1

Komponen Arsitektur Android

View / tampilan

1. Compose

Fungsi

Composable function

Compose

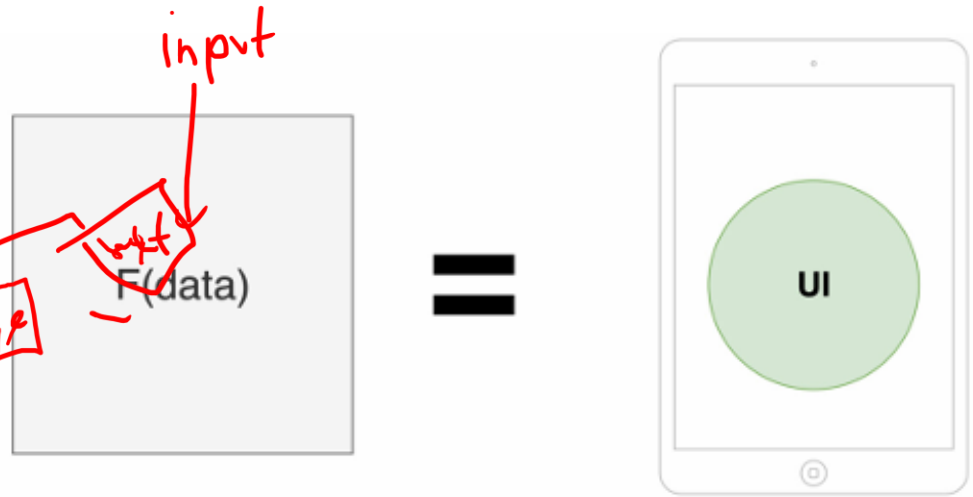
Suggestive Button

Button

Row

Image

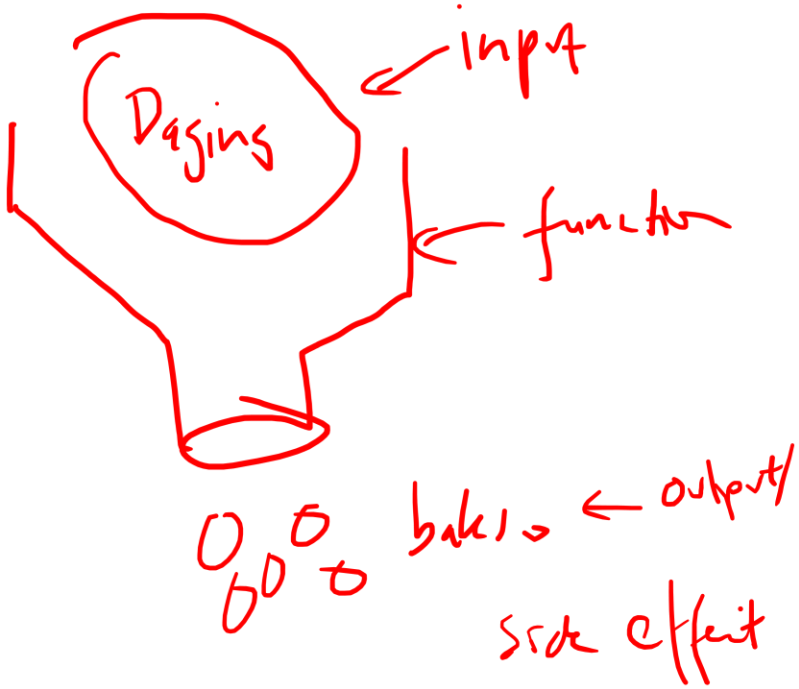
In compose, UI is a function of data



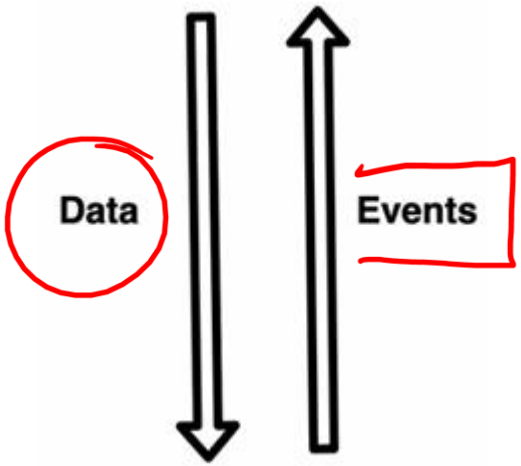
anotasi

named argument

```
@Composable
fun SuggestiveButton() {
    Button(onClick = { }) {
        Row() {
            Image(painter =
                    painterResource(R.drawable.drawable),
                contentDescription = "")
            Text(text = "Press me")
        }
    }
}
```



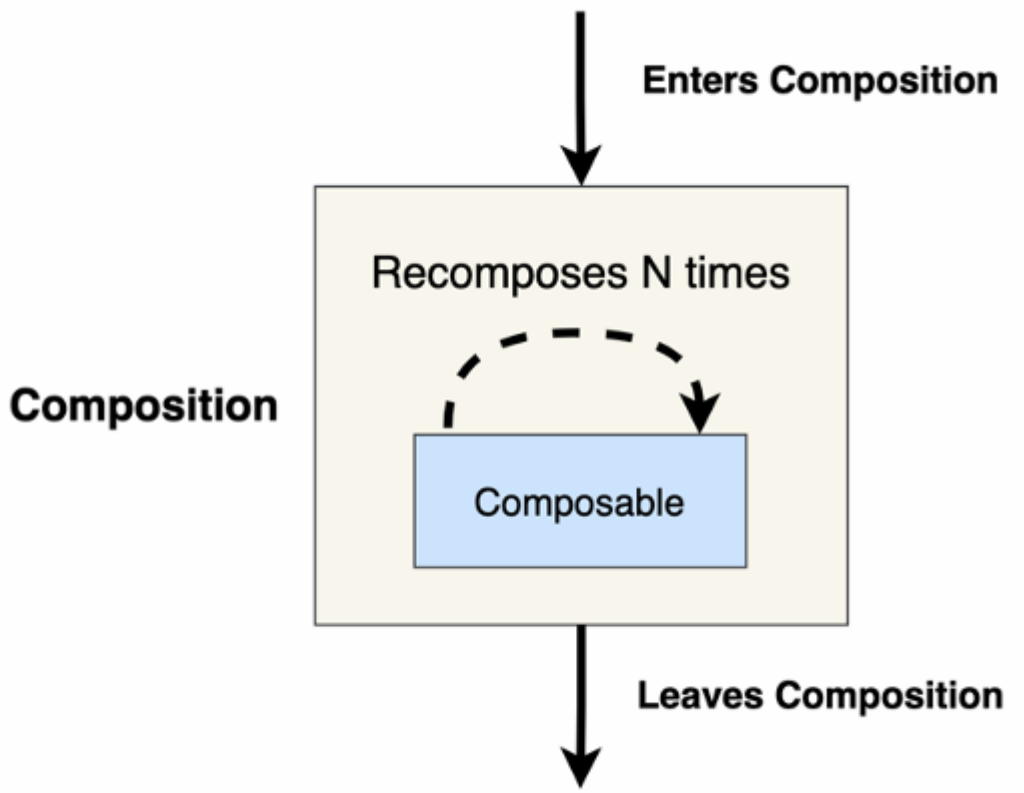
Unidirectional flow of data




```
@Composable
fun MailButton(
    mailId: Int,
    mailPressedCallback: (Int) -> Unit
) {
    Button(onClick = { mailPressedCallback(mailId) }) {
        Text(text = "Expand mail $mailId")
    }
}
```

Recomposition

- When inputs change, Compose automatically triggers the recomposition process for us and rebuilds the UI widget tree, redrawing the widgets emitted by the composables so that they display the newly received data.
- Yet recomposing the entire UI hierarchy is computationally expensive, which is why Compose only calls the functions that have new input while skipping the ones whose input hasn't changed.
- Optimizing the process of rebuilding the composable tree is a complex job and is usually referred to as intelligent recomposition.



