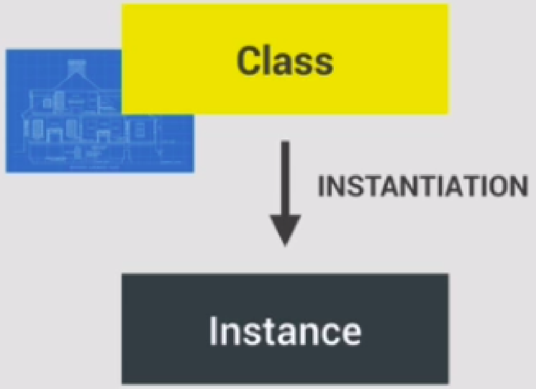
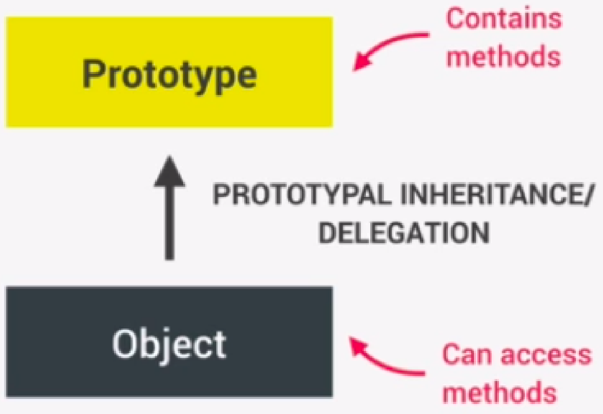
OOP IN JAVASCRIPT: PROTOTYPES

"CLASSICAL OOP": CLASSES



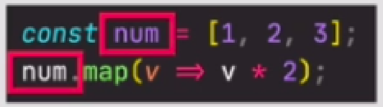
- Objects (instances) are **instantiated** from a class, which functions like a blueprint;
- Behavior (methods) is **copied** from class to all instances.

OOP IN JS: PROTOTYPES

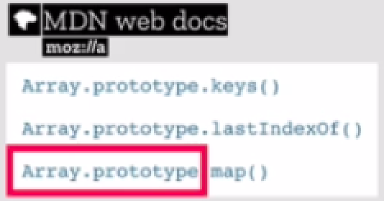


- Objects are **linked** to a prototype object;
- **Prototypal inheritance:** The prototype contains methods (behavior) that are **accessible to all objects linked to that prototype;**
- Behavior is **delegated** to the linked prototype object.

Example: Array



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

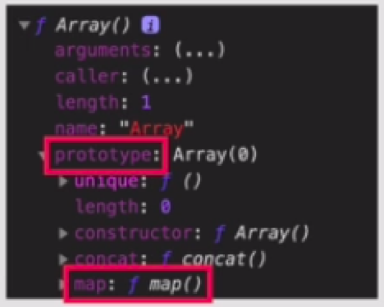


MDN web docs
moz://a

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

3 WAYS OF IMPLEMENTING PROTOTYPAL INHERITANCE IN JAVASCRIPT

🤔 *"How do we actually create prototypes? And how do we link objects to prototypes? How can we create new objects, without having classes?"*

