

1	Every value is 'owned' by a single variable, argument, struct, vector, etc at a time	Ownership
2	Reassigning the value to a variable, passing it to a function, putting it into a vector, etc, <i>moves</i> the value. The old owner can't be used to access the value anymore!	
3	You can create many read-only references to a value that exist at the same time	Borrowing
4	You can't move a value while a ref to the value exists	
5	You can make a writeable (mutable) reference to a value <i>only if</i> there are no read-only references currently in use. One mutable ref to a value can exist at a time	
6	You can't mutate a value through the owner when any ref (mutable or immutable) to the value exists	
7	Some types of values are <i>copied</i> instead of moved (numbers, bools, chars, arrays/tuples with copyable elements)	Lifetimes
8	When a variable goes out of scope, the value owned by it is <i>dropped</i> (cleaned up in memory)	
9	Values can't be dropped if there are still active references to it	
10	References to a value can't outlive the value they refer to	
11	<b>These rules will dramatically change how you write code (compared to other languages)</b>	
12	<b>When in doubt, remember that Rust wants to minimize unexpected updates to data</b>	

Tons of corner cases and tiny things to remember  
around ownership + borrowing



**Above all, Rust wants to avoid  
'unexpected updates'**

## ***mustang***

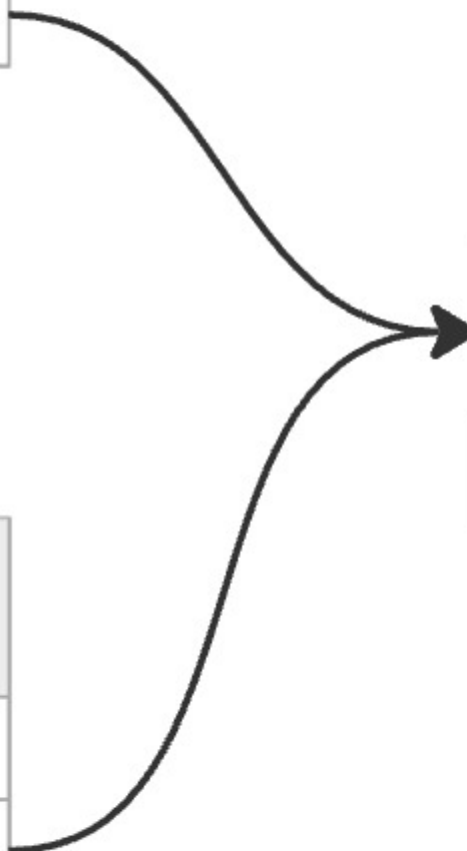
<i>Field</i>	<i>Value</i>
name	"Mustang"
engine	*

## ***camaro***

<i>Field</i>	<i>Value</i>
name	"Camaro"
engine	*

## ***engine***

<i>Field</i>	<i>Value</i>
working	false



1

Every value is 'owned' by a **single variable** at a time

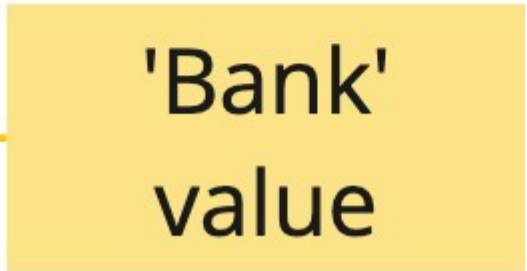
2

Reassigning the value to another variable ***moves*** the value. The old variable can't be used to access the value anymore!

*Slightly simplified text*


```
fn main() {  
    let bank = Bank::new();  
  
    let other_bank = bank;  
  
    println!("{:#?}", bank);  
}
```

'Bank'  
value



What are all the things  
that can store the Bank  
value?

```
fn main() {  
  let bank = Bank::new();  
  
  let other_bank = bank;  
  
  println!("{:#?}", bank);  
}
```

