Page Replacement Algorithms

- Page fault forces choice
  - which page must be removed
  - make room for incoming page
- Modified page must first be saved
  - unmodified just overwritten
- Better not to choose an often used page
  - will probably need to be brought back in soon

Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future
  - Optimal but unrealizable

- Estimate by …
  - logging page use on previous runs of process
  - although this is impractical

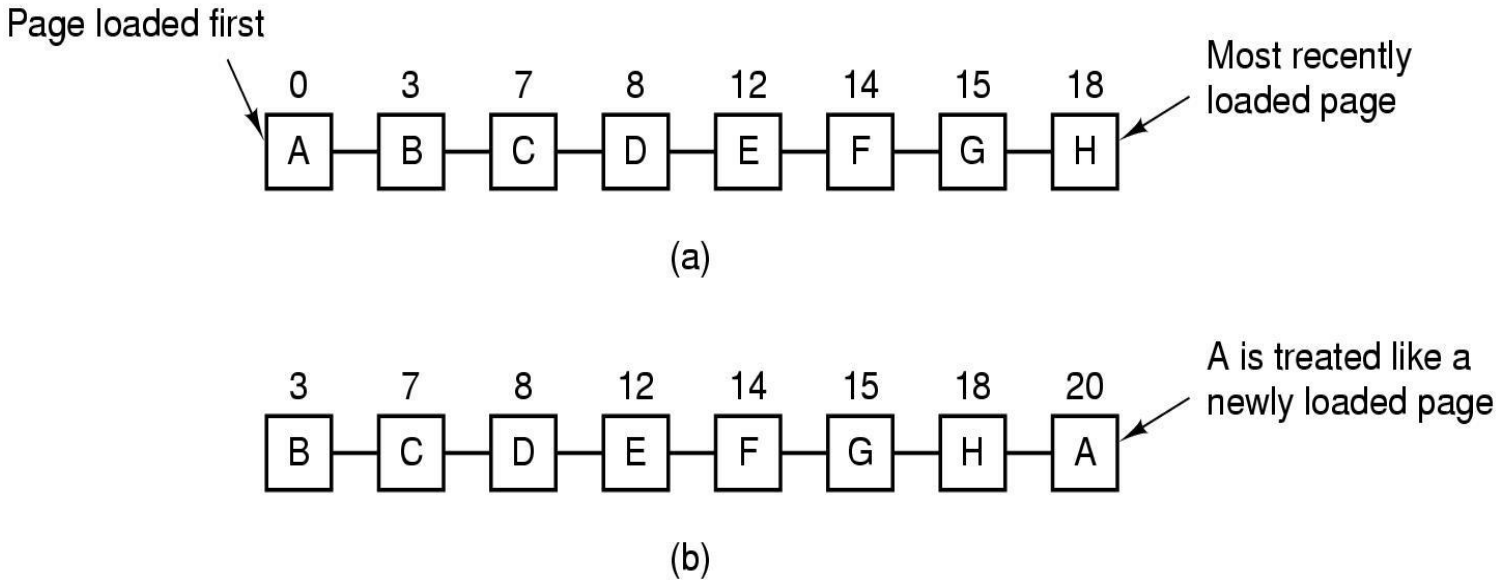
Not Recently Used Page Replacement Algorithm

- Each page has:
  - R bit: set when page is referenced(read/written)
  - M bit: set when the page is modified(written)
- Periodically, the R bit is cleared.
  - To distinguish pages that have not been referenced recently from those that have been
- Pages are classified
  1. not referenced, not modified
  2. not referenced, modified
  3. referenced, not modified
  4. referenced, modified
  - Better to remove a modified page that has not been referenced in at least one clock tick than a clean page that is in heavy use
- NRU removes page at random from lowest numbered nonempty class
- Easy to understand, moderately efficient to implement, gives an adequate performance

FIFO Page Replacement Algorithm

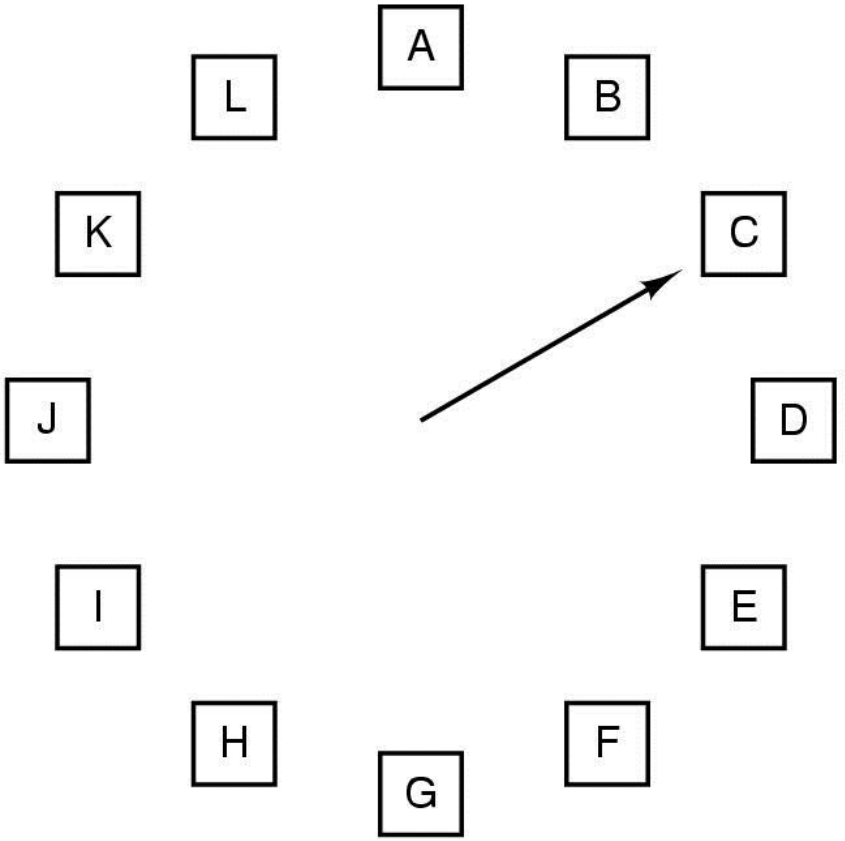
- Maintain a linked list of all pages
  - in order they came into memory
  - with the page at the head of the list the oldest one and the page at the tail the most recent arrival
- Page at beginning of list replaced
- Disadvantage
  - page in memory the longest may be often used

Second Chance Page Replacement Algorithm



- Operation of a second chance
  - pages sorted in FIFO order
  - The page list when a fault occurs at time 20 is shown in the figure above.
  - If A has *R* bit cleared, it is evicted from memory, either by being written to disk (if it is dirty), or just abandoned (if it is clean)
  - If the R bit is set, A is put onto the end of the list and the R bit is cleared. The search continues.
  - If all the pages have been referenced, it degenerates into pure FIFO.
  - Inefficient because it constantly moves pages around on its list

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:

