Virtual Memory Management

Dr. Hussain

Virtual Memory Management

- Background
- Demand Paging
- Demand Segmentation
- Paging Considerations
- Page Replacement Algorithms
- Virtual Memory Policies

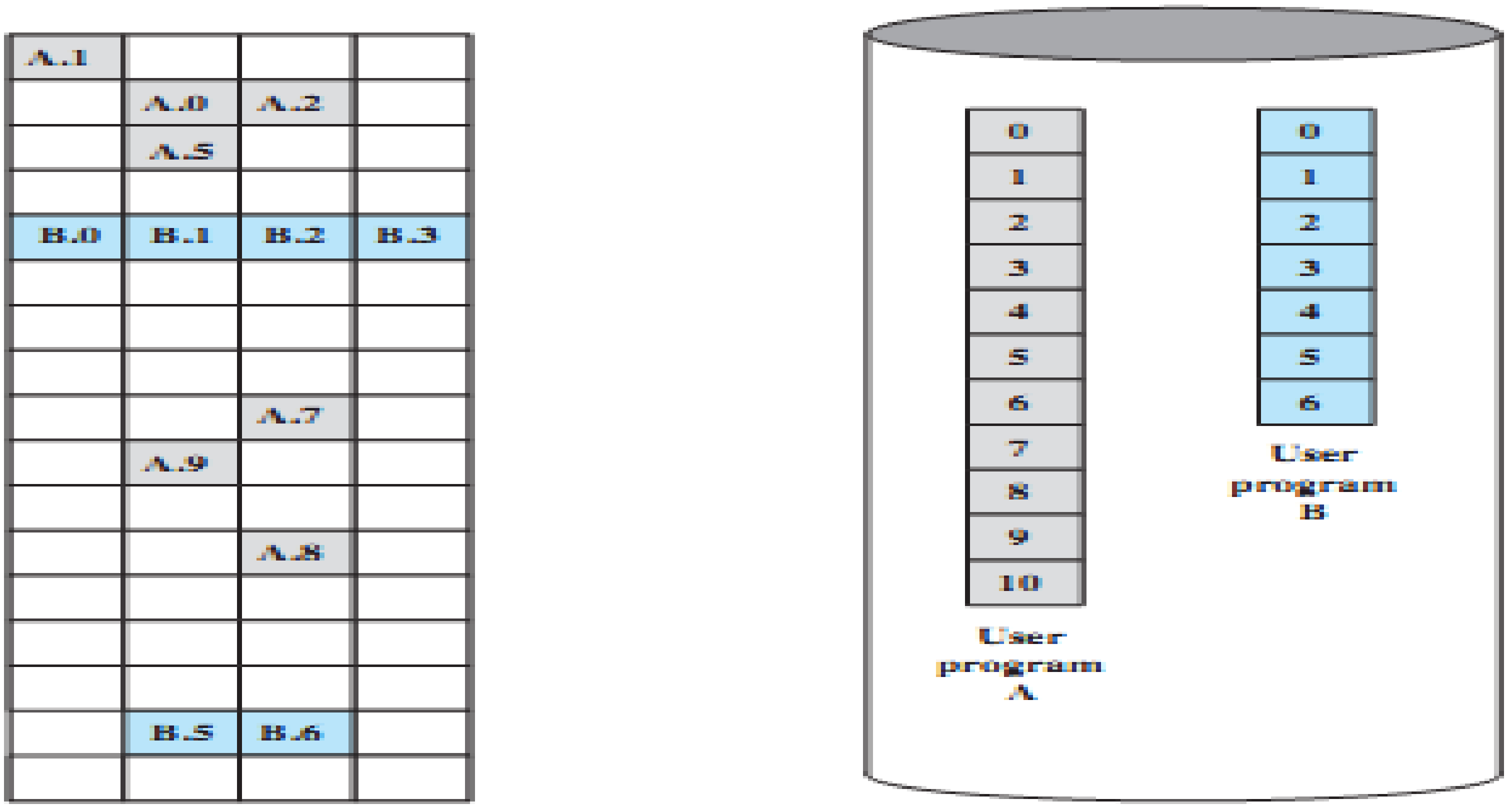
Background (1)

- Code needs to be in memory to execute, but entire program rarely used:
  - Error code, unusual routines, large data structures.
- Entire program code not needed at same time.
- Consider ability to execute partially-loaded program:
  - Program no longer constrained by limits of physical memory.
  - Each program takes less memory while running -> more programs run at the same time:
    - Increased CPU utilization and throughput with no increase in response time or turnaround time.
  - Less I/O needed to load or swap programs into memory -> each user program runs faster.

Background (2)

- **Virtual memory** – separation of user logical memory from physical memory:
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.
  - More programs running concurrently.
  - Less I/O needed to load or swap processes.

