Object References and Scope

Matt Henry
VP of Product & Independent Developer

fancymatt.com

const vs let

const

```
const myNum = 1.138;
```

Can't be redefined later

VS

let

```
let myNum = 1.138;
```

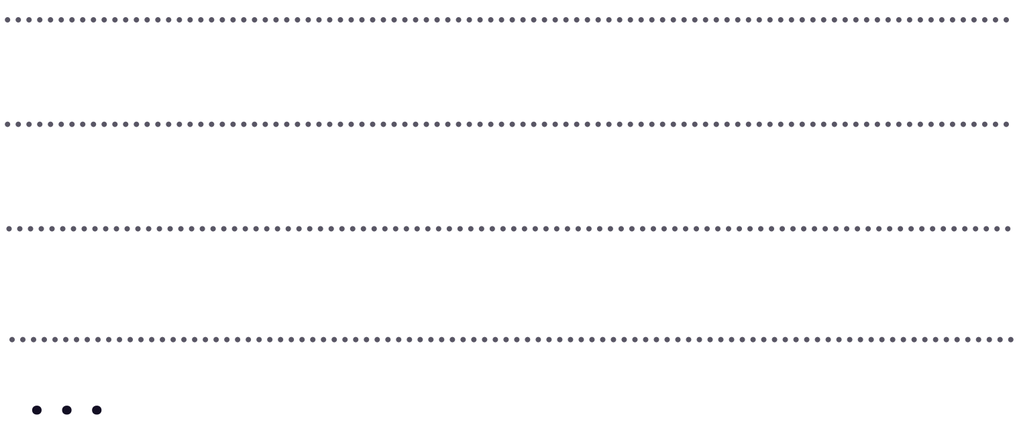
Can be redefined later

Objects are reference types

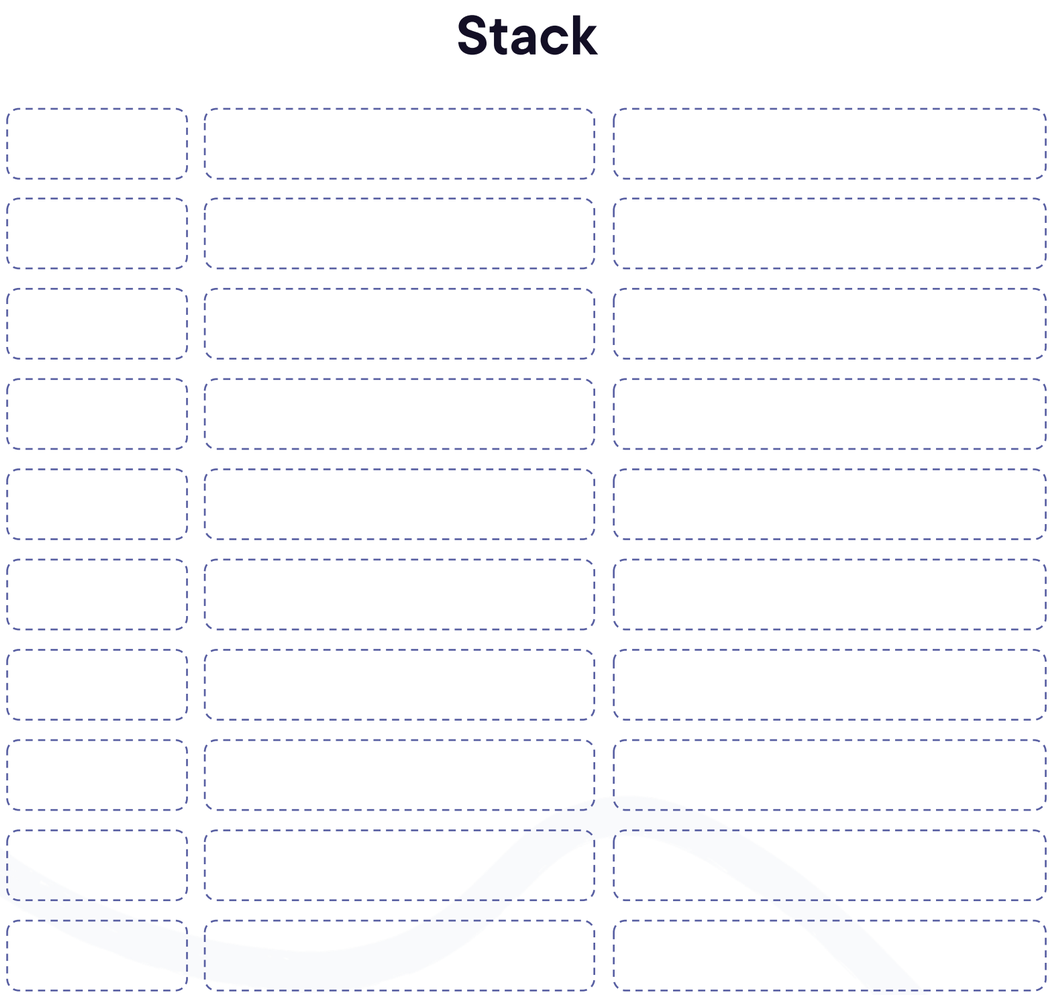
Stack vs. Heap

Stack

Heap



Stack vs. Heap



Stack vs. Heap

Stack

Heap

...

