PART IV

Composite Types

Gopher Level

Intermediate

PART IV

Composite Types

✓ **Arrays**
Collection
of
Elements
Indexable
Fixed Length

✓ **Slices**
Collection
of
Elements
Indexable
Dynamic length

✓ **Strings Revisited**
Read-Only Byte Slices
Indexable
Fixed Length

✓ **Maps**
Collection
of
Indexable
Key-Value Pairs

Structs
Groups
different types
in a
single type

STRUCTS

Structs allow us to represent concepts

Person
Name
Lastname
Age

Movie
Title
Genre
Rating

Rental
Address
Rooms
Size
Price

STRUCTS

Structs allows us to represent concepts

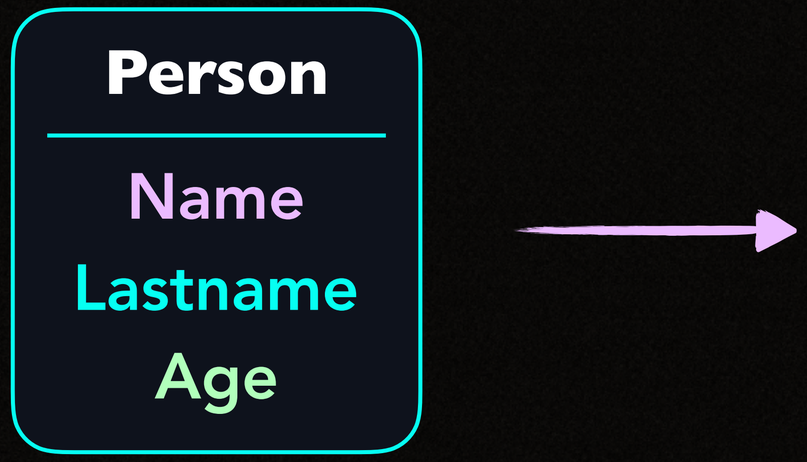
Person
Name
Lastname
Age

Movie
Title
Genre
Rating

Rental
Address
Rooms
Size
Price

STRUCTS

Structs are blueprints — They are fixed at compile-time



It's like a **blueprint**

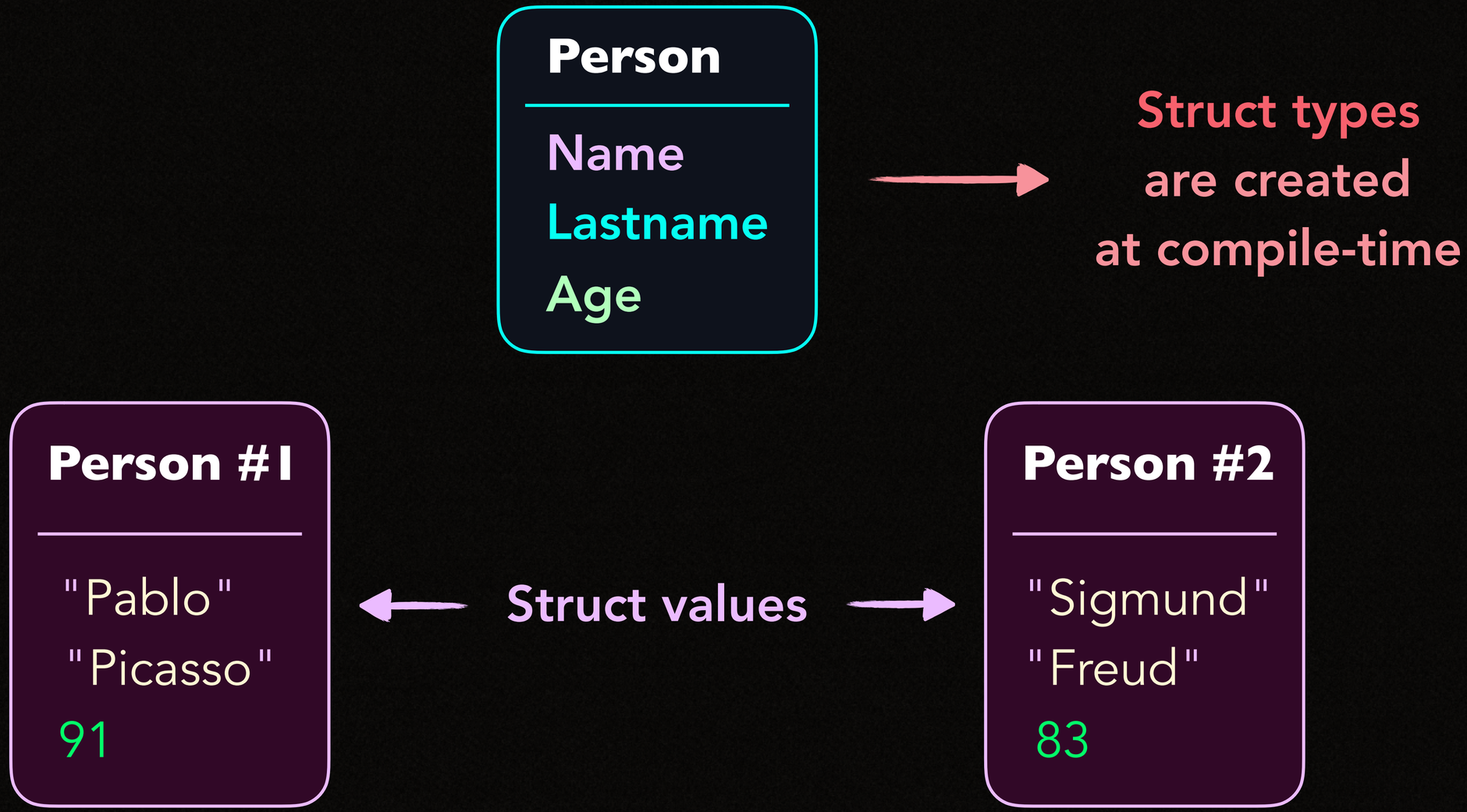
It's like a **class** in OOP languages

Groups related data in a **single type**

Fixed at compile-time

STRUCTS

From a single static blueprint (*struct type*) you can create struct values



STRUCT FIELDS

A struct may store different types of data

Field Names	Field Types	Field Values
Name	string	"Pablo"
Lastname	string	"Picasso"
Age	int	91

STRUCT FIELDS

Struct fields declared at compile-time; struct values fill them in runtime

Field Names	Field Types	Field Values
Name	string	"Pablo"
Lastname	string	"Picasso"
Age	int	91

The field names and types are declared at compile-time
They are fixed and cannot change in runtime

STRUCT FIELDS

Struct fields declared at compile-time; struct values fill them in runtime

Field Names	Field Types	Field Values
Name	string	"Pablo"
Lastname	string	"Picasso"
Age	int	91

