

DOSES

Breath Feels Fresh  
Up To 5X Longer\*  
L'haleine reste fraîche  
jusqu'à 5 fois plus longtemps\*  
\*vs. Brushing Alone/vs brossage des dents seulement

Scope®

Edward Z. Yang

98890288

750 mL



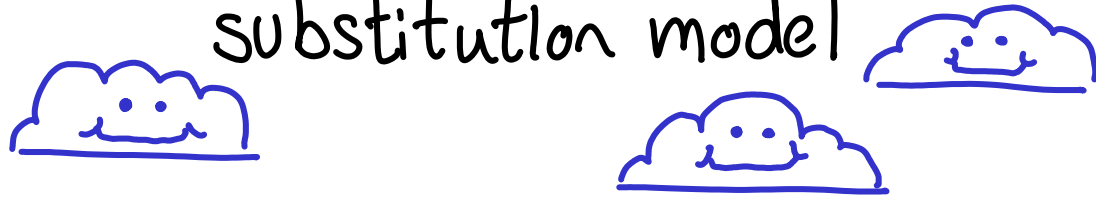
$$(\lambda x. f\ x\ x)\ (\lambda y. z)$$
$$\rightarrow_{\beta} f\ (\lambda y. z)\ (\lambda y. z)$$

```
var z = Ø;  
z++;  
console.log(z);
```

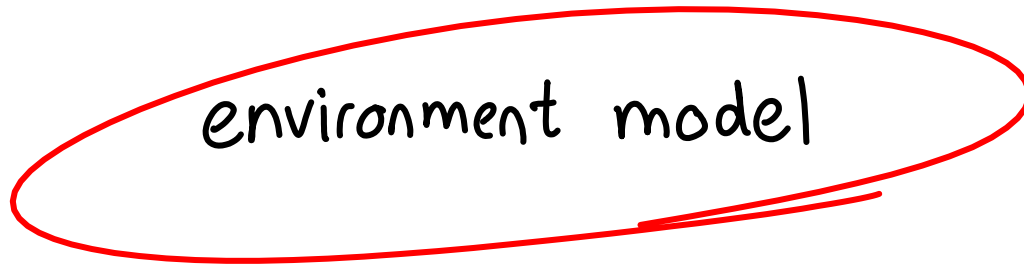


```
Ø++;  
console.log(Ø);
```


substitution model



environment model



the  
arrow of  
abstraction



machine model

# The Environment Model *by example*

**EX1** Anatomy of a scope

**EX2** First-order functions

control  
links

**EX3** Free variables

access  
links

**EX4** Higher-order functions

# The Environment Model *by example*

**EX1** Anatomy of a scope

**EX2** First-order functions

control  
links

**EX3** Free variables

access  
links

**EX4** Higher-order functions

## EX1 Anatomy of a scope

y	1
z	∅



environment pointer

```
var y = 1;  
var z = ∅;  
z++;  
console.log(z);
```

## EX1 Anatomy of a scope

### Environment/Activation Record

y	1
z	∅

var y = 1;

var z = ∅;

z++;

console.log(z);



environment pointer



## EX1 Anatomy of a scope

y	1
z	∅



environment pointer

```
var y = 1;  
var z = ∅;  
z++;  
console.log(z);
```

e.g. %esp on x86-32

## EX1 Anatomy of a scope

r-value: the "value"

y	1
z	∅

```
var y = 1;
```

```
var z = ∅;
```

```
z++;
```

```
console.log(z);
```



environment pointer

## EX1 Anatomy of a scope

l-value: the location

y	1
z	∅

```
var y = 1;
```

```
var z = ∅;
```

```
z++;
```

```
console.log(z);
```



environment pointer

## EX1 Anatomy of a scope

y	1
z	∅



environment pointer

```
var y = 1;
```

```
var z = ∅;
```

```
z++;
```

```
console.log(z);
```

## EX1 Anatomy of a scope

y	1
z	∅

```
var y = 1;
```

```
var z = ∅;
```

```
z++;
```

```
console.log(z);
```



environment pointer

## EX1 Anatomy of a scope

y	1
z	∅



environment pointer

```
var y = 1;
```

```
var z = ∅;
```

```
z++;
```

```
console.log(z);
```

## EX1 Anatomy of a scope

y	1
z	∅



environment pointer

```
var y = 1;
```

```
var z = ∅;
```

```
z++;
```

```
console.log(z);
```

## EX1 Anatomy of a scope

y	1
z	1



environment pointer

```
var y = 1;
```

```
var z = Ø;
```

```
z++;
```

```
console.log(z);
```



## EX1 Anatomy of a scope

y	1
z	1



environment pointer

```
var y = 1;
```

```
var z = Ø;
```

```
z++;
```

```
console.log(z);
```

**Ex2**

## First-order functions

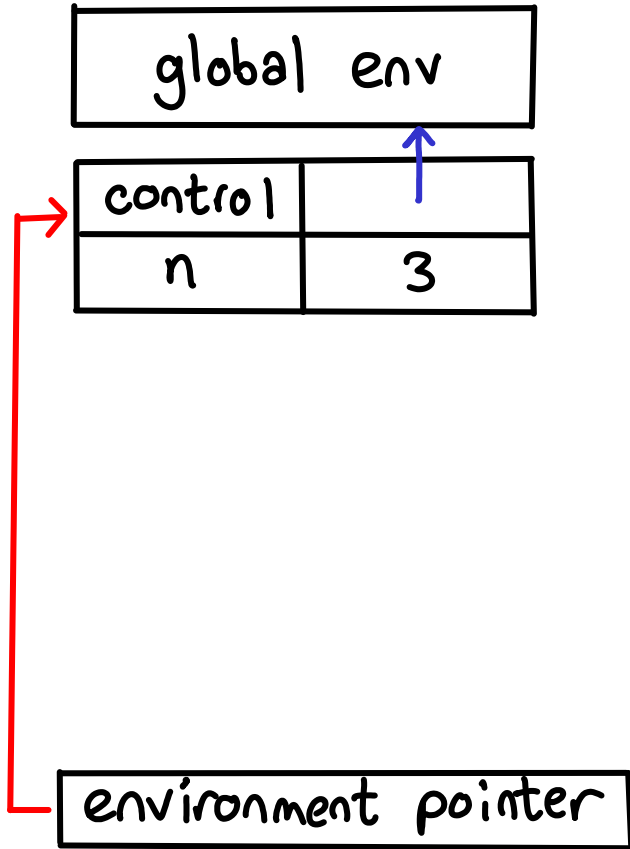
→ global env

environment pointer

```
function fact(n) {  
  if (n ≤ 1) {  
    return 1;  
  } else {  
    return n * fact(n-1)  
  }  
}  
fact(3);
```

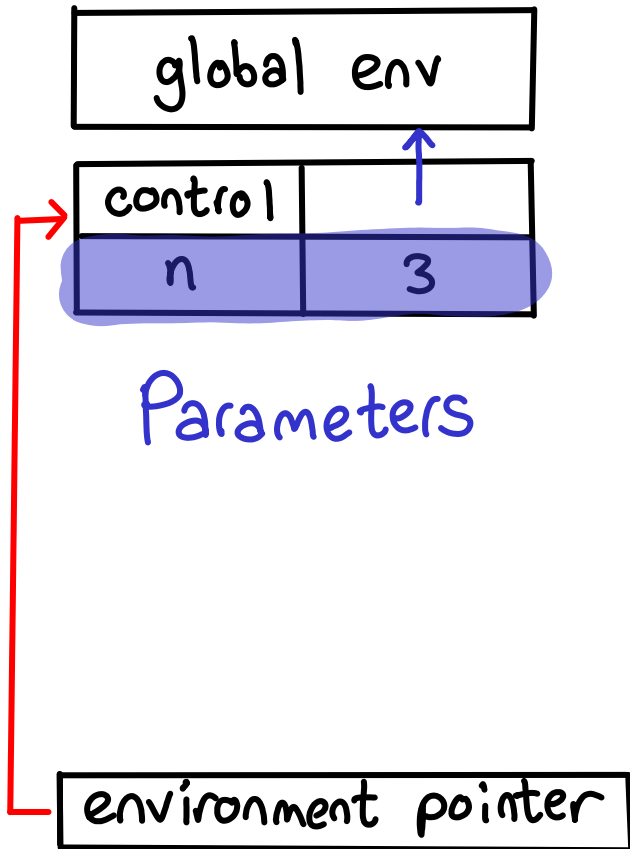
## Ex2

## First-order functions



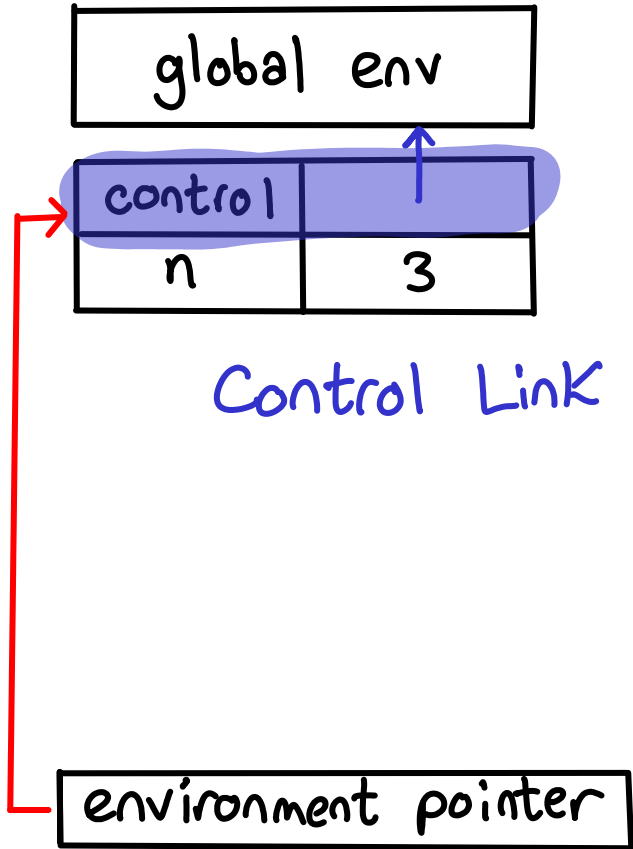
```
function fact(n) {  
  if ( $n \leq 1$ ) {  
    return 1;  
  } else {  
    return  $n * \text{fact}(n-1)$   
  }  
}  
fact(3);
```

## Ex2 First-order functions



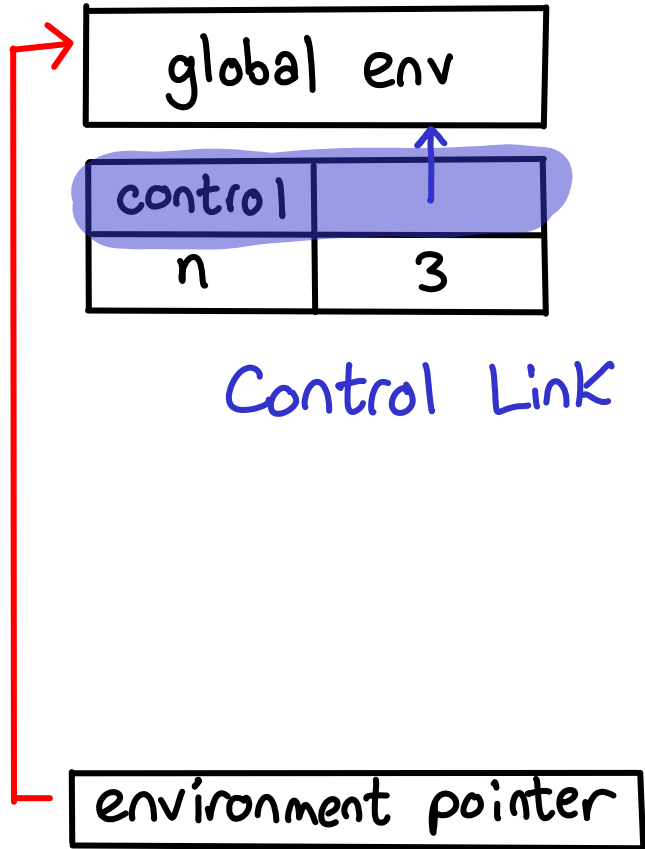
```
function fact(n) {  
  if ( $n \leq 1$ ) {  
    return 1;  
  } else {  
    return  $n * \text{fact}(n-1)$ ;  
  }  
}  
fact(3);
```

