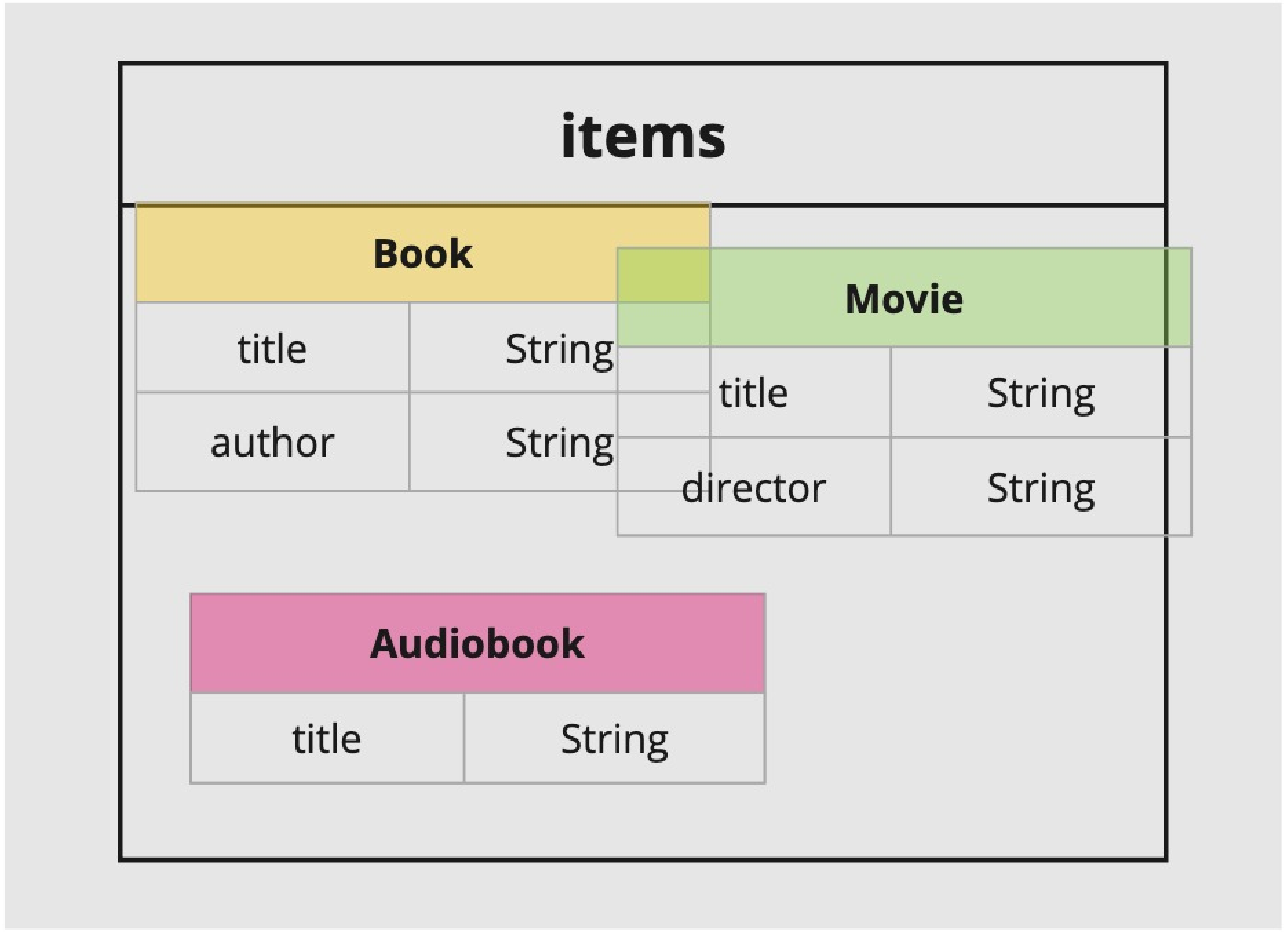
Bank

Description	Method or Assoc. Func?	Name	Args	Returns
Create a 'Bank' instance	Assoc. Func	new()	-	Bank
Add an account to the list of accounts	Method	add_account()	account: Account	-
Calculate the total balance of all accounts	Method	total_balance()	-	i32
Create a Vec containing the summaries of all accounts	Method	summary()	-	Vec<String>

