Memory management

Knut Omang
Ifi/Oracle
13 Oct, 2010

(with slides from V. Goebel, C. Griwodz (Ifi/UiO), P. Halvorsen (Ifi/UiO), K. Li (Princeton), A. Tanenbaum (VU Amsterdam), and M. van Steen (VU Amsterdam))

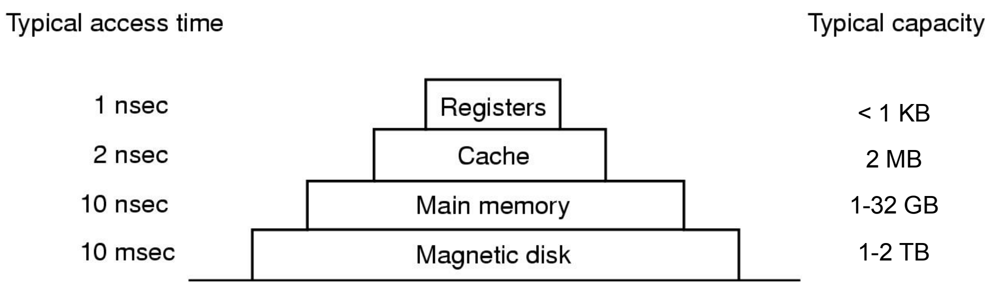
Today

- Basic memory management
- Swapping
- Page as memory unit
- Segmentation
- Virtual memory
- Page/segment allocation implementation

Memory Management

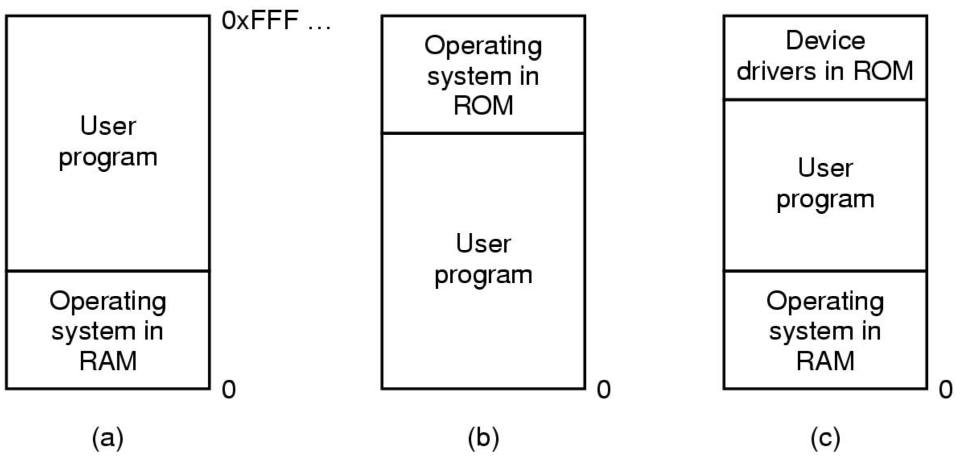
- Ideally programmers want memory that is
  - large
  - fast
  - non volatile
- Memory hierarchy
  - small amount of fast, expensive memory – cache
  - some medium-speed, medium price main memory
  - gigabytes of slow, cheap disk storage
- Memory manager handles the memory hierarchy

Computer Hardware Review



- Typical memory hierarchy
  - numbers shown are rough approximations

Models for Memory Management with no memory abstraction support



- Physical addresses used directly
- still need to organize

No memory abstraction - implications

- Only one program can easily be present at a time!
- If another program needs to run, the entire state must be saved to disk
- No protection...

Multiprogramming

- Processes have to wait for I/O
- Goal
  - Do other work while a process waits
  - Give CPU to another process

- Processes may be concurrently ready
- So
  - If I/O waiting probability for all processes is *p*
  - Probable CPU utilization can be estimated as

CPU utilization = 1 – p^n

Multiprogramming

- Arrival and work requirements of 4 jobs
- CPU utilization for 1-4 jobs with 80% I/O wait

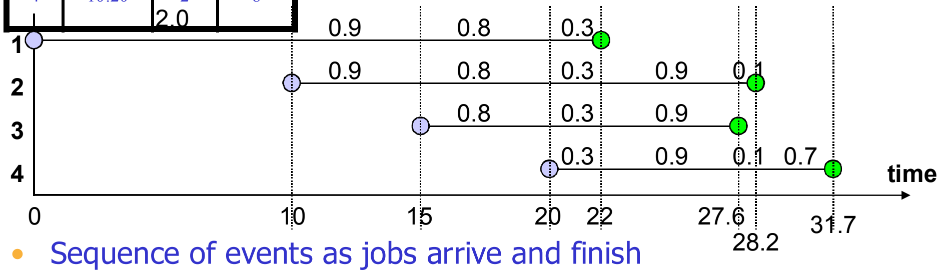
Job#	Arrival time	CPU use time	IO wait time
1	10:00	4	16
2	10:10	3	12
3	10:15	2	8
4	10:20	2	8

