Sharing of Address Spaces

S. Brobst
Advanced Computer Systems Architecture
Lecture #11
October 8, 1986

Consensus: Demand Paging is a Win

- Large virtual address spaces are desirable from a software point of view.
- Makes efficient use of primary memory.

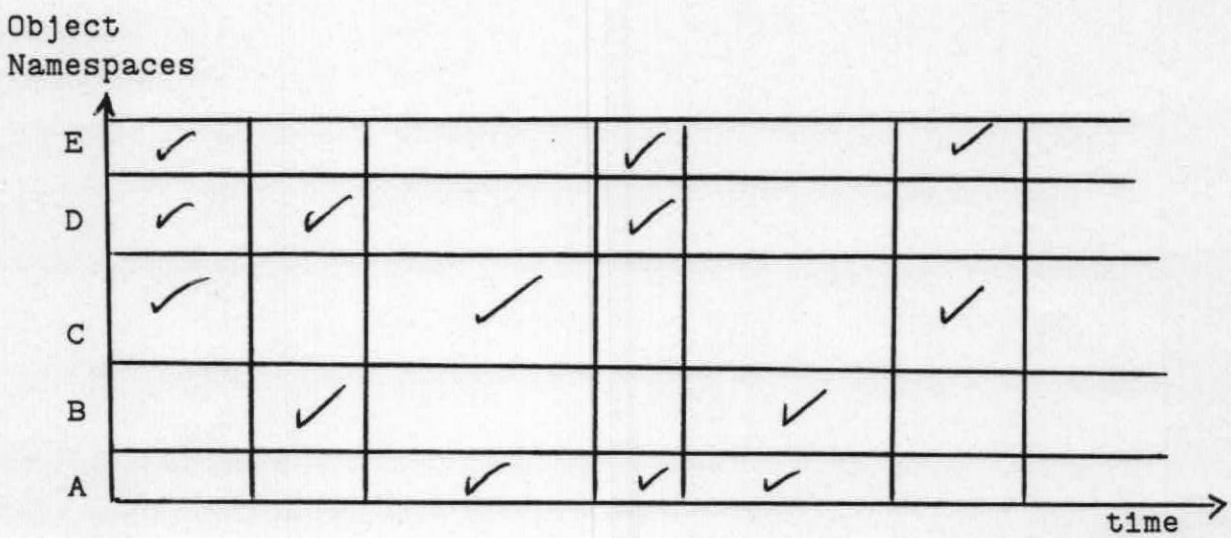
Why are Large Virtual Address Spaces Needed?

Because we may need to give a unique name to each object used during a computation.

\- or -

We can't use all names effectively.

