# Sistem Operasi

Memori Virtual

2016

UNIVERSITAS Telkom

# Outline

- Paging ← sudah
- Penggantian page ← sudah
- Algoritma penggantian page
- Masalah-masalah Perancangan dan Implementasi
- Contoh Manajemen Memori

# Virtual Memory

- A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory.
- The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses.
- The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of secondary memory available and not by the actual number of main storage locations.

# Algoritma Penggantian Page

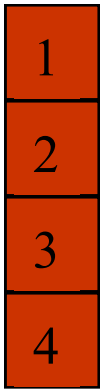
- Kegagalan page memaksa pilihan
  - page mana yang harus dibuang
  - memberi ruang untuk page yang datang
- Diinginkan laju *kegagalan page* terendah
  - lebih baik tidak memilih sebuah page yang sering digunakan
  - Mungkin akan diperlukan lagi segera
- Evaluasi dengan menjalankannya melalui urutan referensi memori tertentu dan menghitung jumlah kegagalan page
  - Contoh: 1,2,3,4,1,2,5,1,2,3,4,5

# Optimal

- Ganti page yang tidak **akan** digunakan pada perioda waktu yang lama

1,2,3,4,2,1,5,1,2,3,4,5

4 Frames / Process




4

5

6 Kegagalan Page

Darimana kita tahu?
Menggunakan *benchmark*

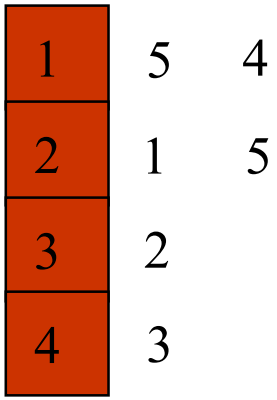
# Algoritma Penggantian Page FIFO

- Memelihara sebuah list hubungan semua page
  - Dengan urutan kedatangan ke memori
- Page diawal list diganti
- Kekurangan
  - page terlama di memori mungkin sering digunakan

