Unit 5: Memory Management

5.3. Virtual Address Translation

Roadmap for Section 5.3.

- From virtual to physical addresses
- Address space layout
- Address translation
- Page directories, page tables
- Page faults, invalid page table entries
- Page frame number database
- Recap: structuring of the memory manager

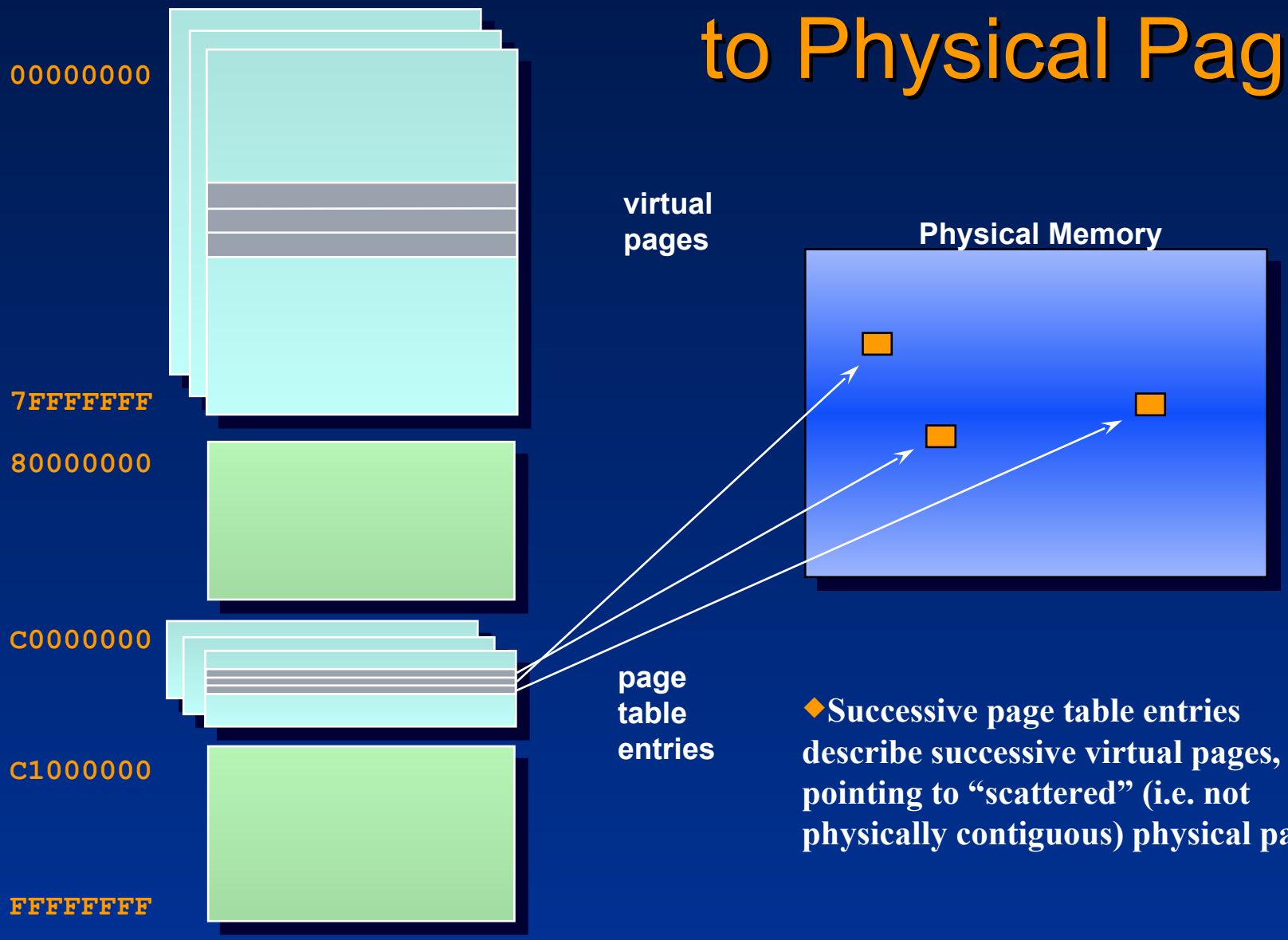
Virtual Memory - Concepts

- Application always references “virtual addresses”
- Hardware and software translates, or *maps,* virtual addresses to physical
- Not all of an application’s virtual address space is in physical memory at one time...
  - ...But hardware and software fool the application into thinking that it is
  - The rest is kept on disk, and is brought into physical memory automatically as needed

Virtual address descriptors (VADs)

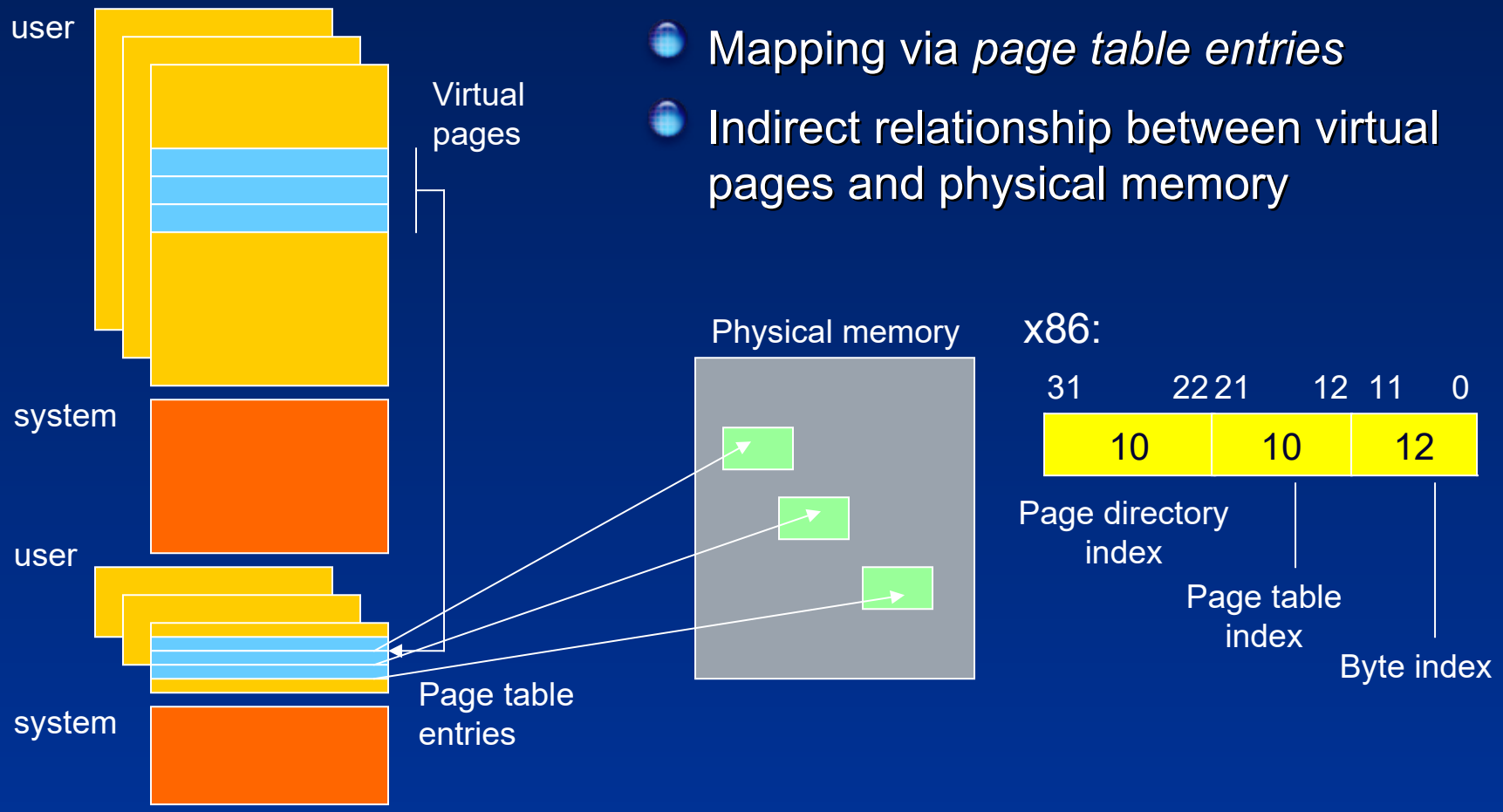
- Memory manager uses demand paging algorithm
- Lazy evaluation is also used to construct page tables
  - Reserved vs. commited memory
  - Even for commited memory, page tables are constructed on demand
- Memory manager maintains VAD structures to keep track of reserved virtual addresses
  - Self-balancing binary tree
- VAD store:
  - range of addresses being reserved;
  - whether range will be shared or private;
  - Whether child process can inherit contents of the range
  - Page protection applied to pages within the address range

