Memory Management

Abstraction is our Business

- What I have
  - A single (or a finite number) of CPUs
  - Many programs I would like to run
- What I want: a Thread
  - Each program has full control of one or more CPUs

Abstraction is our Business

- What I have
  - A certain amount of physical memory
  - Multiple programs I would like to run
    - together, they may need more than the available physical memory
- What I want: an Address Space
  - Each program has as much memory as the machine's architecture will allow to name
  - All for itself

Address Space

- Set of all names used to identify and manipulate unique instances of a given resource
  - memory locations (determined by the size of the machine's word)
    - for 32-bit-register machine, the address space goes from 0x00000000 to 0xFFFFFFFF
  - memory locations (determined by the number of memory banks mounted on the machine)
  - phone numbers (XXX) (YYY-YYYY)
  - colors: R (8 bits) + G (8 bits) + B (8 bits)

Virtual Address Space: An Abstraction for Memory

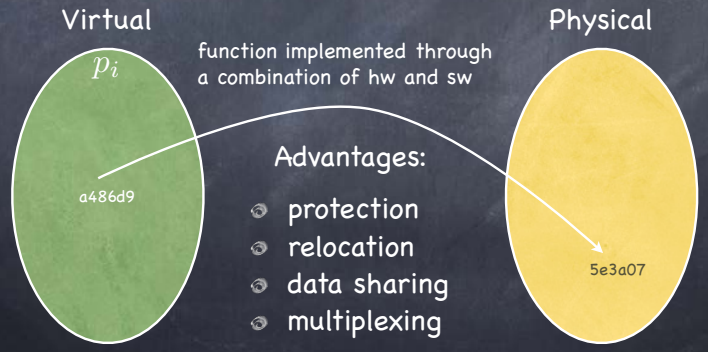
- Virtual addresses start at 0
- Heap and stack can be placed far away from each other, so they can nicely grow
- Addresses are all contiguous
- Size is independent of physical memory on the machine

Physical Address Space: How memory actually looks

- Processes loaded in memory at some memory location
  - virtual address 0 is not loaded at physical address 0
- Multiple processes may be loaded in memory at the same time, and yet...
- ...physical memory may be too small to hold even a single virtual address space in its entirety
  - 64-bit registers, anyone?

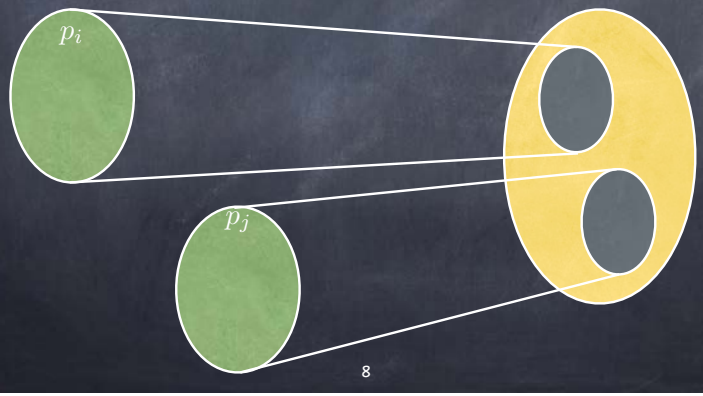
Address Translation

- A function that maps $\langle pid, virtual\ address \rangle$ into a corresponding *physical address*



Protection

- The functions used by different processes map their virtual addresses to disjoint ranges of physical addresses

