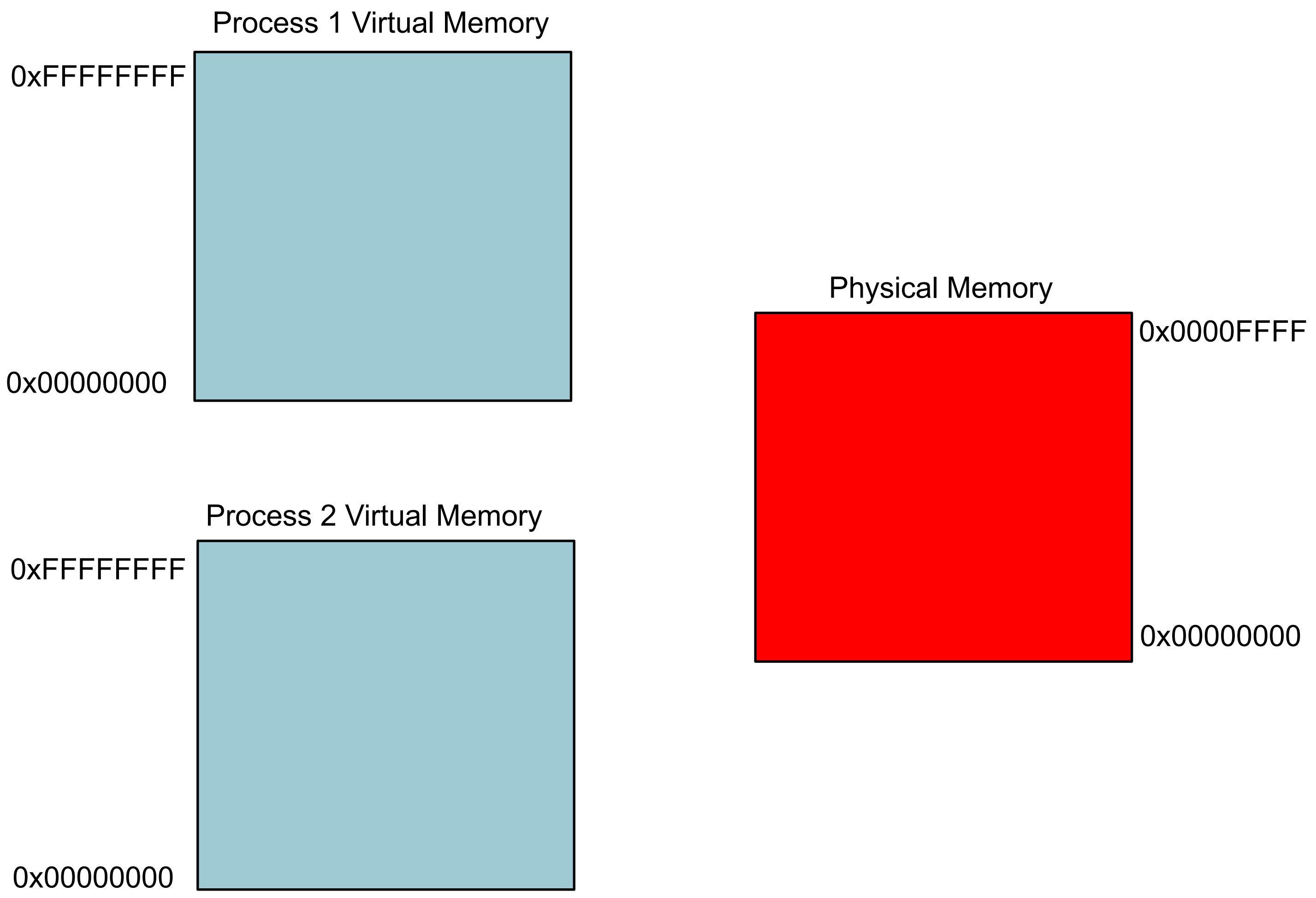
Chapter 5 – Part 2

Large and Fast Memory: Exploiting Memory Hierarchy

Virtual Memory

- There is a need to run programs that are too large to fit in memory
- Solution adopted in the 1960s, split programs into little pieces, called overlays (pages)
  - Kept on the **disk**, swapped in and out of **main memory**
- Virtual memory : each program has its own address space, broken up into chunks called pages
  - Each page is a continuous range of addresses that are mapped onto physical memory

