Comp 310
Computer Systems and Organization

Lecture #17

Virtual Memory

(The Basics – Part 1)

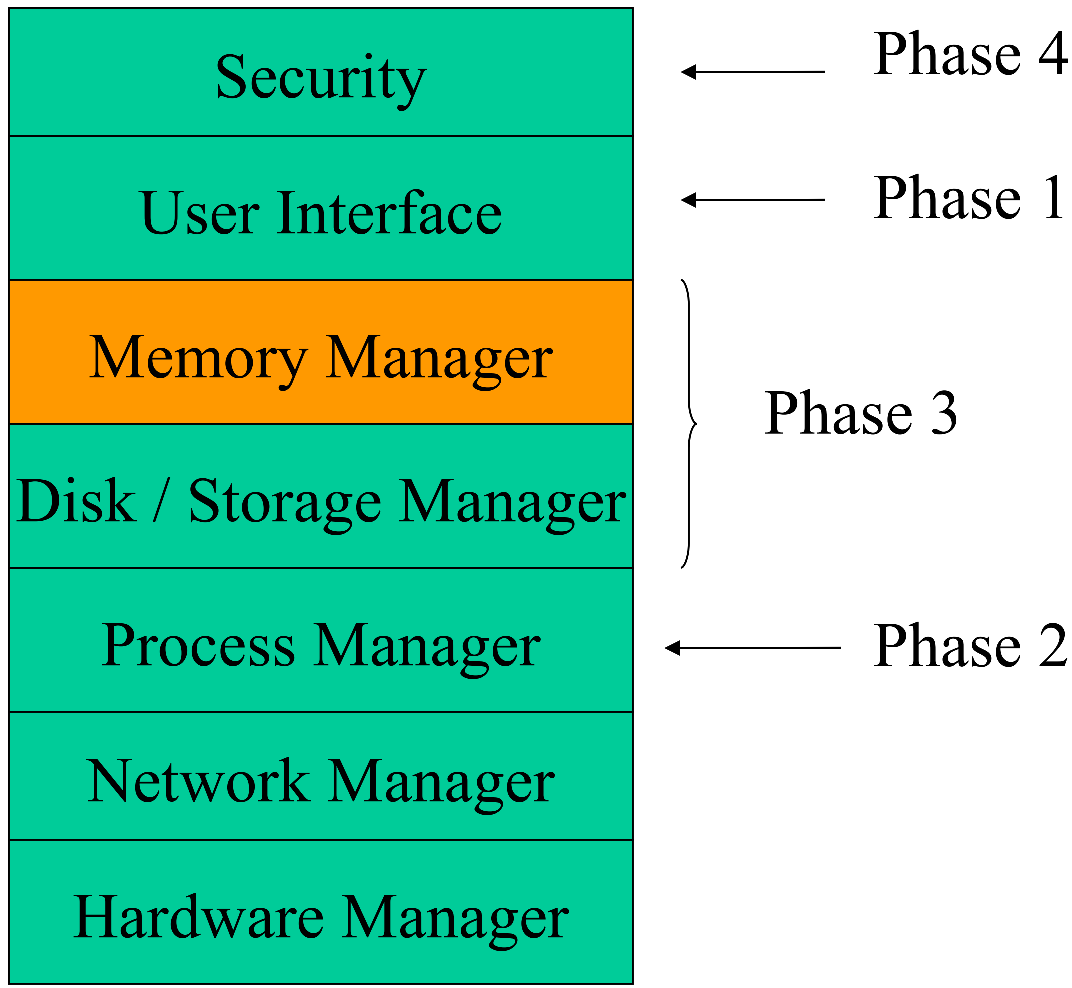
Prof. Joseph Vybihal

Announcements

- Course Evaluations – Reminder

Basic OS Architecture

(Course Table of Contents)