Remaining CPU time

1	[illegible]
2	[illegible]
3	[illegible]
4	[illegible]

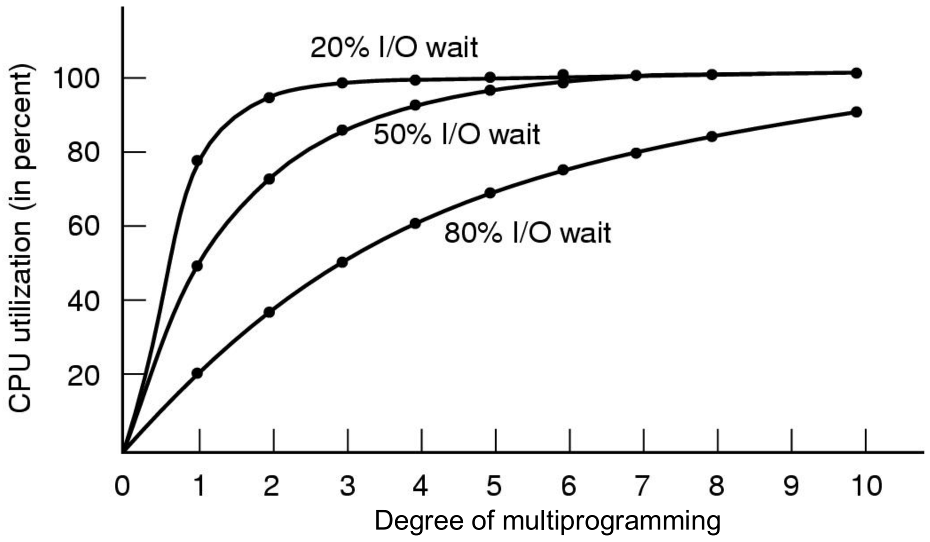
processors

%	1	2	3	4
CPU idle	80	64	51	41
CPU busy	20	36	49	59
CPU per process	20	18	16	15



- Sequence of events as jobs arrive and finish
  - Note numbers show amount of CPU time jobs get each interval

Multiprogramming



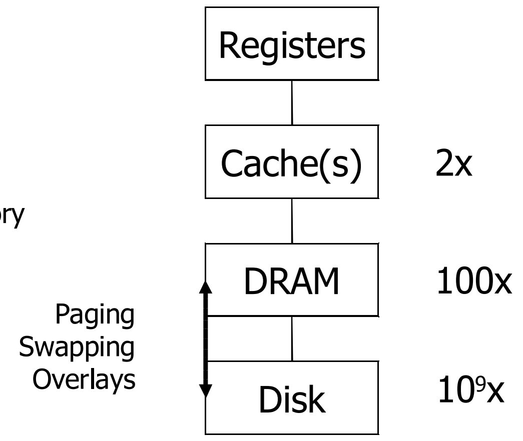
- CPU utilization as a function of number of processes in memory

Multiprogramming

- Several programs
  - Concurrently loaded into memory
  - OS must arrange memory sharing
  - Memory partitioning
- Memory
  - Needed for different tasks within a process
  - Shared among processes
  - Process memory demand may change over time
- Use of secondary storage
  - Move (parts of) blocking processes from memory
  - Higher degree of multiprogramming possible
  - Makes sense if processes block for long times

Memory Management for Multiprogramming

- Process may not be entirely in memory
- Reasons
  - Other processes use memory
    - Their turn
    - Higher priority
    - Process is waiting for I/O
  - Too big
    - For its share
    - For entire available memory
- Approaches
  - Swapping
  - Paging
  - Overlays



Memory Management for Multiprogramming

- Swapping
  - Remove a process from memory
    - With all of its state and data
    - Store it on a secondary medium
      - Disk, Flash RAM, other slow RAM, historically also Tape
- Paging
  - Remove part of a process from memory
    - Store it on a secondary medium
    - Sizes of such parts are fixed
    - Page size
- Overlays
  - Manually replace parts of code and data
    - Programmer’s rather than OS’s work
    - Only for very old and memory-scarce systems

Memory Management Techniques
- How to assign memory to processes

- Memory partitioning:
  - Fixed partitioning
  - Dynamic partitioning
  - Simple paging
  - Simple segmentation
  - Virtual memory paging
  - Virtual memory segmentation

