CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

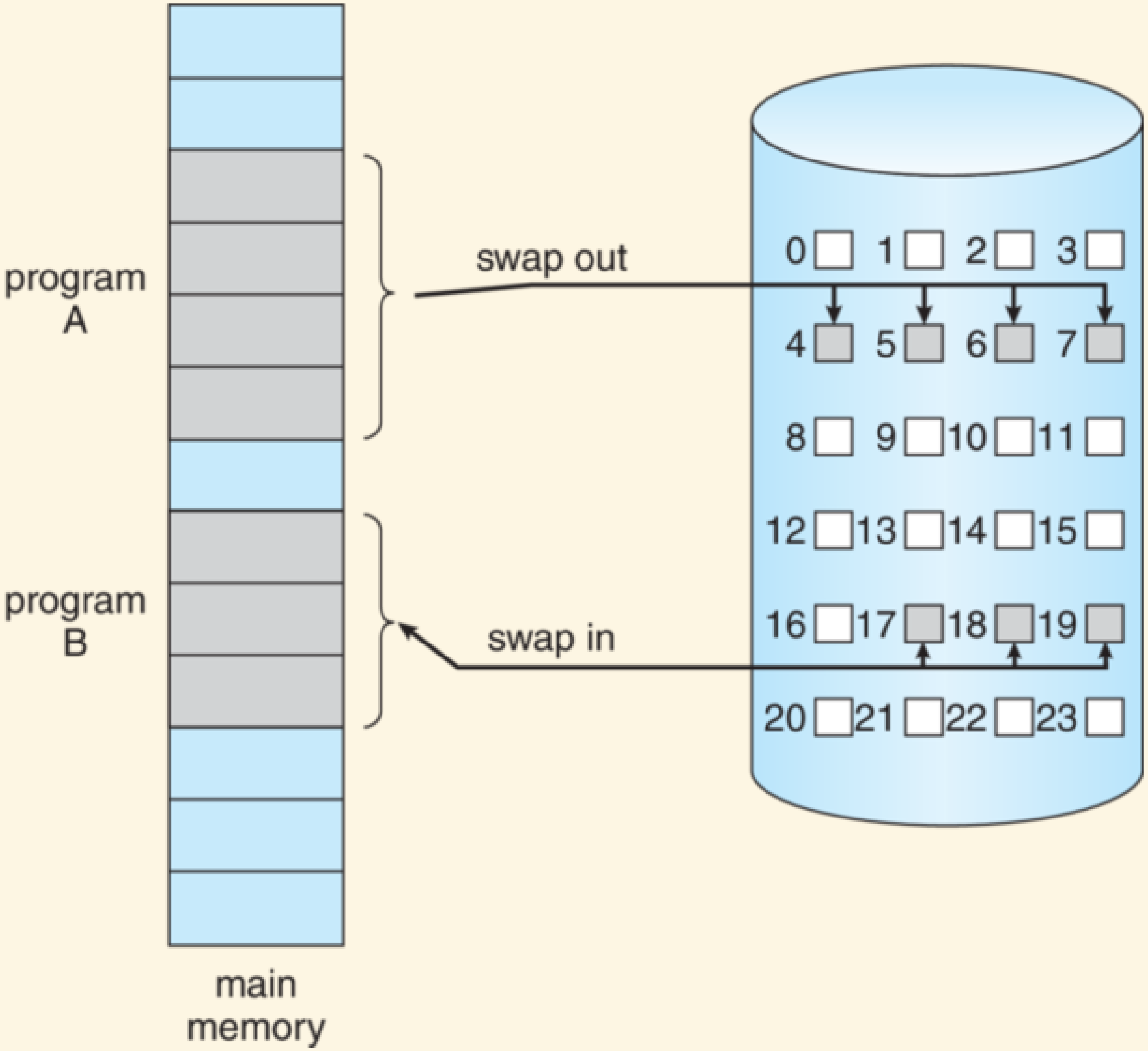
Demand Paging

In virtual memory, logical memory can be larger than physical memory

This can be acheived with **demand paging**

Demand paging -- load each page into memory only when the page is needed

Similar to swapping, but only a part of the process instead of the whole thing


program A
program B
swap out
swap in
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
main memory

A new issue arises from this: if not all pages are in memory, how do we know whether the page we want to access is currently in memory?

