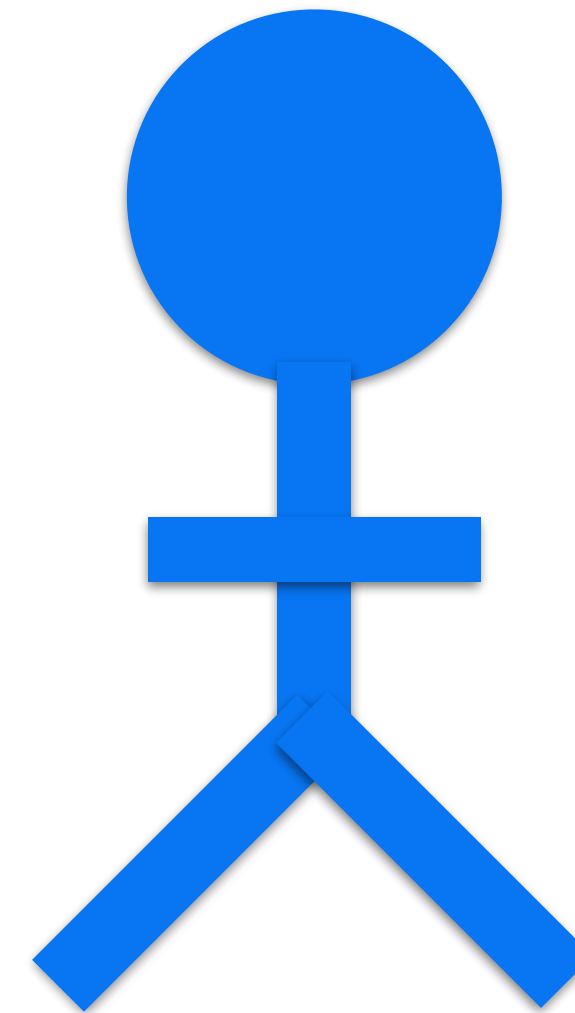
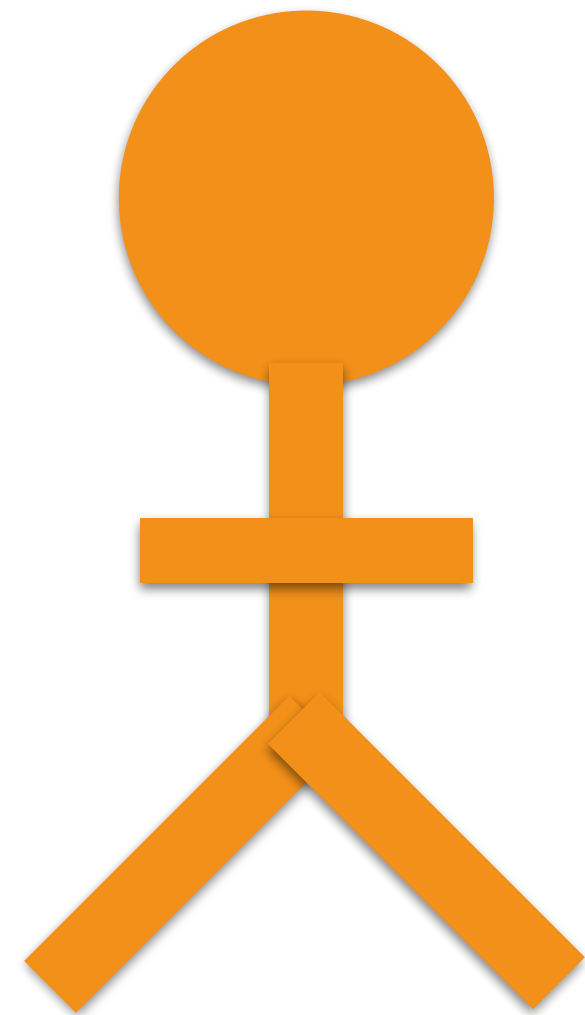




Mutable Borrowing



Borrowing: Mutable Borrows



Borrowing: Mutable Borrows



Borrowing: Mutable Borrows



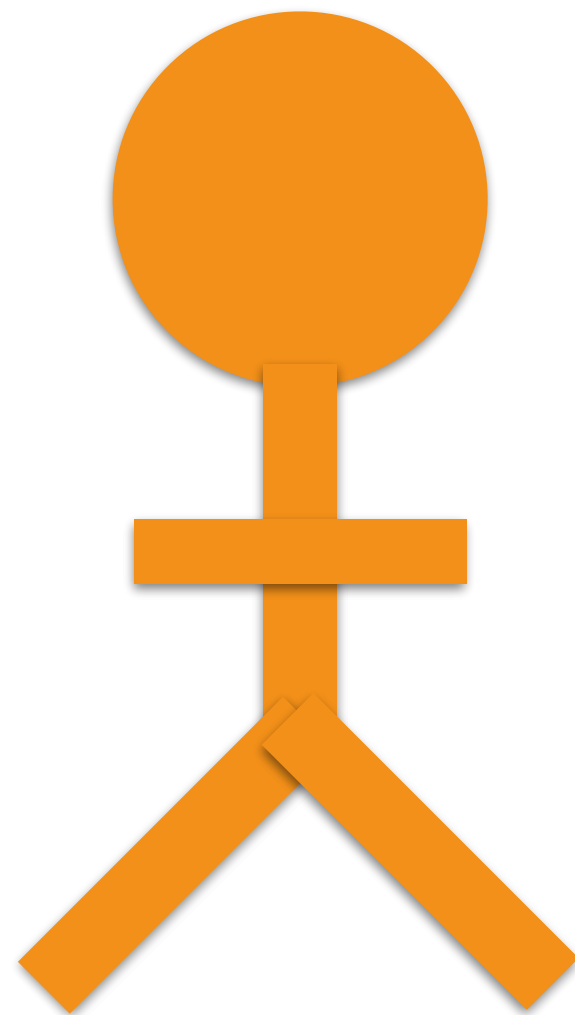
Borrowing: Mutable Borrows



Borrowing: Mutable Borrows

```
fn main() {  
    ➔ let mut name = ...;  
    update(&mut name);  
    println!("{}", name);  
}
```

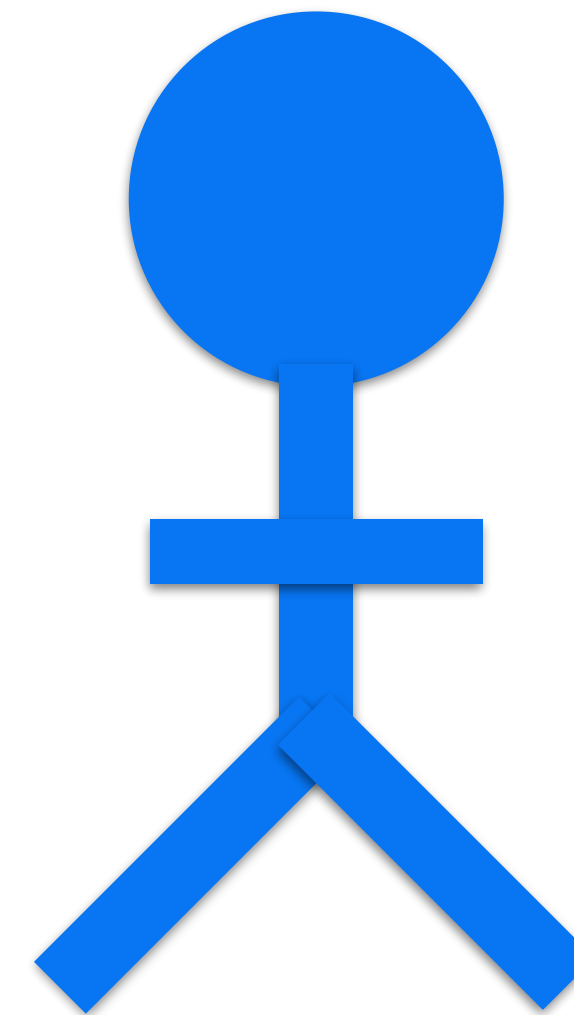
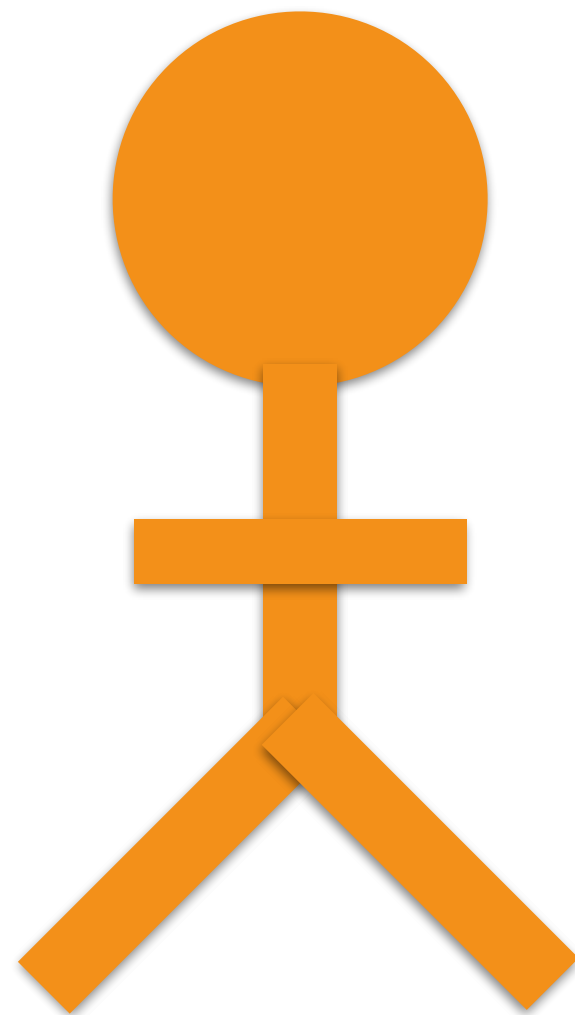
```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```



Mutable borrow

```
fn main() {  
    let mut name = ...;  
    → update(&mut name);  
    println!("{}", name);  
}
```

```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```

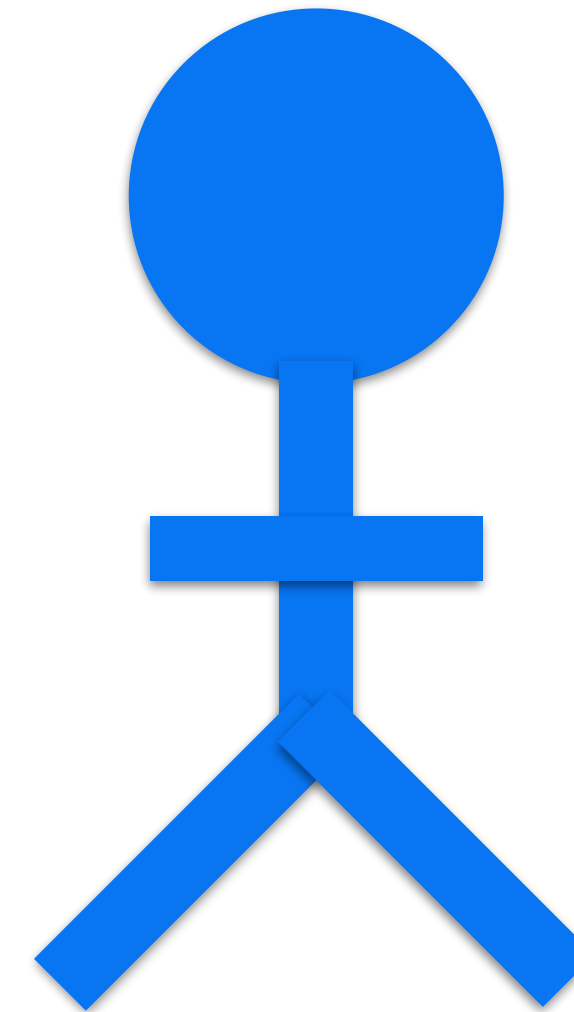
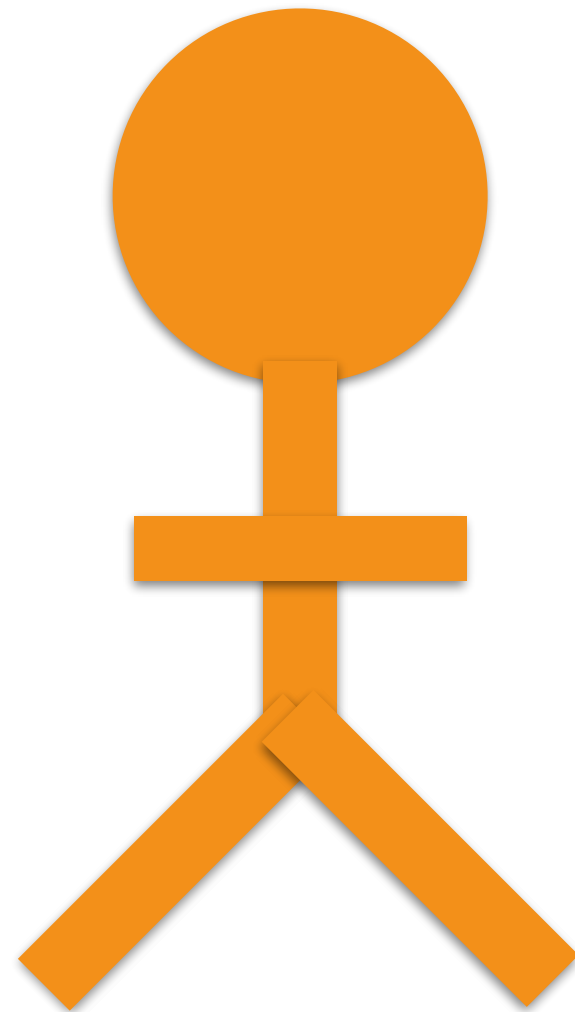