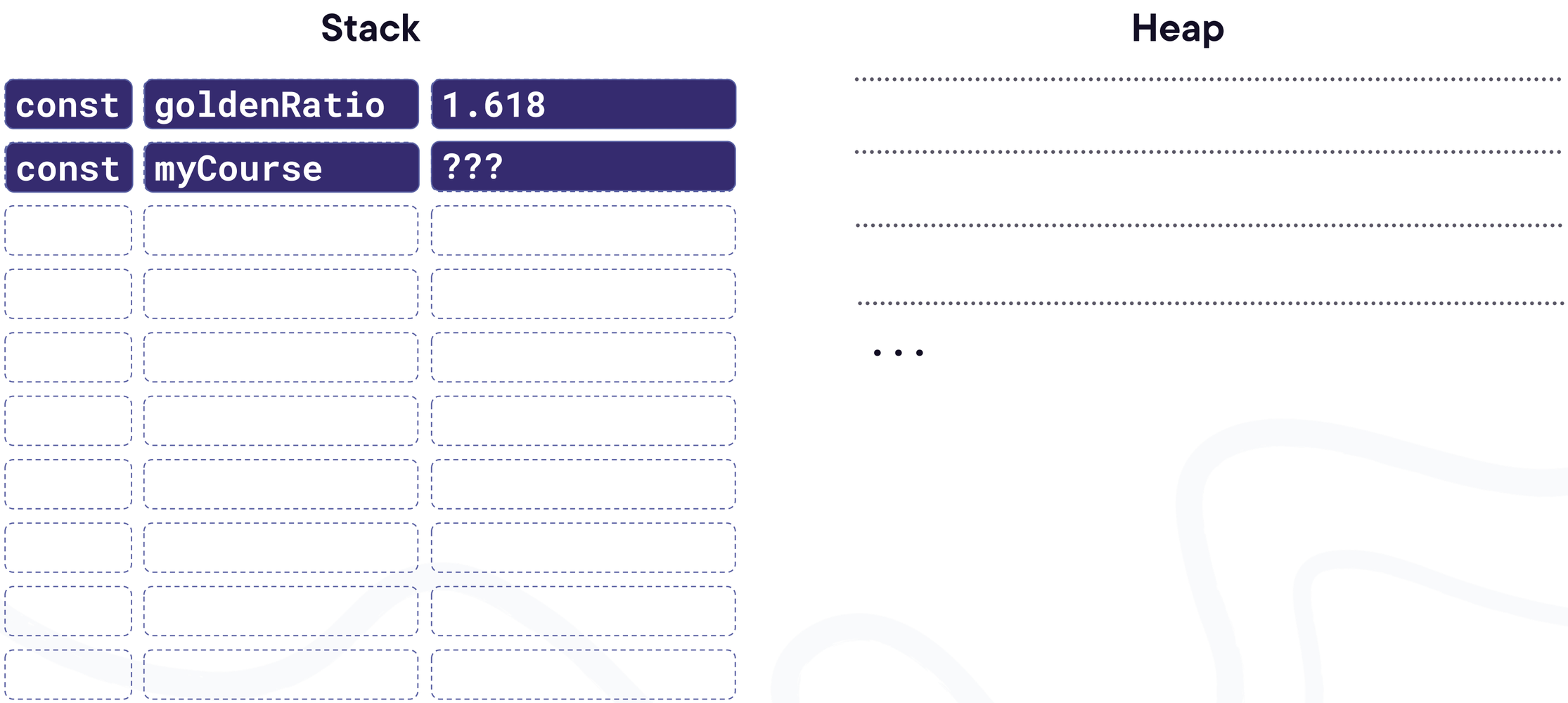
Stack vs. Heap

Stack

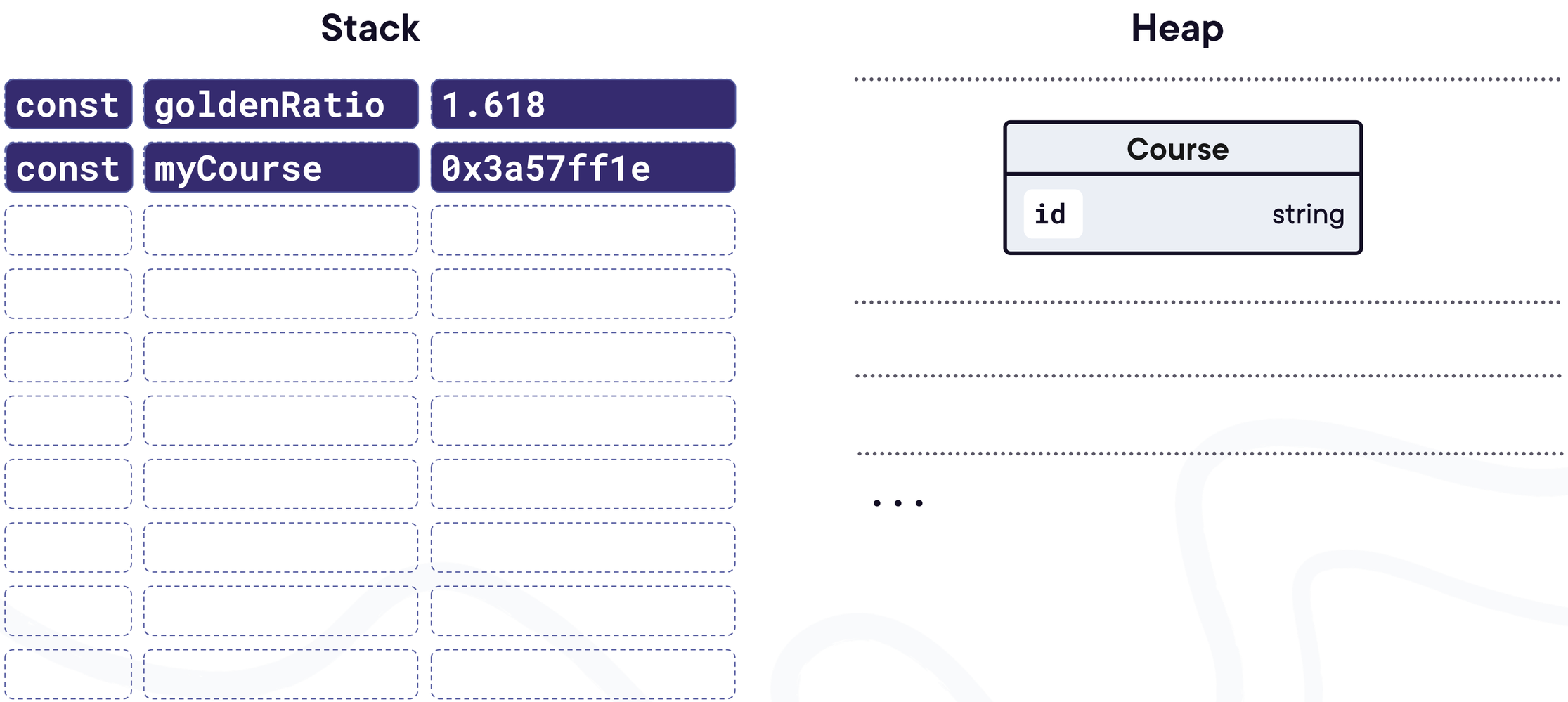
Heap

Stack vs. Heap

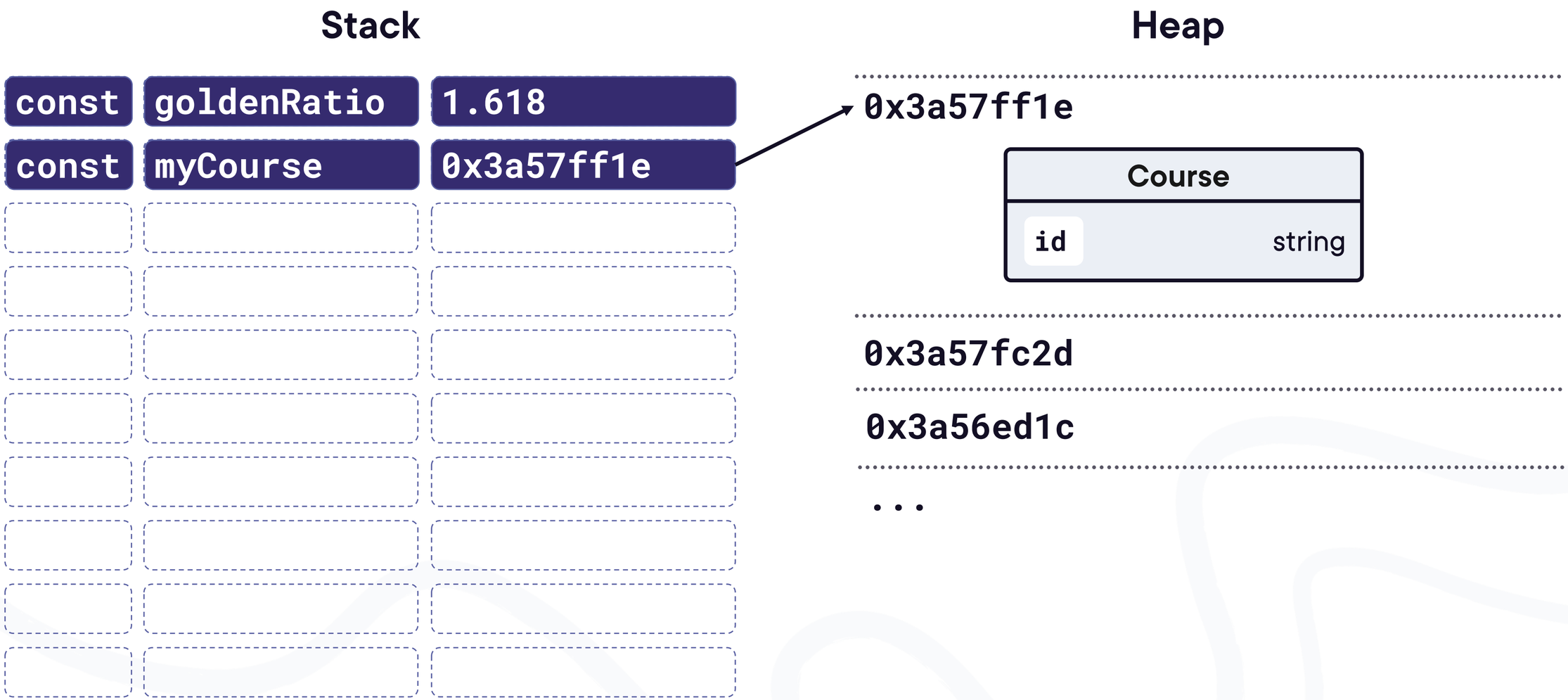
Stack vs. Heap



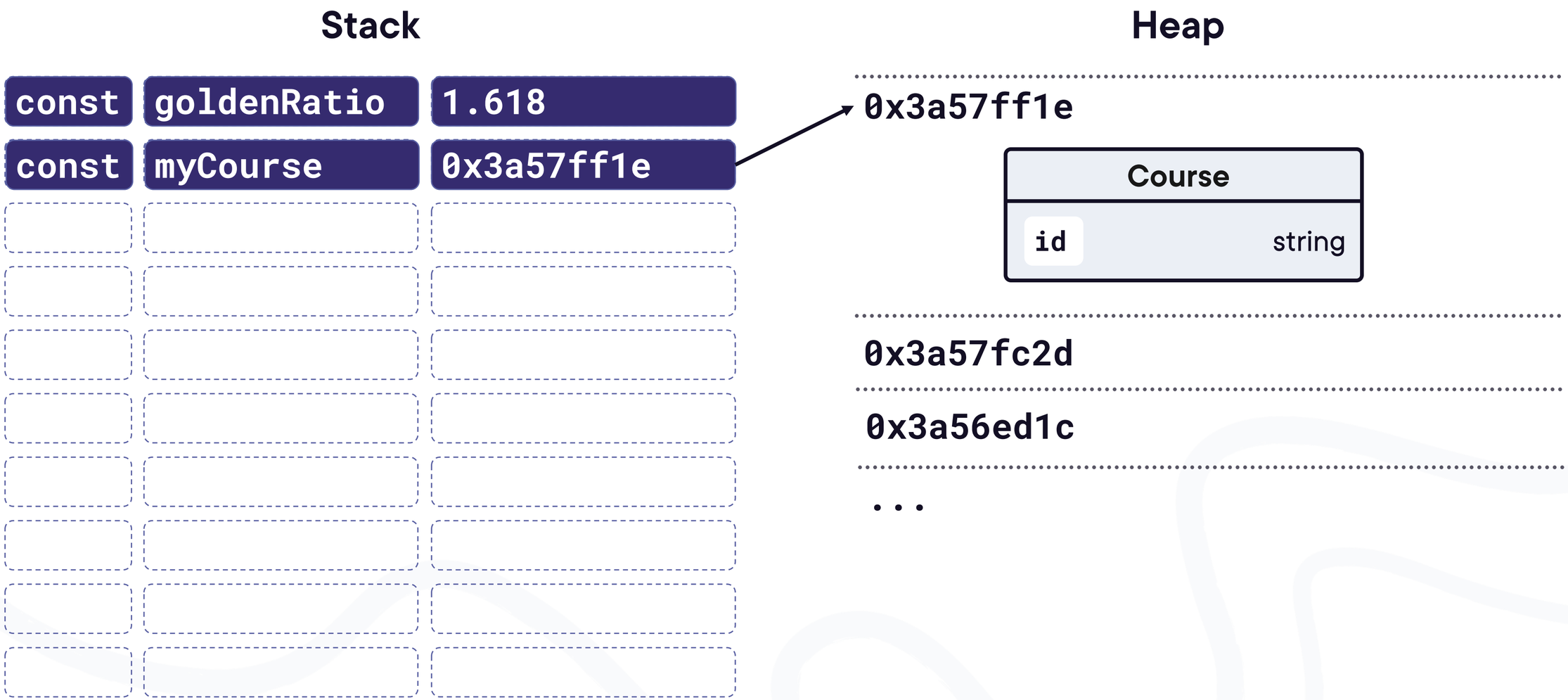
Stack vs. Heap



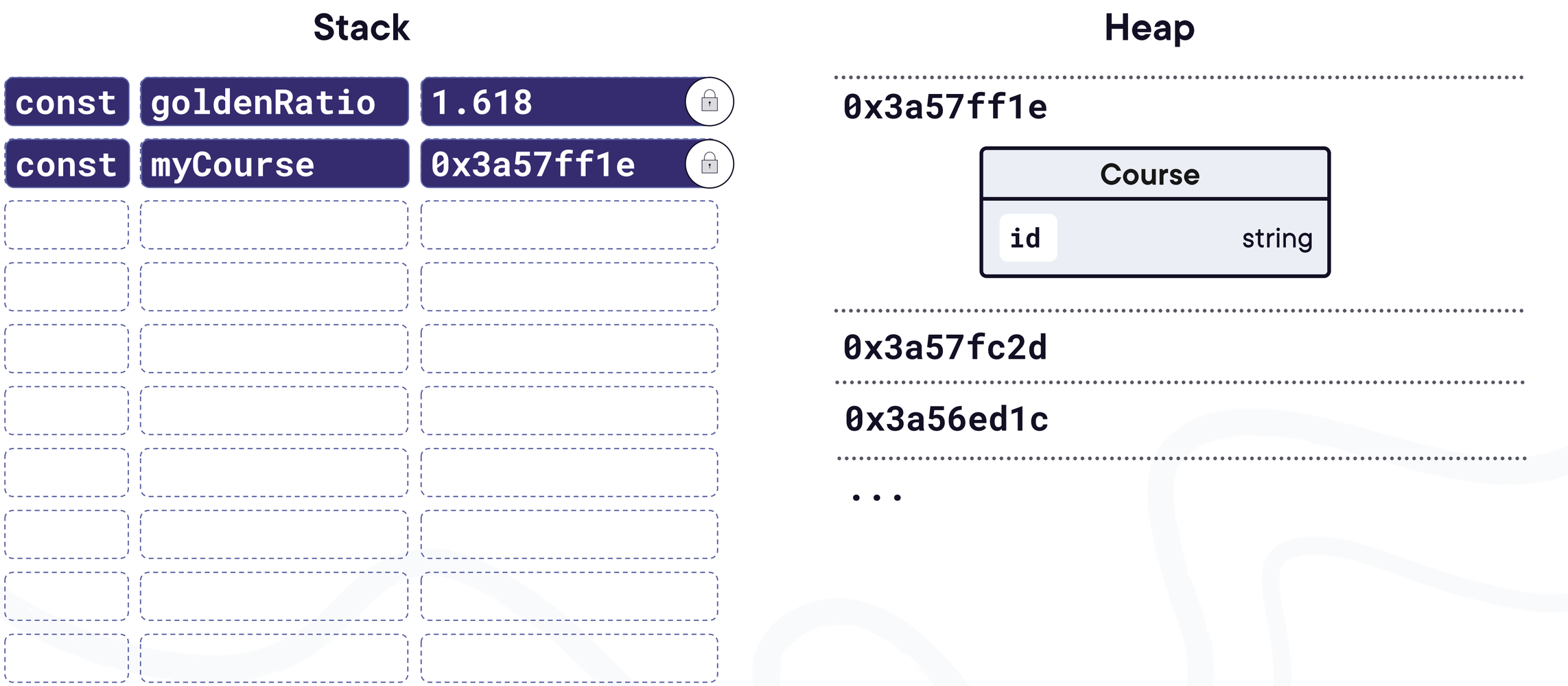
Stack vs. Heap



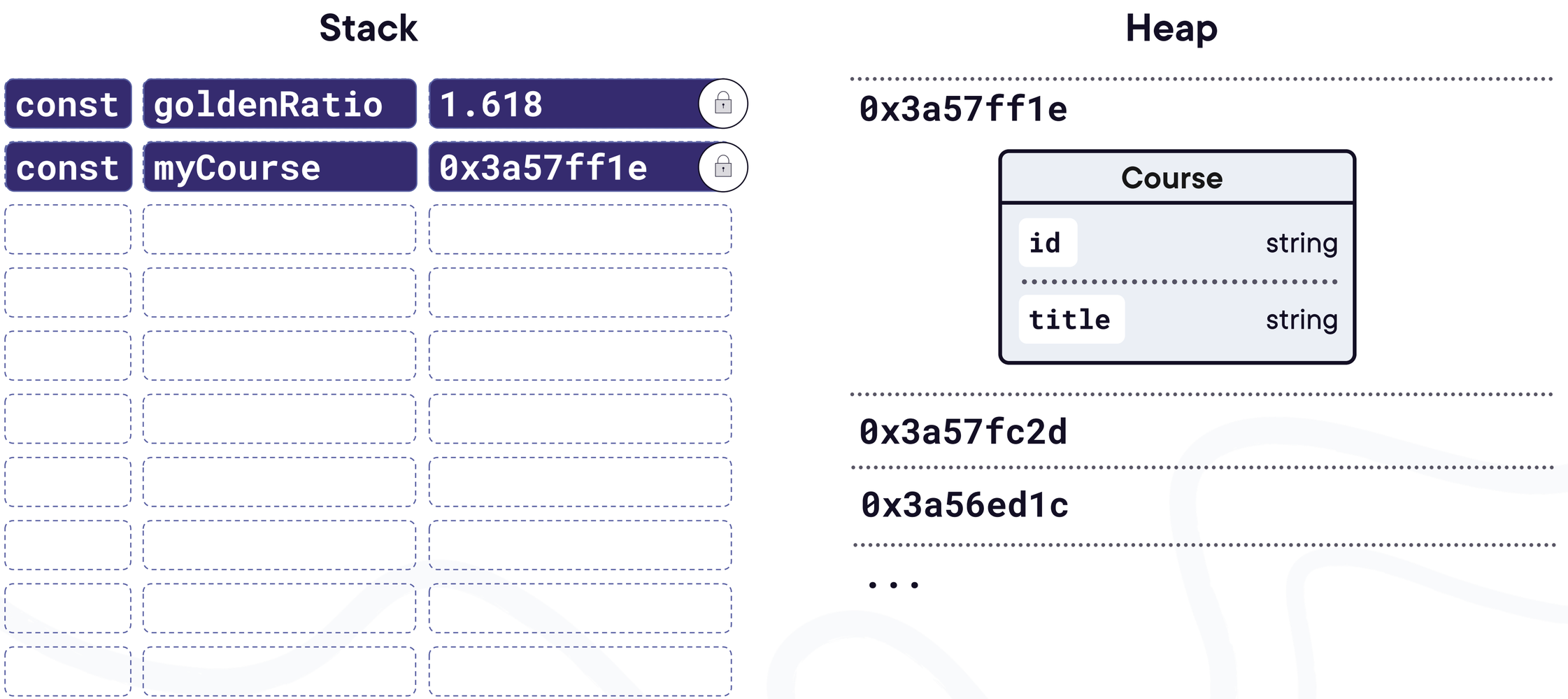
Stack vs. Heap



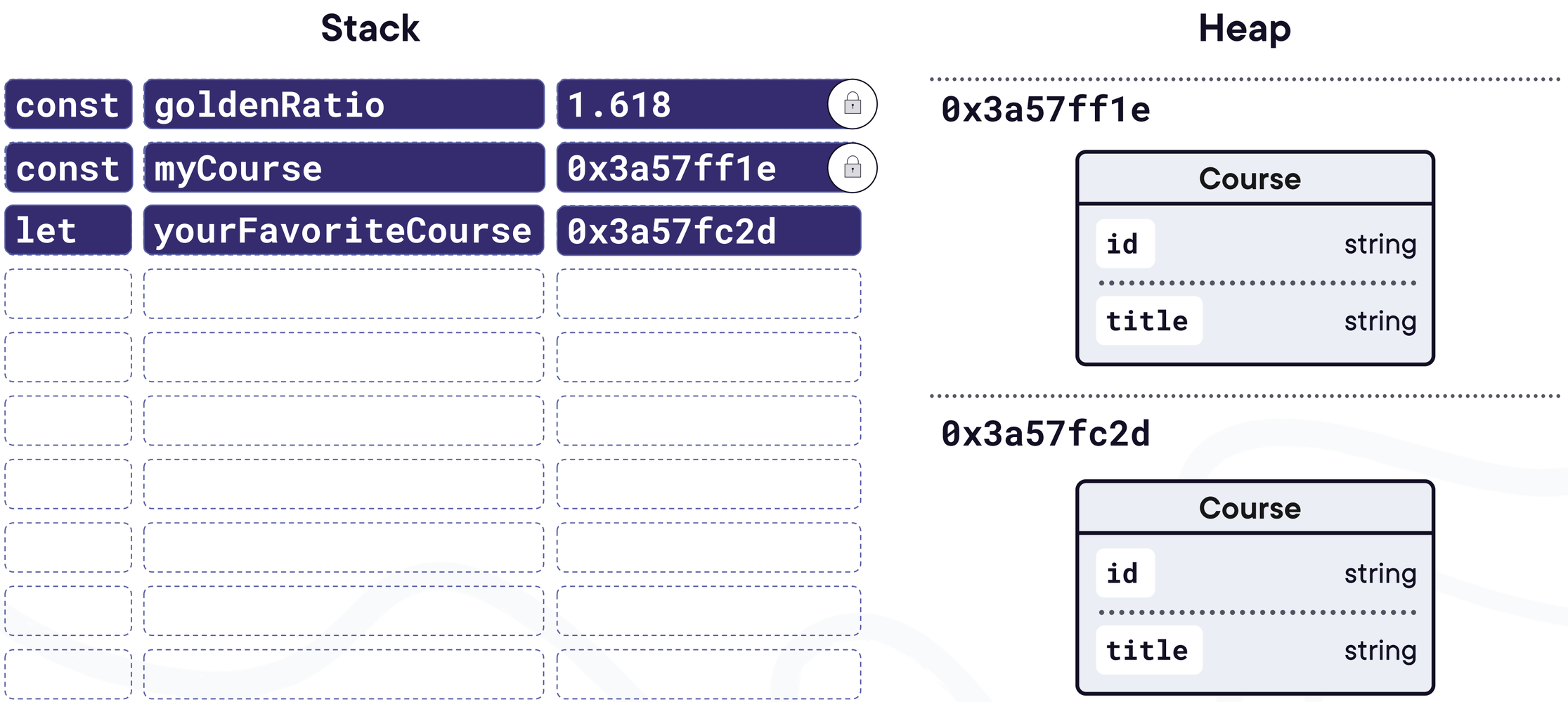
