Memory Management

OPERATING SYSTEMS COURSE
THE HEBREW UNIVERSITY
SPRING 2022

Basic Numbers

- Basic units:
  - KB = 2^{10} bytes
  - MB = 2^{20} bytes
  - GB = 2^{30} bytes

- Basic calculations:
  - 4GB / 8KB = $(2^2 / 2^3) * (2^{30} / 2^{10}) = 2^{(20-1)} = 2^{19}$
  - How many numbers can 8 digit number present?
    - 2^8 numbers (values between 0 to $2^8 - 1$)
  - What is the decimal value of 1101?
    - $1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 1 * 2^3 = 13$

CPU

- Can only use data that is stored in registers and memory.
- Executes a set of instructions:
  - Data handling – set a register, store (register in memory), load (from memory to register)
  - Arithmetic operations on registers - Bitwise operations, compare, and basic mathematics operations.
  - Control flow using registers– branch, conditional branch
- Therefore, programs must be brought (from disk) into memory for them to be run

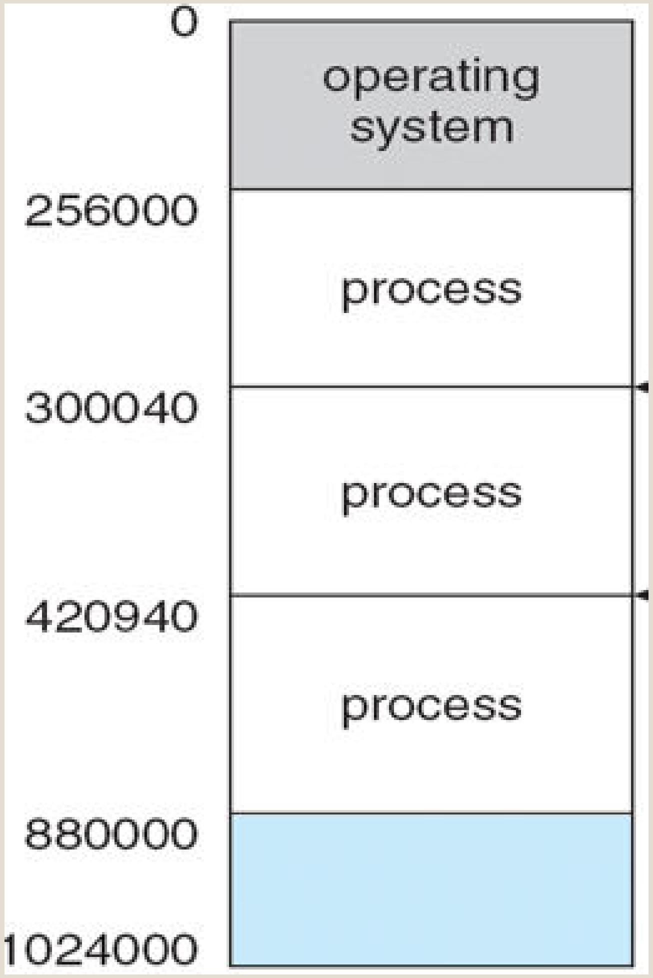
Physical Address

- The address in the particular storage cell of main memory.

- The physical address space depends on the memory size
  - If we have 8GB of RAM, the address length is 33 [log2(8GB)].

- Also called real address, or binary address

Physical Address Basic Example



Virtual and Physical address spaces

- OS can reserve addresses on the memory, and use physical address to access them
- For "regular" processes we want to choose the exact address dynamically, during run time.

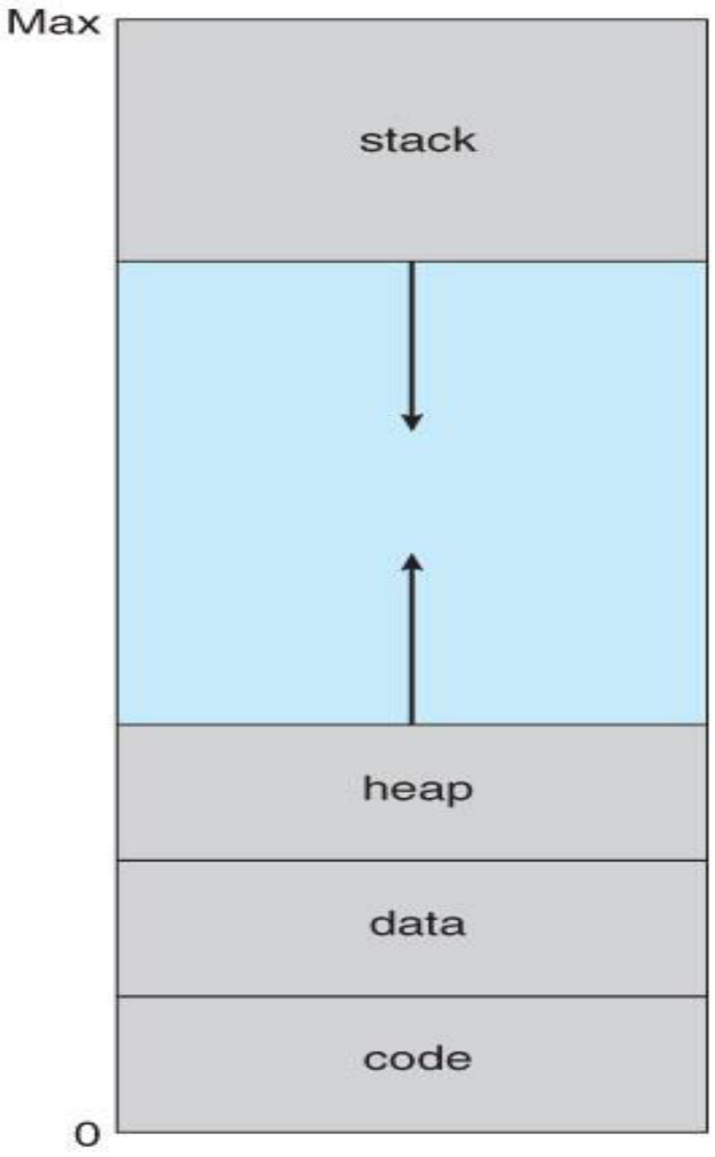
- Therefore, processes don't know their physical addresses during compile time
- Solution:
  - Programs use "virtual memory" to describe their addresses. Virtual memory starts from 0 for each process
  - Hardware (MMU) & the OS translate virtual memory to physical memory.
- New problem: how to do this translation.

Virtual Address Space (VAS)

- The set of ranges of virtual addresses that an operating system makes available to a process
- Also called logical address space.
- In 32-bit operating system, the virtual address space size is 2^32 = 4GB, in 64-bit it's 2^64 (= a lot)
- For example, a pointer to a function or variable is actually a virtual address.

Virtual Address Space

Data contains static variables and globals.



MMU

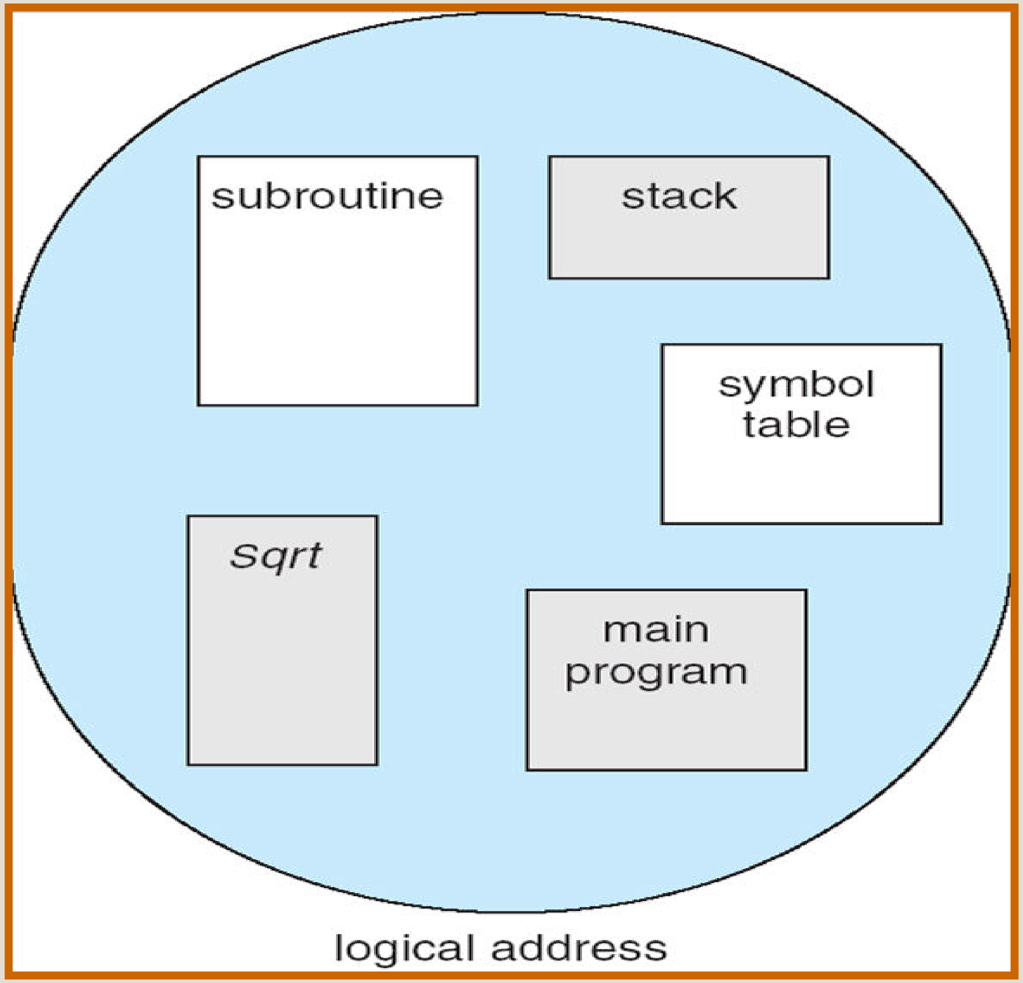
- The user program deals with *logical* addresses - it never sees the *real* physical addresses

- Memory-Management Unit (MMU) is the hardware device that maps virtual to physical address

Segmentation

- Memory segmentation is the division of a computer's primary memory into segments
- A program is a collection of segments. A segment is a logical unit such as:
  - main program
  - function
  - object
  - local variables, global variables
  - common block
  - stack
  - symbol table
  - arrays

User's View of a Program

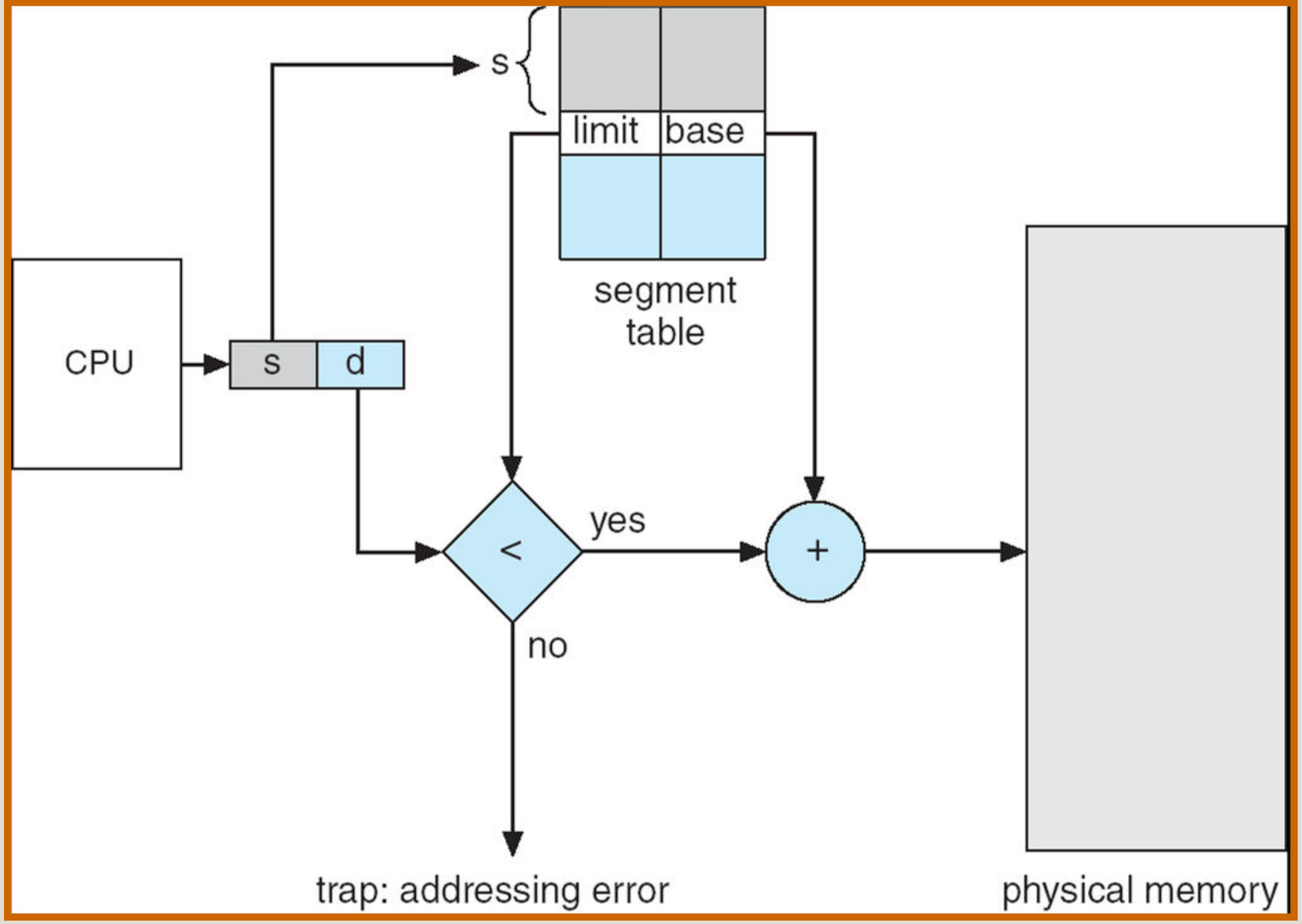


Segmentation Architecture

- A Logical address consists of a two tuple:

  <segment-number, offset>
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segment resides in memory
  - **limit** – specifies the length of the segment. Offset o is valid if o < limit
- **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

Segmentation Hardware

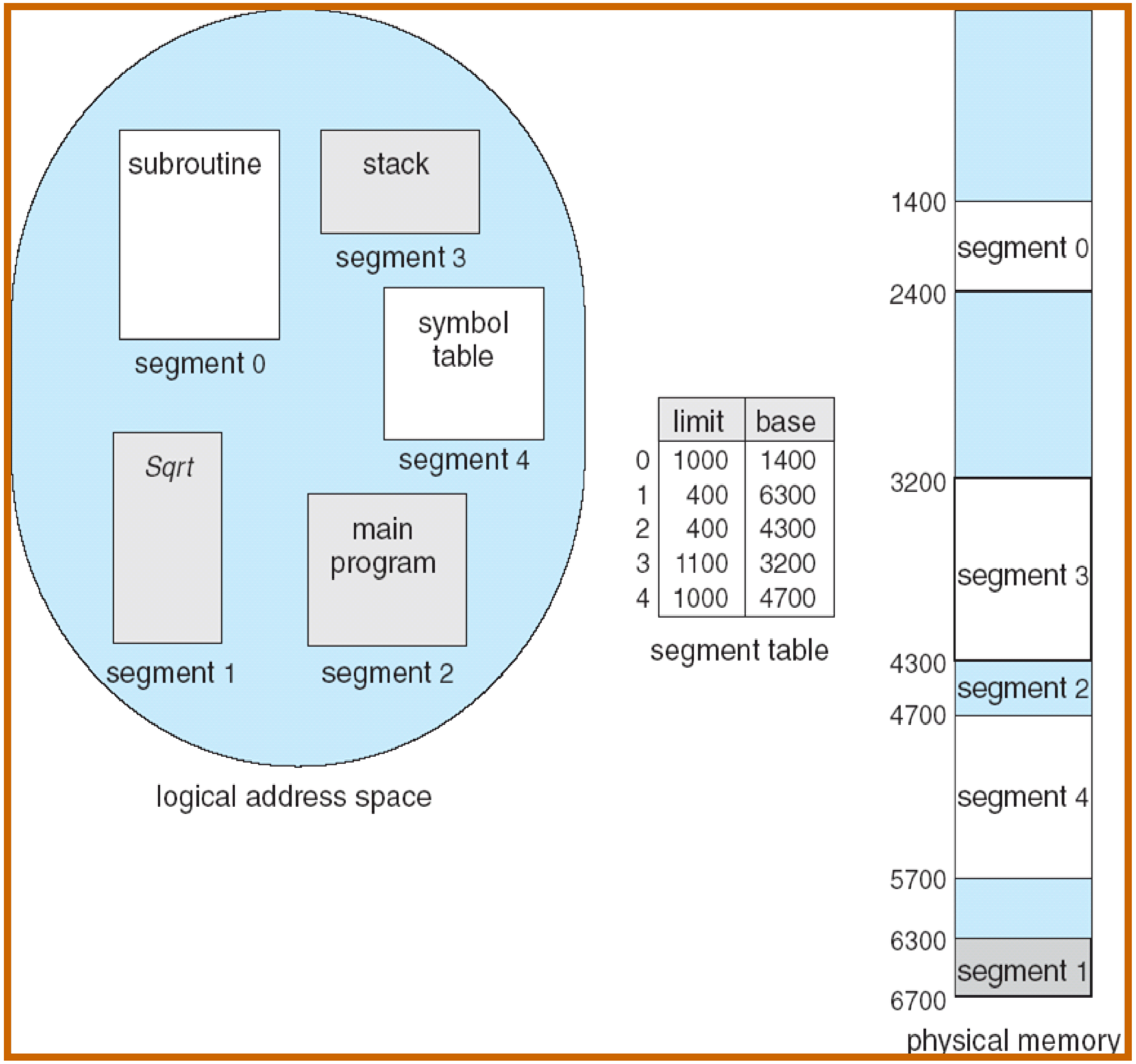


Segmentation Architecture - Cont.

