DON'T INTERFACE EVERYTHING

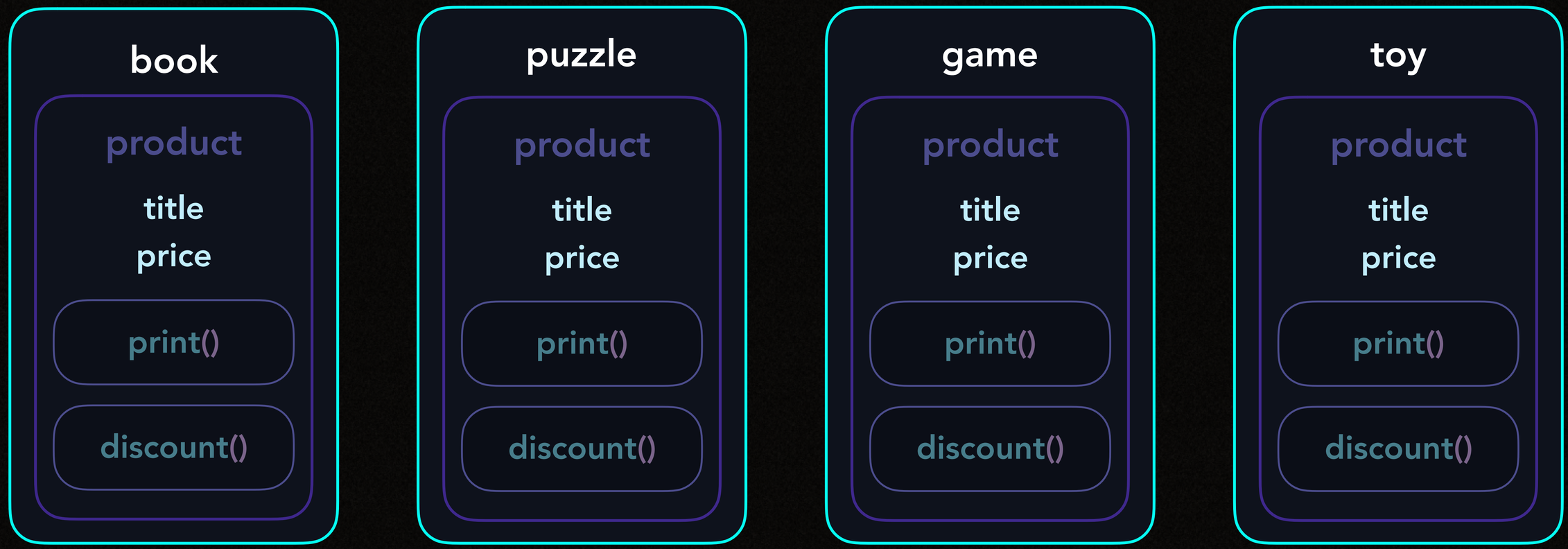
Prefer concrete types

GOALS

- ★ Simplify the program from the last lecture
  - ★ Use concrete types
- ★ Create a new timestamp type
  - ★ Separate the responsibilities
  - ★ Discover the power of embedding

UNNECESSARY COMPLEXITY

All the product types uses the same set of fields and methods



UNNECESSARY COMPLEXITY

Only the book has an additional field: "published."

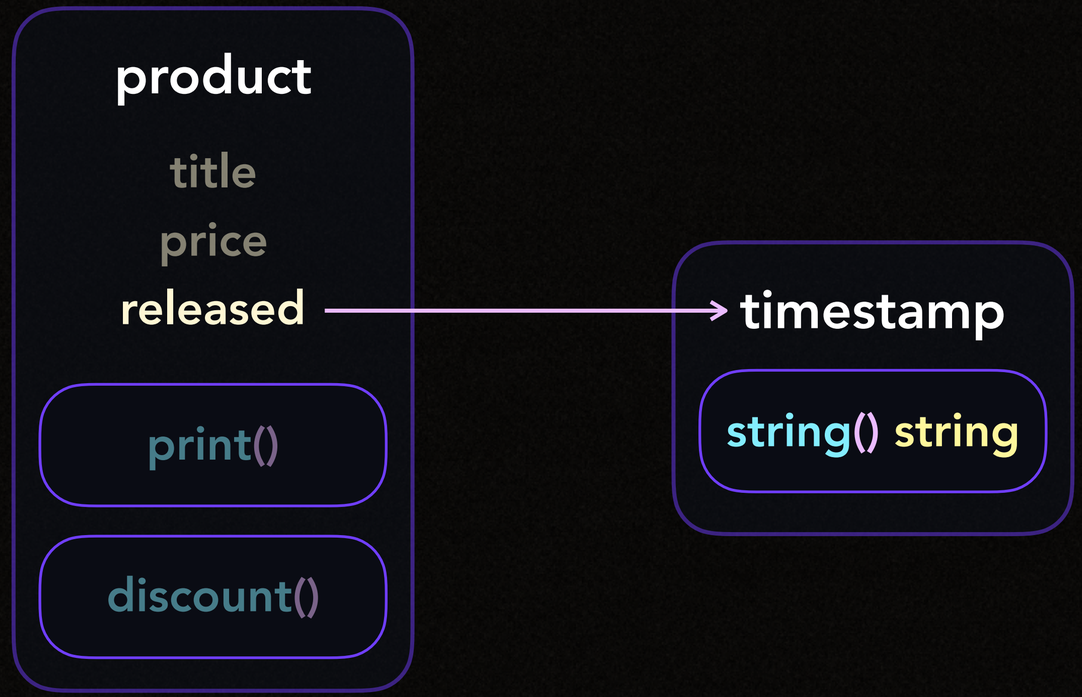
SIMPLIFIED

One type to rule'em all, one type to bring'em all, and in the light bind them

Instead of having multiple product types,
you could add a **category** field
to the **product** type.

