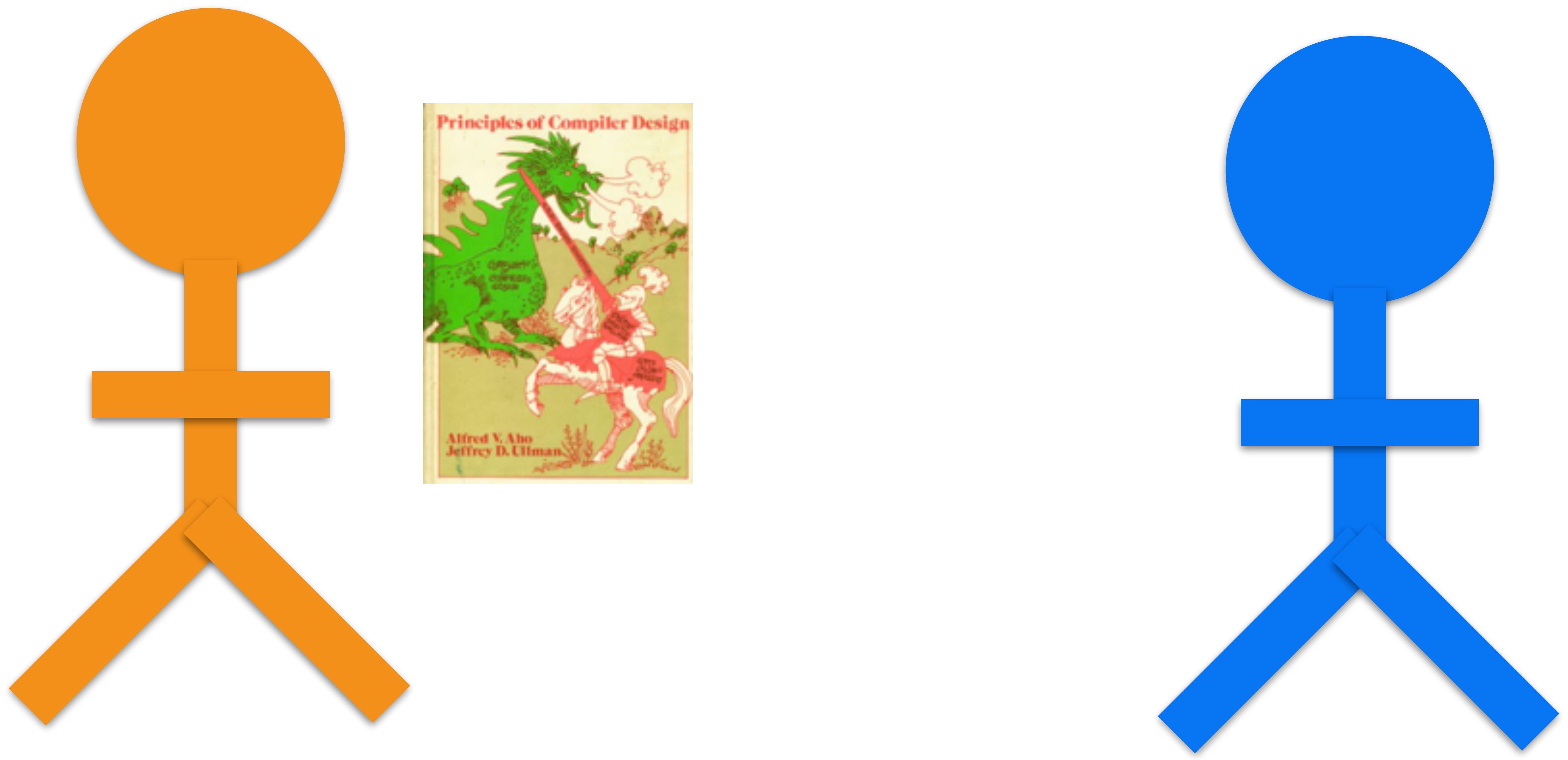
Shared Borrows

Borrowing: Shared Borrows

Borrowing: Shared Borrows



Borrowing: Shared Borrows

```rust
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```rust
fn helper(name: &String) {
  println!(..);
}
```

Shared borrow

```rust
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```rust
fn helper(name: &String) {
  println!(..);
}
```

Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```
fn helper(name: &String) {
  println!(..);
}
```

Lend the string,
creating a reference

Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```
fn helper(name: &String) {
  println!(..);
}
```

Lend the string, creating a reference

Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```
fn helper(name: &String) {
  println!(..);
}
```

Lend the string, creating a reference

Change type to a **reference** to a String

Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```
fn helper(name: &String) {
  println!(..);
}
```

