Pinning

By Jonathan Louie

Topics

Crash Course on Pointers

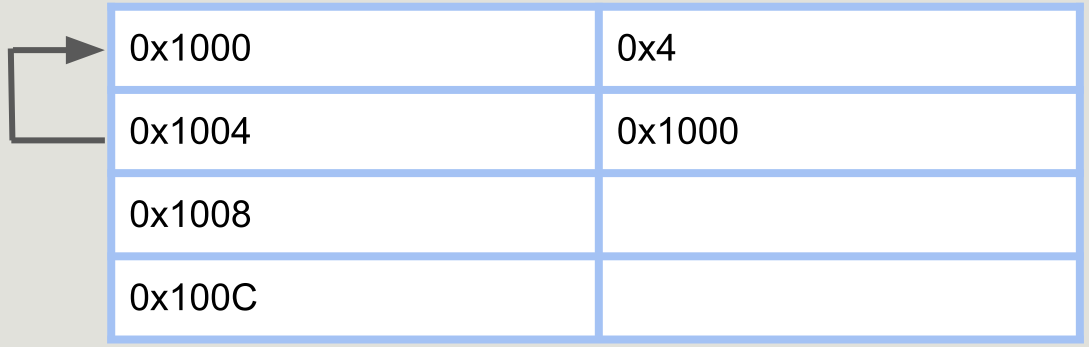
What is Pinning?

How Pin and Unpin work

Why Pinning?

Crash Course on Pointers

Memory



Address	Value
0x1000	0x4
0x1004	0x1000
0x1008	
0x100C	

```
let i = 4;
let ptr = &i;
```

Pointer Types in Rust

Raw Pointers	*const T
	*mut T
References	&T
	&mut T
Smart Pointers	Box<T>
	Rc<T>
	…and many more!

Raw Pointer Syntax

```
let ptr: *mut i32 = &mut x;

unsafe { let y: i32 = *ptr; }

unsafe { *ptr = 3; }
```

Unsafe Rust

The `unsafe` keyword allows you to do 5 things:

- **Dereference a raw pointer**
- **Call an unsafe function or method**
- **Access or modify a mutable static variable**
- **Implement an unsafe trait**
- **Access fields of unions**

Unsafe Rust

The `unsafe` keyword allows you to do 5 things:

- **Dereference a raw pointer**
- **Call an unsafe function or method**
- **Access or modify a mutable static variable**
- **Implement an unsafe trait**
- **Access fields of unions**

What is Pinning?

Pinning

- **Sometimes, we want the placement of an object in memory to be unable to change**
    - **In other words, we want to “pin” data to its location in memory**
- **This is useful when working with self-referential data**

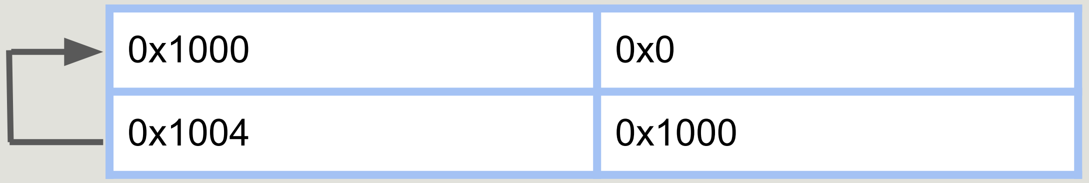
Example

Imagine we have defined the following struct Foo:

```
struct Foo {
        a: usize,
        a_ptr: *const usize
}
```

