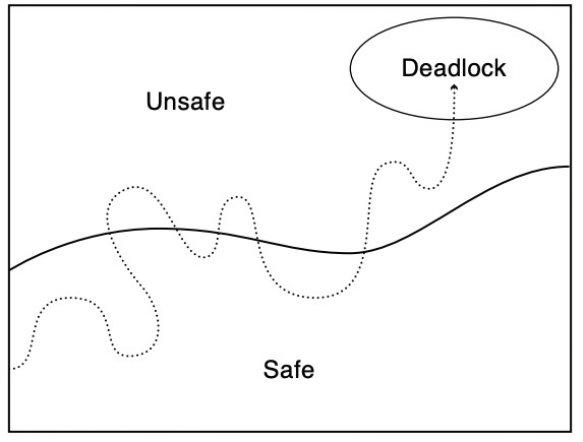
4/22/24

Deadlock Wrapup & Memory Management

Deadlock Avoidance



3 Chopsticks, 3 philosophers — request 1 chopstick at a time / max 2 requests

	A	B	C	
safe	0 (2)	0 (2)	0 (2)	avail = 3
safe	1 (1) ✓ can finish	0 (2) ✓	0 (2) ✓	avail = 2 (starts out w/ 1 avail)
safe	1 (1) ✓	0 (2) ✓	C (1) ✓	avail = 1
unsafe	1 (1) ✗	1 (1)	1 (1)	avail = 0

to be avail = 2: 2 − 1 + 2 = 3 (A request 1 more) (A will finish & return both)

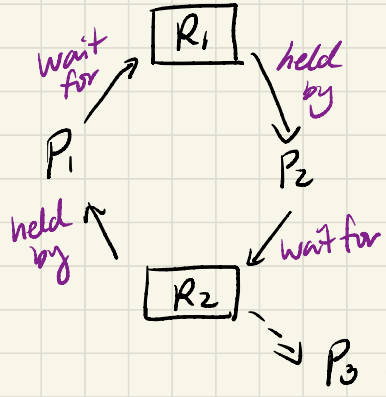
to be avail = 1: 1 − 1 + 2 = 2 A can finish; 2 − 0 + 2 = 2 B can finish; 2 − 1 + 2 = 3 C can finish

to be avail = 0 no one can finish their future request

Deadlock Detection

→ rare event, let it happen, detect & recover when it happens

→ Resource Allocation Graph



if there's a cycle:

- single instance resource ⇒ deadlock
- multi instance resource ⇒ potential deadlock

* high false positive

→ recover from deadlock

→ abort / terminate a process in the cycle

Ostrich Algorithm

→ pretend nothing's wrong, let user deal w/ it

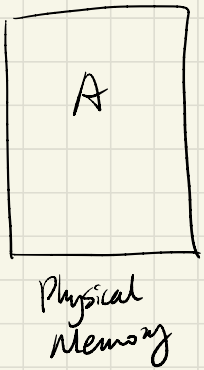
Physical Memory Management

→ volatile, byte addressable, order of GBs

→ ~200 cycles access latency

Problem: many processes, limited physical memory

→ Attemp #1: let one process use all of physical memory



→ no need for translation

→ context switch

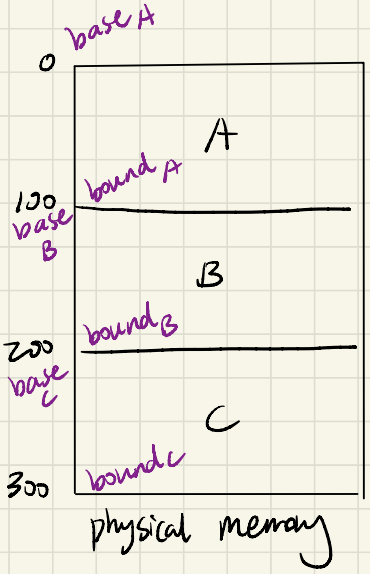
scheduling overhead {

→ write A's VAS to disk

→ load B's VAS into memory

→ how do we enforce kernel/user memory separation

→ Attempt #2: Divide up physical memory among processes



→ low context switch overhead

→ virtual to physical address translation

→ base & bound registers (hw support)

PA = VA + base (VA < bound)