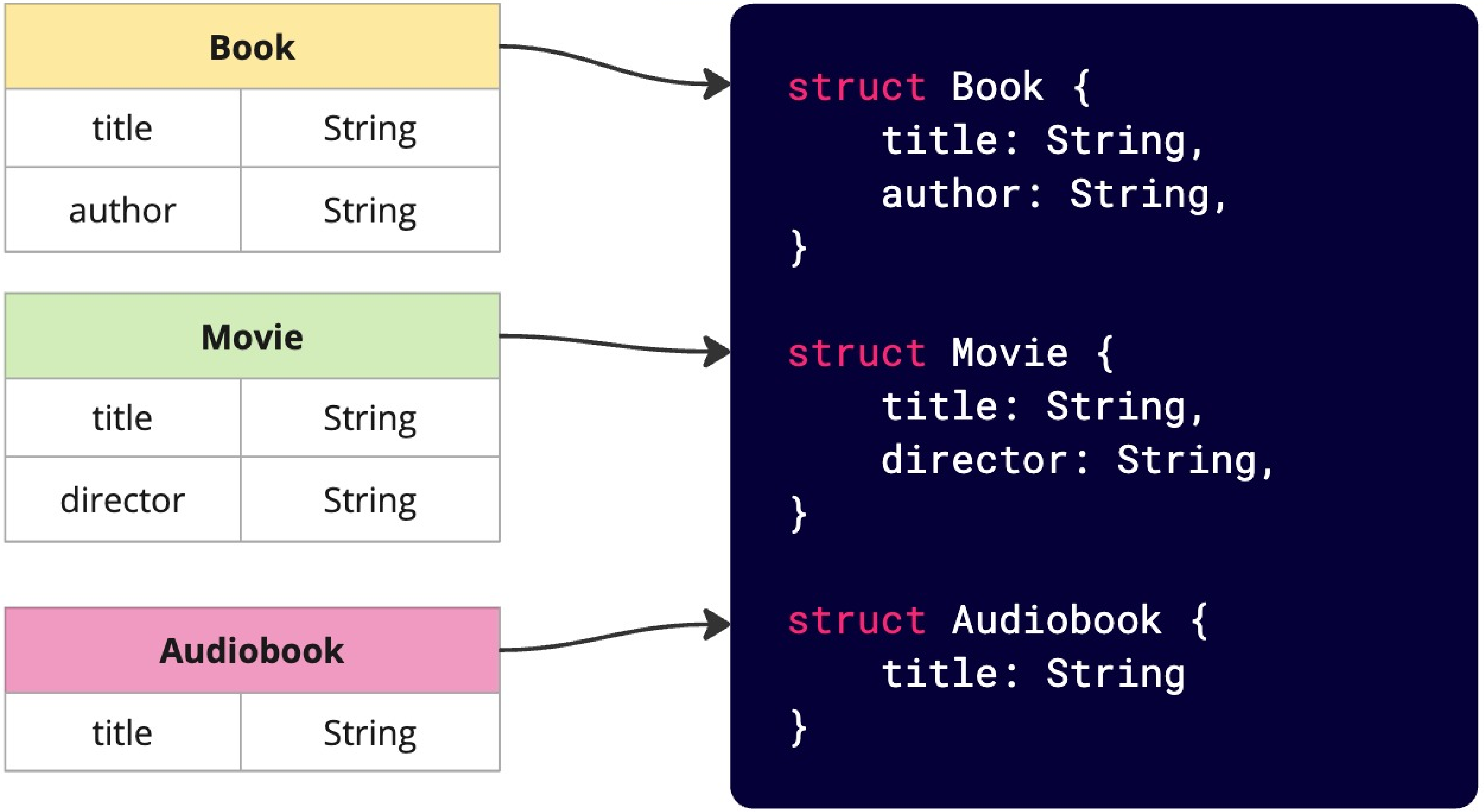
Account

Description	Method or Assoc. Func?	Name	Args	Returns
Create an 'Account' instance	Assoc. Func	new()	id: u32 holder: String	Account
Add the given amount of money to the accounts 'balance'	Method	deposit()	amount: i32	i32
Remove the given amount of money from the accounts 'balance'.	Method	withdraw()	amount: i32	i32
Create an account summary as a string and return it	Method	summary()	-	String

Catalog


items
Book
title
String
author
String
Movie
title
String
director
String
Audiobook
title
String

One way we could model Books, Movies, and Audiobooks



Book	
title	String
author	String

Movie	
title	String
director	String

Audiobook	
title	String

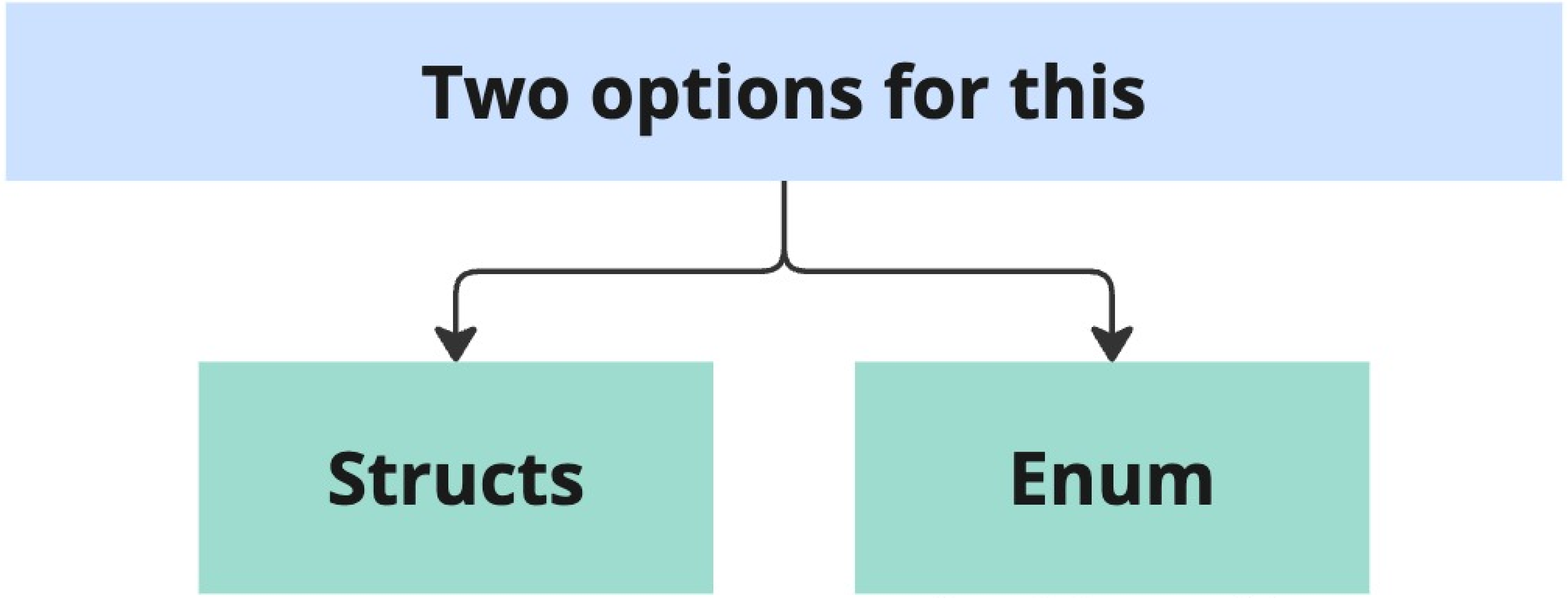
```
struct Book {
    title: String,
    author: String,
}

struct Movie {
    title: String,
    director: String,
}

struct Audiobook {
    title: String
}
```