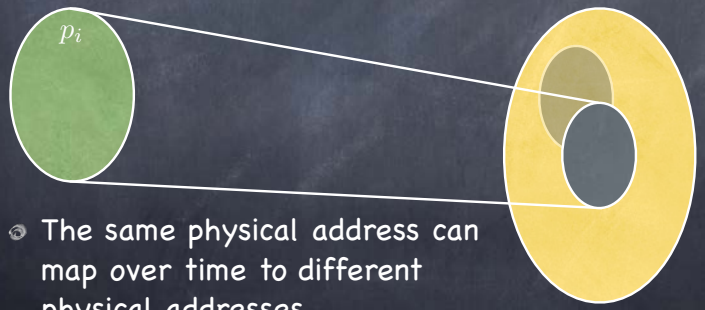


Relocation

- The range of the function used by a process can change over time

Relocation

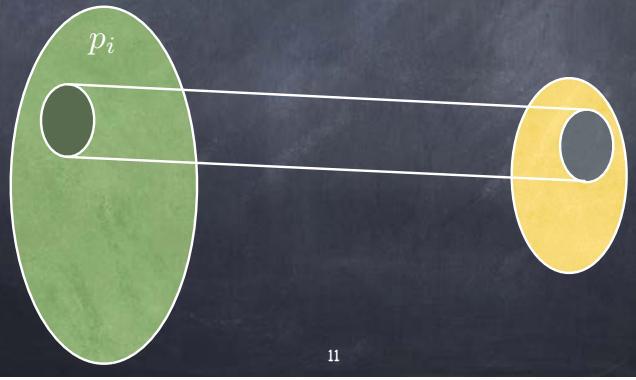
- The range of the function used by a process can change over time



- The same physical address can map over time to different physical addresses
  - or the mapping can be (temporarily) undefined

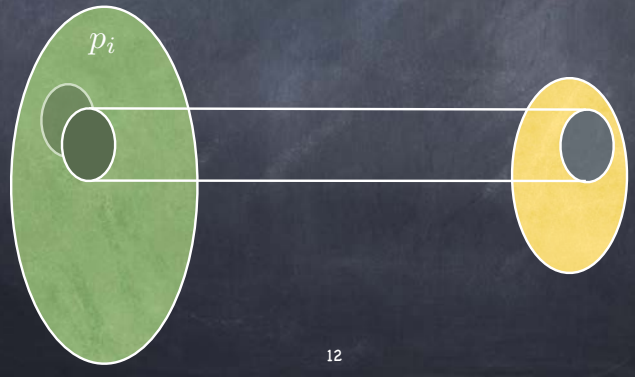
Multiplexing

- The set of virtual addresses that map to a given range of physical addresses can change over time



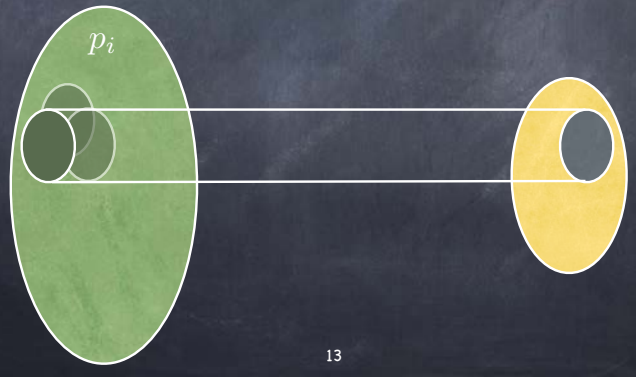
Multiplexing

- The set of virtual addresses that map to a given range of physical addresses can change over time



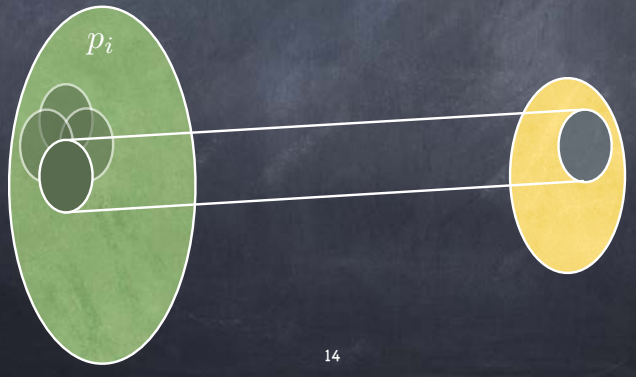
Multiplexing

- The set of virtual addresses that map to a given range of physical addresses can change over time



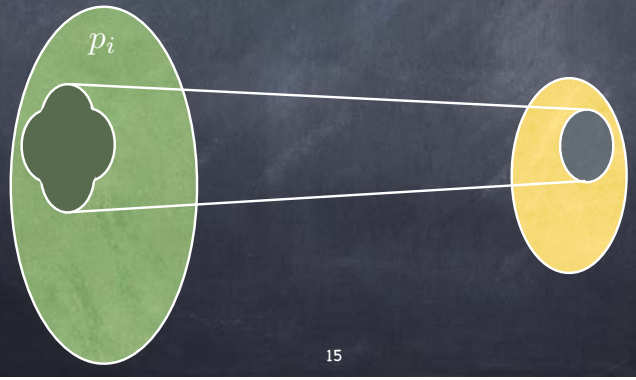
Multiplexing

- The set of virtual addresses that map to a given range of physical addresses can change over time



