ackee edu

BI-IOS

Igor Rosocha, Jakub Olejník, Rostislav Babáček

Lecture 3 - SwiftUI

SwiftUI

Declarative UI

Based on existing UI librarires (UIKit / AppKit)

One ”language” for all the platforms

Interactful

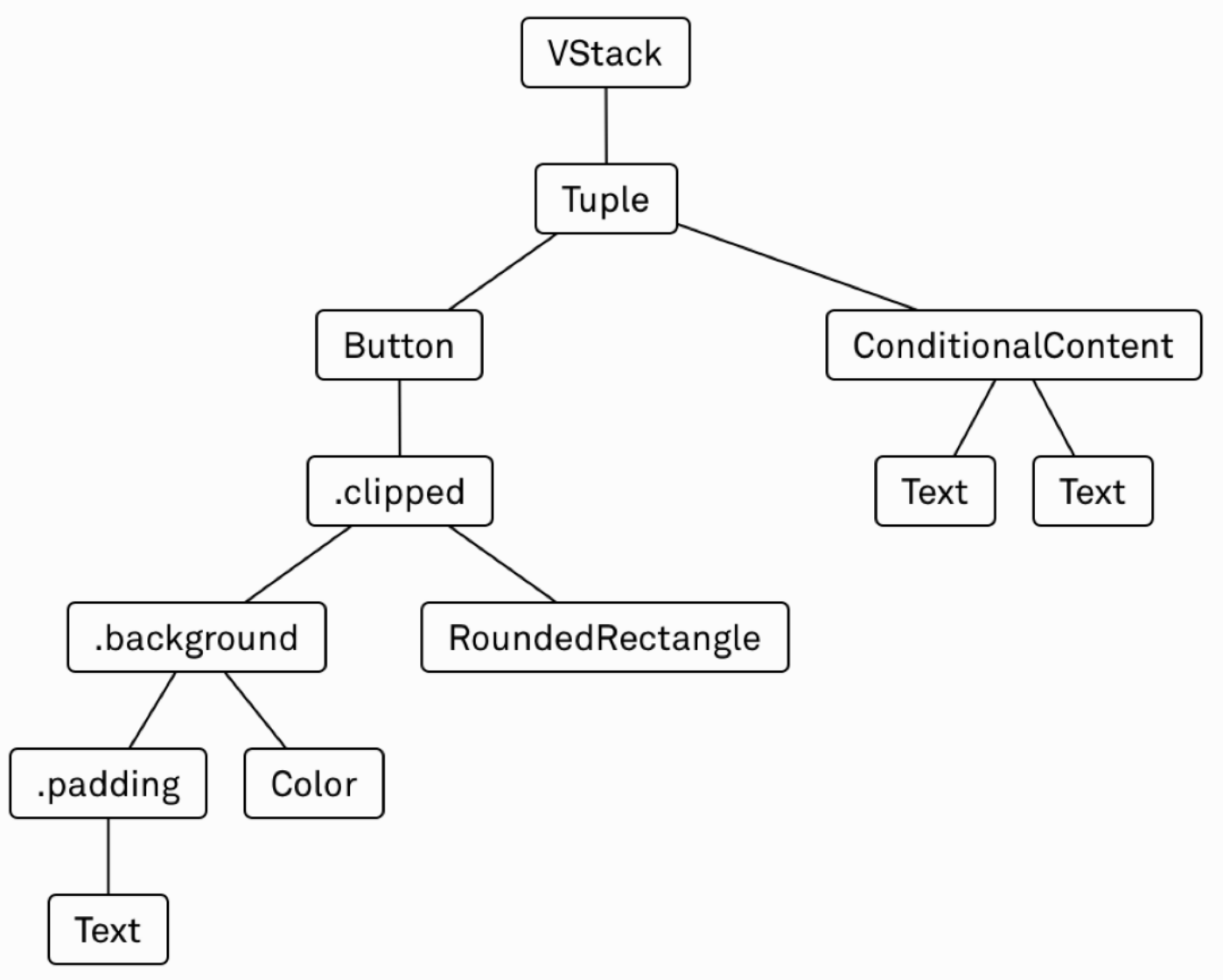
Views

Everything is `View`

Everything is `struct`

A tree representing the UI is created by successive assembly

Tree


VStack
Tuple
Button
ConditionalContent
.clipped
Text
Text
.background
RoundedRectangle
.padding
Color
Text

