Virtual Memory

COMP375 Computer Architecture and Organization

Cache and Android Performance

- A recent email mentioned how a programmer had to be concerned about how the cache is accessed in a multi-threaded Android program

“Because of the cache layout, multiple threads writing to the same array risk performance degradation due to a potential high level of cache thrashing.”

Goals for Memory Management

- Have the programs in memory when they need to run
- Convert program addresses to physical addresses.
- Protect one program from another
- Allow programs to share data and instructions
- Support read-only or execute-only segments

Sub Goals

- Fast address translation
- Simple for the OS to allocate and release memory
- Minimize fragmentation

This is an example of a bin packing problem and is **NP-complete** even if you have all of the memory requests at one time.

Program to Physical

- Everybody's program is compiled and linked with the first byte of the program at address zero
- Since many programs are run simultaneously on the computer, each cannot be loaded starting at physical address zero
- The hardware converts each memory reference from a program address to a hardware physical address

Physical Mapping Schemes

- Base Address
- Paged Memory
- Virtual Memory

- Segmented addresses
- Flat single address scheme

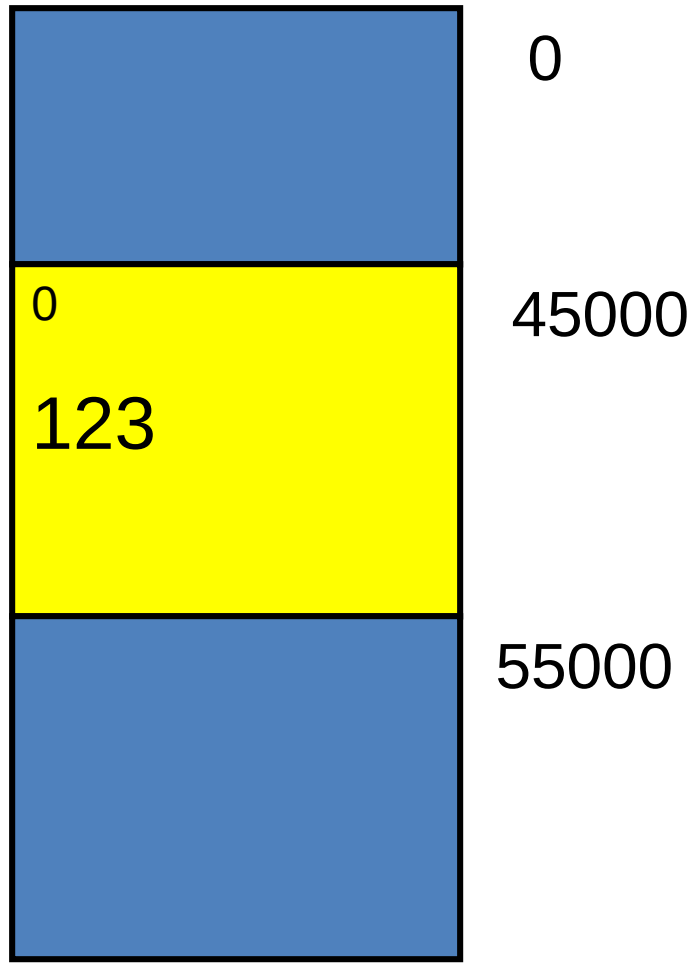
Base Address

- Programs are located in any convenient contiguous part of memory
- When a program is to execute, the base register is loaded with the start address of the program
- The program effective address from each memory reference is added to the value in the base register to create the physical memory location

Base Address Translation

Base Address Register

45000



Program address 123 is located in RAM at the real physical address 45123

Limit Register

- The limit register contains the highest address of the program
- The memory system checks each memory reference to make sure it is not greater than the value in the limit register
- An addressing exception interrupt occurs if the address is too large