Mutable borrow


```
fn main() {  
    let mut name = ...;  
    → update(&mut name);  
    println!("{}", name);  
}
```

```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```

Take a **mutable
reference** to a String



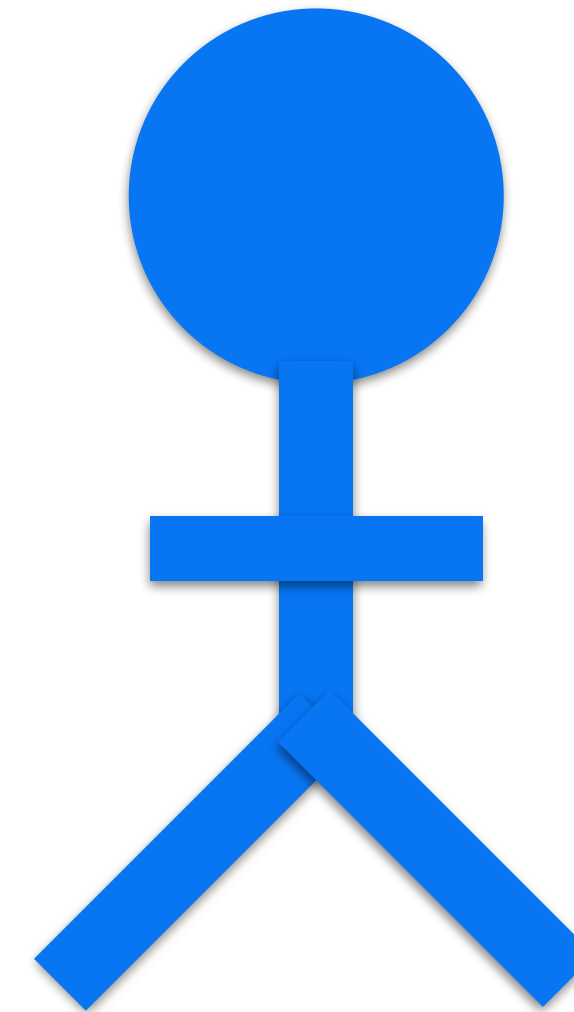
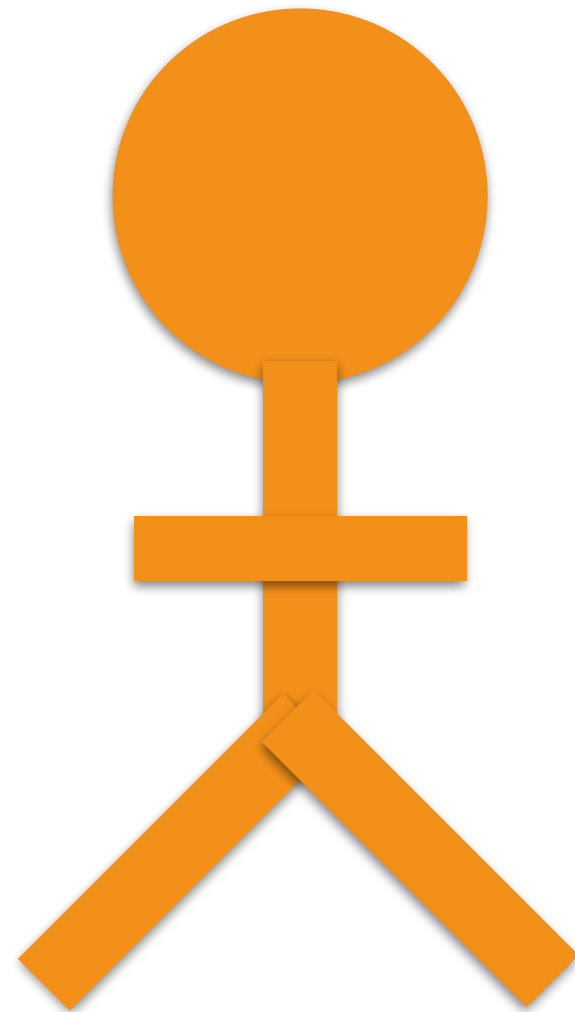
Mutable borrow

```
fn main() {
  let mut name = ...;
  → update(&mut name);
  println!("{}", name);
}
```

Lend the string
mutably

```
fn update(name: &mut String) {
  name.push_str("...");
}
```

Take a **mutable
reference** to a String



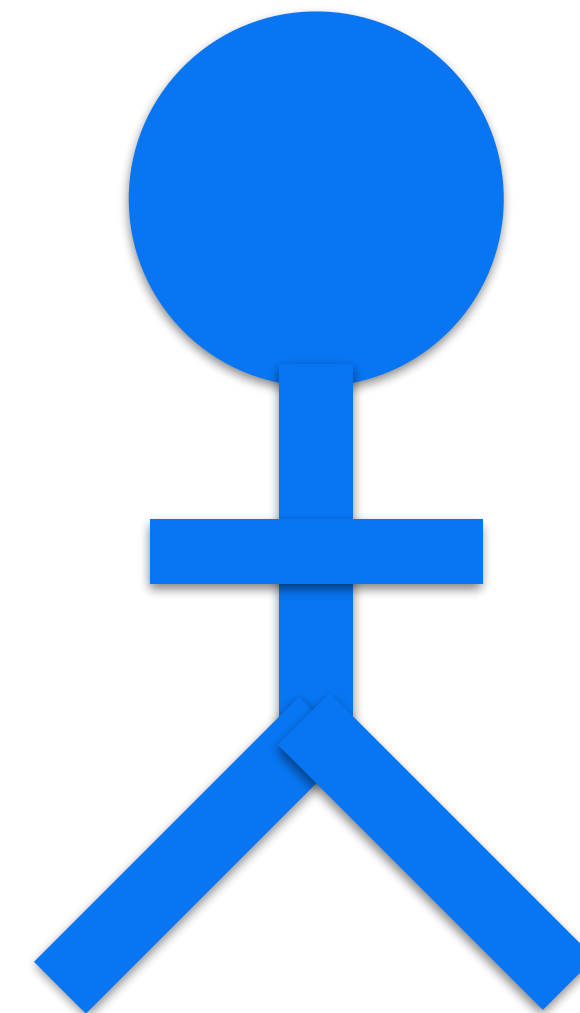
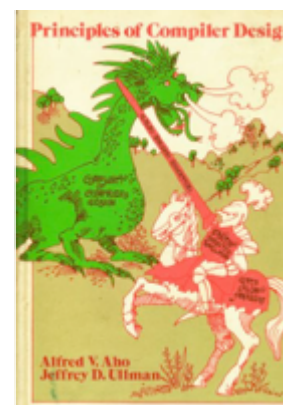
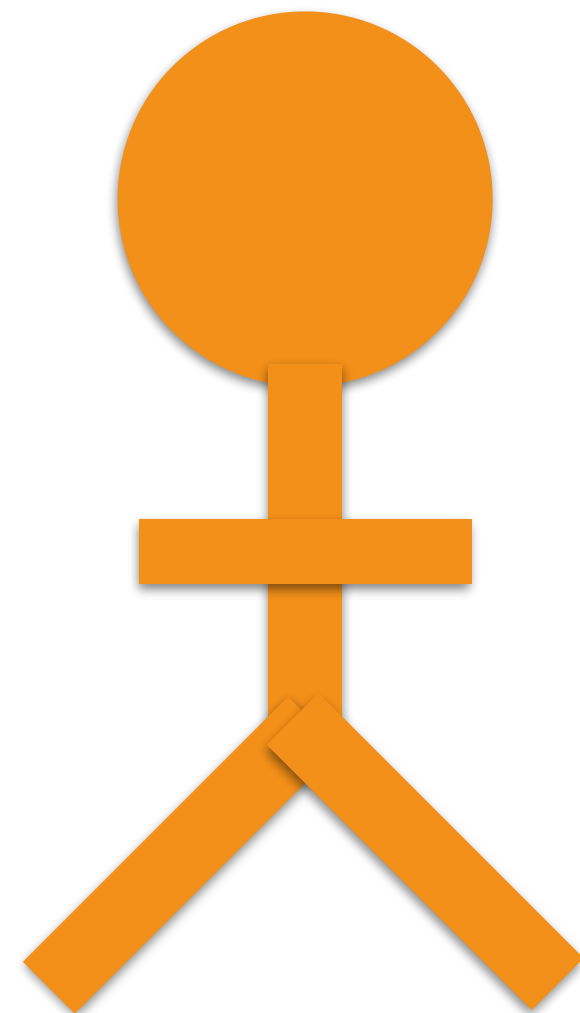
Mutable borrow

```
fn main() {
  let mut name = ...;
  → update(&mut name);
  println!("{}", name);
}
```

Lend the string
mutably

```
fn update(name: &mut String) {
  name.push_str("...");
}
```

Take a **mutable
reference** to a String



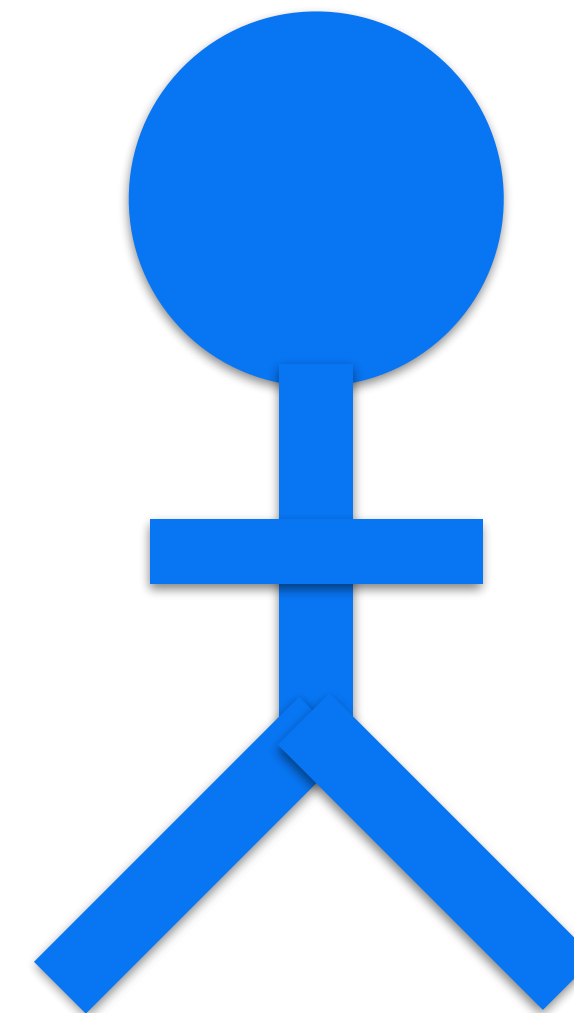
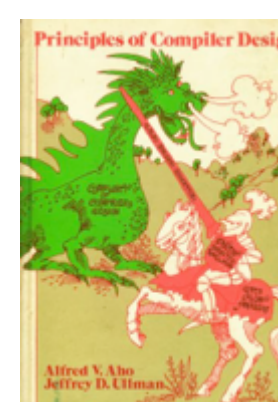
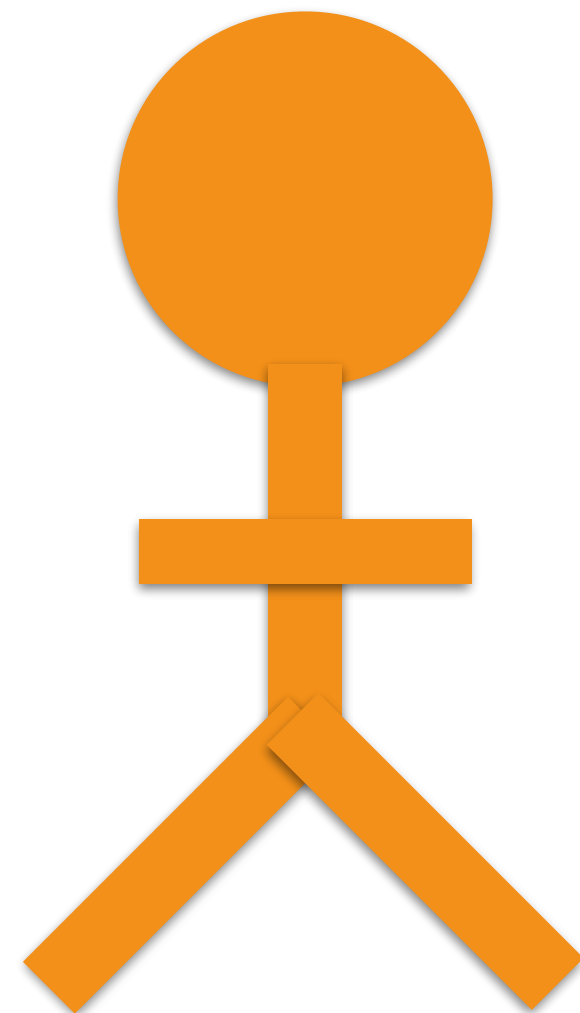
Mutable borrow

```
fn main() {
  let mut name = ...;
  → update(&mut name);
  println!("{}", name);
}
```

Lend the string
mutably

```
fn update(name: &mut String) {
  name.push_str("...");
}
```

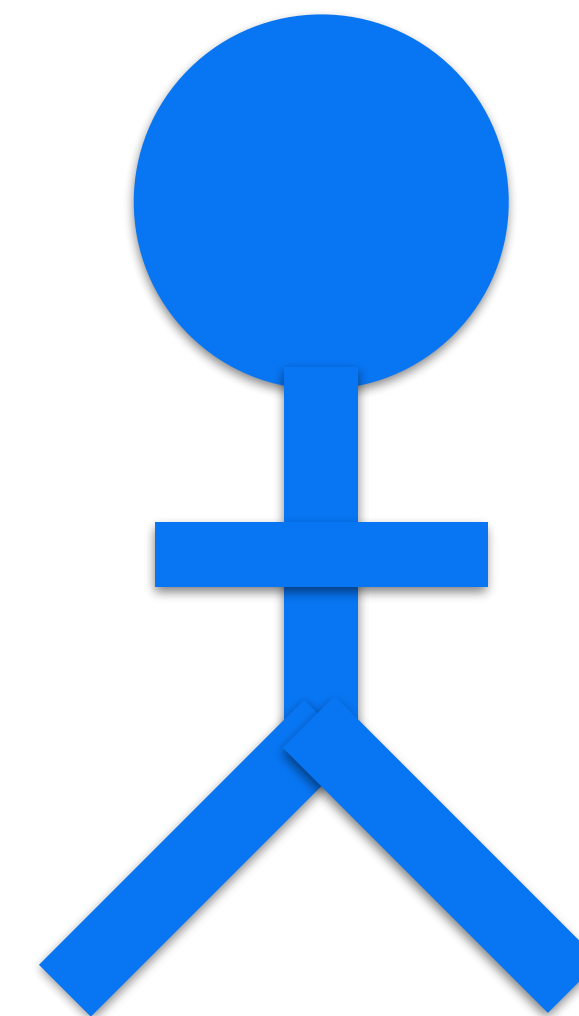
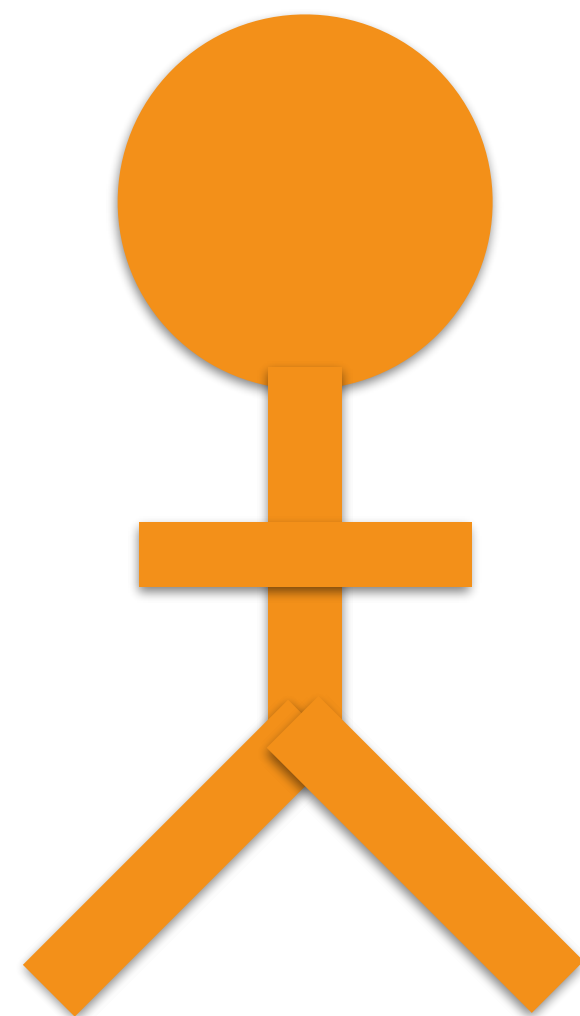
Take a **mutable
reference** to a String



Mutable borrow

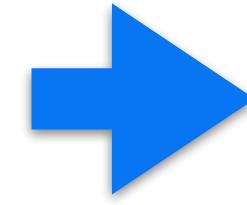
```
fn main() {  
    let mut name = ...;  
    → update(&mut name);  
    println!("{}", name);  
}
```