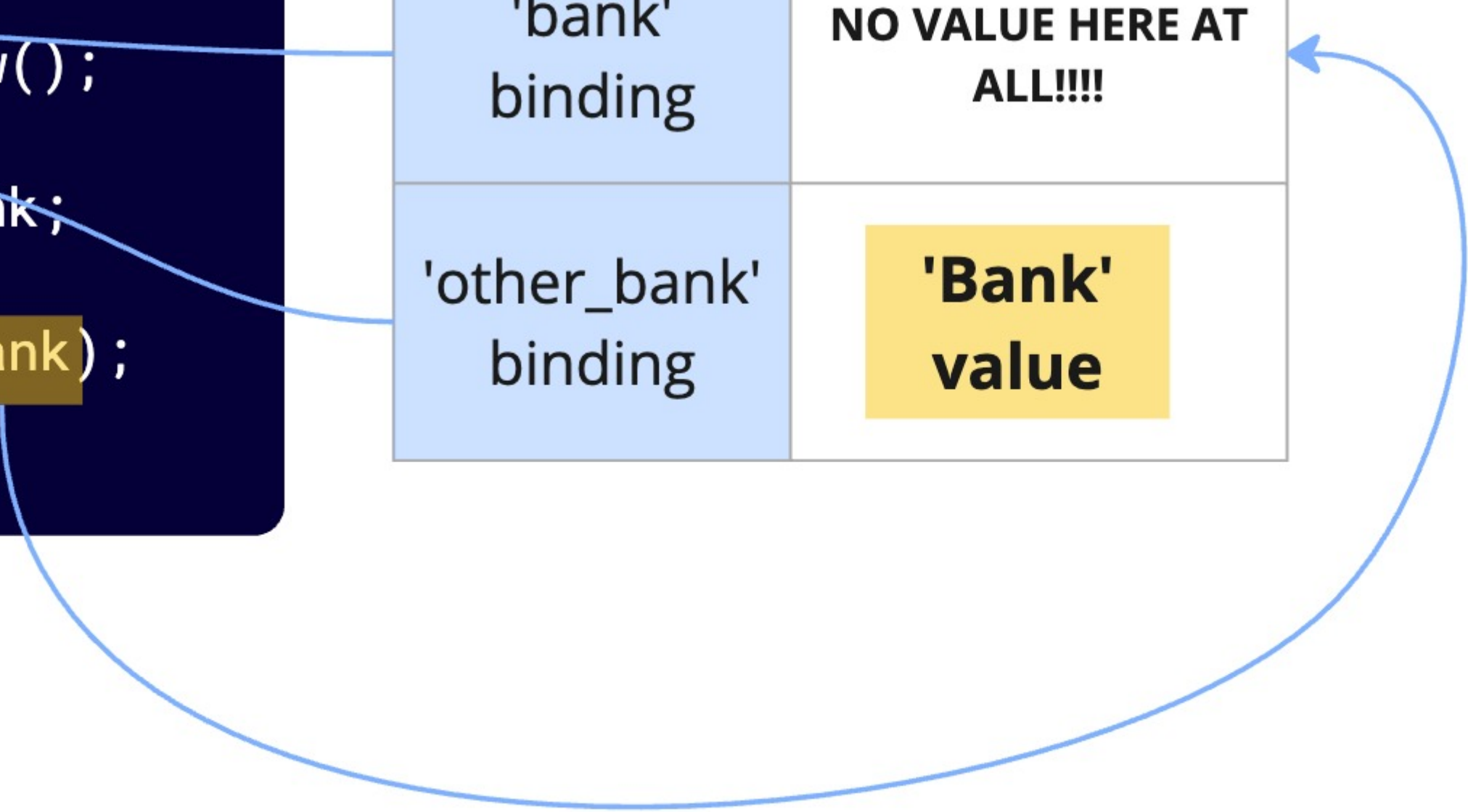


'bank' binding	
'other_bank' binding	



```
fn main() {  
  let bank = Bank::new();  
  let other_bank = bank;  
  println!("{:#?}", bank);  
}
```