☝️ **The 4 pillars of OOP are still valid!**

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

1 Constructor functions

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

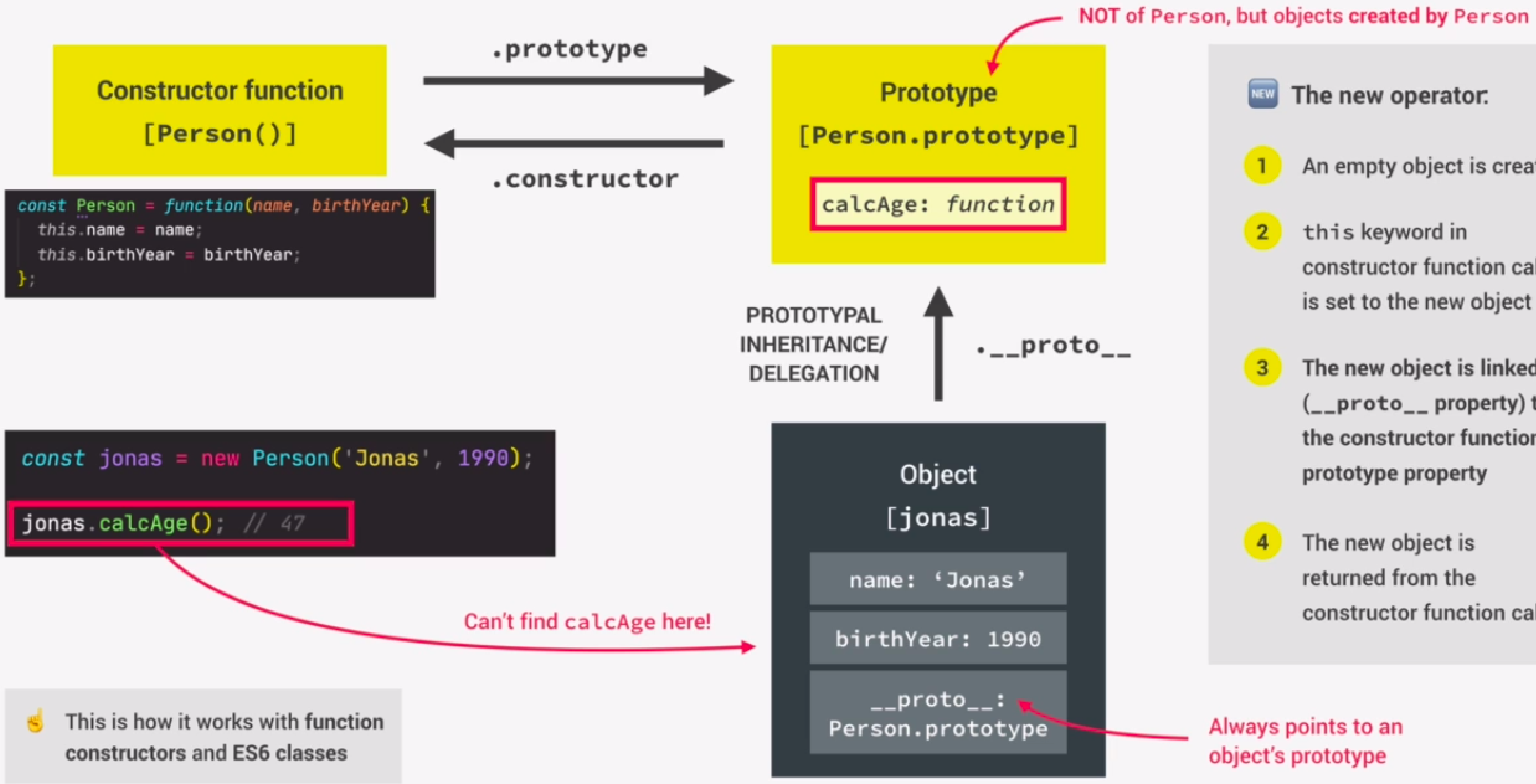
2 ES6 Classes

- 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" (last lecture).

3 `Object.create()`

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

HOW PROTOTYPAL INHERITANCE / DELEGATION WORKS



```
const Person = function(name, birthYear) {
  this.name = name;
  this.birthYear = birthYear;
};
```

