MEMORY MANAGEMENT

SAMIA SHAFIQUE
UNITED INTERNATIONAL UNIVERSITY

MAIN MEMORY

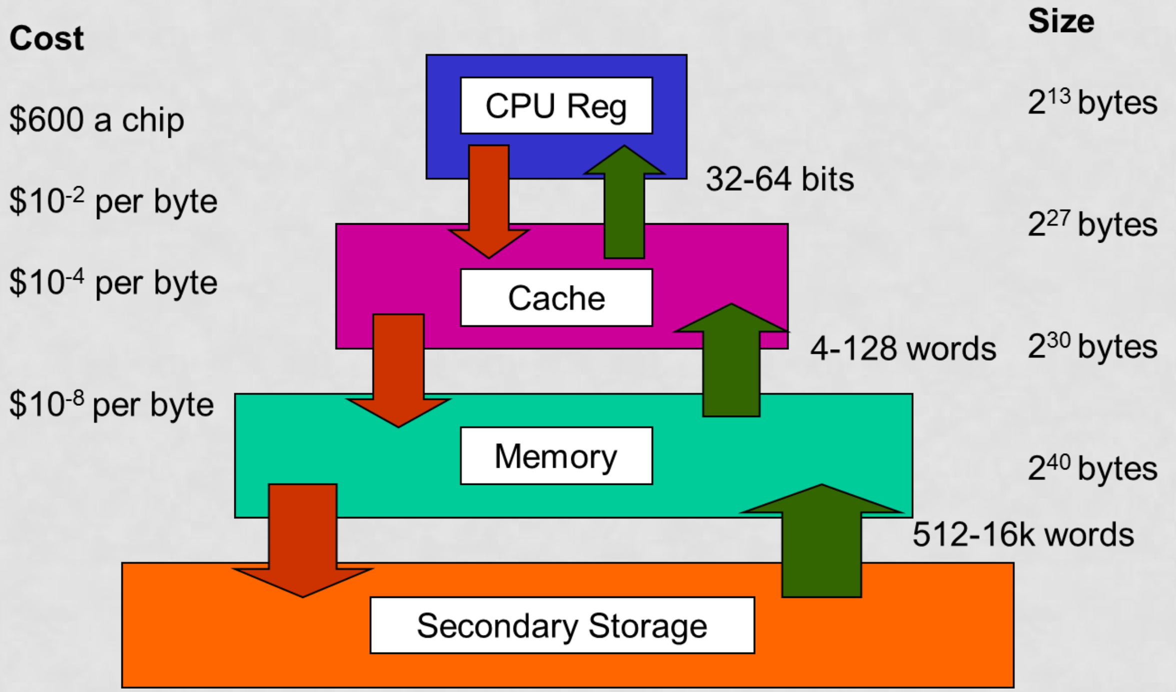
- A program resides on a disk as binary executable file
- To be executed the program must be brought into RAM
- The CPU fetches instructions from RAM according to the value of the PC
- The instruction operands may be needed to be fetched from RAM
- After execution the results may be stored back to RAM

MEMORY MANAGEMENT

- Ideally programmers want memory that is
  - private
  - large
  - fast
  - non volatile
  - Cheap

MEMORY MANAGEMENT

- But in real world…



- It is the job of the operating system to
  - **abstract** this hierarchy into a useful model
  - and then **manage** the abstraction.

MEMORY MANAGEMENT

- The part of the operating system that manages the memory hierarchy is called the *memory manager*.
  - Keep track of used memory
  - Allocate memory to processes
  - Deallocate it when they are done

