Operating Systems (A)
(Honor Track)

Lecture 11: Windows Virtual Memory

Yao Guo (郭耀)

Peking University

Fall 2021

Acknowledgements: Prof. Xiangqun Chen & Tao Wang at PKU and Prof. Yuanyuan Zhou at UCSD

Buzz Words

Protected Mode

Virtual Address Descriptor (VAD)

Working set

Self-Map

This Lecture

Windows MM

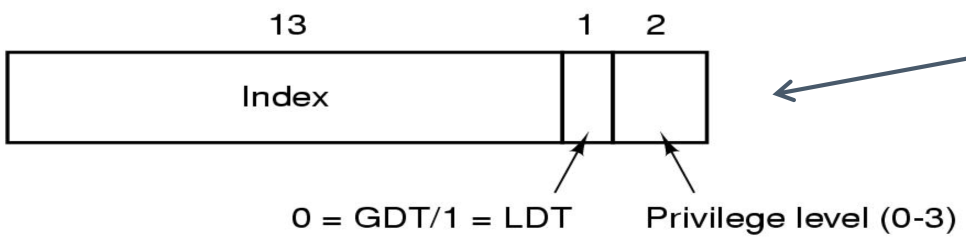
Intel x86 Virtual Memory

Windows MM

Address Mode for x86

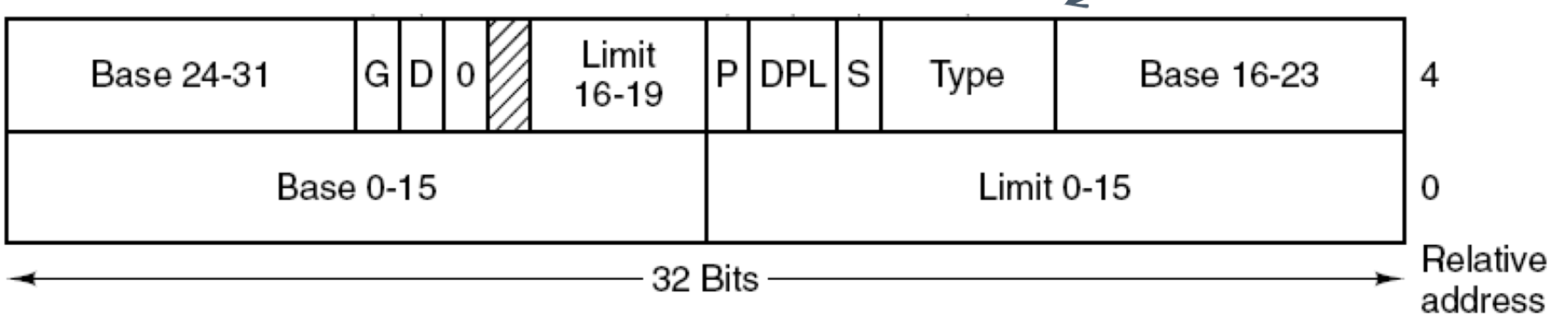
- Real mode (8086)
  - Using physical address
  - For DOS
- Protected mode (80286)
  - Protected virtual address mode
  - Supports virtual memory, paging and safe multi-tasking
  - Supports both 16 and 32 bits
- Intel 8086 virtual mode (80386)
  - Hardware virtualization
  - Run real mode in the protected mode