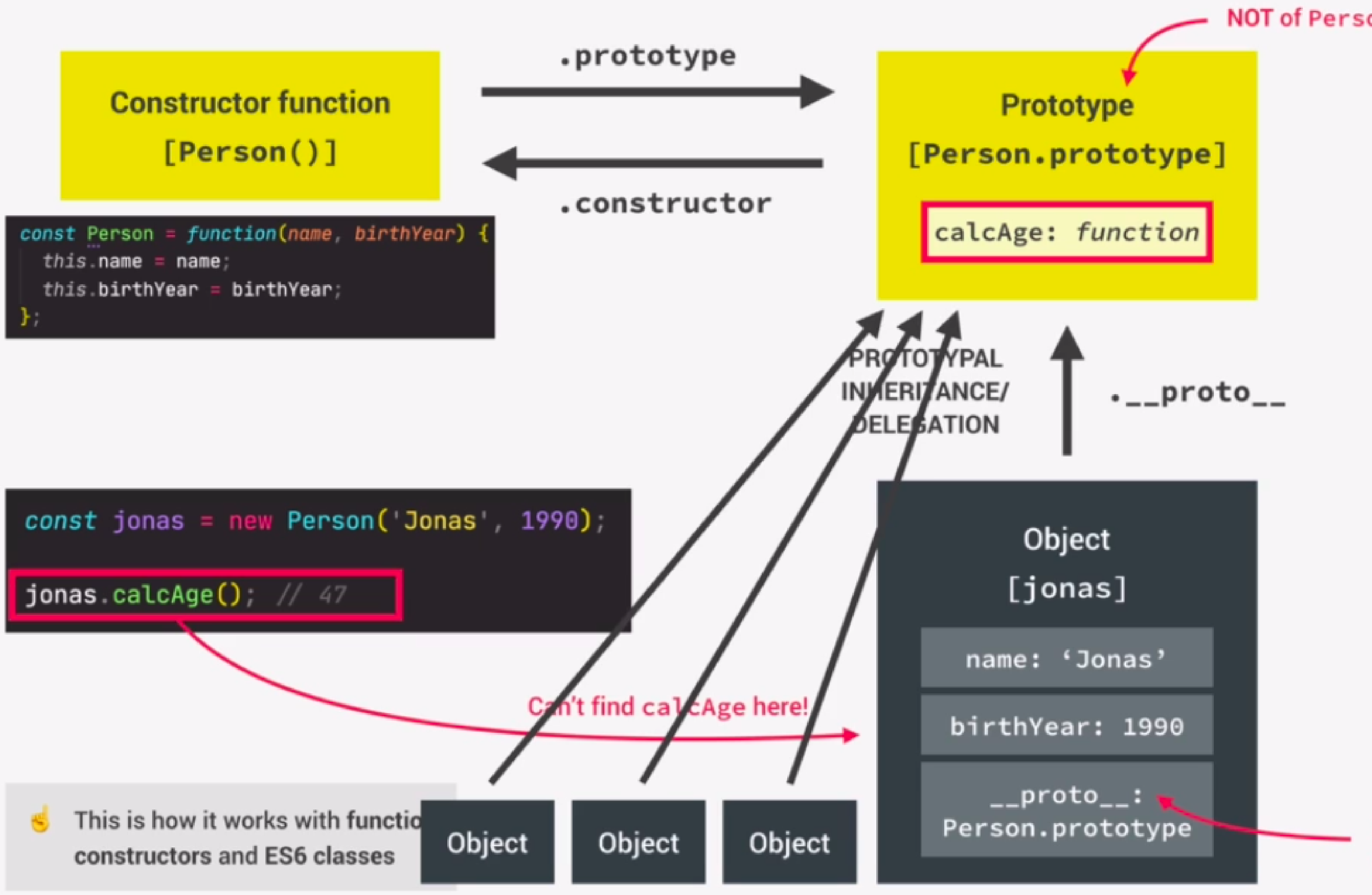
```
const jonas = new Person('Jonas', 1990);

jonas.calcAge(); // 47
```

☝ This is how it works with function constructors and ES6 classes

🆕 The new operator:

1. An empty object is created
2. this keyword in constructor function call is set to the new object
3. The new object is linked (__proto__ property) to the constructor function's prototype property
4. The new object is returned from the constructor function call

HOW PROTOTYPAL INHERITANCE / DELEGATION WORKS



```
const Person = function(name, birthYear) {
  this.name = name;
  this.birthYear = birthYear;
};
```