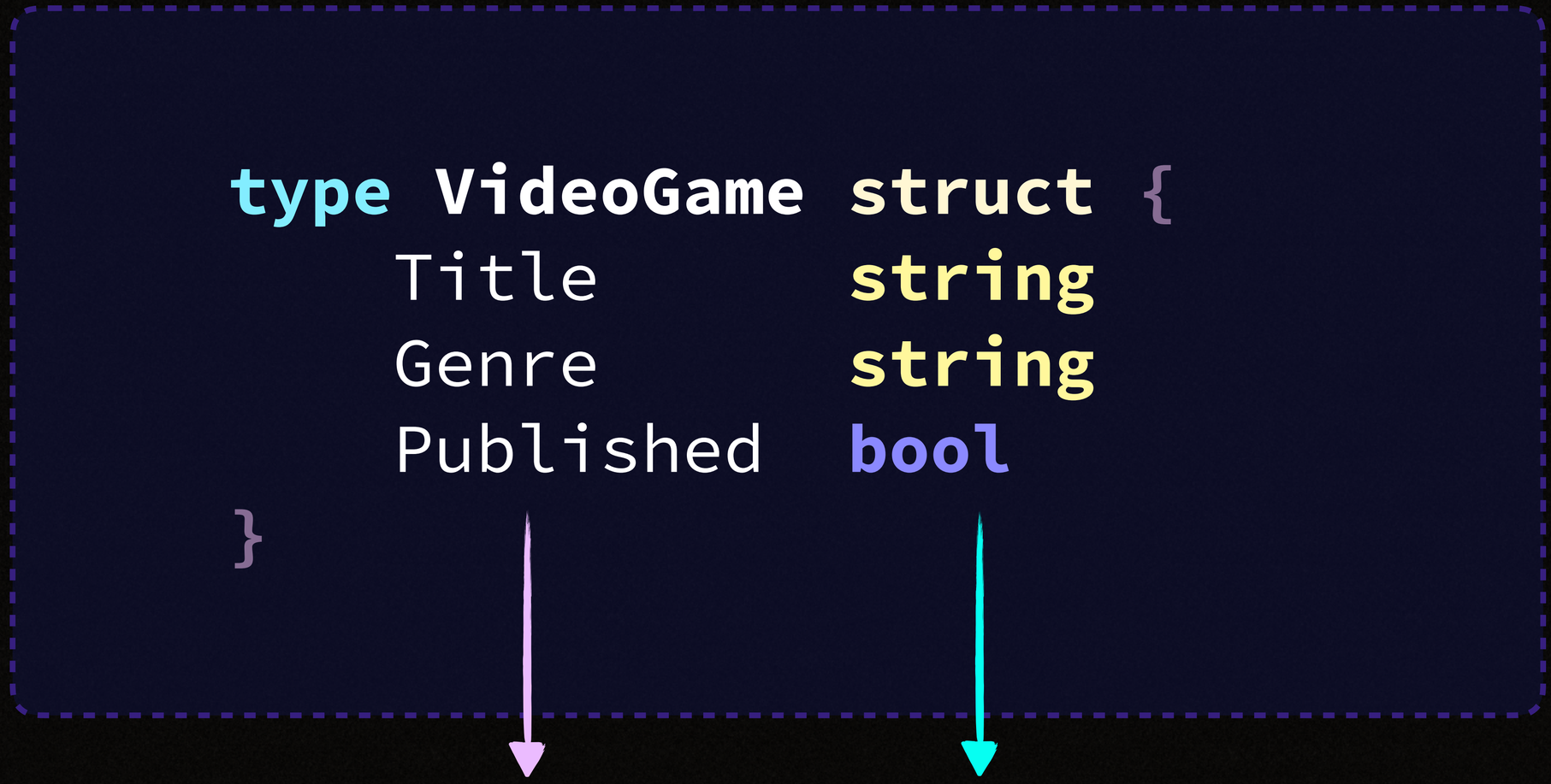
Field values belong to runtime
You can change them in runtime

SUMMARY

Structs cannot dynamically grow but they can have different set of types

SLICE & MAP	STRUCT
☹️ Single Element Type	🥳 Different Types of Fields
🥳 Dynamic Number of Elements	☹️ Fixed Number of Fields

```
type VideoGame struct {
    Title       string
    Genre       string
    Published   bool
}
```



```
type VideoGame struct {
    Title       string
    Genre       string
    Published   bool
}

pacman := VideoGame{
    Title:      "Pac-Man",
    Genre:      "Arcade Game",
    Published: true,
}
```

```
type VideoGame struct {
    Title       string
    Genre       string
    Published   bool
}

pacman := VideoGame{
    Title:      "Pac-Man",
    Genre:      "Arcade Game",
    Published: true,
}
```

```
type VideoGame struct {
    Title       string
    Genre       string
    Published   bool
}

pacman := VideoGame{
    Title:      "Pac-Man",
    Genre:      "Arcade Game",
    Published: true,
}
```

STRUCT TYPES & VALUES

Struct types are just the *blueprints* — Struct values are *alive*

COMPARISON & ASSIGNMENT

Structs are bare value types

STRUCTS

Structs are bare values *(just like arrays)*

Song Struct	
Title:	" "
Artist:	" "