Virtual Memory Components



Main Memory

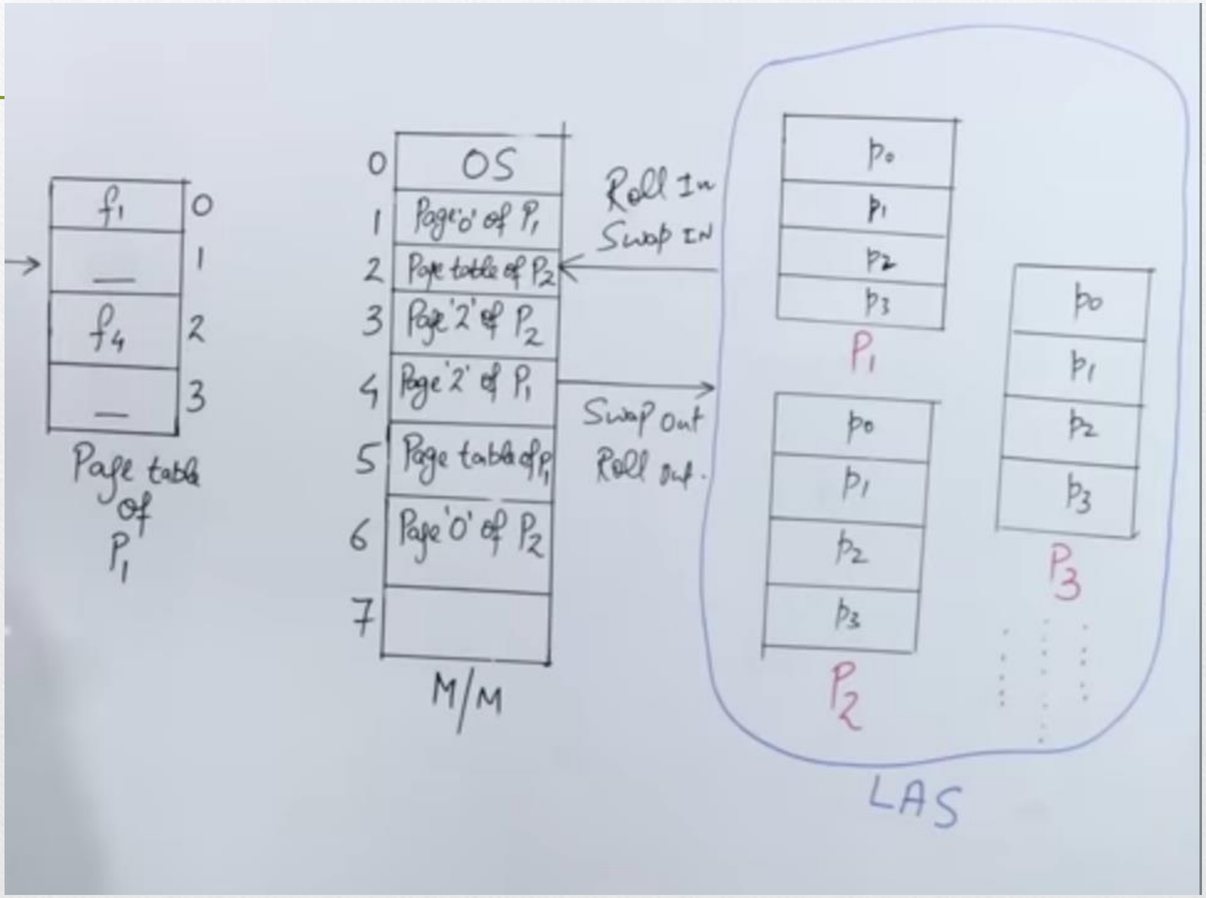
Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Disk

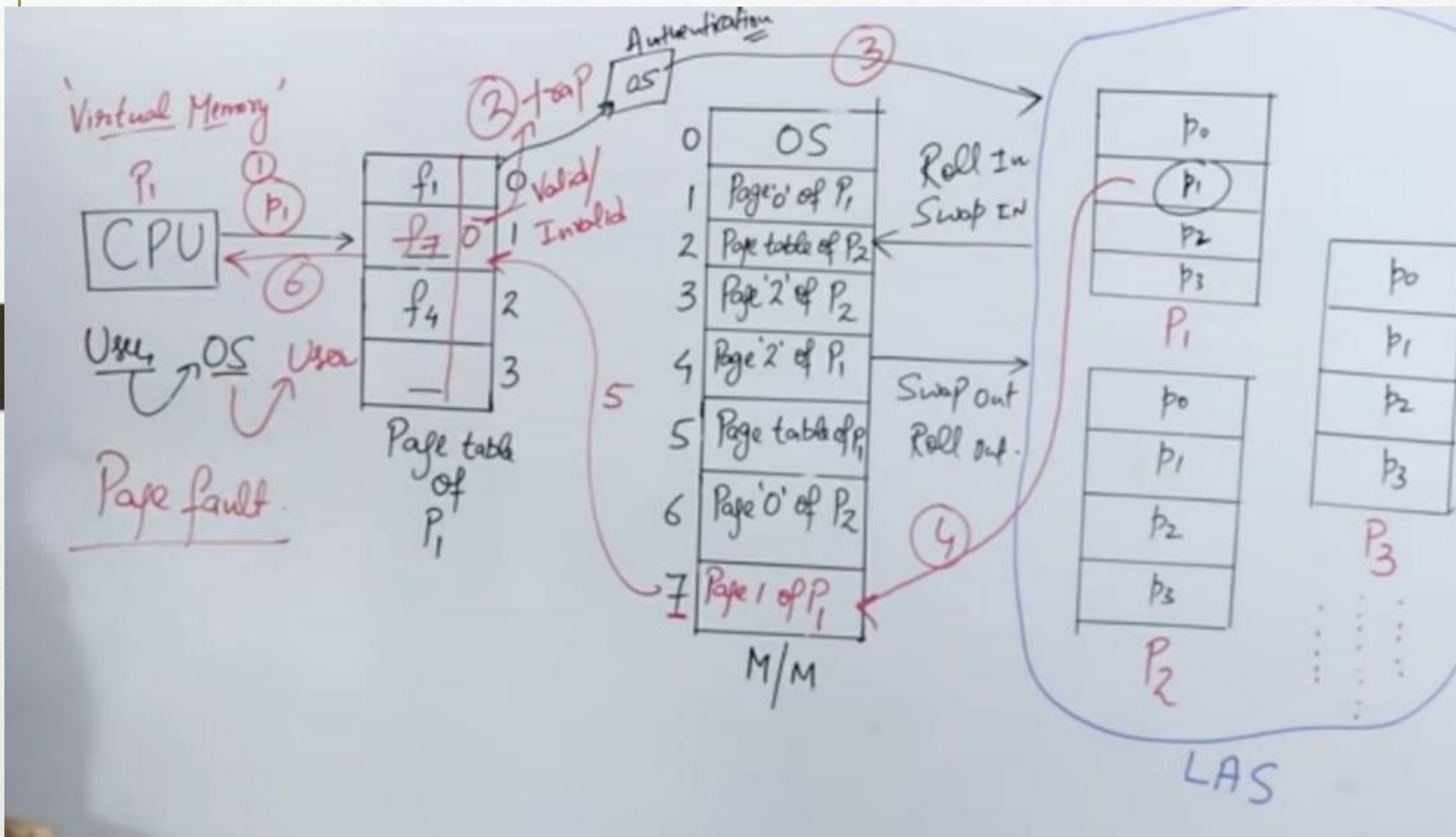
Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

