THE UNIVERSITY *of* EDINBURGH
informatics

Operating Systems (INFR10079)

2022/2023 Semester 2

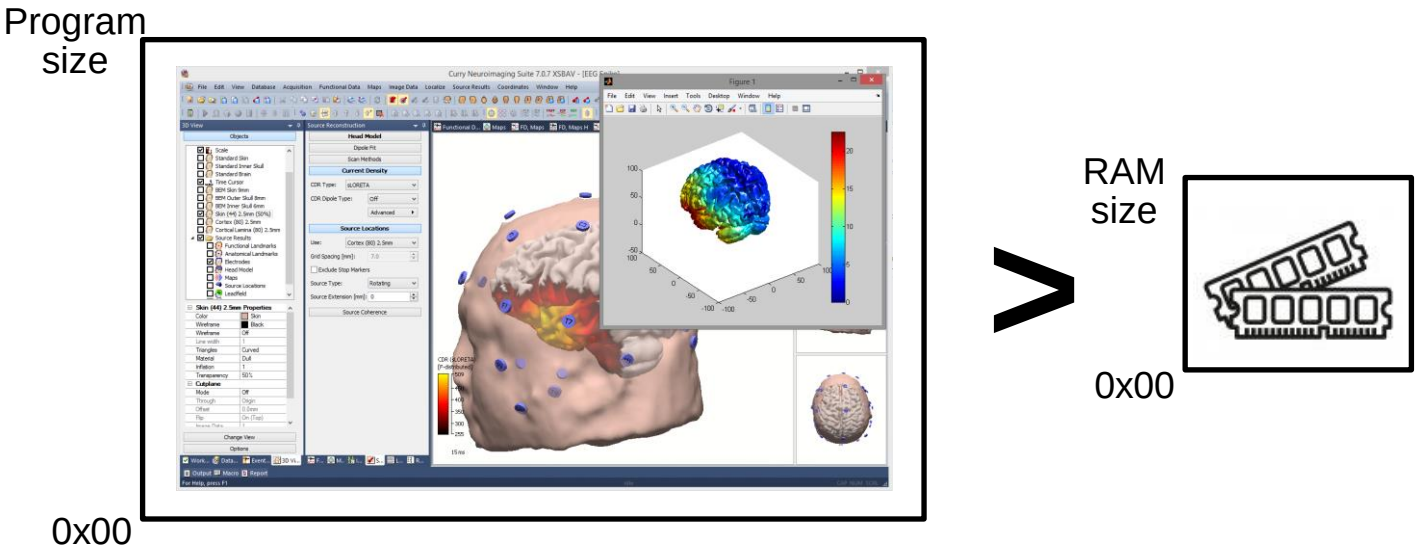
Virtual Memory
(Basics)

abarbala@inf.ed.ac.uk

Chapter 10

Large Programs

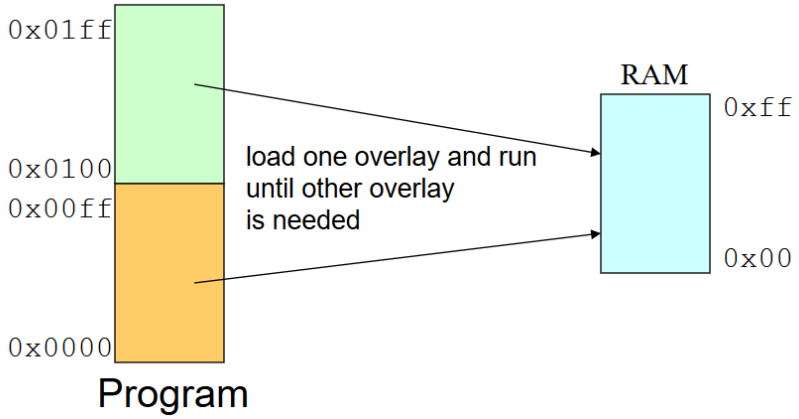
- How to deal with a program that is **bigger than** the available physical memory?



