OOP with JavaScript

Outline

- OOP in JavaScript
- Implementing Prototypal Inheritance
- Prototype: __proto__ / [[Prototype]] and Property
- Prototypal Inheritance / Delegation
- Prototype Chain

OOP in JavaScript

"CLASSICAL OOP": CLASSES

Class

INSTANTIATION

Instance

- Objects (instances) are **instantiated** from a class, which functions like a blueprint;
- Behavior (methods) is **copied** from class to all instances.

OOP IN JS: PROTOTYPES

Prototype

Contains methods

PROTOTYPAL INHERITANCE/ DELEGATION

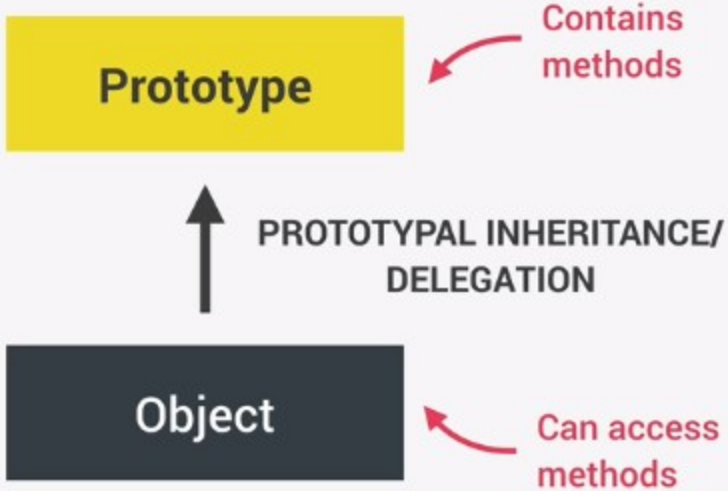
Object

Can access methods

- Objects are **linked** to a prototype object;
- **Prototypal inheritance:** The prototype contains methods (behavior) that are **accessible to all objects linked to that prototype;**
- Behavior is **delegated** to the linked prototype object.

OOP in JavaScript

OOP IN JS: PROTOTYPES



- Objects are **linked** to a prototype object;
- **Prototypal inheritance:** The prototype contains methods (behavior) that are **accessible to all objects linked to that prototype;**
- Behavior is **delegated** to the linked prototype object.

Example: Array

```
const num = [1, 2, 3];
num.map(v => v * 2);
```

MDN web docs
moz://a

```
Array.prototype.keys()
Array.prototype.lastIndexOf()
Array.prototype.map()
```

`Array.prototype` is the **prototype** of all array objects we create in JavaScript

Therefore, all arrays have access to the `map` method!

```
▼ ƒ Array()
    arguments: (...)
    caller: (...)
    length: 1
    name: "Array"
  ▼ prototype: Array(0)
    ▶ unique: ƒ ()
      length: 0
    ▶ constructor: ƒ Array()
    ▶ concat: ƒ concat()
    ▶ map: ƒ map()
```

Implementing Prototypal Inheritance

1 **Constructor functions**

- 👉 Technique to create objects from a function;
- 👉 This is how built-in objects like Arrays, Maps or Sets are actually implemented.

2 **ES6 Classes**

- 👉 Modern alternative to constructor function syntax;
- 👉 "Syntactic sugar": behind the scenes, ES6 classes work **exactly** like constructor functions;
- 👉 ES6 classes do **NOT** behave like classes in "classical OOP"

3 **`Object.create()`**

- 👉 The easiest and most straightforward way of linking an object to a prototype object.

Prototype: [[Prototype]] / __proto__

```
function Person(firstName, birthYear) {
    // Instance properties
    this.firstName = firstName;
    this.birthYear = birthYear;
}

var person1 = new Person("Sachin", 1973);
console.log(person1);

var person2 = new Person("Rohit", 1987);
console.log(person2);
```

Output:

```
▼ Person
    birthYear: 1973
    firstName: "Sachin"
  ▶ [[Prototype]]: Object

▼ Person
    birthYear: 1987
    firstName: "Rohit"
  ▶ [[Prototype]]: Object
```

