Virtual Memory

Learning Objectives

- Understand the concept of virtual memory
- Understand the concepts of pages, page frames, page faults, virtual address translation
- Understand the design of page tables
- Understand various page replacement algorithms
- Understand the impact of page reference streams, thrashing, and working sets on program behavior

Virtual Memory Techniques

Two methods used for implementing virtual memory:

- **Paging**. The virtual address space is one-dimensional and is broken up into units of one page each. All pages are of the same size. The programmer does not have to be aware of the pages.
- **Segmentation**. The virtual address space is two dimensional: <segment, offset>. Segments can be defined explicitly by the programmer or implicitly by program semantics. Segments are variable sized.

Paging Concepts

- Pages and Page Frames.
- Mapping virtual addresses to physical addresses. The role of MMUs (Memory Management Unit).
- Page Faults: Handled via the interrupt system. Instructions may have to be undone and repeated.
- Page Table design.
  - page table in hardware registers.
  - page table in memory.
  - multi-level page table.

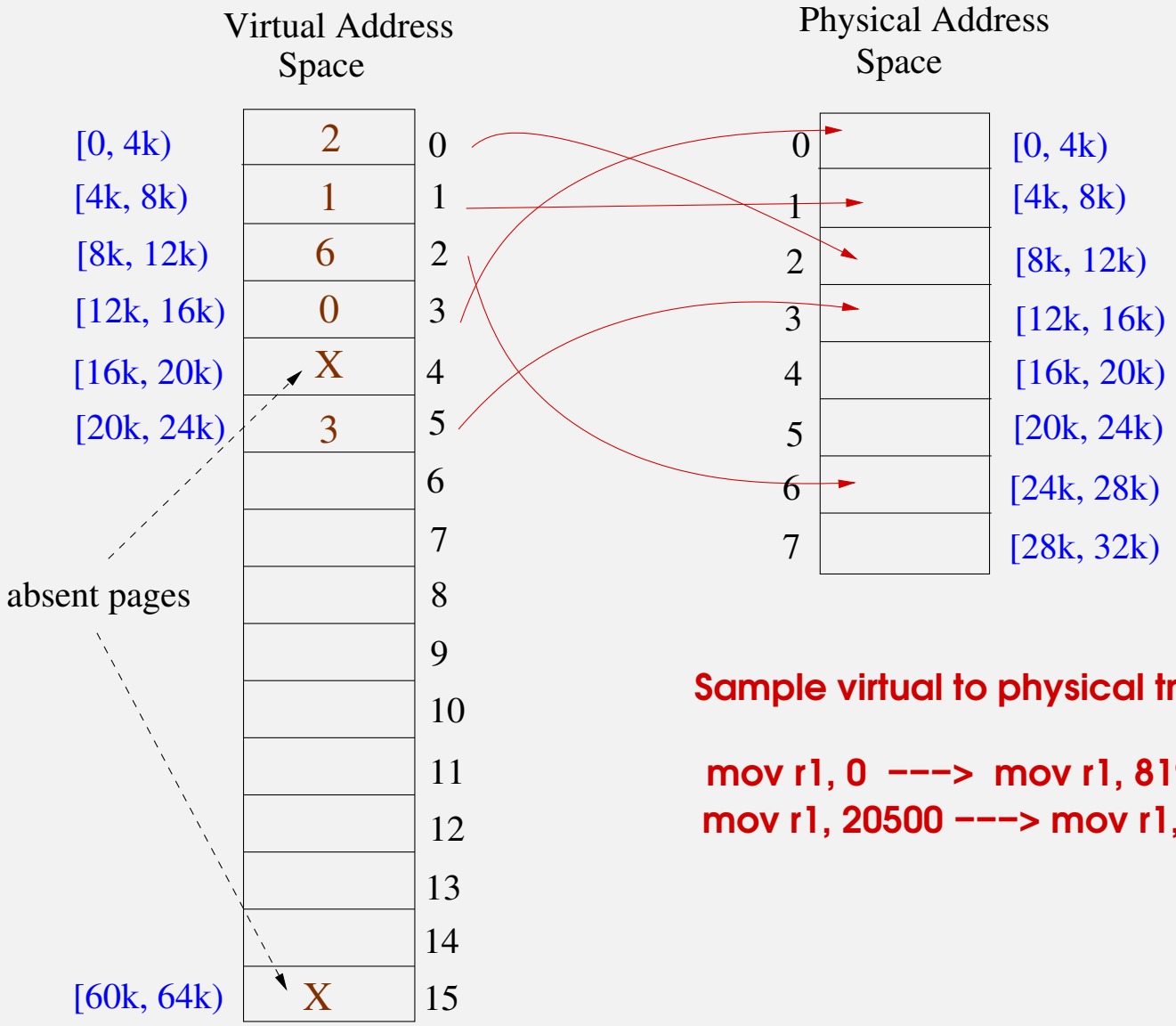
A Simple Example

A detailed simple example of virtual memory. Show the mapping from virtual to physical address space. Explain the functionality of the Memory Management Unit (MMU).

Physical memory = 32K
Virtual memory = 64K
page size = 4K

Number of page frames = 8
Number of pages = 16
Number of bits in physical address = 15
Number of bits in virtual address = 16

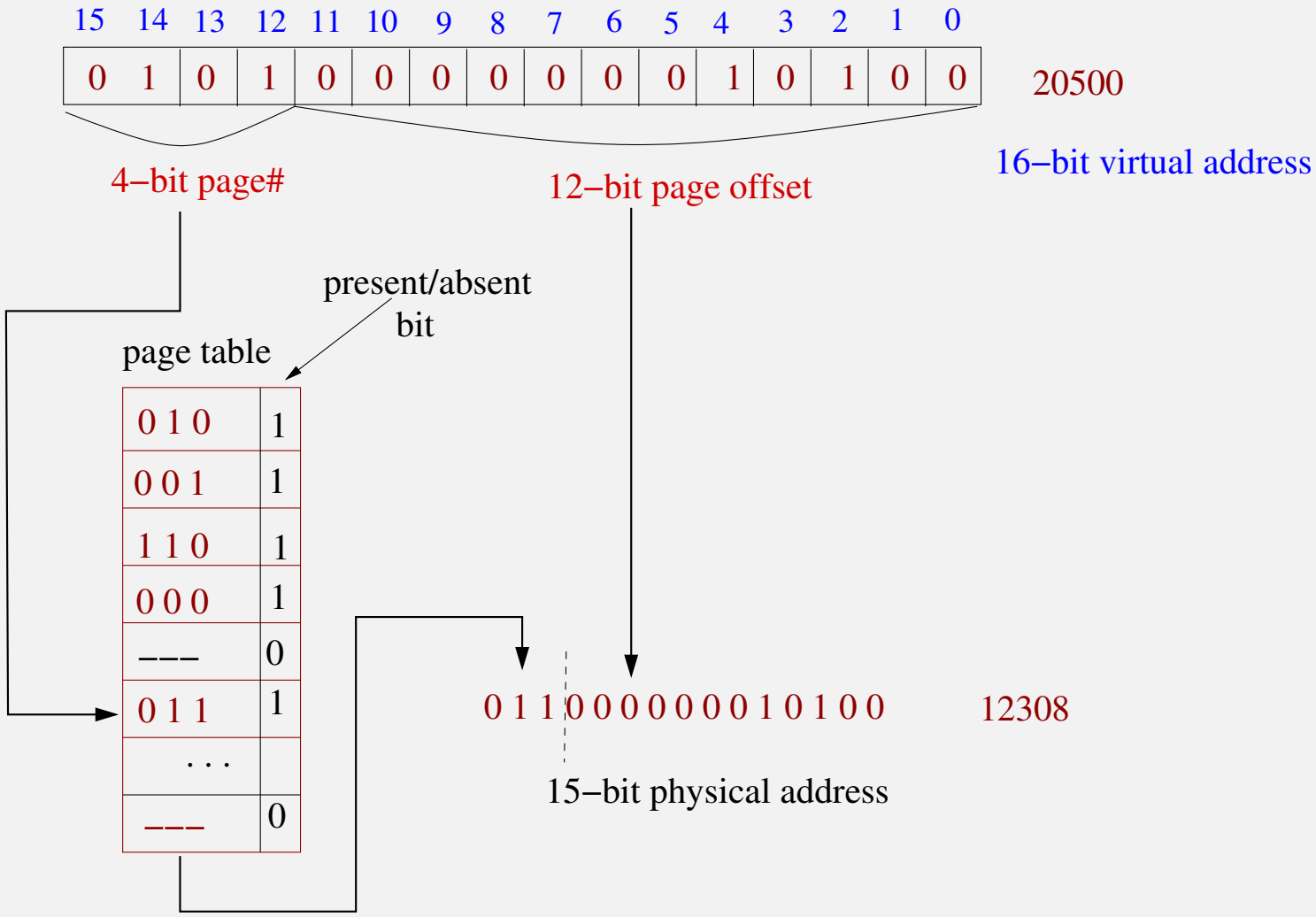
Simple example (contd)



