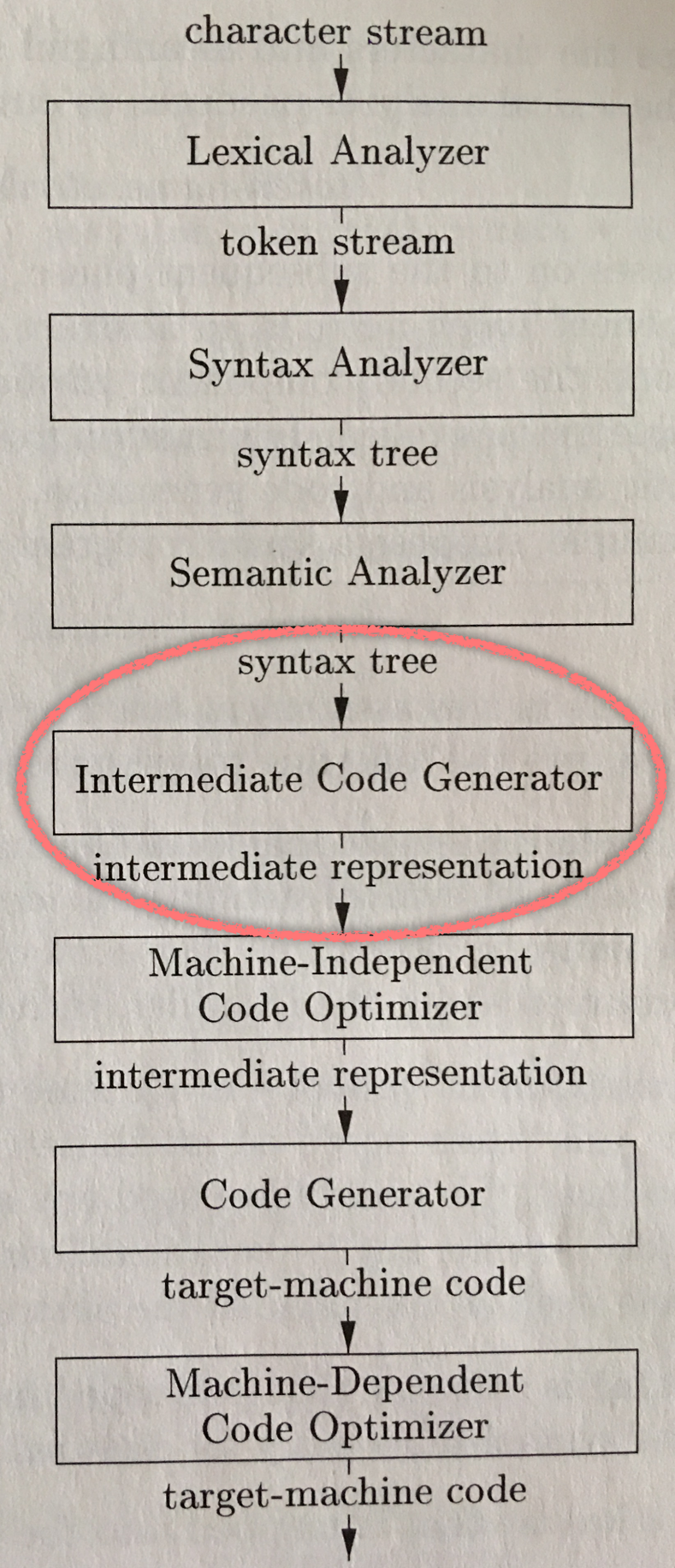
CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

Phases



Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text

Reminder:
Friday is a workshop day

- Come ready to work
- Ensure all team members are present
- Bring along a list of questions

function calls

Function calls

```
int foo(int x, int y) {
    int temp;
    temp = 2 * x + 3 * y;
    return temp;
}

int main() {
    ...
    int a = ...
    int b = ...
    int c = foo(a * b, a + b);
    ...
}
```

What happens during function call?

Function calls

- Basic form: id(e1,e2,...,ek)

Function calls

- Basic form: id(e1,e2,...,ek)
- General form: assignable(e1,e2,...,ek)
  - If f is a function, g(4,5) yields a function, and r.h yields a function, then the following are legal:

    f(3) g(4,5)(3) r.h(3)

How is function call carried out?

1. evaluate each of the argument expressions
2. mark the resulting values as parameters
3. invoke the function

How is function call carried out?

1. evaluate each of the argument expressions

   use compiler-generated temporaries

2. mark the resulting values as parameters

   use 'param' IR instruction

3. invoke the function

   use 'call(f,n)' IR instruction:
   f is a function
   n is arity of function

   # of parameters

examples

$f(x+1)$

examples

f(x+1)

Remember that the function call has structure.

examples

f(x+1)

t1 = x + 1

Generate code for the argument expression

examples

```
f(x+1)

t1 = x + 1
param t1
```

Mark the result as a parameter of the function call

examples

f(x+1)

```
t1 = x + 1
param t1
t2 = call(f,1)
```

Call the function. The second argument of the call indicates the arity of the function (i.e. how many parameters it has)

examples

f(x+1)

```
t1 = x + 1
param t1
t2 = call(f,1)
```

f(x+1,2*y)

examples

f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)

f(x+1,2*y)

Remember that the function call has structure.

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
```

Evaluate the first argument expression.

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
```

Mark the result as a parameter.

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
```

Evaluate the second argument expresssion.

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
```

Mark the result as a parameter.

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)
```

Call the function.

examples

f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)

f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)

f(g(3*z),h(a+b,a*b))

A slightly more involved example.

exercise

f(g(3*z),h(a+b,a*b))

What intermediate code do you come up with for this example?

examples

f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)

f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)

f(g(3*z),h(a+b,a*b))

As before,
remember the
structure...

examples

f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)

f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)

g(3*z)

...view this as a function call in isolation.

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)
```

```
g(3*z)

t1 = 3 * z
```

...first compute the argument value...

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)
```

```
g(3*z)

t1 = 3 * z
param t1
t2 = call(g,1)
```

...then mark t1 as a parameter and call the function g.

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)
```

```
f(g(3*z),h(a+b,a*b))

t1 = 3 * z
param t1
t2 = call(g,1)
```

This translation will happen automatically due to the recursive structure of the function call for f...

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)
```

```
f(g(3*z),h(a+b,a*b))

t1 = 3 * z
param t1
t2 = call(g,1)
param t2
```

Mark the result as a parameter.

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)
```

```
f(g(3*z),h(a+b,a*b))

t1 = 3 * z
param t1
t2 = call(g,1)
param t2
```

More structure!

