Operating systems

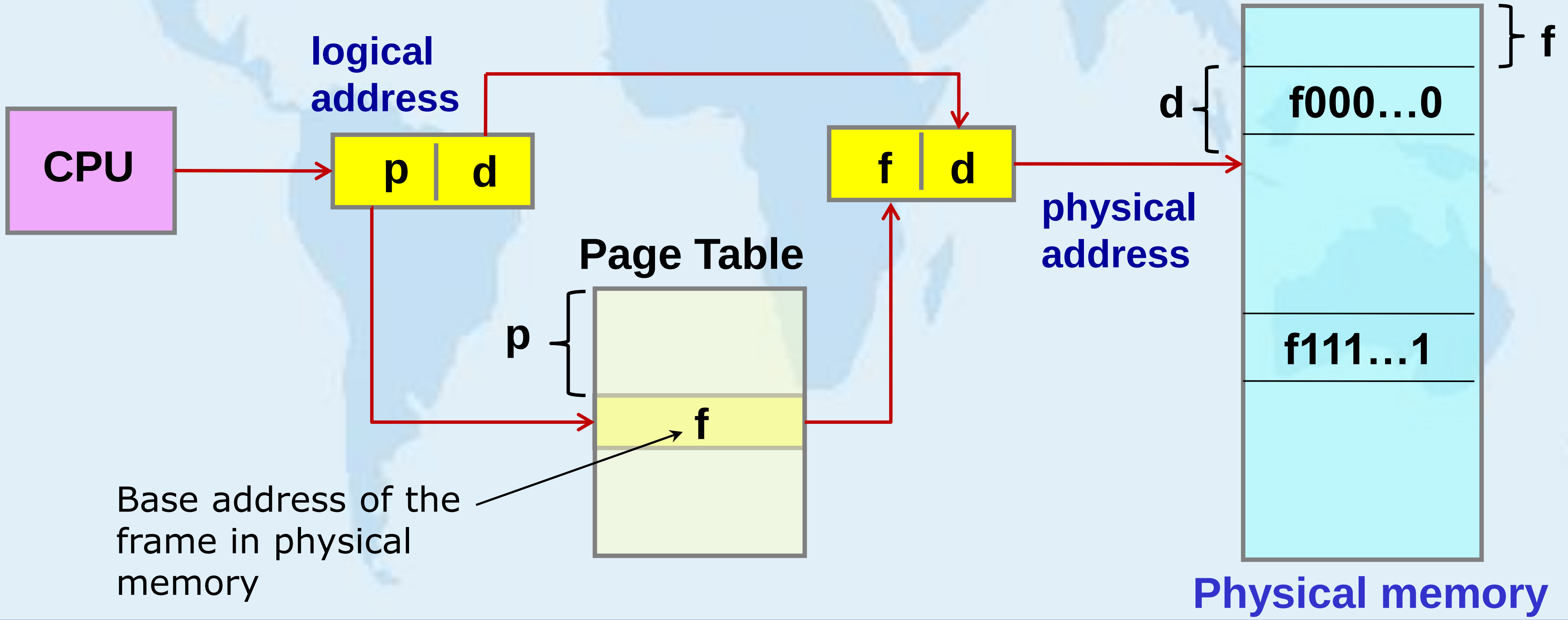
Lecture 7:
Memory management
- Paging
- Segmentation

Paging

- **Paging** – memory management scheme that allows physical address space to be non-contiguous.
- **Basic idea:**
  - Divide <u>logical</u> memory into fixed-sized blocks (size is power of 2, between 512 and 8192 bytes) called **pages.**
  - Divide <u>physical</u> memory into blocks of the same size called **page frames.**
  - Keep track of all free page frames.
  - To run a program of size **n** pages, need to find **n** free page frames and load the program.

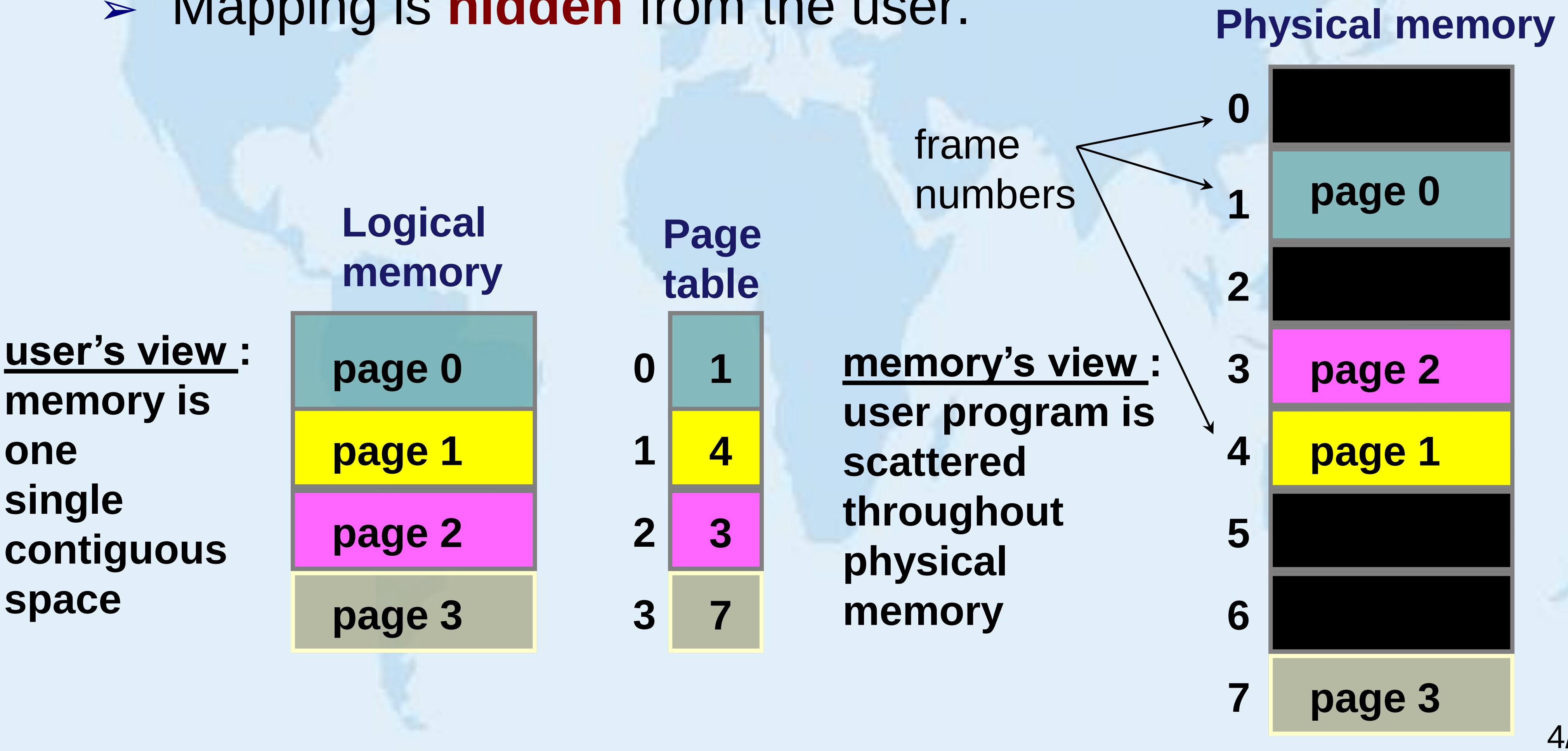
Paging

- Addresses generated by the CPU are divided into:
  - **Page number (p)** - used as an index into a **Page Table** which contains base address of each page in the physical memory.
  - **Page offset (d)** - combined with base address to define the physical memory address that is sent to the memory unit.

