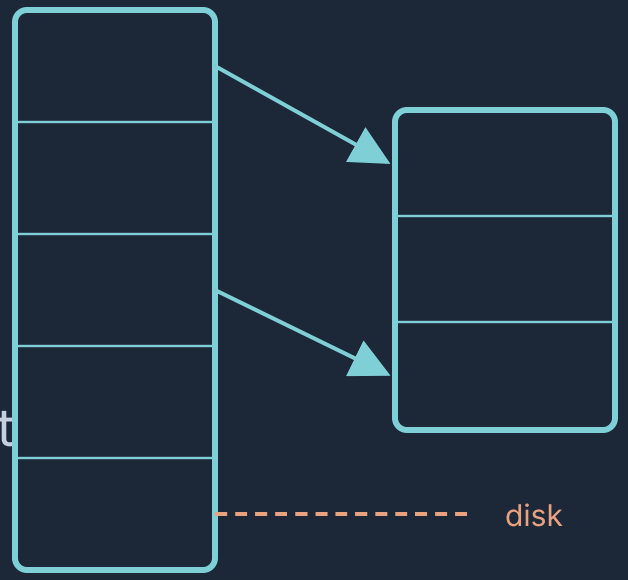
OPERATING SYSTEMS · LECTURE 9

Virtual Memory

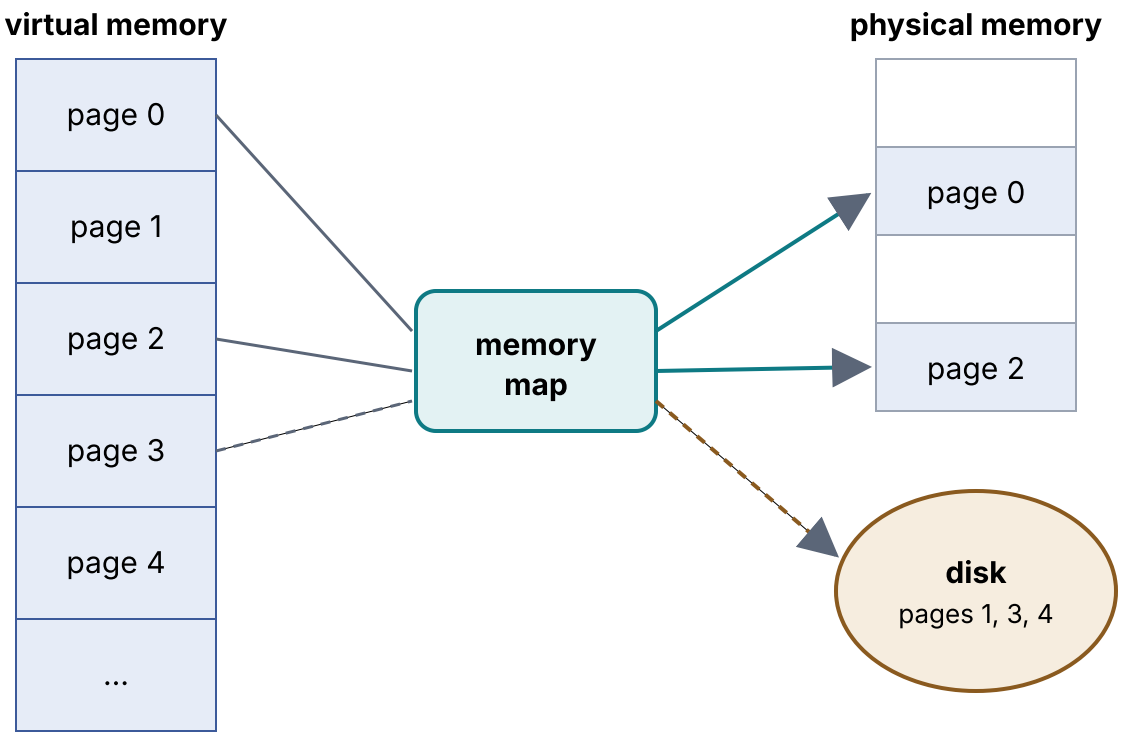
Demand paging, page faults, copy-on-write, page replacement



Lecturer: Aidana Karibayeva

01 What is virtual memory?

A process can run when it is **not completely in memory**. Programs can be larger than physical memory.



