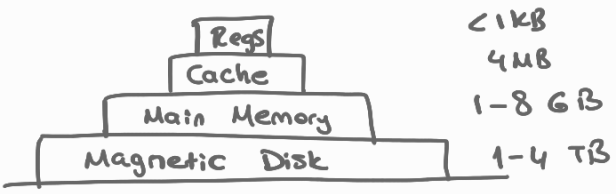
MEMORY MANAGEMENT

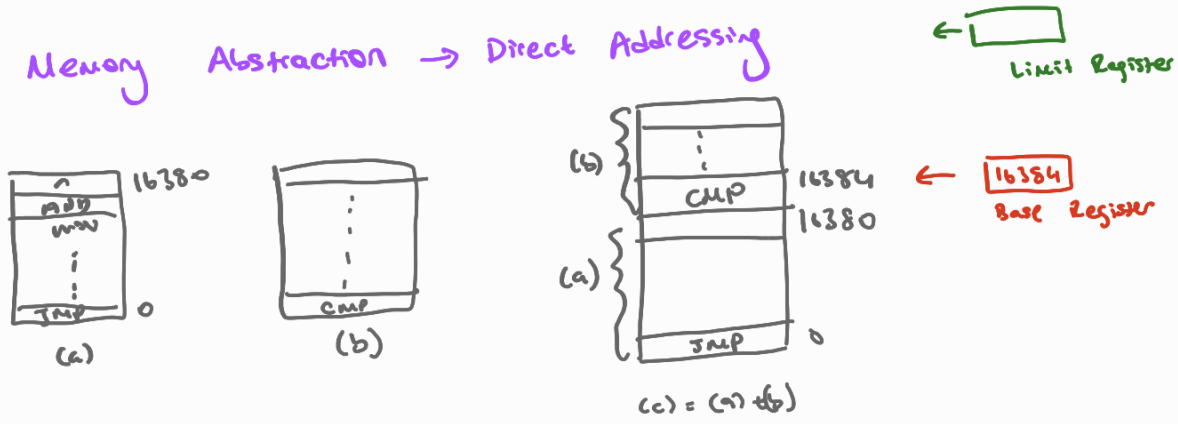
- No Memory Abstraction
- Swapping
- Fixed Partioning
- Fragmentation
- Variable Partioning
- First Fit
- Best Fit
- Worst Fit
- Virtual Memory
- Paging
- Sharing Pages

- Page Replacement Algorithms
  - Optimal Algorithm
- Trashing
- Working Set Model

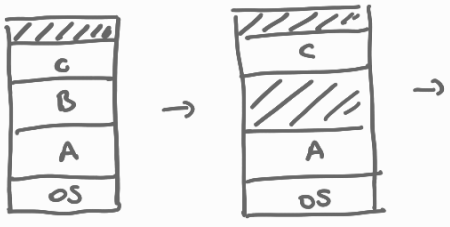
Reference: Uğur Halıcı BE442 Lecture Notes