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)
```

```
f(g(3*z),h(a+b,a*b))

t1 = 3 * z
param t1
t2 = call(g,1)
param t2
t3 = a + b
```

expression

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)
```

```
f(g(3*z),h(a+b,a*b))

t1 = 3 * z
param t1
t2 = call(g,1)
param t2
t3 = a + b
param t3
```

parameter marking

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)
```

```
f(g(3*z),h(a+b,a*b))

t1 = 3 * z
param t1
t2 = call(g,1)
param t2
t3 = a + b
param t3
t4 = a * b
```

expression

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)
```

```
f(g(3*z),h(a+b,a*b))

t1 = 3 * z
param t1
t2 = call(g,1)
param t2
t3 = a + b
param t3
t4 = a * b
param t4
t5 = call(h,2)
```

parameter marking
and call

examples

```
f(x+1)

t1 = x + 1
param t1
t2 = call(f,1)
```

```
f(x+1,2*y)

t1 = x + 1
param t1
t2 = 2 * y
param t2
t3 = call(f,2)
```

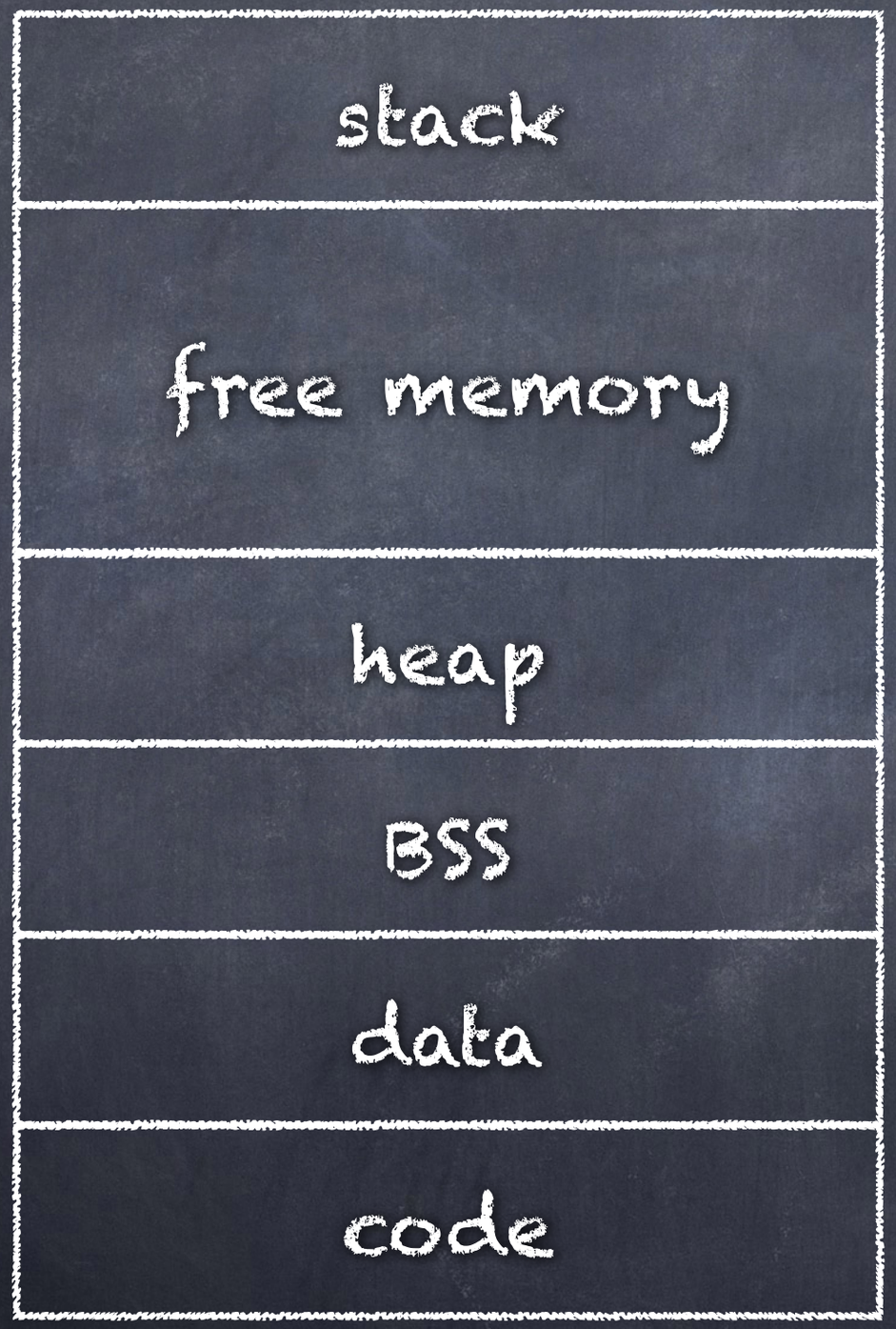
```
f(g(3*z),h(a+b,a*b))

t1 = 3 * z
param t1
t2 = call(g,1)
param t2
t3 = a + b
param t3
t4 = a * b
param t4
t5 = call(h,2)
param t5
t6 = call(f,2)
```

