OZONtech

# Анатомия LazyVStack

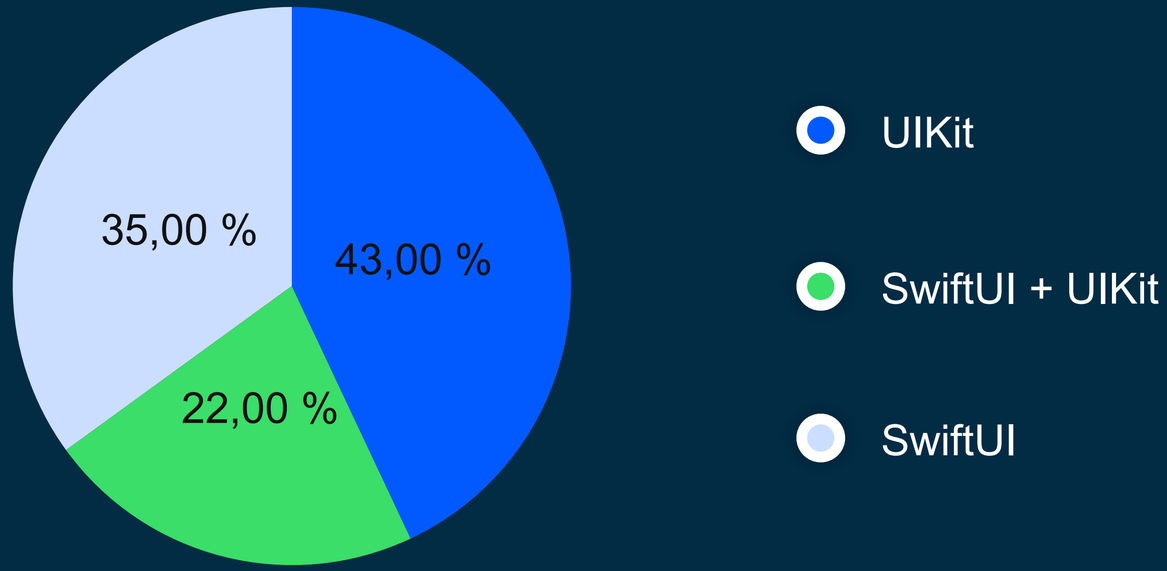
Проблемы, скрытые возможности, оптимизации

**Алексей Таран**
iOS TechLead Ozon Seller

# SwiftUI vs UIKit Usage

## Процент разработчиков, использующих инструмент

Конец 2025




## SwiftUI Adoption

**57%** используют SwiftUI

**43%** не используют SwiftUI

Источник: https://rentamac.io/ios-app-development-statistics/

# Вопросы к SwiftUI 

Много «магии»
под капотом

Ограничения
кастомизации
стандартных компонентов

@State-management

Слабый контроль
над Layout/Rendering

Сложно дебажить

Обёртки над UIKit
«съедают» перфоманс

# Вопросы к SwiftUI

Много «магии»
под капотом

Ограничения
кастомизации
стандартных компонентов

@State-management

Слабый контроль
над Layout/Rendering

Сложно дебажить

Обёртки над UIKit
«съедают» перфоманс

@alextar04

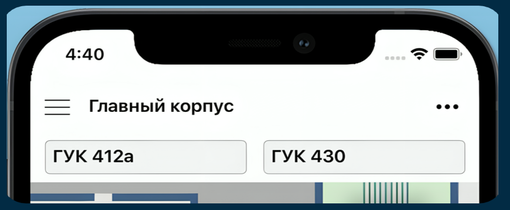
# Алексей Таран

iOS TechLead Ozon Seller

**FinTech**

ВТБ

**Indoor geolocation & maps**




**AR Plants Care**


**OzonSeller App**

# Mobius 2024 Autumn


Non-Copyable intro

# Let’s GO!

# Кто писал такой код?

```
ConditionalScrollView {
    LazyVStack(spacing: 16) {
        Text("Текст 1")
            .frame(height: 20)

        Text("Текст 2")
            .frame(height: 30)

        Text("Текст 3")
            .frame(height: 40)

        Text("Текст 4")
            .frame(height: 500)

        Text("Текст 5")
            .frame(height: 400)
    }
    .padding()
}
```

```
struct ConditionalScrollView<Content: View>: View {
    @State private var contentHeight: CGFloat = 0

    var body: some View {
        GeometryReader { geo in
            if contentHeight > geo.size.height {
                ScrollView {
                    measuredContent
                }
            } else {
                measuredContent
            }
        }
    }

    private var measuredContent: some View {
        content
            .background(
                GeometryReader { geo in
                    Color.clear
                        .preference(
                            key: ContentHeightKey.self,
                            value: geo.size.height
                        )
                }
            )
            .onPreferenceChange(ContentHeightKey.self) {
                contentHeight = $0
            }
    }
}
```

ozon tech

# Какой размер у contentHeight?

```
struct ConditionalScrollView<Content: View>: View {
    @State private var contentHeight: CGFloat = 0

    var body: some View {
        GeometryReader { geo in
            if contentHeight > geo.size.height {
                ScrollView {
                    measuredContent
                }
            } else {
                measuredContent
            }
        }
    }

    private var measuredContent: some View {
        content
            .background(
                GeometryReader { geo in
                    Color.clear
                        .preference(
                            key: ContentHeightKey.self,
                            value: geo.size.height
                        )
                }
            )
            .onPreferenceChange(ContentHeightKey.self) {
                contentHeight = $0
            }
    }
}
```

**О контенте**

- Высоты Text: *20, 30, 40, 500, 400*
- Spacing: *16*
- Padding: *есть*

# Какой размер у contentHeight?

```swift
struct ConditionalScrollView<Content: View>: View {
    @State private var contentHeight: CGFloat = 0

    var body: some View {
        GeometryReader { geo in
            if contentHeight > geo.size.height {
                ScrollView {
                    measuredContent
                }
            } else {
                measuredContent
            }
        }
    }

    private var measuredContent: some View {
        content
            .background(
                GeometryReader { geo in
                    Color.clear
                        .preference(
                            key: ContentHeightKey.self,
                            value: geo.size.height
                        )
                }
            )
            .onPreferenceChange(ContentHeightKey.self) {
                contentHeight = $0
            }
    }
}
```

**О контенте**

- Высоты Text: *20, 30, 40, 500, 400*
- Spacing: *16*
- Padding: *есть*

**Значения contentHeight**

```
Высота: 0.0
Высота: 221.0
Высота: 1086.0
Высота: 221.0
Высота: 1086.0
```

# Сколько раз был пересчитан body?

```swift
struct ConditionalScrollView<Content: View>: View {
    @State private var contentHeight: CGFloat = 0

    var body: some View {
        GeometryReader { geo in
            if contentHeight > geo.size.height {
                ScrollView {
                    measuredContent
                }
            } else {
                measuredContent
            }
        }
    }

    private var measuredContent: some View {
        content
            .background(
                GeometryReader { geo in
                    Color.clear
                        .preference(
                            key: ContentHeightKey.self,
                            value: geo.size.height
                        )
                }
            )
            .onPreferenceChange(ContentHeightKey.self) {
                contentHeight = $0
            }
    }
}
```

**О контенте**

- Высоты Text: *20, 30, 40, 500, 400*
- Spacing: *16*
- Padding: *есть*

# Сколько раз был пересчитан body?

```swift
struct ConditionalScrollView<Content: View>: View {
    @State private var contentHeight: CGFloat = 0

    var body: some View {
        GeometryReader { geo in
            if contentHeight > geo.size.height {
                ScrollView {
                    measuredContent
                }
            } else {
                measuredContent
            }
        }
    }

    private var measuredContent: some View {
        content
            .background(
                GeometryReader { geo in
                    Color.clear
                        .preference(
                            key: ContentHeightKey.self,
                            value: geo.size.height
                        )
                }
            )
            .onPreferenceChange(ContentHeightKey.self) {
                contentHeight = $0
            }
    }
}
```

## О контенте

- Высоты Text: *20, 30, 40, 500, 400*
- Spacing: *16*
- Padding: *есть*

| Module / View Type | Count | Total Durati... | Min Duration | Avg Duration | Max Durati... | Std Dev Duration |
|---|---|---|---|---|---|---|
| * All * | 13 | 451.00 µs | 2.46 µs | 34.69 µs | 223.25 µs | 61.23 µs |
| SwiftUI | 7 | 204.67 µs | 3.46 µs | 29.24 µs | 87.08 µs | 27.98 µs |
| [redacted] | 6 | 246.33 µs | 2.46 µs | 41.06 µs | 223.25 µs | 89.27 µs |
| ConditionalScrollView<ModifiedContent<LazyVStac... | 5 | 23.08 µs | 2.46 µs | 4.62 µs | 6.83 µs | 1.62 µs |
| TestConditionalScrollView | 1 | 223.25 µs | 223.25 µs | 223.25 µs | 223.25 µs | – |

- Кол-во расчётов body: 5

# Изменения активной ветки

```swift
struct ConditionalScrollView<Content: View>: View {
    @State private var contentHeight: CGFloat = 0

    var body: some View {
        GeometryReader { geo in
            if contentHeight > geo.size.height {
                ScrollView {
                    measuredContent
                }
            } else {
                measuredContent
            }
        }
    }

    private var measuredContent: some View {
        content
            .background(
                GeometryReader { geo in
                    Color.clear
                        .preference(
                            key: ContentHeightKey.self,
                            value: geo.size.height
                        )
                }
            )
            .onPreferenceChange(ContentHeightKey.self) {
                contentHeight = $0
            }
    }
}
```

**О контенте**

- Высоты Text: *20, 30, 40, 500, 400*
- Spacing: *16*
- Padding: *есть*

```
Высота: 0.0
Высота: 221.0
Высота: 1086.0
Высота: 221.0
Высота: 1086.0
```

```
Ветка: без ScrollView
Ветка: без ScrollView
Ветка: ScrollView
Ветка: без ScrollView
Ветка: ScrollView
```

# Настоящий размер контента

```swift
struct ConditionalScrollView<Content: View>: View {
    @State private var contentHeight: CGFloat = 0

    var body: some View {
        GeometryReader { geo in
            if contentHeight > geo.size.height {
                ScrollView {
                    measuredContent
                }
            } else {
                measuredContent
            }
        }
    }

    private var measuredContent: some View {
        content
            .background(
                GeometryReader { geo in
                    Color.clear
                        .preference(
                            key: ContentHeightKey.self,
                            value: geo.size.height
                        )
                }
            )
            .onPreferenceChange(ContentHeightKey.self) {
                contentHeight = $0
            }
    }
}
```

## О контенте

- Высоты Text: *20, 30, 40, 500, 400*
- Spacing: *16*
- Padding: *есть*

## Считаем реальную высоту

n: количество элементов = 5

Высоты ячеек +
Spacing * (n - 1) +
Padding.height (top + bottom) = 1086.0

# Настоящий размер контента

```swift
struct ConditionalScrollView<Content: View>: View {
    @State private var contentHeight: CGFloat = 0

    var body: some View {
        GeometryReader { geo in
            if contentHeight > geo.size.height {
                ScrollView {
                    measuredContent
                }
            } else {
                measuredContent
            }
        }
    }

    private var measuredContent: some View {
        content
            .background(
                GeometryReader { geo in
                    Color.clear
                        .preference(
                            key: ContentHeightKey.self,
                            value: geo.size.height
                        )
                }
            )
            .onPreferenceChange(ContentHeightKey.self) {
                contentHeight = $0
            }
    }
}
```

## О контенте

- Высоты Text: *20, 30, 40, 500, 400*
- Spacing: *16*
- Padding: *есть*

## Считаем реальную высоту

n: количество элементов = 5

Высоты ячеек +
Spacing * (n - 1) +
Padding.height (top + bottom) = 1086.0

# Гипотеза

GeometryReader неверно считает размер контента и это приводит к цепочке проблем

# Заменяем LazyVStack → VStack

```
var body: some View {
    ConditionalScrollView {
        VStack(spacing: 16) {
            Text("Текст 1")
                .frame(height: 20)

            Text("Текст 2")
                .frame(height: 30)

            Text("Текст 3")
                .frame(height: 40)

            Text("Текст 4")
                .frame(height: 500)

            Text("Текст 5")
                .frame(height: 400)
        }
        .padding()
    }
}
```

# Заменяем LazyVStack → VStack

```swift
var body: some View {
    ConditionalScrollView {
        VStack(spacing: 16) {
            Text("Текст 1")
                .frame(height: 20)

            Text("Текст 2")
                .frame(height: 30)

            Text("Текст 3")
                .frame(height: 40)

            Text("Текст 4")
                .frame(height: 500)

            Text("Текст 5")
                .frame(height: 400)
        }
        .padding()
    }
}
```

Размер
контента

```
Высота: 0.0
Высота: 1086.0
```

Количество смен веток
«со ScrollView / без ScrollView»

```
Ветка: без ScrollView
Ветка: ScrollView
```

Количество пересчётов «body» у ConditionalScrollView

| Module / View Type | Count | Total Durati... |
|---|---|---|
| * All * | 16 | 356.71 µs |
| SwiftUI | 12 | 223.96 µs |
| [redacted] | 4 | 132.75 µs |
| ConditionalScrollView<ModifiedContent<VStack<Tu... | 2 | 16.79 µs |

# Вопросы

# Вопросы к GeometryReader + LazyVStack

Почему GeometryReader возвращает **неверный** размер для LazyVStack?

Что за числа возвращал GeometryReader?

Почему нет «моргающего» экрана при вычислении неверного размера?

Как понять, что размер **«стабилизировался»**?

# Поиск источника магии

## О зависимостях SwiftUI


# Зависимости SwiftUI

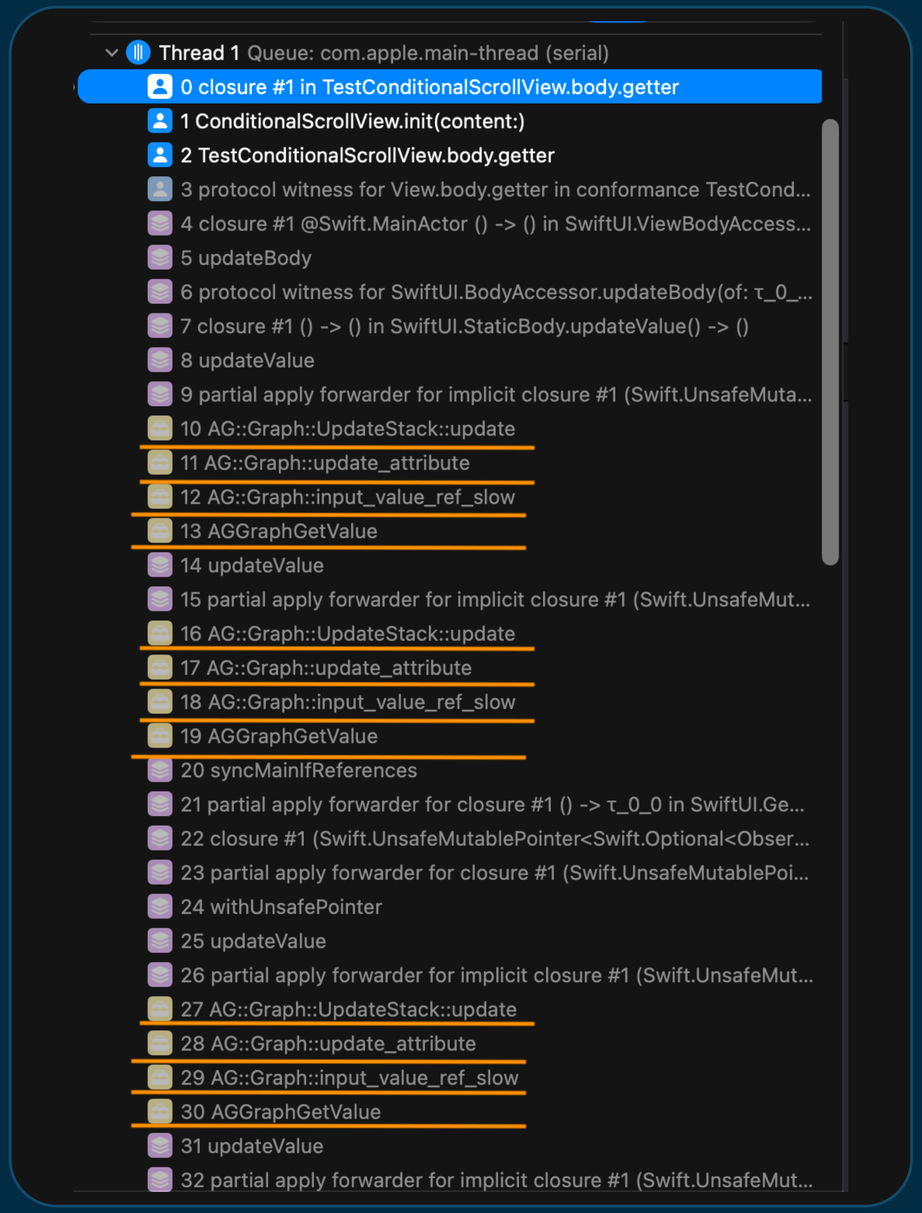
```
otool -L /Library/Developer/…/SwiftUI.framework/SwiftUI
```

| Публичные | Непубличные |
| --- | --- |
| SwiftUICore | CoreUI |
| UIKit | RenderBox |
| CoreGraphics | DesignLibrary |
| CoreText | CoreMaterial |
| ImageIO | AttributeGraph |
| … | … |

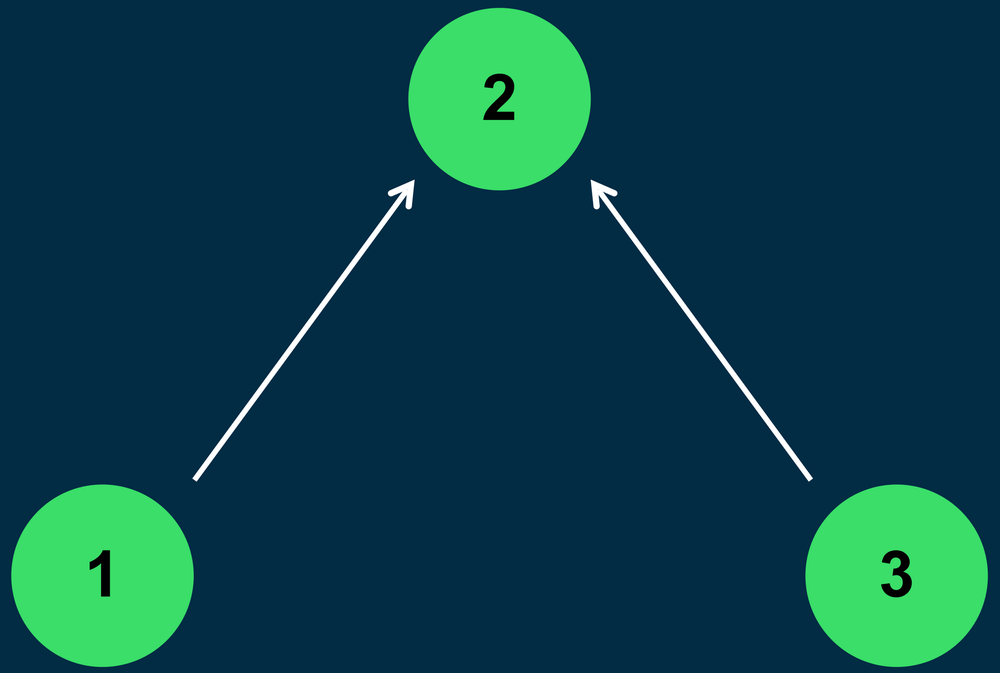
# Зависимости SwiftUI

```
otool -L /Library/Developer/…/SwiftUI.framework/SwiftUI
```

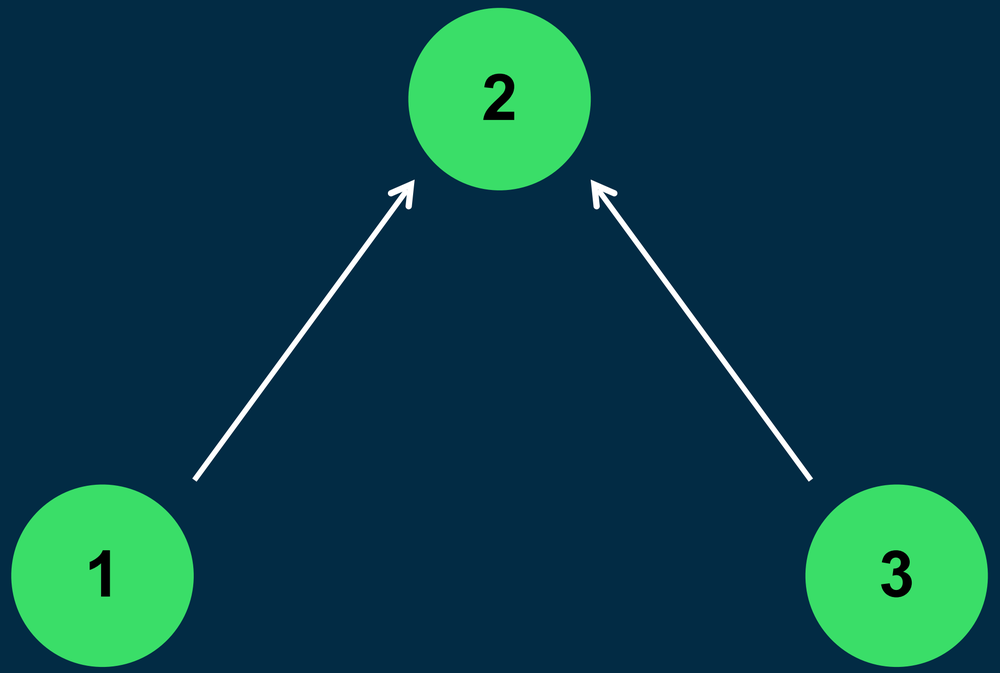
| Публичные | Непубличные |
| --- | --- |
| SwiftUICore | CoreUI |
| UIKit | RenderBox |
| CoreGraphics | DesignLibrary |
| CoreText | CoreMaterial |
| ImageIO | AttributeGraph |
| … | … |



# Что такое «граф»?



# Что такое «граф»?




Нужно искать триггеры вызова нашего кода

Ищем такой визуальный граф для нашего примера :)

# Ищем граф приложения под управлением AttributeGraph

# Как найти граф?

## Шаг 1

- Читаем «базу» по AttributeGraph от Rens Breur

# Как найти граф?

## Шаг 1

- Читаем «базу» по AttributeGraph от Rens Breur

## Шаг 2

- Собираем демо-проект

# Как найти граф?

## Шаг 1

- Читаем «базу» по AttributeGraph от Rens Breur

## Шаг 2

- Собираем демо-проект

## Шаг 3

**We can get such a graph by stealing**

# Как «украсть» граф?

# Этап 1. Ищем pointer

**Шаг 1**

Поиск pointer’а на граф

**Шаг 2**

Скармливаем pointer в функцию для вывода графа:

AG::Graph::print()

# Этап 1. Ищем pointer

```
8  final class A {
9      func someFunc() {
10         print(self)
11     }
12 }
13
14 func run() {
15     let aInstance = A()
16     aInstance.someFunc()
17 }
```

# Этап 1. Ищем pointer

```
8  final class A {
9      func someFunc() {
10         print(self)
11     }
12 }
13
14 func run() {
15     let aInstance = A()
16     aInstance.someFunc()
17 }
```

Как видит функцию компилятор

```
func someFunc(self: A Instance) {
    print(self)
}
```

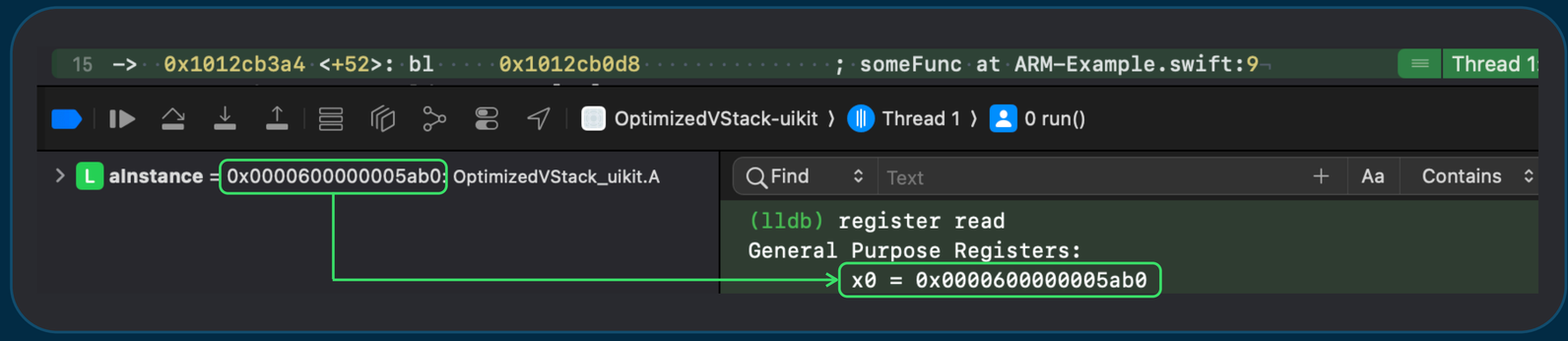
# Этап 1. Ищем pointer

```
8  final class A {
9      func someFunc() {
10         print(self)
11     }
12 }
13
14 func run() {
15     let aInstance = A()
16     aInstance.someFunc()
17 }
```

Как видит функцию компилятор

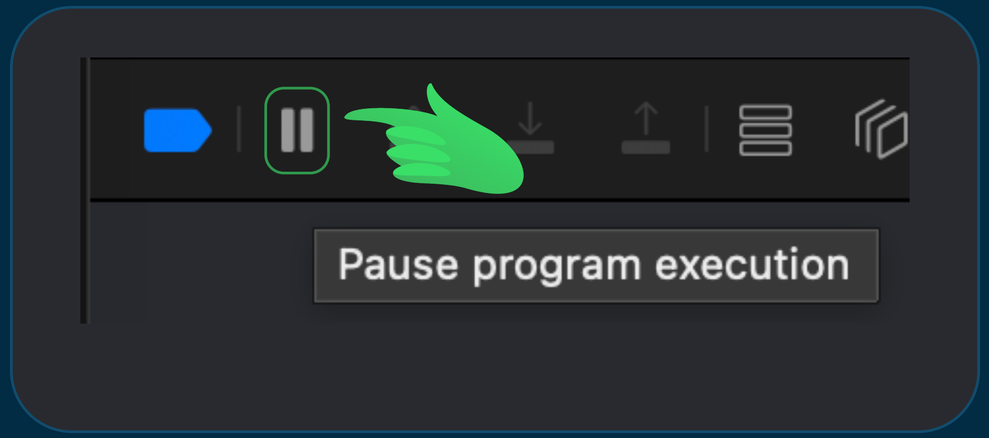
```
func someFunc(self: A Instance) {
    print(self)
}
```

В ARM64 параметры someFunc передаются через регистры CPU

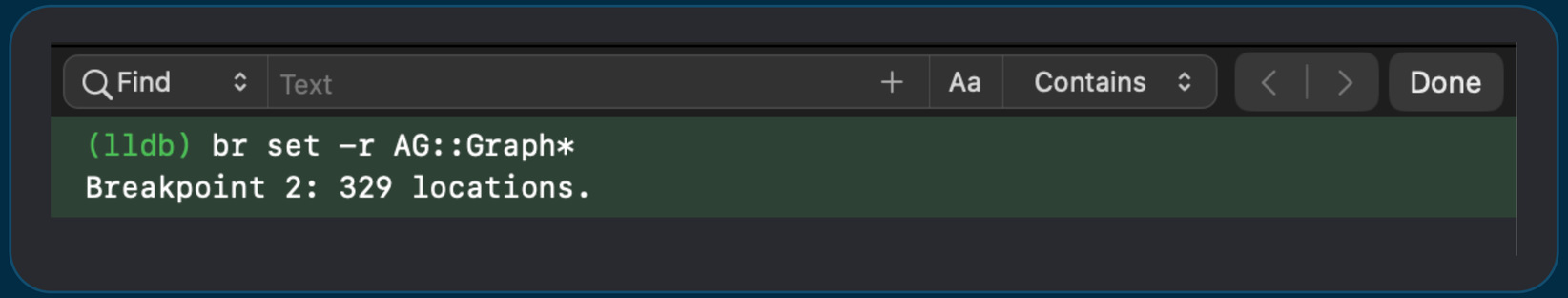


# Этап 1. Ищем pointer

Ставим приложение с «отображенным списком» на паузу

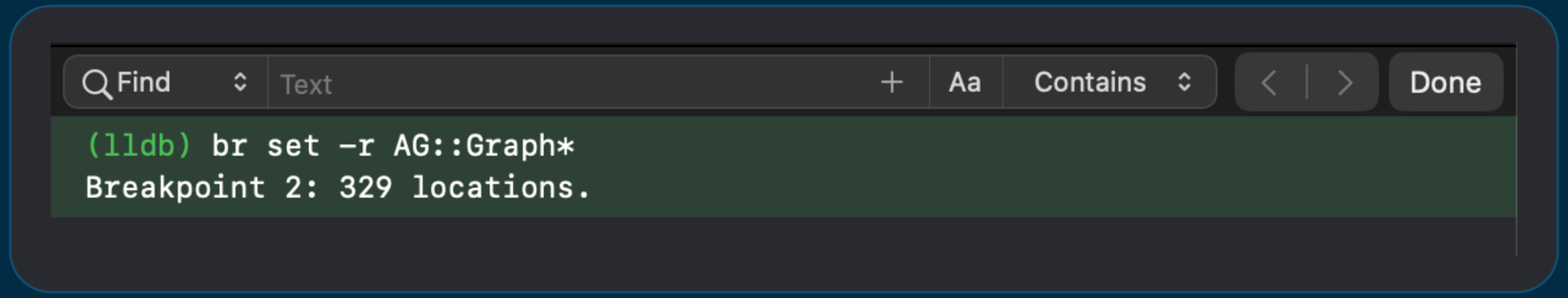


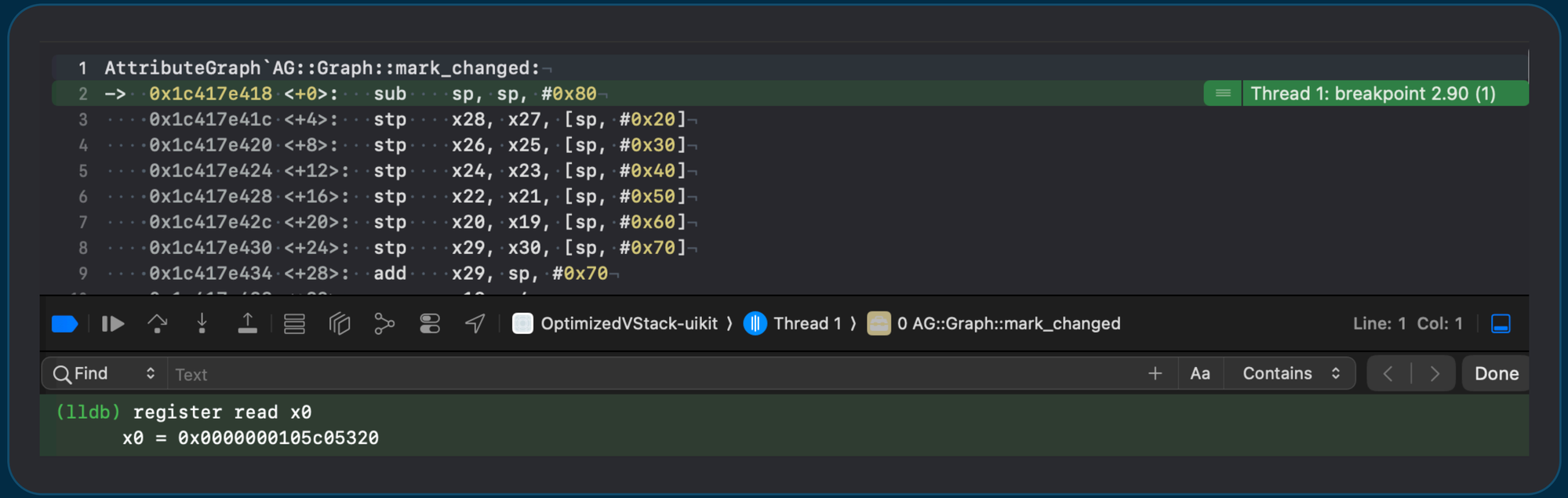
# Этап 1. Ищем pointer




+ «Сворачиваем» приложение

# Этап 1. Ищем pointer



# Этап 2. Вызываем AG::Graph::print()

AG::Graph::print() = адрес AttributeGraph в RAM + адрес функции print() внутри AttributeGraph

Адрес AttributeGraph

```
(lldb) image list | grep AttributeGraph
[  0] F3C7B4A6-9CCB-36F2-A13B-79700D884727 0x00000001c4174000
/Library/Developer/CoreSimulator/Volumes/iOS_23B86/Library/Developer/CoreSimulator/Profi
les/Runtimes/iOS
26.1
.simruntime/Contents/Resources/RuntimeRoot/System/Library/PrivateFrameworks/AttributeGra
ph.framework/AttributeGraph
```

# Этап 2. Вызываем AG::Graph::print()

AG::Graph::print() = адрес AttributeGraph в RAM + адрес функции print() внутри AttributeGraph

Адрес AttributeGraph

```
(lldb) image list | grep AttributeGraph
[  0] F3C7B4A6-9CCB-36F2-A13B-79700D884727 0x00000001c4174000
/Library/Developer/CoreSimulator/Volumes/iOS_23B86/Library/Developer/CoreSimulator/Profi
les/Runtimes/iOS
26.1
.simruntime/Contents/Resources/RuntimeRoot/System/Library/PrivateFrameworks/AttributeGra
ph.framework/AttributeGraph
```

Адрес функции print()

```
                                                % nm -C /Library/Developer/CoreSimulator/Volumes/iOS_23B86/Library
/Developer/CoreSimulator/Profiles/Runtimes/iOS\ 26.1.simruntime/Contents/Resources/RuntimeRoot/System/Library/Private
Frameworks/AttributeGraph.framework/AttributeGraph | grep AG::Graph::print\(\)
0000000000027be4 t AG::Graph::print() const
```

# Этап 2. Вызываем AG::Graph::print()

AG::Graph::print() = адрес AttributeGraph в RAM + адрес функции print() внутри AttributeGraph

