Last time...

- Memory
  - Stack vs. heap
- pointers
  - malloc

```
size_t len = 12;
int* arr = malloc(sizeof(int) * len);
```

- how do we get data from one block of memory to another?

Today:

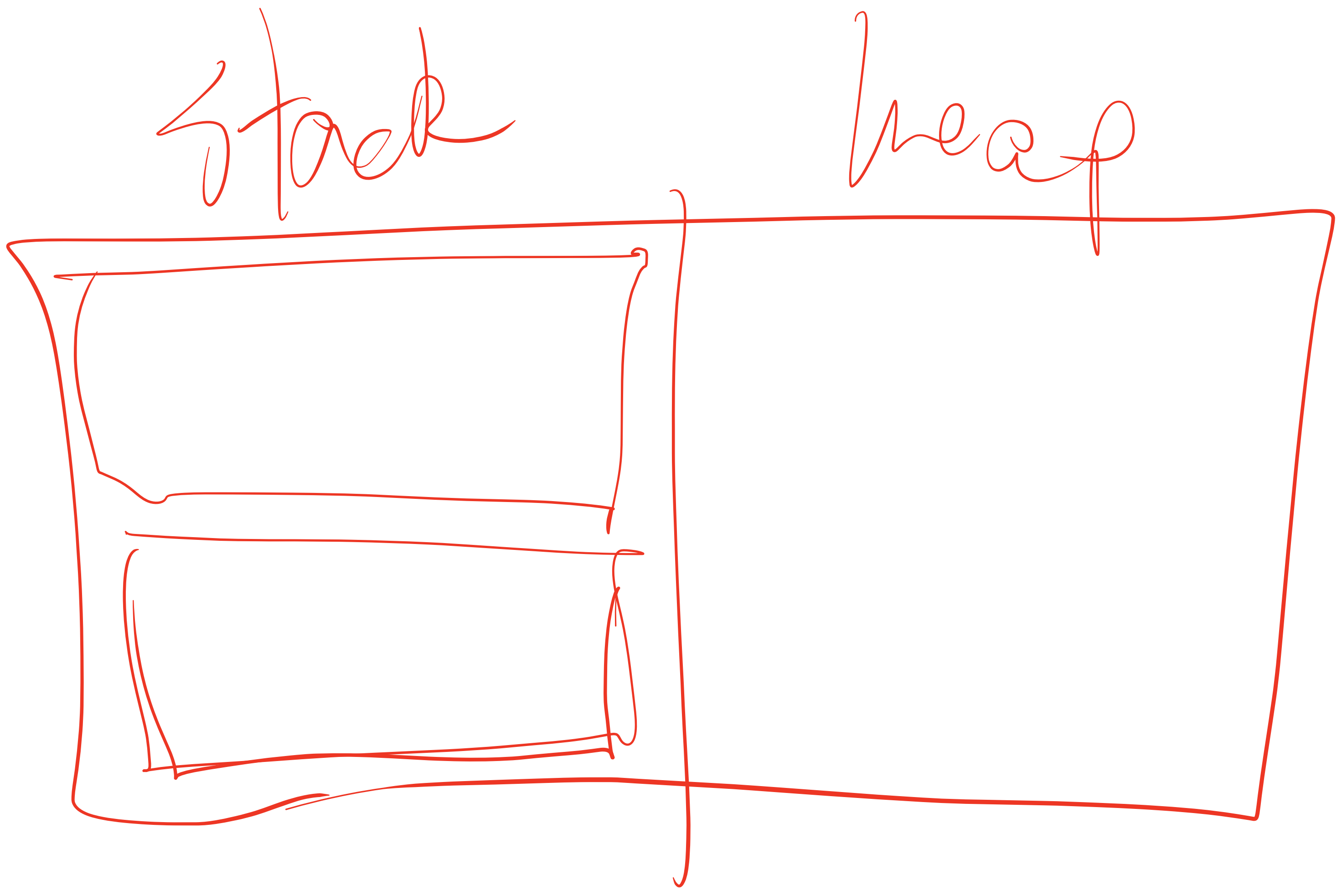
- copying arrays (more generally, memory)
  - what happens to the old array?
- calloc
- how to think about pointers/malloc
- resizing arrays
  - amortized analysis
- ADT teaser

```
int main() {
    size_t arrlen = 4;
    int *arr = malloc(sizeof(int)*arrlen);
    arr[0] = 75;
    for (size_t i = 0; i < arrlen; i++) {
        arr[i] = i * 3;
    }

    printf("%p\n", (void*)arr);

    // Make arr bigger
    arrlen *= 2;
    arr = malloc(sizeof(int)*arrlen);

    printf("%p\n", (void*)arr);
```