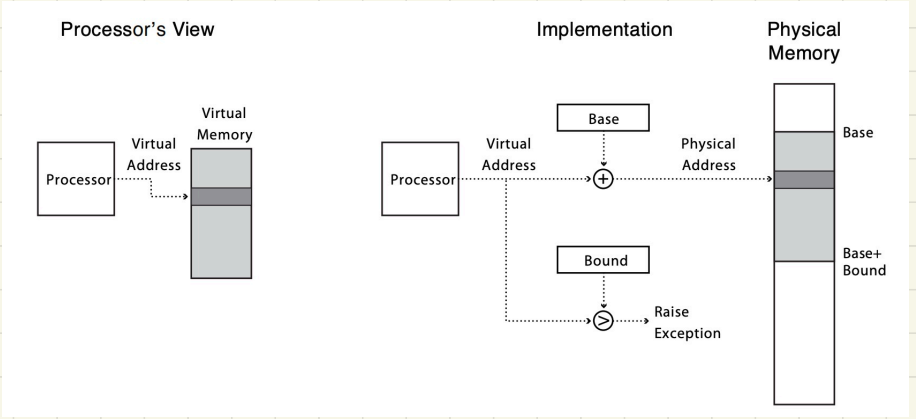
base register 0 for process A

bound register 100

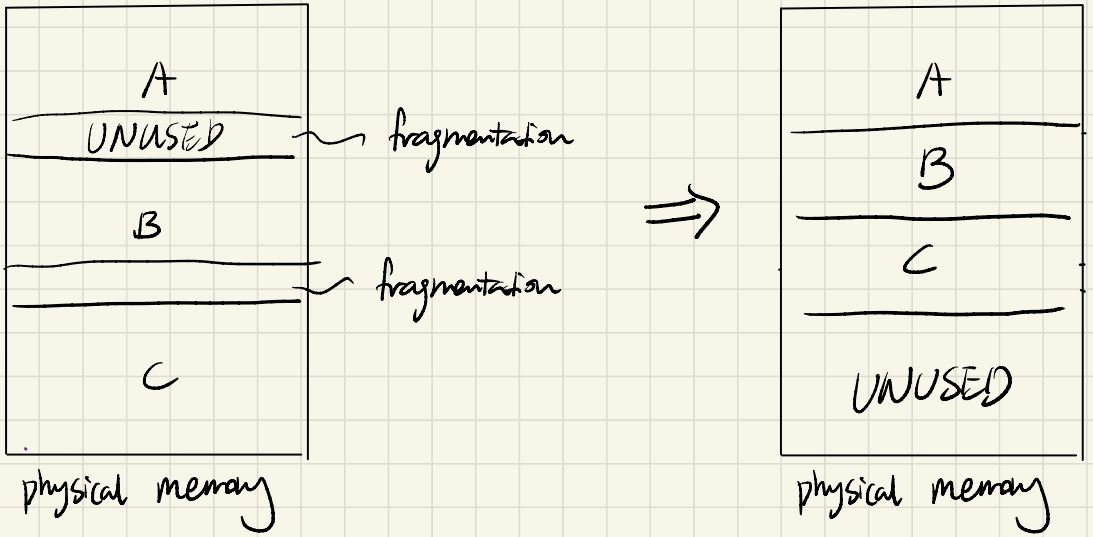
→ how do we support memory growth?

→ variable sized allocation leads to external fragmentation

D doesn't fit, can't allocate



Compaction



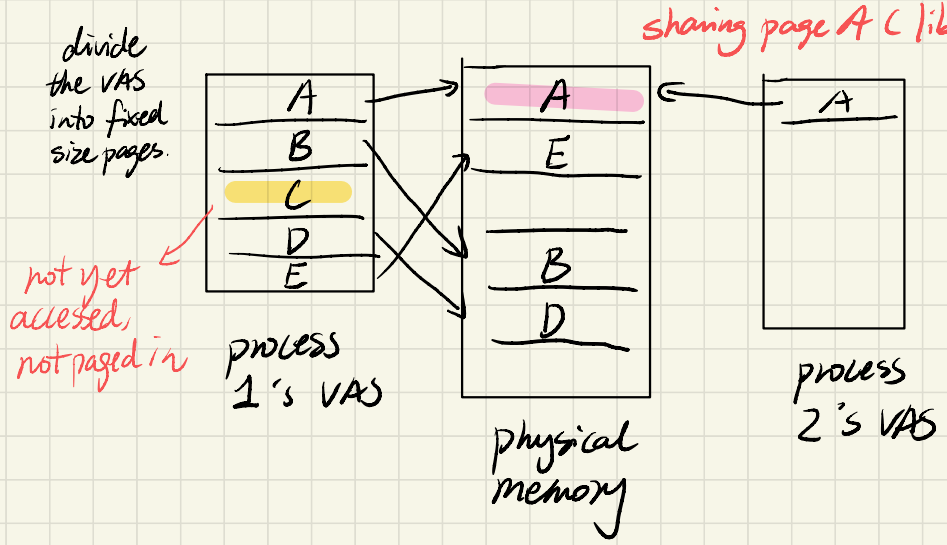
update base & bound registers for each moved (compacted) process

expensive operation, causes variability in performance.

→ Can we share library code or support cow fork with this approach?

→ translation done on the entire VAS, can't share part of it

→ Attempt #3 : finer grained translation & permission



→ better control for sharing

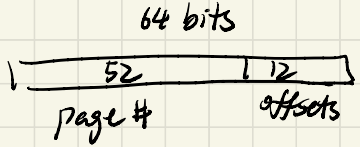
→ fixed sized allocation (page, 4kB)
no external fragmentation
(may see internal ↑ instead)

→ easy to support memory growth

→ only load pages in use into memory (demand paging)

What should be translated

virtual address



offset within a page
= offset within a physical page.
only need to translate page #