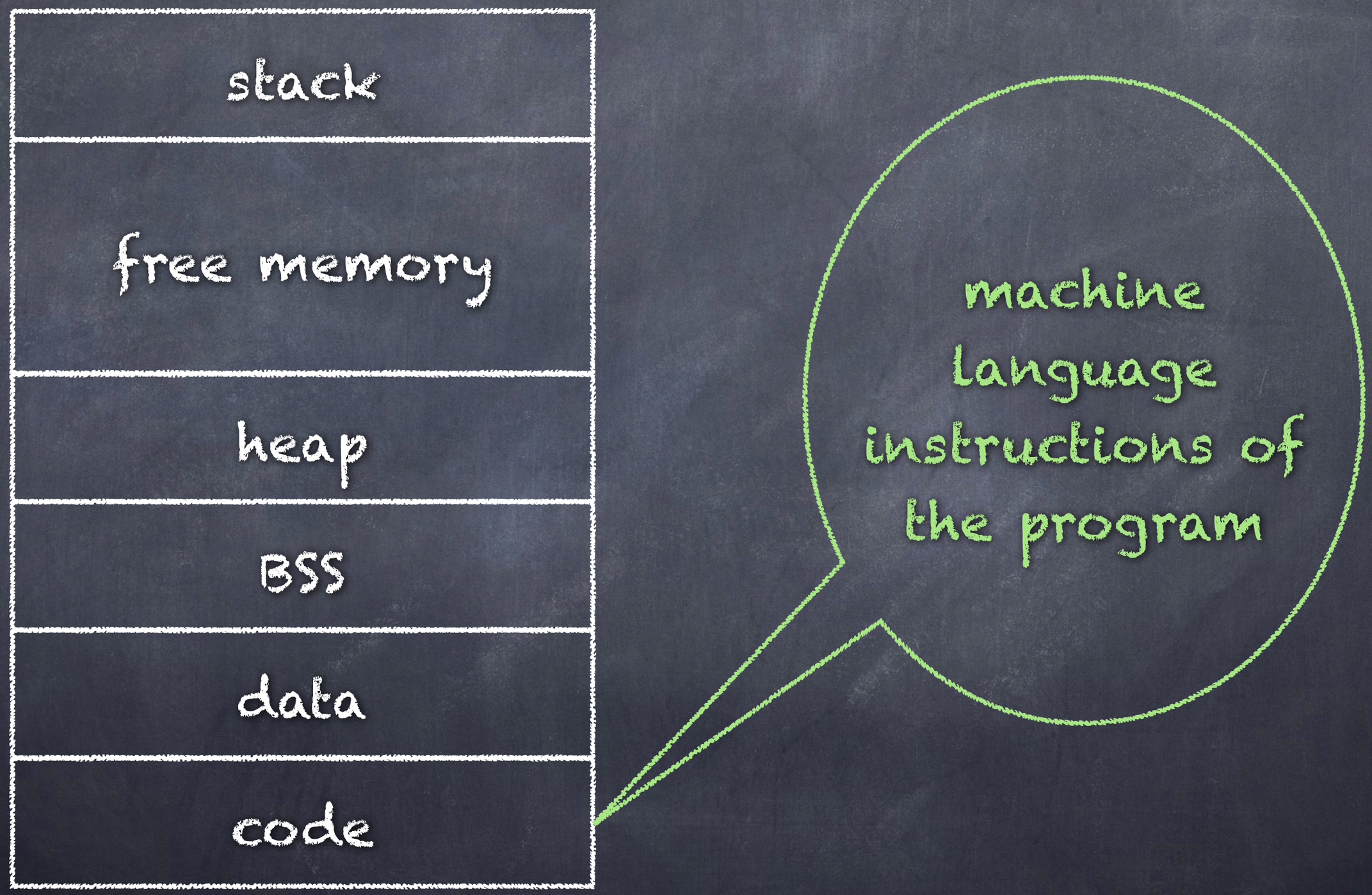
parameter marking and call

Memory Organization

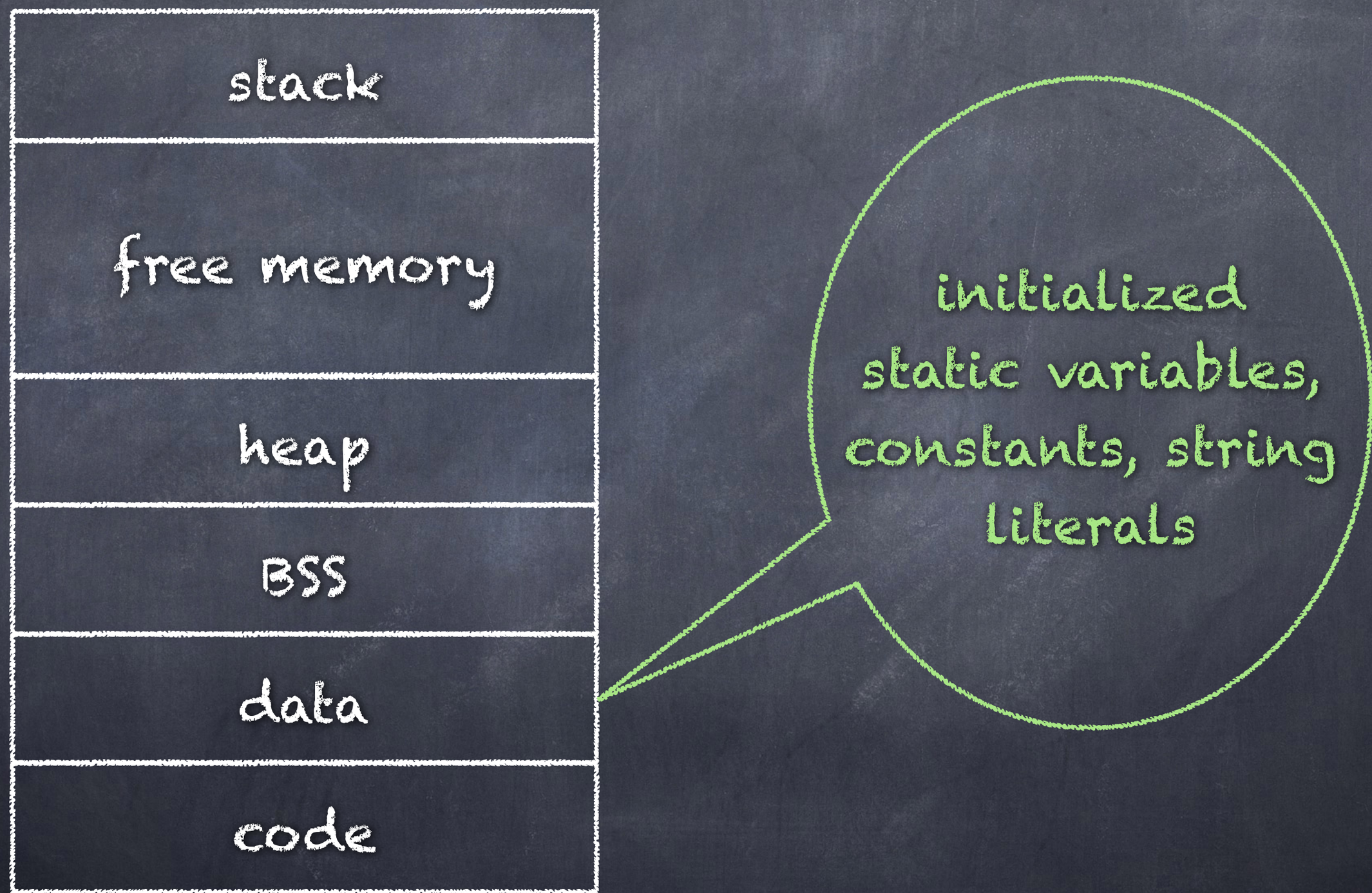
Memory organization



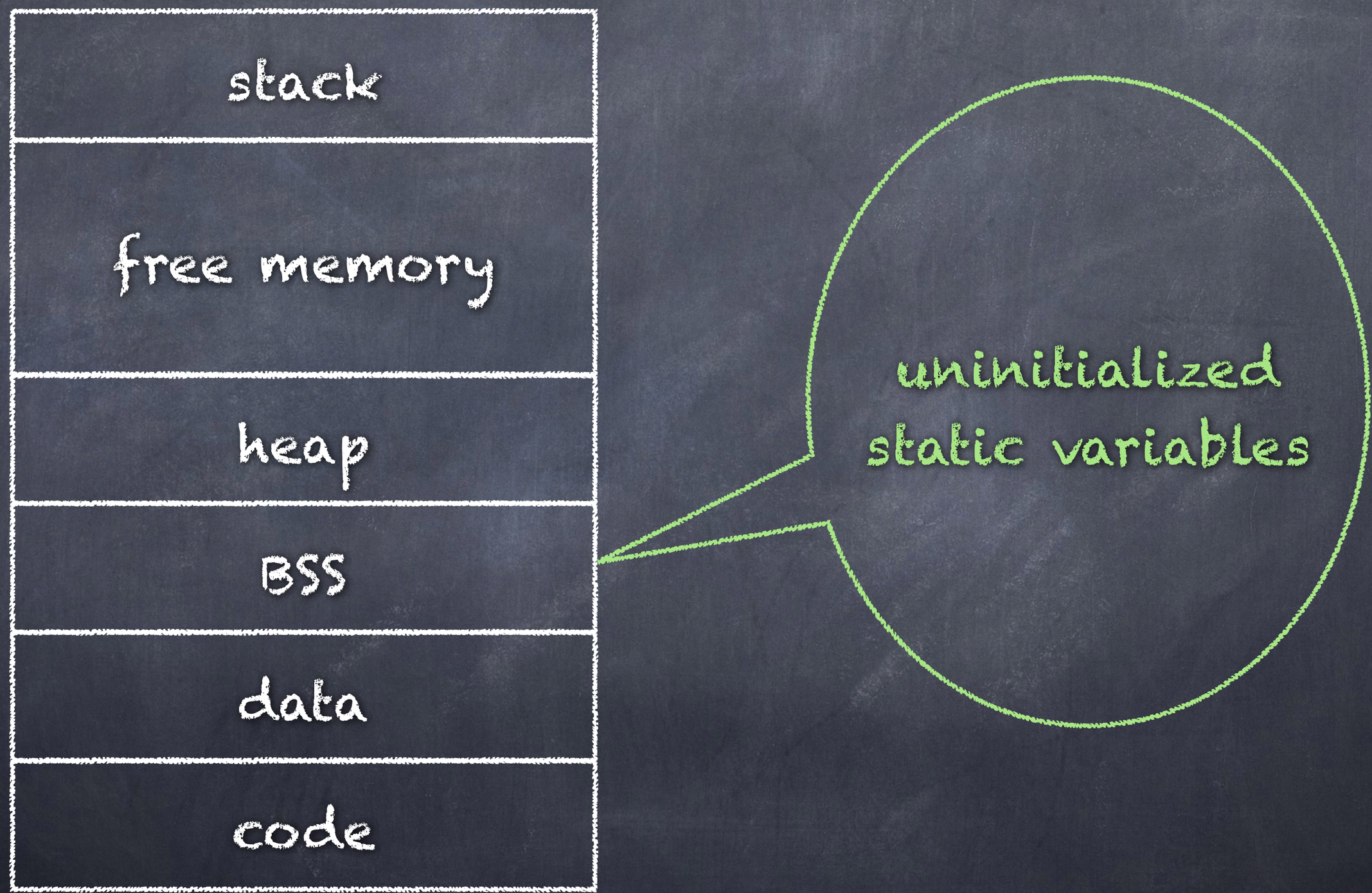