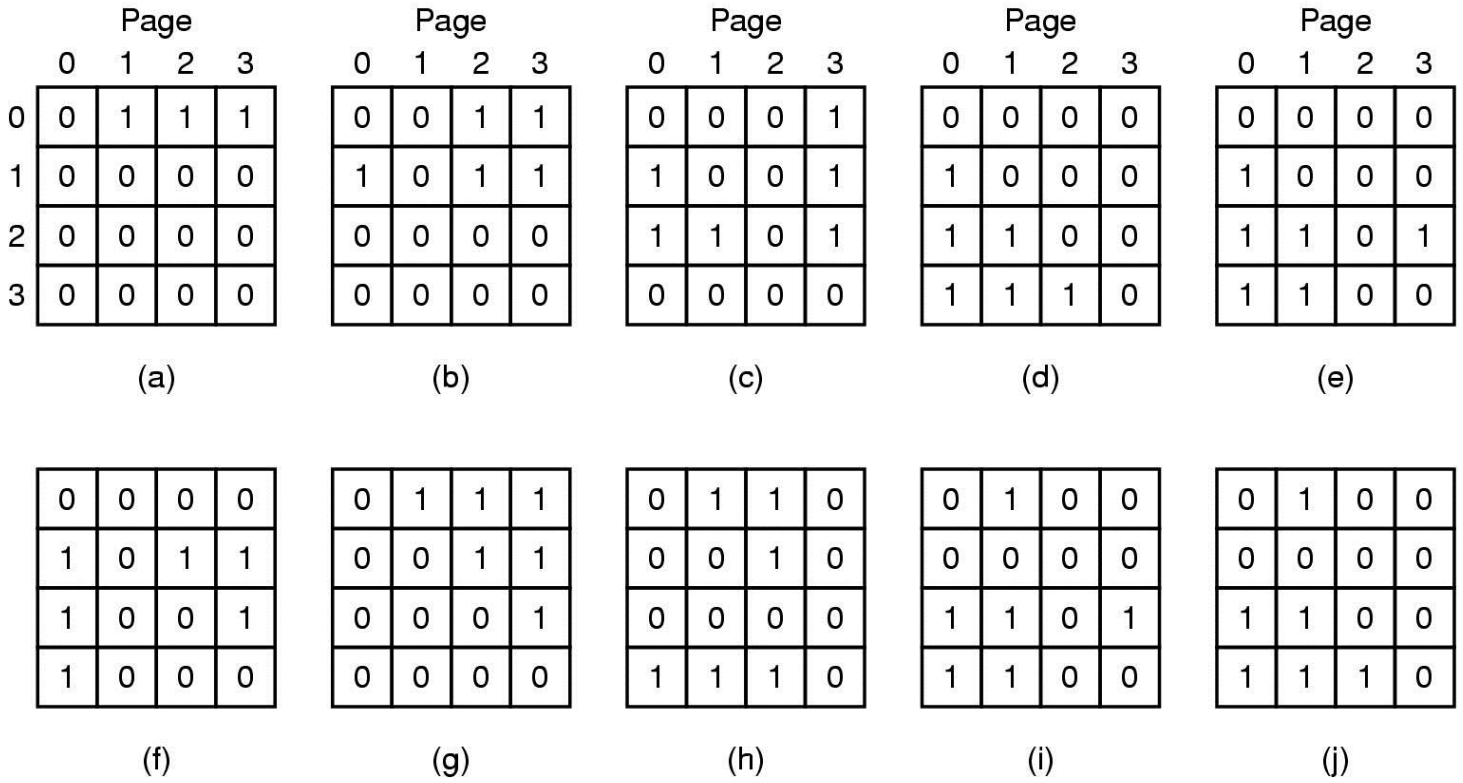
R = 0: Evict the page

R = 1: Clear R and advance hand

Least Recently Used (LRU)

- Assume pages used recently will used again soon
  - throw out page that has been unused for longest time
- Must keep a linked list of all pages in memory
  - most recently used at front, least at rear
  - update this list every memory reference !!
- Alternatively keep counter in each page table entry
  - A counter, C is automatically incremented after each instruction.
  - After each memory reference, the current value of C is stored in the page table entry for the page just referenced.
  - When a page fault occurs, the operating system examines all the counters in the page table to find the lowest one, which is the least recently used.

Another LRU



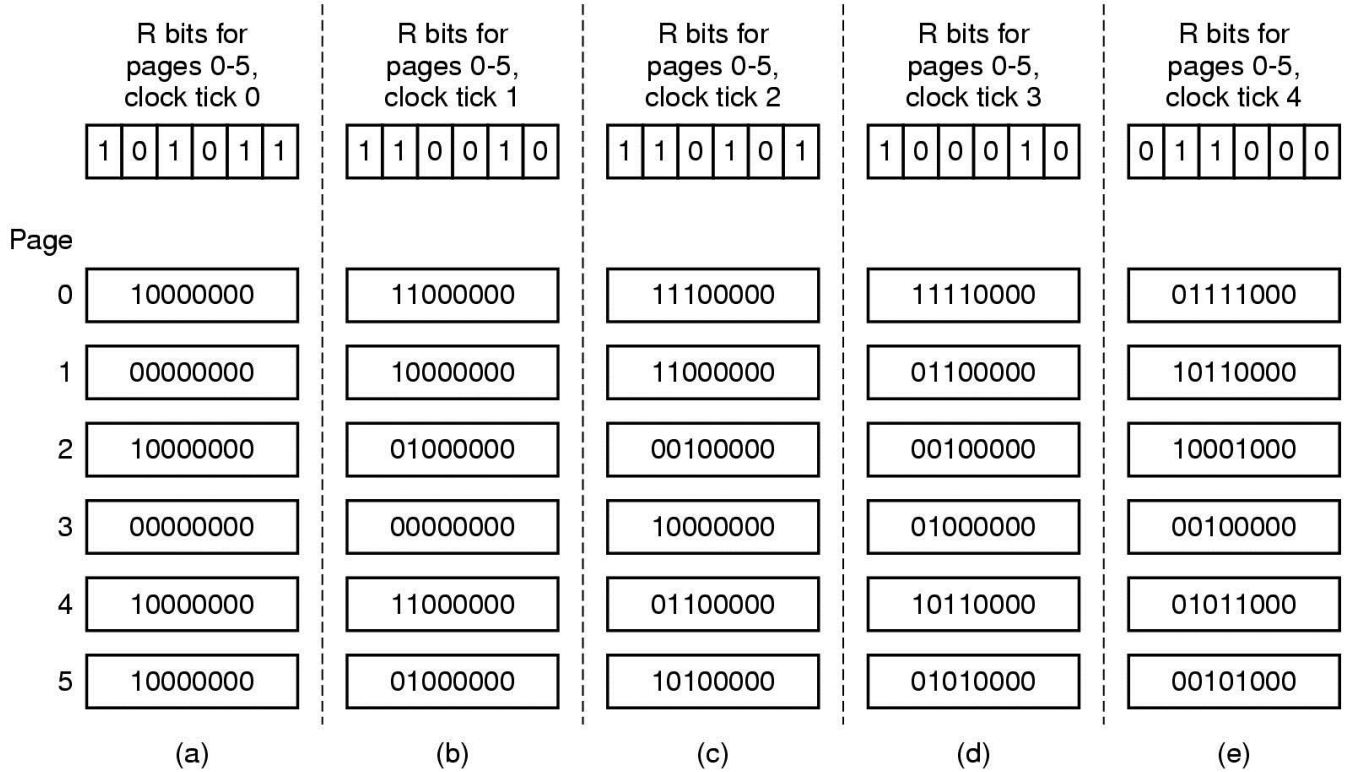
- LRU using a matrix – pages referenced in order 0,1,2,3,2,1,0,3,2,3
- The row whose binary value is lowest the least recently used.