## Ex2 First-order functions



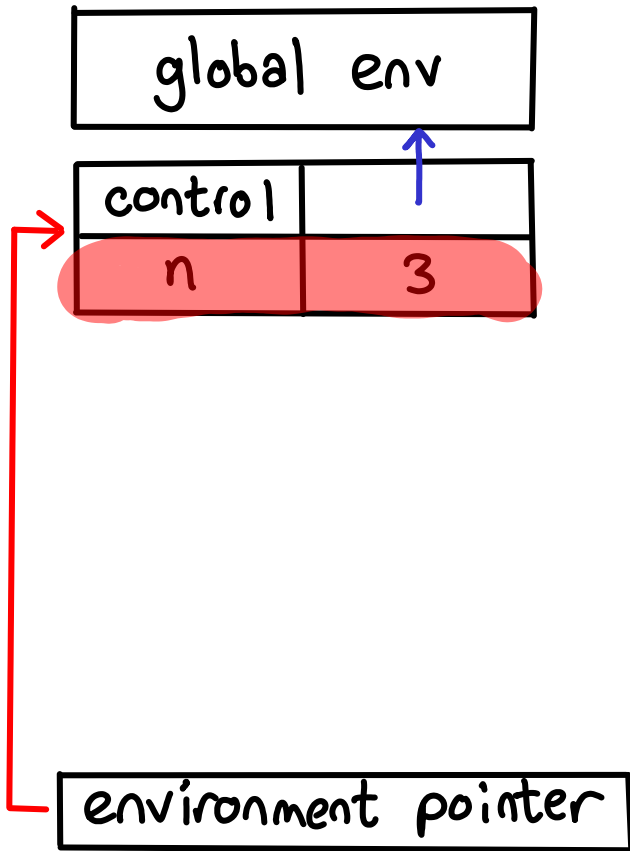
```
function fact(n) {  
  if (n ≤ 1) {  
    return 1;  
  } else {  
    return n * fact(n-1);  
  }  
}  
fact(3);
```

## Ex2 First-order functions



```
function fact(n) {  
  if (n ≤ 1) {  
    return 1;  
  } else {  
    return n * fact(n-1)  
  }  
}  
fact(3);
```

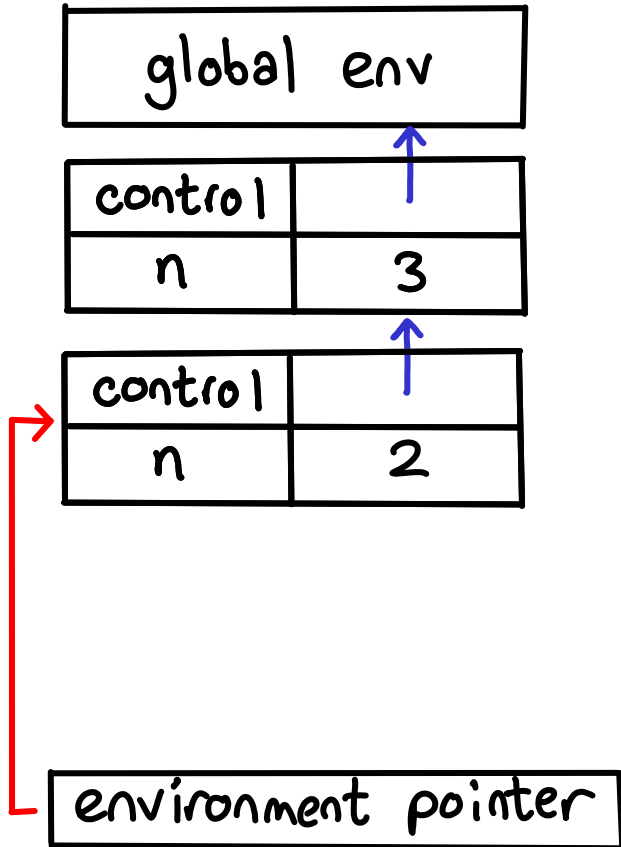
## Ex2 First-order functions



```
function fact(n) {  
  if (n ≤ 1) {  
    return 1;  
  } else {  
    return n * fact(n-1);  
  }  
}  
fact(3);
```

## Ex2

## First-order functions

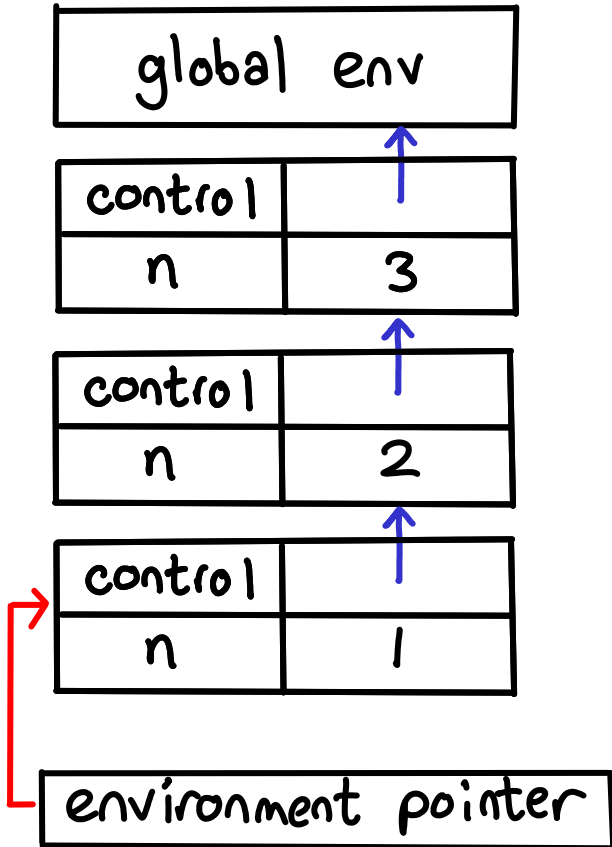


```
function fact(n) {  
  if ( $n \leq 1$ ) {  
    return 1;  
  } else {  
    return  $n * \text{fact}(n-1)$   
  }  
}  
fact(3);
```



## Ex2

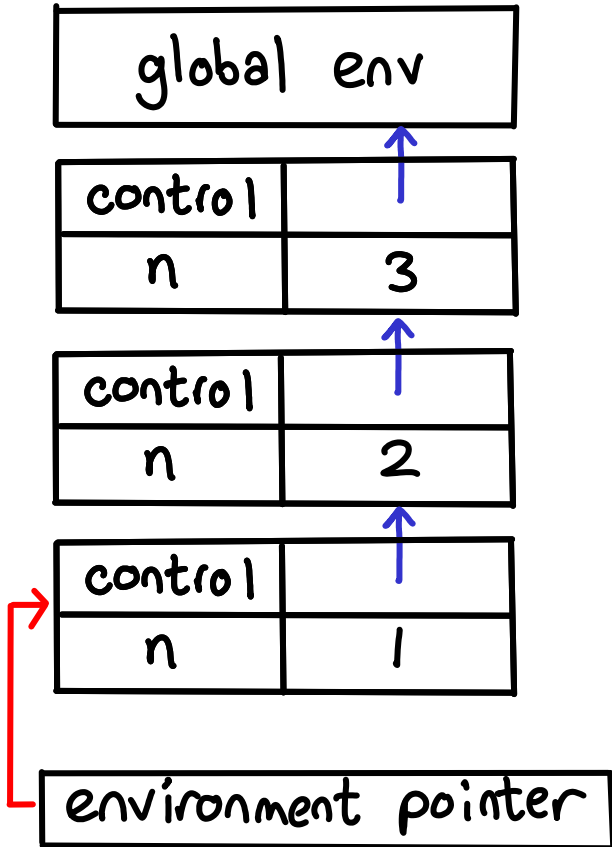
## First-order functions



```
function fact(n) {  
  if (n ≤ 1) {  
    return 1;  
  } else {  
    return n * fact(n-1);  
  }  
}  
fact(3);
```

## Ex2

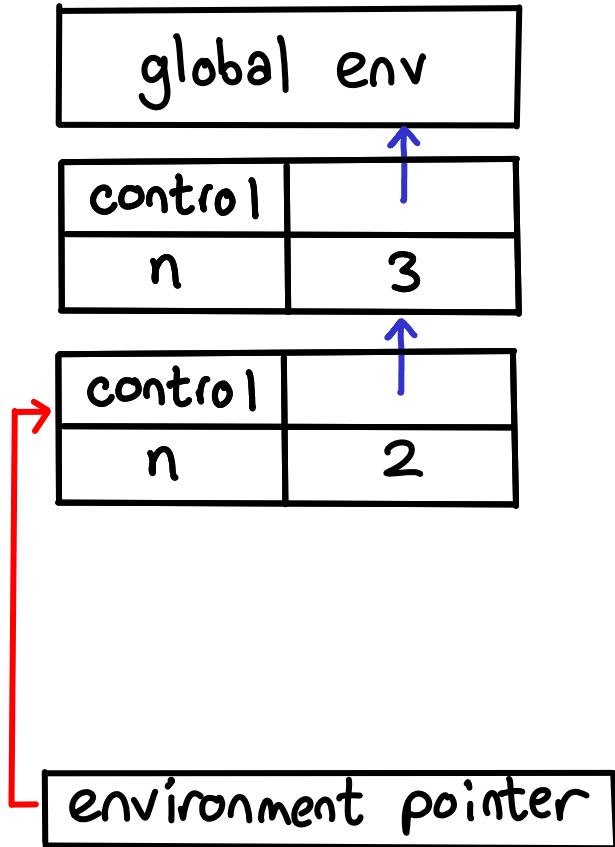
## First-order functions



```
function fact(n) {  
  if (n ≤ 1) {  
    return 1;  
  } else {  
    return n * fact(n-1)  
  }  
}  
fact(3);
```

## Ex2

## First-order functions

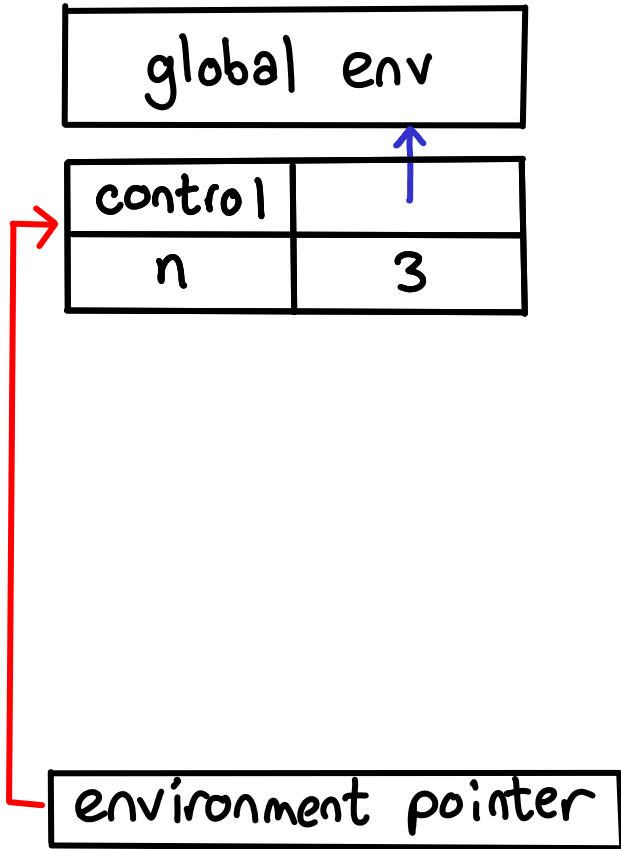


```
function fact(n) {  
  if (n ≤ 1) {  
    return 1;  
  } else {  
    return n * fact(n-1);  
  }  
}  
fact(3);
```

A red arrow points from the number 1 in the code to the return value 1 in the environment frame diagram.