# First-In-First-Out (FIFO)

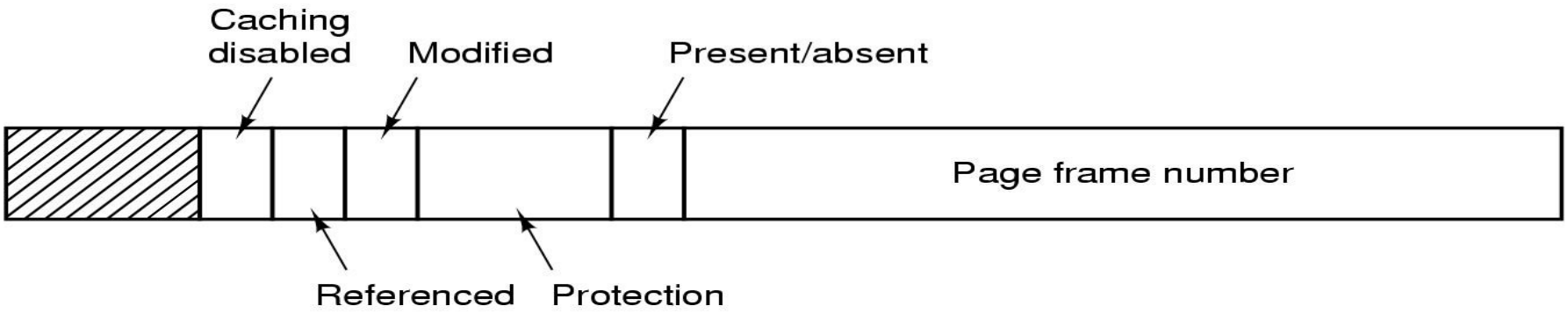
1,2,3,4,2,1,5,1,2,3,4,5

4 Frames / Process




10 Kegagalan Page

# Tabel Page

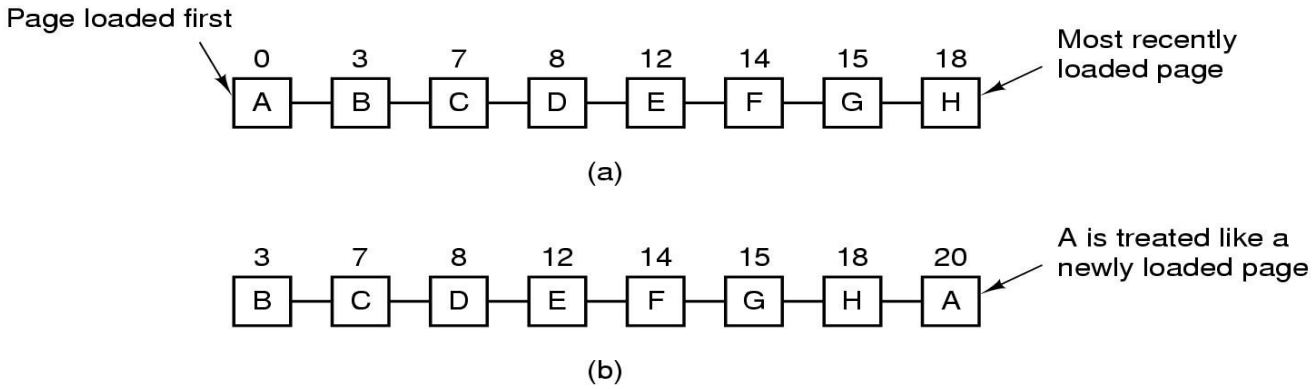



Entri tabel page umum

# Kesempatan-Kedua

- Penggantian FIFO, tapi …
  - Dapatkan yang pertama di FIFO
  - Lihat bit referensinya
    - bit == 0 maka ganti
    - bit == 1 maka set bit = 0, dapatkan yang selanjutnya di FIFO
- Jika referensi page mencukupi, jangan pernah diganti

# Algoritma Penggantian Page Kesempatan Kedua

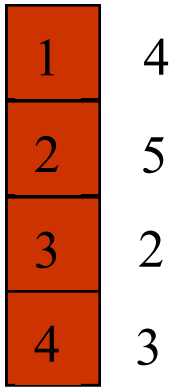



- Operasi dari sebuah kesempatan kedua
  - Page diurut di urutan FIFO
  - Daftarkan page jika terjadi kegagalan di waktu 20, *A* mempunyai bit *R* ter-set
    (angka diatas page adalah waktu loading)

# Kesempatan Kedua

1,2,3,4,2,1,5,1,2,3,4,5

4 Frames / Process




9 Kegagalan Page

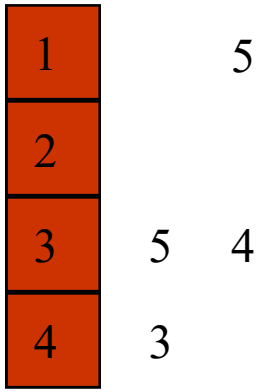
# Least Recently Used (LRU)

- Asumsi page yang baru digunakan akan digunakan kembali segera
  - buang page yang paling lama tidak digunakan
- Harus menyimpan daftar link (atau stack) page
  - yang baru digunakan didepan, yang paling akhir dibelakang
  - update list setiap referensi memori !!
- Alternatif: simpan waktu penggunaan (counter) di setiap entri tabel page
  - pilih page dengan nilai terendah
  - secara periodik set ke zero