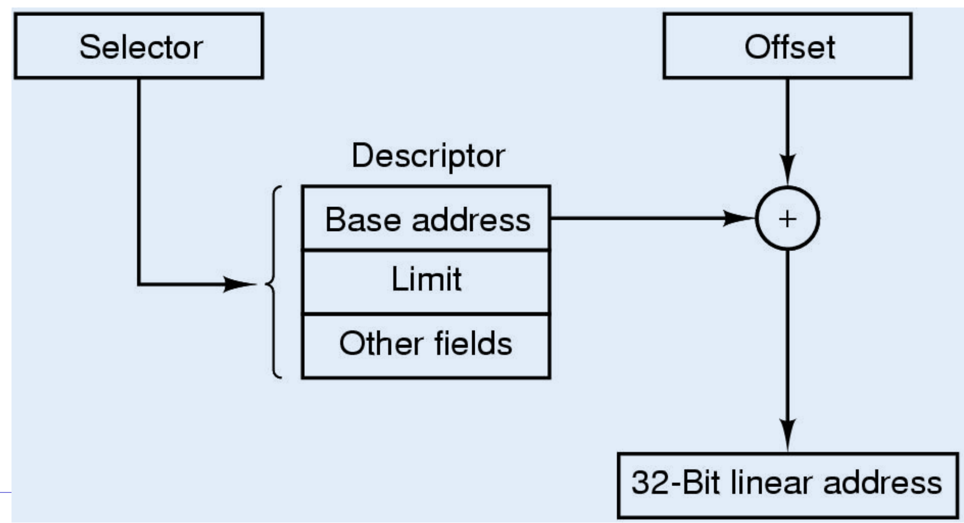
Protected Mode



Segment selector

Segment descriptor





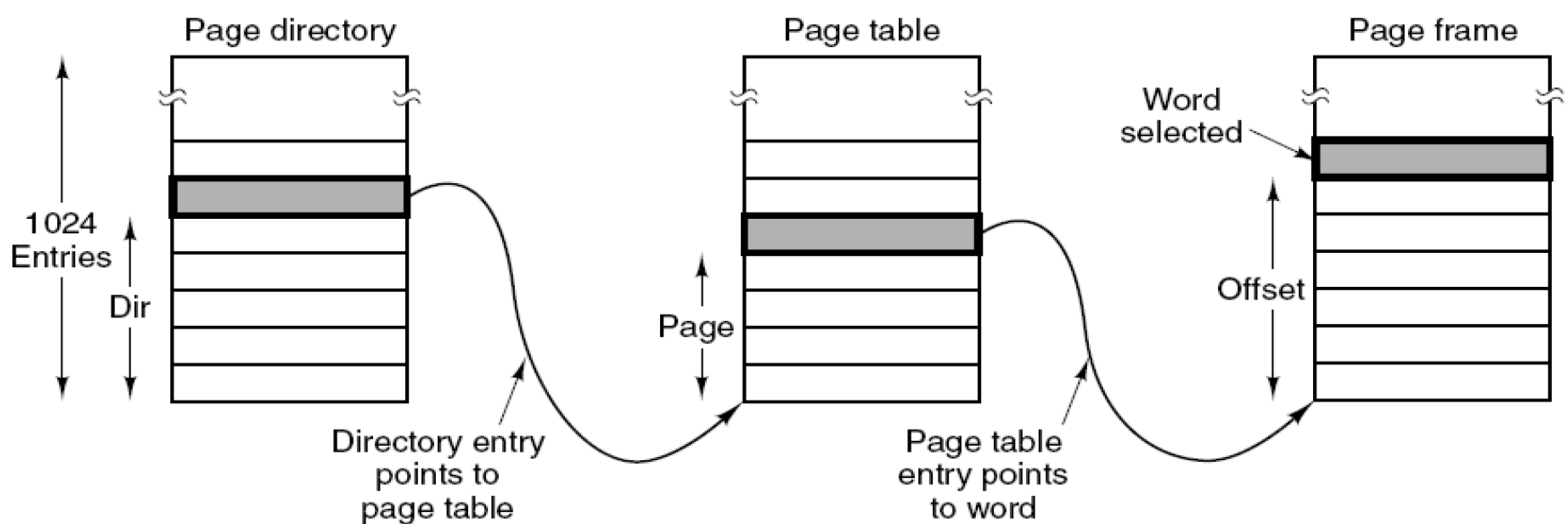
Logical to Physical Address

Logical address → Segmentation → Linear address → Paging → Physical address

Linear address

Bits	10	10	12
	Dir	Page	Offset

(a)



i386 Page Table Entries

PDE (Page Directory Entry)

PFN	Avail	G	PS	0	A	PCD	PWT	U/S	R/W	P

PTE (Page Table Entry)

PFN	Avail	G	0	D	A	PCD	PWT	U/S	R/W	P

63	62 52	51 12	11 9	8	7	6	5	4	3	2	1	0
NX	AVL	Physical page number	AVL	G	PAT	D	A	PCD	PWT	U/S	R/W	P

NX No eXecute
AVL AVaiLable to the OS
G Global page
PAT Page Attribute Table
D Dirty (modified)
A Accessed (referenced)

PCD Page Cache Disable
PWT Page Write-Through
U/S User/Supervisor
R/W Read/Write access
P Present (valid)

