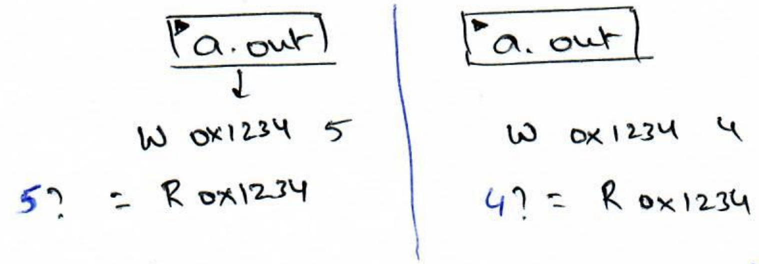
OS background: Process virtualization

②

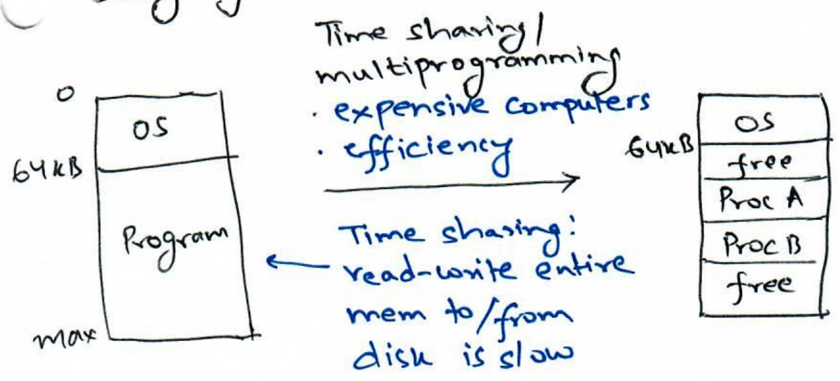
main.c → a.out



Memory of Processes are completely isolated

Early systems

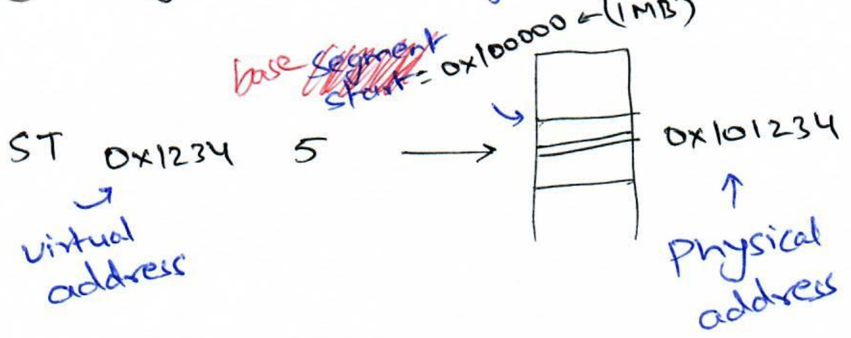
③



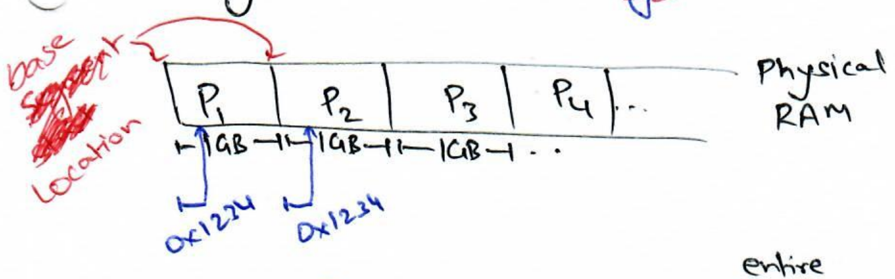
Physical memory

Base and bound
~~Segmentation~~: ~~Performance~~

②

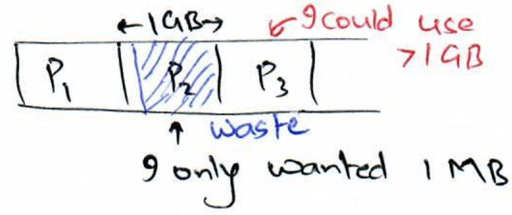


Memory virtualization; ~~Segmentation~~ base & bound ③



+ Virtualization: Process think it owns ~~the~~ entire memory
+ Performance : Just adder
+ Isolation/security - P2 cannot see P1's memory
- Utilization

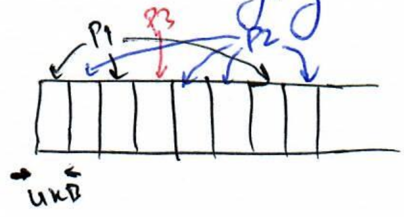
~~Segmentation~~ Problems - Low utilization ③



Low flexibility

: I can only start 4 applications?