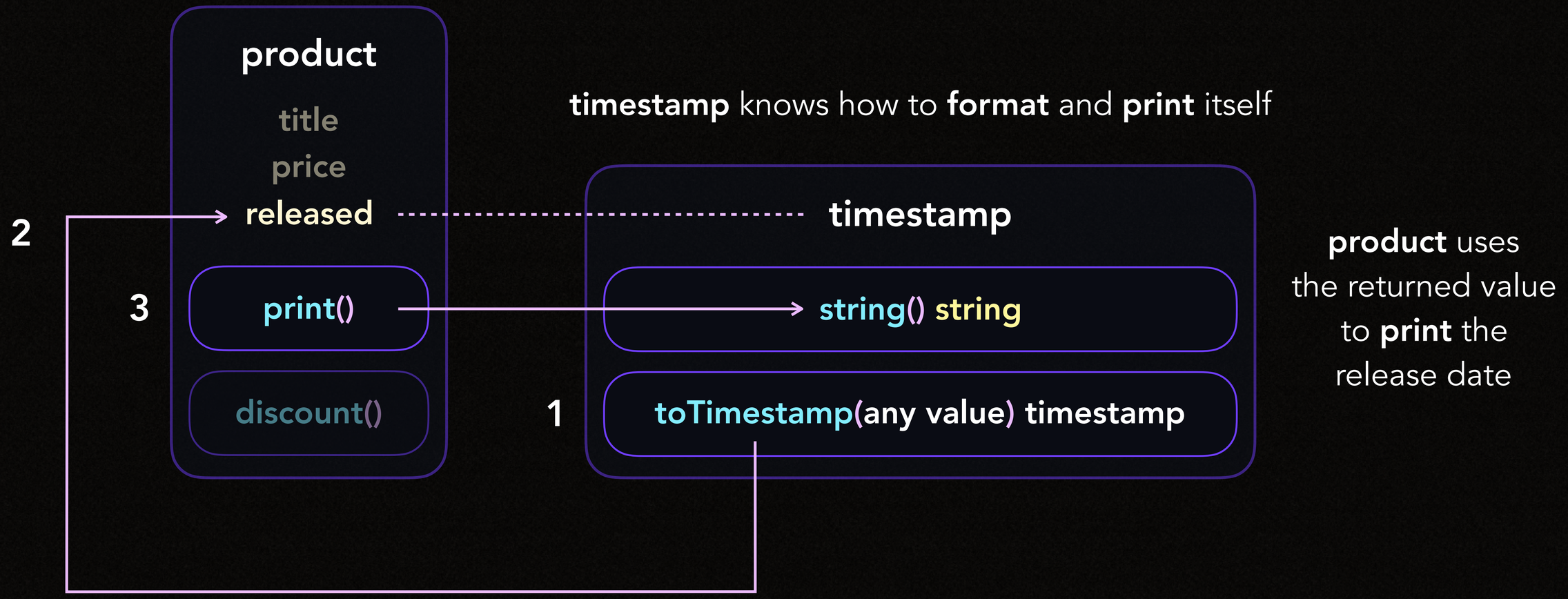
RESPONSIBILITIES

"product" should not know how to print "timestamps"



CLEAR RESPONSIBILITIES

product is much simpler thanks to the timestamp



SUMMARY

★ **Prefer to work directly with concrete types**
- Leads to a simple and easy to understand code
- Abstractions (interfaces) can unnecessarily complicate your code

★ **Separating responsibilities is critical**
- **Timestamp** type can represent, store, and print a UNIX timestamp

★ When a type **anonymously embeds** a type, it can use the methods of the embedded type as its own.
- **Timestamp** embeds a **time.Time**
- So you can call the methods of the **time.Time** through a timestamp value

STRINGER

Can you make a type print itself automatically as a string?

Customize the way a type represents itself as a string

```
type Stringer interface {
    String() string
}
```

The printing functions
can automatically call the
String method

fmt.Stringer

javascript	java	python	ruby
toString()	toString()	__str__	to_str()

fmt.Stringer

io.Writer

sort.Interface

io.Reader

error

json.Marshaler

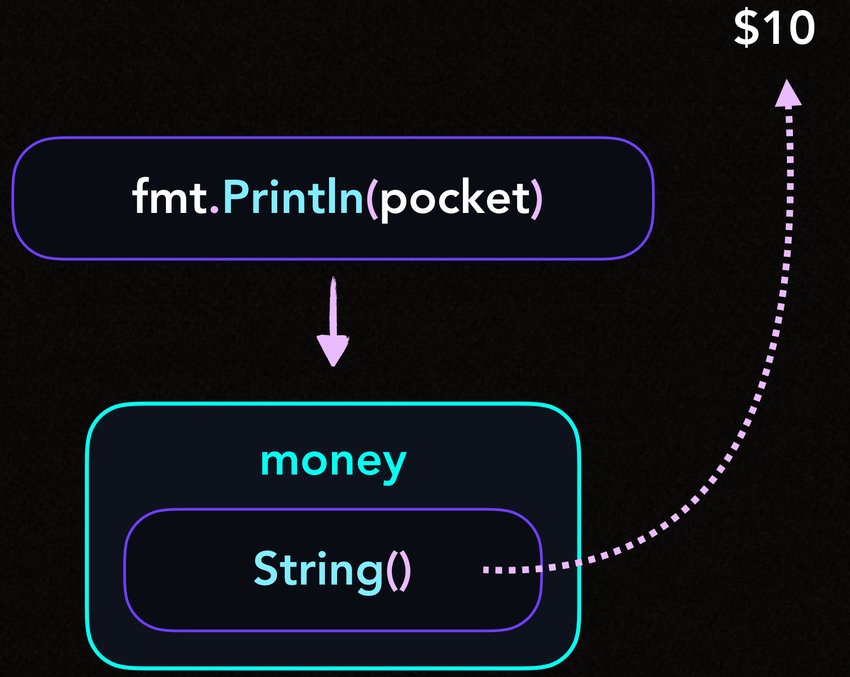
... and so on

HOW DOES IT WORK?