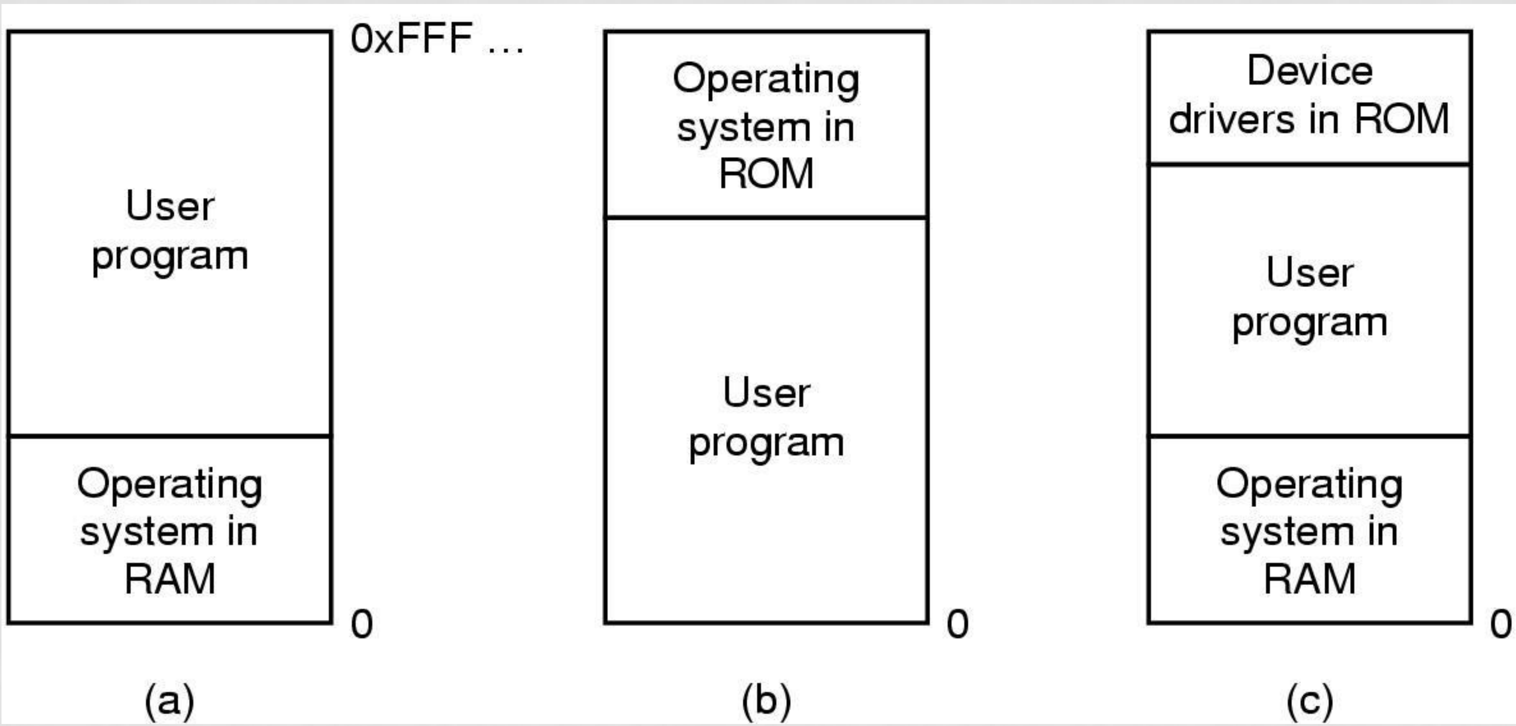
OBJECTIVE

- In this chapter we will study
  - how operating systems create abstractions from memory
  - and how they manage them.
- The focus will be on the **programmer's model** of main memory
- Memory Abstraction: the view/illusion of the memory presented to the programmer by the OS

NO MEMORY ABSTRACTION

- The simplest memory abstraction
- Every program simply saw the physical memory
  - MOV REG1, 1000
  - move the contents of physical memory location 1000 to REGISTER1
- Model of memory presented to the programmer is simply physical memory
- Not possible to have 2 running programs in memory at the same time

ONE PROGRAM AT A TIME IN MEMORY



- Three simple ways of organizing memory with an operating system and one user process

REALLY WANT TO RUN MORE THAN ONE PROGRAM

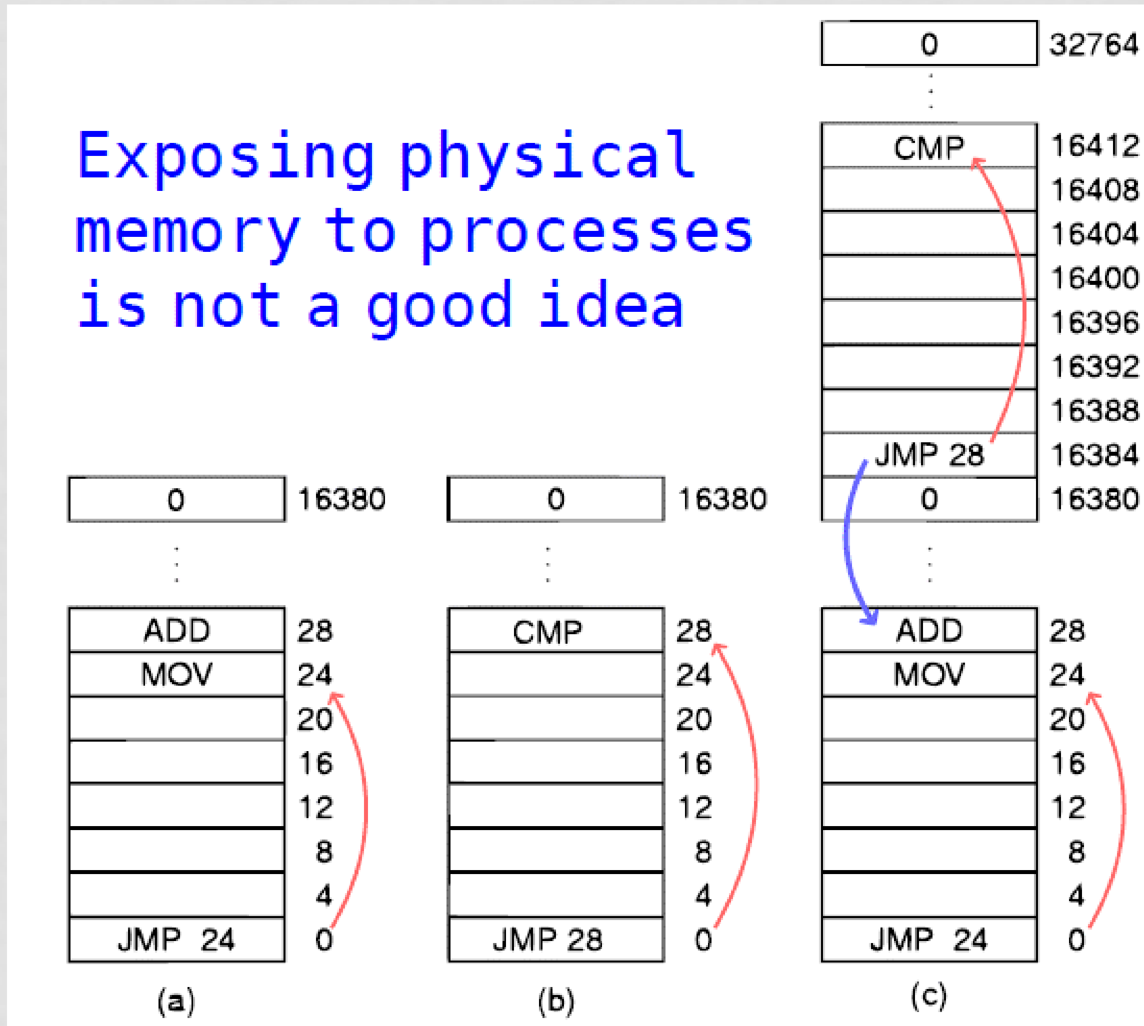
- Could swap new program into memory from disk and send old one to disk
- Not really concurrent

