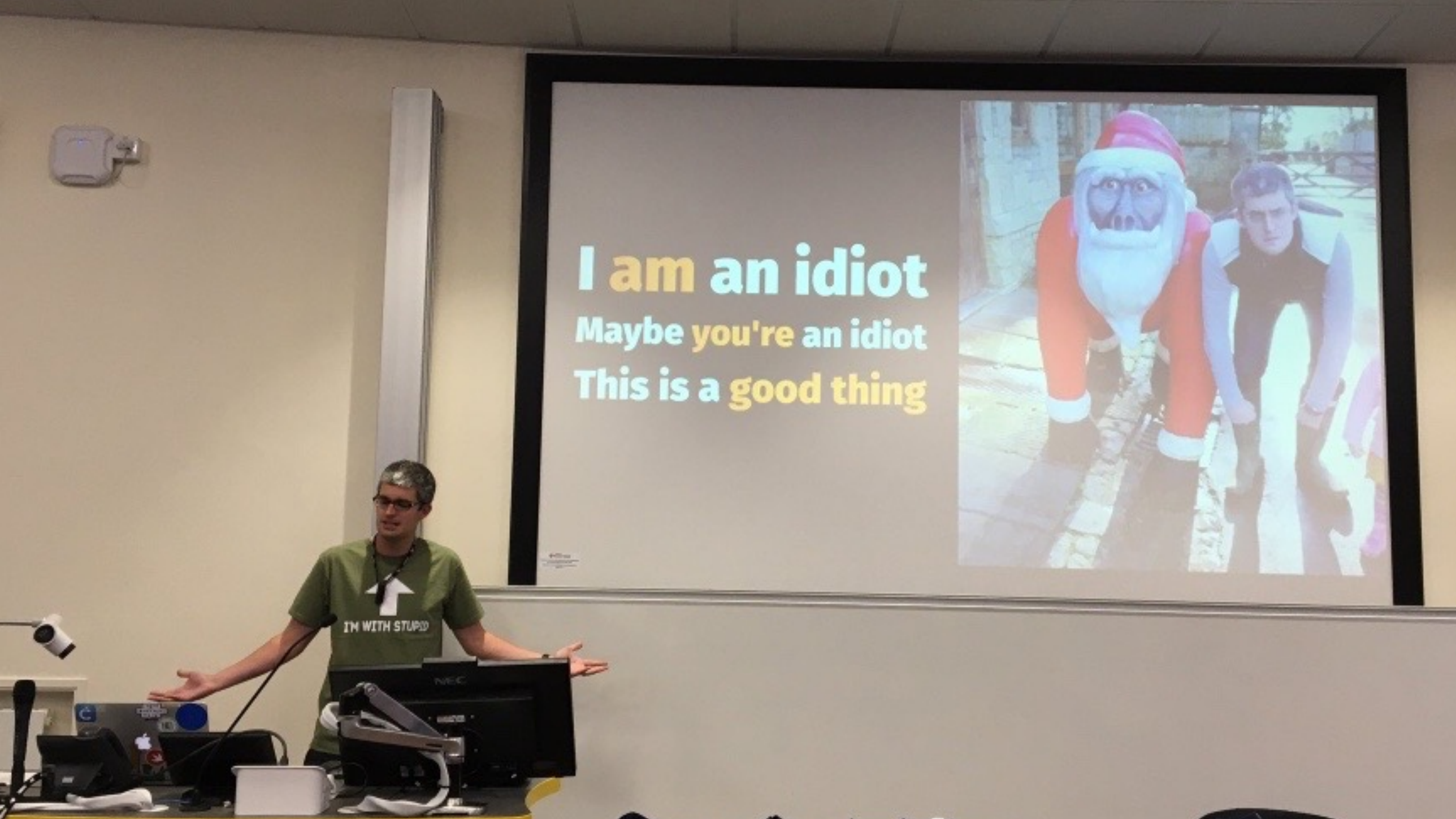
You've got a canvas, now make some art*!

Richard Turton
@richturton
iOSDevUK 2022

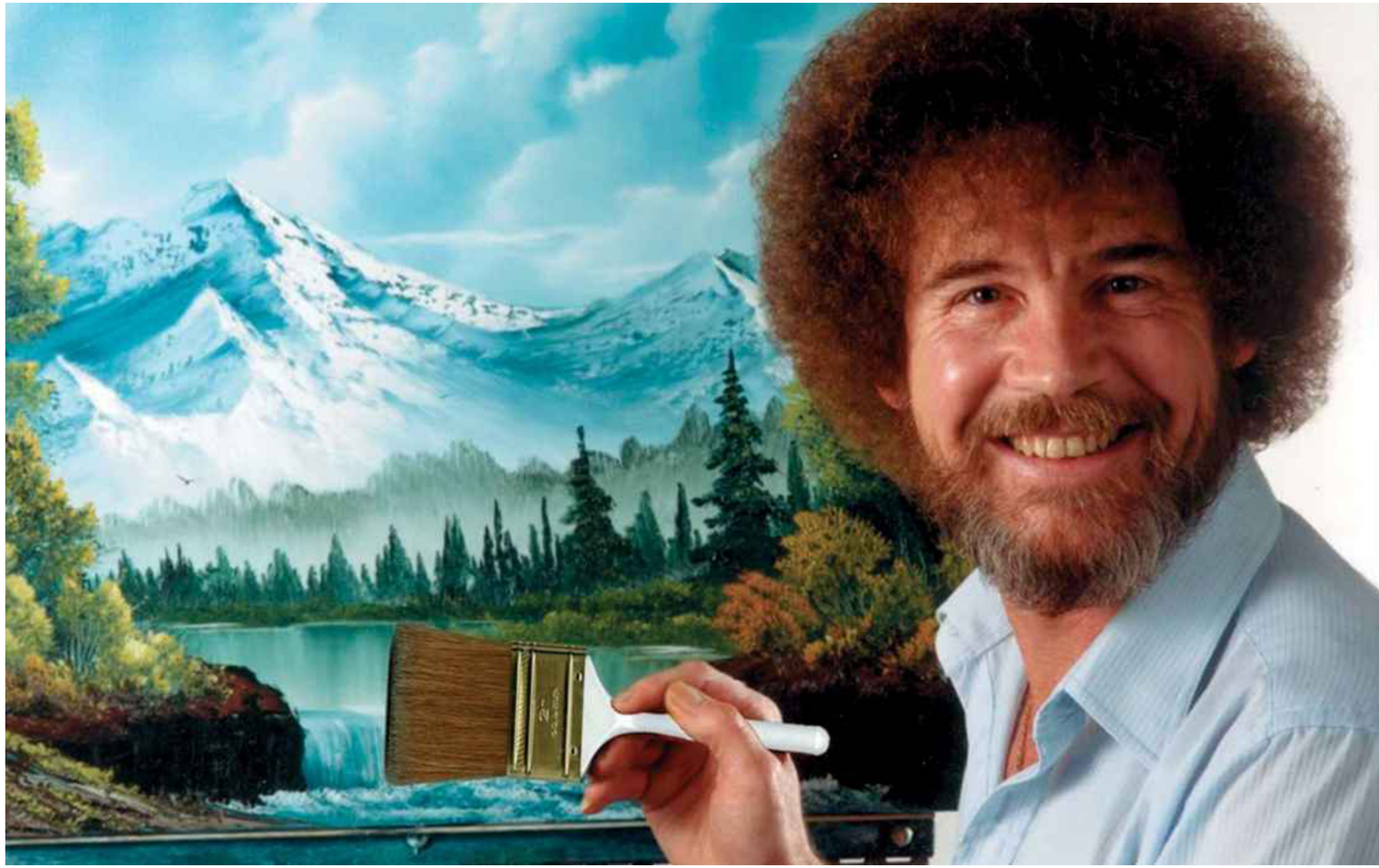
*Art in the Edward Tufte sense, I'm afraid


I am an idiot
Maybe you're an idiot
This is a good thing
I'M WITH STUPID

Lowering your expectations

Because a pessimist is never disappointed

- “I hope there’s lots of math(s)!”
- “It’d better not have loads of maths!”
- “I’m looking forward to seeing your artwork”



SwiftUI

SwiftUI

Canvas

SwiftUI

Canvas

Transforms

SwiftUI

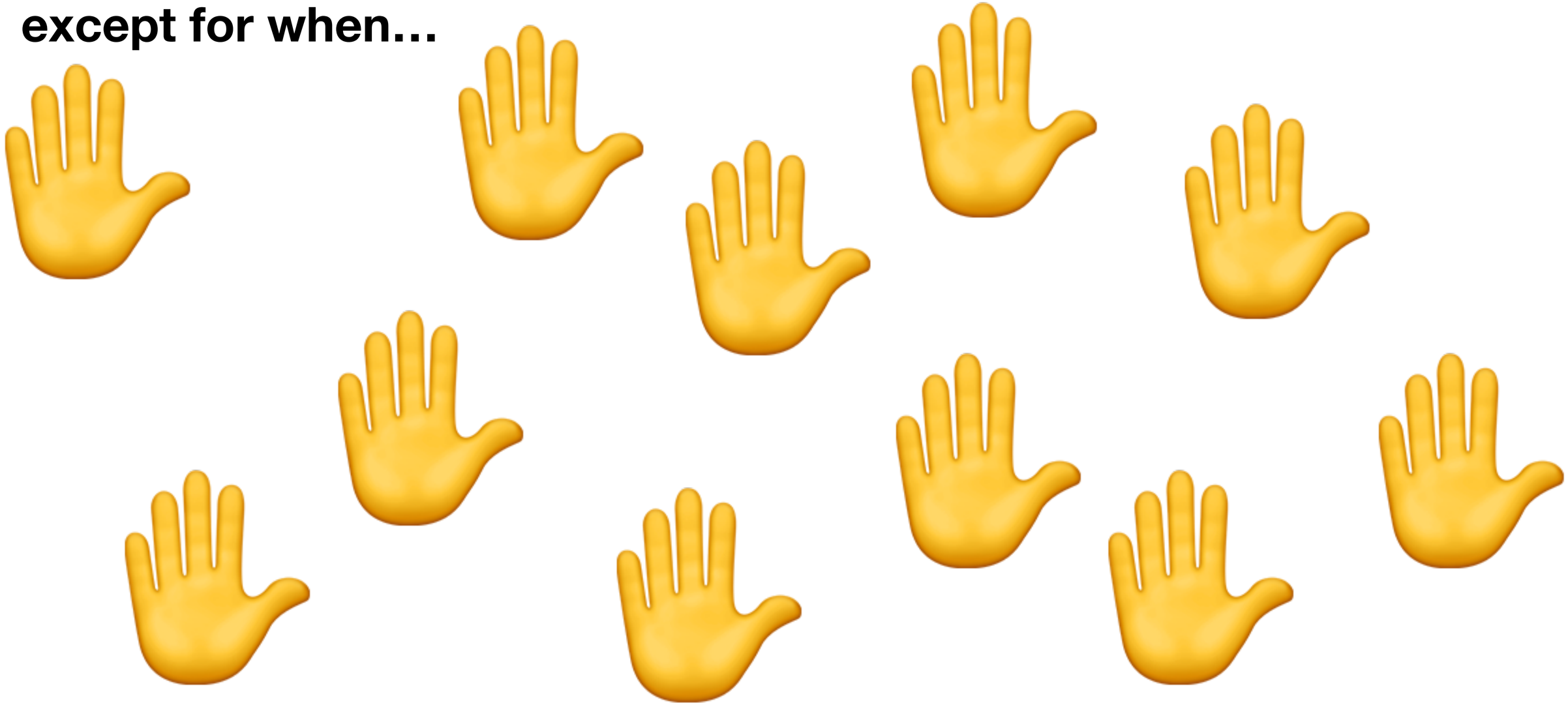
Canvas

Transforms

Data Visualisation

SwiftUI is Great!

except for when…



The beautiful tyranny of ViewBuilders

```
VStack {
  Text(“North”)
  Spacer()
  Text(“South”)
}
```

The beautiful tyranny of ViewBuilders

```
VStack {
  Text(“North”)
  Spacer()
  Text(“South”)
}
```

```
VStack(contents: {
  let v0 = Text(“North”)
  let v1 = Spacer()
  let v2 = Text(“South”)
  return ViewBuilder.buildBlock(v0, v1, v2)
  // TupleView<(Text, Spacer, Text)>
  }
)
```

The beautiful tyranny of ViewBuilders

```
VStack {
  Text(“North”)
  Spacer()
  Text(“South”)
}
```

```
VStack(contents: {
  let v0 = Text(“North”)
  let v1 = Spacer()
  let v2 = Text(“South”)
  return ViewBuilder.buildBlock(v0, v1, v2)
  // TupleView<(Text, Spacer, Text)>
  }
)
```

- Becca Royal-Gordon WWDC21 "Write a DSL in Swift using Result Builders”

The beautiful tyranny of ViewBuilders

Inline logic

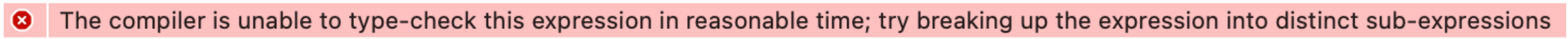
- All lines must return void, or a view
- `if let`, `if / else` and `switch` are supported by result builders
- Evaluated conditions must be calculated in-line, or outside the view builder

~~`let showWarning = cheese < 10 || baguetteAge > 0`~~

```
if showWarning {
 Text(“Bad picnic!”)
}
```

The beautiful tyranny of ViewBuilders

Type checking



- The compiler is unable to type-check this expression in reasonable time; try breaking up the expression into distinct sub-expressions
- Multiply nested view builders count as a single expression
- Every view modifier returns a new instance, with potentially a new type
- Lots of generics are involved