Memory organization



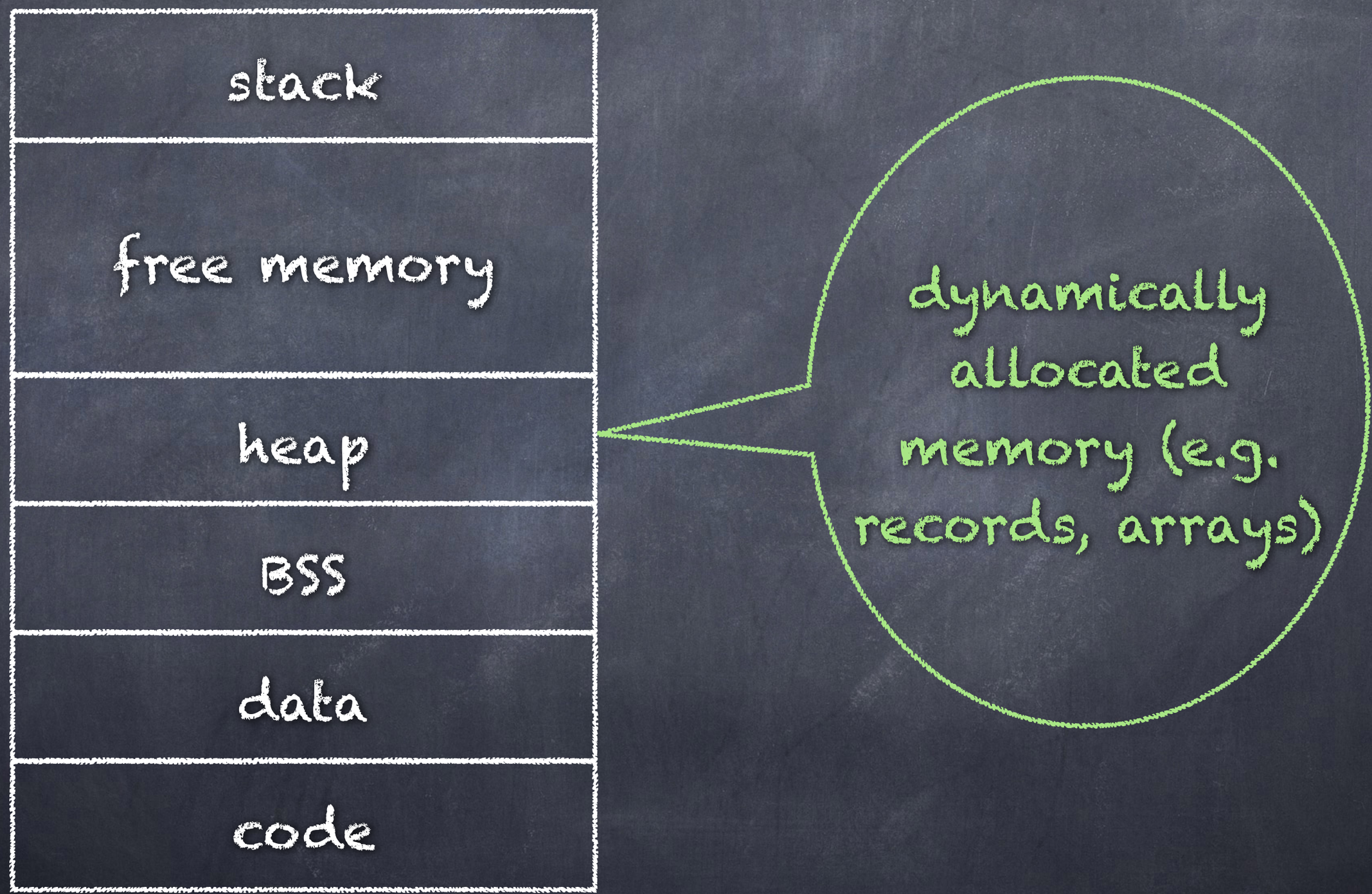
Memory organization



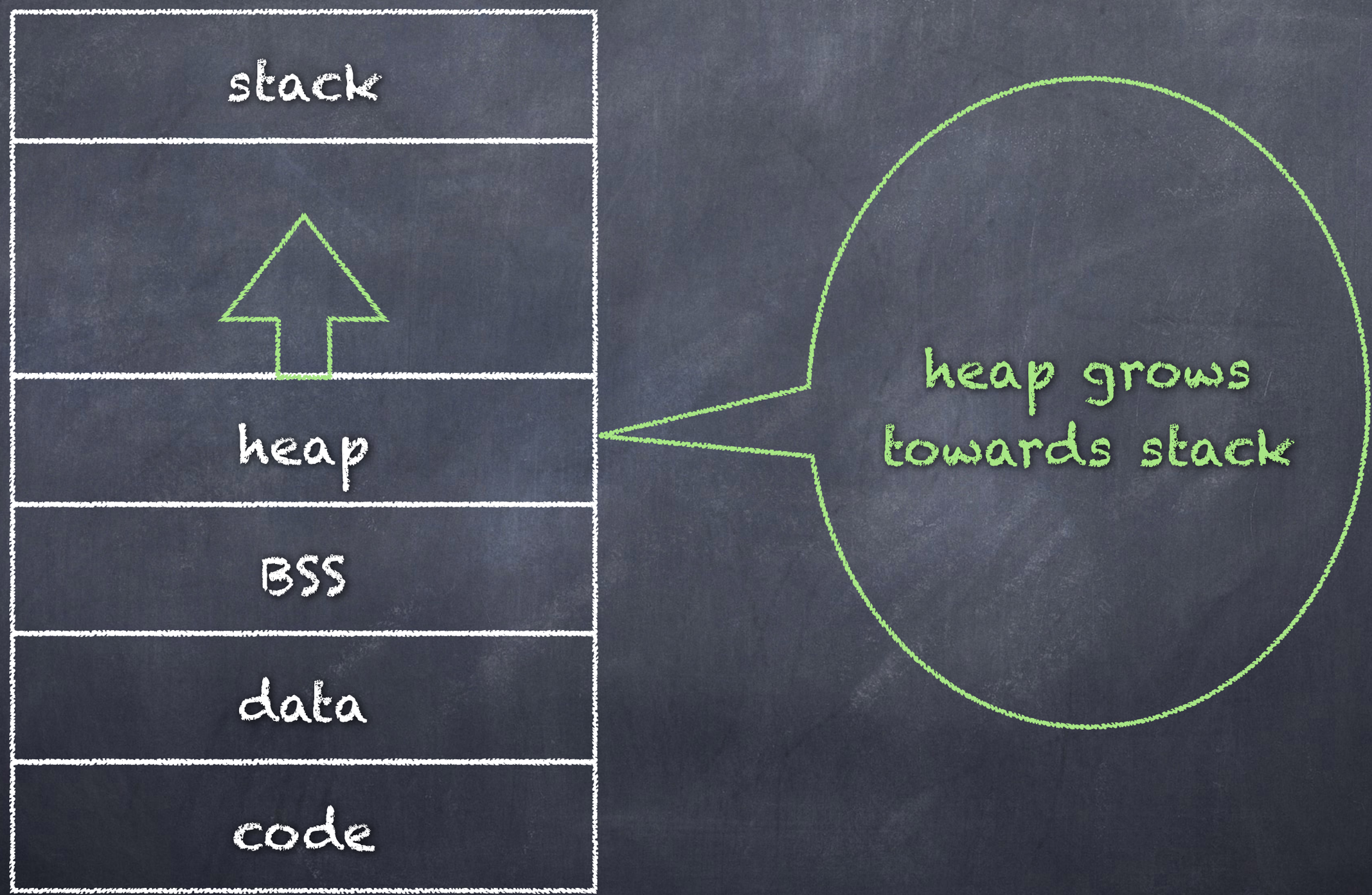
Memory organization



