Demand Paging

> *Can a machine containing of 1 or 2 MB of physical memory execute processes whose sizes are 4 or 5 MB?*
> *Ans: Yes, by using demand paging*

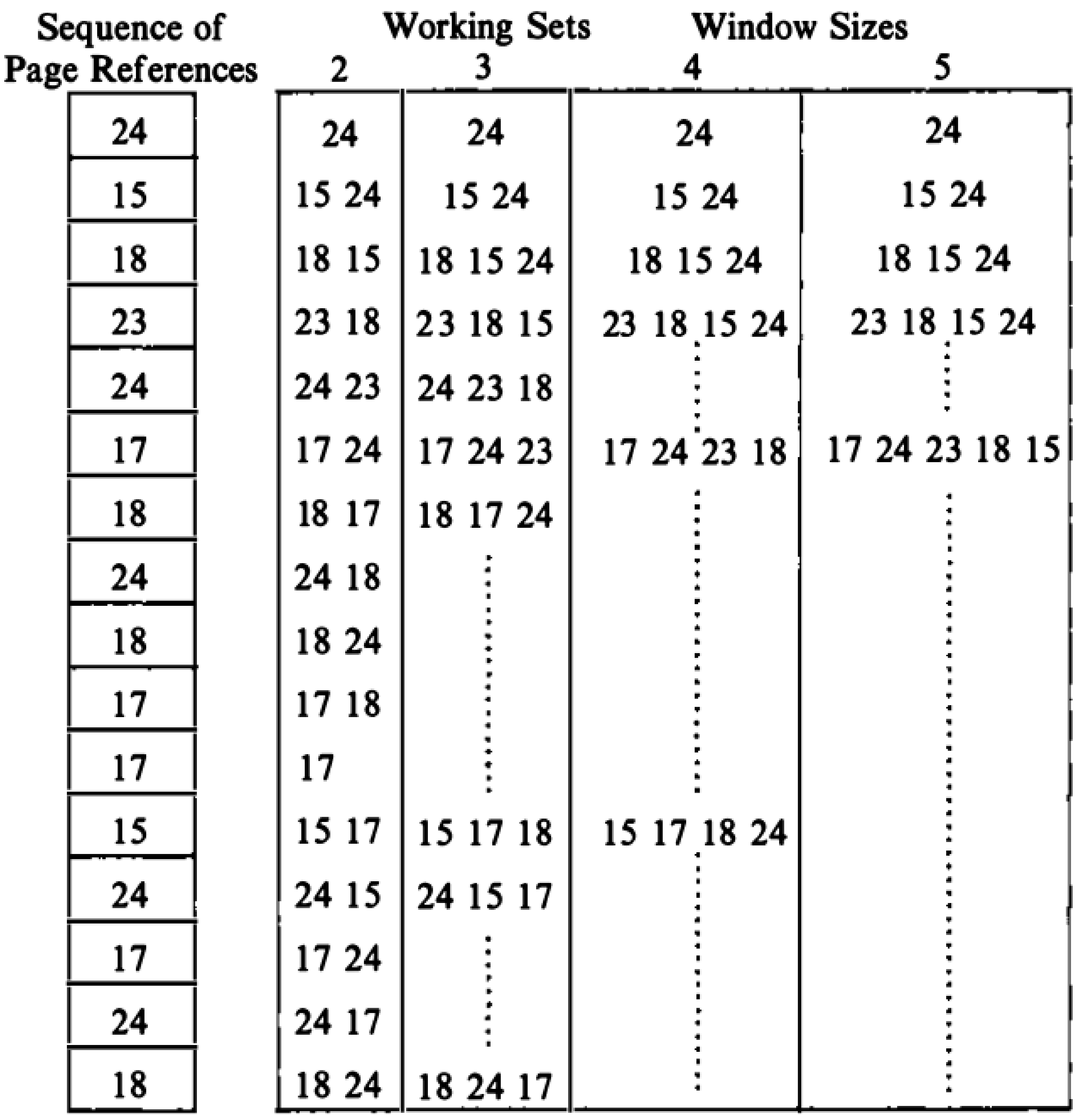
Locality

Processes tend to execute instructions in small portions of their text space, such as program loops and frequently called subroutines, and their data references tend to cluster in small subsets of the total data space of the process.

Working Set

The set of pages that the process has referenced in its last n memory references; the number n is called the window of the working set.

- It is a fraction of the entire process
- **More processes may fit simultaneously into main memory than in swapping system**, potentially **increasing system throughput** because of reduced swapping traffic.
- When a process addresses a page that is not in its working set, it incurs a **page fault**.
- To handle the page fault, the kernel updates the working set.
- The kernel adds the new page reference in the front of the list and removes the last referenced page from the working set. (No duplicates)
- This model is expensive and impractical.
- Instead, systems approximate a working set model by setting **reference bit** and **ages**.



Sequence of Page References	Working Sets Window Sizes			
	2	3	4	5
24	24	24	24	24
15	15 24	15 24	15 24	15 24
18	18 15	18 15 24	18 15 24	18 15 24
23	23 18	23 18 15	23 18 15 24	23 18 15 24
24	24 23	24 23 18		
17	17 24	17 24 23	17 24 23 18	17 24 23 18 15
18	18 17	18 17 24		
24	24 18			
18	18 24			
17	17 18			
17	17			
15	15 17	15 17 18	15 17 18 24	
24	24 15	24 15 17		
17	17 24			
24	24 17			
18	18 24	18 24 17		

Figure 9.12. Working Set of a Process

Paging

Implementation of a paging subsystem has 2 parts:

- Swapping rarely used pages to a swapping device
- Handling page faults

Data structures used

4 data structures:

- Page table entries
- Disk block descriptors
- Page frame data table (`pfdata`)
- swap use table

The kernel allocates space for the `pfdata` **once** for the lifetime of the system but allocates memory pages for the other structures dynamically.

Page table entries

- Region contains page tables to access physical memory.
- Each entry of a page table contains
  - The physical address of the page
  - Protection bits indicating whether the process can read, write, or execute from the page
  - The Following bits to support demand paging:
    - Valid
    - Reference
    - Modify
    - Copy on write
    - Age

VALID BIT

The kernel turns on the valid bit to indicate that the content of page are legal, but the page reference is not necessary illegal if the valid bit is off.

REFERENCE BIT

Whether a process recently referenced a page

MODIFY BIT

Whether a process recently modified the contents of a page

COPY ON WRITE BIT

- used in the `fork()`
- indicates the kernel must create a new copy of the page when a process modify its contents

AGE BIT

Kernel modifies the age bits to indicate how long a page has been a member of the working set of the process.

Disk block descriptors

- Describes the disk copy of the virtual page
- Processes that share a region therefore access common page table entries and disk clock descriptors
- The contents of the of the virtual page are either in a **particular block on the swap device**, in **an executable file**, or not on a swap device
    - If the page is on a swap device, the **disk block descriptor contain the logical device number** and **block number** containing the page contents
    - If the page is contained in an executable file, the **disk block descriptor contains the logical block number** in the file that contains the page; the kernel can quickly map this number into its disk address.

