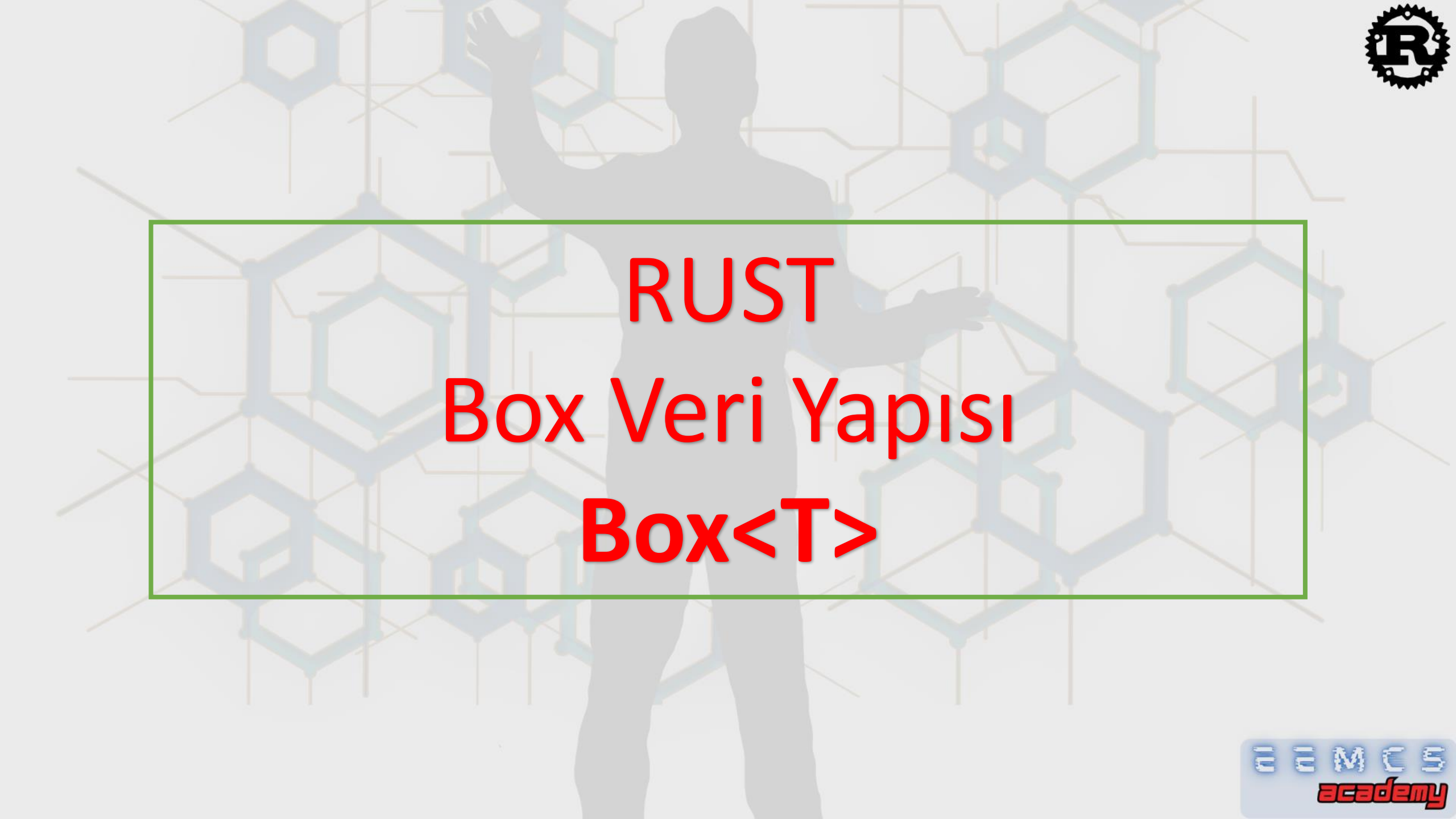

RUST
Box Veri Yapısı
Box<T>

Box<T> Nedir?

Smart Pointer olarak geçer yani heap memory de tutulan verinin adresini saklarlar.

Veriyi stack memory yerine heap memory de saklamak için kullanılır.

Veriyi stack yerine heap memoryde tutmak dışında herhangi bir performans kaybı oluşturmaz.