- Could a program **correctly execute** even if **it is not all in memory at all times**?

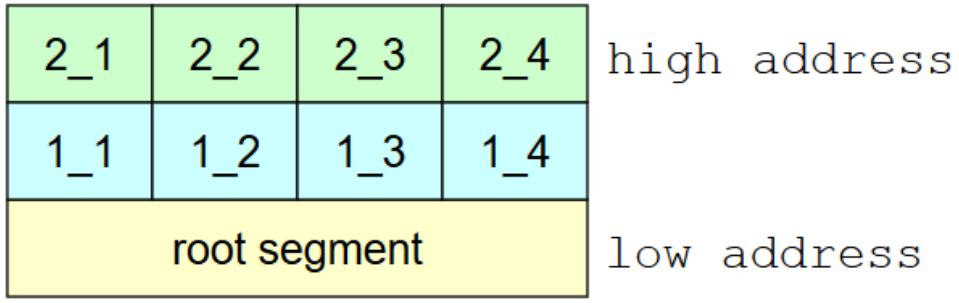
Overlays #1

- Only load **part** of the program at any time
- Programmer breaks **address space into pieces** that fit into memory
  - Constrained by **physical memory size**
- Pieces called **overlays**, are loaded and unloaded by the program



Overlays #2

- **Overlay manager** (part of the program, not the OS)
  - Loads an overlay when it is not in RAM
  - Eventually unloads an overlay previously in RAM
- Overlays **Mechanism**
  - One root segment (always in RAM)
    - Includes overlay manager
  - 2 or more memory areas
  - Within each area any number of overlay segments
  - Only 1 overlay segment can be in a partition at a given time