We don’t talk about layout

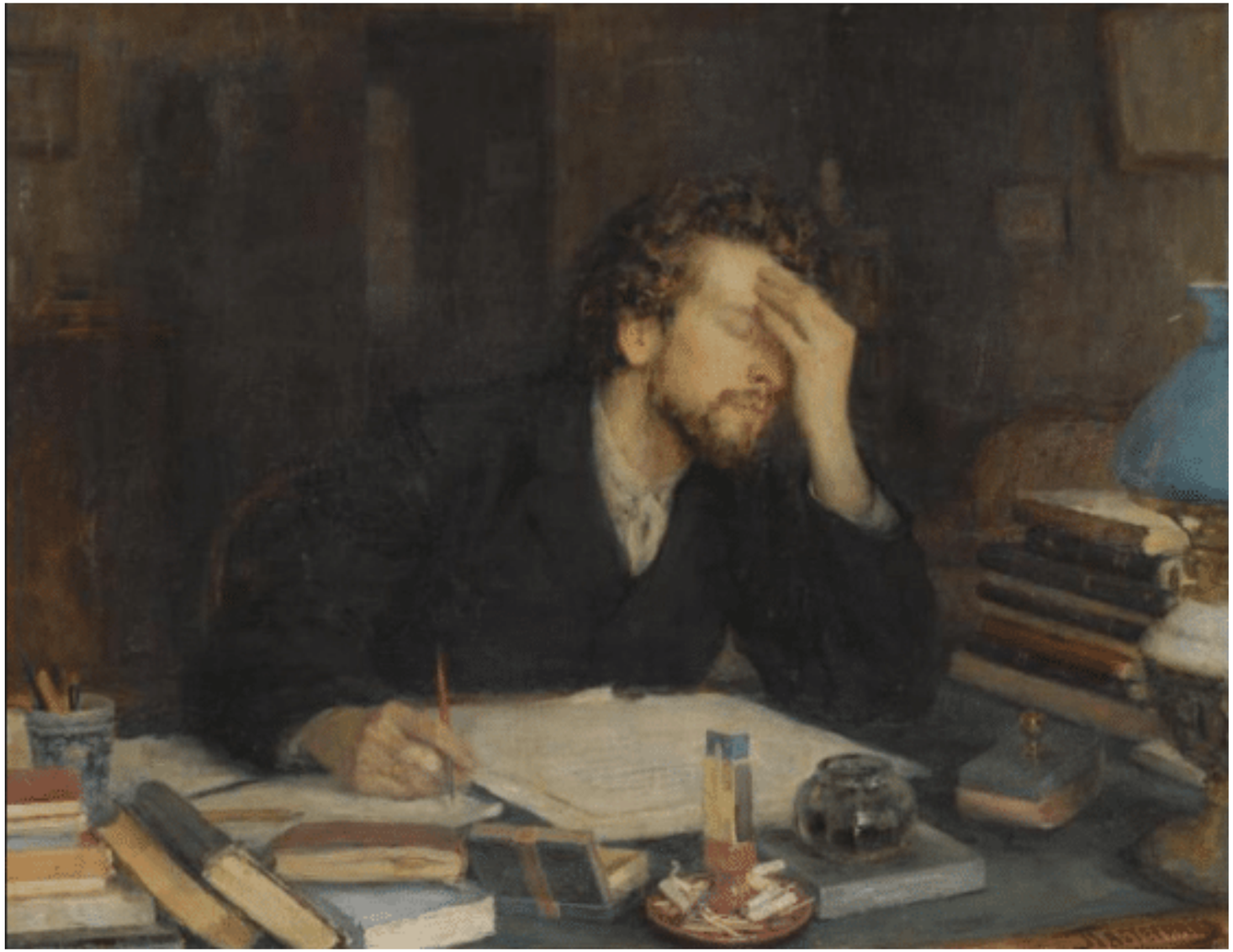
But I *want* to talk about it…

We don’t talk about layout

But I *want* to talk about it…

```
GeometryReader { geometry in
  // Oh no…
  // It’s another ViewBuilder!
}
```

Artistic Expression



leslieshershberger.com

Introducing Canvas

```
Canvas { context, size in
  // Just a closure. Remember those?
}
```

Basic Commands

Basic Commands

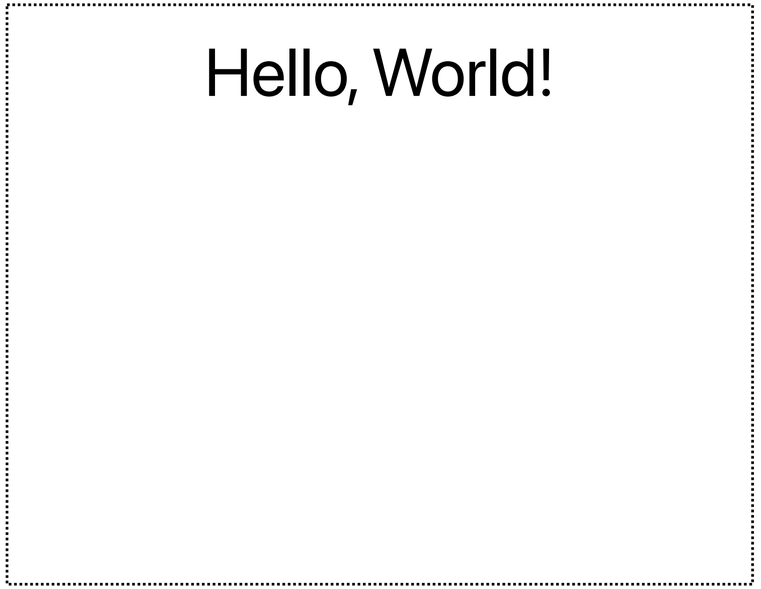
```
Canvas { context, size in
```

Basic Commands

```
Canvas { context, size in
  let midX = size.width * 0.5
  let midY = size.height * 0.5
```

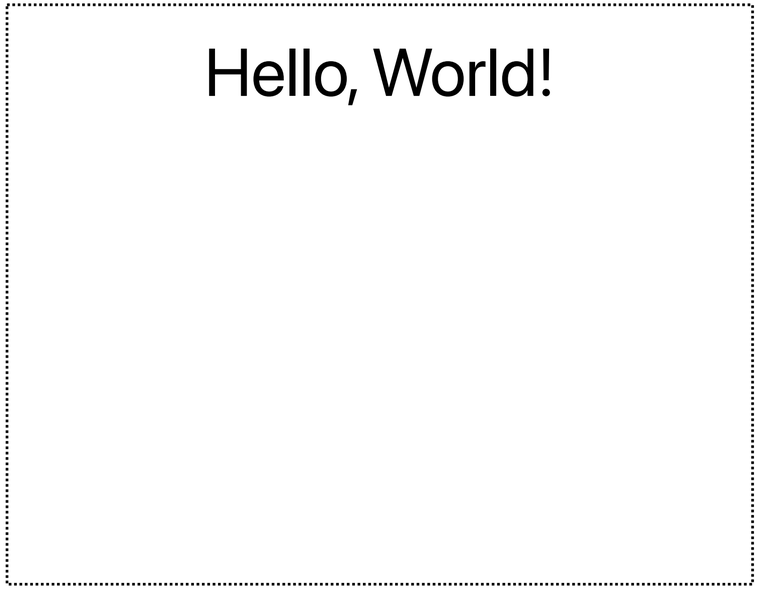
Basic Commands

```
Canvas { context, size in
  let midX = size.width * 0.5
  let midY = size.height * 0.5
  context.draw(
    Text("Hello, World!"),
    at: CGPoint(x: midX, y: 20))
```



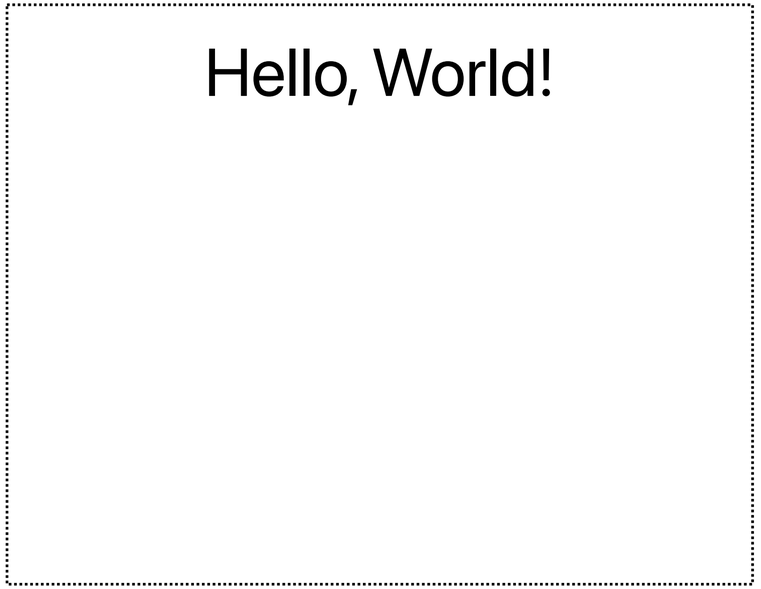
Basic Commands

```
Canvas { context, size in
  let midX = size.width * 0.5
  let midY = size.height * 0.5
  context.draw(
    Text("Hello, World!"),
    at: CGPoint(x: midX, y: 20))
  let circleRect = CGRect(
    x: midX - 30,
    y: midY - 90,
    width: 60,
    height: 60)
```



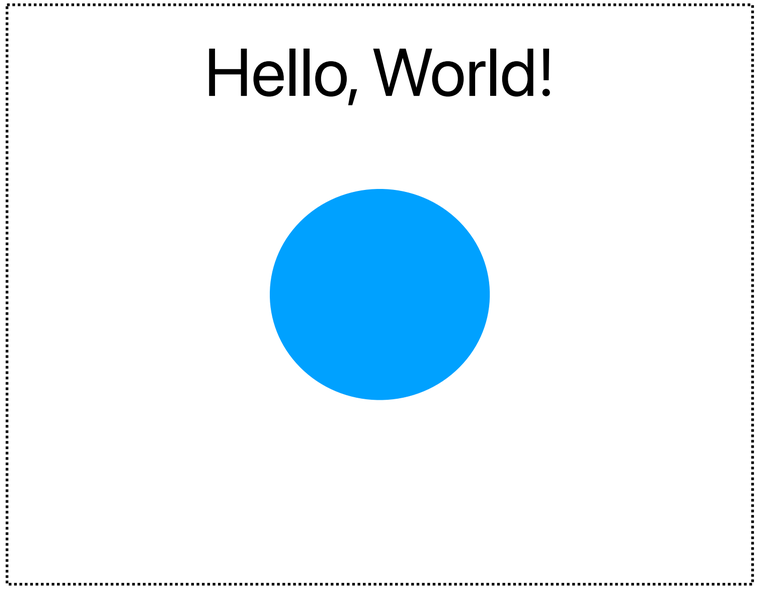
Basic Commands

```
Canvas { context, size in
  let midX = size.width * 0.5
  let midY = size.height * 0.5
  context.draw(
    Text("Hello, World!"),
    at: CGPoint(x: midX, y: 20))
  let circleRect = CGRect(
    x: midX - 30,
    y: midY - 90,
    width: 60,
    height: 60)
  let circle = Path(ellipseIn: circleRect)
```



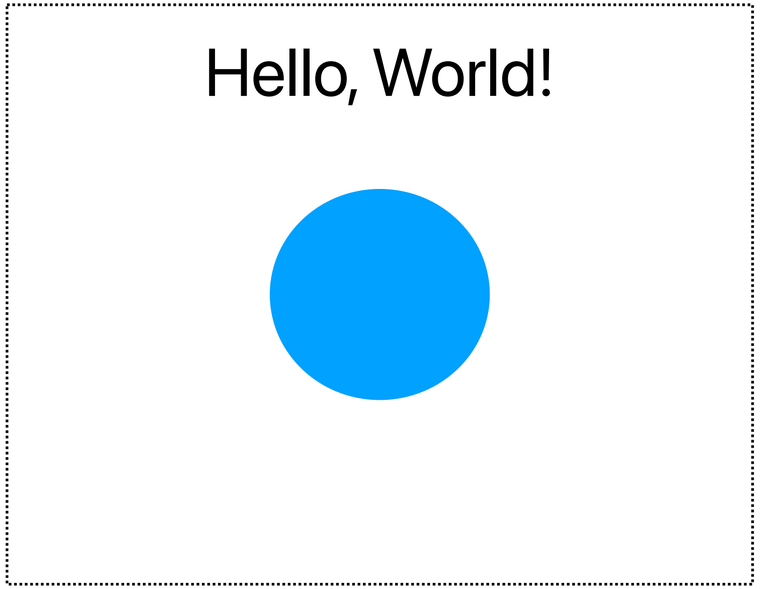
Basic Commands

```
Canvas { context, size in
  let midX = size.width * 0.5
  let midY = size.height * 0.5
  context.draw(
    Text("Hello, World!"),
    at: CGPoint(x: midX, y: 20))
  let circleRect = CGRect(
    x: midX - 30,
    y: midY - 90,
    width: 60,
    height: 60)
  let circle = Path(ellipseIn: circleRect)
  context.fill(circle, with: .color(.blue))
```



Basic Commands

```
Canvas { context, size in
  let midX = size.width * 0.5
  let midY = size.height * 0.5
  context.draw(
    Text("Hello, World!"),
    at: CGPoint(x: midX, y: 20))
  let circleRect = CGRect(
    x: midX - 30,
    y: midY - 90,
    width: 60,
    height: 60)
  let circle = Path(ellipseIn: circleRect)
  context.fill(circle, with: .color(.blue))
  let globe = Image(systemName: "globe.europe.africa.fill")
```



Basic Commands

```
Canvas { context, size in
  let midX = size.width * 0.5
  let midY = size.height * 0.5
  context.draw(
    Text("Hello, World!"),
    at: CGPoint(x: midX, y: 20))
  let circleRect = CGRect(
    x: midX - 30,
    y: midY - 90,
    width: 60,
    height: 60)
  let circle = Path(ellipseIn: circleRect)
  context.fill(circle, with: .color(.blue))
  let globe = Image(systemName: "globe.europe.africa.fill")
  context.draw(globe, in: circleRect)
}
```

