Overview: Object Modules

Module 3: Prototypes and Properties

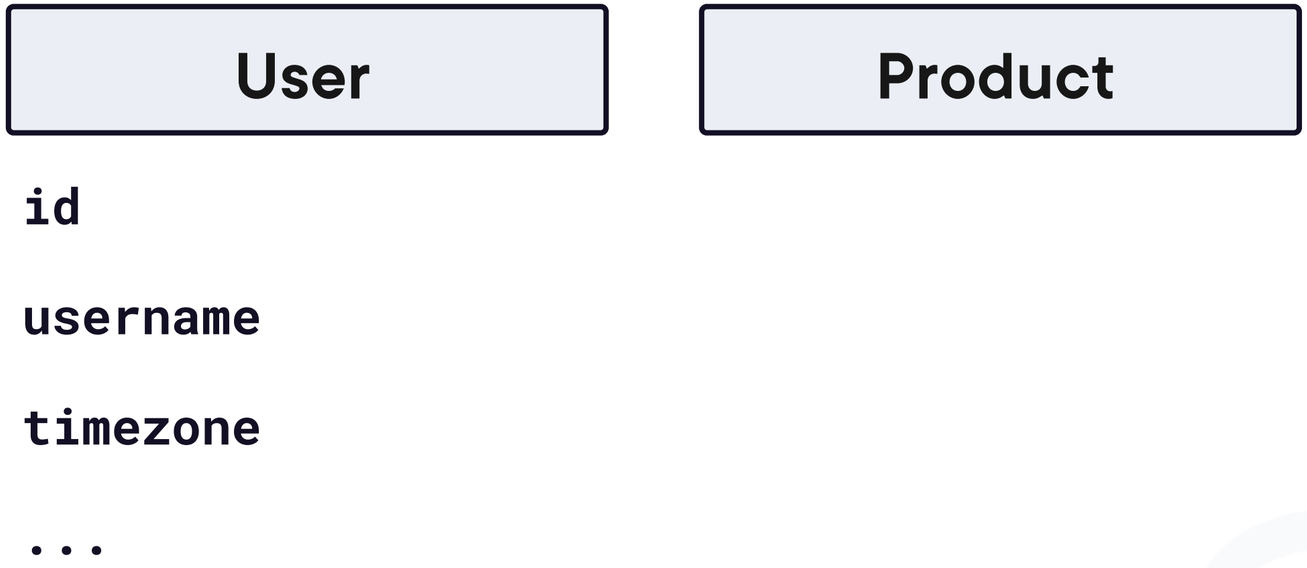
- What is an Object?
- Creating Objects
- What is a Prototype?
- Defining Prototypes
- Creating Properties
- Accessing Properties
- Deleting Properties
- Protecting Properties
- Enumerating Properties

Module 4: References and Scope

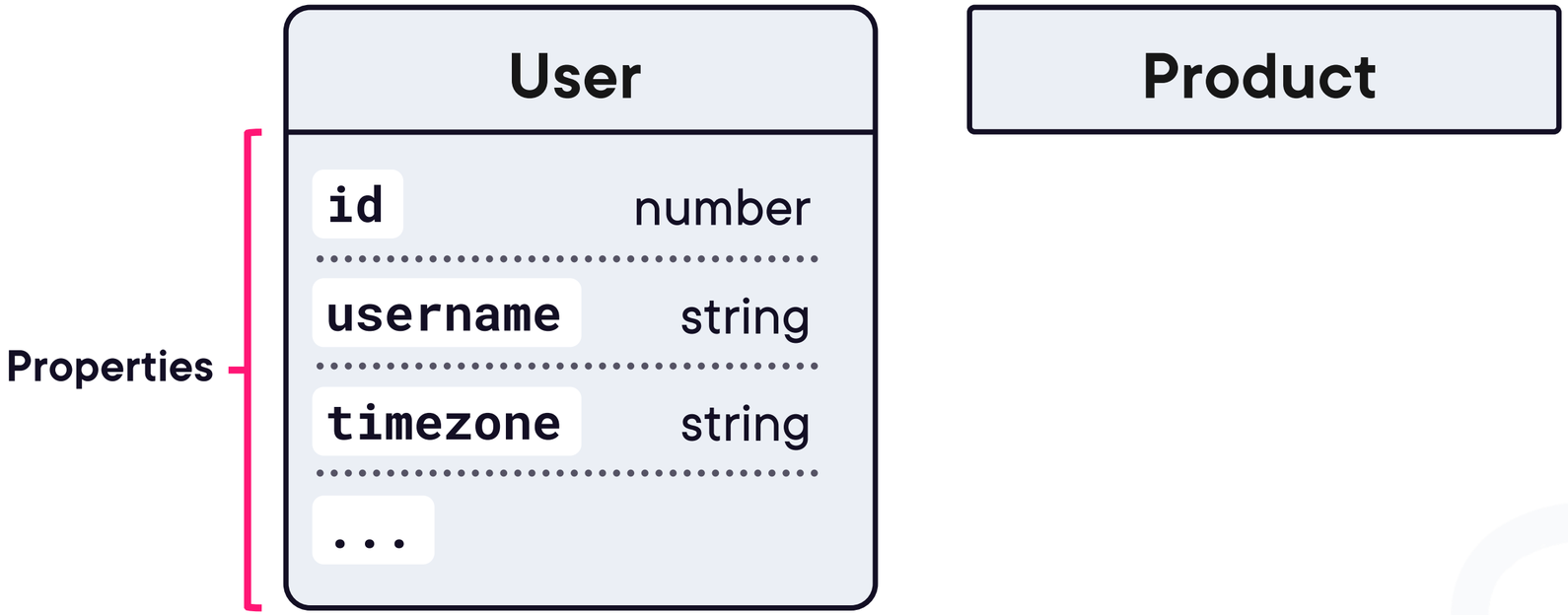
- Reference Types
- Comparing and Cloning
- Functions as Properties
- Scope in Objects
- Primitive Wrappers

Objects behave the way you expect—until they don’t

Objects are Containers for Other Datatypes



Objects are Containers for Other Datatypes



Three Facts About JavaScript Objects

Containers for other datatypes

Passed via reference

Link to a prototype

Overview: Object Modules

Module 3: Prototypes and Properties

- What is an Object?
- Creating Objects
- What is a Prototype?
- Defining Prototypes
- Creating Properties
- Accessing Properties
- Deleting Properties
- Protecting Properties
- Enumerating Properties

Module 4: References and Scope

- Reference Types
- Comparing and Cloning
- Functions as Properties
- Scope in Objects
- Primitive Wrappers

Overview: Object Modules

Module 3: Prototypes and Properties

- What is an Object?
- Creating Objects
- What is a Prototype?
- Defining Prototypes
- Creating Properties
- Accessing Properties
- Deleting Properties
- Protecting Properties
- Enumerating Properties

Module 4: References and Scope

- **Reference Types**
- **Comparing and Cloning**
- Functions as Properties
- Scope in Objects
- Primitive Wrappers

Overview: Object Modules

Module 3: Prototypes and Properties

- What is an Object?
- Creating Objects
- **What is a Prototype?**
- **Defining Prototypes**
- Creating Properties
- Accessing Properties
- Deleting Properties
- Protecting Properties
- Enumerating Properties

Module 4: References and Scope

- Reference Types
- Comparing and Cloning
- Functions as Properties
- Scope in Objects
- Primitive Wrappers

Objects are Passed Via Reference

Primitive Datatypes

```
const myBoolean = false
```

myBoolean → false

Objects

```
const myObj = new Object()
```

myObj → 0x6a99aa2f

0x3a57ff1e	...
0x5de34065	...
0x6a99aa2f	new Object
0x6b49bc4d	...

Up Next:

Creating Objects

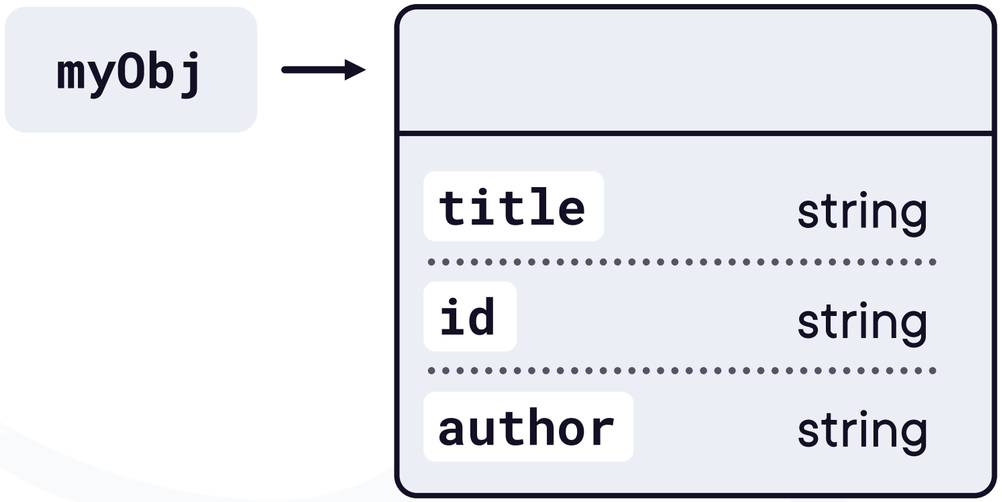
Creating Objects: Object Literal

```
const myCourse = {
   title: “Arrays and Objects in JavaScript”,
   id: “ps_js-arrobj”,
   author: “Matt Henry”
};
```

Creating Objects

Object Literal

```
const myObj = { ... };
```



Object Literals and JSON

JSON: JavaScript Object Notation

Object Literal

```
const myCourse = {
  title: "Arrays and Objects",
  id: "ps_js-arrobj",
  author: "Matt Henry",
};
```

JSON Document

```
{
  "title": "Arrays and Objects",
  "id": "ps_js-arrobj",
  "author": "Matt Henry"
}
```

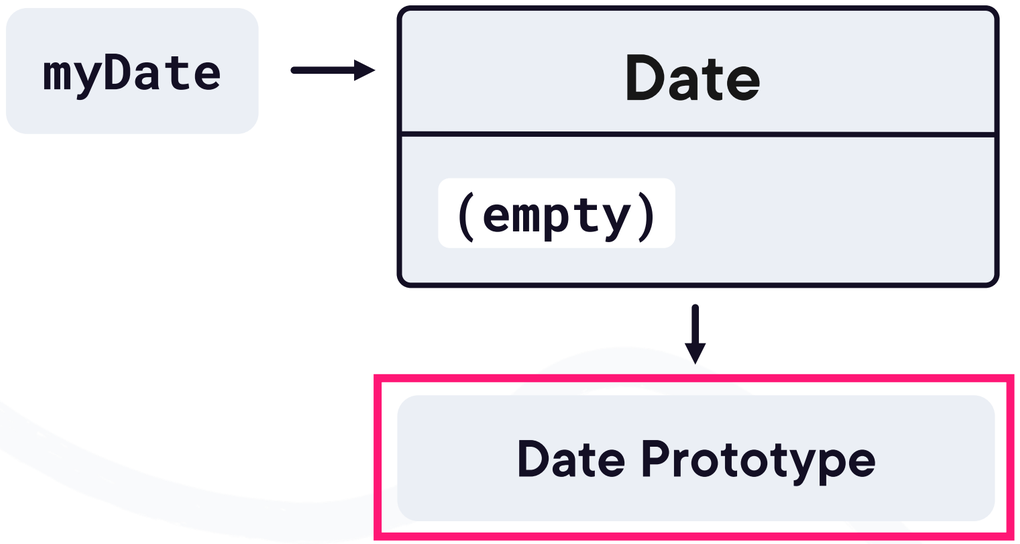
Creating Objects: new Syntax

```
const myDate = new Date();
myDate.getDate(); // 2
```

Creating Objects

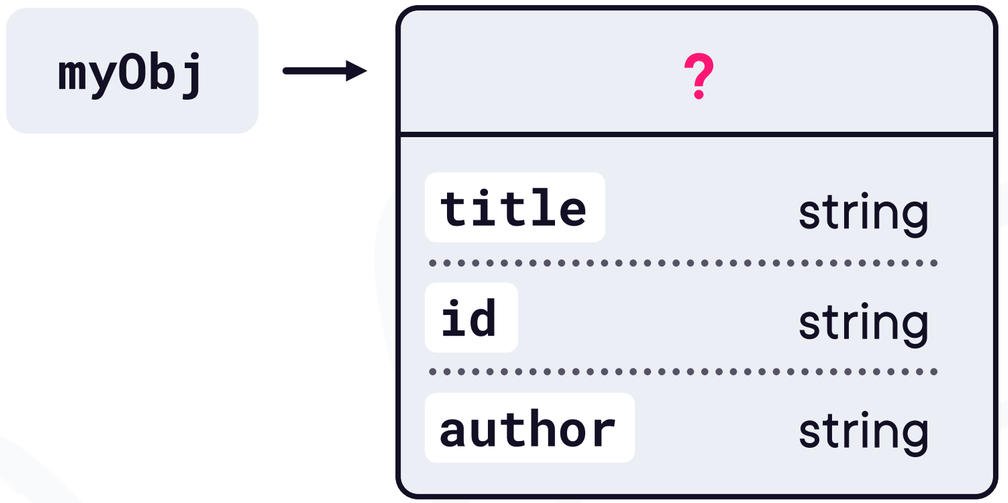
New Syntax

```
const myDate = new Date();
```



Object Literal

```
const myObj = { ... };
```

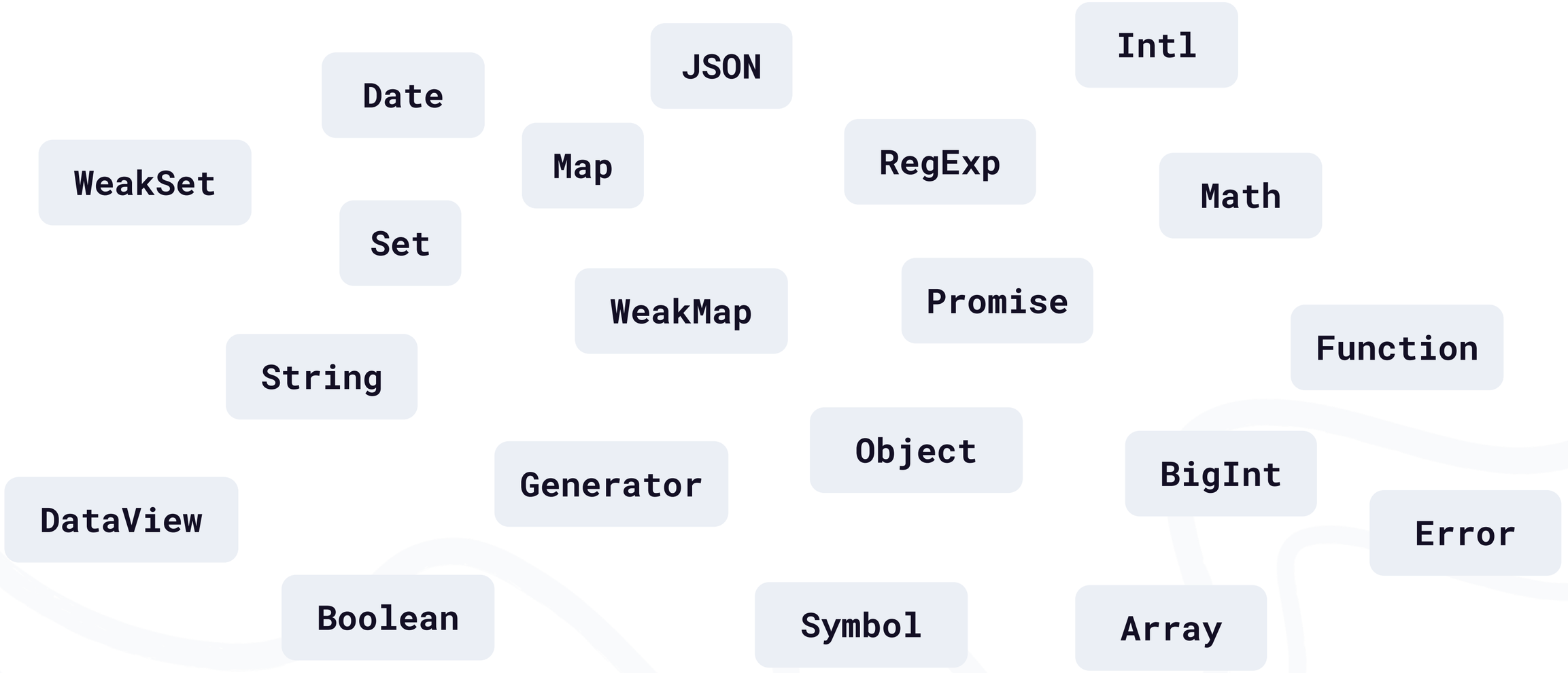


All objects link to a prototype

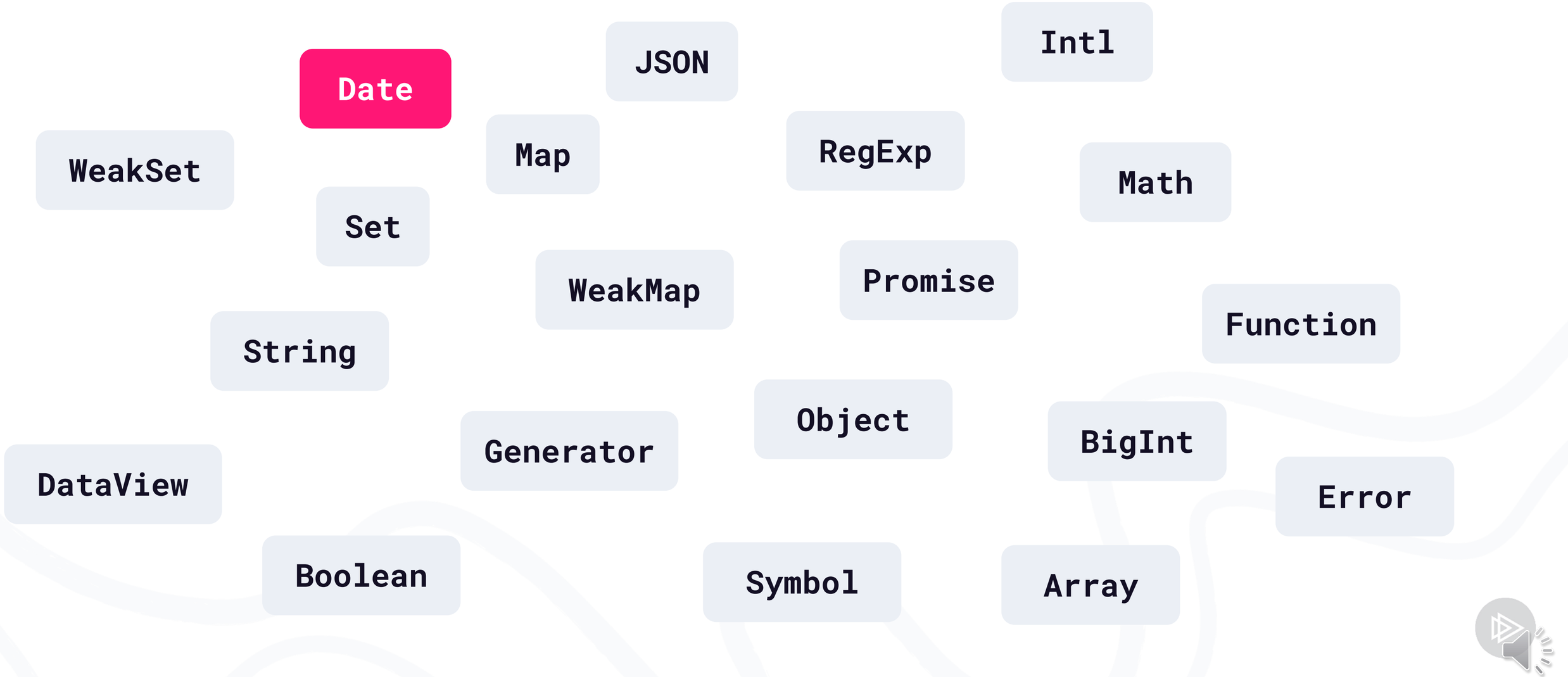
Up Next:

What is a Prototype?