Mapping Virtual to Physical Pages



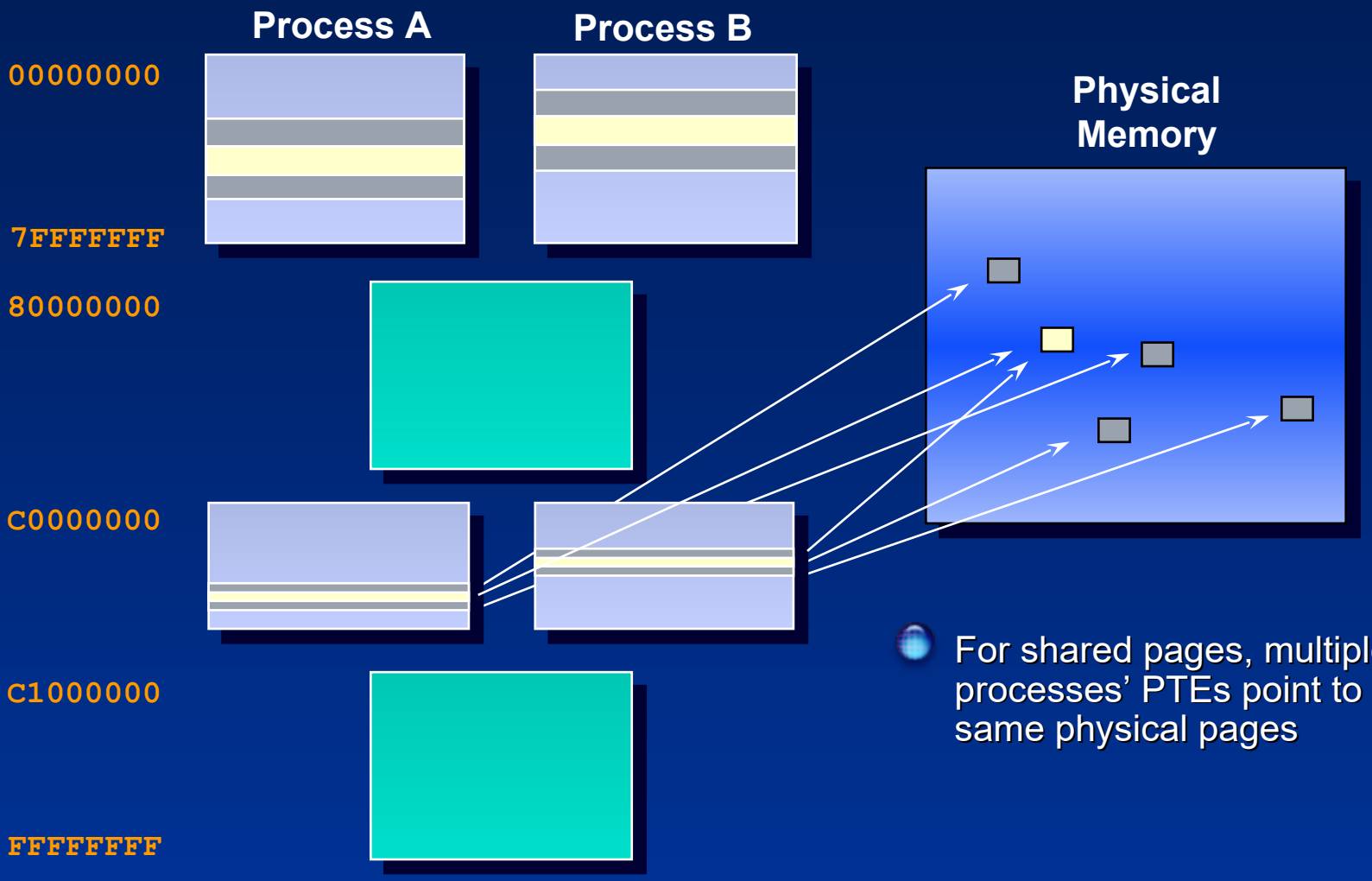
- Successive page table entries describe successive virtual pages, pointing to "scattered" (i.e. not physically contiguous) physical pages

Address Translation - Mapping virtual addresses to physical memory



- Mapping via *page table entries*
- Indirect relationship between virtual pages and physical memory