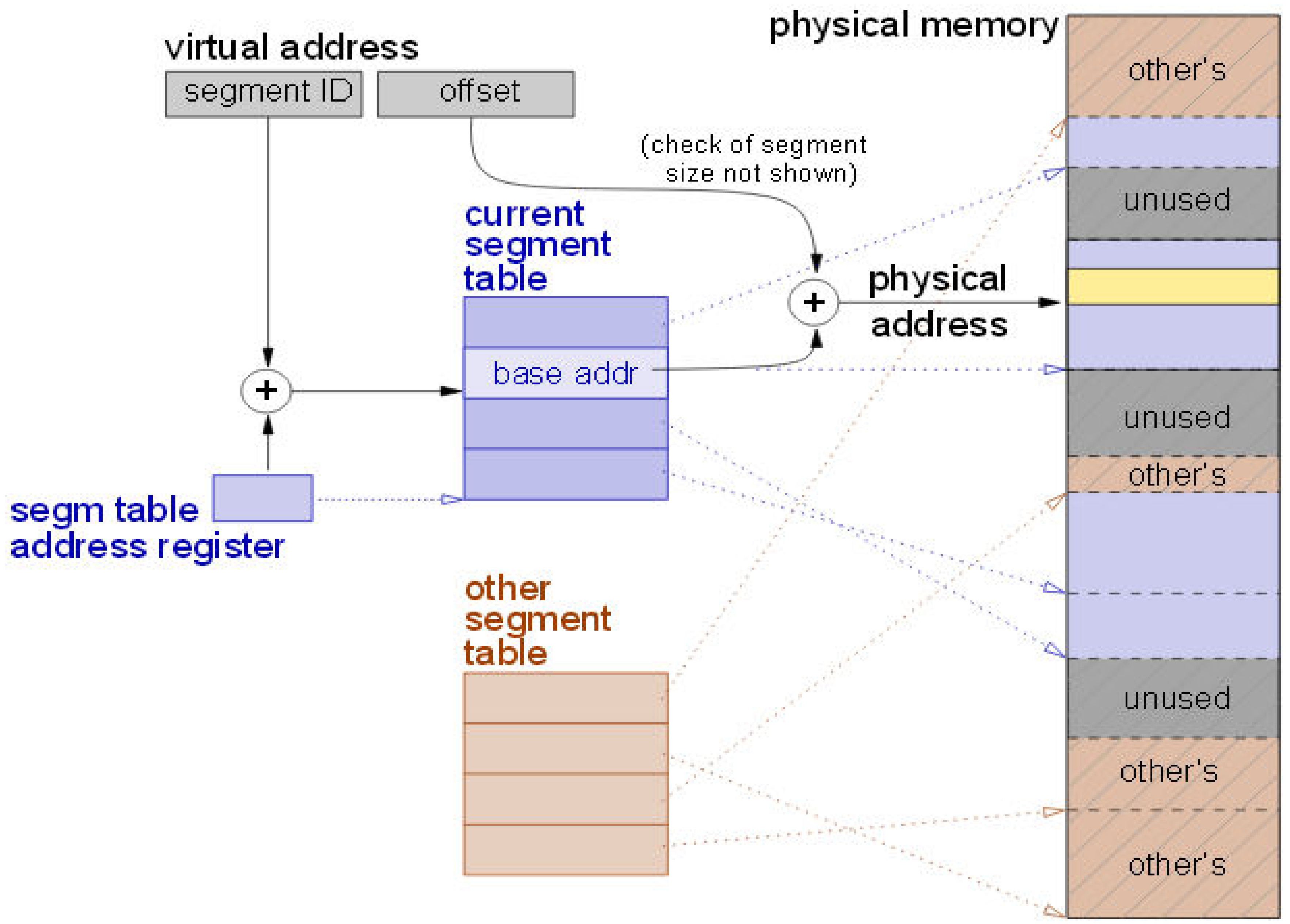
- Additional bits in each entry in the segment table:
  - validation bit (also called present-bit)
  - read/write/execute privileges

- Using the validation bit, when there is no space in the RAM, we can store segments in the disk and mark the validation bit 0.

Example of Segmentation

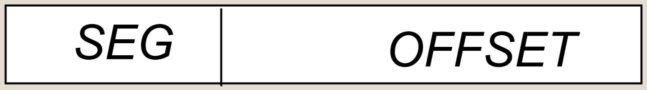


	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700


virtual address
segment ID
offset
physical memory
other's
(check of segment size not shown)
unused
current segment table
+
physical address
base addr
+
unused
other's
segm table address register
other segment table
unused
other's
other's

Segmentations: Summary

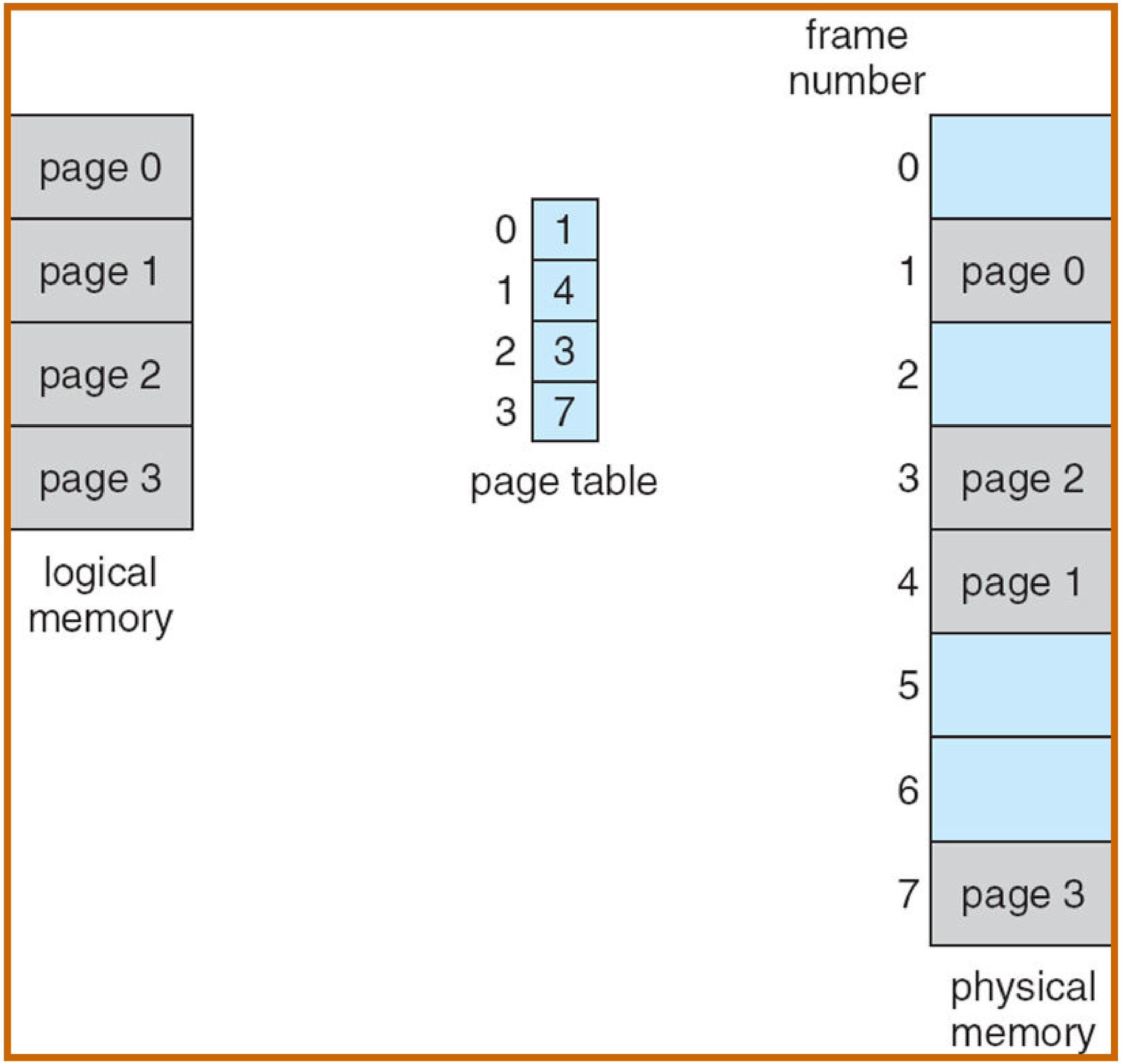
- Divide the program into segments such as:
  code, heap, stack, data-structures.
- The addresses of each segment are sequential on the physical memory
- Given a virtual address number X in segment Y, we need to:
  - Get the base address of segment Y in the physical memory (Y_PHYSICAL)
  - Check that the segment's size is bigger than X (otherwise, throw segmentation fault)
  - Access physical address Y_PHYSICAL+X.
- This is done practically by:
  - Given virtual address with n bits, the first bits are the segment number, and the rest of the bits to the offset within the segment
  - There is a table, called Segment table, that maps for each segment its base address and size
- Problem: external fragmentation



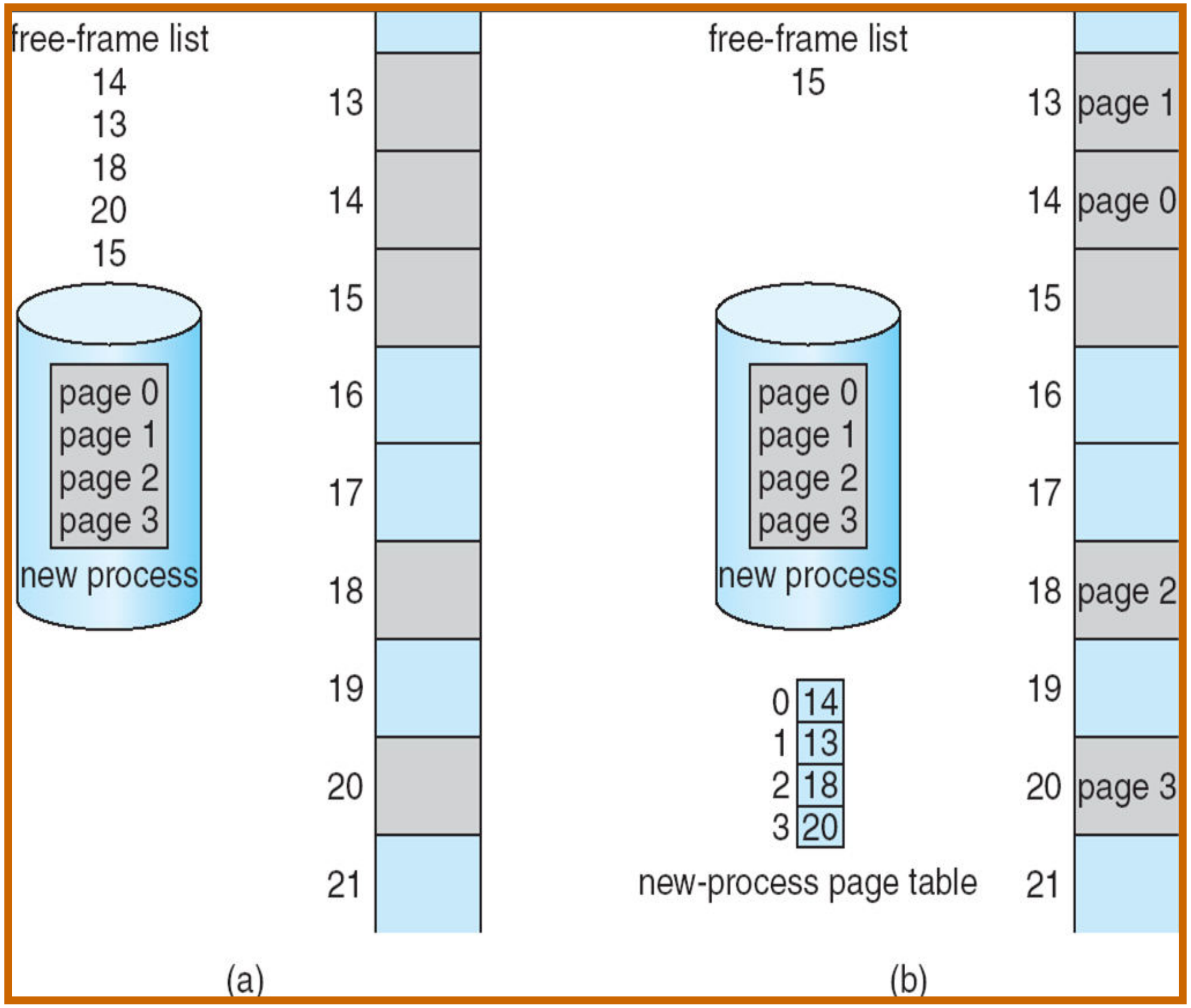
Paging

- The logical address space of process can be noncontiguous; process is allocated physical memory whenever the latter is available.
- Physical memory is partitioned into fixed-sized blocks called **frames**.
- logical memory is partitioned into blocks of the same size called **pages**.
- Keep track of all free frames.
- To run a program of size n pages, need to find n free frames and load program.
- Set up a page table to translate logical to physical addresses.
- No external fragmentation but there is bounded internal fragmentation.