Box<T> Tanımlaması

```
let değişken_adı = Box::new(değer);
```

```
fn main() {
    let b = Box::new(5);
    println!("b = {}", b);
}
```

Box<T> ve *-Dereferans Operatörü

Stack verisi referans alındığında kullanılan * operatörü, Box<T> ile de kullanılabilir.

```
fn main() {
    let x = 5;
    let y = Box::new(x);

    if 5 == x {
        println!("Eşitlik var")
    }

    if 5 == *y {
        println!("Eşitlik var")
    }
    let z = x * *y;
    println!("{z}");
}
```

```
fn main() {
    let x = 5;
    let y = &x;

    if 5 == x {
        println!("Eşitlik var")
    }

    if 5 == *y {
        println!("Eşitlik var")
    }
    let z = x * *y;
    println!("{z}");
}
```

Box<T> Neden Kullanılır ?

1 - Derleme (compile time) zamanında boyutu bilinemeyen bir veri yapınız olduğunda, bu veri yapısını boyutu belli olan bir yapıya dönüştürmek için kullanırız.

Recursive (kendi kendini tekrar eden) yapılarda kullanılmaktadır.

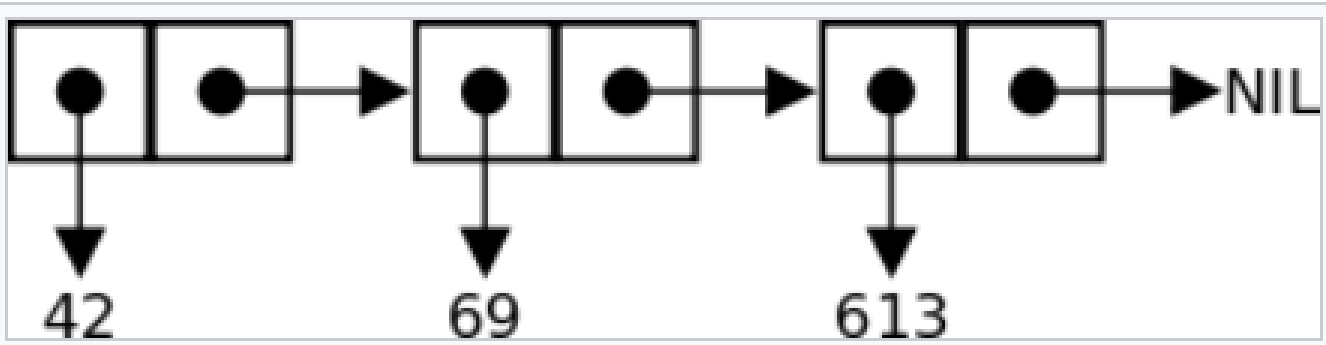
```
enum List {

        Cons(i32, List),
        Nil,
}
```

HATA OLUŞUR
infinite size

https://en.wikipedia.org/wiki/Cons

```
enum List {
        Cons(i32, Box<List>),
        Nil,
}
```

Lists [edit]



Cons cell diagram for the list (42 69 613), written with `cons`:

```
(cons 42 (cons 69 (cons 613 nil)))
```

and written with `list`:

```
(list 42 69 613)
```

```
use List::{Cons, Nil};

fn main() {

      let list = Cons(1, Box::new(Cons(2, Box::new(Cons(3,
Box::new(Nil))))));

 }
```

2 - Büyük miktarda veriye sahip olduğunuzda ve sahipliği aktarmak istediğinizde ancak bunu yaptığınızda verilerin kopyalanmayacağından emin olmak istediğinizde

Stack memorydeki büyük miktarlardaki verilerin sahipliğini transfer uzun sürer. Çünkü veri kopyalanır.

Böyle durumlarda performansı arttırmak için verilerin Box ile heap memoryde tutulması gerekir. Böylece sadece pointer verisi kopyalanarak işlem yapılmış olur.

```
fn main() {
    let buyuk_veri = Box::new([0; 2_000_000]);
    println!("{}", big_data[0]);
}
```

3 - Kullanmak istediğiniz bir veri için ; verinin türünün önemli olmadığı ama bu veri tipinin bir trait'i implenment etmesinin daha önemli olduğu durumlarda Box kullanabilirsiniz.