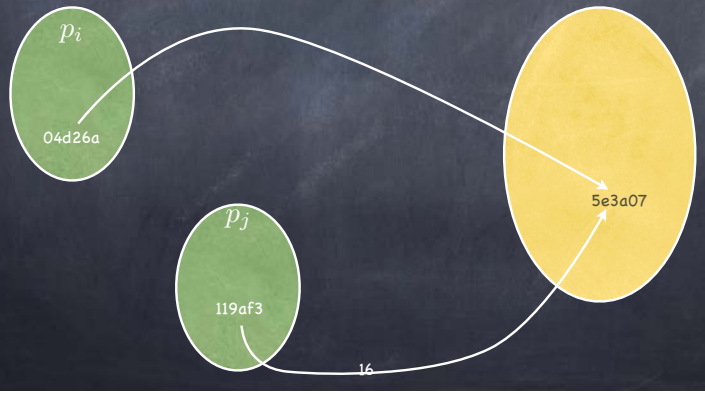
Multiplexing

- The set of virtual addresses that map to a given range of physical addresses can change over time



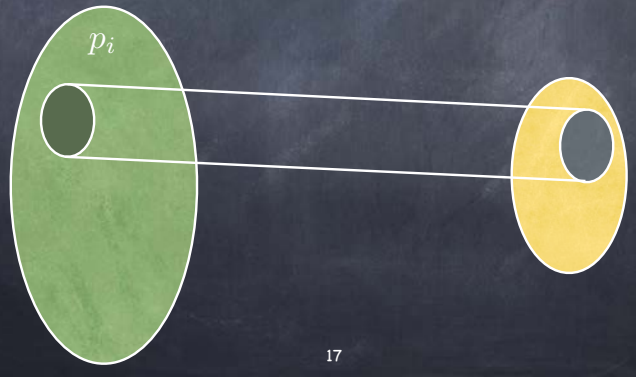
Data Sharing

- Map different virtual addresses of different processes to the same physical address



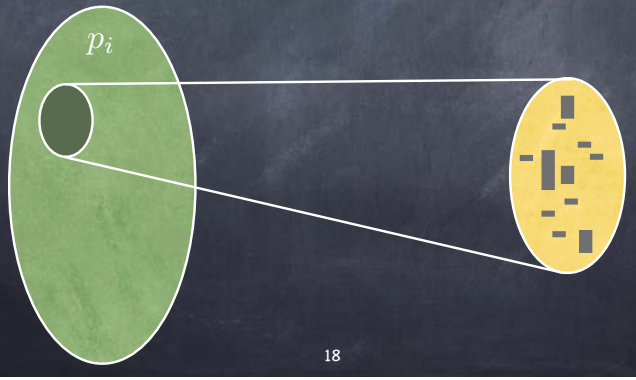
Contiguity

- Contiguous virtual addresses need not map to contiguous physical addresses



Contiguity

- Contiguous virtual addresses need not map to contiguous physical addresses



The Identity Mapping

- Map each virtual address onto the identical physical address
  - Virtual and physical address spaces have the same size
  - Run a single program at a time
    - OS can be a simple library
    - very early computers
- Friendly amendment: leave some of the physical address space for the OS
  - Use loader to relocate process
    - early PCs

More sophisticated address translation

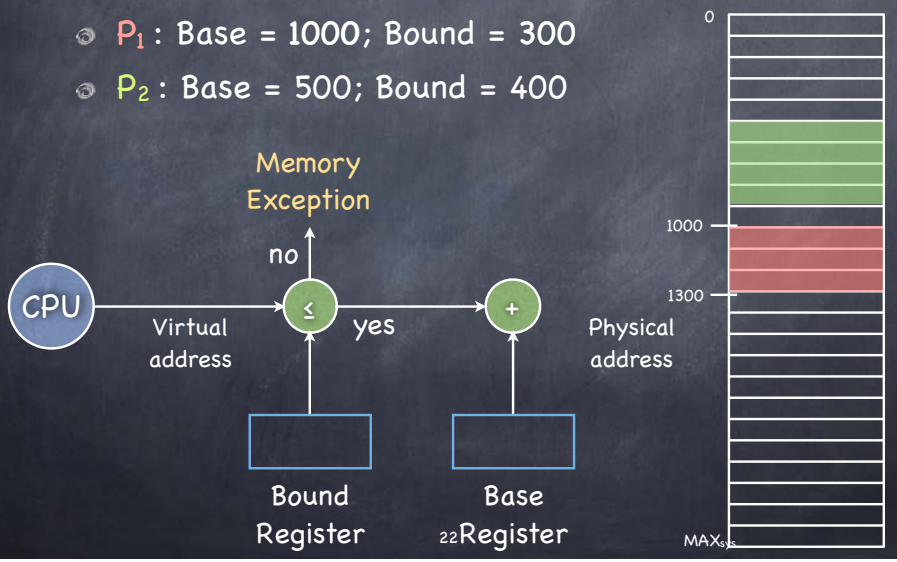
- How to perform the mapping efficiently?
  - So that it can be represented concisely?
  - So that it can be computed quickly?
  - So that it makes efficient use of the limited physical memory?
  - So that multiple processes coexist in physical memory while guaranteeing isolation?
  - So that it decouples the size of the virtual and physical addresses?
- Ask hardware for help!

