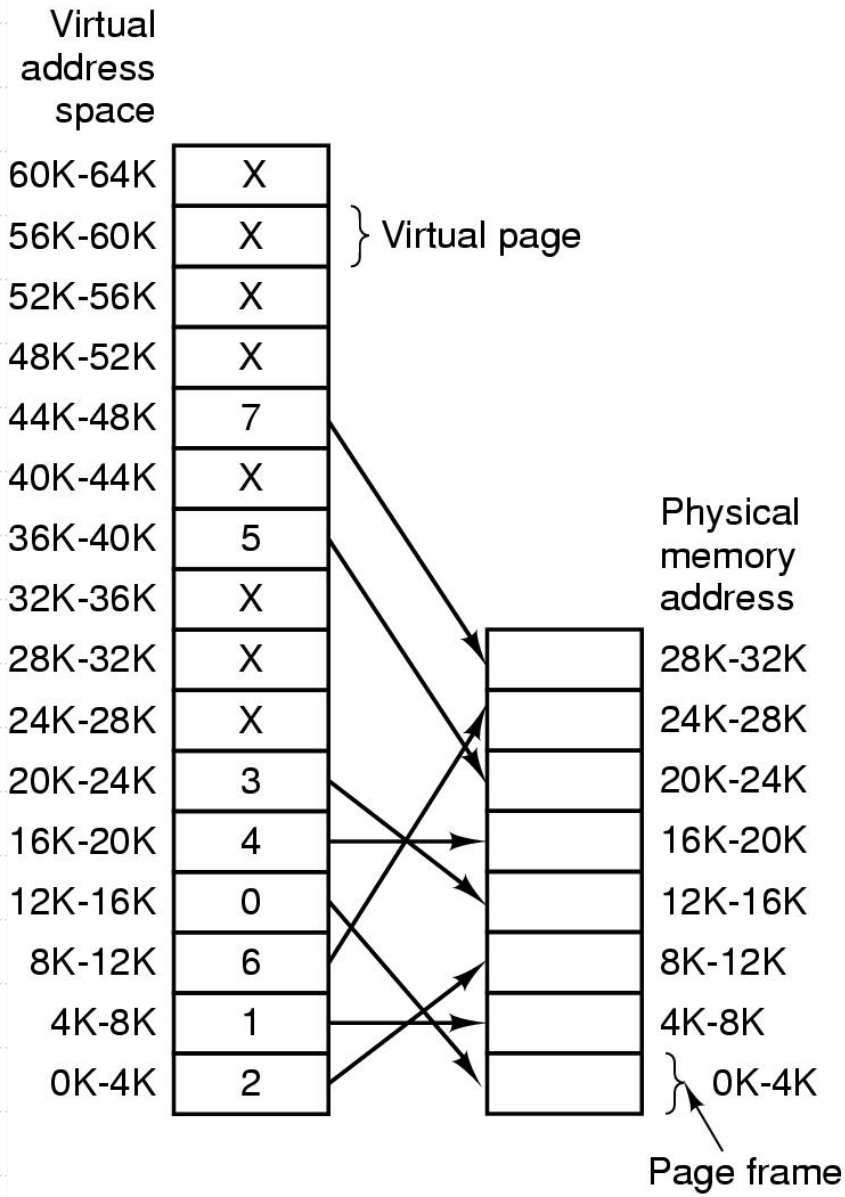
# Virtual Memory Management

B.Ramamurthy

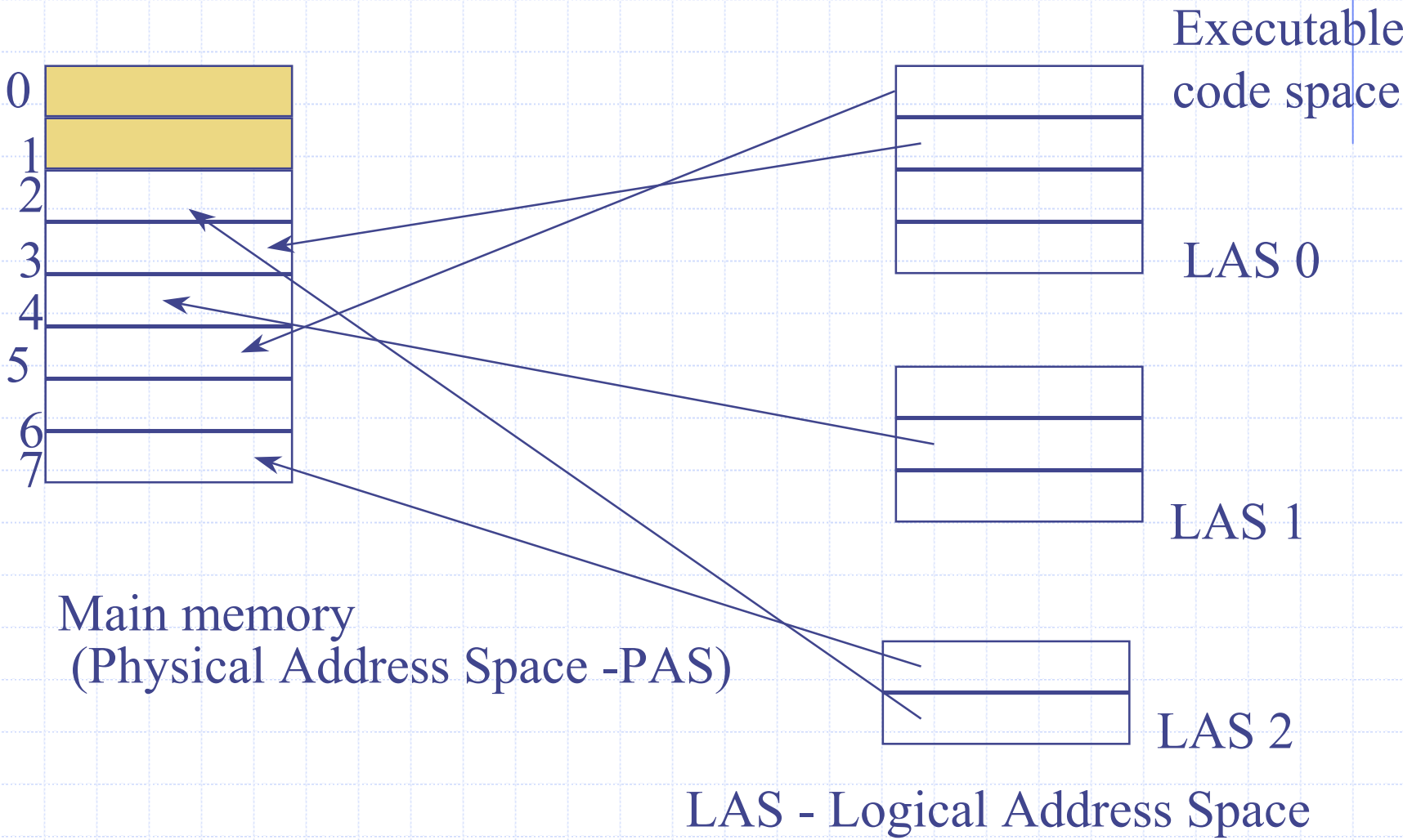
Chapter 10

# Paging

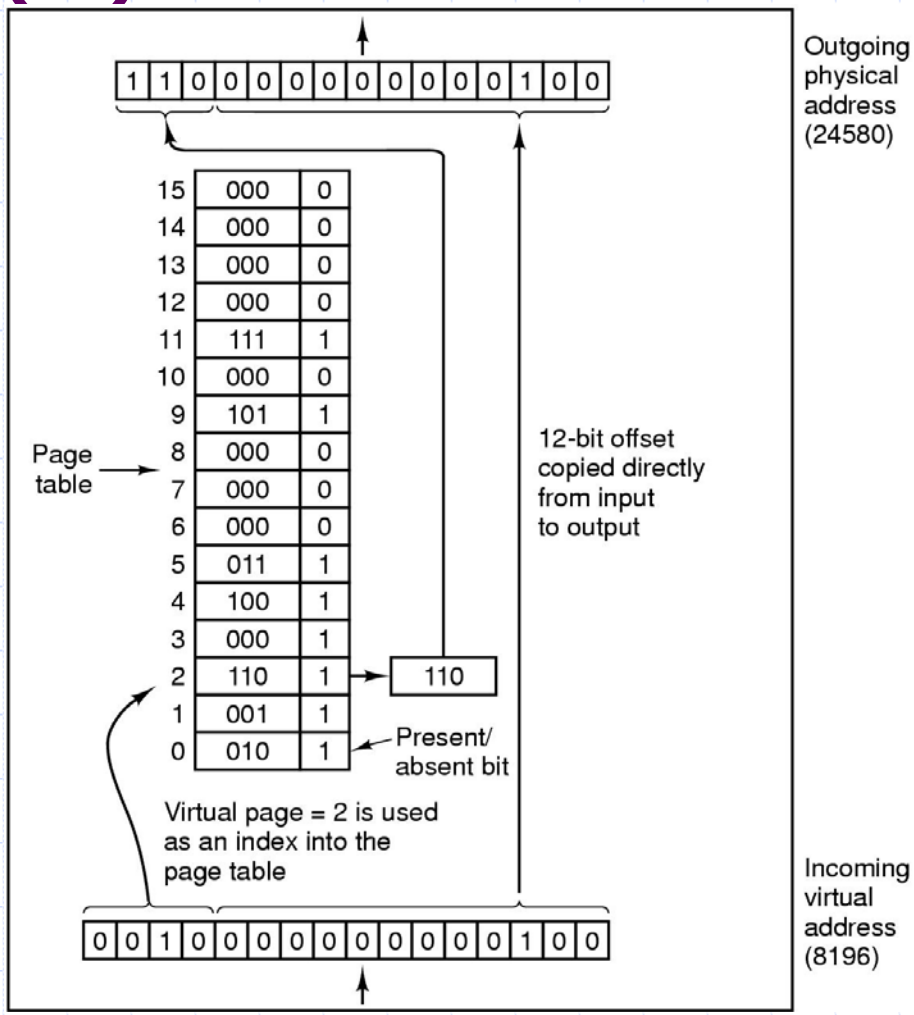
The relation between virtual addresses and physical memory addresses given by page table