Intel x86 or AMD x64 PTEs

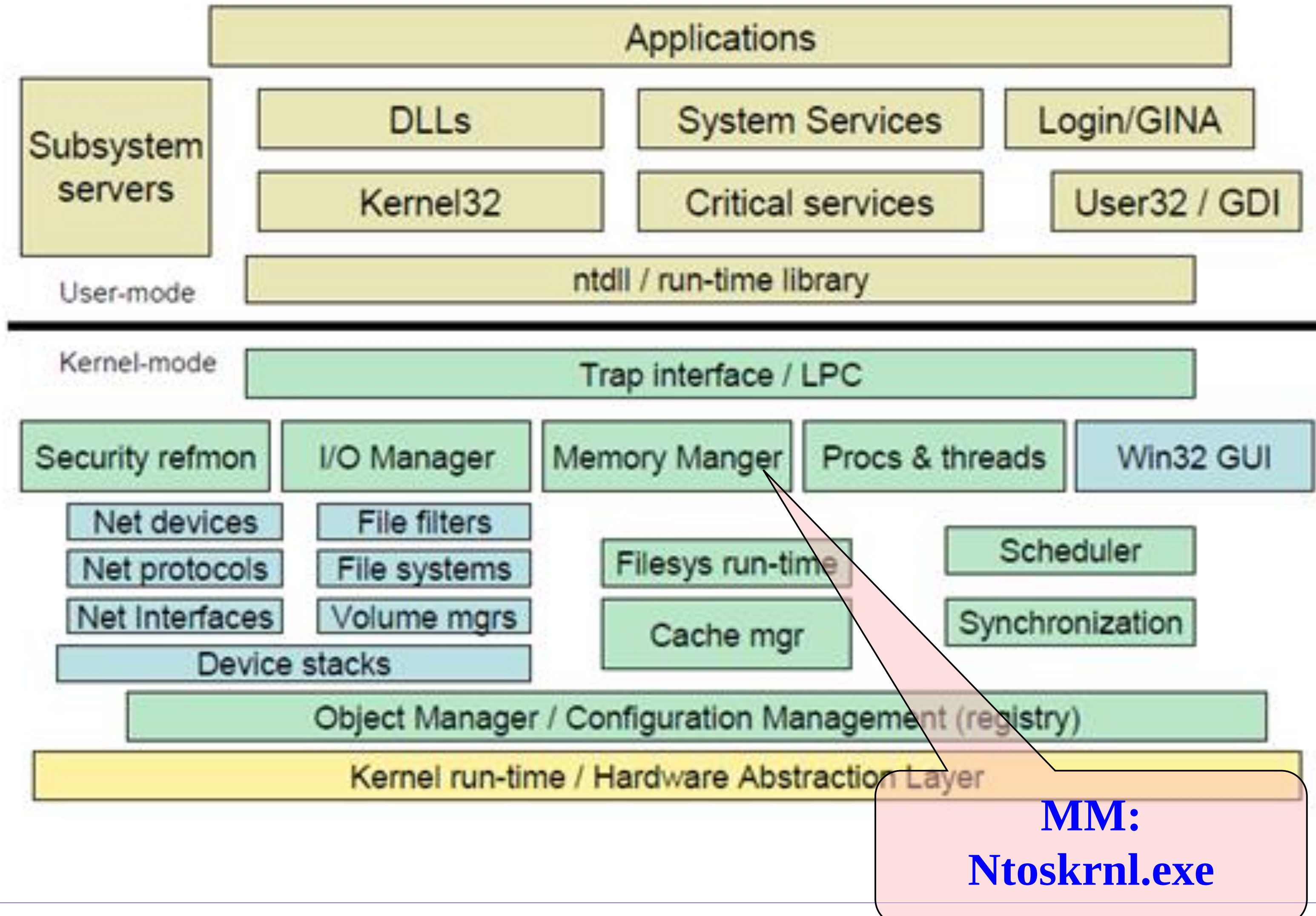
This Lecture

Windows MM

Intel x86 Virtual Memory

Windows MM

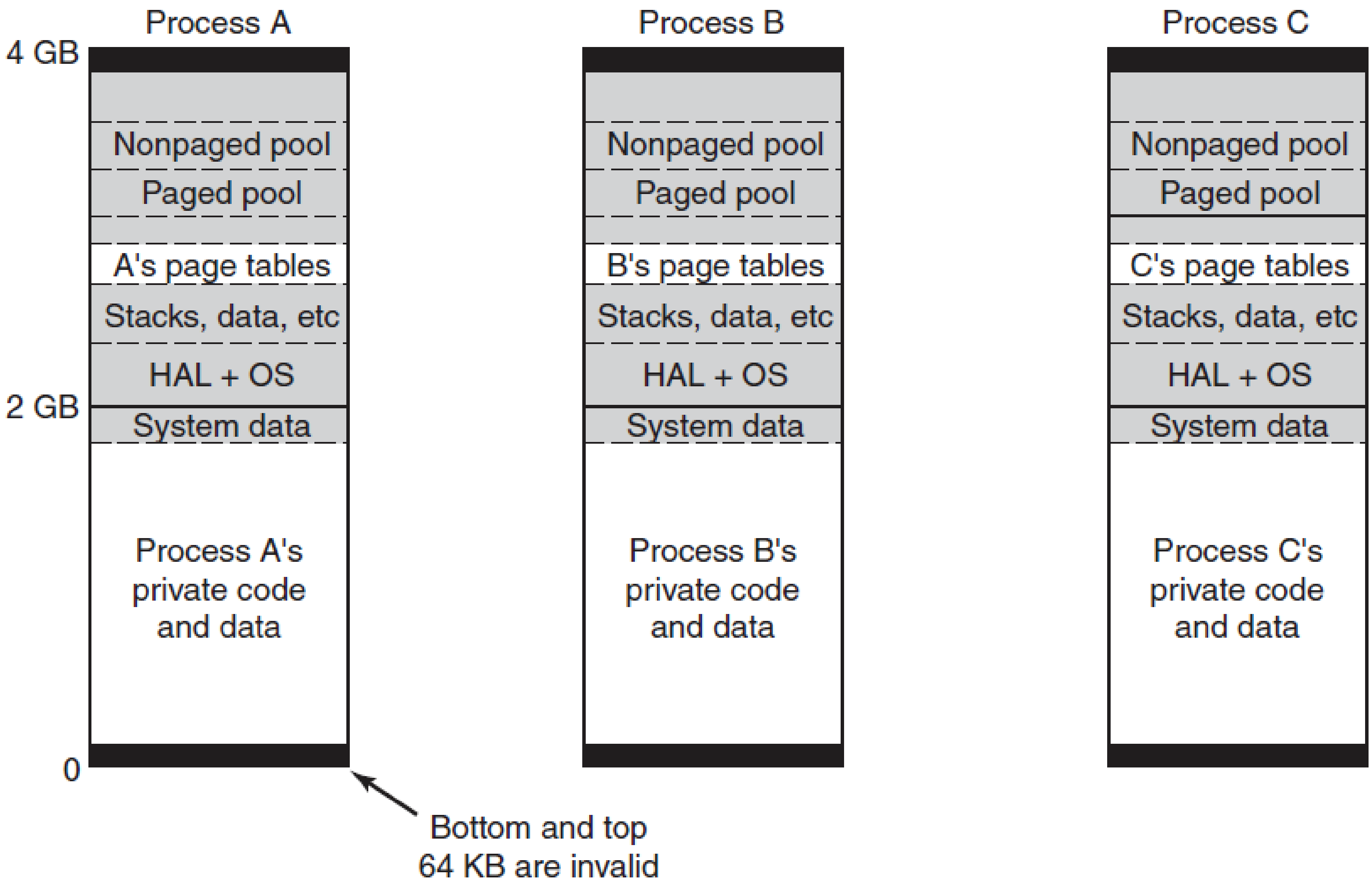
The Windows Architecture



Windows Virtual Memory

- x86
  - Virtual addresses: 32 bits
  - Address space for each process: 4GB
    - Kernel: 2GB
    - User : 2GB
- x86-64
  - Large enough
- Demand paging
  - Page size: 4KB
    - (2MB pages are also used)

Virtual Address Space Layout



Virtual Address Allocation

- Each page of virtual addresses can be in one of three states:
  - Invalid: not currently mapped to a memory section object and a reference to it causes a page fault
  - Committed: code or data is mapped onto a virtual page
  - Reserved: invalid but has the property that those virtual addresses will never be allocated by the memory manager for another purpose
    - function as guard pages to keep the stack from growing too far and overwriting other process data.

Windows MM System Calls

Win32 API function	Description
VirtualAlloc	Reserve or commit a region
VirtualFree	Release or decommit a region
VirtualProtect	Change the read/write/execute protection on a region
VirtualQuery	Inquire about the status of a region
VirtualLock	Make a region memory resident (i.e., disable paging for it)
VirtualUnlock	Make a region pageable in the usual way
CreateFileMapping	Create a file-mapping object and (optionally) assign it a name
MapViewOfFile	Map (part of) a file into the address space
UnmapViewOfFile	Remove a mapped file from the address space
OpenFileMapping	Open a previously created file-mapping object

