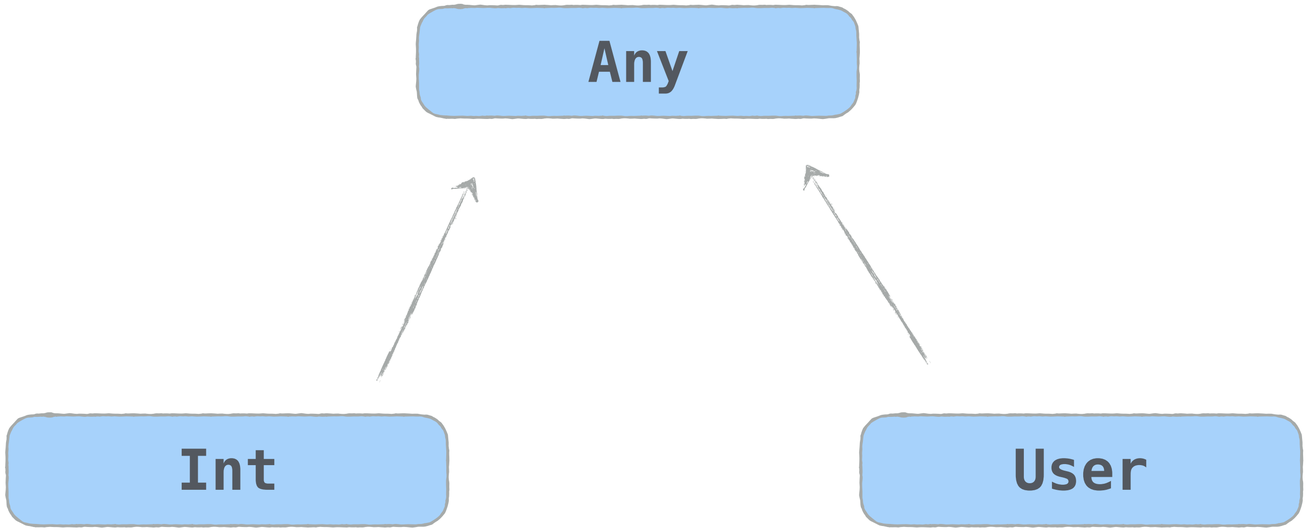
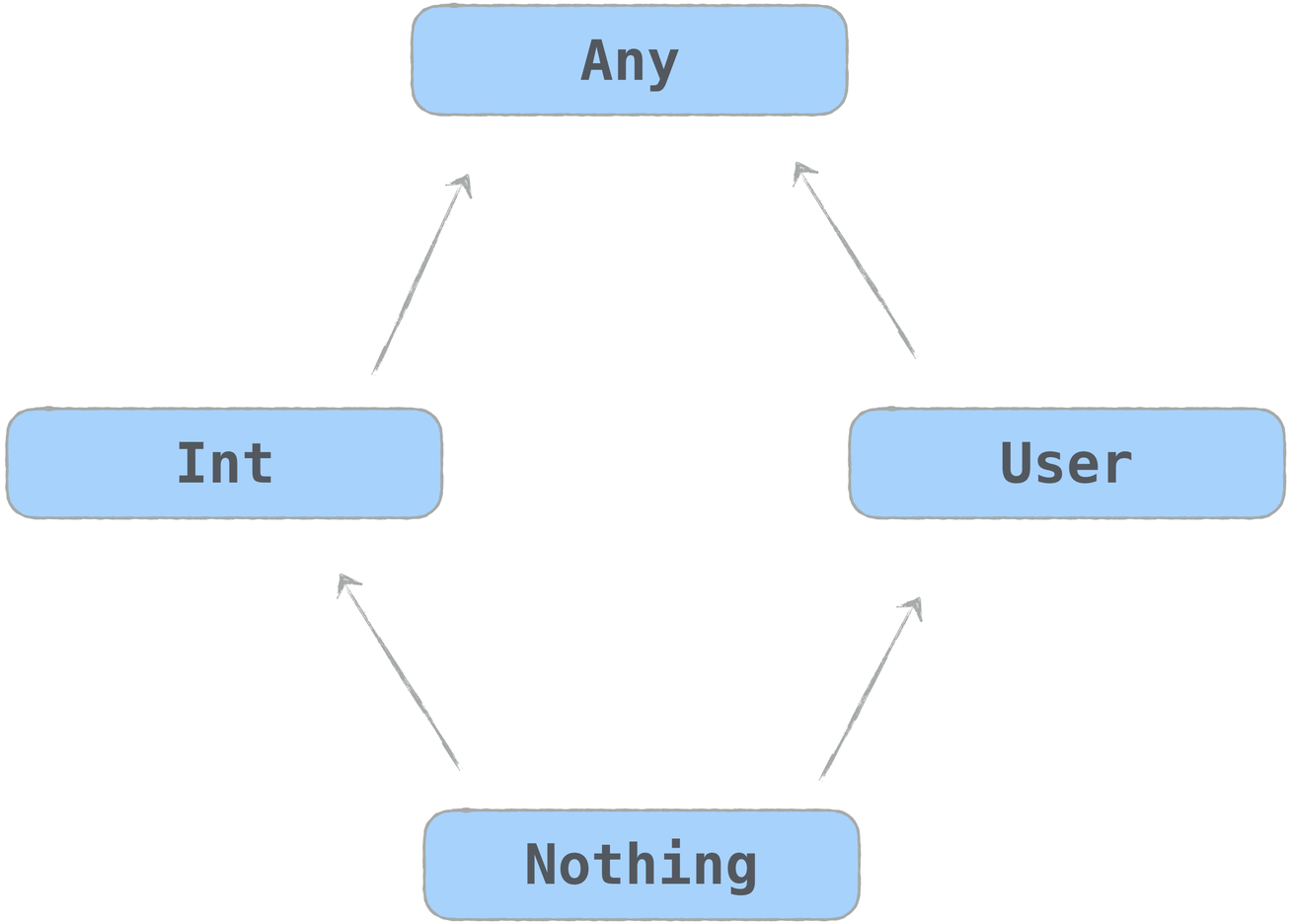
Type hierarchy