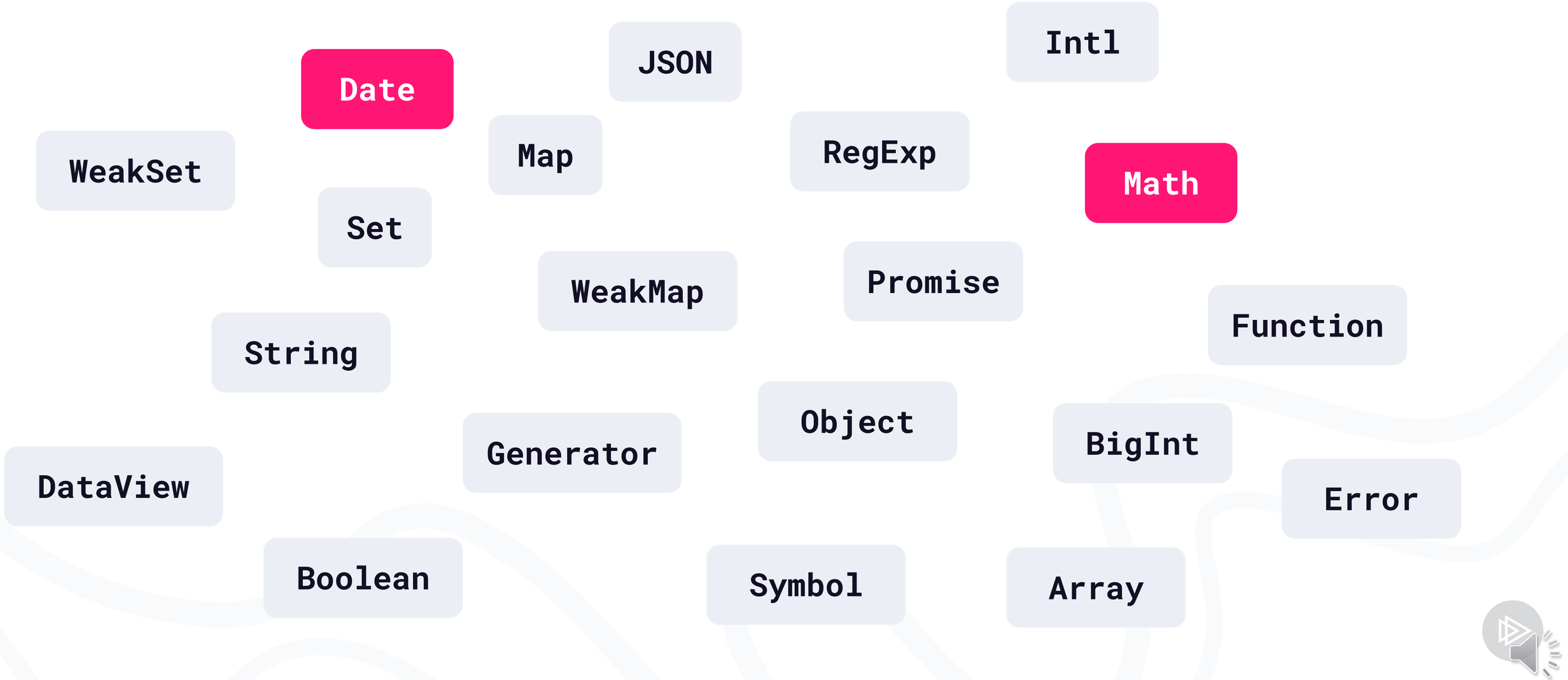
JavaScript Standard Built-in Objects



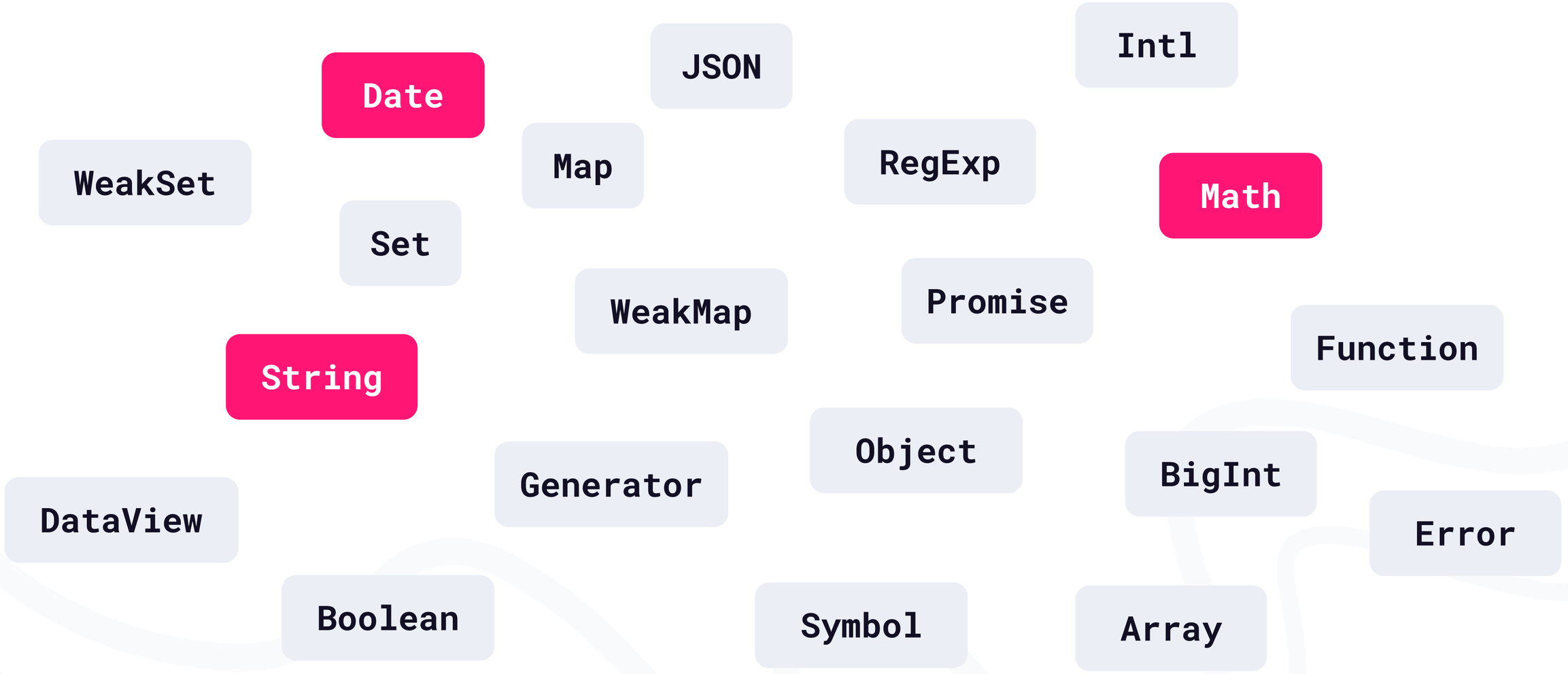
JavaScript Standard Built-in Objects



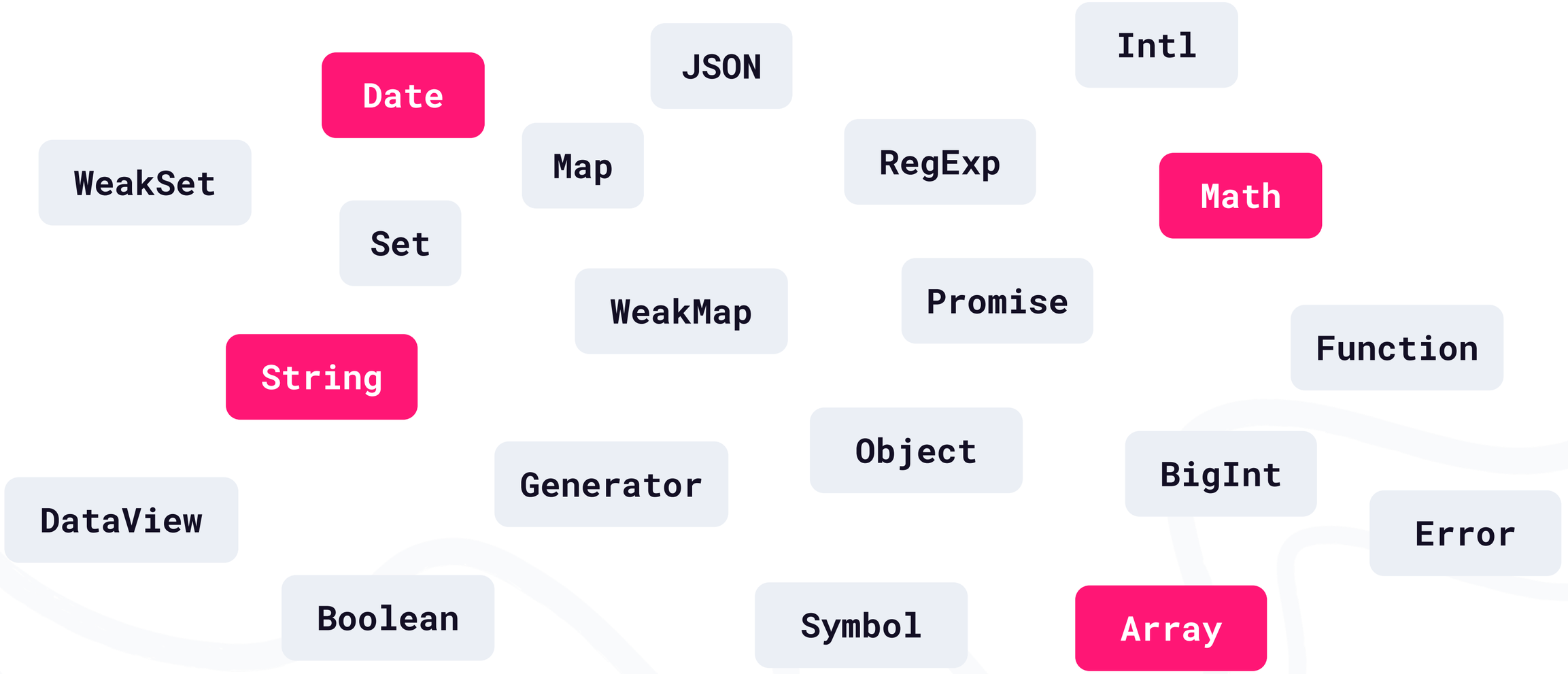
JavaScript Standard Built-in Objects



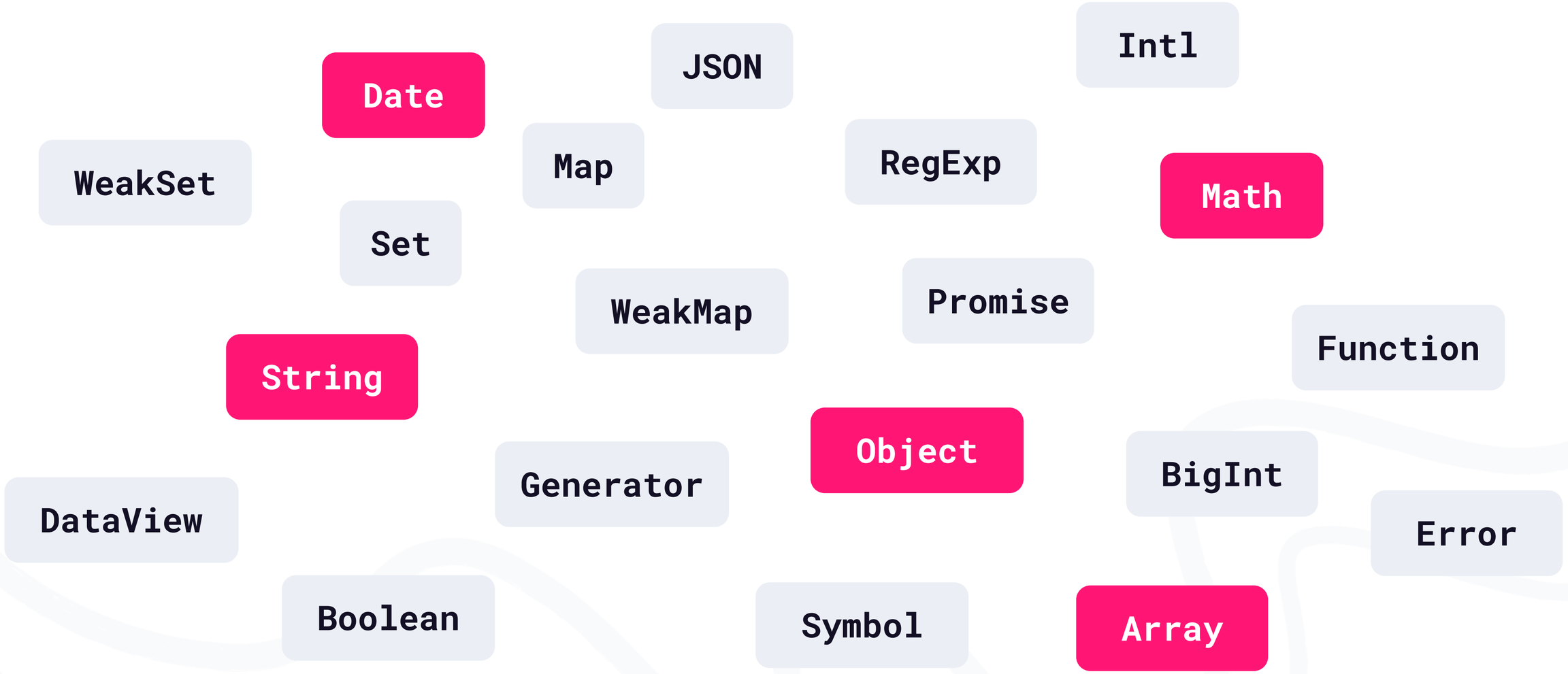
JavaScript Standard Built-in Objects



JavaScript Standard Built-in Objects



JavaScript Standard Built-in Objects

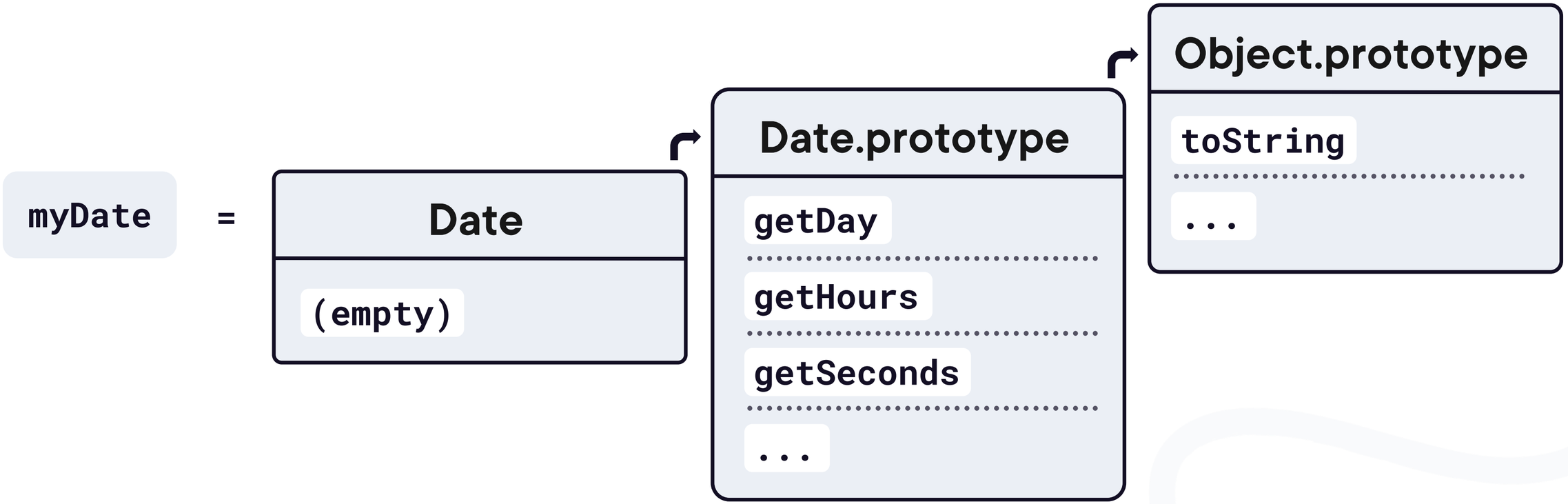


JavaScript Object Prototype Takeaways

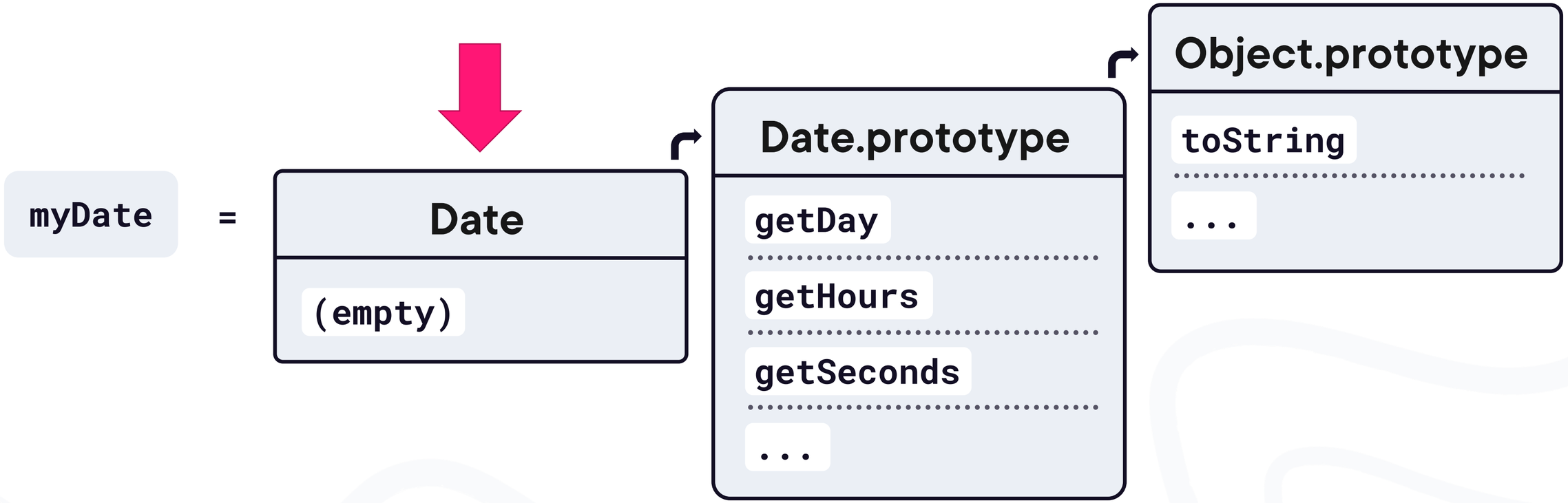
Many objects are predefined for you

Objects link to prototypes which define their behavior

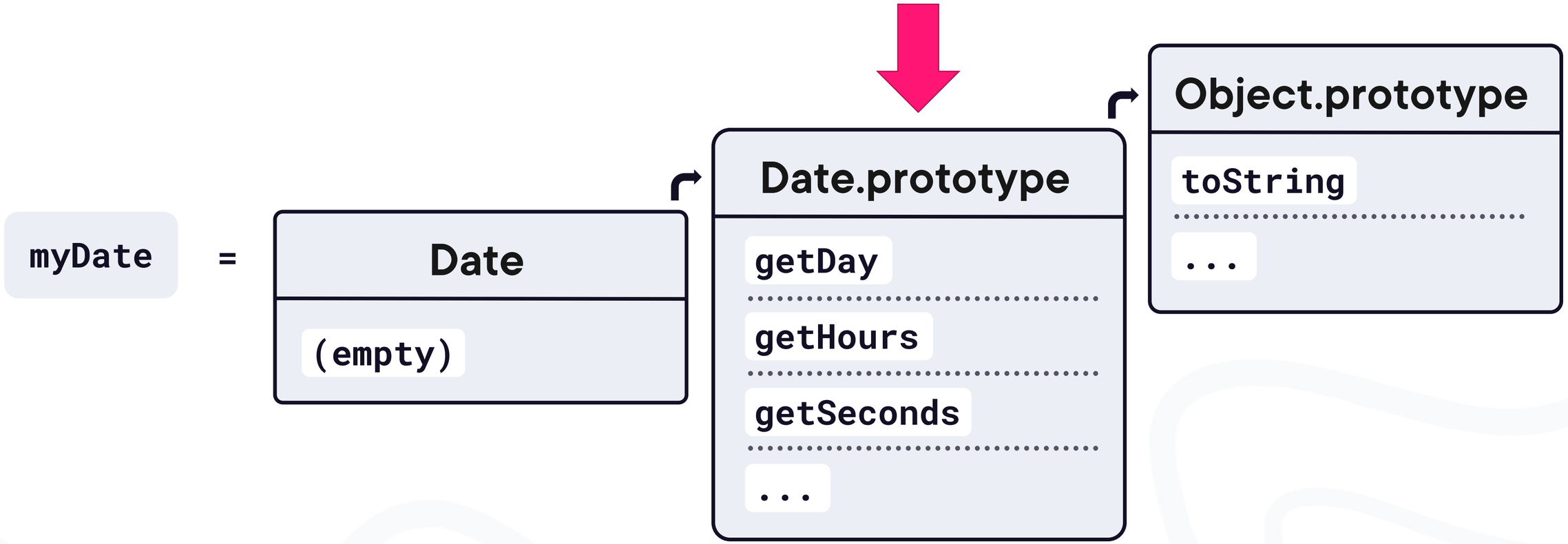
Calling a property on an object will look through the prototype chain


myDate
=
Date
(empty)
Date.prototype
getDay
getHours
getSeconds
...
Object.prototype
toString
...