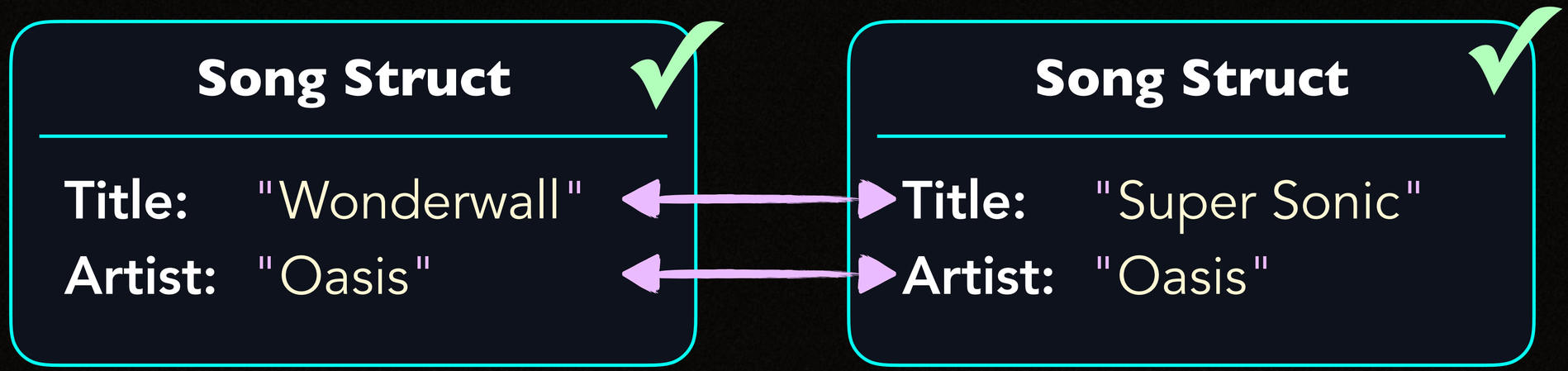
Song Struct	
Title:	"Wonderwall"
Artist:	"Oasis"

STRUCTS

Structs values are copied along with their fields

EQUAL?