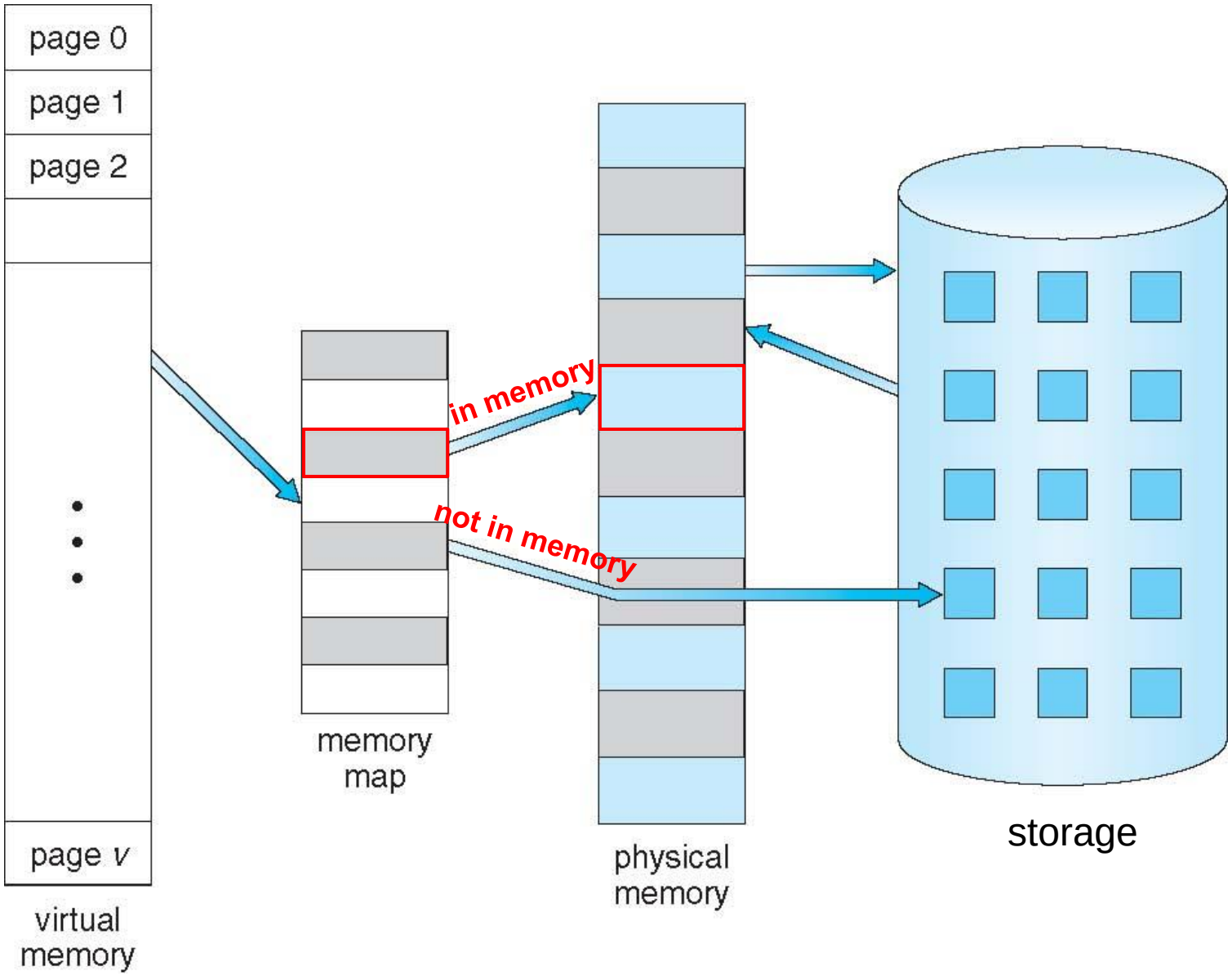


- segments 1_1 .. 1_4 share the same memory area
- so do segments 2_1 .. 2_4

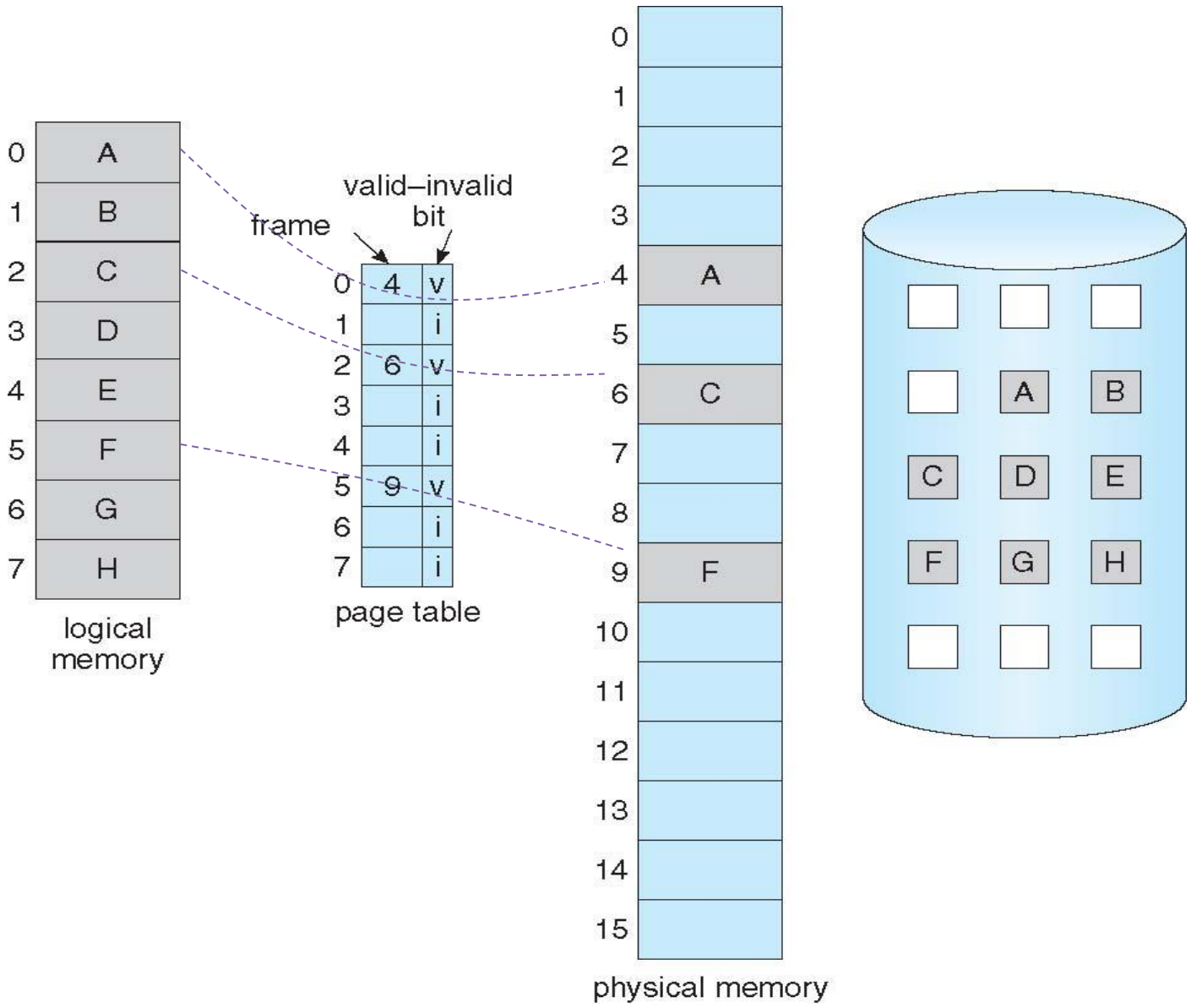
Virtual Memory

- Fully **decouples** address space from physical memory
- Allows a **larger logical address space** than physical memory
- **Paged Virtual Memory**
  - Based on hardware, with operating system support
  - Transparent to programmer, no programmer involvement
- All pages of address space **do not need** to be in memory
  - The full address space **on disk**
    - page-sized blocks
  - Main memory used as a **cache**
- Needed pages transferred to a free page frame in memory
  - If no free page frames available, find one to evict

