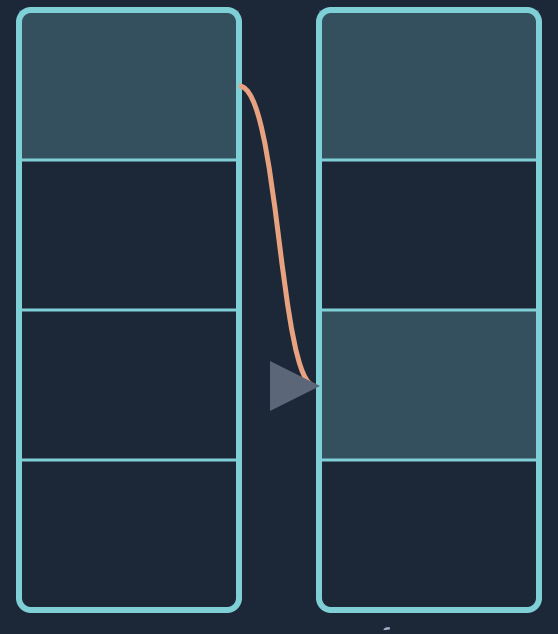
OPERATING SYSTEMS · LECTURE 8

Main Memory

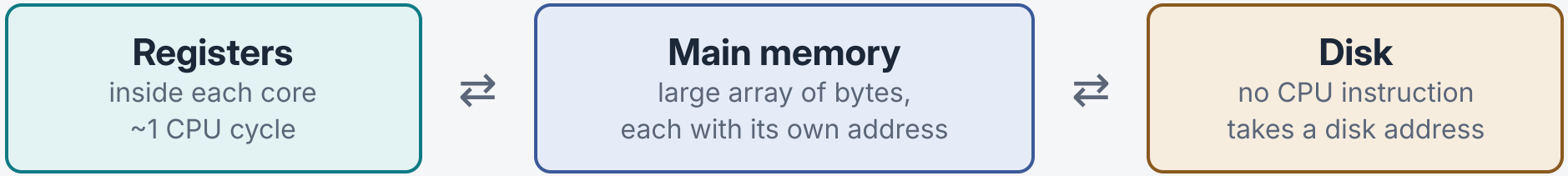
Address spaces, allocation, paging and page tables



Lecturer: Aidana Karibayeva

01 What the CPU can access directly

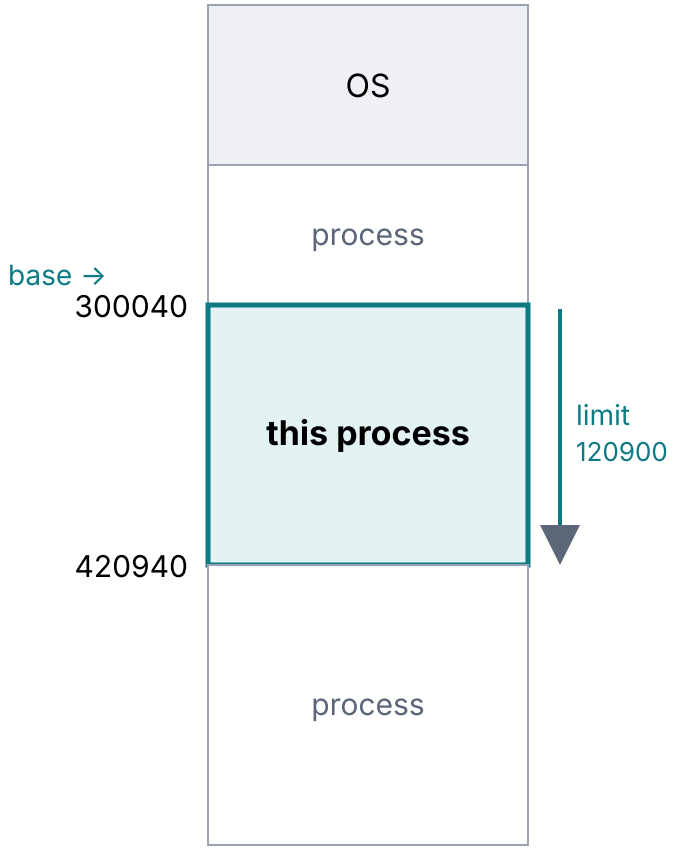
Only **registers** and **main memory**. Data on disk must be loaded into memory before the CPU can use it.