Sample virtual to physical translations

mov r1, 0 ---> mov r1, 8192
mov r1, 20500 ---> mov r1, 12308

Simple example (contd)



Virtual to Physical Address Translation

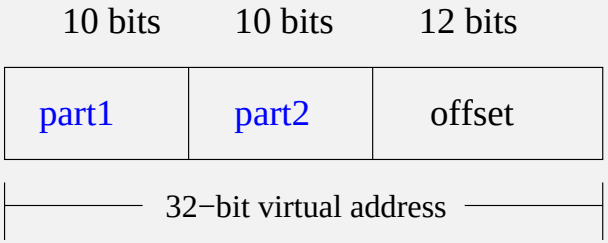
Page Table Design Issues

virtual address space	page size	number of pages
32 bits	4K	2^{20} (about 1 million entries)
64 bits	4K	2^{52} (about 4500 trillion entries!)

- Page tables can be extremely large..it may not be feasible to store the entire page table!
- The mapping from virtual address to physical address using the page table must be fast (since it is happening on every memory reference)
- Each process needs its own page table.

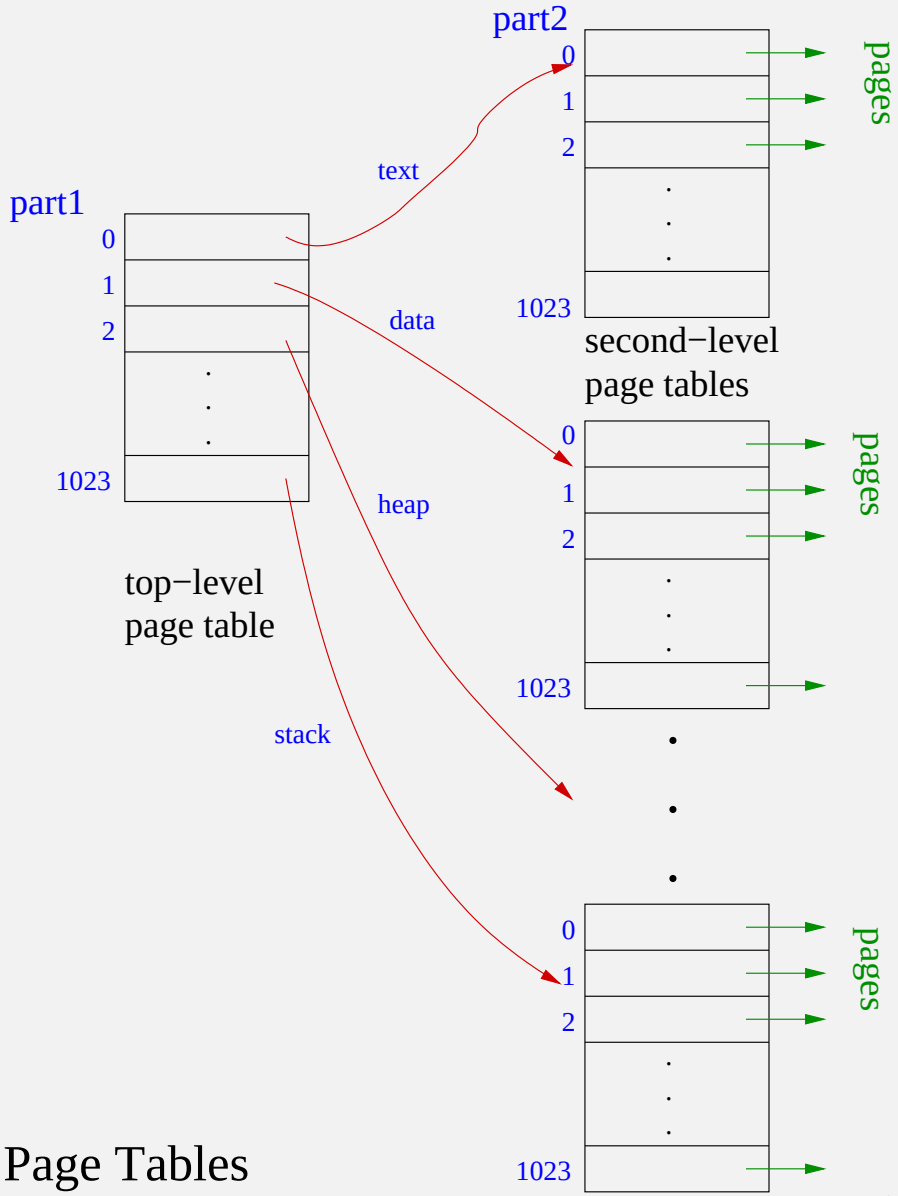
A Range of Page Table Designs

- **Simplest Design**. Page table consists of an array of hardware registers, one per page. The registers are loaded when the process starts so no memory references are needed later. High performance. Context-switch, however, can be expensive.
- **Cheapest Design**. Keep entire page table in memory. Need just one register to point to the start of the page table. Slow performance. Context switching is fast.
- **Multi-level Page Tables**. Split the bits for the page number in the virtual address into multiple fields (usually three or four). The page table is then arranged as a multi-way tree with each node in the tree being a small page table. Far less memory requirements. Combined with a Translation Lookaside Buffer (a cache for virtual to physical address translations), this design gives good performance.