Any



Any & Nothing types



Unit vs Nothing vs void ?

`Unit` instead of `void`

No meaningful value is returned

Two equivalent syntactic forms:

```
fun f() { /*...*/ }
```

```
fun f(): Unit { /*...*/ }
```

Unit

Kotlin	Java
Unit	void

`Nothing` is different to `Unit/void`

```
fun fail(message: String): Nothing {
    throw IllegalStateException(message)
}
```

It means “this function never returns”

`Unit`

“the function completes successfully”

“a type that allows only one value and thus can hold no information”

`Nothing`

“the function never completes”

“a type that has no values”

Nothing type

```
val answer: Int = if (timeHasPassed()) {
    42
} else {
    fail("No answer yet")
}
```

Which of the following are expressions of **`Nothing`** type?

1. `throw IllegalStateException()`
2. `Unit`
3. `null`
4. `TODO("Needs to be done")`

Which of the following are expressions of **`Nothing`** type?

1. `throw IllegalStateException()`
2. `Unit`
3. `null`
4. `TODO("Needs to be done")`

Expressions of Nothing type

```
val answer: Int = if (timeHasPassed()) {
    42
} else {
    TODO("Needs to be done")
}
```

```
inline fun TODO(reason: String): Nothing =
    throw NotImplementedError("An operation is not implemented: $reason")
```

Expressions of Nothing type

```
fun greetPerson(person: Person) {
    val name: String = person.name
            ?: throw IllegalArgumentException("Name is unspecified")
    println("Hello, $name!")
}
```

