


Comp 310
Computer Systems and Organization

Lecture #15
Memory Management
(Memory Allocation)

Prof. Joseph Vybihal



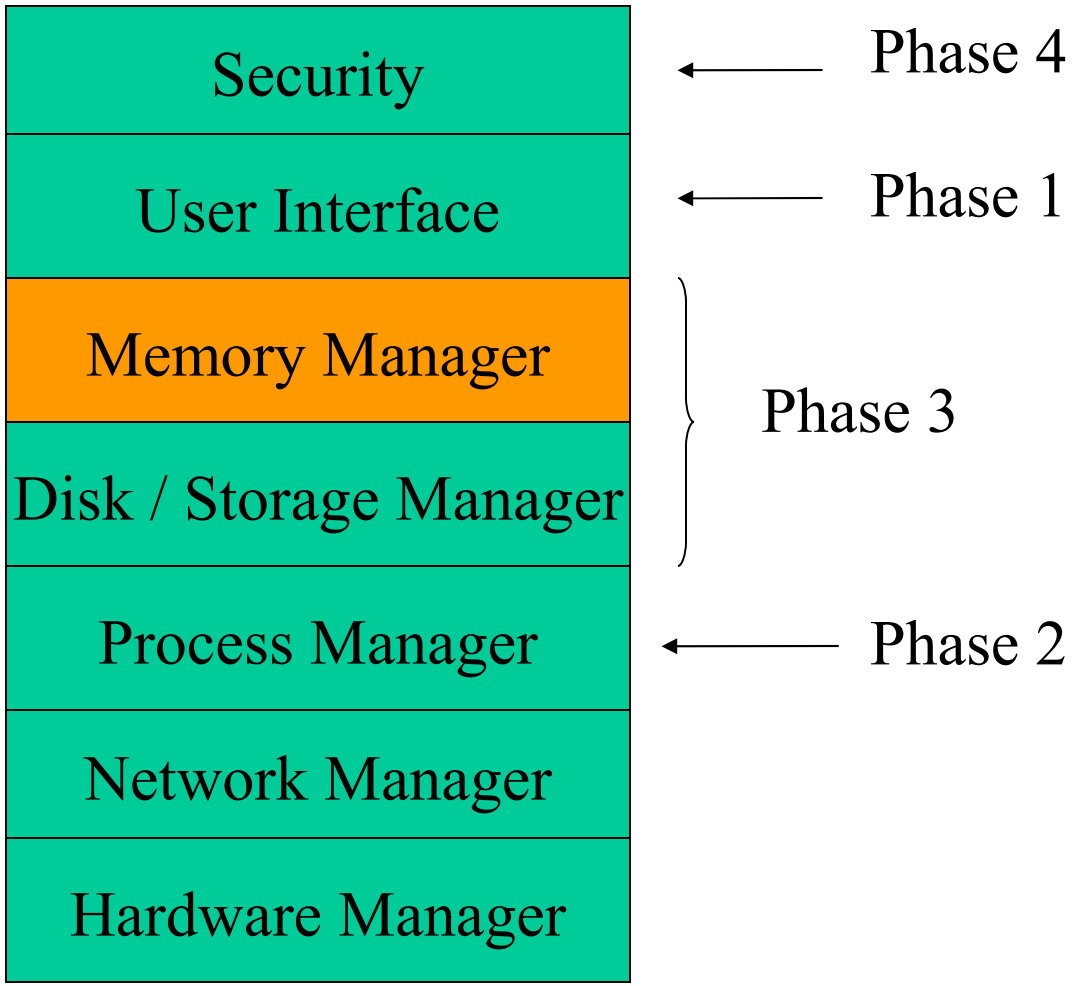

Announcements

- Ass#3?
- Course evaluation:
  - Minerva
  - Start date??
  - Important to participate
- Tentative Final Exam:
  - Dec 9 2PM




Basic OS Architecture

(Course Table of Contents)

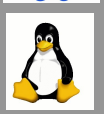





The Kernel & Loading Process

- When does loading occur in the Kernel?
  - How does a shell and login fit?
    - Is login a privileged process, is shell?
- About the kernel loop and spawning initial processes and how it connects
  - Kernel not tasked switched, defines task switch
    - Invoked as a task switched process
      - Invokes initial process: login
      - Login invokes shell, user's connection to OS
      - Shell invokes user processes






Memory Allocation

- Maybe it is obvious, but when the kernel invokes a process we need to find space for it in RAM.

- Two techniques:
  - All process in RAM
    - Continuous object, or variation → swapping
    - Static / dynamic separation (RAM & PCB)
  - Virtual memory
    - Like 'All Process' but simulated
      - Paging and segmentation with “backing store”