IBM STATIC RELOCATION IDEA

- IBM 360 –
  - divide memory into 2 KB blocks,
  - associate a 4 bit protection key with chunk.
  - Keep keys in registers.
- Put key into PSW (Program Status Word) for program
- Hardware prevents program from accessing block with another protection key

PROBLEM WITH RELOCATION

- JMP 28 in program cause program counter to move to ADD instruction in location 28.
- Program crashes



CONCLUSION

- exposing physical memory to processes has several major drawbacks.
  - if user programs can address every byte of memory, they can easily trash the operating system intentionally or by accident
  - it is difficult to have multiple programs running at once

STATIC RELOCATION

- Static relocation – load first instruction of program at address x, and add x to every subsequent address during loading
  - This is too slow and
  - Not all addresses can be modified
    - Mov register 1, 28 can’t be modified

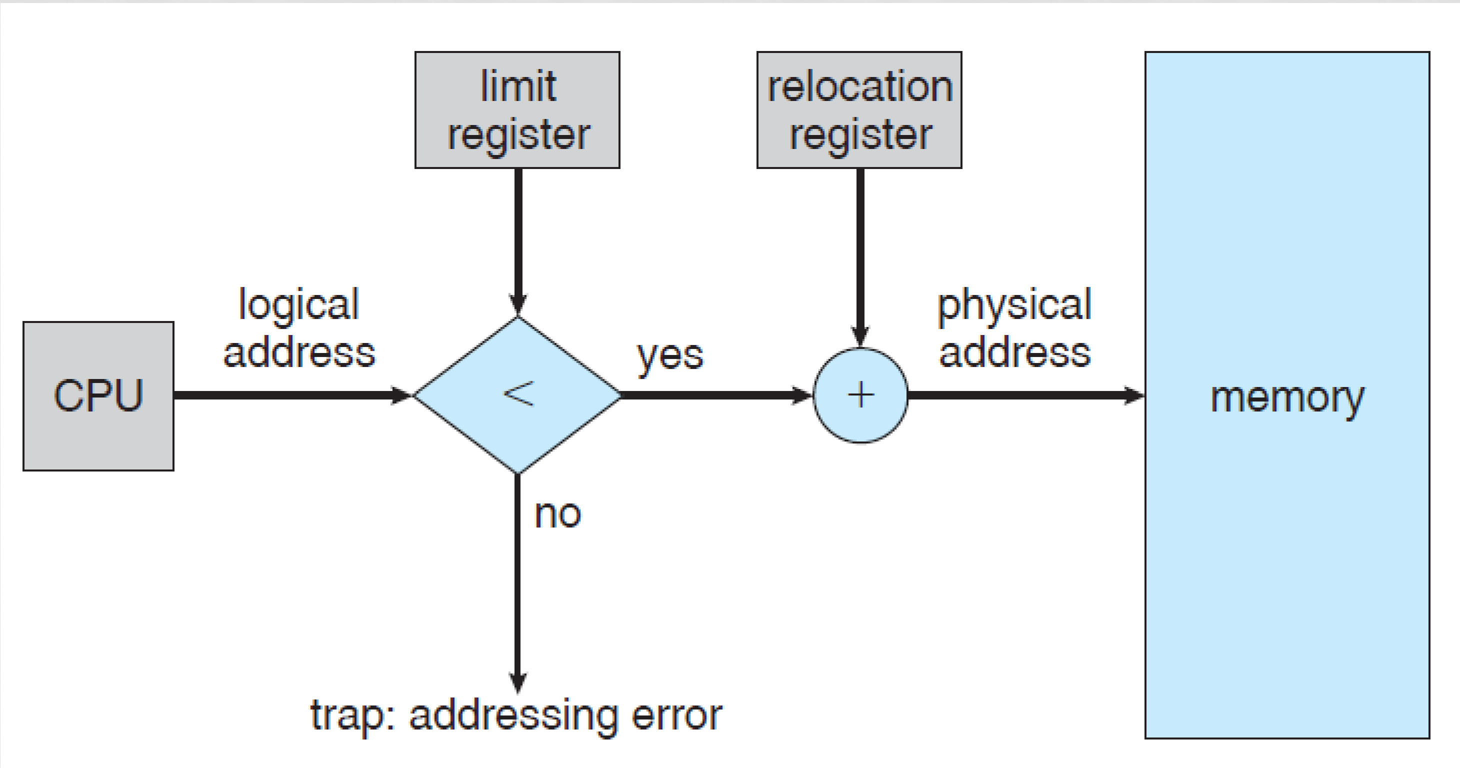
ADDRESS SPACE

- Two problems have to be solved:
  - Protection
  - Relocation
- Solution: Create abstract memory space for program to exist in
  - Each program has its own independent set of addresses
  - The addresses are different for each program
  - Call it the address space of the program

BASE AND LIMIT REGISTERS

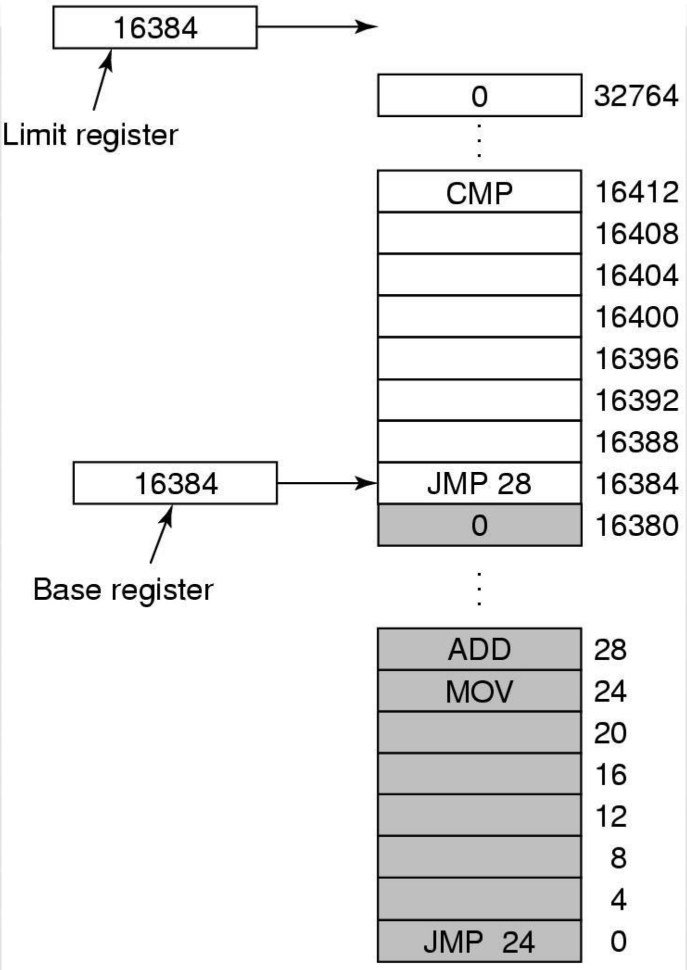
- A form of dynamic relocation
- Base contains beginning address of program
- Limit contains length of program
- Program references memory, adds base address to address generated by process. Checks to see if address is larger then limit. If so, generates fault

BASE AND LIMIT REGISTERS



BASE AND LIMIT REGISTERS

- Add 16384 to JMP 28.
- Hardware adds 16384 to 28 resulting in JMP 16412



(c)

BASE AND LIMIT REGISTERS

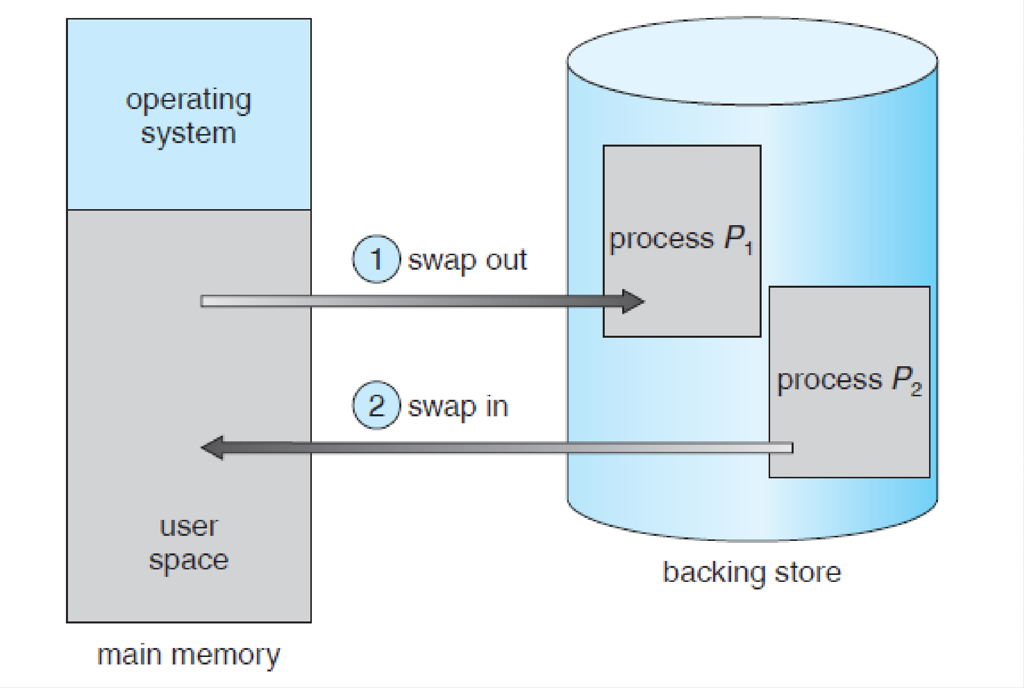
- Disadvantage-addition and comparison have to be done on every instruction