Virtual Memory

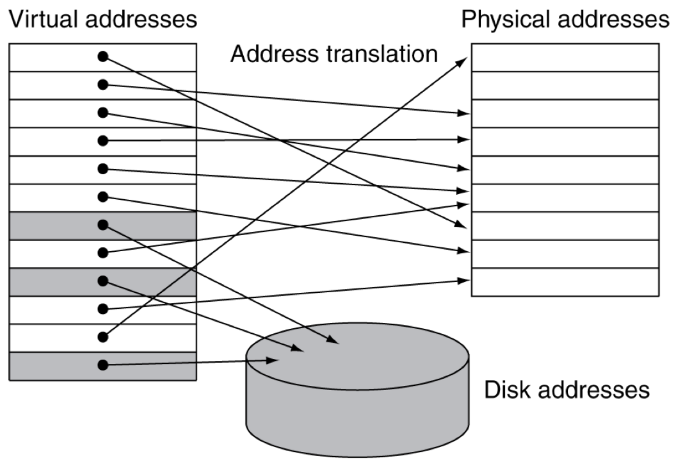


Virtual Memory

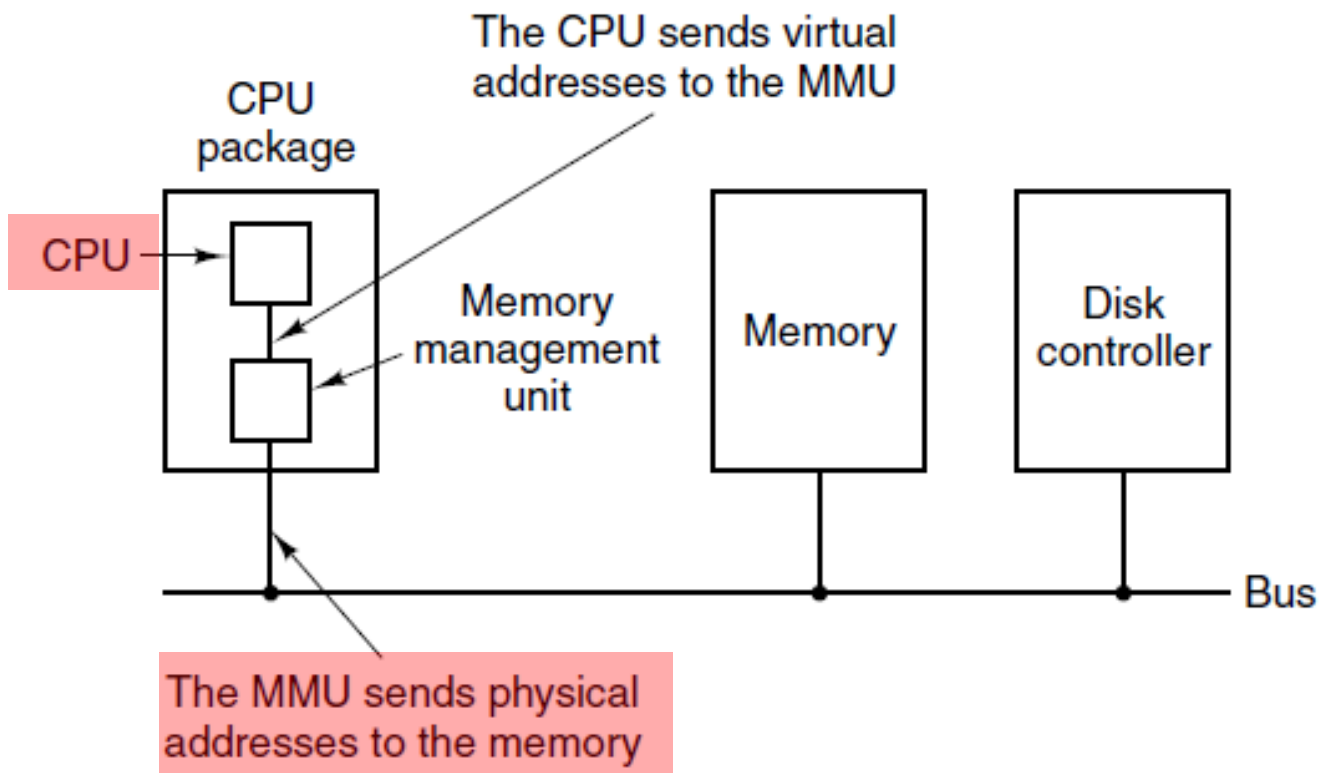
- Use main memory as a "cache" for secondary (disk) storage
- Programs share main memory
- CPU and OS translate virtual addresses to physical addresses
  - VM "block" is called a page

Virtual Memory

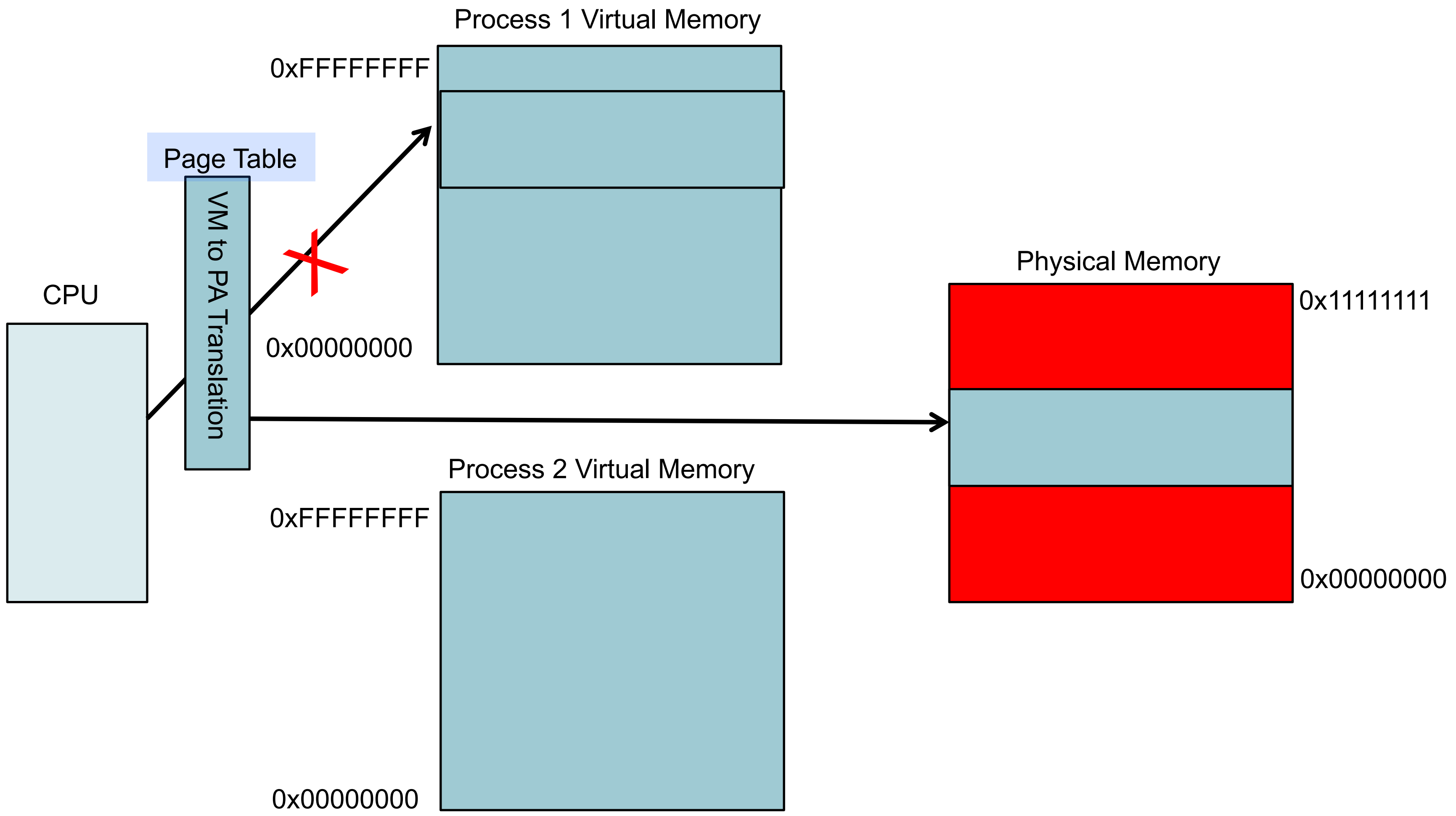
- VM adds a level of indirection between the virtual program addresses (VA) and the physical RAM addresses (PA)
- But there is a bit of a price (performance) to pay for
  - VA to PA translation (some bookkeeping required which impacts the performance)
- Three ways to achieve this VA to PA translation:
  1. VA to PA translation on the fly using a Memory Management Unit (MMU)
  2. Page tables
  3. Fast translation via TLB