Print functions call String() if a type is a Stringer

Println() type asserts
and sees that
the money is a Stringer

Println() runs
String() method
of the money



Println() prints
the string value that
it gets from the money

Interfaces

group types by behavior

Satisfy
existing interfaces

Interfaces
are abstract bridges
between types

Reusability

Most Go programmers prefer to satisfy existing Go interfaces

SUMMARY

★ **fmt.Stringer has one method: String()**
- That returns a **string**.
- It is better to be an **fmt.Stringer** instead of printing directly.

★ **Implement the String()** on a type and the type can represent itself as a **string**.
- Bonus: The functions in the **fmt package** can print your type.
- They use **type assertion** to detect if a type implements a **String()** method.

★ **strings.Builder** can **efficiently** combine multiple string values.

SORTER

You don't need to implement a sort algorithm

fmt.Stringer

io.Writer

sort.Interface

io.Reader

error

json.Marshaler

... and so on

fmt.Stringer

io.Writer

sort.Interface

io.Reader

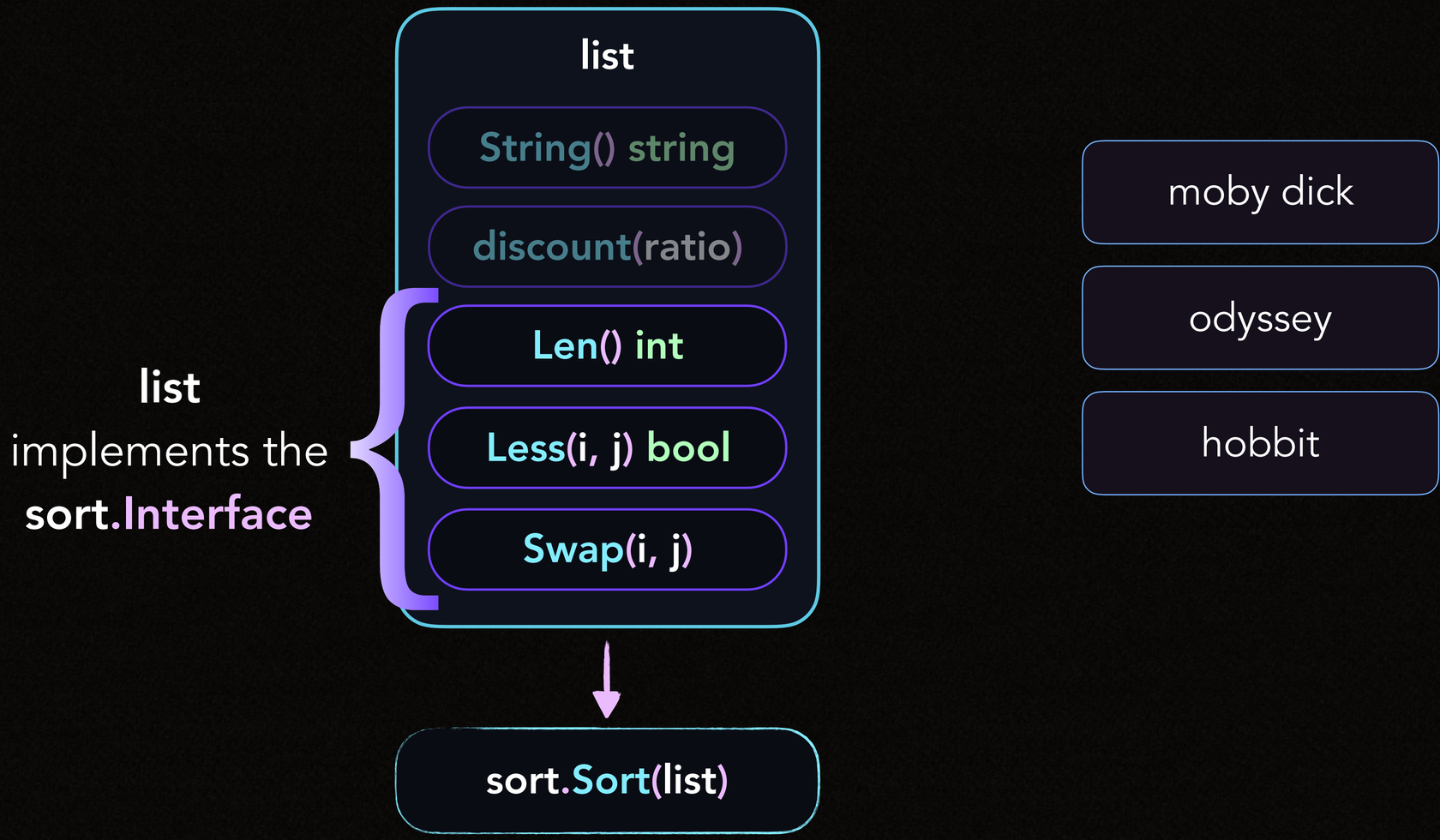
error

json.Marshaler

... and so on

HOW DOES IT WORK?

