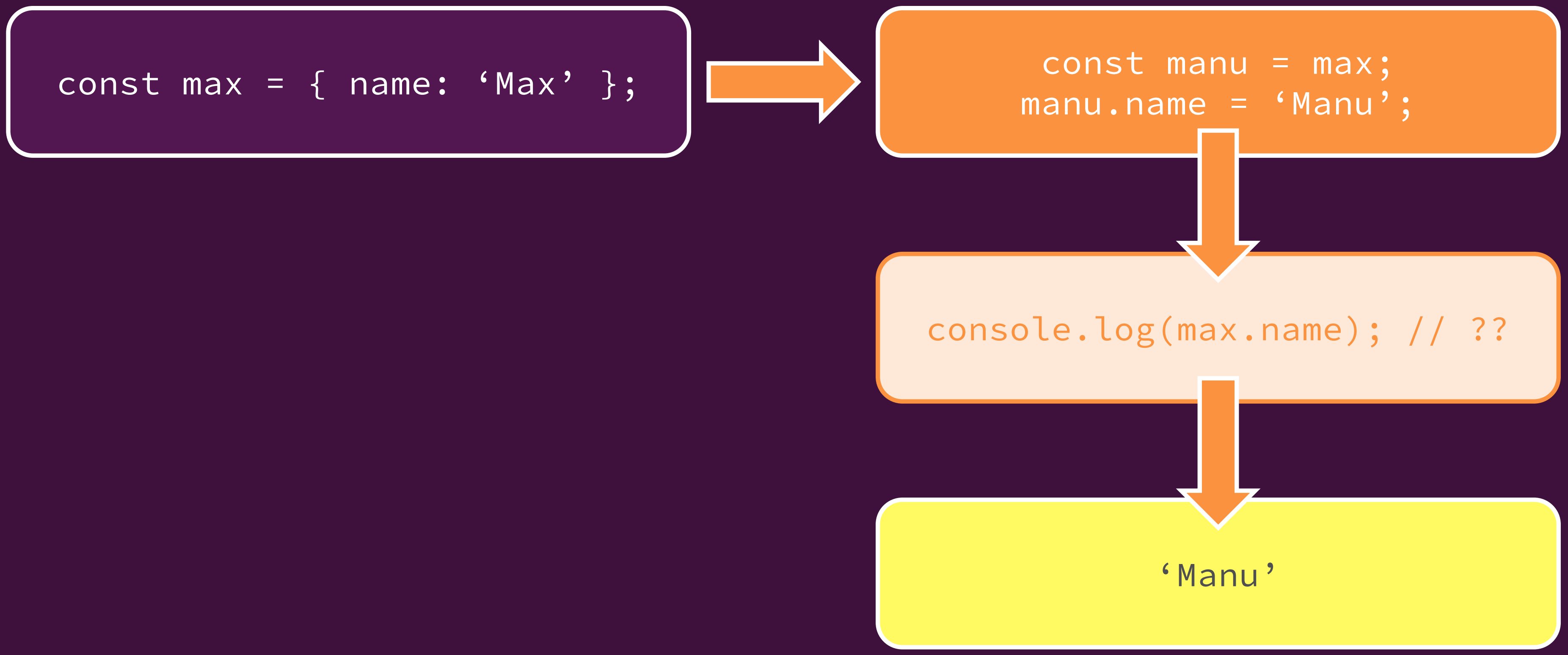
What?

Primitive Values	Reference Values
number string boolean symbol null undefined bigint	Objects (i.e. plain objects, arrays, functions, ...)
Immutable non-object values	Mutable object values