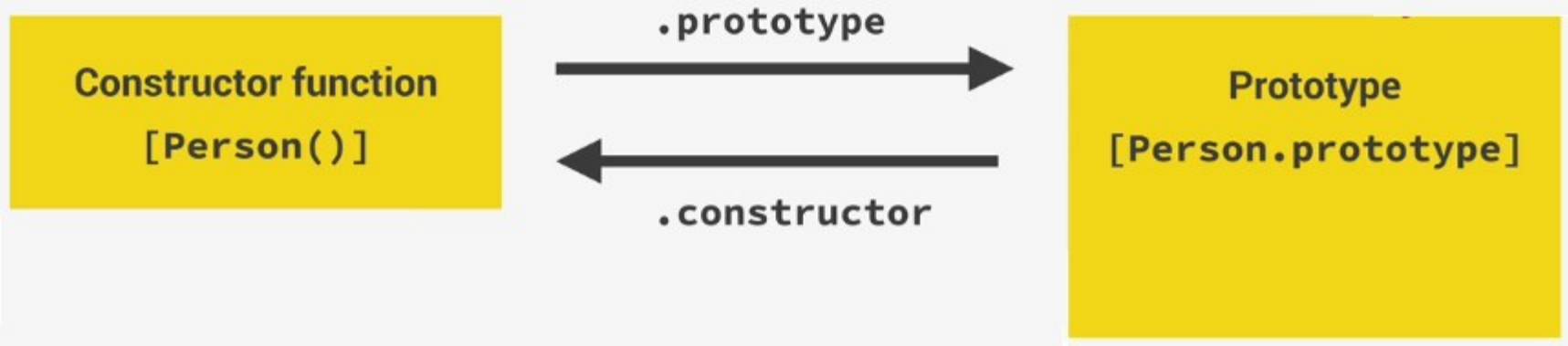
Prototype: [[Prototype]] and Property

```
"use strict";

function Person(firstName, birthYear) {
    // Instance properties
    this.firstName = firstName;
    this.birthYear = birthYear;
}

var person1 = new Person("Sachin", 1973);
console.log(person1);

console.log(Person.prototype);

console.log(Person.prototype === person1.__proto__);
```

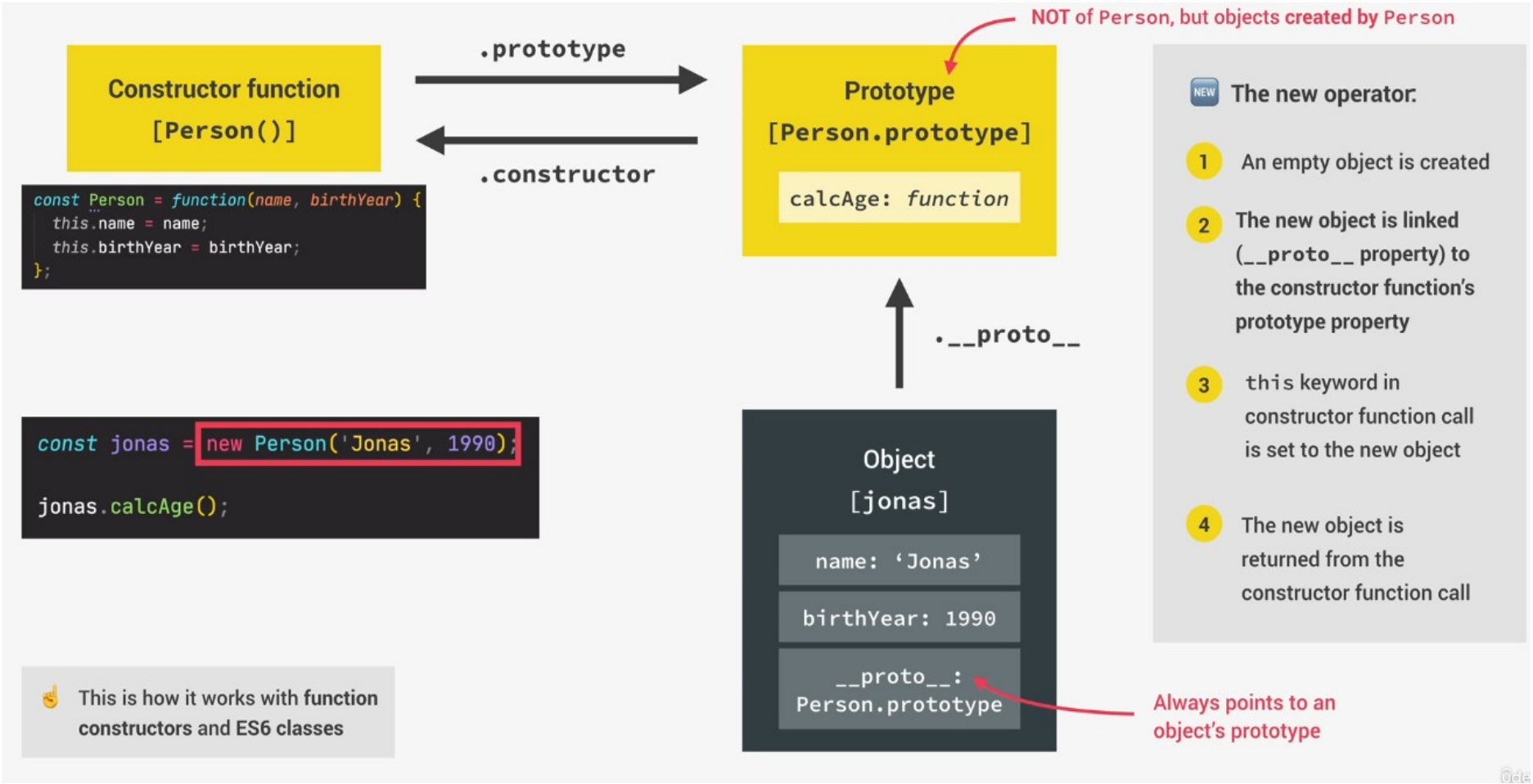
Output:

```
▼ Person {firstName: "Sachin", birthYear: 1973}
    birthYear: 1973
    firstName: "Sachin"
  ▼ [[Prototype]]: Object
    ▶ constructor: ƒ Person(firstName, birthYear)
    ▶ [[Prototype]]: Object

▼ {constructor: ƒ}
  ▶ constructor: ƒ Person(firstName, birthYear)
  ▶ [[Prototype]]: Object

true
```