Stack vs. Heap



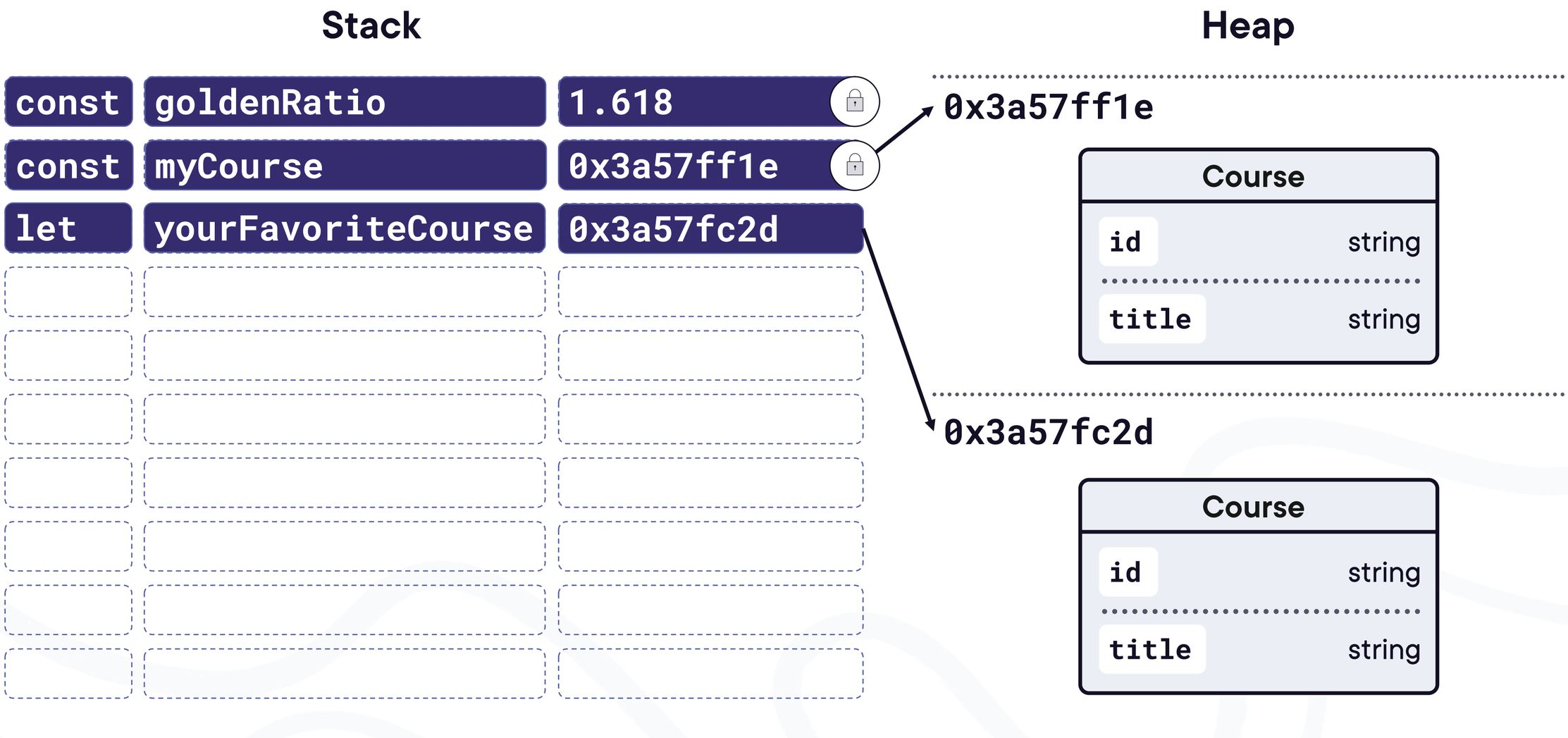
Stack vs. Heap



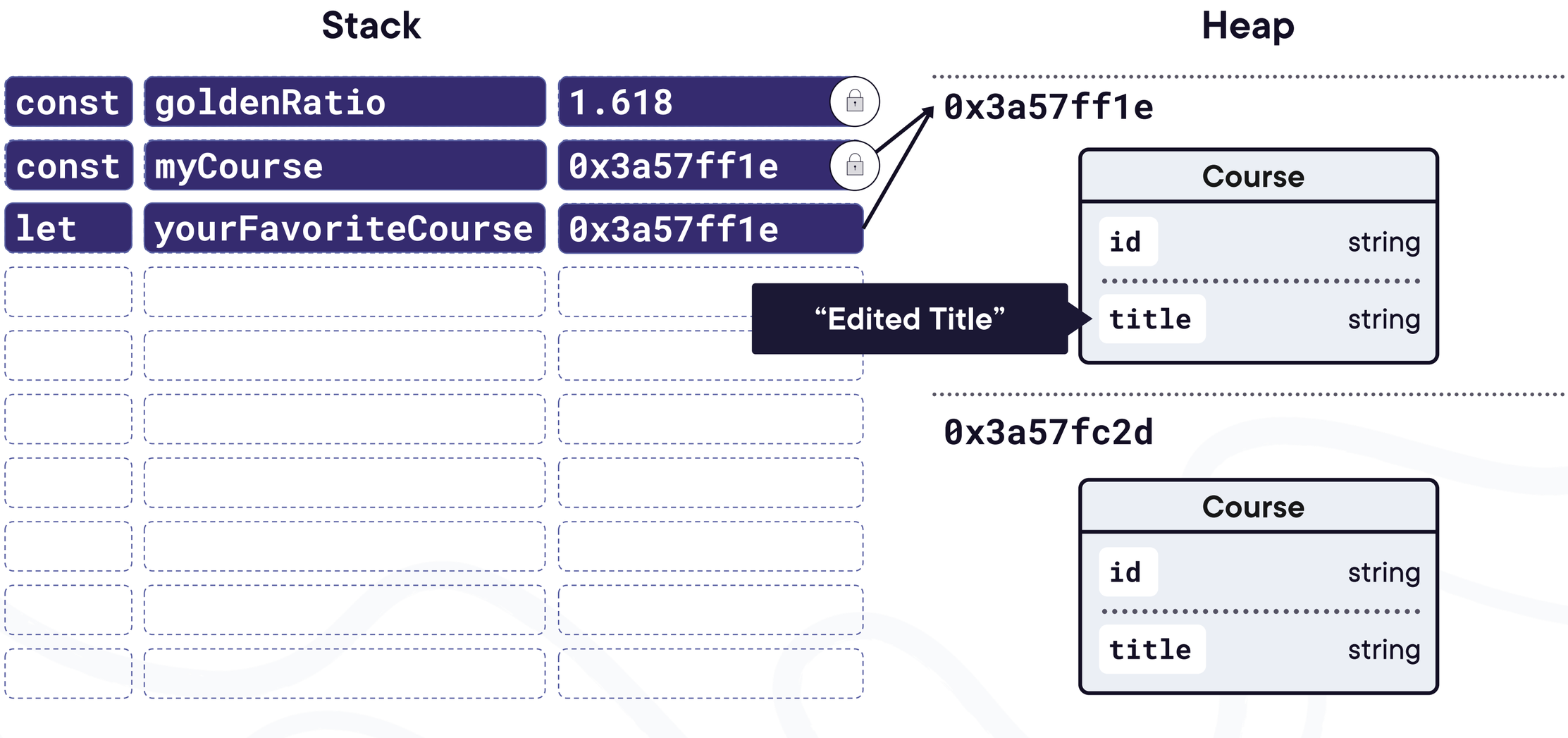
Stack vs. Heap



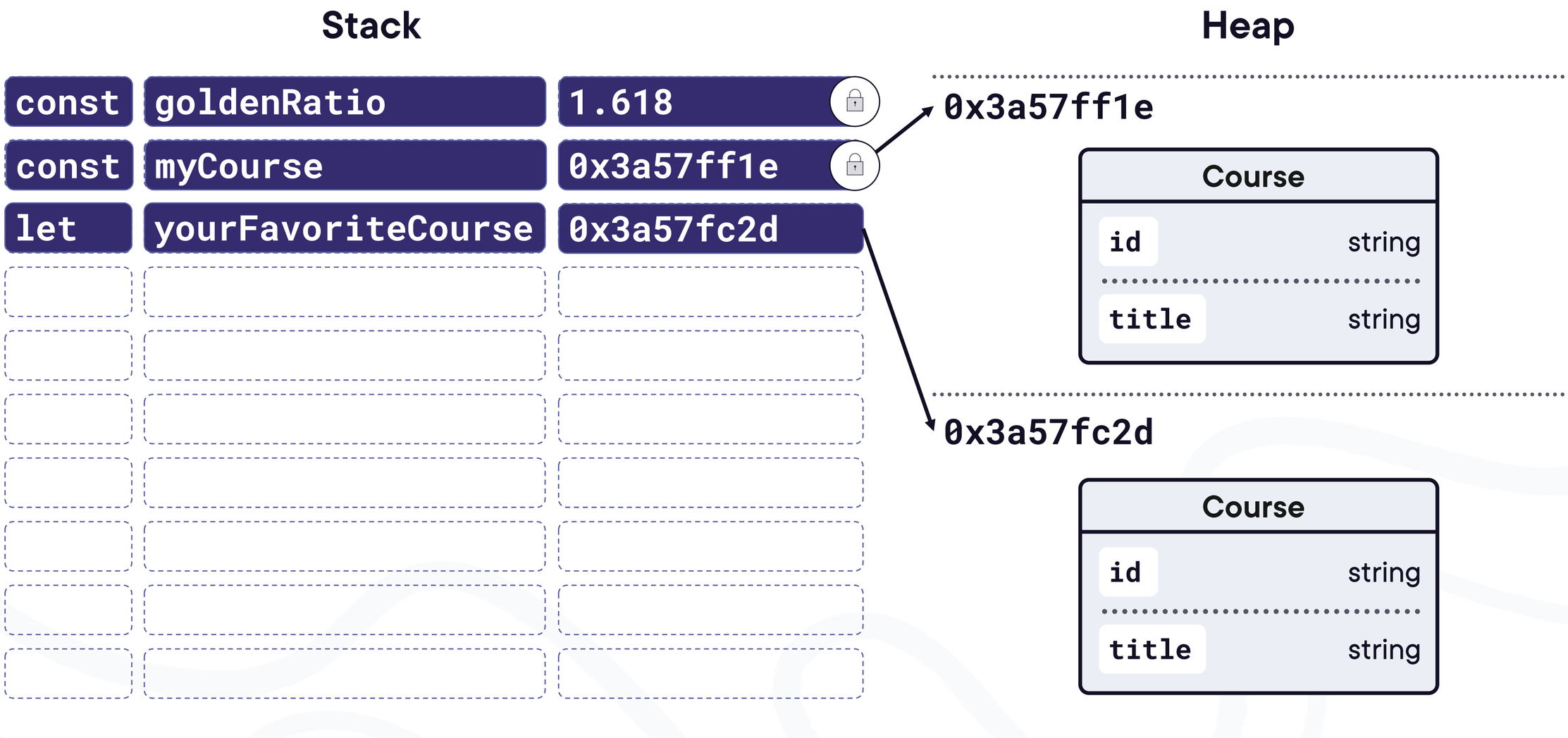
Stack vs. Heap



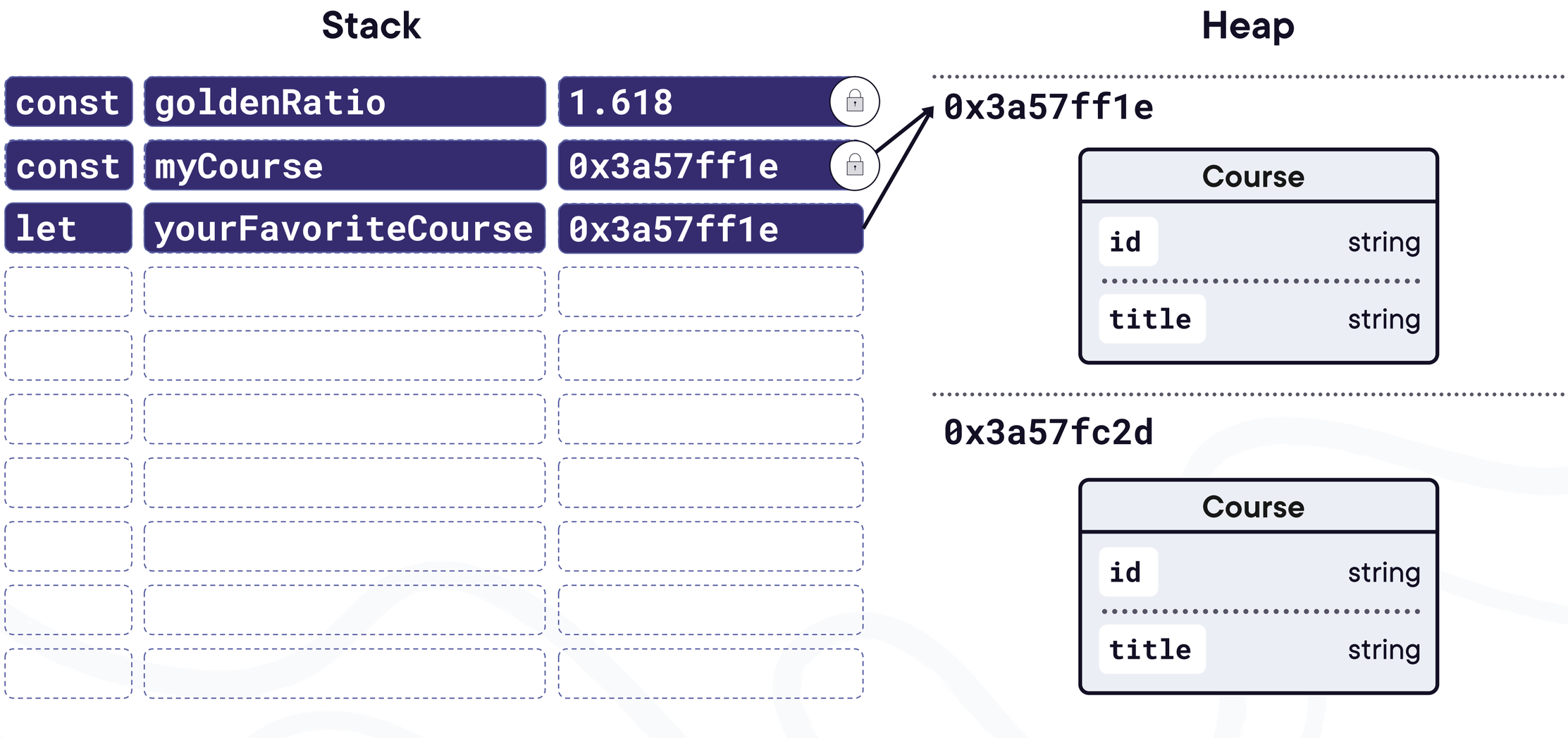
Stack vs. Heap



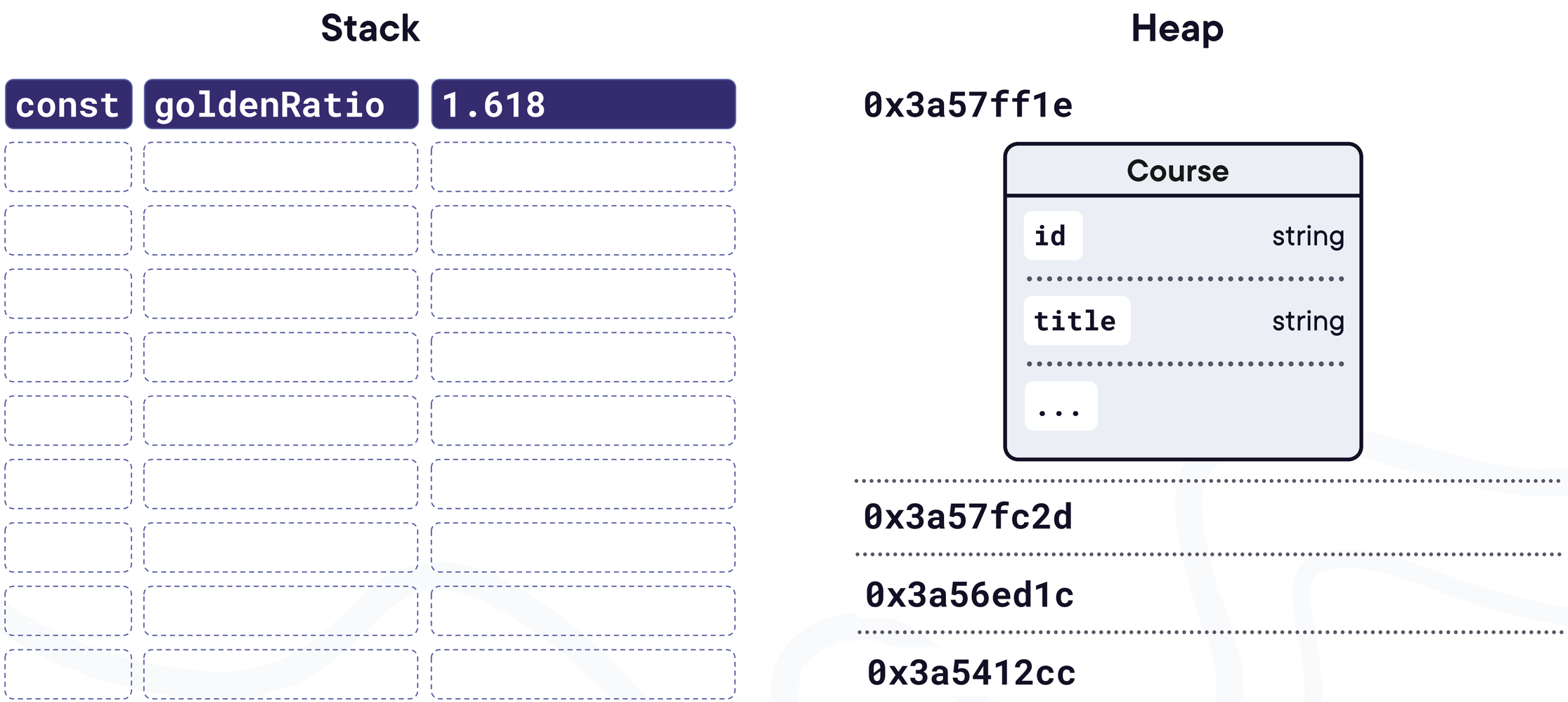
Stack vs. Heap



Stack vs. Heap



Stack vs. Heap



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

Up Next:

Comparing and Cloning Objects

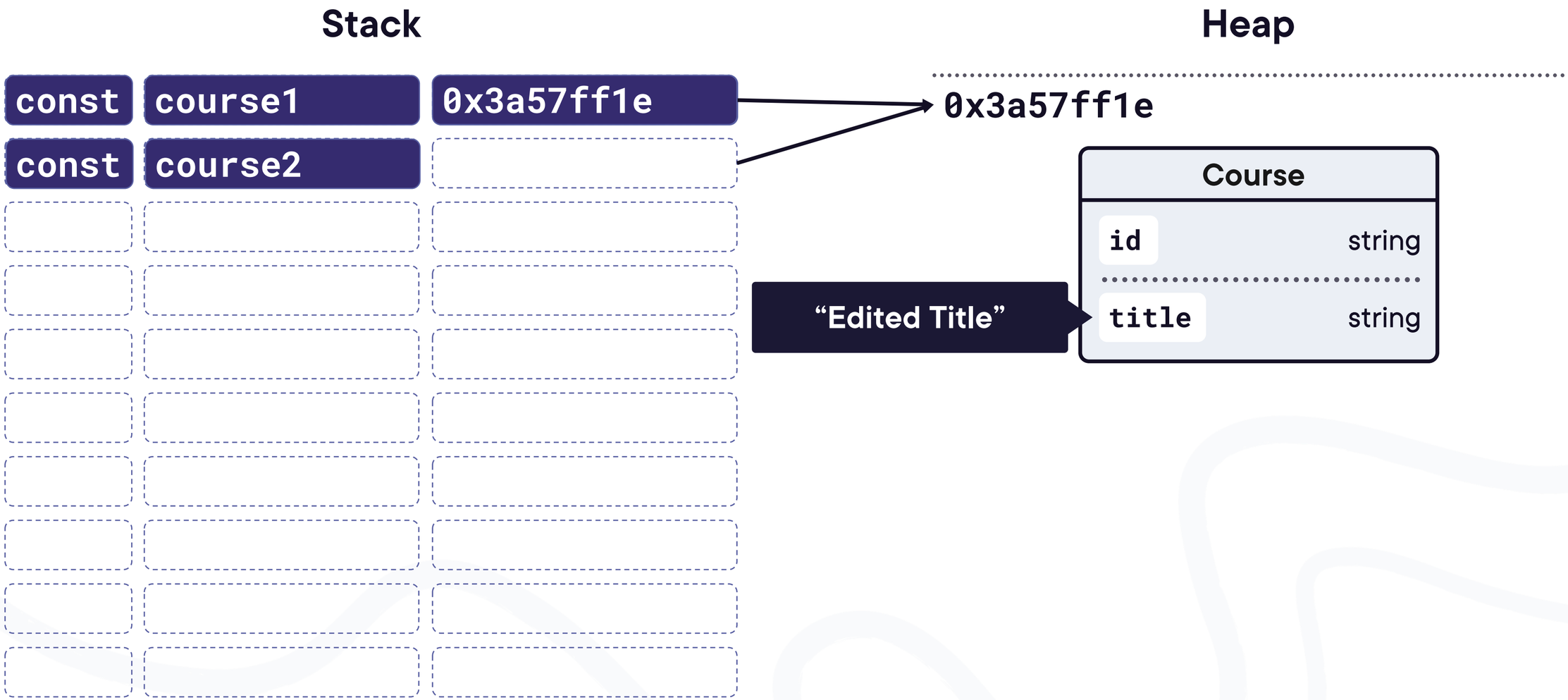
Reference Types: Challenges

Cloning

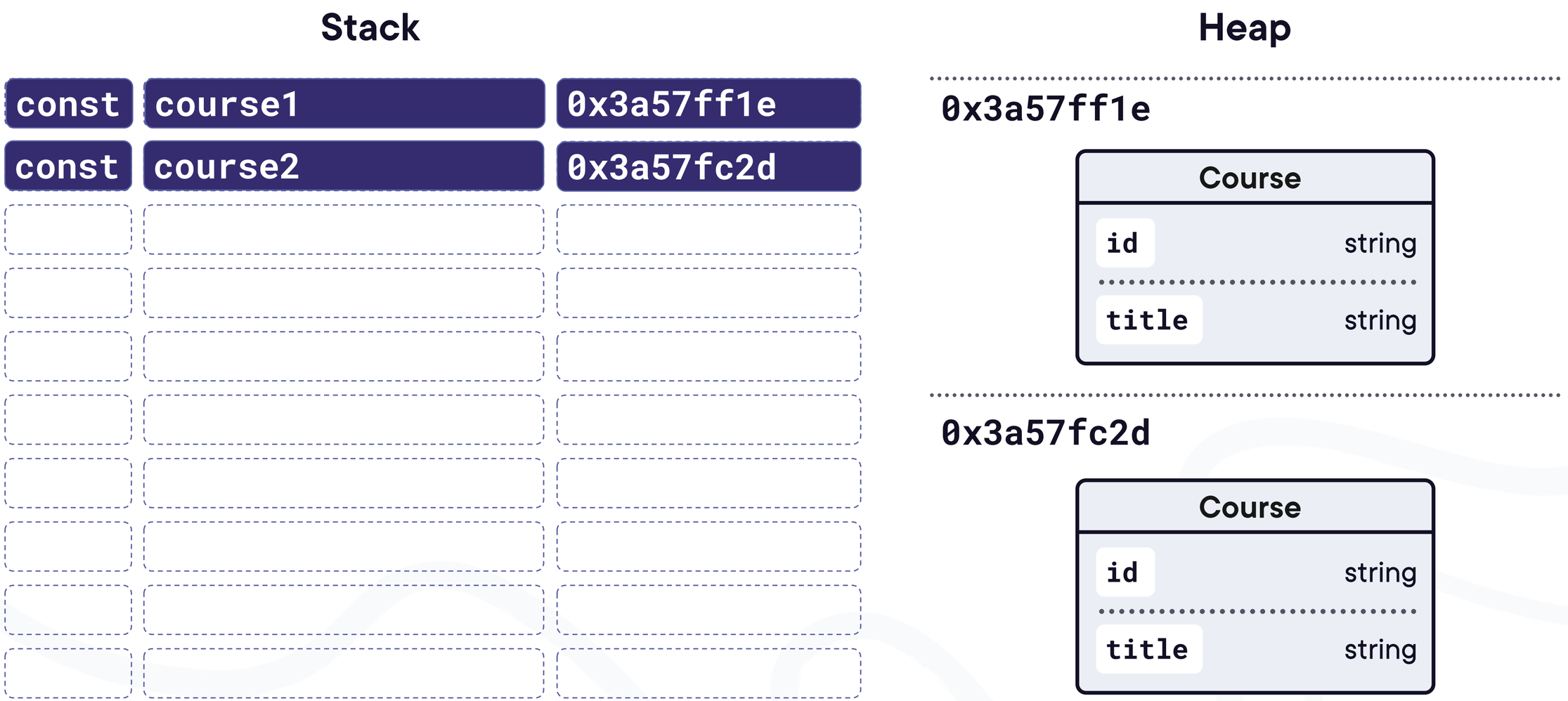
Comparing

Stack vs. Heap

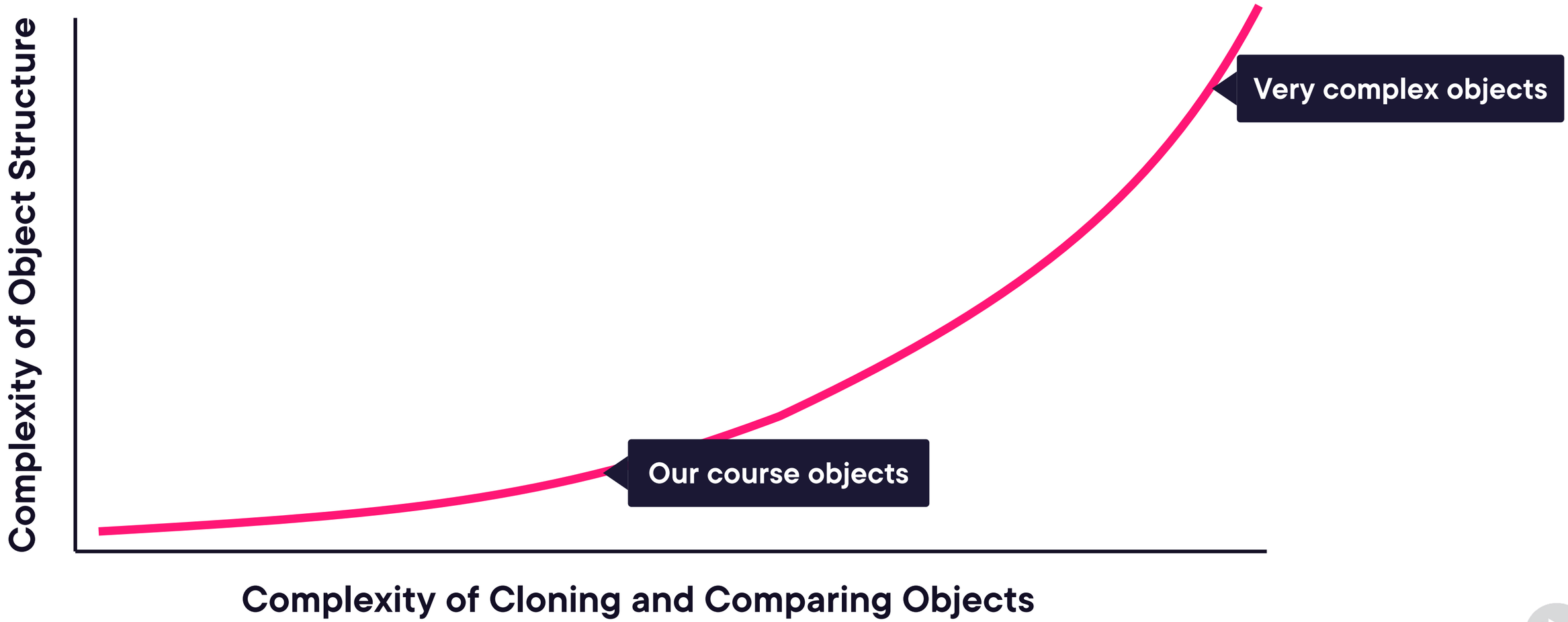
Stack vs. Heap



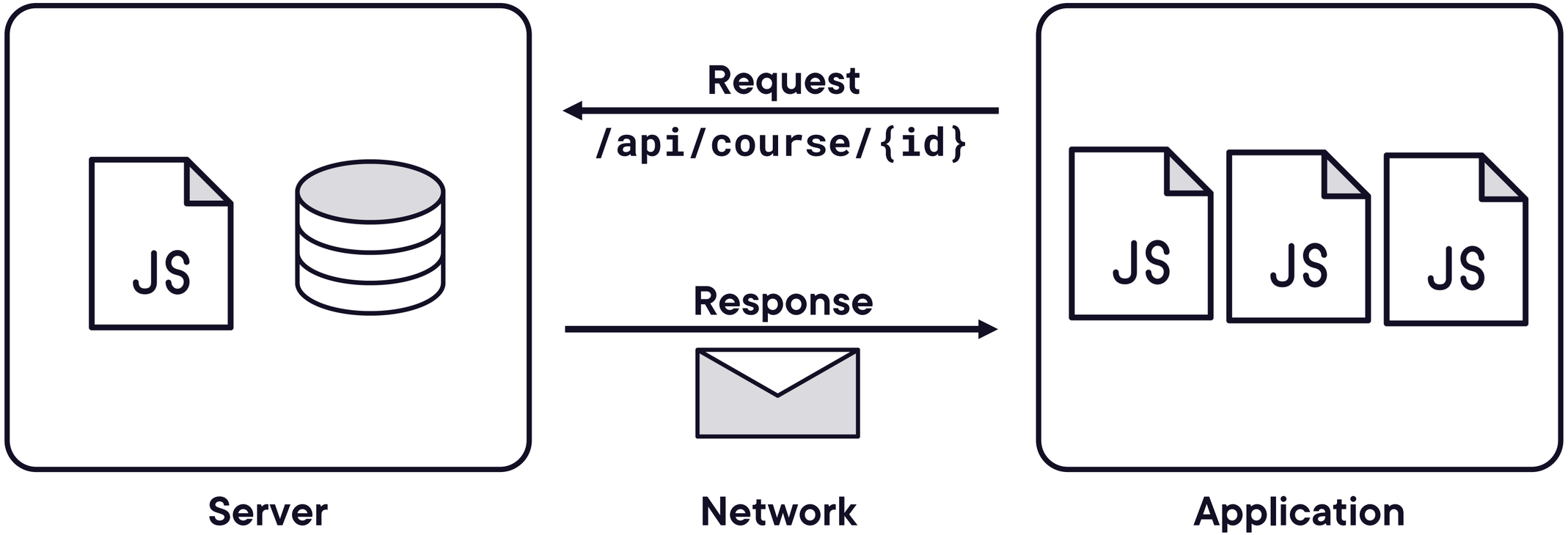
Stack vs. Heap



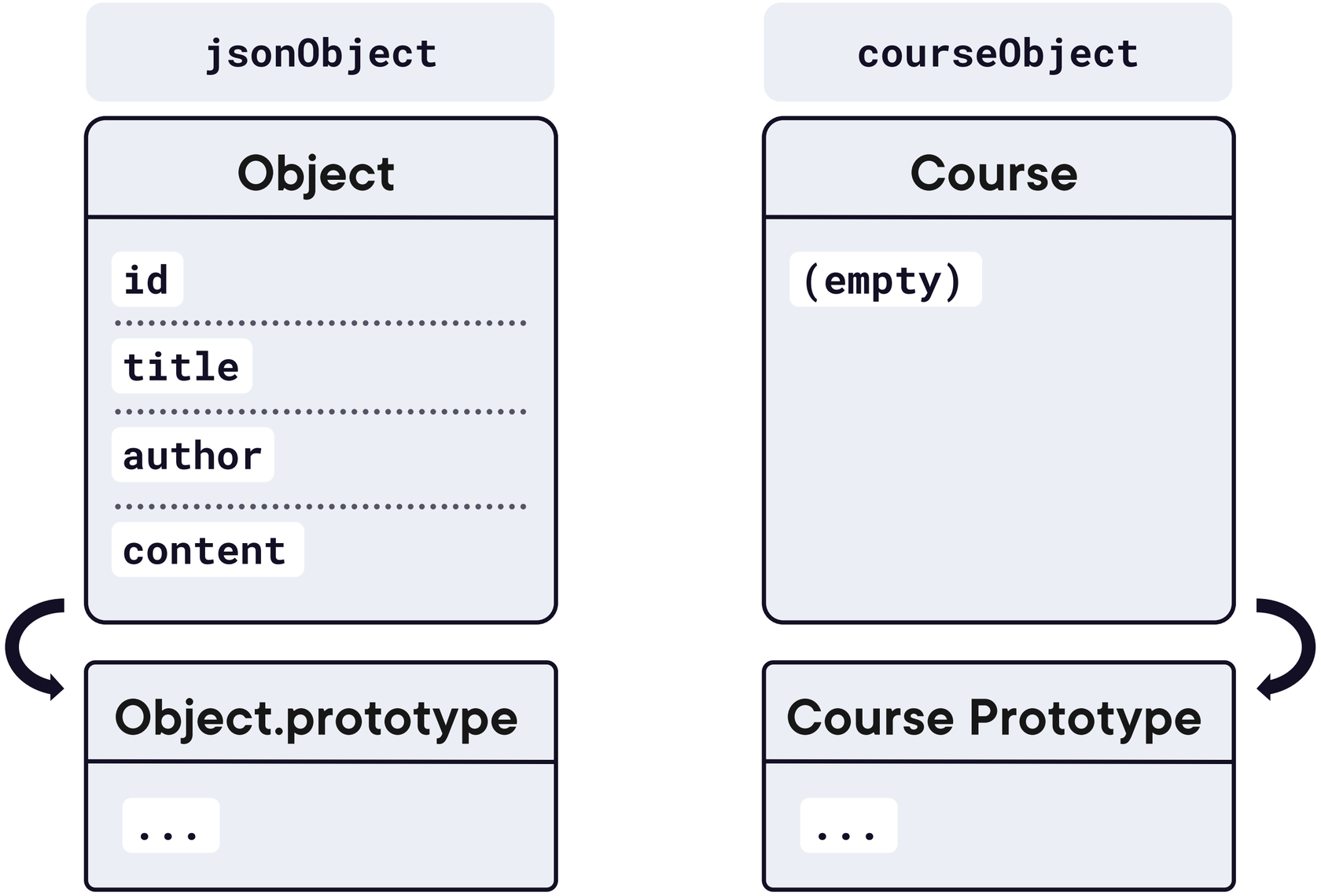
Cloning and Comparing Complexity