'bank' binding	NO VALUE HERE AT ALL!!!!
'other_bank' binding	<b>'Bank' value</b>



1

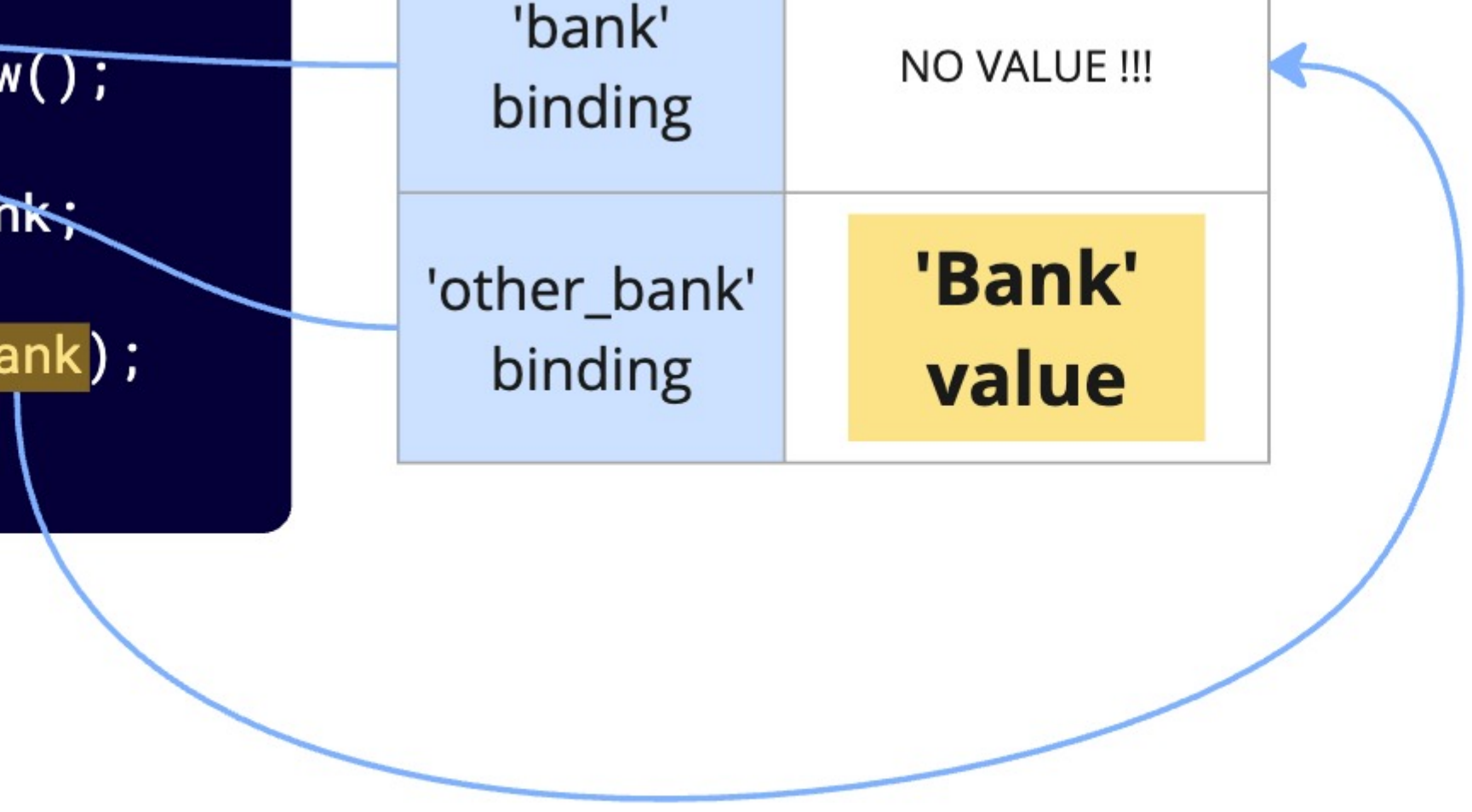
Every value is '**owned**' by a **single variable** at a time

2

Reassigning the value to another variable **moves** the value. The old variable can't be used to access the value anymore!

```
fn main() {  
    let bank = Bank::new();  
    let other_bank = bank;  
    println!("{}", bank);  
}
```

'bank' binding	NO VALUE !!!
'other_bank' binding	<b>'Bank' value</b>





1

Every value is '**owned**' by a **single variable** at a time

2

Reassigning the value to another variable **moves** the value. The old variable can't be used to access the value anymore!



1

Every value is '**owned**' by a **single** variable, argument, struct, vector, etc at a time