Basic Commands

```
Canvas { context, size in
  let midX = size.width * 0.5
  let midY = size.height * 0.5
  context.draw(
    Text("Hello, World!"),
    at: CGPoint(x: midX, y: 20))
  let circleRect = CGRect(
    x: midX - 30,
    y: midY - 90,
    width: 60,
    height: 60)
  let circle = Path(ellipseIn: circleRect)
  context.fill(circle, with: .color(.blue))
  let globe = Image(systemName: "globe.europe.africa.fill")
  context.draw(globe, in: circleRect)
}
```

Resolution

View Type	Resolved Type
`Text`	`ResolvedText`
`Color`	`Shading`
`Image`	`ResolvedImage`
`View`	`ResolvedSymbol`

Symbols

- Pass in symbols via a view builder ☠️ init variant
- Each view has a `tag`
- `context.resolveSymbol(id: 🤦)`
- `GraphicsContext.ResolvedSymbol?`

Limitations

Limitations

Limitations

- What happens in canvas, stays in canvas

Limitations

- What happens in canvas, stays in canvas
- Interaction

Limitations

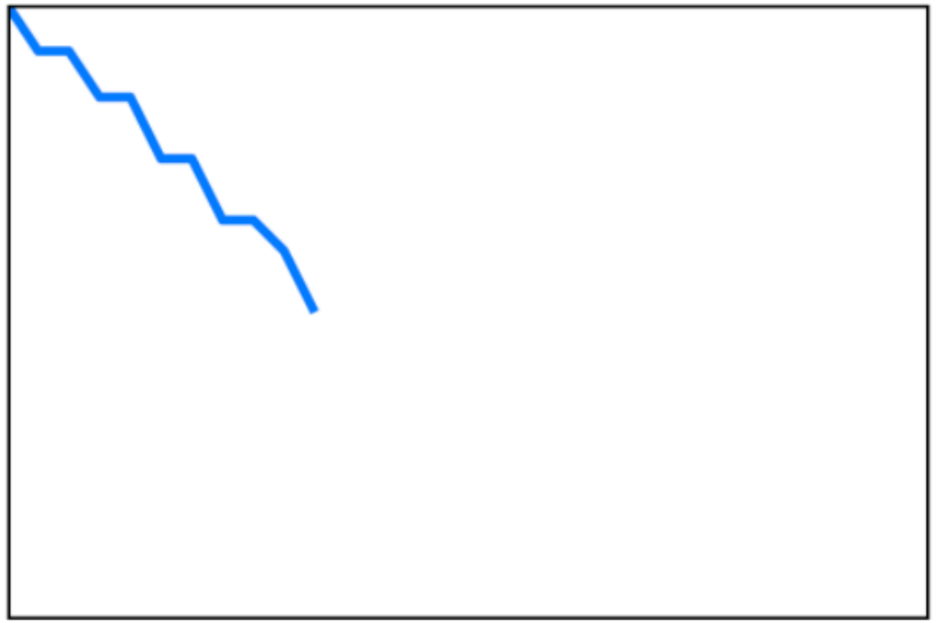
- What happens in canvas, stays in canvas
- Interaction
- Accessibility

Data-driven drawing

```
context, size in

let points = [
  CGPoint(x: 0, y: 0),
  CGPoint(x: 10, y: 15),
  CGPoint(x: 20, y: 15),
  CGPoint(x: 30, y: 30),
  CGPoint(x: 40, y: 30),
  CGPoint(x: 50, y: 50),
  CGPoint(x: 60, y: 50),
  CGPoint(x: 70, y: 70),
  CGPoint(x: 80, y: 70),
  CGPoint(x: 90, y: 80),
  CGPoint(x: 100, y: 100)
]

let path = Path { p in
  p.addLines(points)
}

context.stroke(
  path,
  with: .color(.blue),
  style: StrokeStyle(lineWidth: 3)
)
```

Data-driven drawing

```
context, size in

let points = [
  CGPoint(x: 0, y: 0),
  CGPoint(x: 10, y: 15),
  CGPoint(x: 20, y: 15),
  CGPoint(x: 30, y: 30),
  CGPoint(x: 40, y: 30),
  CGPoint(x: 50, y: 50),
  CGPoint(x: 60, y: 50),
  CGPoint(x: 70, y: 70),
  CGPoint(x: 80, y: 70),
  CGPoint(x: 90, y: 80),
  CGPoint(x: 100, y: 100)
]

let path = Path { p in
  p.addLines(points)
}

context.stroke(
  path,
  with: .color(.blue),
  style: StrokeStyle(lineWidth: 3)
)
```



Data-driven drawing

```
context, size in

let points = [ … ]

let path = Path { p in
  p.addLines(points)
}



let transform = CGAffineTransform.identity




context.stroke(
  path.applying(transform),
  with: .color(.blue),
  style: StrokeStyle(lineWidth: 3)
)
```

Data-driven drawing

```
context, size in

let points = [ … ]

let path = Path { p in
  p.addLines(points)
}



let transform = CGAffineTransform.identity



context.stroke(
  path.applying(transform),
  with: .color(.blue),
  style: StrokeStyle(lineWidth: 3)
)
```

Data-driven drawing

```
context, size in

let points = [ … ]

let path = Path { p in
  p.addLines(points)
}
```

```
let transform = CGAffineTransform.identity
```

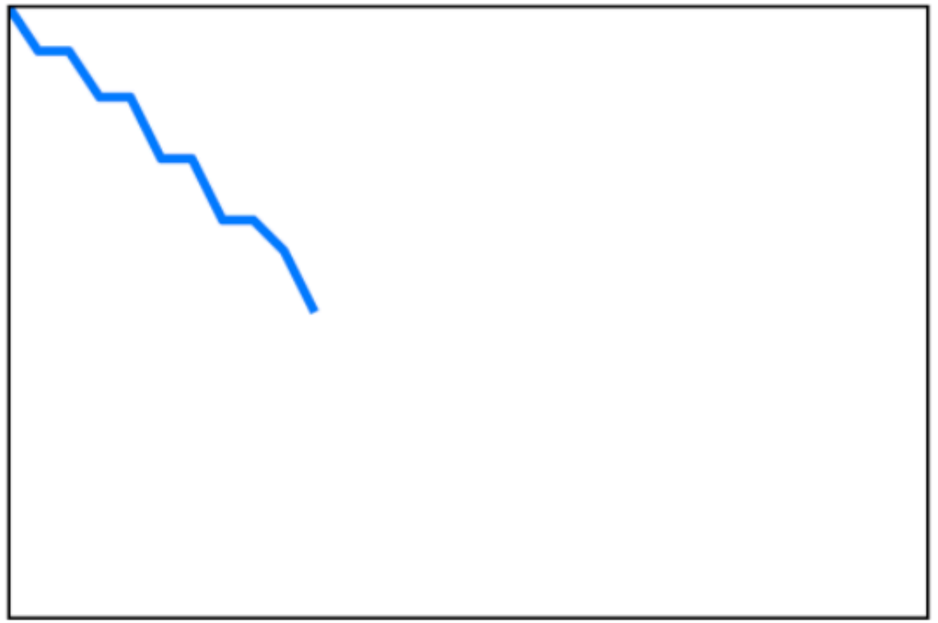
```
context.stroke(
  path.applying(transform),
  with: .color(.blue),
  style: StrokeStyle(lineWidth: 3)
)
```

Data-driven drawing

```
context, size in

let points = [ … ]

let path = Path { p in
  p.addLines(points)
}
```

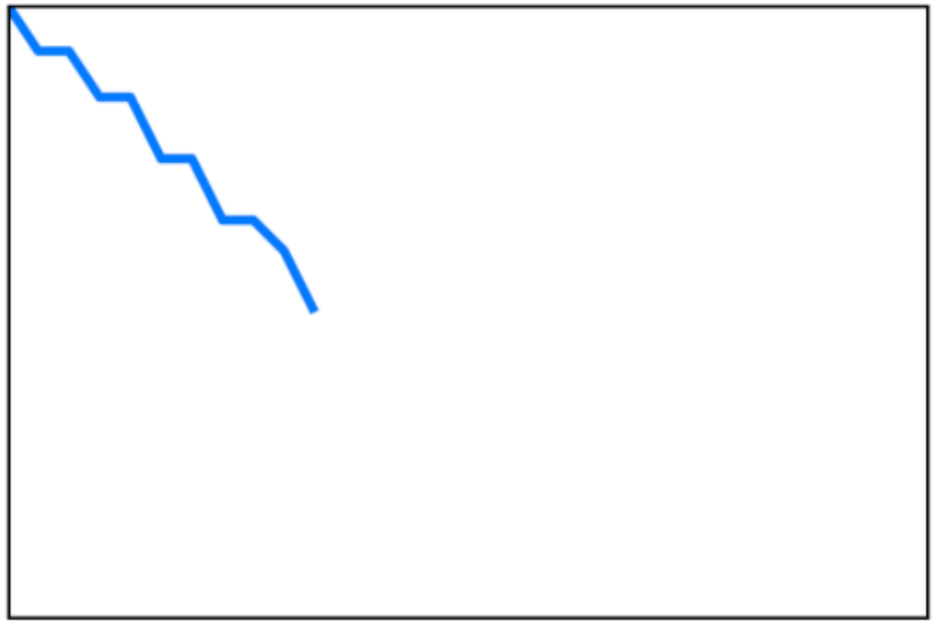
```
let transform = CGAffineTransform.identity
```

```
context.stroke(
  path.applying(transform),
  with: .color(.blue),
  style: StrokeStyle(lineWidth: 3)
)
```



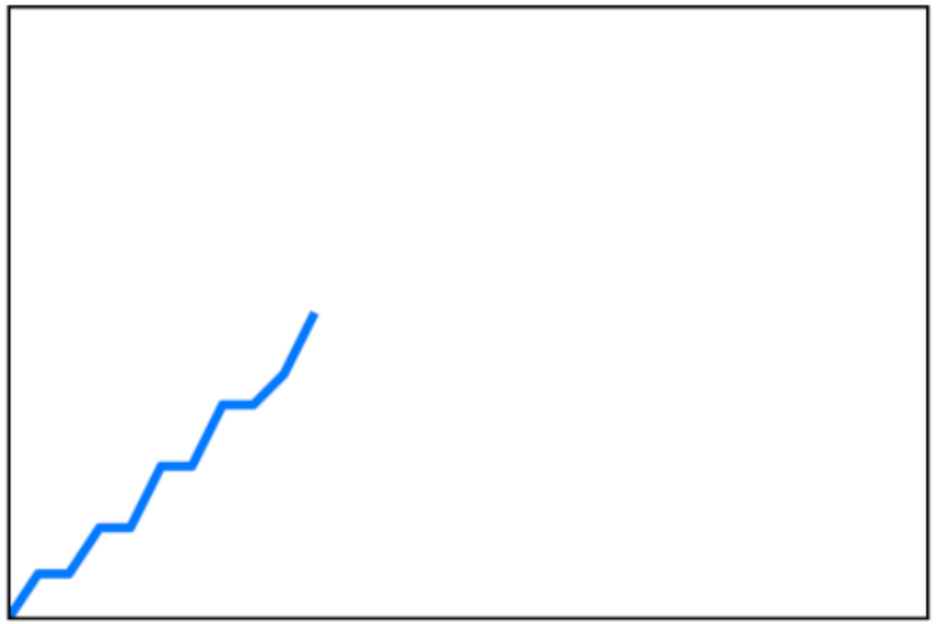
Data-driven drawing

```
context, size in

let points = [ … ]

let path = Path { p in
  p.addLines(points)
}



let transform = CGAffineTransform.identity




context.stroke(
  path.applying(transform),
  with: .color(.blue),
  style: StrokeStyle(lineWidth: 3)
)
```



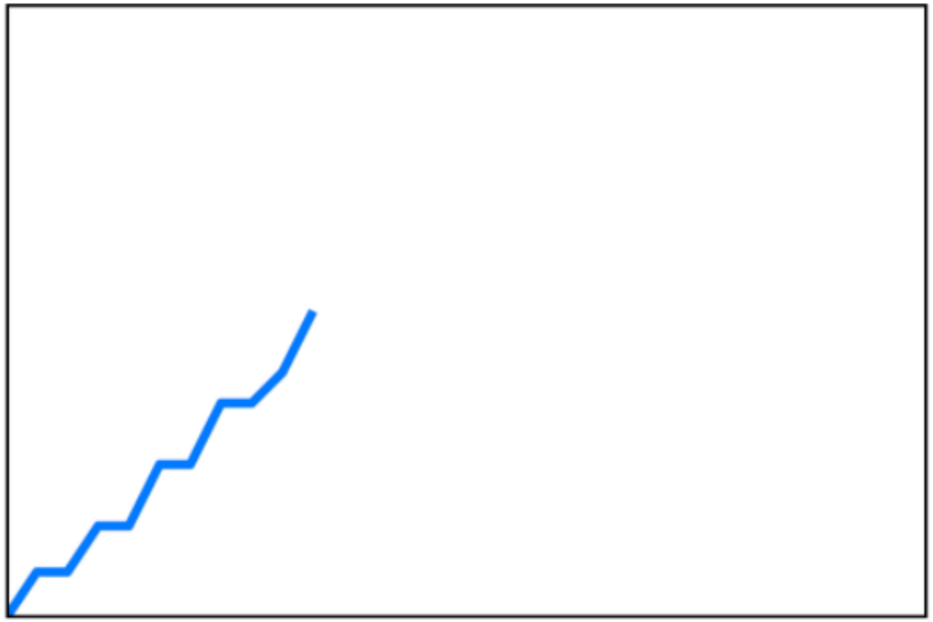
Data-driven drawing

```
context, size in

let points = [ … ]

let path = Path { p in
  p.addLines(points)
}



let transform = CGAffineTransform.identity
  .translatedBy(x: 0, y: size.height)
  .scaledBy(x: 1, y: -1)



context.stroke(
  path.applying(transform),
  with: .color(.blue),
  style: StrokeStyle(lineWidth: 3)
)
```