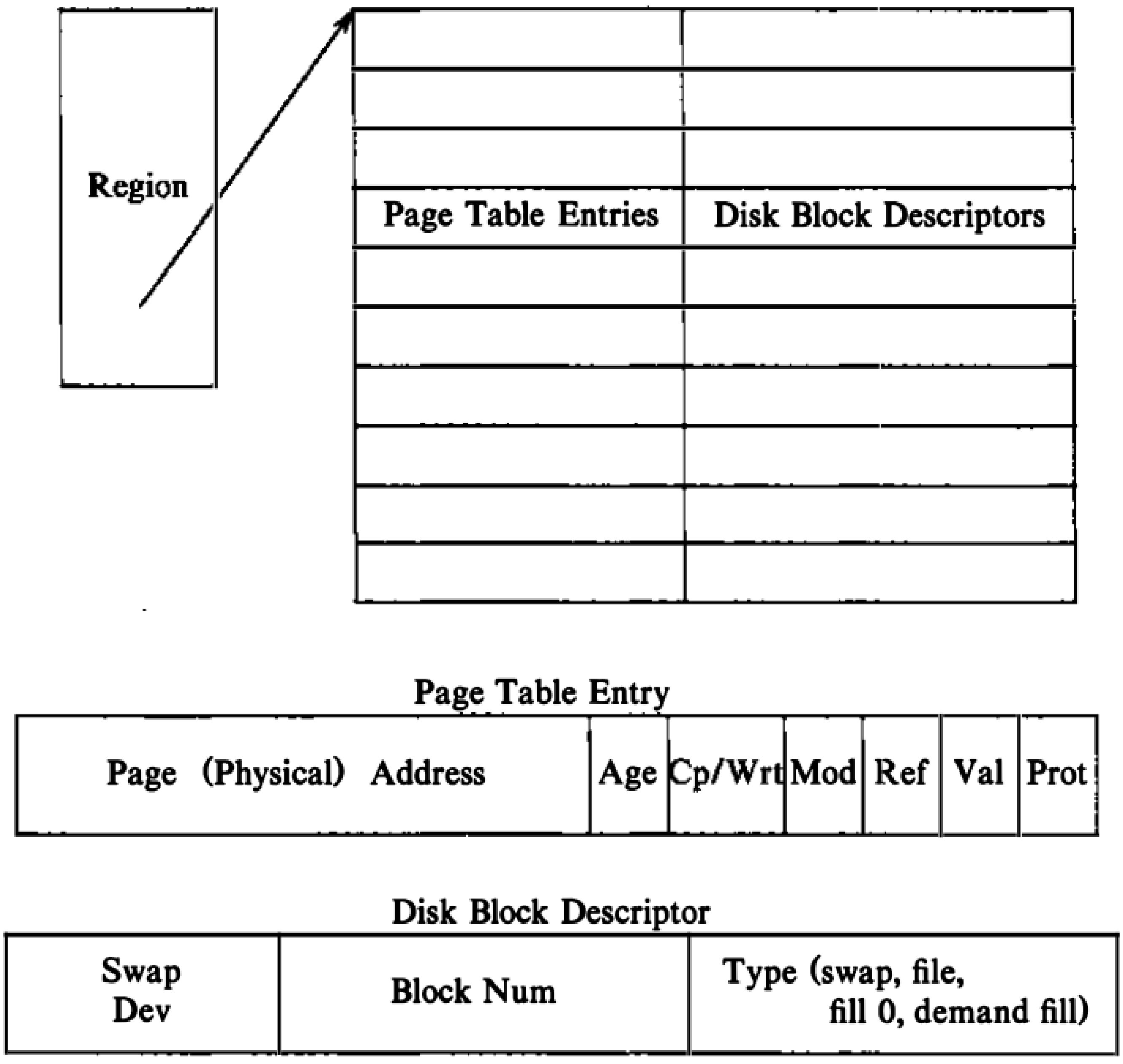


Figure 9.13. Page Table Entries and Disk Block Descriptors

pfdata table

- Describes each page of physical memory and is indexed by page number. The fields of an entry are:
    - The **page state**, indicating whether the page is on a swap device or exec file
    - The **number of process that reference the page**
        - The **reference count equals the number of valid page table entries** that reference the page
        - It may differ from the number of processes that share regions containing the page

> - The **logical device** (swap or file system) and a **block number** that contains the copy of the page
> - **Pointers** to other `pfdata` table entries on a list of free pages and on a hash queue of pages.

FREE LIST

- A cache of pages that are available for **reassignment**.
- A process may fault on an address and still find the corresponding page intact on the free list.
- Allows the kernel to avoid unnecessary read operations from the swap device.
- **Allocates new pages from the list in least recently used order**.
- The kernel also **hashes the pfdata entry according to its swap device number and block number**.
- To **assign a physical page to a region**, the kernel removes a free page frame entry from the head of the free list, updates its swap device and block numbers, and puts it onto the correct hash queue.

Swap use table

- Contains an entry for every page on a swap device
- The entry consists of a r**eference count of how many page table entries point to a page on a swap device**.

