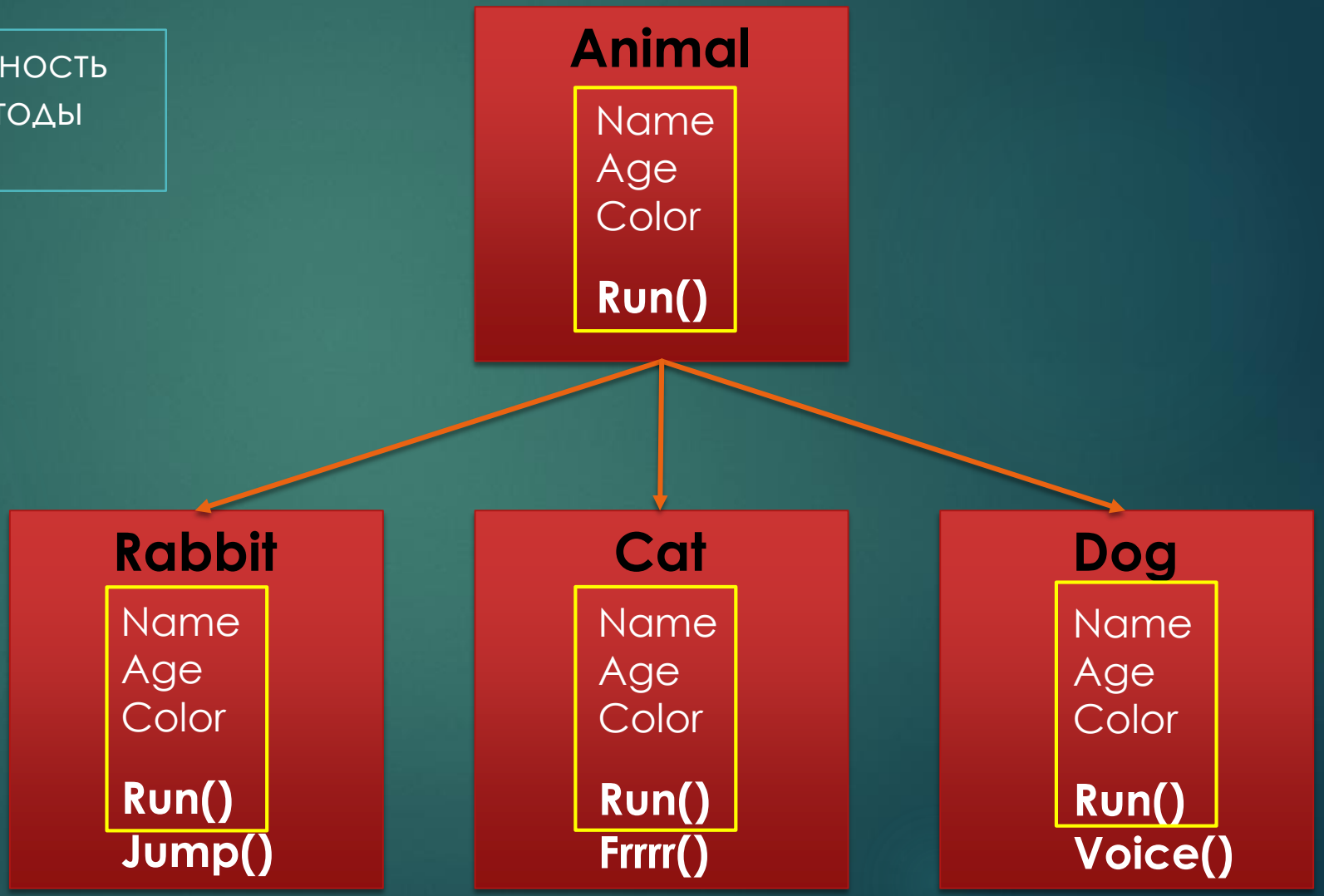
Объекты в JavaScript

НАСЛЕДОВАНИЕ В ФУНКЦИОНАЛЬНОМ СТИЛЕ

Наследование

Наследование – это возможность использовать свойства и методы одного объекта, другим



Наследование. Call and Apply

В JS есть возможность явного указания контекста выполнения.

```
functionName . call ( this , arg1, arg2 ,arg3 ... )
```

```
var number = {
      num : 4;
};

function plus(a){
      return this.num + a;
}
```

```
plus . call ( number ,  8 )
```

→ 12

```
functionName . apply ( this , [ arg1, arg2 ] )
```

Наследование. Функциональный подход

```
function Animal (data){
    this.name = data.name;
    this.age = data.age;
    this.color = data.color;
    this.run = function (m){
        console.log('the ' + this.name + ' run ' + m + 'meters');
    }
}
```

```
function Rabbit (data){
    Animal.call(this, data);

    this.jump = function(){
        // some code here
    }
}
```

```
function Cat (data){
    Animal.call(this, data);

    this.frrrr = function(){
        // some code here
    }
}
```

```
function Dog (data){
    Animal.call(this, data);

    this.voice = function(){
        // some code here
    }
}
```

Наследование

Наследник не имеет доступа к приватным свойствам родителя!