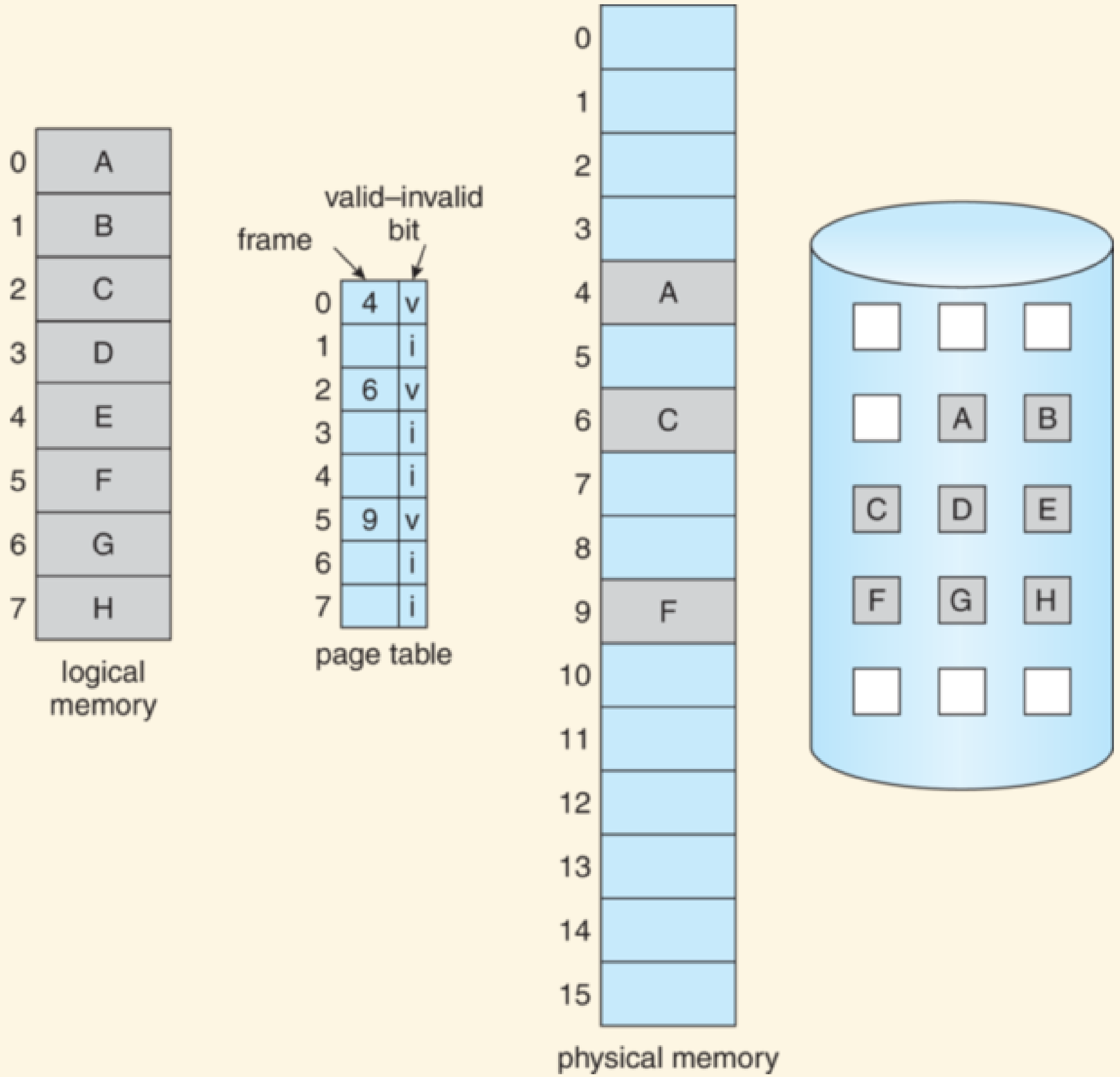
What happens when a page not in memory is requested?

Add **valid bit** to each entry of page table, just as we do with caches

If valid bit is set, page table entry holds frame number that

- belongs to this process, and
- has correct page loaded

Such a page is **memory resident**


0 A
1 B
2 C
3 D
4 E
5 F
6 G
7 H
logical memory
valid–invalid bit
frame
0 4 v
1 i
2 6 v
3 i
4 i
5 9 v
6 i
7 i
page table
0
1
2
3
4 A
5
6 C
7
8
9 F
10
11
12
13
14
15
physical memory
A
B
C
D
E
F
G
H

If valid bit not set, hardware issues a **page fault**

Page fault -- hardware interrupt triggered when page requested by CPU does not currently reside in memory

It is job of OS to keep page table up to date, so system must trap to OS when page fault occurs