Lend the string, creating a reference

Change type to a **reference** to a String

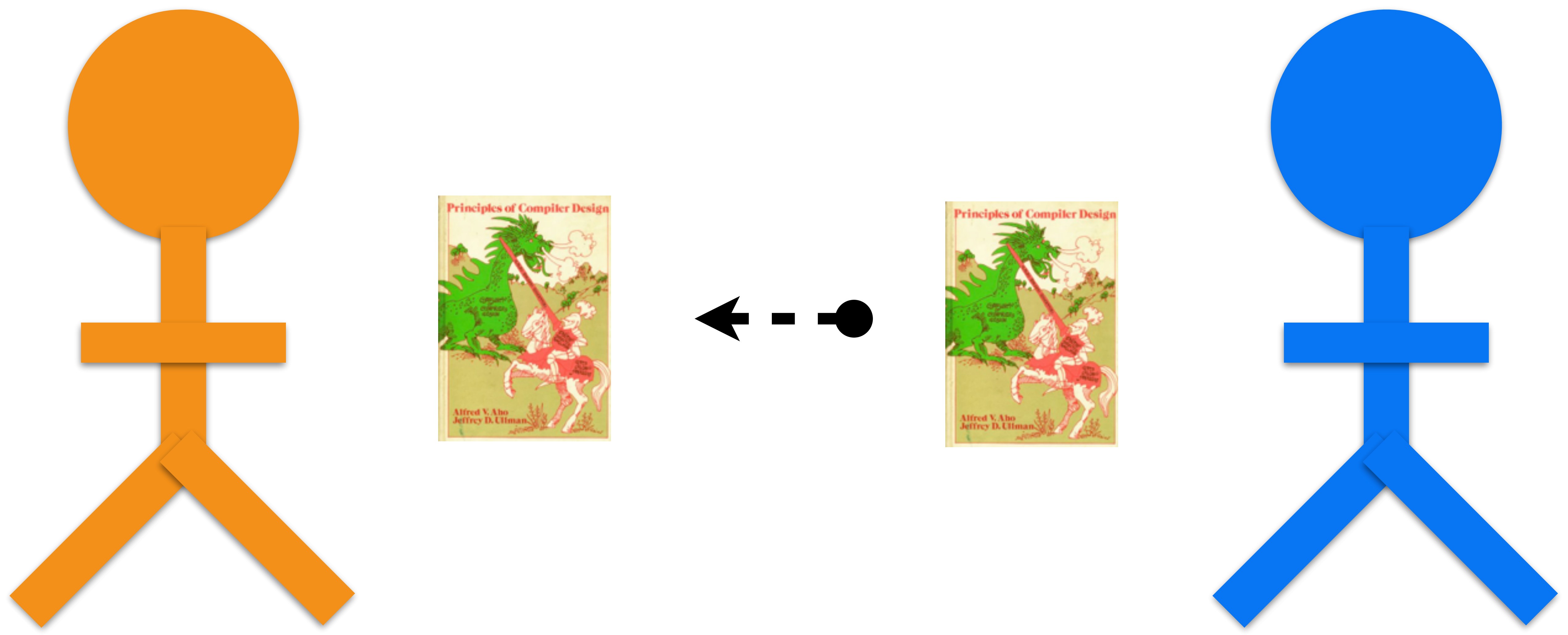
Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```
fn helper(name: &String) {
  println!(..);
}
```

Lend the string, creating a reference

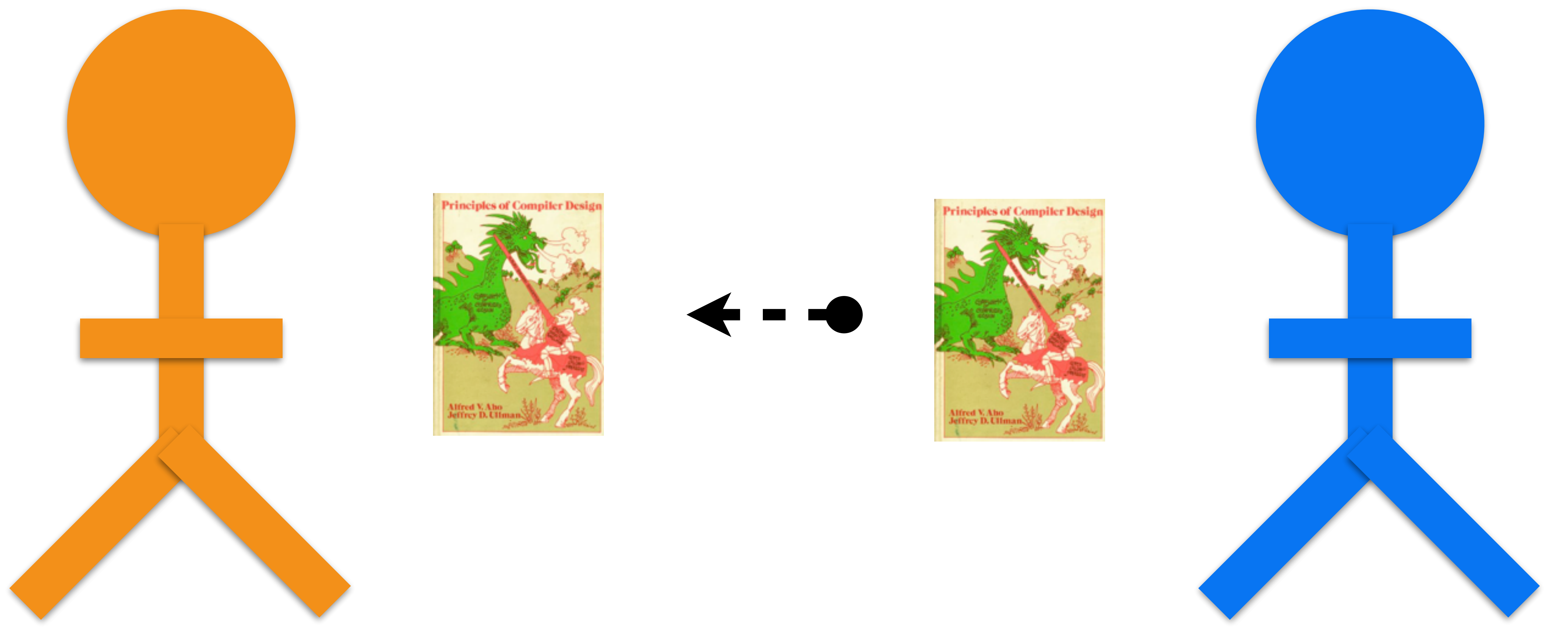
Change type to a **reference** to a String



Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

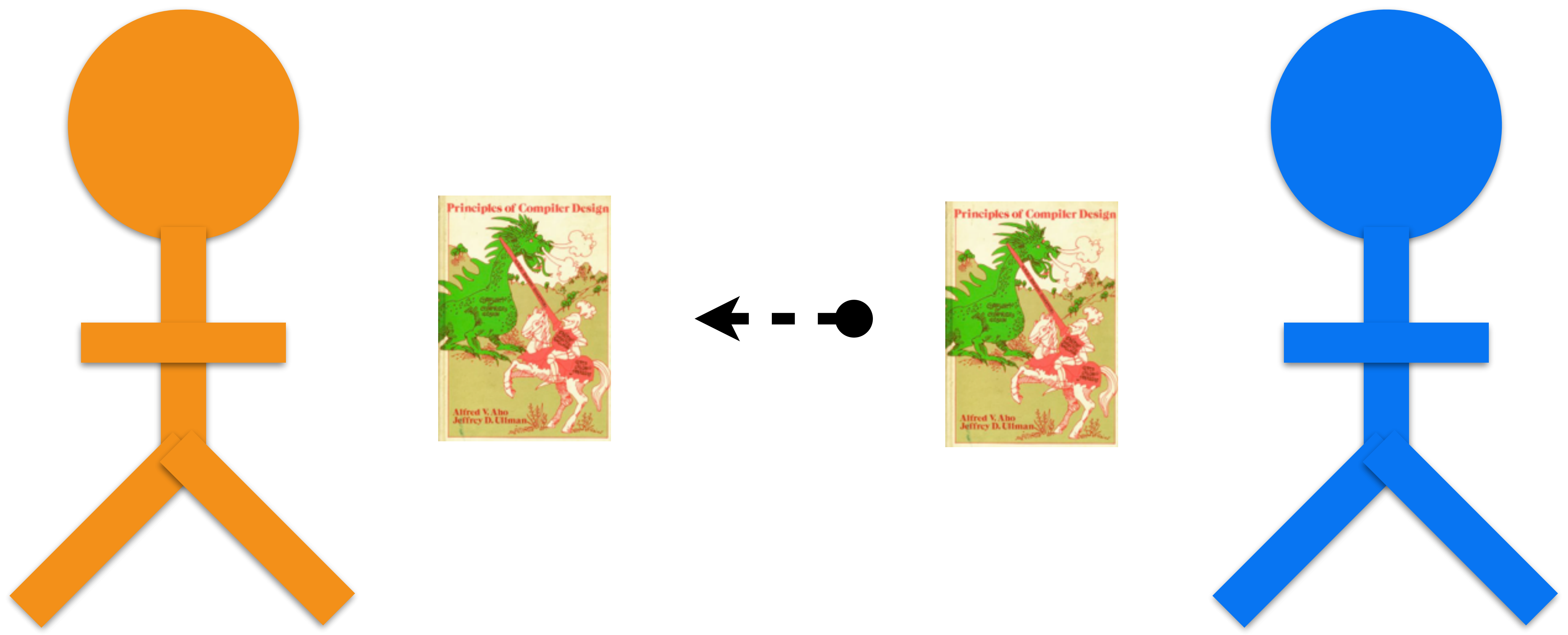
```
fn helper(name: &String) {
  println!(..);
}
```



Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

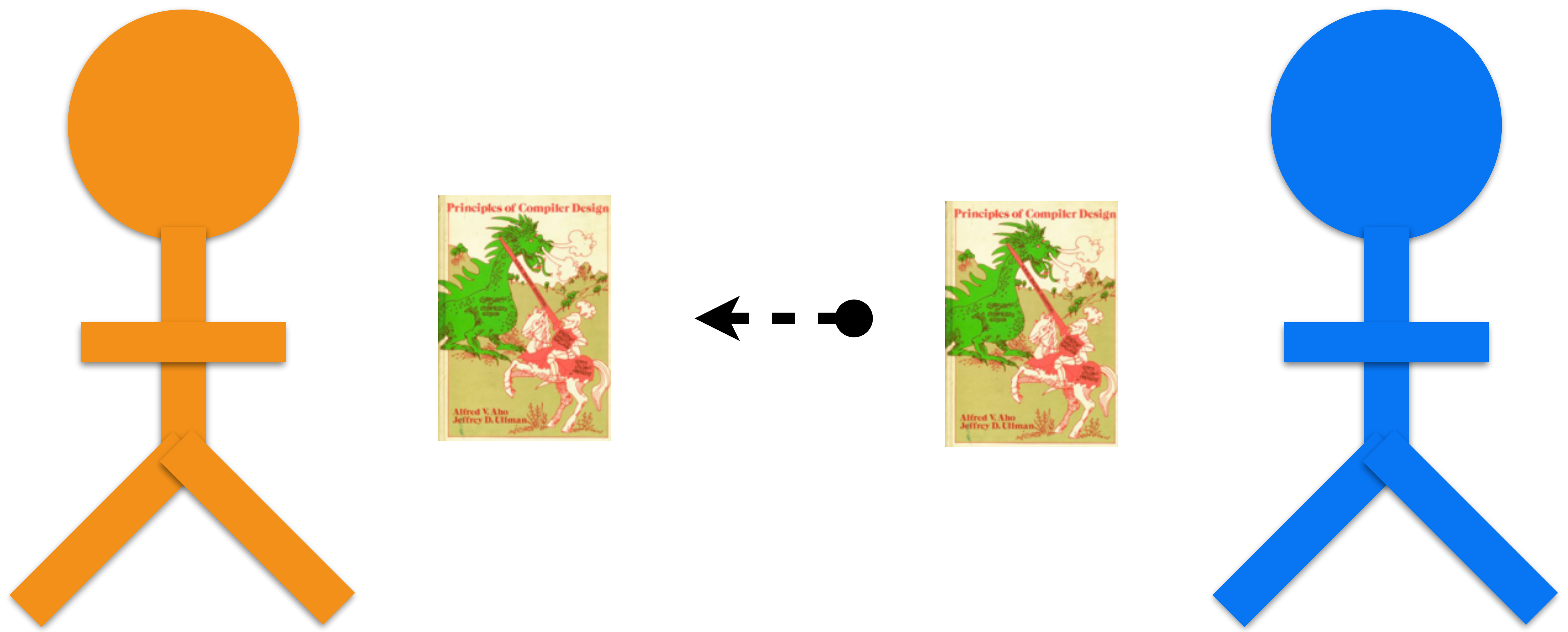
```
fn helper(name: &String) {
  println!(..);
}
```



Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```
fn helper(name: &String) {
  println!(..);
}
```



Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```
fn helper(name: &String) {
  println!(..);
}
```

Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```
fn helper(name: &String) {
  println!(..);
}
```

Shared borrow

```rust
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```rust
fn helper(name: &String) {
  println!(..);
}
```

Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```
fn helper(name: &String) {
  println!(..);
}
```

Shared borrow

```
fn main() {
  let name = format!("…");
  let reference = &name;
  helper(reference);
  helper(reference);
}
```

```
fn helper(name: &String) {
  println!(..);
}
```

Shared borrow

Shared == Immutable

```
fn helper(name: &String) {
  println!(“{}”, name);
}

fn helper(name: &String) {
  name.push(‘x’);
}
```

Shared == Immutable

```
fn helper(name: &String) {
  println!("{}", name);
}
```

OK. Just reads.