Part 1

The Elements of Virtual Memory

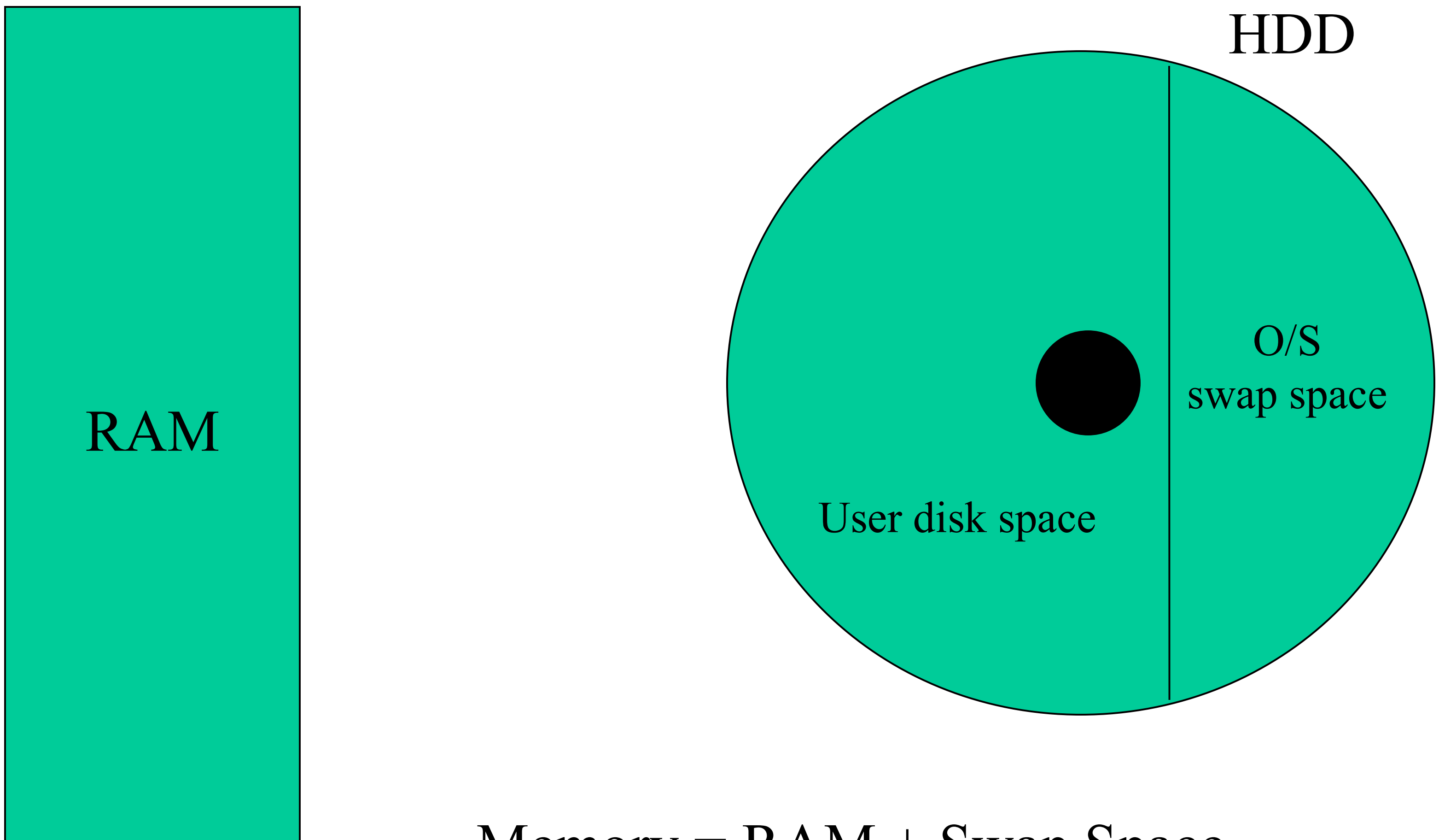
Question

- Should all of a program be in memory?
  - Error handlers rarely executed
  - Various program features may be rarely user (e.g. Thesaurus)
  - Only portion of large data structure accessed