Two structs are equal if all their fields are equal



==

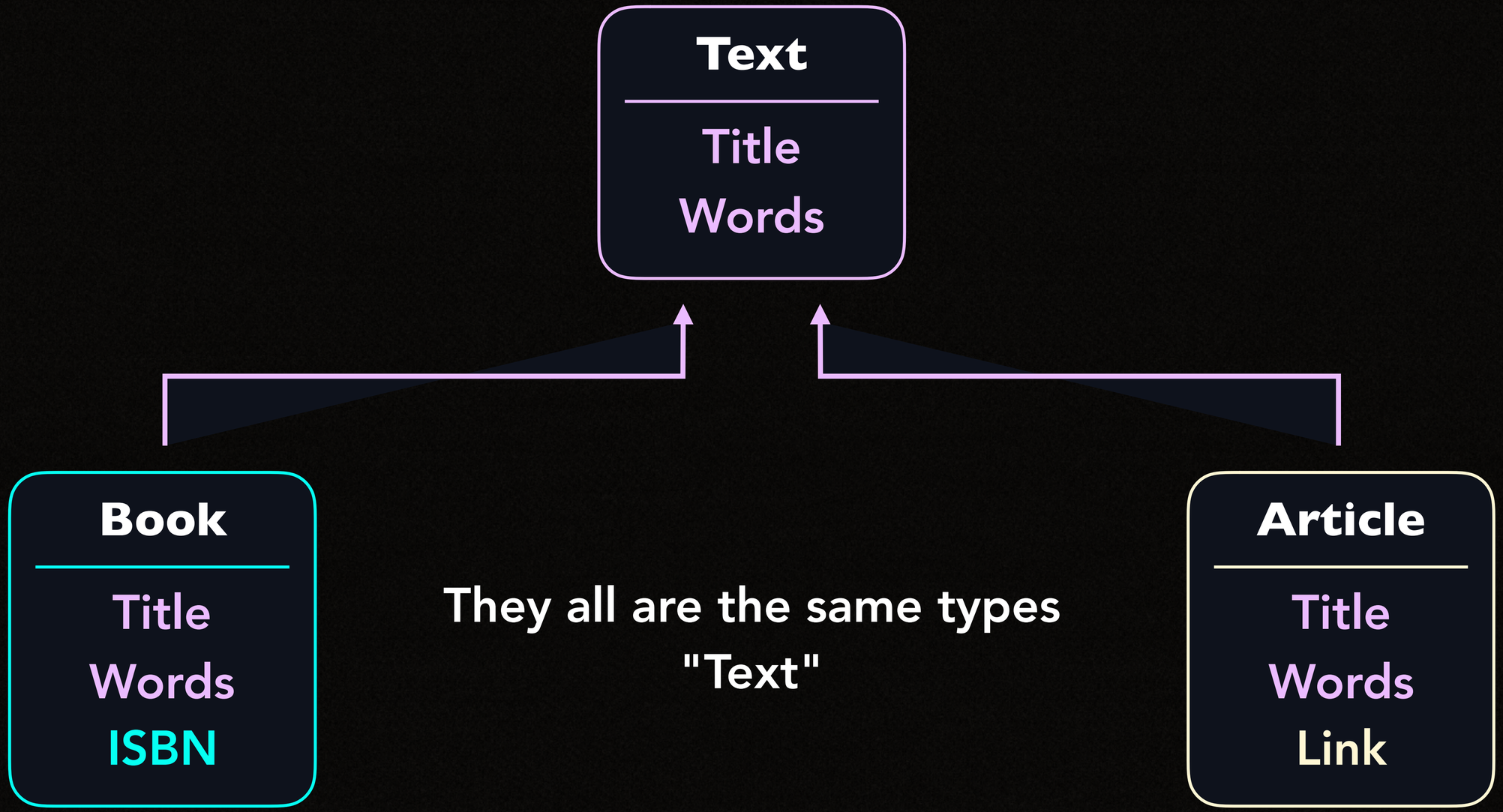
The types should be identical

EMBEDDING

Object-Oriented Composition with Go Sauce

INHERITANCE

"is-a" relationship: book is-a text — article is-a text



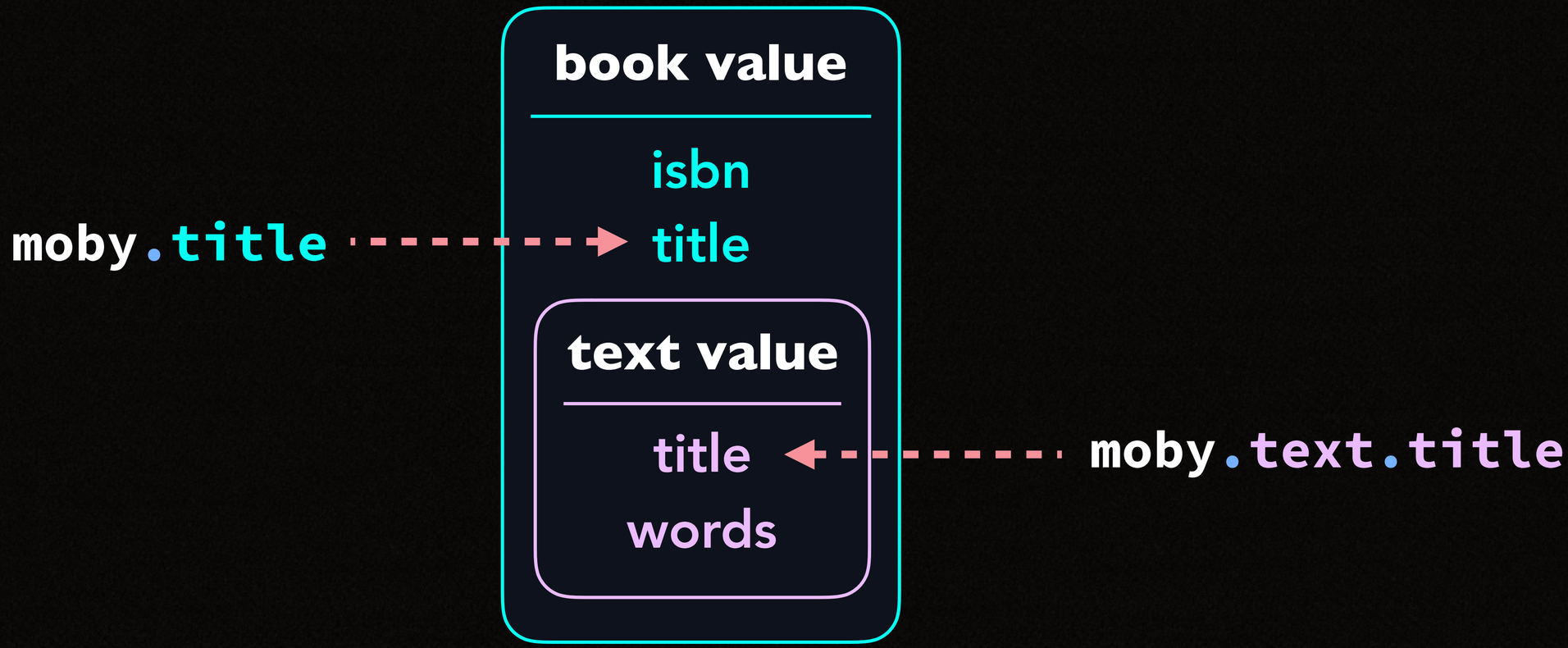
EMBEDDING

"has-a" relationship: book has text — article has text

Each one is a different type

ANONYMOUS FIELDS

When the field names conflict the parent type takes priority



LOG PARSER

Let's enhance the log parser using structs!

JSON ENCODING

Encode JSON data using structs, maps and slices

JSON ENCODING

Why you might want to use JSON data exchange format?