```
fn helper(name: &String) {
  name.push('x');
}
```

Shared == Immutable

```
fn helper(name: &String) {
  println!("{}", name);
}
```

OK. Just reads.

```
fn helper(name: &String) {
  name.push('x');
}
```

Error. Writes.

Shared == Immutable

```
fn helper(name: &String) {
  println!("{}", name);
}
```

OK. Just reads.

```
fn helper(name: &String) {
  ~~name.push('x');~~
}
```

Error. Writes.

```
error: cannot borrow immutable borrowed content `*name`
       as mutable
    name.push_str("s");
    ^^^^
```

Shared == Immutable*

```
fn helper(name: &String) {
  println!("{}", name);
}
```

OK. Just reads.

```
fn helper(name: &String) {
  ~~name.push('x');~~
}
```

Error. Writes.

```
error: cannot borrow immutable borrowed content `*name`
       as mutable
    name.push_str("s");
    ^^^^
```

* **Actually:** mutation only in **controlled circumstances**.

Play time

Waterloo, Cassius Coolidge, c. 1906

http://is.gd/no0tTH

```
fn main() {
  let name = format!(“…”);
  helper(&name[1..]);
  helper(&name);
}
```

```
fn helper(name: &str) {
  println!(..);
}
```

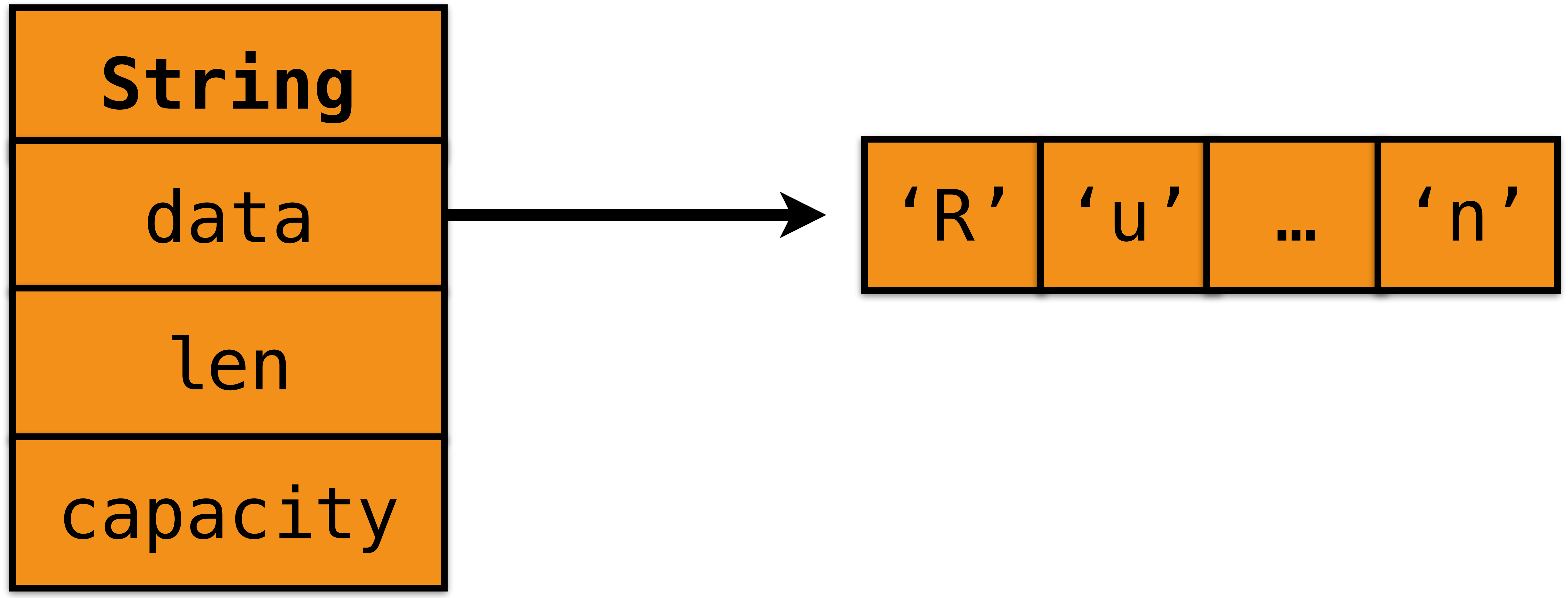
Looks like other languages:

- Python: name[1:]
- Ruby: name[1..-1]

But no copying at runtime.

```
fn main() {
  let name = format!("…");
  helper(&name[1..]);
  helper(&name);
}
```

```
fn helper(name: &str) {
  println!(..);
}
```



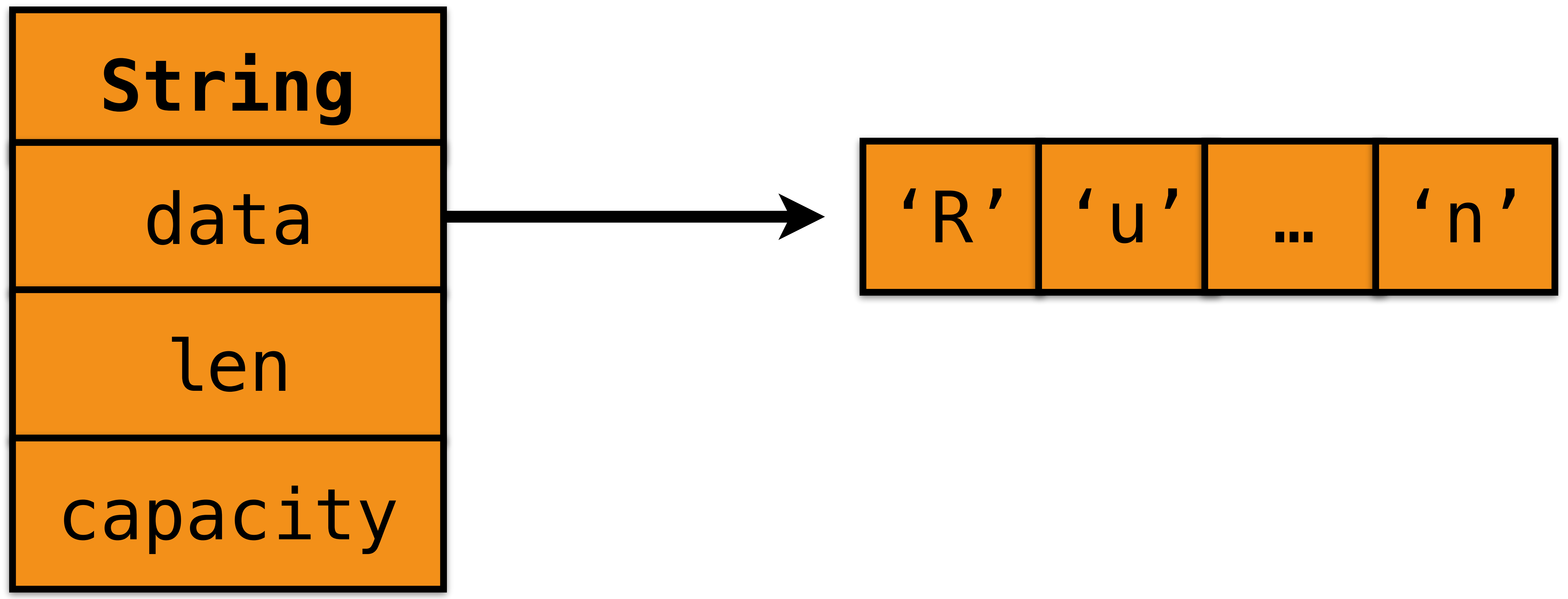
Looks like other languages:

- Python: name[1:]
- Ruby: name[1..-1]

But no copying at runtime.

