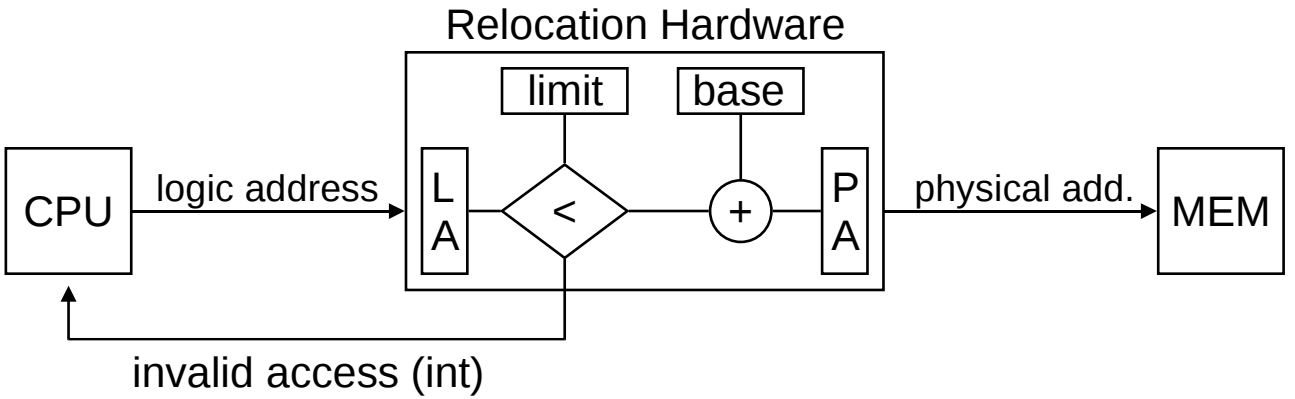
Memory Management

- Processor fetches instructions from main memory
  - Programs must be loaded into memory before they can be executed
- Address referenced by programs (e.g. variables) must be bound to memory
  - Compile-time: absolute addresses
  - Load-time: relocatable code
  - Run-time: relocation hardware
- Programs bigger than memory
  - Overlays are replaced during program execution
  - Might be supported by the OS
- Often replicated programs
  - Can be organized as shared libraries

Single-Process Memory Allocation

- Without OS support
  - Simple dedicated systems (e.g. embedded)
  - No memory manager
- With OS support
  - OS memory is protected through a base register
  - User process is loaded just after OS
- Dynamic relocation with hardware support
  - Compiler and CPU issue *logic* memory addresses
  - Relocation hardware adds logic address to a base address to generate *physical* memory addresses



Multi-Process Memory Allocation without Hardware Support

- List of free memory blocks
  - First-fit: allocates the first block that is large enough to hold the process
  - Best-fit: allocates the smallest block that is large enough to hold the process
  - Worst-fit: allocates the largest available block
- External fragmentation
  - Large amount of small-size blocks
  - Enough free memory to satisfy a request but not contiguously
  - Tend to 1/3 of all the memory for n-fit algorithms

Multi-Process Memory Allocation without Hardware Support

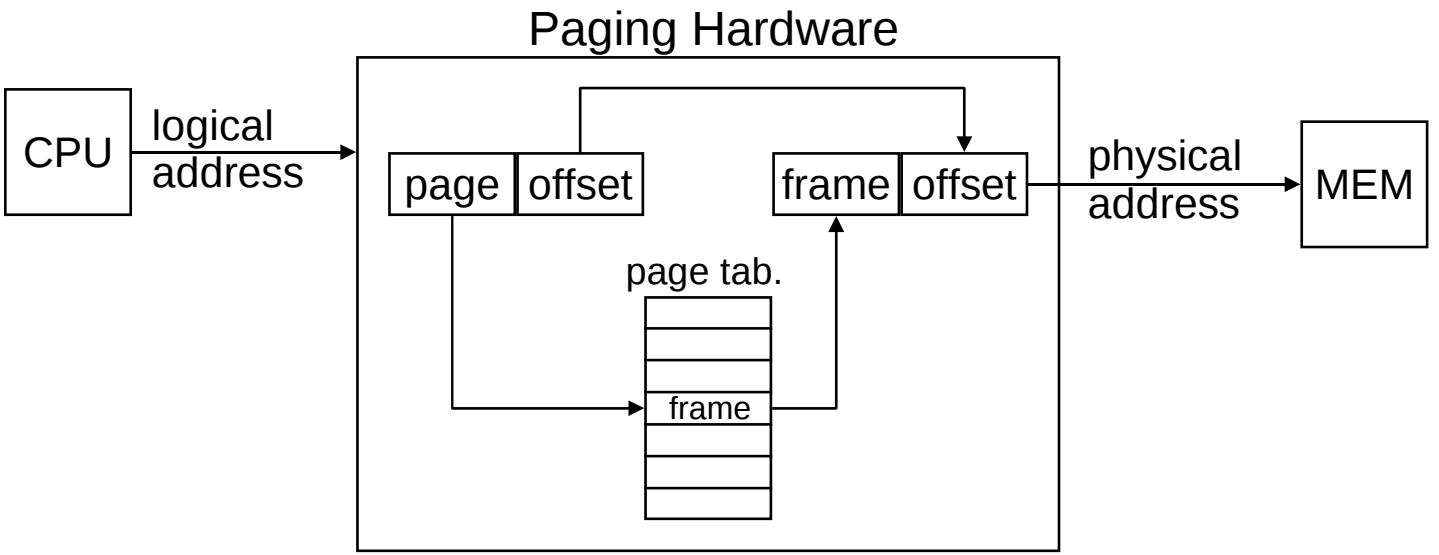
- Protection
  - Through base and limit registers
- Compaction
  - Dynamically relocates processes in order to group free blocks together
  - Relocation support
    - Relative addresses-only (implicitly relocatable)
    - Re-linking by OS application loader (expensive)
    - External relocation support