```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```

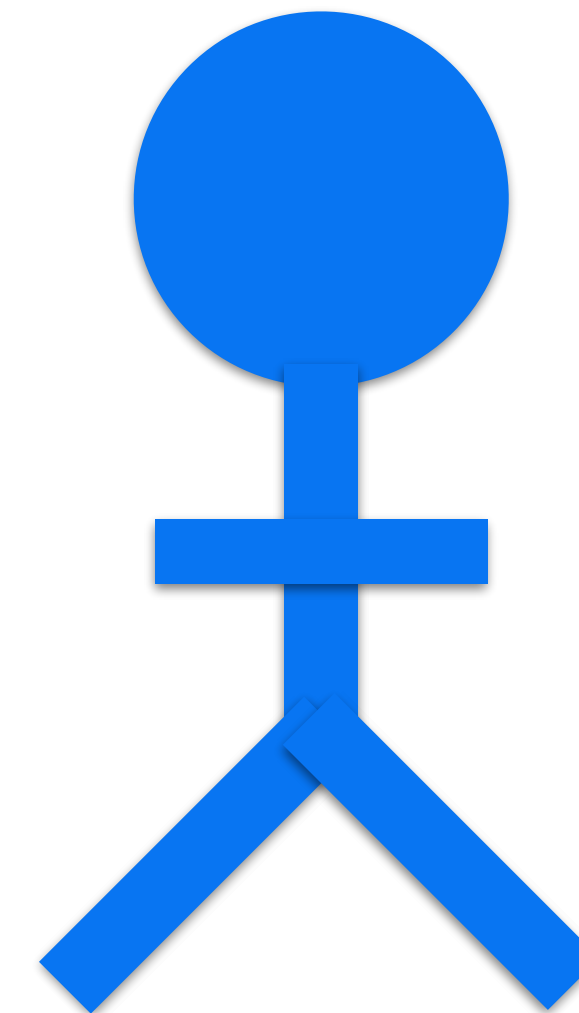
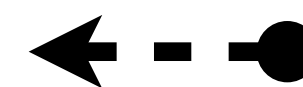
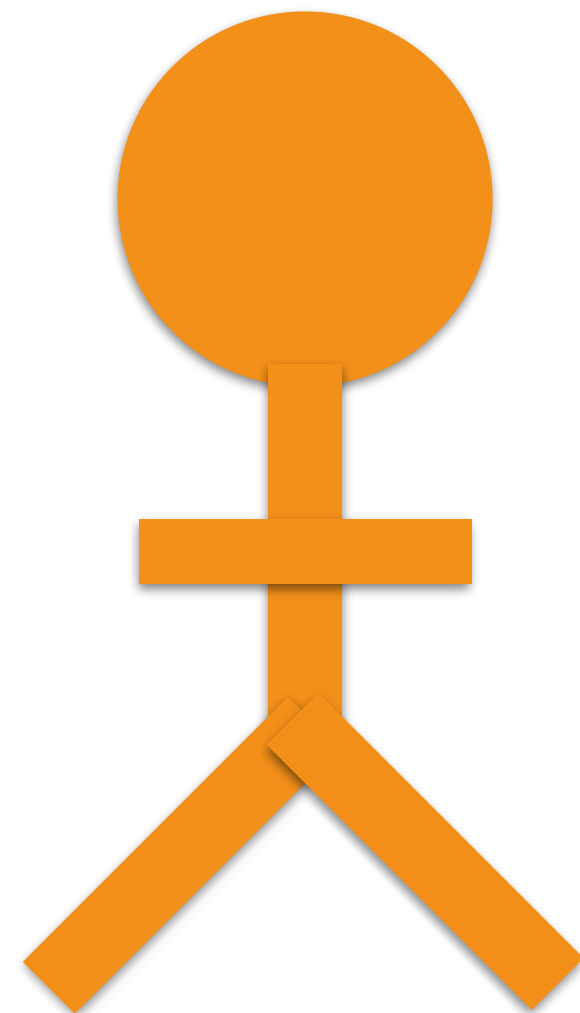


Mutable borrow

```
fn main() {  
    let mut name = ...;  
    update(&mut name);  
    println!("{}", name);  
}
```

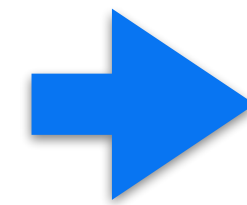


```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```

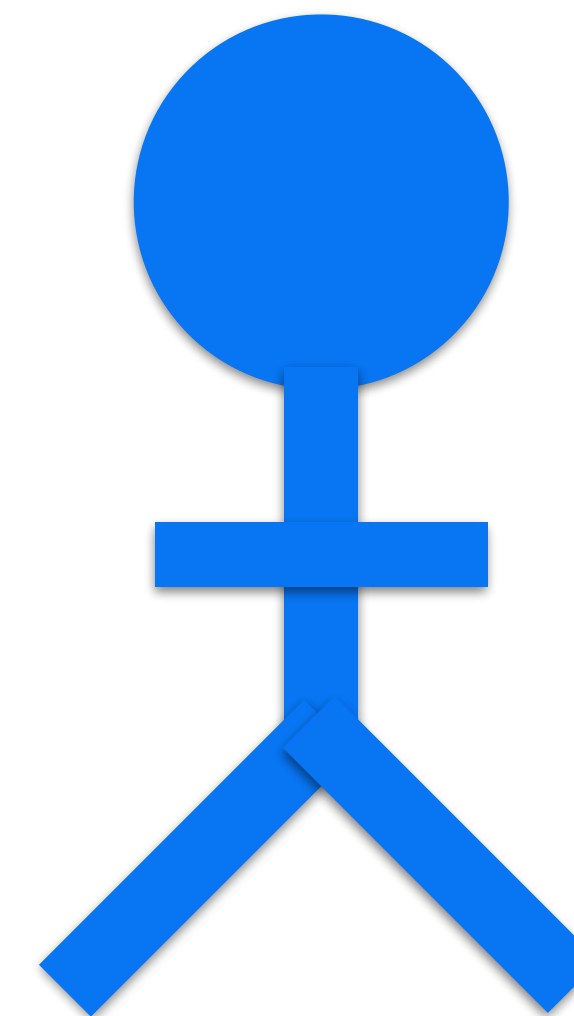
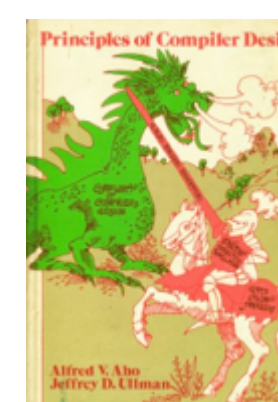
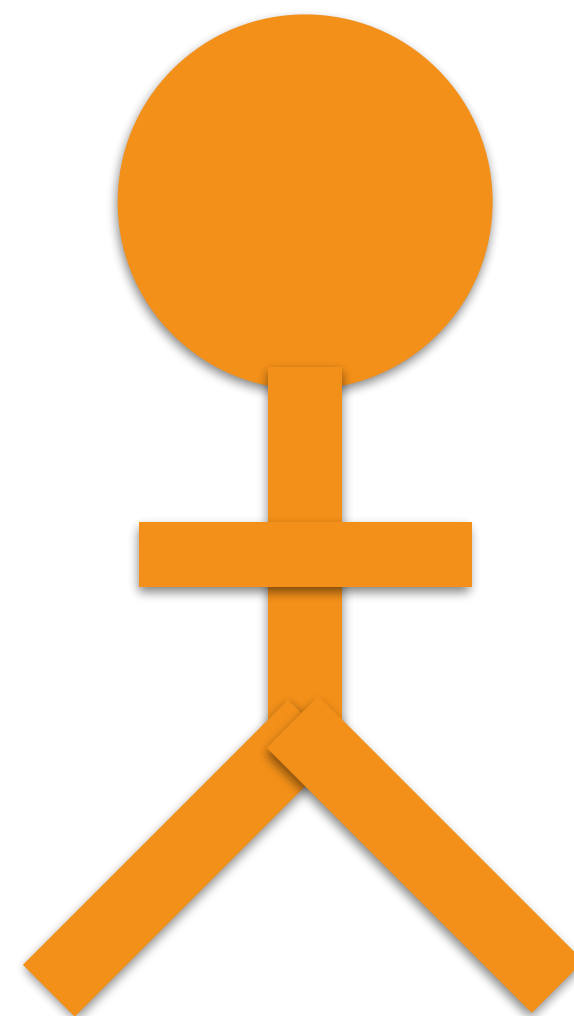


Mutable borrow

```
fn main() {  
    let mut name = ...;  
    update(&mut name);  
    println!("{}", name);  
}
```

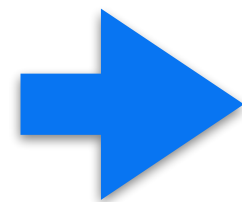


```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```



Mutable borrow

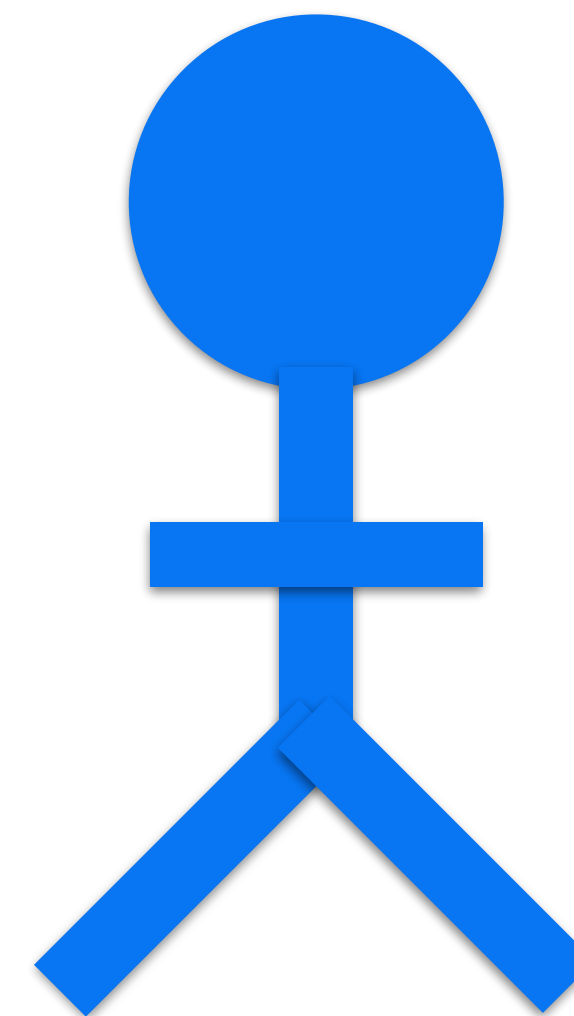
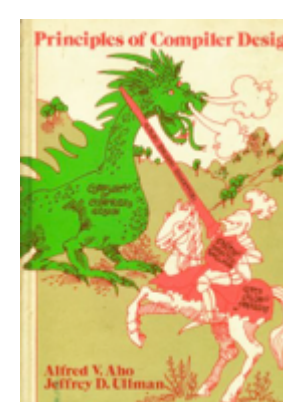
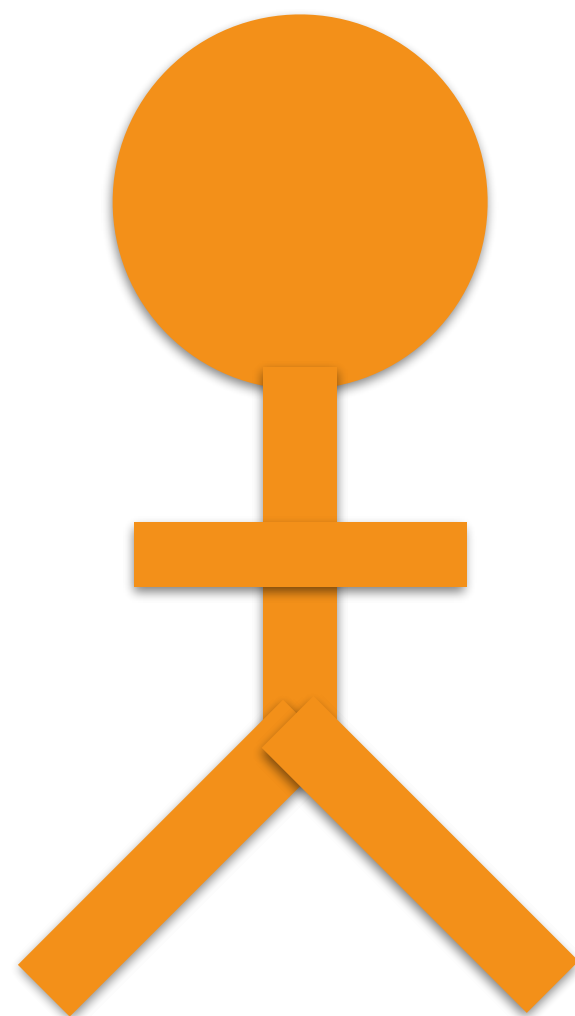

```
fn main() {
  let mut name = ...;
  update(&mut name);
  println!("{}", name);
}
```



```
fn update(name: &mut String) {
  name.push_str("...");
}
```