Expressions of Nothing type

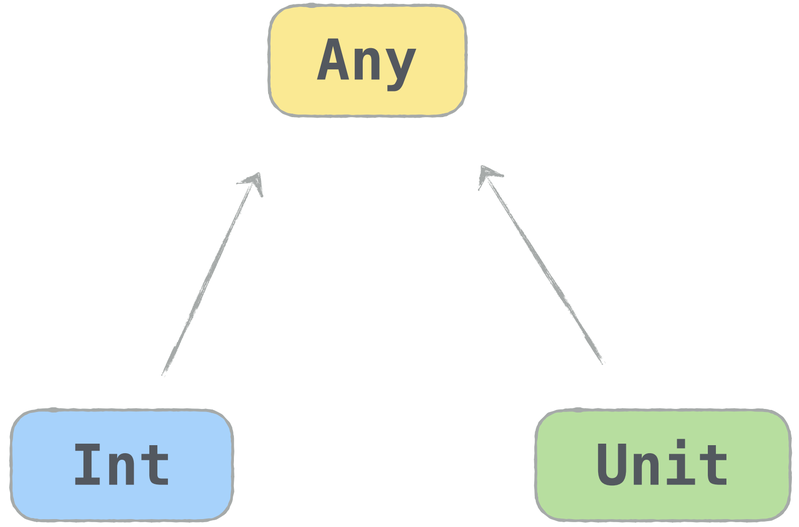
```
fun greetPerson(person: Person) {
    val name = person.name ?: fail("Name is unspecified")
    println("Hello, $name!")
}
```

Expressions of Nothing type

```
fun greetPerson(person: Person) {
    val name = person.name ?: return
    println("Hello, $name!")
}
```

Let's say, `fail` function returns `Unit`

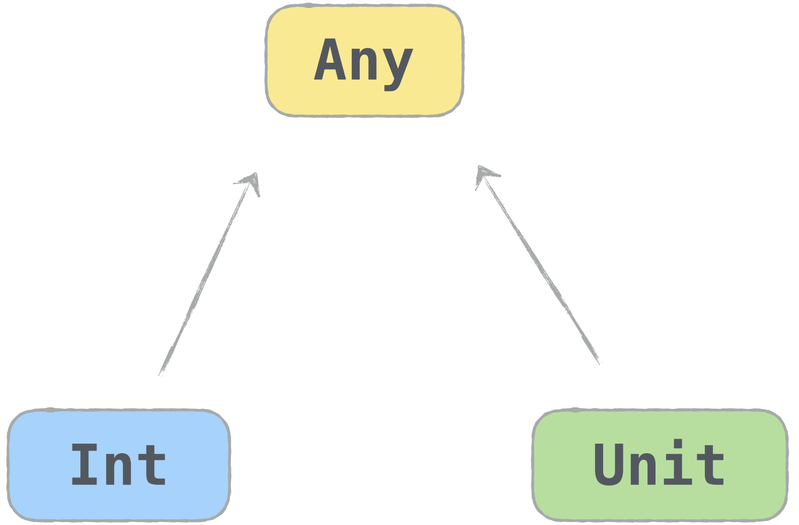
```
val answer = if (timeHasPassed()) {
    42
} else {
    fail("Not ready")
}
```




```
fun fail(message: String) : Unit {
    throw IllegalStateException(message)
}
```

Let's say, `fail` function returns `Unit`

```
val answer: Any = if (timeHasPassed()) {
    42
} else {
    fail("Not ready")
}
```




```
fun fail(message: String) {
    throw IllegalStateException(message)
}
```

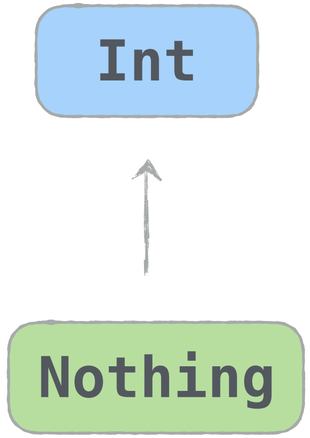
Let's say, `fail` function returns `Unit`

```
val answer = if (timeHasPassed()) {
    42
} else {
    fail("Not ready")
}
```

```
fun fail(message: String): Nothing {
    throw IllegalStateException(message)
}
```

Let's say, `fail` function returns `Unit`

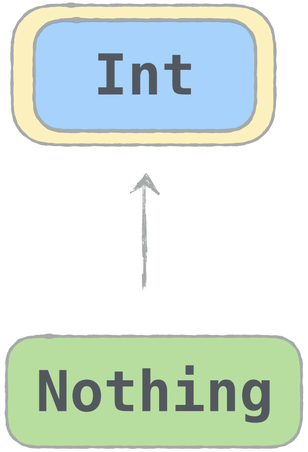
```
val answer = if (timeHasPassed()) {
    42
} else {
    fail("Not ready")
}
```




```
fun fail(message: String): Nothing {
    throw IllegalStateException(message)
}
```