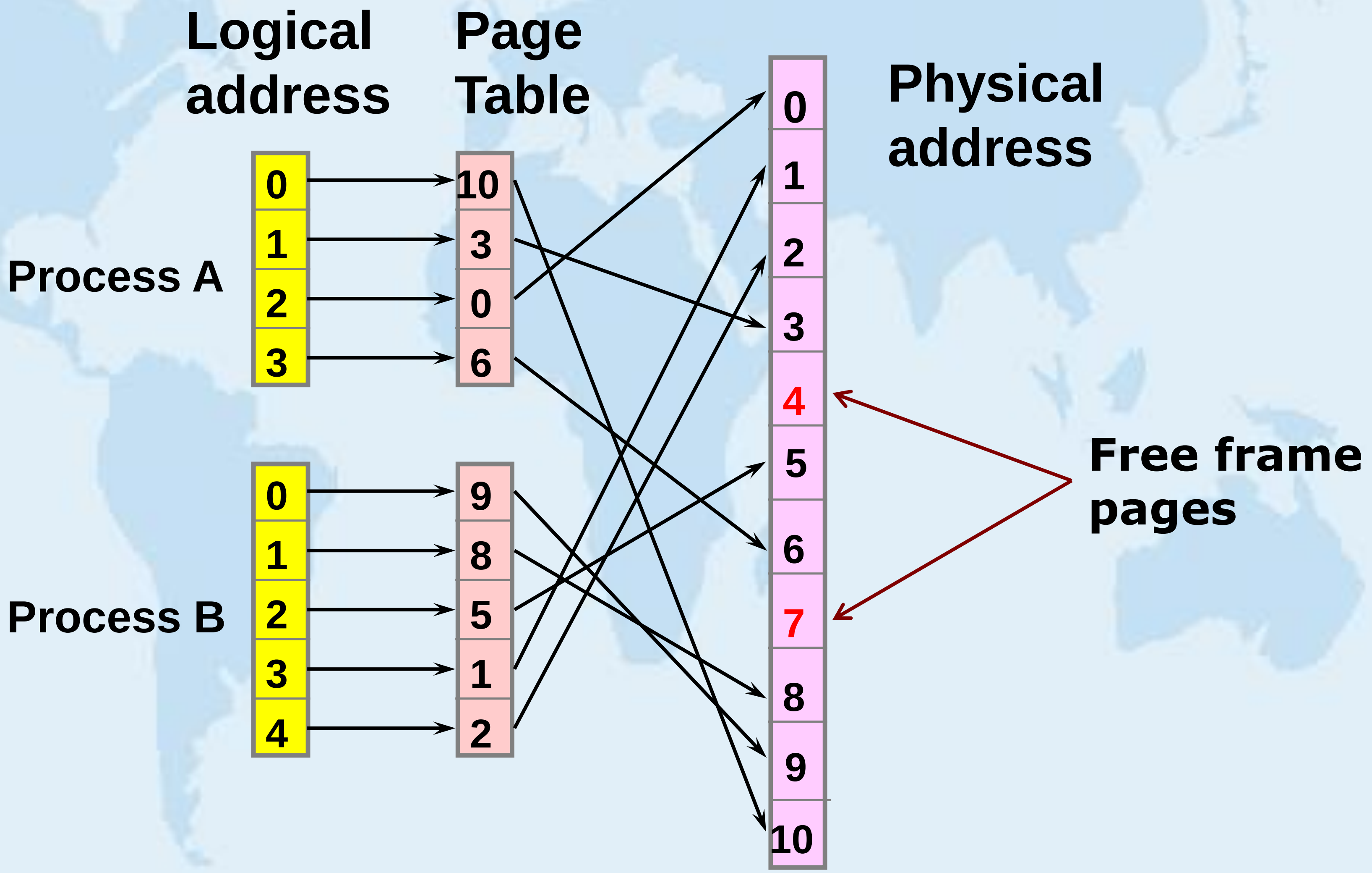


Paging

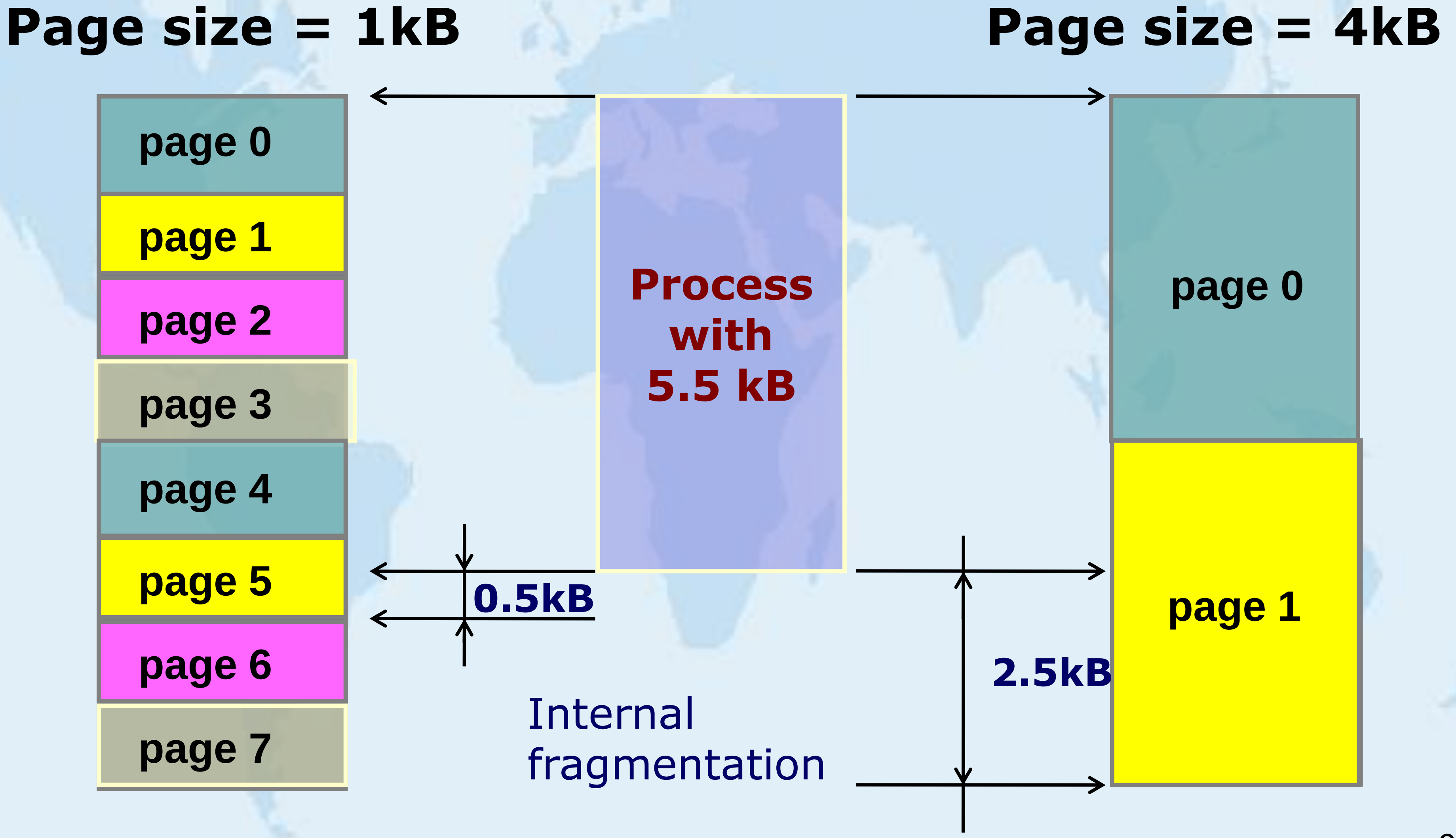
- **Logical-to-physical** address mapping uses paging.
- Mapping is **hidden** from the user.