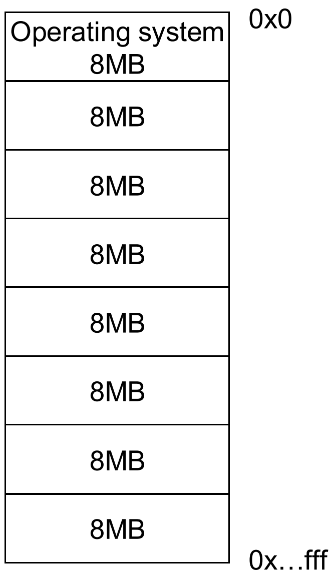
Fixed Partitioning

- Divide memory
  - Into static partitions
  - At system initialization time (boot or earlier)

- Advantages
  - Very easy to implement
  - Can support swapping process in and out

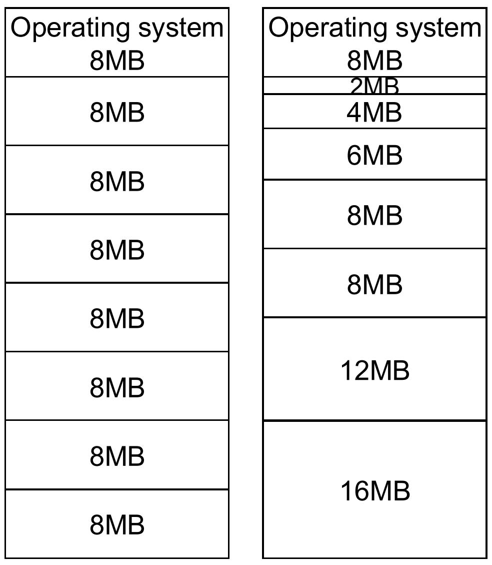
Fixed Partitioning

- Two fixed partitioning schemes
  - Equal-size partitions
  - Unequal-size partitions
- Equal-size partitions
  - Big programs can not be executed
    - Unless program parts are loaded from disk
  - Small programs use entire partition
    - A problem called "internal fragmentation"



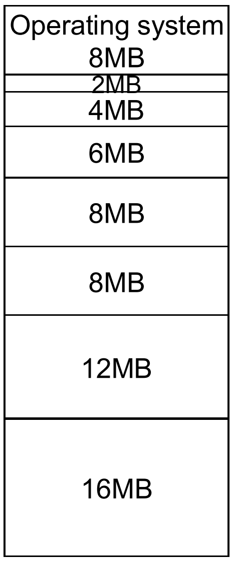
Fixed Partitioning

- Two fixed partitioning schemes
  - Equal-size partitions
  - Unequal-size partitions
- Unequal-size partitions
  - Bigger programs can be loaded at once
  - Smaller programs can lead to less internal fragmentation
  - Advantages require assignment of jobs to partitions



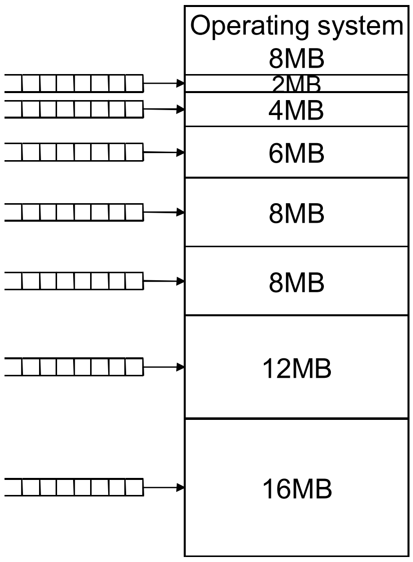
Fixed Partitioning

- Approach
  - Has been used in mainframes
  - Uses the term job for a running program
  - Jobs run as batch jobs
  - Jobs are taken from a queue of pending jobs
- Problem with unequal partitions
  - Choosing a job for a partition



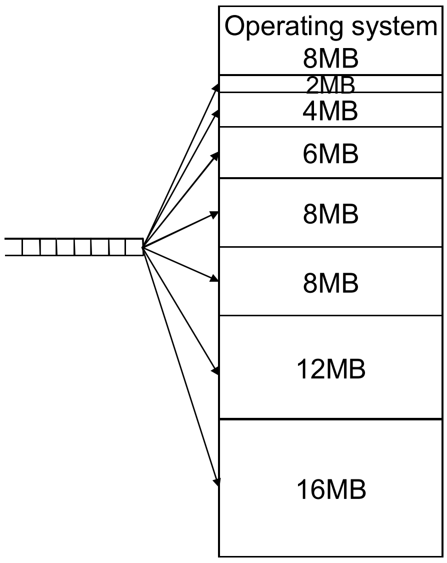
Fixed Partitioning

- One queue per partition
  - Internal fragmentation is minimal
  - Jobs wait although sufficiently large partitions are available