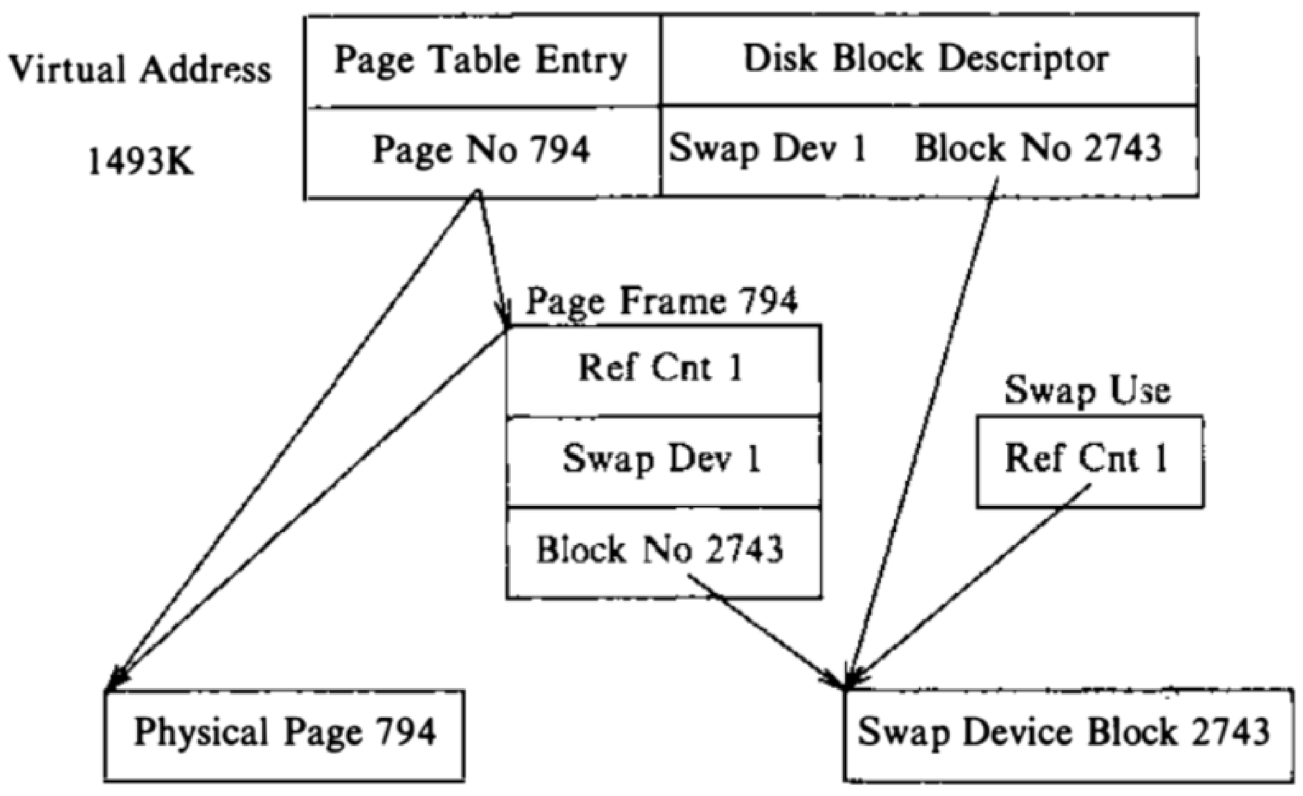


Figure 9.14. Relationship of Data Structures for Demand Paging

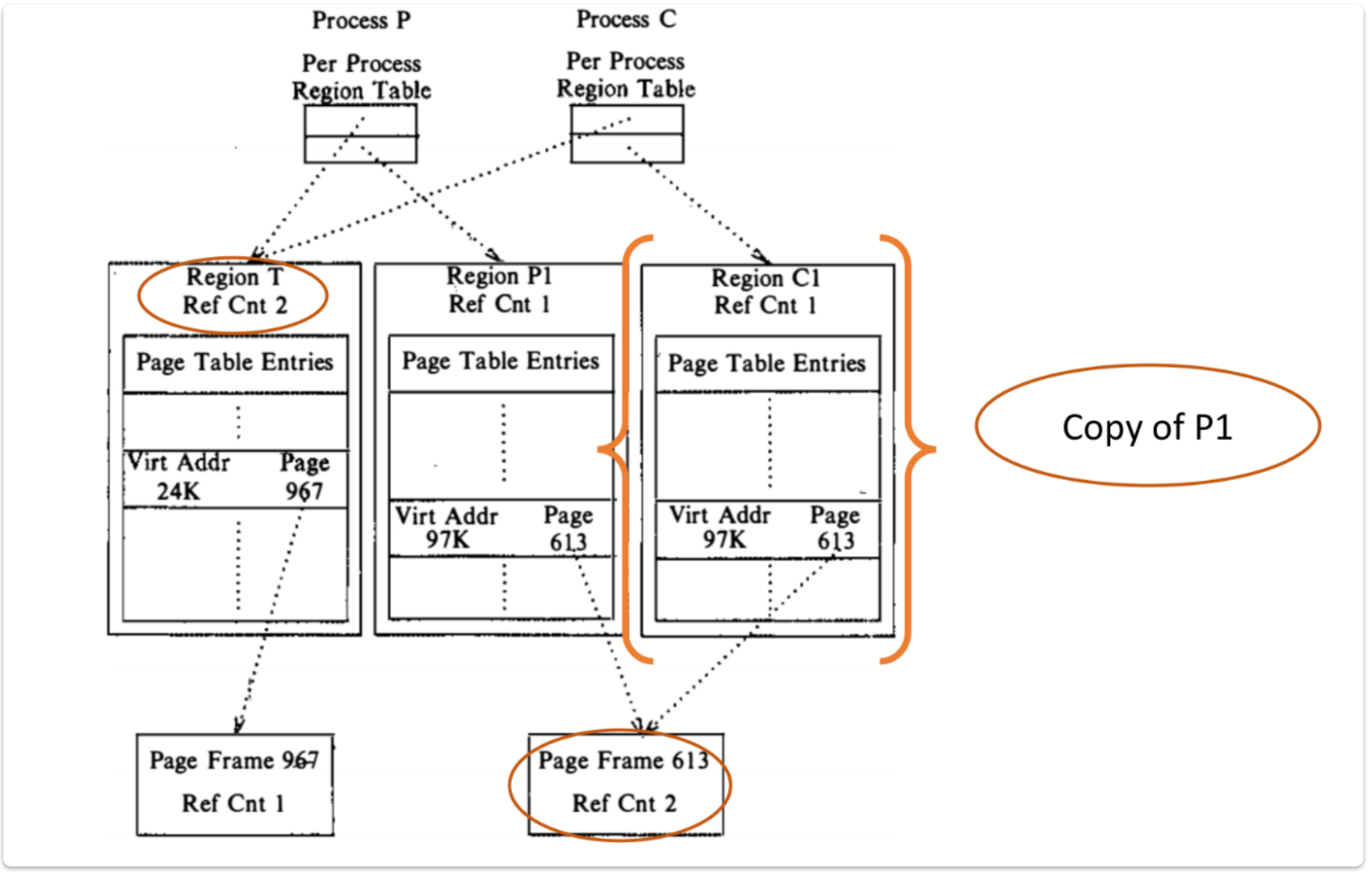
The page on the swap device is a copy of the physical page.

Fork in a paging system

- The kernel duplicates every region of the parent process during `fork()` and attaches it to the child.
- The kernel avoids copying pages by manipulating region table, page table entries, and pfdata table entries.
- Increments the region reference count of shared regions.
- For a private regions such as data and stack, allocates a new region table entry and page table.
- Examines the each parent page table entry,
  - If a page is valid, increments the reference count in the pfdata table entry
  - If the page exists on a swap device, increments the swap use reference count.
- If a process writes on a shared page
  - The kernel copies the page so that each region has a private version
  - To do this, the kernel turns on the **copy on write** bit
  - If either process writes the page, it incurs a protection fault
  - In handling the fault, the kernel makes a new copy of the page for the faulting process.

- The physical copy is thus deferred until the process really needs it.

In the below diagram, Region P1 is copied to form a new region C1.



Exec in a paging system

- The kernel reads the executable file into memory from the file system.
- On a demand pages system
  - The executable file may be too large to fit in the available main memory
  - The kernel, therefore, does not preassign memory to the executable file but "faults" it in, assigning memory as need.
  - It first assigns the page tables and disk block descriptors for the executable file, marking the page table entries "demand fill" (meaning the contents of the page will be overwritten immediately with the contents of the executable file) or "demand zero".
  - The fault handler notes if the page is marked "demand zero" or "demand fill".
  - If the process cannot fit into memory, the page stealer process periodically swaps pages from memory, making room for the incoming file.
- Inefficiencies:
  - First, a process incurs a page fault when reading each page of the exec file, even though it may never access the page
  - Second, the page stealer may swap pages from memory before the exec is done, resulting in 2 extra swap operations per page if the process needs the page early.

- To make `exec` more efficient, the kernel can demand page directly from the executable file if the data is properly assigned, as indicated by a special magic number.

Page Stealer Process

- Swaps out pages that are no longer part of the working set of a process.
- The kernel creates it during system intialization
- Examines every active, unlocked region, and increments the age field of all valid pages.
- The **kernel locks a region when a process faults on a page in the region**, so that the page stealer cannot steal the page being faulted in.
- Two paging states
  - **The page is ageing and not yet available for swapping**
  - **The page is available for swapping and is available for reassignment to other virtual pages**.