The CPU fetches instructions from memory using the **program counter**.

Instructions may load from and store to memory addresses.

02 Protection: base and limit registers



CPU address → base ≤ address < base + limit ? → memory

No → **trap** to the OS (addressing error).

Legal range	`300040 … 420939`
Who loads the registers	only the OS, with a privileged instruction in kernel mode
Protects	the OS and other users from a process

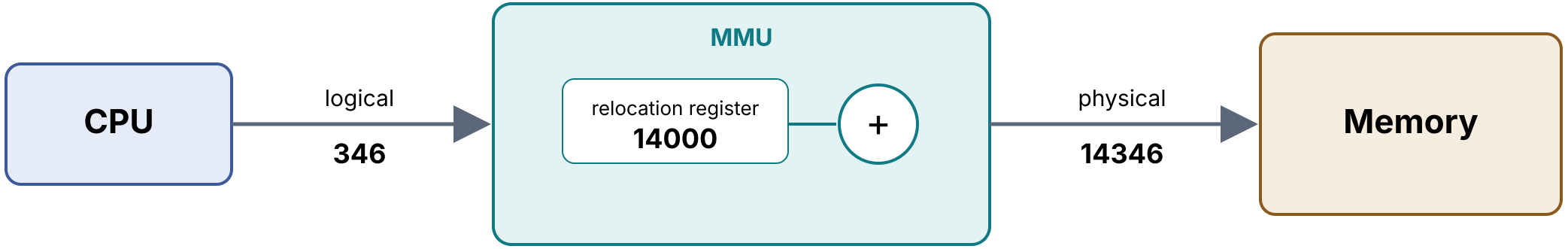
03 Address binding

A process can live anywhere in physical memory. Its addresses are bound to physical ones at one of three moments.

WHEN	WHAT HAPPENS	LOGICAL = PHYSICAL?
Compile time	Location known in advance; absolute code is generated	yes
Load time	Relocatable code; addresses fixed when the program is loaded	yes
Execution time	Process can move during execution; needs hardware (MMU)	no

04 Logical vs physical addresses and the MMU

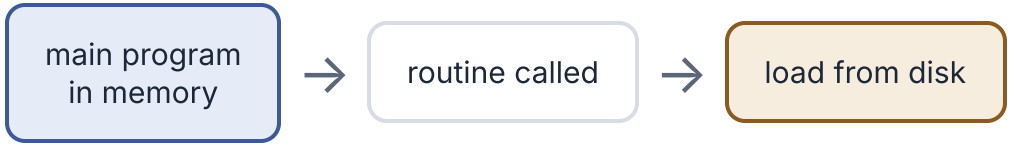
	LOGICAL (VIRTUAL) ADDRESS	PHYSICAL ADDRESS
Who sees it	generated by the CPU / program	seen by the memory unit
Address space	all logical addresses of a program	all physical addresses they map to



The program only works with logical addresses; the MMU translates them at run time.