Ex2

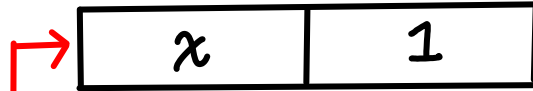
## First-order functions



```
function fact(n) {  
  if (n ≤ 1) {  
    return 1;  
  } else {  
    return n * fact(n-1);  
  }  
}  
fact(3);
```

A red arrow points from the number "2" below to the `fact(n-1)` expression in the function definition.

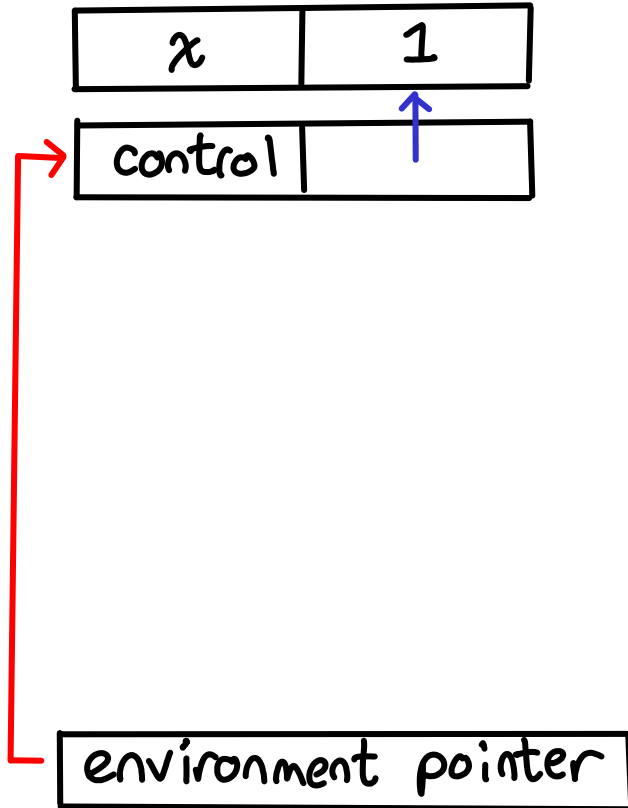
### Ex3 Free variables



```
var x = 1;  
function f() {  
    console.log(x);  
}  
f();
```

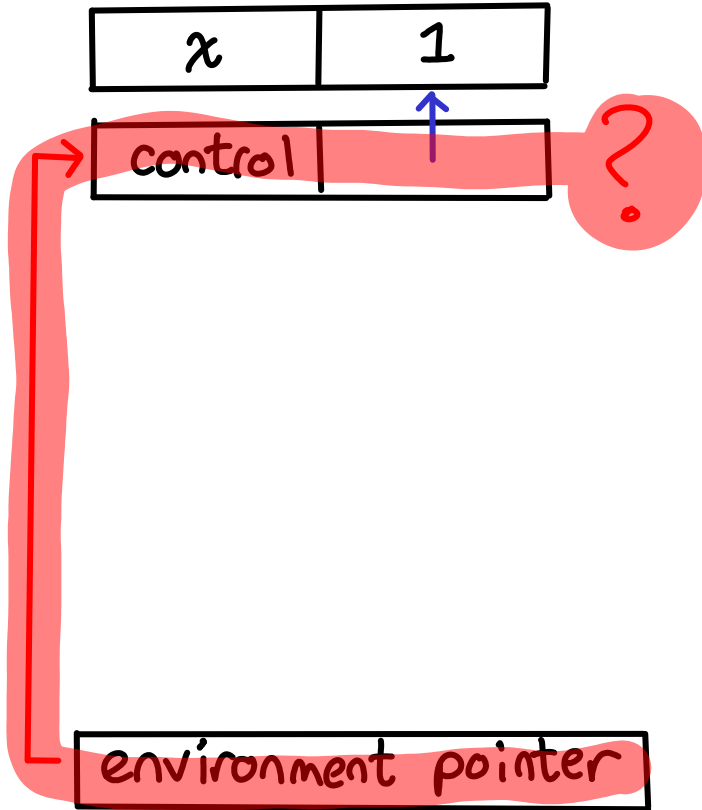
environment pointer

### Ex3 Free variables



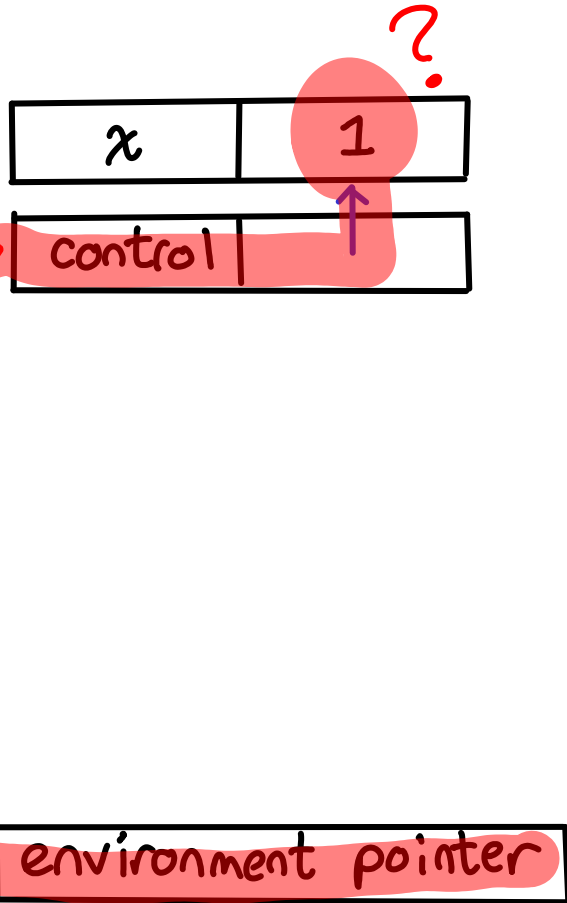
```
var x = 1;  
function f() {  
  console.log(x);  
}  
f();
```

### Ex3 Free variables



```
var x = 1;
function f() {
  console.log(x);
}
f();
```

### Ex3 Free variables

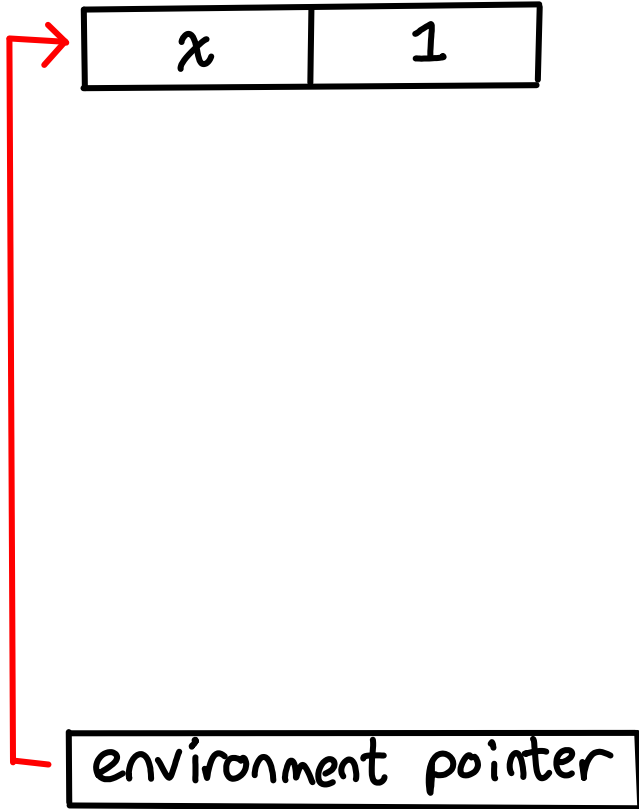


```
var x = 1;  
function f() {  
    console.log(x);  
}  
f();
```

The diagram shows the execution of the code. A red arrow points from the "x" in the code to the "x" in the environment box. Another red arrow points from the "x" in the code to the "1" in the environment box. A red circle is drawn around the "x" in the code, and a red arrow points from it to the "x" in the environment box.



### Ex3 Free variables



```
var x = 1;
function f() {
  console.log(x);
}
function g() {
  var x = 2;
  f();
}
g();
```

### Ex3 Free variables

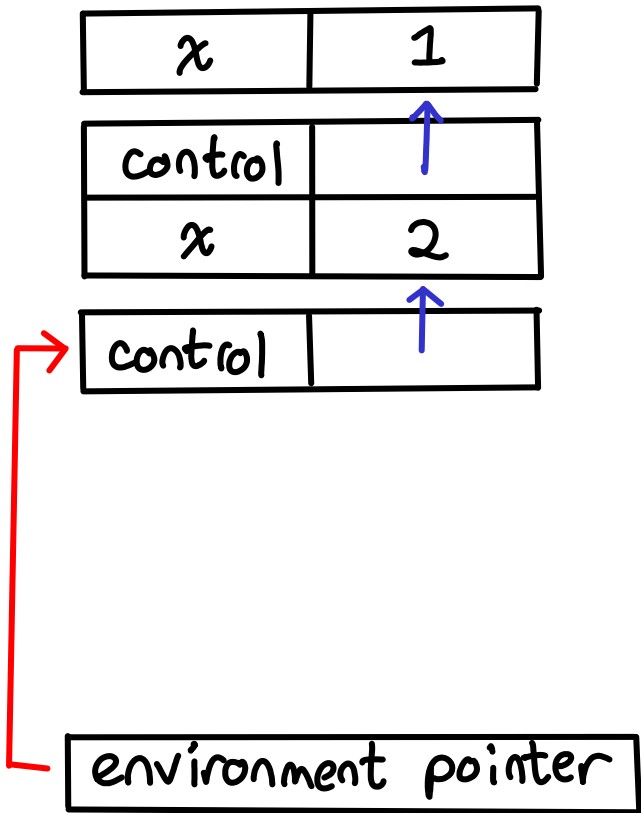
x	1
control	↑
x	2



environment pointer

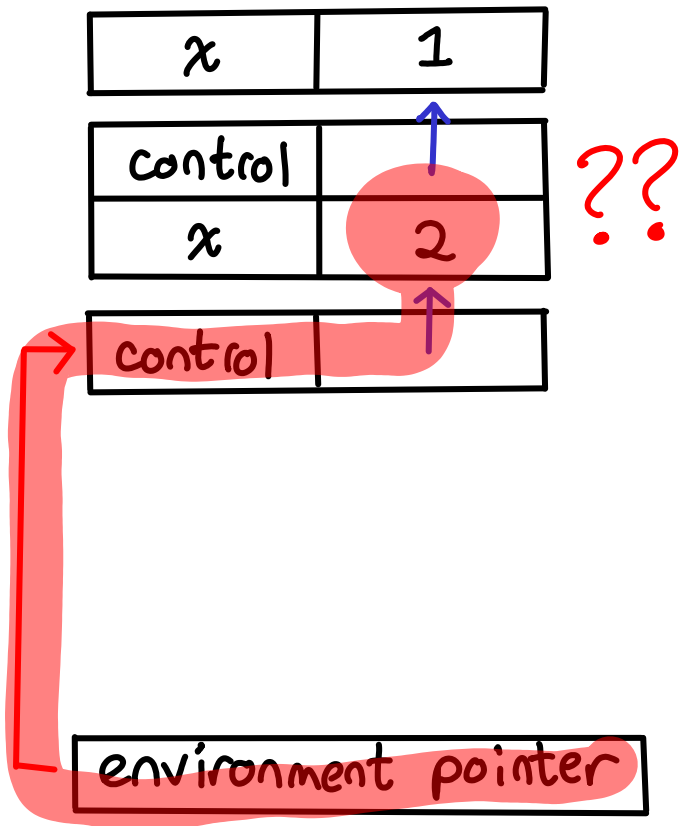
```
var x = 1;
function f() {
  console.log(x);
}
function g() {
  var x = 2;
  f();
}
g();
```

