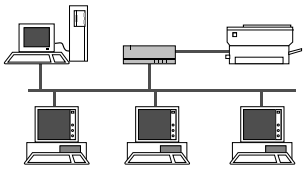


Operating Systems
600.418
Memory Management

Department of Computer Science
The Johns Hopkins University

Memory Management

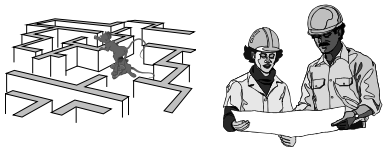
Lecture 5

Reading:
- Silberschatz & Galvin
- chapter 8

Additional Reading:
- Stallings
- chapter 7

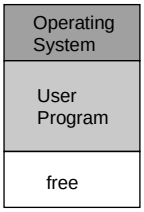
Memory Management

- Operating System's task that allows accommodation of multiple processes in memory.
- Memory needs to be allocated efficiently in order to allow as many processes into memory as possible.
- If not enough processes are in memory, multiprogramming will not be effective and the CPU will be idle large part of the time.
- In most operating systems, the kernel occupies some part of main memory and the rest is shared by multiple processes.



Contiguous Allocation

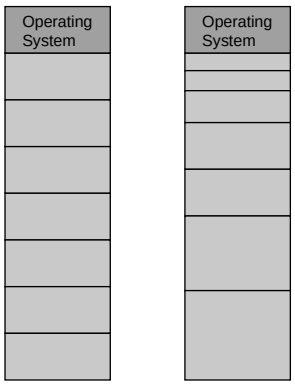
- **Single-Partition allocation**
  - Resident Operating System in low memory.
  - One user process in high memory.



- **Multiple-Partition allocation**
  - When a process arrives, it is allocated memory from a contiguous block of available memory, large enough to accommodate the process.
  - The operating system maintains information about free and allocated memory.
  - Partitioning: fixed versus dynamic.

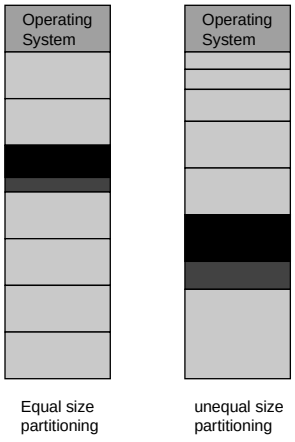
Fixed Partitioning

- The main memory is partitioned into a set of non-overlapping blocks.
- Partitions can be of equal or unequal size.
- A process can be loaded into a partition with **higher or equal size**.



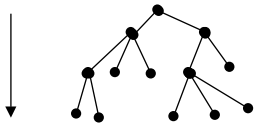
Internal Fragmentation

- Fixed Partitions suffer from inefficient memory use - any process, no matter how small, occupies an entire partition.
- This waste is called **Internal Fragmentation**.
- Unequal size partitions are **better** in terms of internal fragmentation.



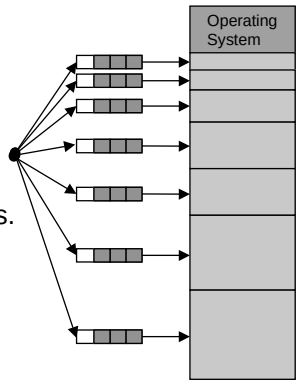
Overlays

- Problem: What if the process is larger than any of the partitions?
- Solution: **Keep in memory only those instructions and data that are needed at any given time.**
- Implemented by the programmer and compiler. Can be an extremely complex and time consuming task.