Virtual Memory Components

CPU



Virtual Memory Steps



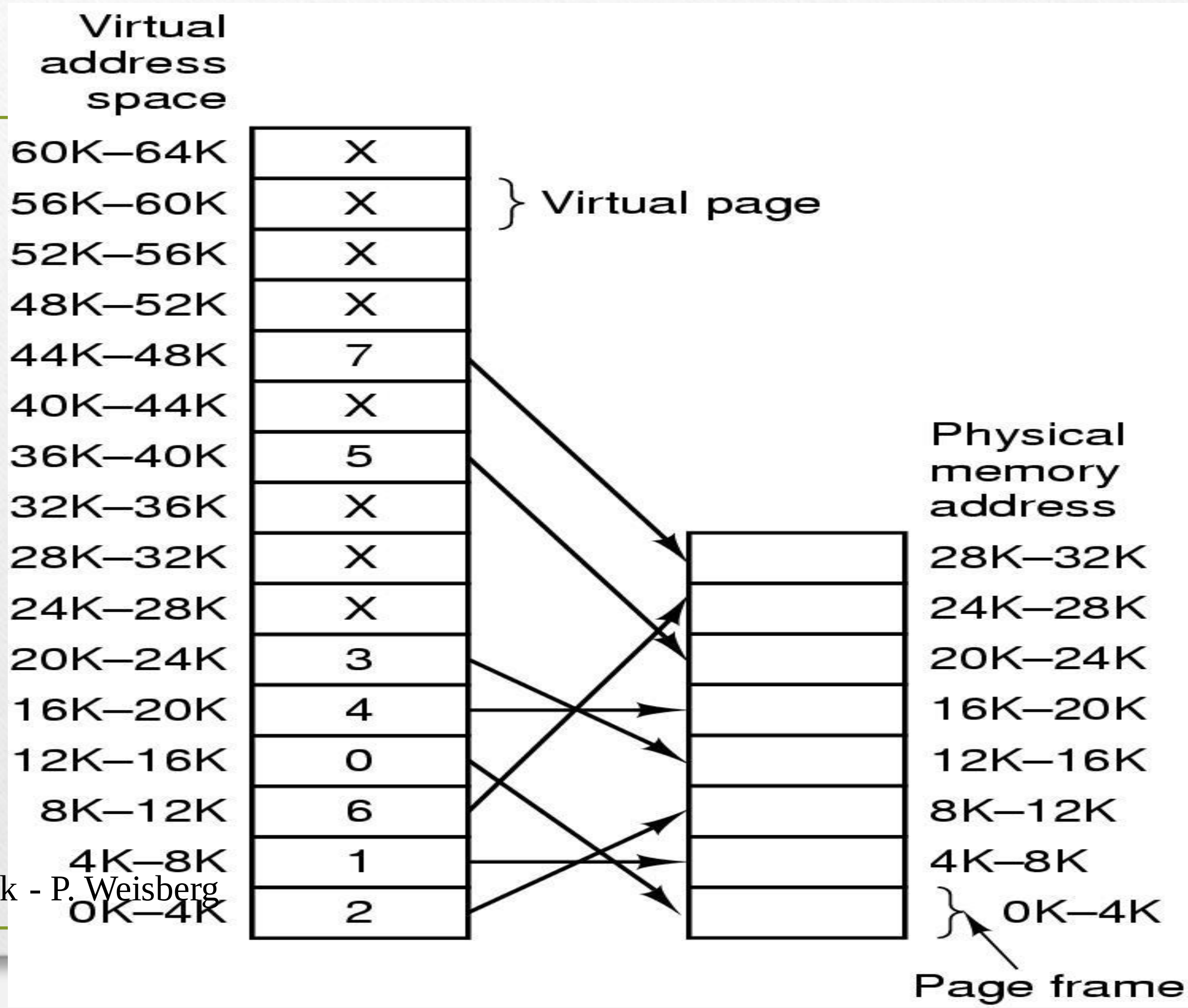
Background (3)

- **Virtual address space** – logical view of how process is stored in memory:
  - Usually start at address 0, contiguous addresses until end of space.
  - Meanwhile, physical memory organized in page frames.
  - MMU must map logical to physical.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