Fixed Partitioning

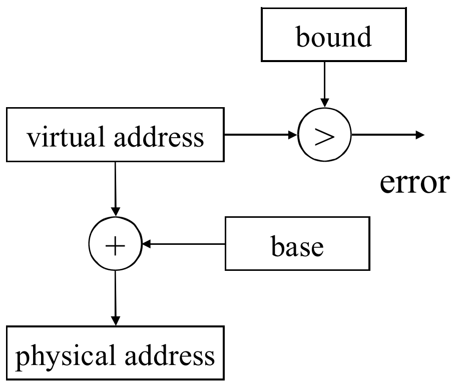
- Single queue
  - Jobs are put into next sufficiently large partition
  - Waiting time is reduced
  - Internal fragmentation is bigger
  - A swapping mechanism can reduce internal fragmentation
    - Move a job to another partition



Problems: Relocation and Protection

- Cannot be sure where program will be loaded in memory
  - address locations of variables, code routines cannot be absolute
  - must keep a program out of other processes' partitions
- Base and limit values: Simplest form of virtual memory (translate: virt --> phys)
  - address locations added to base value to map to phys. addr
  - address locations larger than limit value is an error

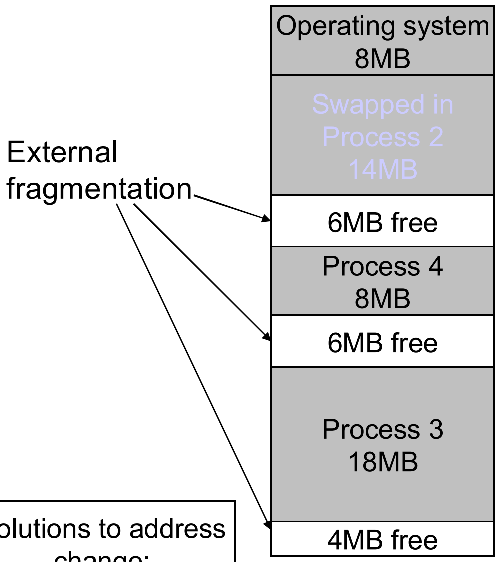
2 Registers: Base and Bound



- Built in Cray-1
- A program can only access physical memory in [base, base+bound]
- On a context switch: save/restore base, bound registers
- Pros: Simple
- Cons: fragmentation, hard to share, and difficult to use disks

Dynamic Partitioning

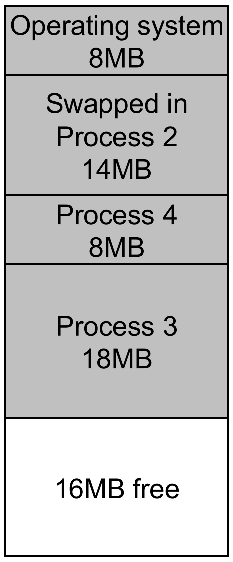
- Divide memory
  - Partitions are created dynamically for jobs
  - Removed after jobs are finished
- External fragmentation
  - Problem increases with system running time
  - Occurs with swapping as well
  - Addresses of process 2 changed



Solutions to address change:
Address Translation

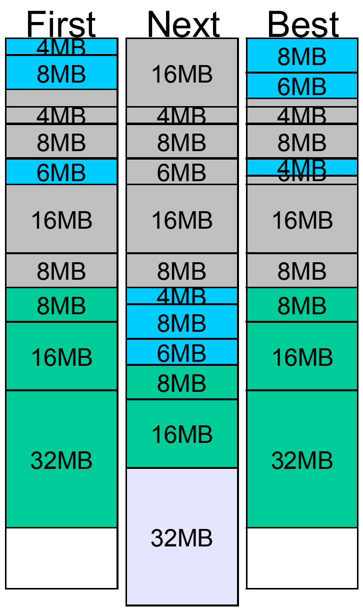
Dynamic Partitioning

- Reduce external fragmentation
  - Compaction
- Compaction
  - Takes time
  - Consumes processing resources
- Reduce compaction need
  - Placement algorithms



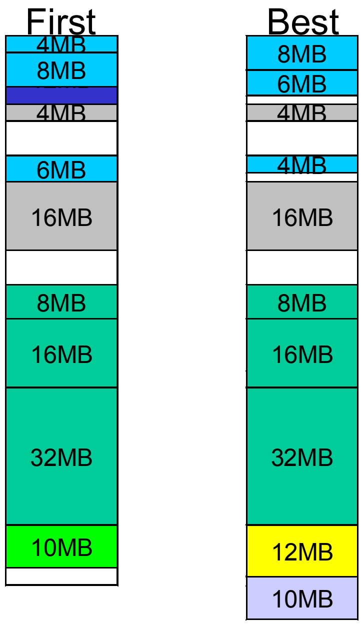
Dynamic Partitioning: Placement Algorithms

- Use most suitable partition for process
- Typical algorithms
  - First fit
  - Next fit
  - Best fit