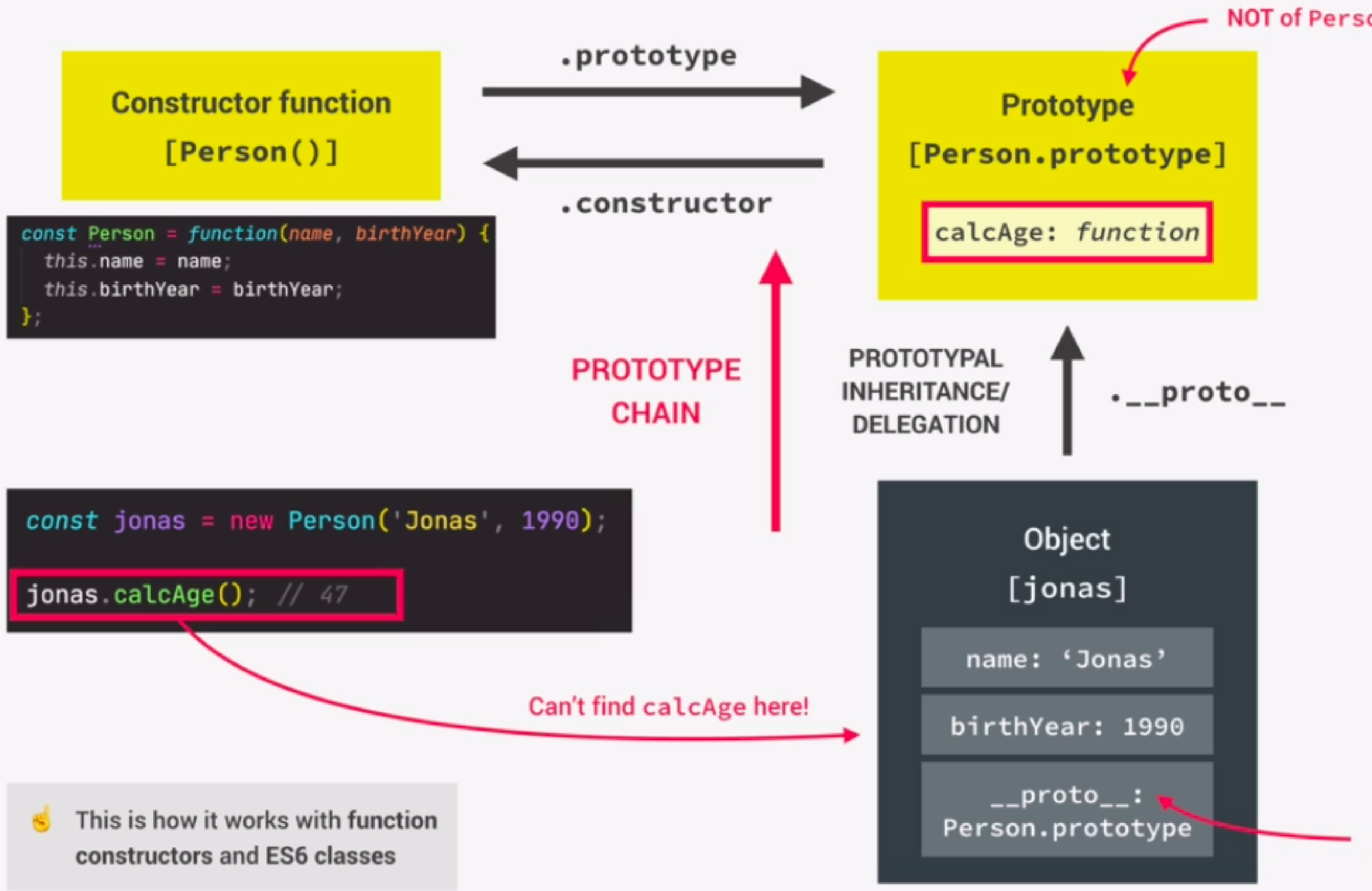
```
const jonas = new Person('Jonas', 1990);

jonas.calcAge(); // 47
```

☝ This is how it works with function constructors and ES6 classes

The new operator:

1. An empty object is created
2. this keyword in constructor function call is set to the new object
3. The new object is linked (__proto__ property) to the constructor function's prototype property
4. The new object is returned from the constructor function call

HOW PROTOTYPAL INHERITANCE / DELEGATION WORKS



```js
const Person = function(name, birthYear) {
  this.name = name;
  this.birthYear = birthYear;
};
```