Чтобы наследник имел доступ к свойству, оно должно быть записано в **this**

```
function Animal (data){
    this._weight = data.weight;
}
```

```
function Cat (data){
    Animal.call(this, data);
    this.getWeight = function(){
        return this._weight;
    }
}
```

Наследование

Свойства и методы родителя могут быть переопределены у наследника

```
function Animal (data){
    this.name = data.name;
    this.age = data.age;
    this.color = data.color;
    this.run = function (m){
        //some code…
    }
}
```



```
function Dog (data){
    Animal.call(this, data);

    this.color = data.color;
    this.run = function (m){
        //some code…
    }
}
```

Наследование

Однако, как правило, мы хотим не заменить, а расширить метод родителя добавив к нему что-то.
Для этого метод родителя предварительно копируют в переменную, и затем вызывают внутри нового метода – там, где считают нужным

Общая схема расширения метода

1. Создаем локальную переменную и копируем в нее метод родителя
2. Заменяем метод родителя на свой
3. Вызываем внутри нашего метода родительский, передавая в него контекст

```
function Dog (data){
    var parentRun = this.run;
    this.run = function (m){
        this.voice();
        parentRun.call(this);
    }
}
```

Объекты в JavaScript

ПРОТОТИПЫ

Прототипы

Объекты в JavaScript можно организовать в цепочки так, чтобы свойство, не найденное в одном объекте, автоматически искалось бы в другом.

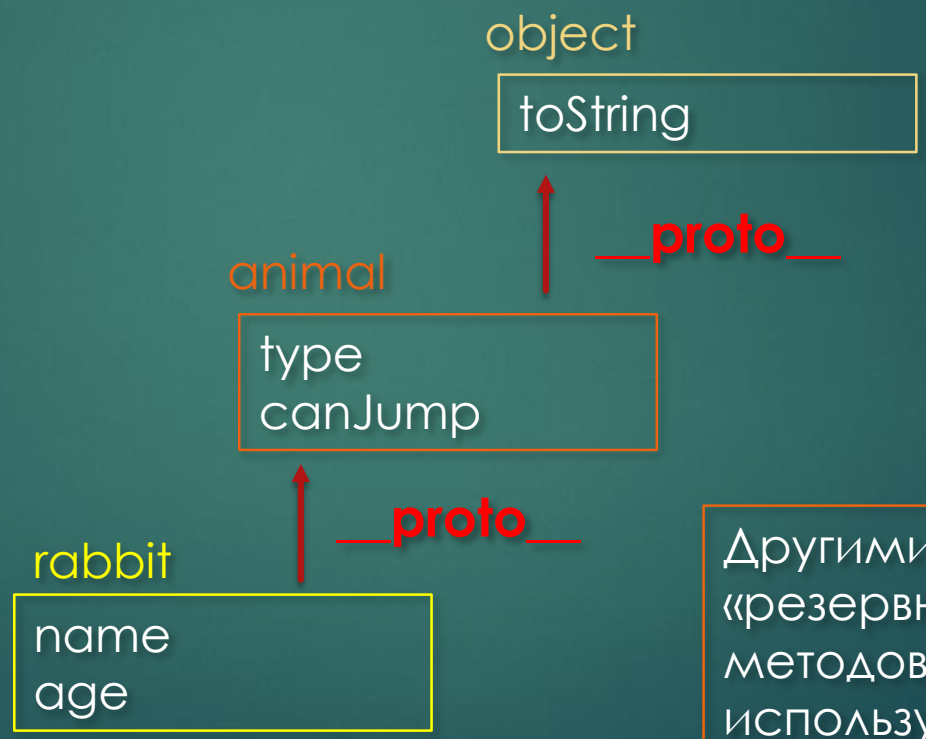
Прототипы

Если один объект имеет специальную ссылку __proto__ на другой объект, то при чтении свойства из него, если свойство отсутствует в самом объекте, оно ищется в объекте __proto__

```
var animal = {
    type : “animal”,
    canJump : true
};
var rabbit = {
    name = “Roger”,
    age = 5
};

rabbit.__proto__ = animal;

alert(rabbit.type);
alert(rabbit.tostring());
```



Другими словами, прототип – это «резервное хранилище свойств и методов» объекта, автоматически используемое при поиске.

Прототипы

Для оператора **in** нет разницы, где найдено свойство в самом объекте или его прототипе.