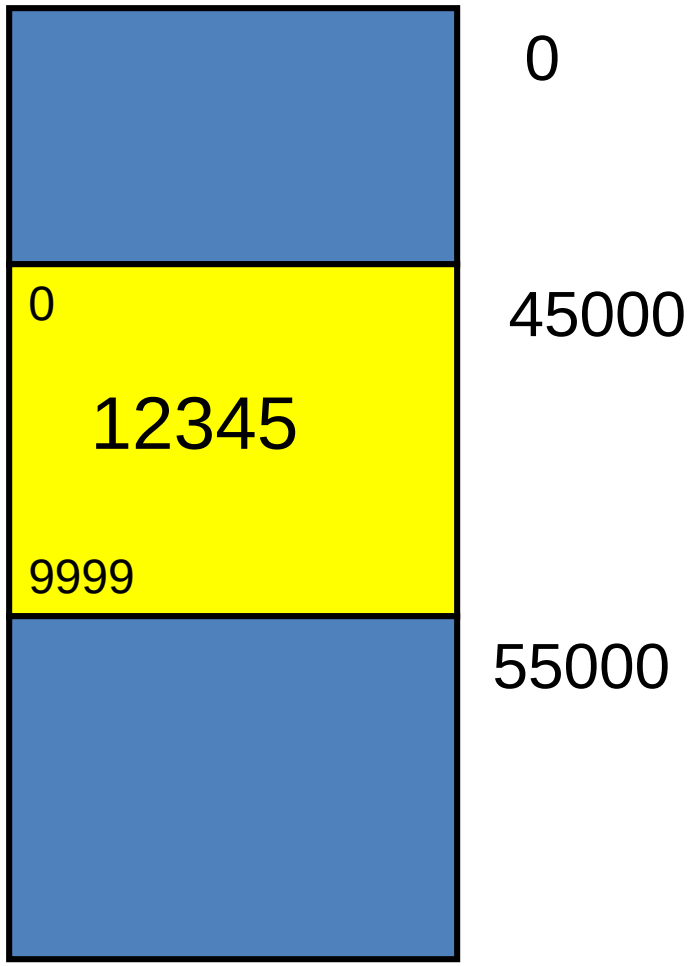
Base Address with Limits

Base Address Register

45000

Limit Register

10000



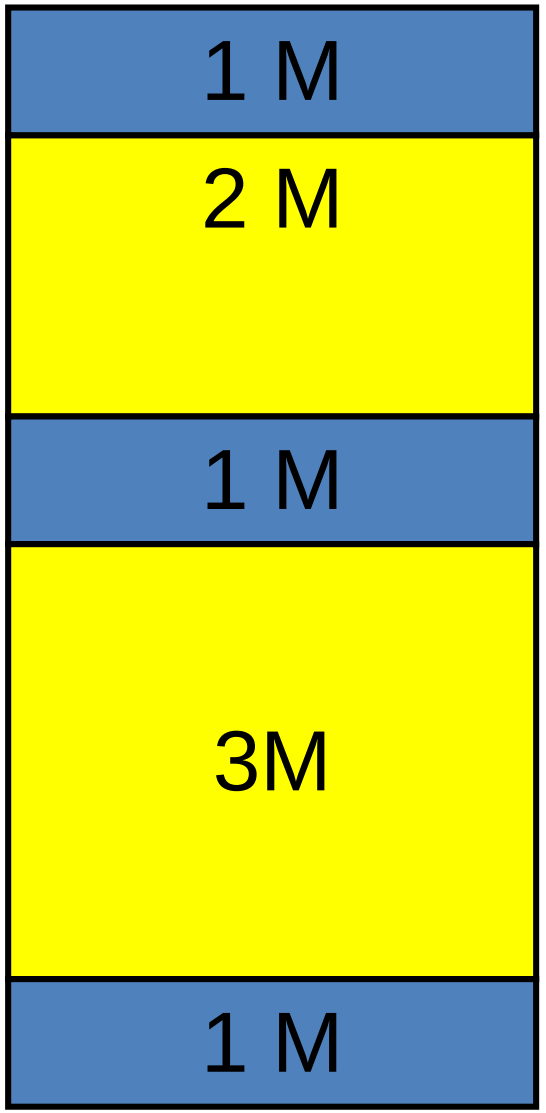
Program address 12345 is too large

Fragmentation

- As memory is allocated and released, the usable areas become fragmented
- **Internal fragmentation** occurs when a program is allocated to a memory area that is bigger than needed. The leftover is lost due to internal fragmentation
- **External fragmentation** occurs when there are many small unallocated areas. These areas are hard to use because they are small and divided

Fragmented Memory

- In this memory diagram two programs use 5 MB of the 8 MB in the system
- There is 3 MB available
- A 2 MB program cannot be put in memory because the available space is fragmented



If you allocate memory in 1K blocks, what is the maximum internal fragmentation per program?

A. 4 GB
B. 1 KB
C. 1023 bytes
D. 256 bytes

Evaluation of Base Register

- Convert program addresses to physical addresses
  - Simple and efficient
- Make it easy for the operating system to place the program in memory
  - Programs have to be located in contiguous memory. It can be a challenge to fit the programs into the available space

Evaluation of Base Register

- Allow sharing of data and instructions
  - Each program is separate. No sharing
- Detect out of range memory accesses.
  - Simple and efficient
- Provide read-only and no-execute memory segments
  - All of memory is treated the same

