W4118 Operating Systems

Instructor: Junfeng Yang

Outline

- Dynamic memory allocation
  - Stack
  - Heap
    - Heap allocation strategies
- Intro to memory management

Dynamic memory allocation

- Static (compile time) allocation is not possible for all data

- Two ways of dynamic allocation
  - Stack allocation
    - Restricted, but simple and efficient
  - Heap allocation
    - More general, but less efficient
    - More difficult to implement

Stack organization

- Memory is freed in opposite order from allocation. Last in First out (LIFO)
- When useful?
  - Memory usage pattern follows LIFO
    - E.g., function call frames
- Implementation
  - Pointer separating allocated and free space
  - Allocate: increment pointer
  - Free: decrement pointer

Pros and cons of stack organization

- Pros
  - Simple and efficient
  - Keeps all free space continuous

- Cons
  - Not for general data structures

Heap organization

- Allocate from random locations
  - Memory consists of allocated area and free area (or holes)
- When useful?
  - Allocate and free are unpredictable
  - Complex data structures
    - new in C++, malloc in C, kmalloc in Linux kernel

Pros and cons of heap organization

- Pros
  - General, works on arbitrary allocation and free patterns

- Cons
  - End up with small chunks of free space

Dynamic allocation issue: fragmentation

- Small trunks of free memory, too small for future allocation requests
  - External fragment: visible to system
  - Internal fragment: visible to process (e.g. if allocate at some granularity)
- Goal
  - Reduce number of holes
  - Keep holes large
- Stack fragmentation v.s. heap fragmentation

Heap implementation

- Data structure: linked list of free blocks
  - free list: chains free blocks together
- Allocation
  - Choose block large enough for request
  - Update free list
- Free
  - Add block back to list
  - Merge adjacent free blocks

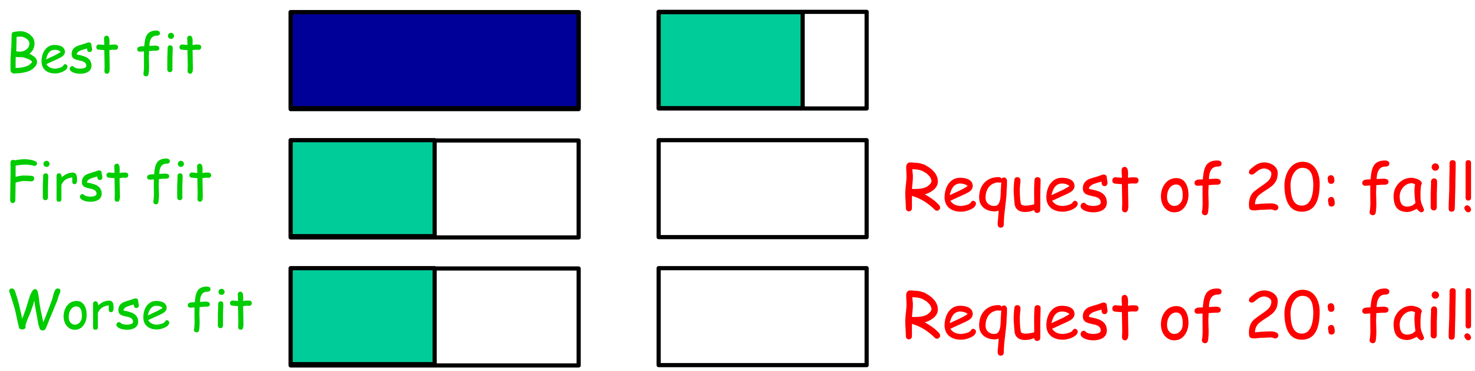
Heap allocation strategies

- Best fit
  - Search the whole list on each allocation
  - Choose the smallest block that can satisfy request
  - Can stop search if exact match found
- First fit
  - Choose first block that can satisfy request
- Worst fit
  - Choose largest block (most leftover space)