Book	
title	String
author	String

Movie	
title	String
director	String

Audiobook	
title	String

We need to model several different things that are all kind of similar



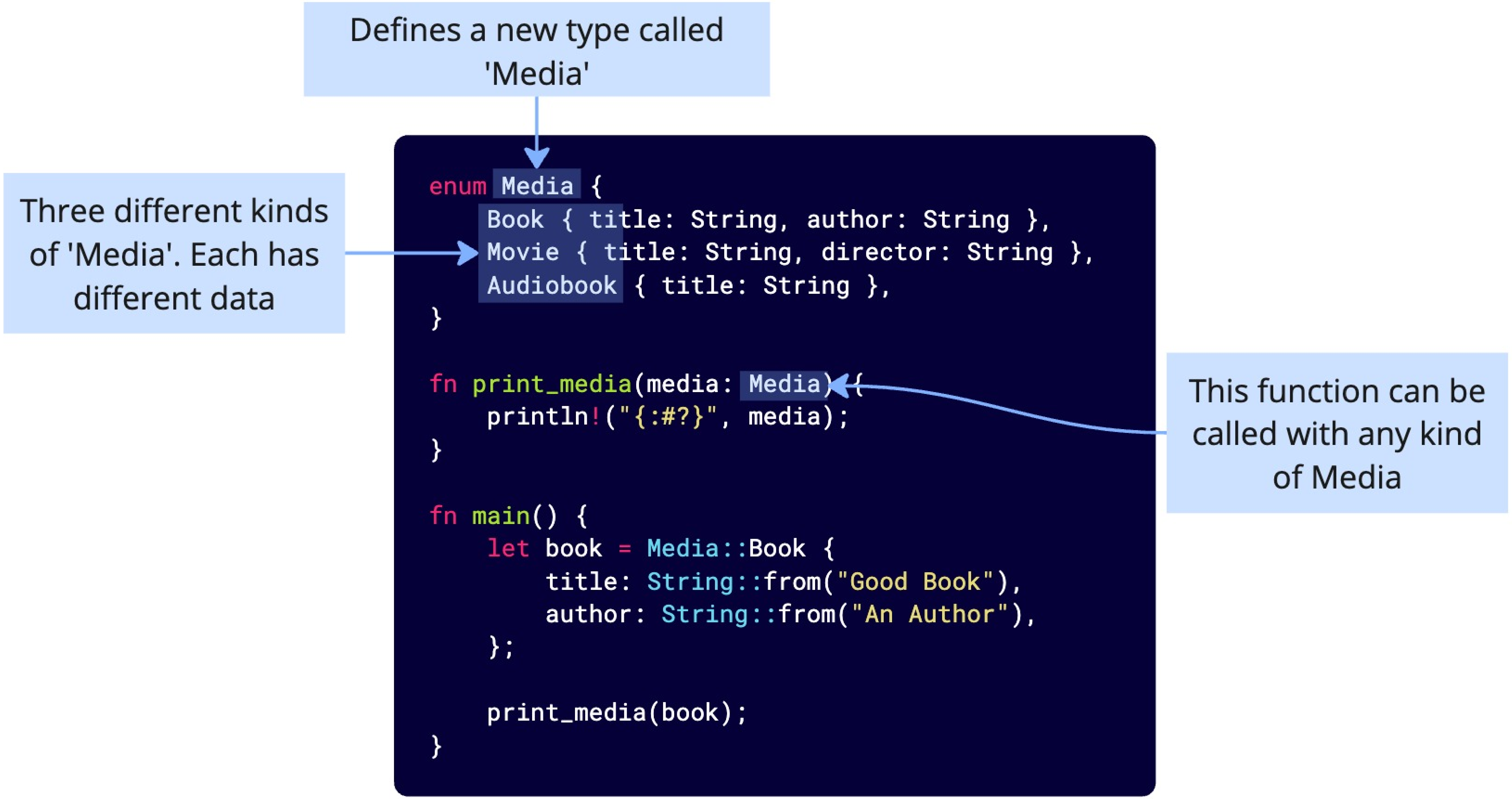
Enums in Rust are a little different than enums in other languages

