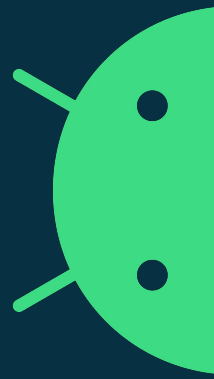
Декларативные UI фреймворки

От динозавров к Jetpack Compose



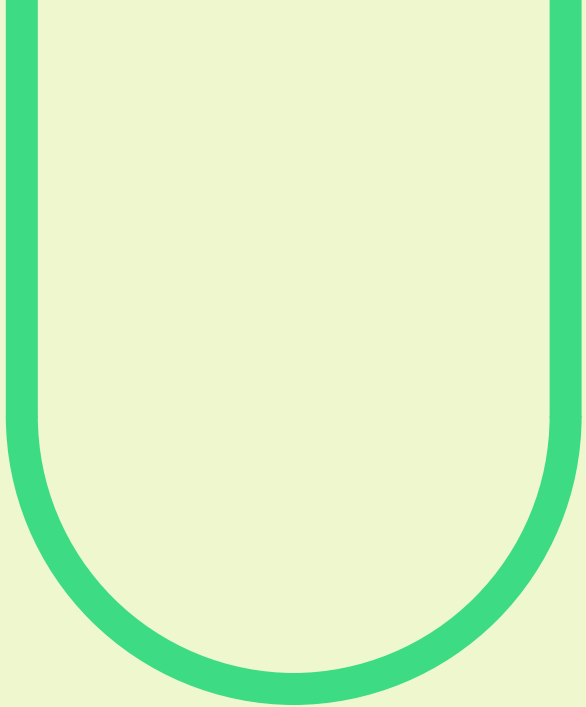
Матвей Мальков, Google UK, Android Toolkit

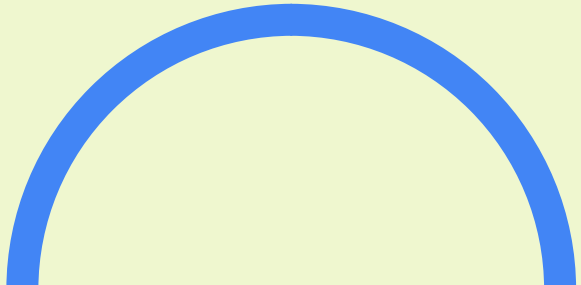
В программе

- **Вчера:** что такое декларативные фреймворки и откуда они взялись
- **Сегодня:** Как мы пишем UI и какие с этим проблемы
- **Завтра:** как решаются сегодняшние проблемы; строим свой декларативный UI фреймворк
- **Счастливое завтра:** как мы строим Jetpack Compose

Вчера

О декларативном и императивном





Declarative > Imperative

Declarative ≠ Imperative

Declarative &> Imperative

In computer science, **declarative programming** is a programming paradigm — a style of building the structure and elements of computer programs—that expresses the logic of a computation without describing its control flow

Не КАК, а ЧТО

How

```
fun sum(arr: Array<Int>) {
    var result = 0
    for (i in 0 until arr.size) {
        result += arr[i]
    }
    return result
}
```

What

```
fun sum(arr: Array<Int>): Int {
    return arr.reduce { acc, item ->
        acc + item
    }
}
```

Пример попроще

```
val a = 4

val b = -2

val result = a + b
```

Поколения

языков программирования

Поколение 1: машинные языки

1010011010110100110101101101101011010110100110101101
1010110110110101101011010011010110100110101101101011
0
1011010011010110100110101101101011010110100110101101
0011010110110110101101011010011010110100110101101101
0110101101001101011010011010110100110101101001101010
1101001101011011010110100110101101101011010011010110
1010110101101001101011010011010110110101101011010011
0101101001101011011011010110101101001101.

Поколение 1: машинные языки

```
00100001 00000100
00110001 00000101
00110001 00000110
01110000 00000001
00000000 00000100
11111111 11111110
00000000 00000000
```

Поколение 1: машинные языки

```
00100001 00000100
00110001 00000101
00110001 00000110
01110000 00000001
00000000 00000100
11111111 11111110
00000000 00000000
```

Поколение 1: машинные языки

```
00100001 00000100
00010001 00000101
00110001 00000110
01110000 00000001
00000000 00000100
11111111 11111110
00000000 00000000
```

Поколение 2: ассемблерные языки

How

```
00100001 00000100
00010001 00000101
00110001 00000110
01110000 00000001
00000000 00000100
11111111 11111110
00000000 00000000
```

What

```
ORG 100
LDA A
ADD B
STA C
HLT
A, DEC 4
B, DEC -2
C,
DEC 0
END
```

Поколение 2: ассемблерные языки

How

```
00100001 00000100
00010001 00000101
00110001 00000110
01110000 00000001
00000000 00000100
11111111 11111110
00000000 00000000
```

What

```
ORG 100
LDA A
ADD B
STA C
HLT
A, DEC 4
B, DEC -2
C,
DEC 0
END
```

Поколение 3: языки высокого уровня

How

```
ORG 100
LDA A
ADD B
STA C
HLT
A, DEC 4
B, DEC -2
C,
DEC 0
END
```

What

```
val a = 4

val b = -2

val result = a + b
```

Поколение 4: языки уровня ***повыше***

How

```
fun sum(arr: Array<Int>) {
    var result = 0
    for (i in 0 until arr.size) {
        result += arr[i]
    }
    return result
}
```

What

```
fun sum(arr: Array<Int>): Int {
    return arr.reduce { acc, item ->
        acc + item
    }
}
```

4 поколение

- Абстрагирует 3е
- Более простое в использовании
- Для более **сложных** задач
- Делают **новые**, более сложные задачи более простыми
- 1C, R, SQL, RegEx
- DSLs: React, RxJava, JetpackCompose

Декларативность сделана на императивности

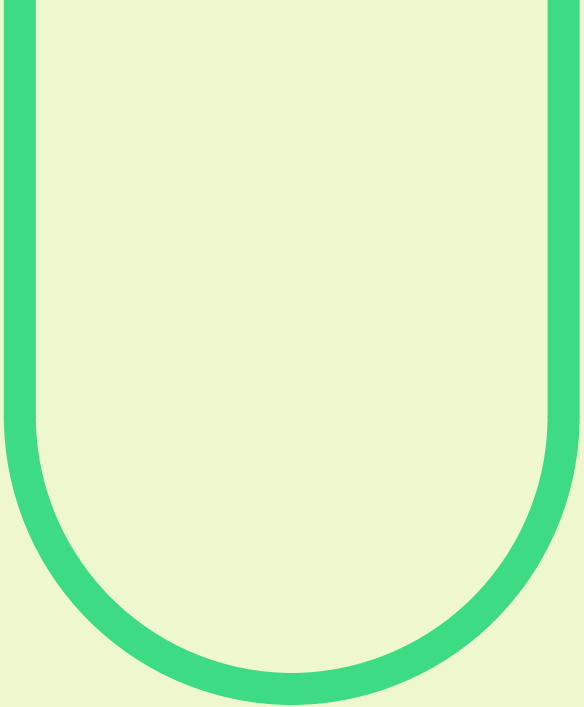
Кто-то должен объяснить *“КАК”*, чтобы мы говорили *“ЧТО”*

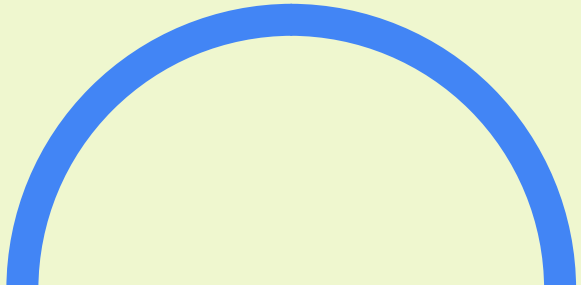
Возможность убежать в императив

- **C, C++:** intrinsics, inline ASM
- **React:** element reference, `componentDidUpdate`
- **RxJava:** `PublishSubject`
- **Jetpack Compose:** imperative drawing, imperative touch-handling, `onCommit { … }`

UI фреймворки

Вчера и сегодня





HTML

- Язык разметки: размечаем, ЧТО должно быть на экране
- Строим дерево и показываем его
- Очень декларативно, но не зависит от данных
- UI = $f()$

main_layout.xml

- Язык разметки: размечаем, ЧТО должно быть на экране
- Строим дерево и показываем его
- Очень декларативно, но не зависит от данных
- UI = $f()$

main_layout.xml

- Язык разметки: размечаем, ЧТО должно быть на экране
- Строим дерево и показываем его
- Очень декларативно, но не зависит от данных
- UI = $f()$

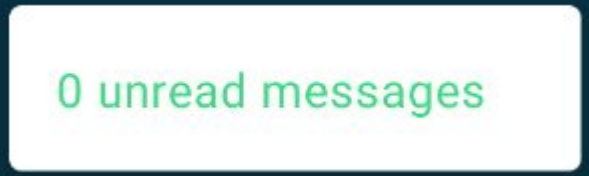

eBay - Your Personal Trading Community - Netscape
File Edit View Go Communicator Help
home | my eBay | site map
Browse Sell Services Search Help Community
eBay
what are you looking for?
Find it! tips
categories
Automotive NEW!
Antiques (71865)
Books, Movies, Music (633477)
Coins & Stamps (117638)
Collectibles (978255)
Computers (101886)
Dolls, Figures (39138)
Jewelry, Gemstones (128668)
Photo & Electronics (55292)
Pottery & Glass (191042)
Sports Memorabilia (345439)
Toys, Bean Bag Plush (290114)
Miscellaneous (311778)
all categories...
Shop by photos in the Gallery
great collections
Visit eBay Great Collections, our new area for fine art, antiques, and collectibles!
Sell your item
Get news and chat
new users Click here
Register It's free and fun
featured
New State Quarters Album - Low Shipping
Softest Leather Passport Wallet ! Msrp 34.95!
LEWIS v HOLYFIELD $2000 face value tickets
M Jordan Basketball*From The Restaurant* Pic
Gorgeous Genuine Sapphire 1 Carat Nice Blue
Ernie Els' Omega Constellation Two-Tone Watch
more! see all featured....
eBay Visa Apply now!
welcome
What is eBay?
How do I bid?
How do I sell?
Register, it's free!
fun stuff
cool happenings...
Joe DiMaggio
Welcome back Rosie! Bid on items to benefit the For All Kids Foundation
Hot off the press - eBay Magazine! Get your subscription now!
Go Local! Now you can search within your region!
stats
3,090,394 items for sale in 2,201 categories now!
Over 1.5 billion page views per month!
35% of 2K

Динамический UI

Изменения на лету

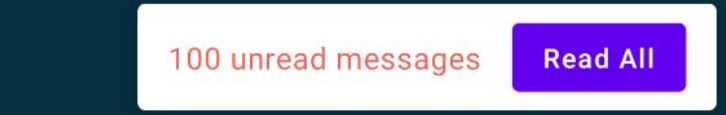

100 unread messages
Read All

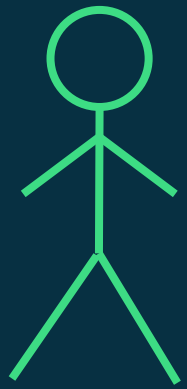

100 unread messages
Read All

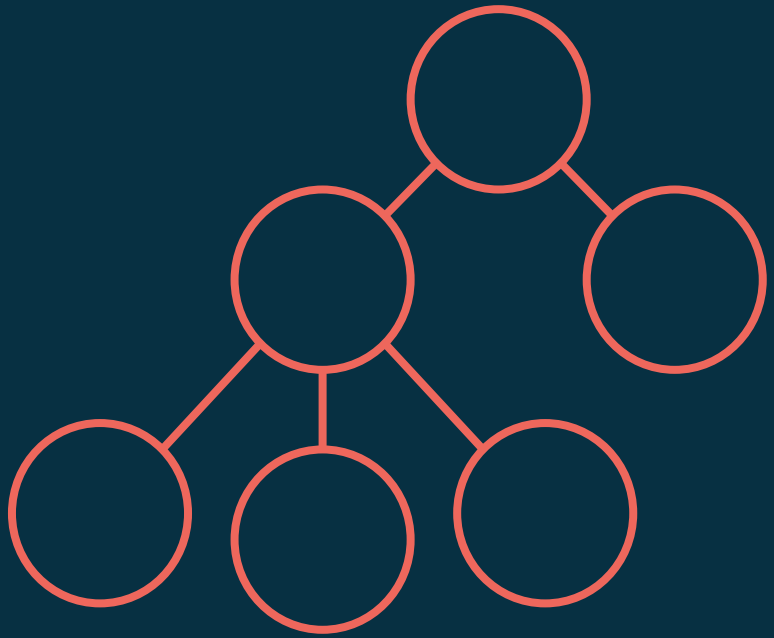

0 unread messages

Динамический UI

Динамический UI

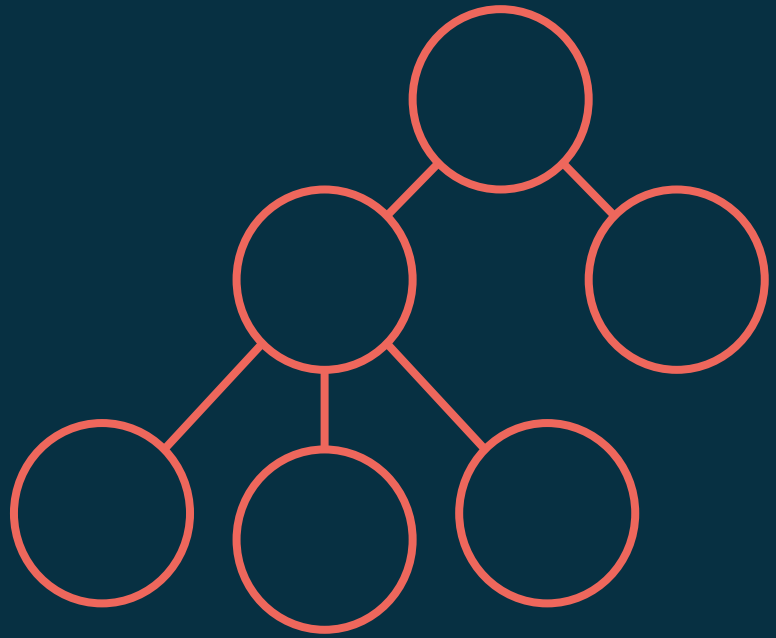




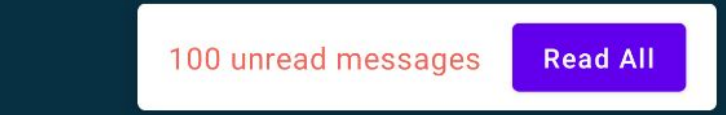


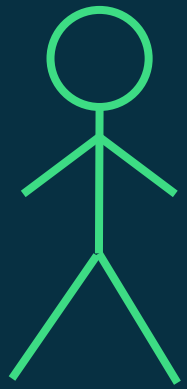
Динамический UI

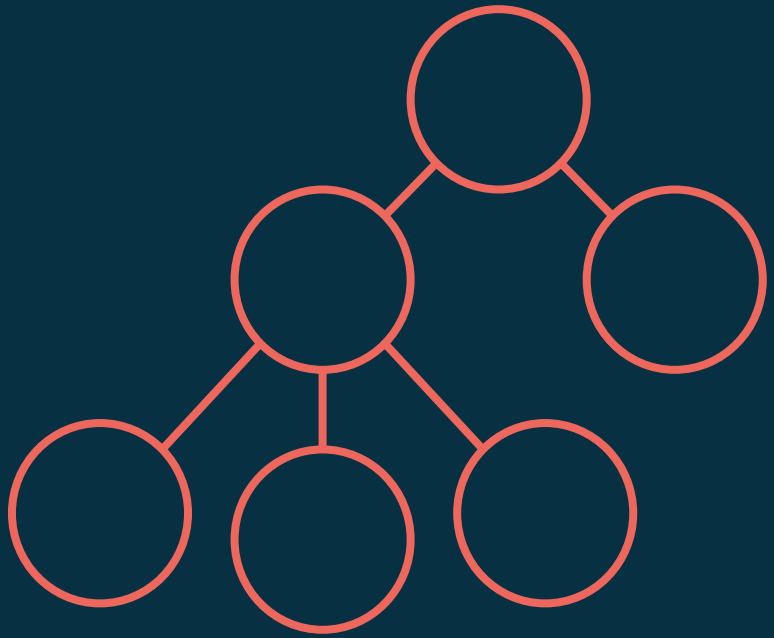
Капитан
Динамичность



Динамический UI

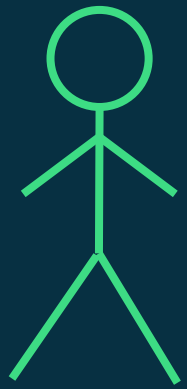






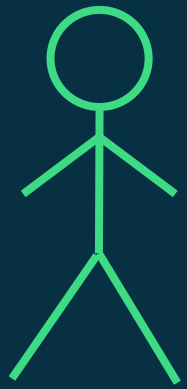
Динамический UI

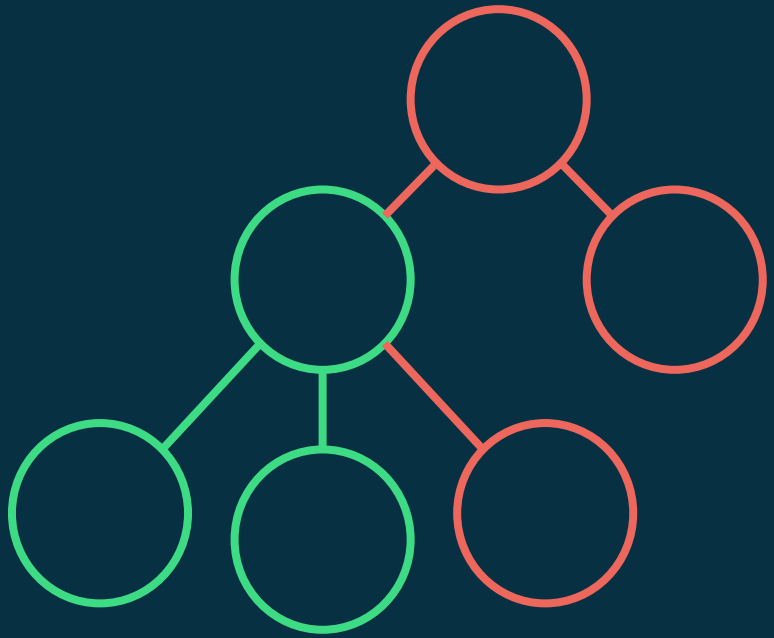
- Нажали “Read All”



Динамический UI

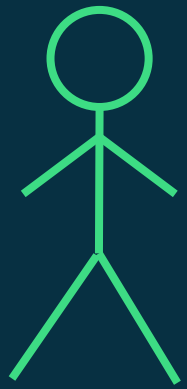
- Нажали “Read All”





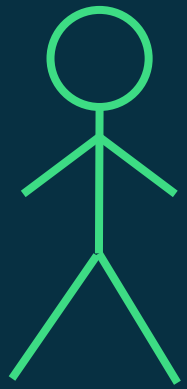
Динамический UI

- Нажали “Read All”

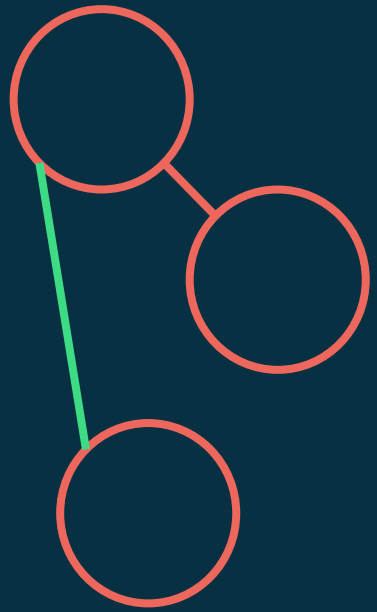


Динамический UI

- Нажали “Read All”

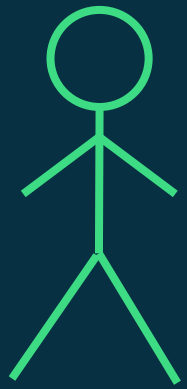


0 unread messages



Динамический UI

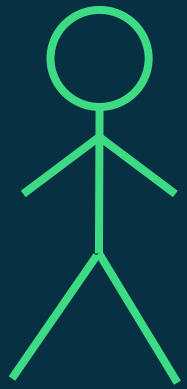
- Нажали “Read All”



0 unread messages

Динамический UI

- Нажали “Read All”
- Пришло новое сообщение

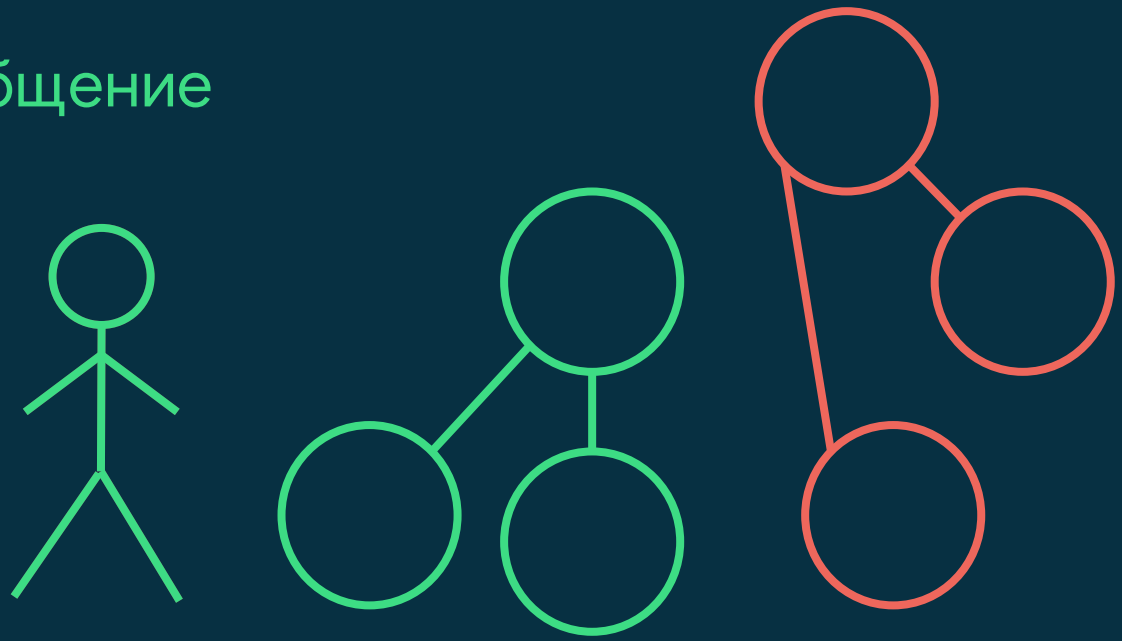


0 unread messages

Динамический UI

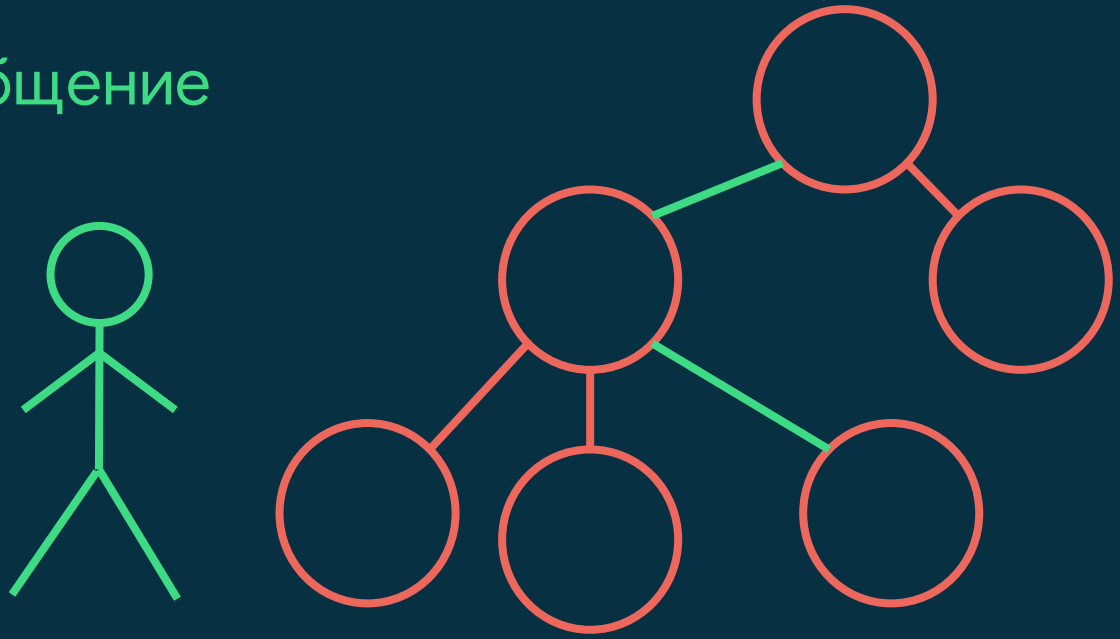
- Нажали “Read All”
- Пришло новое сообщение

0 unread messages



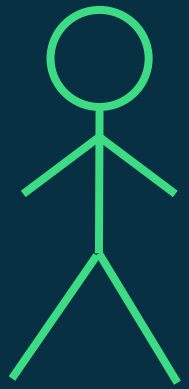
Динамический UI

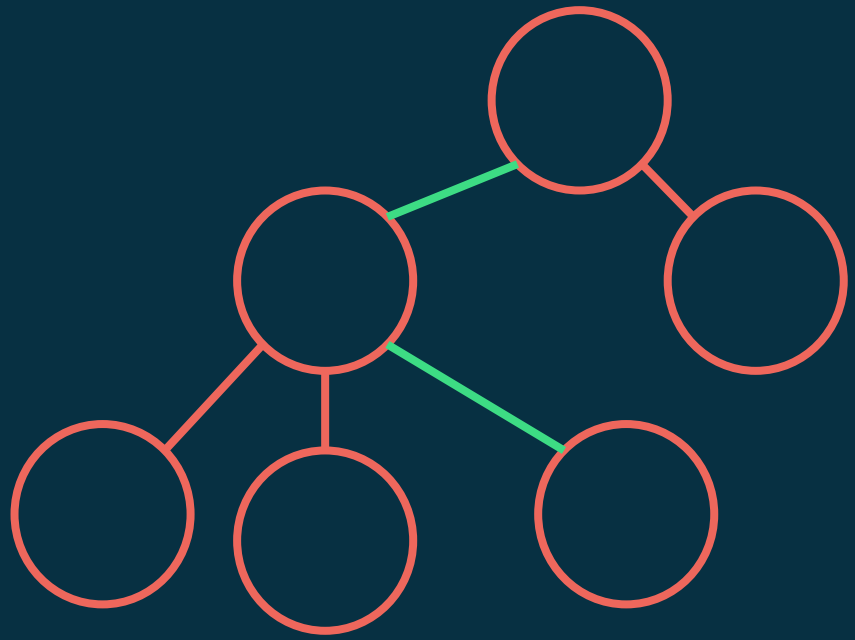
- Нажали “Read All”
- Пришло новое сообщение



Динамический UI

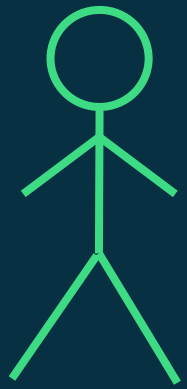
- Нажали “Read All”
- Пришло новое сообщение

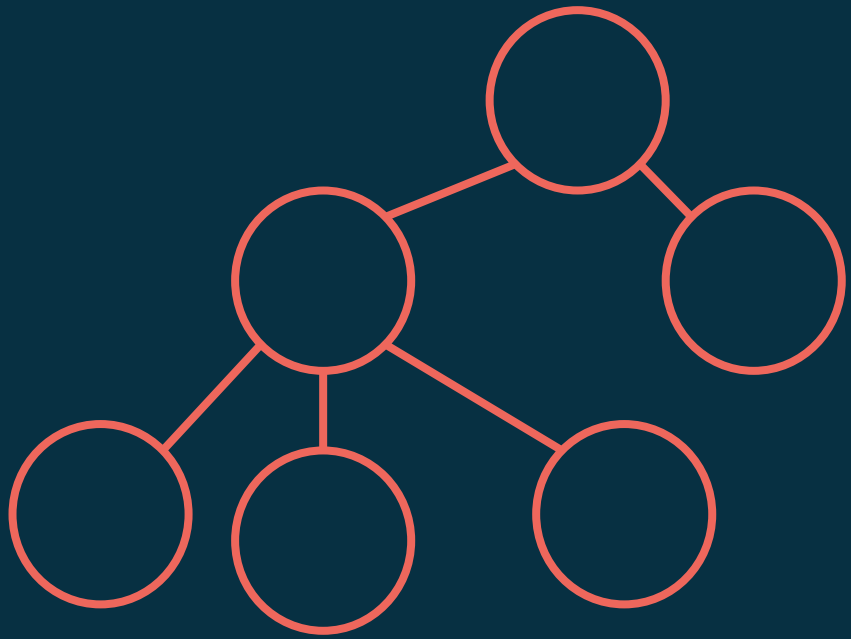




Динамический UI

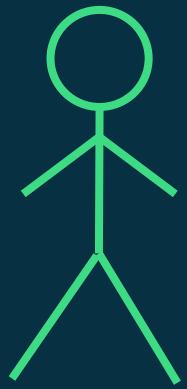
- Нажали “Read All”
- Пришло новое сообщение

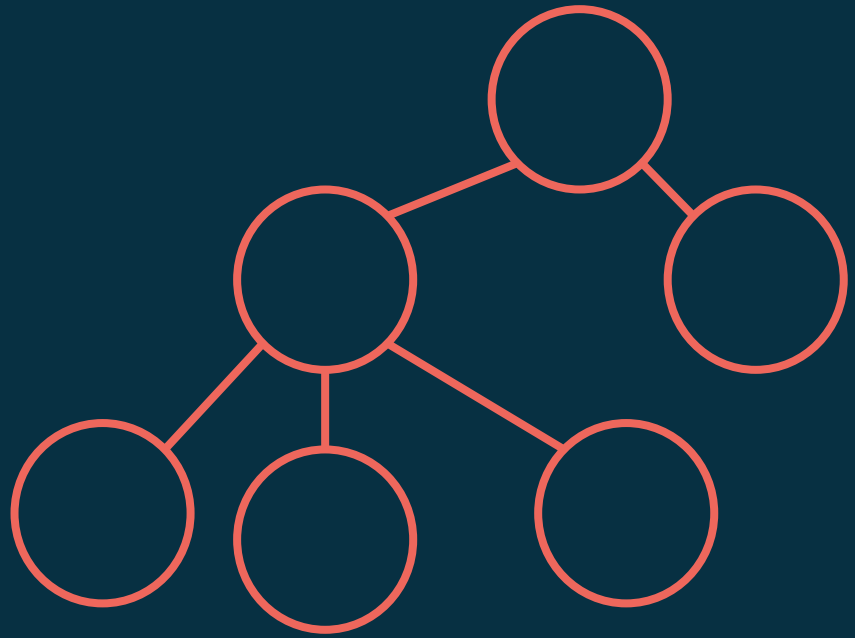




Динамический UI

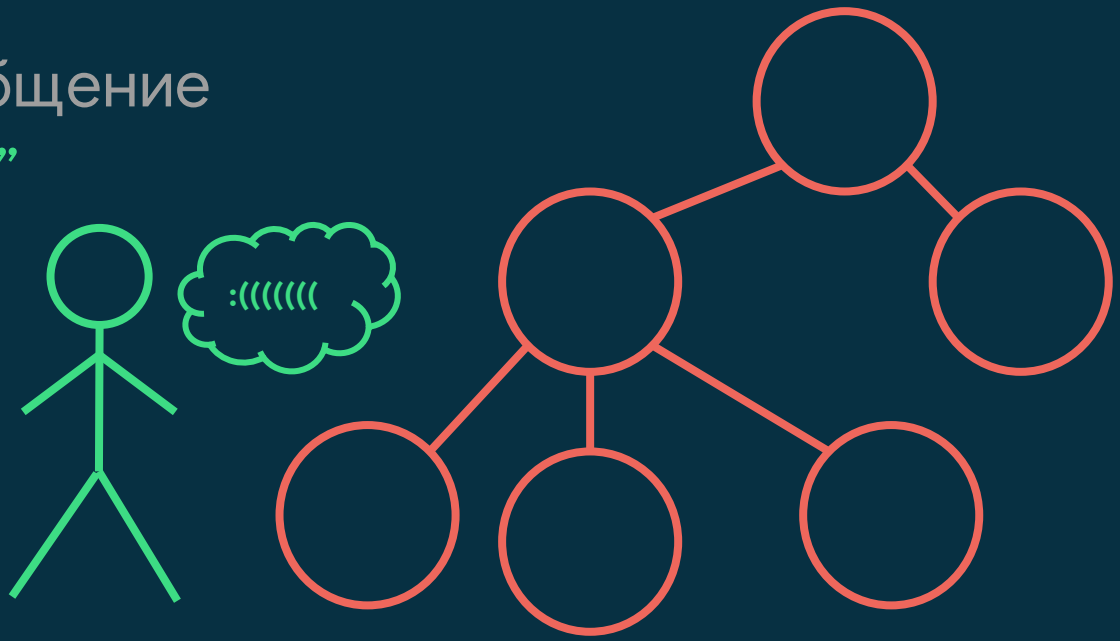
- Нажали “Read All”
- Пришло новое сообщение
- Новая кнопка “undo”