Paging Model of Logical and Physical Memory

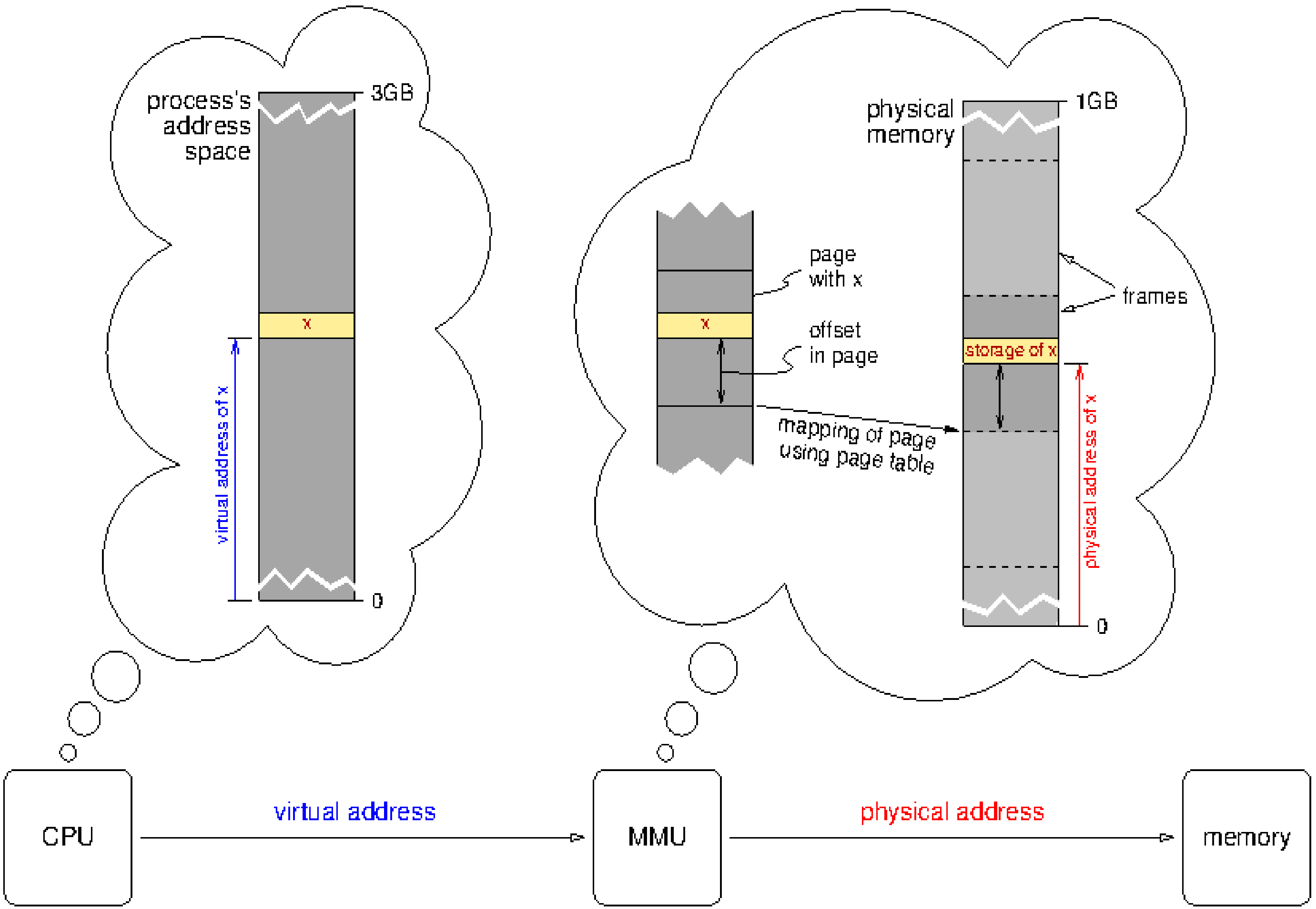


Frames Allocation Example



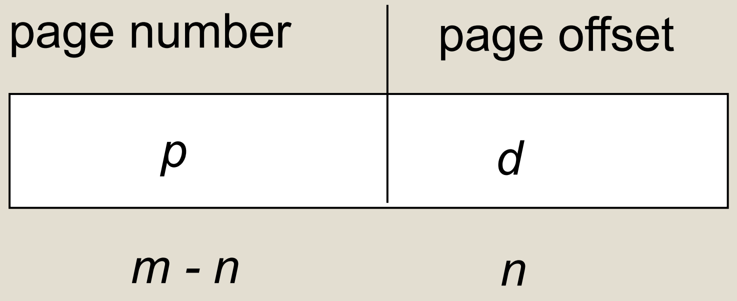
Before allocation

After allocation


process's address space
3GB
x
virtual address of x
0
page with x
x
offset in page
mapping of page using page table
physical memory
1GB
frames
storage of x
physical address of x
0
CPU
virtual address
MMU
physical address
memory

Address Translation Scheme

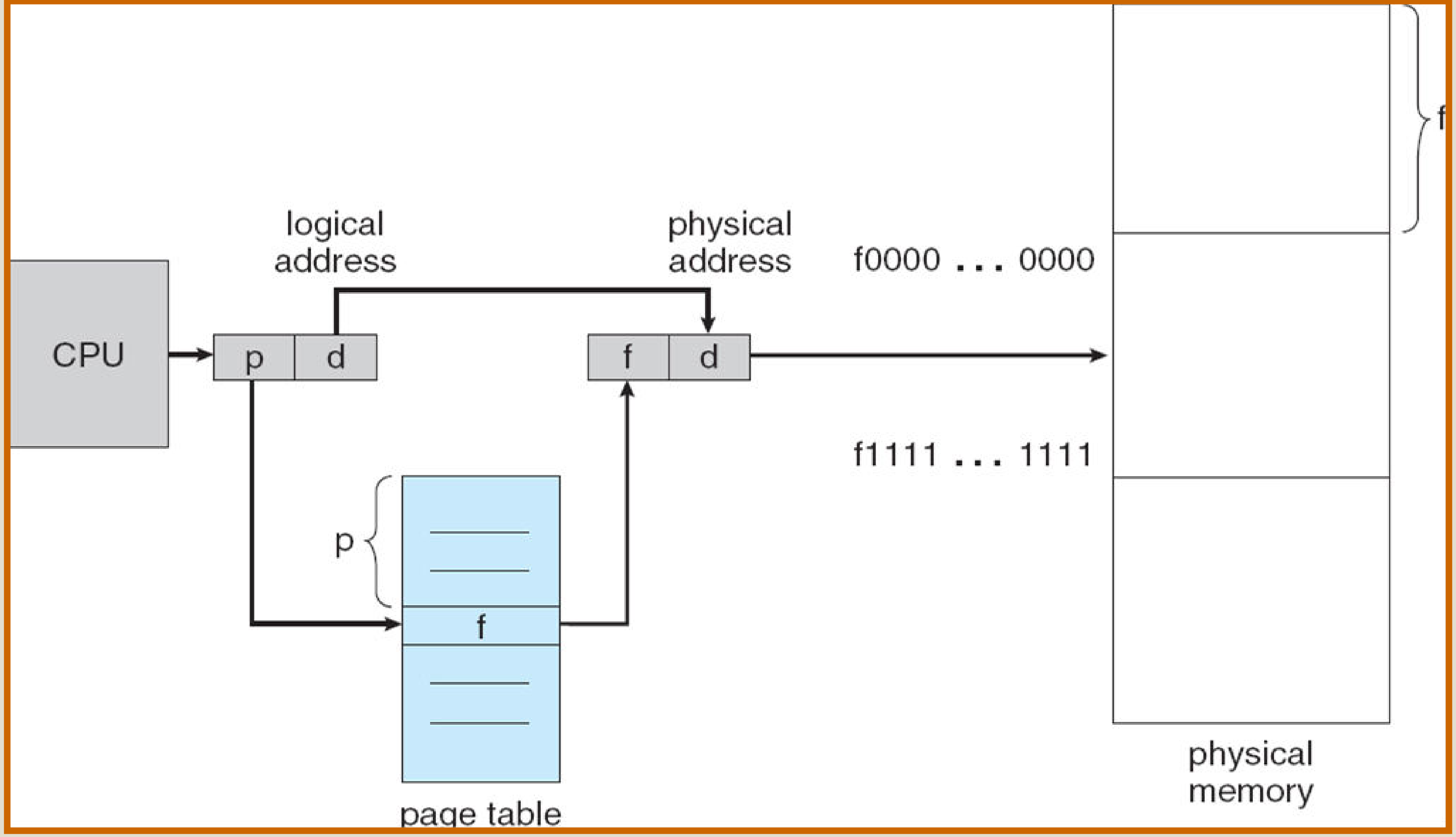
- Address generated by CPU is divided into:
- **Page number (p)** – used as an index into a ***page table*** which contains base address of each page in physical memory
- **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit
- For a given logical address space of size 2^m words and **page size** 2^n words:



Implementation of Page Table

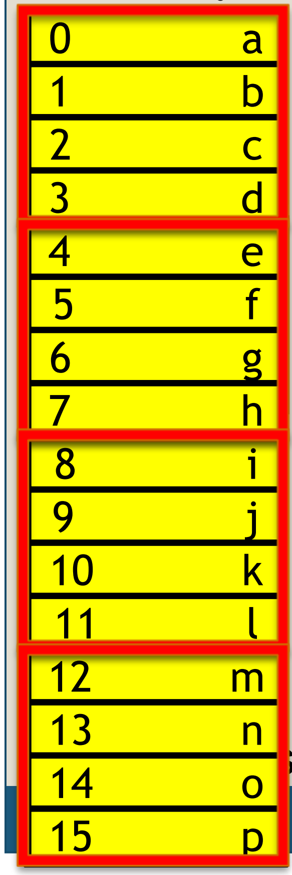
- Page table is kept in main memory
- **Page-table base register** (**PTBR**) points to the page table
- **Page-table length register** (**PRLR**) indicates size of the page table

Paging Hardware



Working with the page table

Logical memory

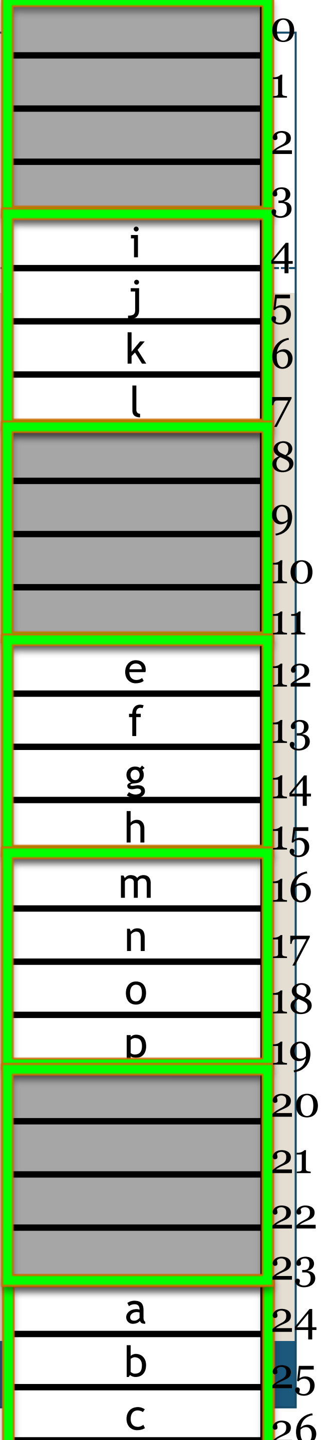


Page	Frame
0	6
1	3
2	1
3	4

(per process)

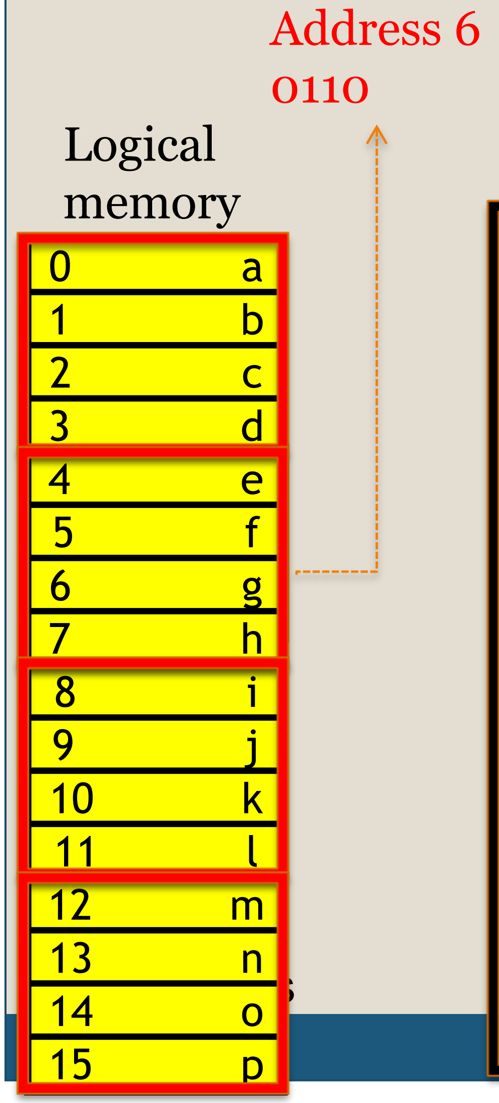
Used by the MMU

Physical memory



Slide from lecture

Working with the page table

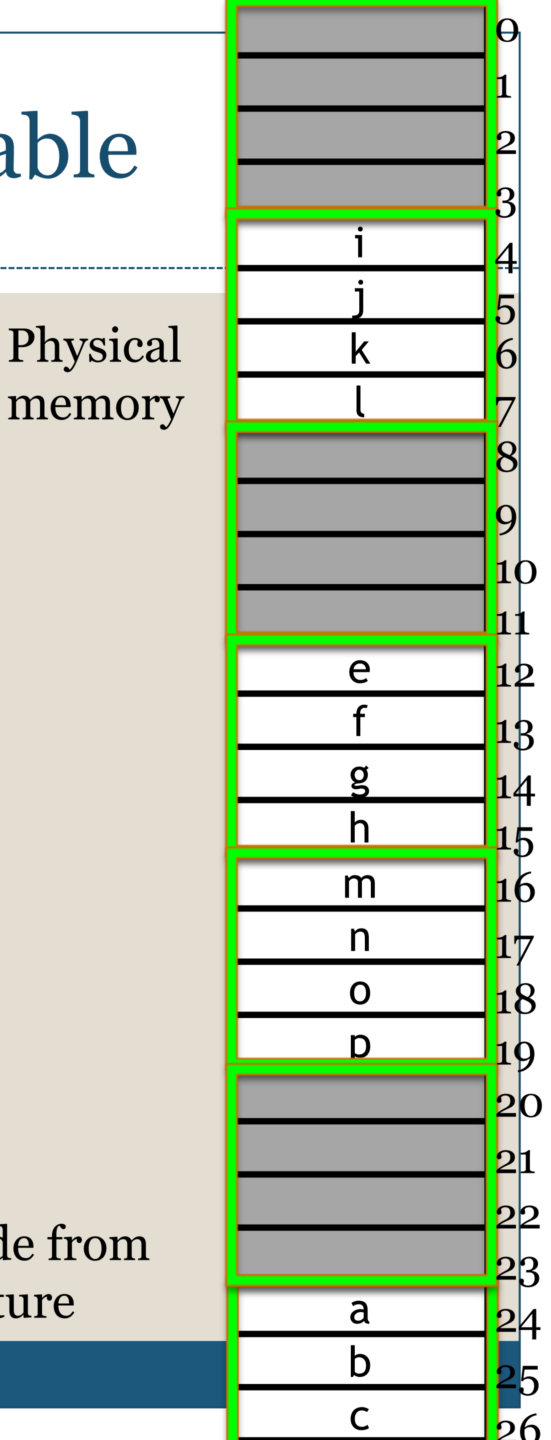


Page	Frame
0	6
1	3
2	1
3	4

(per process)