Dynamic Partitioning: Placement Algorithms

- Use most suitable partition for process
- Typical algorithms
  - First fit
  - Next fit
  - Best fit



Dynamic Partitioning: Placement Algorithms

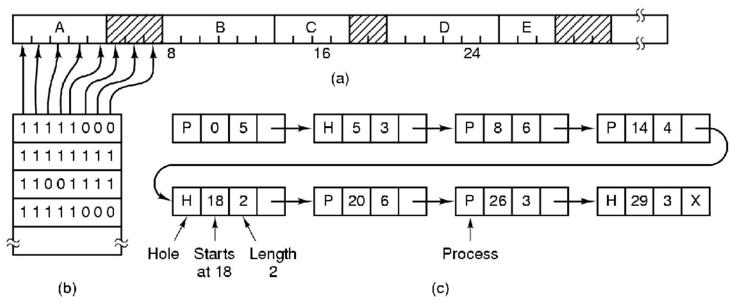
- Comparison of First fit, Next fit and Best fit
- Example is naturally artificial
  - First fit
    - Simplest, fastest of the three
    - Typically the best of the three
  - Next fit
    - Typically slightly worse than first fit
    - Problems with large segments
  - Best fit
    - Slowest
    - Creates lots of small free blocks
    - Therefore typically worst

Memory management: bookkeeping

Two main strategies:

- Bitmaps
  - Bit indicate free/allocated
- Using linked lists
  - Keep list(s) of free/allocated segments

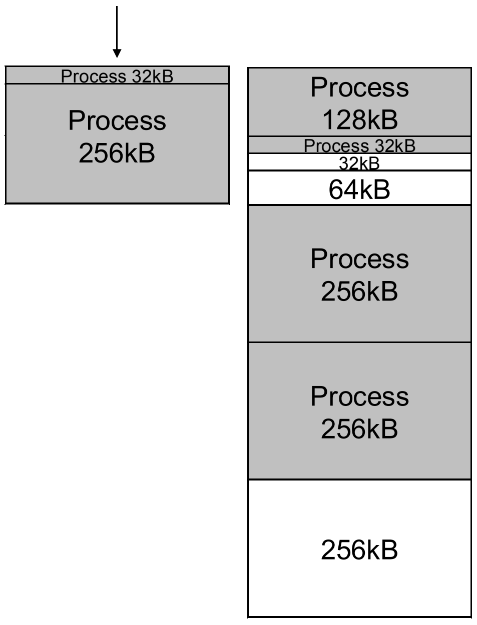
Memory Management: bitmaps/lists



- Part of memory with 5 processes, 3 holes
  - tick marks show allocation units
  - shaded regions are free
- Corresponding bit map
- Same information as a list

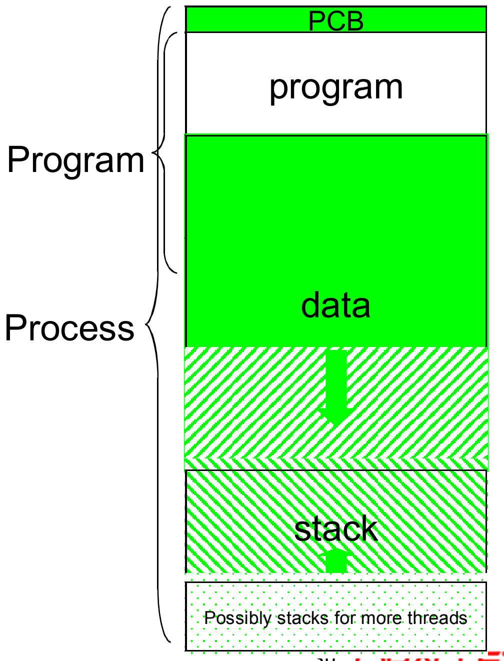
Buddy System

- Mix of fixed and dynamic partitioning
  - Partitions have sizes 2^k, $L \leq k \leq U$
- Maintain a list of holes with sizes
- Assign a process
  - Find smallest k so that process fits into 2^k
  - Find a hole of size 2^k
  - If not available, split smallest hole larger than 2^k
    - Split recursively into halves until two holes have size 2^k



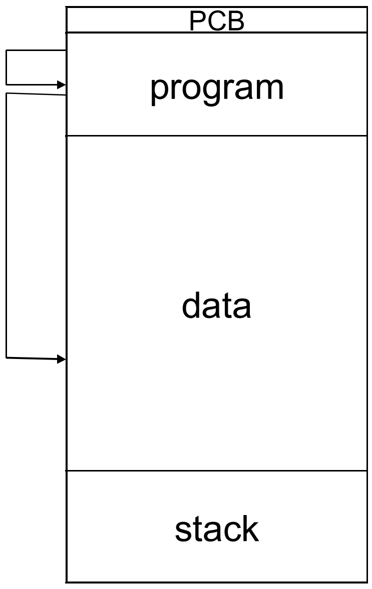
Memory use within a process

- Memory needs of known size
  - Program code
  - Global variables
- Memory needs of unknown size
  - Dynamically allocated memory
  - Stack
    - Several in multithreaded programs