THE PAGE IS AGING AND IS AVAILABLE FOR SWAPPING

- Indicates the process recently accessed the page because it is aging and is therefore in the working set.
- The page stealer turns off the reference bit for such pages but remembers how many examinations has passed since last reference.
- When the number exceeds a threshold value, the kernel puts the page into the second state, **ready to be swapped**.

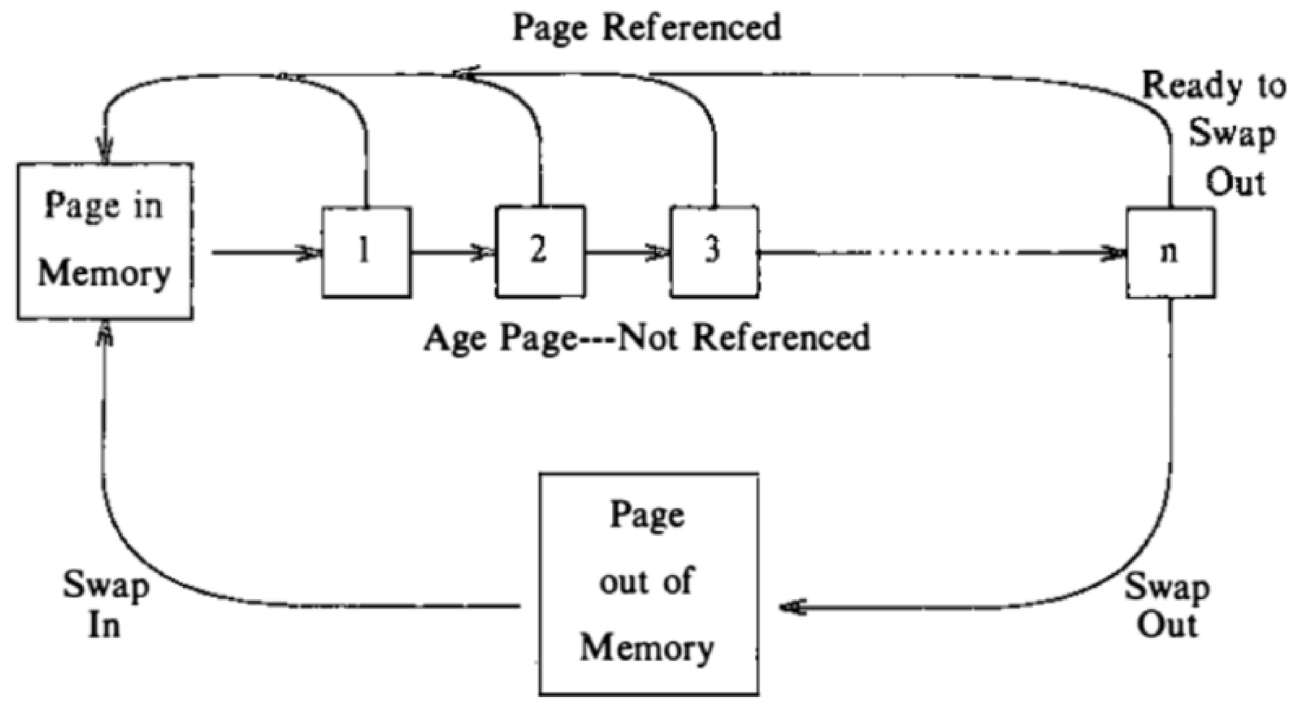
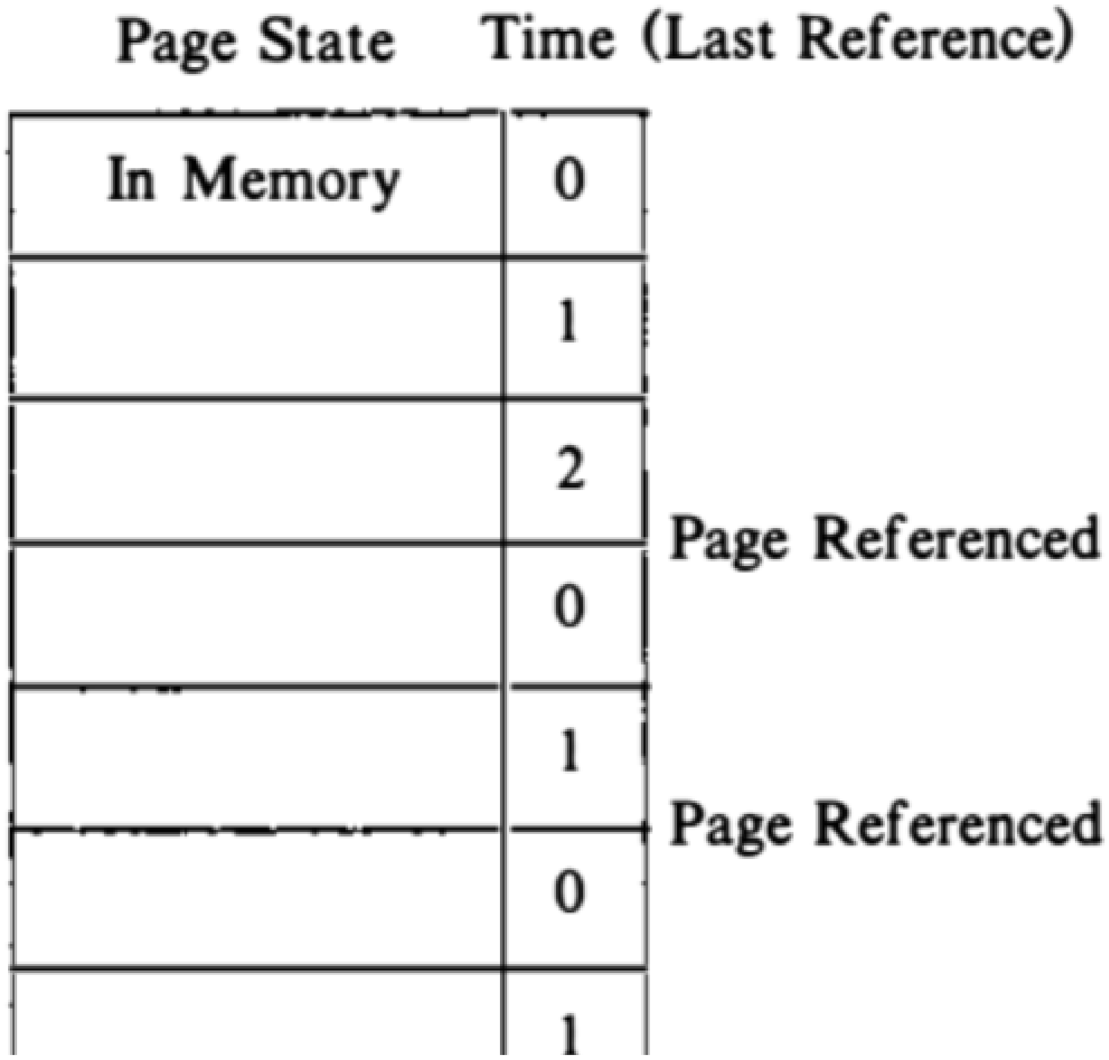
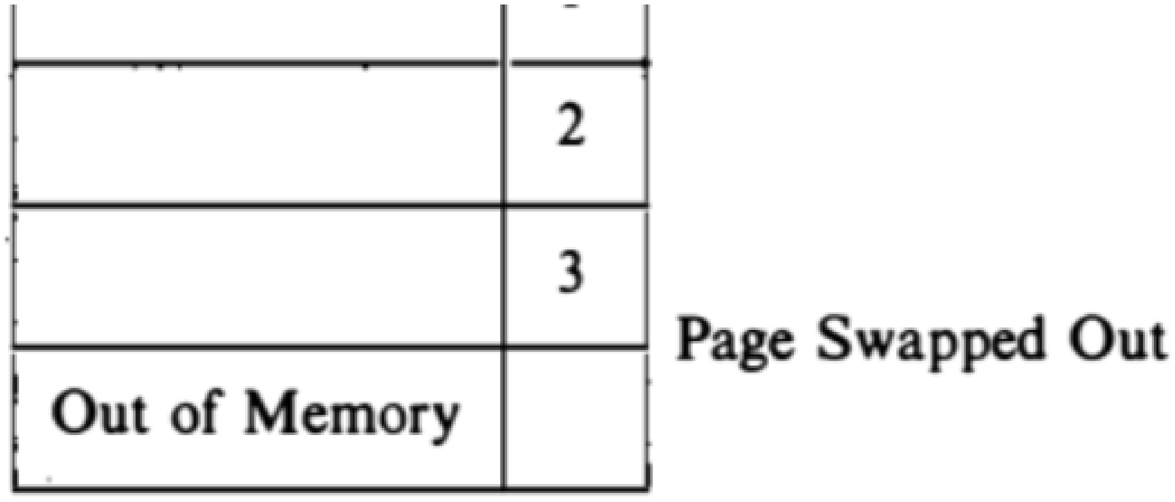