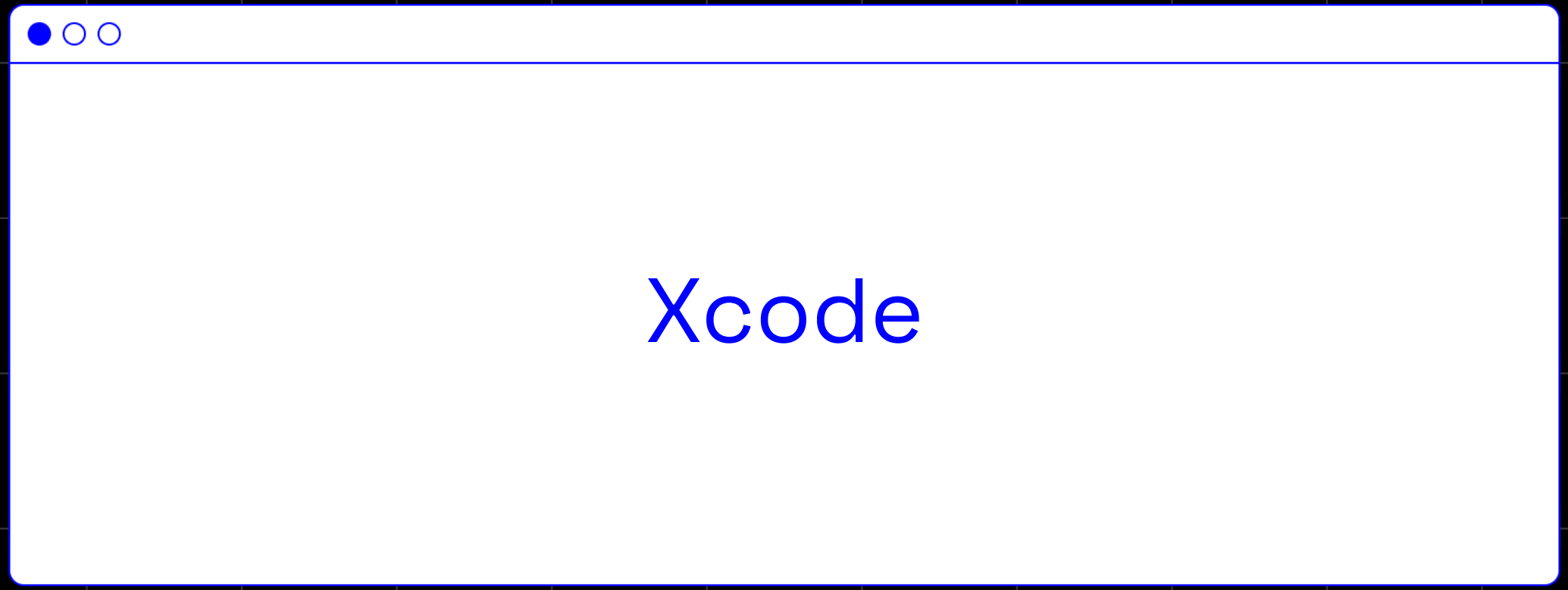
`some View`

Opaque type

Resolves generics - no need to specify explicitly

Compilator does know about the exact type it's working with

