ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Лекция № 2 / **10**
21.11.2017 г.

«БОЛЕЕ ЛУЧШЕ», ЧЕМ C++

- Java (1995 г.)
- C# (2000 г.)
- D (2001 г.)

- При всех различиях (виртуальная машина, сборка мусора, модули и т.д.) объектная система та же («скандинавская»).

- 1984 г. Впервые использован в системе NeXTSTEP (компьютер NeXT).

- Сочетание C и Smalltalk. Динамическая диспетчеризация.

Objective-C

```objc
#import <Foundation/Foundation.h>

@interface SampleClass:NSObject
- (void)sampleMethod;
@end

@implementation SampleClass

- (void)sampleMethod{
    NSLog(@"Hello, World! \n");
}

@end

int main()
{
   /* my first program in Objective-C */
   SampleClass *sampleClass = [[SampleClass alloc]init];
   [sampleClass sampleMethod];
   return 0;
}
```

Hello World на Objective C

```objc
#import <Foundation/Foundation.h>

@interface Square:NSObject
{
   float area;
}
- (void)calculateAreaOfSide:(CGFloat)side;
- (void)printArea;
@end

@implementation Square
- (void)calculateAreaOfSide:(CGFloat)side
{
    area = side * side;
}
- (void)printArea
{
    NSLog(@"The area of square is %f",area);
}
@end

@interface Rectangle:NSObject
{
   float area;
}
- (void)calculateAreaOfLength:(CGFloat)length andBreadth:(CGFloat)breadth;
- (void)printArea;
@end
```

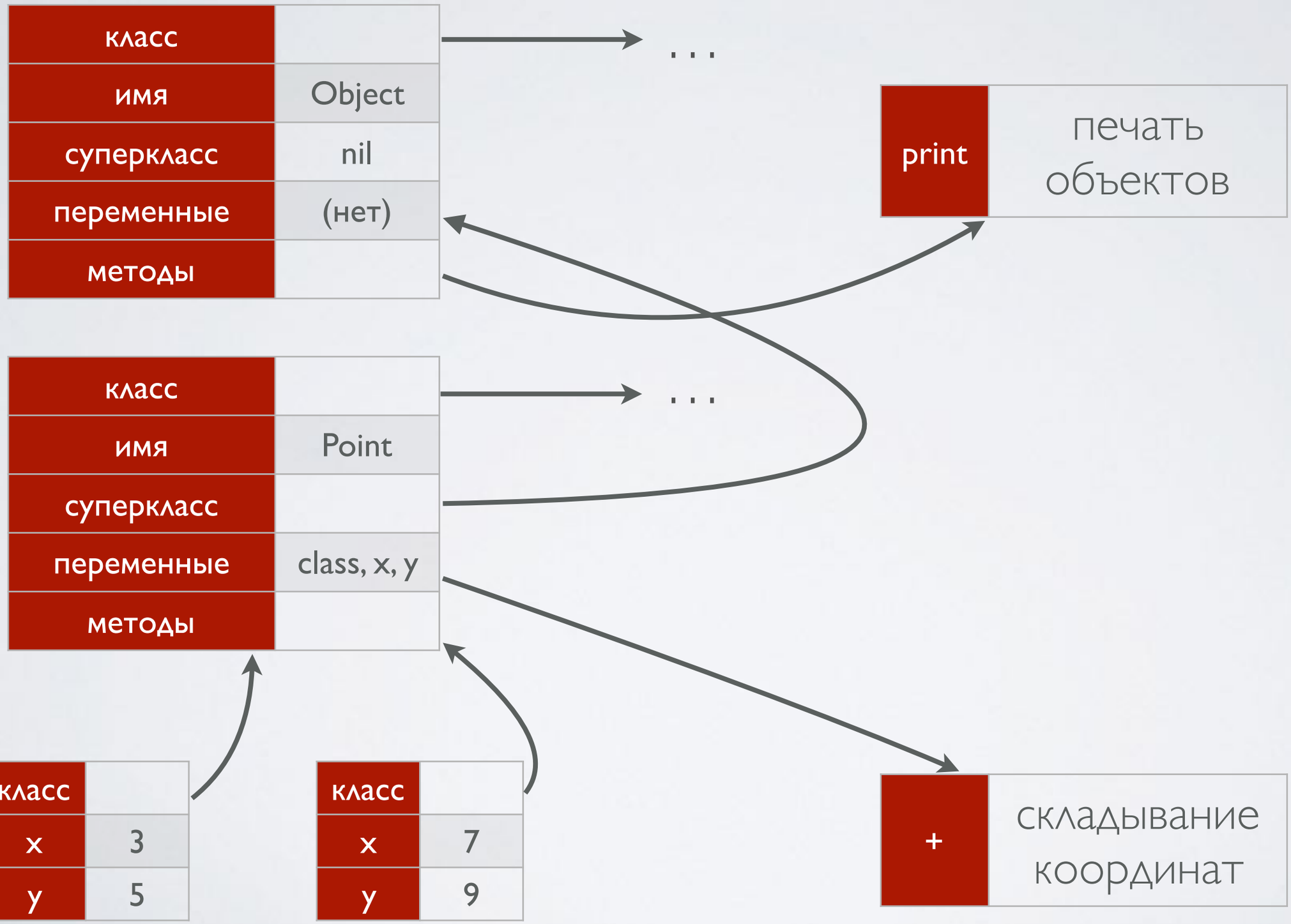
Динамическая типизация (ч. 1)

```objc
@implementation Rectangle
- (void)calculateAreaOfLength:(CGFloat)length andBreadth:(CGFloat)breadth
{
    area = length * breadth;
}
- (void)printArea
{
    NSLog(@"The area of Rectangle is %f",area);
}
@end

int main()
{
    Square *square = [[Square alloc]init];
    [square calculateAreaOfSide:10.0];
    Rectangle *rectangle = [[Rectangle alloc]init];
    [rectangle calculateAreaOfLength:10.0 andBreadth:5.0];
    NSArray *shapes = [[NSArray alloc]initWithObjects: square, rectangle,nil];
    id object1 = [shapes objectAtIndex:0];
    [object1 printArea];
    id object2 = [shapes objectAtIndex:1];
    [object2 printArea];
    return 0;
}
```