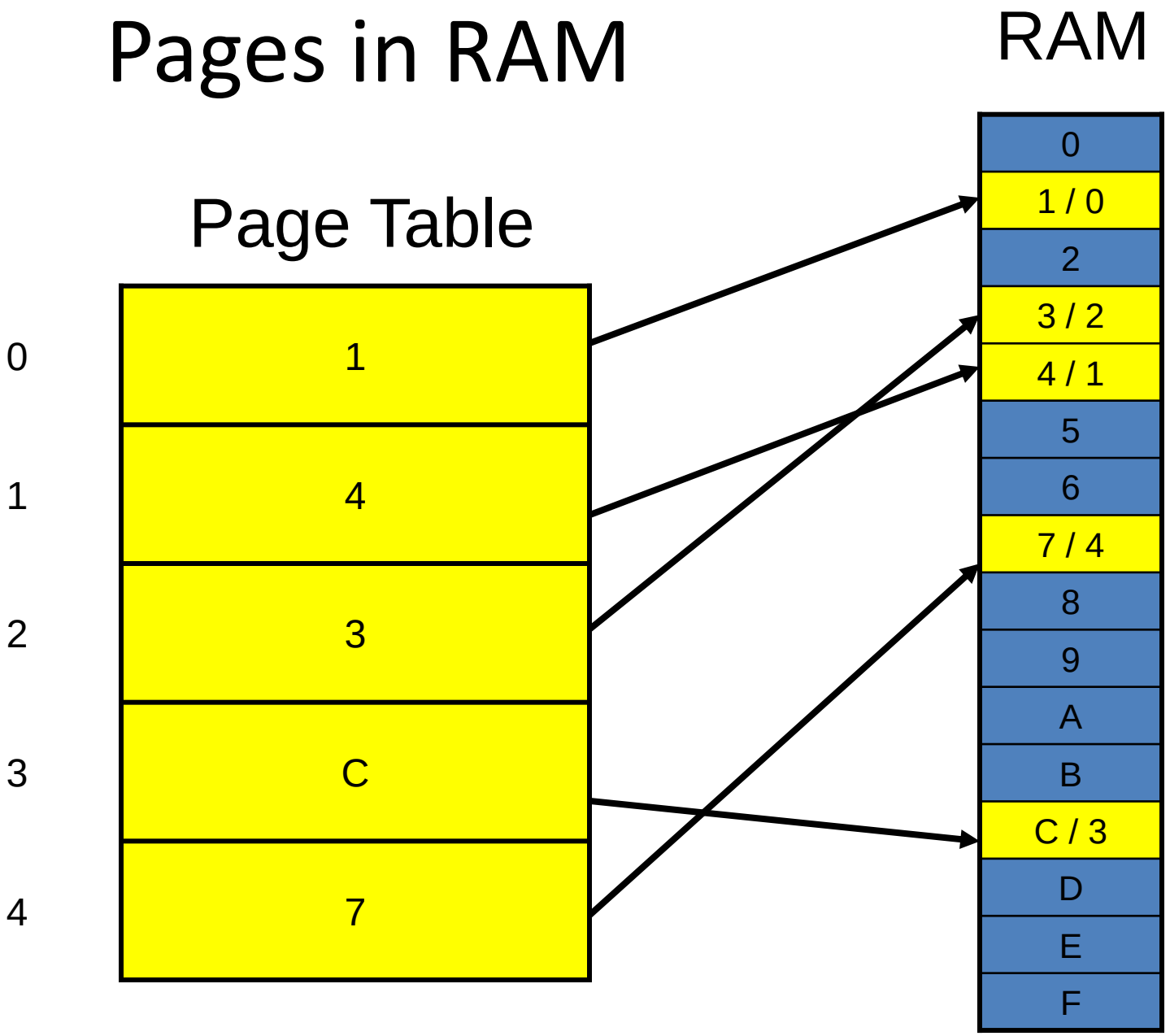
Using the Base Register Method

- Some ancient OS set the base register to certain fixed addresses to create fixed sized areas for programs. This was not very effective.
- Better OS put the programs wherever they would fit and set the base register to the beginning of the program

Paged Memory

- RAM and programs are divided into fixed sized pages
- The page size is usually fixed for a given architecture, often between 512 -8K bytes
- The pages of a program can be put anywhere in RAM. They do not have to be contiguous.
- The page table keeps track of the physical location of pages
- The page table is indexed by the page number portion of a program address