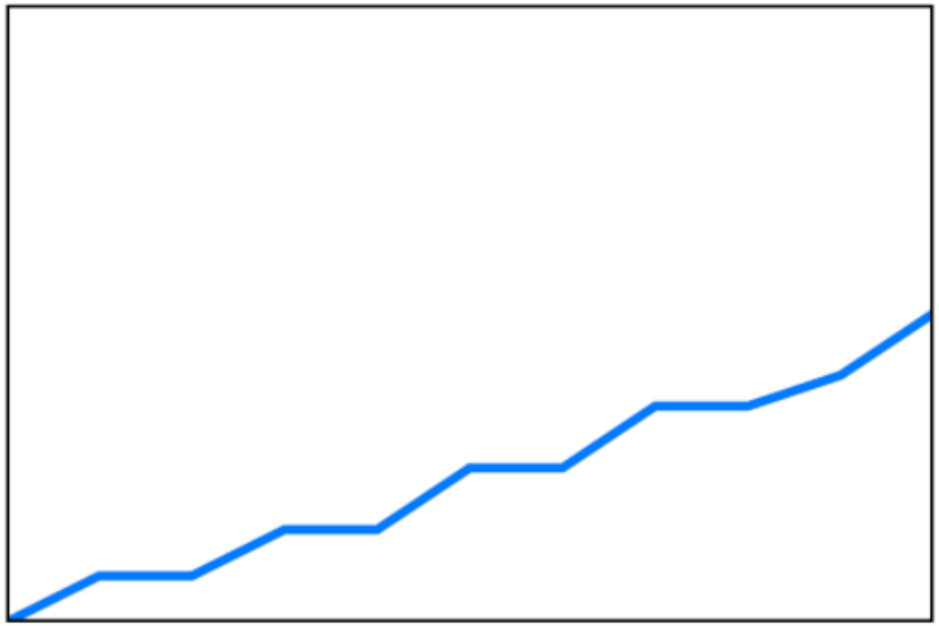
Data-driven drawing

```
context, size in

let points = [ … ]

let path = Path { p in
  p.addLines(points)
}

let dataSize = path.boundingRect.size

let transform = CGAffineTransform.identity
  .translatedBy(x: 0, y: size.height)
  .scaledBy(x: 1, y: -1)



context.stroke(
  path.applying(transform),
  with: .color(.blue),
  style: StrokeStyle(lineWidth: 3)
)
```



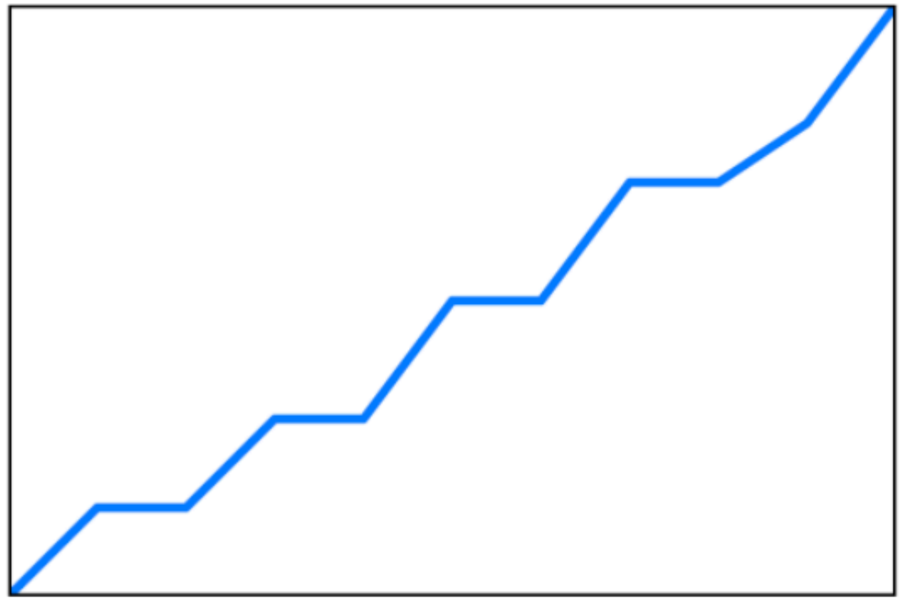
Data-driven drawing

```
context, size in

let points = [ … ]

let path = Path { p in
  p.addLines(points)
}

let dataSize = path.boundingRect.size

let transform = CGAffineTransform.identity
  .translatedBy(x: 0, y: size.height)
  .scaledBy(x: 1, y: -1)
  .scaledBy(x: size.width / dataSize.width, y: 1)


context.stroke(
  path.applying(transform),
  with: .color(.blue),
  style: StrokeStyle(lineWidth: 3)
)
```



Data-driven drawing

```
context, size in

let points = [ … ]

let path = Path { p in
  p.addLines(points)
}

let dataSize = path.boundingRect.size

let transform = CGAffineTransform.identity
  .translatedBy(x: 0, y: size.height)
  .scaledBy(x: 1, y: -1)
  .scaledBy(x: size.width / dataSize.width, y: 1)
  .scaledBy(x: 1, y: size.height / dataSize.height)

context.stroke(
  path.applying(transform),
  with: .color(.blue),
  style: StrokeStyle(lineWidth: 3)
)
```



I feel the needometer…

```
let radius = size.width * 0.5
context.translateBy(x: radius, y: radius)

for labelSpeed: Double in
  stride(from: 0, through: 150, by: 10) {
     var labelContext = context
     labelContext.rotate(by: angle(for: labelSpeed))

     labelContext.draw(
         Text(labelSpeed.formatted()),
         at: CGPoint(x: 0, y: -radius),
         anchor: .top)
}
```

I feel the needometer…

```
let radius = size.width * 0.5
context.translateBy(x: radius, y: radius)

for labelSpeed: Double in
  stride(from: 0, through: 150, by: 10) {
     var labelContext = context
     labelContext.rotate(by: angle(for: labelSpeed))

     labelContext.draw(
         Text(labelSpeed.formatted()),
         at: CGPoint(x: 0, y: -radius),
         anchor: .top)
}
```

I feel the needometer…

```
let radius = size.width * 0.5
context.translateBy(x: radius, y: radius)

for labelSpeed: Double in
  stride(from: 0, through: 150, by: 10) {
    var labelContext = context
    labelContext.rotate(by: angle(for: labelSpeed))

    labelContext.draw(
        Text(labelSpeed.formatted()),
        at: CGPoint(x: 0, y: -radius),
        anchor: .top)
}
```

I feel the needometer…

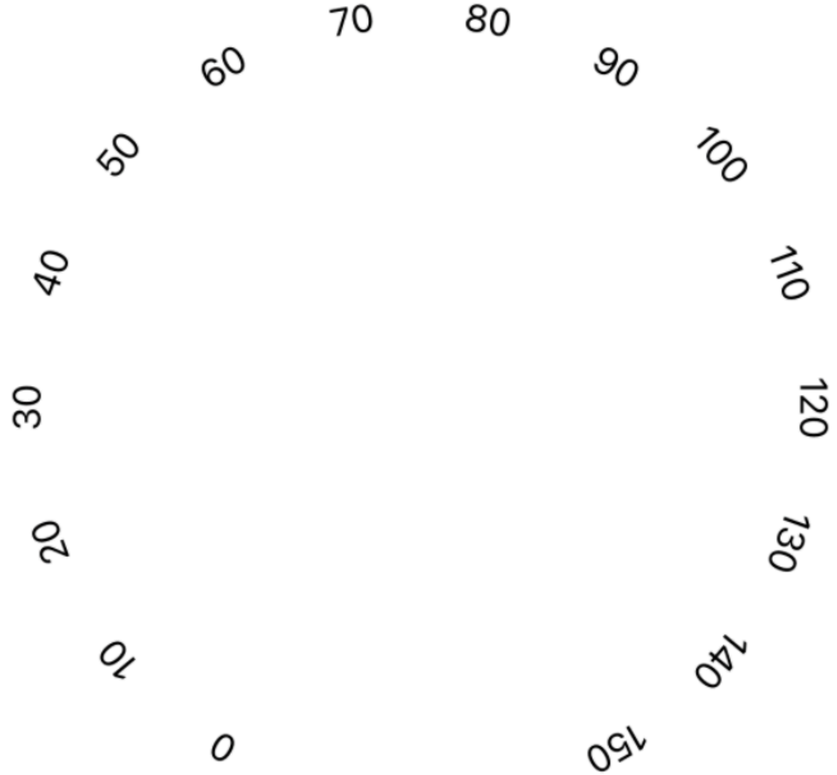
```
let radius = size.width * 0.5
context.translateBy(x: radius, y: radius)

for labelSpeed: Double in
  stride(from: 0, through: 150, by: 10) {
    var labelContext = context
    labelContext.rotate(by: angle(for: labelSpeed))

    labelContext.draw(
        Text(labelSpeed.formatted()),
        at: CGPoint(x: 0, y: -radius),
        anchor: .top)
}
```

I feel the needometer…

```
let radius = size.width * 0.5
context.translateBy(x: radius, y: radius)

for labelSpeed: Double in
  stride(from: 0, through: 150, by: 10) {
     var labelContext = context
     labelContext.rotate(by: angle(for: labelSpeed))

     labelContext.draw(
         Text(labelSpeed.formatted()),
         at: CGPoint(x: 0, y: -radius),
         anchor: .top)
}
```

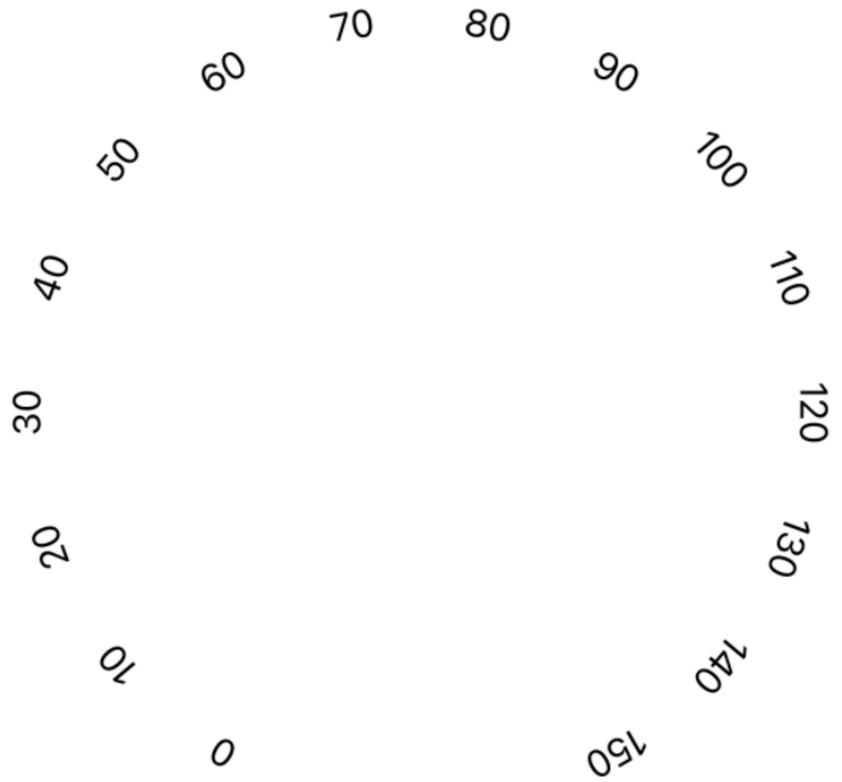
I feel the needometer…

```
let radius = size.width * 0.5
context.translateBy(x: radius, y: radius)

for labelSpeed: Double in
  stride(from: 0, through: 150, by: 10) {
     var labelContext = context
     labelContext.rotate(by: angle(for: labelSpeed)

     labelContext.draw(
         Text(labelSpeed.formatted()),
         at: CGPoint(x: 0, y: -radius),
         anchor: .top)
}
```



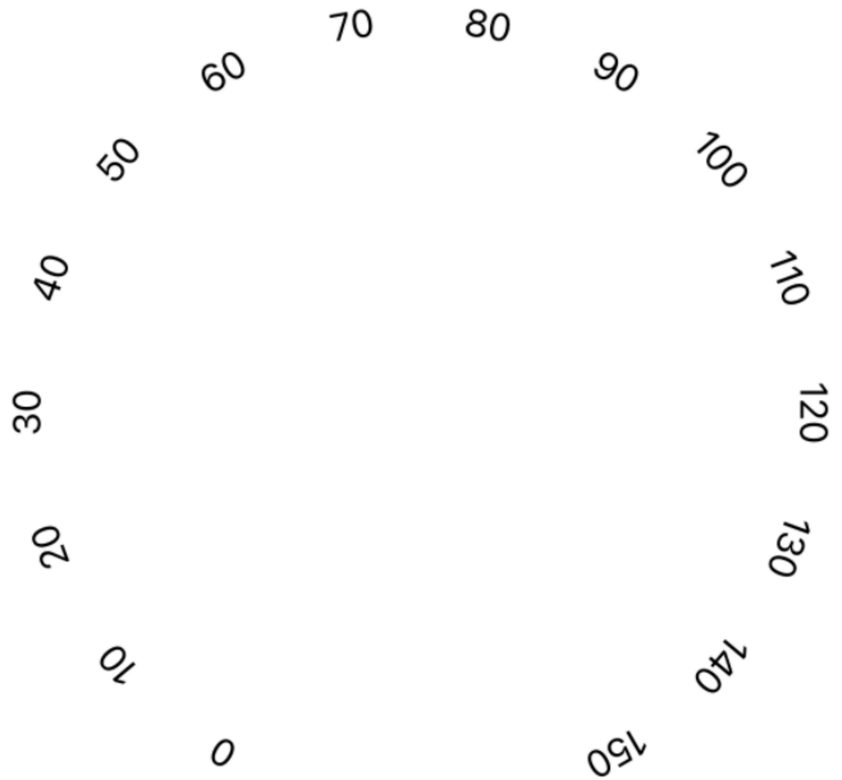
I feel the needometer…

```
var arcContext = context
arcContext.rotate(by: .degrees(-90))

let arc = Path { p in
    p.addArc(
      center: .zero,
      radius: radius - 30,
      startAngle: angle(for: 0),
      endAngle: angle(for: 150),
      clockwise: false)
}

let gradient = Gradient(colors: [
  .green, .yellow, .orange, .red, .red
])

let shading = GraphicsContext.Shading.conicGradient(
  gradient,
  center: .zero,
  angle: angle(for: 0))

arcContext.stroke(arc, with: shading, style: StrokeStyle(lineWidth: 10, dash: [2, 2]))
```