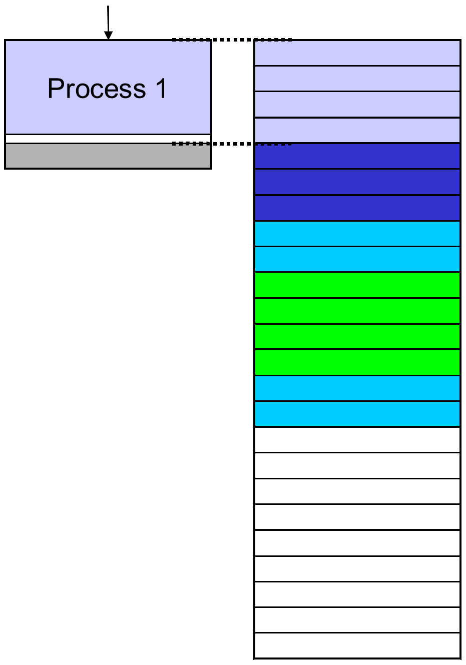
Memory Addressing

- Addressing in memory
  - Addressing needs are determined during programming
  - Must work independently of position in memory
  - Actual physical address are not known
  - Leave enough room for growth (but not too much!)

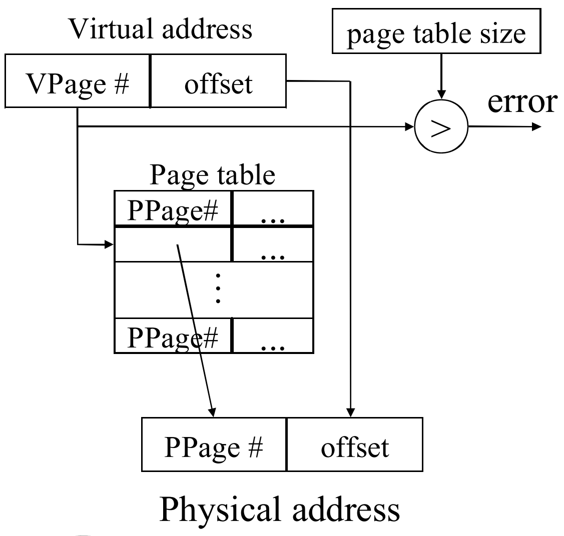


Paging

- Paging
  - Equal lengths
  - Determined by processor
  - One page moved into one memory frame
- Process is loaded into several frames
  - Not necessarily consecutive
- No external fragmentation
- Little internal fragmentation
  - Depends on frame size



Paging

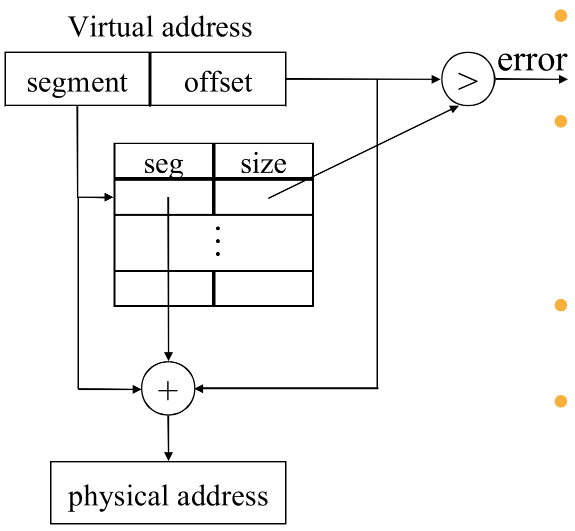


- Use a page table to translate
- Various bits in each entry
- Context switch: similar to the segmentation scheme
- What should be the page size?
- Pros: simple allocation, easy to share
- Cons: big page table and cannot deal with holes easily

Segmentation

- Segmentation
  - Different lengths
  - Determined by programmer
  - Memory frames
- Programmer (or compiler toolchain) organizes program in parts
  - Move control
  - Needs awareness of possible segment size limits
- Pros and Cons
  - Principle as in dynamic partitioning
  - No internal fragmentation
  - Less external fragmentation because on average smaller segments