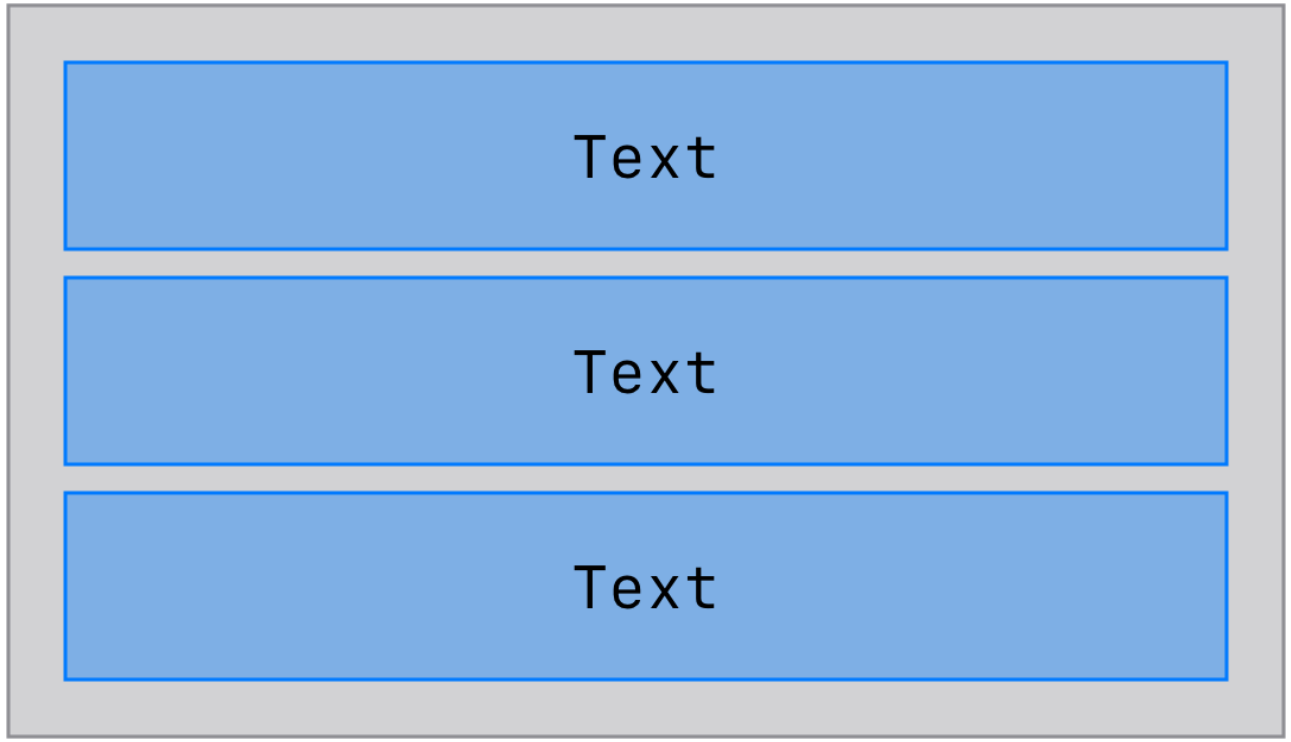
Return type has to be the same for all possible passes


Xcode

Questions?

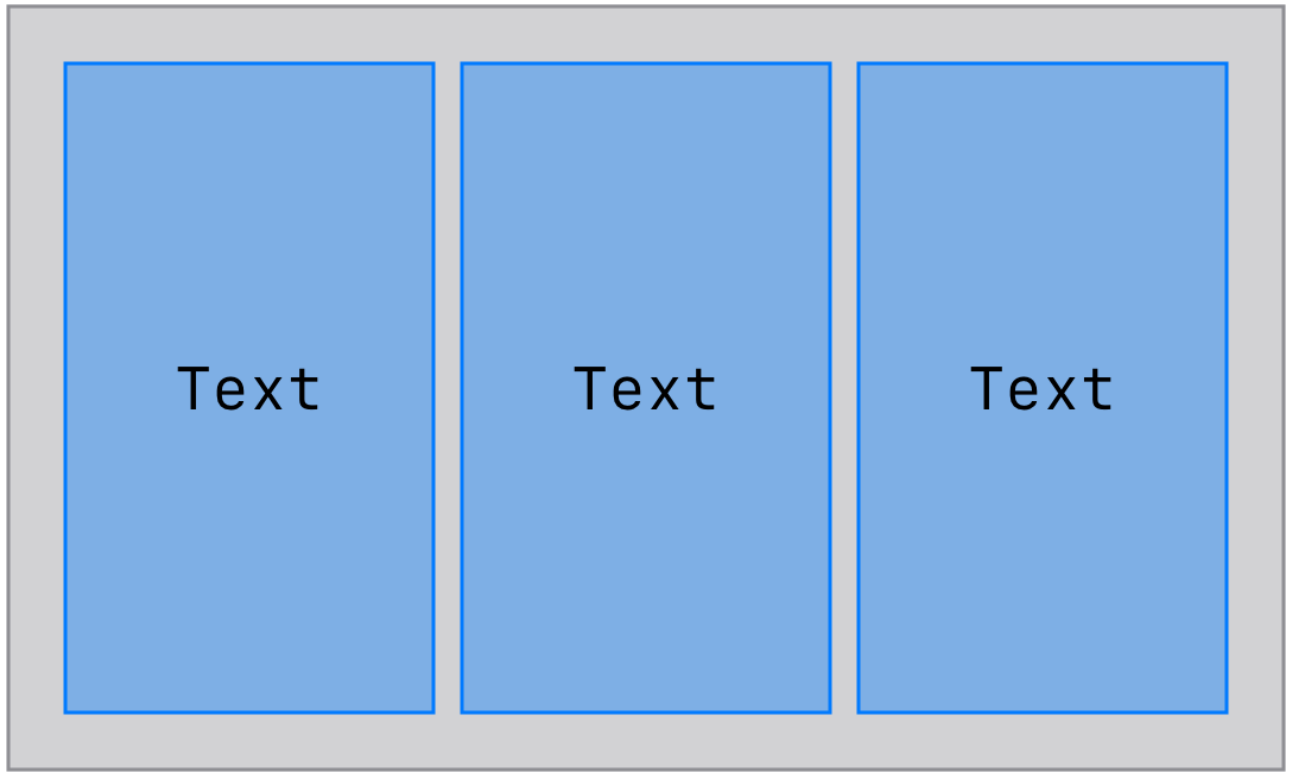
- ❑ SwiftUI
- ❑ Views
- ❑ `some View`
- ❑ Xcode

