Paging

Paging: Introduction

segmentation: chop things up into variable sized pieces.

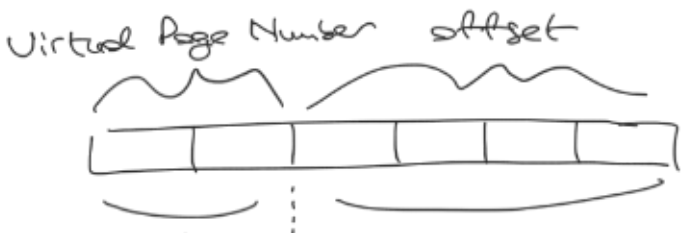
paging: chop up space into fixed-sized pieces.

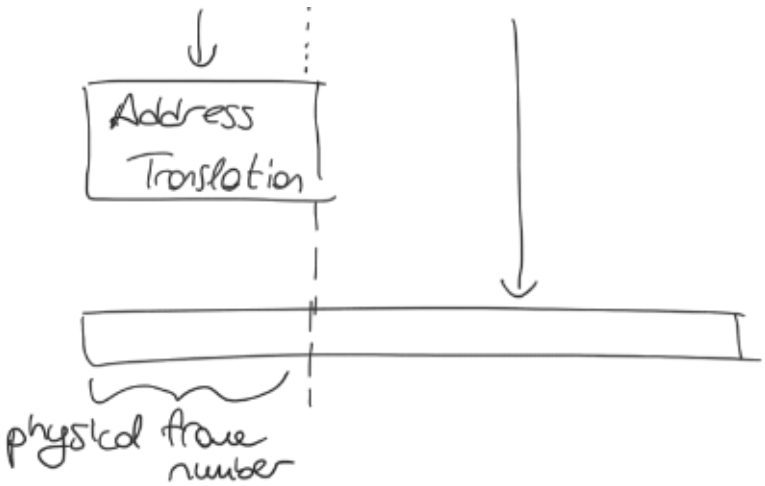
These fixed sized pieces are called page. Correspondingly, we view physical memory as an array of fixed size page frames.

Paging is much more flexible (no heap, stack, code info.) and simple (easy to maintain.).

To record where each virtual page of the address space is placed in physical memory, OS keeps PER-PROCESS data structure known as page table.

Virtual page 0 → Physical frame 3
VP 1 → PF 7
⋮





32 bit address, 4 KB pages

$$\frac{2^{32}}{2^{12}} = 2^{20} \text{ pages}$$

page entry for each page is needed. Assume that 4 byte

$2^{20} * 2^2 = 2^{22}$

4 MB for each process.

virtual address ⟶ physical address

page table data structure.

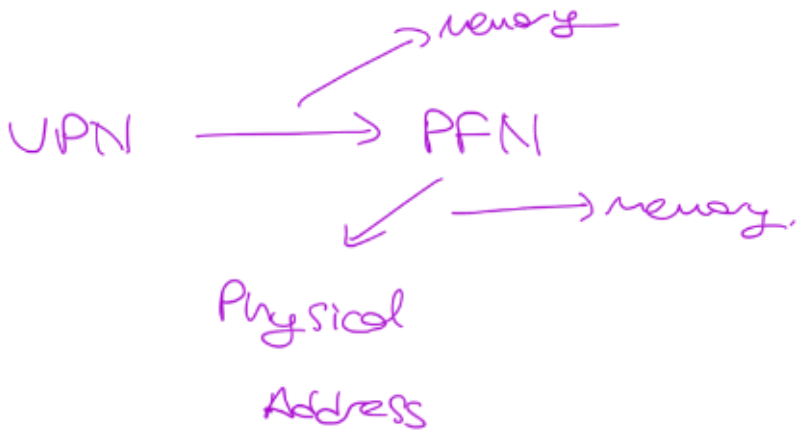
Valid bit is used to indicate whether the translation is valid.

Protection bits → read, write, execute?

Present bit indicates if this page is in memory or on disk

Dirty bit indicates page has been modified.

Reference bit indicates this page is accessed recently



one to the page table to find the physical frame
one to the instruction itself.
two memory job.

Paging prevents external fragmentation.

Paging: Faster Translation (TLB)

TLB is part of the Memory-Management-Unit (MMU)
Hardware cache for translations.

VPN → TLB Hit → PFN
VPN → TLB Miss → Memory → Update TLB → PFN

spatial locality: accessing A[i+1] after A[i]
temporal locality: accessing i from two different thread. If it is recently accessed, it will be re-accessed soon in the future

soon in the future.

CISC : hardware handles TLB miss.

RISC : software managed TLB.

↳ TLB miss → raise an exception → kernel ↓ handle TLB miss

*flexible

*ease for hardware.

fully associative: any given translations can be anywhere in TLB.

VPN | PFN | other

TLB entry.

VPN	PFN	valid	prot	ASID
10	100	1	rwx	1
—	—	0	—	
10	170	1	rwx	
—	—	0	—	—

~~when~~ context switch occurs ~~what~~ to do?

a) flush all info.

b) add new identifier called address space identifier (ASID)

(ASID)

- Least-recently-used can be chosen as replacement policy.

- Global-bit is used for pages that are globally shared among processes in TLB. Thus if the global bit is set ASID is ignored

- OS keeps some of the TLB entries for itself in case of emergency.

- # of pages program accesses > # of pages fit into TLB → too many TLB miss.