Paging Example

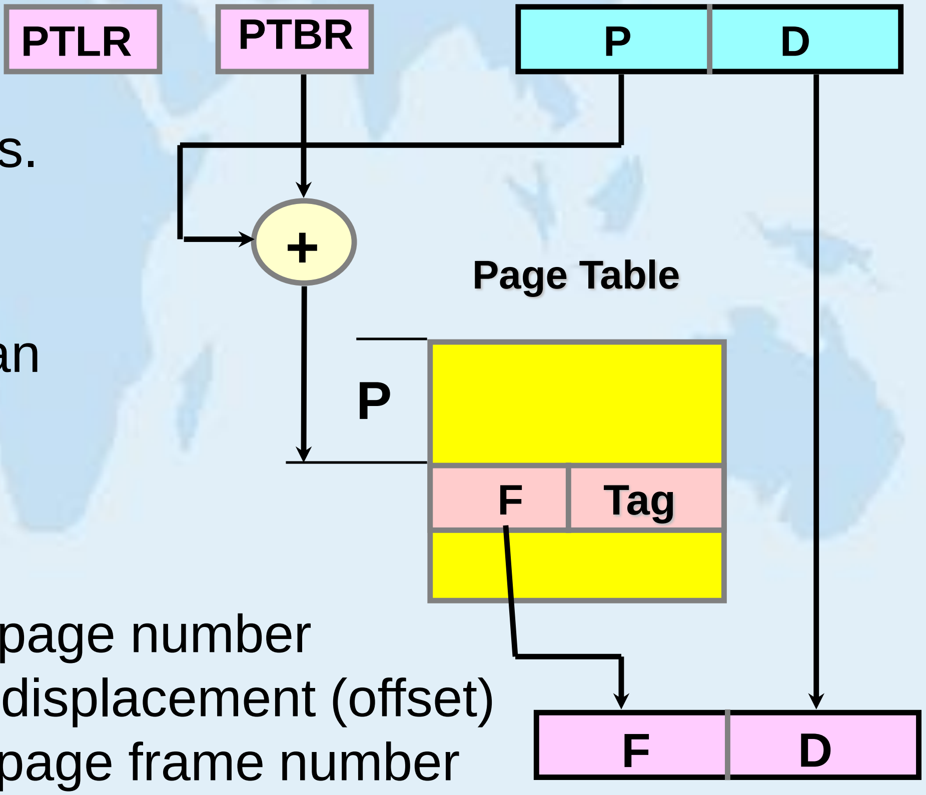


Page Size


Page size = 1kB
Page size = 4kB
page 0
page 1
page 2
page 3
page 4
page 5
page 6
page 7
Process with 5.5 kB
0.5kB
page 0
page 1
2.5kB
Internal fragmentation

Page Table

- Page table is kept in **main** memory. One table per process.
- **Page-table base register (PTBR)** points to the page table.
- **Page-table length register (PTLR)** indicates size of page table.

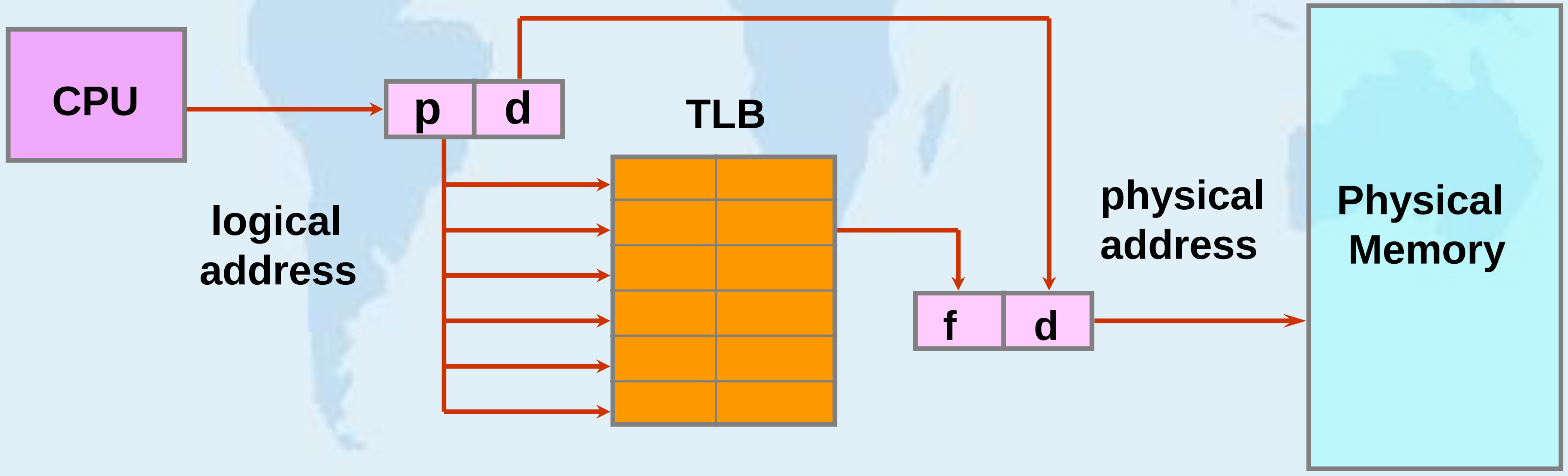


- In this scheme every data/instruction access requires **two** memory accesses. One for the page table and one for the data/instruction.
- The two memory accesses problem can be solved by the use of a special fast-lookup hardware cache called **associative registers** or **translation look-aside buffers (TLB).**

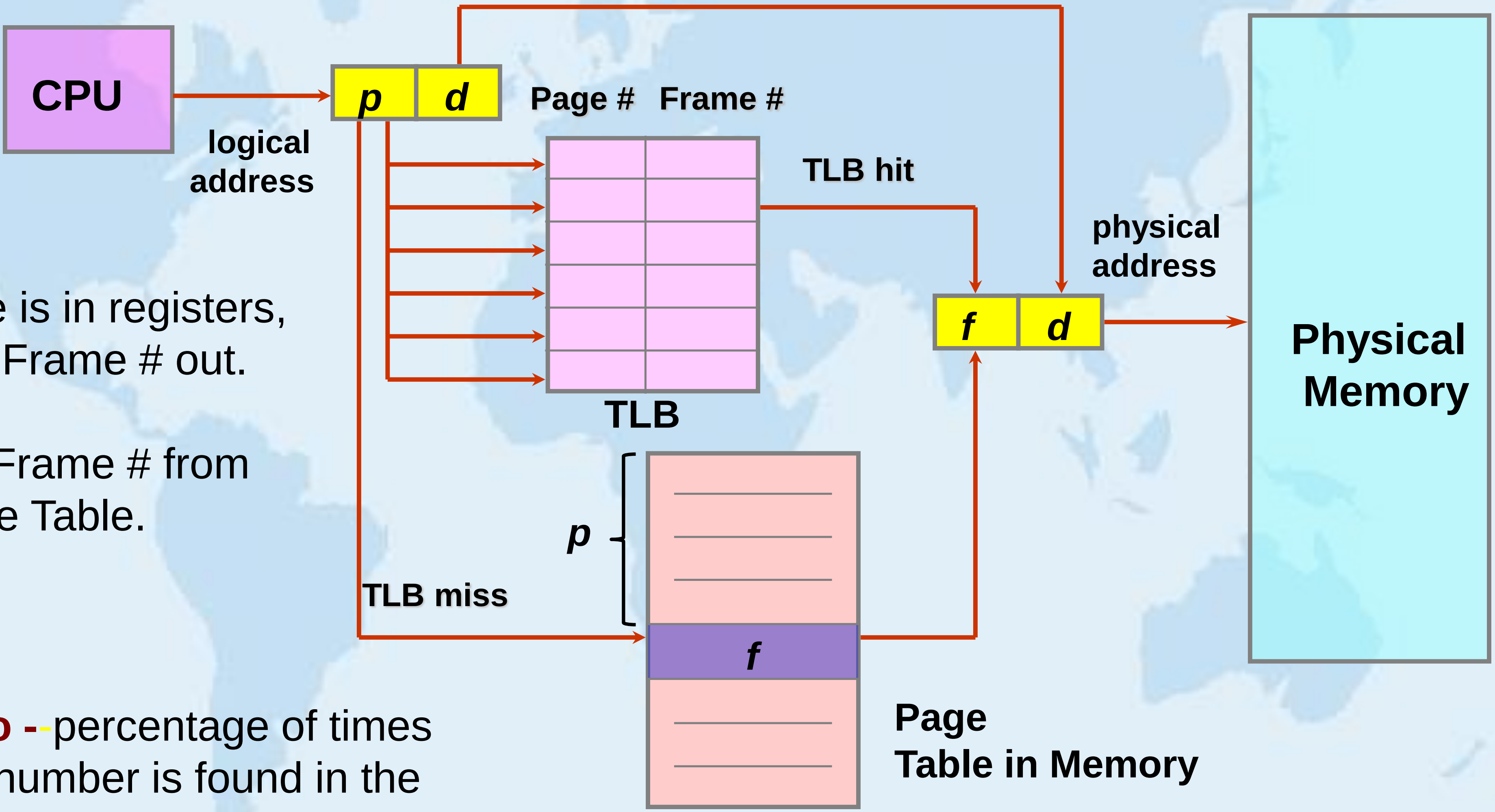
P - page number
D - displacement (offset)
F - page frame number

Associative Memory TLB

- **Associative memory** - Memory organized in such a way that memory cells can be searched (in parallel) by contents, not the address or index.
- Implemented with a high-speed logic such as **registers**.
- Used to store and read a **small** amount of frequently used information (64 – 1024 bytes).