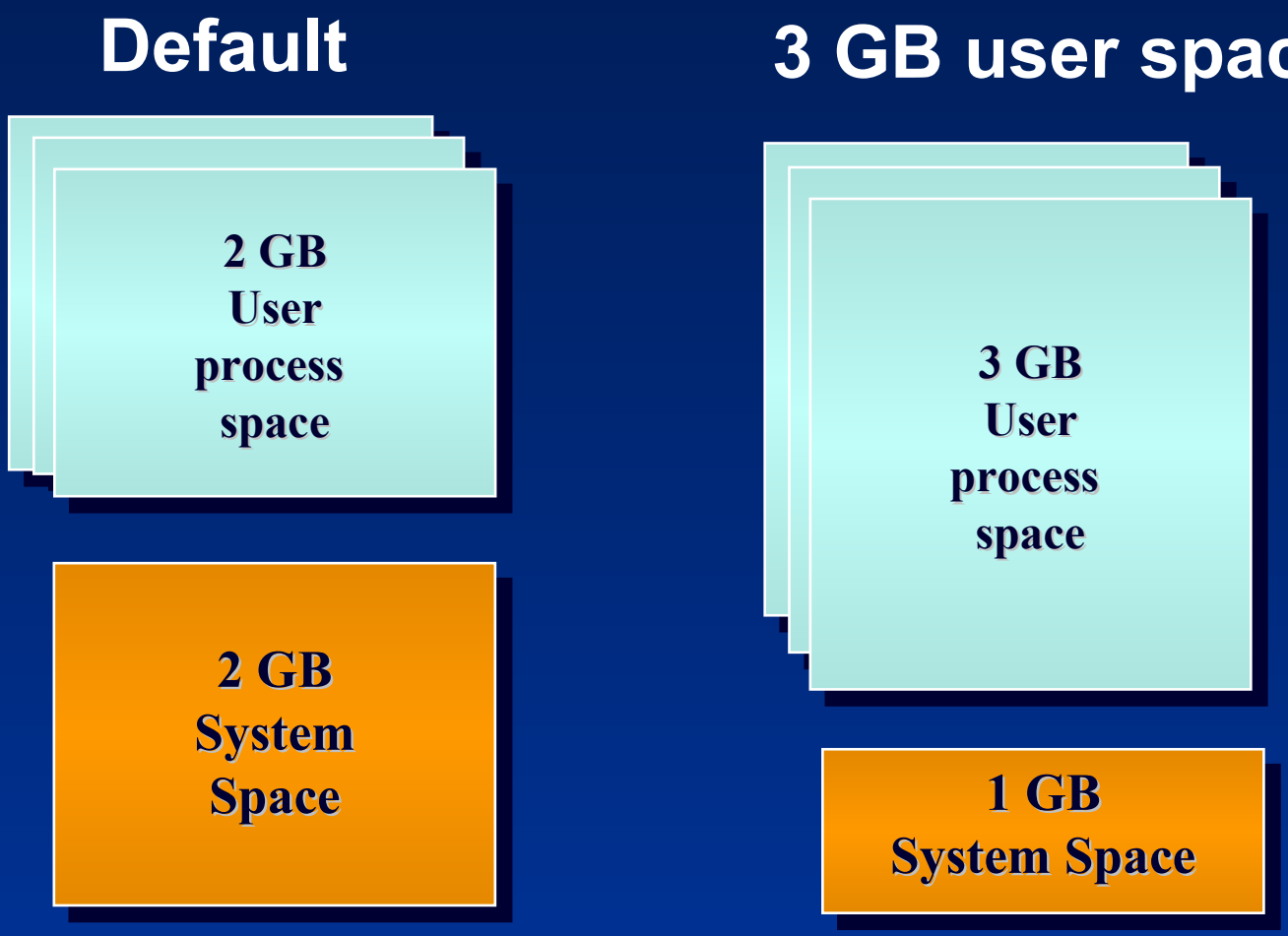
Shared and Private Pages



- For shared pages, multiple processes' PTEs point to same physical pages

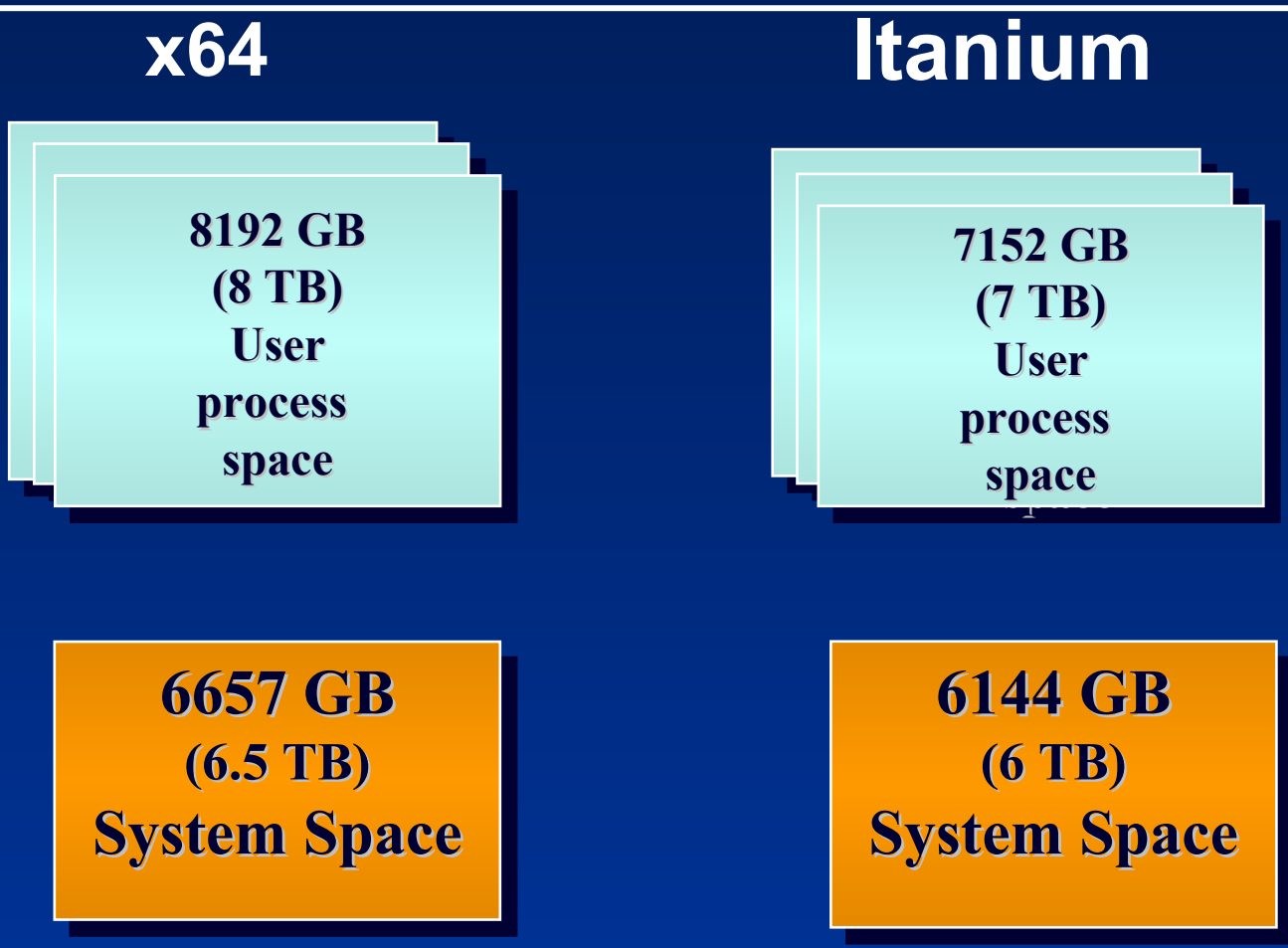
32-bit x86 Address Space

- 32-bits = 4 GB



64-bit Address Spaces

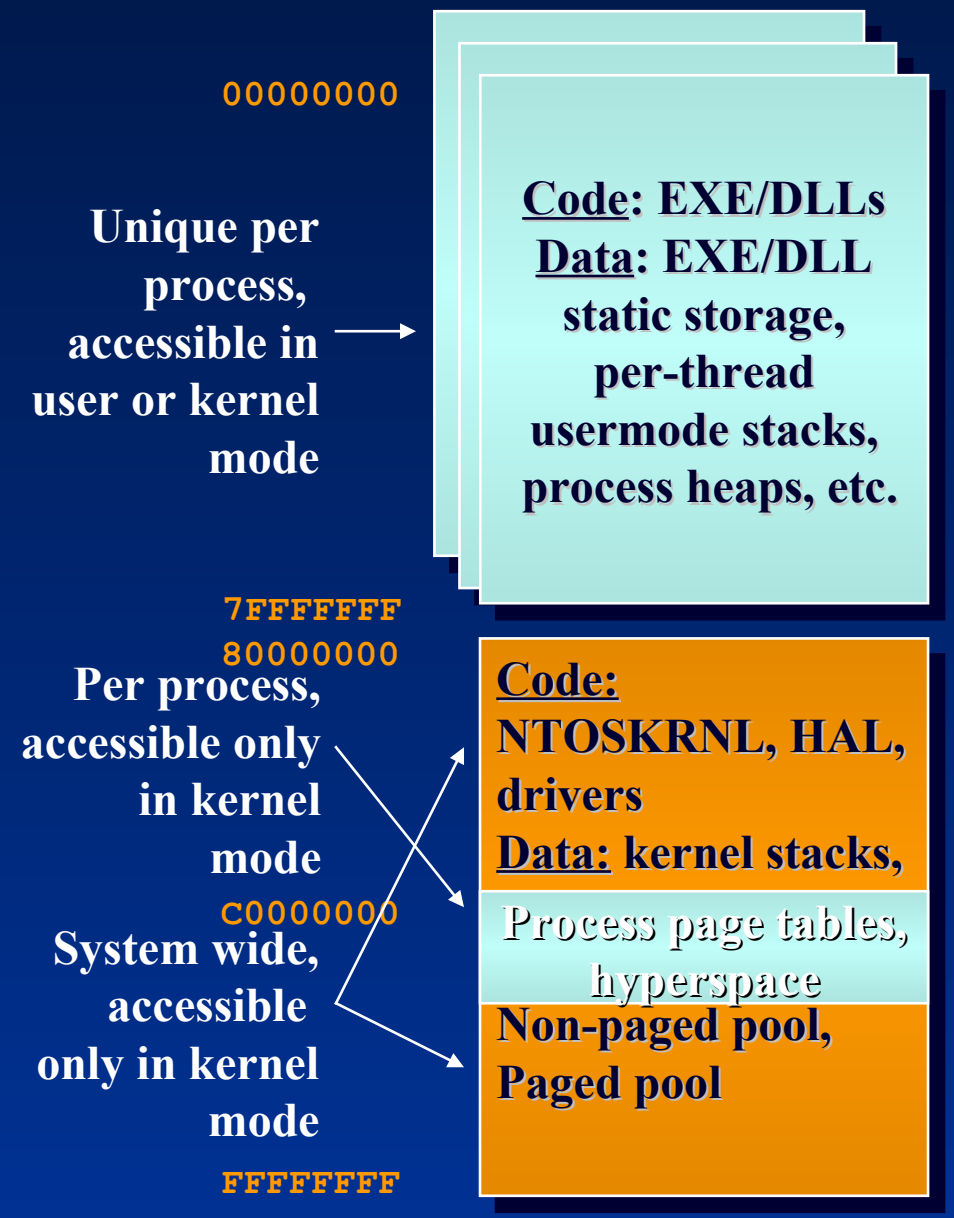
- 64-bits = 17,179,869,184 GB
  - x64 today supports 48 bits virtual = 262,144 GB
  - IA-64 today support 50 bits virtual = 1,048,576 GB



Increased Limits in 64-bit Windows

	Itanium	x64	x86
User Address Space	7152 GB	8192 GB	2-3 GB
Page file limit	16 TB	16 TB	4095 MB PAE: 16 TB
Max page file space	256 TB	256 TB	~64 GB
System PTE Space	128 GB	128 GB	1.2 GB
System Cache	1 TB	1 TB	960 MB
Paged pool	128 GB	128 GB	470-650 MB
Non-paged pool	128 GB	128 GB	256 MB