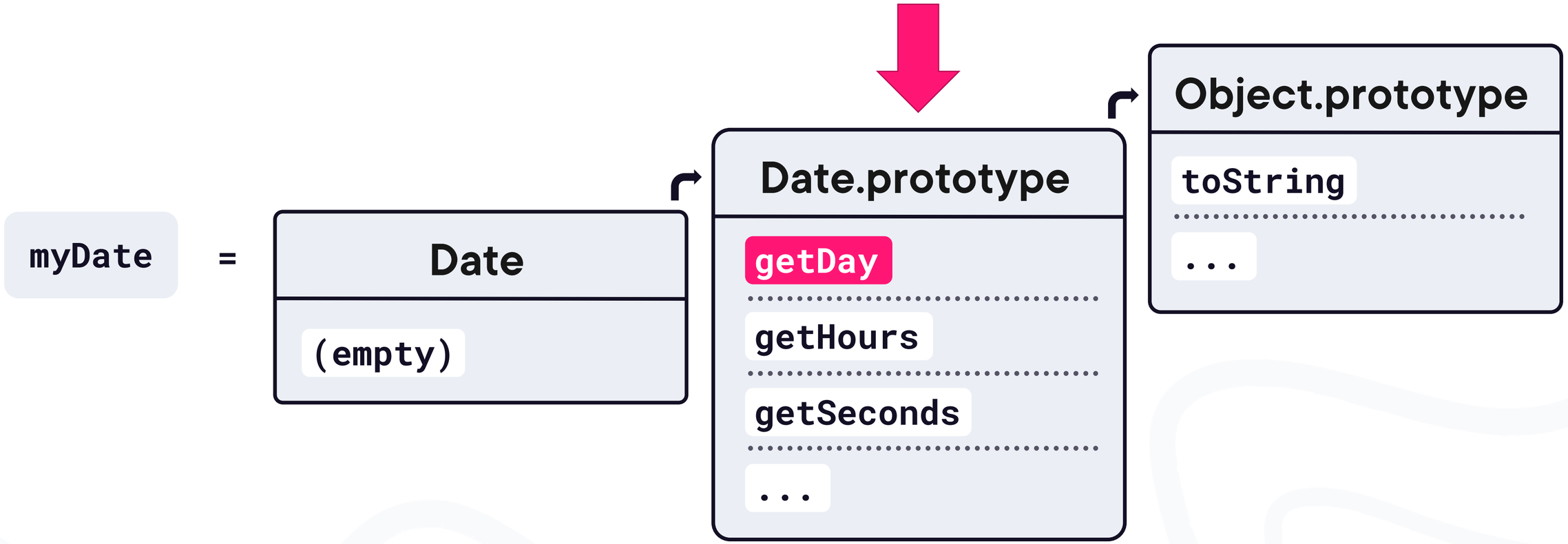
Traversing the Prototype Chain



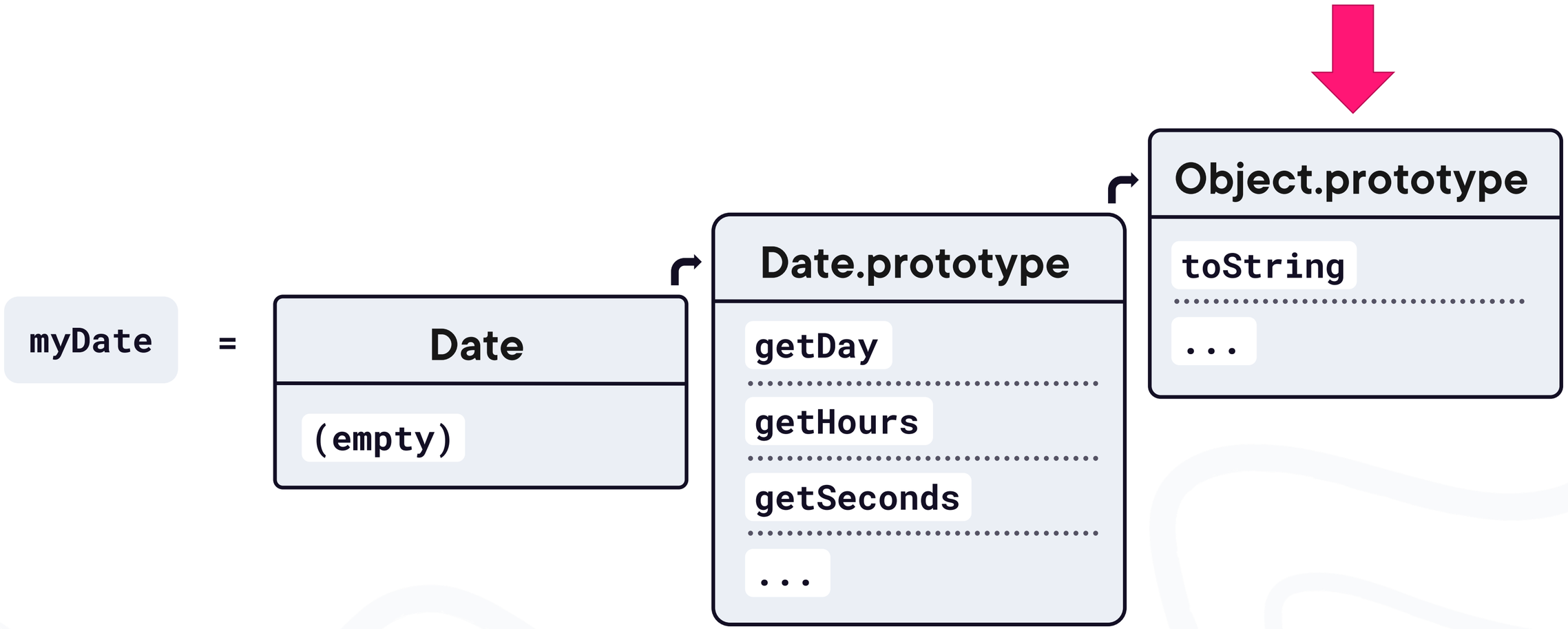
Traversing the Prototype Chain



Traversing the Prototype Chain



Traversing the Prototype Chain



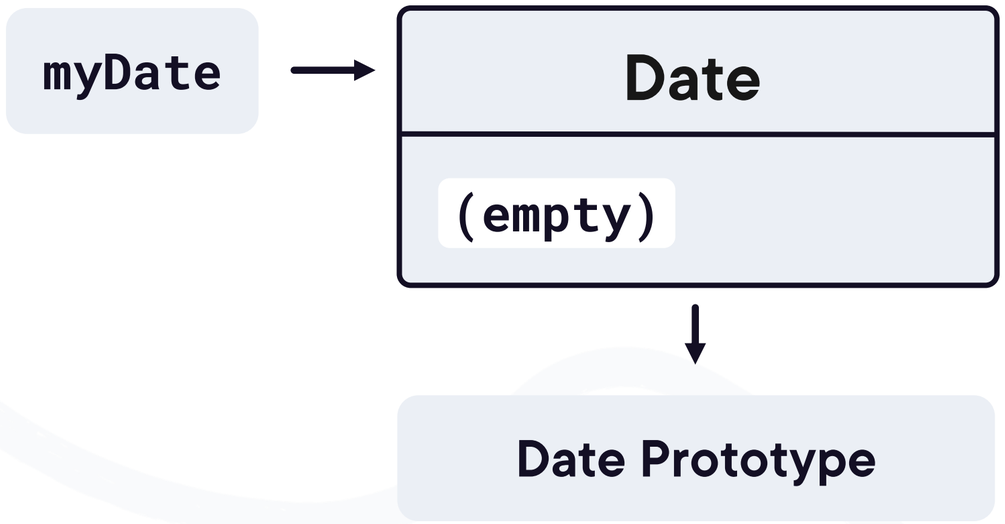
Up Next:

Defining Prototypes with Object.create

Creating Objects

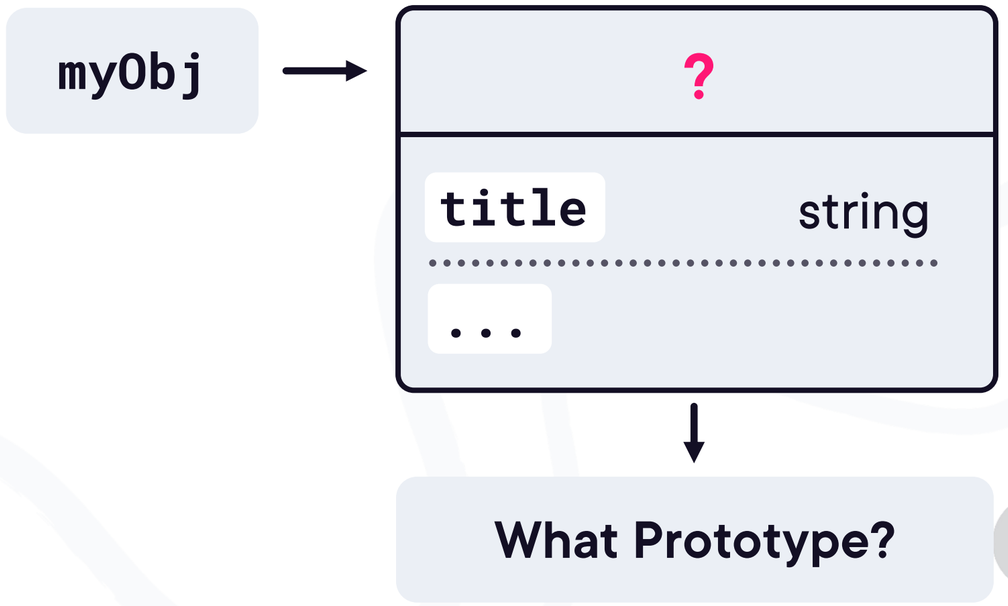
New Syntax

```
const myDate = new Date();
```



Object Literal

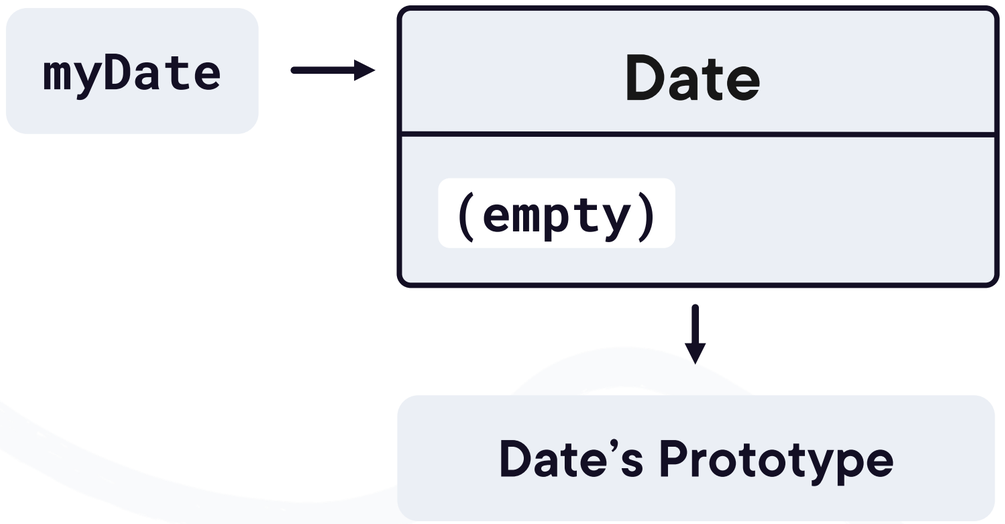
```
const myObj = { ... };
```



Creating Objects

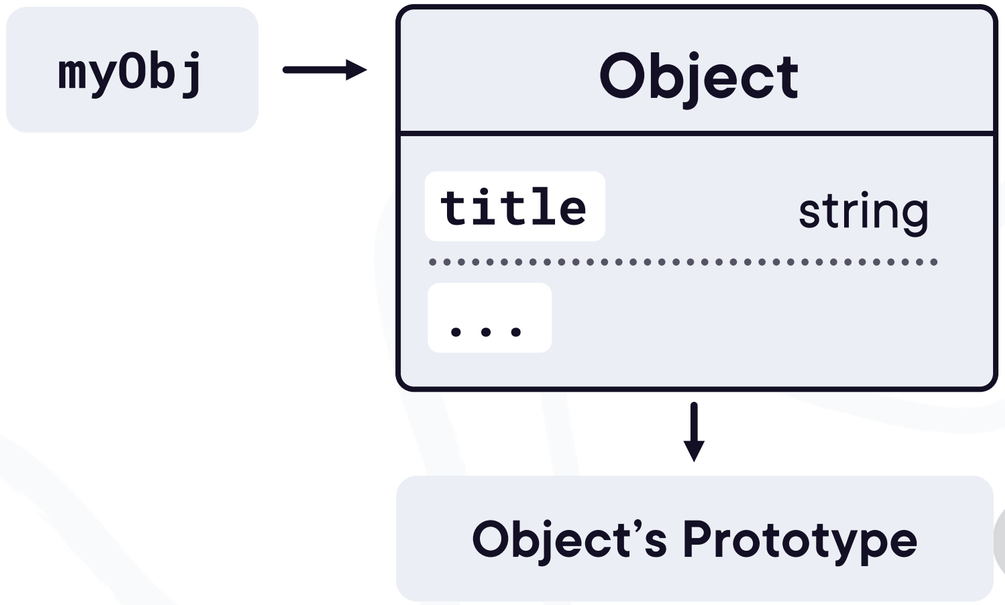
New Syntax

```
const myDate = new Date();
```