Placement Algorithms for Fixed Partitions

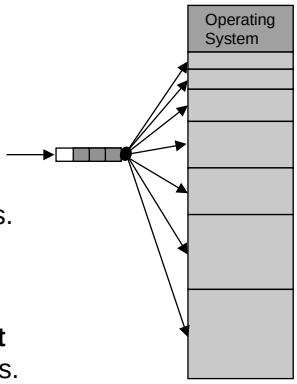
- Equal size partitions
  - **Trivial**
- Unequal size partitions
  - **Multiple queues**
    - Assigns each process to the **smallest partition** within which it fits.



unequal size partitioning

Placement Algorithms for Fixed Partitions

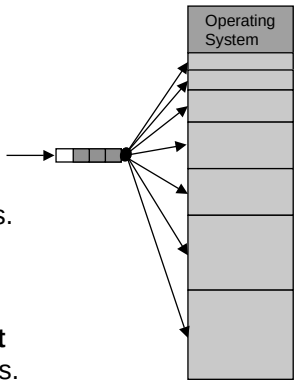
- Equal size partitions
  - **Trivial**
- Unequal size partitions
  - **Multiple queues**
    - Assigns each process to the **smallest partition** within which it fits.
  - **Single queue**
    - Assign each process to the **smallest available partition** within which it fits.
  - Which is best?



unequal size partitioning

Placement Algorithms for Fixed Partitions

- Equal size partitions
  - **Trivial**
- Unequal size partitions
  - **Multiple queues**
    - Assigns each process to the **smallest partition** within which it fits.
    - Minimizes internal fragmentation.
  - **Single queue**
    - Assign each process to the **smallest available partition** within which it fits.
    - Increases the level of multiprogramming on the expense of internal fragmentation.

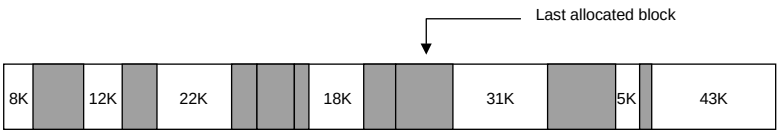


unequal size partitioning

Dynamic Partitioning

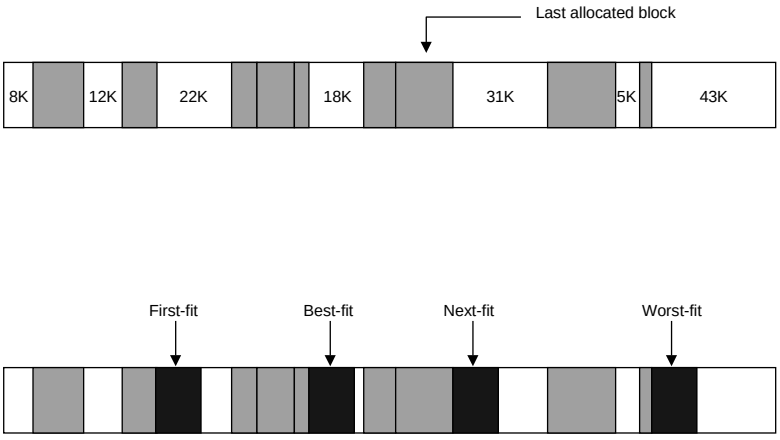
- Partitions are of variable length and number.
- Each process is allocated a contiguous memory block of exactly the size it requires - no internal fragmentation.
- Termination of a process releases its memory block.
- Possible allocation algorithms:
  - **Best-fit: allocate the smallest hole that fits.**
  - **Worst-fit: allocate the largest hole.**
  - **First-fit: allocate the first hole that fits from the beginning.**
  - **Next-fit: allocate the first hole that fits from last placement.**

Dynamic Partitioning - Example



Please allocate 16K for the next process.

Dynamic Partitioning - Example



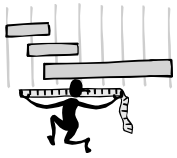
External Fragmentation

Please allocate 63K for the next process.

- Eventually, main memory forms holes too small to hold any process. This is external fragmentation.
- Total memory space may exist to satisfy a request but it is not contiguous.
- Compaction reduces external fragmentation by shuffling memory contents to place all free memory together in one large block.
- Think what is needed to be able to move processes around?