Метод **hasOwnProperty(**prop**)** предназначен для проверки, что свойство принадлежит имеено объекту, а не его прототипу

```
var animal = {
      type : “animal”,
      canJump : true
};
var rabbit = {
      name = “Roger”,
      age = 5
};

rabbit.__proto__ = animal;
```

```
for(var prop in rabbit){
      if(rabbit.hasOwnProperty(prop)){
            console.log(‘rabbit has property ‘ + prop);
      }
}
```

Прототипы и функция конструктор

```
var animal = {
    type : “animal”,
    canJump : true,
    run : function(m){
        //some code
    }
};
```

```
function Rabbit (data){
    this.name = data.name;
    this.age = data.age;

    this.jump = function(){
        //some code
    }
}

Rabbit.prototype = animal;
```

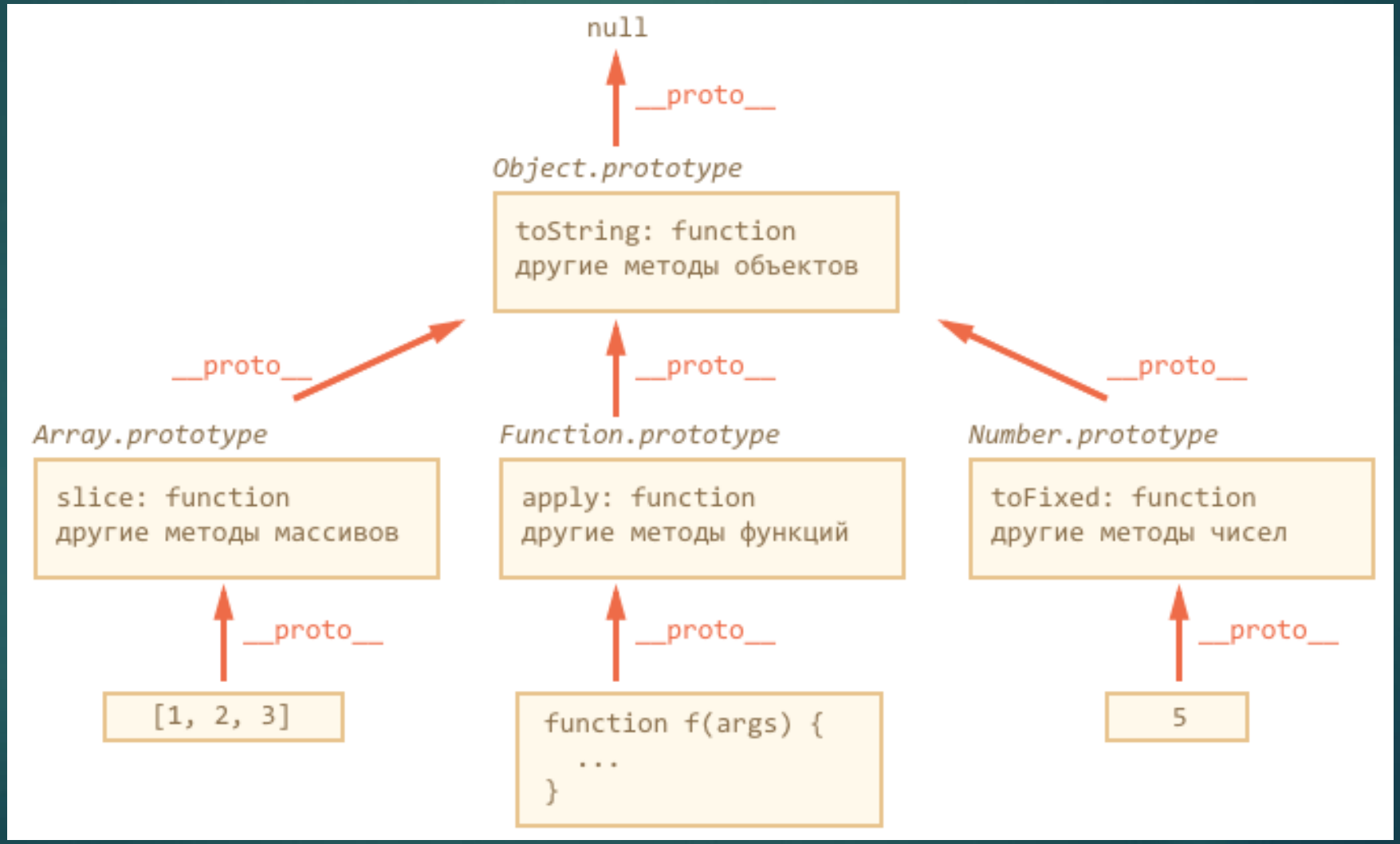
```
var r1 = new Rabbit({
    name : ‘Roger’,
    age : 4
});

alert(r1.canJump);
alert(r1.run(5));
alert(r1.name);
alert(r1.jump());
```

Установка Rabbit.**prototype** = animal буквально говорит интерпретатору следующее: "При создании объекта через new Rabbit запиши ему **__proto__** = animal"

! Свойство **prototype** имеет смысл только у **конструктора** !

Прототипы


null
__proto__
Object.prototype
toString: function
другие методы объектов
__proto__
__proto__
__proto__
Array.prototype
slice: function
другие методы массивов
Function.prototype
apply: function
другие методы функций
Number.prototype
toFixed: function
другие методы чисел
__proto__
__proto__
__proto__
[1, 2, 3]
function f(args) {
...
}
5