# Least Recently Used

- Ganti page yang sudah **lama** tidak digunakan

1,2,3,4,2,1,5,1,2,3,4,5



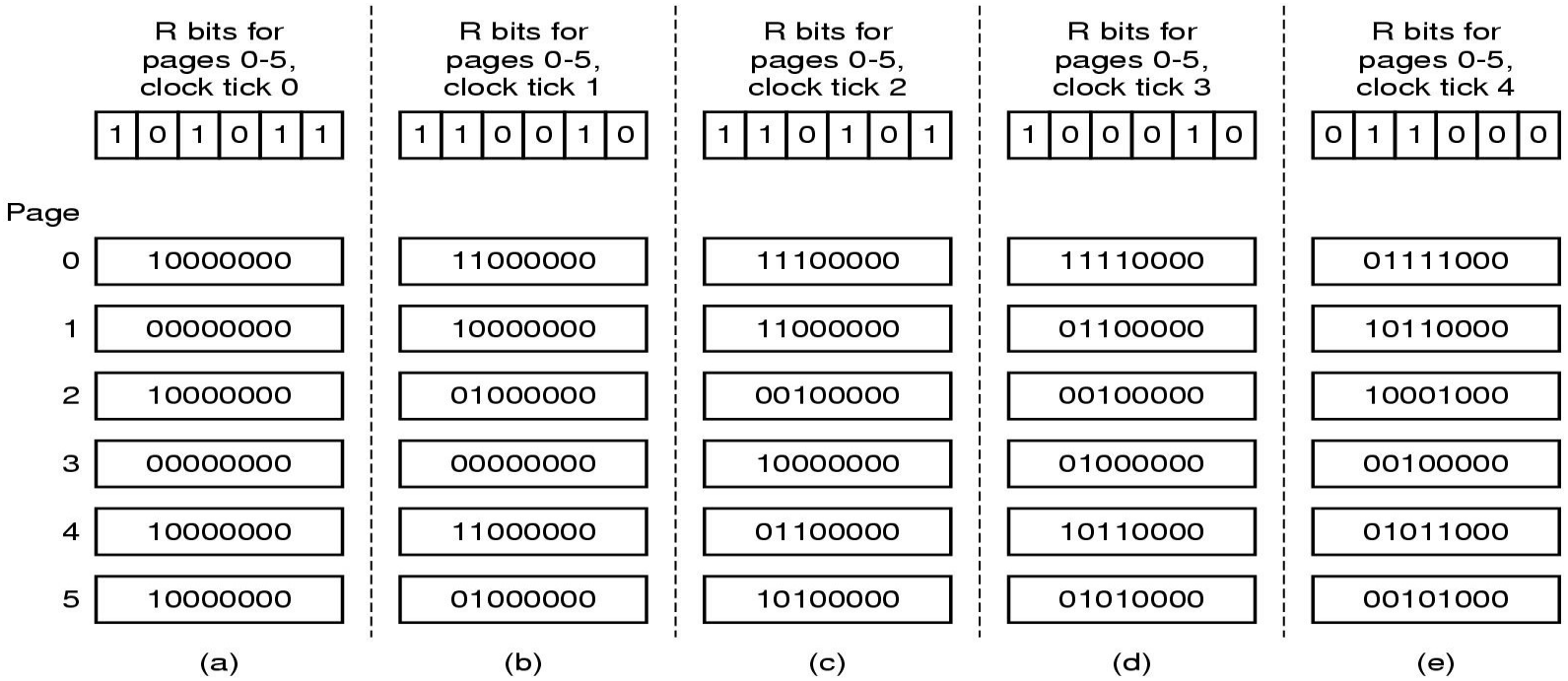

8 Kegagalan Page

# Simulasi LRU di Software (1)

- NFU (Not Frequently Used)
- Sebuah counter software berasosiasi dengan setiap page
  - Initial:  zero
  - Setiap interrupt clock, R ditambahkan ke counter
  - Korban adalah page dengan counter terendah
- Masalah utama?
  - Tidak boleh lupa.
  - Count dengan nilai tinggi akan sangat lama ada

# Simulasi LRU di Software (2)

- Disebut juga algoritma Aging
- Simban sebuah 8-bit byte untuk setiap page
- OS mengambil kendali pada interval teratur
  - Shift  8-bit byte ke kanan 1-bit
  - Bit referensi ➔ bit MSB dari 8-bit byte
  - Baca 8-bit byte tersebut sebagai integer
  - Pilih page dengan nomor terendah sebagai korban
- Catatan:  6 page untuk 5 clock tick, (a) – (e)
- Jika hanya satu bit digunakan selain8-bit byte, maka menjadi **Kesempatan-Kedua**


R bits for pages 0-5, clock tick 0
1 0 1 0 1 1
R bits for pages 0-5, clock tick 1
1 1 0 0 1 0
R bits for pages 0-5, clock tick 2
1 1 0 1 0 1
R bits for pages 0-5, clock tick 3
1 0 0 0 1 0
R bits for pages 0-5, clock tick 4
0 1 1 0 0 0
Page
0 10000000 11000000 11100000 11110000 01111000
1 00000000 10000000 11000000 01100000 10110000
2 10000000 01000000 00100000 00100000 10001000
3 00000000 00000000 10000000 01000000 00100000
4 10000000 11000000 01100000 10110000 01011000
5 10000000 01000000 10100000 01010000 00101000
(a)
(b)
(c)
(d)
(e)