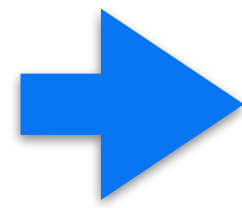


Mutate string
in place



Mutable borrow

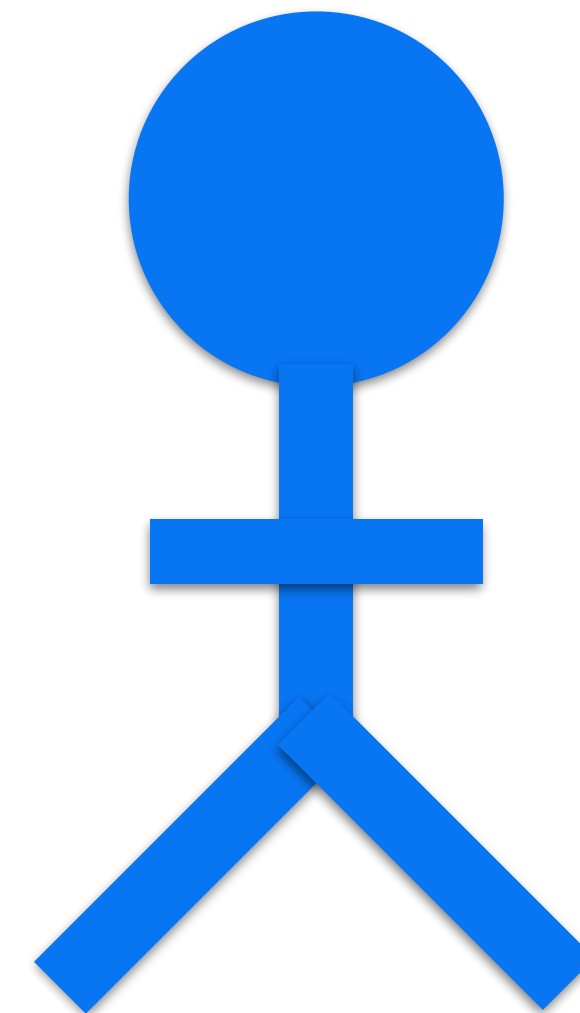
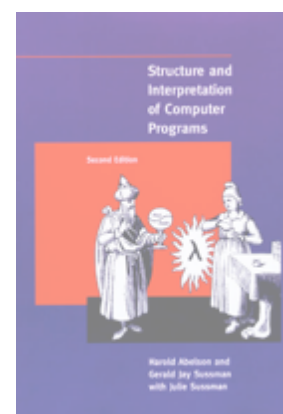
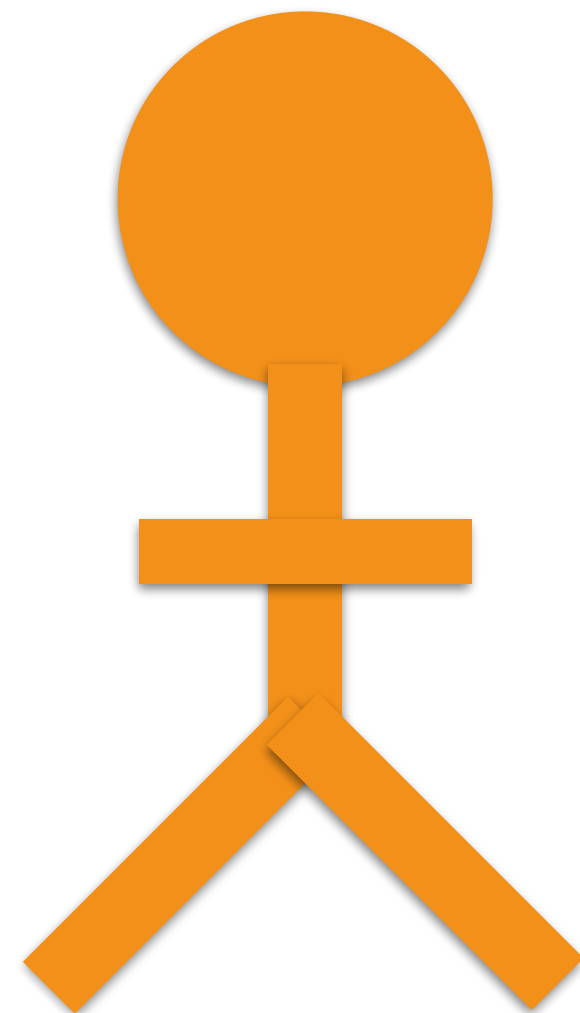

```
fn main() {
  let mut name = ...;
  update(&mut name);
  println!("{}", name);
}
```



```
fn update(name: &mut String) {
  name.push_str("...");
}
```

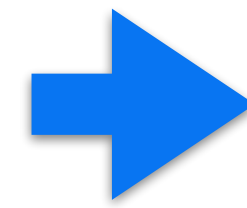


Mutate string
in place

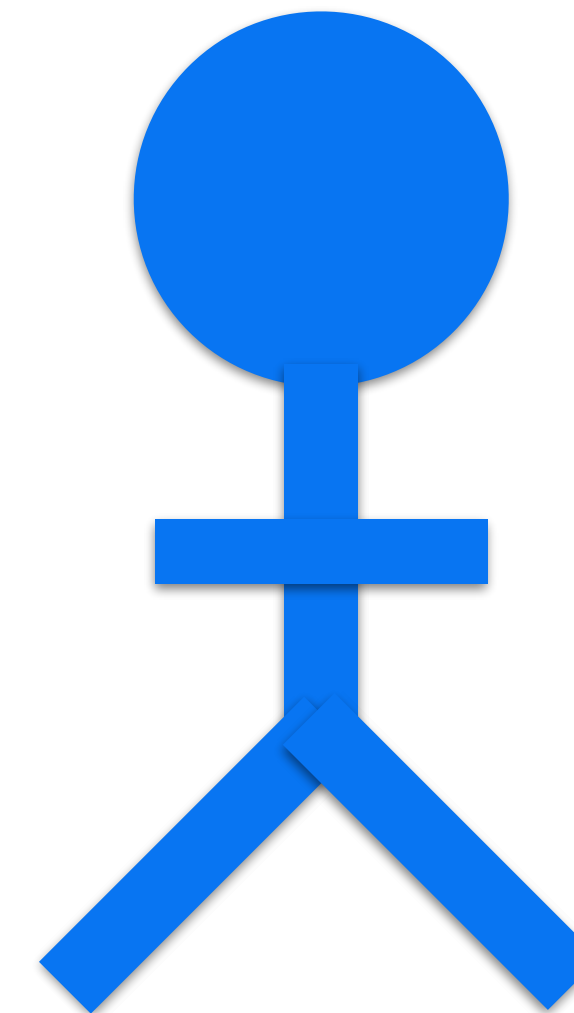
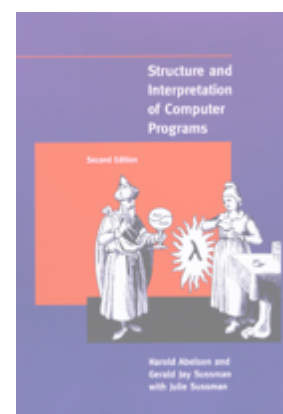
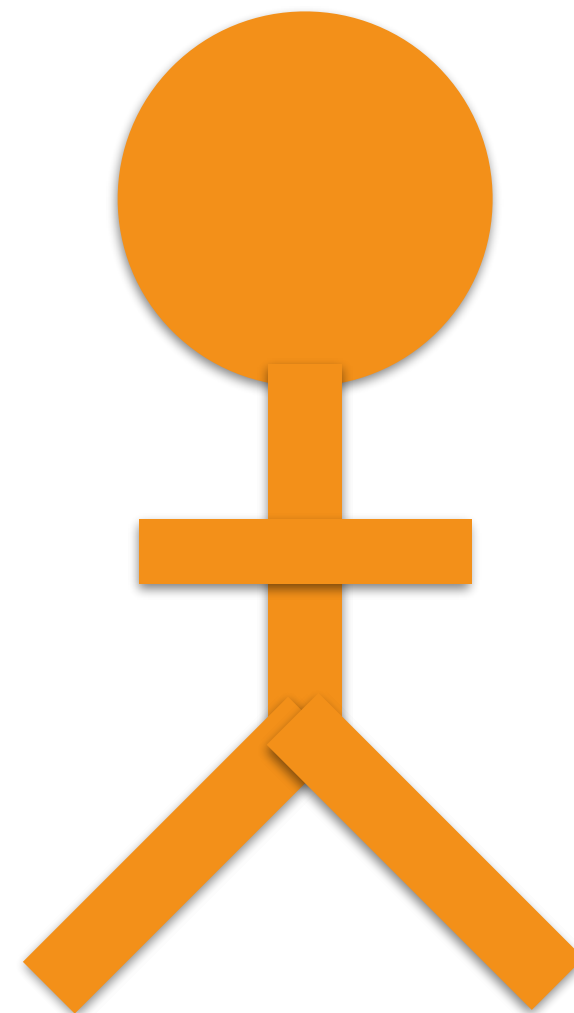


Mutable borrow

```
fn main() {  
  let mut name = ...;  
  update(&mut name);  
  println!("{}", name);  
}
```

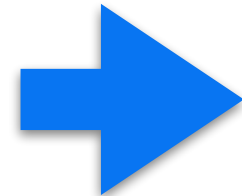


```
fn update(name: &mut String) {  
  name.push_str("...");  
}
```

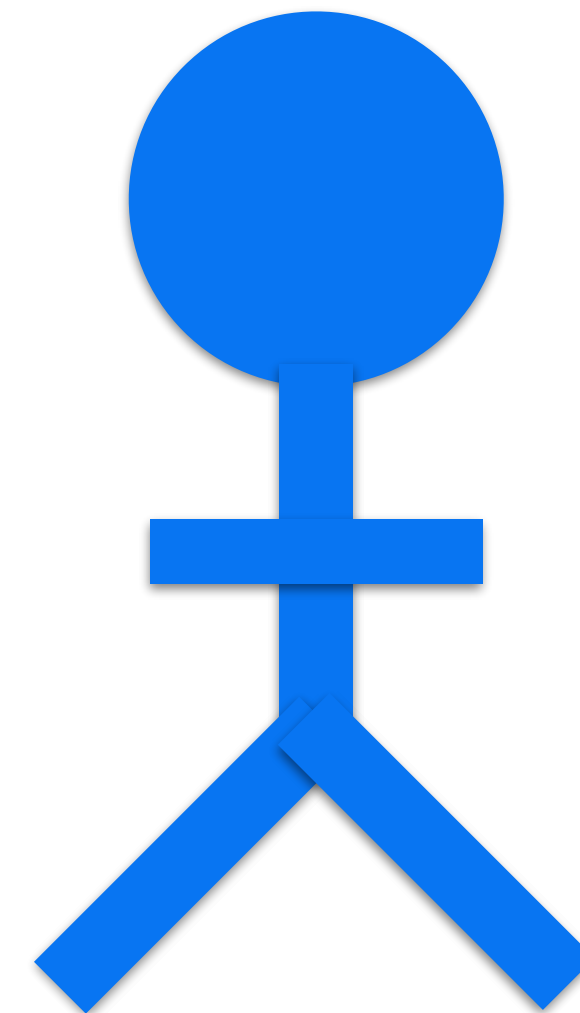
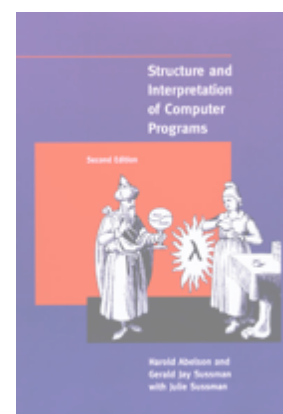
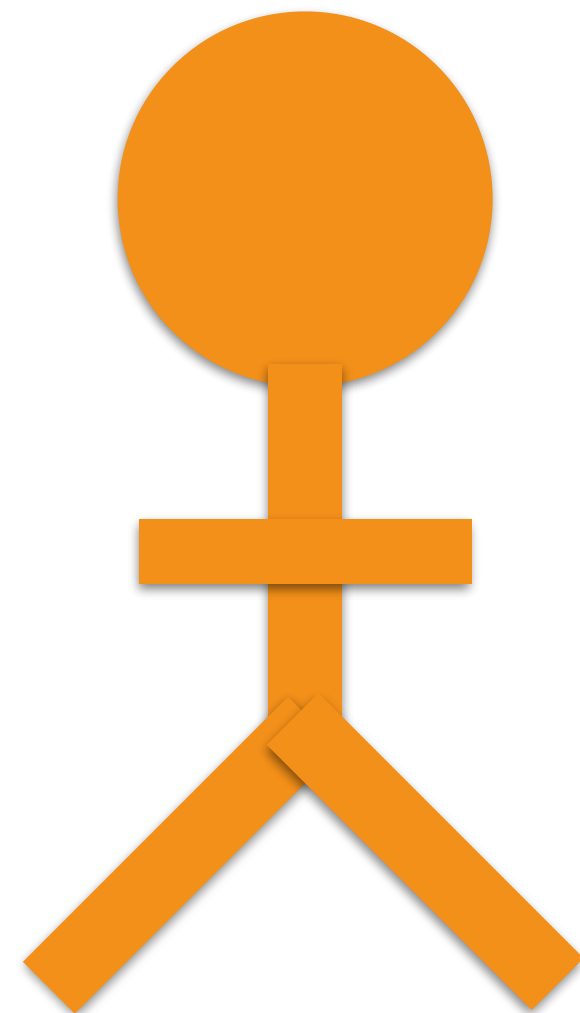


Mutable borrow

```
fn main() {  
    let mut name = ...;  
    update(&mut name);  
    println!("{}", name);  
}
```

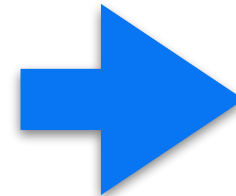


```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```

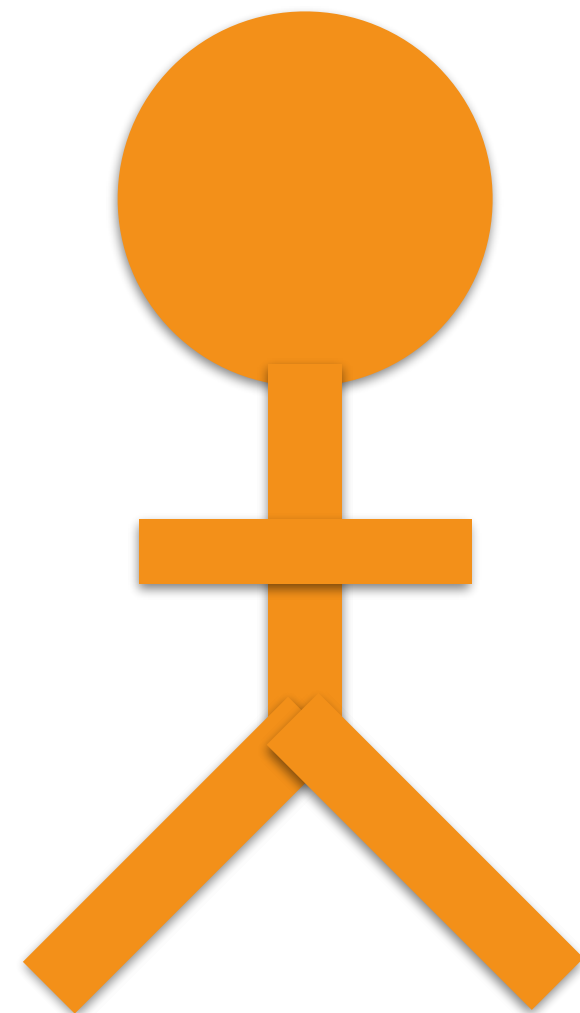


Mutable borrow

```
fn main() {  
    let mut name = ...;  
    update(&mut name);  
    println!("{}", name);  
}
```



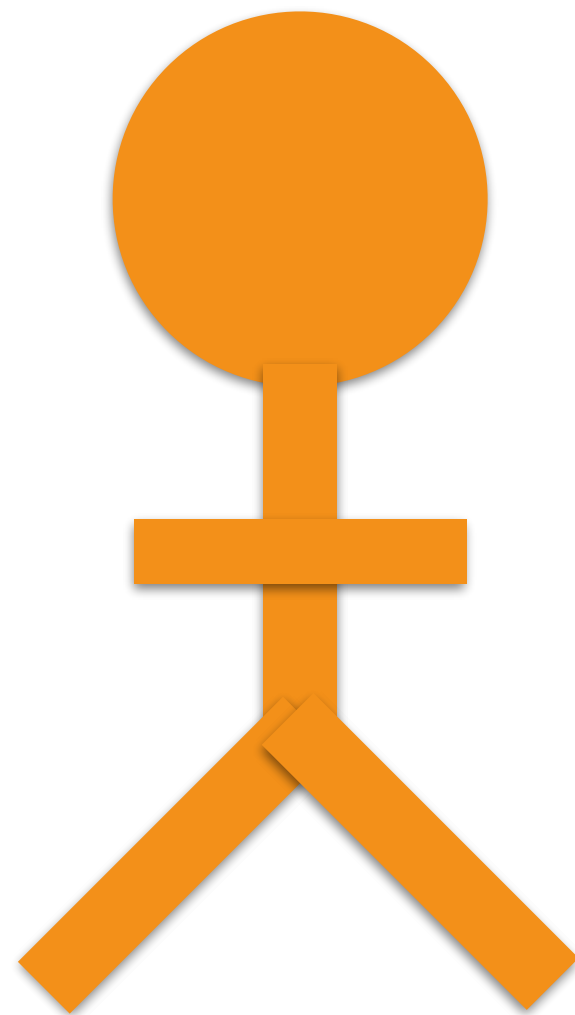
```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```



