CISC 7310X
C08: Virtual Memory

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Concepts of virtual address space, paging, virtual page, page frames
- Design of virtual memory system for efficiency
  - Concept of page table
  - Speeding-up page table
    - TLB
  - Large virtual address space
    - Multilevel page tables
    - Inverted page tables
  - Page replacement algorithms

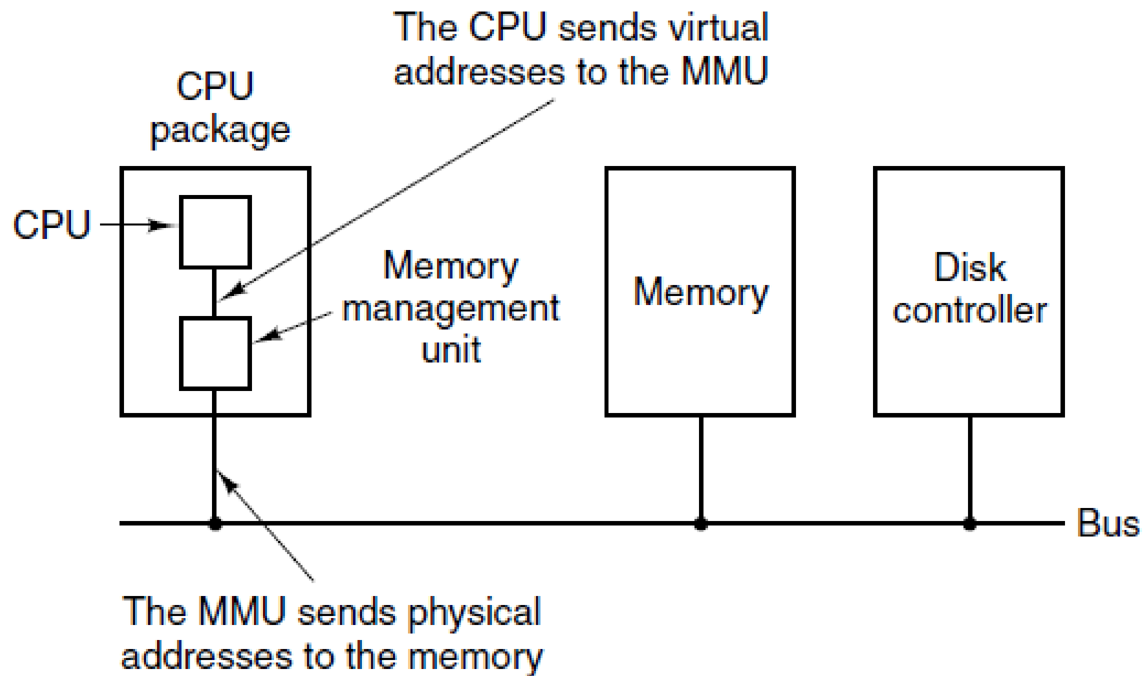
Virtual Memory

- Two problems
  - A program may become too large to fit in memory
  - Collectively in a multiprogramming system, the programs exceed memory although each fits.
- Virtual Memory
  - Address space
    - each program has its own address space
  - Paging
    - The address space broken up into chunks called pages, each is a contiguous range of addresses
    - Pages are mapped onto physical memory, but not all at the same time

Virtual Address

- Programs access data referencing memory with virtual address
- Example
  - MOV REG, 1000
- What happens in today's computer systems?
  - Virtual address is passed to a Memory Management Unit (MMU)
  - MMU maps the virtual address onto the physical memory

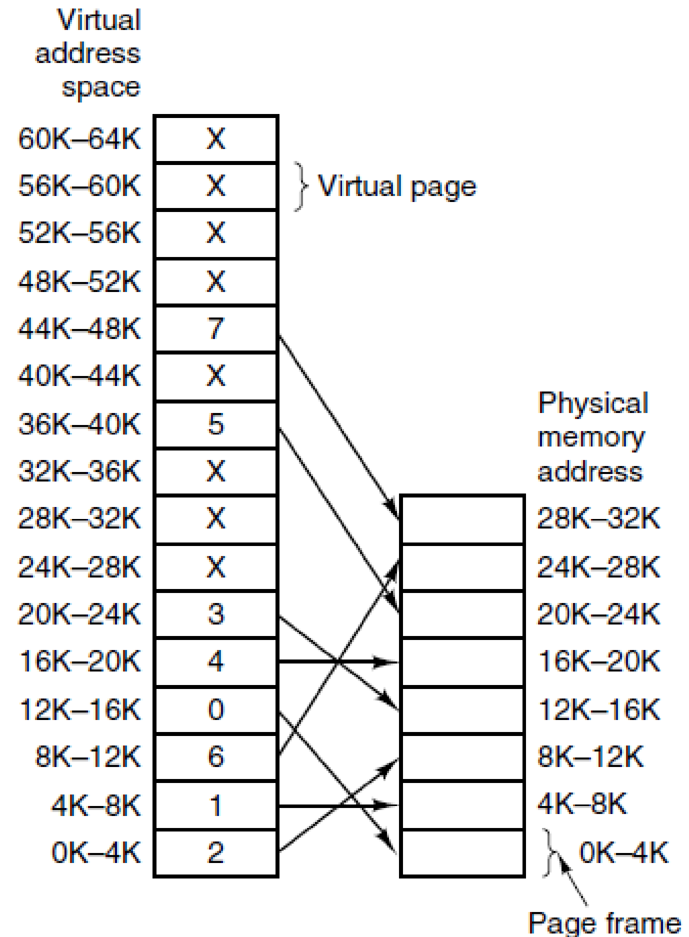
Virtual Address to Physical Address



- [Figure 3-8 in Tanenbaum & Bos, 2014]

