Chapter 3
Memory Management

Virtual Memory-the history

- Keep multiple parts of programs in memory
- Swapping is too slow (100 Mbytes/sec disk transfer rate=>10 sec to swap out a 1 Gbyte program)
- Overlays-programmer breaks program into pieces which are swapped in by overlay manager
  - Ancient idea-not really done
  - Too hard to do-programmer has to break up program

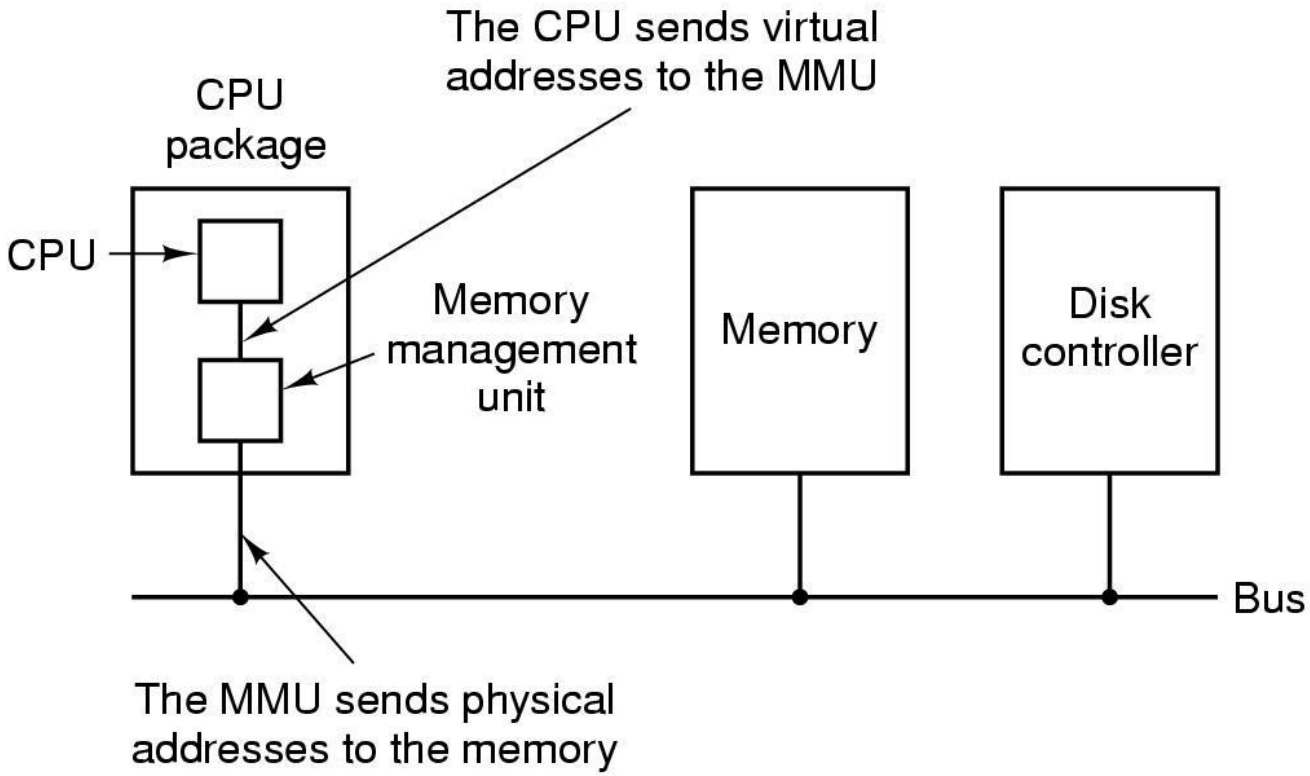
Virtual Memory

- Program's address space is broken up into fixed size pages
- Pages are mapped to physical memory
- If instruction refers to a page in memory, fine
- Otherwise OS gets the page, reads it in, and re-starts the instruction
- While page is being read in, another process gets the CPU

Memory Management Unit

- Memory Management Unit generates physical address from virtual address provided by the program