Mutable borrow

```
fn main() {  
    let mut name = ...;  
    update(&mut name);  
    println!("{}", name);  
}
```

```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```

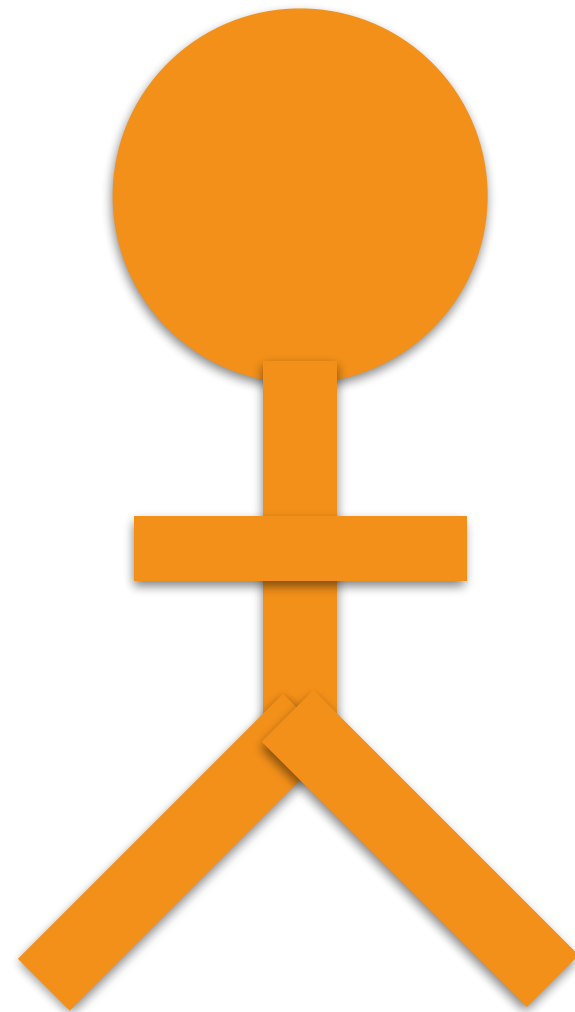


Mutable borrow

```
fn main() {  
    let mut name = ...;  
    update(&mut name);  
    println!("{}", name);  
}
```

```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```

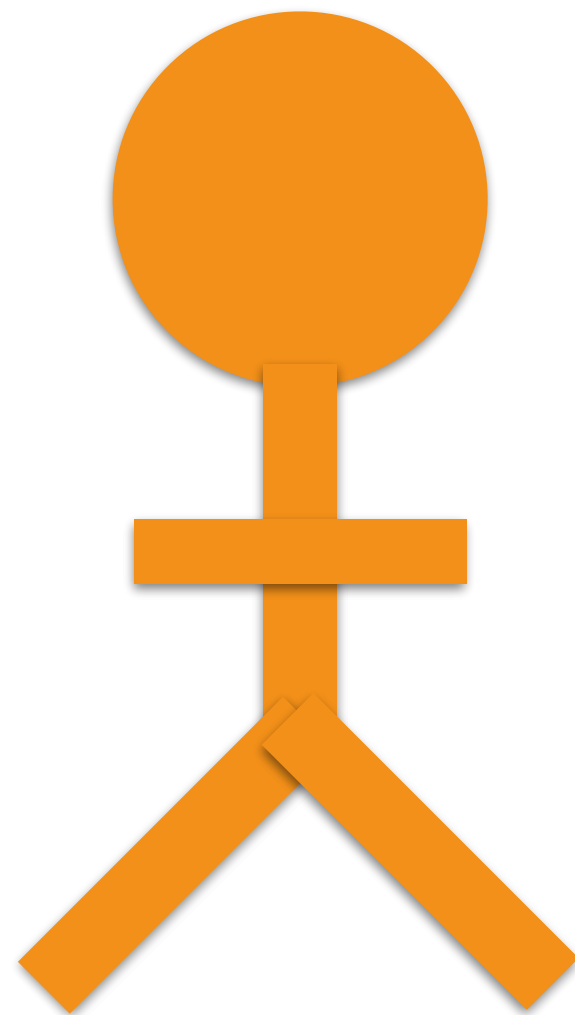
Prints the
updated string.



Mutable borrow

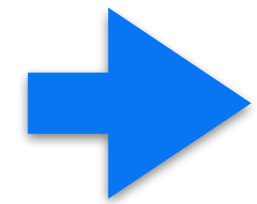
```
fn main() {  
    let mut name = ...;  
    update(&mut name);  
    println!("{}", name);  
}
```

```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```

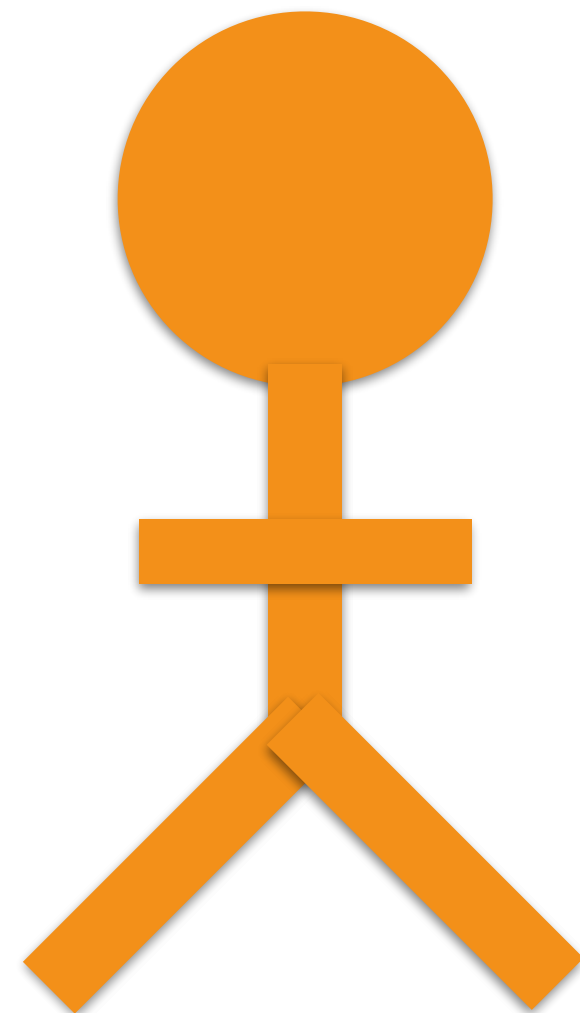


Mutable borrow

```
fn main() {  
    let mut name = ...;  
    update(&mut name);  
    println!("{}", name);  
}
```

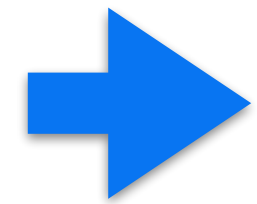


```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```



Mutable borrow


```
fn main() {  
    let mut name = ...;  
    update(&mut name);  
    println!("{}", name);  
}
```



```
fn update(name: &mut String) {  
    name.push_str("...");  
}
```

Mutable borrow

name: String

Ownership:

control all access, will free when done

name: &String

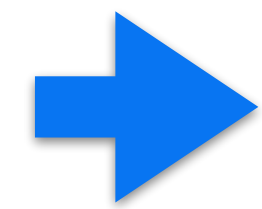
Shared reference:

many readers, no writers

name: &mut String

Mutable reference:

no readers, one writer



name: String

Ownership:

control all access, will free when done

name: &String

Shared reference:

many readers, no writers

name: &mut String

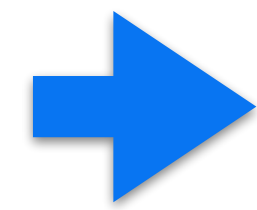
Mutable reference:

no readers, one writer

name: String

Ownership:

control all access, will free when done



name: &String

Shared reference:

many readers, no writers

name: &mut String

Mutable reference:

no readers, one writer

name: String

Ownership:

control all access, will free when done

name: &String

Shared reference:

many readers, no writers



name: &mut String

Mutable reference:

no readers, one writer

Play time



Waterloo, Cassius Coolidge, c. 1906

How do we get safety?