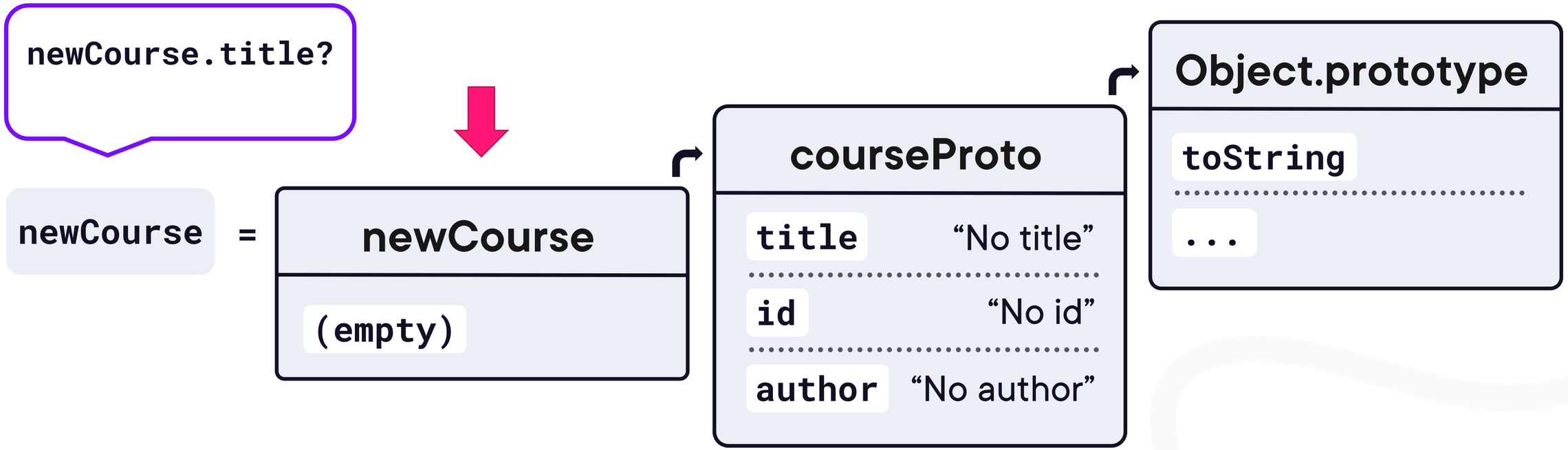


Object Literal

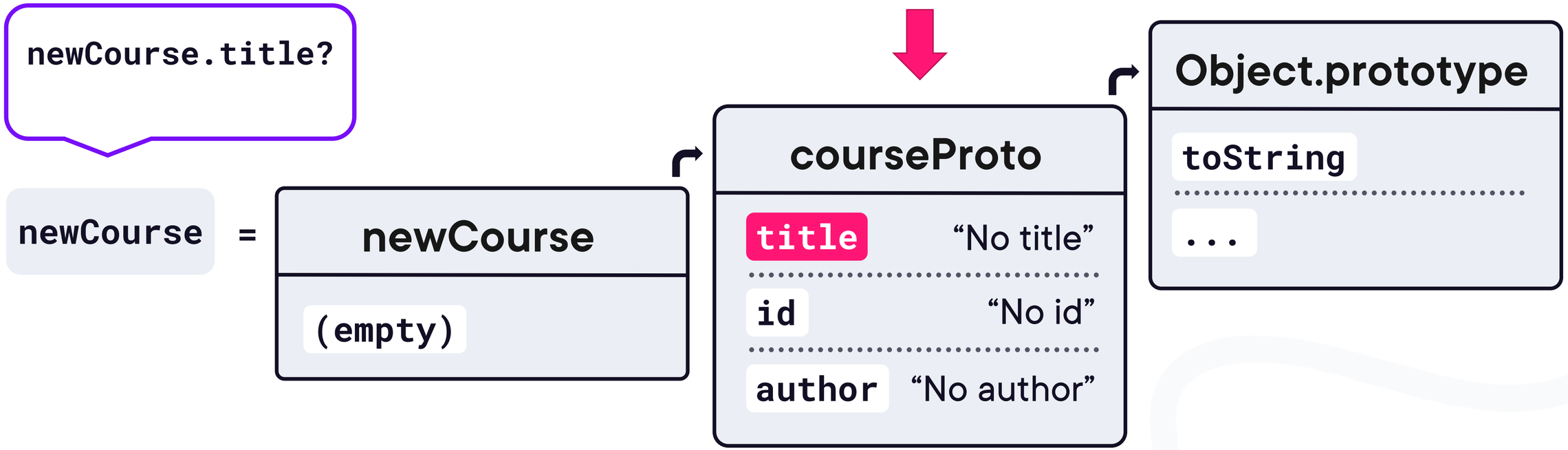
```
const myObj = { ... };
```



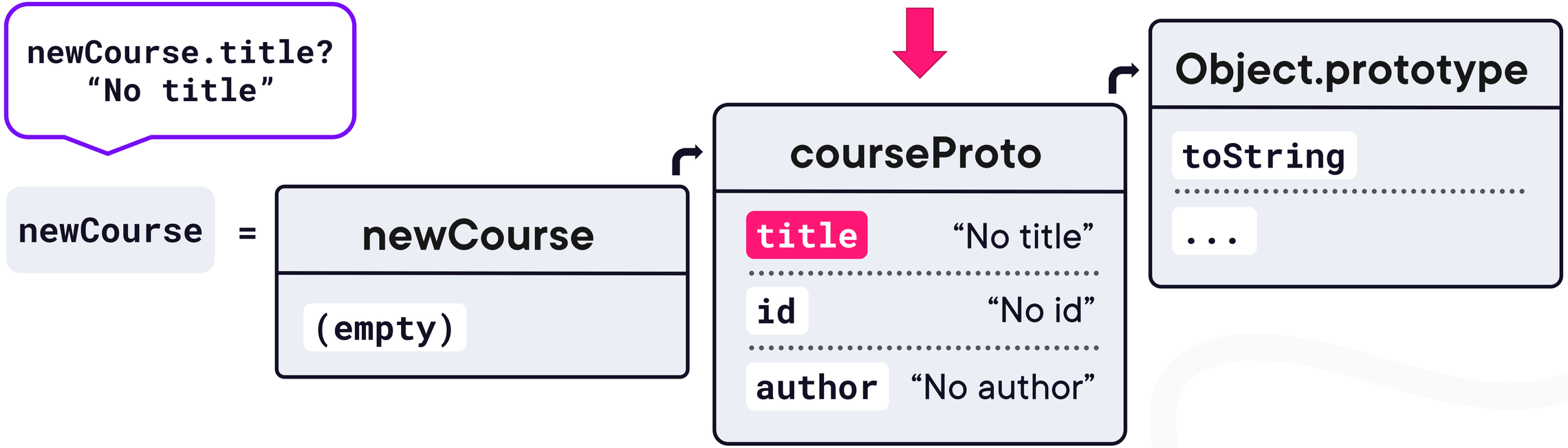
myCourse's Prototype Chain



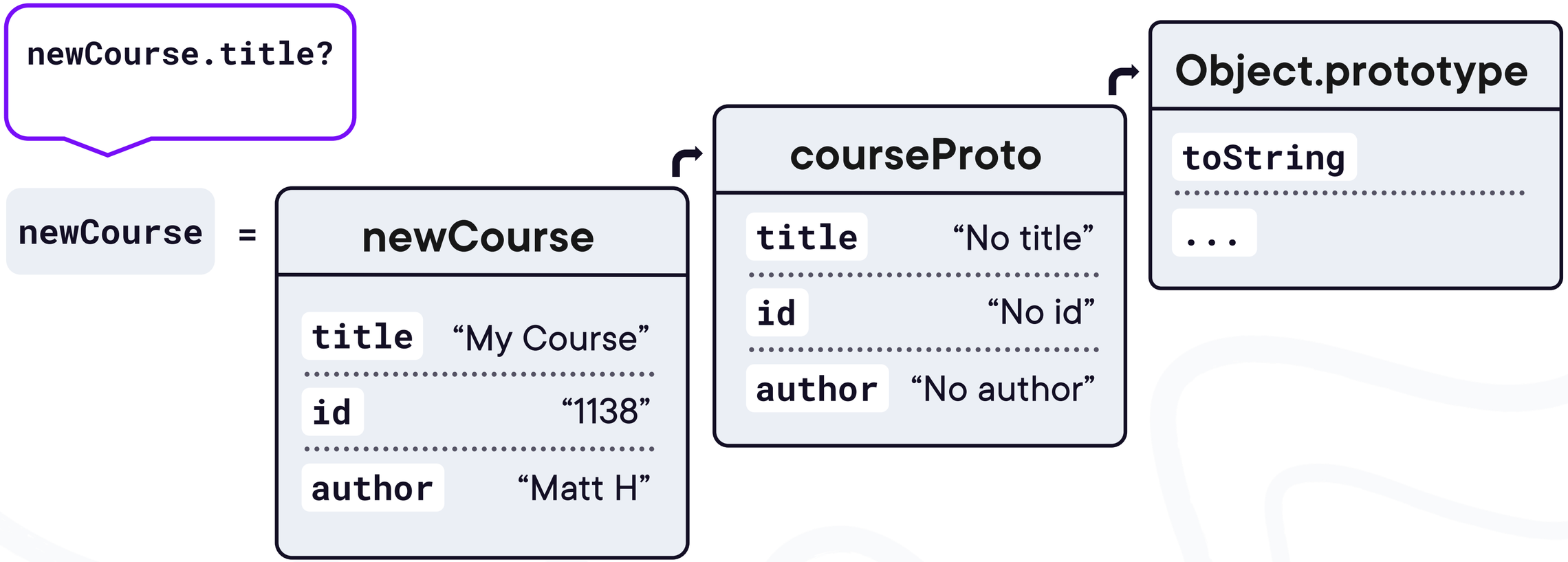
myCourse's Prototype Chain



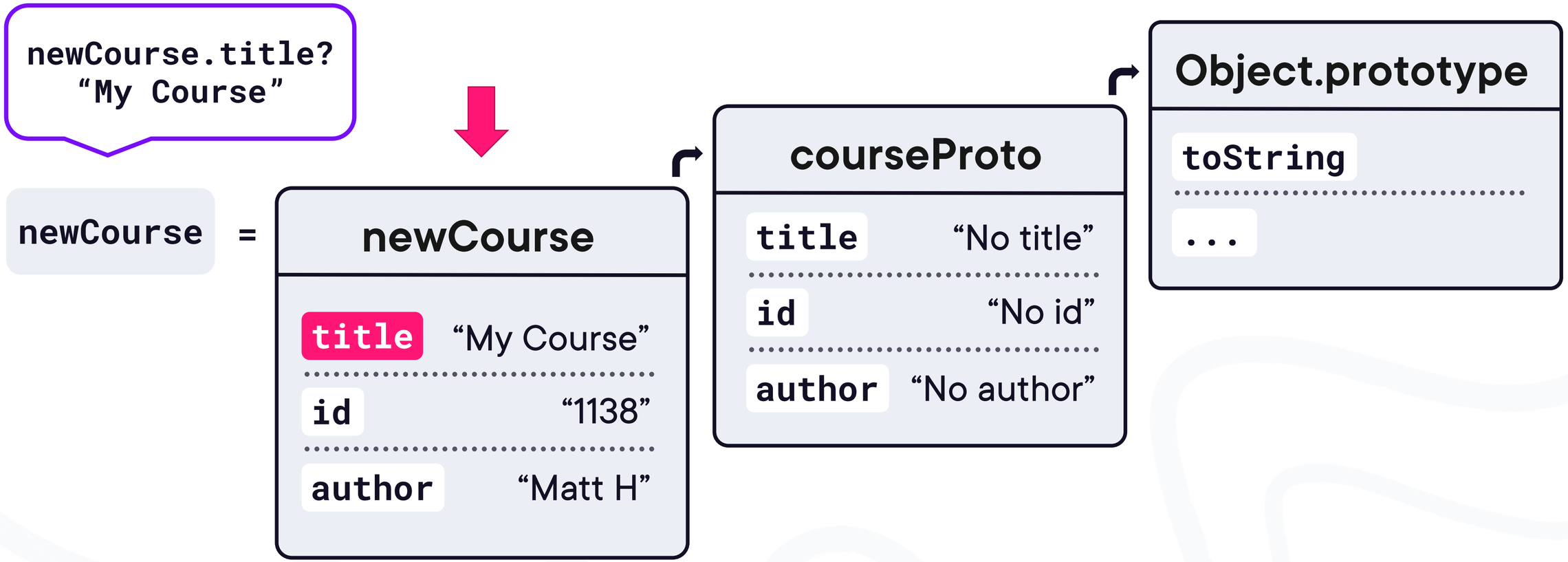
myCourse's Prototype Chain



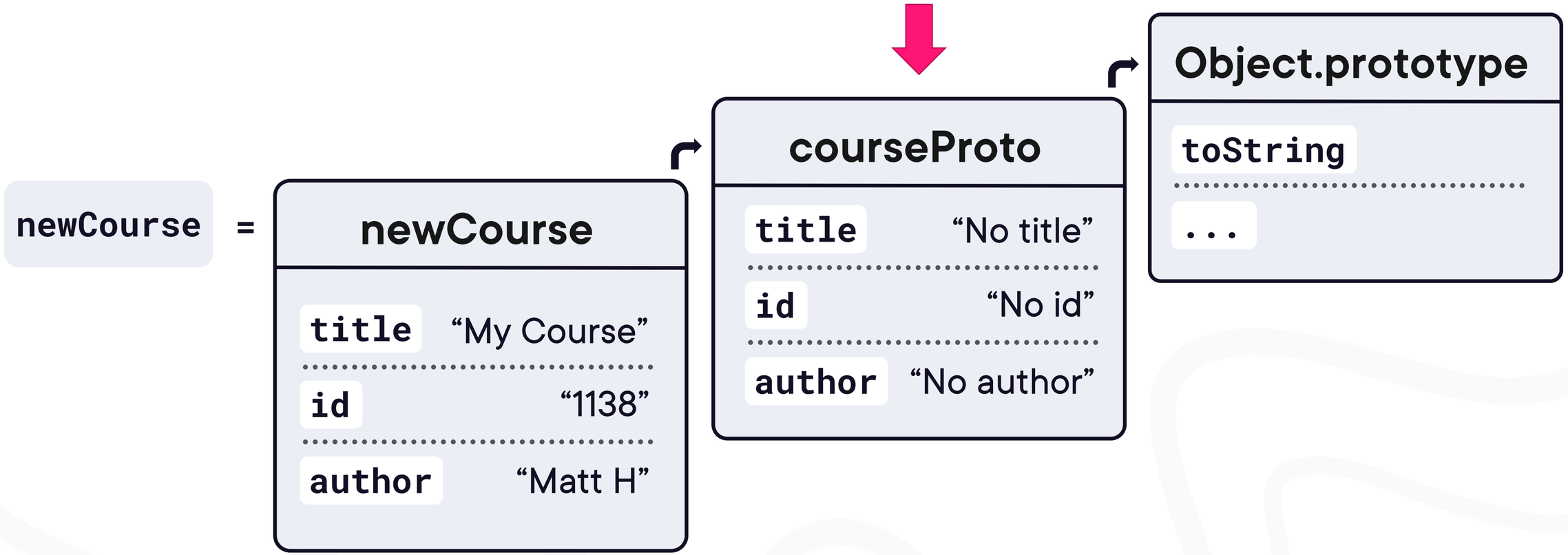
myCourse's Prototype Chain



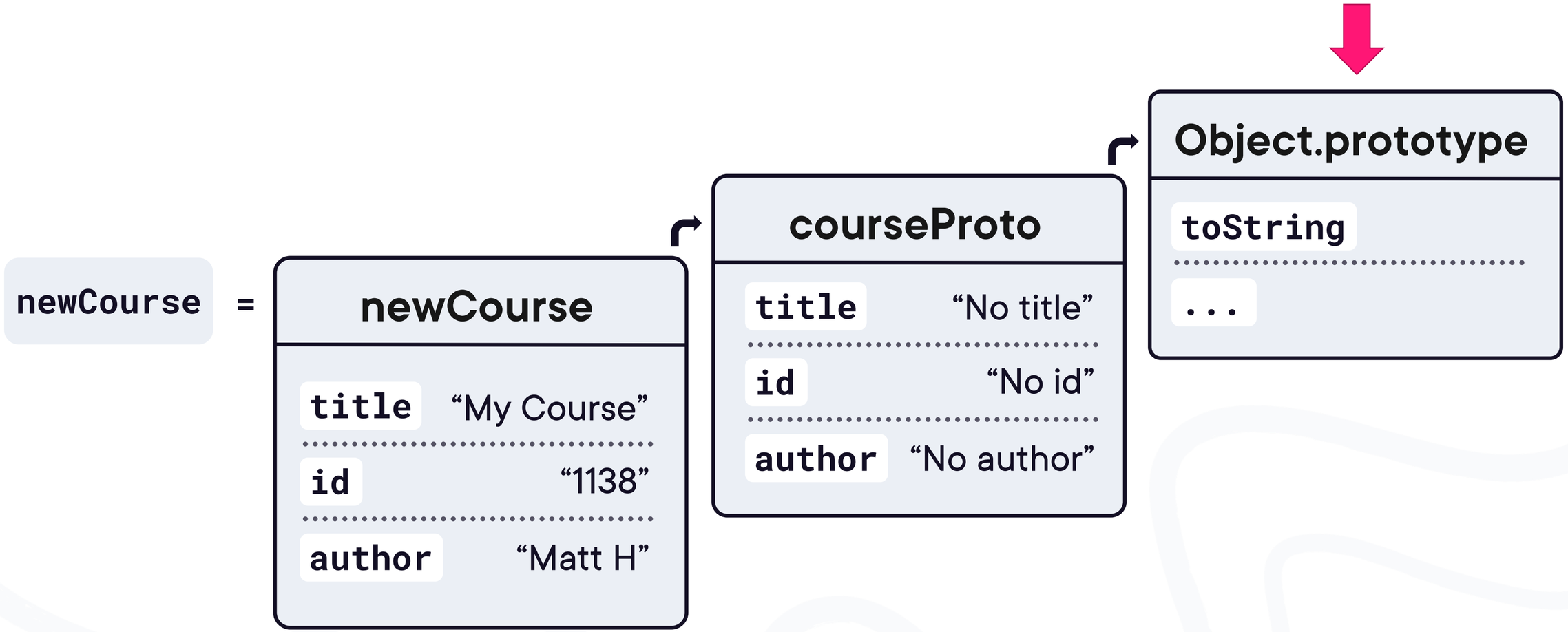
myCourse's Prototype Chain



myCourse's Prototype Chain



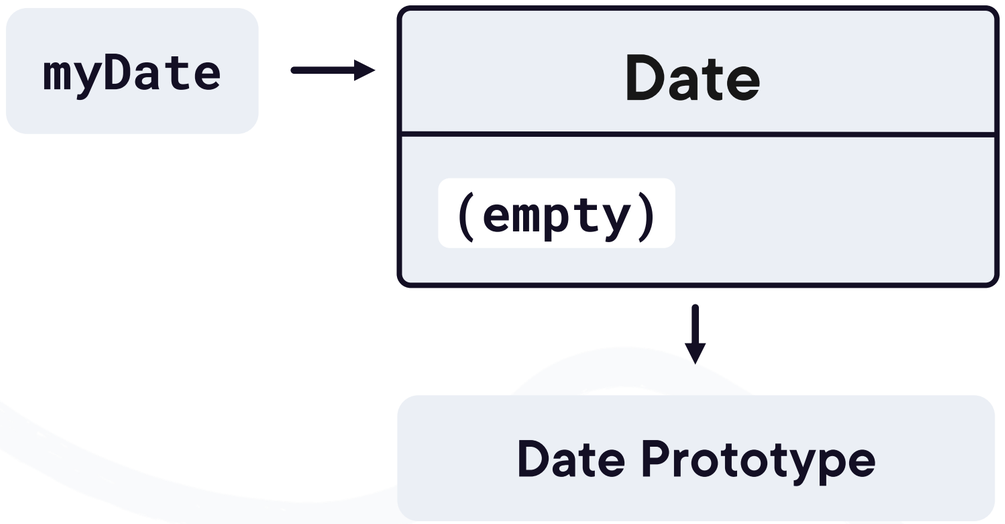
myCourse's Prototype Chain



Creating Objects

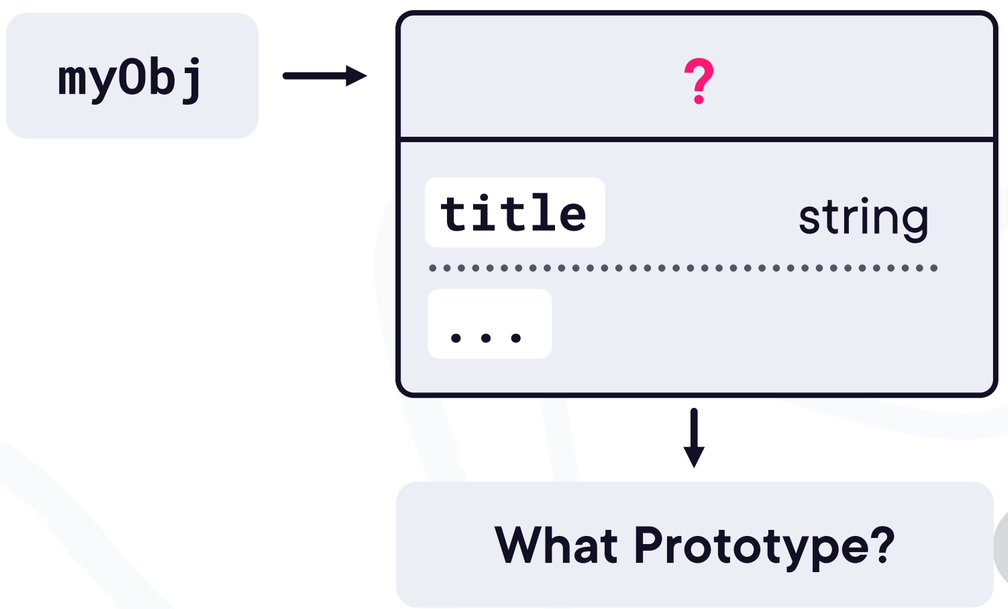
New Syntax

```
const myDate = new Date();
```

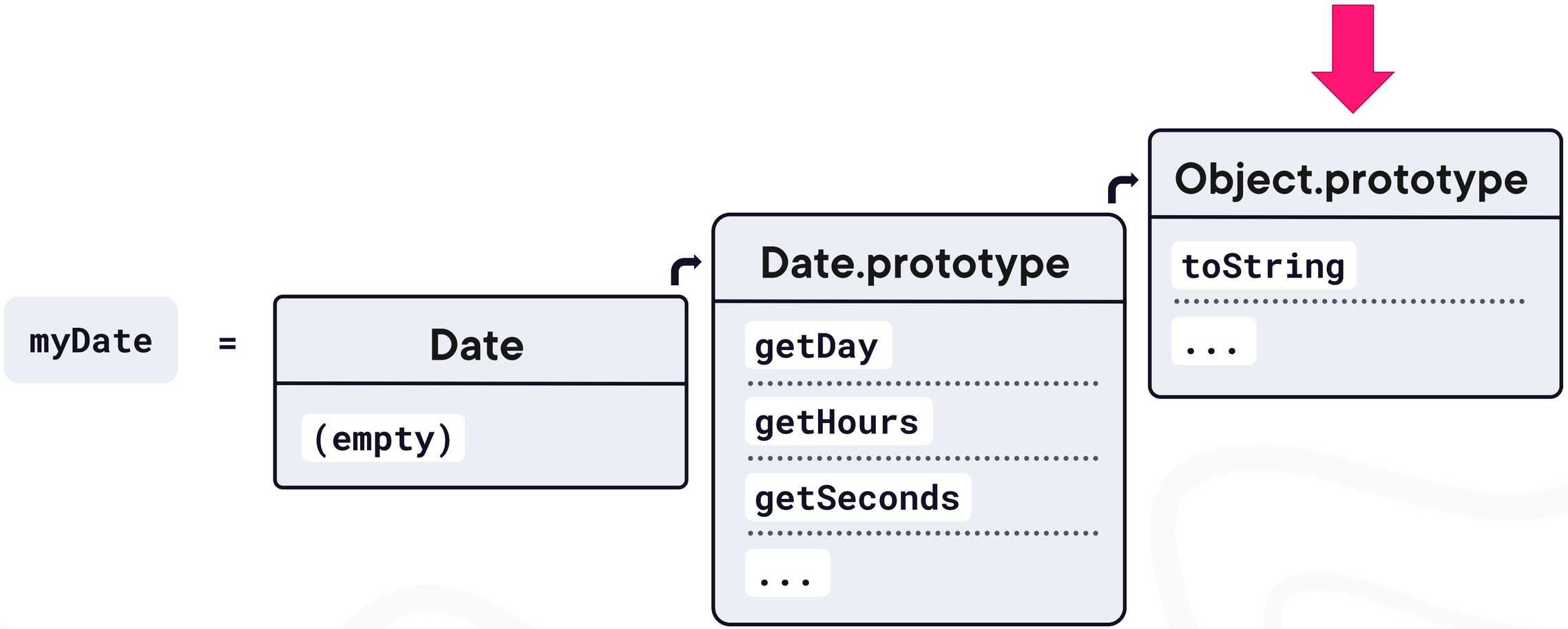


Object Literal

```
const myObj = { ... };
```



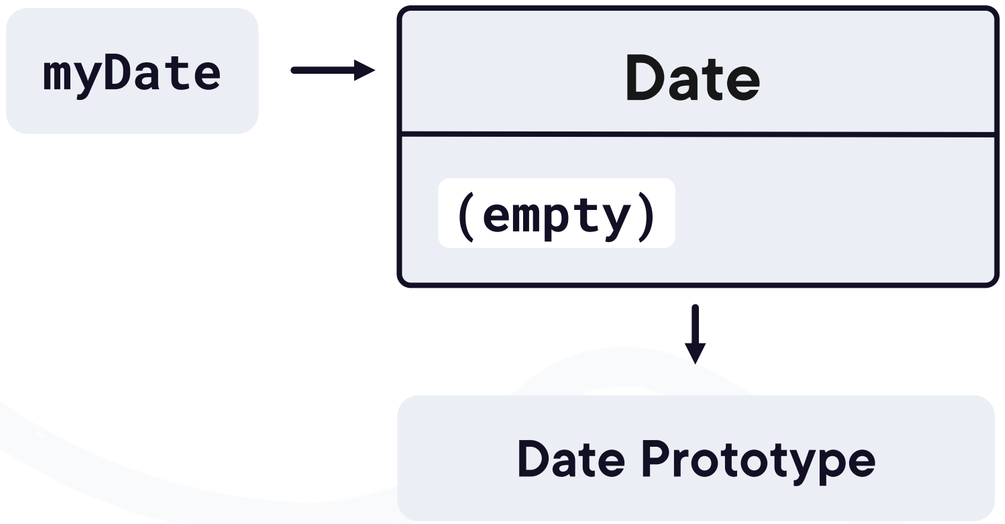
Traversing the Prototype Chain



Creating Objects

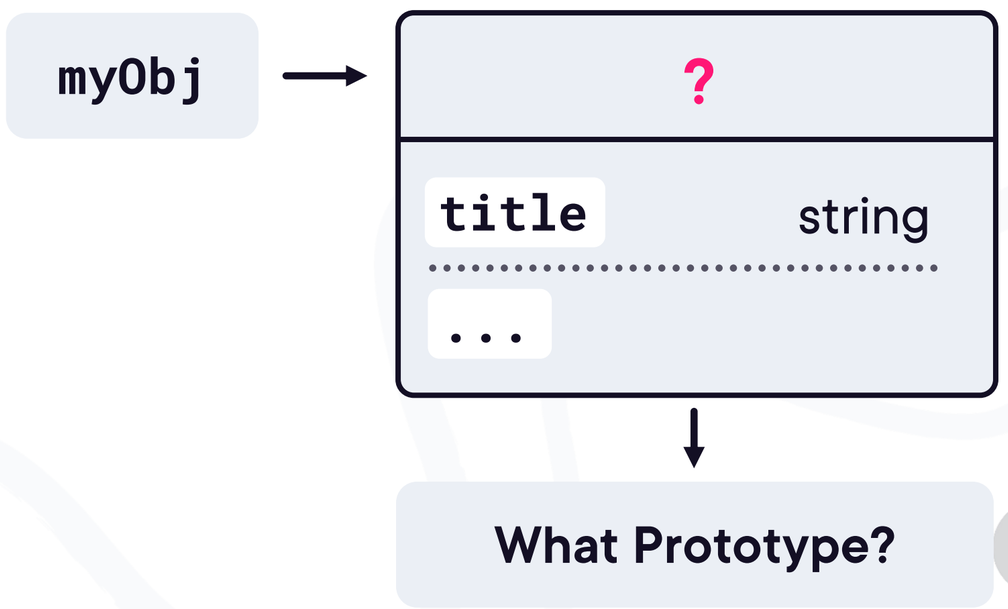
New Syntax

```
const myDate = new Date();
```