### Ex3 Free variables



```
var x = 1;
function f() {
  console.log(x);
}
function g() {
  var x = 2;
  f();
}
g();
```

### Ex3 Free variables



```
var x = 1;
function f() {
  console.log(x);
}
function g() {
  var x = 2;
  f();
}
g();
```

Red annotations in the code:

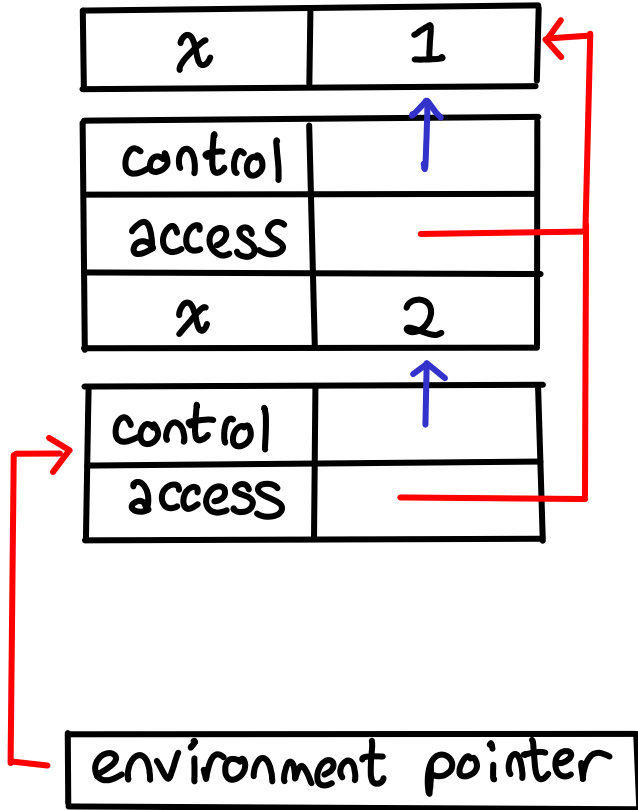
- A red circle highlights the `x` in `console.log(x);`.
- A red arrow points from the `f();` call in the `g()` function to the `x` in `console.log(x);` in the `f()` function.
- Two red question marks (`??`) are placed below the `g();` line.

Congratulations:



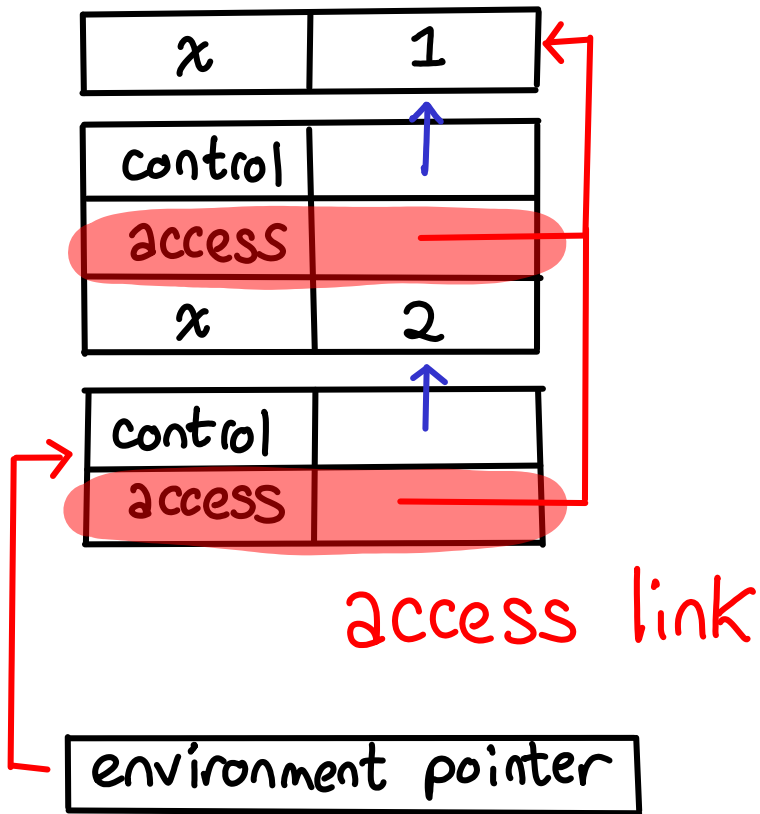
You just invented  
dynamic scoping

### Ex3 Free variables



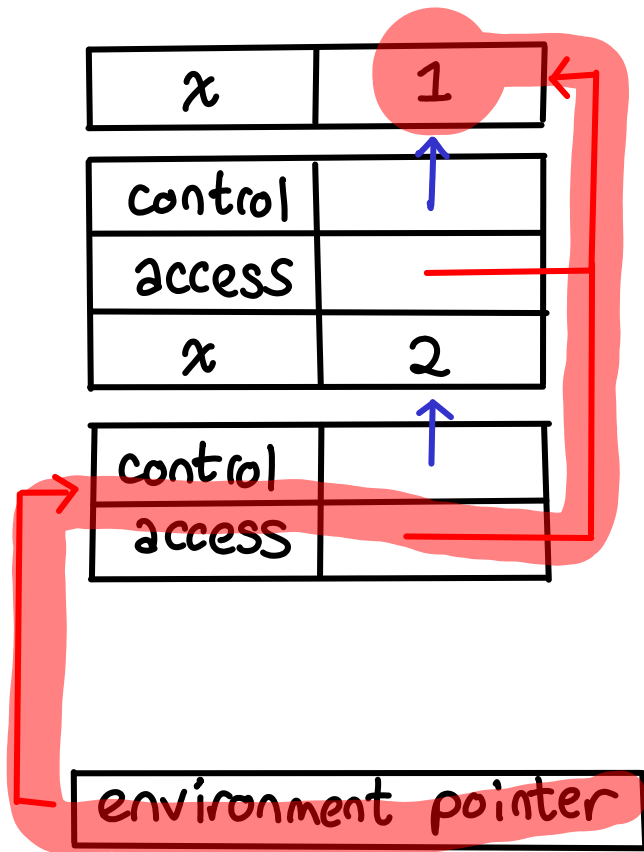
```
var x = 1;
function f() {
  console.log(x);
}
function g() {
  var x = 2;
  f();
}
g();
```

### Ex3 Free variables



```
var x = 1;
function f() {
  console.log(x);
}
function g() {
  var x = 2;
  f();
}
g();
```

### Ex3 Free variables



```
var x = 1;
function f() {
  console.log(x);
}
function g() {
  var x = 2;
  f();
}
g();
```

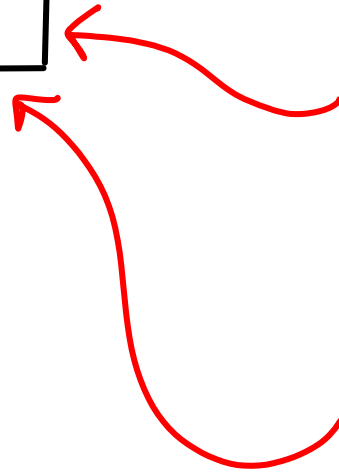


But how do we know how  
to wire up the **access links**?

### Ex3 Free variables

$x$	1
-----	---

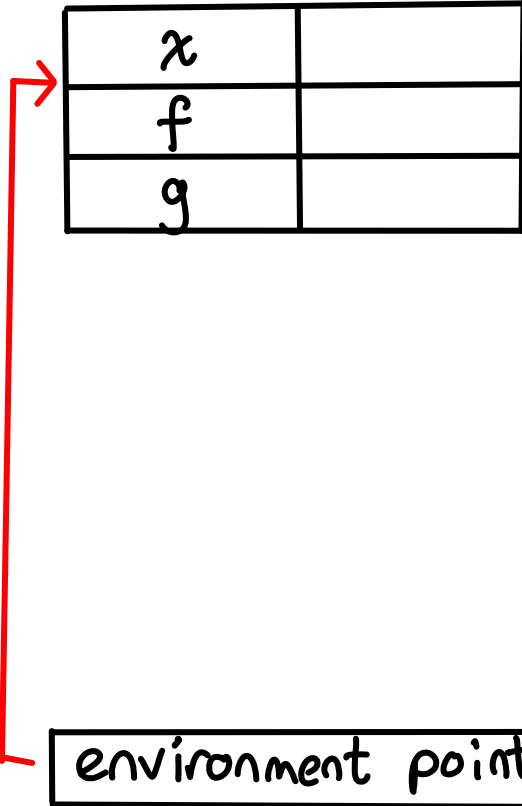
```
var x = 1;  
function f() {  
  console.log(x);  
}  
function g() {  
  var x = 2;  
  f();  
}  
g();
```



static?