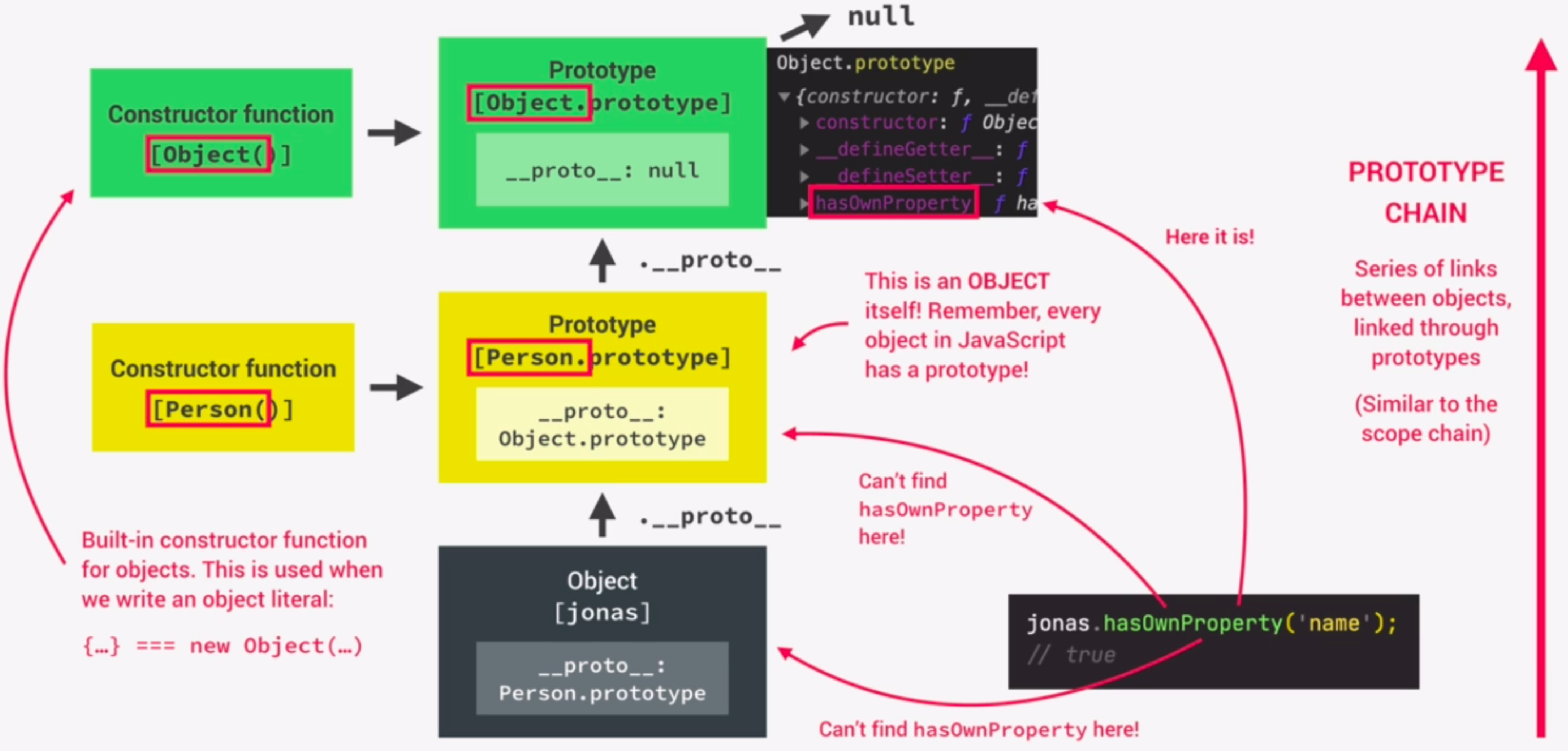
```js
const jonas = new Person('Jonas', 1990);

jonas.calcAge(); // 47
```

☝ This is how it works with function constructors and ES6 classes

🆕 The new operator:

1. An empty object is created
2. this keyword in constructor function call is set to the new object
3. The new object is linked (__proto__ property) to the constructor function's prototype property
4. The new object is returned from the constructor function call

THE PROTOTYPE CHAIN



```
Object.prototype
▼{constructor: f, __def
  ▶constructor: f Objec
  ▶__defineGetter__: f
  ▶__defineSetter__: f
  ▶hasOwnProperty: f ha
```

Built-in constructor function for objects. This is used when we write an object literal:

`{…} === new Object(…)`

This is an OBJECT itself! Remember, every object in JavaScript has a prototype!

Can't find `hasOwnProperty` here!

Can't find `hasOwnProperty` here!

```
jonas.hasOwnProperty('name');
// true
```

PROTOTYPE CHAIN

Series of links between objects, linked through prototypes

(Similar to the scope chain)