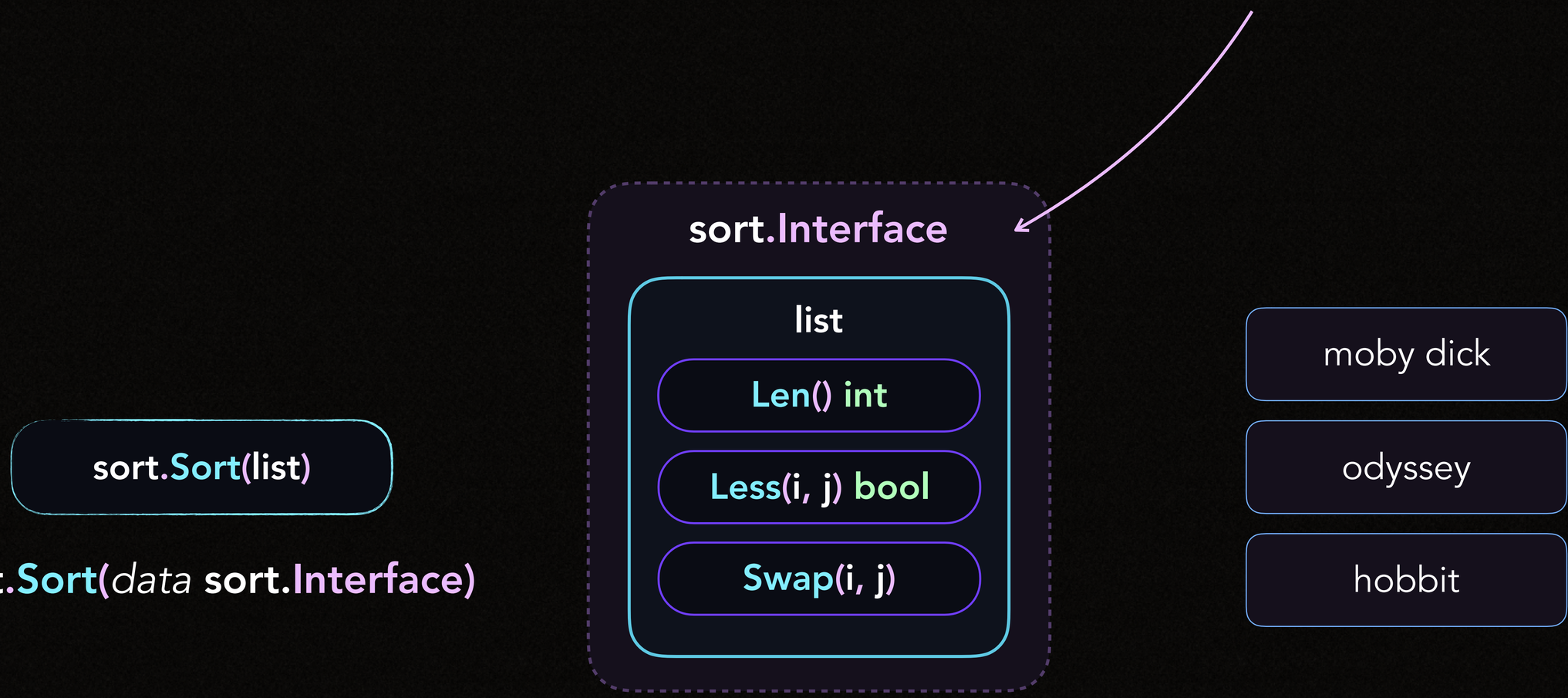
list satisfies the sort.Interface



sort.Sort(*data* sort.Interface)

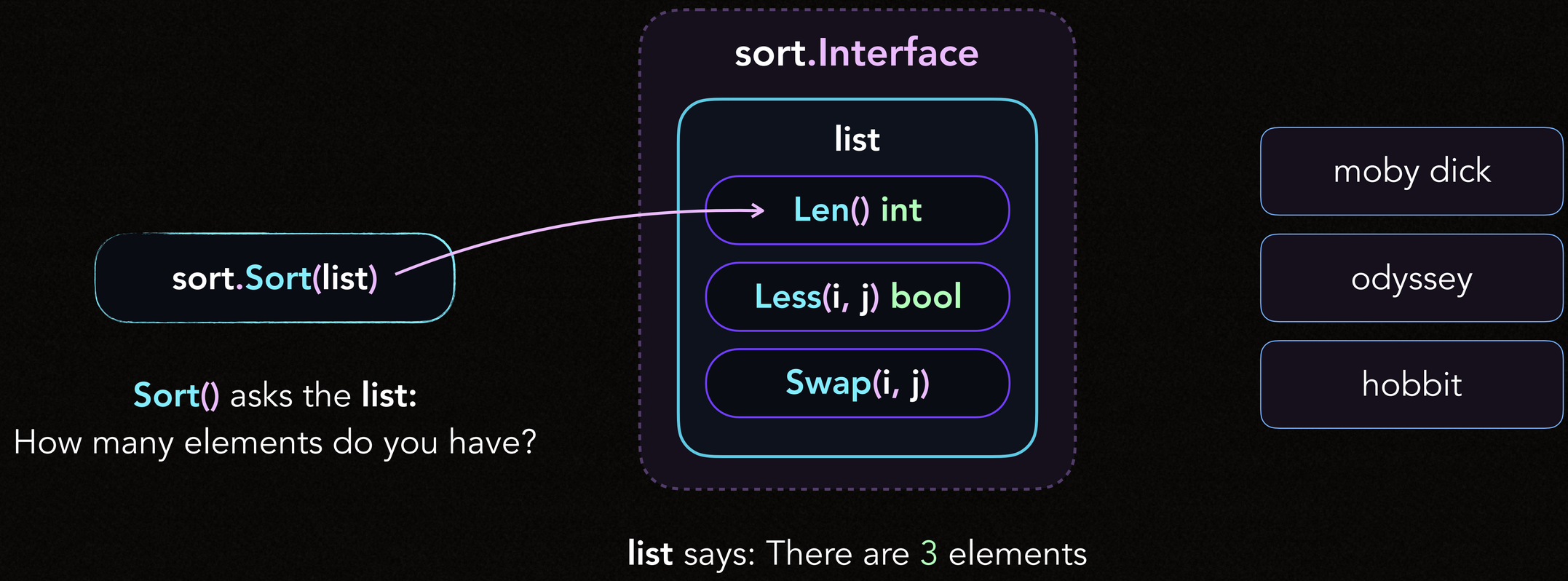
HOW DOES IT WORK?