Simulating LRU in Software
NFU(Not Frequently Used)

- Software counter associated with each page, initially zero.
- At each clock interrupt, the operating system scans all the pages in memory.
- For each page, the R bit, which is 0 or 1, is added to the counter. (In effect, the counters keeps track of how often each page has been referenced.)
- When a page fault occurs, the page with the lowest counter is chosen for replacement.
- Never forgets anything.

Simulating LRU in Software

Aging



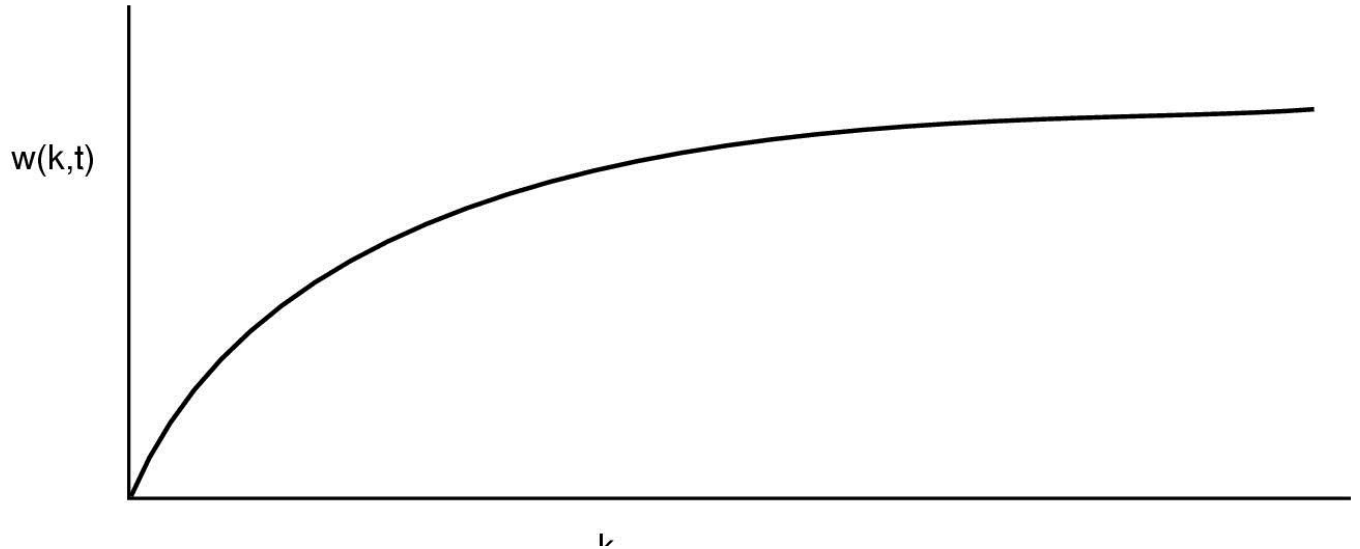
- First, the counters are each shifted right 1 bit.
- Second, the R bit is added to the leftmost bit.
- When a page fault occurs, the page whose counter is the lowest is removed.
- Different from LRU in two ways
  - Can't distinguish two references in the same clock interval
  - Limited by a finite number of bits of the counter

The Working Set Page Replacement Algorithm (1)

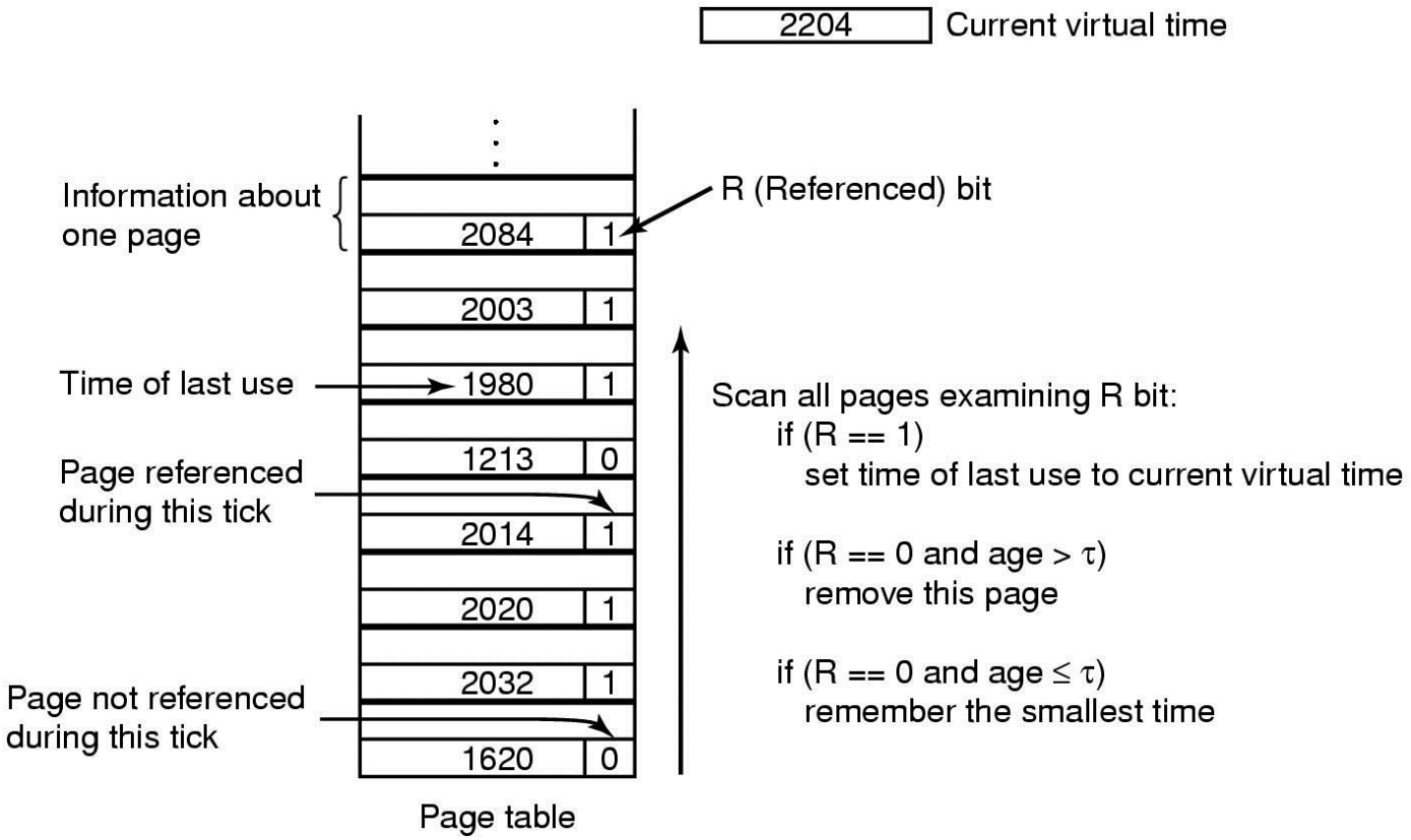
- Demand Paging
  - Pages are loaded only on demand, not in advance.
    - Processes are started up with none of their pages in memory.
    - As soon as the CPU tries to fetch the first instruction, it gets a page fault, causing the OS to bring in the page containing the first instruction.
    - After a while, the process has most of the pages it needs and settles down to run with relatively few page faults.
- Locality of reference
  - A process references only a relatively small fraction of its pages during any phase of execution.
- Working set
  - A set of pages that a process is currently using
- Thrashing
  - A program causing page faults every few instructions is said to be thrashing.
  - Can happen when the available memory is too small to hold the entire working set.
- Working set model (Prepaging)
  - Many paging systems keep track of each process' working set and make sure that it's in memory before letting the process run
  - A working set changes over time.