# Demand Paging (contd.)

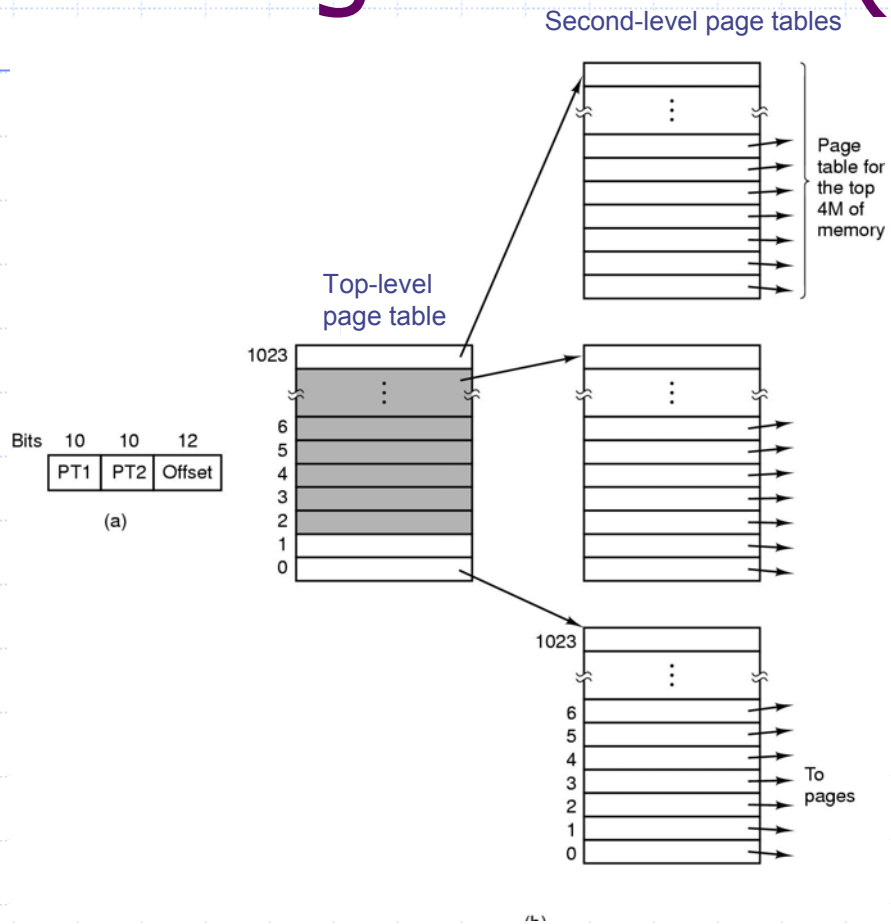


# Page Tables (1)



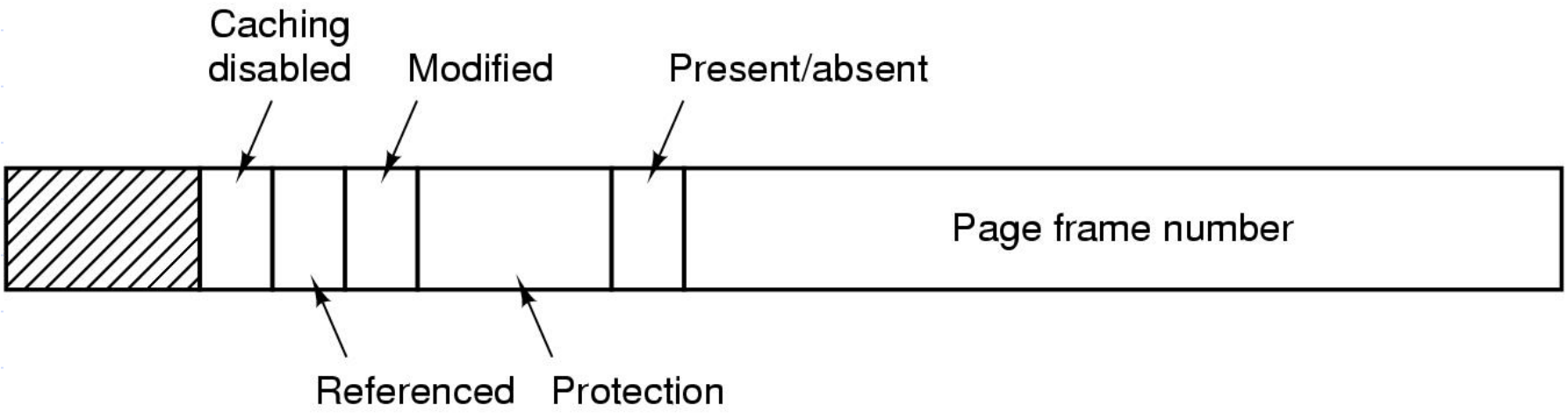

Internal operation of MMU with 16 4 KB pages

# Page Tables (2)




- 32 bit address with 2 page table fields
- Two-level page tables

# Page Tables (3)



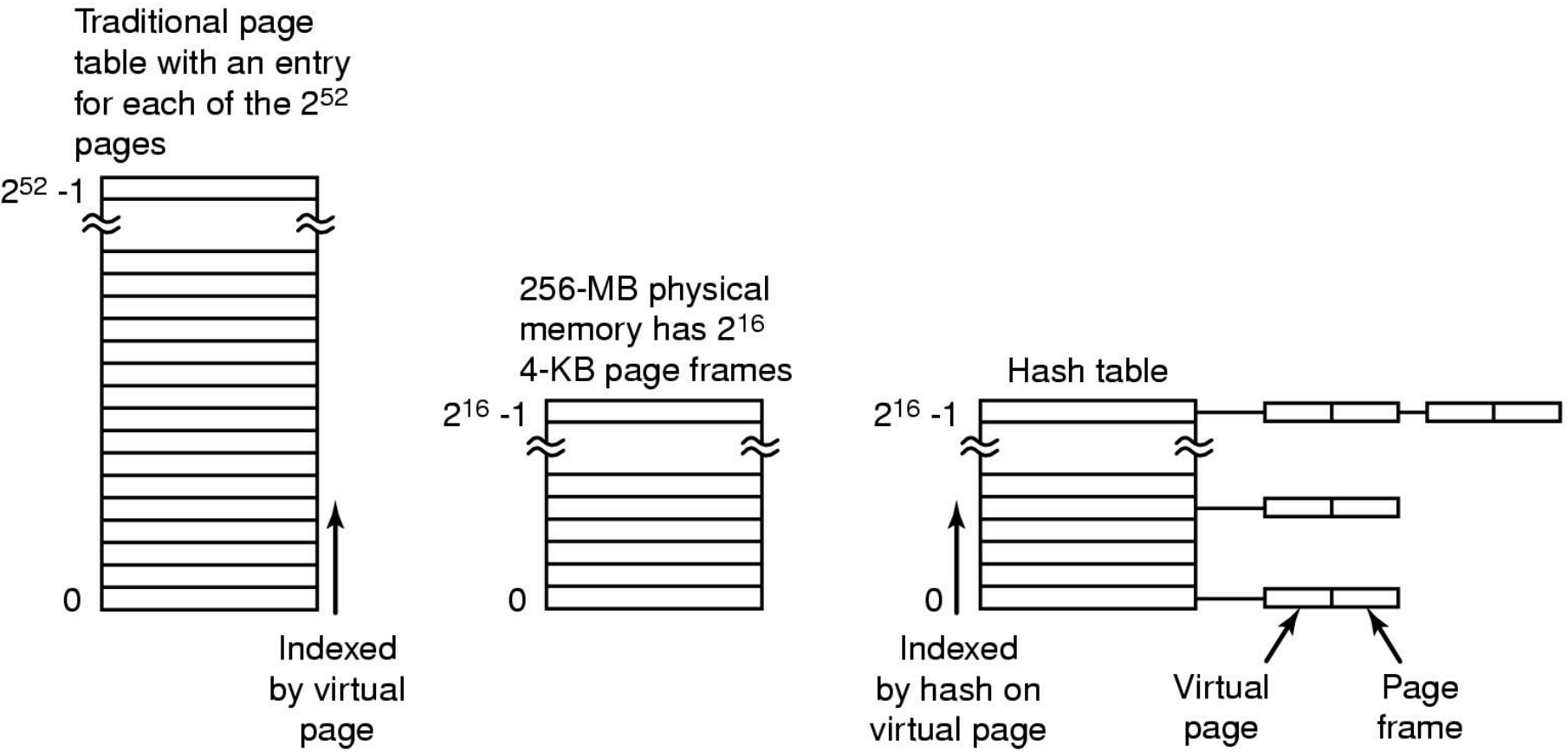

Typical page table entry

# TLBs – 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 |

A TLB to speed up paging