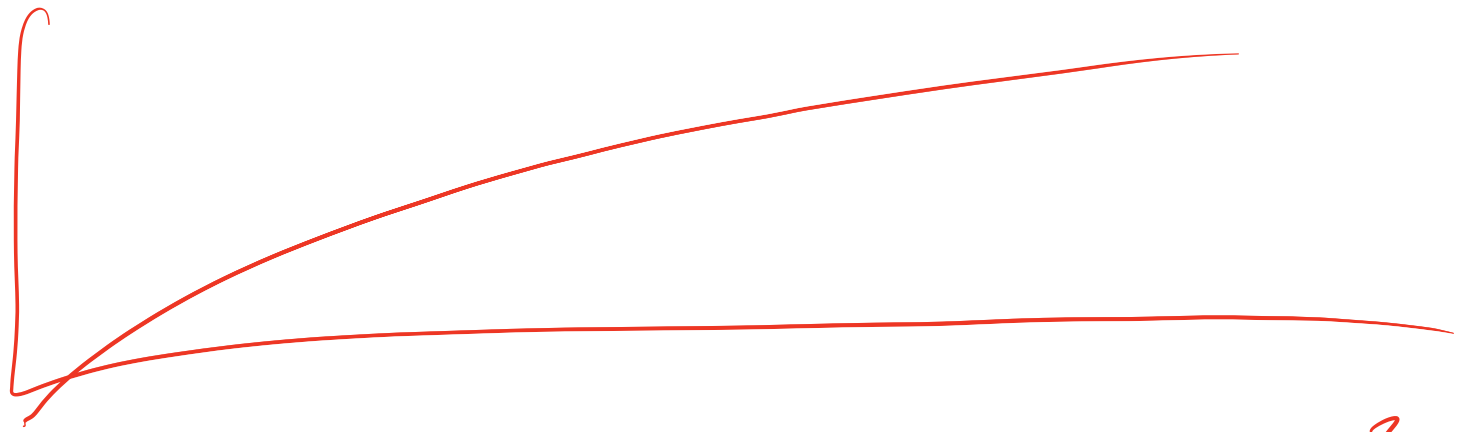


```
int f(int a, int b) {
    int x = a+b;
    return x-1;
}
```