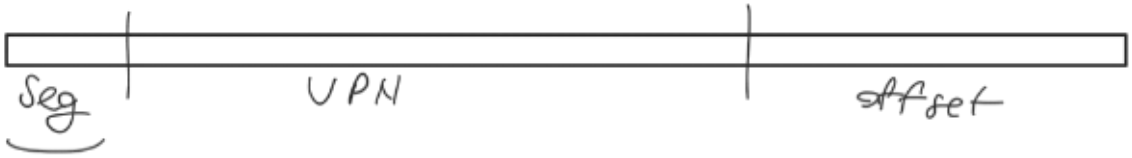
Paging: Smaller Tables:

Bigger pages?

Not a solution.

* it may cause internal fragmentation
* it does not improve that much.

Hybrid approach?



00 : unused
01 : code
10 : heap

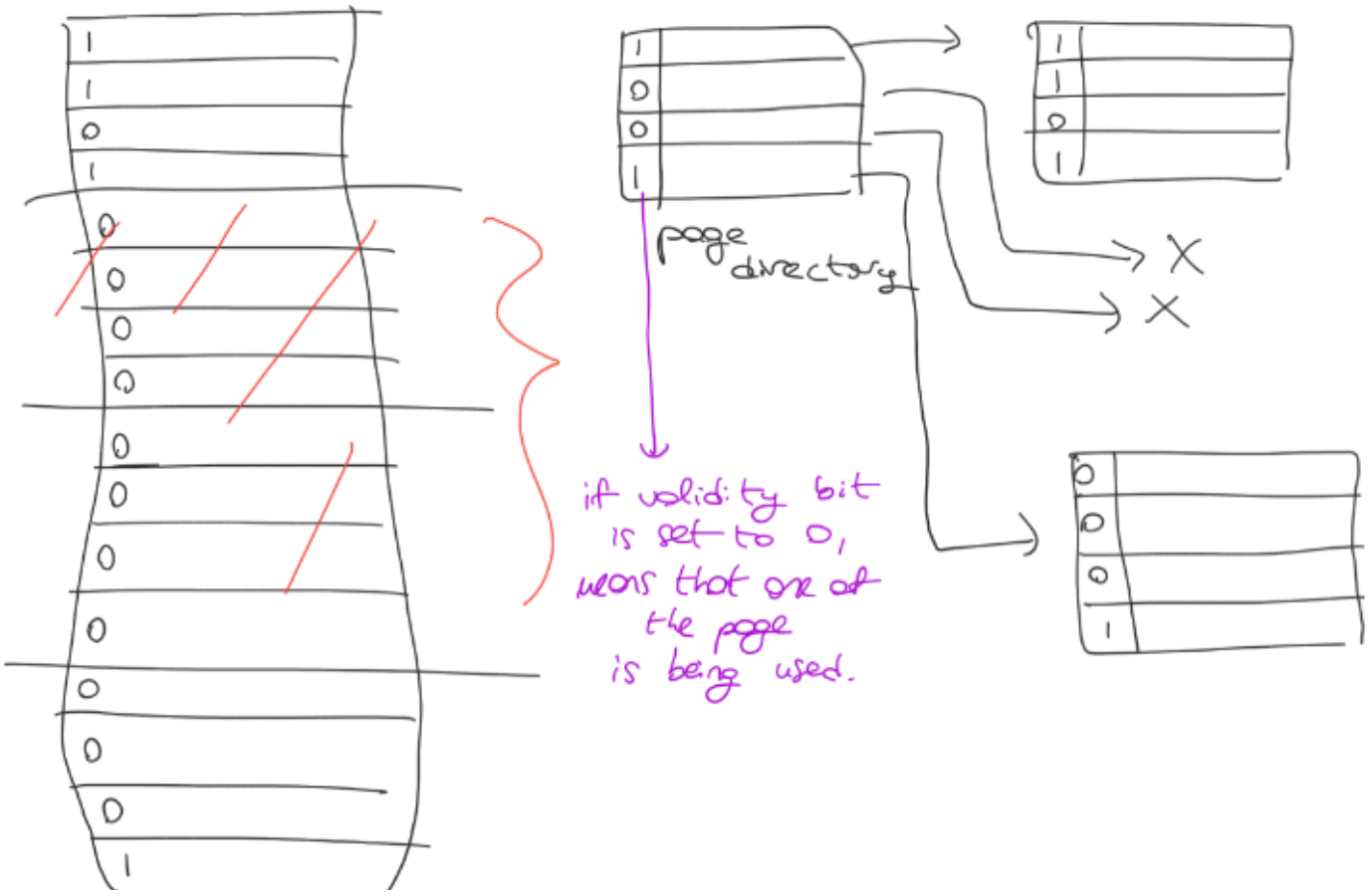
1 1 : stack

for ex. if the code segment is using its first three pages, the code segment page table will only have three entries allocated to it and bound register will be set to 3.

* Flexibility dies.
* Complexity rises.

Multi-level Page Tables:

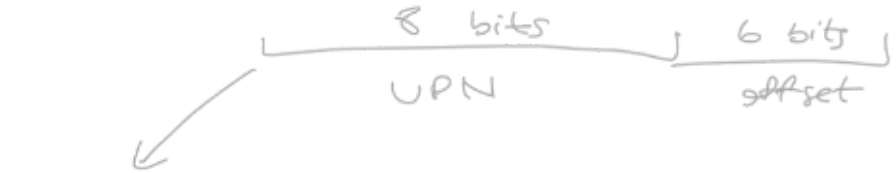
First, chop up the page-table into page-sized units; then, if an entire page of page table entries is invalid, do not allocate that page of the page table at all.



* time space trade off
* complexity

Address space size : 16 KB
Page size : 64 byte

$$\frac{2^{14}}{2^{6}} = 2^{8} \text{ pages}$$



256 entries → 16 * 16 pages

Inverted page tables

Instead of having many page tables, we keep a single page table that has entry for each physical address of the system.

Finding the correct entry is now a matter of searching through.

Last modified: Sep 30, 2019