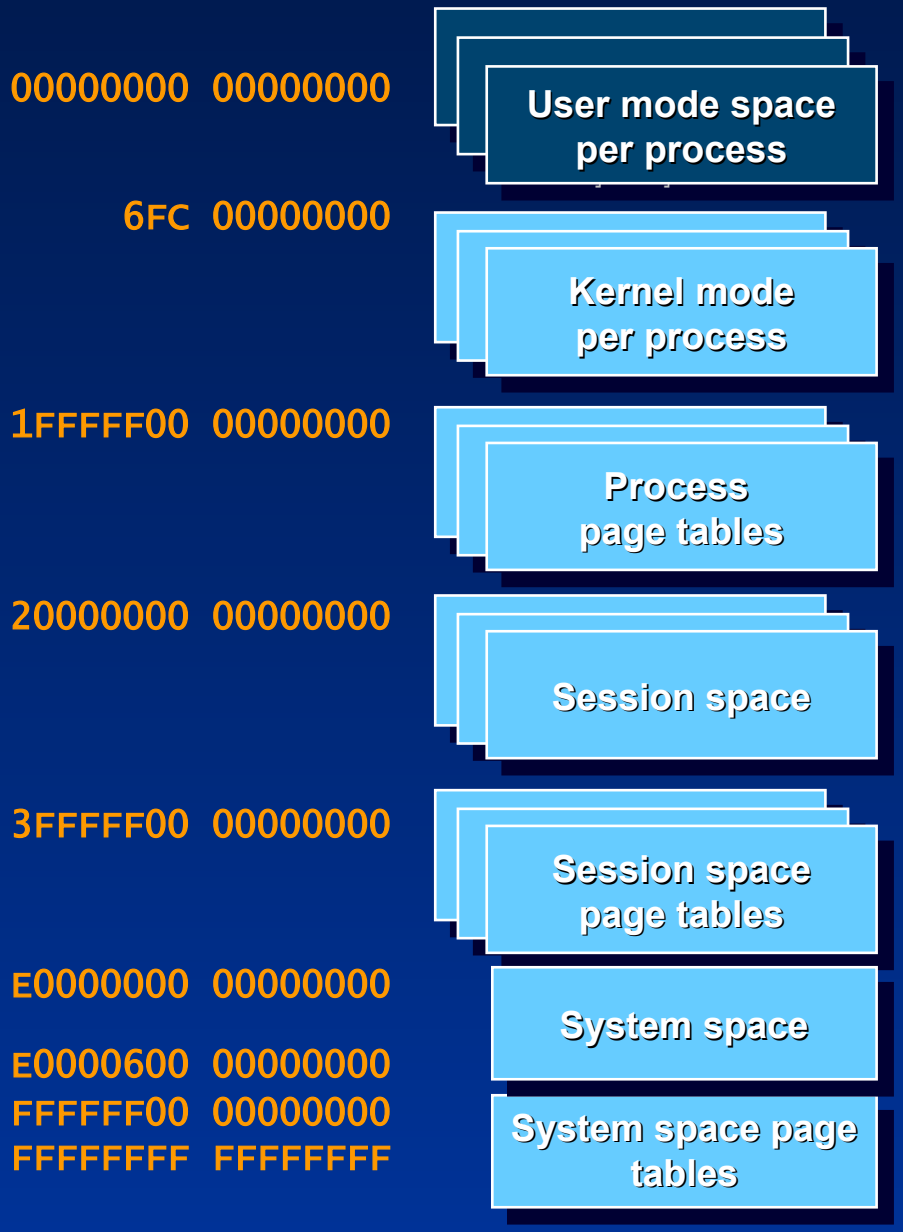
32-bit x86 Virtual Address Space



- 2 GB per-process
  - Address space of one process is not directly reachable from other processes
- 2 GB system-wide
  - The operating system is loaded here, and appears in every process's address space
  - The operating system is not a process (though there are processes that do things for the OS, more or less in "background")
- 3 GB user space option & Address Windowing Extensions (AWE) described later

64-bit ia64 (Itanium) Virtual Address Space

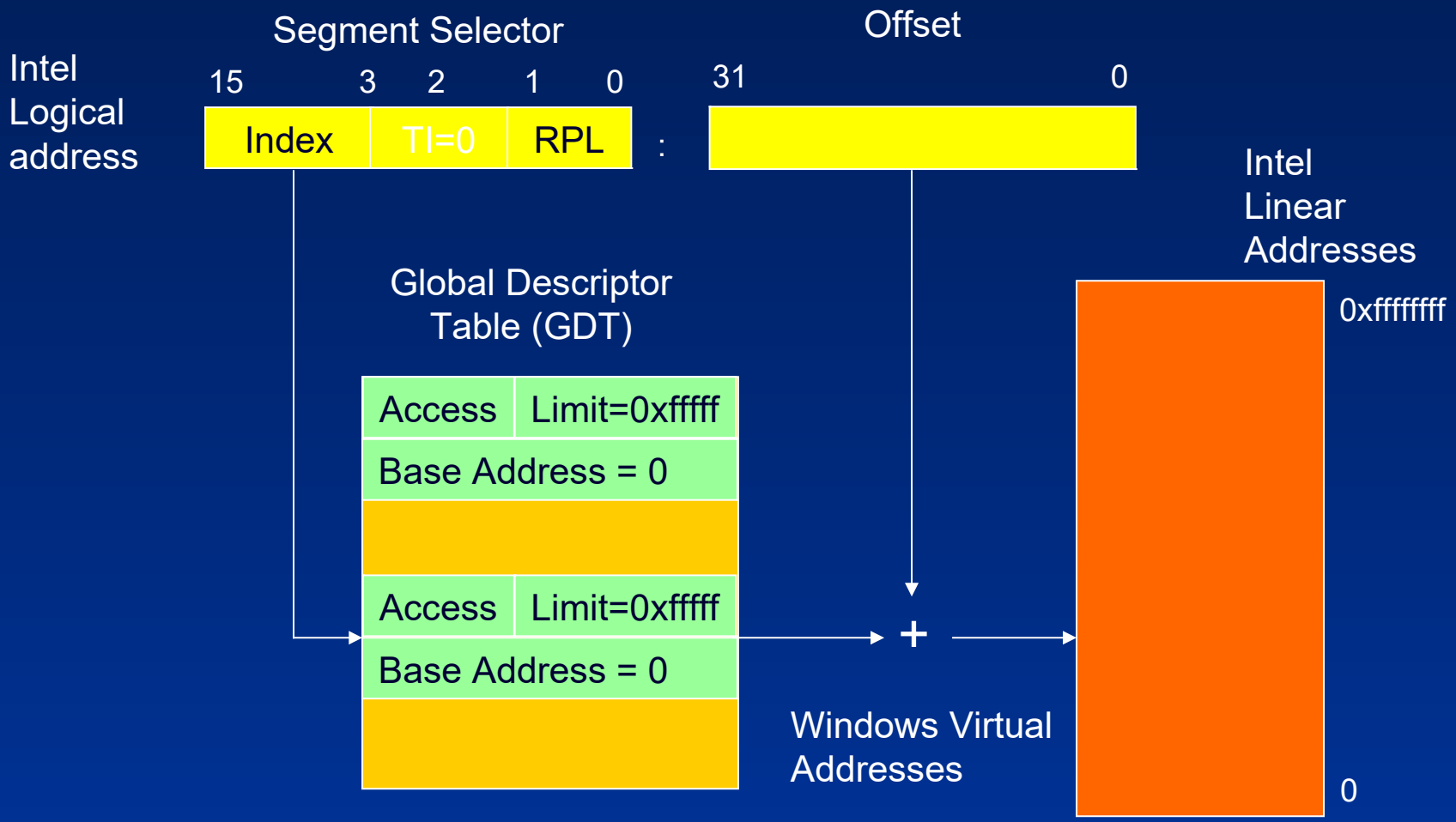
- 64 bits = 2^64 = 17 billion GB (16 exabytes) total
  - Diagram NOT to scale!
- 7152 GB default per-process
- Pages are 8 Kbytes
- All pointers are now 64 bits wide (and not the same size as a ULONG)



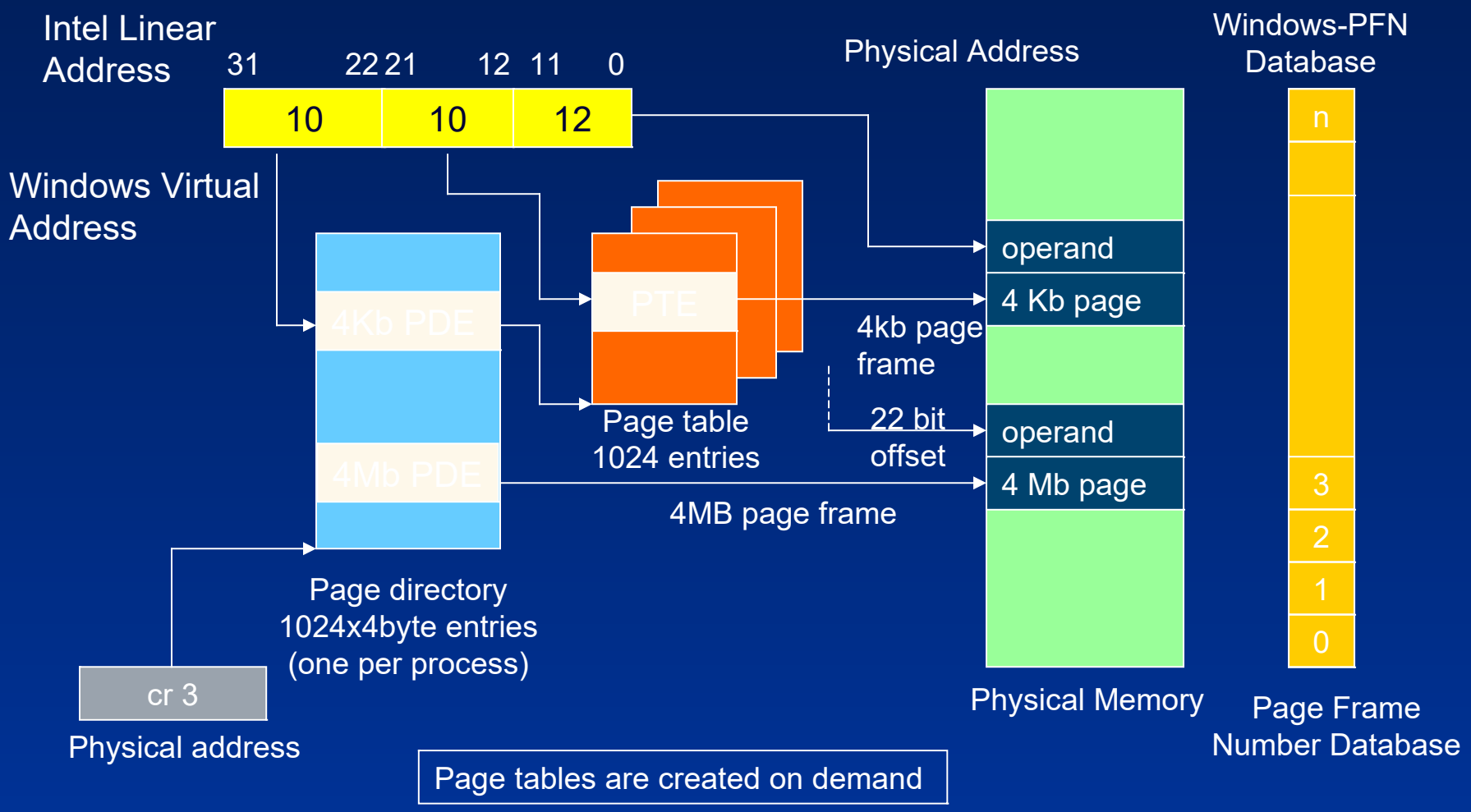
Address Translation 32-bit Windows Hardware Support Intel x86

- Intel x86 provides two levels of address translation
  - Segmentation (mandatory, since 8086)
  - Paging (optional, since 80386)
- Segmentation: first level of address translation
  - Intel: logical address (selector:offset) to linear address (32 bits)
  - Windows virtual address is Intel linear address (32 bits)
- Paging: second level of address translation
  - Intel: linear address (32 bits) to physical address
  - Windows: virtual address (32 bits) to physical address
  - Physical address: 32 bits (4 GB) all Windows versions, 36 bits (64 GB) PAE
  - Page size:
    - 4 kb since 80386 (all Windows versions)
    - 4 MB since Pentium Pro (supported in NT 4, Windows 2000/XP/2003/...)