The Working Set Page Replacement Algorithm (1)

- working set
  - w(k,t)
  - A set of pages used by the *k* most recent memory references at time, t



The Working Set Page Replacement Algorithm (2)

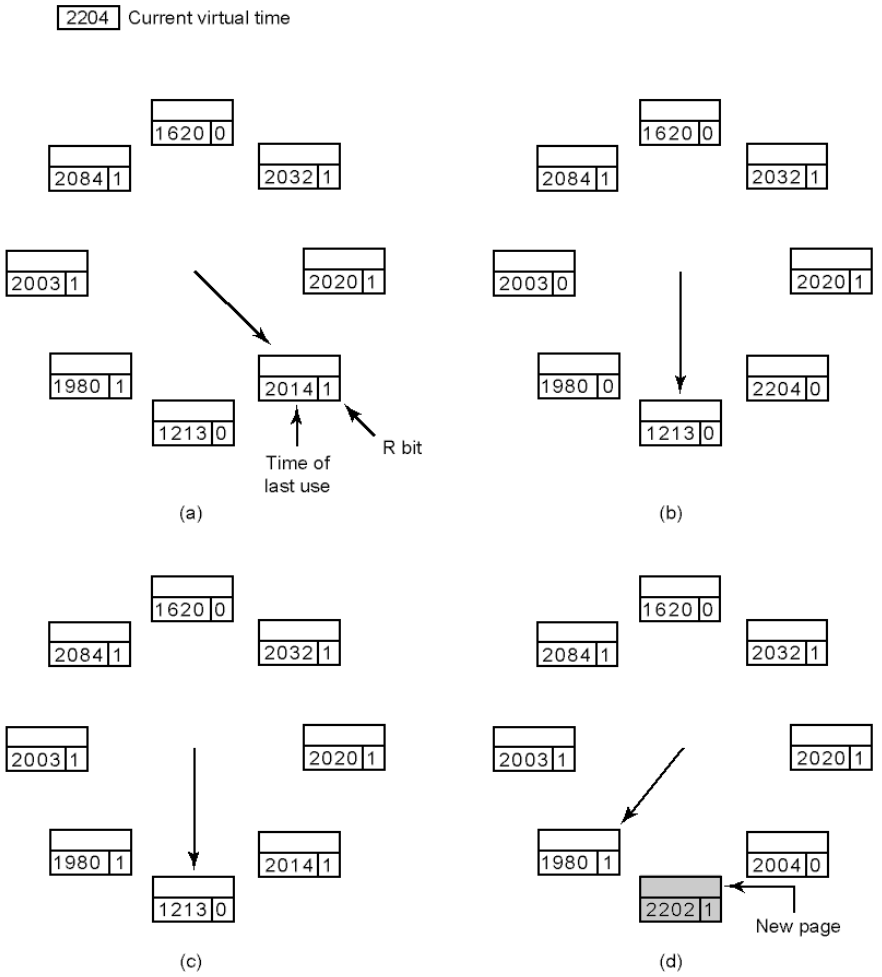


The working set algorithm

The Working Set Page Replacement Algorithm (3)

- Basic strategy
  - When a page fault occurs, find a page not in the working set and evict it.
- Current virtual time
  - Amount of CPU time a process has actually used since it started
- A periodic interrupt clears the R(Referenced) bit on every clock tick.
- On every page fault, the page table is scanned to look for a suitable page to evict.