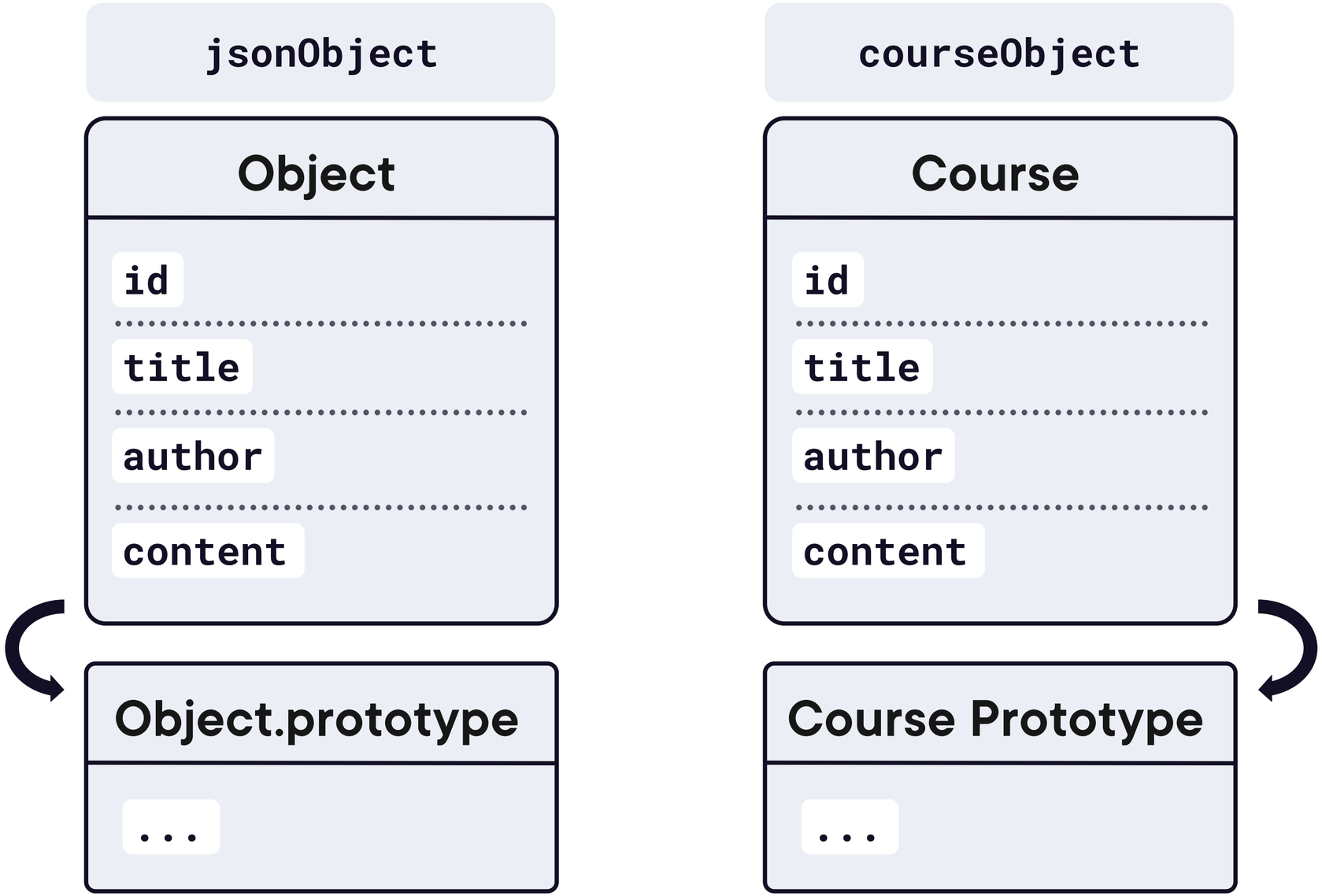
JSON to Transport Data From Server



Copying From JSON Process



Copying From JSON Process



Two Ways to Copy Properties

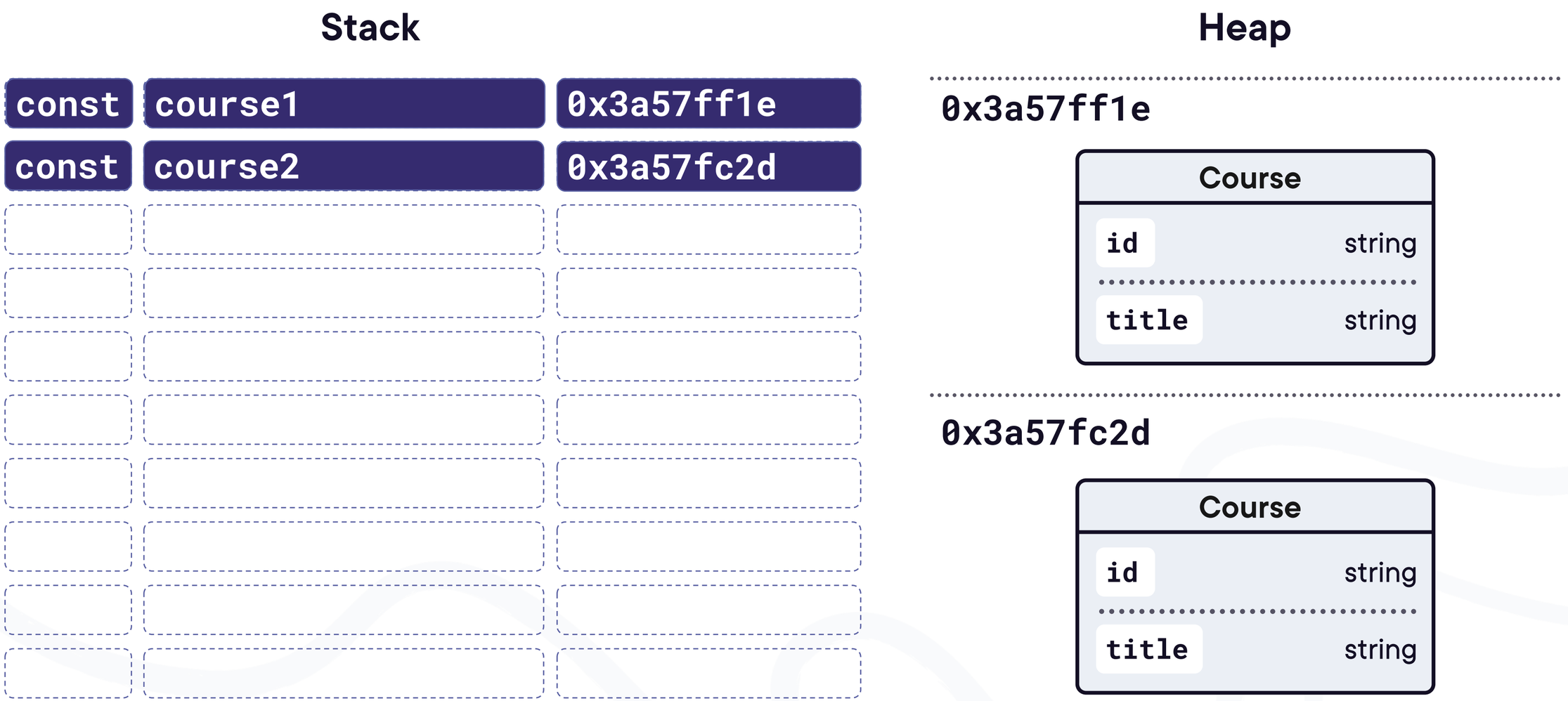
Object.assign

```
Object.assign(targetObject, sourceObject);
```

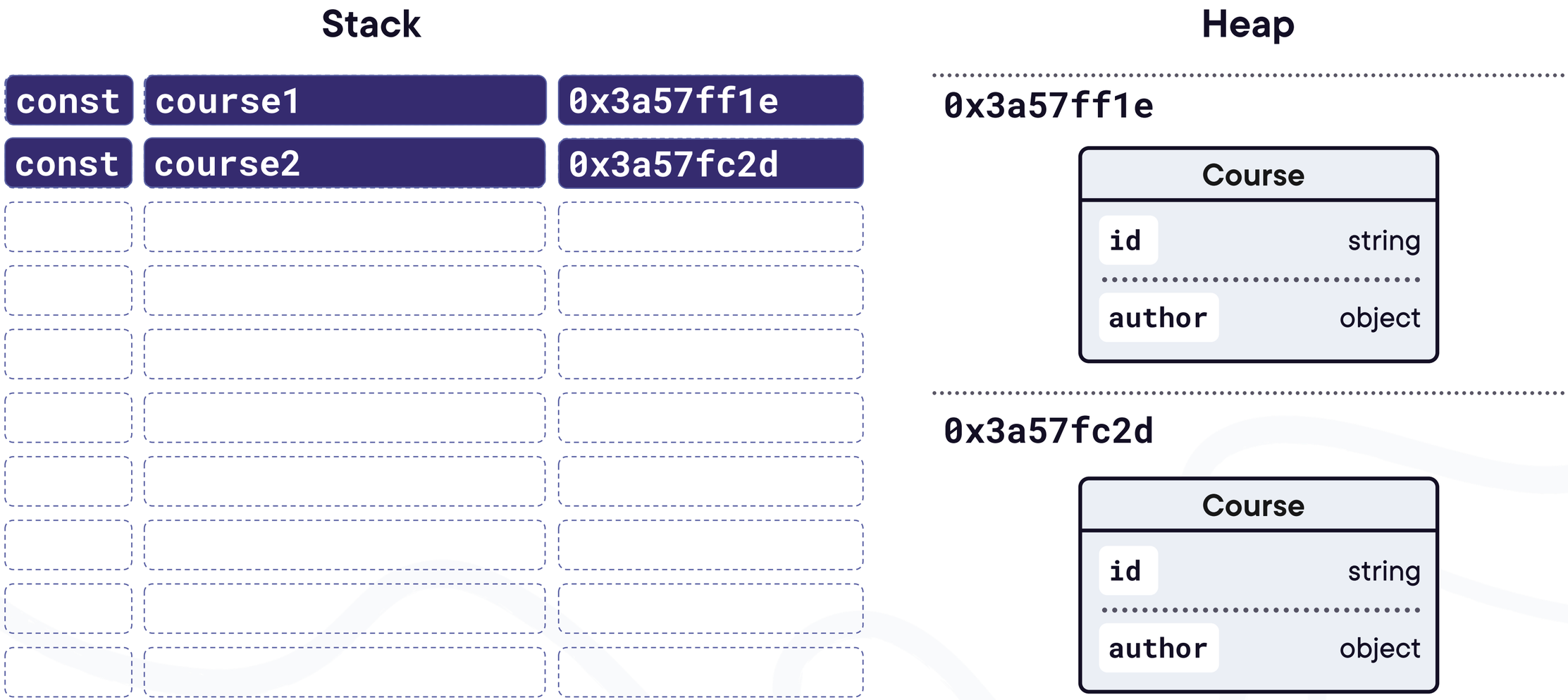
Spread Operator

```
newObject = { ...oldObject };
```