Used by the MMU

Slide from lecture

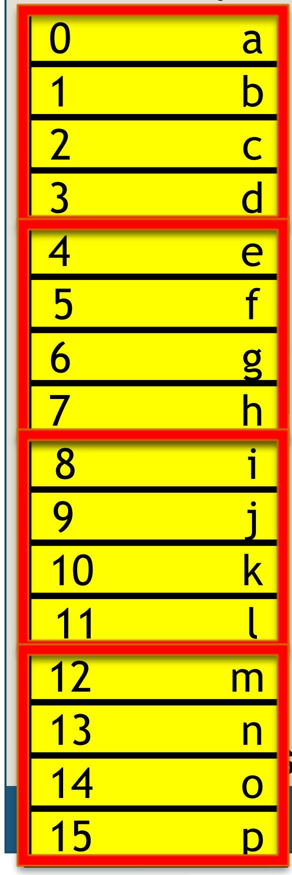


Working with the page table

Address 6
0110

Page 01
Offset 10

Logical memory

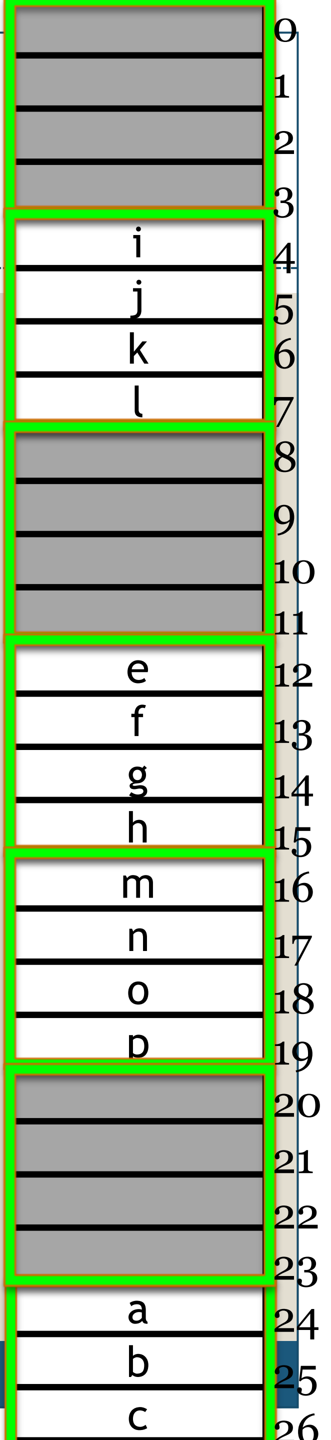


Page	Frame
0	6
1	3
2	1
3	4

(per process)

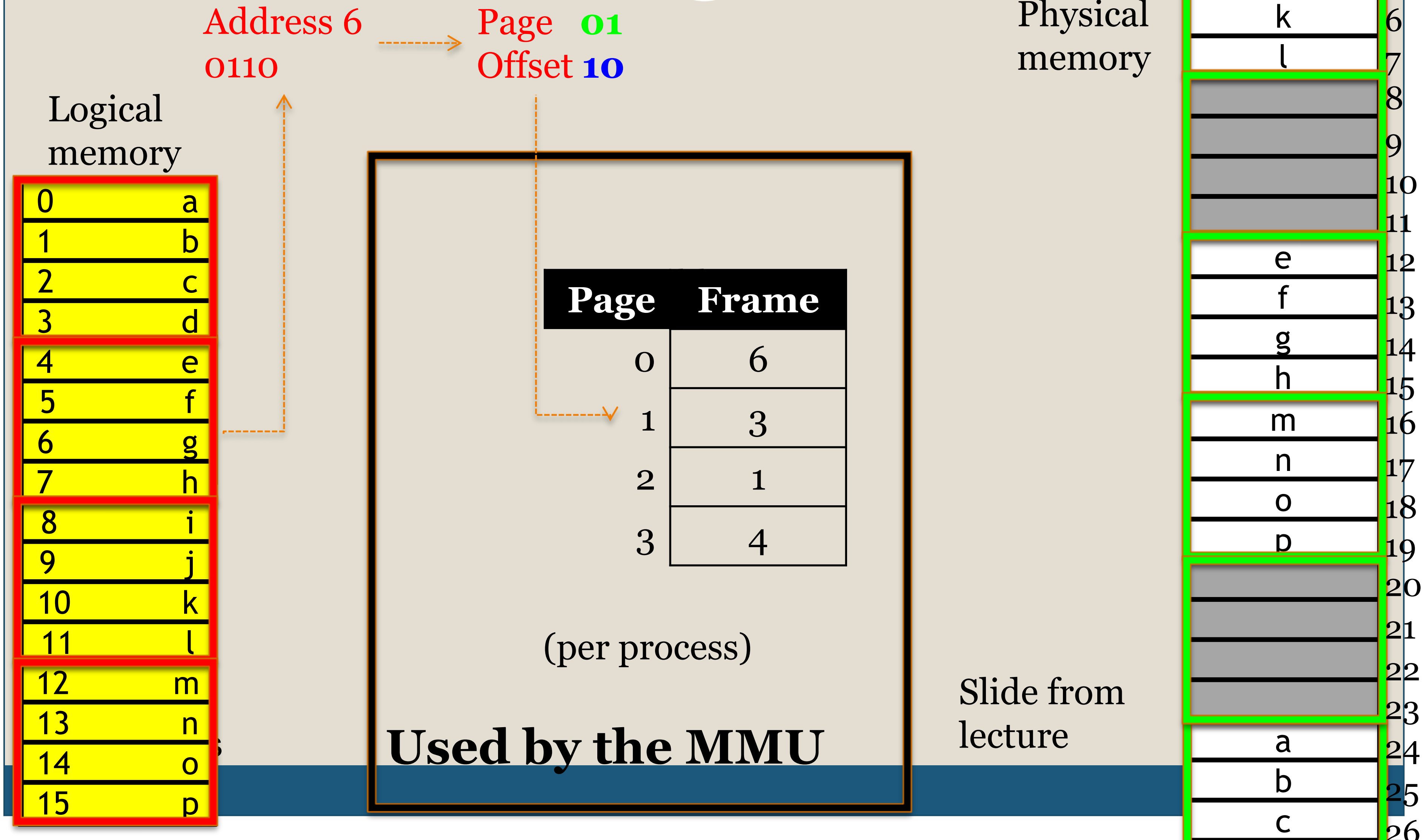
Used by the MMU

Physical memory



Slide from lecture

Working with the page table



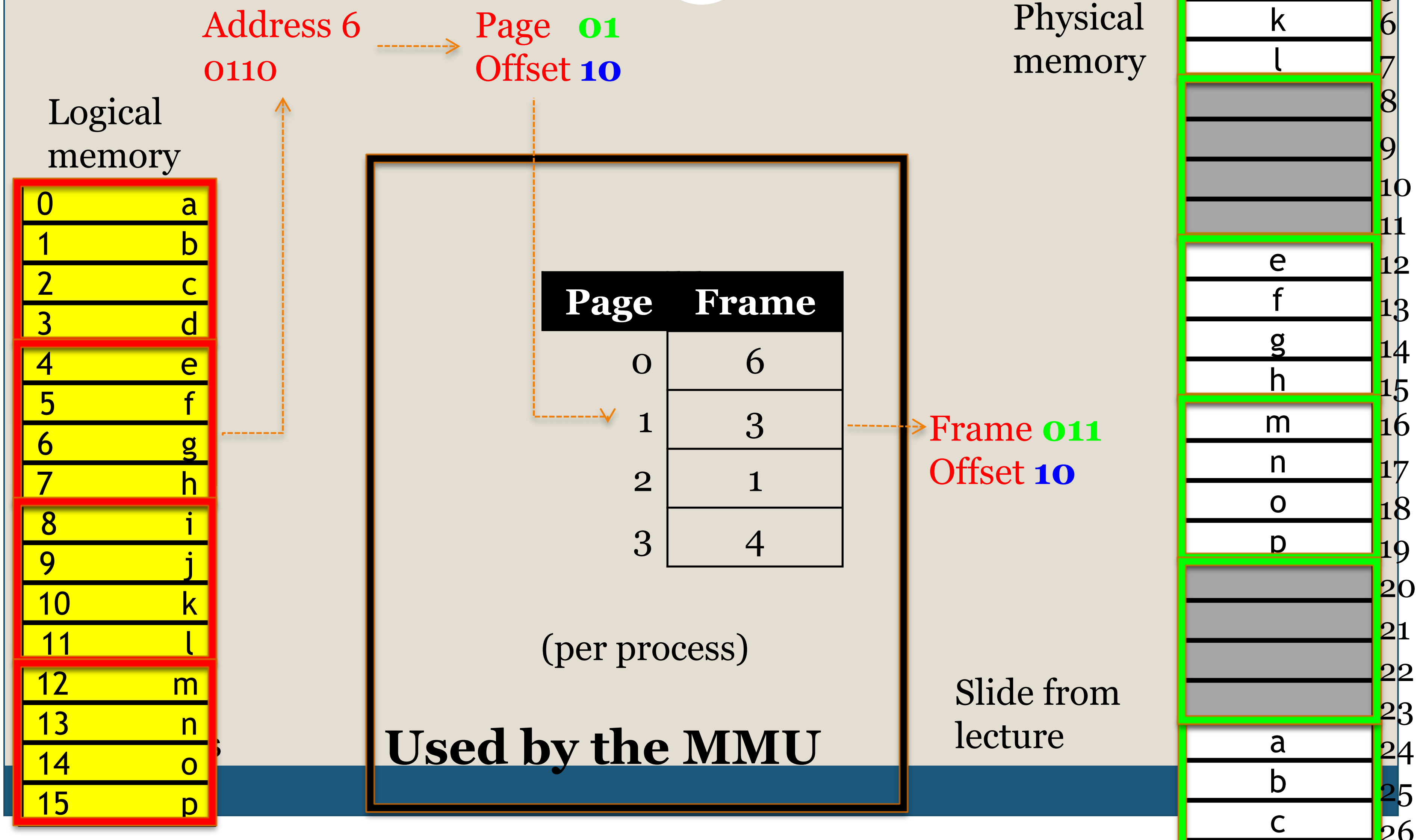
Logical address	Contents
0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

Page	Frame
0	6
1	3
2	1
3	4

Contents	Physical address
▨	0
▨	1
▨	2
▨	3
i	4
j	5
k	6
l	7
▨	8
▨	9
▨	10
▨	11
e	12
f	13
g	14
h	15
m	16
n	17
o	18
p	19
▨	20
▨	21
▨	22
▨	23
a	24
b	25
c	26

Slide from lecture

Working with the page table



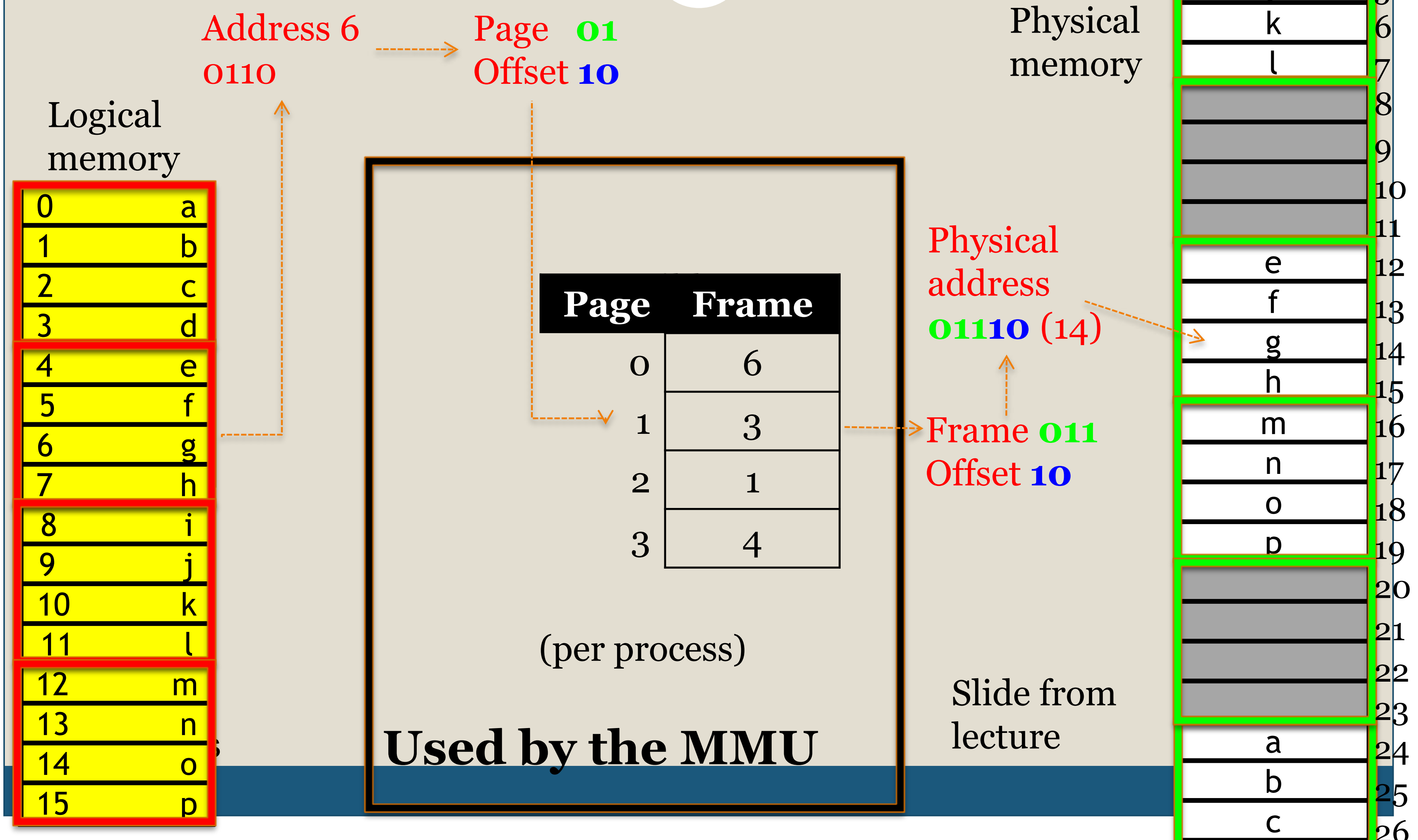
Page	Frame
0	6
1	3
2	1
3	4

(per process)

Used by the MMU

Slide from lecture

Working with the page table



Page	Frame
0	6
1	3
2	1
3	4

(per process)

Used by the MMU

Slide from lecture

Page Table Additional Info

- ***Valid bit*** - indicates whether the page is assigned to a frame.
- **Modified** - **Dirty bit** - indicates whether the page was modified.
- **Used bit** - when was the last accesses to page, i.e. timestamp.
  - E.g. for the clock algorithm for page replacement
- **Access permissions** - read-only, read-write

Page Replacement

- If the valid bit is off, the page is not mapped to a frame
- This caused an exception named page fault
- The OS searches for an available frame
  - If there is one – the page is loaded from the disk to this frame
  - Otherwise, the OS pages out (swaps out) a page to the disk, brings the requested frame to this location in the memory, and updates the page table accordingly.
- To choose the evicted page, the OS uses Page Replacement Algorithms, such as the clock algorithm (using used bit)

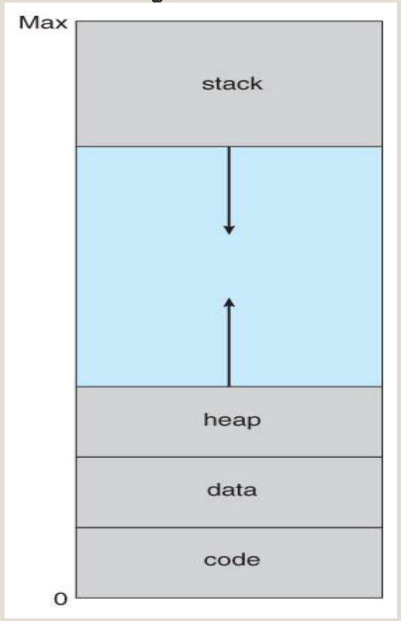
Structure of the Page Table

- As the number of processes increases, the percentage of memory devoted to page tables also increases.