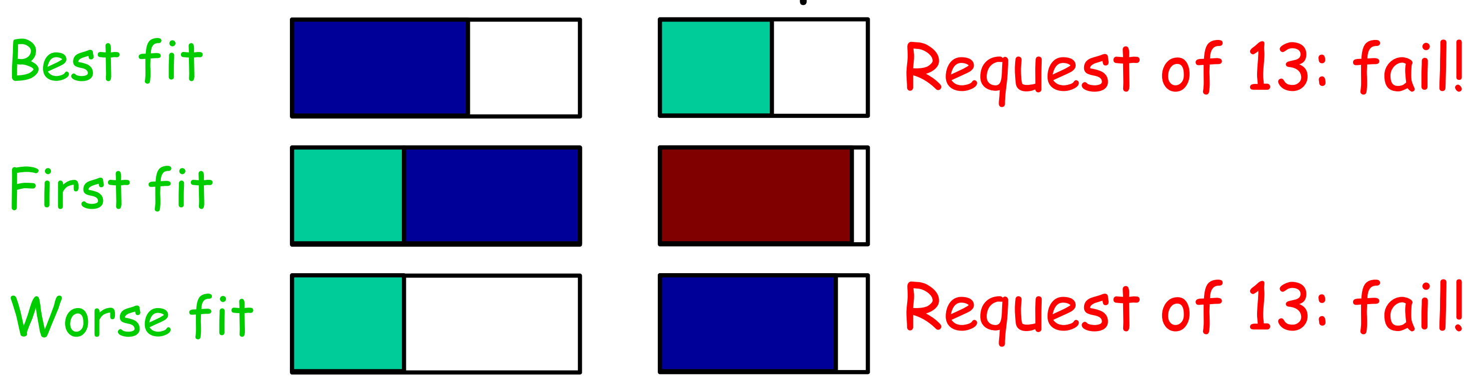
Which is better?

Example

- Free space: 2 blocks, size 20 and 15
- Workload 1: allocation requests: 10 then 20



- Workload 2: allocation requests: 8, 12, then 13



Comparison of allocation strategies

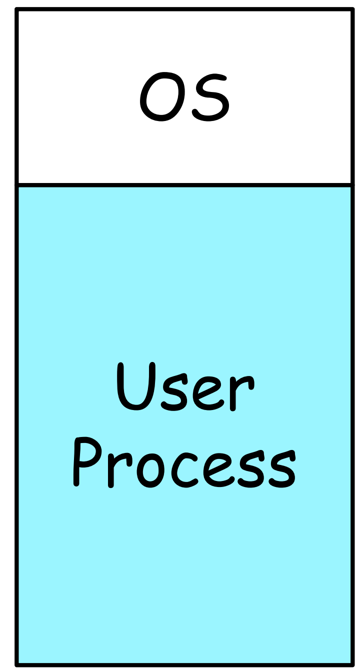
- Best fit
  - Tends to leave very large holes and very small holes
  - Disadvantage: very small holes may be useless
- First fit:
  - Tends to leave "average" size holes
  - Advantage: faster than best fit
- Worst fit:
  - Simulation shows that worst fit is worst in terms of storage utilization