The WSClock Page Replacement Algorithm

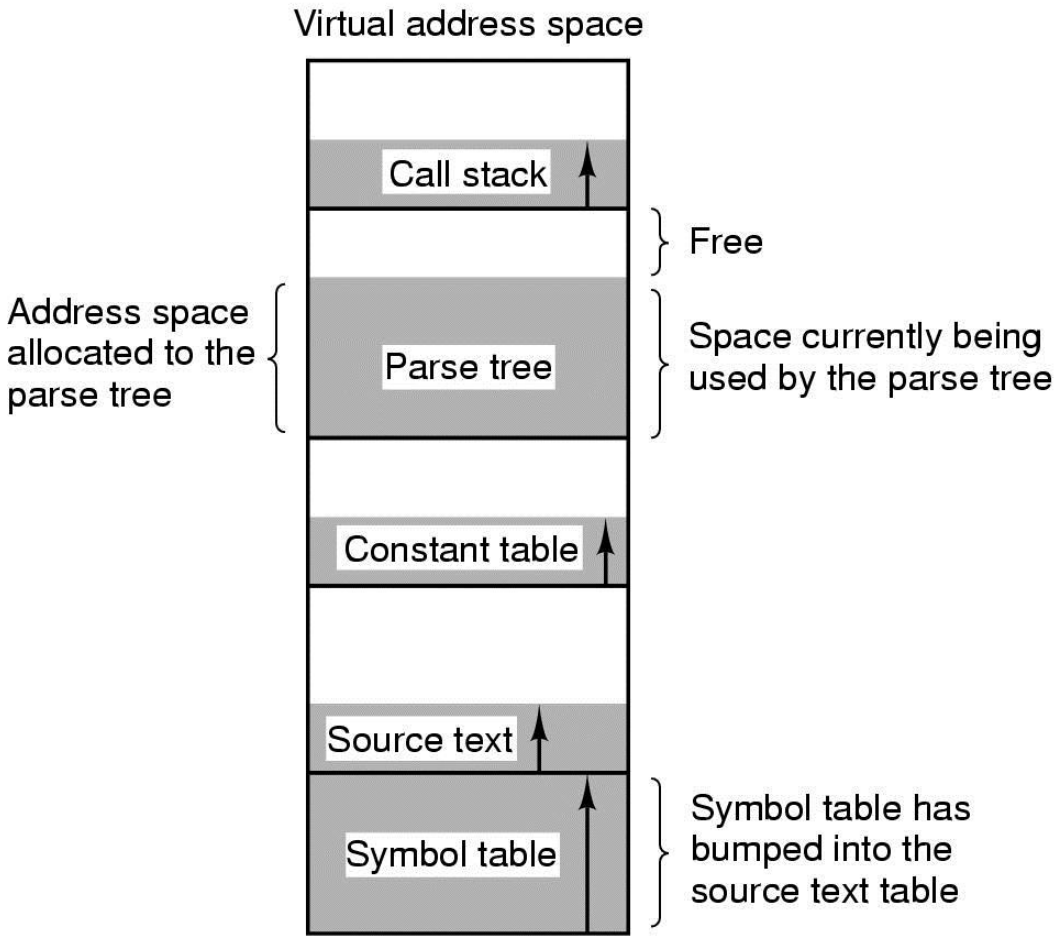


Operation of the WSClock algorithm

Review of Page Replacement Algorithms

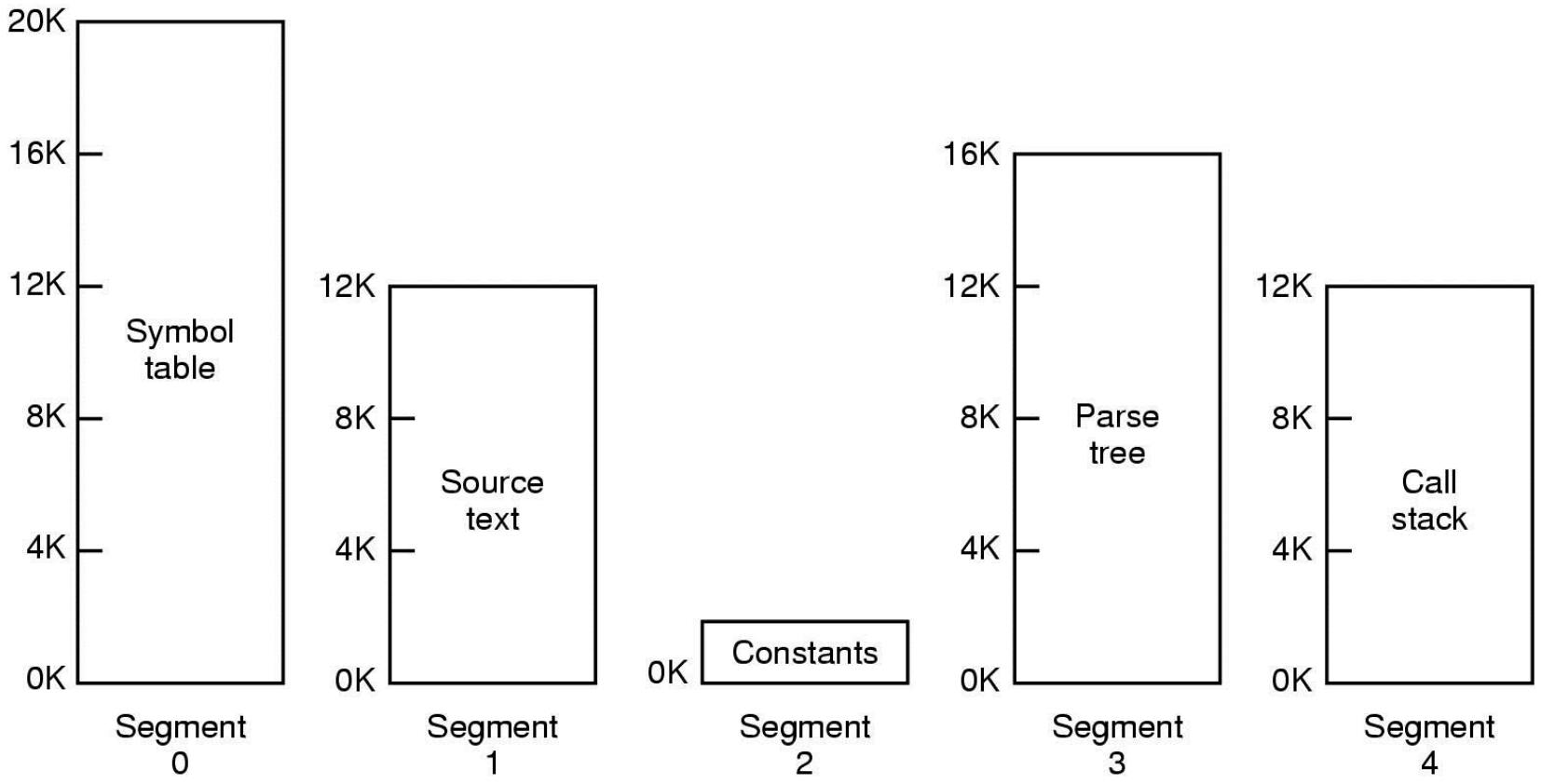
Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not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

Segmentation (1)



- One-dimensional address space with growing tables
- One table may bump into another

Segmentation (2)



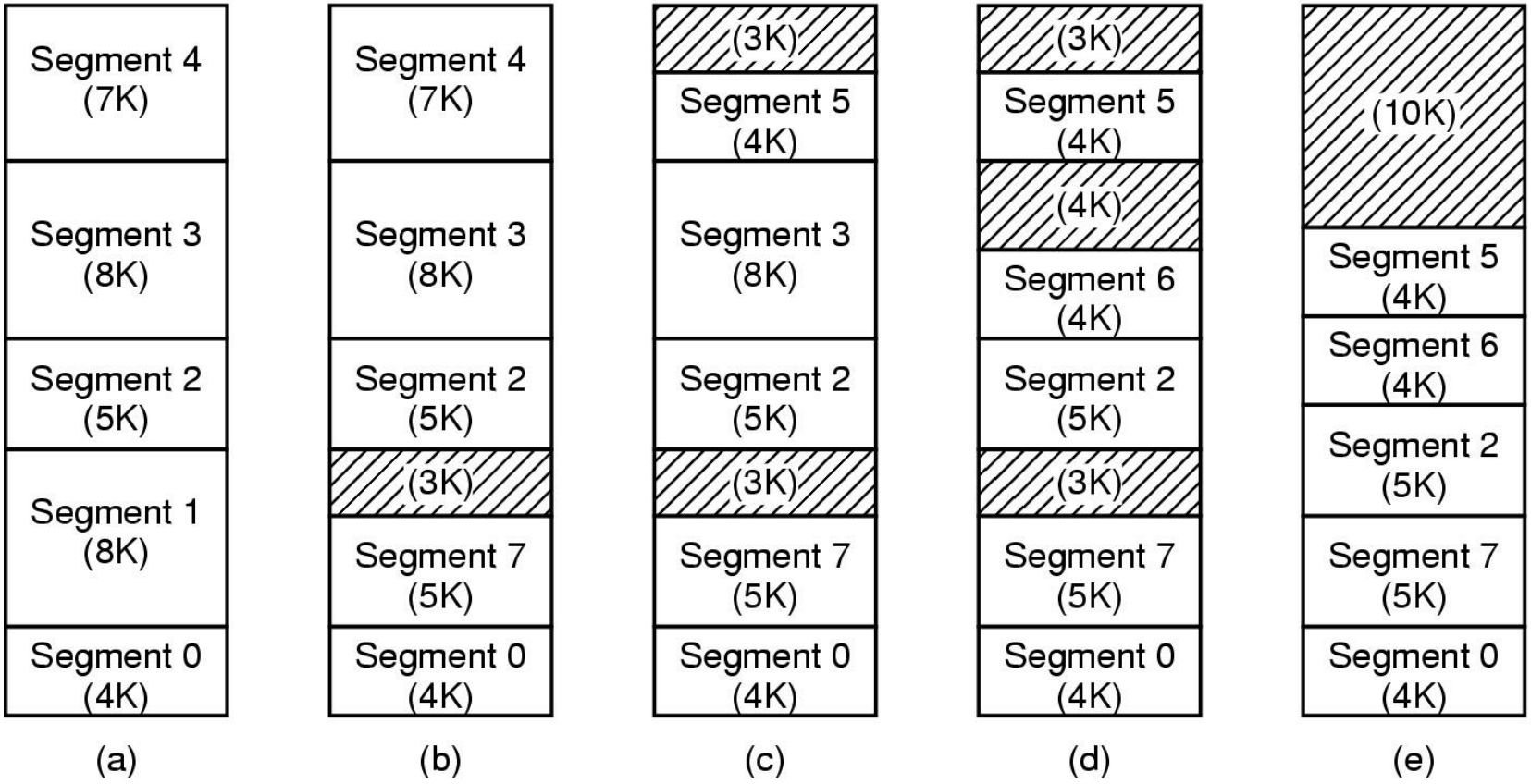
Allows each table to grow or shrink, independently