```
fn main() {
  let name = format!("…");
  helper(&name[1..]);
  helper(&name);
}
```

```
fn helper(name: &str) {
  println!(..);
}
```



Looks like other languages:

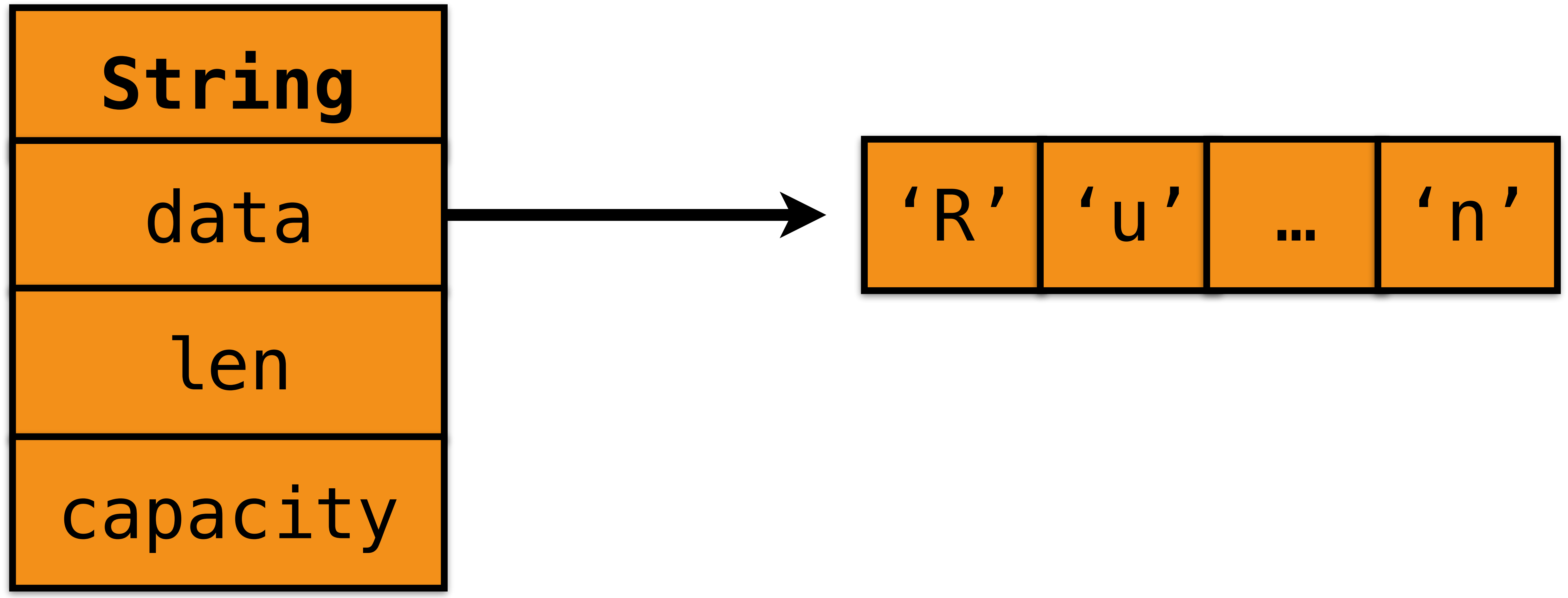
- Python: name[1:]
- Ruby: name[1..-1]

But no copying at runtime.

```
fn main() {
  let name = format!("…");
  helper(&name[1..]);
  helper(&name);
}
```

```
fn helper(name: &str) {
  println!(..);
}
```

Change type from `&String` to a **string slice**, `&str`



Looks like other languages:

- Python: name[1:]
- Ruby: name[1..-1]

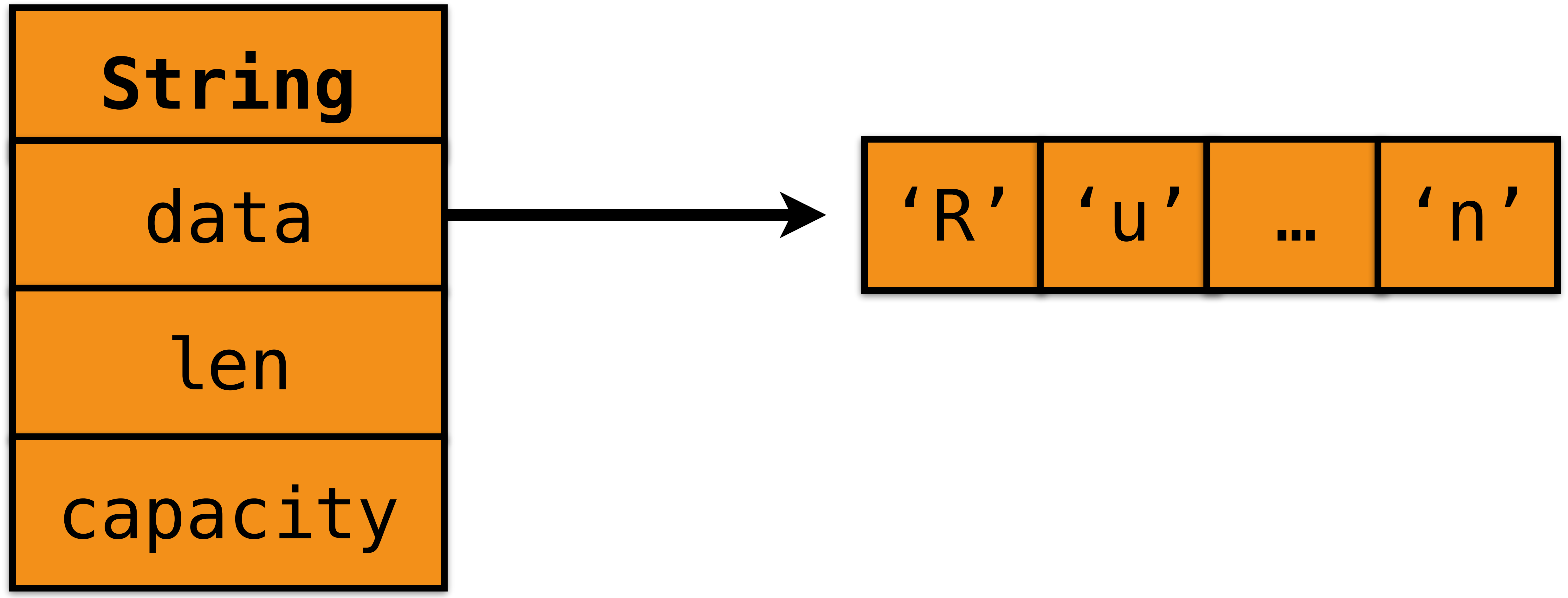
But no copying at runtime.

```
fn main() {
  let name = format!("…");
  helper(&name[1..]);
  helper(&name);
}
```

Lend some of the string