`VStack`



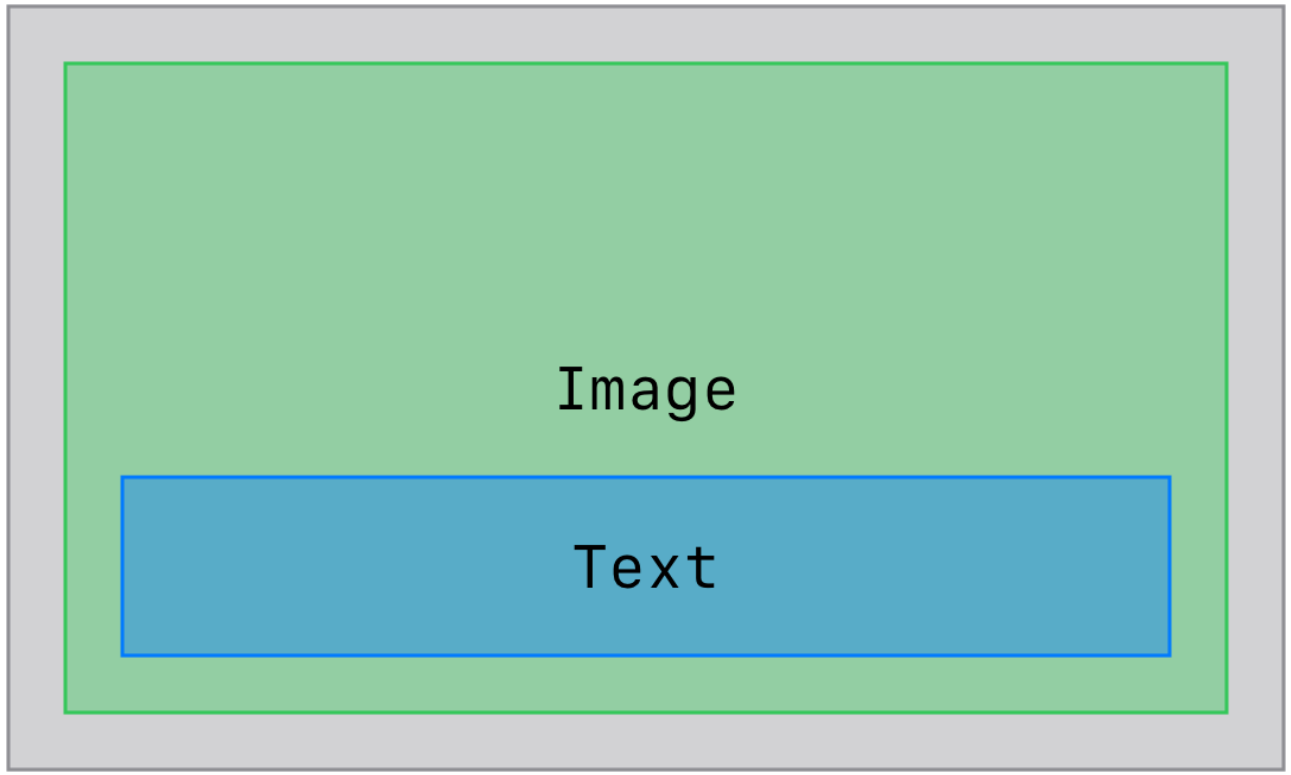
`alignment` a `spacing` can be customized

`HStack`



`alignment` a `spacing` can be customized

`ZStack`

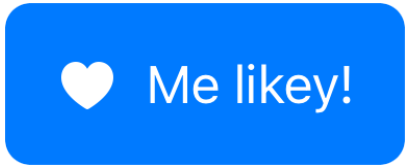


Only `alignment`, but has more options

`Text`

A simple representation of a text string in UI

`Button`



Consists of two parts:

- action – what should happen after tap
- label – how the button should look

Don’t forget about the pressed/highlighted state!