Outline

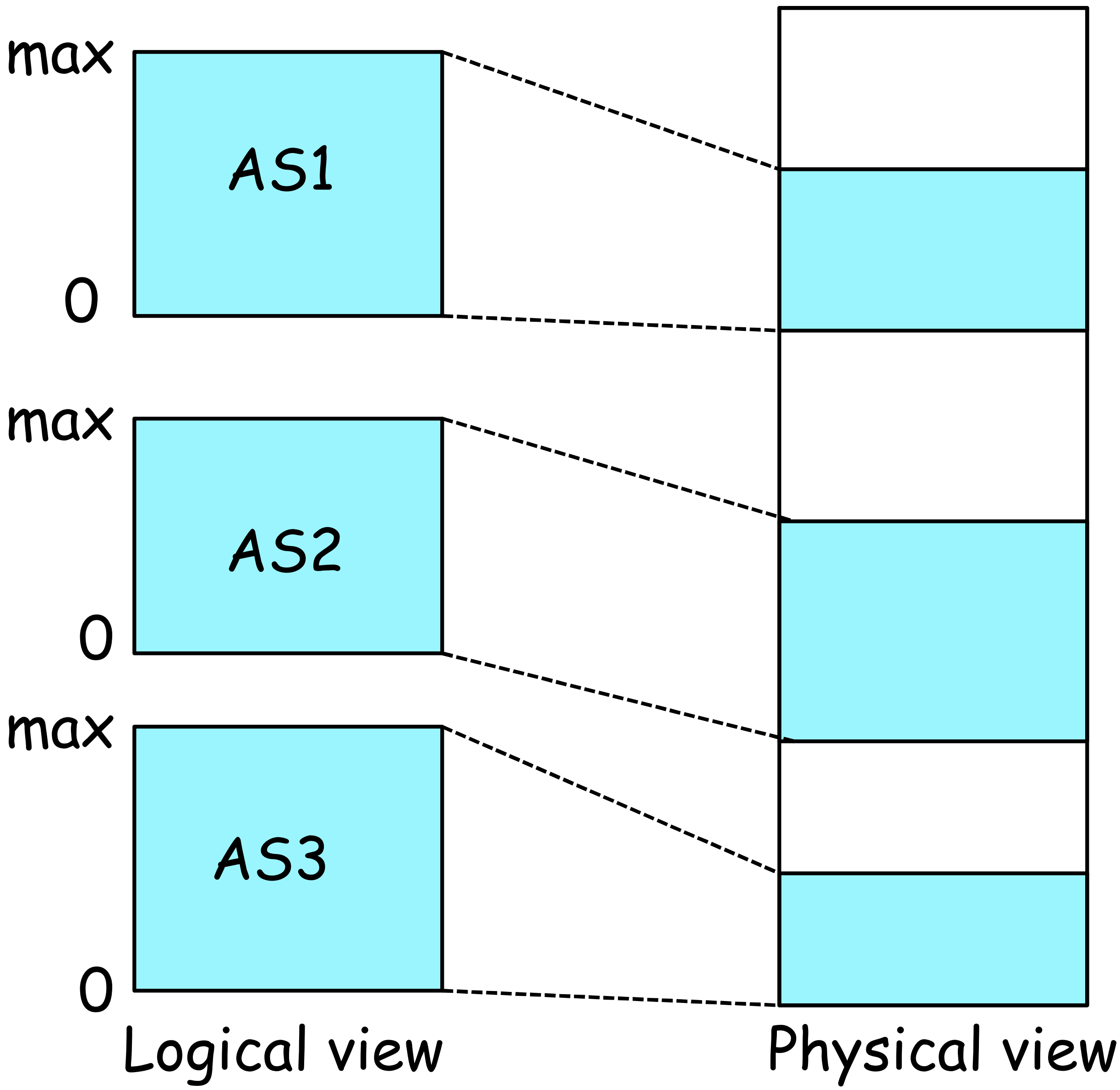
- Dynamic memory allocation
  - Stack
  - Heap
    - Heap allocation strategies
- Intro to memory management

Motivation for memory anagement

- Simple uniprogramming with a single segment per process
- Uniprogramming disadvantages
  - Only one process can run a time
  - Process can destroy OS
- Want multiprogramming!



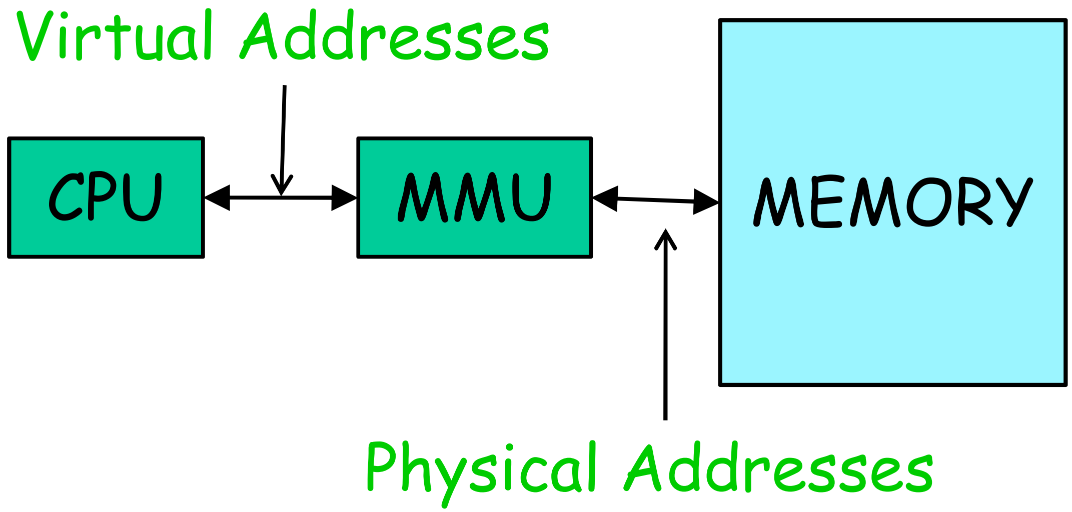
Multiple address spaces co-exist



Multiprogramming wish-list

- Sharing
  - multiple processes coexist in main memory
- Transparency
  - Processes are not aware that memory is shared
  - Run regardless of number/locations of other processes
- Protection
  - Cannot access data of OS or other processes
- Efficiency: should have reasonable performance
  - Purpose of sharing is to increase efficiency
  - Do not waste CPU or memory resources