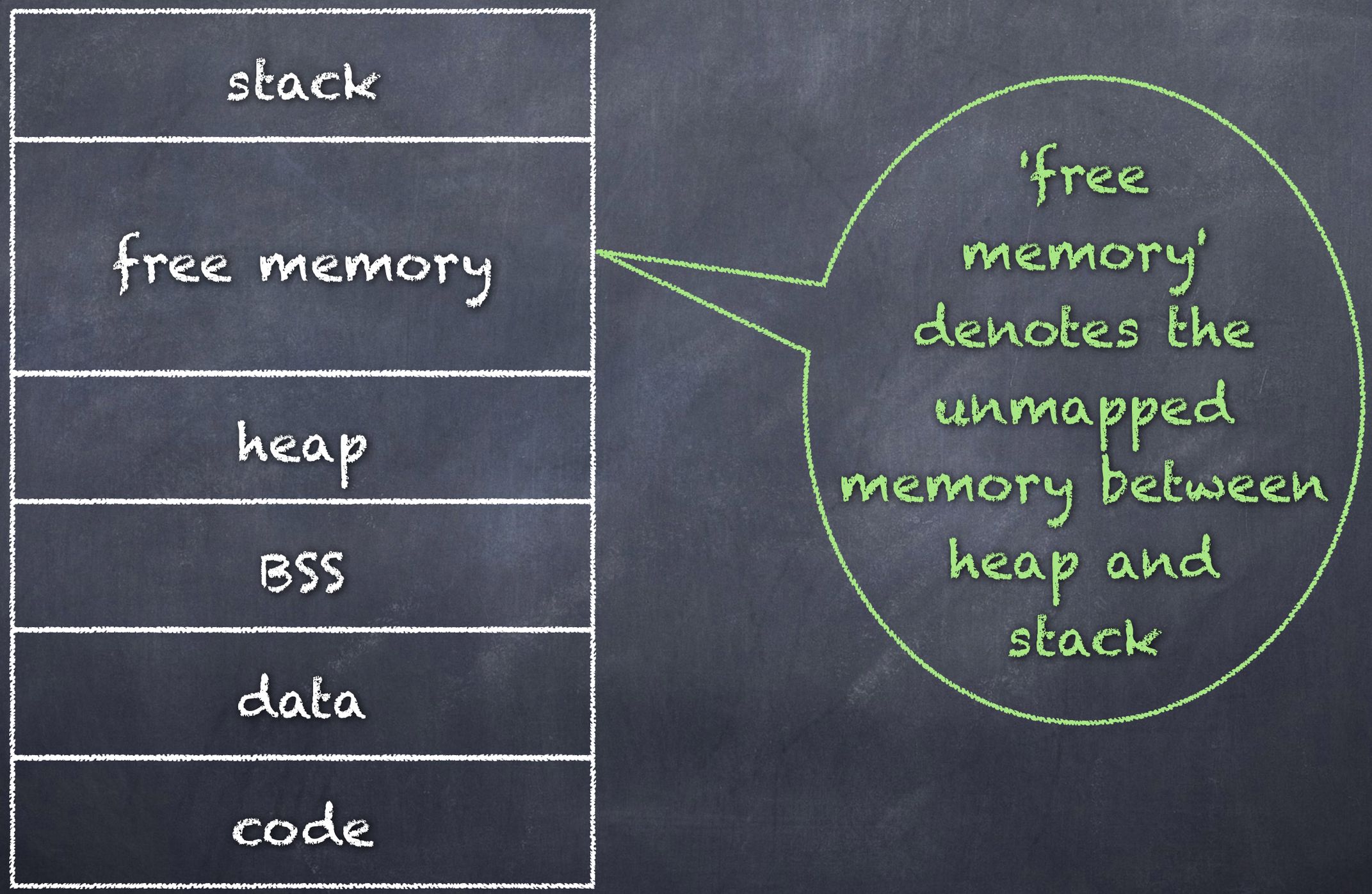
Memory organization



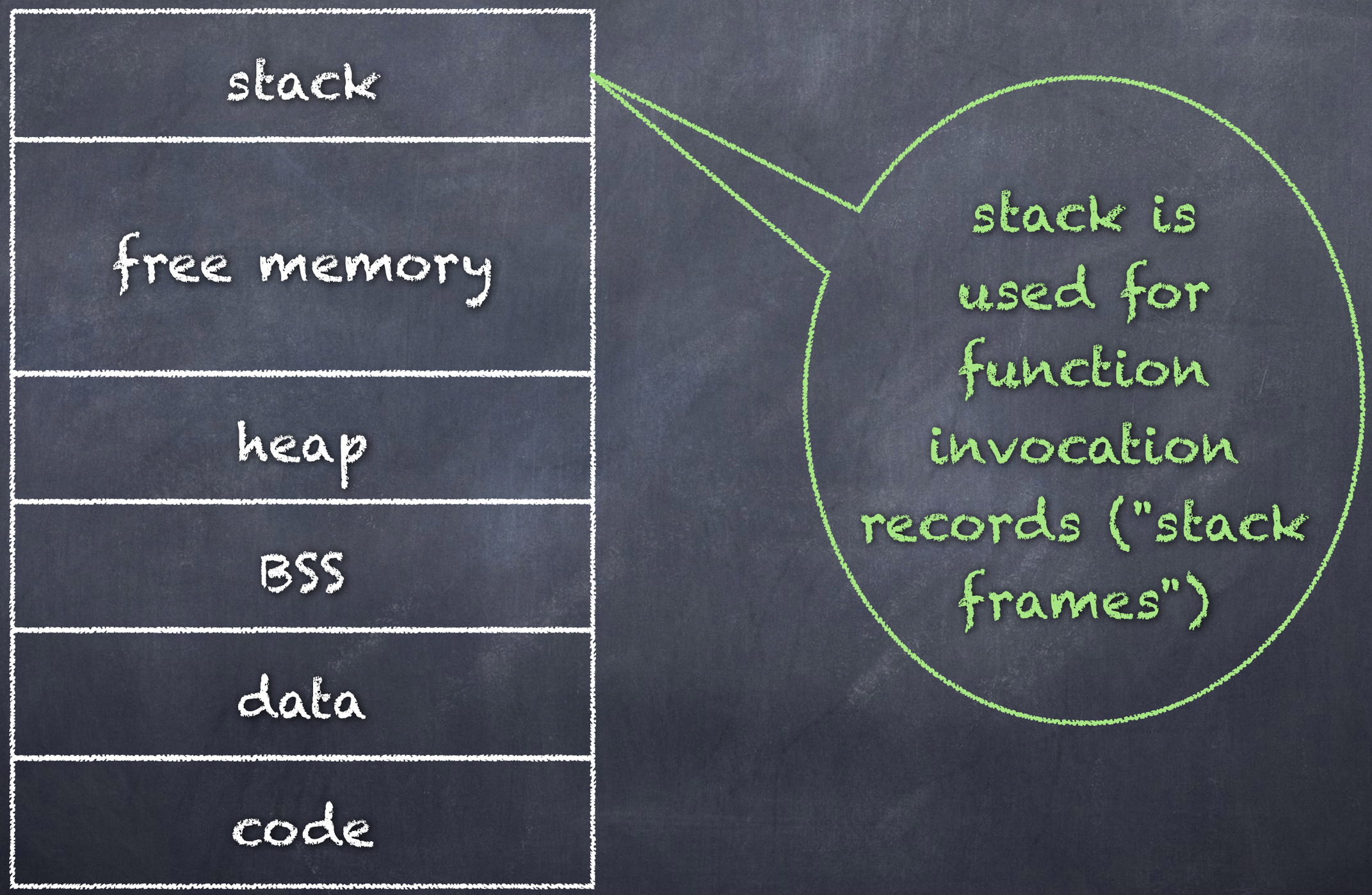
Memory organization



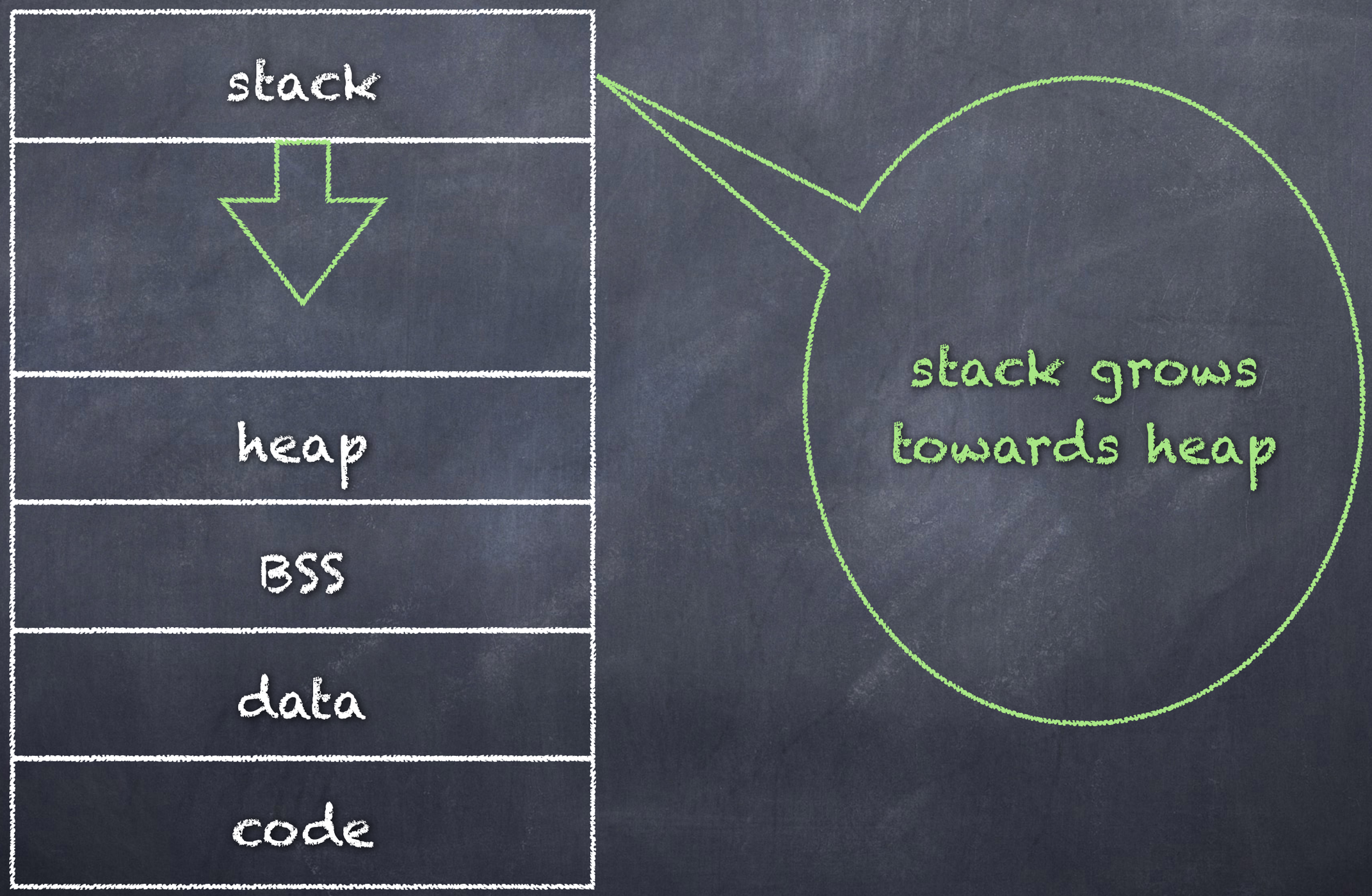