Functions are Data

### Ex3 Free variables



x	
f	
g	

```
var x = 1;
```

```
var f = function () {  
    console.log(x);  
}
```

```
var g = function () {  
    var x = 2;  
    f();  
}
```

```
g();
```

### Ex3 Free variables

x	1
f	
g	



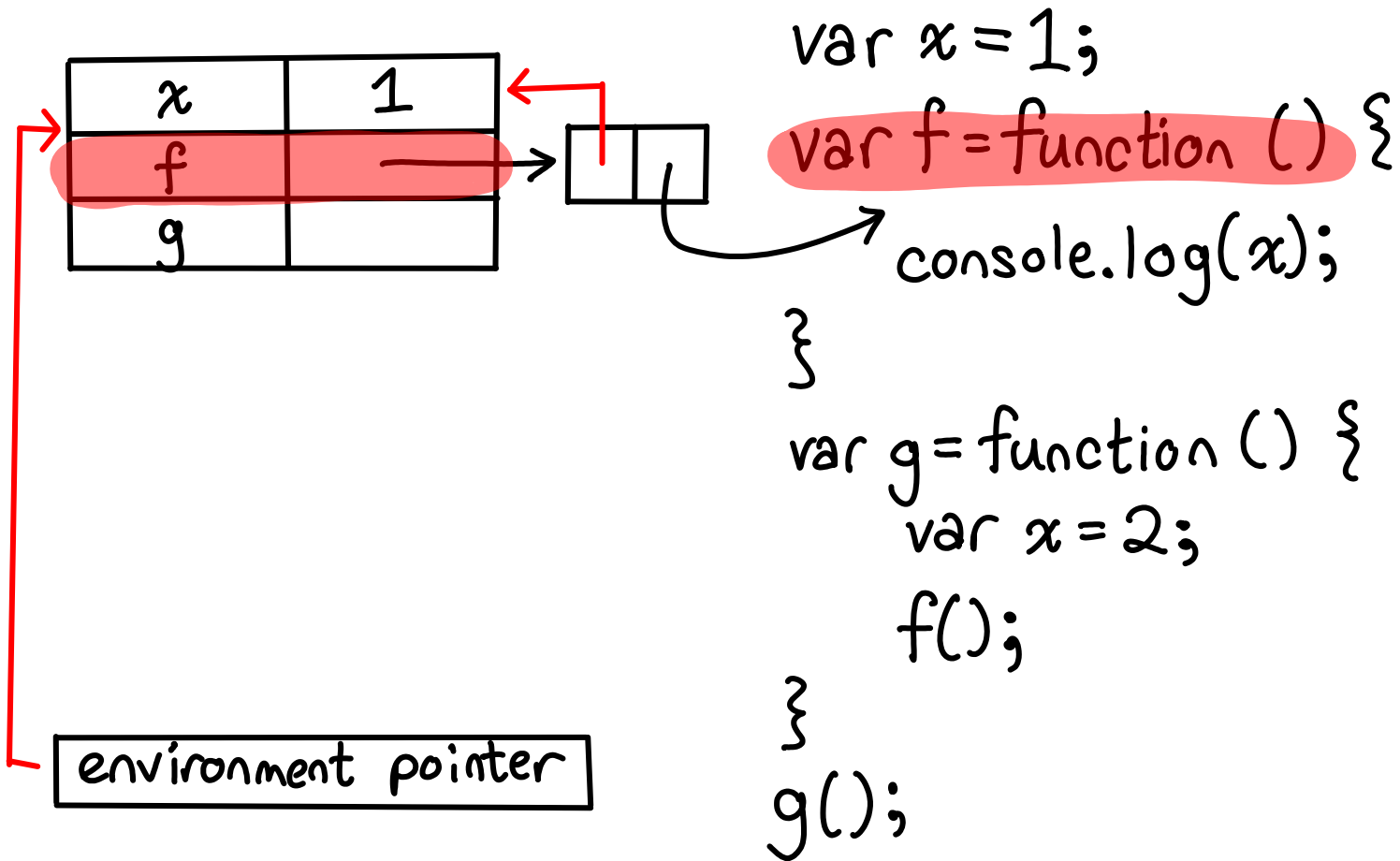
environment pointer

```
var x = 1;
```

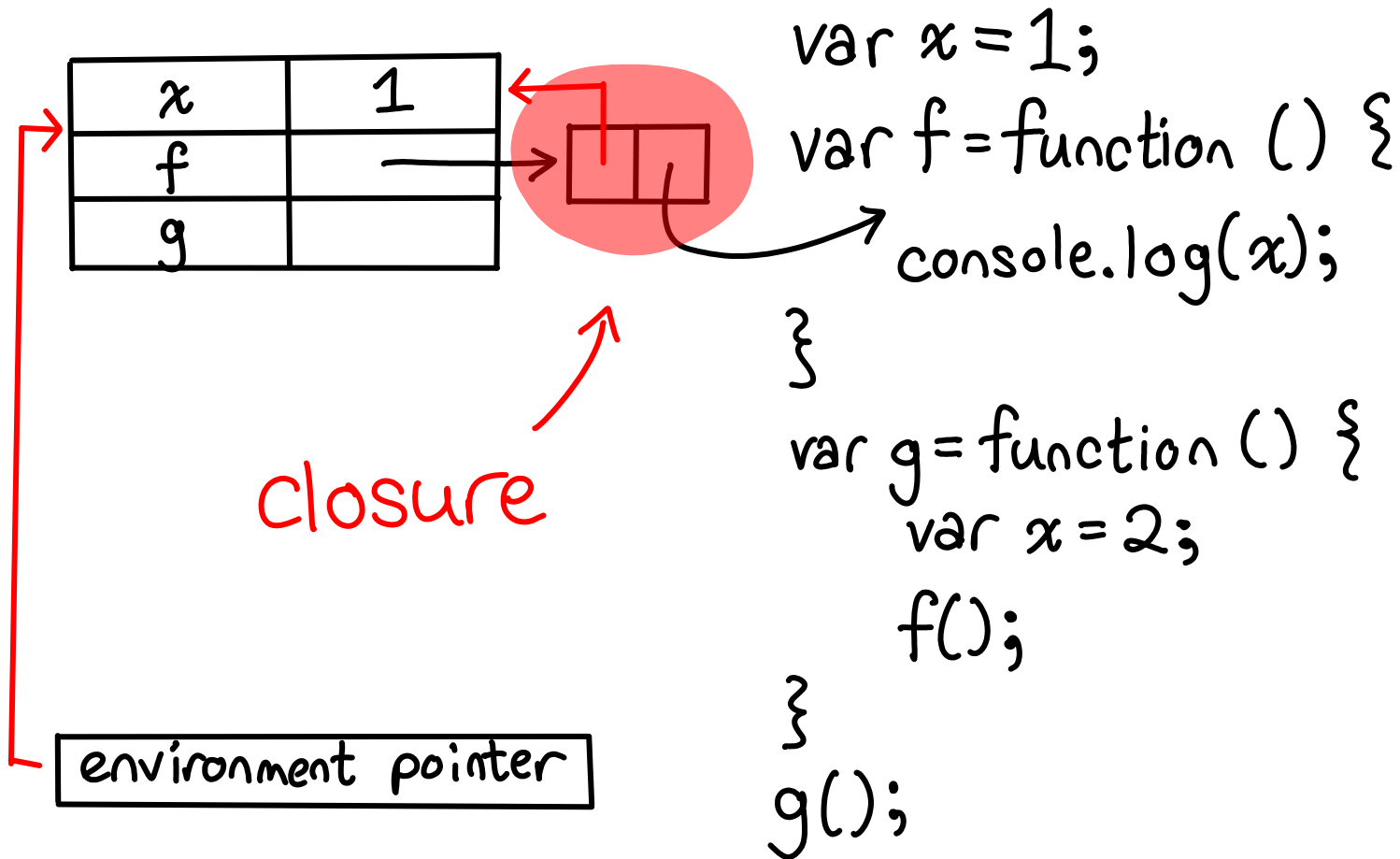
```
var f = function () {  
  console.log(x);  
}
```

```
var g = function () {  
  var x = 2;  
  f();  
}  
g();
```

### Ex3 Free variables



### Ex3 Free variables



### Ex3 Free variables

x	1
f	→ [ ]
g	

environment  
pointer

environment pointer

```
var x = 1;
```

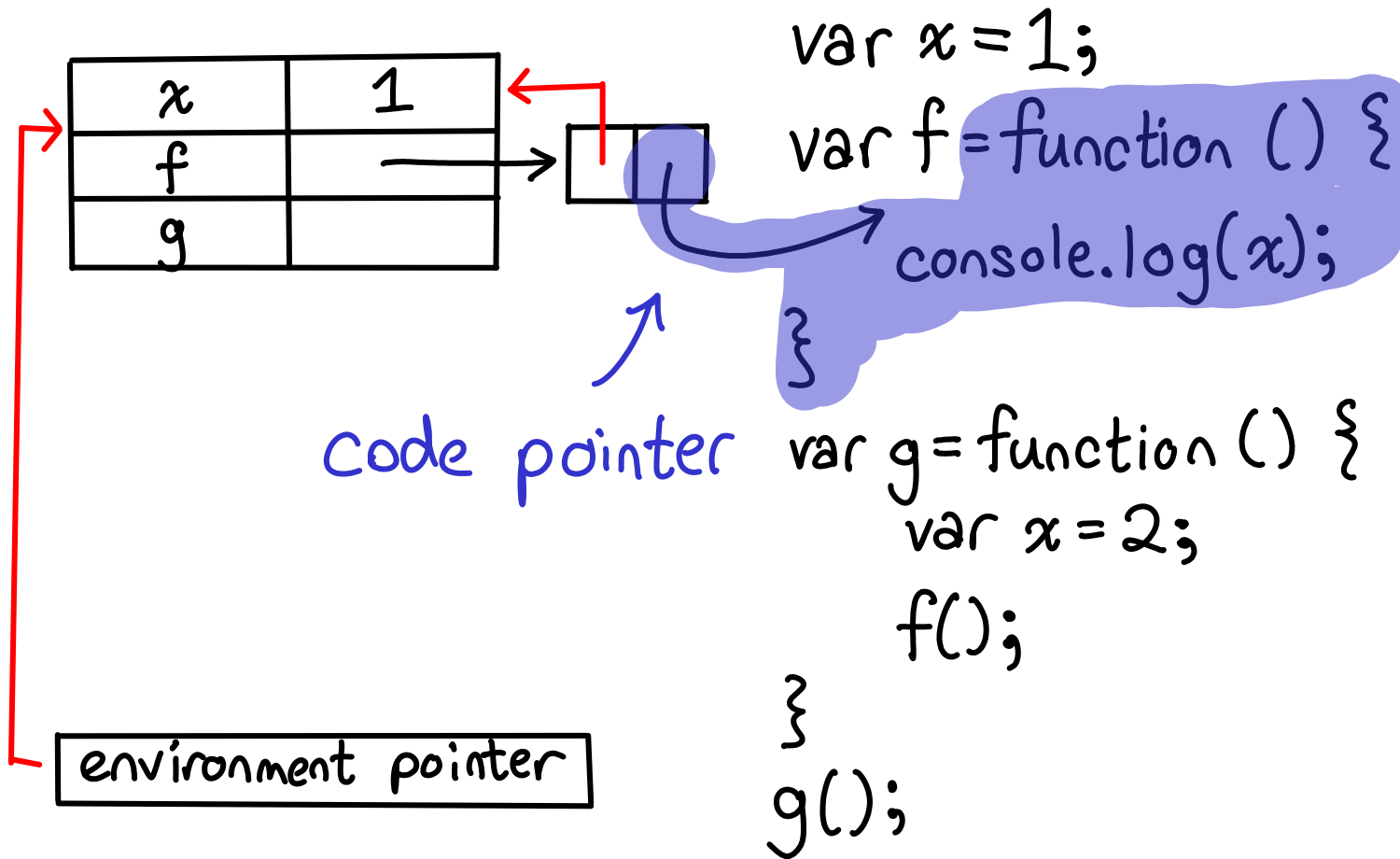
```
var f = function () {  
  console.log(x);  
}
```

```
var g = function () {  
  var x = 2;  
  f();  
}
```