- The following structures solved this problem:
  - Hierarchical Paging
  - Inverted Page Tables

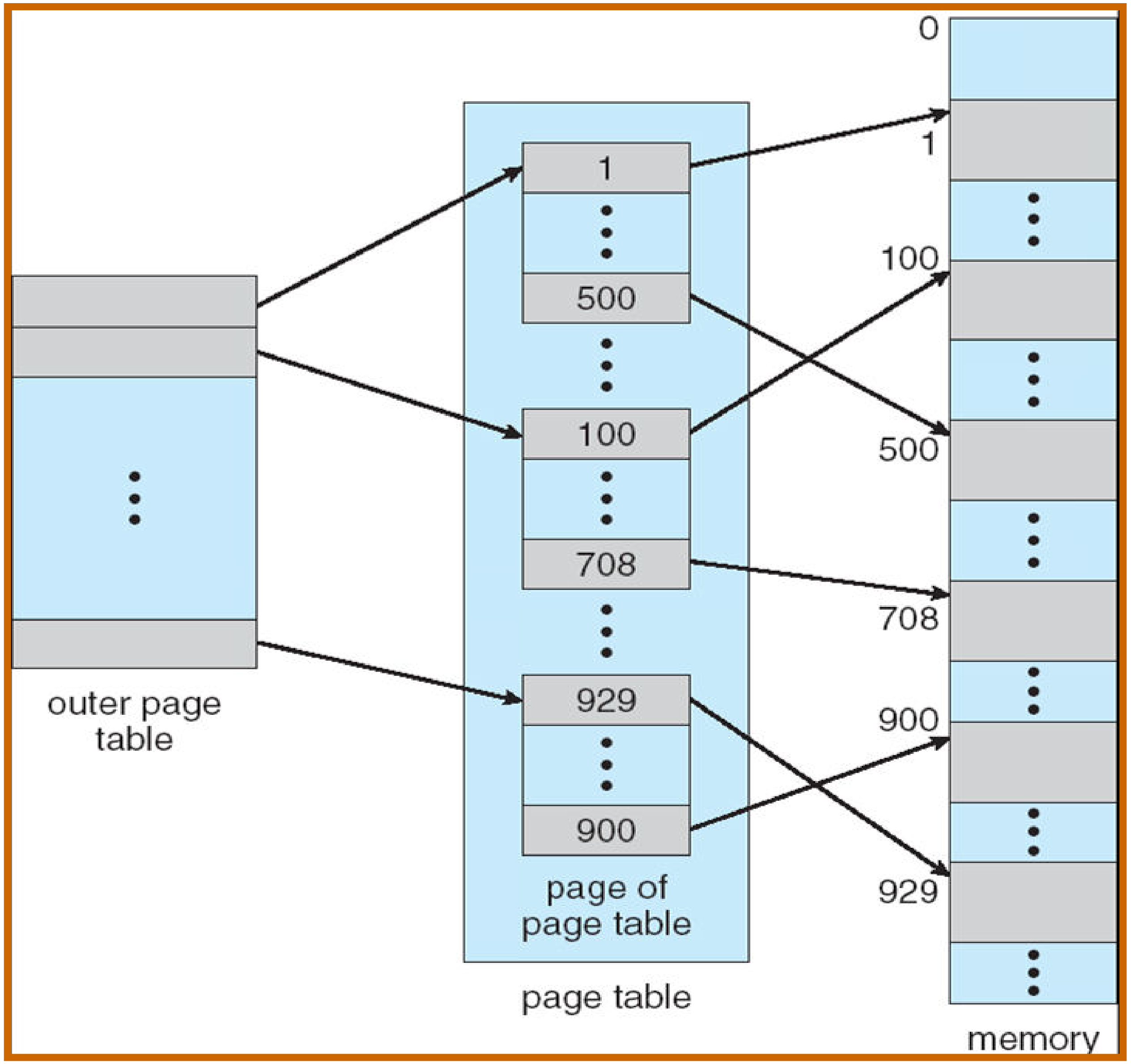
Hierarchical Page Tables

- Also called multilevel page table
- Break up the logical address space into multiple page tables
  - We don't need to preserve all the page tables, but only those that we use.
  - This works well because many times programs don't use the middle addresses ->



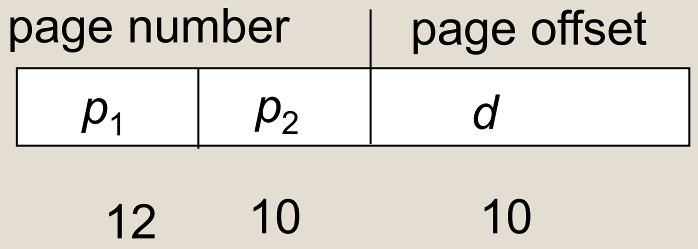
- A simple technique is a two-level page table (the page table is paged).

Two-Level Page-Table Scheme



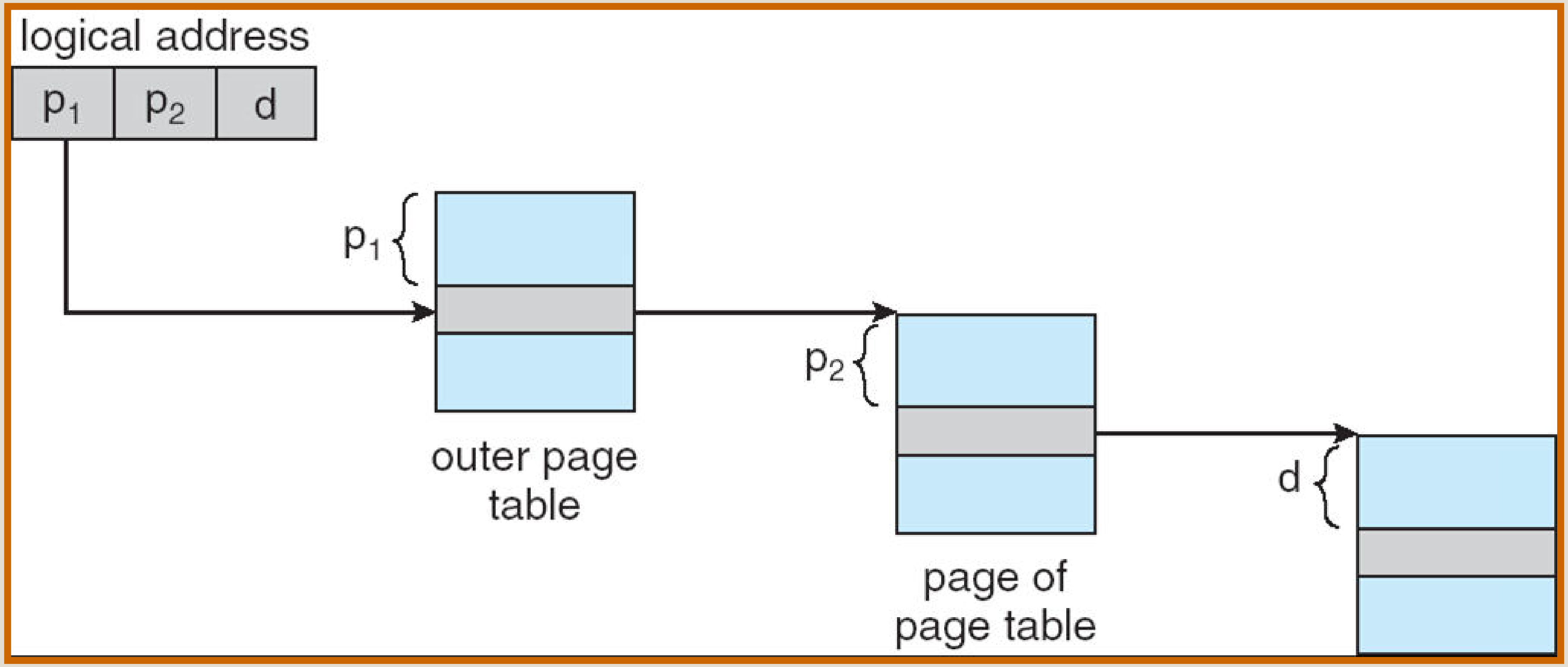
Two-Level Paging Example

- On a 32-bit machine, with 1K page size
- A logical is divided into
  - a page number consisting of 22 bits (p1 + p2)
  - a page offset consisting of 10 bits (d)
- Since the page table is paged, the page number is divided into:
  - a 12-bit page number (p1)
  - a 10-bit page offset (p2)
- Thus, a logical address is as follows:



- where p1 is an index into the outer page table, and p2 is the displacement within the page of the inner page table.

Address-Translation Scheme



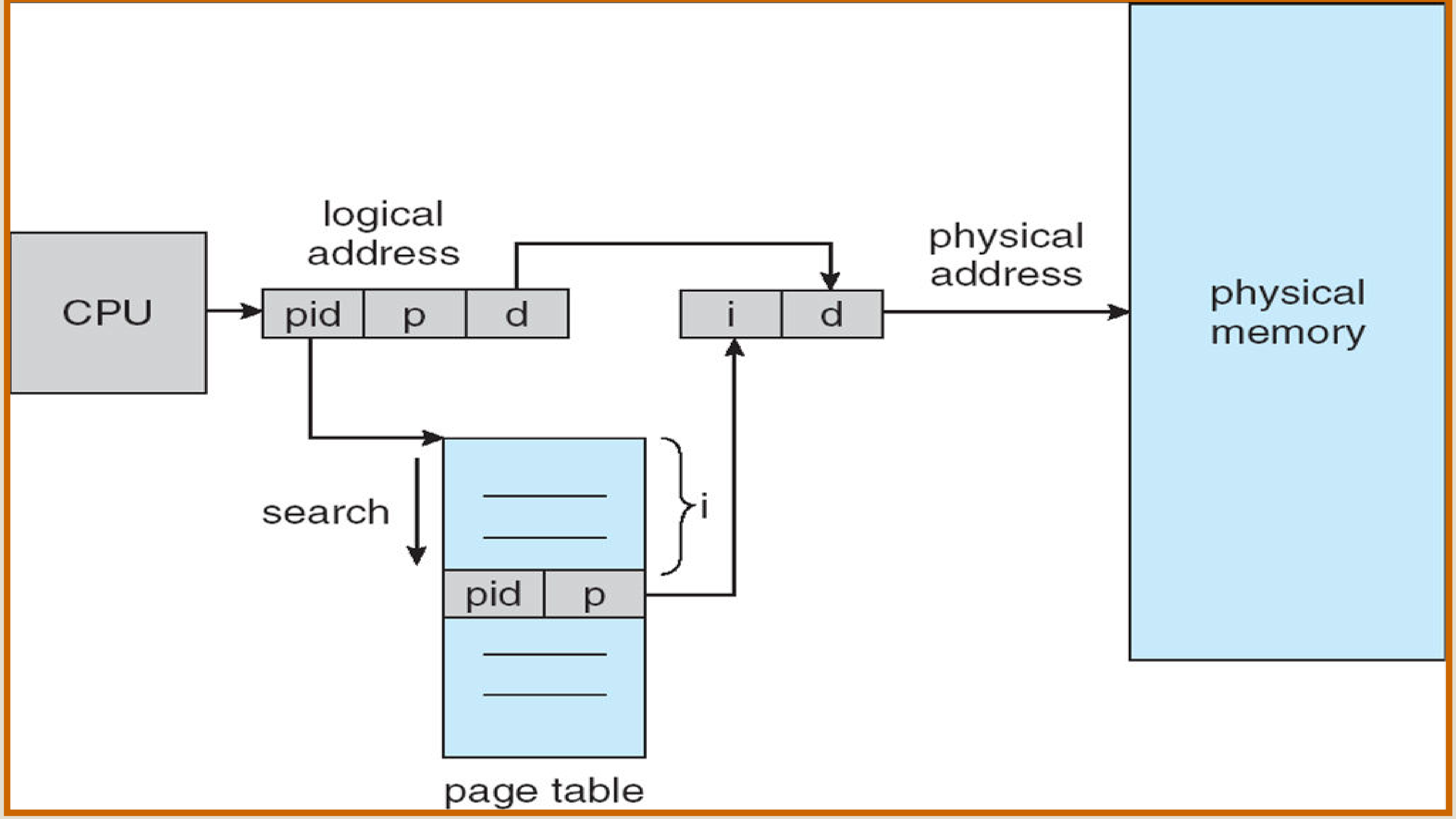
Inverted Page Table

- There is one table with the size of the physical memory <u>shared for all processes.</u>

- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.

- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.

Inverted Page Table Architecture



Page Table - Limitations

- Mapping virtual addresses to physical memory adds overhead to every memory access.
- Even in 1-level paging, every data/ instruction access requires two memory accesses: one for the page table and one for the data/instruction.
  - Usually it's more than two accesses (multi-level paging).

⬇

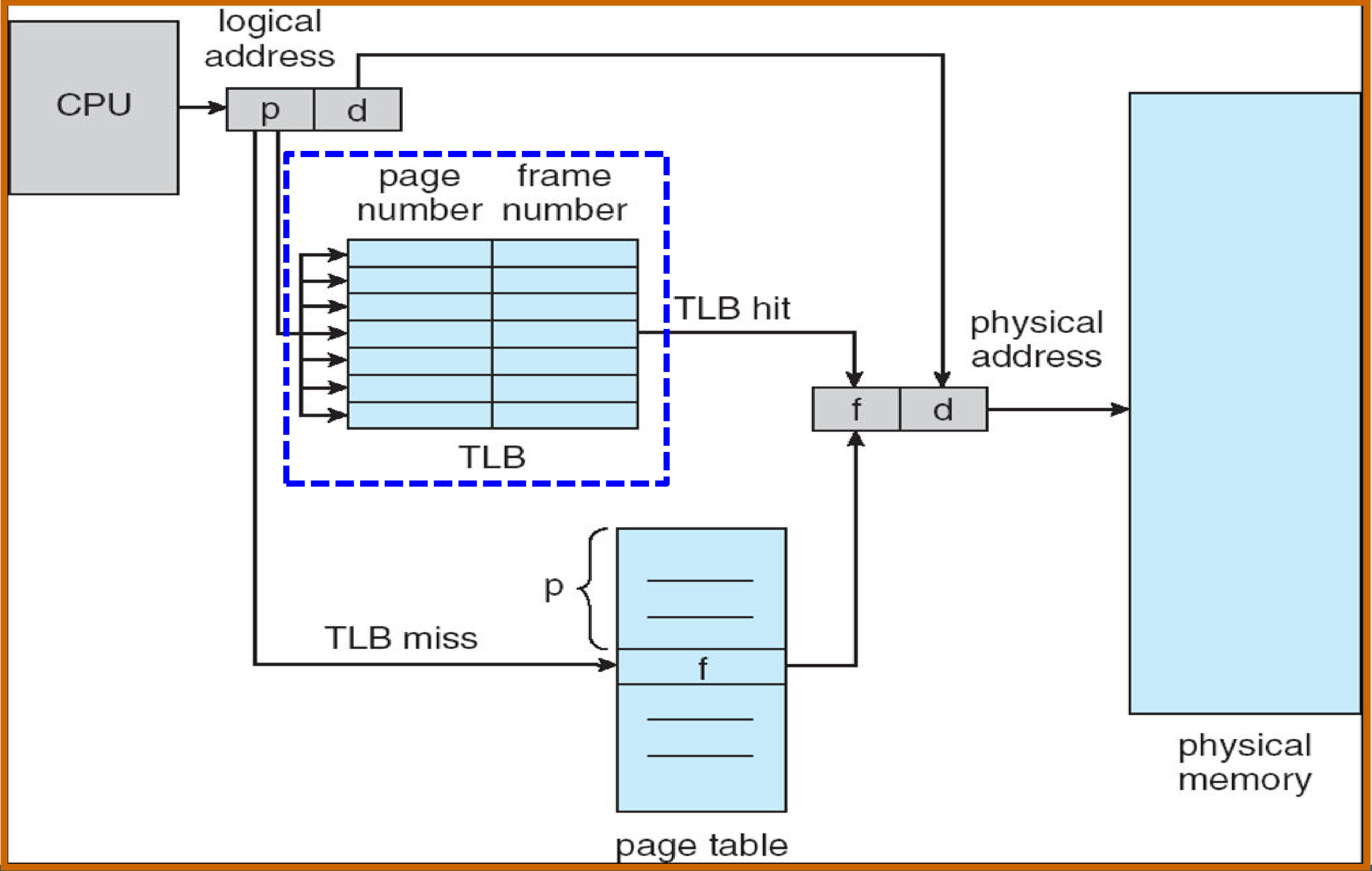
- The CPU uses a cache of recently used mappings from the operating system's page table.
- This cache is called the **Translation look-aside buffers (TLBs)** which is an associative cache.