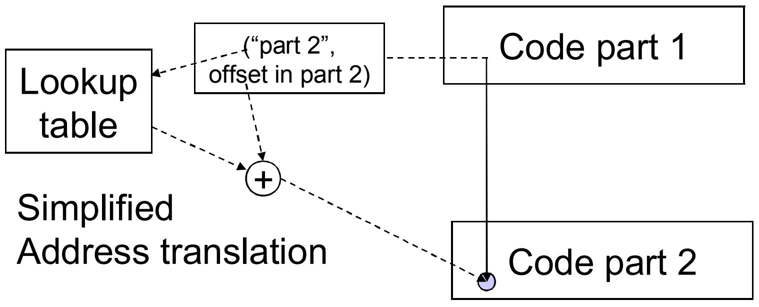
Segmentation



- Have a table of (seg, size)
- Protection: each entry has
  - (nil, read, write)
- On a context switch: save/restore the table or a pointer to the table in kernel memory
- Pros: Efficient, easy to share
- Cons: Complex management and fragmentation within a segment

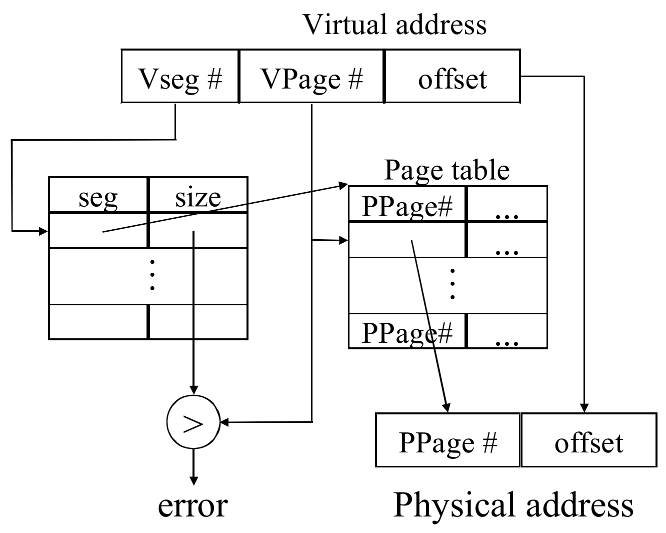
Paging

- Typical for paging and swapping
  - Address translation
  - At execution time
  - With processor support
- Simple paging and segmentation
  - Without virtual memory and protection
  - Can be implemented
    - by address rewriting at load time
    - by jump tables setup at load time



Simplified Address translation

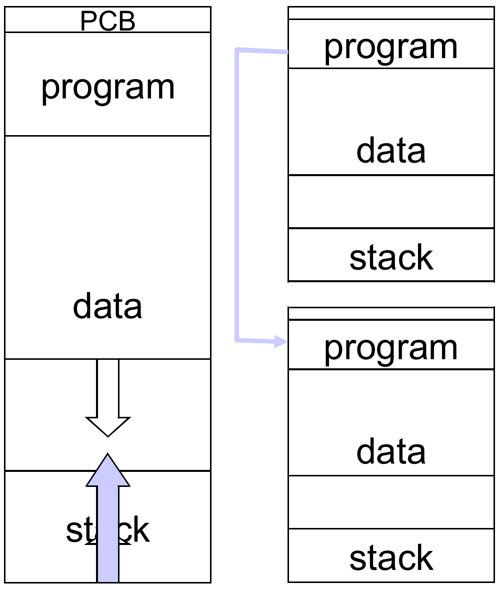
Segmentation with paging and virtual address space



Other needs (protection)

- Protection of process from itself
  - (stack grows into heap)
- Protection of processes from each other
  - (write to other process)

Solutions to protection:
Address Translation



Why Address Translation and Virtual Memory?

- Use secondary storage
  - Extend expensive DRAM with reasonable performance
- Protection
  - Programs do not step over each other and communicate with each other require explicit IPC operations
- Convenience
  - Flat address space
  - Programs share same view of the world
  - Programs/program parts can be moved

Page Replacement Algorithms

- Page fault forces choice
  - which page must be removed
  - make room for incoming page
- Modified page must first be saved
  - unmodified just overwritten
- Better not to choose an often used page
  - will probably need to be brought back in soon

Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future
  - Optimal but unrealizable
- Estimate by …
  - logging page use on previous runs of process
  - although this is impractical

Not Recently Used (NRU)

- Two status bits associated with each page:
  R ⬭ page referenced (read or written)
  M ⬭ page modified (written)
- Pages belong to one of four set of pages according to the status bits:
  - Class 0: not referenced, not modified (R=0, M=0)
  - Class 1: not referenced, modified (R=0, M=1)
  - Class 2: referenced, not modified (R=1, M=0)
  - Class 3: referenced, modified (R=1, M=1)
- NRU removes a page at random
  - from lowest numbered, non-empty class
- Low overhead