Virtual Memory Larger Than Physical Memory



Page Table When Some Pages Are Not in Main Memory

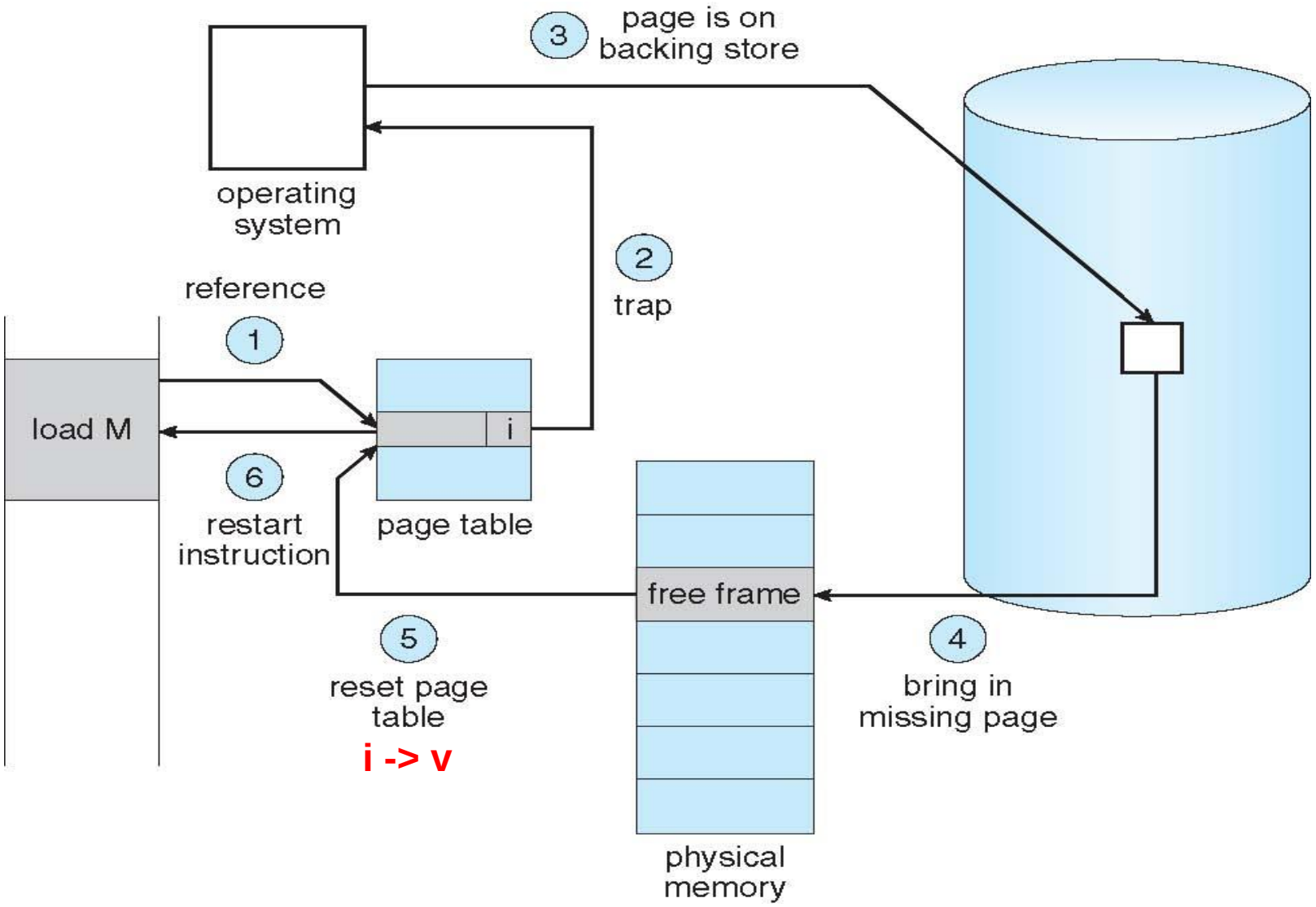


Page Fault

1. Software accesses a page that is not in memory
   - Before the access, the relative page table entry is invalid
2. Hardware triggers a **page fault** (exception)
3. Operating System checks internal data structures
   - Invalid reference, abort the original software
   - Not in memory, continue
4. Operating System finds a free frame
   - Swaps page into frame via scheduled disk operation
5. Operating System set internal data structures to indicate page now in memory
   - Set valid bit
6. Operating System restarts the instruction that caused the **page fault**

(see next slides)