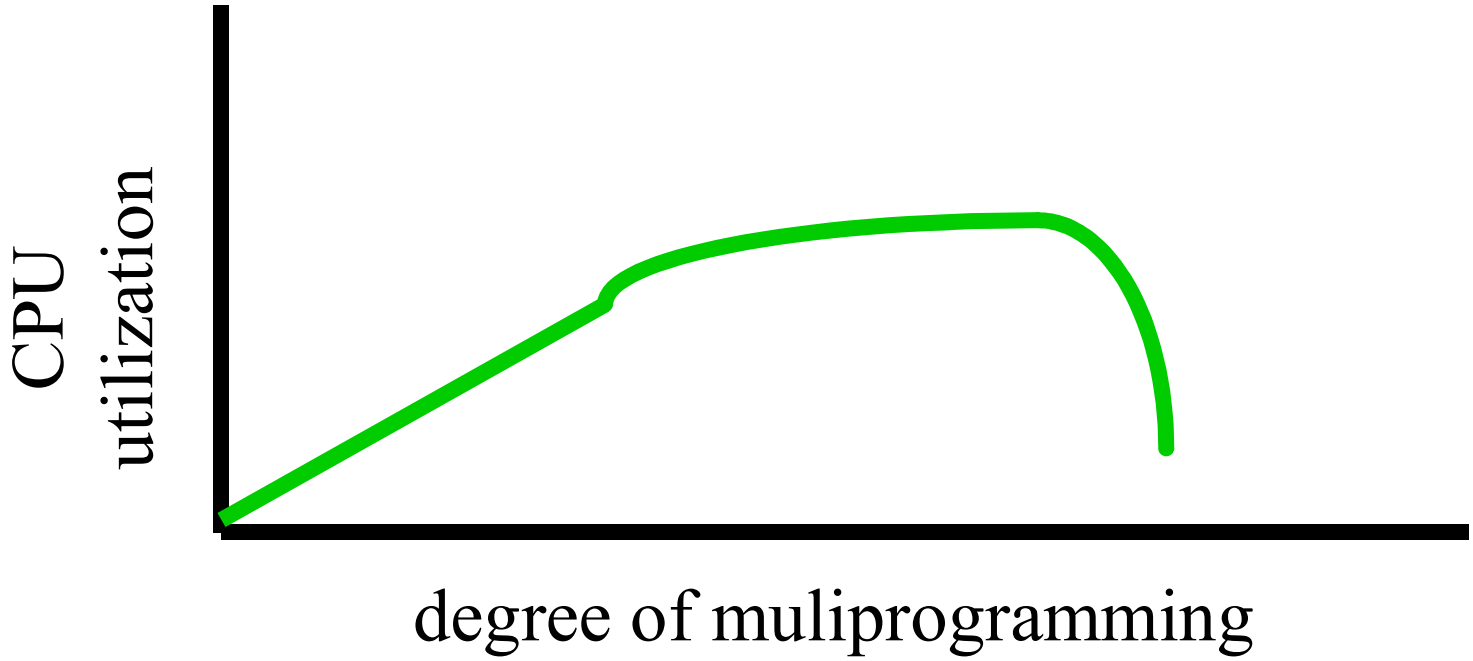
# Kelebihan dan Kekurangan Aging

- Kelebihan Aging
  - LRU tidak menyimpan sejarahnya
  - Perhatikan page 3 dan page 5
  - Menurut LRU,
    - Tidak satupun telah direferensikan ke dua clock tick
    - Keduanya direferensikan di tick sebelumnya
    - Tidak ada bedanya!
  - Menurut Aging
    - Page 5 telah direferensikan antara tick 0 dan 1
    - Pilih page 3 sebagai korban!
- Kekurangan
  - Aging mempunyai jumlah bit terbatas.
  - 8 bit hanya dapat merekam event di 8 tick sebelumnya

# Thrashing

- Jika sebuah proses tidak mempunyai page ‘cukup’, tingkat kegagalan-page sangat tinggi
  - utilitas CPU rendah
  - OS mengira perlu peningkatan multiprogramming
  - menambah proses lain ke sistem
- *Thrashing* adalah ketika sebuah proses sibuk men-swap pages masuk dan keluar