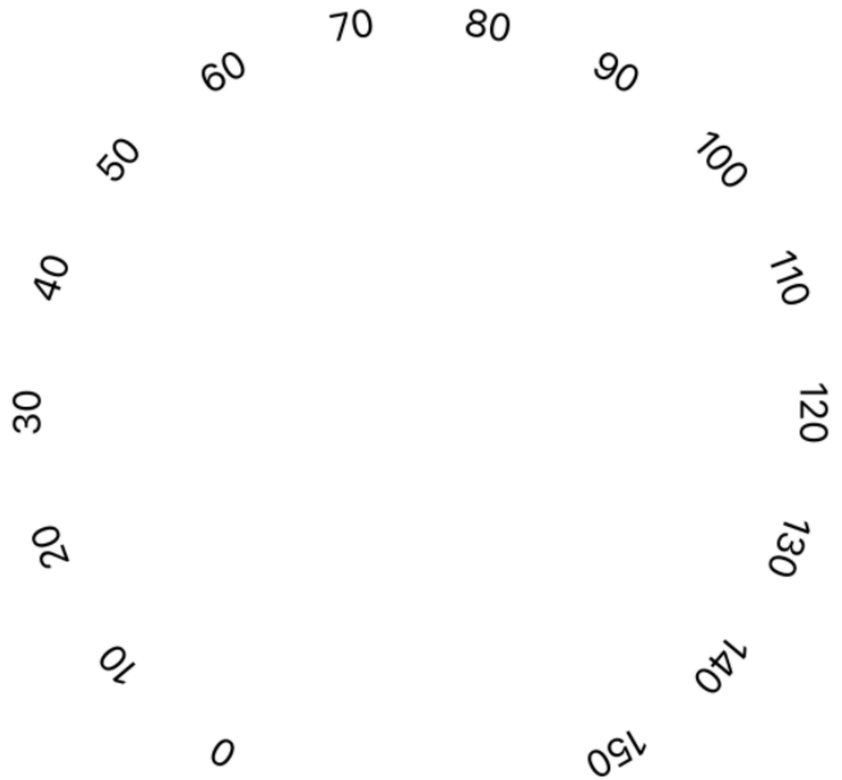
I feel the needometer…

```
var arcContext = context
arcContext.rotate(by: .degrees(-90))

let arc = Path { p in
    p.addArc(
      center: .zero,
      radius: radius - 30,
      startAngle: angle(for: 0),
      endAngle: angle(for: 150),
      clockwise: false)
}

let gradient = Gradient(colors: [
  .green, .yellow, .orange, .red, .red
])

let shading = GraphicsContext.Shading.conicGradient(
  gradient,
  center: .zero,
  angle: angle(for: 0))

arcContext.stroke(arc, with: shading, style: StrokeStyle(lineWidth: 10, dash: [2, 2]))
```



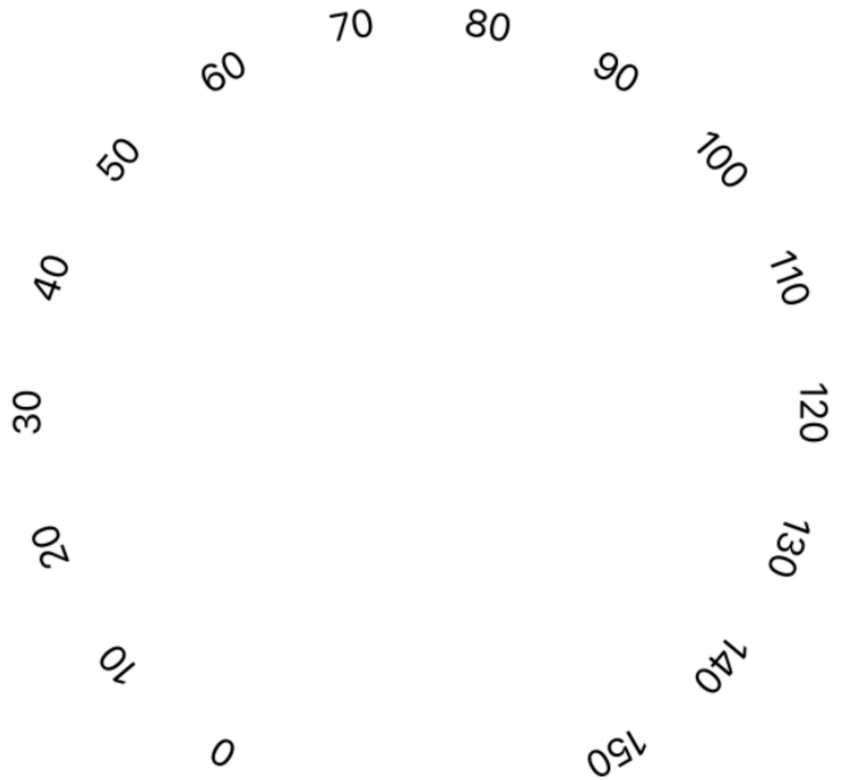
I feel the needometer…

```
var arcContext = context
arcContext.rotate(by: .degrees(-90))

let arc = Path { p in
    p.addArc(
      center: .zero,
      radius: radius - 30,
      startAngle: angle(for: 0),
      endAngle: angle(for: 150),
      clockwise: false)
}

let gradient = Gradient(colors: [
  .green, .yellow, .orange, .red, .red
])

let shading = GraphicsContext.Shading.conicGradient(
  gradient,
  center: .zero,
  angle: angle(for: 0))

arcContext.stroke(arc, with: shading, style: StrokeStyle(lineWidth: 10, dash: [2, 2]))
```



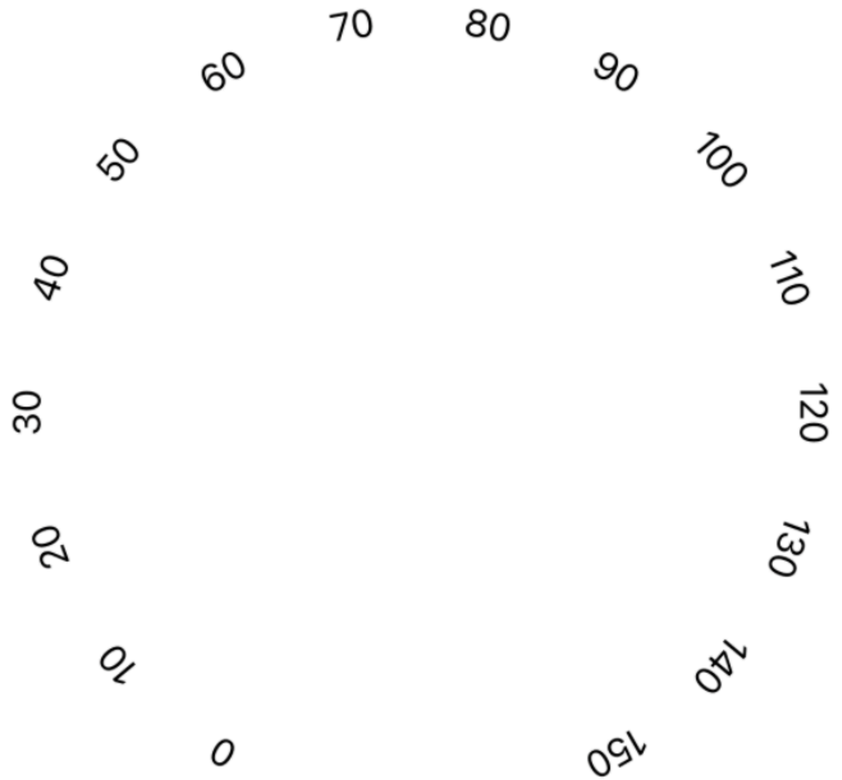
I feel the needometer…

```
var arcContext = context
arcContext.rotate(by: .degrees(-90))

let arc = Path { p in
    p.addArc(
      center: .zero,
      radius: radius - 30,
      startAngle: angle(for: 0),
      endAngle: angle(for: 150),
      clockwise: false)
}

let gradient = Gradient(colors: [
  .green, .yellow, .orange, .red, .red
])

let shading = GraphicsContext.Shading.conicGradient(
  gradient,
  center: .zero,
  angle: angle(for: 0))

arcContext.stroke(arc, with: shading, style: StrokeStyle(lineWidth: 10, dash: [2, 2]))
```



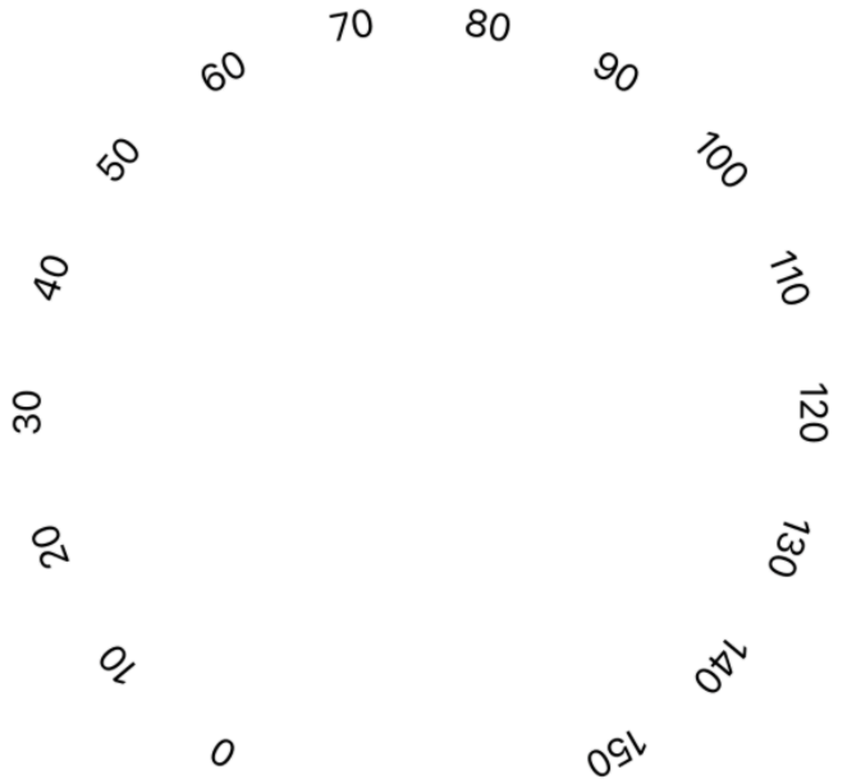
I feel the needometer…

```
var arcContext = context
arcContext.rotate(by: .degrees(-90))

let arc = Path { p in
    p.addArc(
      center: .zero,
      radius: radius - 30,
      startAngle: angle(for: 0),
      endAngle: angle(for: 150),
      clockwise: false)
}

let gradient = Gradient(colors: [
  .green, .yellow, .orange, .red, .red
])

let shading = GraphicsContext.Shading.conicGradient(
  gradient,
  center: .zero,
  angle: angle(for: 0))

arcContext.stroke(arc, with: shading, style: StrokeStyle(lineWidth: 10, dash: [2, 2]))
```



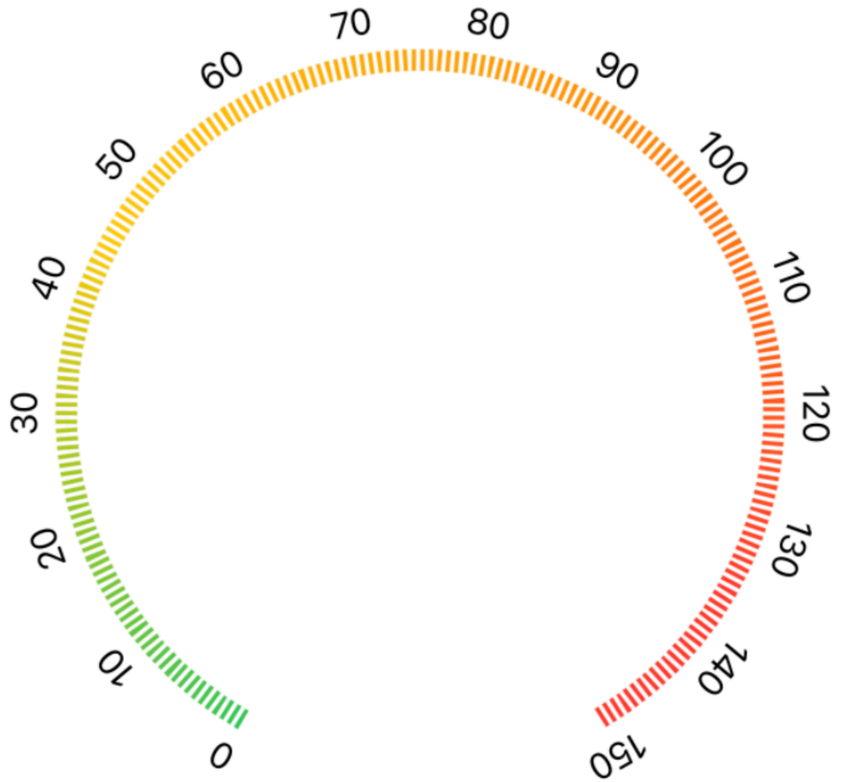
I feel the needometer…

```
var arcContext = context
arcContext.rotate(by: .degrees(-90))

let arc = Path { p in
    p.addArc(
      center: .zero,
      radius: radius - 30,
      startAngle: angle(for: 0),
      endAngle: angle(for: 150),
      clockwise: false)
}

let gradient = Gradient(colors: [
  .green, .yellow, .orange, .red, .red
])

let shading = GraphicsContext.Shading.conicGradient(
  gradient,
  center: .zero,
  angle: angle(for: 0))

arcContext.stroke(arc, with: shading, style: StrokeStyle(lineWidth: 10, dash: [2, 2]))
```