Solution: Paging ③



+ Flexibility: Dynamic mapping
  - Allocate a page only when required
  - Send a page to disk if not used (demand paging)

+Dynamic mapping ⇒ Over provisioning ①

: Can start many more processes

+ over-provisioning ⇒ better resource utilization

Less free/unused memory

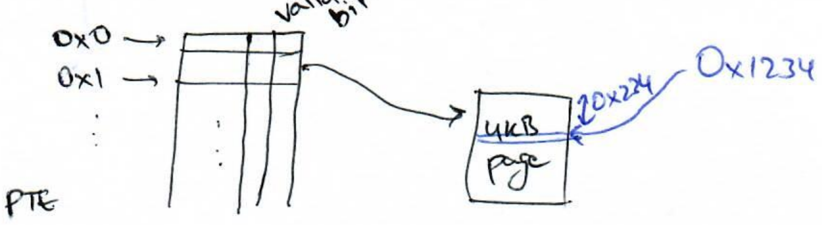
- Flexibility ⇒ complex address translation ①

Virtual address $\underbrace{0x1}\underbrace{234}$ → ?? Physical address

↓

Page number 0x1 → ?? Frame number/ Physical frame number

Page table: Page# → Frame# ③



Read-only bit : Don't accidentally write code section

Dirty bit: Was the page written to

Present bit : Allow sending to disk (over-provisioning)

Page table is huge ⍨ ⑤

32-bit arch ⇒ 2^{32} virtual addresses | 32 GB physical RAM ⇒ 2^{35} physical addresses

Page size → 2^{12} 4kB | # frames → 2^{23}

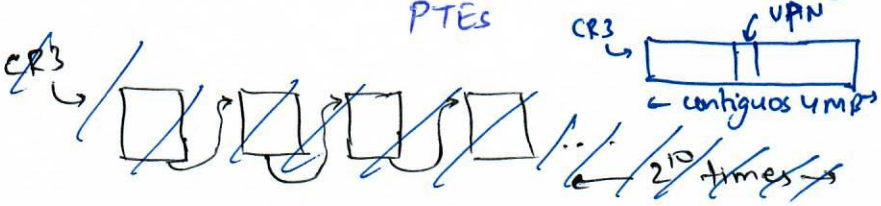
pages → 2^{20}

⇒ 2^{20} PTEs. Each PTE has 23 bits + (V, P, RO, ..)
$\leqslant$ 32 bits ~ 4 B

⇒ One 4KB page can have ③
(2^{12}) $2^{12}/4 = 2^{10}$ PTEs

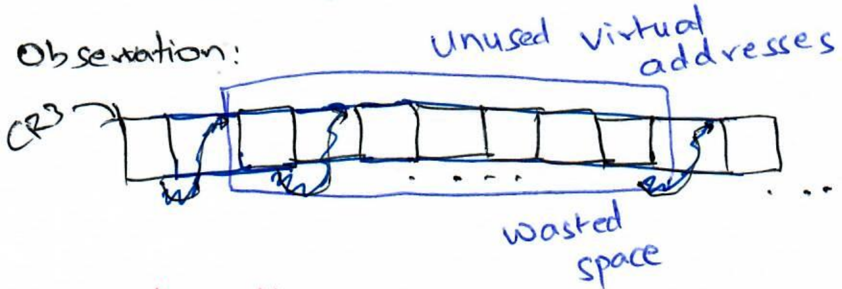
Total PTEs required → 2^{20}

Naive solution: Use 2^{10} pages containing PTEs



⇒ 2^{10} * 4kB = <u>4MB</u> of pages for ③
holding page tables of 1 process!