We can ***imagine***
that this creates three structs

Book, Movie, and Audiobook are all of type 'Media'

We can define functions that accept values of type 'Media', and put in a Book or a Movie or an Audiobook

```
enum Media {
    Book { title: String, author: String },
    Movie { title: String, director: String },
    Audiobook { title: String },
}
```


Defines a new type called 'Media'
Three different kinds of 'Media'. Each has different data
This function can be called with any kind of Media

```
enum Media {
    Book { title: String, author: String },
    Movie { title: String, director: String },
    Audiobook { title: String },
}

fn print_media(media: Media) {
    println!("{:#?}", media);
}

fn main() {
    let book = Media::Book {
        title: String::from("Good Book"),
        author: String::from("An Author"),
    };

    print_media(book);
}
```

Structs

```rust
struct Book {
    title: String,
    author: String,
}

struct Movie {
    title: String,
    director: String,
}

struct Audiobook {
    title: String
}
```

Implementations

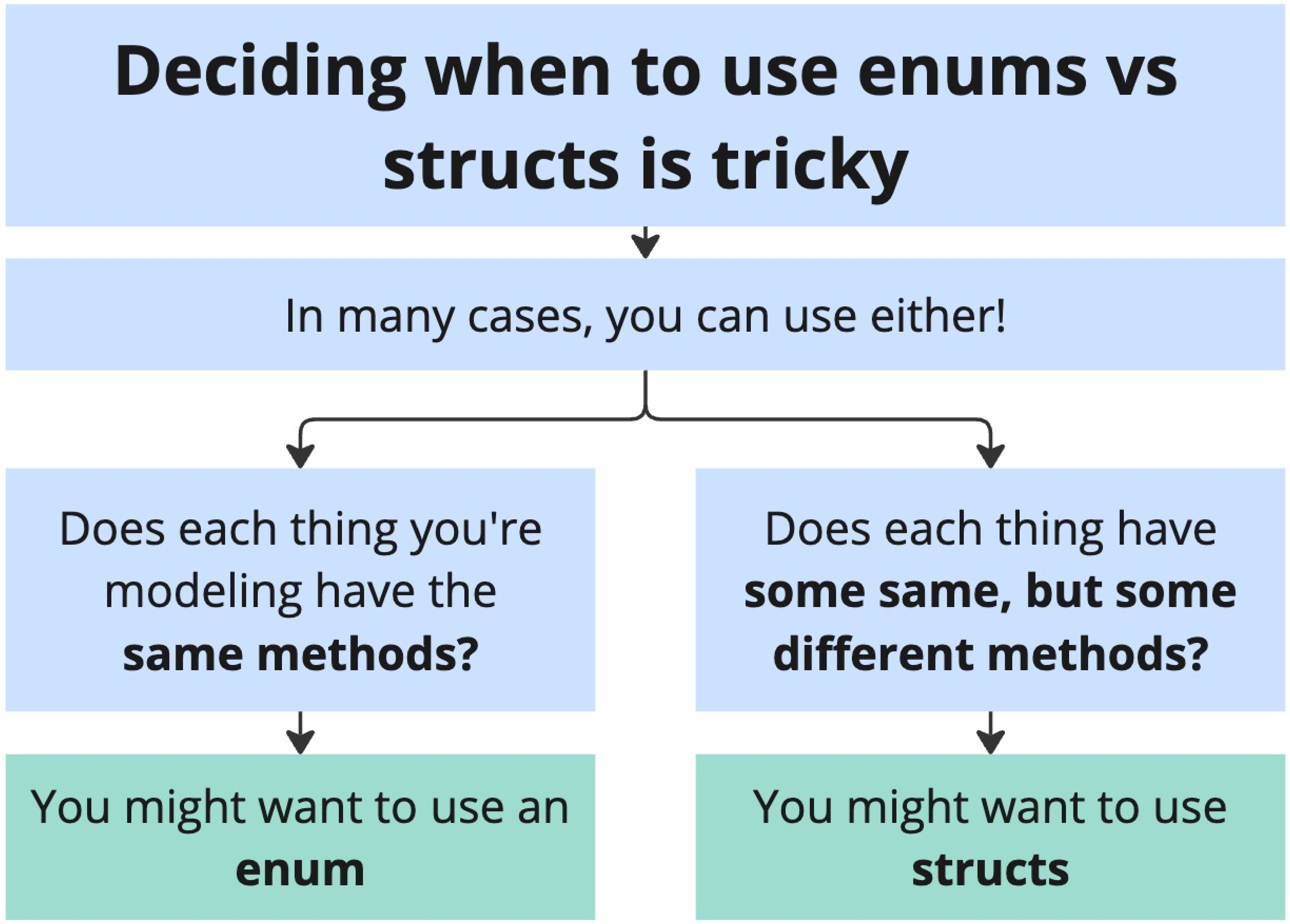
```rust
impl Book {
    fn description(&self) -> String {
        format!("Book: {} {}", self.title, self.author)
    }
}

impl Movie {
    fn description(&self) -> String {
        format!("Movie: {} {}", self.title, self.director)
    }
}

impl Audiobook {
    fn description(&self) -> String {
        format!("Audiobook: {}", self.title)
    }
}
```

```
book.description() // "Book called 'A Biography' by Jane"

movie.description() // "Movie called 'Action!' by John"

audiobook.description() // "Audiobook called 'Fun Time'"
```