Динамическая типизация (ч. 2)

- Диалект Smalltalk (1987 г.).
  «Self is like Smalltalk, only more so»
- «КЛАССЫ НЕ НУЖНЫ !!!» (а также переменные, числа и управляющие структуры).
- Экспериментальный язык:
- Компилятор Self написан на Self.
- JIT-компиляция.
- Sun Microsystems многие наработки впоследствии перенесла в JVM.

ОБЪЕКТЫ В SMALLTALK



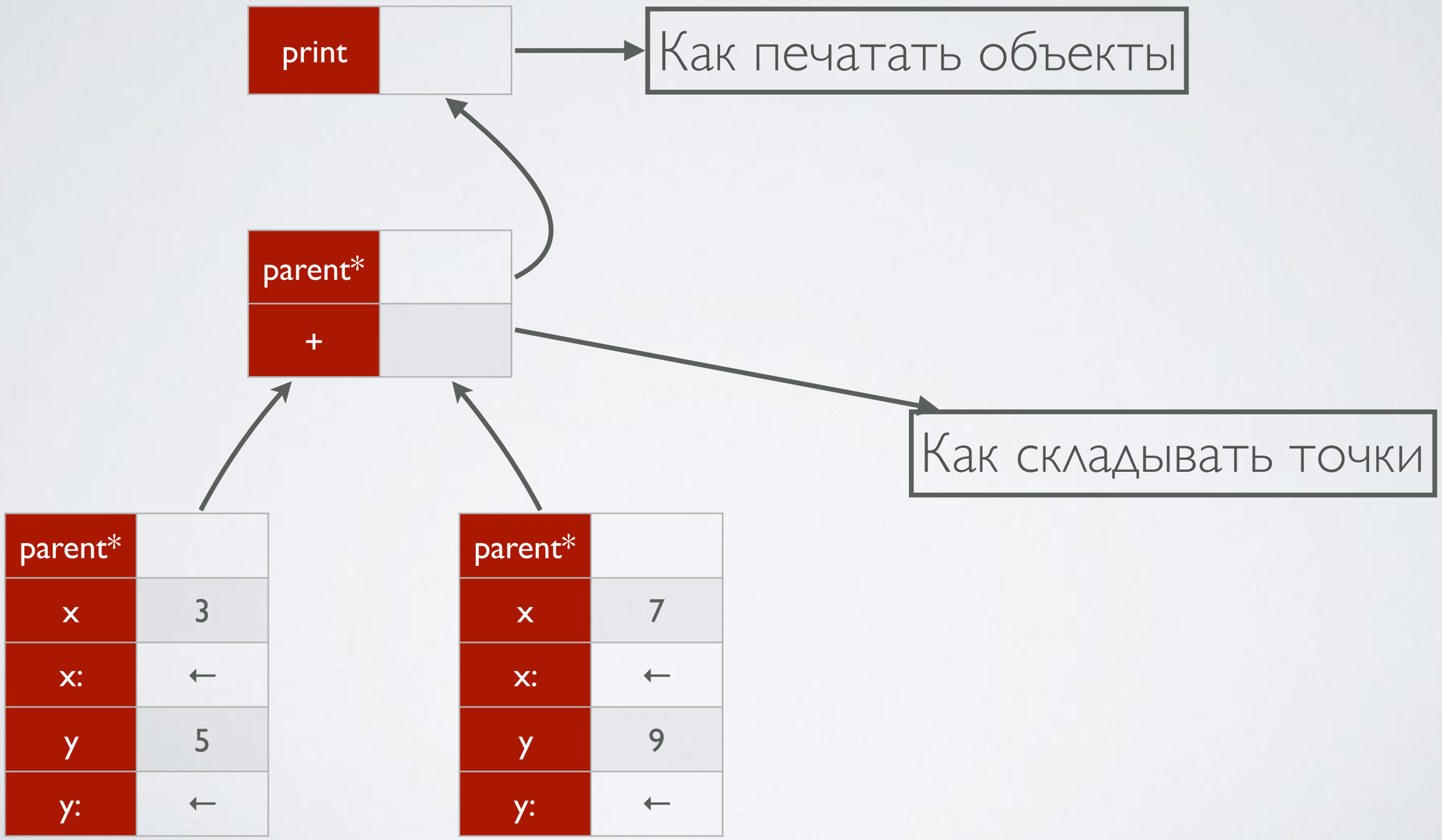
- Каждый объект-точка в Smalltalk содержит ссылку на класс (**Point**) и координаты **x**, **y**.

- Класс **Point** задаёт и формат (список атрибутов), и поведение (методы).

- Дополнительные формат и поведение наследуются у **Object** через ссылку на суперкласс.

- Сами объекты-классы принадлежат ещё какому-то классу (метаклассу).