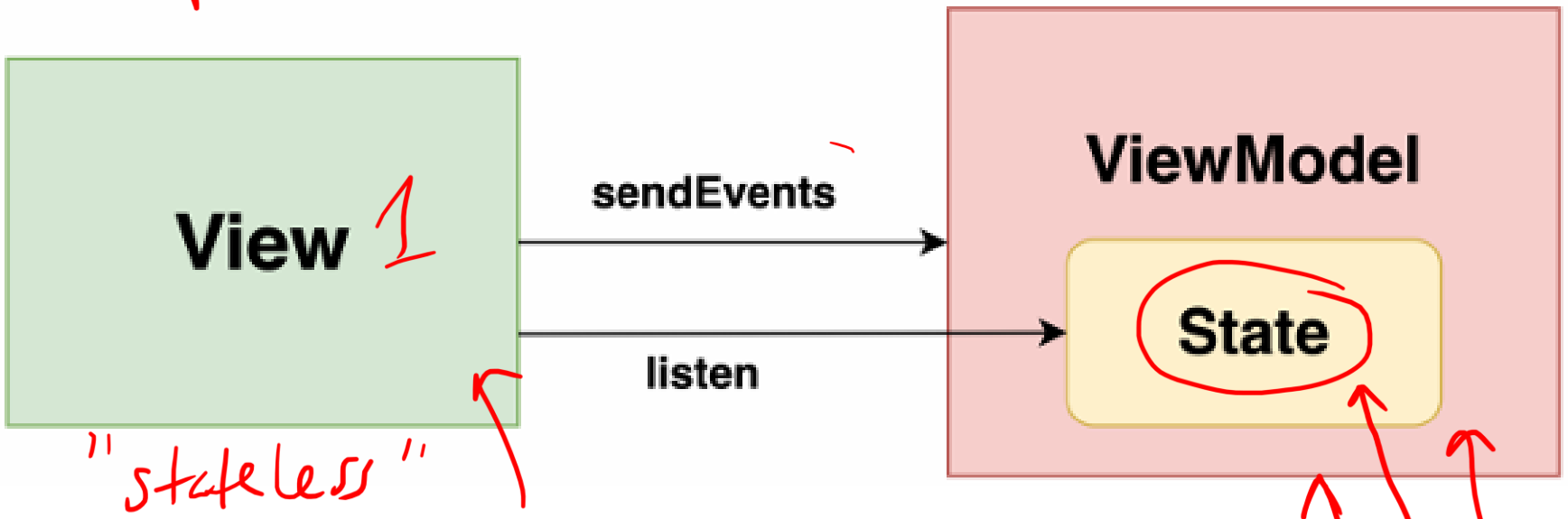

```
var seconds by mutableStateOf(0)
val stopWatchTimer = timer(period = 1000) { seconds++ }
    ...
@Composable
fun TimerText(seconds: Int) {
    Text(text = "Elapsed: $seconds")
}
```

Every time `stopWatchTimer` increases the value of the seconds state object, Compose triggers a recomposition that rebuilds the widget tree and redraws the composables with new arguments.