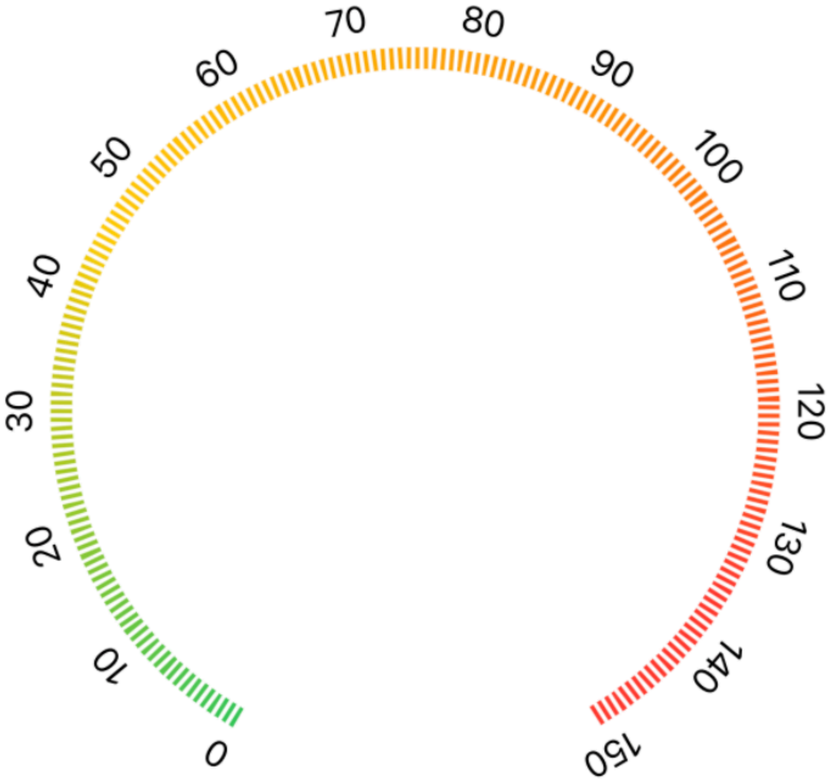
I feel the needometer…

```
var arcContext = context
arcContext.rotate(by: .degrees(-90))

let arc = Path { p in
    p.addArc(
      center: .zero,
      radius: radius - 30,
      startAngle: angle(for: 0),
      endAngle: angle(for: 150),
      clockwise: false)
}

let gradient = Gradient(colors: [
  .green, .yellow, .orange, .red, .red
])

let shading = GraphicsContext.Shading.conicGradient(
  gradient,
  center: .zero,
  angle: angle(for: 0))

arcContext.stroke(arc, with: shading, style: StrokeStyle(lineWidth: 10, dash: [2, 2]))
```



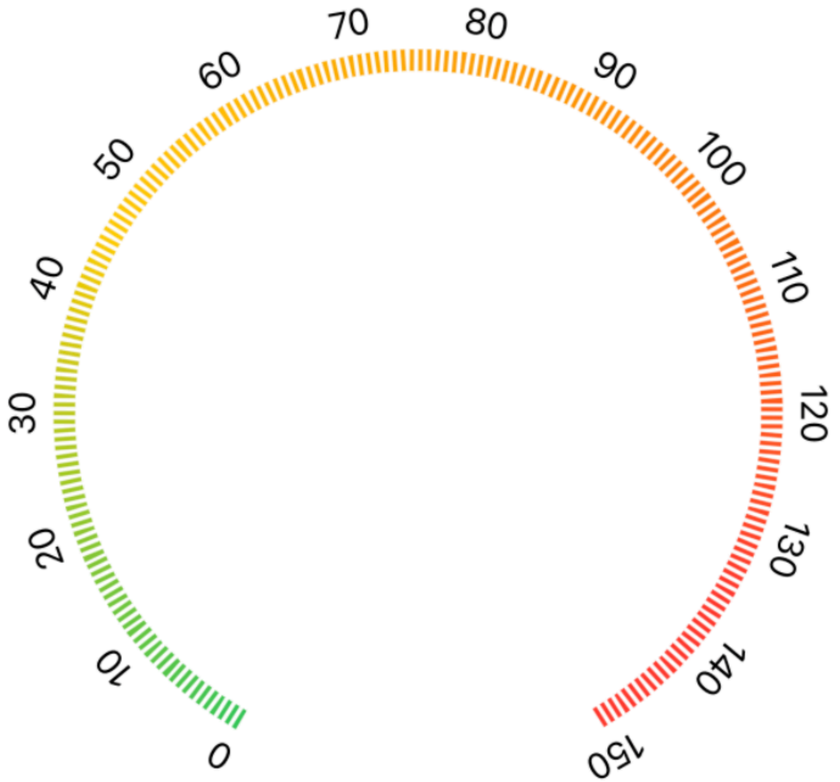
I feel the needometer…

```
let needle = Path { p in
    p.addArc(
      center: .zero,
      radius: 20,
      startAngle: .degrees(-50),
      endAngle: .degrees(230),
      clockwise: false)
    p.addLine(to: CGPoint(x: 0, y: -(radius - 40)))
    p.closeSubpath()
    p.addEllipse(in:
      CGRect(x: -10, y: -10, width: 20, height: 20)
    )
    p.closeSubpath()
}

var needleContext = context
needleContext.rotate(by: angle(for: speed))
needleContext.addFilter(.shadow(radius: 4))
needleContext.fill(needle, with: .color(.primary), style: .init(eoFill: true))
```



I feel the needometer…

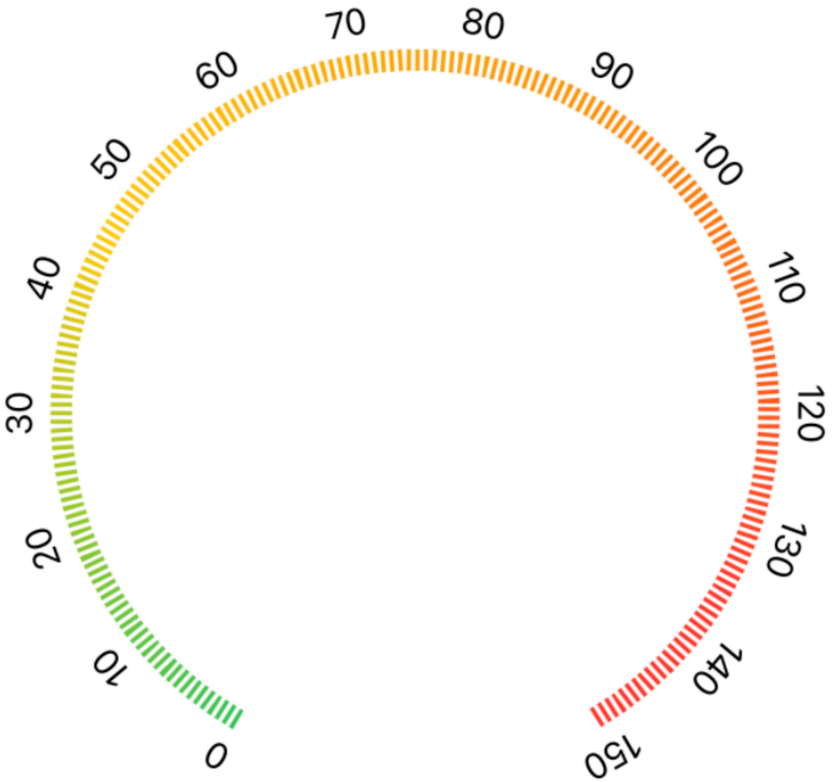
```
let needle = Path { p in
    p.addArc(
      center: .zero,
      radius: 20,
      startAngle: .degrees(-50),
      endAngle: .degrees(230),
      clockwise: false)
    p.addLine(to: CGPoint(x: 0, y: -(radius - 40)))
    p.closeSubpath()
    p.addEllipse(in:
      CGRect(x: -10, y: -10, width: 20, height: 20)
    )
    p.closeSubpath()
}
```




```
var needleContext = context
needleContext.rotate(by: angle(for: speed))
needleContext.addFilter(.shadow(radius: 4))
needleContext.fill(needle, with: .color(.primary), style: .init(eoFill: true))
```

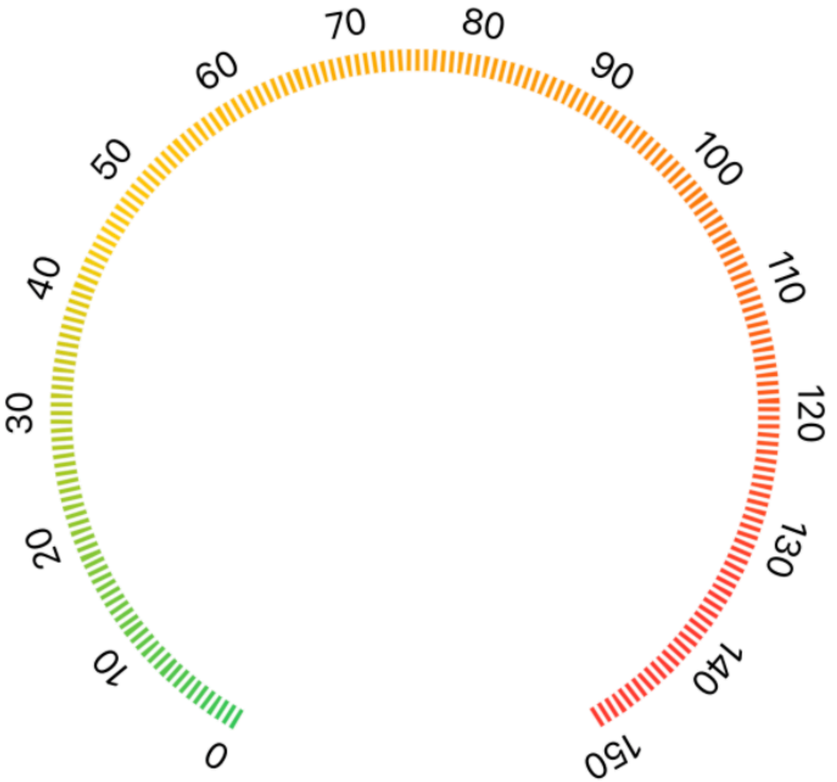
I feel the needometer…

```
let needle = Path { p in
    p.addArc(
      center: .zero,
      radius: 20,
      startAngle: .degrees(-50),
      endAngle: .degrees(230),
      clockwise: false)
    p.addLine(to: CGPoint(x: 0, y: -(radius - 40)))
    p.closeSubpath()
    p.addEllipse(in:
      CGRect(x: -10, y: -10, width: 20, height: 20)
    )
    p.closeSubpath()
}

var needleContext = context
needleContext.rotate(by: angle(for: speed))
needleContext.addFilter(.shadow(radius: 4))
needleContext.fill(needle, with: .color(.primary), style: .init(eoFill: true))
```



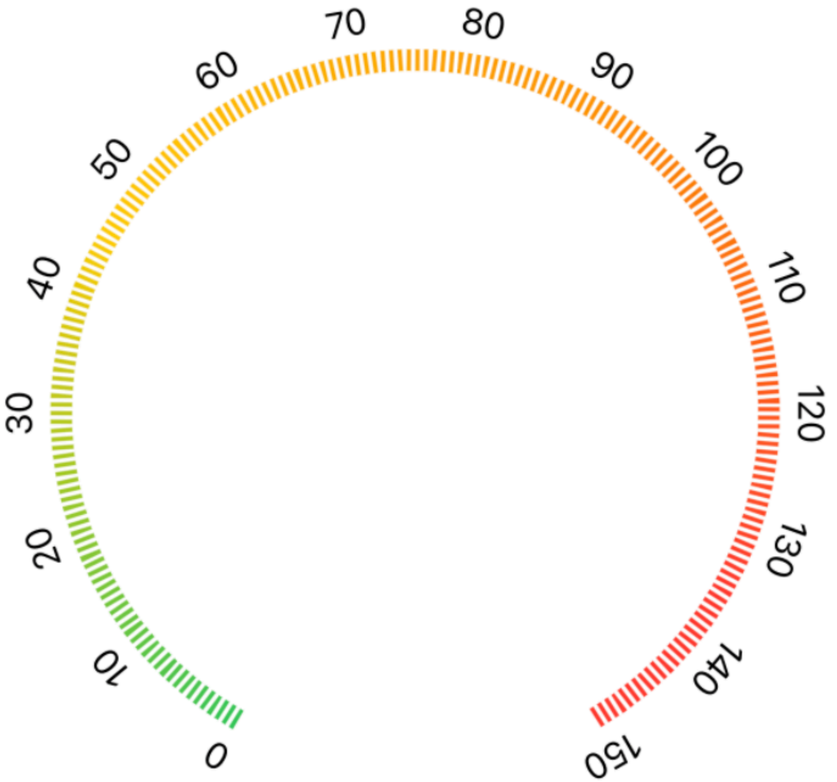
I feel the needometer…

```
let needle = Path { p in
    p.addArc(
      center: .zero,
      radius: 20,
      startAngle: .degrees(-50),
      endAngle: .degrees(230),
      clockwise: false)
    p.addLine(to: CGPoint(x: 0, y: -(radius - 40)))
    p.closeSubpath()
    p.addEllipse(in:
      CGRect(x: -10, y: -10, width: 20, height: 20)
    )
    p.closeSubpath()
}

var needleContext = context
needleContext.rotate(by: angle(for: speed))
needleContext.addFilter(.shadow(radius: 4))
needleContext.fill(needle, with: .color(.primary), style: .init(eoFill: true))
```