Reference Values in Action



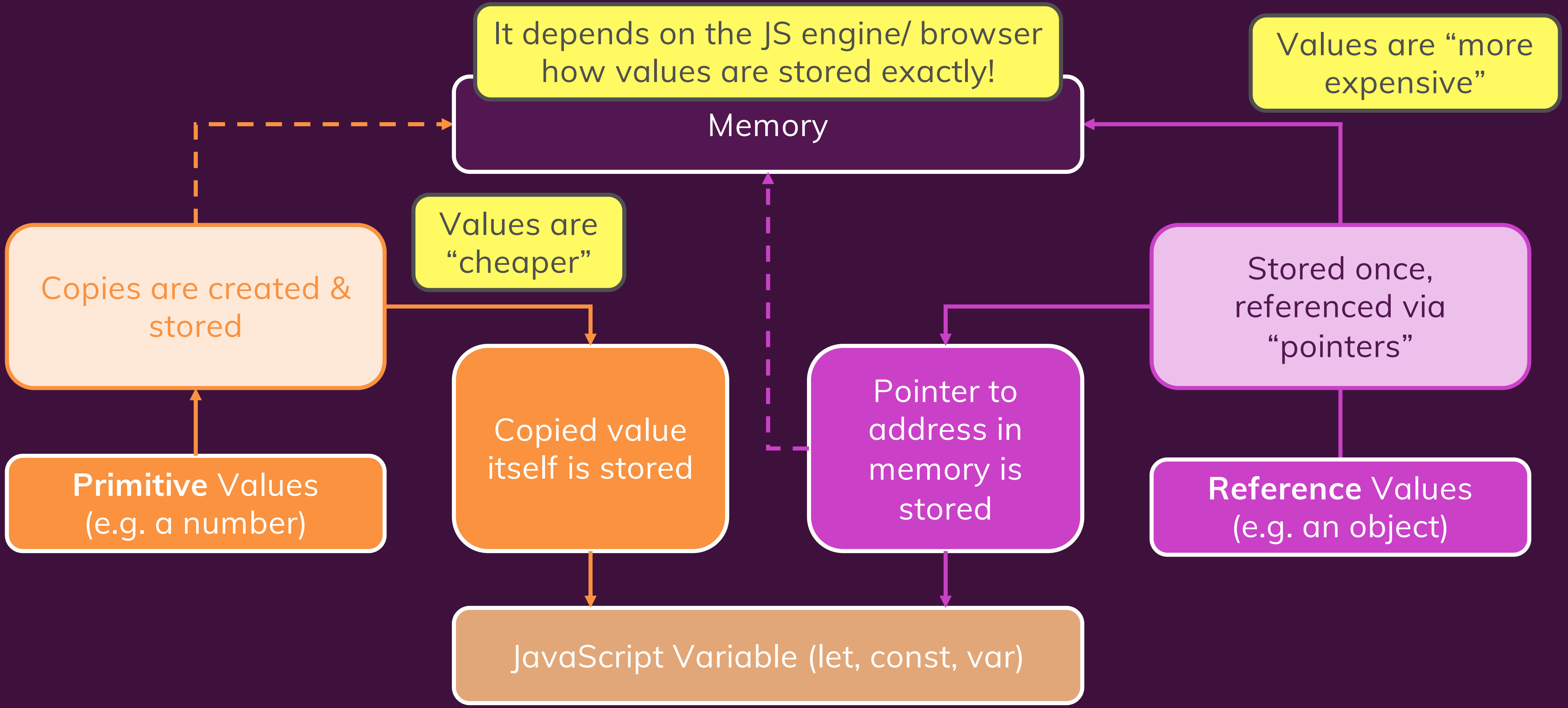
```
const max = { name: ‘Max’ };
```

```
const manu = max;
manu.name = ‘Manu’;
```

```
console.log(max.name); // ??
```

