Declarations with *let* in *for* loops

Level 1 – Section 2

Problem With var in for Loops

var is the reason behind a popular “gotcha” in *for* loops.

```
function loadProfiles(userNames){
//....

  for(var i in userNames){
    _fetchProfile("/users/" + userNames[i], function(){
      console.log("Fetched for ", userNames[i]);
    }
  }
}
```

Accessing `i` from inside the callback

```
loadProfiles(["Sam", "Tyler", "Brook", "Alex"]);
```

Unexpectedly outputs the same value on all iterations

```
> Fetched for Alex
> Fetched for Alex
> Fetched for Alex
> Fetched for Alex
```

Understanding Hoisting and for Loops

var i is hoisted to the top of the function and **shared** across each iteration of the loop.

```
function loadProfiles(userNames){
  var i;
  //....

  for(    i in userNames){
    _fetchProfile("/users/" + userNames[i], function(){
      console.log("Fetched for ", userNames[i]);
    }
  }
}
```

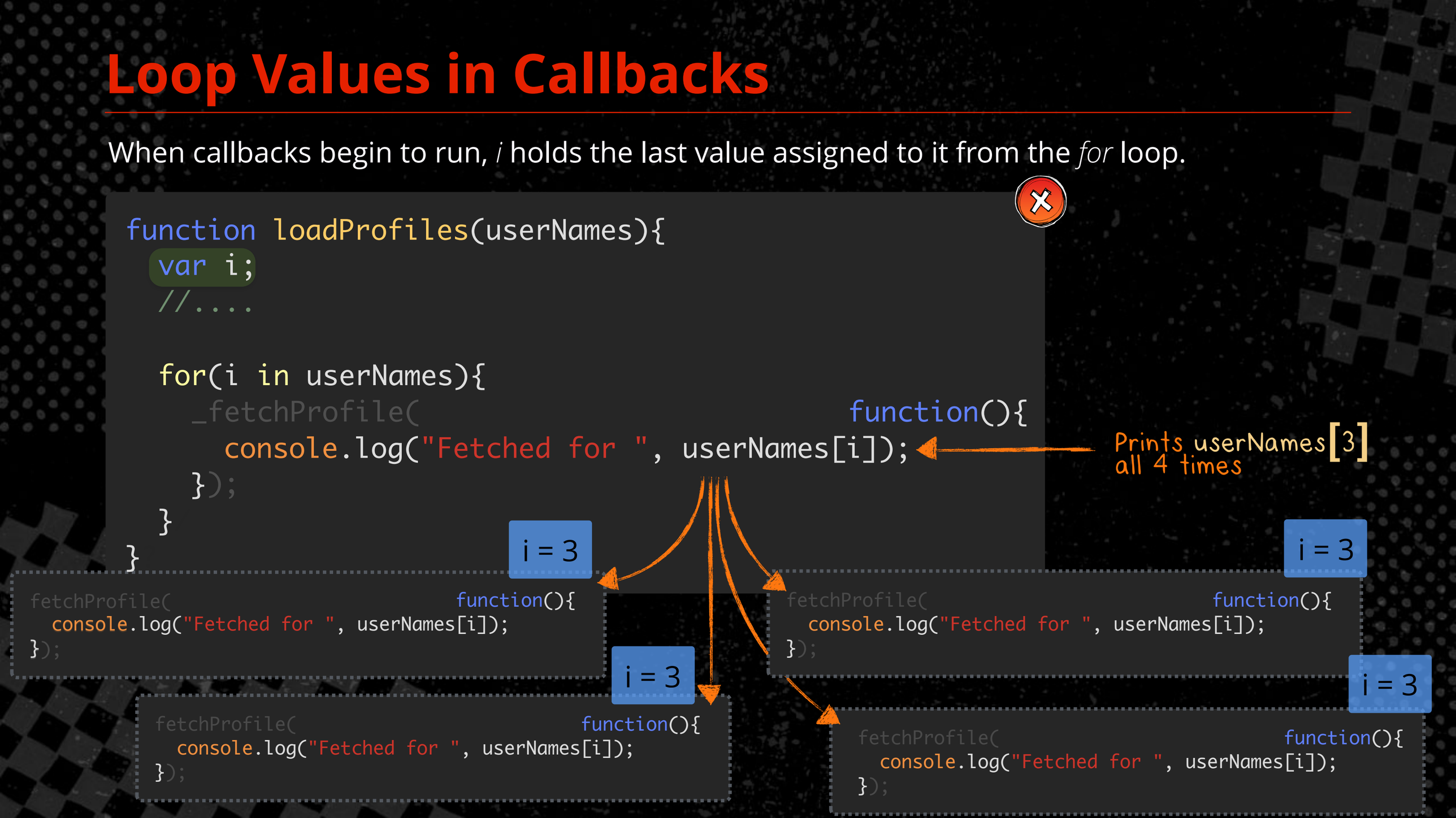
fetchProfile is called 4 times,
before any of the callbacks are invoked