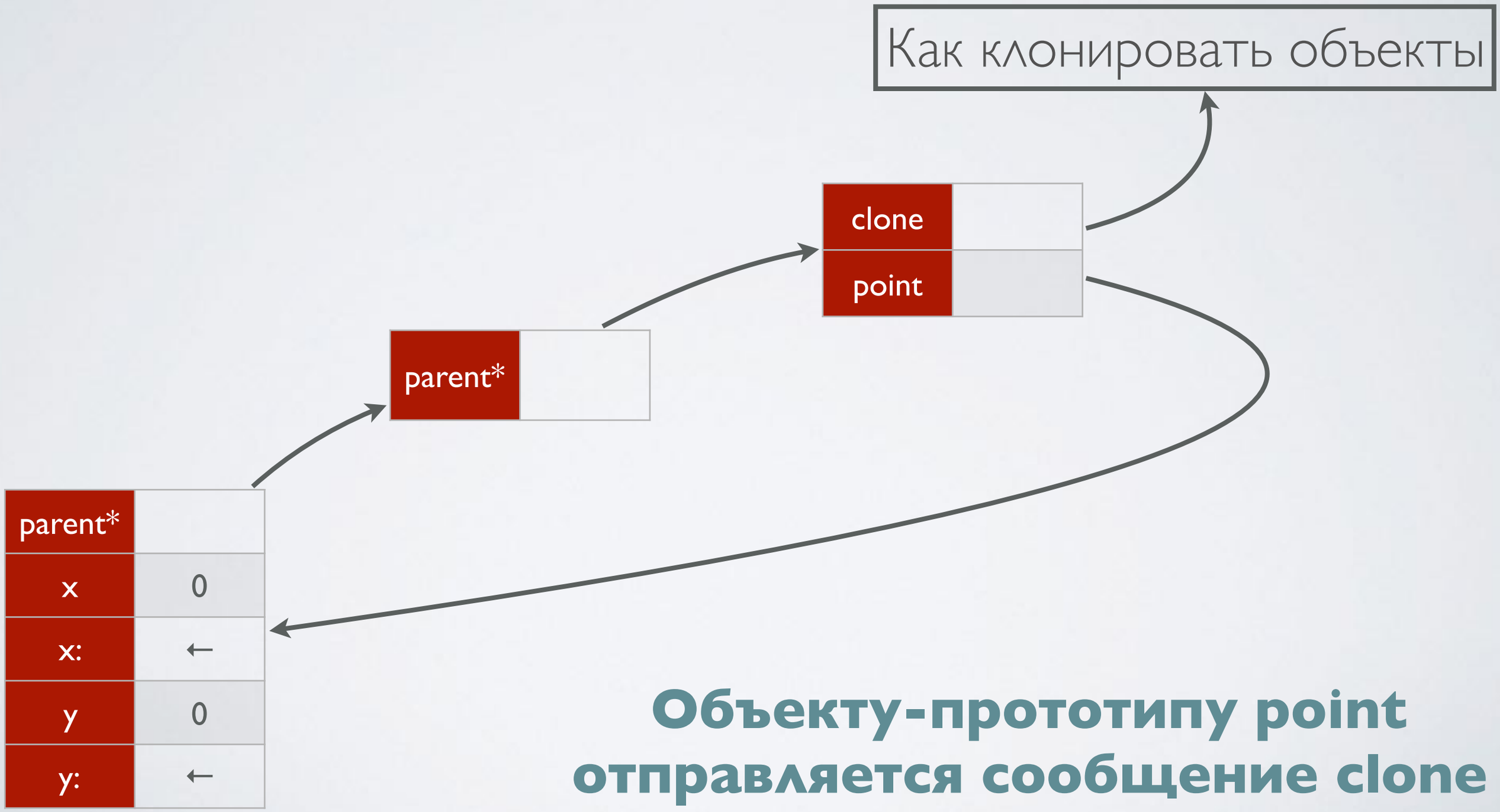
ОБЪЕКТЫ В SELF



- В Self каждый объект-точка задаёт свой формат, но ссылается на другие объекты-родители, чтобы заимствовать у них поведение, общее для всех точек.

- Объект, содержащий общее поведение, в свою очередь, ссылается на еще один объект, содержащий поведение всех объектов в системе. У этого корневого объекта нет родителя.

ПОРОЖДЕНИЕ ОБЪЕКТОВ В SELF



Объекту-прототипу point отправляется сообщение clone

JAVASCRIPT

- 1995 г. Язык создавался для поддержки динамических страниц в браузере Netscape Navigator (прародителе Mozilla Firefox).

- Динамическая типизация, наследование с помощью прототипов — всё как мы любим.

```
const hibye = {
  hi: function(name) { alert("Hi, " + name); },
  bye: function(name) { alert("Bye, " + name); },
};

hibye.hi('Oleg');   // Hi, Oleg
hibye.bye('Oleg');  // Bye, Oleg
```

```
hibye.hello = name => {
  alert("Hello, " + name);
};

hibye.hello('Oleg');  // Hello, Oleg
```

Классы не нужны, объекты рулят

```
function User(name, birthday) {
  function calcAge() {
    return new Date().getFullYear() - birthday.getFullYear();
  }

  return {
    sayHi() {
      alert(name + ', age:' + calcAge());
    }
  };
}

let user = User("John", new Date(2000,0,1));
user.sayHi(); // John
```

Функция-конструктор

- Создаётся новый объект (**{}**).
- Функция **User** запускается в контексте (*binding*) этого объекта, т.е. **this** указывает на него.
- **new User** возвращает созданный объект после вызова **User**.