Example: Virtual Address

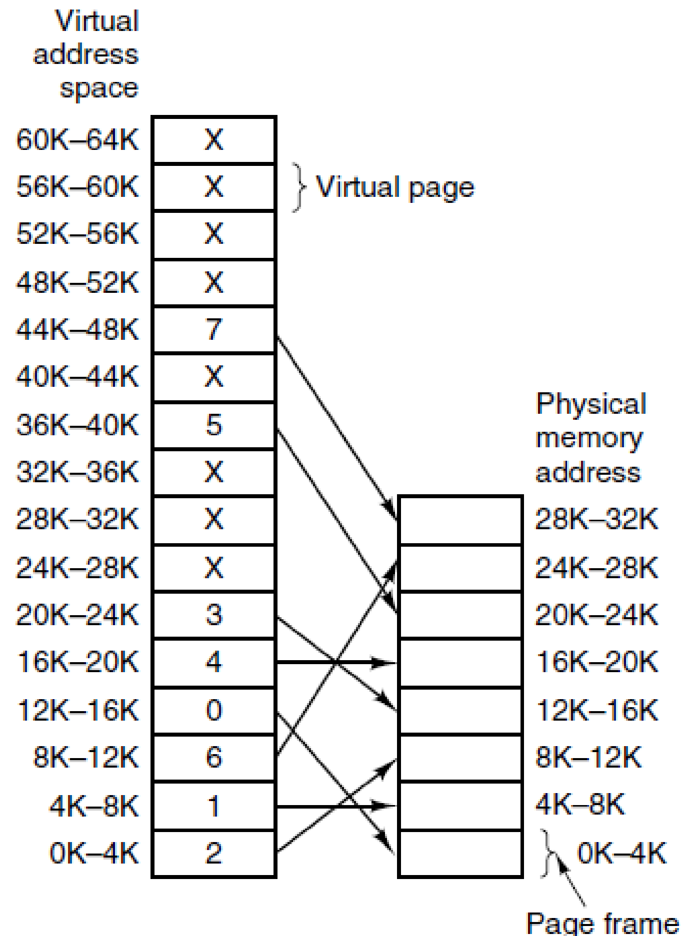
- Virtual address
  - 16-bit address
  - Address space
    - 0 ~ (2^{16} – 1 = 64K - 1)
    - Divided into pages, each 4KB
- 32 KB physical memory
  - Page frames: pages in the physical memory
- 64 KB virtual space: 16 x 4 = 64, so 16 virtual pages
- 32 KB physical memory: 8 x 4 = 32, so 8 page frames
- Transfer between memory and disk is always in whole pages



- [Figure 3-9 in Tanenbaum & Bos, 2014]

Example: Access Virtual Address

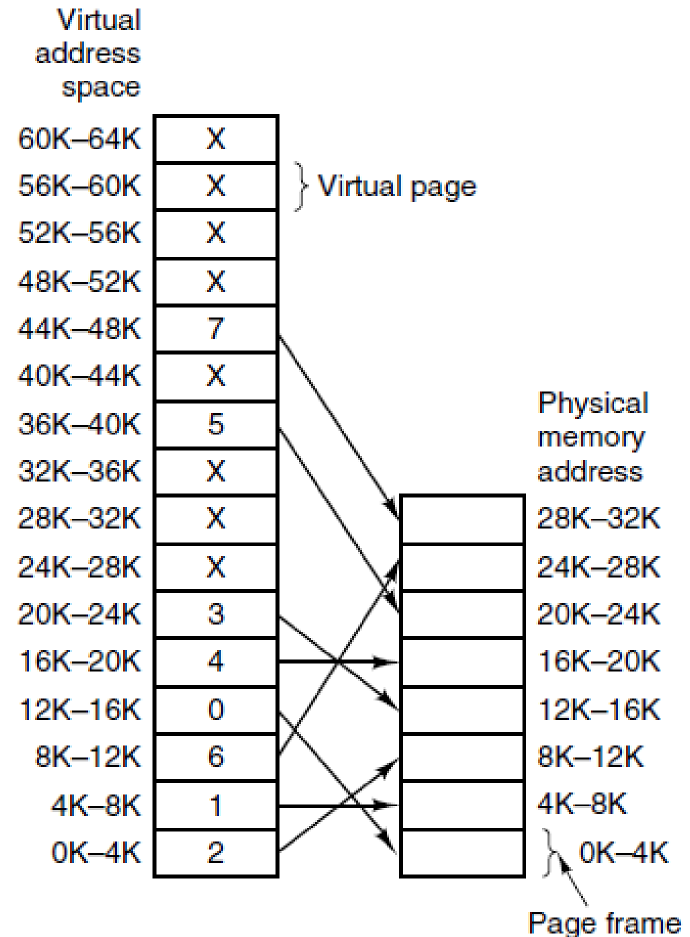
- MMU maintains a map
  - Page size: 4K
- What if
  - MOV REG, 8203
- 8203 is a virtual address, passed to MMU (8K = 8192)
  - determines that 8203 is in page 2 in virtual address space
  - determines that the page is mapped to page frame 6 in physical memory
  - Maps the virtual address to physical address
    - 8203 / 4K = 2 (table lookup → 6)
    - 8203 % 4K + 6 * 4K = 24587



- [Figure 3-9 in Tanenbaum & Bos, 2014]

Example: Page Fault

- MMU maintains a map
  - Page size: 4K
- What if
  - MOV REG, 56930
- Each entry in the map has a present/absent bit
  - 56930 / 4K = 13
  - Absent according to the present/absent bit
  - Causes a page fault, a hardware trap
  - OS catches the trap, select a page frame, write it to the disk, fetch page 13 from disk to the page frame, update the MMU map

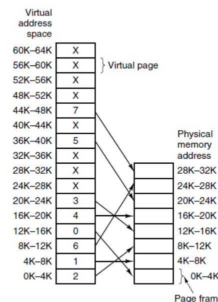


- [Figure 3-9 in Tanenbaum & Bos, 2014]