# Thrashing



# Penyebab Thrashing

- Kenapa paging berhasil?
  - Model Lokal
    - referensi proses adalah bagian yang relatif kecil
    - Proses berpindah dari satu lokal ke lainnya
    - lokalitas dapat overlap
- Kenapa terjadi thrashing?
  - Jumlah lokalitas > ukuran memori total
- Cara menghindari thrashing?
  - *Working Set Model*
  - *Page Fault Frequency*

# Allocation - Scope

| | Local Replacement | Global Replacement |
|---|---|---|
| **Fixed Allocation** | • Number of frames allocated to process is fixed.<br>• Page to be replaced is chosen from among the frames allocated to that process. | • Not possible. |
| **Variable Allocation** | • The number of frames allocated to a process may be changed from time to time, to maintain the working set of the process.<br>• Page to be replaced is chosen from among the frames allocated to that process. | • Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary. |

# Working-Set Model

- Sebuah aproksimasi dari lokalitas program
- Teliti referensi page terbaru Δ
- set dari page itu adalah working set
- Buat Δ = 5
- 1 2 3 2 3 1 2 4 3 4 7 4 3 3 4 1 1 2 2 2 1

WS={1,2,3}  WS={3,4,7}  WS={1,2}

  - jika Δ terlampau kecil, tidak akan meliputi lokalitas
  - jika Δ terlampau besar, akan meliputi beberapa lokalitas
  - jika Δ => tak terhingga, akan meliputi seluruh program