page size = 4K = 2^{12} words

number of pages = 2^{20}



Multilevel Page Tables

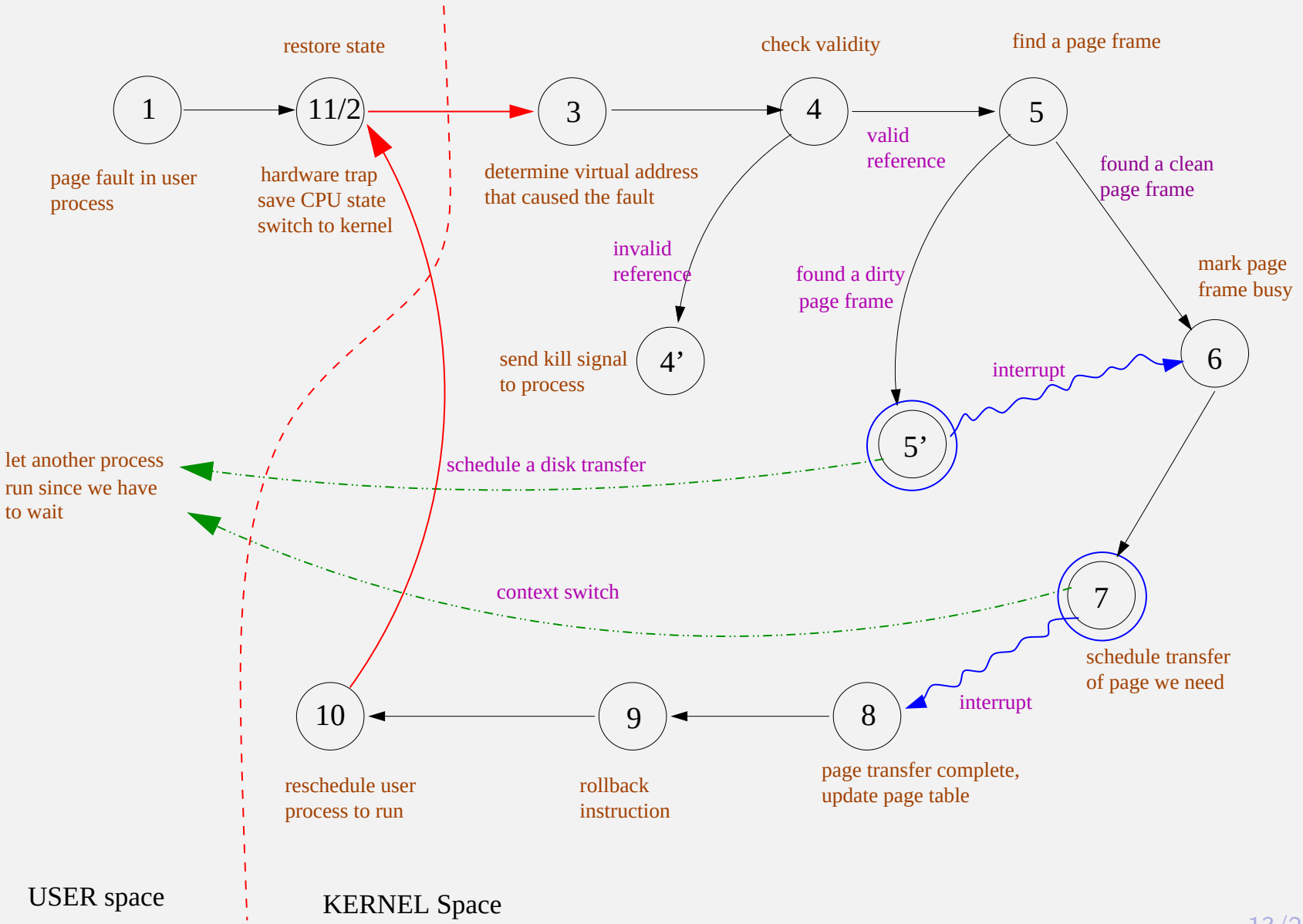
Examples of Paging

- VAX. (2 level virtual page tables) page size=512 bytes, 21 bit virtual page number, 2 bits for memory partitions (00: User, 01: User stack, 10: System, 11: Reserved). (Introduced the concepts of *associative memory* for implementing a *Translation Look-aside Buffer(TLB)* to improve the performance).
- SUN SPARC. (3 level page tables) Page size = 4K, The 32-bit virtual address is broken up into four fields of sizes $8, 6, 6, 12$, with the last field being the offset into the page.
- MIPS R2000. (zero level paging) (32 bit address, 4K page size).

Inverted Page Tables

- If number of pages in the virtual address space >> number of page frames, then it is better to keep track of page frames and which pages are mapped to it rather than a per process page table.
- An inverted page table is always used with an associative memory.

Page Fault Handling



Paging Algorithms

When should a page be *fetched*, which page (if any) should be *replaced*, and where should the new page be *placed*?
Static paging assumes that the amount of memory requested by a process is fixed at the start and does not change.
Dynamic paging algorithms adjust the amount of memory allocated based on the behavior of the program.

Paging Concepts. Page reference stream, demand paging, page replacement algorithms, thrashing.

Measuring performance of virtual memory subsystem: `vmstat` utility on Linux,

Page Replacement Algorithms

- Random Replacement.
- Belady's Optimal Algorithm.
- Least Recently Used (LRU).
- Least Frequently Used (LFU).
- First In First Out (FIFO).

See
http://en.wikipedia.org/wiki/Page_replacement_algorithm
for more details.

Implementation of the LRU PRA

- Exact implementation requires keeping track of the time when the page was last referenced. This is expensive. Instead, approximate schemes are used.
- Use one bit per page which is periodically set to zero. Each time the page is read from or written to, hardware automatically sets the reference bit to one. Inexpensive, though crude, implementation of LRU scheme.
- Use a shift register to keep track of reference information per page. The register contents are shifted to the right periodically. On each reference the most significant bit of the register is set.