```
fn helper(name: &str) {
  println!(..);
}
```

Change type from `&String` to a **string slice**, `&str`



Looks like other languages:

- Python: name[1:]
- Ruby: name[1..-1]

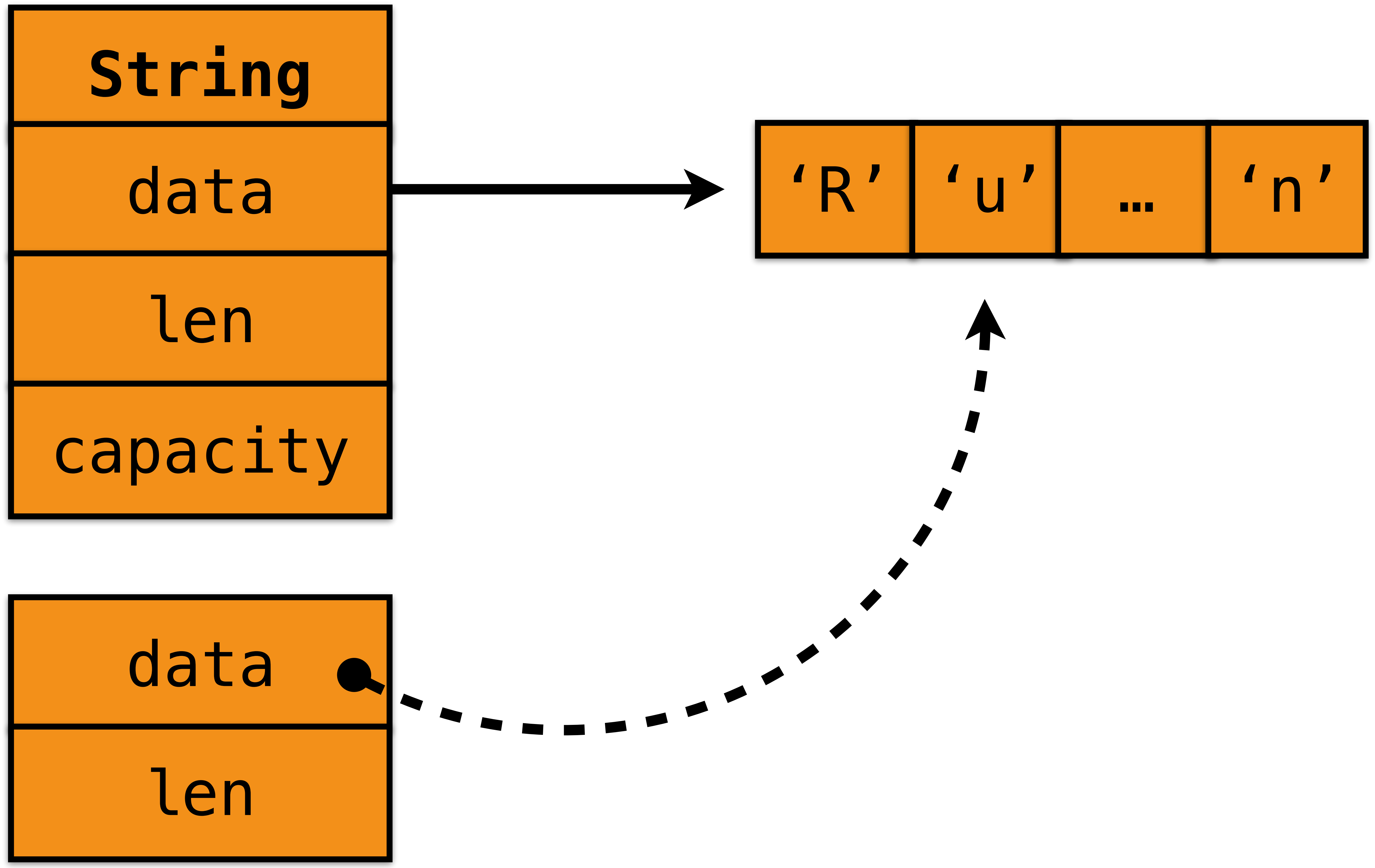
But no copying at runtime.

```
fn main() {
  let name = format!("…");
  helper(&name[1..]);
  helper(&name);
}
```

Lend some of the string

```
fn helper(name: &str) {
  println!(..);
}
```

Change type from `&String` to a **string slice**, `&str`



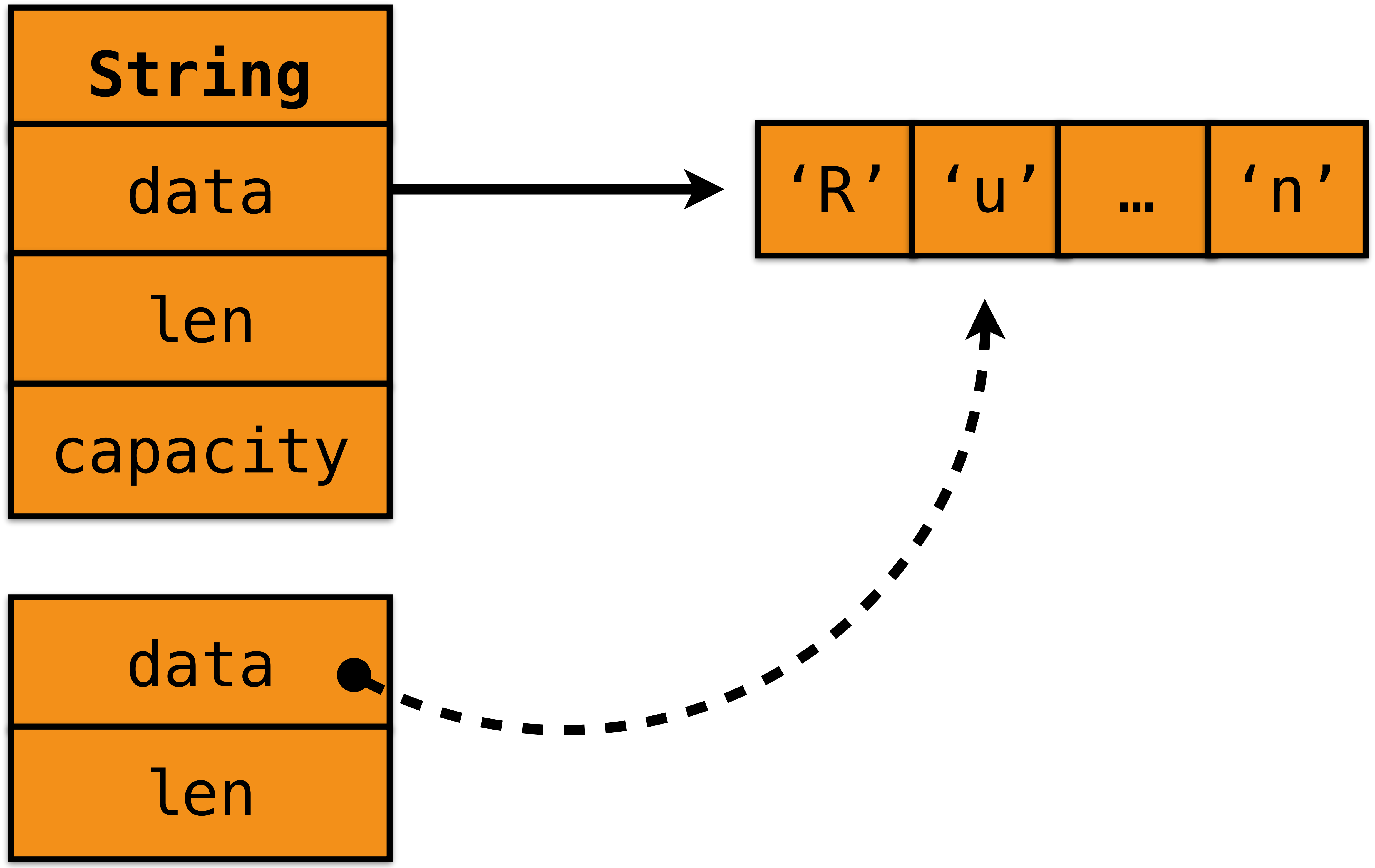
Looks like other languages:

- Python: name[1:]
- Ruby: name[1..-1]

But no copying at runtime.

```
fn main() {
  let name = format!("…");
  helper(&name[1..]);
  helper(&name);
}
```

```
fn helper(name: &str) {
  println!(..);
}
```

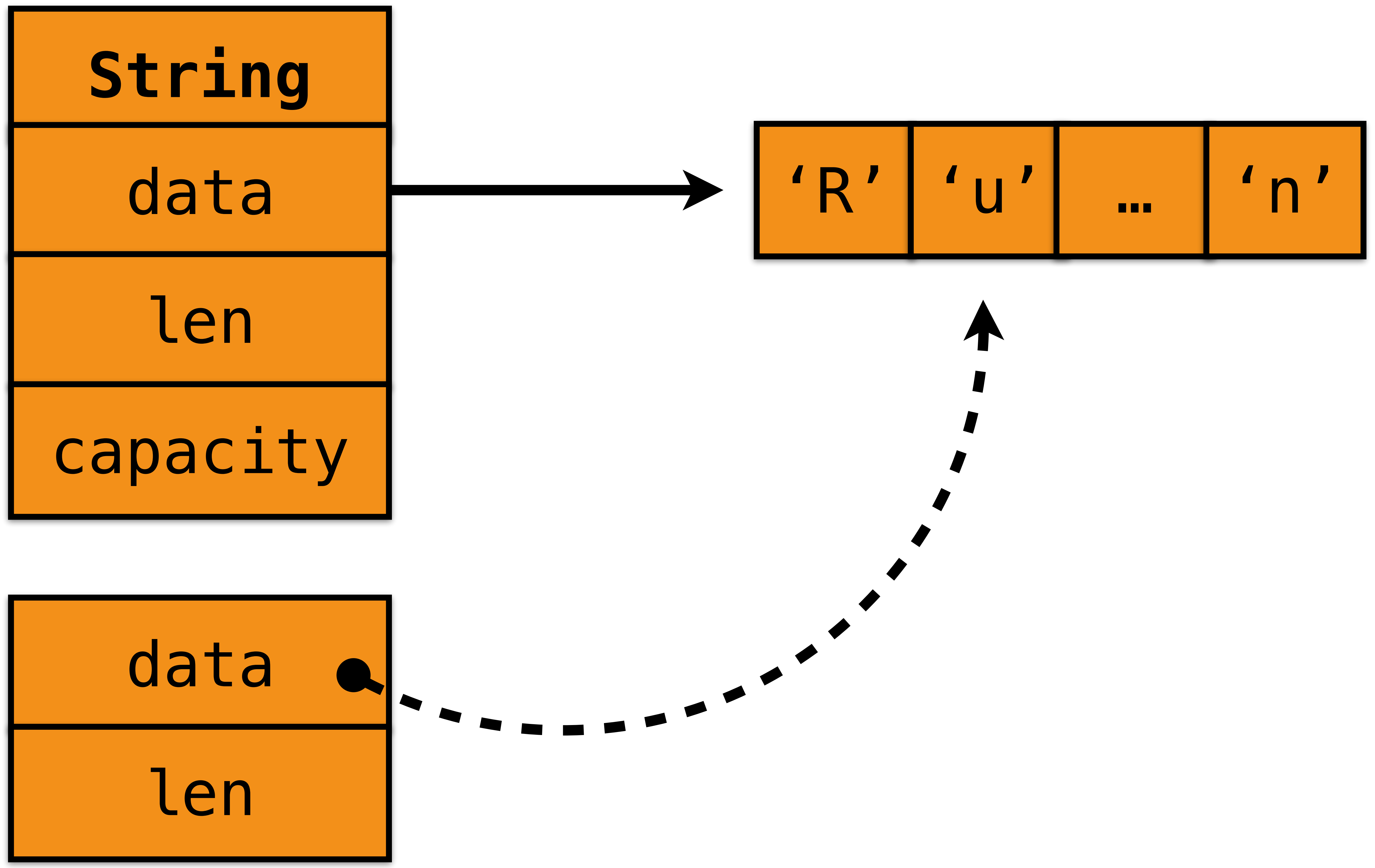


Looks like other languages:
- Python: name[1:]
- Ruby: name[1..-1]

But no copying at runtime.