Адрес AttributeGraph

```
(lldb) image list | grep AttributeGraph
[  0] F3C7B4A6-9CCB-36F2-A13B-79700D884727 0x00000001c4174000
/Library/Developer/CoreSimulator/Volumes/iOS_23B86/Library/Developer/CoreSimulator/Profi
les/Runtimes/iOS
26.1
.simruntime/Contents/Resources/RuntimeRoot/System/Library/PrivateFrameworks/AttributeGra
ph.framework/AttributeGraph
```

Адрес функции print()

```
                                                % nm -C /Library/Developer/CoreSimulator/Volumes/iOS_23B86/Library
/Developer/CoreSimulator/Profiles/Runtimes/iOS\ 26.1.simruntime/Contents/Resources/RuntimeRoot/System/Library/Private
Frameworks/AttributeGraph.framework/AttributeGraph | grep AG::Graph::print\(\)
0000000000027be4 t AG::Graph::print() const
```

AG::Graph::print() = 0x1C4174000+0x27BE4

# Этап 2. Вызываем AG::Graph::print()

Передаём в функцию AG::Graph::print() указатель на адрес графа

```
(lldb) expr -language c -- typedef void (*$PrintGraphFuncType)(void *);
(lldb) e -l c -- $PrintGraphFuncType $printGraphFunc = ($PrintGraphFuncType)(0x1C4174000+0x27be4);
(lldb) e -l c -- $printGraphFunc((void *)0x105c05320);
```

Адрес функции

Указатель на граф

# Этап 2. Вызываем AG::Graph::print()

Передаём в функцию AG::Graph::print() указатель на адрес графа

```
(lldb) expr -language c -- typedef void (*$PrintGraphFuncType)(void *);
(lldb) e -l c -- $PrintGraphFuncType $printGraphFunc = ($PrintGraphFuncType)(0x1C4174000+0x27be4);
(lldb) e -l c -- $printGraphFunc((void *)0x105c05320);
```

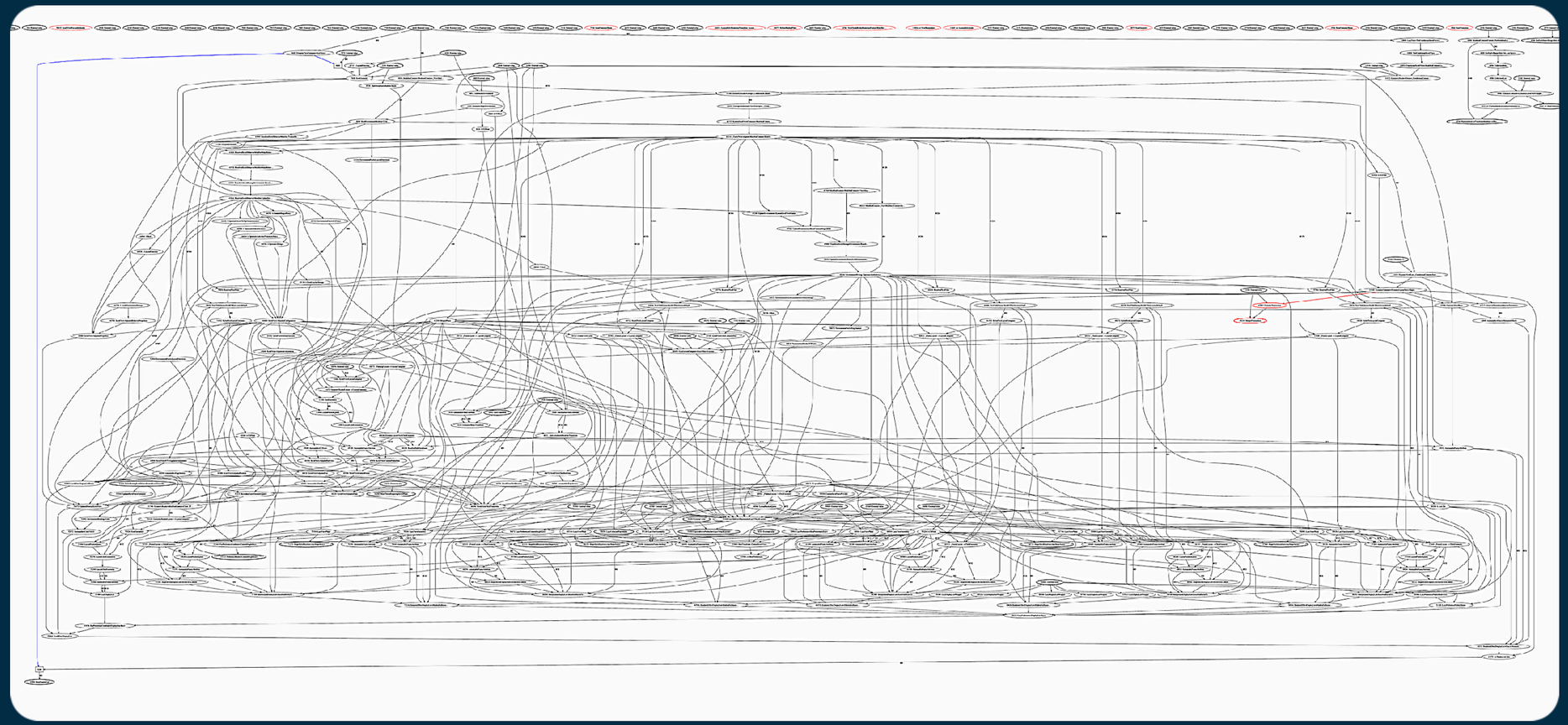
Адрес функции

Указатель на граф

```
digraph {
  _992[label="992: External value" style="bold"];
  _936[label="936: External value" style="bold"];
  _872[label="872: External value" style="bold"];
  _832[label="832: External value" style="bold"];
  _800[label="800: External value" style="bold"];
  _768[label="768: External value" style="bold"];
  _736[label="736: External value" style="bold"];
  _696[label="696: External value" style="bold"];
  _656[label="656: External value" style="bold"];
  _624[label="624: External value" style="bold"];
  _576[label="576: External value" style="bold"];
  _536[label="536: External value" style="bold"];
  _1816[label="1816: WriteSceneList"];
  _1560 -> _1816[];
  _1744 -> _1816[];
  _1744[label="1744: WindowSceneList<WindowGroupConfiguration…"];
  _1560 -> _1744[];
  _1660 -> _1744[ label="@0"];
  _1660[label="1660: WindowSceneList<WindowGroupConfiguration…"];
  _1628 -> _1660[];
  _1628[label="1628: WindowGroup<TestConditionalScrollView>"];
  _1600 -> _1628[];
  _1600[label="1600: External value" style="bold"];
  _1560[label="1560: RootEnvironment"];
(lldb)
```

# Этап 3. Экспорт .graphviz графа приложения в Reader

Воспользуемся Graphviz-Reader



OZON tech

# Краткий пошаговый гайд по получению графа

# Обзор графа

# Ищем узел с вхождением GeometryReader

# Ищем узел с вхождением GeometryReader

1 Ставим брейпоинт

```
10  struct ConditionalScrollView<Content: View>: View {
36      private var measuredContent: some View {
37          content
38              .background(
39                  GeometryReader { geo in
40                      Color.clear
41                          .preference(
42                              key: ContentHeightKey.self,
43                              value: geo.size.height
44                          )
45                  }
46              )
47              .onPreferenceChange(ContentHeightKey.self) {
48                  contentHeight = $0
49              }
50      }
51  }
```

# Ищем узел с вхождением GeometryReader

1 Ставим брейпоинт

```
10 struct ConditionalScrollView<Content: View>: View {
36     private var measuredContent: some View {
37         content
38             .background(
39                 GeometryReader { geo in
40                     Color.clear
41                         .preference(
42                             key: ContentHeightKey.self,
43                             value: geo.size.height
44                         )
45                 }
46             )
47             .onPreferenceChange(ContentHeightKey.self) {
48                 contentHeight = $0
49             }
50     }
51 }
```

2 Получаем граф

```
Find  Text  Aa  Contains  Done
(lldb) e -l c -- $printGraphFunc((void *)0x105107340);
digraph {
  _992[label="992: External value" style="bold"];
  _936[label="936: External value" style="bold"];
  _872[label="872: External value" style="bold"];
  _832[label="832: External value" style="bold"];
(lldb)  Executing command
```

# Ищем узел с вхождением GeometryReader

## 1 Ставим брейпоинт

```
10  struct ConditionalScrollView<Content: View>: View {
36      private var measuredContent: some View {
37          content
38              .background(
39                  GeometryReader { geo in
40                      Color.clear
41                          .preference(
42                              key: ContentHeightKey.self,
43                              value: geo.size.height
44                          )
45                  }
46              )
47              .onPreferenceChange(ContentHeightKey.self) {
48                  contentHeight = $0
49              }
50      }
51  }
```

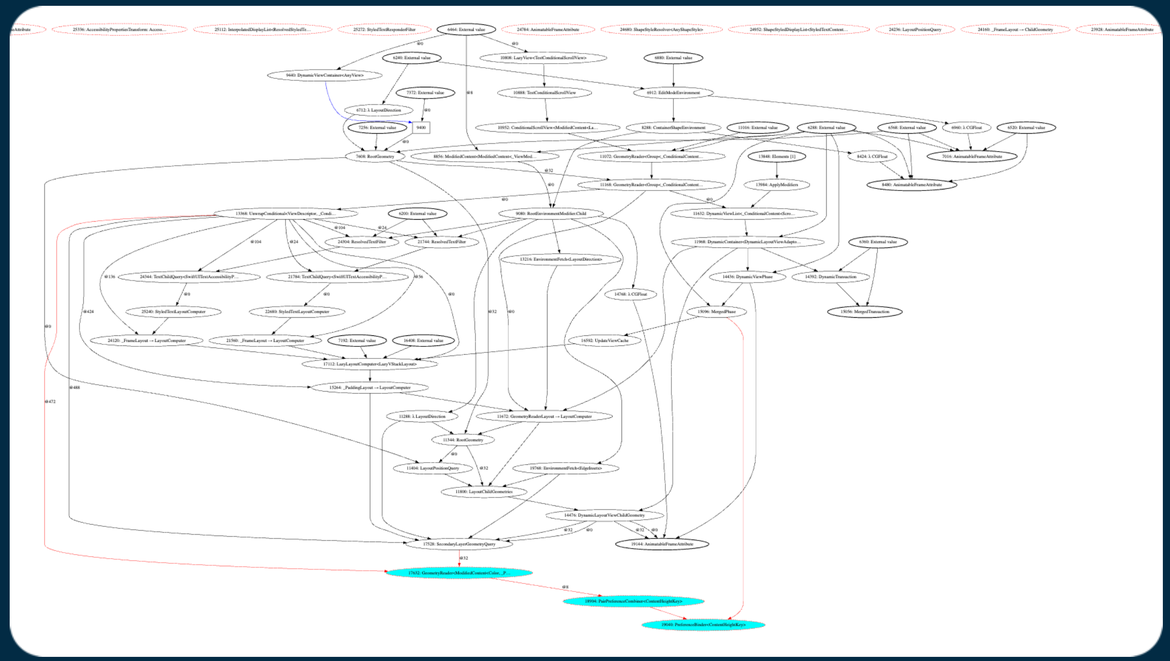
## 2 Получаем граф

Find | Text | Aa | Contains | Done

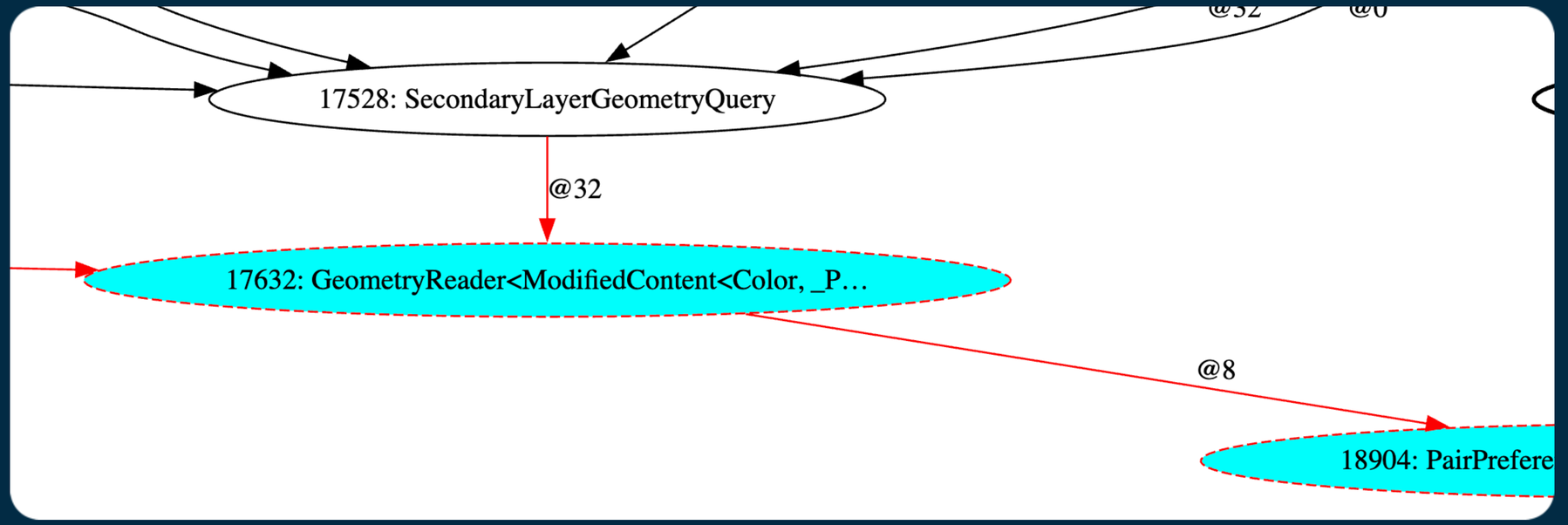
```
(lldb) e -l c -- $printGraphFunc((void *)0x105107340);

digraph {
  _992[label="992: External value" style="bold"];
  _936[label="936: External value" style="bold"];
  _872[label="872: External value" style="bold"];
  _832[label="832: External value" style="bold"];
```

(lldb) Executing command

## 3 Экспорт графа в Graphviz - reader



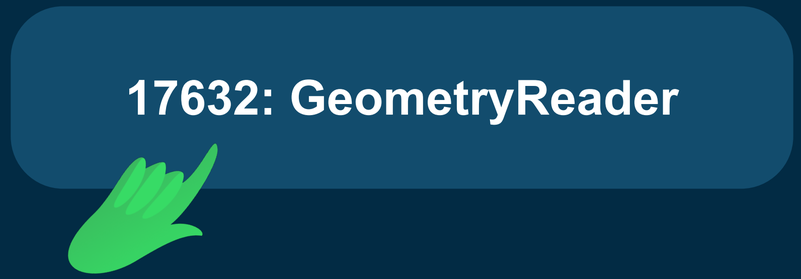
# Ищем узел с вхождением GeometryReader



# Извлекаем часть графа

**17632: GeometryReader**

# Извлекаем часть графа



# Извлекаем часть графа

**17632: GeometryReader**

AG::Graph::print_attribute(AttributeID)

# Извлекаем часть графа

Поиск через «nm»
по аналогии
с AG::Graph::print()

**17632: GeometryReader**

AG::Graph::print_attribute(AttributeID)

Адрес AG::Graph::print_attribute(AttributeID) = 0x2823C

# Извлекаем часть графа

Поиск через «nm» по аналогии с AG::Graph::print()

**17632: GeometryReader**

AG::Graph::print_attribute(AttributeID)

Адрес AG::Graph::print_attribute(AttributeID) = 0x2823C

Вызываем метод:

```
(lldb) e -l c -- typedef void (*$PrintAttributeFuncType)(void *, void *);
(lldb) e -l c -- $PrintAttributeFuncType $printAttributeFunc =
($PrintAttributeFuncType)(0x1C4174000+0x2823C);
(lldb) e -l c -- $printAttributeFunc((void *)0x105107340, (void *)17632);
```

Адрес графа

# Извлекаем часть графа

Поиск через «nm» по аналогии с AG::Graph::print()

**17632: GeometryReader**

AG::Graph::print_attribute(AttributeID)

Адрес AG::Graph::print_attribute(AttributeID) = 0x2823C

Вызываем метод:

```
(lldb) e -l c -- typedef void (*$PrintAttributeFuncType)(void *, void *);
(lldb) e -l c -- $PrintAttributeFuncType $printAttributeFunc =
($PrintAttributeFuncType)(0x1C4174000+0x2823C);
(lldb) e -l c -- $printAttributeFunc((void *)0x105107340, (void *)17632);
```

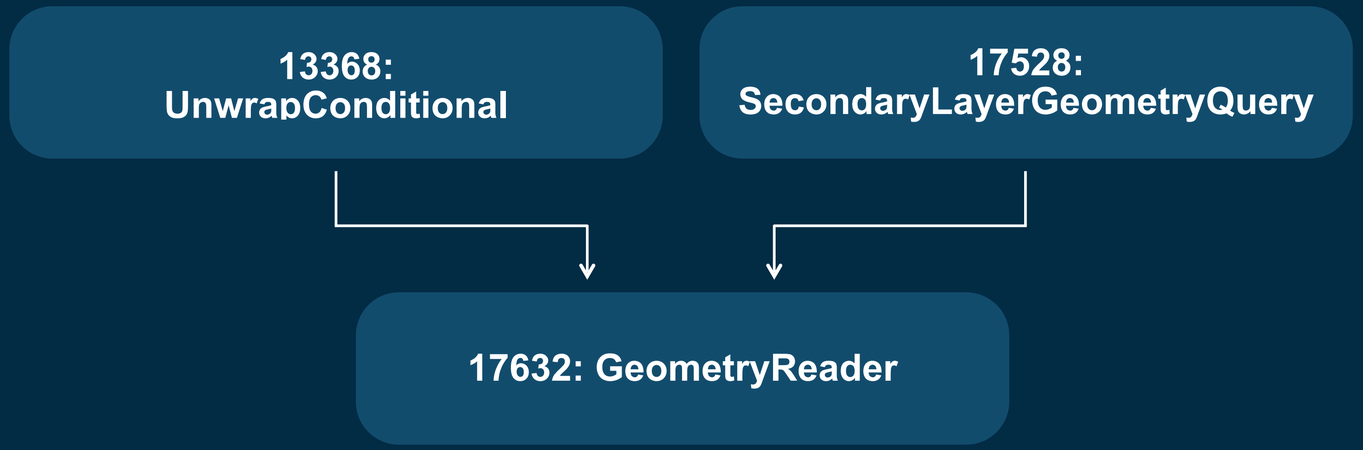
Адрес графа

Результат:

```
identifier = 17632, type = 107, self_size = 28, value_size = 16, pod_self = true,
bitwise_takable_self = true, bitwise_takable_value = true, input_count = 2,
output_count = 1, dirty = 1, updating = 1, self = GeometryReader<ModifiedContent<Color,
_PreferenceWritingModifier<ContentHeightKey>>>.Child
```

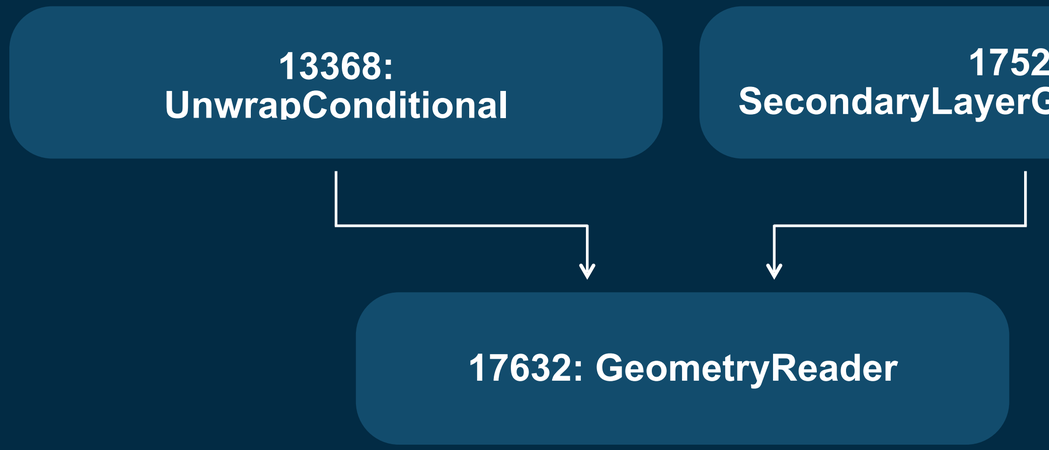
# Извлекаем часть графа

Поднимаемся вверх по графу и вызываем AG::Graph::print_attribute(AttributeID) для родительских узлов



# Извлекаем часть графа

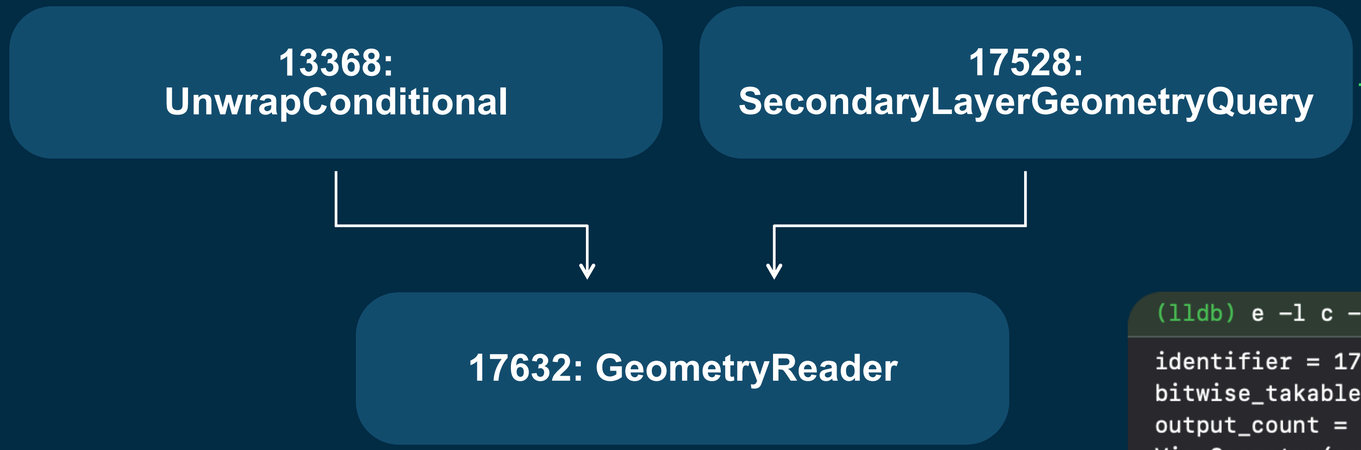
Поднимаемся вверх по графу и вызываем AG::Graph::print_attribute(AttributeID) для родительских узлов




```
(lldb) e -l c -- $printAttributeFunc((void *)0x105107340, (void *)17528);
identifier = 17528, type = 106, self_size = 24, value_size = 64, pod_self = true,
bitwise_takable_self = true, bitwise_takable_value = true, input_count = 6,
output_count = 1, dirty = 0, updating = 0, self = SecondaryLayerGeometryQuery, value =
ViewGeometry(origin: (0.0, 62.0), dimensions: SwiftUI.ViewDimensions(guideComputer:
SwiftUI.LayoutComputer(box: SwiftUI.(unknown context at
$1db481110).LayoutEngineBox<SwiftUI.LayoutComputer.DefaultEngine>, seed: 0), size:
SwiftUI.ViewSize(value: (402.0, 221.0), _proposal: (402.0, 221.0))))
```

# Извлекаем часть графа

Поднимаемся вверх по графу и вызываем AG::Graph::print_attribute(AttributeID) для родительских узлов




```
(lldb) e -l c -- $printAttributeFunc((void *)0x105107340, (void *)17528);
identifier = 17528, type = 106, self_size = 24, value_size = 64, pod_self = true,
bitwise_takable_self = true, bitwise_takable_value = true, input_count = 6,
output_count = 1, dirty = 0, updating = 0, self = SecondaryLayerGeometryQuery, value =
ViewGeometry(origin: (0.0, 62.0), dimensions: SwiftUI.ViewDimensions(guideComputer:
SwiftUI.LayoutComputer(box: SwiftUI.(unknown context at
$1db481110).LayoutEngineBox<SwiftUI.LayoutComputer.DefaultEngine>, seed: 0), size:
SwiftUI.ViewSize(value: (402.0, 221.0), _proposal: (402.0, 221.0))))
```

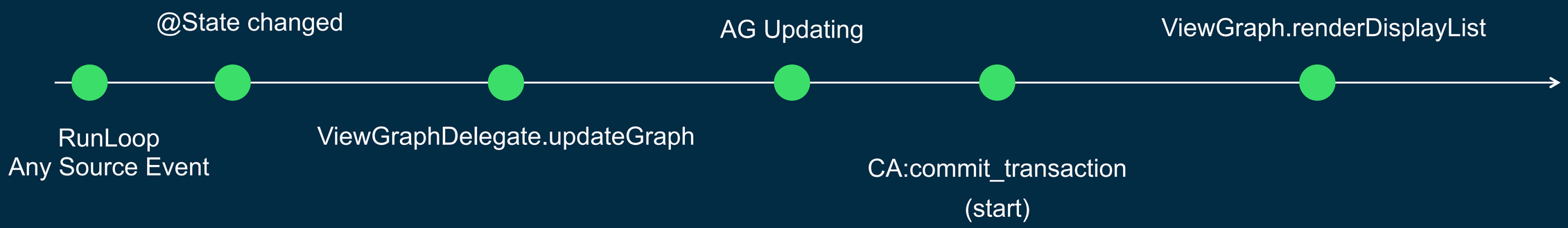
Размеры «приносит» внешний узел

# Размеры в GeometryReader «приносит» внешний узел

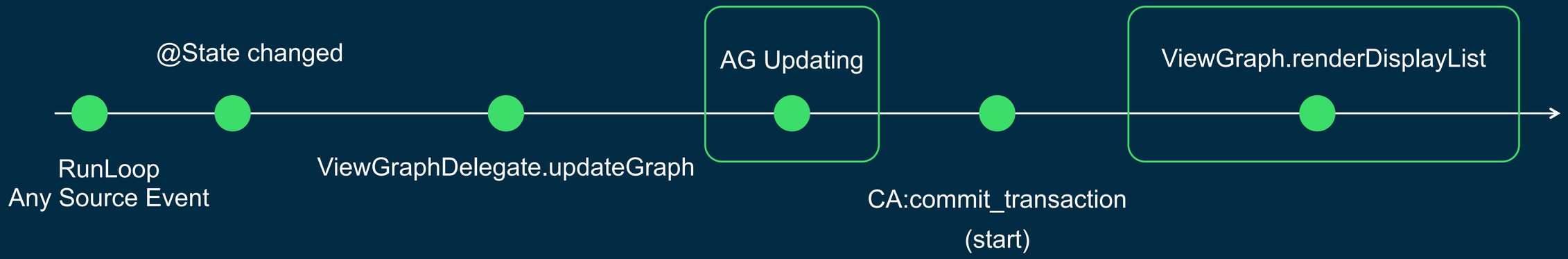
Но какой узел и как он рассчитал размер?

# Обратимся к пайплайну обновления кадров в SwiftUI

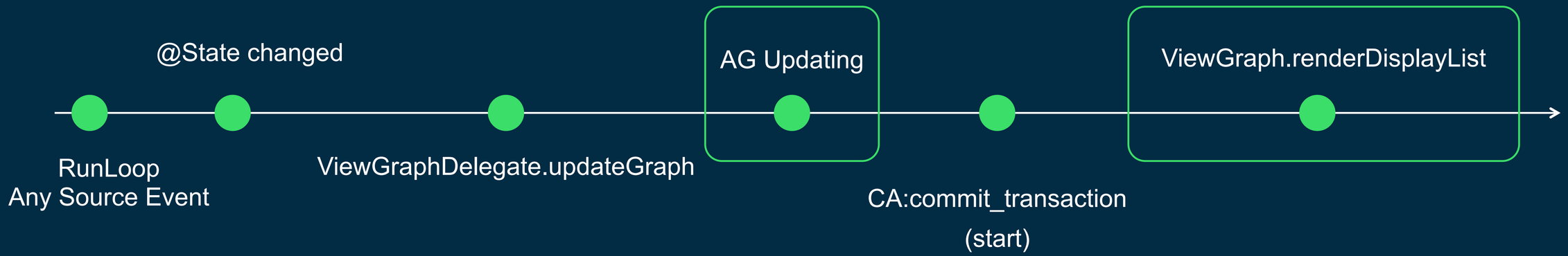
# Пайплайн обновления кадра в SwiftUI



# Пайплайн обновления кадра в SwiftUI



# Пайплайн обновления кадра в SwiftUI




AG Updating

**Распространение загрязнения**

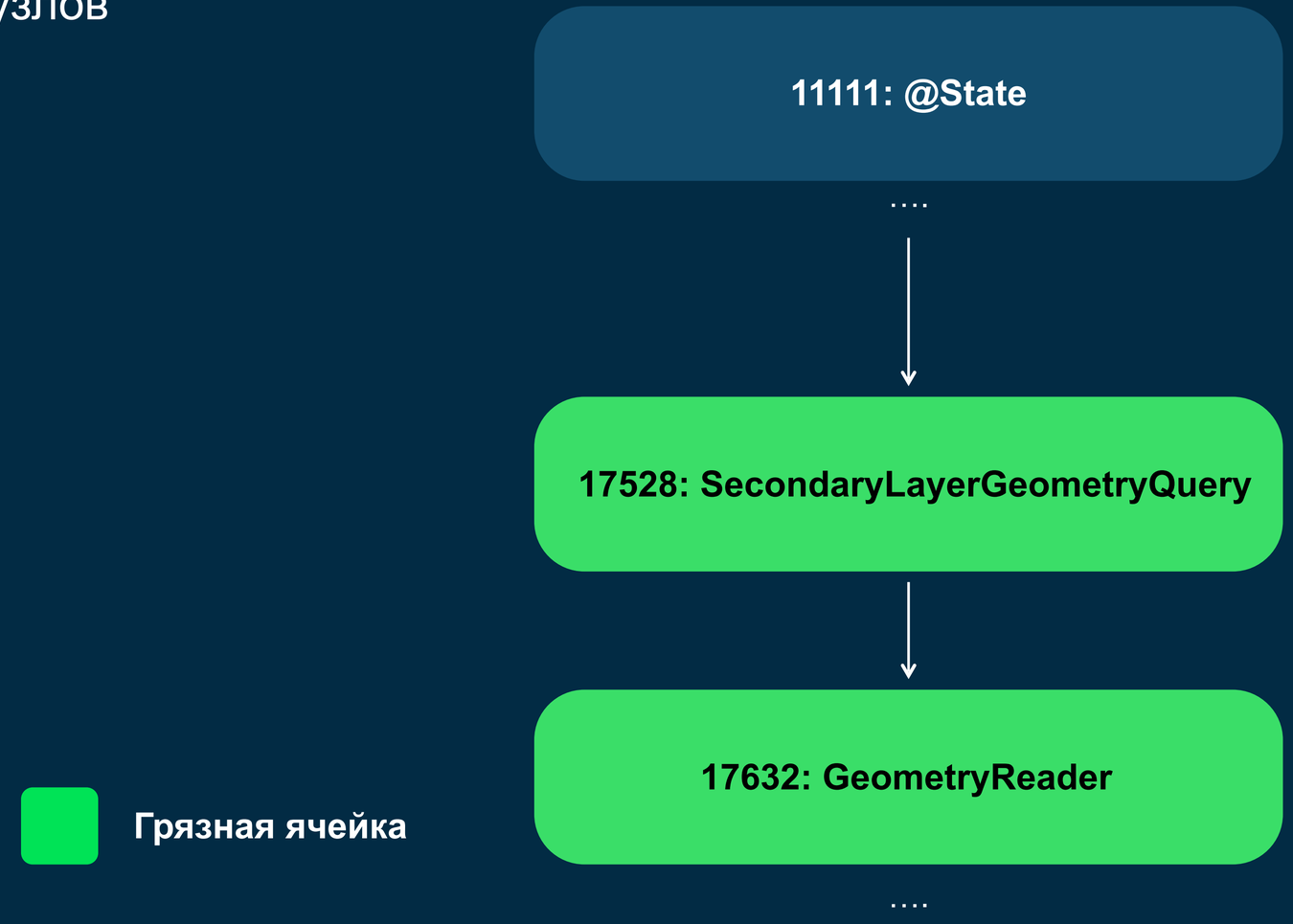
**Обновление загрязненных узлов**

ViewGraph.render DisplayList

**Обновление корневого узла**

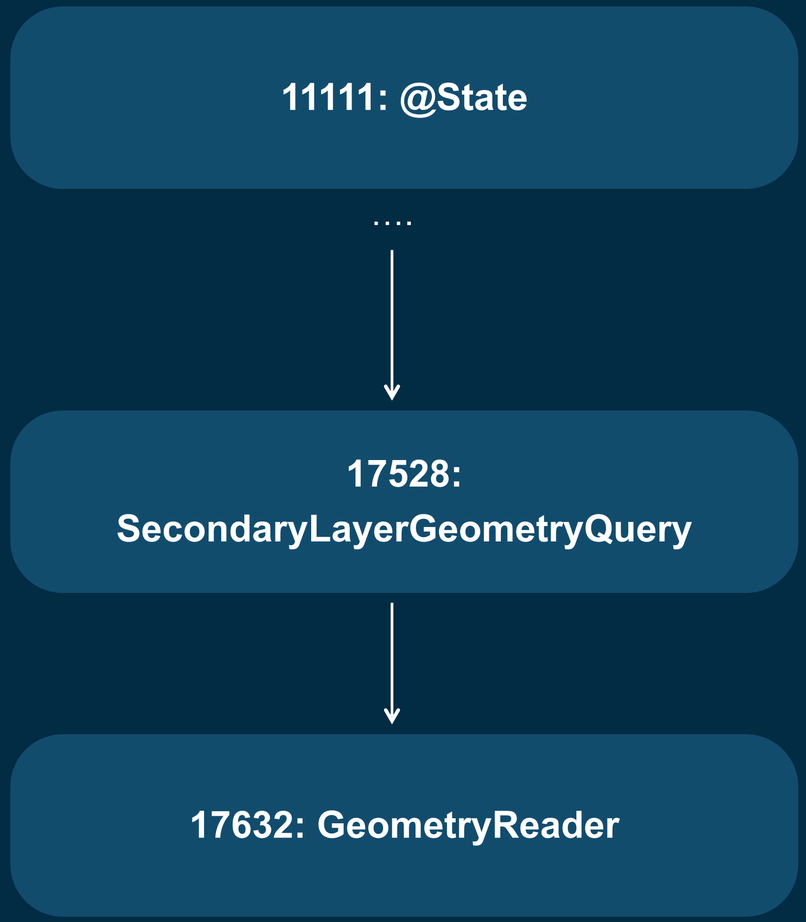
# 1. Распространение загрязнения

При инициализации или обновлении @State
происходит «загрязнение» всех дочерних узлов



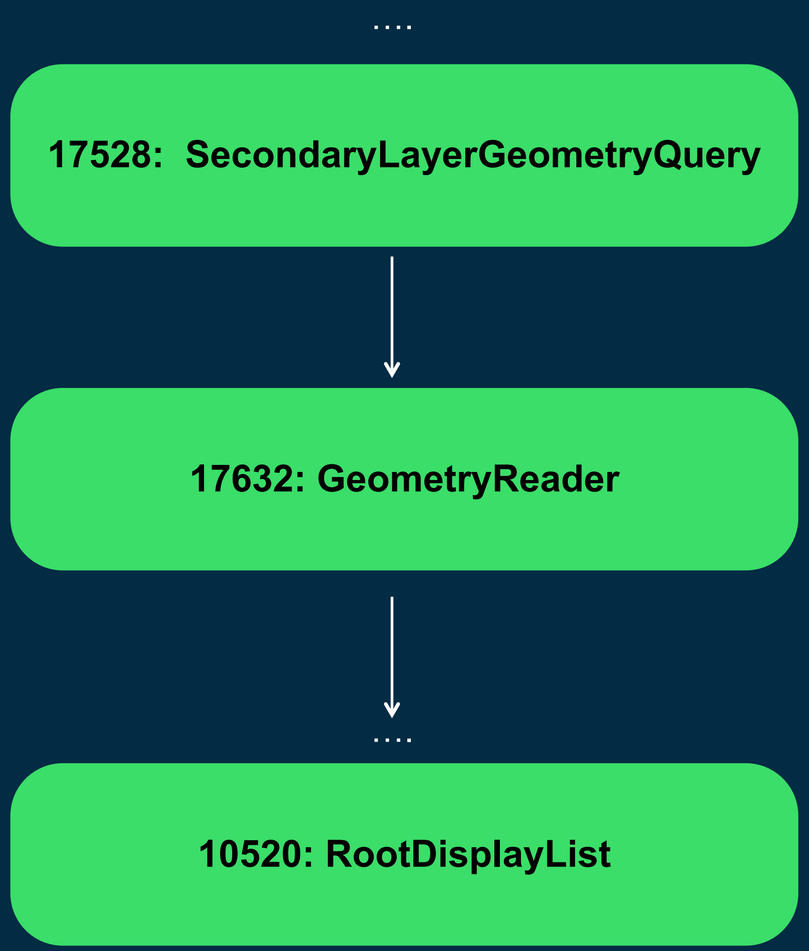
# 1. Распространение загрязнения

При инициализации или обновлении @State происходит «загрязнение» всех дочерних узлов




```
(lldb) e -l c -- $printAttributeFunc((void *)0x105107340, (void *)17528);

identifier = 17528, type = 106, self_size = 24, value_size = 64, pod_self = true,
bitwise_takable_self = true, bitwise_takable_value = true, input_count = 6,
output_count = 1, dirty = 0, updating = 0, self = SecondaryLayerGeometryQuery, value =
ViewGeometry(origin: (0.0, 62.0), dimensions: SwiftUI.ViewDimensions(guideComputer:
SwiftUI.LayoutComputer(box: Swi(unknown context at
$1db481110).LayoutEngineBox<Swi.LayoutComputer.DefaultEngine>, seed: 0), size:
SwiftUI.ViewSize(value: (402.0, 221.0), _proposal: (402.0, 221.0))))
```

# 2. Обновление загрязнённых узлов




**Алгоритм получения RootDisplayList:**

1. Положить в стек узел, требующий обновления (10520: RootDisplayList)
2. Если стек пуст → выход
3. У узла на вершине стека есть «грязные» родители?