sort.Sort(data sort.Interface) wraps the list in a sort.Interface value



HOW DOES IT WORK?

sort.Sort() can sort a type that satisfies the sort.Interface

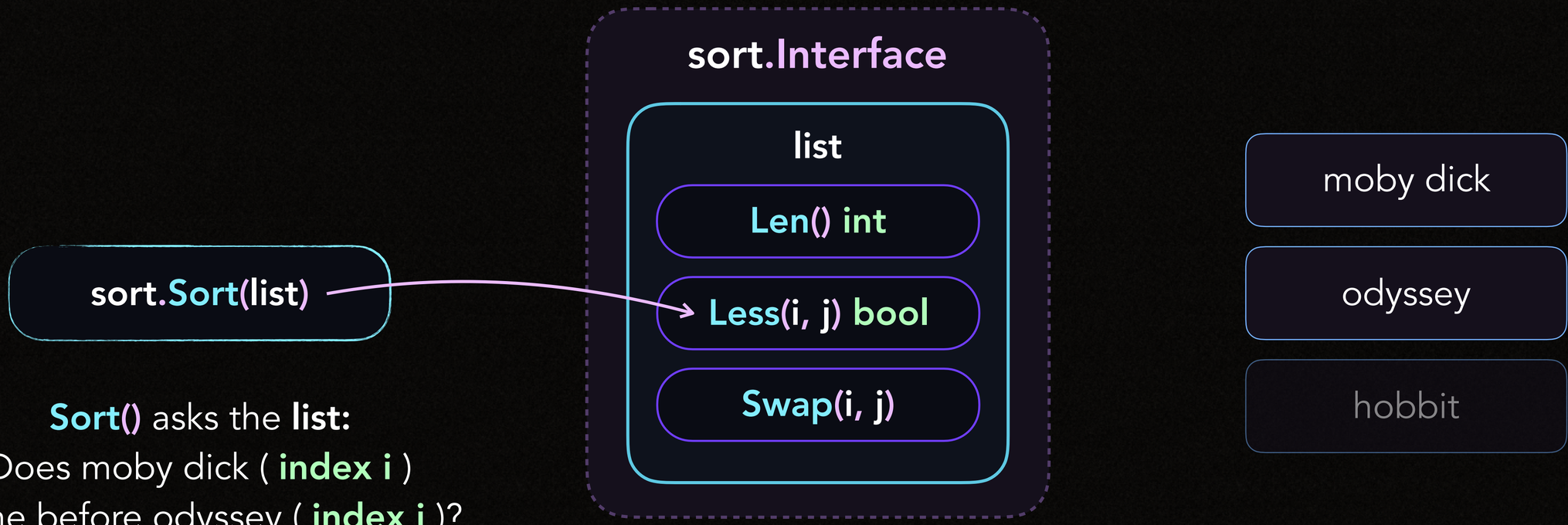


Sort() asks the **list:**
How many elements do you have?

list says: There are 3 elements

HOW DOES IT WORK?

sort.Sort() runs list.Less(0, 1) — list returns true

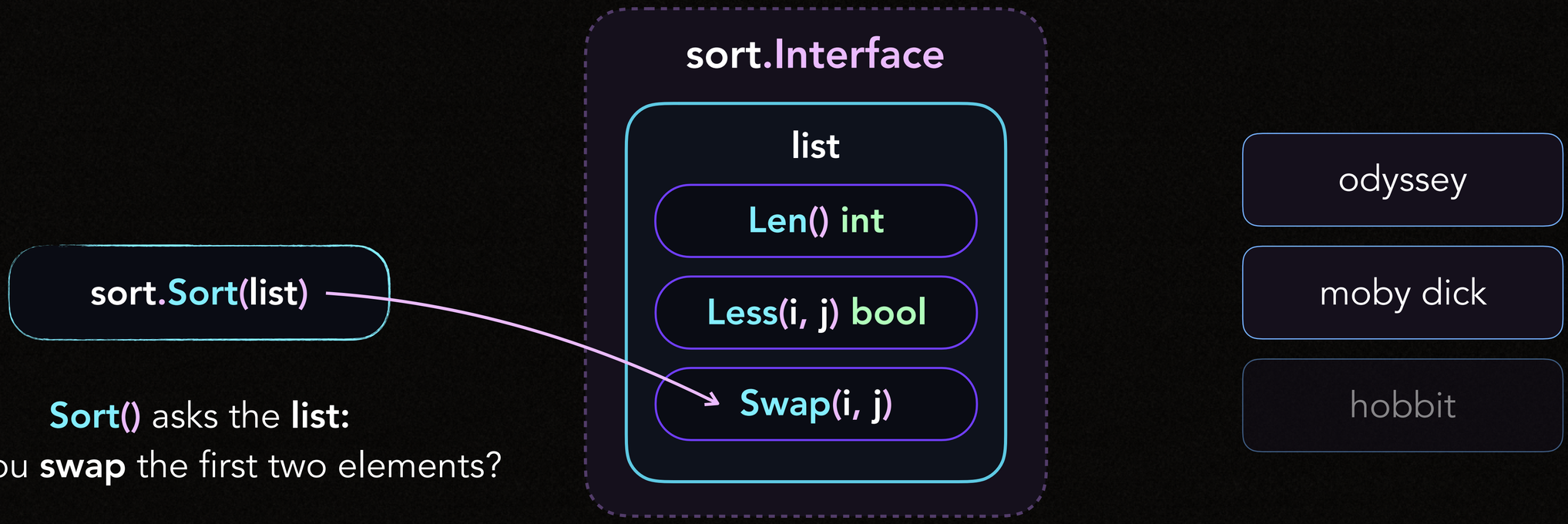


Sort() asks the **list:**
Does moby dick (**index i**)
come before odyssey (**index j**)?