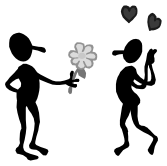
Placement Algorithms for Dynamic Partitioning

- First-fit and next-fit can allocate faster than best-fit and worst-fit (no need to scan the complete list).
- First-fit favors allocation near the beginning and tends to create less fragmentation then next-fit.
- Worst-fit is the worst method both in terms of fragmentation and in allocation speed.
- There is no clear winner between first-fit and best-fit in terms of external fragmentation.
- How do we know all that? Usually, by running simulations. But this may have a problem - what are the base assumptions?



Buddy System

- A reasonable compromise between fixed and dynamic partitioning schemes.
- Memory blocks are available in size of 2^k where L≤k≤U and where
  - 2^L **= smallest size of block that can be allocated.**
  - 2^U **= largest size of block that can be allocated (the entire memory).**
- The OS maintains several lists of holes
  - **The i-list has holes of size** 2^i **.**
  - **A pair of available buddies in the i-list are collapsed into a single hole in the i+1 list.**

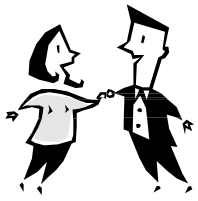


Example of Buddy System

1 Mbyte block	1 M					
Request 100 K	A = 128 K	128 K	256 K	512 K		
Request 240 K	A = 128 K	128 K	B = 256 K	512 K		
Request 64 K	A = 128 K	C = 64 K	64 K	B = 256 K	512 K	
Request 256 K	A = 128 K	C = 64 K	64 K	B = 256 K	D = 256 K	256 K
Release B	A = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
Release A	128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
Request 75 K	E = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
Release C	E = 128 K	128 K	256 K	D = 256 K	256 K	
Release E	512 K			D = 256 K	256 K	
Release D	1 M					

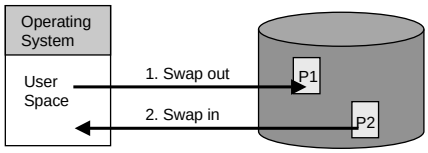
Buddy System - Properties

- What can you say about fragmentation ?
- No external fragmentation.
  - **No need for compaction.**
- On average, internal fragmentation is 25%.
  - **Each block is at least 50% occupied.**



Swapping

- **A process can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution.**
- **Backing store - fast disk large enough to accommodate copies of all memory images for all users.**
- **Major part of swap time is transfer time, which is proportional to the amount of memory swapped.**
- **Roll out, roll in - swapping used for priority-based medium term scheduling (lower priority process is swapped out so higher priority process can be loaded).**

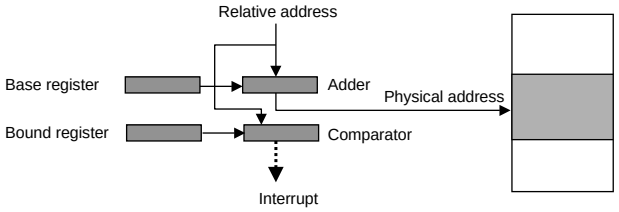


Relocation

- **Because of swapping and compaction, a process may occupy different main memory locations during its lifetime.**
- **Therefore, physical memory references by a process cannot be fixed.**
- **Physical address versus logical address:**
  - Physical address is an absolute address in main memory.
  - Logical address is a reference to a memory location independent of the physical organization of memory.
  - Relative address is a logical address in which the address is expressed as a location relative to some known point in the program (e.g. the beginning).