2. ViewModel

menampilkan state

menyimpan state

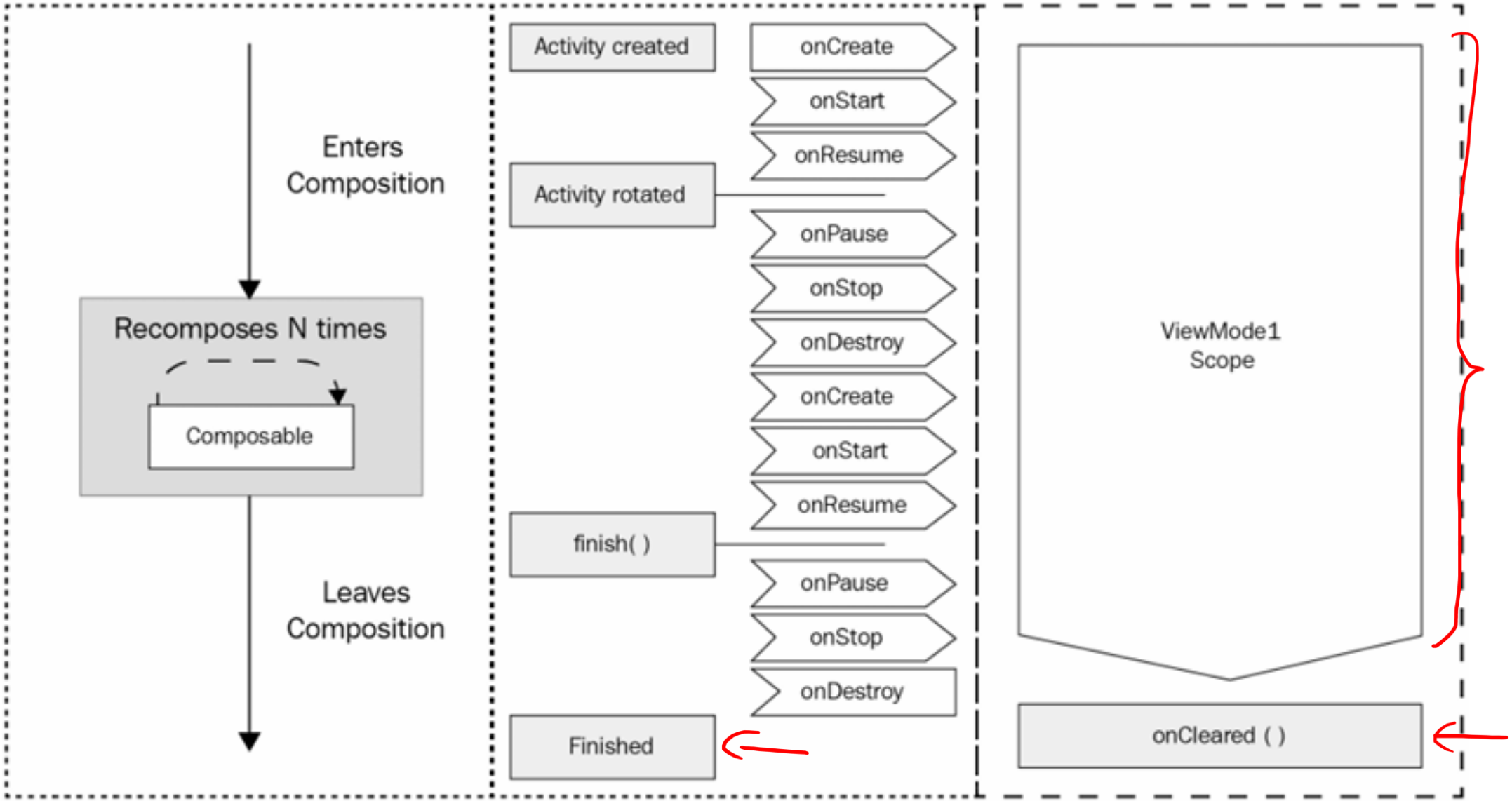


Separation of Concern

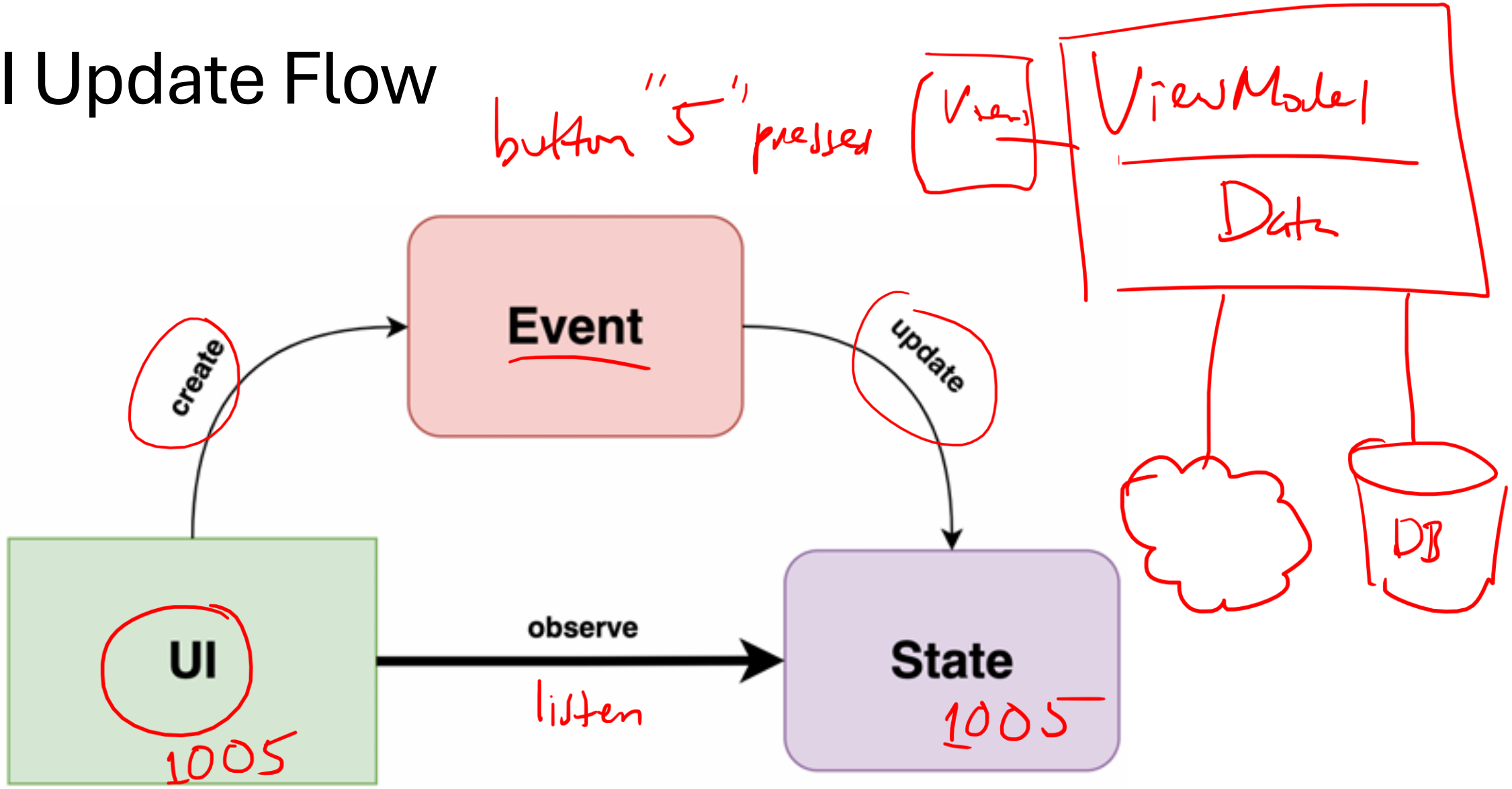
View 2

Note

`ViewModel` should not have a reference to a UI controller and should run independently of it. This reduces coupling between the UI layer and `ViewModel` and allows multiple UI components to reuse the same `ViewModel`.

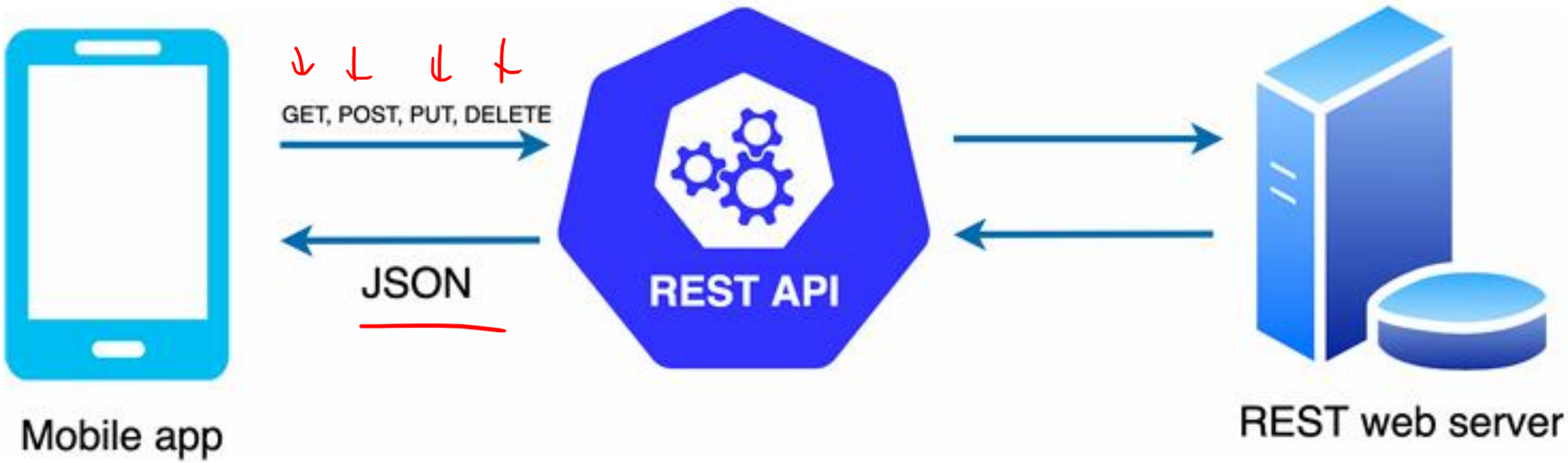

Enters Composition
Recomposes N times
Composable
Leaves Composition
Activity created
onCreate
onStart
onResume
Activity rotated
onPause
onStop
onDestroy
onCreate
onStart
onResume
finish( )
onPause
onStop
onDestroy
Finished
ViewMode1 Scope
onCleared ( )
Composable Composition
Activity Lifecycle
ViewModel Lifecycle

UI Update Flow



3. HTTP Communication

Retrofit ← library



"takes time"