Memory Management Unit

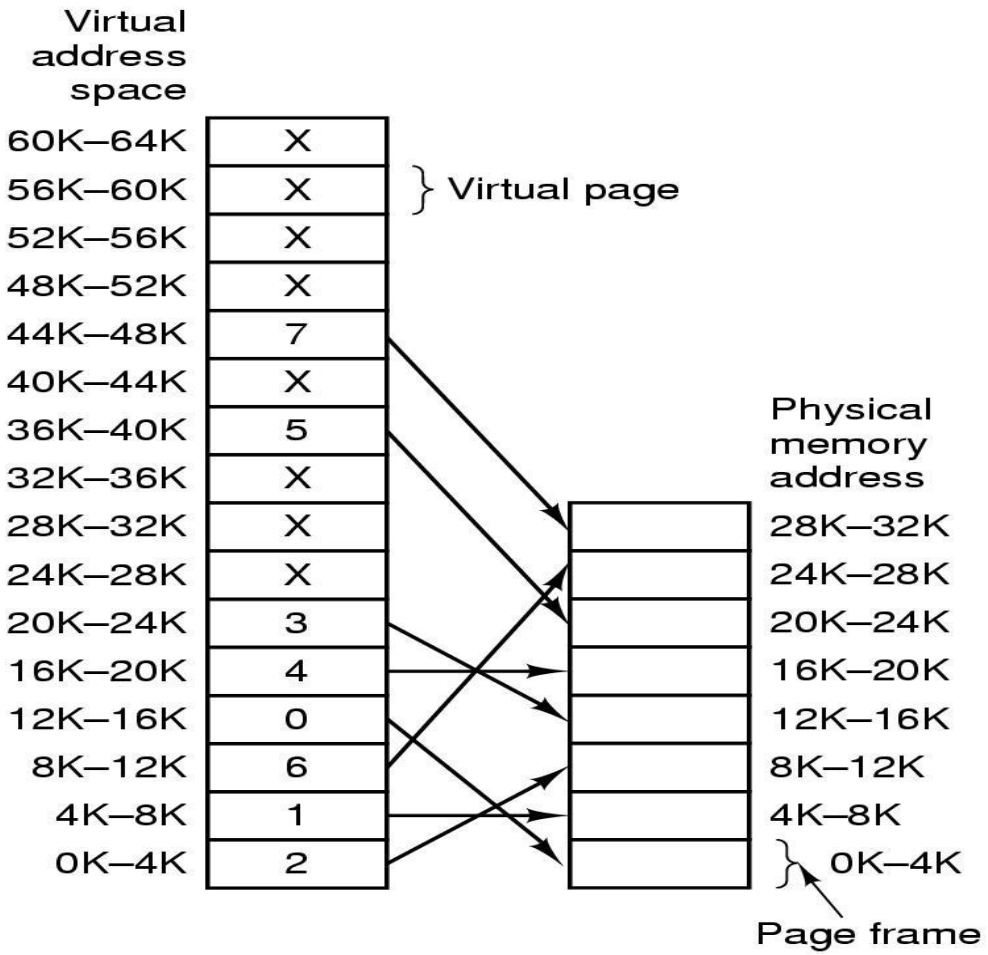


MMU maps virtual addresses to physical addresses and puts them on memory bus

Pages and Page Frames

- Virtual addresses divided into pages
  - 512 bytes-64 KB range
  - Transfer between RAM and disk is in whole pages
  - Example on next slide