Segmentation (3)

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Comparison of paging and segmentation

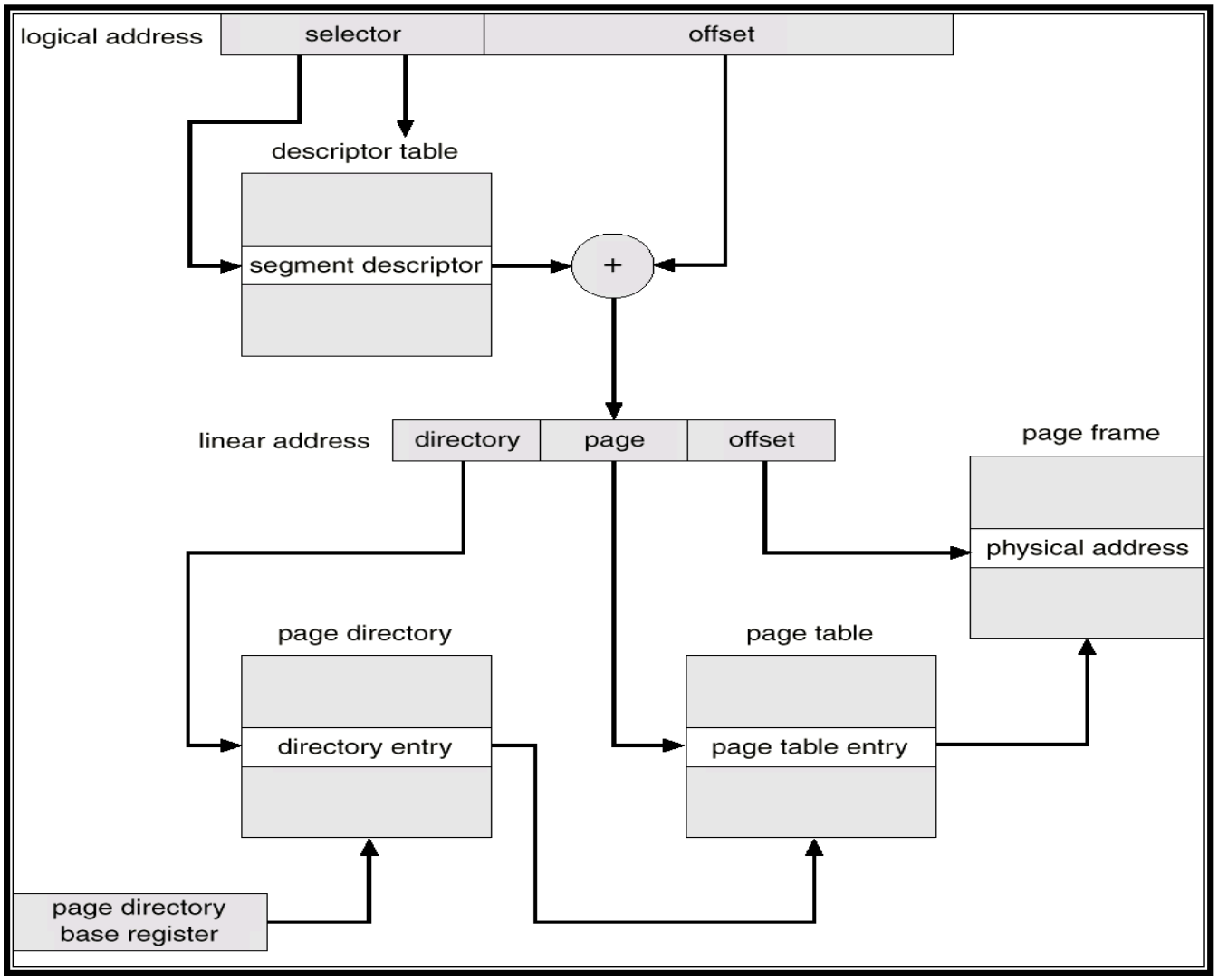
Implementation of Pure Segmentation



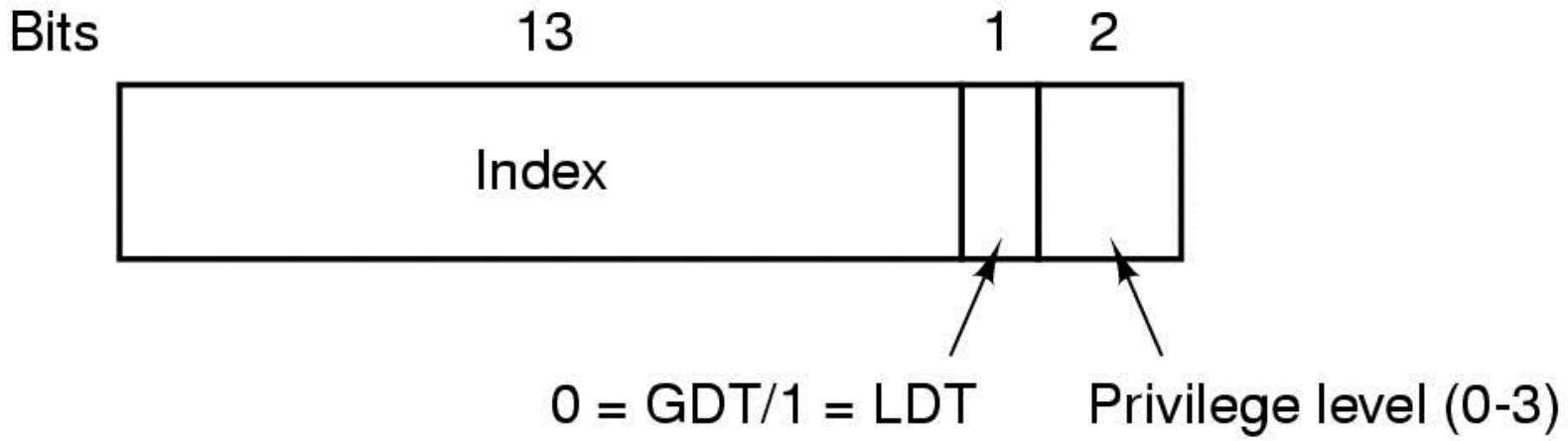
(a)-(d) Development of checkerboarding

