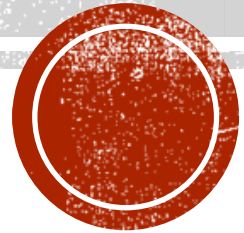
PROJECT 2: DESIGNING A VIRTUAL MEMORY MANAGER



Some slides are adopted from PPTs offered with textbook <Operating Systems Concepts>, 10th Edition, Abraham Silberschatz, Peter B. Galvin, Greg Gagne , WILEY.

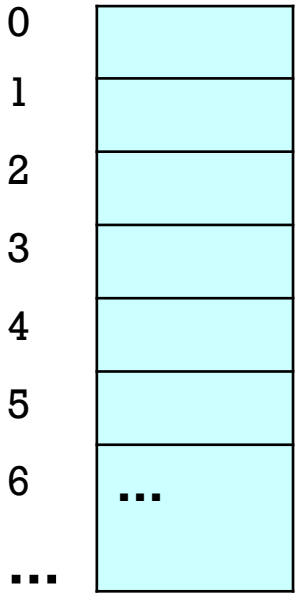
PROJECT: THE SYSTEM

page size 256

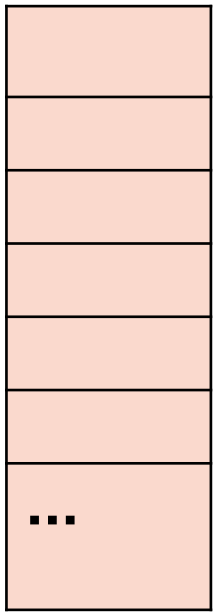
256 pages/frames, page size 256 B [8 bits, 8bits]
Total size: 256 x 256 B = 2^16 B = 64KB (i.e. 65,536B)

The backing store, each address stored one byte data:
256 * 256 = 65536 bytes in size, also 256 * 256 addresses

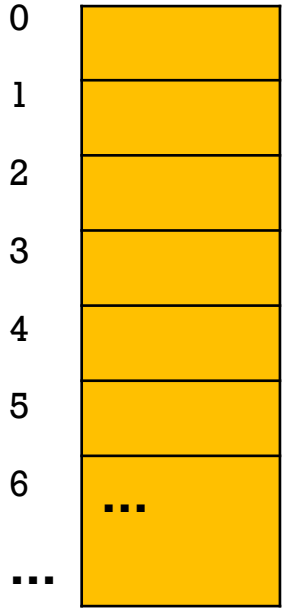
Virtual address space, 256 pages 16-bit address space

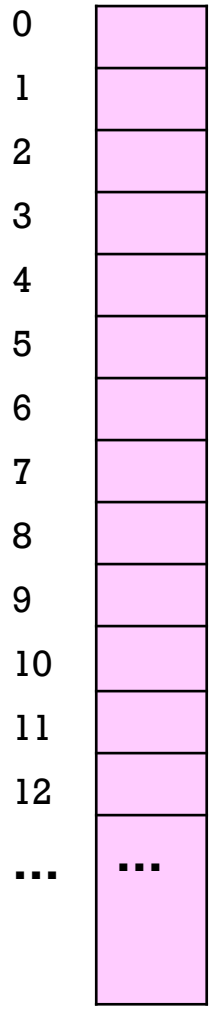


page table 256 entry



Physical address space, 256 frames 16-bit address space, or less





TLB 16 entry

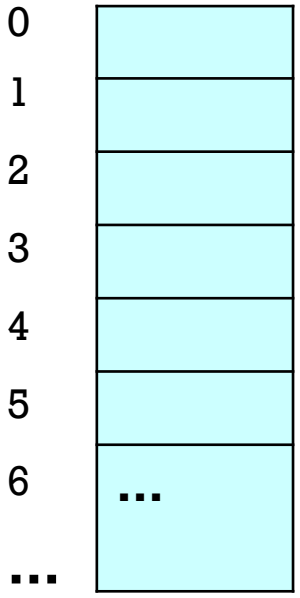
Page #	Frame #

- Addresses: the inputs are too long (32-bits), so mask off the first 16bits, **use the later 16 bits as real virtual address**

