Worked!

Virtual Memory

Techniques:

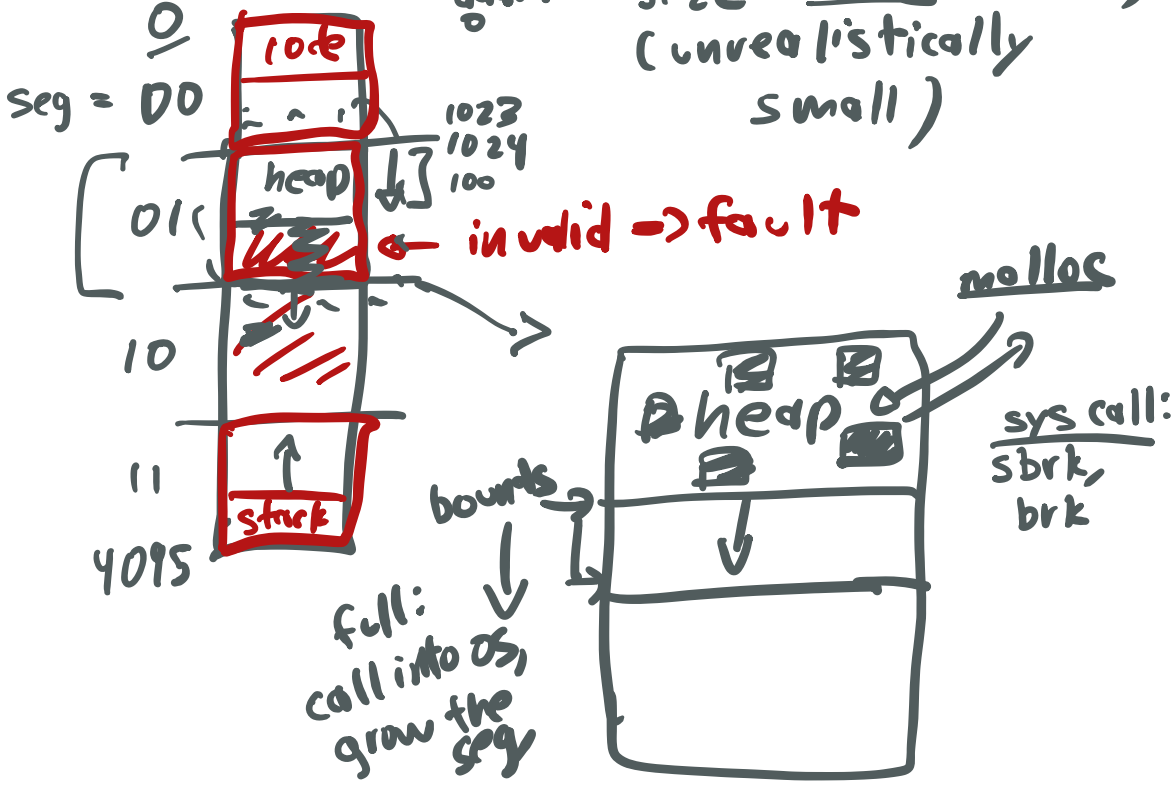
- Base / Bounds
- **Segmentation**
- Paging

MMU: hardware
per segment
base/bounds

=> virtual Address Space

addr: 0

size: 4KB (4096)
(unrealistically small)



Realistic: 32-bit virtual addr. space

10 bits → 1 KB (~ 1000)

20 bits → 1 MB (~ million)

30 bits → 1 GB (~ billion)

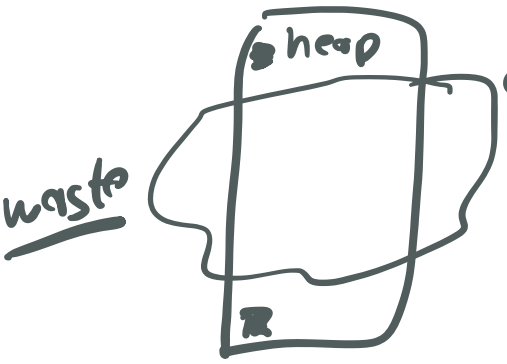
Problems w/ segmentation:

=> fragmentation (external):

free space becomes full of small used spots + free spots intermixed

-> not fully flexible

e.g. "sparse" heap



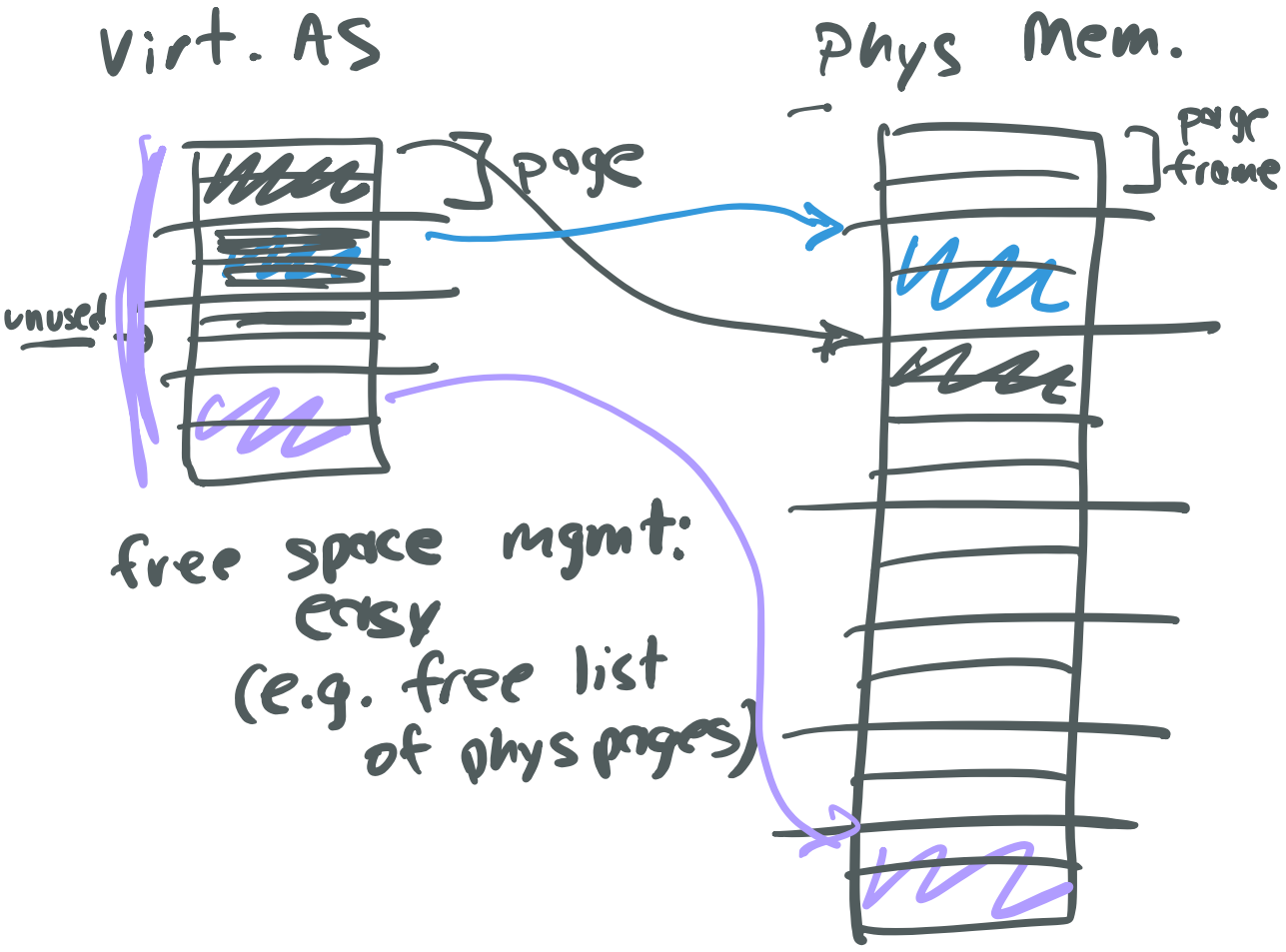
even though unused, must keep in memory

(internal fragmentation)