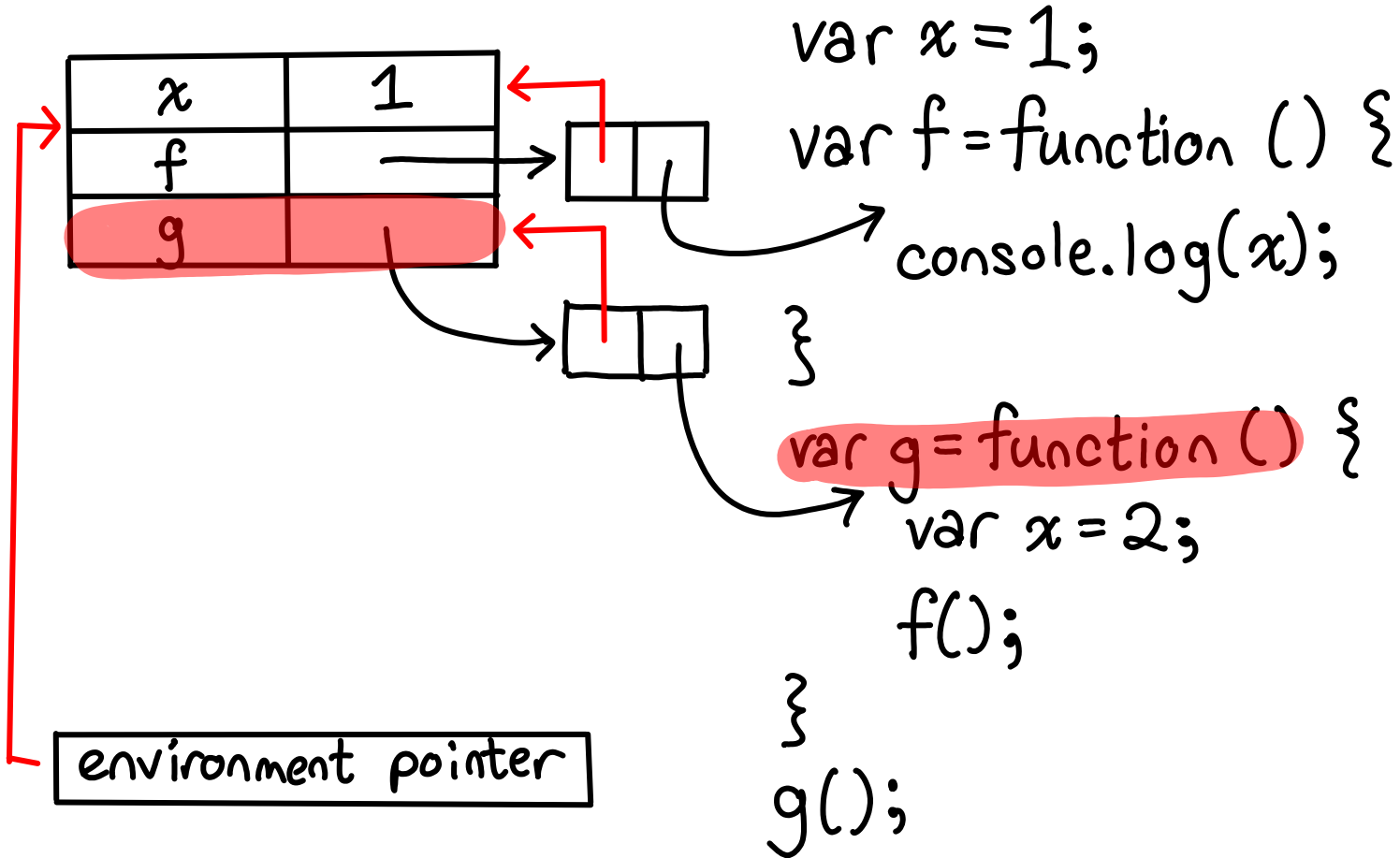
```
g();
```



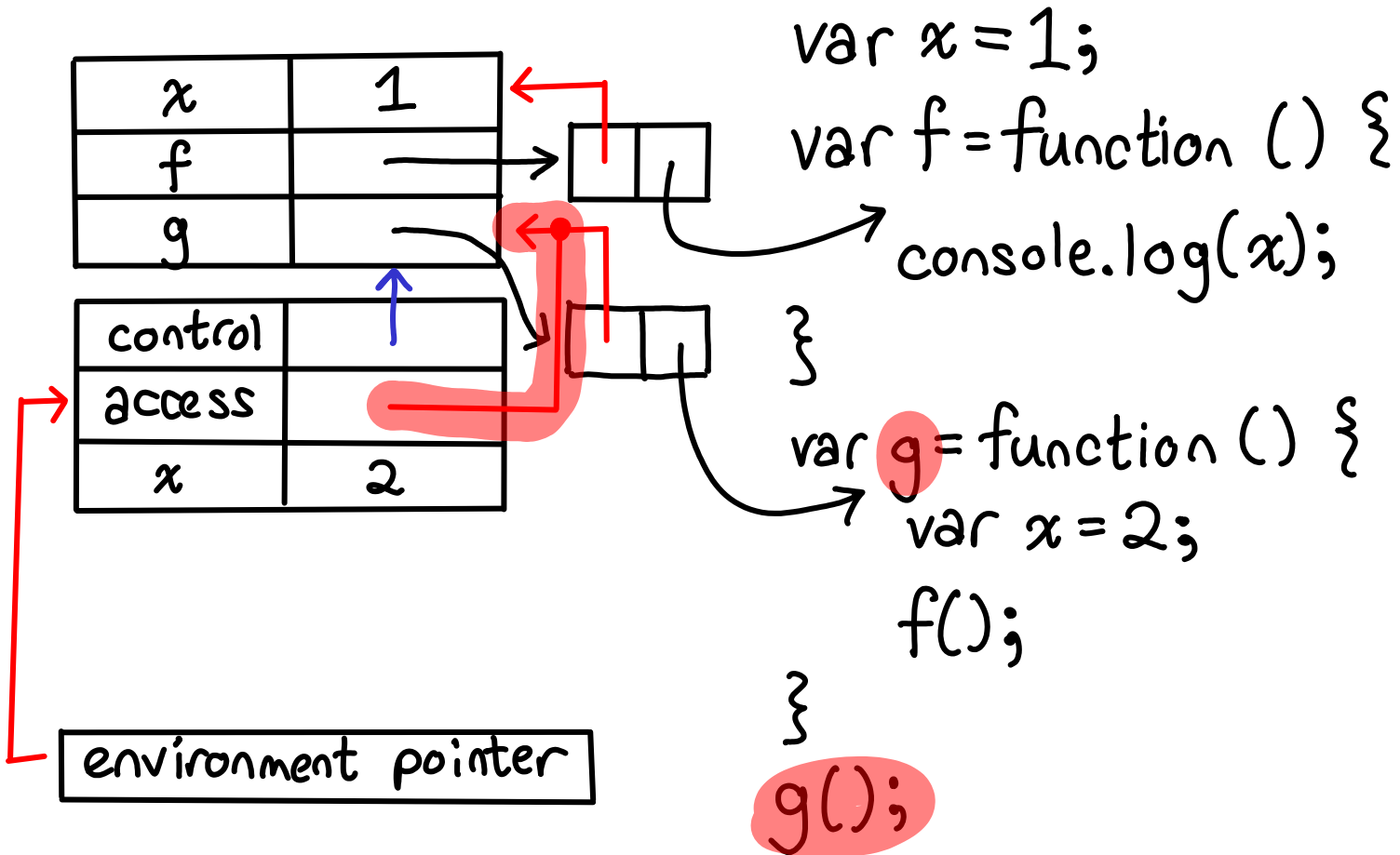
### Ex3 Free variables



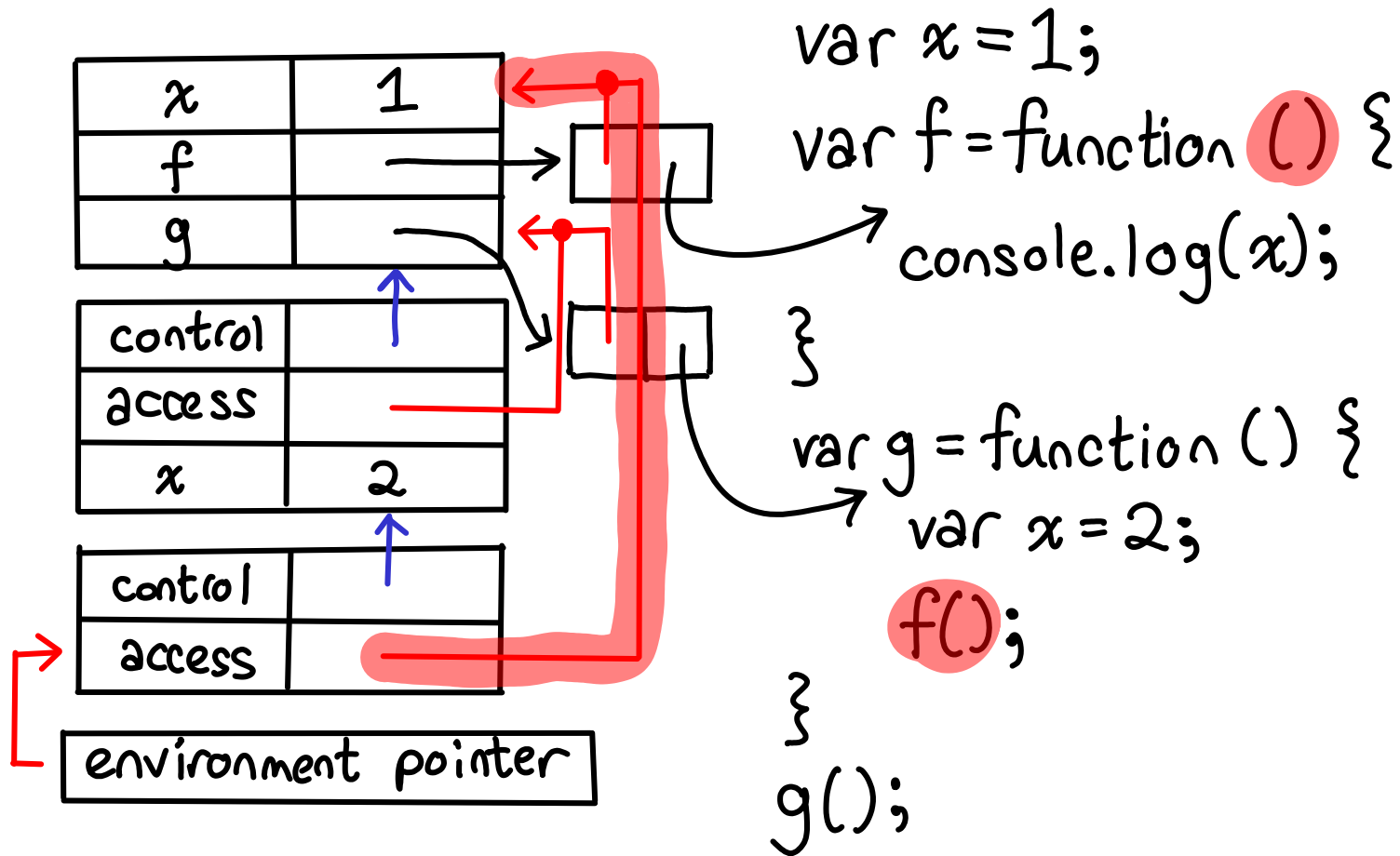
### Ex3 Free variables



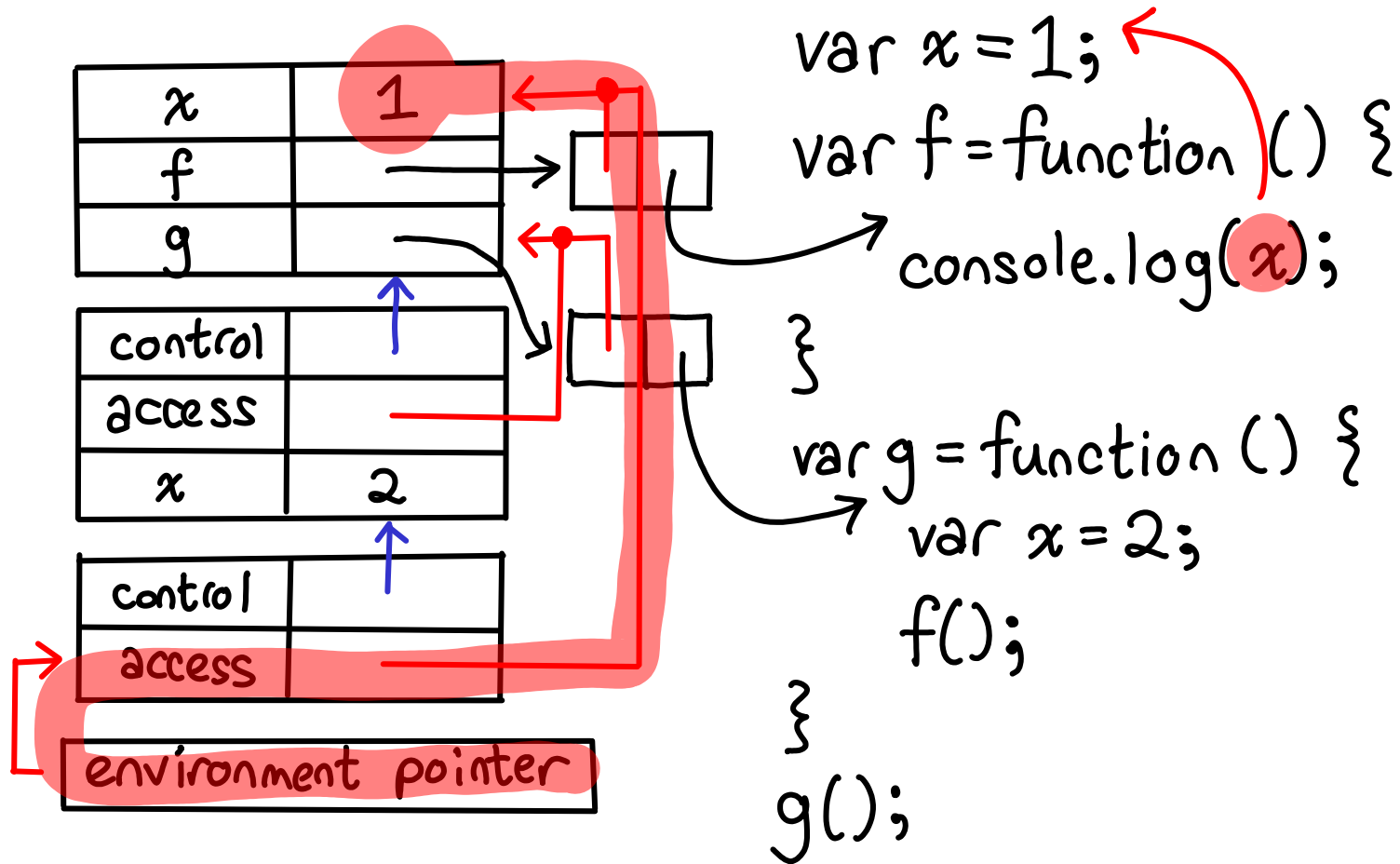
### Ex3 Free variables



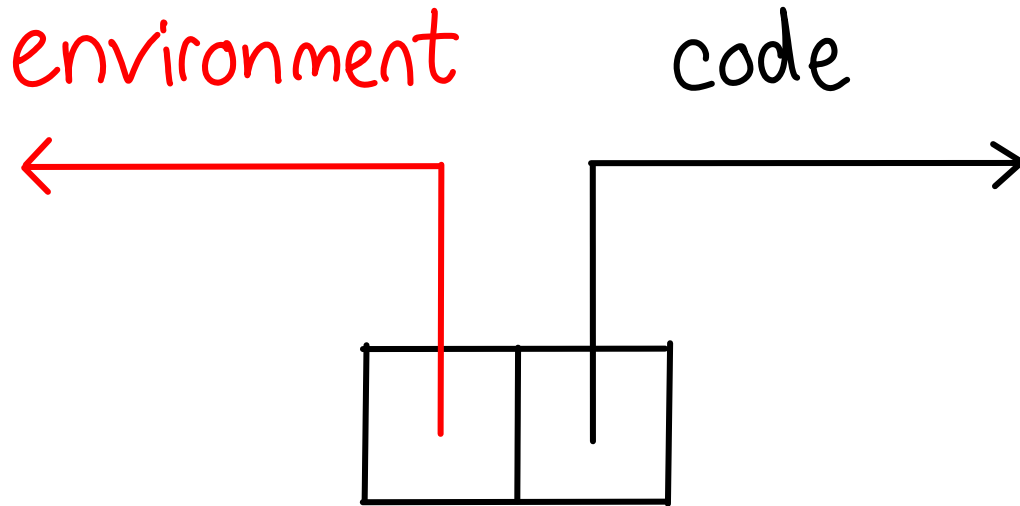
### Ex3 Free variables



### Ex3 Free variables



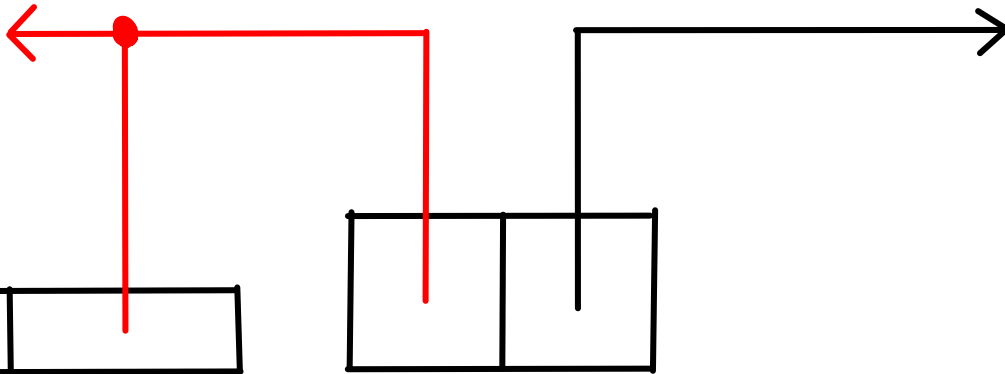
# Closure (the "double-bubble" model)



# Closure (the "double-bubble" model)

environment

code



new activation record

Ex 4

## Higher-order functions

x	
y	

sp

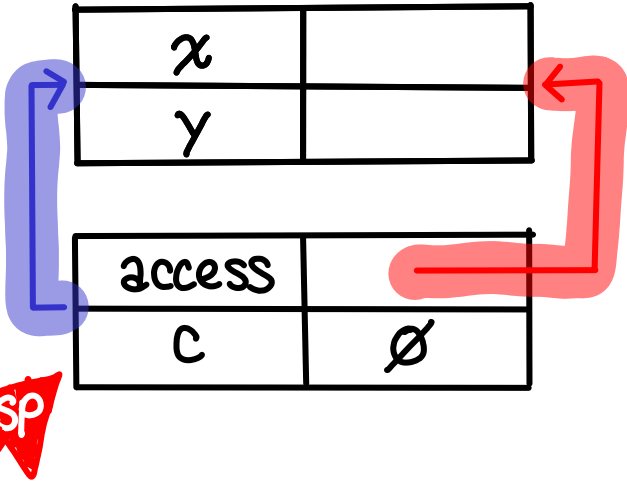