Name Requirements of a Process



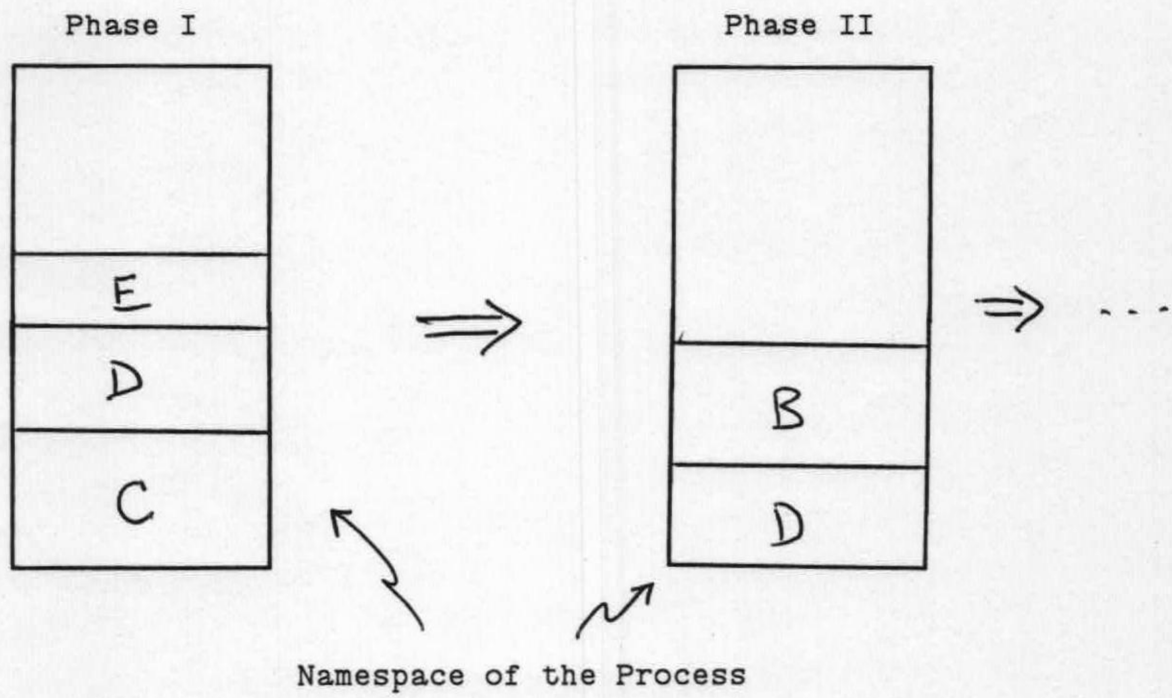
If the name space is not large enough to concurrently accomodate all of the objects A, B, C, D, E then

1. Enlarge the namespace, or

2. The process must rename some objects at the time of phase transition. This is known as "The Overlay Problem."

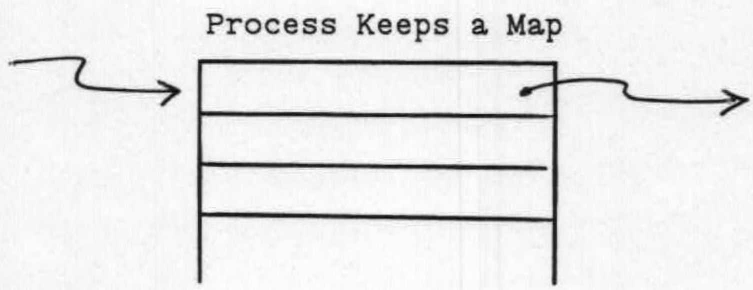
The Overlay Problem

If sufficent namespace does not exist, then objects must be renamed to make room for new objects as needed.



Renaming Objects

1. Data Objects: Double indexing or indirect addressing is needed.

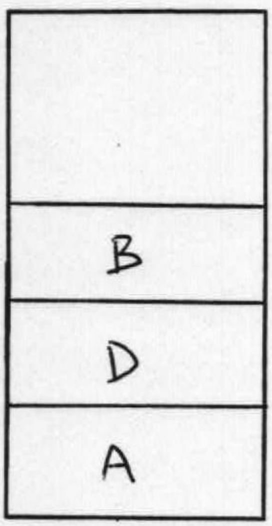


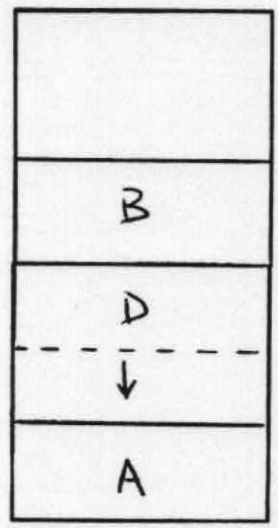
2. Procedure Objects: Requires a procedure base register, otherwise the usual problems of static relocation crop up.

Renaming will often result in movement of information in the physical memory. Either that, or control over the page map is needed.

How to Deal with Variable Size Data Objects?

To avoid renaming each time an object increases its size, a large enough namespace is needed so that it is sparsely occupied by the totality of information.





To avoid renaming in A, a large enough address space should be allocated to D to accommodate growth in size.