New approach: [Paging]

⇒ divide virtual address space into (e.g. 4KB) small, fixed-sized units called pages

⇒ similarly, divide phys mem into 4KB page frames

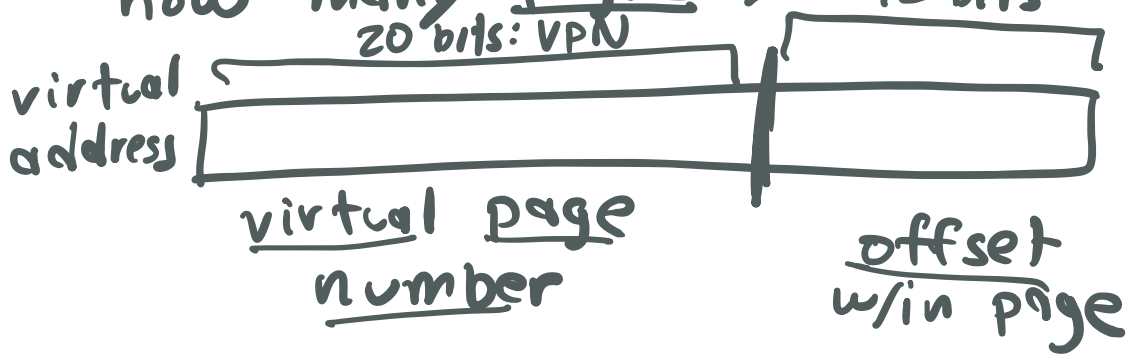


Translation: virtual addr => phys. addr

e.g. virtual address space: 32-bit ~4 GB

page size: 4 KB

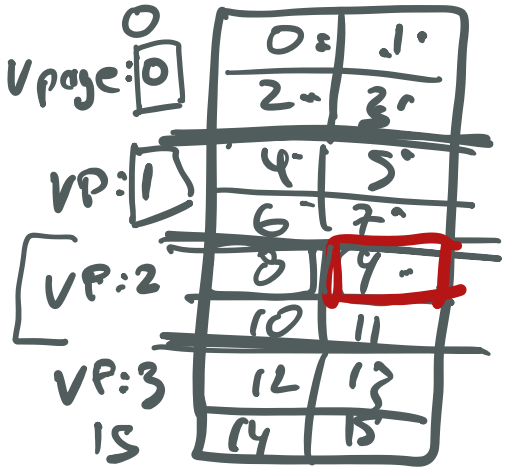
how many virtual pages? $\Rightarrow 2^{20}$



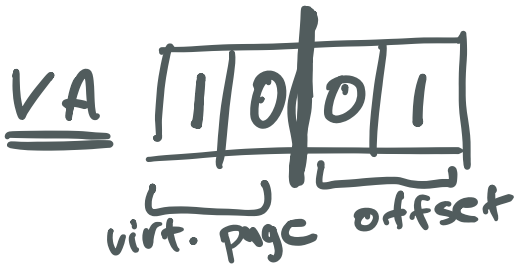
tiny e.g.

