
# Лекция 3:
# прототипное наследование

**Tinkoff**.ru

# Нарисуем сову

```
const owl = {
  name: 'Маленький сов'
};
```

| owl |
| --- |
| **name:** *Маленький сов* |

# Нарисуем сову

```
const owl = {
  name: 'Маленький сов',
  says: function () {
    console.log(this.name + ' говорит: ух!');
  }
};
```

| owl |
| --- |
| **name:** *Маленький сов*<br>**says:** *function () {...}* |

# И ещё одну

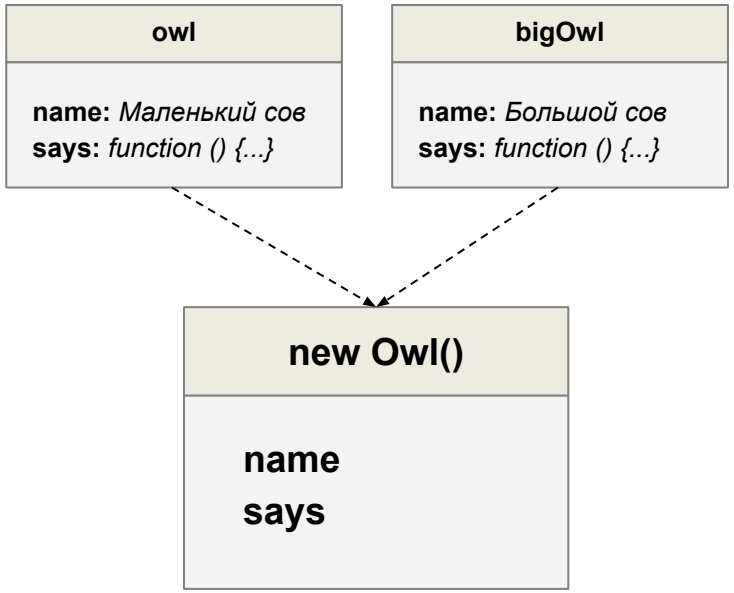
```
const owl = {
   name: 'Маленький сов',
   says: function () {
      console.log(this.name + ' говорит: ух!');
   }
};

const bigOwl = {
   name: 'Большой сов',
   says: function () {
      console.log(this.name + ' говорит: ух!');
   }
};
```

| owl | bigOwl |
| --- | --- |
| **name:** *Маленький сов*<br>**says:** *function () {...}* | **name:** *Большой сов*<br>**says:** *function () {...}* |

# КАК СОЗДАТЬ МНОГО СОВ

# Конструктор

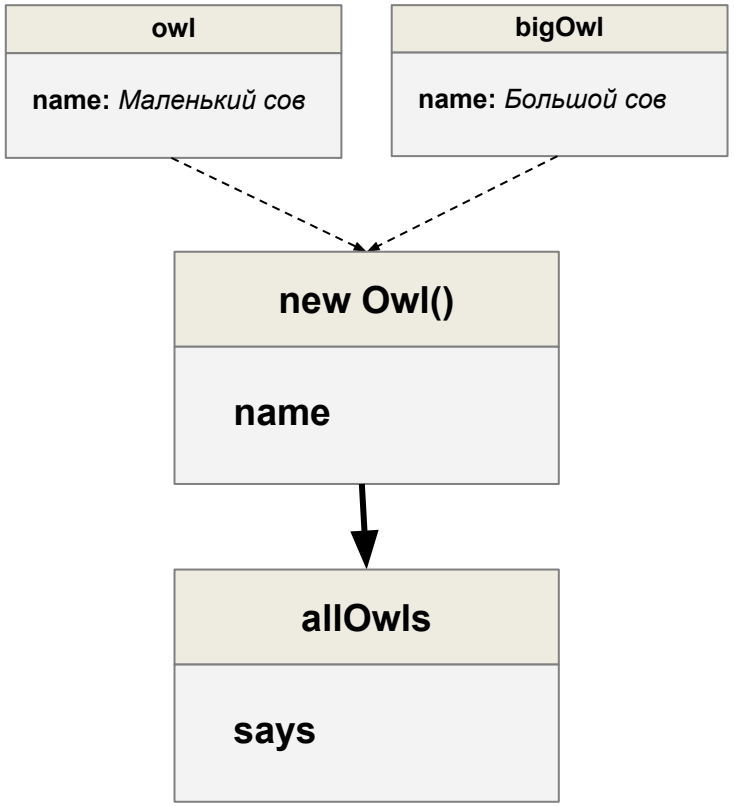
```
function Owl(name) {
   this.name = name;
   this.says = function () {
      console.log(this.name + ' говорит: ух!');
   }
}

const owl = new Owl('Маленький сов');
const bigOwl = new Owl('Большой сов');
```



У КАЖДОЙ СОВЫ
СВОЙ ЭКЗЕМПЛЯР
**SAYS**

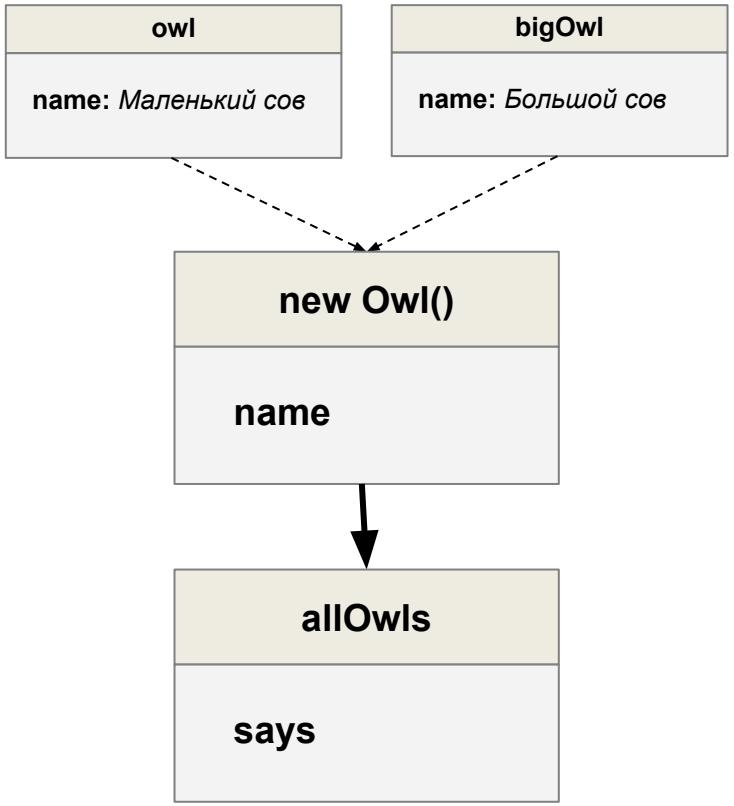
# Прототип

```
const allOwls = {
  says: function () {
    console.log(this.name + ' говорит: ух!');
  }
};

function Owl(name) {
  this.name = name;
}

Owl.prototype = allOwls;

const owl = new Owl('Маленький сов');
const bigOwl = new Owl('Большой сов');
```

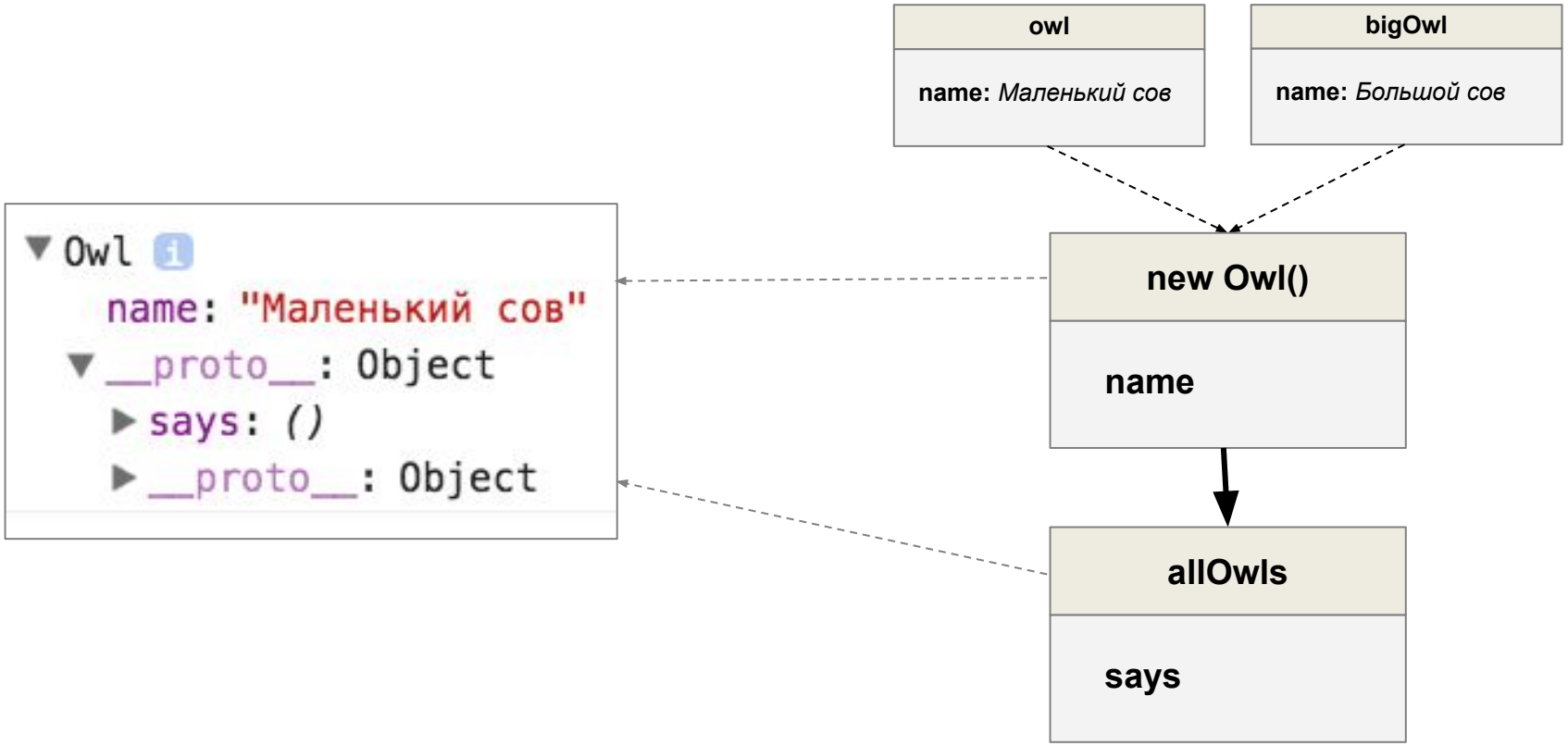


# Класс

```
const allOwls = {
  says: function () {
    console.log(this.name + ' говорит: ух!');
  }
};

function Owl(name) {
  this.name = name;
}

Owl.prototype = allOwls;

const owl = new Owl('Маленький сов');
const bigOwl = new Owl('Большой сов');

owl.says();
// Маленький сов говорит: ух!
```



# __proto__


owl
name: Маленький сов
bigOwl
name: Большой сов
Owl
name: "Маленький сов"
__proto__: Object
says: ()
__proto__: Object
new Owl()
name
allOwls
says

# Прототип объекта

# Прототип прототипа объекта – *null*


# У *null* нет :(

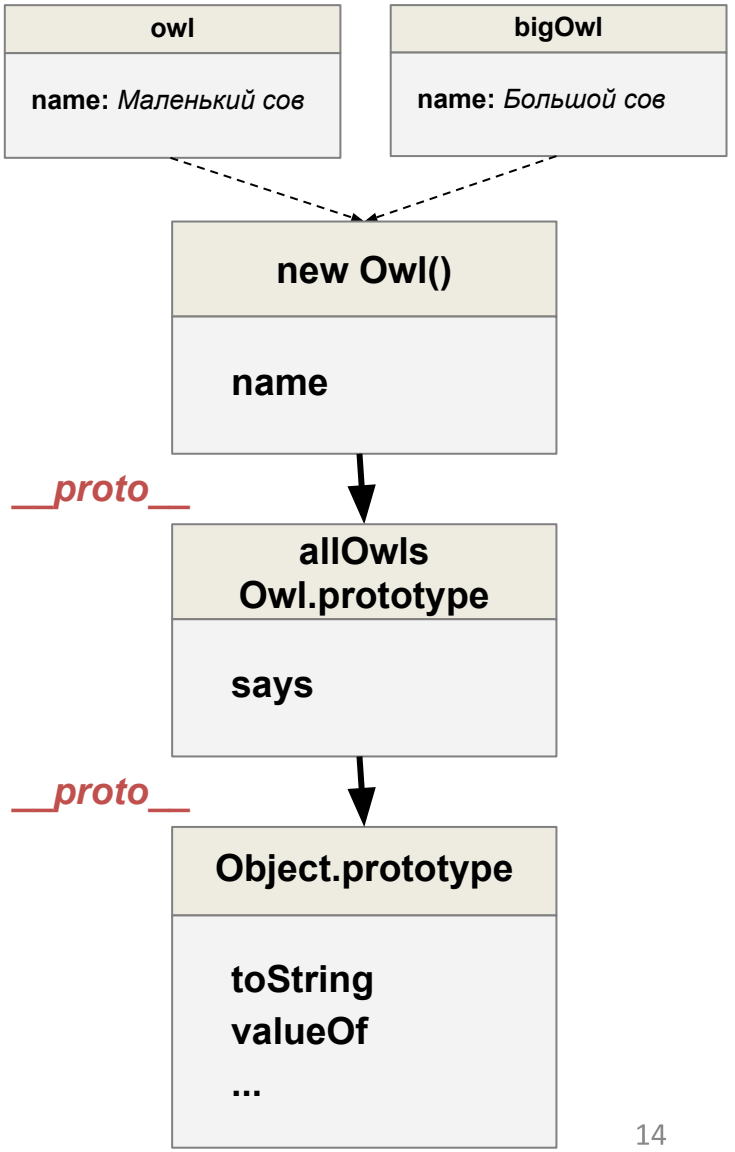
```
> ({}).__proto__
<- ▼Object i
    ▶__defineGetter__: __defineGetter__()
    ▶__defineSetter__: __defineSetter__()
    ▶__lookupGetter__: __lookupGetter__()
    ▶__lookupSetter__: __lookupSetter__()
    ▶constructor: Object()
    ▶hasOwnProperty: hasOwnProperty()
    ▶isPrototypeOf: isPrototypeOf()
    ▶propertyIsEnumerable: propertyIsEnumerable()
    ▶toLocaleString: toLocaleString()
    ▶toString: toString()
    ▶valueOf: valueOf()
    ▶get __proto__: __proto__()
    ▶set __proto__: __proto__()
```

```
> ({}).__proto__.__proto__
<- null
```

```
> ({}).__proto__.__proto__.__proto__
⊗ ▶Uncaught TypeError: Cannot read property '__proto__'
  of null
      at <anonymous>:1:25
```

# Цепочка прототипов

```
const allOwls = {
  says: function () {
    console.log(this.name + ' говорит: ух!');
  }
};

function Owl(name) {
  this.name = name;
}

Owl.prototype = allOwls;

const owl = new Owl('Маленький сов');

owl.toString(); // "[object Object]"
```

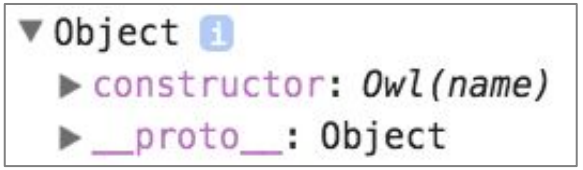


# Цепочка прототипов


# Пустой прототип

Ссылка на прототип создается автоматически

```
function Owl(name) {
  this.name = name;
}

console.log(Owl.prototype);
```



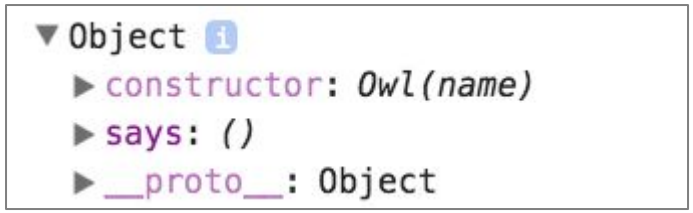
# Ссылка на конструктор

```
function Owl(name) {
  this.name = name;
}

console.log(Owl.prototype.constructor);
```

```
console.log(Owl.prototype.constructor);
function Owl(name) {
    this.name = name;
}
```

# Добавление свойств в прототип

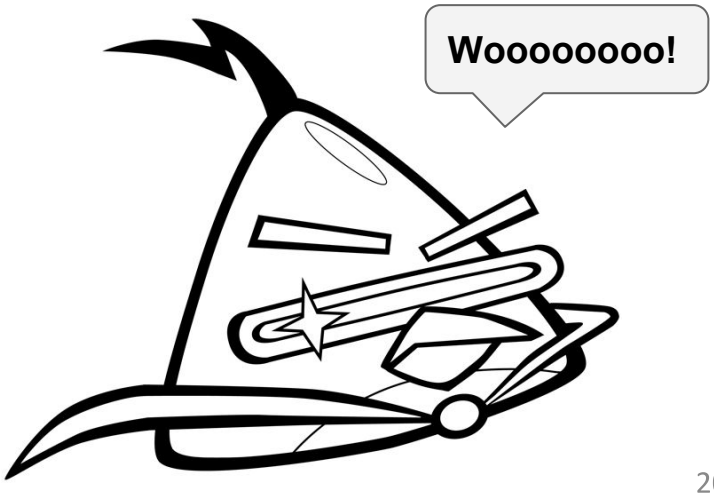
```
function Owl(name) {
  this.name = name;
}

Owl.prototype.says = function() {
  console.log(this.name + ' говорит: ух!');
};

console.log(Owl.prototype);
```



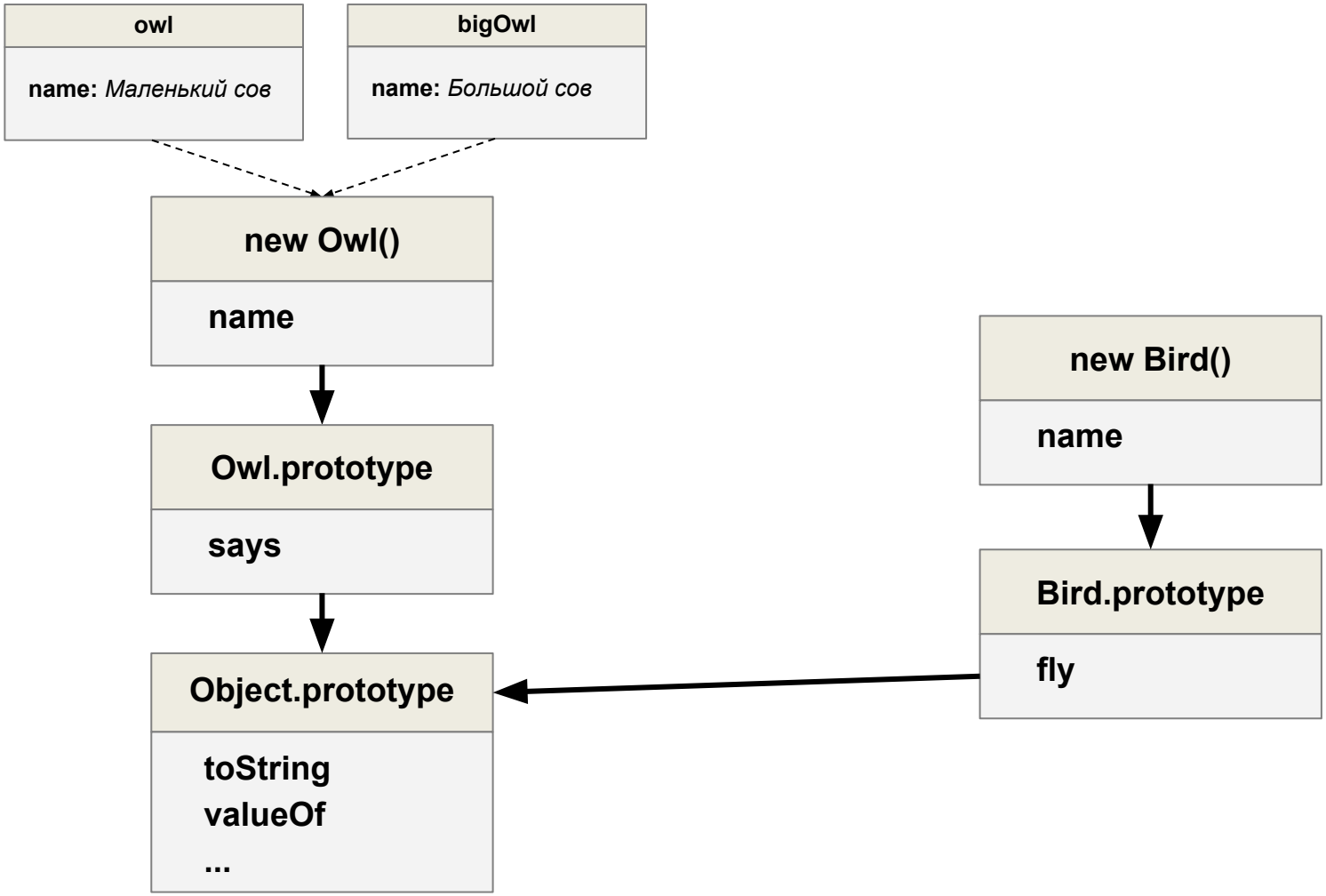
# НАСЛЕДОВАНИЕ

# Наследование — fly

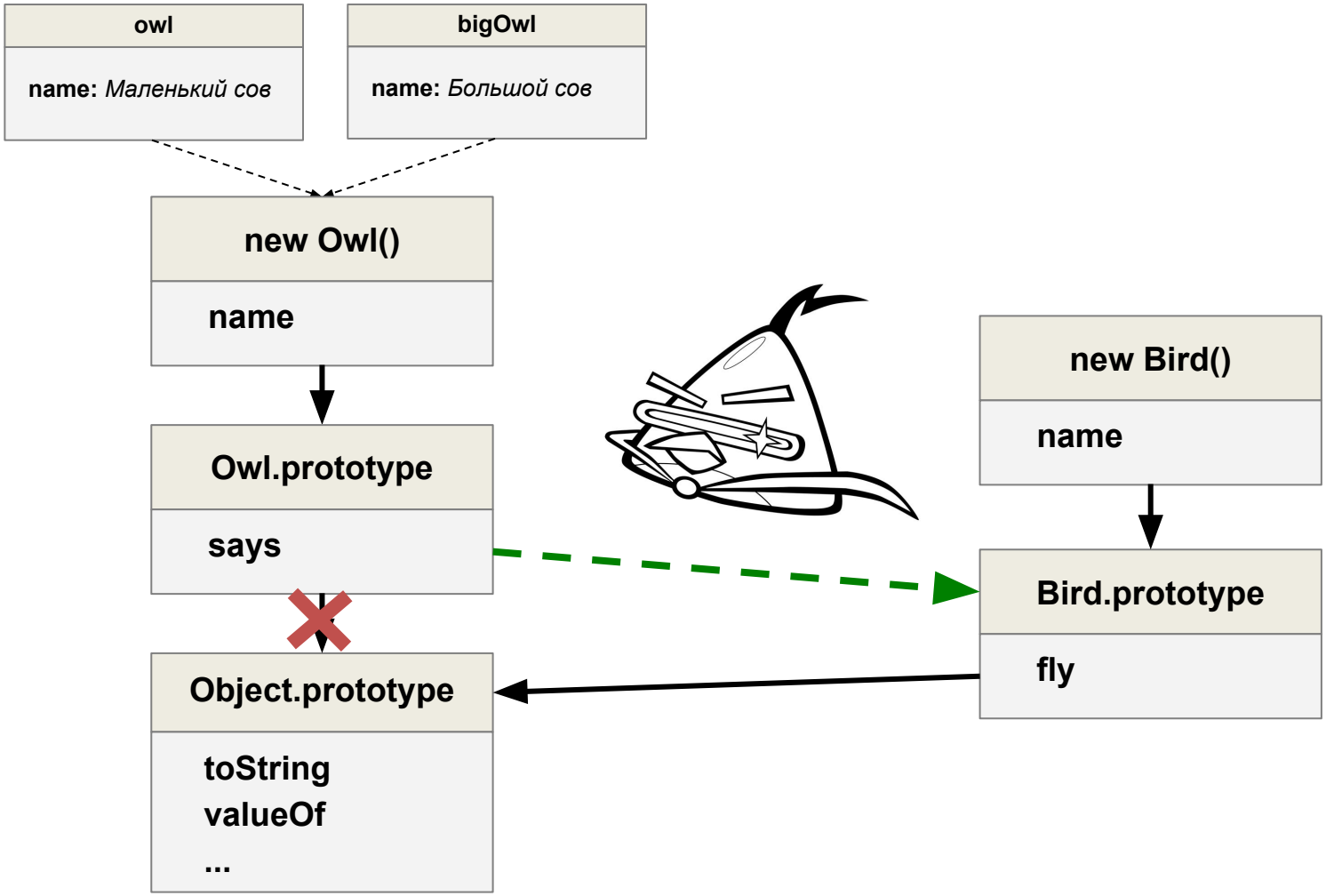
```
function Bird(name) {
  this.name = name;
}

Bird.prototype.fly = function() {
  console.log('Wooooooooo!');
}
```



# Наследование



# Наследование



# Создаем цепочку прототипов

```
function Owl(name) {
  this.name = name;
}

function Bird(name) {
  this.name = name;
}

Bird.prototype.fly = function() {
  console.log('Wooooooooo!');
};

Owl.prototype = Object.create(Bird.prototype);

console.log(Owl.prototype);
```

# Object.create

```
function Owl(name) {
  this.name = name;
}

function Bird(name) {
  this.name = name;
}

Bird.prototype.fly = function() {
  console.log('Wooooooooo!');
};

Owl.prototype = Object.create(Bird.prototype);

console.log(Owl.prototype);
```

```
const a = Object.create({x: 1});
Object.keys(a); // []
a.x; // 1
```

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/create

# Создаем цепочку прототипов

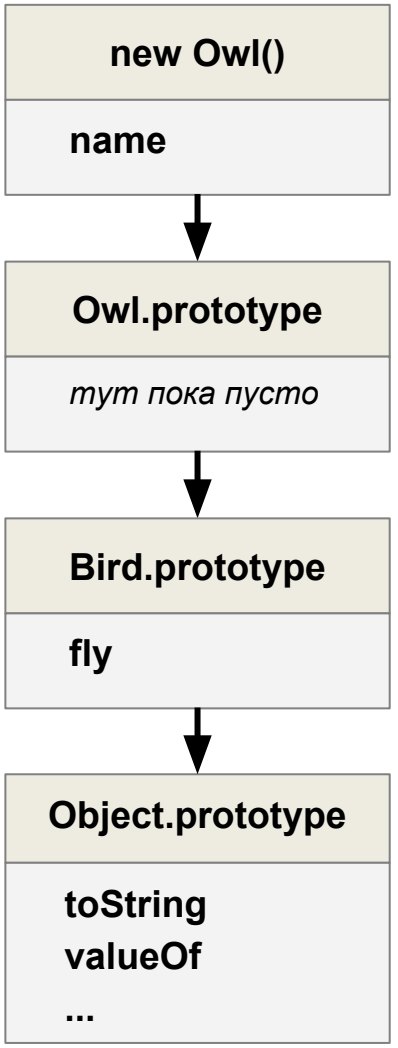
```
function Owl(name) {
  this.name = name;
}

function Bird(name) {
  this.name = name;
}

Bird.prototype.fly = function() {
  console.log('Wooooooooo!');
};

Owl.prototype = Object.create(Bird.prototype);

console.log(Owl.prototype);
```

```
▼ Bird i
  ▼ __proto__: Object
    ▶ constructor: Bird(name)
    ▶ fly: ()
    ▶ __proto__: Object
```

# Создаем цепочку прототипов

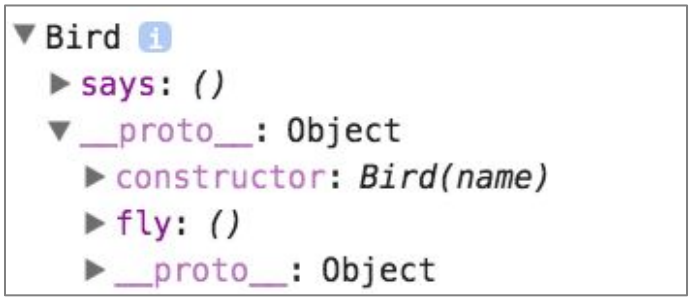
```
function Owl(name) {
  this.name = name;
}

function Bird(name) {
  this.name = name;
}

Bird.prototype.fly = function() {
  console.log('Wooooooooo!');
};

Owl.prototype = Object.create(Bird.prototype);

console.log(Owl.prototype);
```

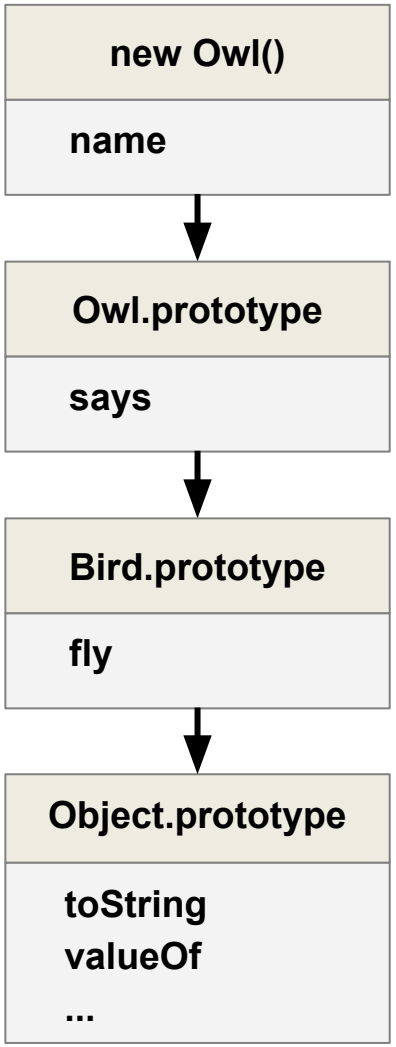
```
▼ Bird
  ▼ __proto__: Object
    ▶ constructor: Bird(name)
    ▶ fly: ()
    ▶ __proto__: Object
```



# Добавляем методы в прототип

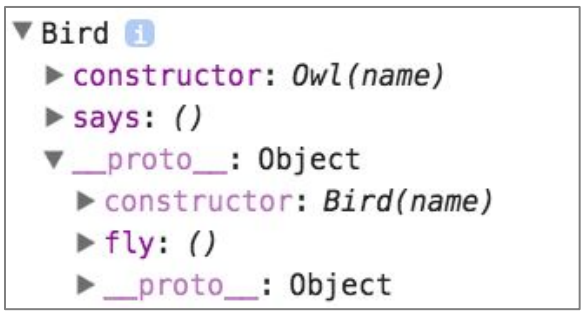
```
function Owl(name) {
  this.name = name;
}

function Bird(name) {
  this.name = name;
}

Bird.prototype.fly = function() {
  console.log('Wooooooooo!');
};

Owl.prototype = Object.create(Bird.prototype);

Owl.prototype.says = function() {
  console.log(this.name + ' говорит: ух!');
};
```

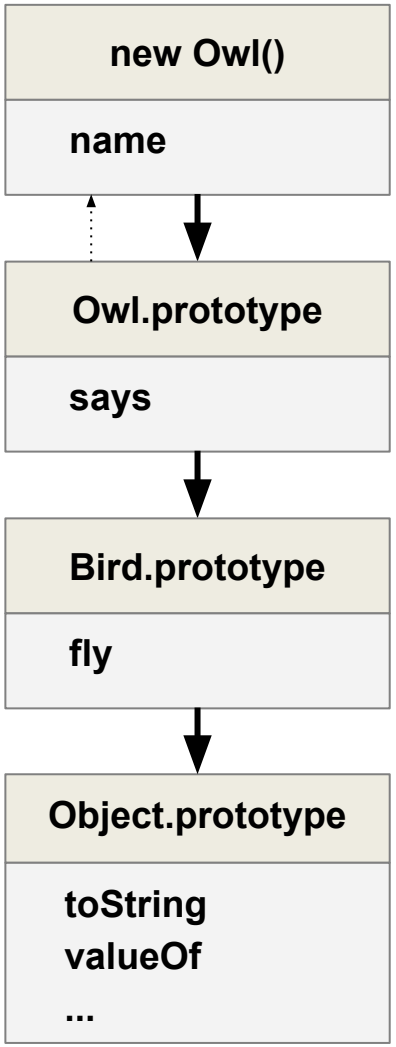


# Восстановим ссылку на конструктор

```
function Owl(name) {
  this.name = name;
}

function Bird(name) {
  this.name = name;
}

Bird.prototype.fly = function() {
  console.log('Wooooooooo!');
};

Owl.prototype = Object.create(Bird.prototype);
Owl.prototype.constructor = Owl;

Owl.prototype.says = function() {
  console.log(this.name + ' говорит: ух!');
};
```



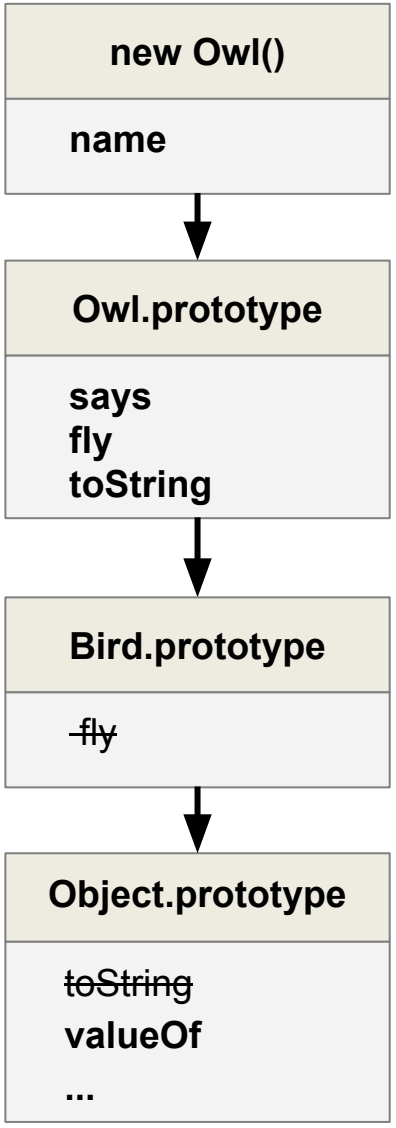
# Вызовем конструктор родителя

```
function Bird(name) {
  this.name = name;
}

function Owl(name) {
  this.name = name;
}
```

```
function Bird(name) {
  this.name = name;
}

function Owl(name) {
  Bird.apply(this, arguments);
}
```

# Переопределение методов

```
Owl.prototype.fly = function() {
  console.log(this.name + ' летит!');
};

Owl.prototype.toString = function() {
  return `сова по имени «${this.name}».`
};

const owl = new Owl('Маленький сов');

owl.fly(); // Маленький сов летит!

console.log('Это ' + owl);
// Это сова по имени «Маленький сов».
```



# ECMAScript 6

```
function Bird(name) {
  this.name = name;
}

Bird.prototype.fly = function() {
  console.log('Wooooooooo!');
};

function Owl(name) {
  Bird.apply(this, [name]);
}

Owl.prototype =
Object.create(Bird.prototype);
Owl.prototype.constructor = Owl;

Owl.prototype.says = function() {
  console.log(this.name + ' говорит: ух!');
};

const owl = new Owl('Маленький сов');
```

ES6

```
class Bird {
  constructor(name) {
    this.name = name;
  }

  fly() {
    console.log('Wooooooooo!');
  }
}

class Owl extends Bird {
  constructor(name) {
    super(name);
  }

  says() {
    console.log(this.name + ' говорит: ух!');
  }
}

const owl = new Owl('Маленький сов');
```

# ES 6: class

ES6

```
function Bird(name) {
  this.name = name;
}

Bird.prototype.fly = function() {
  console.log('Wooooooooo!');
};

function Owl(name) {
  Bird.apply(this, [name]);
}

Owl.prototype =
Object.create(Bird.prototype);
Owl.prototype.constructor = Owl;

Owl.prototype.says = function() {
  console.log(this.name + ' говорит: ух!');
};

const owl = new Owl('Маленький сов');
```

```
class Bird {
  constructor(name) {
    this.name = name;
  }

  fly() {
    console.log('Wooooooooo!');
  }
}

class Owl extends Bird {
  constructor(name) {
    super(name);
  }

  says() {
    console.log(this.name + ' говорит: ух!');
  }
}

const owl = new Owl('Маленький сов');
```

# Геттеры и сеттеры

```
class Bird {
  constructor(name, species) {
    this.name = name;
    this.species = species;
  }

  get nameWithSpecies() {
    return `Это ${this.name}. Он ${this.species}.`
  }

  set nameWithSpecies(newValue) {
    [this.species, this.name] = newValue.split(' ');
  }
}

const bird = new Bird('Таня', 'цапля');
console.log(bird.nameWithSpecies); // Это Таня. Она цапля.
bird.nameWithSpecies = 'дятел Миша';
console.log(bird.name, bird.species); // Миша дятел
```

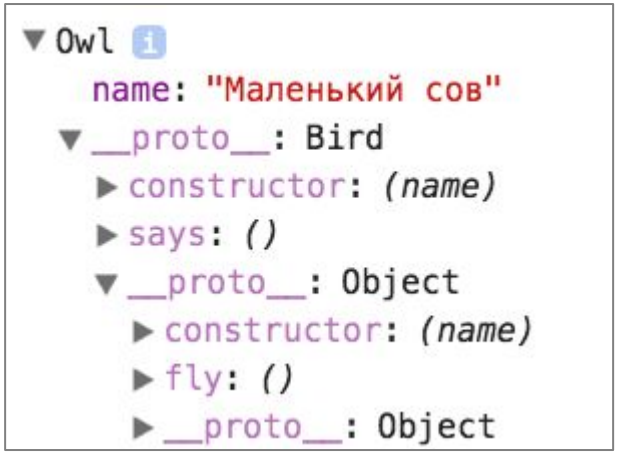
https://learn.javascript.ru/es-class

# Статические свойства

```
class Bird {
  fly() {
    console.log('Wooooooooo! | last was: ' + Bird.getFormattedTime(new Date()));
  }

  static getFormattedTime(date) {
    return date;
  }
}

const bird = new Bird();
bird.fly(); // Wooooooooo! | last was: Fri Mar 24 2017 ...
```

# hasOwnProperty и Object.keys

```
const owl = new Owl('Маленький сов');

for (let key in owl) {
  if (owl.hasOwnProperty(key)){
    console.log(key);
  }
}

Object.keys(owl); // ['name']
```



# instanceof

```
const owl = new Owl('Маленький сов');

console.log(owl instanceof Owl); // true
console.log(owl instanceof Bird); // true
```

http://learn.javascript.ru/instanceof

# instanceof

```
function Owl() {}
const owl = new Owl();
console.log(owl instanceof Owl); // true
Owl.prototype = {};
console.log(owl instanceof Owl); // false


new String('foo') instanceof String; // true
new String('foo') instanceof Object; // true

'foo' instanceof String; // false
'foo' instanceof Object; // false
```

http://shamansir.github.io/JavaScript-Garden/#types.instanceof

# Временные контейнеры для примитивов

```
const owl = 'Большой сов';
console.log(owl.toUpperCase()); // БОЛЬШОЙ СОВ

owl.vzhuh = 'Вжух';
console.log(owl.vzhuh); // undefined
```

http://learn.javascript.ru/native-prototypes#примитивы

# Вопросы?