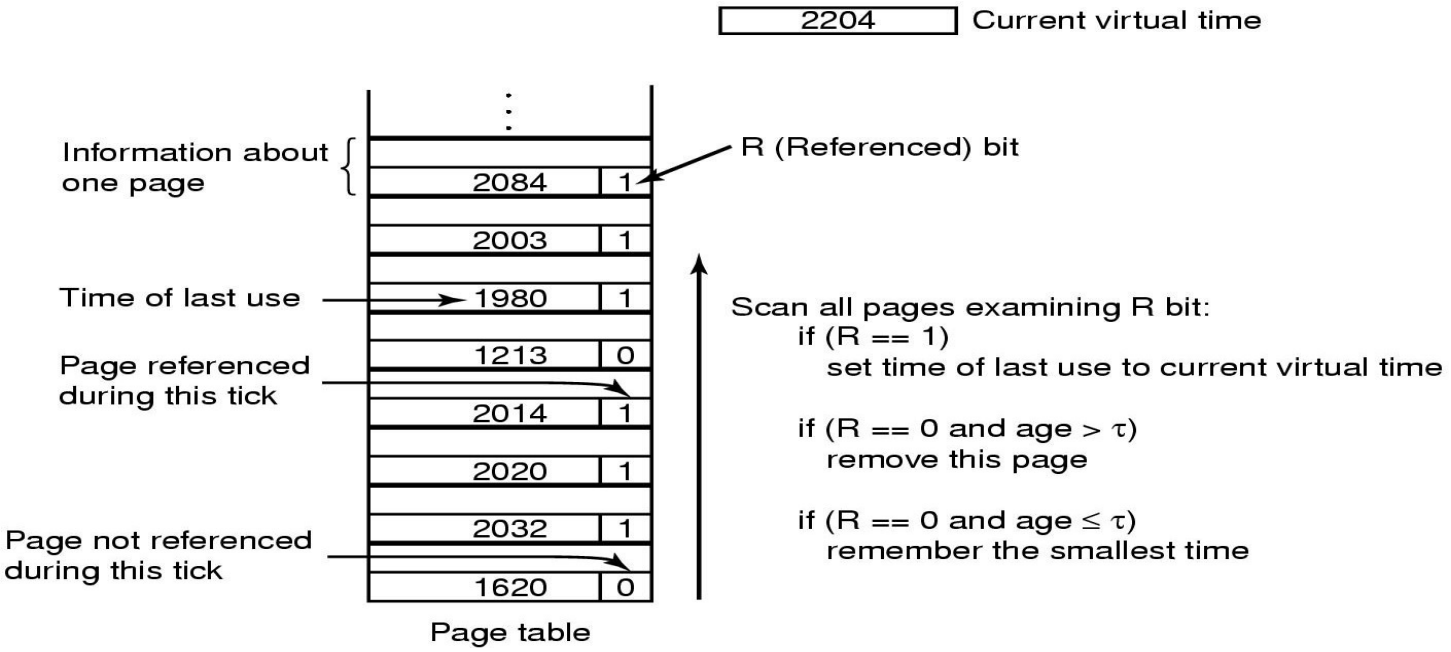
# Working Set

- Ukuran working set (WSS) adalah jumlah page berbeda di working set itu
- Total adalah jumlah ukuran dari setiap working set.
- m adalah nomor frame total number di memori fisik
- jika *Total* > *m* ➔ thrashing, sehingga men-suspend sebuah process
- Ubah LRU untuk memasukkan Working Set

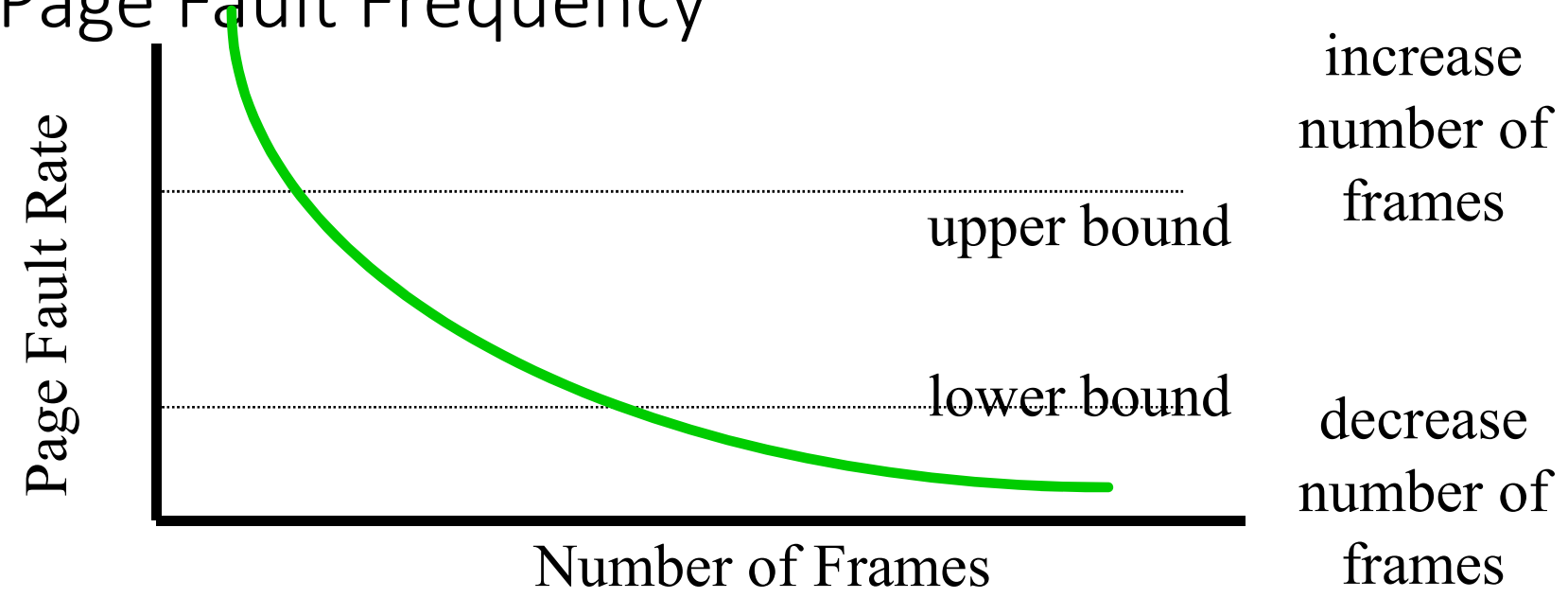
# Algoritma Penggantian Page Working Set (1)

- Page table supports R and virtual time fields.
- The hardware is assumed to set the R bits
- A periodic clock causes to clear the R bit every clock tick
- For each page entry, R bit is examined
  - 1: update virtual time to current time
  - 0 and age > t: remove
  - 0 and age <= t; remember the one has oldest age
- If no victim after scanning all entries
  - Remove the one has oldest age.