Figure 9.18. State Diagram for Page Aging



Page State	Time (Last Reference)	
In Memory	0	
	1	
	2	
	0	Page Referenced
	1	
	0	Page Referenced
	1	



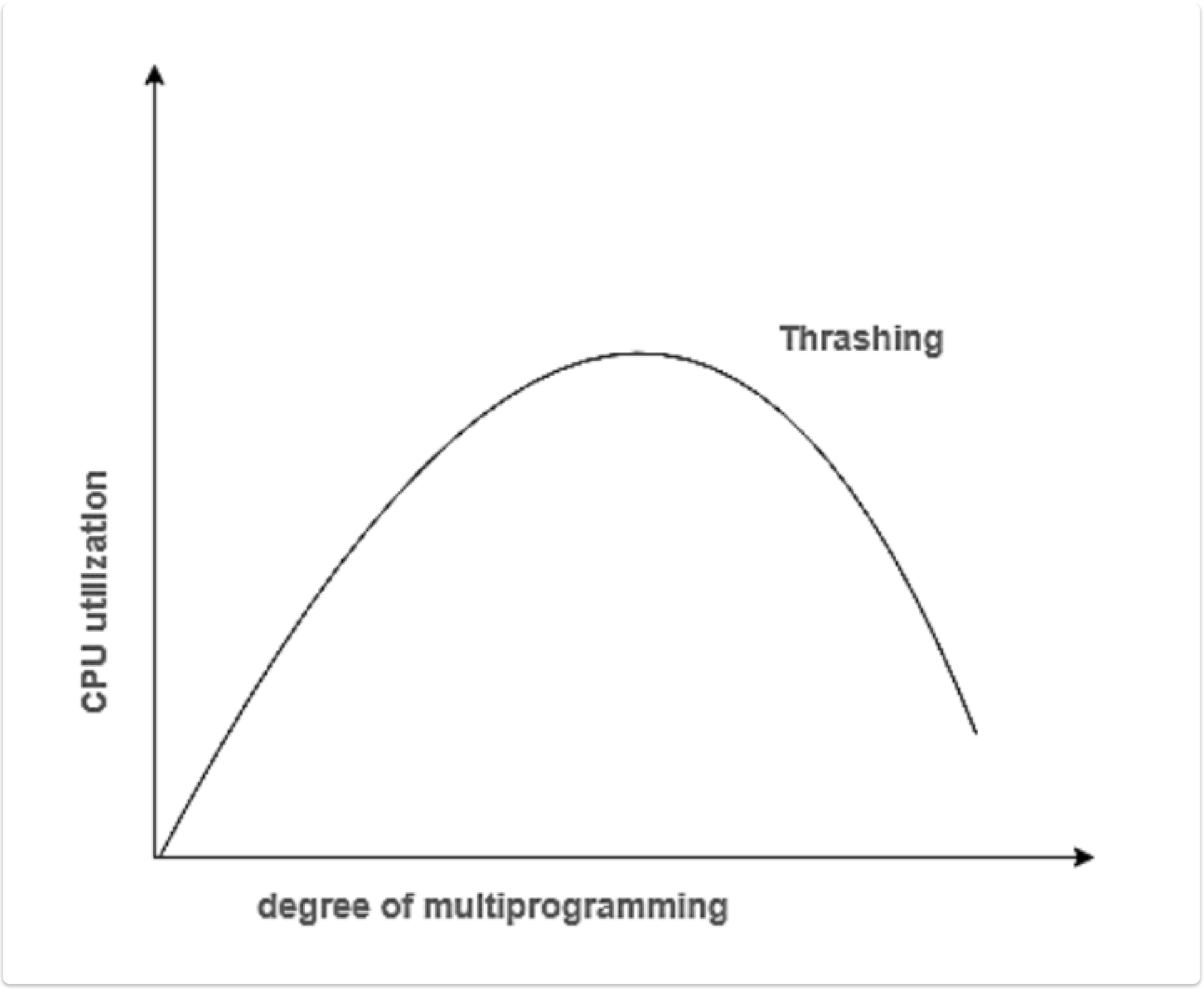
TWO OR MORE PROCESSES SHARE A REGION

- Processes **update the reference bits** of same set of page table entries.

- Pages can be part of the working set of more than one process.
- The page stealer does not attempt to swap out equal number of pages from all active regions.

LOW WATER MARK AND HIGH WATER MARK

- Kernel wakes up the page stealer when the available free memory in the system is below a **low water mark**
- Swaps out pages until the available free memory exceeds the a **high water mark**.
- Effectively works when memory is in the range [low water mark, high water mark]
- This is done to reduce **thrashing** (when most of the time is gone in servicing page faults rather than actual processing).



When the page stealer decides to swap out a page?

- If no copy of the page is on the swap device
- If the copy of the page is on the swap device and no process had modified it in core contents
- If the copy of the page is on the swap device and a process had modified it in core contents in the memory.

The page stealer fills the list of pages to be swapped, possibly from different regions, and swaps them to a swap device when the list is full.

When the kernel writes a page to a swap device

- Turns off the valid bit
- Decrements the use count of its pfdata table entry.
- If count is 0, it places the pfdata entry at the end of the free list, caching it until reassignment.