list says: **true — sort the elements**
"moby dick" comes before "odyssey"

HOW DOES IT WORK?

sort.Sort() runs list.Swap(0, 1)



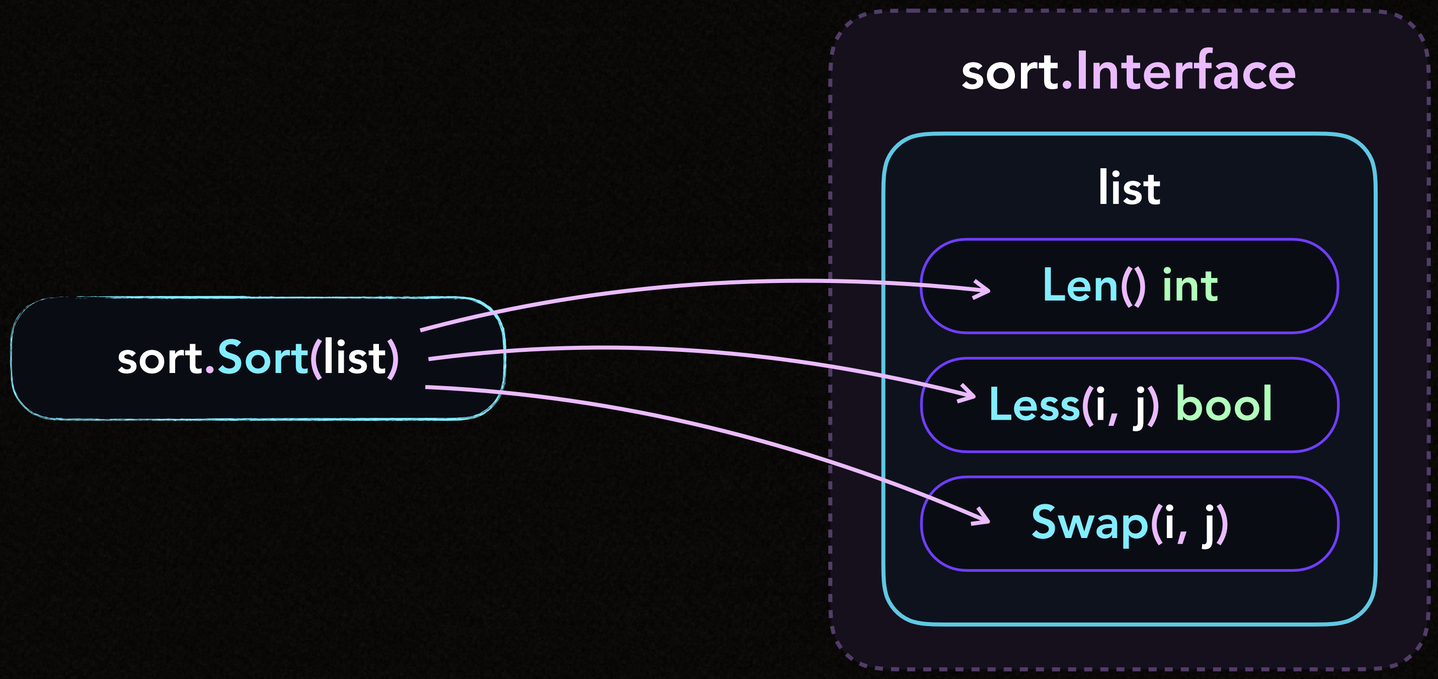
Sort() asks the **list:**
Can you **swap** the first two elements?

list swaps the order of
"moby dick" and "odyssey"

HOW DOES IT WORK?