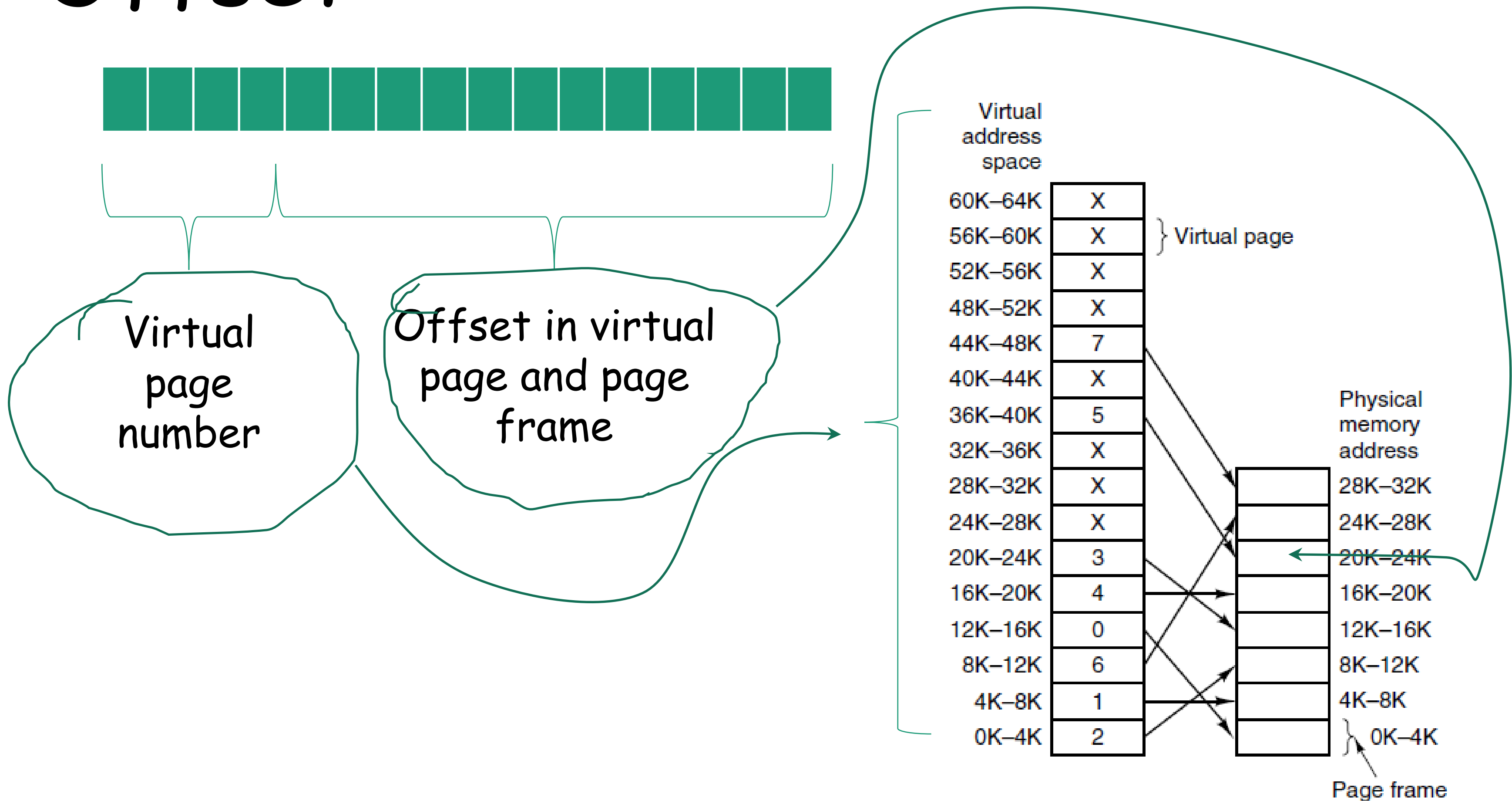
Page Size: Power of 2

- Theoretically, page size can be any size
- In practice, page size is always a power of 2

Virtual page number   Offset in virtual page and page frame

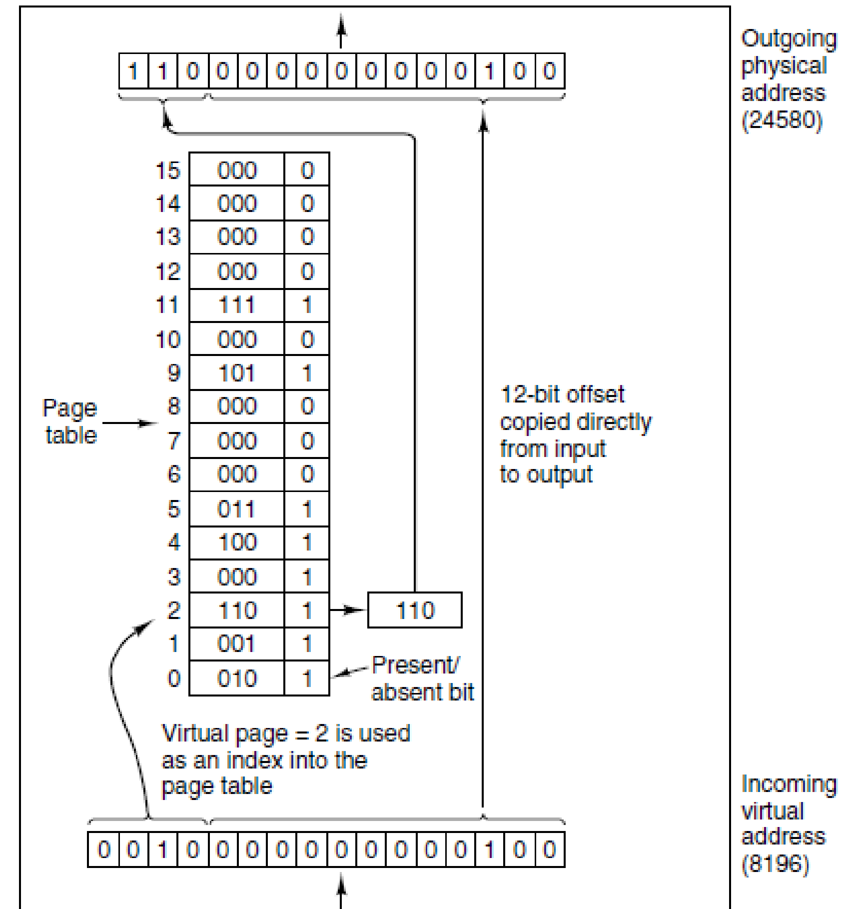


Page Number and Address Offset



Example: Page Table and Power of 2 Page Size

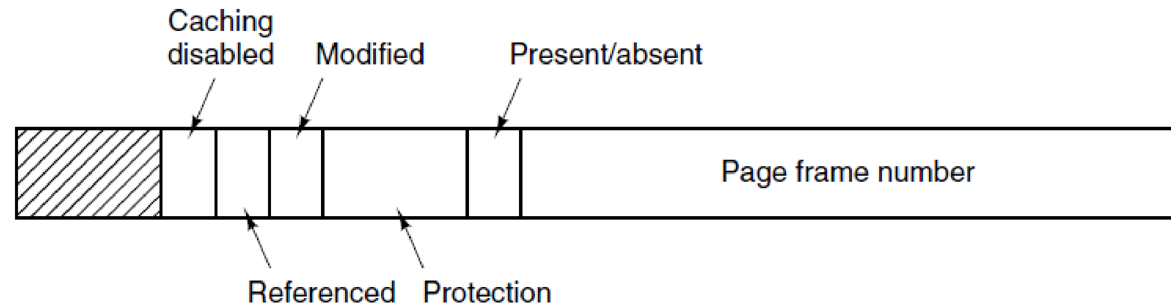
- 16 bit virtual address space
- 4 bit page number
  - 0xf000
- 12 bit offset in page (4096 = 4K)
  - 0x0fff
- Page number = Address & 0xf000
- Page offset = Address & 0x0fff
- Example:
  - Page number: 0b0010 = 2
  - Page offset: 0b0000 0000 0100
  - Page frame number: 0b110 = 6 in the table
  - 6 x 4096 = 0b110 << 12
  - Physical address: (0b110 << 12) ^ 0b … 010



- [Figure 3-10 in Tanenbaum & Bos, 2014]

Design of Page Table

- A data structure an OS must maintain to support virtual memory
- Design page size as power of 2
- Essential fields
  - Page frame number, present/absent, protection (e.g., rwx), modified (is dirty?), referenced (is rwx'ed?), caching disabled (page may be mapped to cache)



- [Figure 3-11 in Tanenbaum & Bos, 2014]

Questions?

