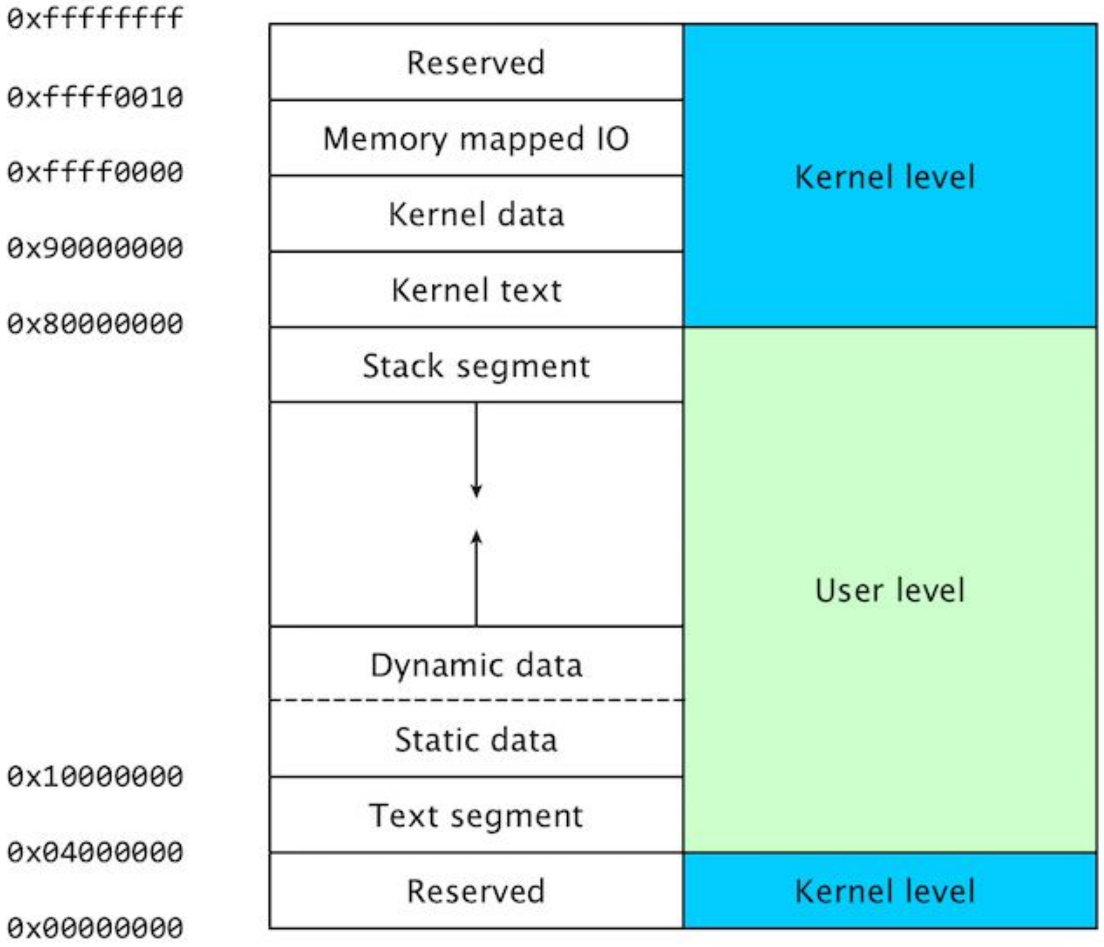
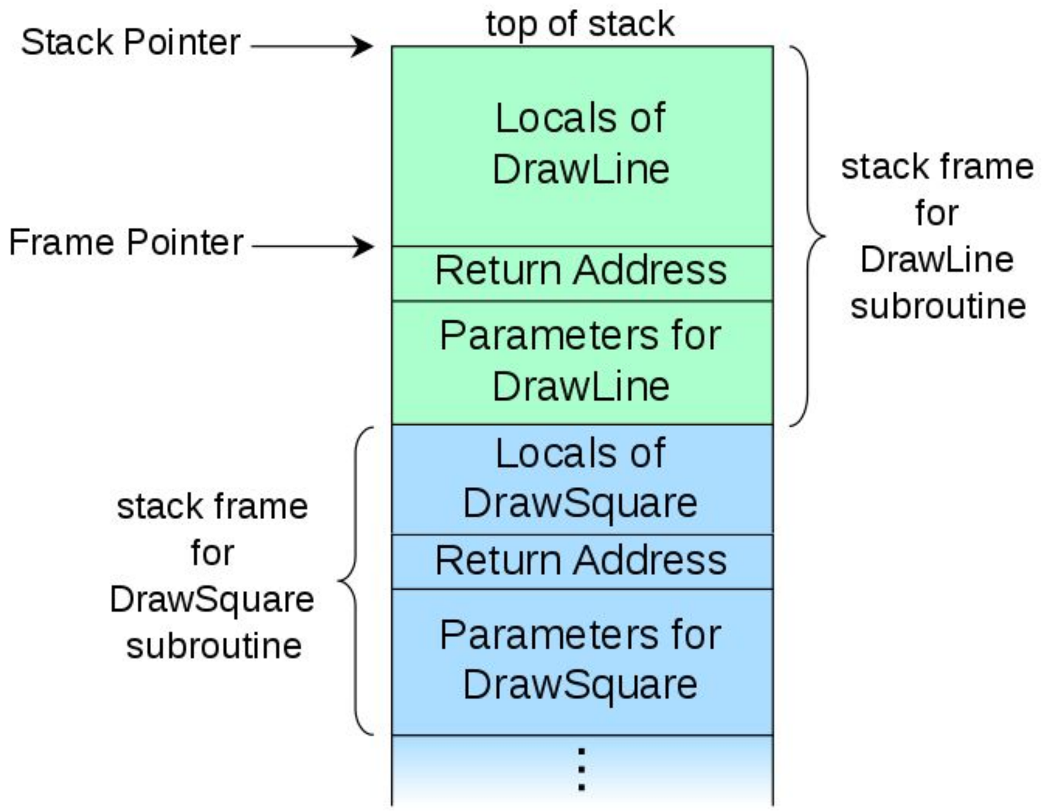
A heap, a stack, a bottle and a rack