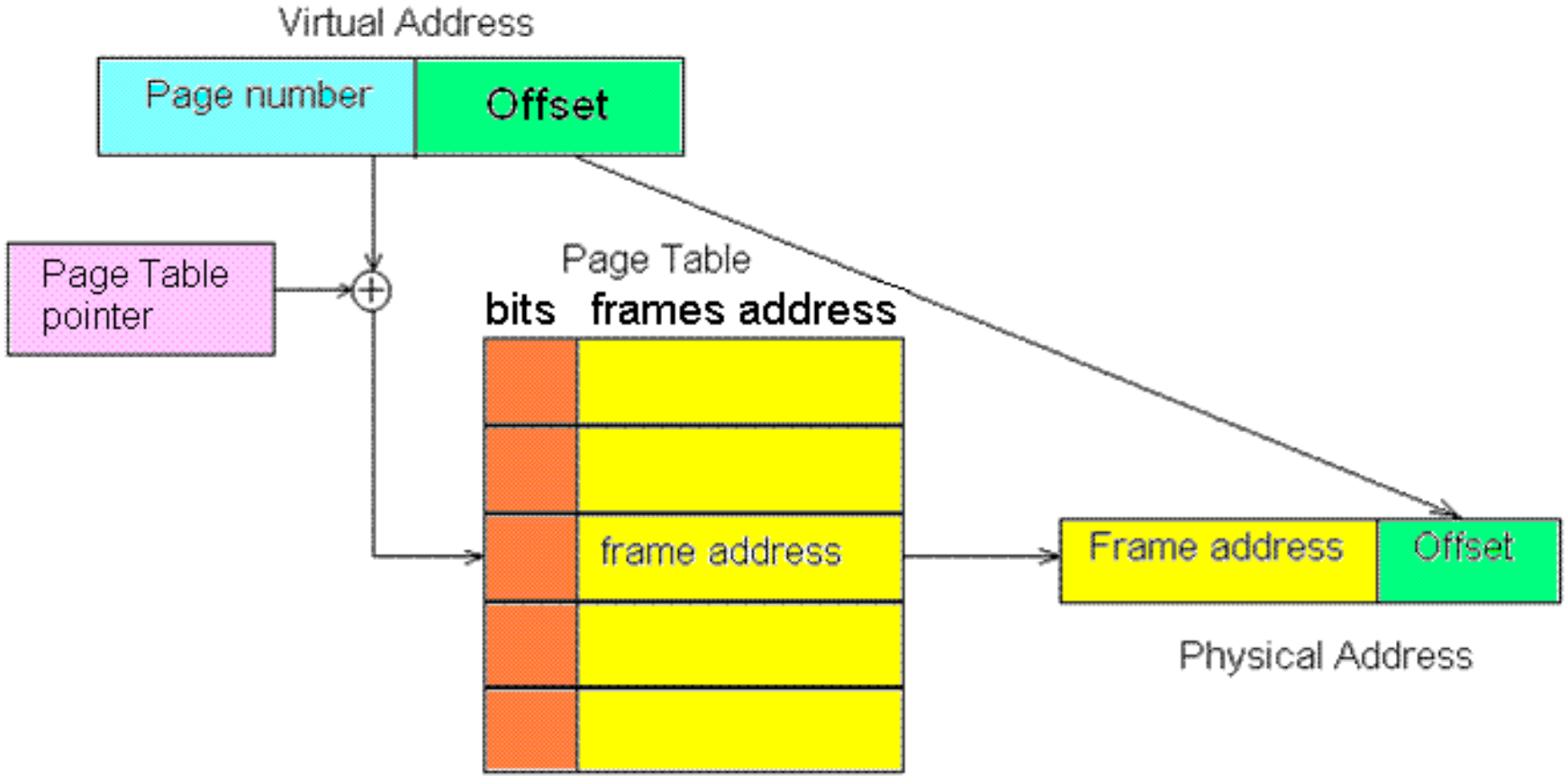
Pages in RAM



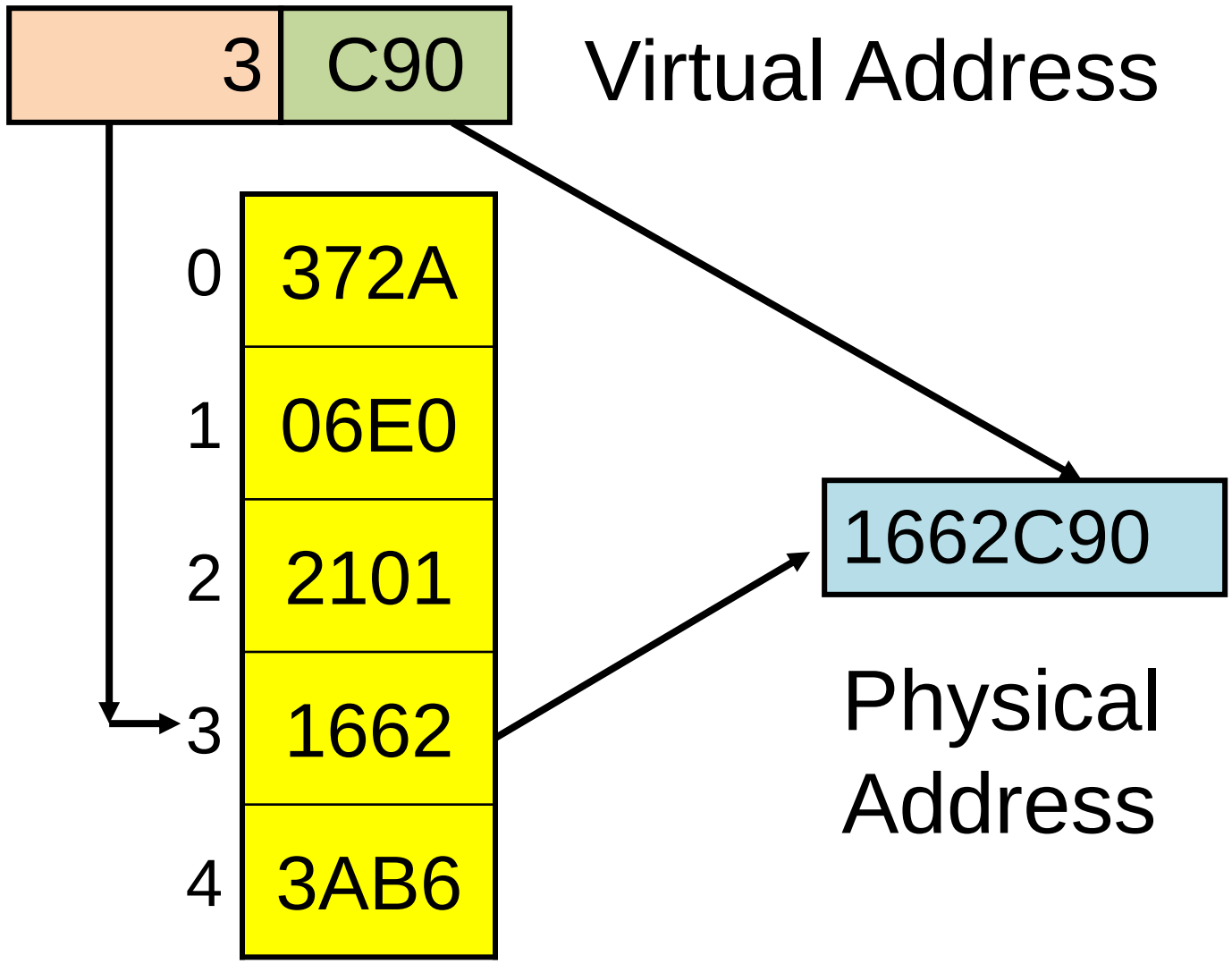
Virtual to Physical Address

- The upper bits of a program address are used as an index into a page table
- Each page table entry contains the physical address for that page of the program
- The lower bits of the program address (which indicate which byte in the page is desired) are concatenated to the end of the physical page address

Address Translation



Example with 4K pages



Dividing the Address

- The offset portion of the address is always the lower $\log_2$(size of a page) bits of the address
- The page number portion are all of the address bits that are not part of the offset

Fragmentation

- When using paging, programs can be loaded into RAM without worrying about fragmenting memory into small pieces. There is no **external fragmentation**.
- If the program size is not an even multiple of the page size, the last page will only be partially filled. This is **internal fragmentation** With large memories this is unimportant.
- Paging makes much better use of RAM

How many bits are used for the page number and offset?

Assume **1 GB** address space and 2K byte pages.

A. page# = 21 offset = 11
B. page# = 22 offset = 10
C. page# = 19 offset = 11
D. page# = 20 offset = 12

Big Programs

- Even when using every byte of RAM, it is not always possible to load all the programs users would like
- Frequently large parts of programs are never executed. There are many features of Microsoft Word® you have never used.
- More programs could fit in memory if only the used portions were loaded into RAM