➔A paging system subdivides programs nicely

➔A swap in/out area would maximize RAM for more processes

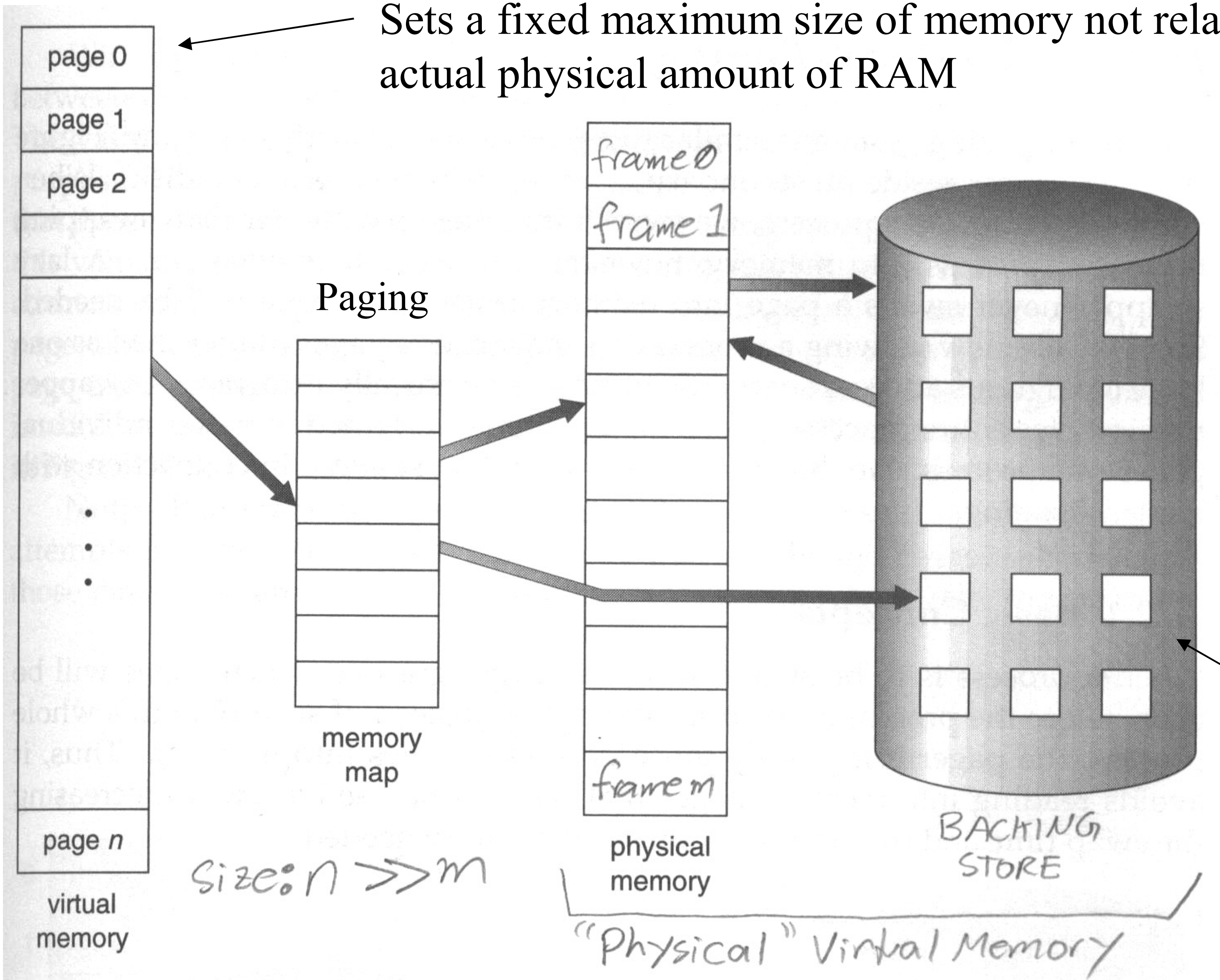
Memory



Memory = RAM + Swap Space

The Elements of Virtual Memory