It should be clear at first glance what user can interact with

`Image`

Representation of image in UI

Images are stored in Asset catalog

Apple system icons can be used

Other Basic Views

`Color`

Shapes (`Rectangle`, `RoundedRectangle`, `Capsule`, `Ellipse`, `Circle`)

`Label`

`Group`

`Spacer`

`Divider`

SFSymbols

Apple system icons

Simple and good-looking icons that match the native system style

Native app for browsing the icons

Questions?

- ❏ `VStack`, `HStack`, `ZStack`
- ❏ `Text`
- ❏ `Button`
- ❏ `Image`
- ❏ SFSymbols

Sizing

Two phases

1. Top-down

`View` receives the maximum possible size to render from its parent

- ❏ If it's a leaf, the top-down does not continue
- ❏ If it's not a leaf, `View` gradually passes size for his descendants

Sizing

Two phases

2. Bottom-up

- ❏ Stars in a leaf, which is rendered to given size and then sends its size to parent
- ❏ Parent collects the sizes of its children, renders itself, and sends the size to its parent

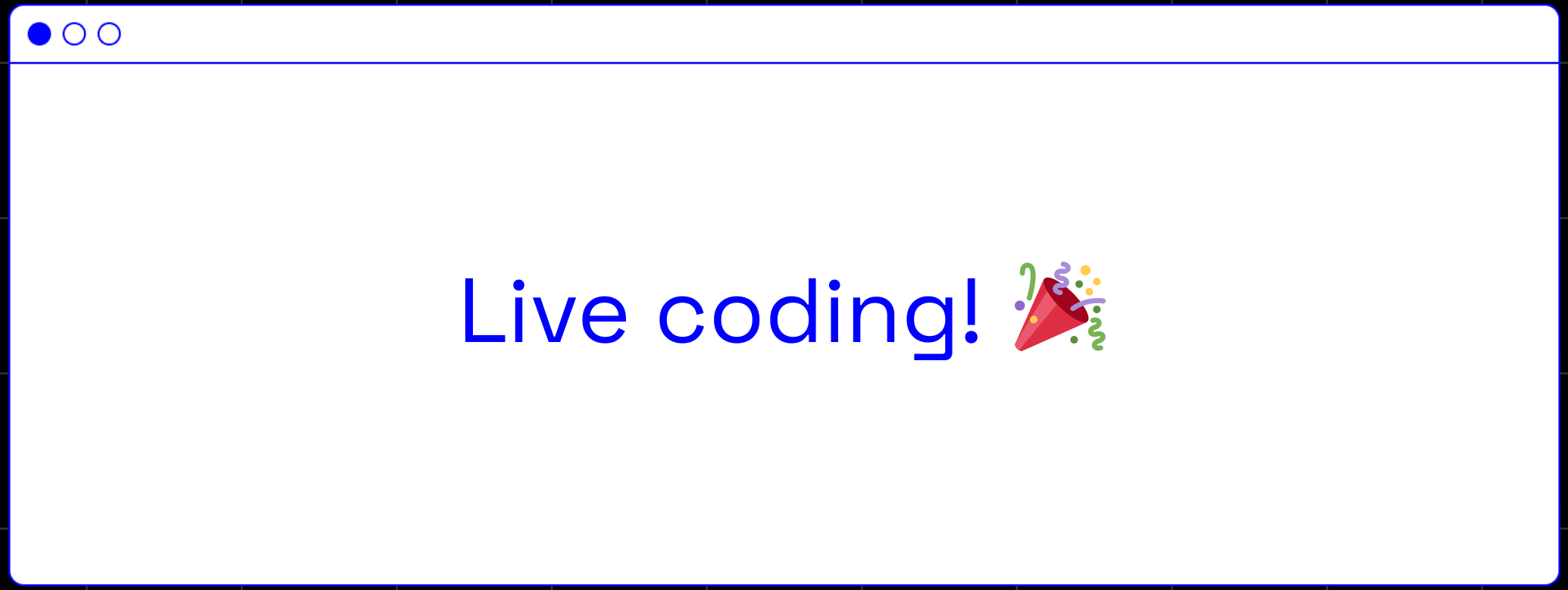
View modifiers

Modifies given `View`, or all its nested descendants

Applying view modifier creates a new `View`

Are propagated top-down

Order matters ❗


Live coding!

jmeno.uzivatele

176 240 To se mi líbí

jmeno.uzivatele Super popisek mého příspěvku na FITstagram, který bude mít určitě hromady lajků.