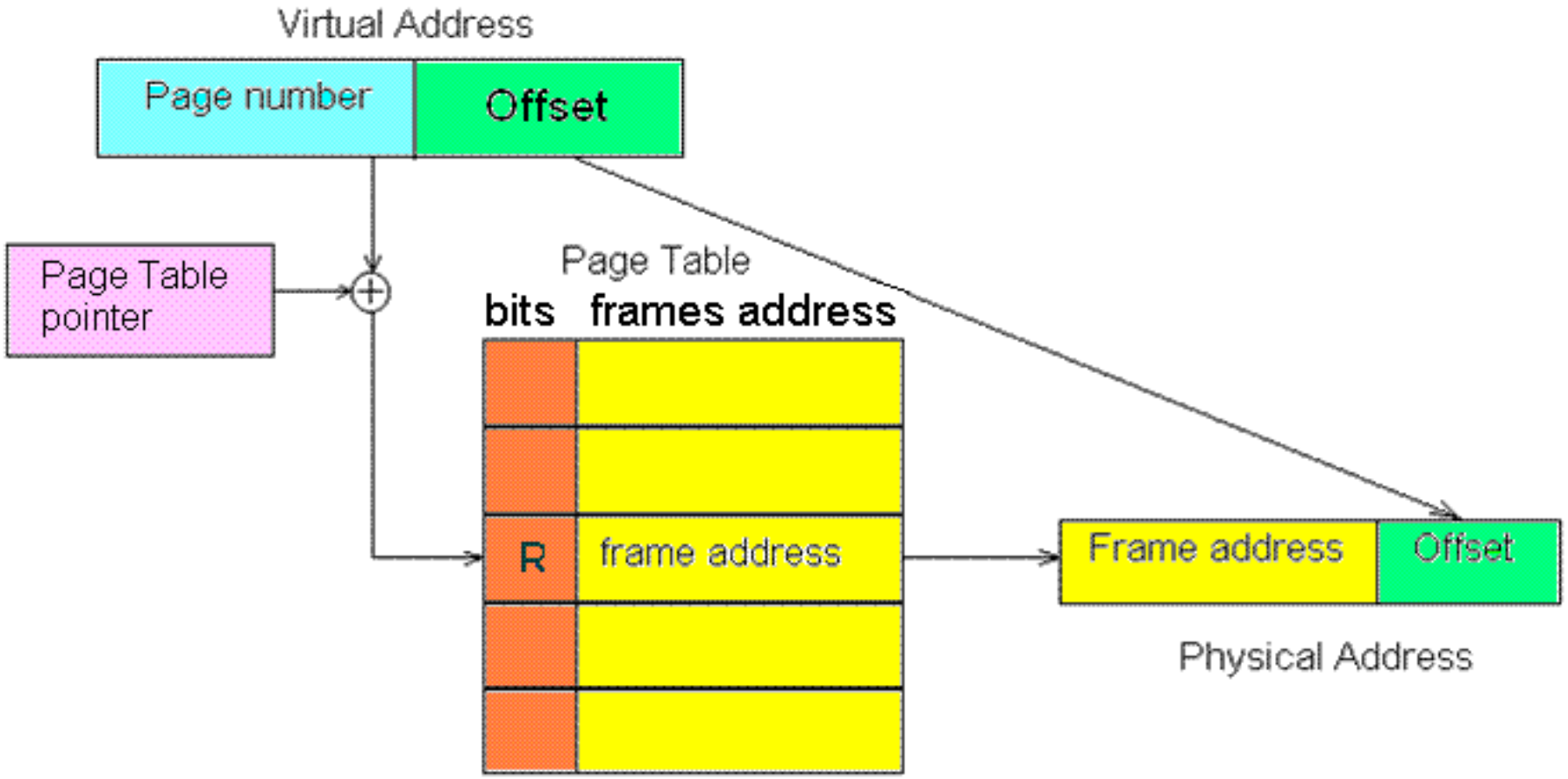
Virtual Memory

- Virtual Memory is an extension of paging
- Only the pages that are being used are in RAM
- A copy of all pages of a program are on the page file
- If a program accesses an address in a page not in RAM, the hardware creates a page fault interrupt and the OS copies the desired page into RAM

Virtual Memory Implementation

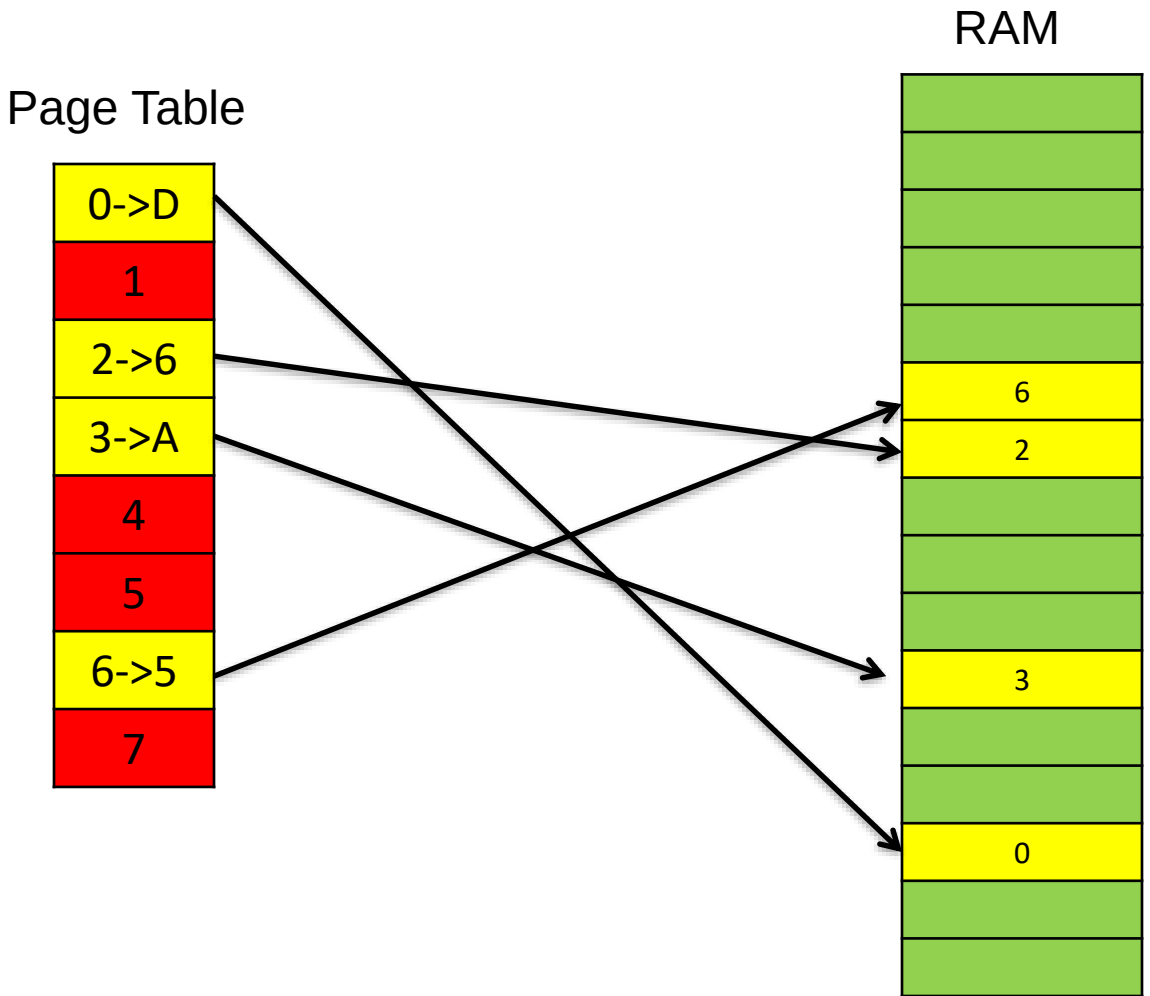
- Unused pages of a program do not need to be in RAM to execute the program
- A “resident” bit is added to the page table
  - Pages in RAM have the resident bit set
  - Pages not in RAM have the resident bit cleared
- All pages are stored on disk
- When a program references a page with the resident bit clear, the hardware creates a page fault interrupt