```
fn main() {
  let name = format!("…");
  helper(&name[1..]);
  helper(&name);
}
```

```
fn helper(name: &str) {
  println!(..);
}
```



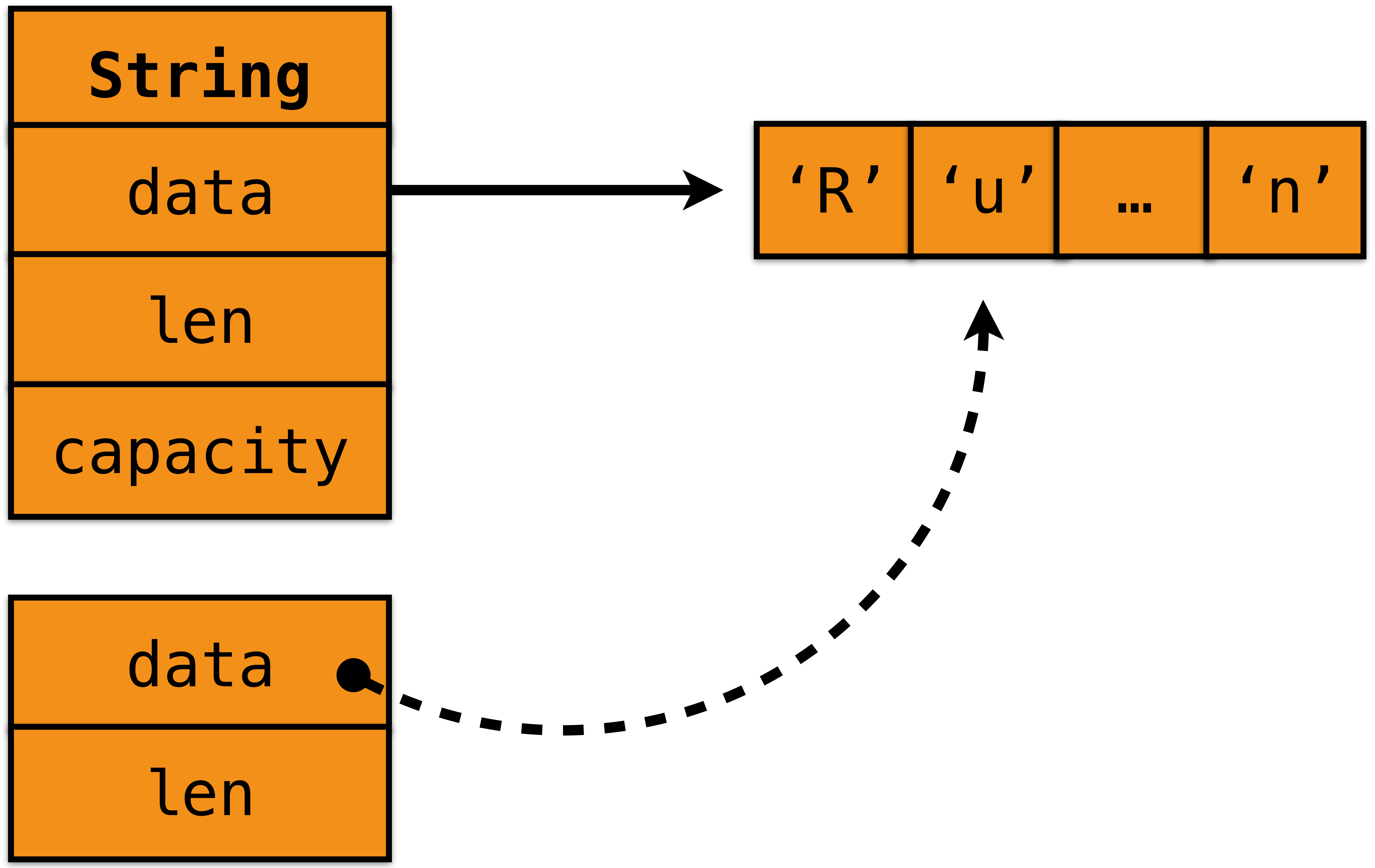
Looks like other languages:

- Python: name[1:]
- Ruby: name[1..-1]

But no copying at runtime.

```
fn main() {
  let name = format!("…");
  helper(&name[1..]);
  helper(&name);
}
```

```
fn helper(name: &str) {
  println!(..);
}
```



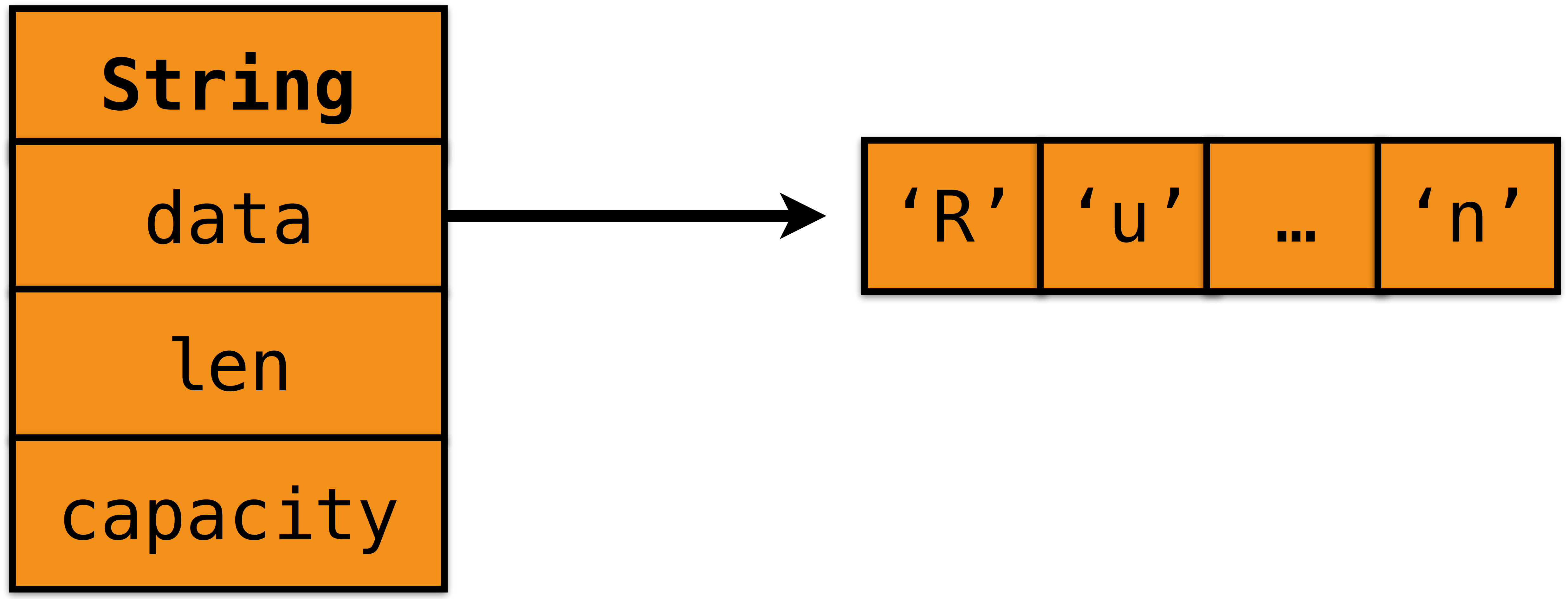
Looks like other languages:

- Python: name[1:]
- Ruby: name[1..-1]

But no copying at runtime.