- Virtual address
- Virtual pages and page frames
- Concept of page table
- Some design issue of page table
- Group discussion and work
  - Questions 1 and 2

Speeding Up Paging

- Two major issues:
  - The mapping from virtual address to physical address must be fast.
    - Mapping have to done at every memory reference
  - If the virtual address space is large, the page table will be large.
    - 32 bit address pace, 4 KB page, how big is the page table?
    - How many page tables does an OS must maintain?

Translation Lookaside Buffer

- Speeding up paging implementation
  - TLB or associative memory
    - Usually a part of MMU
  - A table consists of a small number of entries
    - Rarely more than 256
    - Each entry is about one page
      - Virtual page number, Is modified (dirty), protection (e.g., rwx) , physical page frame number
      - One-to-one correspondence to the fields in the page table

Example: TLB

- A process may have a TLB as follows,
  - Pages 19 – 21 for process code
  - Pages 129 – 130 for heap, pages 860 – 861 stack, page 140 others data

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

- [Figure 3-12 in Tanenbaum & Bos, 2014]

Example: TLB Functions

1. an address is passed to the MMU for translation
2. MMU compares it to all entries simultaneously (require hardware support)
3. If found, access is permitted, compute physical address directly using the TBL entry
4. If found, but access is not permitted, a protection fault is generated
5. If not found (TLB miss), MMU does an ordinary page table lookup, replace TLB entry with the one from the page table

Design of TLB: Software or Hardware?

- Hardware solution
  - CPU/MMU does TLB management & TLB faults
- Software solution
  - Many systems do page management in software
    - Hardware only generates TLB and page faults
    - OS loads TLB entries explicitly
    - OS fetches page table entry and replace missing TLB entry
  - Experimental evidences
    - When TLB is moderately large, TLB miss rate is acceptable
    - Hardware spaces can be used to have more cache

Design of TLB: TLB Miss

- Page tables is a data structure that OS maintains
- When a TLB miss occurs, where is the page table entry to be fetched?
  - It could be on a page that isn't in the page frames!
  - The page's entry isn't in the TLB
  - Another TLB miss

TLB Miss and Page Fault

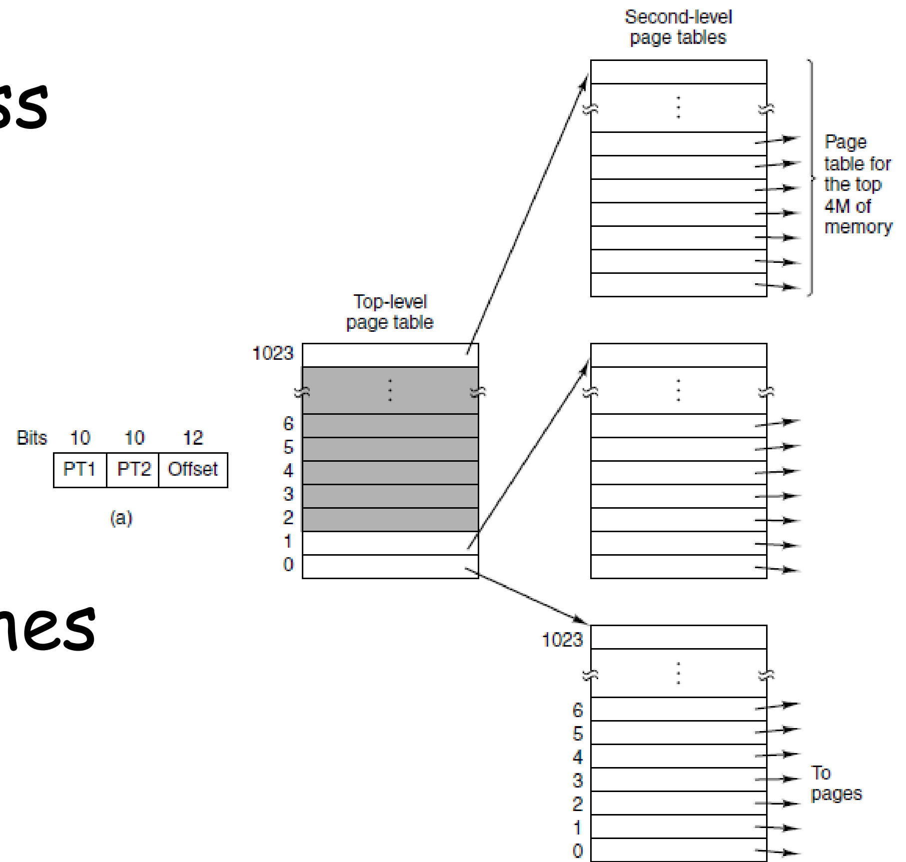
- TLB miss
  - Soft miss
    - The page referenced is not in the TLB, but in the memory; No disk access required
  - Hard miss
    - The page referenced is not in the memory at all; Require disk access
- Page fault
  - Minor page fault
    - Page loaded by other process (indexed in other process's table table); No disk access
  - Major page fault
    - Page not in the memory; Require disk access
  - Accessed invalid address, segmentation fault

Page Tables for Large Memories

- Multi-level page tables
- Inverted page tables

Multilevel Page Tables: Example

- 32-bit virtual address
- 10-bit PT1
- 10-bit PT2
- 12-bit offset
- Only selected branches are in memory



- [Figure 3-13 in Tanenbaum & Bos, 2014]

Multilevel Page Tables: Example

- Intel x64 uses 4-levels
  - PT1: 9 bits
  - PT2: 9 bits
  - PT3: 9 bits
  - PT4: 9 bits
  - Page size: 12 bits
  - Virtual address space: $2^{(9+9+9+9+12)} = 2^{48}$
  - How an OS takes advantage of this differ
    - See Memory Limits for Windows and Windows Server Releases