2

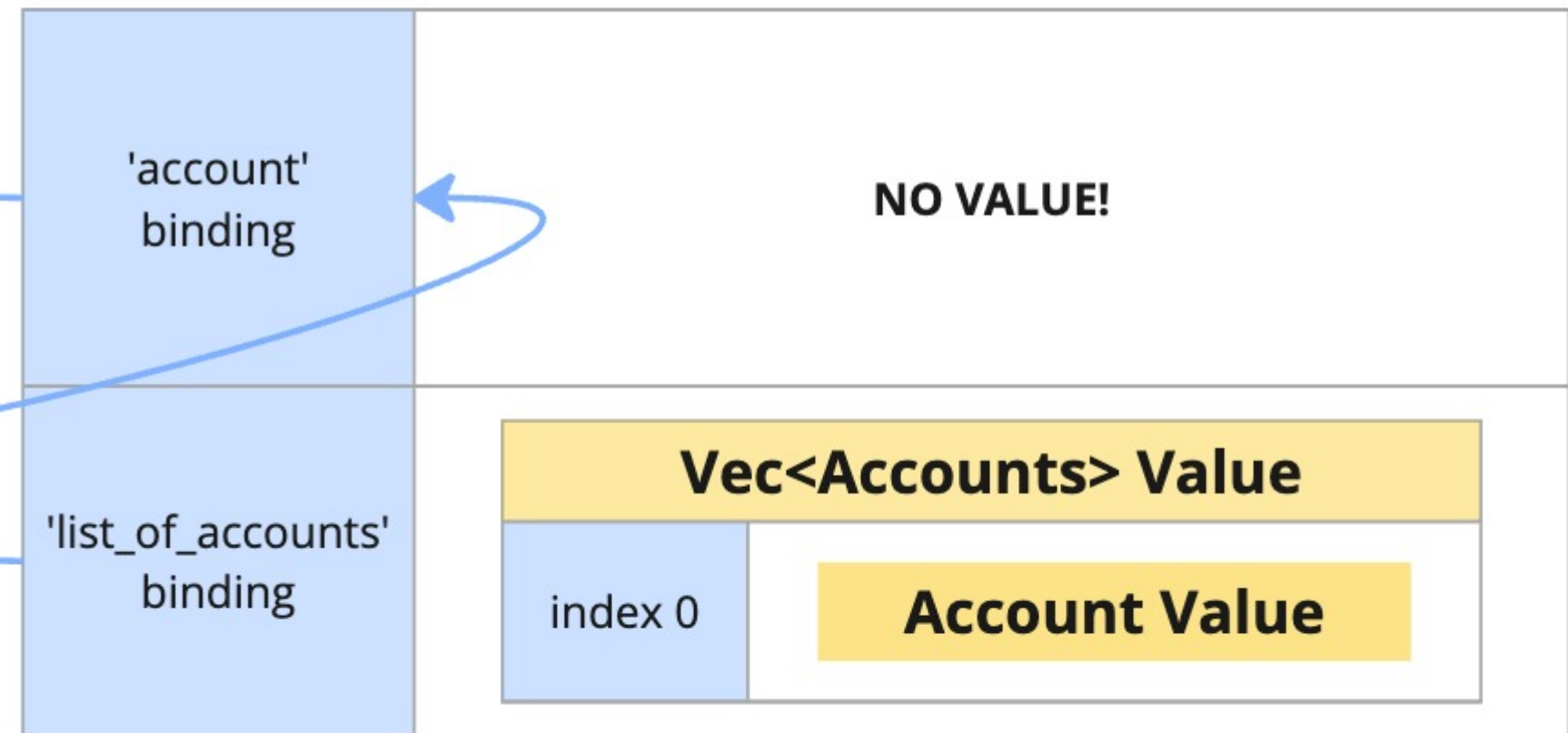
Reassigning the value to a variable, passing it to a function, putting it into a vector, etc, **moves** the value. The old owner can't be used to access the value anymore!

```
fn print_account(account: Account) {  
    println!("{:#?}", account);  
}  
  
fn main() {  
    let account = Account::new(  
        1,  
        String::from("me")  
    );  
    print_account(account);  
    print_account(account);  
}
```