```
fn main() {
  let name = format!("…");
  helper(&name[1..]);
  helper(&name);
}
```

```
fn helper(name: &str) {
  println!(..);
}
```



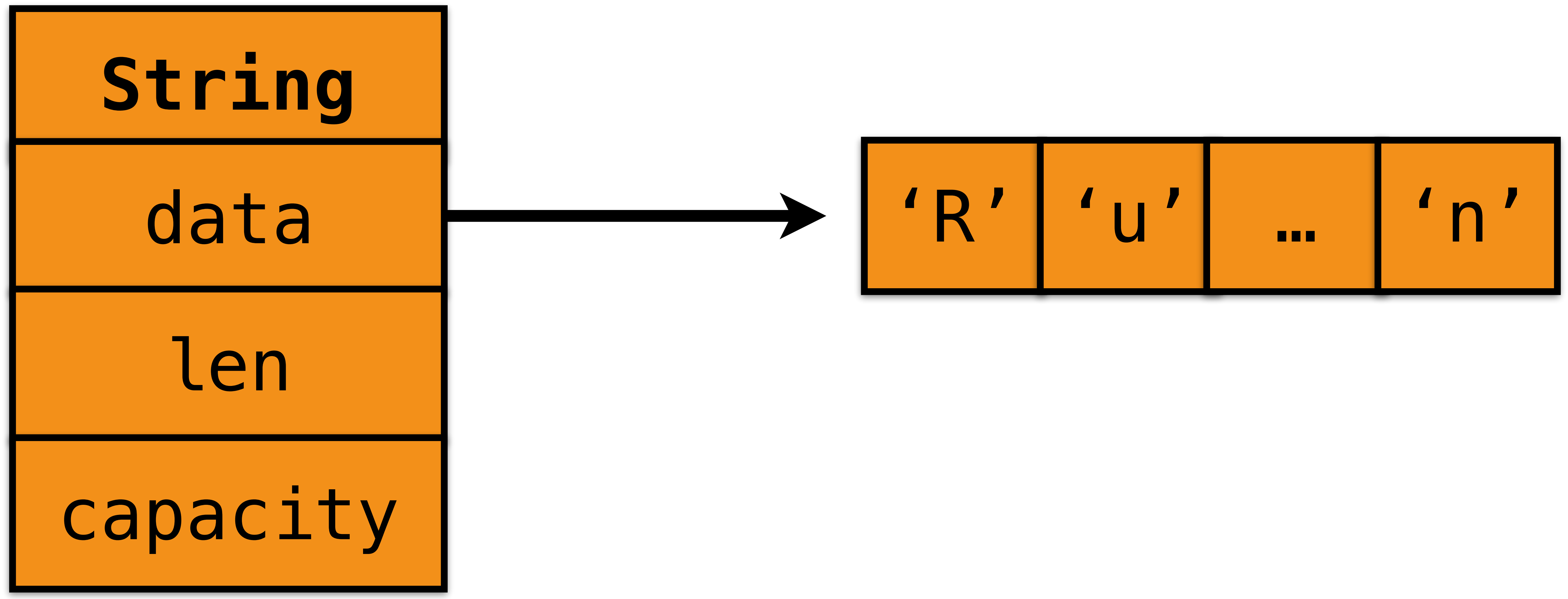
Looks like other languages:

- Python: name[1:]
- Ruby: name[1..-1]

But no copying at runtime.

```
fn main() {
  let name = format!("…");
  helper(&name[1..]);
  helper(&name);
}
```

```
fn helper(name: &str) {
  println!(..);
}
```



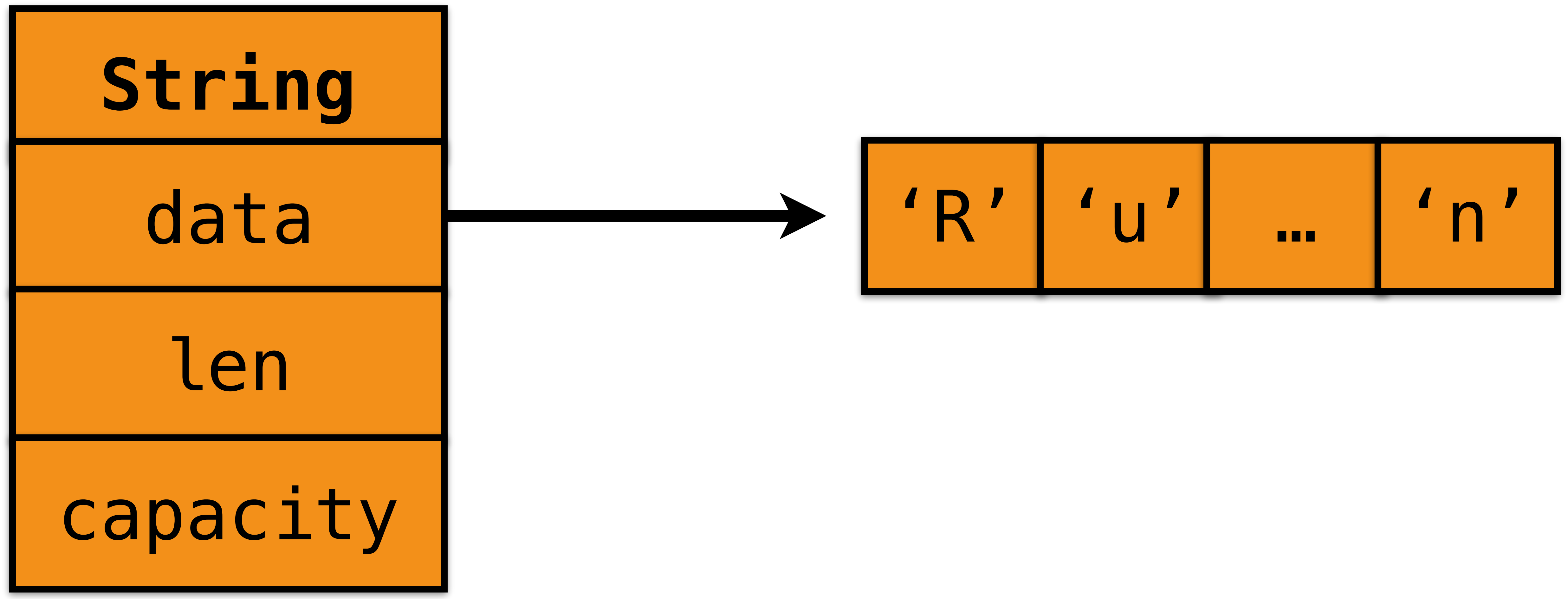
Looks like other languages:

- Python: name[1:]
- Ruby: name[1..-1]

But no copying at runtime.

```
fn main() {
  let name = format!("…");
  helper(&name[1..]);
  helper(&name);
}
```

```
fn helper(name: &str) {
  println!(..);
}
```



Looks like other languages:

- Python: name[1:]
- Ruby: name[1..-1]

But no copying at runtime.

High-level code, low-level efficiency

```
for word in line.split(' ') {
    sum += word.len();
}
```

No copying, no allocations.

High-level code, low-level efficiency

```
for word in line.split(' ') {
    sum += word.len();
}
```

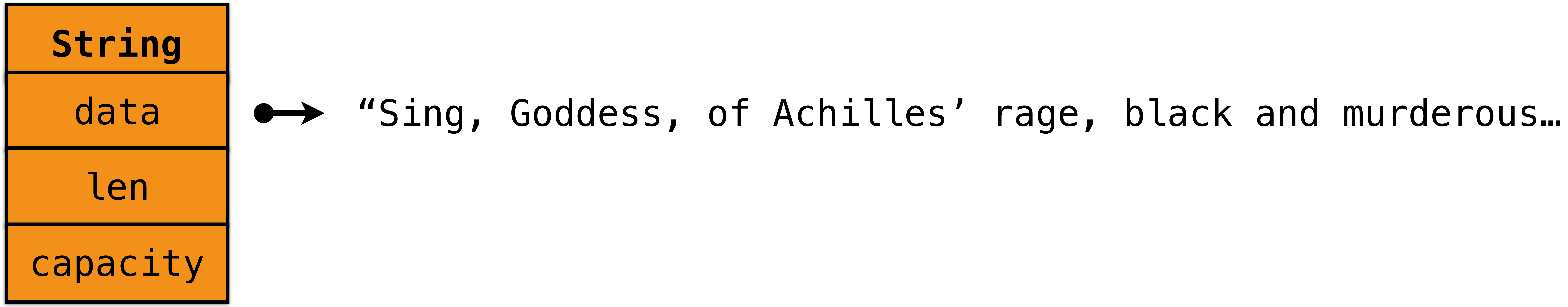
Iterator over slices borrowed from **line**.

No copying, no allocations.

High-level code, low-level efficiency