TLB

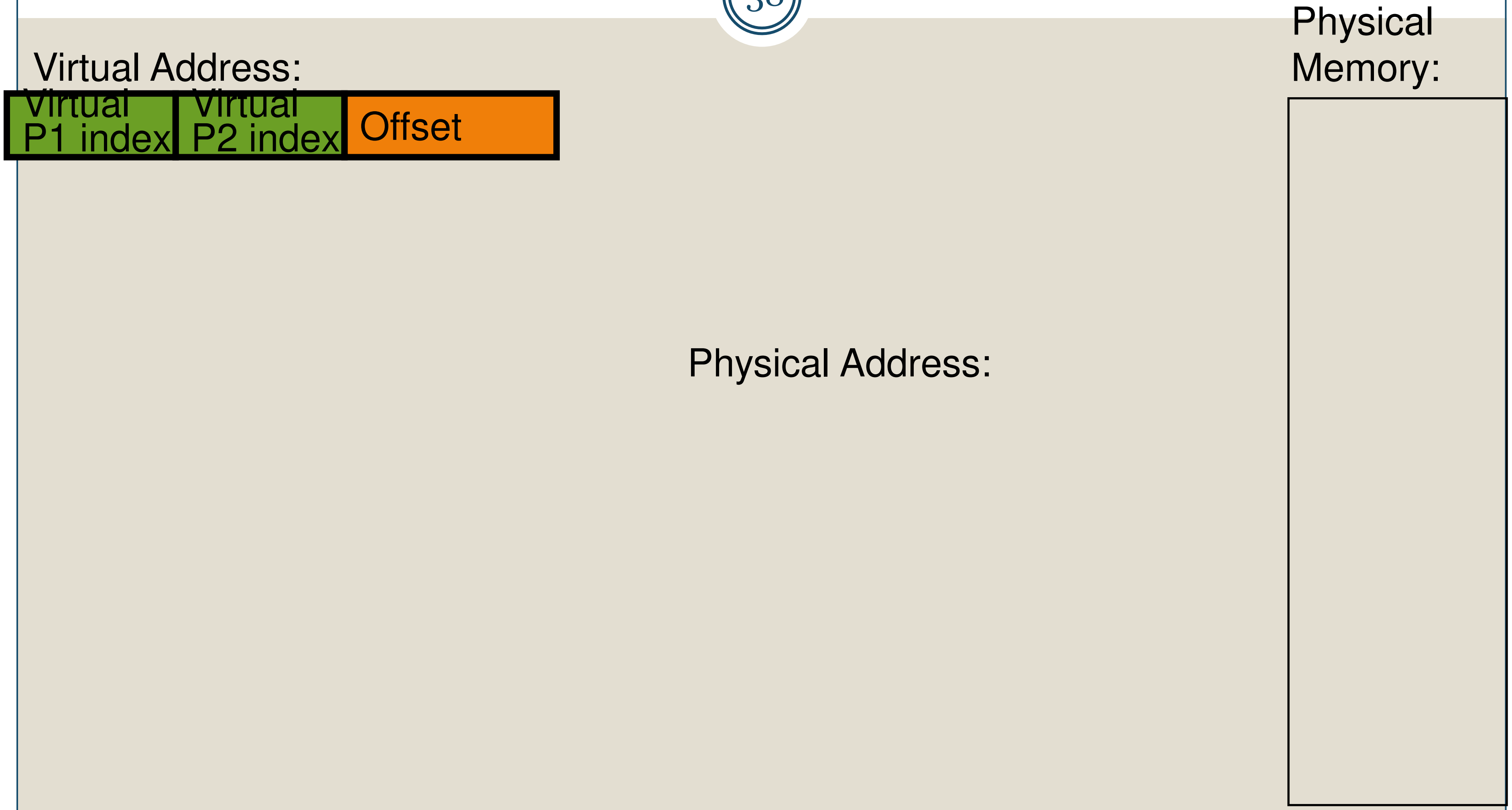
- Usually it's fully associative with 64 entries

- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process.

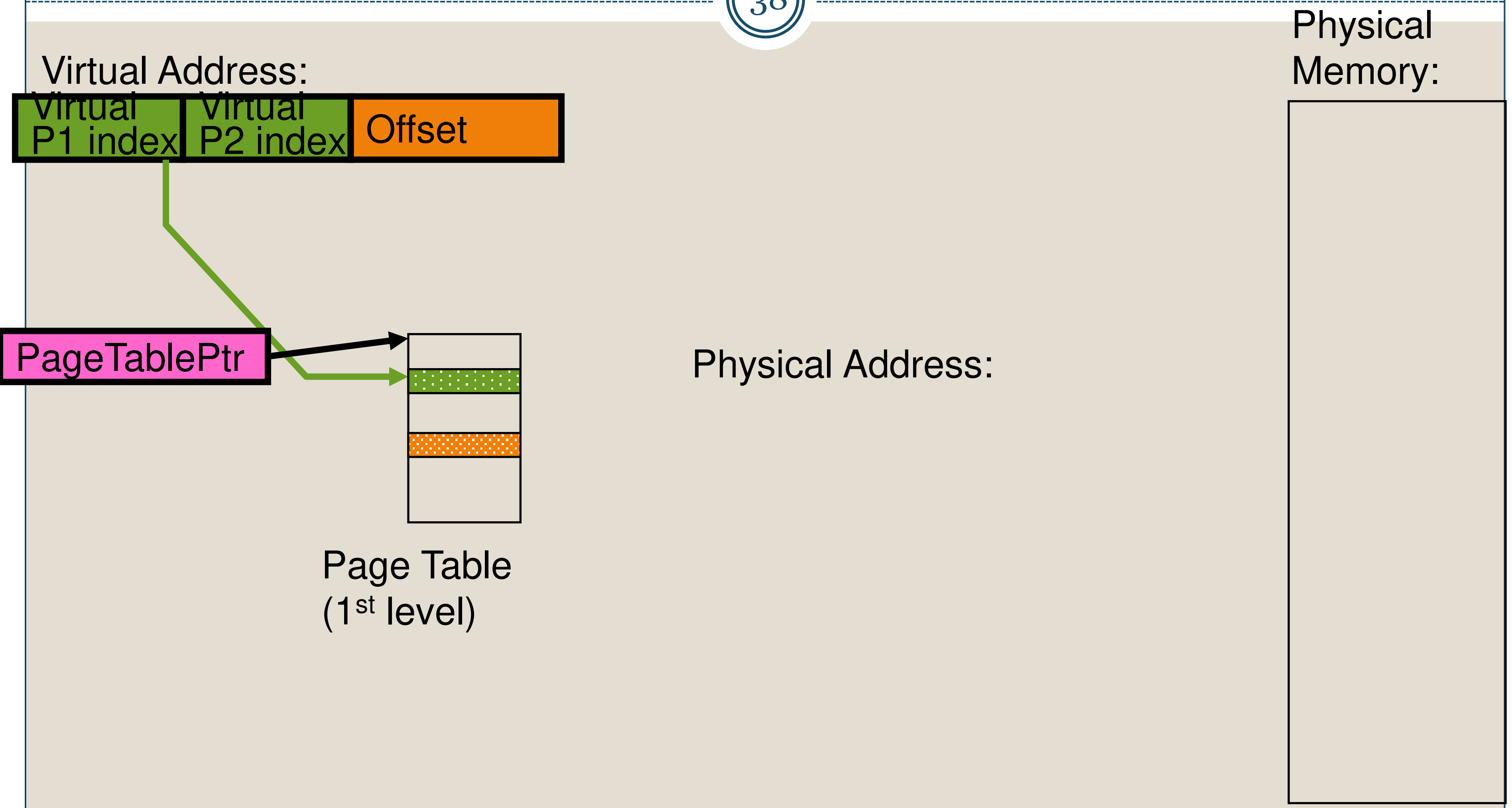
Paging Hardware With TLB



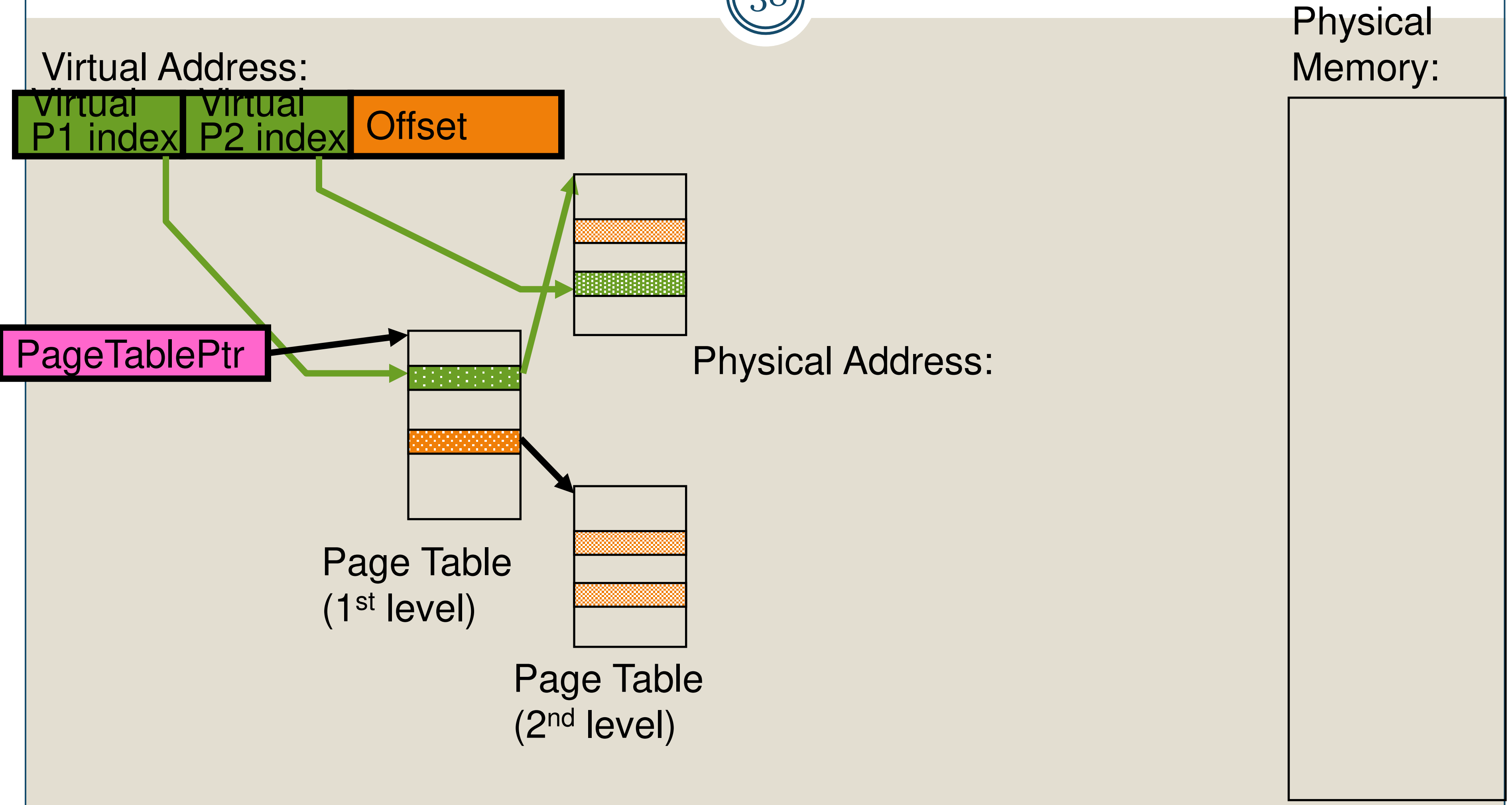
Putting Everything Together: Address Translation



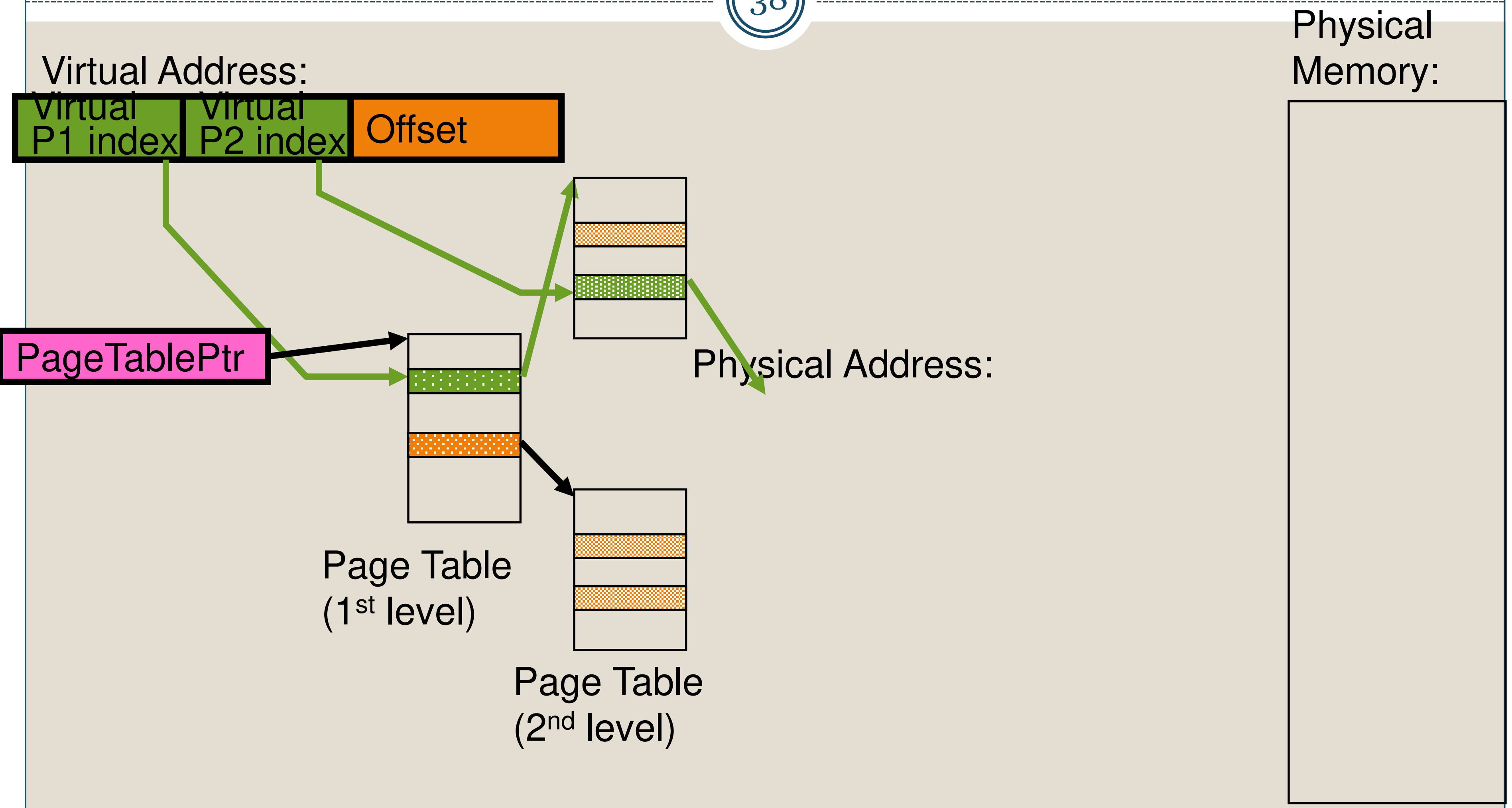
Putting Everything Together: Address Translation