Windows MM Implementation

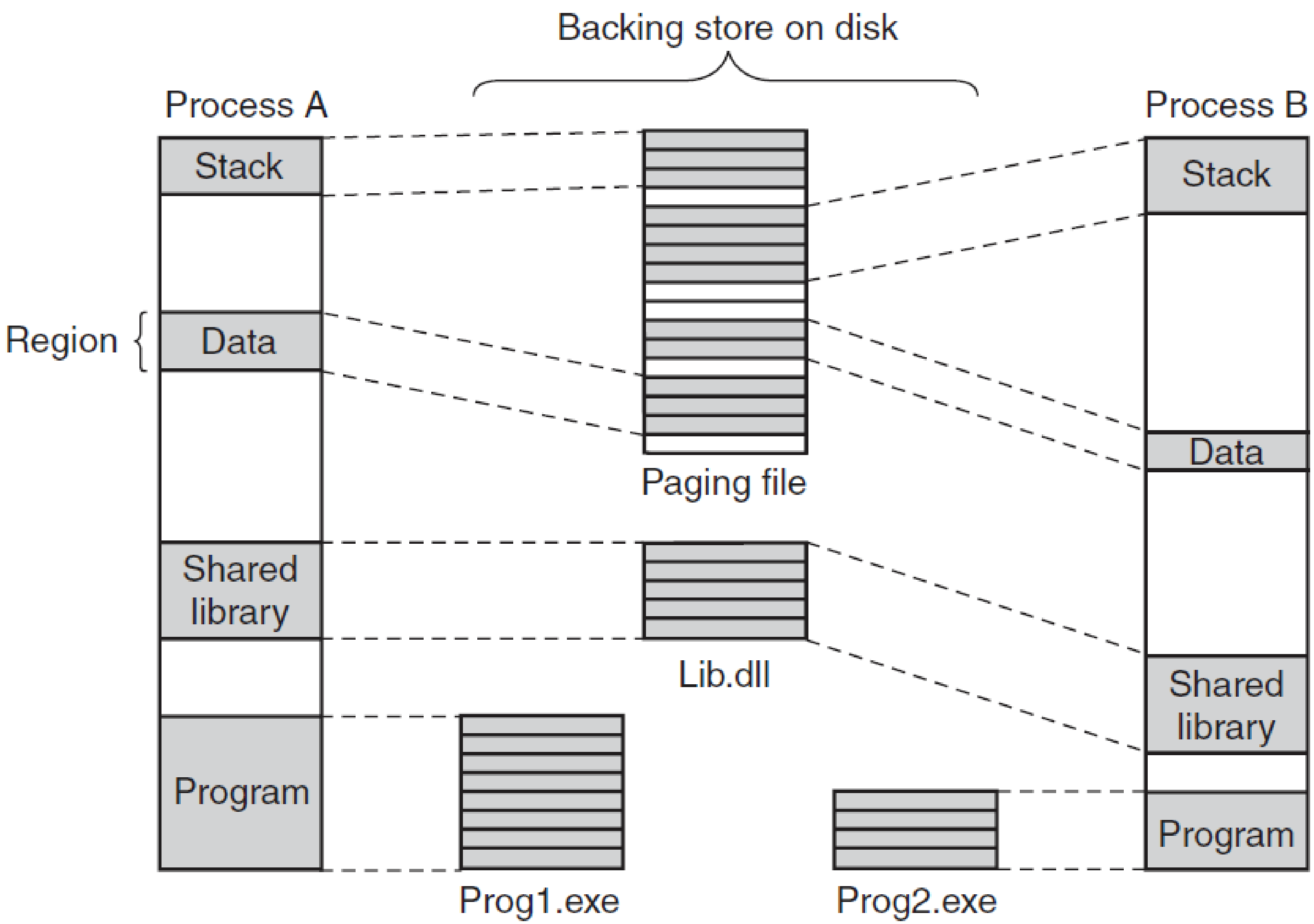


Figure 11-30. Mapped regions with their shadow pages on disk. The *lib.dll* file is mapped into two address spaces at the same time.

VAD (Virtual Address Descriptor)

- MM creates a VAD (Virtual Address Descriptor) for each process
  - Listing the range of addresses mapped, the section representing the backing store file and offset where it is mapped, and the permissions
  - An address space is completely defined by the list of its VADs
  - VADs are organized into a balanced tree

Page Fault Handling

- Types of pages faults:
  - The page referenced is not committed.
  - Access to a page has been attempted in violation of the permissions.
  - A shared copy-on-write page was about to be modified.
  - The stack needs to grow.
  - The page referenced is committed but not currently mapped in.