Bigger programs
logical space ≫ physical

Only part in RAM
the rest stays on disk

Sharing
address spaces shared by processes

Fast creation
processes start quicker

More programs
run at the same time

Less I/O
to load or swap

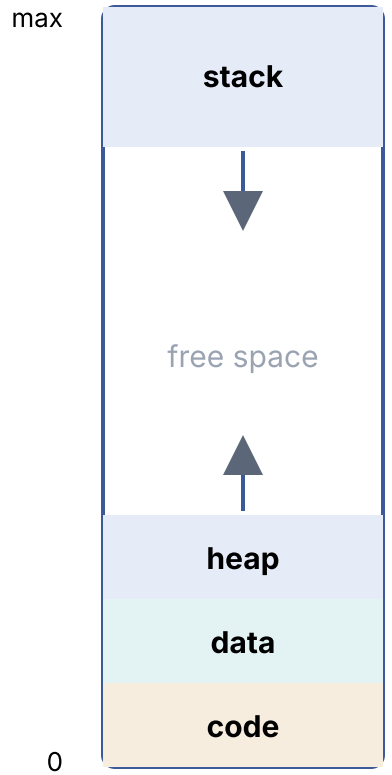
02 Virtual address space

- Logical view of how a process is stored: starts at address **0**, contiguous to the end
- Physical memory is organised in **page frames**
- The **MMU** maps logical addresses to physical ones

Implemented by

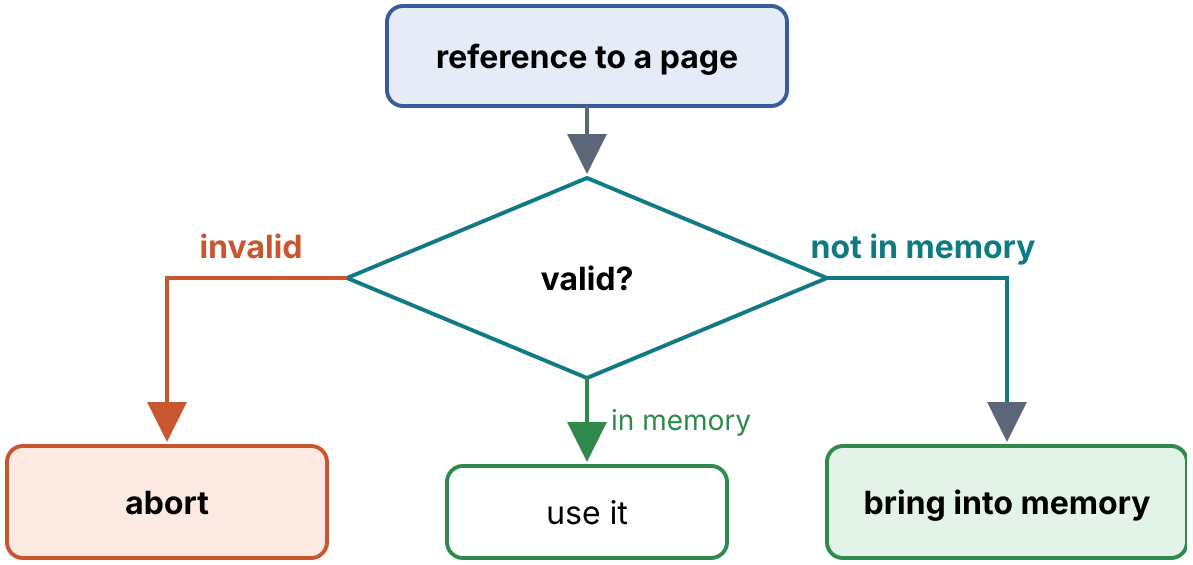
Demand paging

Demand segmentation



03 Demand paging

Load a page **only when it is needed**. Pages that are never used are never loaded.