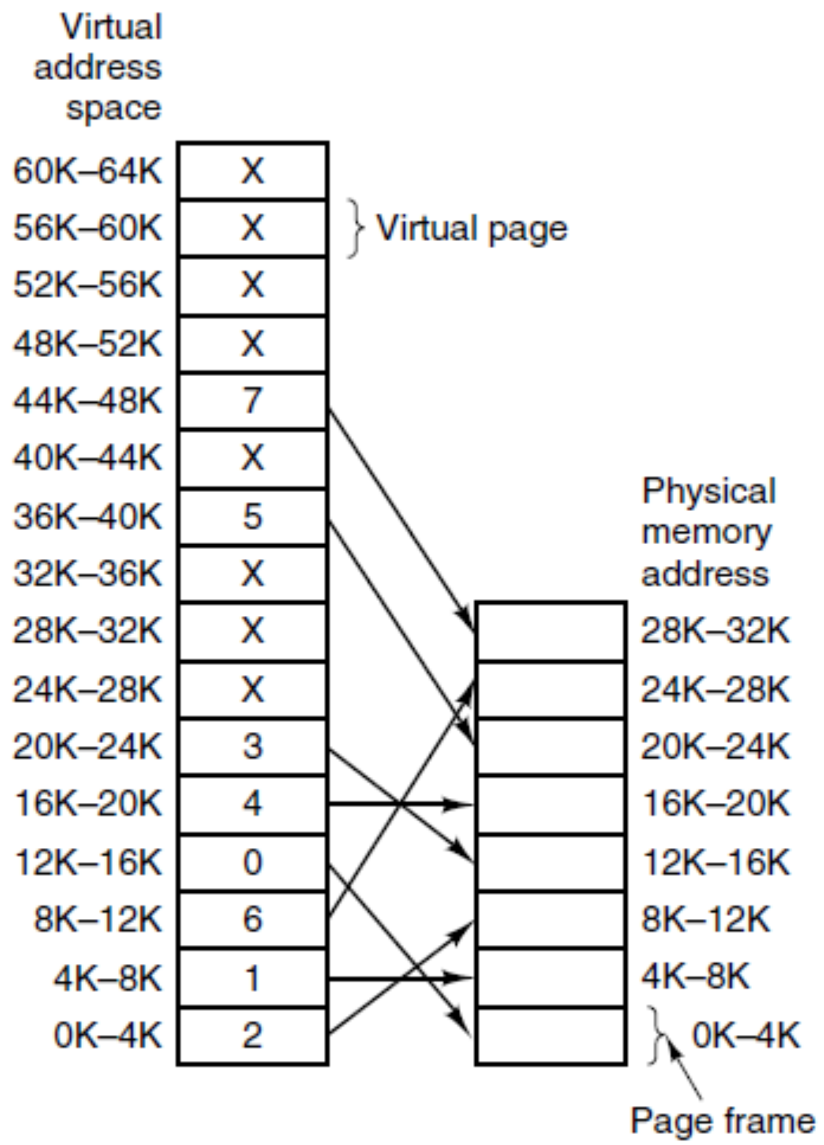
Paging


The CPU sends virtual addresses to the MMU
CPU package
CPU
Memory management unit
Memory
Disk controller
Bus
The MMU sends physical addresses to the memory