Not Recently Used Page Replacement Algorithm

- Each page has Reference bit, Modified bit
  - bits are set when page is referenced, modified
- Pages are classified
  - not referenced, not modified
  - not referenced, modified
  - referenced, not modified
  - referenced, modified
- NRU removes page at random
  - from lowest numbered non empty class

FIFO Page Replacement Algorithm

- Maintain a linked list of all pages
  - in order they came into memory
- Page at beginning of list replaced
- Disadvantage
  - page in memory the longest may be often used

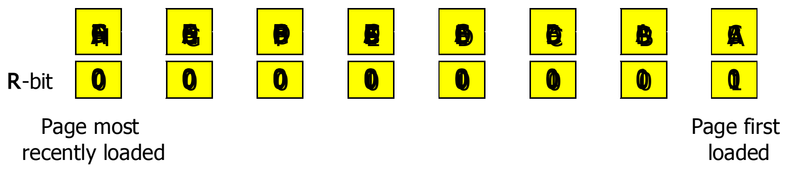
Second Chance

- Modification of FIFO
- *R* bit: when a page is referenced again, the R bit is set, and the page will be treated as a newly loaded page

Reference string: A B C D A E F G H I

Page I will be inserted, find a page to page out by looking at the first page loaded:

Page A's R-bit = 1 → move last in chain and clear R-bit, look at new first page (B)
Page B's R-bit = 0 → page out, shift chain left, and insert I last in the chain
-if R-bit = 0 → replace
Now the buffer is full, next page fault results in a replacement
-if R-bit = 1 → clear R-bit, move page last, and finally look at the new first page

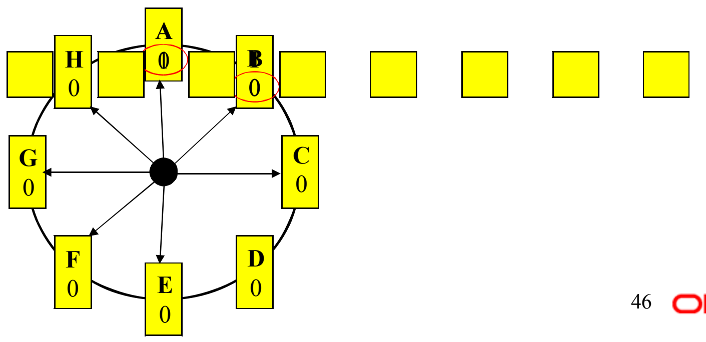


Second chance is a reasonable algorithm, but inefficient because it is moving pages around the list

Clock

- More efficient way to implement Second Chance
- Circular list in form of a clock
- Pointer to the oldest page:
  - R-bit = 0 → replace and advance pointer
  - R-bit = 1 → set R-bit to 0, advance pointer until R-bit = 0, replace and advance pointer

Reference string: A B C D A E F G H I

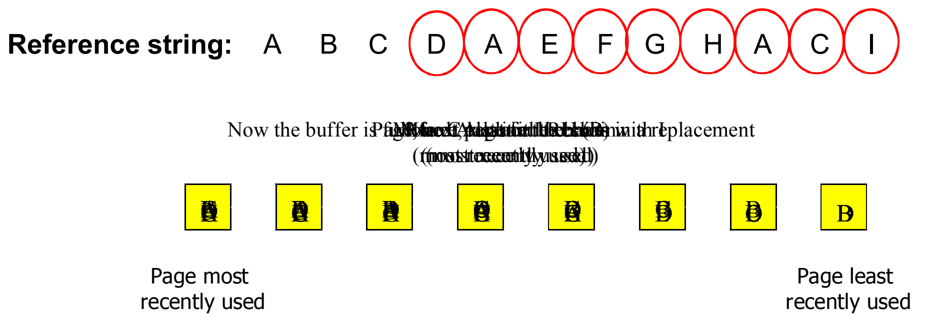


Least Recently Used (LRU)

- Replace the page that has the longest time since last reference
- Based on observation:
  - pages that are heavily used in the last few instructions will probably be used again in the next few instructions
- Several ways to implement this algorithm

Least Recently Used (LRU)

- LRU as a linked list:



- **Expensive** - maintaining an ordered list of all pages in memory:
  - most recently used at front, least at rear
  - update this list every memory reference !!

Summary: Memory Management

- Algorithms
  - Paging and segmentation
    - Extended in address translation and virtual memory lectures
  - Placement algorithms for partitioning strategies
    - Mostly obsolete for system memory management
      - since hardware address translation is available
    - But still necessary for managing
      - kernel memory
      - memory within a process
      - memory of specialized systems (esp. database systems)

- Address translation solves
  - Solves addressing in a loaded program
- Hardware address translation
  - Supports protection from data access
  - Supports new physical memory position after swapping in
- Virtual memory provides
  - Provide larger logical (virtual) than physical memory
  - Selects process, page or segment for removal from physical memory