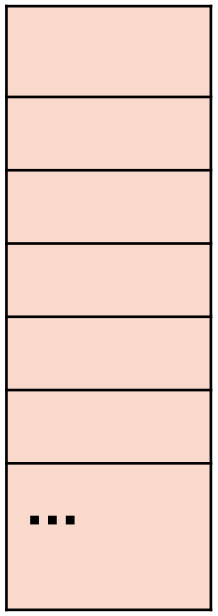
PROJECT: ADDRESS TRANSLATION, PAGE FAULT, LOAD CONTENTS

E.g., a reference list: 2 1 5 6 1 2 0 …

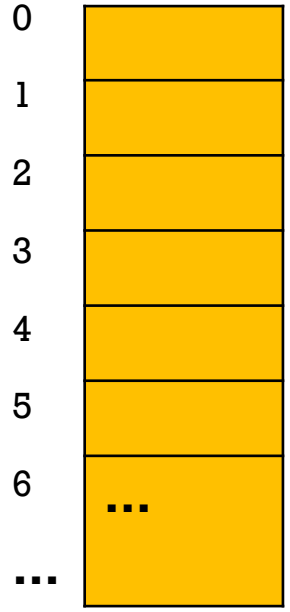
Virtual address space, 256 pages
16-bit address space



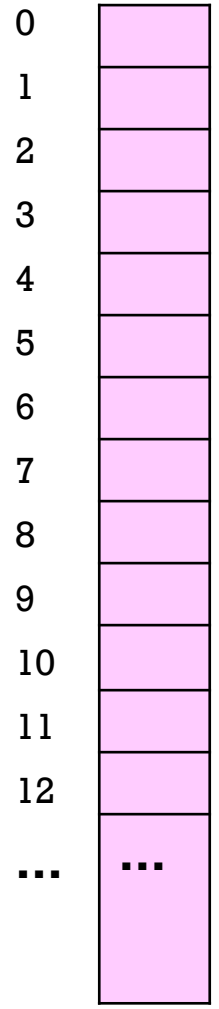
page table 256 entry



Physical address space, 256 frames
16-bit address space, or less



Free frame list



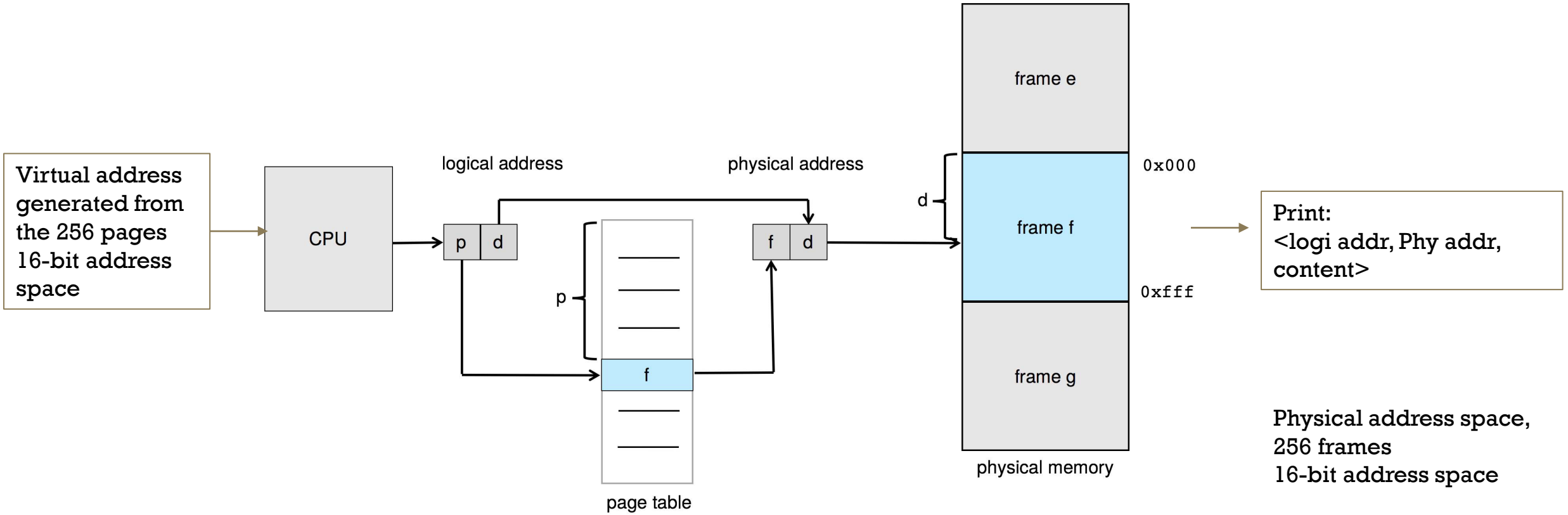
- Pure demand paging!
  - Starting with the first page: Page fault
- Update page table, TLB entry, if needed