Dangers of mutation

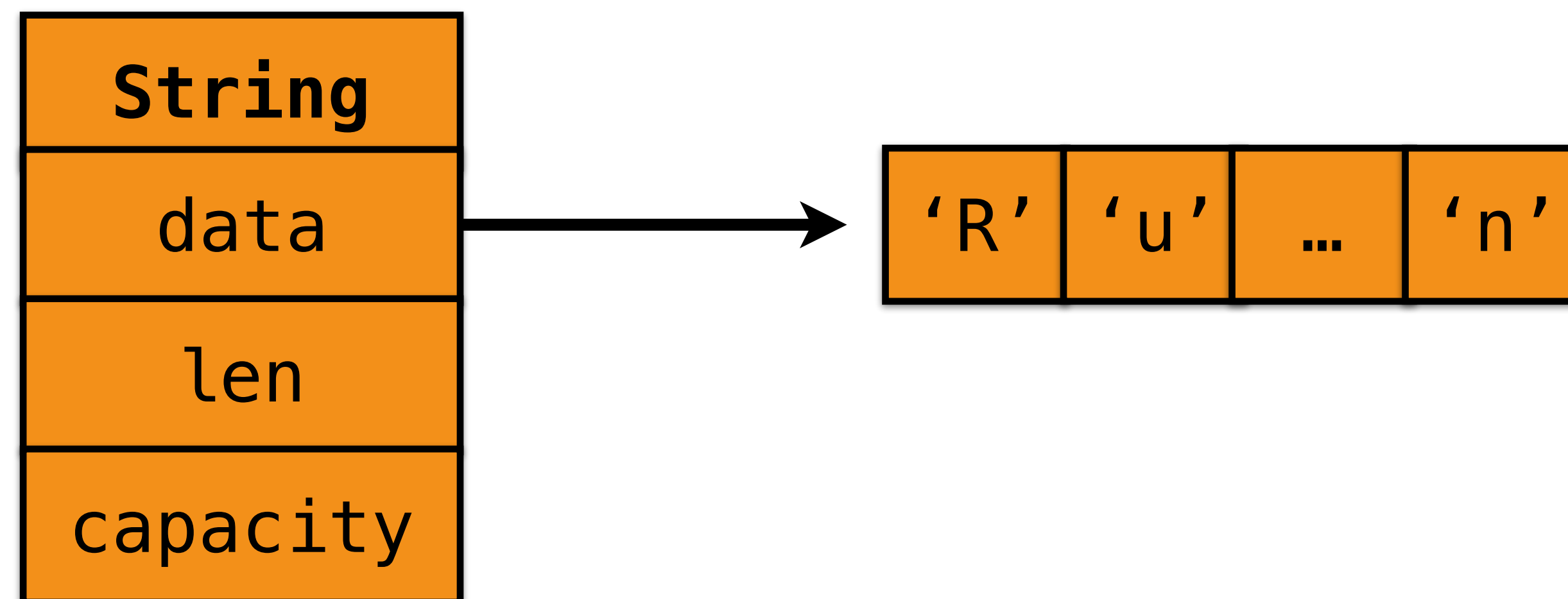
```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```


Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```

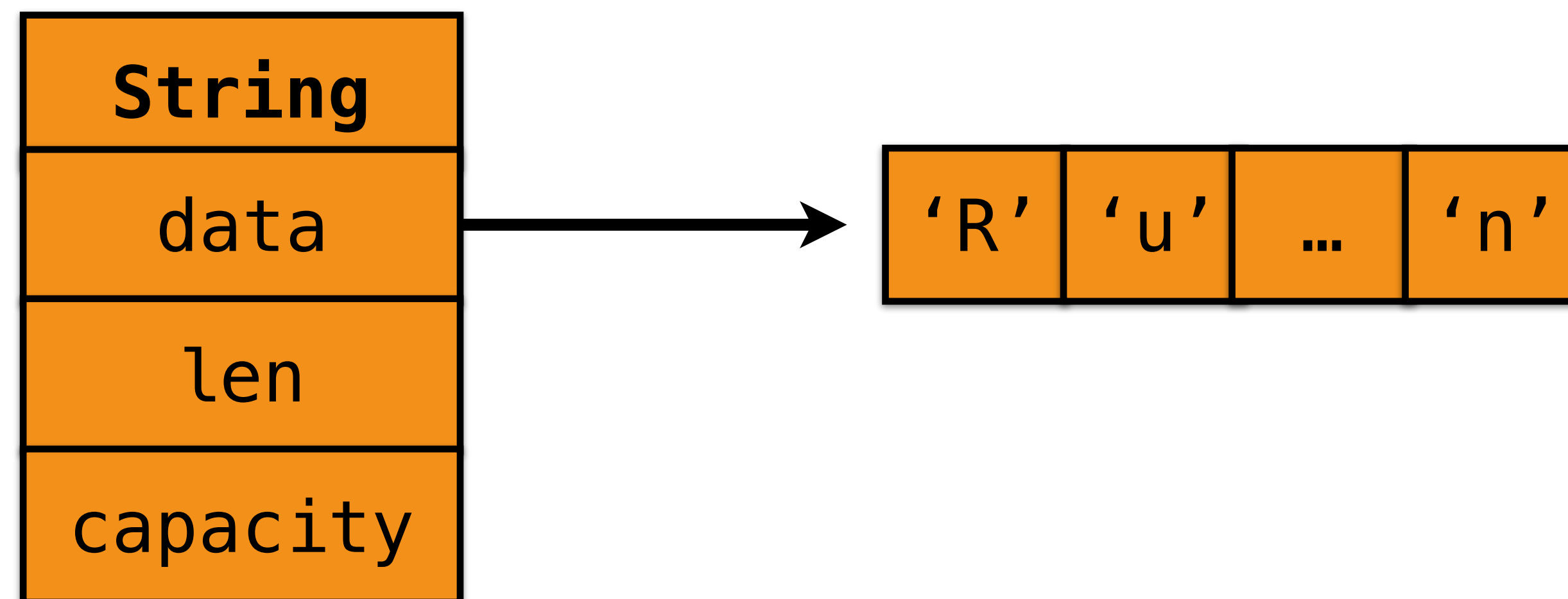
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



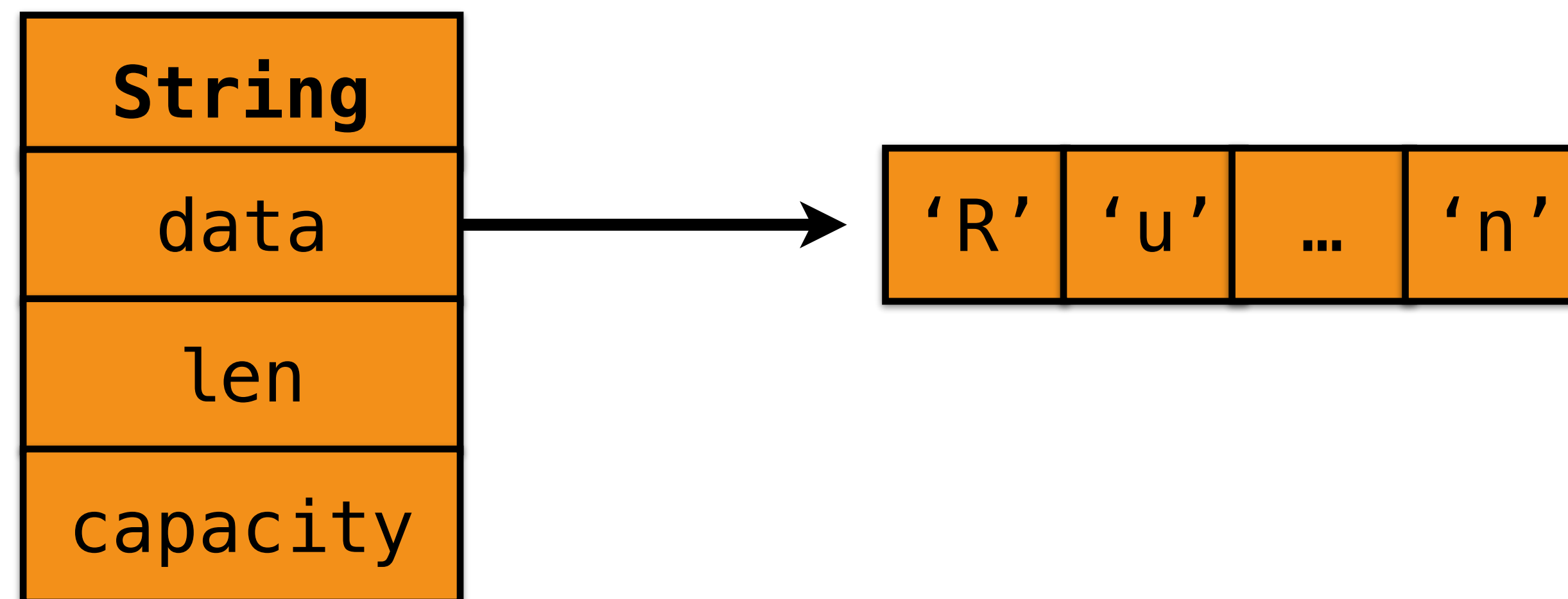
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



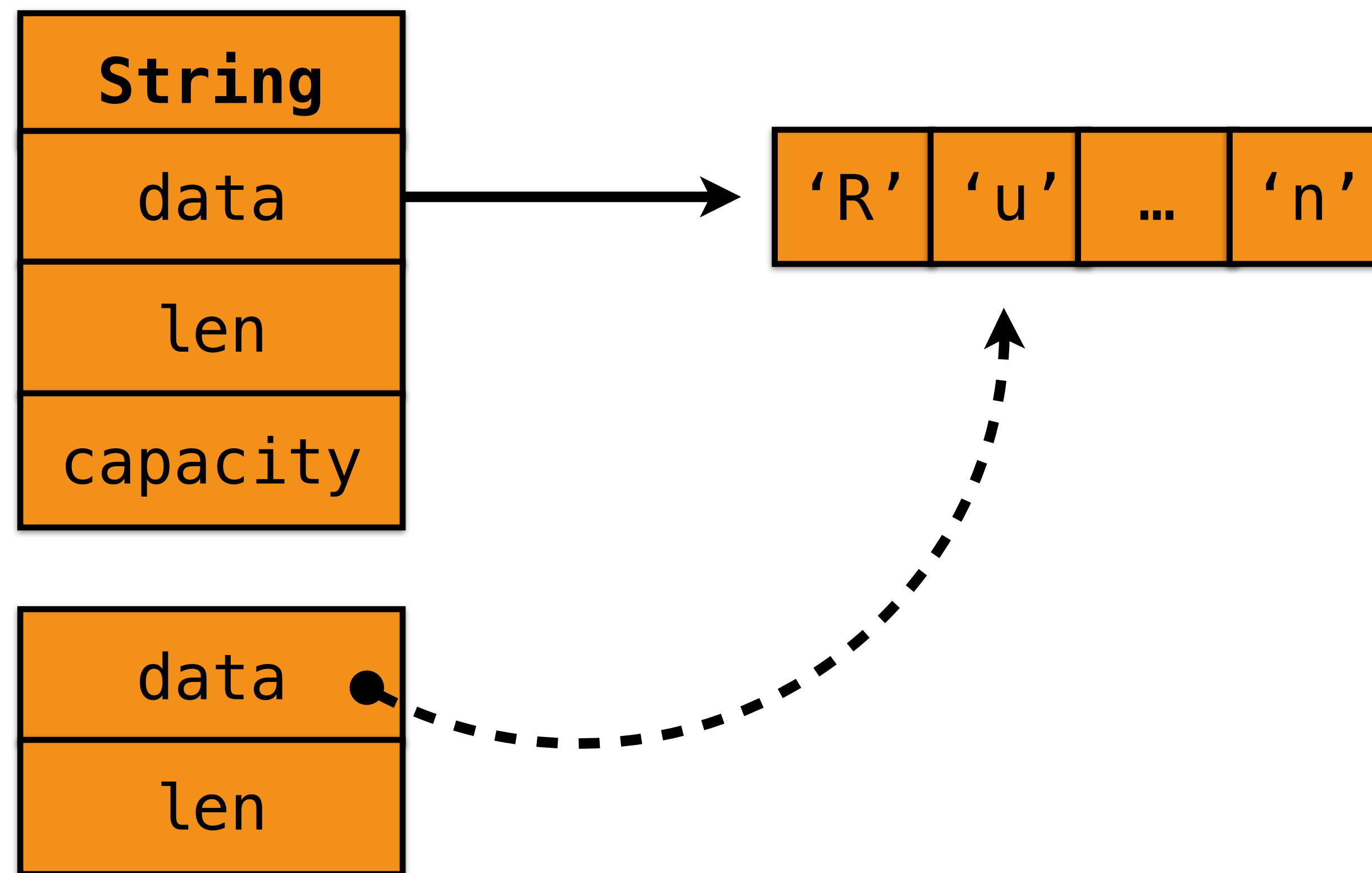
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



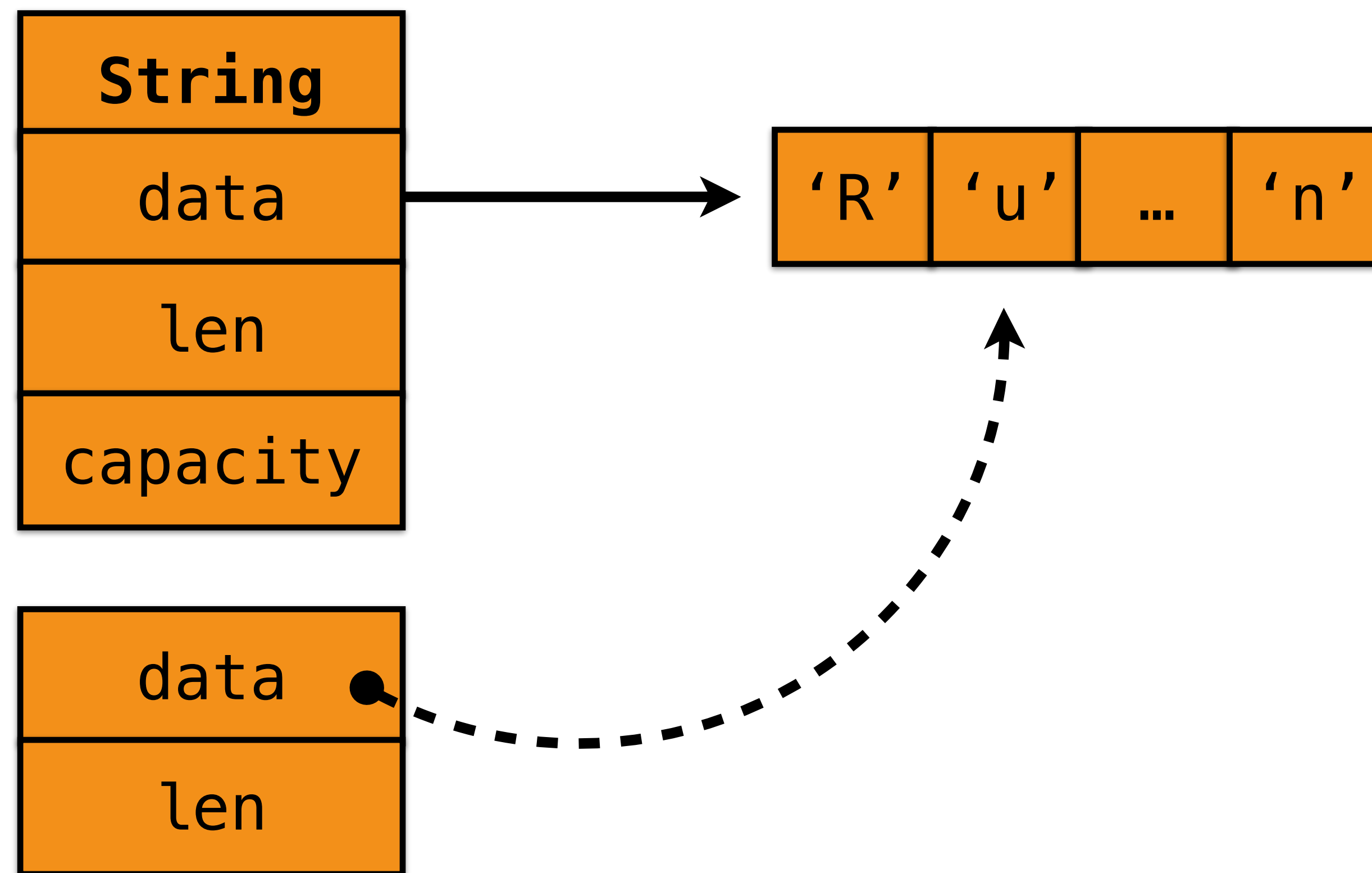
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



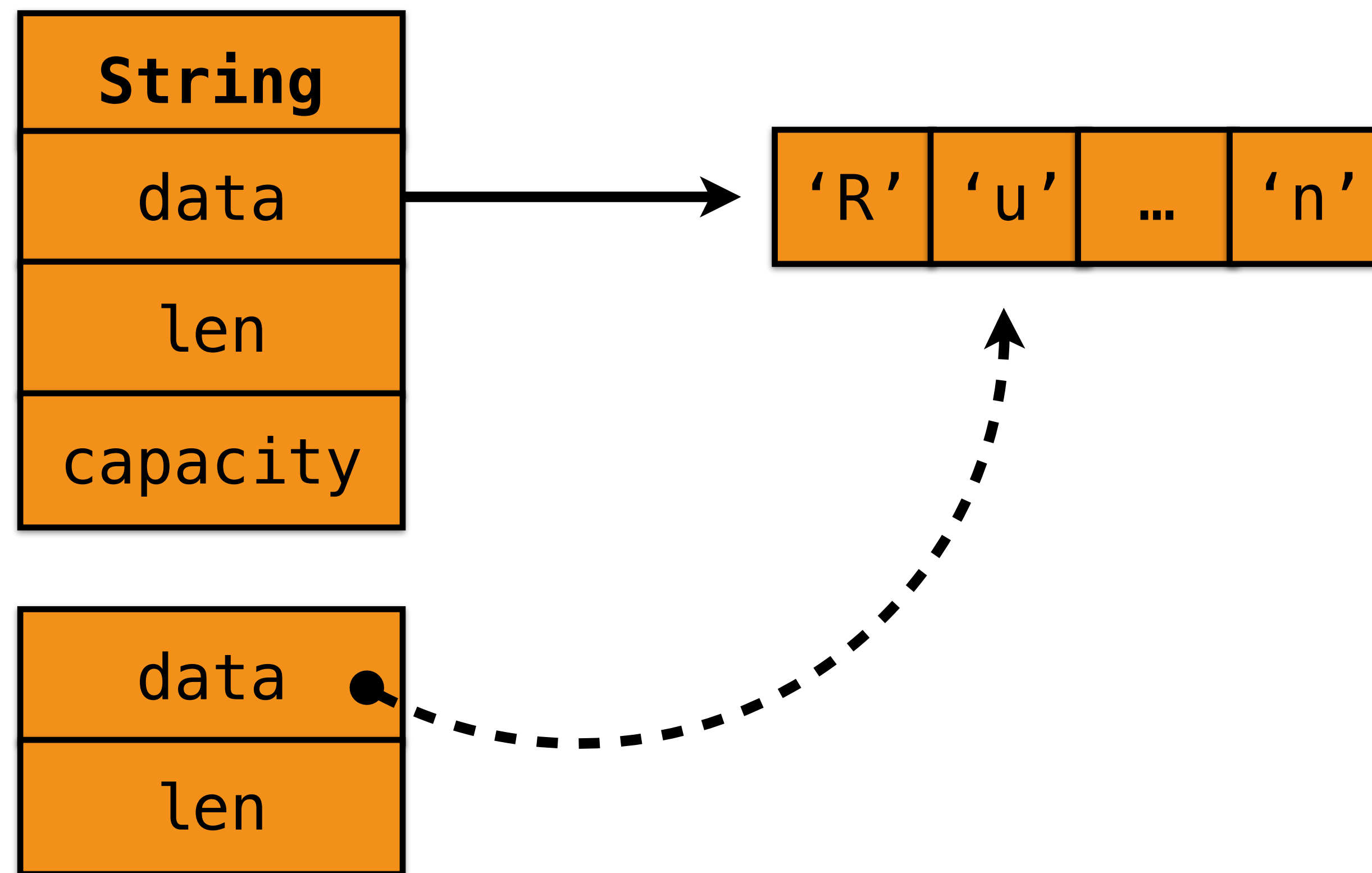
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