Base & Bound

- Goal: allow multiple processes to coexist in memory while guaranteeing isolation
- Needed hardware
  - two registers: Base and Bound (a.k.a. Limit)
  - Stored in the PCB
- Mapping
  - pa = va + Base
    - as long as 0 ≤ va ≤ Bound
  - On context switch, change B&B (privileged instruction)

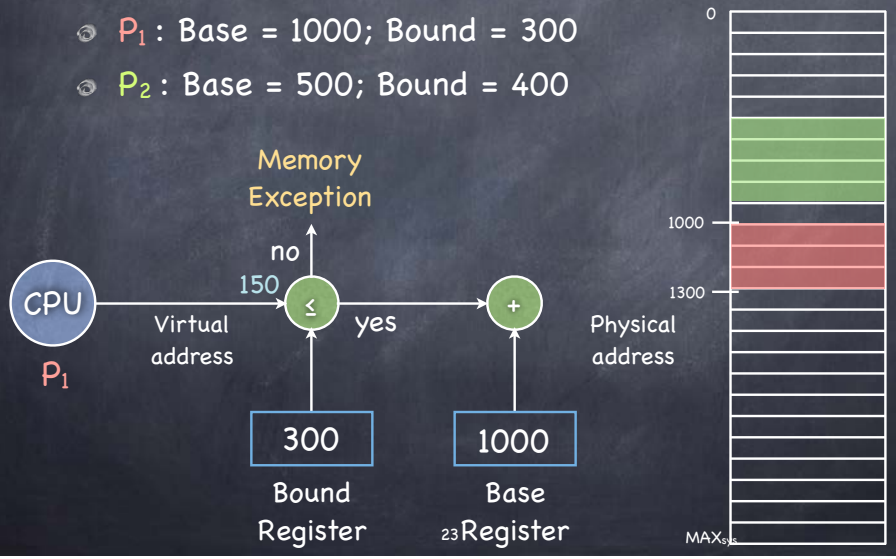
Base & Bound

- P_1 : Base = 1000; Bound = 300
- P_2 : Base = 500; Bound = 400



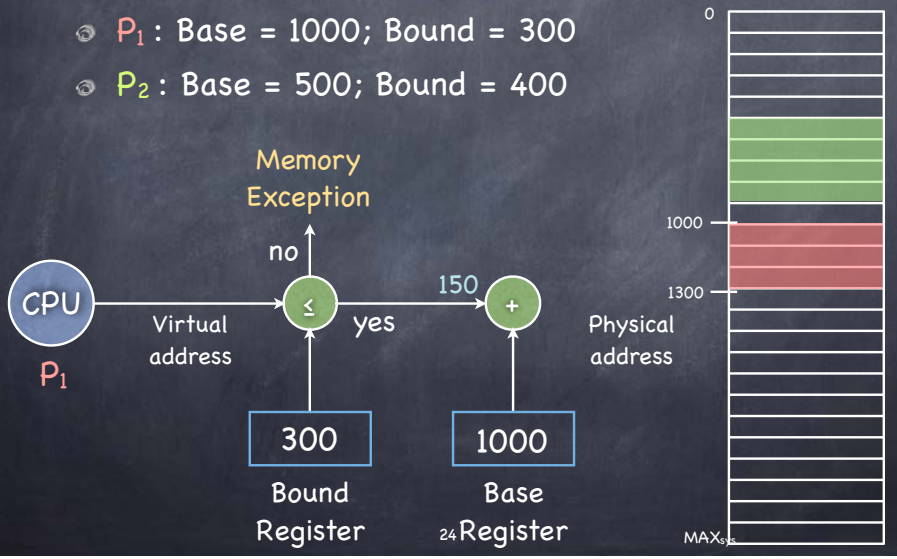
Base & Bound

- P_1 : Base = 1000; Bound = 300
- P_2 : Base = 500; Bound = 400



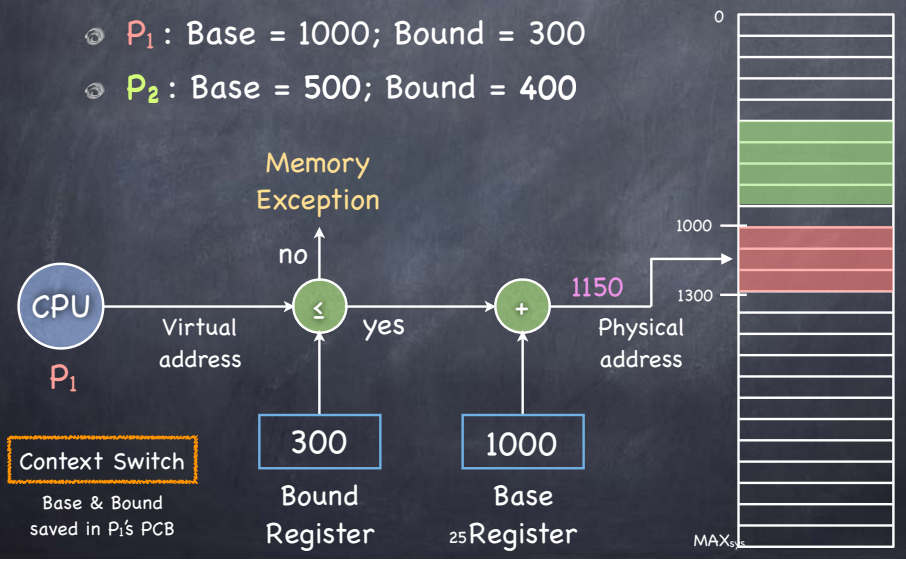
Base & Bound