Intel x86 Segmentation



Intel x86 Paging – Address Translation



Interpreting a Virtual Address

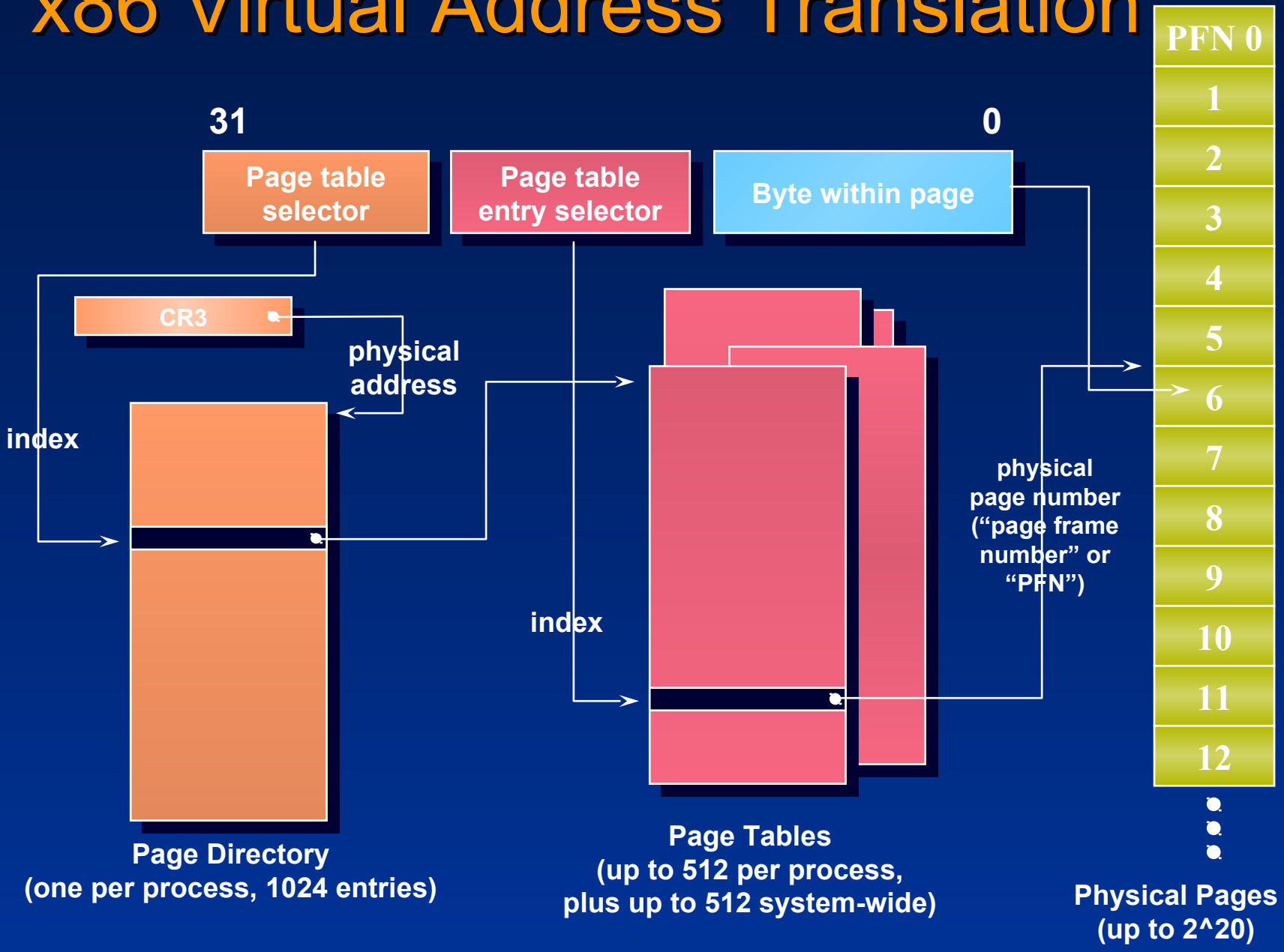
x86 32-bit

x64 64-bit (48-bit in today's processors)

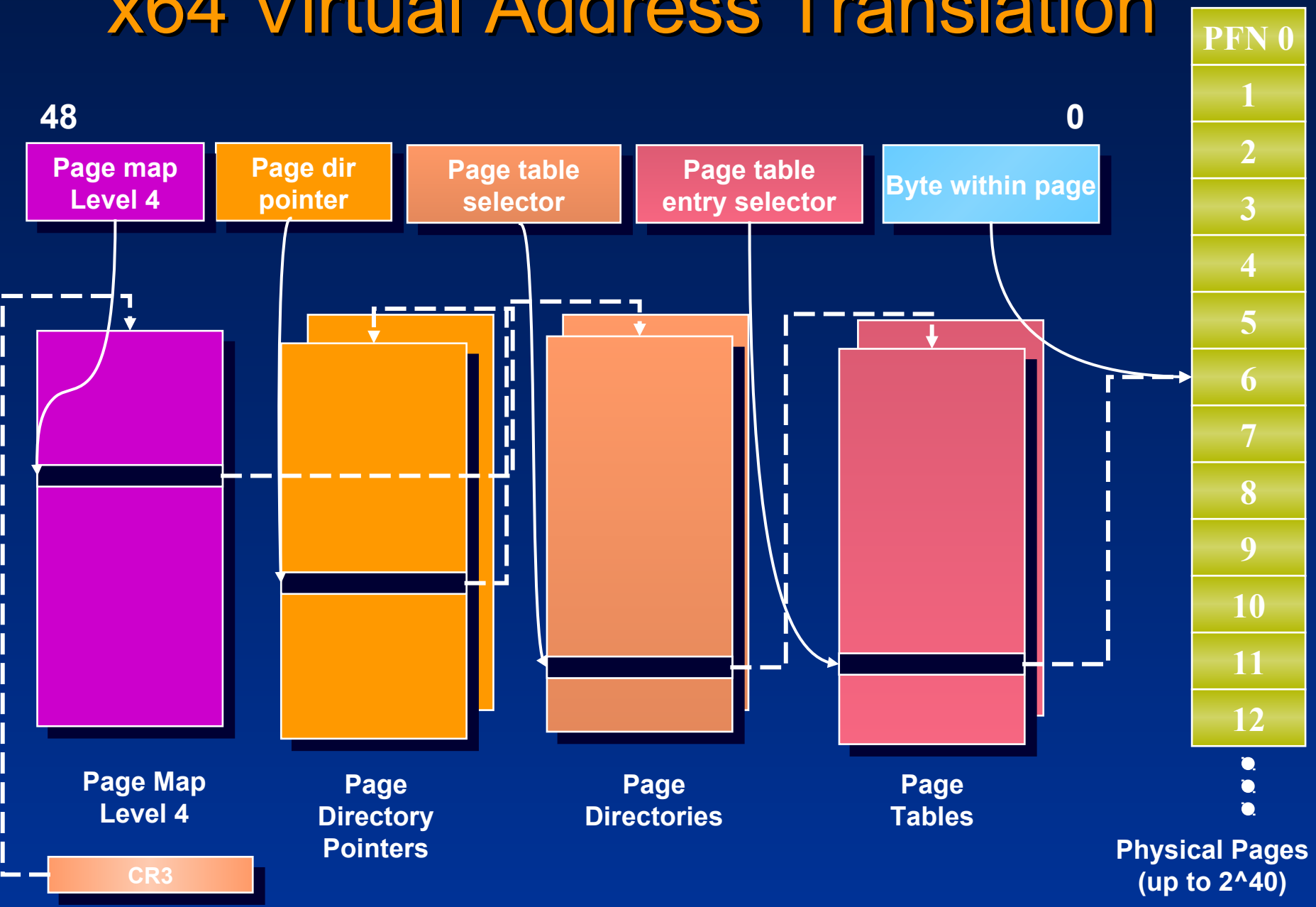
Windows Virtual Memory Use Performance Counters