(e) Removal of the checkerboarding by compaction

Segmentation with Paging: Pentium

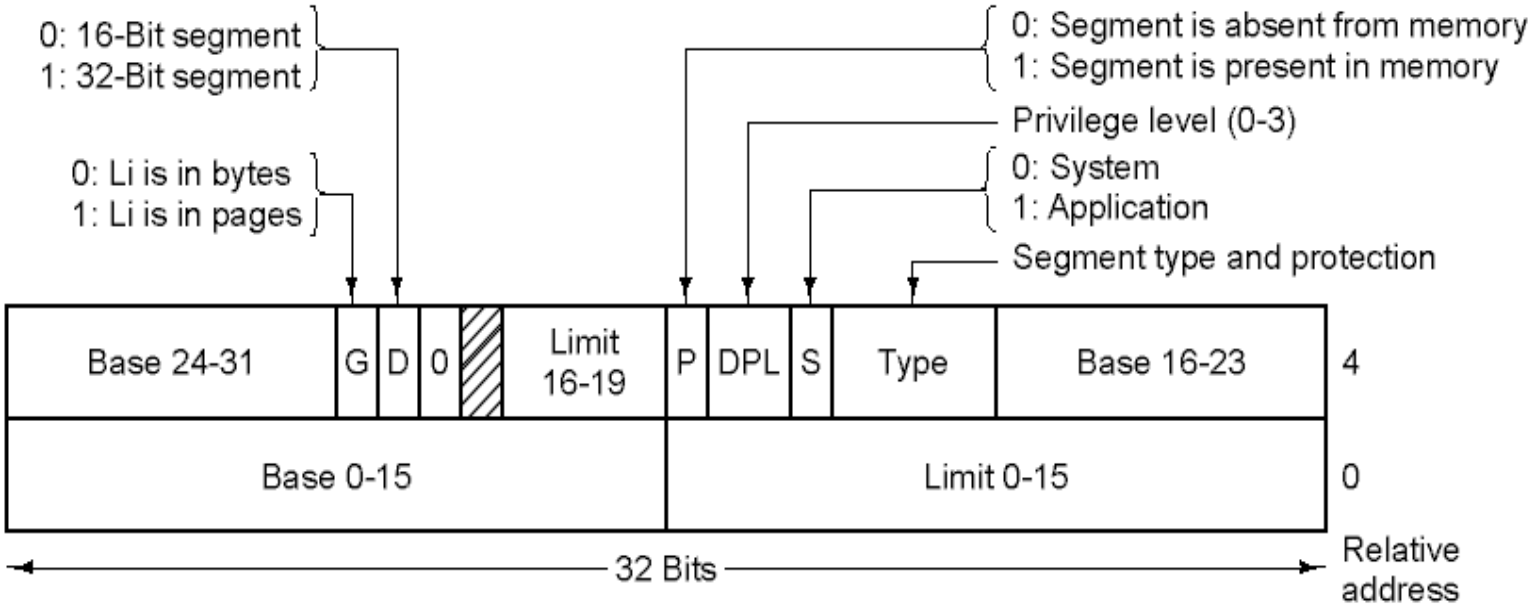


Segmentation with Paging: Pentium (1)



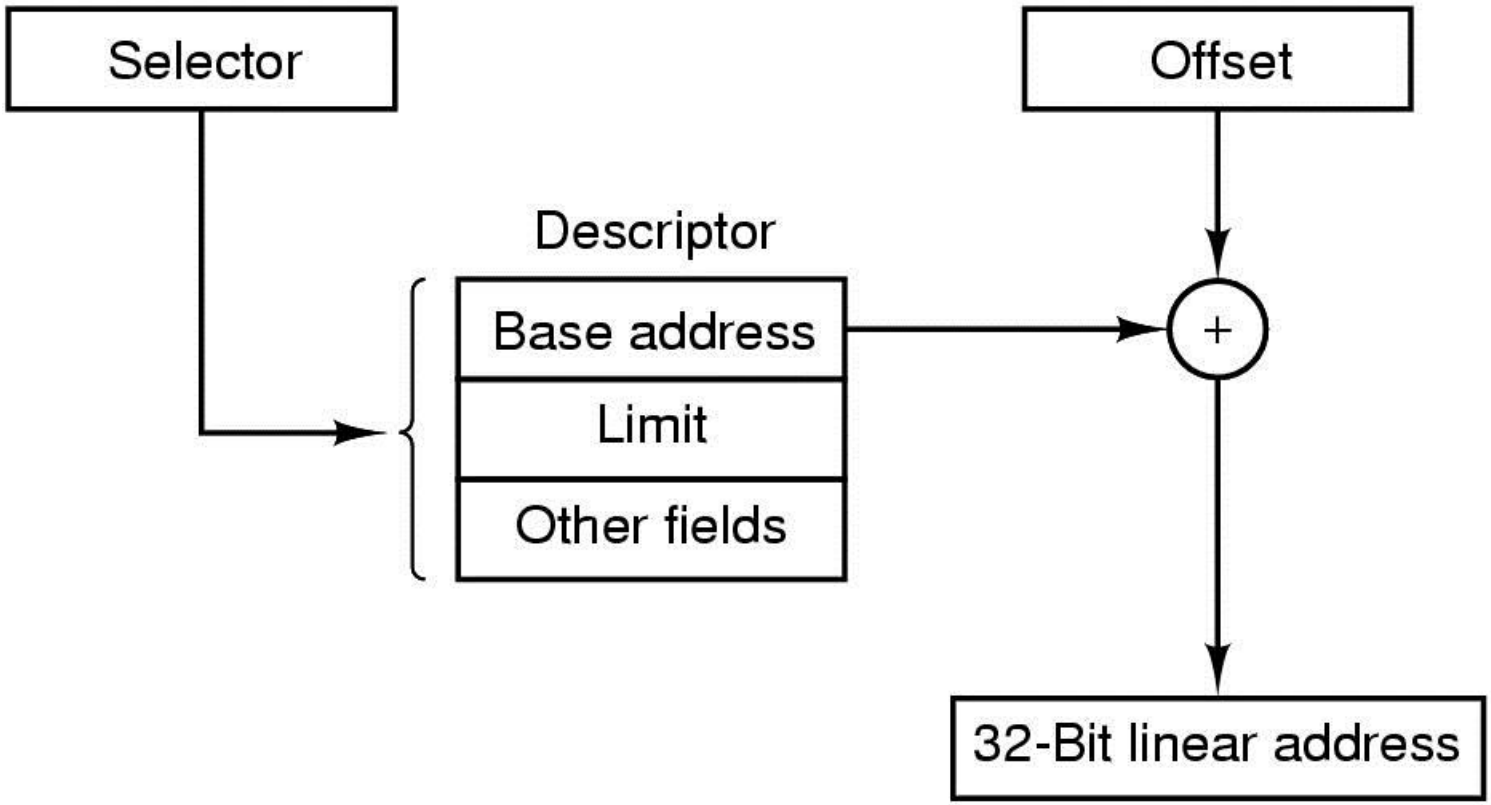
A Pentium selector

Segmentation with Paging: Pentium (2)