Memory organization



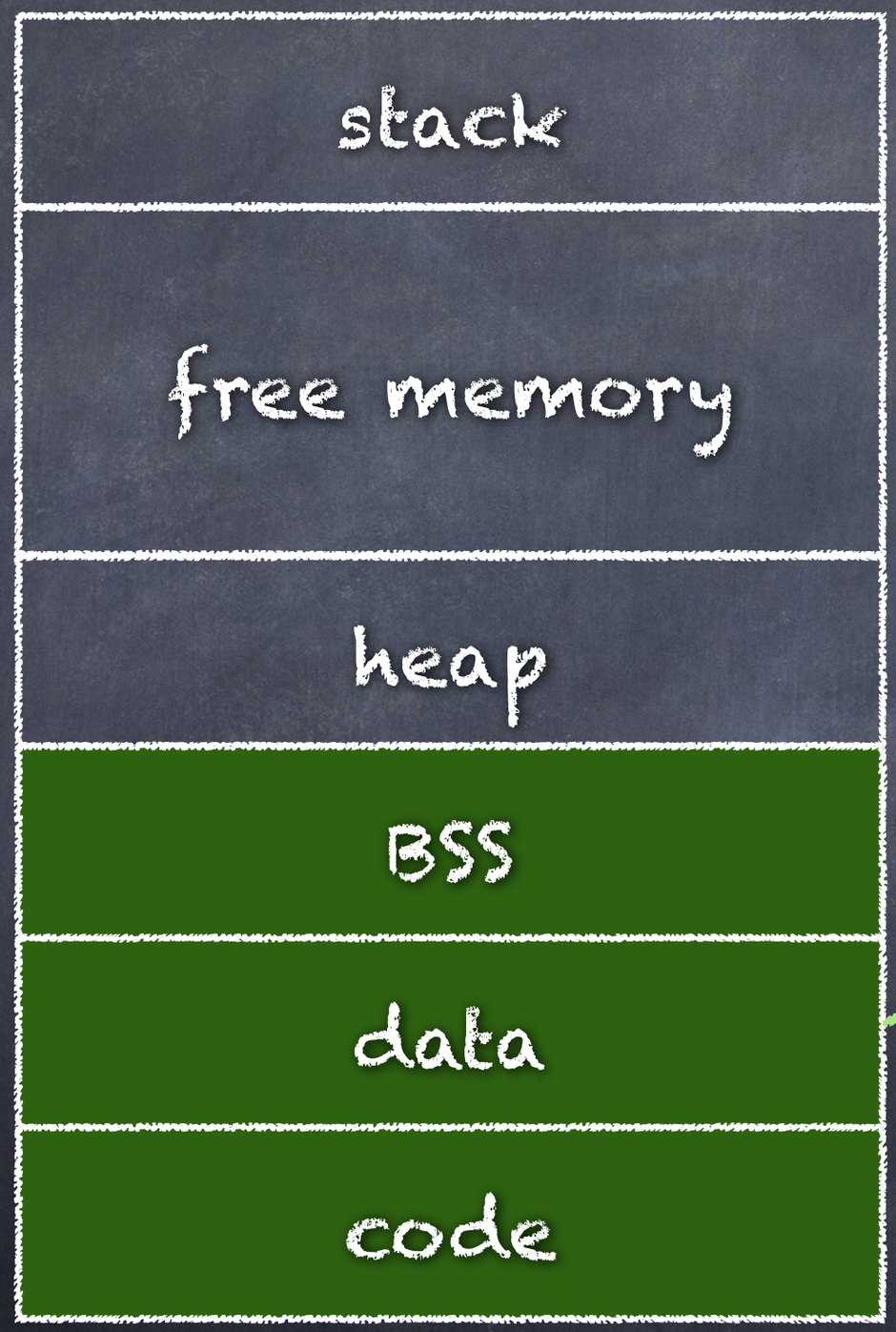
Memory organization



Memory organization

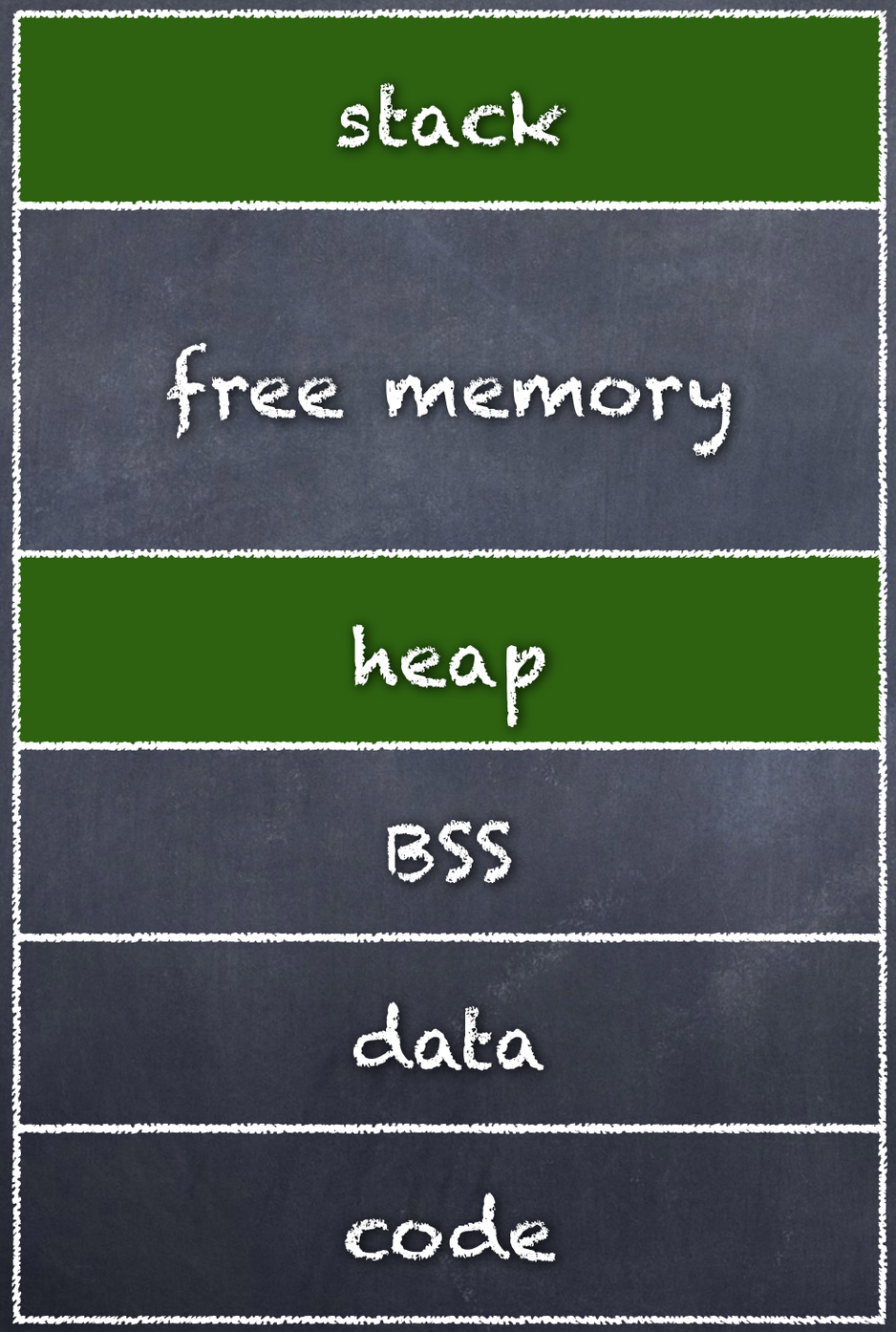


Memory organization