No Memory Abstraction → Direct Addressing

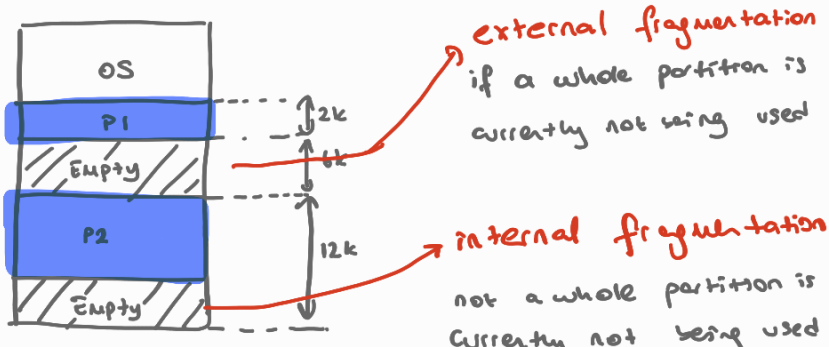


Swapping



Fixed Partioning

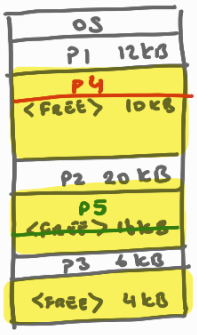
Fragmentation



Variable Partoning

→ First Fit

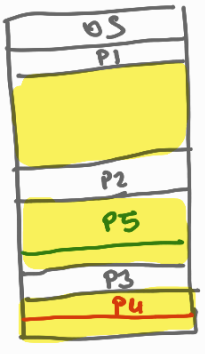
A fast algorithm



P4: 3kB
P5: 15kB

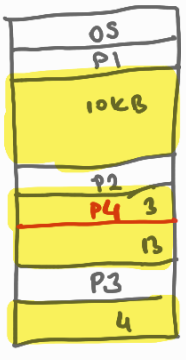
→ Best Fit

Sorting is used



→ Worst Fit

Sorting is used

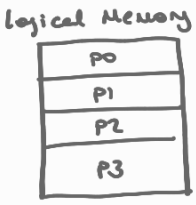


P5: 15 kB !

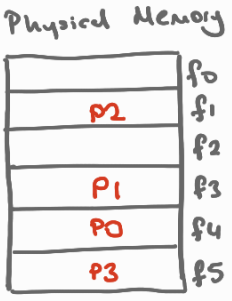
↓

Compaction is needed

Paging



Page Table	
page	frame
0	4
1	3
2	1
3	5



→ paging permits a program to allocate noncontiguous blocks of memory

S → page size

p → page number

d → displacement

Physical address

(f * S + d)

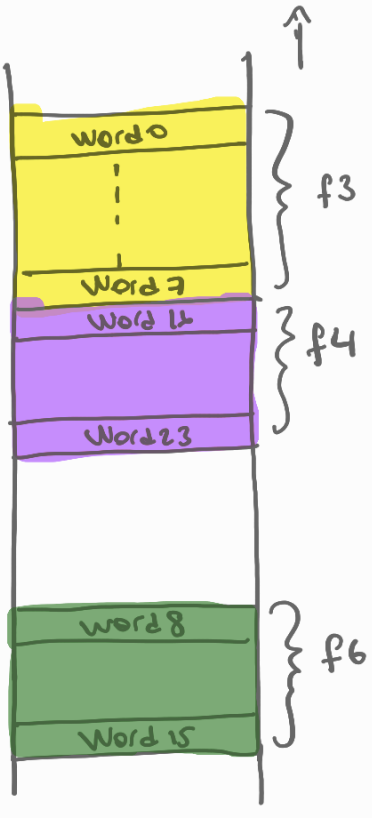
Ex

Page Size = 8 words → d: 3 bits

Physical Memory Size = 128 words 128/8 = 16 frames → f: 4bits

Assume there are 3 pages → 2bits

Page Table	
p	f
0	3
1	6
2	4



(p,d)