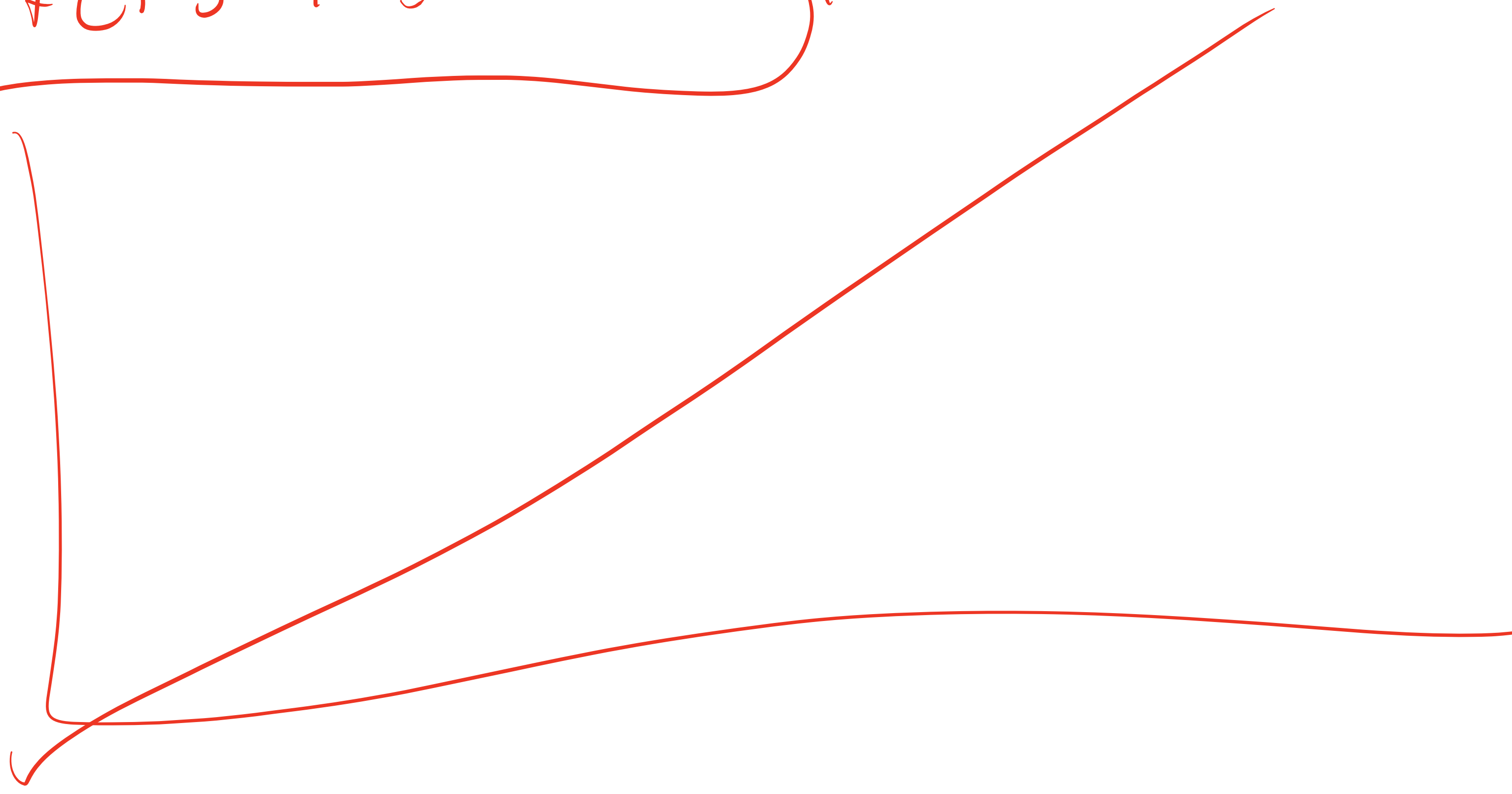
1. Allocate stack for 3 ints
2. Execute fn
3. Deallocate stack frame

$$\underbrace{1+2+4+\ldots}+1000000\text{th}$$