Lazy swapper	never brings a page in unless it is needed
Pager	a swapper that works with pages

- less I/O
- less memory
- faster response
- more users

04 Hardware support: valid–invalid bit

PAGE	FRAME	BIT
0	4	v
1	—	i
2	6	v
3	—	i
4	—	i
5	9	v

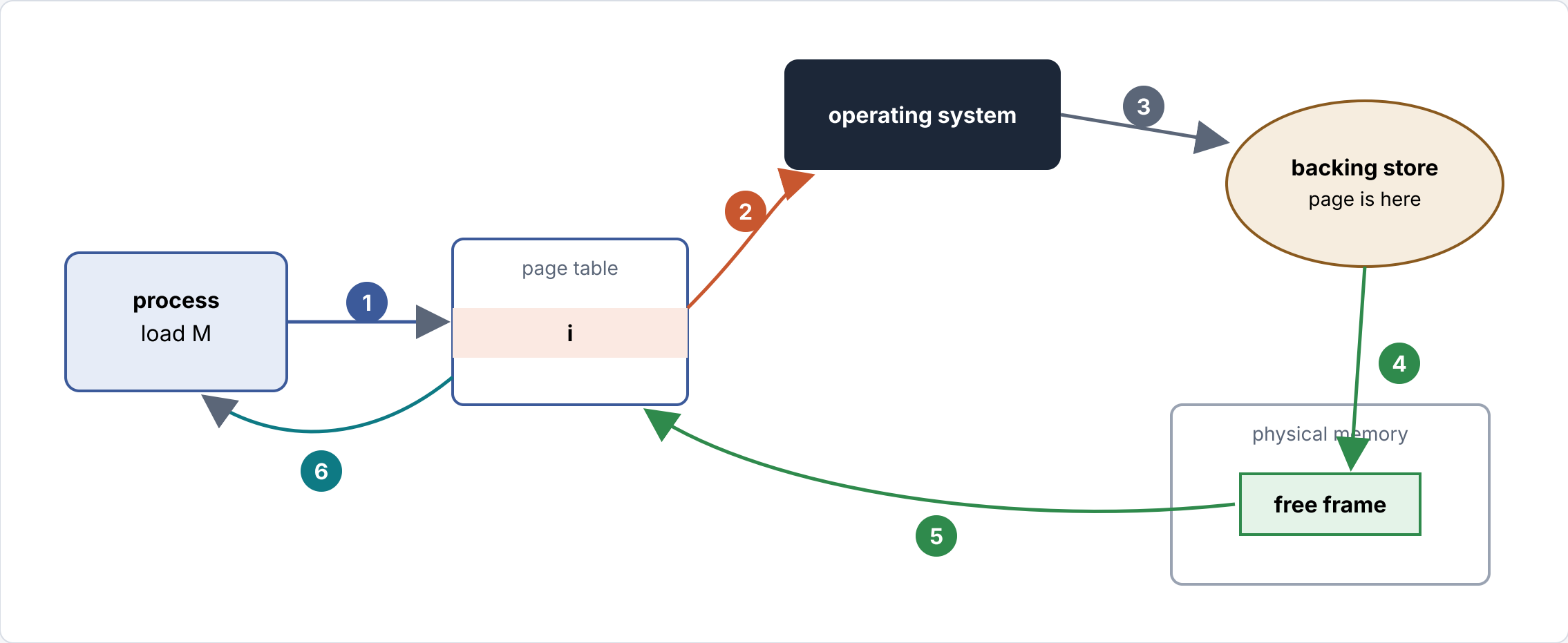
v page is in memory (memory resident)

