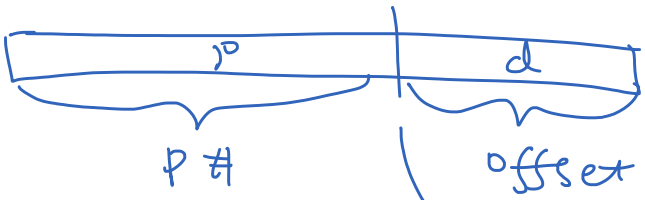
page table

page # ⟷ frame #

DB design's perspective

key: id	p#	f#

logical address



This 'chopping' operation is done automatically by MMU.

# of bits used to represent the offset is determined by the page size

page size (4KB) would need 12 bits

$$2^{12} = 4K$$

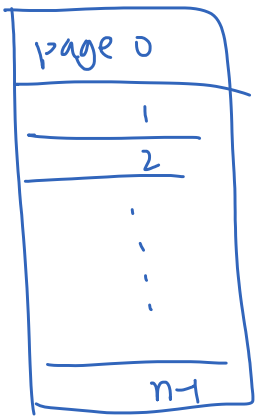
Why we need to use hierarchical structure? ↑ for page table

for page table

Size of page table varies

Solution: store page table into pages.

e.g. process has n pages w/ page size N Bytes

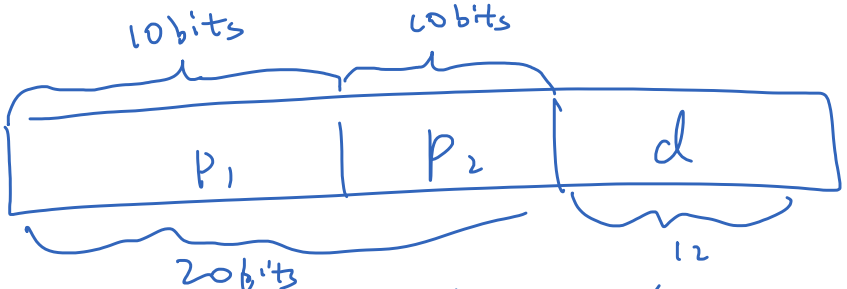


for user process data/instr

page table is stored and managed by kernel.

it could use different sizes of the page.

for kernel data structures.



32 bits page size is 4KB $\Rightarrow$ 12 bits for

32 bits ... page size is 4KB $\Rightarrow$ 12 bits for offset

$(2^{12} = 4K)$

if we store page table into pages (4KB sizes)

then

One page (4KB) would store 1K entries

[P.T.E needs 4 Bytes]

So # of bits to represent the "address" of each entry is 10 bits

$2^{10} = 1K$

register access time a

cache " " b

mem " " c

A two level page table, given a logical

address a_1, the reference to a_1 would need _____ time units

a) a

b) a+b

c) 3·c.

d) 4·c

from effective memory access perspective,

Contiguous allocation is the most effective.

effective access time is (2)a + c

OS maintains a base [register] & length [register] for each process

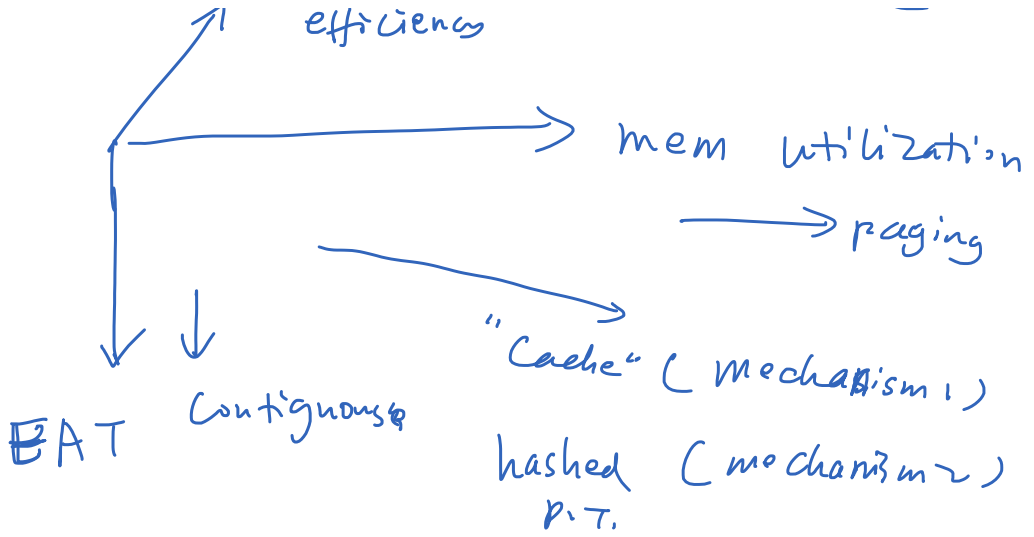
given a logical address a_1,

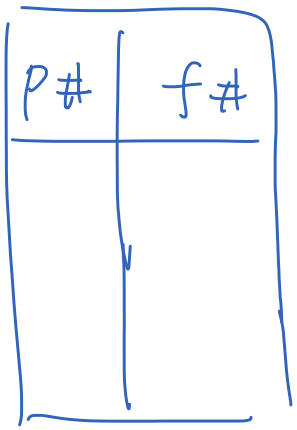
physical address is base + a_1

EAT = access to base register + length

management efficiency ↗

2a + c





if we could combine all the page tables together, then we could find that p# values are not unique.