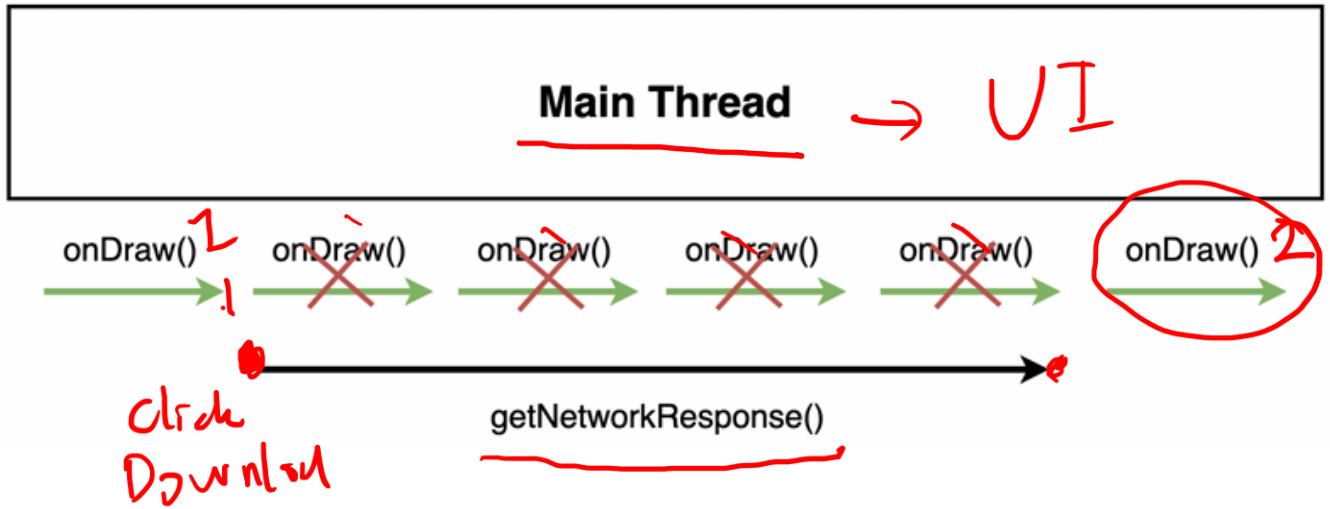
download data
saving data
} I/O

4. Async Operations with Coroutines

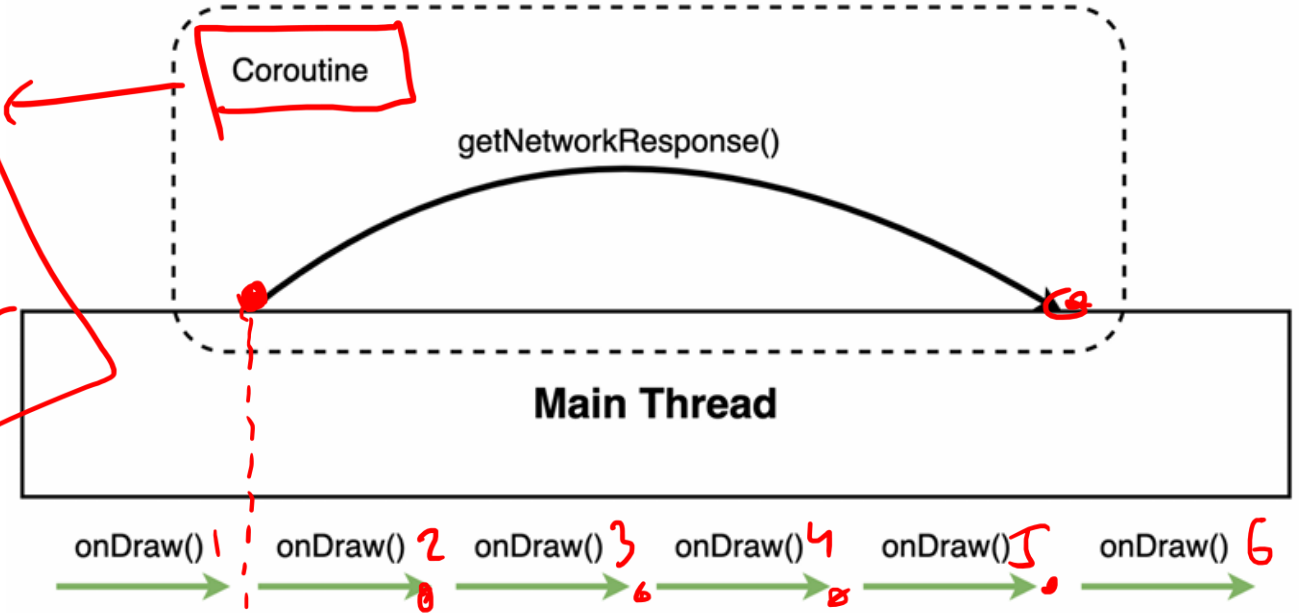
What is a coroutine?

- A coroutine is a concurrency design pattern for async work.
- A coroutine represents an instance of suspendable computation.
- A coroutine is a lightweight version of a thread but not a thread. Coroutines are light because creating coroutines doesn't allocate new threads.
- Like threads, coroutines can run in parallel, wait for each other, and communicate.
- Unlike threads, coroutines are very cheap: we can create thousands of them and pay very few penalties in terms of performance.