Mapping of pages to page frames



16 bit addresses, 4 KB pages

32 KB physical memory,

16 virtual pages and 8 page frames

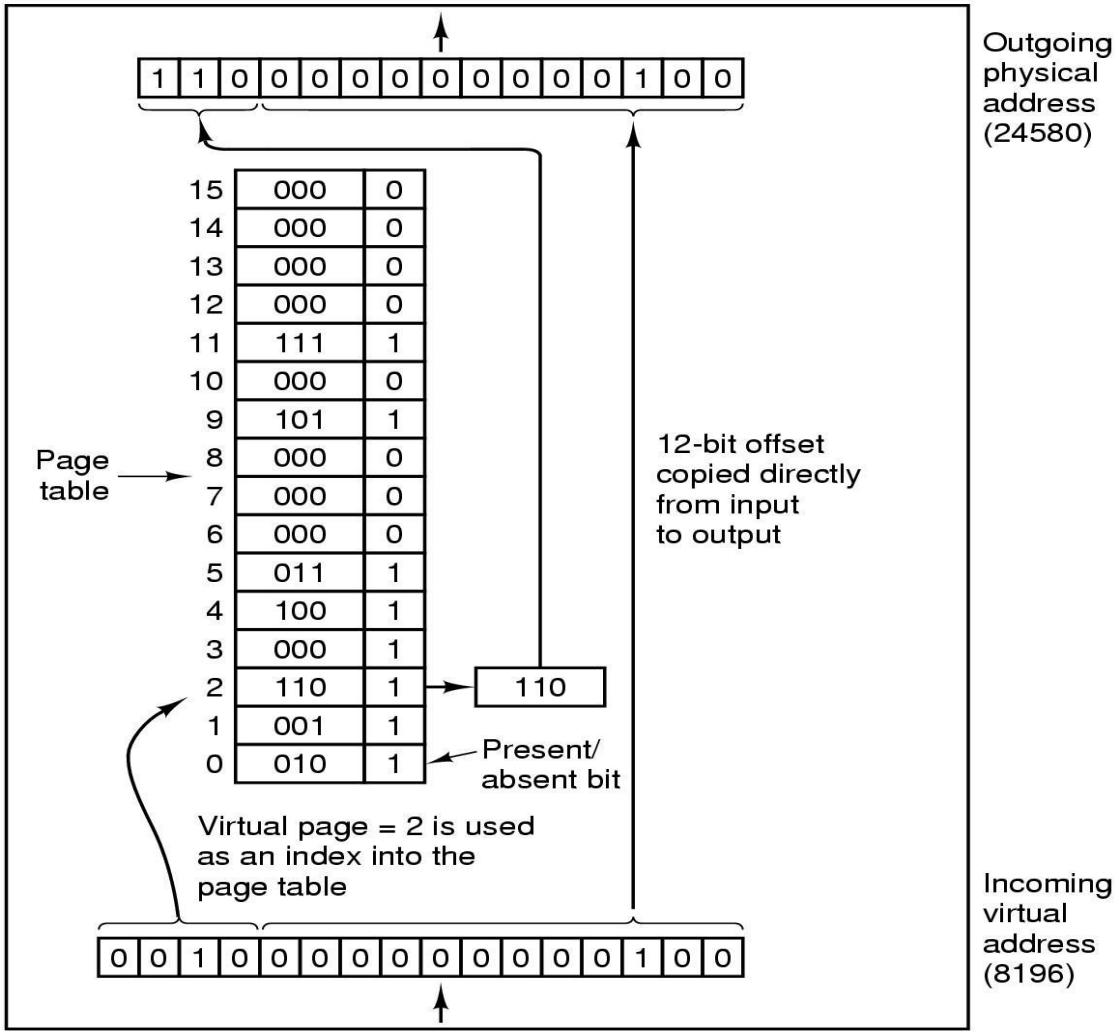
Page Fault Processing

- Present/absent bit tells whether page is in memory
- What happens If address is not in memory?
- Trap to the OS
  - OS picks page to write to disk
  - Brings page with (needed) address into memory
  - Re-starts instruction