Performance Counter	System Variable	Description
Memory: Committed Bytes	MmTotalCommitedPages	Amount of committed private address space that has a backing store
Memory: Commit Limit	MmTotalCommit-Limit	Amount of memory (in bytes) that can be committed without increasing size of paging file
Memory: %Commited Bytes in Use	MmTotalCommittedPages / MmTotalCommitLimit	Ratio of committed bytes to commit limit

x86 Virtual Address Translation

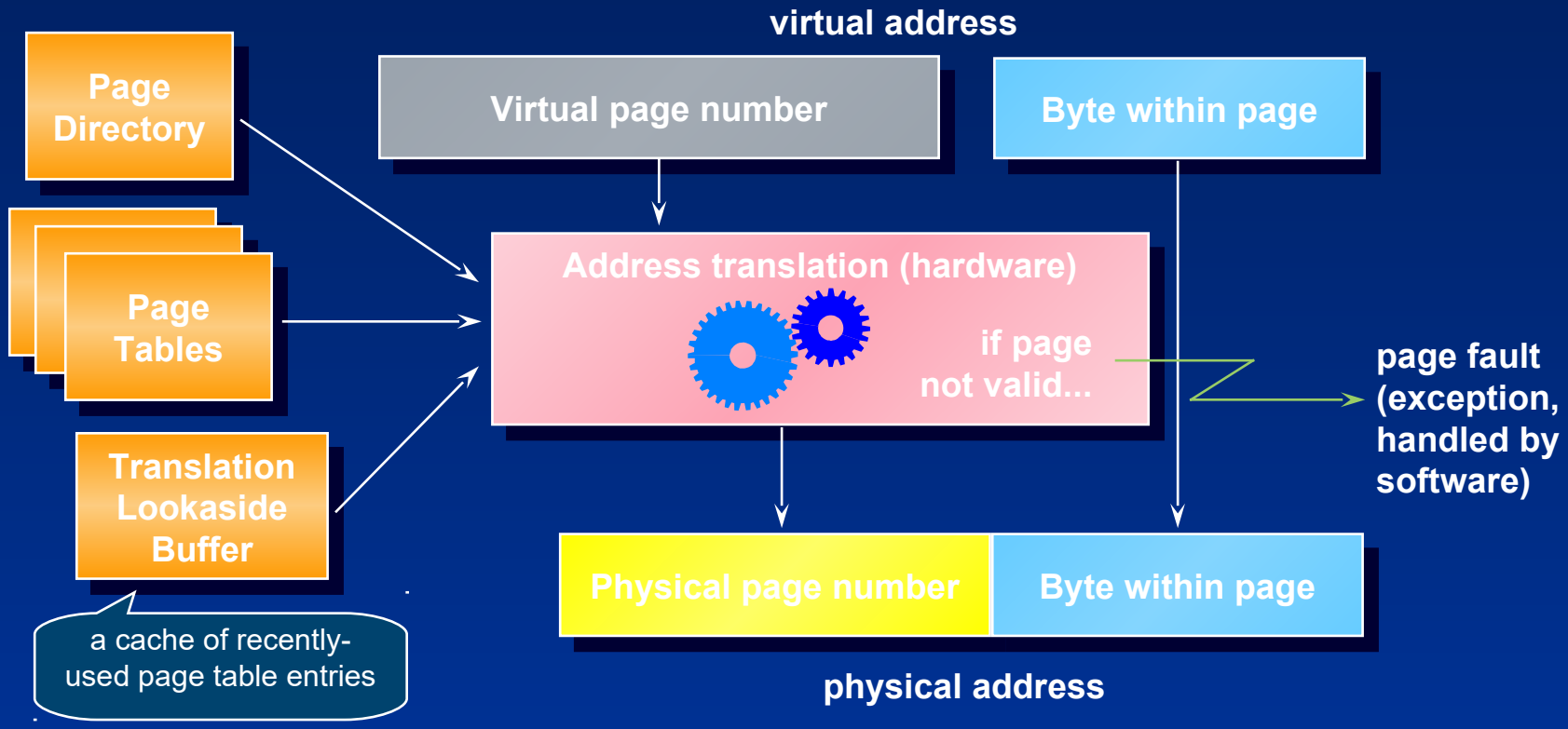


x64 Virtual Address Translation



Virtual Address Translation

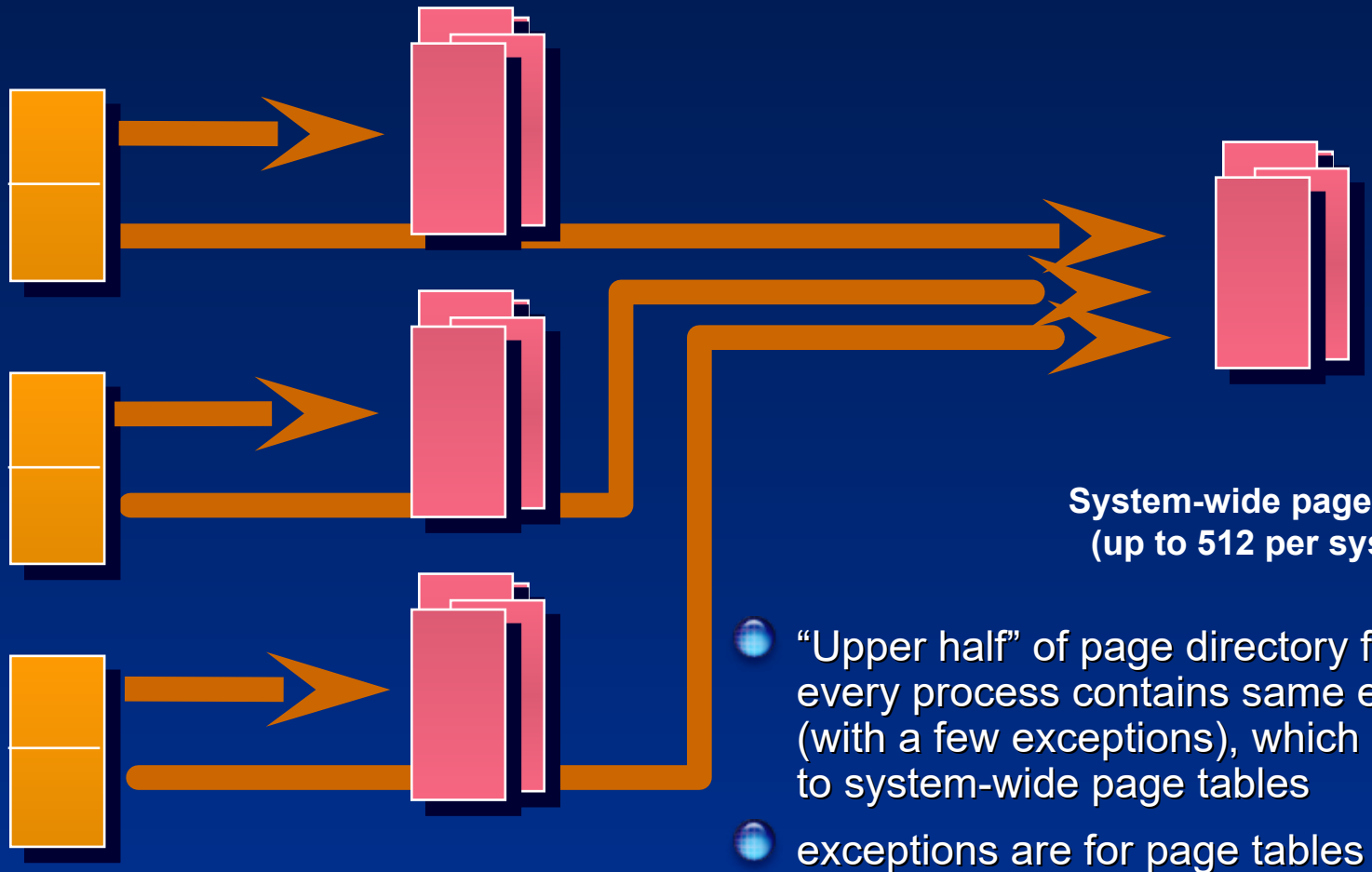
- The hardware converts each valid virtual address to a physical address



Itanium Address Translation

- 3 level page table (vs 2 on x86)
  - 43 bit virtual addressing
  - 44 bit physical addressing
- Two TLBs
  - Instruction TLB – translates instruction addresses
  - Data TLB – translates data addresses
- Each have OS-managed *translation registers* and hardware managed *translation cache*
  - OS can insert TLB entries
    - OS decides which slots when inserting into translation registers
    - Hardware decides when inserting into translation cache
  - Itanium: 96 instruction translation cache entries; 128 data translation cache entries

Mapping Process vs. System-Space Addresses



System-wide page tables (up to 512 per system)

Page Directories (one per process)

Sets of per-process page tables (up to 512 per process)

- "Upper half" of page directory for every process contains same entries (with a few exceptions), which point to system-wide page tables
- exceptions are for page tables that map the process page tables (not shown)