TLB Paging Hardware



-If page is in registers,
get Frame # out.
- **Else**
get Frame # from
Page Table.

Hit ratio -- percentage of times a page number is found in the registers; this ratio is related to number of associative registers.

Effective Access Time

- **Example:**
  - TLB access time: T_B (~ 20 ns)
  - Memory access time: T_M (~ 100 ns)
  - Hit ratio: P_{HIT} (80% ~ 98%)

  - Effective Memory Access Time (**EAT**):

$$\mathbf{T_E = P_{HIT}\,(T_B + T_M) + P_{MISS}(T_B + T_M + T_M)}$$

$$\mathbf{P_{HIT} + P_{MISS} = 1}$$

P_{hit} = 80% **EAT** = 0.80 x 120 + 0.20 x 220 = 140 ns (40% slowdown)

P_{hit} = 98% **EAT** = 0.98 x 120 + 0.02 x 220 = 122 ns (22% slowdown)

Memory Protection

- Protection **bits** are associated with each frame.
  - **Read-write** – process can read and write to this frame.
  - **Read-only** – process can only read from this frame
  - **Valid-invalid** -
    "**valid**" indicates that the associated page is in the process' logical address space, and is thus a legal page.

    "**invalid**" indicates that the page is not in the process' logical address space.

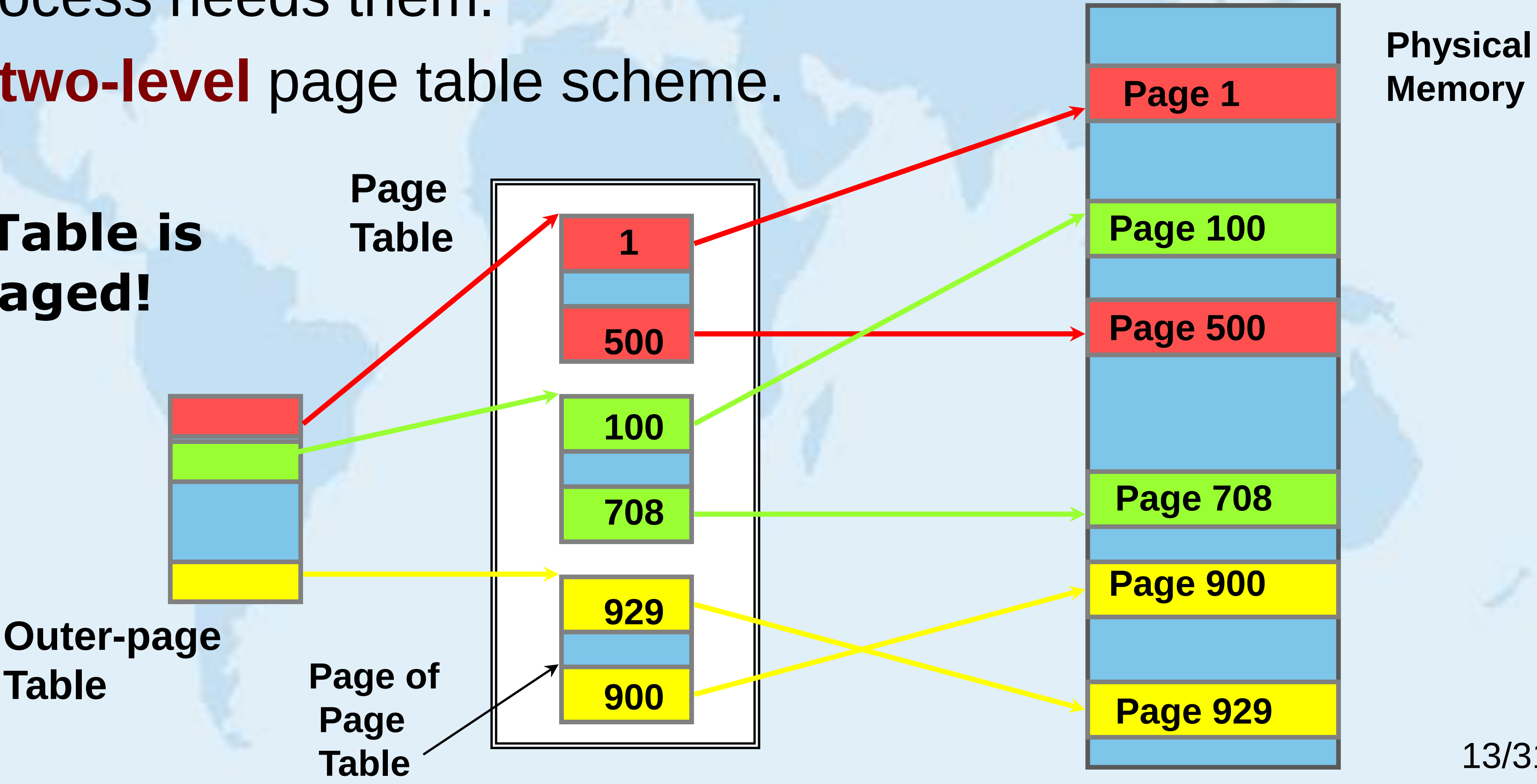
Avoiding Table Fragmentation

- Each process has a Page Table which **takes** memory:
  - 32-bit logical address space 4GB (2^{32}).
  - Page size is 4K bytes (2^{12}).
  - Page Table may consist of up to 1 M entries ($2^{32}/ 2^{12}$).
    - Each entry is 4 bytes => each process needs up to 4 Mbytes of physical address space for the Page Table.
- **Problem** – Need 4M of **contiguous** main memory!
- Solutions:
  - – **Multilevel Paging.**
  - – **Hashed Page Table.**
  - – **Inverted Page Table.**

Multilevel Paging

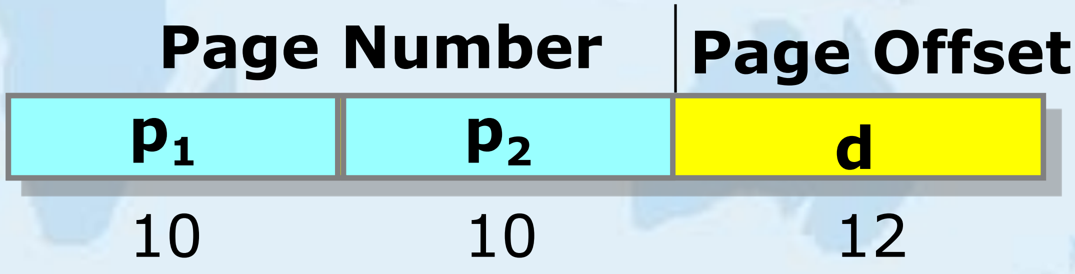
- **Multilevel Paging** - partitioning the Page Table allows the operating system to leave partitions unused until a process needs them.
- A **two-level** page table scheme.

Page Table is also paged!