05 Dynamic loading and dynamic linking

DYNAMIC LOADING



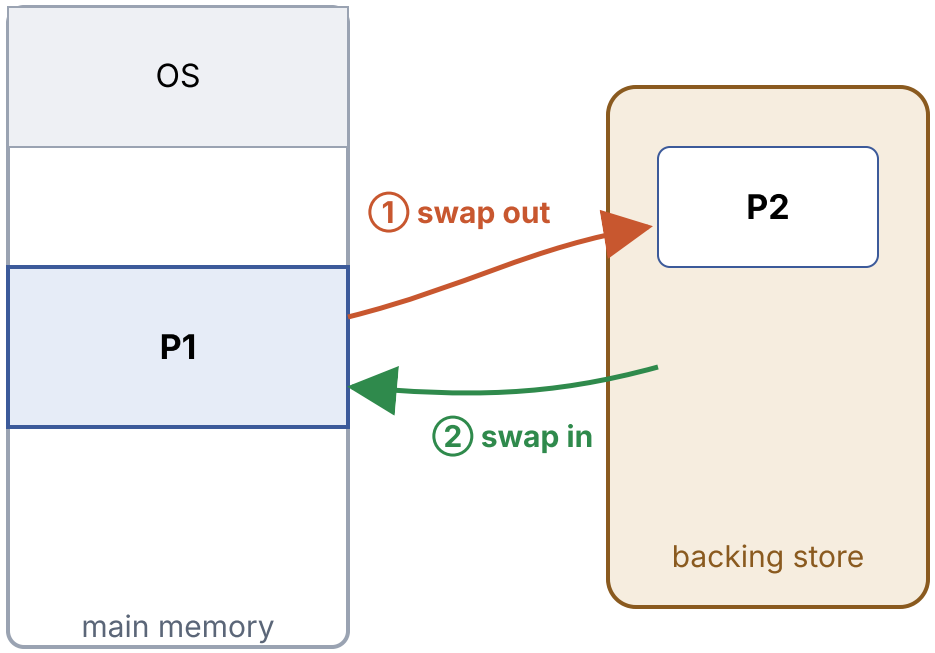
- A routine is loaded only when it is called
- Good for rarely used code (e.g. error handlers)

DYNAMIC LINKING

Static	libraries copied into the binary by the loader
Dynamic	linking postponed until execution

- A small **stub** finds the library routine and replaces itself with its address
- Shared libraries: one copy for all processes; versions may be needed

06 Swapping



- A process is moved temporarily to a **backing store** (fast disk) and brought back later
- Total memory of processes can exceed physical memory
- **Roll out, roll in**: a low-priority process makes room for a high-priority one
- Most of swap time is **transfer time** ∝ amount of memory swapped
- Linux, UNIX, Windows: swapping starts only above a memory threshold

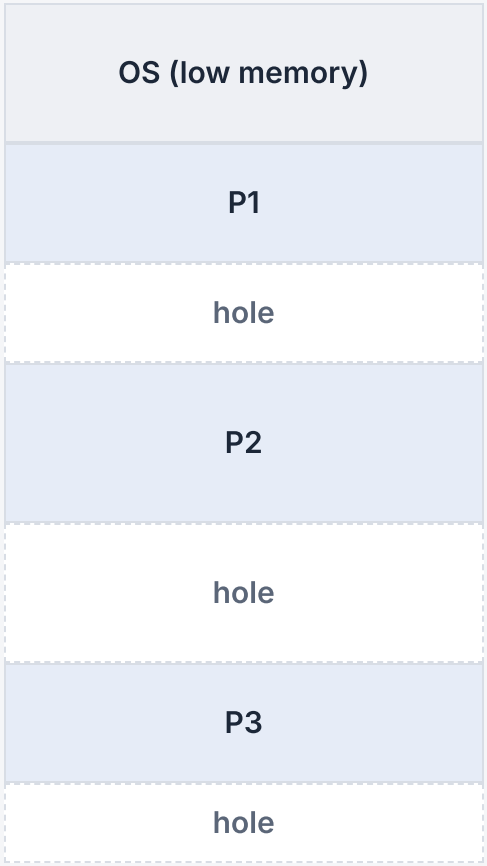
07 Swapping on mobile systems

Usually **not supported**: flash memory is small, wears out after a limited number of writes and is slow to transfer.

SYSTEM	WHAT IT DOES WHEN MEMORY IS LOW
iOS	Asks apps to give memory back voluntarily; read-only data is dropped and reloaded from flash; apps that do not free memory may be terminated
Android	Terminates apps, but first saves the application state to flash for a fast restart

> Both systems support **paging** instead.