- If the count is non 0, several processes are sharing the page, but the kernel still swaps the page out (because its not used in quite some time).
- Finally, the kernel allocates swap space, saves the swap address in the disk block descriptor, and increments the swap-use table count.
- If the **process incurs a page fault while the page is on the free list**, however, t**he kernel can rescue the page from memory** (free list) instead of having to retrieve it from the swap device.

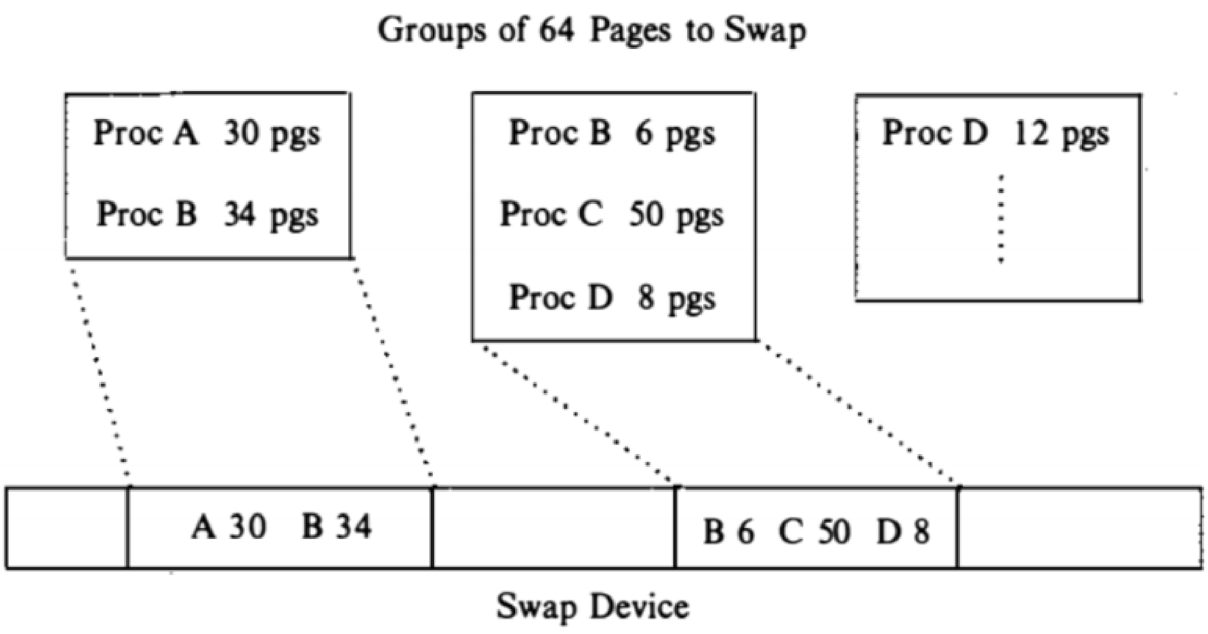


Figure 9.20. Allocation of Swap Space in Paging Scheme

Suppose the page stealer swaps out $30, 40, 50, 20$ pages from A, B, C, and D respectively
Page stealer can swap out 64 pages in one go (filling the list), so it will swap out 30 pages of A and the remaining it can swap is $64 - 30 = 34$ pages of B, the next 6 pages will be swapped out in the next iteration, and lastly, it will not swap out the 12 pages of D because the list is not full.

Page Faults

Types

- Validity faults
- Protection faults

fault handlers are an exception to the general rule that interrupt handlers cannot sleep, because they can sleep when the page is being brought in the memory by a means of an I/O operation

Validity Fault Handler

- Valid bit is not set $\rightarrow$ kernel invokes validity fault handler
- Valid bit is not set for pages outside the virtual address space of a process and for pages that are part of the virtual address space but not assigned a physical page.
- The kernel locks the region containing the page table entry to prevent page stealer from the stealing a page
- If the disk block descriptor has record of the faulted page
    - The attempted memory reference is invalid
    - kernel sends a segmentation violation to the offending process

STAGES OF THE FAULTED PAGE

- On a swap device not in memory
- On the free page list in memory
- In an executable file
- Marked "demand zero"
- Marked "demand fill"

Case 1: On a swap device but not in memory

- Once resided in memory but the stealer swapped it out
- From the disk block descriptor, it finds the swap device and block number and verifies that the page is not in page cache.
- Updates the page table entry to point to the page about to be read in, places the pfdata table entry on the hash list to speed up fault handler operation
- Faulting process sleeps until the I/O is done.
- In the below diagram, validity fault occurs on address 66K and swap device block number is read in which verifies it as a legal page.
- After its been swapped, a physical page is assigned to it and state is changed to "valid".

	Page Table Entries		Disk Block Descriptors	
Virt Addr	Phys Page	State	State	Block
0				
1K	1648	Inv	File	3
2K				
3K	None	Inv	DF	5
4K				
⋮				
64K	1917	Inv	Disk	1206
65K	None	Inv	DZ	
66K	1036	Inv	Disk	847
67K				