Operasi sinkron



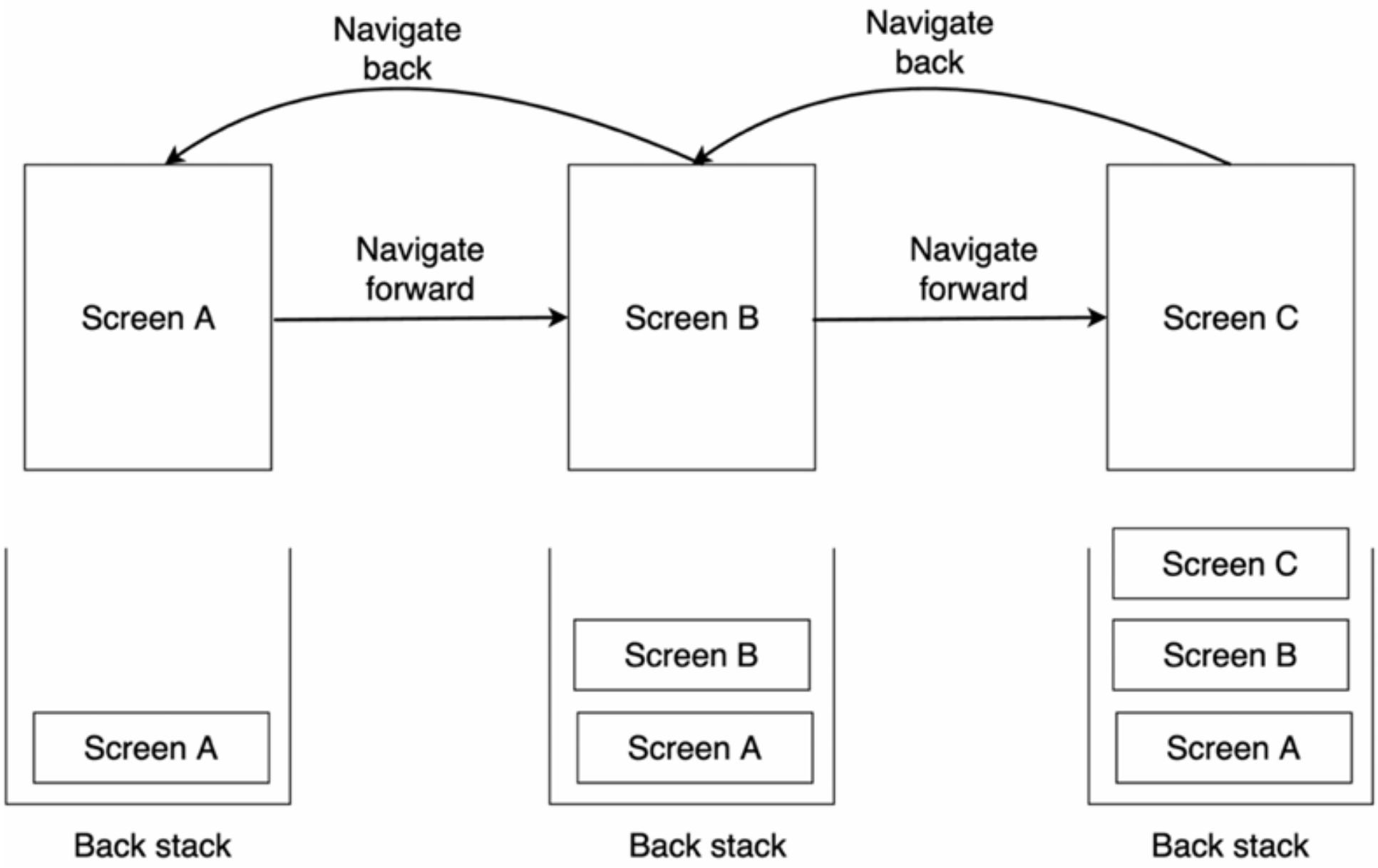
Operasi asinkron dengan coroutine



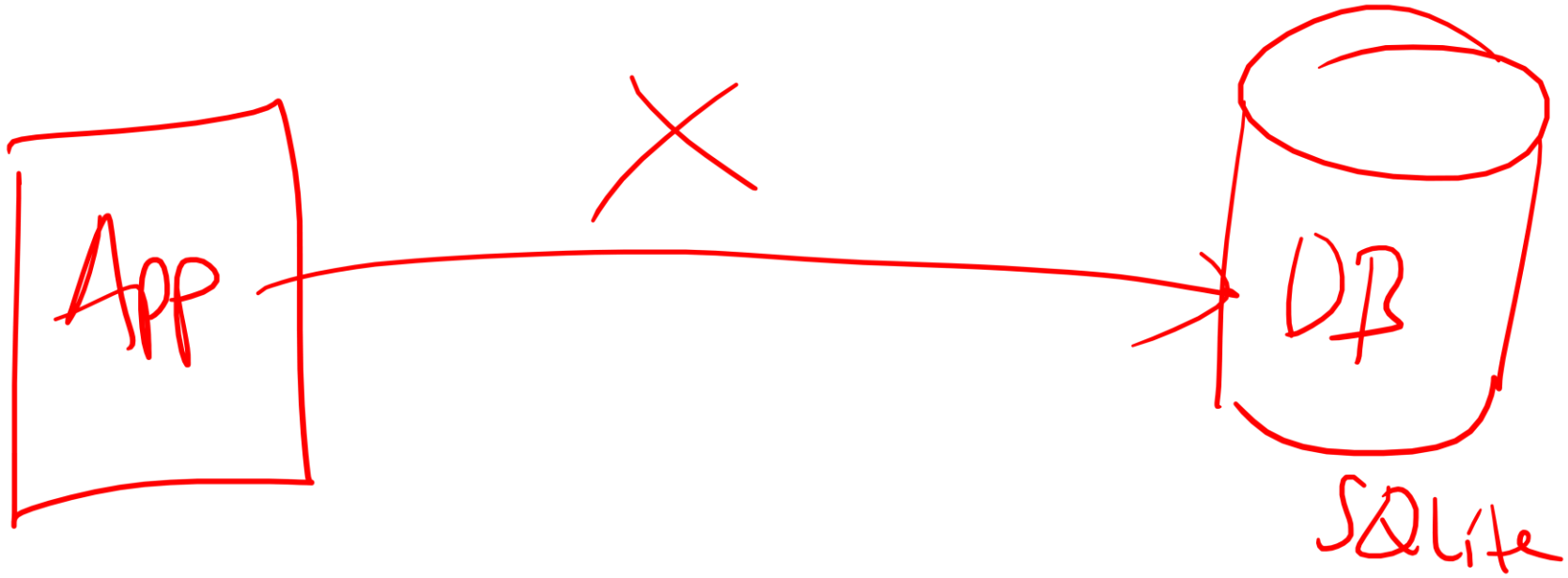
Pemrograman asinkron tp style sinkron

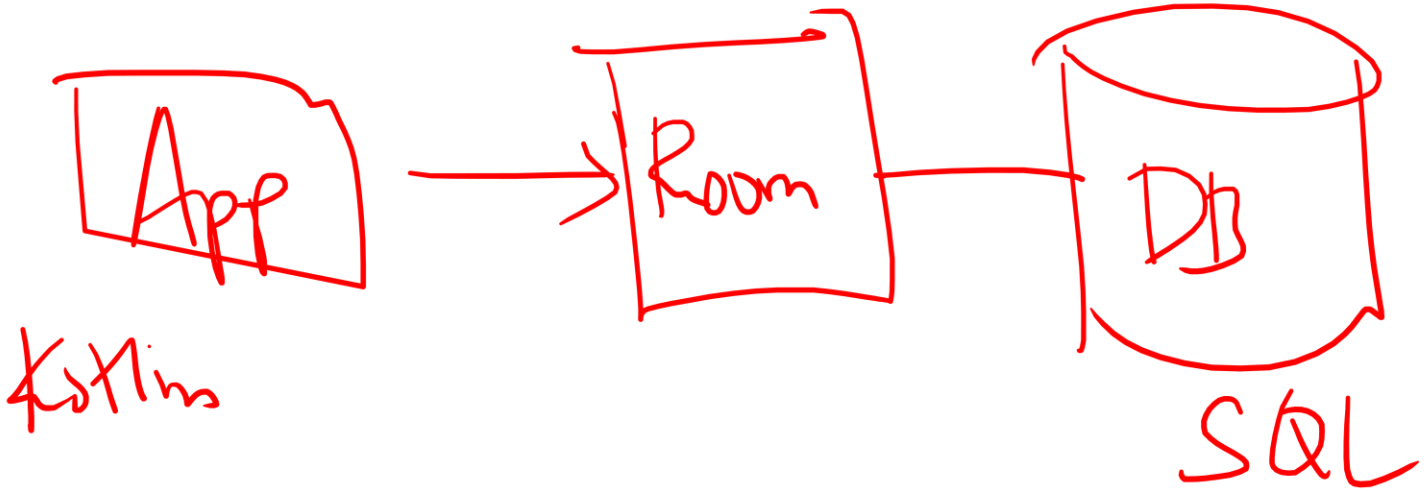
Pemrograman Asinkron ↓ Rumit

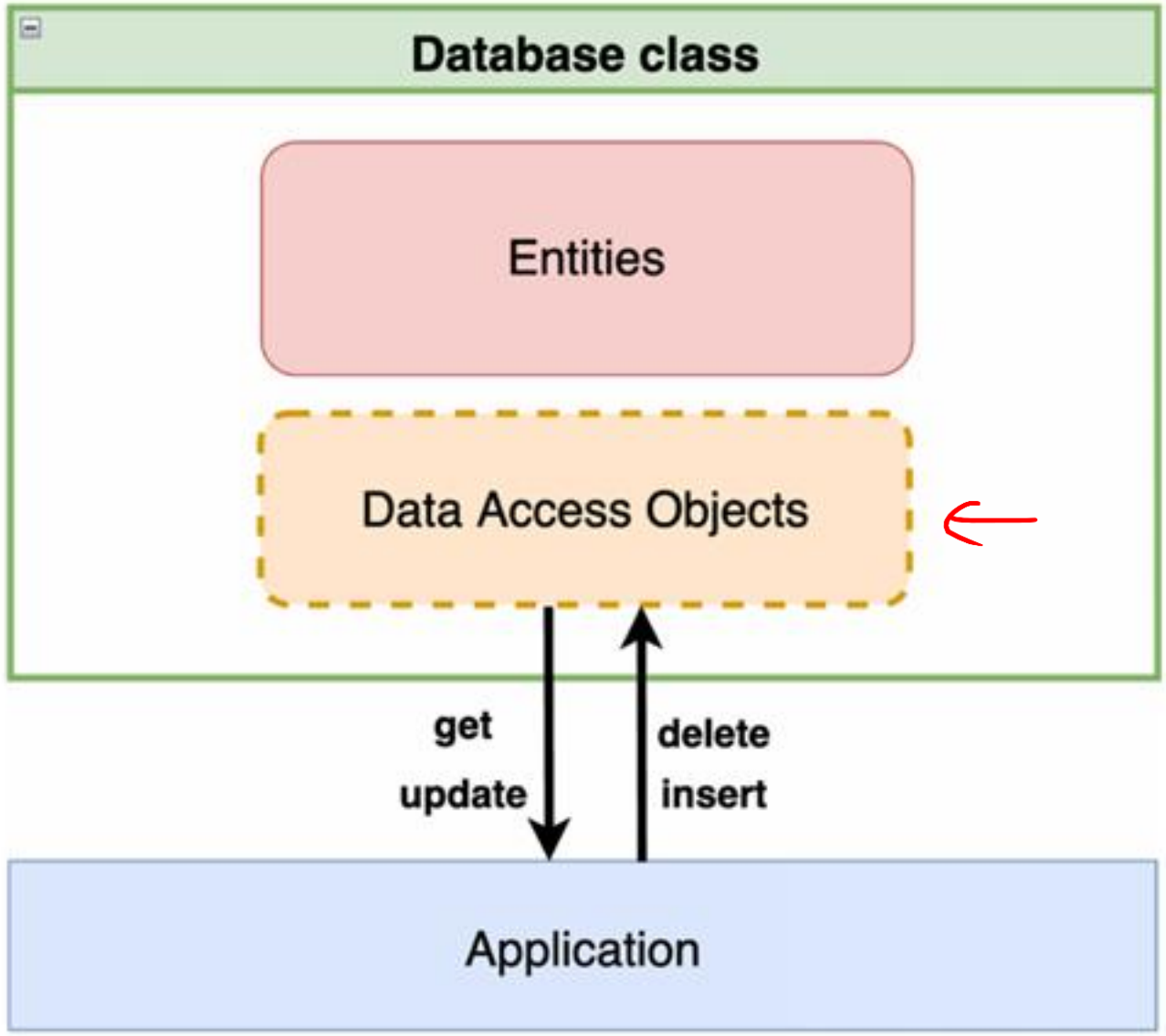
5. Navigation


Navigate back
Navigate back
Screen A
Navigate forward
Screen B
Navigate forward
Screen C
Screen A
Back stack
Screen B
Screen A
Back stack
Screen C
Screen B
Screen A
Back stack