- P_1 : Base = 1000; Bound = 300
- P_2 : Base = 500; Bound = 400



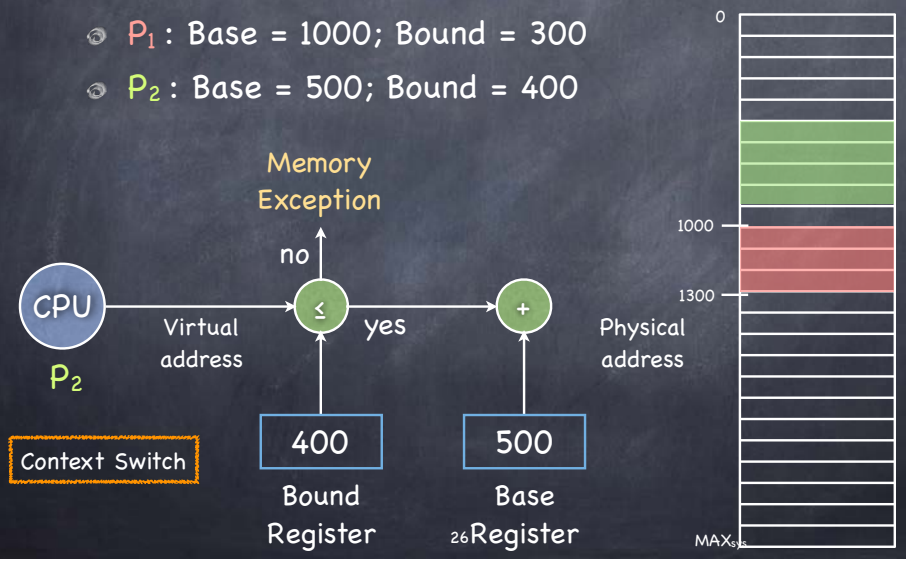
Base & Bound

- P_1 : Base = 1000; Bound = 300
- P_2 : Base = 500; Bound = 400



Base & Bound

- P_1 : Base = 1000; Bound = 300
- P_2 : Base = 500; Bound = 400



On Base & Bound

- Contiguous Allocation
  - contiguous virtual addresses are mapped to contiguous physical addresses
- But mapping entire address space to physical memory
  - is wasteful
    - lots of free space between heap and stack…
    - makes sharing hard
  - does not work if the address space is larger than physical memory
    - think 64-bit registers…