Paging

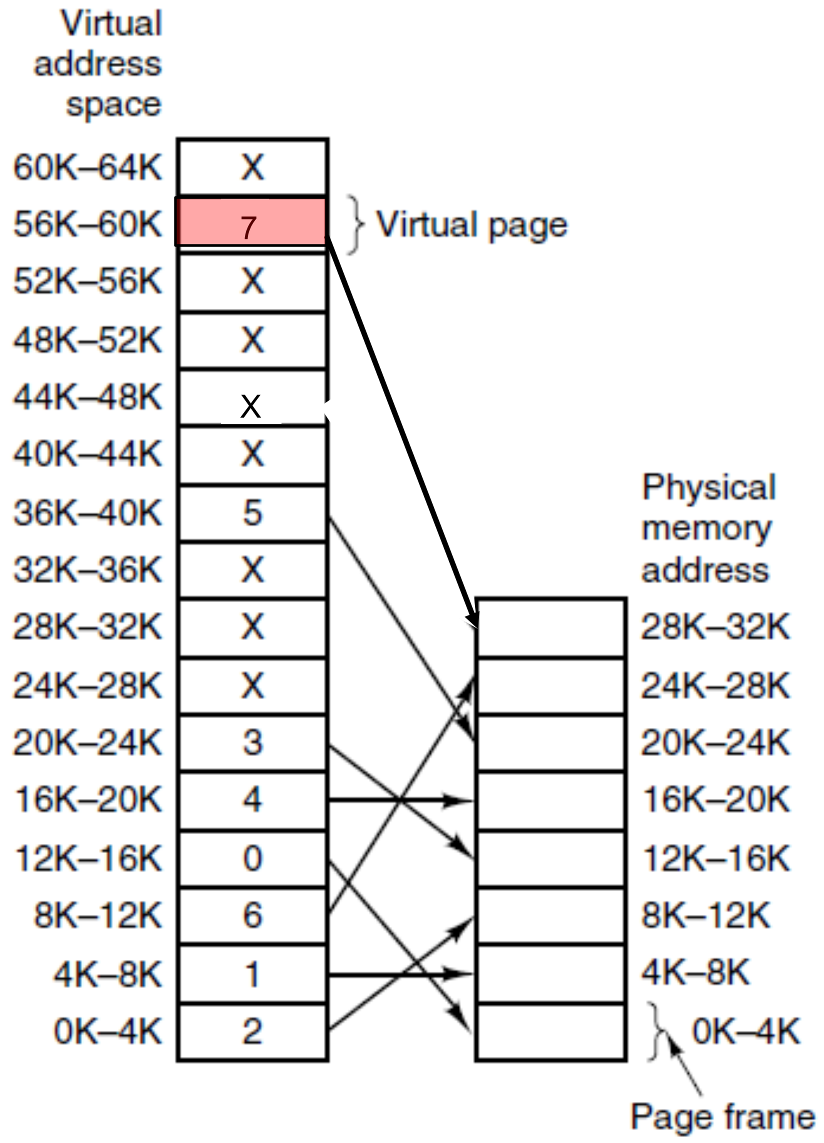

Process 1 Virtual Memory
0xFFFFFFFF
0x00000000
Page Table
VM to PA Translation
CPU
Physical Memory
0x11111111
0x00000000
Process 2 Virtual Memory
0xFFFFFFFF
0x00000000

Paging



Virtual address space
60K–64K X
56K–60K X
52K–56K X
48K–52K X
44K–48K 7
40K–44K X
36K–40K 5
32K–36K X
28K–32K X
24K–28K X
20K–24K 3
16K–20K 4
12K–16K 0
8K–12K 6
4K–8K 1
0K–4K 2
Virtual page
Physical memory address
28K–32K
24K–28K
20K–24K
16K–20K
12K–16K
8K–12K
4K–8K
0K–4K
Page frame

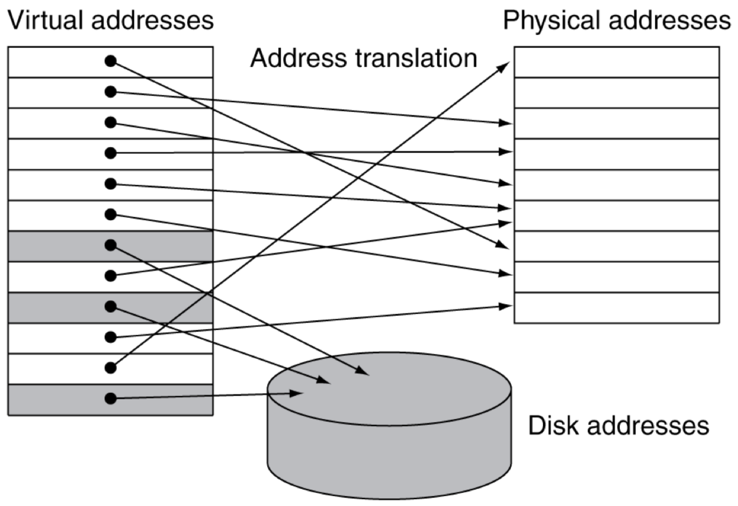
Paging

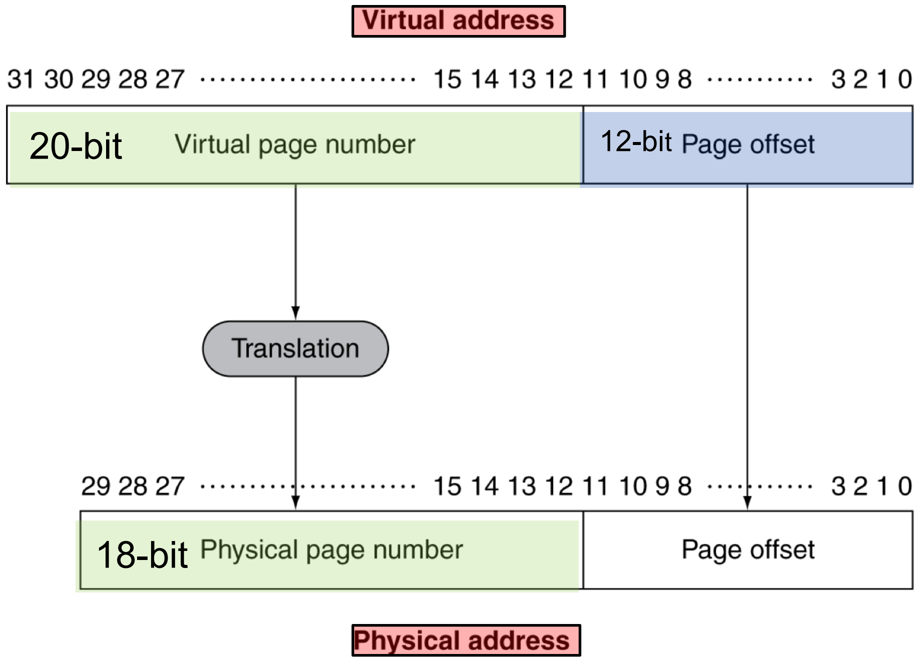


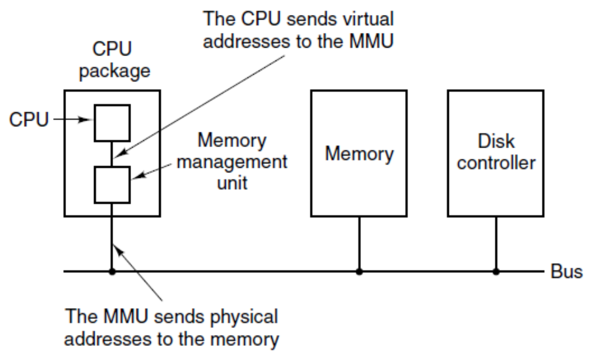
Virtual address space
60K–64K X
56K–60K 7
52K–56K X
48K–52K X
44K–48K X
40K–44K X
36K–40K 5
32K–36K X
28K–32K X
24K–28K X
20K–24K 3
16K–20K 4
12K–16K 0
8K–12K 6
4K–8K 1
0K–4K 2
Virtual page
Physical memory address
28K–32K
24K–28K
20K–24K
16K–20K
12K–16K
8K–12K
4K–8K
0K–4K
Page frame