Algorithm Abstraction

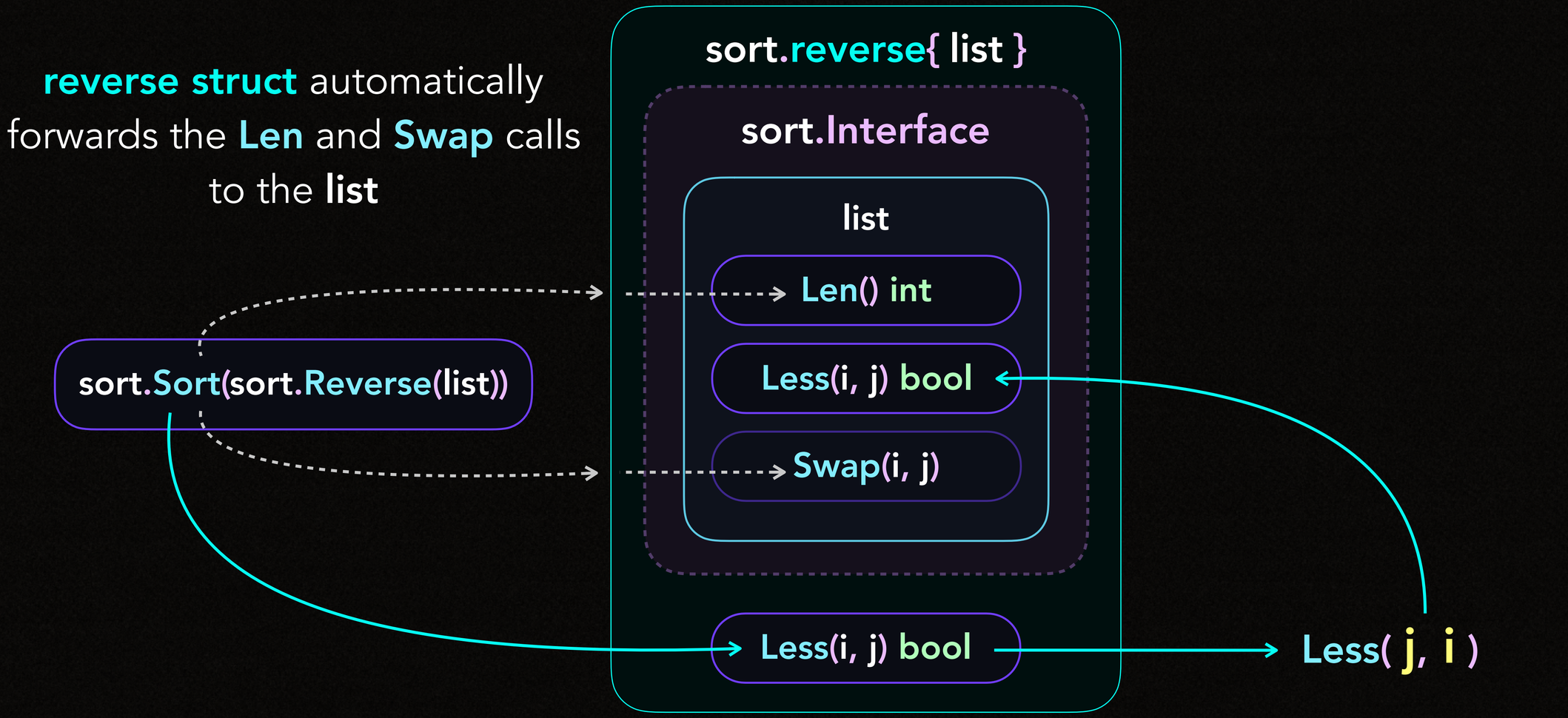
sort package can sort the **list**
using the **sort.Interface**



The length of the list

Whether an element comes before the other one

Tells the list to swap the elements



SUMMARY

★ **sort.Sort() can sort any type that implements the sort.Interface**

★ **sort.Interface has three methods: Len(), Less(), Swap()**

- **Len()** returns the length of a collection.
- **Less(i, j)** should return **true** when an element comes before another one.
- **Swap(i, j)**s the elements when the **Less()** returns **true**.

★ **sort.Reverse()** can reverse sort a type that satisfies the **sort.Interface**.

★ **You can customize the sorting:**

- ***Either*** by implementing the **sort.Interface** methods,
- or by **anonymously embedding** a type that already satisfies the **sort.Interface**
- and adding a **Less()** method.

★ **Anonymous embedding** means **auto-forwarding** method calls to an embedded type.

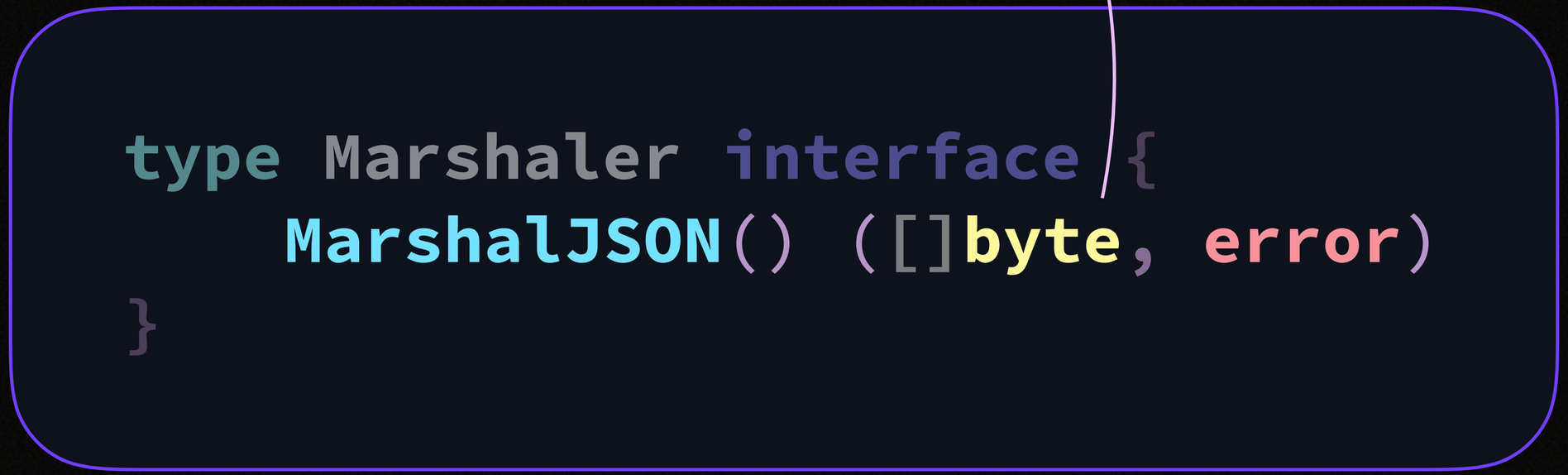
- Check out the source-code for details.