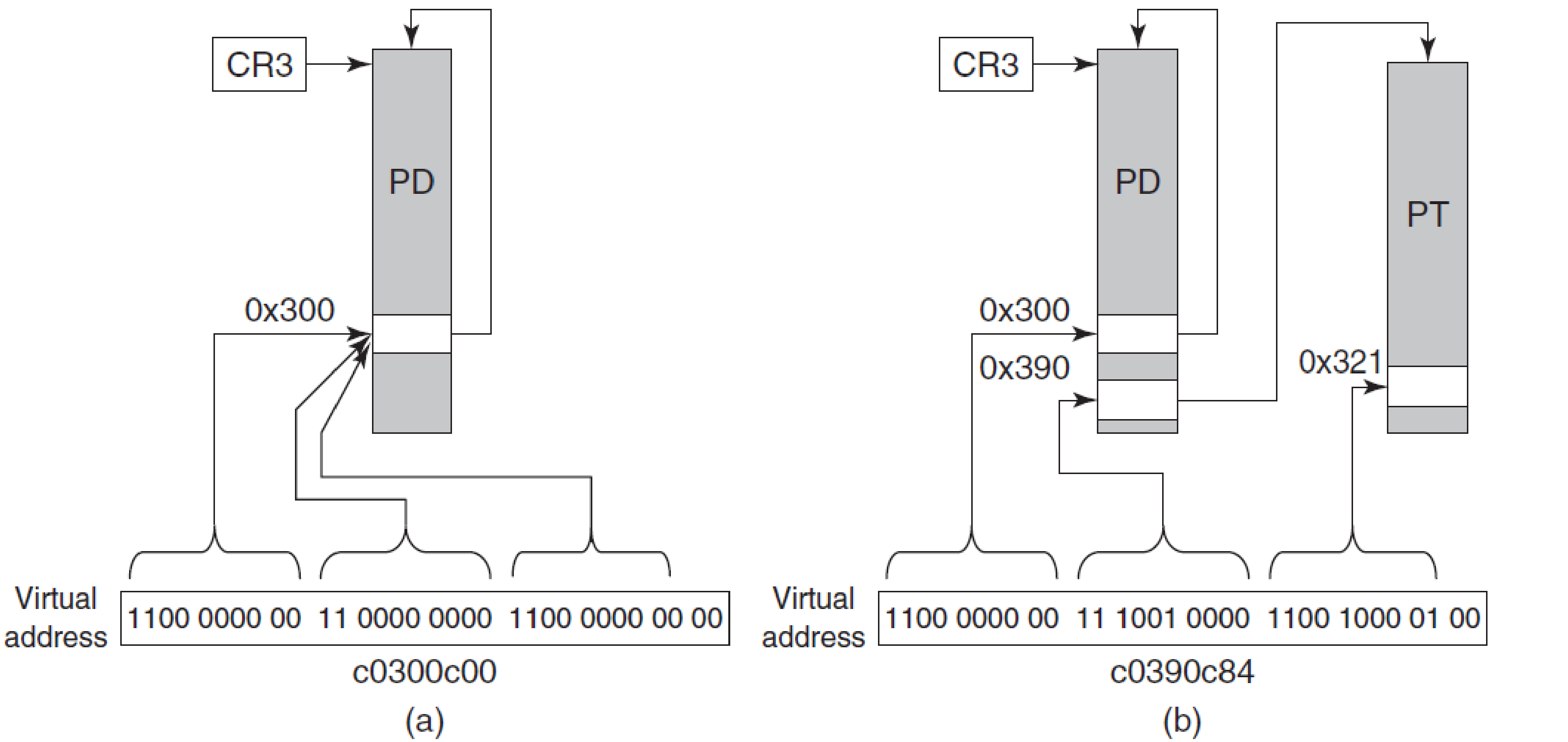
- How does each of these types occur?

Hard Faults vs. Soft Faults

- Hard fault: needs reading from disk
- Soft fault: the memory manager can satisfy a page fault by finding the needed page in memory rather than reading it in from disk
- When might a soft fault occur?
  - A shared page has already been mapped into another process
  - Only a new zero page is needed
  - The needed page was trimmed from the process' working set but is being requested again before it has had a chance to be reused
  - Pages have been compressed to effectively increase the size of physical memory.

Self-Map Entries

- The self-map entries are used to map the physical pages of the page tables and page directory into kernel virtual addresses (shown for 32-bit PTEs).