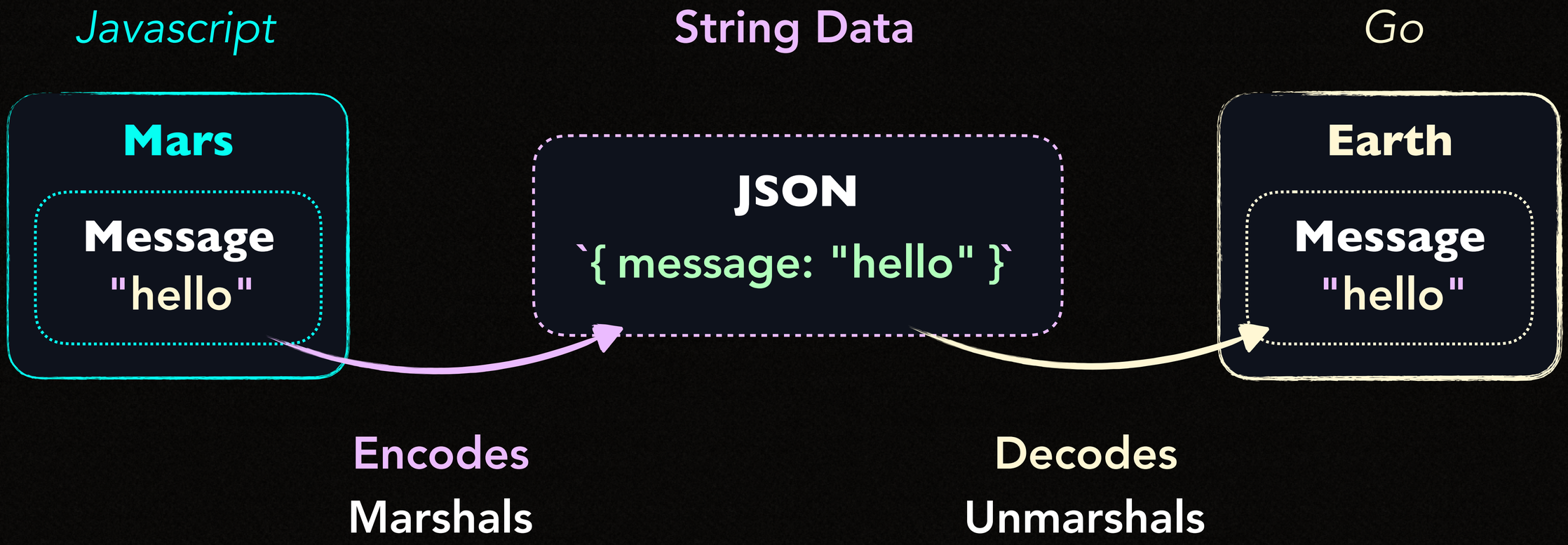
Human Readable

Computers can easily to understand it

Widely used and supported

JSON ENCODING

JSON is a structured data exchange format



FIELD TAGS

Associates a static string metadata to a field

```
type user struct {
    Name        string      `json:"username"`
    Password    string      `json:"-"`
    Permissions permissions `json:"perms,omitempty"`
}
```

