# CSCE 451/851
# Operating Systems Principles

## Segmentation & Shared Memory

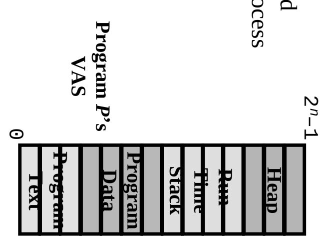
Steve Goddard
*goddard@cse.unl.edu*

***http://www.cse.unl.edu/~goddard/Courses/CSCE451***



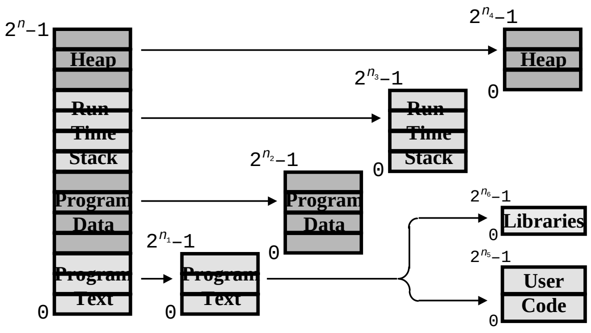
## Memory Management
### Name spaces

- Schemes to date have considered only a single name space per process
  - » Why?
    (Why not have multiple spaces?)



## Multiple Name Spaces

### Example — Protection/Fault isolation & sharing




3



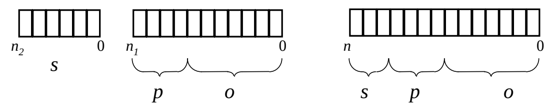
## Supporting Multiple Name Spaces

### Segmentation

- Segment — a memory "object"
  - » A virtual address space
- A process now addresses objects —a pair (*s*, *addr*)
  - » *s* — segment number
  - » *addr* — a virtual address within an object offset
    - virtual address as before: (*p*, *o*)
- A virtual address is now a triple (*s*, *p*, *o*)

Segment + Address register scheme:

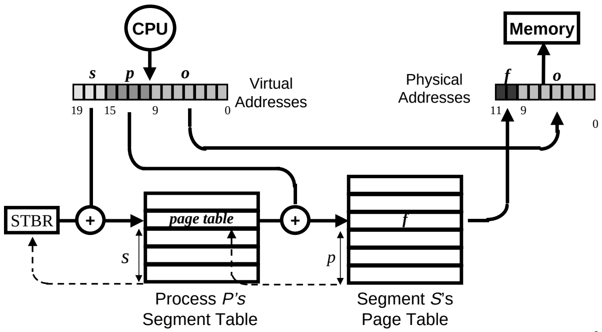
Single address scheme:




4

## Segmentation
### Virtual address translation

- One additional level of indirection — The *segment table*




5



## Shared Memory
### Sharing of procedures and data

- Why share?

- Levels of sharing
  - » source code
  - » object code
  - » executable binaries

6

# Shared Memory
## Requirements for sharing

### Sharing Code

- Programs cannot modify their own code
  - » "Pure procedures"
- Serially sharable code
  - » Programs that are self-initializing
  - » Programs that contain their own global data
- Concurrently sharable code
  - » Programs must be *reentrant*
    - ❖ Run-time stack cannot be shared

### Sharing Data

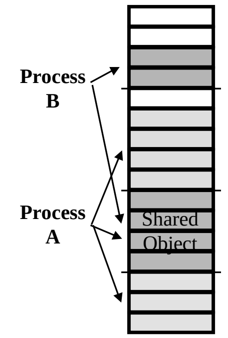
- Data cannot (typically) contain pointers

7



# Sharing in Non-Segmented/Paged Systems
## Trading-off protection for sharing

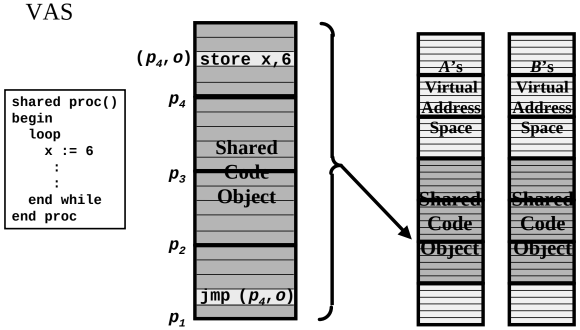
- Space can no longer be allocated contiguously
  - » Base + Limit register schemes not applicable
- Loaders must determine if copies of shared objects already exist in memory
  - » And if so where is it?
- Similar problems with garbage collection