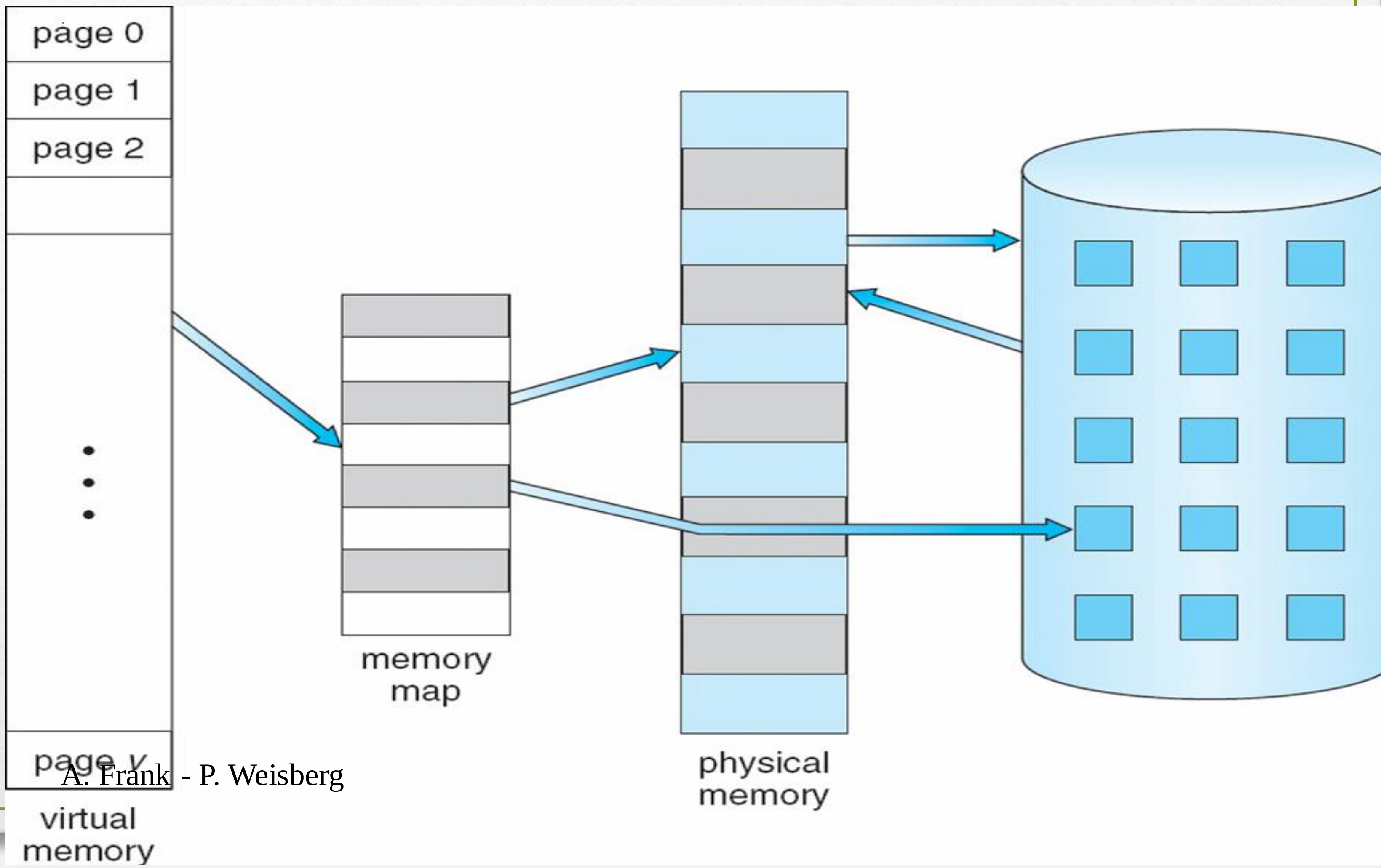
(4) Background

- Based on Paging/Segmentation, a process may be broken up into pieces (pages or segments) that do not need to be located contiguously in main memory.
- Based on the Locality Principle, all pieces of a process do not need to be loaded in main memory during execution; all addresses are virtual.
- The memory referenced by a virtual address is called virtual memory:
  - It is mainly maintained on secondary memory (disk).
  - pieces are brought into main memory only when needed.

Virtual Memory Example



Virtual Memory that is larger than Physical Memory



Advantages of Partial Loading

- More processes can be maintained in main memory:
  - only load in some of the pieces of each process.
  - with more processes in main memory, it is more likely that a process will be in the Ready state at any given time.
- A process can now execute even if it is larger than the main memory size:
  - it is even possible to use more bits for logical addresses than the bits needed for addressing the physical memory.