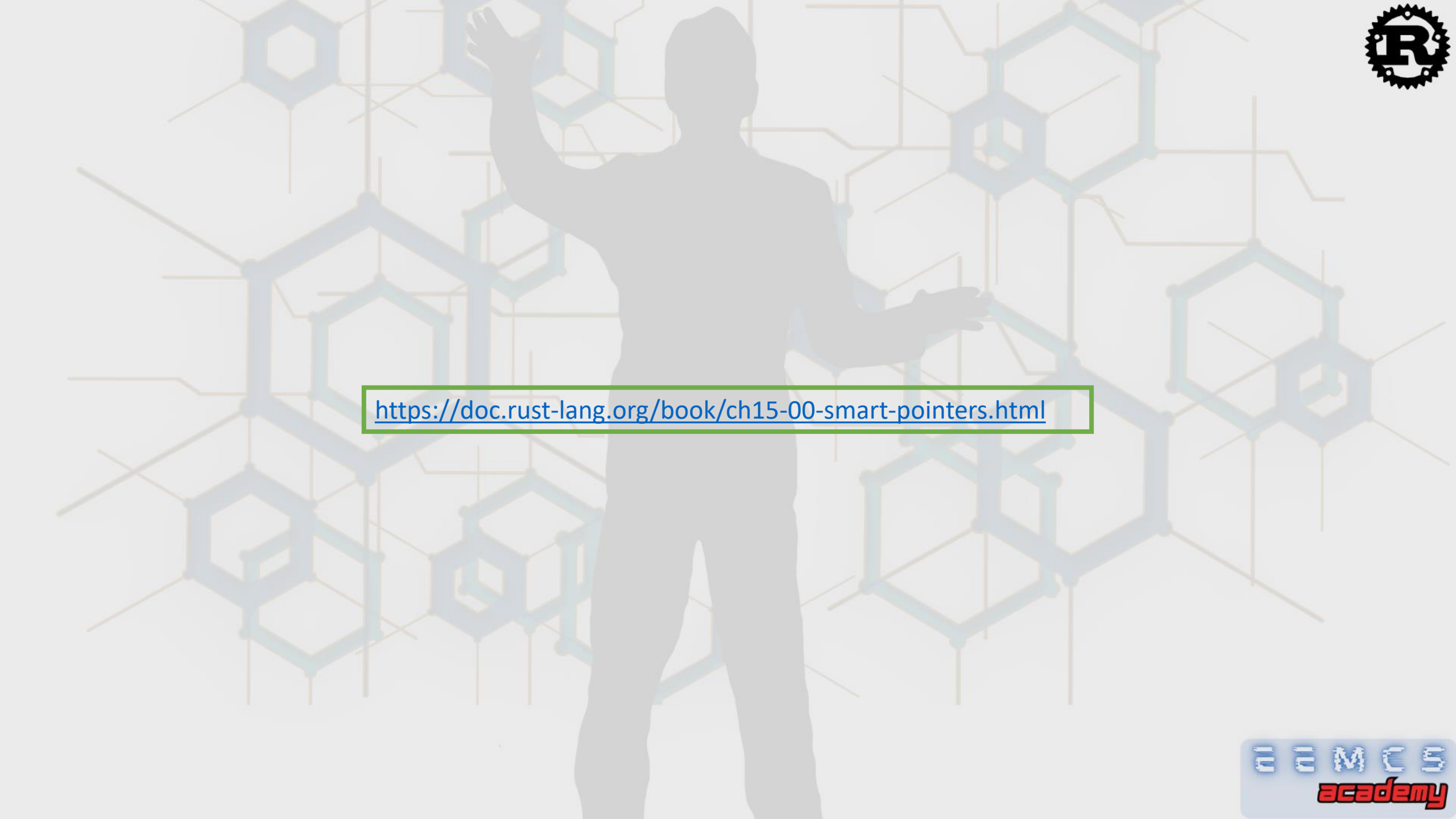
Kısaca veriler için aynı Trait implementasyonu sağlamak için kullanılır.

(TRAIT KONUSUNDAN SONRA AÇIKLANACAK)

Kendi Smart Pointer Tipini Oluşturur :

Rust içinde kendi smart pointer tipini oluşturabilirsiniz. Konuların fazla karışmaması için sonraki konulara bırakmamız gerekiyor.

Çünkü Trait, Generic Types konularını bilerek bu konuyu öğrenmek daha kolay olur.


https://doc.rust-lang.org/book/ch15-00-smart-pointers.html

Celal AKSU
Bilişim Teknolojileri Öğretmeni

celalaksu@gmail.com

https://www.linkedin.com/in/cllaksu/

https://twitter.com/ksacll

https://www.youtube.com/@eemcs