# Inverted Page Tables




Comparison of a traditional page table with an inverted page table

# Page Fault Handling (1)

- Hardware traps to kernel
- General registers saved
- OS determines which virtual page needed
- OS checks validity of address, seeks page frame
- If selected frame is dirty, write it to disk

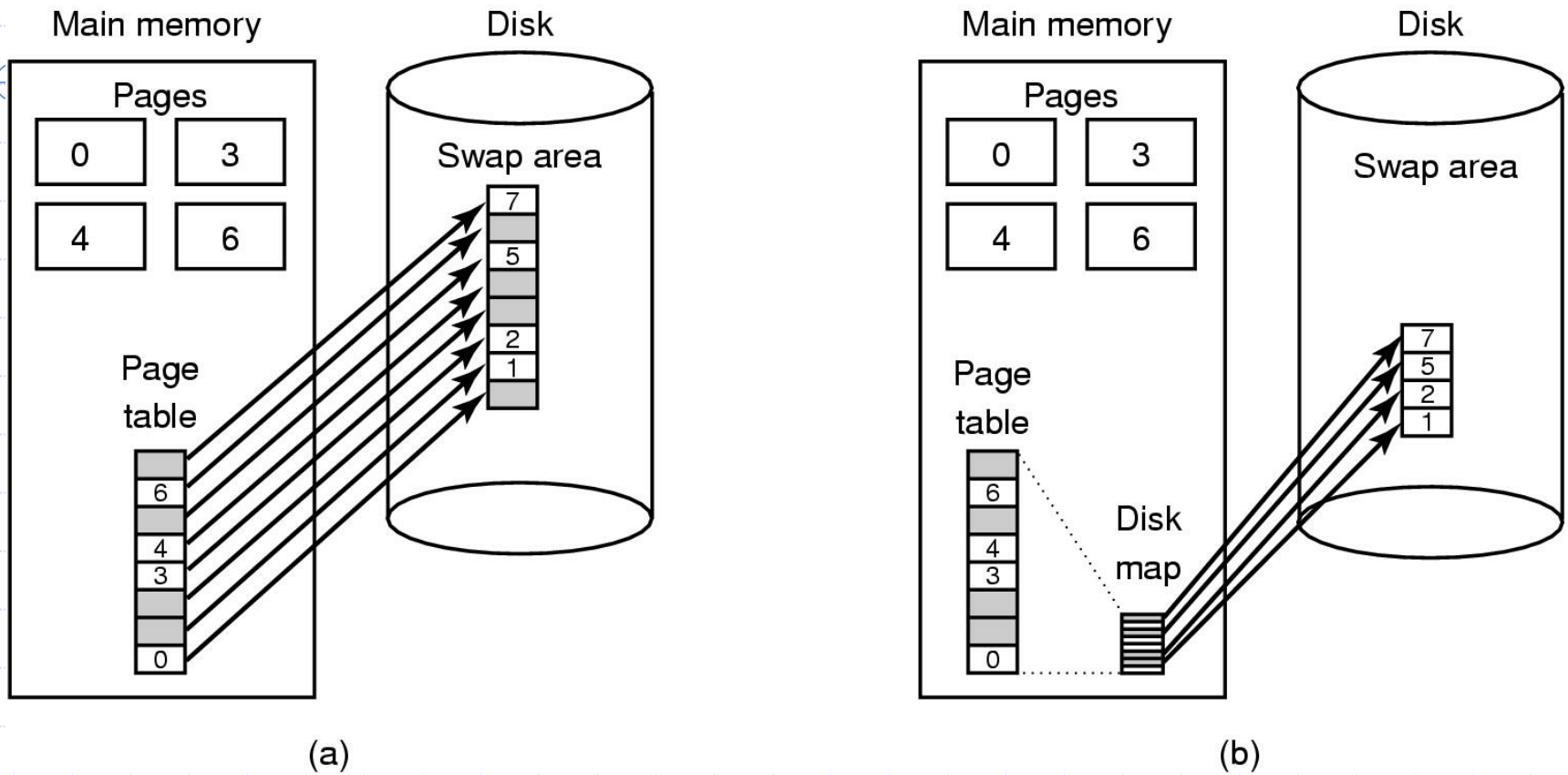
# Page Fault Handling (2)