Page #	Frame #

TLB 16 entry

Read one page a time (using logical address) store it to a frame:
File I/O uses random access.

PROJECT: STEPS OF PAGING (1) WITHOUT TLB FIRST

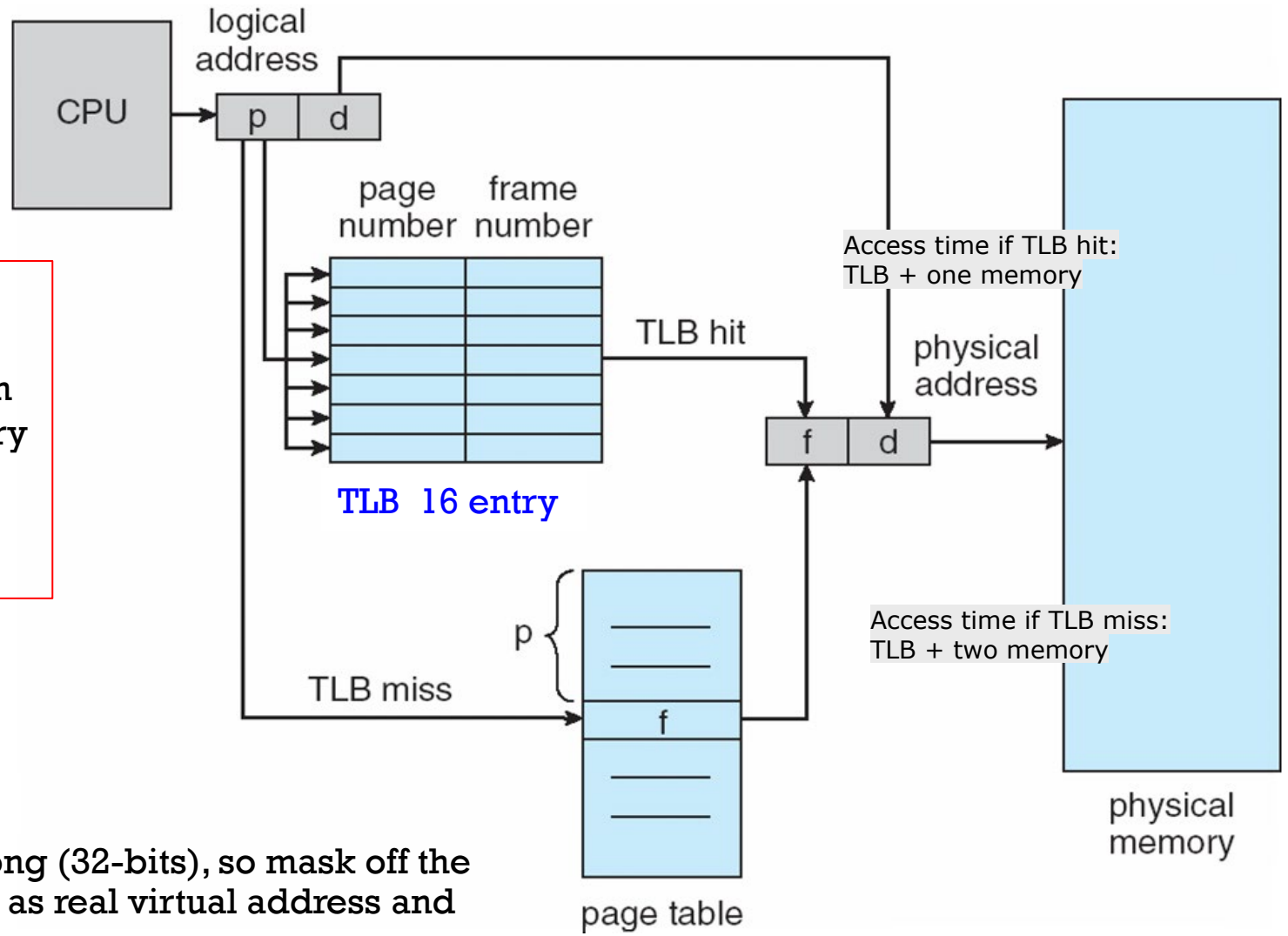


Implement and test with only paging first!
Add TLB only when paging part is correct.

PROJECT: STEPS OF PAGING WITH TLB (2)

Address translation (p, d):
If p is in TLB, get frame #;