Zobrazit všechny komentáře (2 356)

VStack

HStack

Text

jmeno.uzivatele

Button

Image

HStack

Button Button Button Button

Text

176 240 To se mi líbí

Text

jmeno.uzivatele Super popisek mého příspěvku na FITstagram, který bude mít určitě hromady lajků.

Text

Zobrazit všechny komentáře (2 356)

Questions?

`ScrollView`

Allows to create scrolling containers of views

Can be either vertical or horizontal

Automatically sizes itself to fit the content that is placed inside it

Content width affects scrollability ❗

`ForEach`

Looping over a sequence to create views

It's a view struct, which means you can return it directly from your view body

Requires identifier to identify each of the items

`LazyVStack` / `LazyHStack`

View that doesn't create items until it needs to render them onscreen

View will remain in memory

Automatically has a flexible preferred width ❗

`LazyVGrid` / `LazyHGrid`

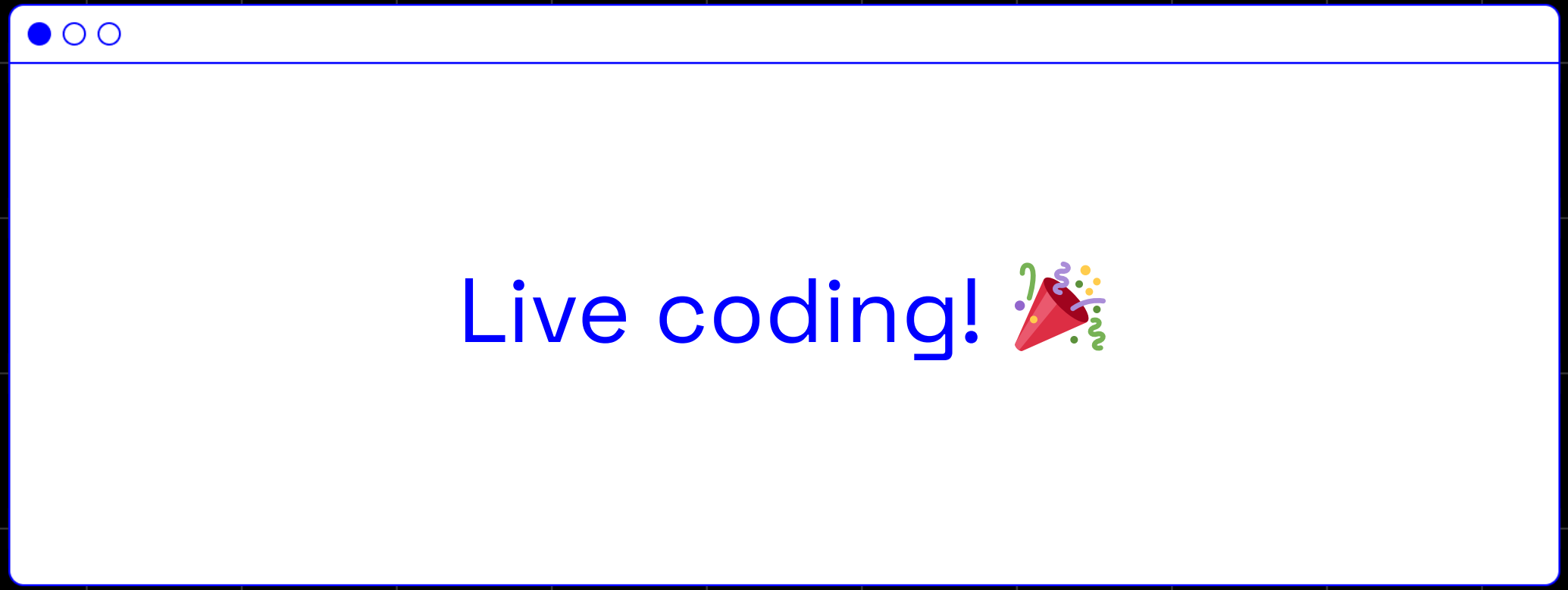
Grid layouts with a fair amount of flexibility

Number of GridItems specify the number of columns/rows

Columns fit the width/height of the screen, height/width is specified by the views itself

Questions?