Address Translation

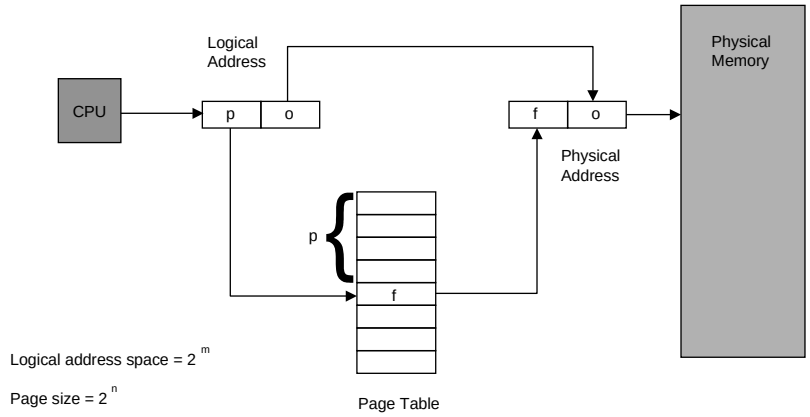
- Compilers produce code in which all memory references are logical addresses.
- Physical addresses are calculated on the fly as the instructions are executed.
- Hardware support is needed. For example:
  - **Base register and bound register**
  - **Each process can only access memory within its process image.**



Paging

- Main memory is partitioned into relatively small, equal sized blocks named frames. Frame size is power of 2.
- Each process is also divided into blocks of the same size called pages.
- Logical address space of a process can be noncontiguous.
- A process is allocated physical memory wherever it is available.
- The system keeps track of all free frames.
- A page table translates logical to physical addresses.
- **No external fragmentation.**
- **Small internal fragmentation**.

Basic Paging Architecture

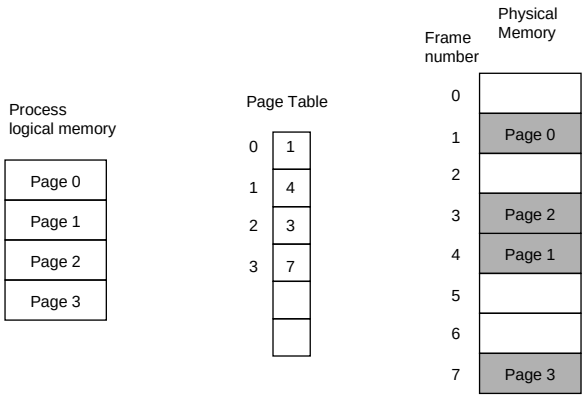


Logical address space = 2^m

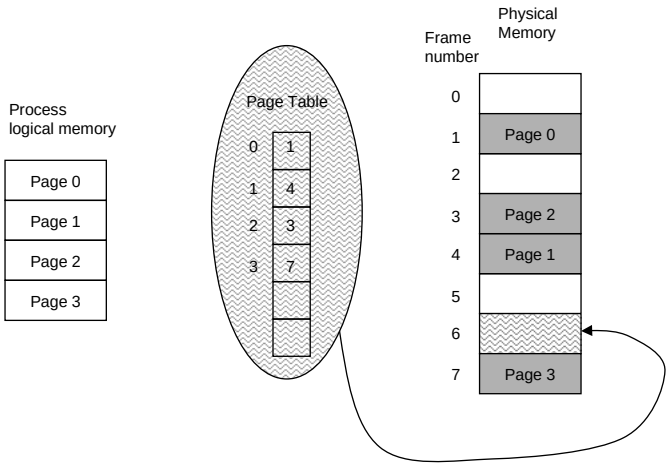
Page size = 2^n

o = offset in page, represented by n bits
p = page number, represented by m - n bits
f = frame number in physical memory.

Example of Paging



Example of Paging

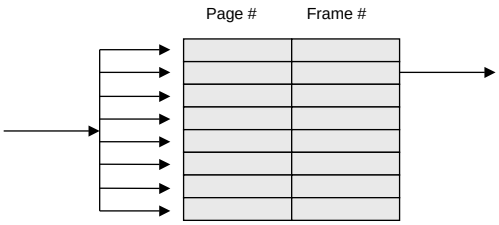


Implementation of a Page Table

- **Each process has its own page table.**
- **Page table is kept in memory.**
- **Page table** base register **points to the page table.**
- **Page table** length register **indicates the size of the page table.**
- How many physical memory accesses are required for each logical memory access?
- Answer: 2. (one for the page table and one for the data/instruction).
- Can we do better ?