Да:

Добавить «грязного» родителя в стек

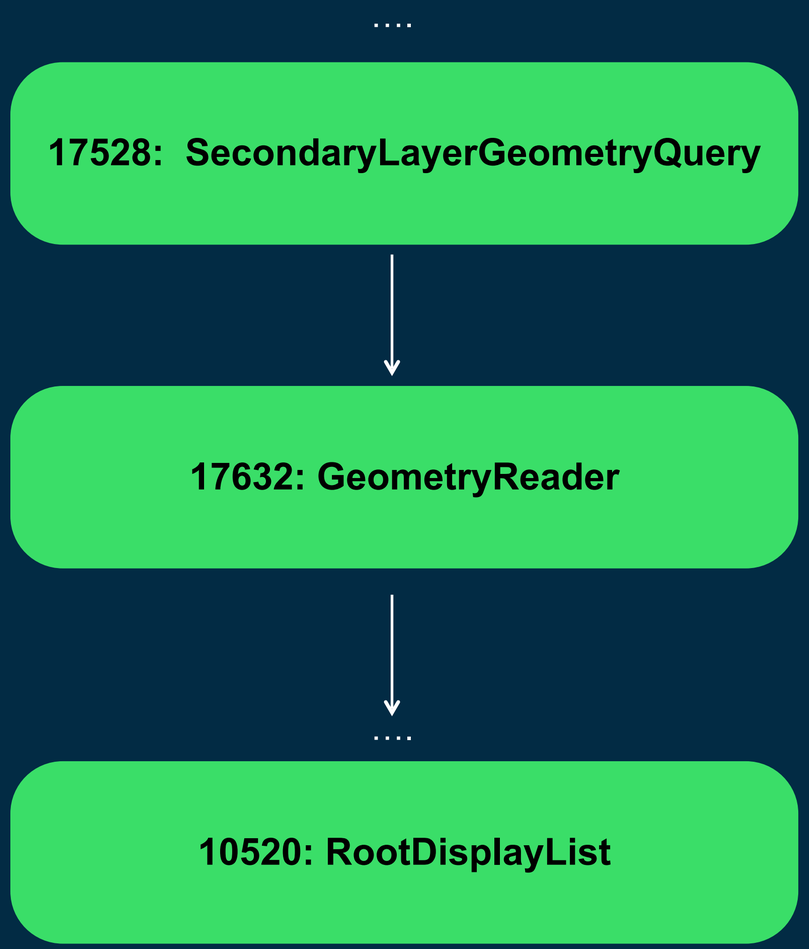
Нет:

Обновить узел из вершины стека и удалить его из стека

К шагу 2

**Грязная ячейка**

# 2. Обновление загрязнённых узлов




**Алгоритм получения RootDisplayList:**

1. Положить в стек узел, требующий обновления (10520: RootDisplayList)
2. Если стек пуст → выход
3. У узла на вершине стека есть «грязные» родители?

Да:

Добавить «грязного» родителя в стек

Нет:

Обновить узел из вершины стека и удалить его из стека

К шагу 2

AG::Graph::UpdateStack::update()
AG::Graph::print_stack()

**Грязная ячейка**

# Переформулируем задачу

Нужно перехватить AG::Graph::UpdateStack::update()
в момент обновления узла, рассчитывающего размер «списка»

# Обновление какого узла перехватывать?

# Уменьшаем область поиска «вычисляющего» узла

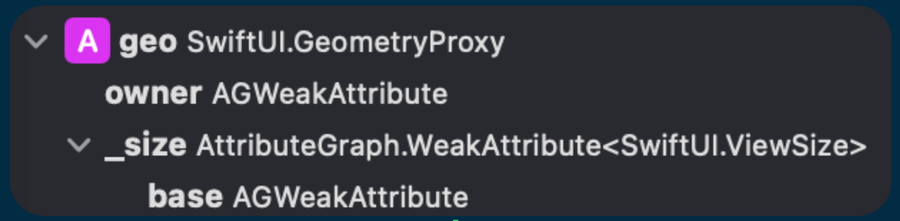
1. Ставим брейкпоинт на метод геттер «geo.size»

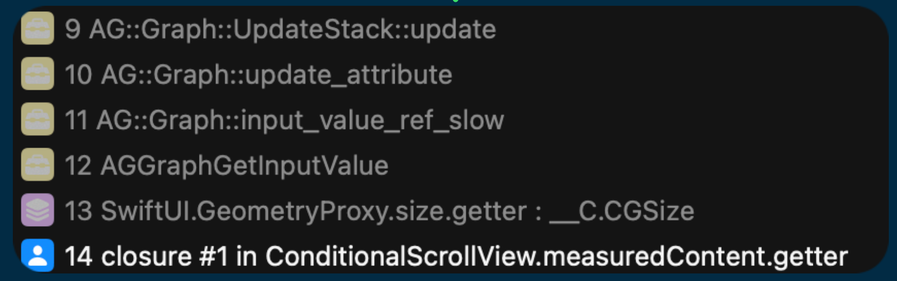
```
32      private var measuredContent: some View {
33          content
34              .background(
35                  GeometryReader { geo in
36                      Color.clear
37                          .preference(
38                              key: ContentHeightKey.self,
39                              value: geo.size.height
40                          )
```

# Уменьшаем область поиска «вычисляющего» узла

1. Ставим брейкпоинт на метод геттер «geo.size»

```
32      private var measuredContent: some View {
33          content
34              .background(
35                  GeometryReader { geo in
36                      Color.clear
37                          .preference(
38                              key: ContentHeightKey.self,
39                              value: geo.size.height
40                          )
```



# Уменьшаем область поиска «вычисляющего» узла

2. Ставим брейкпоинт на «sizeThatFits» как на универсальный метод для получения размеров в SwiftUI

```
(lldb) br set -r sizeThatFits
Breakpoint 2: 485 locations.
```

# Уменьшаем область поиска «вычисляющего» узла

2. Ставим брейкпоинт на «sizeThatFits» как на универсальный метод для получения размеров в SwiftUI

```
(lldb) br set -r sizeThatFits
Breakpoint 2: 485 locations.
```

3. Срабатывает брейкпоинт

```
1  SwiftUICore`sizeThatFits:
2  ····0x1db13f50c·<+0>:··sub····sp,·sp,·#0x40
```

# Уменьшаем область поиска «вычисляющего» узла

2. Ставим брейкпоинт на «sizeThatFits» как на универсальный метод для получения размеров в SwiftUI

```
(lldb) br set -r sizeThatFits
Breakpoint 2: 485 locations.
```

3. Срабатывает брейкпоинт

```
1  SwiftUICore`sizeThatFits:
2      0x1db13f50c <+0>:  sub    sp, sp, #0x40
```

4. Проваливаемся в внутренний вызов sizeThatFits

```
1  SwiftUICore`protocol witness for SwiftUI.LayoutEngine.sizeThatFits(SwiftUI._ProposedSize) -> __C.CGSize in conformance
       SwiftUI.LazyLayoutComputer<τ_0_0>.Engine : SwiftUI.LayoutEngine in SwiftUI:
2  ->  0x1dad72364 <+0>: b      0x1dad71b2c               ; SwiftUI.LazyLayoutComputer.Engine.sizeThatFits(SwiftUI._ProposedSize) -> __C.CGSize   Thread 1:...
```

# Уменьшаем область поиска «вычисляющего» узла

2. Ставим брейкпоинт на «sizeThatFits» как на универсальный метод для получения размеров в SwiftUI

```
(lldb) br set -r sizeThatFits
Breakpoint 2: 485 locations.
```

3. Срабатывает брейкпоинт

```
SwiftUICore`sizeThatFits:
    0x1db13f50c <+0>:  sub    sp, sp, #0x40
```

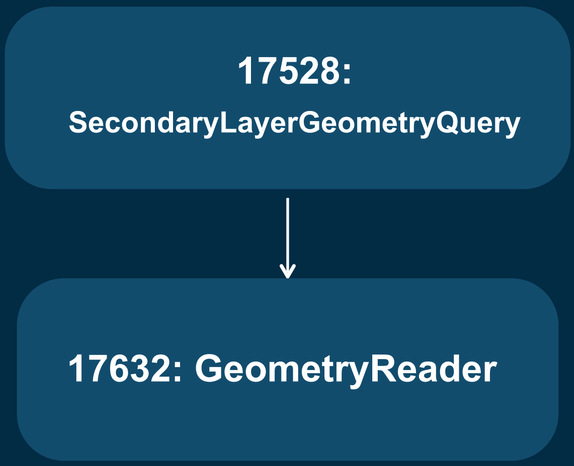
В методе — только чтение размеров из кэша

4. Проваливаемся в внутренний вызов sizeThatFits

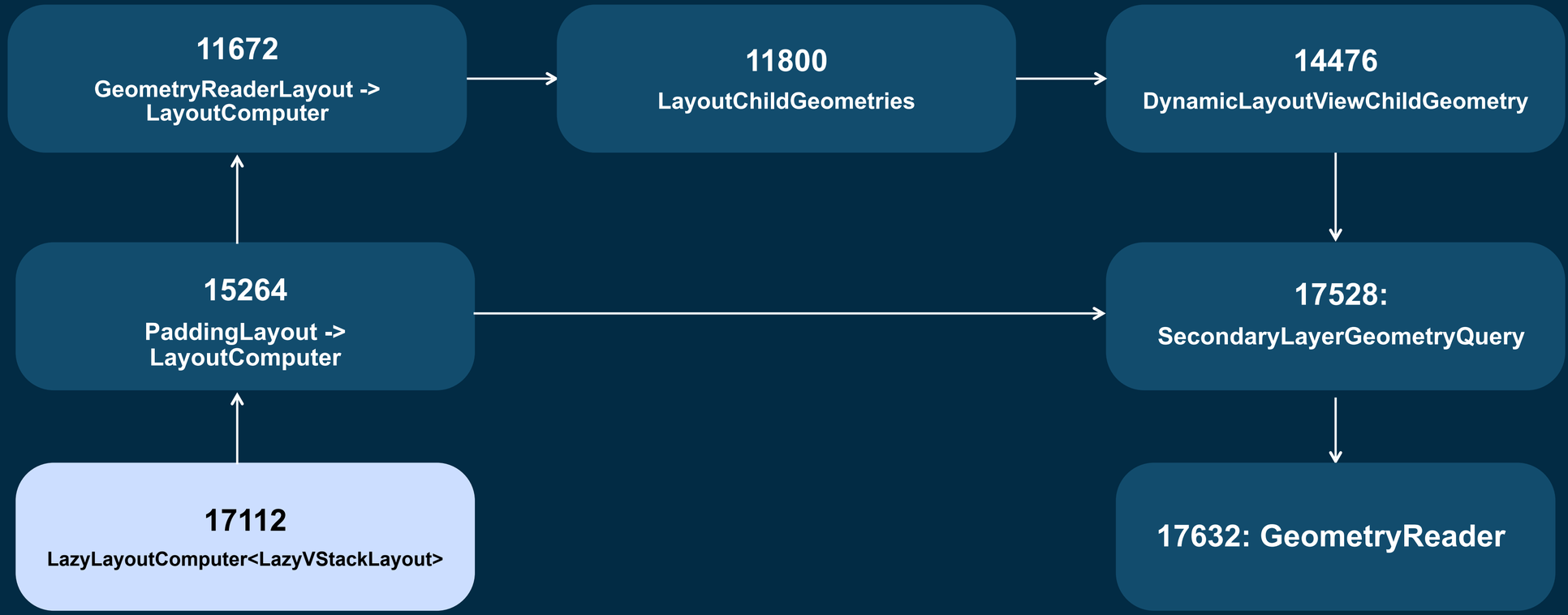
```
SwiftUICore`protocol witness for SwiftUI.LayoutEngine.sizeThatFits(SwiftUI._ProposedSize) -> __C.CGSize in conformance
    SwiftUI.LazyLayoutComputer<τ_0_0>.Engine : SwiftUI.LayoutEngine in SwiftUI:
-> 0x1dad72364 <+0>: b    0x1dad71b2c               ; SwiftUI.LazyLayoutComputer.Engine.sizeThatFits(SwiftUI._ProposedSize) -> __C.CGSize
```

# Ищем LazyLayoutComputer!

# Родительский подграф для GeometryReader

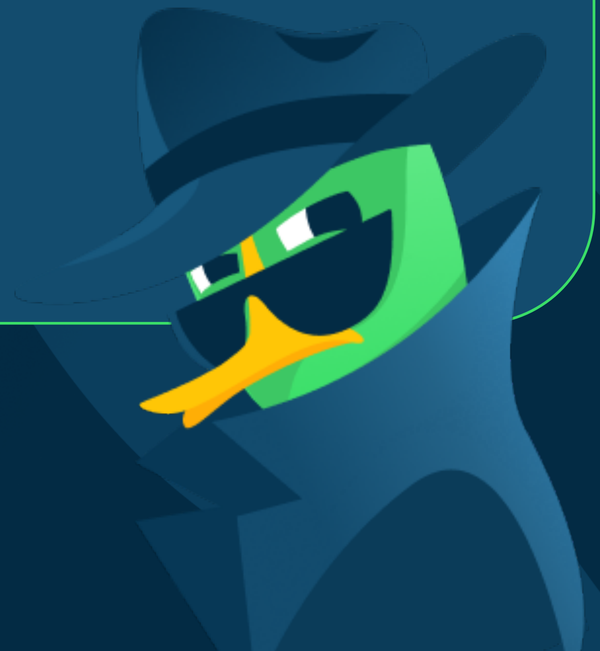


# Родительский подграф для GeometryReader



# Переформулируем задачу

Нужно перехватить AG::Graph::UpdateStack::update()
в момент обновления узла 17112: LazyLayoutComputer



# Момент обновления LazyLayoutComputer

1. Перезагружаем демо-проект
2. Ставим брейкпоинт на AG::Graph::UpdateStack::update()

```
(lldb) br set -r AG::Graph::UpdateStack::update() -s AttributeGraph
Breakpoint 9: 2 locations.
```

**17112**

**LazyLayoutComputer<LazyVStackLayout>**

# Момент обновления LazyLayoutComputer

1. Перезагружаем демо-проект

2. Ставим брейкпоинт на AG::Graph::UpdateStack::update()

```
(lldb) br set -r AG::Graph::UpdateStack::update() -s AttributeGraph
Breakpoint 9: 2 locations.
```

3. Останавливаем поиск в момент, когда на вершине «стека обновления» «LazyLayoutComputer» — узел

```
(lldb) e -l c -- $printUpdateGraphStackFunc((void *)0x105107340);
```

```
frame 0.7: attribute 17112; count=1, index=6/6 PDV
frame 0.6: attribute 15264; count=1, index=2/2 DV
frame 0.5: attribute 11672; count=1, index=4/4 PDV
frame 0.4: attribute 11344; count=1, index=3/3 DV
frame 0.3: attribute 11404; count=1, index=2/2 DV
frame 0.2: attribute 11800; count=1, index=1/4 DV
frame 0.1: attribute 14476; count=1, index=1/2 DV
frame 0.0: attribute 19144; count=1, index=2/4 DV
```

**17112**

**LazyLayoutComputer<LazyVStackLayout>**

# Момент обновления LazyLayoutComputer

Переходим в функцию «обновления» LazyLayoutComputer

**17112**

**LazyLayoutComputer<LazyVStackLayout>**

```
[redacted] 〉 Thread 1 〉 0 AG::Graph::UpdateStack::update 〉 No Selection
124 -> 0x1c417f724 <+488>: blr x8
```

```
1 SwiftUICore`generic specialization <SwiftUI.Axis.Set, AttributeGraph.Map<SwiftUI.EnvironmentValues, SwiftUI.Axis.Set>> of
      implicit closure #1 (Swift.UnsafeMutableRawPointer, __C.AGAttribute) -> () in closure #1 () ->
      (Swift.UnsafeMutableRawPointer, __C.AGAttribute) -> () in closure #1 (Swift.UnsafePointer<τ_1_0>) ->
      AttributeGraph.Attribute<τ_0_0> in AttributeGraph.Attribute.init<τ_0_0 where τ_0_0 == τ_1_0.Value, τ_1_0:
      AttributeGraph.Rule>(τ_1_0) -> AttributeGraph.Attribute<τ_0_0>:
2 -> 0x1dae9b25c <+0>: sub sp, sp, #0x50          Thread 1: instruction step into
```

(Переходим с ASM на человеческий)

# Алгоритм обновления Layout-узла

1) Вызов итератора по всем ячейкам:

SwiftUICore`SwiftUI.ForEachState.forEachItem:

n = Первый проход ? 2 : кол-во ячеек на экране

Для n ячеек:

1.1) Расчёт размера ячейки

SwiftUICore`SwiftUI.ViewLayoutEngine.sizeThatFits(ProposedSize) -> __C.CGSize:

1.2) Записать в кэш размеры ячейки

SwiftUI.EstimationCache.add(length: CGFloat, spacing: CGFloat>) -> ():

**17112**

**LazyLayoutComputer<LazyVStackLayout>**

# Алгоритм обновления Layout-узла

1) Вызов итератора по всем ячейкам:

SwiftUICore`SwiftUI.ForEachState.forEachItem:

n = Первый проход ? 2 : кол-во ячеек на экране

Для n ячеек:

1.1) Расчёт размера ячейки

SwiftUICore`SwiftUI.ViewLayoutEngine.sizeThatFits(ProposedSize) -> __C.CGSize:

1.2) Записать в кэш размеры ячейки

SwiftUI.EstimationCache.add(length: CGFloat, spacing: CGFloat>) -> ():

2) Расчёт размера списка

SwiftUI.LazyStack<…>.sizeThatFits(ProposedViewSize, _LazyLayout_Subviews) -> __C.CGSize:

2.1) Расчёт среднего размера ячейки

average = SwiftUI.EstimationCache.average.getter

average = (1-я ячейка + 2-я ячейка) / 2

2.2) Высота списка = Сумма точных высот + average * кол-во неизмеренных элементов

**17112**

**LazyLayoutComputer<LazyVStackLayout>**

# Напомним вывод «высот» в консоль

```
ConditionalScrollView {
    LazyVStack(spacing: 16) {
        Text("Текст 1")
            .frame(height: 20)

        Text("Текст 2")
            .frame(height: 30)

        Text("Текст 3")
            .frame(height: 40)

        Text("Текст 4")
            .frame(height: 500)

        Text("Текст 5")
            .frame(height: 400)
    }
    .padding()
}
```

```
Высота: 0.0
Высота: 221.0
Высота: 1086.0
Высота: 221.0
Высота: 1086.0
```

1-ый LazyLayoutComputer

Estimate = (20 + 30) / 2 = 25

Высота списка = 1-я + 2-я ячейка + estimate ячейки * 3 + spacer * 4 + paddings

Высота контента = Высота списка + paddings

# Напомним вывод «высот» в консоль

```
ConditionalScrollView {
    LazyVStack(spacing: 16) {
        Text("Текст 1")
            .frame(height: 20)

        Text("Текст 2")
            .frame(height: 30)

        Text("Текст 3")
            .frame(height: 40)

        Text("Текст 4")
            .frame(height: 500)

        Text("Текст 5")
            .frame(height: 400)
    }
    .padding()
}
```

```
Высота: 0.0
Высота: 221.0
Высота: 1086.0
Высота: 221.0
Высота: 1086.0
```

1-ый LazyLayoutComputer

Estimate = 25

Высота списка = 1-я + 2-я ячейка + estimate ячейки * 3 + spacer * 4 + paddings

Высота контента = Высота списка + paddings

Высота списка = (20 + 30) + 25 * 3 + 16 * 4 = 189

Высота контента = 189 + 32 = **221**

# LazyVStack пишет логи в OSLog!

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```

# Вопросы к GeometryReader + LazyVStack

Это результат sizeThatFits.
В первом измерении:
2 ячейки измерены
+ 3 ячейки предугаданы

Почему GeometryReader возвращает **неверный** размер для LazyVStack?

Результаты метода sizeThatFits. Чем больше измеренных ячеек, тем точнее результат

Что за числа возвращал GeometryReader?

Почему нет «моргающего» экрана при вычислении неверного размера?

Как понять, что размер **«стабилизировался»**?

# Перед прощанием с «первым» демо-примером

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```


| Timestamp | Type | Proc... | Subsystem | Category | Message |
|---|---|---|---|---|---|
| 00:00.817.498 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: sizeThatFits(370.0) -> 189.0 |
| 00:00.818.514 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: Measured estimates -> 41.0 |
| 00:00.818.617 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | (StackPlacement in _973C9973BC16DEAF0CF3109FFDE31321)<LazyVStackLayout>: placing from #0, 0.0 in 0.0...796.0 |
| 00:00.819.210 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: invalid #2; 1054.0 vs 189.0 |

...

| | | | | | |
|---|---|---|---|---|---|
| 00:00.848.180 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | (StackPlacement in _973C9973BC16DEAF0CF3109FFDE31321)<LazyVStackLayout>: placing from #0, 0.0 in 0.0...796.0 |
| 00:00.848.239 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: placed(0.0...796.0) -> 0..<5, 0.0...1054.0, invalid: false |

# Перед прощанием с «первым» демо-примером

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```


| Timestamp | Type | Proc... | Subsystem | Category | Message |
|---|---|---|---|---|---|
| 00:00.817.498 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: sizeThatFits(370.0) -> 189.0 |
| 00:00.818.514 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: Measured estimates -> 41.0 |
| 00:00.818.617 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | (StackPlacement in _973C9973BC16DEAF0CF3109FFDE31321)<LazyVStackLayout>: placing from #0, 0.0 in 0.0...796.0 |
| 00:00.819.210 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: invalid #2; 1054.0 vs 189.0 |
| ... | | | | | |
| 00:00.848.180 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | (StackPlacement in _973C9973BC16DEAF0CF3109FFDE31321)<LazyVStackLayout>: placing from #0, 0.0 in 0.0...796.0 |
| 00:00.848.239 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: placed(0.0...796.0) -> 0..<5, 0.0...1054.0, invalid: false |

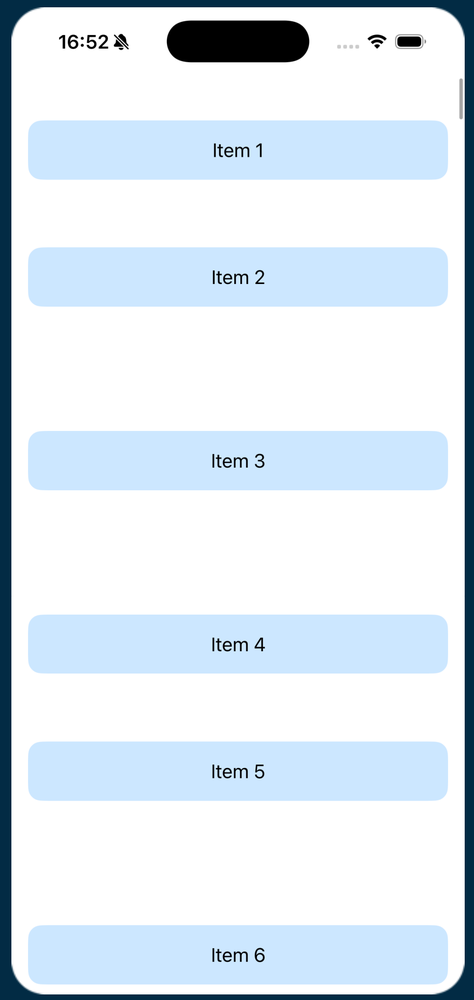
Очень долго :(

# Вопросы к LazyVStack вне GeometryReader

Медленный расчёт кадров

# Новый демо-пример

# Новый демо-пример




```
LazyVStack {
    Для 1…1000 {
        Ячейка
            .frame(height: index % 3 == 0 ? 200: 100)
    }
}
```

# Анализ OSLog для нового примера

Загружаем новый демо-проект: 1000 ячеек с разными размерами

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```

# Анализ OSLog для нового примера

Загружаем новый демо-проект: 1000 ячеек с разными размерами

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```

Пример последовательного лога для «быстрого скролла большого списка»:

```
00:13.674.301   LazyVStackLayout: placed(20054.6...20928.6) -> 534..<559, 20025.0...20975.0, invalid: false
00:13.692.178   LazyVStackLayout: placed(22712.0...23586.0) -> 605..<629, 22697.9482...23597.9482, invalid: false
```

# Анализ OSLog для нового примера

Загружаем новый демо-проект: 1000 ячеек с разными размерами

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```

Пример последовательного лога для «быстрого скролла большого списка»:

```
00:13.674.301  LazyVStackLayout: placed(20054.6...20928.6) -> 534..<559, 20025.0...20975.0, invalid: false
00:13.692.178  LazyVStackLayout: placed(22712.0...23586.0) -> 605..<629, 22697.9482...23597.9482, invalid: false
```

Чем меньше ячеек было показано на экране,
тем менее точную цифру выведет GeometryReader

**Результат GeometryReader непредсказуем :(**

# Вопросы к GeometryReader + LazyVStack

Почему GeometryReader возвращает **неверный** размер для LazyVStack?

Что за числа возвращал GeometryReader? Почему они невоспроизводимы в больших гетерогенных списках?

Вина на алгоритме «аппроксимации» размеров «списка» по «увиденным ранее» ячейкам

Почему нет «моргающего» экрана при вычислении неверного размера?

Как понять, что размер **«стабилизировался»**?

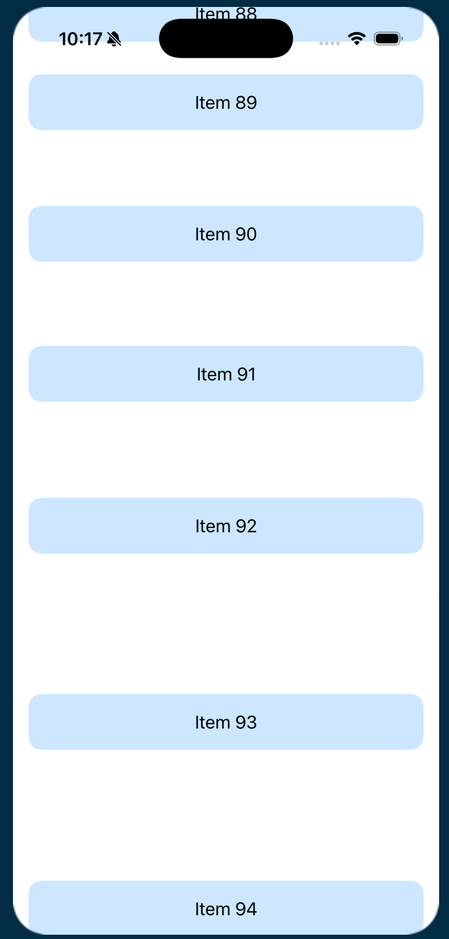
# Продолжаем анализ логов

Пример с большим кол-вом ячеек разной высоты

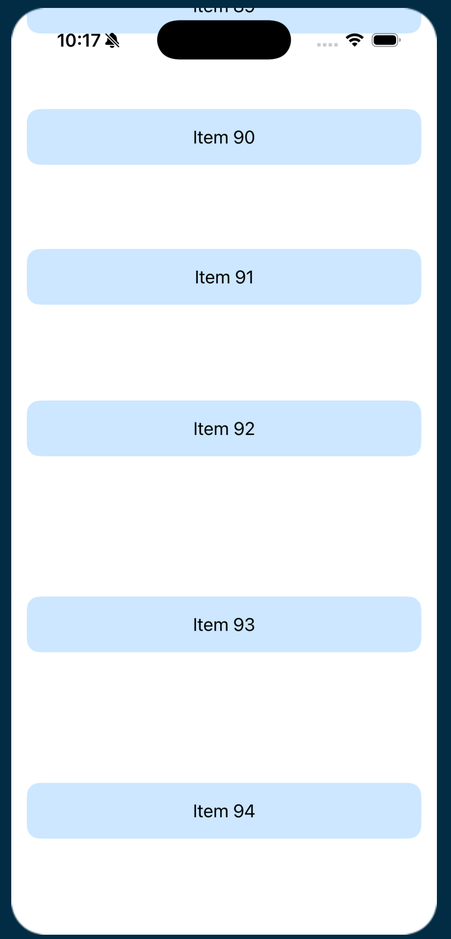
```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```

# Продолжаем анализ логов

Пример с большим кол-вом ячеек разной высоты

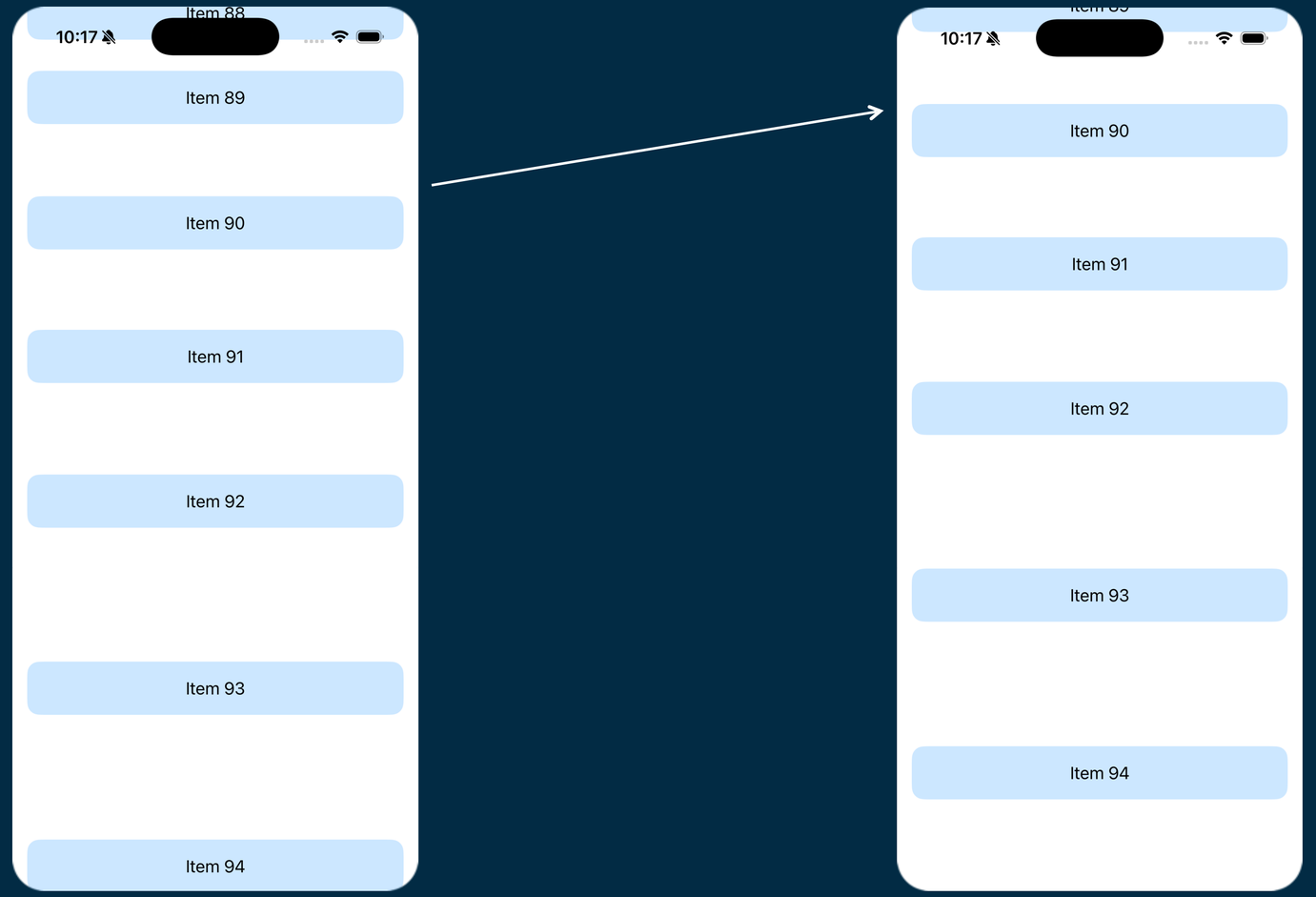



Скроллим
снизу-вверх



# Продолжаем анализ логов

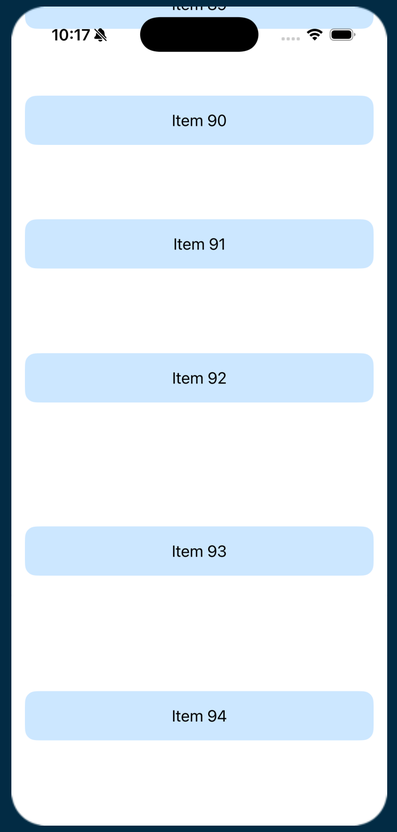
Пример с большим кол-вом ячеек разной высоты



# Продолжаем анализ логов

Пример с большим кол-вом ячеек разной высоты

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```

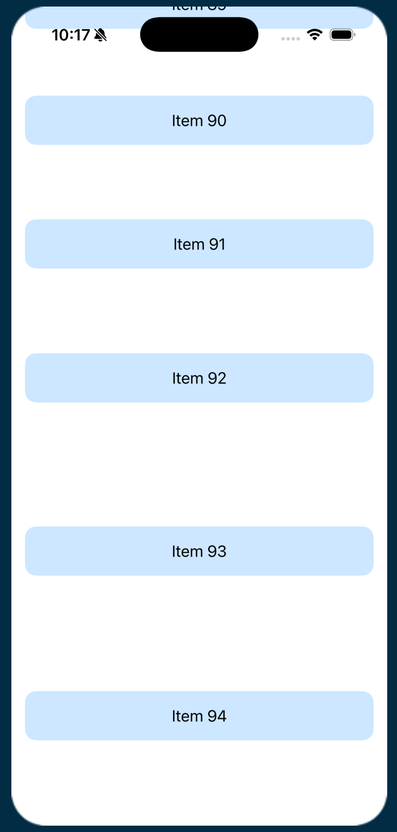