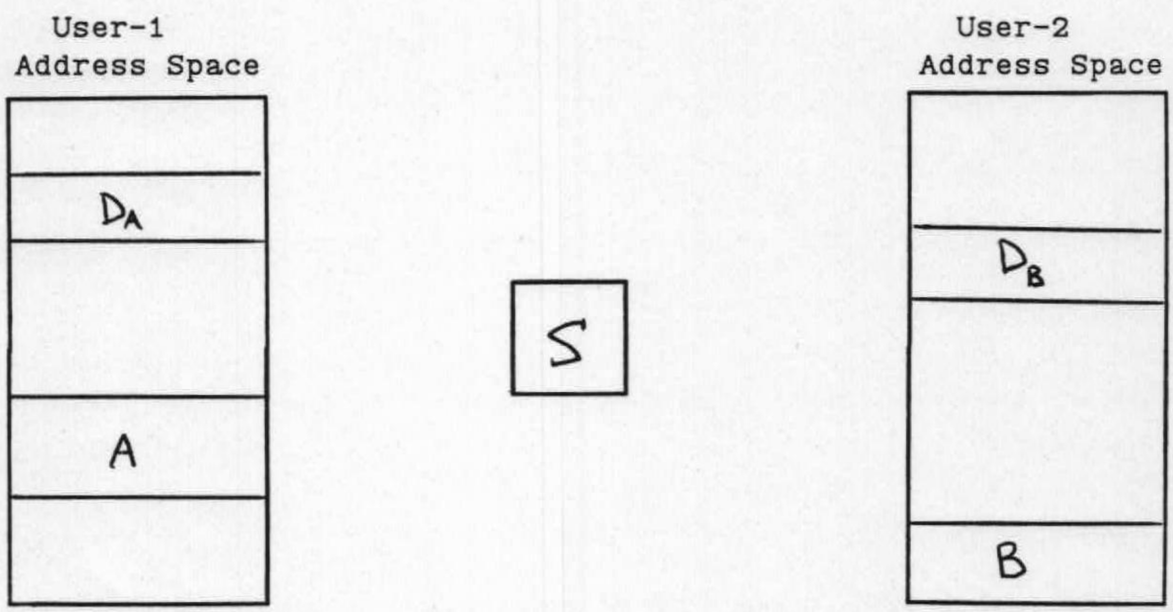
Note: It is acceptable to waste virtual address space as long as we don't waste physical address space.

Sharing of Information

1. Some information cannot be duplicated and thus must be shared.
    - Which pages are free?
    - A file of airline reservations.
2. Sharing can save storage, as well as unnecessary copying and I/O activity.
    - Text editors.
    - Compilers.
    - Large applications programs.

Sharing of Information

Two users $\equiv$ Two independent address spaces



Suppose:

- User-1 and User-2 both want to share subroutine S.
- S needs data which is private to the caller.

Sharing of Information

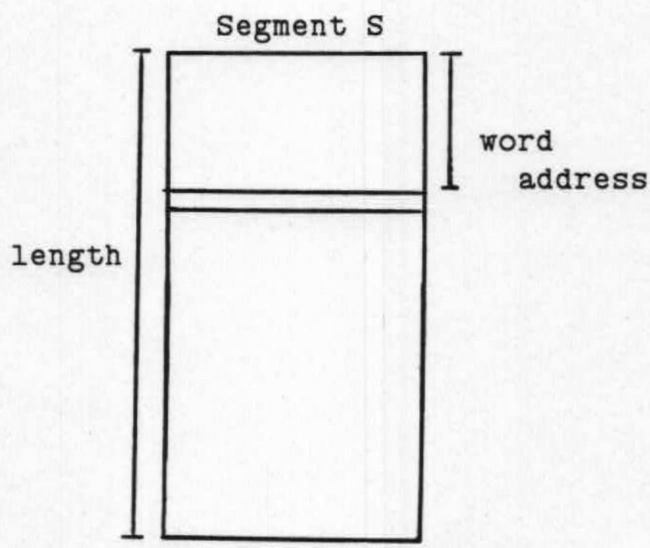
Possible solutions:

1. Create a new namespace S (*i.e.,* a new process) to sexecute S.
   - Allocate names for D in the new namespace S.
   - Change the address map in going from User-1 to S.
2. Allocate space for S in the namespace of both User-1 and User-2.
   - Internal names of S pose a problem if User-1 and User-2 assign S different locations in their respective address spaces.
   - Could provide a procedure base register for S, or
   - Can provide a very large address space to the users so that same address can be assigned to S in both namespaces (sharing by convention).

