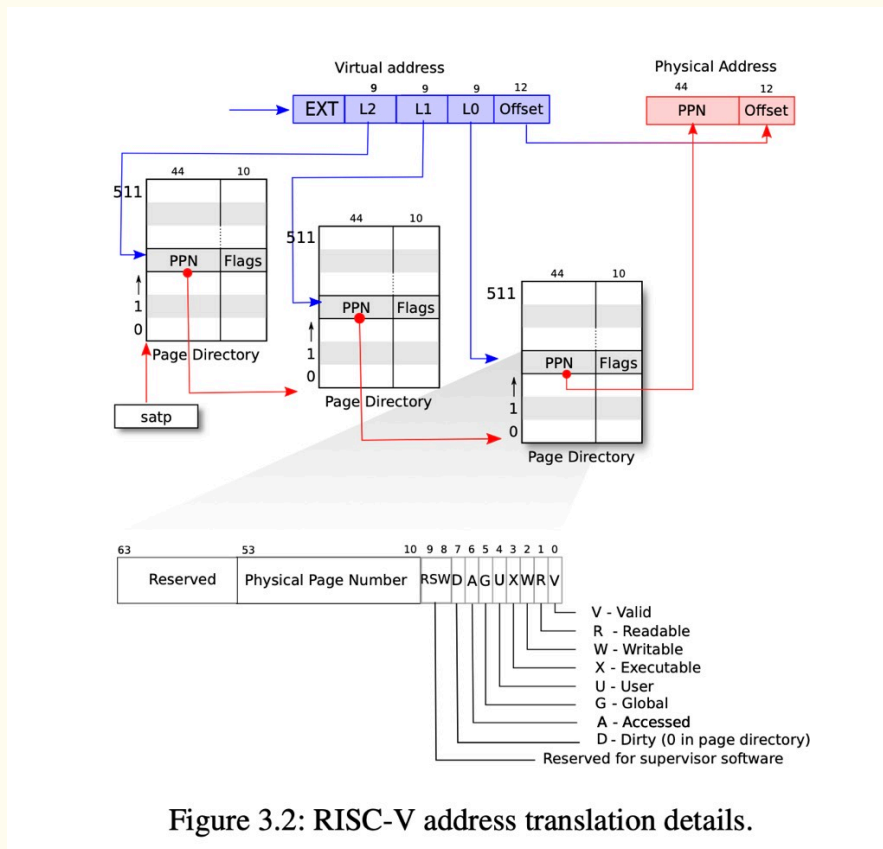


recap: ISA-support for memory virtualization.



page faults.

① exceptions

↳ unmapped memory,
not in heap or
stack.

② cow

↳ mapped and
marked read-
only

③ lazy allocations

④ swapped (to disk)
pages.

⑤ malloc

↳ user space library fⁿ.

↳ returns → an address of the address space
of calling process

virtual address

↳ which address?

↳ unused / unallocated / unreserved

↳ intent to reserve memory on the address
space for some data objects.

VA (int*)
`int *p = malloc(10, sizeof(int));`

`*p = 23;`
`*(p+5) = 23;`
`...`

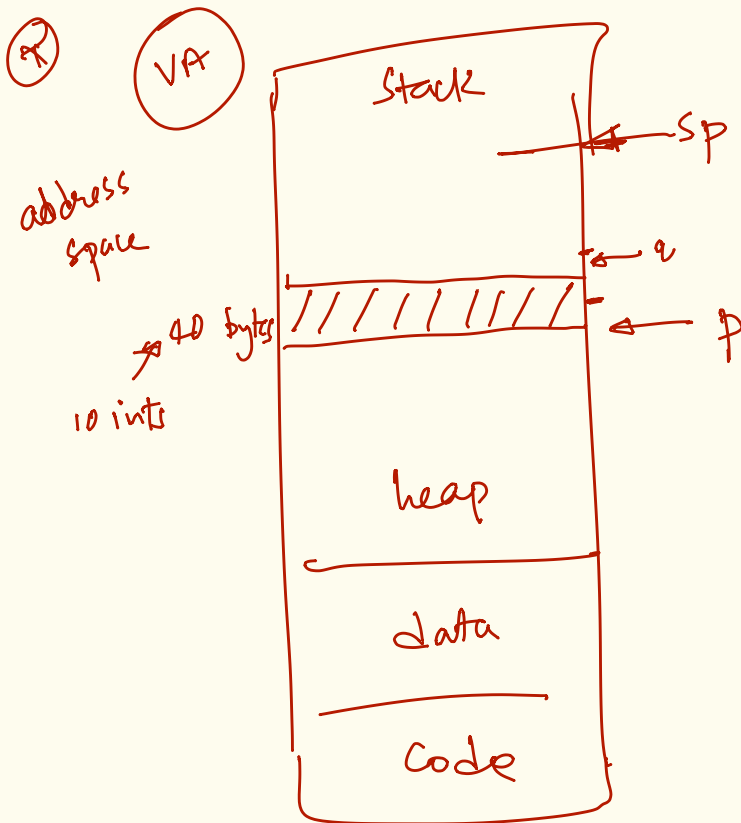
`free(p);`

`p = NULL;`

how much to free?