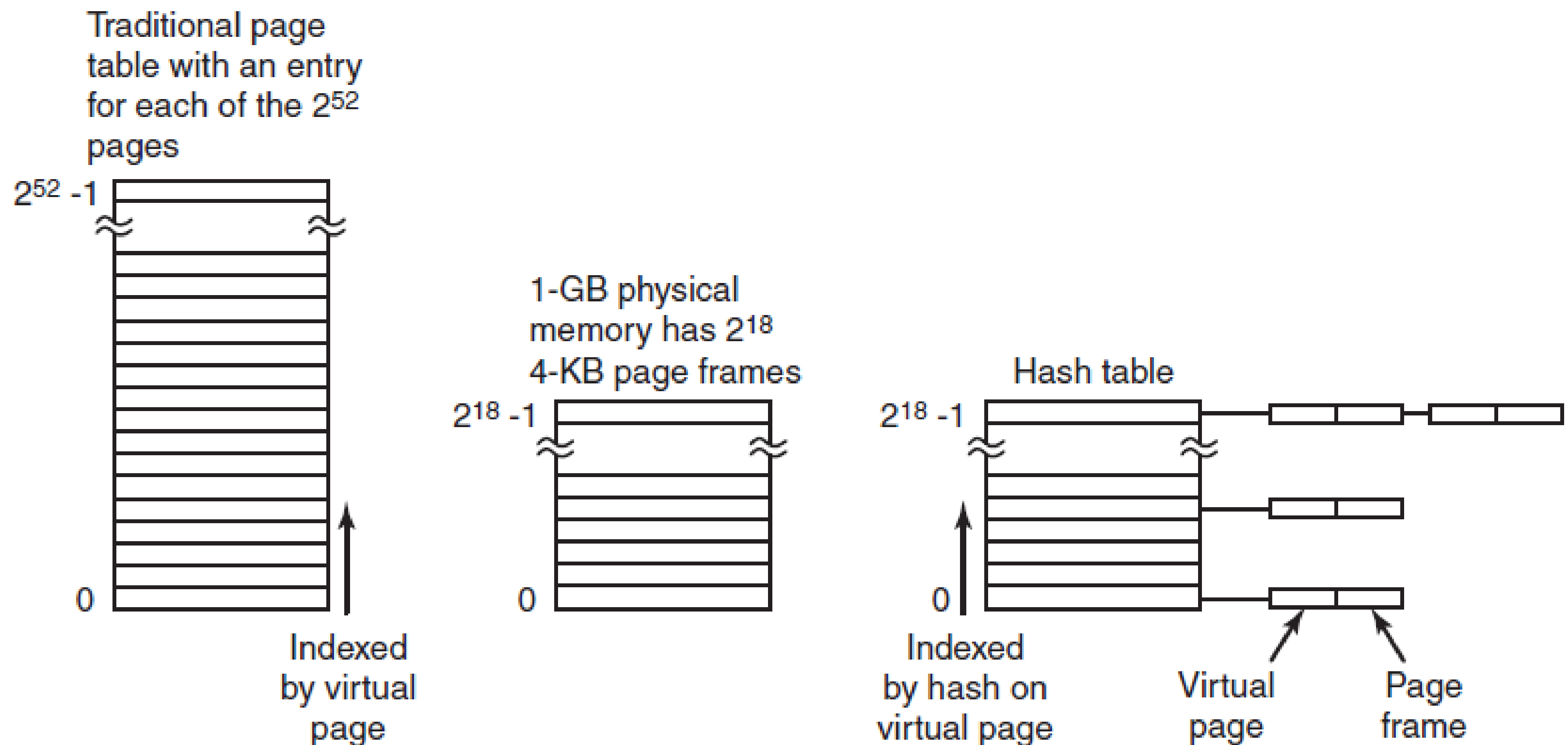
Inverted Page Tables

- In previous discussion, there is one page table entry for every virtual page
- In an inverted page table, there is on entry per page frame in real memory.
  - Example:
    - a 64-bit virtual address
    - 4-KB page size
    - 4 GB of RAM
    - An inverted page table has table entries
      - 4 GB / 4KB = 2^{20} page table entries
- How to do virtual-to-physical address translation?

Inverted Page Table: Address Translation

- Must search the inverted page table to translate a virtual address to a physical address
- Take advantage of TLB
  - A search of the inverted table required only when a TLB miss
- How to make the search more efficient?
  - Use additional data structure
    - Hash table with virtual address as key

Inverted Page Tables: Example Design



- [Figure 3-14 in Tanenbaum & Bos, 2014]

Questions

- Speeding-up page table via TLB
- Issues in TLB design
- Design of large virtual address spaces
  - Multi-level page tables
  - Inverted page tables

Page Replacement

- Page faults occurs
  - Reference to a virtual page that is not mapped to a page frame
- Handling page faults
  - Select and evict a page frame from memory by saving the page to disk and update the page table
  - Fetch the virtual page from the disk
  - Replace the memory space occupied by the evicted page frame, update the page table
- Question
  - Which page frame evict?
  - An algorithm that selects page frame to evict is called a page replacement algorithm

The Optimal Page Replace Algorithm

- If we know past and future, we can select page to evict to reduce page faults to minimum.
- Used as a reference to evaluate page replace algorithms
- Question?
  - Do we allow evict other processes’ page frame or just our own?
  - If we allow evict other processes’ page frame, is it always fair to minimize page faults? (System-wide page faults may be minimized, but how about an individual process?)

Page Replacement Algorithms

- Optimal algorithm
- Not-recently-used algorithm
- First-in, first-out (FIFO) algorithm
- Second-chance algorithm
- Clock algorithm
- Least recently used (LRU) algorithm
- Working set algorithm
- WSClock algorithm

Referenced and Modified Bits

- Page table entries contain
  - Reference bit: R bit (0 not referenced; 1 otherwise)
  - Modified bit: M bit (0 not modified; 1 otherwise)
- OS maintains the R and M bits
  - When a process is being created, R & M bits of all pages are set 0
  - Periodically (e.g., on each clock interrupt), the R bit is cleared
  - When a page is rewritten to the disk, the M bit is cleared
  - When a page is being referenced, the R bit is set.
  - When a page is being modified, the M bit is set

