CS 219 Lecture 16

step1 `int * p = malloc (sizeof(int) * 100);`

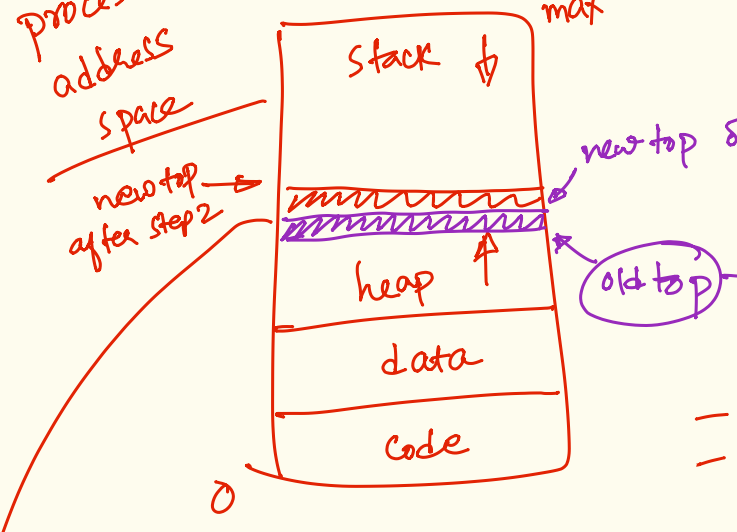
`*p = 23;`

VA ~ virtual address

system call

sbrk

~ find a va-range
(i) for new VA space allocation
- update VA usage into in PCB



(ii) build/update page table for va to pa mappings.
- per process
- pgdir ptr. as stored in PCB

step 2: `int *q = malloc (sizeof(int) * 200);`

VA

(iii) returns start of va-range as return value.

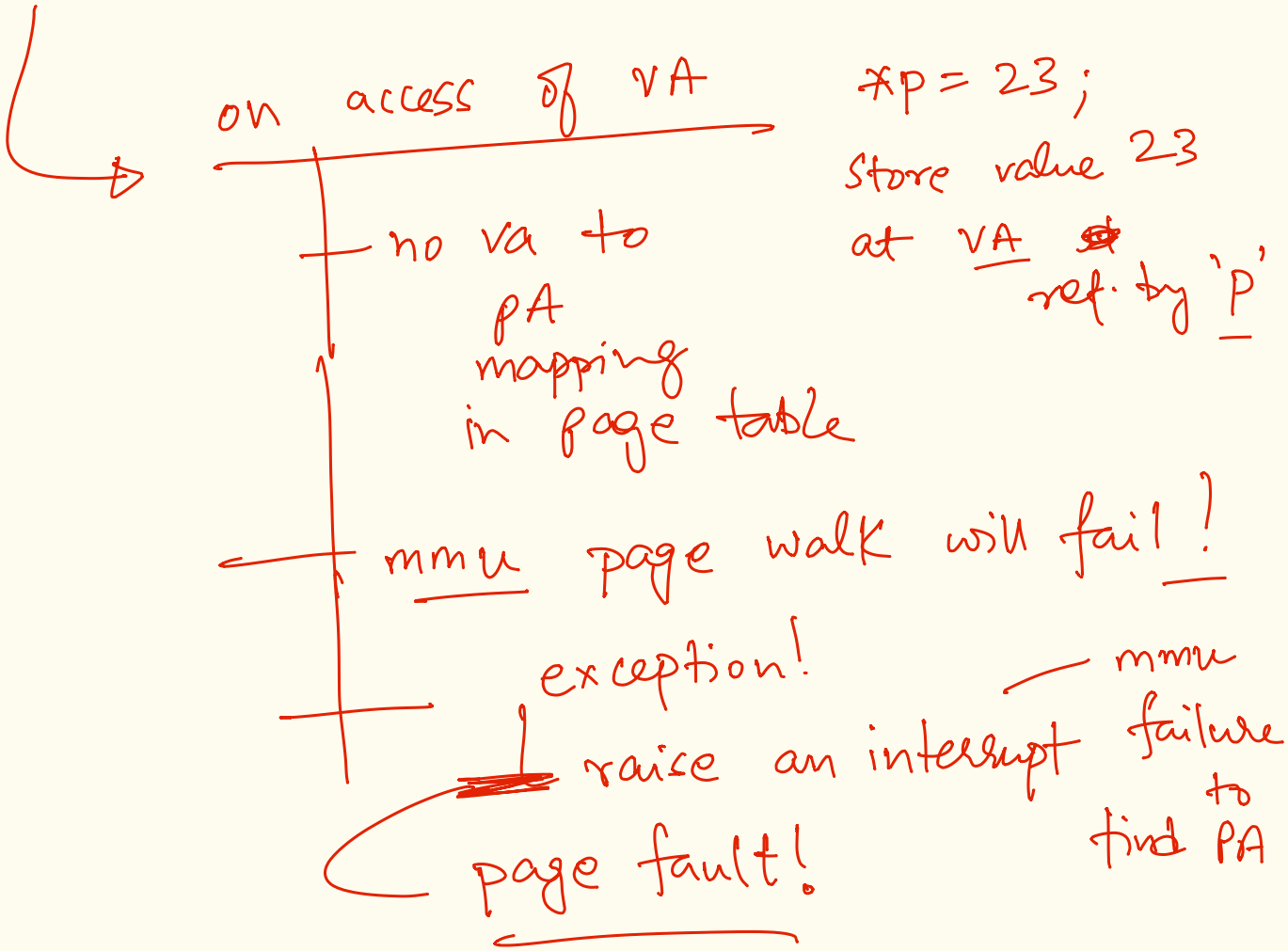
eager sbrk

allocate physical pages to the entire va-range & map

does step (i) & step (iii)

lazy sbrk

memory alloc




```
int *p = malloc(...);
*p = 23;
```

an address is being accessed

- with paging enabled this is a VA.
- mmu (pgtable, VA) → find PA } in h/w

```
int *p = malloc(...);
check pgtable (p);
if ok *p = 23;
```

```
int *p= malloc (100);
*p = 23;
*(p+8000)= 23;
```

On page fault

- operations / execution switches to Kernel-mode
  - lands in page fault handler } interrupt handler of interrupt vector corresponding to a page fault
    - x86
      - CR2 — faulting VA
    - risc
      - stval — faulting VA.
    - check if faulting VA is a valid VA

① if not free & heap grows linearly & contiguously
if (faulting VA < top heap)

⇒ if valid
build mapping in pgtable for va to pa
find a free phy. page!

if not valid
- exception
  - undefined behaviour!

demand paging / swapping

- where content of pages written to disk
- reread to create new v2p mappings when needed.

pte flags (valid ↙, present ↙ bits)

V	P	
0	0	← no mapping
1	0	← swapped to disk.
0	1	← invalid
1	1	← all good, mmu walk possible

in page fault handler

if not lazy check for swapped

- swap content back to memory of new v2p.