$= 2\text{million}$



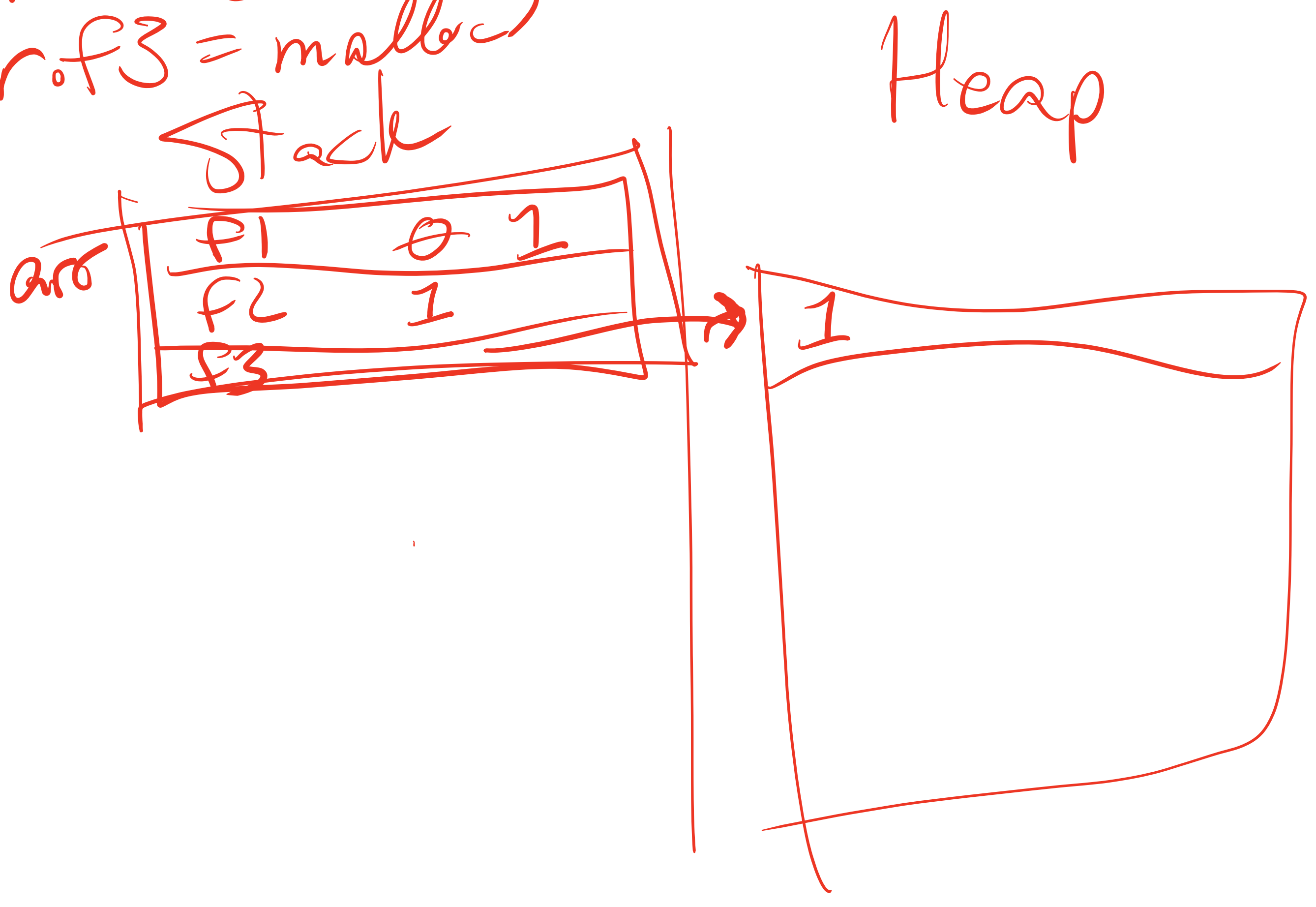
$$\underbrace{1+2+3+4\ldots}\ 1000000 = 1m^2$$



```
struct dA {
    f1;
    f2;
    f3;
    :
};
```