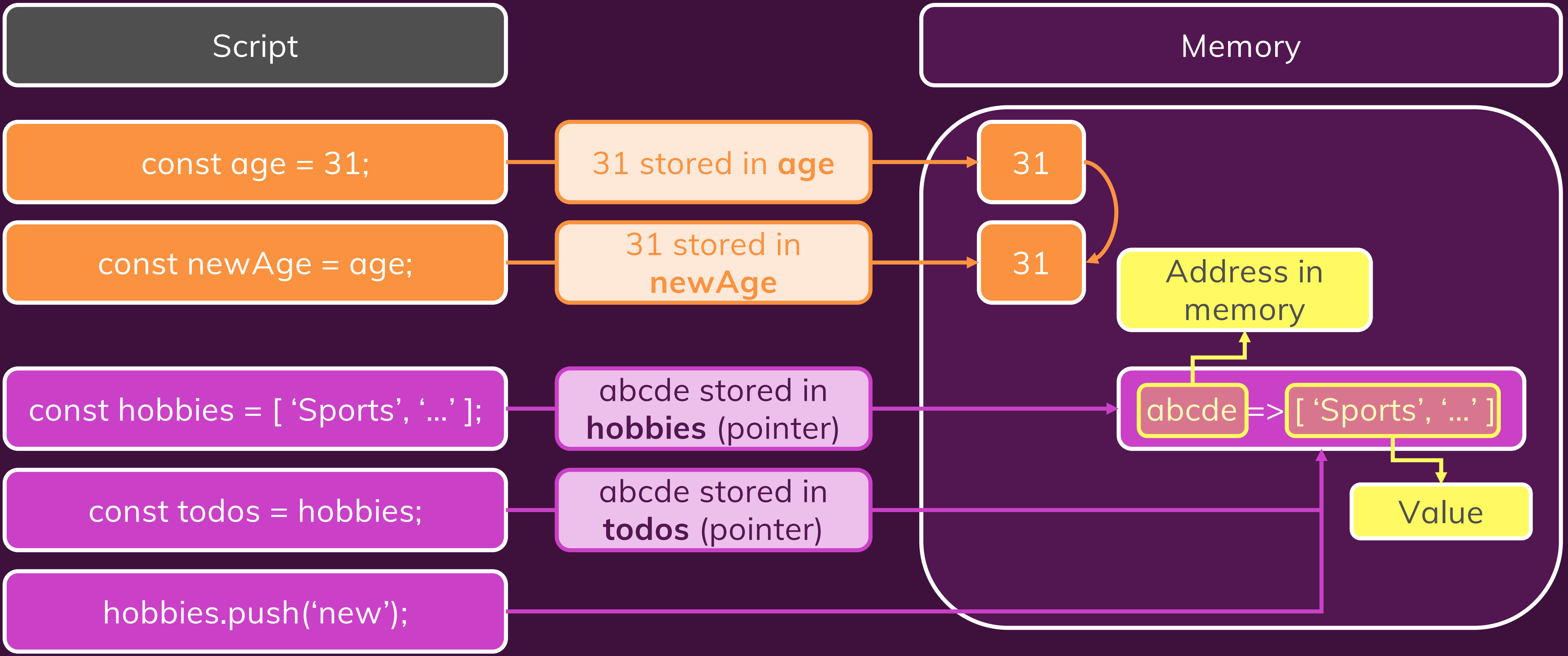
```
‘Manu’
```

How Data Is Stored



It depends on the JS engine/ browser how values are stored exactly!
Memory
Values are "more expensive"
Values are "cheaper"
Copies are created & stored
Stored once, referenced via "pointers"
Copied value itself is stored
Pointer to address in memory is stored
Primitive Values (e.g. a number)
Reference Values (e.g. an object)
JavaScript Variable (let, const, var)

Behind the Scenes



Primitive Wrapper Objects

Primitive Values are **not** Objects!

↓

But to simplify working with them, there are "invisible object wrappers" to expose utility methods etc.

↓

String(), Number(), Boolean(), BigInt(), Symbol()

What about “const” Arrays & Objects?

Primitives: Value can’t be changed anyways (immutable)

const user = { name: ‘Max’ };

const means: “Don’t assign a new value to the ‘variable’ user”.

It does NOT mean: “Don’t change the stored value”!

Reference values: Underlying value CAN be changed because only a **pointer** is stored in the ‘variable’

Summary – Primitive vs Reference Values

JavaScript has various data types but two kind of data type categories: **Primitive Values** (“Primitives”) and **Reference Values**

Primitives:

- Number
- String
- Boolean
- Symbol
- null & undefined
- BigInt

Reference Values:

- All Objects (incl. Arrays, Functions)

Primitive values are shared by **copy** and **immutable**.
Reference values are shared by **reference** (i.e. NOT copied) and are **mutable**.

“**Mutable**” means that data can be edited without copying the value first.

“**Reference**” means that a pointer to the object in memory is used/ shared.