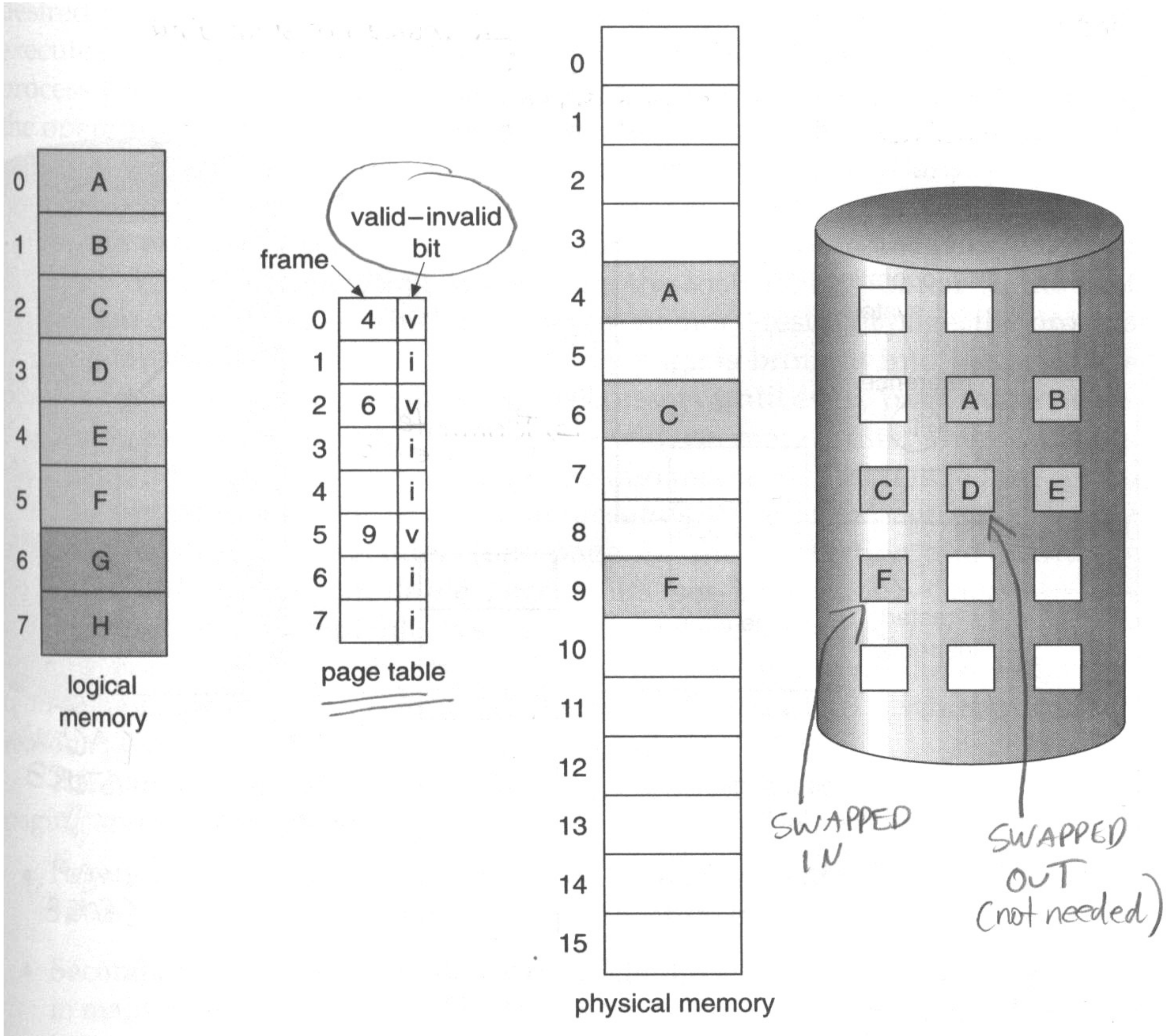
Sets a fixed maximum size of memory not related to the actual physical amount of RAM



Hard disk used to simulate the extra memory

Basic Algorithm (many exist)