Otherwise, get frame # from page table in memory (a TLB miss); load the page entry to TLB

TLB replacement using FIFO



- Addresses: the inputs are too long (32-bits), so mask off the first 16bits, use the later 16 bits as real virtual address and physical

PROJECT: STEPS OF PURE DEMANDING PAGING WITH TLB - SUMMARY

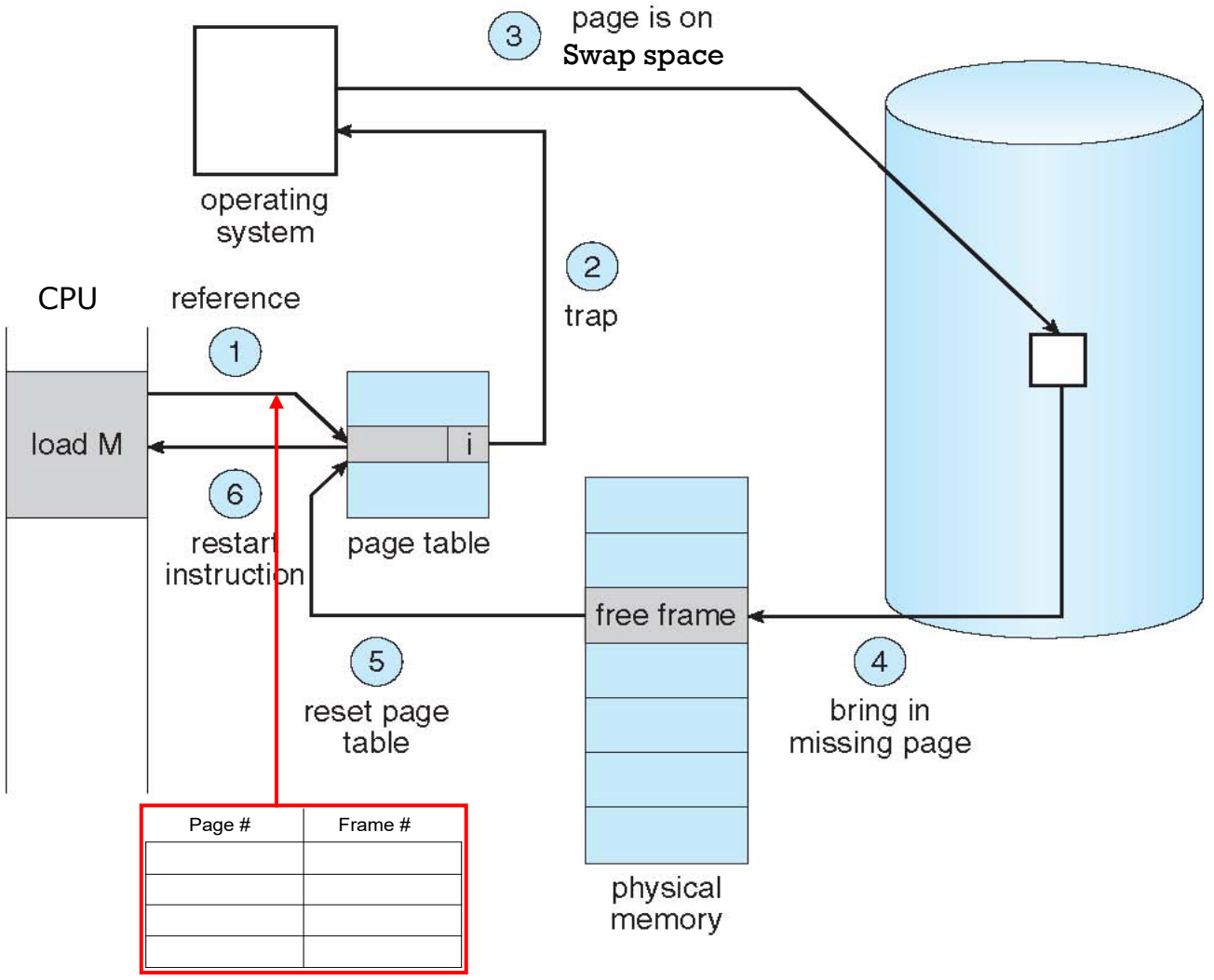
- Pure demand paging!
  - Starting with the first page: Page fault