Handling page fault

1. OS must check whether the access attempt was valid

OS is still responsible for keeping processes within their memory space

If access is invalid, process will be terminated

2. Otherwise, begin "paging in" page from disk to memory

Handling page fault

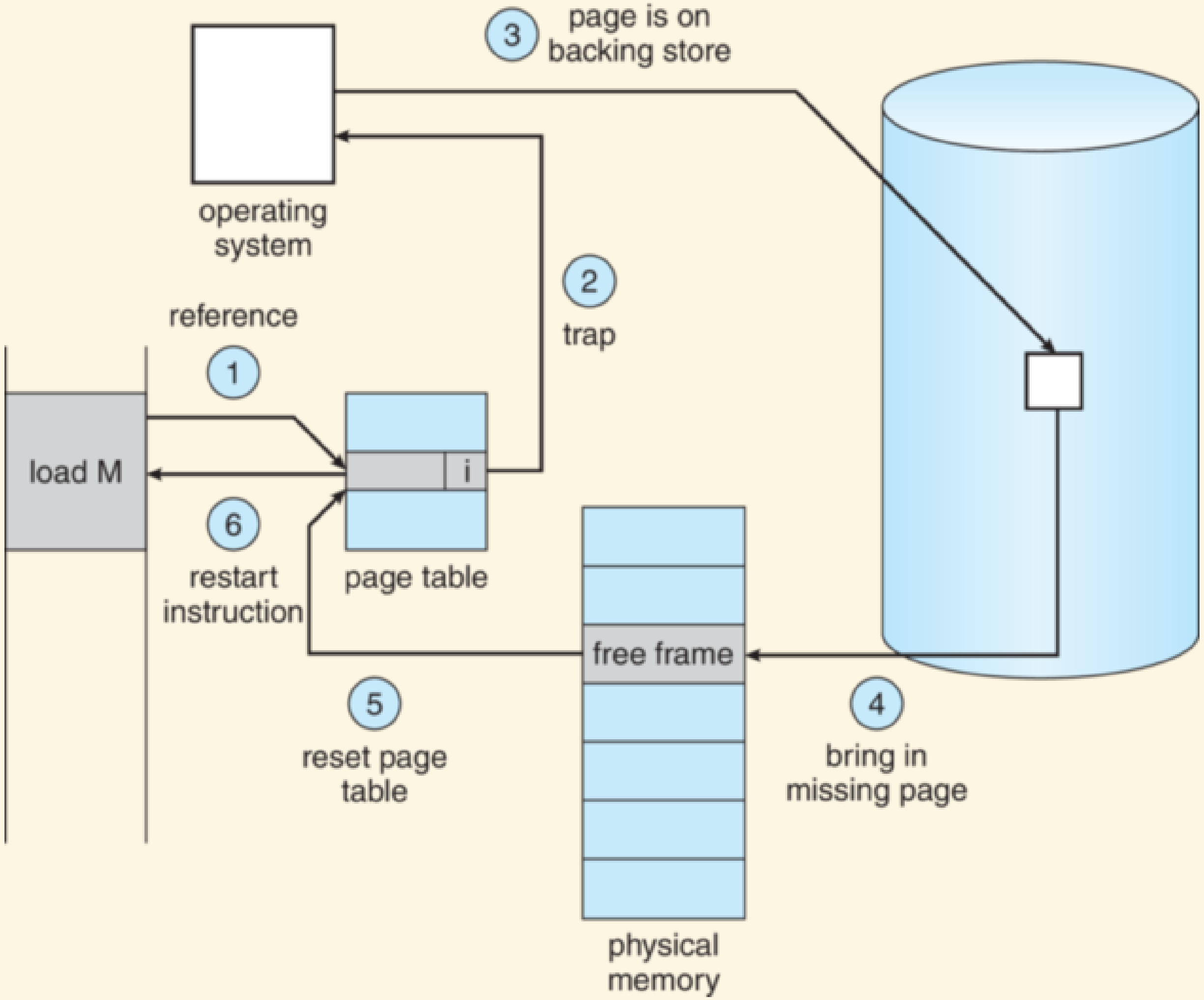
3. OS must find a free frame for page to be loaded into
4. OS requests I/O to read desired page from disk into selected frame
5. Page table is updated with new frame number (and valid bit)

Handling page fault

6. Finally, instruction that caused page fault must be restarted

Remember -- all of this should be transparent to process

As far as requesting process knows, it made a memory access, completed the relevant instruction, and moved on to next instruction as it normally would


3 page is on backing store
operating system
2 trap
reference
1
load M
i
6
restart instruction
page table
free frame
5
reset page table
4
bring in missing page
physical memory

How to know what goes into memory in the first place?

Will see better ways later, but simplest is **pure demand paging**

Processes start with nothing in memory and pages are added only on page faults