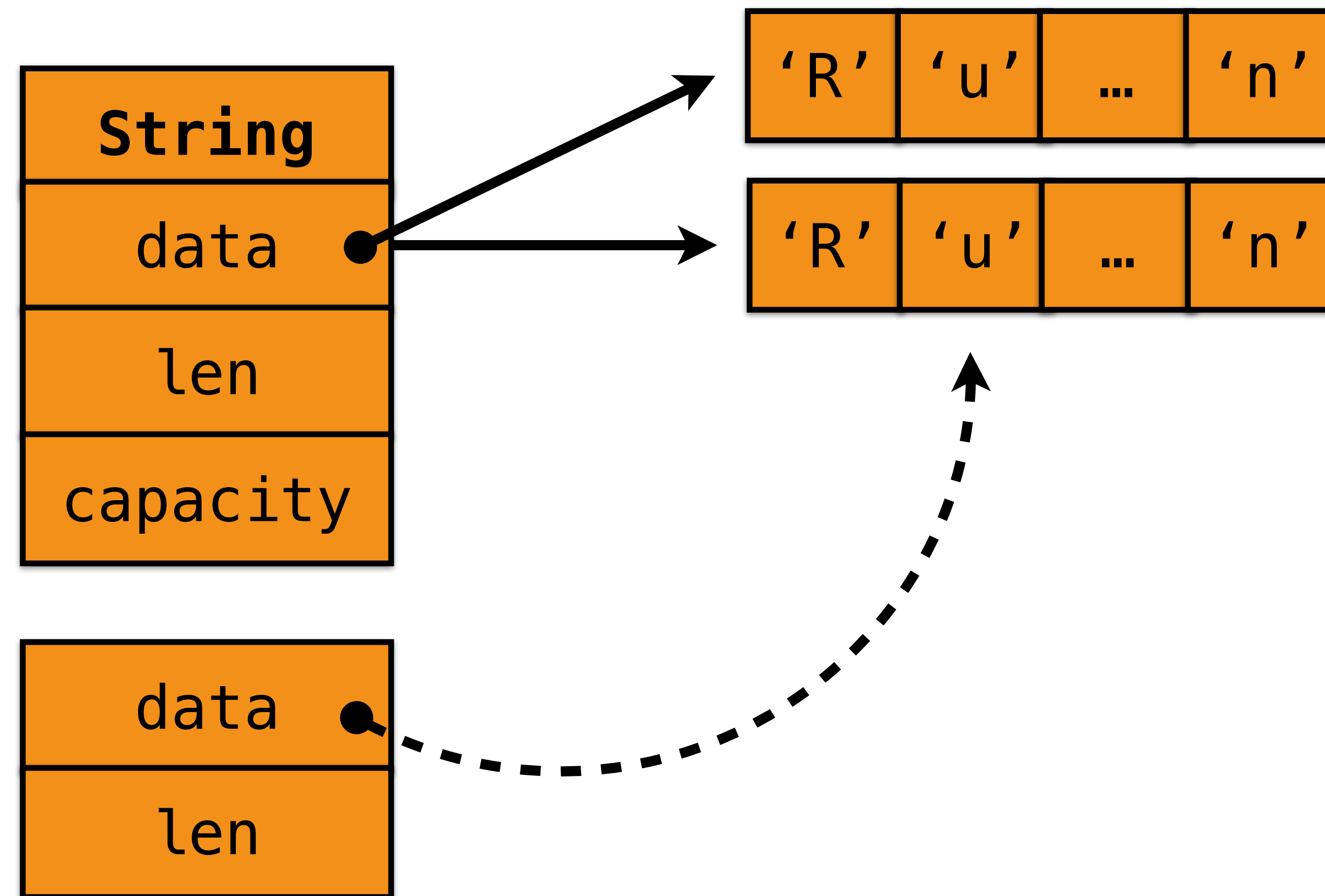
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



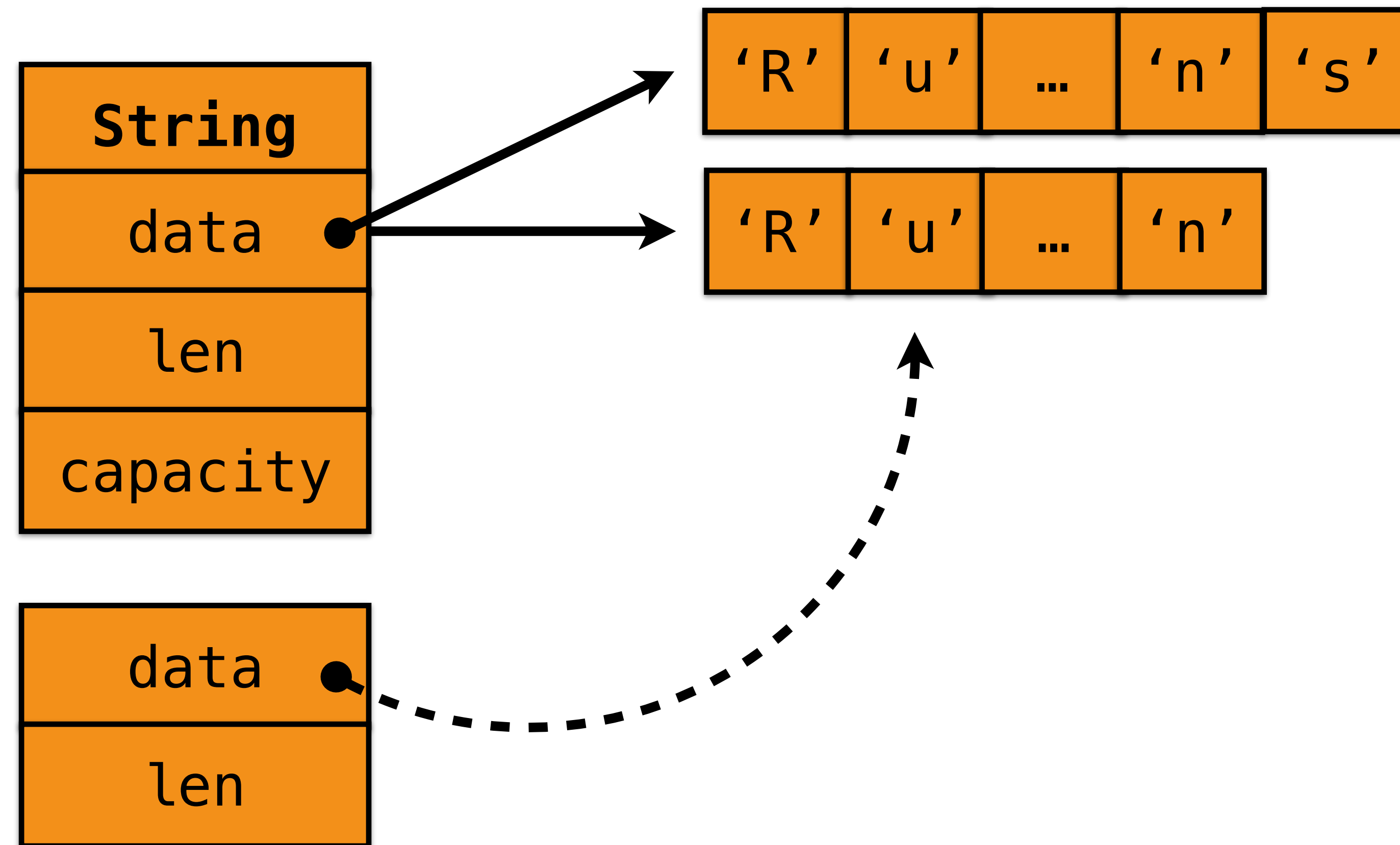
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



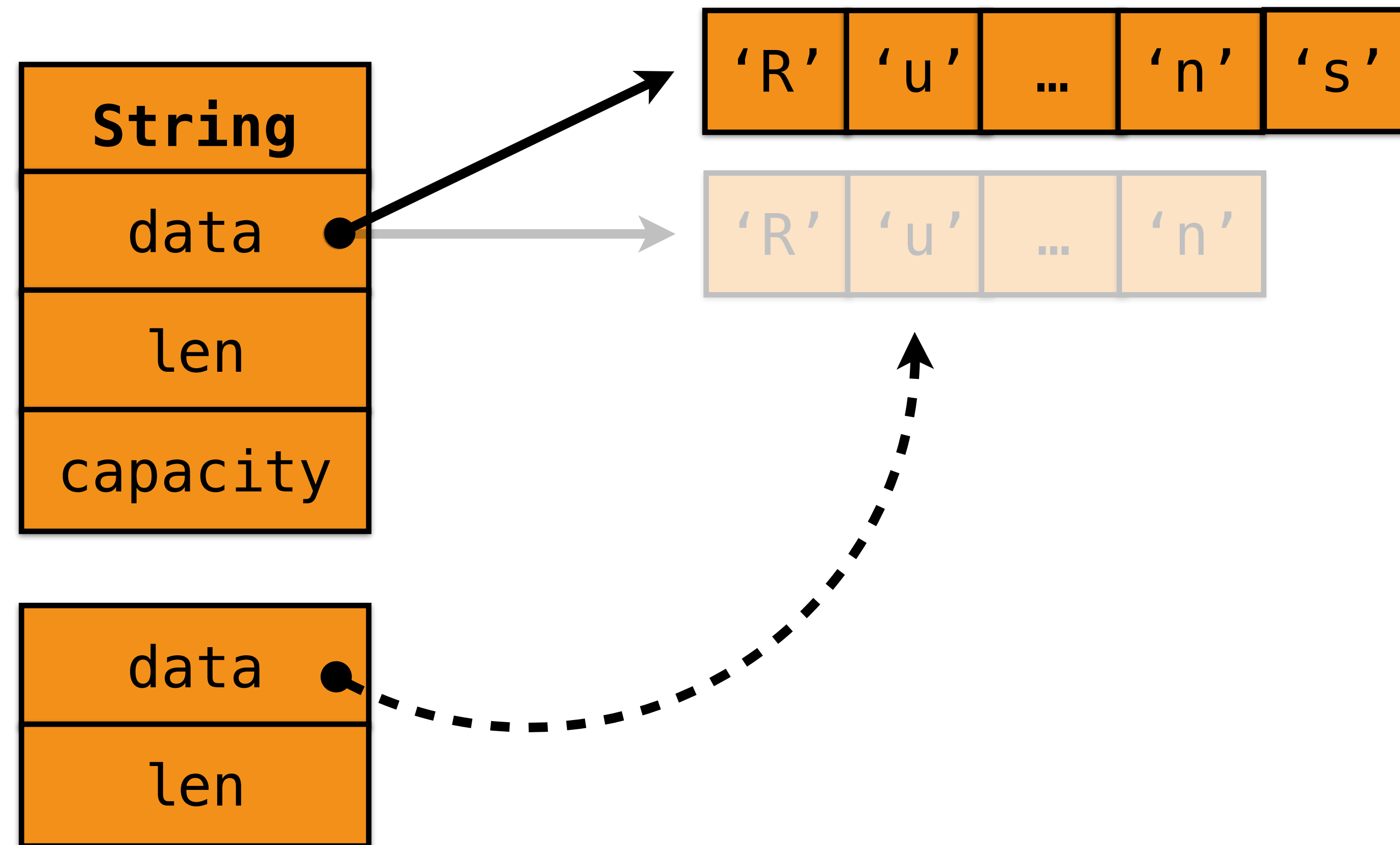
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



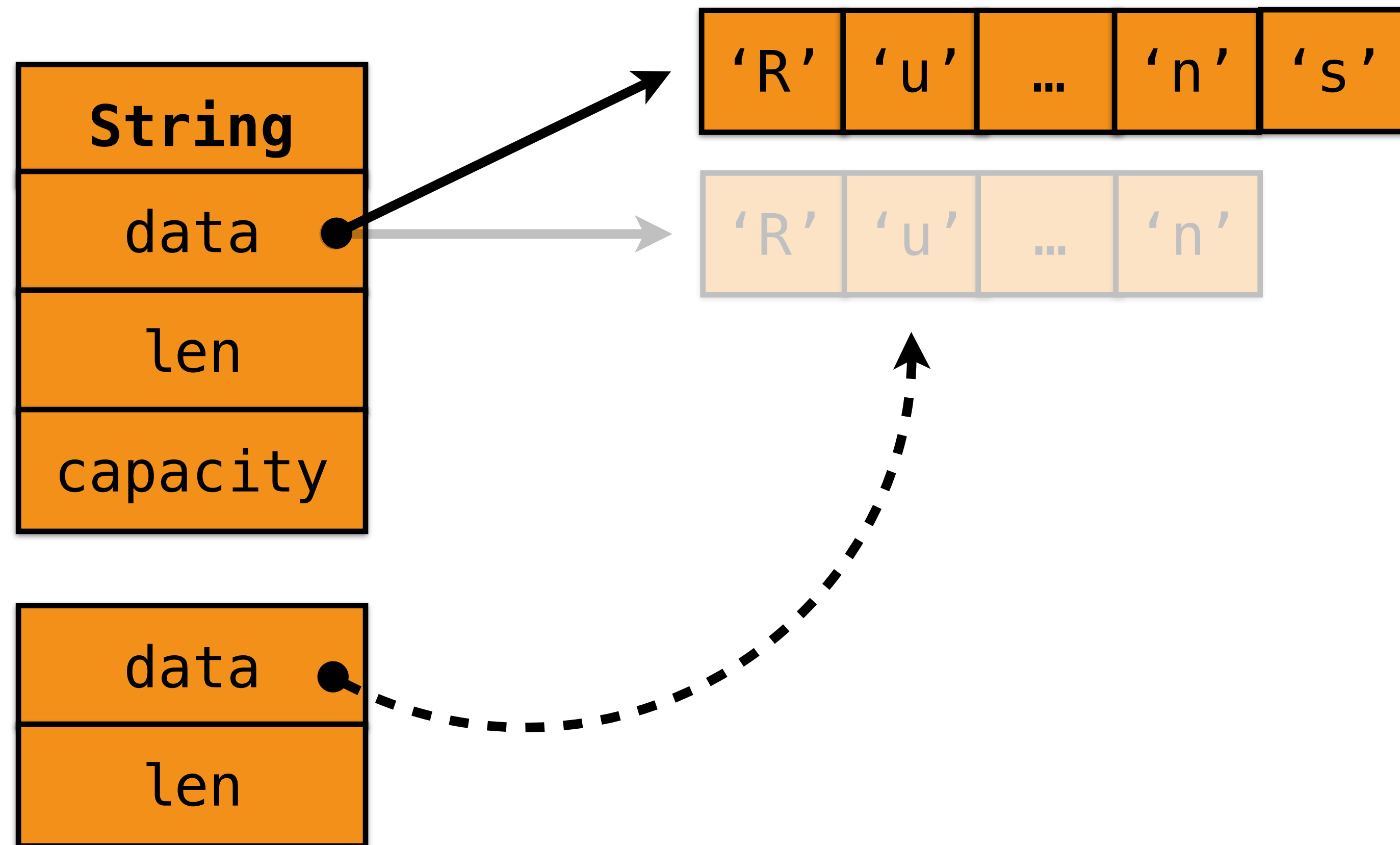
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



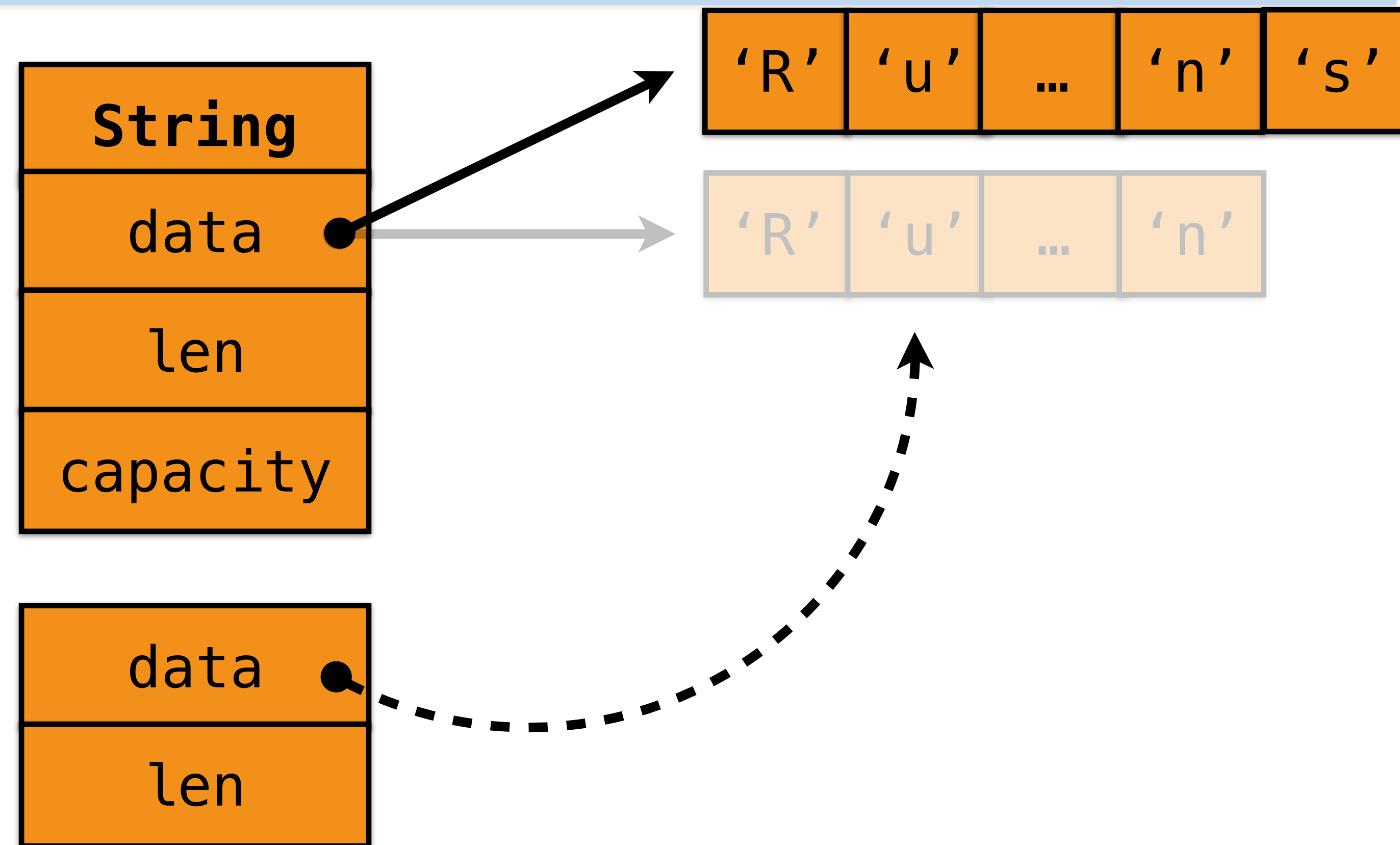
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



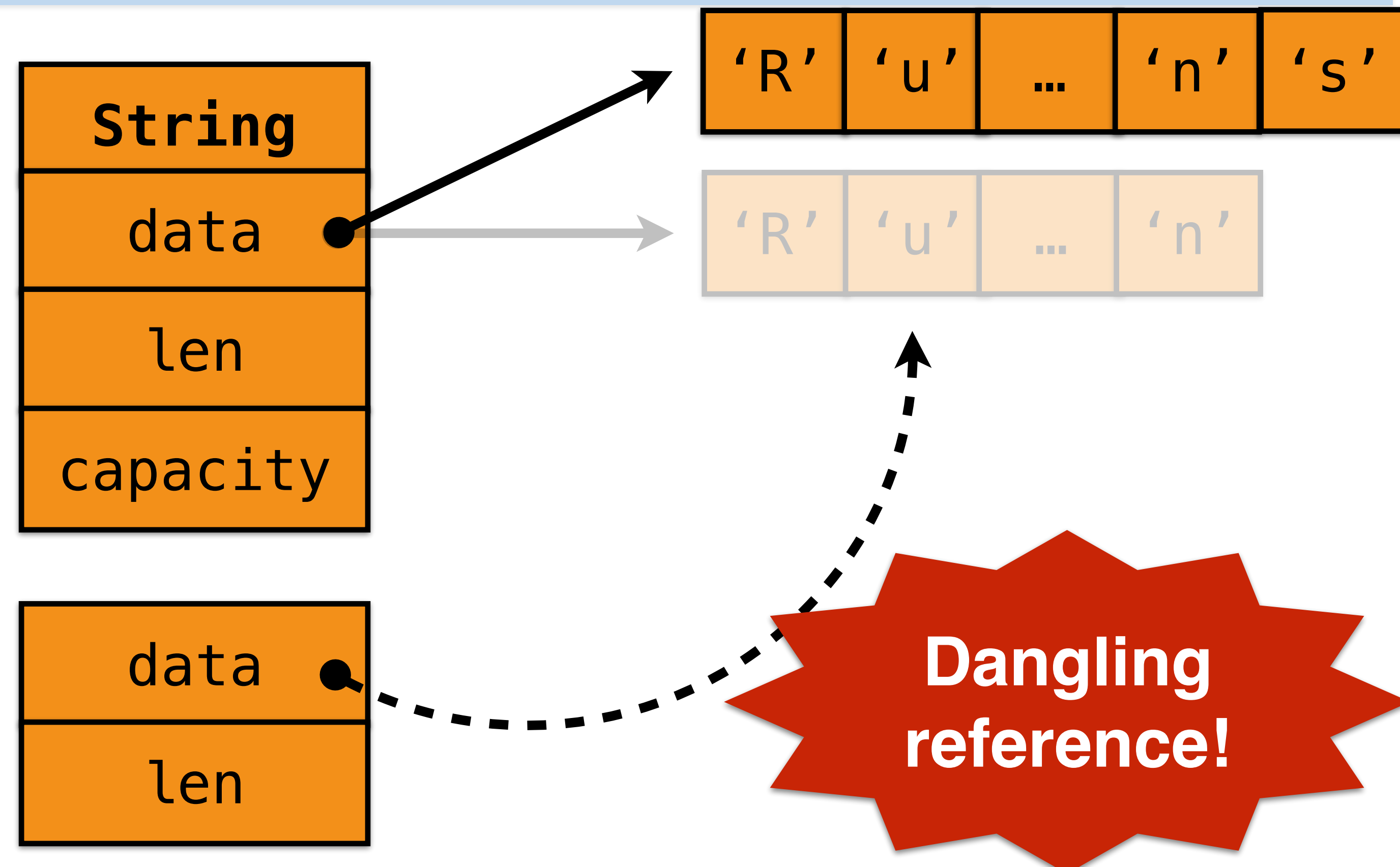
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



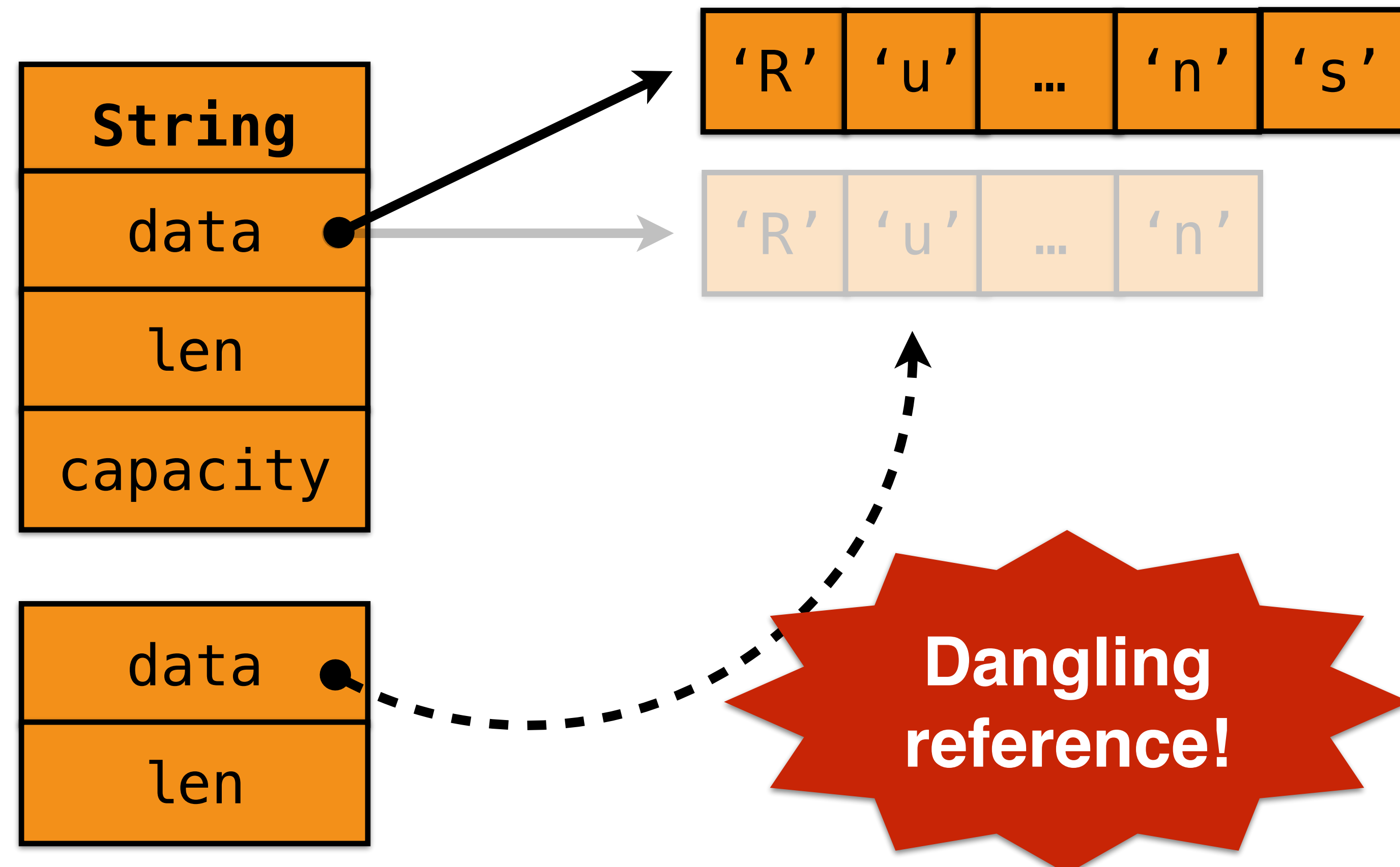
Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



Dangers of mutation

