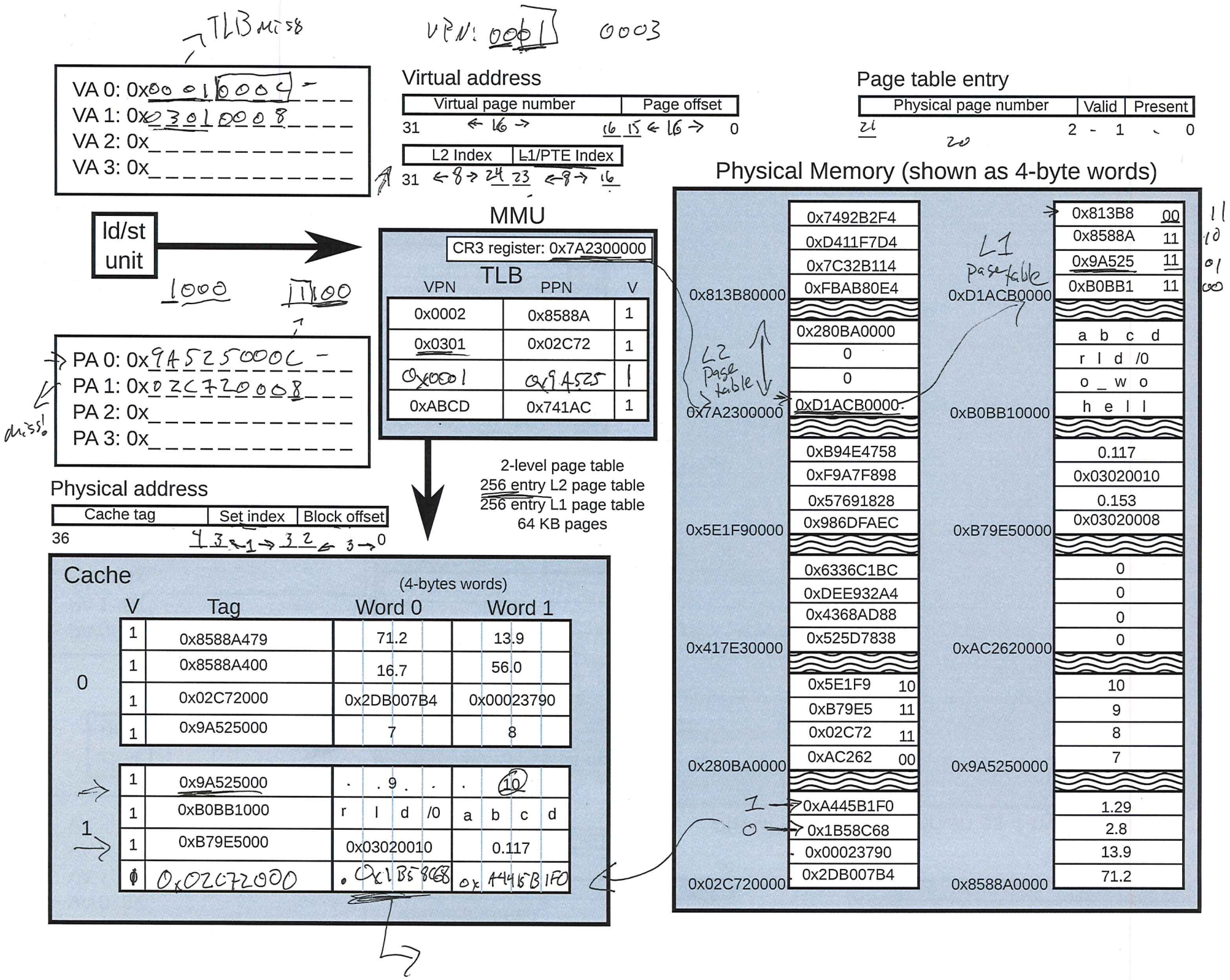

TLB miss
VPN: 0001
0003
VA 0: 0x00 01 0004
VA 1: 0x03 01 0008
VA 2: 0x
VA 3: 0x
Virtual address
Virtual page number
Page offset
31 ← 16 → 16 15 ← 16 → 0
L2 Index
L1/PTE Index
31 ← 8 → 24 23 ← 8 → 16
Page table entry
Physical page number
Valid
Present
21 2 - 1 0
20
ld/st unit
1000
11100
MMU
CR3 register: 0x7A2300000
TLB
VPN PPN V
0x0002 0x8588A 1
0x0301 0x02C72 1
0x0001 0x9A525 1
0xABCD 0x741AC 1
PA 0: 0x9A525000C
PA 1: 0x02C72008
PA 2: 0x
PA 3: 0x
miss!
Physical address
Cache tag
Set index
Block offset
36
13 → 4 3 → 2 1 → 0
2-level page table
256 entry L2 page table
256 entry L1 page table
64 KB pages
Physical Memory (shown as 4-byte words)
0x813B80000
0x7492B2F4
0xD411F7D4
0x7C32B114
0xFBAB80E4
0x280BA0000
0
0
0x7A2300000
0xD1ACB0000
0xB94E4758
0xF9A7F898
0x57691828
0x5E1F90000
0x986DFAEC
0x6336C1BC
0xDEE932A4
0x4368AD88
0x417E30000
0x525D7838
0x5E1F9 10
0xB79E5 11
0x02C72 11