```
function f(c) {  
    return function() {  
        return ++c;  
    }  
}
```

```
var x = f(0);  
var y = f(1);  
console.log(x());
```



**Ex 4**

## Higher-order functions



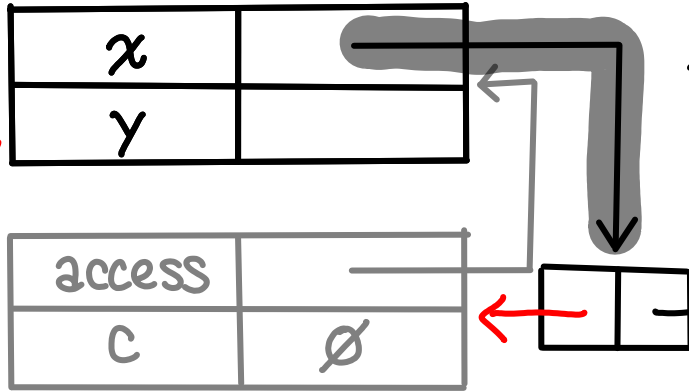
```
function f(c) {  
  return function() {  
    return ++c;  
  }  
}
```

```
var x = f(∅);  
var y = f(1);  
console.log(x());
```



**Ex 4**

## Higher-order functions



x	
y	

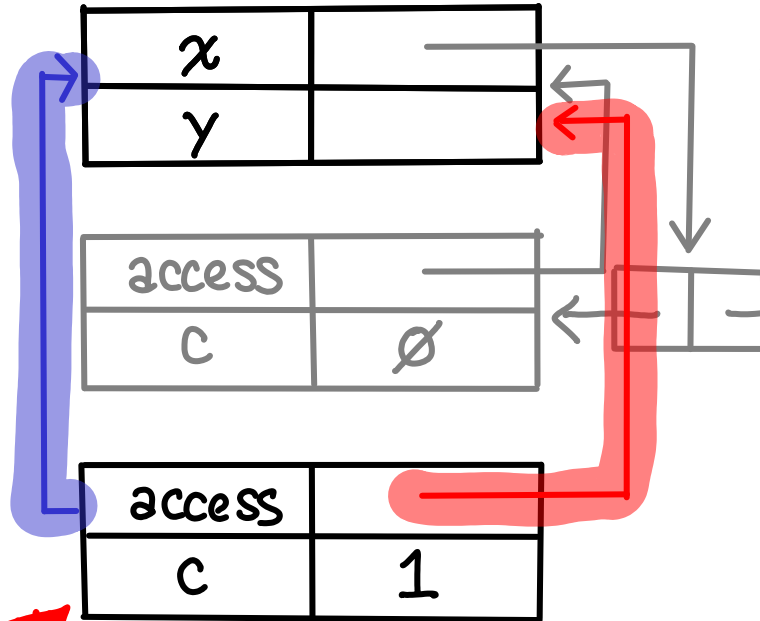
access	
c	∅

```
function f(c) {  
  return function() {  
    return ++c;  
  }  
}
```

```
var x = f(∅);  
var y = f(1);  
console.log(x());
```