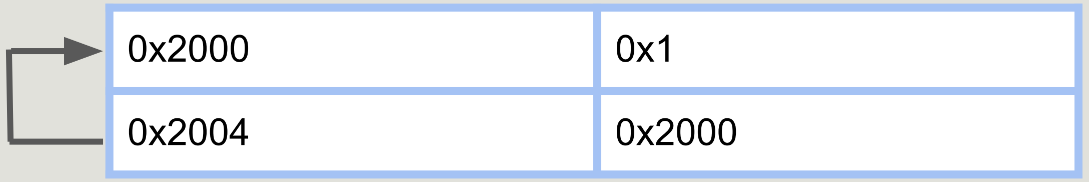
Example

Foo 1



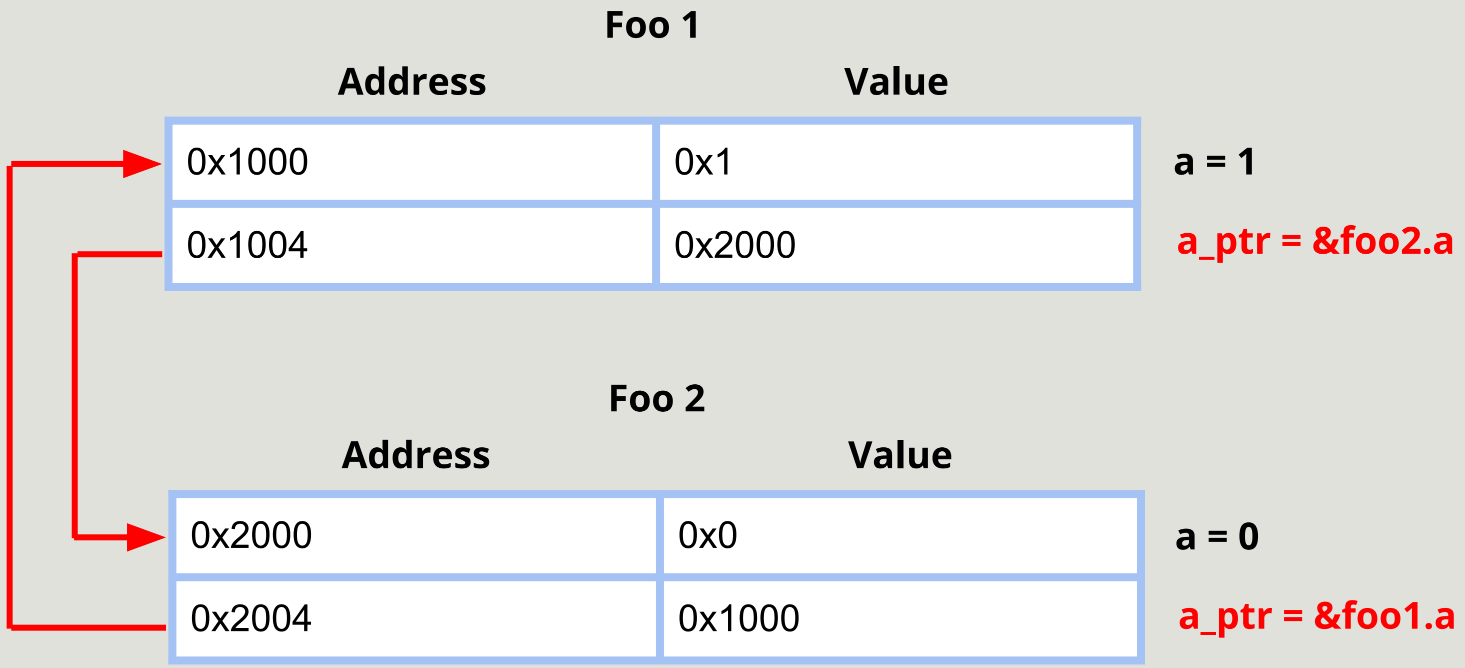
Address	Value	
0x1000	0x0	**a = 0**
0x1004	0x1000	**a_ptr = &foo1.a**

Foo 2



Address	Value	
0x2000	0x1	**a = 1**
0x2004	0x2000	**a_ptr = &foo2.a**

Example (After mem::swap)



Foo 1

Address	Value	
0x1000	0x1	**a = 1**
0x1004	0x2000	**a_ptr = &foo2.a**

Foo 2

Address	Value	
0x2000	0x0	**a = 0**
0x2004	0x1000	**a_ptr = &foo1.a**

Example Code

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=cd09081a8994e55ddfe65041fa755a1a

Pin and Unpin

Pin<P>

- **Wrapper around a kind of pointer that makes the pointer “pin” its value in place**
- **Prevents value pointed to by a pointer of type P from being moved around in memory**

Unpin Trait

```
pub auto trait Unpin { }
```

Auto Traits

- Automatically implemented for all types
- Must explicitly opt out using a negative impl
  - Example: impl !Unpin for Type {}
- Documentation: https://doc.rust-lang.org/beta/unstable-book/language-features/auto-traits.html

Unpin Trait

- Implementing Unpin for a type removes the restriction of pinning on that type
- When implemented for a type T, it allows you to move a value of type T out of a pinned pointer to T (Pin<Ptr<T>>)
  - Example: Pin<&mut i32>
- For non-pinned data, it has no effect

PhantomPinned

- Marker type that implements !Unpin
- Any type containing PhantomPinned will not implement Unpin
- Documentation: https://doc.rust-lang.org/std/marker/struct.PhantomPinned.html

Example (Pin and Unpin interactions)

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=efc2ca9ab778ea0f520667b07594616c

Why Pinning?

Where Pinning is used

- **Self-referential data structures**
  - **E.g. Intrusive-doubly linked list** https://doc.rust-lang.org/std/pin/#example-intrusive-doubly-linked-list
- **Async code**
  - **Async/await syntax often generates state machines that use self-referential types under the hood**
  - **Example:** https://rust-lang.github.io/async-book/04_pinning/01_chapter.html

Example (Using Pin with Futures)

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=3279543c412187ce49770465259d3b01

Further Reading

- Pin module documentation: https://doc.rust-lang.org/std/pin/
- Pinning in Async code: https://rust-lang.github.io/async-book/04_pinning/01_chapter.html
    - Great summary of pinning rules at bottom of page
- Great example of Pin in a practical setting: https://blog.cloudflare.com/pin-and-unpin-in-rust/