Part 1

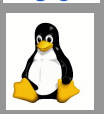
Memory Allocation Issues




Memory Allocation Issues

- The Dynamic Storage Allocation Problem
- The Memory Fragmentation Problem



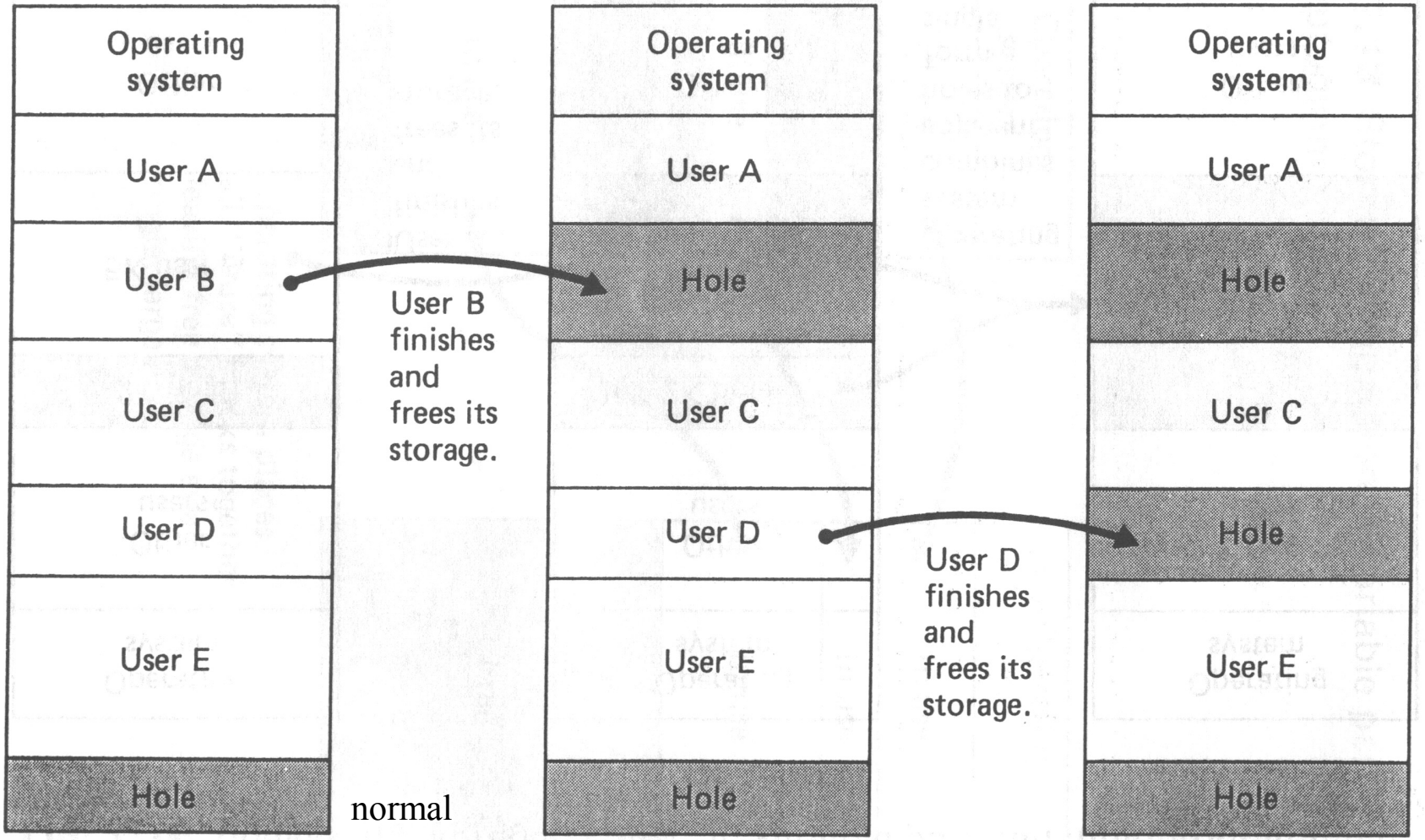



Memory Allocation Issues

- The Dynamic Storage Allocation Problem

Problem: Given free RAM, where should we load the processes?
Issue : When process terminates and memory becomes free, a **hole** is left in RAM.
Result : Best way to use memory to reduce Fragmentation

Holes


Operating system
User A
User B
User C
User D
User E
Hole
normal
User B finishes and frees its storage.
Operating system
User A
Hole
User C
User D
User E
Hole
User D finishes and frees its storage.
Operating system
User A
Hole
User C
Hole
User E
Hole




Hole Management

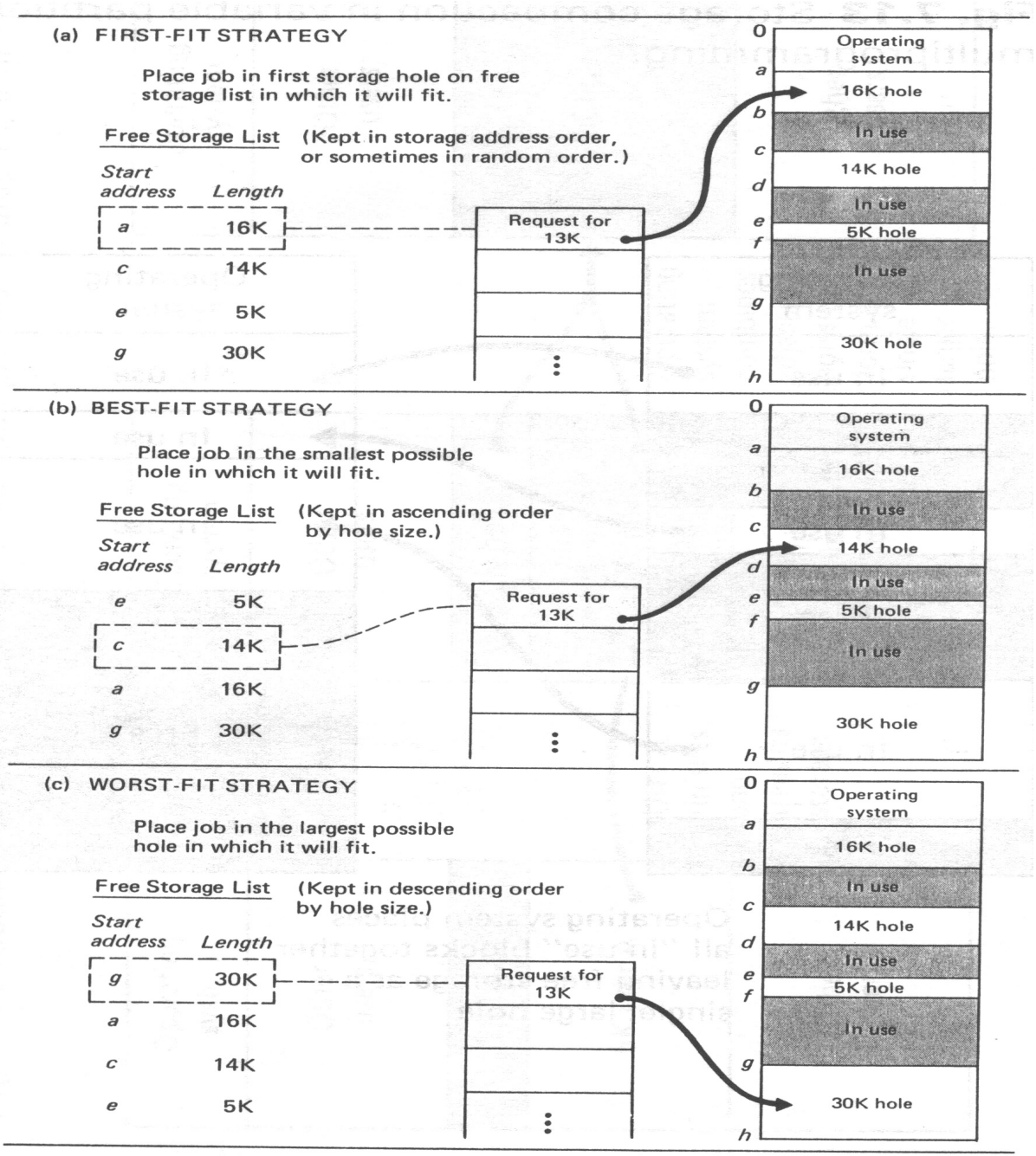
If we have holes and a new process is executed…

Management of holes:

- First fit (best processing time)
- Best fit (smallest left over hole)
- Worst fit (largest left over hole)

Advantages?