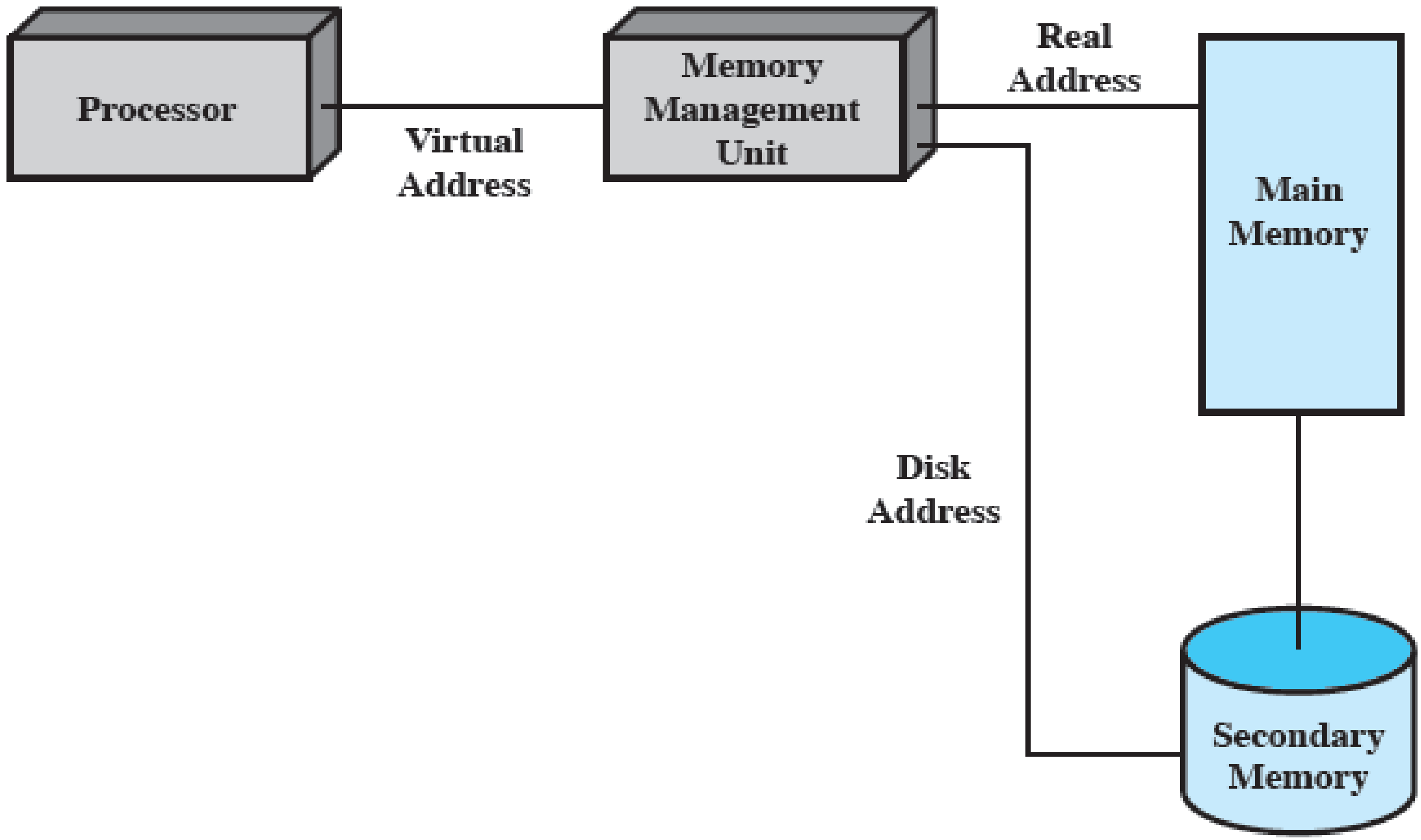
Virtual Memory: Large as you wish!

- Example:
  - Just 16 bits are needed to address a physical memory of 64KB.
  - Lets use a page size of 1KB so that 10 bits are needed for offsets within a page.
  - For the page number part of a logical address we may use a number of bits larger than 6, say 22 (a modest value!!), assuming a 32-bit address.

Support needed for Virtual Memory

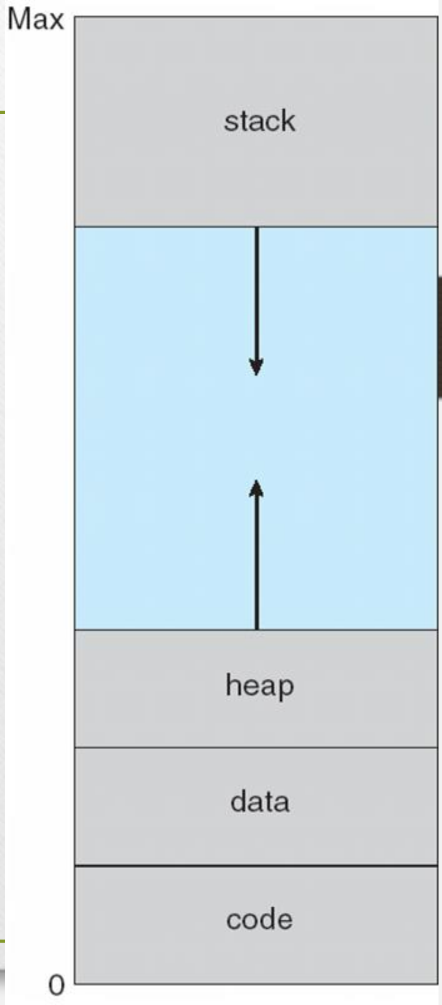
- Memory management hardware must support paging and/or segmentation.
- OS must be able to manage the movement of pages and/or segments between external memory and main memory, including placement and replacement of pages/segments.