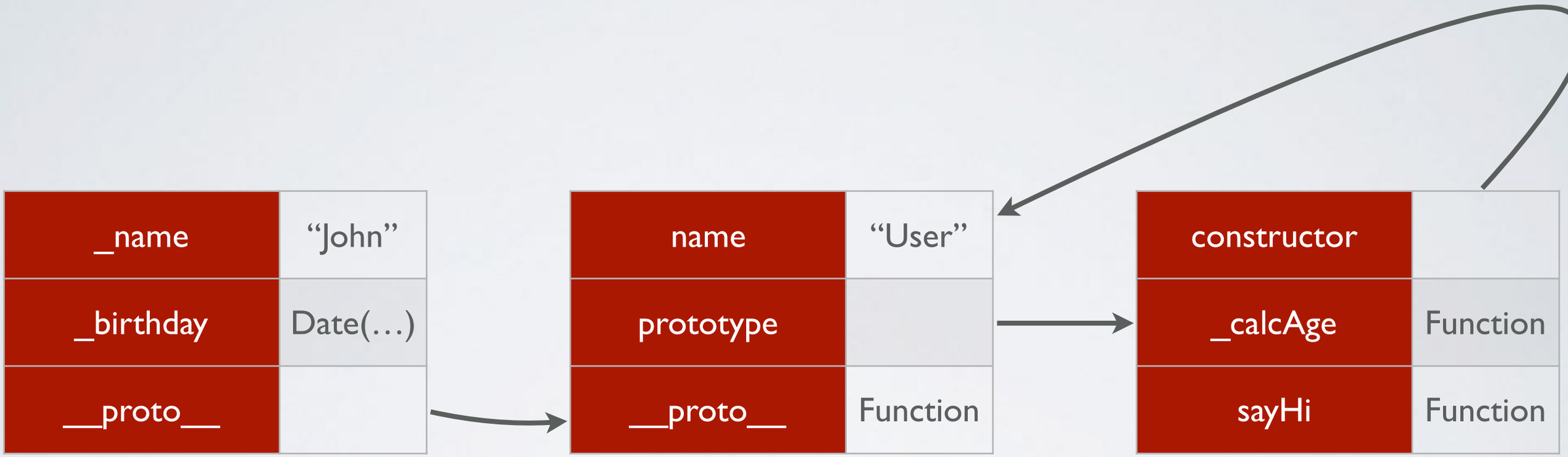
```
function User(name) {
  this.sayHi = function() {
    alert(name);
  };
}

let user = new User("John");
user.sayHi(); // John
```

Использование new

```
function User(name, birthday) {
  this._name = name;
  this._birthday = birthday;
}

User.prototype._calcAge = function() {
  return new Date().getFullYear() - this._birthday.getFullYear();
};

User.prototype.sayHi = function() {
  alert(this._name + ', age:' + this._calcAge());
};

let user = new User("John", new Date(2000,0,1));
user.sayHi(); // John
```

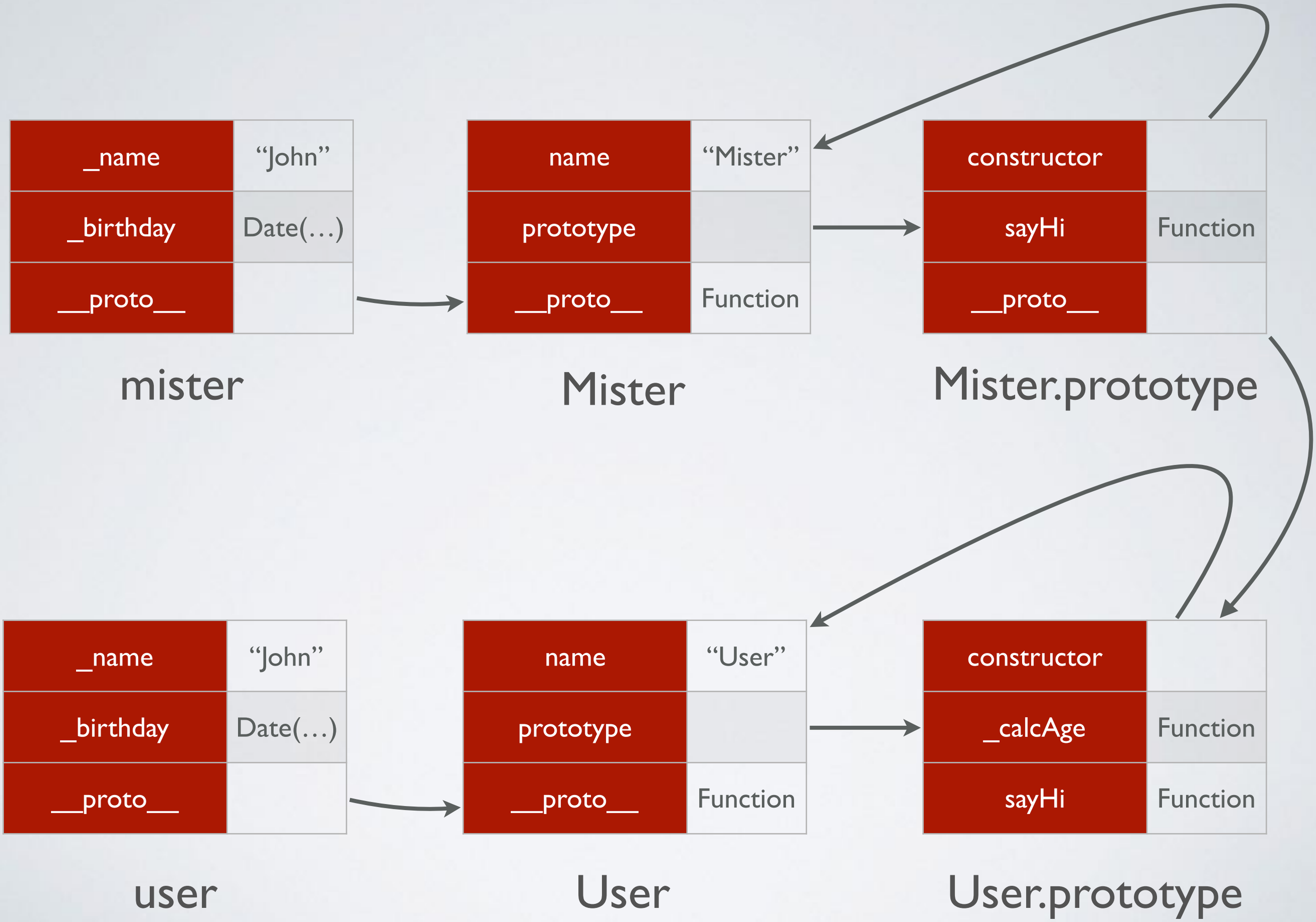
Использование прототипов

СТРУКТУРА ОБЪЕКТОВ



```
function Mister(name, birthday) {
  User.call(this, name, birthday);
}

Mister.prototype = Object.create(User.prototype);
Mister.prototype.constructor = Mister;
Mister.prototype.sayHi = function() {
  alert('Mr. ' + this._name + ', age:' + this._calcAge());
}

let mister = new Mister("John", new Date(2000,0,1));
mister.sayHi(); // John
```

Наследование


_name
“John”
_birthday
Date(…)
__proto__
mister
name
“Mister”
prototype
__proto__
Function
Mister
constructor
sayHi
Function
__proto__
Mister.prototype
_name
“John”
_birthday
Date(…)
__proto__
user
name
“User”
prototype
__proto__
Function
User
constructor
_calcAge
Function
sayHi
Function
User.prototype

RUBY

- 1995 г.
- Динамическая типизация, система классов, заимствованная у Smalltalk.
- Ruby ~= Perl + Smalltalk.

```ruby
require 'date'

class User
  def initialize(name, birthday)
    @name, @birthday = name, birthday
  end

  def say_hi
    puts "#{@name}, age: #{age}"
  end

  protected

  def age
    Date.today.year - @birthday.year
  end
end

class Mister < User
  def say_hi
    puts "Mr. #{@name}, age: #{age}"
  end
end

user = User.new('John', Date.new(2000, 1, 1))
user.say_hi

mister = Mister.new('John', Date.new(2000, 1, 1))
mister.say_hi
```

```ruby
sum = 0

1.upto(7) do |i|
  puts i
  sum += i
end

puts sum
```

```ruby
(1..100).
  select { |i| i.odd? }.
  take(9).
  inject(0) { |s, i| s + i*2 }
```

Блоки (реализация замыканий)

```ruby
class Integer
  def iterate(arg)
    self.times do
      arg = yield arg
    end

    arg
  end
end

puts 10.iterate(0) { |x| x + 1 }
```

“Reopening a class”

```ruby
module RoundHelpers
  def area
    Math::PI * radius**2
  end

  def perimeter
    2 * Math::PI * radius
  end
end

class Circle < Shape
  # ...
  include RoundHelpers
end

class RoundButton < Button
  # ...
  include RoundHelpers
end
```

Mixin

```ruby
array = [1, 2, 3]
class <<array
  def foo
    puts "Foo!"
  end
end
```

```ruby
array.foo
#=> Foo!

[1, 2, 3].foo
#=> NoMethodError: undefined method `foo' for [1, 2, 3]:Array
```

«Singleton»

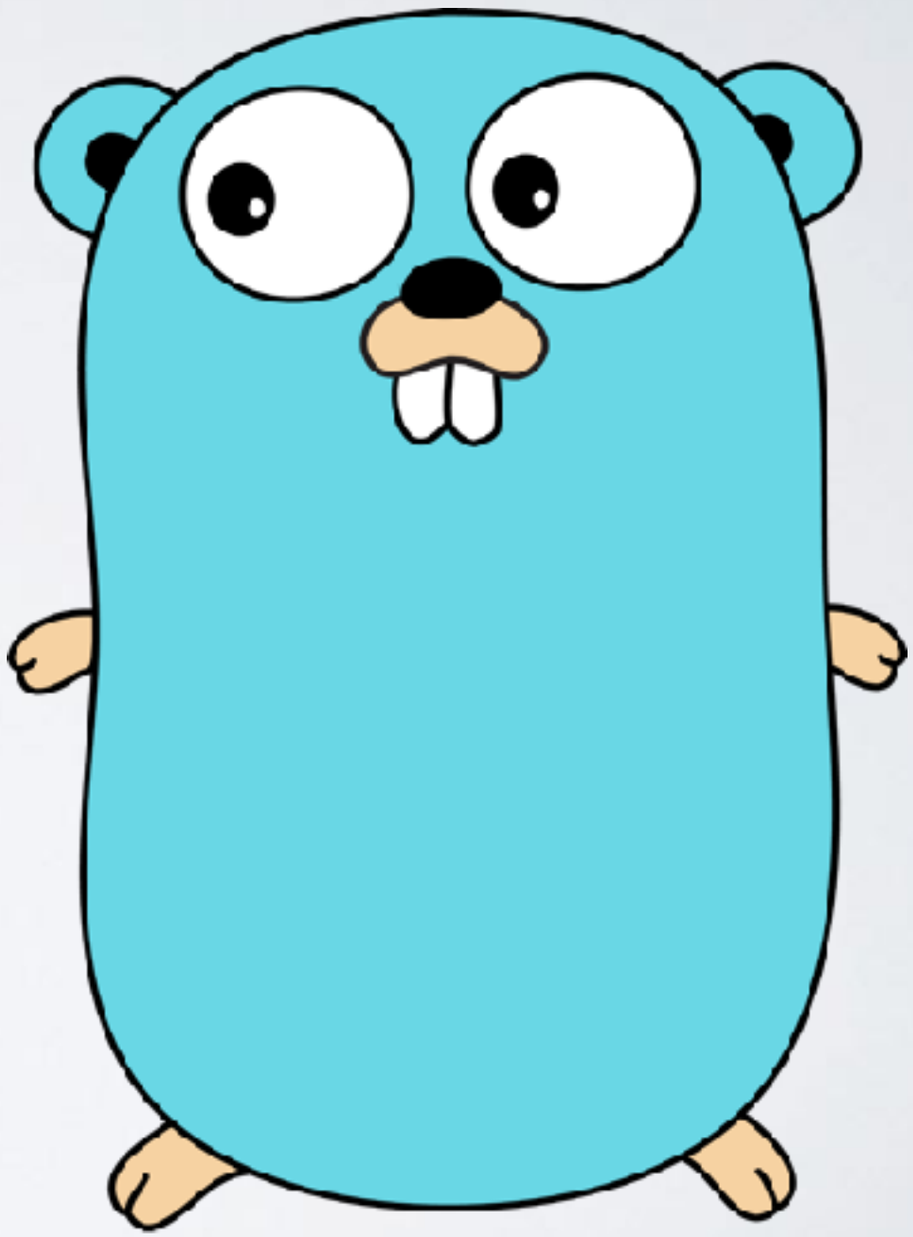
- 1991 г.
- Динамическая типизация и диспетчеризация.
- «Правильный JavaScript».
- Есть классы и множественное наследование.

```python
from datetime import date

class User:
    def __init__(self, name, birthday):
        self.name = name
        self.birthday = birthday

    def sayhi(self):
        print("{}, age: {}".format(self.name, self.age()))

    def age(self):
        return date.today().year - self.birthday.year

user = User("John", date(2000, 1, 1))
user.sayhi()

class Mister(User):
    def sayhi(self):
        print("Mr. {}, age: {}".format(self.name, self.age()))

mister = Mister("John", date(2000, 1, 1))
mister.sayhi()
```

А ЧТО ТАМ В СТАТИЧЕСКИХ ЯЗЫКАХ?

GO

- 2009 г. (Google)
- «Современный C».



```go
package main

import (
    "fmt"
    "time"
)

type User struct {
    Name     string
    Birthday time.Time
}

func (u User) SayHi() {
    fmt.Printf("%s, age: %d\n", u.Name, u.CalcAge())
}

func (u User) CalcAge() int {
    return time.Now().Year() - u.Birthday.Year()
}

type Mister struct {
    User
}

func (m Mister) SayHi() {
    fmt.Printf("Mr. %s, age: %d\n", m.Name, m.CalcAge())
}
```

```go
func main() {
    birthday := time.Date(2000, time.January, 1,
        0, 0, 0, 0, time.UTC)
    u := User{
        Name:     "John",
        Birthday: birthday,
    }
    u.SayHi()

    m := Mister{
        User{
            Name:     "John",
            Birthday: birthday,
        },
    }
    m.SayHi()
}
```

Невиртуальное наследование

```
type Greeter interface {
    SayHi()
}

func main() {
    birthday := time.Date(2000, time.January, 1,
        0, 0, 0, 0, time.UTC)

    greeters := []Greeter{
        User{
            Name:     "John",
            Birthday: birthday,
        },
        Mister{
            User{
                Name:     "John",
                Birthday: birthday,
            },
        },
    }

    for _, g := range greeters {
        g.SayHi()
    }
}
```

Реализация интерфейса

Наследование

Наследование интерфейса

В Go достаточно реализовать все методы интерфейса.

Наследование реализации

В Go используется композиция структур. Методы получаются невиртуальные.

О НАСЛЕДОВАНИИ …

Однажды я участвовал во встрече группы пользователей Java, где Джеймс Гослинг (создатель Java) был ведущим спикером. Когда дело дошло до ответов на вопросы, кто-то спросил: «Если бы вы сейчас заново создавали Java, что бы вы сделали иначе?». Гослинг ответил: «Я бы выкинул классы». Когда смех утих, он объяснил, что проблемой были не классы как таковые, а наследование реализации (ключевое слово **extends**). Наследование интерфейса (**implements**) является предпочтительным. Вам следует по возможности избегать наследования реализации.

Аллен Голуб. «Почему extends — это зло?»
JavaWorld, 2003 г.

RUST

- 2010 г. (Mozilla Research)
- Изначально создавался как замена C++ для разработки движка браузера Mozilla.

```rust
extern crate chrono;

use chrono::prelude::*;

fn calc_age(date: &NaiveDate) -> i32 {
    Utc::now().year() - date.year()
}

trait Greet {
    fn say_hi(&self);
}

struct User {
    name: String,
    birthday: NaiveDate,
}

impl Greet for User {
    fn say_hi(&self) {
        println!("{}, age: {}", self.name, calc_age(&self.birthday));
    }
}

struct Mister {
    name: String,
    birthday: NaiveDate,
}

impl Greet for Mister {
    fn say_hi(&self) {
        println!("Mr. {}, age: {}", self.name, calc_age(&self.birthday));
    }
}
```

```rust
fn main() {
    let user = User {
        name: "John".to_string(),
        birthday: NaiveDate::from_ymd(2000, 1, 1),
    };

    let mister = Mister {
        name: "John".to_string(),
        birthday: NaiveDate::from_ymd(2000, 1, 1),
    };

    let v: Vec<&Greet> = vec![&user, &mister];
    for u in v.iter() {
        u.say_hi();
    }
}
```