When should we kick someone out?

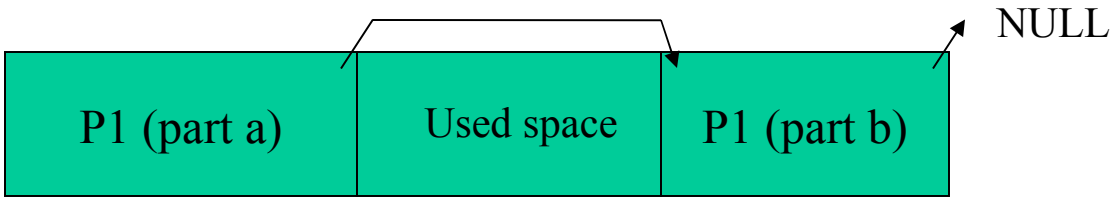


How can any of these techniques be optimized for speed and memory?



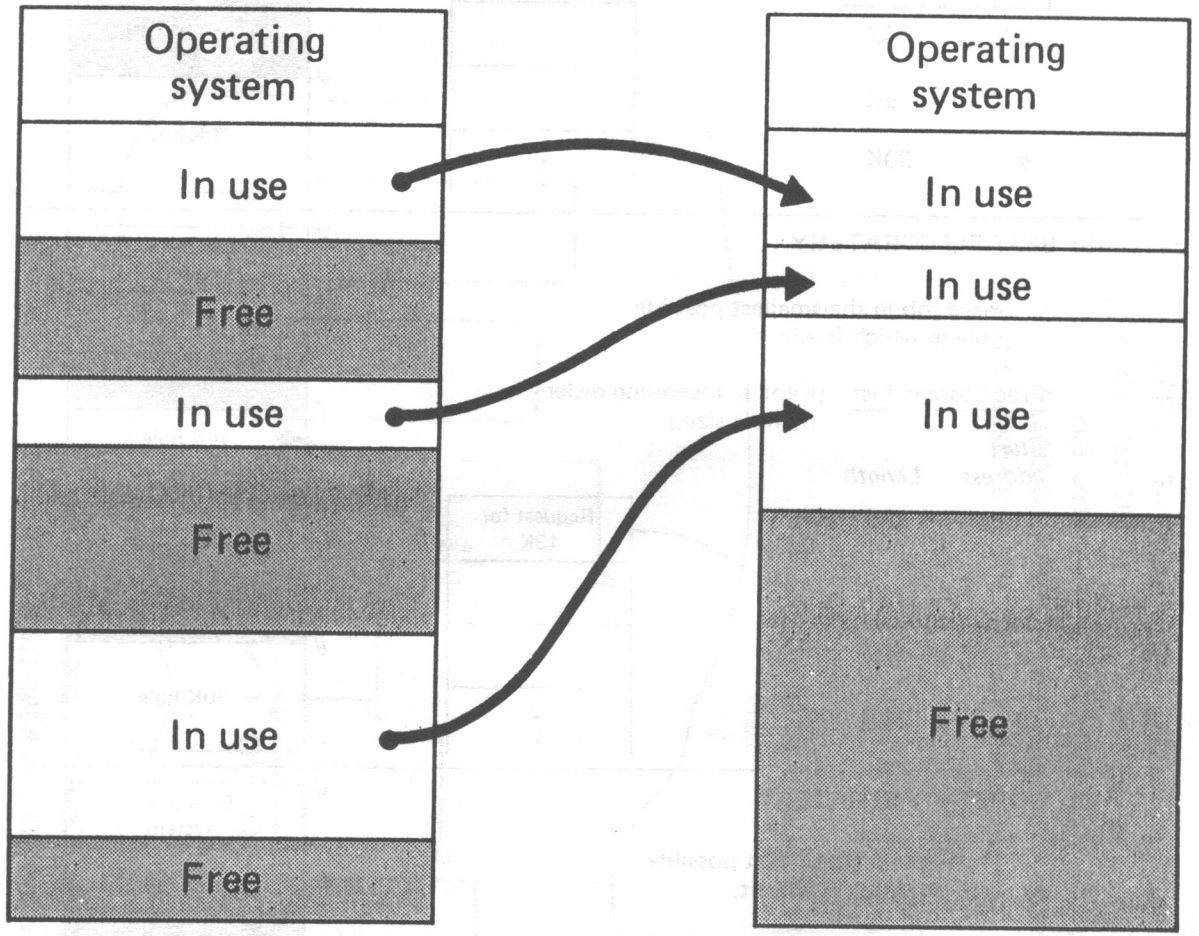

Memory Allocation Issues

- The Memory Fragmentation Problem
  - Solution: De-fragmenting
    - Method:
      - Stop all process execution
      - Compact memory by moving processes byte by byte
      - Update PCB addresses
      - Resume processing
    - Issues:
      - Time intensive
      - Frequency
  - Solution: Non-contiguous process memory



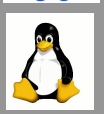



Compaction



Operating system places all "in use" blocks together leaving free storage as a single, large hole.






Questions

- Is memory compaction optimal?
  - w.r.t
    - Time
    - Space
- When should it be used?
- When would it fit into the OS run-time cycle?
- How could we implement it?