Books, movies, and audiobooks should have a 'description' method

'description' should work slightly differently depending on whether we're working on a book, movie, or audiobook

```rust
impl Media {
    fn description(&self) -> String {
        match self {
            Media::Book { title, author } => format!("Book: {} {}", title, author),
            Media::Movie { title, director } => format!("Movie: {} {}", title, director),
            Media::Audiobook { title } => format!("Audiobook: {}", title),
        }
    }
}
```



Deciding when to use enums vs structs is tricky

In many cases, you can use either!

Does each thing you're modeling have the **same methods?**

You might want to use an **enum**

Does each thing have **some same, but some different methods?**

You might want to use **structs**

Book	
title	String
author	String

Movie	
title	String
director	String

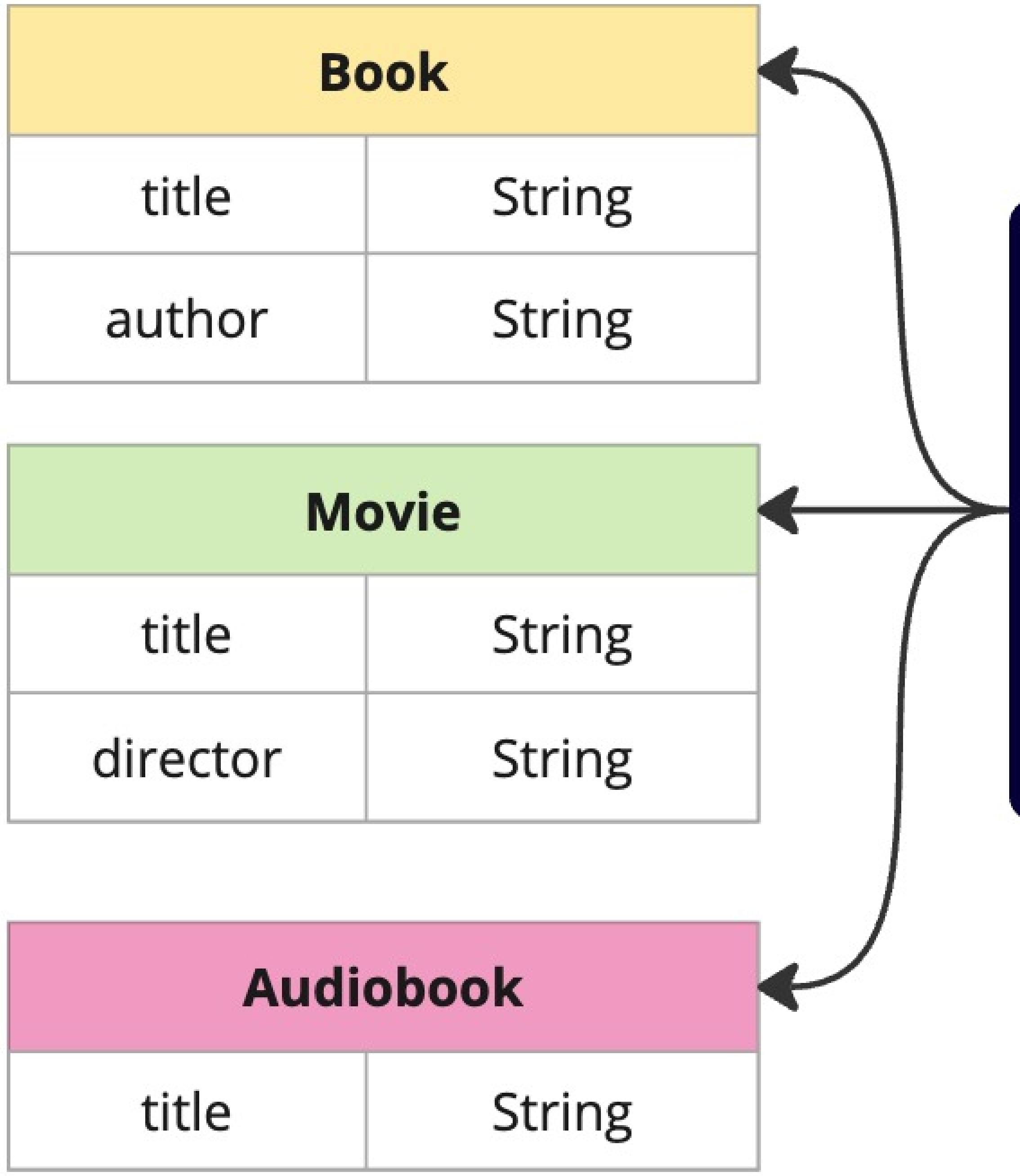
Audiobook	
title	String

For our app, as described, each thing will have very few methods

```
book.description() // "Book called 'A Biography' by Jane"

movie.description() // "Movie called 'Action!' by John"

audiobook.description() // "Audiobook called 'Fun Time'"
```