i page is not in memory — at start **all** entries are i

Swap space on a fast disk / NVM holds the pages that are not in memory

MMU finds bit = i during translation → **page fault**

05 Handling a page fault

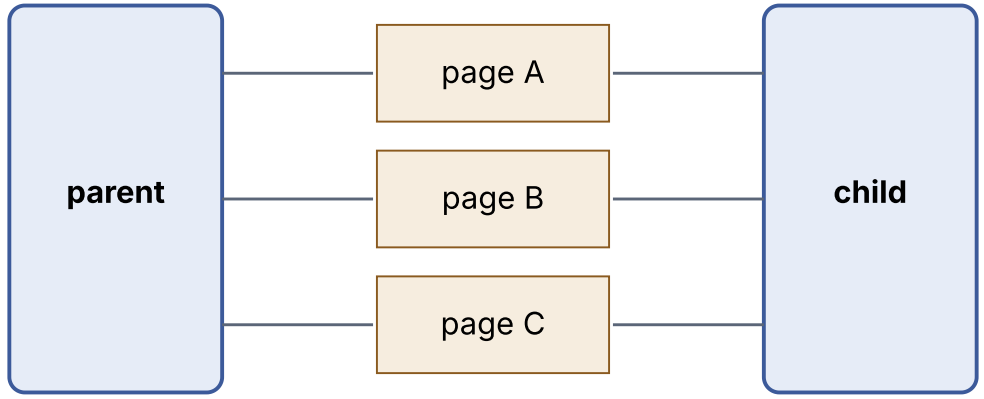


1 reference **2** trap: page fault **3** find page on disk **4** load into free frame **5** reset table: bit = v **6** restart instruction

06

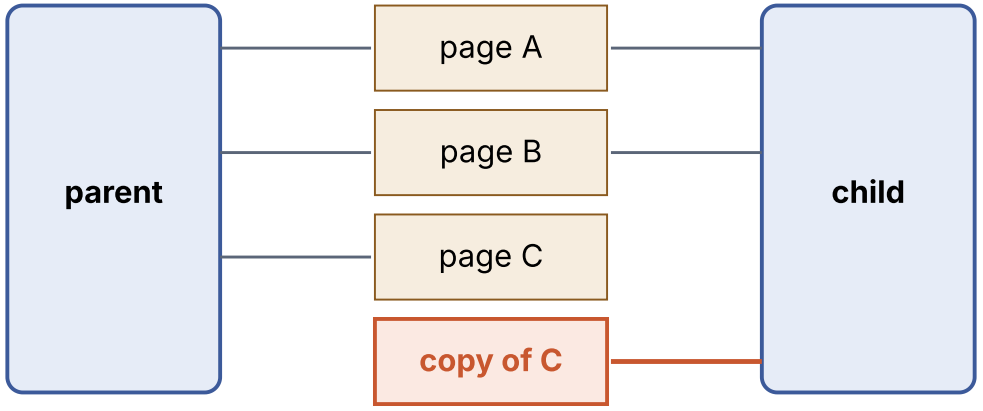
Copy-on-write (COW)

before a write



Parent and child **share** the same pages.

after the child writes to page C



Only the **modified page** is copied.

Faster process creation (`fork`)

Free pages come from a **zero-fill-on-demand** pool

`vfork()`: parent waits, child uses parent's space, then calls `exec()`

07 Basic page replacement

When there is **no free frame**, the OS chooses a **victim** page to replace.

1. Find the page on disk
2. Free frame? use it. No → pick a **victim**; write it to disk if **dirty**
3. Load the page; update page and frame tables
4. Restart the instruction

A page fault may now need **two** page transfers (victim out + page in), which increases the effective access time.

08 Frame allocation and page replacement

Frame-allocation algorithm

how many frames each process gets, and which frames to replace

Page-replacement algorithm

goal: the lowest page-fault rate

Evaluated on a reference string (page numbers only):

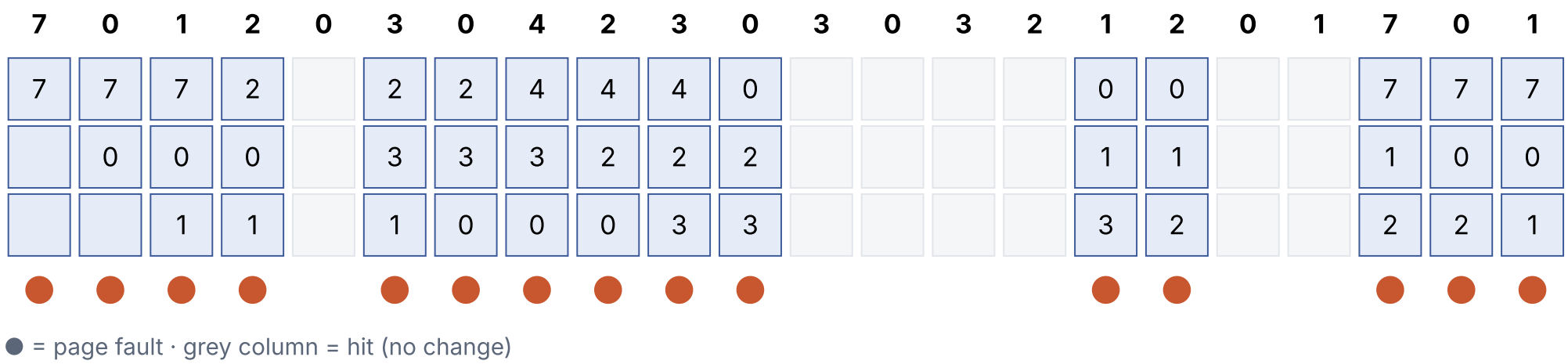
`7` `0` `1` `2` `0` `3` `0` `4` `2` `3` `0` `3` `0` `3` `2` `1` `2` `0` `1` `7` `0` `1`

