CS 326-01 P1 Design, Local Dev
Advanced Heap

Bootstrapping

The problem: sh needs working malloc/free.

We have

```
void free(p)
void* malloc_name(n,s)
void* malloc(n)
```

→

Temp solution:

add malloc_name_given()
have malloc()
call
malloc_name_given()

add myfree(p)
have memtest call myfree
memtest calls
malloc_name()
myfree()

```
p = malloc(nbytes)
      \
       malloc_name(nbytes, name)

free(p)
```

```
p = malloc_name(nbytes, name)
    if (nbytes < BLOCK_MIN_SIZE)
            nbytes = BLOCK_MIN_SIZE;
    f = find_free_block()

    if (f == 0) {     // not found
        f = request_more_heap()
    }

    p = split_free_block(f, nbytes)

    return p;
```

```
request_more_heap()
{   amt = nbytes + sizeof(struct block_hdr)
    [Determine request_size]

    if (last block in block list is free)
        amt = amt - (sizeof(struct block_hdr))
                    + fp->size)

    }

    Determine request size based on amt

     sbrk(request_size)
     make new free block
    if (last block in block list is free)

        merge(last, new)
    }

    return free block.

}
```

```
void *split_free_block(fp, nbytes)
{

    if (we can split fp) {
        fp becomes used, update size
        make new free block
        add to block list
    } else {
        turn fp into used block
            keep size
    }

    determine p

    return p
}
```

free(p)

hp = p - (sizeof(struct block_hdr))

set used to 0

set name to empty string

---

hp = merge(hp, list_next(hp))

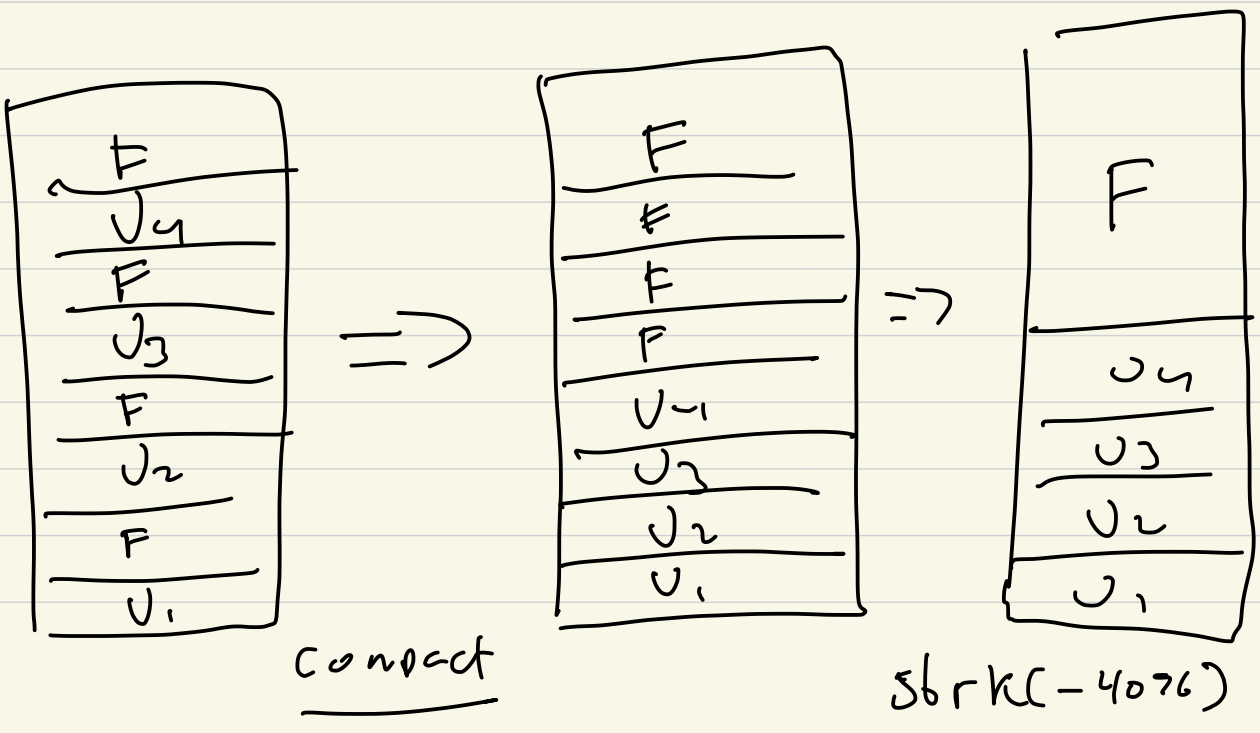
merge(list.prev(hp), hp)