0x280BA0000
0xAC262 00
0xA445B1F0
0x1B58C68
0x00023790
0x02C720000
0x2DB007B4
L1 page table
0xD1ACB0000
0x813B8 00
0x8588A 11
0x9A525 11
0xB0BB1 11
0xB0BB10000
a b c d
r l d /0
o _ w o
h e l l
0.117
0x03020010
0.153
0xB79E50000
0x03020008
0
0
0
0xAC2620000
0
10
9
8
0x9A5250000
7
1.29
2.8
13.9
0x8588A0000
71.2
Cache
(4-bytes words)
V Tag Word 0 Word 1
0
1 0x8588A479 71.2 13.9
1 0x8588A400 16.7 56.0
1 0x02C72000 0x2DB007B4 0x00023790
1 0x9A525000 7 8
1
1 0x9A525000 . . 9 . . 10
1 0xB0BB1000 r l d /0 a b c d
1 0xB79E5000 0x03020010 0.117
0 0x02C72000 0x1B58C68 0xA445B1F0

Managing the heap!

heap for ~~dynamic~~ dynamic memory allocation

C → malloc

Java → new

C++ → new

C/C++ we use explicit memory management

explicit → programmer is required to explicitly allocate + deallocate memory

C → malloc + free

Java: implicit → memory is garbage collected

---

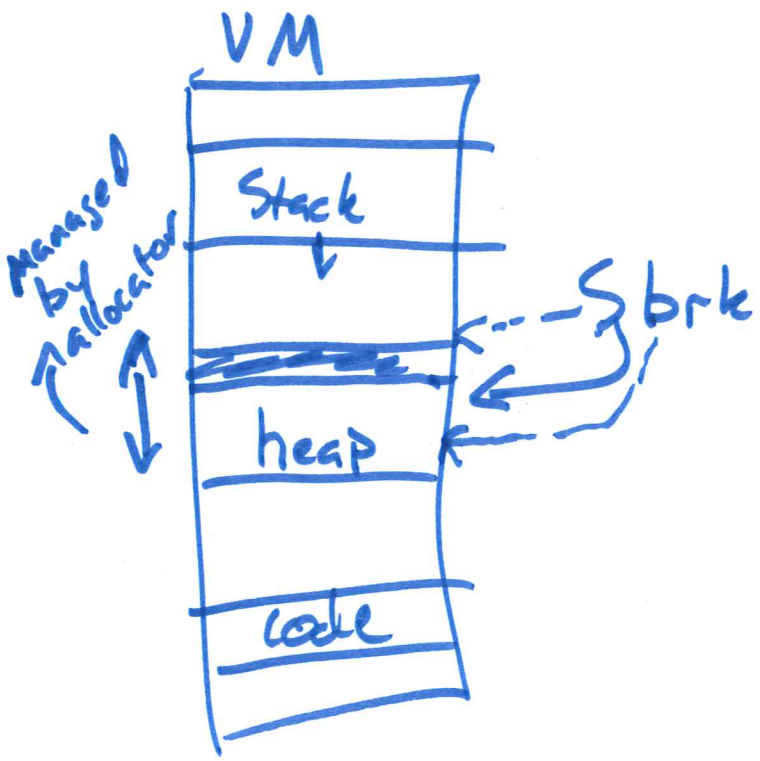
malloc (size) + free (void *)