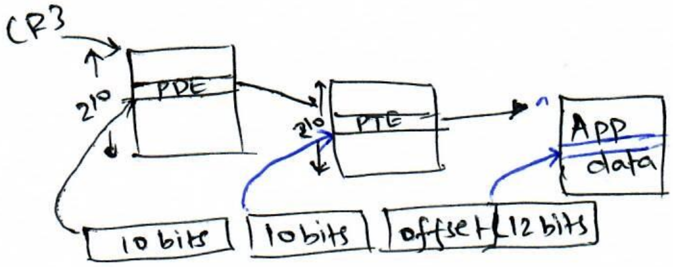
Observation:



Same observation as segmentation

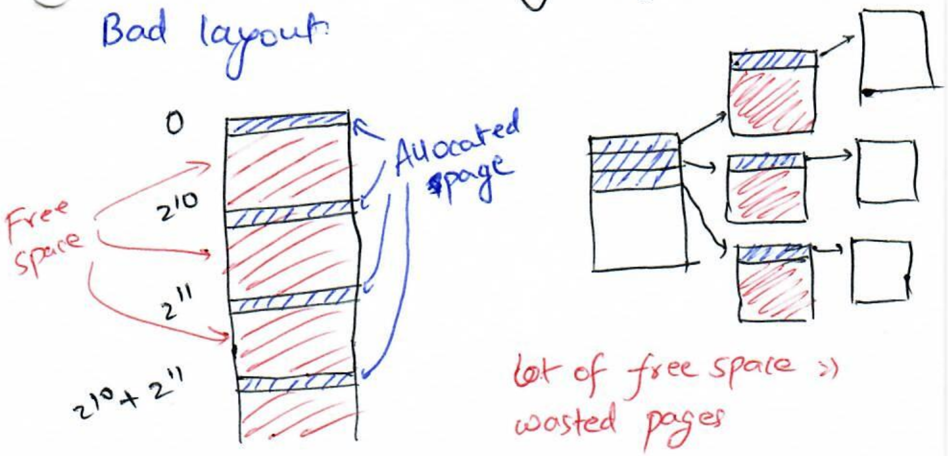
Same solution: Lazily allocate page to page table ③

Hierarchical page table



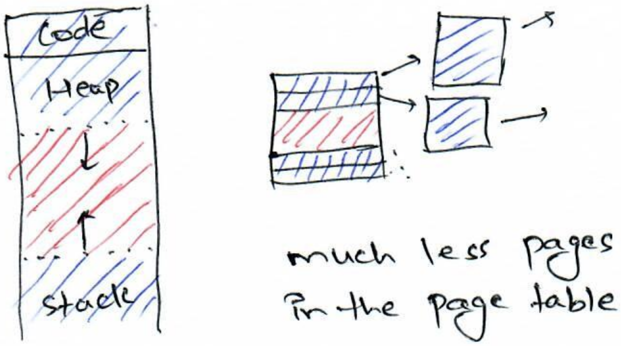
Effect on memory layout ④

Bad layout:



lot of free space => wasted pages

Good layout: segments ③



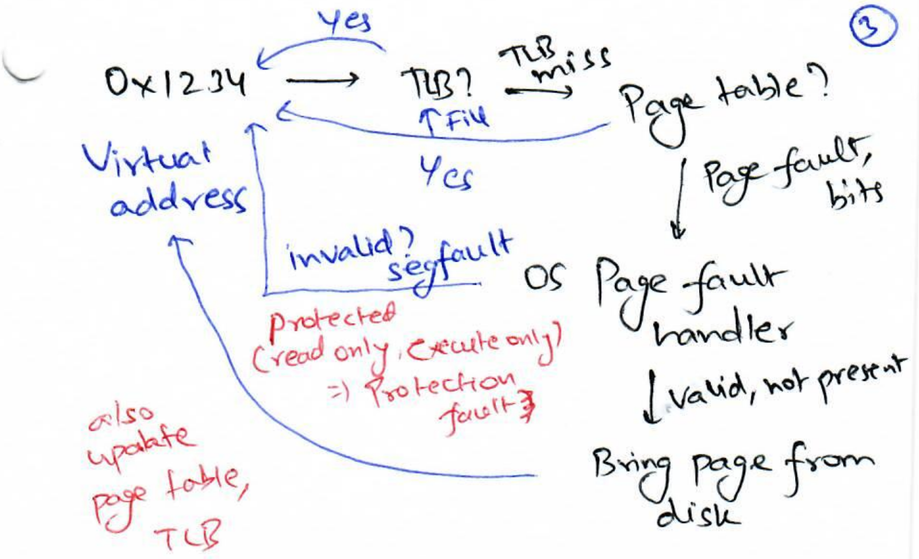
much less pages in the page table

Problem: Every Load/Store needs 3 ②
memory accesses 2 page table + 1 actual

For 64 bit architectures

3 → 6

Soln: TLB. remember page # → frame # mapping



Memory virtualization paging vs static segmentation ②

* Each process thinks it own entire (virtual) address space ⇒ Transparently move pages to disk

~ Security

+ Overprovisioning ⇒ better resource Utilization

- Lot more complexity in hardware/os