Object Literal

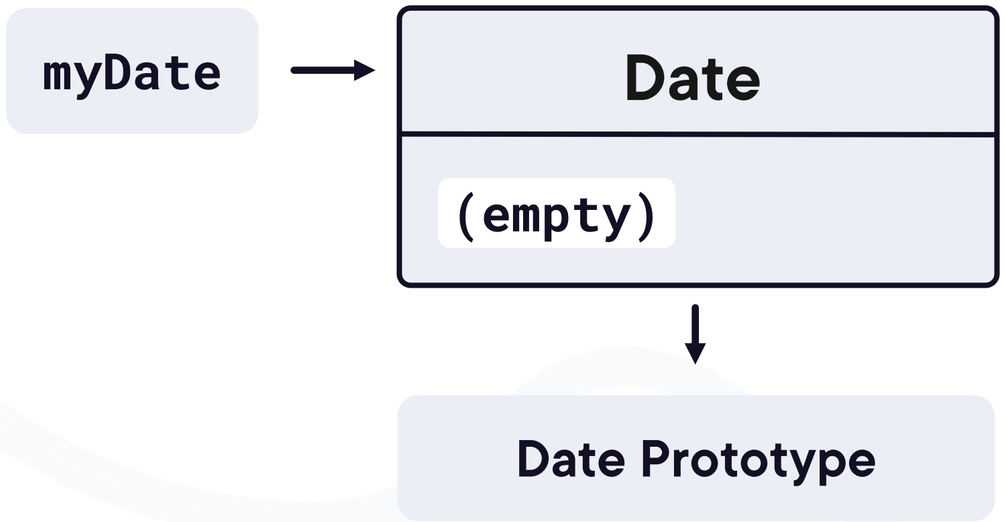
```
const myObj = { ... };
```



Creating Objects

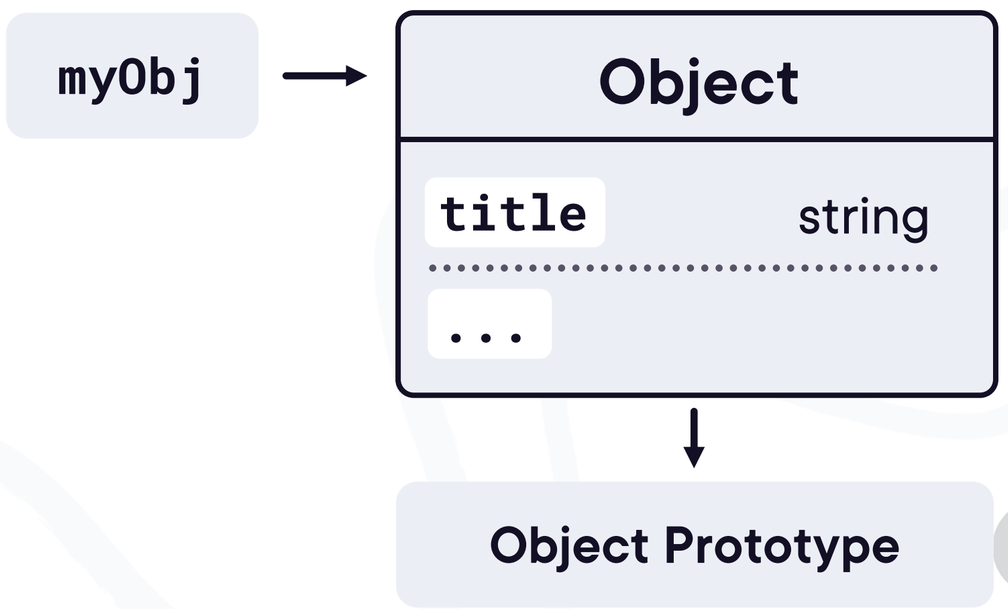
New Syntax

```
const myDate = new Date();
```



Object Literal

```
const myObj = { ... };
```



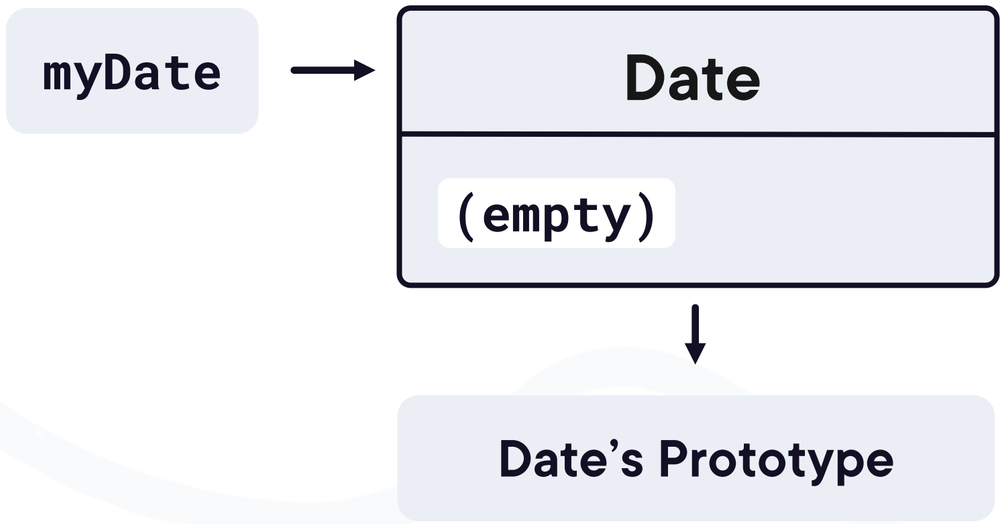
“object” vs “Object”

object	VS	Object
The general word for the datatype		A standard built-in object All objects link to Object.prototype

Creating Objects

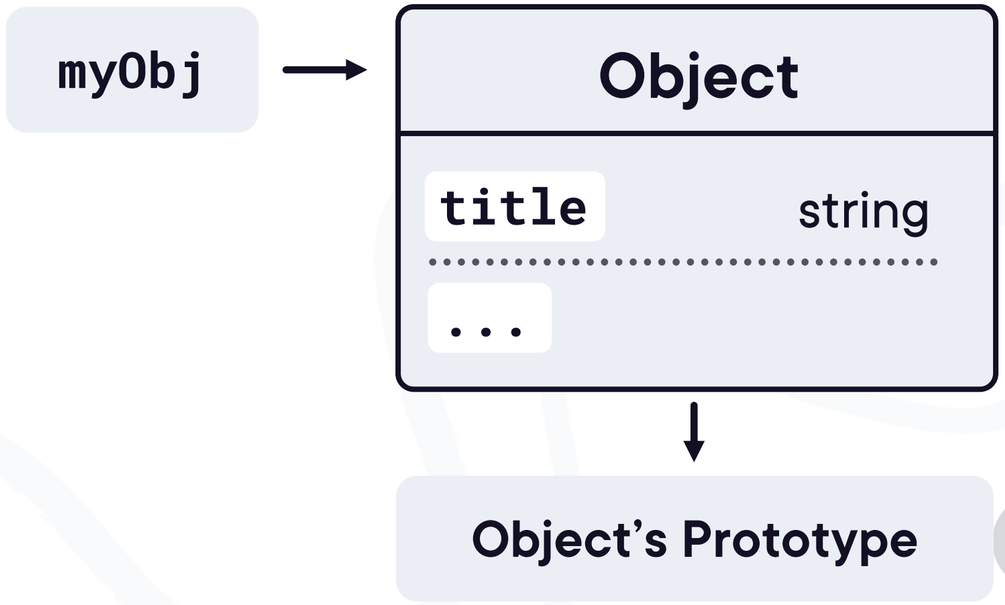
New Syntax

```
const myDate = new Date();
```



Object Literal

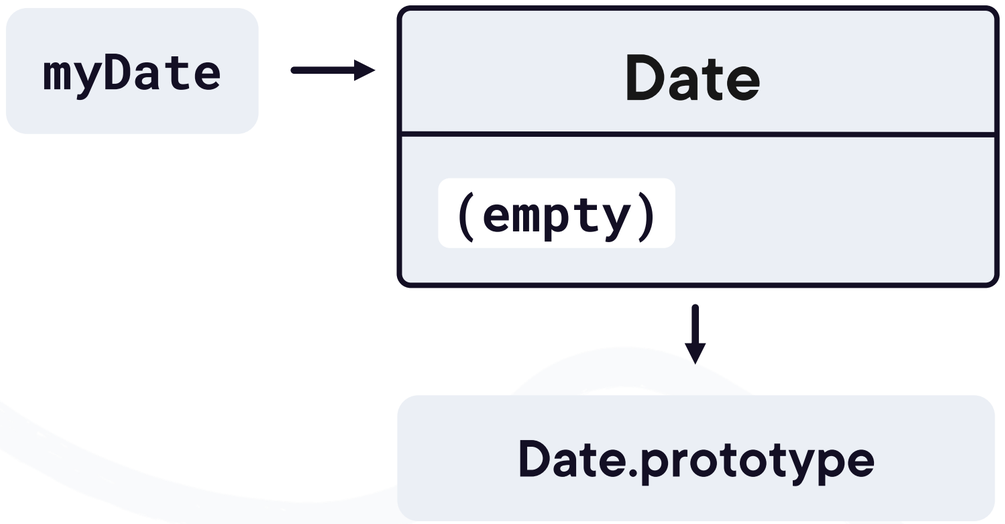
```
const myObj = { ... };
```



Creating Objects

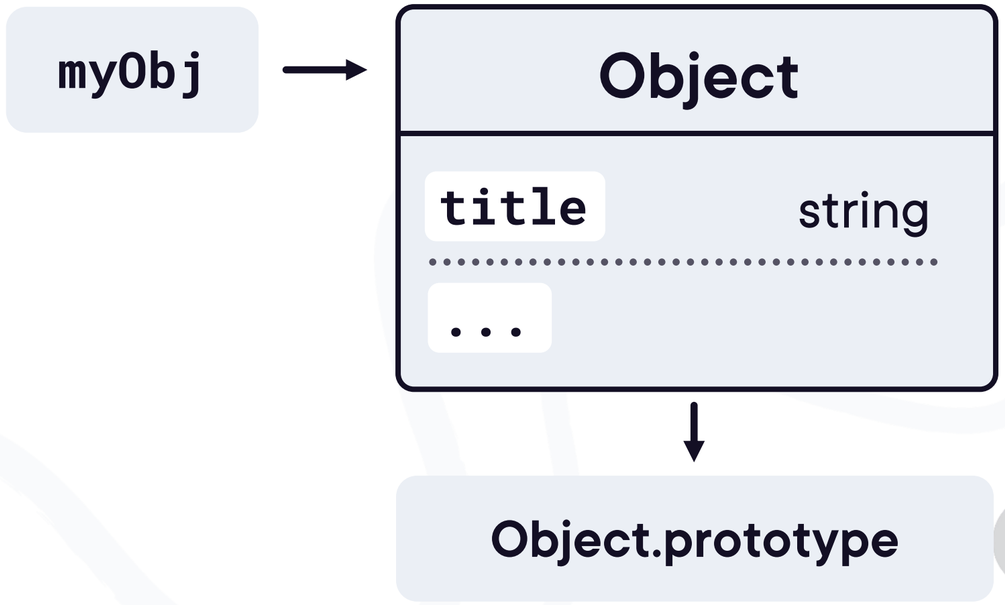
New Syntax

```
const myDate = new Date();
```



Object Literal

```
const myObj = { ... };
```



Creating Objects: Object.create

```
const myObj = Object.create(Object.prototype);
```

Two Ways to Create Objects

Object Literal

```
const myObj = { ... };
```

New Syntax

```
const myObj = new Object();
```

Three Ways to Create Objects

Object Literal

```
const myObj = { ... };
```

New Syntax

```
const myObj = new Object();
```

Object.create

```
const myObj = Object.create(Object.prototype);
```

Creating Objects: Object.create

```
const myObj = Object.create();
```

Creating Objects: Object.create

```
const myObj = Object.create(Object.prototype);
```

Three Ways to Create Objects

	Object	Date
Object Literal	`{ ... };`	*- Not applicable -*
New Syntax	`new Object();`	`new Date();`
Object.create	`Object.create(Object.prototype);`	`Object.create(Date.prototype);`

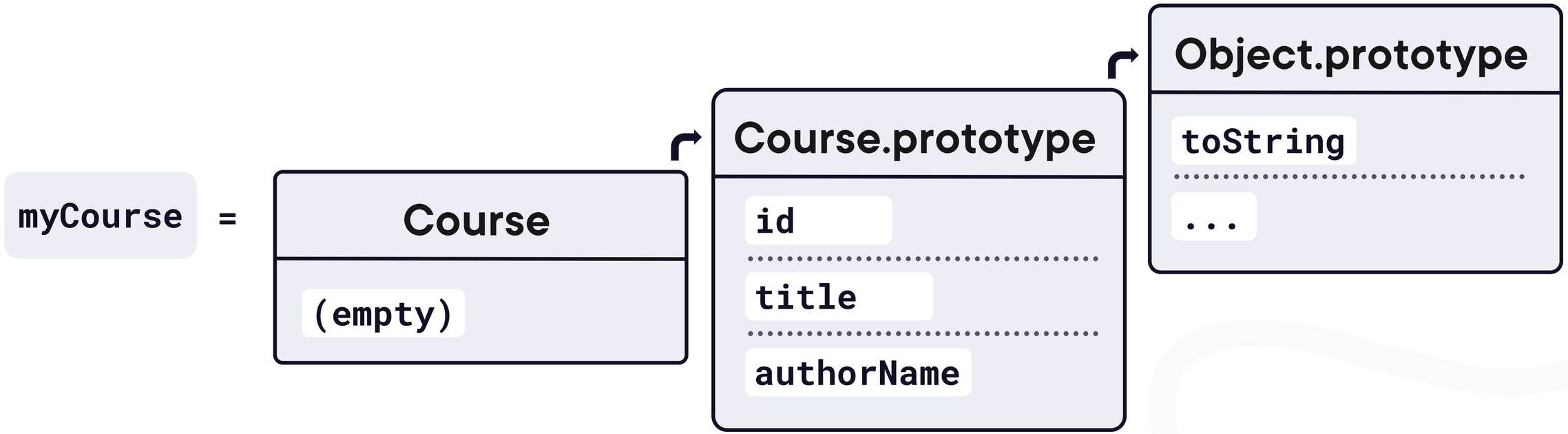
Creating Objects: Object.create

```
const myCourse = Object.create()
```

Creating Objects: Object.create

```
const myCourse = Object.create()
```

Traversing the Prototype Chain



All objects link to a prototype

All objects link to a prototype*

Creating an Object With No Prototype

```
const myNullObj = Object.create(null);
```

JavaScript Object Prototypes

Many objects are predefined for you

Objects link to prototypes which define their behavior

Calling a property on an object will look through the prototype chain

Object literals link to Object.prototype

Object.prototype sits at the end of prototype chains

An object's prototype can be set...

- With the new syntax
- By using Object.create()

Up Next:

Creating Properties

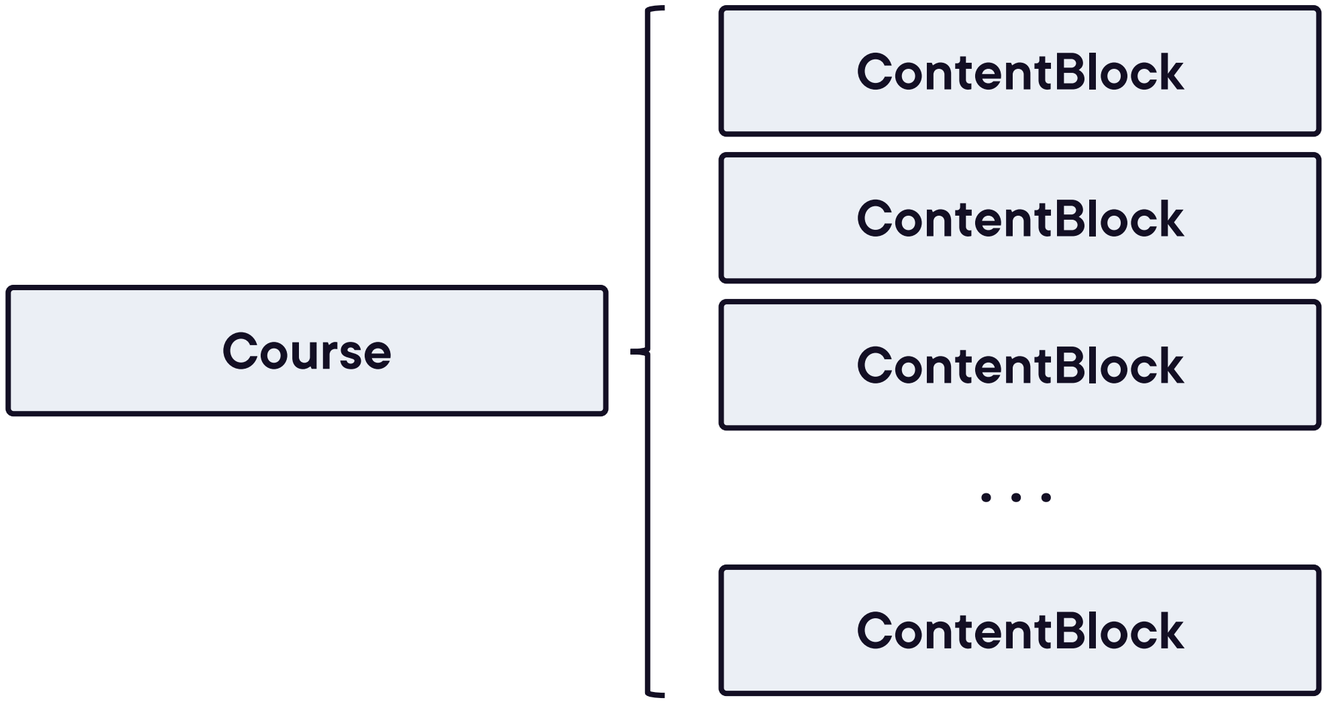
Courses are Made Up of ContentBlocks

ContentBlock

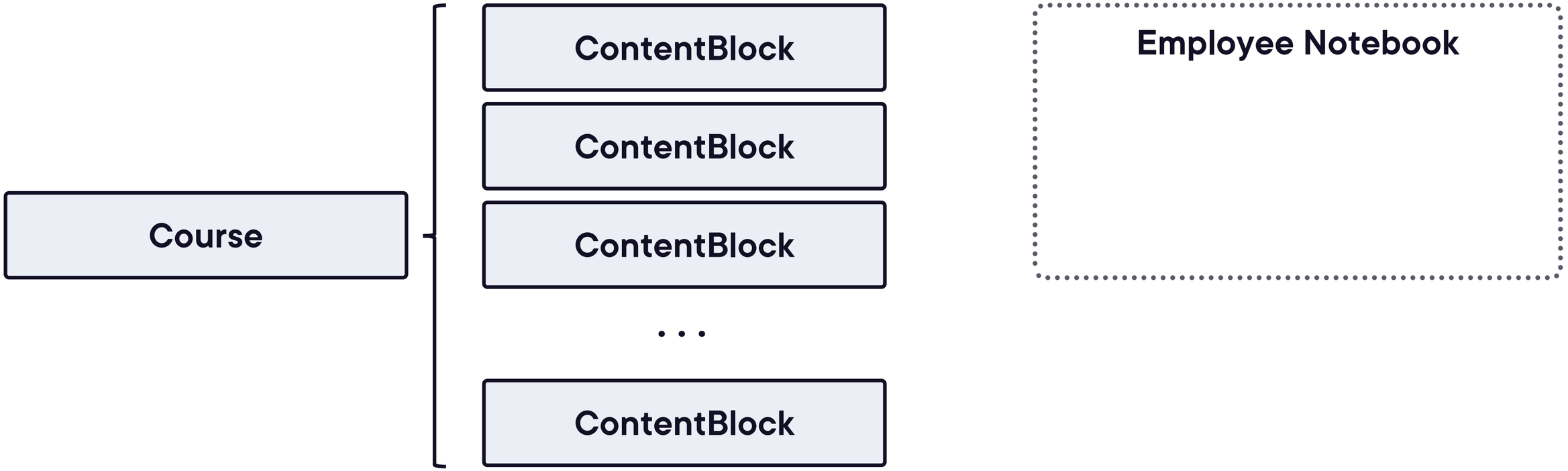
“Employees are required to use the company VPN to access shared resources. Access information will be provided to you on your first working day…”