```
00:08.324.749  LazyVStackLayout: sizeThatFits(370.0) -> 13260.88
00:08.324.859  (StackPlacement in _*)<LazyVStackLayout>: placing from #89, 11549.38 in 11702.0...12576.0
00:08.324.949  LazyStack    LazyVStackLayout: placed(11702.0...12576.0) -> 89..<96, 11549.38...12725.43, invalid: false
```

# Продолжаем анализ логов

Пример с большим кол-вом ячеек разной высоты

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```




00:08.324.749 LazyVStackLayout: sizeThatFits(370.0) -> 13260.88

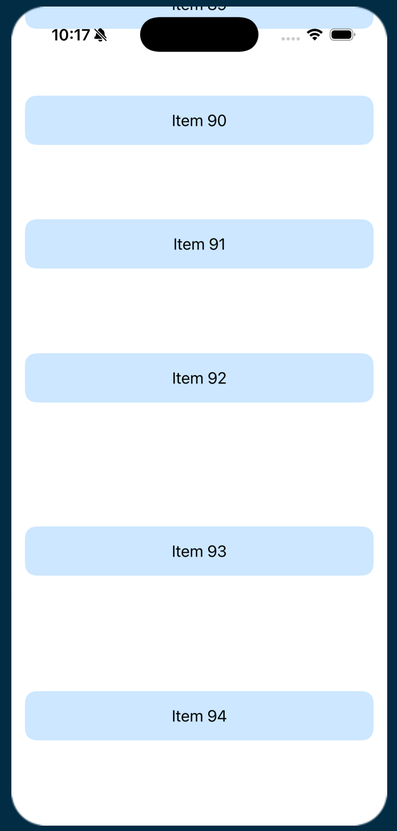
00:08.324.859 (StackPlacement in _*)<LazyVStackLayout>: placing from #89, 11549.38 in 11702.0...12576.0

00:08.324.949 LazyStack LazyVStackLayout: placed(11702.0...12576.0) -> 89..<96, 11549.38...12725.43, invalid: false

# Продолжаем анализ логов

Пример с большим кол-вом ячеек разной высоты

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```




00:08.324.749 LazyVStackLayout: sizeThatFits(370.0) -> 13260.88

Размер контента

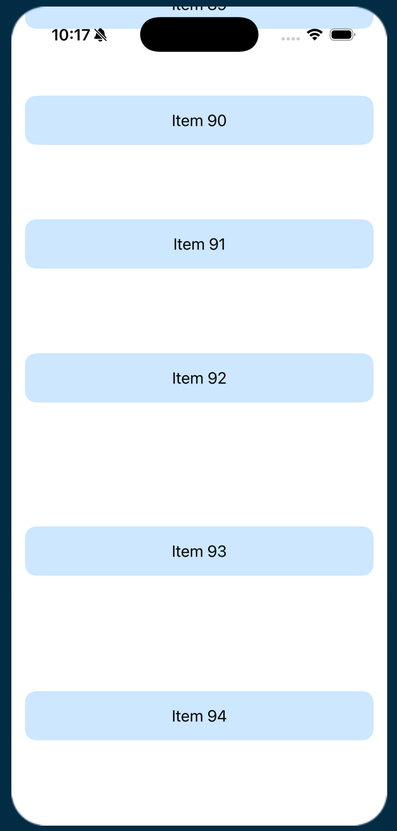
00:08.324.859 (StackPlacement in _*)<LazyVStackLayout>: placing from #89, 11549.38 in 11702.0...12576.0

00:08.324.949 LazyStack LazyVStackLayout: placed(11702.0...12576.0) -> 89..<96, 11549.38...12725.43, invalid: false

# Продолжаем анализ логов

Пример с большим кол-вом ячеек разной высоты

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```




00:08.324.749 LazyVStackLayout: sizeThatFits(370.0) -> 13260.88

Размер контента

00:08.324.859 (StackPlacement in _*)<LazyVStackLayout>: placing from #89, 11549.38 in 11702.0...12576.0

Индекс

Стартовая позиция установки элемента

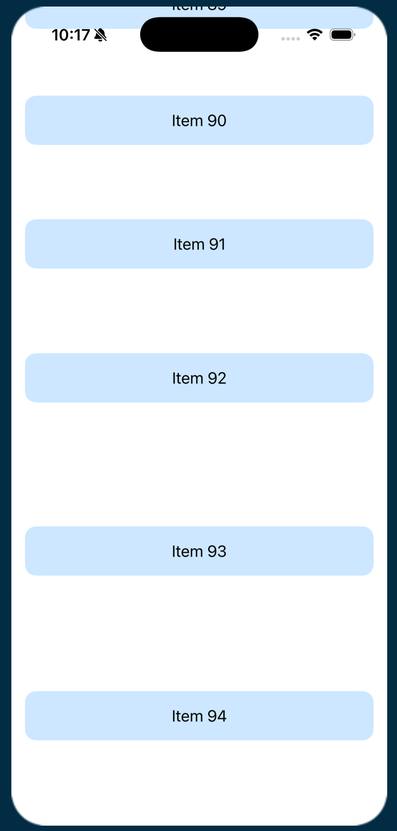
VisibleArea

00:08.324.949 LazyStack LazyVStackLayout: placed(11702.0...12576.0) -> 89..<96, 11549.38...12725.43, invalid: false

# Продолжаем анализ логов

Пример с большим кол-вом ячеек разной высоты

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```




00:08.324.749 LazyVStackLayout: sizeThatFits(370.0) -> 13260.88

Размер контента

00:08.324.859 (StackPlacement in _*)<LazyVStackLayout>: placing from #89, 11549.38 in 11702.0...12576.0

Индекс

Стартовая позиция установки элемента

VisibleArea

00:08.324.949 LazyStack LazyVStackLayout: placed(11702.0...12576.0) -> 89..<96, 11549.38...12725.43, invalid: false

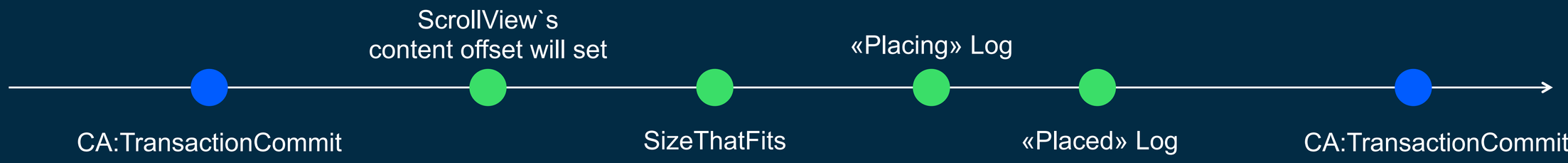
VisibleArea

Rendered Range

Расположение элементов

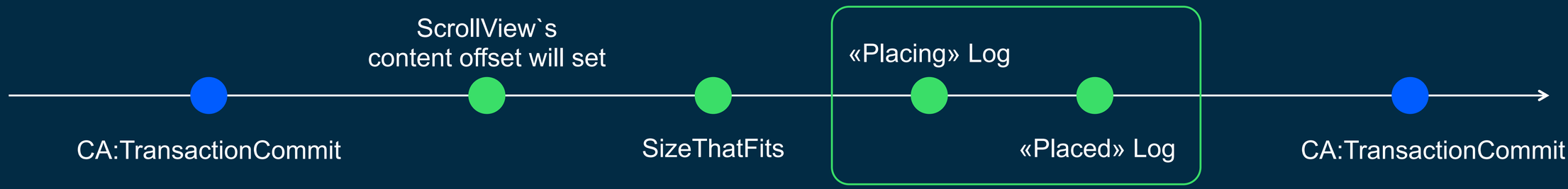
# Продолжаем анализ логов

Pipeline для формирования кадра



# Продолжаем анализ логов

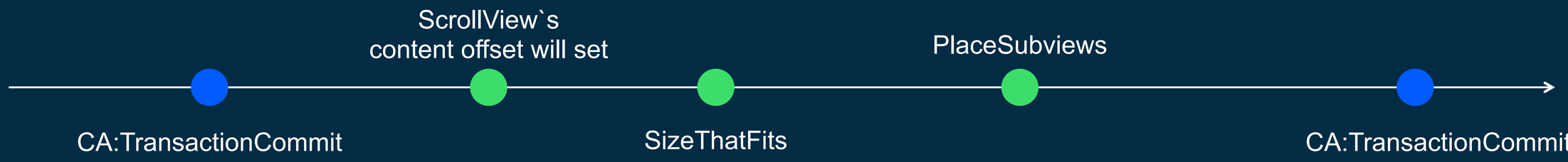
Pipeline для формирования кадра




SwiftUI.LazySubviewPlacements.placeSubviews(...) -> ():

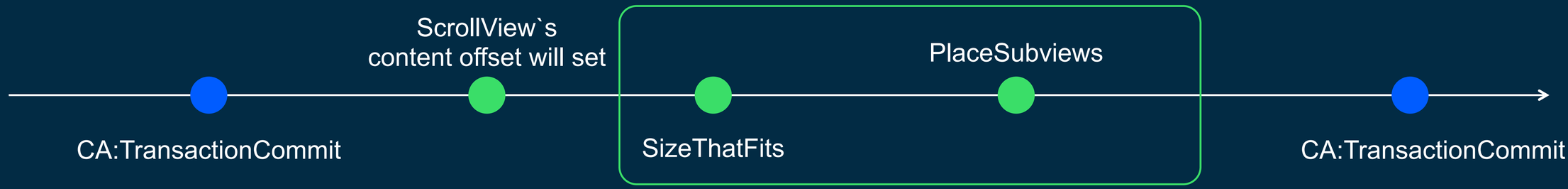
# Продолжаем анализ логов

Pipeline для формирования кадра



# Продолжаем анализ логов

Pipeline для формирования кадра




SwiftUI / Layout

Protocol

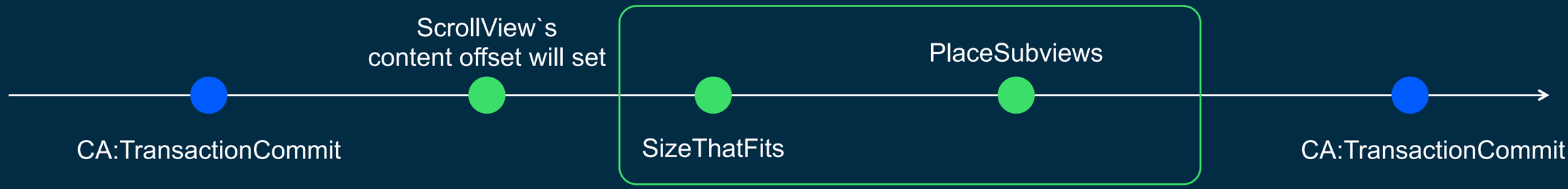
## Layout

A type that defines the geometry of a collection of views.

iOS 16.0+ | iPadOS 16.0+ | Mac Catalyst 16.0+ | macOS 13.0+ | tvOS 16.0+ | visionOS 1.0+ | watchOS 9.0+

# Продолжаем анализ логов

Pipeline для формирования кадра




SwiftUI / Layout

Protocol

## Layout

A type that defines the geometry of a collection of views.

iOS 16.0+ | iPadOS 16.0+ | Mac Catalyst 16.0+ | macOS 13.0+ | tvOS 16.0+ | visionOS 1.0+ | watchOS 9.0+

**Рискнём реализовать сами?**

# Сохраняемся

# Вопросы к GeometryReader + LazyVStack

Почему GeometryReader возвращает **неверный** размер для LazyVStack?

Что за числа возвращал GeometryReader?

Почему нет «моргающего» экрана при вычислении неверного размера?

Как понять, что размер **«стабилизировался»**?

# Вопросы к LazyVStack вне GeometryReader

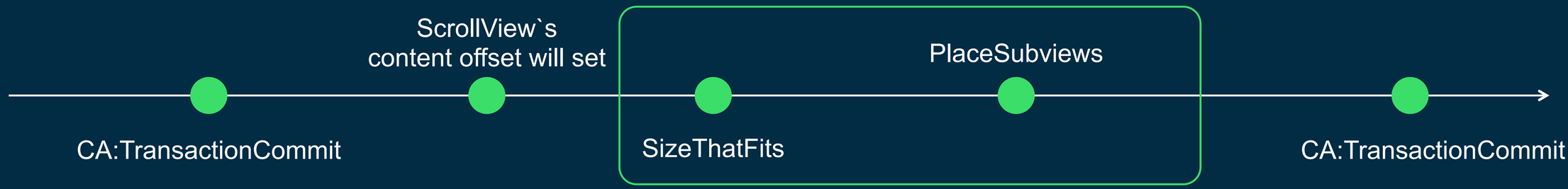
Медленный расчёт кадров

Реализация №1

# Сами реализуем SizeThatFits + PlaceSubview

# Реализация №1

Продолжим анализ логов




SwiftUI / Layout

Protocol

## Layout

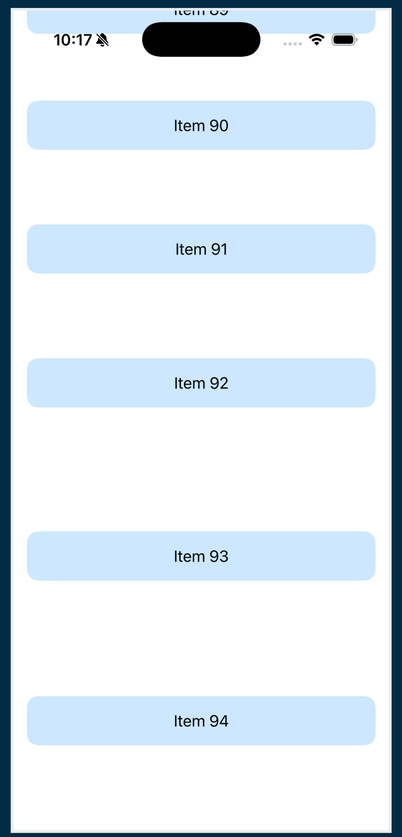
A type that defines the geometry of a collection of views.

iOS 16.0+ | iPadOS 16.0+ | Mac Catalyst 16.0+ | macOS 13.0+ | tvOS 16.0+ | visionOS 1.0+ | watchOS 9.0+

# Реализация №1

Продолжим анализ логов

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```




00:08.324.749 LazyVStackLayout: sizeThatFits(370.0) -> 13260.88

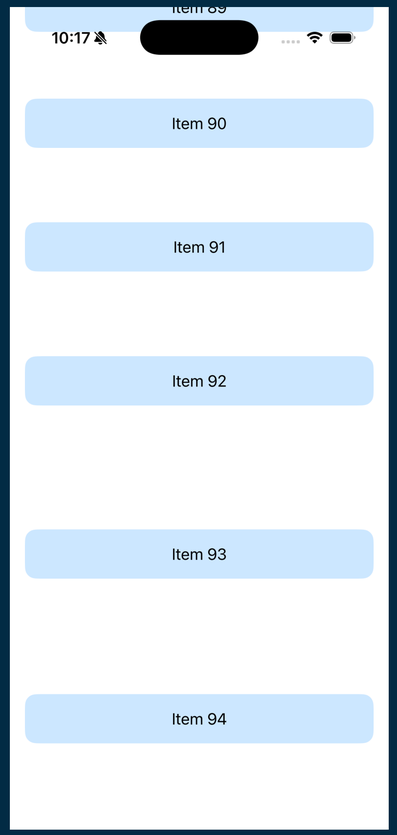
00:08.324.859 (StackPlacement in _*)<LazyVStackLayout>: placing from #89, 11549.38 in 11702.0...12576.0

00:08.324.949 LazyStack LazyVStackLayout: placed(11702.0...12576.0) -> 89..<96, 11549.38...12725.43, invalid: false

# Реализация №1

Продолжим анализ логов

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```




00:08.324.749 LazyVStackLayout: sizeThatFits(370.0) -> 13260.88

Оценка размеров контента

00:08.324.859 (StackPlacement in _*)<LazyVStackLayout>: placing from #89, 11549.38 in 11702.0...12576.0

Placing: Определение позиции стартовой отображаемой ячейки

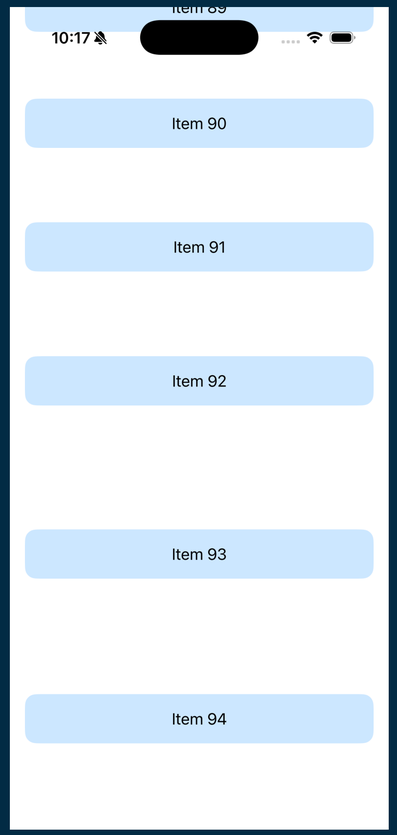
00:08.324.949 LazyStack LazyVStackLayout: placed(11702.0...12576.0) -> 89..<96, 11549.38...12725.43, invalid: false

Placed: Результат размещения ячеек в видимой области

# Реализация №1

Продолжим анализ логов

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```




00:08.324.749 LazyVStackLayout: sizeThatFits(370.0) -> 13260.88

Оценка размеров контента

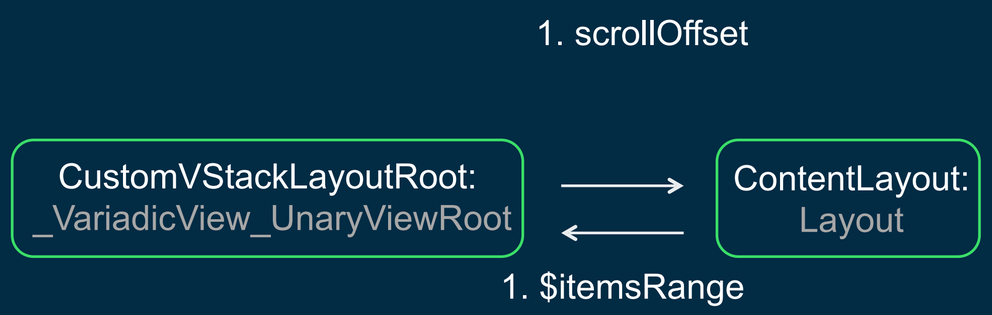
00:08.324.859 (StackPlacement in _*)<LazyVStackLayout>: placing from #89, 11549.38 in 11702.0...12576.0

Placing: Определение позиции стартовой отображаемой ячейки

00:08.324.949 LazyStack LazyVStackLayout: placed(11702.0...12576.0) -> 89..<96, 11549.38...12725.43, invalid: false

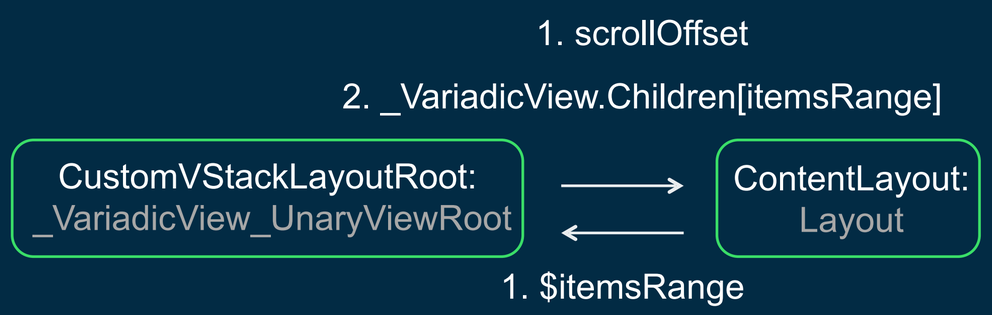
Placed: Результат размещения ячеек в видимой области

# Реализация №1



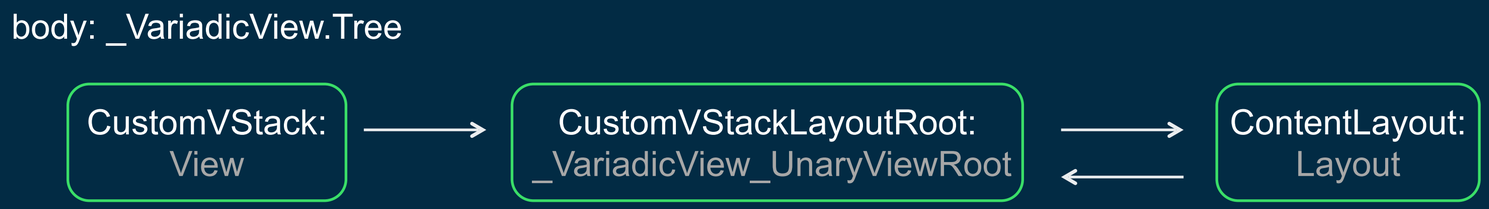

```
struct ContentLayout: Layout {
    sizeThatFits(…) {
        Заказываем itemsRange у CustomVStackLayoutRoot по
scrollOffset
    }



…


}
```

# Реализация №1

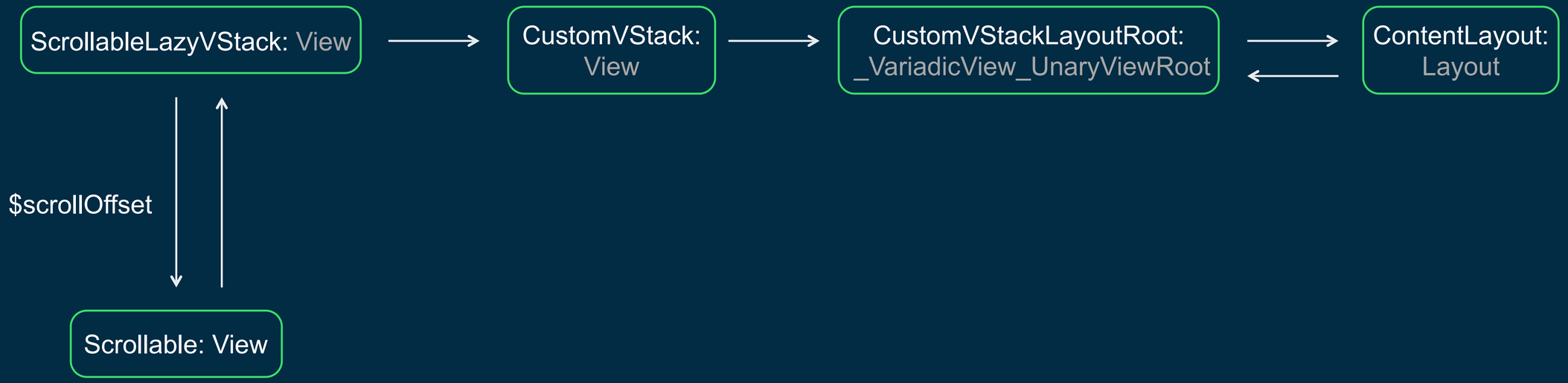



```
struct ContentLayout: Layout {
    sizeThatFits(…) {
        Заказываем itemsRange у CustomVStackLayoutRoot по
scrollOffset
    }