MARSHALERS

Customize JSON encoding and decoding of a type

```
type Marshaler interface {
    MarshalJSON() ([]byte, error)
}
```

You need to encode a timestamp as a number and return it as a []byte

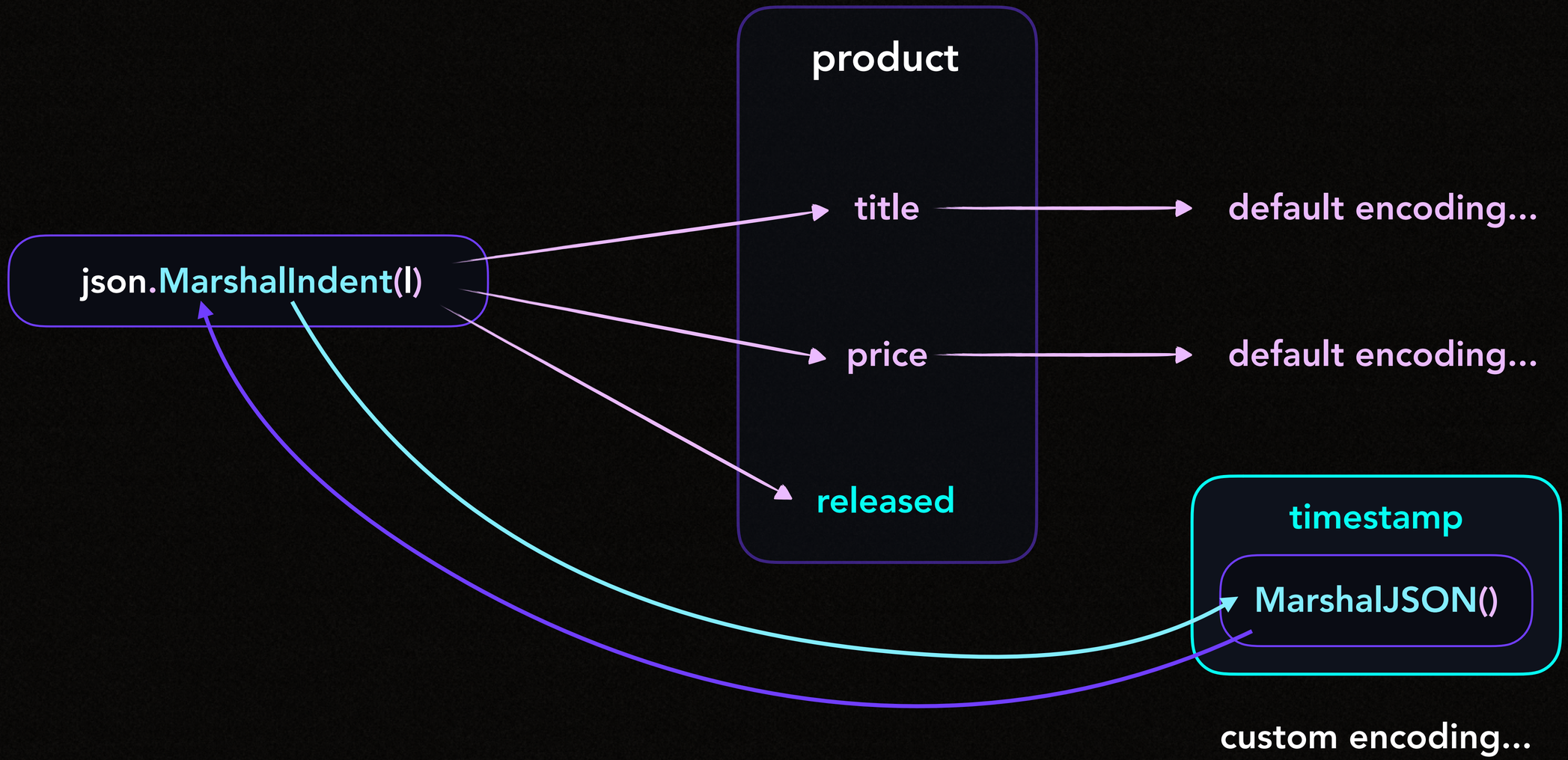


```
type Marshaler interface {
    MarshalJSON() ([]byte, error)
}
```

You can satisfy it to customize the JSON encoding of a type

HOW DOES IT WORK?

Encoder calls MarshalJSON() if a type is a json.Marshaler



```
type Unmarshaler interface {
    UnmarshalJSON([]byte) error
}
```

You can satisfy it to customize the JSON decoding of a type

```
type Unmarshaler interface {
    UnmarshalJSON([]byte) error
}
```

It will receive JSON encoded data (*timestamp*)

PART VI

Go OOP: Methods and Interfaces

Methods ✓

Value Receivers

Pointer Receivers

Non-Struct Methods

Interfaces ✓

Type Assertion

Type Switch

Empty Interface

Promoted Methods

Advanced Interfaces ✓

Stringer

Sorter

Marshaler

Unmarshaler

Streaming Interfaces

PNG Detector

Stdin & Stdout

io.Reader

io.Writer

+ Unit Testing +

SUMMARY

★ **json.Marshal() and json.MarshalIndent() can only encode primitive types.**

- Custom types can tell the encoder how to encode.
- To do that satisfy the **json.Marshaler** interface.

★ **json.Unmarshal() can only decode primitive types.**

- Custom types can tell the decoder how to decode.
- To do that satisfy the **json.Unmarshaler** interface.

★ **strconv.AppendInt() can append an int value to a []byte.**

- There are several other functions in the **strconv package** for other primitive types as well.
- Do not make unnecessary **string** <-> **[]byte** conversions.

★ **log.Fatal() can print the given error message and terminate the program.**

- Use it only from the **main()**. Do not use it in other functions.
- **main()** is should be the **main** driver of a program.