Not-Recently-Used Algorithm

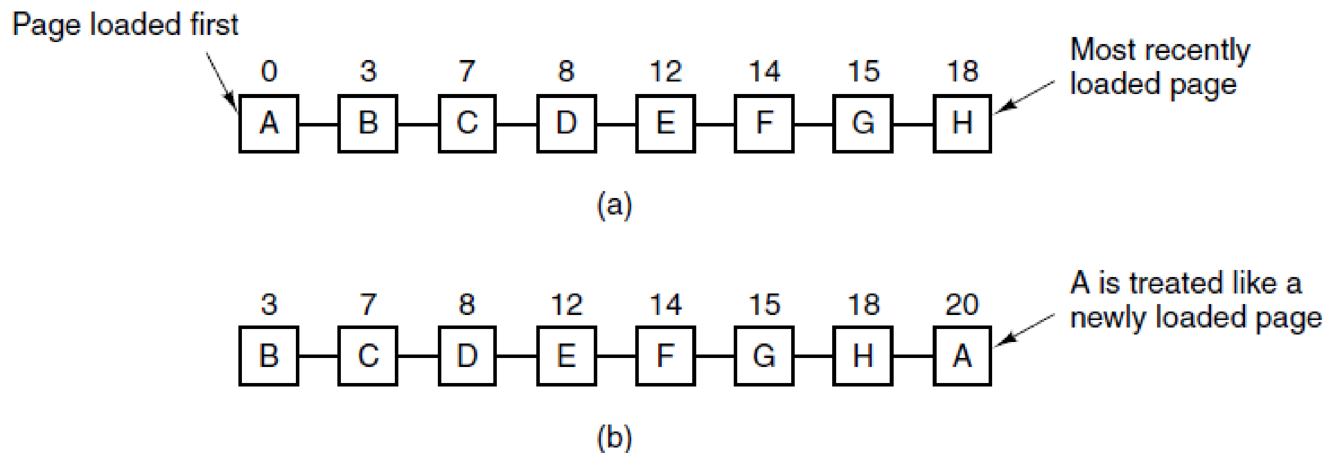
- At page fault, system inspects pages
- Categories of pages based on the current values of their R and M bits:
  - Class 0: not referenced, not modified. (when R bit of page in case 3 is cleared at a clock interrupt)
  - Class 1: not referenced, modified.
  - Class 2: referenced, not modified.
  - Class 3: referenced, modified.
- NRU selects at random a page in the lowest non-empty class and replaces the page by the newly fetched one.

First-in, First-Out (FIFO) Algorithm

- An OS organizes the page frames in a list
  - Most recently fetched at the tail (least recently fetched at the head)
  - On a page fault, evict and replace the page at the head

Second-Chance Algorithm

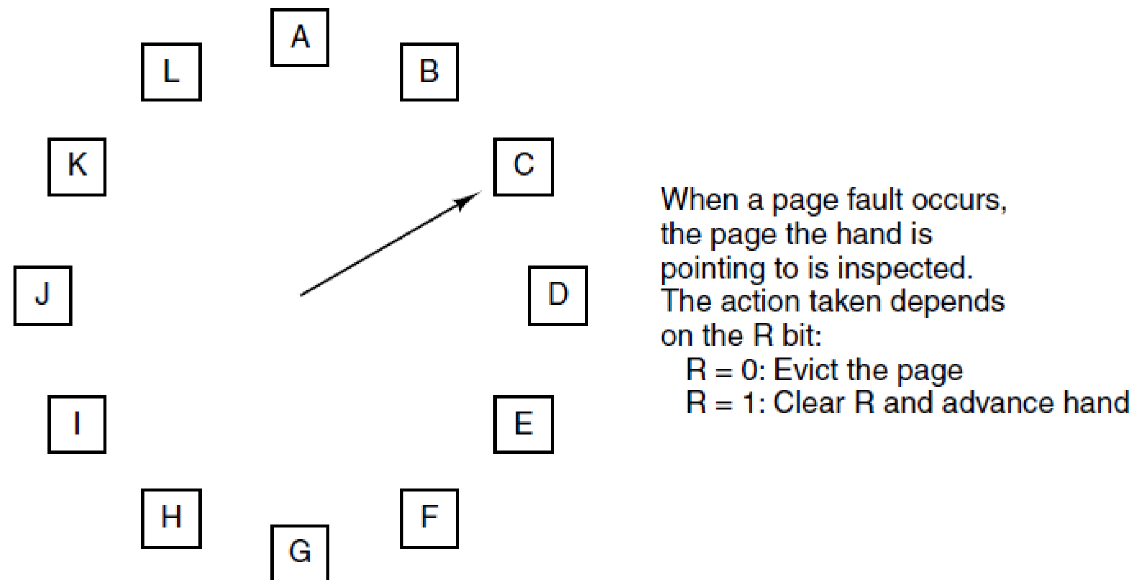
- A simple modification to the FIFO algorithm to avoid throwing out a heavily used old page
  - Inspect the R bit of the oldest page (at the head of the list). If 0, replace the page; if 1, clear the R bit and move the page to tail of the list.



- [Figure 3-15 in Tanenbaum & Bos, 2014]

Clock-Page Replacement Algorithm

- To avoid moving pages in the list, replace the list by a circular list, i.e., maintains a pointer to the oldest page.



- [Figure 3-16 in Tanenbaum & Bos, 2014]

Least Recently Used (LRU) Algorithm

- Intuition: a heavily used page in the last few instructions will probably be heavily used again soon.
- At a page fault, evict and place the page that has not been used for the longest time
- Need an additional list to record the order of reference
- Require to update the list at every reference
- Contrast to the other algorithms where the list only need to be updated at every fetch

LRU Implementation: Hardware Approach

- Design consideration: the counter approach
  - Hardware maintains a long (e.g., 64-bit) counter;
  - Page table entry also has the copy of the counter
  - Increment the counter at each reference, update replaced page with the counter
  - At page fault, search the page with the lowest counter (need additional data structure to support efficient search)

LRU Implementation: Software

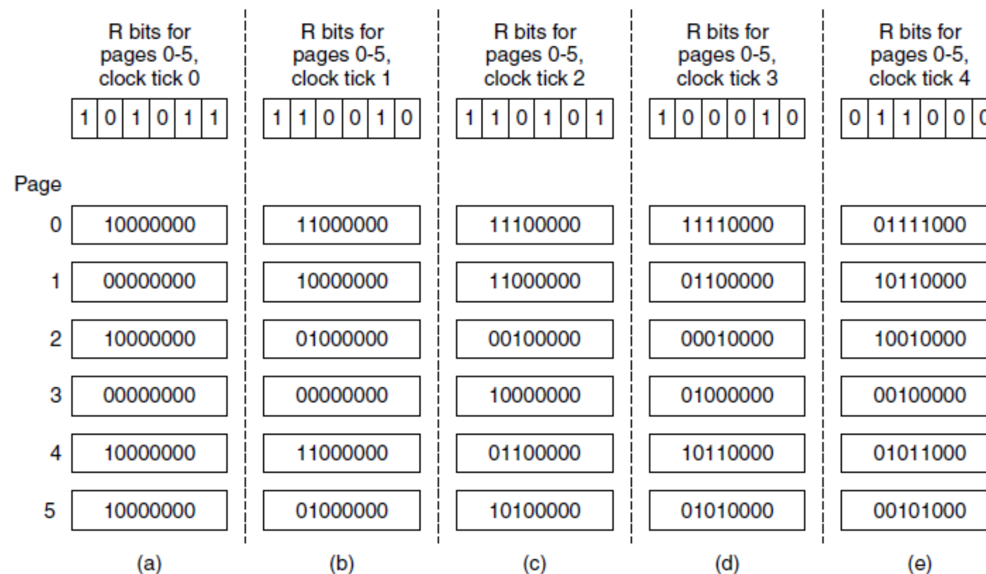
- A software counter associated with each page
- At each clock interrupt, scan each page, and increment the page's counter if R bit is set
- At page fault, select the page with the lowest count for replacement
- Called Least-frequently-used (LFU or NFU)
- Problem
  - LFU does not forget.