Read a page (256 bytes) and store in physical memory (update page table, TLB entry, if needed)

- Always has free physical frames in this part

File I/O uses random access.

Address translation (p, d):
If p is in TLB, get frame #;
Otherwise, get frame # from page table in memory (a TLB miss)
load the page entry to TLB
TLB replacement using FIFO



PROJECT: TESTING

- Memory references: read in from an addresses.txt

Reference string

- Address translation and access. Possibly handle two cases:
  - TLB miss
  - Page fault
- Reading in the page from backing store
- Output in one line

<logical address, phy address, content byte>

<integer, integer, signed byte>

Add statistics:

- TLB hit rate
- Page fault rate

Testing cases:

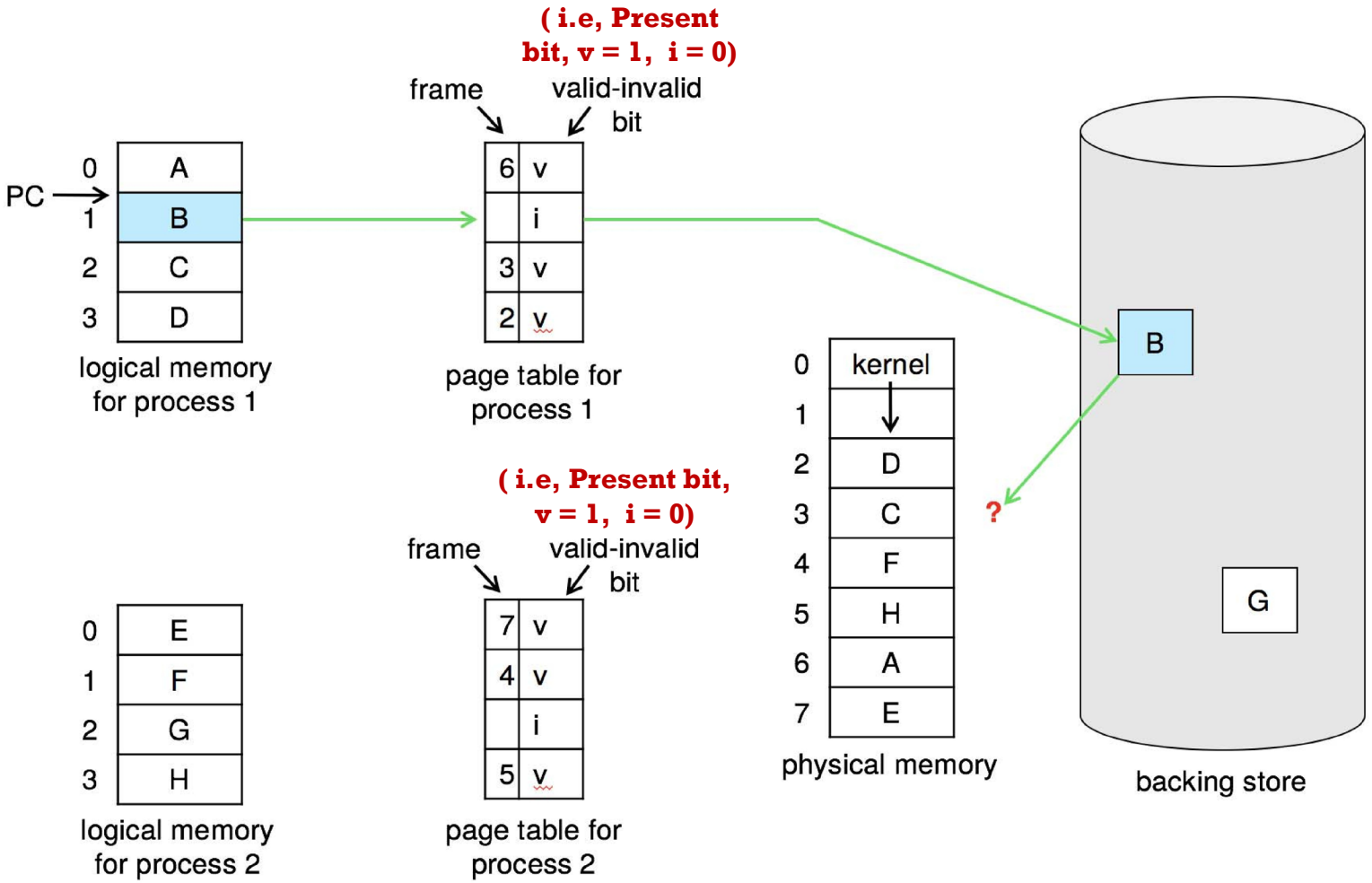
(1) Give a couple logical addresses, manually work out the physical addresses. See whether your program output the same

(2) You can create your own small test file manually.
- e.g., 16 or 32 reference addresses.

(3) One large test file and answer is provided.

PROJECT: MODIFICATION - ADDING PAGE REPLACEMENT

- Reduce the physical memory to 128 frames (reduced half)
- Need to add the ability to handle free frames
  - Handling Page Fault with Page Replacement
- Page replacement using LRU
- Memory references: read in from an addresses.txt
- Address translation and access. Possibly handle three cases:
  - TLB miss
  - Page fault
  - Page fault and replacement
- Reading in the page from backing store
- Output in one line



HANDLING PAGE FAULT WITH PAGE REPLACEMENT

1. If there is a reference to a page, first reference to that page will trap to operating system (0 in present bit)
   - Page fault
2. Operating system looks at page table to decide present or not
3. // find a free frame

   If there is a free frame, use it
   If there is no free frame, use a page replacement algorithm to select a **victim frame;** write the victim frame to disk if dirty
4. Swap the page in disk into the (newly) free frame via scheduled disk operation
   (the disk address of this page is known to OS)
5. Reset tables to indicate the page now in memory
   (set frame # and present bit = **1**)
6. Restart the instruction that caused the page fault

- Note 3: now potentially 2 page disk transfers for a page fault:

  increasing Effective Access Time