Translation Look-aside Buffers (TLB)

- Associative registers - parallel search.
- Hardware support!
- If page number is in associative registers, get frame number.
- Otherwise get frame number from page table in memory.



Effective Access Time

Suppose:

- TLB lookup takes 5 nano sec.
- Memory access time is 100 nano sec.
- Hit ratio (probability to find page number in TLB) is α

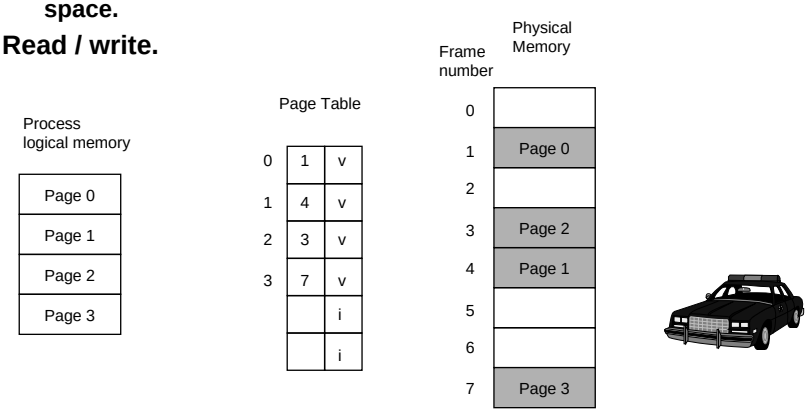
Effective Access Time = $(5+100)*\alpha + (5+100+100)*(1-\alpha)$

Suppose α = 80% (for example, TLB size = 16)
EAT = 105*.8 + 205*.2 = 125 nano sec.

Suppose α = 98% (for example, TLB size = 64)
EAT = 105*.98 + 205*.02 = 107 nano sec.

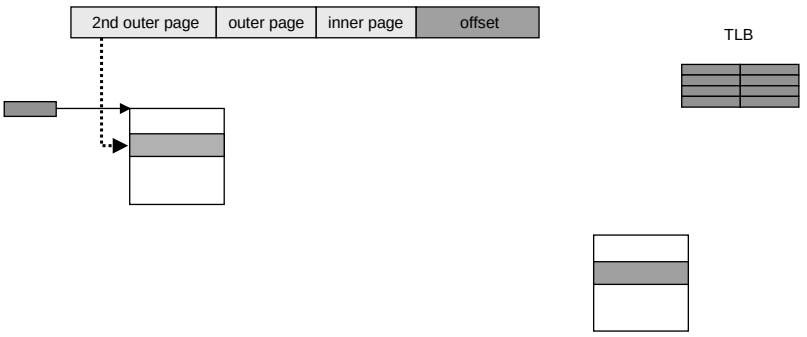
Memory Protection

- **Memory protection is implemented by associating protection bits with each frame.**
- **Valid- invalid bit attached to each entry in the page table.**
  - **Valid - the associated page is in the process' logical address space.**
- **Read / write.**



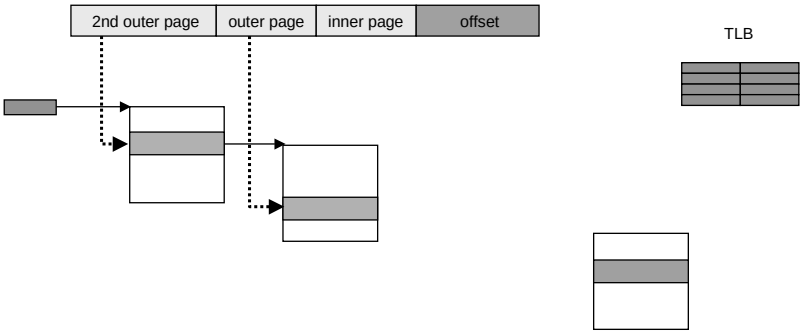
Multilevel Paging

- The page table might be too big to fit in a contiguous space, so we may have a hierarchy with several levels.
- Example:
  - Three level paging:



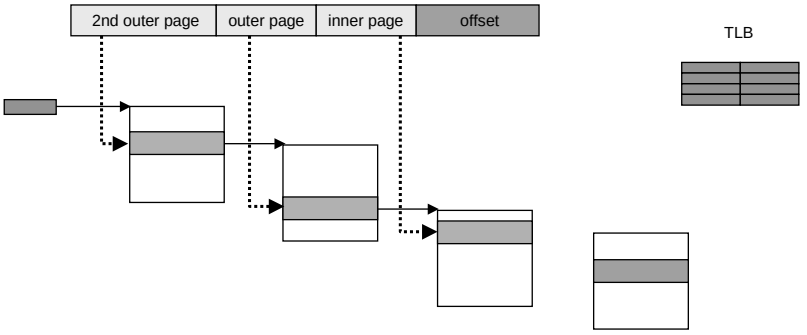
Multilevel Paging

- The page table might be too big to fit in a contiguous space, so we may have a hierarchy with several levels.
- Example:
  - Three level paging:



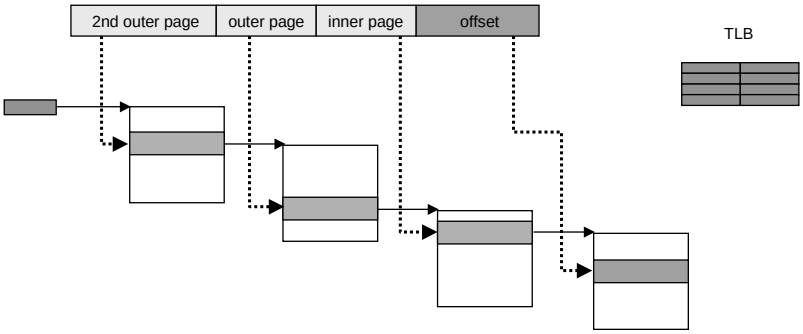
Multilevel Paging

- The page table might be too big to fit in a contiguous space, so we may have a hierarchy with several levels.
- Example:
  - Three level paging:



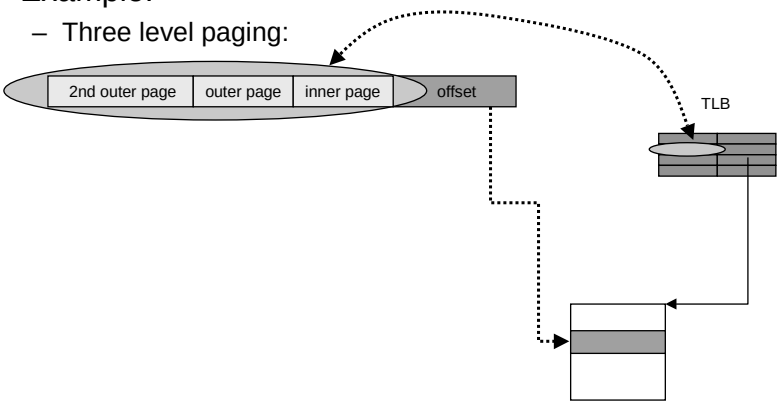
Multilevel Paging

- The page table might be too big to fit in a contiguous space, so we may have a hierarchy with several levels.
- Example:
  - Three level paging:



Multilevel Paging

- The page table might be too big to fit in a contiguous space, so we may have a hierarchy with several levels.
- Example:
  - Three level paging:



Performance of Multilevel Paging

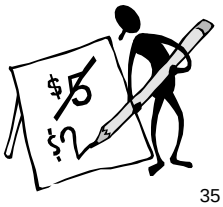
Suppose:

- Three level paging
- TLB lookup takes 5 nano sec.
- Memory access time is 100 nano sec.
- Hit ratio (probability to find page number in TLB) is α