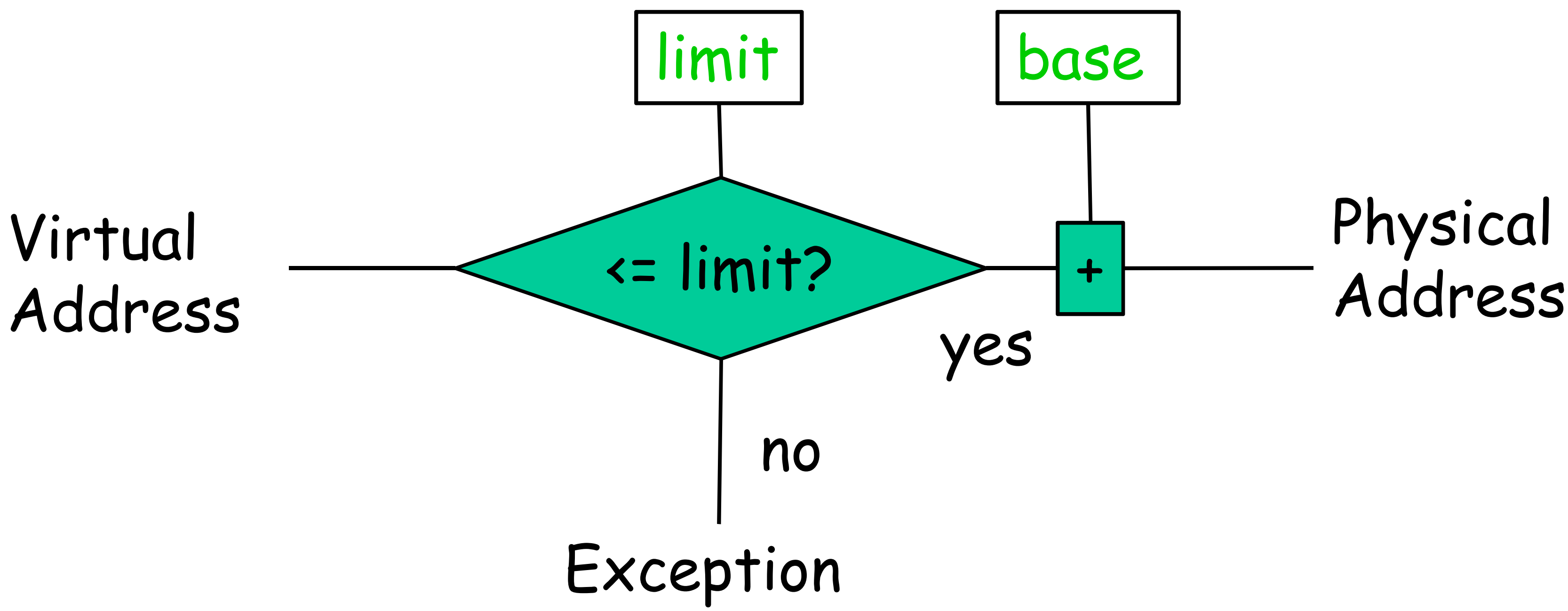
Memory translation and protection



- Map program-generated address (virtual address) to hardware address (physical address) dynamically at every reference
  - MMU: Memory Management Unit
  - Controlled by OS

Simple implementation of memory translation and protection

- Compare logical address to limit register
  - If greater, generate exception
- Add base register to logical address to generate physical address



Managing processes with base and limit

- Does base contain logical or physical address?
- How to relocate process?
- Are base and limit registers per-process or global?
- What to do on a context switch?
- Can user processes modify base and limit registers?

Pros and cons of base and limit

- Advantages
  - Supports dynamic relocation of address space
  - Supports protection across multiple spaces
  - Cheap: few registers and little logic
  - Fast: add and compare can be done in parallel
- Disadvantages
  - Process must be allocated contiguously
  - May allocate memory not used
  - Cannot share limited parts of address space