Address Translation

- Fixed-size pages (e.g., 4K)






CPU package
The CPU sends virtual addresses to the MMU
CPU
Memory management unit
Memory
Disk controller
Bus
The MMU sends physical addresses to the memory

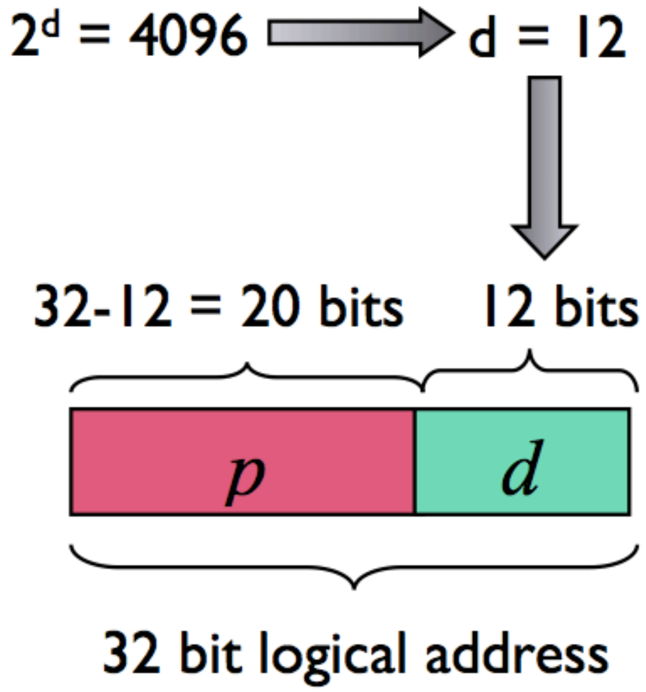
Mapping Virtual to Physical Address

4KB-page system

32-bit word addressing

- Split address from CPU into two pieces
  - Page number (*p*)
  - Page offset (*d*)
- Page number
  - Index into page table
  - Page table contains base address of page in physical memory
- Page offset
  - Added to base address to get actual physical memory address
- Page size = 2^d bytes

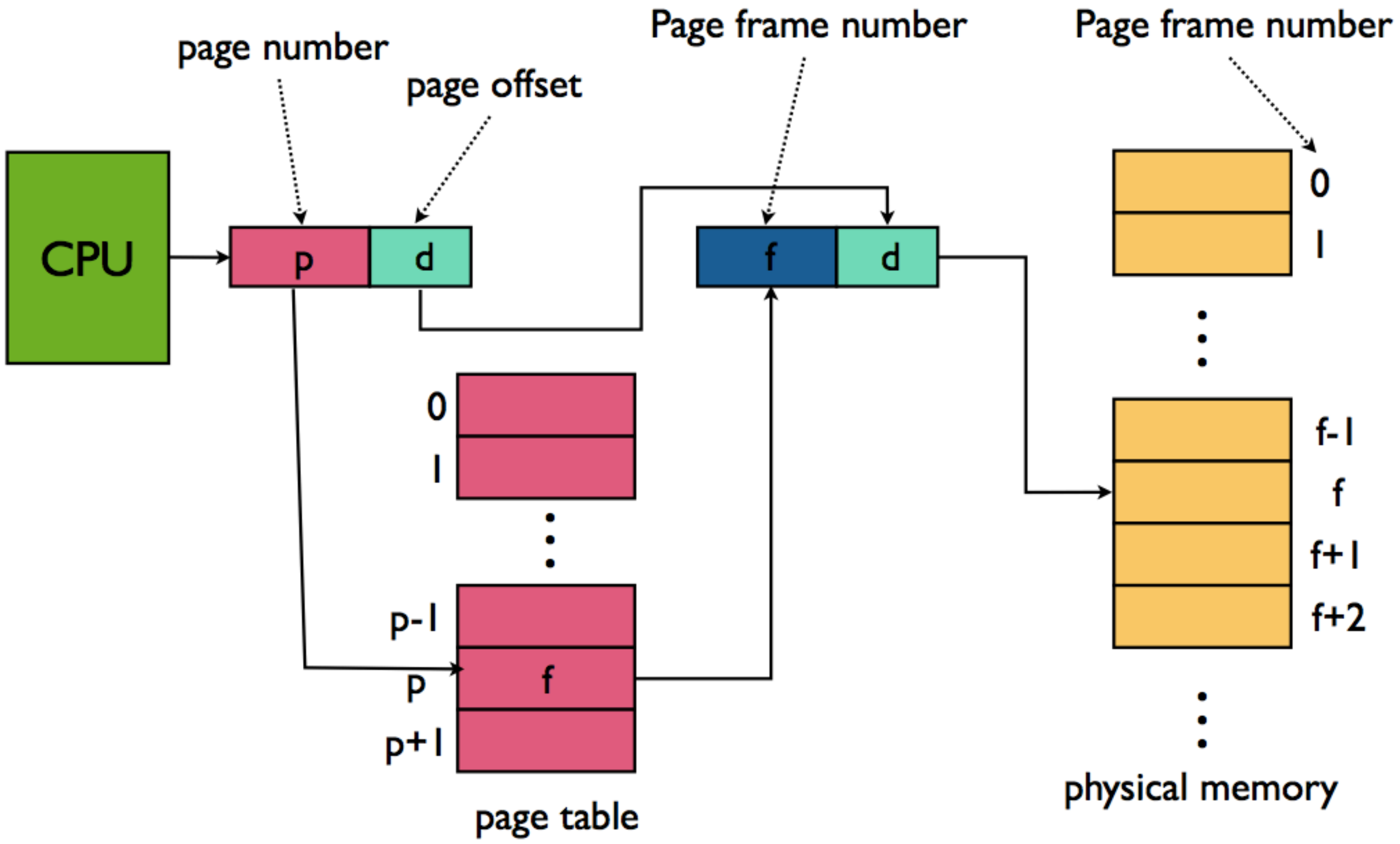
Example:
- 4 KB (=4096 byte) pages
- 32 bit logical addresses