Every thing has the exact same set of methods

Probably want to use an **enum**



Book	
title	String
author	String

Movie	
title	String
director	String

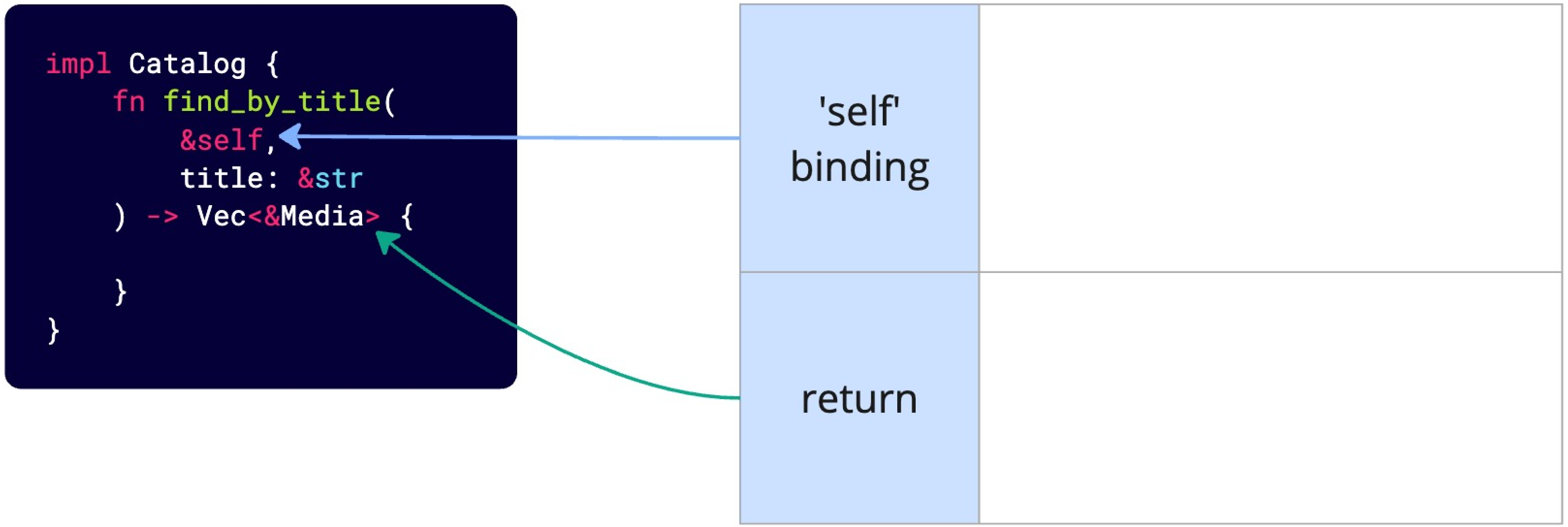
Audiobook	
title	String

If our app was more complex, and each thing different methods...

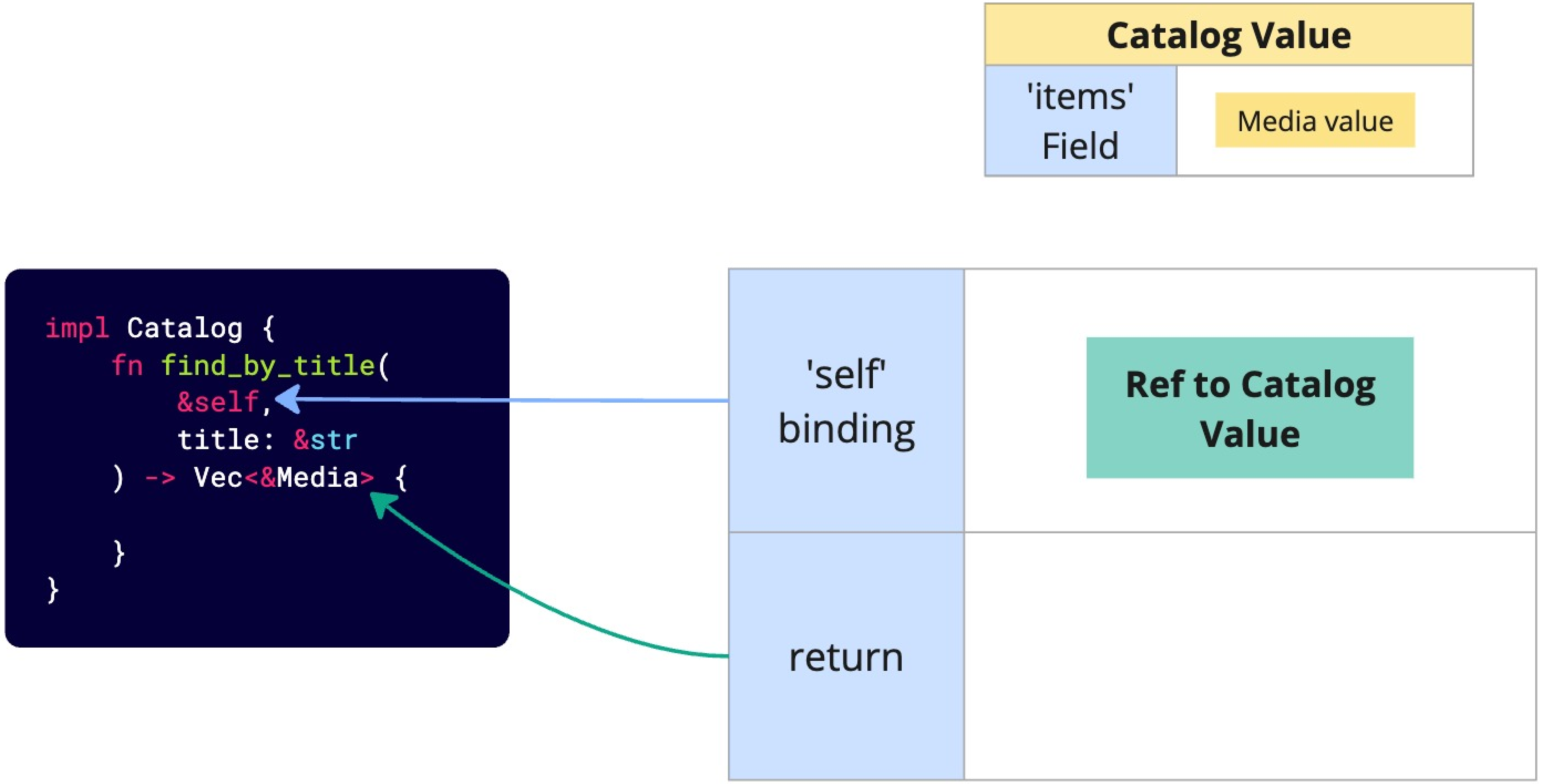
```
book.description() // "Book called 'A Biography' by Jane"
book.read(); // A book can be 'read'

movie.description() // "Movie called 'Action!' by John"
movie.play(); // A movie can be 'played'

audiobook.description() // "Audiobook called 'Fun Time'"
audiobook.listen(); // An audiobook can be 'listened'
```