HOW TO RUN MORE PROGRAMS THEN FIT IN MAIN MEMORY AT ONCE

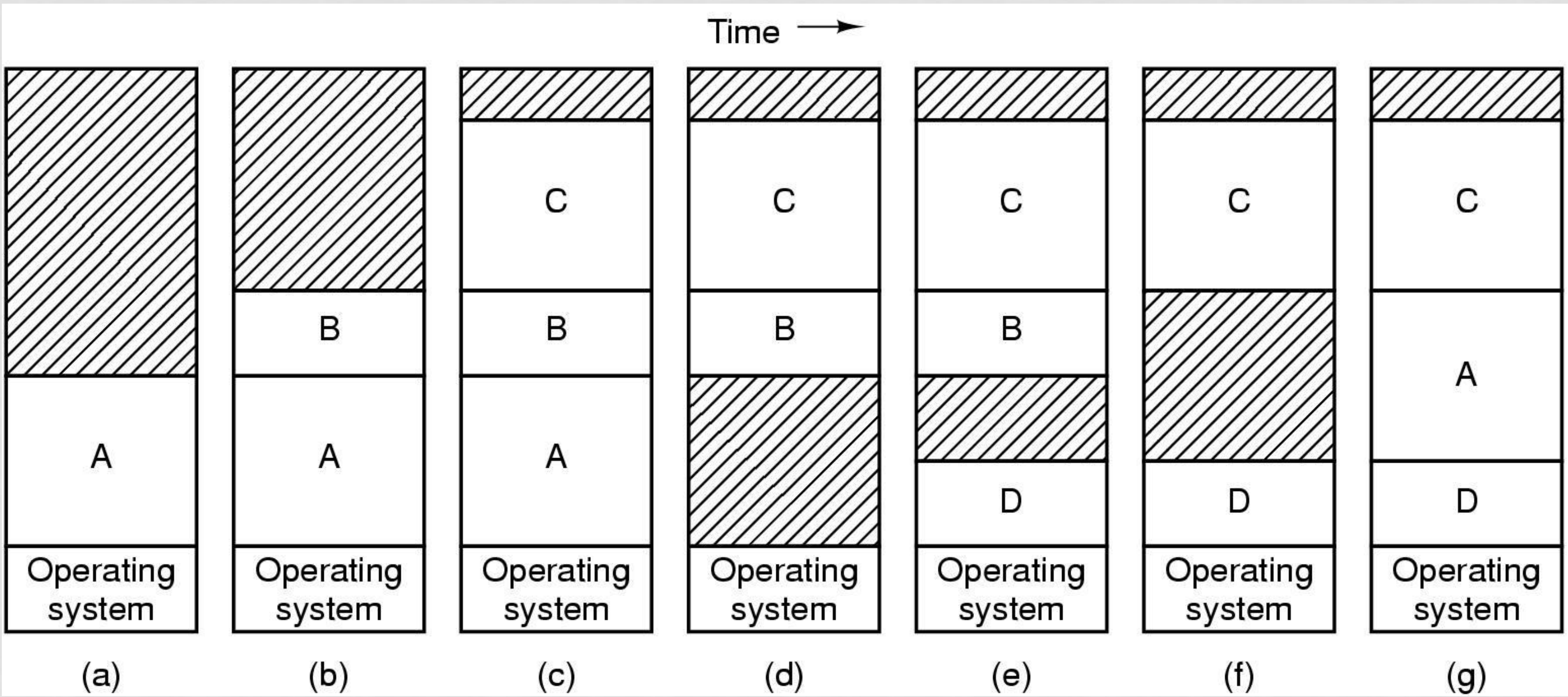
- Can't keep all processes in main memory
  - Too many (hundreds)
  - Too big (eg 200MB program)
- Two approaches
  - Swap-bring program in and run it for a while
  - Virtual memory – allow program to run even if only part of it is in main memory

SWAPPING

- Bringing in each process in its entirety,
- running it for a while
- then putting it back on the disk