Self-map: PD[0xc0300000>>22] is PD (page-directory)
Virtual address (a): (PTE *)(0xc0300c00) points to PD[0x300] which is the self-map page directory entry
Virtual address (b): (PTE *)(0xc0390c84) points to PTE for virtual address 0xe4321000

Self-Map Entries

MiGetPdeAddress():

Given a virtual address *va*, compute its PDE

((PMMPTE)(((((ULONG)(*va*))>>22)<< 2) + PDE_BASE))

MiGetPteAddress():

Given a virtual address *va*, computer its PTE

((PMMPTE)(((((ULONG)(*va*))>>12)<< 2) + PTE_BASE))

Page Replacement Algorithm

- Working set
  - Each process' working set is described by two parameters: the minimum size and the maximum size.
    - Default minimum: 20–50 pages
    - Default maximum: 45–345 pages

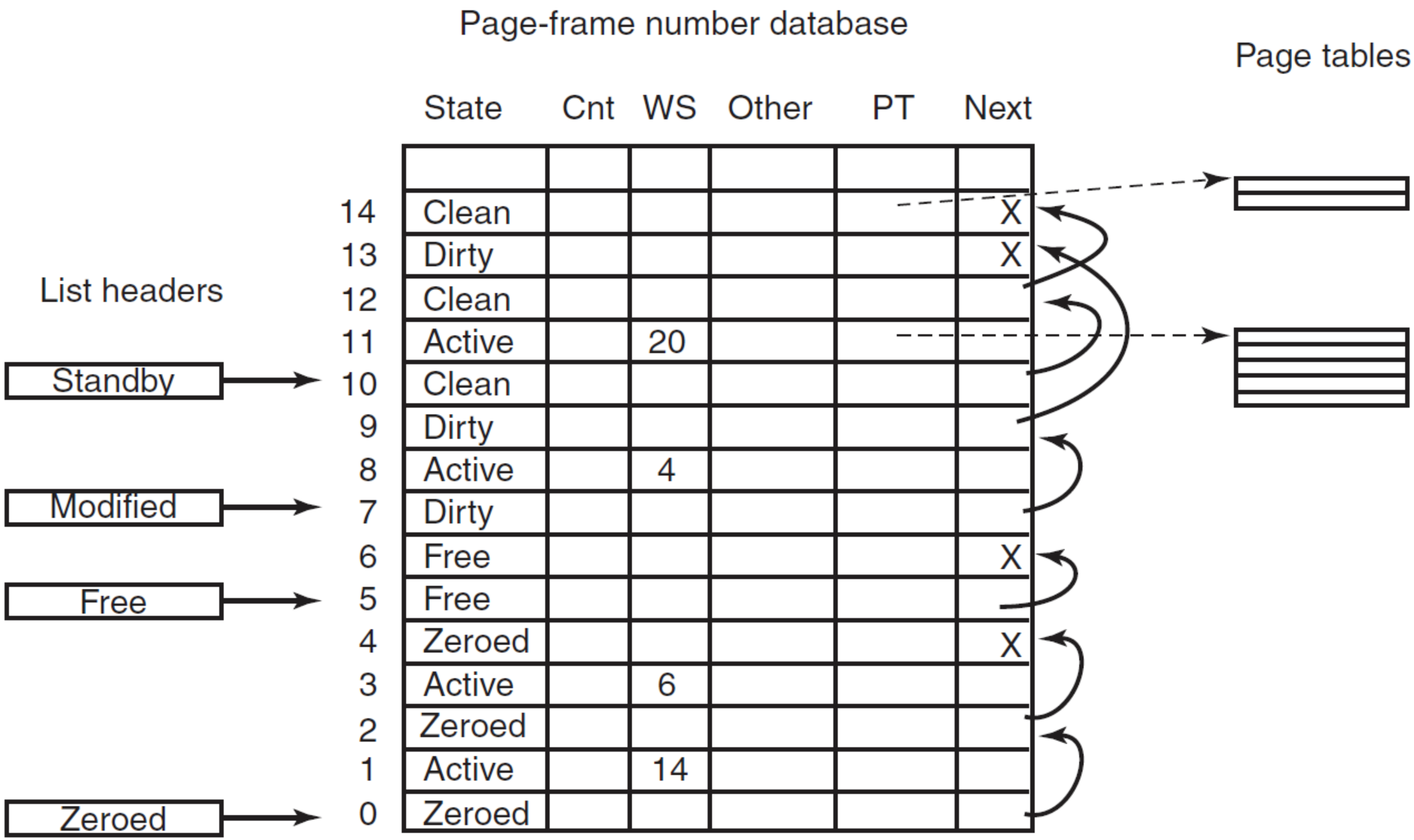
- The working set manager runs every second
  - The working-set manager throttles the amount of work it does to keep from overloading the system.