1. Copy program into Backing Store (Swap Area) as pages
   - Possible arbitrary cutting of program into pages
   - Create a PCB and Page Table in RAM
2. Load a subset of pages into RAM
   - Swap out a page if there is not enough room
3. Execute program until you need a page that is not in RAM then:
   - Load that needed page
   - If necessary swap out the **most unused** page


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
SWAPPED IN
SWAPPED OUT (not needed)

Page Fault Algorithm

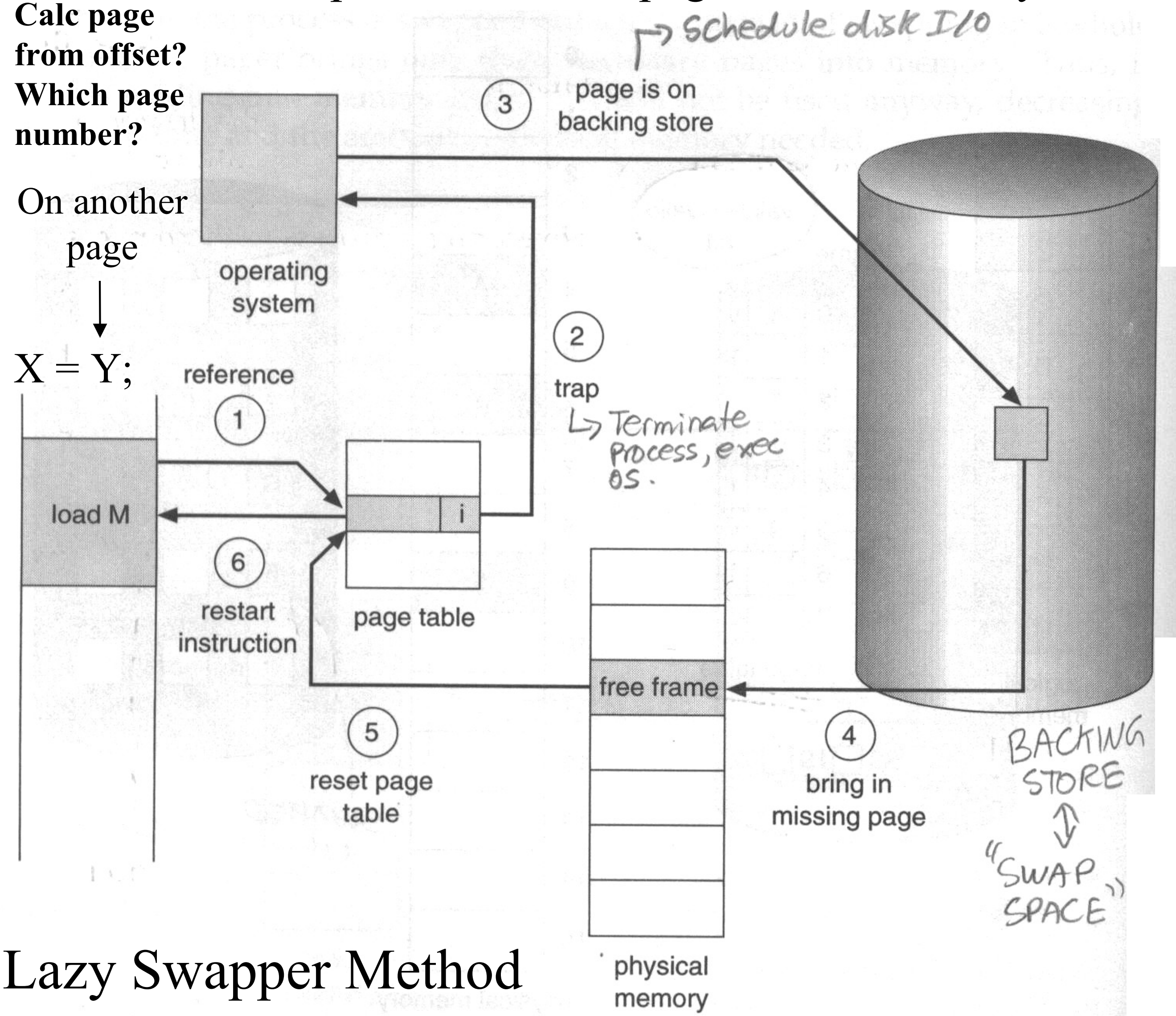
When a process needs a page that is currently not in RAM

Calc page from offset? Which page number?