Динамический UI

- Нажали “Read All”
- Пришло новое сообщение
- Новая кнопка “undo”



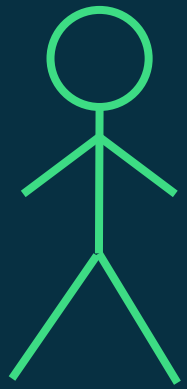
Динамический UI. JavaScript

```
val unreadRoot =
document.getElementsByClassName("unreadRoot");

val count = document.createElement('div');

count.textContent = 'You have $count unread messages';

unreadRoot.appendChild(count);
```

Динамический UI. Kotlin

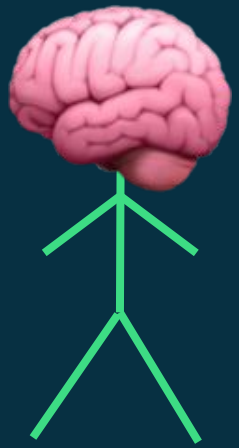
```
val unreadIcon = findViewById(R.id.unread)
if (unreadCount == 0) {
    unreadIcon.setVisibility(GONE)
} else if (unreadCount > 0) {
    unreadIcon.setVisibility(VISIBLE)
    unreadIcon.setCount(unreadCount)
    var hasImportant = unreadMessages.exists(it.isImportant)
    if (hasImportant) {
        unreadCount.setTextColor(Red)
    } else {
        unreadCount.setTextColor(Blue)
    }
}
```

Динамический UI



Динамический UI

- Много состояний
- Еще больше -- переходов между ними



UI фреймворки

Сегодня и завтра

Новый UI фреймворк

- Убирает переходы между состояниями (для разработчика)
- UI = f(state)
- Объединяет markup и динамическое обновление

UI = ƒ(state)

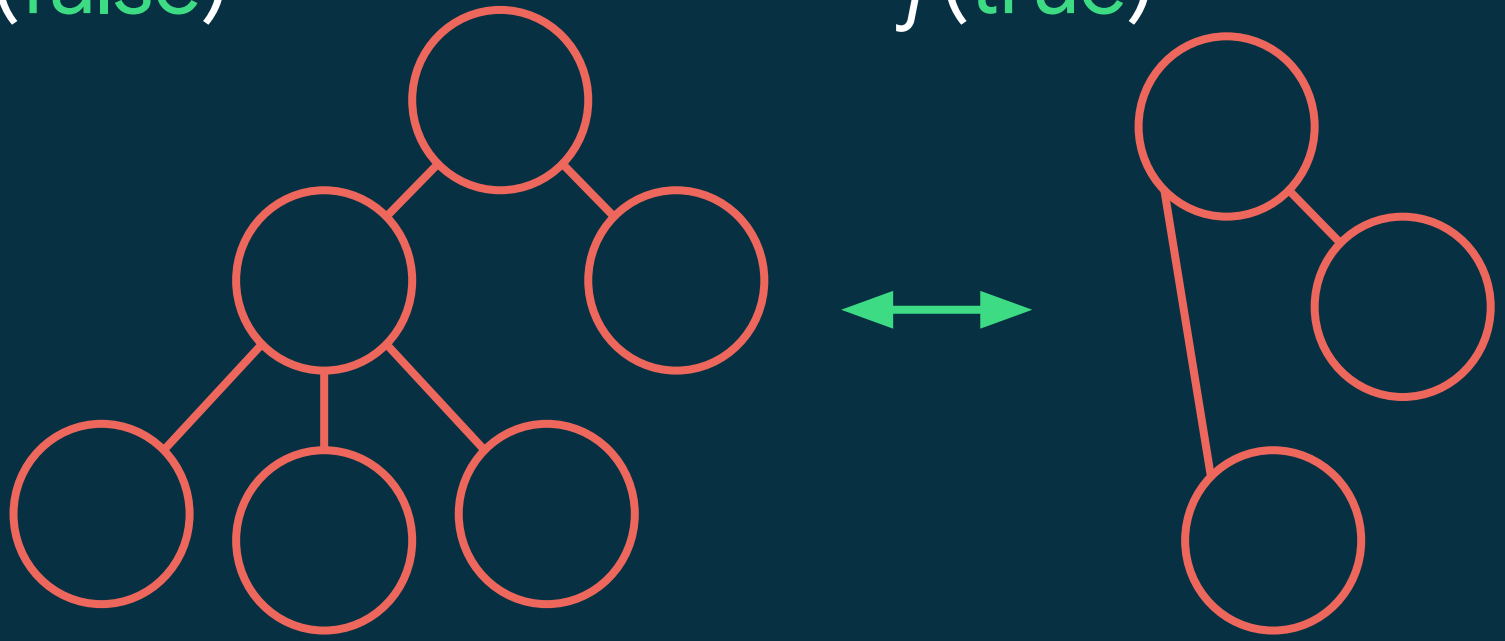
ƒ(false)

ƒ(true)

UI = ƒ(state)

ƒ(false)

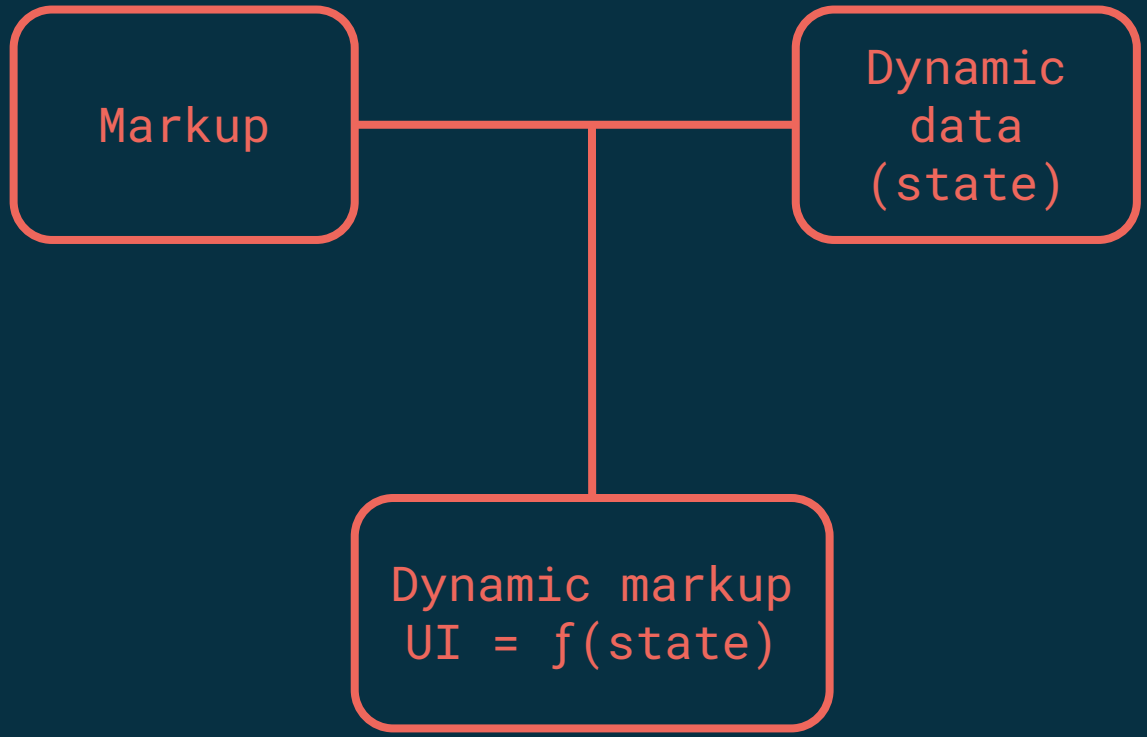
ƒ(true)



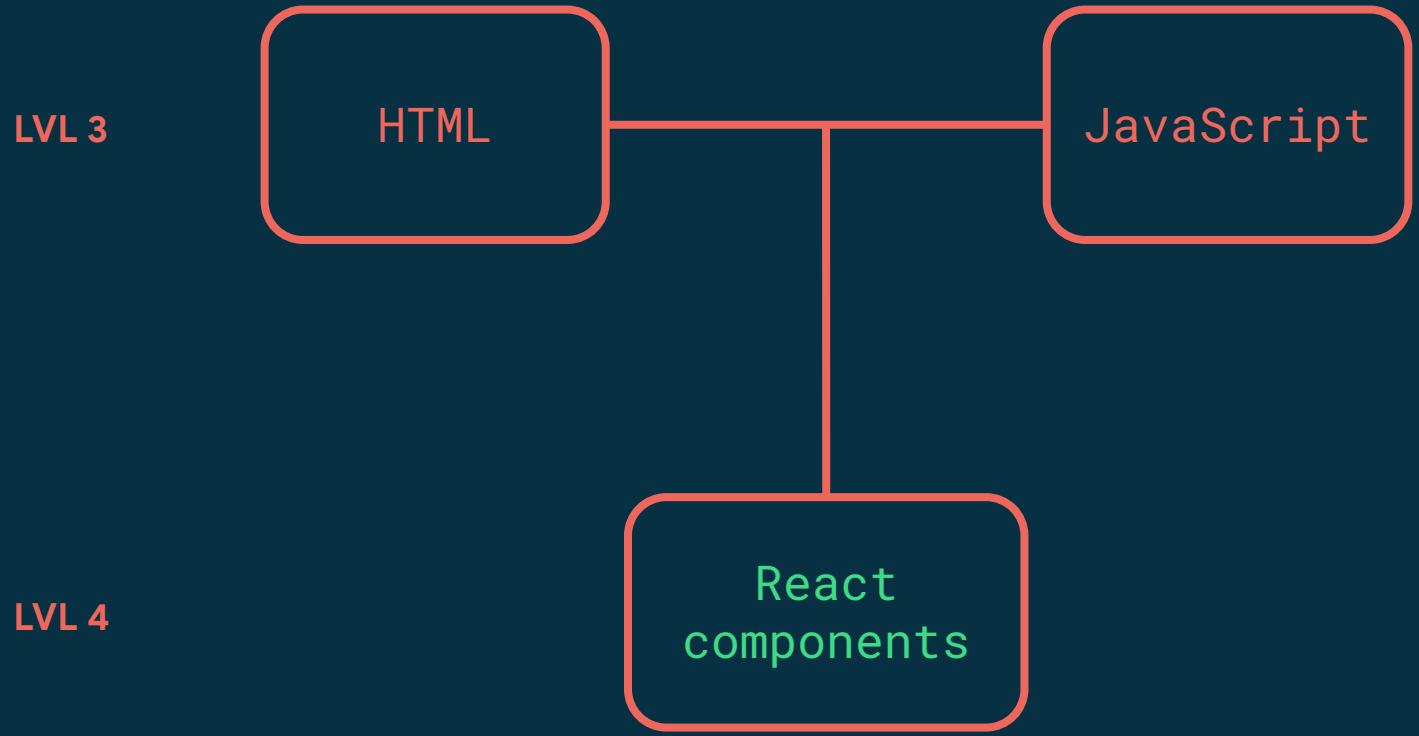
LVL 3

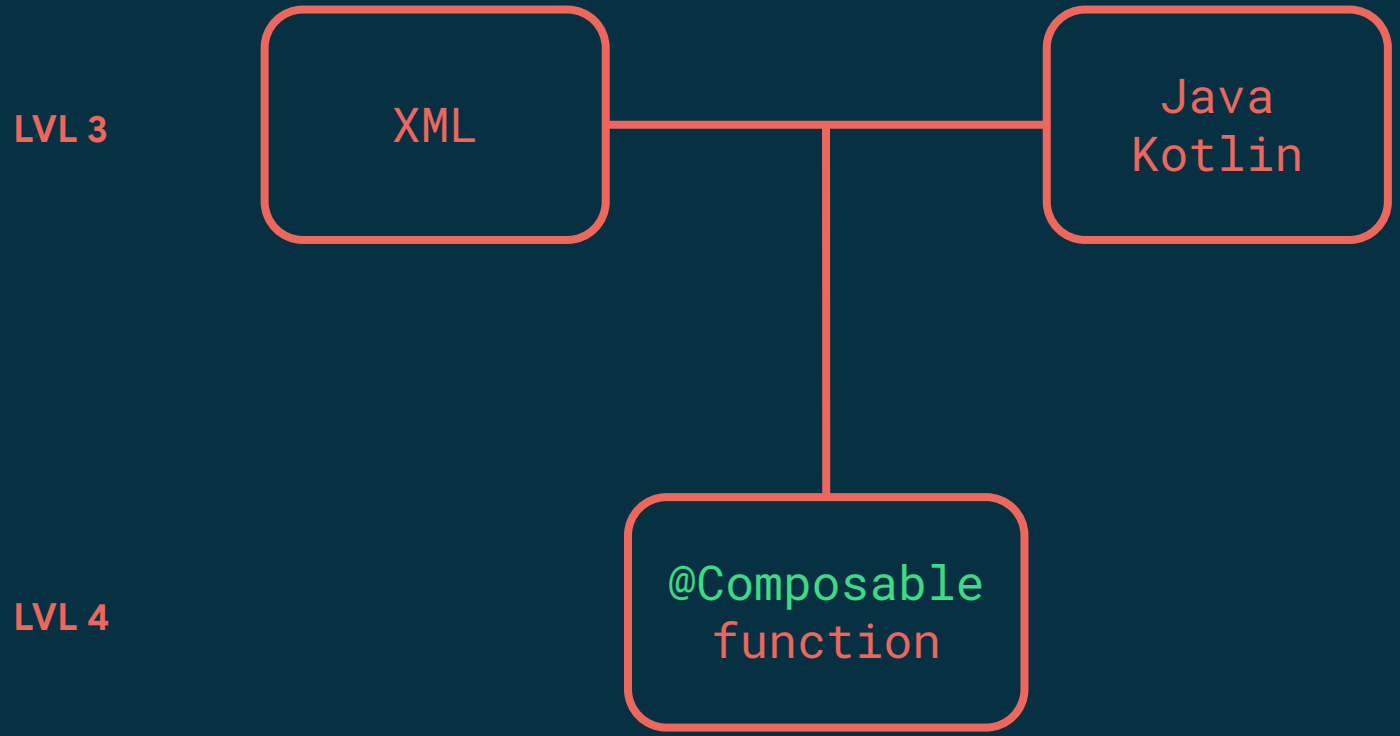
LVL 3

LVL 3



LVL 4


LVL 3
HTML
JavaScript
LVL 4
React
components


LVL 3
XML
Java
Kotlin
LVL 4
@Composable
function

@Composable

```
@Composable
fun UnreadSection(count: Int) {
    Text(
        text = "You have $count unread messages",
        color = if (count > 10) Color.Red else Color.Green
    )
    if (count > 0) {
        ReadAllButton()
    }
}
```

@Composable

```
@Composable
fun UnreadSection(count: Int) {
    Text(
        text = "You have $count unread messages",
        color = if (count > 10) Color.Red else Color.Green
    )
    if (count > 0) {
        ReadAllButton()
    }
}
```

@Composable

```
@Composable
fun UnreadSection(count: Int) {
    Text(
        text = "You have $count unread messages",
        color = if (count > 10) Color.Red else Color.Green
    )
    if (count > 0) {
        ReadAllButton()
    }
}
```

@Composable

```
@Composable
fun UnreadSection(count: Int) {
    Text(
        text = "You have $count unread messages",
        color = if (count > 10) Color.Red else Color.Green
    )
    if (count > 0) {
        ReadAllButton()
    }
}
```

@Composable

```
@Composable
fun UnreadSection(count: Int) {
    Text(
        text = "You have $count unread messages",
        color = if (count > 10) Color.Red else Color.Green
    )
    if (count > 0) {
        ReadAllButton()
    }
}
```

@Composable

```
@Composable
fun UnreadSection(count: Int) {
    Text(
        text = "You have $count unread messages",
        color = if (count > 10) Color.Red else Color.Green
    )
    if (count > 0) {
        ReadAllButton()
    }
}
```

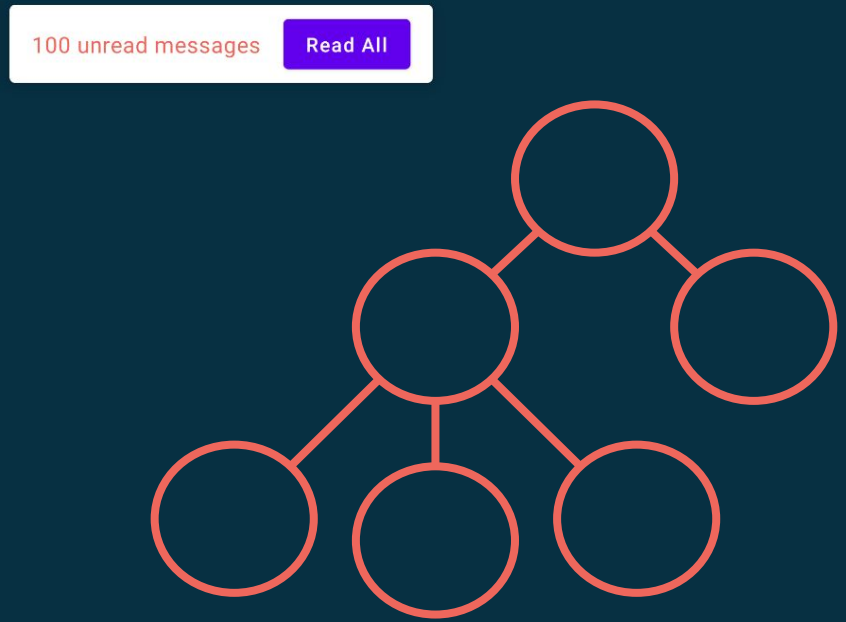
UI = ƒ(state)

f(false)

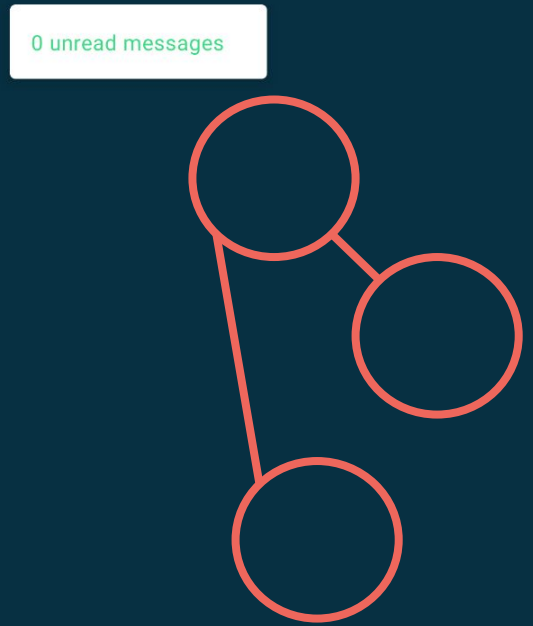
f(true)

UI = ƒ(state)

`UnreadSection(count = 100)`

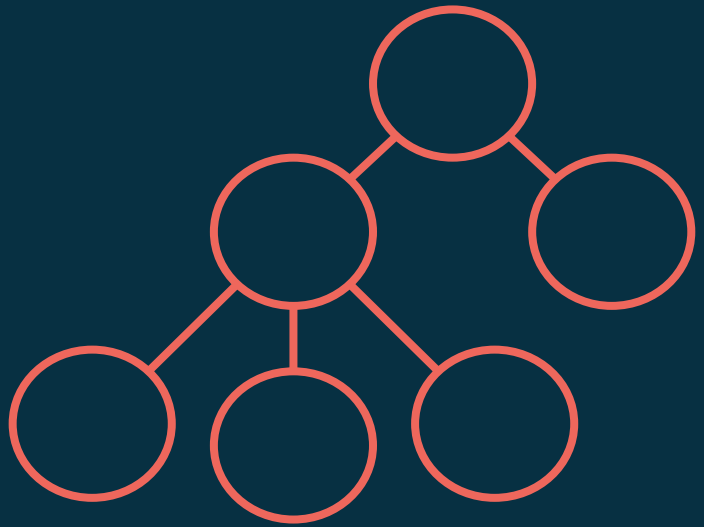


`UnreadSection(count = 0)`



UI = ƒ(state)

`UnreadSection(count = 100)`

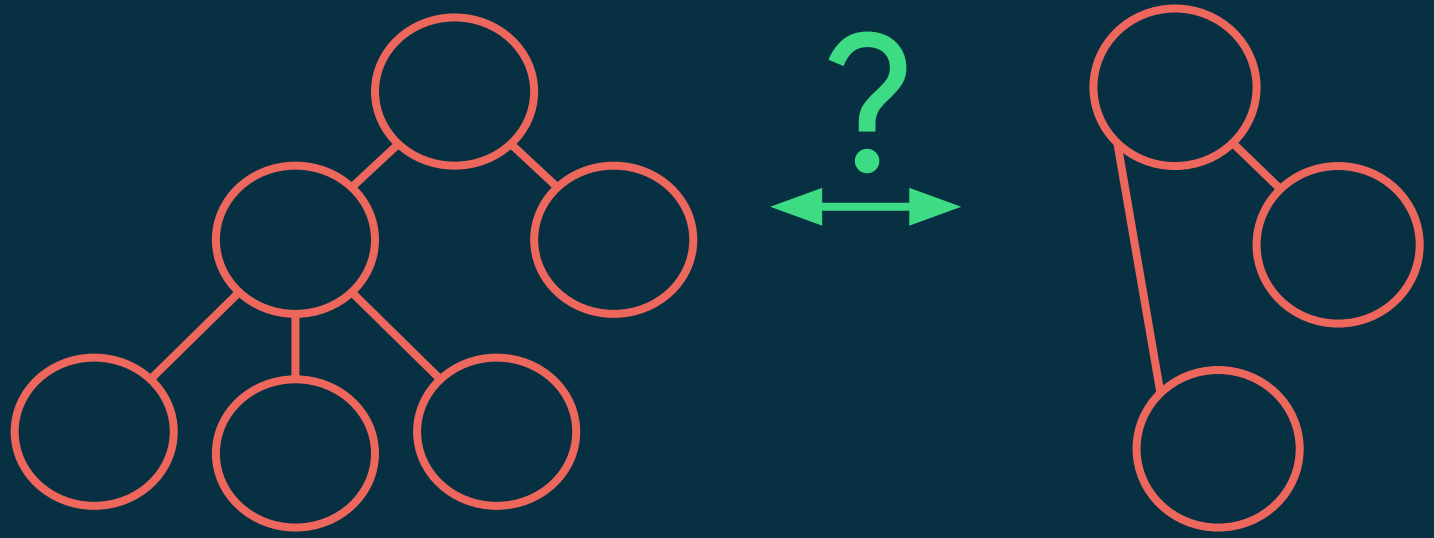


`UnreadSection(count = 0)`

UI = ƒ(state)

`UnreadSection(count = 100)`

`UnreadSection(count = 0)`



Заменить экран с одного на другой -- дорого :(

Идея!

- Используем **виртуальную**, дешевую версию реального UI
- При изменении состояния строим новую виртуальную версию
- Считаем diff между виртуальными версиями
- Выражаем diff как императивные операции над реальным UI

UI = ƒ(true)

UI = ƒ(true)

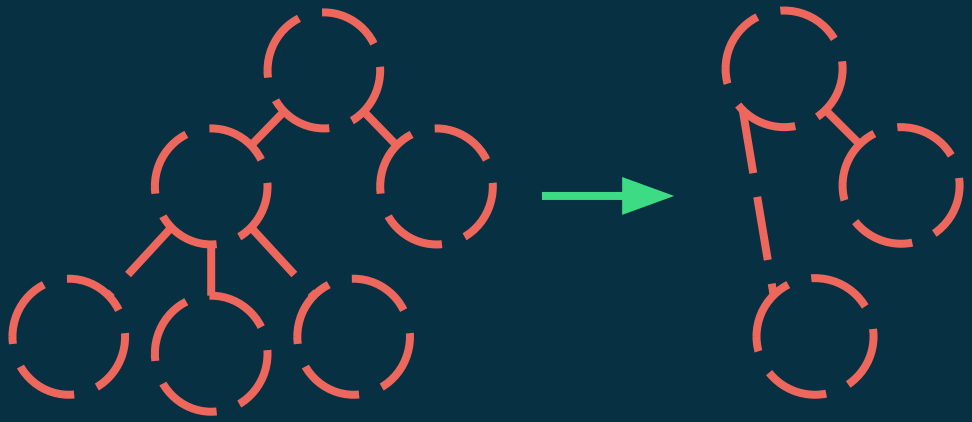
Виртуальная репрезентация

Реальный UI (экран)

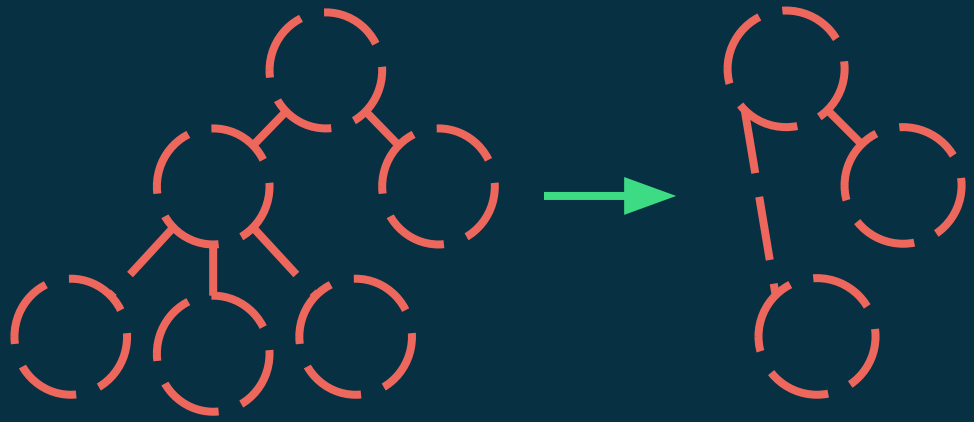
UI = ƒ(true)

UI = ƒ(false)

UI = ƒ(false)



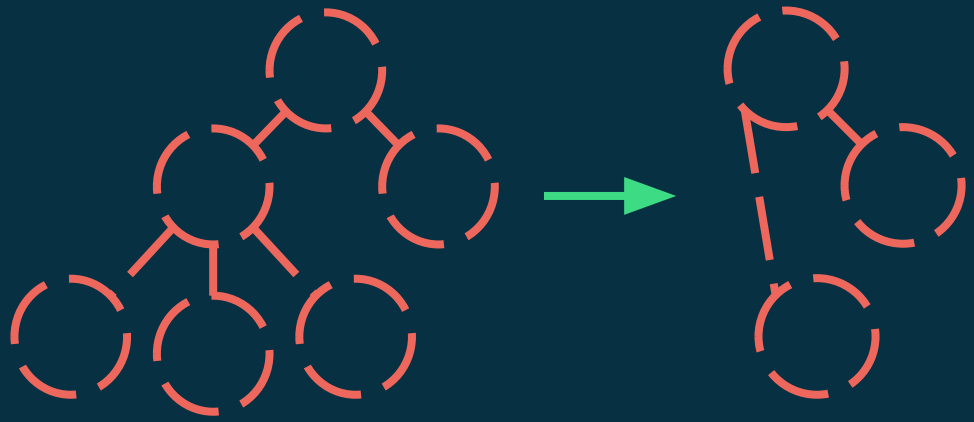
UI = ƒ(false)



preudocode log:

1. Remove first child of the root
2. Create new leaf
3. Assign leaf as a first child of the root

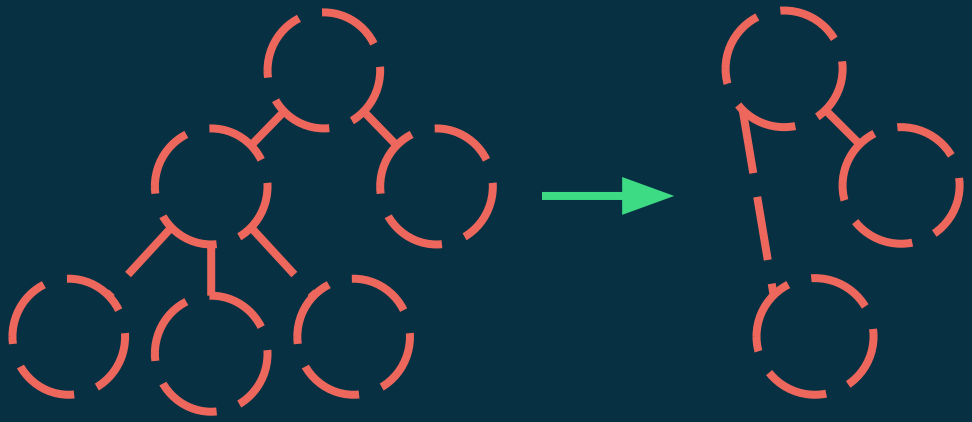
UI = ƒ(false)



preudocode log:

1. Remove first child of the root <---
2. Create new leaf
3. Assign leaf as a first child of the root

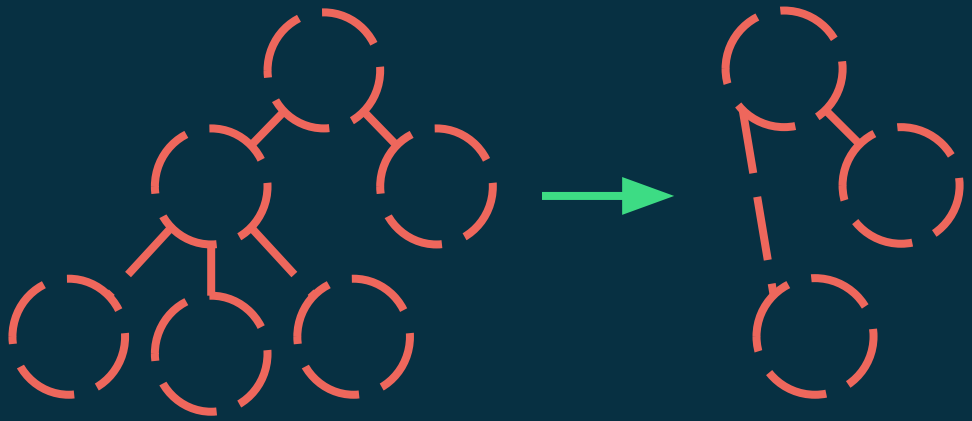
UI = ƒ(false)



preudocode log:

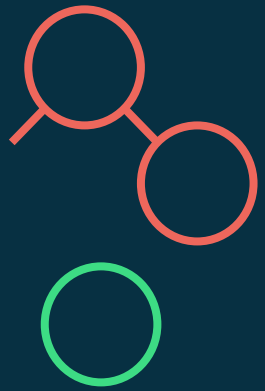
1. Remove first child of the root <---
2. Create new leaf
3. Assign leaf as a first child of the root

UI = ƒ(false)

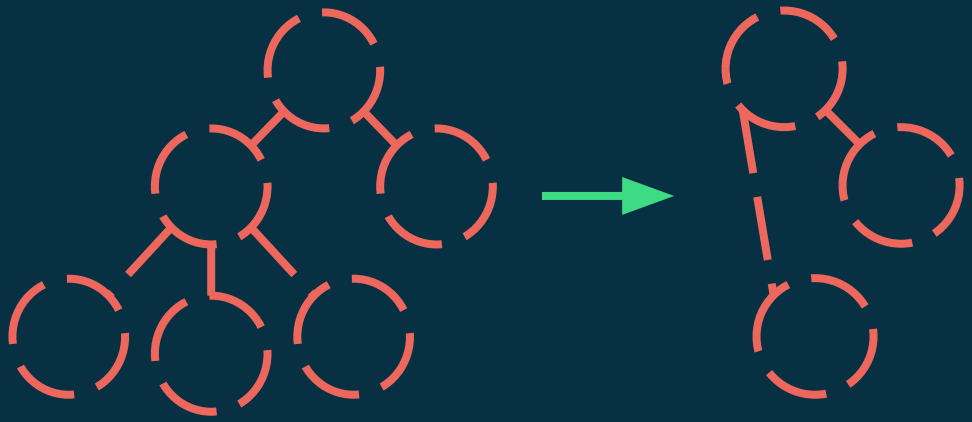


preudocode log:

1. Remove first child of the root
2. Create new leaf <---
3. Assign leaf as a first child of the root

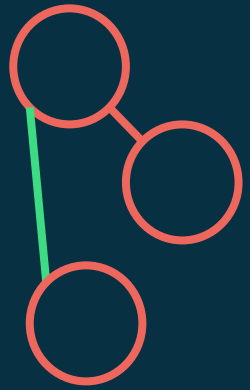


UI = ƒ(false)

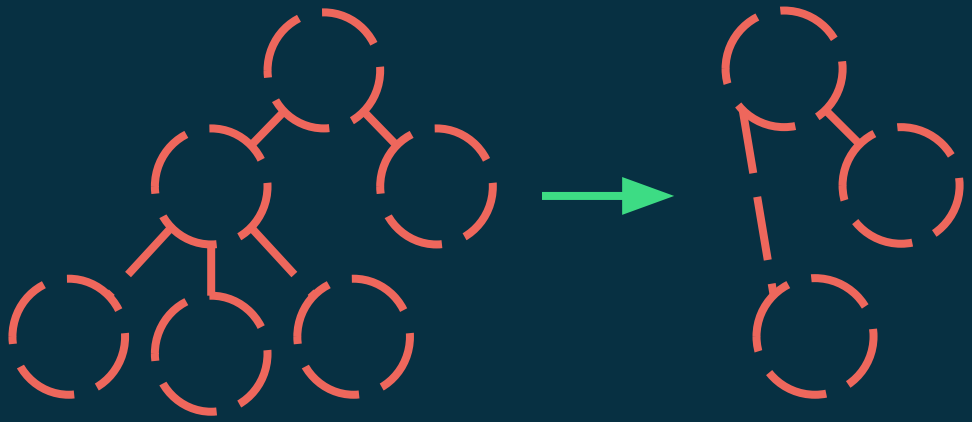


preudocode log:

1. Remove first child of the root
2. Create new leaf
3. Assign leaf as a first child of the root <---

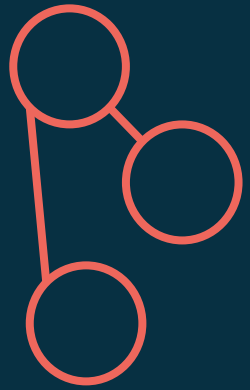


UI = ƒ(false)

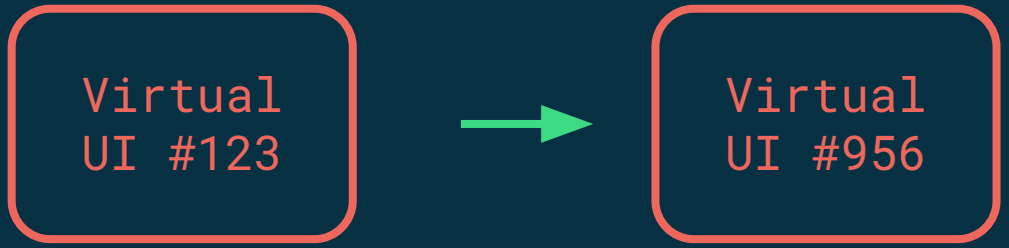


preudocode log:

1. Remove first child of the root
2. Create new leaf
3. Assign leaf as a first child of the root

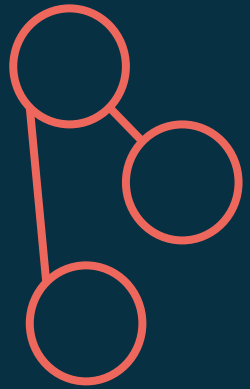


UI = ƒ(false)

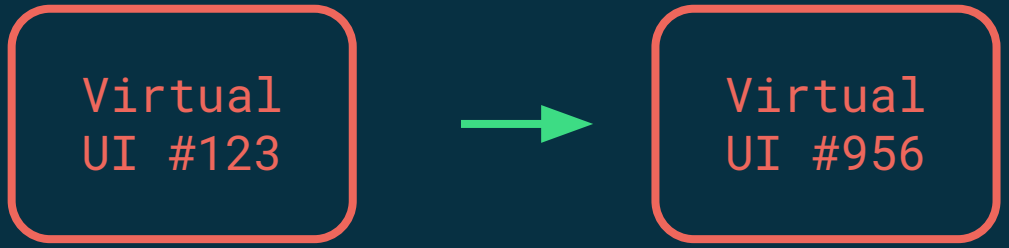


preudocode log:

1. Remove first child of the root
2. Create new leaf
3. Assign leaf as a first child of the root

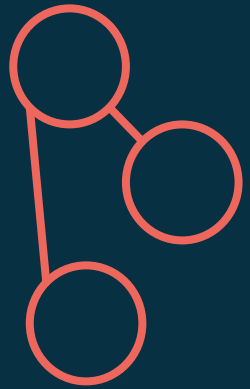


UI = ƒ(false)

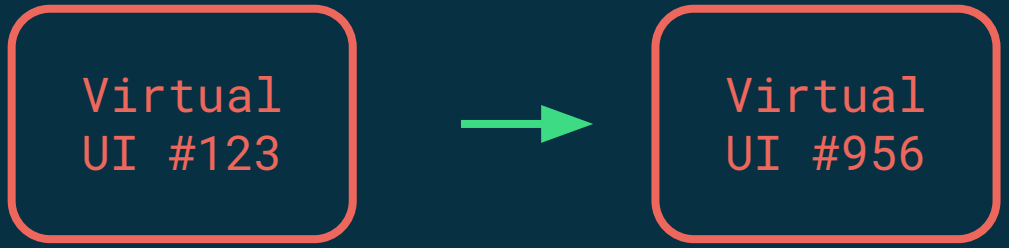


Reconciliation

1. Remove first child of the root
2. Create new leaf
3. Assign leaf as a first child of the root



UI = ƒ(false)



Reconciliation (auto):

1. Remove first child of the root
2. Create new leaf
3. Assign leaf as a first child of the root

Reconciliation

auto 

Reconciliation

- **Задача:** построить минимальный набор команд для трансформации дерева А в дерево Б

- **Проблема:** Общий алгоритм построения набора операций, необходимых для трансформации одного дерева в другое имеет скорость O(n^3)

Reconciliation. Решение

- Хочешь быстрее? Поставь кэшик!
- Упрощение **1:** Если это ***тот же самый*** элемент, это мы можем его обновить, а не пересчитывать полностью его поддерево
- Упрощение **2:** Если входные данные этого компонента ***не меняются*** -- не меняется и поддерево, которое ему соответствует
- Тогда можем обновлять за **O(n)**

Декларативный UI фреймворк

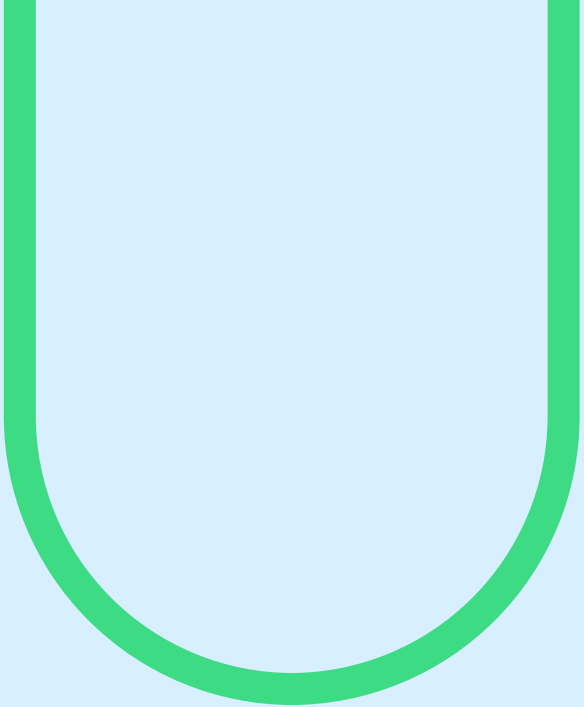
- UI = f(state)
- **Для нас:** Убирает переходы между состояниями
- **Внутри:** Оптимально осуществляет эти переходы (Reconciliation)

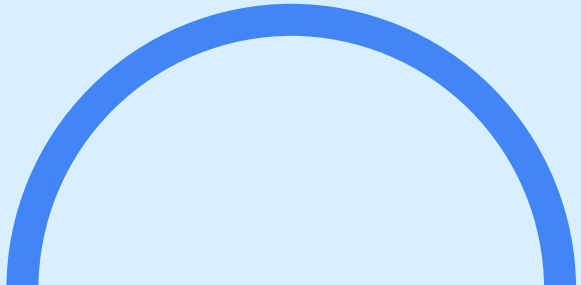
Декларативный UI фреймворк

- UI = f(state)
- **Для нас:** Убирает переходы между состояниями
- **Внутри:** Оптимально осуществляет эти переходы (Reconciliation)

Jetpack Compose

внутри*

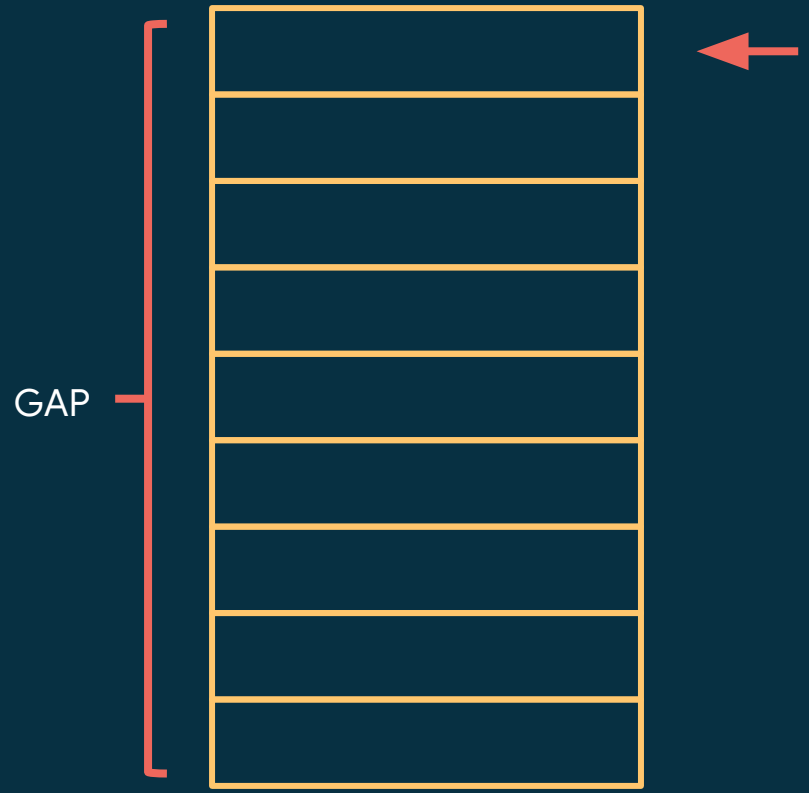




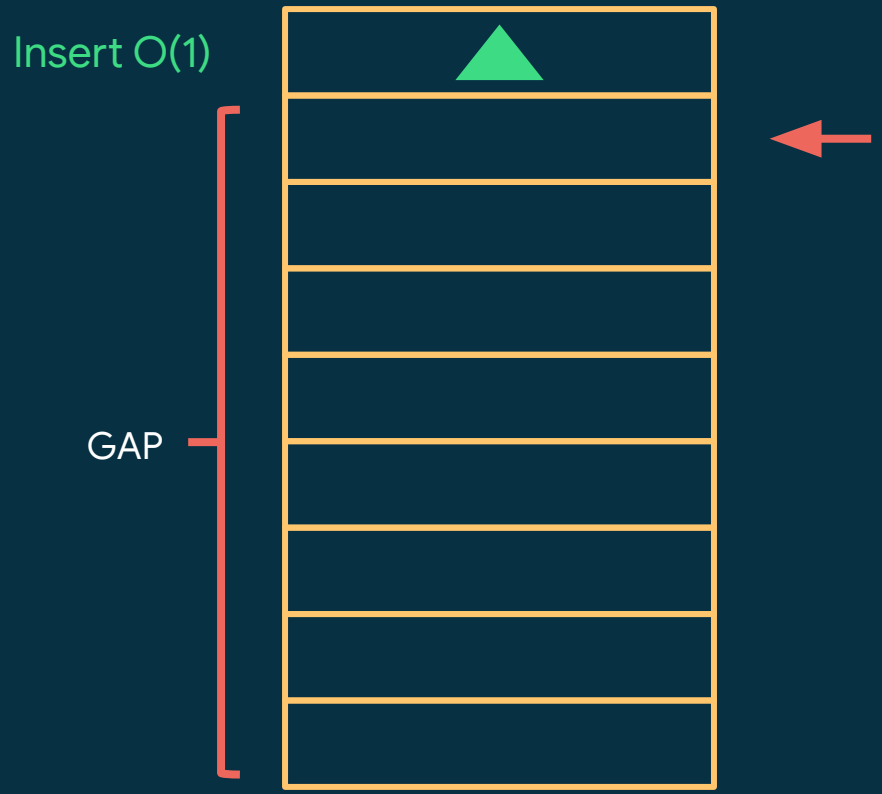
***Далее идут детали имплементации и они могут (и будут!) меняться**

Хранение виртуальной репрезентации

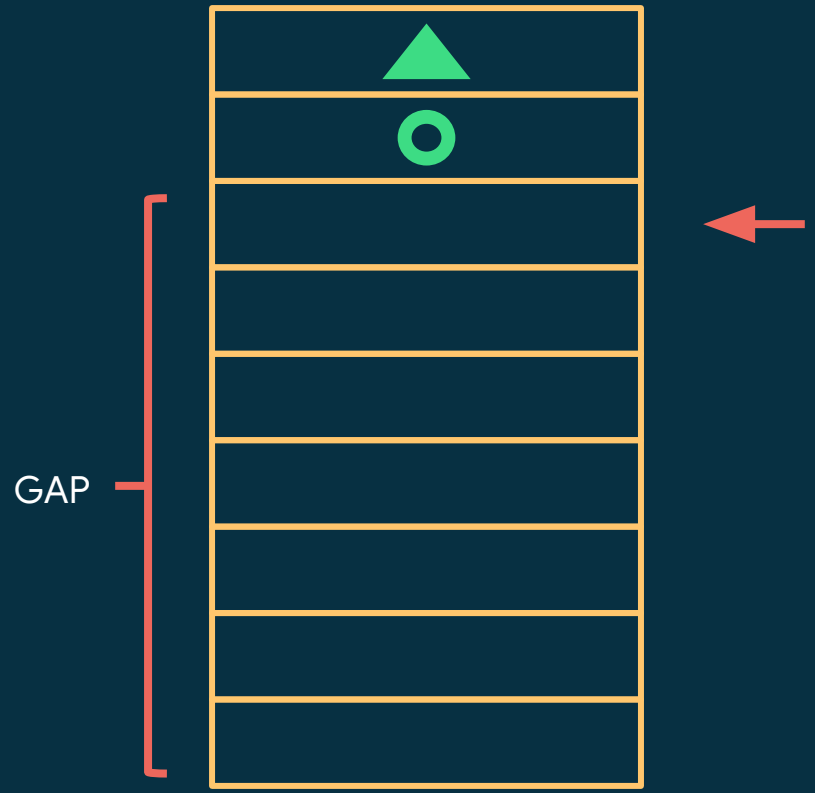
Gap buffer



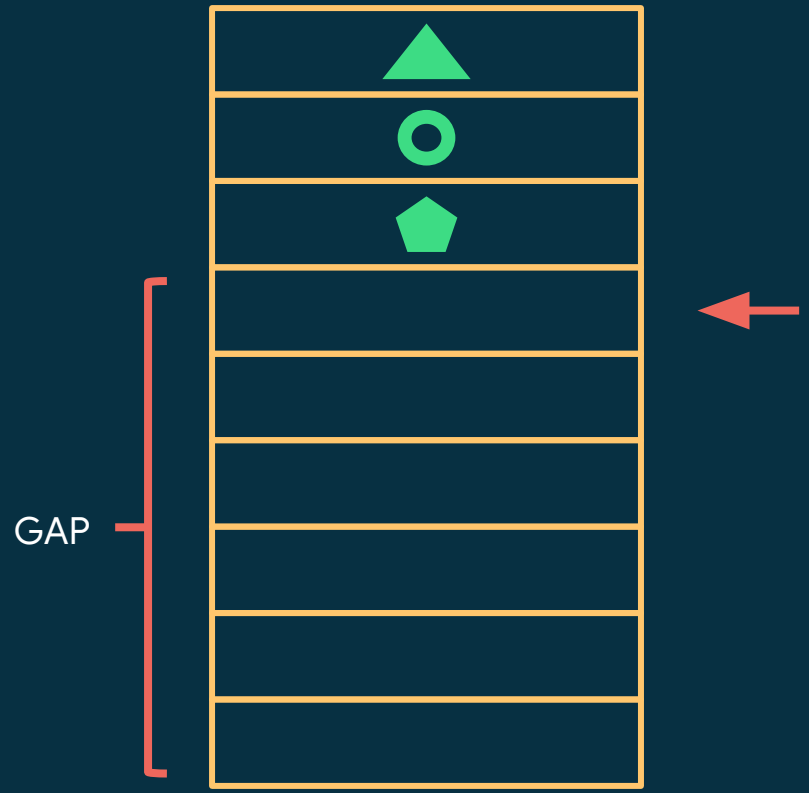
Gap buffer



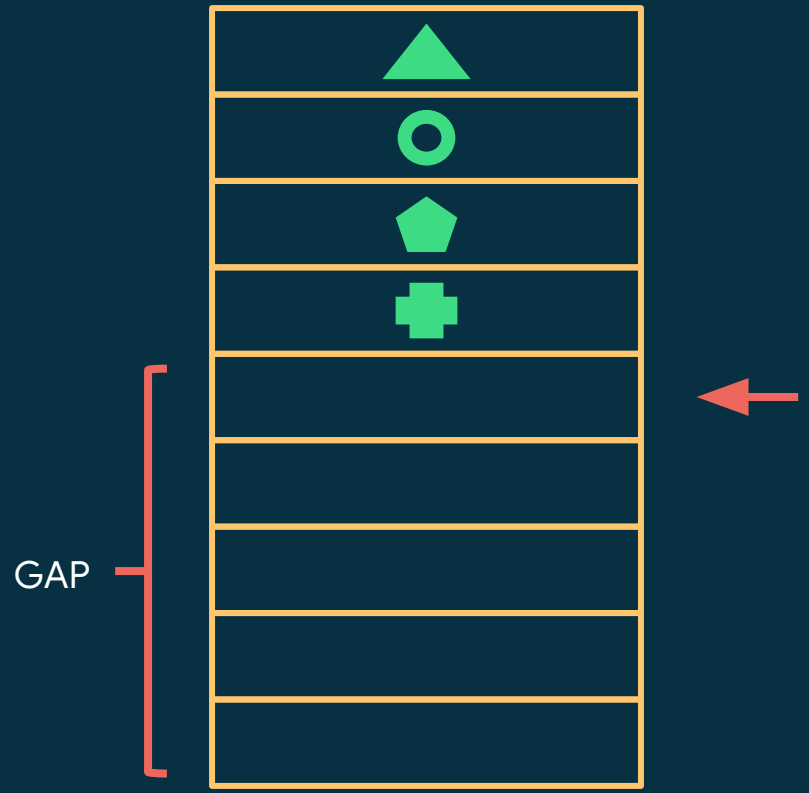
Gap buffer



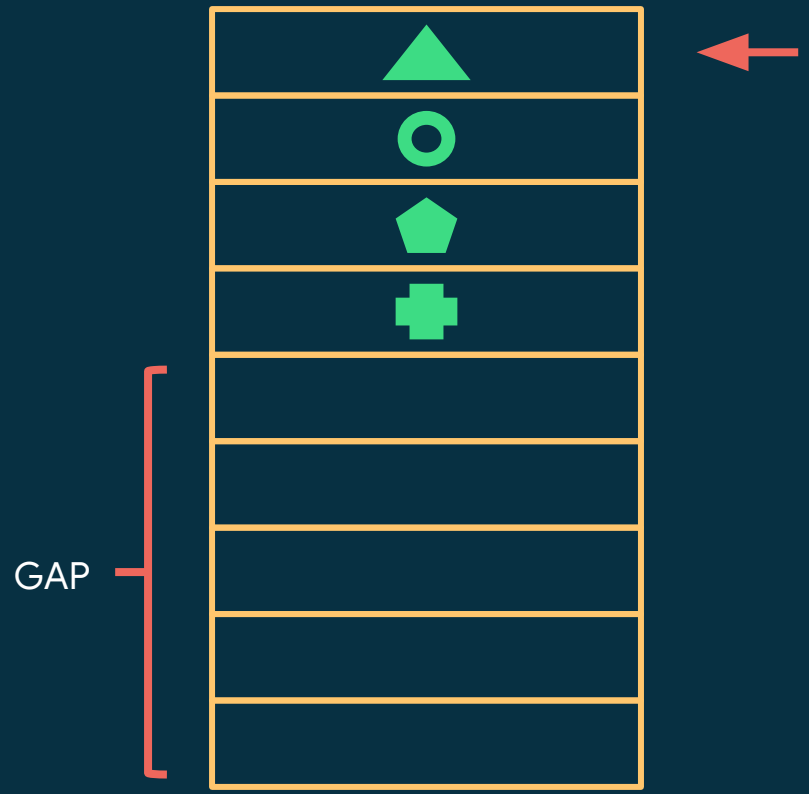
Gap buffer



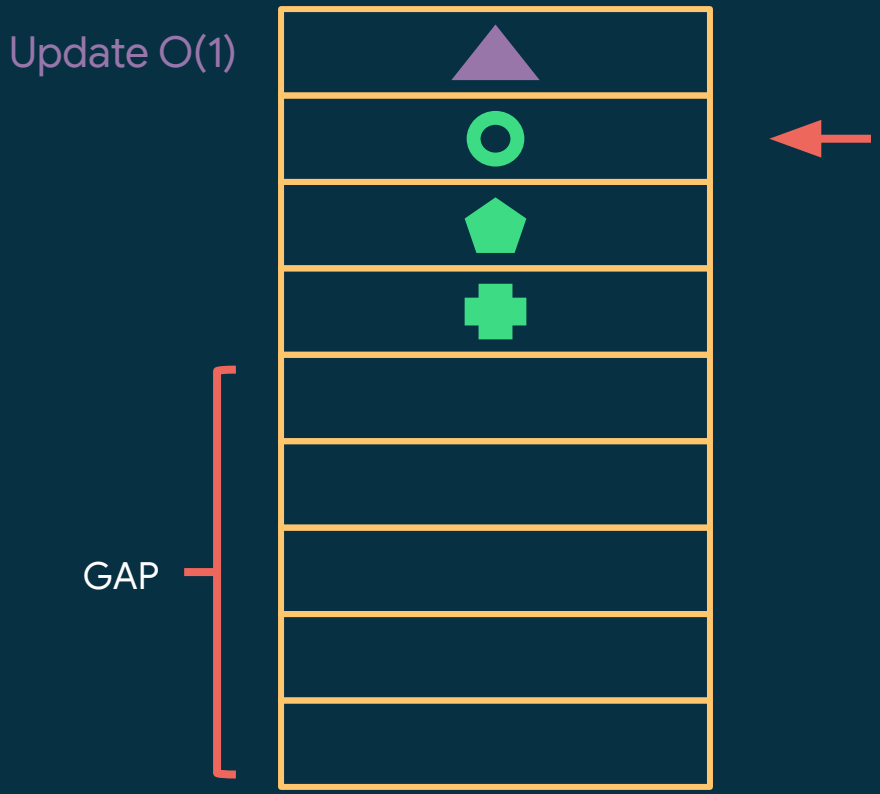
Gap buffer



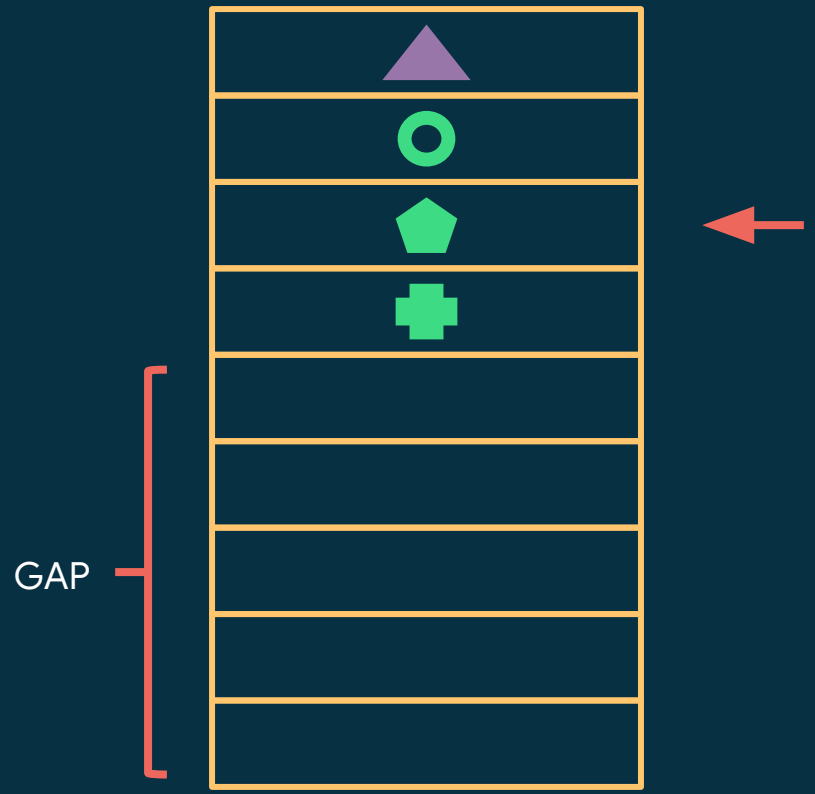
Gap buffer invalidation



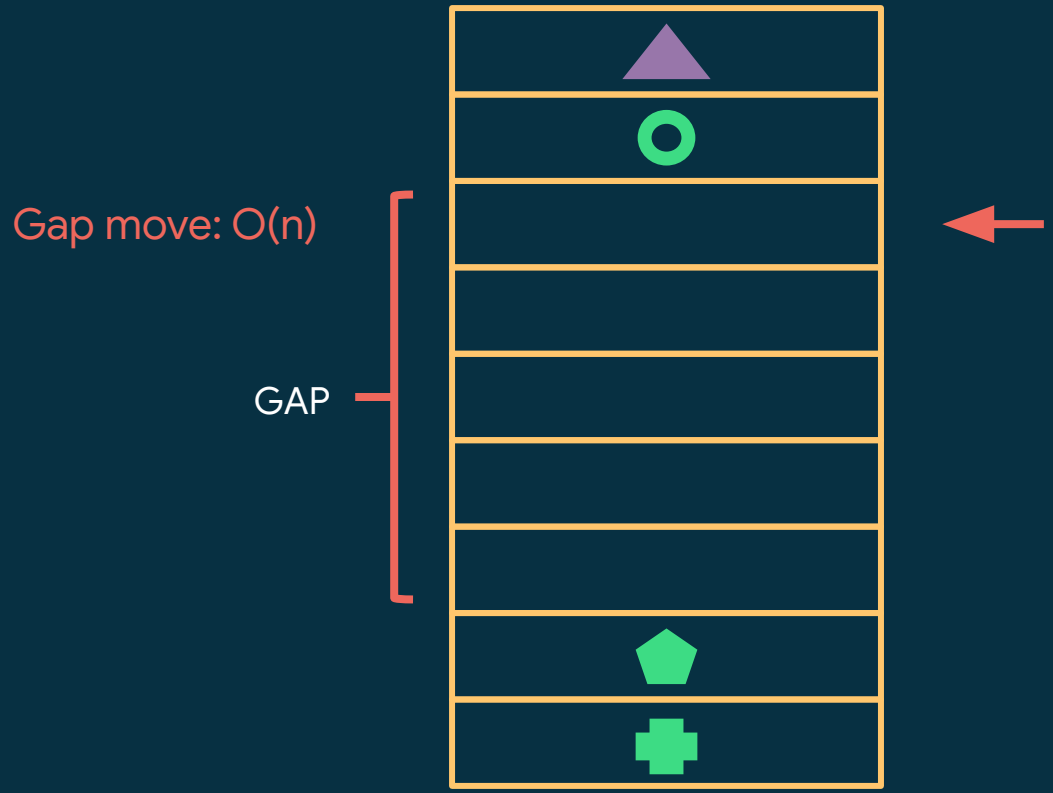
Gap buffer invalidation



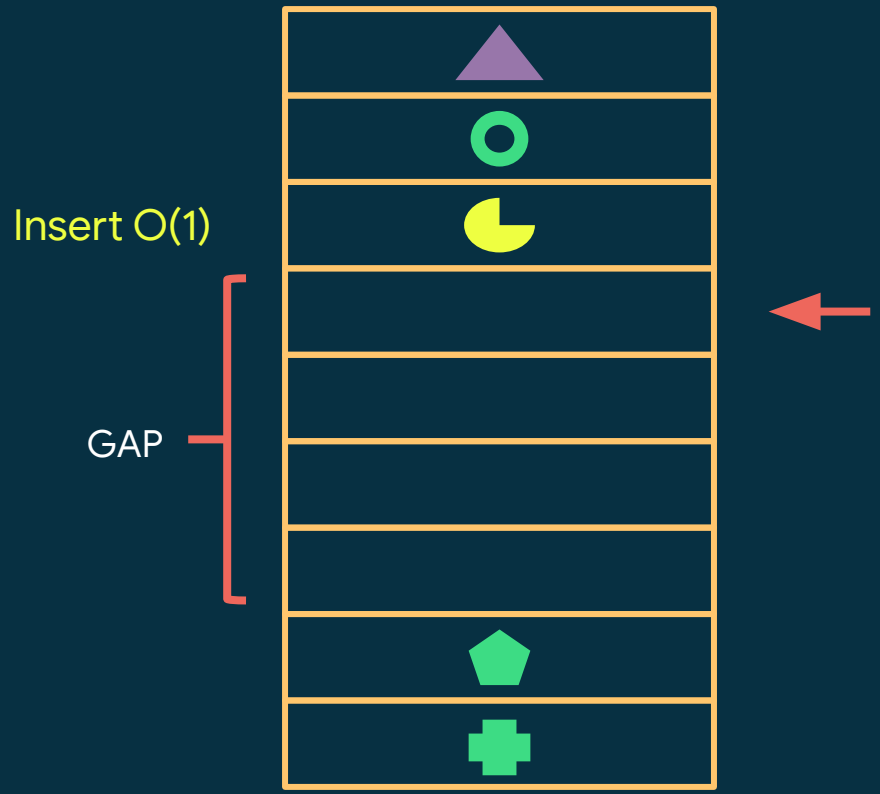
Gap buffer invalidation



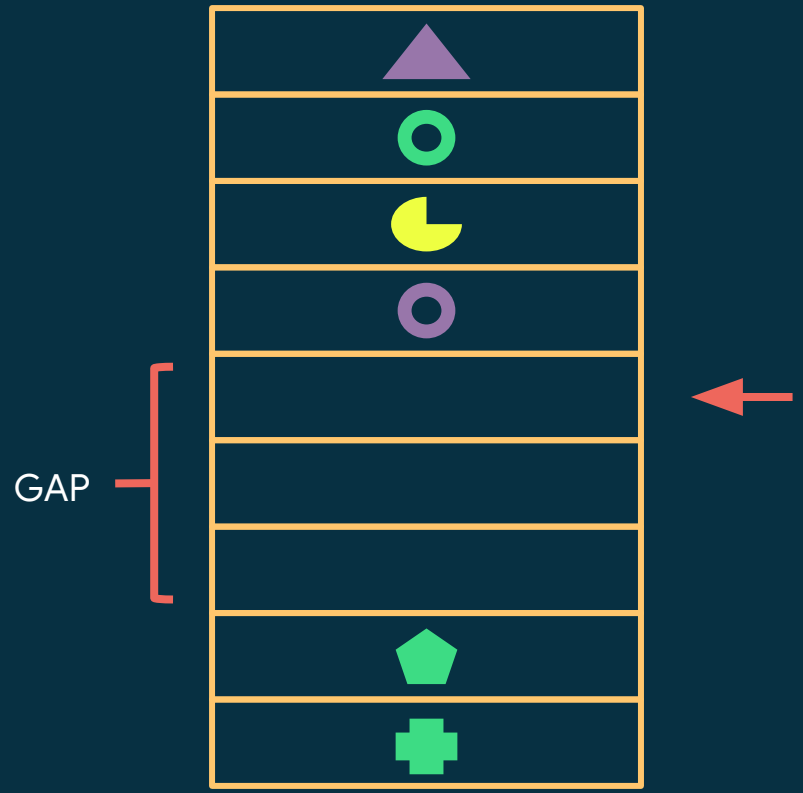
Gap buffer invalidation



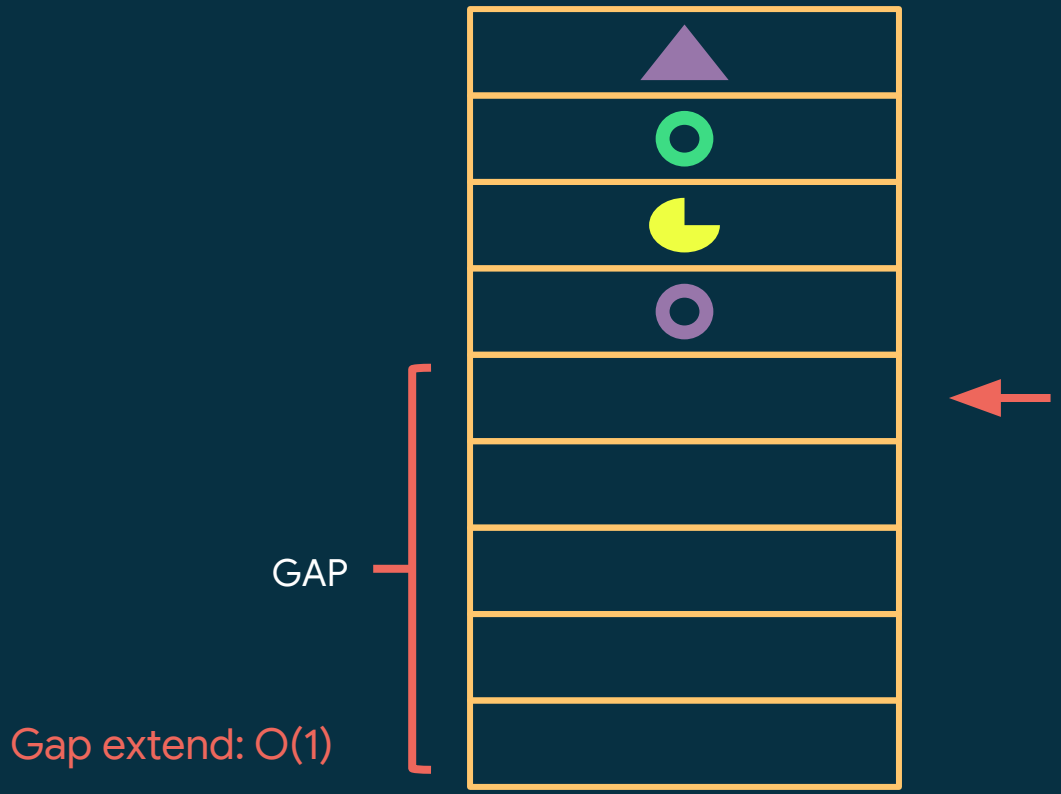
Gap buffer invalidation

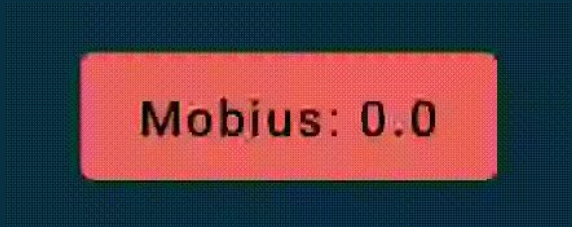


Gap buffer invalidation



Gap buffer invalidation




Mobius: 0.0

```
@Composable
fun Counter(title: String) {
    var count by state { 0f }
    Button(
        text = "$title: $count",
        onClick = { count++ }
    )
}
```

```
@Composable
fun Counter(title: String) {
    var count by state { 0f }
    Button(
        text = "$title: $count",
        onClick = { count++ }
    )
}
```

```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```

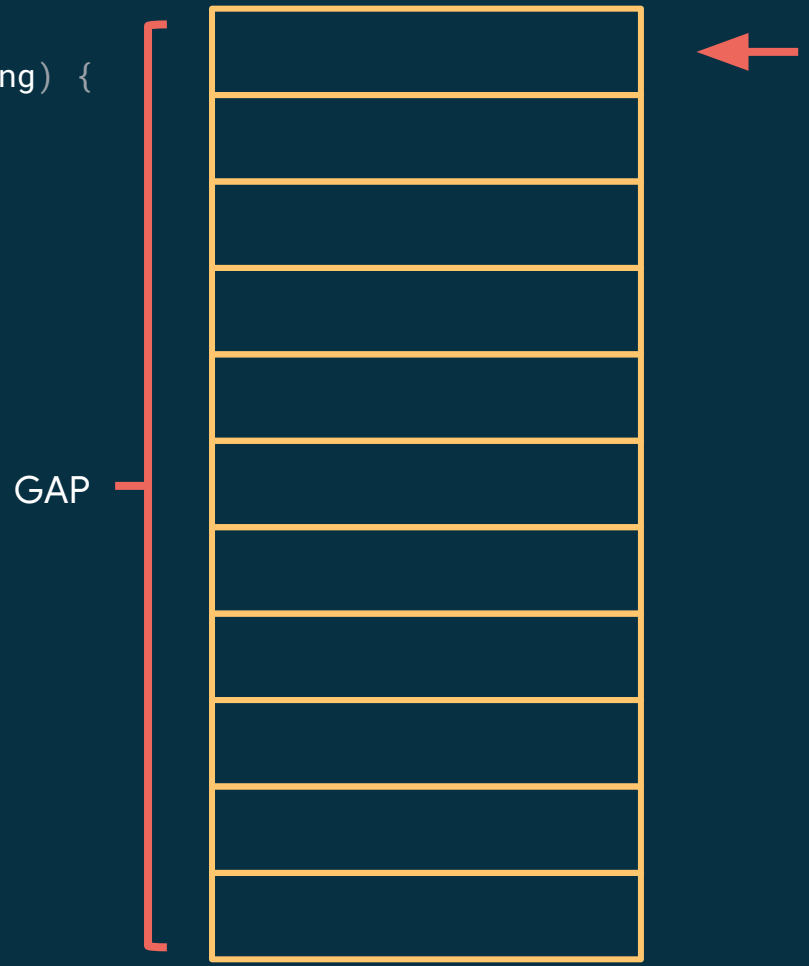
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```

```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```

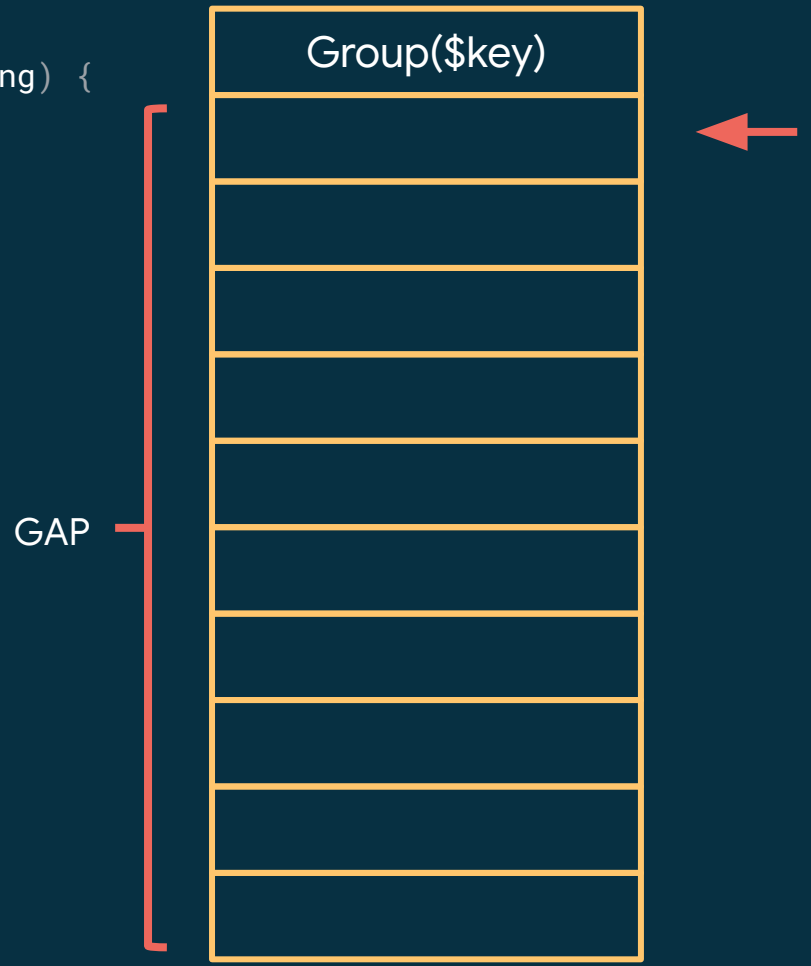
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```

```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```

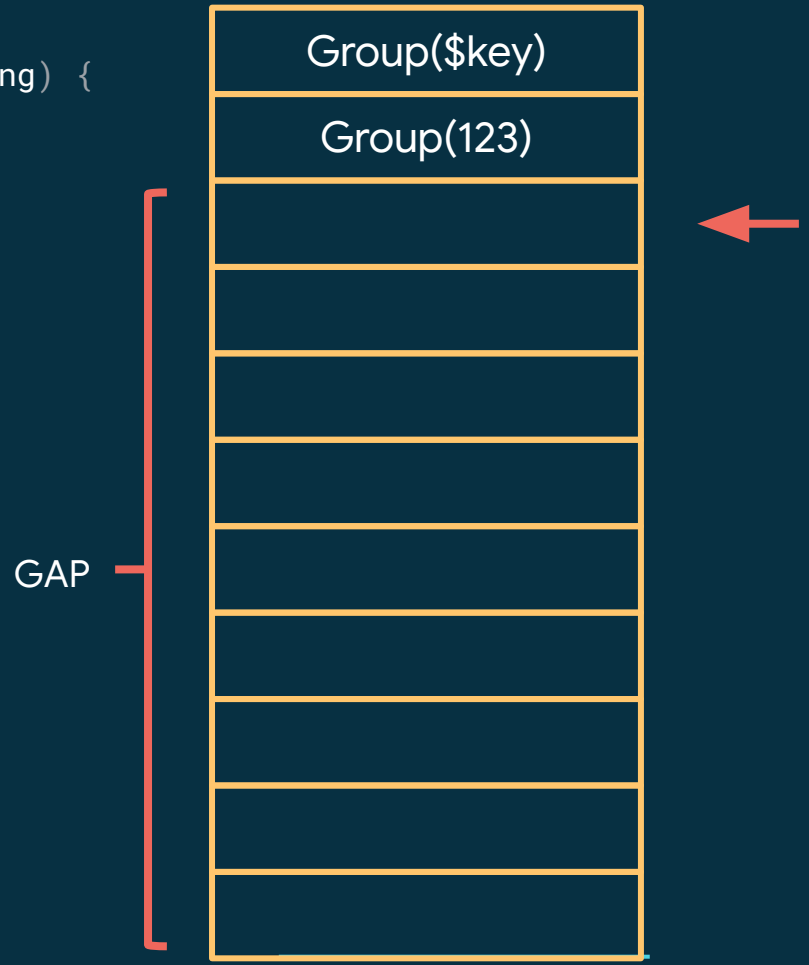
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



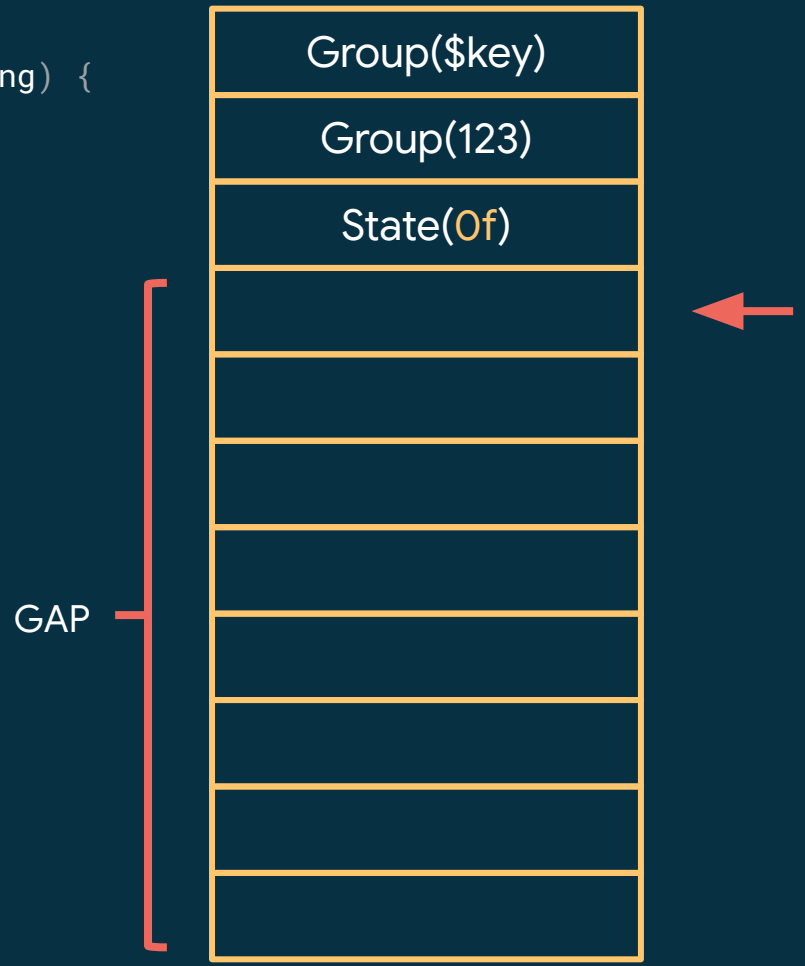
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



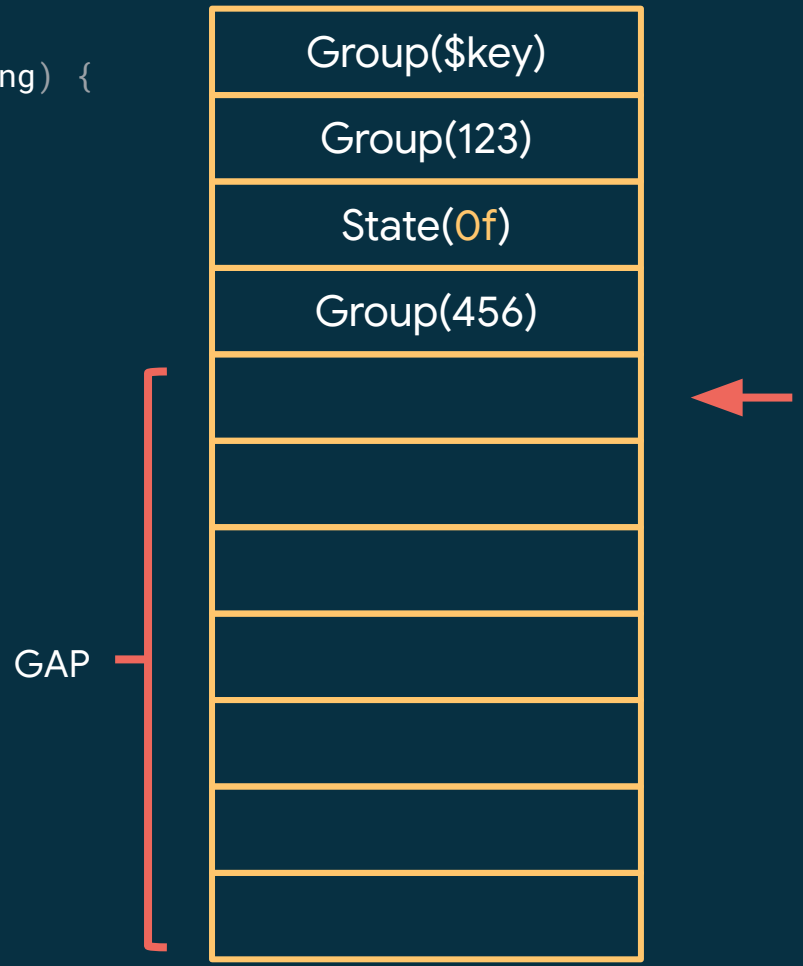
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



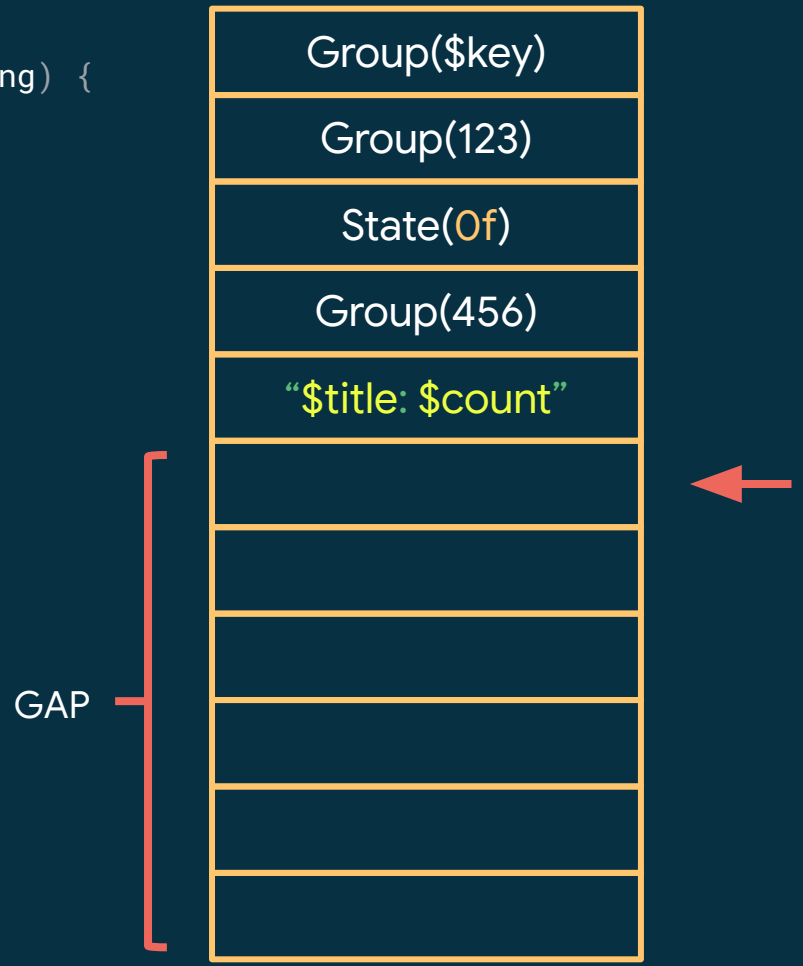
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



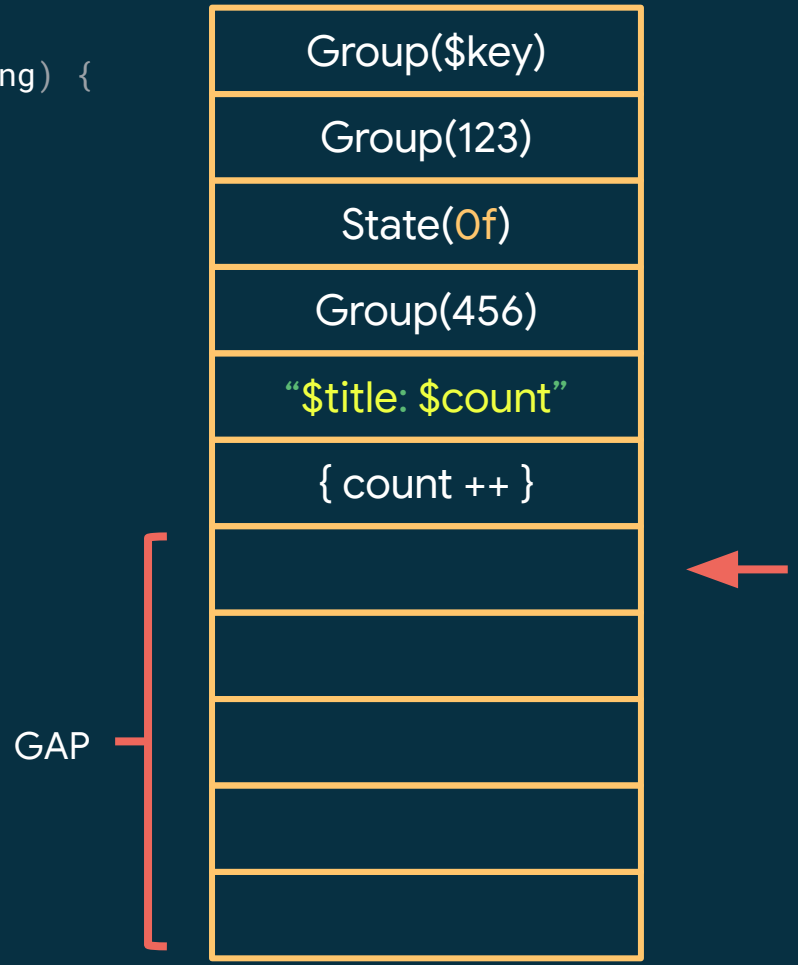
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



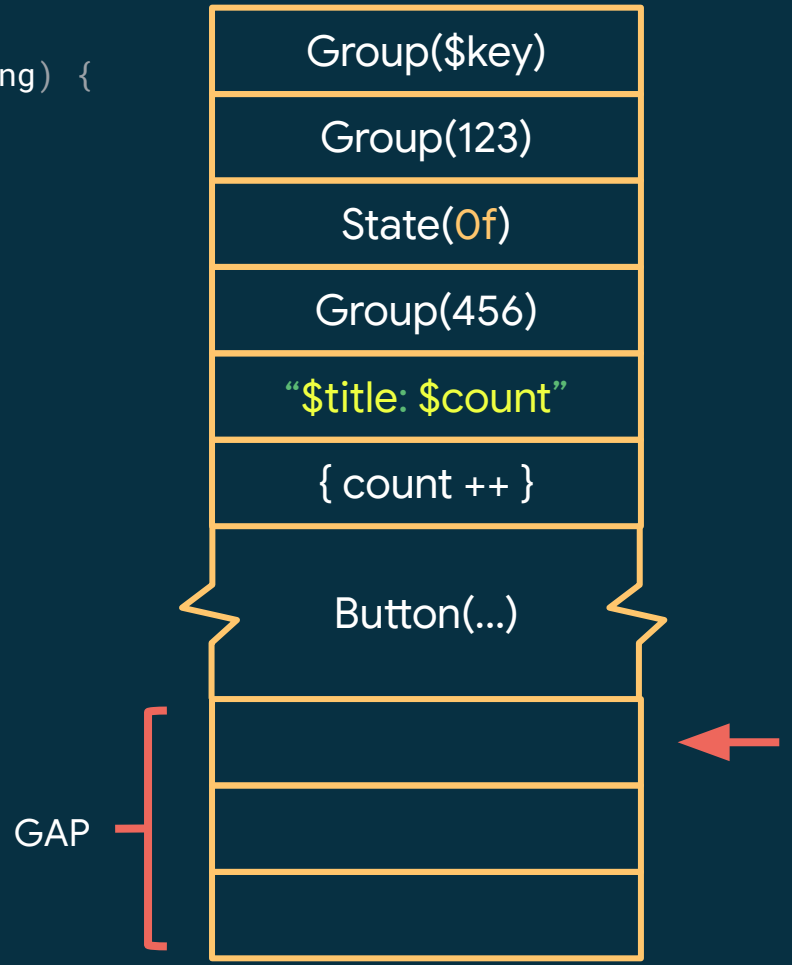
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```

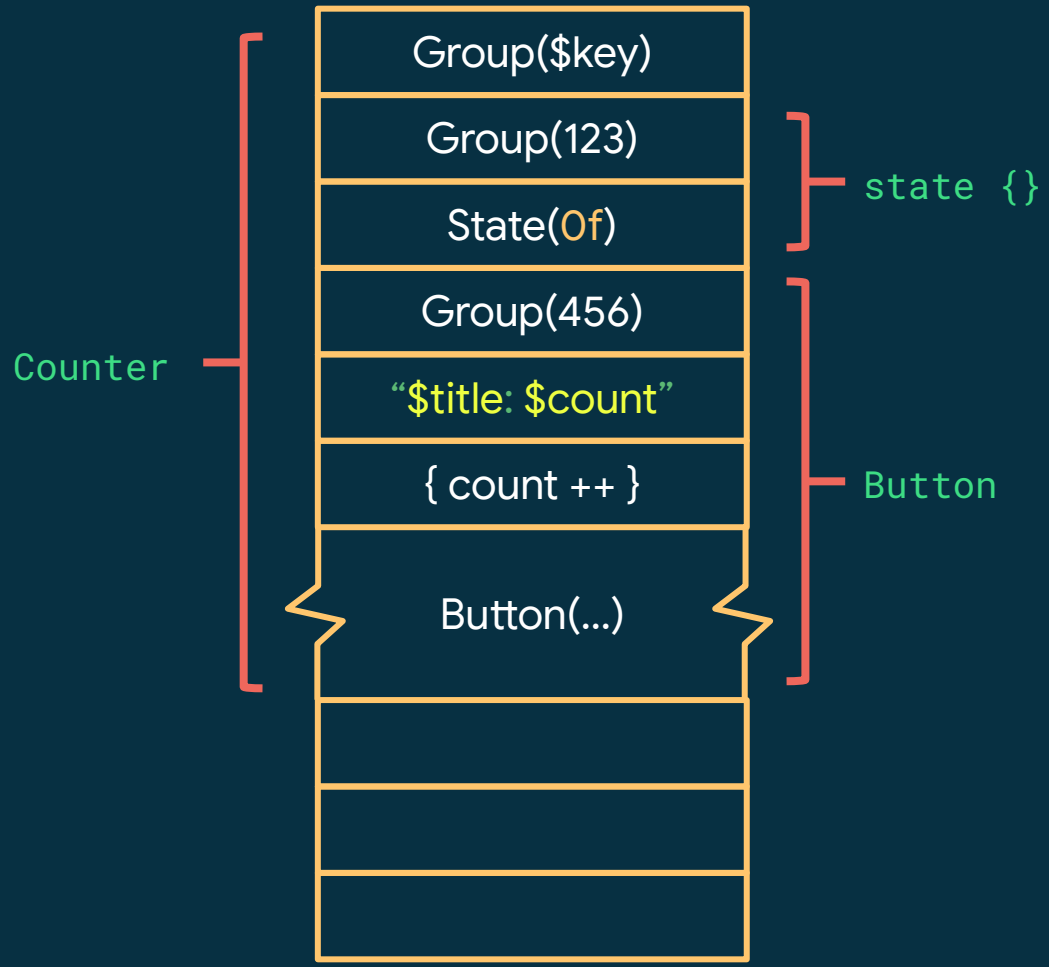


```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



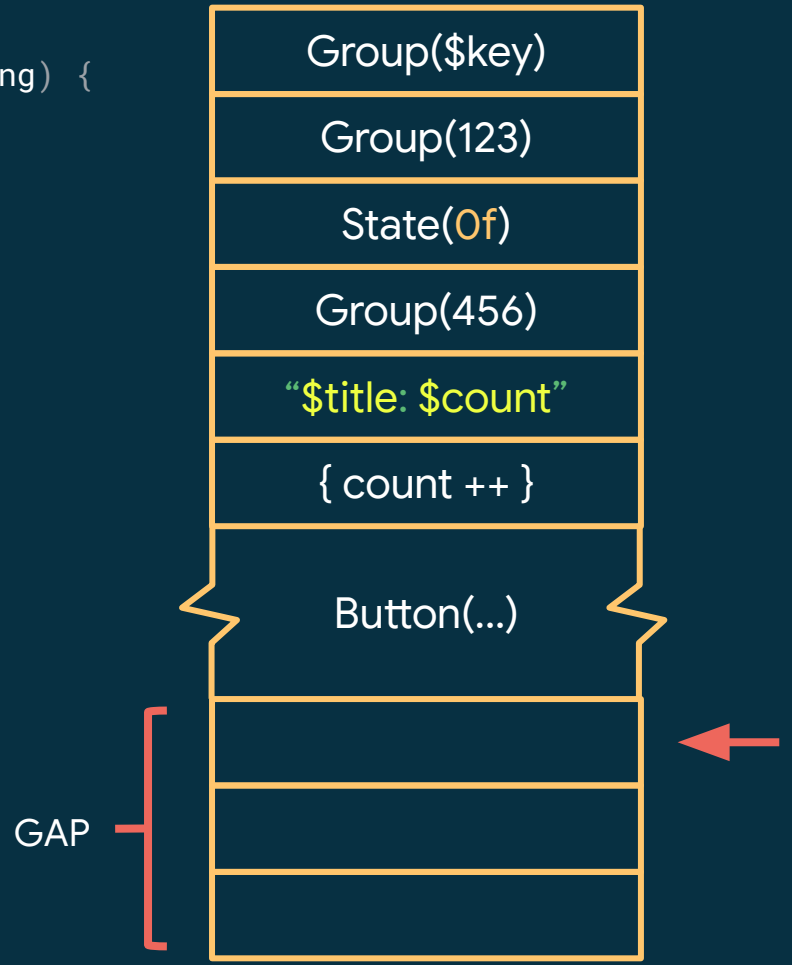
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```




Group($key)
Group(123)
State(0f)
Group(456)
“$title: $count”
{ count ++ }
Button(...)
Counter
state {}
Button

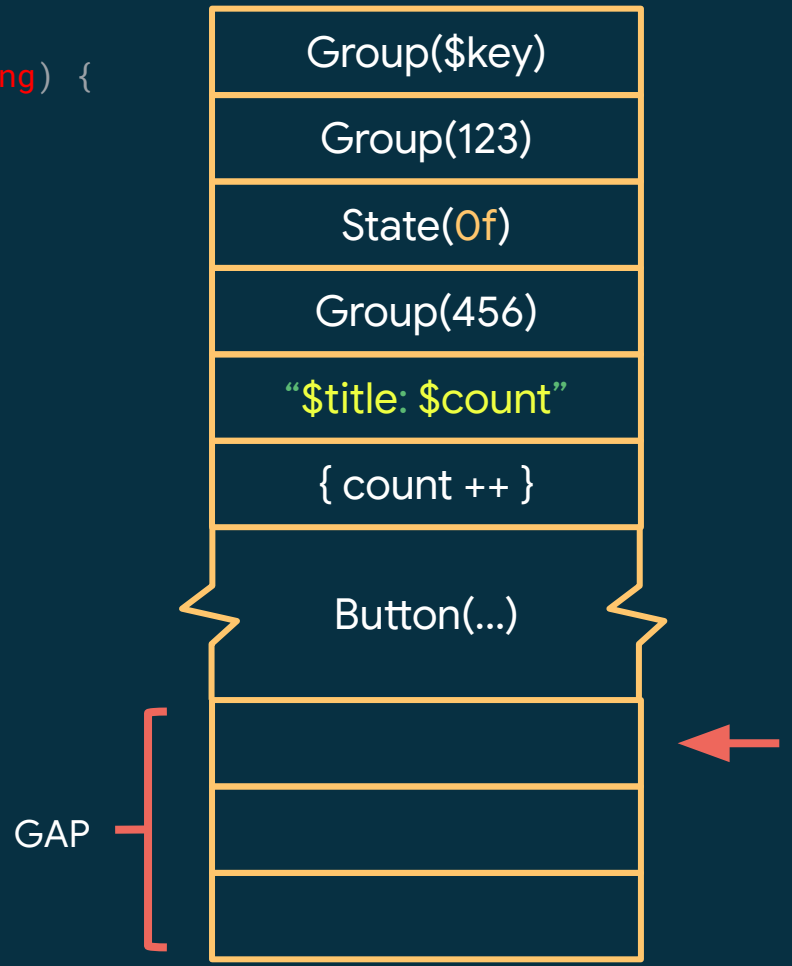
Обновление виртуальной репрезентации

```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```

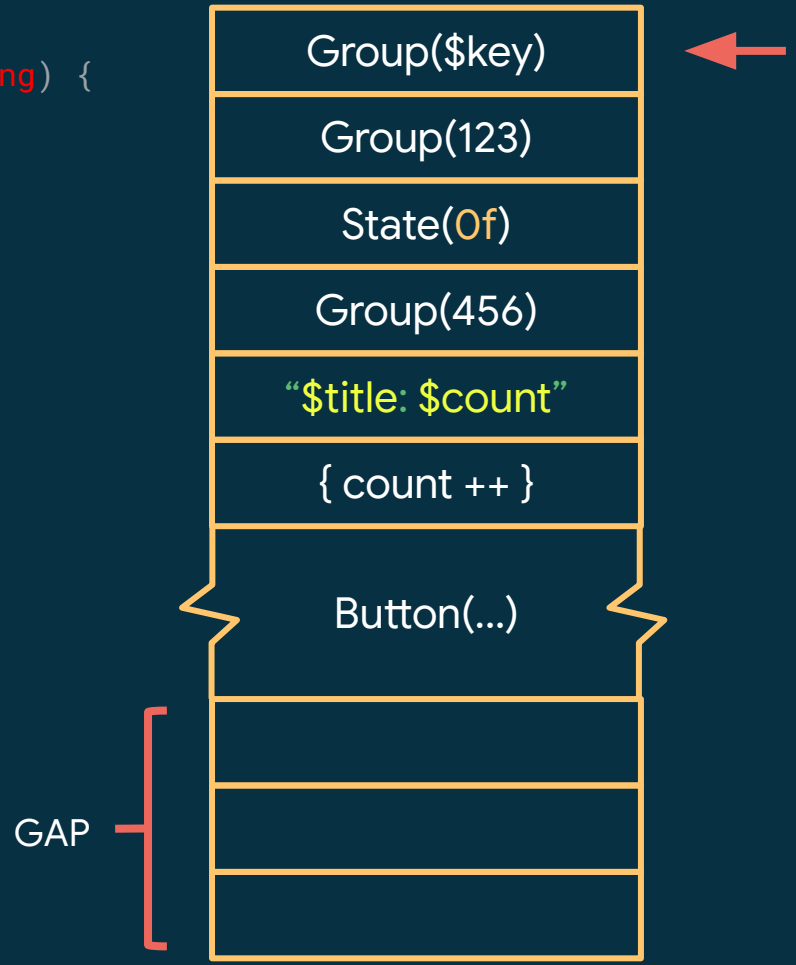


New value came

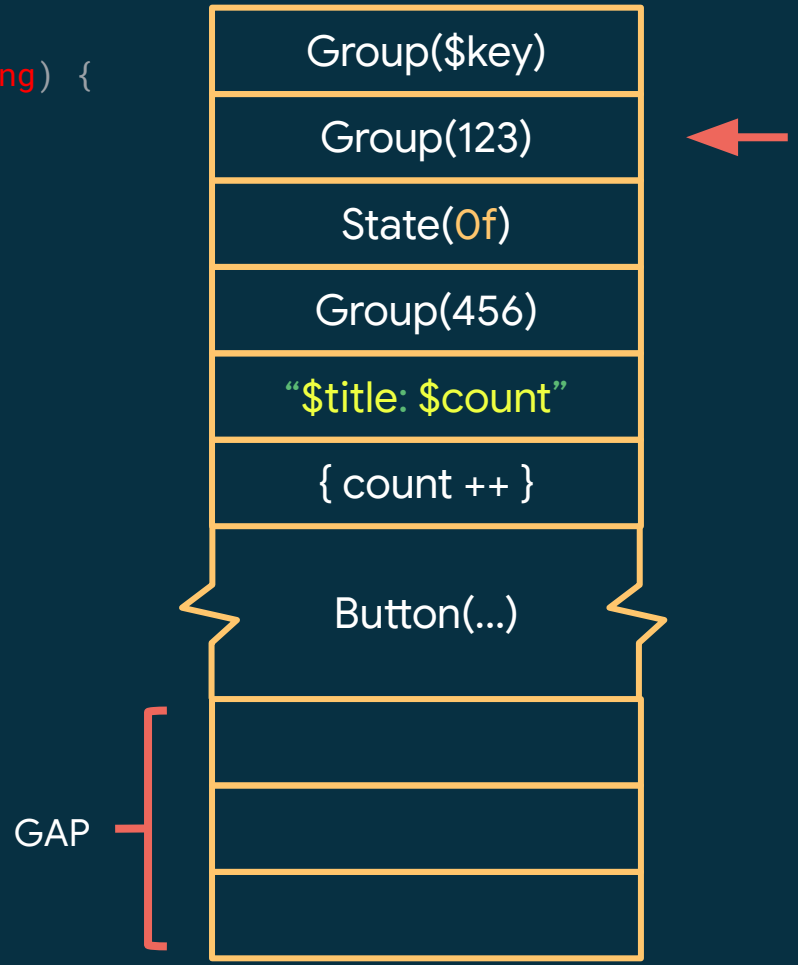
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



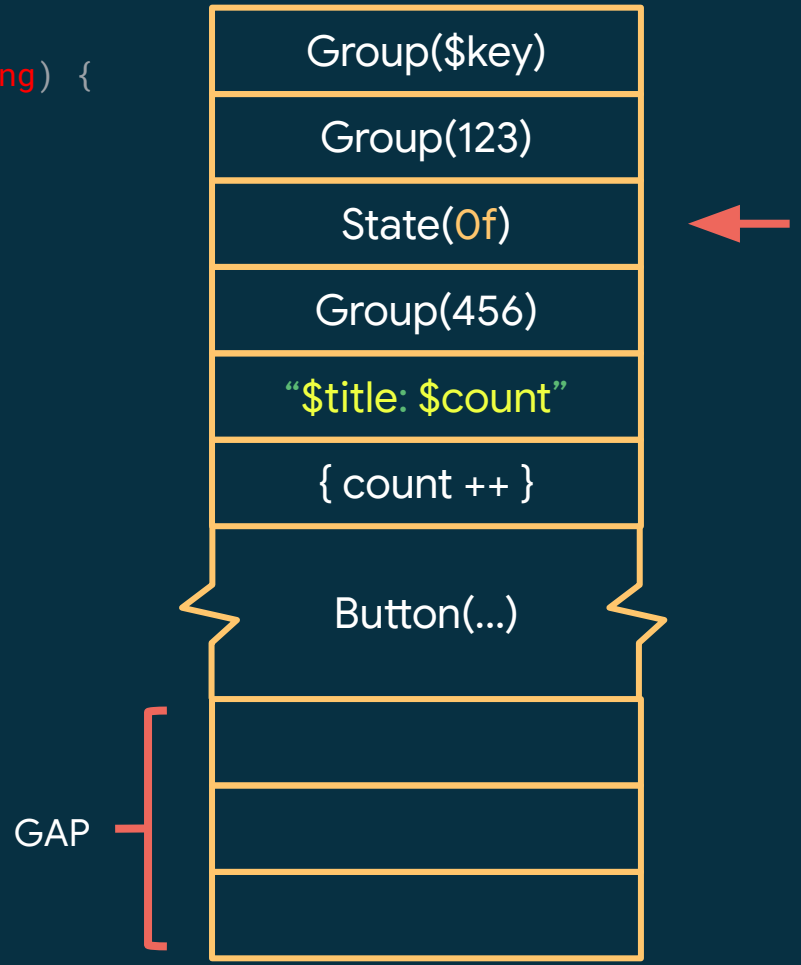
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



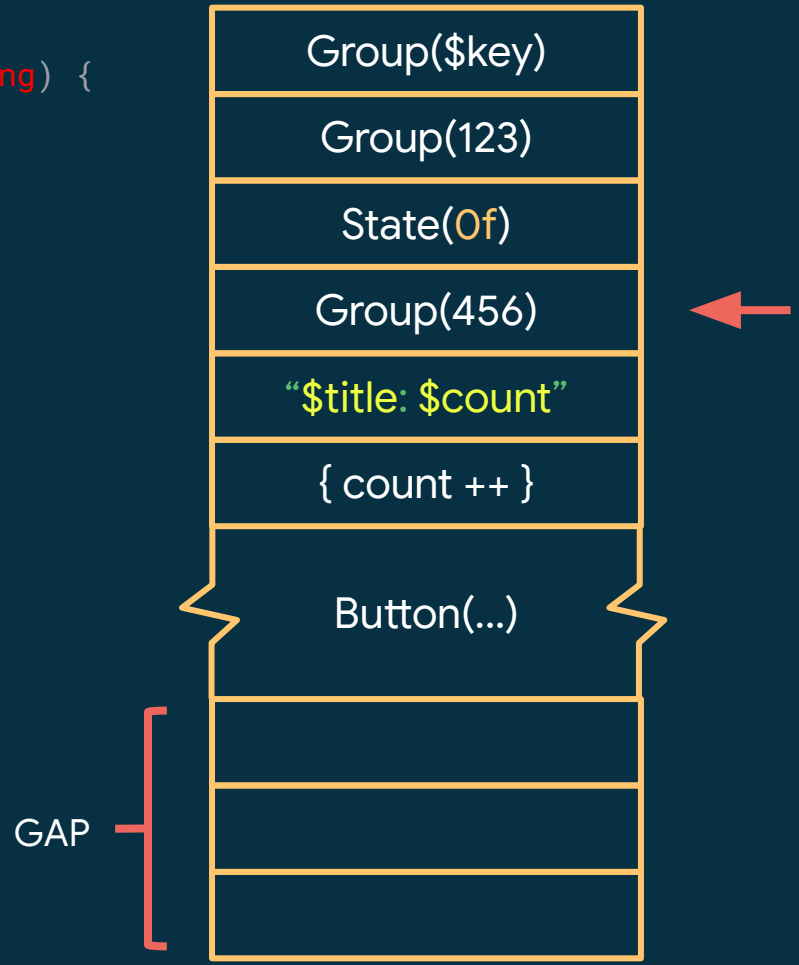
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



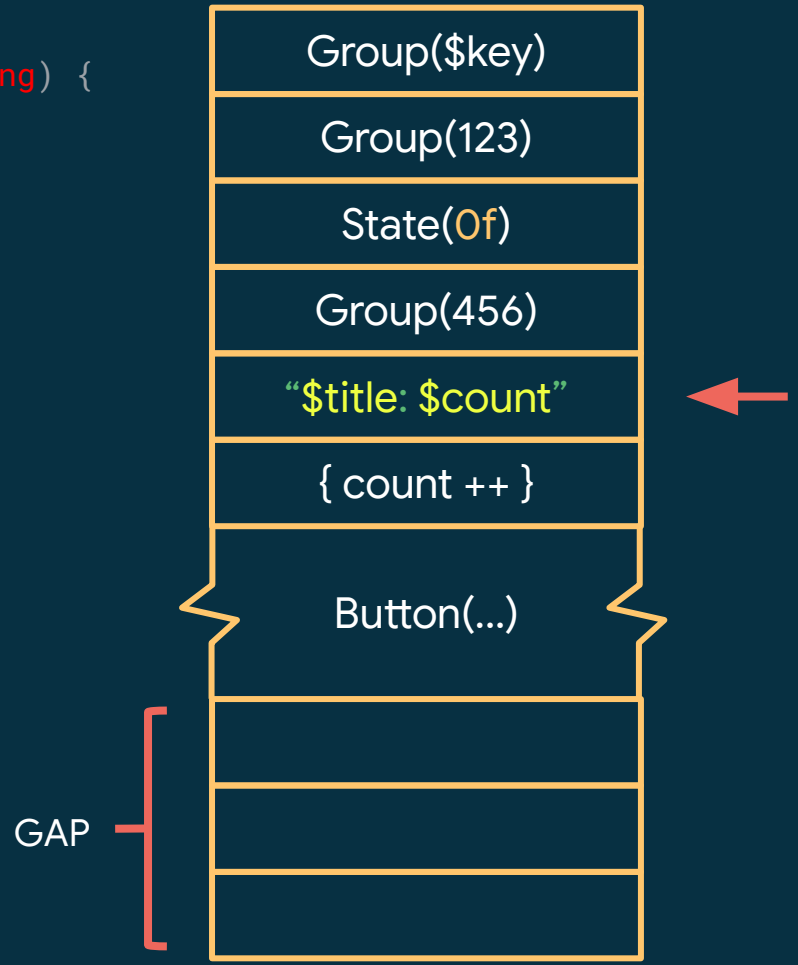
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



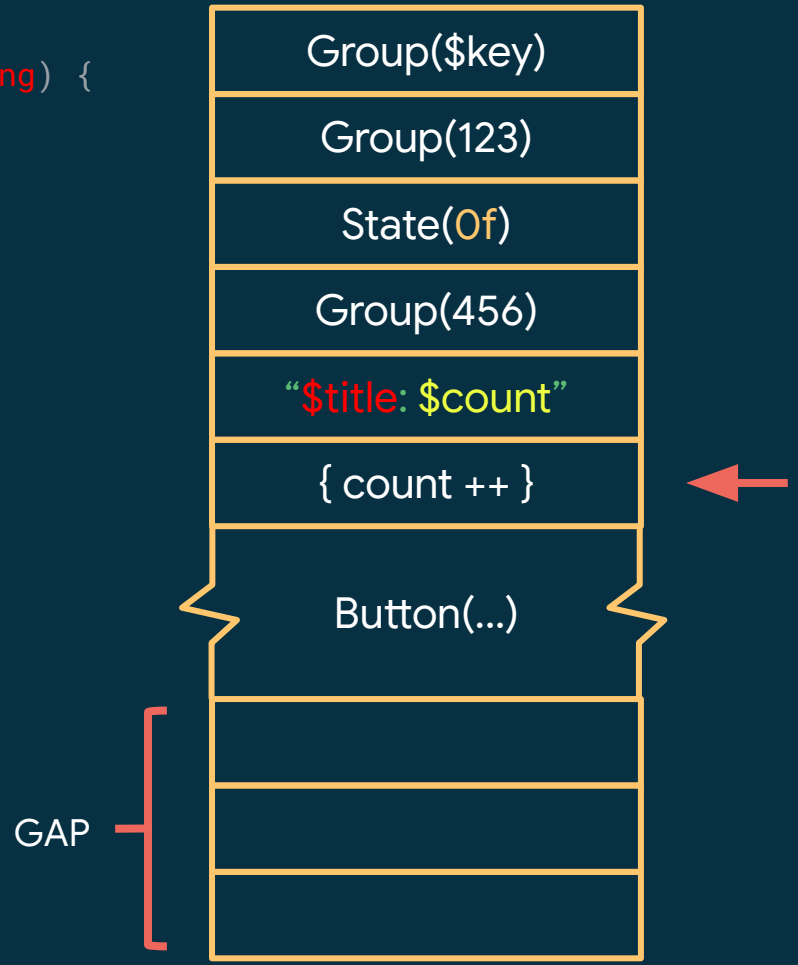
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



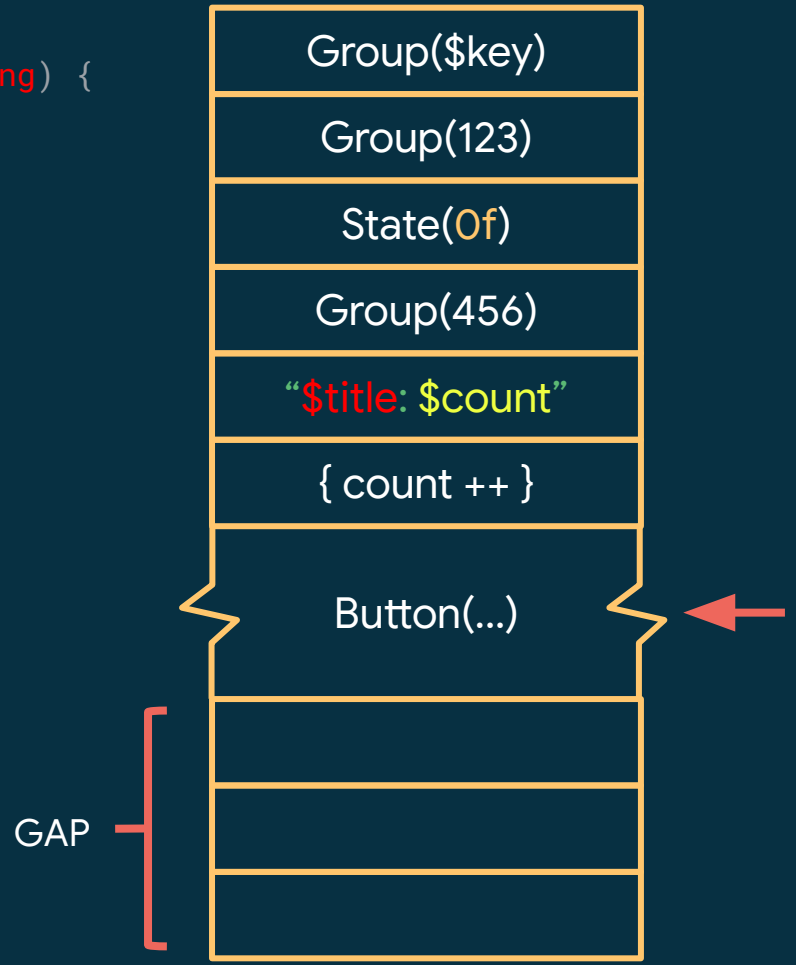
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



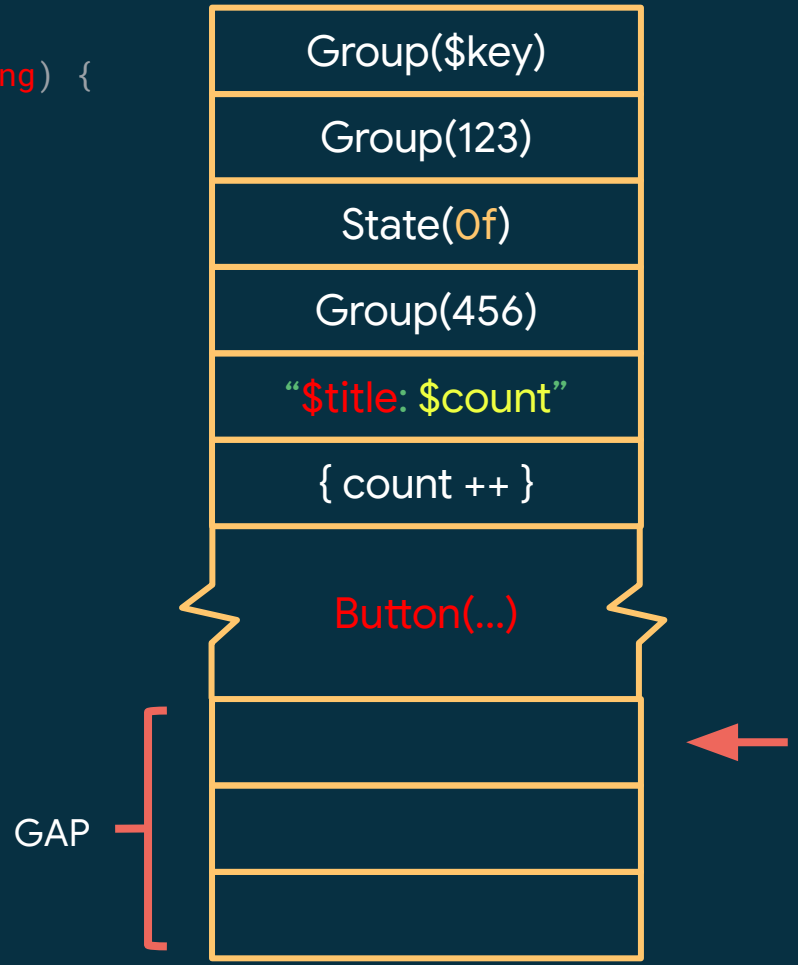
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



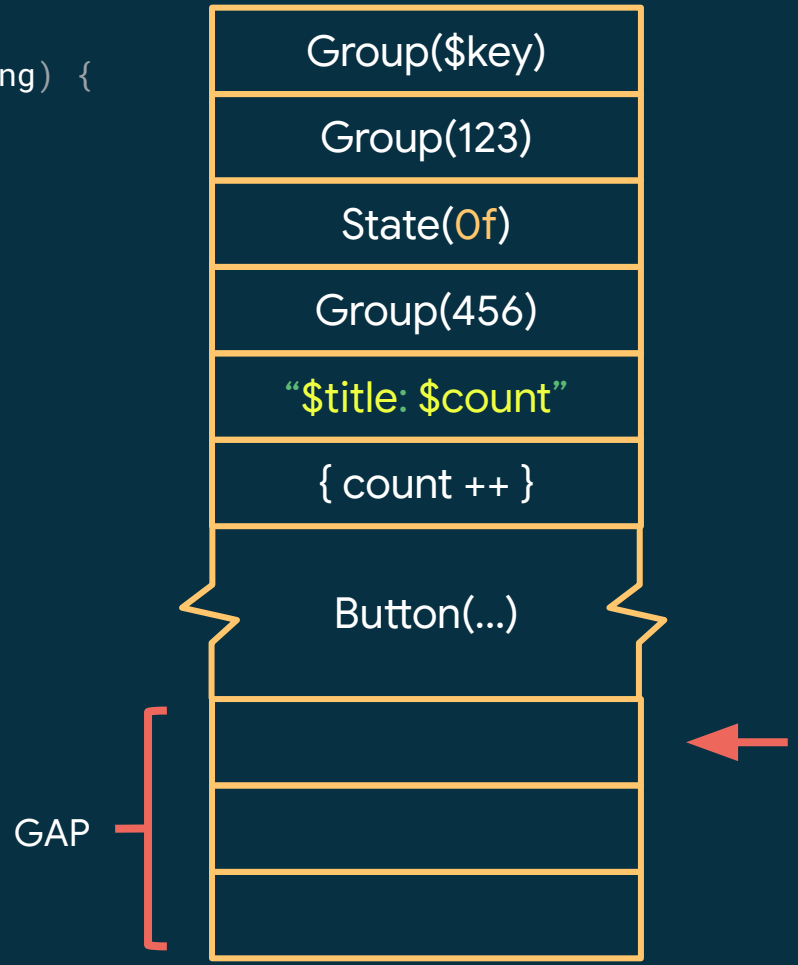
```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



```
fun Counter($composer:Composer,$key: Int,title: String) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```

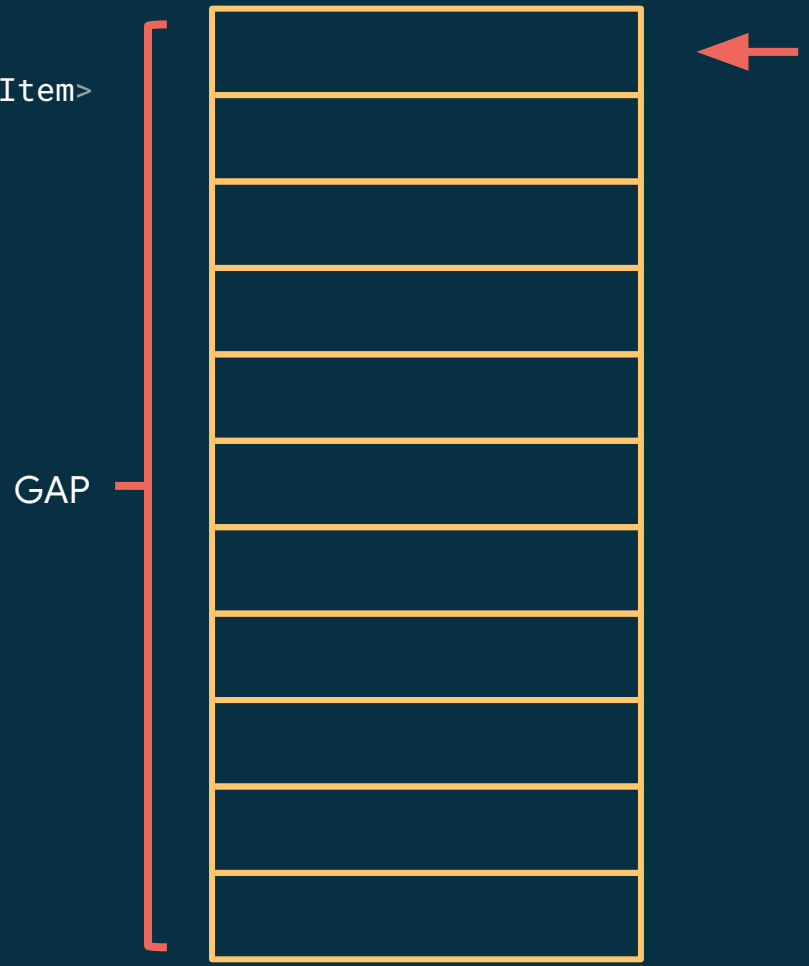


Обновление виртуальной репрезентации
Conditional logic

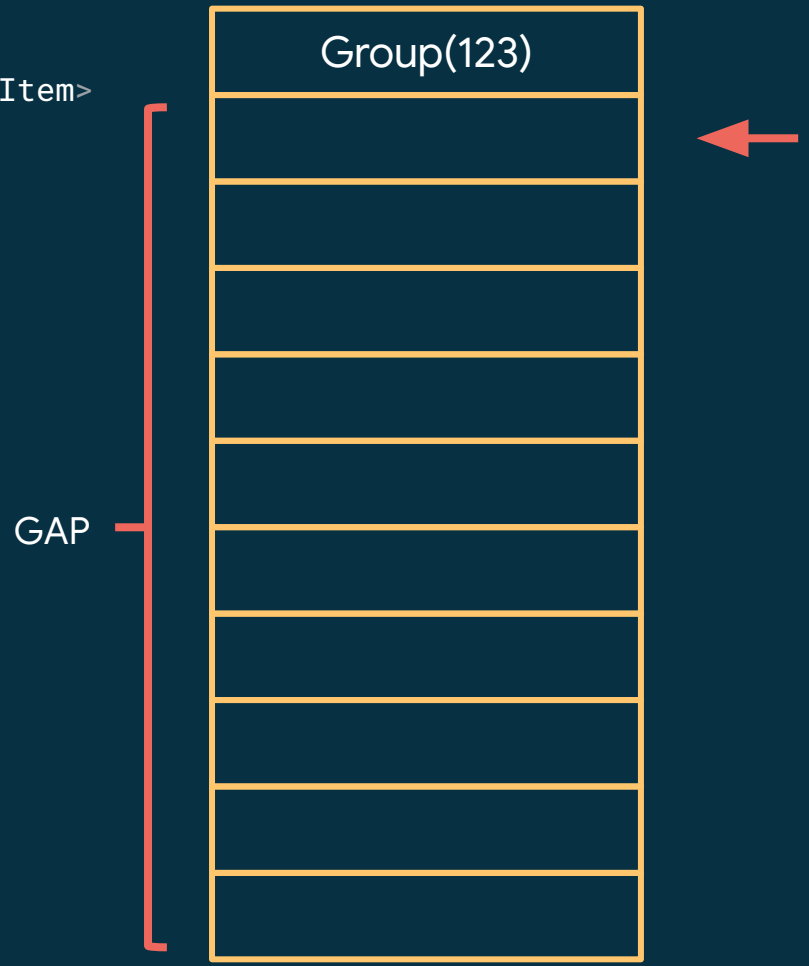
```
@Composable
fun Feeds(items: List<FeedItem>) {
    if (items.isEmpty) {
        Text("No feeds")
    } else {
        FeedList(items)
    }
}
```

```
fun Feeds($composer:Composer, $key: Int,items: List<FeedItem>) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```

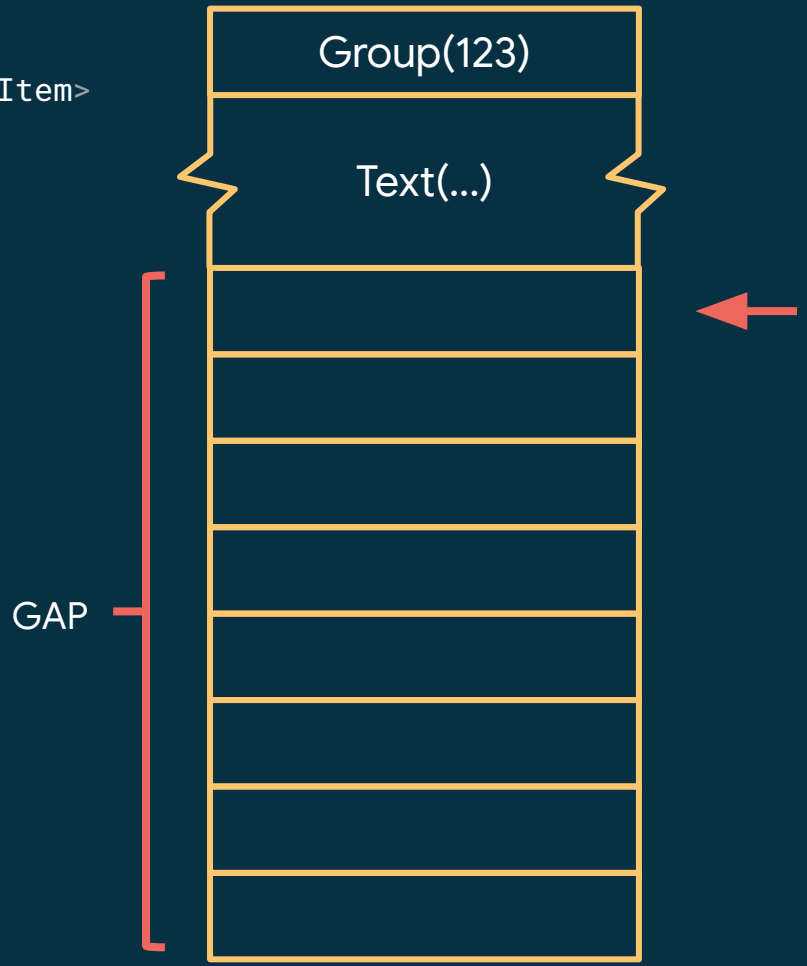
```
fun Feeds(
    $composer: Composer, $key: Int, items: List<FeedItem>
) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```



```
fun Feeds(
    $composer: Composer, $key: Int, items: List<FeedItem>
) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```

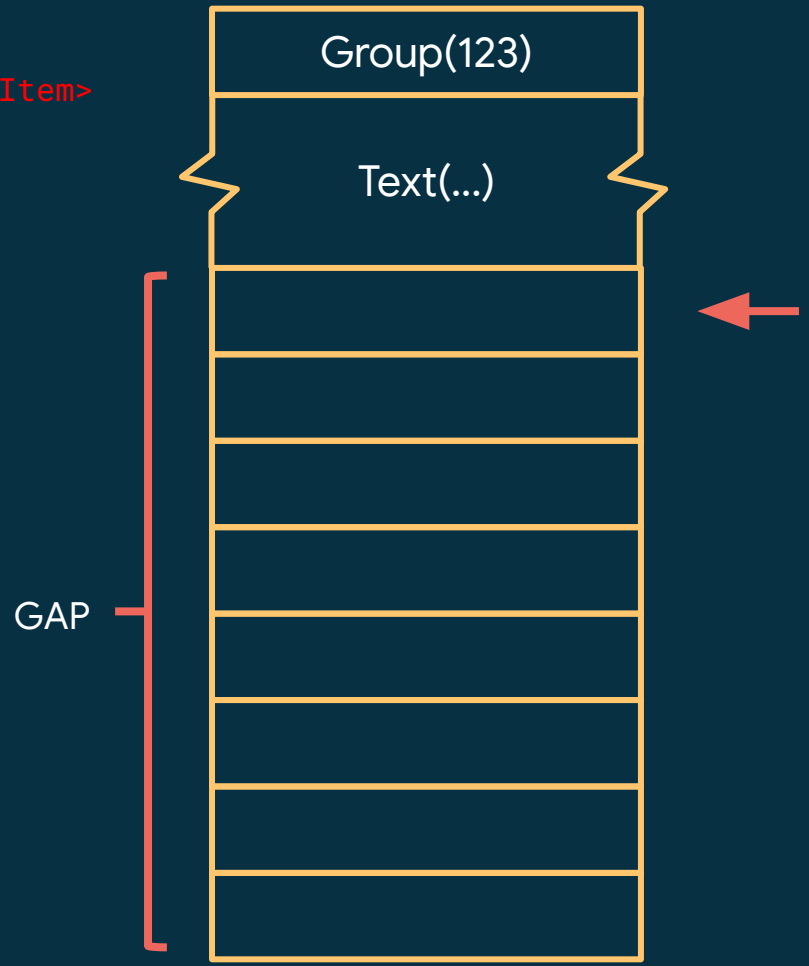


```
fun Feeds(
    $composer: Composer, $key: Int, items: List<FeedItem>
) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```

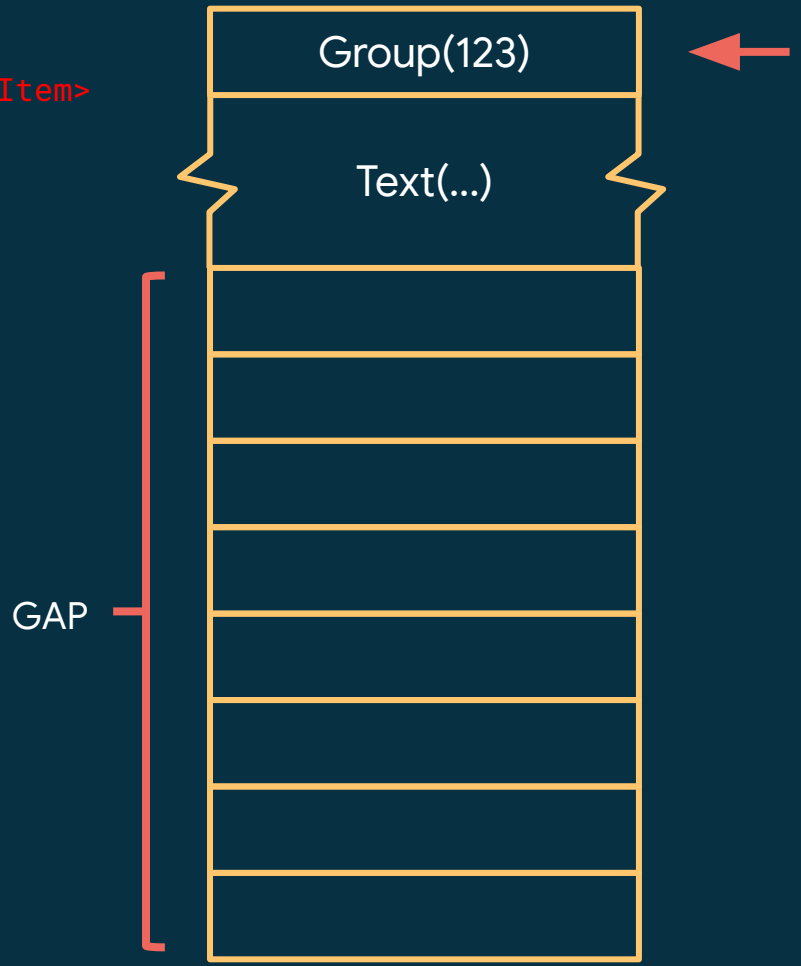


New value came

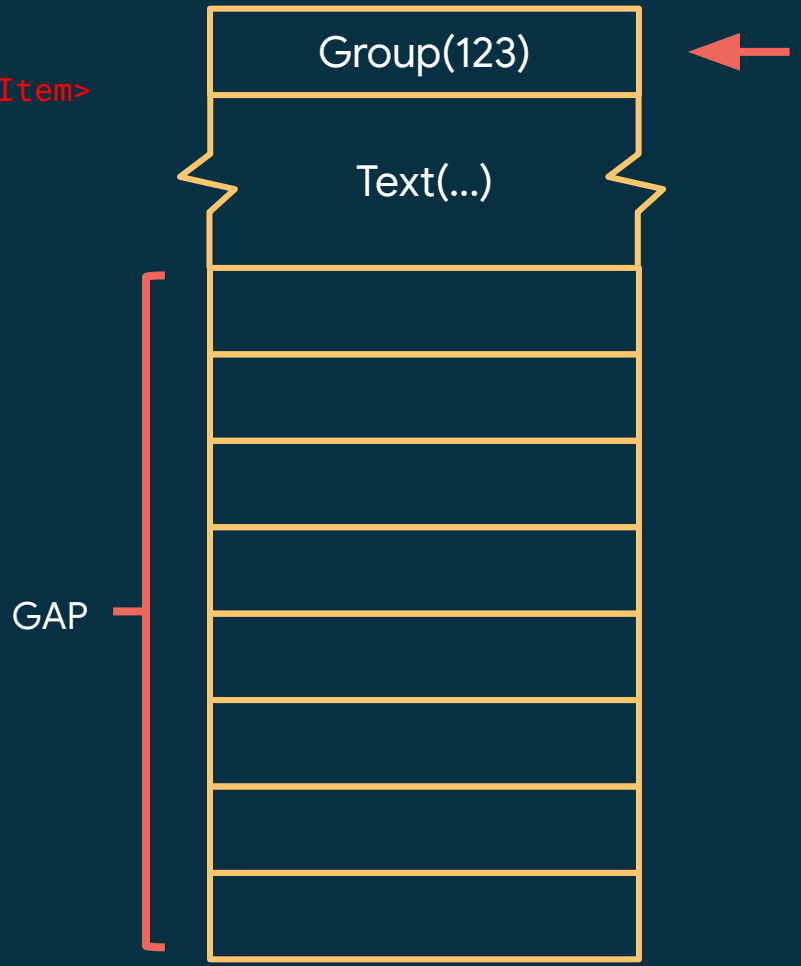
```
fun Feeds(
    $composer: Composer, $key: Int, items: List<FeedItem>
) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```



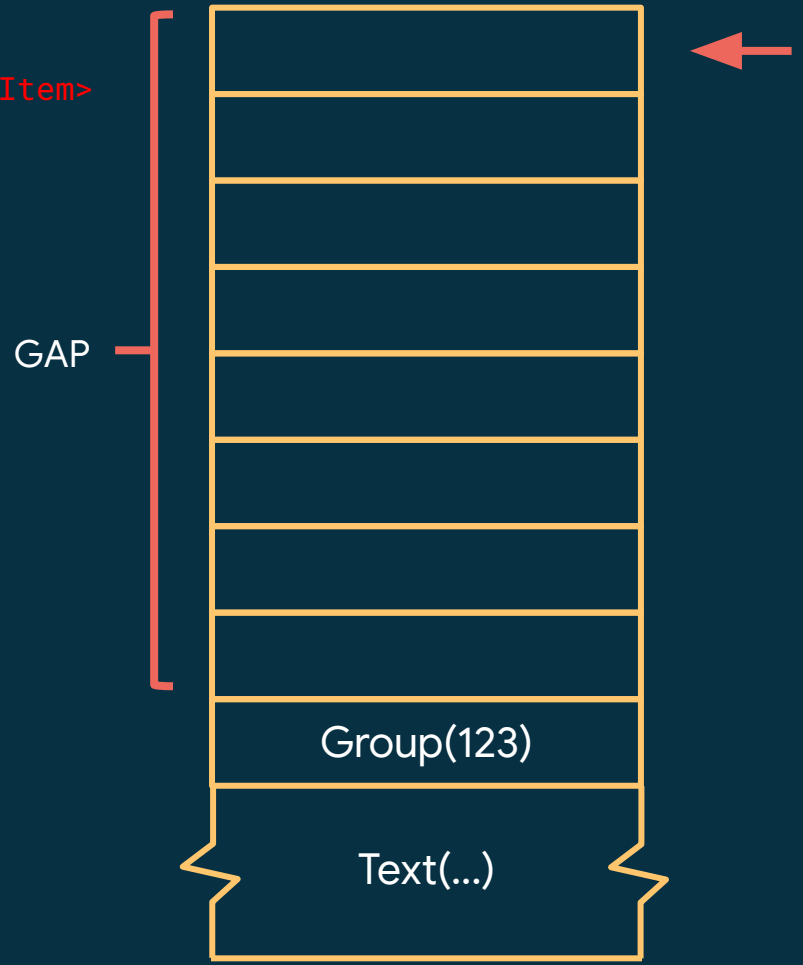
```
fun Feeds(
    $composer: Composer, $key: Int, items: List<FeedItem>
) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```



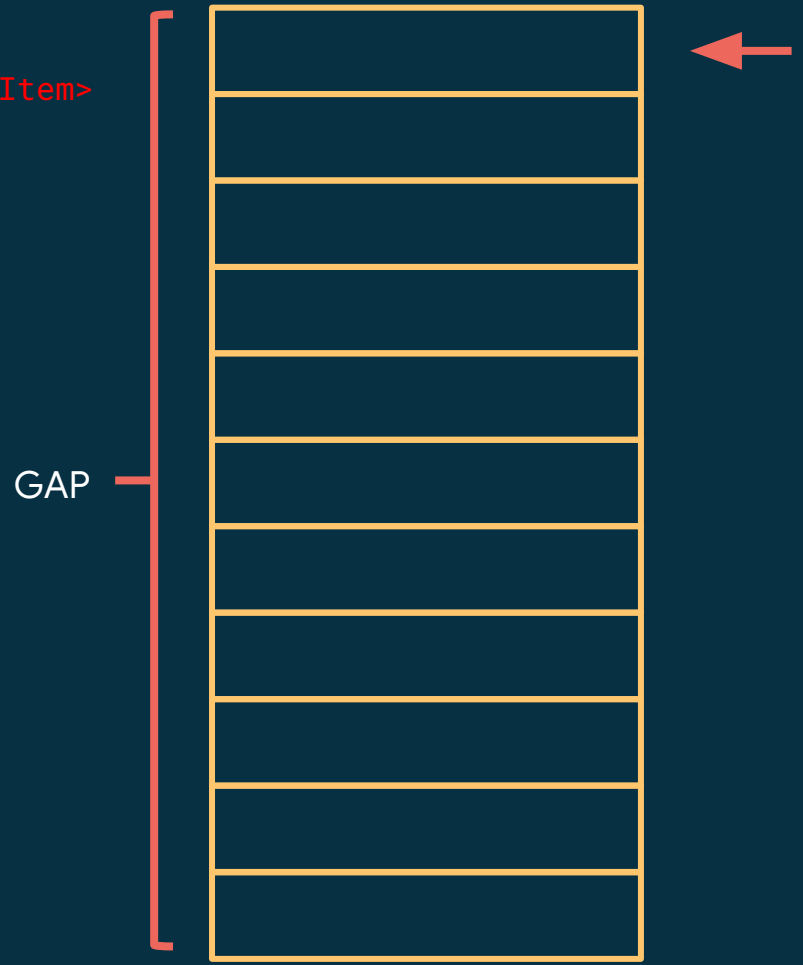
```
fun Feeds(
    $composer: Composer, $key: Int, items: List<FeedItem>
) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```



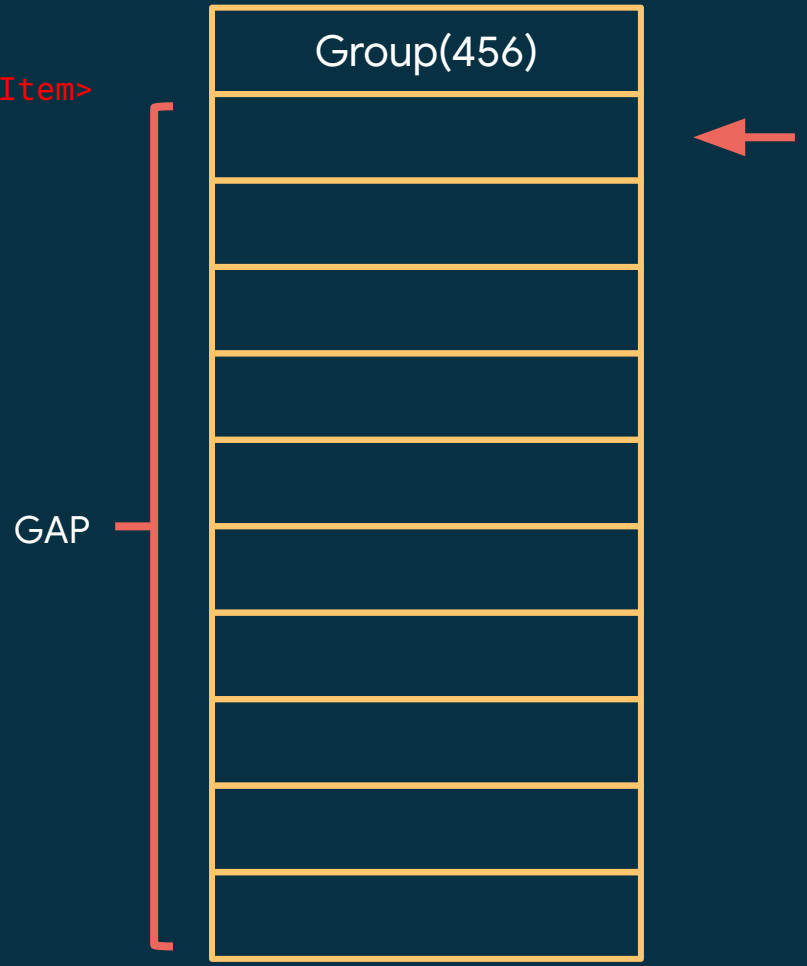
```
fun Feeds(
    $composer: Composer, $key: Int, items: List<FeedItem>
) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```



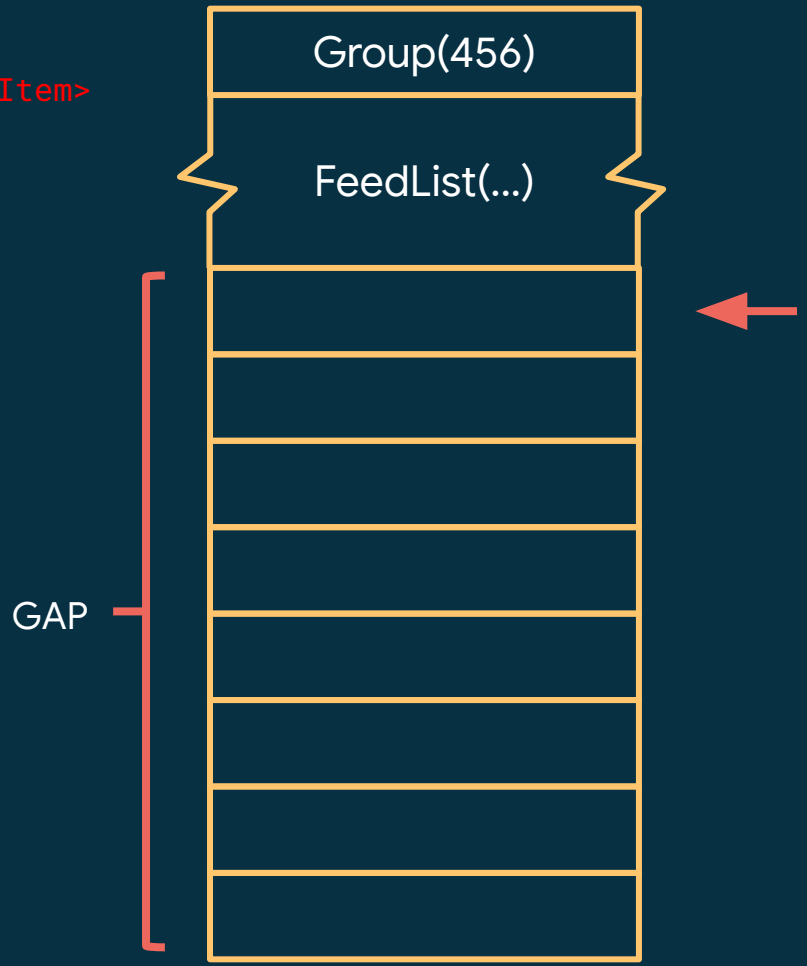
```
fun Feeds(
    $composer: Composer, $key: Int, items: List<FeedItem>
) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```



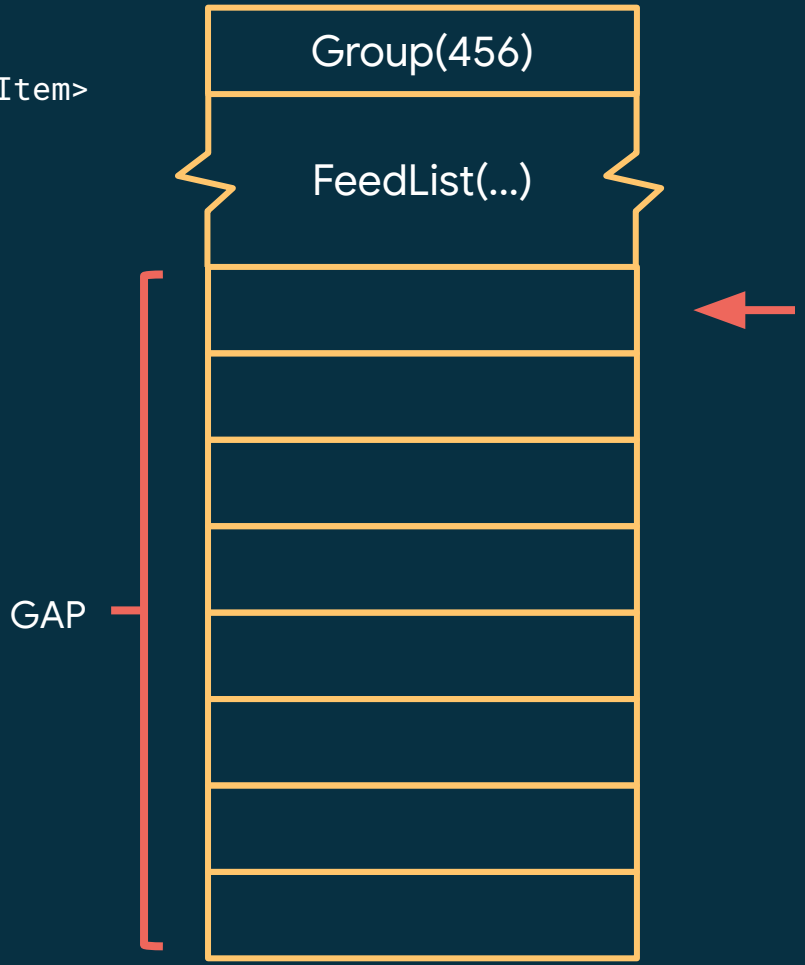
```
fun Feeds(
    $composer: Composer, $key: Int, items: List<FeedItem>
) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```



```
fun Feeds(
    $composer: Composer, $key: Int, items: List<FeedItem>
) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```



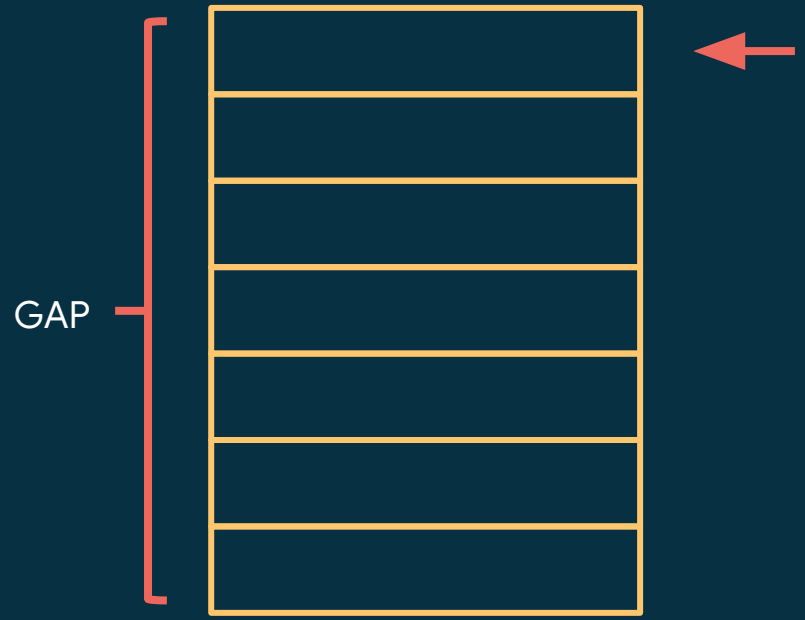
```
fun Feeds(
    $composer: Composer, $key: Int, items: List<FeedItem>
) {
    if (items.isEmpty) {
        $composer.start(123)
        Text($composer, key = …, "No feeds")
        $composer.end
    } else {
        $composer.start(456)
        FeedList($composer, key = …, items)
        $composer.end
    }
}
```



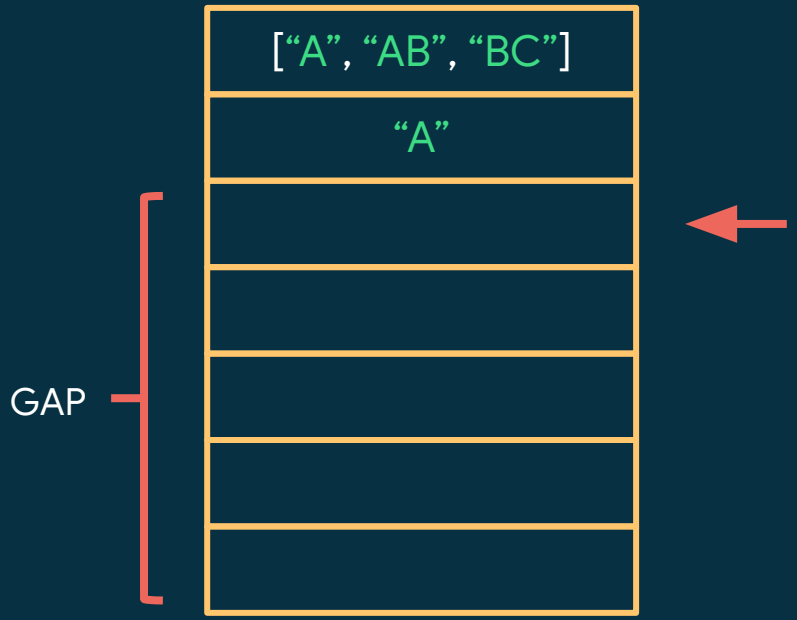
Positional memoization

```
@Composable
fun FilterList(items: List<Item>, query: String) {
  val filteredItems = remember(items, query) {
    items.filter(it.name.contains(query))
  }
  FeedList(items)
}
```

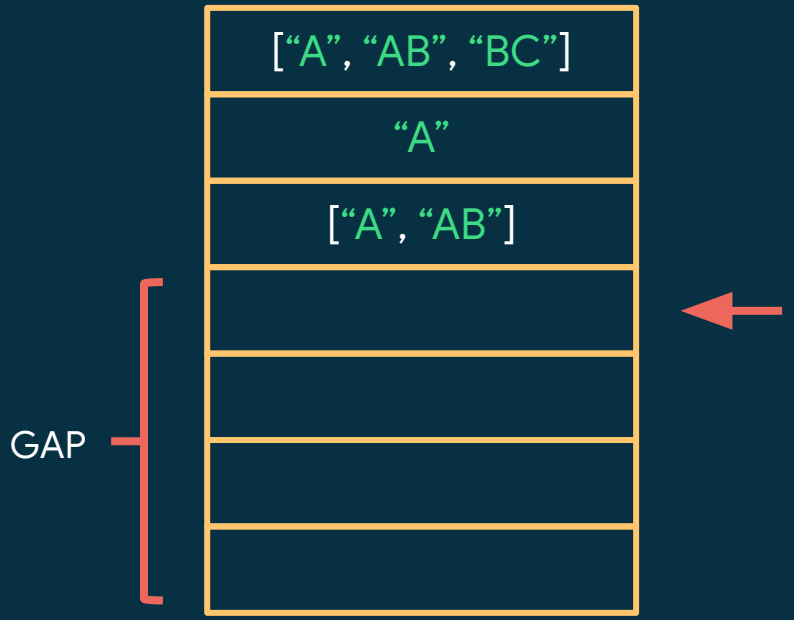
```
@Composable
fun FilterList(items: List<Item>, query: String) {
  val filteredItems = remember(items, query) {
    items.filter(it.name.contains(query))
  }
  FeedList(items)
}
```



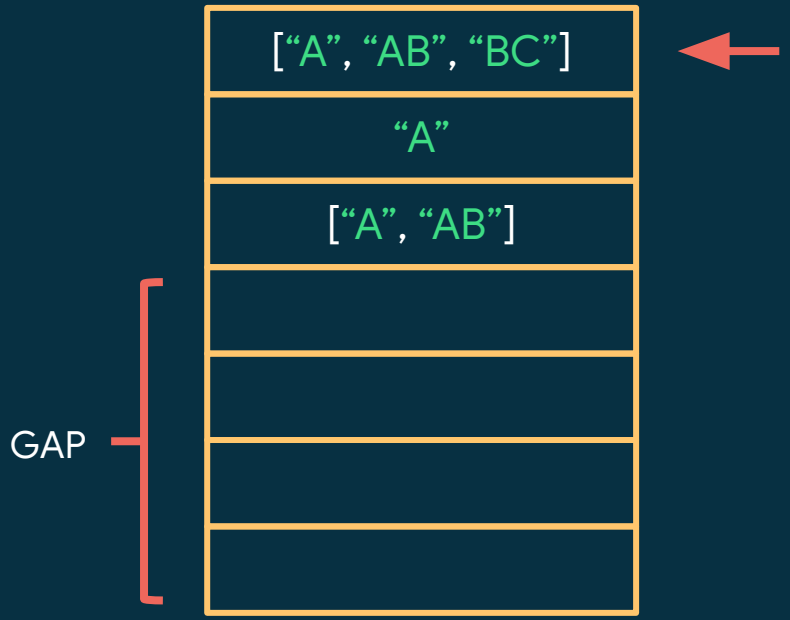
```
@Composable
fun FilterList(items: List<Item>, query: String) {
  val filteredItems = remember(items, query) {
    items.filter(it.name.contains(query))
  }
  FeedList(items)
}
```



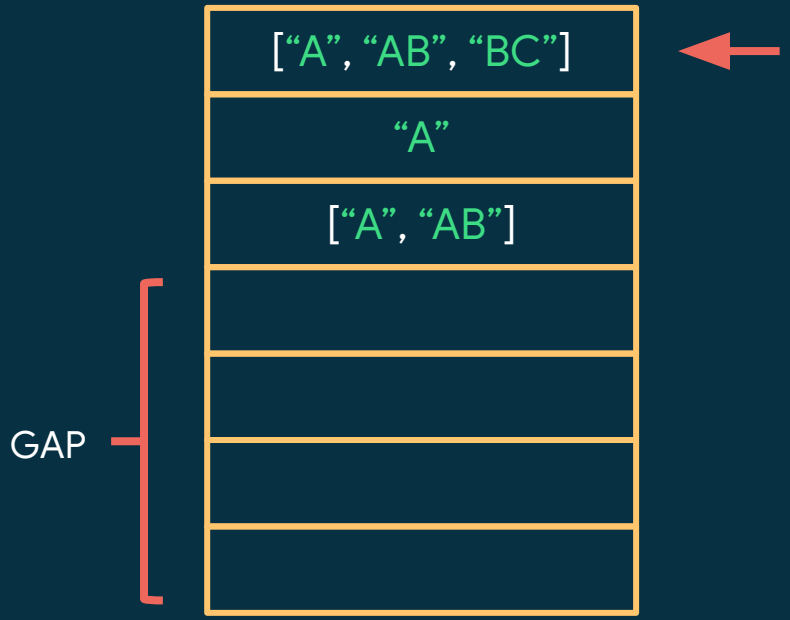
```
@Composable
fun FilterList(items: List<Item>, query: String) {
  val filteredItems = remember(items, query) {
    items.filter(it.name.contains(query))
  }
  FeedList(items)
}
```



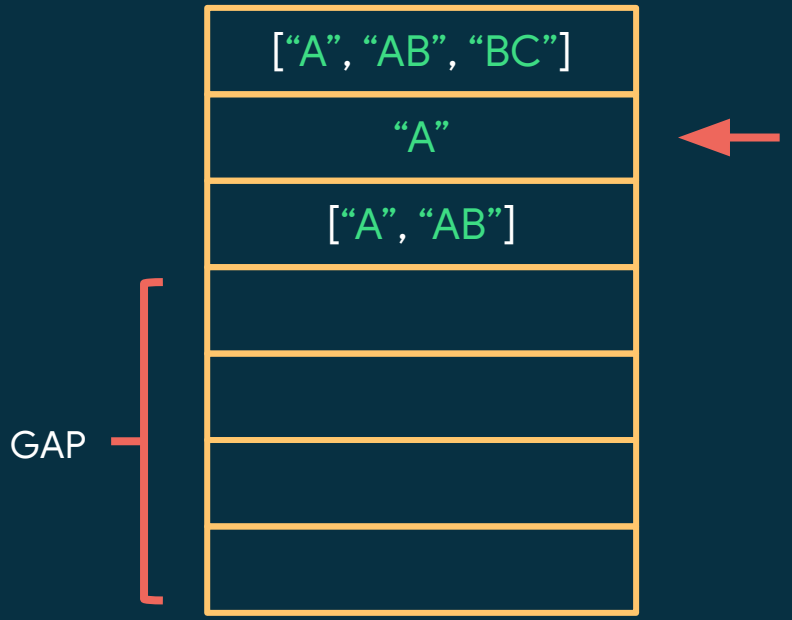
```
@Composable
fun FilterList(items: List<Item>, query: String) {
  val filteredItems = remember(items, query) {
    items.filter(it.name.contains(query))
  }
  FeedList(items)
}
```



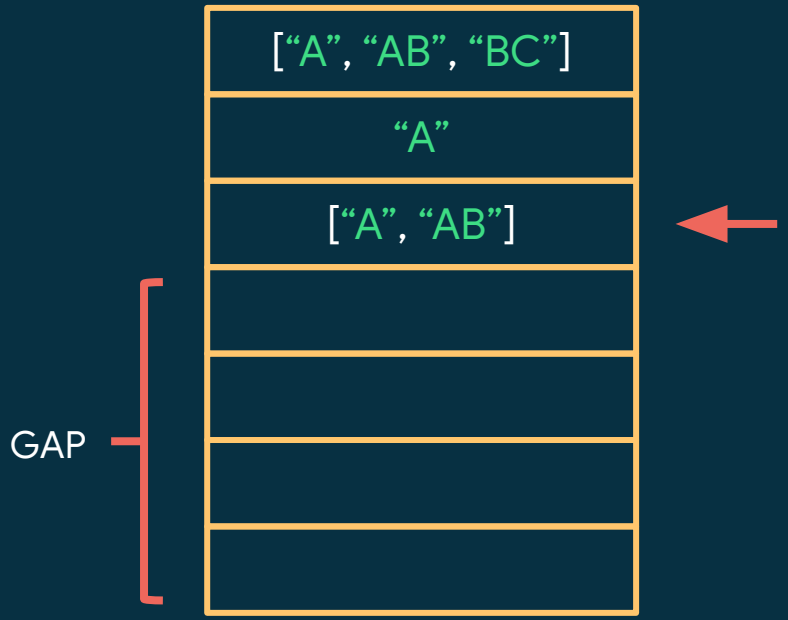
```
@Composable
fun FilterList(items: List<Item>, query: String) {
  val filteredItems = remember(items, query) {
    items.filter(it.name.contains(query))
  }
  FeedList(items)
}
```



```
@Composable
fun FilterList(items: List<Item>, query: String) {
  val filteredItems = remember(items, query) {
    items.filter(it.name.contains(query))
  }
  FeedList(items)
}
```



```
@Composable
fun FilterList(items: List<Item>, query: String) {
  val filteredItems = remember(items, query) {
    items.filter(it.name.contains(query))
  }
  FeedList(items)
}
```



```
@Composable
fun Item(name: String) {
    val result = remember {
        calculation()
    }
    Text("$name: $result")
}
```

```
@Composable
fun Counter(title: String) {
    var count by state { 0f }
    Button(
        text = "$title: $count",
        onClick = { count++ }
    )
}
```

```
@Composable
fun Counter(title: String) {
    var count by state { 0f }
    Button(
        text = "$title: $count",
        onClick = { count++ }
    )
}
```

```
@Composable
inline fun <T> state(init: () -> T): MutableState<T> =
    remember { mutableStateOf(init()) }
```

```
@Composable
inline fun <T> state(init: () -> T): MutableState<T> =
    remember { mutableStateOf(init()) }


fun <T> mutableStateOf(value: T): MutableState<T> { … }
```

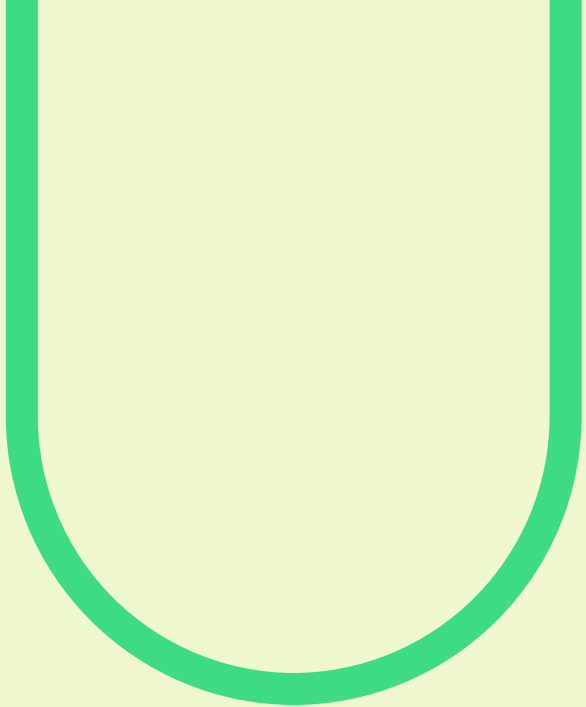
```
interface MutableState<T> : State<T> {
    override var value: T
}
```

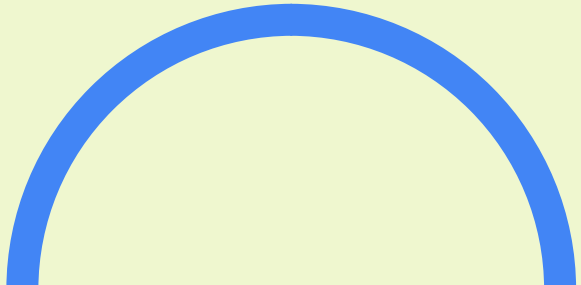
```
interface MutableState<T> : State<T> {
    override var value: T
}

interface State<T> {
    val value: T
}
```

mutableStateOf<T>(initial: T)

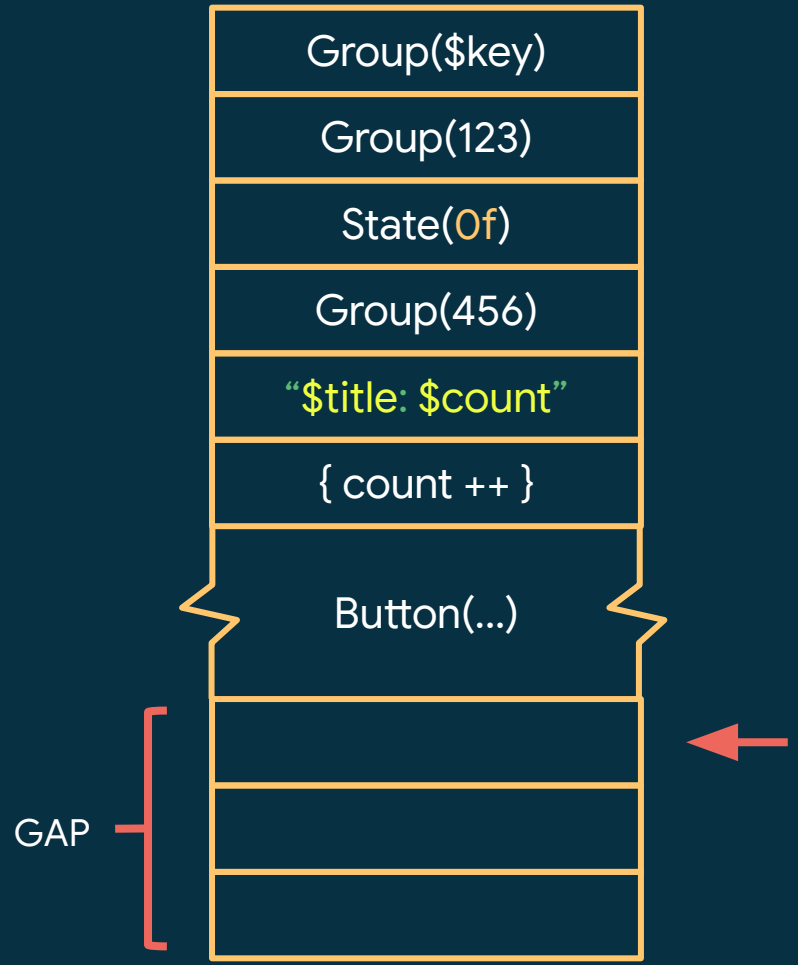
- Создает инстанс MutableState<T>, который
- Позволяет отслеживать прочтения
- Позволяет обновлять только прочитавших

Scoped observations with State<T>

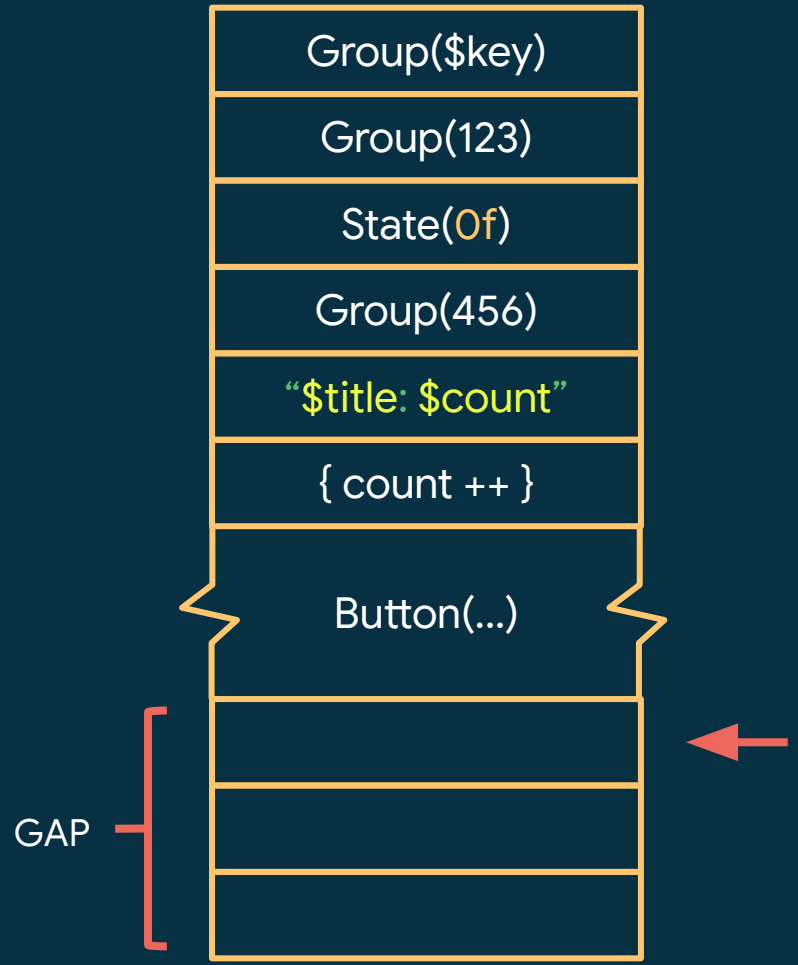




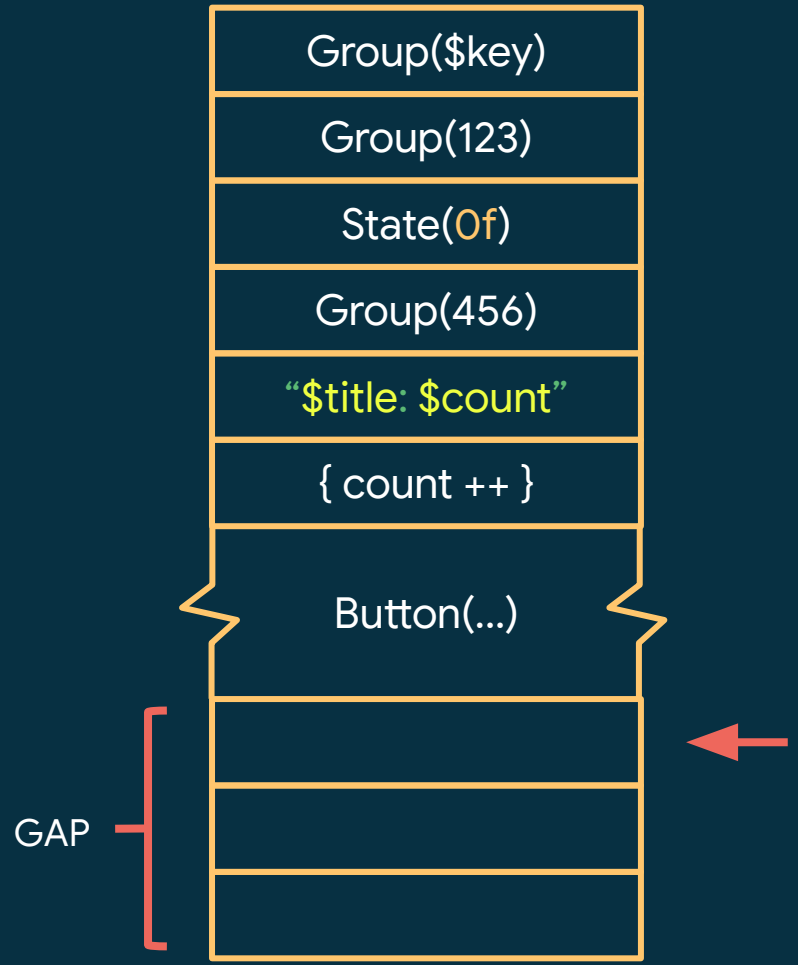
```
fun Counter(
    $composer:Composer,
    $key: Int,
    title: String
) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



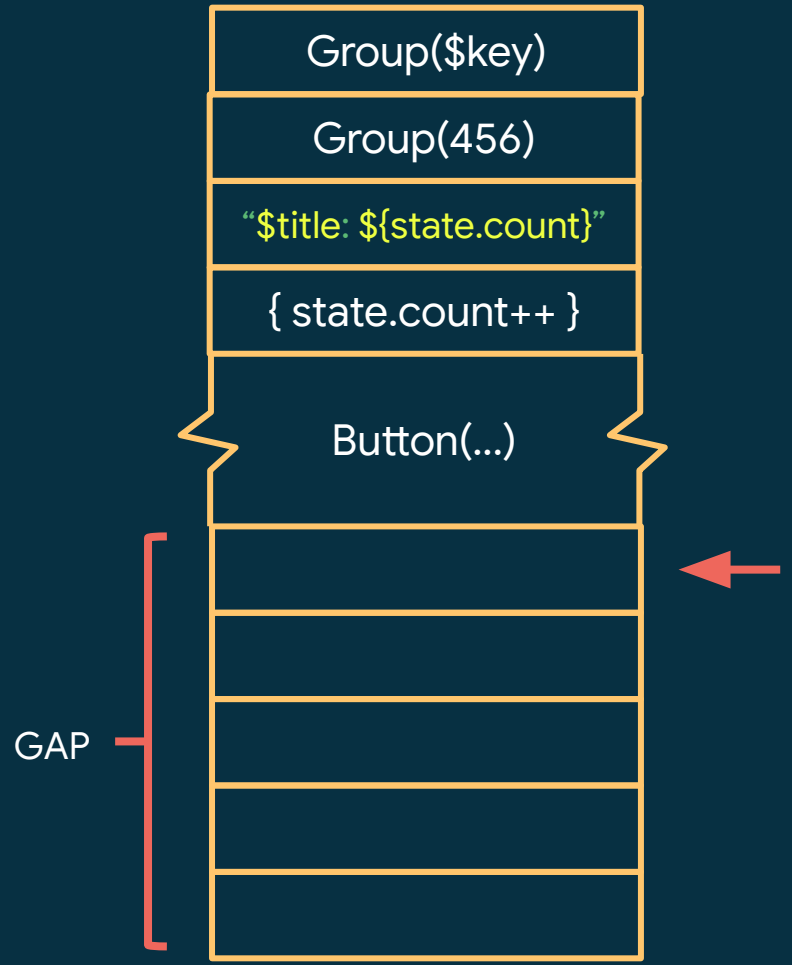
```
fun Counter(
    $composer:Composer,
    $key: Int,
    title: String
) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



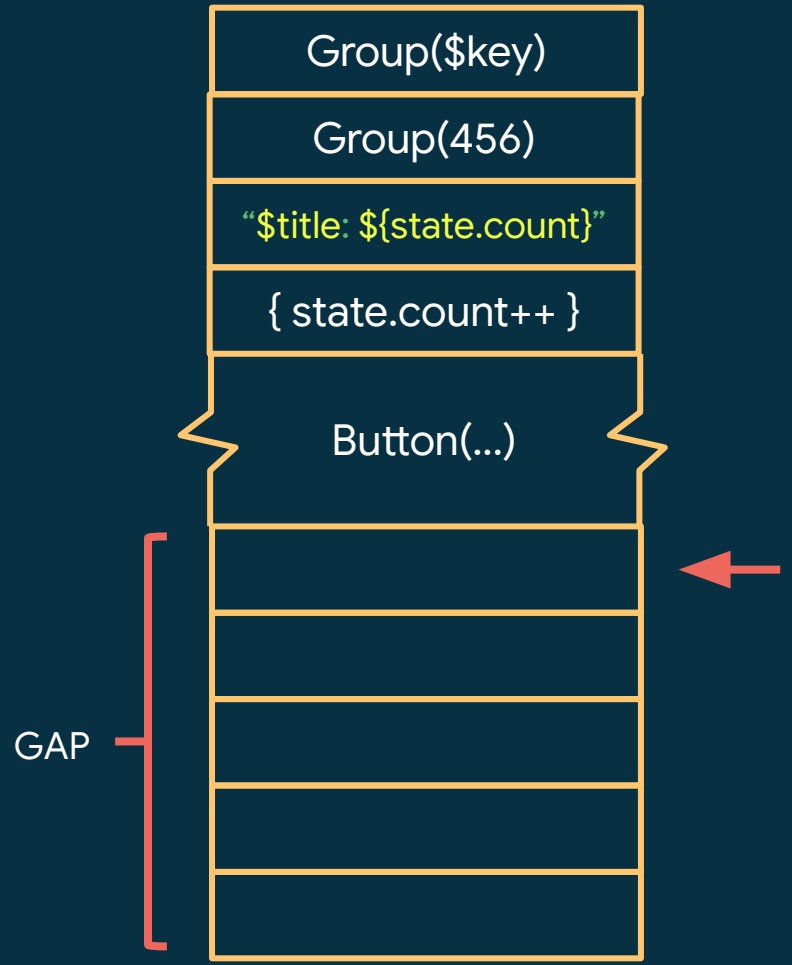
```
fun Counter(
    $composer:Composer,
    $key: Int,
    title: String
) {
    $composer.start($key)
    var count by state($composer,key=123) { 0f }
    Button($composer, key=456
        text = "$title: $count",
        onClick = { count++ }
    )
    $composer.end
}
```



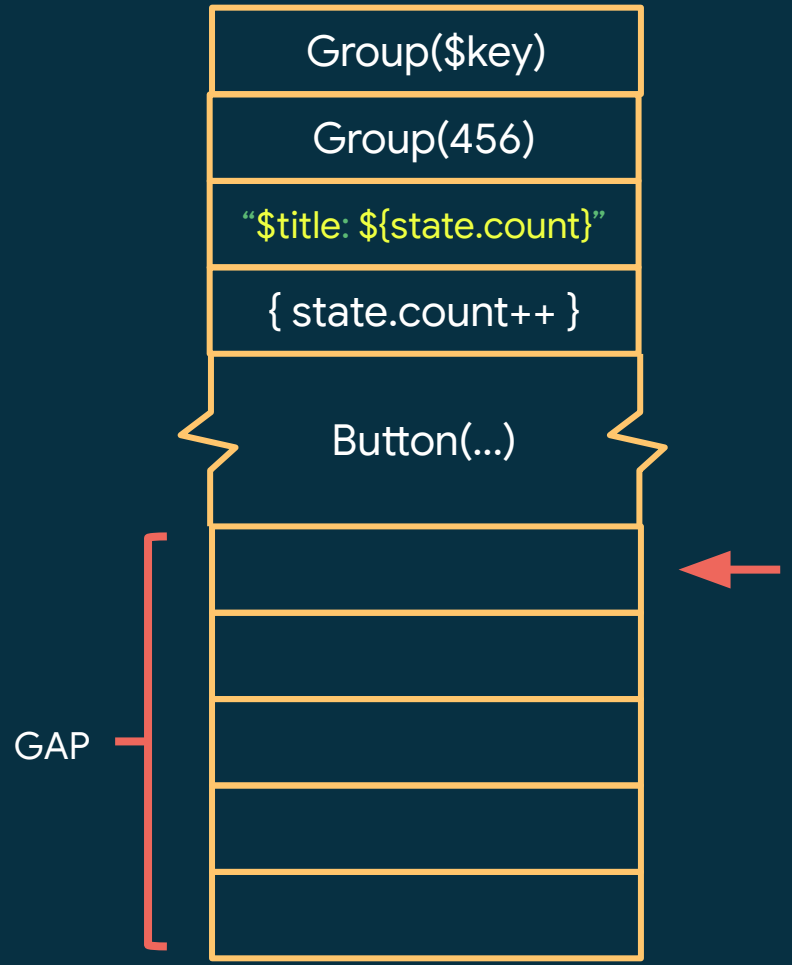
```
fun Counter(
    $composer:Composer,
    $key: Int,
    title: String,
    state: MutableState<Int>
) {
    $composer.start($key)
    Button($composer, key=456
        text = "$title: ${state.value}",
        onClick = { state.value++ }
    )
    $composer.end
}
```



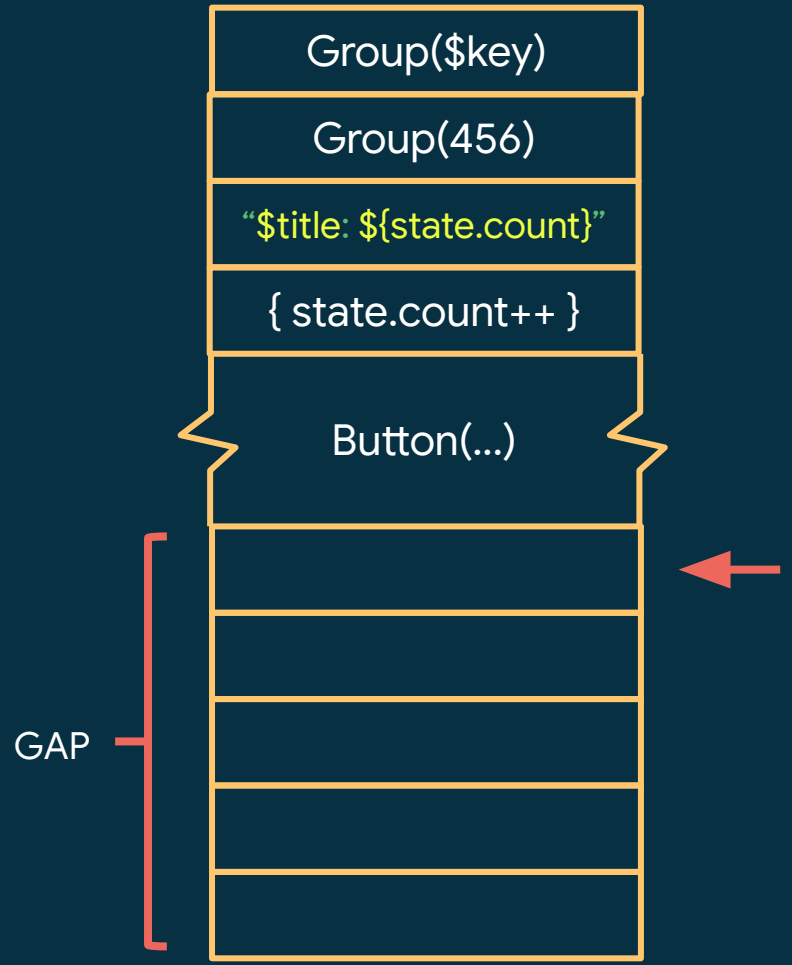
```
fun Counter(
    $composer:Composer,
    $key: Int,
    title: String,
    state: MutableState<Int>
) {
    $composer.start($key)
    Button($composer, key=456
        text = "$title: ${state.value}",
        onClick = { state.value++ }
    )
    $composer.end
}
```



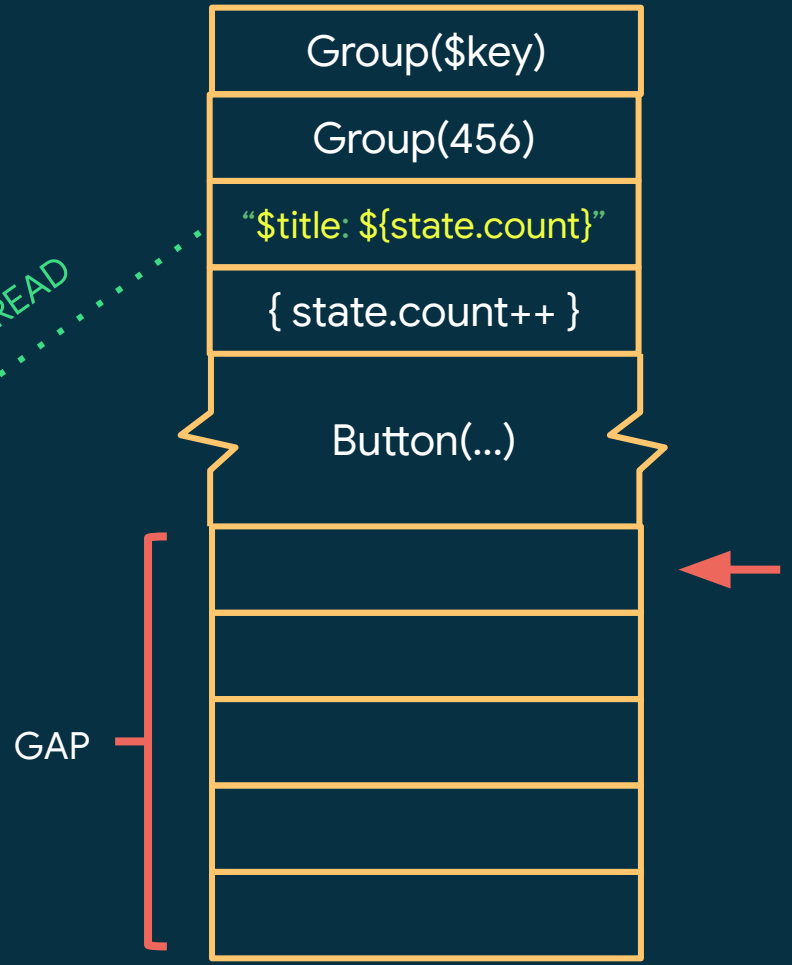
```
fun Counter(
    $composer:Composer,
    $key: Int,
    title: String,
    state: MutableState<Int>
) {
    $composer.start($key)
    Button($composer, key=456
        text = "$title: ${state.value}",
        onClick = { state.value++ }
    )
    $composer.end
}
```



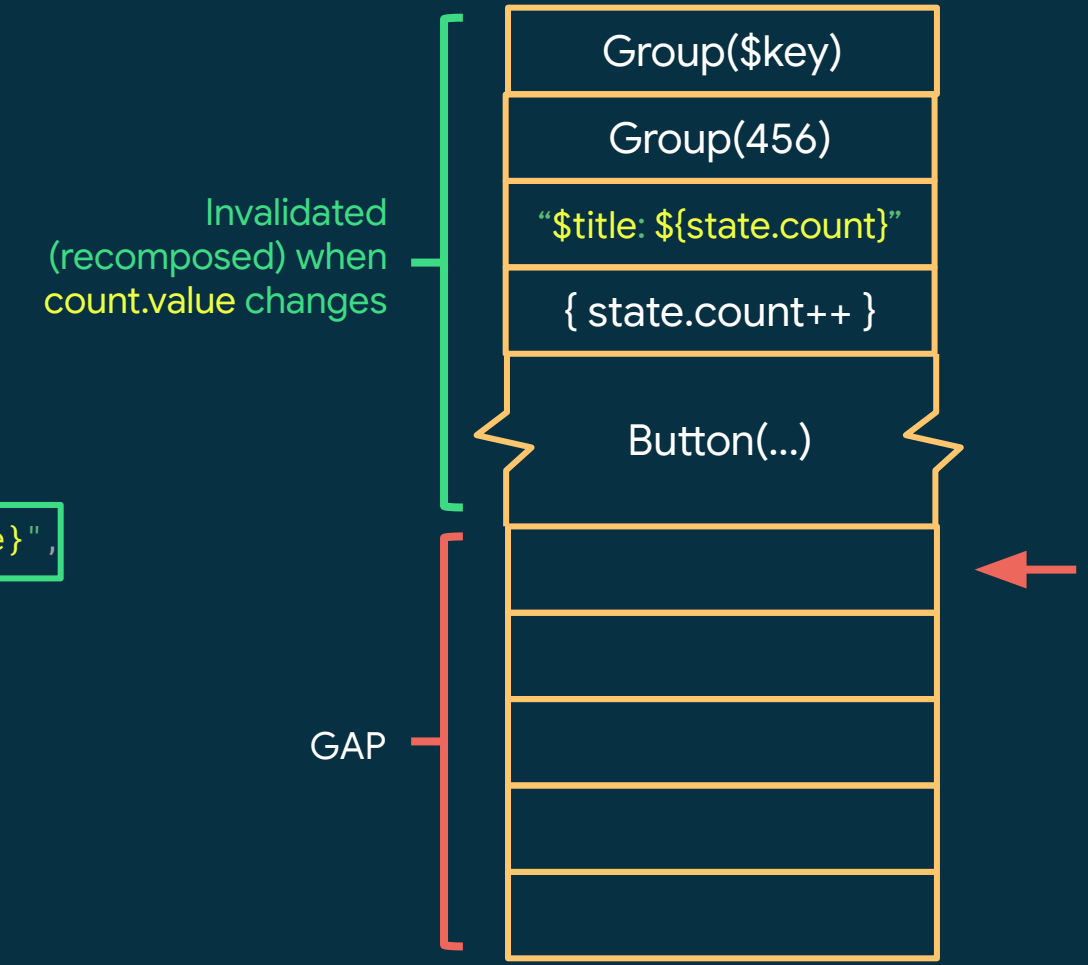
```
fun Counter(
    $composer:Composer,
    $key: Int,
    title: String,
    state: MutableState<Int>
) {
    $composer.start($key)
    Button($composer, key=456
        text = "$title: ${state.value}",
        onClick = { state.value++ }
    )
    $composer.end
}
```

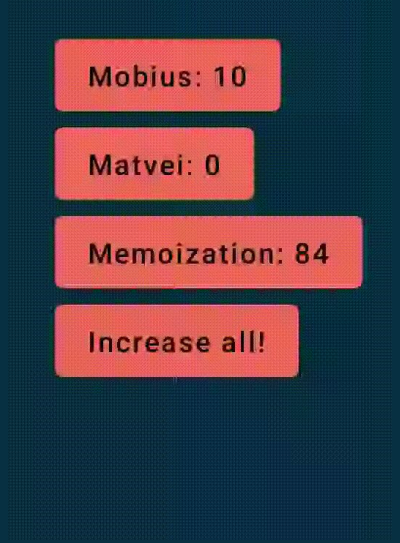


```
fun Counter(
    $composer:Composer,
    $key: Int,
    title: String,
    state: MutableState<Int>
) {
    $composer.start($key)
    Button($composer, key=456
        text = "$title: ${state.value}",
        onClick = { state.value++ }
    )
    $composer.end
}
```



```
fun Counter(
    $composer:Composer,
    $key: Int,
    title: String,
    state: MutableState<Int>
) {
    $composer.start($key)
    Button($composer, key=456
        text = "$title: ${state.value}",
        onClick = { state.value++ }
    )
    $composer.end
}
```




Mobius: 10
Matvei: 0
Memoization: 84
Increase all!

```
fun App(counterData: Map<String, MutableState<Int>>) {
```

```
fun App(counterData: Map<String, MutableState<Int>>) {
```

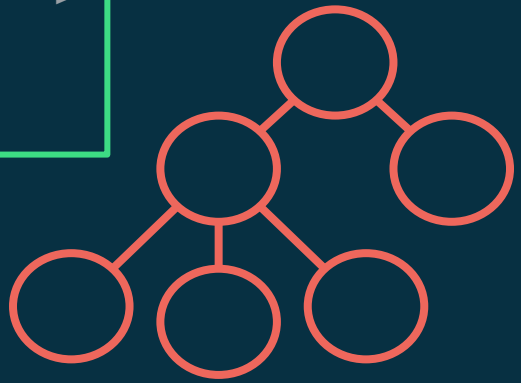
```
fun App(counterData: Map<String, MutableState<Int>>) {
  Column {
```

```
fun App(counterData: Map<String, MutableState<Int>>) {
  Column {
    counterData.forEach { (title, counterState) ->
      Counter(title, counterState)
    }
  }
}
```