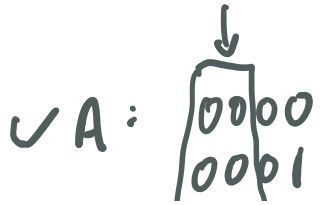
16 bytes: virtual address space size

4 byte page size



virt. Address:





#2

Problem: too many pages!

e.g. 2^{20} (per process)

(a lot of information)

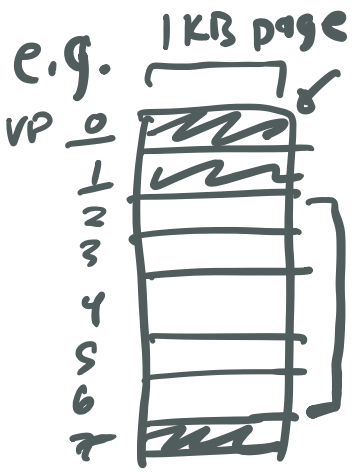
where to store?

=> main memory (managed by OS)

in a data structure

called "page table"

(store virtual -> phys page translations)



page table:

(for process)

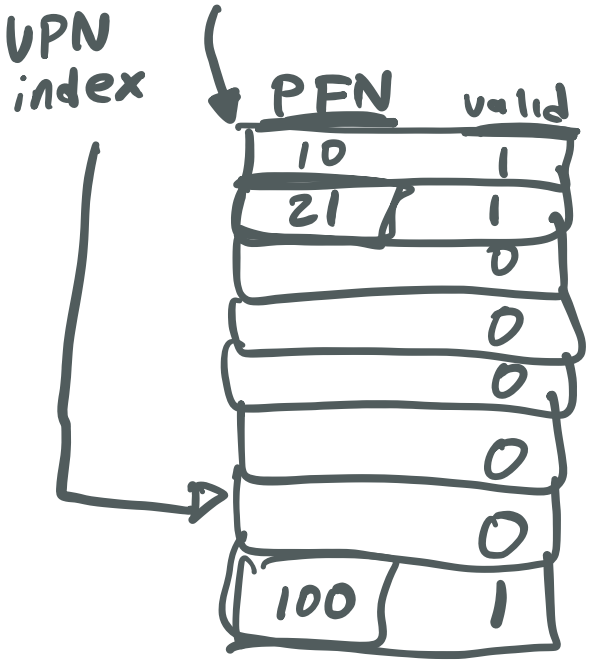
store up to 8 translations

VPN	-> PFN	valid
0	10	1
1	21	1
2		0
3		0
4		0

naturally supports "sparse" addr. space

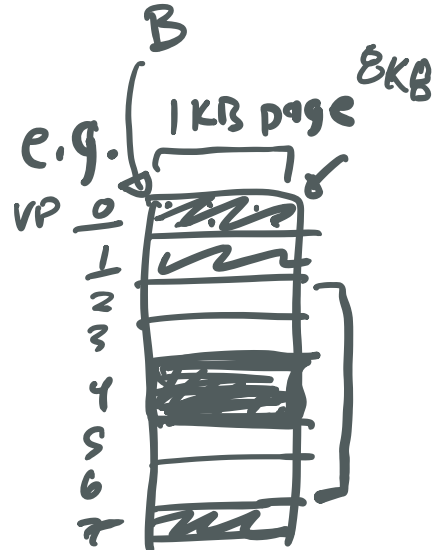
5		0
6		0
7	100	1

First page table: linear page table (array)

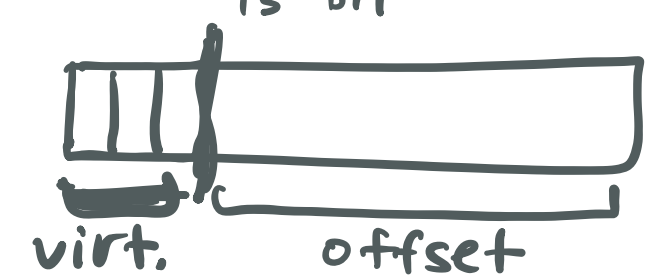


[array of page table entries]

translation:



virtual address:



hardware: [base address of page table:
VPN

B + (VPN x sizeof(page table entry))]

[load PTE from memory
extract PFN]

form full phys. addr. (PA):

PA | PFN | offset

go to memory for access

=) Slow (extra mem. ref. per mem ref.)