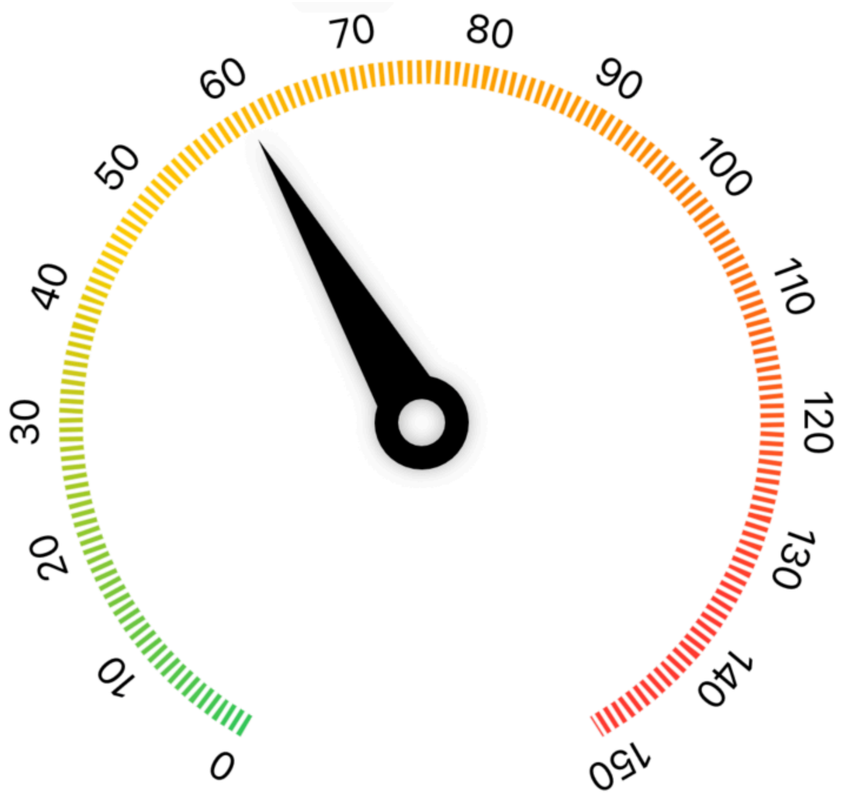
I feel the needometer…

```
let needle = Path { p in
    p.addArc(
      center: .zero,
      radius: 20,
      startAngle: .degrees(-50),
      endAngle: .degrees(230),
      clockwise: false)
    p.addLine(to: CGPoint(x: 0, y: -(radius - 40)))
    p.closeSubpath()
    p.addEllipse(in:
      CGRect(x: -10, y: -10, width: 20, height: 20)
    )
    p.closeSubpath()
}

var needleContext = context
needleContext.rotate(by: angle(for: speed))
needleContext.addFilter(.shadow(radius: 4))
needleContext.fill(needle, with: .color(.primary), style: .init(eoFill: true))
```



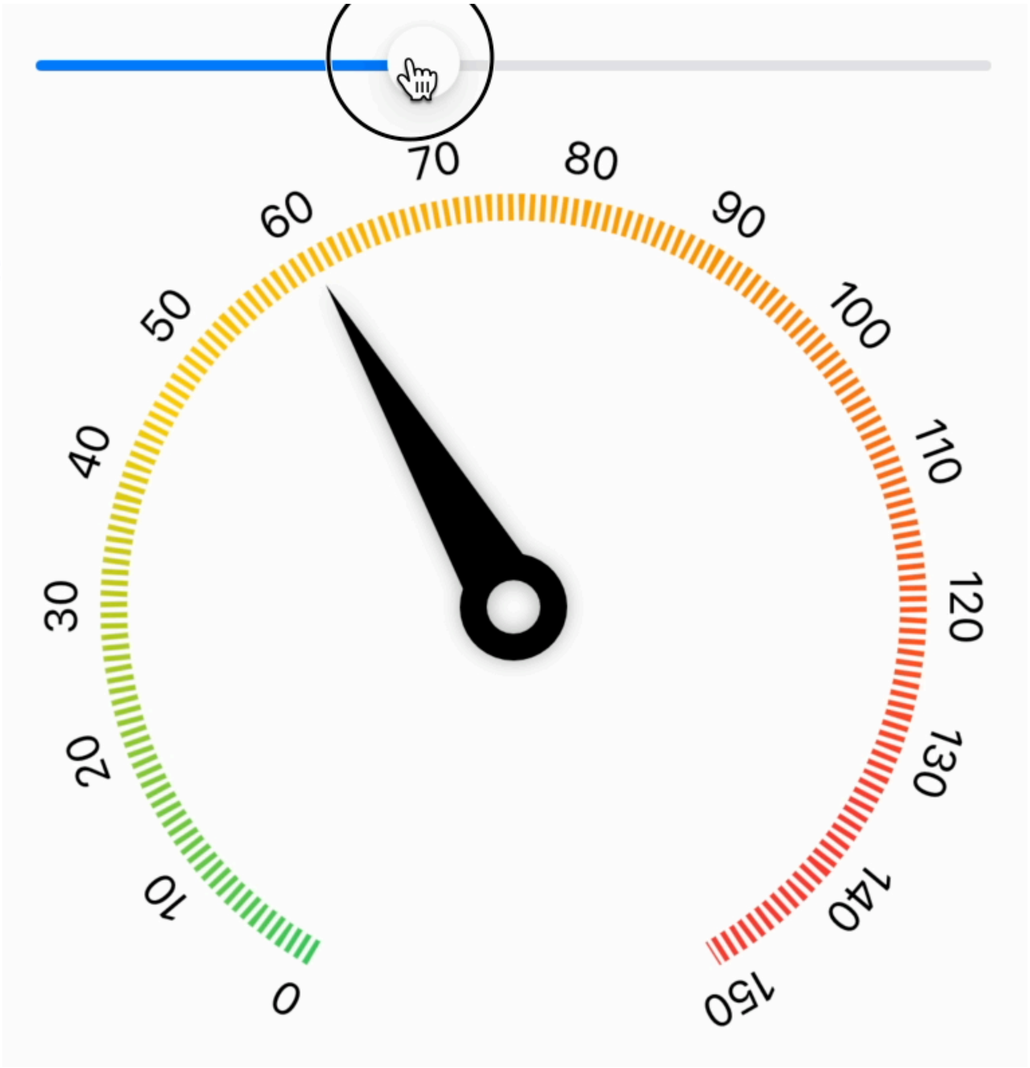
I feel the needometer…

```
let needle = Path { p in
    p.addArc(
      center: .zero,
      radius: 20,
      startAngle: .degrees(-50),
      endAngle: .degrees(230),
      clockwise: false)
    p.addLine(to: CGPoint(x: 0, y: -(radius - 40)))
    p.closeSubpath()
    p.addEllipse(in:
      CGRect(x: -10, y: -10, width: 20, height: 20)
    )
    p.closeSubpath()
}

var needleContext = context
needleContext.rotate(by: angle(for: speed))
needleContext.addFilter(.shadow(radius: 4))
needleContext.fill(needle, with: .color(.primary), style: .init(eoFill: true))
```



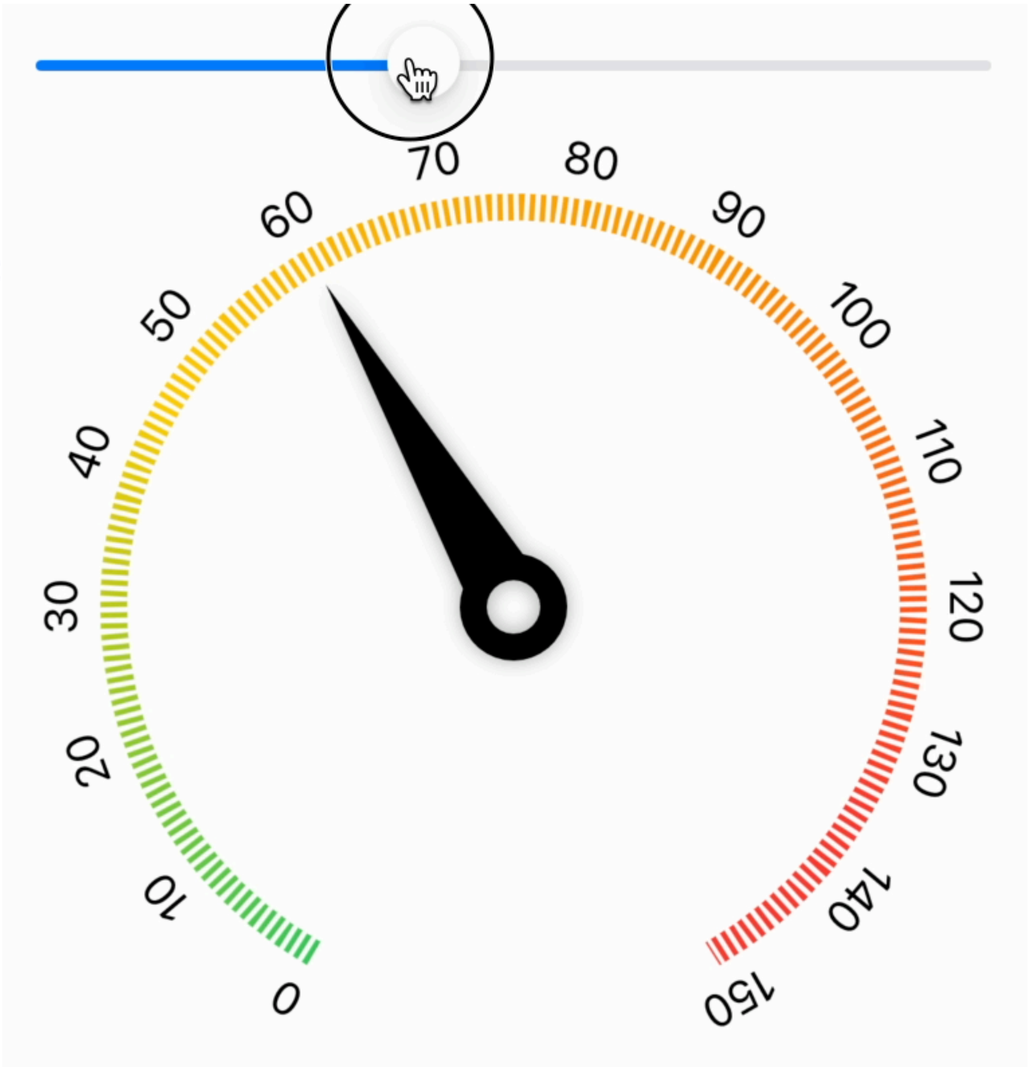
Moving the needle

```
@State var speed: Double = 60
var body: some View {
    VStack {
        Slider(value: $speed, in: 0...150
        Speedometer(speed: speed)
    }
}
```



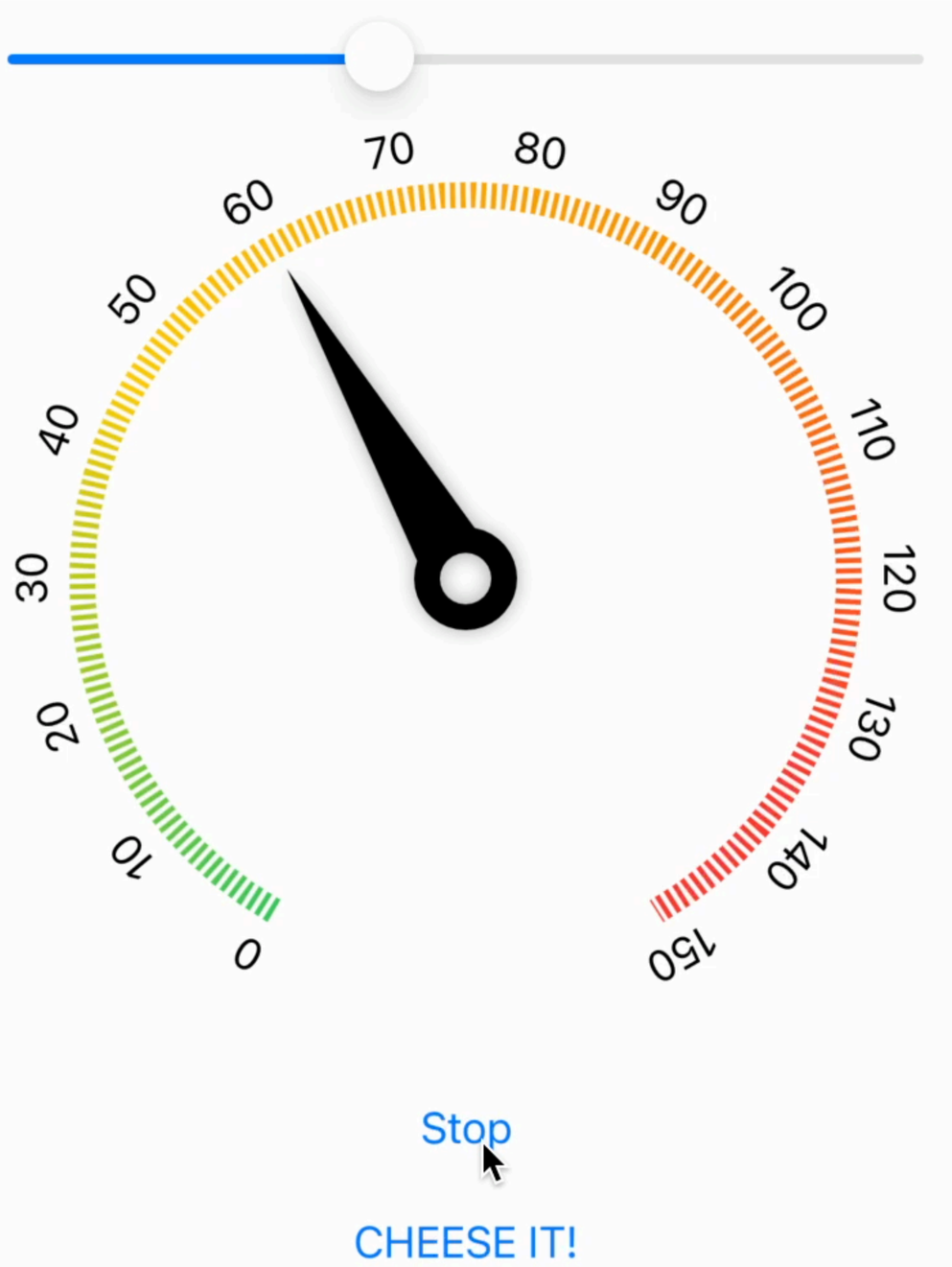
Moving the needle

```
@State var speed: Double = 60
var body: some View {
    VStack {
        Slider(value: $speed, in: 0...150
        Speedometer(speed: speed)
    }
}
```