6. Room






Database class
Entities
Data Access Objects
get
update
delete
insert
Application

7. Presentation Patterns

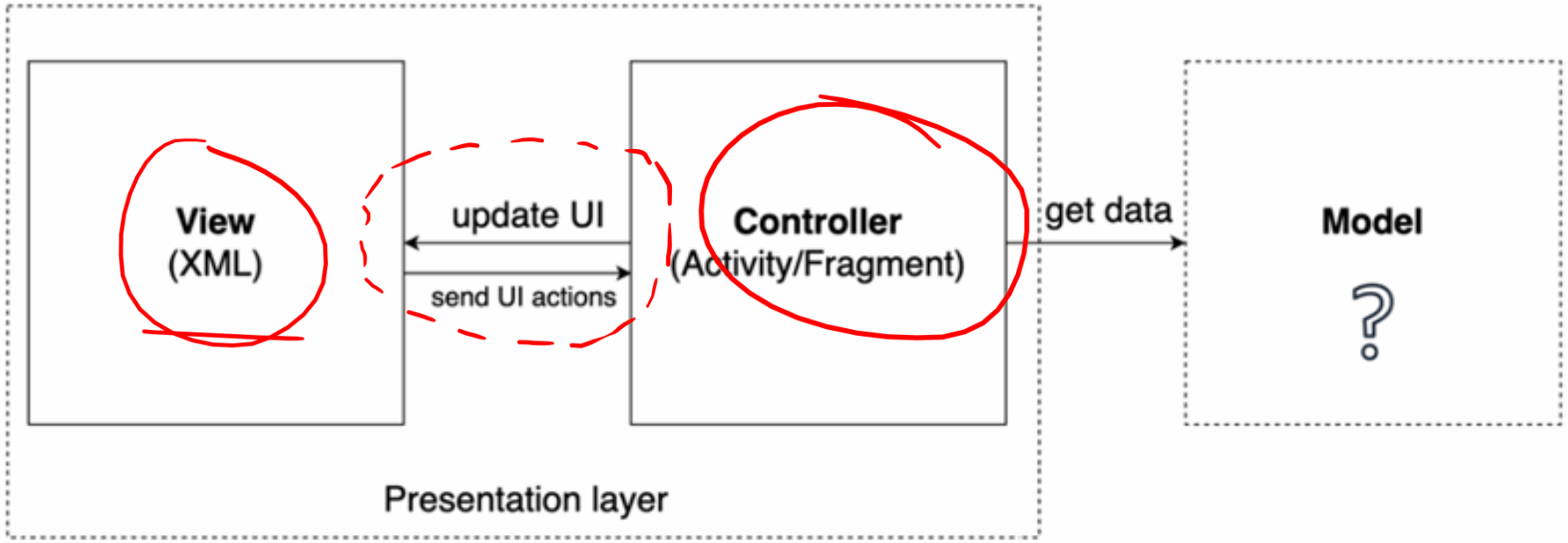
Problems with Activity or Fragment

- Fragile and difficult to scale
- Difficult to test
- Difficult to debug

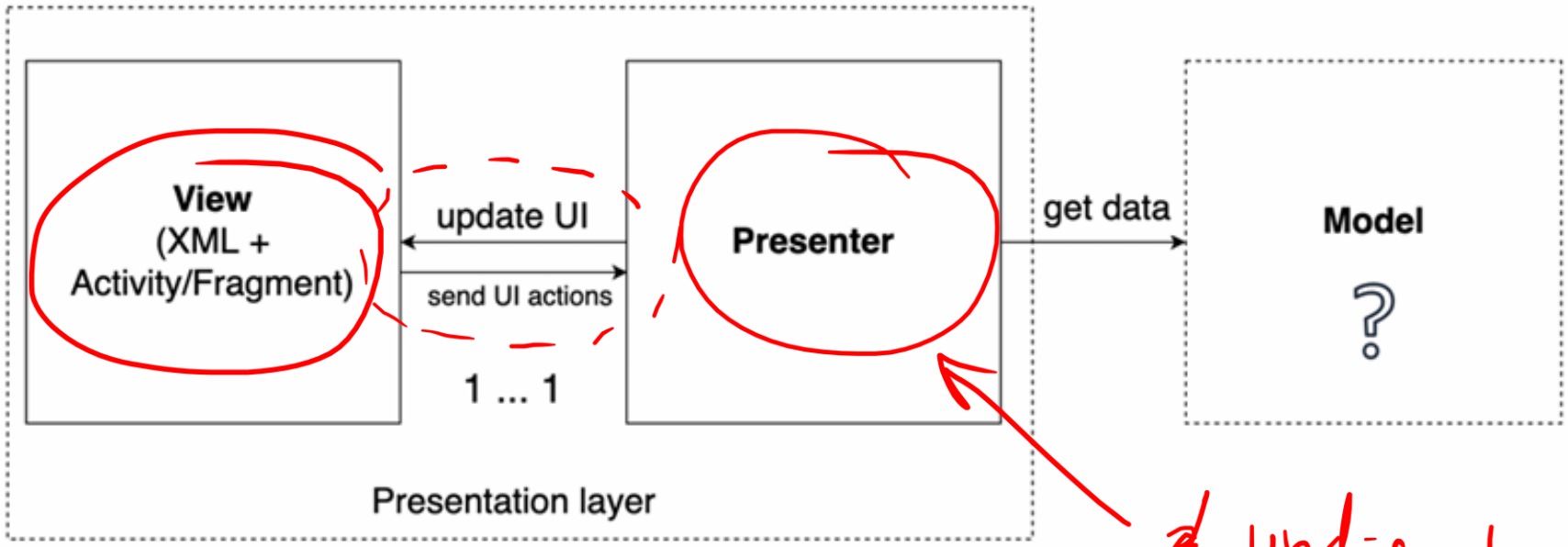
Solution: separation of concerns (SoC)

Single Responsibility

Model-View-Controller (MVC)

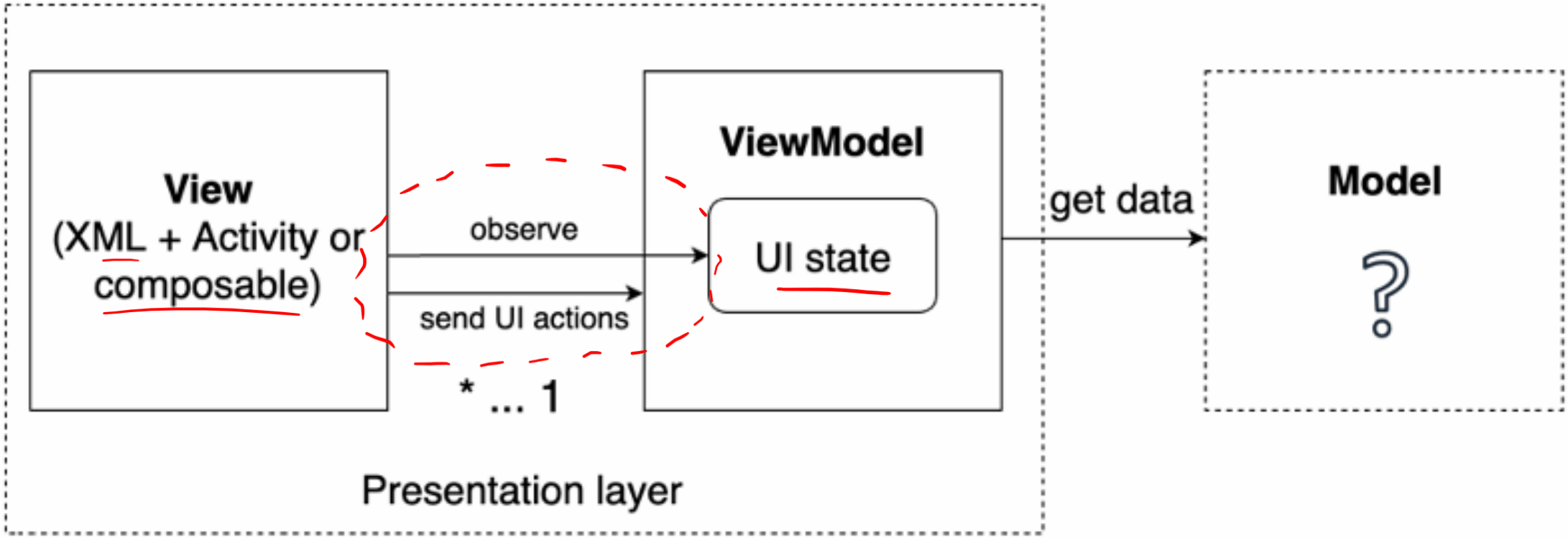


Model-View-Presenter (MVP)