Physical Memory Management

- PFN (Page Frame Number) Database
  - All pages in the system either are referenced by a valid page-table entry or are on one of these five lists

- Different page frame lists
  - The free list
  - The standby list
  - The modified list
  - The zeroed list (free & zeroed)
  - The bad memory page list (frames with hardware errors)

Physical Memory Management



Transitions between Page Lists

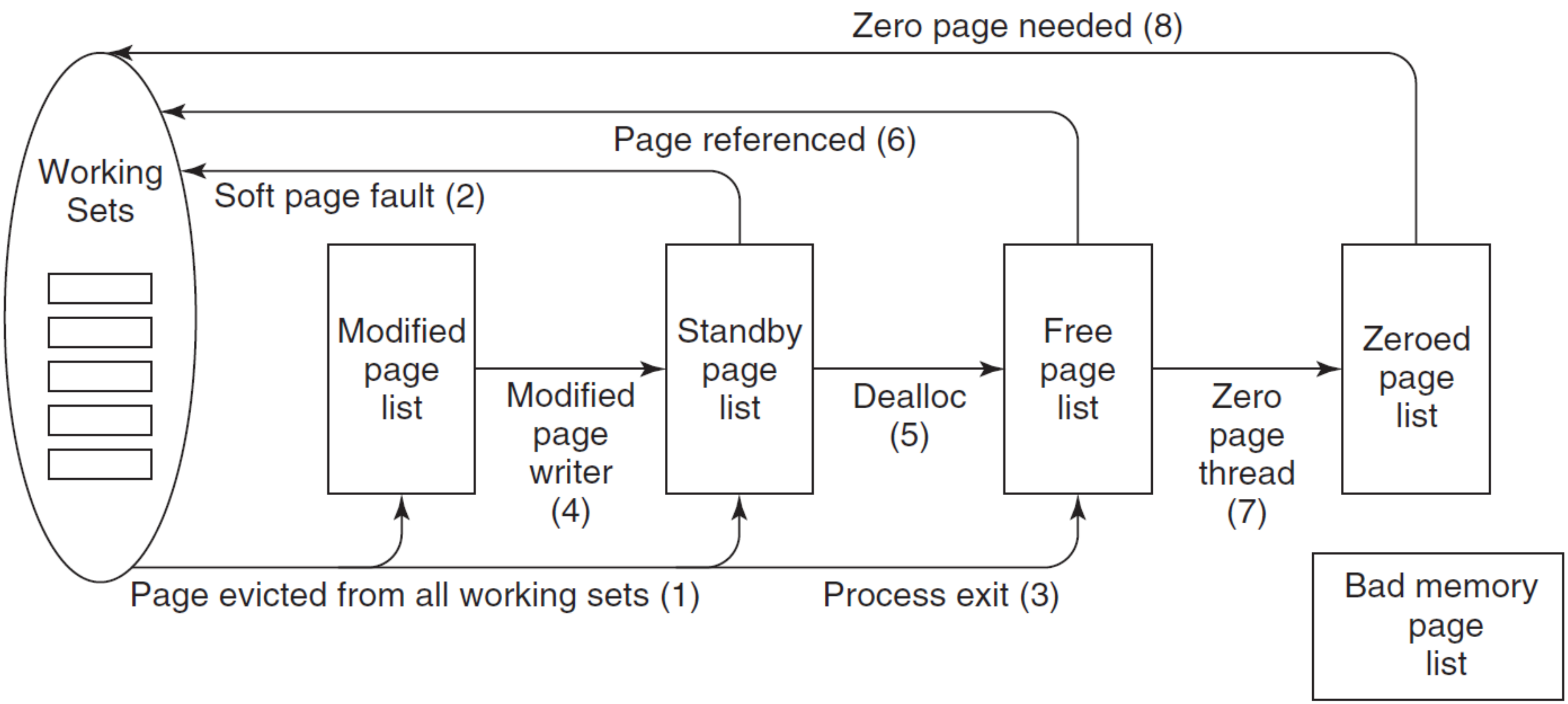


Figure 11-34. The various page lists and the transitions between them.

Summary

- Windows Memory Management
  - Virtual Address Descriptor
  - Self-Map Entries
  - Various Kinds of Page Lists

- Next Lecture: Scheduling