Each thing has some **different methods**

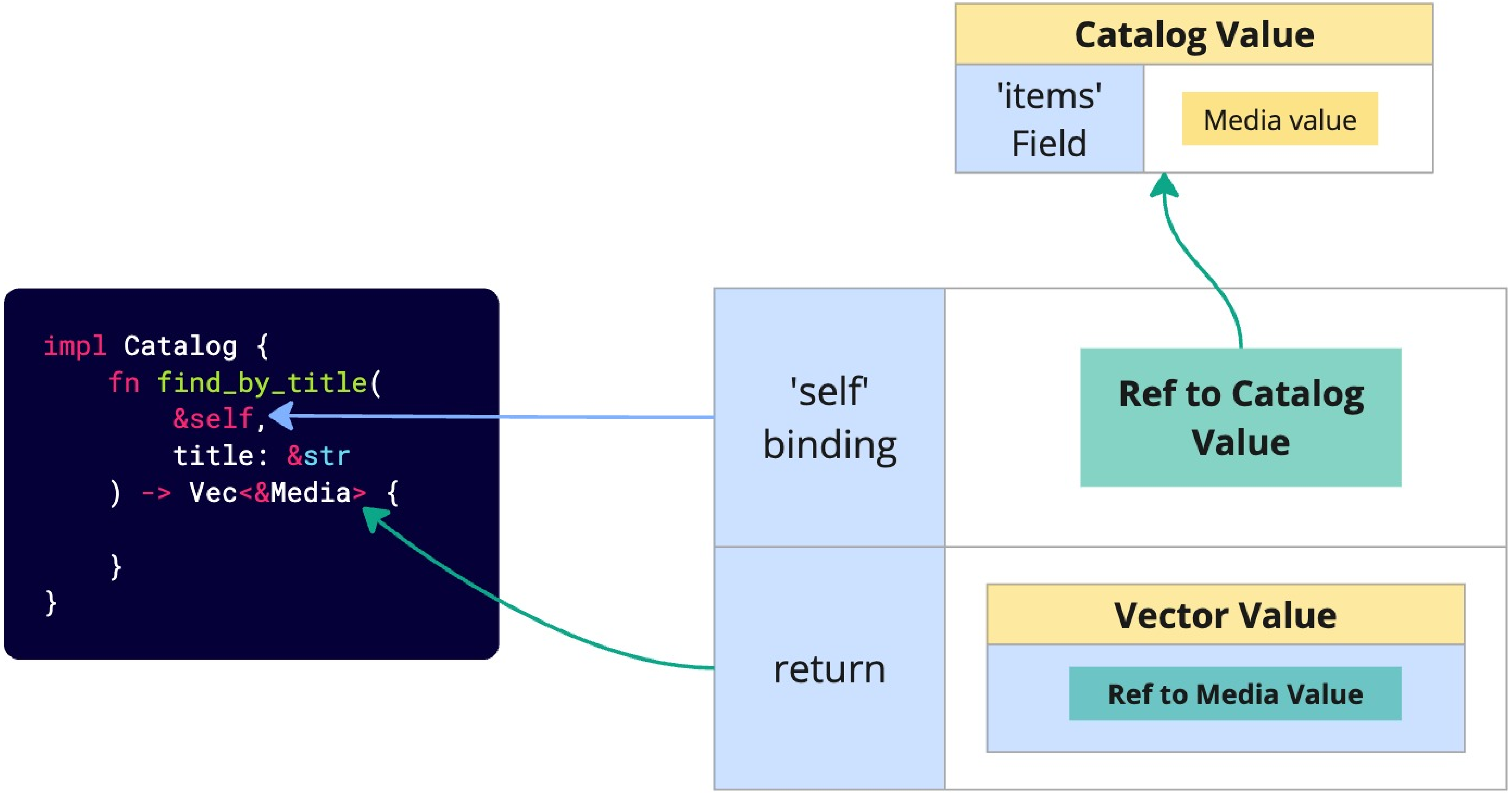
Probably want to use **structs**



```rust
impl Catalog {
    fn find_by_title(
        &self,
        title: &str
    ) -> Vec<&Media> {

    }
}
```



```
impl Catalog {
    fn find_by_title(
        &self,
        title: &str
    ) -> Vec<&Media> {

    }
}
```