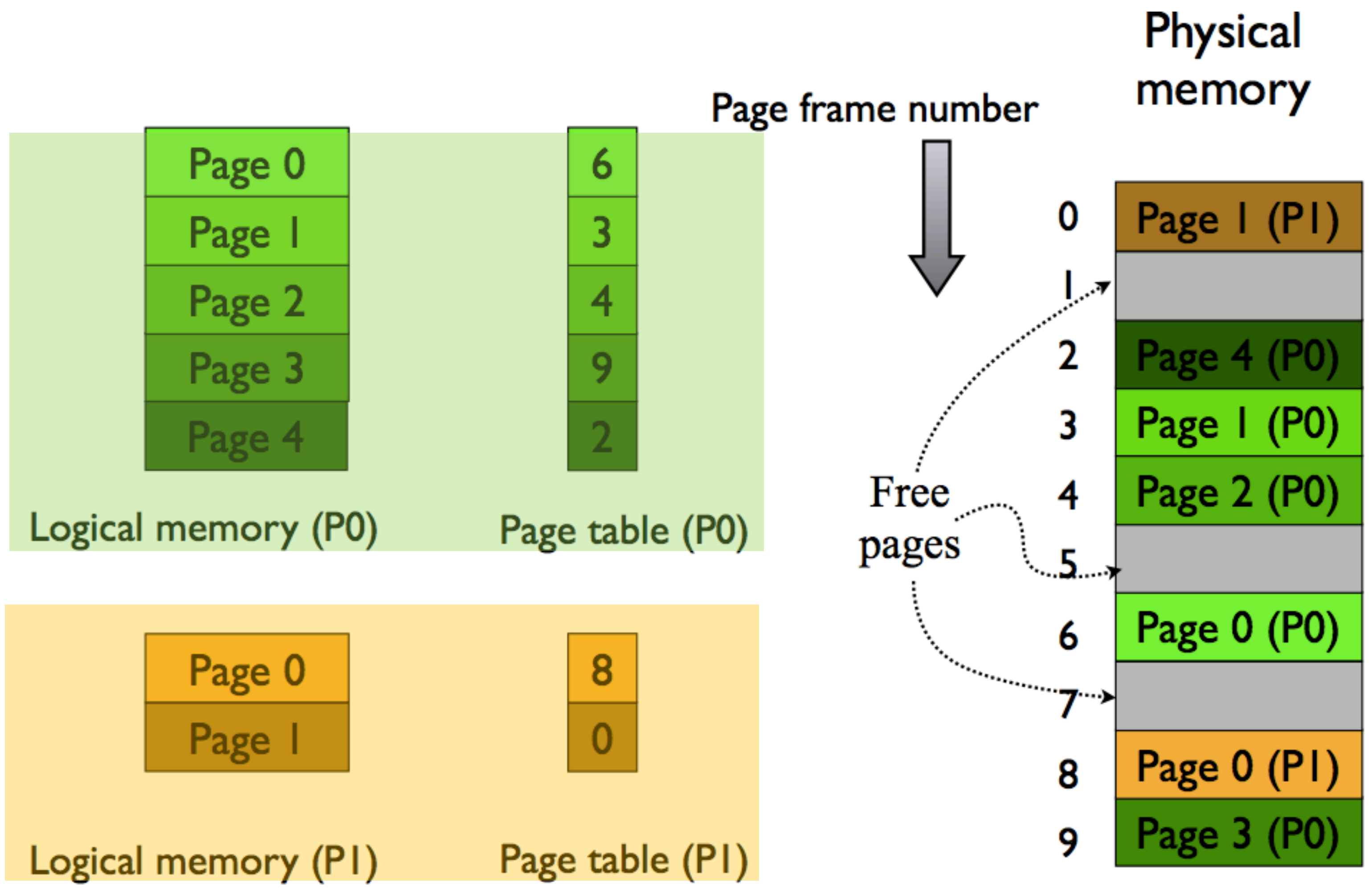
Page Tables

- Stores placement information
- If page is present in memory
- If page is not present (page fault)