On another page

↓

X = Y;



Lazy Swapper Method

<u>Process View</u>

1. Page fault
2. Task Switch
3. Wait queue
   - After load
4. Wake-up
5. Ready queue
6. Gets CPU & Redo instr.

Questions

- List the data structures in C we use to implement / manage page faults?
  - Interrupt table
  - PCB
  - Waiting queue
  - Backing store
  - Frames
  - Page table

Part 2

Performance Issues
(Demand Paging)

Page Fault Sequence

1. Interrupt issued & Task Switch
   - Trap to OS
   - Save registers at task switch
   - Determine interrupt was a page fault
   - Check for legal page reference
   - Determine location on disk

   } 50 micro-seconds

2. Issue Page fault
   - Wait queue for process ← ? milliseconds
   - Wait for device to seek and load page } 24 milliseconds
   - Issue wake-up interrupt } 1 millisecond

3. Do task switch and Interrupt issues procedure
   1. Determine wake-up
   2. Correct page table
   3. Restore registers

   } 50 micro-seconds

TOTAL > 25 milliseconds

Effective Access Time (EAT)

EAT = (1 – p) * ma + p * page_fault_time

- Where:
  - P, probability of a page fault (0<= p <= 1)
  - MA, memory access time

EAT = (1 – p) * 100 micro-sec + p(25 millisec)
= (1 – p) * 0.0001 + .025p
= 0.0001 +0.2499 p

Proportional to the page-fault-time

If p = 10% → 0.0001 + 0.2499*0.1 = 0.02509 seconds
~ 25 millisec

Swapping is expensive… solutions?

In what way do these help?