Dynamic Paging Algorithms

- Each program usually uses a *working set* of pages. Ensuring that these are in the memory minimizes the number of page faults. Allocating less pages than the size of this set causes many page faults. On the other hand, allocating a lot more pages does not reduce the number of page faults significantly.
- The working set of pages for a process changes over its lifetime. The hard part is to keep track of it and adjust the memory allocation accordingly.

Working Set Clock Algorithm(s)

- *Clock Algorithm*. The page frames of all the processes are logically arranged in a circular list. Each page frame contains a reference bit (used in a way similar to in the LRU algorithm). Behaves like a global LRU algorithm.
- *WSClock Algorithm*. Extension of the basic Clock algorithm by approximating a window size. Each page frame has an additional variable called `lastref`, which is set to the *virtual time* for the process currently using it. To find a page frame the algorithm uses the following equation:

$$Time_{p_i} - \texttt{lastref}[frame] > \tau,$$

where τ is the window size and $Time_{p_i}$ is the virtual time for process p_i.

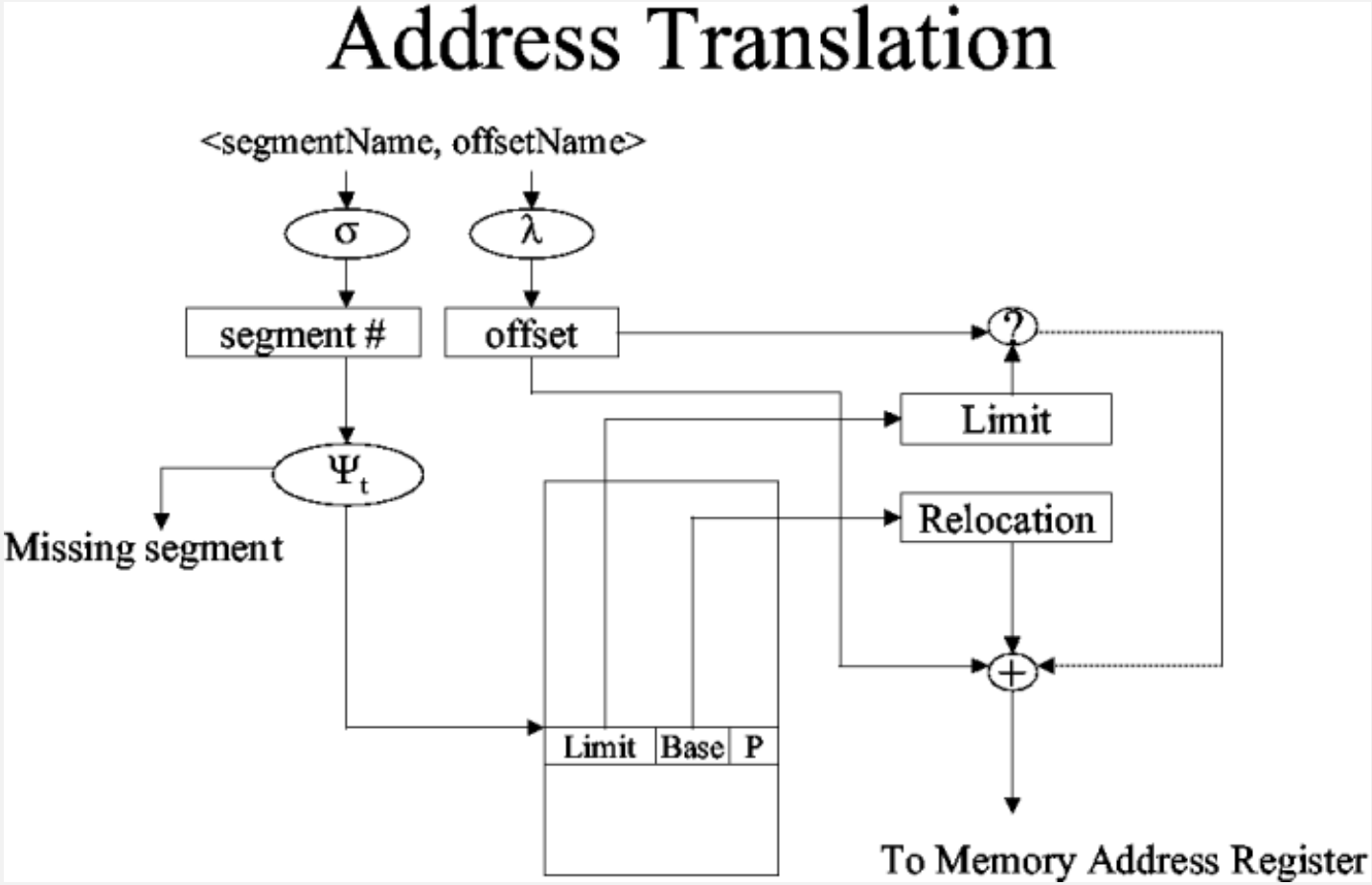
Effect of program structure on paging

See example virtual-memory/page-fault-test.c for significant difference in execution time based on row-major versus column-major access of a two-dimensional array. Abbreviated version shown below.

```
#include <unistd.h>
// make size big enough to cause page faults
#define SIZE 4097
int A[SIZE][SIZE];

void main(void)
{
    int i,j;
    printf(" page size = %d\n", sysconf(_SC_PAGESIZE));

    for (i=0; i<SIZE; i++)
            for (j=0; j<SIZE; j++)
                    A[i][j] = 0;

    for (j=0; j<SIZE; j++)
            for (i=0; i<SIZE; i++)
                    A[i][j] = 0;
}
```

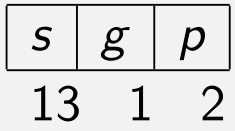
Segmentation

- Segmentation provides a two-dimensional virtual address space. A program can consist of several independent segments, each of which can grow or shrink independently.