Ex 4

## Higher-order functions

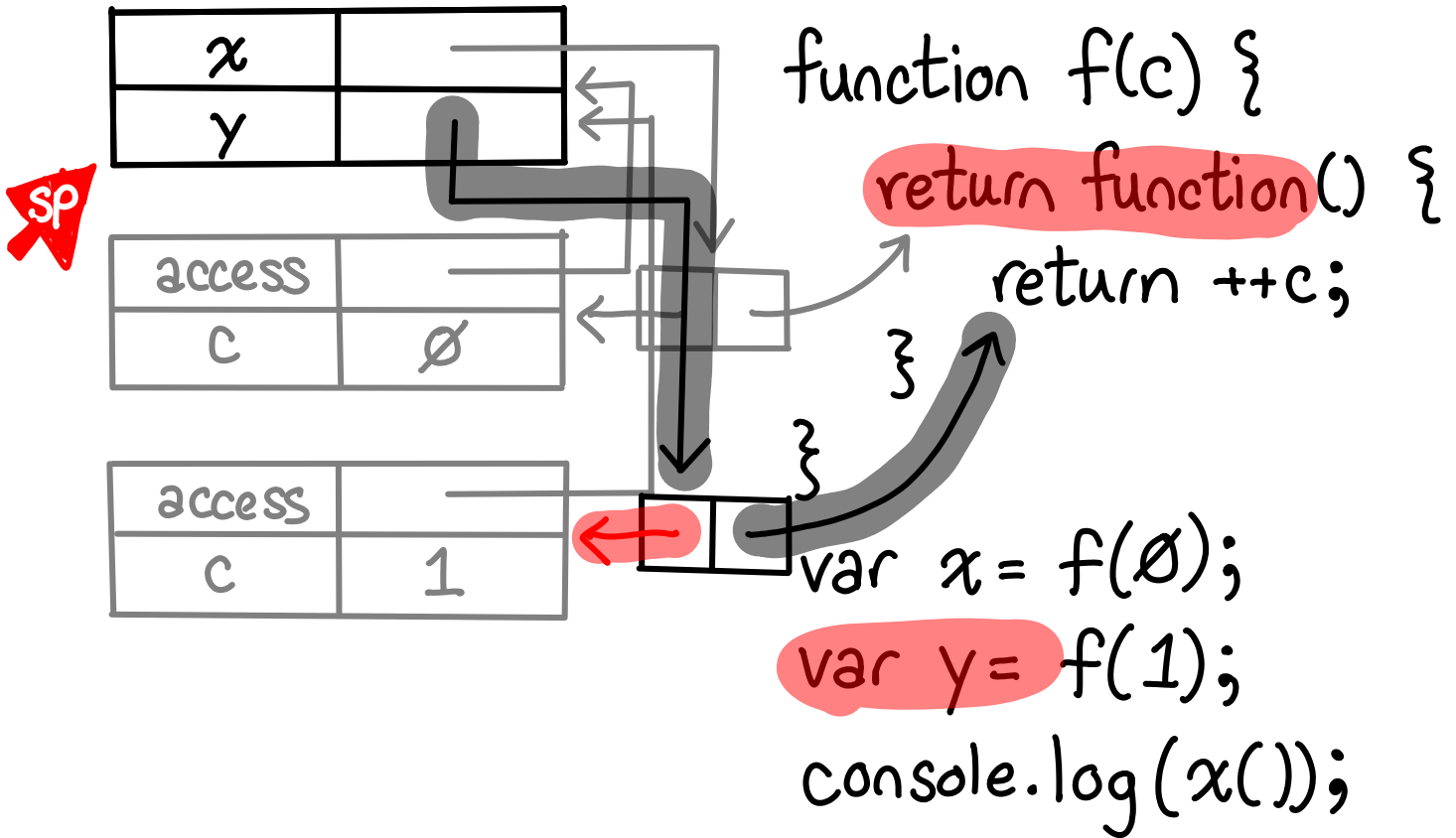


```
function f(c) {  
  return function() {  
    return ++c;  
  }  
}
```

```
var x = f( $\emptyset$ );  
var y = f(1);  
console.log(x());
```

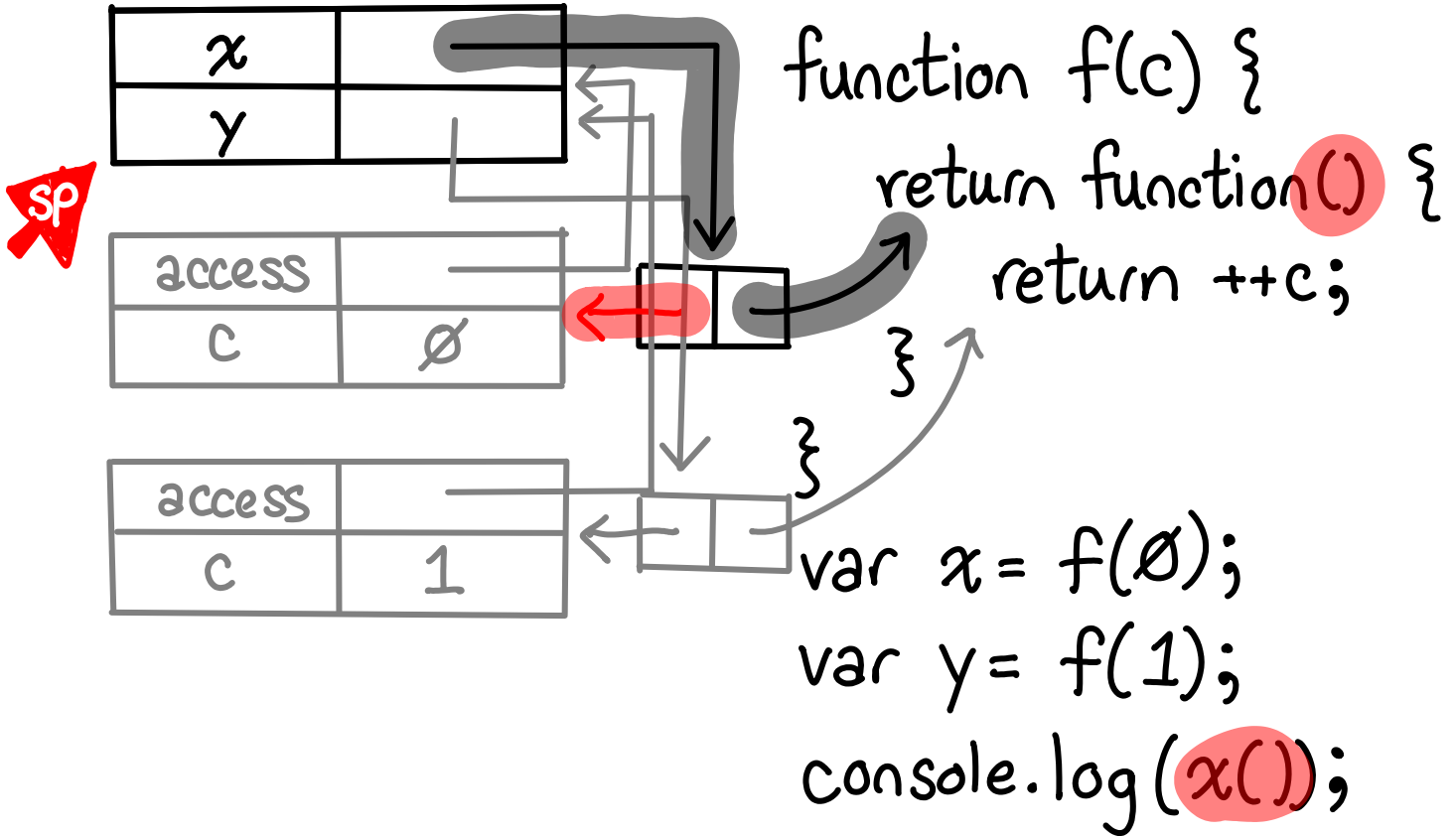
SP

## Ex 4 Higher-order functions



## Ex 4

## Higher-order functions



## Ex 4

## Higher-order functions

x	—
y	—

access	—
c	∅

access	—
c	1

access	—
--------	---

```
function f(c) {  
  return function() {  
    return ++c;  
  }  
}
```

```
var x = f(∅);  
var y = f(1);  
console.log(x());
```



sp

## Ex 4

## Higher-order functions

x	—
y	~~~~

access	—
c	∅

access	
c	1

access	
--------	--

```
function f(c) {  
  return function() {  
    return ++c;  
  }  
}
```

```
var x = f(∅);  
var y = f(1);  
console.log(x());
```



## Ex 4

## Higher-order functions

x	—
y	~~~~

access	—
c	1

access	—
c	1

access	—
--------	---

```
function f(c) {  
  return function() {  
    return ++c;  
  }  
}
```

```
var x = f(0);  
var y = f(1);  
console.log(x());
```

Ex 4

$x$	
$y$	

access	
c	<u>1</u>

access	
c	1

```
function f(c) {  
    return function() {  
        return ++c;  
    }  
}
```

```
var x = f(0);  
var y = f(1);
```

```
console.log(x());
```

1

You can store **data** with  
activation frames

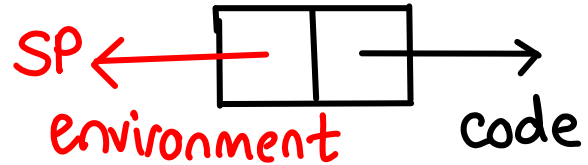
However, the caller must then manage both the chart function (assuming you have multiple types of charts to pick from) and the configuration object. To bind the chart configuration to the chart function, we need a [closure](#):

```
function chart(config) {  
  return function() {  
    // generate chart here, using `config.width` and `config.height`  
  };  
}
```

"Towards Reusable Charts" —Mike Bostock

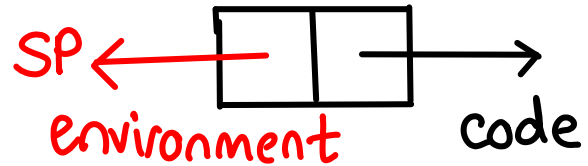
# Environment Model

Function definition

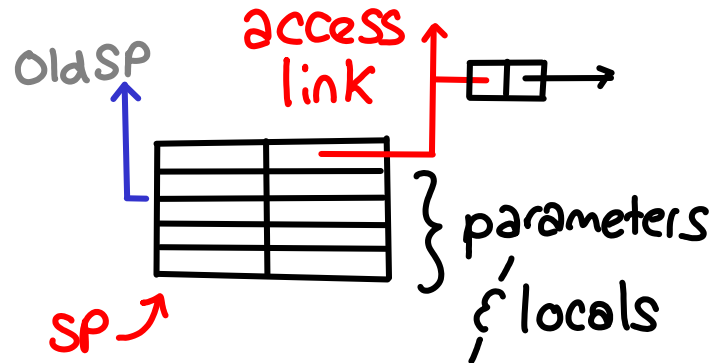


# Environment Model

Function definition

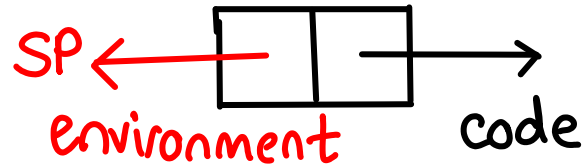


Function application

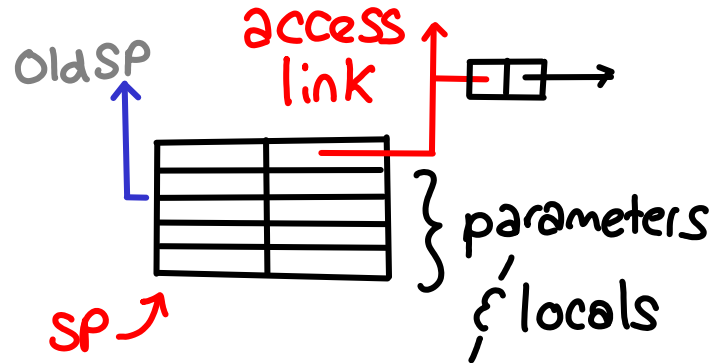


# Environment Model

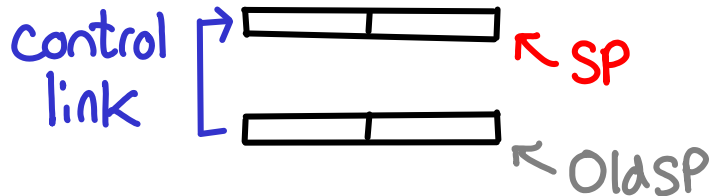
Function definition



Function application



Function return



# Bonus

- Parameter passing
- GC concerns
- JavaScript "blocks" aren't scopes
- Variable Hoisting



```
var y = 10;
function g() {
    function f() {
        y++;
    }
    f();
    var y = 0;
    console.log(y);
}
g();
console.log(y);
```