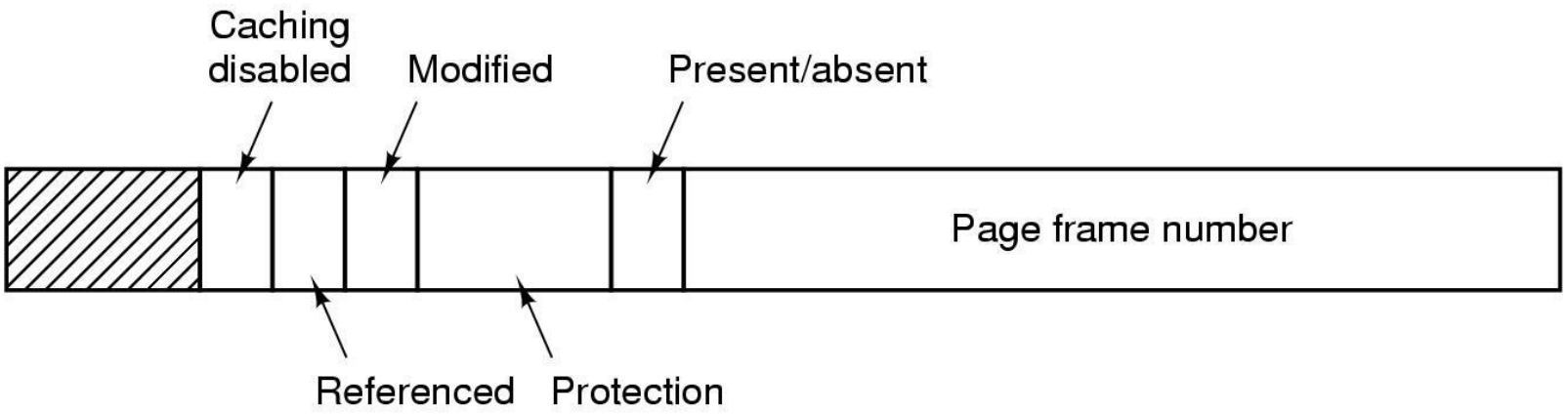
Page Table

- Virtual address={virtual page number, offset}
- Virtual page number used to index into page table to find page frame number
- If present/absent bit is set to 1, attach page frame number to the front of the offset, creating the physical address
- which is sent on the memory bus

MMU operation



Structure of Page Table Entry



- Modified (dirty) bit: 1 means written to => have to write it to disk. 0 means don't have to write to disk.
- Referenced bit: 1 means it was either read or written. Used to pick page to evict. Don't want to get rid of page which is being used.
- Present (1) / Absent (0) bit
- Protection bits: r, w, r/w

Problems for paging

- Virtual to physical mapping is done on every memory reference => mapping must be fast
- If the virtual address space is large, the page table will be large. 32 bit addresses now and 64 bits becoming more common

Stupid solutions

- Bring page table for a process into MMU when it is started up and store it in registers
- Keep page table in main memory

Speed up Address Translation

- Most programs access a small number of pages a great deal
- Add Translation Lookaside Buffer (TLB) to MMU
  - Stores frequently accessed frames

Translation Lookaside Buffers

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

Valid bit indicates whether page is in use or not

Translation Lookaside Buffer(TLB)

- If address is in MMU, avoid page table
- Uses parallel search to see if virtual page is in the TLB
- If not, does page table look up and evicts TLB entry, replacing it with page just looked up

Software TLB management

- Risc machines manage TLB in software
- TLB fault processed by OS instead of by MMU hardware
- Results less hardware in MMU and OK performance
- Software can figure out which pages to pre-load into TLB (eg. Load server after client request)
- Keeps cache of frequently used pages

Page Replacement Algorithms

- If new page is brought in, need to chose a page to evict
- Don’t want to evict heavily used pages
- If page has been written to, need to copy it to disk.
- Otherwise, a good copy is on the disk=>can write over it

Page Replacement Algorithms-the Laundry List

- Optimal page replacement algorithm
- Not recently used page replacement
- First-in, first-out page replacement
- Second chance page replacement
- Clock page replacement
- Least recently used page replacement
- Working set page replacement
- WSClock page replacement

Optimal Page Replacement

- Pick the one which will not used before the longest time
- Not possible unless know when pages will be referenced (crystal ball)
- Used as ideal reference algorithm