Logical Address

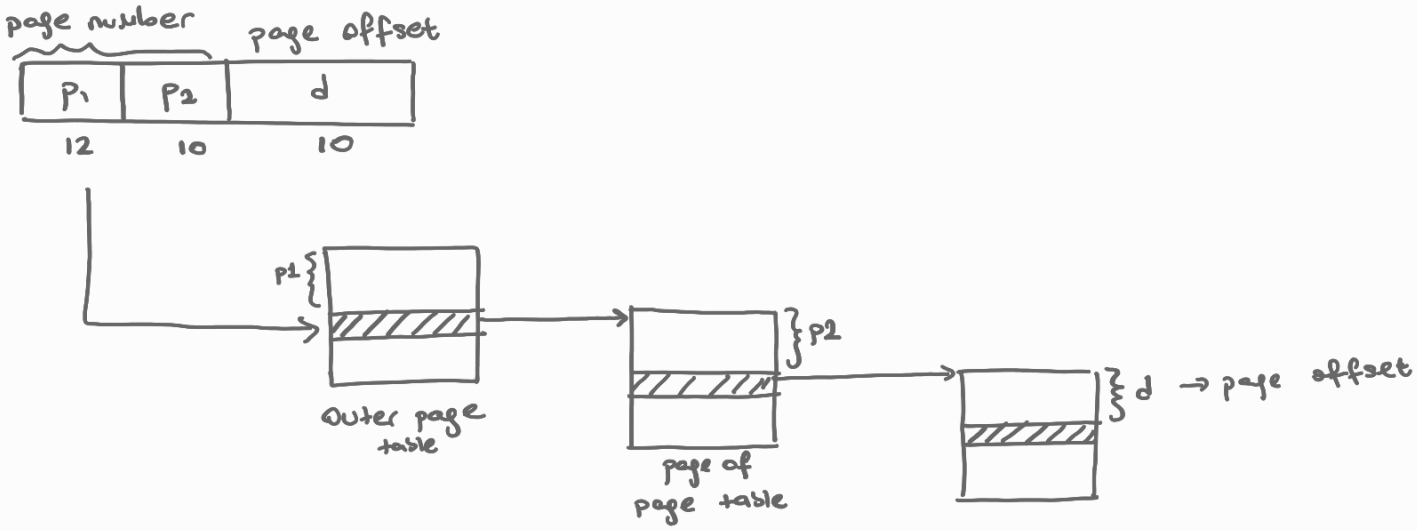
p	d
00	000
:	:
00	111
01	000
01	:
:	111
01	000
10	:
10	111
:	
10	
10	

Physical Address

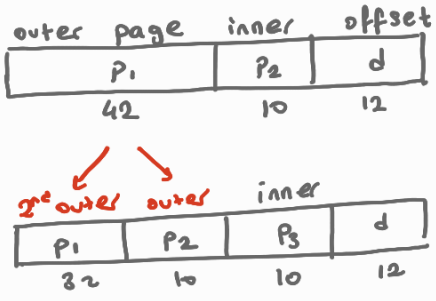
f	d
0011	000
:	:
0011	111
0110	000
:	:
0110	111
0100	000
:	:
0100	111

Hierarchical Page Tables

→ Two-level



→ Three-level paging



Sharing Pages

Read-Only codes can be shared among users. Only one copy of the editor is enough

Ex Editor: 90 MB (3 pages)

Data: 30 MB (1 page)

If 3 user exists → 3x (90+30) = 360 MB

↓ w page sharing

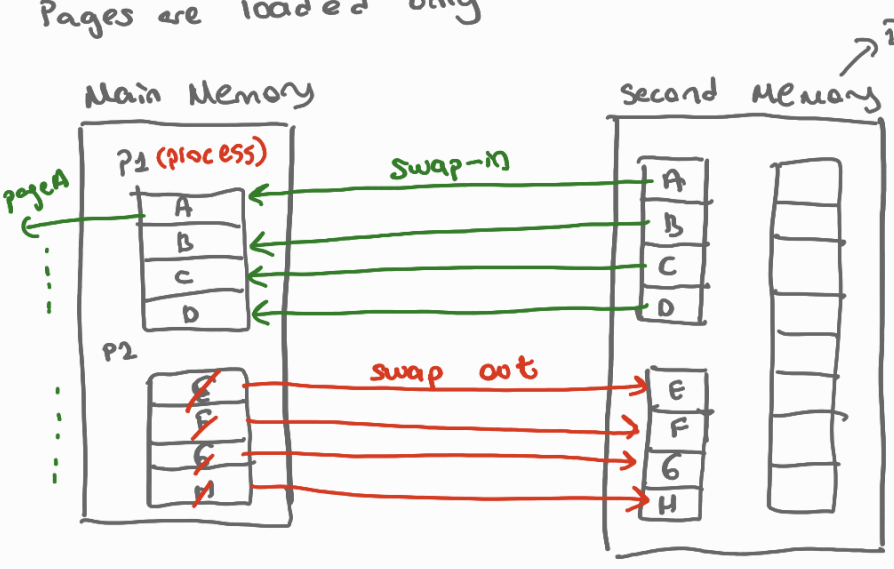
90 + 3x30 = 180 MB

Demand Paging

The processes reside in secondary memory

Pages are loaded only on demand.

what is segmentation fault?