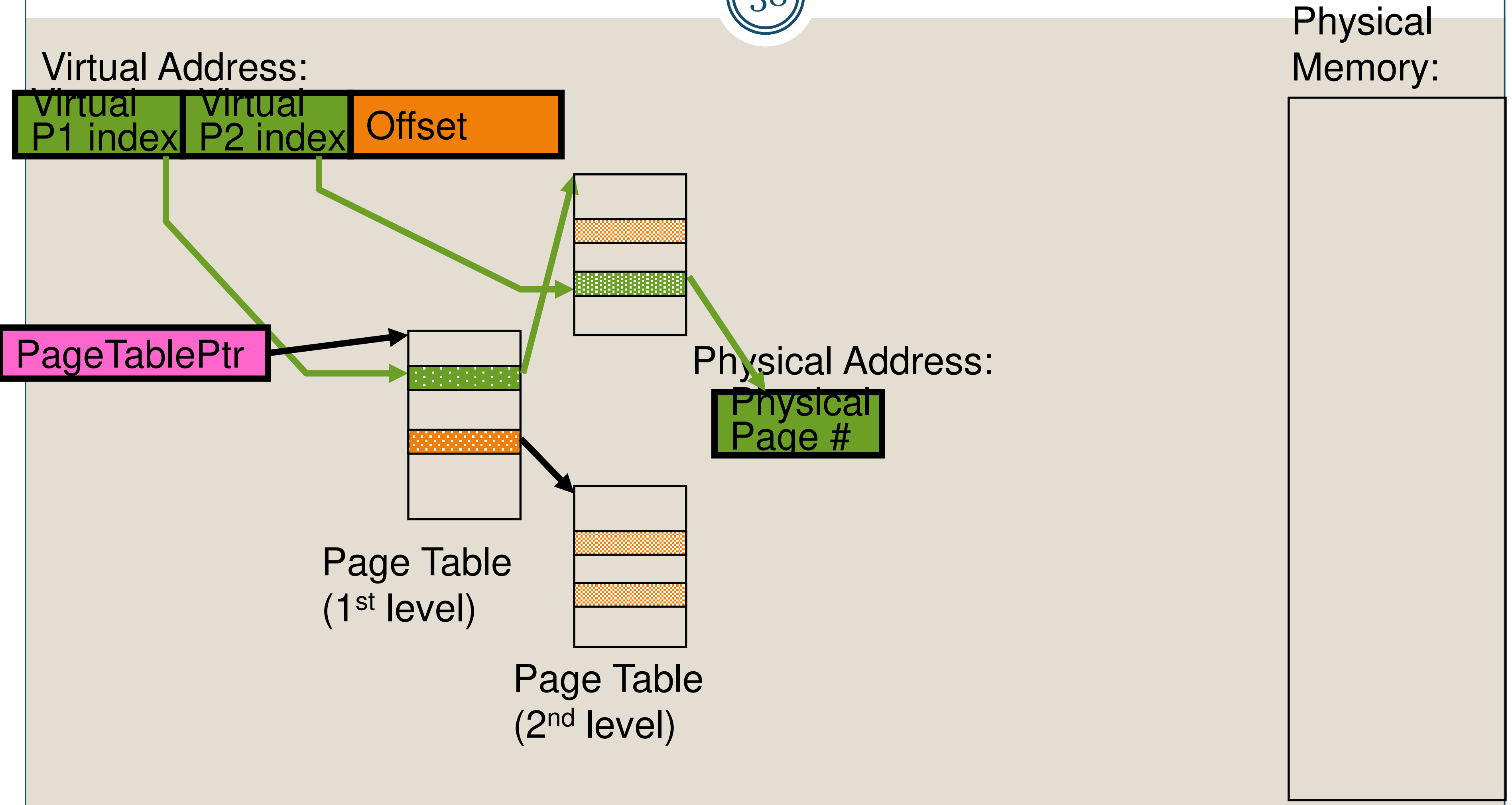
Putting Everything Together: Address Translation



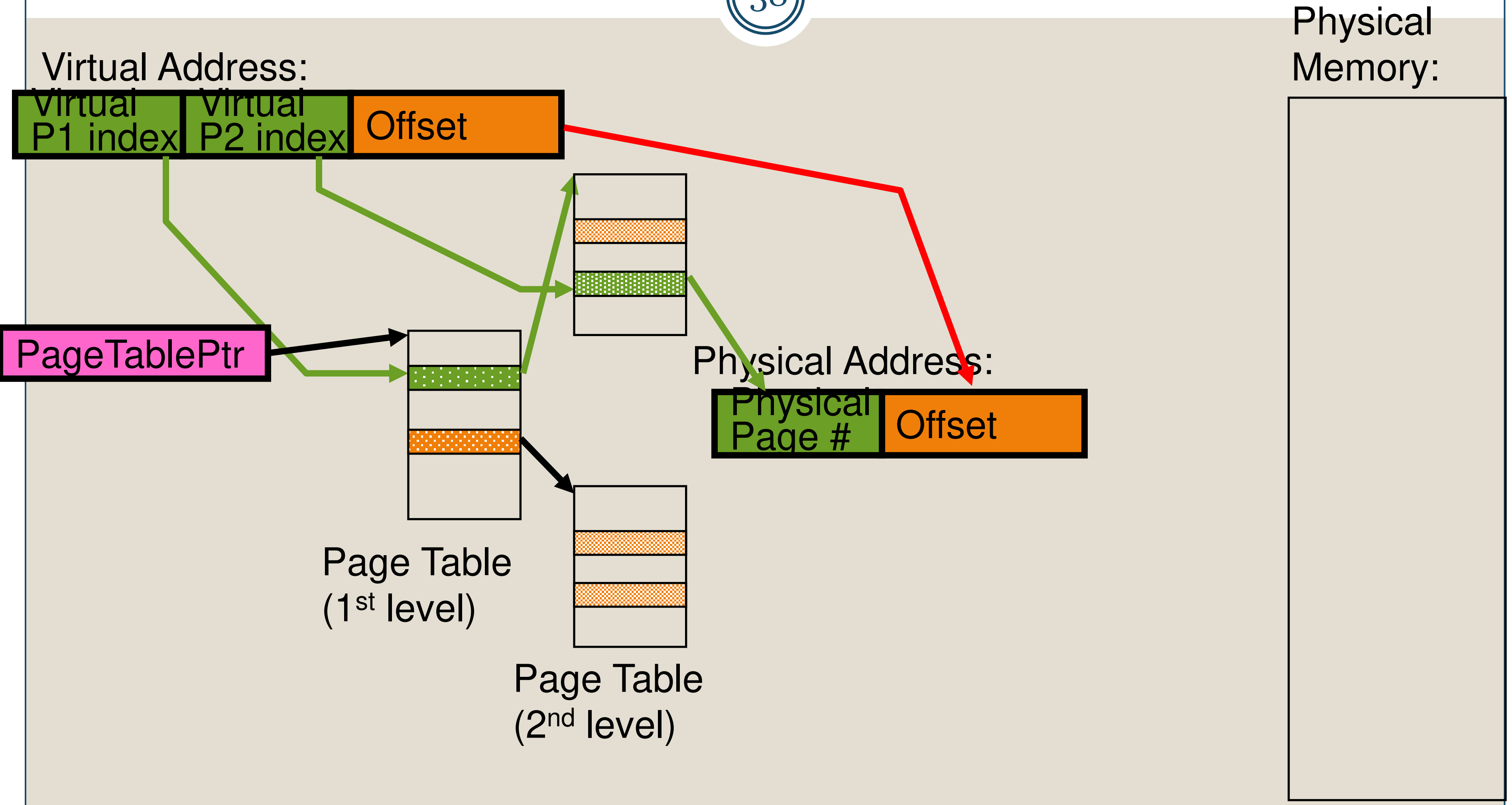
Putting Everything Together: Address Translation



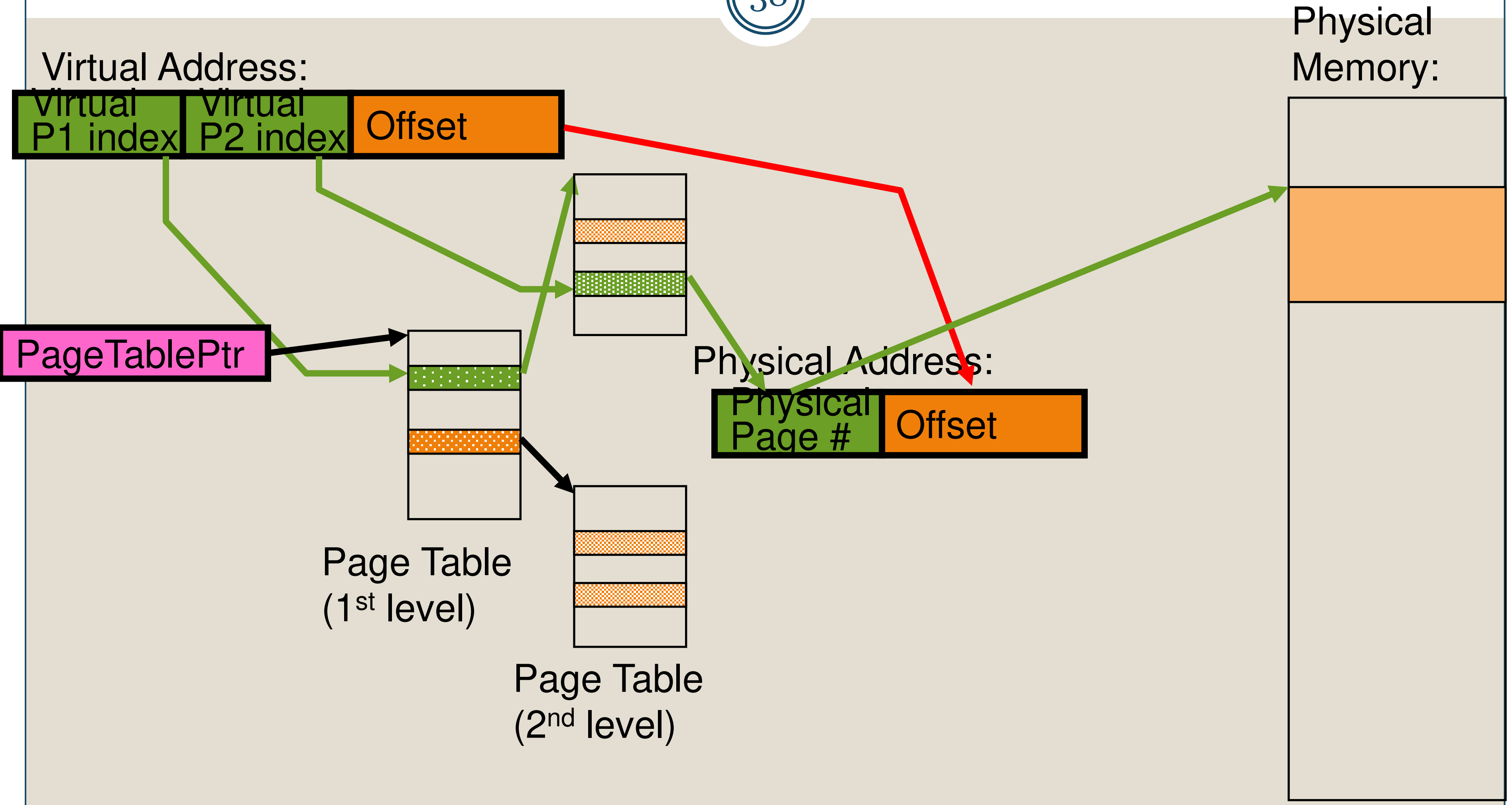
Putting Everything Together: Address Translation



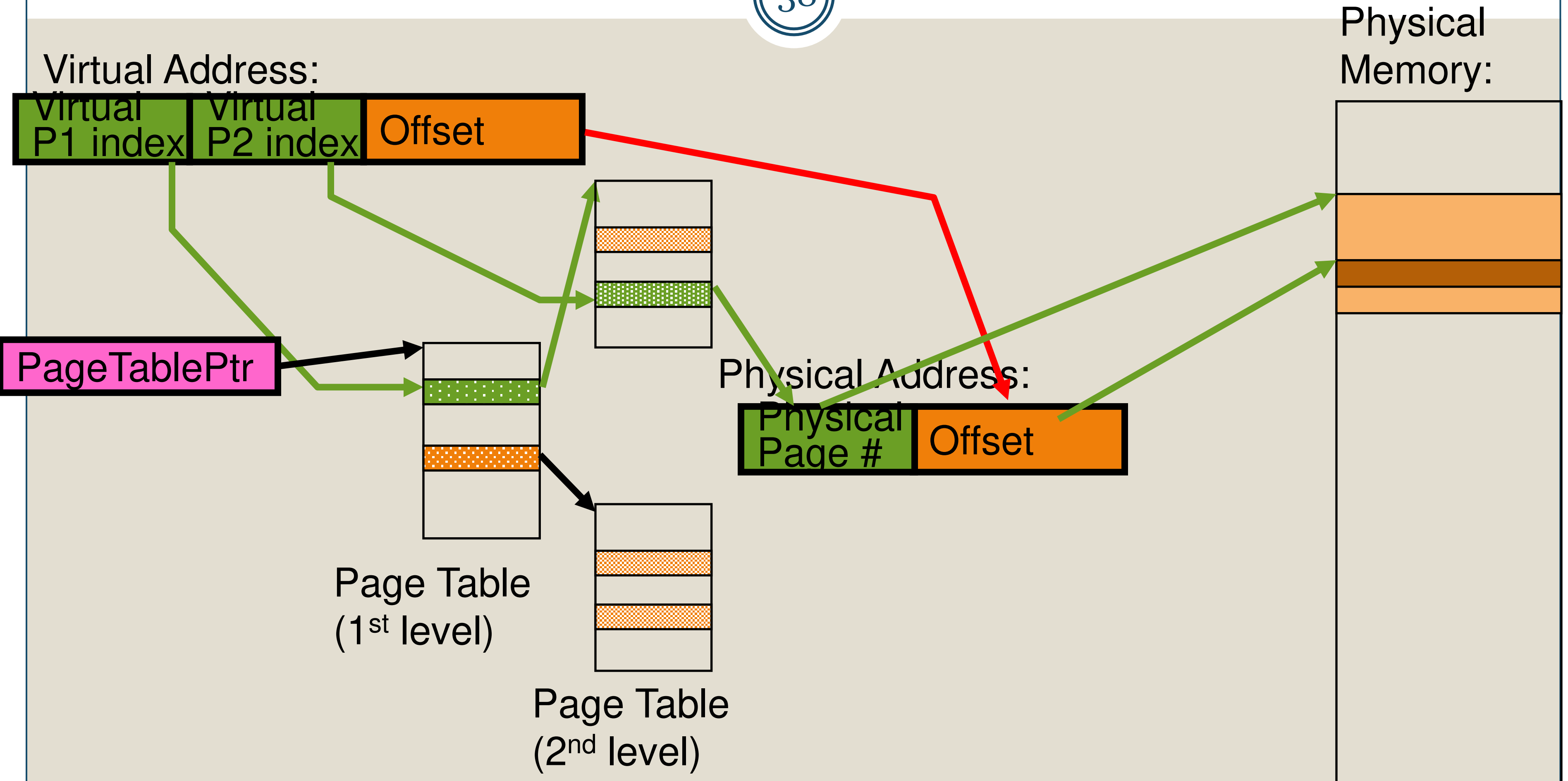
Putting Everything Together: Address Translation