Page Frames		
Page	Disk Block	Count
1036	387	0
⋮		
1648	1618	1
⋮		
1861	1206	0

Figure 9.22. Occurrence of a Validity Fault

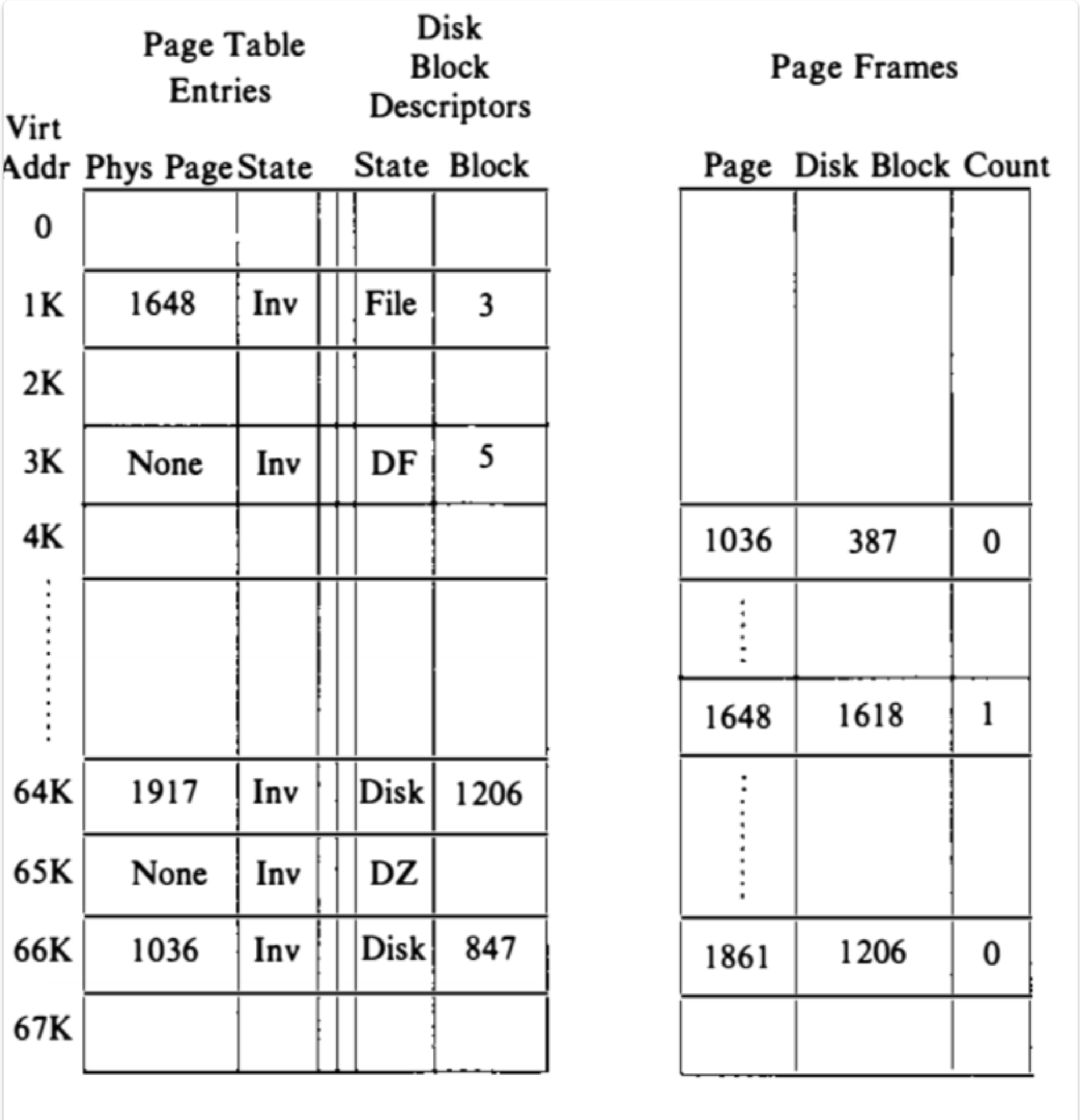
	Page Table Entries		Disk Block Descriptors	
Virt Addr	Phys Page	State	State	Block
66K	1776	Val	Disk	847

Page Frames		
Page	Disk Block	Count
1776	847	1

Figure 9.23. After Swapping Page into Memory

Case 2: On the free page list in memory

- Possible that the kernel has never reassigned the physical page after swapping it out
- Fault handler finds the page in the page cache, keying off the block number in the disk block descriptor
- Reassigns the page table entry to the point where the page was found, increments its page reference count, and removes the page from the free list.
- In the below diagram, it finds the page on the page frames table wit disk block 1206.



	Page Table Entries		Disk Block Descriptors	
Virt Addr	Phys Page	State	State	Block
0				
1K	1648	Inv	File	3
2K				
3K	None	Inv	DF	5
4K				
⋮				
64K	1917	Inv	Disk	1206
65K	None	Inv	DZ	
66K	1036	Inv	Disk	847
67K				

Page Frames		
Page	Disk Block	Count
1036	387	0
⋮		
1648	1618	1
⋮		
1861	1206	0

Figure 9.22. Occurrence of a Validity Fault

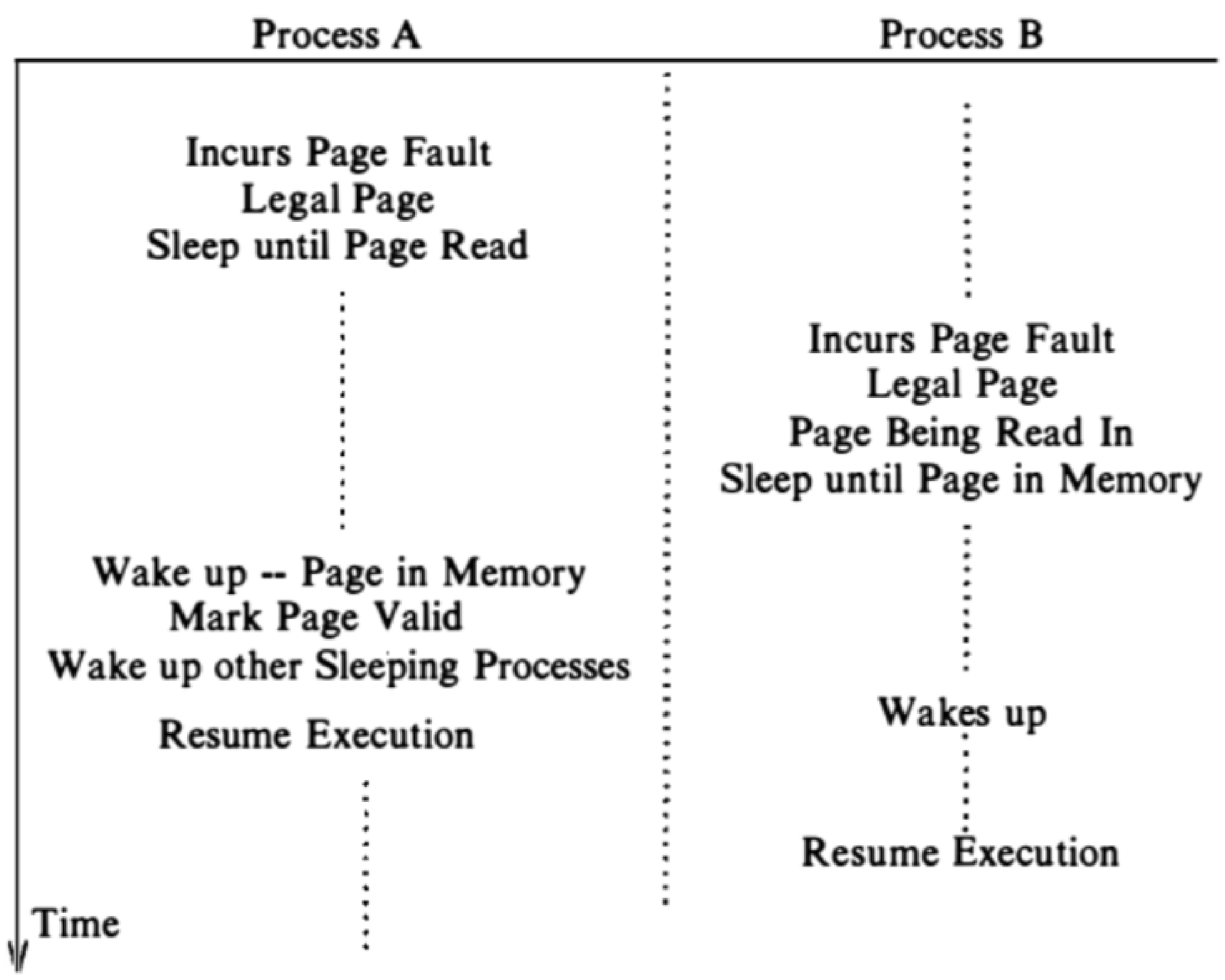


Figure 9.24. Double Fault on a Page

In the above diagram, both wanted to access the same page, after being read in, they continued their execution.