08 Contiguous allocation and holes

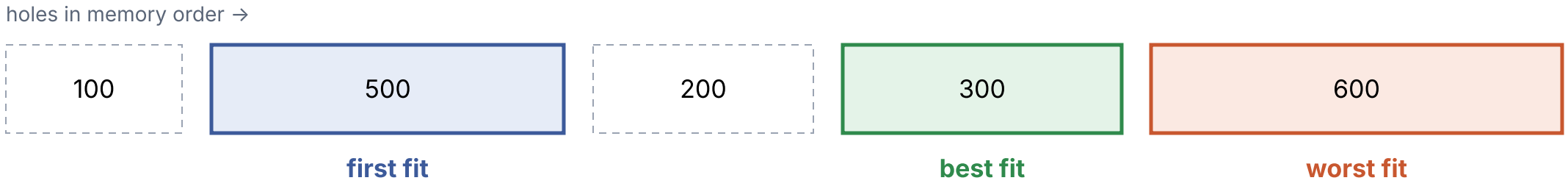


- Memory has two parts: the **resident OS** (usually low, with the interrupt vector) and **user processes**
- Each process occupies **one contiguous section**
- **Variable partitions**: the OS keeps a table of free and used blocks
- Free blocks are called **holes**; when a process ends, its block becomes a hole and is merged with adjacent holes

> Problem: how to satisfy a request of size n from the list of holes?

09 First fit, best fit, worst fit

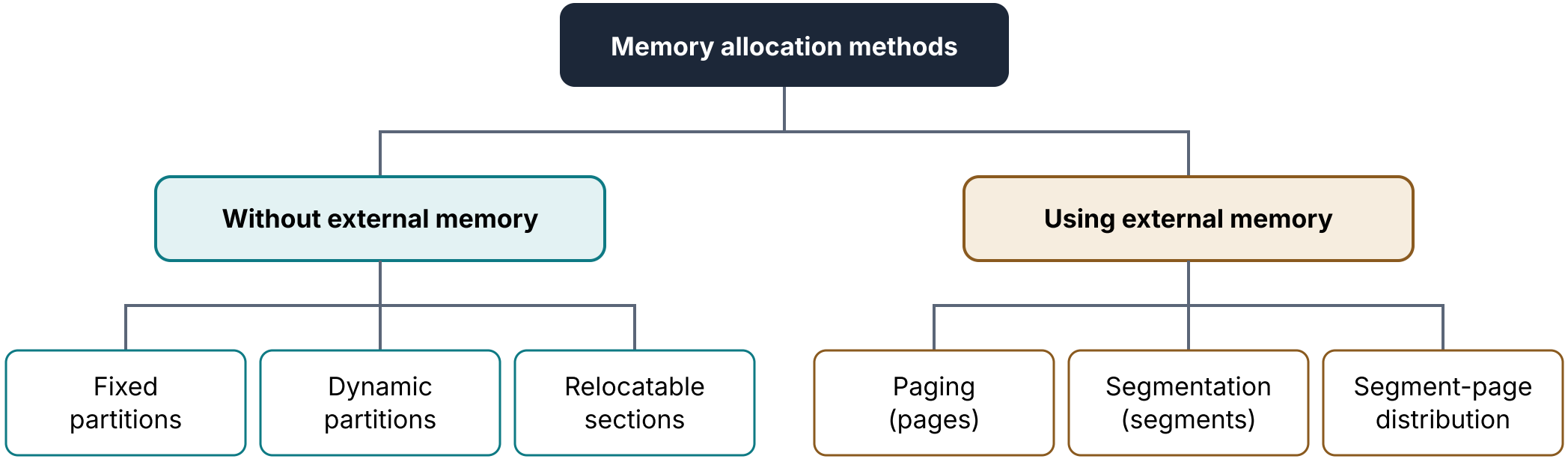
Example: holes of 100, 500, 200, 300, 600 KB. A process needs **212 KB**.