Address Translation



Only Necessary Pages in RAM

Program addresses showing pages that are referenced



Hardware and Operating System

- Virtual memory requires both hardware and operating system support
- When a page fault interrupt occurs, the OS reads the desired page from disk into an available page of RAM
- The user’s page table is updated to point to the newly loaded page
- The program is placed back on the ready list to be executed

User Programs and Virtual Memory

- Virtual memory is transparent to user programs
- Programs are unaware of page faults
- The only impact of virtual memory on user programs is performance
- If a program creates frequent page faults, the program will run very slowly

Page Table Flags

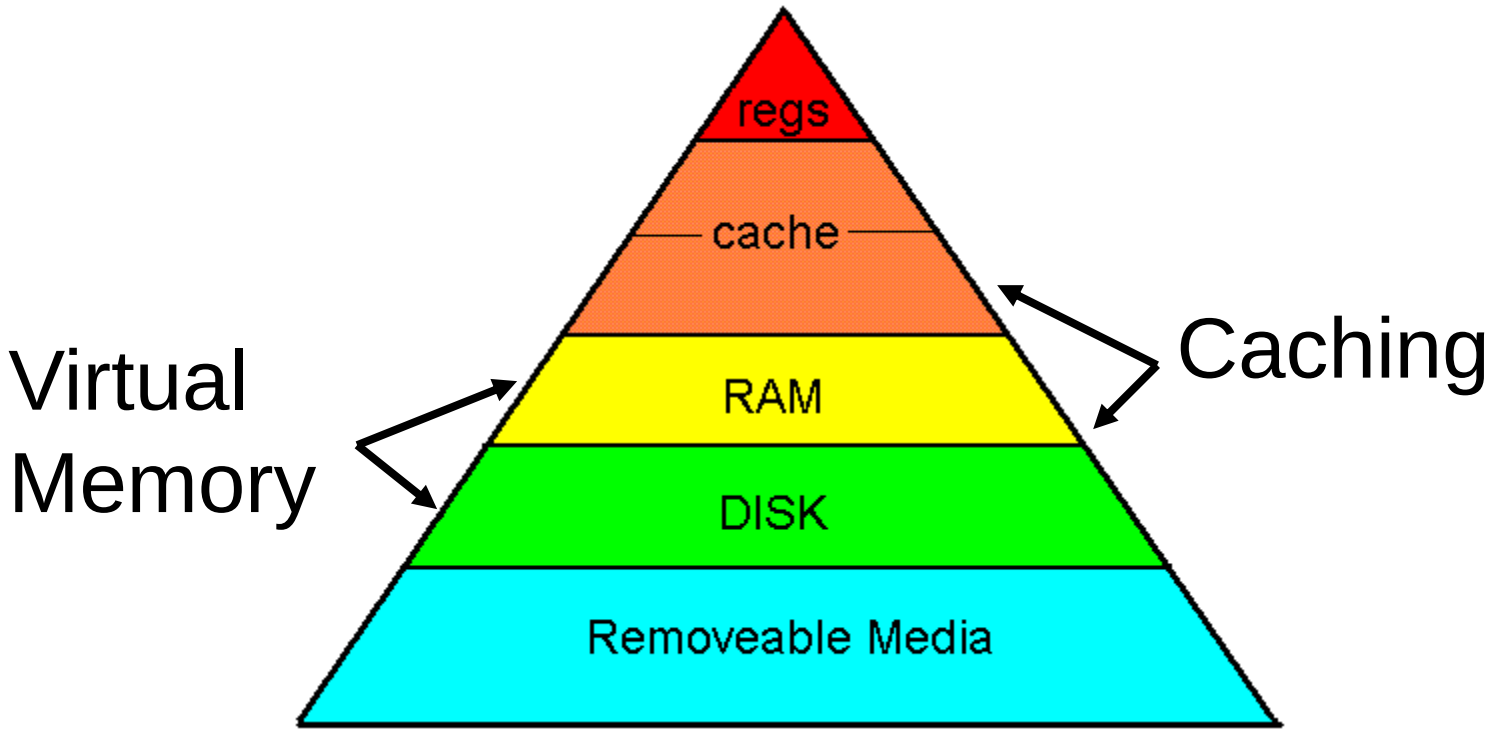
- Each page table entry has flag bits
  - **Resident** – set if the virtual address is in RAM and clear otherwise.
  - **Used** – set when the page is referenced
  - **Dirty** – set when the page is changed
  - **No Execute** – Instructions cannot be fetched from this page
- The Used and Dirty bits are set by the hardware and cleared by the OS