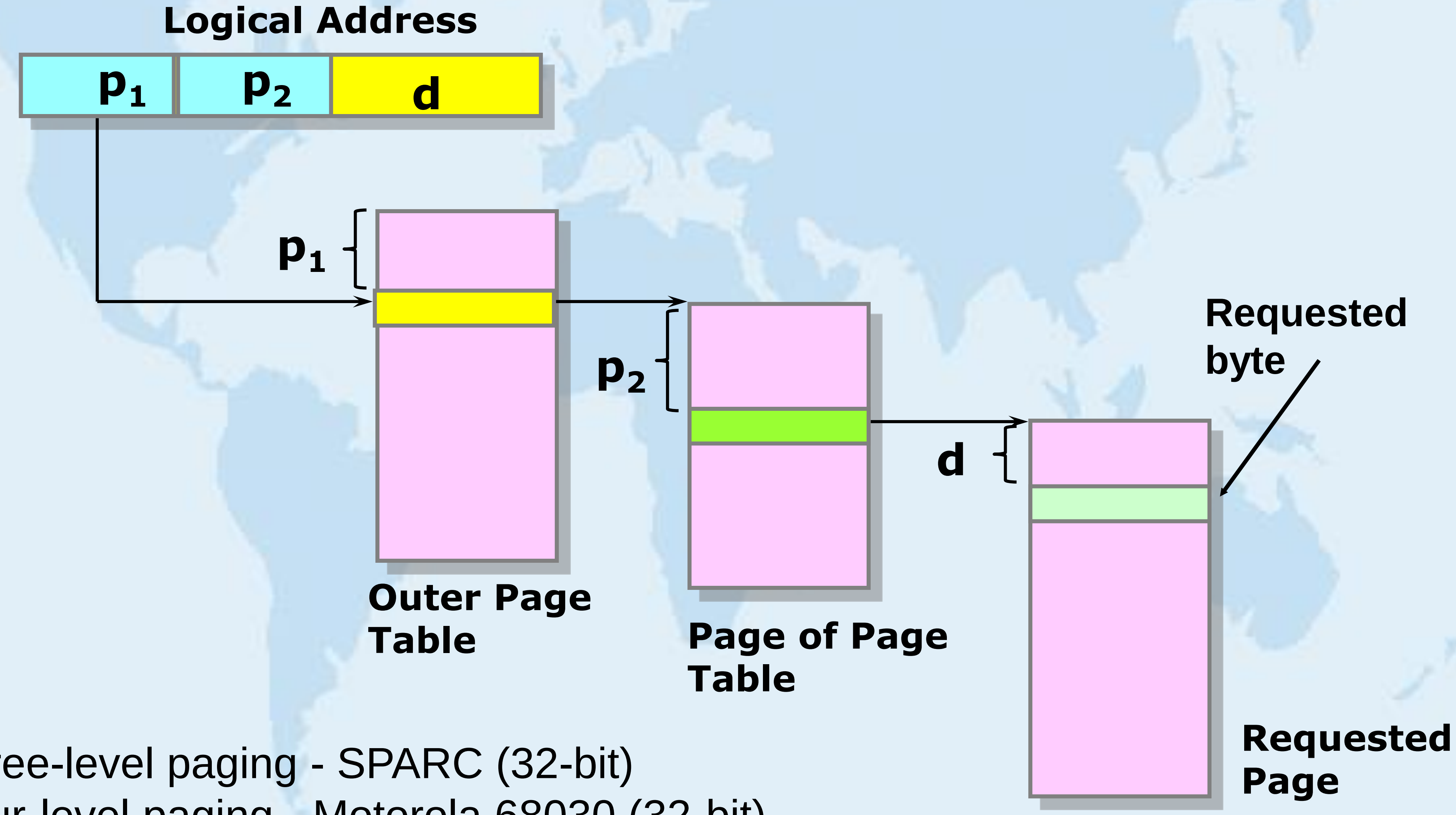
Multilevel Paging

- **Example:** 32-bit machine with 4K page size.
- A logical address is divided into:
  - a **page number** consisting of 20 bits.
  - a **page offset** consisting of 12 bits.
- Since the Page Table is paged, the page number is further divided into:
  - a 10-bit page number $\mathbf{p_1}$.
  - a 10 bit page offset $\mathbf{p_2}$.
- Thus, a logical address is as follows:



p_1: index of the Outer Page Table.
p_2: displacement within the page of the Outer-page Table.

Two-level Paging



Three-level paging - SPARC (32-bit)
Four-level paging - Motorola 68030 (32-bit)

Multilevel Paging Performance

- Since each level is stored as a separate table in memory, converting a logical address to a physical one may take up to **four** memory accesses.
- Even though time needed for one memory access is quintupled (five times as much), **caching** permits performance to remain reasonable.
- Cache hit ratio of 98% yields:

  EAT = 0.98 x 120 + 0.02 x 520 = 128 ns.

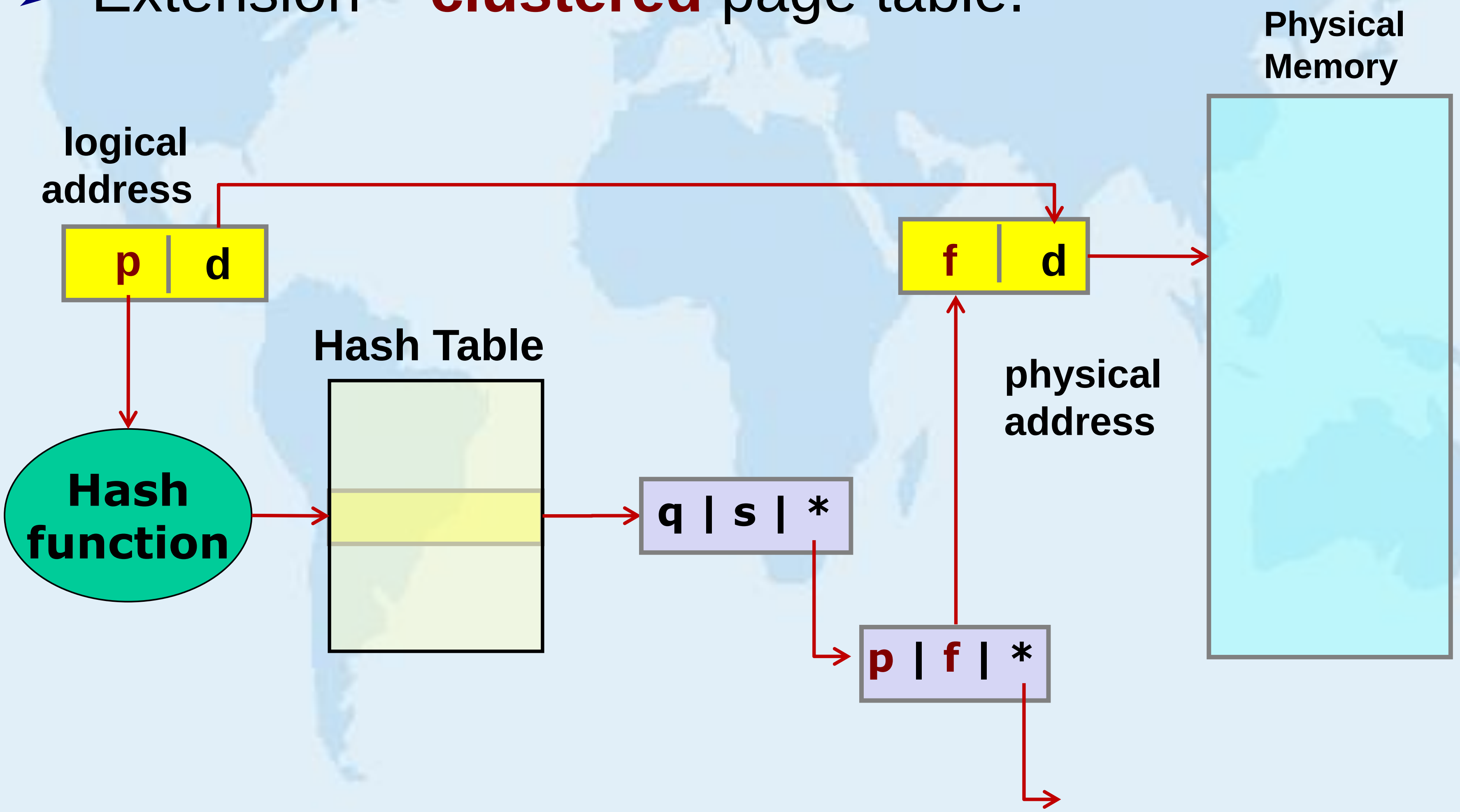
  which is only a 28% slowdown in memory access time.

Hashed Page Table

- For handling address spaces **larger** than 32 bits.
- Uses a **hash** table:
  - Hash value – virtual page number.
  - Each table entry - linked list of elements:
    a) Virtual page number.
    b) Value of mapped page frame.
    c) Pointer to the next element.
- Algorithm:
  1. Virtual page number is **hashed** into the table.
  2. It is **compared** with the field a). If match, field b) is used.
  3. Else, **next** element is checked.