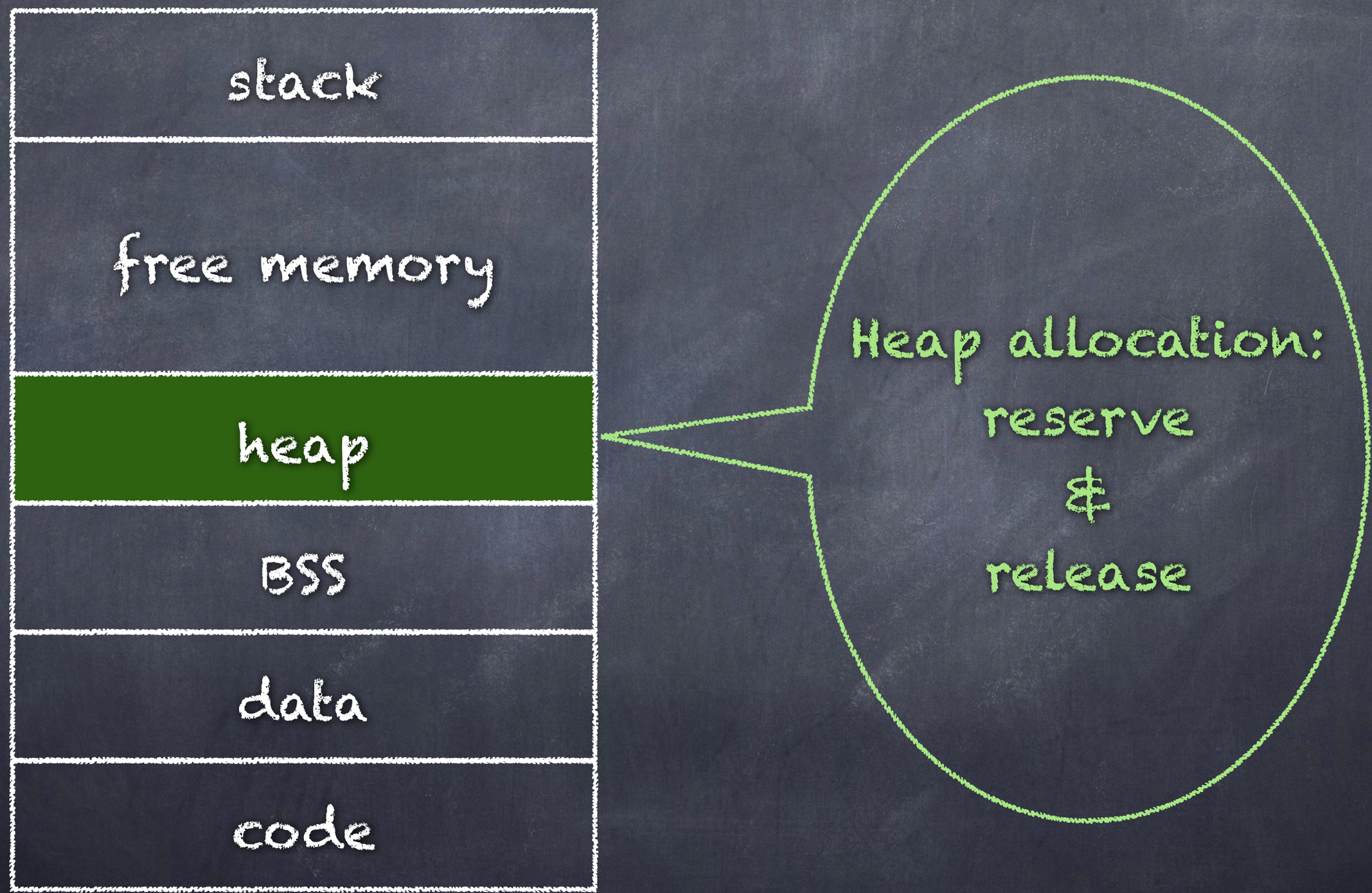
The size, layout and contents of both the code and static regions are determined at compile time

Memory organization

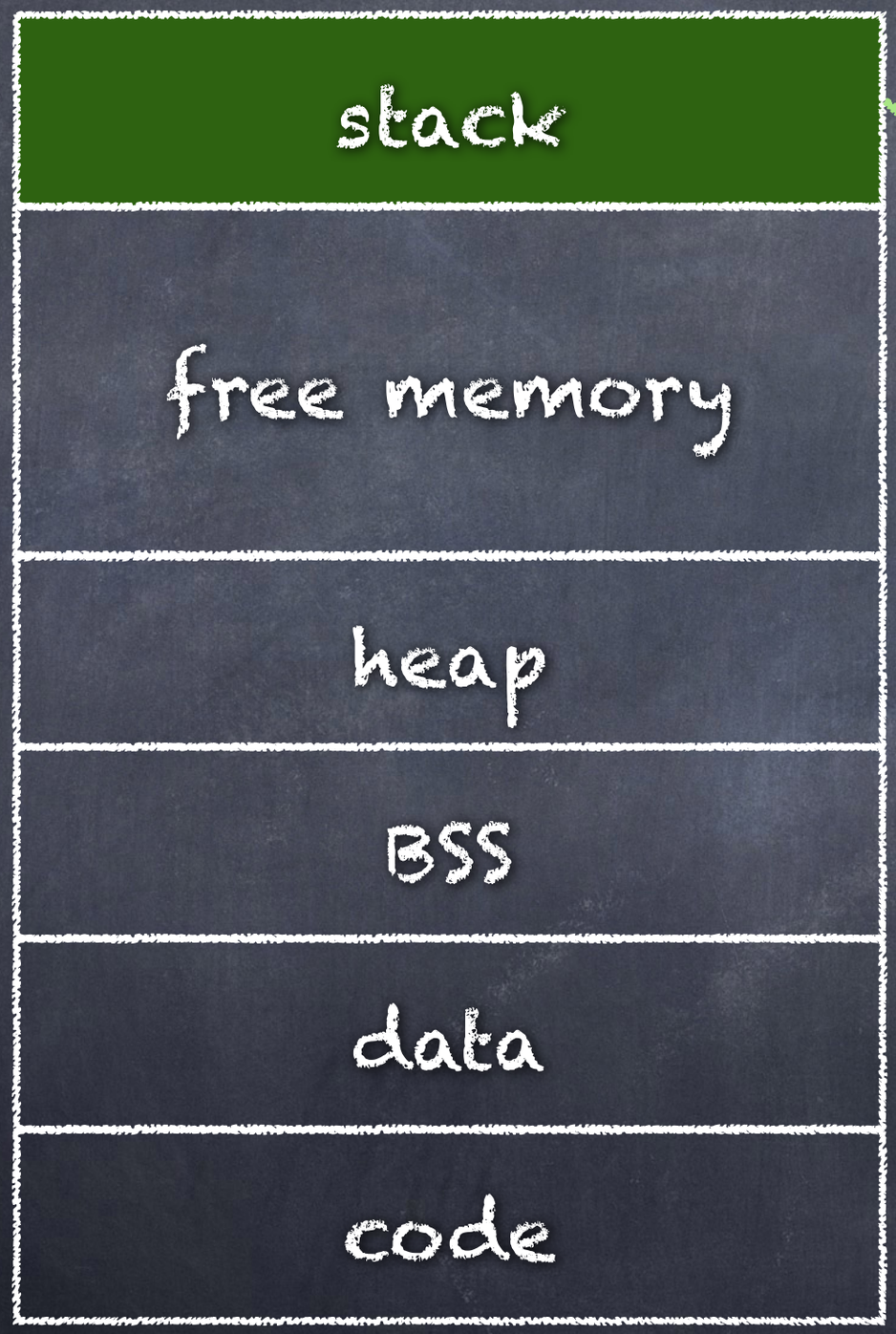


These regions are handled dynamically (i.e. at runtime)

Memory organization



Memory organization



Stack allocation:
function call

Stack frame organization