DSTs Florob Last Time The Problem The Solution: DSTs Last Time 2 Last Time 2	
Florob Last Time The Problem The Solution: DSTs Last Time 2 Condition:	DSTs
Last Time The Problem DSTs DSTs Last Time 2 Last Time 2 Elorion "Eloroh" Zeitz	Florob
The Problem DSTs The Solution: DSTs Last Time 2 Elorion "Elorob" Zeitz	ast Time
The Solution: DSTs Last Time 2 Continue 2	The Problem
Last Time 2 Generation Selection Sel	The Solution:
Elorion "Floroh" 7eitz	
	Cuestions
	adonona

2019-04-03

Florob

Last Time The Problem The Solution: DSTs Last Time 2 Questions

1 Last Time

2 The Problem

3 The Solution: DSTs

4 Last Time 2

Florob

Last Time

The Problem The Solution: DSTs Last Time 2 Questions

1 Last Time

2 The Problem

3 The Solution: DSTs

4 Last Time 2

< □ > < □ > < □ > < ⊇ > < ⊇ > < ⊇ > < ⊇ > < ⊇ > < ⊇ < つへ(~ 3/17

Last time's discussion

DSTs

Florob

Last Time

The Problem The Solution: DSTs Last Time 2 Questions

- Can you resize the object behind a Box<[u8]>
- Different from assigning a different Box<[u8]> to the same binding
- Maybe std::mem::replace(&mut *the_box, [1, 2, 3, 4][..])?

Florob

Last Time

The Problem The Solution: DSTs Last Time 2 Questions

Last Time

2 The Problem

3 The Solution: DSTs

4 Last Time 2

< □ > < □ > < □ > < ⊇ > < ⊇ > < ⊇ > < ⊇ > < ⊇ > 5/17

When Size Doesn't Matter

DSTs

- Last Time
- The Problem The Solution: DSTs
- Questions

- Most types have a size known at compile time
- We can store them on the stack, pass them to functiions, etc.
- Sometimes we want to pass something we don't know the size of
 - array of arbitrary size
 - part of an array/vector
 - implementer of a Trait (without generics)



Florob

```
Last Time
The Problem
The Solution:
DSTs
Last Time 2
Questions
```

```
struct Slice<'a, T> {
    ptr: *const T,
    len: usize,
    __phantom: PhantomData<&'a T>,
    }
```

Can point to an array of any length, or slice thereofDone, right?



```
Last Time
The Problem
The Solution:
DSTs
Last Time 2
Questions
```

```
struct Slice<'a, T> {
    ptr: *const T,
    len: usize,
    __phantom: PhantomData<&'a T>,
    }
```

- Can point to an array of any length, or slice thereof
- Done, right?
- What about mutably borrowing from this type?



```
Last Time
The Problem
The Solution:
DSTs
Last Time 2
Questions
```

```
struct Slice<'a, T> {
    ptr: *const T,
    len: usize,
    __phantom: PhantomData<&'a T>,
    }
```

- Can point to an array of any length, or slice thereof
- Done, right?
- What about mutably borrowing from this type?
- Well, what if we want to pass ownership?

More Structs

DSTs

```
Florob
```

```
Last Time
The Problem
The Solution:
```

```
Last Time 2
Questions
```

```
struct SliceMut<'a, T> {
1
2
       ptr: *mut T,
       len: usize,
3
       phantom: PhantomData<&'a T>,
4
5
   }
6
7
   struct BoxStruct < T > { ... }
8
   struct RcStruct<T> { ... }
0
10
   struct ArcStruct<T> { ... }
11
   This is not composable
```

```
    And therefore doesn't scale well
```

Florob

Last Time The Problem The Solution: DSTs Last Time 2 Questions

Last Time

2 The Problem

3 The Solution: DSTs

4 Last Time 2

<ロト < 部 > < 注 > < 注 > 注) へ () 9/17



Florob

Last Time The Problem The Solution: DSTs Last Time 2

Rust provides two build in types to solve this problem

[T]

- dyn Trait (formerly Trait)
- Size of both is unknown at compile time
- They can't stand on their own as a variable or argument type

DSTs as Existential Types



- [T] is an existential array type: \exists n. [T; n]
- dyn Trait is an existential value type: ∃ T. T: Trait

Unsizing

Florob

Last Time The Problem The Solution: DSTs Last Time 2 Questions

- As one of its few coercions Rust does unsizing:
 - [T; n] **to** [T]
 - T implementing Trait to dyn Trait
- This allows:
 - $\blacksquare \& [T; 4] \rightarrow \& [T]$
 - Box<[T; 4]> → Box<[T]>
 - $\blacksquare \operatorname{Rc} < [T; 4] > \rightarrow \operatorname{Rc} < [T] >$
- These pointers to DSTs have an additional word:
 - For [T] the length
 - For dyn Trait a pointer to the vtable

DSTs in Structs

DSTs

Florob

Last Time The Problem The Solution: DSTs Last Time 2 Questions

- DSTs can occur in structs
- This makes the struct a DST too
- Only as the last field
- You can only create them via unsizing and via generics

DSTs and Traits and Generics

DSTs

- Last Time The Problem The Solution: DSTs Last Time 2
- All regular types automatically implement the sized trait
- All generics are per default bound by the Sized trait
- You need to opt out of this default by requiring (allowing) ?Sized

```
1 struct Foo<T: ?Sized> {
2   foo: u16,
3   bar: T
4  }
5
6 let x: &Foo<[u8]> = &Foo { foo: 12, bar: [0; 4] };
```

Florob

Last Time The Problem The Solution: DSTs Last Time 2 Questions

Last Time

2 The Problem

3 The Solution: DST

4 Last Time 2

Last time's discussion

DSTs

Florob

- Last Time The Problem The Solution: DSTs Last Time 2 Questions
- Can you resize the object behind a Box<[u8]>
- Different from assigning a different Box<[u8]> to the same binding
- Maybe std::mem::replace(&mut *the_box, [1, 2, 3, 4][..])?

pub fn replace<T>(dest: &mut T, src: T) -> T

■ There is an implicit Sized bound

Florob

Last Time The Problen The Solution DSTs Last Time 2

Questions

Thank you for your attention. Any questions?



https://babelmonkeys.de/~florob/talks/RC-2019-04-03-dsts.pdf

<ロト < 部 > < 注 > < 注 > 注 の < で 17 / 17