STRATEGY	CHOOSES	SEARCH	LEFTOVER HOLE
First fit	first hole that is big enough	stops early — fastest	500 − 212 = 288
Best fit	smallest hole that is big enough	whole list (unless sorted)	300 − 212 = 88 (smallest)
Worst fit	largest hole	whole list (unless sorted)	600 − 212 = 388 (largest)

10

Classification of memory allocation methods



11 Fragmentation

External



free 120 in total, but a request of 100 does not fit

Enough free memory exists in total, but it is **not contiguous**.

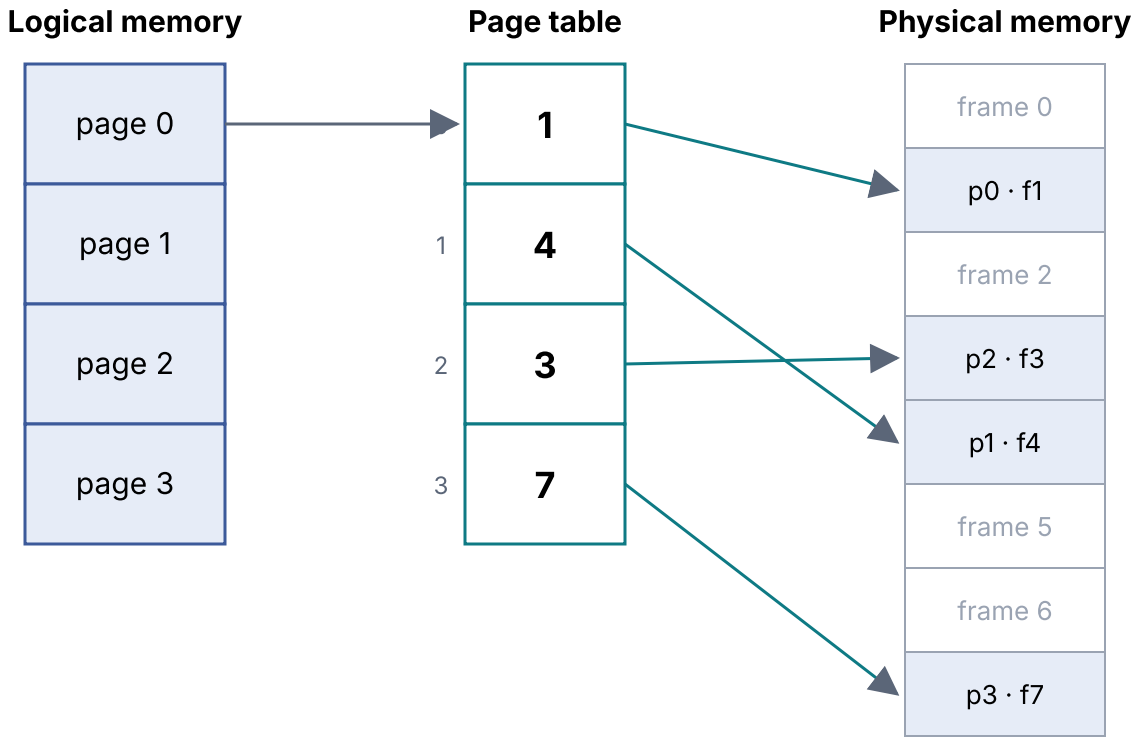
Internal



block of 20 KB, the last 2 KB are wasted

Allocated block is **slightly larger** than requested; the rest is unused inside the partition.