Courses are Made Up of ContentBlocks

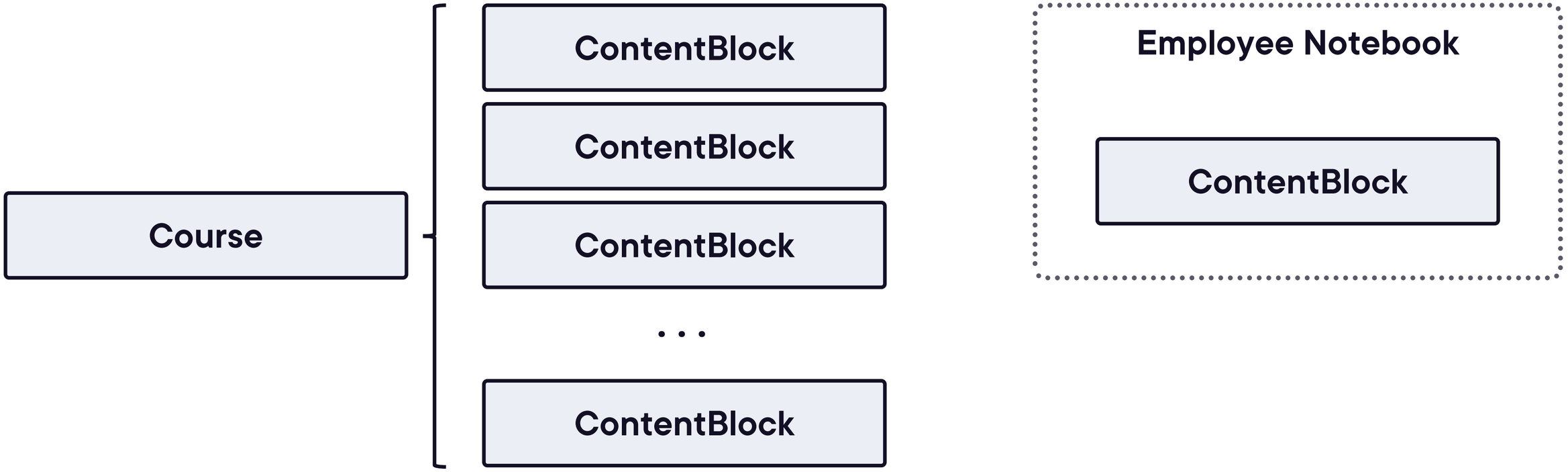
Courses are Made Up of ContentBlocks



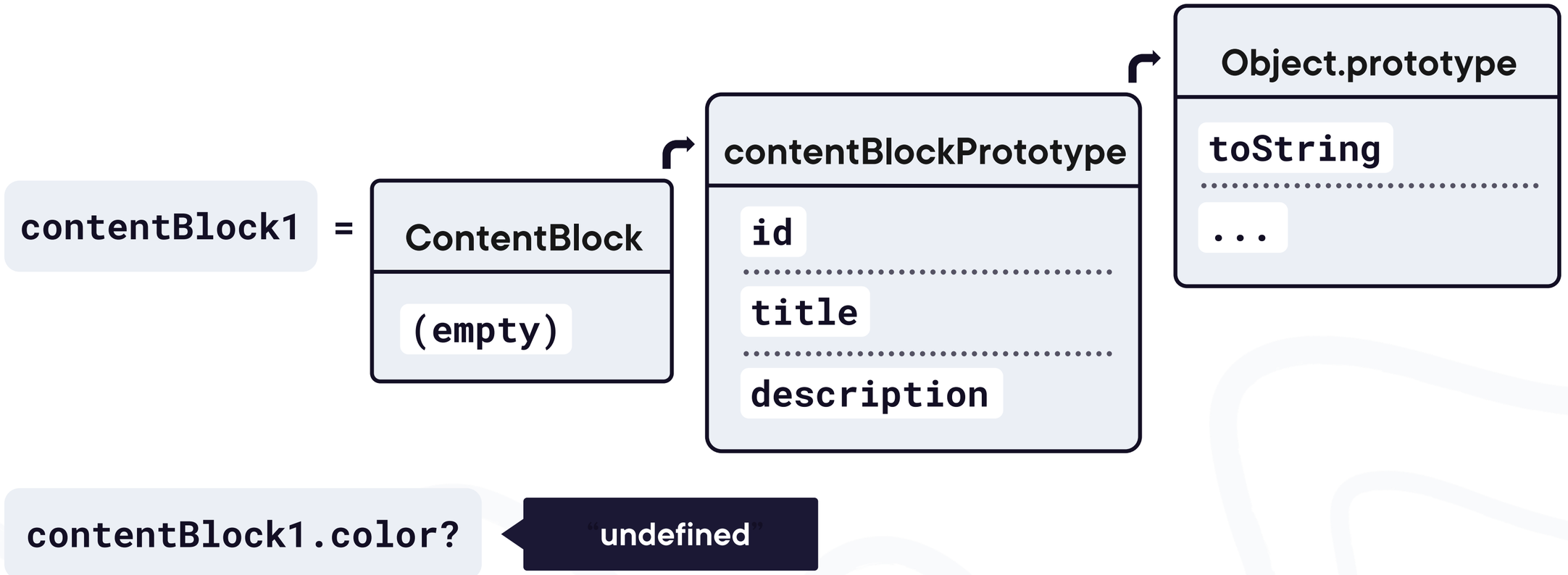
Courses are Made Up of ContentBlocks



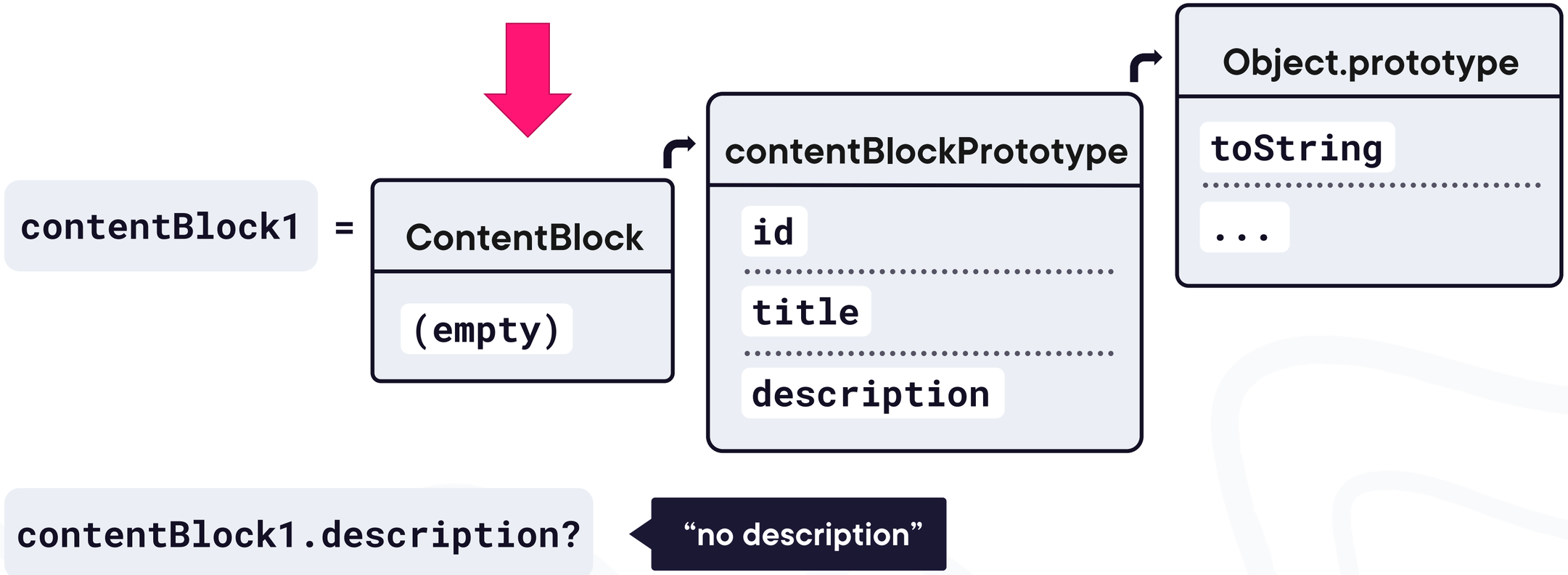
Courses are Made Up of ContentBlocks



ContentBlock's Prototype

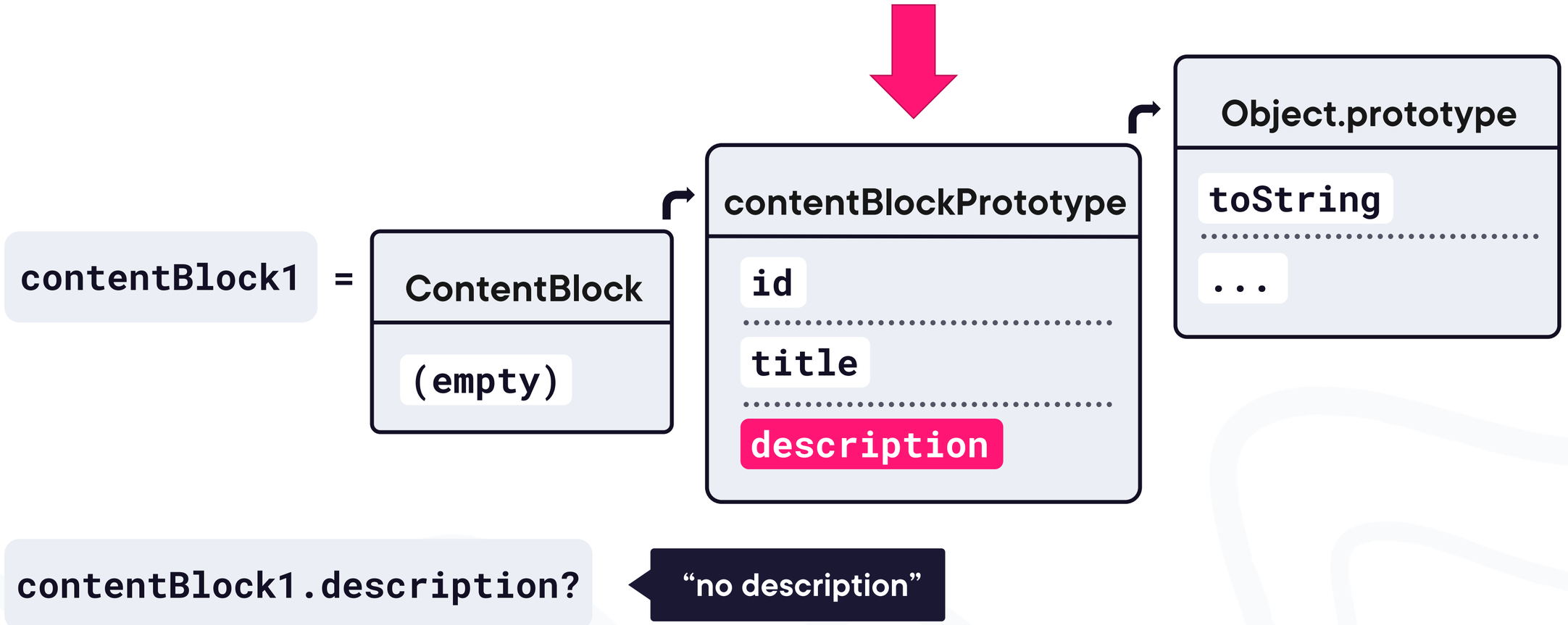


ContentBlock's Prototype



`contentBlock1.description?` → "no description"

ContentBlock's Prototype



`contentBlock1.description?` → "no description"

Dot Notation vs. Bracket Notation

Dot Notation

`contentBlock1.description`

Shortest way to access properties

Can’t be used for illegal variable names

VS

Bracket Notation

`contentBlock1["description"]`

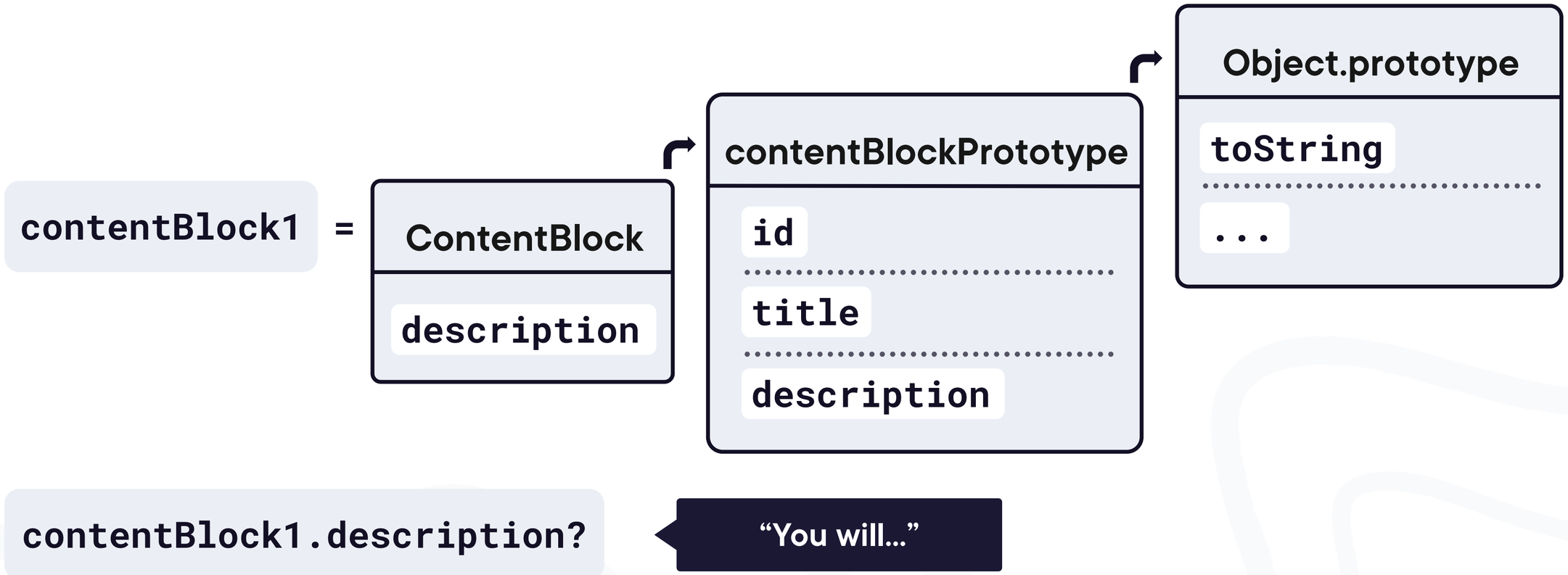
More flexible

Can compute property name at runtime

Computing Property Names at Runtime

```
for (let i = 0; i < 5; i++) {
    contentBlock["info" + i] = "info";
}
```

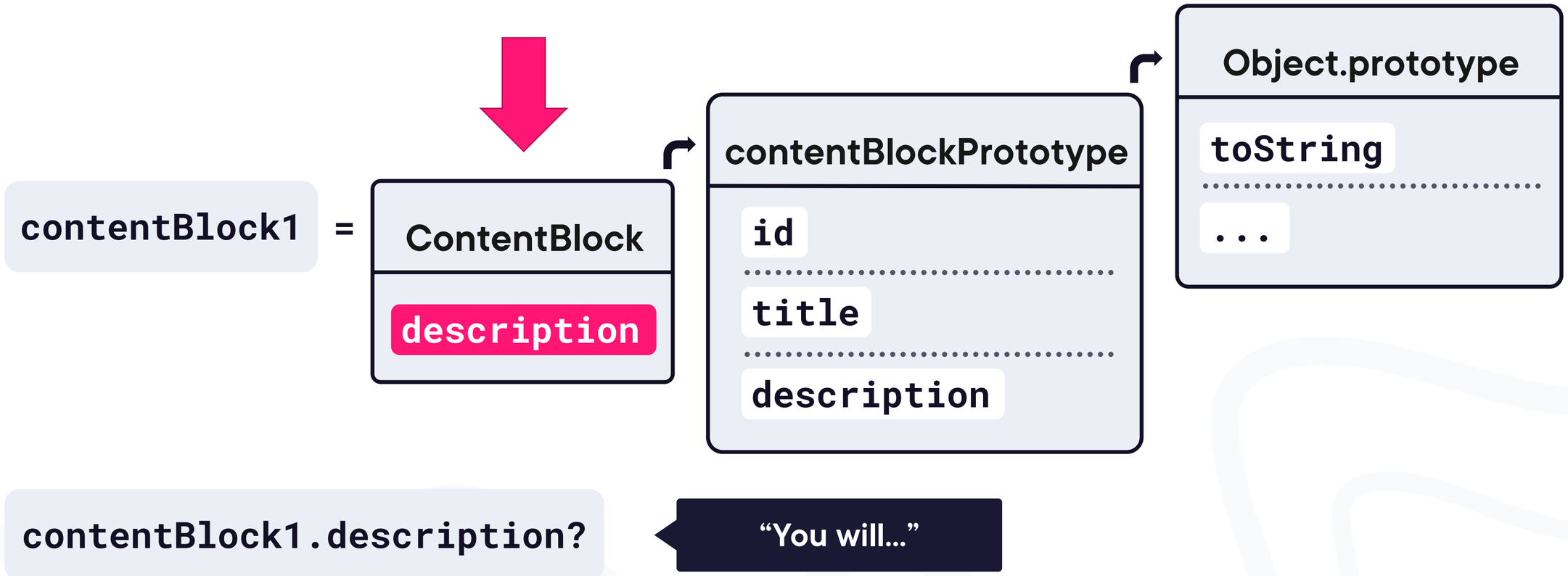
ContentBlock's Prototype



`contentBlock1.description?`

"You will..."

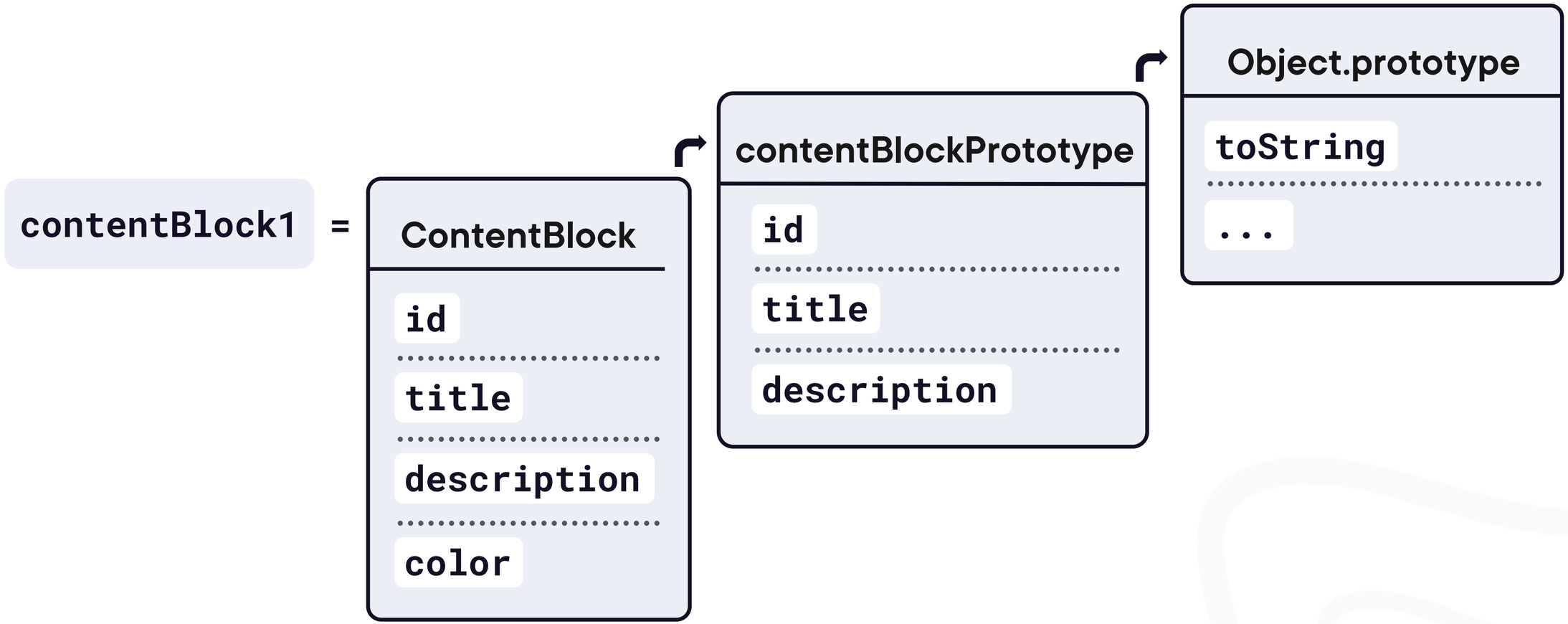
ContentBlock's Prototype



`contentBlock1.description?`

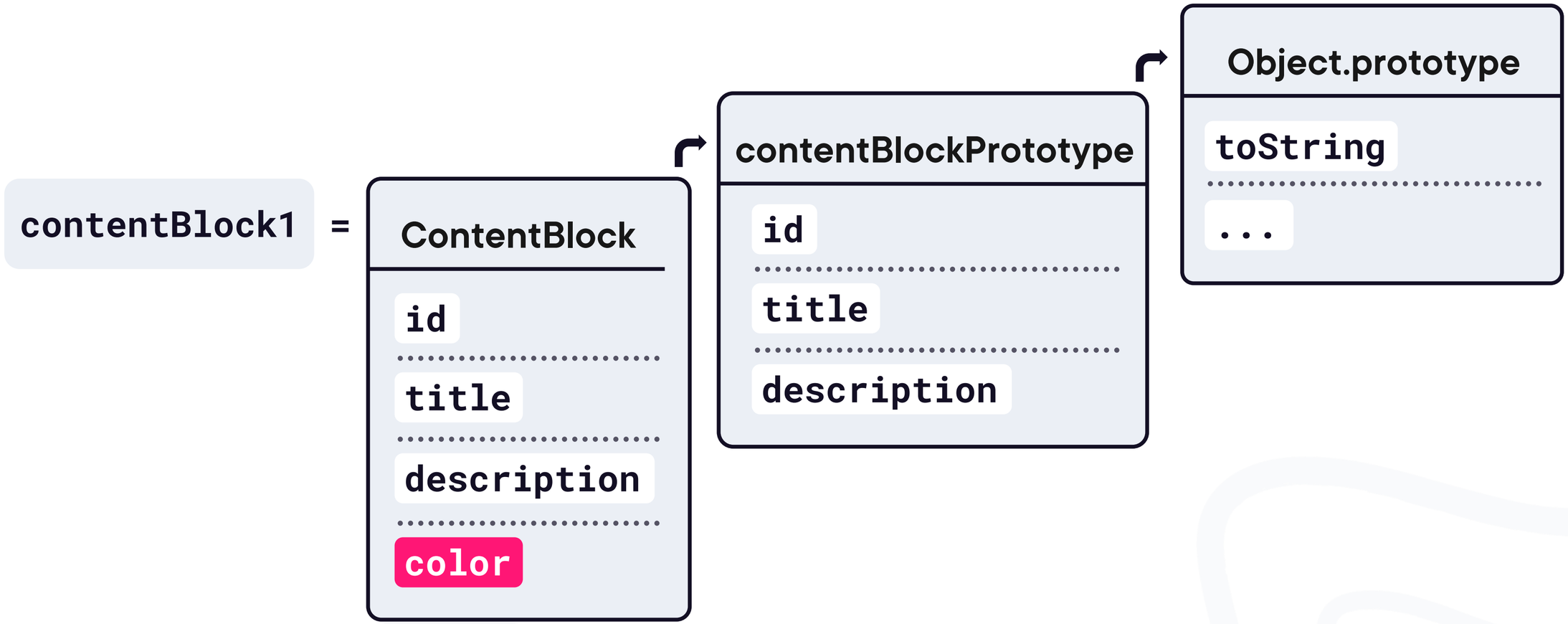
"You will..."