0xffffffff
0xffff0010
0xffff0000
0x90000000
0x80000000
0x10000000
0x04000000
0x00000000
Reserved
Memory mapped IO
Kernel data
Kernel text
Stack segment
Dynamic data
Static data
Text segment
Reserved
Kernel level
User level
Kernel level

The Stack


top of stack
Stack Pointer
Frame Pointer
Locals of DrawLine
Return Address
Parameters for DrawLine
stack frame for DrawLine subroutine
Locals of DrawSquare
Return Address
Parameters for DrawSquare
stack frame for DrawSquare subroutine
⋮

Canary Birds

The heap

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]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0    [vsyscall]
```

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?

```
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.

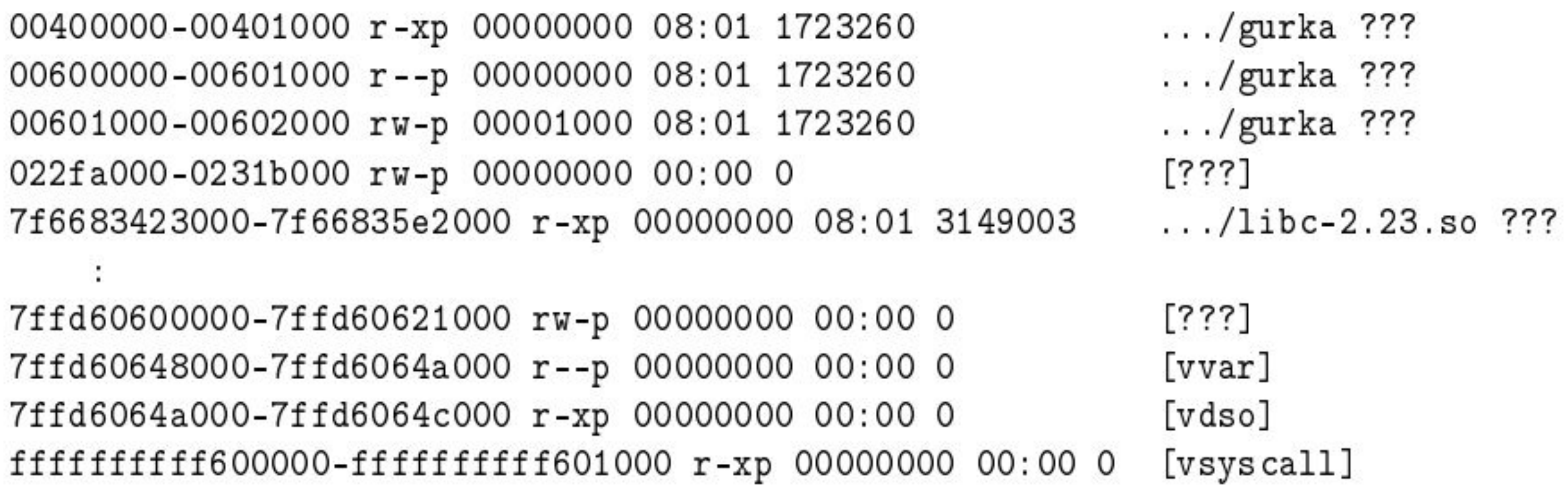
```
void zot(unsigned long *stop, int a1, int a2, int a3, int a4, int a5, int a6) {
  unsigned long r = 0x456;
  unsigned long *i;
  for(i = &r;  i <= stop; i++){
    printf("%p      0x%lx\n", i, *i);
  }
}

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 (<--).

```
0x7ffeb3331f58      0x456
0x7ffeb3331f60      0x7ffeb3331f60   <-- ??
0x7ffeb3331f68      0x3a7dbfad7df4b100
0x7ffeb3331f70      0x7ffeb3331fa0
0x7ffeb3331f78      0x400663    <-- ??
0x7ffeb3331f80      0x6   <-- ??
0x7ffeb3331f88      0x4004a0
0x7ffeb3331f90      0x123
  back: 0x400667
```



```
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]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0   [vsyscall]
```

Answer:

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 nd the segment of the **stack**.

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?

```
#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;
}
```