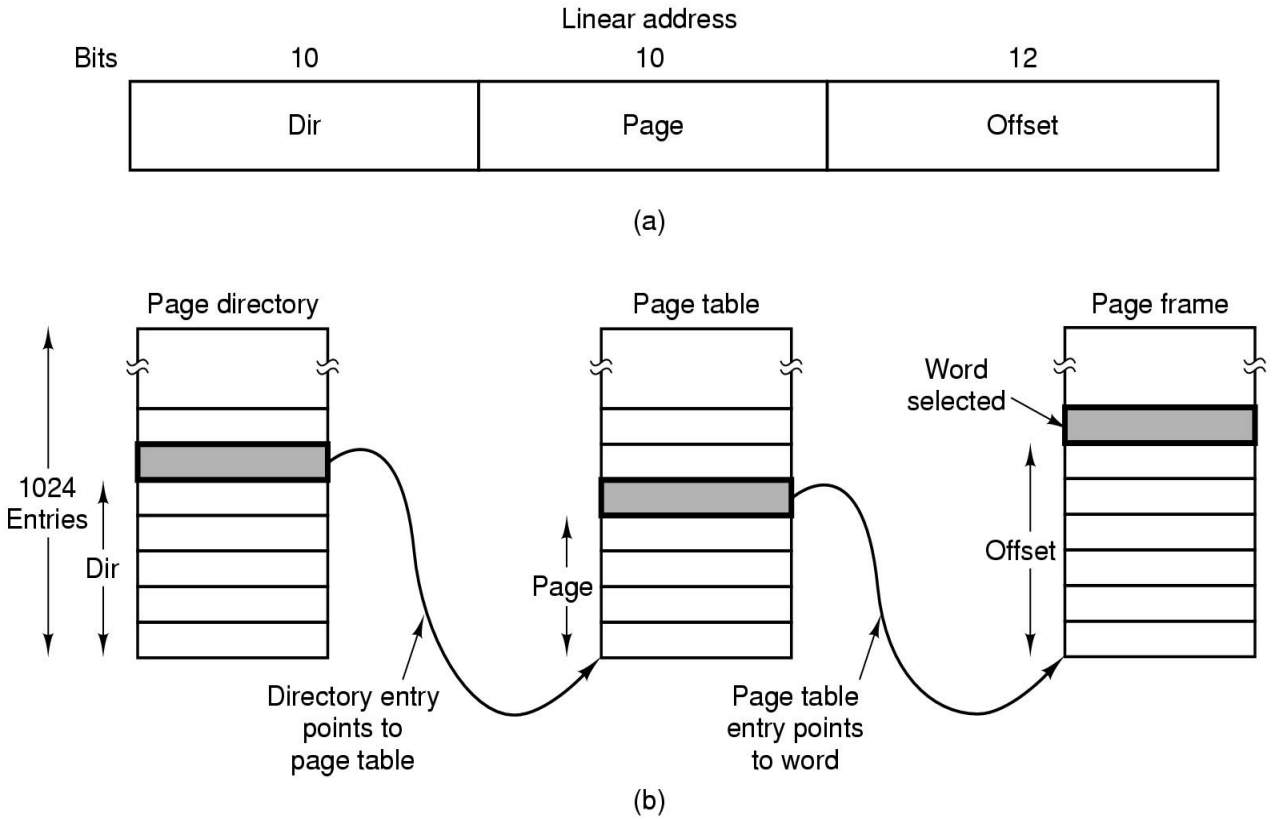
- Pentium code segment descriptor
- Data segments differ slightly

Segmentation with Paging: Pentium (3)



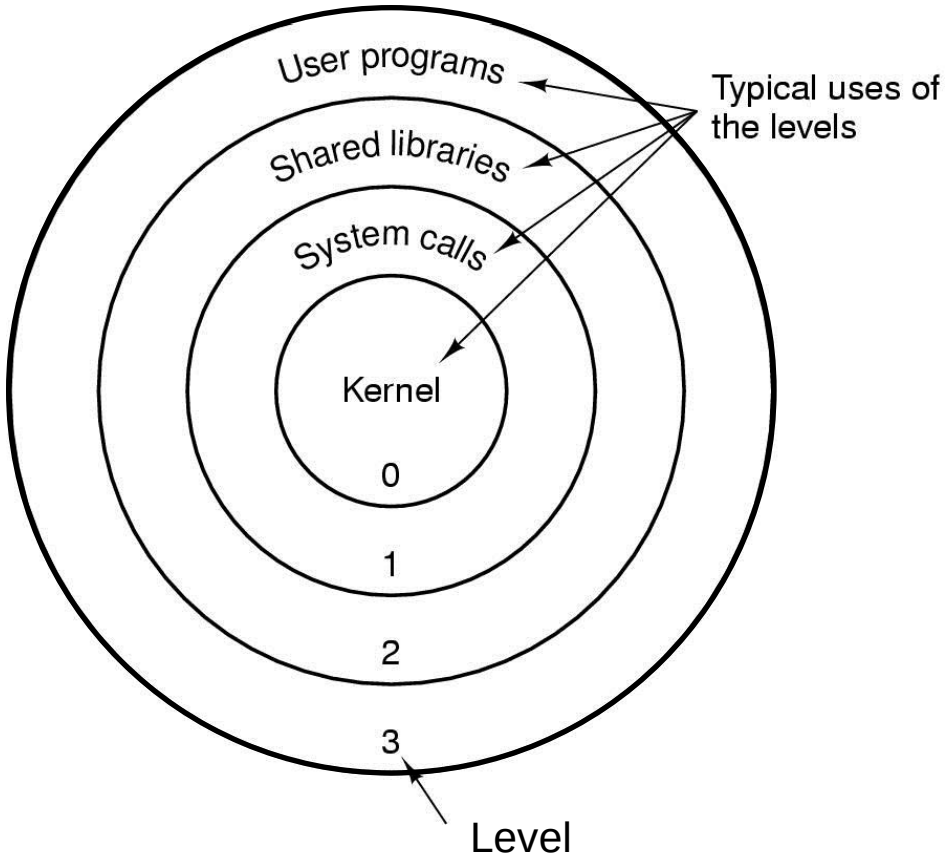
Conversion of a (selector, offset) pair to a linear address

Segmentation with Paging: Pentium (4)



Mapping of a linear address onto a physical address

Segmentation with Paging: Pentium (5)



Protection on the Pentium

Segmentation with Paging: Pentium

- Attempts to access to data at a higher level are permitted.
- Attempts to access to data at a lower level are illegal and cause traps.
- Interlevel call is made via call gate.
- Protection rings
  - Level 0 : kernel (I/O, memory mngmnt, etc.)
  - Level 1 : system call handler
  - Level 2 : shared library
  - Level 3 : user programs