- ❏ `ScrollView`
- ❏ `ForEach`
- ❏ `LazyVStack` / `LazyHStack`
- ❏ `LazyVGrid` / `LazyHGrid`


Live coding!

Why structs?

All the views are trivial structs and are almost free to create - no other part holds the reference

Structs are simpler and faster than classes

SwiftUI encourages us to move to a more functional design approach

Essential protocols

Equatable

Hashable

Identifiable

Comparable

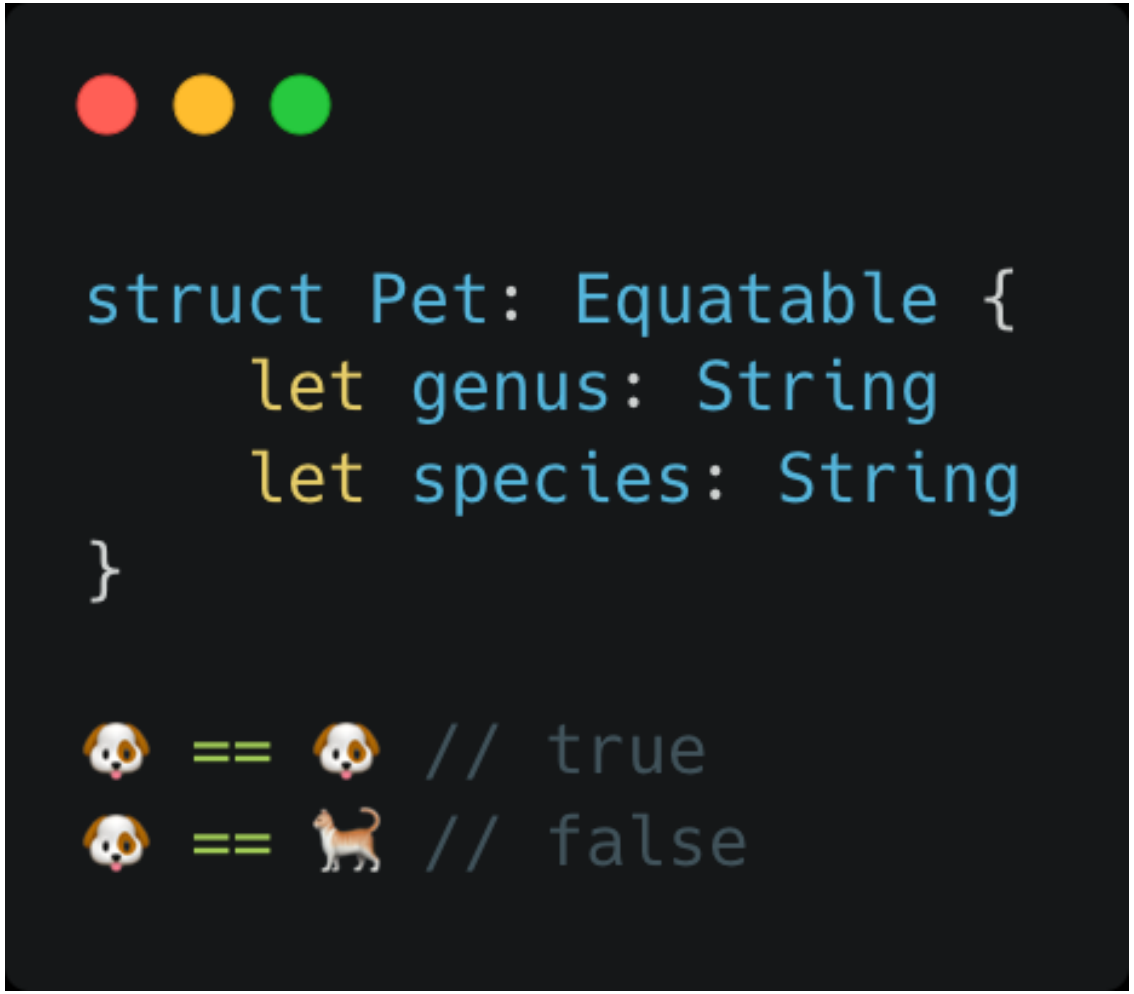
Equatable

Values conforming to the Equatable protocol can be evaluated for equality

Compiler can automatically synthesize conformance for structures with Equatable properties