Hashed Page Table

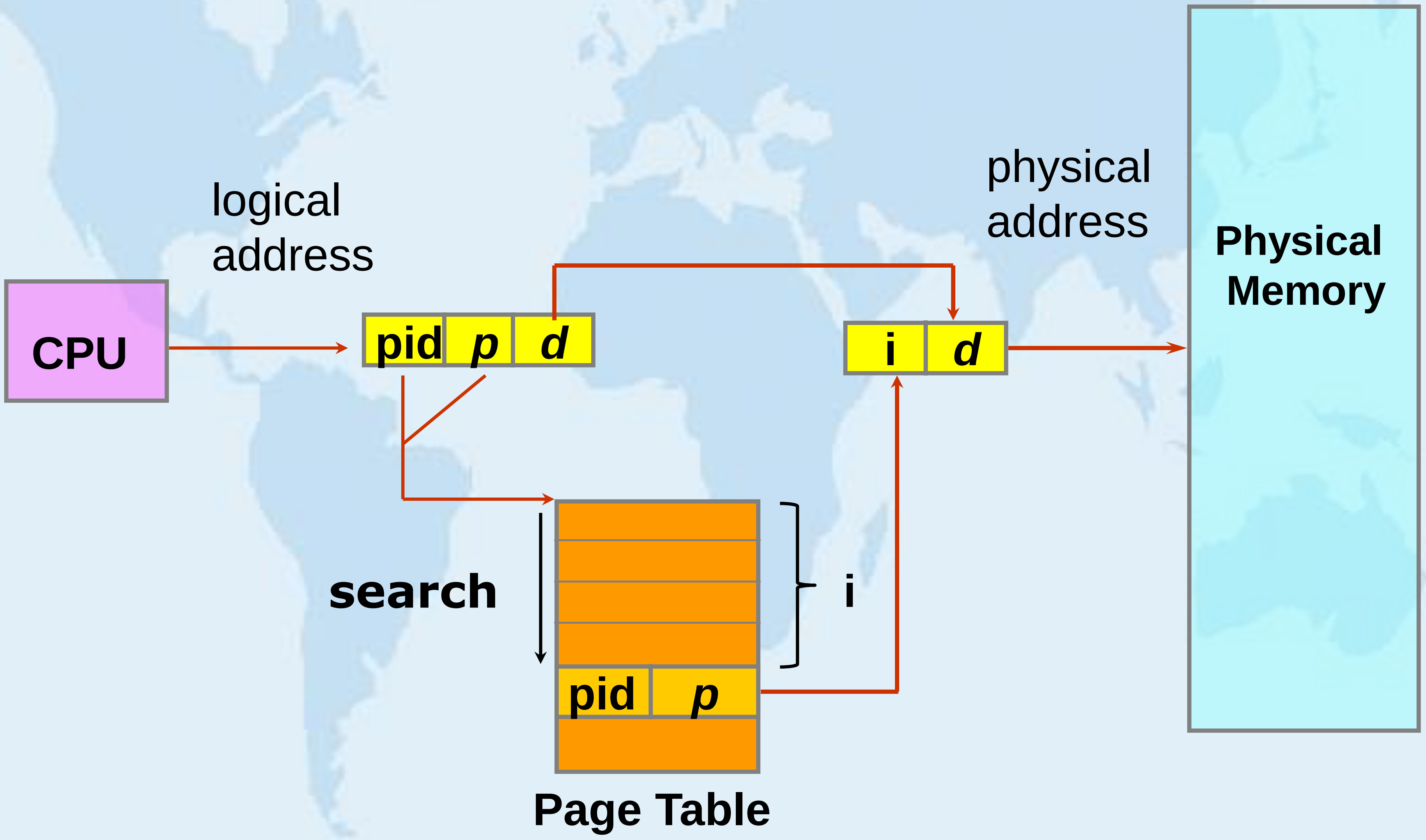
- Used for handling address spaces **larger** than 32 bits.
- Extension – **clustered** page table.



Inverted Page Table

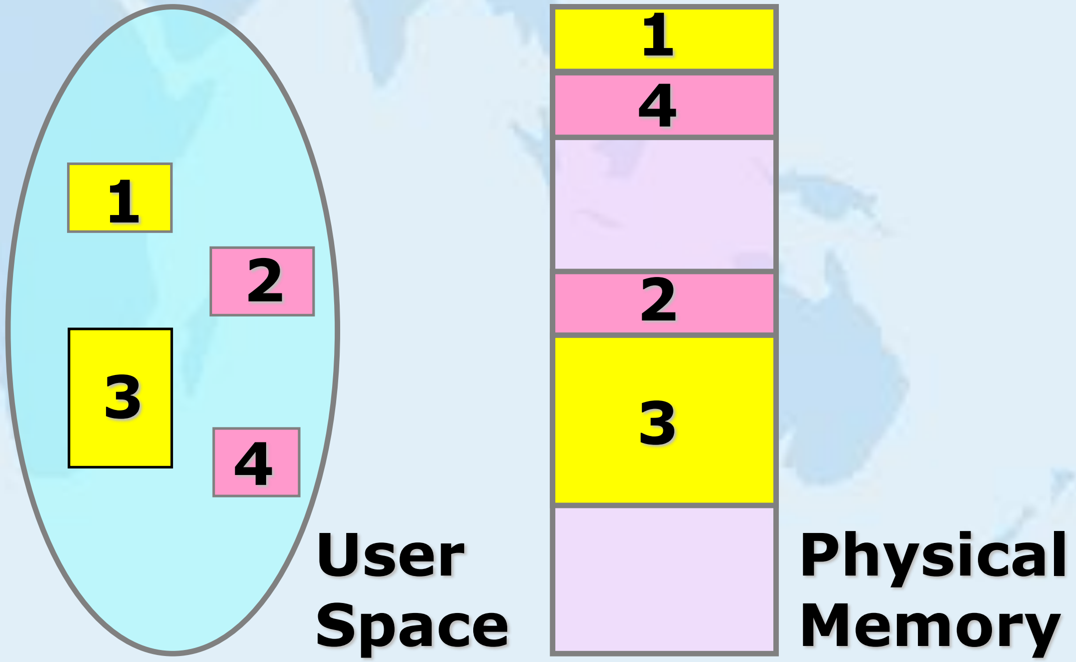
- **Inverted Page Table** - one entry for each real frame page of memory;
- **Inverted Page Table** - only one for all processes (before - each process has a own Page Table).
- Each table entry consists of the virtual address of the page stored in that real memory location, with information about the **process** that owns that page.
  - **Decreases** memory needed to store each Page Table.
  - **Increases** time needed to search the table.
- Example systems:
  - 64-bit UltraSparc, PowerPC

Inverted Page Table



Segmentation

- **Segmentation** - memory-management scheme that supports user view of memory.
- A program is a collection of segments. Each segment has a name and a length. A segment is a local unit such as:
  - Main Program;
  - Procedure;
  - Function;
  - Local / Global Variables;
  - Common Block;
  - Stack;
  - Symbol Table, Arrays.