SWAPPING



SWAPPING

- Memory allocation changes as
  - processes come into memory
  - leave memory
- Shaded regions are unused memory
- Hole – block of available memory; holes of various size are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it
- Operating system maintains information about:
  a) allocated partitions b) free partitions (hole)

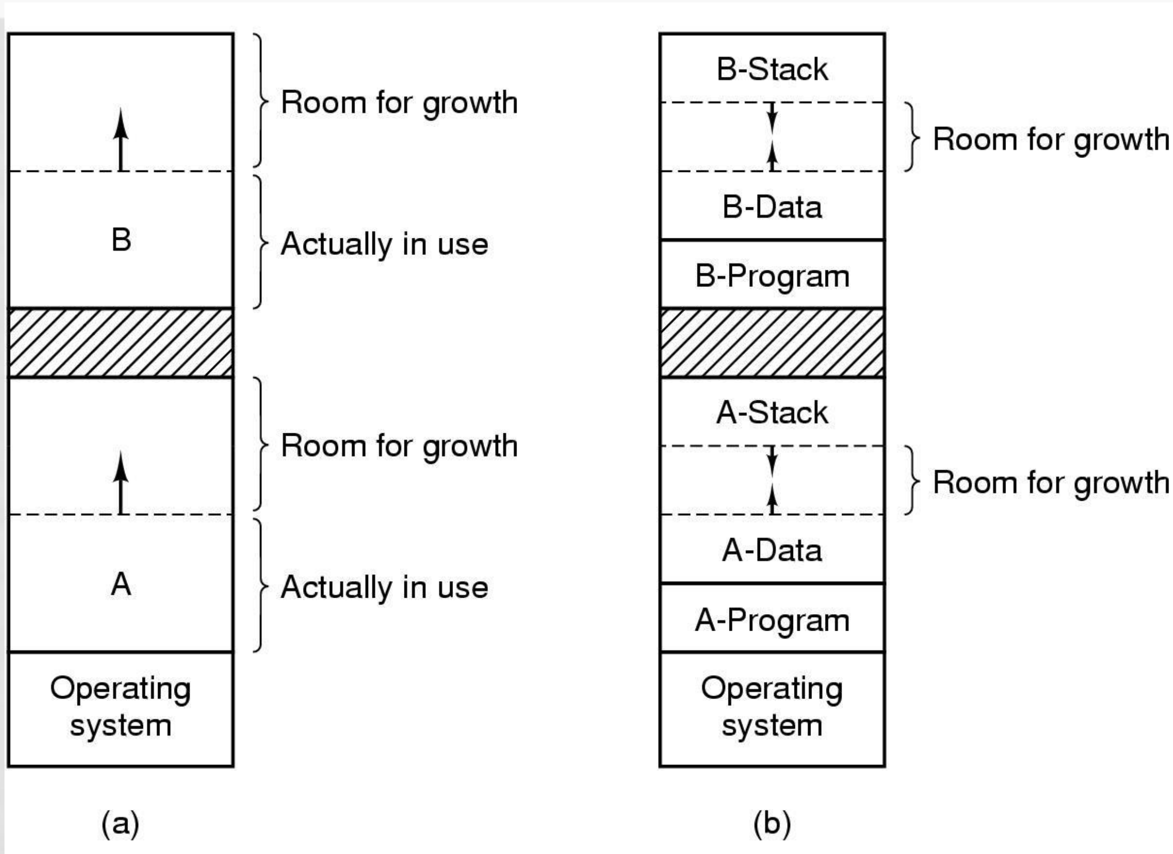
SWAPPING

- When swapping creates multiple holes in memory, it is possible to combine them all into one big one by moving all the processes downward as far as possible.
- This technique is known as memory compaction

PROGRAMS GROW AS THEY EXECUTE

- Stack (return addresses and local variables)
- Data segment (heap for variables which are dynamically allocated and released)
- Good idea to allocate extra memory for both
- When program goes to disk, don't bring holes along with it!!