Effective Access Time = $(5+100)*\alpha + (5+300+100)*(1-\alpha)$

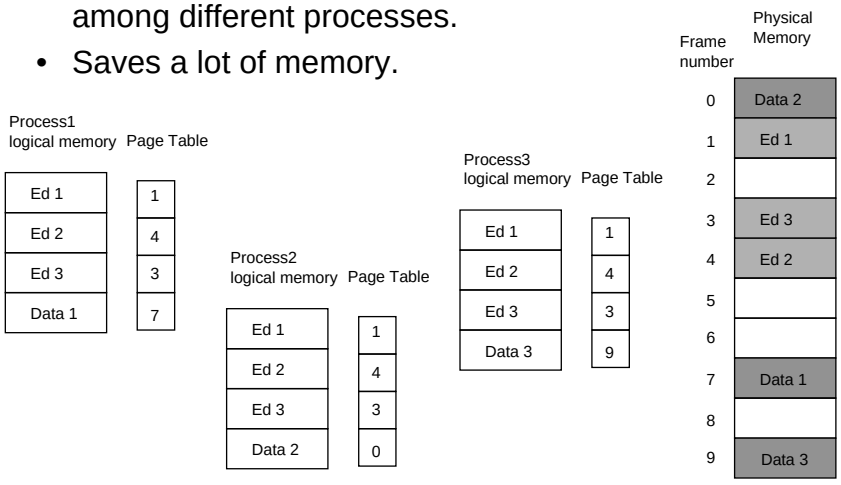
Suppose α = 80% (for example, TLB size = 16)
EAT = 105*.8 + 405*.2 = 165 nano sec.

Suppose α = 98% (for example, TLB size = 64)
EAT = 105*.98 + 405*.02 = 111 nano sec.



Page Sharing

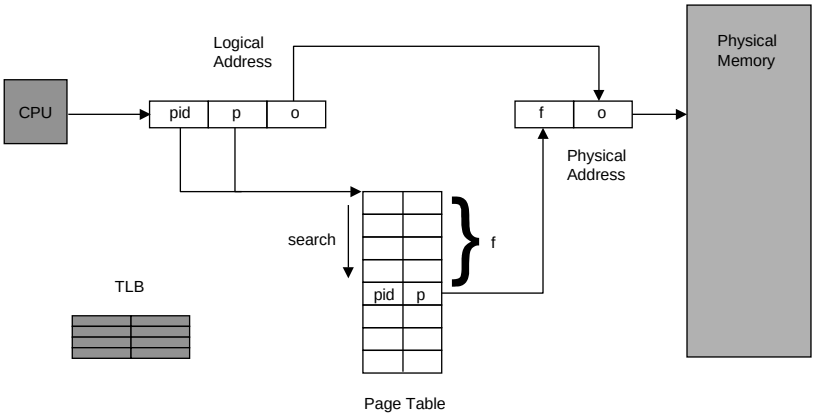
- One copy of read-only (reentrant) code shared among different processes.
- Saves a lot of memory.



Inverted Page Table

- With each process having its own page table, and with each page table consuming considerable amount of memory…We use a lot of memory to keep track of memory.
- Inverted page table has one entry for each real page of memory.
- Lookup time is increased because it requires a search on the inverted table.
- Hash table can be used to reduce this problem.
- Of course, we should also use TLB.
- Say goodbye to sharing ?