- OS brings schedules new page in from disk
- Page tables updated
- Faulting instruction backed up to when it began
- Faulting process scheduled
- Registers restored
- Program continues

# Locking Pages in Memory

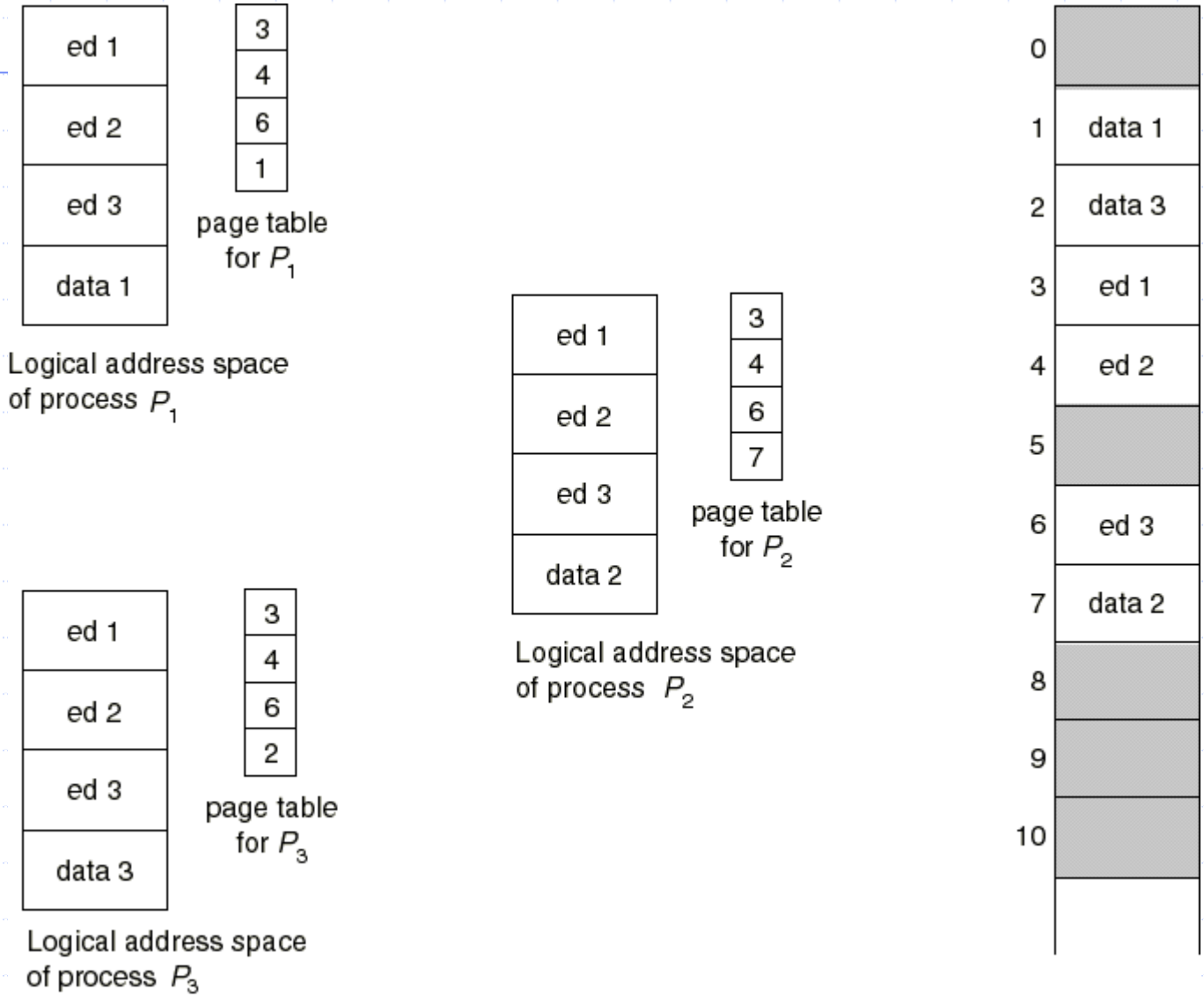
- Virtual memory and I/O occasionally interact
- Proc issues call for read from device into buffer
  - while waiting for I/O, another processes starts up
  - has a page fault
  - buffer for the first proc may be chosen to be paged out
- Need to specify some pages locked
  - exempted from being target pages

# Backing Store




(a) Paging to static swap area
(b) Backing up pages dynamically

# Sharing Pages: a text editor



# Implementation Issues

## Operating System Involvement with Paging

Four times when OS involved with paging

1. Process creation
   - determine program size
   - create page table
2. Process execution
   - MMU reset for new process
   - TLB flushed
3. Page fault time
   - determine virtual address causing fault
   - swap target page out, needed page in
4. Process termination time
   - release page table, pages

# 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

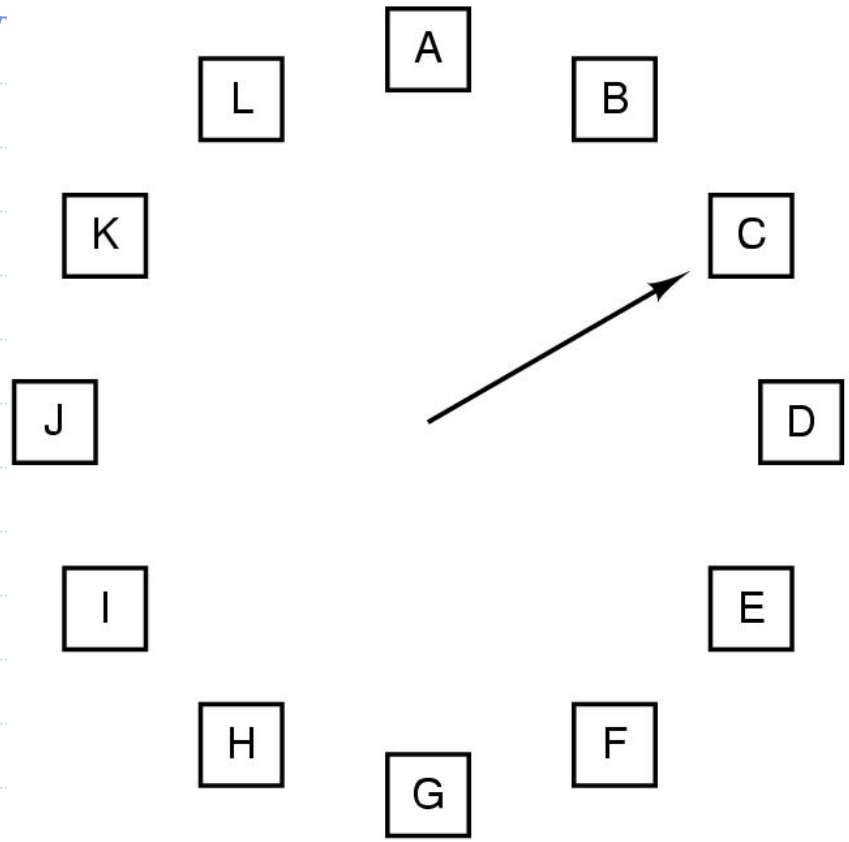
# Not Recently Used Page Replacement Algorithm