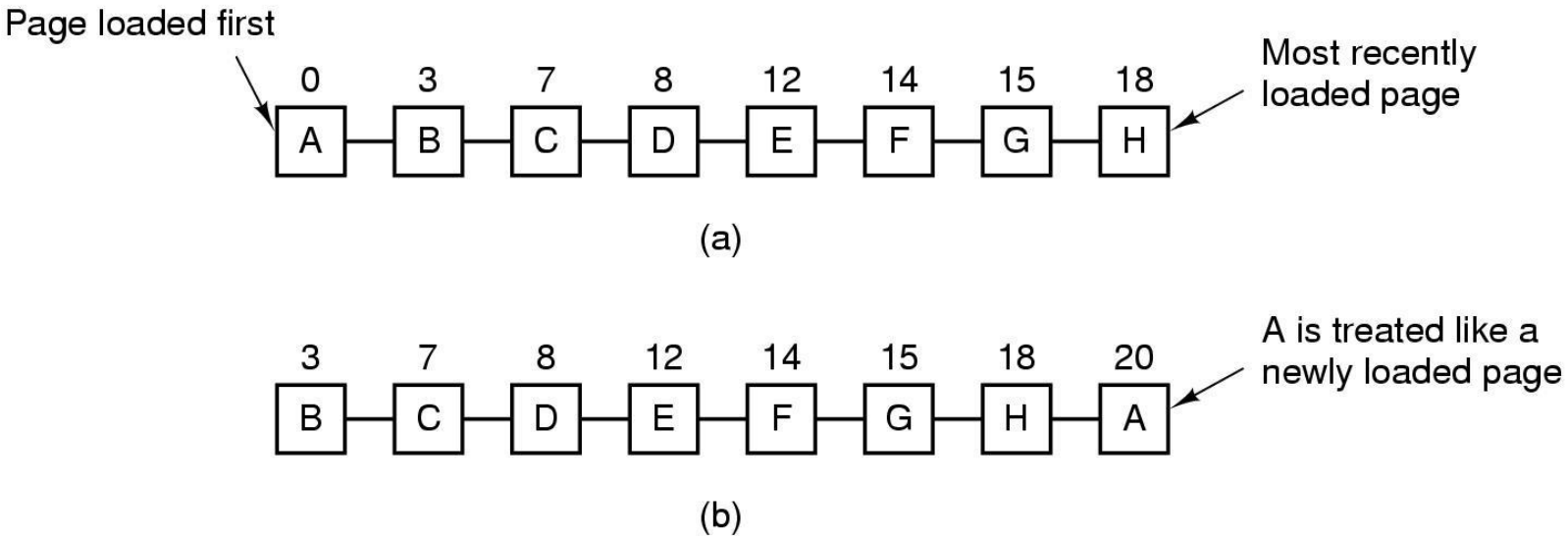
Not recently used

- Use R and M bits
- Periodically clear R bit
  - Class 0: not referenced, not modified
  - Class 1: not referenced, modified
  - Class 2: referenced, not modified
  - Class 3: referenced, modified
- Pick lowest priority page to evict

FIFO

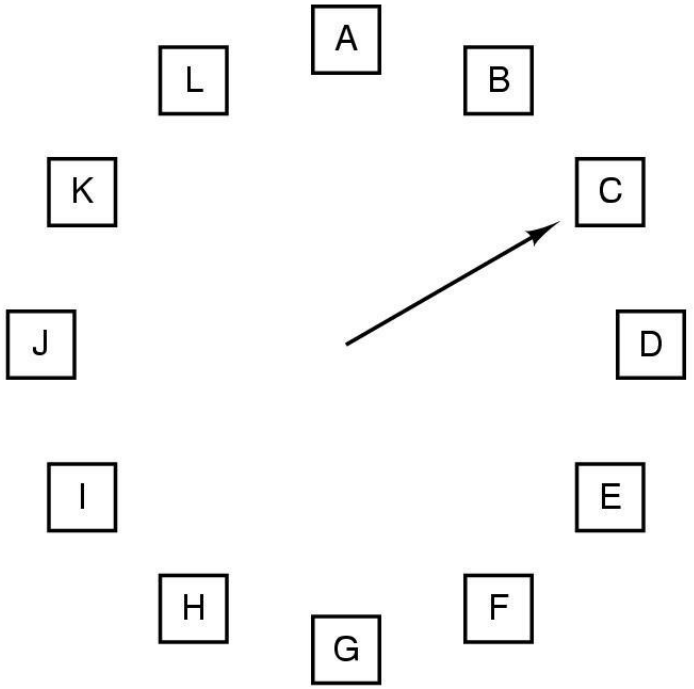
- Keep list ordered by time (latest to arrive at the end of the list)
- Evict the oldest, i.e. head of the line
- Easy to implement
- Oldest might be most heavily used! No knowledge of use is included in FIFO

Second Chance Algorithm



- Pages sorted in FIFO order by arrival time.
- Examine R bit. If zero, evict. If one, put page at end of list and R is set to zero.
- If change value of R bit frequently, might still evict a heavily used page

The Clock Page Replacement Algorithm



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

R = 0: Evict the page

R = 1: Clear R and advance hand

Clock

- Doesn’t use age as a reason to evict page
- Faster-doesn’t manipulate a list
- Doesn’t distinguish between how long pages have not been referenced