ContentBlock's Prototype




```
contentBlock1.color?
```

ContentBlock's Prototype



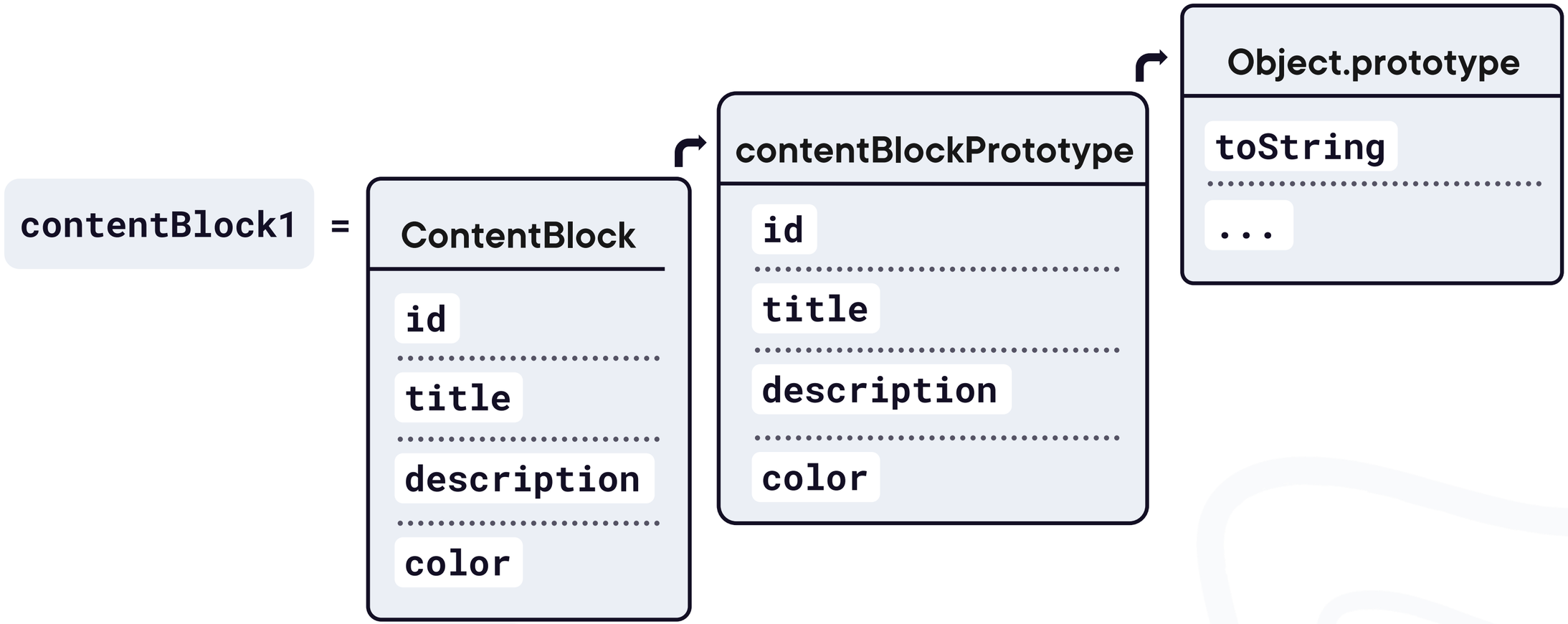
contentBlock1.color?

"#00aced"

Up Next:

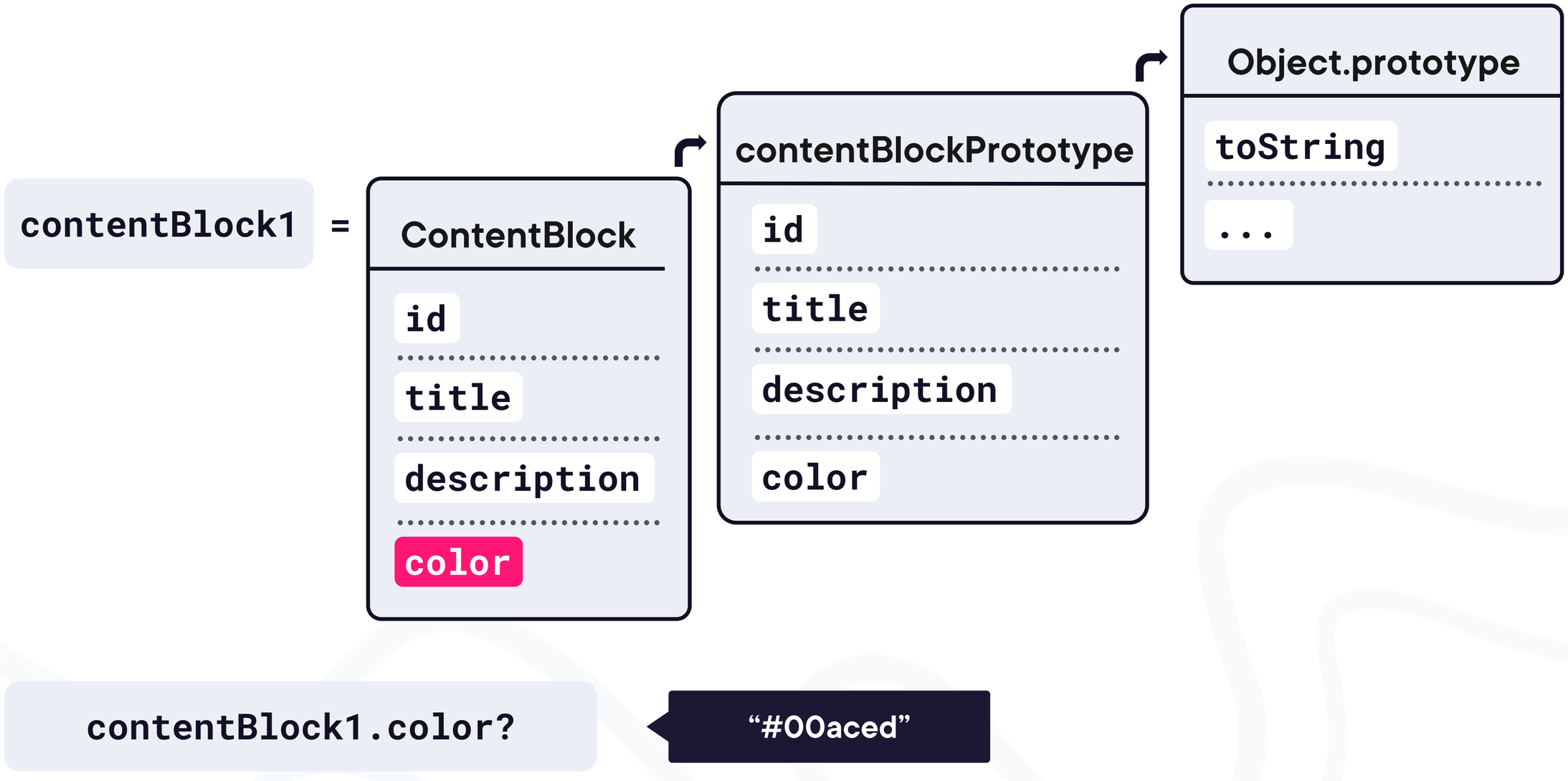
Accessing Properties

ContentBlock's Prototype

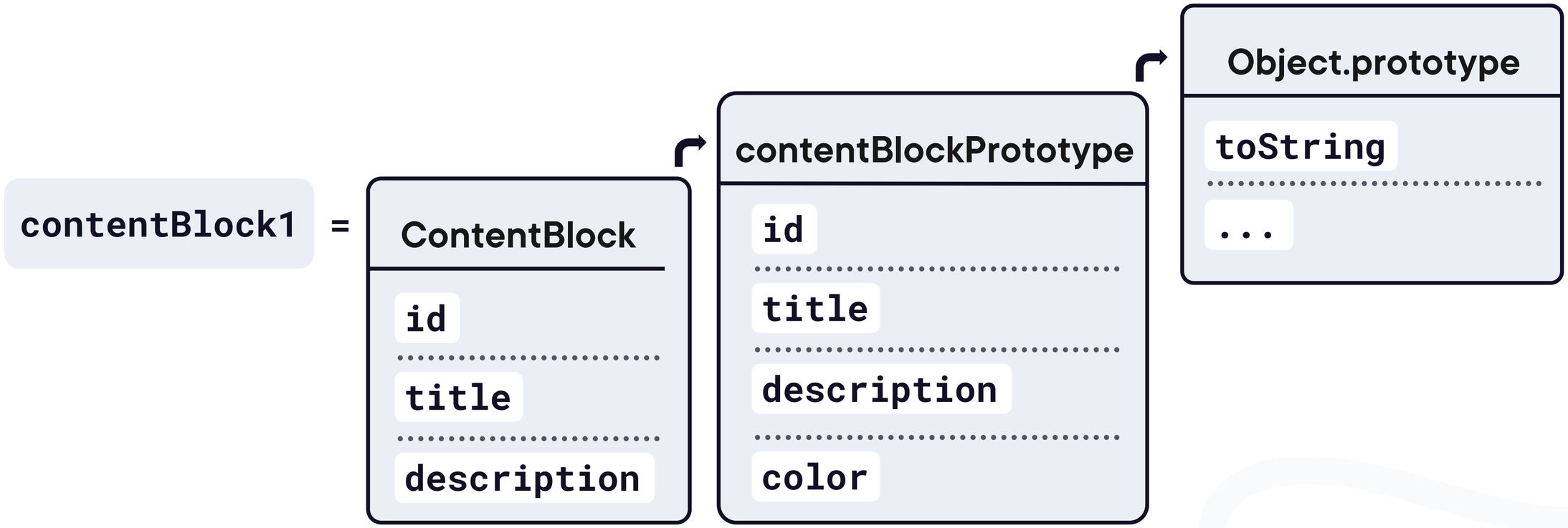



```
contentBlock1.color?
```

ContentBlock's Prototype

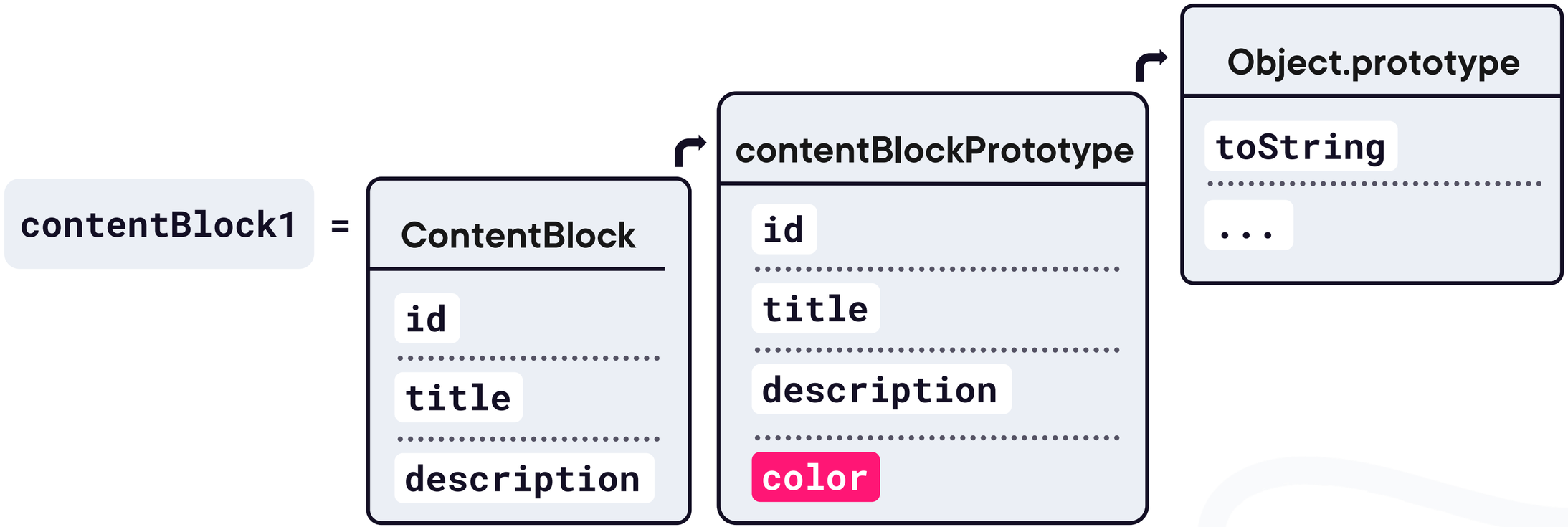


ContentBlock's Prototype




```
contentBlock1.color?
```

ContentBlock's Prototype



`contentBlock1.color?`

"white"

Checking if a Property is Defined on an Object

Check if not undefined

```
if (!object.property == undefined)
```

Use hasOwn

```
if (object.hasOwn("property"))
```

Use “in” operator

```
if ("property" in object)
```

Up Next:

Deleting Properties

What is CRUD?