LRU Implementation: Aging

- How to forget "ancient" history? There are many strategies.
- Since memory reference must be resolved very quickly, it must be simple

LRU Implementation: Aging: A Simple Algorithm

- Using shift counter
  - Shift counter 1 bit to the right
  - Add the value of R bit to the left



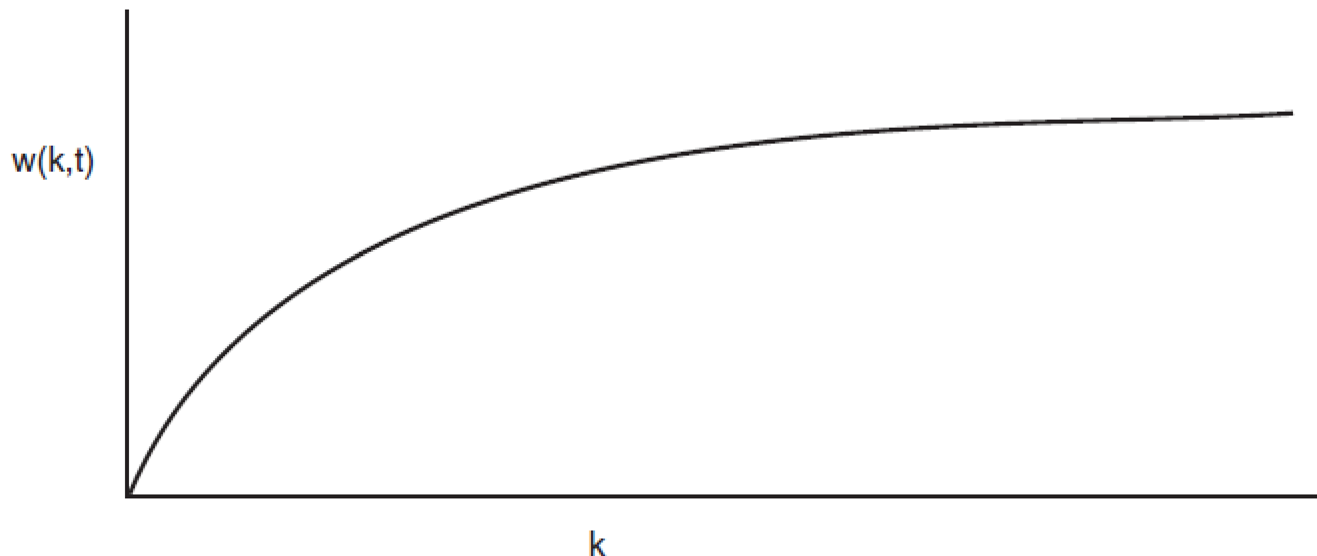
- [Figure 3-16 in Tanenbaum & Bos, 2014]

Paging Implementation Models

- Demand paging model
  - Processes are started up with none of their pages in memory
  - Page faults, as CPU starts executing a process instruction
  - The OS fetches the page
- Working set model
  - During any phase of execution, a process references only a relatively small fraction of its pages
  - Working set: the set of pages that a process is currently using
  - Keep track of working set, and make sure that it is in memory before letting the process run
  - Pre-paging: loading the pages before letting processes run

Locality of Reference

- Programs rarely reference their pages uniformly, tend to cluster on a small number of pages
- The set of pages used in the k most recent memory references



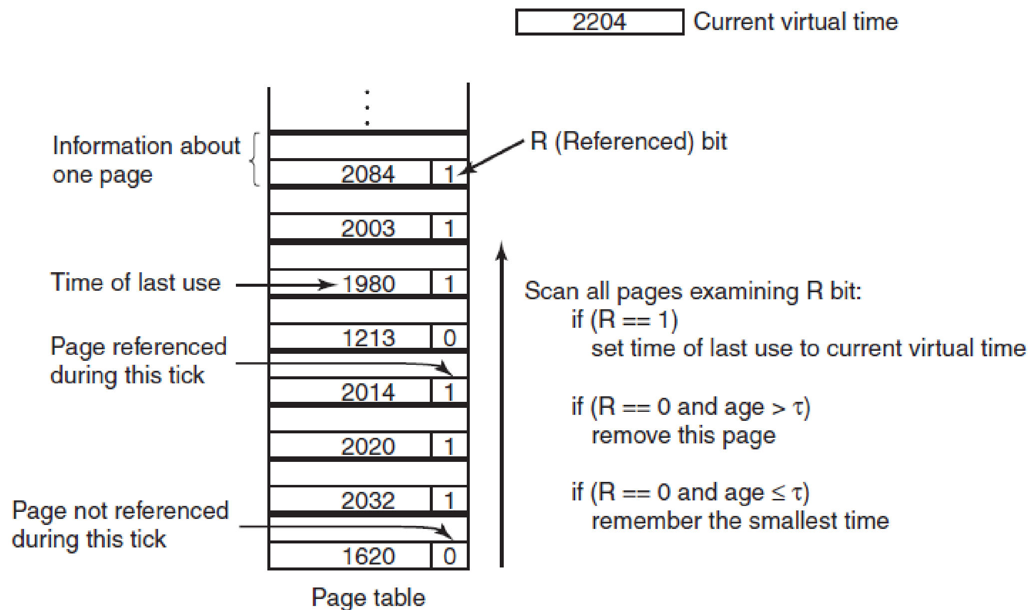
- [Figure 3-18 in Tanenbaum & Bos, 2014]

Working Set Algorithm

- Maintain a working set; upon a page fault, evict one that is not in the working set.
- Need to determine k beforehand
- Need efficient implementation

Working Set Algorithm: Example Implementation

- Current virtual time: CPU time a process actually used
- At a clock interrupt, scan pages. If R bit set, the current virtual time is written into the Time of last use field