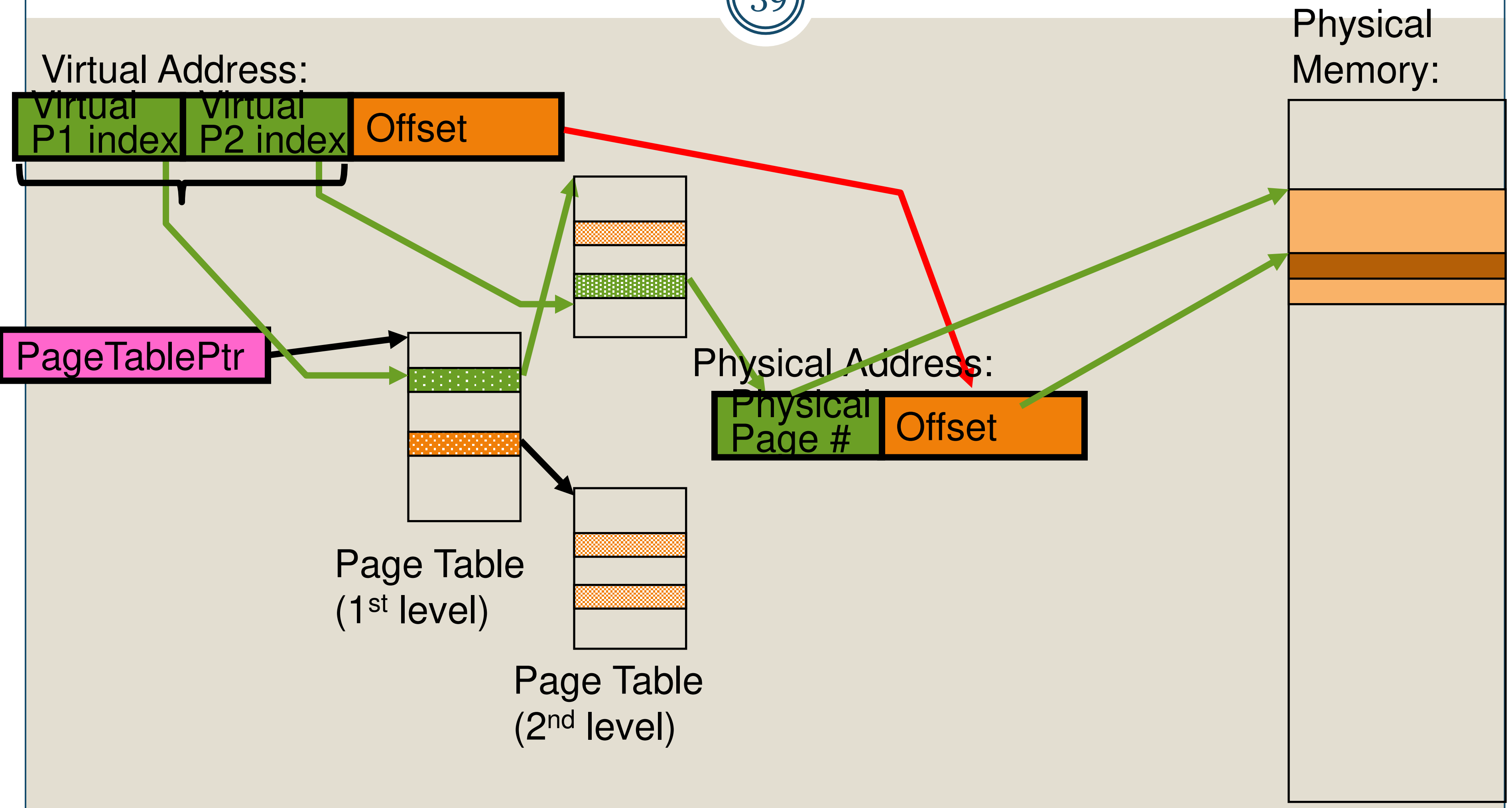
Putting Everything Together: Address Translation



Putting Everything Together: Address Translation

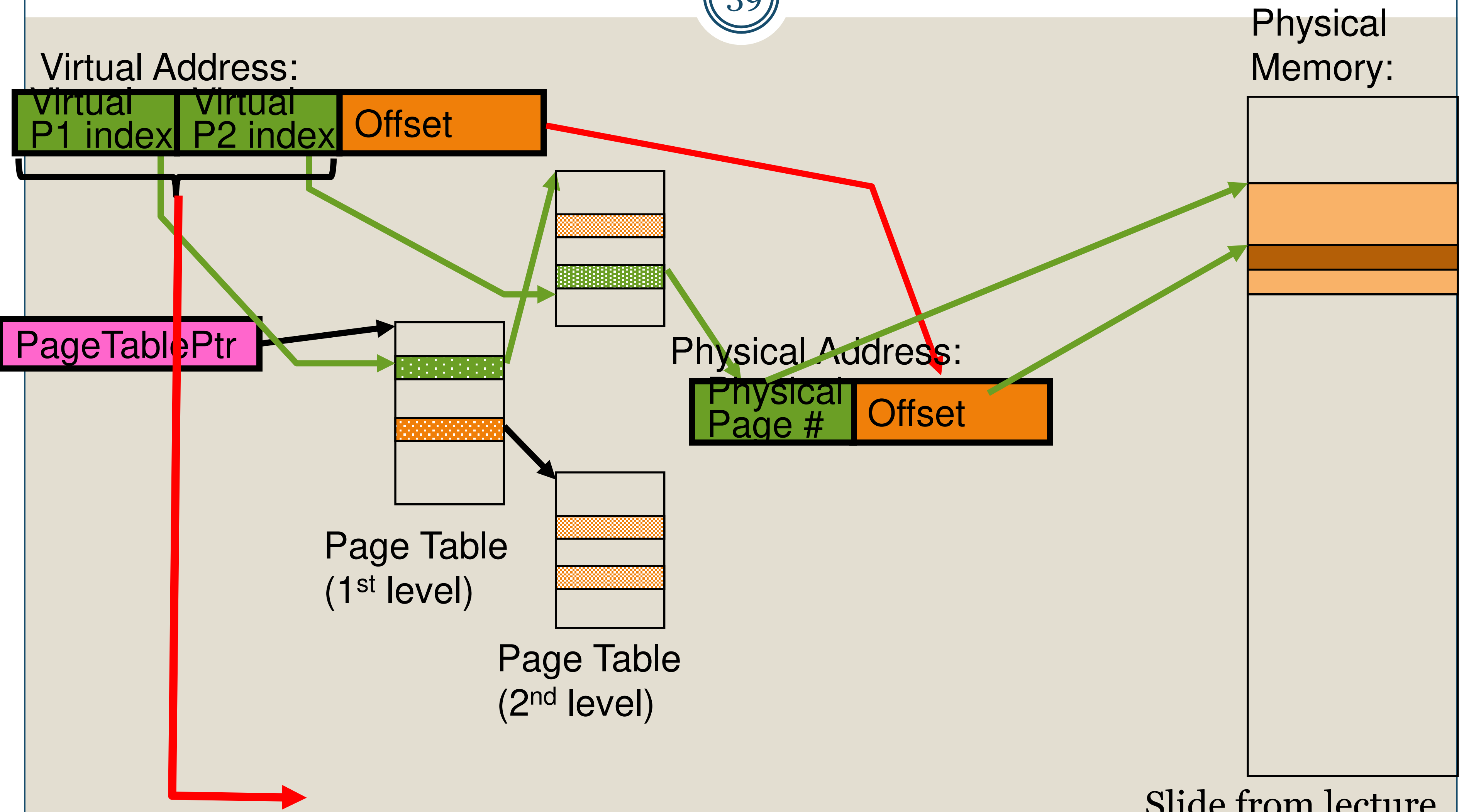


Putting Everything Together: TLB



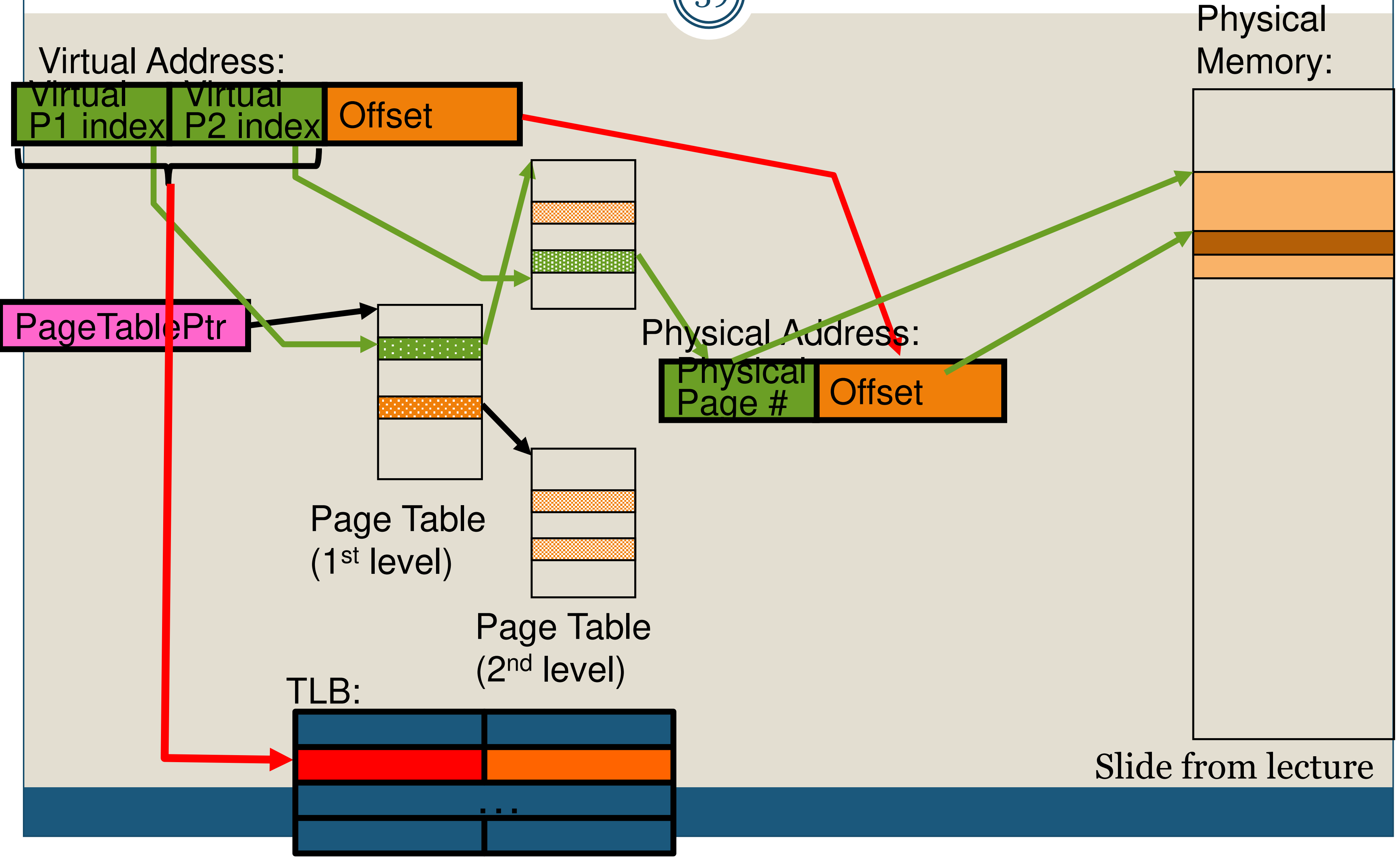
Slide from lecture

Putting Everything Together: TLB



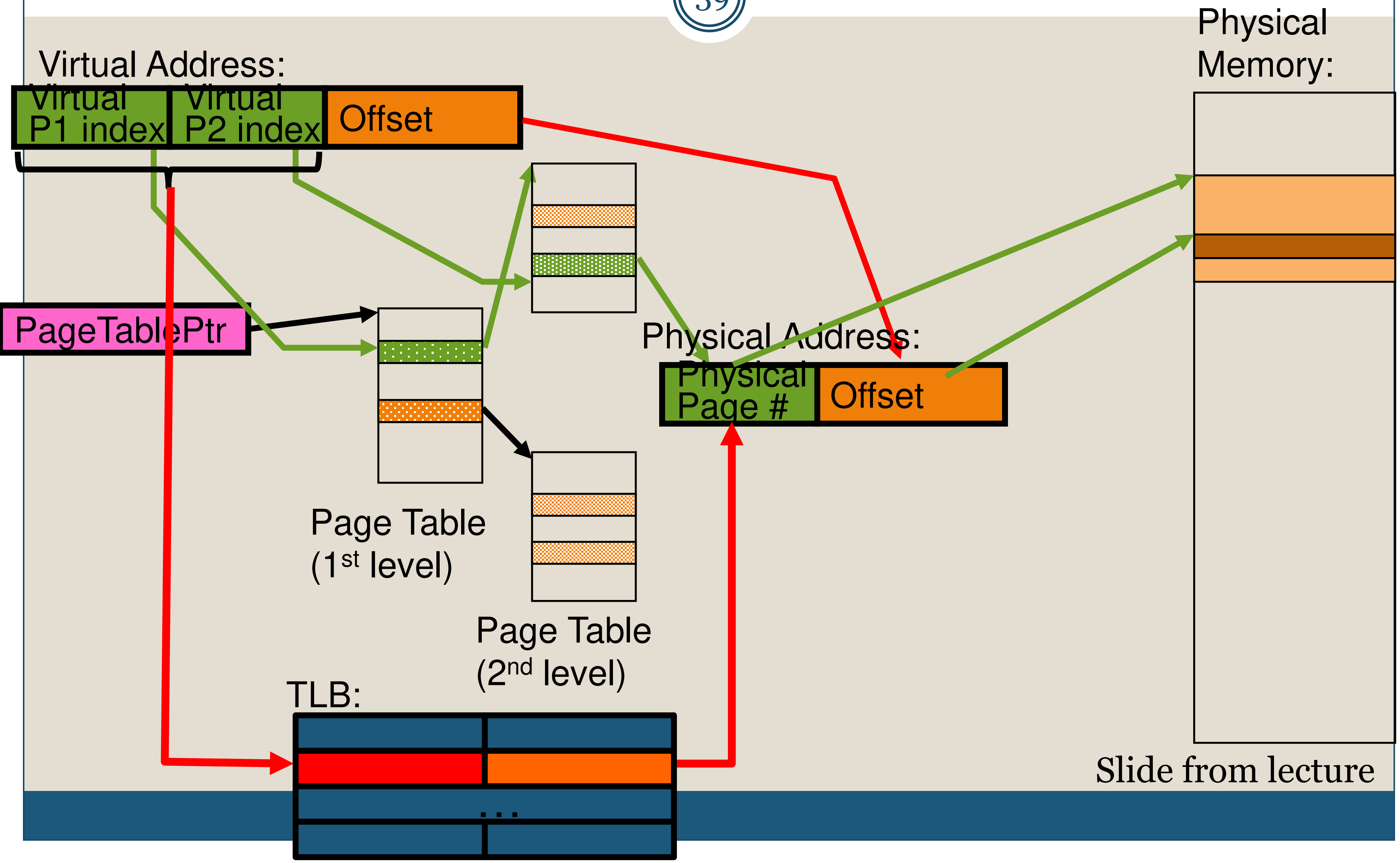
Slide from lecture

Putting Everything Together: TLB



Slide from lecture

Putting Everything Together: TLB



Slide from lecture

Ex 4

- In EX 4 you will implement a virtual memory interface using hierarchical page tables of arbitrary depth using simulated physical memory.
- You'll have to implement 2 functions:
  - VMread(*virtualAddress, *value*) – reads the word from the virtual address *virtualAddress* into **value*. Returns 1 on success and 0 on failure.
  - VMwrite(*virtualAddress, value*) – writes the word *value* into the virtual address *virtualAddress*. Returns 1 on success and 0 on failure.

Q & A