- Each page has Reference bit, Modified bit
  - bits are set when page is referenced, modified
- Pages are classified
  1. not referenced, not modified
  2. not referenced, modified
  3. referenced, not modified
  4. referenced, modified
- NRU removes page at random
  - from lowest numbered non empty class

# FIFO Page Replacement Algorithm

- Maintain a linked list of all pages
  - in order they came into memory
- Page at beginning of list replaced
- Disadvantage
  - page in memory the longest may be often used

# 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:

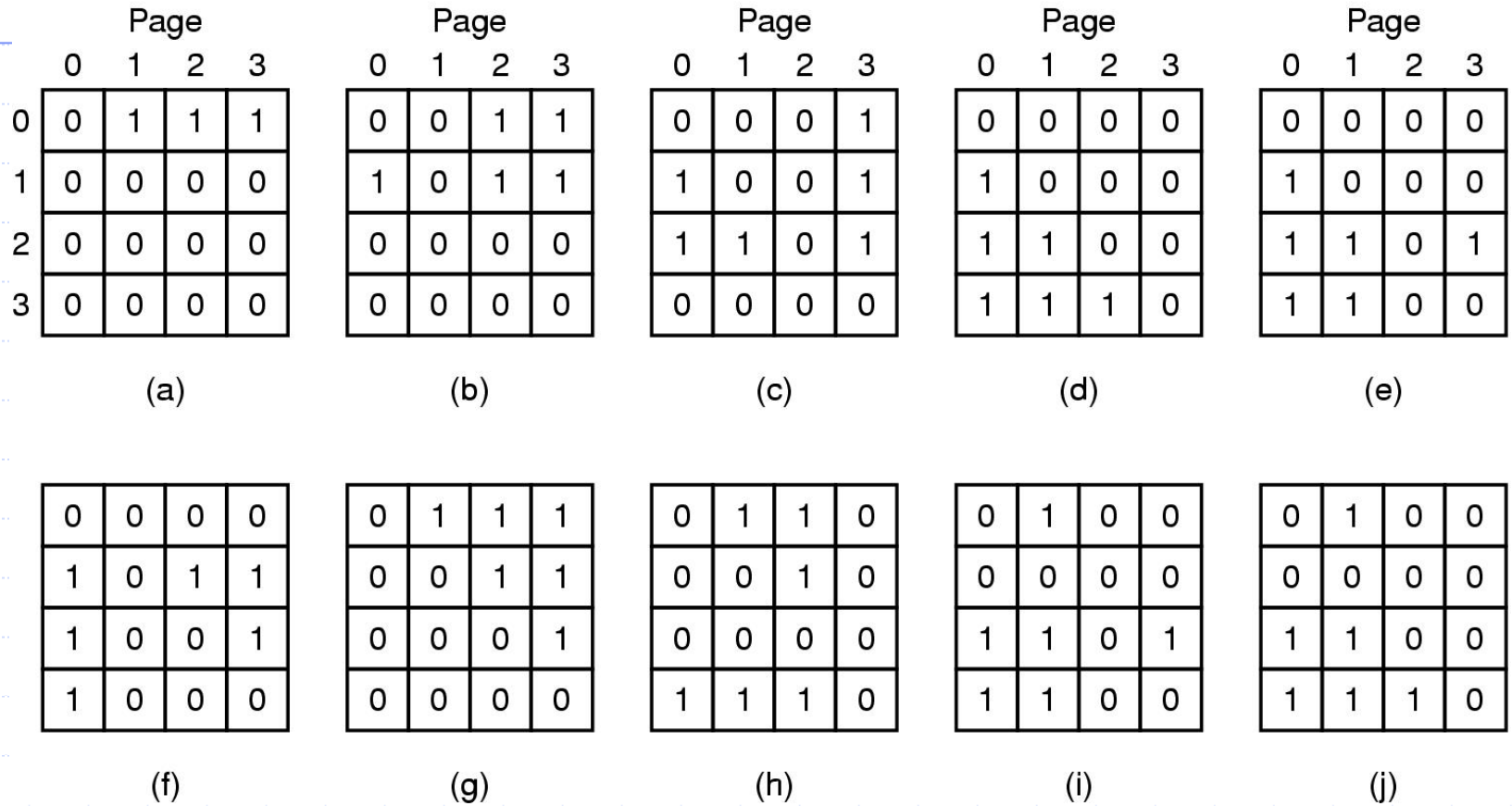
R = 0: Evict the page

R = 1: Clear R and advance hand

# Least Recently Used (LRU)

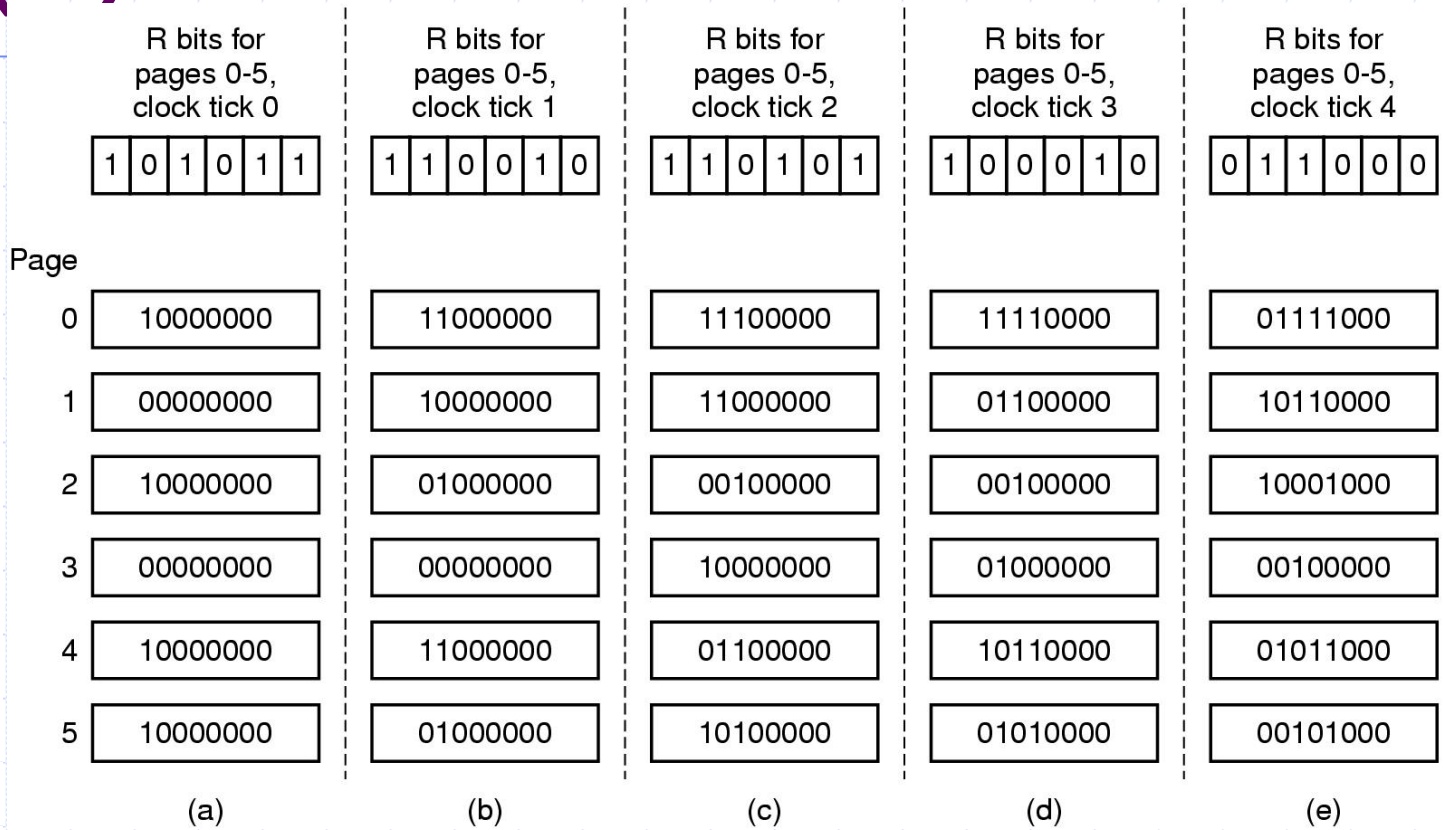
- Assume pages used recently will used again soon
  - throw out page that has been unused for longest time
- Must keep a linked list of pages
  - most recently used at front, least at rear
  - update this list every memory reference !!
- Alternatively keep counter in each page table entry
  - choose page with lowest value counter
  - periodically zero the counter