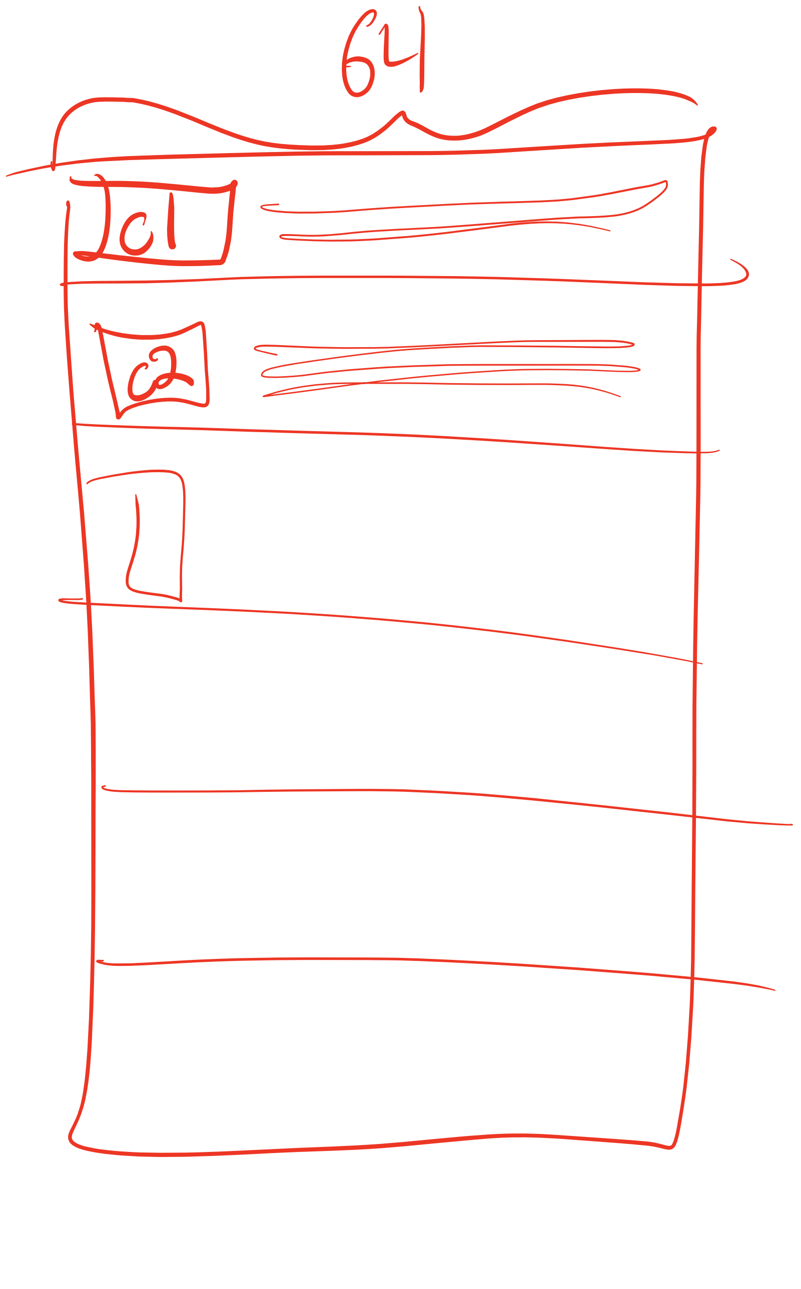
```
struct dA arr;
arr.f3 = malloc
```

f(&arr)