8

## Sharing in Paged Systems
### Code sharing

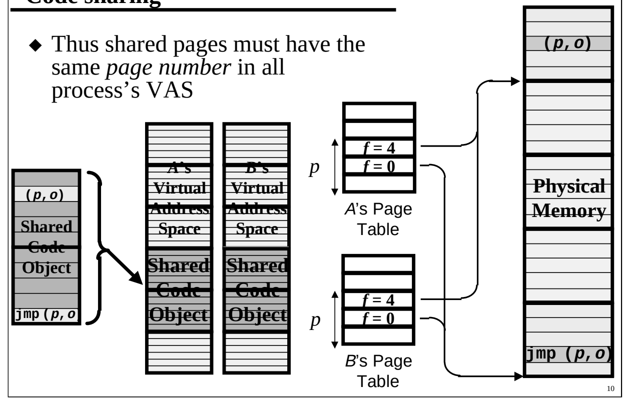
- Shared code must reside in the same place in all process's VAS




9

## Sharing in Paged Systems
### Code sharing

- Thus shared pages must have the same *page number* in all process's VAS



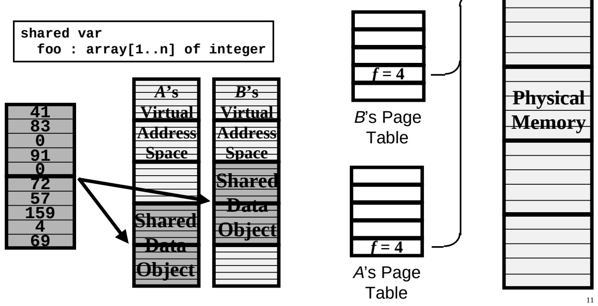

10

# Sharing in Paged Systems

## Data sharing

- "Raw data" — pages can appear anywhere in a process's VAS

```
shared var
  foo : array[1..n] of integer
```




11

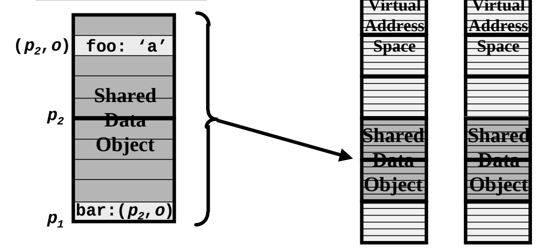


# Sharing in Paged Systems

## Data sharing

- Shared data containing pointers must also reside in the same place in all process's VAS

```
shared var
  foo : char
  bar : ptr to char
```

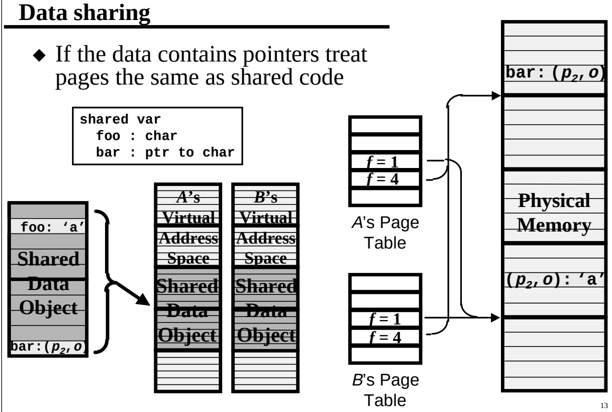



12

# Sharing in Paged Systems
## Data sharing

- If the data contains pointers treat pages the same as shared code

```
shared var
  foo : char
  bar : ptr to char
```

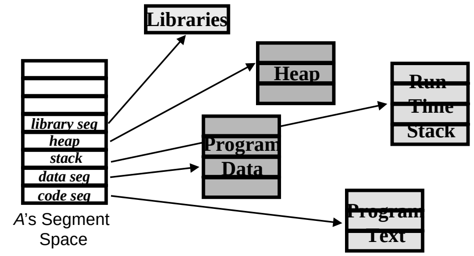



13



# Sharing in Segmented Systems
## The simplest form of sharing

- Sharing portion of a process's "segment space"

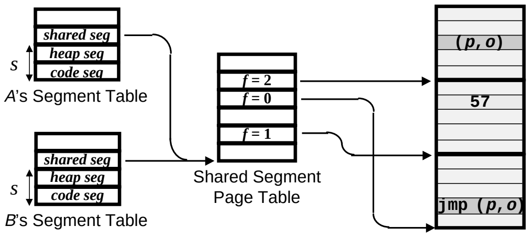



14

# Sharing in Segmented Systems

## Sharing segments

- Unit of sharing a *segment* rather than *a set of pages*
  - » Processes need only agree on 1 number rather than a sequence of numbers
  - » If segments are paged then the page tables are automatically shared








# Dynamic Linking & Sharing

## A key technology for realizing shared memory

- Concept
  - » Dynamically bind to shared modules at run-time
- Advantages
  - » On demand linking ensures that only those modules that are actually used will be loaded
  - » Possibly allows modules to be replaced or upgraded at run-time
- Disadvantages
  - » Possible performance penalty at run-time for invoking new modules
  - » Overhead of indirect/interpretive access
  - » Complexity of the overall scheme