- Many systems implement segmentation and paging simultaneously by paging the segments.


Address Translation
<segmentName, offsetName>
σ
λ
segment #
offset
?
Limit
Relocation
Ψt
Missing segment
Limit
Base
P
To Memory Address Register

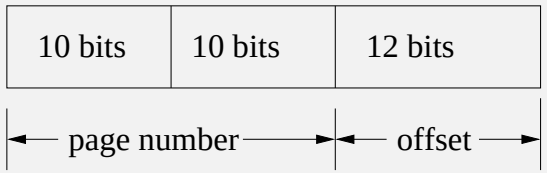
Intel x86 Segmentation

- Uses segmentation combined with paging for memory management.
- Maximum number of segments per process is 16K, and each segment can be as large as 4 GB. The page size is 4KB.
- The logical address space of a process is divided into two partitions. The first partition consists of up to 8K segments that are private to the process. The second partition consists of up to 8K segments that are shared among all processes.
- The **local descriptor table (LDT)** keeps track of the private segments. The **global descriptor table (GDT)** keeps track of the shared segments. Each entry in these tables is 8 bytes long, with detailed information about a particular segment including the base location and length of the segment.
- The **logical address** is a pair (selector, offset), where the selector is a 16-bit number, and the offset is a 32-bit number.



Intel x86 Segmentation (continued)

- The machine has six segment registers, allowing six segments to be addressed at any one time by a process. It has six 8-byte registers to hold the corresponding descriptors from either the LDT or the GDT.
- The logical segment address is 48 bits: (16-bit segment, 32-bit segment-offset). The base and limit information about the segment in question is used to generate a 32-bit **linear address**. First the limit is checked for the validity of the address. Then the base is added to the segment-offset, resulting a 32-bit linear address (that is still virtual). In the the next step, the 32-bit linear address is converted into a physical address using paging as described below.
- Page size of 4KB. A two-level paging scheme: the first part is 10 bits, the second part is 10 bits and the least significant 12 bits are for the offset within a page.