Let's say, `fail` function returns `Unit`

```
val answer: Int = if (timeHasPassed()) {
    42
} else {
    fail("Not ready")
}
```



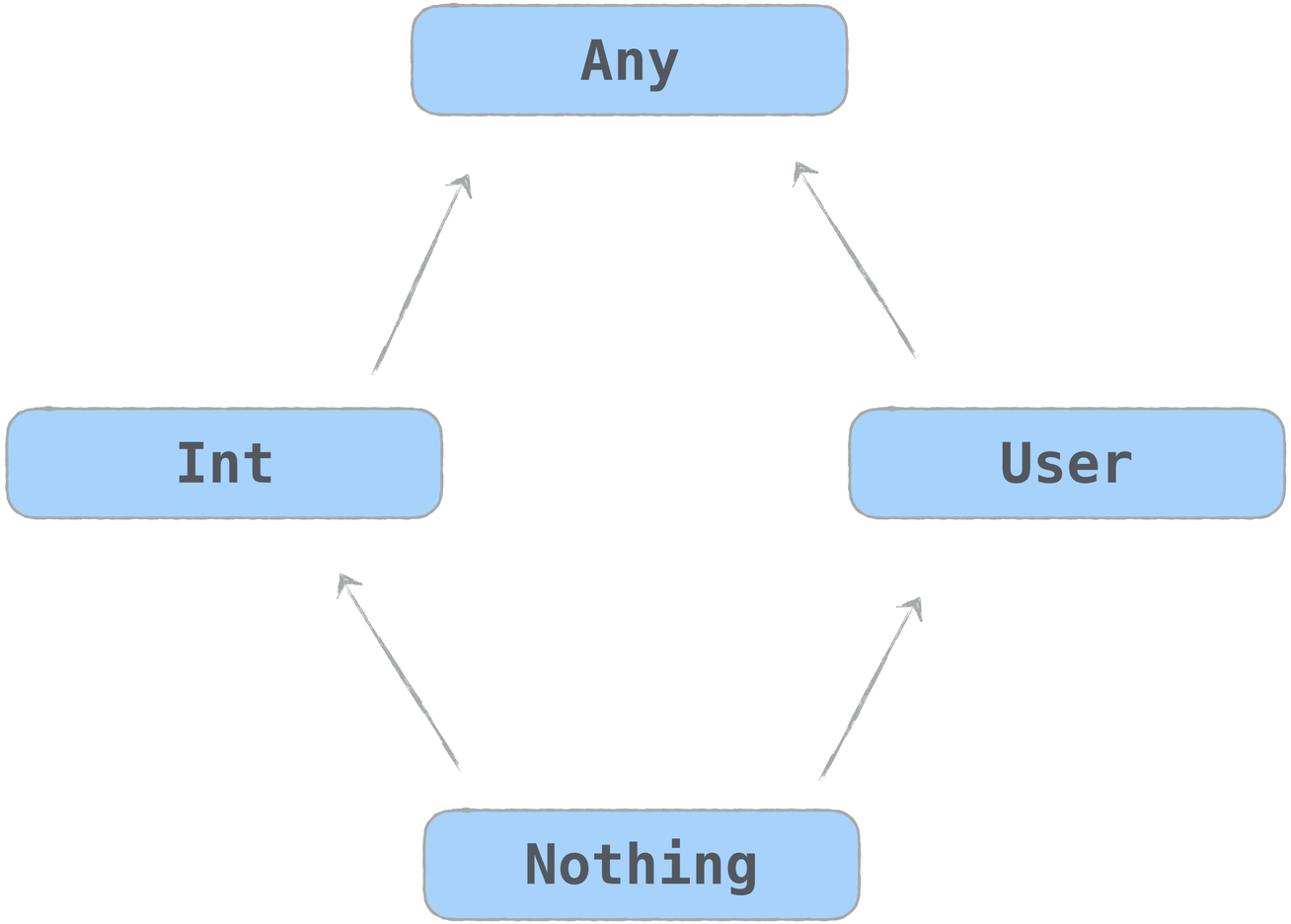

```
fun fail(message: String): Nothing {
    throw IllegalStateException(message)
}
```