Locality of Reference

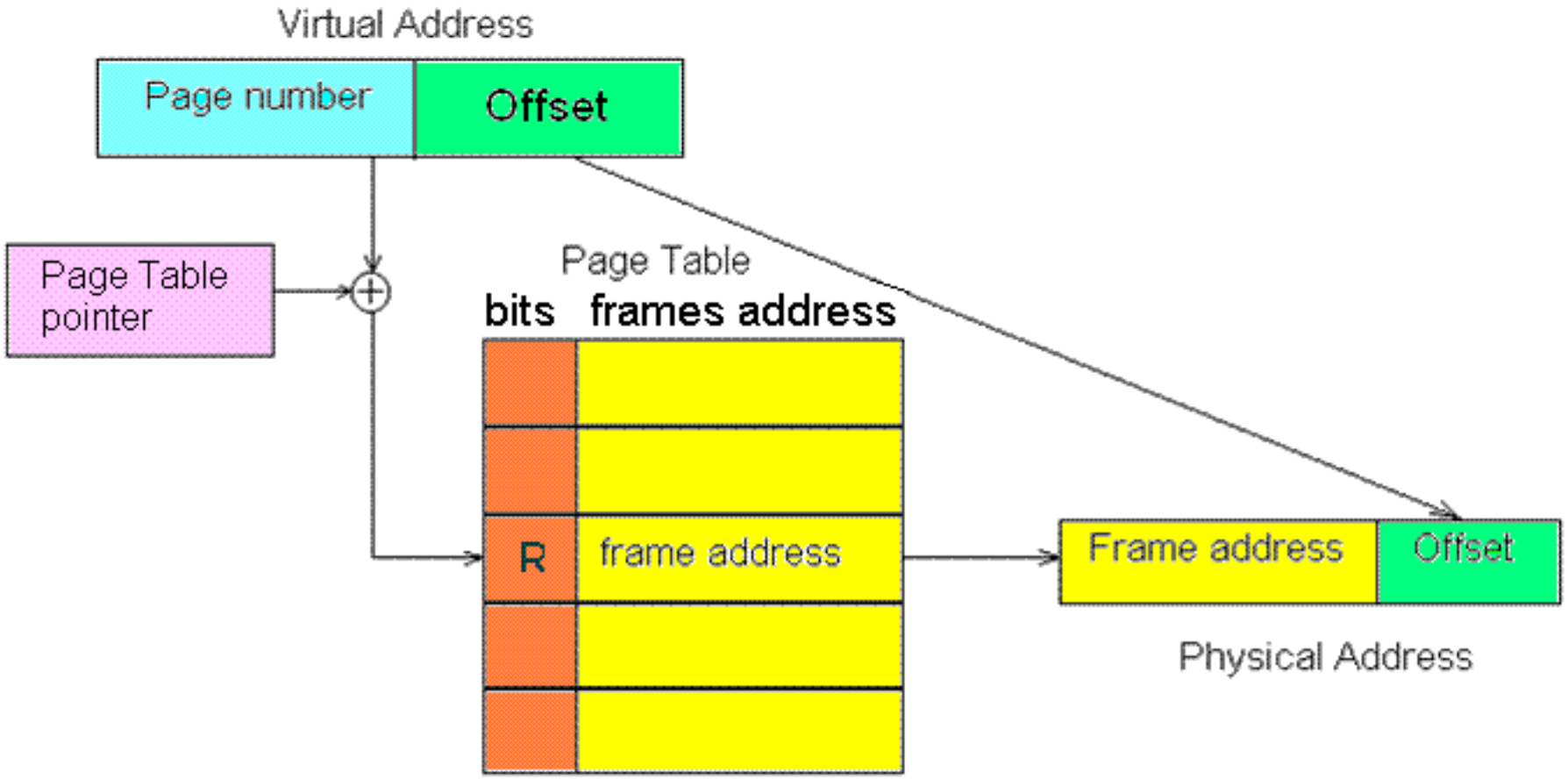
- **Temporal locality** - a referenced location is likely to be referenced again

- **Spatial locality** - nearby locations are likely to be referenced soon

Memory Hierarchy



Address Translation



Translation Lookaside Buffer

- The Translation Lookaside Buffer (TLB) is a special cache to hold recently used page table entries
- The CPU looks in the TLB for a page entry before looking in the page table
- Without the TLB each memory reference would require two memory accesses
- Most TLBs are small and use fully associative mapping

Performance Factors

- The hardware handles the normal address translation
- When a page fault occurs, the OS determines where the page will be loaded and what page will be overwritten
- The OS must minimize the number of page faults
- The OS determines which and how many pages of a program are in RAM

Virtual Memory Performance

- Computers can retrieve a word from RAM in about 60 ns (6×10^{-8} sec)
- A good disk drive can read a block of data in about 6 ms (6×10^{-3} sec)
- If the needed page is not in RAM and has to be read from the disk, it takes about 100,000 times longer to get the data.
- Page fault interrupts have to be kept to a minimum.

A High Hit Rate Is Important

- Consider RAM in access in 60 ns and disk drive access in 6 ms
- If the hit ratio is 99.99%

$$\text{avg access} = 0.9999 * 6 \times 10^{-8} \text{ sec} + 0.0001 * 6 \times 10^{-3} \text{ sec} = 660 \text{ nsec}$$

- The average memory access time is about 11 times slower

What is the average access time for 99.9% hit rate with 9.0 ms disk and 9.0 ns memory?

A. 8.1 ms
B. 8.1 ns
C. 9.0 ms
D. 9.0 µs
E. 9.0 ns

Program to Physical Translation Goals

- Convert program addresses to physical addresses
  - More complicated than base address. May require two memory accesses.
- Make it easy for the operating system to place the program in memory
  - Much easier than base address
- Allow sharing of data and instructions
  - Pages can be shared between programs

Program to Physical Translation Goals

- Detect out of range memory accesses
  - If the page table index is too large, a segmentation fault interrupt with occur
- Provide read-only and no-execute memory segments
  - Different pages can be marked read-only or no-execute

Virtual Memory Advantages

- Allows you to fit many large programs into a relatively small RAM
- Only part of a program needs to be loaded into memory
- Eliminates the need to “fit” programs into memory holes