Part 2

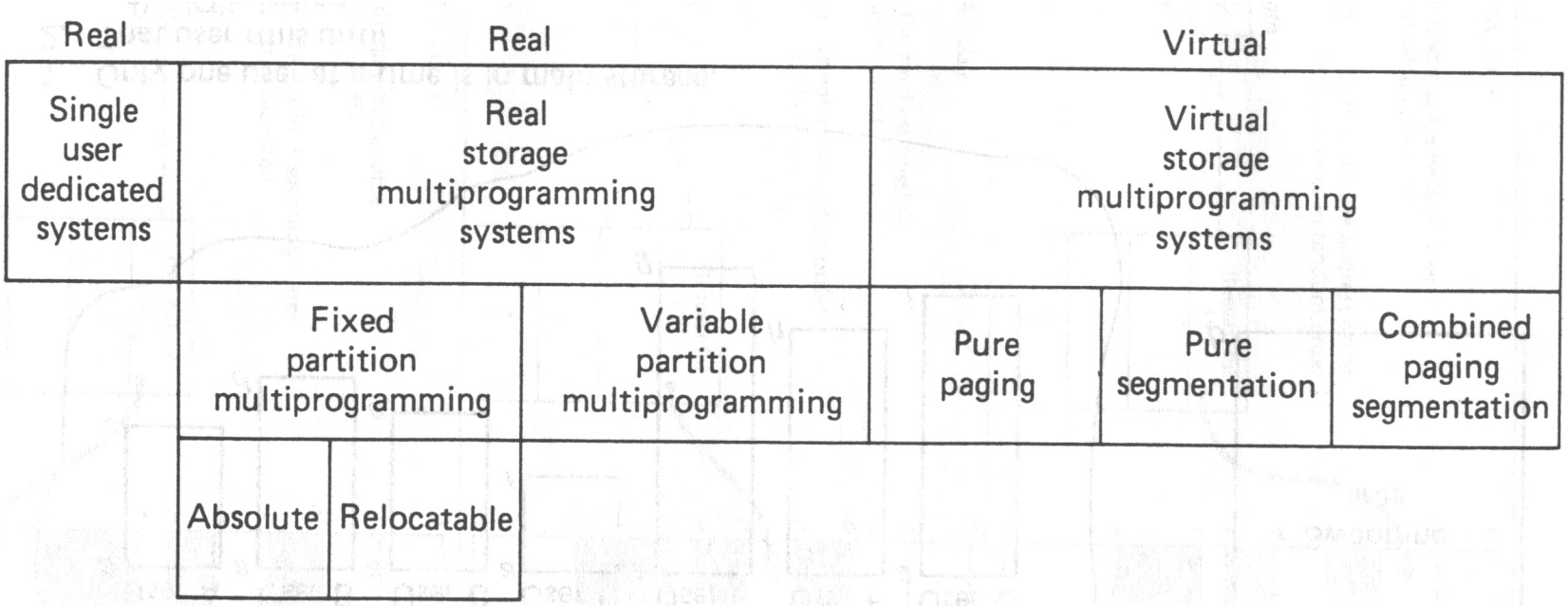
Memory Allocation Methods




Memory Allocation Methods

- Single User Absolute/Default Addressing
- Multi Process Fixed Partition Addressing
- Paging
- Segmentation
- Hybrid Models
- Virtual Memory (will be covered in its own lecture)

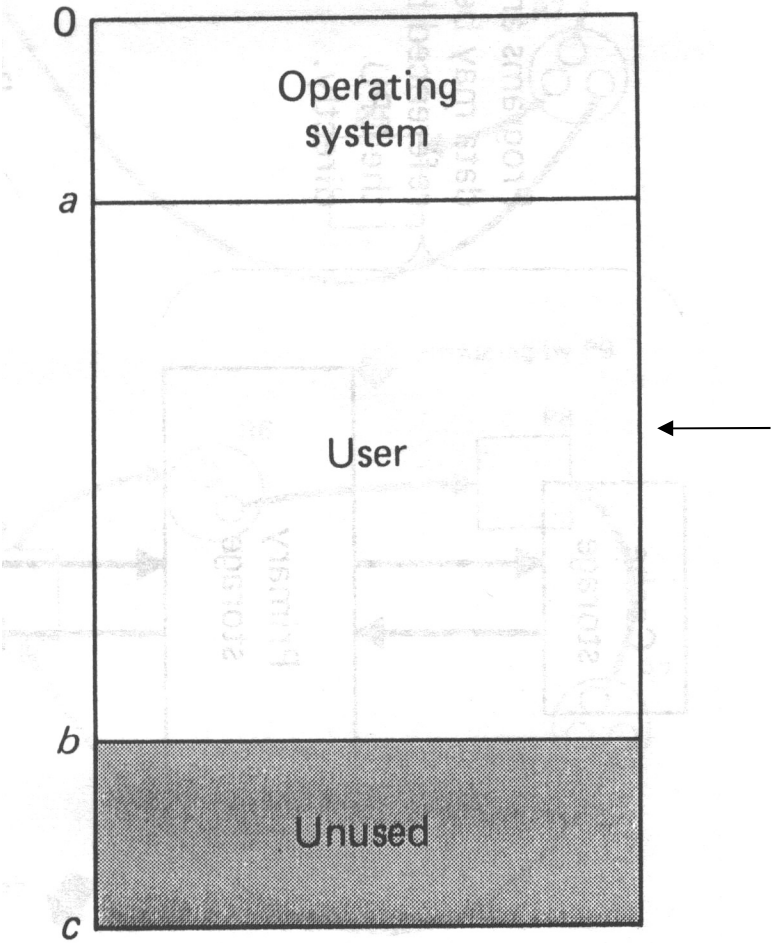
Evolution of Memory






Single User Absolute/Default Addressing

Single User Absolute/Default Addressing

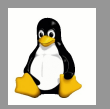


Two options possible:

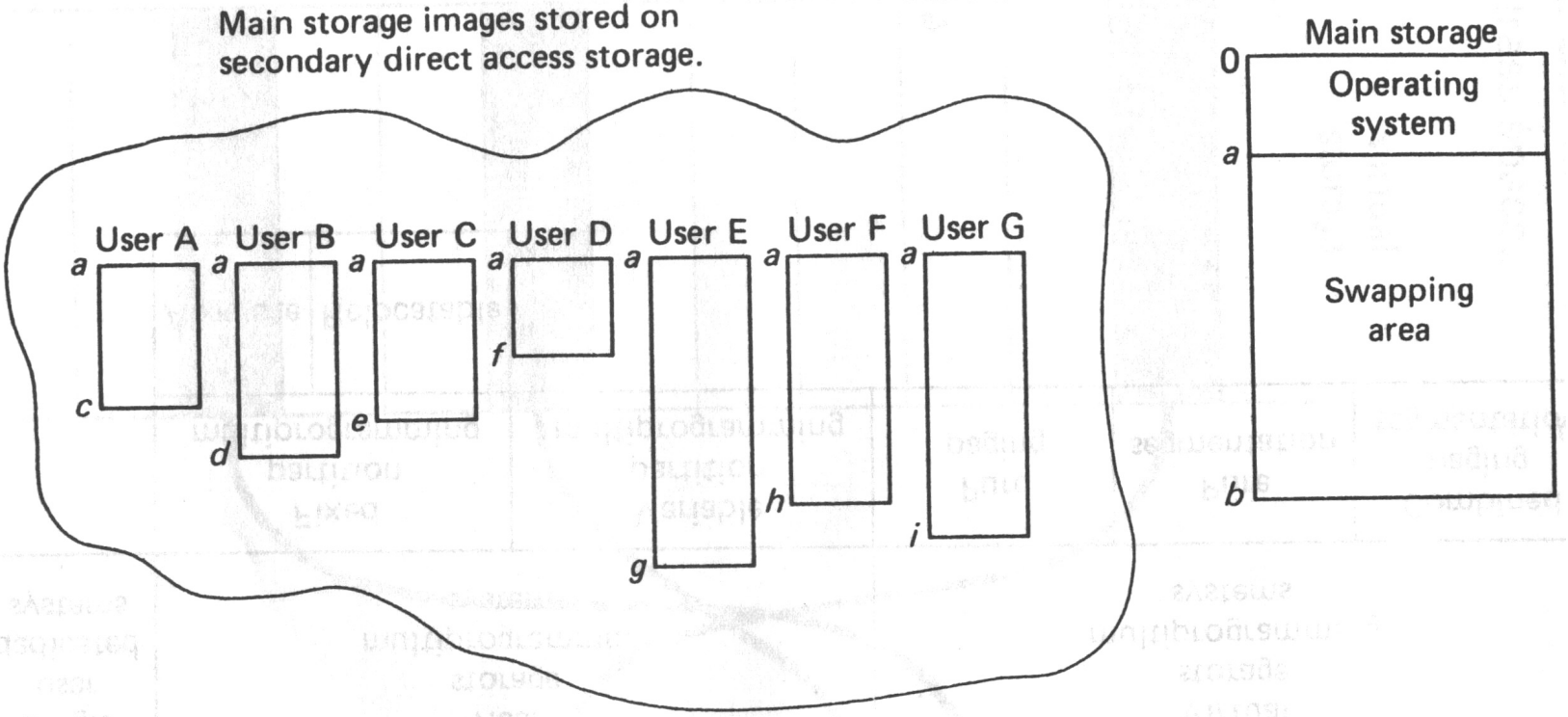
- Absolute Addressing:
  - User specifies exact address in RAM where process will load and execute from
- Default Addressing:
  - User does not specify address and OS loads to a default location and executes from there

No concerns about holes – there is only one hole.






Multiprogramming in a swapping system in which only a single user at a time is in main storage.



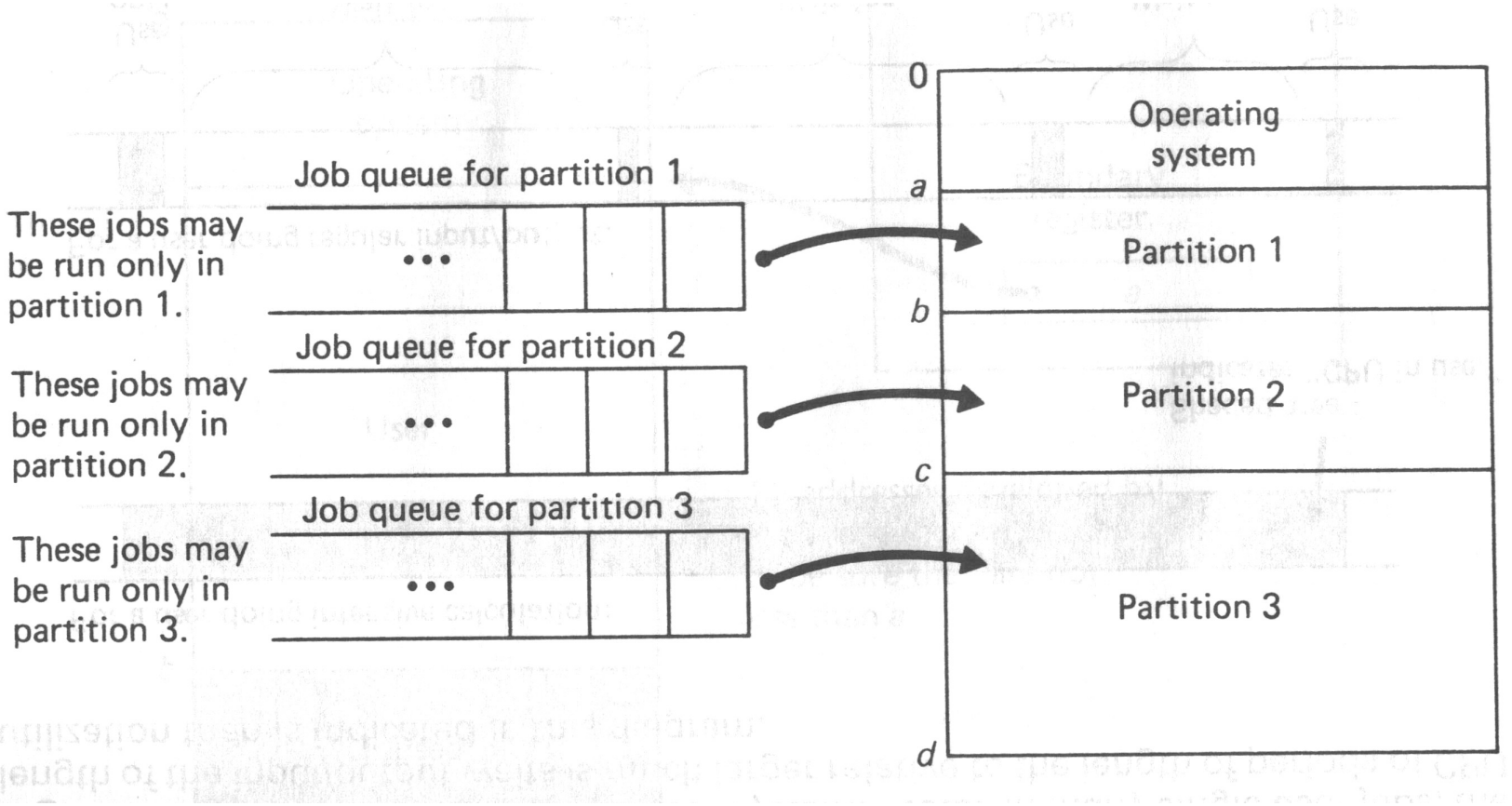
1. Only one user at a time is in main storage.
2. That user runs until
   a) I/O is issued,
   b) timer runout,
   c) voluntary termination.
3. Swapping area (main storage image) for that user is then copied to secondary storage (i.e., "swapped out").
4. Main storage image for next user is read into the swapping area (i.e., "swapped in") and that user runs until it is eventually swapped out and the next user is swapped in, etc.
5. This scheme was common in early timesharing systems.