Multi-Process Memory Allocation with Hardware Support

- Detachment of address space and memory concepts
  - Compilers, processors and processes operate on an address space that is mapped into memory by an MMU
- Memory Management Unit (MMU)
  - Translates logical addresses into physical ones
  - Thus maps processes' address spaces into memory
- Typical strategies
  - Paging
  - Segmentation
  - Paged segmentation

Paging

- Memory organized and allocated in pages
- Processes address spaces
  - Organized in pages
  - Mapped into frames (physical memory pages) through page tables



Paging and Fragmentation

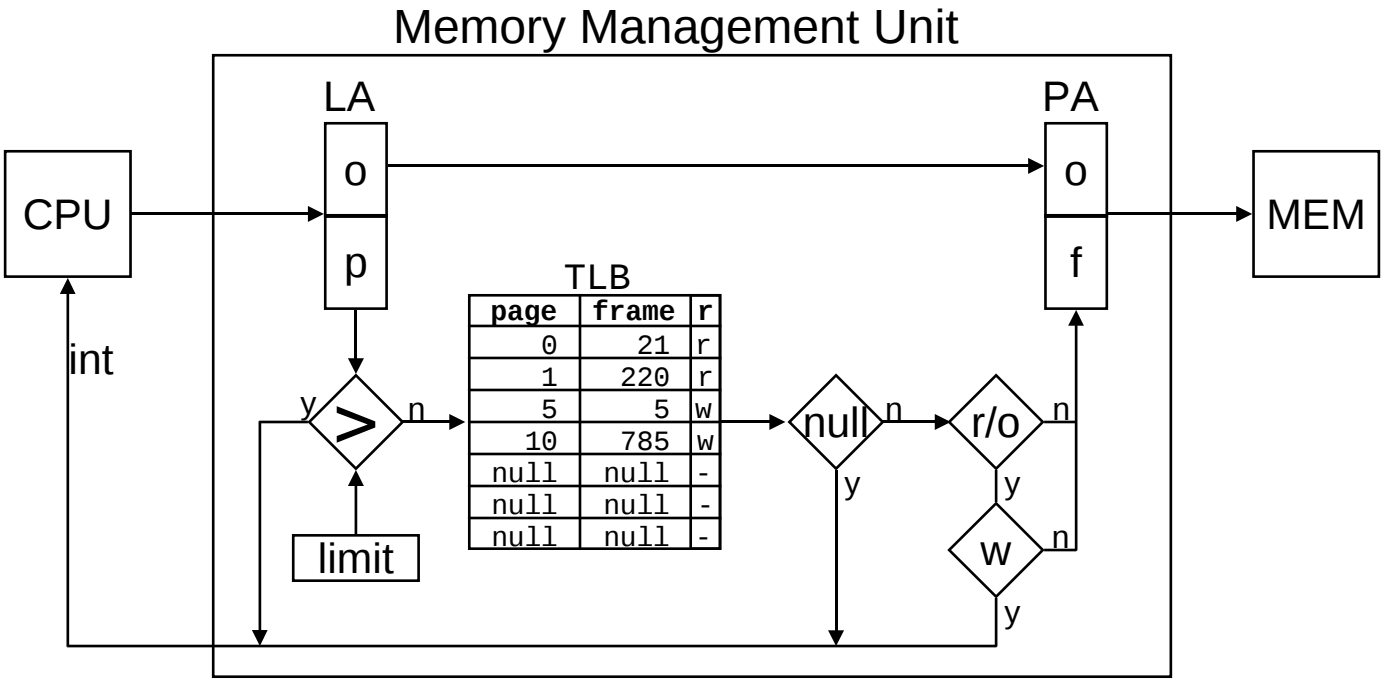
- No external fragmentation
- Internal fragmentation
  - Unused fraction of a page that cannot be allocated to other processes
- Internal fragmentation x page size
  - Small page size -> less fragmentation -> more memory for page tables
  - Example
    - Allocation request for 1 Gbyte
    - Pages of 4 Kbytes -> 256 Kpages
    - Pages of 4 Mbytes -> 256