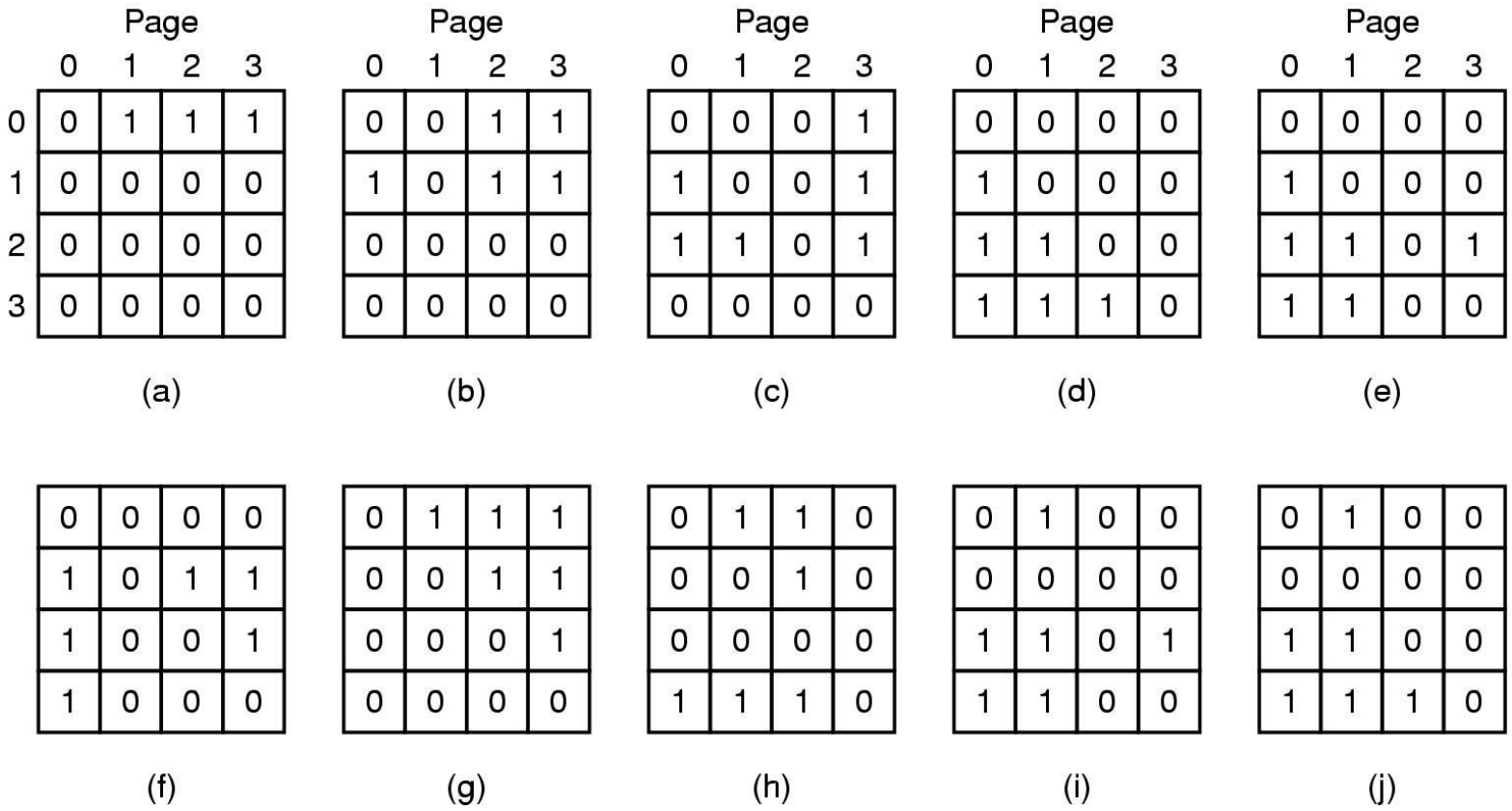
LRU

- Approximate LRU by assuming that recent page usage approximates long term page usage
- Could associate counters with each page and examine them but this is expensive

LRU-the hardware array

- Associate counter with each page.
  - At each reference increment counter.
  - Evict page with lowest counter
- Keep n x n array for n pages.Upon reference page k, put 1’s in row k and 0’s in column k.
  - Row with smallest binary value corresponds to LRU page. Evict k!
  - Easy hardware implementation