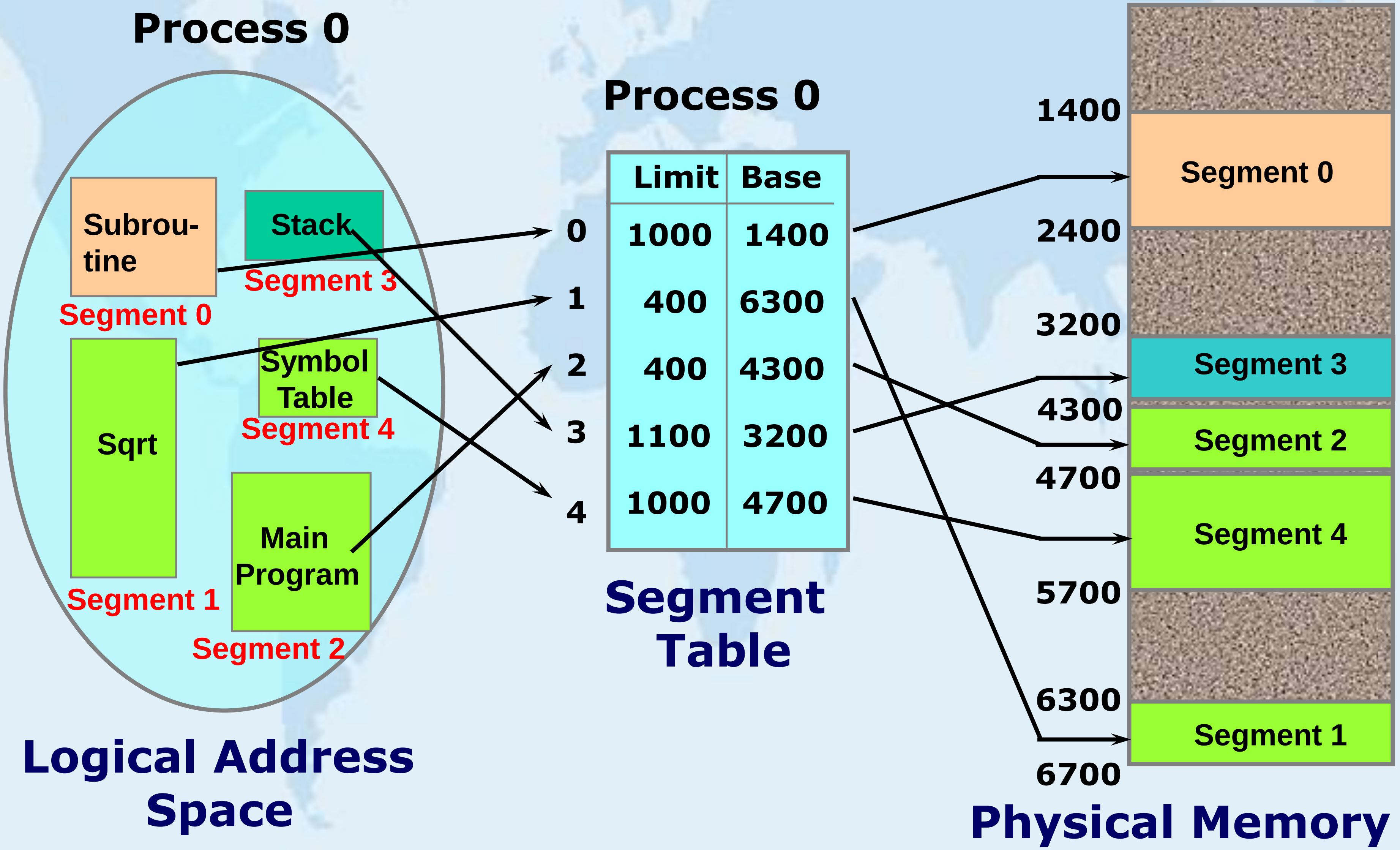
Segmentation

- Logical address consists of following tuple:

  < segment-number, offset >

- **Segment Table** - maps two-dimensional user-defined addresses into one-dimensional physical addresses;
- Each entry of the **Segment Table** has:
  - **base** - contains the starting physical address where the segments reside in memory.
  - **limit** - specifies the length of the segment.