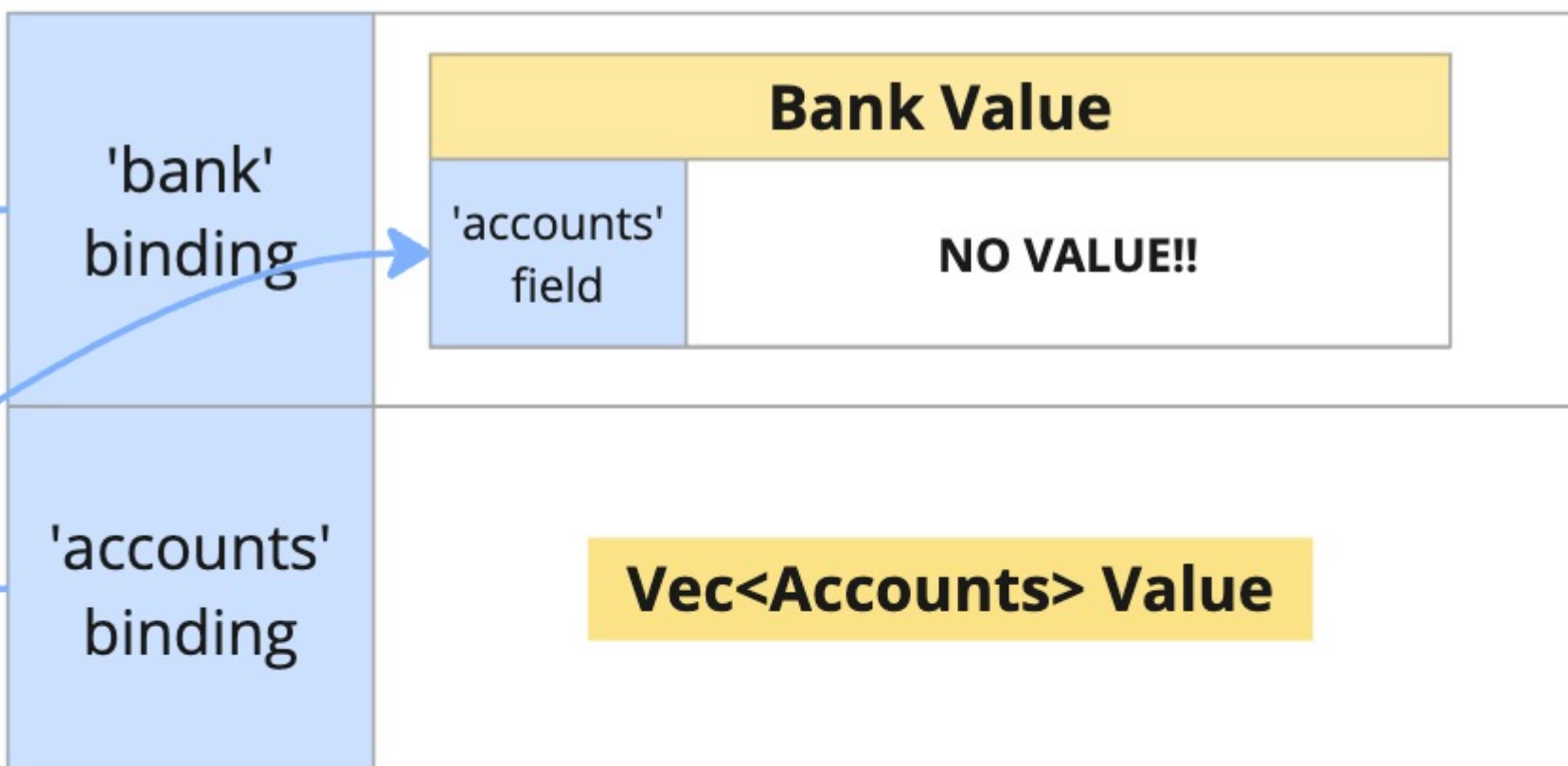
'account' binding	NO VALUE!
'print_account' function #1	<b>'Account' value</b>
'print_account' function #2	



```
fn main() {  
    let account = Account::new(  
        1,  
        String::from("me")  
    );  
  
    let list_of_accounts = vec![account];  
  
    println!("{}", account);  
}
```



```
fn main() {  
    let bank = Bank::new();  
    let accounts = bank.accounts;  
    println!("{}", bank.accounts);  
}
```





```
fn print_account(account: Account) {  
    println!("{:#?}", account);  
}  
  
fn main() {  
    let account = Account::new(  
        1,  
        String::from("me")  
    );  
    print_account(account);  
    println!("{:#?}", account.holder);  
}
```

'account' binding	NO VALUE!	
'print_account' function	Account Value	
	'holder' Field	'Holder' String Value



```
fn print_account(account: Account) {  
    println!("{:#?}", account);  
}  
  
fn print_holder(holder: String) {  
    println!("{}", holder);  
}  
  
fn main() {  
    let account = Account::new(  
        1,  
        String::from("me")  
    );  
    print_holder(account.holder);  
    print_account(account);  
}
```

'account' binding	Account Value	
	'holder' Field	NO VALUE!!!
'print_holder' function	'Holder' String Value	
'print_account' function		

Given the ownership system, how do we write useful code?