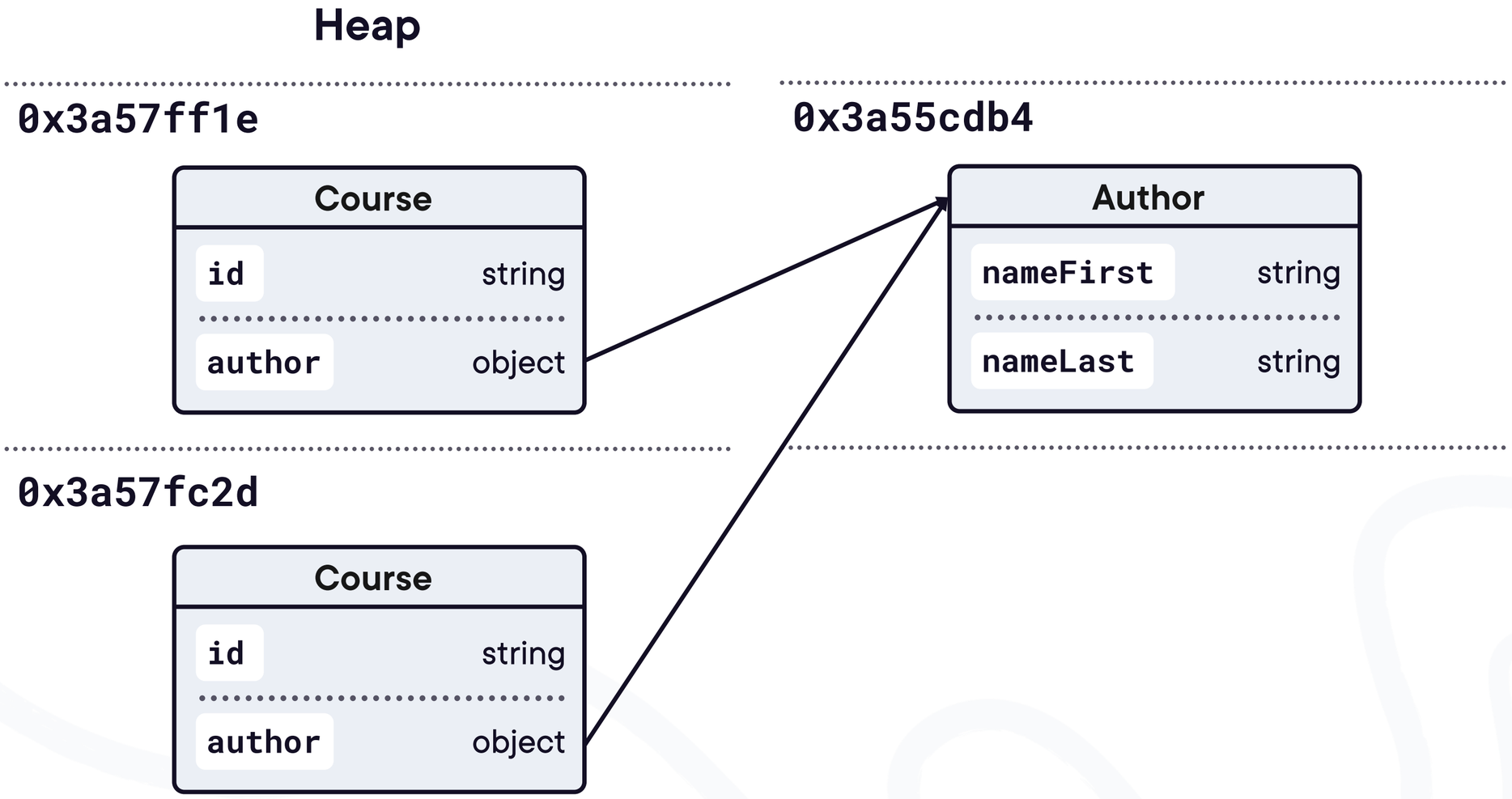
Stack vs. Heap



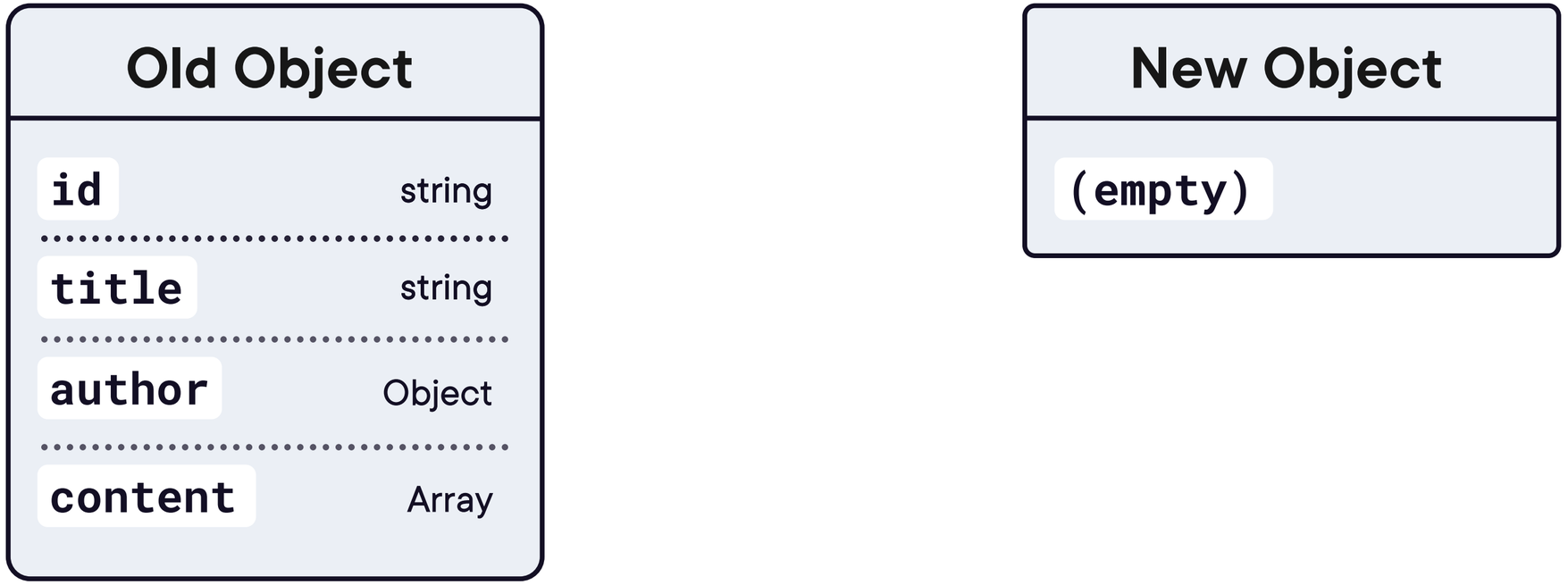
Stack vs. Heap



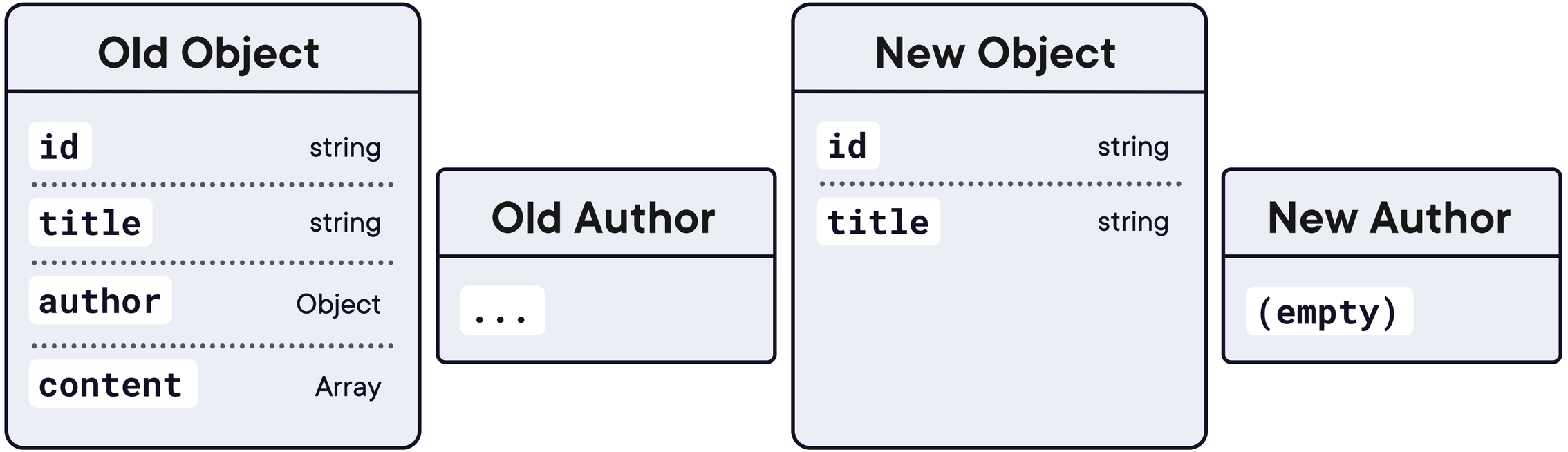
Stack vs. Heap



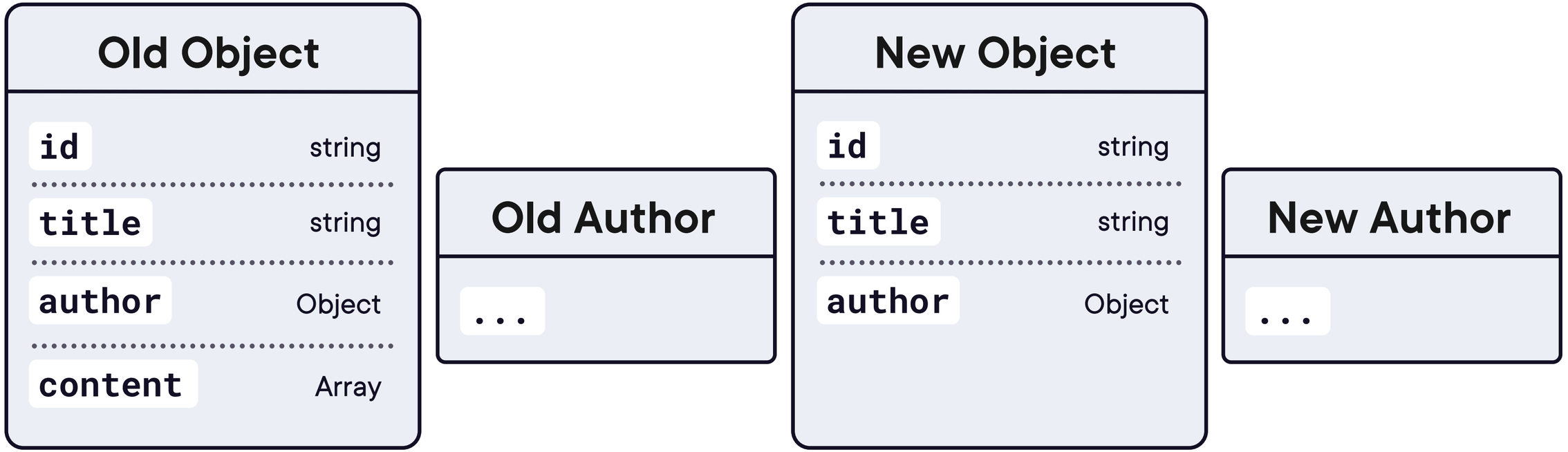
Deep Copying Approach



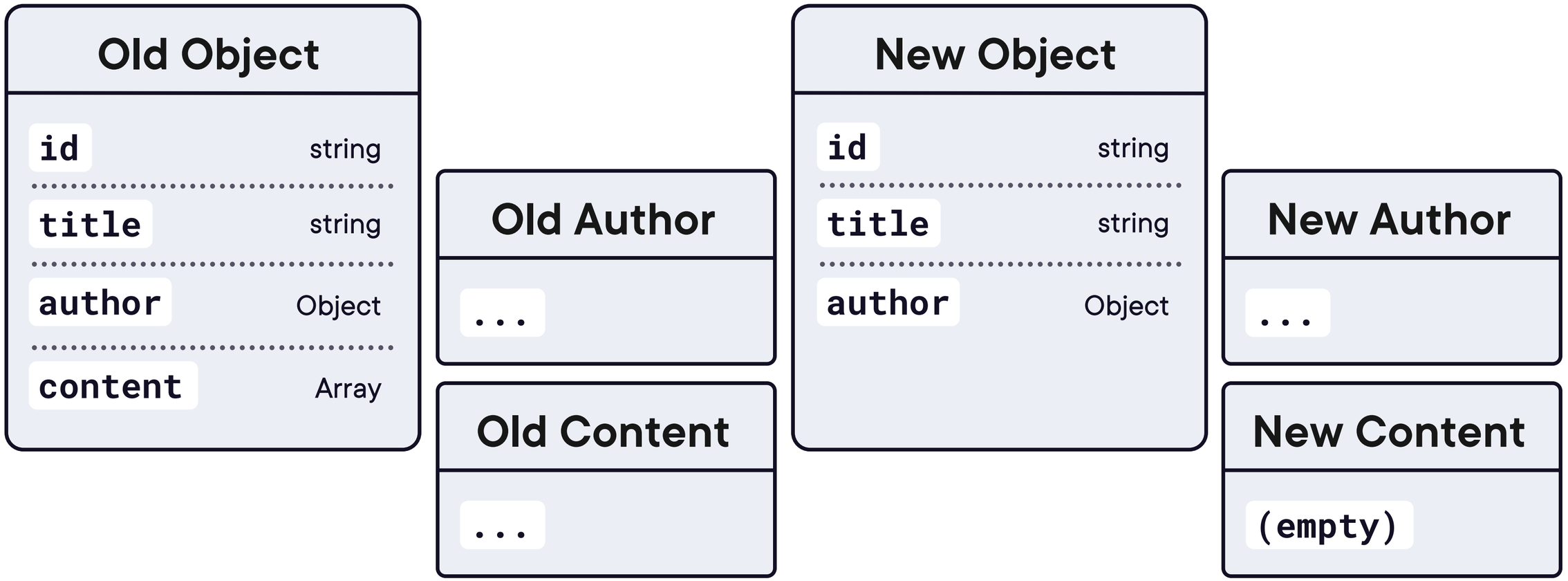
Deep Copying Approach



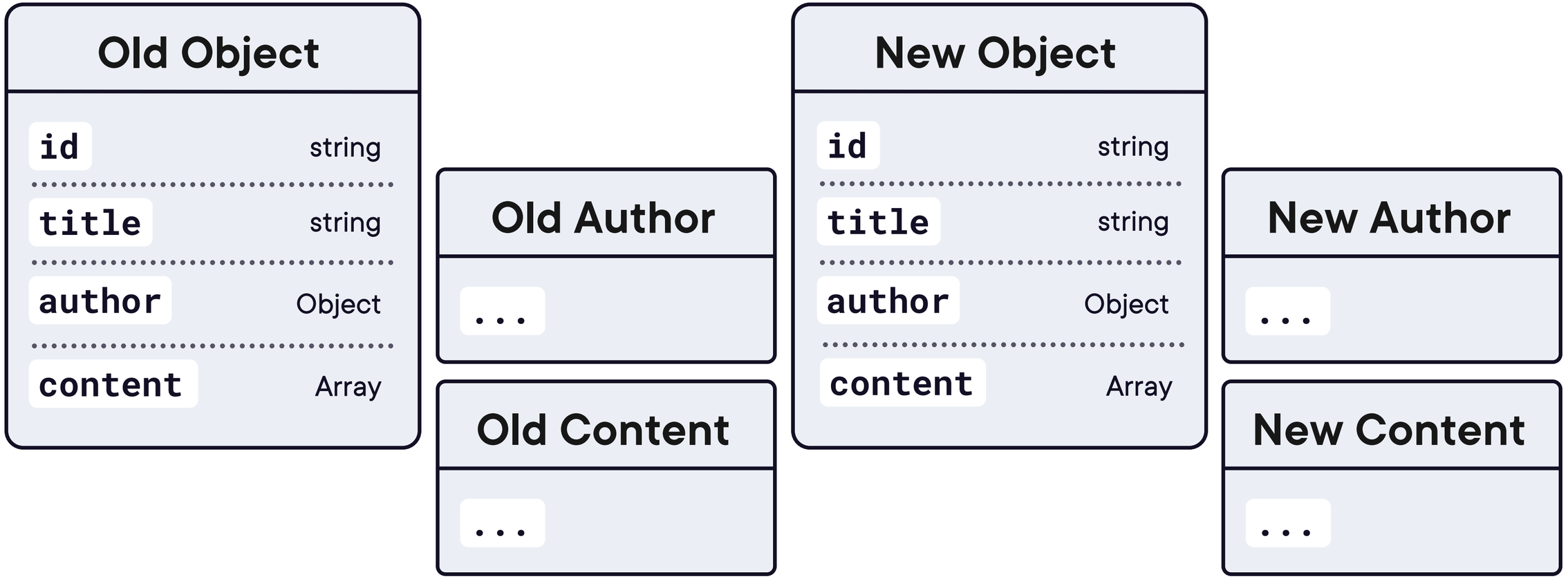
Deep Copying Approach



Deep Copying Approach



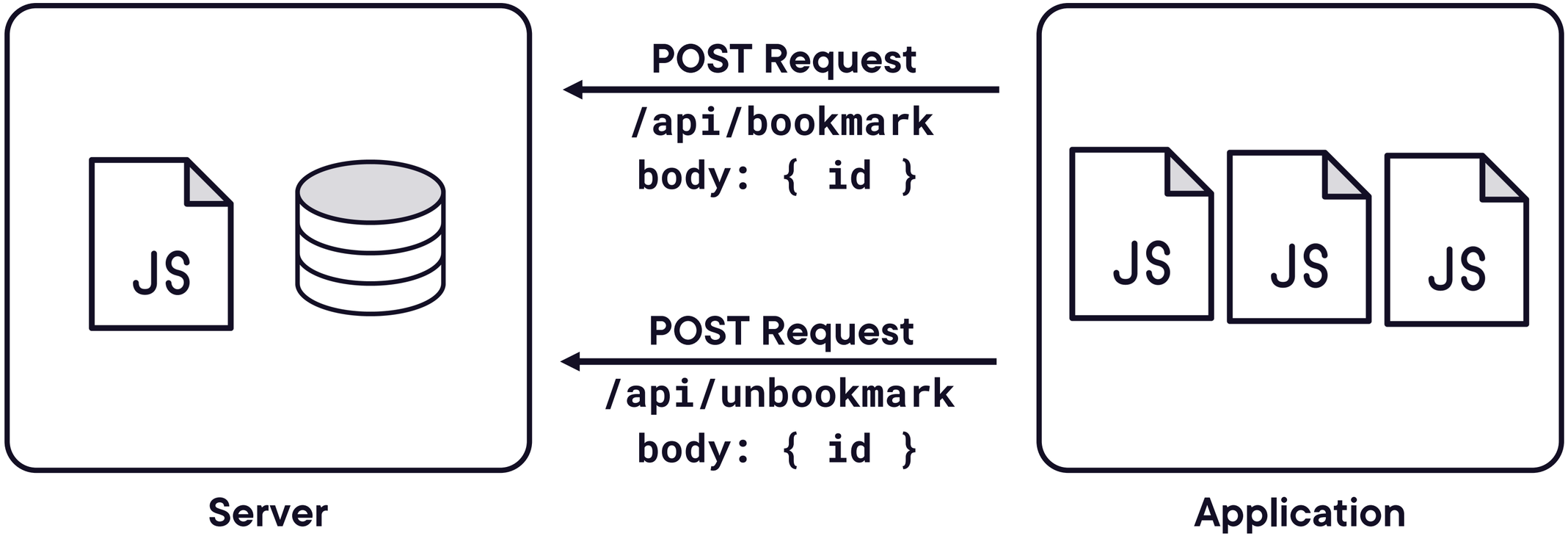
Deep Copying Approach



Up Next:

Functions as Properties

API Calls for Notebook



Two Ways to Define Functions in an Object Literal

Traditional Way

```
{
    newFunc: function() { ... }
}
```

Since ES6

```
{
    newFunc(): { ... }
}
```

Up Next:

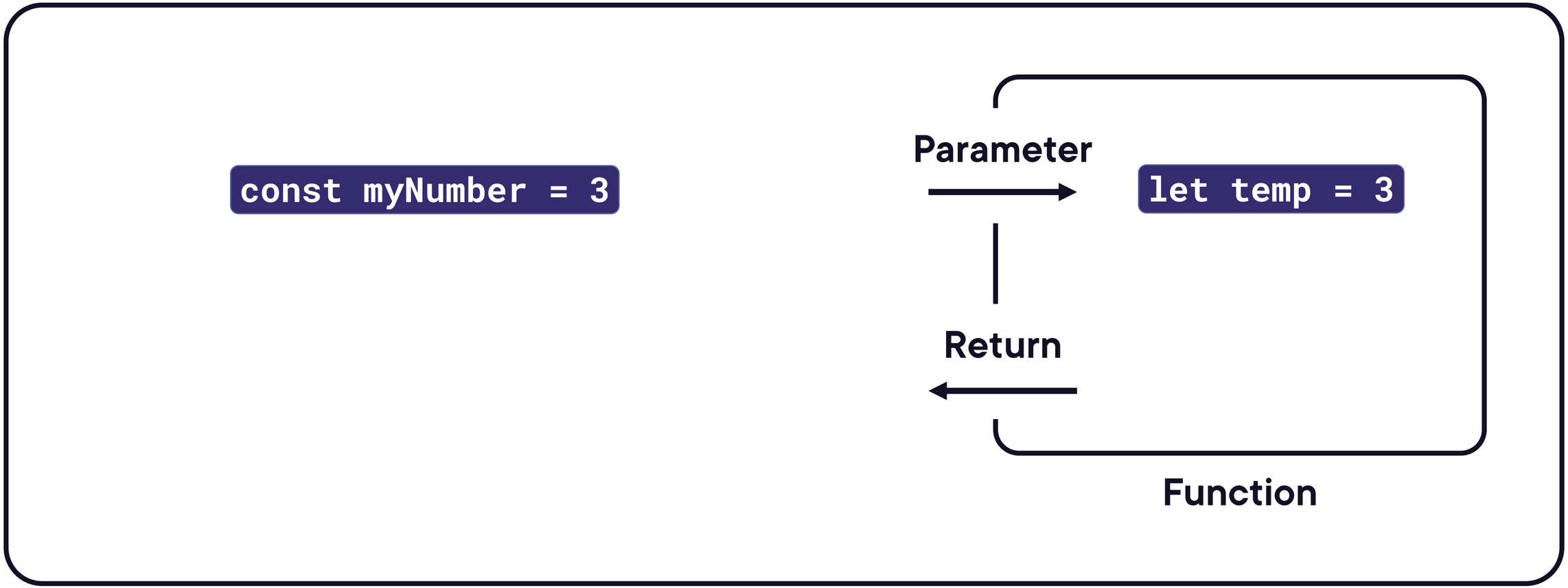
Scope in Objects

Object Scope Topics

What “this” Means

Objects as Function Parameters

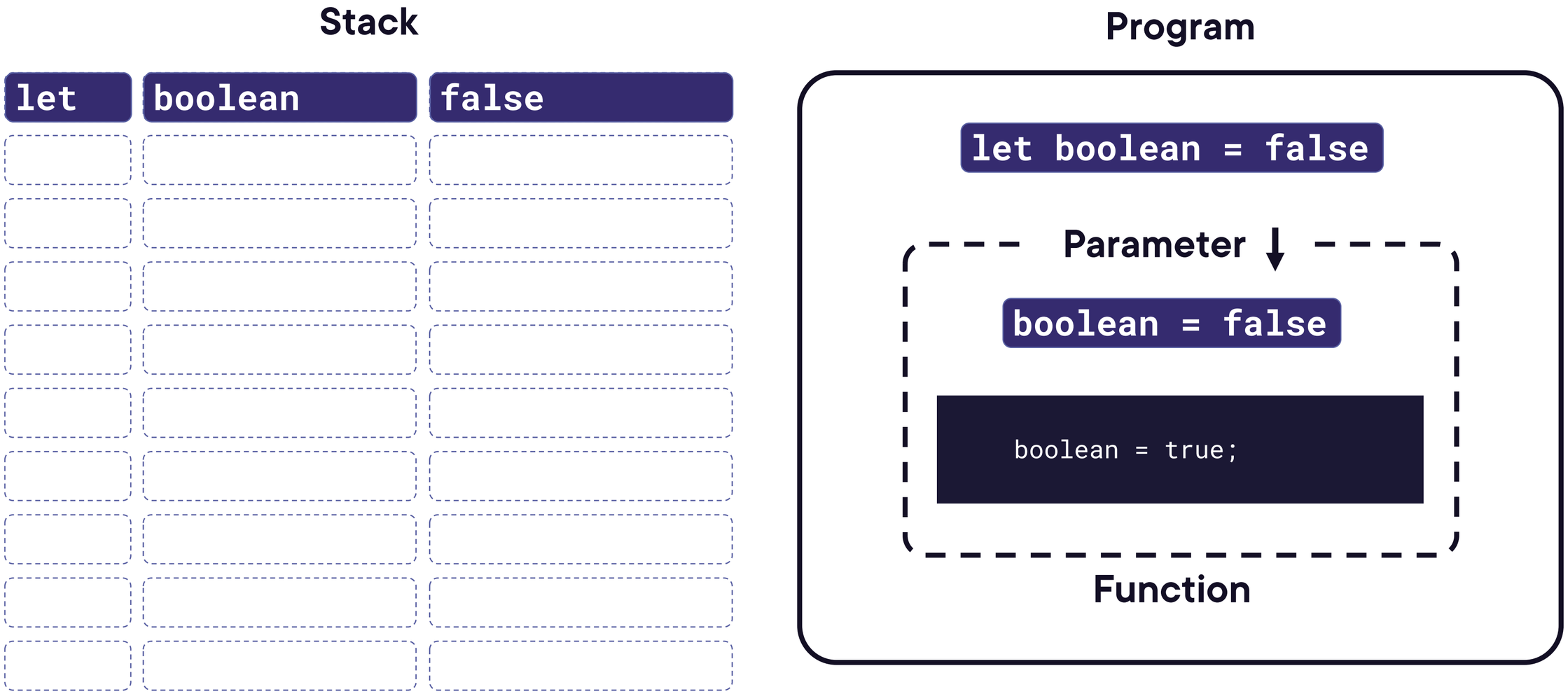
API Calls for Notebook



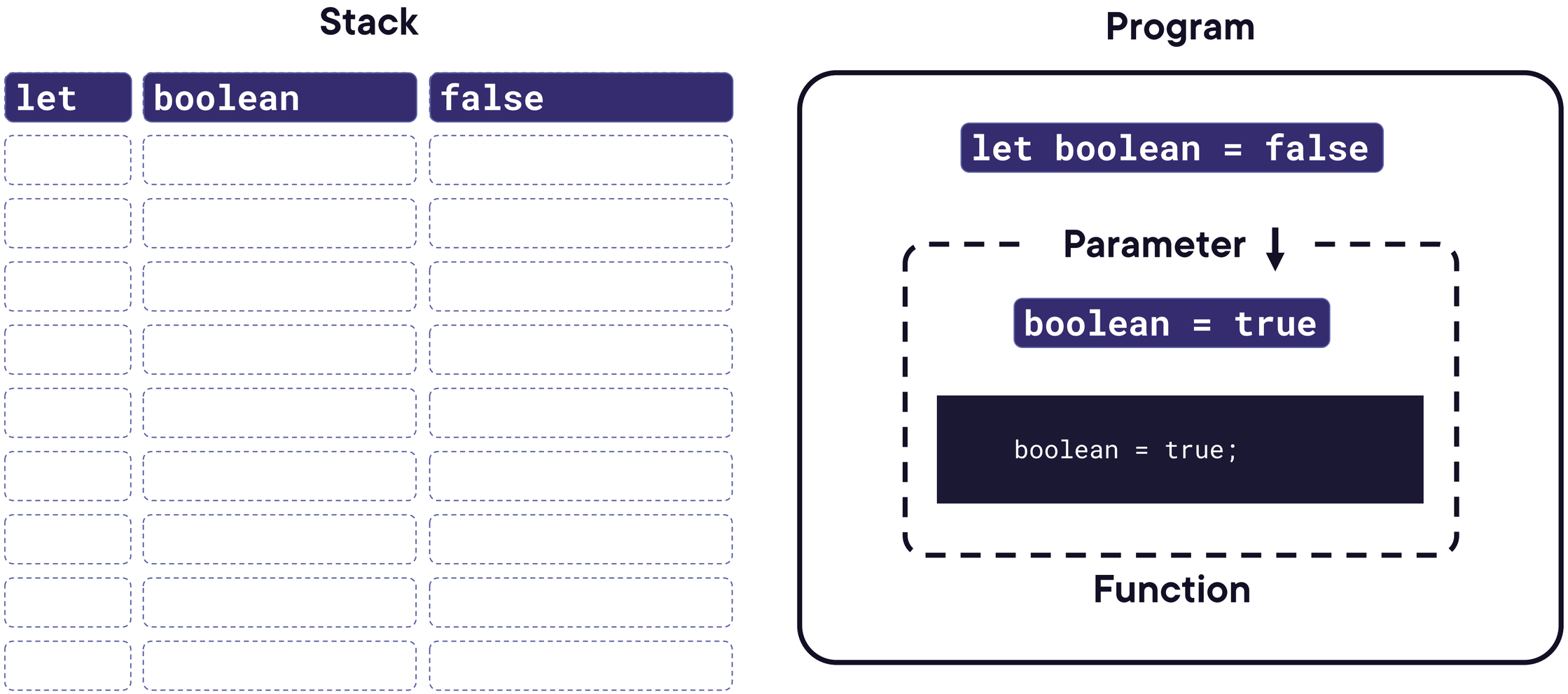
POST Request

```
/api/bookmark
body: { id }
```