2 WAYS TO ALLOCATE SPACE FOR GROWTH

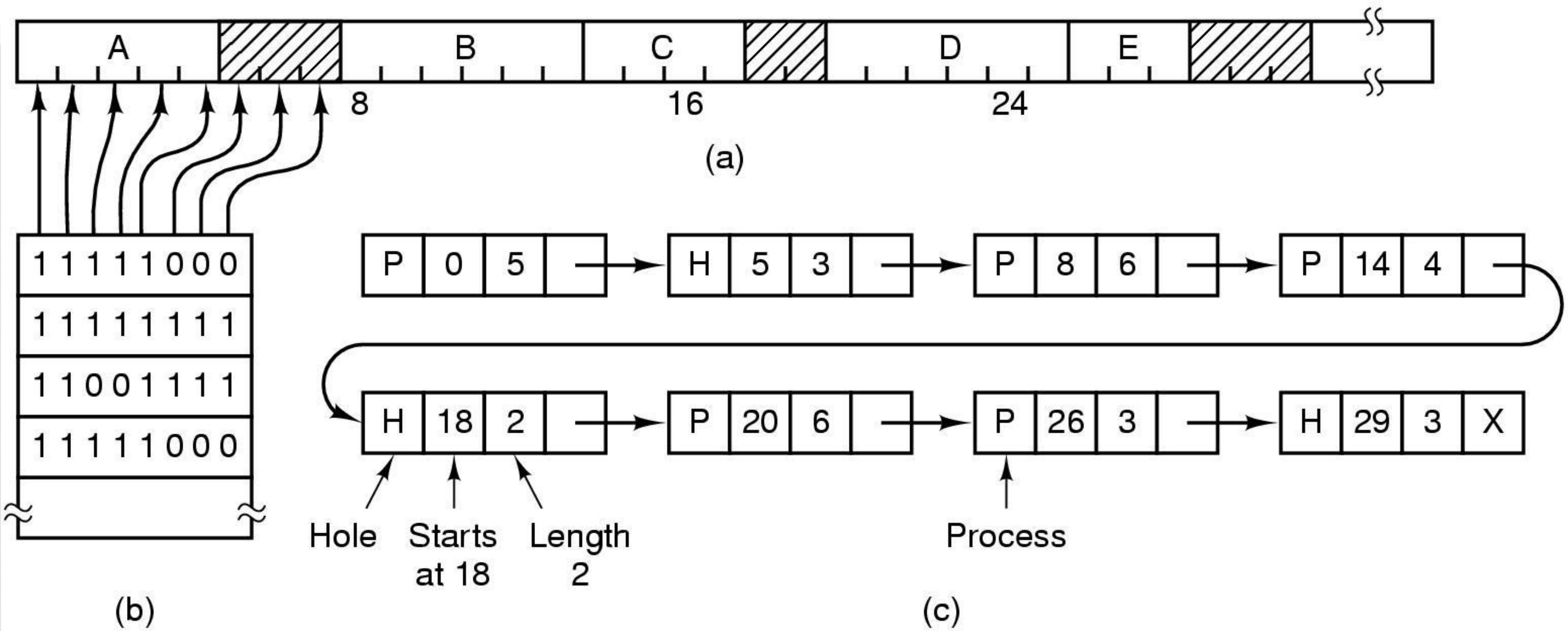


- Allocating space for growing data segment
- Allocating space for growing stack & data segment

MANAGING FREE MEMORY

- Operating system maintains information about
  - Allocated memory
  - Free memory (holes)
- Two techniques to keep track of memory usage
  - Bitmaps
  - Linked lists