    placeSubviews(… subviews: Subviews …) {
        Размещаем subviews (itemsRange ячеек).
        Свободное пространство заполняем «пустотой»
    }
}
```

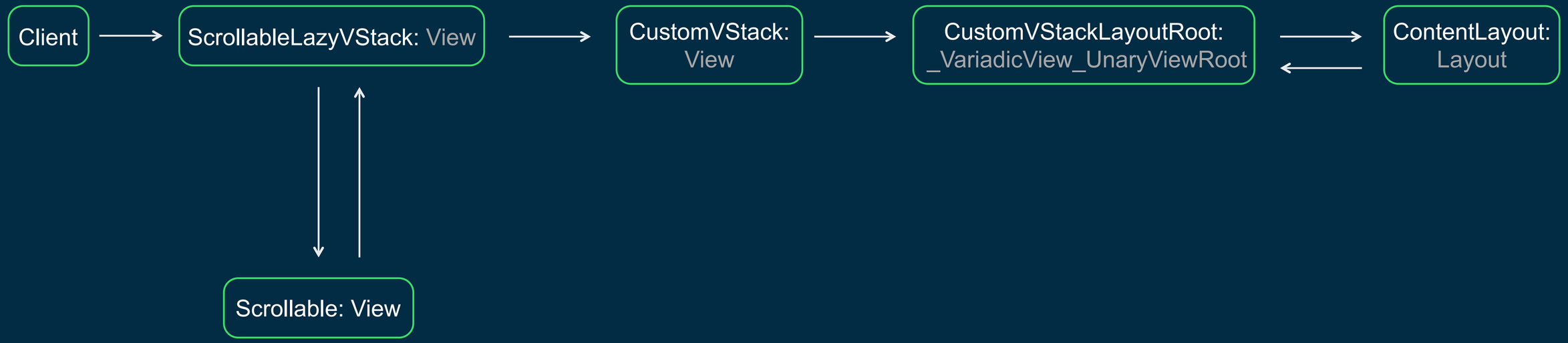
# Реализация №1



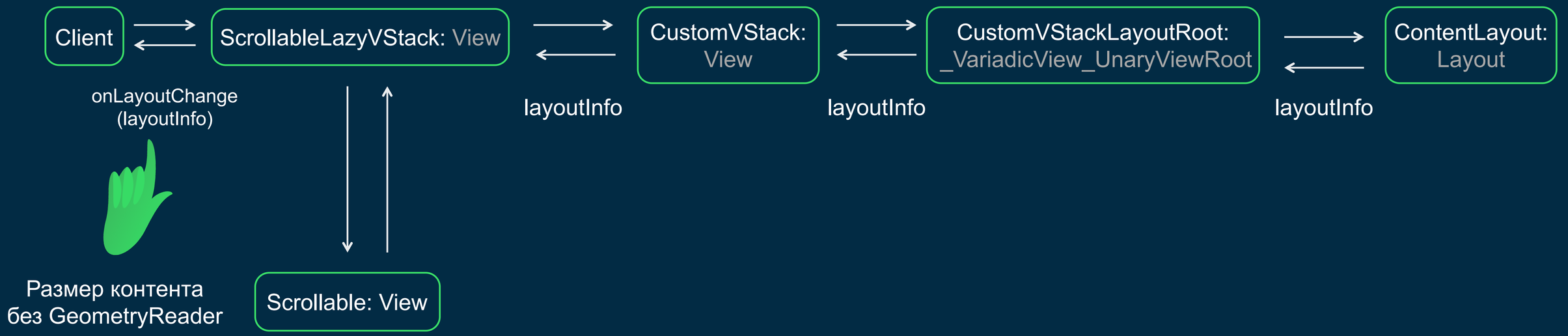
# Реализация №1



# Реализация №1



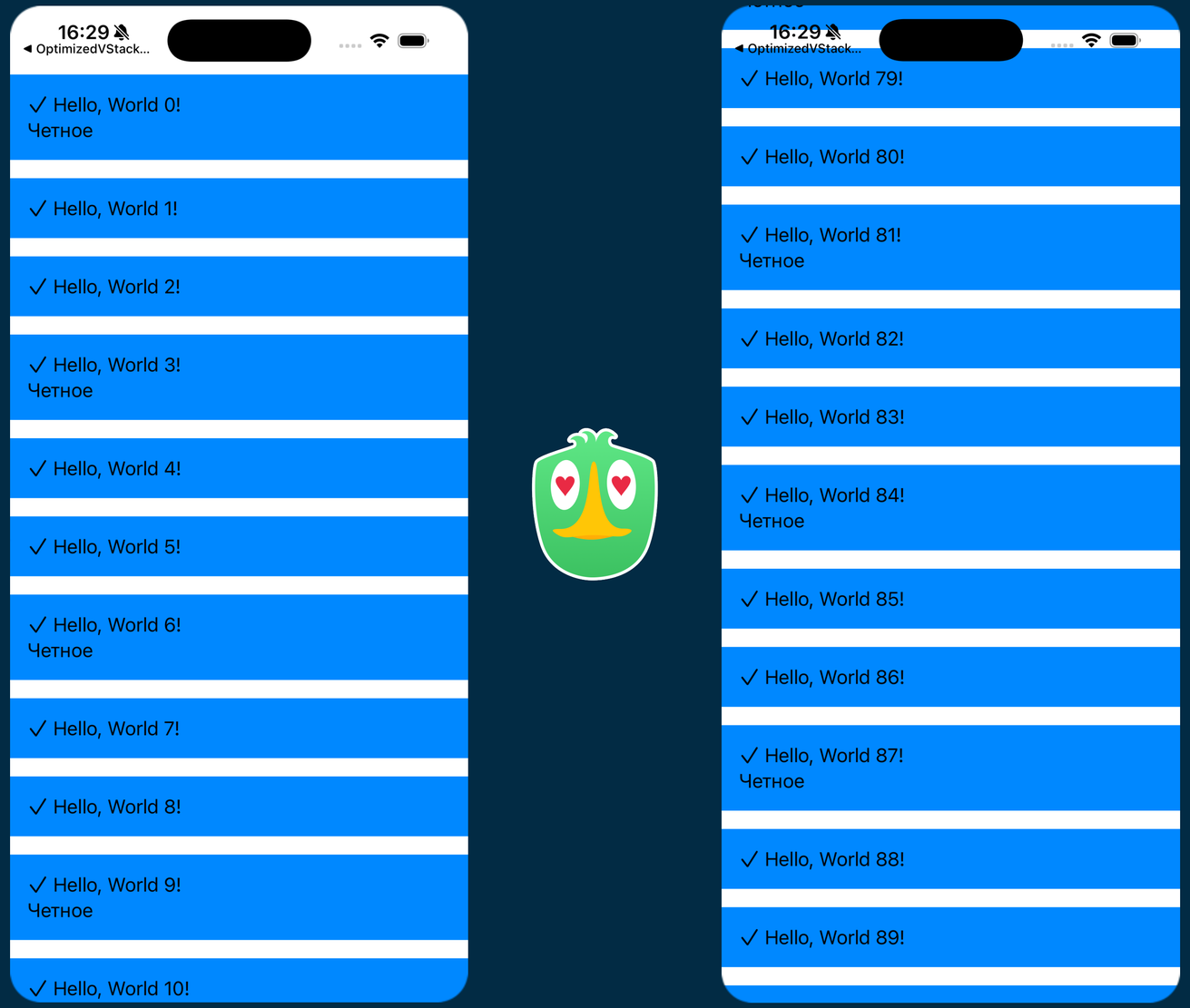
# Реализация №1



# Реализация №1

Пример списка ячеек

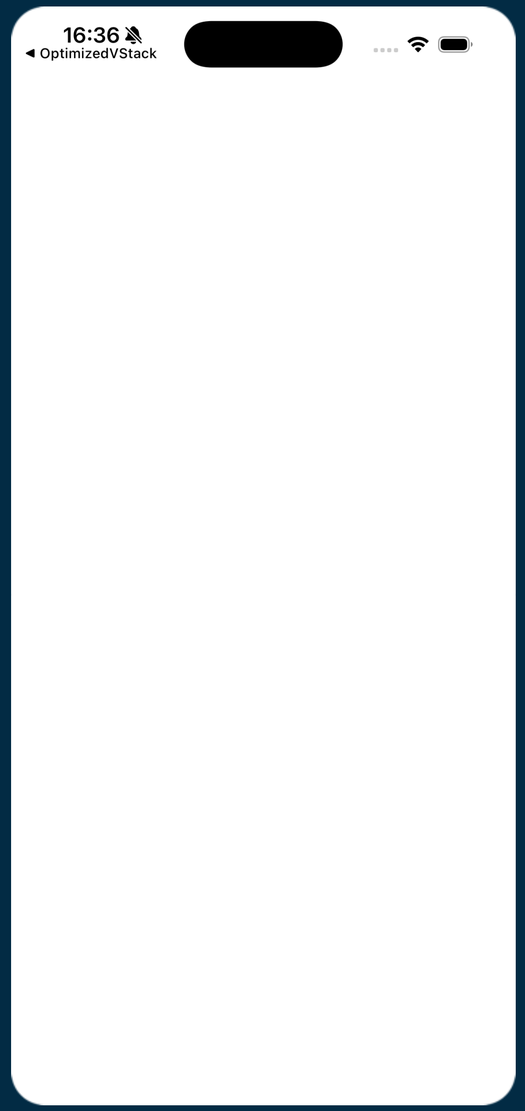
Slow scroll

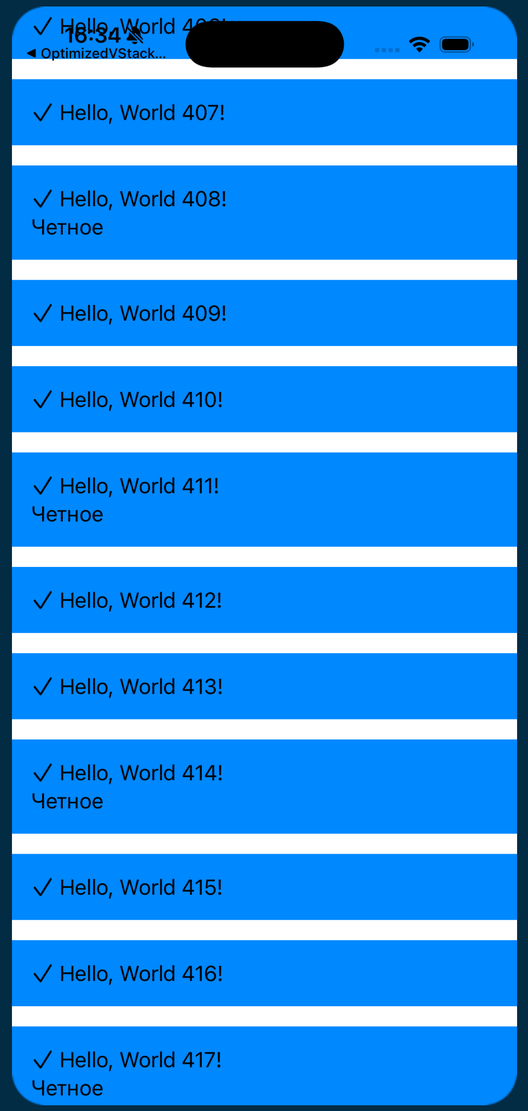


# Реализация №1

Пример списка ячеек

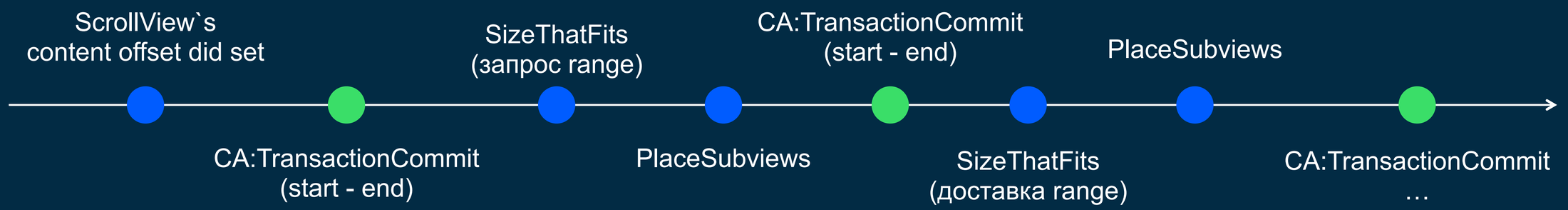
Fast scroll



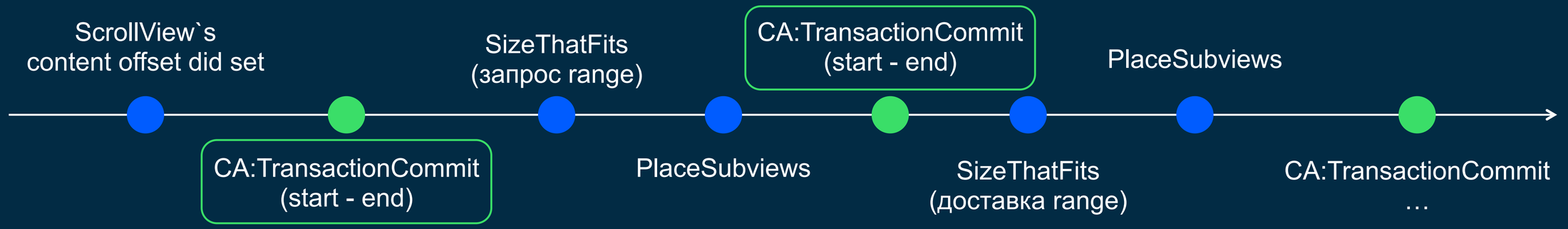
# Реализация №1

Почему «белый» экран?



# Реализация №1

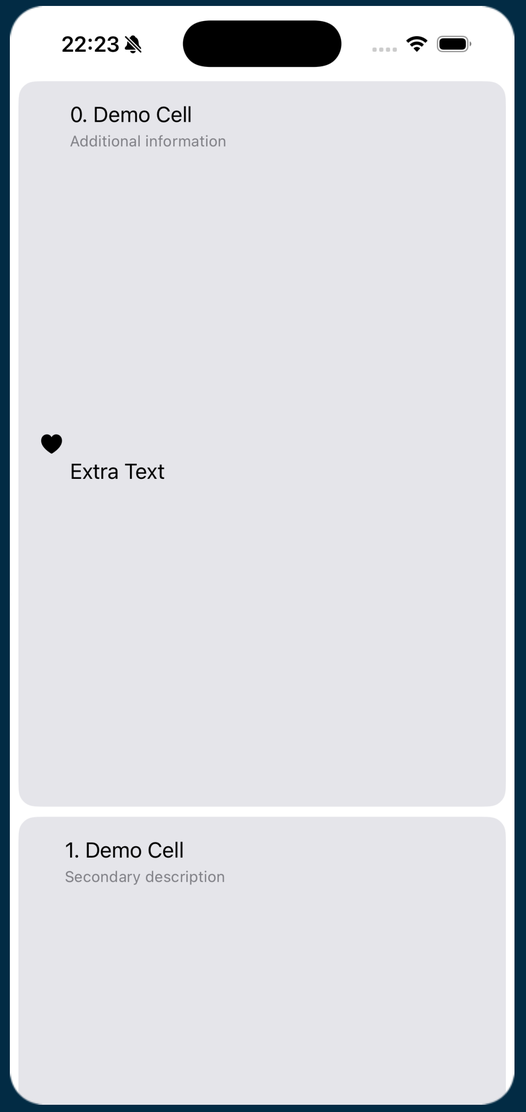
Почему «белый» экран?

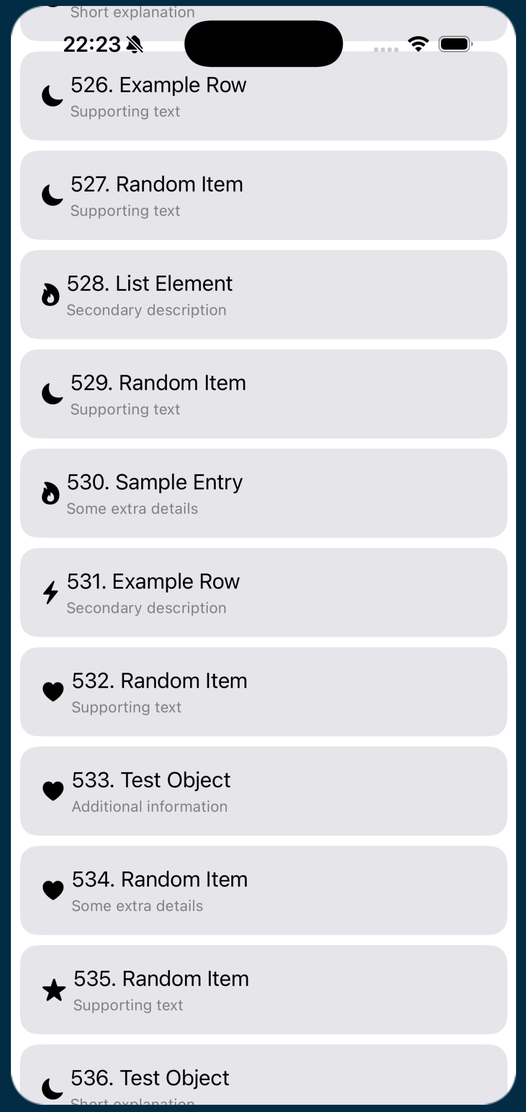


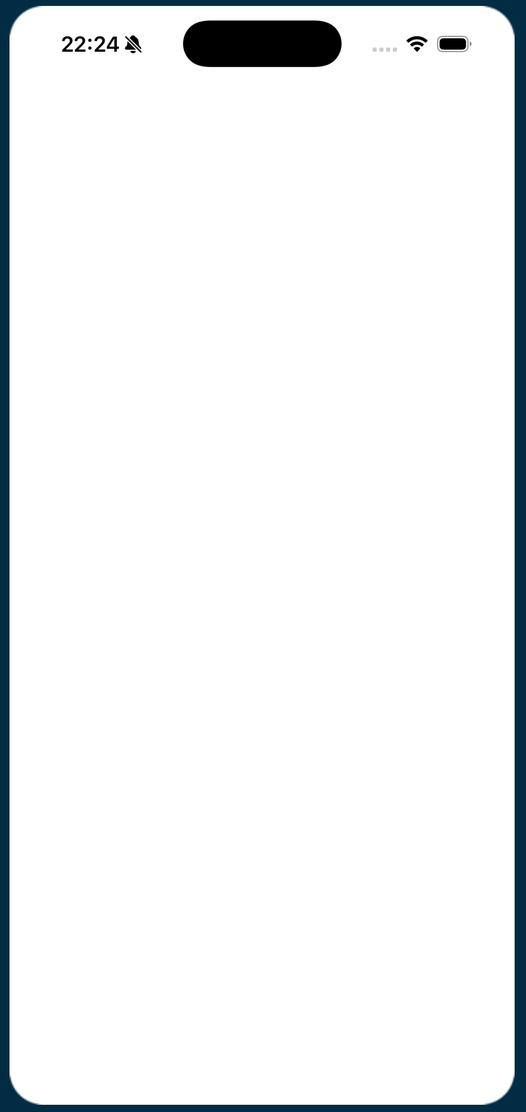

**Потеря кадров из-за реализации ScrollView’s content offset did set + низкой скорости Binding’s**

# Кто сталкивался с «белым» экраном у нативного LazyVStack?

# Проблема «белого» экрана LazyVStack



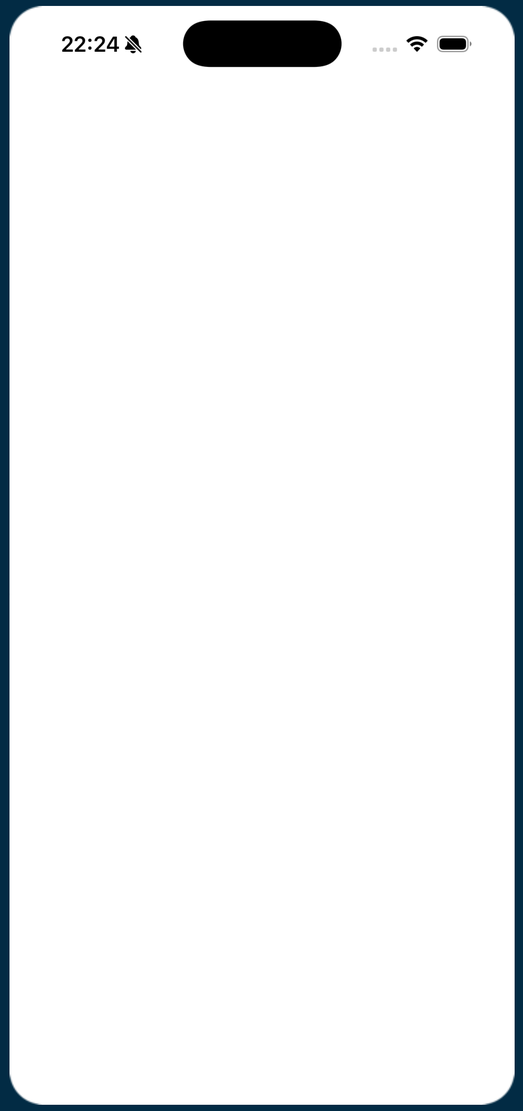




# Проблема «белого» экрана LazyVStack

При резком изменении visibleArea активируется следующая логика:

**Шаг 1. Расчёт в SwiftUICore`resolveIndexAndPosition:**

anchorIndex = (scrollOffset / cachedSizeThatFits) * totalRows



# Проблема «белого» экрана LazyVStack

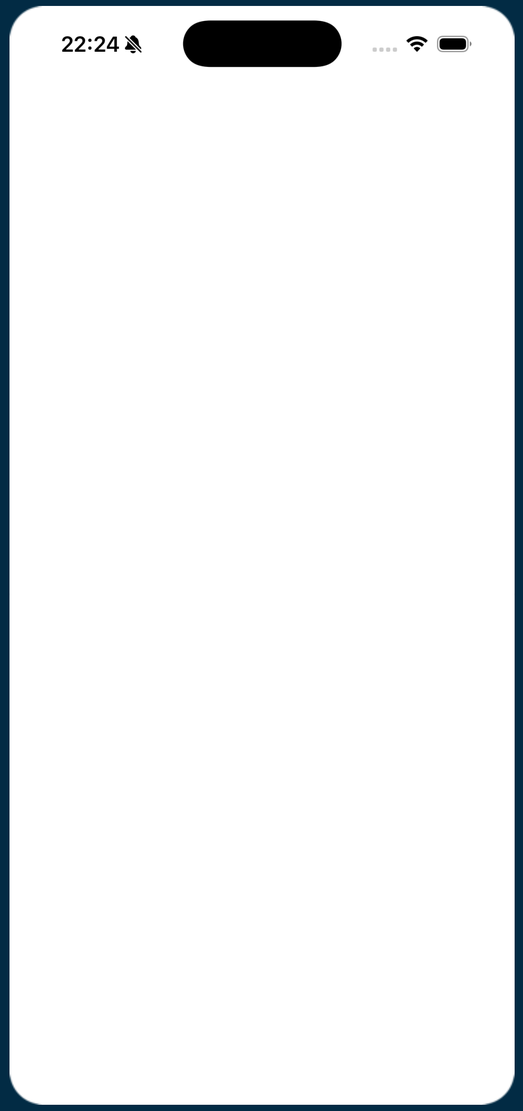
При резком изменении visibleArea активируется следующая логика:

**Шаг 1. Расчёт в SwiftUICore`resolveIndexAndPosition:**

anchorIndex = (scrollOffset / cachedSizeThatFits) * totalRows

**Шаг 2. Расчёт anchorPosition в SwiftUICore`resolveIndexAndPosition:**

anchorPosition = anchorIndex * (estimatedCellSize + spacing)



# Проблема «белого» экрана LazyVStack

При резком изменении visibleArea активируется следующая логика:

**Шаг 1.** **Расчёт в SwiftUICore`resolveIndexAndPosition:**

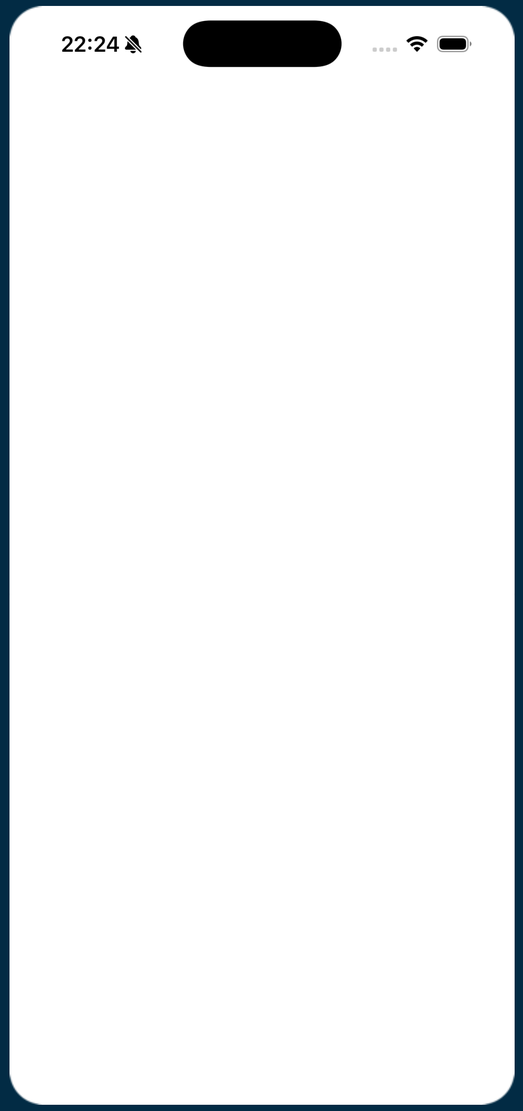
anchorIndex = (scrollOffset / cachedSizeThatFits) * totalRows

**Шаг 2.** **Расчёт anchorPosition в SwiftUICore`resolveIndexAndPosition:**

anchorPosition = anchorIndex * (estimatedCellSize + spacing)

**Шаг 3.** **Поиск месторасположения первой ячейки в visibleArea**

Пересчет yMaxLastCell (максимальная Y координата посл. ячейки)



# Проблема «белого» экрана LazyVStack

При резком изменении visibleArea активируется следующая логика:

**Шаг 1.** **Расчёт в SwiftUICore`resolveIndexAndPosition:**

anchorIndex = (scrollOffset / cachedSizeThatFits) * totalRows

**Шаг 2.** **Расчёт anchorPosition в SwiftUICore`resolveIndexAndPosition:**

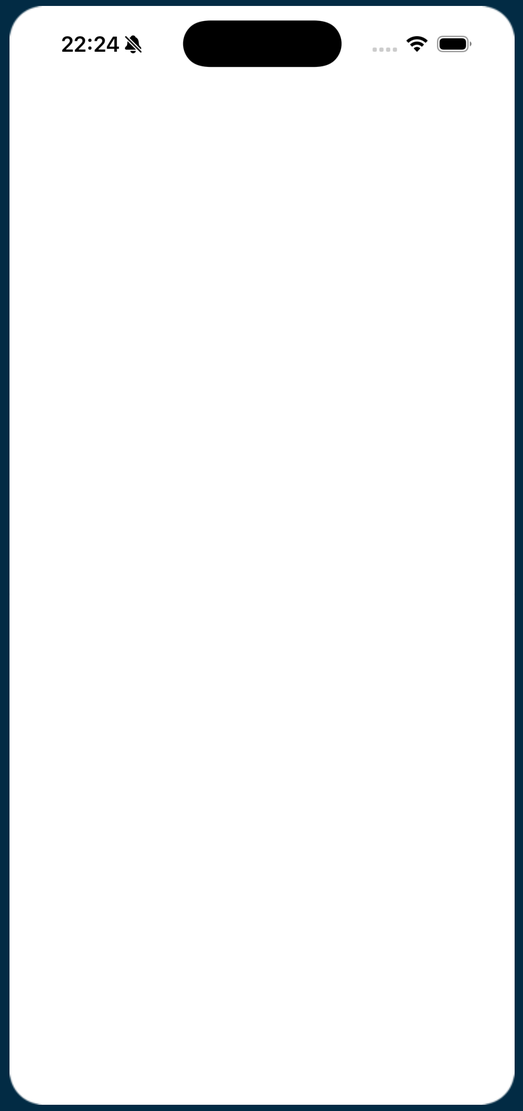
anchorPosition = anchorIndex * (estimatedCellSize + spacing)

**Шаг 3.** **Поиск месторасположения первой ячейки в visibleArea**

Пересчет yMaxLastCell (максимальная Y координата посл. ячейки)

**Шаг 4.** **Проверка валидности позиций установленных элементов**

Размеры yMaxLastCell И cachedSizeThatFits различаются
НЕ более чем на 10%?



# Проблема «белого» экрана LazyVStack

При резком изменении visibleArea активируется следующая логика:

**Шаг 1.** **Расчёт в SwiftUICore`resolveIndexAndPosition:**

anchorIndex = (scrollOffset / cachedSizeThatFits) * totalRows

**Шаг 2.** **Расчёт anchorPosition в SwiftUICore`resolveIndexAndPosition:**

anchorPosition = anchorIndex * (estimatedCellSize + spacing)

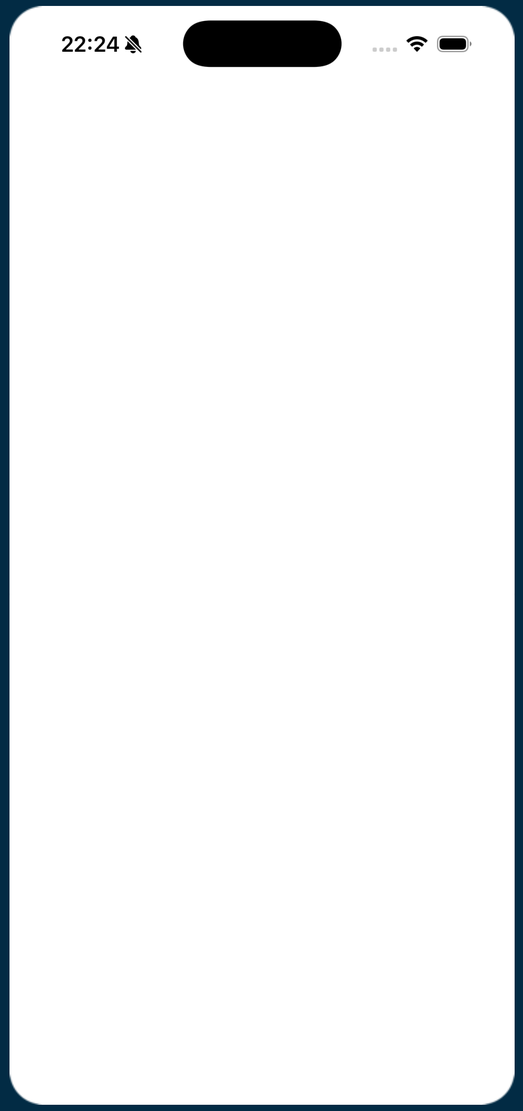
**Шаг 3.** **Поиск месторасположения первой ячейки в visibleArea**

Пересчет yMaxLastCell (максимальная Y координата посл. ячейки)

**Шаг 4.** **Проверка валидности позиций установленных элементов**

Размеры yMaxLastCell И cachedSizeThatFits различаются
НЕ более чем на 10%?

**ДА** => коммитим CA транзакцию
**НЕТ** => cachedSizeThatFits = SizeThatFits(), к шагу 1




# Проблема «белого» экрана LazyVStack

«Невалидное расположение»

```
00:25.103.058  placing from #646, 296 983.93 in 328 499.0...329 373.0
00:25.228.387  invalid #3; 324 831.93 vs 507 631.0
00:25.228.668  placed(328 499.0...329 373.0) -> 1000..<1000, 324 831.93...324 831.93, invalid: true
00:25:229:311  sizeThatFits…
```

# Проблема «белого» экрана LazyVStack

«Невалидное расположение»

```
00:25.103.058  placing from #646, 296 983.93 in 328 499.0...329 373.0
00:25.228.387  invalid #3; 324 831.93 vs 507 631.0
00:25.228.668  placed(328 499.0...329 373.0) -> 1000..<1000, 324 831.93...324 831.93, invalid: true
00:25:229:311  sizeThatFits…
```

«Валидное расположение»

```
00:25.291.182  placing from #998, 176 568.95 in 328 499.0...329 373.0
00:25.291.585  placed(328 499.0...329 373.0) -> 1000..<1000, 324 831.93...324 831.934, invalid: false
00:27.291.852  Transaction Commit
```

White Screen

# Момент, когда описанная схема ломается :(

«Невалидное расположение»

```
00:25.103.058  placing from #646, 296 983.93 in 328 499.0...329 373.0
00:25.228.387  invalid #3; 324 831.93 vs 507 631.0
00:25.228.668  placed(328 499.0...329 373.0) -> 1000..<1000, 324 831.93...324 831.93, invalid: true
00:25:229:311  sizeThatFits…
```

anchorPosition = anchorIndex * (estimatedCellSize + spacing)

«Резкое» падение значения

«Валидное расположение»

```
00:25.291.182  placing from #998, 176 568.95 in 328 499.0...329 373.0
00:25.291.585  placed(328 499.0...329 373.0) -> 1000..<1000, 324 831.93...324 831.934, invalid: false
00:27.291.852  Transaction Commit
```

# Что не так с алгоритмом расчёта кадра LazyVStack?

«Невалидное расположение»

00:25.103.058 placing from #646, 296 983.93 in 328 499.0...329 373.0
00:25.228.387 invalid #3; 324 831.93 vs 5 07 631.0
00:25.228.668 placed(328 499.0...329 [illegible].0) -> 1000..<1000, 324 831.93...324 831.93, invalid: true
00:25:229:311 sizeThatFits…

1. В кэше 646…1000 ячейки => нагрузка на RAM

# Что не так с алгоритмом расчёта кадра LazyVStack?

«Невалидное расположение»

```
00:25.103.058  placing from #646, 296 983.93 in 328 499.0...329 373.0
00:25.228.387  invalid #3; 324 831.93 vs 5 07 631.0
00:25.228.668  placed(328 499.0...329 [illegible].0) -> 1000..<1000, 324 831.93...324 831.93, invalid: true
00:25:229:311  sizeThatFits…
```

1. В кэше 646…1000 ячейки => нагрузка на RAM

2. Возможные 250 мс «белого» экрана (CA не считает это за hitch!)

# Что не так с алгоритмом расчёта кадра LazyVStack?

«Невалидное расположение»

```
00:25.103.058  placing from #646, 296 983.93 in 328 499.0...329 373.0
00:25.228.387  invalid #3; 324 831.93 vs 5 07 631.0
00:25.228.668  placed(328 499.0...329 [illegible].0) -> 1000..<1000, 324 831.93...324 831.93, invalid: true
00:25:229:311  sizeThatFits…
```

1. В кэше 646…1000 ячейки => нагрузка на RAM

2. Возможные 250 мс «белого» экрана (CA не считает это за hitch!)

3. Если в вашем стеке «сверху — большие ячейки, снизу — маленькие ячейки» — ждите беды

# Вопросы к GeometryReader + LazyVStack

Почему GeometryReader возвращает **неверный** размер для LazyVStack?

Что за числа возвращал GeometryReader?

Почему нет «моргающего» экрана при вычислении неверного размера?

Алгоритм расчёта кадра понимает, что не все транзакции с рассчитанными размерами подлежат коммиту

Как понять, что размер **«стабилизировался»**?

# Вспомним пример с GeometryReader


```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```

| Timestamp | Type | Proc... | Subsystem | | Message |
|---|---|---|---|---|---|
| 00:00.817.498 | Defa... | Opti... | com.apple.SwiftU | | LazyVStackLayout: sizeThatFits(370.0) -> 189.0 |
| 00:00.818.514 | Defa... | Opti... | com.apple.SwiftU | | LazyVStackLayout: Measured estimates -> 41.0 |
| 00:00.818.617 | Defa... | Opti... | com.apple.SwiftUI | | (StackPlacement in _973C9973BC16DEAF0CF3109FFDE31321)<LazyVStackLayout>: placing from #0, 0.0 in 0.0...796.0 |
| 00:00.819.210 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: invalid #2; 1054.0 vs 189.0 |

Не коммитим

...

| | | | | | |
|---|---|---|---|---|---|
| 00:00.848.180 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | (StackPlacement in _973C9973BC16DEAF0CF3109FFDE31321)<LazyVStackLayout>: placing from #0, 0.0 in 0.0...796.0 |
| 00:00.848.239 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: placed(0.0...796.0) -> 0..<5, 0.0...1054.0, invalid: false |