Virtual Memory Addressing

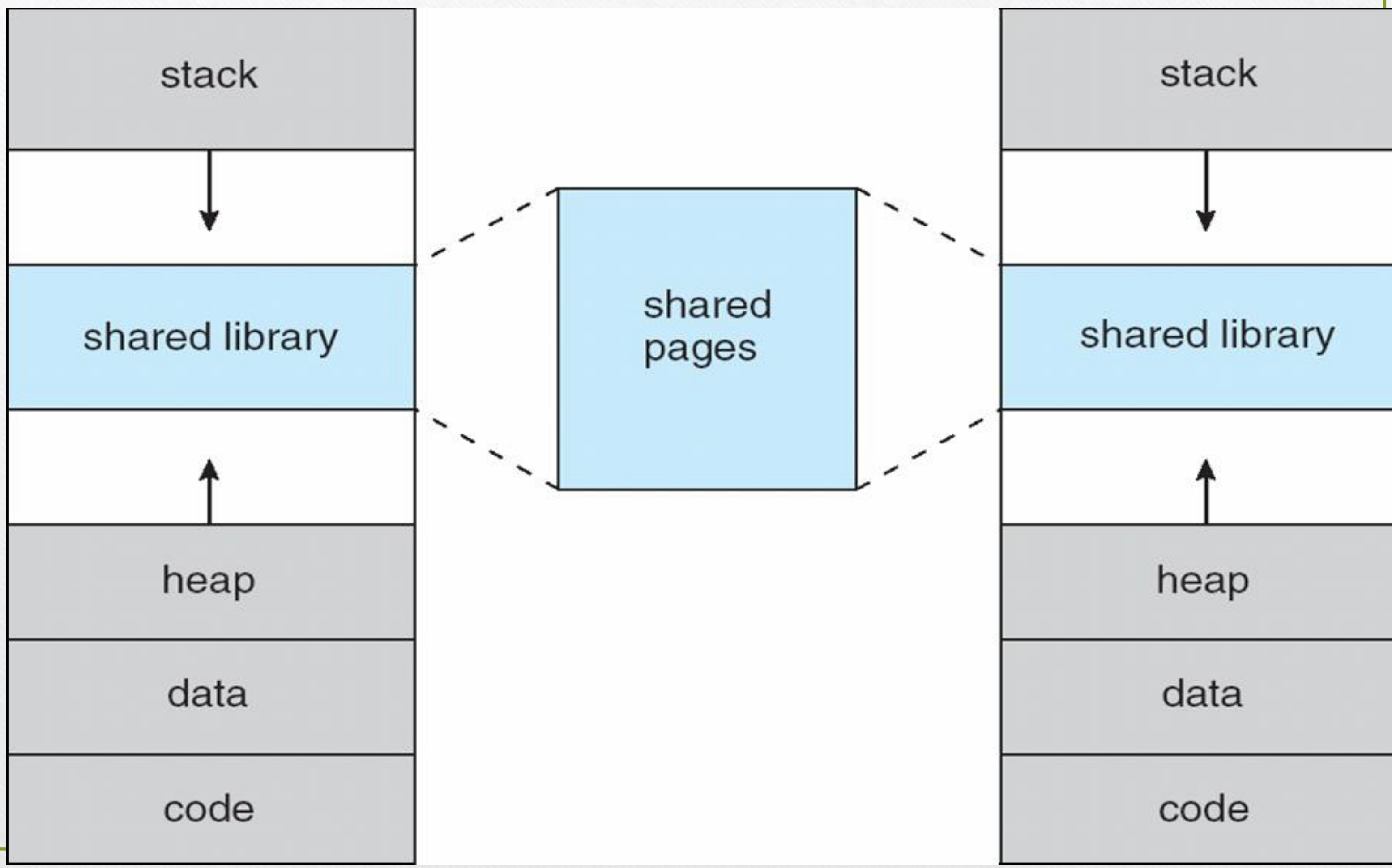


Virtual-address Space

- Usually design logical address space for stack to start at Max logical address and grow "down" while heap grows "up":
  - Maximizes address space use.
  - Unused address space between the two is hole.
  - No physical memory needed until heap or stack grows to a given new page.
- Enables sparse address spaces with holes left for growth, dynamically linked libraries, etc.
- System libraries shared via mapping into virtual address space.
- Shared memory by mapping pages read-write into virtual address space.
- Pages can be shared during `fork()`, speeding process creation.



Shared Library using Virtual Memory



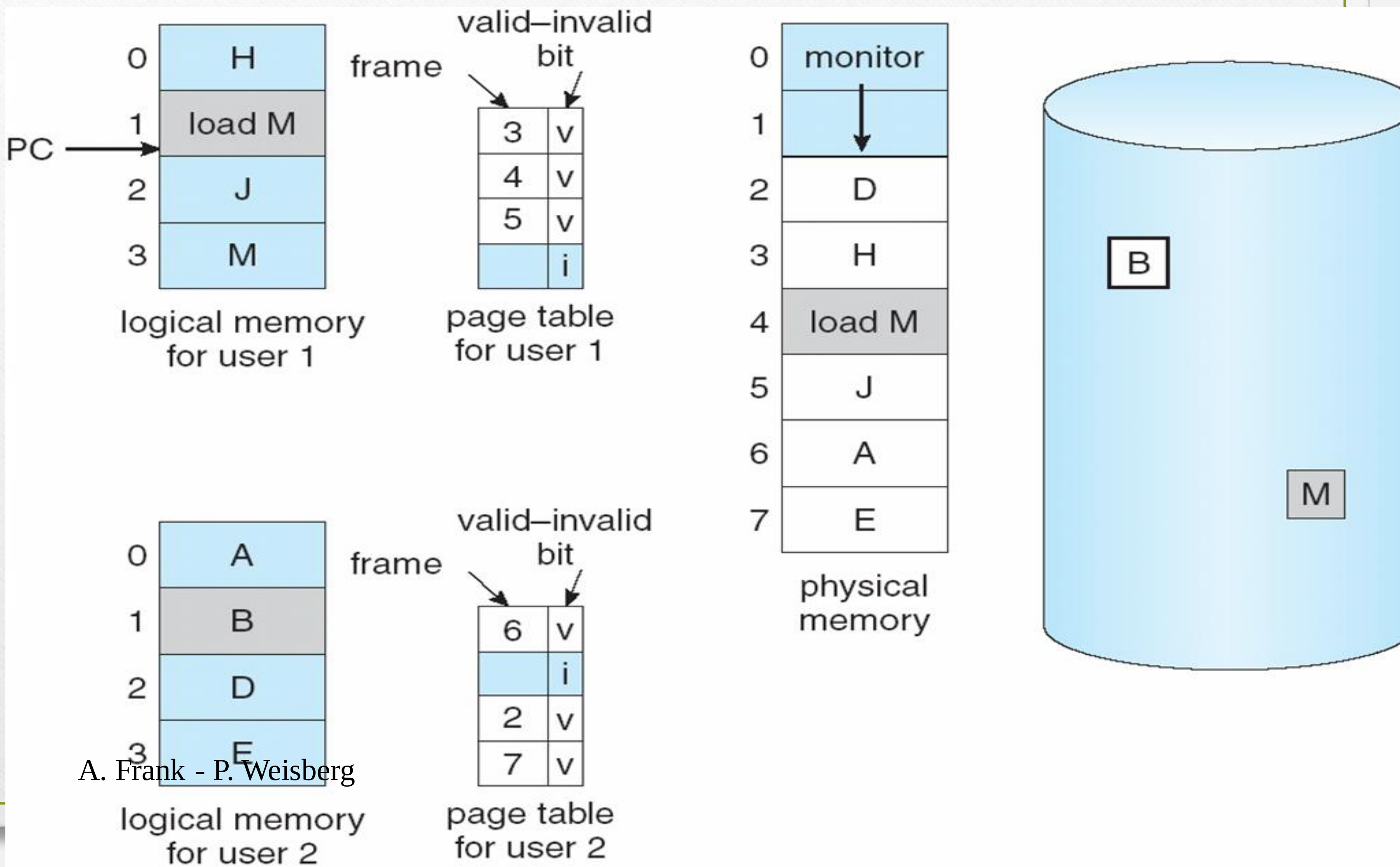
Process Execution (1)