Case 3: A copy of the page is in the original exec file but not in swap device

- The fault handler examines the disk block descriptor.
- Find the logical block number in the file that contains the page, and find the inode associated with the region table entry.
- Uses logical block number as an offset into array of disk block numbers attached to the inode during exec.
- Knowing the disk block number, it reads the page into memory

Case 4 or 5: Incurs a page fault for a page marked "demand zero" or "demand fill"

- Allocates a free page in memory and updates the appropiate page table entry
- For demand zero, it clears the page

- Clears the "demand zero" and "demand fill" flags
- The page is now in memory unduplicated.

Protection Fault Handler

WHEN IT OCCURS?

- The process accessed a valid page but the permission bits did not permit access
- Attempts to write a page when its copy on write bit is set during `fork()`.

STEPS

- The fault handler finds the appropiate region and page table entry

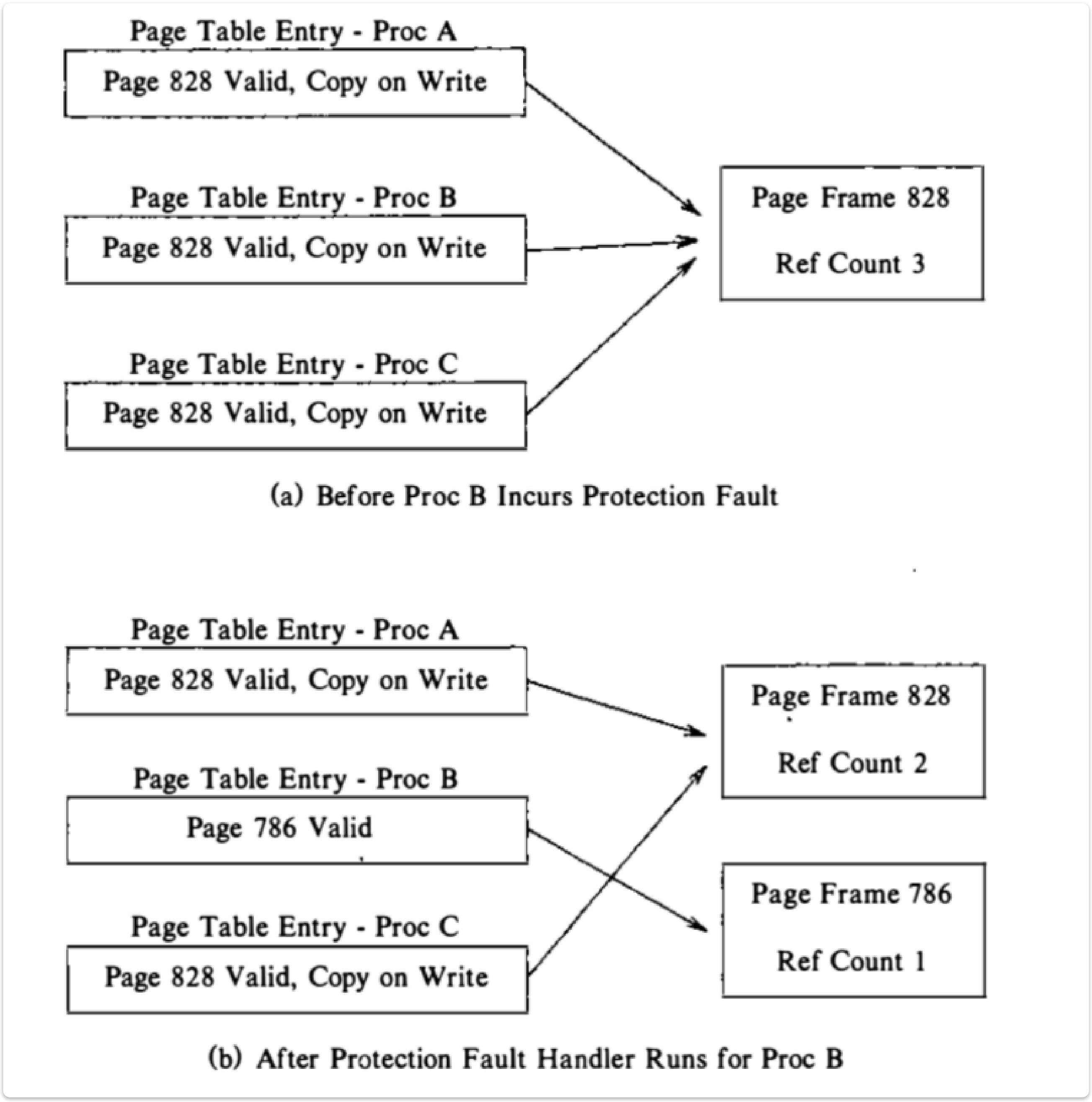
- Locks the region

- If the fault handler determines its copy on write bit was set

  > - If the page is shared with other processes
  > - The kernel allocates a new page
  > - Copies the old page into it
  > - The other processes retain their references to the old page
  > - The reference count of the old pfdata table page is decremented

- If the copy on write was set but no other process were sharing it (ref. ct = 1), then the kernel allows the process to reuse the physical page, turns off the copy on write bit
  - Disassociates the page from the disk copy, bec. others may be using it
  - removes the pfdata table entry

- decrements the swap-use count


Page Table Entry - Proc A
Page 828 Valid, Copy on Write
Page Table Entry - Proc B
Page 828 Valid, Copy on Write
Page Table Entry - Proc C
Page 828 Valid, Copy on Write
Page Frame 828
Ref Count 3
(a) Before Proc B Incurs Protection Fault
Page Table Entry - Proc A
Page 828 Valid, Copy on Write
Page Table Entry - Proc B
Page 786 Valid
Page Table Entry - Proc C
Page 828 Valid, Copy on Write
Page Frame 828
Ref Count 2
Page Frame 786
Ref Count 1
(b) After Protection Fault Handler Runs for Proc B

- If the page table entry is invalid and its copy on write bit is set
    - Assume that the validity fault is handled first
    - The protection fault handler must check that the page is still valid, because it could sleep when locking a region, and the page stealer may steal it
    - If the page is invalid, the protection fault handler returns immediately.
    - If the kernel handles the validity fault, but the process will incur a protection fault again. Now it will handle without interference.
- When the protection fault handler finishes
    - It sets the modify and protection bits

- Clears the copy on write
- Recalculates the process priority and checks for signals.

Demand Paging on less sophisticated hardware

- Possible to implement the algos described if only valid and protection bits were recognised by the hardware.
- If the valid bit os duplicated by a s/w valid bit, it can simulate any other bit with that hardware bit
- Scenario
  - A page fault occurs when the valid bit is off
  - The page fault handler examines the page
  - Because s/w valid bit is set, page is valid and sets the s/w reference bit
  - Turns on the s/w valid bit, acquired the knowledge that the page is referenced

Hybrid system with demand paging and swapping

- Situations can arise where the page stealer and validity fault handler can thrash
- If sum of all working sets of processes > physical memory, fault handler will sleep.
- Page stealer will not be able to steal pages fast enough, system throughput goes down

SOLUTION: HYBRID SYSTEM (FOLLOWED IN SYSTEM V)

- When the kernel cannot allocates pages for a process, it wakes up the swapper and puts the process in "ready to run but swapped" space.
- The swapper swaps out entire processes until avail. mem. exceeds the highwater mark
- For each swapped process, the swapper makes one "ready to run but swapped" process "ready to run".