PROCESSES IN WAITING QUEUE




Multi Process Fixed Partition Addressing

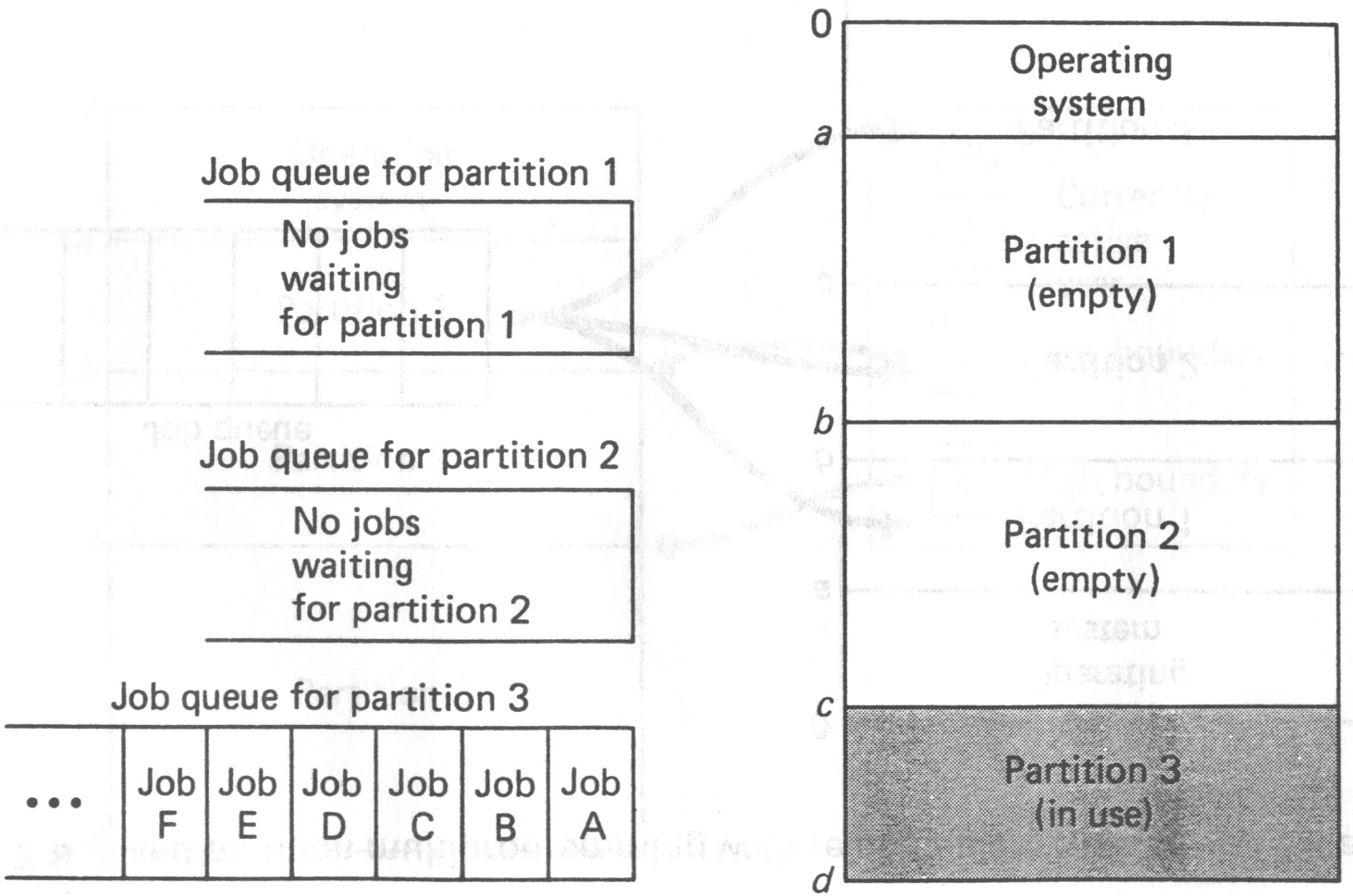
Multi Process Fixed Partition Addressing



No problem with holes.

The problem is now about the queues and partition sizes.