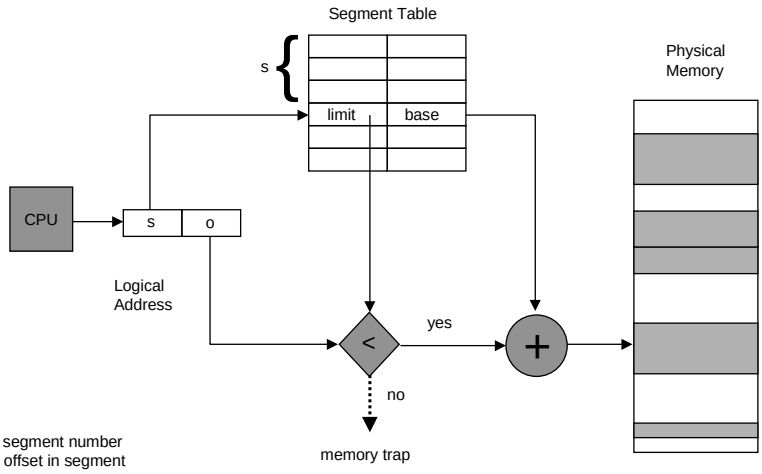
Inverted Page Table Architecture



Segmentation

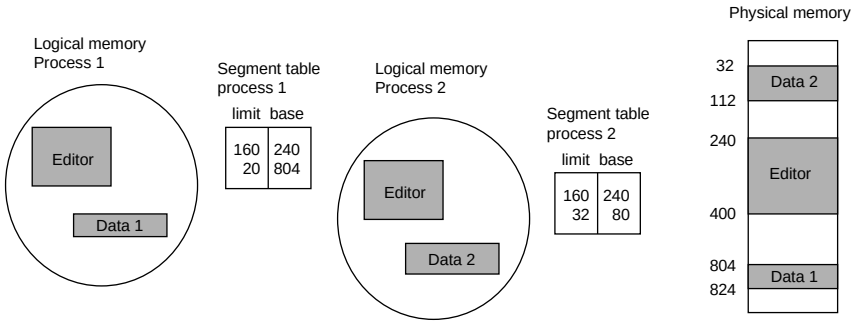
- Each program is divided into blocks of unequal size but with semantics meaning.
- Example: Code, global variables, local variables, stack, …
- Contiguous memory is allocated for each segment for the exact amount needed (first-fit, best-fit).
- A logical address is two dimensional with segment number and offset in segment.
- Each process has a segment table that specify where the segments are allocated.

Basic Segmentation Architecture



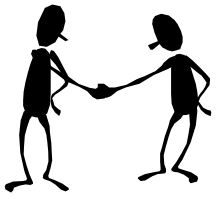
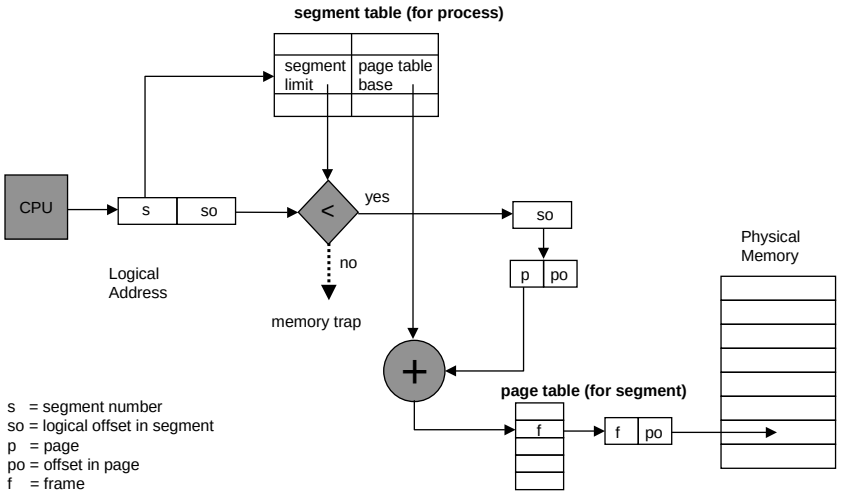
Segmentation Properties

- **Segments are constructed according to user semantics.**
- **This easily enables segment sharing and protection according to user semantics.**
- **External fragmentation exists. Less severe compared with dynamic allocation because average size of each segment is smaller (several in one process).**



Segmentation with Paging

- Exploiting good properties of paging:
  - **No external fragmentation**
  - **faster allocation.**
- Exploiting good properties of segmentation:
  - **Sharing.**
  - **Protection.**
- Paying a small performance cost.
- Many variations exist...



Architecture for Segmentation with Paging