# Вспомним пример с GeometryReader


```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```

| Timestamp | Type | Proc... | Subsystem | | Message |
|---|---|---|---|---|---|
| 00:00.817.498 | Defa... | Opti... | com.apple.SwiftU | | LazyVStackLayout: sizeThatFits(370.0) -> 189.0 |
| 00:00.818.514 | Defa... | Opti... | com.apple.SwiftU | | LazyVStackLayout: Measured estimates -> 41.0 |
| 00:00.818.617 | Defa... | Opti... | com.apple.SwiftUI | | (StackPlacement in _973C9973BC16DEAF0CF3109FFDE31321)<LazyVStackLayout>: placing from #0, 0.0 in 0.0...796.0 |
| 00:00.819.210 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: invalid #2; 1054.0 vs 189.0 |

Не коммитим

…

| | | | | | |
|---|---|---|---|---|---|
| 00:00.848.180 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | (StackPlacement in _973C997 …31321)<LazyVStackLayout>: placing from #0, 0.0 in 0.0...796.0 |
| 00:00.848.239 | Defa... | Opti... | com.apple.SwiftUI | LazyStack | LazyVStackLayout: placed(0.0… …1054.0, invalid: false |

Признак стабилизации размера

# Вопросы к LazyVStack вне GeometryReader

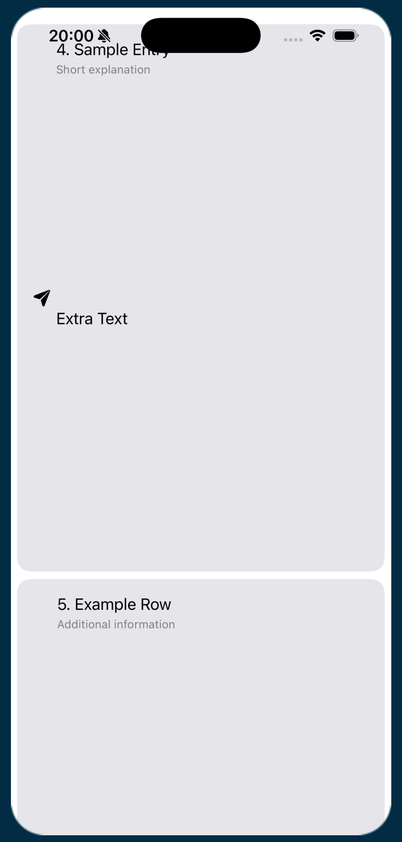
Медленный расчёт кадров

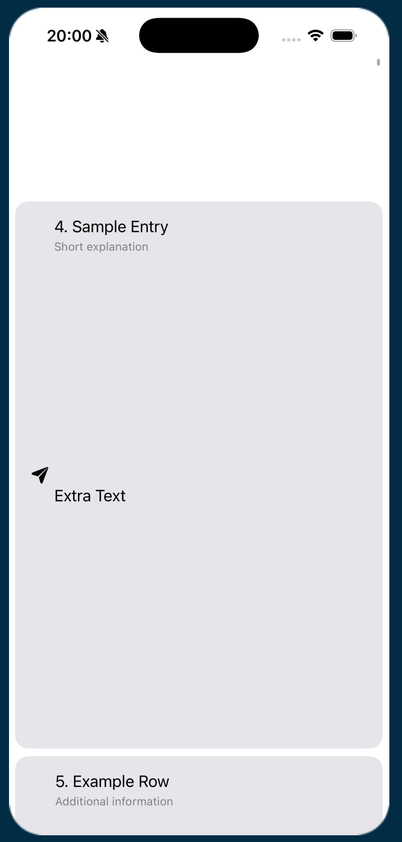
Преждевременное заполнение кэша размеров ячеек

«White screen» в больших гетерогенных списках

# Еще одна проблема LazyVStack

## Почему начало списка «обрезалось»?



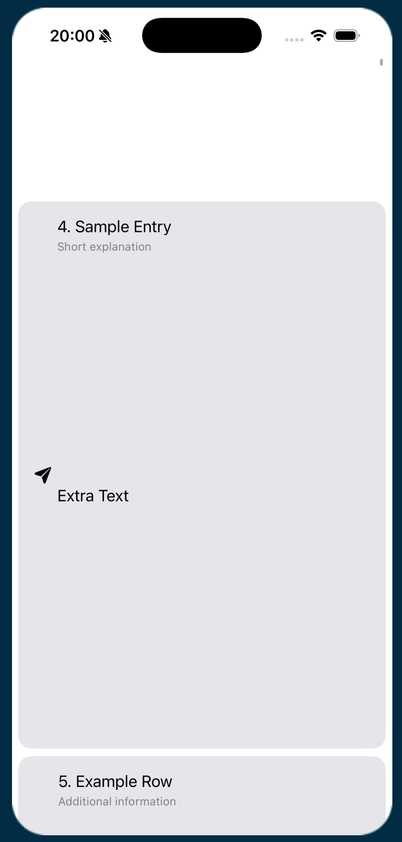
# Еще одна проблема LazyVStack

## Почему начало списка «обрезалось»?

00:30.532.290 (StackPlacement in _*)<LazyVStackLayout>: placing from #4, -562.80 in 0.0...413.66

00:30.532.496 LazyVStackLayout: translate by 486.4769759450171

00:30.532.558 LazyVStackLayout: placed(0.0...413.66) -> 4..<6, -76.32...1089.00, invalid: false



# Еще одна проблема LazyVStack

## Почему начало списка «обрезалось»?

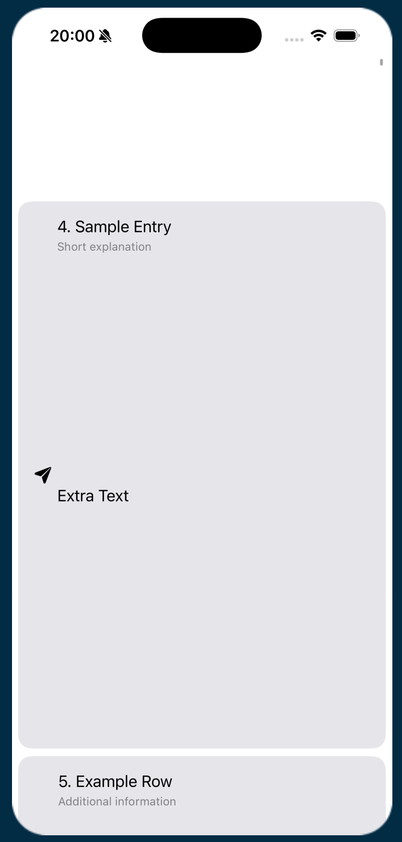
00:30.532.290 (StackPlacement in _*)<LazyVStackLayout>: placing from #4, -562.80 in 0.0...413.66

Отрицательная стартовая позиция установки ячеек при скролле вверх

(т.е. к 1-му элементу списка)

00:30.532.496 LazyVStackLayout: translate by 486.4769759450171

00:30.532.558 LazyVStackLayout: placed(0.0...413.66) -> 4..<6, -76.32...1089.00, invalid: false



# Еще одна проблема LazyVStack

## Почему начало списка «обрезалось»?

00:30.532.290 (StackPlacement in _*)<LazyVStackLayout>: placing from #4, -562.80 in 0.0...413.66

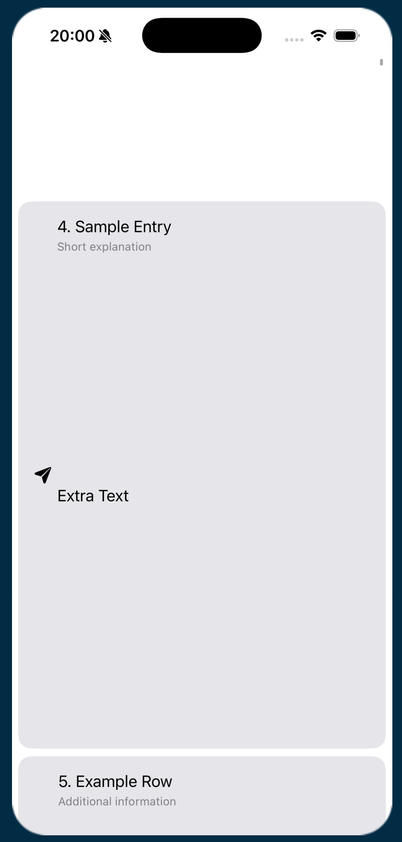
Отрицательная стартовая позиция установки ячеек при скролле вверх

(т.е. к 1-му элементу списка)

00:30.532.496 LazyVStackLayout: translate by 486.4769759450171

Подвинуть контент вниз на сумму estimate ячеек №0…№3

00:30.532.558 LazyVStackLayout: placed(0.0...413.66) -> 4..<6, -76.32...1089.00, invalid: false



# Еще одна проблема LazyVStack

## Почему начало списка «обрезалось»?

00:30.532.290 (StackPlacement in _*)<LazyVStackLayout>: placing from #4, -562.80 in 0.0...413.66

Отрицательная стартовая позиция установки ячеек при скролле вверх

(т.е. к 1-му элементу списка)

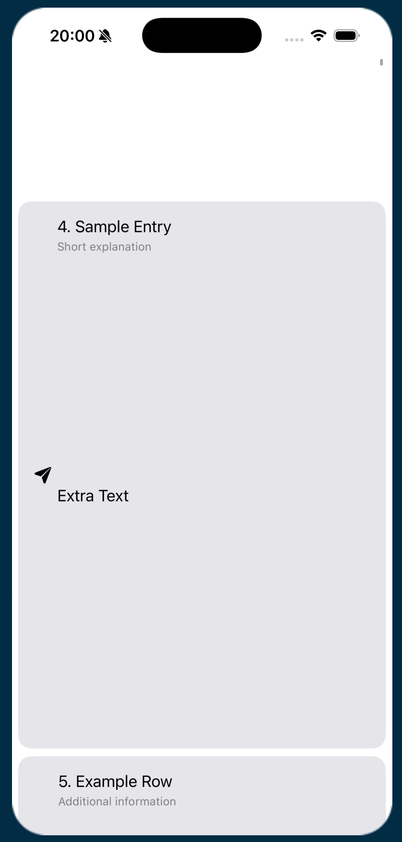
00:30.532.496 LazyVStackLayout: translate by 486.4769759450171

Подвинуть контент вниз на сумму estimate ячеек №0…№3

00:30.532.558 LazyVStackLayout: placed(0.0...413.66) -> 4..<6, -76.32...1089.00, invalid: false

«Отрицательный старт» 4-го элемента

Обрезка ячеек №0…№3



# Еще одна проблема LazyVStack

## Почему начало списка «обрезалось»?

00:30.532.290 (StackPlacement in _*)<LazyVStackLayout>: placing from #4, -562.80 in 0.0...413.66

Отрицательная стартовая позиция установки ячеек при скролле вверх

(т.е. к 1-му элементу списка)

Но ведь известен их точный размер…

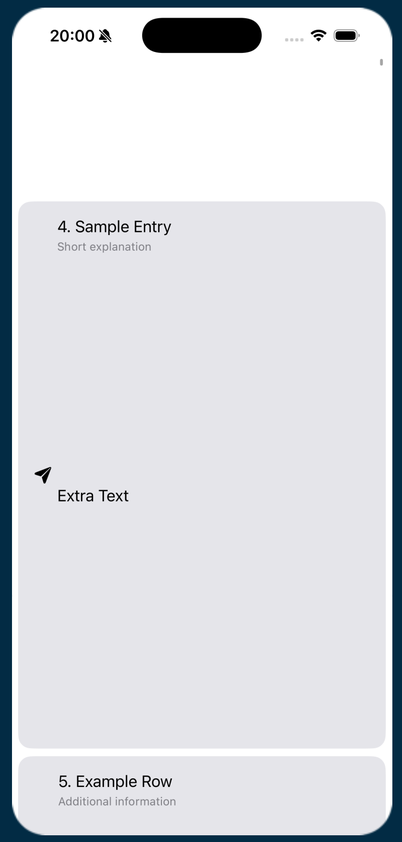
ayout: translate by 486.4769759450171

Подвинуть контент вниз на сумму estimate ячеек №0…№3

00:30.532.558 LazyVStackLayout: placed(0.0...413.66) -> 4..<6, -76.32...1089.00, invalid: false

«Отрицательный старт» 4-го элемента

Обрезка ячеек №0…№3




Идея «перемещения контента», явно, не доведена до конца

# Вопросы к LazyVStack вне GeometryReader

Медленный расчёт кадров

Преждевременное заполнение кэша размеров ячеек

Неожиданная «обрезка» списка

«White screen» в больших гетерогенных списках

Реализация №2

# Прибегаем к помощи UIViewRepresentable

# Реализация №2

UIViewRepresentable
Основа: UIScrollView

# Реализация №2

UIViewRepresentable
Основа: UIScrollView


Изменение scrollOffset

# Реализация №2

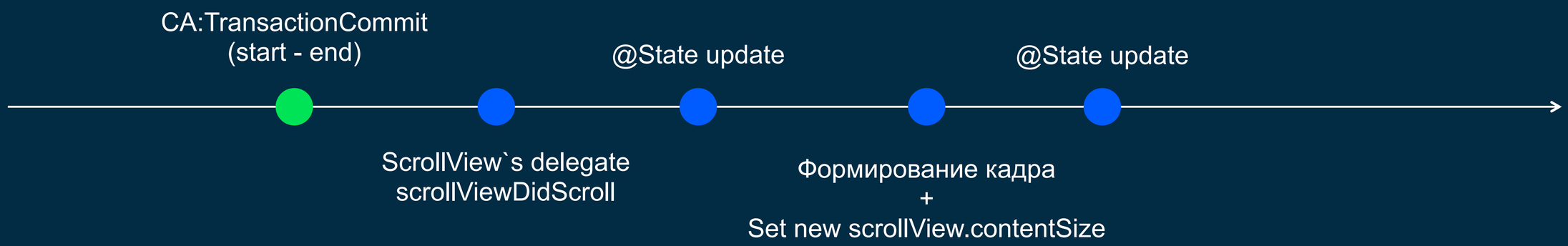
UIViewRepresentable
Основа: UIScrollView


Изменение scrollOffset

1) Определяем range ячеек в VisibleArea
2) Устанавливаем ячейки
3) Set new scrollView.contentSize

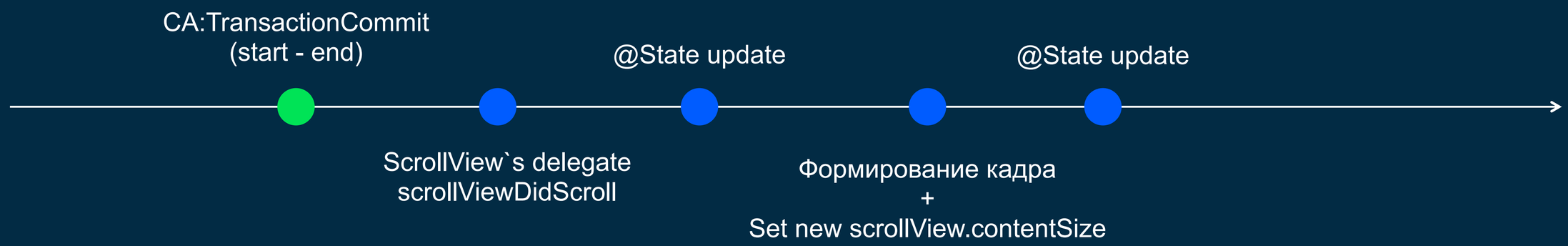
# Реализация №2

UIViewRepresentable
Основа: UIScrollView



# Реализация №2

UIViewRepresentable
Основа: UIScrollView

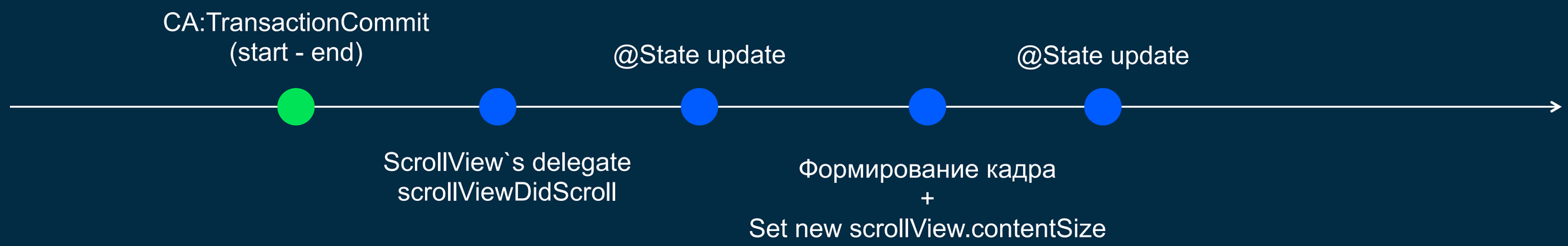



Проблемы

1) CA:TransactionCommit выполняется «перед» scrollViewDidScroll
Следствие: моргающий «white» screen

# Реализация №2

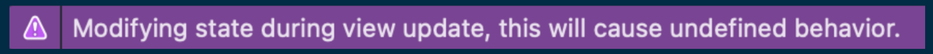
UIViewRepresentable
Основа: UIScrollView




Проблемы

1) CA:TransactionCommit выполняется «перед» scrollViewDidScroll
Следствие: моргающий «white» screen

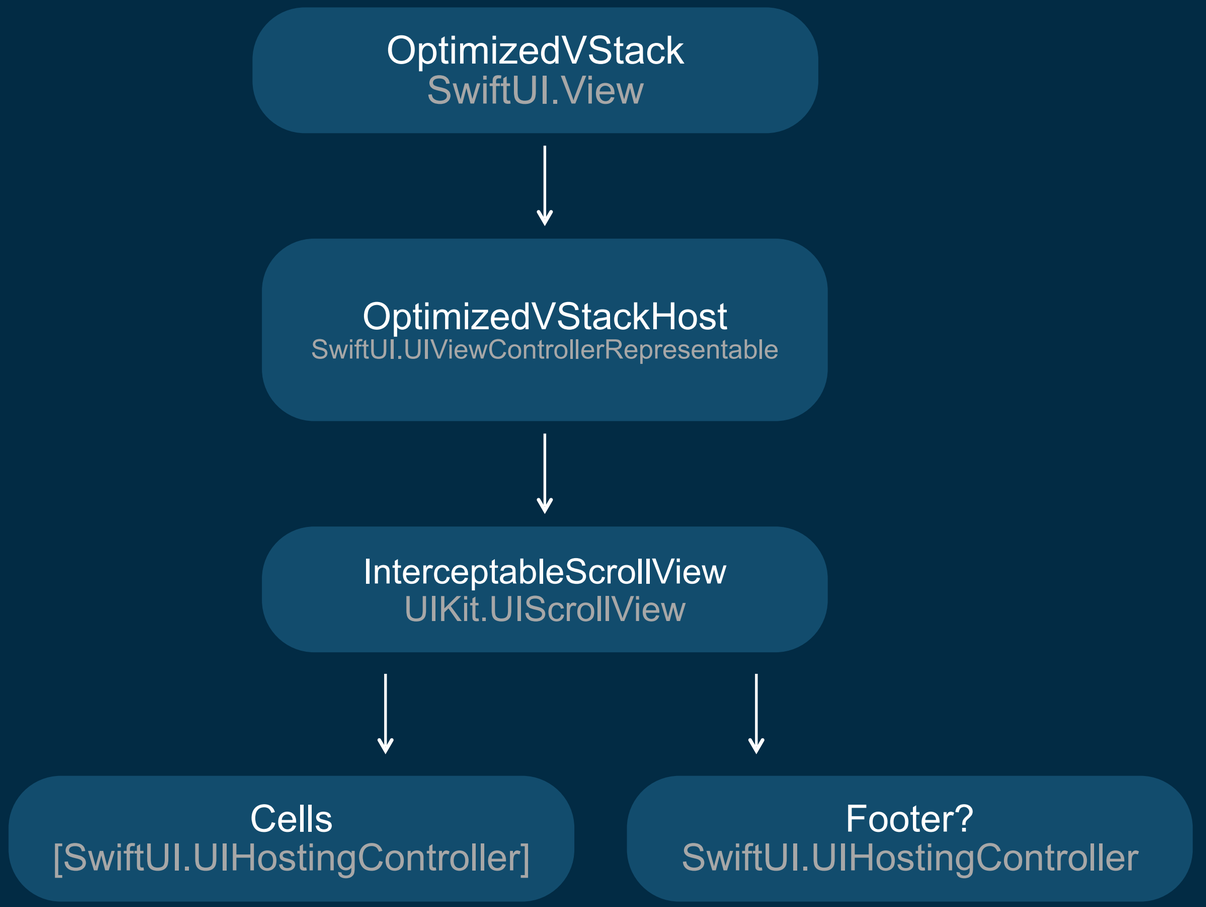
2) Лишнее обновление @State за frame
Следствие: undefined behaviour



Реализация №3

# Собираем рабочую схему для OptimizedVStack

# OptimizedVStack



# Алгоритм

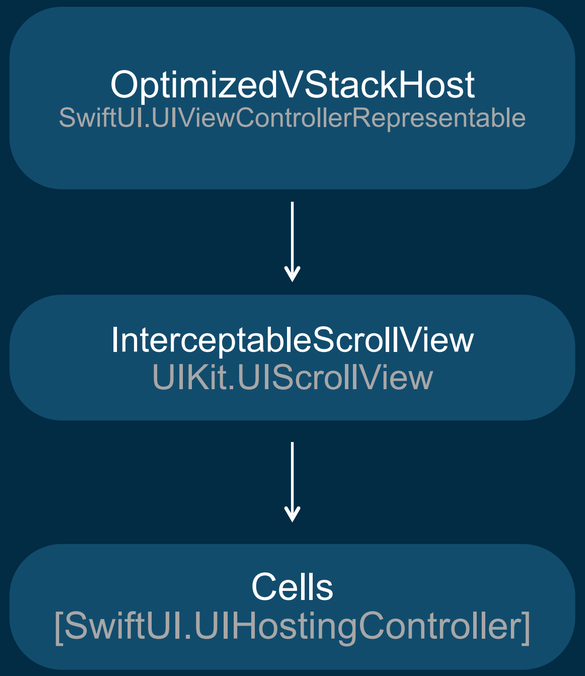
# Расчёт 1-го frame

# OptimizedVStack

Расчёт 1-го frame

Для OptimizedVStackHost: UIViewControllerRepresentable

В момент 1-го sizeThatFits(_ proposal: ProposedViewSize):



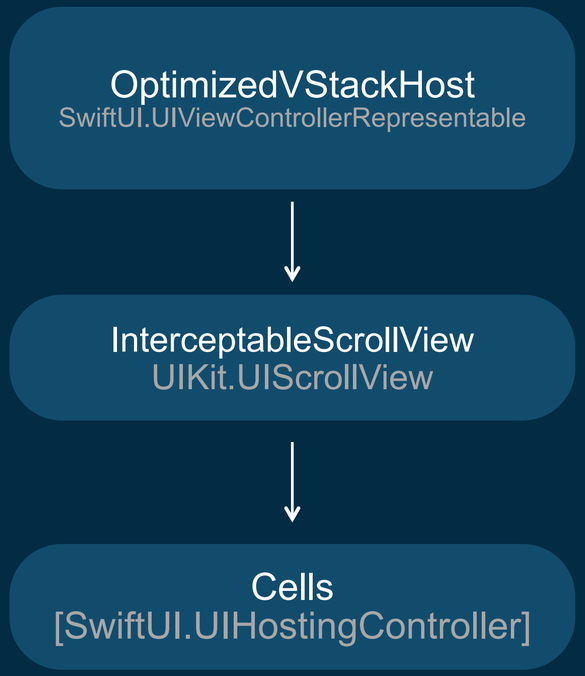
# OptimizedVStack

Расчёт 1-го frame

Для OptimizedVStackHost: UIViewControllerRepresentable

В момент 1-го sizeThatFits(_ proposal: ProposedViewSize):

1) **Считаем VisibleArea внутри ScrollView**
   Добавляем +/-15% от границ реального VisibleArea



# OptimizedVStack

Расчёт 1-го frame

Для OptimizedVStackHost: UIViewControllerRepresentable

В момент 1-го sizeThatFits(_ proposal: ProposedViewSize):

1) **Считаем VisibleArea внутри ScrollView**
Добавляем +/-15% от границ реального VisibleArea

**2) Пока ячейки помещаются в VisibleArea:**

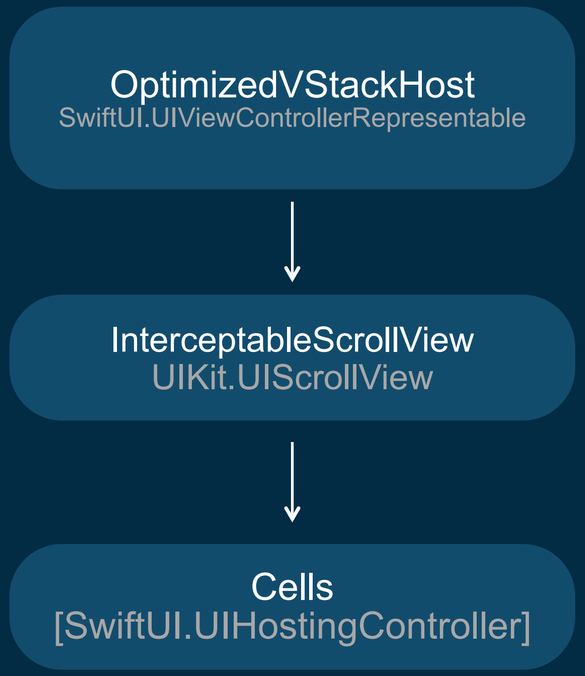
2.1. cell = UIHostingController(SwiftUI.View)

2.2. size = cell.sizeThatFits(in: proposal)

2.3. Кэшируем size в UIViewControllerRepresentable.Coordinator

2.4. Устанавливаем ячейки в InterceptableScrollView

2.5. Кэшируем позиции ячеек в UIViewControllerRepresentable.Coordinator



# OptimizedVStack

Расчёт 1-го frame

Для OptimizedVStackHost: UIViewControllerRepresentable

В момент 1-го sizeThatFits(_ proposal: ProposedViewSize):

1) **Считаем VisibleArea внутри ScrollView**
Добавляем +/-15% от границ реального VisibleArea

**2) Пока ячейки помещаются в VisibleArea:**

2.1. cell = UIHostingController(SwiftUI.View)

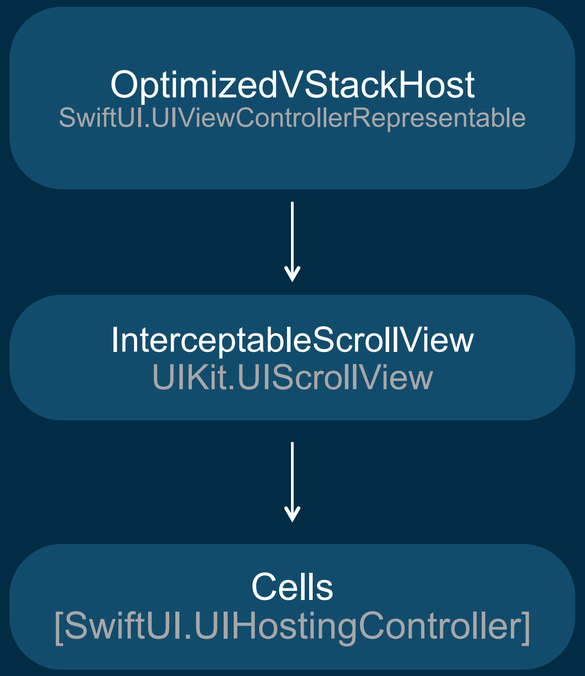
2.2. size = cell.sizeThatFits(in: proposal)

2.3. Кэшируем size в UIViewControllerRepresentable.Coordinator

2.4. Устанавливаем ячейки в InterceptableScrollView

2.5. Кэшируем позиции ячеек в UIViewControllerRepresentable.Coordinator

3) **Обновить** InterceptableScrollView.contentSize



# OptimizedVStack

Расчёт 1-го frame

Для OptimizedVStackHost: UIViewControllerRepresentable

В момент 1-го sizeThatFits(_ proposal: ProposedViewSize):

1) **Считаем VisibleArea внутри ScrollView**
   Добавляем +/-15% от границ реального VisibleArea

**2) Пока ячейки помещаются в VisibleArea:**

2.1. cell = UIHostingController(SwiftUI.View)

2.2. size = cell.sizeThatFits(in: proposal)

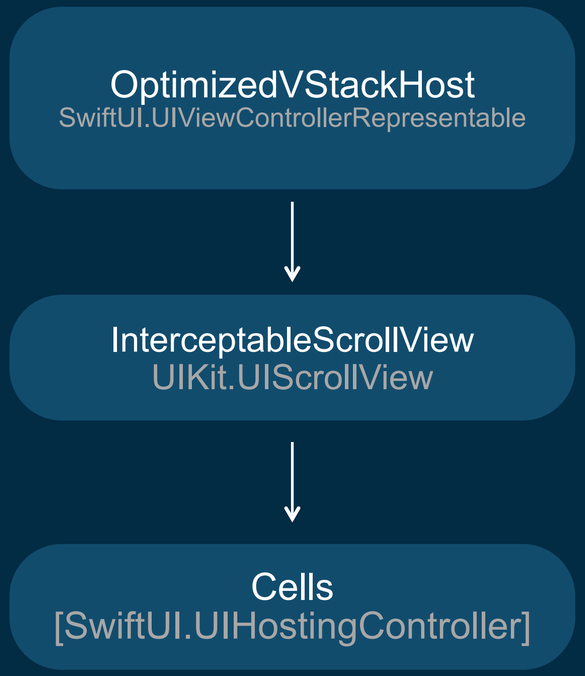
2.3. Кэшируем size в UIViewControllerRepresentable.Coordinator

2.4. Устанавливаем ячейки в InterceptableScrollView

2.5. Кэшируем позиции ячеек в UIViewControllerRepresentable.Coordinator

3) **Обновить** InterceptableScrollView.contentSize

4) **(Optional) Сообщить «размер контента» клиенту**

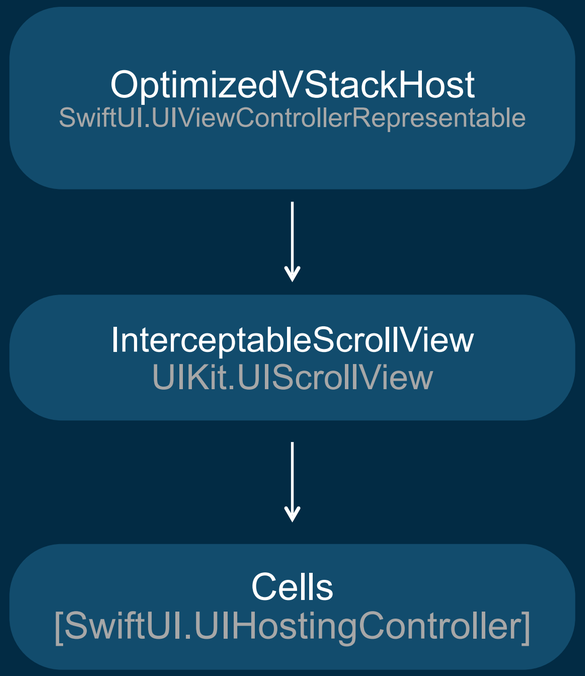


# OptimizedVStack

Расчёт 1-го frame

Что за «размер контента» получает клиент

```
let newValue: MeasuredHeight = if coordinator.isAllCellsMeasured {
    .exact(value: coordinator.measuredHeight)
} else {
    .estimated(
        total: coordinator.estimatedHeight,
        calculated: coordinator.measuredHeight,
        approximated: coordinator.estimatedHeight - coordinator.measuredHeight
    )
}
```

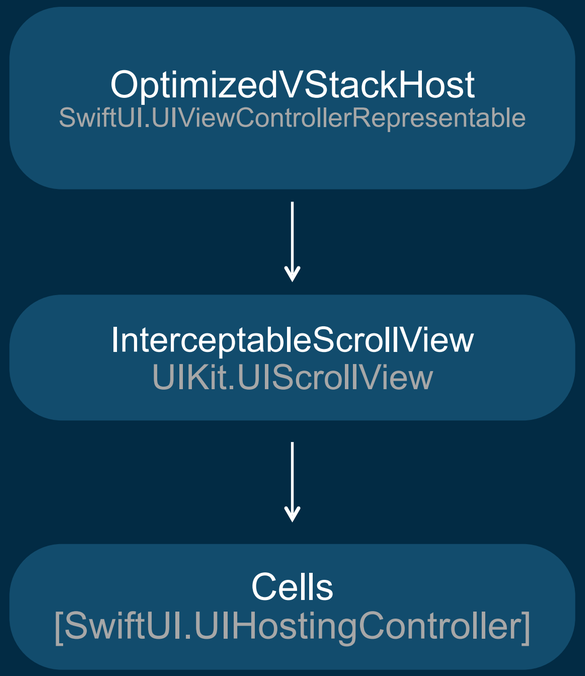


# OptimizedVStack

Расчёт 1-го frame

Что за «размер контента» получает клиент

```
let newValue: MeasuredHeight = if coordinator.isAllCellsMeasured {
    .exact(value: coordinator.measuredHeight)
} else {
    .estimated(
        total: coordinator.estimatedHeight,
        calculated: coordinator.measuredHeight,
        approximated: coordinator.estimatedHeight - coordinator.measuredHeight
    )
}
```




Конкретный источник происхождения размера

Размеры без GeometryReader

# Расчёт frame при scroll

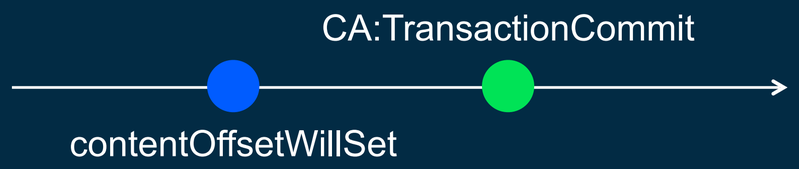
OZON tech

# Этап 1.
# Решение корневой проблемы с «белым» экраном

# OptimizedVStack

Решение проблемы «белого» экрана

Нужно «перехватить» изменение contentOffset и подготовить новый кадр



# OptimizedVStack

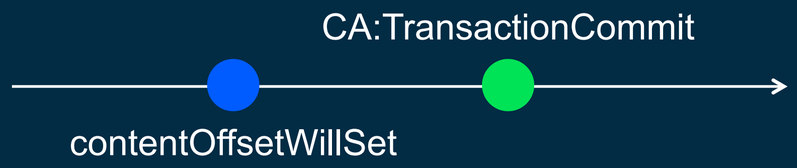
Решение проблемы «белого» экрана

Нужно «перехватить» изменение contentOffset и подготовить новый кадр

Swizzling

1. Хукаем #selector сеттера contentOffset

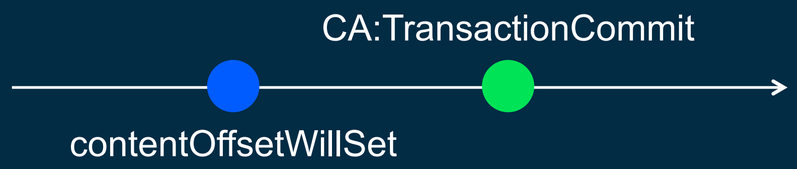
```
func overrideContentOffsetSetter( in subclass: AnyClass, originalClass: AnyClass ) {
    let selector = #selector(setter: InterceptableScrollView.contentOffset)
    guard let method = class_getInstanceMethod(originalClass, selector) else {
        return
    }
```



# OptimizedVStack

Решение проблемы «белого» экрана

Нужно «перехватить» изменение contentOffset и подготовить новый кадр




Swizzling

1. Хукаем #selector сеттера contentOffset

```
func overrideContentOffsetSetter( in subclass: AnyClass, originalClass: AnyClass ) {
    let selector = #selector(setter: InterceptableScrollView.contentOffset)
    guard let method = class_getInstanceMethod(originalClass, selector) else {
        return
    }
```

2. Замещенный сеттер contentOffset аггрегирует: клиентский метод + оригинальный #selector

```
func overrideContentOffsetSetter( in subclass: AnyClass, originalClass: AnyClass ) {
    let newSetterBlock: NewSetContentOffsetSetterType = { scrollView, offset in
        scrollView.contentOffsetWillSet?(offset.y)

        let originalSetter = unsafeBitCast(
            originalIMP, to: OriginalSetContentOffsetSetterType.self
        )
        originalSetter(scrollView, selector, offset)
    }
```

# OptimizedVStack

Решение проблемы «белого» экрана

Нужно «перехватить» изменение contentOffset и подготовить новый кадр


Swizzling

1. Хукаем #selector сеттера contentOffset

```
func overrideContentOffsetSetter( in subclass: AnyClass, originalClass: AnyClass ) {
    let selector = #selector(setter: InterceptableScrollView.contentOffset)
    guard let method = class_getInstanceMethod(originalClass, selector) else {
        return
    }
```

2. Замещенный сеттер contentOffset аггрегирует: клиентский метод + оригинальный #selector