# Simulating LRU in Software (1)




LRU using a matrix – pages referenced in order 0,1,2,3,2,1,0,3,2,3

# Simulating LRU in Software (2)

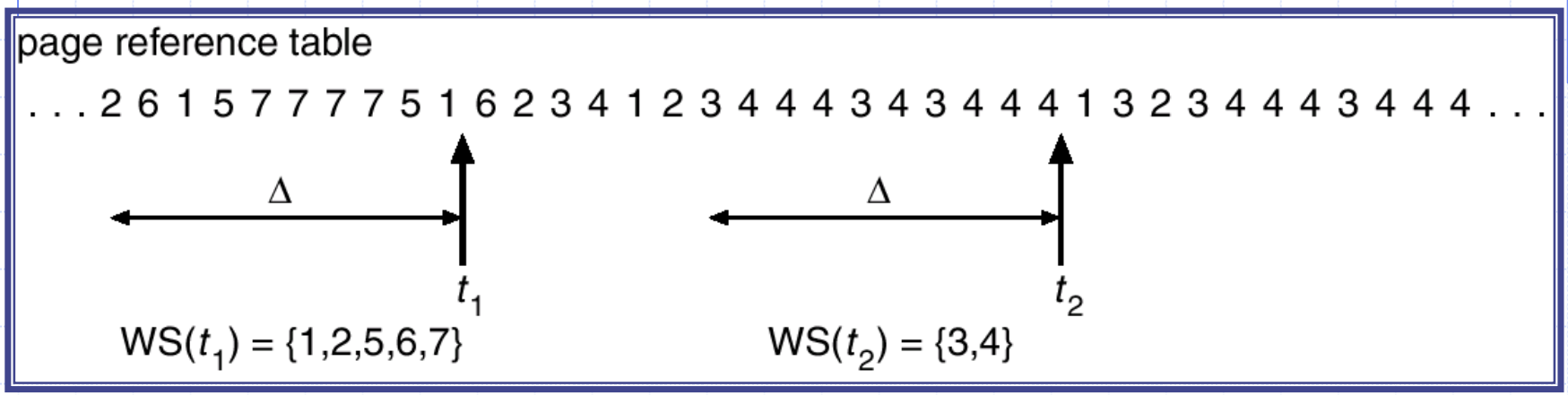



- The aging algorithm simulates LRU in software
- Note 6 pages for 5 clock ticks, (a) – (e)

# Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instruction
- $WSS_i$ (working set of Process $P_i$) =
  total number of pages referenced in the most recent $\Delta$ (varies in time)
  - if $\Delta$ too small will not encompass entire locality.
  - if $\Delta$ too large will encompass several localities.
  - if $\Delta = \infty \Rightarrow$ will encompass entire program.
- $D = \Sigma\ WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if $D >$ m, then suspend one of the processes.