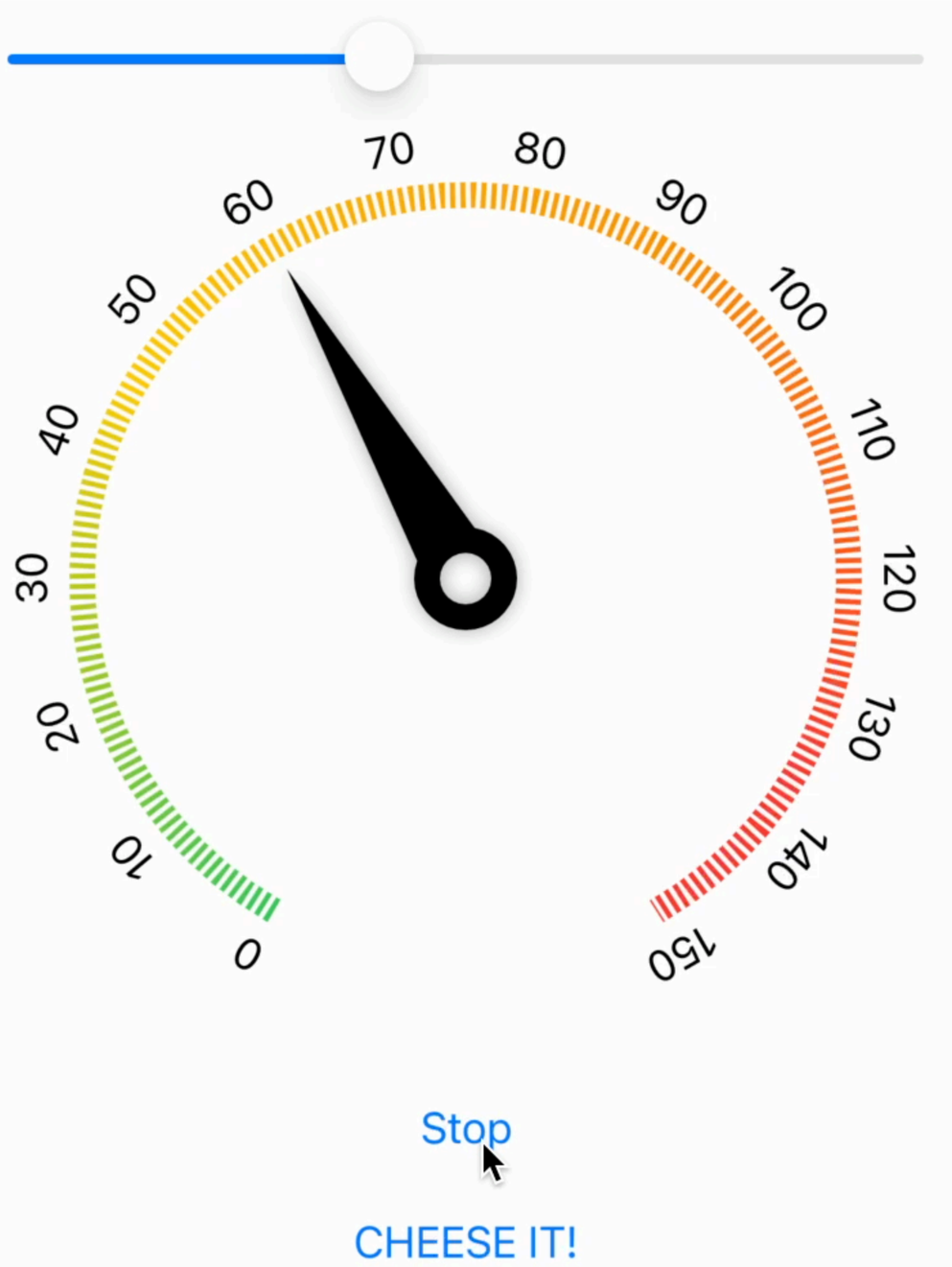
Moving the needle

```
Button("Stop") {
    withAnimation {
        speed = 0
    }
}

Button("CHEESE IT!") {
    withAnimation {
        speed = 150
    }
}
```



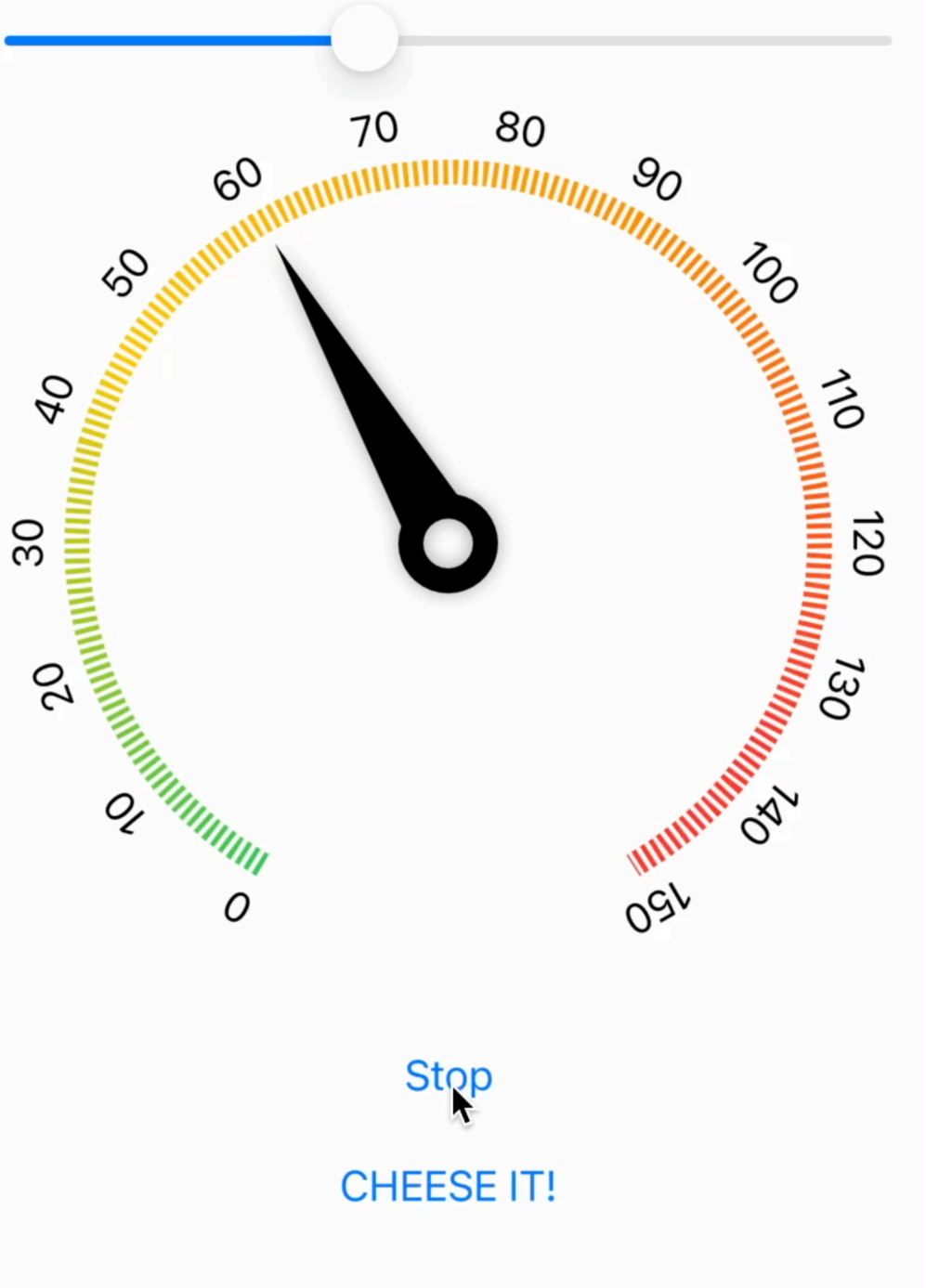
Moving the needle

```
Button("Stop") {
    withAnimation {
        speed = 0
    }
}

Button("CHEESE IT!") {
    withAnimation {
        speed = 150
    }
}
```



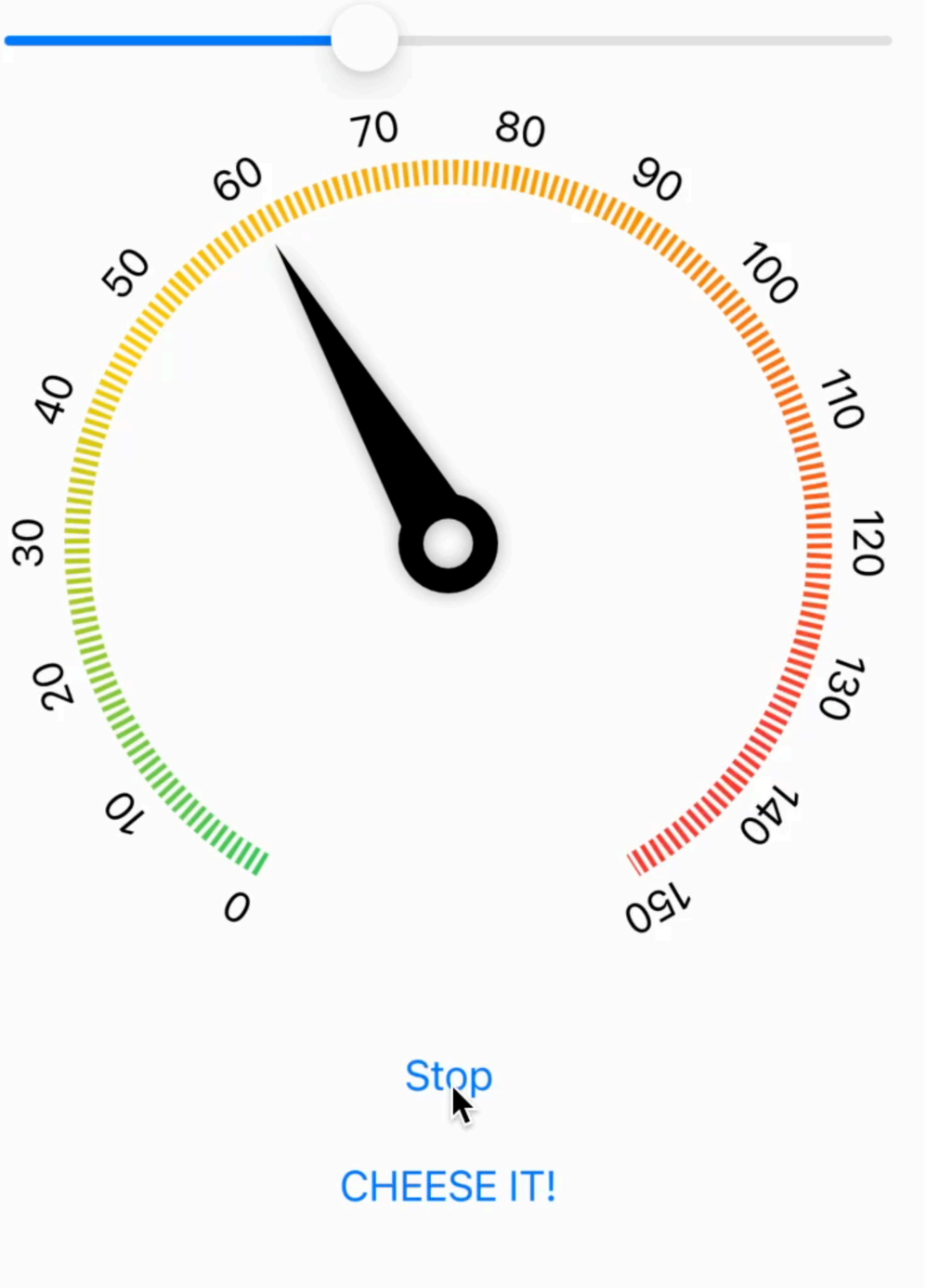
Moving the needle

```
struct Speedometer: View, Animatable {

  var animatableData: Double {
    get { speed }
    set { speed = newValue }
  }

  …

}
```



Moving the needle

```
struct Speedometer: View, Animatable {

  var animatableData: Double {
    get { speed }
    set { speed = newValue }
  }

  …

}
```



SwiftUI
Canvas
Transforms
Data Visualisation

Thank you for listening!