Segmentation Solution

A *segment* is an independent linear address space (which can grow to be very large).

Processes address information by generating a *segment name* and a *word address*.

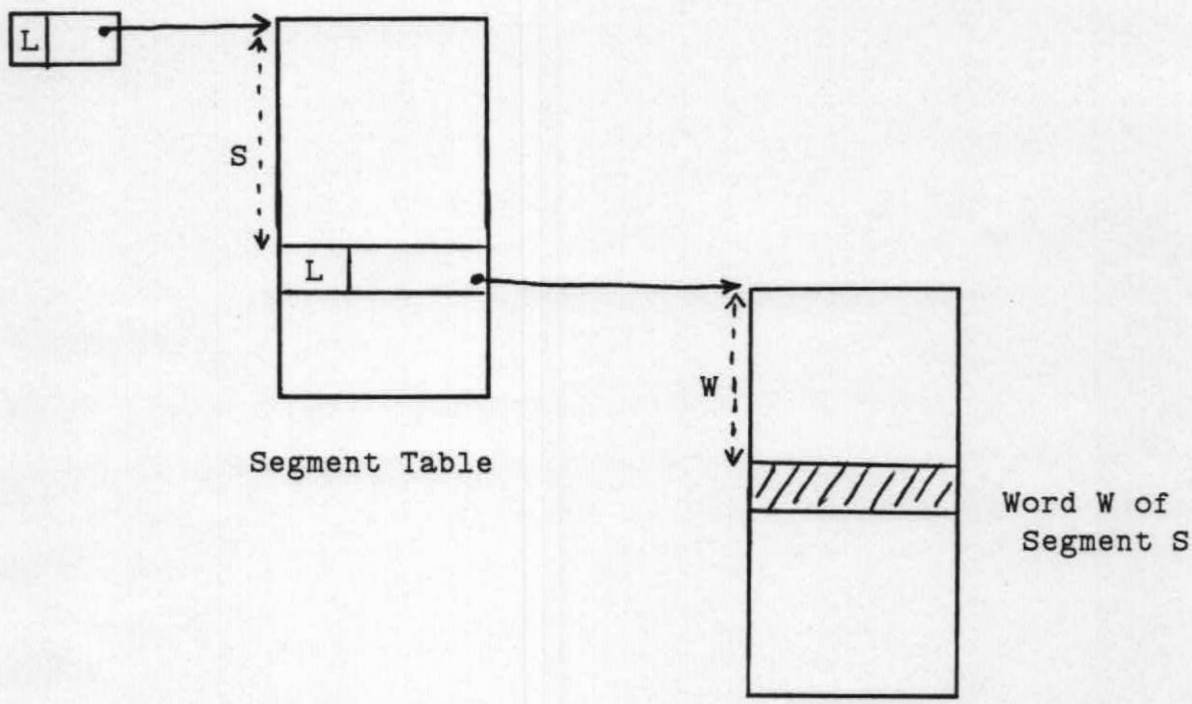


Segmentation

- Segments are arbitrary objects (procedures, data, etc.)
- Segments have protection rights associated with them (read, write, execute, etc.)
- Not all segments are of equal length.
- Two segments are indpendent of one another; segment names do not constitute a linear address space - no arithmetic allowed on segment names.

Segmentation

Addresses are interpreted as <segment number, word address> tuples.



- Origin of the segment table is kept in a special register of the processor.
- Both the segment table and the segment reside in main memory.
- References are not allowed to go past the length, L, of a segment.