Intel x86 Translation

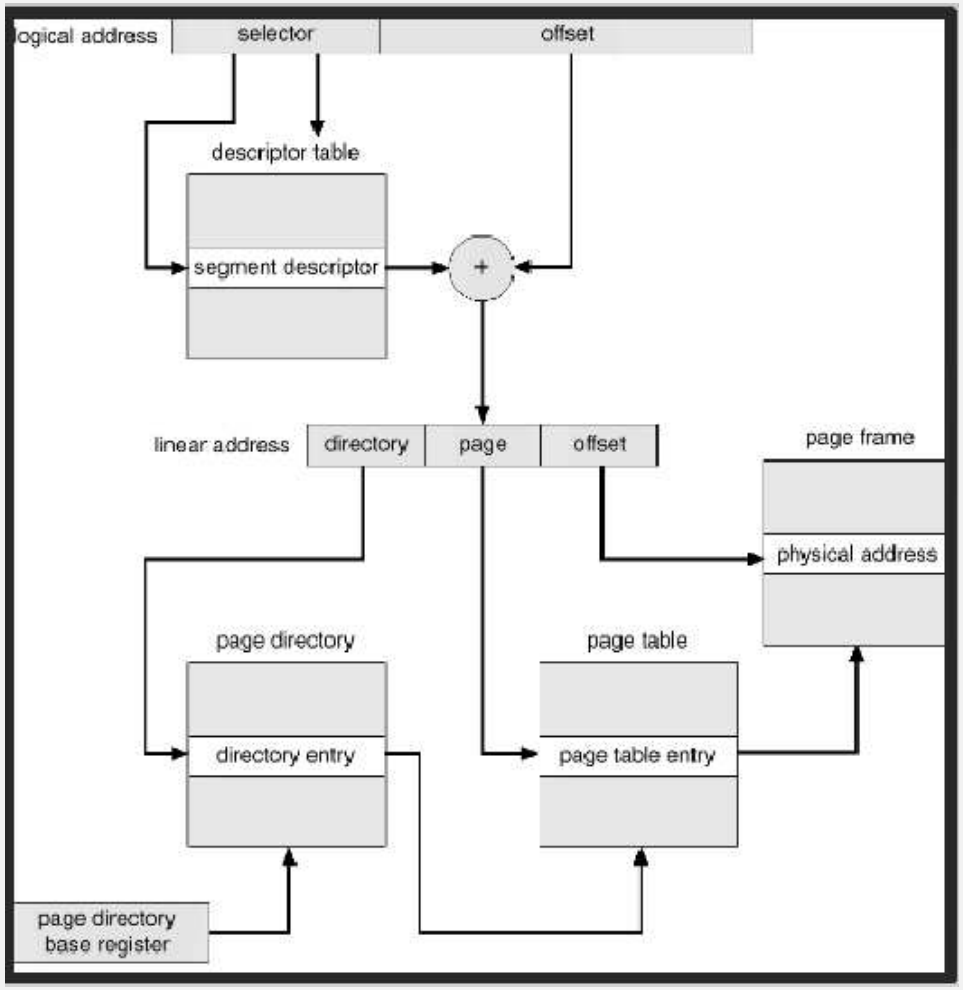


Diagram from *Operating System Concepts, 6th Ed.* by Silberschatz and Galvin

64-bit extensions to 32-bit x86 Architecture

- Initiated by AMD and named as x86-64 (later *AMD64*). Intel adopted it later and named it *EM64T* (renamed to *Intel 64*).
- Current features:
  - 64-bit registers and pointers.
  - 48-bit virtual address but can be extended in future.
  - 40-bit physical address space. Newest version is 48-bit physical address space but can be extended to 52-bits.
  - Legacy 32-bit code can run without recompilation or performance hit. But converting to 64-bit does enhance performance.
  - PAE (Physical Address Extensions) mode for legacy software has increased from 36 bits to 52-bits.
  - Segmentation support only in 32-bit legacy mode.
  - Page size can be 4KB, 2 MB or 1GB.
  - Four-level page table for 48-bit addresses. Each level is 9-bits and page offset is 12-bits (or more for larger page sizes)

Paging versus Segmentation

	Paging	Segmentation
Number of address spaces	1	>2
Total address space > physical space	Yes	Yes
Has separate data, text, stack segments	No	Yes
Programmer awareness	No	Yes
Easy to accommodate fluctuating tables	No	Yes
Easy to share memory	No	Yes