**Option #1**

Manually move values  
back and forth between  
different owners

**Option #2**

Use the borrowing  
system

```
fn print_account(account: Account) -> Account {  
    println!("{:#?}", account);  
    account  
}  
  
fn main() {  
    let mut account = Account::new(  
        1,  
        String::from("me")  
    );  
    account = print_account(account);  
    account = print_account(account);  
    println!("askjdfj", account)  
}
```

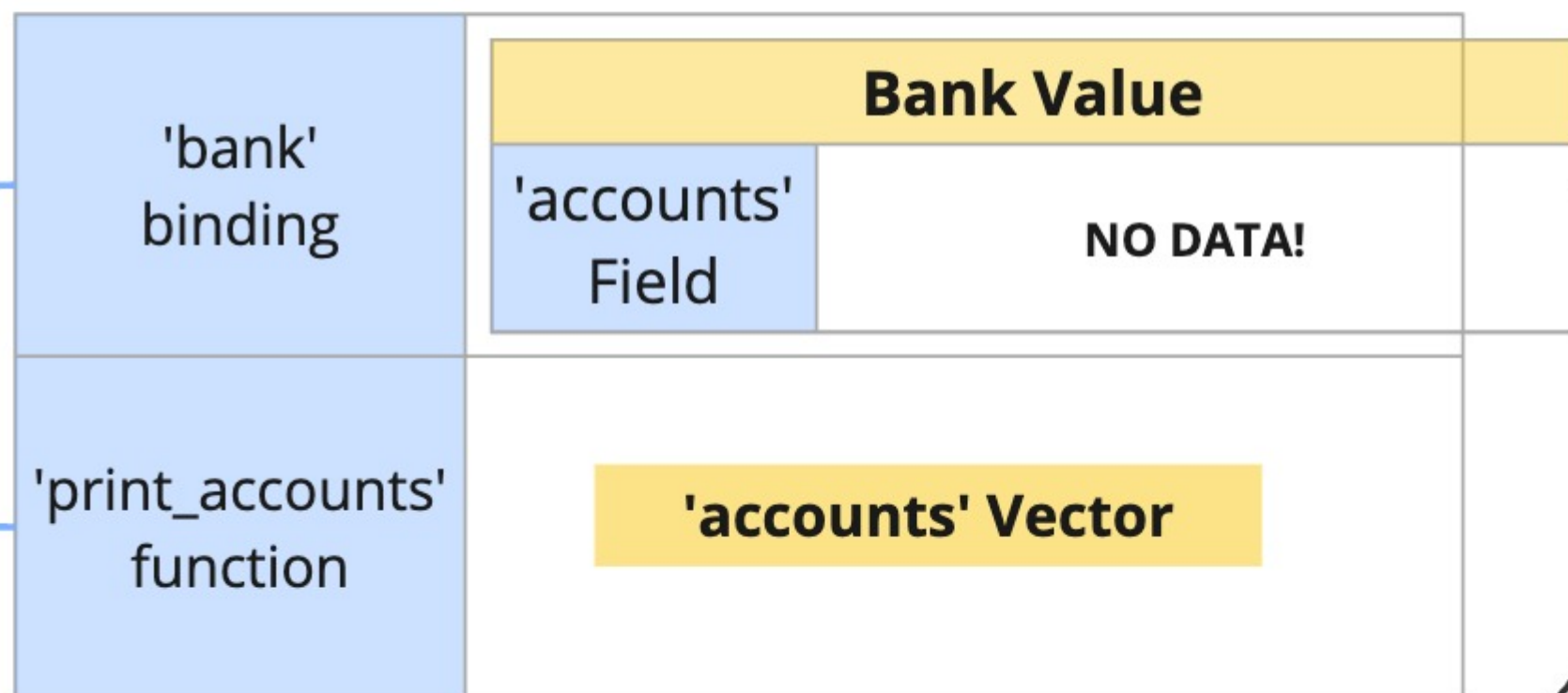
'account' binding	<b>'Account' value</b>
'print_account' function #1	
'print_account' function #2	







```
fn print_bank(bank: Bank) {  
    println!("{:#?}", bank);  
}  
  
fn print_accounts(accounts: Vec<Account>) {  
    println!("{:#?}", accounts);  
}  
  
fn main() {  
    let bank = Bank::new();  
    print_accounts(bank.accounts);  
    print_bank(bank);  
}
```





```
fn add_account(bank: &mut Bank, account: Account) {  
    bank.accounts.push(account);  
}  
  
fn main() {  
    let mut bank = Bank::new();  
    let account = Account::new(1, String::from("me"));  
    add_account(&mut bank, account);  
    println!("{:#?}", bank);  
}
```