```
for word in line.split(' ') {
    sum += word.len();
}
```

Iterator over slices borrowed from **line**.

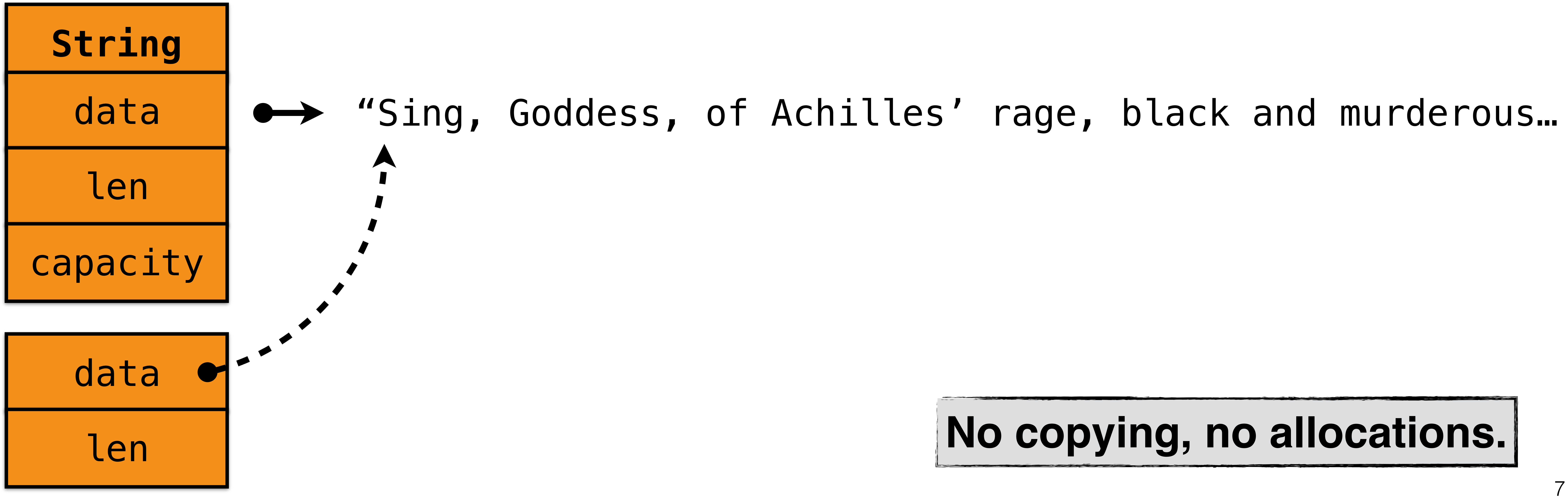


No copying, no allocations.

High-level code, low-level efficiency

```
for word in line.split(' ') {
    sum += word.len();
}
```

Iterator over slices borrowed from **line**.

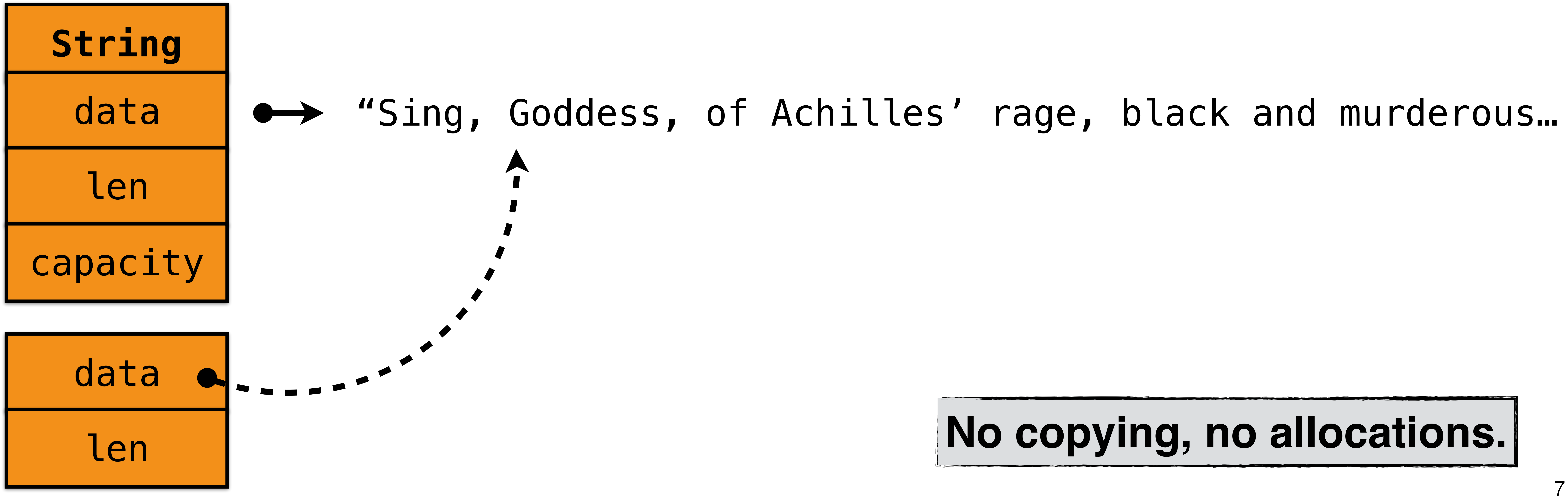


No copying, no allocations.

High-level code, low-level efficiency

```
for word in line.split(' ') {
    sum += word.len();
}
```

Iterator over slices borrowed from **line**.

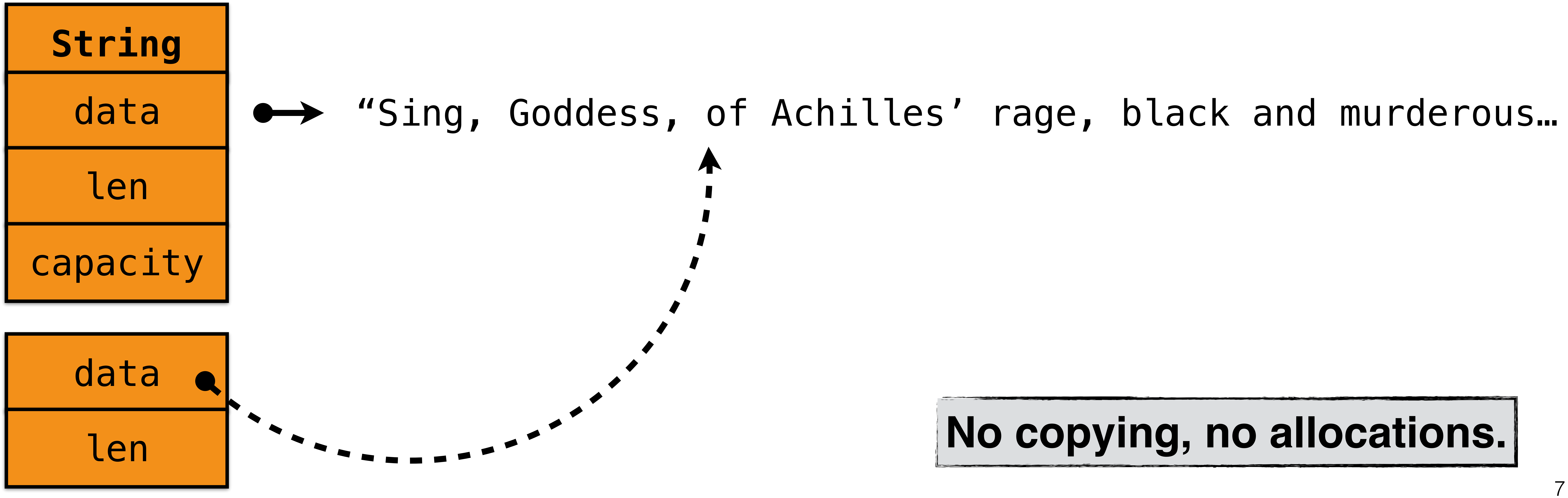


No copying, no allocations.

High-level code, low-level efficiency

```
for word in line.split(' ') {
    sum += word.len();
}
```

Iterator over slices borrowed from **line**.



No copying, no allocations.

Exercise: **shared borrow**

http://rust-tutorials.com/RustConf17

Cheat sheet:

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

fn greet(name: &String) {..}

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

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