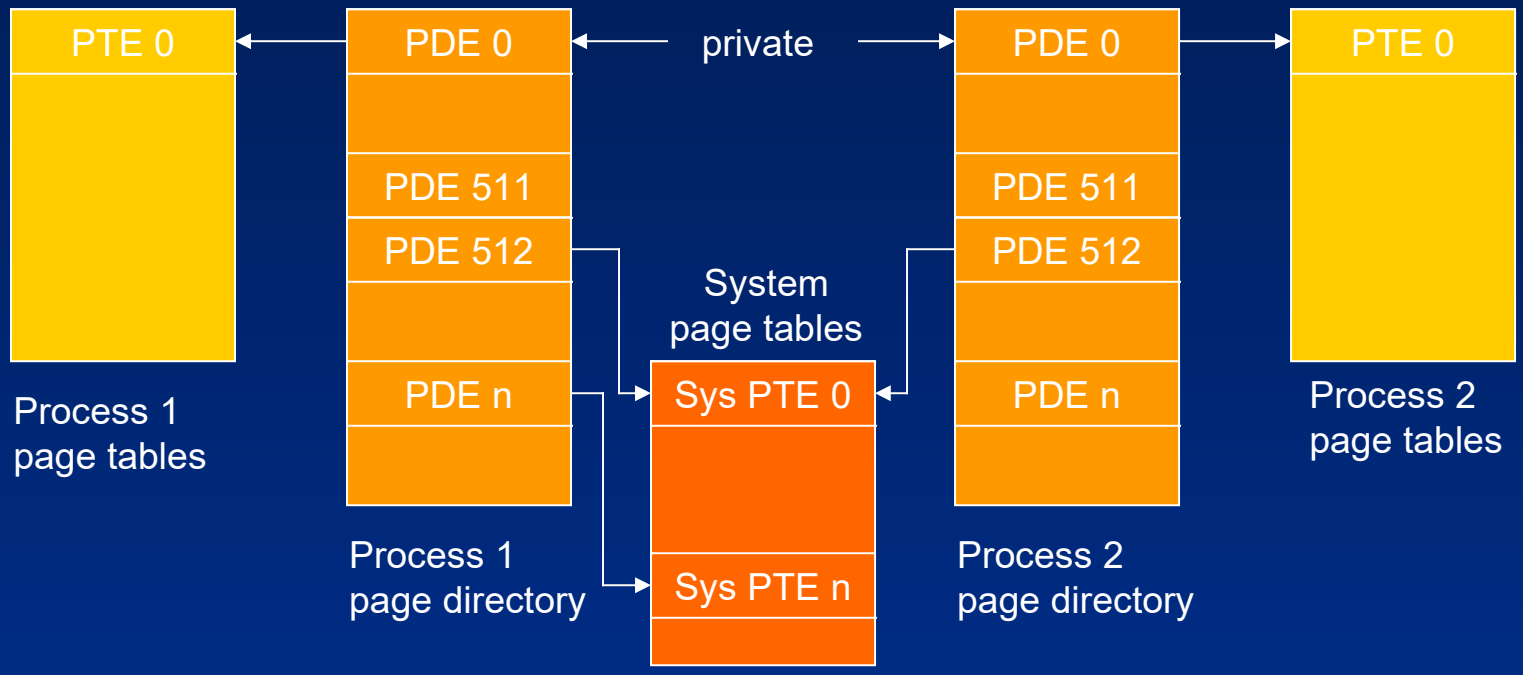
Translating a virtual address:

1. Memory management HW locates page directory for current process (cr3 register on Intel)
2. Page directory index directs to requested page table
3. Page table index directs to requested virtual page
4. If page is valid, PTE contains physical page number (PFN – page frame number) of the virtual page
   - Memory manager fault handler locates invalid pages and tries to make them valid
   - Access violation/bug check if page cannot be brought in (protection fault)
5. When PTE points to valid page, byte index is used to locate address of desired data

Page directories & Page tables

- Each process has a single page directory (physical address in KPROCESS block, at 0xC0300000, in cr3 (x86))
  - cr3 is re-loaded on inter-process context switches
  - Page directory is composed of page directory entries (PDEs) which describe state/location of page tables for this process
    - Page tables are created on demand
  - x86: 1024 page tables describe 4GB
- Each process has a private set of page tables
- System has one set of page tables
  - System PTEs are a finite resource: computed at boot time
  - HKLM\System...\Control\SessionManager\SystemPages

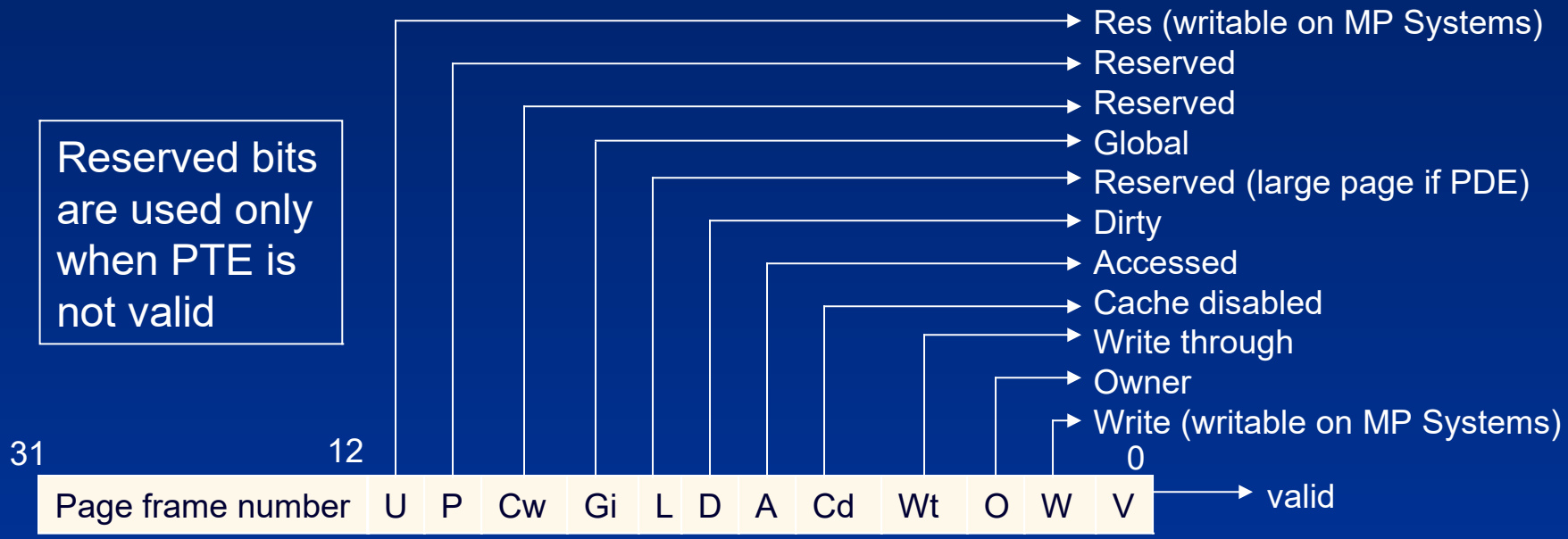
System and process-private page tables



- On process creation, system space page directory entries point to existing system page tables
- Not all processes have same view of system space (after allocation of new page tables)

Page Table Entries

- Page tables are array of Page Table Entries (PTEs)
- Valid PTEs have two fields:
  - Page Frame Number (PFN)
  - Flags describing state and protection of the page

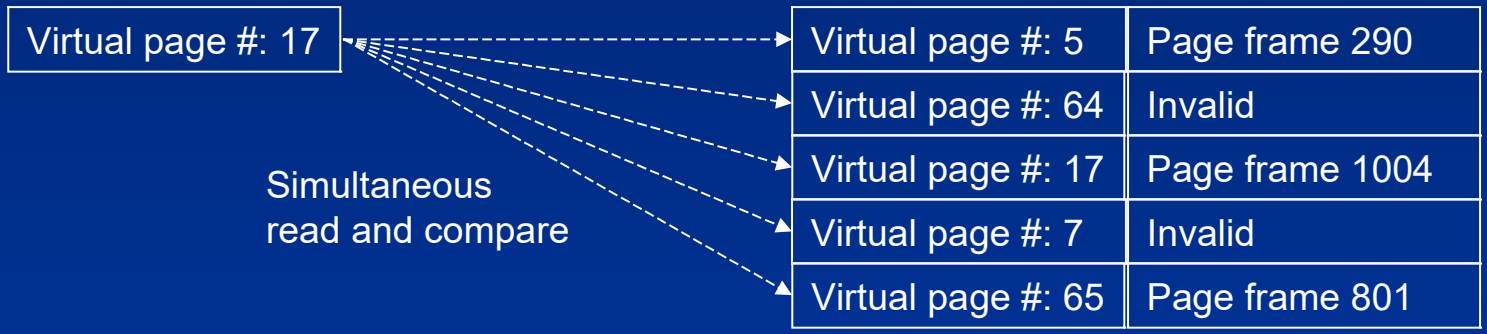


PTE Status and Protection Bits (Intel x86 only)