Mostly used for controlling the encoding/decoding behavior

FIELD TAGS

It doesn't have a meaning on its own

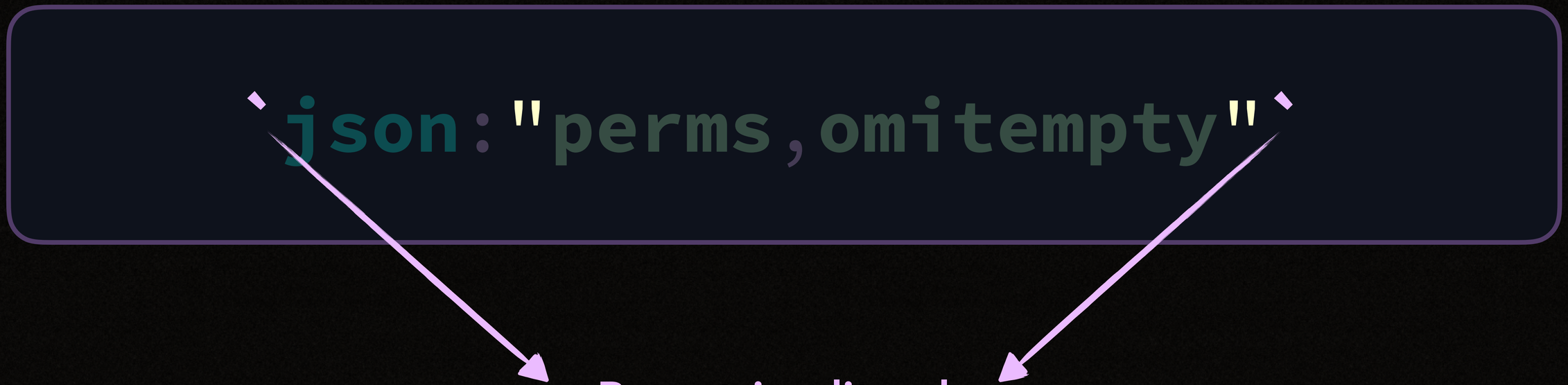
FIELD TAGS

It doesn't have a meaning on its own

```
`json:"perms,omitempty"`
```

FIELD TAGS

It doesn't have a meaning on its own



Raw string literal

Usually, double-quotes are used in a field tag

FIELD TAGS

json package only reads a field tag if it starts with: *json:*

Key Name

Usually it denotes a package name

Here: the json package will read this tag

FIELD TAGS

Use the options to tell the json package how to encode and decode a field

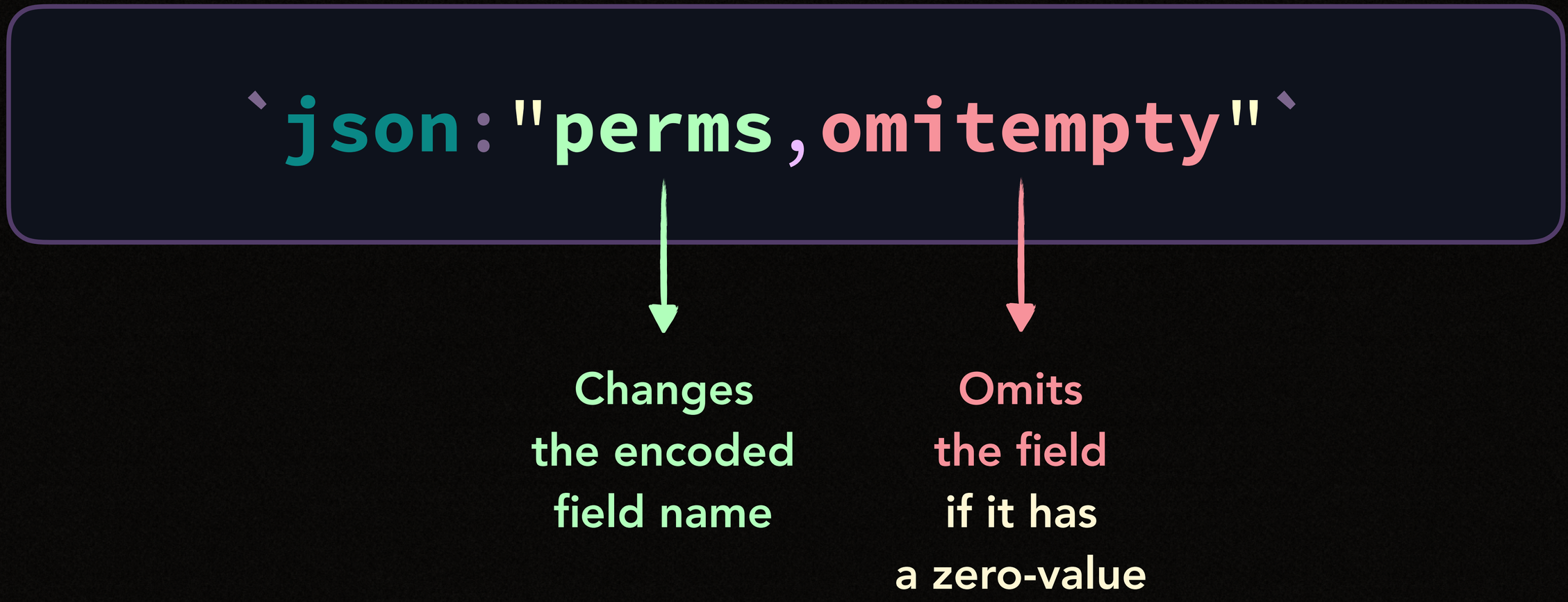
```
`json:"perms,omitempty"`