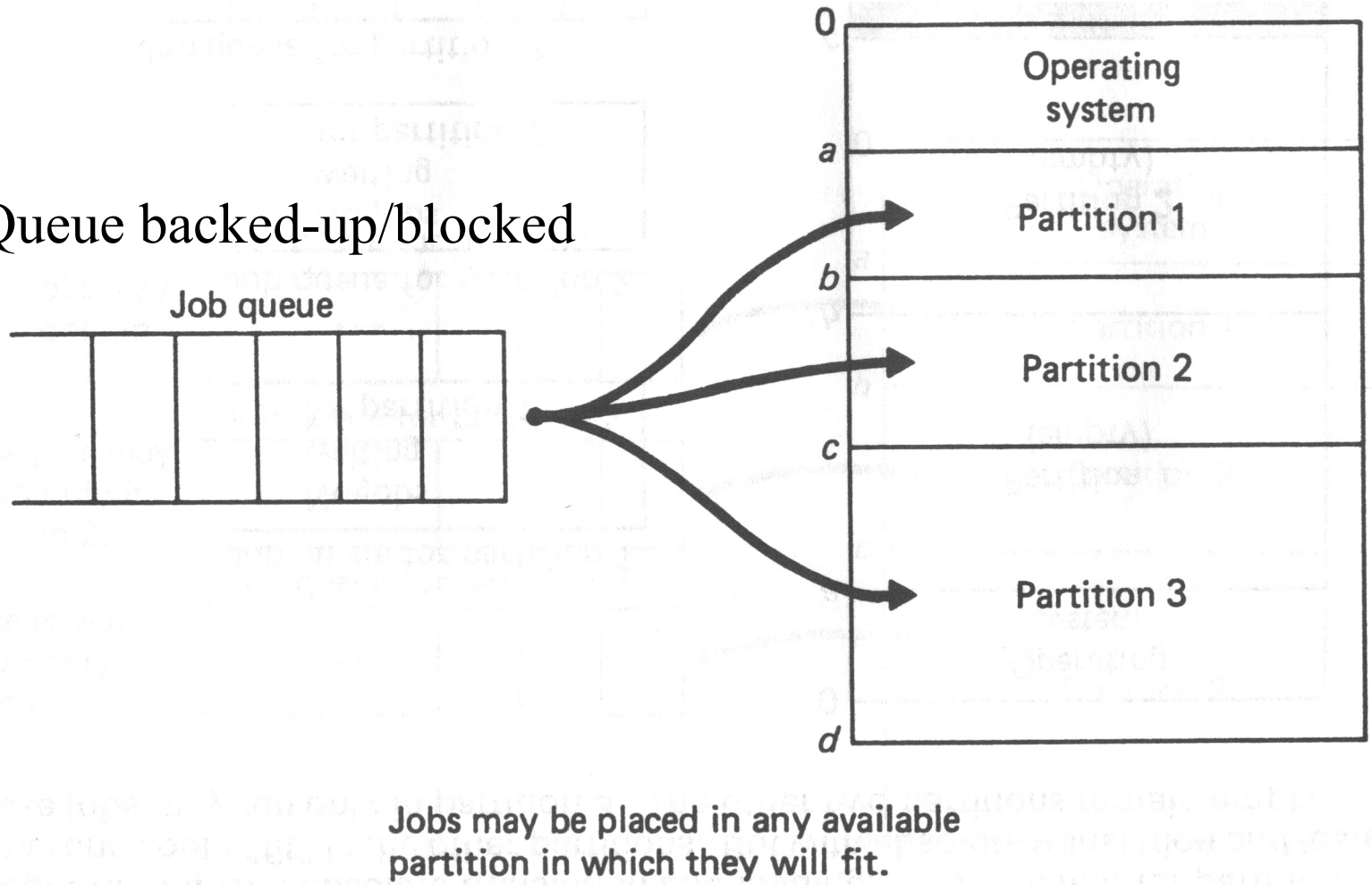
Queue Problems



Extreme example of a queue problem

Queue Problem Solution

Queue backed-up/blocked



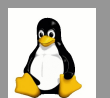
Back to the problem with holes… (add to it indefinite postponement)

Except these holes have a fixed size



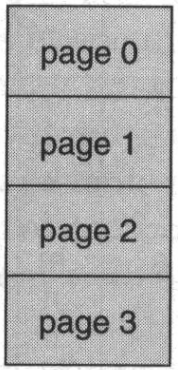

Paging



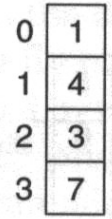



Paging Elements

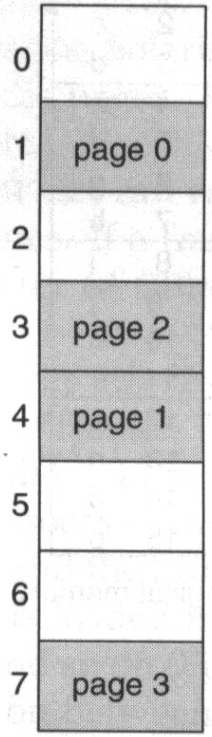
- Due to memory limitations, want:
  - Ability to swap in/our programs
  - Don't want to worry about holes
  - The user to think there are no memory limits
- Problem
  - We do not know the size of user's program
  - User programs have different sizes
  - Swapping an entire program is a lot of overhead
- Solution
  - Setup a fixed page size
  - User's program is segmented into n pages
  - Swap in/out by the page