If the page is not present in main memory

↓

Page Fault

↓

CPU transfers control from program to OS to demand the page back into main memory

eat → effective access time

$eat_{no\text{-}pf}$ = emat (effective memory access time) ← no page fault

$eat_{pf} = pfst + emat \simeq pfst$

↳ page fault service time

$$eat = p * eat_{pf} + (1-p)\, eat_{no\text{-}pf}$$

pfst ↑ emat ↑

Dirty bit

to reduce PF service time

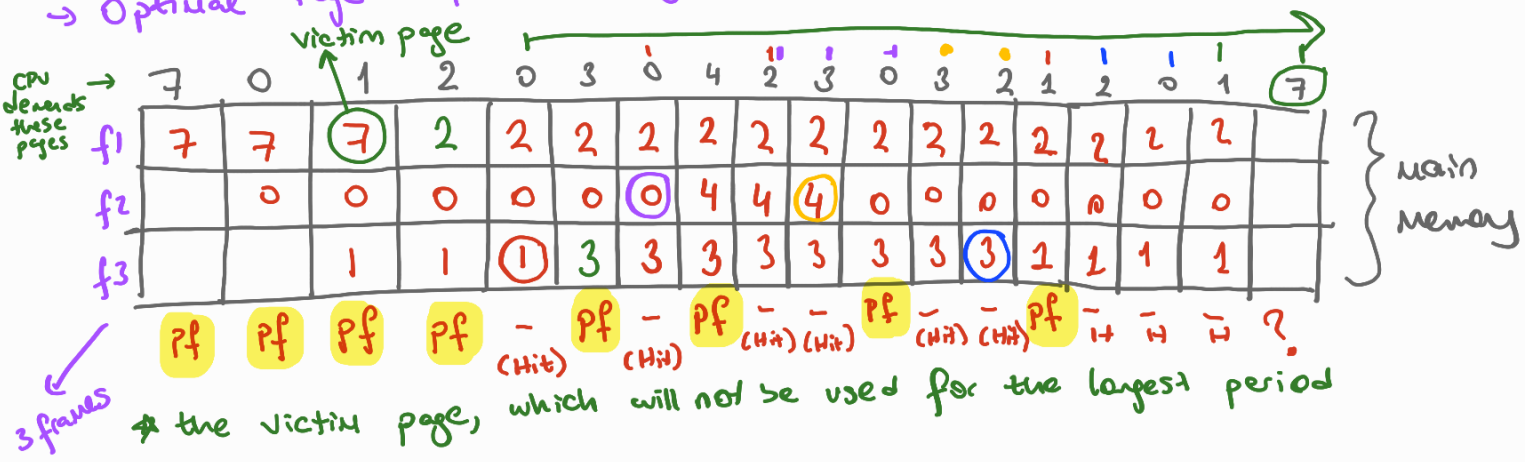
when page is modified (dirty bit is set to 1)

$$eat = p * \left[\underbrace{(1-d)}_{\text{prob(page is not dirty)}} * swap\text{-}in + d * (swap\text{-}in + swap\text{-}out) \right] + (1-p) * emat$$

Page Replacement Algorithms

→ Optimal Page Replacement Algorithm

↗ not practical since it requires future knowledge.