how to be sure
argument is start
of malloc?

e.g: `free(p+2);`



malloc.
`q = malloc( — );`

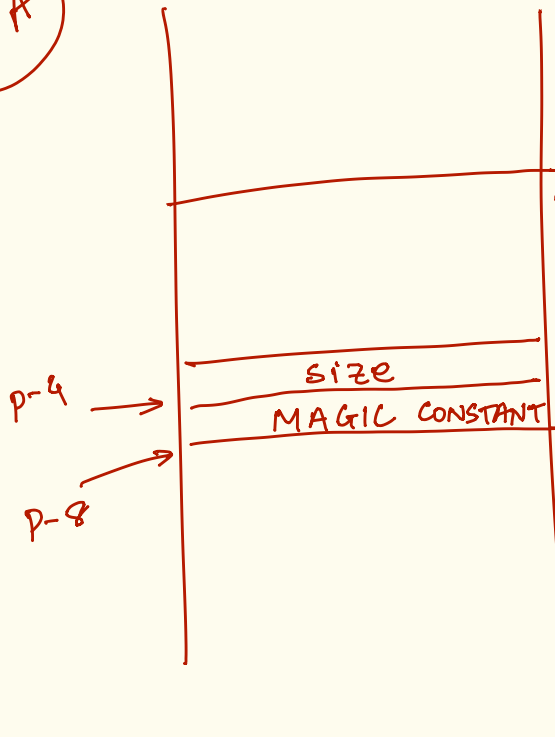
Q

(int*)
`*p = malloc(10 * sizeof(int));`

`*p = 23;` ✓

`*(p+15) = 23;` ✓ ?

VA



malloc (40 bytes)

} 40 bytes

← return addr of malloc

} malloc header

allocate

system calls

reserves (48 bytes)

free (p) {

size = *(p-4);

magic = *(p-8);

if (magic != MAGIC CONSTANT)

bad free!

else

free

deallocate

system call

for OS to update

process address usage metadata.

lazy allocation

no page table mappings!

in malloc path.

system call

fork / join

mmap (linux)



① — find unused address space (size > req. size)

② — update PCB to indicate new address space reservation/allocation

③ — build page table mappings! for new addresses reserved on heap.

⊛ lazy allocation!

on access on VA

⇒ page fault!

OS needs to identify a lazy allocation page fault.

$p = \text{malloc}(40 \text{ bytes})$

$*p = 23;$

$*(p+5000) = 23;$

lazy alloc
of 40 bytes

pte flags = $\underbrace{V=0}_{\text{no pte}}$

if (fault address is valid VA)

then ~~we~~ add ^{new} mapping for faulting address in page table

evict pages from self or other processes!

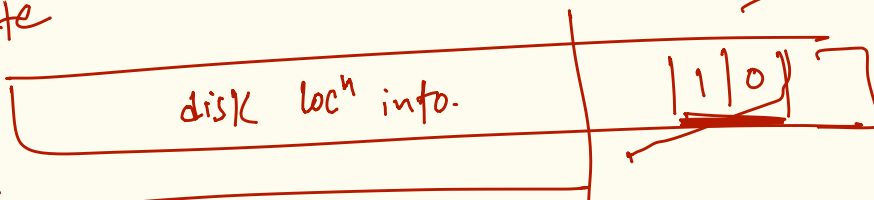
swapping ~ evict pages to disk

swap area

identify swapped pages

(i) pte: $V=1$ $P=0$ ~ $x86$
 $V=0$ ~ riscv
?? bits

(ii) pte



use this to read data from disk & copy to a free page, add page to page table.

⑧ malloc (40 bytes)

```
p = sbrk(48);
```

```
* (p+4) = 40;
```

```
* p = MAGIC CONSTANT;
```

```
return (p+8);
```

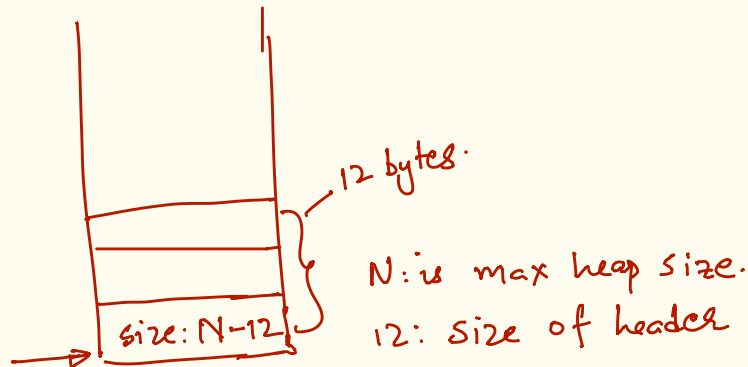
page fault!
on larry
allocation.