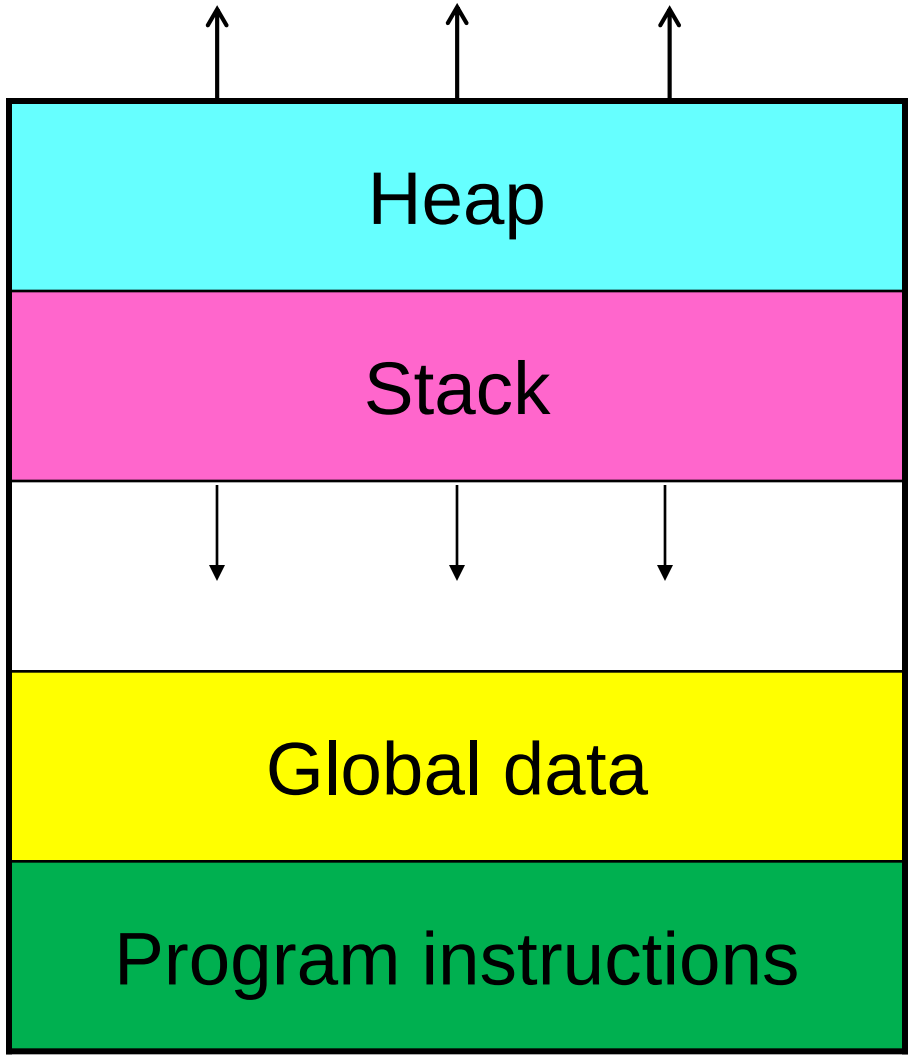
Virtual Memory Disadvantages

A. Makes address translation much more complicated
B. Can reduce performance
C. Makes program execution time less predictable
D. All of the above
E. None of the above

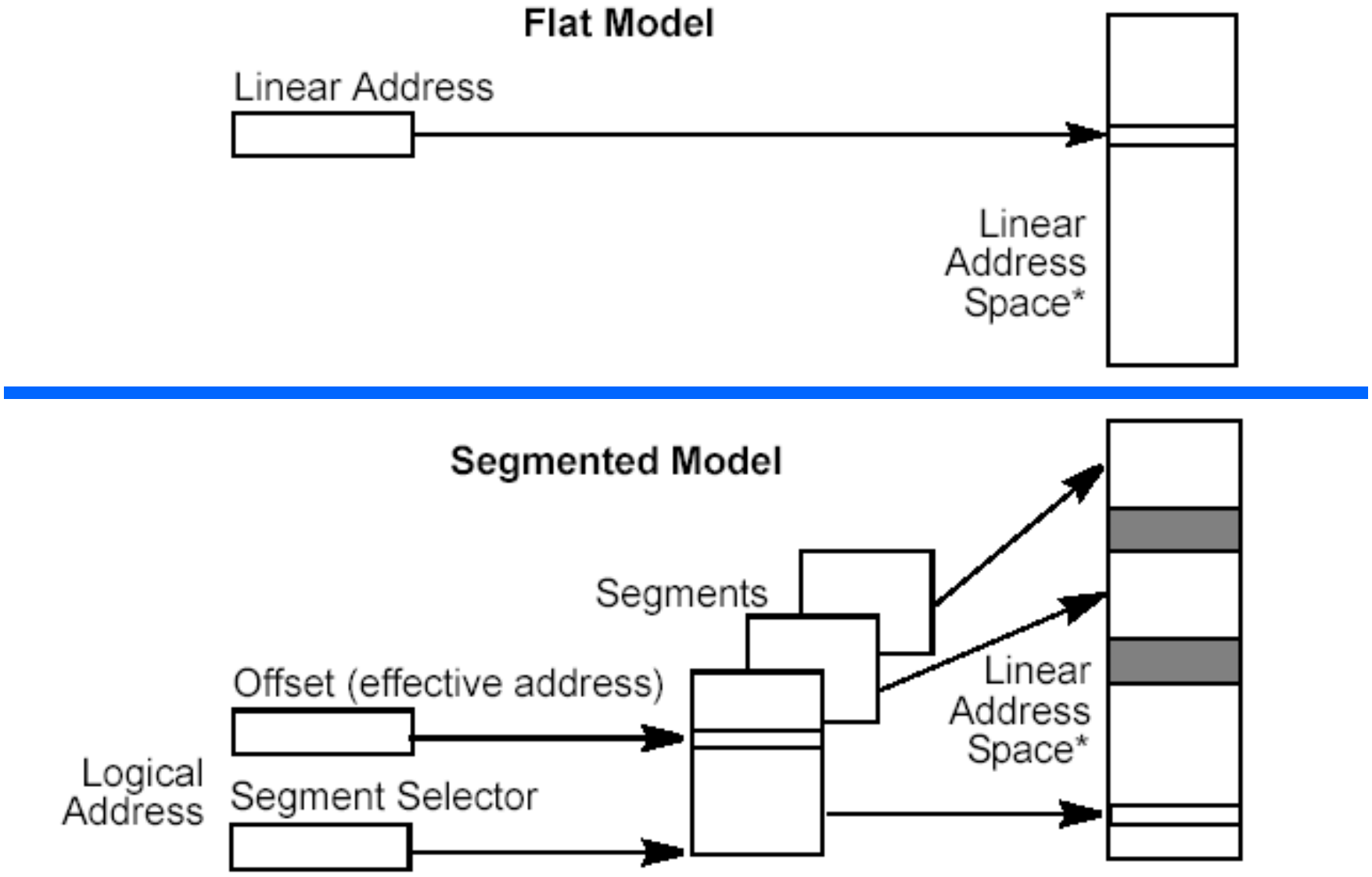
Segmented Memory

- Memory can be separated into segments based on the program
  - instruction segments for each function
  - data segments for global data and heap
  - stack segments
- Segmentation provides greater function isolation and makes linking easier
- Segments can have virtual memory pages

Program Memory Organization



Memory Models



Intel Segment Addresses