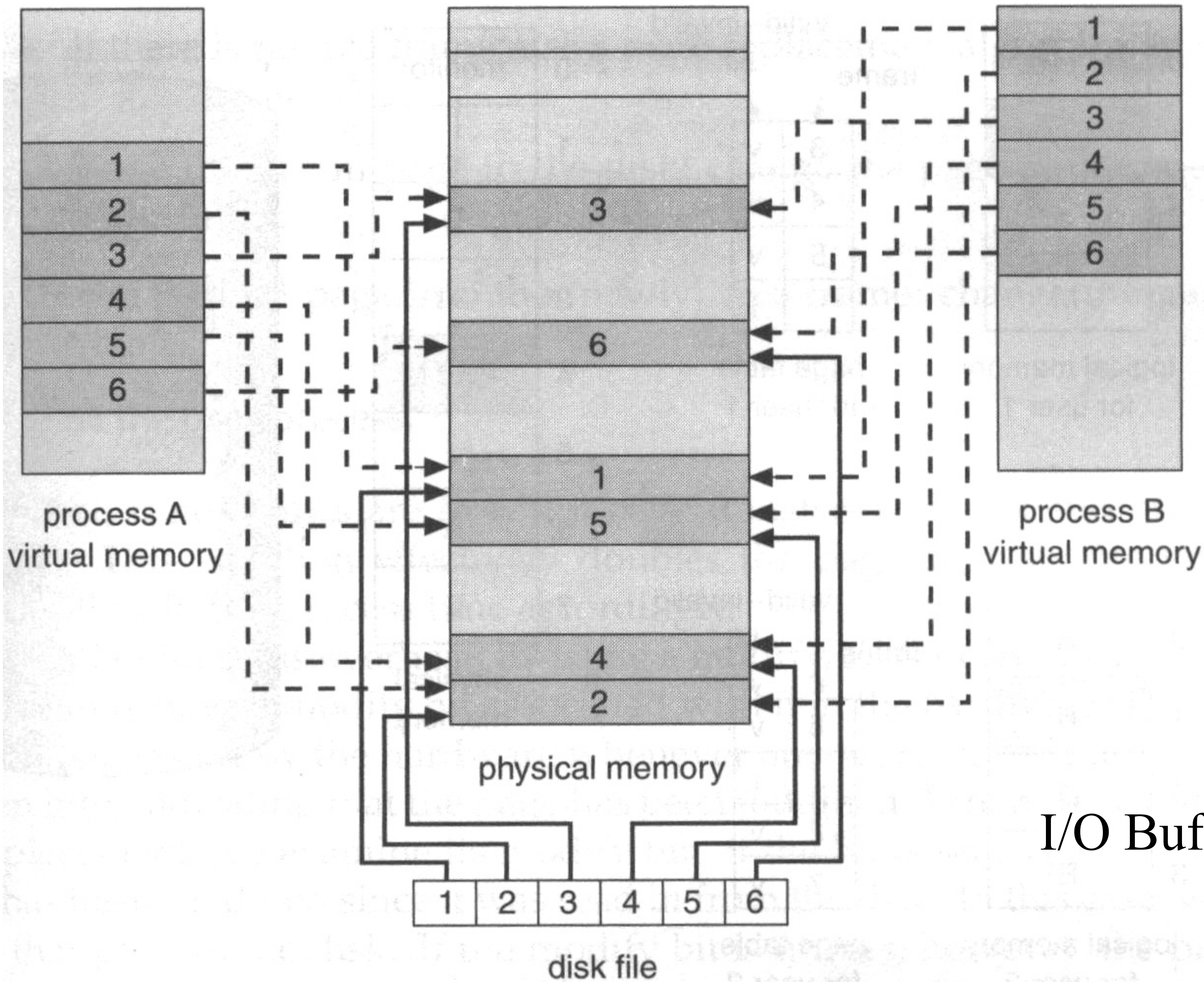
- Light-weight process (fork)
  - Shares code and data space, no load from disk
- Memory-mapped Files (buffers == Block)
  - Read/write into this memory area, buffers
- Pick a good page size for an average size:
  - Function, File?
  - Minimize swapping
- Share pages so that they don't get bumped out

How does Mapping Work?

Goal: Reduce disk I/O

- fopen
  - Copy page of file into a frame
- fread/fwrite
  - Carry out operations on frame
  - When space runs out either
    - Load in another page
    - Swap out and then load another page
- fclose
  - Write out file to disk
  - Cancel all paging tables / backing store data (if any)

Memory-mapped Files



I/O Buffers

Sharing Memory

```
#include <stdio.h>    #include <stdlib.h>    #include <fcntl.h>
#include <unistd.h>    #include <sys/types.h>    #include <sys/mman.h>
#include <sys/stat.h>    #include <errno.h>

 int main(int argc, char *argv[])    {
    int fd, offset;
    char *data;
    struct stat sbuf;

    if (argc != 2) {  fprintf(stderr, "usage: mmapdemo offset\n");  exit(1);       }

    if ((fd = open("mmapdemo.c", O_RDONLY)) == -1) {   perror("open");    exit(1);       }

    if (stat("mmapdemo.c", &sbuf) == -1) {  perror("stat");  exit(1);     }

    offset = atoi(argv[1]);
    if (offset < 0 || offset > sbuf.st_size-1) {  fprintf(stderr, "mmapdemo: offset must be in the range 0-%d\n", \
                                                   sbuf.st_size-1);   exit(1);
    }

    if ((data = mmap((caddr_t)0, sbuf.st_size, PROT_READ, MAP_SHARED, fd, 0)) == (caddr_t)(-1)) {
       perror("mmap");
       exit(1);
    }

    printf("byte at offset %d is '%c'\n", offset, data[offset]);

    return 0;
 }
```

Part 3

At Home

Things to try out

- Find the virtual memory page swap area in your OS. Change its size!!

- Web Resources (Memory mapped):
  - http://msdn2.microsoft.com/en-us/library/ms810613.aspx
  - http://en.wikipedia.org/wiki/Memory-mapped_file