The other way around,

if there is 4GB mem (physical) page size is 4KB then

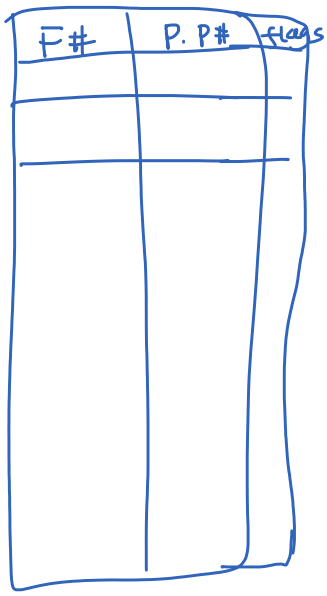
there is $\frac{4GB}{4KB} = 1M$ frames

assume P.T.E uses 4Bytes, then an inverted page table needs

$1M * 4Bytes = 4MB$

$1M * 4\,Bytes = \underline{4MB}$

4MB i.P.T.



data structure maintained by O.S. for both used and free memory

Compared to the previous page table mechanism

dynamically manage page tables

↓

a bit map

x ↑ Current # of processes in the system

V.M.

if we have 4GB physical memory (main)

on the Average, a process needs 100MB mem in total.

M.M. could store $\frac{4GB}{100MB} \doteq 40.9$ processes

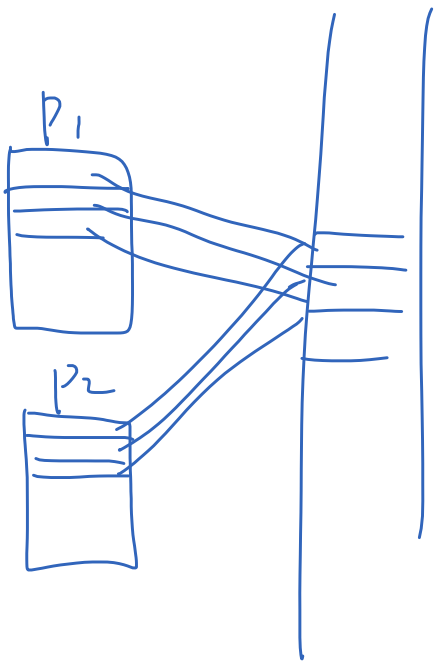
if we load only 10MB for each process.

then m.m could store

$$\frac{4\text{GB}}{100\text{MB}} \doteq 40 \text{ processes}$$

by using VM, these 40 processes have an illusion of m.m. size of

$$40 \times 100\text{MB} = 40\text{GB}$$



Total logical address space >> 4GB physical address space

```
if ( ){
    if_block
} else {
    else_block
```

load to entire process into m.m, OS allocate 2 pages

if we go of demand paging

}

if we go w/ demand paging
then
1 at most.

Page fault rate p

$p = 0 \Rightarrow$ no page fault (all the needed pages are in mem)

$p = 1 \Rightarrow$ every reference causes a page fault.

page size is 1 byte.

4KB page size

each instruction ~~is~~ needs 4 Bytes.

One page may have 1K instructions

each instruction involves one address

one page may have 1K mem references

```
for ( i < 100 ) {
    [ x ]
}
```

100 references to x

2K for each page

then 400,000 references

means $\frac{400,000}{2K} = 200$ pages

$200 \times 4KBy =$ 800 Kilobytes

page 18.

if page D of process 2 is replaced.

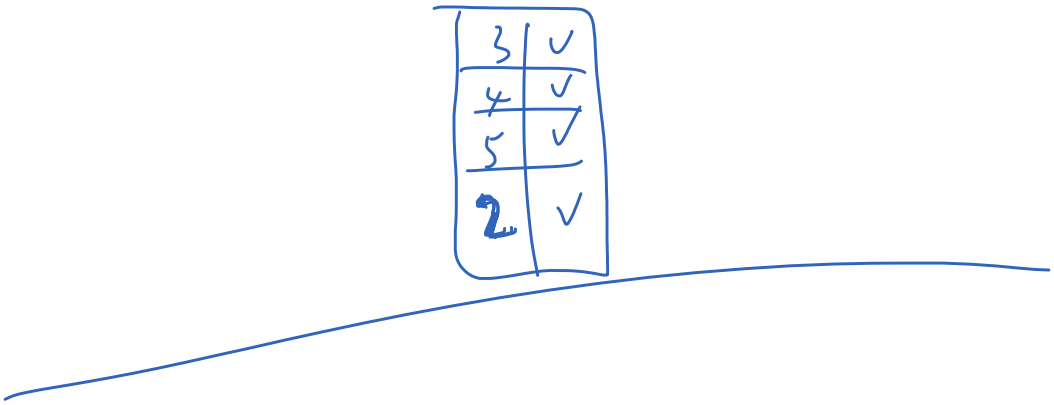
page table for p_2

then

6	V
	i
	i
7	V

page D is swapped out to disk

page table for p_1 is now


3 ✓
4 ✓
5 ✓
2 ✓