i = 0
i = 1
i = 2
i = 3

i is incremented
on each iteration

Loop Values in Callbacks

When callbacks begin to run, *i* holds the last value assigned to it from the *for* loop.



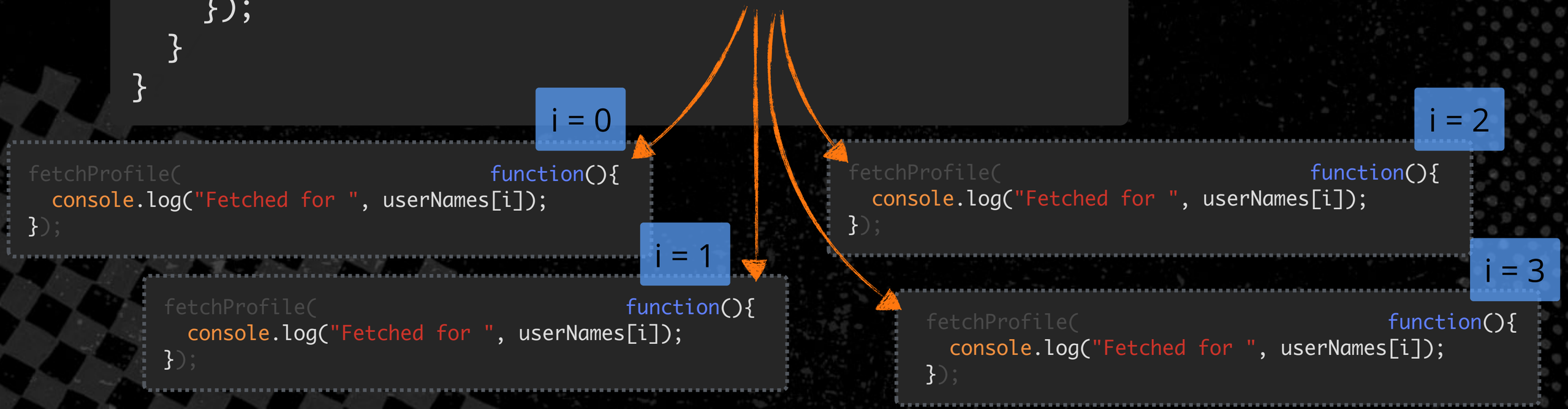
Using let in for Loops

ES2015

With *let,* there's **no sharing** in *for* loops. A new variable is created on each iteration.

```
function loadProfiles(userNames){
  //....

  for(let i in userNames){
    _fetchProfile("/users/" + userNames[i], function(){
      console.log("Fetched for ", userNames[i]);
    });
  }
}
```



Using let in for Loops

Each callback function now holds a reference to their **own version** of *i*.

```
function loadProfiles(userNames){
  //....

  for(let i in userNames){
    _fetchProfile("/users/" + userNames[i], function(){
      console.log("Fetched for ", userNames[i]);
    });
  }
}
```

```
loadProfiles(["Sam", "Tyler", "Brook", "Alex"]);
```

Outputs the correct value for each iteration

```
> Fetched for Sam
> Fetched for Tyler
> Fetched for Brook
> Fetched for Alex
```

let Cannot Be Redeclared

Variables declared with *let* can be reassigned, but cannot be **redeclared** within the same scope.

```
let flashMessage = "Hello";
flashMessage = "Goodbye";
```

Reassigning is allowed

```
let flashMessage = "Hello";
let flashMessage = "Goodbye";
```

Redeclaring is not allowed

```
> TypeError: Identifier 'flashMessage' has already been declared
```

```
let flashMessage = "Hello";

function loadProfiles(userNames){
  let flashMessage = "Loading profiles";
  return flashMessage;
}
```

Different scopes