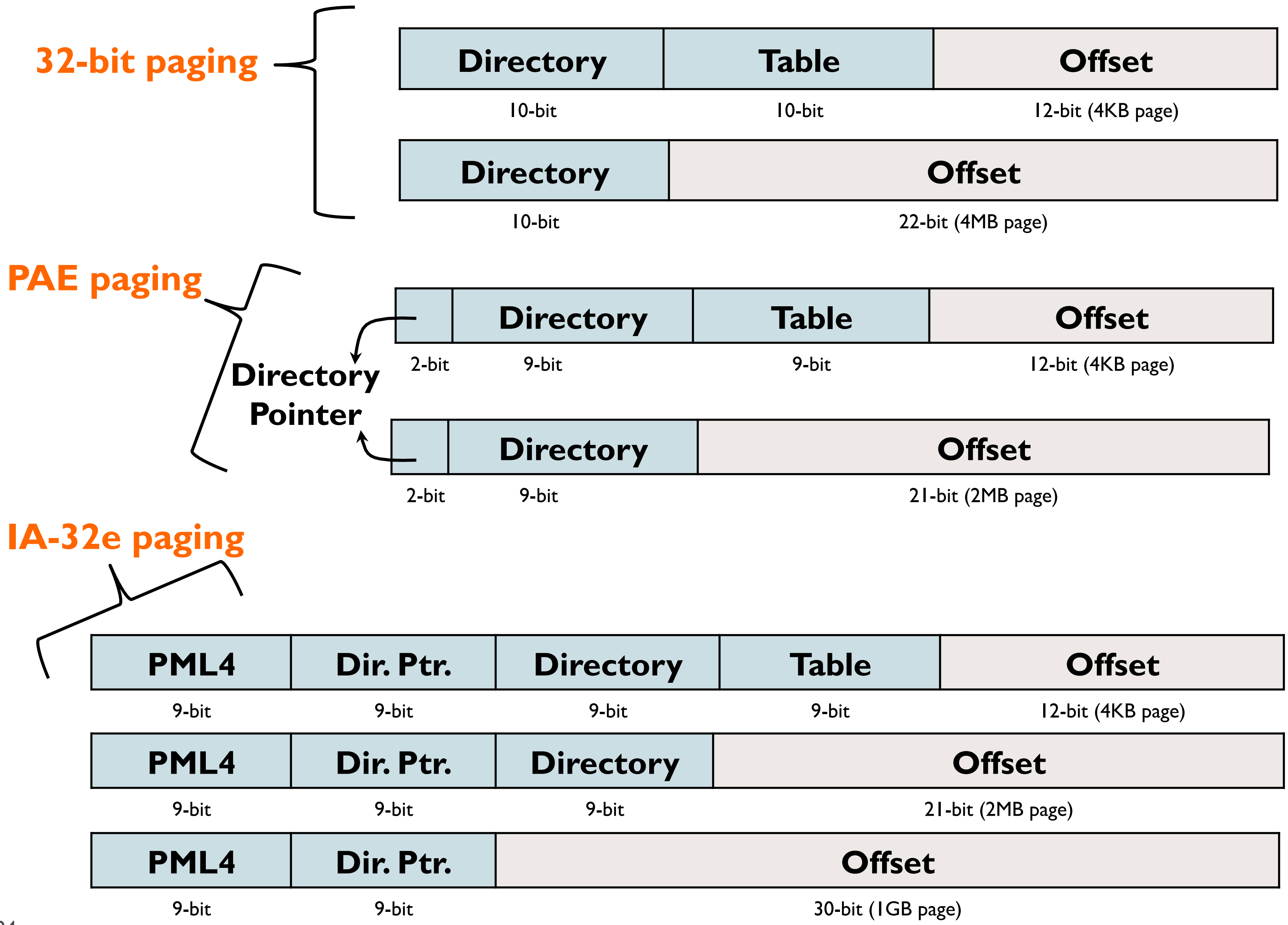
23

Page Directory Entry



set (1) if yes, otherwise clear (0)

x86 Paging Modes



32-bit paging

Directory	Table	Offset
10-bit	10-bit	12-bit (4KB page)

Directory	Offset
10-bit	22-bit (4MB page)

PAE paging

Directory Pointer

	Directory	Table	Offset
2-bit	9-bit	9-bit	12-bit (4KB page)

	Directory	Offset
2-bit	9-bit	21-bit (2MB page)

IA-32e paging

PML4	Dir. Ptr.	Directory	Table	Offset
9-bit	9-bit	9-bit	9-bit	12-bit (4KB page)

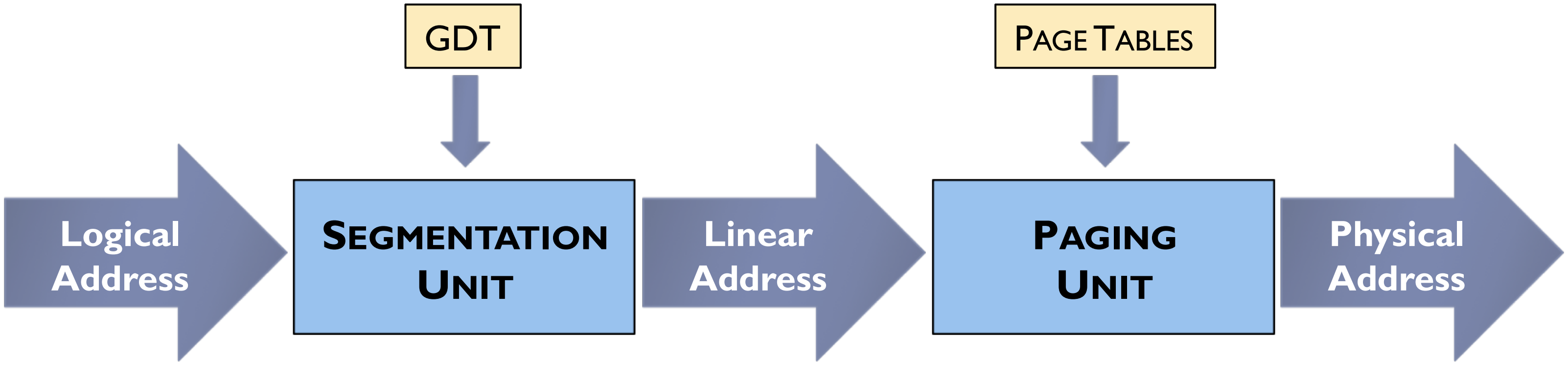
PML4	Dir. Ptr.	Directory	Offset
9-bit	9-bit	9-bit	21-bit (2MB page)

PML4	Dir. Ptr.	Offset
9-bit	9-bit	30-bit (1GB page)

Inverted Page Table

- One entry for each frame in memory
- Each entry consists of
  - the page stored in that frame
  - the process that owns that page
- Decreases memory needed to store each page table
  - only one page table in a system
- Increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to a few page table entries

Address Translation Full Chain



Note: Flat Segmentation (segments span entire address space) is typical these days

References

- Chapter 3.3, Modern Operating Systems, A. Tanenbaum and H. Bos, 4th Edition.