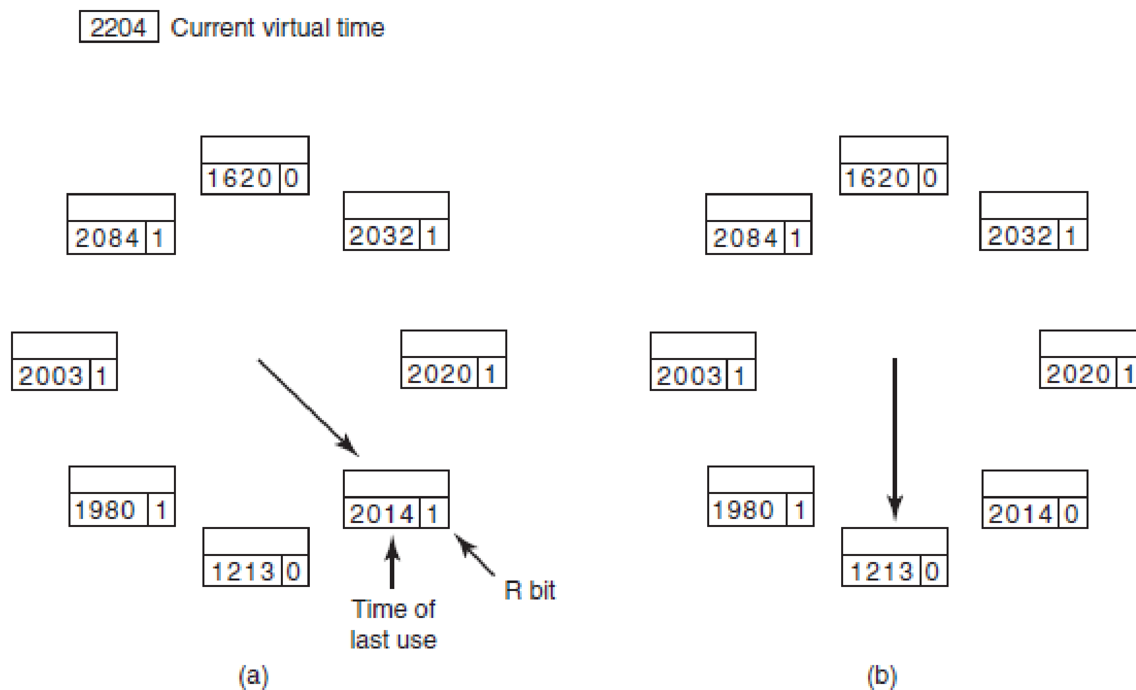
- [Figure 3-18 in Tanenbaum & Bos, 2014]

WSClock Page Algorithm

- Scan the entire page table is an expensive operation
- Based on both the clock algorithm and the working set algorithm
- Maintains a circular list of page frames as in the clock algorithm
- Each page has the Time of last use field

WSClock Algorithm: Scenarios

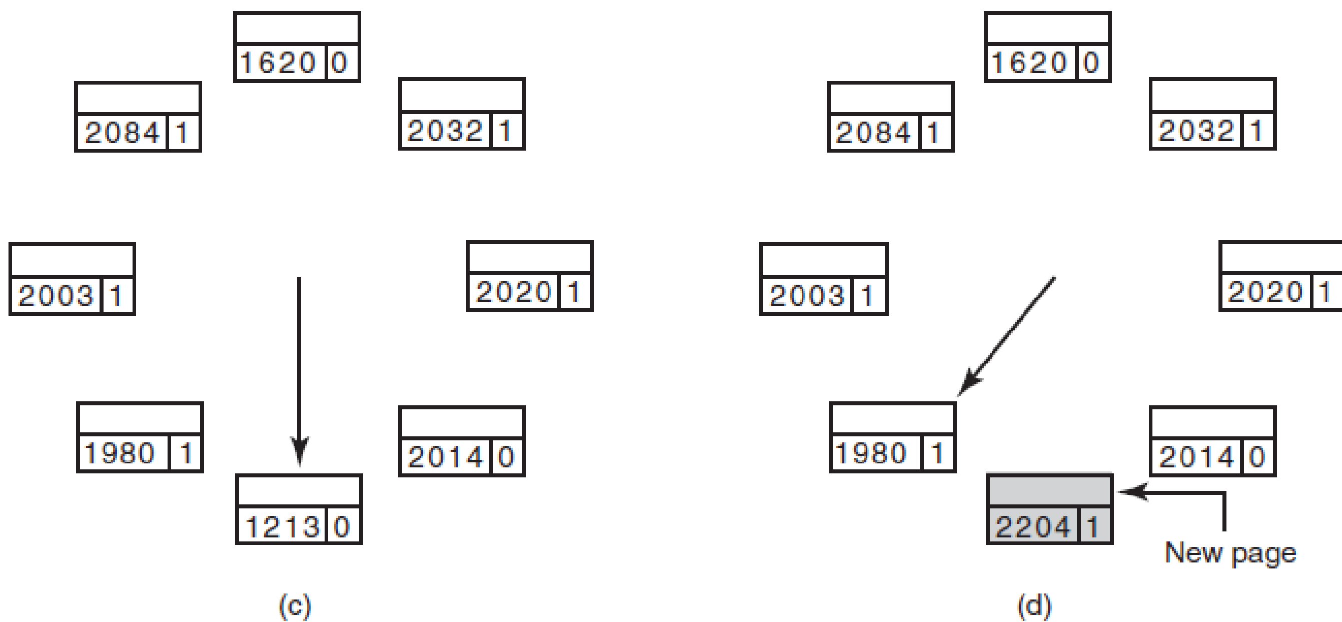
- If R bit is set, clear the bit, and advance the pointer



- [Figure 3-20 in Tanenbaum & Bos, 2014]

WSClock Algorithm: Scenarios

- If R bit not set, if old page and M bit not set, replace it ; otherwise, advance the pointer



- [Figure 3-20 in Tanenbaum & Bos, 2014]

Summary of Page Replacement Algorithms

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude approximation of LRU
FIFO (First-In, First-Out)	Might throw out important pages
Second, chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

Questions?

- Various page replacement algorithms

Design Issues in Paging Systems

- Allocation Policies
- Load control
- Page size
- Separate instruction and data spaces
- Shared pages
- Shared libraries
- Cleaning policy
- Virtual memory interface

Implementation Issues

- OS involvement with paging
  - Process creation, process execution, page fault, and process termination
- Page fault handling
- Instruction backup
- Locking pages in memory
- Backing store
- Policy and mechanism separation

Page Fault

- Oops!

Page Fault Handling

1. The hardware traps to kernel, saving program counter on stack.
2. Assembly code routine started to save general registers and other volatile info
3. system discovers page fault has occurred, tries to discover which virtual page needed
4. Once virtual address caused fault is known, system checks to see if address valid and the protection consistent with access

Page Fault Handling

5. If frame selected dirty, page is scheduled for transfer to disk, context switch takes place, suspending faulting process
6. As soon as frame clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in.
7. When disk interrupt indicates page has arrived, tables updated to reflect position, and frame marked as being in normal state.

Page Fault Handling

8. Faulting instruction backed up to state it had when it began and program counter is reset
9. Faulting process is scheduled, operating system returns to routine that called it.
10. Routine reloads registers and other state information, returns to user space to continue execution

Questions

- A quick overview of a design and implementation issues in paging systems

Assignment

- Practice assignment