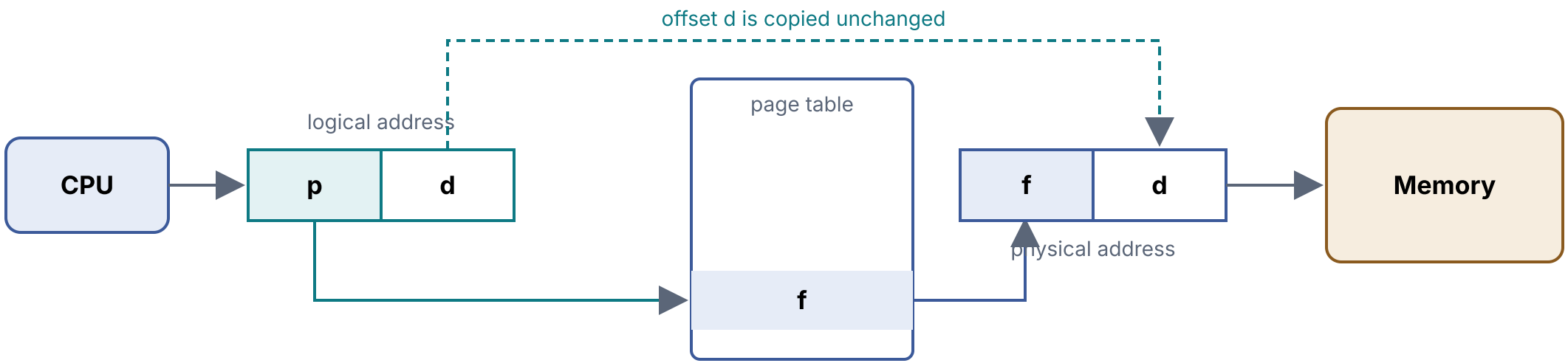
12 Paging



Page	fixed-size block of logical memory
Frame	block of physical memory of the same size
Page table	maps page number → frame number
Size	power of 2: 512 B … 16 MB

> A process does not need contiguous memory: pages go into **any free frames**. A program of N pages needs N free frames.

13 Address translation with paging



1

Take page number **p**, use it as an index into the page table

2

Read the frame number **f**

3

Replace p with f; offset **d** stays the same

Logical address space 2^m, page size 2^n: page number = high m − n bits, offset = low n bits.

14 Paging: advantages and disadvantages

Advantages

No external fragmentation	pages fit in any free frame
Efficient use	small free spaces are used
Virtual memory	more memory than physically present
Easy swapping	single pages move to disk
Isolation	a process sees only its pages