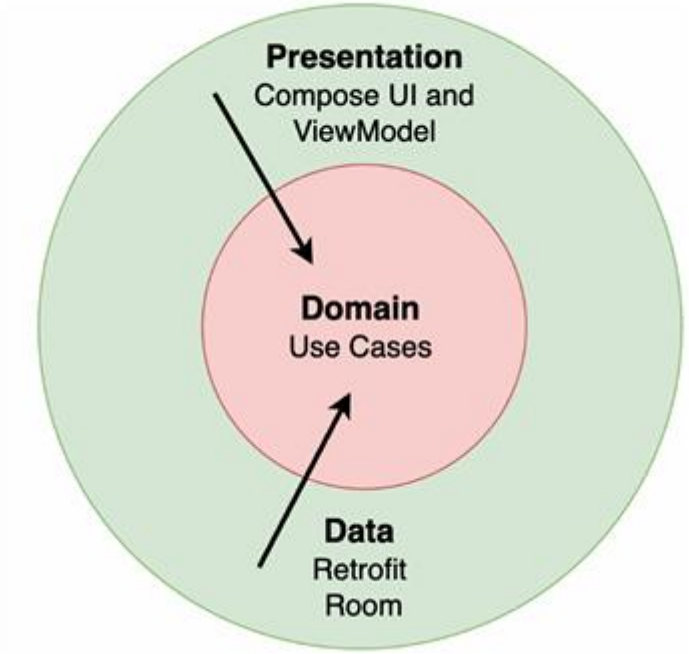
indirect manipulation the UI

Model-View-ViewModel (MVVM)



8. Clean Architecture

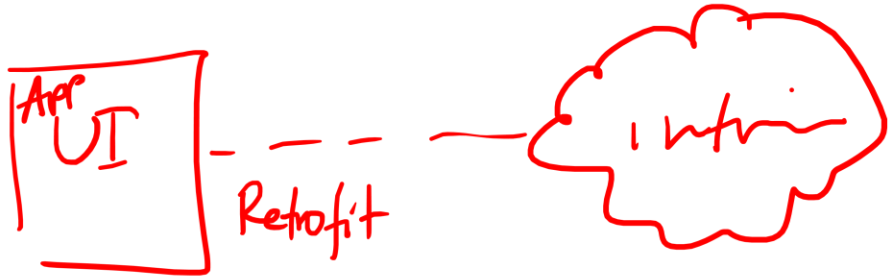
The Dependency Rule



unidirectional

The Dependency Rule states that within a project, dependencies can only point inward.

Dependency Injection (DI)

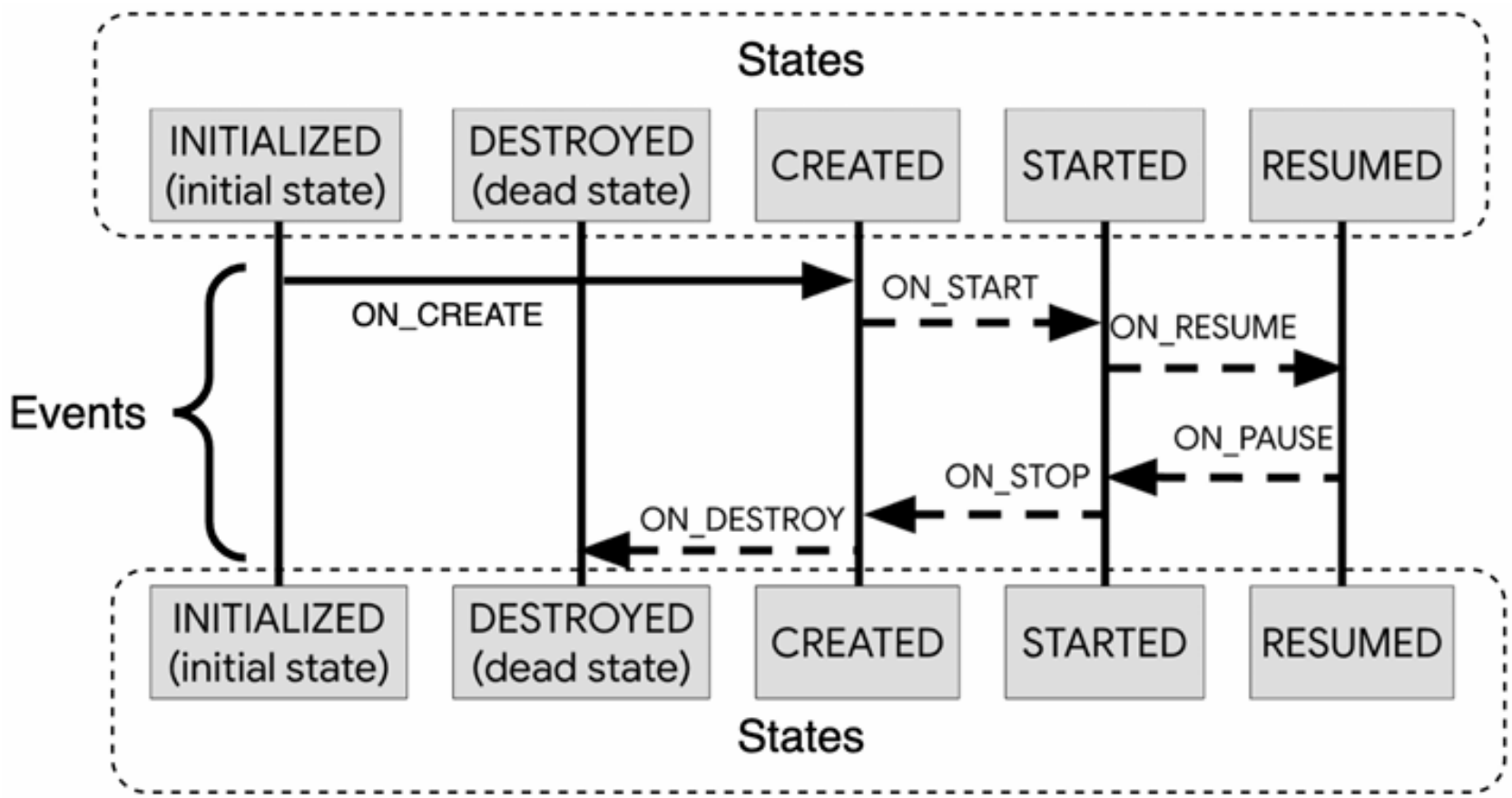


- DI represents the concept of providing the instances of the dependencies that a class needs, instead of having it construct them itself.
- Dependencies are other classes that a certain class depends on.
- DI advantages:
  - Write less boilerplate code
  - Write testable classes