Segmentation

Segments can be shared, yet a protection mechanism is required to provide controlled access to information.

- Access rights can be stored in the segment table (*e.g.,* read, write, execute).
- Different processes may have different privileges to the same segment.

Segmentation

We must have the ability to store the names of shared segments in other segments (*e.g.*, shared subroutines, linked data structures).

Question: How permanent should these names be?

Question: What does <s, w> mean when it is found in a segment?

- Segment names must be independent of a user process when stored in a segment.
- The integer name "S" for a particular segment is the same for all user processes.

Segmentation

There are two views of sharing that we can take in the world:

1. Processes share large pieces of information (*e.g.,* files, utility packages).
   $\Rightarrow$ | segment | $>$ | page |

2. Processes share small pieces of information (*e.g.,* LISP functions, single records in a database).

If the names of entities are to be directly understood by the processor, only one of the above two views can be implemented efficiently.

The first view is much easier to implement than the second.

Examples of 1: Multics, VAX-VMS

Examples of 2: Intel 432, Hydra, Cap, IBM System 38

Segmentation

1. Processes have a large enough namespace such that all information referenced can be assigned unique names in the form of a segment number.

2. Data objects are expandable without requiring a reallocation of namespace.

3. Information referenced by several processes has the same name (segment number) for all processes that reference it.

4. A protection mechanism operates in the namespace to permit access to information by a process only in an authorized manner.

VAX-VMS Virtual Memory Organization

Each user has two segments:

- Private (not shared)
- System (shared by everyone)

Fine grained sharing of the system segment is done by convention.

VAX-VMS Memory Management

- 32-bit virtual address space.
- $2^{32} = 4$ Billion bytes
- With 512 bytes per page, the system must support 2^{23} pages in the address space of each user.

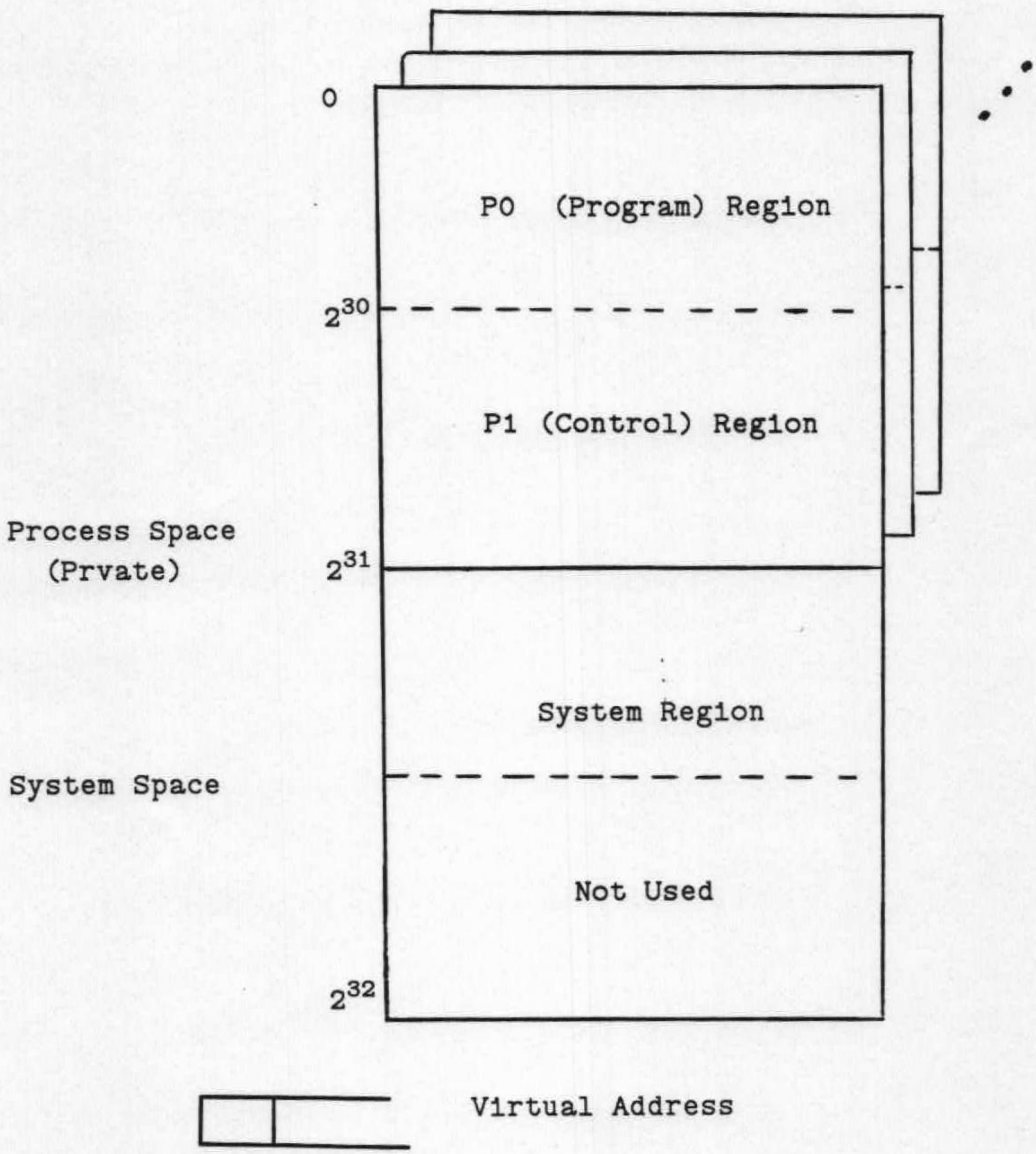
Virtual Address:

23 bit virtual page #	9 bit (byte) offset

Physical Address:

21 bit frame #	9 bit (byte) offset

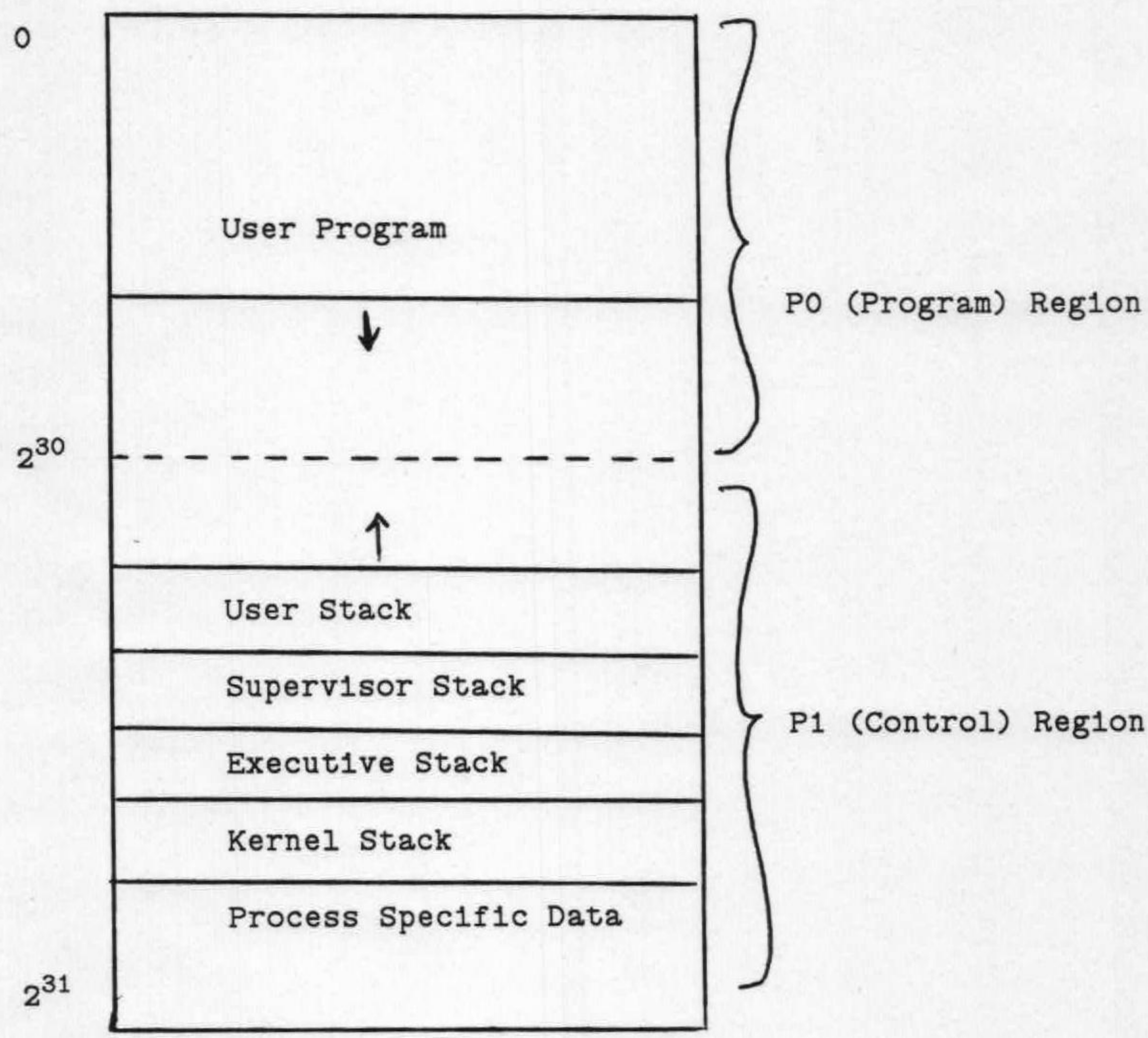
VAX-VMS Virtual Memory Organization



00 = P0 region
01 = P1 region
10 = System region
11 = Not used

VAX-VMS Virtual Memory Organization

Process Space:



VAX-VMS Virtual Memory Organization

Each region has its own page table.

- A page table is described by two registers representing its BASE and LENGTH.
- There is only one page table for the system space (**S**ystem **P**age **T**able).
- The system page table is resident in main memory.
- There are P0 and P1 page tables associated with each user process.
- Page tables for all user areas are stored in the system space.

System Base Register (SBR): A *physical* address specifying the base of the System Page Table.

System Length Register: A *physical* address specifying the length of the System Page Table.

P0 Base Register (P0BR): A *virtual* address specifying the base of the User Page Table.

P0 Length Register: A *virtual* address specifying the length of the User Page Table.