BITMAPS

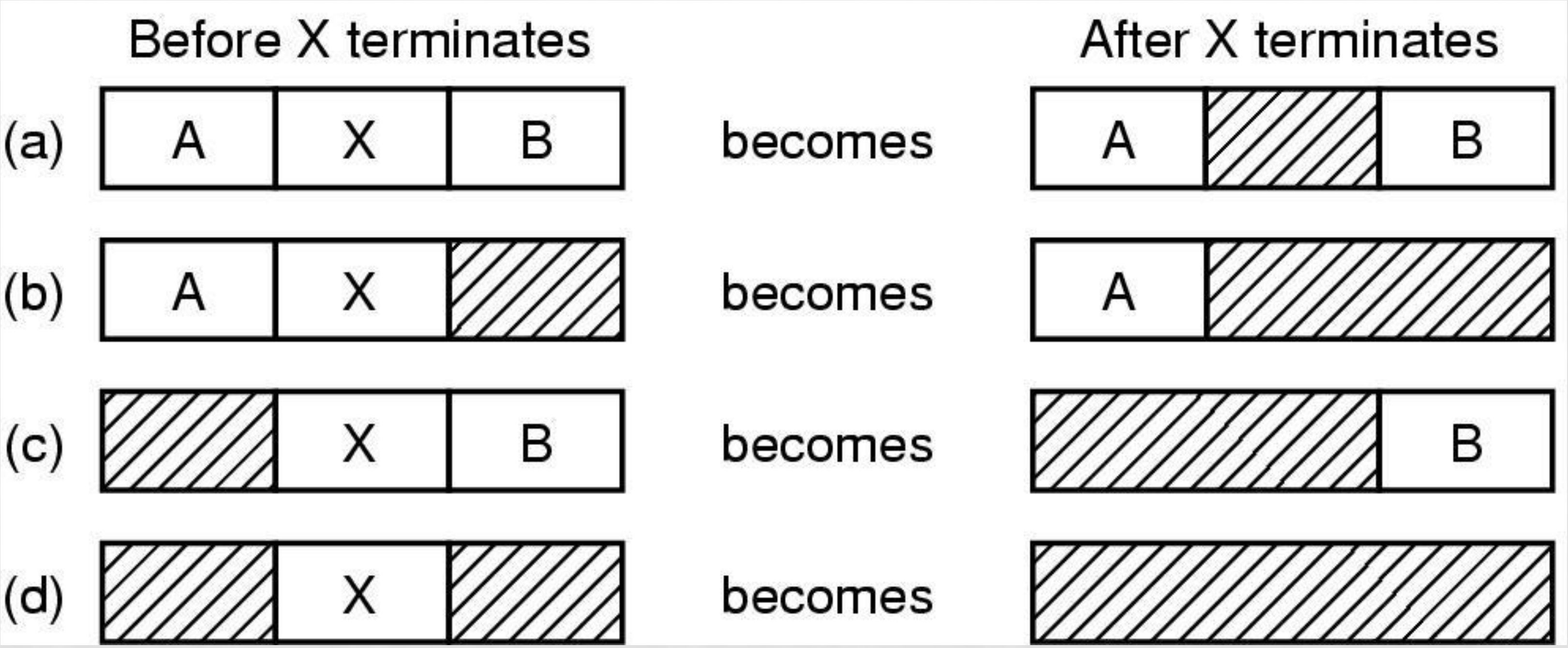


(a) Picture of memory
(b) Each bit in bitmap corresponds to a unit of storage (eg. bytes) in memory
(c) Linked list: P - process, H - hole

BITMAPS

- The good – compact way to keep track of memory
- The bad – need to search memory for k consecutive zeros to bring in a file k units long
- Units can be bits or bytes or …….

LINKED LISTS



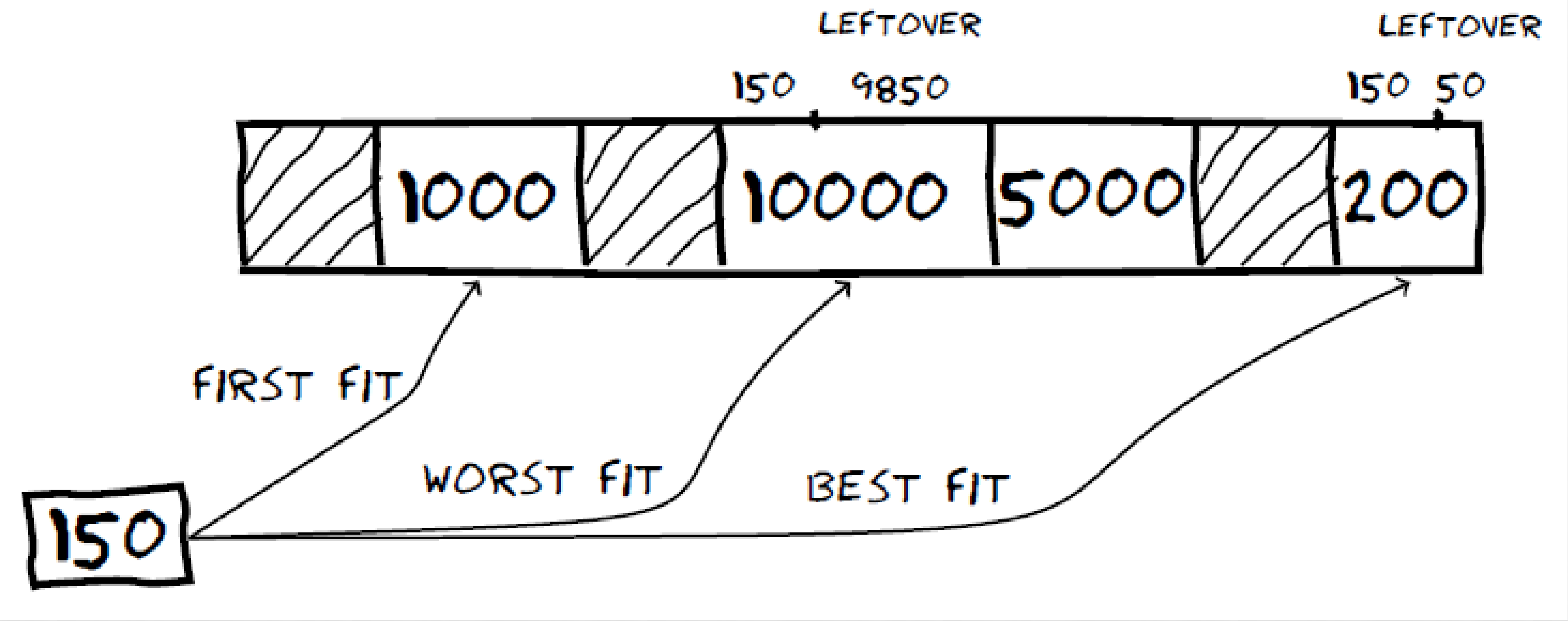
- Four neighbor combinations for terminating process, X.

LINKED LISTS

- Might want to use doubly linked lists to merge holes more easily
- Algorithms to fill in the holes in memory
  - Next fit
  - Best fit
  - Worst fit
  - Quick fit

LINKED LISTS

- Memory allocation algorithms : First fit, next fit, best fit, worst fit, quick fit



THE FITS

- First fit-fast
- Next fit-starts search wherever it is
  - Slightly worse
- Best fit-smallest hole that fits
  - Slower, results in a bunch of small holes (i.e. worse algorithm)
- Worst fit-largest hole that fits
  - Not good (simulation results)
- Quick fit- keep list of common sizes
  - Quick, but can’t find neighbors to merge with

THE FITS

- Conclusion – the fits couldn't out-smart the unknowable distribution of hole sizes.
- The extra work to deal with something which you can't predict failed to produce good results