```
let mut buffer: String = format!("Rustacean");  
let slice = &buffer[1..];  
buffer.push_str("s");  
println!("{:?}", slice);
```



Rust solution

Compile-time read-write-lock:

Creating a shared reference to X “**read locks**” X.

- Other readers OK.
- No writers.
- Lock lasts until reference goes out of scope.

Creating a mutable reference to X “**writes locks**” X.

- No other readers or writers.
- Lock lasts until reference goes out of scope.

Never have a reader/writer at same time.


```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    let slice = &buffer[1..];  
    buffer.push_str("s");  
    println!("{:?}", slice);  
}
```

Rule: No mutation during **lifetime of borrow**.

Lifetime: span of code where reference is used.

```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    let slice = &buffer[1..];  
    buffer.push_str("s");  
    println!("{:?}", slice);  
}
```

Rule: No mutation during **lifetime of borrow**.

Lifetime: span of code where reference is used.

```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    let slice = &buffer[1..];  
    buffer.push_str("s");  
    println!("{:?}", slice);  
}
```

Rule: No mutation during **lifetime of borrow**.


Lifetime: span of code where reference is used.

```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    let slice = &buffer[1..];  
    buffer.push_str("s");  
    println!("{:?}", slice);  
}
```

Rule: No mutation during **lifetime of borrow**.

Lifetime: span of code where reference is used.

```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    let slice = &buffer[1..];  
    buffer.push_str("s");  
    println!("{:?}", slice);  
}
```



Rule: No mutation during **lifetime of borrow**.

Lifetime: span of code where reference is used.

Borrow “locks”
`buffer` for lifetime of
resulting reference

```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    let slice = &buffer[1..];  
    buffer.push_str("s");  
    println!("{:?}", slice);  
}
```


Rule: No mutation during **lifetime of borrow**.

Lifetime: span of code where reference is used.

Borrow “locks”
`buffer` for lifetime `l`
of resulting reference

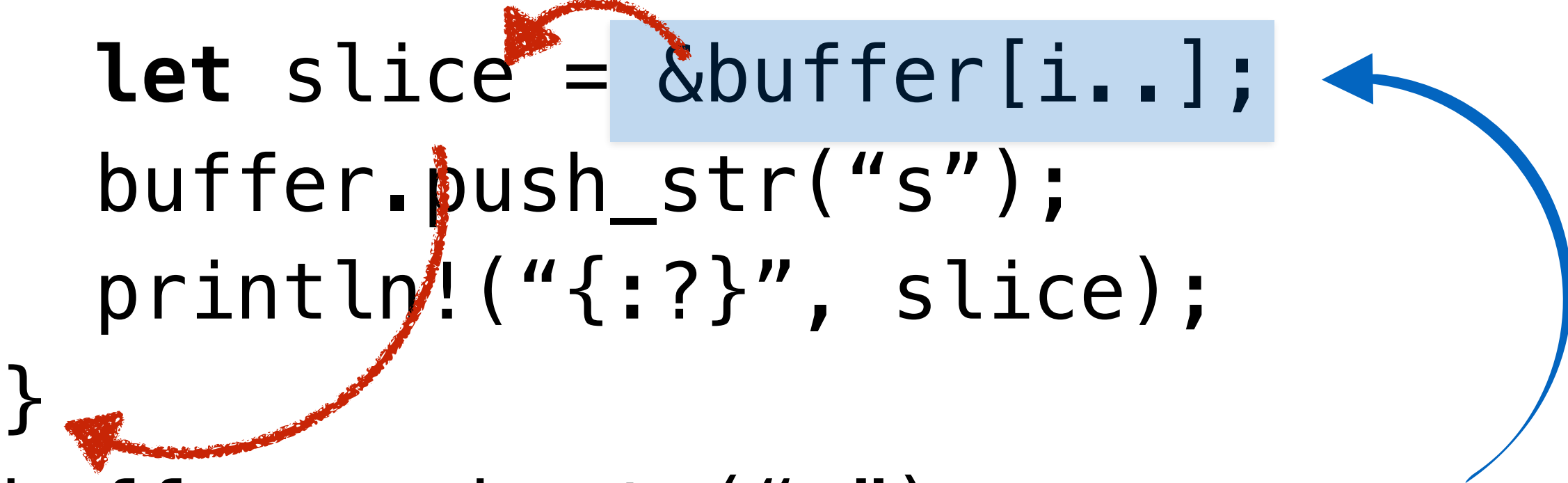
```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    for i in 0 .. buffer.len() {  
        let slice = &buffer[i..];  
        buffer.push_str("s");  
        println!("{:?}", slice);  
    }  
    buffer.push_str("s");  
}
```

```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    for i in 0 .. buffer.len() {  
        let slice = &buffer[i..];  
        buffer.push_str("s");  
        println!("{:?}", slice);  
    }  
    buffer.push_str("s");  
}
```



Borrow “locks”
`buffer` until `slice`
goes out of scope


```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    for i in 0 .. buffer.len() {  
        let slice = &buffer[i..];  
        buffer.push_str("s");  
        println!("{:?}", slice);  
    }  
    buffer.push_str("s");  
}
```



The diagram illustrates the borrow relationship between the variable `buffer` and the variable `slice`. A blue arrow originates from the `slice` variable and points to the `buffer` variable, indicating that `slice` borrows from `buffer`. A red arrow originates from the `slice` variable and points to the closing brace of the `for` loop, indicating that the borrow of `buffer` by `slice` ends when the loop finishes.

Borrow “locks”
`buffer` until `slice`
goes out of scope

```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    for i in 0 .. buffer.len() {  
        let slice = &buffer[i..];  
        buffer.push_str("s");  
        println!("{:?}", slice);  
    }  
    buffer.push_str("s");  
}
```

Borrow "locks"
`buffer` until `slice`
goes out of scope

```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    for i in 0 .. buffer.len() {  
        let slice = &buffer[i..];  
        buffer.push_str("s");  
        println!("{:?}", slice);  
    }  
    buffer.push_str("s");  
}
```

Borrow “locks”
`buffer` until `slice`
goes out of scope

```
fn main() {  
    let mut buffer: String = format!("Rustacean");  
    for i in 0 .. buffer.len() {  
        let slice = &buffer[i..];  
        buffer.push_str("s");  
        println!("{:?}", slice);  
    }  
    buffer.push_str("s");  
}
```

Borrow “locks”
`buffer` until `slice`
goes out of scope

OK: `buffer` is not borrowed here

Exercise: **mutable borrow**

<http://rust-tutorials.com/RustConf17>

Cheat sheet:

```
&String          // type of shared reference  
&mut String      // type of mutable reference  
&str             // type of string slice
```

```
fn greet(name: &String) {...}  
fn adjust(name: &mut String) {...}
```

```
&name            // shared borrow  
&mut name        // mutable borrow  
&name[x..y]      // slice expression
```

<http://doc.rust-lang.org/std>