Paging Implementation

- Page tables
  - Registers: limited to few pages (small address spaces or large pages)
  - Memory: slow (double memory access time)
  - Translation Look-aside Buffer (TLB)
    - Cache of page translations
    - Fast and expensive (fully associative memory)
    - Good performance if hit ratio is high (replacement policy)
- Page sharing
  - Page tables of distinct processes can reference common pages
  - Explored by shared libraries for immutable code
- Protection
  - Pages are tagged with permission bits that are checked by the MMU
  - Limit register to reduce page table size

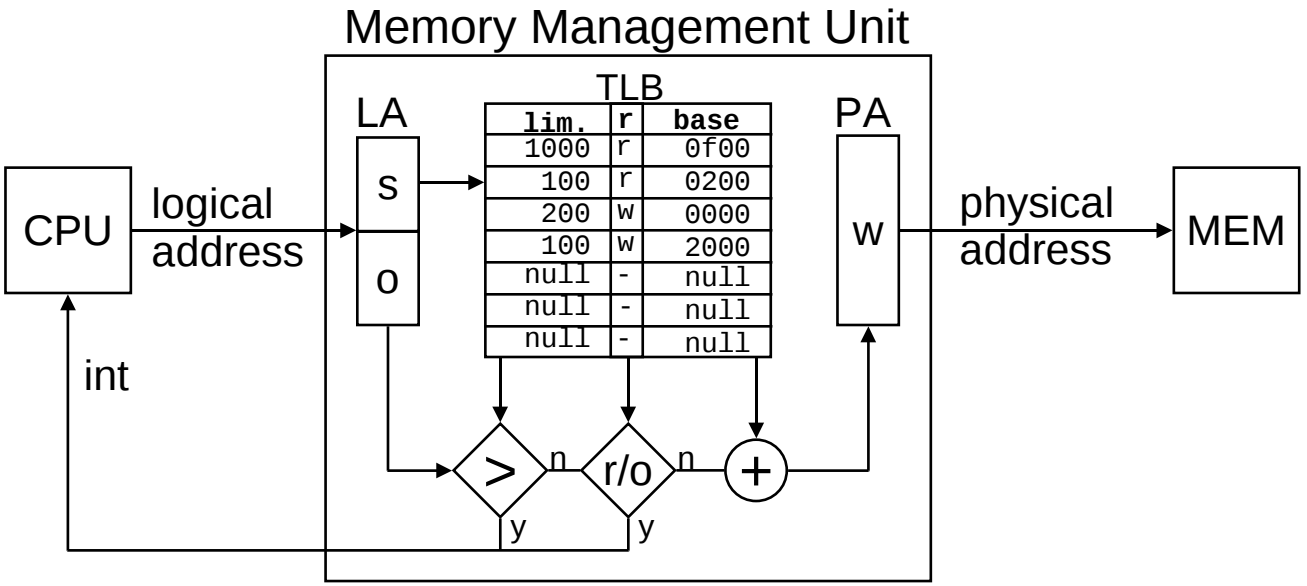
Paging Example

Frame size: 4 Kbytes
Memory size: 4 Gbytes (1 Mframes)
Address space size: 64 Mbytes (16 Kpages)
Physical address (PA): 32 bits (frame = 20, offset = 12)
Logical address (LA): 26 bits (page = 14, offset = 12)
Page table size: 16 Kentries



Segmentation

- Memory organized in segments and allocated in words
- Bi-dimensional addresses: (segment, offset)
- Processes address spaces
  - Organized in segments
  - Mapped into physical memory through segment tables with base and limit for each segment



Segmentation and Fragmentation

- External fragmentation
  - One segment per process -> n-fit (1/3)
  - Fixed-size segments -> paging
  - Word-size segments -> large segment tables and double memory access time
  - One segment per object
    - Intel proposal
    - Not implemented by ordinary compilers
- No internal fragmentation
  - Limit can be adjust to fit used fraction of segment

Segmentation Implementation

- Segment tables
  - Registers: limited to few segments (small address spaces or large segments)
  - Memory: slow (double memory access time)
  - Translation Look-aside Buffer (TLB)
    - Cache of address translations
    - Fast and expensive (fully associative memory)
    - Good performance if hit ratio is high (replacement policy)
- Segment sharing
  - Segment tables of distinct processes can reference common segments
- Protection
  - Segments are tagged with permission bits that are checked by the MMU
  - Segment limits are also checked by the MMU

Paged Segmentation

- Merger of segmentation and pagging
- Address spaces of processes are split in segments, each of which is subsequently paged
  - Segment tables point to page tables