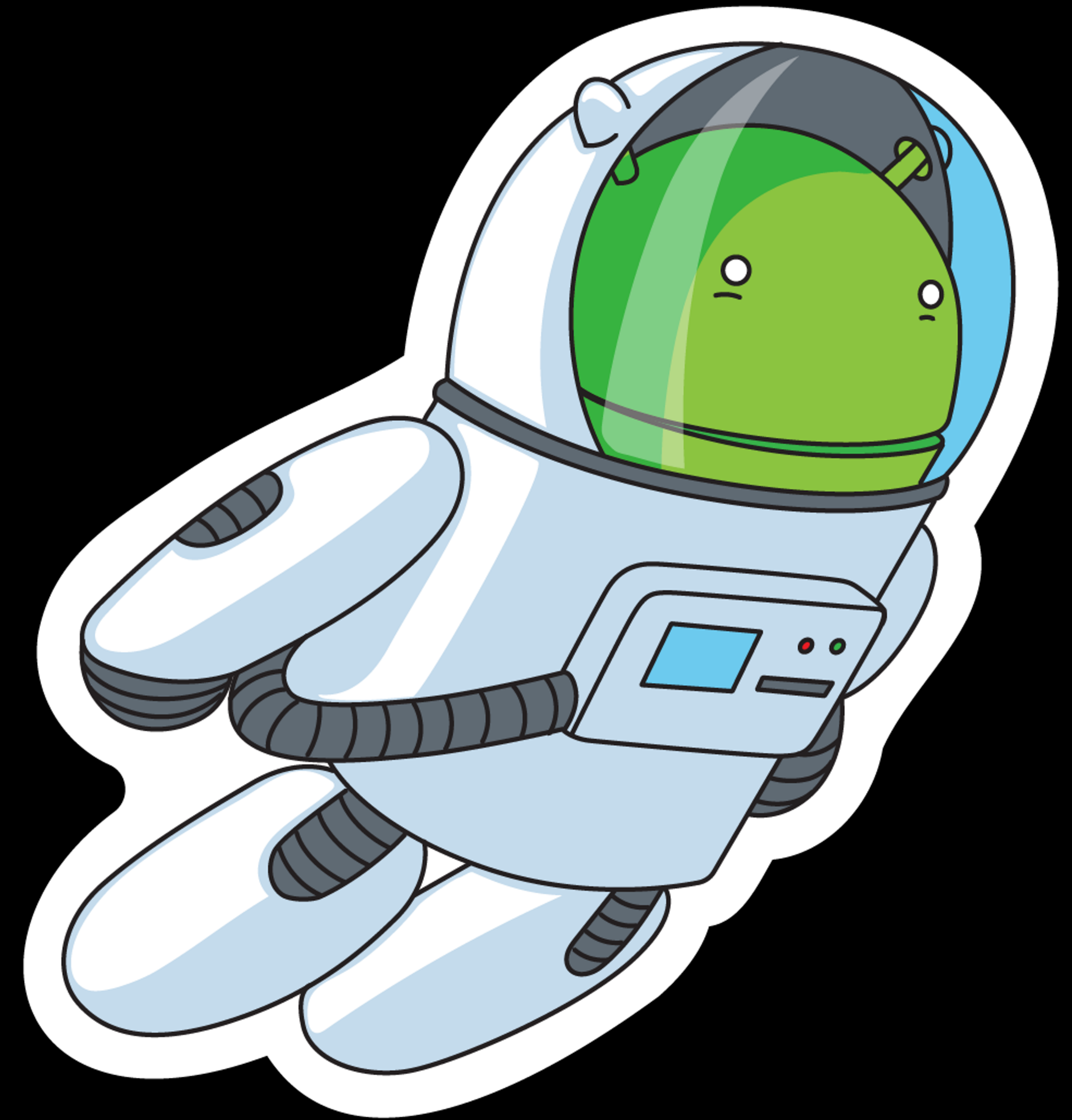
avito.tech

Санкт-Петербург 2024

Кастомные лейауты в Jetpack Compose



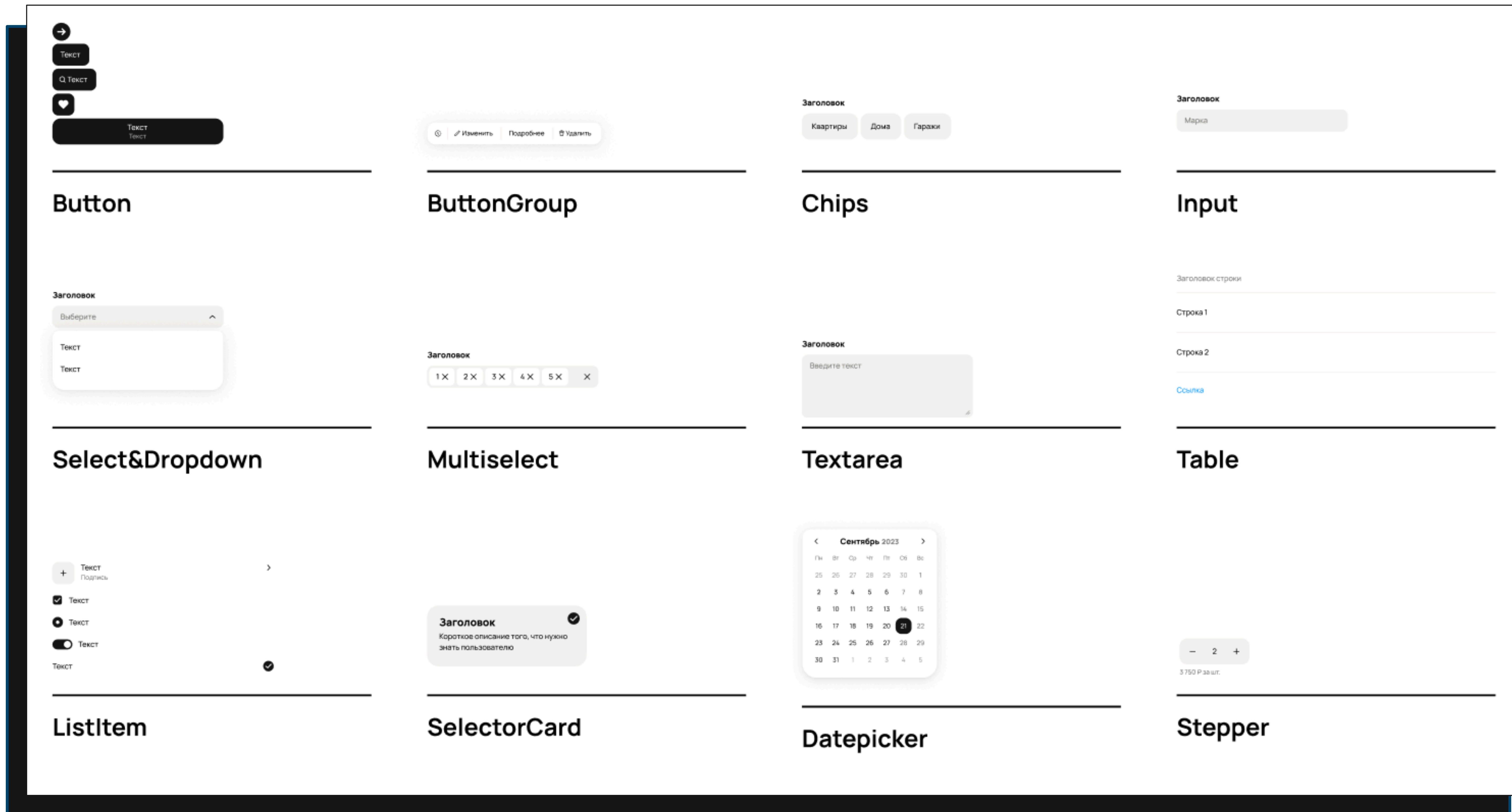
Александр Власюк

Александр Власюк

Старший Android Инженер

- Разрабатываю Авито Кошелёк
- Тг канал: @avvlased

Контекст



Разработка дизайн-системы Авито на Compose

Проблема кастомных лейаутов в Compose

Стандартные лейауты не покрывают все кейсы – необходимо писать кастомные

Останавливает отличие концепции от Custom View и недостаток примеров

1. Как работает layout в Compose

2. Изменение layout отдельного компонента

3. Кастомный Layout

4. Откладывание композиции

5. Кастомный Lazy Layout

Как не стоит изменять layout

```
Image(...)
Text(...)
```

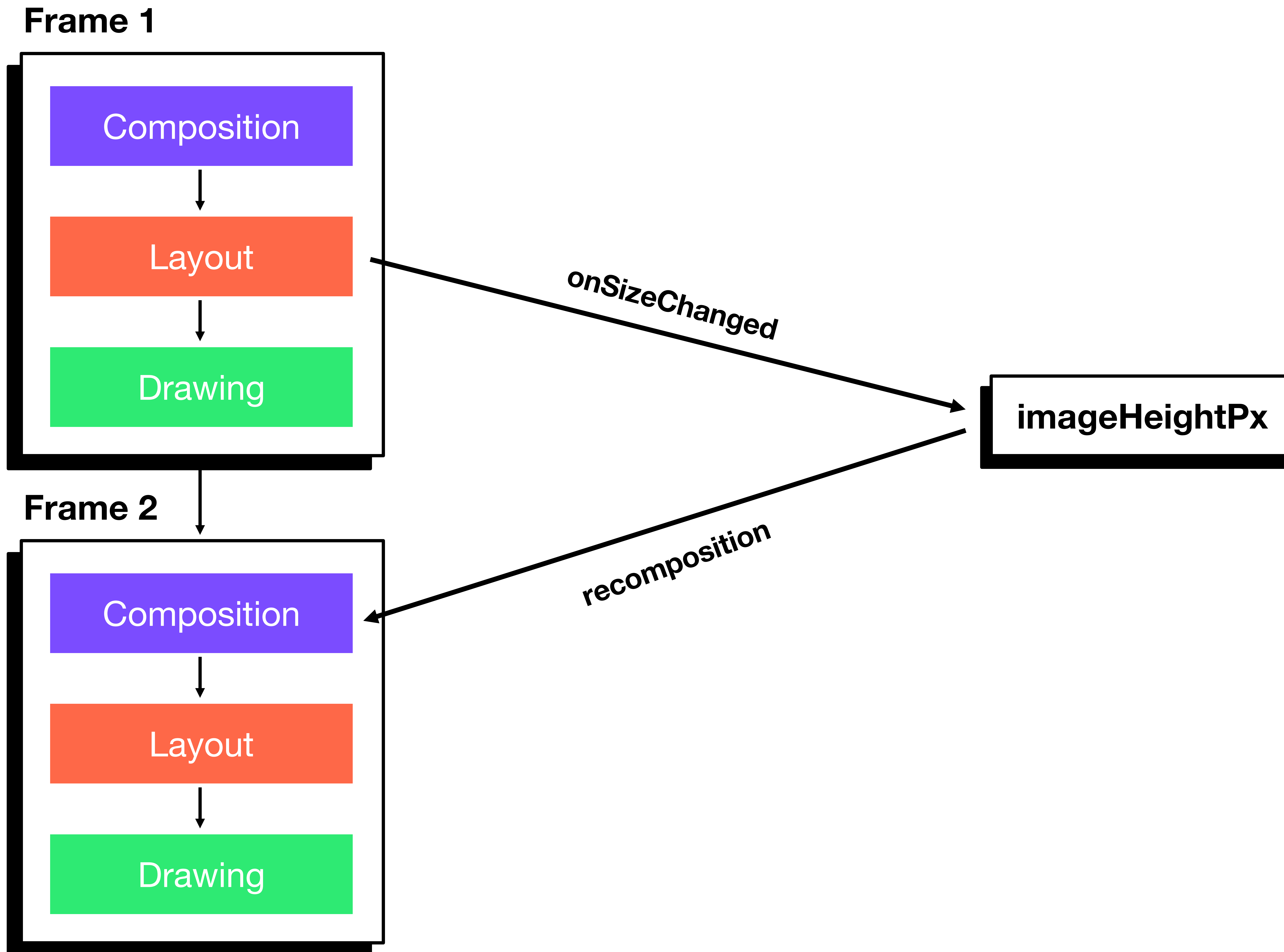
```
Image(
    modifier = Modifier.onSizeChanged { size ->

    }
)

Text()
```

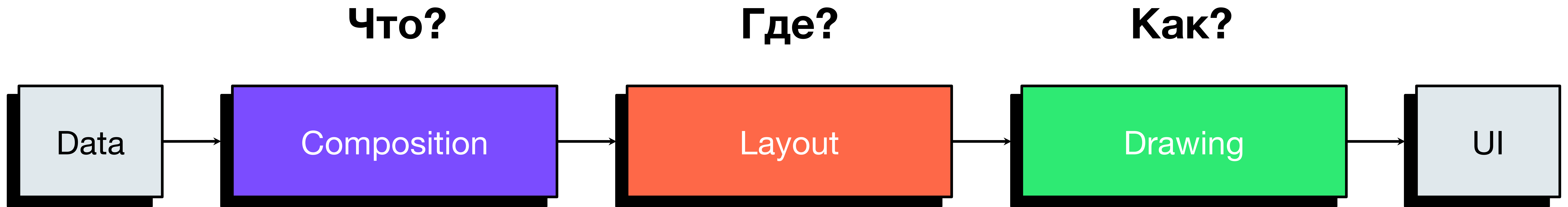
```
var imageHeightPx by remember { mutableStateOf(0) }

Image(
    modifier = Modifier.onSizeChanged { size ->
        imageHeightPx = size.height
    }
)

Text()
```

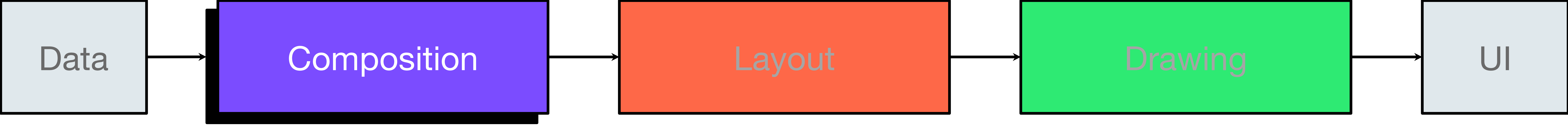
```kotlin
var imageHeightPx by remember { mutableStateOf(0) }

Image(
    modifier = Modifier.onSizeChanged { size ->
        imageHeightPx = size.height
    }
)

Text(
    modifier = Modifier.offset(
        y = imageHeightPx.toDp(),
    )
)
```

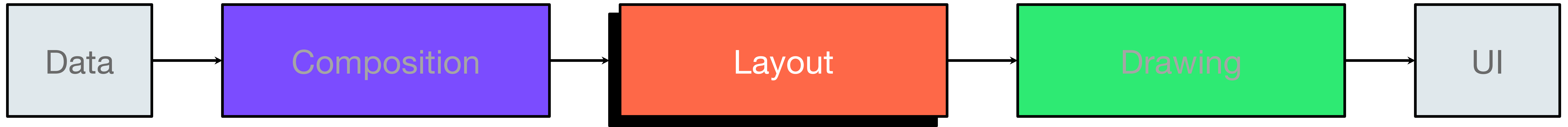
```
var imageHeightPx by remember { mutableStateOf(0) }

Image(
    modifier = Modifier.onSizeChanged { size ->
        imageHeightPx = size.height
    }
)

Text(
    modifier = Modifier.offset(
         y = imageHeightPx.toDp(),
    )
)
```

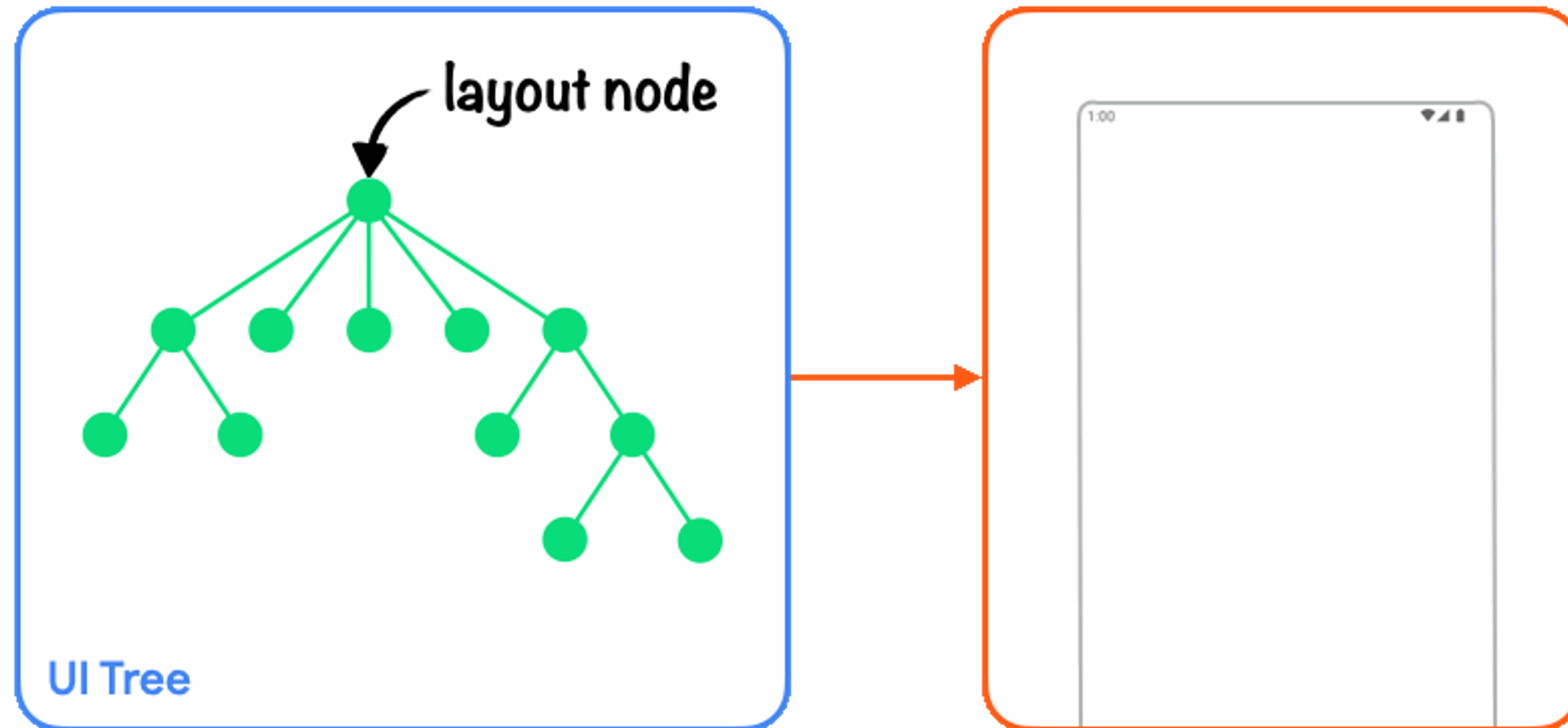
Такой лейаут отрисуется за 2 фрейма!


Frame 1
Composition
Layout
Drawing
Frame 2
Composition
Layout
Drawing
onSizeChanged
imageHeightPx
recomposition

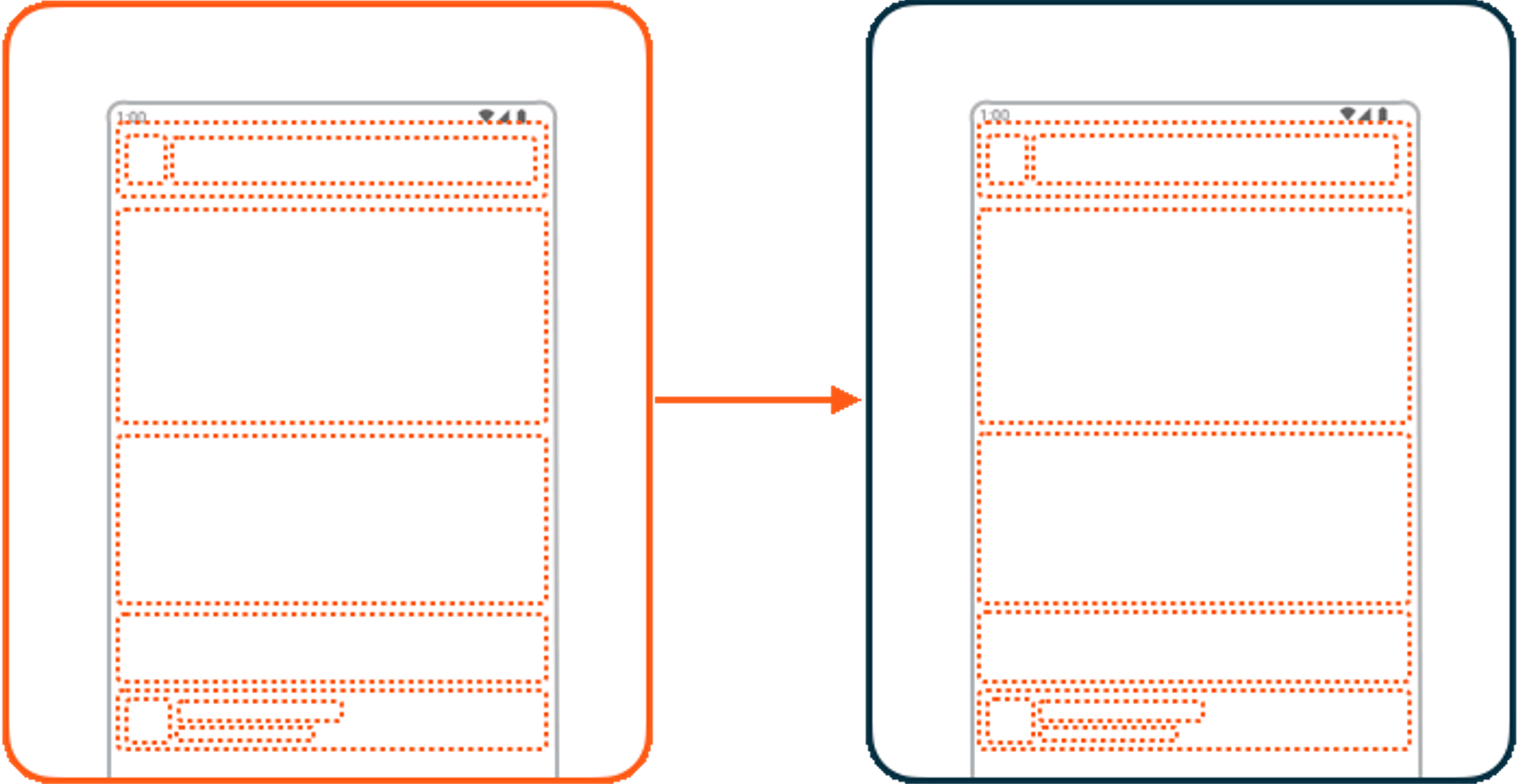

Что?
Где?
Как?
Data
Composition
Layout
Drawing
UI

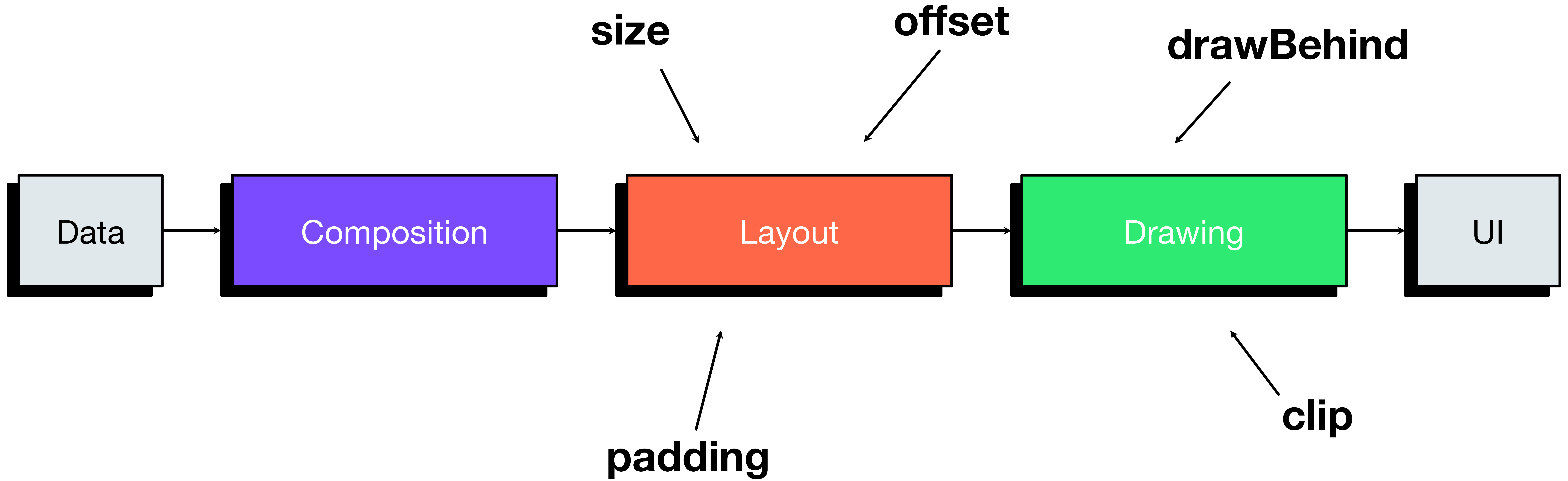

Data
Composition
Layout
Drawing
UI


Data
Composition
Layout
Drawing
UI

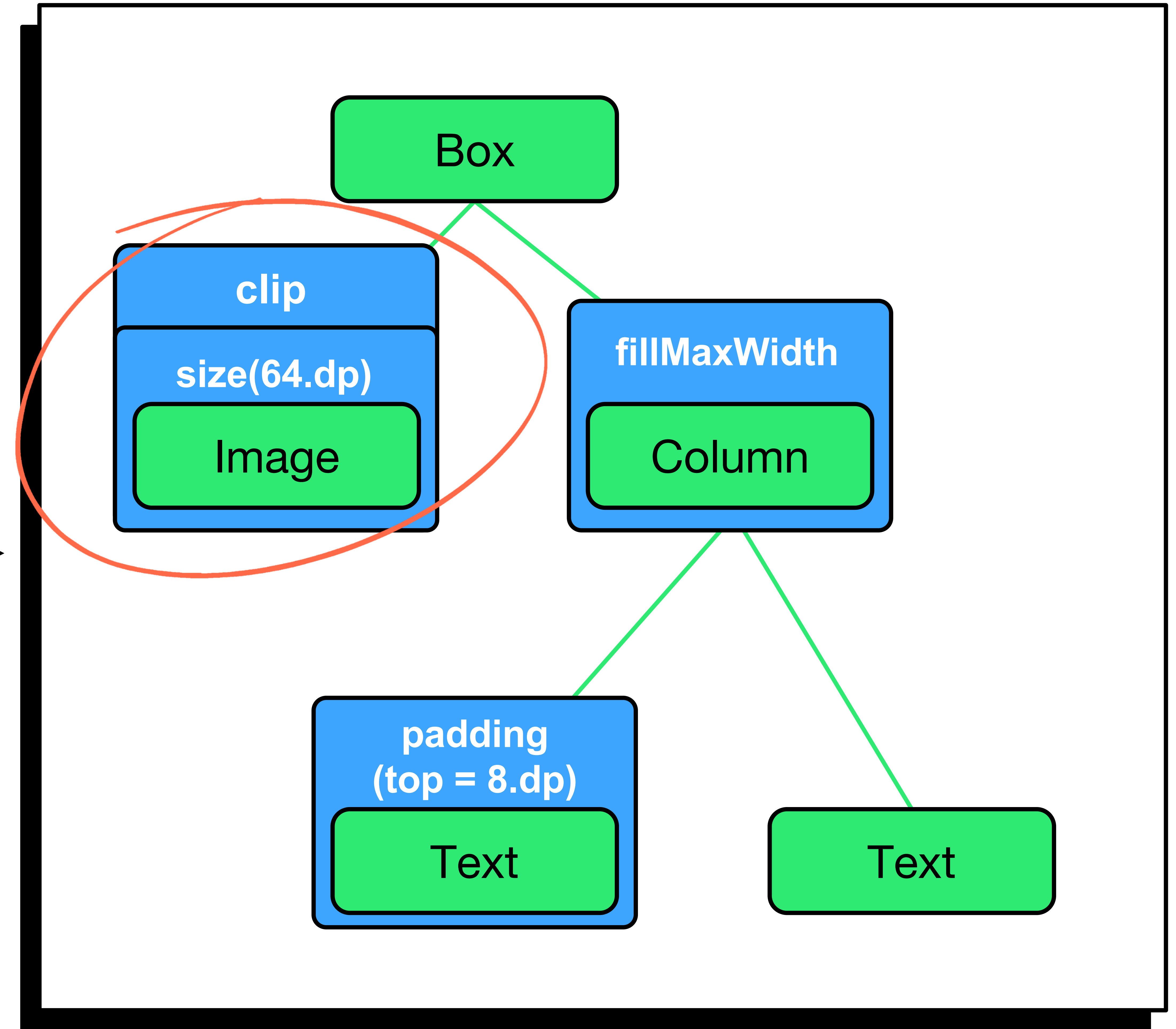

layout node
UI Tree

Data
Composition
Layout
Drawing
UI

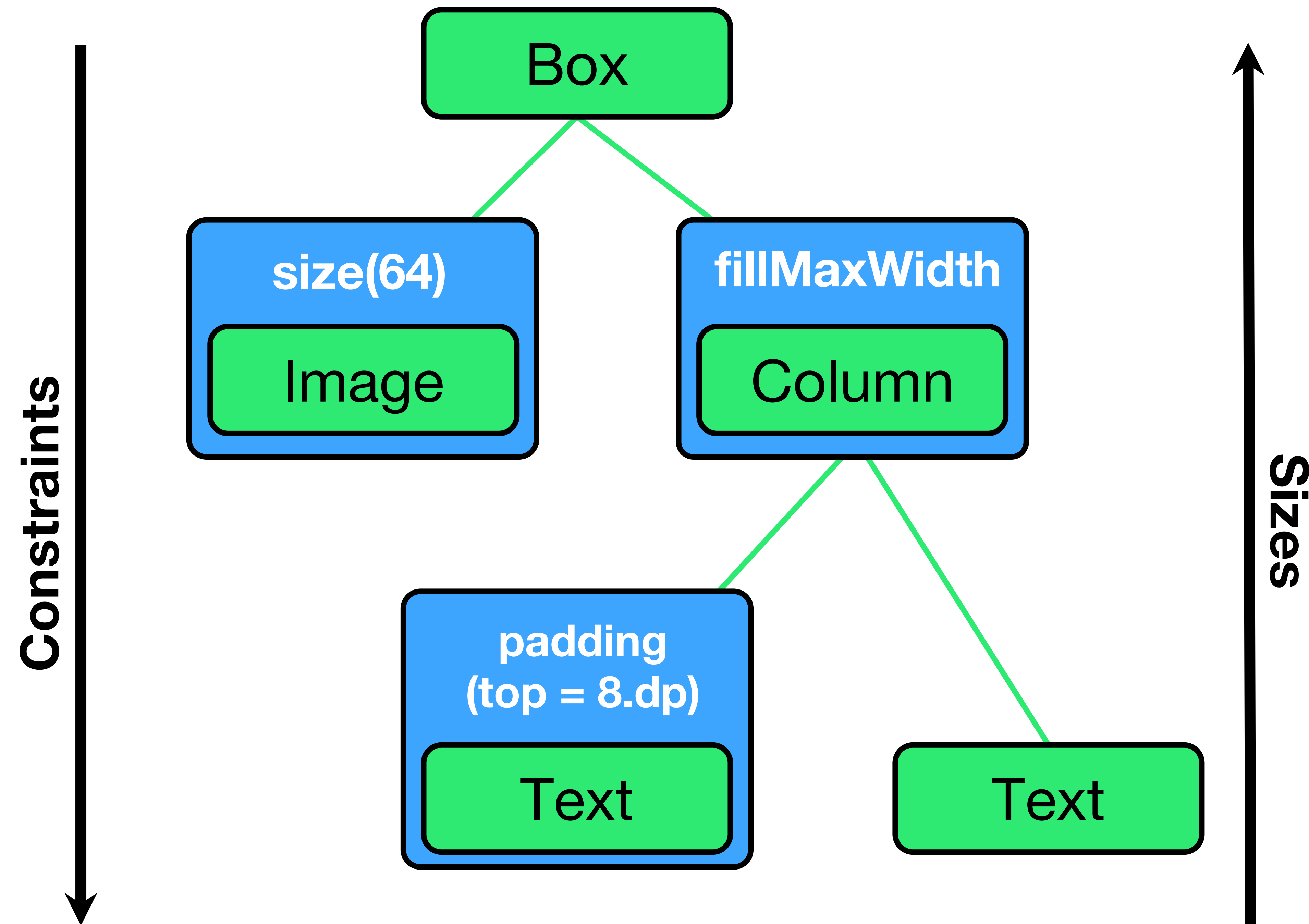



size
offset
drawBehind
Data
Composition
Layout
Drawing
UI
padding
clip

```kotlin
Box {
    Image(
        Modifier
          .clip()
           .size(64.dp),
    )
    Column(
        Modifier.fillMaxWidth(),
    ) {
        Text(
            "...",
            Modifier
                .padding(top = 8.dp)
        )
        Text("...")
    }
}
```

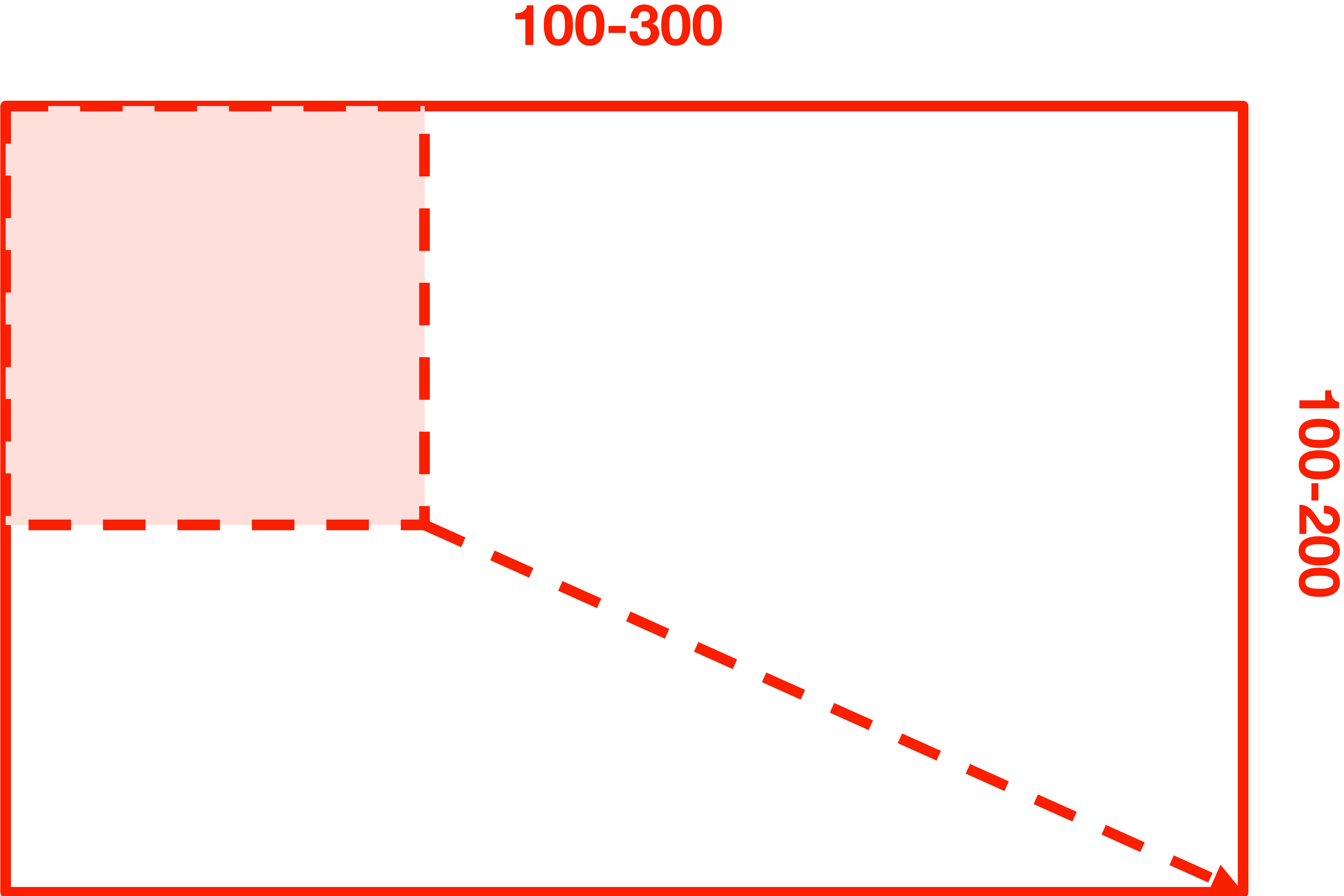


Constraints - границы размера LayoutNode

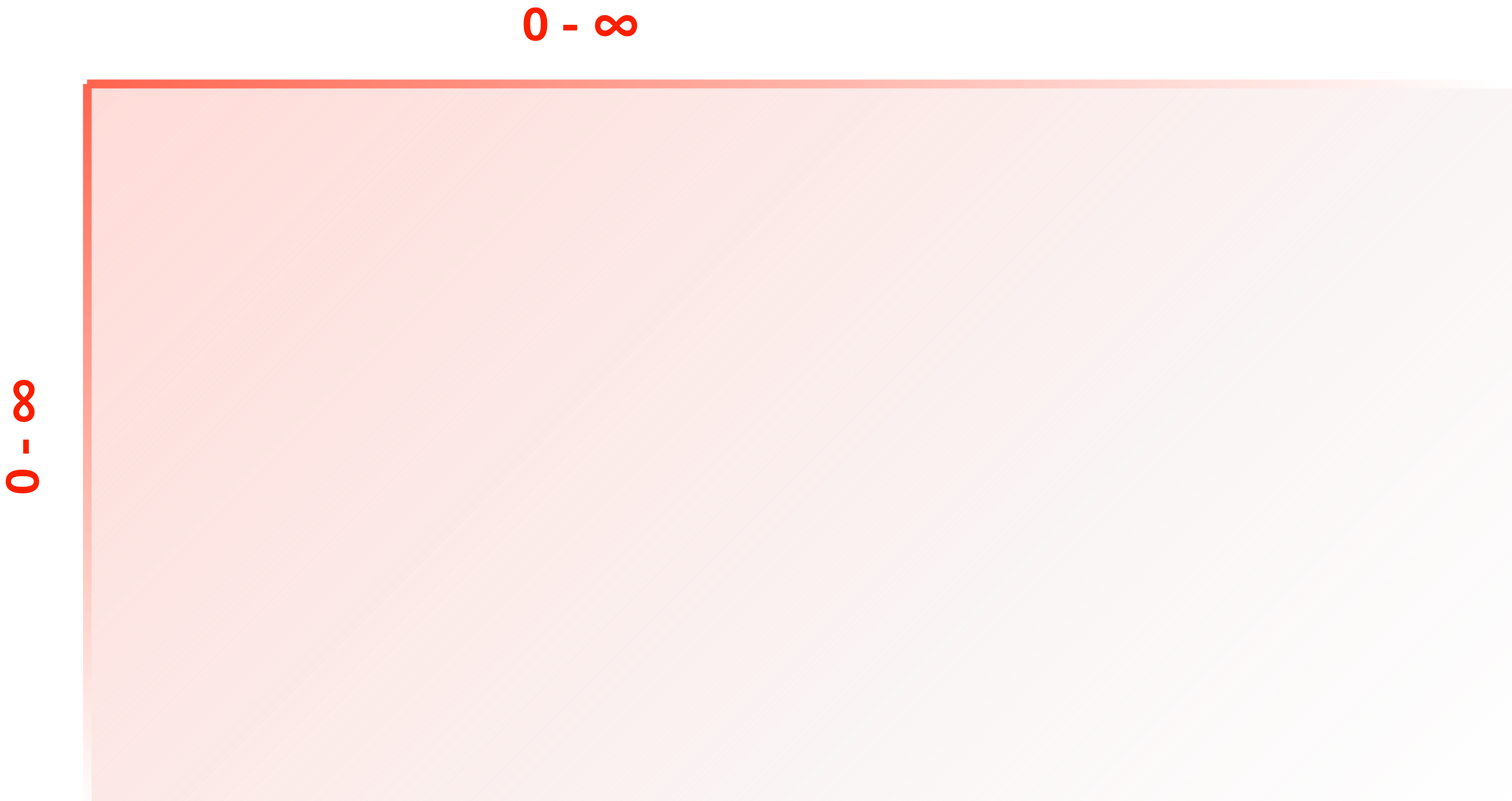


Constraints **передаются вниз по дереву**, от родительских к дочерним нодам

Bound Constraints

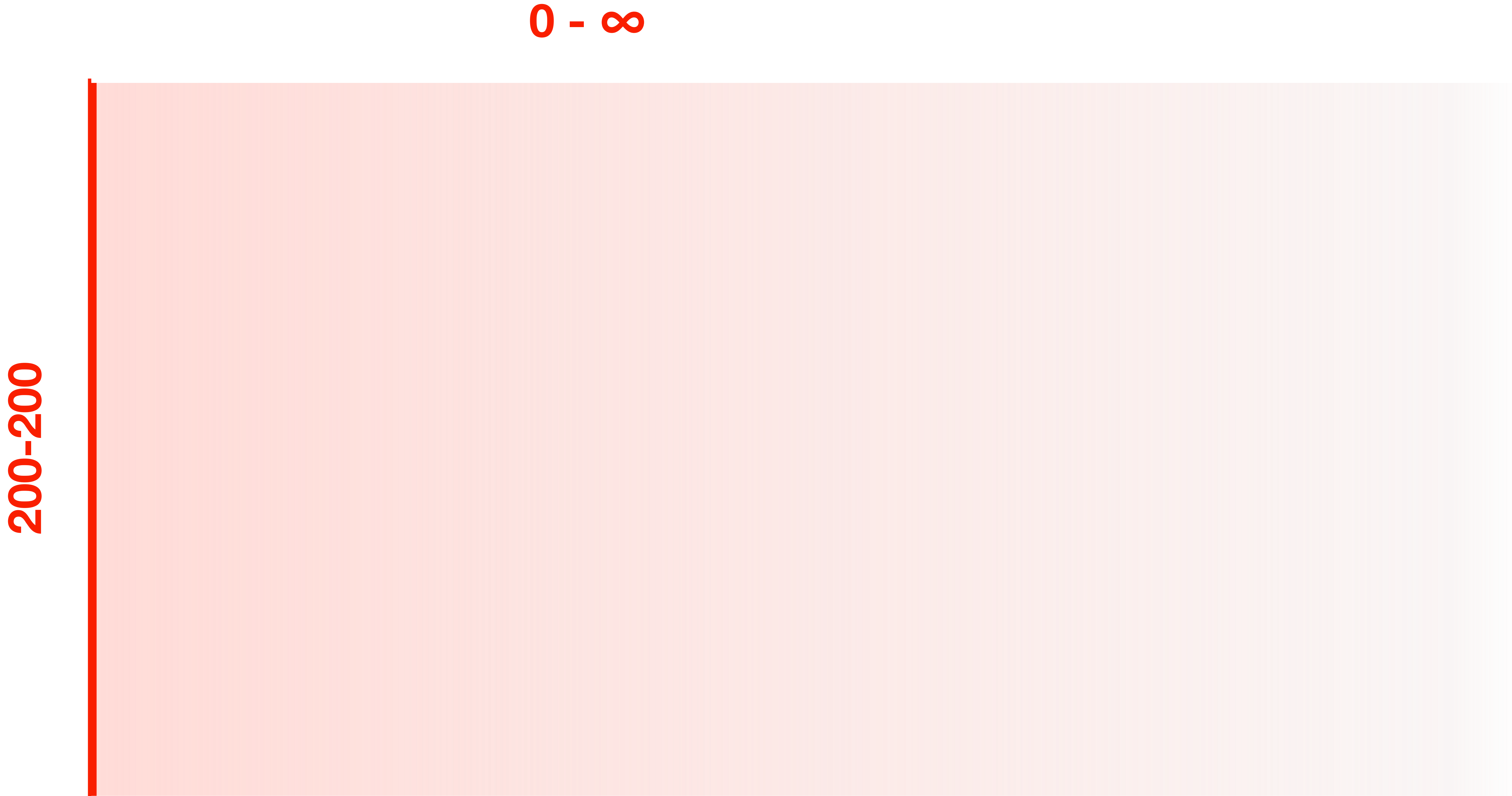


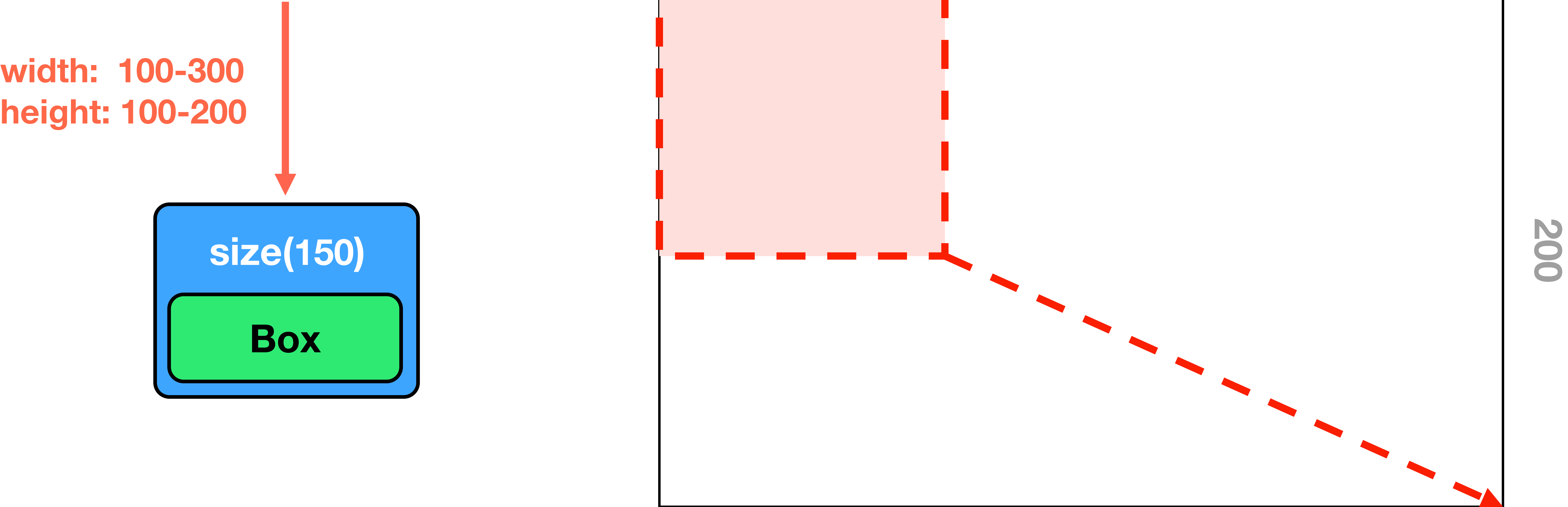
Unbound Constraints

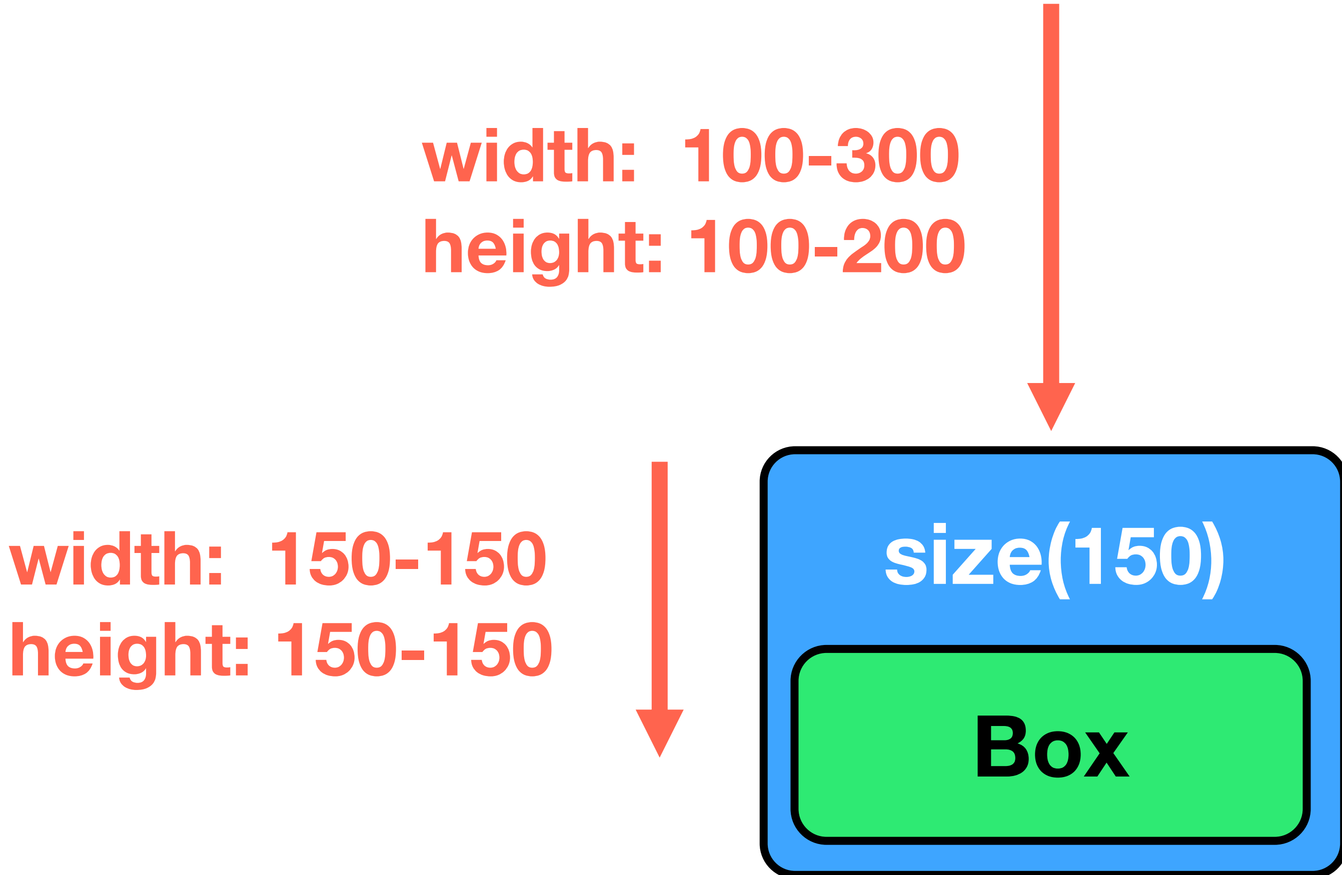


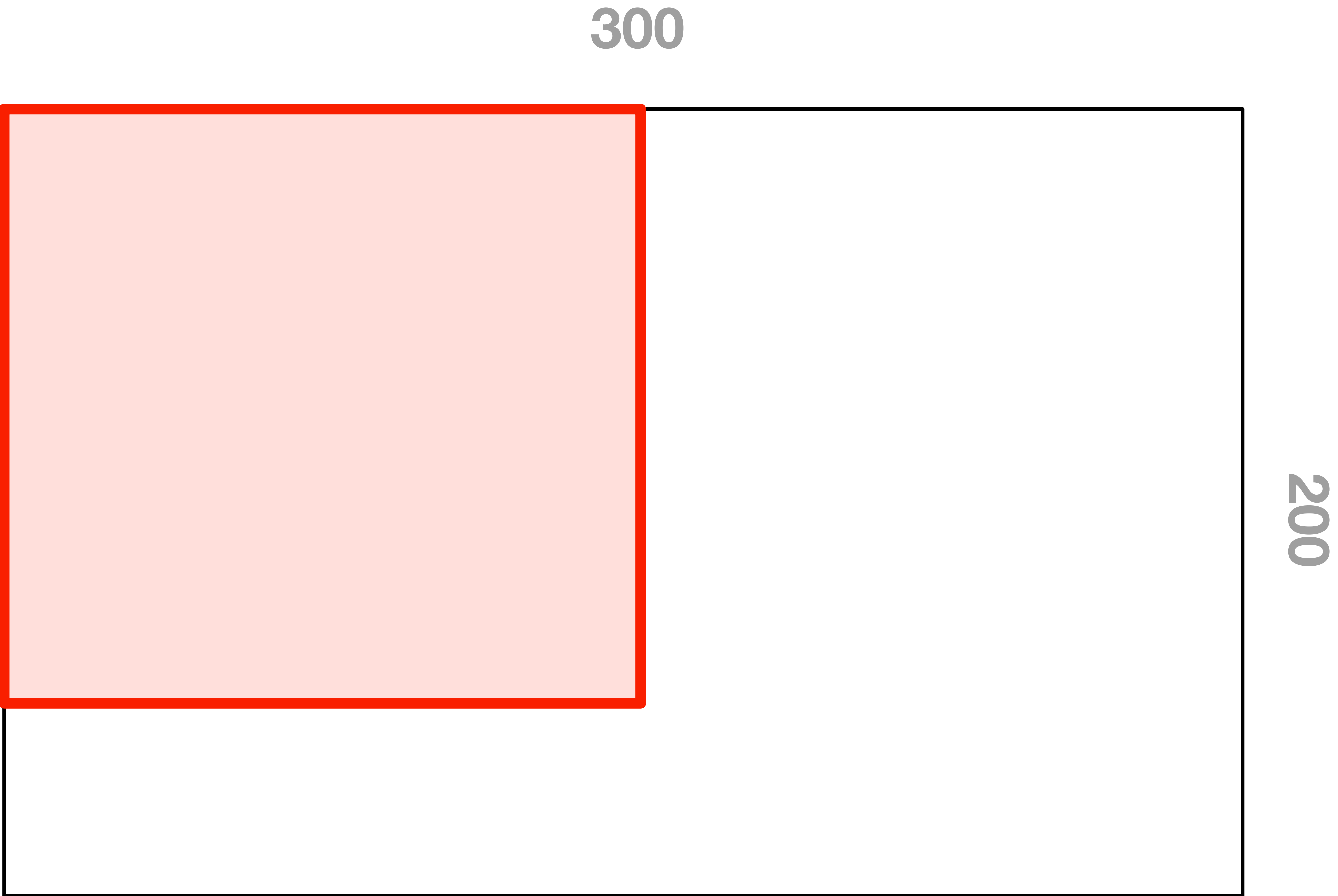
Fixed Constraints

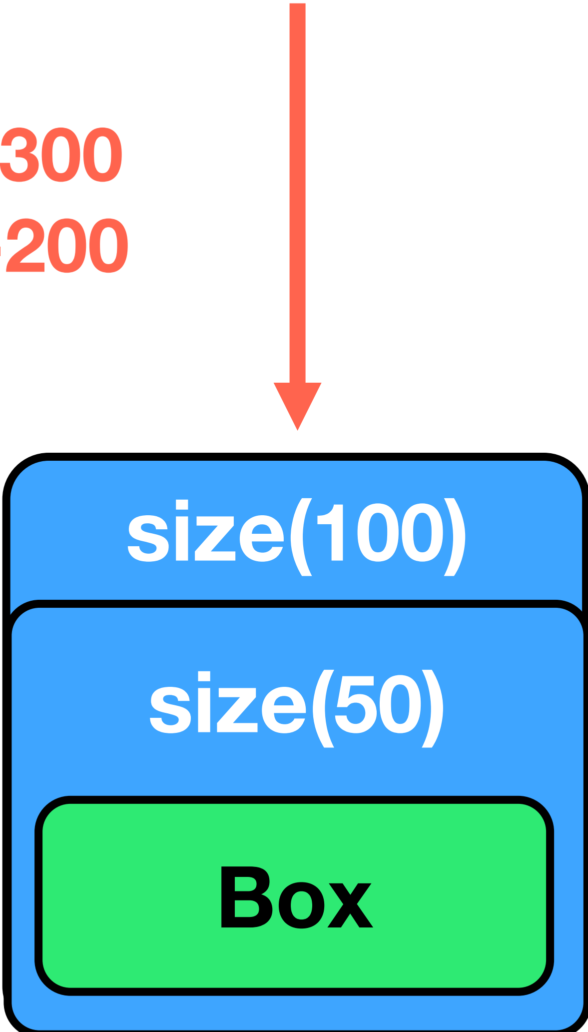
Mixed Constraints

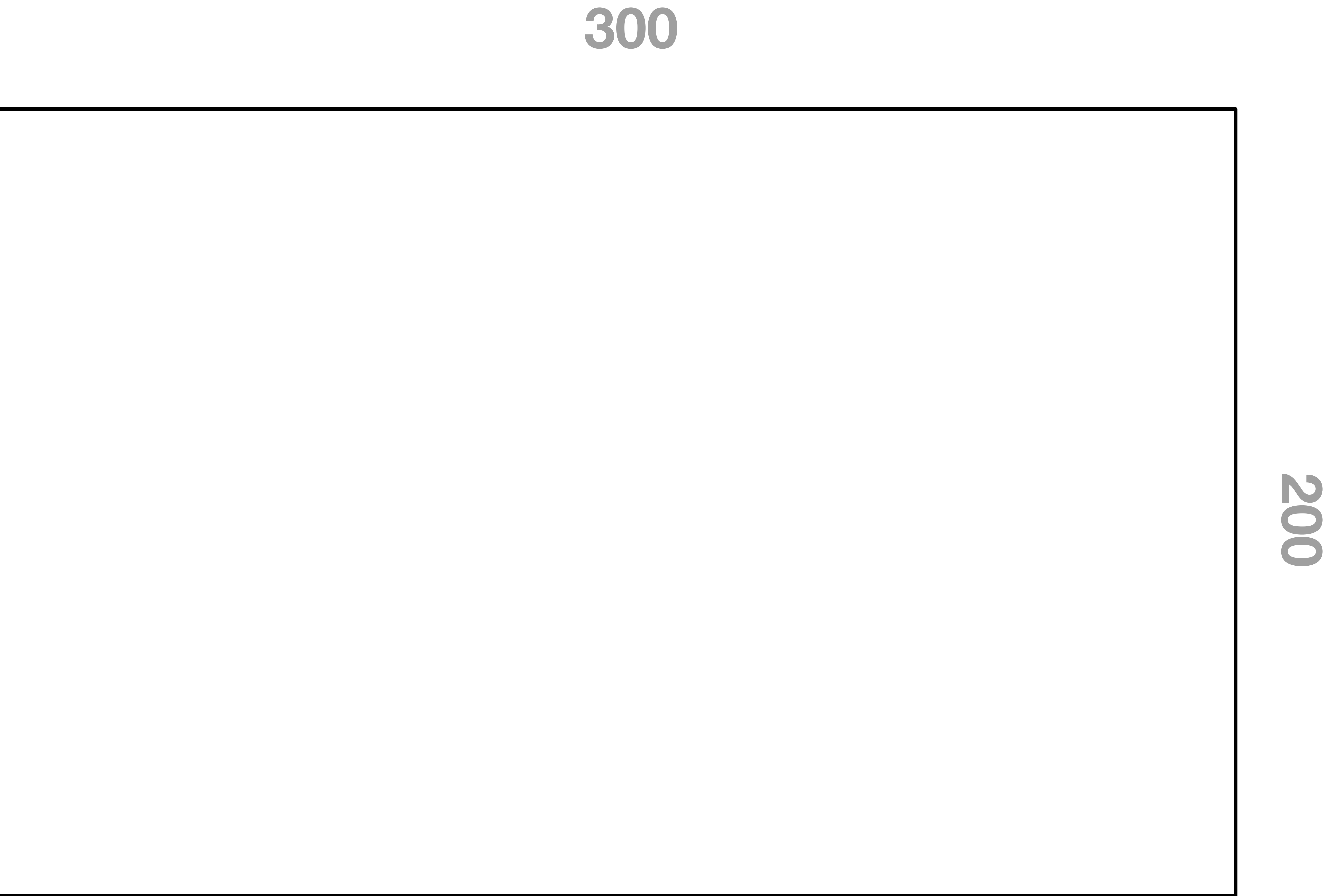


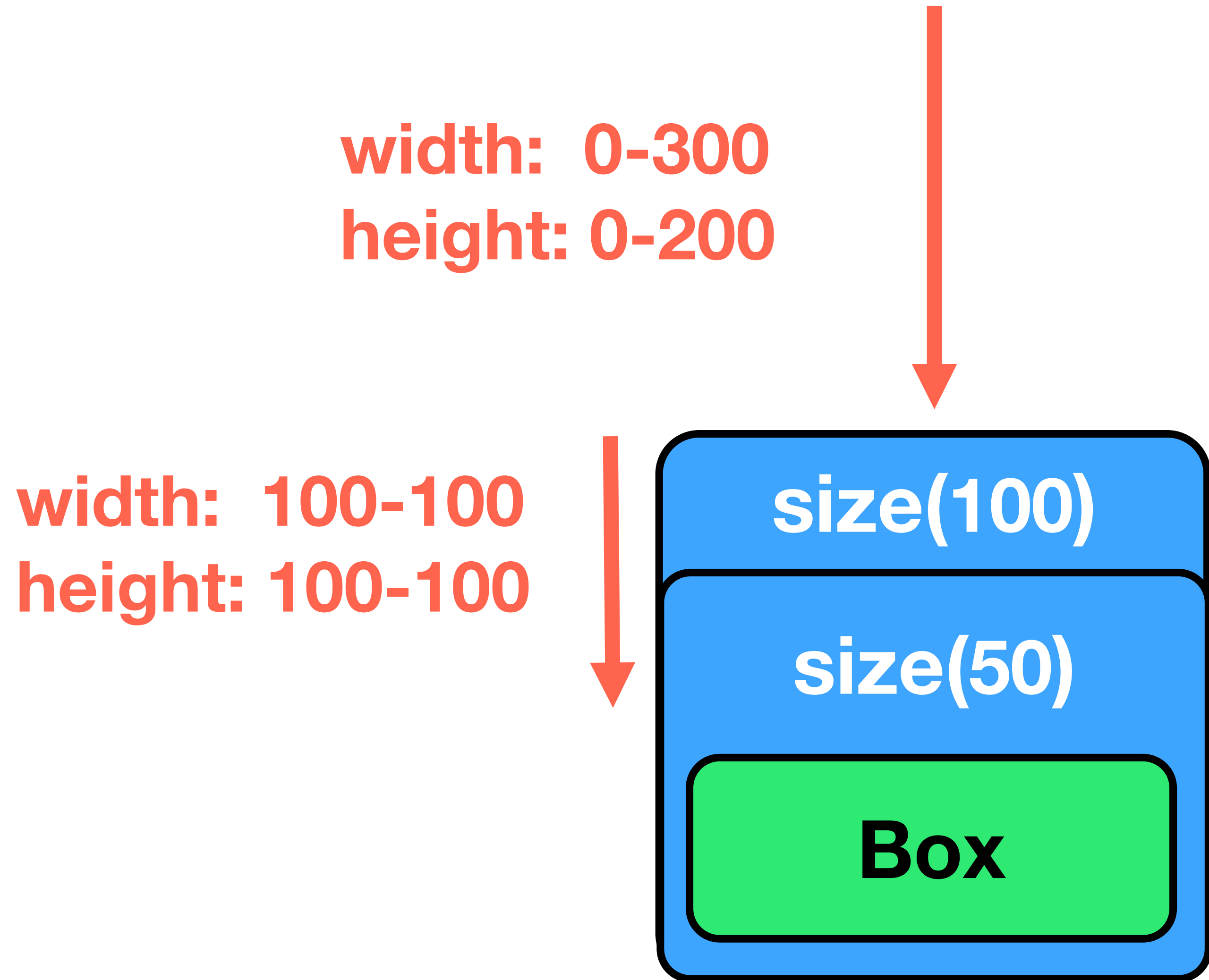

width: 100-300
height: 100-200
size(150)
Box
300
200

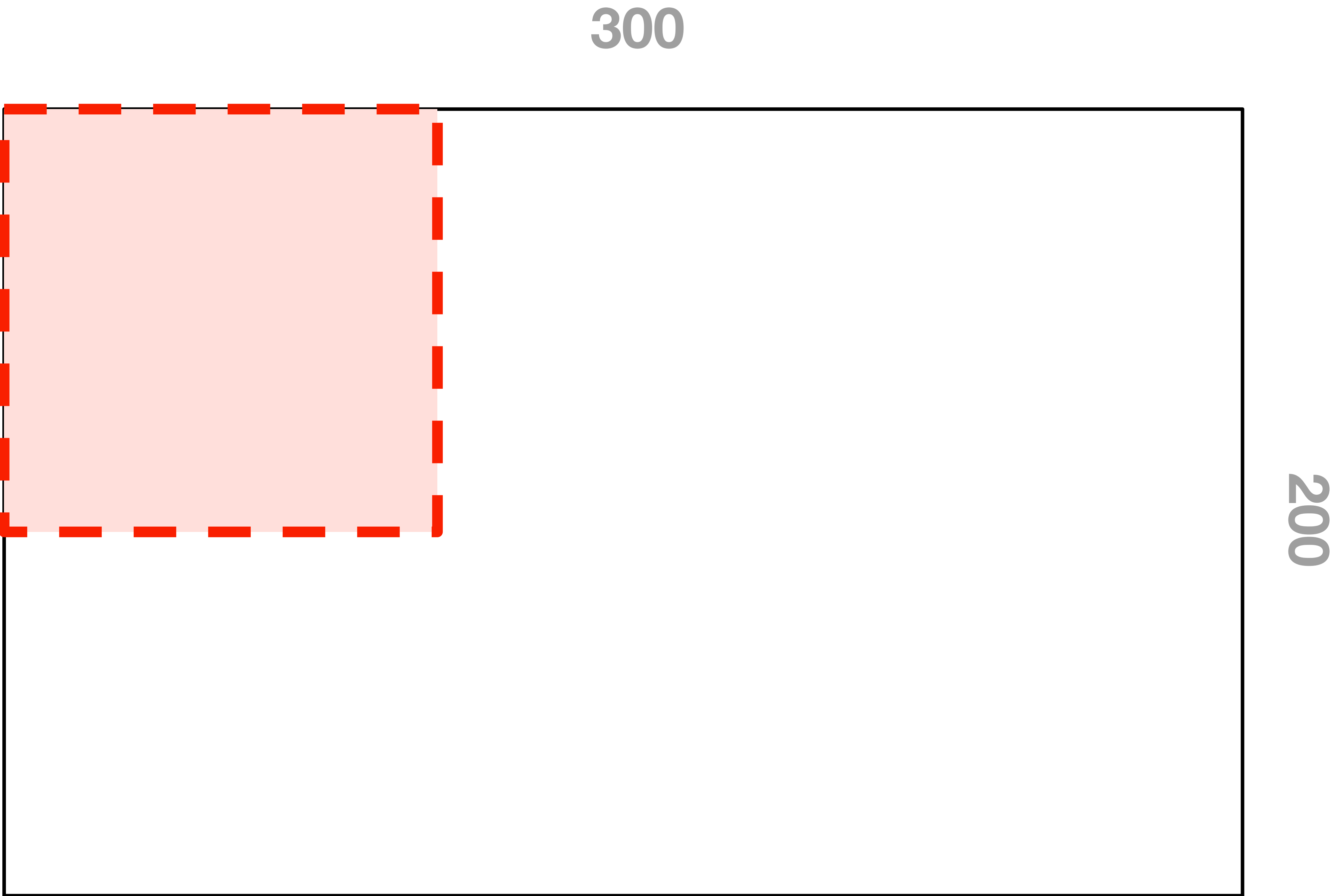

width: 100-300
height: 100-200
width: 150-150
height: 150-150
size(150)
Box

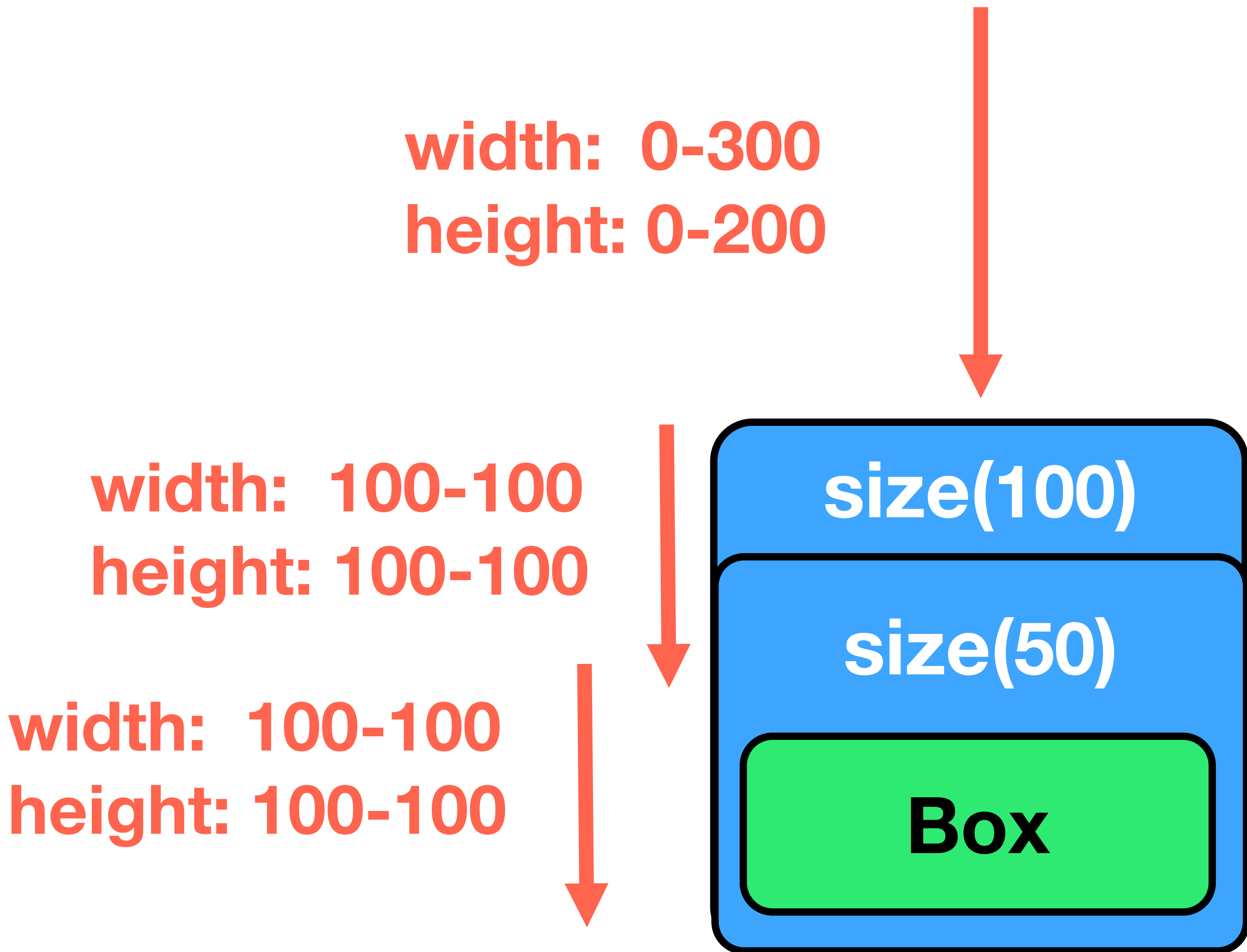

300
200

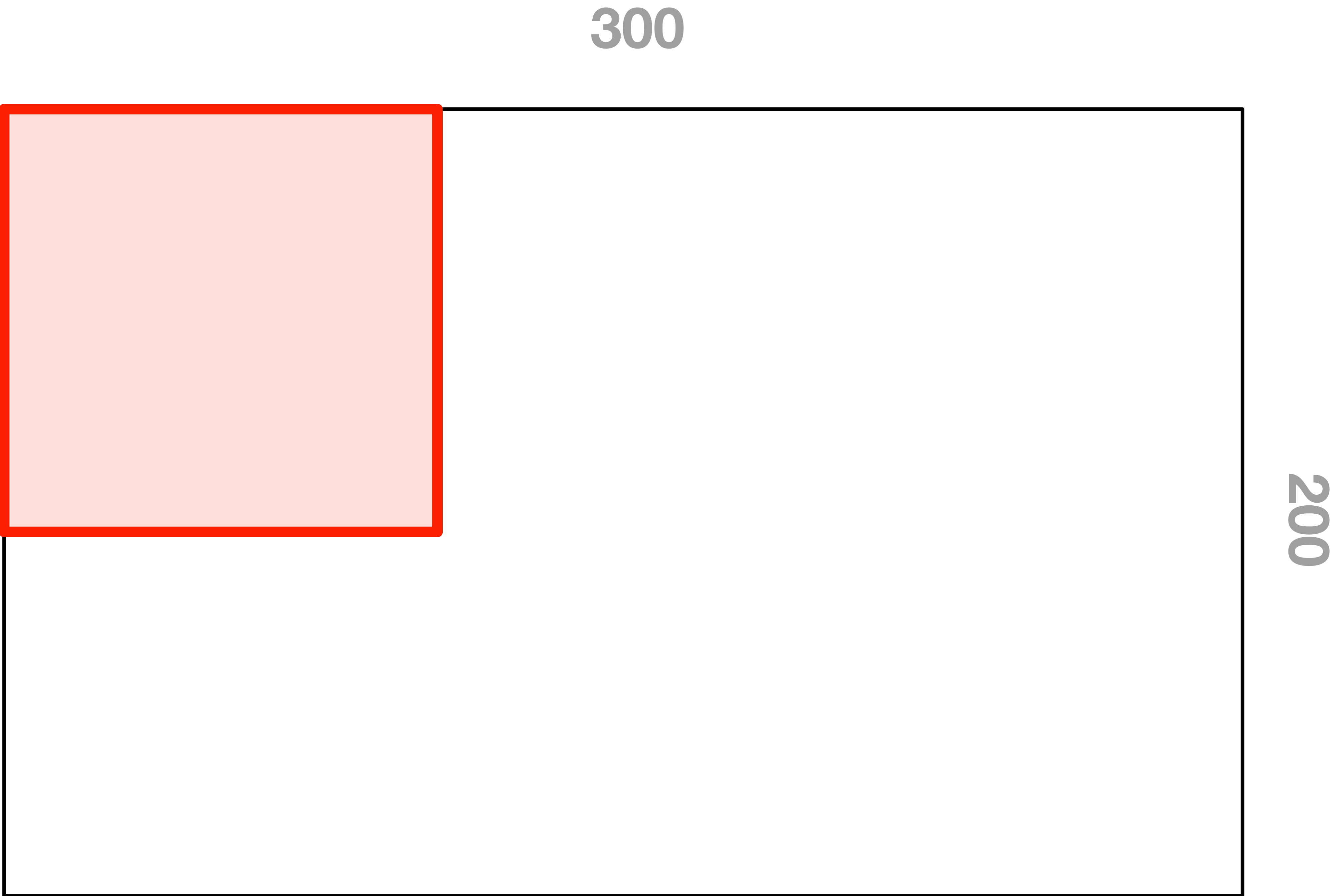

width: 0-300
height: 0-200
size(100)
size(50)
Box


300
200


width: 0-300
height: 0-200
width: 100-100
height: 100-100
size(100)
size(50)
Box


300
200


width: 0-300
height: 0-200
width: 100-100
height: 100-100
width: 100-100
height: 100-100
size(100)
size(50)
Box


300
200

Каждая LayoutNode обходится всего **один раз***

***Но есть исключения**

Modifier.layout

```
fun Modifier.width(width: Dp) =
```

```
fun Modifier.width(width: Dp) = this.then(
    Modifier.layout { measurable: Measurable, constraints: Constraints ->







    }
)
```

```
fun Modifier.width(width: Dp) = this.then(
    Modifier.layout { measurable: Measurable, constraints: Constraints ->









     }
)
```

```
fun Modifier.width(width: Dp) = this.then(
    Modifier.layout { measurable: Measurable, constraints: Constraints ->
        val newConstraints = constraints.copy(
            minWidth = width,
            maxWidth = width,
        )




    }
)
```

```
fun Modifier.width(width: Dp) = this.then(
    Modifier.layout { measurable: Measurable, constraints: Constraints ->
        val newConstraints = constraints.copy(
            minWidth = width,
            maxWidth = width,
        )




    }
)


interface Measurable {
   fun measure(constraints: Constraints): Placeable
}
```

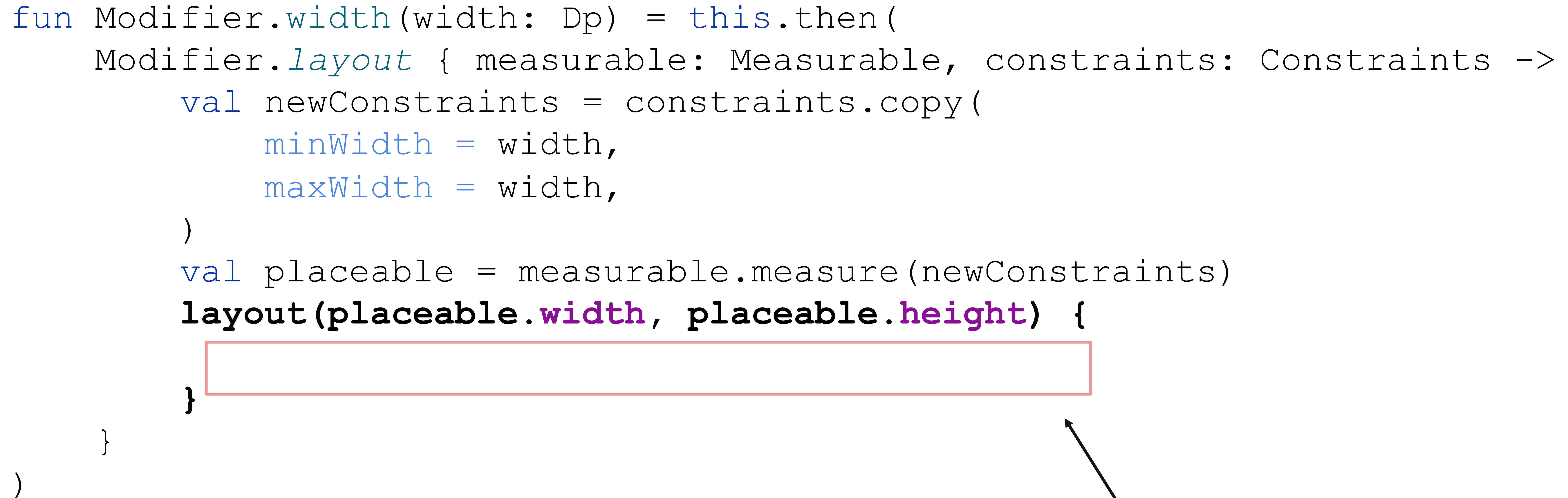
```
fun Modifier.width(width: Dp) = this.then(
    Modifier.layout { measurable: Measurable, constraints: Constraints ->
        val newConstraints = constraints.copy(
            minWidth = width,
            maxWidth = width,
        )
        val placeable = measurable.measure(newConstraints)



    }
)
```

```
abstract class Placeable : Measured {
    var width: Int
    var height: Int
}
```

```kotlin
fun Modifier.width(width: Dp) = this.then(
    Modifier.layout { measurable: Measurable, constraints: Constraints ->
        val newConstraints = constraints.copy(
            minWidth = width,
            maxWidth = width,
        )
        val placeable = measurable.measure(newConstraints)



    }
)
```

```
interface MeasureScope {
    fun layout(
        width: Int,
        height: Int,
        alignmentLines: Map<AlignmentLine, Int> = emptyMap(),
        placementBlock: Placeable.PlacementScope.() -> Unit
    ): MeasureResult
}
```

```
interface MeasureScope {
    fun layout(
        width: Int,
        height: Int,
        alignmentLines: Map<AlignmentLine, Int> = emptyMap(),
        placementBlock: Placeable.PlacementScope.() -> Unit
    ): MeasureResult
}
```


fun Modifier.width(width: Dp) = this.then(
Modifier.layout { measurable: Measurable, constraints: Constraints ->
val newConstraints = constraints.copy(
minWidth = width,
maxWidth = width,
)
val placeable = measurable.measure(newConstraints)
layout(placeable.width, placeable.height) {
}
}
)
PlacementScope

```kotlin
fun Modifier.width(width: Dp) = this.then(
    Modifier.layout { measurable: Measurable, constraints: Constraints ->
        val newConstraints = constraints.copy(
            minWidth = width,
            maxWidth = width,
        )
        val placeable = measurable.measure(newConstraints)
        layout(placeable.width, placeable.height) {
          placeable.place(0, 0)
        }
    }
)
```

```kotlin
fun Modifier.width(width: Dp) = this.then(
    Modifier.layout { measurable: Measurable, constraints: Constraints ->
        val newConstraints = constraints.copy(
            minWidth = width,
            maxWidth = width,
        )
        val placeable = measurable.measure(newConstraints)
        layout(placeable.width, placeable.height) {
          placeable.place(0, 0)
        }
    }
)
```

1. **Измеряем дочерние ноды**
2. Определяем собственный размер
3. Располагаем дочерние ноды

```
fun Modifier.width(width: Dp) = this.then(
    Modifier.layout { measurable: Measurable, constraints: Constraints ->
        val newConstraints = constraints.copy(
            minWidth = width,
            maxWidth = width,
        )
        val placeable = measurable.measure(newConstraints)
        layout(placeable.width, placeable.height) {
          placeable.place(0, 0)
        }
    }
)
```

1. Измеряем дочерние ноды
2. **Определяем собственный размер**
3. Располагаем дочерние ноды

```
fun Modifier.width(width: Dp) = this.then(
    Modifier.layout { measurable: Measurable, constraints: Constraints ->
        val newConstraints = constraints.copy(
            minWidth = width,
            maxWidth = width,
        )
        val placeable = measurable.measure(newConstraints)
        layout(placeable.width, placeable.height) {
          placeable.place(0, 0)
        }
    }
)
```

1. Измеряем дочерние ноды
2. Определяем собственный размер
3. **Располагаем дочерние ноды**

```
class LayoutModifierImpl
```

```
class LayoutModifierImpl(
   var measureBlock: MeasureScope.(Measurable, Constraints) -> MeasureResult
) : LayoutModifierNode
```

```
class LayoutModifierImpl(
   var measureBlock: MeasureScope.(Measurable, Constraints) -> MeasureResult
) : LayoutModifierNode {

   override fun MeasureScope.measure(
       measurable: Measurable,
       constraints: Constraints
   ) = measureBlock(measurable, constraints)

}
```

Modifier, который влияет и на layout, и на drawing фазы:

```
class TextStringSimpleNode(...) : LayoutModifierNode, DrawModifierNode {

    override fun MeasureScope.measure(
        measurable: Measurable,
        constraints: Constraints
    ) { ... }


    override fun ContentDrawScope.draw() { ... }

}
```

```
Image(...)
Text(...)
```

@Composable Layout

```
@Composable
fun Column() {
   Layout(...)
}
```

```
@Composable
fun Column() {
   Layout(
       content = {
           Image()
           Text()
       },



   )
}
```

```
@Composable
fun Column() {
    Layout(
        content = {
            Image()
            Text()
        },
        measurePolicy = { measurables: List<Measurable>, constraints: Constraints ->
            ...
            val placeables = measurables.map {
                it.measure(constraints)
            }
            ...
            layout(width = width, height = height) {
                placeables.forEach { placeable ->
                    placeable.placeRelative(x, y)
                }
            }
        },
    )
}
```

```
@Composable
fun Column() {
   Layout(
       content = {
           Image()
           Text()
       },
       measurePolicy = { measurables: List<Measurable>, constraints: Constraints ->
           ...
           val placeables = measurables.map {
               it.measure(constraints)
           }
           ...
           layout(width = width, height = height) {
               placeables.forEach { placeable ->
                   placeable.placeRelative(x, y)
               }
           }
       },
   )
}
```

Идентифицируем LayoutNode

```
Layout(
    content = {
        Image()
        Text()
    }
) { measurables, constraints ->


}
```

```
Layout(
    content = {
        Image(Modifier.layoutId(ImageId))
        Text(Modifier.layoutId(TextId))
    }
) { measurables, constraints ->


}
```

```
Layout(
   content = {
       Image(Modifier.layoutId(ImageId))
       Text(Modifier.layoutId(TextId))
   }
) { measurables, constraints ->
   val imageMeasurable = measurables.firstOrNull { it.layoutId == ImageId }
   ...
}
```

Получаем данные от LayoutNode

```
interface Measurable {

   val parentData: Any?

   ...
}
```

```
class ParentDataModifierImpl() : ParentDataModifier {
    override fun Density.modifyParentData(parentData: Any?) { ... }
}
```

```
class ParentDataModifierImpl() : ParentDataModifier {

    override fun Density.modifyParentData(parentData: Any?) { ... }

}
```

```
Layout(
    content = content,
) { measurables, constraints ->

     measurables.forEach {
         it.parentData?.let { ... }
     }

}
```

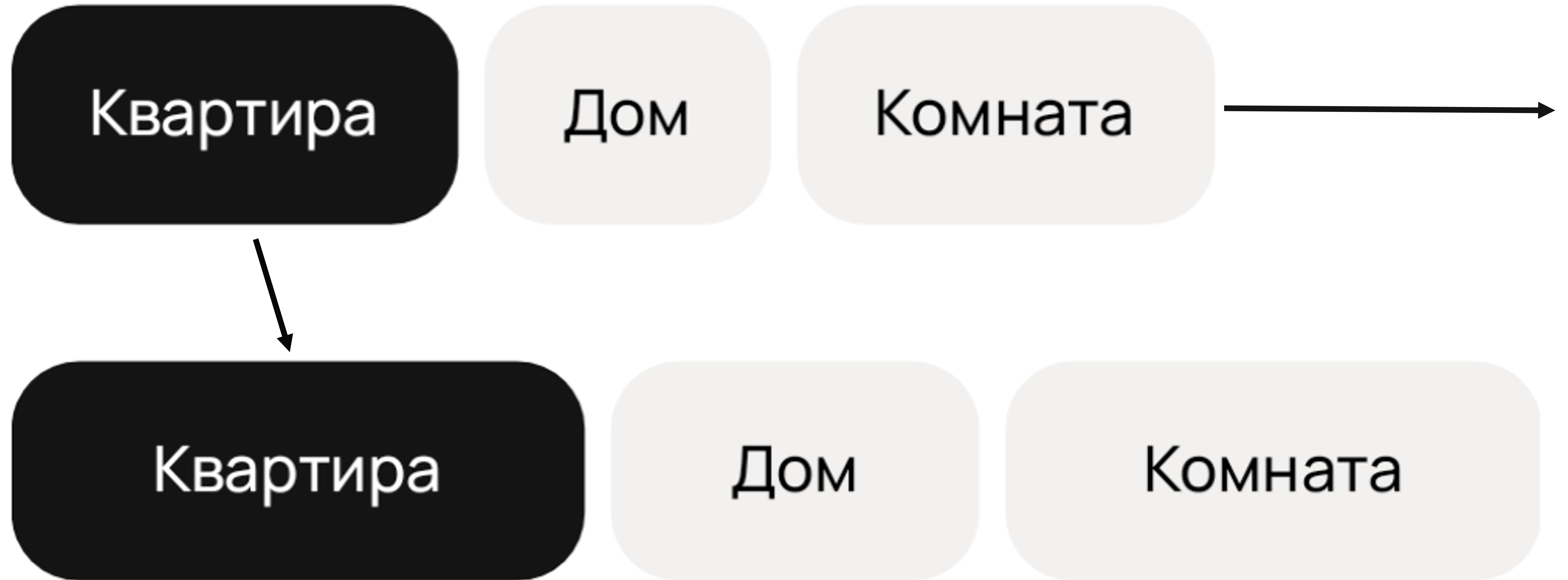
```
val Measurable.layoutId: Any?
    get() = (parentData as? LayoutIdParentData)?.layoutId
```

Квартира

Дом

Комната

Офисно


Квартира
Дом
Комната
Квартира
Дом
Комната

```
Layout { measurables, constraints ->
    var placeables = measurables.map { it.measure(constraints) }
    if (placeables.sumOf { it.width } < constraints.maxWidth) {
        placeables = measurables.mapIndexed { i, measurable ->
            val fixedWidthConstraints = calculateConstraints(...)
            measurable.measure(fixedWidthConstraints)
        }
    }
    ...
}
```

```
Layout { measurables, constraints ->
    var placeables = measurables.map { it.measure(constraints) }
    if (placeables.sumOf { it.width } < constraints.maxWidth) {
        placeables = measurables.mapIndexed { i, measurable ->
            val fixedWidthConstraints = calculateConstraints(...)
            measurable.measure(fixedWidthConstraints)
        }
    }
    ...
}
```

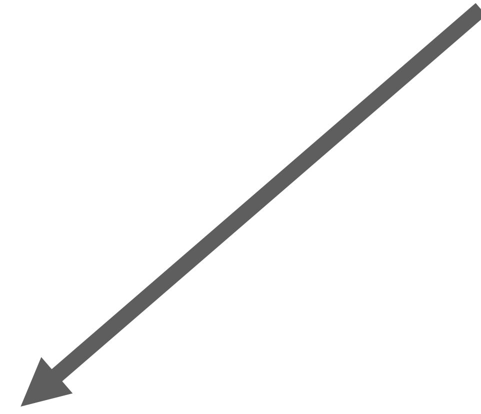
```
Layout { measurables, constraints ->
    var placeables = measurables.map { it.measure(constraints) }
    if (placeables.sumOf { it.width } < constraints.maxWidth) {
        placeables = measurables.mapIndexed { i, measurable ->
            val fixedWidthConstraints = calculateConstraints(...)
            measurable.measure(fixedWidthConstraints)
        }
    }
    ...
}
```

```
Layout { measurables, constraints ->
   var placeables = measurables.map { it.measure(constraints) }
   if (placeables.sumOf { it.width } < constraints.maxWidth) {
       placeables = measurables.mapIndexed { i, measurable ->
           val fixedWidthConstraints = calculateConstraints(...)
           measurable.measure(fixedWidthConstraints)
       }
   }
   ...
}
```

IllegalStateException: measure() may not be called multiple times on the same Measurable

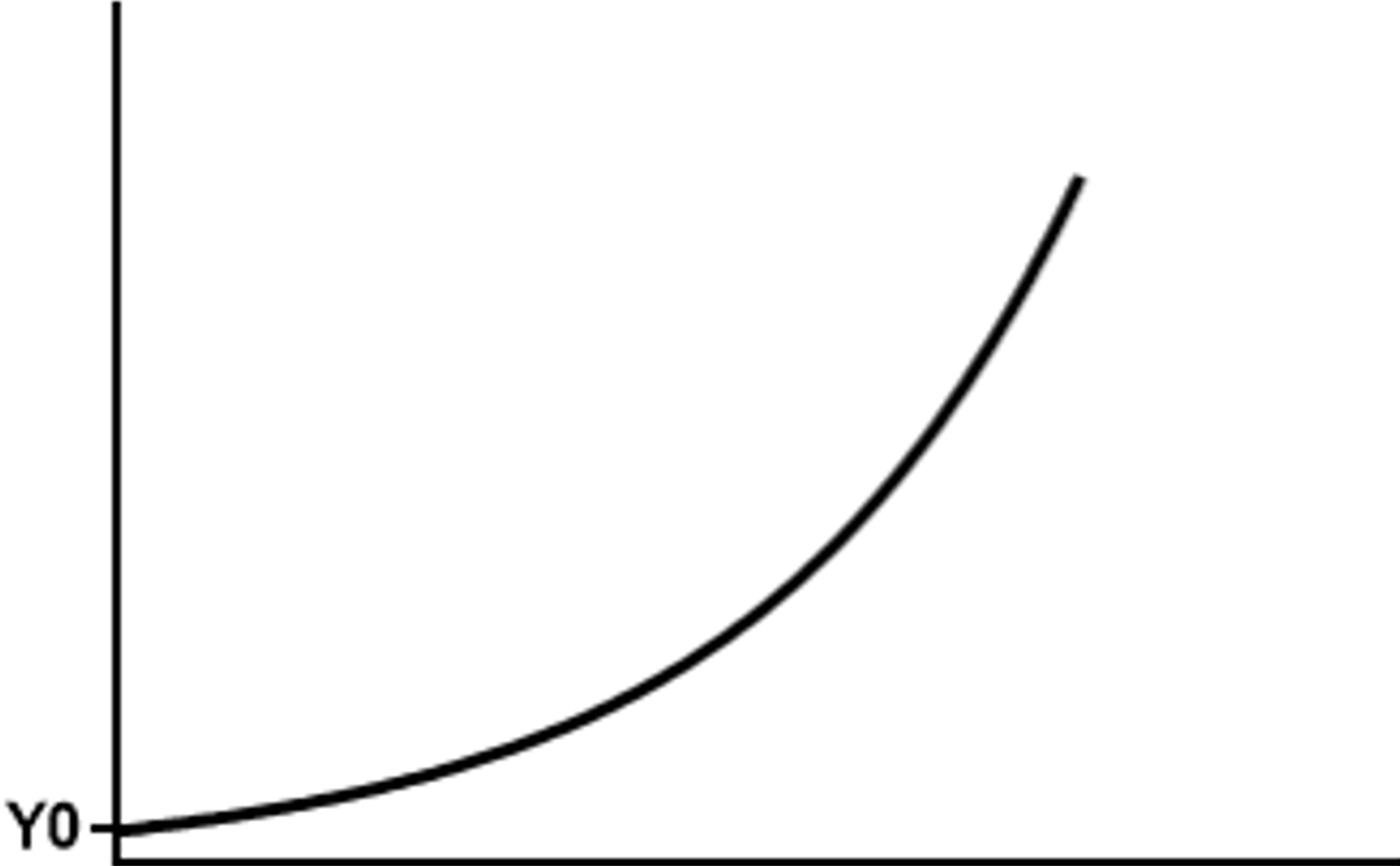
Каждая LayoutNode обходится всего **один раз***

***Но есть исключения**



```
MultiMeasureLayout { measurables, constraints ->
    var placeables = ... // first measure
    if (placeables.sumOf { it.width } < constraints.maxWidth) {
        // second measure
    }
    ...
}
```

```
@Composable
@Deprecated("...This API should be avoided whenever possible.")
fun MultiMeasureLayout(...)
```

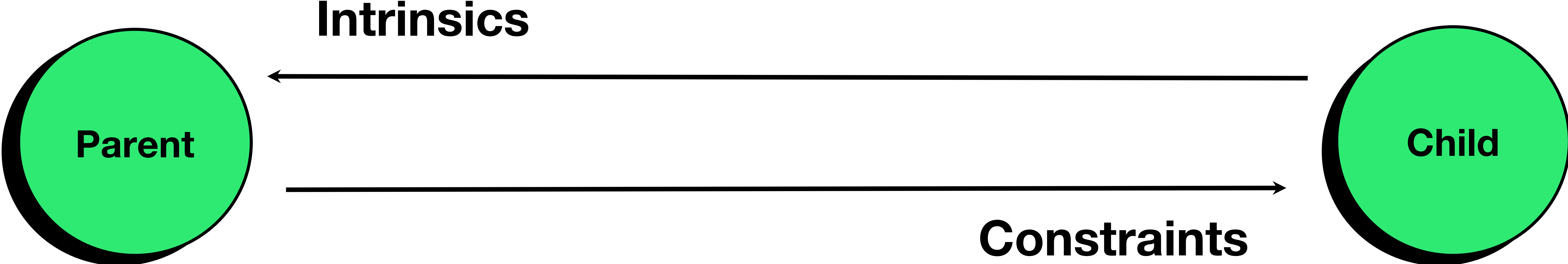

Y0

```
@Composable
fun ConstraintLayout(...) {

}
```

```
@Composable
fun ConstraintLayout(...) {
   MultiMeasureLayout(...)
}
```

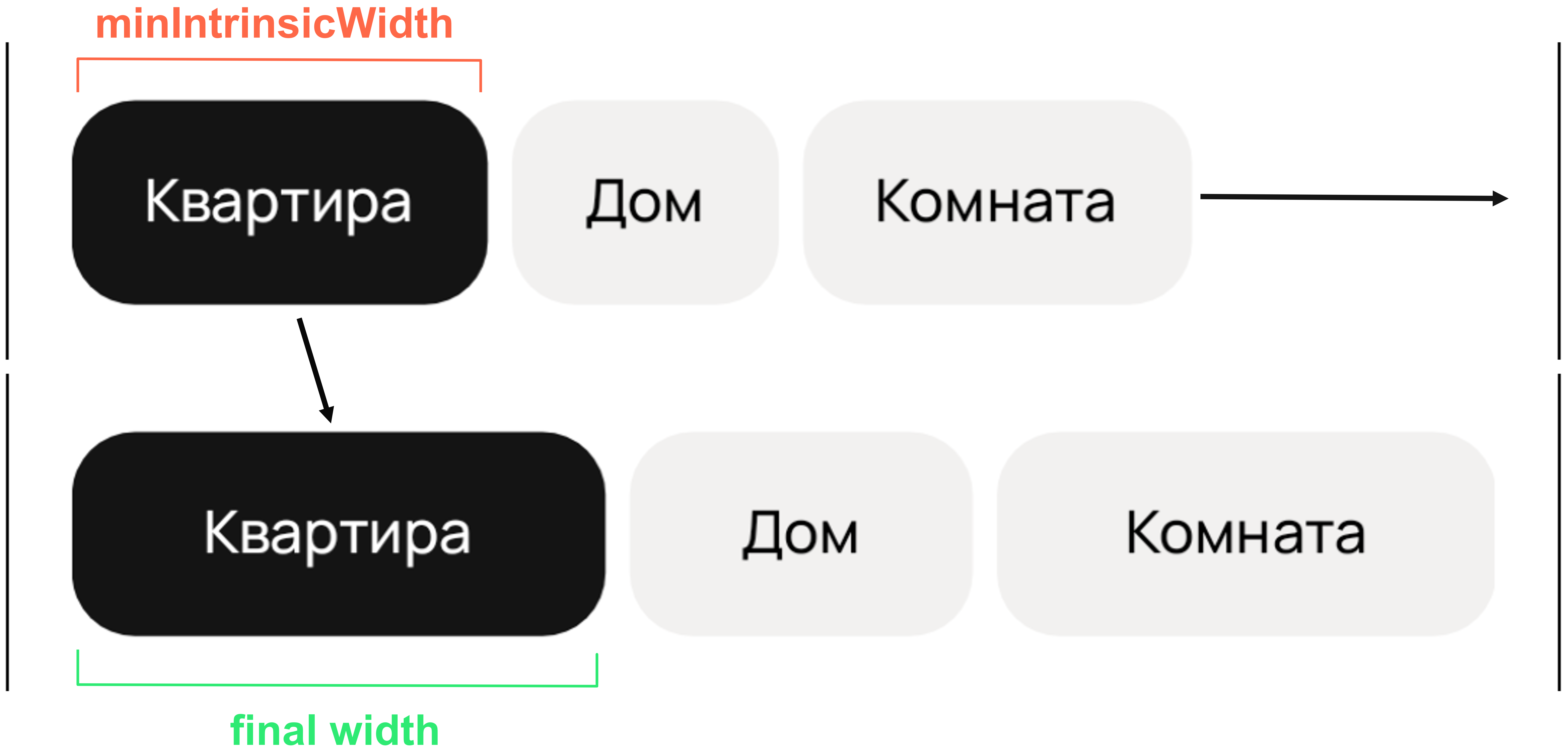
Intrinsic Measurements

Intrinsics позволяют опрашивать дочерние ноды **до их измерения**


Intrinsics
Parent
Child
Constraints

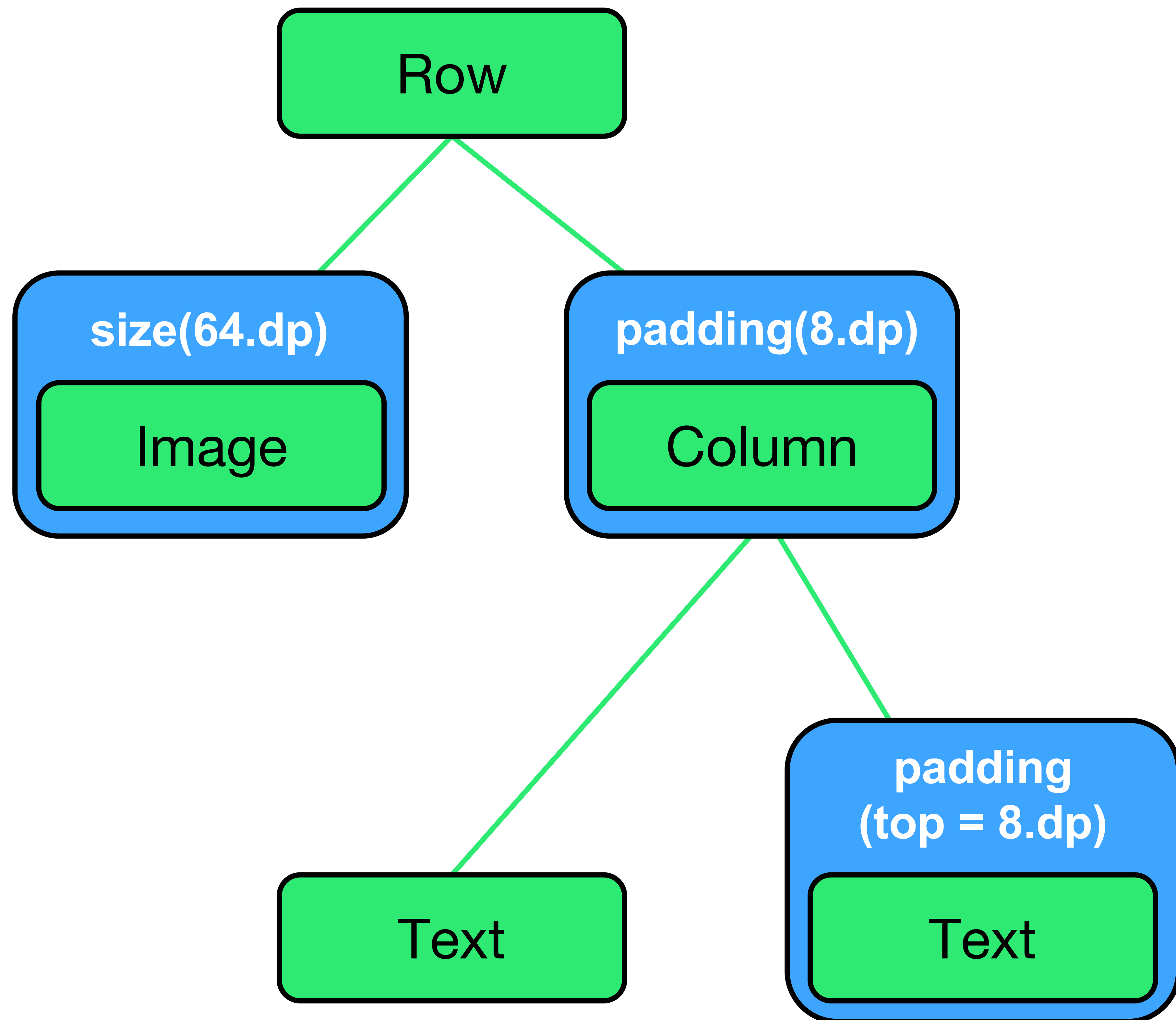
```
interface Measurable : IntrinsicMeasurable

interface IntrinsicMeasurable {
  fun minIntrinsicWidth(height: Int): Int
  fun maxIntrinsicWidth(height: Int): Int
  fun minIntrinsicHeight(width: Int): Int
  fun maxIntrinsicHeight(width: Int): Int
}
```

`min/maxIntrinsicWidth` - какую **минимальную/максимальную ширину может занять** measurable при заданной высоте?

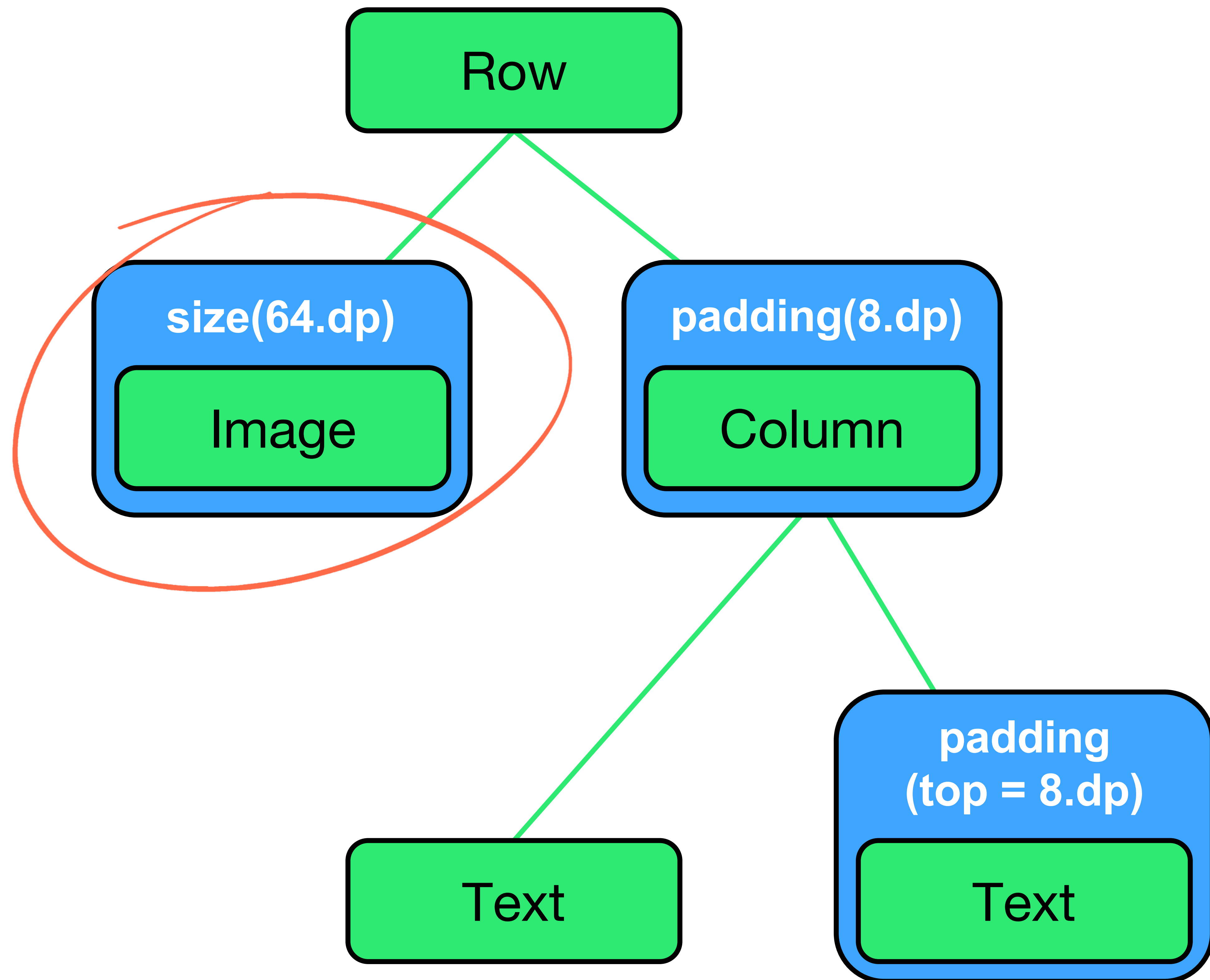

minIntrinsicWidth
Квартира
Дом
Комната
Квартира
Дом
Комната
final width

```
Layout { measurables, constraints ->
    val widths = measurables.map {
        it.minIntrinsicWidth(constraints.maxHeight)
    }
    val widthCoef = constraints.maxWidth / widths.sumOf { it.width }
    val placeables = measurables.mapIndexed { i, measurable ->
        val fixedWidthConstraints = calculateConstraints(widths[i], widthCoef)
        measurable.measure(fixedWidthConstraints)
    }
    ...
}
```

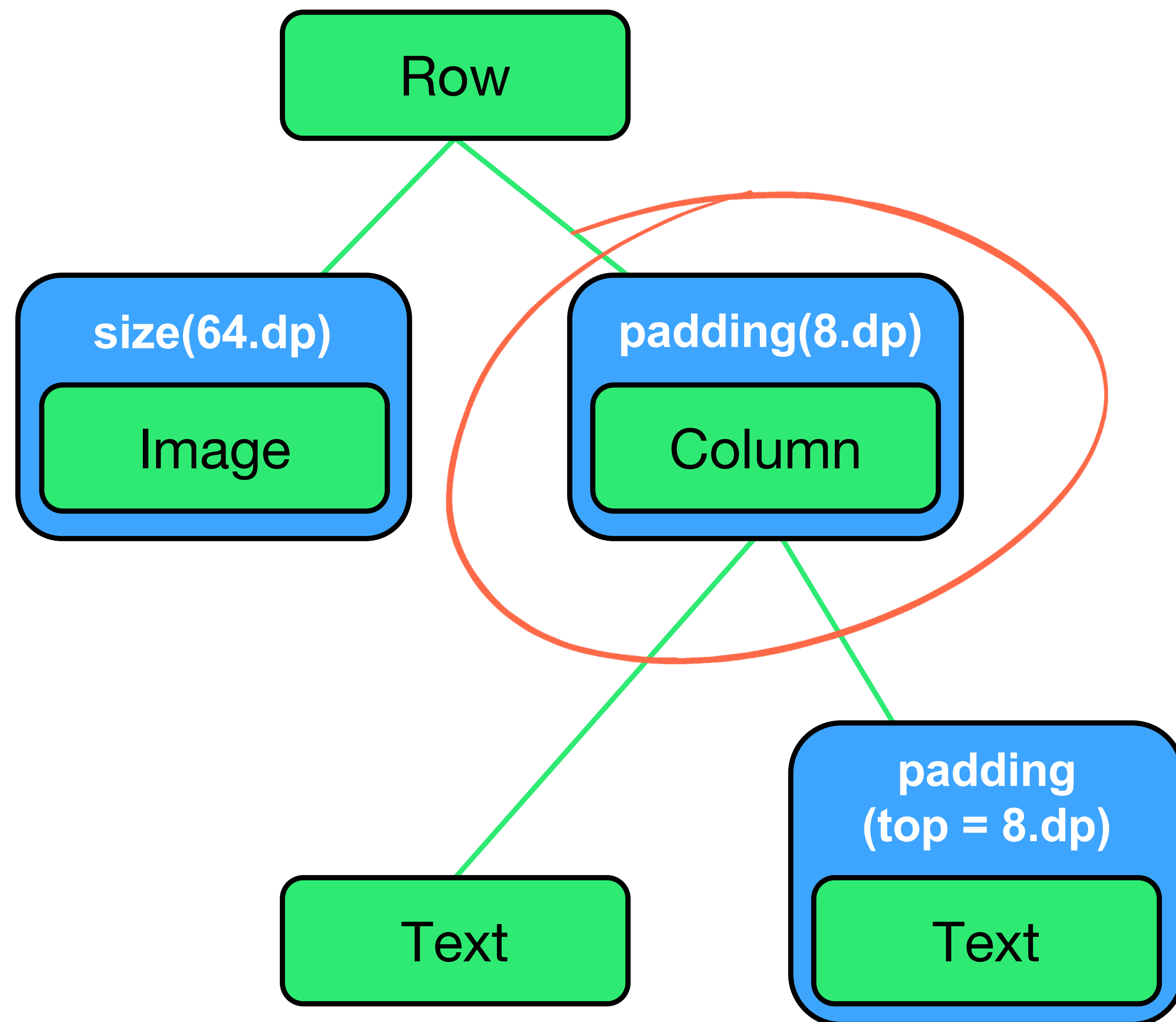
```
Layout { measurables, constraints ->
    val widths = measurables.map {
        it.minIntrinsicWidth(constraints.maxHeight)
    }
    val widthCoef = constraints.maxWidth / widths.sumOf { it.width }
    val placeables = measurables.mapIndexed { i, measurable ->
        val fixedWidthConstraints = calculateConstraints(widths[i], widthCoef)
        measurable.measure(fixedWidthConstraints)
    }
    ...
}
```



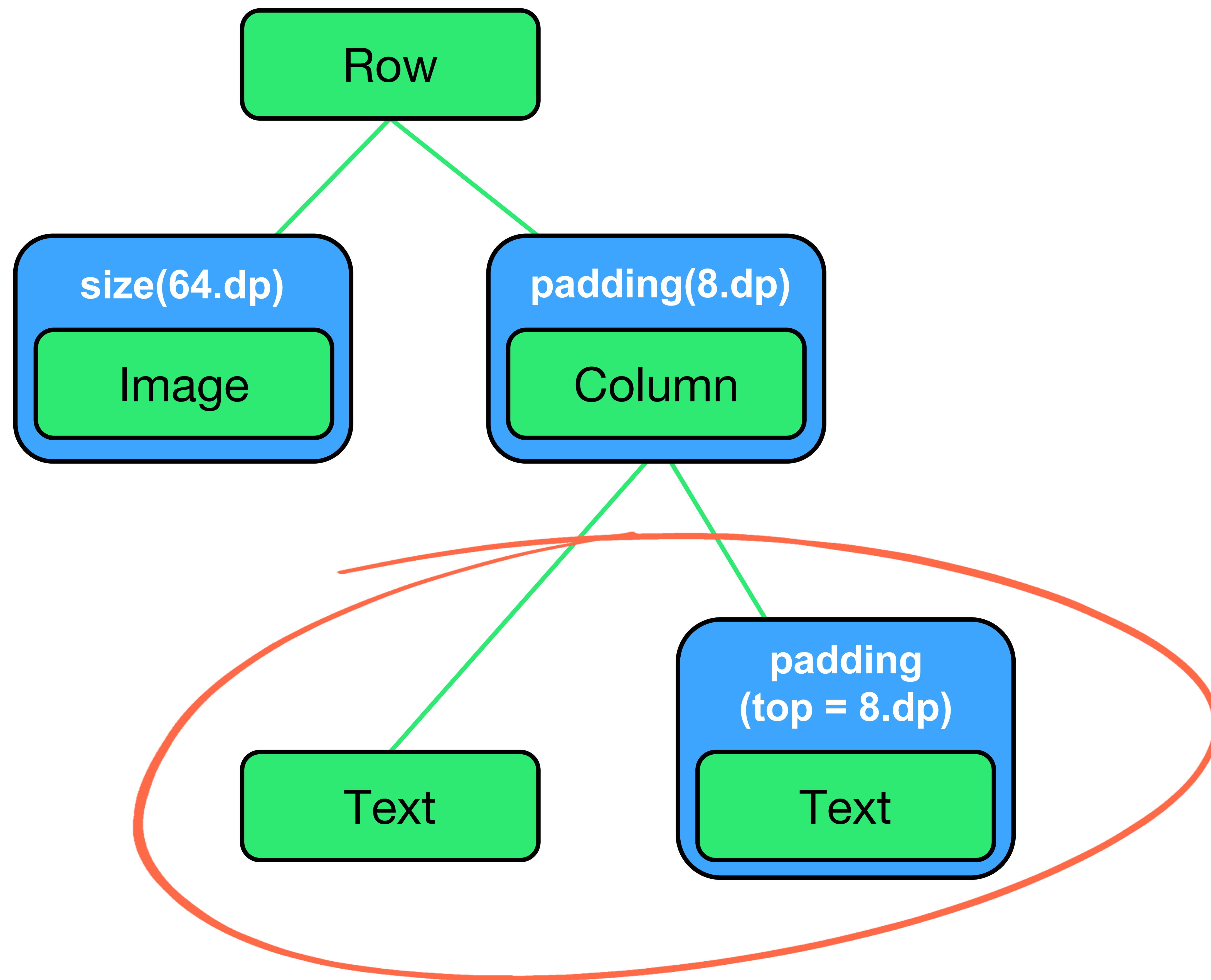
Row.minIntrinsicWidth = ?



Row.minIntrinsicWidth = 64 +



Row.minIntrinsicWidth = 64 + 8 * 2 +



Row.minIntrinsicWidth = 64 + 8 * 2 + max(Text1.minIntrinsicWidth, Text2.minIntrinsicWidth)

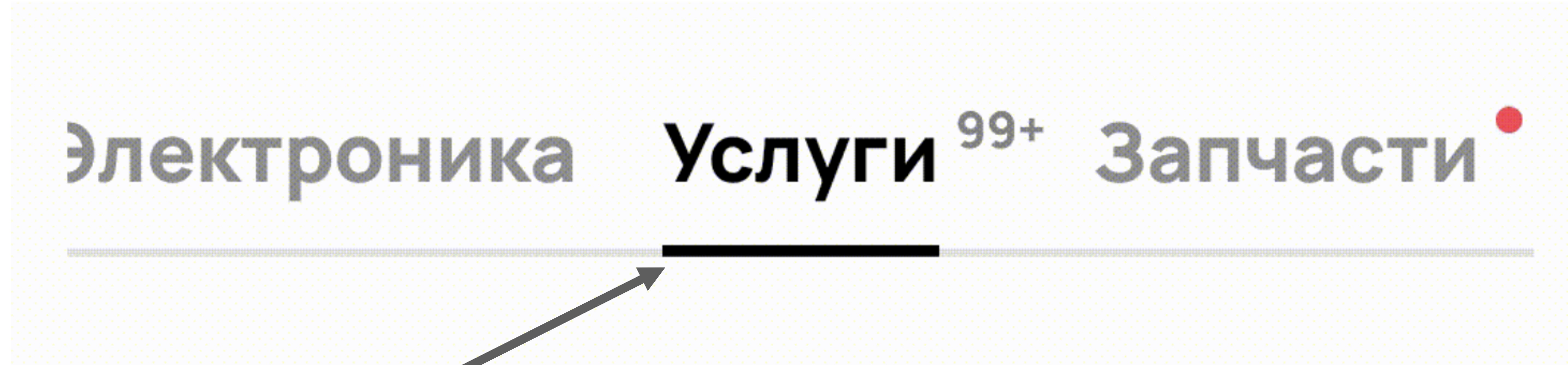
```
Layout(object: MeasurePolicy {

    override fun IntrinsicMeasureScope.minIntrinsicWidth(
        measurables: List<IntrinsicMeasurable>,
        height: Int
    ): Int {
        TODO("Not yet implemented")
    }

    ...
})
```

Откладываем композицию

Распродажа   Молл [17]   Недвижимост

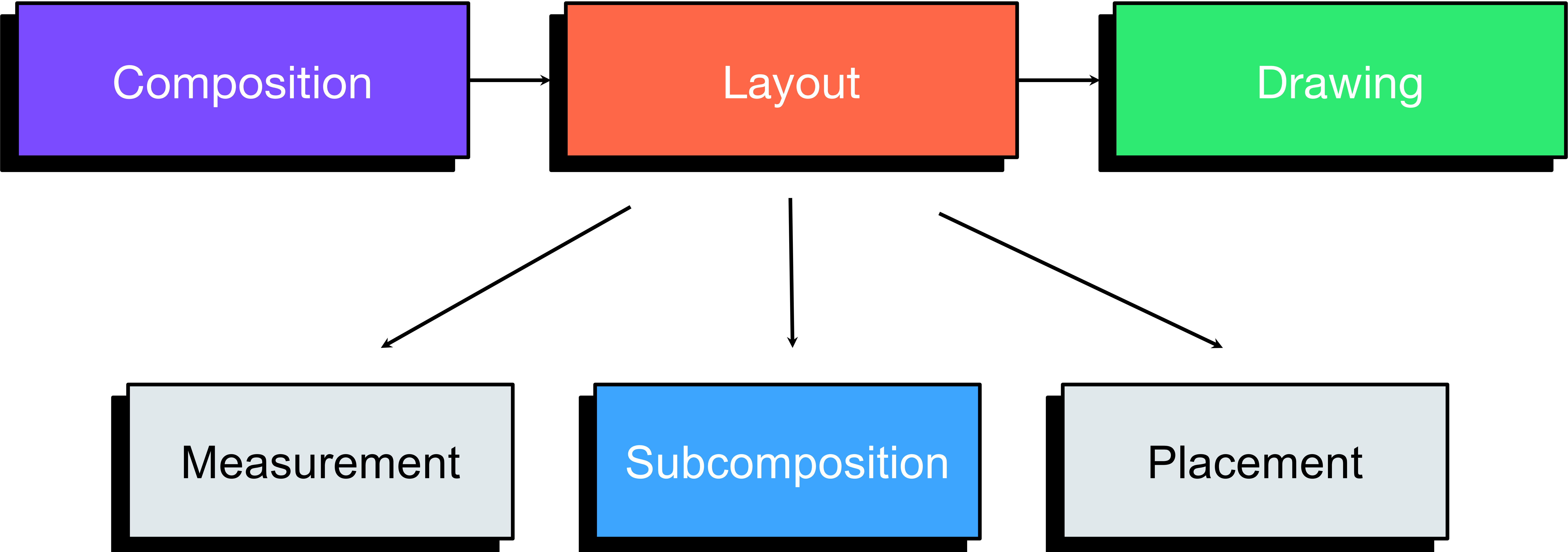
```
@Composable
fun Tab() {
    Row {
        Text()
        Text()
    }
}
```

```kotlin
@Composable
fun Indicator(
    selectedTabPosition: SelectedTabPosition
) {
     val selectedWidth by animateIntAsState(selectedTabPosition.width)
     val selectedX by animateIntAsState(selectedTabPosition.x)

     Canvas {
         drawRect(...)
         drawRect(...)
     }
}
```




```
@Composable
fun Indicator(
    selectedTabPosition: SelectedTabPosition
) {
    val selectedWidth by animateIntAsState(selectedTabPosition.width)
    val selectedX by animateIntAsState(selectedTabPosition.x)

    Canvas {
        drawRect(...)
        drawRect(...)
    }
}
```

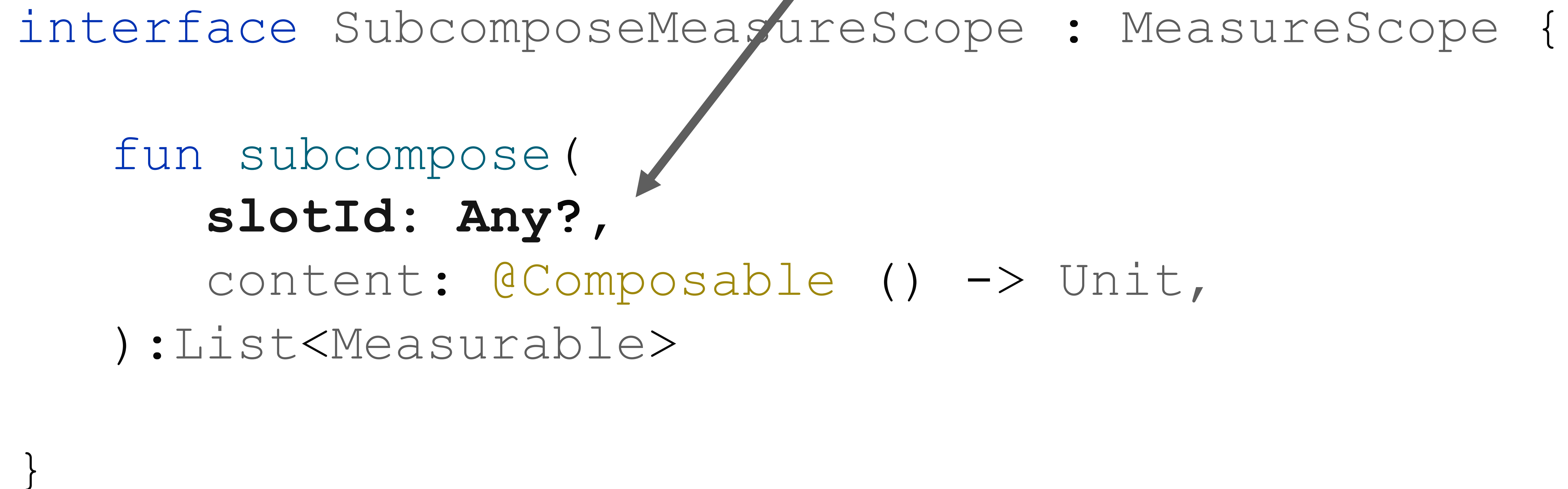
Нужно **использовать размеры** одной ноды **при композиции** другой


Composition
Layout
Drawing
Measurement
Subcomposition
Placement

```
@Composable
fun SubcomposeLayout(
    measurePolicy: SubcomposeMeasureScope.(Constraints) -> MeasureResult
)
```

```
interface SubcomposeMeasureScope : MeasureScope {

   fun subcompose(
      slotId: Any?,
      content: @Composable () -> Unit,
   ):List<Measurable>

}
```

Уникальный ID, используемый для идентификации слота при рекомпозиции



```
enum class TabGroupSlots {
    Tabs,
    Indicator
}
```

```kotlin
SubcomposeLayout { constraints ->
    val tabMeasurables = subcompose(TabSlots.Tabs) { tabs() }
    val tabPlaceables = tabMeasurables.map { it.measure(constraints) }
    ...
    layout(width, height) {
        var selectedTabPosition = SelectedTabPosition(0, 0)

        tabPlaceables.forEachIndexed { index, placeable ->
            placeable.placeRelative(x, y)
            if (index == selectedTabIndex) selectedTabPosition = getTabPosition(...)
        }

        subcompose(TabSlots.Indicator) { indicator(selectedTabPosition) }.first()
            .measure(constraints).placeRelative(0, 0)
    }
}
```

```
SubcomposeLayout { constraints ->
    val tabMeasurables = subcompose(TabSlots.Tabs) { tabs() }
    val tabPlaceables = tabMeasurables.map { it.measure(constraints) }
    ...
    layout(width, height) {
        var selectedTabPosition = SelectedTabPosition(0, 0)

        tabPlaceables.forEachIndexed { index, placeable ->
            placeable.placeRelative(x, y)
            if (index == selectedTabIndex) selectedTabPosition = getTabPosition(...)
        }

        subcompose(TabSlots.Indicator) { indicator(selectedTabPosition) }.first()
            .measure(constraints).placeRelative(0, 0)
    }
}
```

1. Композируем и измеряем табы

```kotlin
SubcomposeLayout { constraints ->
   val tabMeasurables = subcompose(TabSlots.Tabs) { tabs() }
   val tabPlaceables = tabMeasurables.map { it.measure(constraints) }
   ...
   layout(width, height) {
       var selectedTabPosition = SelectedTabPosition(0, 0)

       tabPlaceables.forEachIndexed { index, placeable ->
           placeable.placeRelative(x, y)
           if (index == selectedTabIndex) selectedTabPosition = getTabPosition(...)
       }

       subcompose(TabSlots.Indicator) { indicator(selectedTabPosition) }.first()
           .measure(constraints).placeRelative(0, 0)
   }
}
```

2. Размещаем табы ...

```
SubcomposeLayout { constraints ->
   val tabMeasurables = subcompose(TabSlots.Tabs) { tabs() }
   val tabPlaceables = tabMeasurables.map { it.measure(constraints) }
   ...
   layout(width, height) {
       var selectedTabPosition = SelectedTabPosition(0, 0)

       tabPlaceables.forEachIndexed { index, placeable ->
           placeable.placeRelative(x, y)
           if (index == selectedTabIndex) selectedTabPosition = getTabPosition(...)
       }

       subcompose(TabSlots.Indicator) { indicator(selectedTabPosition) }.first()
           .measure(constraints).placeRelative(0, 0)
   }
}
```

2. Размещаем табы **и находим позицию выбранного таба**

Как получить размер текста?

```
val textMeasurer = rememberTextMeasurer()

...
```

```
val textMeasurer = rememberTextMeasurer()

...

val tabTextWidth = textMeasurer.measure(
    text = tabText,
    style = textStyle,
).size.width
```

```
SubcomposeLayout { constraints ->
   val tabMeasurables = subcompose(TabSlots.Tabs) { tabs() }
   val tabPlaceables = tabMeasurables.map { it.measure(constraints) }
   ...
   layout(width, height) {
       var selectedTabPosition = SelectedTabPosition(0, 0)

       tabPlaceables.forEachIndexed { index, placeable ->
           placeable.placeRelative(x, y)
           if (index == selectedTabIndex) selectedTabPosition = getTabPosition(...)
       }

       subcompose(TabSlots.Indicator) { indicator(selectedTabPosition) }.first()
           .measure(constraints).placeRelative(0, 0)
   }
}
```

3. Композируем, измеряем и размещаем индикатор

```
SubcomposeLayout { constraints ->
   val tabMeasurables = subcompose(TabSlots.Tabs) { tabs() }
   val tabPlaceables = tabMeasurables.map { it.measure(constraints) }
   ...
   layout(width, height) {
       var selectedTabPosition = SelectedTabPosition(0, 0)

       tabPlaceables.forEachIndexed { index, placeable ->
           placeable.placeRelative(x, y)
           if (index == selectedTabIndex) selectedTabPosition = getTabPosition(...)
       }

       subcompose(TabSlots.Indicator) { indicator(selectedTabPosition) }.first()
           .measure(constraints).placeRelative(0, 0)
   }
}
```

3. Композируем, **измеряем и размещаем индикатор**

Распродажа Молл 17 Недвижимост

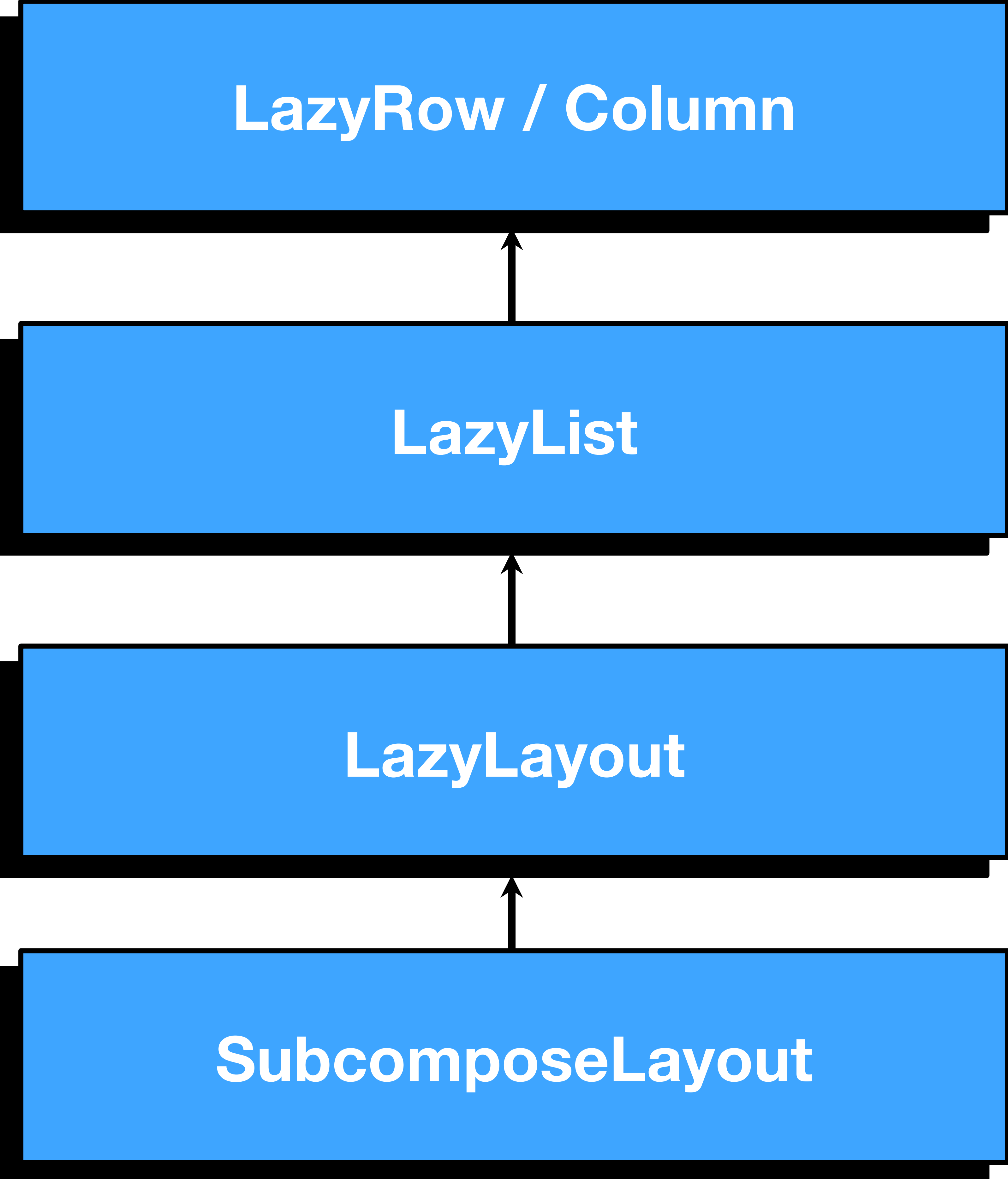
Что еще умеет SubcomposeLayout?

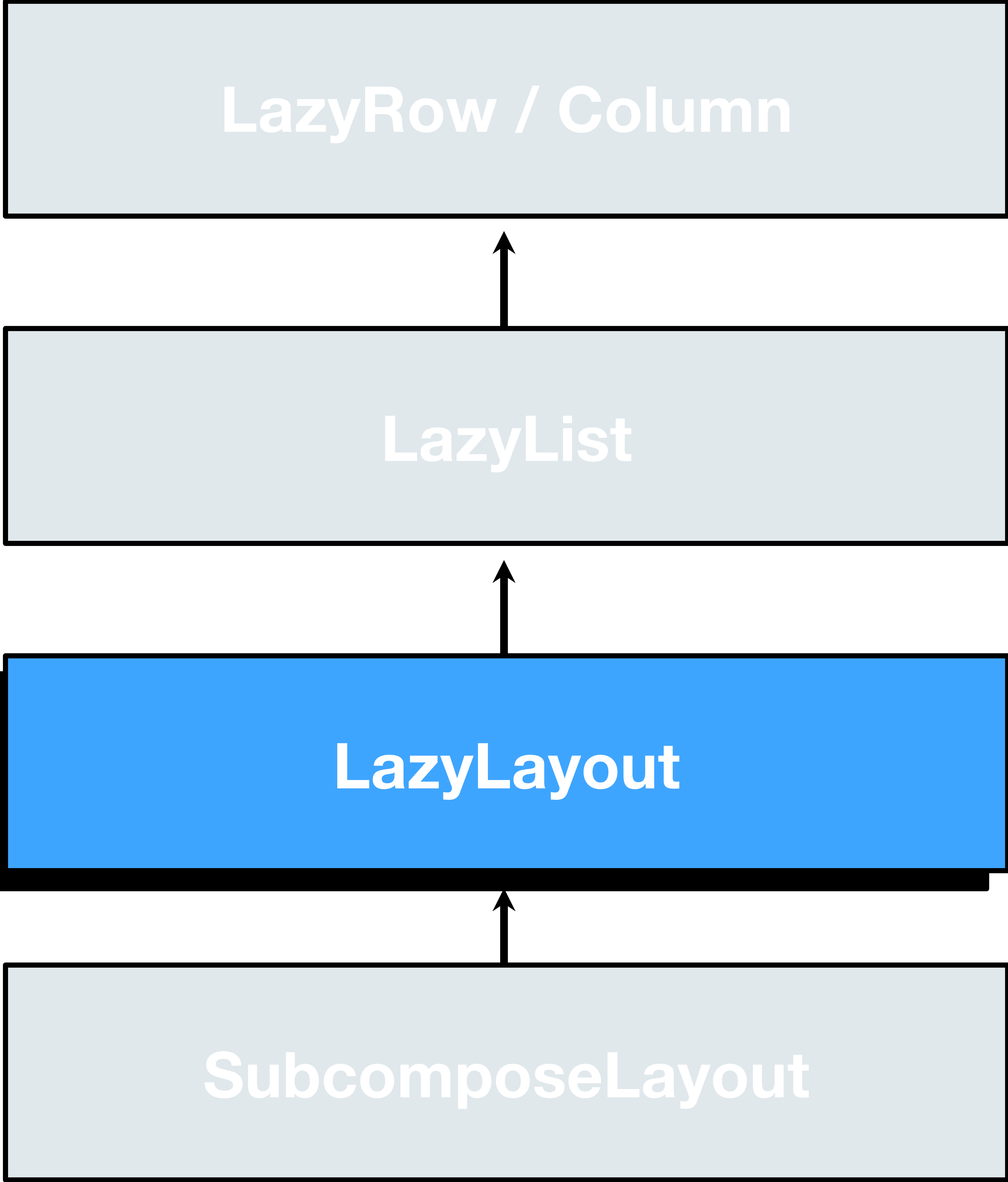
```kotlin
interface SubcomposeSlotReusePolicy {

 fun getSlotsToRetain(slotIds: SlotIdsSet)

 fun areCompatible(slotId: Any?, reusableSlotId: Any?): Boolean

}
```

SubcomposeSlotReusePolicy **сохраняет слоты для переиспользования** в будущем.

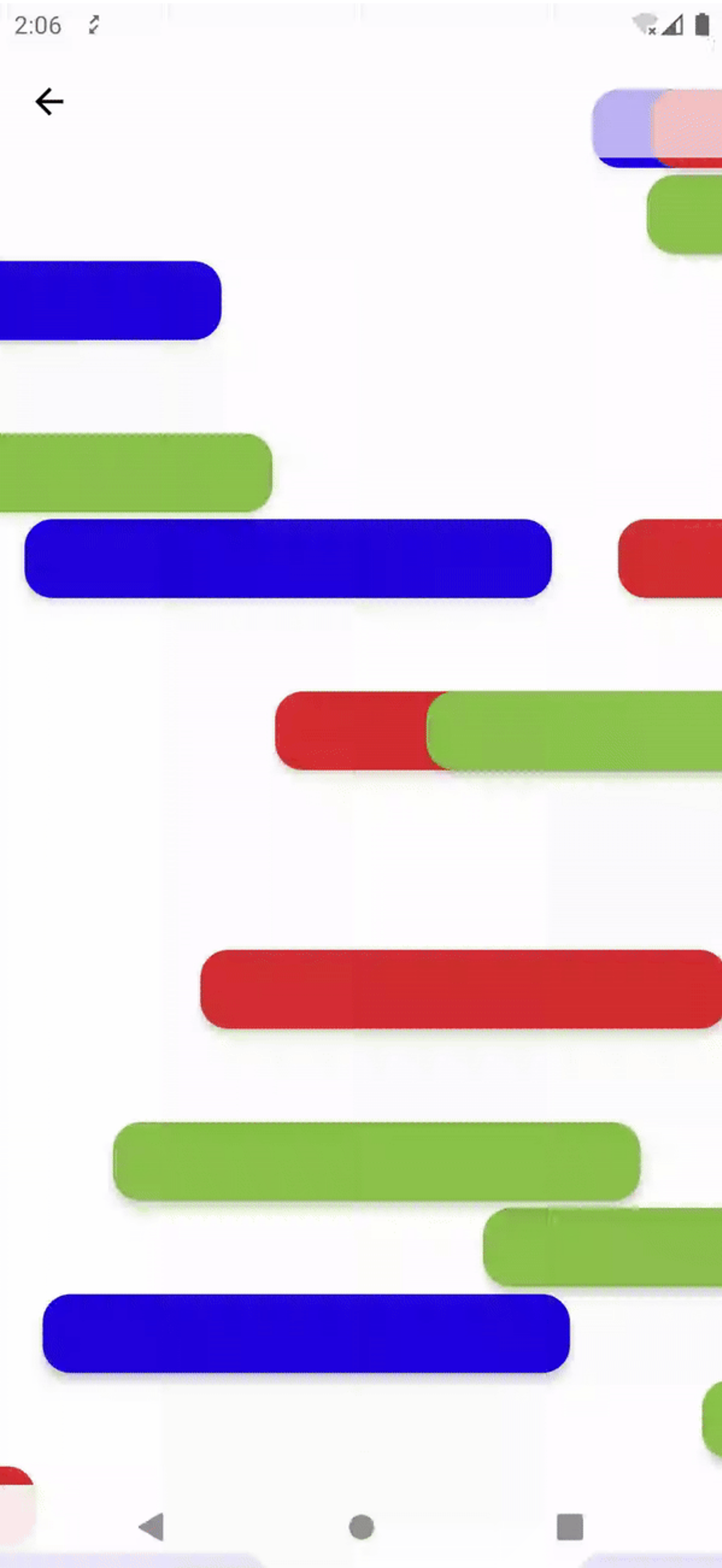
`~ RecycledViewPool`

LazyRow/Column <- SubcomposeLayout

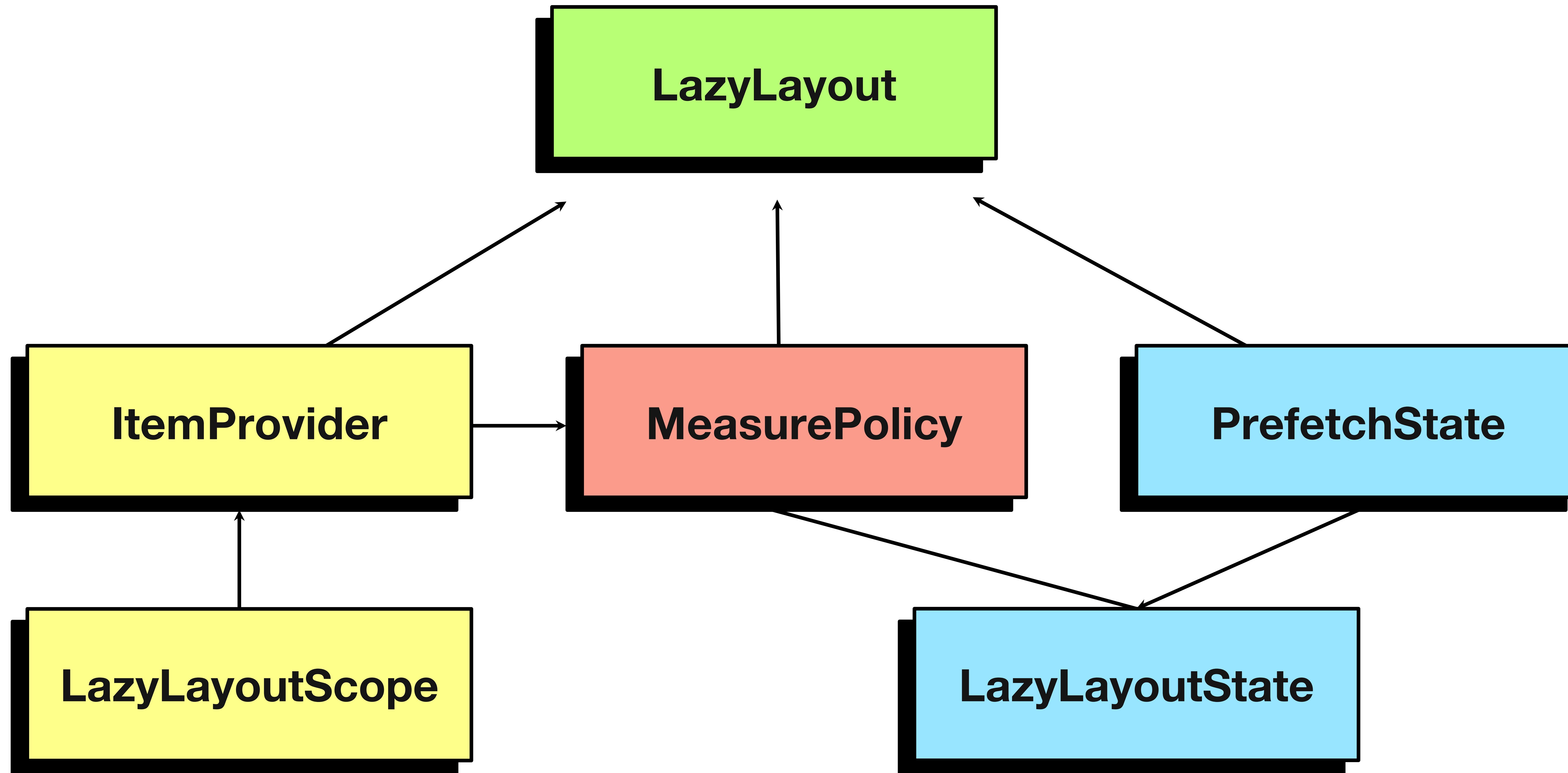

LazyRow / Column
LazyList
LazyLayout
SubcomposeLayout


LazyRow / Column
LazyList
LazyLayout
SubcomposeLayout

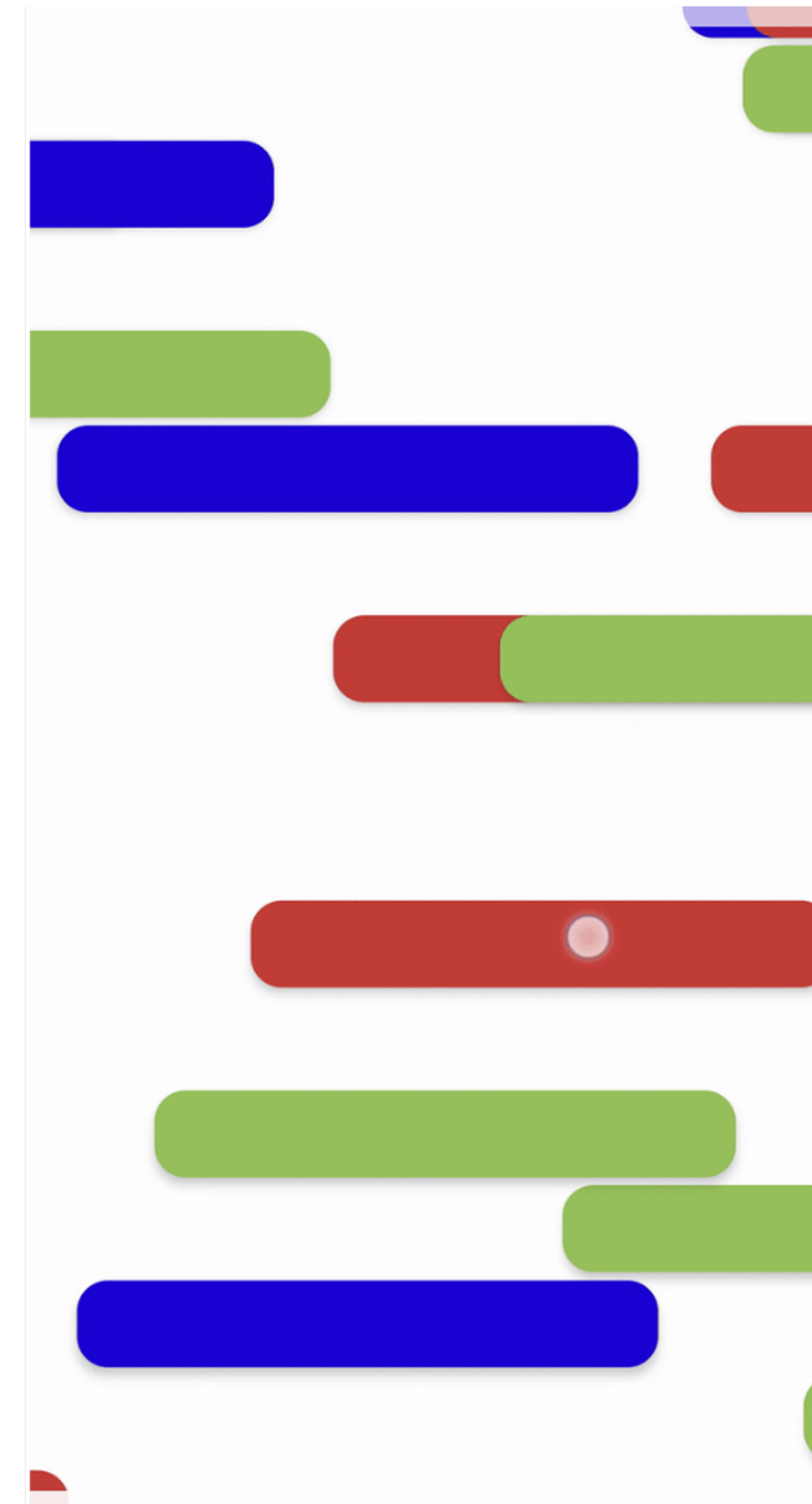
Кастомный LazyLayout


2:06

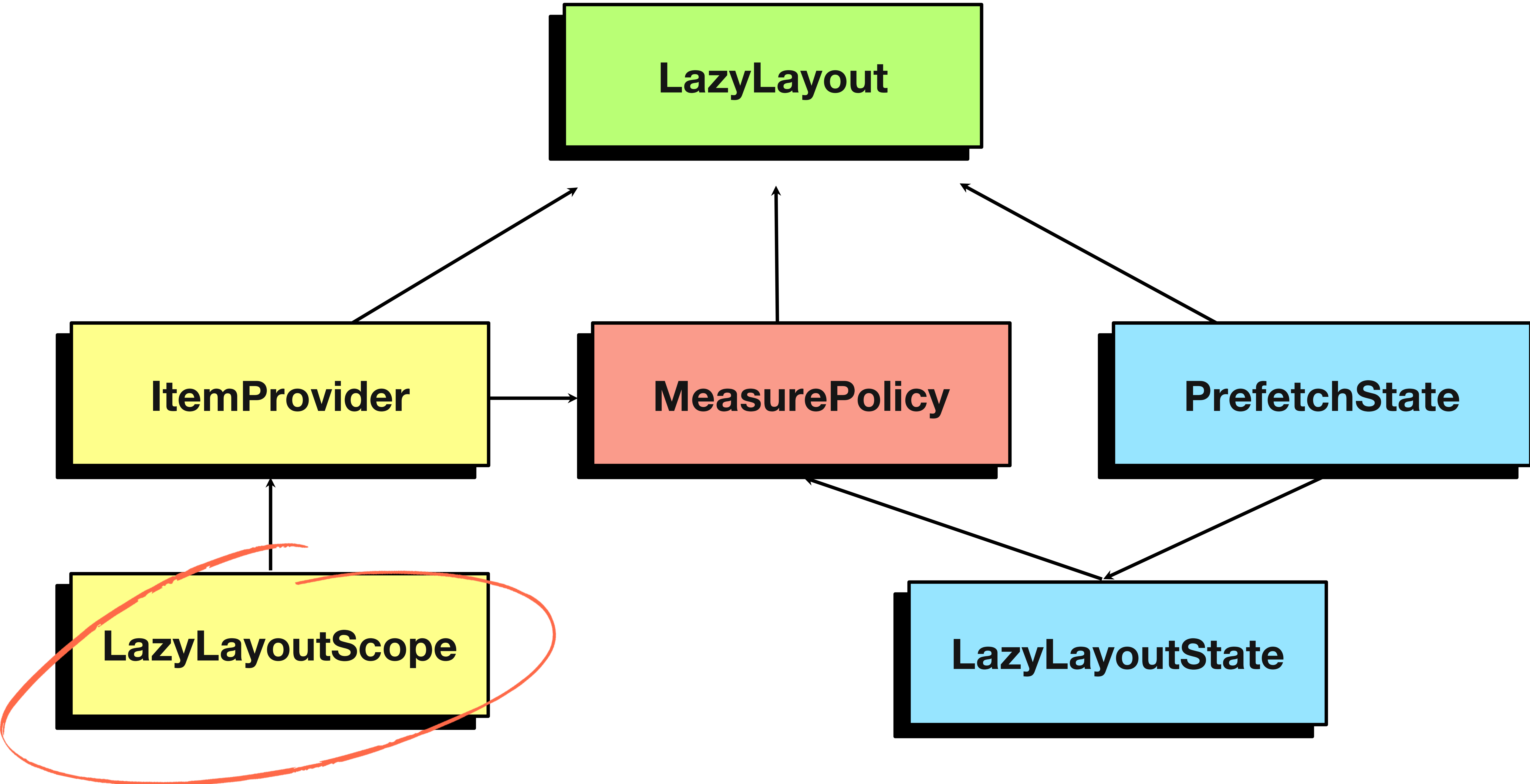
```kotlin
@Composable
fun LazyLayout(
    itemProvider: LazyLayoutItemProvider,
    prefetchState: LazyLayoutPrefetchState?,
    measurePolicy: LazyLayoutMeasureScope.(Constraints) -> MeasureResult,
   ...
)
```


LazyLayout
ItemProvider
MeasurePolicy
PrefetchState
LazyLayoutScope
LazyLayoutState

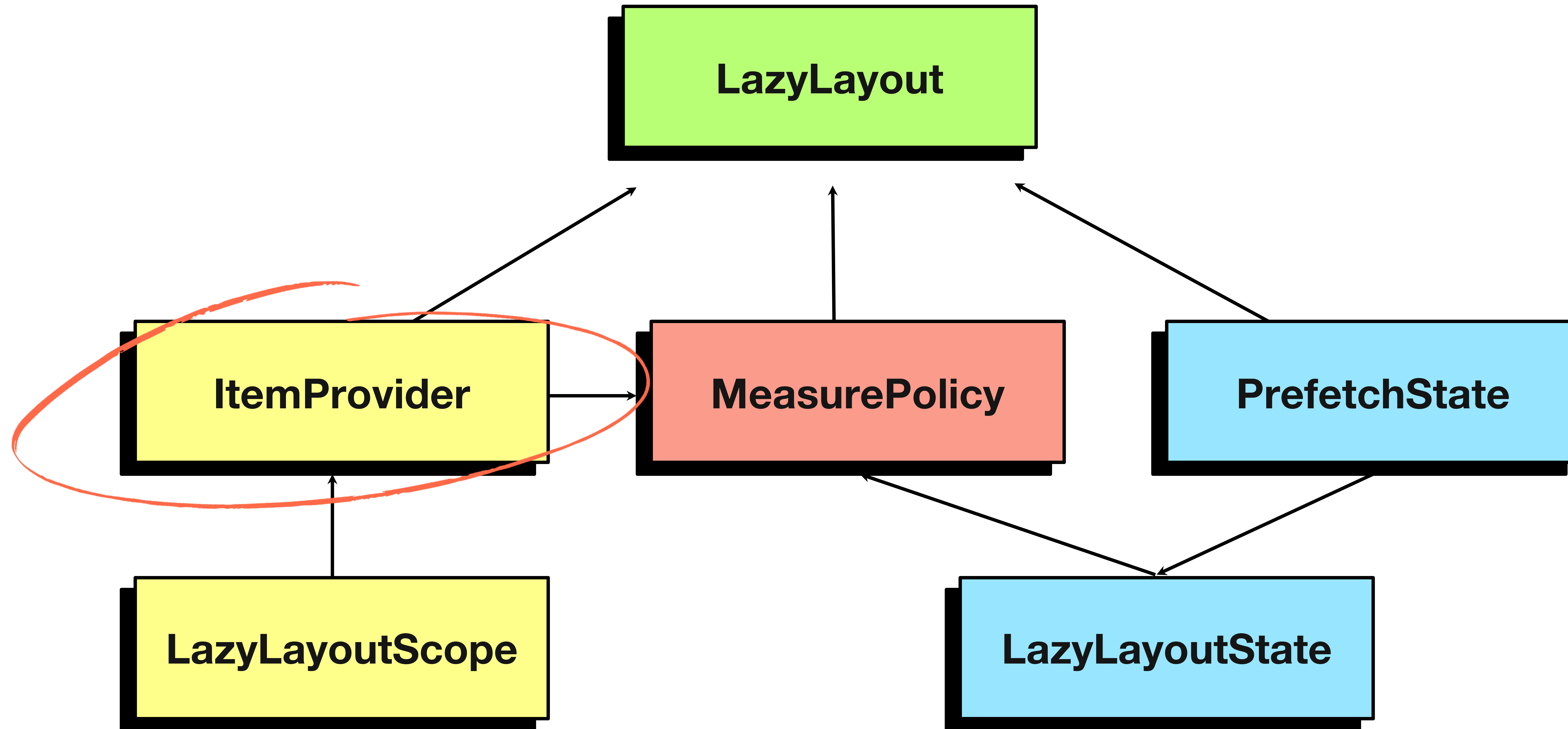
```kotlin
data class ItemState(
    val x: Int,
    val y: Int,
    val color: Color,
    ...
)
```



```
CustomLazyLayout(...) {

  items(state.items) { item: ItemState ->
      ItemComposable(item)
  }

}
```


LazyLayout
ItemProvider
MeasurePolicy
PrefetchState
LazyLayoutScope
LazyLayoutState

```kotlin
class CustomLazyLayoutScope {

    val items: List<Item>

    fun items(items: List<ItemState>, itemContent: @Composable (ItemState) -> Unit) {
        items.forEach { _items.add(Item(it, itemContent)) }
    }
}
```


LazyLayout
ItemProvider
MeasurePolicy
PrefetchState
LazyLayoutScope
LazyLayoutState

```kotlin
interface LazyLayoutItemProvider {

    val itemCount: Int

    @Composable
    fun Item(index: Int, key: Any)

    fun getContentType(index: Int): Any?

    fun getKey(index: Int): Any

    fun getIndex(key: Any): Int
}
```

```kotlin
interface LazyLayoutItemProvider {

    val itemCount: Int

    @Composable
    fun Item(index: Int, key: Any)

    fun getContentType(index: Int): Any?

    fun getKey(index: Int): Any

    fun getIndex(key: Any): Int
}
```

```kotlin
abstract class RecyclerView.Adapter {

    fun getItemCount(): Int


    fun onBindViewHolder(holder, position)

    fun getItemViewType(position: Int): Int

    fun getItemId(position: Int): Long

    ...
}
```

```kotlin
interface LazyLayoutItemProvider {

    val itemCount: Int

    @Composable
    fun Item(index: Int, key: Any)

    fun getContentType(index: Int): Any?

    fun getKey(index: Int): Any

    fun getIndex(key: Any): Int
}
```

```kotlin
abstract class RecyclerView.Adapter {

    fun getItemCount(): Int


    fun onBindViewHolder(holder, position)

    fun getItemViewType(position: Int): Int

    fun getItemId(position: Int): Long

    ...
}
```

```
interface LazyLayoutItemProvider {

    val itemCount: Int

    @Composable
    fun Item(index: Int, key: Any)

    fun getContentType(index: Int): Any?

    fun getKey(index: Int): Any

    fun getIndex(key: Any): Int
}
```

```
abstract class RecyclerView.Adapter {

    fun getItemCount(): Int


    fun onBindViewHolder(holder, position)

    fun getItemViewType(position: Int): Int

    fun getItemId(position: Int): Long

    ...
}
```

```kotlin
interface LazyLayoutItemProvider {

    val itemCount: Int

    @Composable
    fun Item(index: Int, key: Any)

    fun getContentType(index: Int): Any?

    fun getKey(index: Int): Any

    fun getIndex(key: Any): Int
}
```

```kotlin
abstract class RecyclerView.Adapter {

    fun getItemCount(): Int


    fun onBindViewHolder(holder, position)

    fun getItemViewType(position: Int): Int

    fun getItemId(position: Int): Long

    ...
}
```

```kotlin
interface LazyLayoutItemProvider {

    val itemCount: Int

    @Composable
    fun Item(index: Int, key: Any)

    fun getContentType(index: Int): Any?

    fun getKey(index: Int): Any

    fun getIndex(key: Any): Int
}
```

```kotlin
abstract class RecyclerView.Adapter {

    fun getItemCount(): Int


    fun onBindViewHolder(holder, position)

    fun getItemViewType(position: Int): Int

    fun getItemId(position: Int): Long

    ...
}
```

```kotlin
interface LazyLayoutItemProvider {

    val itemCount: Int

    @Composable
    fun Item(index: Int, key: Any)

    fun getContentType(index: Int): Any?

    fun getKey(index: Int): Any

    fun getIndex(key: Any): Int
}
```

```
interface LazyLayoutItemProvider {

    val itemCount: Int

    @Composable
    fun Item(index: Int, key: Any)

    fun getContentType(index: Int): Any?

    fun getKey(index: Int): Any

    fun getIndex(key: Any): Int
}
```

Опционально (но необходимо для переиспользования слотов)

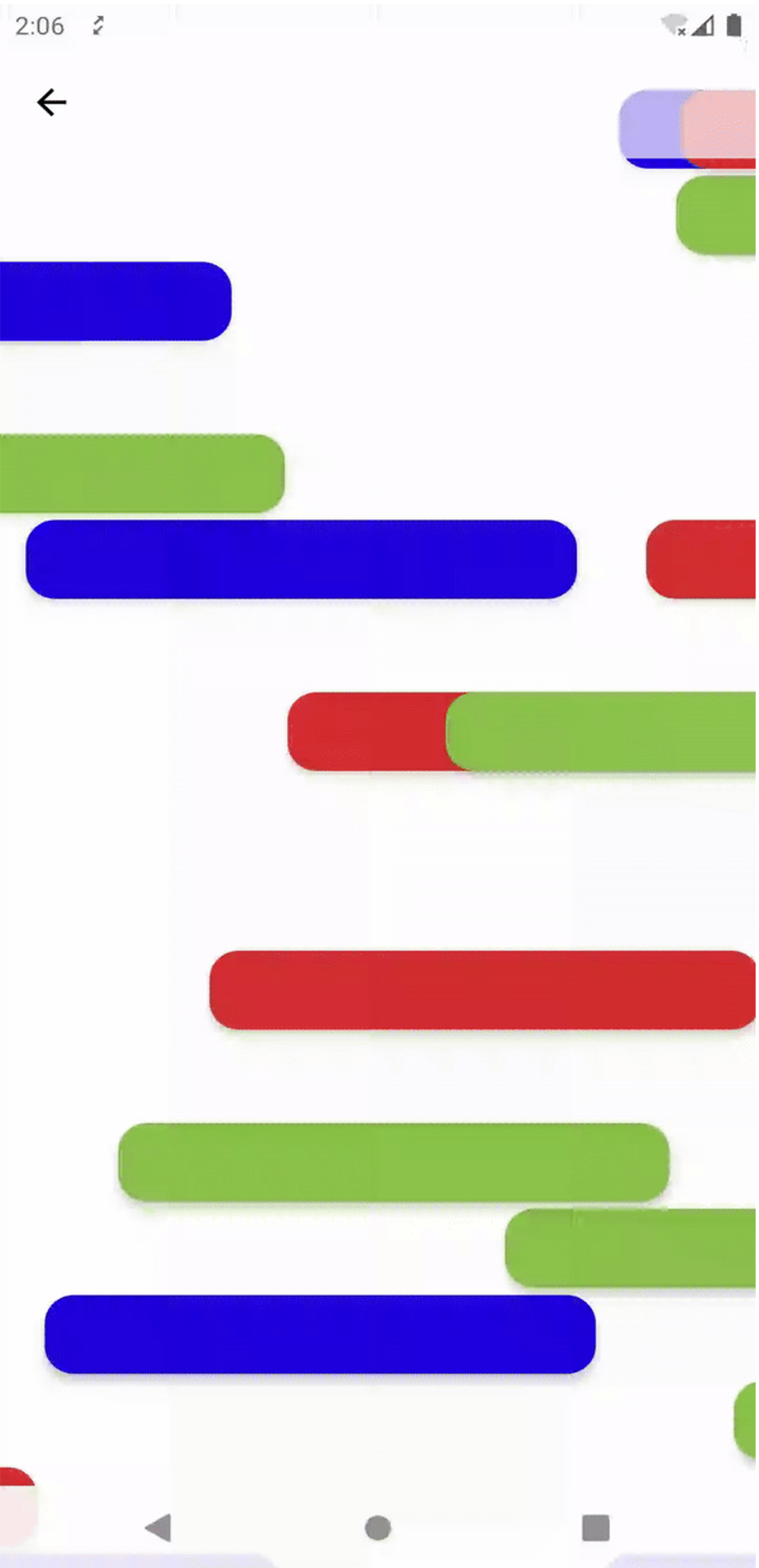
```kotlin
class ItemProvider(
   private val itemsState: List<Item>,
) : LazyLayoutItemProvider {

   override val itemCount
       get() = itemsState.size

   @Composable
   override fun Item(index: Int, key: Any) {
       val item = itemsState.getOrNull(index)
       item?.content?.invoke(item.item)
   }
}
```

```kotlin
class ItemProvider(
   private val itemsState: List<Item>,
) : LazyLayoutItemProvider {

   override val itemCount
       get() = itemsState.size

   @Composable
   override fun Item(index: Int, key: Any) {
       val item = itemsState.getOrNull(index)
       item?.content?.invoke(item.item)
   }
}
```

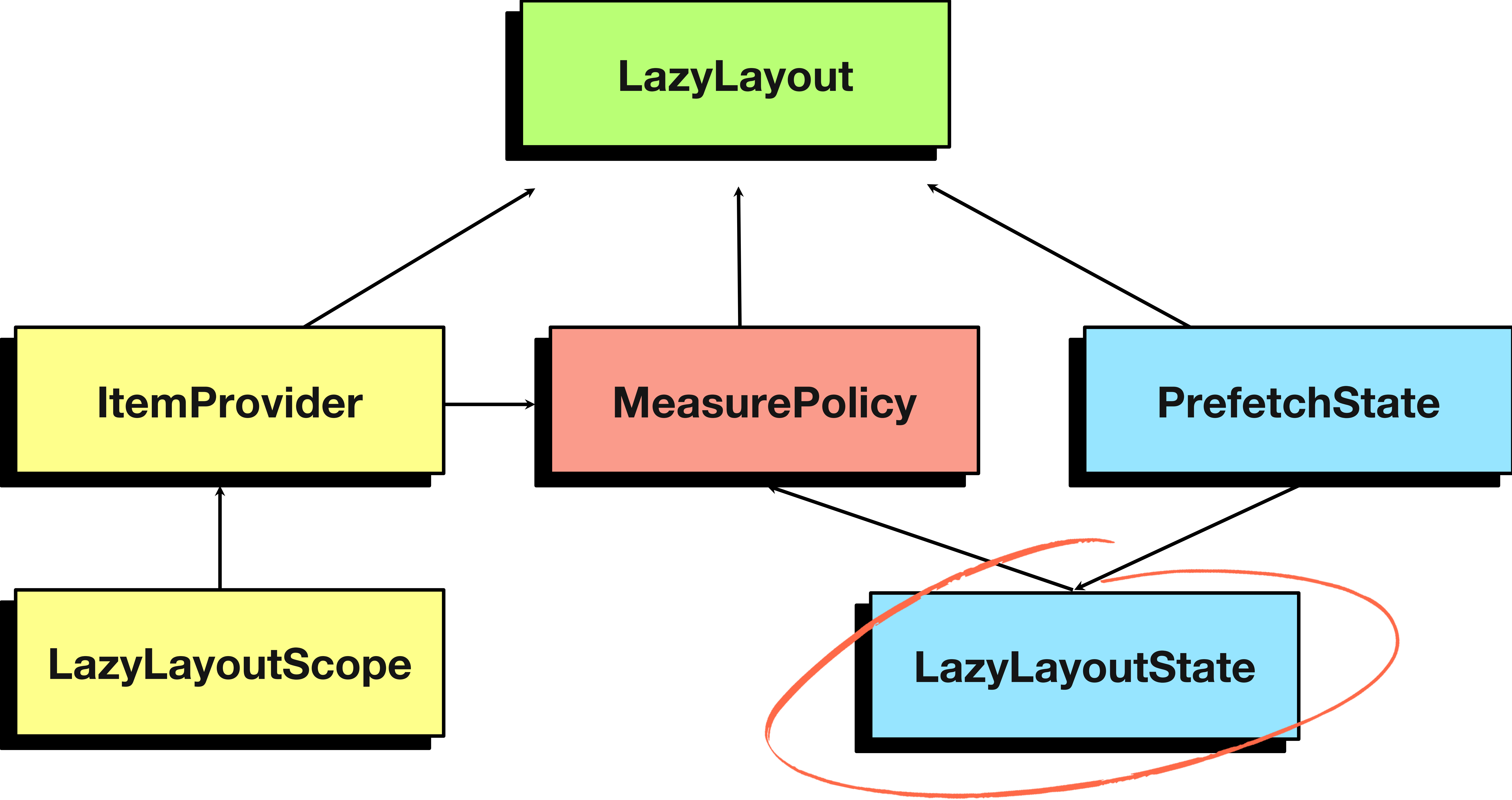
```kotlin
class ItemProvider(
   private val itemsState: List<Item>,
) : LazyLayoutItemProvider {

   override val itemCount
       get() = itemsState.size

   @Composable
   override fun Item(index: Int, key: Any) {
       val item = itemsState.getOrNull(index)
       item?.content?.invoke(item.item)
   }
}
```

```
@Composable
fun CustomLazyLayout(
    content: CustomLazyListScope.() -> Unit,
) {
    val itemProvider = rememberItemProvider(content)
    LazyLayout(
        itemProvider = itemProvider,
    )
}
```

Добавим скролл



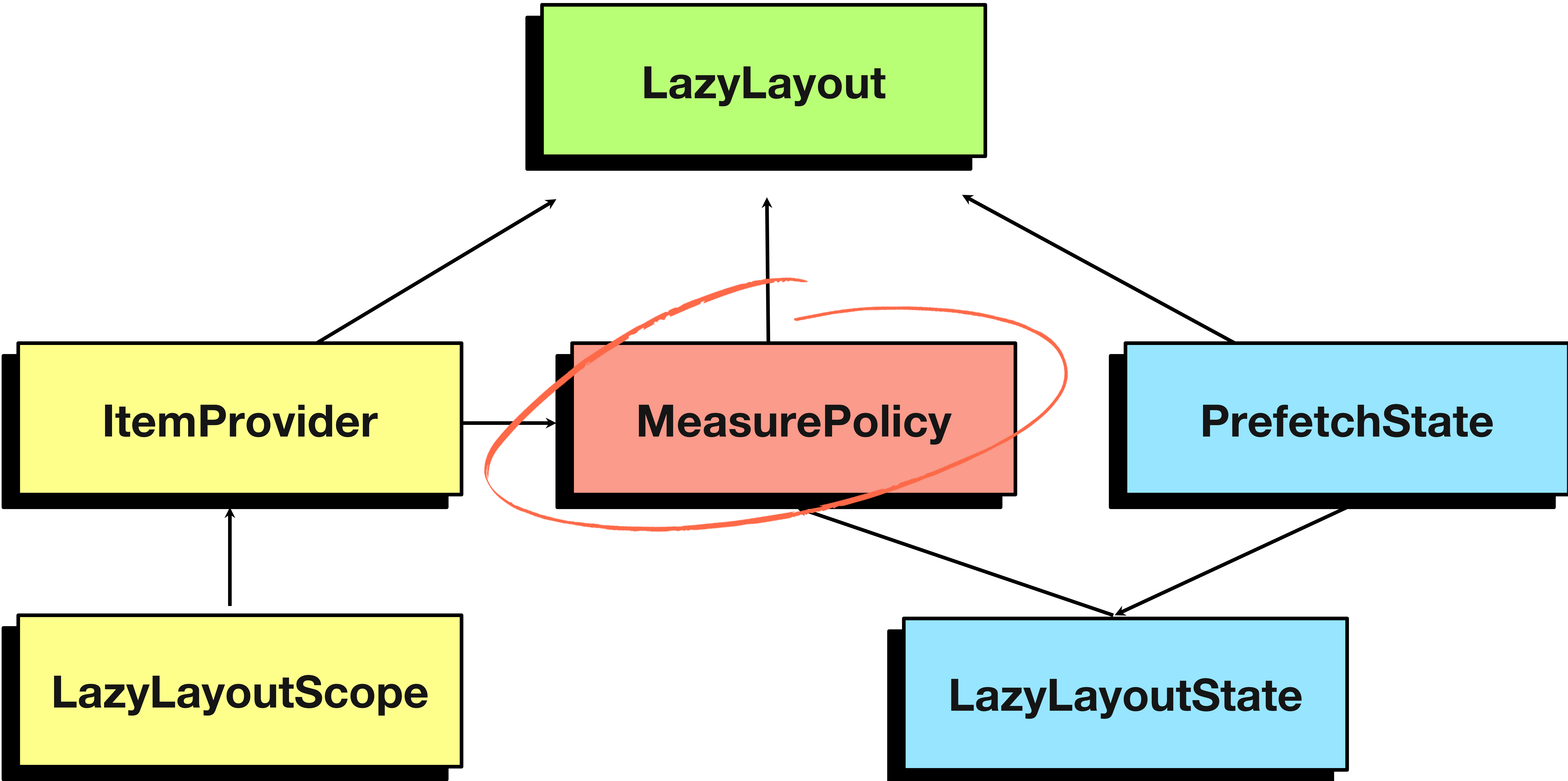
```
@Composable
fun CustomLazyLayout(
    content: CustomLazyListScope.() -> Unit,
) {
    val itemProvider = rememberItemProvider(content)
    LazyLayout(
        modifier = Modifier.pointerInput(Unit) {
            detectDragGestures { change, dragAmount ->
                change.consume()
                ...
            }
        },
        itemProvider = itemProvider,
    )
}
```


LazyLayout
ItemProvider
MeasurePolicy
PrefetchState
LazyLayoutScope
LazyLayoutState

```kotlin
@Stable
class CustomLazyLayoutState {

   val offsetState: State<IntOffset>

   fun onDrag(offset: IntOffset) {
       // update offsetState
   }
}
```

```
@Composable
fun CustomLazyLayout(
    content: CustomLazyListScope.() -> Unit,
) {
    val itemProvider = rememberItemProvider(content)
    val state = rememberLazyLayoutState()
    LazyLayout(
        modifier = Modifier.pointerInput(Unit) {
            detectDragGestures { change, dragAmount ->
                change.consume()
                state.onDrag(dragAmount)
            }
        },
        itemProvider = itemProvider,
    )
}
```

Осталось определить, какие из айтемов видны на экране, и отобразить их


LazyLayout
ItemProvider
MeasurePolicy
PrefetchState
LazyLayoutScope
LazyLayoutState

```
@Stable
sealed interface LazyLayoutMeasureScope : MeasureScope {

    fun measure(index: Int, constraints: Constraints): List<Placeable>

}
```

```
LazyLayout(...) { constraints ->

    val boundaries = getBoundaries(constraints, state.offsetState.value)
    val indexes = itemProvider.getItemIndexesInRange(boundaries)

    val indexesWithPlaceables = indexes.associateWith {
        measure(it, Constraints())
    }

    layout(constraints.maxWidth, constraints.maxHeight) {
        indexesWithPlaceables.forEach { (index, placeables) ->
            val item = itemProvider.getItem(index)
            placeable.placeRelative(item.x, item.y)
        }
    }
}
```

```
LazyLayout(...) { constraints ->

    val boundaries = getBoundaries(constraints, state.offsetState.value)
    val indexes = itemProvider.getItemIndexesInRange(boundaries)

    val indexesWithPlaceables = indexes.associateWith {
        measure(it, Constraints())
    }

    layout(constraints.maxWidth, constraints.maxHeight) {
        indexesWithPlaceables.forEach { (index, placeables) ->
            val item = itemProvider.getItem(index)
            placeable.placeRelative(item.x, item.y)
        }
    }
}
```

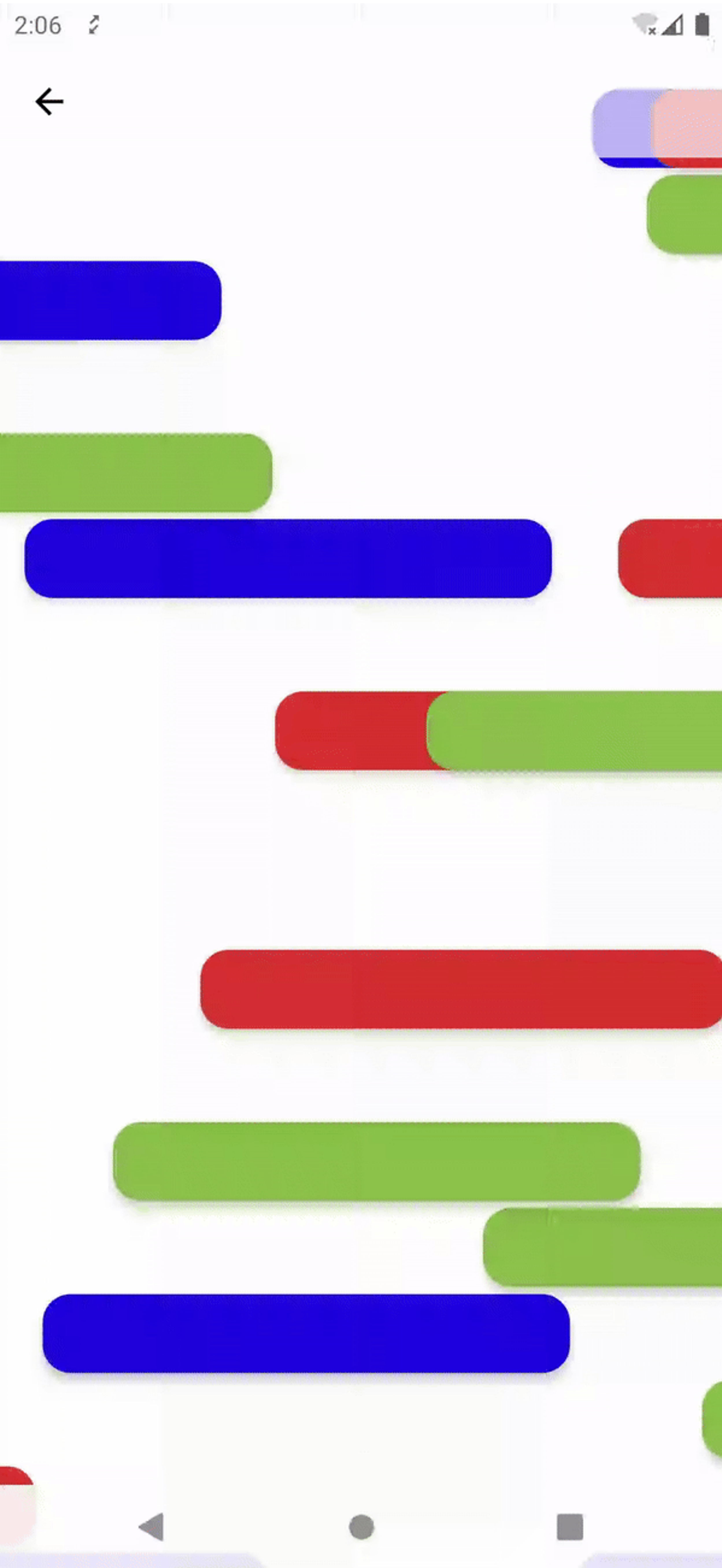
1. Находим видимые айтемы

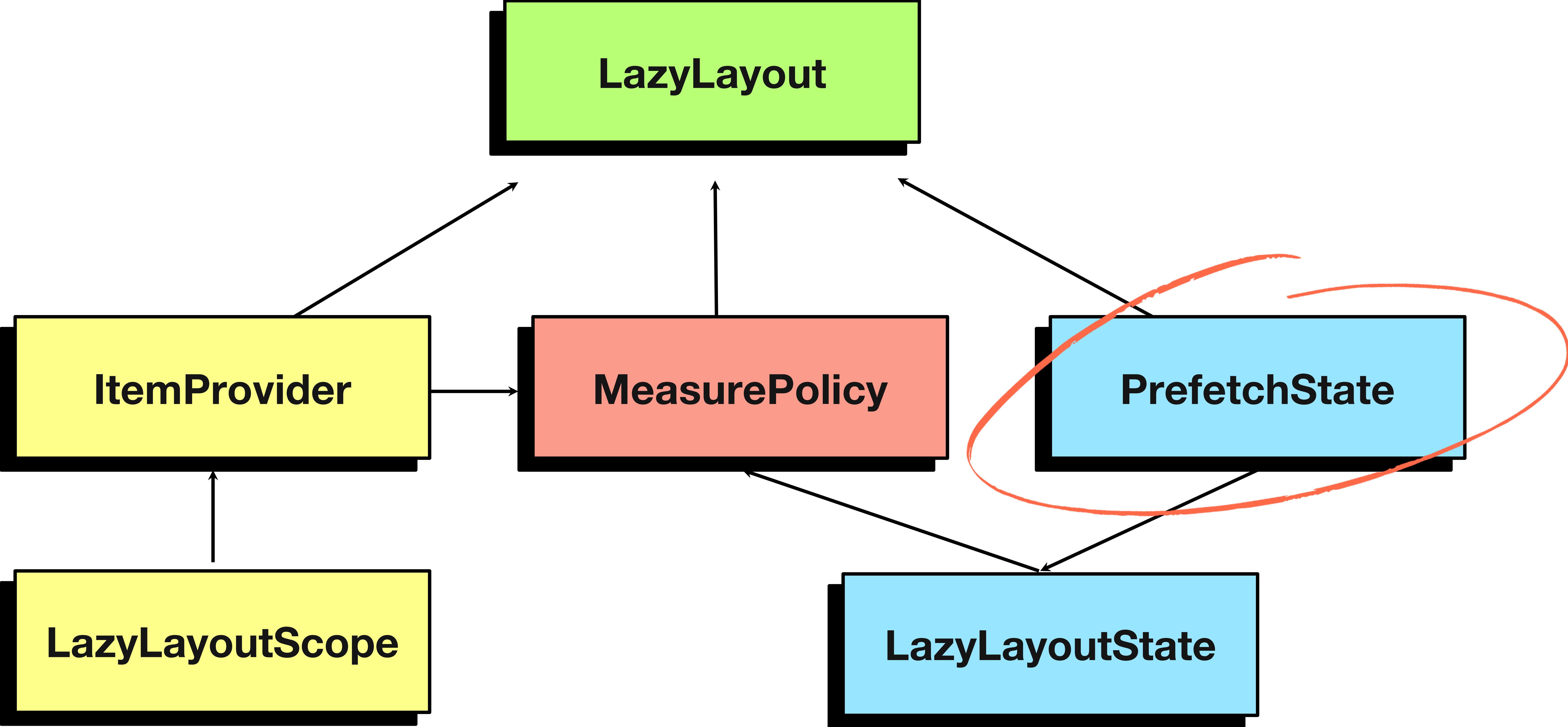
```
LazyLayout(...) { constraints ->

    val boundaries = getBoundaries(constraints, state.offsetState.value)
    val indexes = itemProvider.getItemIndexesInRange(boundaries)

    val indexesWithPlaceables = indexes.associateWith {
        measure(it, Constraints())
    }

    layout(constraints.maxWidth, constraints.maxHeight) {
        indexesWithPlaceables.forEach { (index, placeables) ->
            val item = itemProvider.getItem(index)
            placeable.placeRelative(item.x, item.y)
        }
    }
}
```

2. Измеряем айтемы (вызов subcompose под капотом)

```
LazyLayout(...) { constraints ->

    val boundaries = getBoundaries(constraints, state.offsetState.value)
    val indexes = itemProvider.getItemIndexesInRange(boundaries)

    val indexesWithPlaceables = indexes.associateWith {
        measure(it, Constraints())
    }

    layout(constraints.maxWidth, constraints.maxHeight) {
        indexesWithPlaceables.forEach { (index, placeables) ->
            val item = itemProvider.getItem(index)
            placeable.placeRelative(item.x, item.y)
        }
    }
}
```

3. Размещаем айтемы


2:06

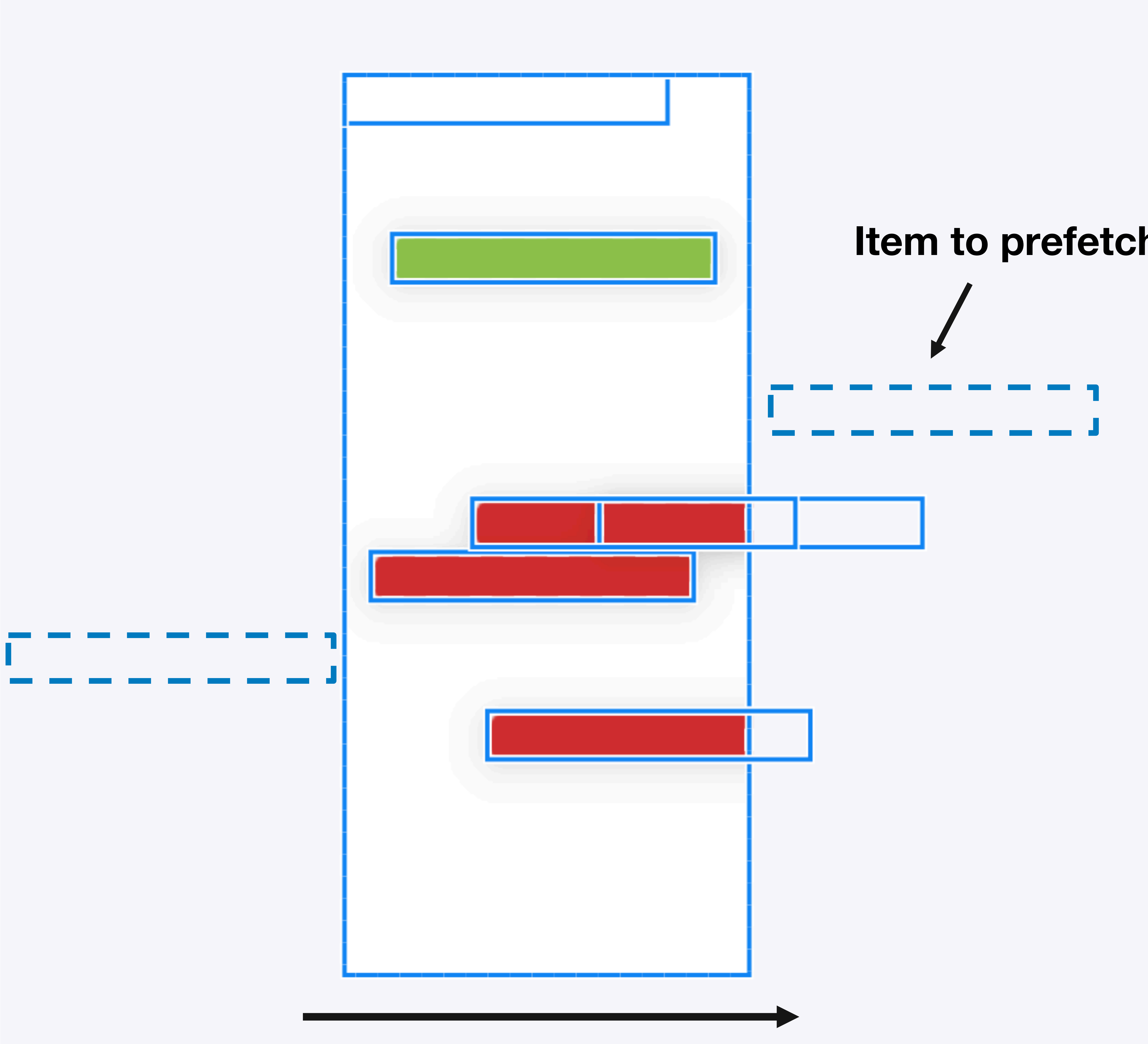

LazyLayout
ItemProvider
MeasurePolicy
PrefetchState
LazyLayoutScope
LazyLayoutState

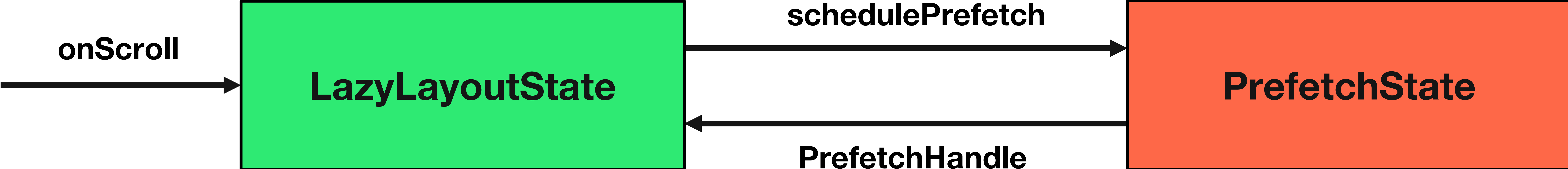
```kotlin
@ExperimentalFoundationApi
class LazyLayoutPrefetchState {

    fun schedulePrefetch(index: Int, constraints: Constraints): PrefetchHandle

    interface PrefetchHandle {
        fun cancel()
    }

}
```

~ RecyclerView.LayoutPrefetchRegistry

```kotlin
@ExperimentalFoundationApi
class LazyLayoutPrefetchState {

   fun schedulePrefetch(index: Int, constraints: Constraints): PrefetchHandle

   interface PrefetchHandle {
      fun cancel()
   }

}
```

```kotlin
@ExperimentalFoundationApi
class LazyLayoutPrefetchState {

   fun schedulePrefetch(index: Int, constraints: Constraints): PrefetchHandle

   interface PrefetchHandle {
      fun cancel()
   }

}
```


Item to prefetch


onScroll
LazyLayoutState
schedulePrefetch
PrefetchHandle
PrefetchState

Выводы

- [ ] **`Modifier.layout` – Изменяет layout отдельного компонента**
- [ ] `fun Layout()` – Кастомный layout для нескольких компонентов
- [ ] `@Deprecated fun MultiMeasureLayout` – Используйте Intrinsics вместо него!
- [ ] `fun SubComposeLayout()` – Откладывает композицию контента
- [ ] `fun LazyLayout()` – Кастомный lazy layout

- [ ] `Modifier.layout` – Изменяет layout отдельного компонента
- [ ] **`fun Layout()` – Кастомный layout для нескольких компонентов**
- [ ] `@Deprecated fun MultiMeasureLayout` – Используйте Intrinsics вместо него!
- [ ] `fun SubComposeLayout()` – Откладывает композицию контента
- [ ] `fun LazyLayout()` – Кастомный lazy layout

- [ ] `Modifier.layout` – Изменяет layout отдельного компонента
- [ ] `fun Layout()` – Кастомный layout для нескольких компонентов
- [ ] **`@Deprecated fun MultiMeasureLayout` – Используйте Intrinsics вместо него!**
- [ ] `fun SubComposeLayout()` – Откладывает композицию контента
- [ ] `fun LazyLayout()` – Кастомный lazy layout

- [ ] `Modifier.layout` – Изменяет layout отдельного компонента
- [ ] `fun Layout()` – Кастомный layout для нескольких компонентов
- [ ] `@Deprecated fun MultiMeasureLayout` – Используйте Intrinsics вместо него!
- [ ] **`fun SubComposeLayout()` – Откладывает композицию контента**
- [ ] `fun LazyLayout()` – Кастомный lazy layout

- [ ] `Modifier.layout` – Изменяет layout отдельного компонента
- [ ] `fun Layout()` – Кастомный layout для нескольких компонентов
- [ ] `@Deprecated fun MultiMeasureLayout` – Используйте Intrinsics вместо него!
- [ ] `fun SubComposeLayout()` – Откладывает композицию контента
- [ ] **`fun LazyLayout()` – Кастомный lazy layout**

Пишите кастомные лейауты в Jetpack Compose!

In/avvlas

@avvlased