- The OS brings into main memory only a few pieces of the program (including its starting point).
- Each page/segment table entry has a valid-invalid bit that is set only if the corresponding piece is in main memory.
- The resident set is the portion of the process that is in main memory at some stage.

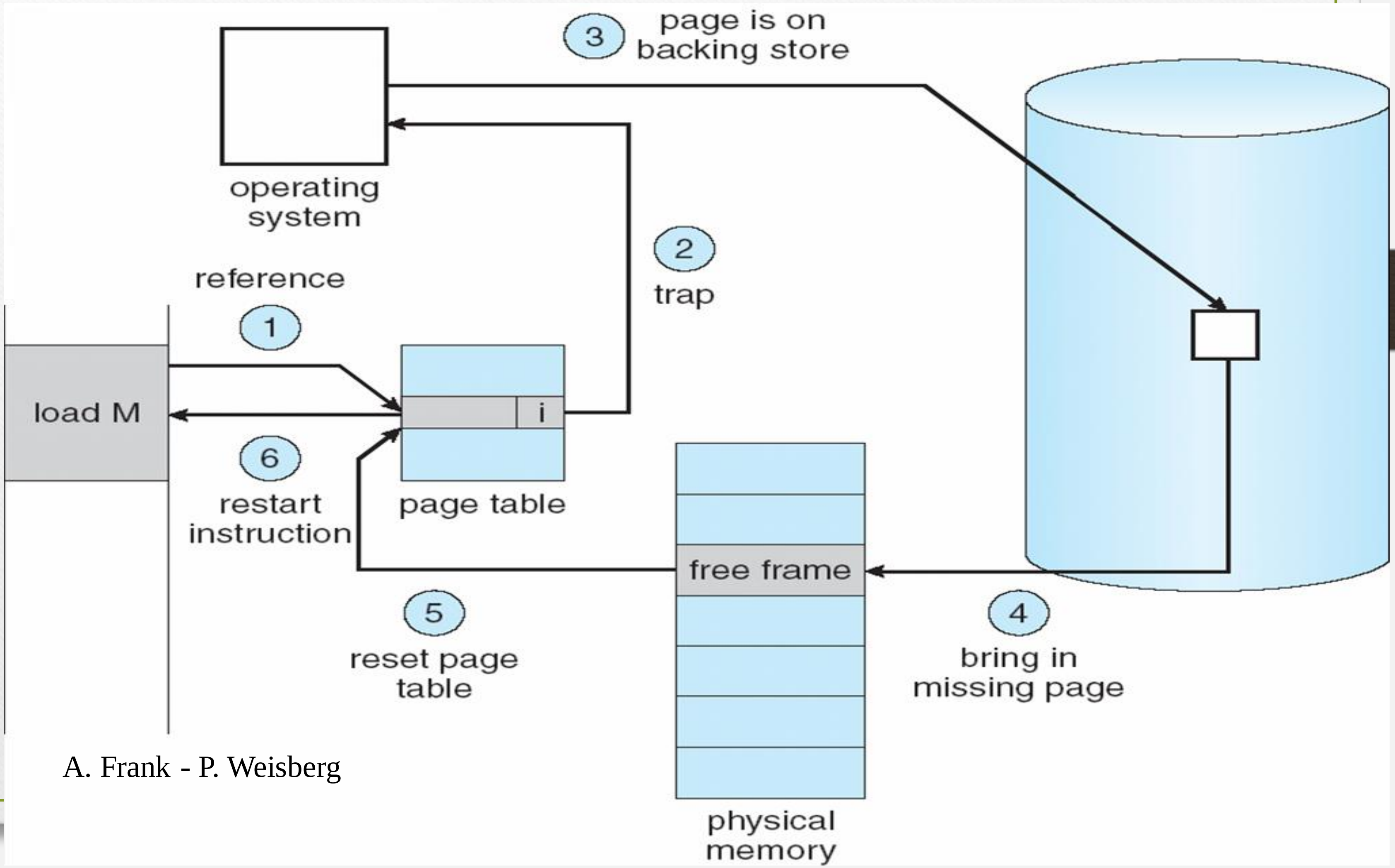
Process Execution (2)

- An interrupt (memory fault) is generated when the memory reference is on a piece that is not present in main memory.
- OS places the process in a Blocking state.
- OS issues a disk I/O Read request to bring into main memory the piece referenced to.
- Another process is dispatched to run while the disk I/O takes place.
- An interrupt is issued when disk I/O completes; this causes the OS to place the affected process back in the Ready state.