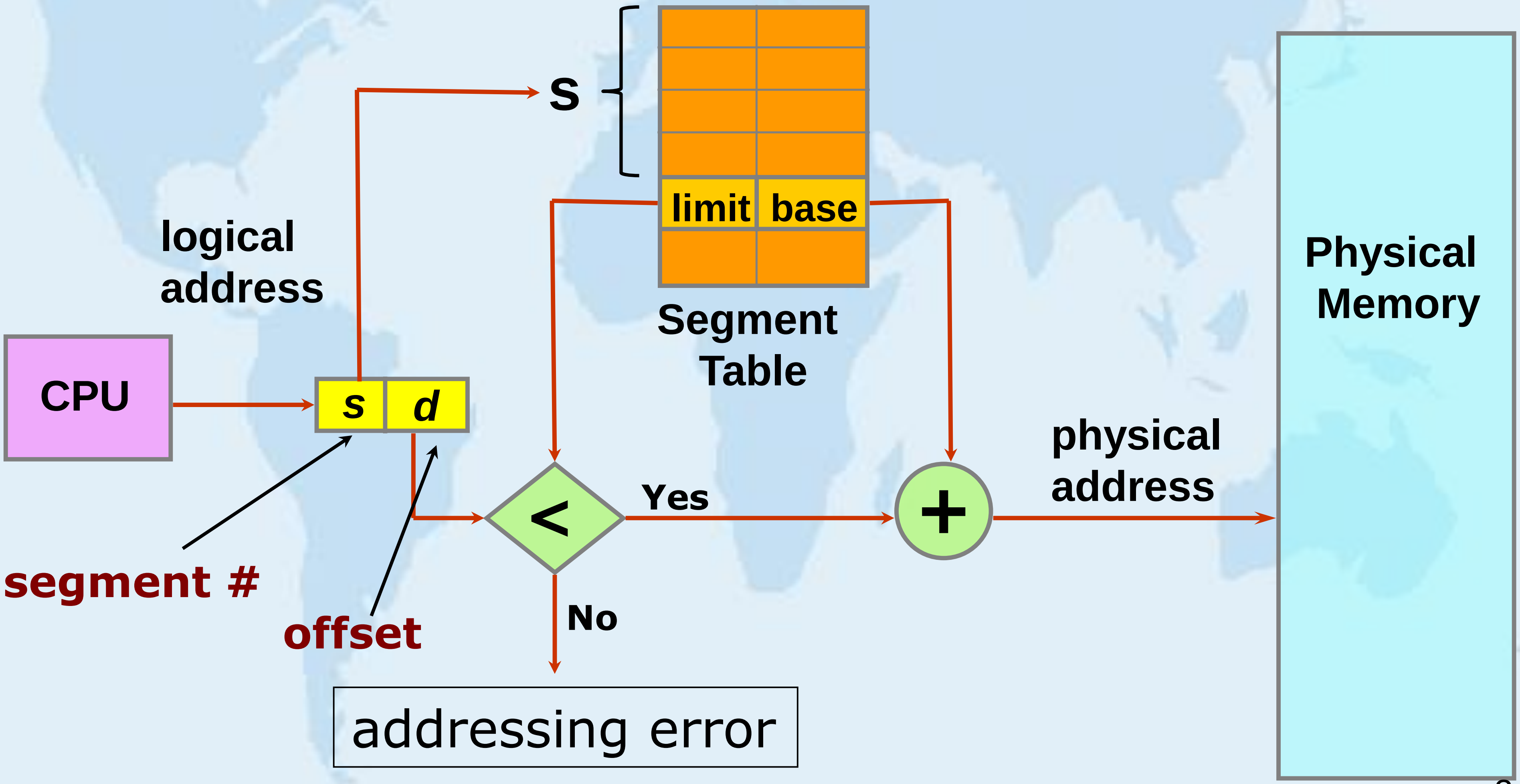
Segmentation



	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

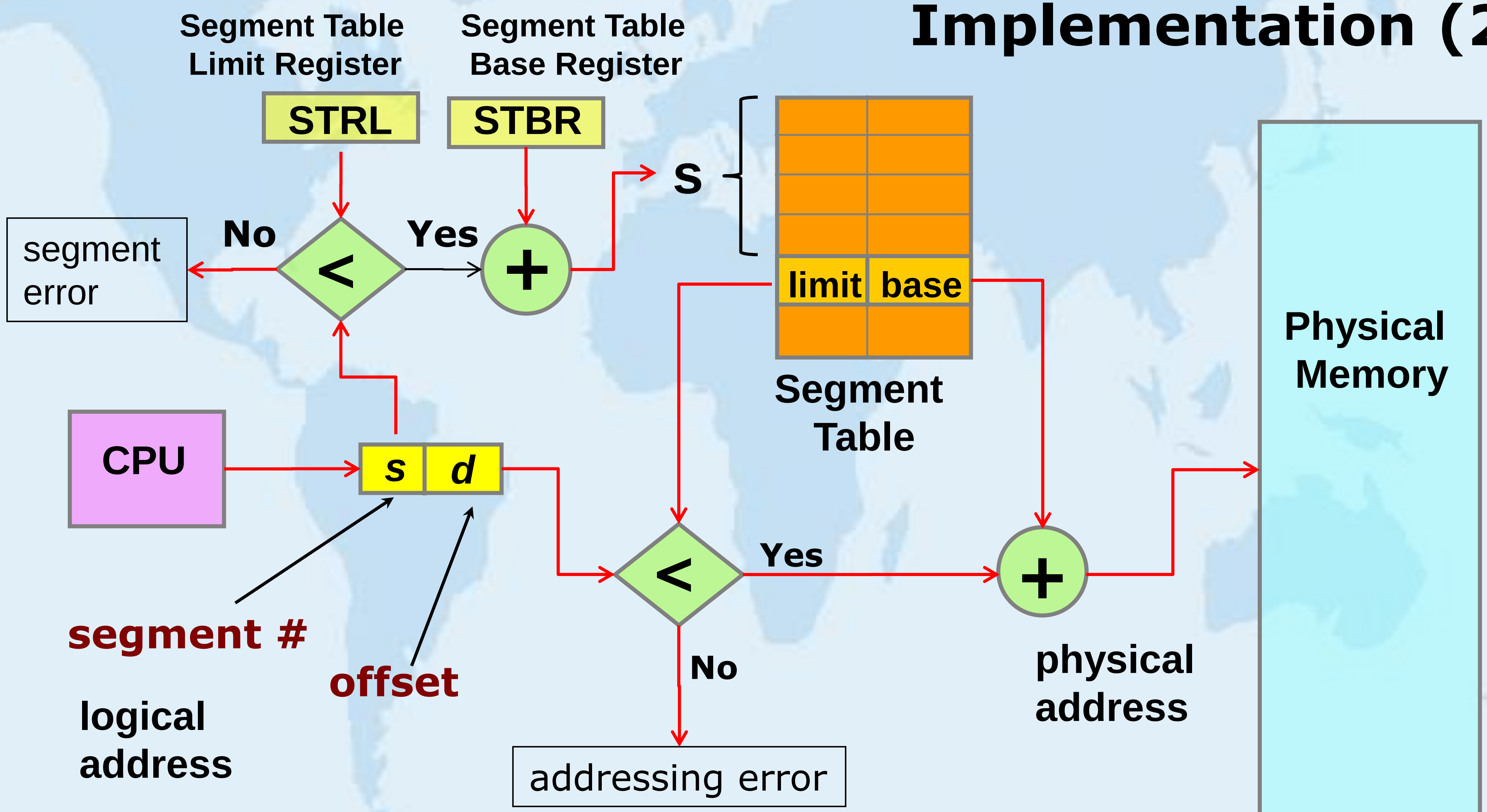
Segmentation

Implementation (1)



Segmentation

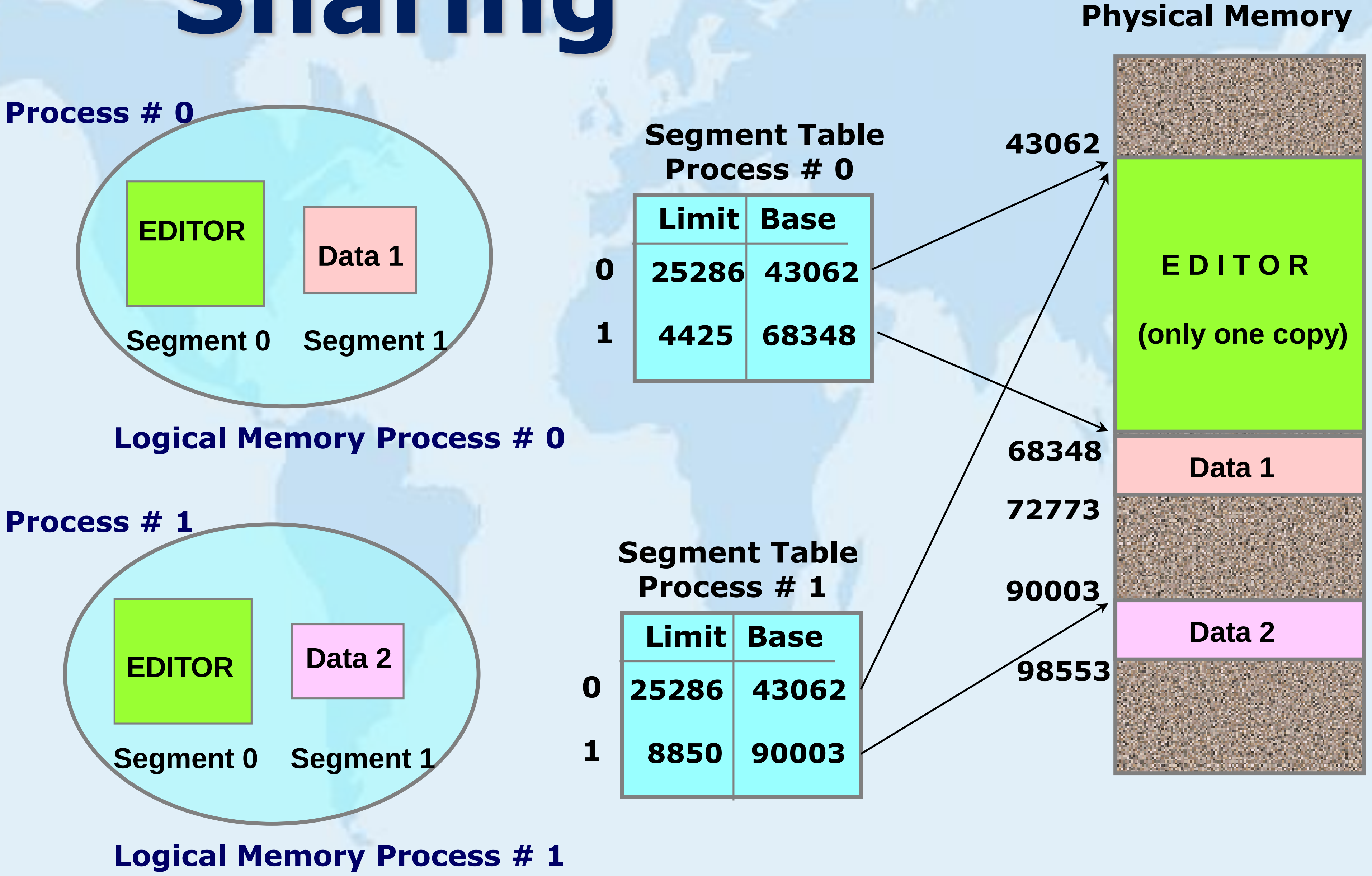
Implementation (2)



Segmentation

- **Protection** and **Sharing** of data can be implemented naturally at the segment level.
  - **Protection:** segment - semantically defined portion of program;
    - instructions $\rightarrow$ execute-only or read-only;
    - data $\rightarrow$ read- or write-only);
    - protection bits associated with segments.
  - **Sharing:** segments are shared when entries in the Segment Tables of two different processes point to the same physical locations.

Segmentation - Sharing



Segment Table Process # 0

	Limit	Base
0	25286	43062
1	4425	68348

Segment Table Process # 1

	Limit	Base
0	25286	43062
1	8850	90003

Segmentation-Issues

- Since segments vary in length, memory allocation is **dynamic storage-allocation** problem (best-fit or first-fit).
- Segmentation may cause **external fragmentation**, when all blocks of free memory (holes) are too small to accommodate a segment.
- But we can compact memory whenever we want. Average segment size?
  - **One extreme:** each process to be one segment.
  - **The other extreme:** every byte put in its own segment (doubling memory use!).

Segmentation-Issues

- The next logical step - fixed-sized, small segments - is **paging.**
- It is possible to combine **Segmentation** and **Paging** to improve on each.
- This combination is best illustrated by two different architectures: MULTICS system and Intel 386.

Segmentation with Paging

- **MULTICS Operating System**
  - Segmentation with Paging.
  - Dynamic linking.
  - Ring protection.
  - Most code was written in PL/1.
- **Hardware**
  - GE 635 was modified into GE 645.
  - Word machine; 1 word = 36 bits.
  - Max. # of segments = 256 K.
  - Segment size = 64 K words.
  - Page size = 1 K words.

That is all for today!