Name of Bit	Meaning on x86
Accessed	Page has been read
Cache disabled	Disables caching for that page
Dirty	Page has been written to
Global	Translation applies to all processes (a translation look-aside buffer flush won‘t affect this PTE)
Large page	Indicates that PDE maps a 4MB page (used to map kernel)
Owner	Indicates whether user-mode code can access the page or whether the page is limited to kernel mode access
Valid	Indicates whether translation maps to page in physical memory
Write through	Disables caching of writes; immediate flush to disk
Write	Uniprocessor: indicates whether page is read/write or read-only; Multiprocessor: indicates whether page is writeable/write bit in reserved bit

Translation Look-Aside Buffer (TLB)

- Address translation requires two lookups:
  - Find right table in page directory
  - Find right entry in page table
- Most CPU cache address translations
  - Array of associative memory: translation look-aside buffer (TLB)
  - TLB: virtual-to-physical page mappings of most recently used pages



Page Fault Handling

- Reference to invalid page is called a page fault
- Kernel trap handler dispatches:
  - Memory manager fault handler (MmAccessFault) called
  - Runs in context of thread that incurred the fault
  - Attempts to resolve the fault or raises exception
- Page faults can be caused by a variety of conditions
- Four basic kinds of invalid Page Table Entries (PTEs)

(*see next slides*)

In-Paging I/O due to Access Faults

- Accessing a page that is not resident in memory but on disk in page file/mapped file
  - Allocate memory and read page from disk into working set
- Occurs when read operation must be issued to a file to satisfy page fault
  - Page tables are pageable → additional page faults possible
- In-page I/O is synchronous
  - Thread waits until I/O completes
  - Not interruptible by asynchronous procedure calls
- During in-page I/O: faulting thread does not own critical memory management synchronization objects

  Other threads in process may issue VM functions, but:
  - Another thread could have faulted same page: collided page fault
  - Page could have been deleted (remapped) from virtual address space
  - Protection on page may have changed
  - Fault could have been for prototype PTE and page that maps prototype PTE could have been out of working set

Other reasons for access faults

- Accessing page that is on standby or modified list
  - Transition the page to process or system working set
- Accessing page that has no committed storage
  - Access violation
- Accessing kernel page from user-mode
  - Access violation
- Writing to a read-only page
  - Access violation
- Executing from a no-execute page
  - Access violation

Reasons for access faults (contd.)

- Writing to a guard page
  - Guard page violation (if a reference to a user-mode stack, perform automatic stack expansion)
- Writing to a copy-on-write page
  - Make process-private copy of page and replace original in process or system working set
- Referencing a page in system space that is valid but not in the process page directory
  (if paged pool expanded after process directory was created)
  - Copy page directory entry from master system page directory structure and dismiss exception
- On a multiprocessor system: writing to valid page that has not yet been written to
  - Set dirty bit in PTE

Invalid PTEs and their structure

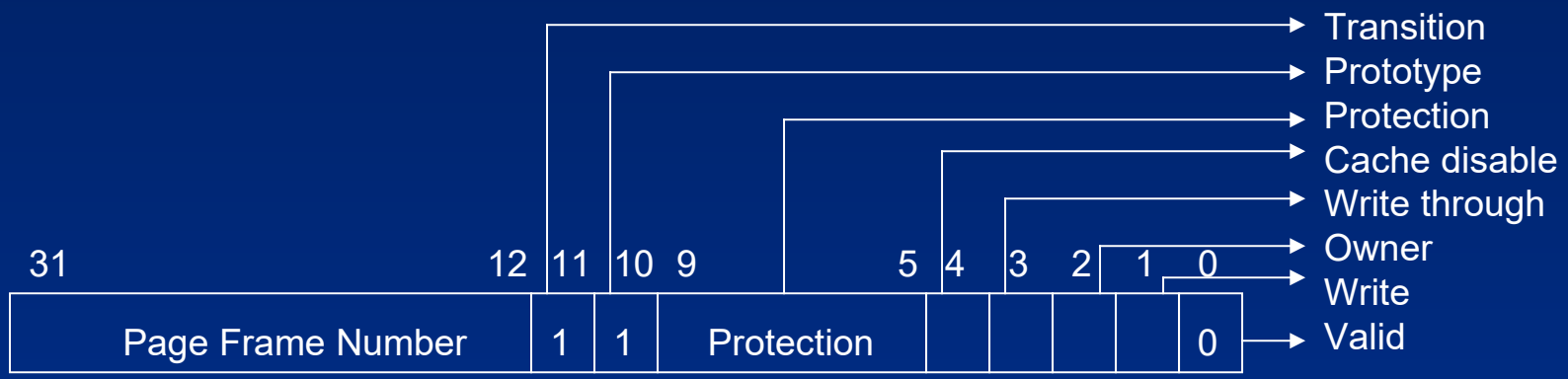
- **Page file**: desired page resides in paging file
  in-page operation is initiated

- **Demand Zero**: pager looks at zero page list;
  if list is empty, pager takes list from standby list and zeros it;

  PTE format as shown above, but page file number and offset are zeros

Invalid PTEs and their structure (contd.)

- **Transition**: the desired page is in memory on either the standby, modified, or modified-no-write list
  - Page is removed from the list and added to working set



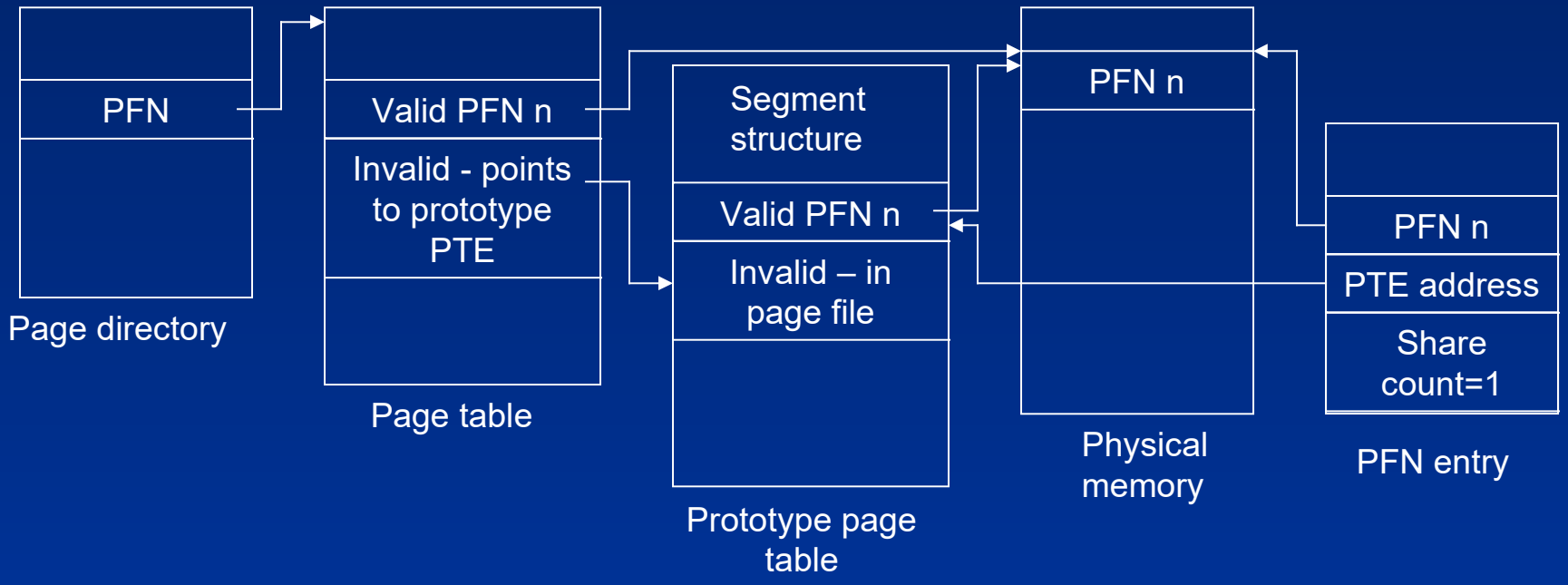
- **Unknown**: the PTE is zero, or the page table does not yet exist
  - examine virtual address space descriptors (VADs) to see whether this virtual address has been reserved
  - Build page tables to represent newly committed space

Prototype PTEs

- Software structure to manage potentially shared pages
  - Array of prototype PTEs is created as part of section object (part of segment structure)
  - First access of a page mapped to a view of a section object: memory manager uses prototype PTE to fill in real PTE used for address translation;
  - Reference count for shared pages in PFN database
- Shared page valid:
  - process & prototype PTE point to physical page
- Page invalidated:
  - process PTE points to prototype PTE
- Prototype PTE describes 5 states for shared page:
  - Active/valid, Transition, Demand zero, Page file, Mapped file
- Layer between page table and page frame database

Prototype PTEs for shared pages – the bigger picture

- Two virtual pages in a mapped view
- First page is valid; 2nd page is invalid and in page file
  - Prototype PTE contains exact location
  - Process PTE points to prototype PTE



Further Reading

- Pavel Yosifovich, Alex Ionescu, et al., “Windows Internals”, 7th Edition, Microsoft Press, 2017.
  - Chapter 5 – Memory management (from pp. 421)
    - Address Translation (from pp. 516)
    - Shared Memory and Mapped Files (from pp. 440)