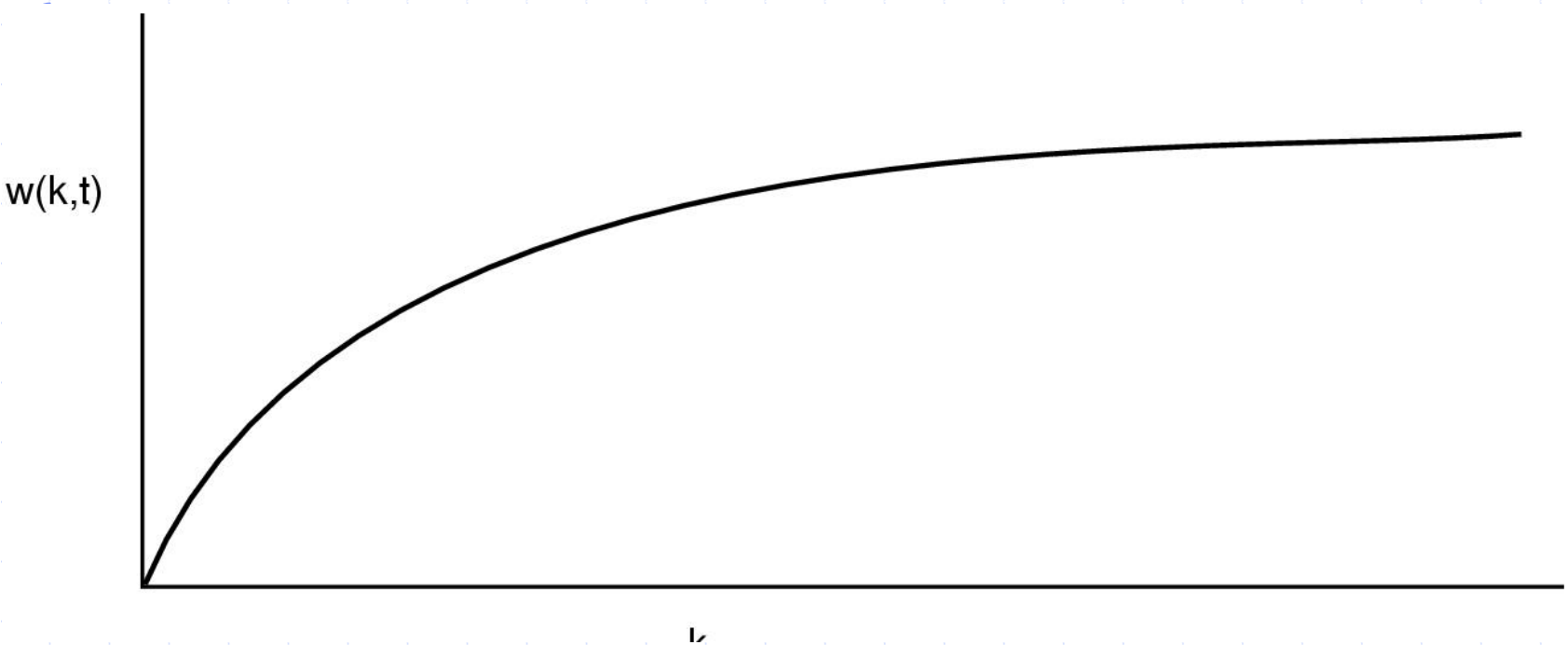
# Working-set model



# Keeping Track of the Working Set

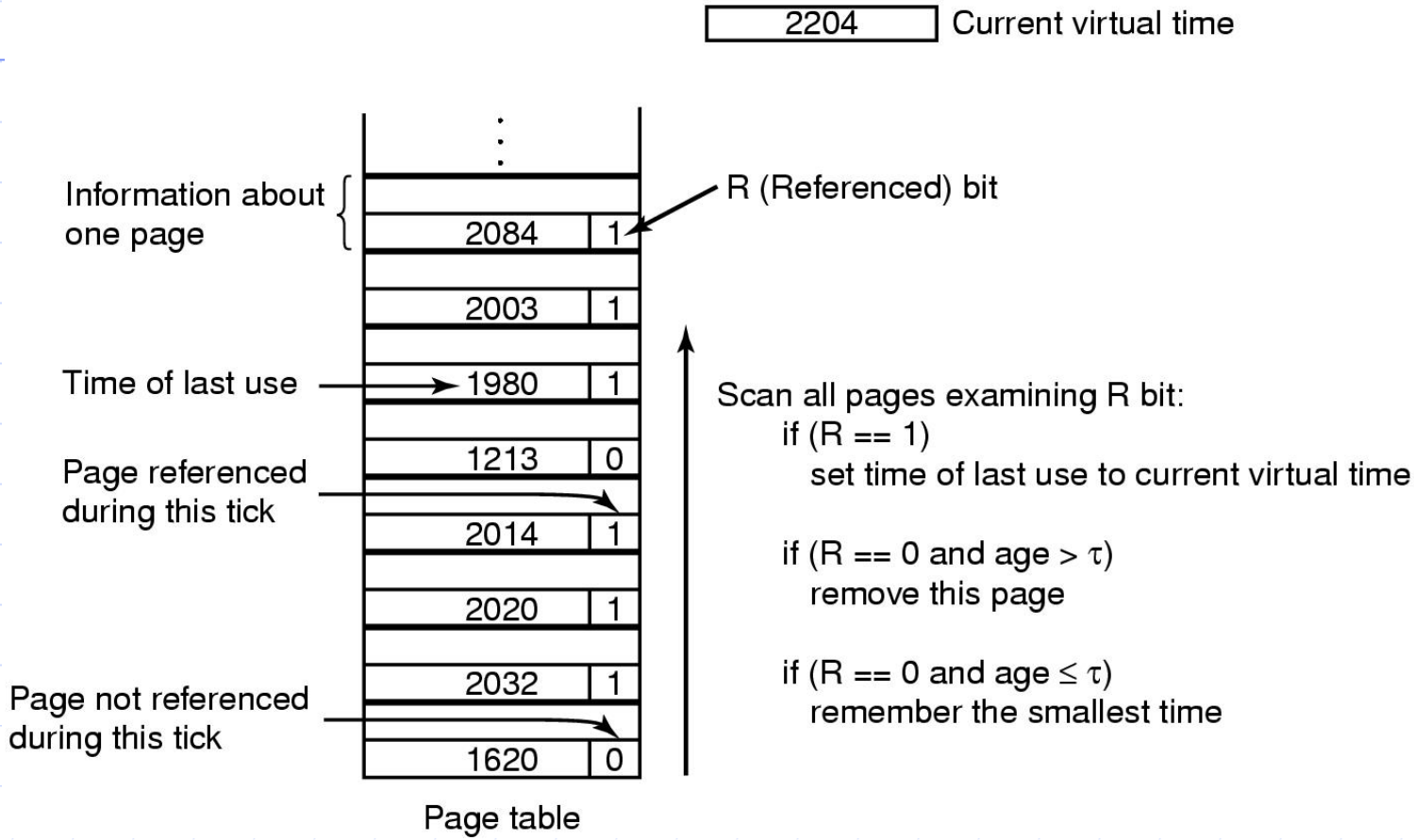
- Approximate with interval timer + a reference bit
- Example: $\Delta$ = 10,000
  - Timer interrupts after every 5000 time units.
  - Keep in memory 2 bits for each page.
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0.
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set.
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units.

# The Working Set Page Replacement Algorithm (1)



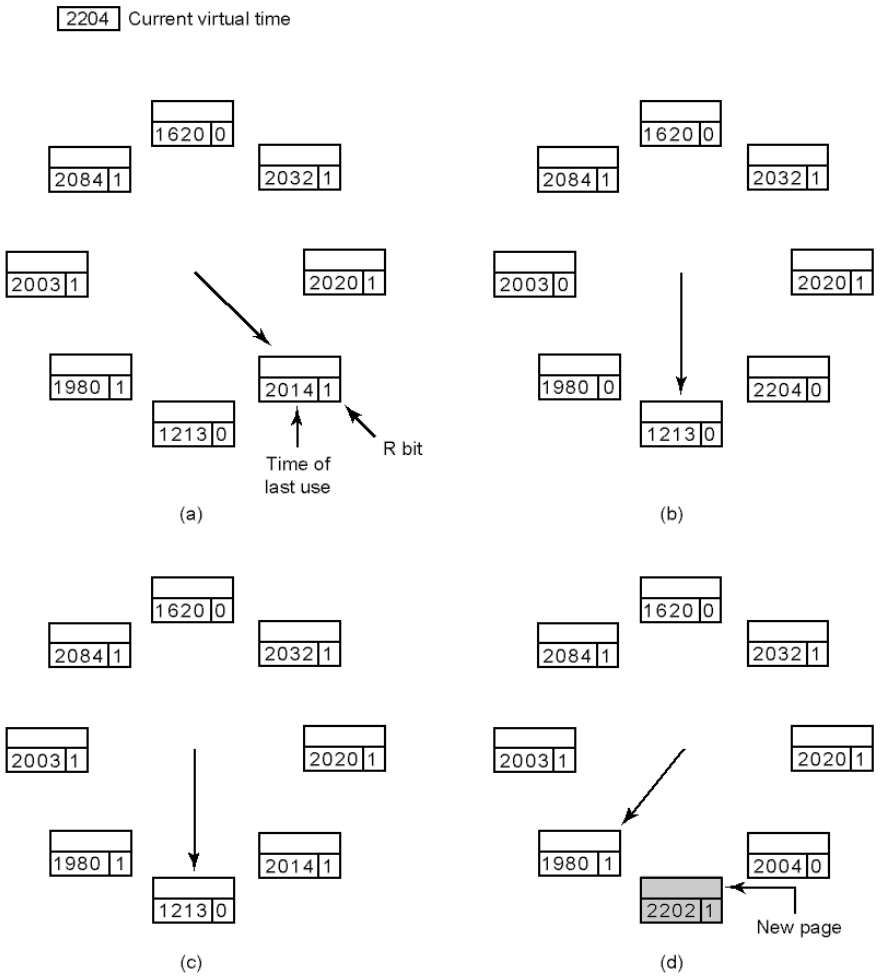

- The working set is the set of pages used by the *k* most recent memory references
- w(k,t) is the size of the working set at time, *t*

# The Working Set Page Replacement Algorithm (2)




The working set algorithm

# 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 |

# Modeling Page Replacement Algorithms Belady's Anomaly

All pages frames initially empty

| | 0 | 1 | 2 | 3 | 0 | 1 | 4 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Youngest page | 0 | 1 | 2 | 3 | 0 | 1 | 4 | 4 | 4 | 2 | 3 | 3 |
| | | 0 | 1 | 2 | 3 | 0 | 1 | 1 | 1 | 4 | 2 | 2 |
| Oldest page | | | 0 | 1 | 2 | 3 | 0 | 0 | 0 | 1 | 4 | 4 |
| | P | P | P | P | P | P | P | | | P | P | |