Steps in Handling a Page Fault



Valid pages are accessed directly by the hardware ***without OS involvement***

Demand Paging

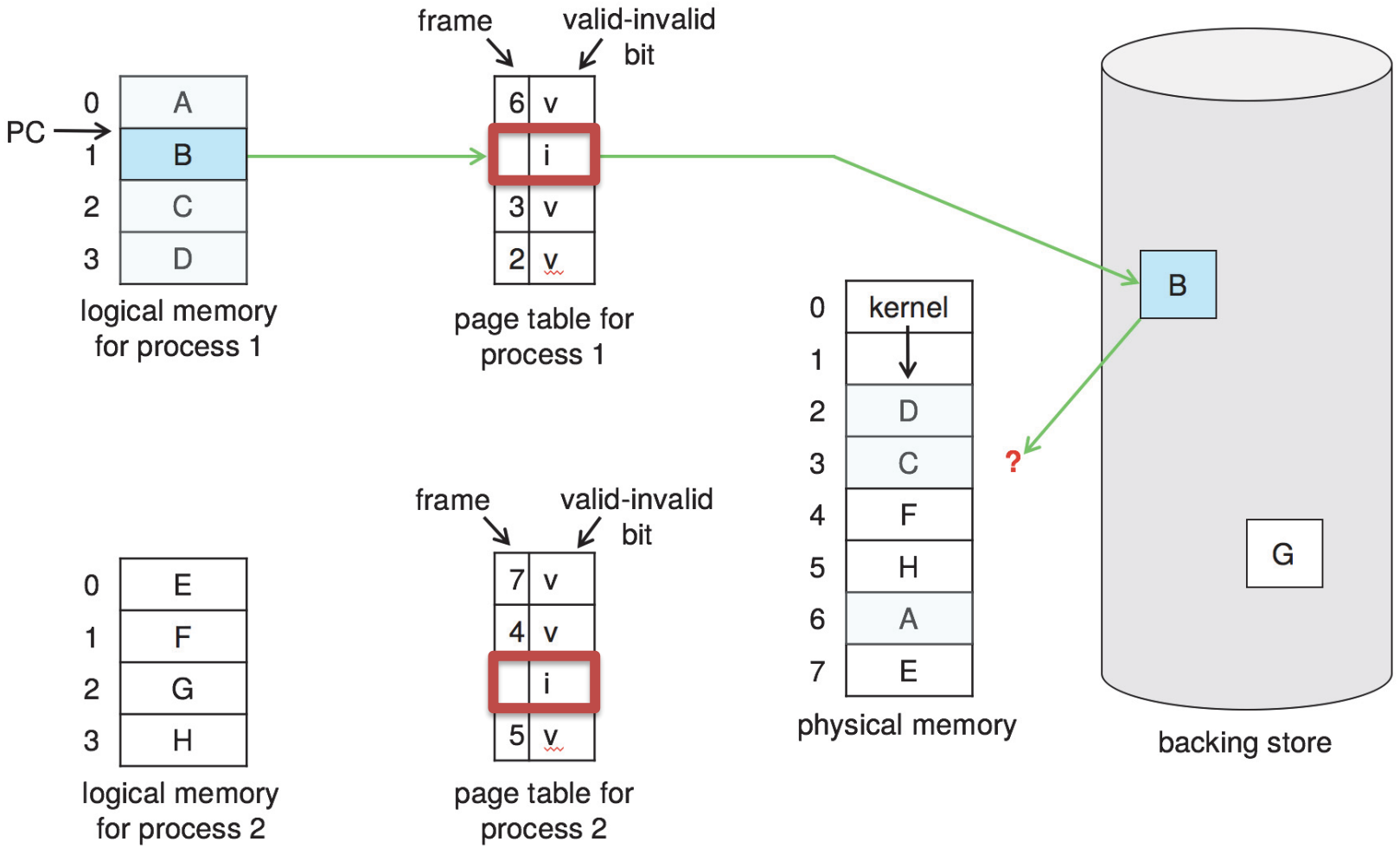
- Pages brought into memory **when accessed first**
  - On program's demand
    - Program may start with no pages in memory
  - Only code/data used by a process needs to be loaded
    - What's used changes over time
- Few systems try to **anticipate** future needs
- Pages may be **clustered**
  - OS keeps track of pages that should come and go together
  - Bring in all when one is referenced

- Demand paging can be **expensive**
  - Heavily depends on storage latency

Page Allocation and Replacement

- When you read in a page, where does it go?
  - If there are free page frames, grab one
    - This is **page allocation**
  - If there are no free page frames, must **evict** one
    - This is **page replacement**
      - Mechanism
      - Algorithm
- OS tries to keep a pool of free pages around
  - To avoid the cost of eviction
- High degree of multiprogramming, causes over-allocation
  - All memory is in use
  - Need to evict

What to Do When All Memory is in Use?



Page Replacement Mechanism