- Repeated access to a page already in memory → **no** page fault
- Result depends on the number of available frames

09

Example: FIFO replacement, 3 frames

FIFO replaces the page that came into memory first.



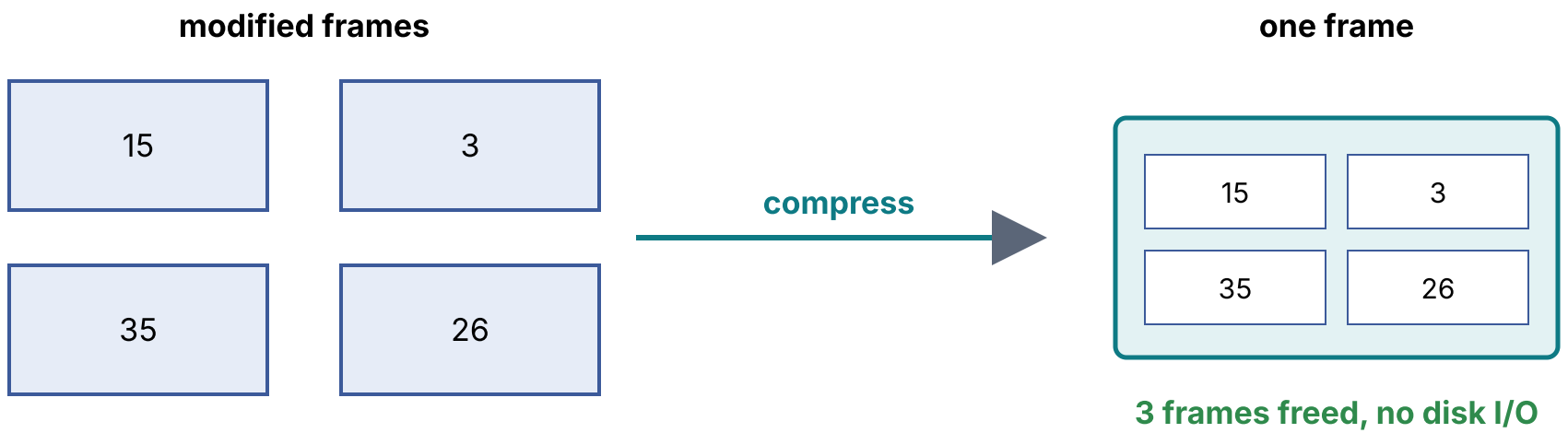
page faults	hits
15	7

10 Global vs local replacement

	GLOBAL	LOCAL
Victim chosen from	all frames — may take a frame from another process	only the process's own frames
Plus	higher throughput (more common)	stable per-process performance
Minus	execution time can vary a lot	memory may be underused

11 Memory compression

Instead of paging modified frames out to swap space, the OS **compresses several frames into one**.



12 Virtual memory in Linux and Windows

OS	APPROACH
Linux	**Demand paging**, pages taken from a list of free frames; kernel memory managed by **slab allocation**
Windows 10	**Demand paging with clustering**: on a fault, also loads nearby pages (locality of reference)

Windows cluster size

Data page: 3 pages



Other pages: 7 pages



13

Summary

Virtual memory

run processes not fully in RAM

Demand paging

load pages only when referenced

Page fault

bit = i → OS loads the page and restarts

Copy-on-write

share pages, copy only on write

Replacement

pick a victim; FIFO, global vs local

In practice

compression, Linux slabs, Windows clustering