```
func overrideContentOffsetSetter( in subclass: AnyClass, originalClass: AnyClass ) {
    let newSetterBlock: NewSetContentOffsetSetterType = { scrollView, offset in
        scrollView.contentOffsetWillSet?(offset.y)

        let originalSetter = unsafeBitCast(
            originalIMP, to: OriginalSetContentOffsetSetterType.self
        )
        originalSetter(scrollView, selector, offset)
    }
```

ozontech

OZON tech

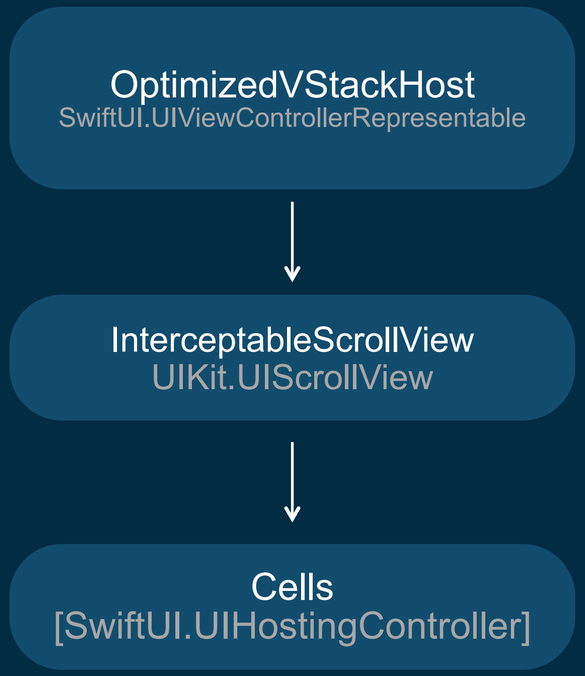
# Этап 2.
# Расчёт frame

# OptimizedVStack

Расчёт frame при скролле

Для OptimizedVStackHost: UIViewControllerRepresentable

В момент перехвата "set'a" на InterceptableScrollView.contentOffset:



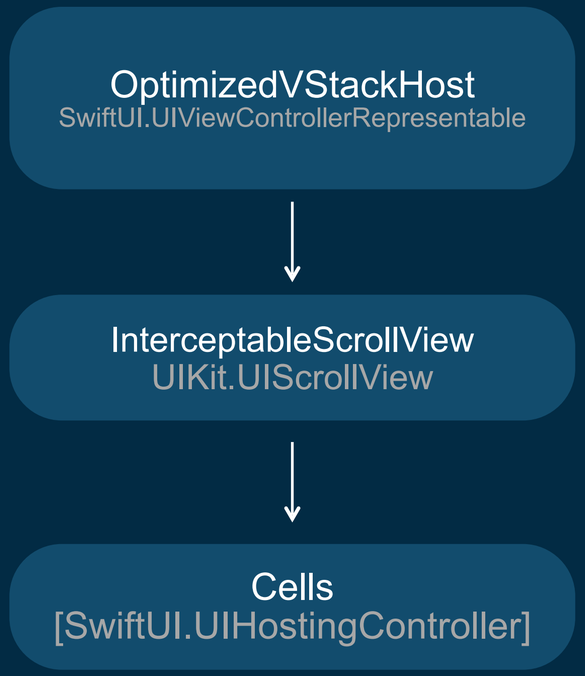
# OptimizedVStack

Расчёт frame при скролле

Для OptimizedVStackHost: UIViewControllerRepresentable

В момент перехвата "set'a" на InterceptableScrollView.contentOffset:

**1) Определяем ScrollDirection**



# OptimizedVStack

Расчёт frame при скролле

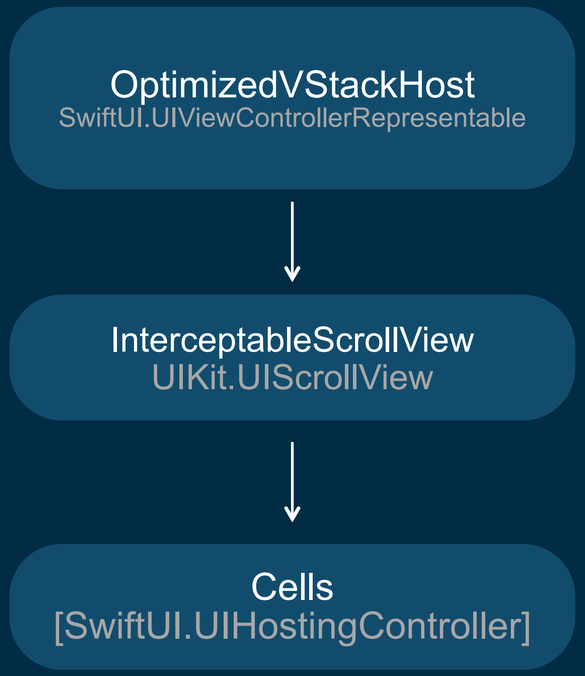
Для OptimizedVStackHost: UIViewControllerRepresentable

В момент перехвата "set'a" на InterceptableScrollView.contentOffset:

**1) Определяем ScrollDirection**

**2) Предсказываем видимый Range ячеек для будущего contentOffset**

- Для "незакэшированных" ячеек считаем высоту как estimate



# OptimizedVStack

Расчёт frame при скролле

Для OptimizedVStackHost: UIViewControllerRepresentable

В момент перехвата "set'a" на InterceptableScrollView.contentOffset:

**1) Определяем ScrollDirection**

**2) Предсказываем видимый Range ячеек для будущего contentOffset**

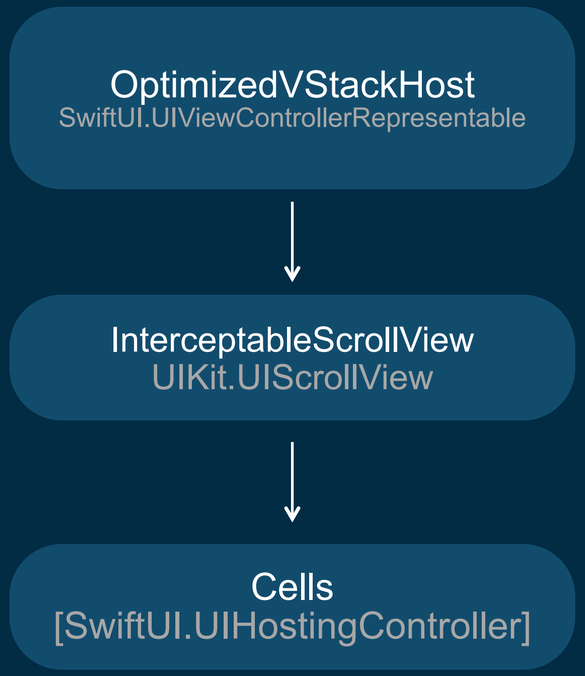
- Для "незакэшированных" ячеек считаем высоту как estimate

**3) Установка ячеек**

3.1. Считаем Diff текущего и будущего состояний

3.2. Одновременно:

- Устанавливаем нужные ячейки/удаляем ненужные ячейки
- Уточняем Range ячеек



# OptimizedVStack

Расчёт frame при скролле

Для OptimizedVStackHost: UIViewControllerRepresentable

В момент перехвата "set'a" на InterceptableScrollView.contentOffset:

**1) Определяем ScrollDirection**

**2) Предсказываем видимый Range ячеек для будущего contentOffset**

- Для "незакэшированных" ячеек считаем высоту как estimate

**3) Установка ячеек**

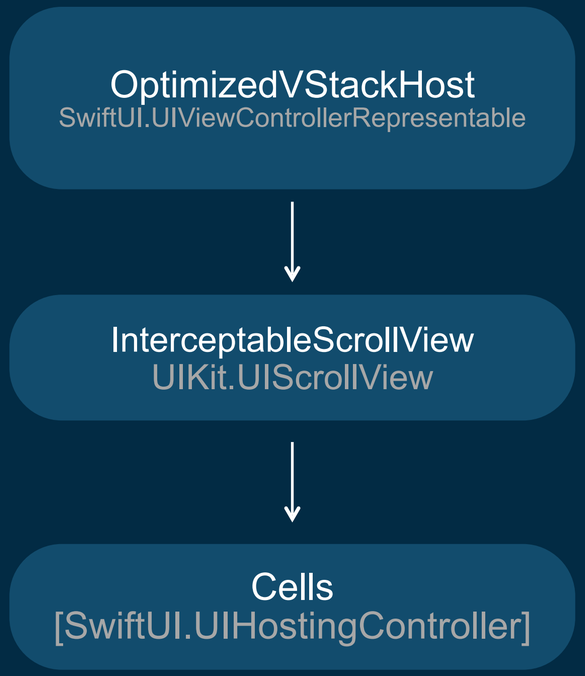
3.1. Считаем Diff текущего и будущего состояний

3.2. Одновременно:

- Устанавливаем нужные ячейки/удаляем ненужные ячейки
- Уточняем Range ячеек

**4) Обновляем InterceptableScrollView.contentSize**

**4.1. (Optional) Сообщаем клиентам об «изменившемся размере списка»**



# Вопросы к LazyVStack вне GeometryReader

Медленный расчёт кадров

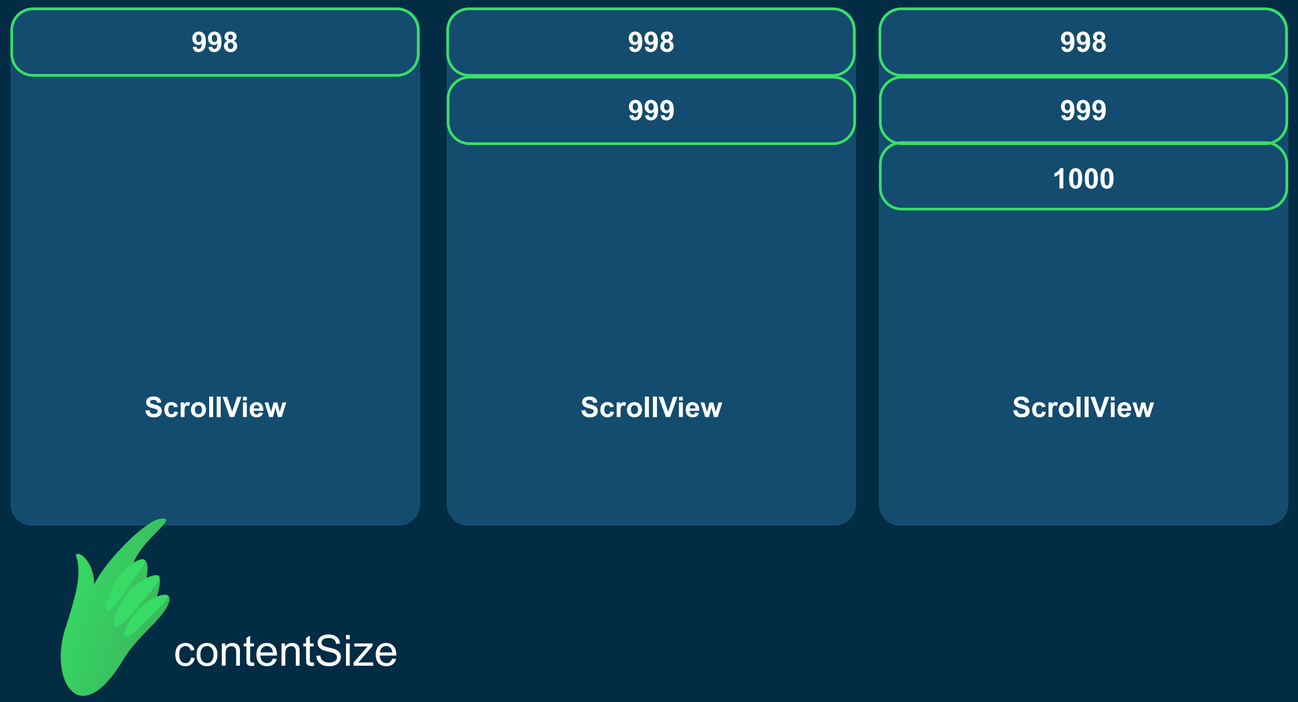
Преждевременное заполнение кэша размеров ячеек

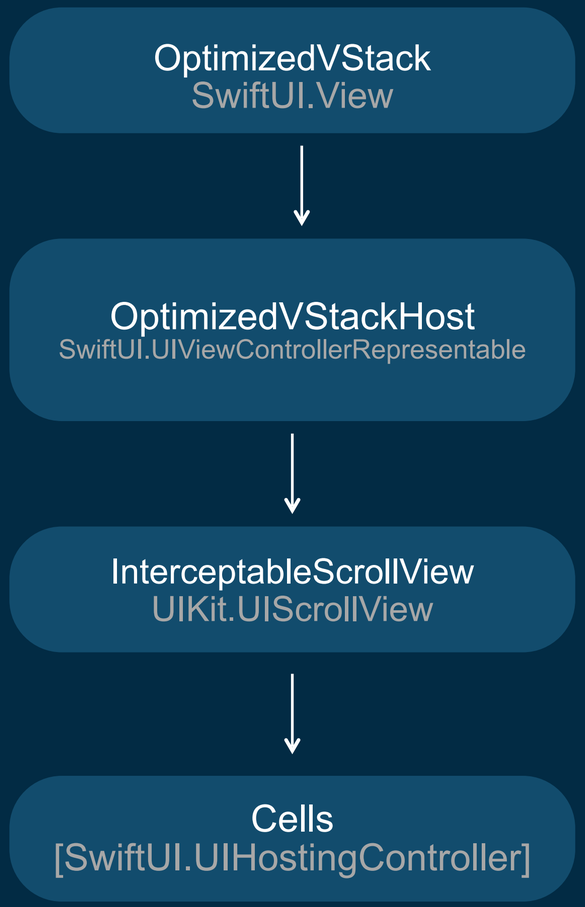
Неожиданная «обрезка» списка

«White screen» в больших гетерогенных списках

# OptimizedVStack

Решение проблемы «белого» экрана LazyVStack
Пример «быстрого скролла» вниз

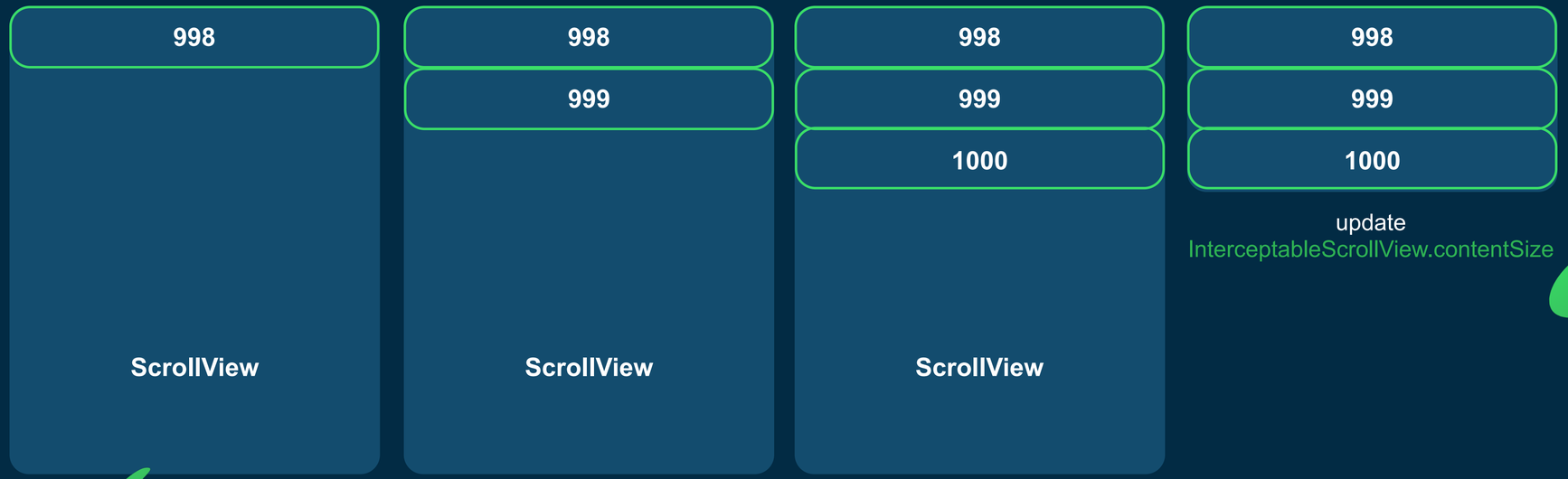


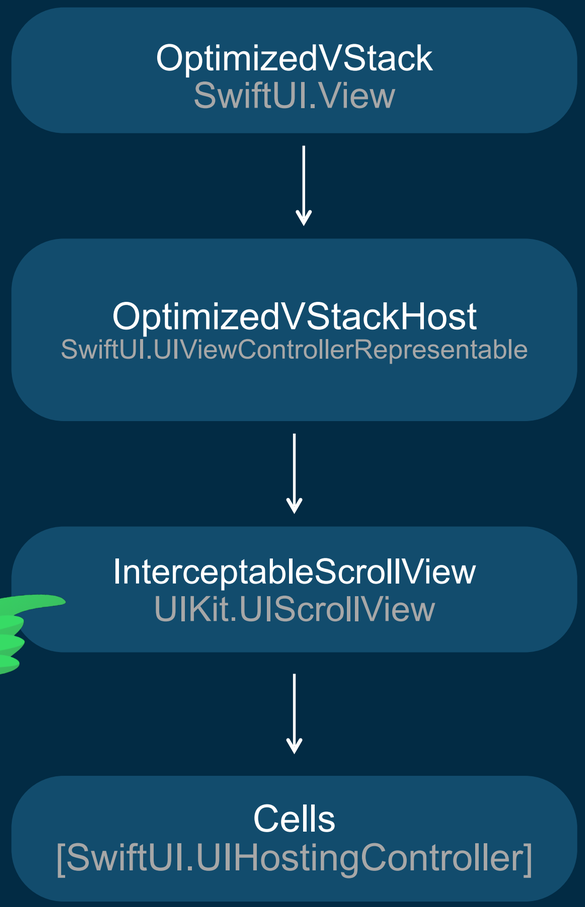

ozon tech

# OptimizedVStack

Решение проблемы «белого» экрана LazyVStack
Пример «быстрого скролла» вниз

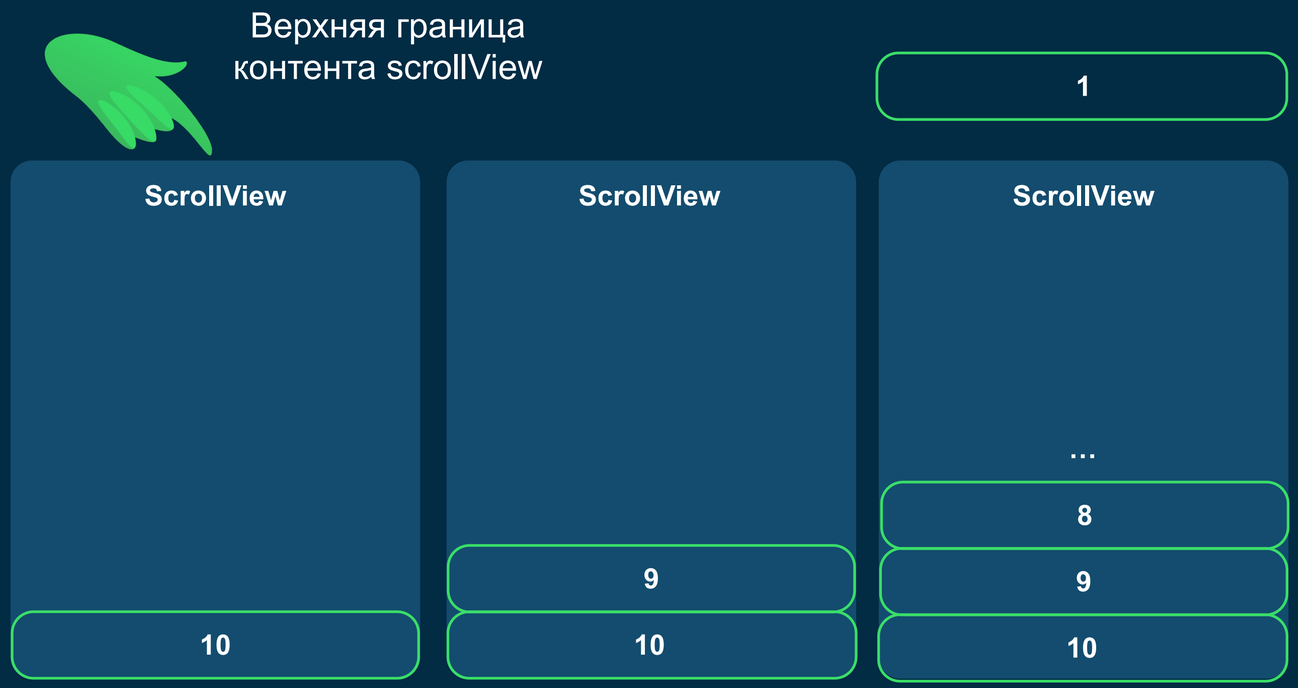


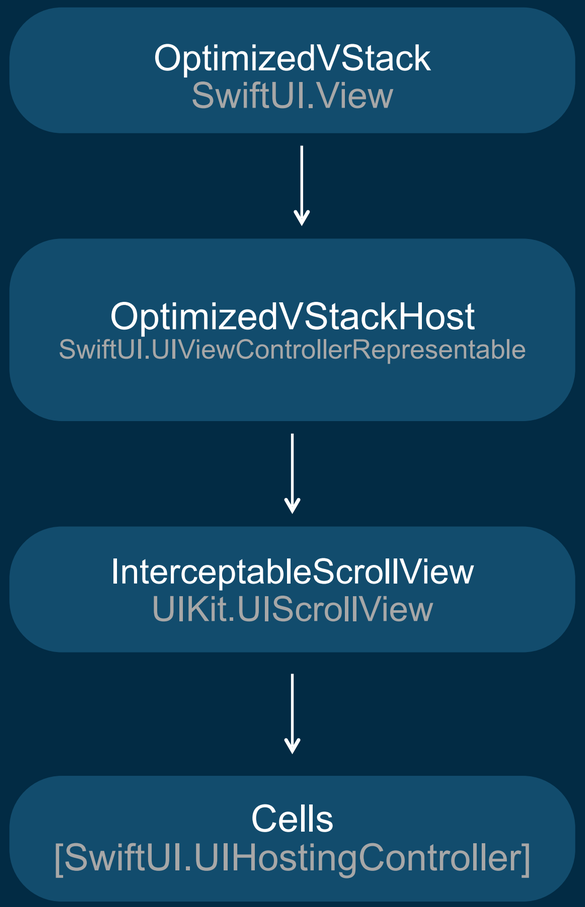

contentSize



# OptimizedVStack

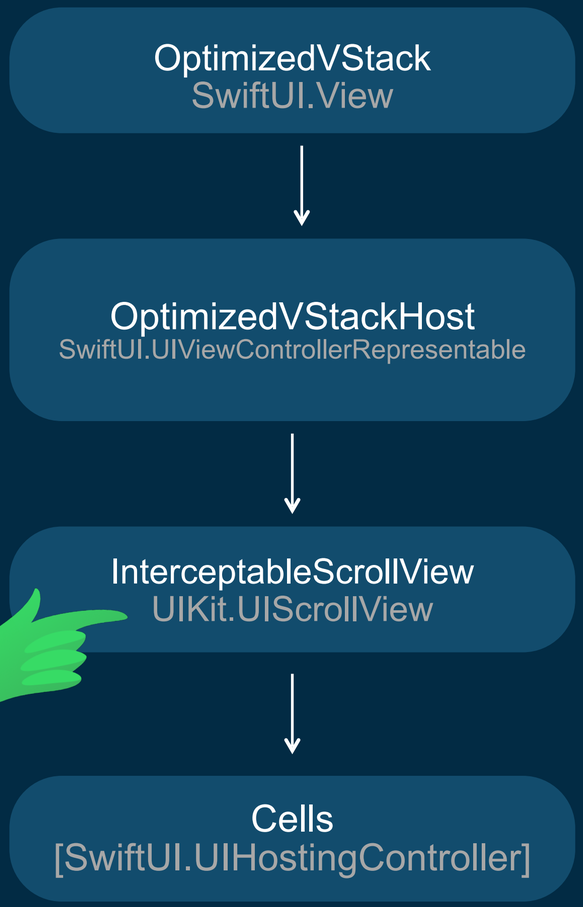
Решение проблемы «белого» экрана LazyVStack
Пример «быстрого скролла» вверх



# OptimizedVStack

Решение проблемы «белого» экрана LazyVStack
Пример «быстрого скролла» вверх

# Вопросы к LazyVStack вне GeometryReader

Медленный расчёт кадров

Преждевременное заполнение кэша размеров ячеек

Неожиданная «обрезка» списка

«White screen» в больших гетерогенных списках

# Проблемы при реализации

# OptimizedVStack

SwiftUI / UIViewControllerRepresentable / sizeThatFits(_:uiViewController:context:)

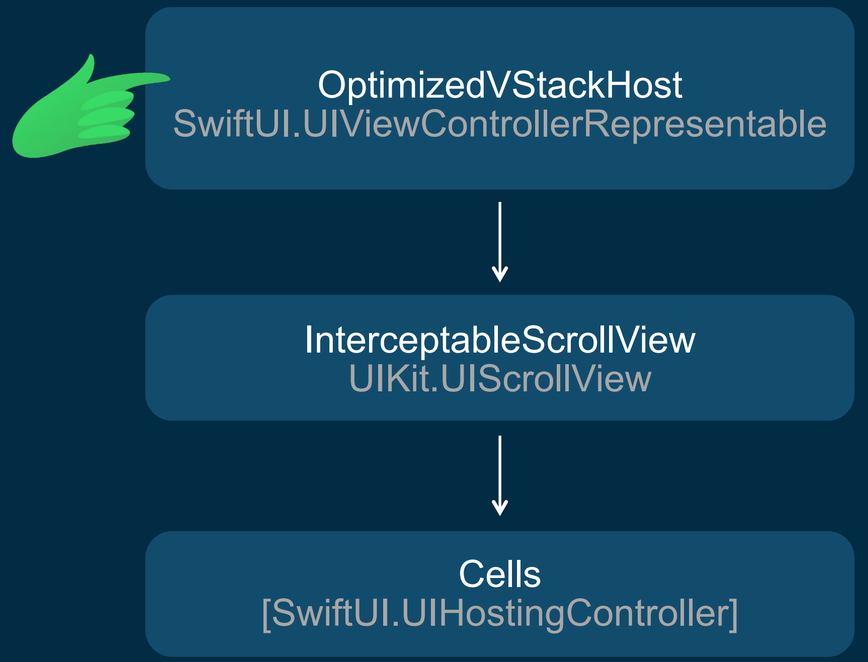
Instance Method

## sizeThatFits(_:uiViewController:context:)

Given a proposed size, returns the preferred size of the composite view.

iOS 16.0+ | iPadOS 16.0+ | Mac Catalyst 16.0+ | tvOS 16.0+ | visionOS 1.0+

```
@MainActor @preconcurrency
func sizeThatFits(
    _ proposal: ProposedViewSize,
    uiViewController: Self.UIViewControllerType,
    context: Self.Context
) -> CGSize?
```



# OptimizedVStack

SwiftUI / UIViewControllerRepresentable / sizeThatFits(_:uiViewController:context:)

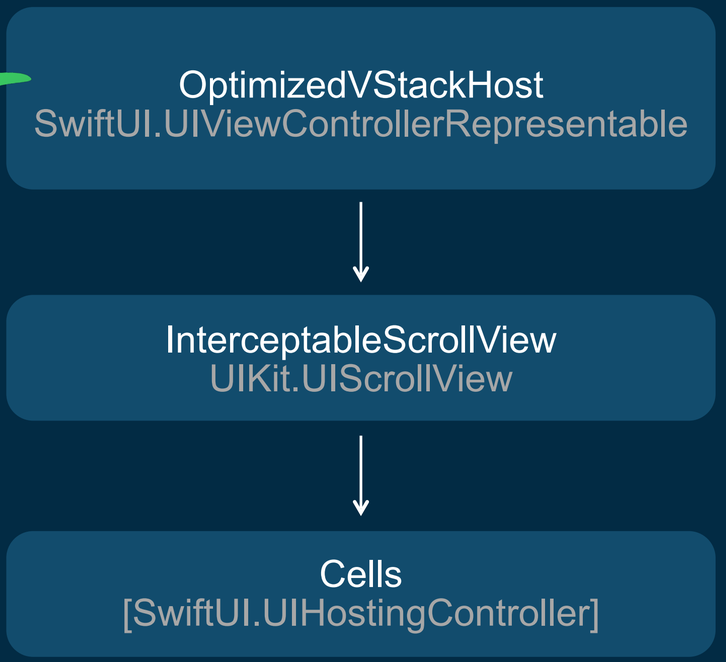
Instance Method

## sizeThatFits(_:uiViewController:context:)

Given a proposed size, returns the preferred size of the composite view.

iOS 16.0+ | iPadOS 16.0+ | Mac Catalyst 16.0+ | tvOS 16.0+ | visionOS 1.0+

```
@MainActor @preconcurrency
func sizeThatFits(
    _ proposal: ProposedViewSize,
    uiViewController: Self.UIViewControllerType,
    context: Self.Context
) -> CGSize?
```



# OptimizedVStack

Бэкпорт Layout-протокола
для 14/15 осей

# OptimizedVStack

```
67 ···      func _overrideSizeThatFits(
68              _ size: inout CoreGraphics.CGSize,
69              in proposedSize: SwiftUI._ProposedSize,
70              uiView: FrameChangePlacerView<L>
71          ) {
```

Бэкпорт Layout-протокола
для 14/15 осей

# OptimizedVStack

```
67 ···      func _overrideSizeThatFits(
68              _ size: inout CoreGraphics.CGSize,
69              in proposedSize: SwiftUI._ProposedSize,
70              uiView: FrameChangePlacerView<L>
71          ) {
```

Бэкпорт Layout-протокола
для 14/15 осей

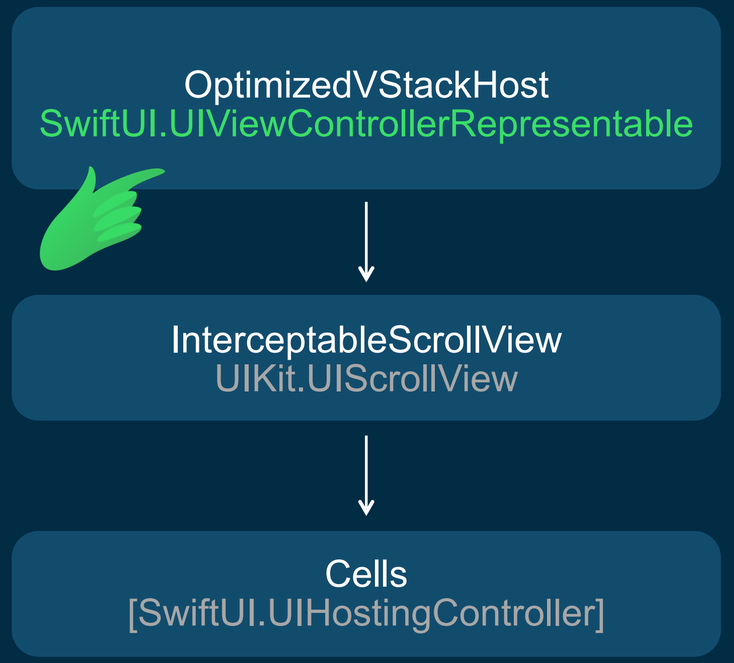
/Applications/Xcode.app/.../Library/Frameworks/SwiftUI.framework/Modules/SwiftUI.swiftmodule

```
public protocol UIViewRepresentable : SwiftUICore.View where Self.Body == Swift.Never {
    @available(iOS 16.0, tvOS 16.0, *)
    func sizeThatFits(_ proposal: SwiftUICore.ProposedViewSize, uiView: Self.UIViewType, context: Self.Context) -> CoreFoundation.CGSize?
    func _overrideSizeThatFits(_ size: inout CoreFoundation.CGSize, in proposedSize: SwiftUICore._ProposedSize, uiView: Self.UIViewType)
}
```

# OptimizedVStack

Бэкпорт Layout-протокола
для 14/15 осей

```
67 ···      func _overrideSizeThatFits(
68              _ size: inout CoreGraphics.CGSize,
69              in proposedSize: SwiftUI._ProposedSize,
70              uiView: FrameChangePlacerView<L>
71          ) {
```




/Applications/Xcode.app/.../Library/Frameworks/SwiftUI.framework/Modules/SwiftUI.swiftmodule

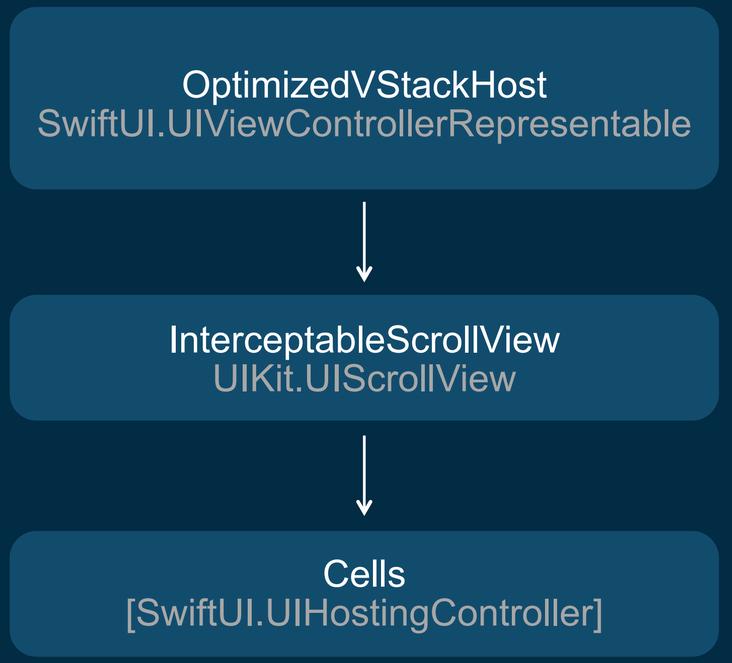
```
public protocol UIViewRepresentable : SwiftUICore.View where Self.Body == Swift.Never {
    @available(iOS 16.0, tvOS 16.0, *)
    func sizeThatFits(_ proposal: SwiftUICore.ProposedViewSize, uiView: Self.UIViewType, context: Self.Context) -> CoreFoundation.CGSize?
    func _overrideSizeThatFits(_ size: inout CoreFoundation.CGSize, in proposedSize: SwiftUICore._ProposedSize, uiView: Self.UIViewType)
}
```

# OptimizedVStack

## Обходной путь для iOS 14+

Реализовать самостоятельный UIViewRepresentable, умеющий получать proposedSize

```
struct ProposedSizeCalculator: UIViewRepresentable {
    @Binding var proposedSize: ProposedSize

    func makeUIView(context: Context) -> UIView {
        UIView()
    }

    func updateUIView(_ uiView: UIView, context: Context) {}

    func _overrideSizeThatFits(
        _ size: inout CGSize,
        in proposedSize: _ProposedSize,
        uiView: UIView
    ) {
        Task { @MainActor in
            self.proposedSize = .init(
                width: proposedSize.width,
                height: proposedSize.height
            )
        }
    }
}
```

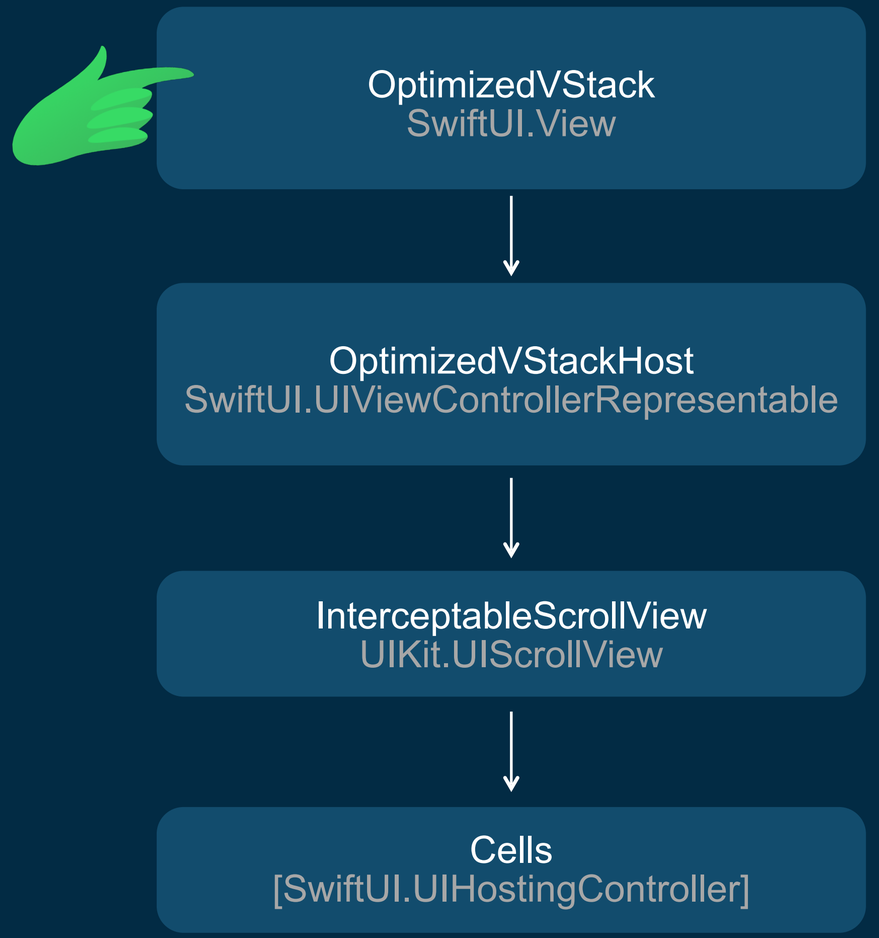


# OptimizedVStack

## Обходной путь для iOS 14+

Реализовать самостоятельный UIViewRepresentable,
умеющий получать proposedSize