Translation Using a Page Table



Multiple Processes in Memory



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

Optimal Algorithm

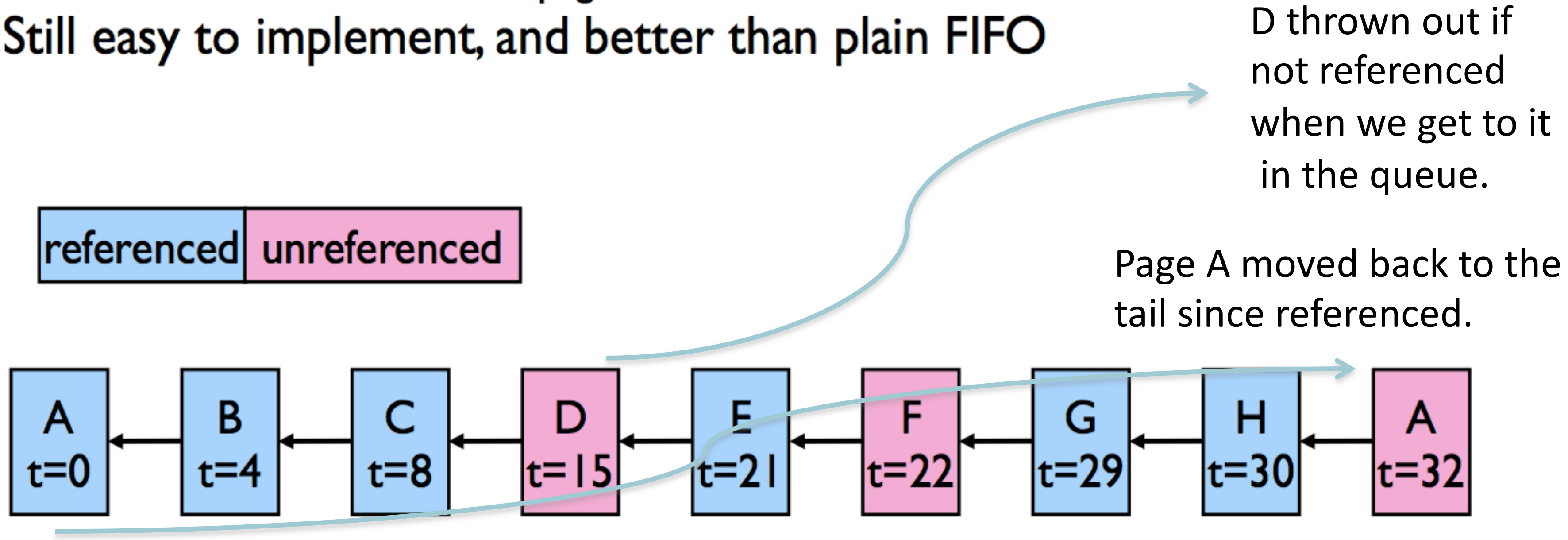
- Best possible but almost impossible to implement
- Almost impossible because the OS has no knowledge on when each page will be referenced next
- Possible solution
  - Run the program on simulator for a specific input data and keep track of those numbers
  - Implement optimal page replacement on the second run based on the info obtained the first time

First-in, First-out (FIFO) Algorithm

- Maintain a linked list of all pages
  - Maintain the order in which they entered memory
- Page at front of list replaced
- Advantage: (really) easy to implement
- Disadvantage: page in memory the longest may be often used
  - This algorithm forces pages out regardless of usage
  - Usage may be helpful in determining which pages to keep

Second-Chance Algorithm

- Modify FIFO to avoid throwing out heavily used pages
  - If reference bit is 0, throw the page out
  - If reference bit is 1
    - Reset the reference bit to 0
    - Move page to the tail of the list
    - Continue search for a free page
- Still easy to implement, and better than plain FIFO


D thrown out if not referenced when we get to it in the queue.
referenced
unreferenced
Page A moved back to the tail since referenced.
A t=0
B t=4
C t=8
D t=15
E t=21
F t=22
G t=29
H t=30
A t=32

Not Recently Used Algorithm

- At page fault, the OS inspects pages and categorizes them based on the current values of their R *(reference)* and *M* (dirty) bits:

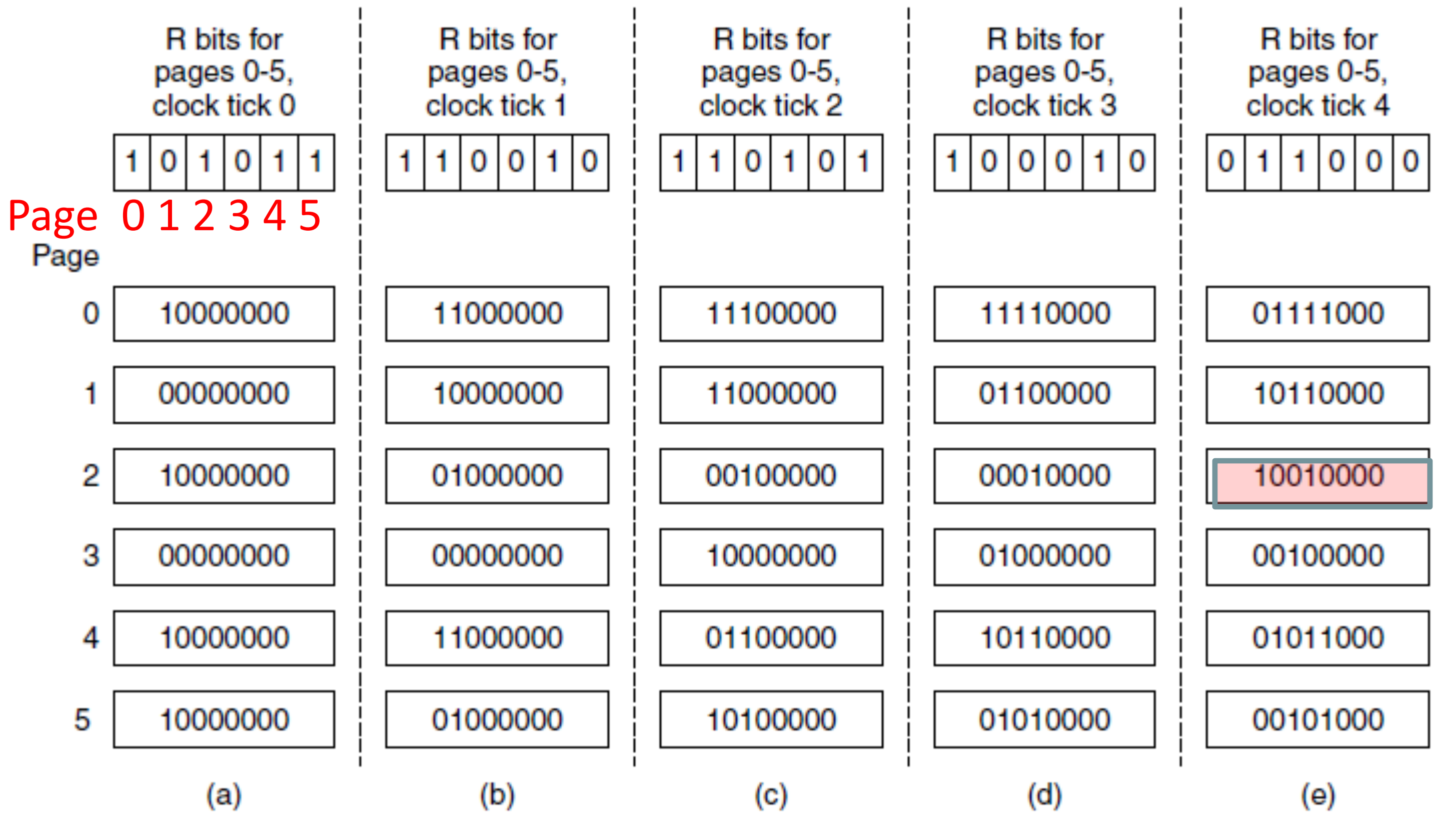
  Class 0: not referenced, not modified.