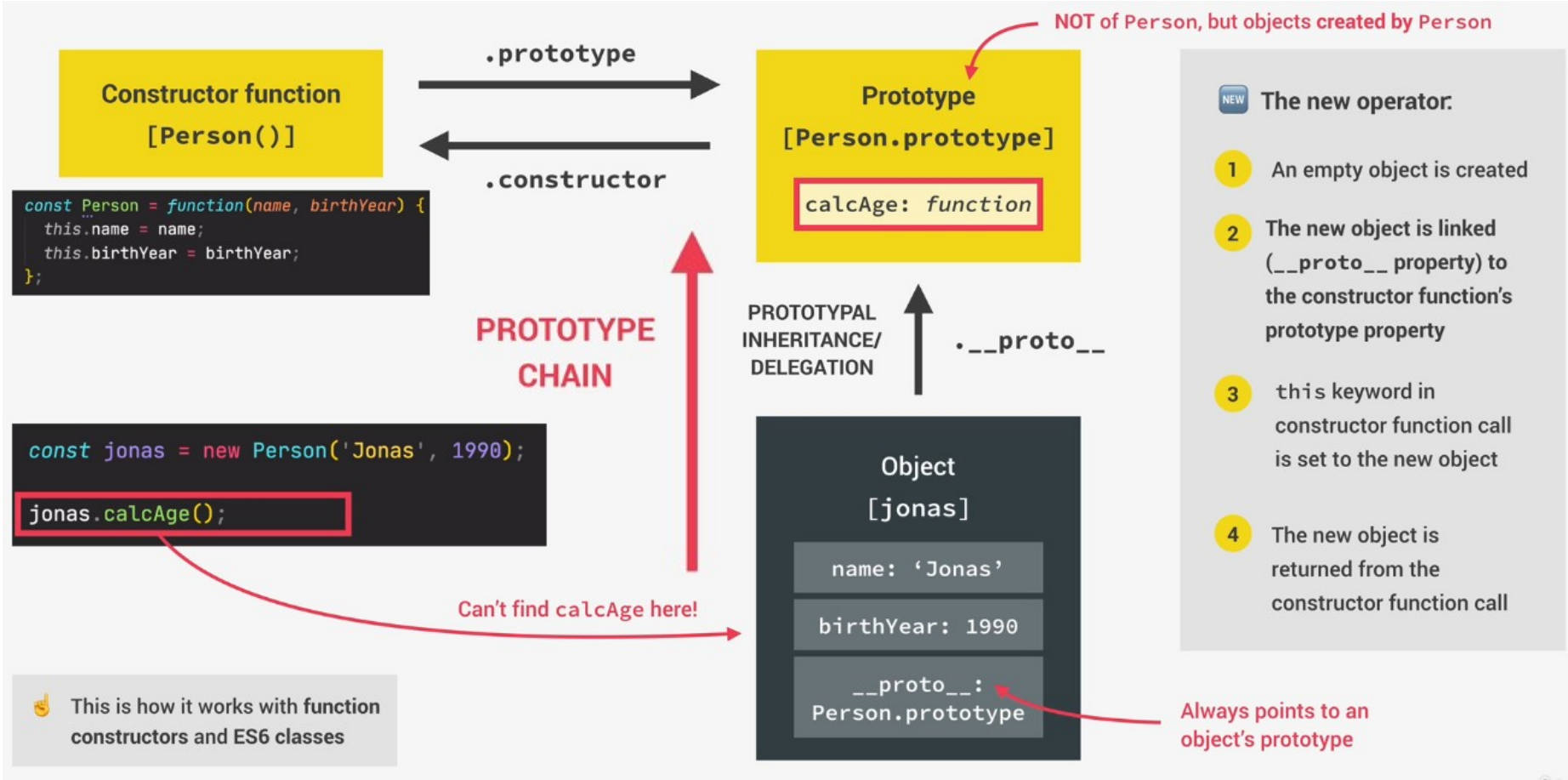
Prototypal Inheritance / Delegation



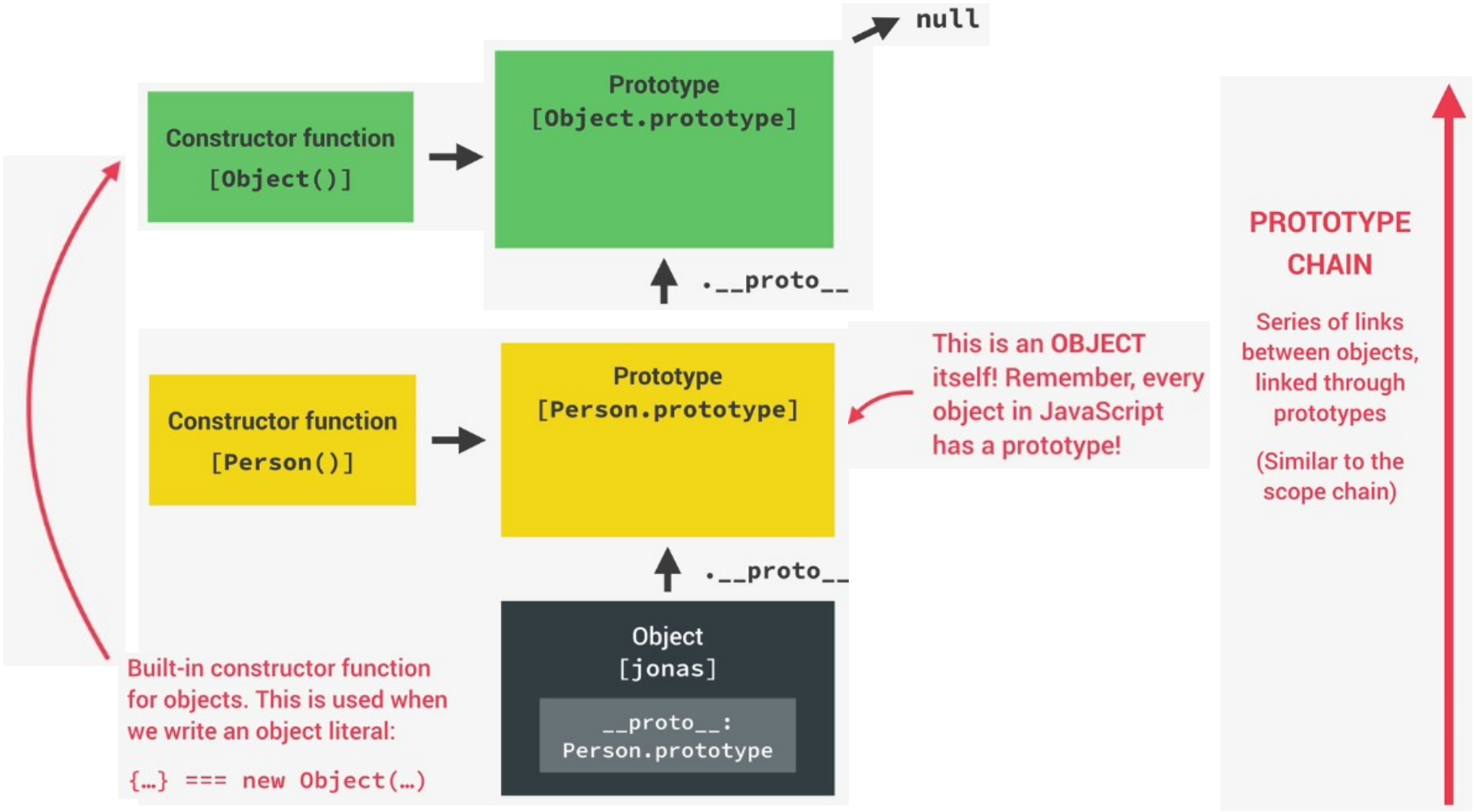
Prototypal Inheritance / Delegation



Prototypal Inheritance / Delegation



Prototye Chain



References

- https://www.udemy.com/course/the-complete-javascript-course