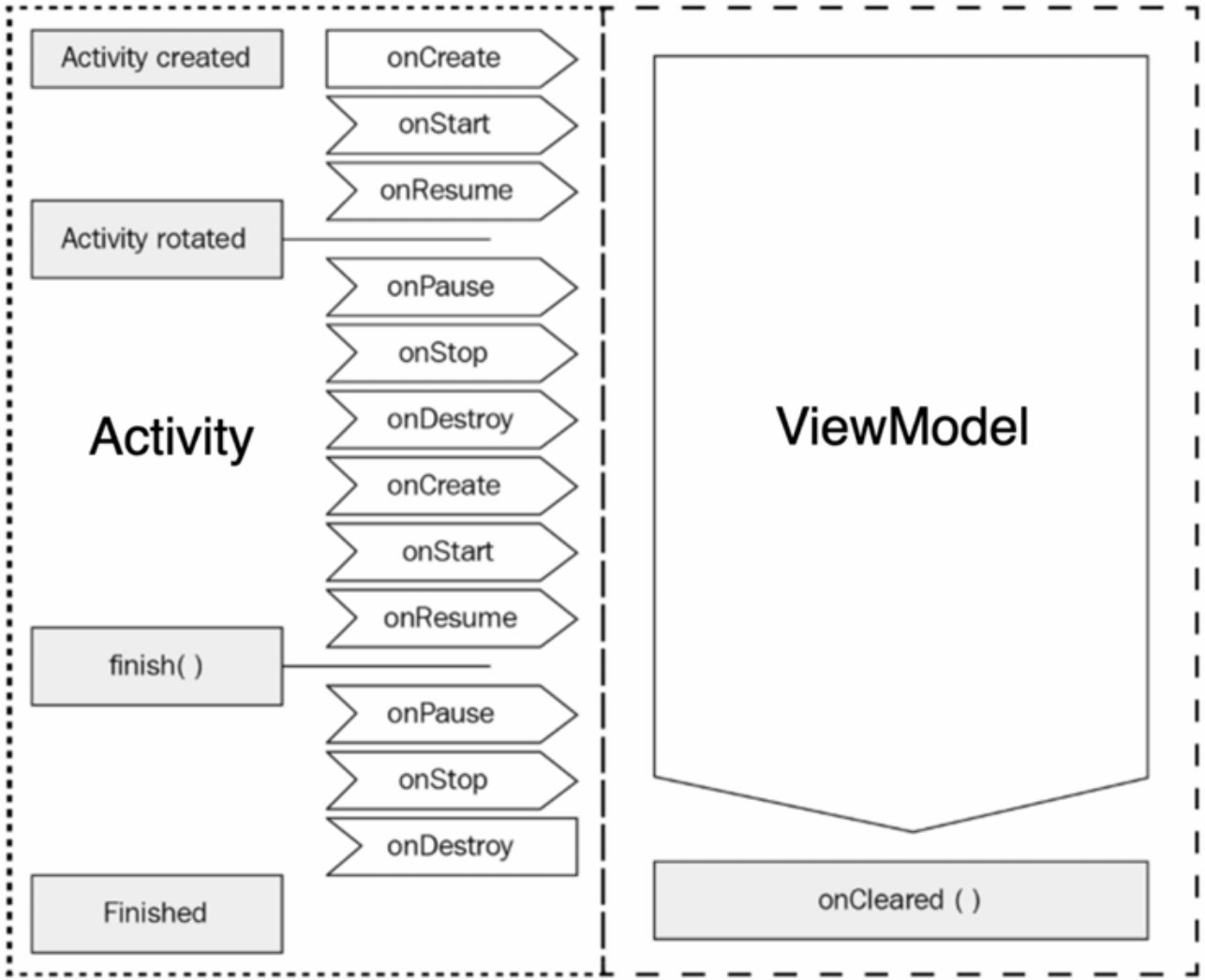
Holywood principles

9. Jetpack Lifecycle components

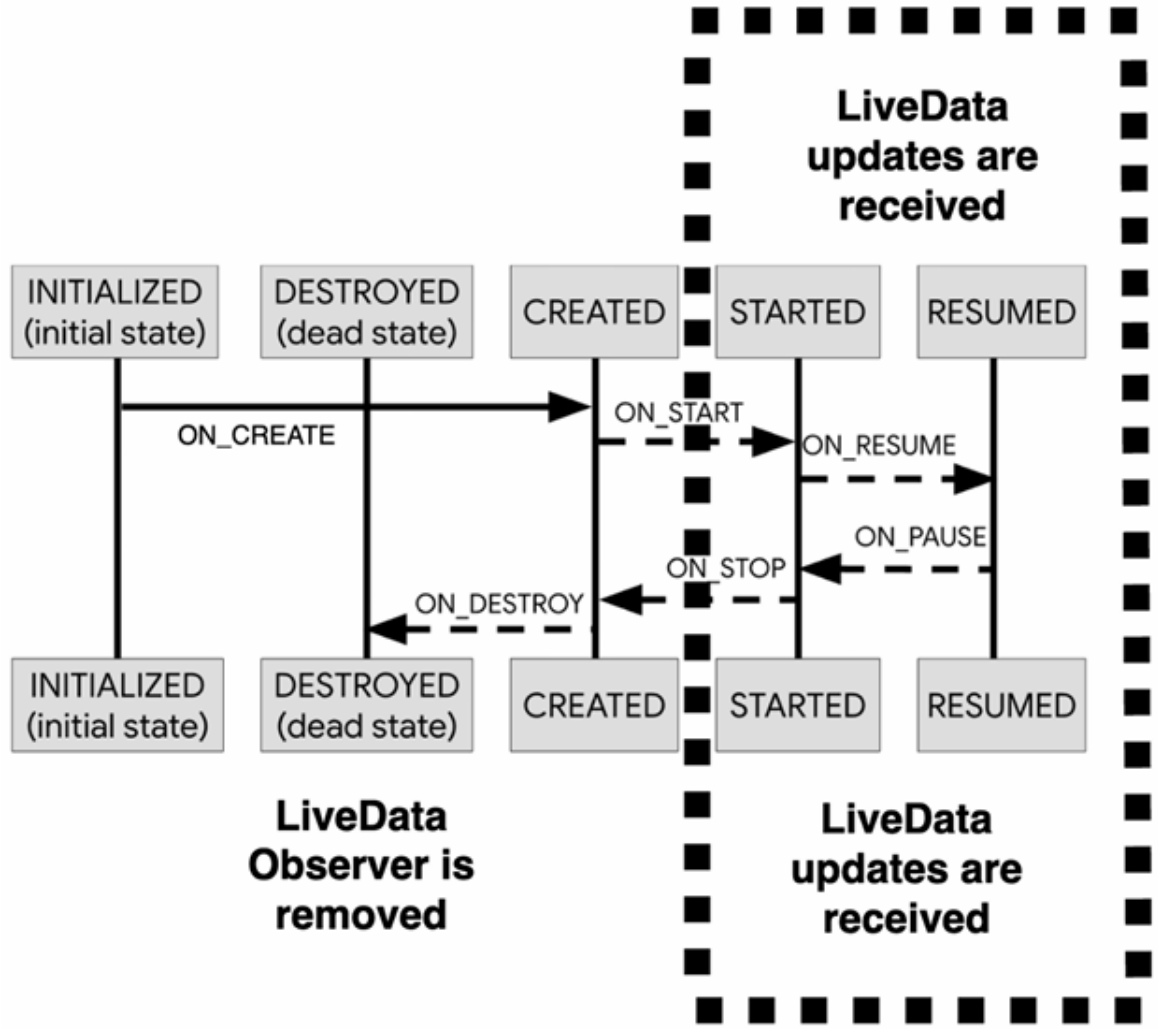
Activity Lifecycle



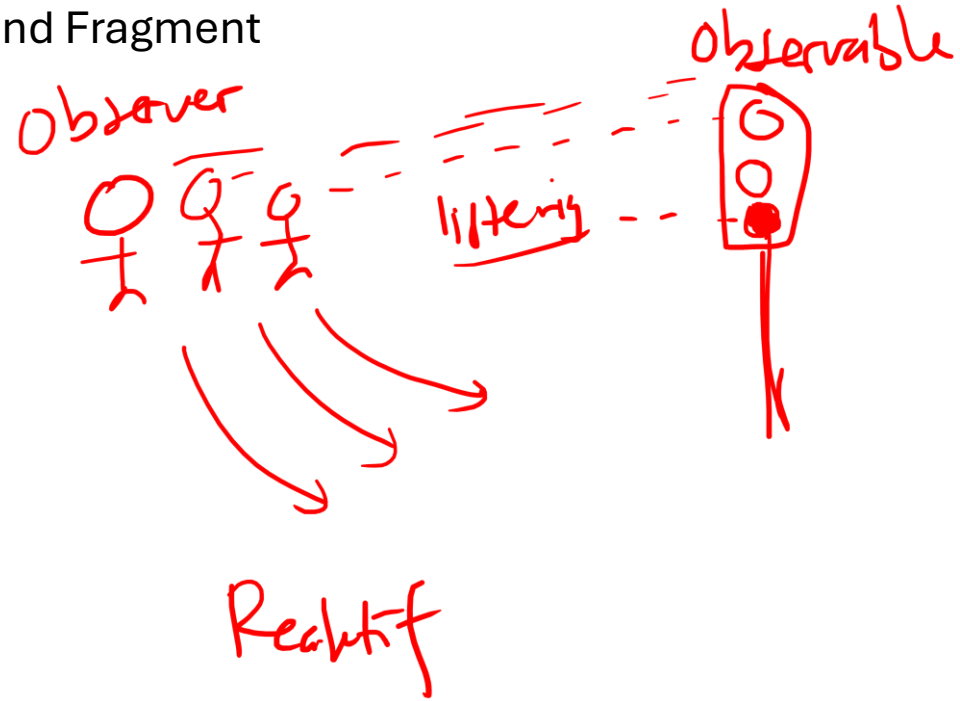
ViewModel Lifecycle



LiveData Lifecycle



LiveData is an observable data holder class that allows us to get data updates in a lifecycle-aware manner inside our Android components, such as Activity and Fragment



Sekian