3

---

heap allocation

explicit allocation



automatic deallocation

implicit deallocation

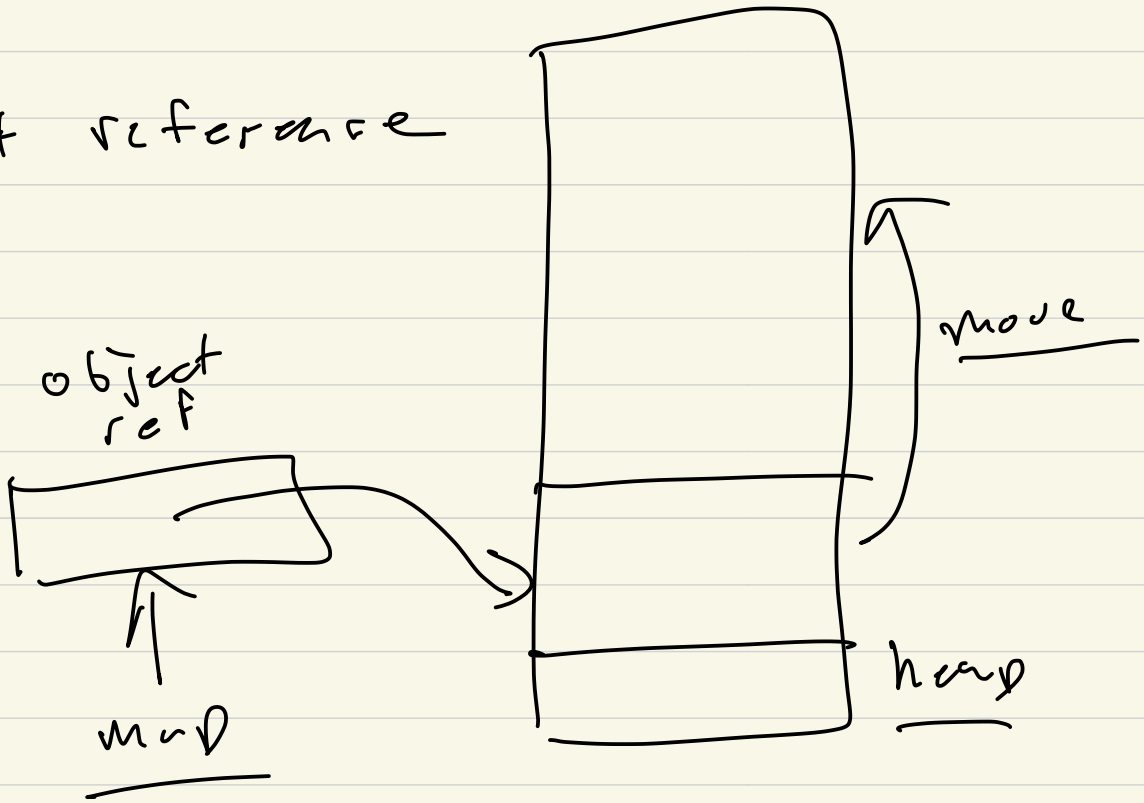
garbage collection

and

compaction

f = new Foo()

↑ object reference



Garbage Collection

Reference counting