Disadvantages

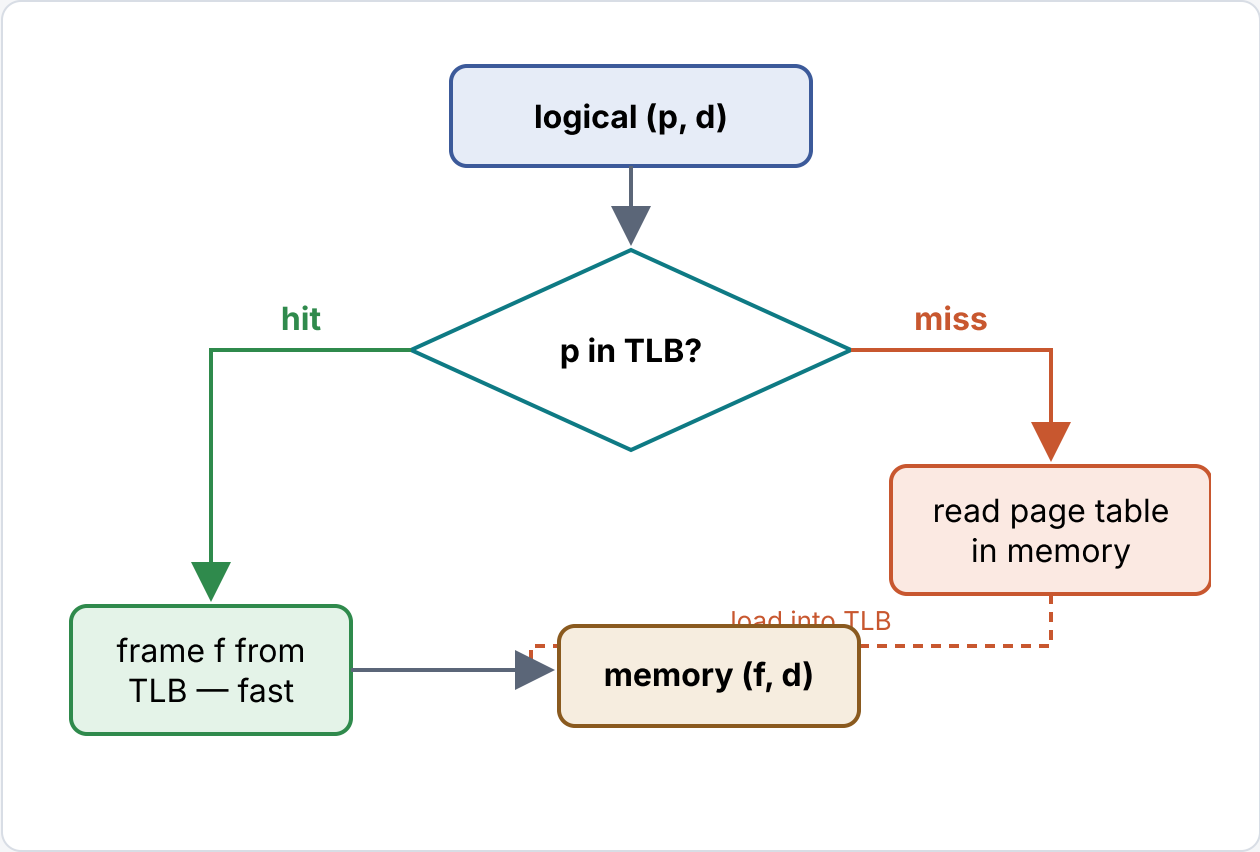
Internal fragmentation	unused part of the last page
Overhead	page tables take memory
Lookup time	extra translation step (TLB helps)
Page-fault I/O	missing pages come from disk
Complexity	needs MMU and replacement algorithms

15 Page table in memory and the TLB

PTBR	points to the page table
PTLR	size of the page table

> Without help, every access needs **two memory accesses**: page table + data.

- **TLB** — fast associative cache, 64–1024 entries, parallel search
- **ASID** in each entry identifies the process; otherwise flush the TLB at every context switch
- Some entries can be **wired down** permanently



16 Memory protection in paging

PAGE	FRAME	BIT	ACCESS
0	2	v	R
1	3	v	RW
2	4	v	RW
3	—	i	—
4	—	i	—

Protection bits per frame: read-only, read-write, execute-only

Valid (v): page belongs to the process's logical address space

Invalid (i): page is not part of the process

Any violation → **trap to the kernel**. A process cannot reach memory outside its page table.

17 Why page tables get large

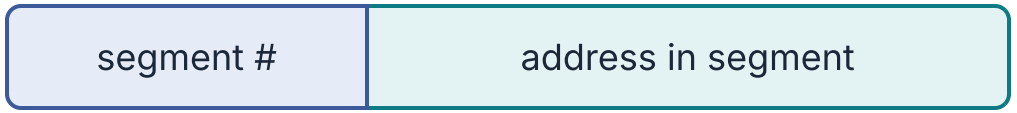


STRUCTURE	IDEA
Hierarchical	Page the page table itself, e.g. a two-level table
Hashed	Hash the virtual page number; each entry: page number, frame, pointer to next in the chain
Inverted	One entry per **physical frame** (virtual page + owning process); one table for the whole system

18 Segmentation and segment-paging

Segments, unlike pages, have **variable size**.

Segmented



two-dimensional virtual address

Segment-page



two-level translation to the physical address

19

Summary

Protection

base + limit registers, valid–invalid bits

Addresses

logical → MMU → physical; binding at compile, load or run time

Swapping

move processes to a backing store and back

Contiguous

holes; first / best / worst fit; fragmentation

Paging

pages → frames via page table; TLB speeds it up

Large tables

hierarchical, hashed, inverted page tables