A heap, a stack, a bottle and a rack
```
<table>
<thead>
<tr>
<th>Address Range</th>
<th>Permissions</th>
<th>Start Time</th>
<th>Size</th>
<th>Pathname</th>
</tr>
</thead>
<tbody>
<tr>
<td>562194b28000-562194b29000</td>
<td>r-xp</td>
<td>08:02</td>
<td>4719974</td>
<td>/home/csd/test-p</td>
</tr>
<tr>
<td>562194d28000-562194d29000</td>
<td>r--p</td>
<td>08:02</td>
<td>4719974</td>
<td>/home/csd/test-p</td>
</tr>
<tr>
<td>562194d29000-562194d2a000</td>
<td>rw-p</td>
<td>08:02</td>
<td>4719974</td>
<td>/home/csd/test-p</td>
</tr>
<tr>
<td>5621961ea000-56219620b000</td>
<td>rw-p</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
</tr>
<tr>
<td>7f6065546000-7f606572d000</td>
<td>r-xp</td>
<td>08:02</td>
<td>1966852</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
</tr>
<tr>
<td>7f606572d000-7f606592d000</td>
<td>r--p</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
</tr>
<tr>
<td>7f606592d000-7f6065931000</td>
<td>r--p</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
</tr>
<tr>
<td>7f6065931000-7f6065933000</td>
<td>rw-p</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/libc-2.27.so</td>
</tr>
<tr>
<td>7f6065933000-7f6065937000</td>
<td>rw-p</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>7f6065937000-7f606595e000</td>
<td>r-xp</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>7f6065b50000-7f6065b52000</td>
<td>rw-p</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>7f6065b5e000-7f6065b5f000</td>
<td>r--p</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>7f6065b5f000-7f6065b60000</td>
<td>rw-p</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>7f6065b60000-7f6065b61000</td>
<td>rw-p</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>7ffe70711000-7ffe70732000</td>
<td>rw-p</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>7ffe707d8000-7ffe707db000</td>
<td>r--p</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>7ffe707db000-7ffe707dd000</td>
<td>r-xp</td>
<td>00:00</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/ld-2.27.so</td>
</tr>
<tr>
<td>ffffffffffffffff6000000-ffffffffffffff6010000</td>
<td>r-xp</td>
<td>00:00</td>
<td>0</td>
<td>[stack]</td>
</tr>
<tr>
<td>ffffffffffffffff6000000-ffffffffffffff6010000</td>
<td>r-xp</td>
<td>00:00</td>
<td>0</td>
<td>[vvar]</td>
</tr>
<tr>
<td>ffffffffffffffff6000000-ffffffffffffff6010000</td>
<td>r-xp</td>
<td>00:00</td>
<td>0</td>
<td>[vdso]</td>
</tr>
<tr>
<td>ffffffffffffffff6000000-ffffffffffffff6010000</td>
<td>r-xp</td>
<td>00:00</td>
<td>0</td>
<td>[syscall]</td>
</tr>
</tbody>
</table>
```
The Stack

- The working memory.

Stack Pointer →

Frame Pointer →

Locals of DrawLine

Return Address

Parameters for DrawLine

Locals of DrawSquare

Return Address

Parameters for DrawSquare
“Canary Birds” (and other stuff on the stack)

- Can be hard to identify all objects on the stack
- Canary birds: To ensure nothing is broken
- Other scap data: To align memory
- Can help to optimize compile (-O) if you want to identify all items
The heap

- `malloc`
- When we need more permanent data
- Called “dynamic data” in previous figure
Exam question 1

What is done in the procedure below and where should gurka be allocated?
Why? Complete the code so that gurka is allocated space.

```c
void tomat(int *a, int *b) {
    // allocate room for gurka

    gurka = *a;
    *a = *b;
    *b = gurka;
}
```
What is done in the procedure below and where should `gurka` be allocated? Why? Complete the code so that `gurka` is allocated space.

```c
int *tomat(int *a, int *b) {
    // allocate room for `gurka`

    *gurka = *a + *b;
    return gurka;
}
```
In the code below we have allocated three arrays where one is on the heap, which array and why is it allocated on the heap and not on the stack?

```c
#include <stdlib.h>
#include <stdio.h>

#define MAX 4

int h[MAX];

int *foo(int *a, int *b, int s) {
    int *r = malloc(s * sizeof(int));
    for(int i = 0; i < s; i++) {
        r[i] = a[i] + b[i];
    }
    return r;
}

int main() {
    int f[MAX];
    for(int i = 0; i < MAX; i++) {
        f[i] = i;
        h[i] = i*10;
    }
    int *g = foo(f, h, 4);
    printf("a[2] + b[2] is %d\n", g[2]);
    return 0;
}
```
1.2 memory map [2 points]