Accessing Primitive Types Within Functions



Accessing Primitive Types Within Functions



Accessing Primitive Types Within Functions

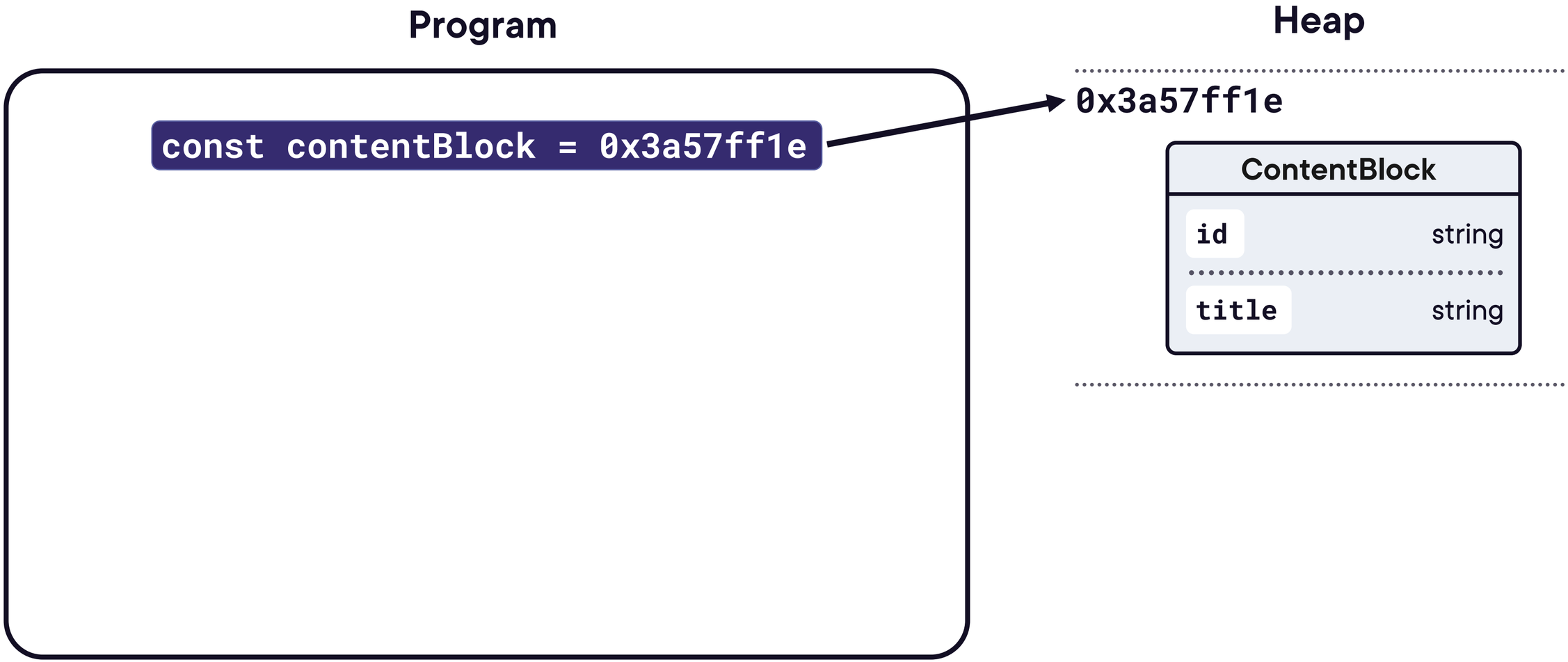
Stack

let	boolean	false

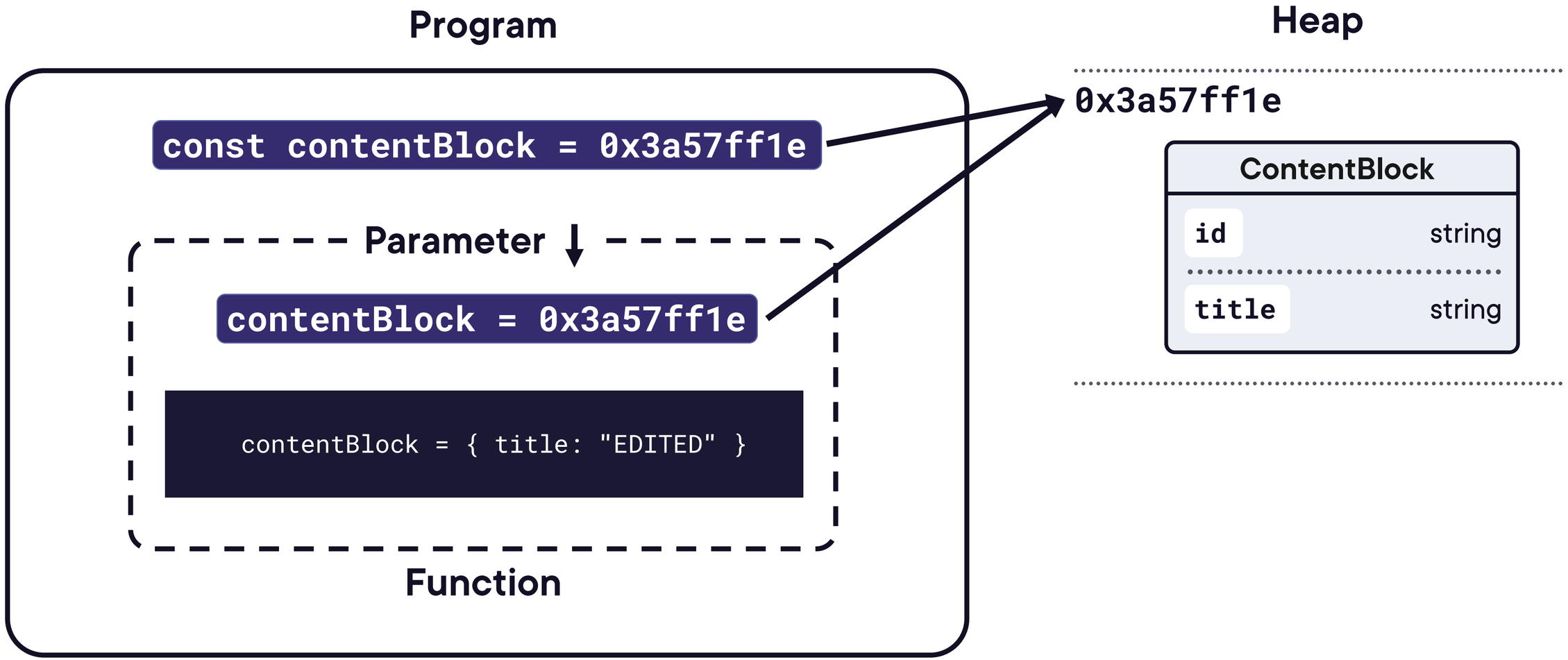
Program

```
let boolean = false
```

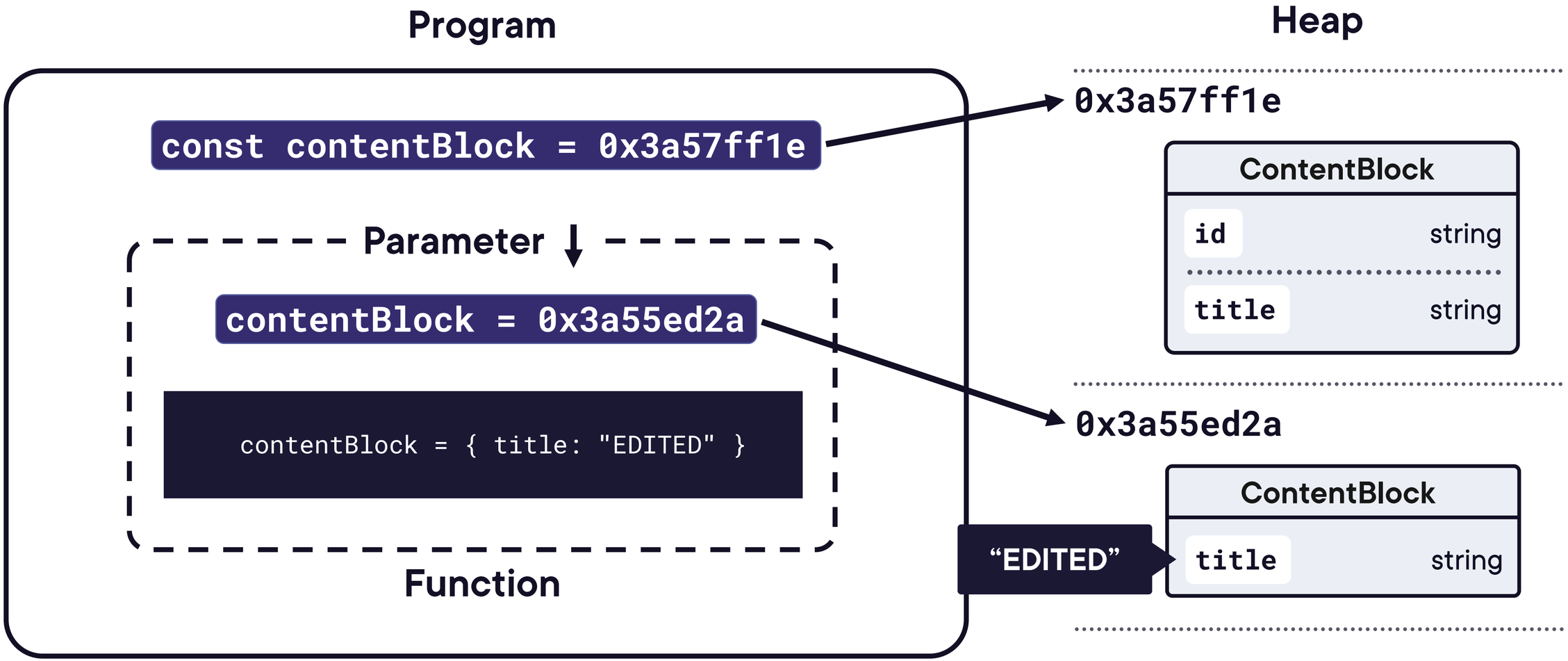
Accessing Objects Within Functions



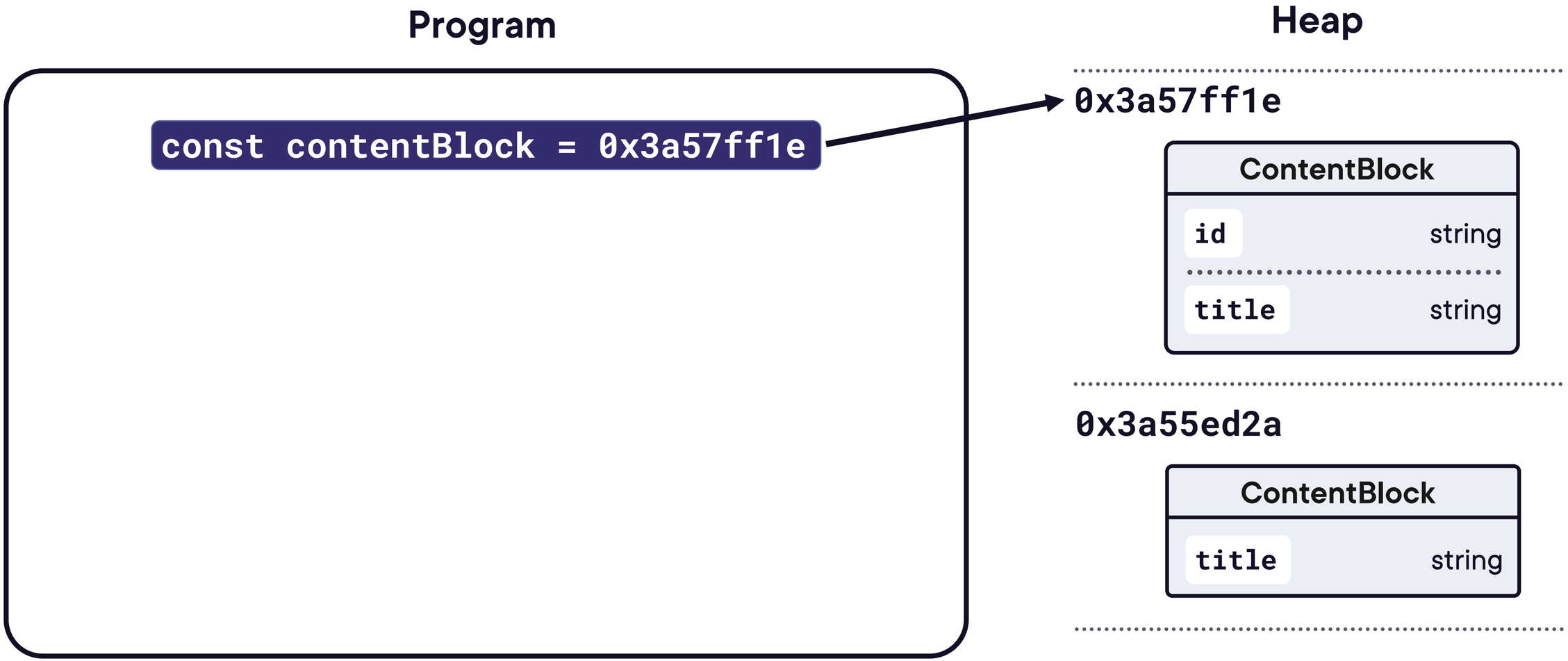
Accessing Objects Within Functions



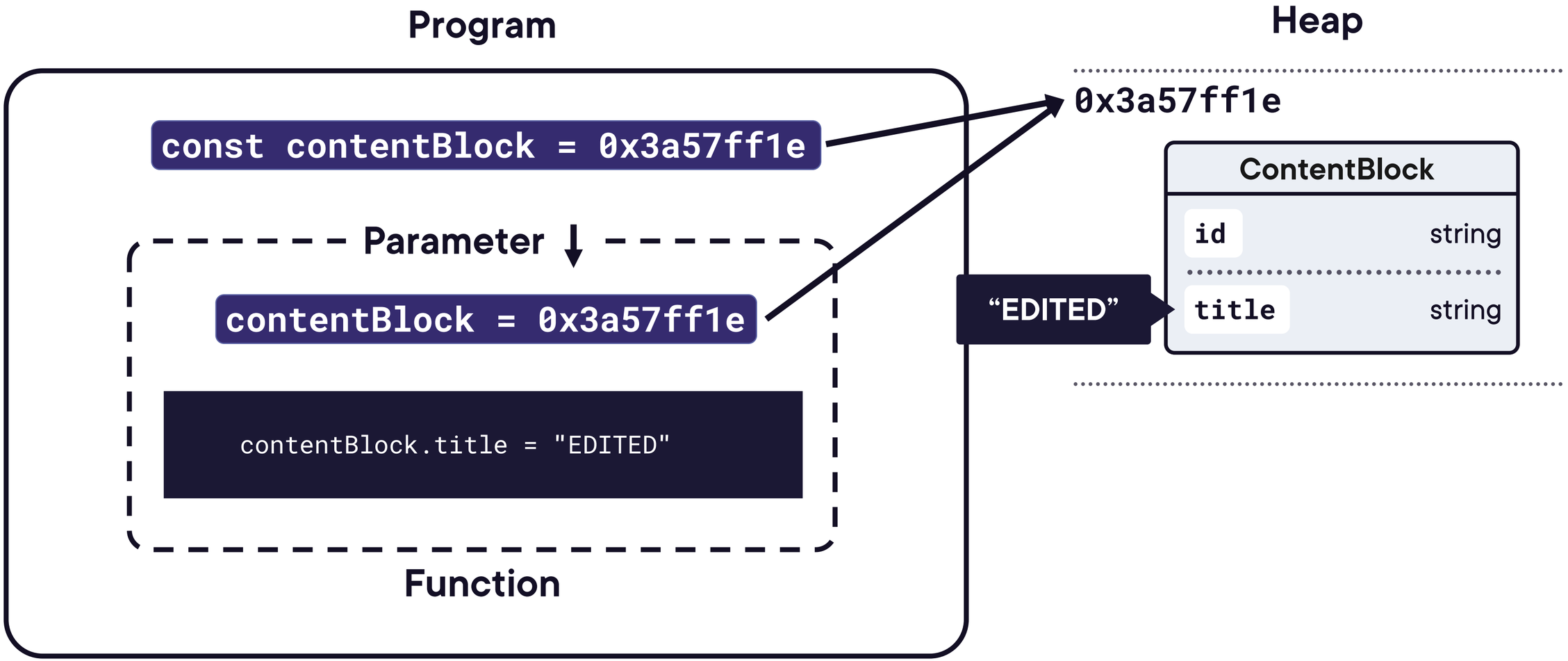
Accessing Objects Within Functions



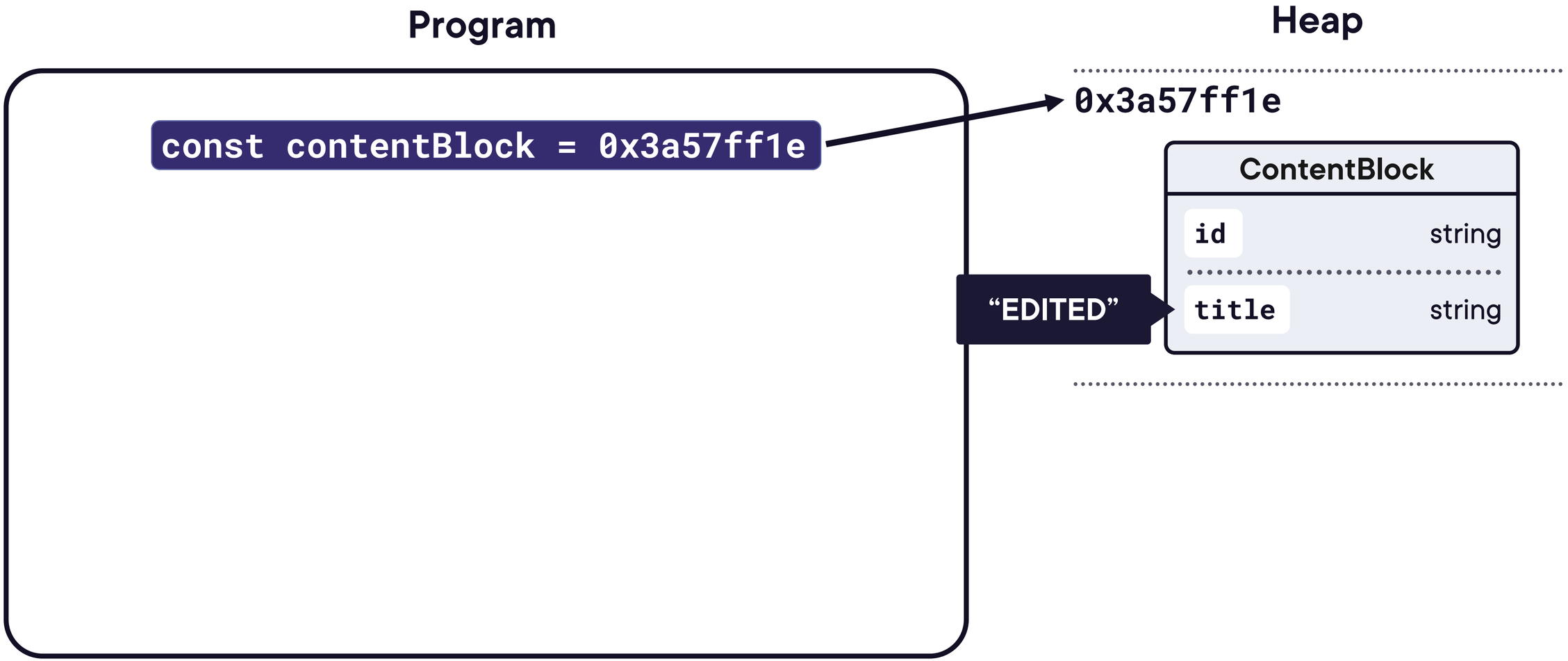
Accessing Objects Within Functions



Accessing Objects Within Functions



Accessing Objects Within Functions



Up Next:

Object Wrappers of Primitives

Objects behave differently than primitive types

Objects behave differently than primitive types*

Temporary Object Wrappers of Primitives

Program

```
let number = 45.8256;
number.toFixed(2); // 45.83
number = number + 2; // 47.8256
```

Heap

Temporary Object Wrappers of Primitives

Program

```
let number = 45.8256;
number.toFixed(2); // 45.83
number = number + 2; // 47.8256
```

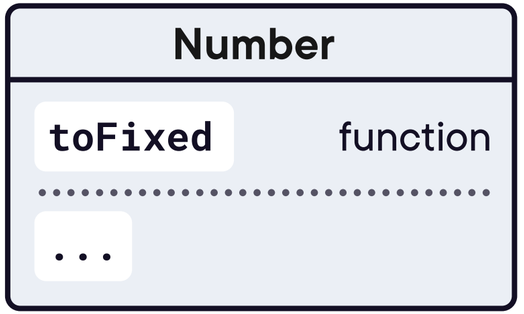
Heap

Temporary Object Wrappers of Primitives

Program

```
let number = 45.8256;
number.toFixed(2); // 45.83
number = number + 2; // 47.8256
```

Heap



Temporary Object Wrappers of Primitives

Program

```
let number = 45.8256;
number.toFixed(2); // 45.83
number = number + 2; // 47.8256
```

Heap

Primitive Wrappers

A convenient way to get access to object-like functionality on primitives, without needing to instantiate an object yourself

Course Overview

Objects

Prototypes & Properties

- What is an Object?
- Defining Prototypes with Object.create
- Creating Properties
- Accessing Properties
- Deleting Properties
- Protecting Properties
- Enumerating Properties

Reference

- Objects are Reference Types
- Comparing and Cloning Objects
- Functions as Properties
- Scope in Objects
- Object Wrappers of Primitives

Arrays

Arrays

- Arrays are Objects
- Creating Arrays
- Identifying Arrays
- Adding and Accessing Array Elements
- Cutting Up Arrays
- Putting Together Arrays
- Cloning Arrays
- Sorting Arrays
- Sparse Arrays
- Maps, Sets, and Other Array Types

Iteration

- Introduction to Array Iteration
- Basic For Loops
- Summarizing Arrays
- Searching Through Arrays
- Map

Course Overview

Objects

Prototypes & Properties

- What is an Object?
- Defining Prototypes with Object.create
- Creating Properties
- Accessing Properties
- Deleting Properties
- Protecting Properties
- Enumerating Properties

Reference

- Objects are Reference Types
- Comparing and Cloning Objects
- Functions as Properties
- Scope in Objects
- Object Wrappers of Primitives

Arrays

Arrays

- Arrays are Objects
- Creating Arrays
- Identifying Arrays
- Adding and Accessing Array Elements
- Cutting Up Arrays
- Putting Together Arrays
- Cloning Arrays
- Sorting Arrays
- Sparse Arrays
- Maps, Sets, and Other Array Types

Iteration

- Introduction to Array Iteration
- Basic For Loops
- Summarizing Arrays
- Searching Through Arrays
- Map