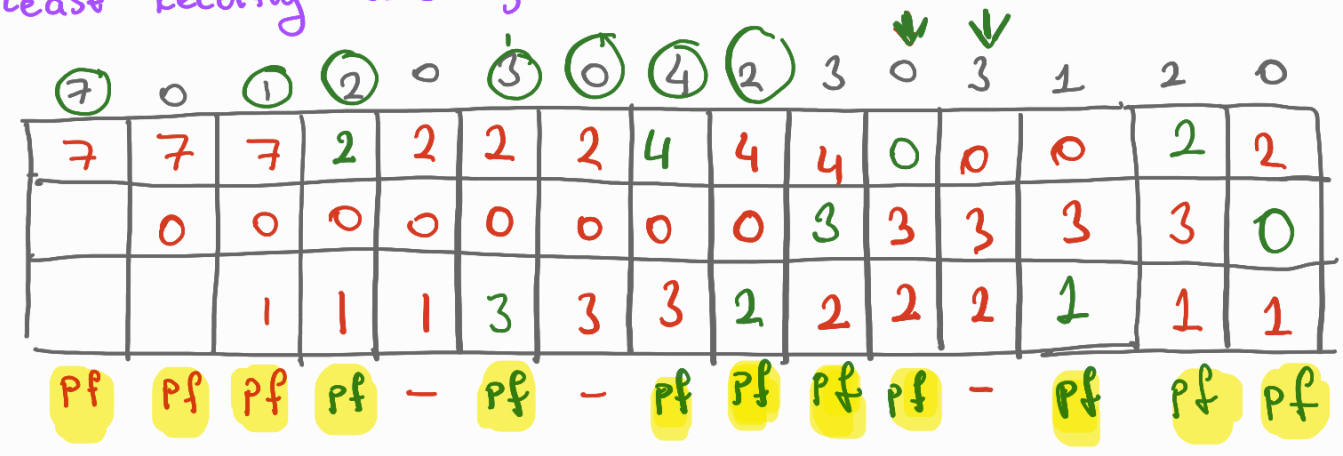


* the victim page, which will not be used for the longest period

→ First In First Out

* the victim page, the oldest page as the victim.

→ Least Recently Used Algorithm (LRU)

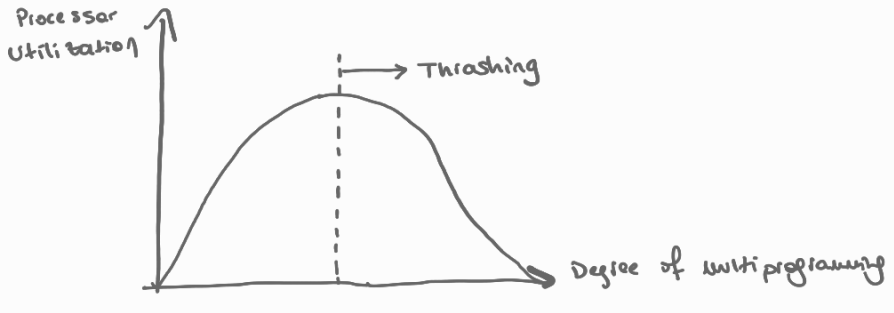


* victim → has not been used for the largest period

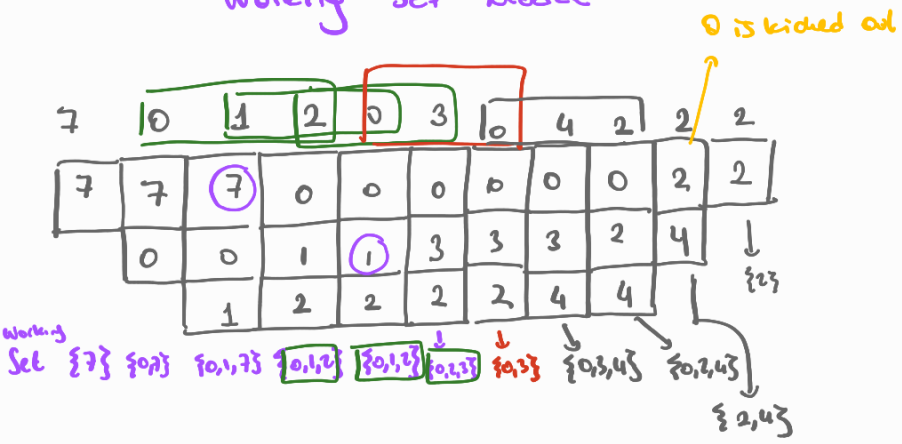
Trashing

The processor depends most of its time swapping pieces rather than executing user instruction.

High page fault rates → Low CPU utilization



Working Set Model



Window Size ($\Delta = 3$)

* Since this algorithm is DYNAMIC. Technically, we begin with zero frames.