9 Page faults

(a)

| | 0 | 1 | 2 | 3 | 0 | 1 | 4 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Youngest page | 0 | 1 | 2 | 3 | 3 | 3 | 4 | 0 | 1 | 2 | 3 | 4 |
| | | 0 | 1 | 2 | 2 | 2 | 3 | 4 | 0 | 1 | 2 | 3 |
| Oldest page | | | 0 | 1 | 1 | 1 | 2 | 3 | 4 | 0 | 1 | 2 |
| | | | | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 0 | 1 |
| | P | P | P | P | | | P | P | P | P | P | P |

10 Page faults

(b)

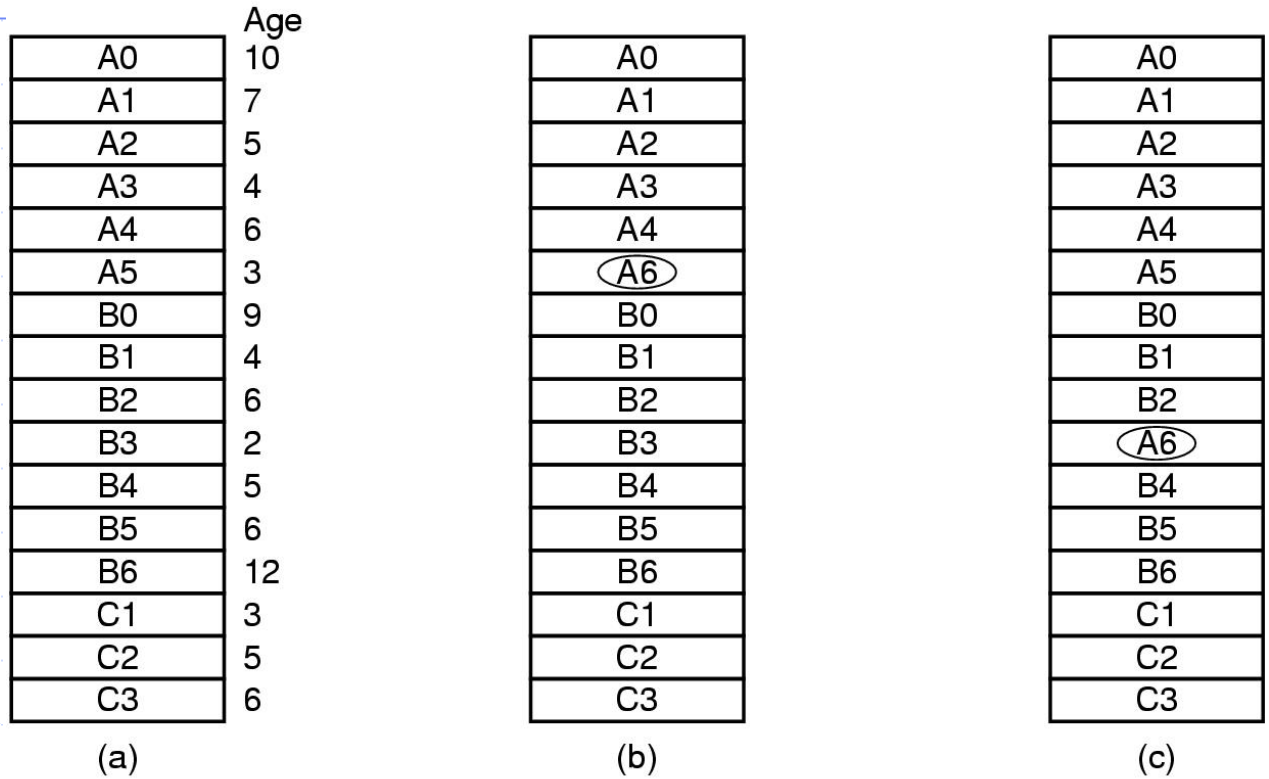
- FIFO with 3 page frames
- FIFO with 4 page frames
- *P*s show which page references show page faults

# Stack Algorithms

| Reference string | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 3 | 3 | 5 | 5 | 3 | 1 | 1 | 1 | 7 | 1 | 3 | 4 | 1 |
| | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 7 | 4 | 7 | 7 | 3 | 3 | 5 | 3 | 3 | 3 | 1 | 7 | 1 | 3 | 4 |
| | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 3 | 3 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 3 | 3 | 7 | 1 | 3 |
| | | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 6 | 6 | 6 | 4 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 7 | 7 |
| | | | | | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 6 | 6 | 6 | 4 | 4 | 4 | 7 | 7 | 7 | 5 | 5 | 5 | 7 |
| | | | | | | 0 | 2 | 1 | 1 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 4 | 4 | 4 | 4 | 4 | 4 | 5 |
| | | | | | | | 0 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| | | | | | | | | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Page faults | P | P | P | P | P | P | P | | P | | | | | P | | | P | | | | | | P | |
| Distance string | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 4 | ∞ | 4 | 2 | 3 | 1 | 5 | 1 | 2 | 6 | 1 | 1 | 4 | 7 | 4 | 6 | 5 |

State of memory array, *M*, after each item in reference string is processed

# Design Issues for Paging Systems
## Local versus Global Allocation Policies (1)



| | Age |
|---|---|
| A0 | 10 |
| A1 | 7 |
| A2 | 5 |
| A3 | 4 |
| A4 | 6 |
| A5 | 3 |
| B0 | 9 |
| B1 | 4 |
| B2 | 6 |
| B3 | 2 |
| 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)

- Original configuration
- Local page replacement
- Global page replacement

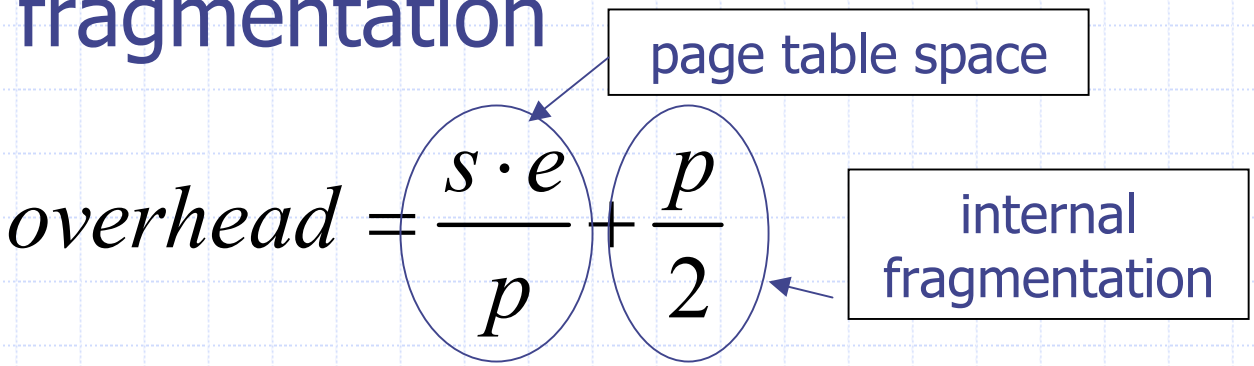
# Page Size (1)

## Small page size

- Advantages
  - less internal fragmentation
  - better fit for various data structures, code sections
  - less unused program in memory
- Disadvantages
  - programs need many pages, larger page tables

# Page Size (2)

- Overhead due to page table and internal fragmentation




- Where
  - s = average process size in bytes
  - p = page size in bytes
  - e = page entry

Optimized when

$$p = \sqrt{2se}$$