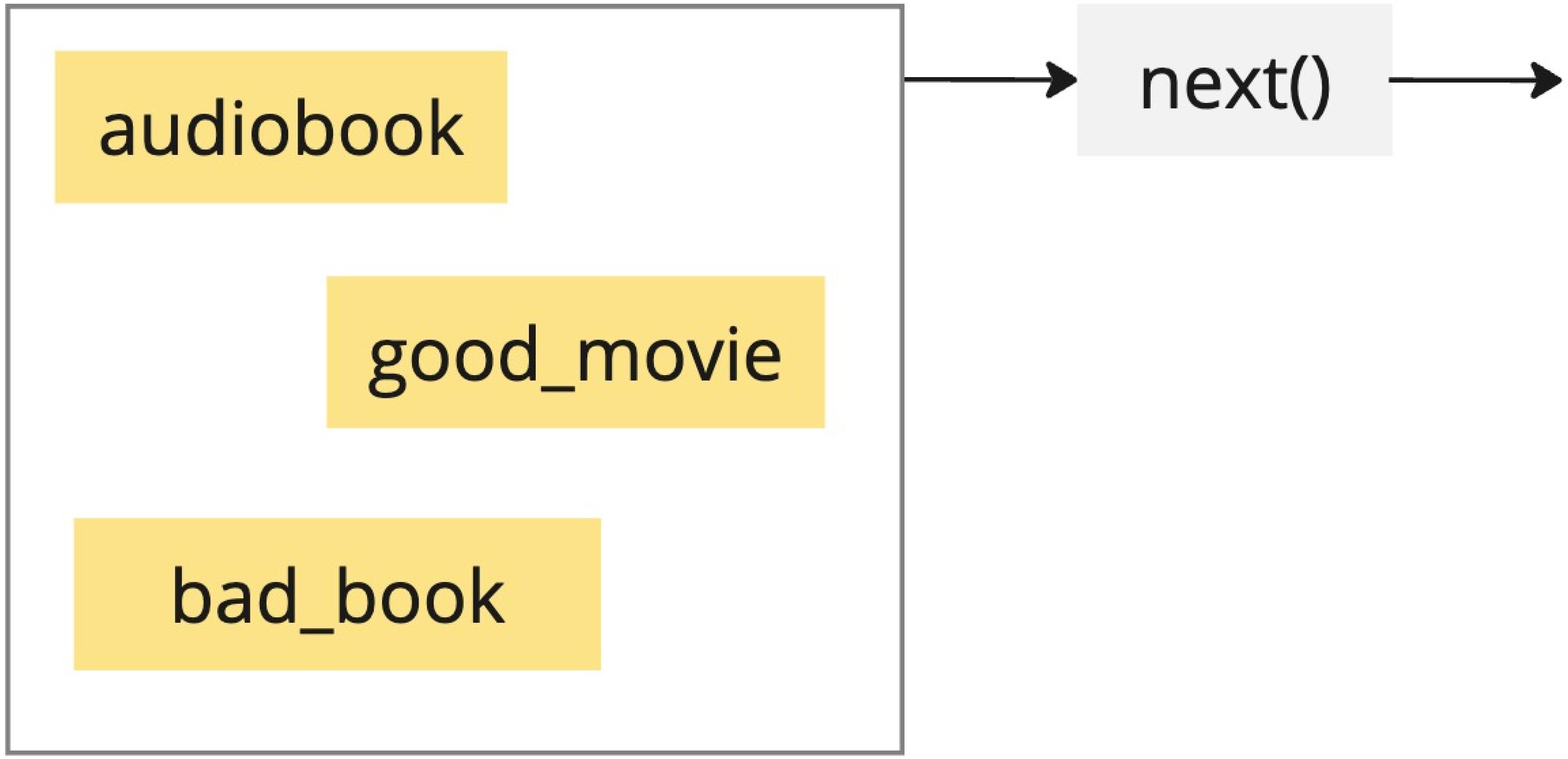
```rust
impl Catalog {
    fn find_by_title(
        &self,
        title: &str
    ) -> Vec<&Media> {

    }
}
```

```
self.items
    .iter()
    .filter(|m| m.title().contains(title))
    .collect::<Vec<&Media>>()
```

Gives us an ***iterator***

Iterators are the #1 tool we have for working with collections of data

Iterator



doc.rust-lang.org/std