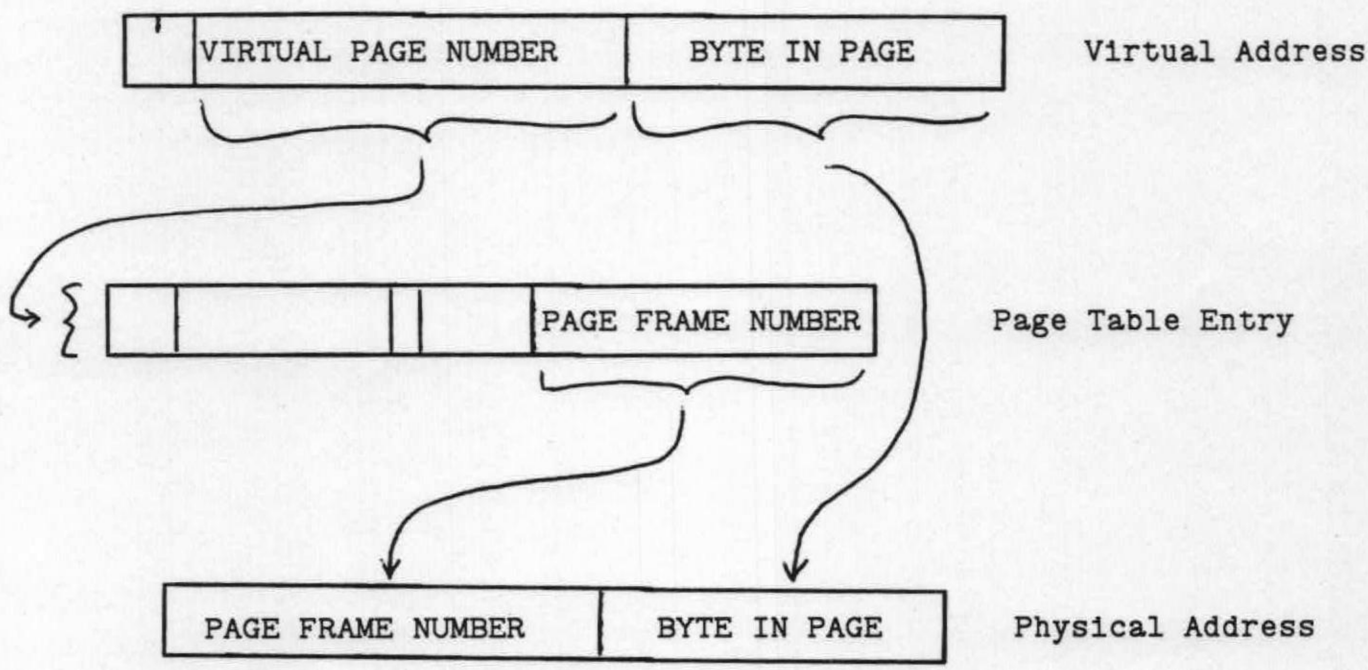
VAX-VMS Virtual Memory Organization

Process page tables are in virtual memory and may be swapped out.

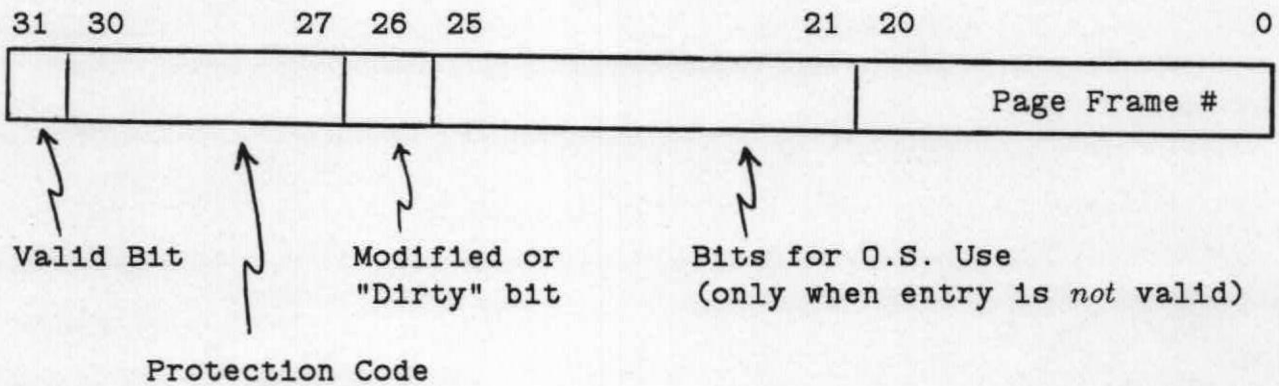
- Some memory references will cause multiple page faults (*i.e.,* page table page is swapped out, and data page is swapped out).

- We never get into trouble because process page table is in the system space, and the system page table is always kept in main memory (as is the page fault handler).

Virtual Address Translation



VAX-11 Page Table Entry Format



PROTECTION CODE	(kernel) K	(executive) E	(supervisor) S	(user) U
0000	-	-	-	-
0010	R,W	-	-	-
0011	R	-	-	-
0100	R,W	R,W	R,W	R,W
1110	R,W	R	R	R
1111	R	R	R	R

.
.
.

VAX-VMS Operating System Hierarchy