# Algoritma Penggantian Page Working Set (2)



# Page Fault Frequency




- Establish "acceptable" page-fault rate
  - If rate too low, process loses frame
  - If rate too high, process gains frame

# Load Control

- Despite good designs, system may still thrash

- When PFF algorithm indicates
  - some processes need more memory
  - but no processes need less

- Solution :
  Reduce number of processes competing for memory
  - swap one or more to disk, divide up pages they held
  - reconsider degree of multiprogramming

# Outline

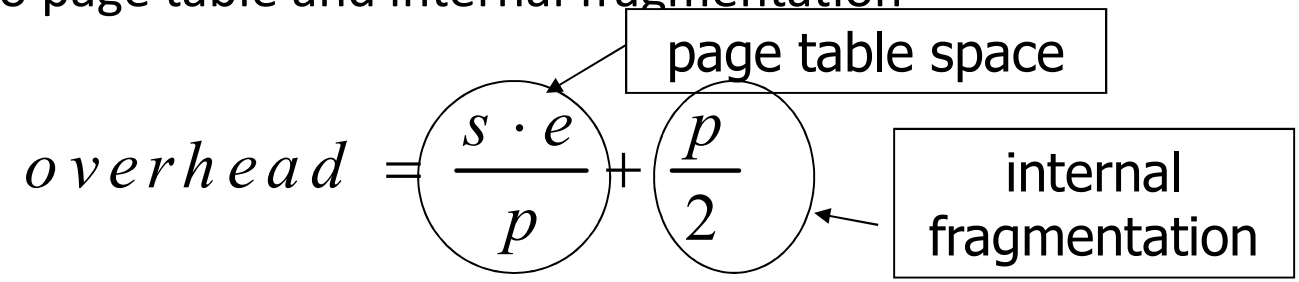
- Paging (Done)
- Page replacement (Done)
- Page replacement algorithms (Done)
- Design and Implementation issues ←
- MM Examples

# Page Size (1)

- Small page size's 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

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

- Historical trend towards larger page sizes

# Cleaning Policy

- Paging works best when there are plenty of free page frames that can be claimed
- Need for a background process, paging daemon
  - periodically inspects state of memory
- When too few frames are free
  - selects pages to evict using a replacement algorithm
- It can use same circular list (clock)
  - as regular page replacement algorithm but with different pointers
    - One maintains a queue of clean pages
    - Another is used for page replacement

# 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 Fault Handling (1)

1. Hardware traps to kernel
2. General registers saved
3. OS determines which virtual page needed
4. OS checks validity of address, seeks page frame
5. If selected frame is dirty, write it to disk

# Page Fault Handling (2)

6. OS brings schedules new page in from disk
7. Page tables updated
8. Faulting instruction backed up to when it began
9. Faulting process scheduled
10. Registers restored
11. 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 victim
- Real-time processes

# Outline

- Paging (Done)
- Page replacement (Done)
- Page replacement algorithms (Done)
- Design and Implementation issues (Done)
- MM Examples ←

# Memory Management in Windows

- 32 bit addresses ($2^{32}$ = 4 GB address space)
  - Upper 2GB shared by all processes (kernel mode)
  - Lower 2GB private per process
- Page size is 4 KB ($2^{12}$, so offset is 12 bits)
- Multilevel paging (2 levels)
  - 10 bits for outer page table (page directory)
  - 10 bits for inner page table
  - 12 bits for offset

# Memory Management in Windows

- Each page-table entry has 32 bits
  - only 20 needed for address translation
  - 12 bits “left-over”
- Characteristics
  - Access: read only, read-write
  - States: valid, zeroed, free …
- Page Replacement Algorithm
  - Working set
  - default is 30
  - take *victim* frame periodically
  - if no fault, reduce set size by 1

# Memory Management in Linux

- 32 bit addresses ($2^{32}$ = 4 GB address space)
  - Upper 3GB user space
  - Lower 1GB kernal
- Page size:
  - Alpha AXP has 8 Kbyte page
  - Intel x86 has 4 Kbyte page
- Multilevel paging (3 levels)
  - “middle-layer” defined to be 1

# Memory Management in Linux

- Re-Examine fork() and exec()
  - exec() creates new page table
  - fork() copies page table
- Page Replacement Algorithm
  - look in reserve pool for free frames
  - enhanced second chance (with more bits)
    - “dirty” pages not taken first