```
extension OptimizedVStack: View {
    public var body: some View {
        if #unavailable(iOS 16) {
            optimizedVStackHost
                .environment(\.proposedSize, $proposedSize)
                .proposedSize($proposedSize)
        } else {
            optimizedVStackHost
        }
    }
```



# Использование OptimizedVStack

# OptimizedVStack

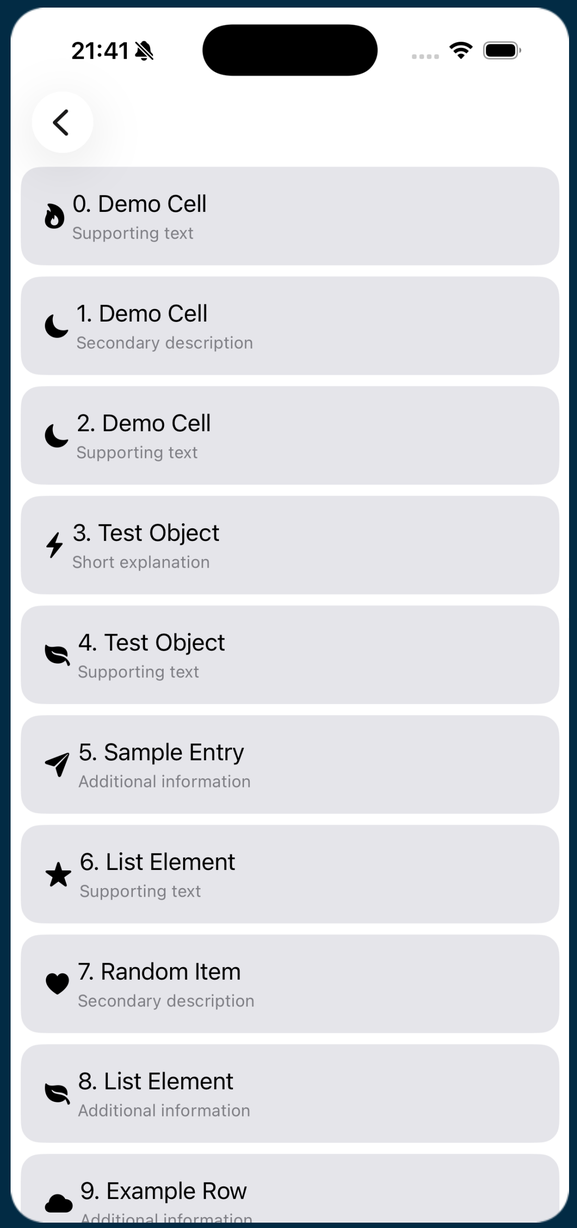
Подходит для всех типов «списков»

- Ячейки одинаковой высоты
- Ячейки различной высоты
- Ячейки с изменением размеров в Runtime
- Список с пагинацией

# OptimizedVStack

Самый простой кейс использования компонента

```
OptimizedVStack(
    data: items,
    spacing: 8
) { index in
    CellView(by: items[index])
}
.padding(.horizontal, 8)
```

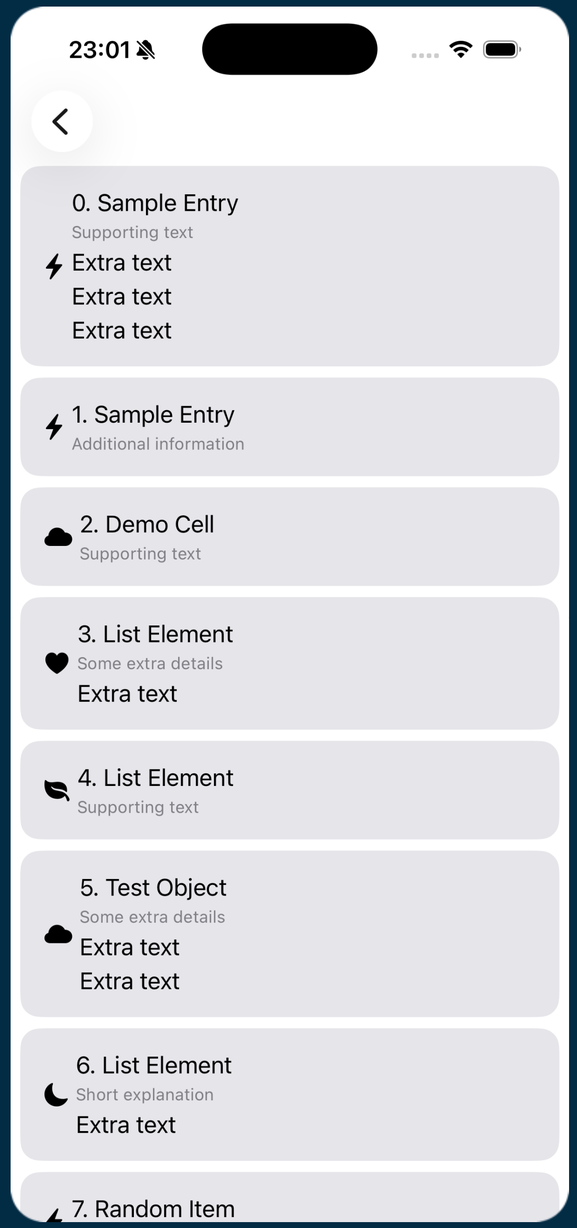


# OptimizedVStack

Возврат информации о размере контента

```
OptimizedVStack(
    data: items,
    spacing: 8,
    onChangeOfHeight: {
        print($0.formatted)
    }
) { index in
    CellView(by: items[index])
}
.padding(.horizontal, 8)
```


```
Line: 25  Col: 33
MeasuredHeight: estimated(total: 10070.79 = calculated: 1100.67 + approximated: 8970.12)
```

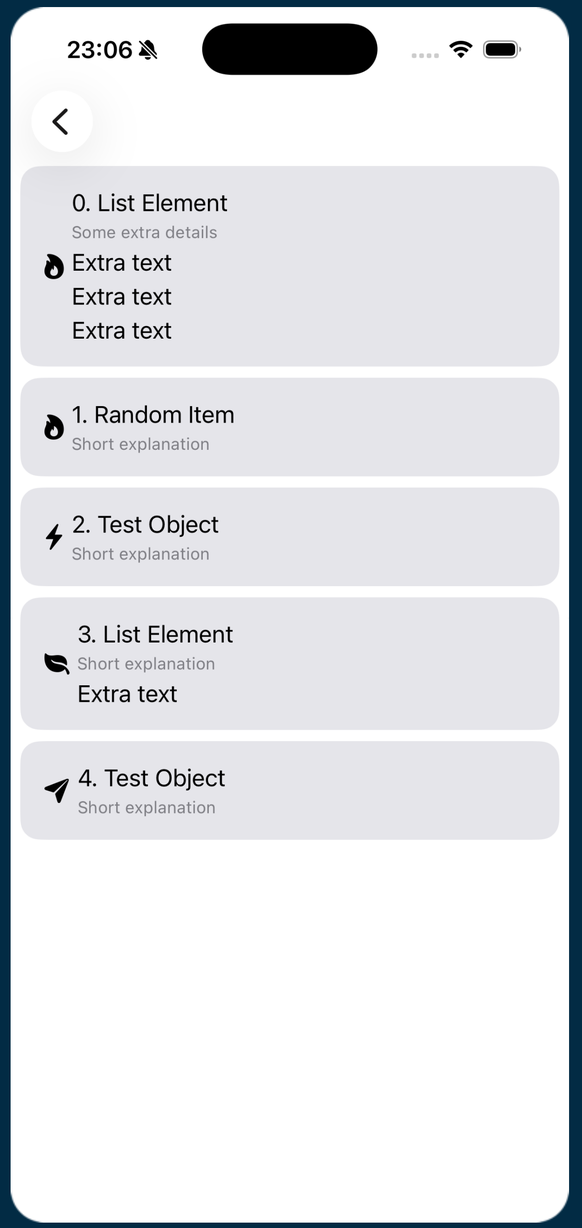


# OptimizedVStack

Возврат информации о размере контента

```
OptimizedVStack(
    data: items,
    spacing: 8,
    onChangeOfHeight: {
        print($0.formatted)
    }
) { index in
    CellView(by: items[index])
}
.padding(.horizontal, 8)
```


```
MeasuredHeight: full(total: 482.67)
```

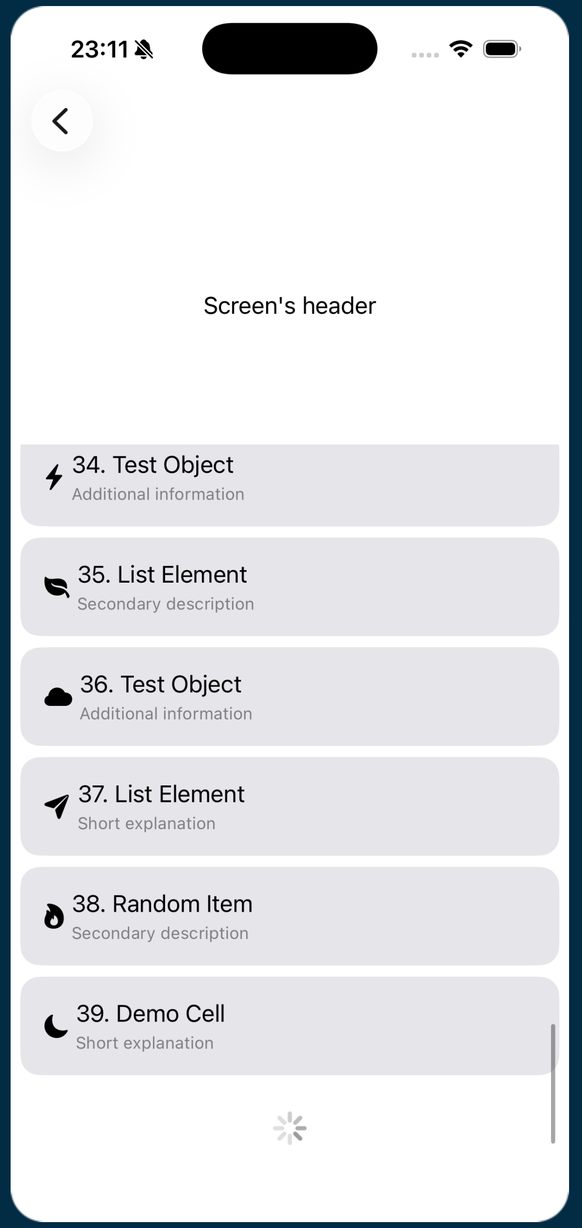


# OptimizedVStack

## Пагинация

```
VStack(spacing: .zero) {
    Text("Screen's header")
        .frame(height: 200)

    OptimizedVStack(
        data: $viewModel.items,
        spacing: 8,
        onChangeOfHeight: {
            print($0.formatted)
        },
        rowContent: { index in
            CellView(by: viewModel.items[index])
        },
        footer: {
            loader
        }
    )
    .padding(.horizontal, 8)
}
```

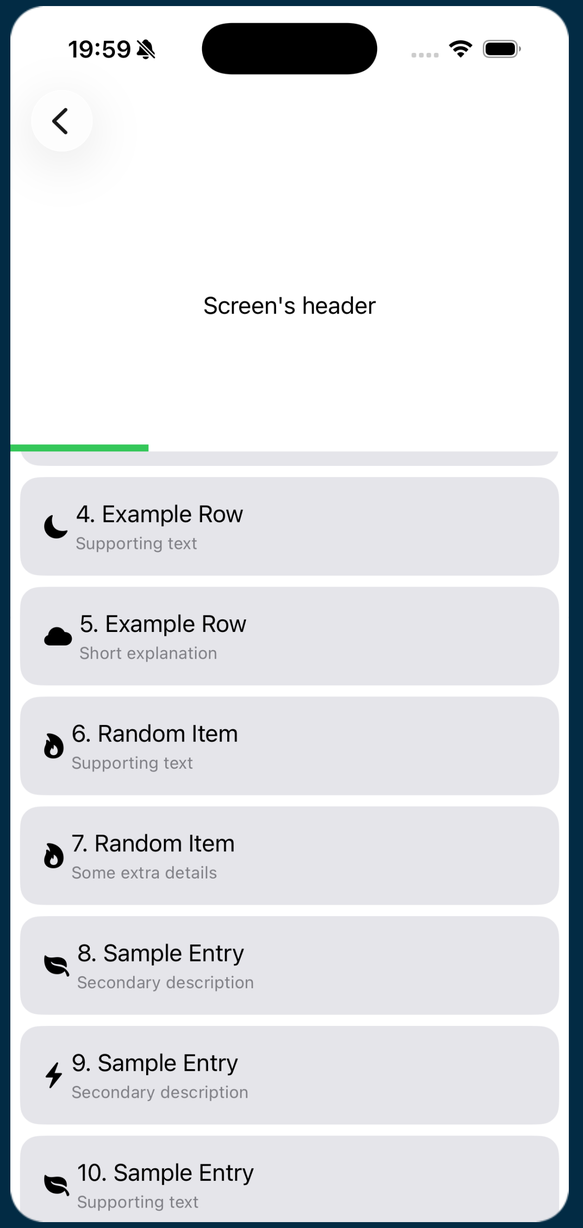

```
Line: 25  Col: 33
MeasuredHeight: estimated(total: 1565.33 = calculated: 778.67 + approximated: 786.67)
MeasuredHeight: full(total: 1565.33)
MeasuredHeight: estimated(total: 3138.67 = calculated: 1801.33 + approximated: 1337.33)
MeasuredHeight: full(total: 3138.67)
```

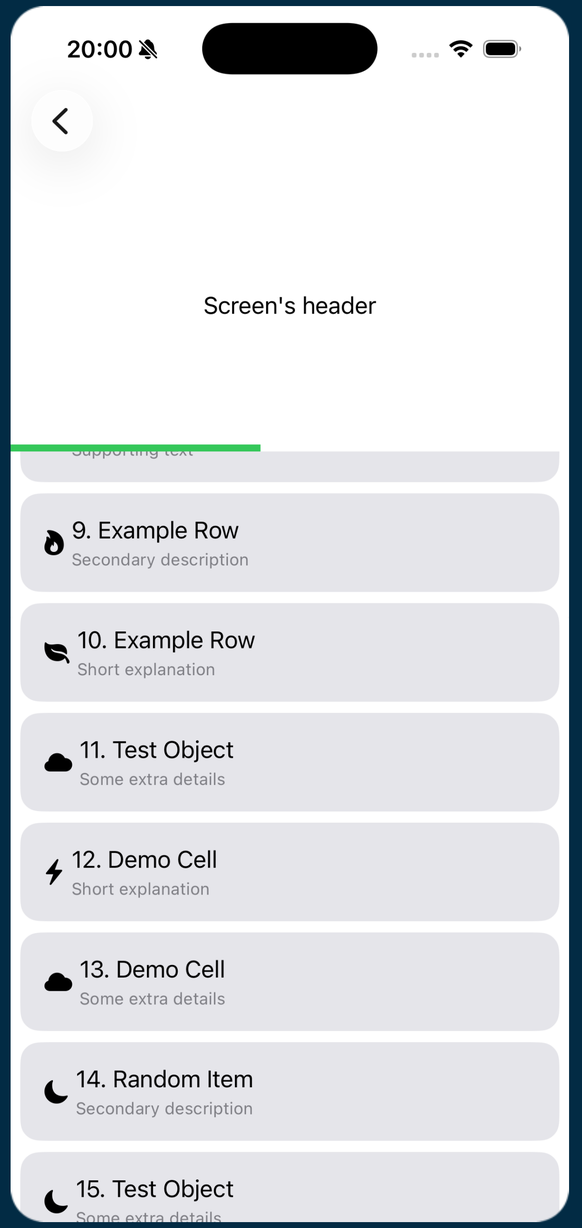


# OptimizedVStack

В сравнении с подходом через «GeometryReader» — размер зависимого контента пересчитывается всего 1 раз на кадр вместо 1+

```
VStack(spacing: .zero) {
    header

    progressView(by: viewModel.scrollProgress)

    OptimizedVStack(
        data: $viewModel.items,
        spacing: 8,
        onChangeOfScrollProgress: { currentYMax, measuredheight in
            viewModel.updateScrollProgress(
                currentYMax: currentYMax, totalHeight: measuredheight
            )
        },
        rowContent: { index in CellView(by: viewModel.items[index]) },
        footer: { loader }
    )
    .padding(.horizontal, 8)
}
```



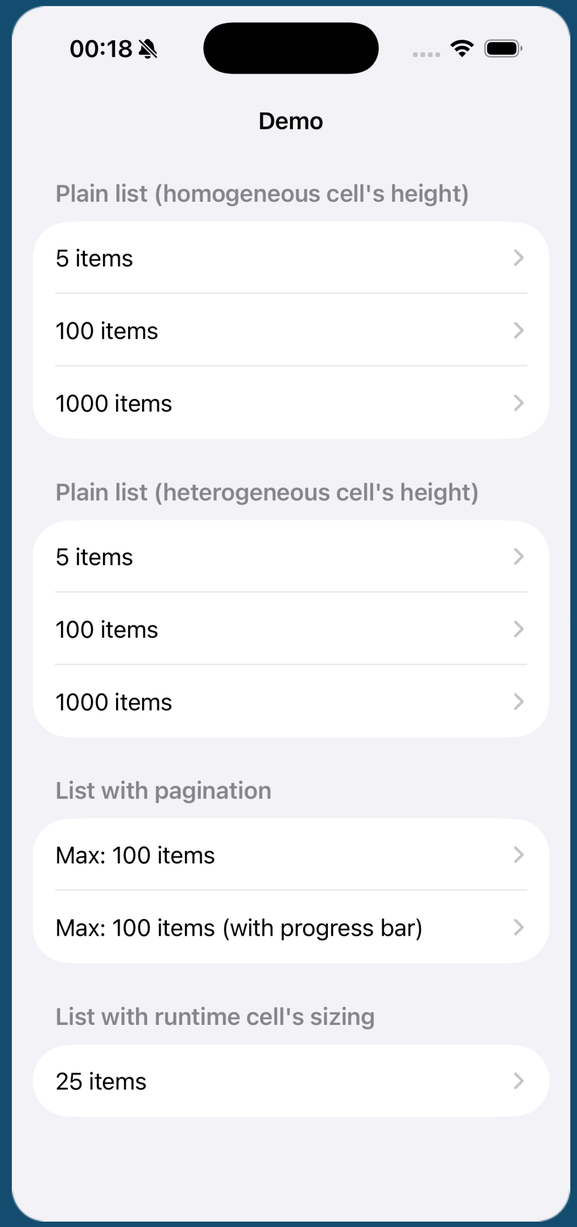



# OptimizedVStack

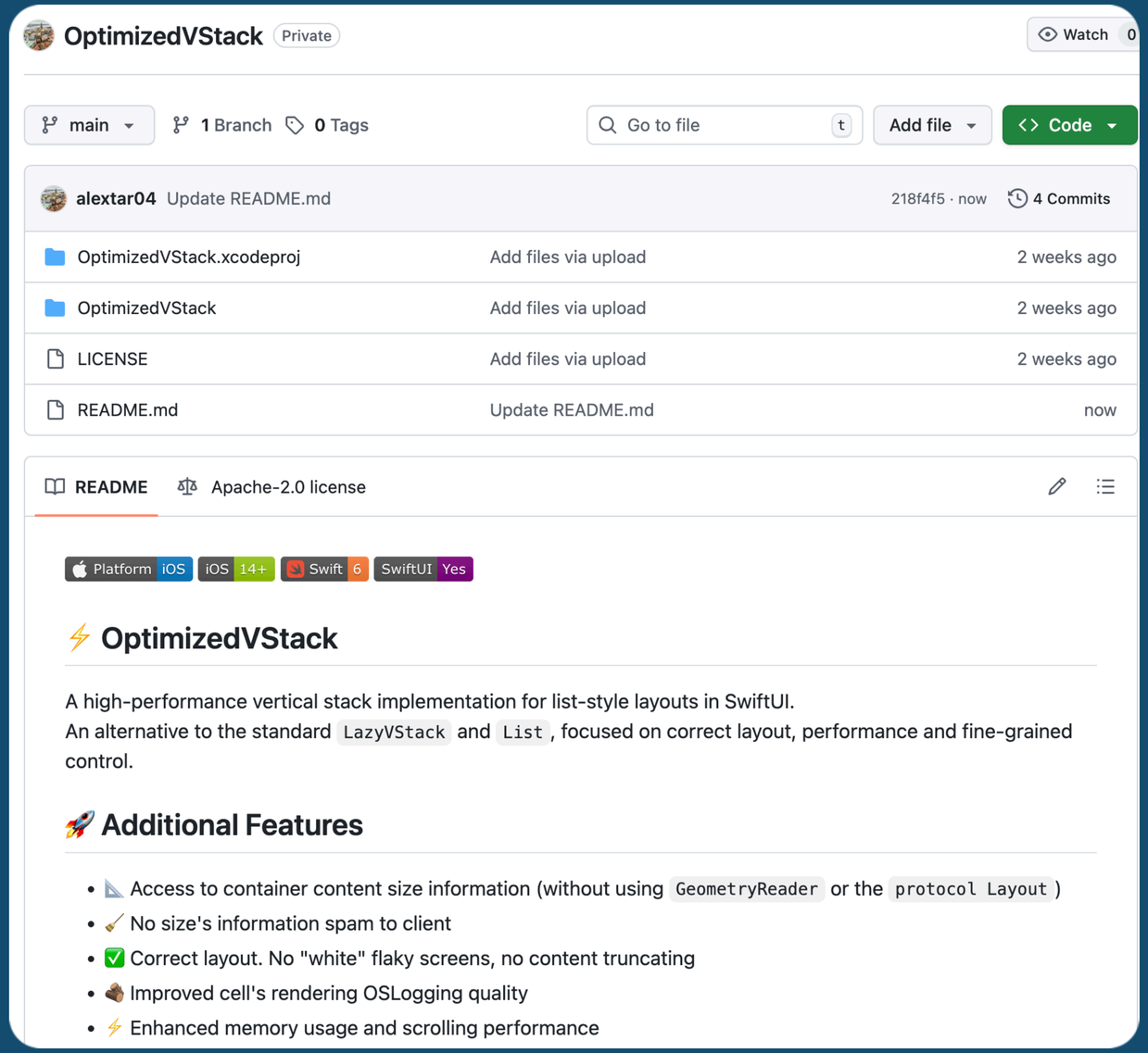
Как и LazyVStack, пишем логи в OSLog

| Timestamp | Type | Process | Subsyst... | Category | Message |
|---|---|---|---|---|---|
| 00:20.072.272 | Def... | OptimizedVSta... | AT.Optimiz... | OptimizedVStackLayout | Current cell's range: 43...57 |
| 00:20.072.282 | Def... | OptimizedVSta... | AT.Optimiz... | OptimizedVStackLayout | Target cell's range: 43...58 |
| 00:20.072.309 | Def... | OptimizedVSta... | AT.Optimiz... | OptimizedVStackLayout | Placing from #43 downward. Start position: 3382.67 in 3454.90...4591.10 |
| 00:20.073.750 | Def... | OptimizedVSta... | AT.Optimiz... | OptimizedVStackLayout | OptimizedVStackLayout: placed(visibleArea: 3454.90...4591.10) -> range: 43...58 |
| 00:20.073.768 | Def... | OptimizedVSta... | AT.Optimiz... | OptimizedVStackLayout | – |
| 00:20.088.953 | Def... | OptimizedVSta... | AT.Optimiz... | OptimizedVStackLayout | Current cell's range: 43...58 |
| 00:20.088.965 | Def... | OptimizedVSta... | AT.Optimiz... | OptimizedVStackLayout | Target cell's range: 44...58 |
| 00:20.088.992 | Def... | OptimizedVSta... | AT.Optimiz... | OptimizedVStackLayout | Placing from #44 downward. Start position: 3461.33 in 3490.90...4627.10 |
| 00:20.089.354 | Def... | OptimizedVSta... | AT.Optimiz... | OptimizedVStackLayout | OptimizedVStackLayout: placed(visibleArea: 3490.90...4627.10) -> range: 44...58 |

https://github.com/alextar04/OptimizedVStack



https://github.com/alextar04/OptimizedVStack



# Цифры

# Получает ли клиент корректные данные о высоте контента?

iPhone 12, iOS 18.6.2

| | Результат | Конечный результат высоты | Кол-во итераций расчета | Примечание к API |
|---|---|---|---|---|
| List + GeometryReader | ✗ | 778 | - | iOS 13+ |
| List + Introspect + uiCollectionView.contentSize | ✗ | 803 | - | iOS 13+ |
| LazyVStack + GeometryReader | ✓ | 1086 | 5 | iOS 14+ |
| LazyVStack + ViewThatFits | ✓ | 1086 | 5 | iOS 16+ |
| LazyVStack + View.scrollBounceBehavior | ✓ | 1086 | 2 | iOS 16.4+ |
| OptimizedVStack + onChangeOfHeight | ✓ | 1086 | 1 | iOS 14+ |

```
Container(spacing: 16) {
    Text("Текст 1")
        .frame(height: 20)

    Text("Текст 2")
        .frame(height: 30)

    Text("Текст 3")
        .frame(height: 40)

    Text("Текст 4")
        .frame(height: 500)

    Text("Текст 5")
        .frame(height: 400)
}
.padding()
```

Высота контента: 1086

# Скорость Layout 1-го кадра

iPhone 12, iOS 18.6.2

| | Результат |
|---|---|
| LazyVStack | 25 мс. |
| OptimizedVStack | 10 мс. |

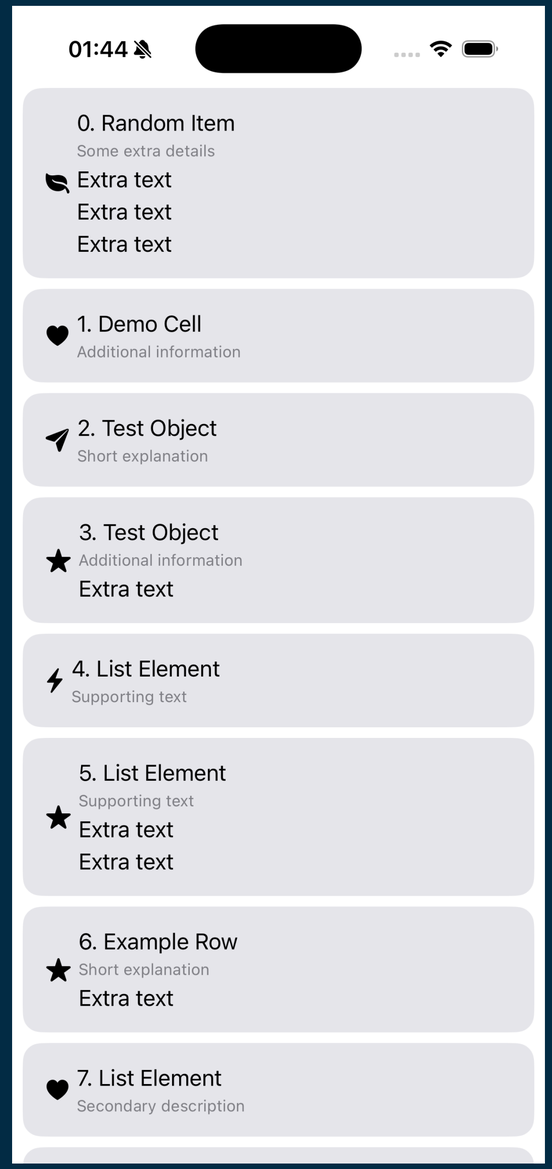
```
Container(spacing: 16) {
    Text("Текст 1")
        .frame(height: 20)

    Text("Текст 2")
        .frame(height: 30)

    Text("Текст 3")
        .frame(height: 40)

    Text("Текст 4")
        .frame(height: 500)

    Text("Текст 5")
        .frame(height: 400)
}
.padding()
```

Демо пример с
«циклическим расчётом высоты 1-го кадра»
у LazyVStack

# Скорость открытия экрана (NavigationTransition) с полноэкранным списком

iPhone 12, iOS 18.6.2

| | Результат |
|---|---|
| LazyVStack | 535 мс. |
| List | 580 мс. |
| OptimizedVStack | 550 мс. |

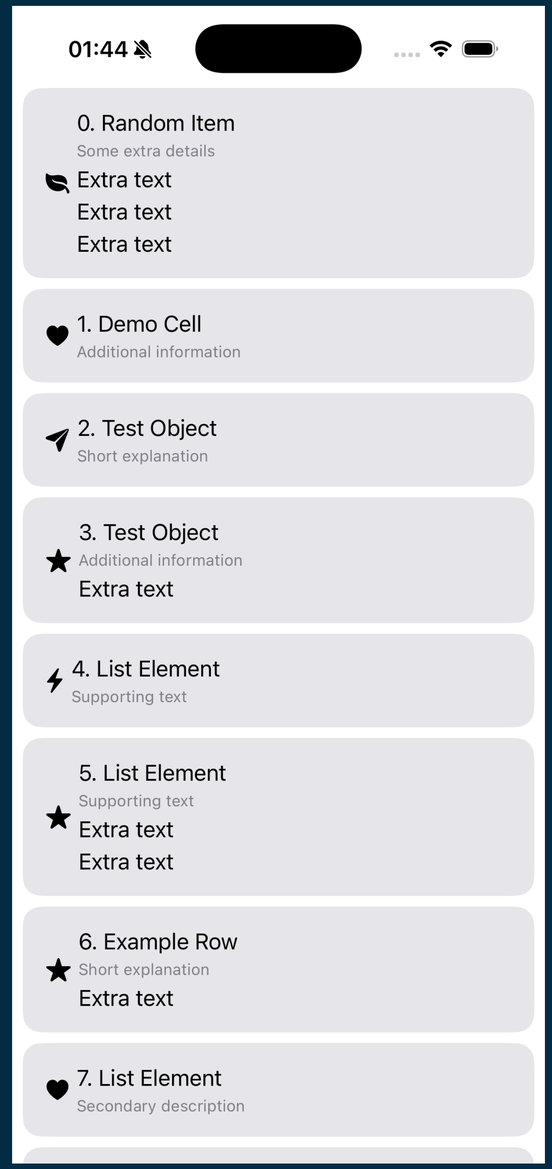



N = 1000

# Hitch-тест «Быстрый скролл через индикатор ScrollView» в течение 30 сек.

iPhone 12, iOS 18.6.2

| | Результат | Max-hitch |
|---|---|---|
| LazyVStack | 610 мс. | 80 мс. |
| List | 3.5 с. | 75 мс. |
| OptimizedVStack | 315 мс. | 70 мс. |

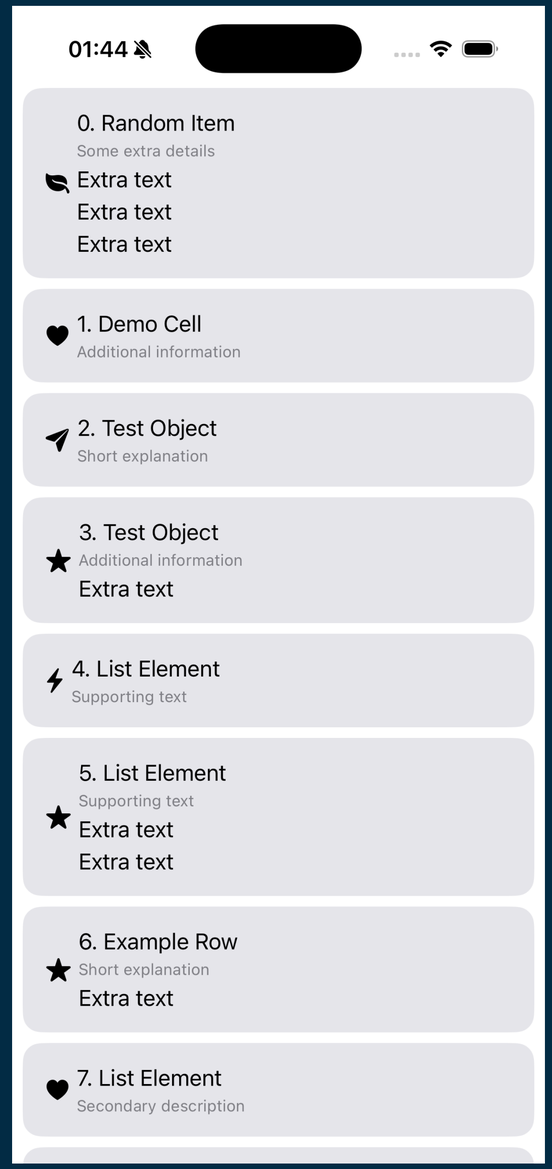



N = 1000

# Пики использования RAM при «Быстром скролле через индикатор ScrollView» в течение 30 сек.

iPhone 12, iOS 18.6.2

| | Результат |
|---|---|
| LazyVStack | 24.5 MB |
| List | 26 MB |
| OptimizedVStack | 21.0 MB |




N = 1000

# Что мы сегодня узнали?

# Вопросы к GeometryReader + LazyVStack

Почему GeometryReader возвращает **неверный** размер для LazyVStack?

Что за числа возвращал GeometryReader?

Почему нет «моргающего» экрана при вычислении неверного размера?

Как понять, что размер **«стабилизировался»**?

# Вопросы к LazyVStack вне GeometryReader

Медленный расчёт кадров

Преждевременное заполнение кэша размеров ячеек

Неожиданная «обрезка» списка

«White screen» в больших гетерогенных списках

# LazyVStack

Включить логирование в OSLog

```
UserDefaults.standard.set(true, forKey: "com.apple.SwiftUI.LazyStackLogging")
```

Найти момент «стабилизации» значения GeometryReader можно через лог

00:00.848.239 Default com.apple.SwiftUI LazyStack LazyVStackLayout: placed(0.0...796.0) -> 0..<5, 0.0...1054.0, **invalid: false**

Если ячейки в LazyVStack **НЕ** имеют одинаковую высоту,

то использовать его — **небезопасно** в контексте ненулевой вероятности «сломанного» Layout

# OptimizedVStack

- Отсутствия «спама» возвращаемой высоты контента на 1 кадр
- Обоснование расчёта «высоты контента»
- Более экономный по памяти и производительный
- Корректный Layout
- Улучшенные логи в OSLog

- Решает проблемы: ScrollView + LazyVStack
- Решает проблемы: GeometryReader + ScrollView + LazyVStack

# OptimizedVStack

- Отсутствия «спама» возвращаемой высоты контента на 1 кадр
- Обоснование расчёта «высоты контента»
- Более экономный по памяти и производительный
- Корректный Layout
- Улучшенные логи в OSLog

- Решает проблемы: ScrollView + LazyVStack
- Решает проблемы: GeometryReader + ScrollView + LazyVStack

**iOS 14 и новее**

OZON tech

# Спасибо! Вопросы?

## Полезные ссылки

https://github.com/alextar04/OptimizedVStack

Гайд по поиску AttributeGraph

Репозиторий с демо-примерами