Lets a value be found in a collection and matched in a switch statement

Equatable


struct Pet: Equatable {
let genus: String
let species: String
}
🐶 == 🐶 // true
🐶 == 🐈 // false

Equatable

❌ Not necessary!

```
extension Pet: Equatable {
    static func == (lhs: Binomen, rhs: Binomen) -> Bool {
        return lhs.genus == rhs.genus &&
               lhs.species == rhs.species
    }
}

🐶 == 🐶 // true
🐶 == 🦙 // false
```

Hashable

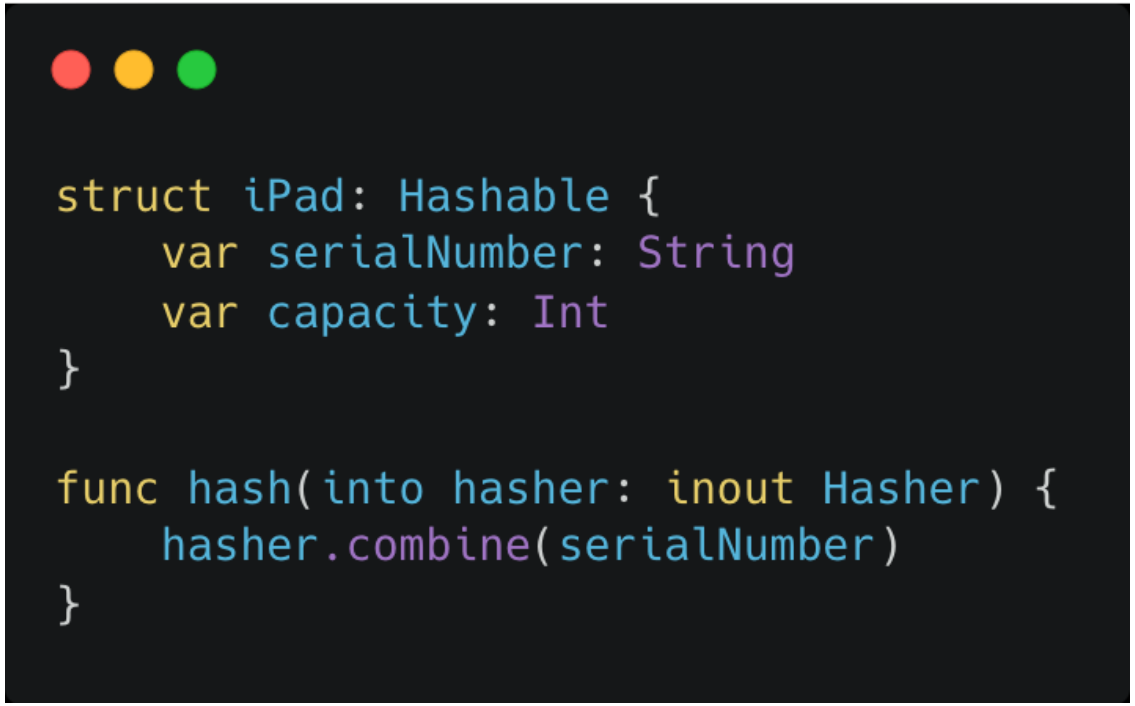
A type that provides an integer hash value

Any type that conforms to Hashable must also conform to Equatable

Mandatory for Set items and Dictionary keys

Conforming is often just as easy as adding Hashable to your struct conformance

Hashable



```
struct iPad: Hashable {
    var serialNumber: String
    var capacity: Int
}

func hash(into hasher: inout Hasher) {
    hasher.combine(serialNumber)
}
```

Identifiable

Values of types adopting the Identifiable protocol provide a stable identifier for the entities they represent

```
protocol Identifiable {
    associatedtype ID: Hashable
    var id: ID { get }
}
```

Identifiable

```
struct Post: Identifiable {
    let id: String
    let username: String
    let likes: Int
    let description: String
}
```

Comparable

Allows for values to be considered less than or greater than other values

You can get away with only implementing the `<` operator

Comparable

```
struct Post: Comparable {
    let username: String
    let likes: Int
    let description: String

    static func < (lhs: Post, rhs: Post) -> Bool {
        lhs.likes < rhs.likes
    }
}
```

Questions?

- ❑ Equatable
- ❑ Hashable
- ❑ Identifiable
- ❑ Comparable

Thank you very much,
see you next week!