⑧ Managing the address space (of the process ^{and} the kernel).

let's assume heap
is empty.

(i)

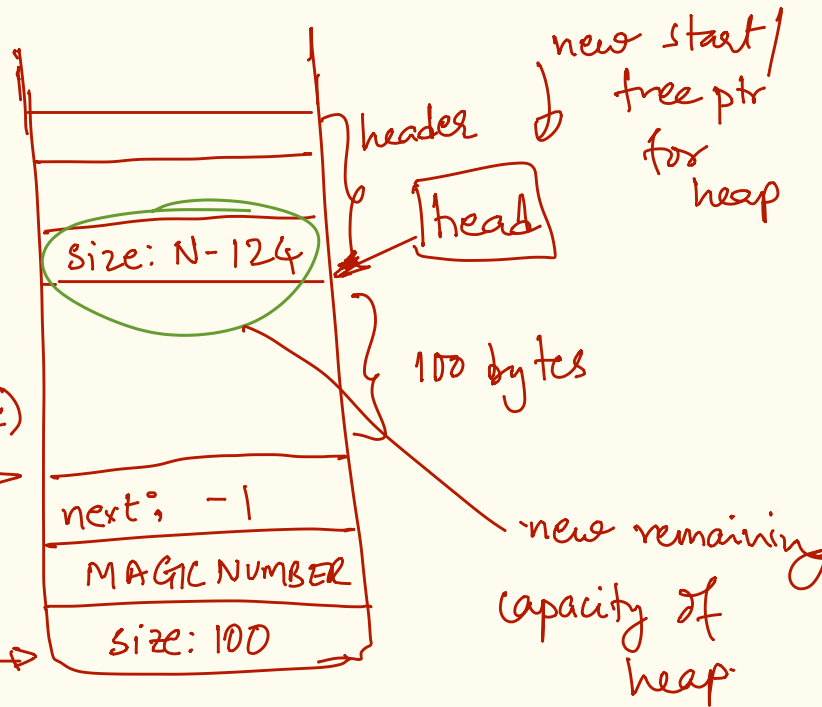
points to → head
a/ first free/unallocated
chunk of memory
in address space.



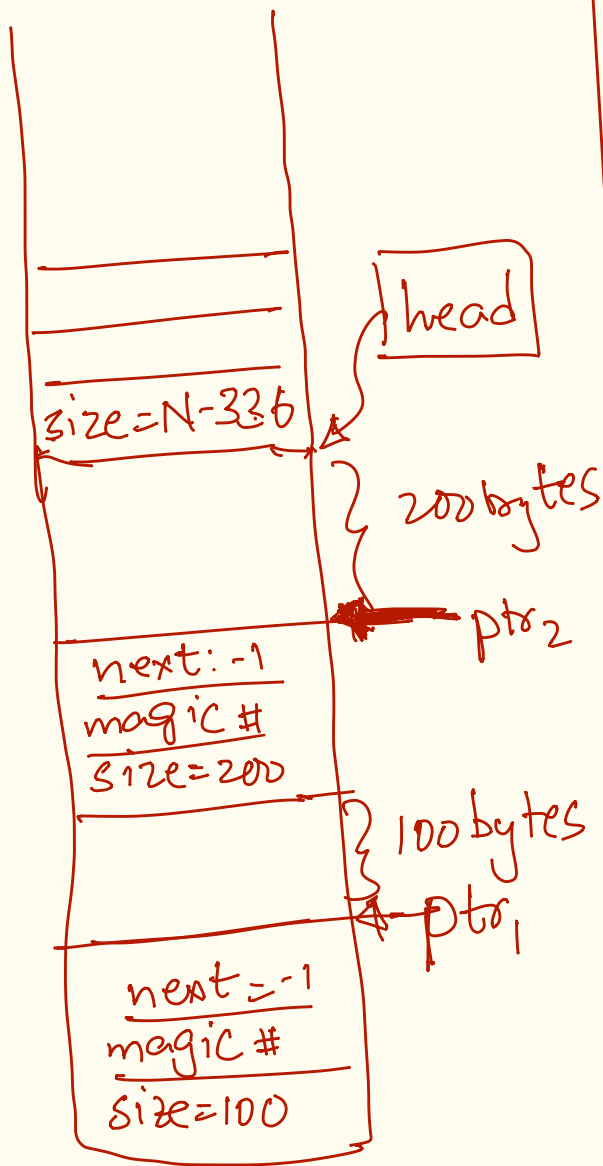
(ii) malloc of 100 bytes.

(address returned
by malloc)
ptr₁ →

address returned
by sbrk →



(iii) $ptr_2 = \text{malloc } 200 \text{ bytes}$



(iv) free (ptr₁)