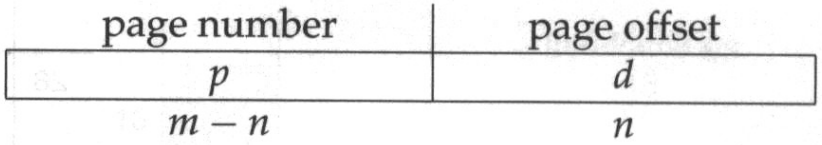
logical memory



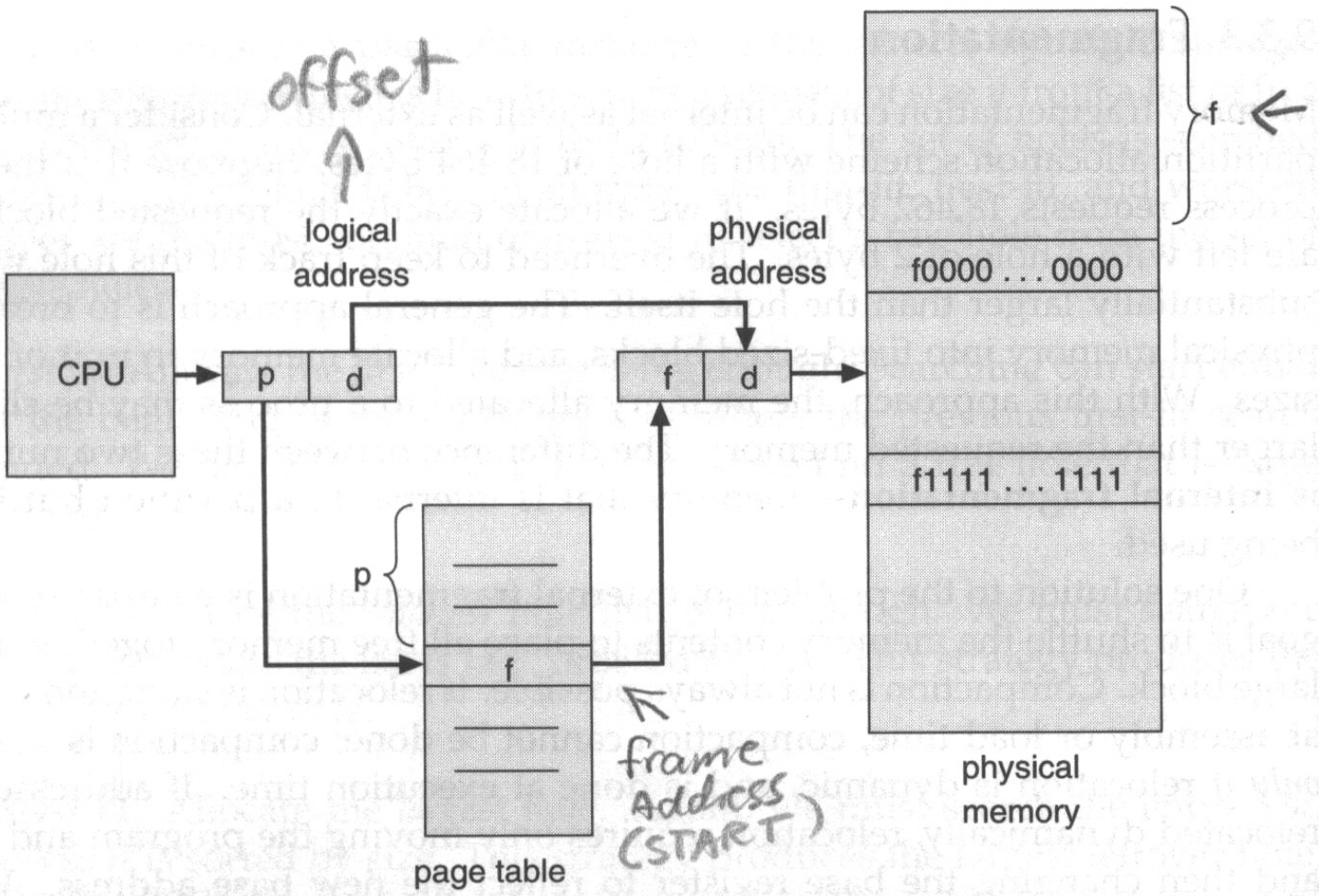
page table



physical memory

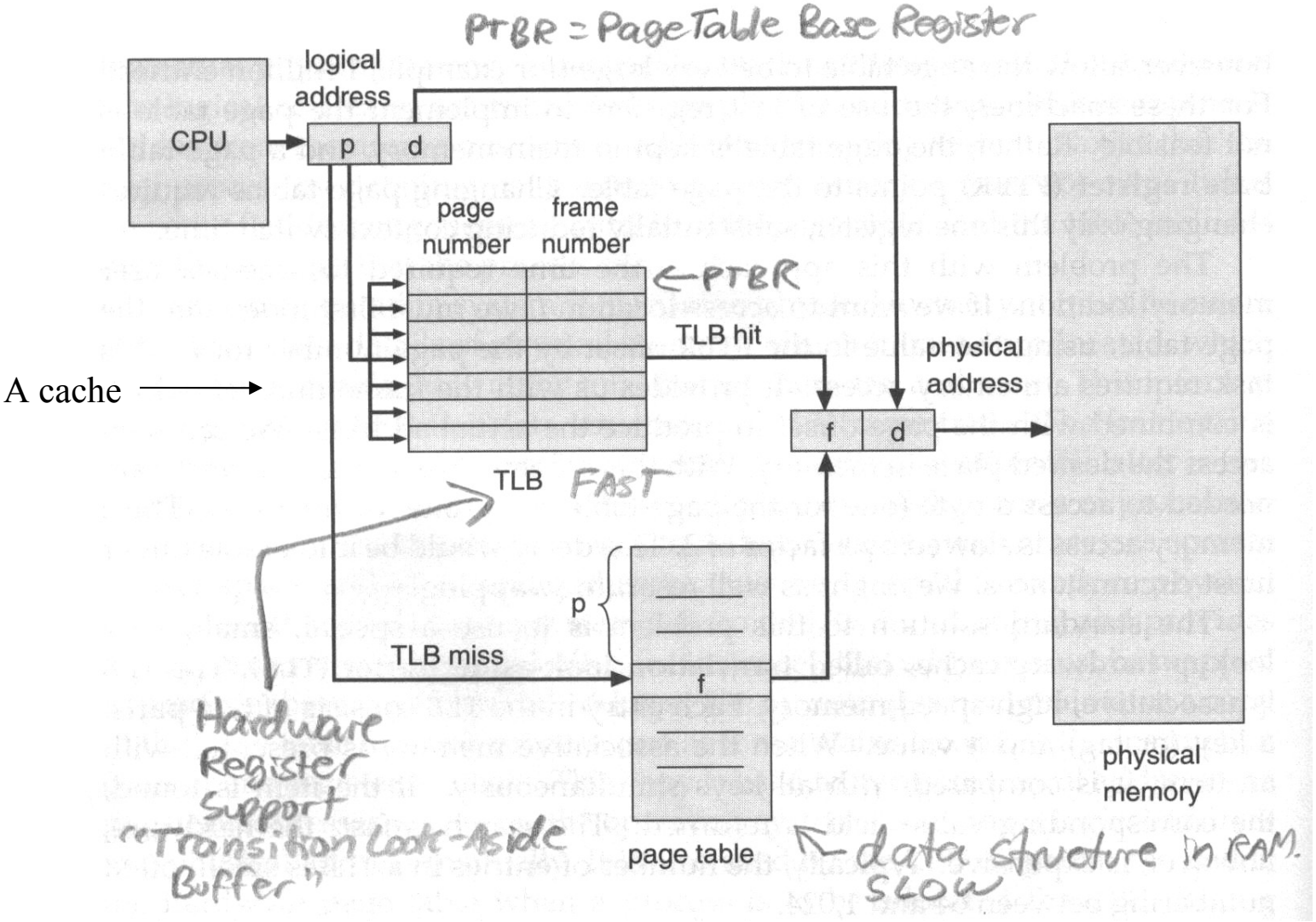


Page Addressing Hardware



Using concatenation

Page Addressing Hardware






Part 3

At Home




Things to try out

1. If you have an advanced compiler like Visual C++ or C++ Builder…
   - Load a large program you have written
   - Go to the compilation options section and play with the values and run your program after each change. Try the following:
     - Change stack & heap sizes
     - Turn on/off page swapping
     - Force entire program in RAM
2. Go to the OS Control Panel and make changes to the memory manager (be careful – try this on a machine that is not too important – like your Dad's…)!