```

Value

Comma-separated Options

FIELD TAGS

Use the options to tell the json package how to encode and decode a field

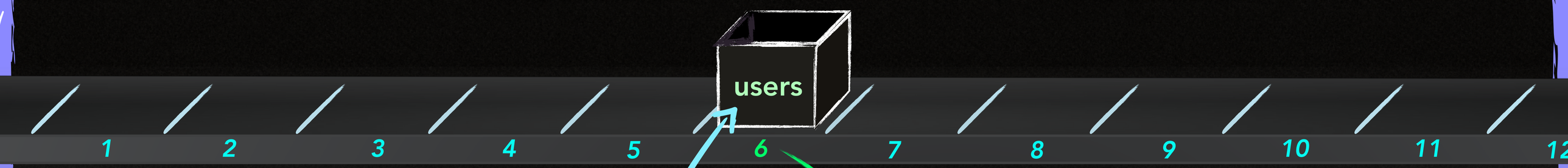


JSON DECODING

Decode JSON data using structs, maps and slices

POINTERS

& finds the memory address of a value (or a variable)



changes the value
through its address

```
json.Unmarshal(input, &users)
```

PART IV

Composite Types

Arrays
Collection
of
Elements
Indexable
Fixed Length

Slices
Collection
of
Elements
Indexable
Dynamic length

Strings Revisited
Read-Only Byte Slices
Indexable
Fixed Length

Maps
Collection
of
Indexable
Key-Value Pairs

Structs
Groups
different types
in a
single type