↳ allocation ↳ de allocation

~~#~~ To get ↑ memory ~~you~~ malloc asks the OS

more virtual memory

Sbrk (pronounced S-ess-break)



System call

sbrk(incr)

↳ allocating new virtual memory

sbrk(4096)

sbrk(-10,000)

↳ freeing memory

When does OS update your page table?

sbrk is called + OS allocates pages in the page table + physical memory

---

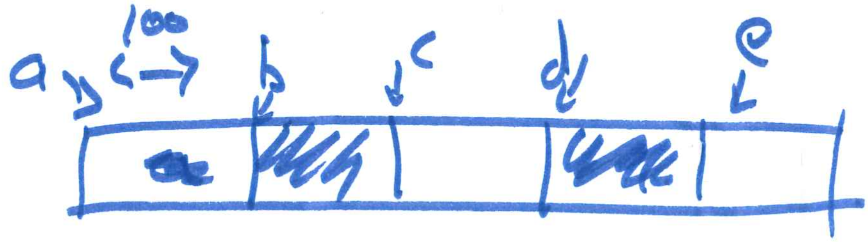
Goals for memory allocator

- track all allocated blocks
- handle arbitrary requests
- can't changes to old allocations
- it can only use the heap
- alignment restrictions

More about fragmentation

- <u>external</u> fragmentation of physical memory
  - ↳ there is enough free memory to satisfy the request <u>but</u> not ~~a~~ a single free block large enough

```
int *a = malloc(100)
    *b = malloc(100)
    *c =   "      "
    *d =   "      "
    *e =   "      "
free (b)
free (d)
malloc (~~200~~ 150)
```



→ there <u>is</u> 200B free but no block is 150B or larger

---

matrix[5][5] → a[5 + 5·10]

```
int ** matrix;
int * a = malloc((100·sizeof(int)); → 0x5000000 ↓ 0x6000000
for (int i=0; i<10; i++){
    Matrix[i] = a + 10·i;
}
```

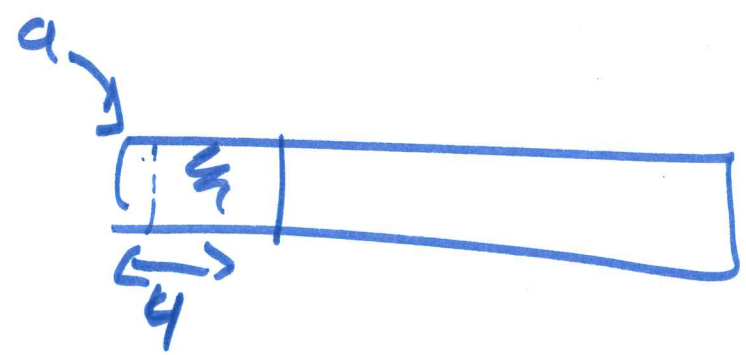
↳ address

an ~~assid~~ aside on why we can't change virtual addresses

internal fragmentation → within an allocated block we don't use all the space

usually occurs because of alignment restrictions

*a = malloc(1)



fragmentation reduces heap utilization

↳ $\frac{\text{memory used}}{\text{memory allocated}}$

---

building a memory allocator

4 things we need to do

1) how to track free blocks
2) how to choose free block when allocating
3) what to do w/ extra space in free block when allocating
4) What to do w/ blocks when deallocated