  Class 1: not referenced, modified.

  Class 2: referenced, not modified.

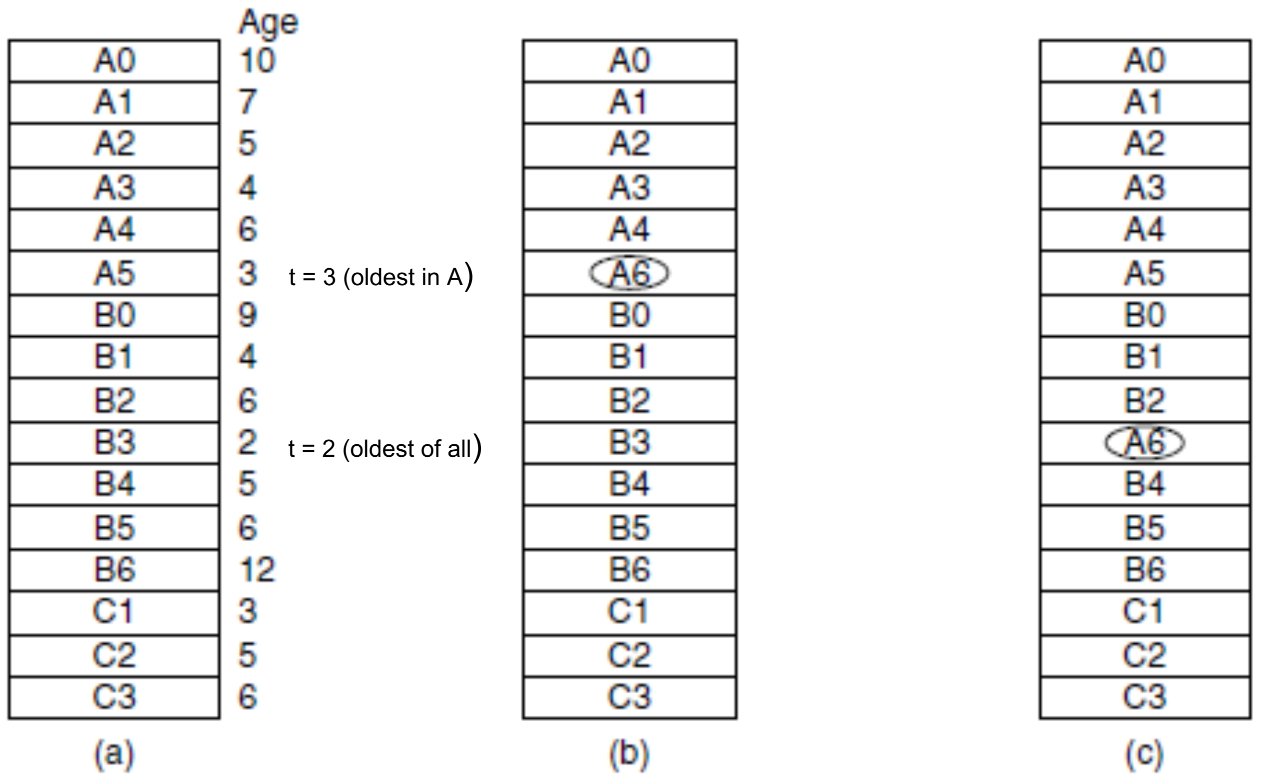
  Class 3: referenced, modified.

Simulating LRU in Software



	R bits for pages 0-5, clock tick 0	R bits for pages 0-5, clock tick 1	R bits for pages 0-5, clock tick 2	R bits for pages 0-5, clock tick 3	R bits for pages 0-5, clock tick 4
	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
Page 0 1 2 3 4 5					
Page					
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00010000	10010000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000
	(a)	(b)	(c)	(d)	(e)

Local versus Global Allocation Policies



Age
A0 10
A1 7
A2 5
A3 4
A4 6
A5 3 t = 3 (oldest in A)
B0 9
B1 4
B2 6
B3 2 t = 2 (oldest of all)
B4 5
B5 6
B6 12
C1 3
C2 5
C3 6
(a)
A0 A1 A2 A3 A4 A6 B0 B1 B2 B3 B4 B5 B6 C1 C2 C3
(b)
A0 A1 A2 A3 A4 A5 B0 B1 B2 A6 B4 B5 B6 C1 C2 C3
(c)

Local versus global page replacement.
(a) Original configuration. (b) Local page replacement.
(c) Global page replacement.

Multi-level Page Tables

- Problem: page tables can be too large
  - In 4KB page size, with 32 bit logical addresses
    - 2^{12} = 4096 (bytes), so 12 bits to get to the page byte offset
    - 2^{20} = 1 million PTEs , so 20 bits to get to the page address
- Solution: use multi-level page tables
  - 1st level page table has pointers to 2nd level page tables
  - 2nd level page table has actual physical page numbers in it
- http://www.youtube.com/watch?v=Z4kSOv49GNc

Fast Translation Using a TLB

- A widely implemented scheme for
  - Speeding up paging
  - Handling large virtual address spaces
- Small HW device for mapping virtual address to physical address without going through the page table in memory if possible
- http://www.youtube.com/watch?v=uyrSn3qbZ8U
- http://www.youtube.com/watch?v=95QpHJX55bM

Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors