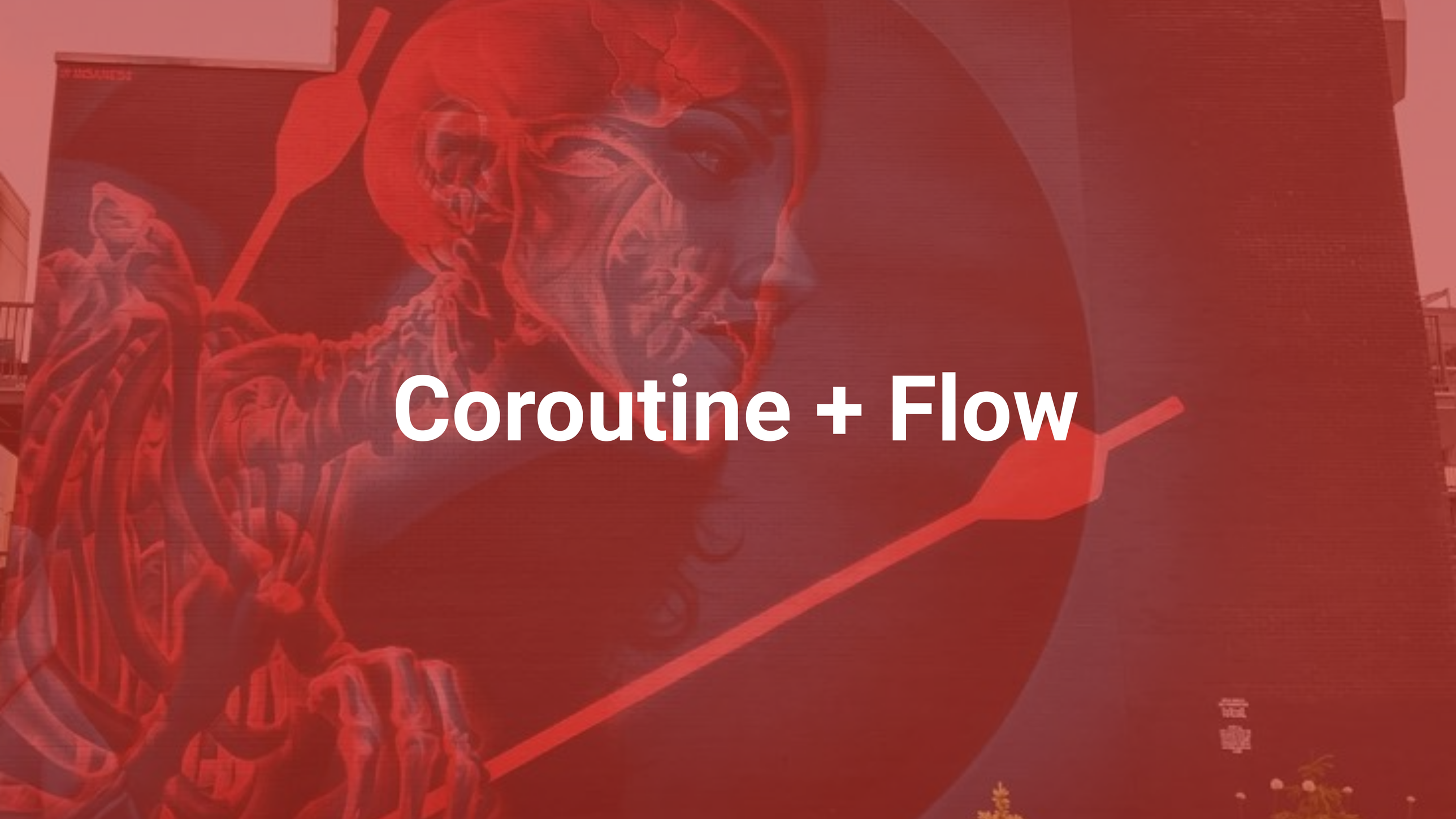
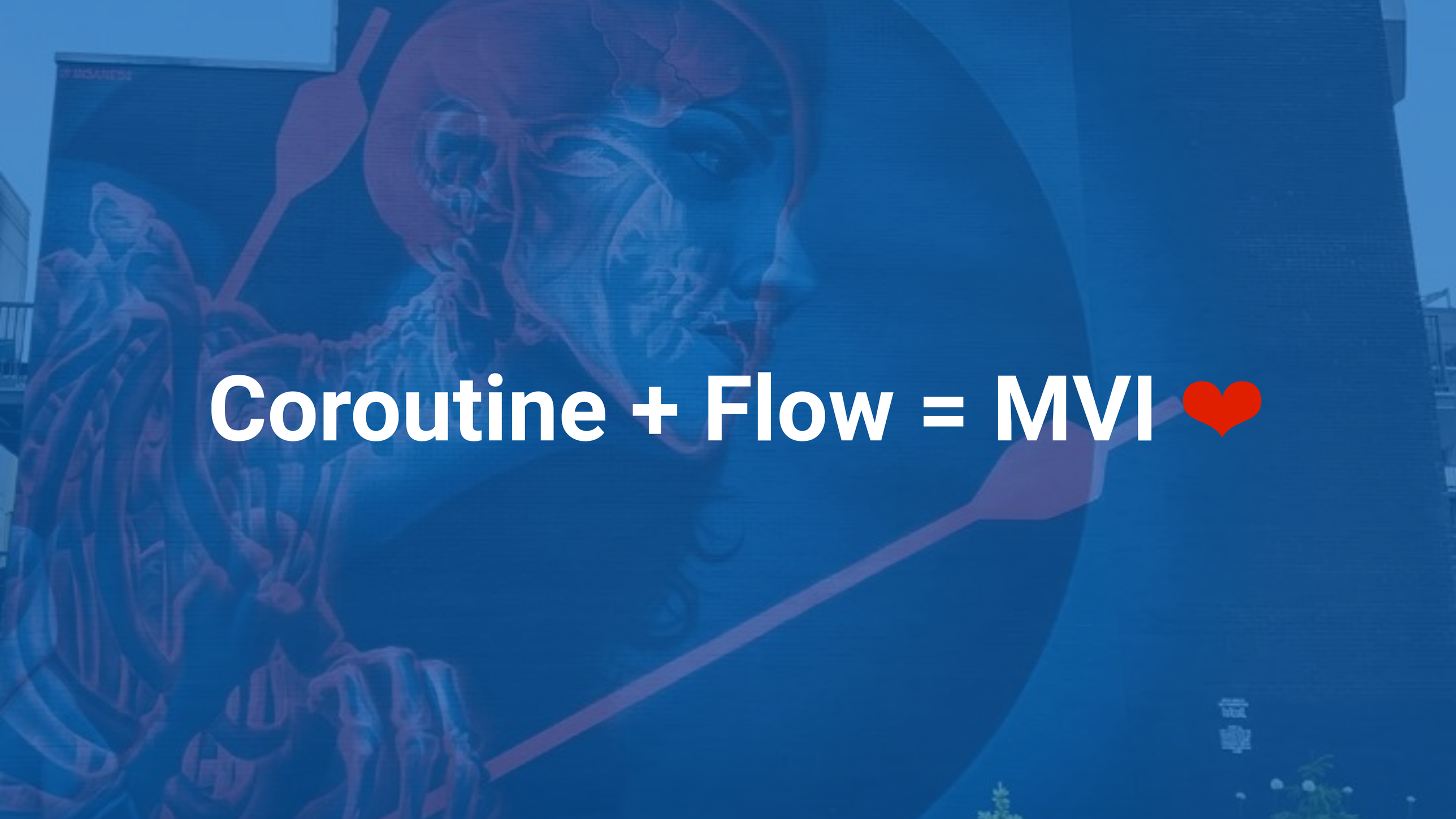
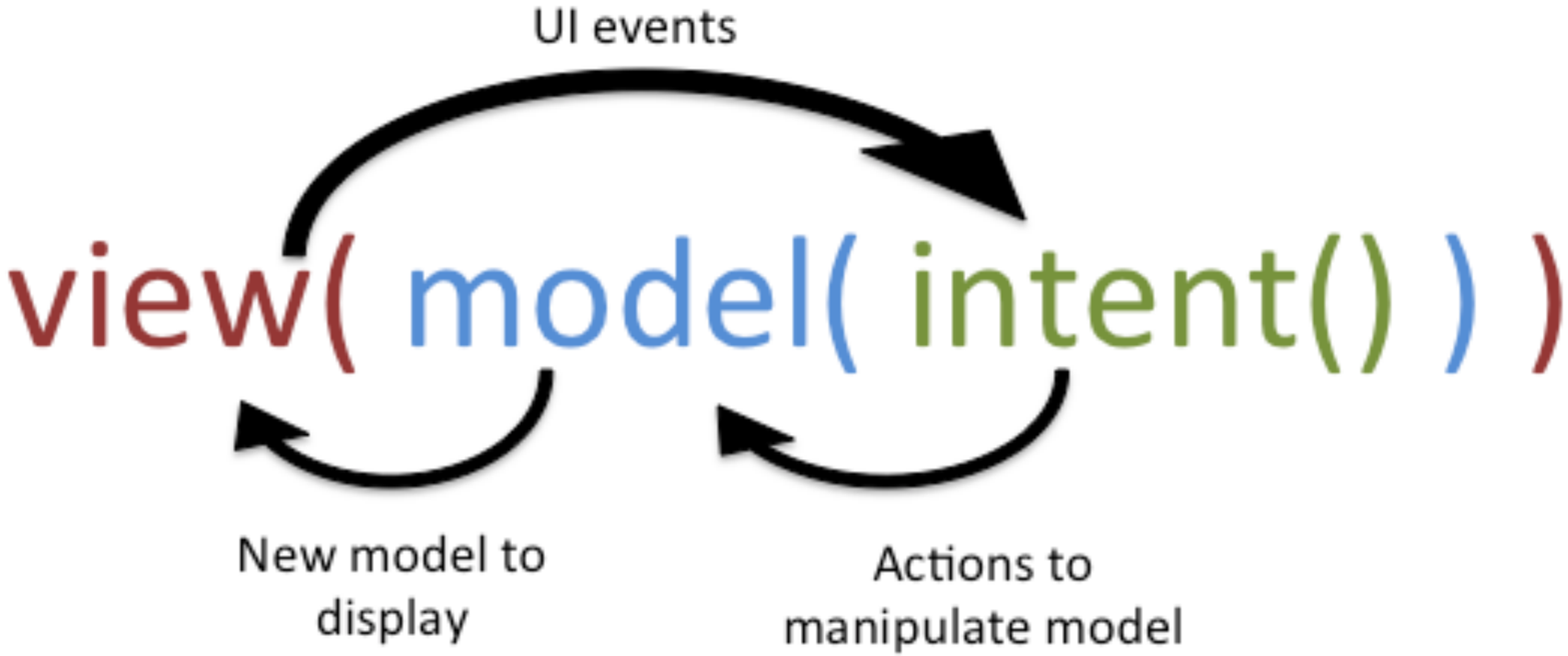
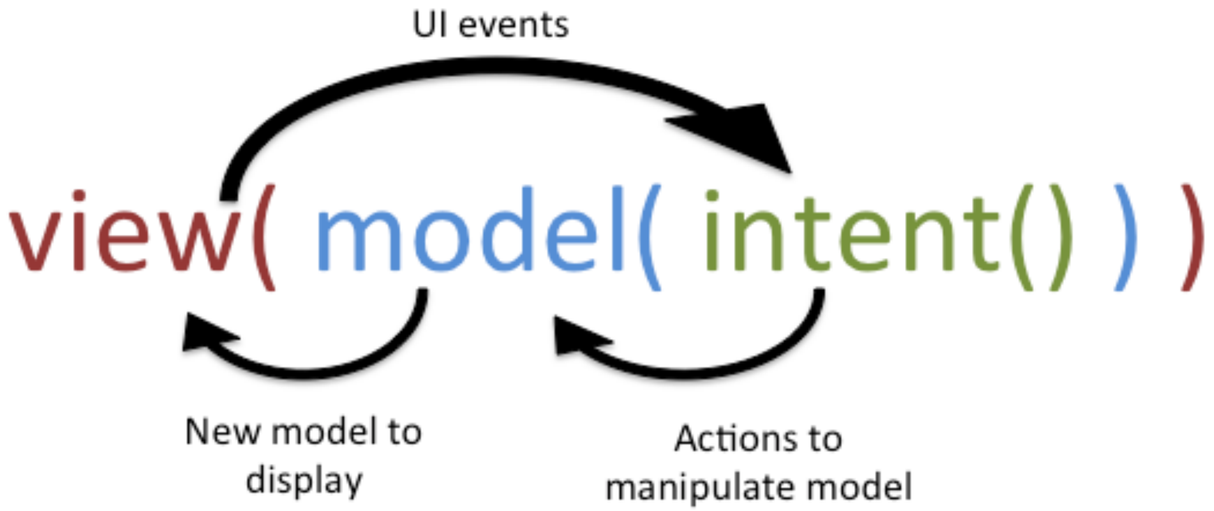


Coroutine + Flow

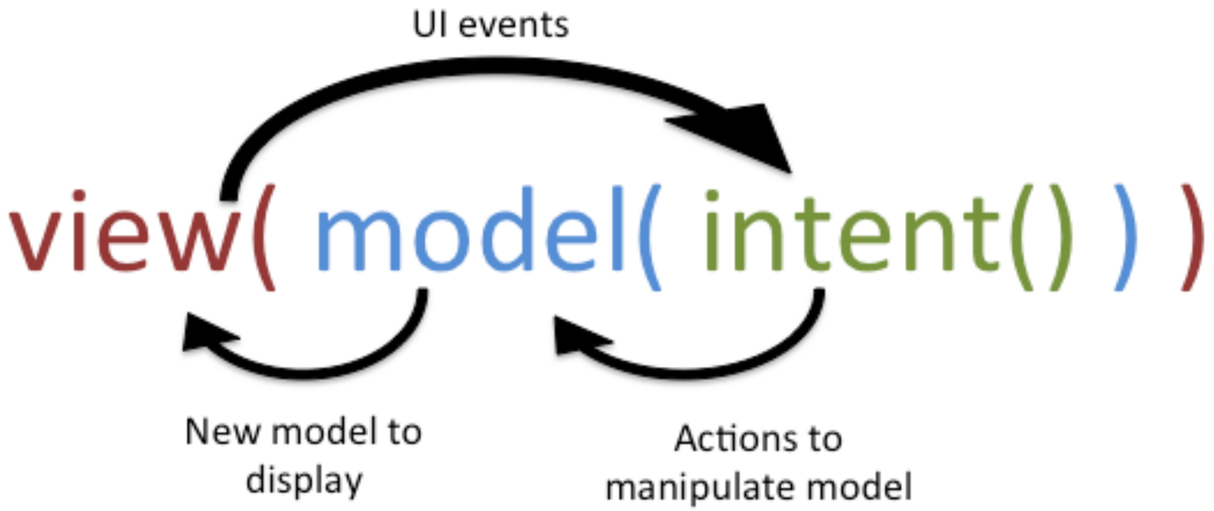

Coroutine + Flow = MVI ❤



http://hannesdorfmann.com/android/mosby3-mvi-1

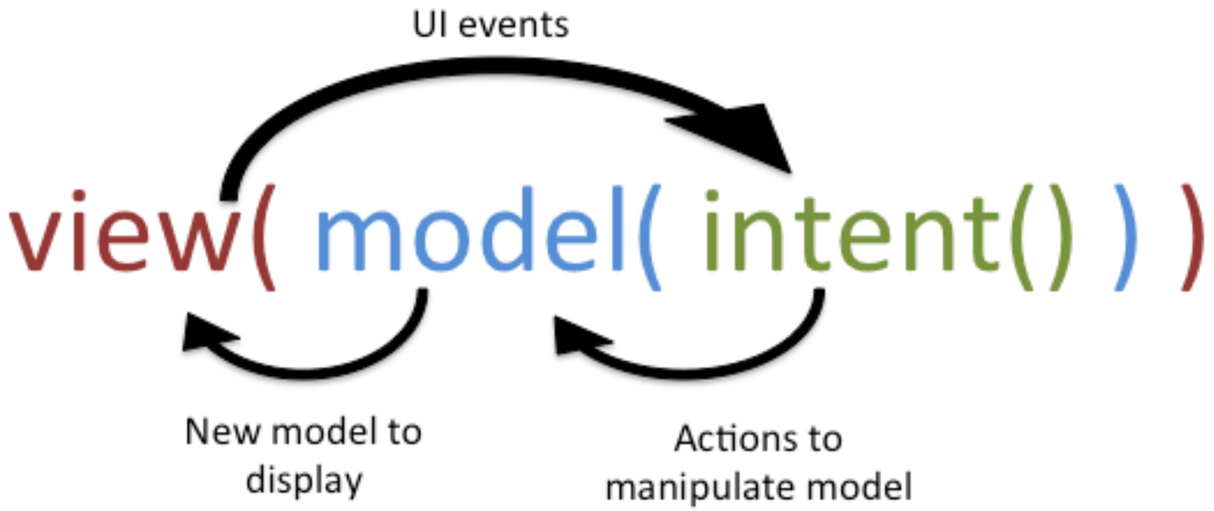

UI events
view(model(intent()))
New model to display
Actions to manipulate model

$$f(x)$$



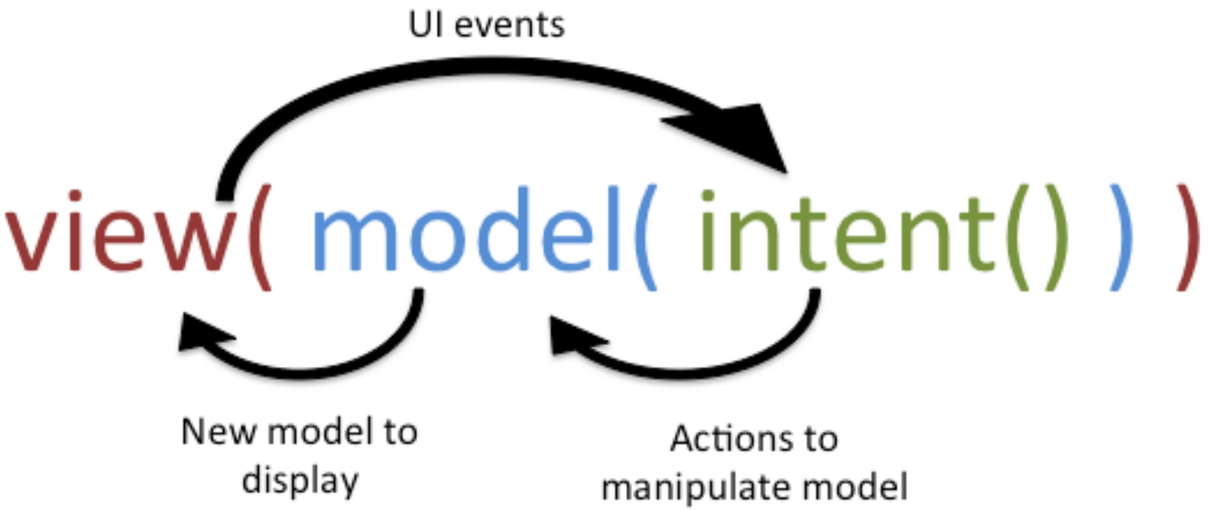
$$f(x)$$

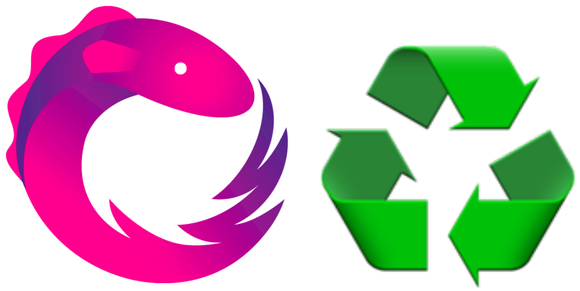
https://xkcd.com/1312



$$f(x)$$

https://xkcd.com/1312


UI events
view(model(intent()))
New model to display
Actions to manipulate model



```
private val disposables = CompositeDisposable()

disposables += streamA().subscribe( )
disposables += streamB().subscribe( )

disposables.clear()
```

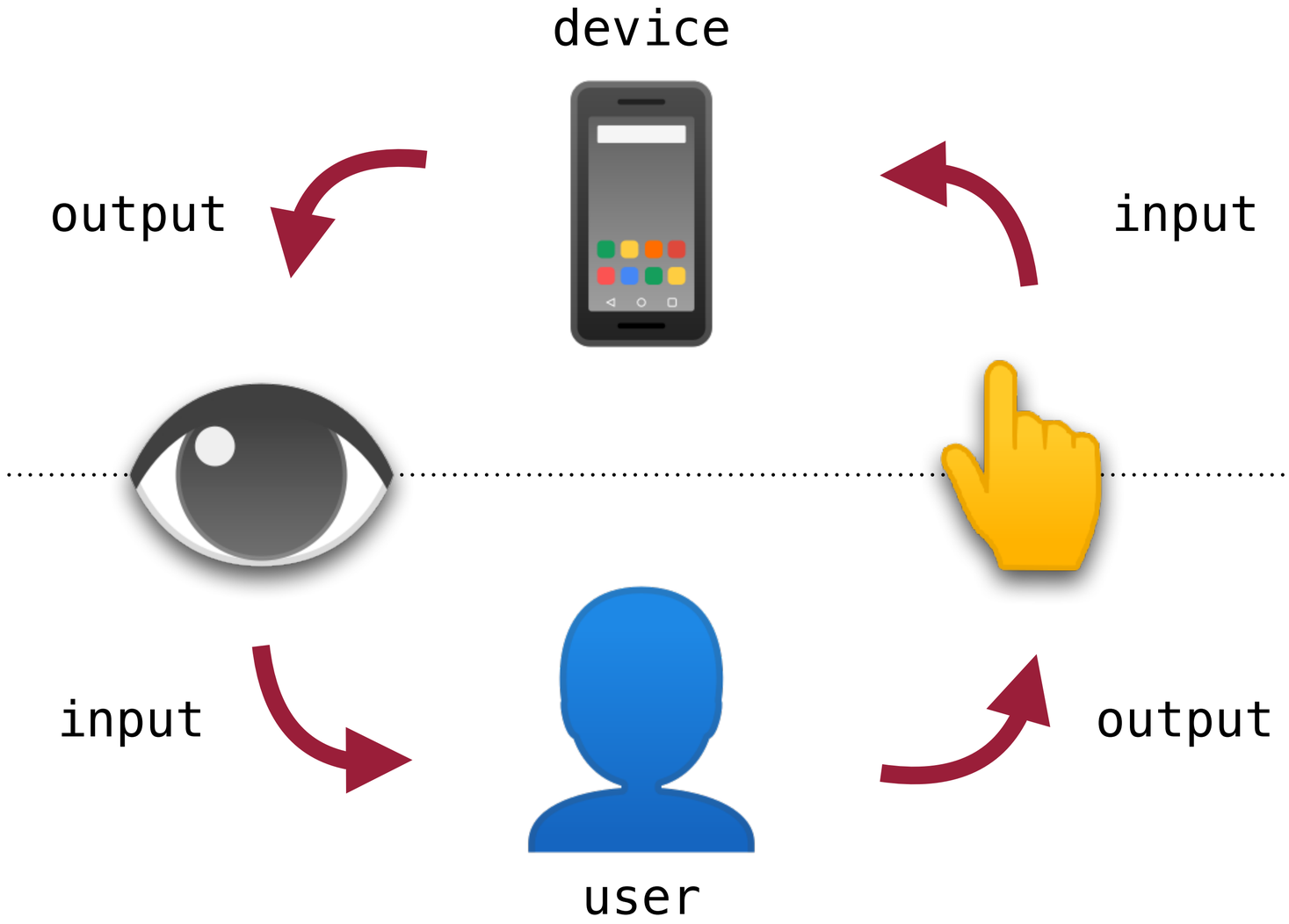
Coroutines & Flow

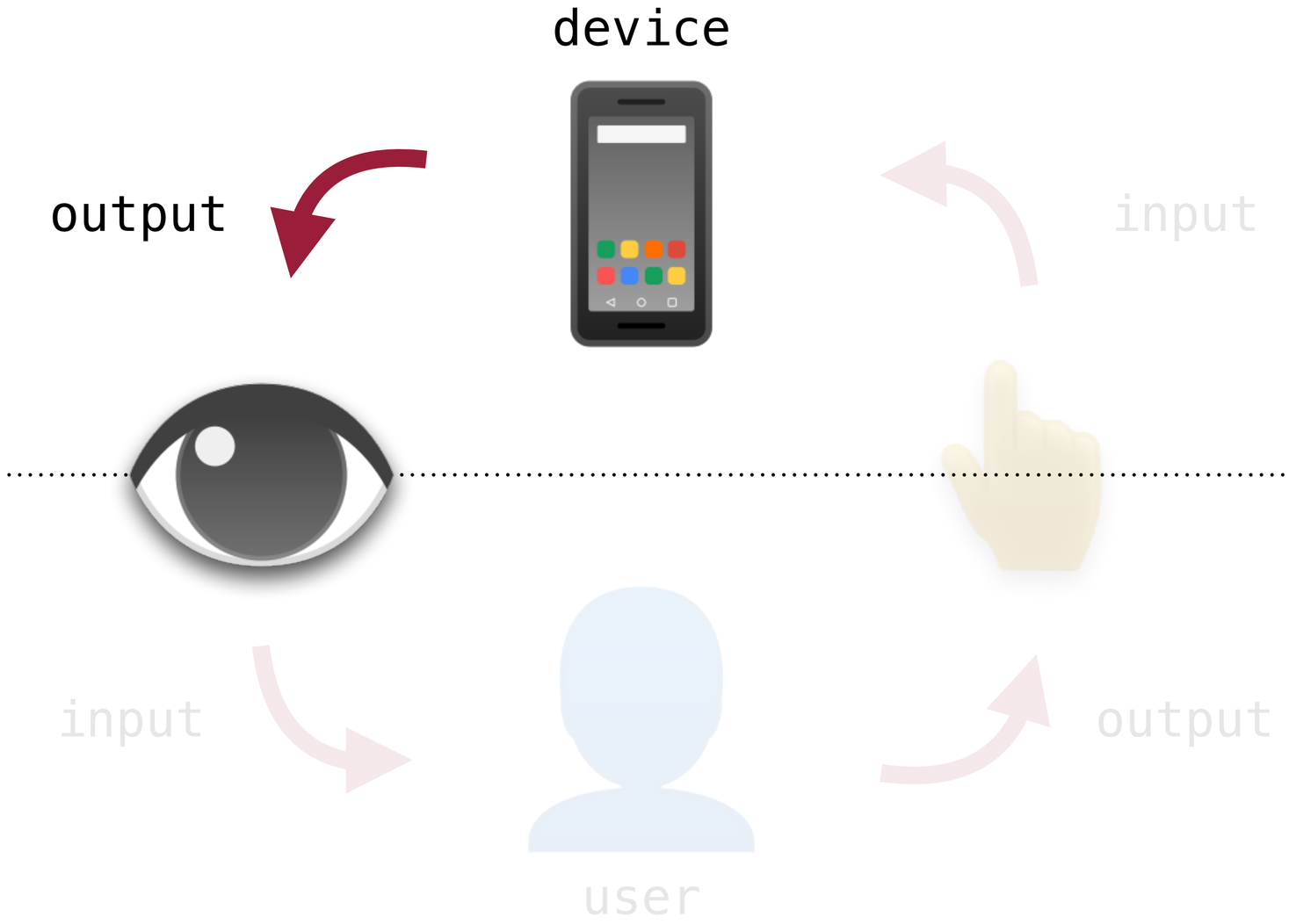
Coroutines & Flow

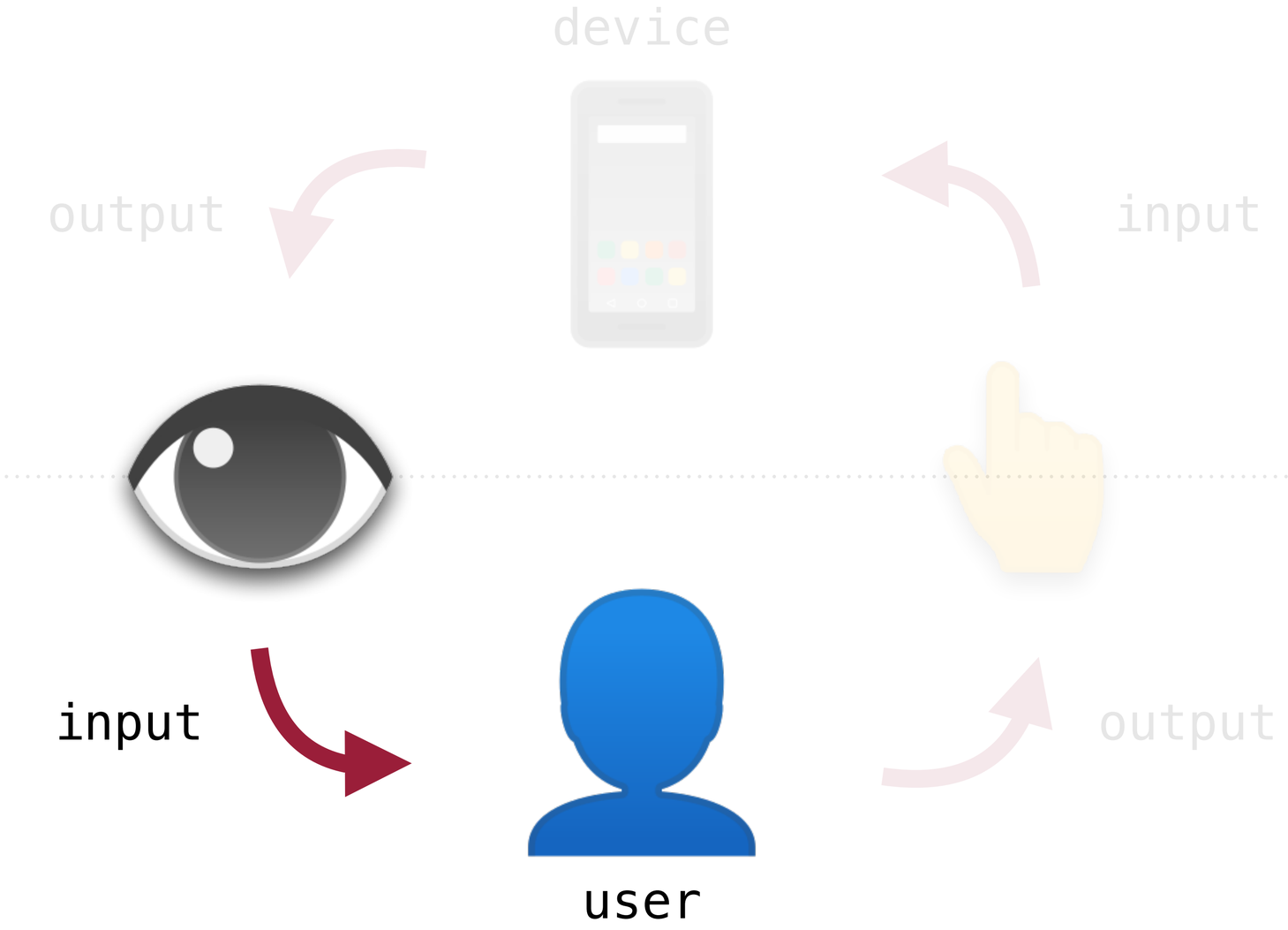
kotlinx-coroutines-android-1.3.0.jar 21 KB
kotlinx-coroutines-core-common-1.3.0.jar 137 KB

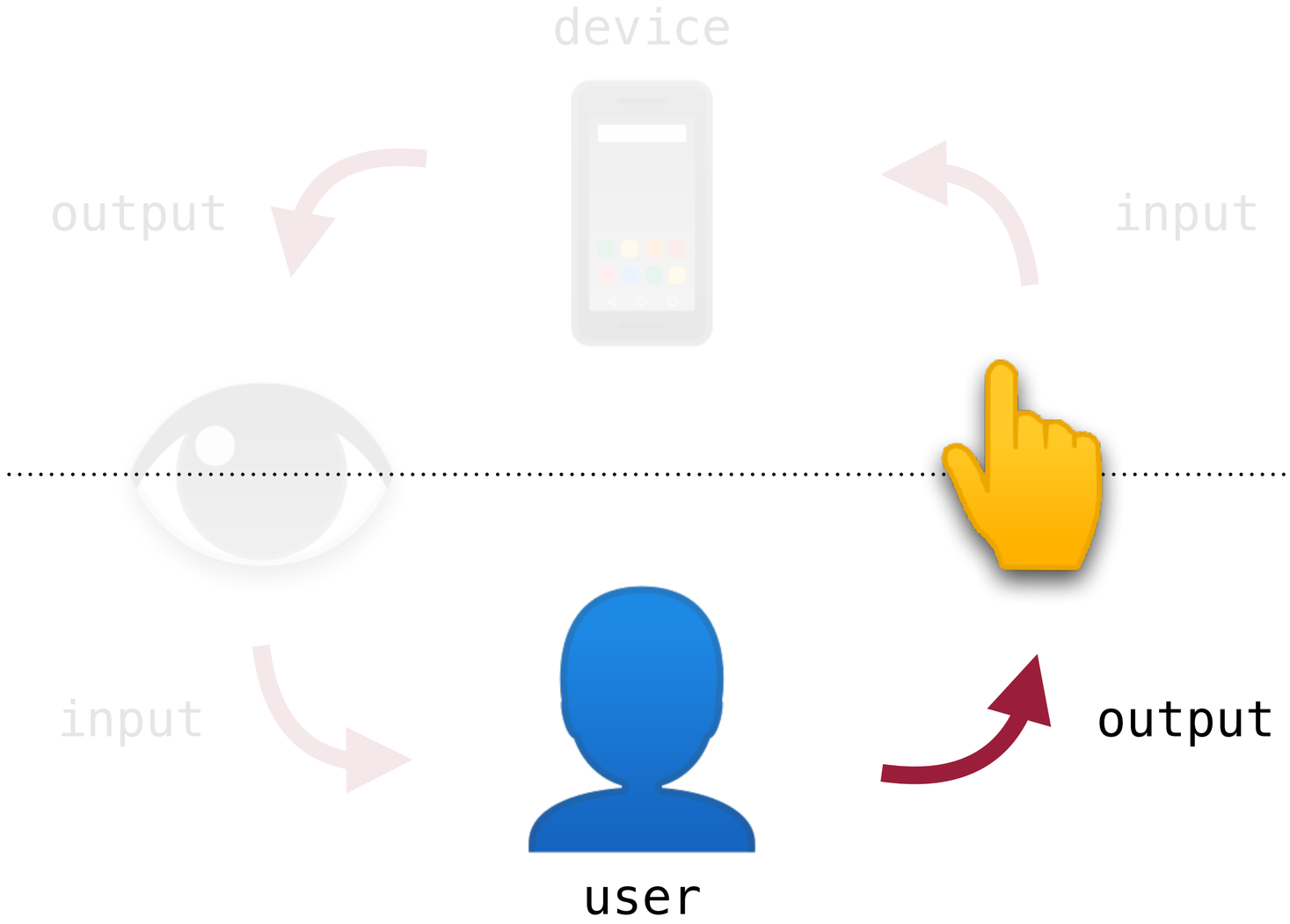
Coroutines & Flow

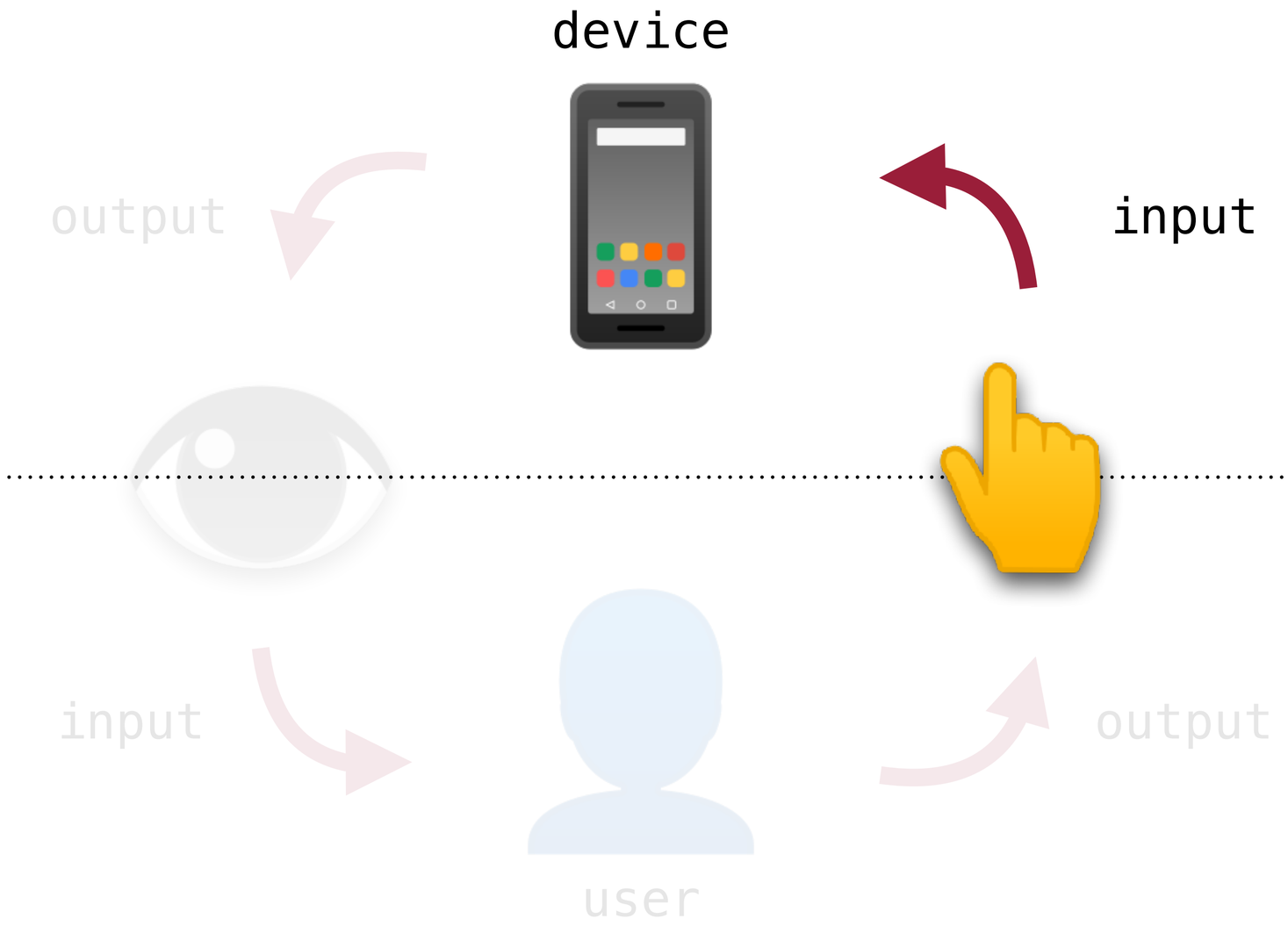
Coroutines & Flow

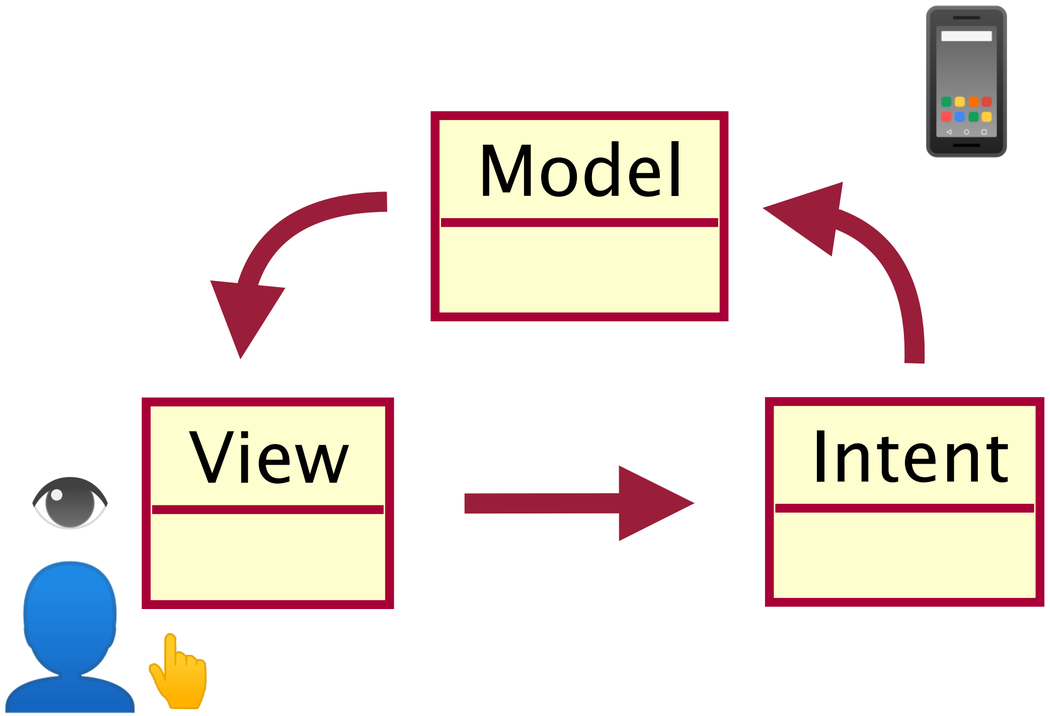

device
output
input
input
output
user

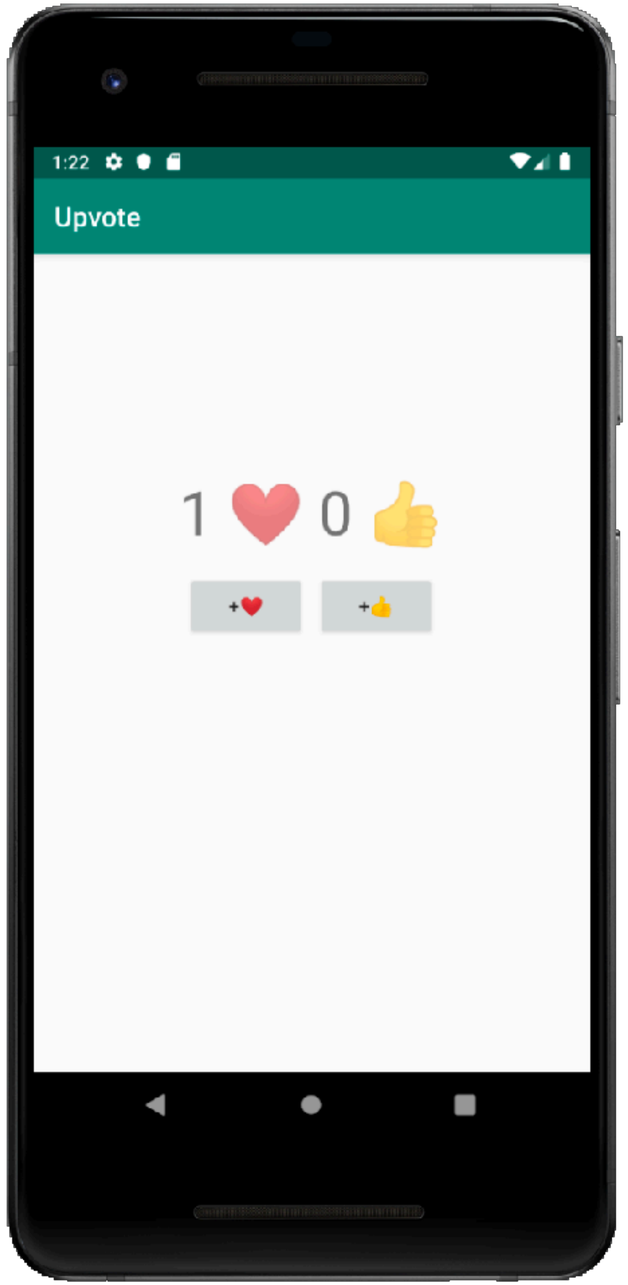

device
output
input
input
output
user

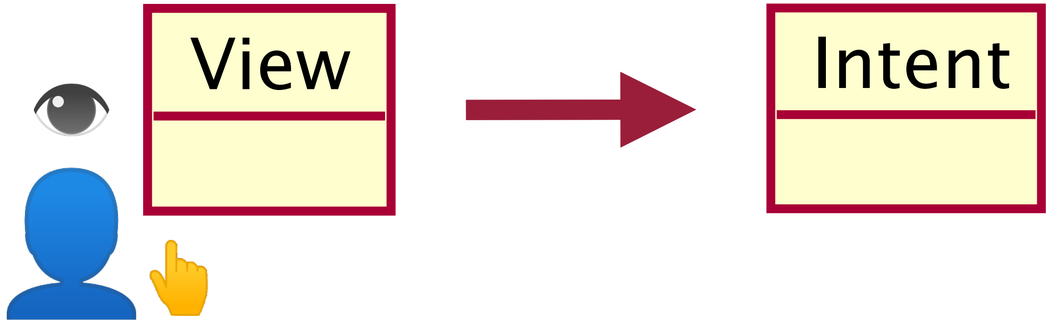

device
output
input
input
output
user

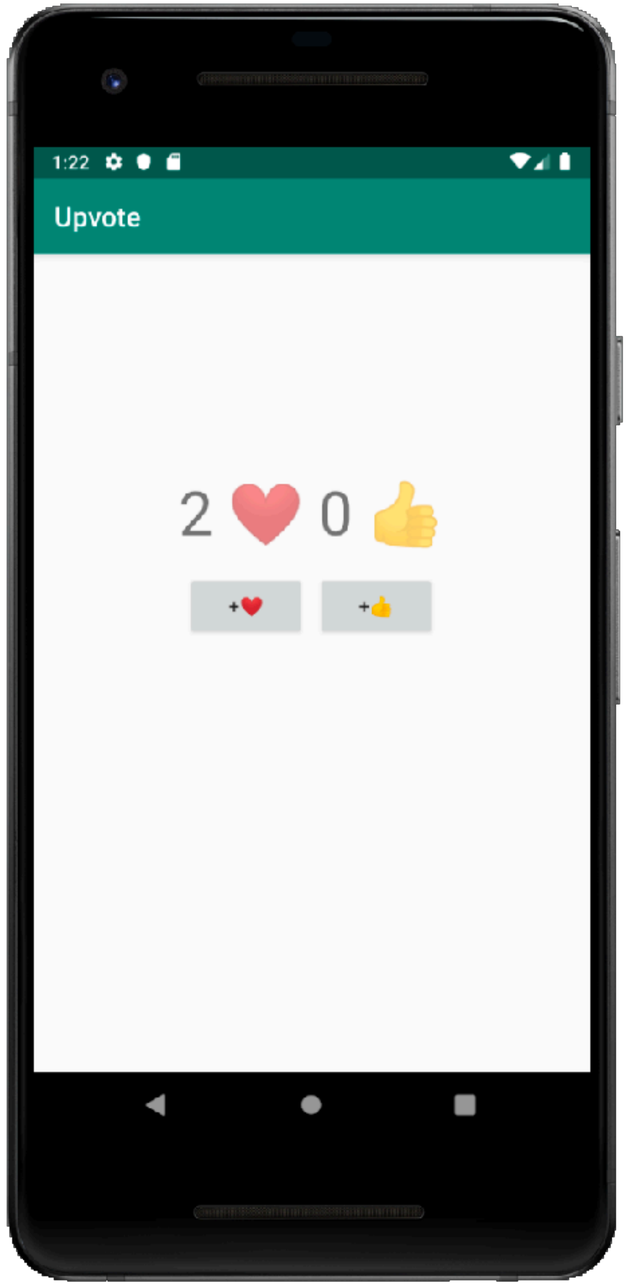

device
output
input
input
output
user

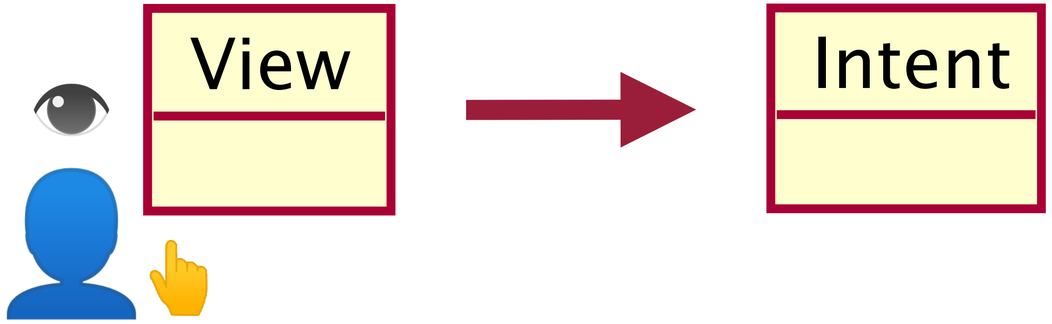

device
output
input
input
output
user

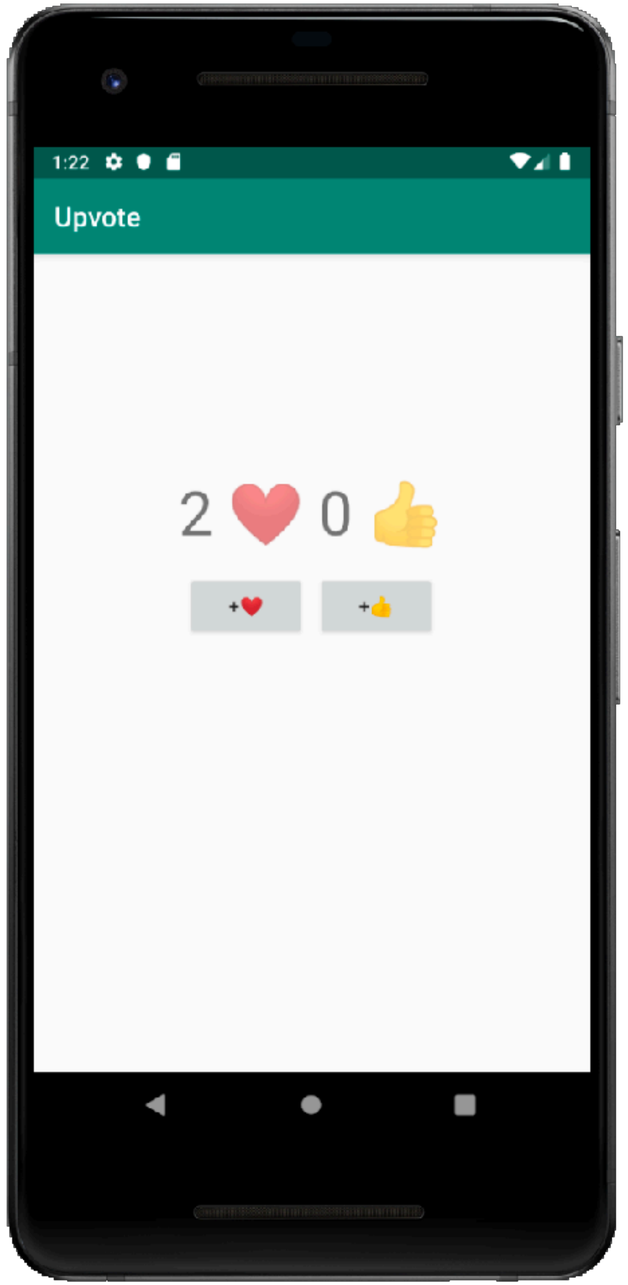

Model
View
Intent

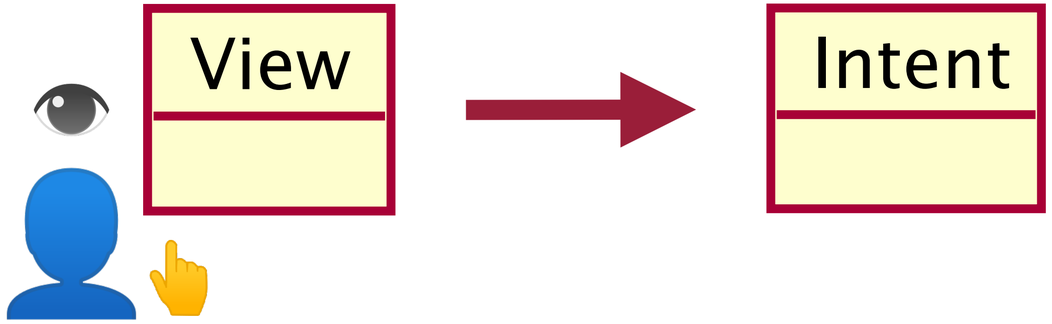

1:22
Upvote
1
0
+
+

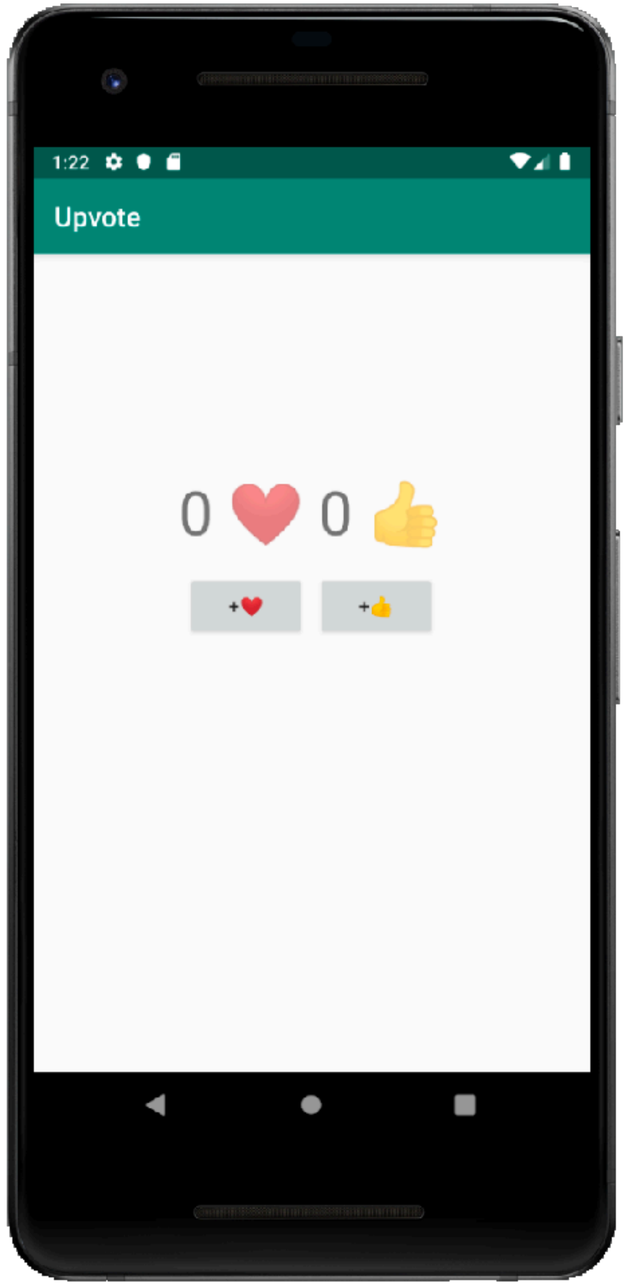

View
Intent


1:22
Upvote
2
0
+
+

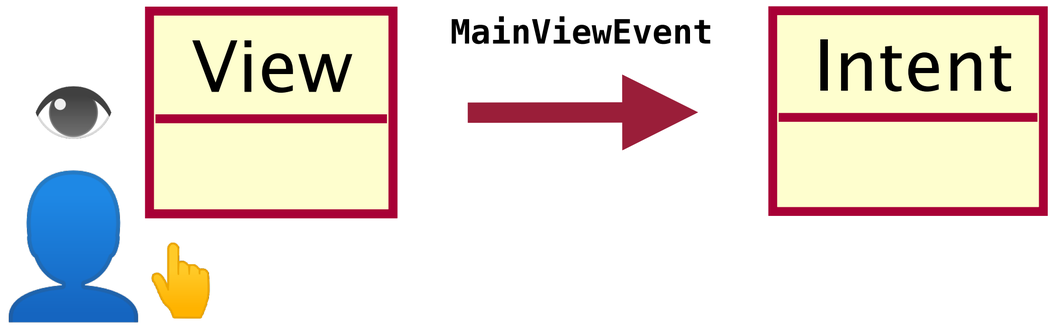

View
Intent

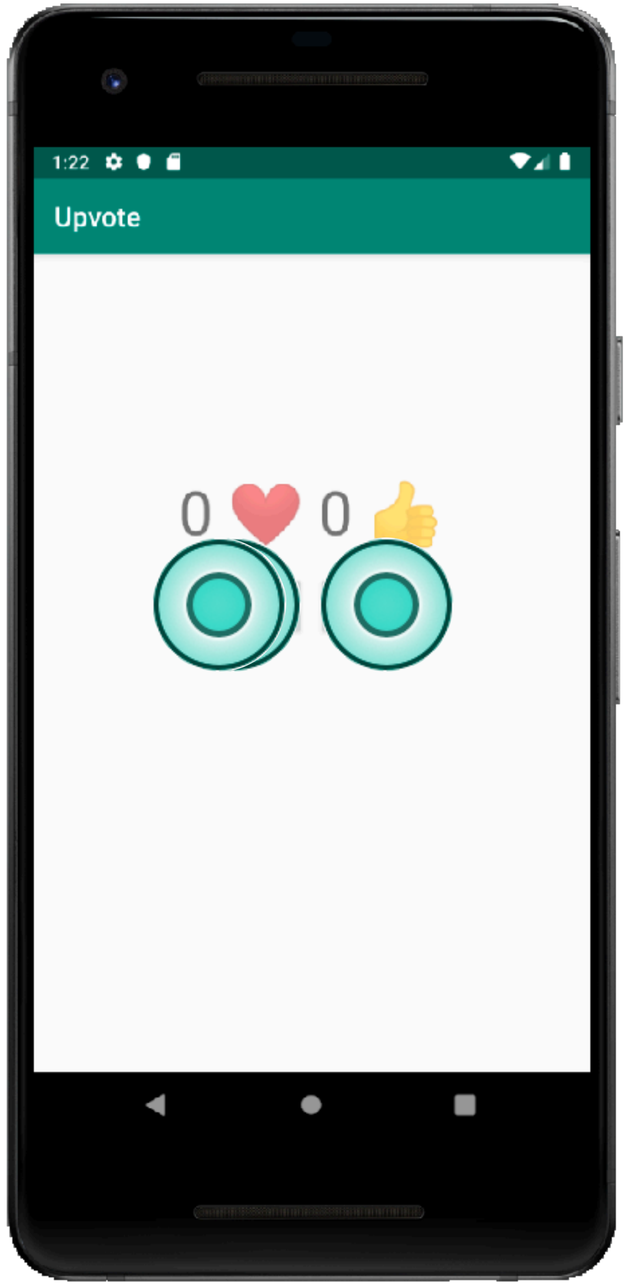

1:22
Upvote
2
0
+
+


View
Intent



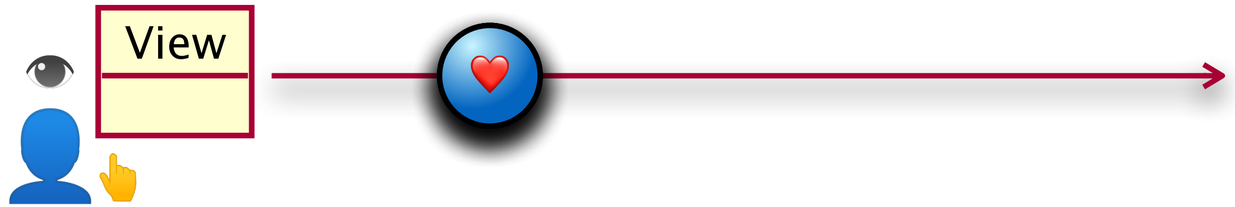

```
sealed class MainViewEvent {
    object ThumbsUpClick : MainViewEvent()
    object LoveItClick : MainViewEvent()
}
```

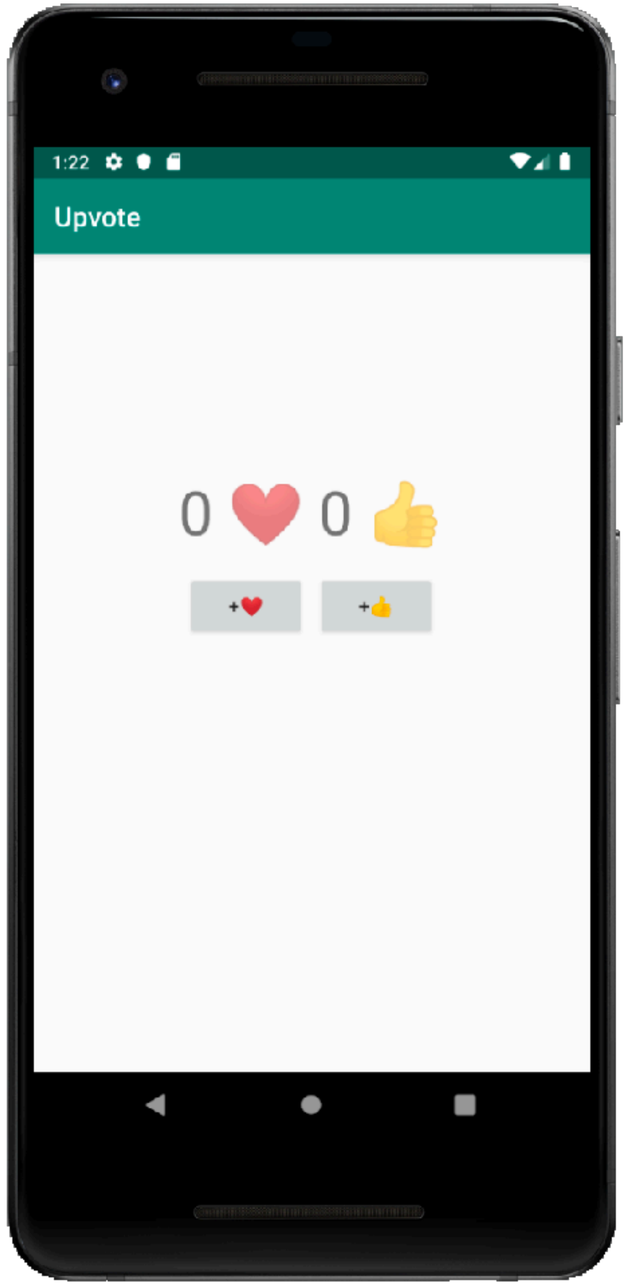




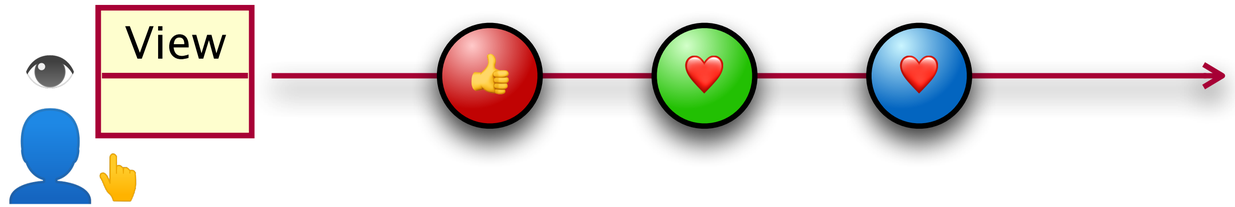

```
sealed class MainViewEvent {
    object ThumbsUpClick : MainViewEvent()
    object LoveItClick : MainViewEvent()
}
```

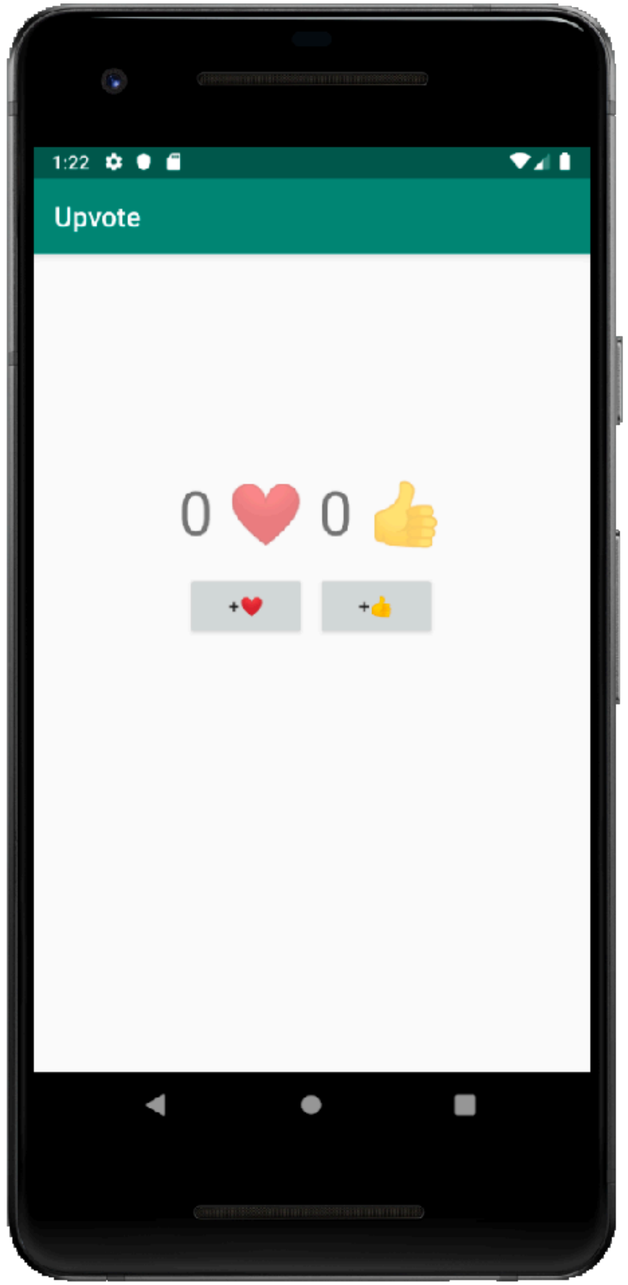
- MainViewEvent
  - +👍 ThumbsUpClick
  - +❤️ LoveItClick



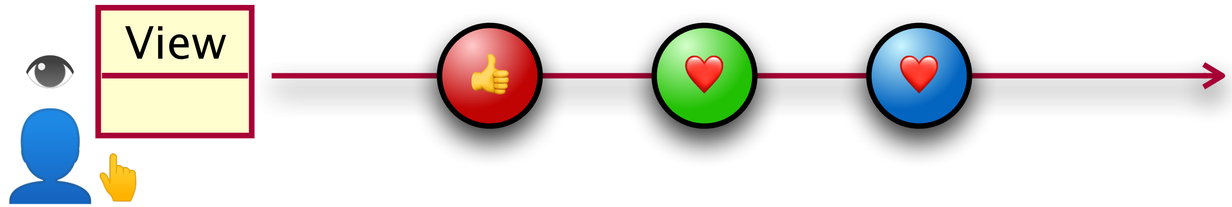


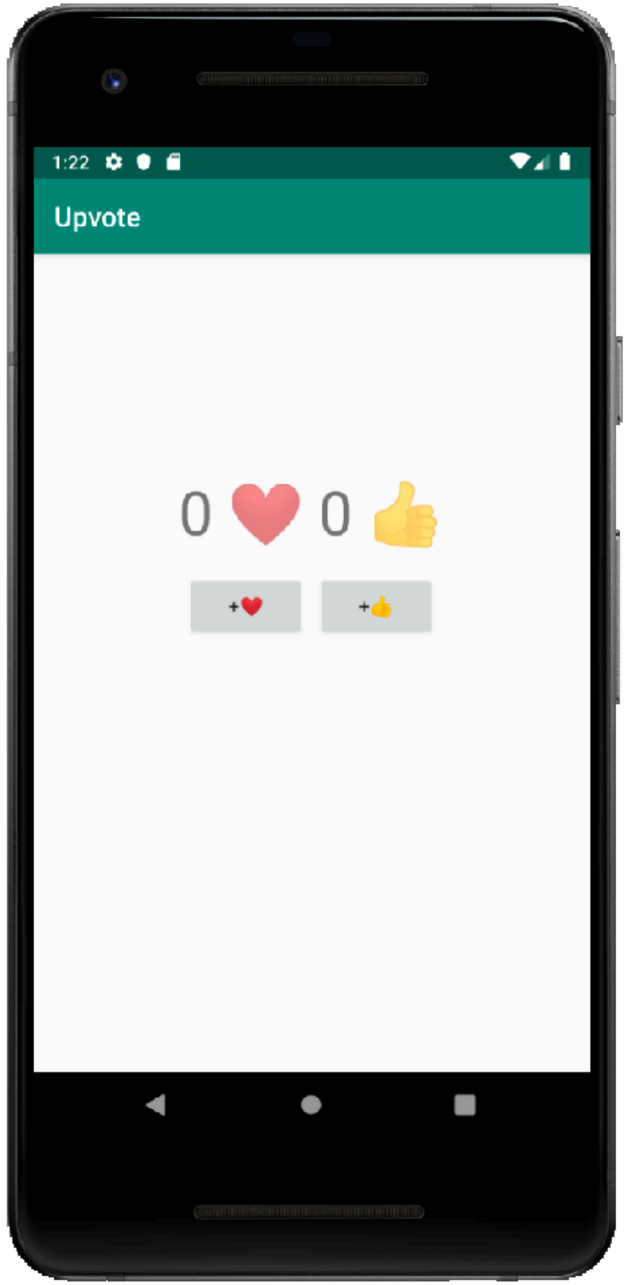

```
fun View.clicks(): Flow<Unit> = callbackFlow { this: ProducerScope<Unit>
    val listener = View.OnClickListener { offer(Unit) }
    setOnClickListener(listener)
    awaitClose {
        setOnClickListener(null)
    }
}
```



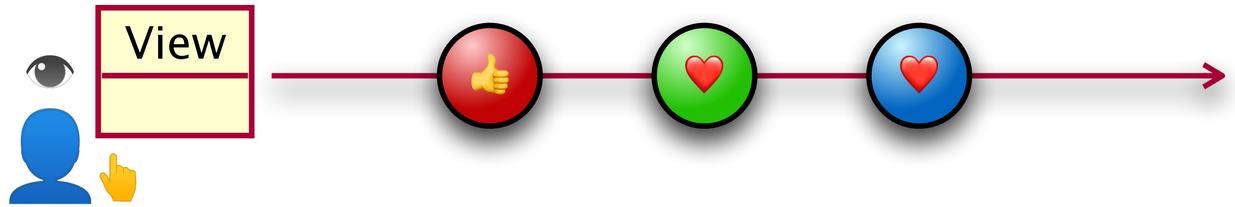


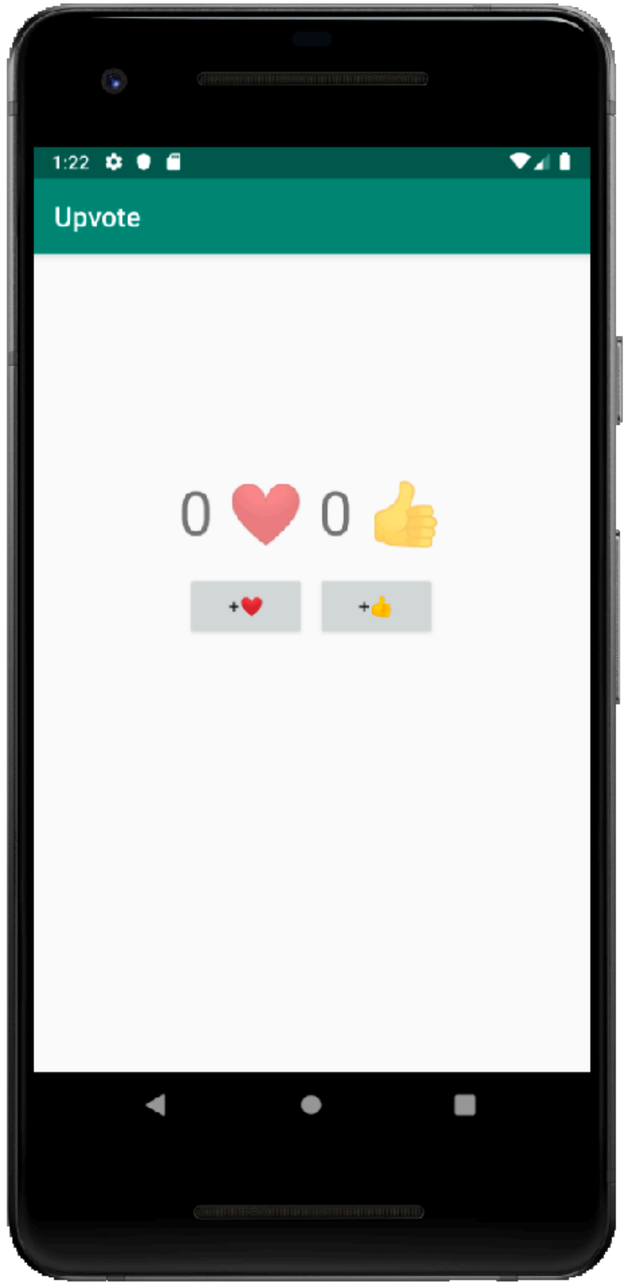

```
fun View.clicks(): Flow<Unit> = callbackFlow { this: ProducerScope<Unit>
    val listener = View.OnClickListener { offer(Unit) }
    setOnClickListener(listener)
    awaitClose {
        setOnClickListener(null)
    }
}
```



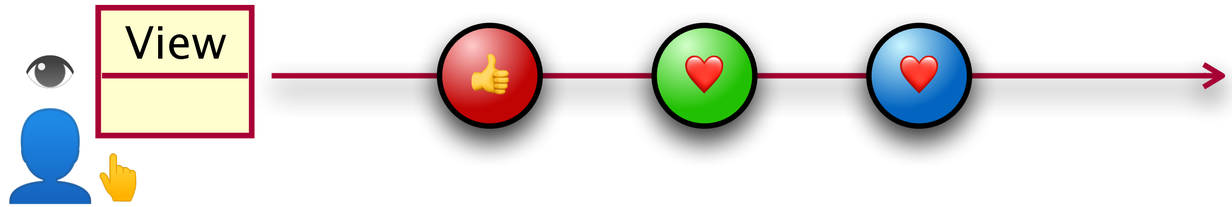


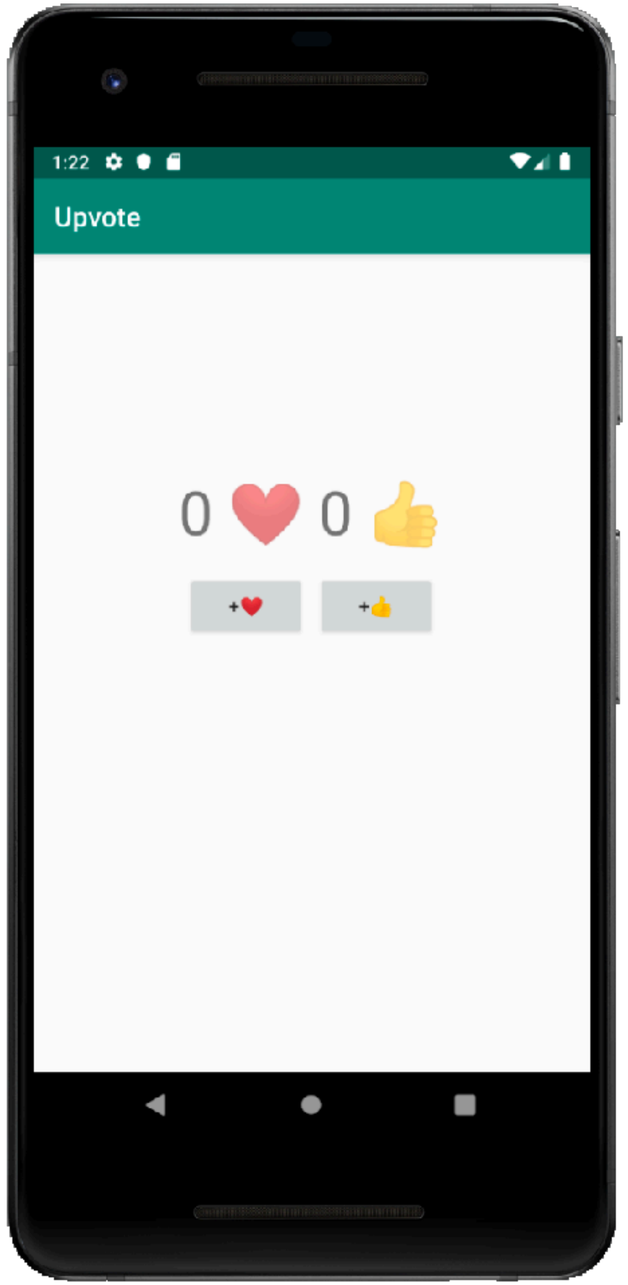

```
fun View.clicks(): Flow<Unit> = callbackFlow { this: ProducerScope<Unit>
    val listener = View.OnClickListener { offer(Unit) }
    setOnClickListener(listener)
    awaitClose {
        setOnClickListener(null)
    }
}
```

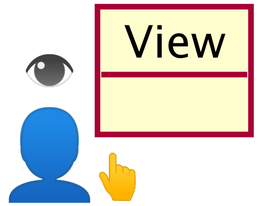


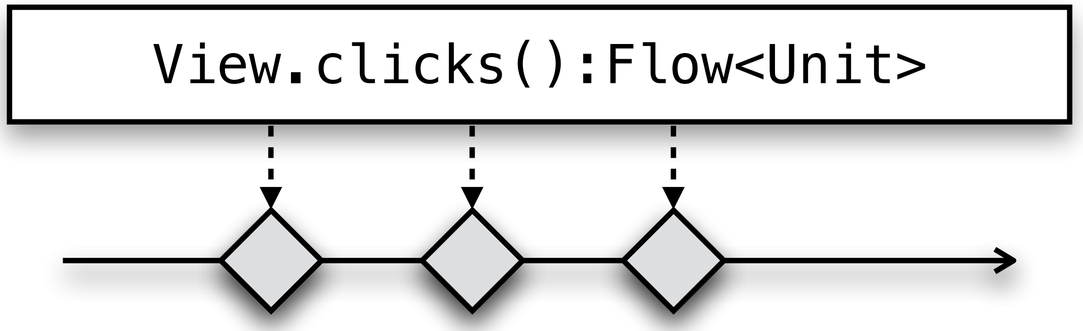


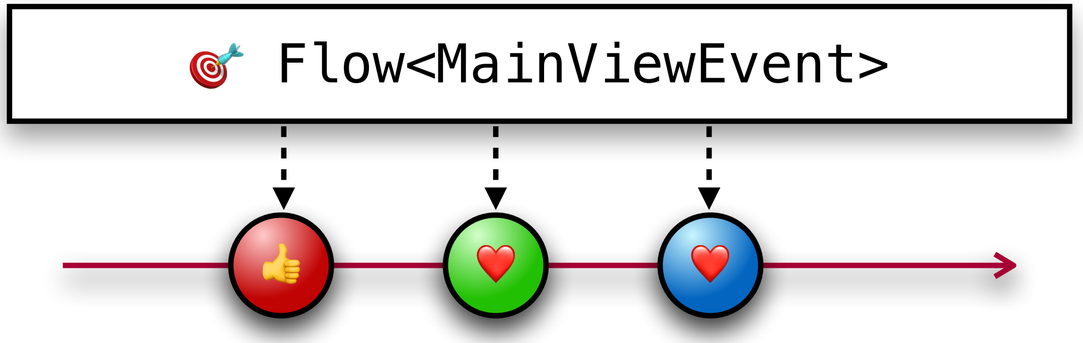

```
fun View.clicks(): Flow<Unit> = callbackFlow { this: ProducerScope<Unit>
    val listener = View.OnClickListener { offer(Unit) }
    setOnClickListener(listener)
    awaitClose {
        setOnClickListener(null)
    }
}
```

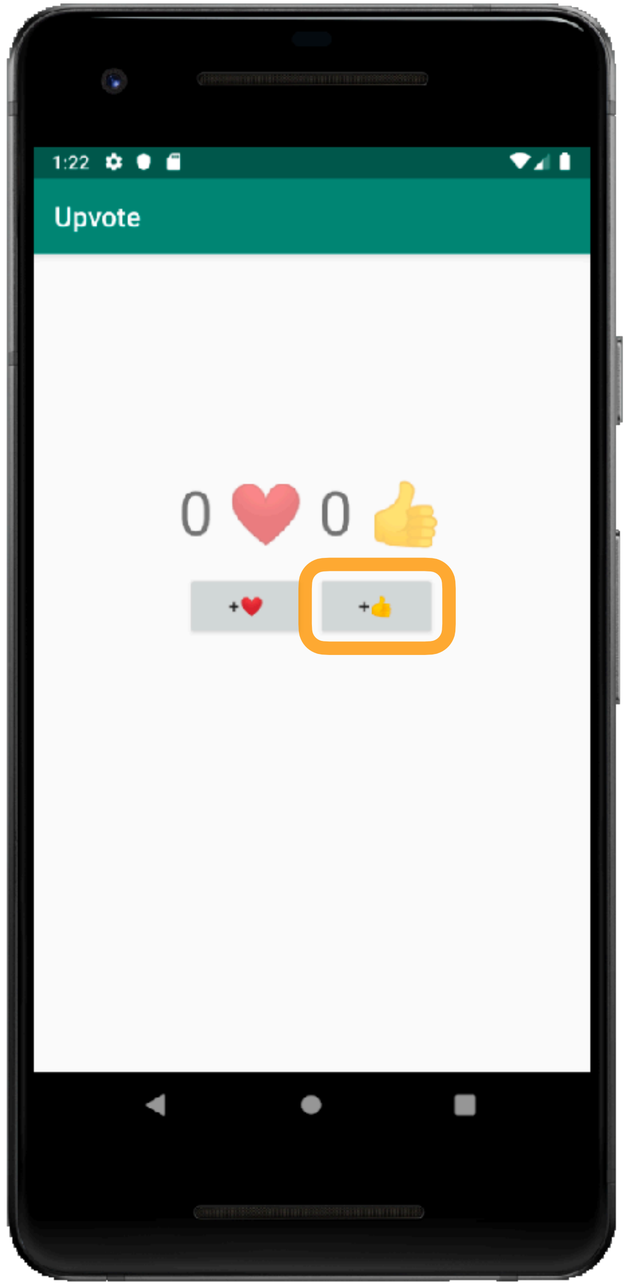


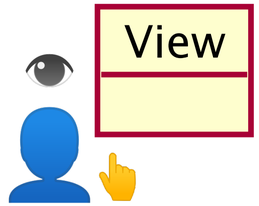

1:22
Upvote
0
0
+
+

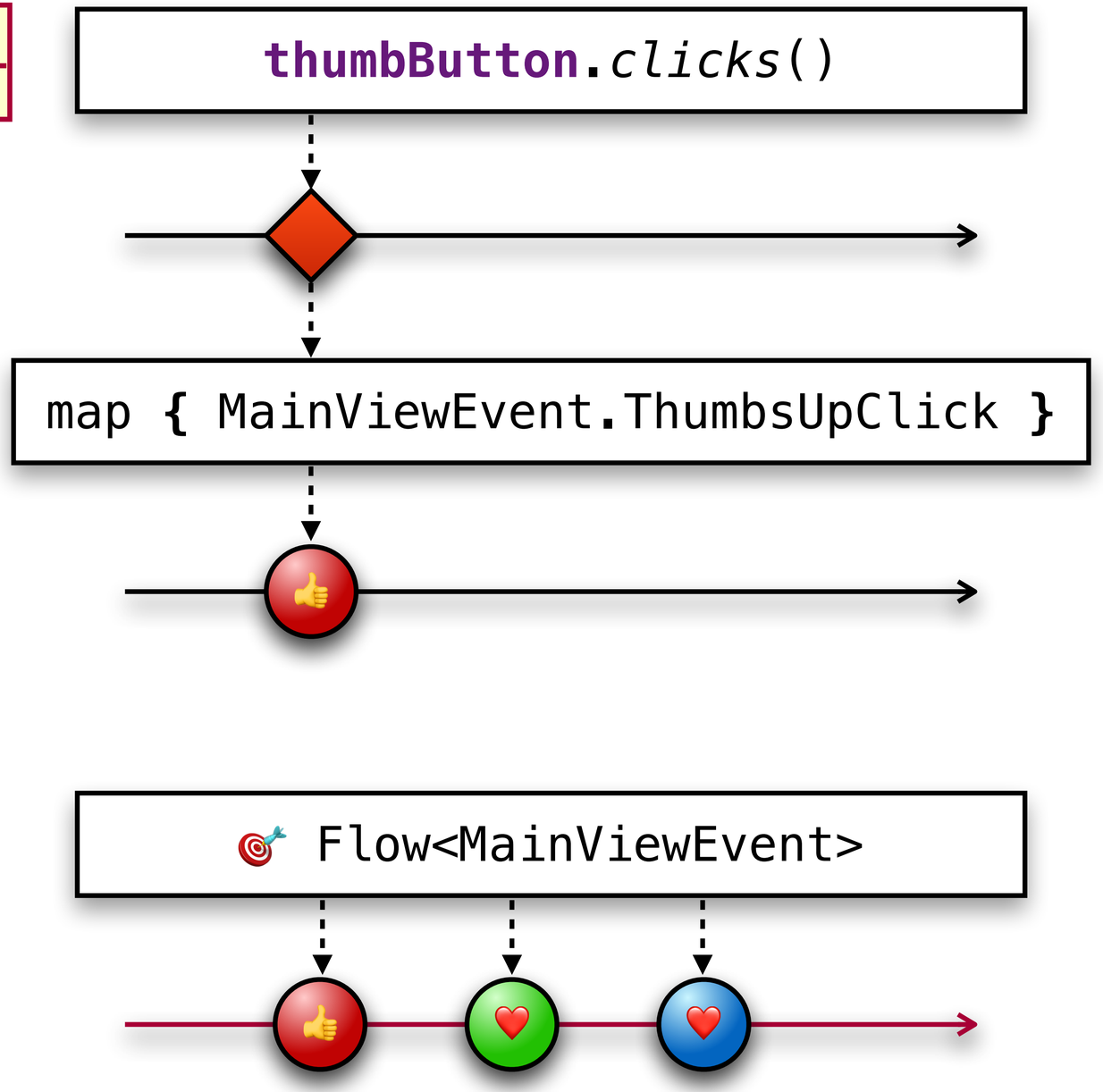

View

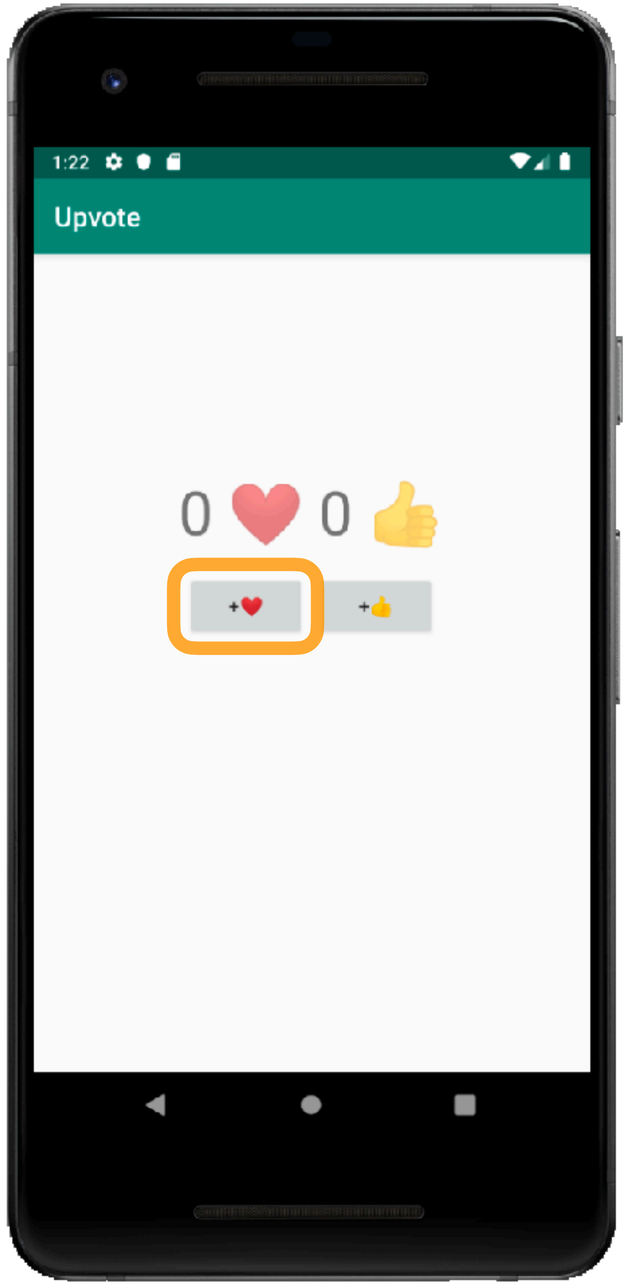

View.clicks():Flow<Unit>

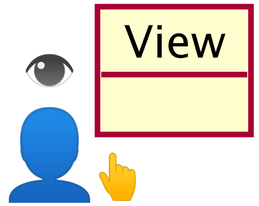

Flow<MainViewEvent>

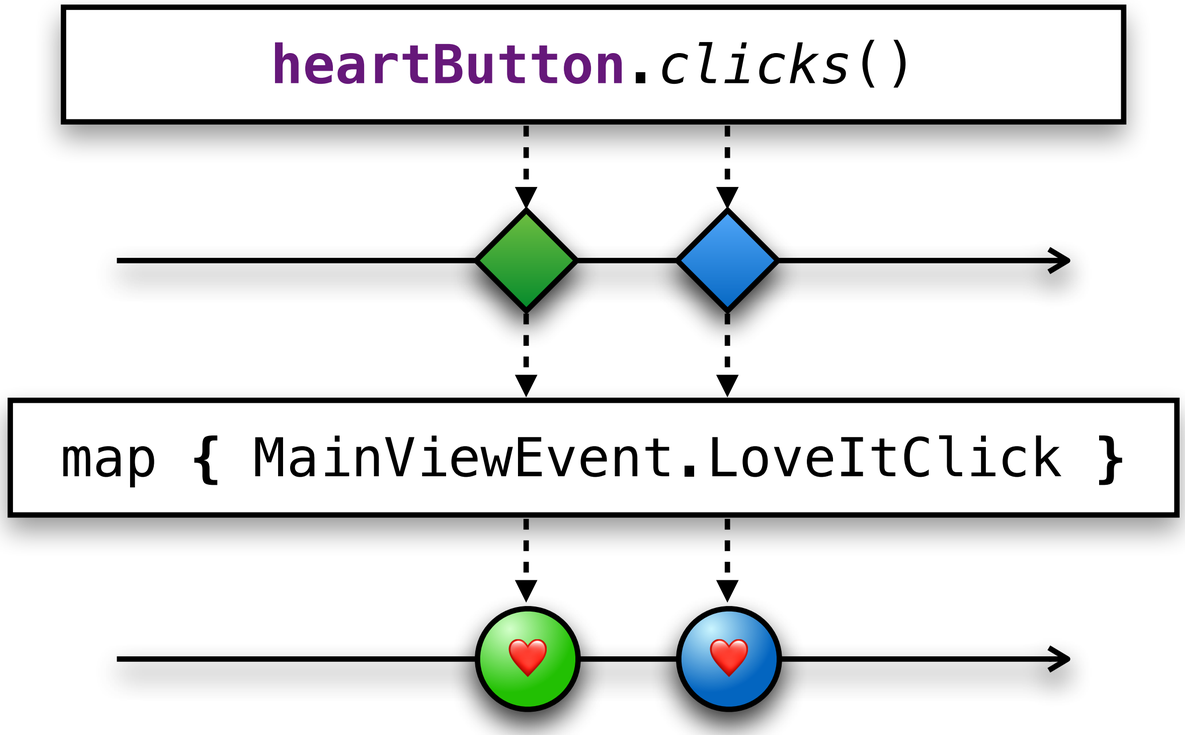

1:22
Upvote
0
0
+
+

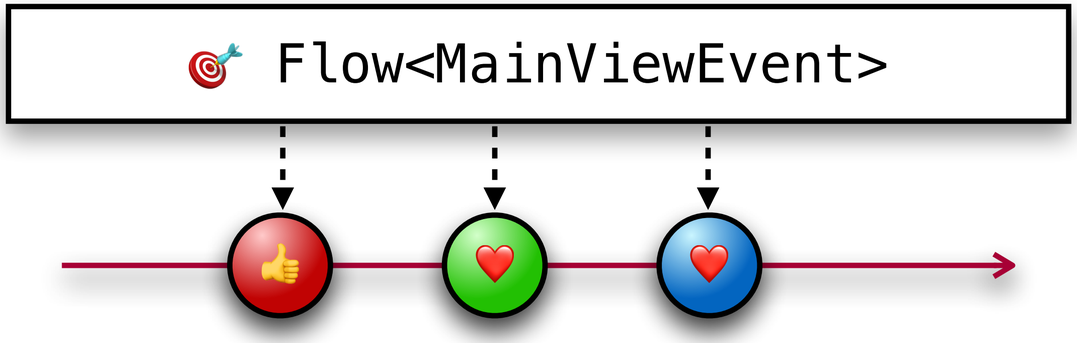

View

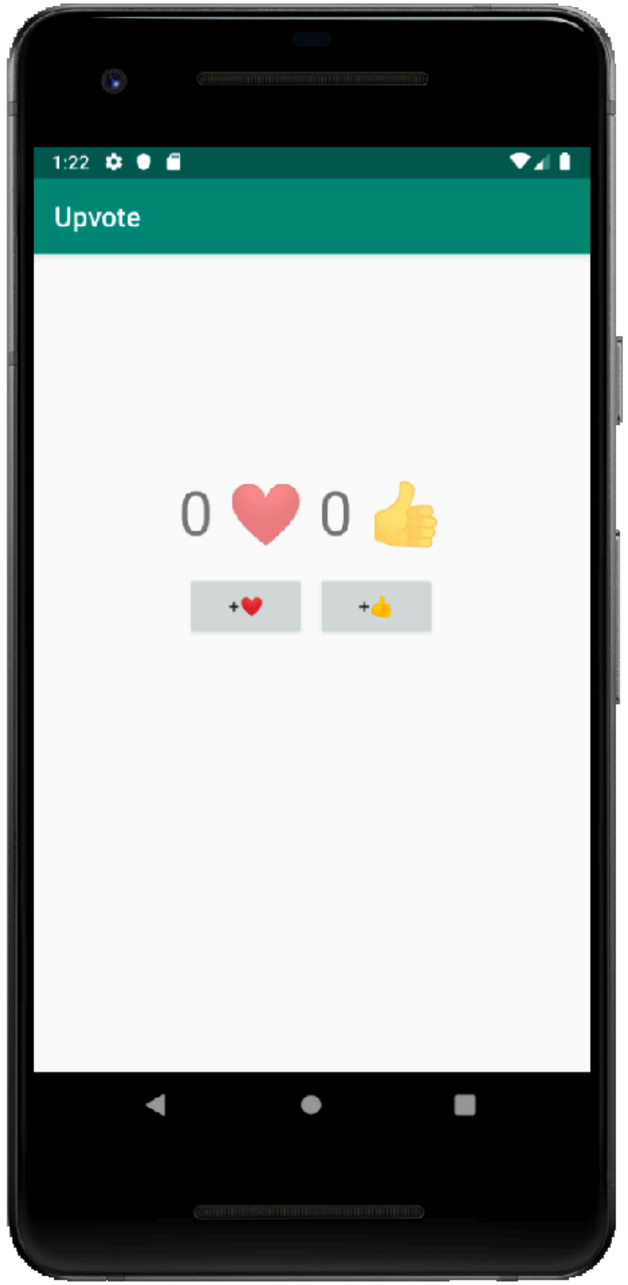

thumbButton.clicks()
map { MainViewEvent.ThumbsUpClick }
Flow<MainViewEvent>


1:22
Upvote
0
0
+
+

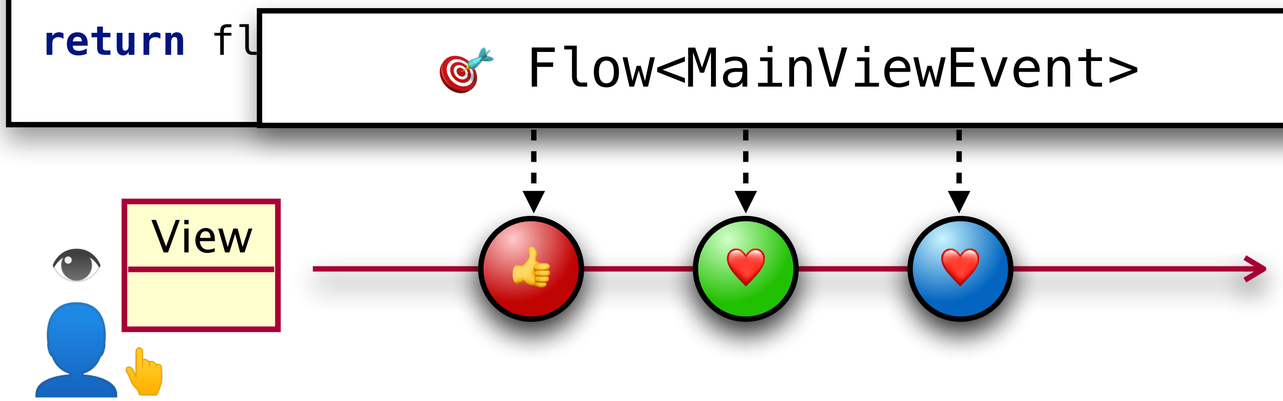

View

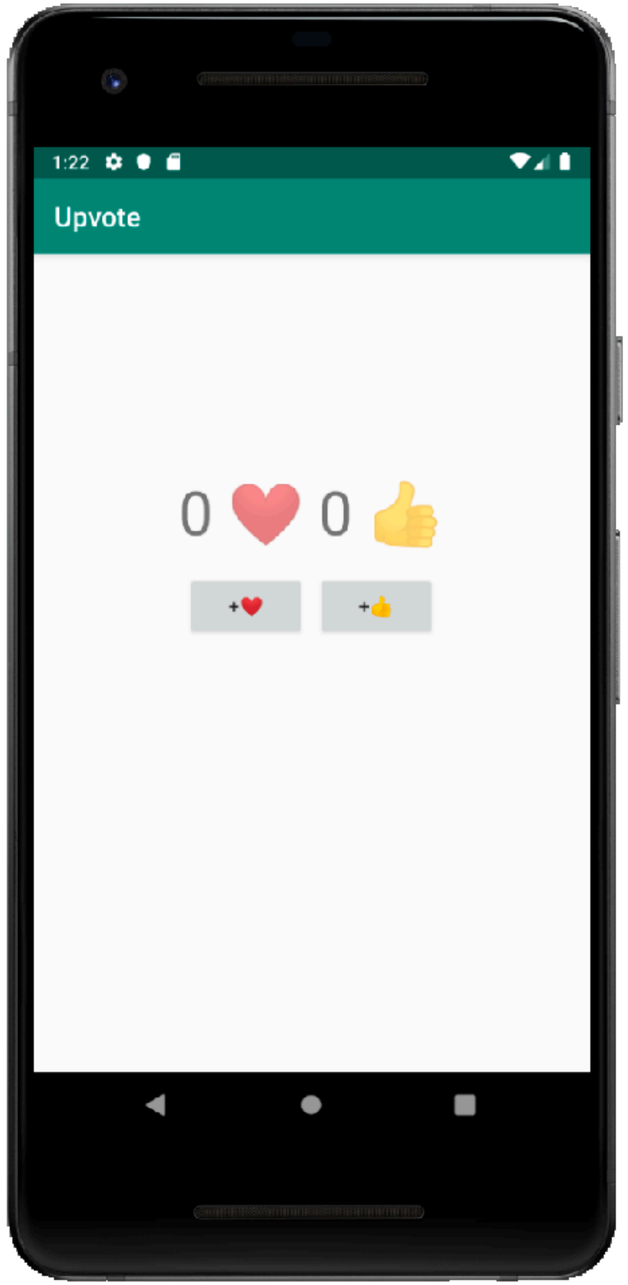

heartButton.clicks()
map { MainViewEvent.LoveItClick }


Flow<MainViewEvent>

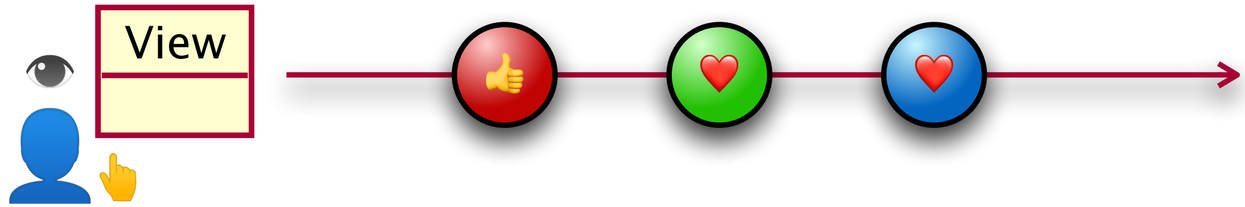


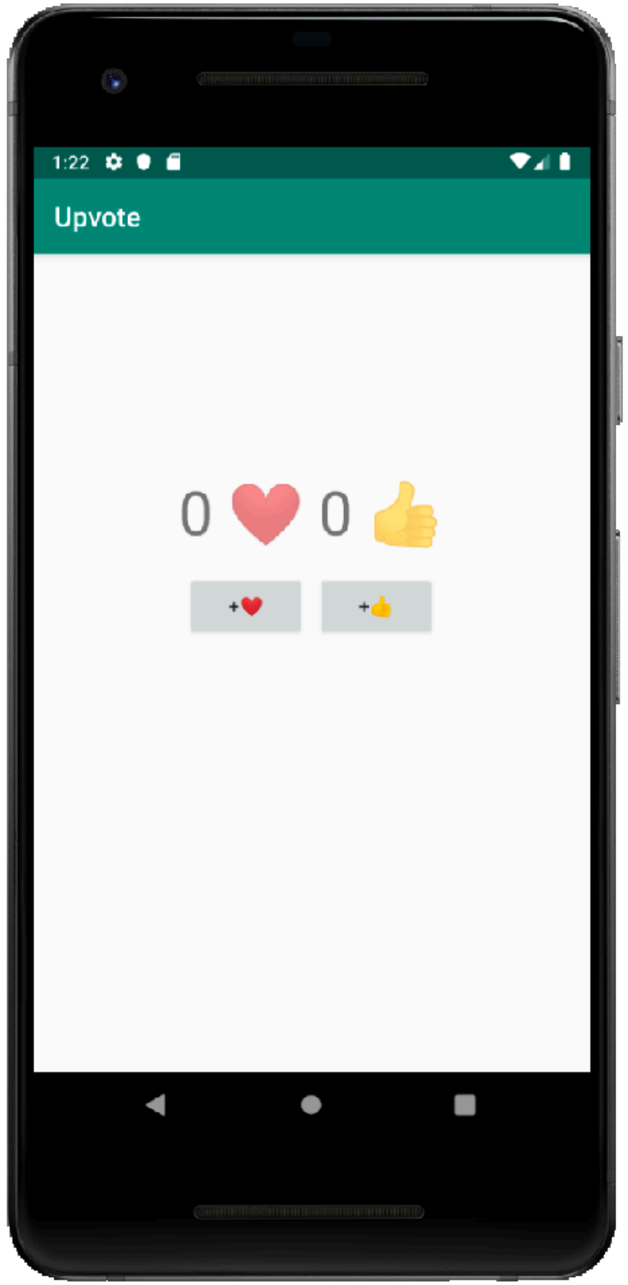

```
val flows = listOf(
    heartButton.clicks().map { MainViewEvent.LoveItClick },
    thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }
)

return fl
```



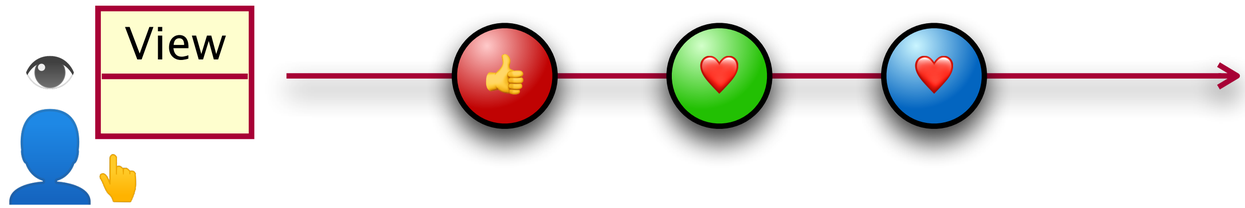


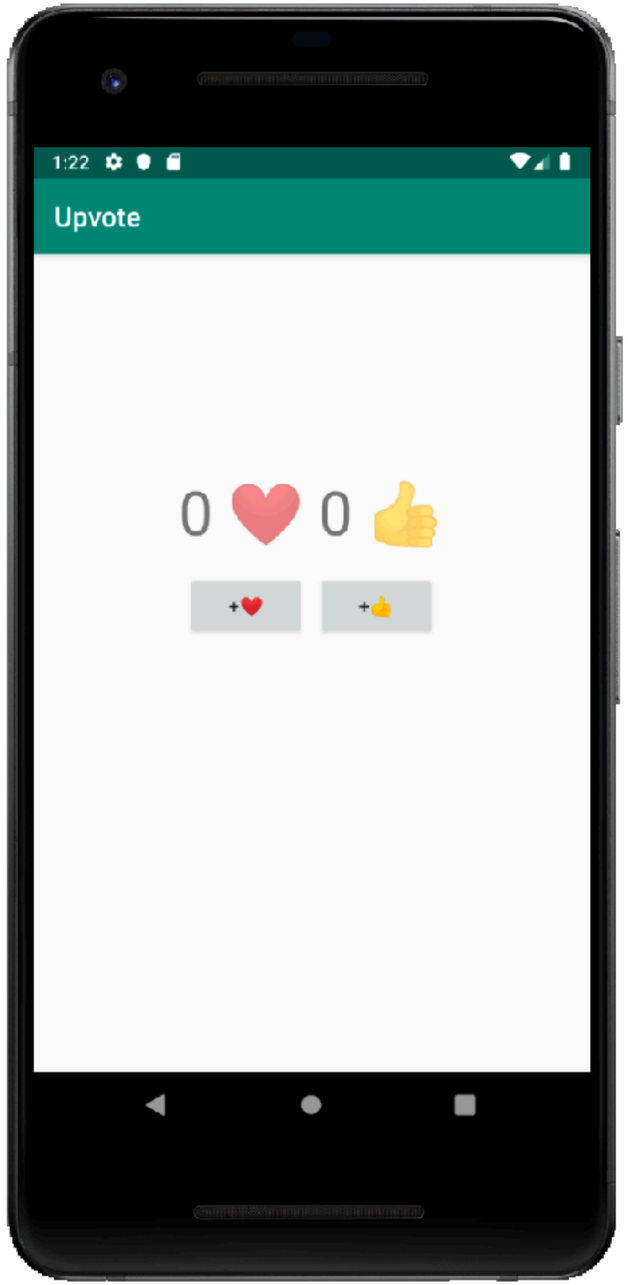

```
class MainActivity {

    override fun viewEvents(): Flow<MainViewEvent> {
        val flows = listOf(
            heartButton.clicks().map { MainViewEvent.LoveItClick },
            thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }
        )

        return flows.asFlow().flattenMerge(flows.size)
    }

}
```



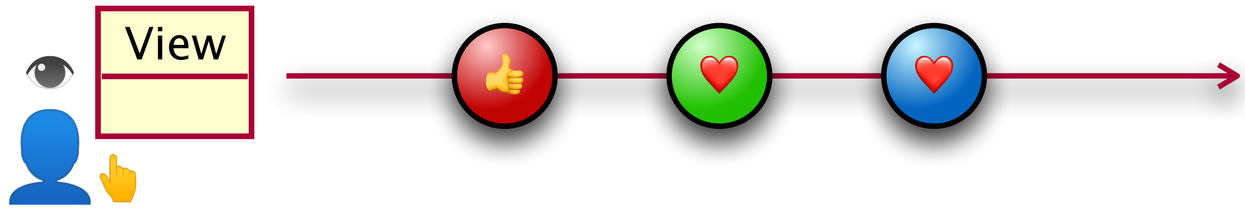


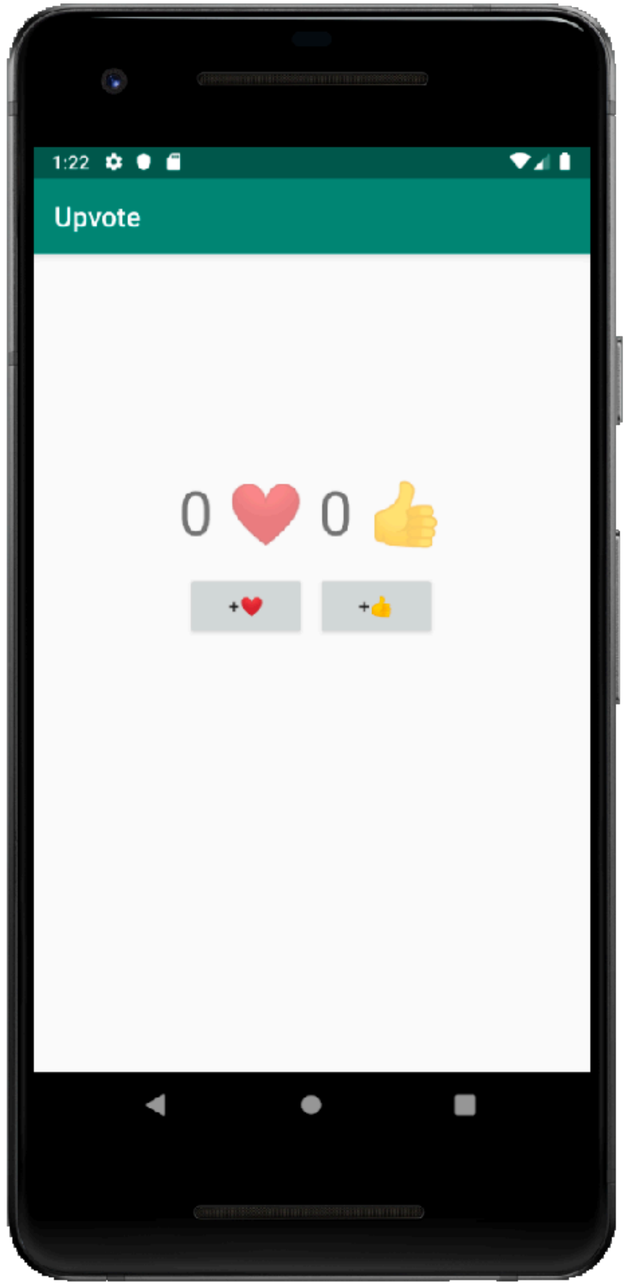

```
class MainActivity {

    override fun viewEvents(): Flow<MainViewEvent> {
        val flows = listOf(
            heartButton.clicks().map { MainViewEvent.LoveItClick },
            thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }
        )

        return flows.asFlow().flattenMerge(flows.size)
    }

}
```



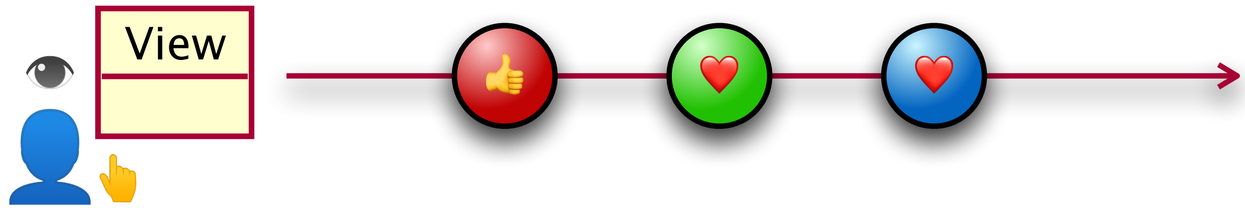


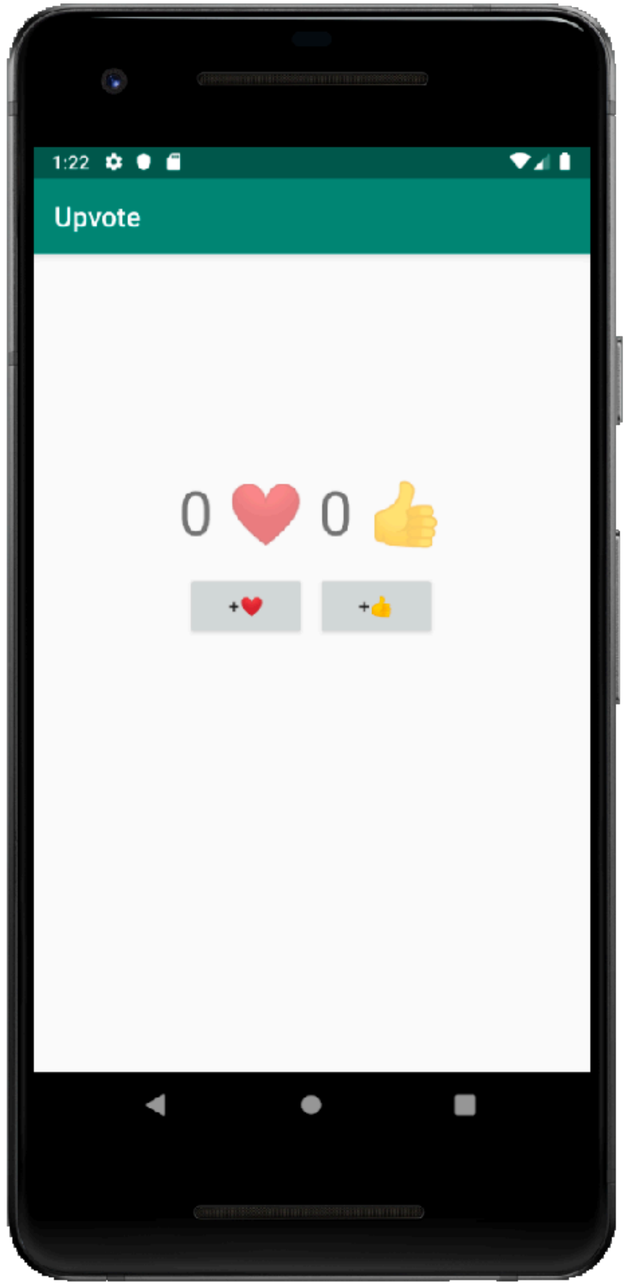

```
class MainActivity {

    override fun viewEvents(): Flow<MainViewEvent> {
        val flows = listOf(
            heartButton.clicks().map { MainViewEvent.LoveItClick },
            thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }
        )

        return flows.asFlow().flattenMerge(flows.size)
    }

}
```



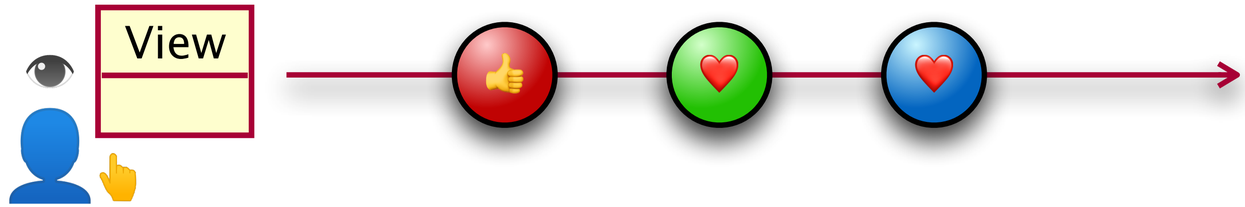


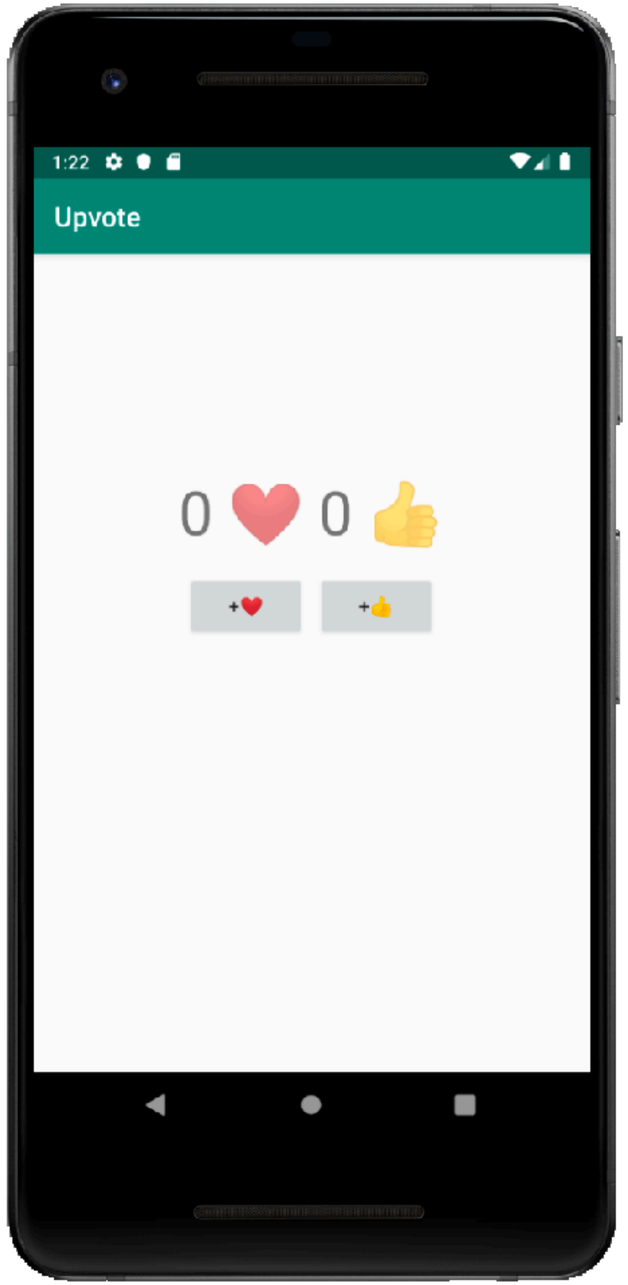

```
class MainActivity {

    override fun viewEvents(): Flow<MainViewEvent> {
        val flows = listOf(
            heartButton.clicks().map { MainViewEvent.LoveItClick },
            thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }
        )

        return flows.asFlow().flattenMerge(flows.size)
    }

}
```



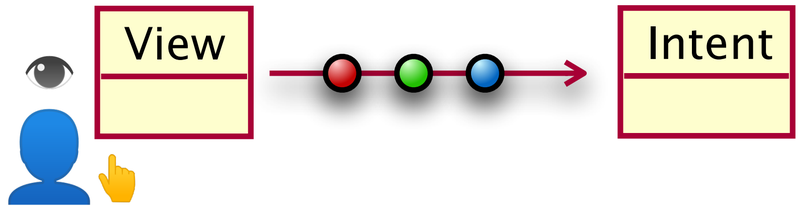


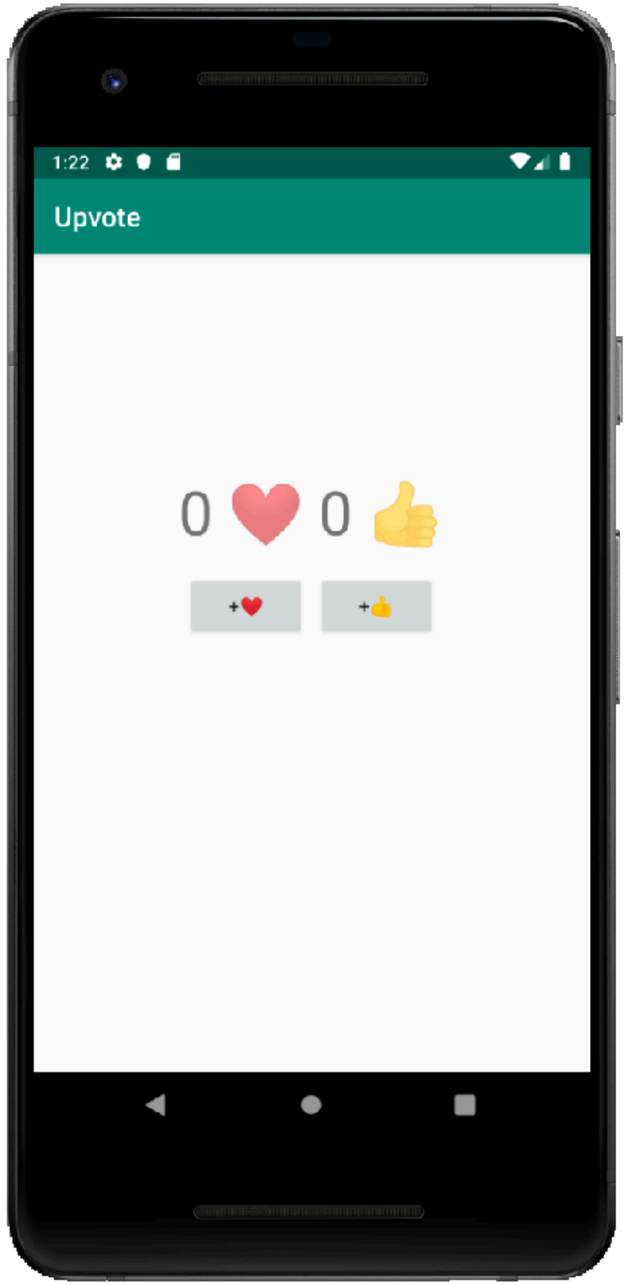

```kotlin
/**
 * This allows us to group all the viewEvents from
 * one view in a single Flow.
 */
interface ViewEventFlow<E> {
    fun viewEvents(): Flow<E>
}

class MainActivity : ViewEventFlow<MainViewEvent> {

    override fun viewEvents(): Flow<MainViewEvent> {
        val flows = listOf(
            heartButton.clicks().map { MainViewEvent.LoveItClick },
            thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }
        )

        return flows.asFlow().flattenMerge(flows.size)
    }

}
```



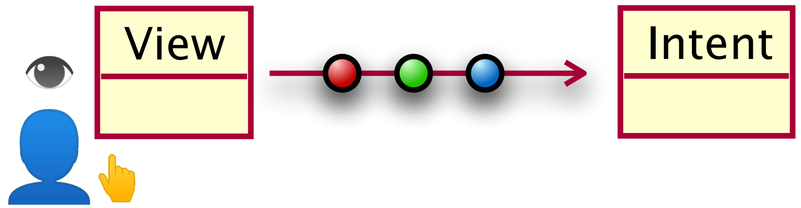


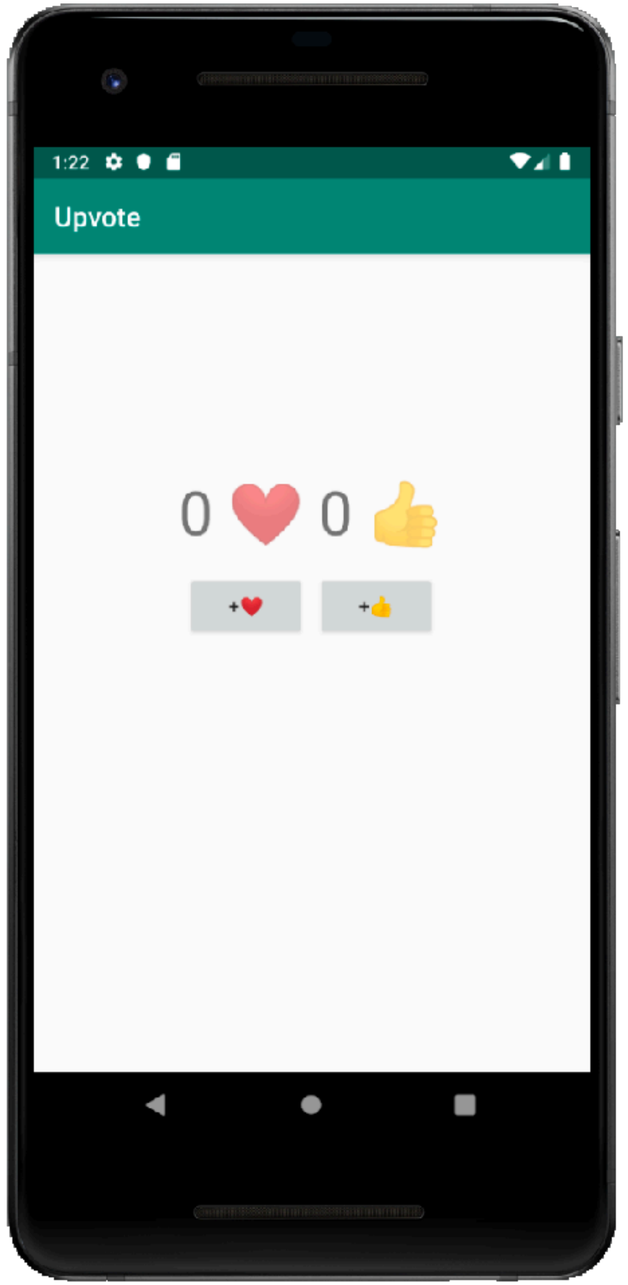

```
class MainActivity : ViewEventFlow<MainViewEvent> {

    val _scope: CoroutineScope = MainScope()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // ...

    }

    override fun onDestroy() {
        super.onDestroy()
        scope.cancel()
    }

    override fun viewEvents(): Flow<MainViewEvent> {
        // ...
    }

}
```



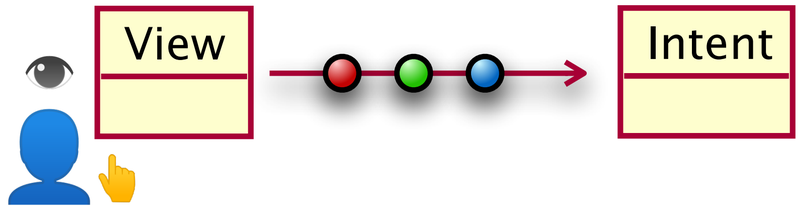


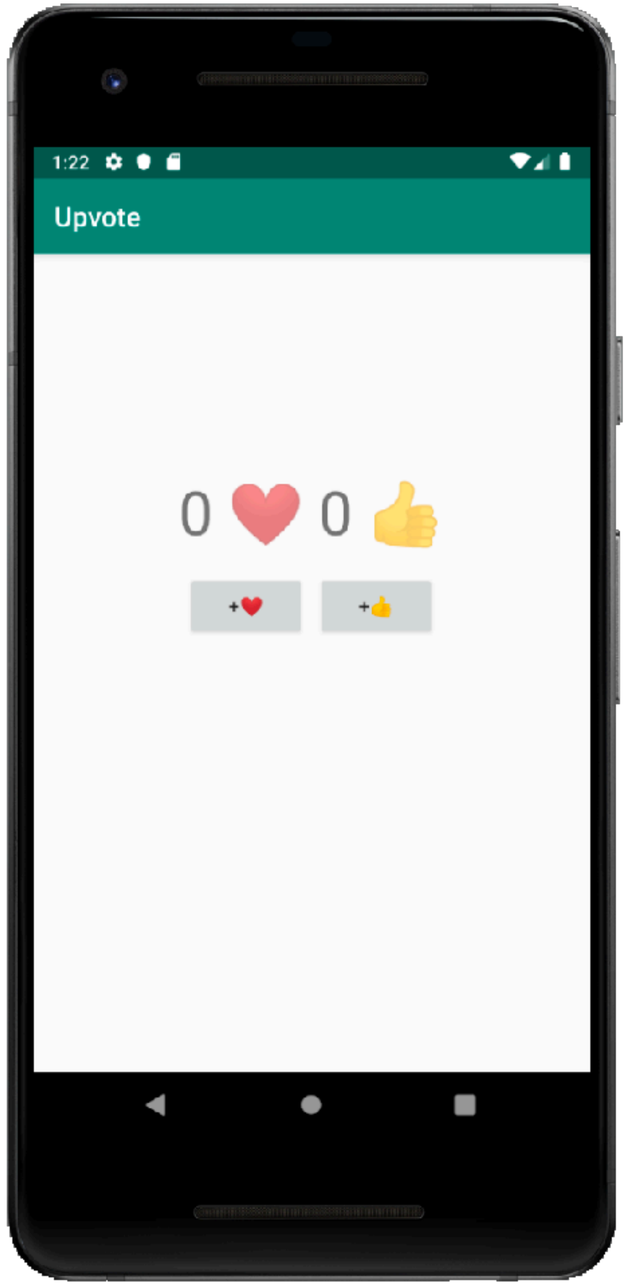

```
class MainActivity : ViewEventFlow<MainViewEvent> {

    val scope: CoroutineScope = MainScope()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // ...

    }

    override fun onDestroy() {
        super.onDestroy()
        scope.cancel()
    }

    override fun viewEvents(): Flow<MainViewEvent> {
        // ...
    }

}
```



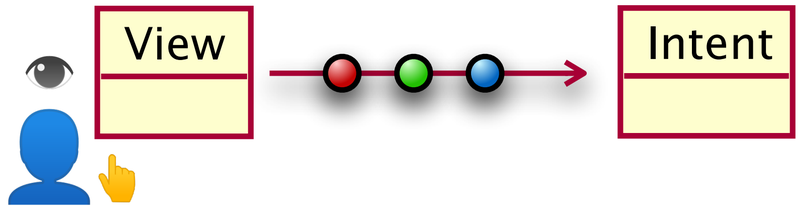


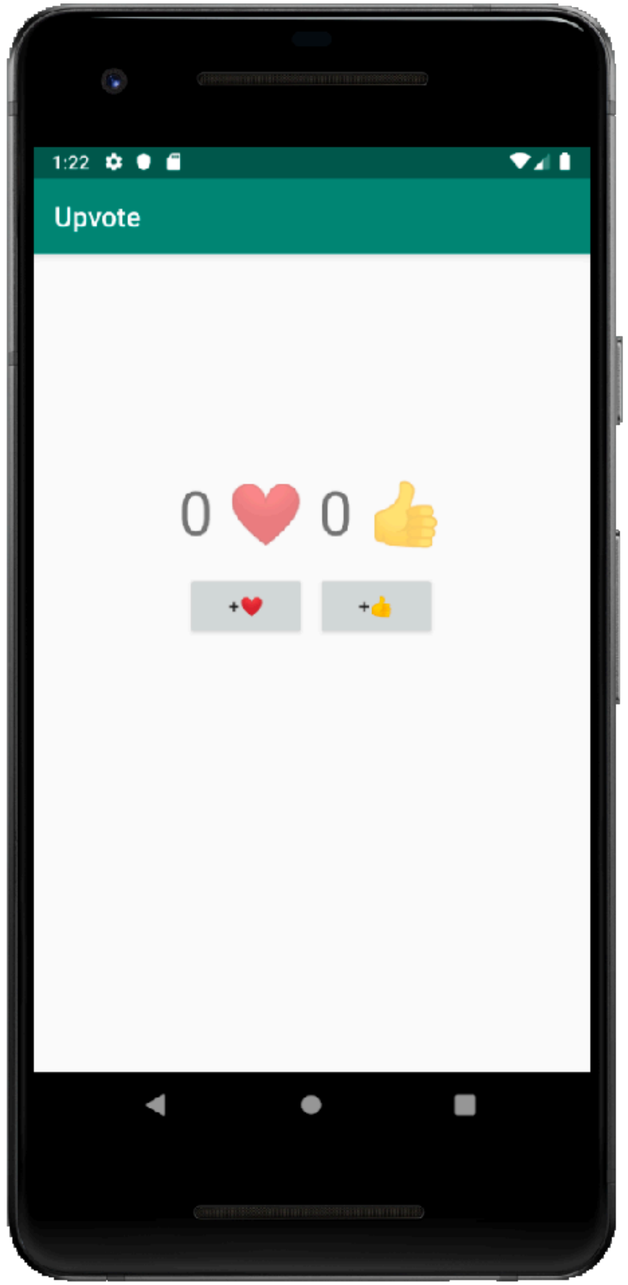

```
class MainActivity : ViewEventFlow<MainViewEvent> {

    val scope: CoroutineScope = MainScope()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // ...
        scope.launch { this: CoroutineScope
            viewEvents()
                .onEach { ? }
                .collect()
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        scope.cancel()
    }

    override fun viewEvents(): Flow<MainViewEvent> {
        // ...
    }

}
```



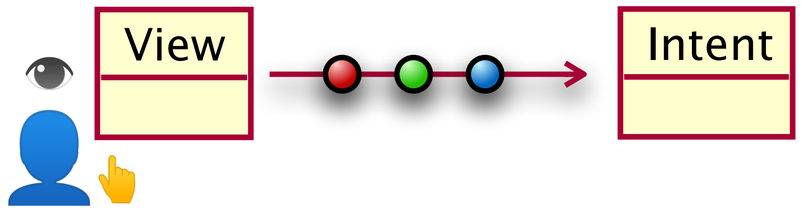


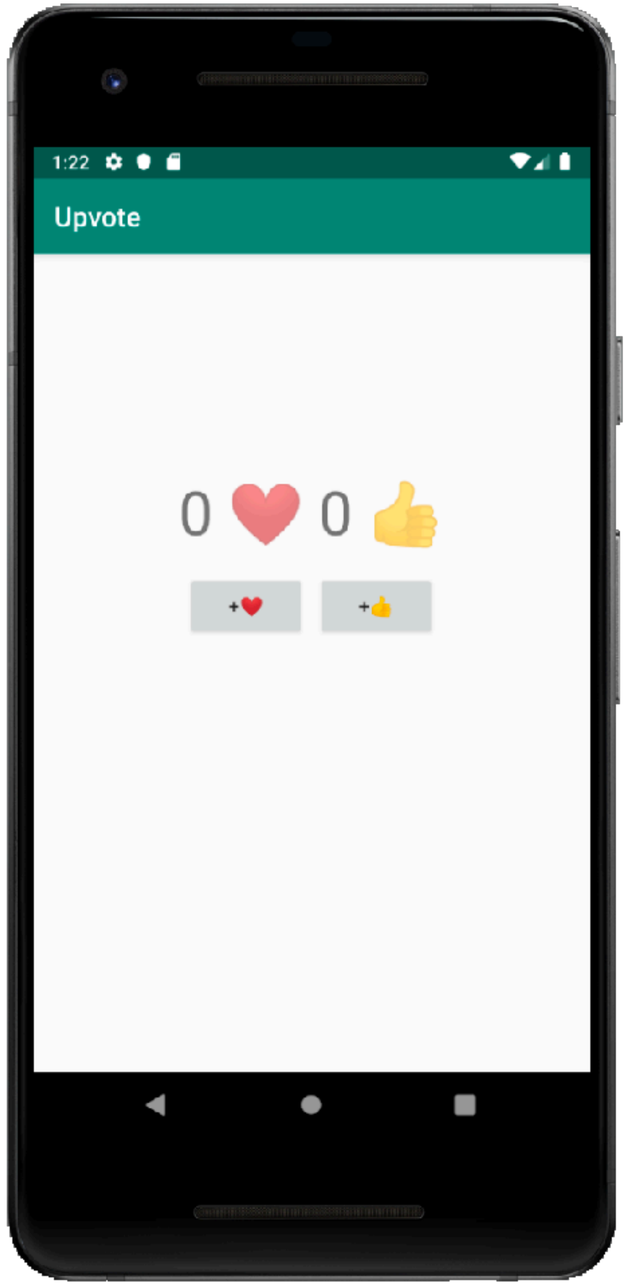

```
class MainActivity : ViewEventFlow<MainViewEvent> {

    val scope: CoroutineScope = MainScope()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // ...
        scope.launch { this: CoroutineScope
            viewEvents()
                .onEach { ? }
                .collect()
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        scope.cancel()
    }

    override fun viewEvents(): Flow<MainViewEvent> {
        // ...
    }

}
```



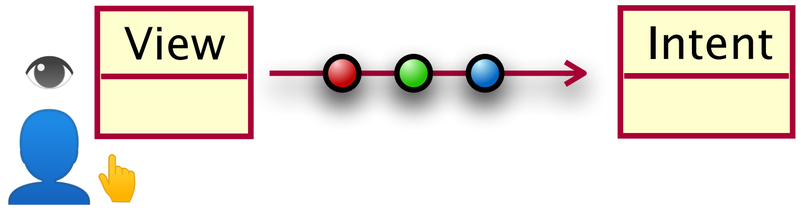


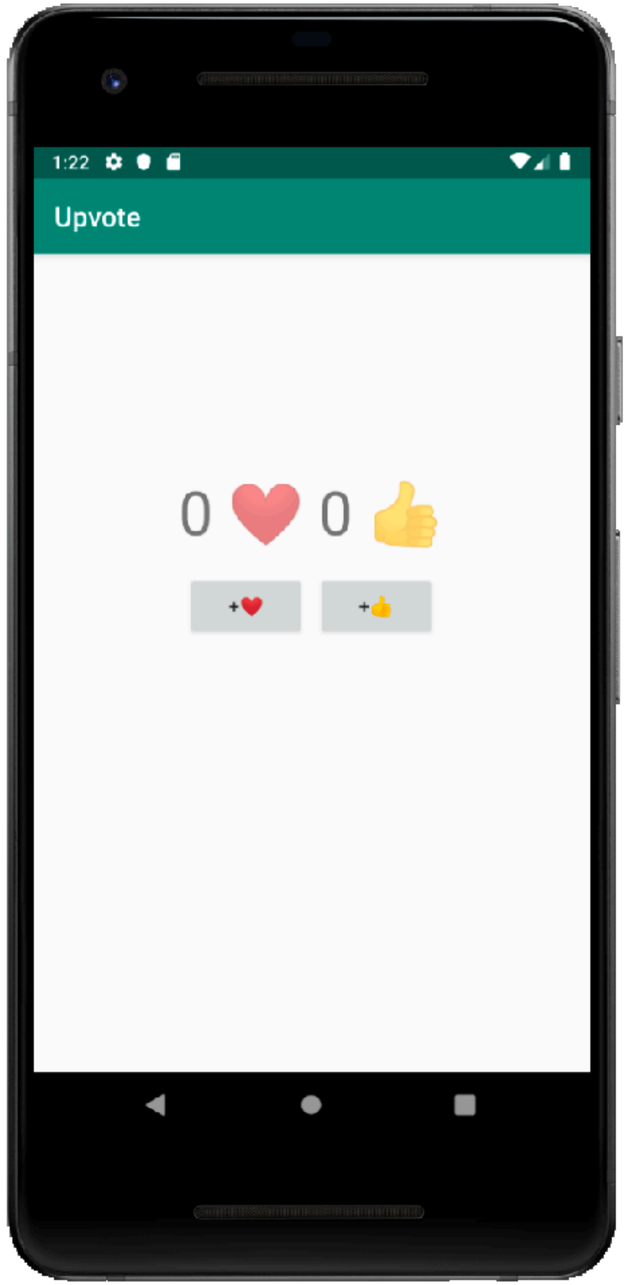

```
class MainActivity : ViewEventFlow<MainViewEvent> {

    val scope: CoroutineScope = MainScope()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // ...
        scope.launch { this: CoroutineScope
            viewEvents()
                .onEach { ? }
                .collect()
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        scope.cancel()
    }

    override fun viewEvents(): Flow<MainViewEvent> {
        // ...
    }

}
```



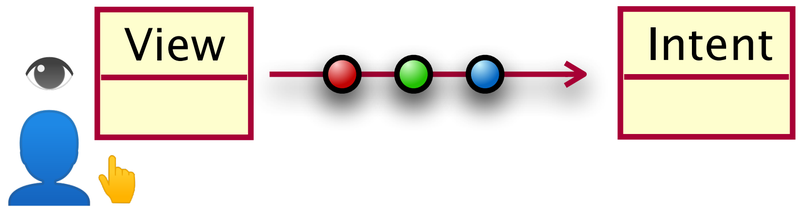


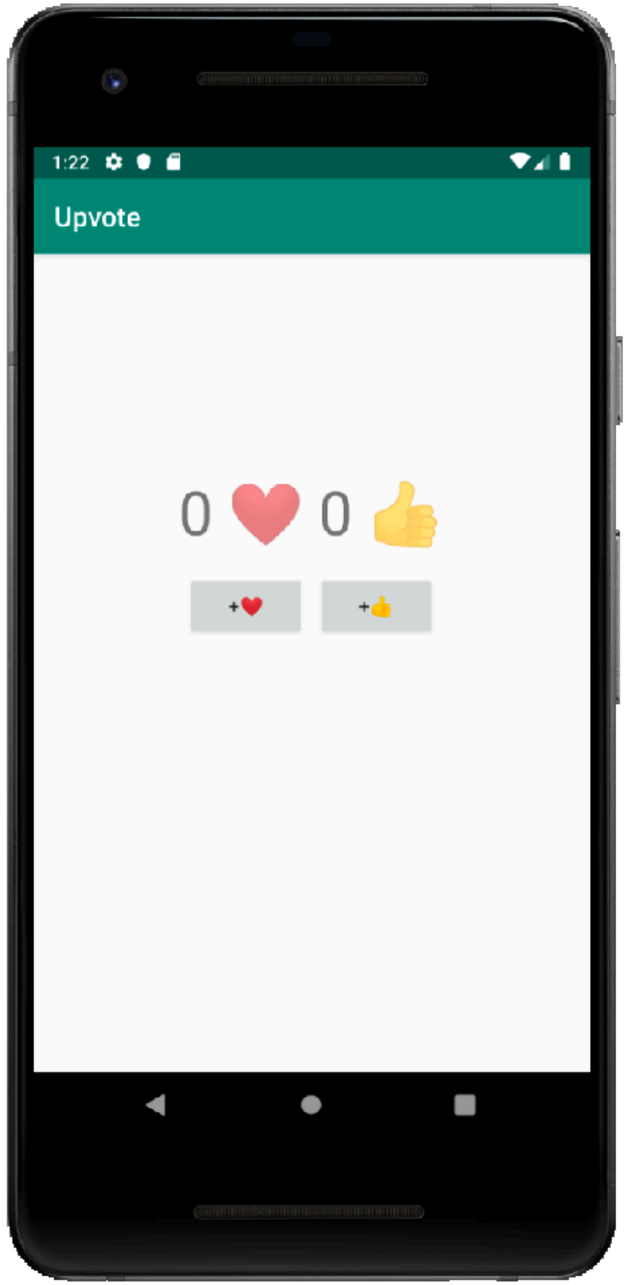

```
class MainActivity : ViewEventFlow<MainViewEvent> {

    val scope: CoroutineScope = MainScope()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // ...
        scope.launch { this: CoroutineScope
            viewEvents()
                .onEach { ? }
                .collect()
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        scope.cancel()
    }

    override fun viewEvents(): Flow<MainViewEvent> {
        // ...
    }

}
```



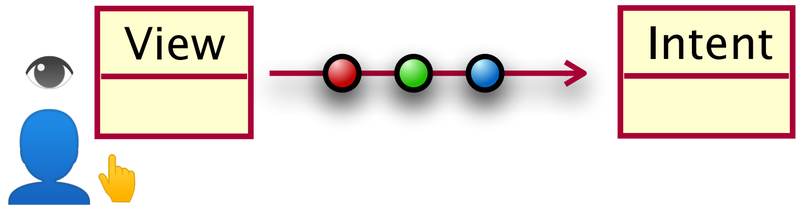


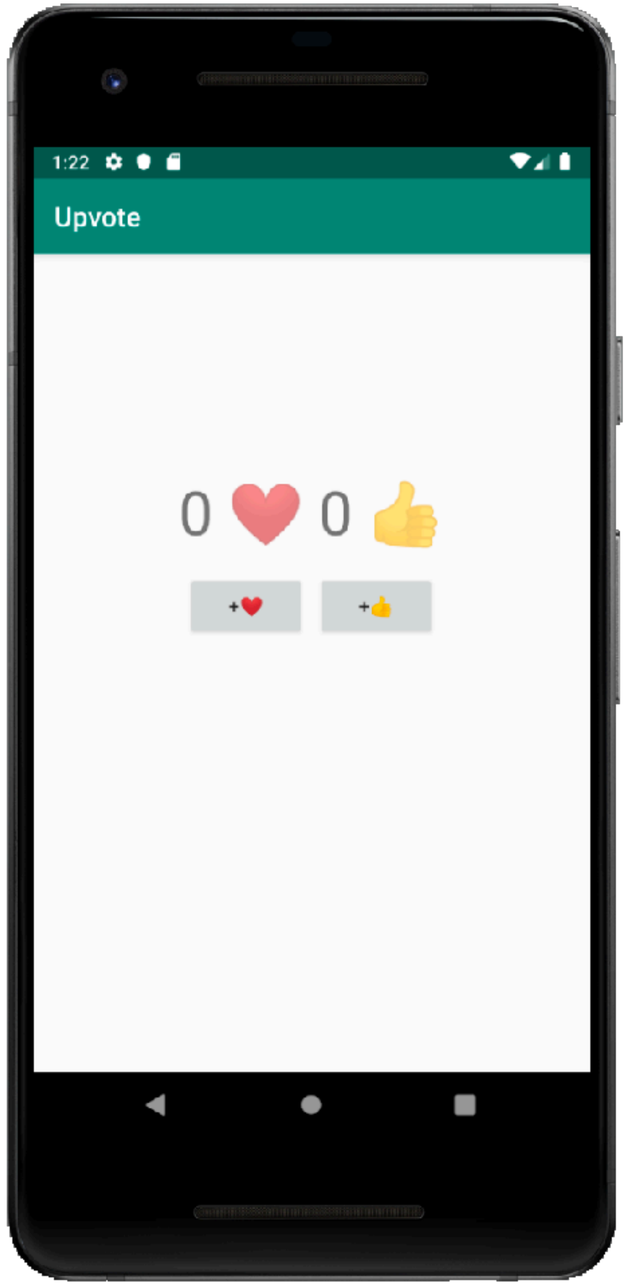

```
class MainActivity : ViewEventFlow<MainViewEvent> {

    val scope: CoroutineScope = MainScope()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // ...
        scope.launch { this: CoroutineScope
            viewEvents()
                .onEach { ? }
                .collect()
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        scope.cancel()
    }

    override fun viewEvents(): Flow<MainViewEvent> {
        // ...
    }

}
```



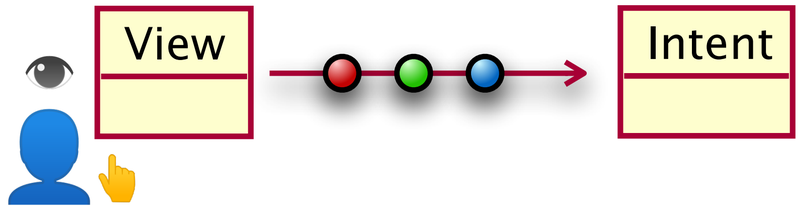


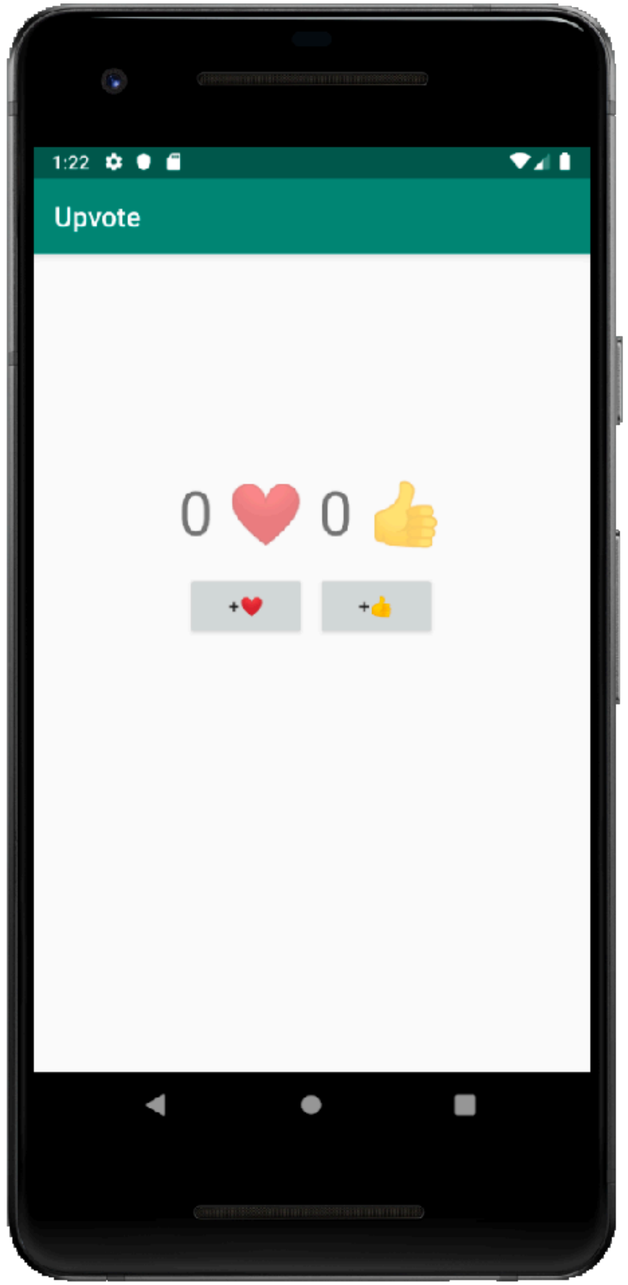

```
class MainActivity : ViewEventFlow<MainViewEvent> {

    val scope: CoroutineScope = MainScope()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // ...
        viewEvents()
            .onEach { ? }
            .launchIn(scope)
    }

    override fun onDestroy() {
        super.onDestroy()
        scope.cancel()
    }

    override fun viewEvents(): Flow<MainViewEvent> {
        // ...
    }

}
```



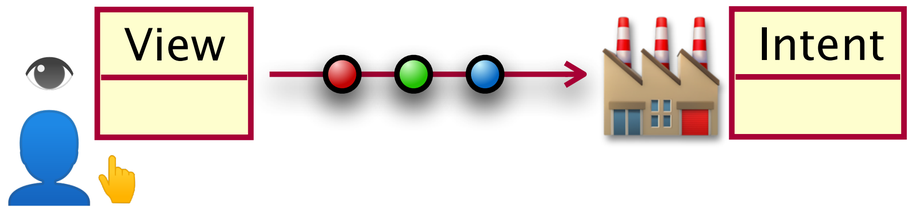


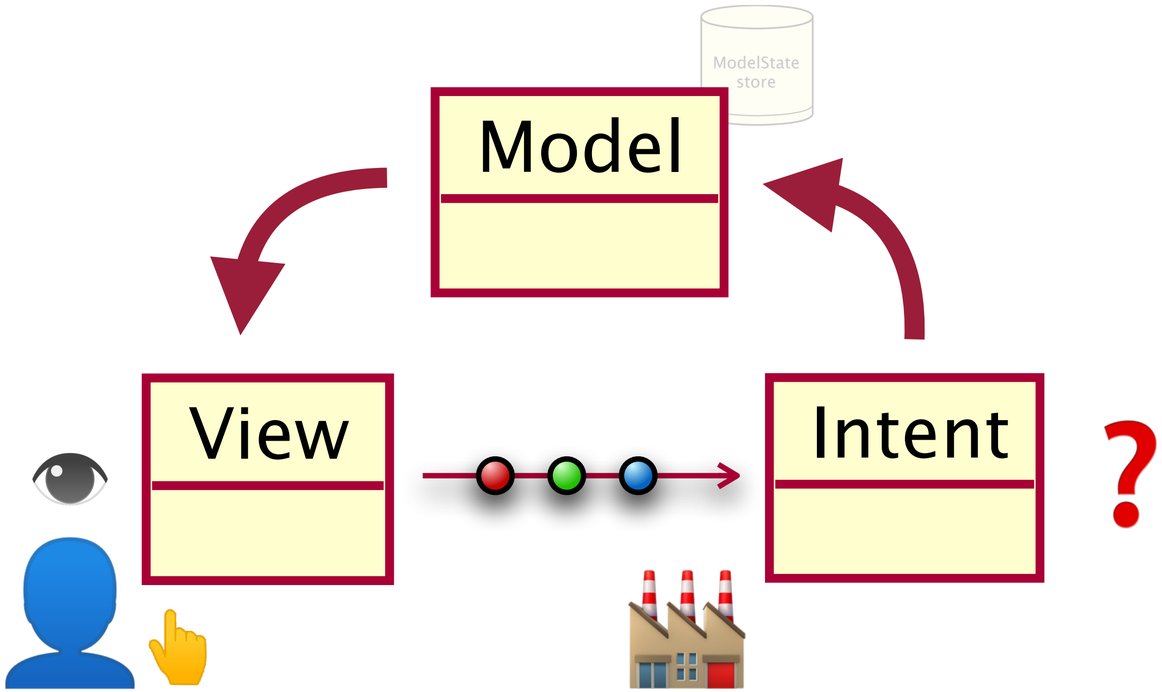

```
class MainActivity : ViewEventFlow<MainViewEvent> {

    val scope: CoroutineScope = MainScope()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // ...
        viewEvents()
            .onEach { event -> ? }
            .launchIn(scope)
    }

    override fun onDestroy() {
        super.onDestroy()
        scope.cancel()
    }

    override fun viewEvents(): Flow<MainViewEvent> {
        // ...
    }

}
```

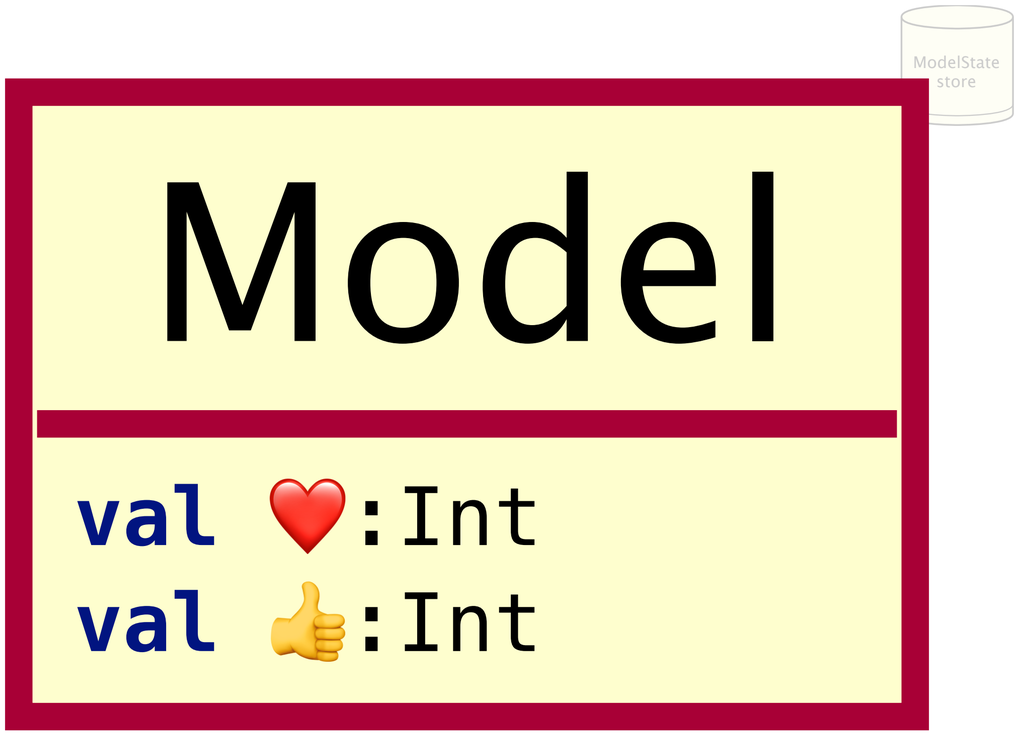


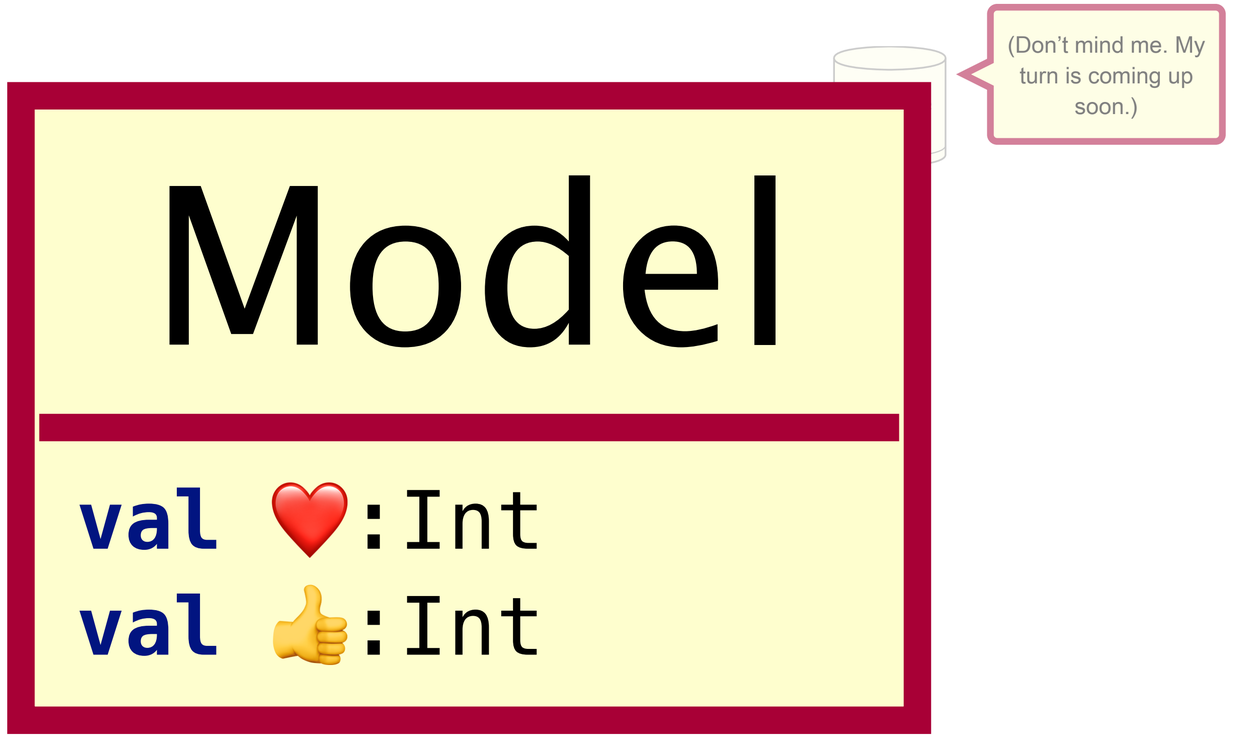



```
class MainActivity : ViewEventFlow<MainViewEvent> {

    val scope: CoroutineScope = MainScope()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // ...
        viewEvents()
            .onEach { event ->
                MainViewIntentFactory.process(event)
            }
            .launchIn(scope)
    }

    override fun onDestroy() {
        super.onDestroy()
        scope.cancel()
    }

    override fun viewEvents(): Flow<MainViewEvent> {
        // ...
    }

}
```



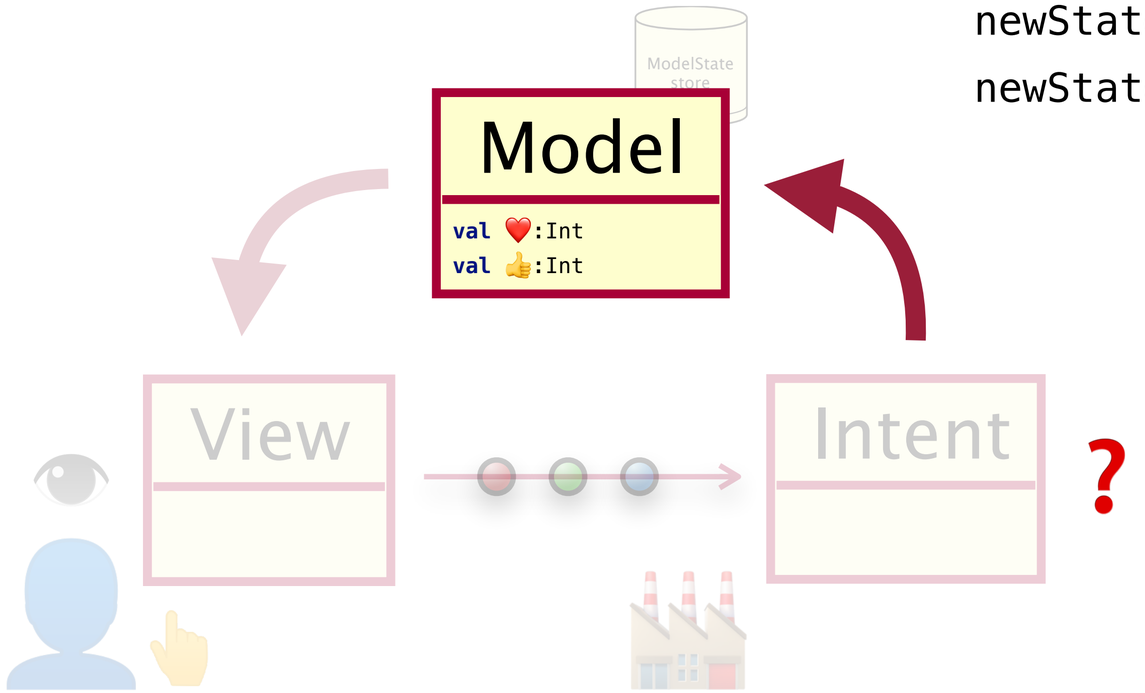

ModelState
store
Model
View
Intent
?


ModelState
store
Model
val ❤️:Int
val 👍:Int

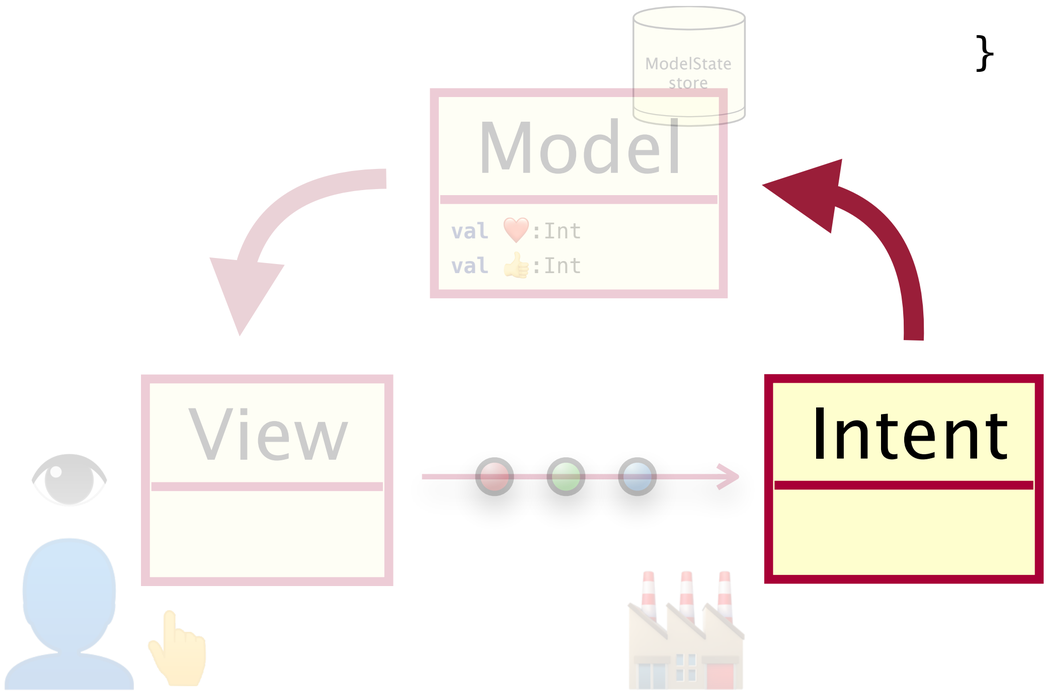


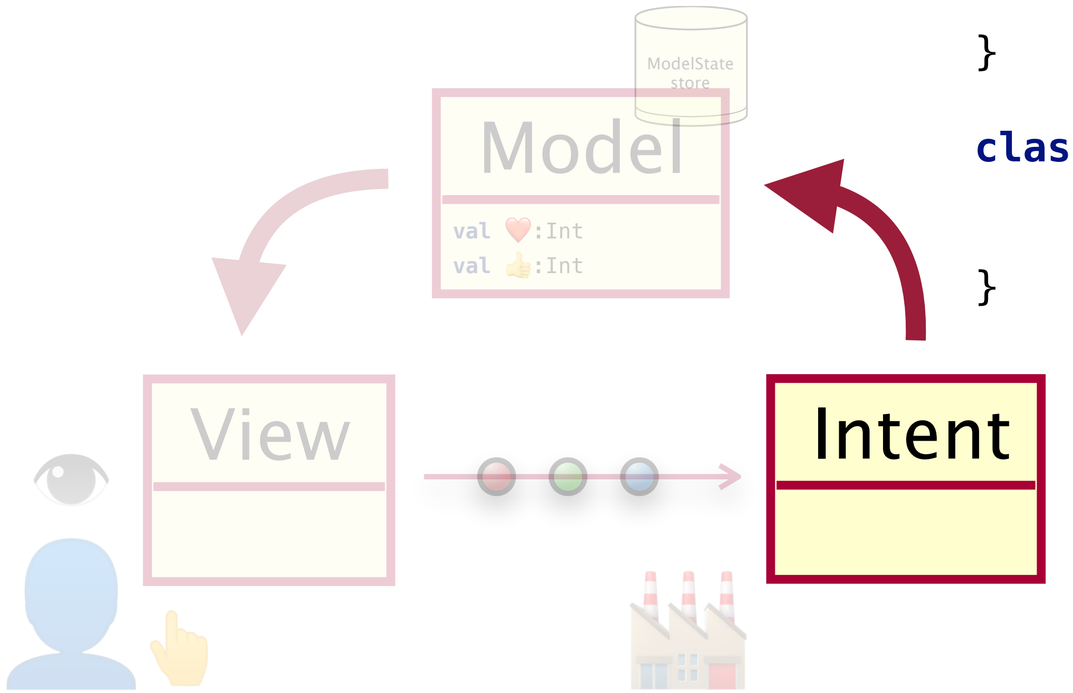

```
data class UpvoteModel(val hearts:Int, val thumbs:Int)
```

```
data class UpvoteModel(val hearts:Int, val thumbs:Int)

newState = oldState.copy(thumbs = thumbs + 1) // 0, 1
newState = newState.copy(hearts = hearts + 1) // 1, 1
newState = newState.copy(hearts = hearts + 1) // 2, 1
```

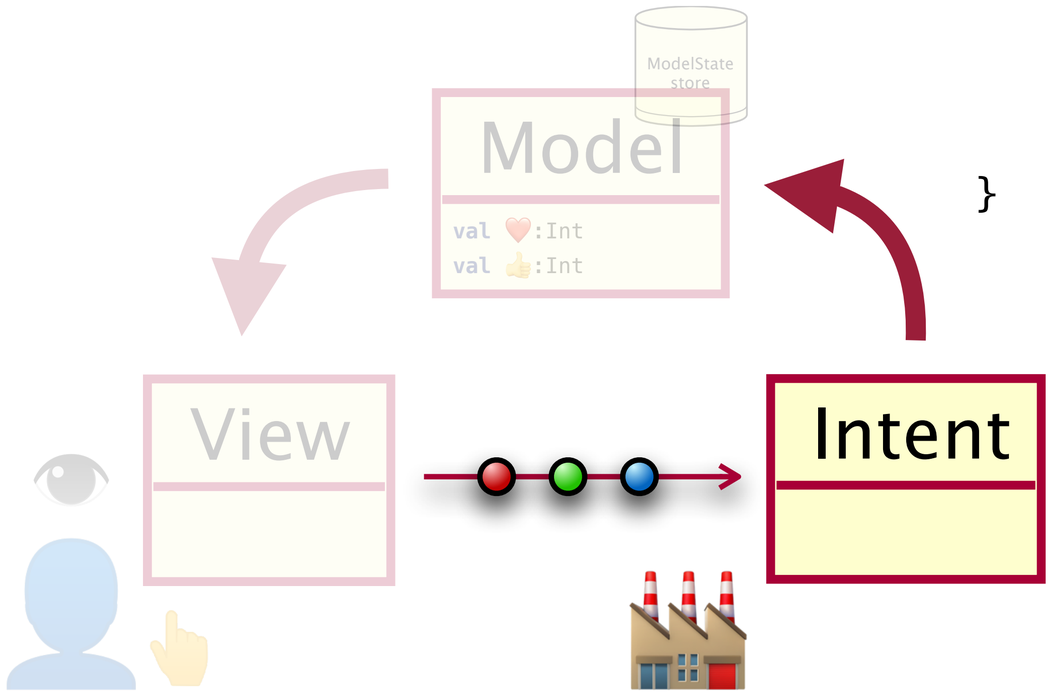


```kotlin
data class UpvoteModel(val hearts:Int, val thumbs:Int)

interface Intent<T> {
    fun reduce(oldState: T): T
}
```



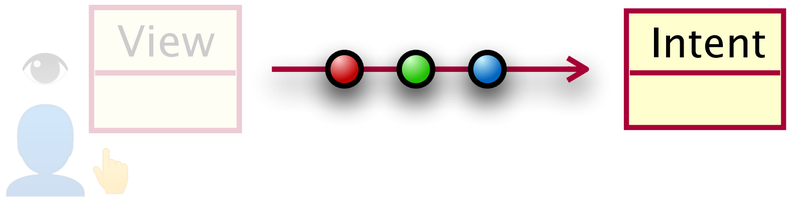



```
data class UpvoteModel(val hearts:Int, val thumbs:Int)

interface Intent<T> {
    fun reduce(oldState: T): T
}

class AddHeart():Intent<UpvoteModel> {
    override fun reduce(oldState: UpvoteModel) =
        oldState.copy(hearts = oldState.hearts + 1)
}
```

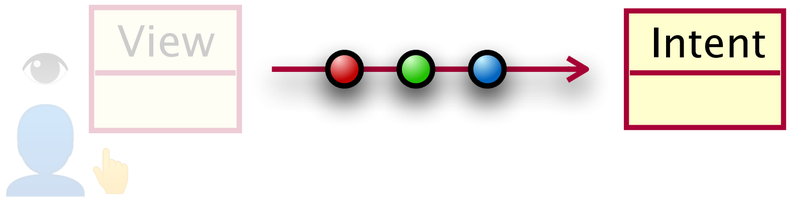



```
data class UpvoteModel(val hearts:Int, val thumbs:Int)

fun toIntent(viewEvent: MainViewEvent):Intent<UpvoteModel> {
    return when (viewEvent) {
        MainViewEvent.LoveItClick -> AddHeart()
        MainViewEvent.ThumbsUpClick -> AddThumb()
    }
}
```

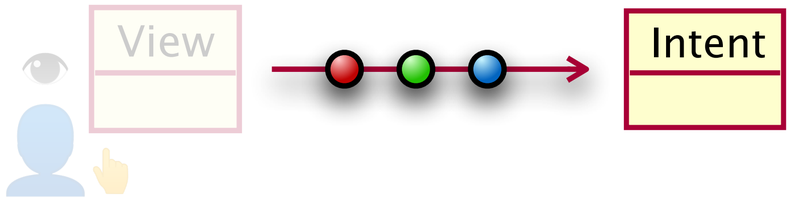
```
interface IntentFactory<E> {
    suspend fun process(viewEvent:E)
}

object MainViewIntentFactory : IntentFactory<MainViewEvent> {

    override suspend fun process(viewEvent: MainViewEvent) {
        UpvoteModelStore.process(toIntent(viewEvent))
    }

    private fun toIntent(viewEvent: MainViewEvent):Intent<UpvoteModel> {
        return when (viewEvent) {
            MainViewEvent.LoveItClick -> AddHeart()
            MainViewEvent.ThumbsUpClick -> AddThumb()
        }
    }

}
```



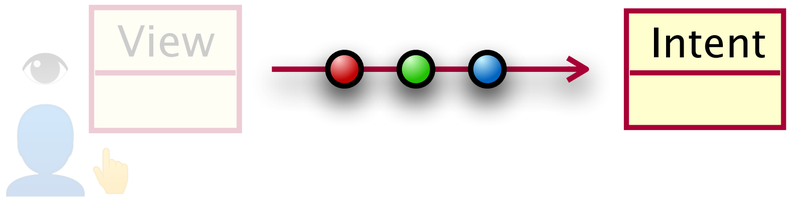
```
interface IntentFactory<E> {
    suspend fun process(viewEvent:E)
}

object MainViewIntentFactory : IntentFactory<MainViewEvent> {

    override suspend fun process(viewEvent: MainViewEvent) {
        UpvoteModelStore.process(toIntent(viewEvent))
    }

    private fun toIntent(viewEvent: MainViewEvent):Intent<UpvoteModel> {
        return when (viewEvent) {
            MainViewEvent.LoveItClick -> AddHeart()
            MainViewEvent.ThumbsUpClick -> AddThumb()
        }
    }

}
```

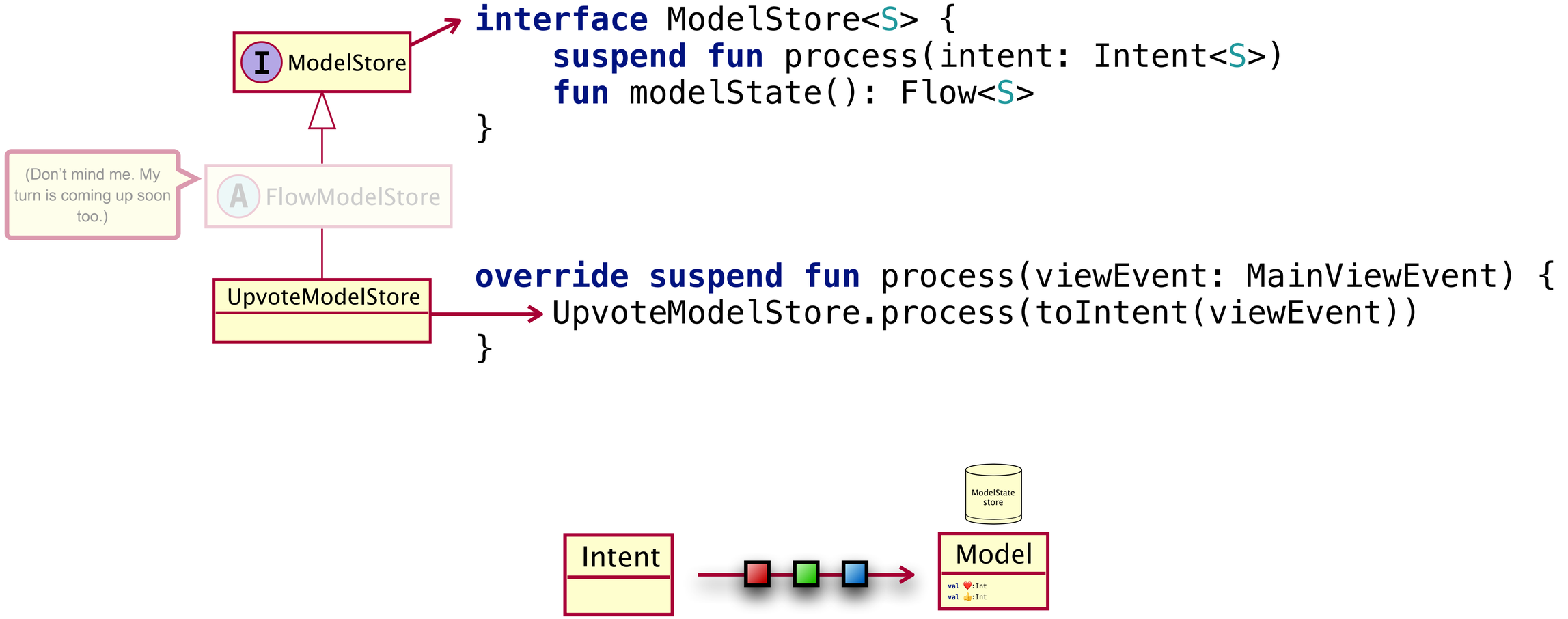


```
interface IntentFactory<E> {
    suspend fun process(viewEvent:E)
}

object MainViewIntentFactory : IntentFactory<MainViewEvent> {

    override suspend fun process(viewEvent: MainViewEvent) {
        UpvoteModelStore.process(toIntent(viewEvent))
    }

    private fun toIntent(viewEvent: MainViewEvent):Intent<UpvoteModel> {
        return when (viewEvent) {
            MainViewEvent.LoveItClick -> AddHeart()
            MainViewEvent.ThumbsUpClick -> AddThumb()
        }
    }

}
```

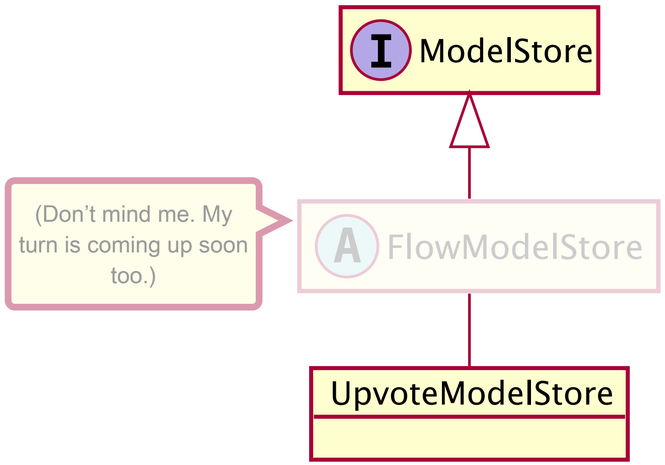


```
interface IntentFactory<E> {
    suspend fun process(viewEvent:E)
}

object MainViewIntentFactory : IntentFactory<MainViewEvent> {

    override suspend fun process(viewEvent: MainViewEvent) {
        UpvoteModelStore.process(toIntent(viewEvent))
    }

    private fun toIntent(viewEvent: MainViewEvent):Intent<UpvoteModel> {
        return when (viewEvent) {
            MainViewEvent.LoveItClick -> AddHeart()
            MainViewEvent.ThumbsUpClick -> AddThumb()
        }
    }

}
```

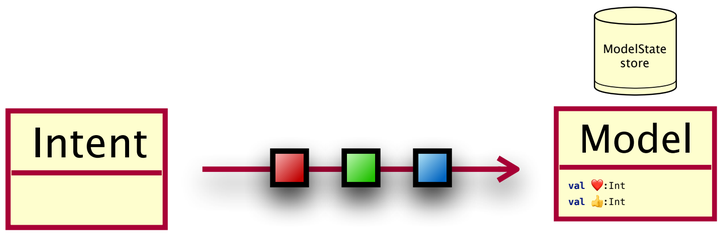


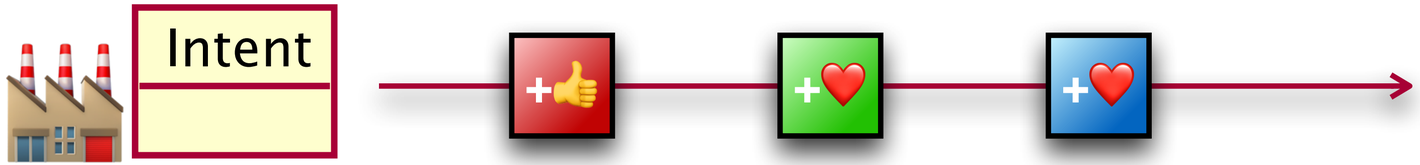

I ModelStore
(Don't mind me. My turn is coming up soon too.)
A FlowModelStore
UpvoteModelStore
interface ModelStore<S> {
suspend fun process(intent: Intent<S>)
fun modelState(): Flow<S>
}
override suspend fun process(viewEvent: MainViewEvent) {
UpvoteModelStore.process(toIntent(viewEvent))
}

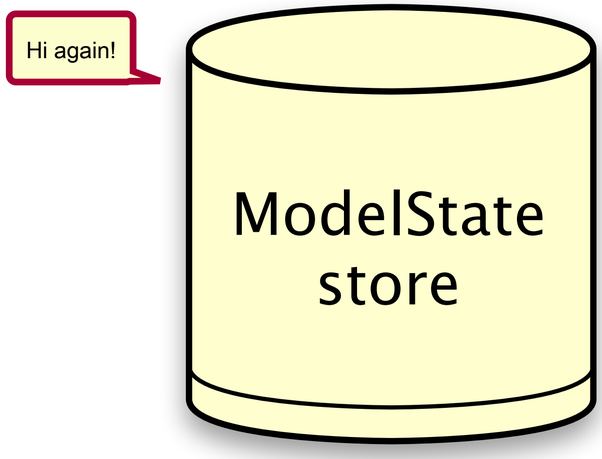
ModelState store
Intent
Model
val ❤️:Int
val 👍:Int



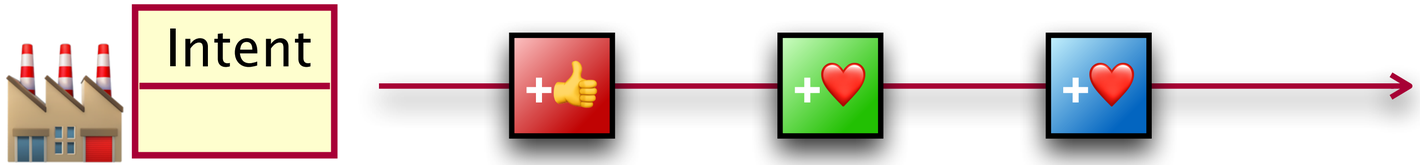

```
interface ModelStore<S> {
    suspend fun process(intent: Intent<S>)
    fun modelState(): Flow<S>
}
```







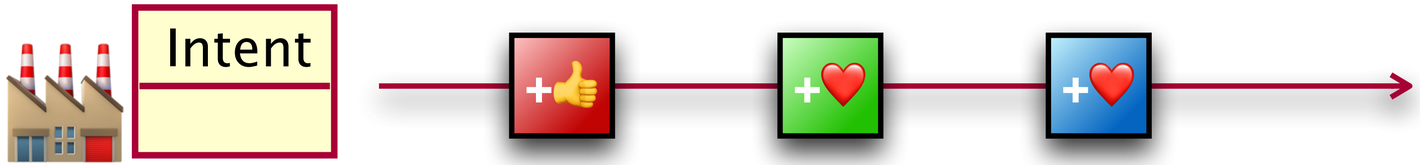

```
interface ModelStore<S> {
    suspend fun process(intent: Intent<S>)
    fun modelState(): Flow<S>
}
```



ModelState store

```
val intents = Channel<Intent<S>>()
```

```
suspend fun process(intent: Intent<S>) {
    intents.send(intent)
}
```



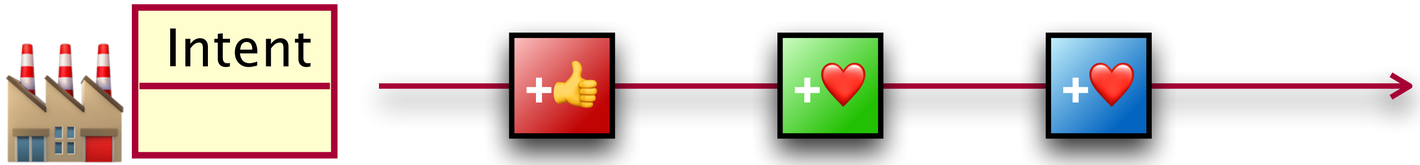
ModelState store

```
val intents = Channel<Intent<S>>()
```

```
suspend fun process(intent: Intent<S>) {
    intents.send(intent)
}
```

```
val store = ConflatedBroadcastChannel(startingState)
```

```
override fun modelState(): Flow<S> {
    return store.asFlow()
}
```

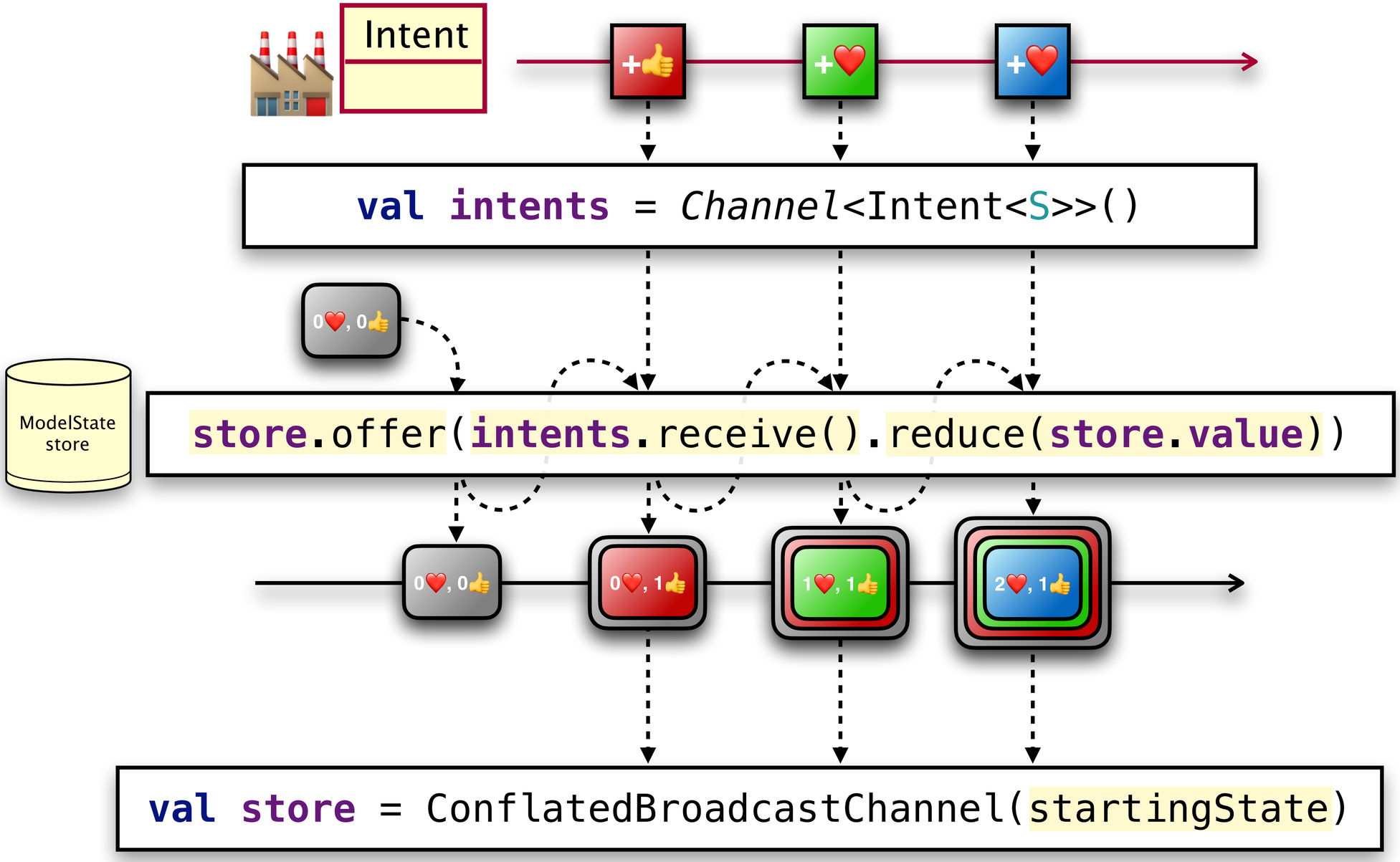


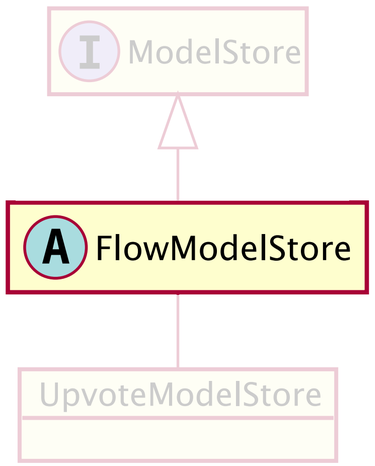
ModelState
store

```
val intents = Channel<Intent<S>>()
```

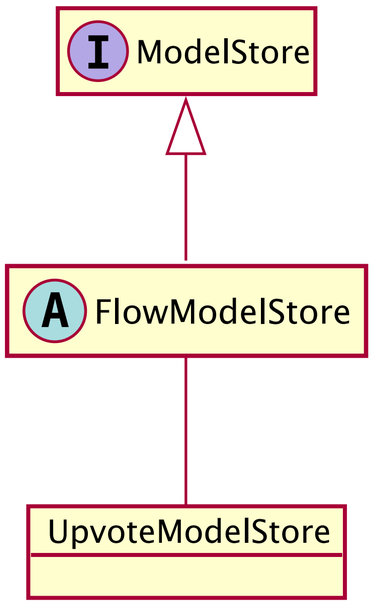
? ? ?

```
val store = ConflatedBroadcastChannel(startingState)
```


Intent
+👍
+❤️
+❤️
val intents = Channel<Intent<S>>()
0❤️, 0👍
ModelState
store
store.offer(intents.receive().reduce(store.value))
0❤️, 0👍
0❤️, 1👍
1❤️, 1👍
2❤️, 1👍
val store = ConflatedBroadcastChannel(startingState)



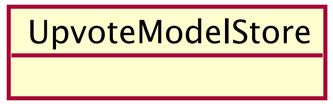

```kotlin
open class FlowModelStore<S>(startingState: S) : ModelStore<S> {
    private val scope = MainScope()
    private val intents = Channel<Intent<S>>()
    private val store = ConflatedBroadcastChannel(startingState)

    init {
        // Reduce from MainScope()
        scope.launch {
            while (isActive) store.offer(intents.receive().reduce(store.value))
        }
    }

    // Could be called from any coroutine scope/context.
    override suspend fun process(intent: Intent<S>) {
        intents.send(intent)
    }

    override fun modelState(): Flow<S> {
        return store.asFlow()
    }

    fun close() {
        intents.close()
        store.close()
        scope.cancel()
    }
}
```




```
interface ModelStore<S> {
    suspend fun process(intent: Intent<S>)
    fun modelState(): Flow<S>
}
```

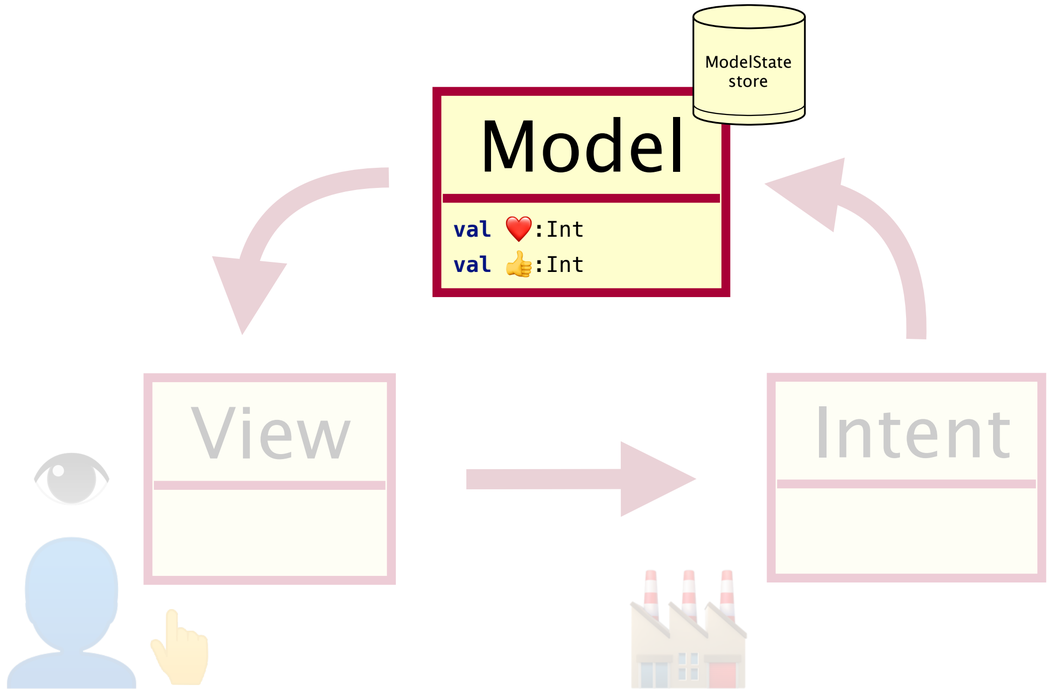
```
open class FlowModelStore<S>(startingState: S) : ModelStore<S> {
        // ...
}
```

```
object UpvoteModelStore :
    FlowModelStore<UpvoteModel>(UpvoteModel(0, 0))
```

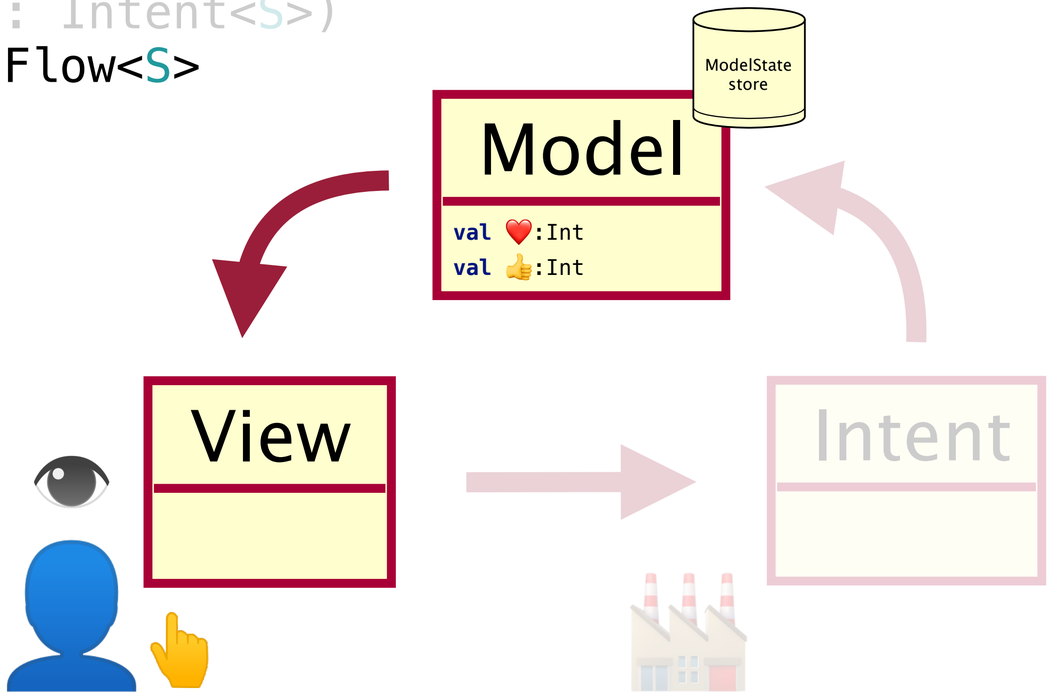


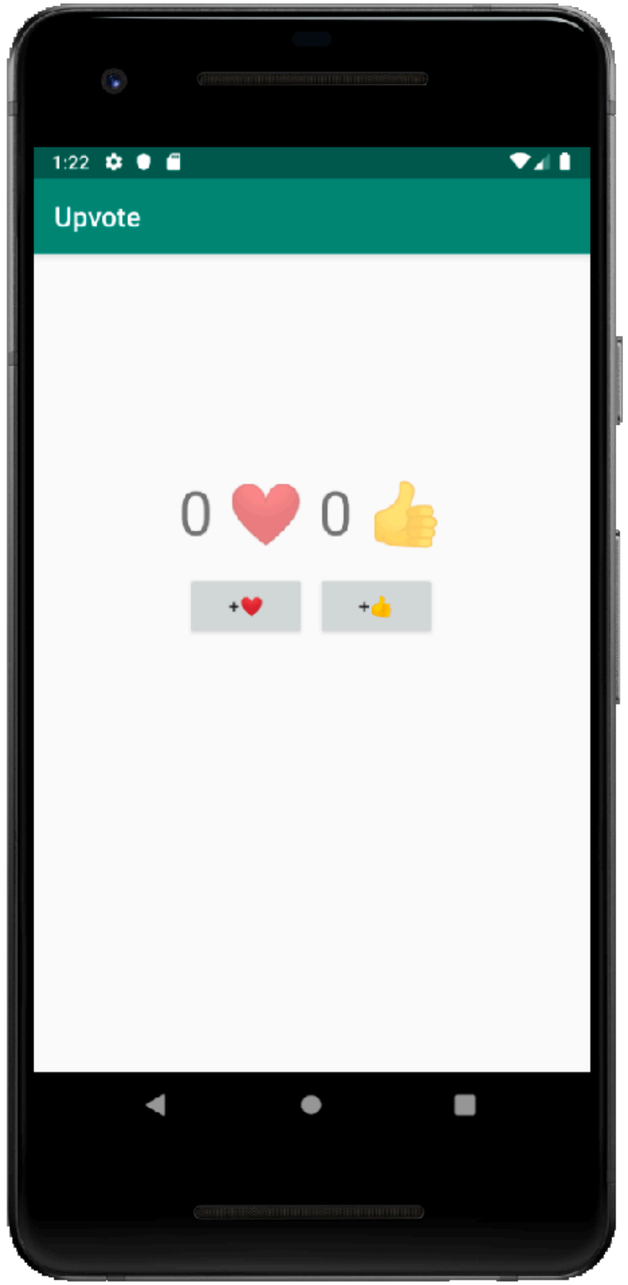

```
object UpvoteModelStore :
    FlowModelStore<UpvoteModel>(UpvoteModel(0, 0))
```

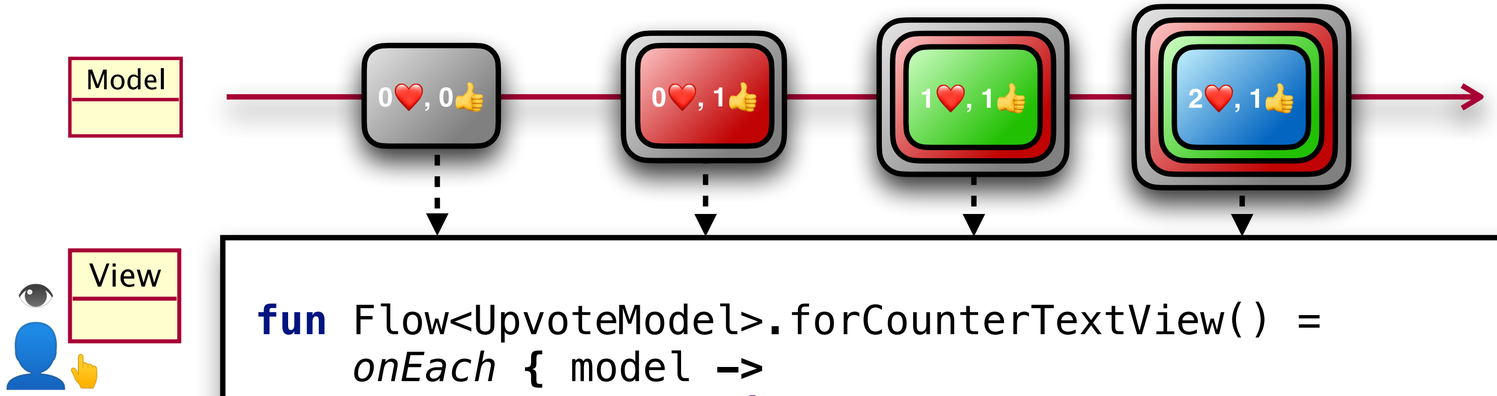
```
object UpvoteModelStore
```



```
object UpvoteModelStore

interface ModelStore<S> {
    fun process(intent: Intent<S>)
    fun modelState(): Flow<S>
}
```

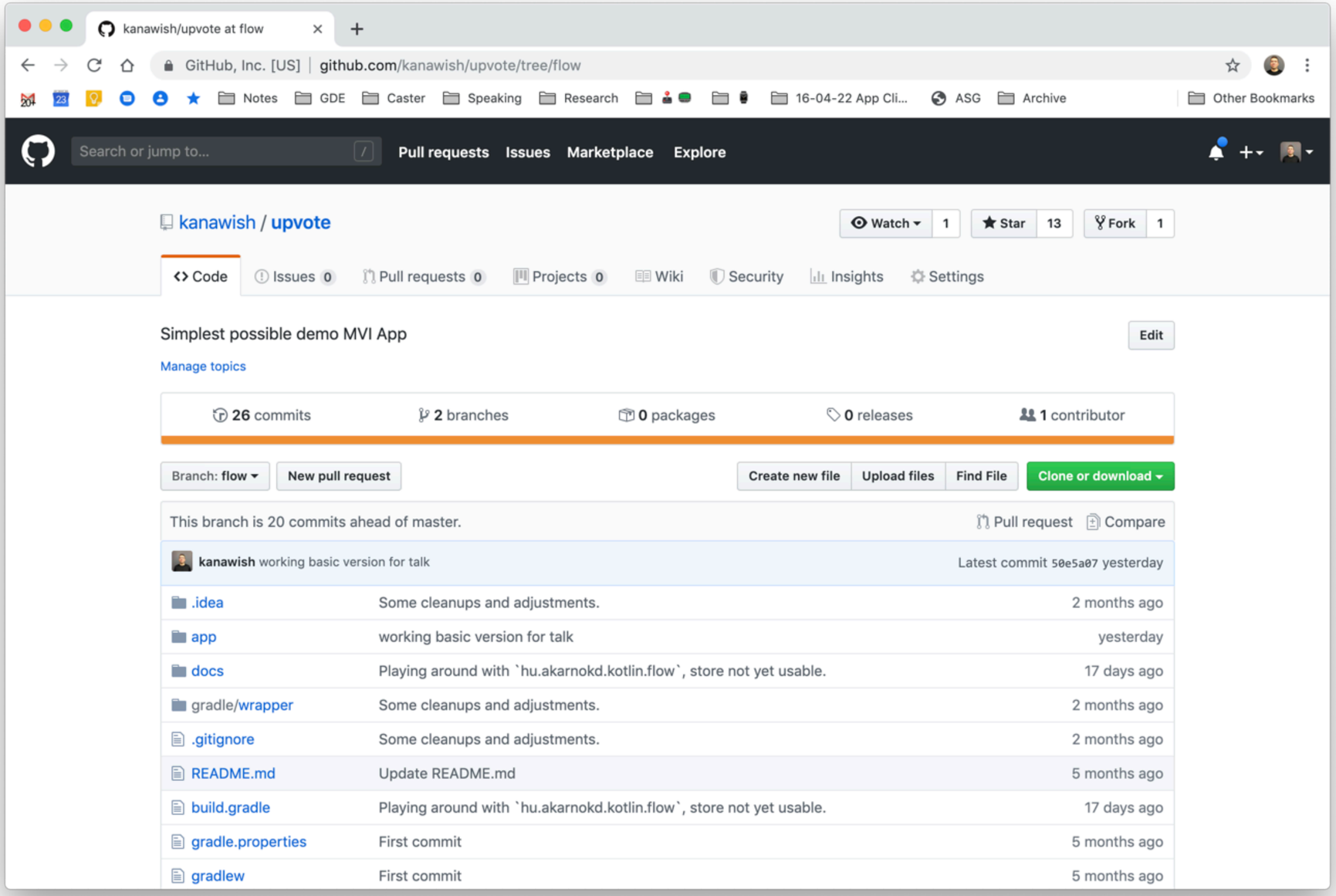




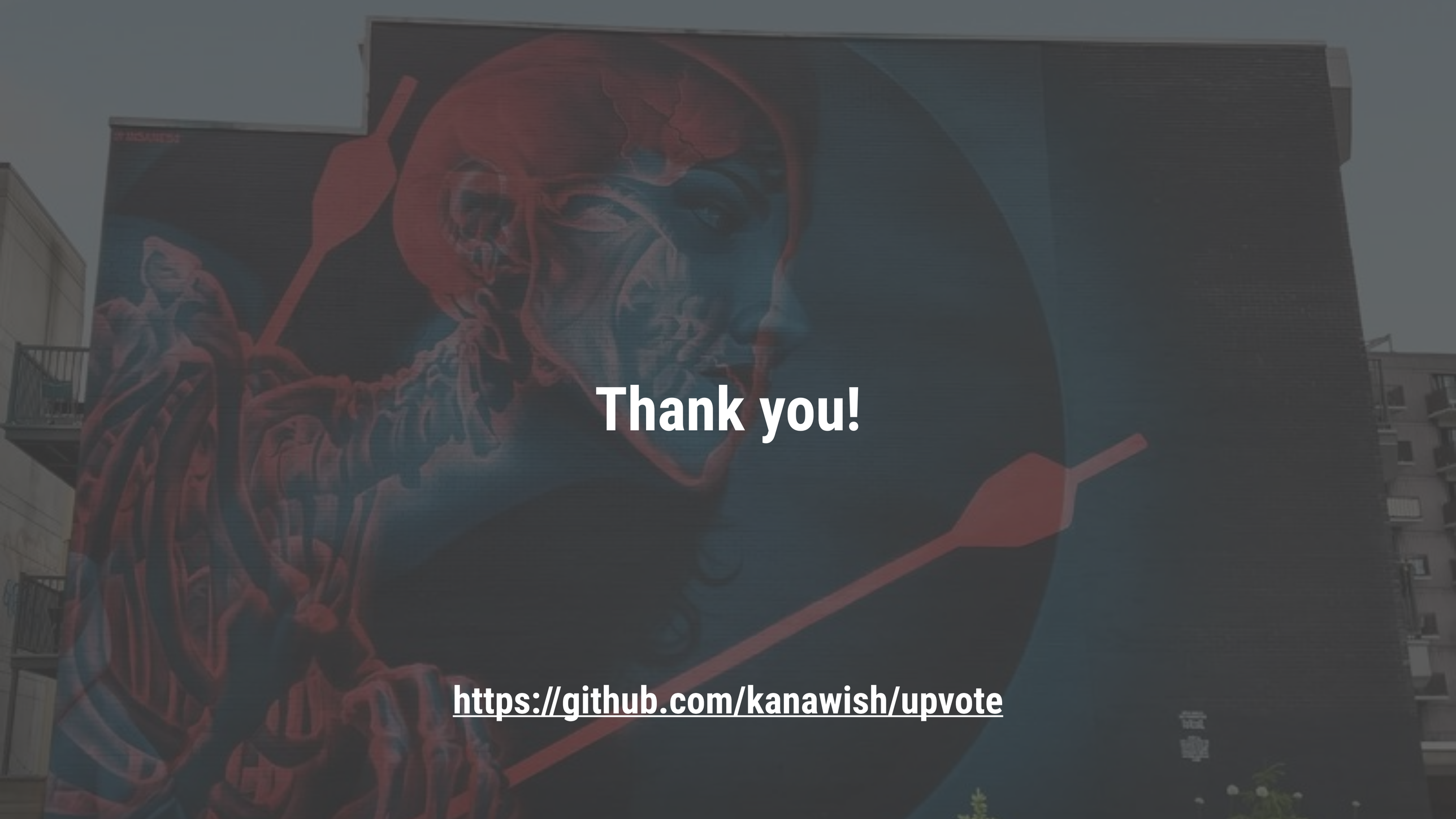



```
fun Flow<UpvoteModel>.forCounterTextView() =
    onEach { model ->
        counterTextView.text =
            "{model.hearts} ❤️ {model.thumbs} 👍"
    }

// Input(s)
UpvoteModelStore
    .modelState()
    .forCounterTextView()
    .launchIn(scope)
```

Live demo?



https://github.com/kanawish/upvote



Thank you!

https://github.com/kanawish/upvote