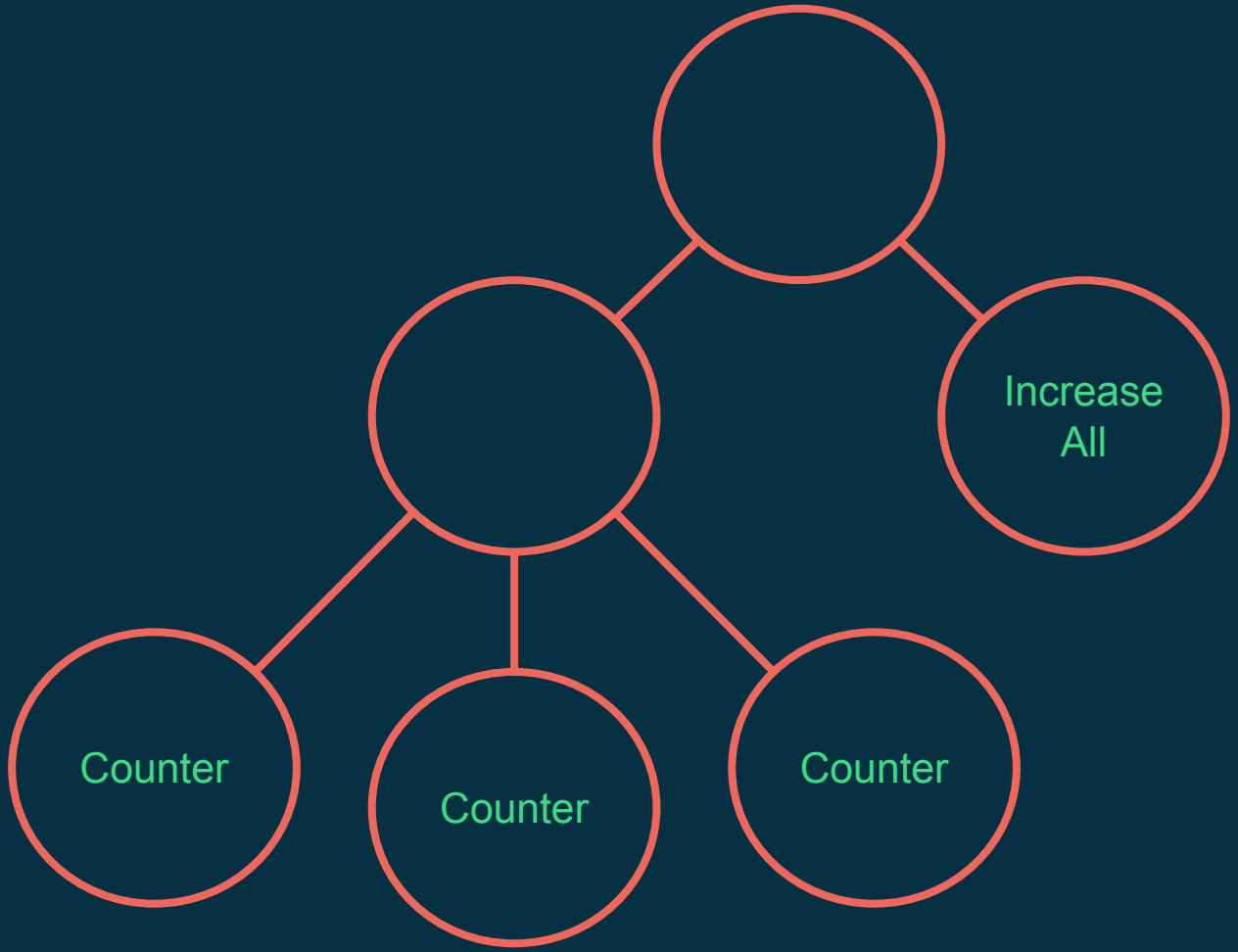
```
fun App(counterData: Map<String, MutableState<Int>>) {
  Column {
    counterData.forEach { (title, counterState) ->
      Counter(title, counterState)
    }
  }
}
```

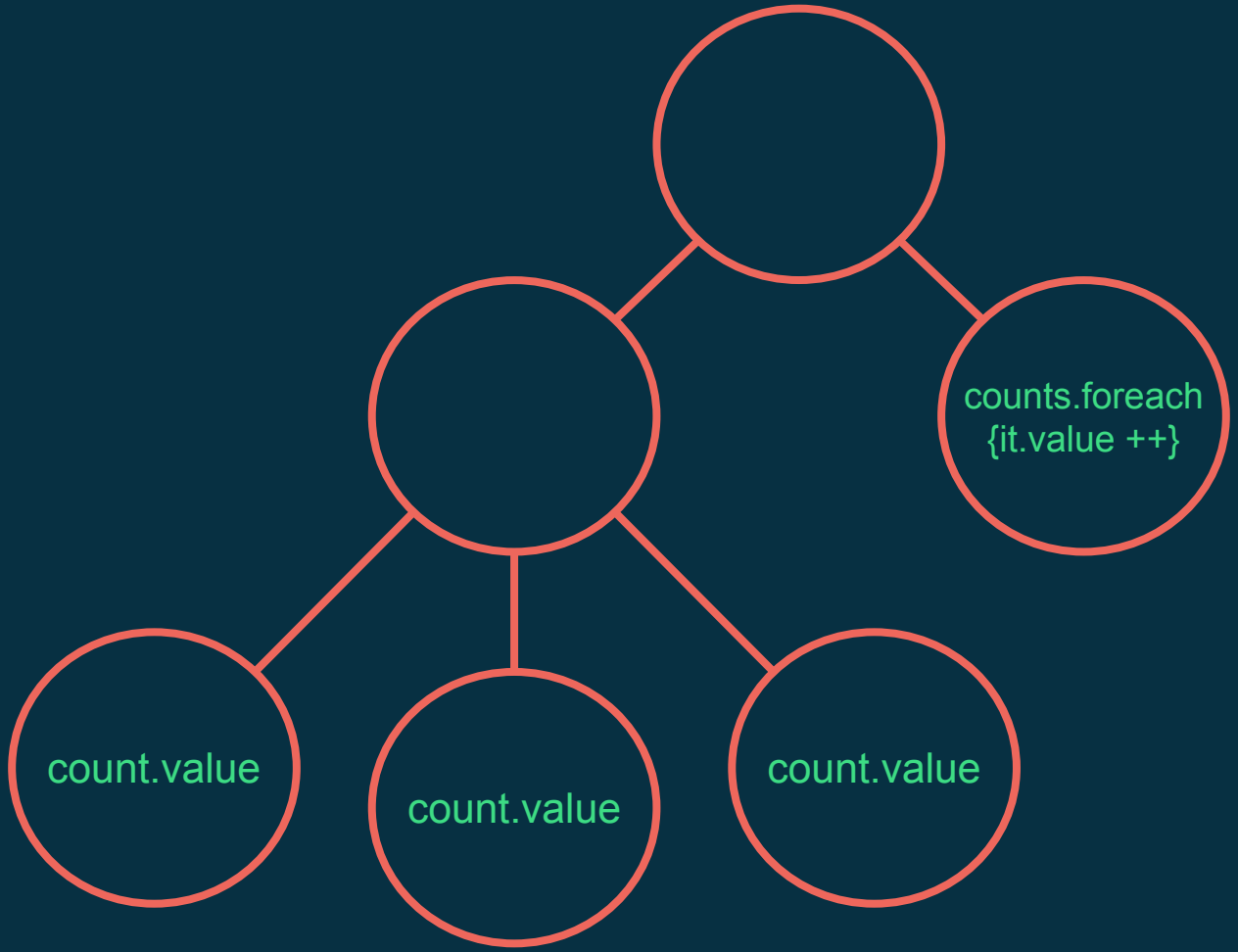
```
fun App(counterData: Map<String, MutableState<Int>>) {
  Column {
    counterData.forEach { (title, counterState) ->
      Counter(title, counterState)
    }
    Button(
      text = "Increase All!",
      onClick = {
        counterData.forEach{ (_, countState) ->
          countState.value++
        }
      }
    )
  }
}
```

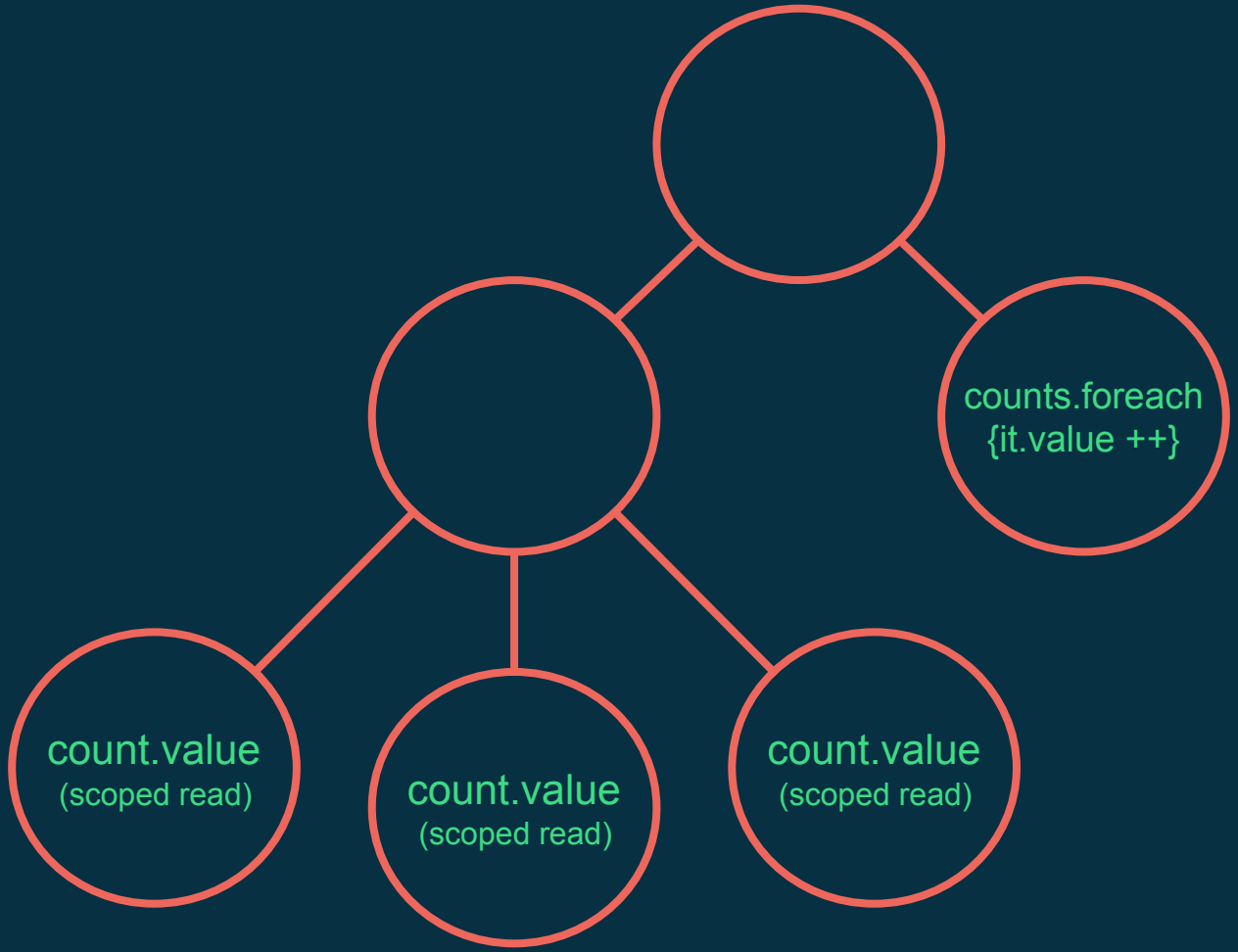
```
fun App(counterData: Map<String, MutableState<Int>>) {
  Column {
    counterData.forEach { (title, counterState) ->
      Counter(title, counterState)
    }
    Button(
      text = "Increase All!",
      onClick = {
        counterData.forEach{ (_, countState) ->
          countState.value++
        }
      }
    )
  }
}
```

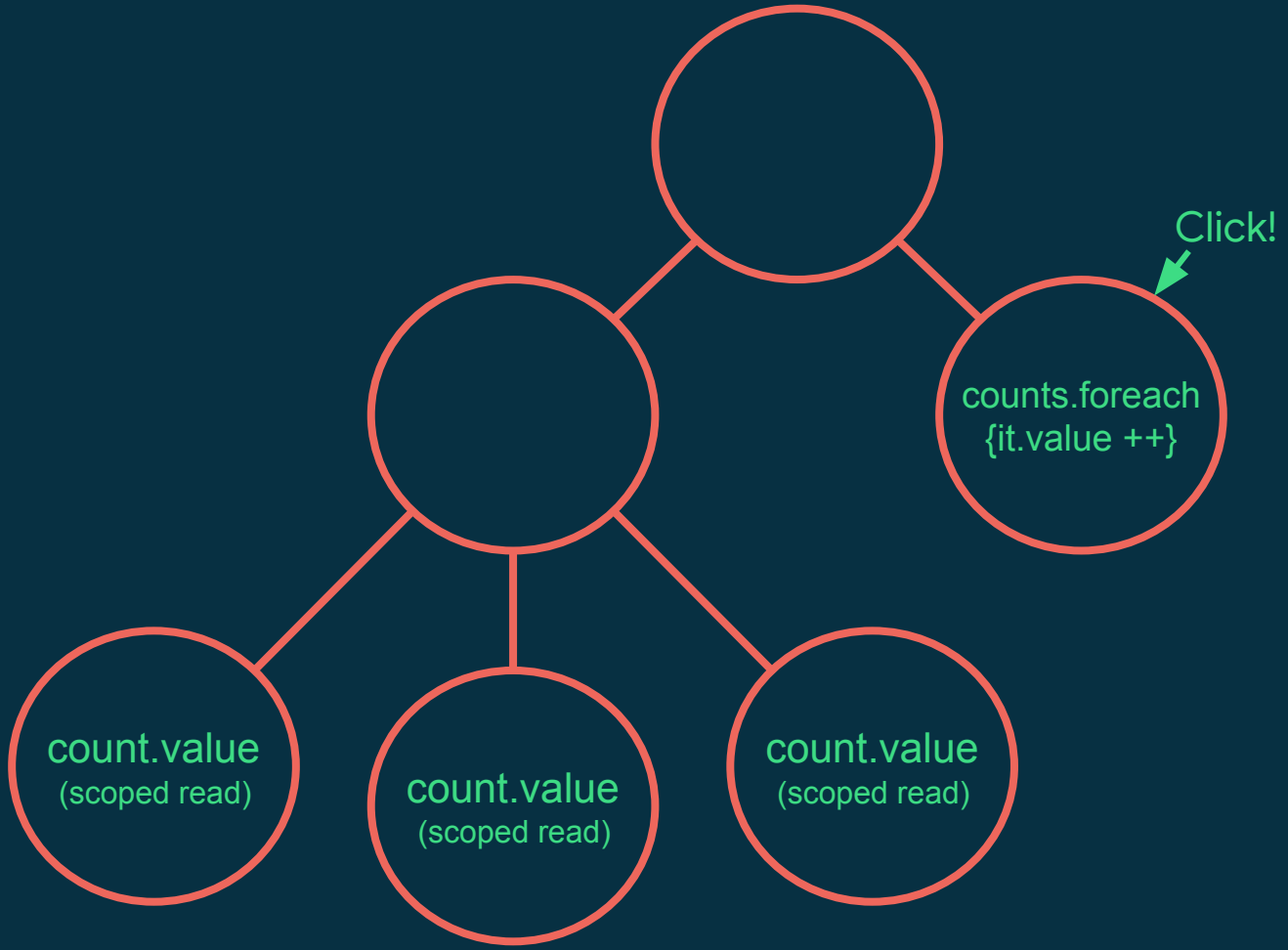


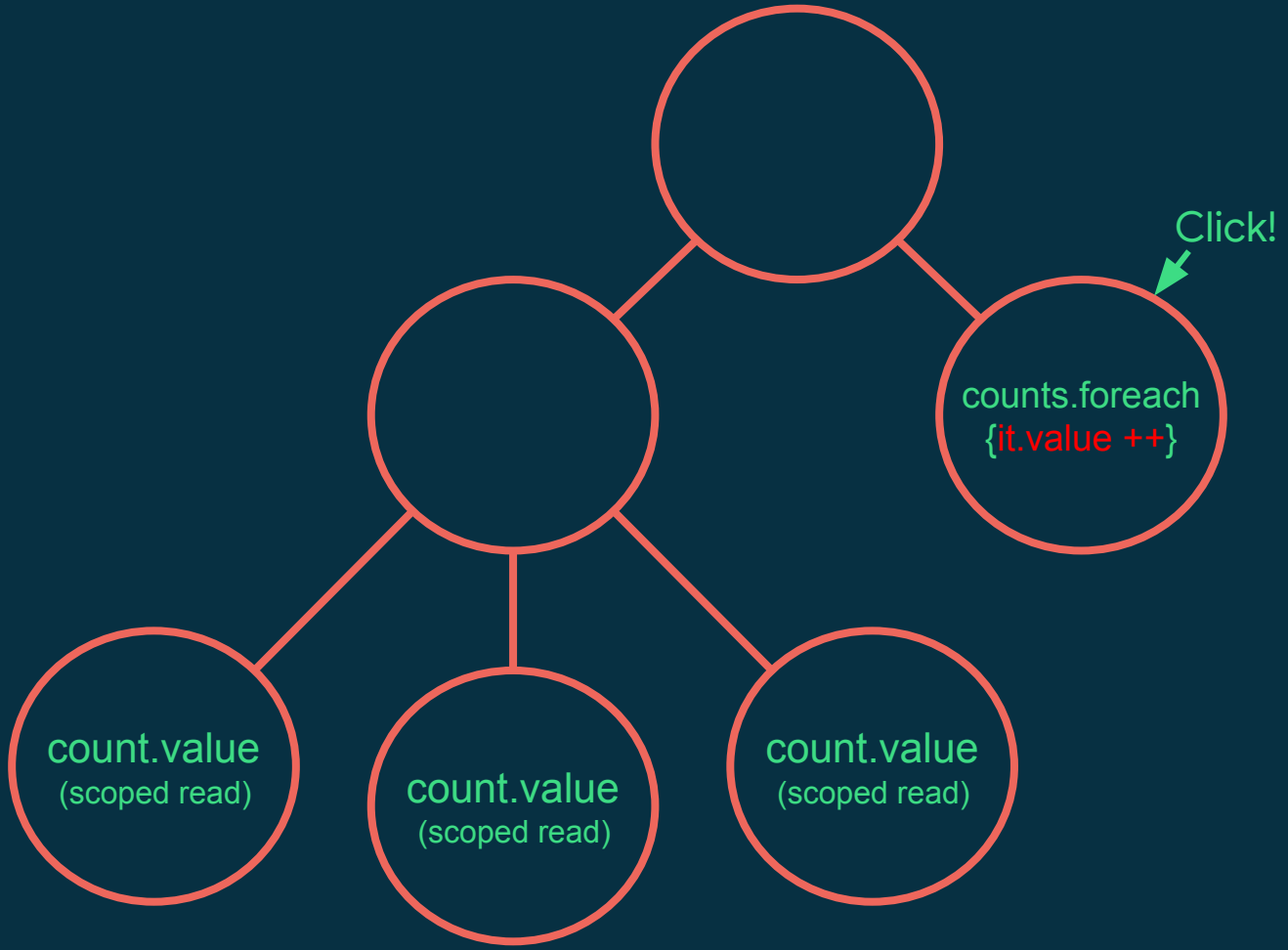
```
fun App(counterData: Map<String, MutableState<Int>>) {
  Column {
    counterData.forEach { (title, counterState) ->
      Counter(title, counterState)
    }
    Button(
      text = "Increase All!",
      onClick = {
        counterData.forEach{ (_, countState) ->
          countState.value++
        }
      }
    )
  }
}
```

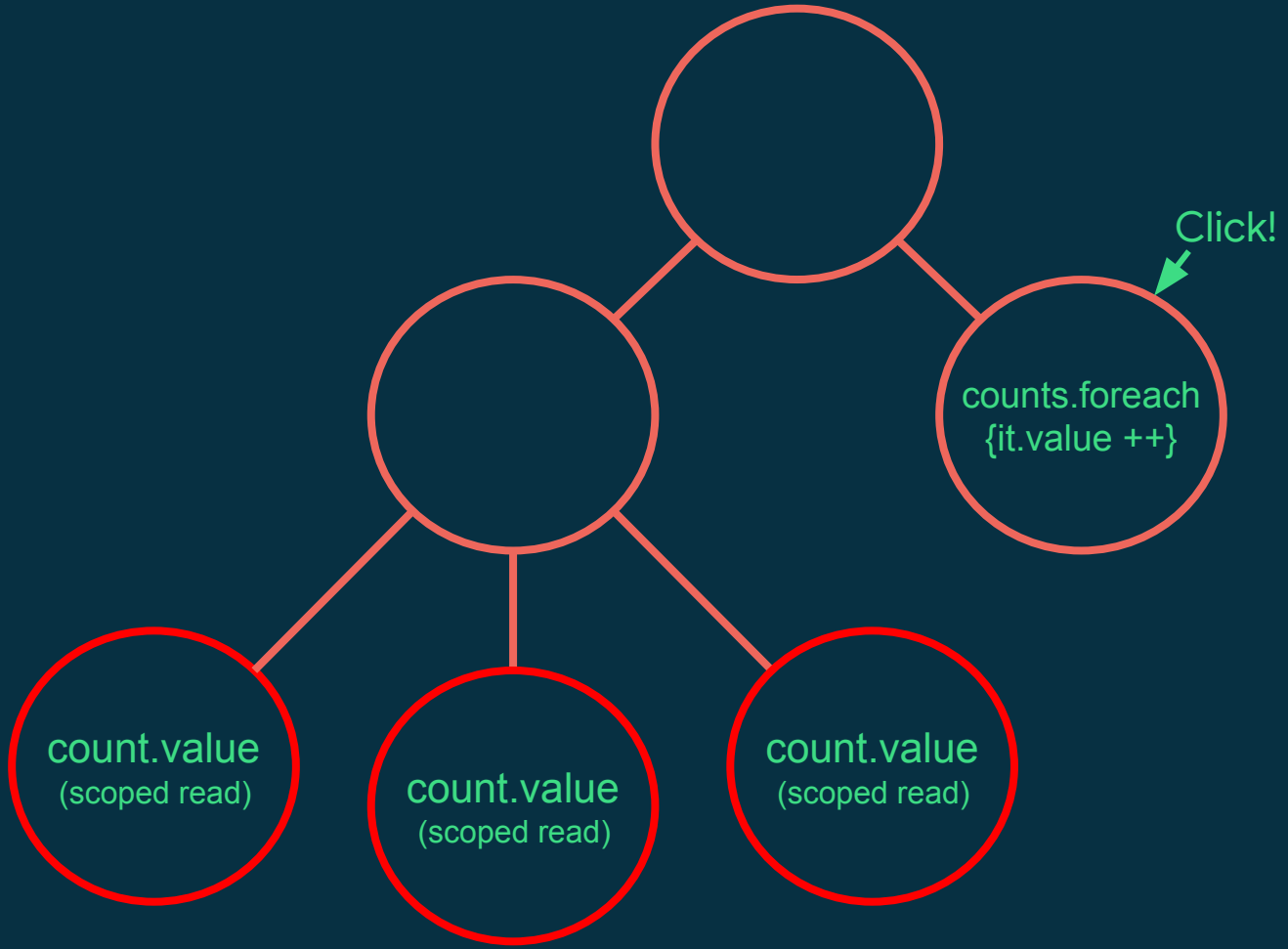

Increase
All
Counter
Counter
Counter


counts.foreach
{it.value ++}
count.value
count.value
count.value


counts.foreach
{it.value ++}
count.value
(scoped read)
count.value
(scoped read)
count.value
(scoped read)


Click!
counts.foreach
{it.value ++}
count.value
(scoped read)
count.value
(scoped read)
count.value
(scoped read)


Click!
counts.foreach
{it.value ++}
count.value
(scoped read)
count.value
(scoped read)
count.value
(scoped read)


Click!
counts.foreach
{it.value ++}
count.value
(scoped read)
count.value
(scoped read)
count.value
(scoped read)

Обновляется только то что поменялось

State<T> & MutableState<T>

- Чтение state.value подписывает вас на обновления этого значения
- Быстрое создание composition-bound стейтов
- Поддержка разных скоупов:

  прочитал в композиции -- компонент перекомпозировали;

  прочитал в лейауте -- компонент перелейаутится;

  прочитал в стадии отрисовки -- компонент перерисуется

Shared MutableState<> не бесплатный

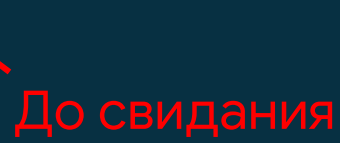
- Если вы принимаете MutableState как параметр компонента
- То этот компонент сможет менять стейт **родителя**
- Станет сложнее следить за общим состоянием системы

```
@Composable
fun Counter(
    countState: MutableState<Int>
) {
    Button(
        text = "count: ${countState.value}",
        onClick = { countState.value += 1 }
    )
}
```

```
@Composable
fun App() {
    val counterState: MutableState<Int> = state { 0f }
    Counter(counterState)
}
```

```
@Composable
fun App() {
    val counterState: MutableState<Int> = state { 0f }
    Counter(counterState)
}
```

```
@Composable
fun App() {
    val counterState: MutableState<Int> = state { 0f }
    Counter(counterState)
}
```



Controlled components
State hoisting

```
@Composable
fun Counter(
    currentCount: Int,
    onCountChange: (Int) -> Unit
) {
    Button(
        text = "count: $currentCount",
        onClick = { onCountChange(currentCount + 1) }
    )
}
```

```
@Composable
fun Counter(
    currentCount: Int,
    onCountChange: (Int) -> Unit
) {
    Button(
        text = "count: $currentCount",
        onClick = { onCountChange(currentCount + 1) }
    )
}
```

```
@Composable
fun Counter(
    currentCount: Int,
    onCountChange: (Int) -> Unit
) {
    Button(
        text = "count: $currentCount",
        onClick = { onCountChange(currentCount + 1) }
    )
}
```

```
@Composable
fun App() {
    val counterState: MutableState<Int> = state { 0f }
    Counter(
        currentCount = counterState.value,
        onCountChange = { counterState.value += 1 }
    )
}
```

```
@Composable
fun App() {
    val counterState: MutableState<Int> = state { 0f }
    Counter(
        currentCount = counterState.value,
        onCountChange = { counterState.value += 1 }
    )
}
```

```
@Composable
fun App() {
    val counterState: MutableState<Int> = state { 0f }
    Counter(
        currentCount = counterState.value,
        onCountChange = { counterState.value += 1 }
    )
}
```

```
@Composable
fun App() {
    val counterState: MutableState<Int> = state { 0f }
    Counter(
        currentCount = counterState.value,
        onCountChange = { counterState.value += 1 }
    )
}
```

```
@Composable
fun App() {
    var counterStateValue: Int by state { 0f }
    Counter(
        currentCount = counterStateValue,
        onCountChange = { counterStateValue += 1 }
    )
}
```

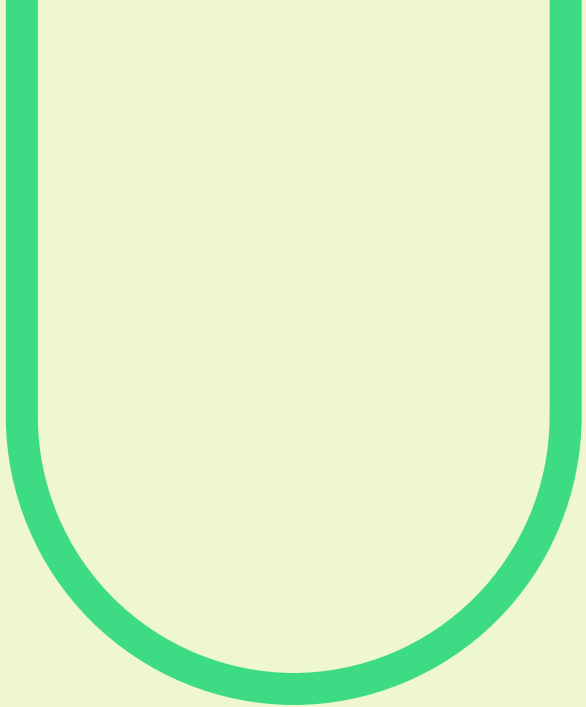
```
@Composable
fun App() {
    var counterStateValue: Int by state { 0f }
    Counter(
        currentCount = counterStateValue,
        onCountChange = { counterStateValue += 1 }
    )
}
```

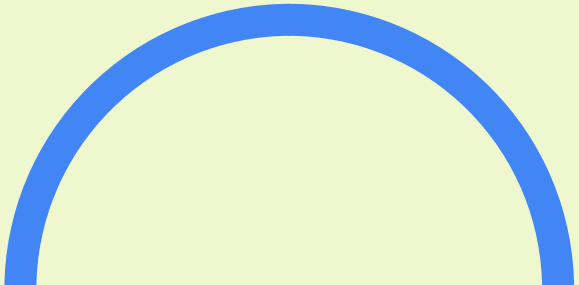
```
@Composable
fun App() {
    var counterStateValue: Int by state { 0f }
    Counter(
        currentCount = counterStateValue,
        onCountChange = { counterStateValue += 1 }
    )
}
```

гибкость == ответственность

- MutableState<T> в публичных АПИ ведет к разделению владения этим стейтом
- Альтернатива 1: controlled components
- Альтернатива 2: принимать State<T> только для чтения
- С иммутабельными классы и структурами таких проблем нет

Вместо выводов





Декларативные фреймворки абстрагируют императивные

Декларативные UI фреймворки абстрагируют обновления UI и дают нам UI = ƒ(state)

Декларативные UI фреймворки абстрагируют обновления UI и дают нам UI = ƒ(state)

Быстро обновляют UI внутри себя

Чтобы вы могли заниматься действительно важными вещами*

***Но понимали, как это работает внутри**

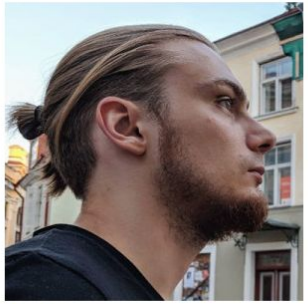
Jetpack Compose over inheritance

RU / Day 4 / 17:30 / Track 2 ☆ To favorites

Jetpack Compose is a new UI framework from the Android team which aims to simplify interface development. It combines the declarative approach and pithiness that Kotlin gives.

In this talk together with the engineer working on Jetpack Compose, we will look at how the UI components work, how to write your ones and how easy to reuse and customize them. We will go through creating a complex UI via a simple example, understand what happens under the hood, what decisions had to be made during the development process and how Kotlin makes our API better.

All talks



Andrey Kulikov

Google

and_kulikov

Andrey has been developing Android applications since 2012. He's the author of the popular Transitions-Everywhere library. having settled in Google he transferred all results from this library to the official AndroidX Transitions library. Currently working on Jetpack Compose.

Anastasia Soboleva

Google

Anastasia is an engineer at Google working on Jetpack Compose. Prior to that she worked on a few exciting projects at Amazon as a backend engineer and parallels as an iOS developer. When not working, Anastasia is doing a bit of everything like sketching, playing badminton, cycling or walking. And she is constantly learning new languages which she forgets pretty quickly, at the moment these are French and Mandarin.

Ссылочки

- Главная
  - https://developer.android.com/jetpack/compose
- Туториал
  - https://goo.gle/compose-codelab
- Багтрекер (фича реквесты и баги)
  - https://goo.gle/compose-feedback
- Kotlin slack, #compose channel
  - http://slack.kotlinlang.org/




Спасибо

Вопросики

Ссылочки

- Главная
  - https://developer.android.com/jetpack/compose
- Туториал
  - https://goo.gle/compose-codelab
- Багтрекер (фича реквесты и баги)
  - https://goo.gle/compose-feedback
- Kotlin slack, #compose channel
  - http://slack.kotlinlang.org/