LRU-hardware

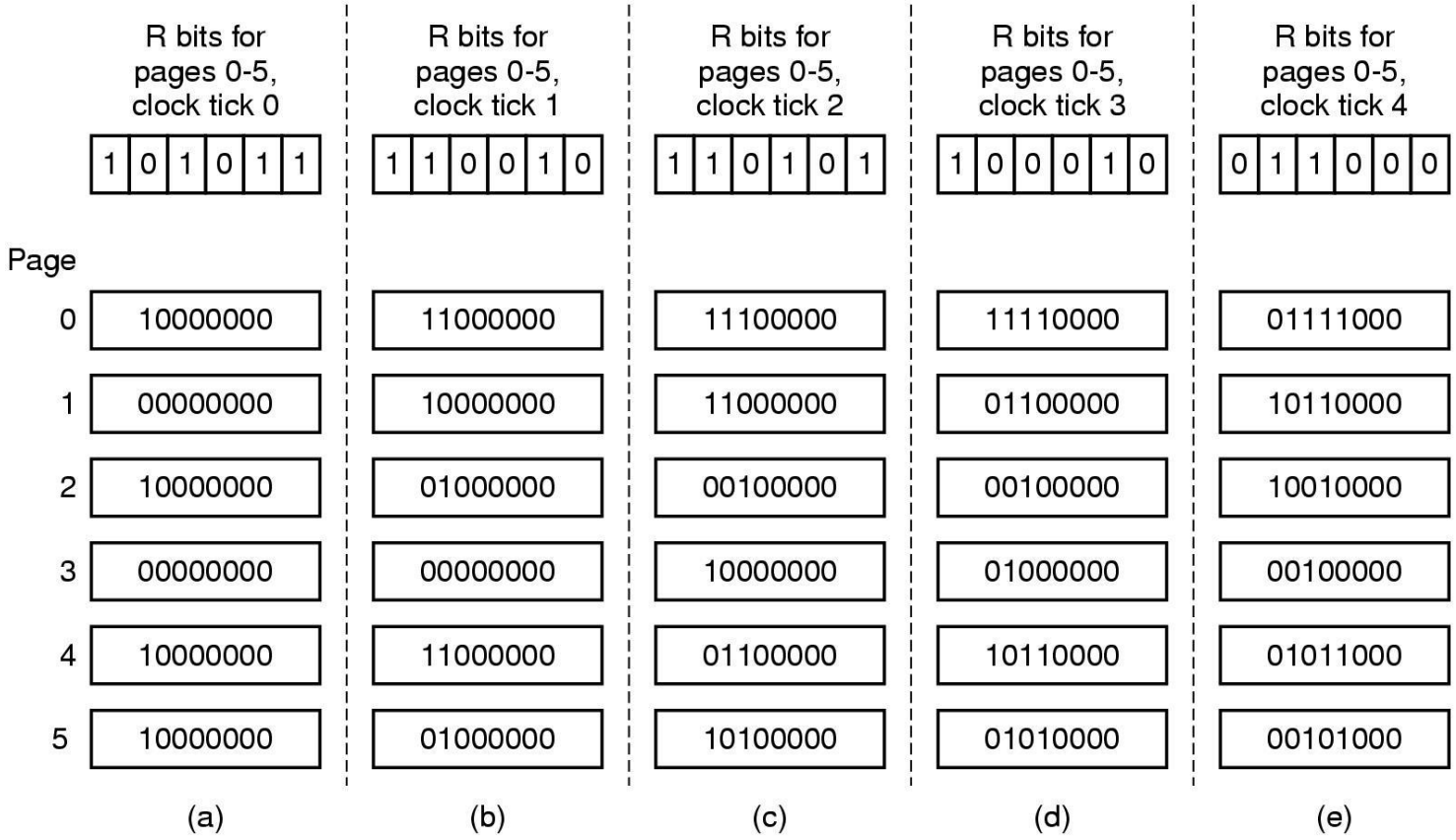


LRU using a matrix when pages are referenced in the order 0, 1, 2, 3, 2, 1, 0, 3, 2, 3.

LRU-software

- Hardware uses space=> software implementation
- Make use of software counters

LRU in Software



LRU-software

- “aging” algorithm
- Keep a string of values of the R bits for each clock tick (up to some limit)
- After tick, shift bits right and add new R values on the left
- On page fault, evict page with lowest counter
- Size of the counter determines the history

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