Below is a, somewhat shortened, printout of a memory mapping of a running process. Briefly describe the role of each segment marked with ???.

```
> cat /proc/13896/maps

00400000-00401000 r-xp 00000000 08:01 1723260  .../gurka ???
00600000-00601000 r--p 00000000 08:01 1723260  .../gurka ???
00601000-00602000 rw-p 00001000 08:01 1723260  .../gurka ???
022fa000-0231b000 rw-p 00000000 00:00 0 [???
7f6683423000-7f66835e2000 r-xp 00000000 08:01 3149003  .../libc-2.23.so ???

: 7ffd60600000-7ffd60621000 rw-p 00000000 00:00 0 [???
7ffd60648000-7ffd6064a000 r--p 00000000 00:00 0 [vvar]
7ffd6064a000-7ffd6064c000 r-xp 00000000 00:00 0 [vdso]
ffffffffffff600000-ffffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
```
The first three segments are: **code**, **read-only data** and **global data** for the running process gurka.

Then there is a segment for the **heap**.

The segment marked with lib-2.23.so is a **shared library**.

In the uppermost region we find the segment of the **stack**.
It is not completely defined what will happen if we run the code below. What is it that we do wrong and what could a possible effect be?

```c
int main() {

    char *heap = malloc(20);
    *heap = 0x61;
    printf("heap pointing to: 0x%x\n", *heap);
    free(heap);

    char *foo = malloc(20);
    *foo = 0x62;
    printf("foo pointing to: 0x%x\n", *foo);

    *heap = 0x63;
    printf("foo pointing to: 0x%x\n", *foo);

    return 0;
}
```
Below we see a program that will print the content of the stack.

```c
void zot(unsigned long *stop, int a1, int a2, int a3, int a4, int a5, int a6) {
    unsigned long r = 0x456;
    unsigned long *i;
    for(i = &r; i <= stop; i++){
        printf("%p  0x%lx\n", i, *i);
    }
}

int main() {
    unsigned long p = 0x123;

    zot(&p, 1, 2, 3, 4, 5, 6);
    back:
    printf(" back: \%p \n", &back);
    return 0;
}
```

When executed we see the following print out. Describe the values indicated with arrows (<-).

```
0x7ffebe3331f58   0x456
0x7ffebe3331f60   0x7ffebe3331f60  <--- ??
0x7ffebe3331f68   0x3a7dbfad7df4b100
0x7ffebe3331f70   0x7ffebe3331fa0
0x7ffebe3331f78   0x400663  <--- ??
0x7ffebe3331f80   0x6  <--- ??
0x7ffebe3331f88   0x4004a0
0x7ffebe3331f90   0x123
back: 0x400667
```
You have written the program below to examine what is on the stack.

```c
void zot(unsigned long *stop ) {
    unsigned long r = 0x3;
    unsigned long *i;
    for(i = &r; i <= stop; i++) { printf("%p 0x%lx\n", i, *i); } 
}

void foo(unsigned long *stop ) {
    unsigned long q = 0x2;
    zot(stop);
}

int main() {
    unsigned long p = 0x1;
    foo(&p);
    back:
    printf(" p: %p \n", &p);
    printf(" back: %p \n", &back);
    return 0;
}
```

Output:
```
0x7ff ca03d1748 0x3
0x7ff ca03d1750 0x7ff ca03d1750
0x7ff ca03d1758 0xb93d7906926a7d00
0x7ff ca03d1760 0x7ff ca03d1790
0x7ff ca03d1768 0x55cdac31d78c
0x7ff ca03d1770 0x7ff ca03d17d8
0x7ff ca03d1778 0x7ff ca03d17b0
0x7ff ca03d1780 0x1
0x7ff ca03d1788 0x2
0x7ff ca03d1790 0x7ff ca03d17c0
0x7ff ca03d1798 0x55cdac31d7c2
0x7ff ca03d17a0 0x55cdac31d810
0x7ff ca03d17a8 0x12acac31d5f0
0x7ff ca03d17b0 0x1
p: 0x7ff ca03d17b0
back: 0x55cdac31d7c2
```
Relevant Sections from the Book

For reading at your own time

- [Chapter 13](#) - The Abstraction: Address Spaces
- [Chapter 14](#) - Interlude: Memory API