Need For Page Fault/Replacement



Steps in handling a Page Fault (1)



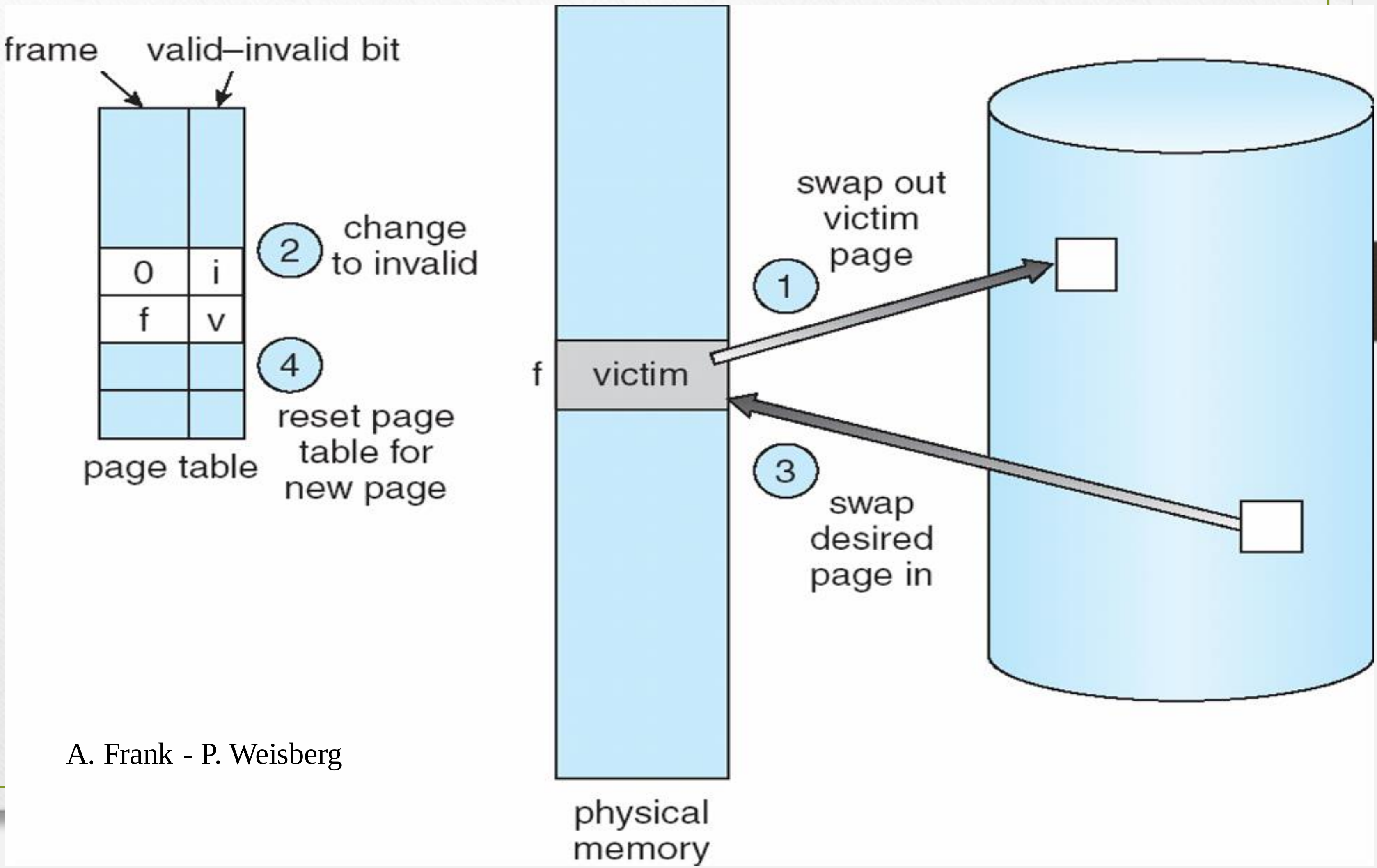
Steps in handling a Page Fault (2)

1. If there is ever a reference to a page not in memory, first reference will cause page fault.
2. Page fault is handled by the appropriate OS service routines.
3. Locate needed page on disk (in file or in backing store).
4. Swap page into free frame (assume available).
5. Reset page tables – valid-invalid bit = v.
6. Restart the instruction that caused the page fault.

What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out.
- Need page replacement algorithm.
- Performance – want an algorithm which will result in minimum number of page faults.
- Same page may be brought into memory several times.

Steps in handling a Page Replacement (1)



Steps in handling a Page Replacement (2)

1. Find the location of the desired page on disk.
2. 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 page.
3. Bring the desired page into the (newly) free frame; update the page and frame tables.
4. Restart the process.

Comments on Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use modify (dirty) bit to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.