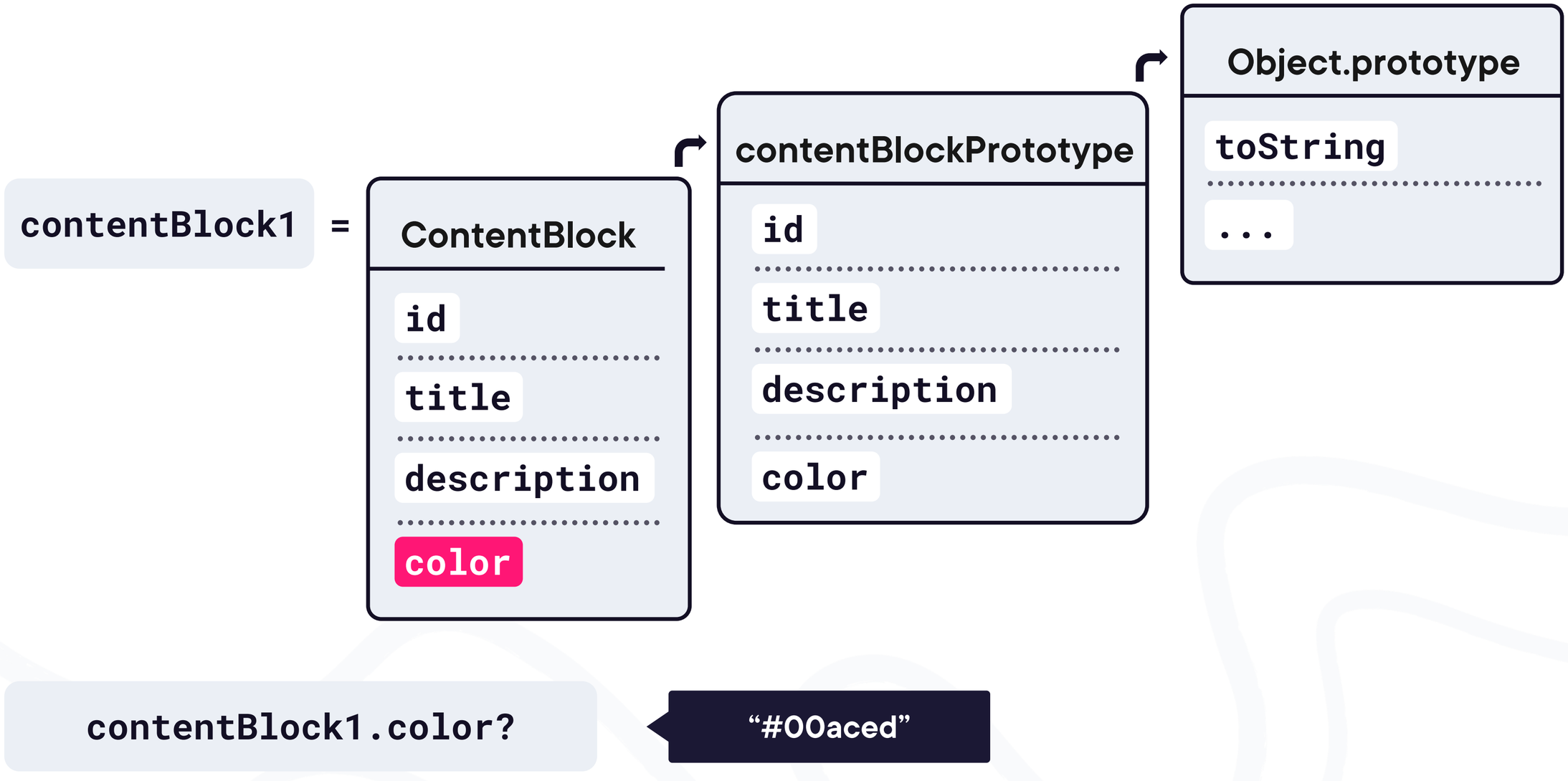
Create

Read

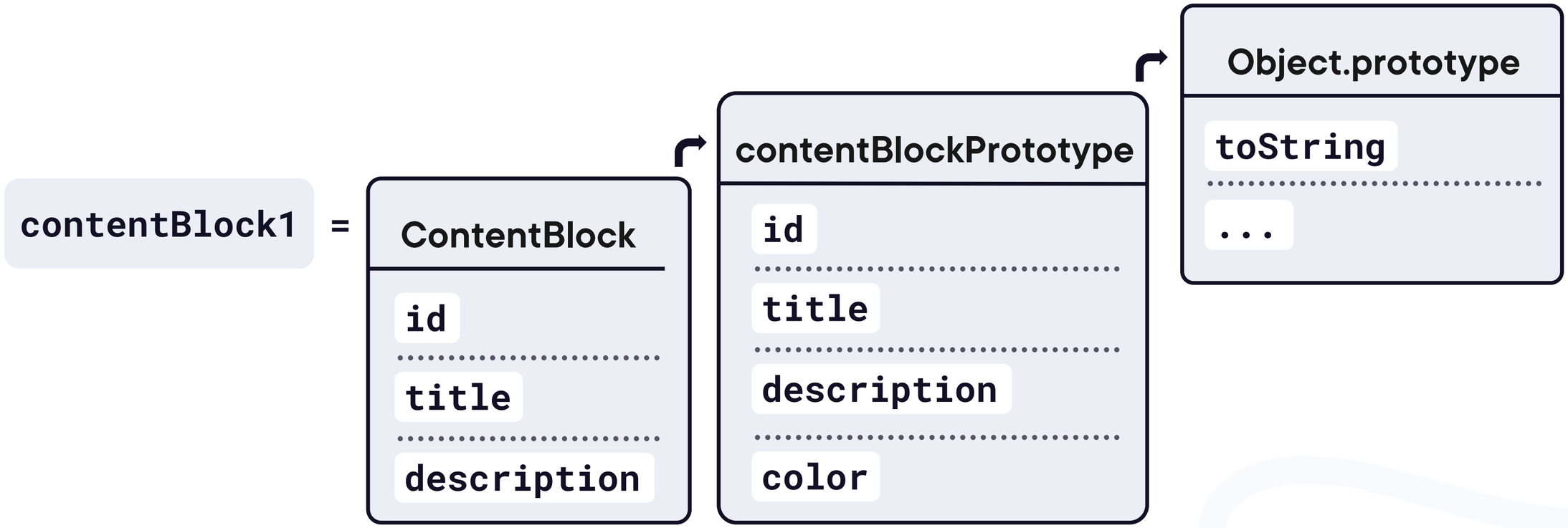
Update

Delete

ContentBlock's Prototype

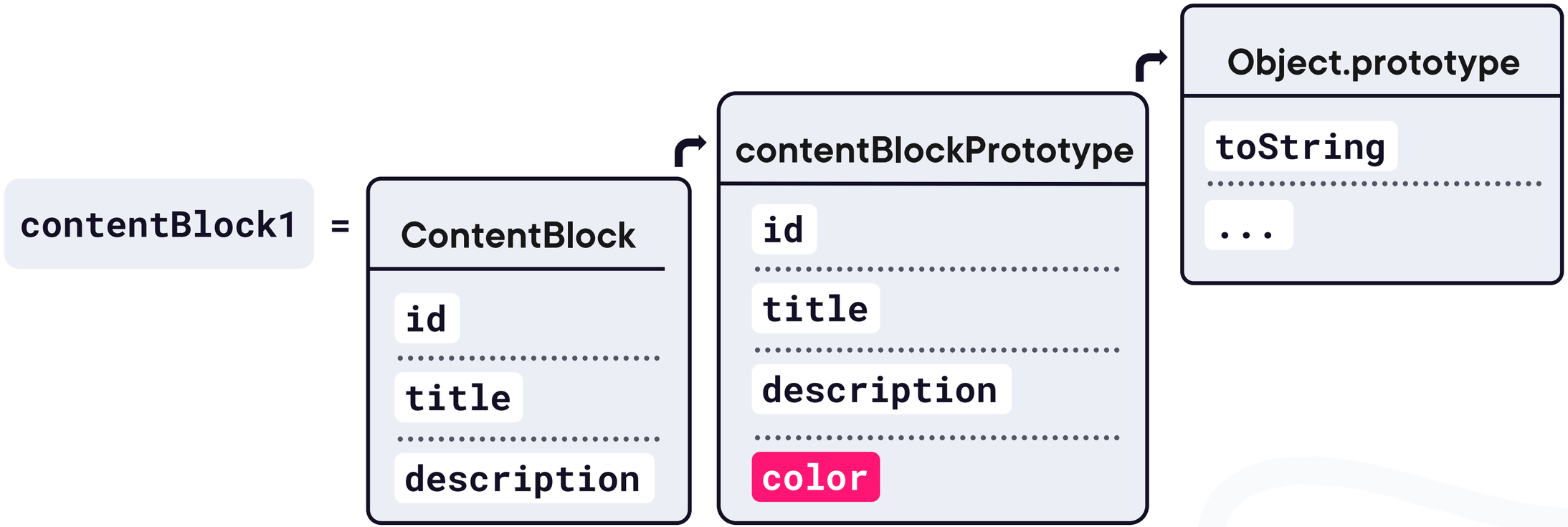


ContentBlock's Prototype




```
contentBlock1.color?
```

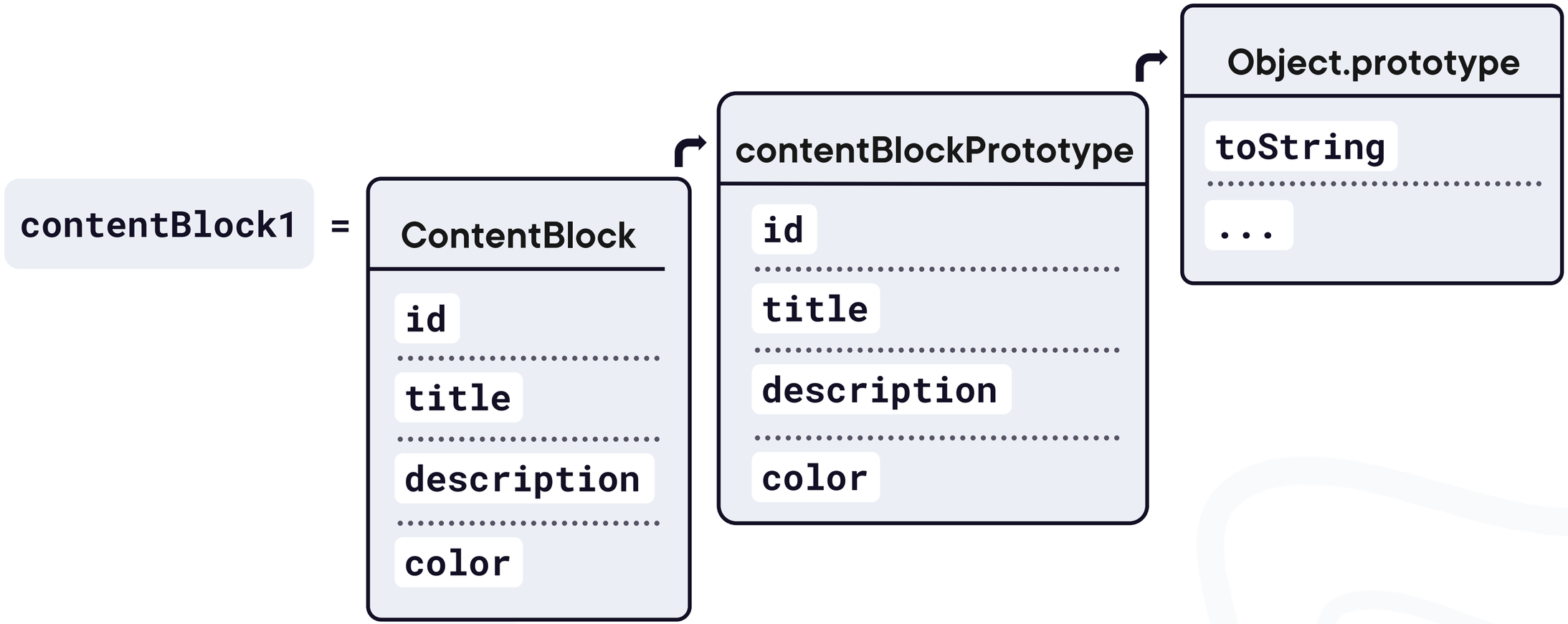
ContentBlock's Prototype



`contentBlock1.color?`

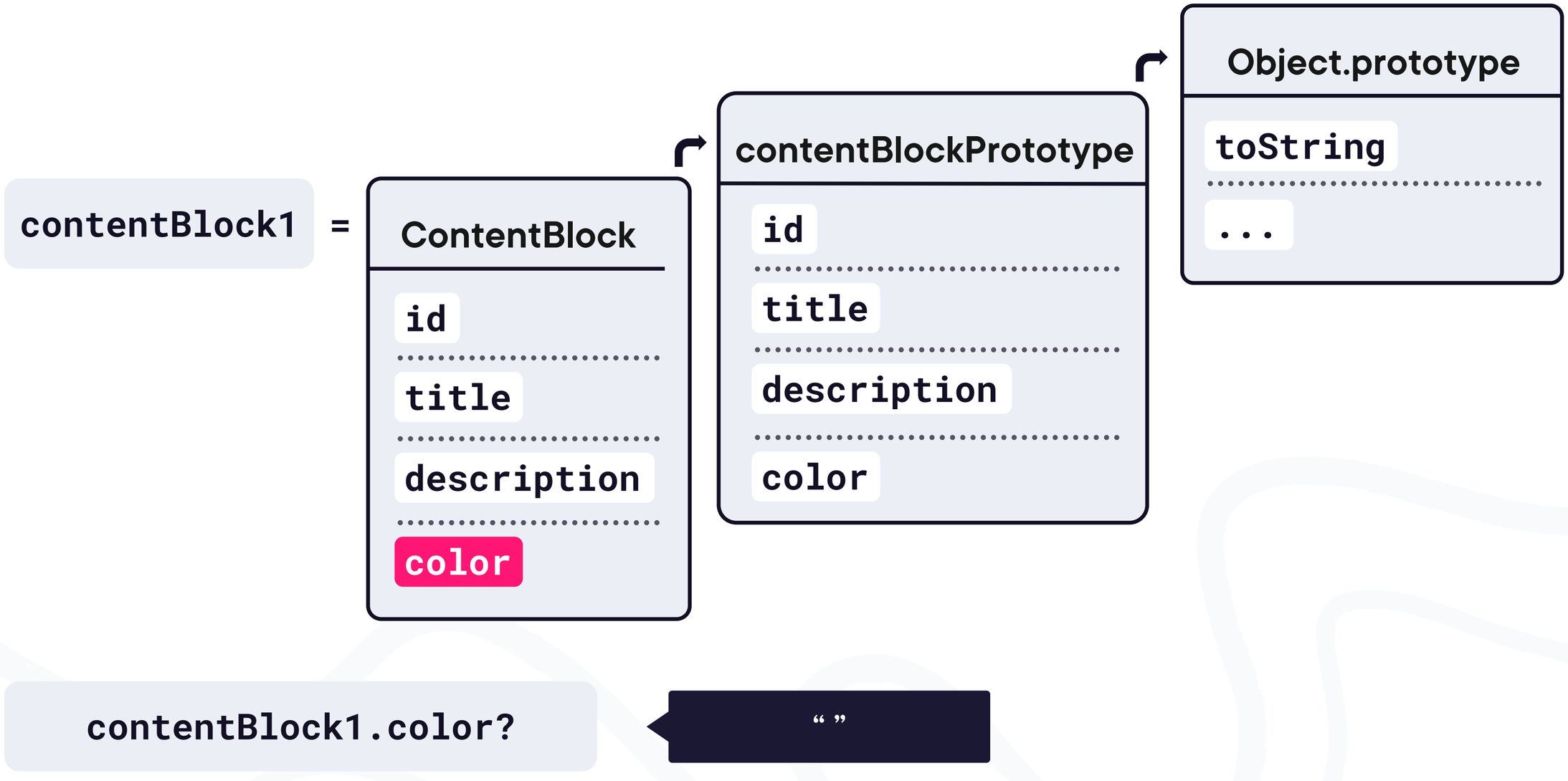
"white"

ContentBlock's Prototype




```
contentBlock1.color?
```

ContentBlock's Prototype

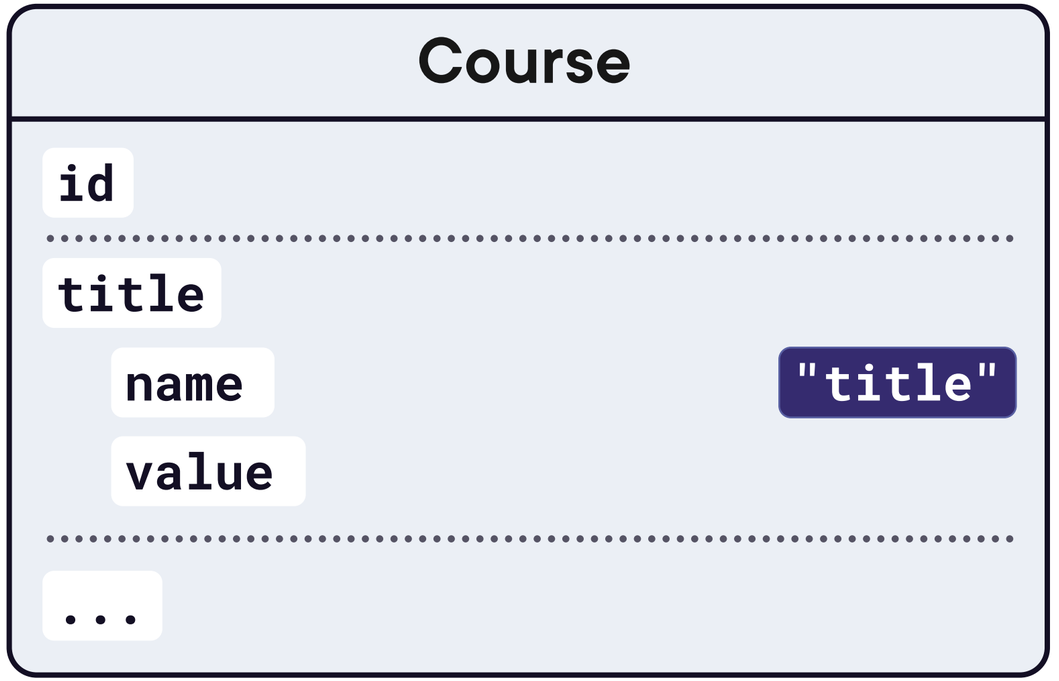


Up Next:

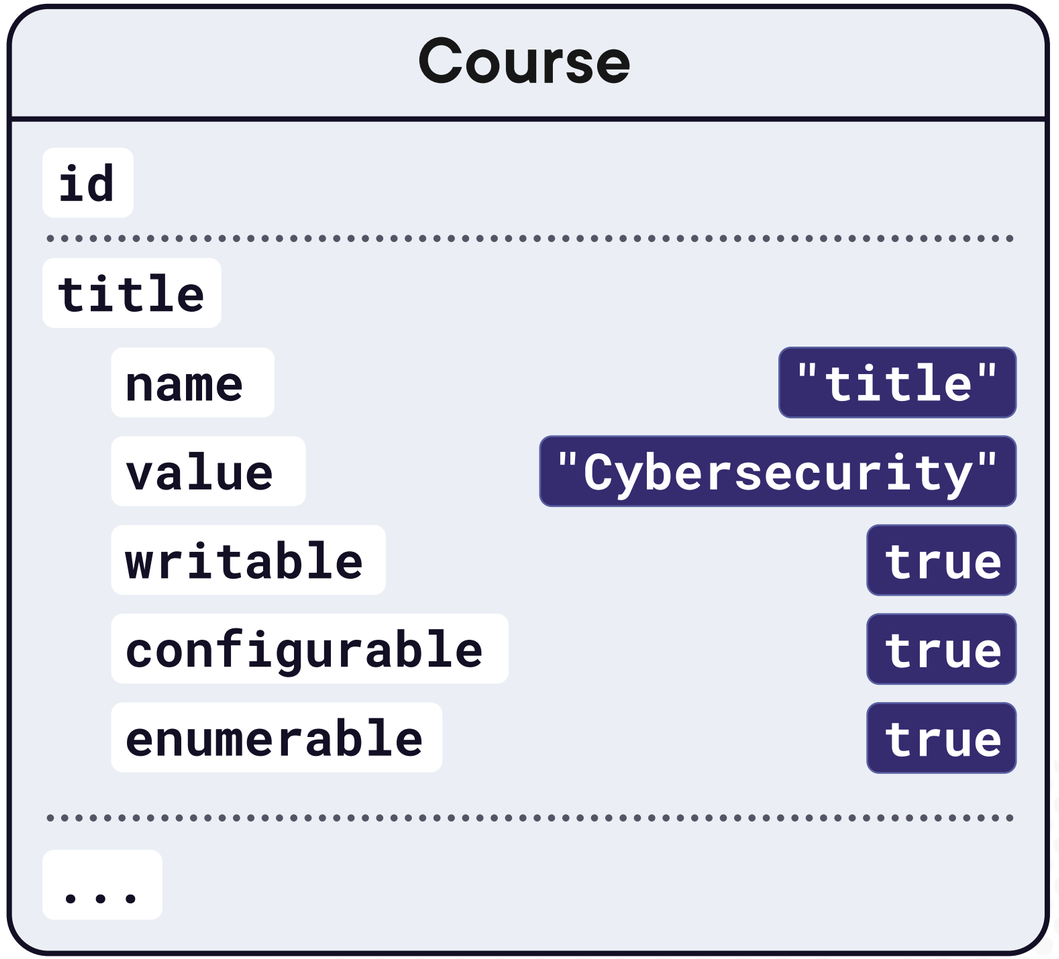
Protecting Properties

Anatomy of a Property

Anatomy of a Property



Anatomy of a Property



Up Next:

Enumerating Properties

Looping Through Enumerable Properties

```
1: id
2: title
3: description
4: color
```

Enumerable

Able to be counted or named, one by one

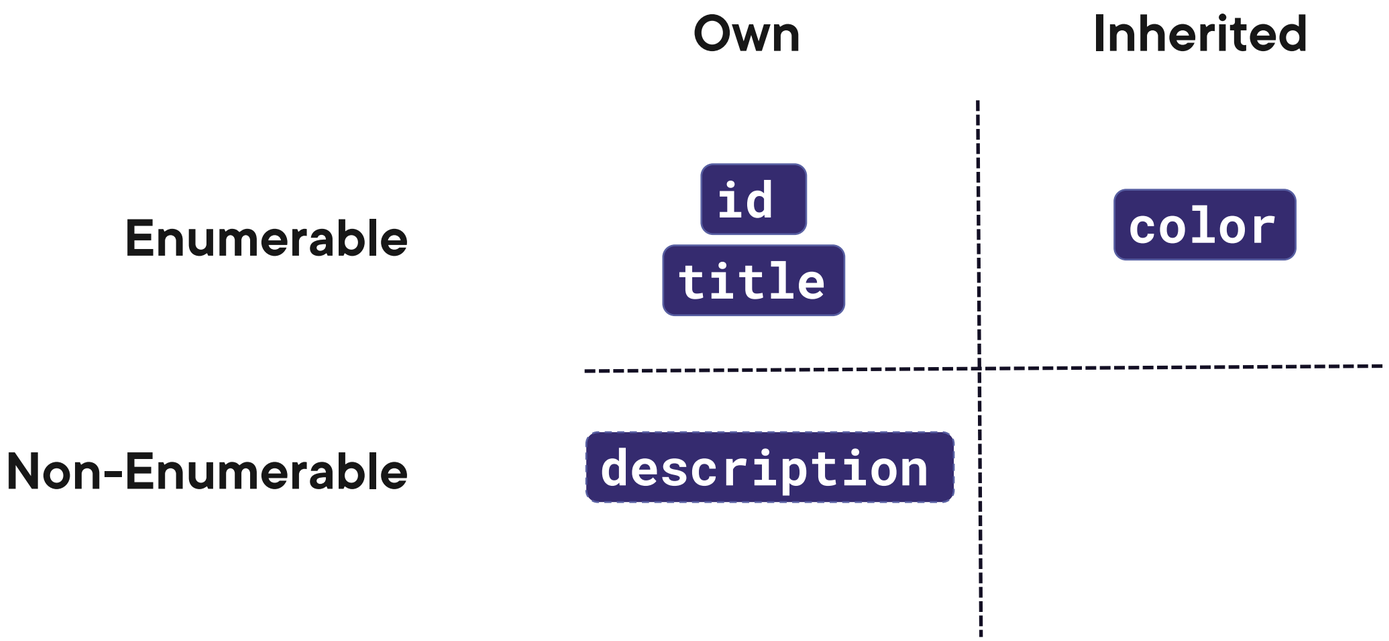
Citation: dictionary.cambridge.org/us/dictionary/english/enumerable

Looping Through Enumerable Properties

Ways to Iterate Through Properties

	for...in	Object.keys	Object.getOwnPropertyNames
Enumerable			
Non-Enumerable			
Own			
Inherited			

The Properties of Our Example Content Block



for...in

1: id

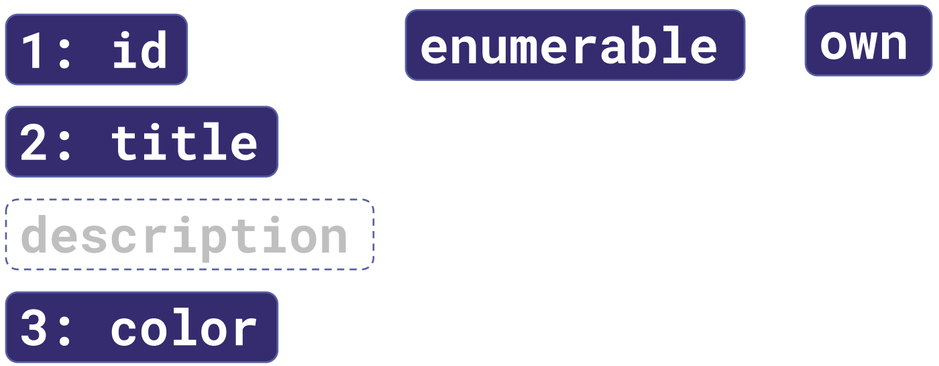
2: title

description

3: color

for...in

All enumerable properties



Object.keys

Only enumerable, own properties

Object .getOwnPropertyNames

Enumerable and non-enumerable own properties

Up Next:

Objects are Reference Types

Ways to Iterate Through Properties

	for...in	Object.keys	Object.getOwnPropertyNames
Enumerable			
Non-Enumerable			
Own			
Inherited			

Ways to Iterate Through Properties

	for...in	Object.keys	Object.getOwnPropertyNames
Enumerable			
Non-Enumerable			
Own			
Inherited			

Ways to Iterate Through Properties

	for...in	Object.keys	Object.getOwnPropertyNames
Enumerable			
Non-Enumerable			
Own			
Inherited			

Ways to Iterate Through Properties

	for...in	Object.keys	Object.getOwnPropertyNames
Enumerable			
Non-Enumerable			
Own			
Inherited