Nothing

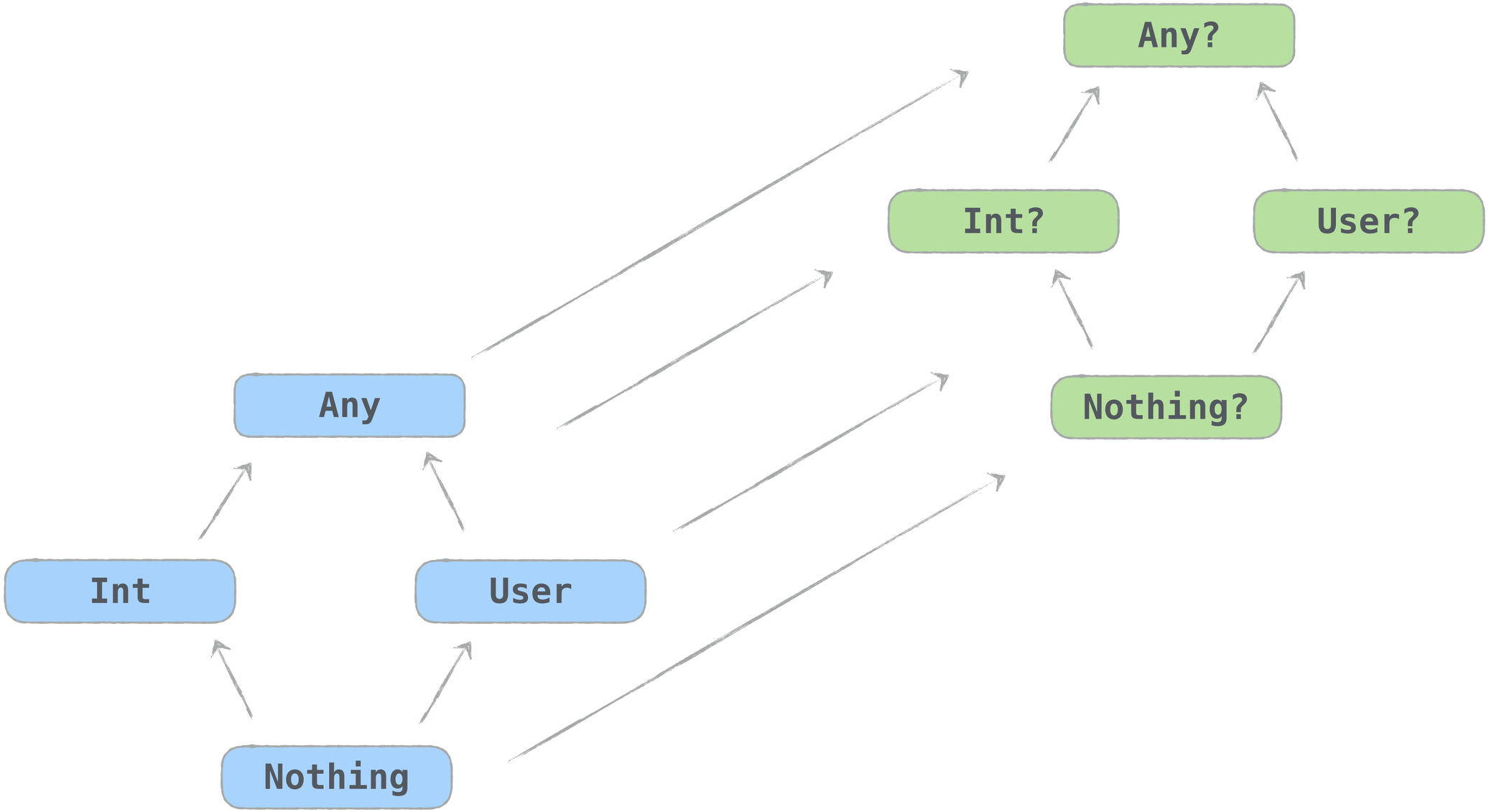
Kotlin	Java
Nothing	void

because we don’t have Nothing at JVM

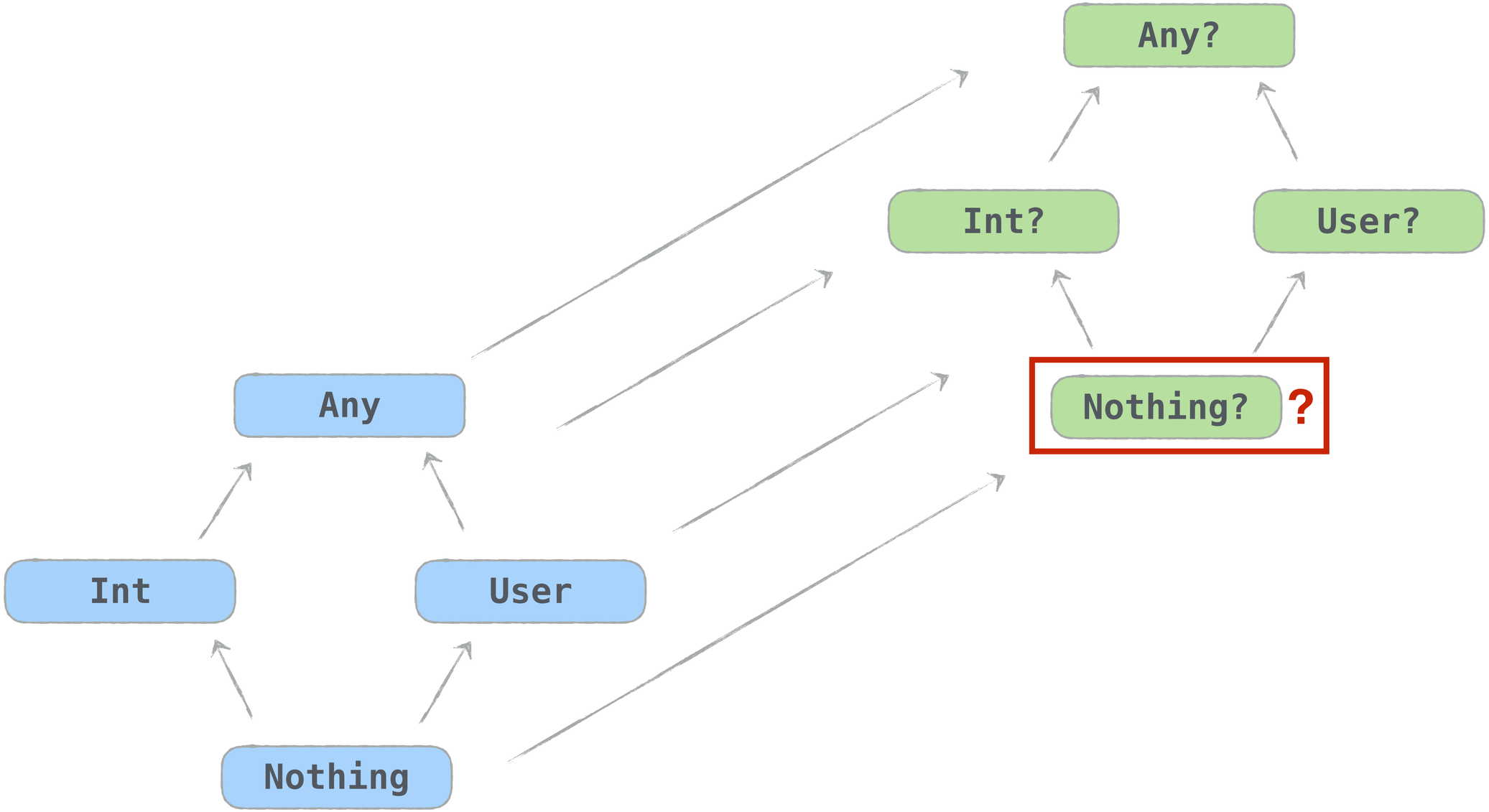
Any & Nothing types



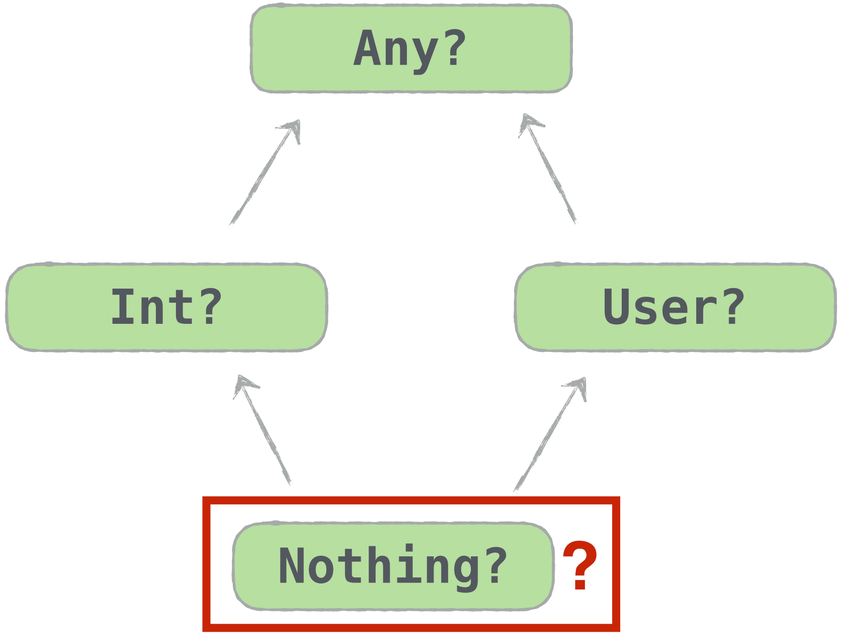
Type hierarchy



Type hierarchy

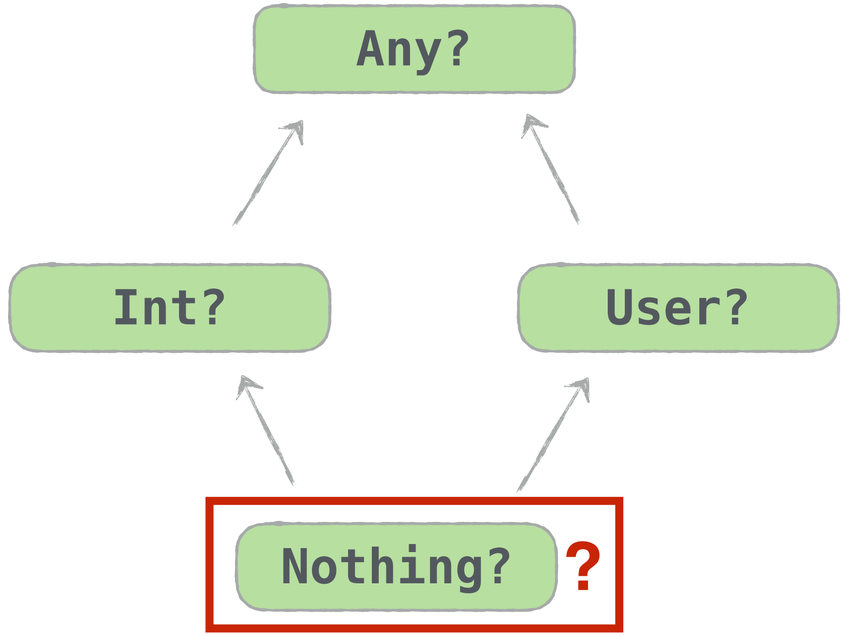


Write the simplest expression of `Nothing?` type



Write the simplest expression of `Nothing?` type

```
var smth: Nothing? = null
```



Type of `null`

```
var user = null
user = User("svtk")
```

```
Error: Type mismatch:
inferred type is User
but Nothing? was expected
```

```
val users = mutableListOf(null)
users.add(User("svtk"))
```

```
Error: Type mismatch:
inferred type is User
but Nothing? was expected
```

Type of `null`

```
var user: Nothing? = null
user = User("svtk")
```

```
Error: Type mismatch:
inferred type is User
but Nothing? was expected
```

```
val users: List<Nothing?> = mutableListOf(null)
users.add(User("svtk"))
```

```
Error: Type mismatch:
inferred type is User
but Nothing? was expected
```

Specify types explicitly

```
var user: User? = null
user = User("svtk")
```

✓

```
val users: List<User?> = mutableListOf(null)
users.add(User("svtk"))
```

✓