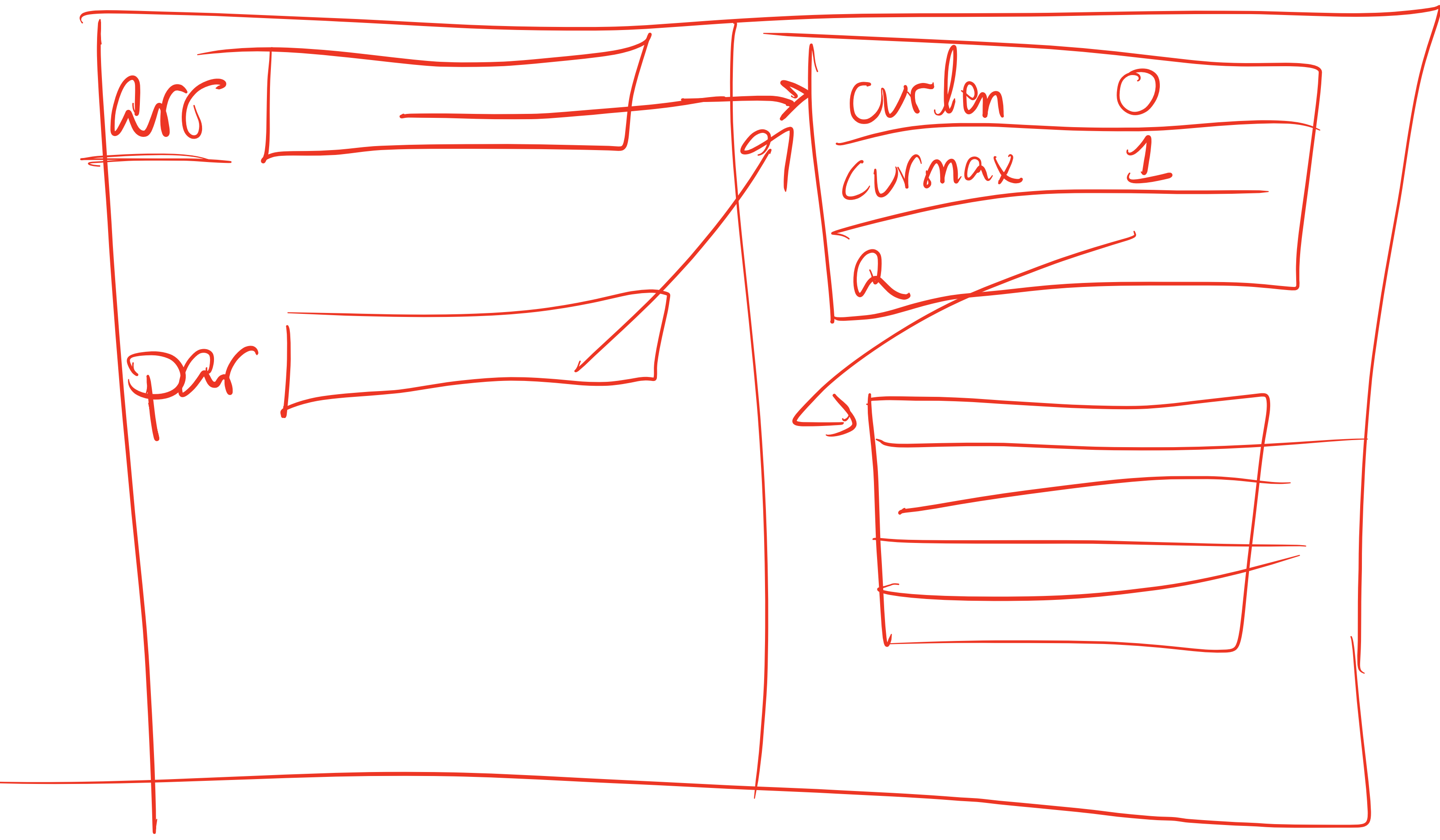


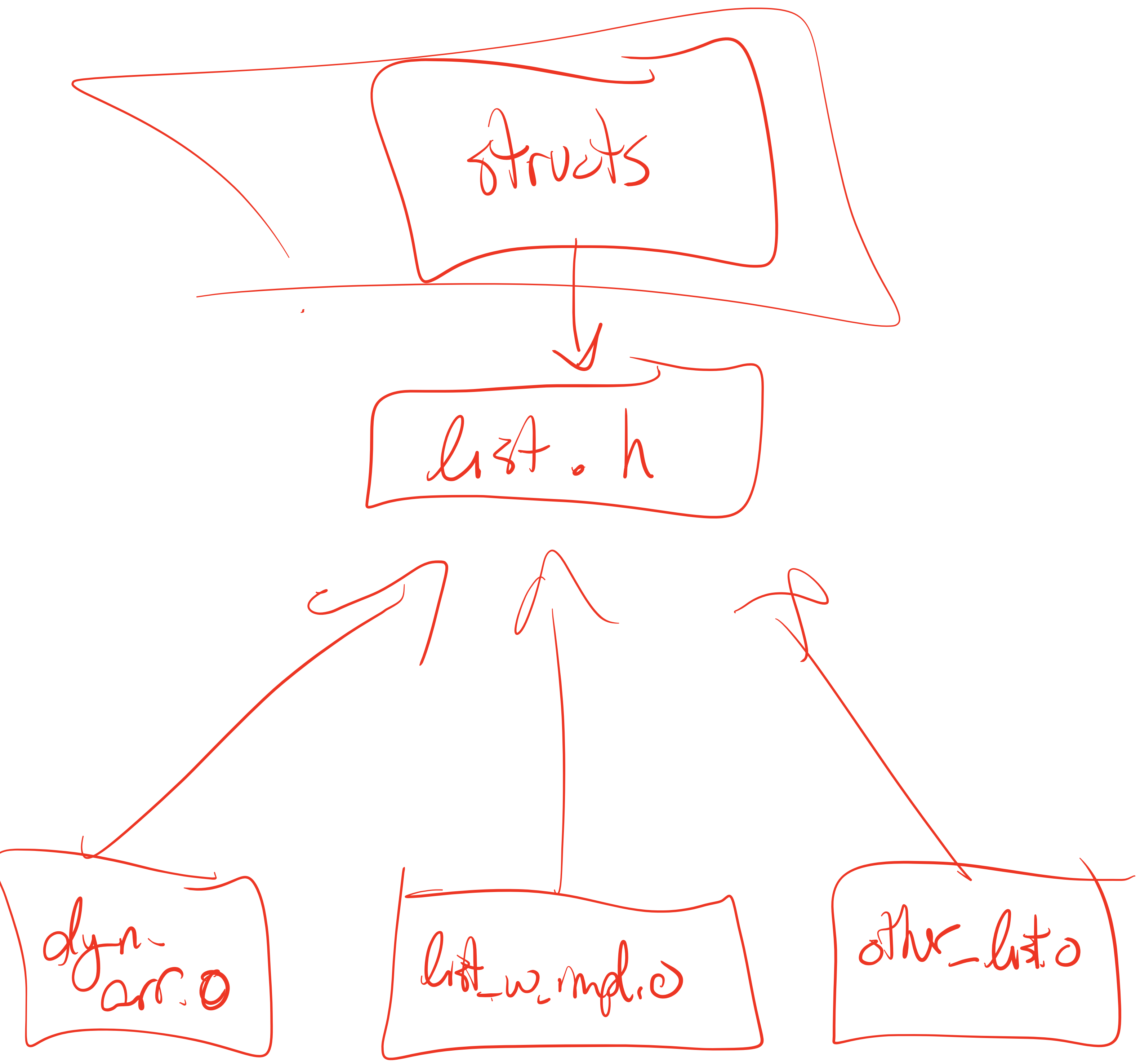
arr -> a $\equiv$ (*arr).a

$\not\equiv$ *arr.a



```
struct {
  char c1;
  char c2;
}
```


arr
par
curlen 0
curmax 1
a


structs
list.h
dyn_arr.o
list_w_mpl.o
other_list.o