TEA IN PROD

ARTUR BADRETDINOV, SQUIRE

ARTUR BADRETDINOV

Google Developer Expert,
Digital Nomad,
Director of Android Engineering at Squire

https://twitter.com/ArtursTwit
https://www.linkedin.com/in/gaket/

THE ELM ARCHITECTURE

IN PROD

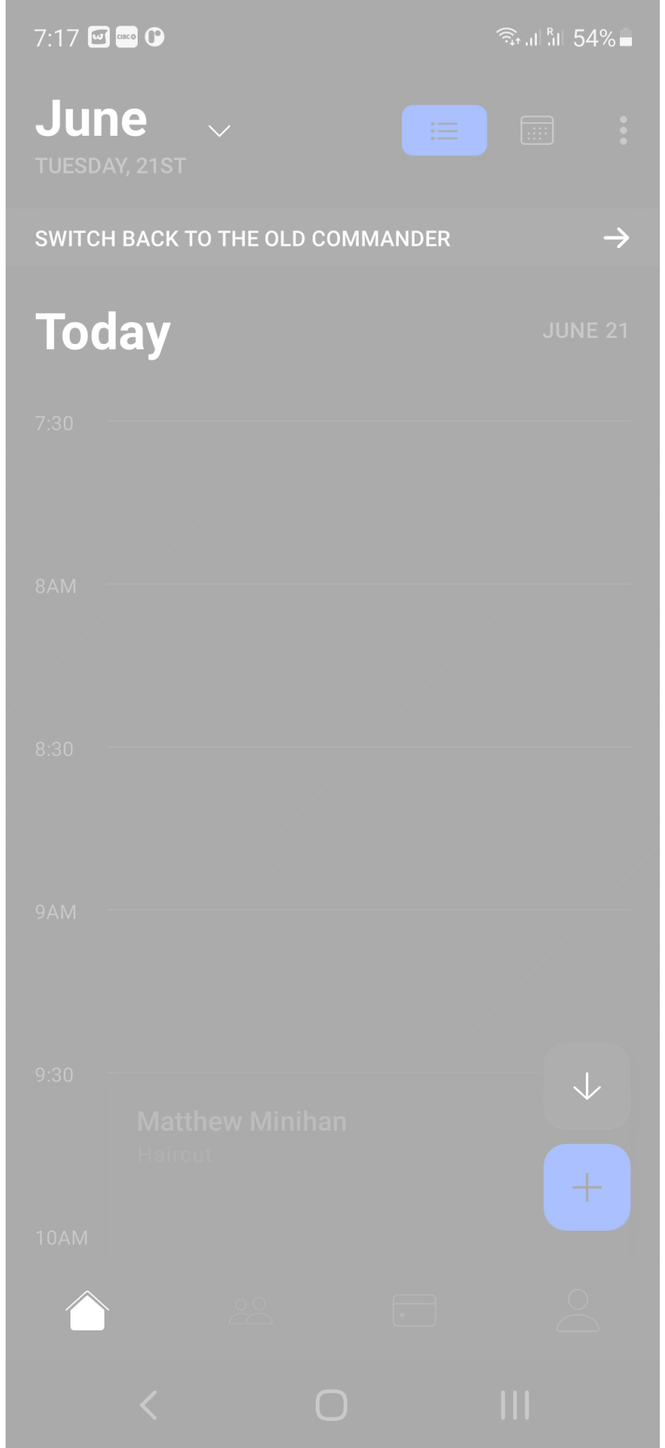
LIVE DEMO AHEAD

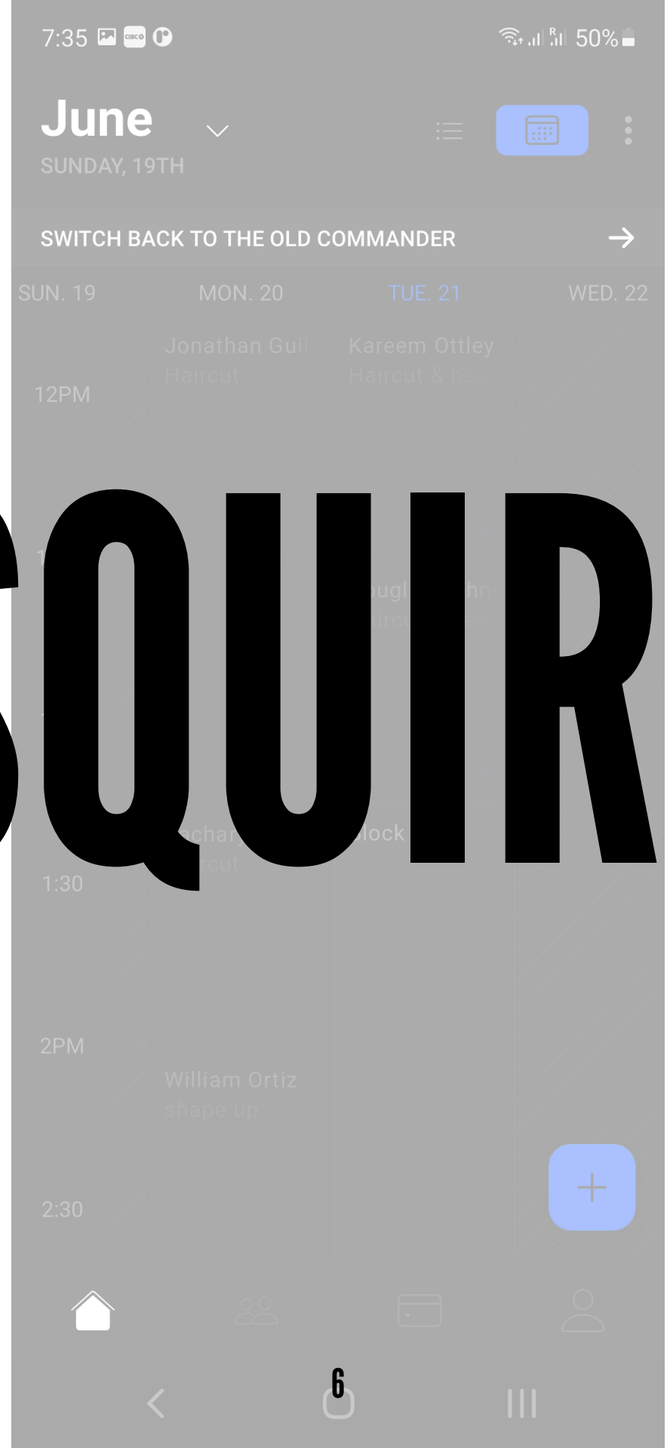
REAL APP APPROACH, 160 FRAGMENTS

Library + Demo Repo:
https://github.com/Gaket/GreenTea

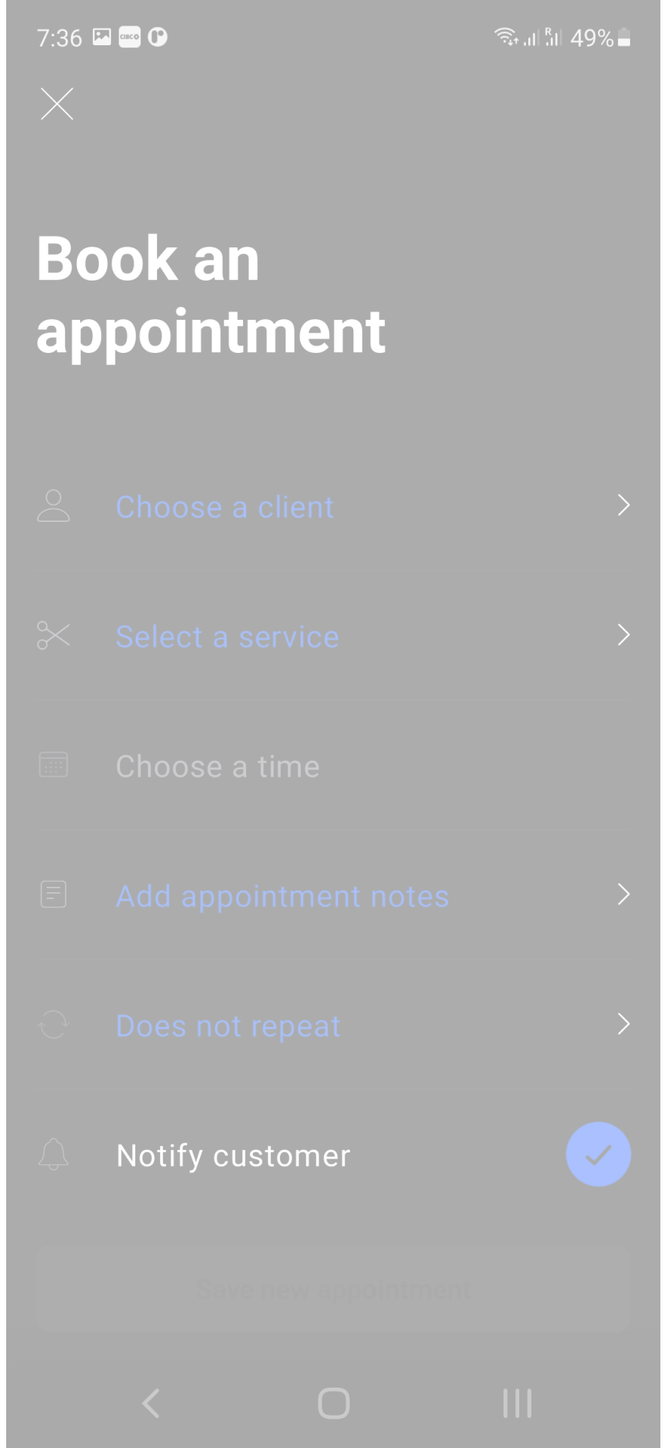
THE GOAL OF THIS TALK

- Not for absolute beginners
- Not about theory
- But about a real example


7:17
54%
June
TUESDAY, 21ST
SWITCH BACK TO THE OLD COMMANDER
Today
JUNE 21
7:30
8AM
8:30
9AM
9:30
Matthew Minihan
Haircut
10AM


7:35
50%
June
SUNDAY, 19TH
SWITCH BACK TO THE OLD COMMANDER
SUN. 19
MON. 20
TUE. 21
WED. 22
12PM
1:30
2PM
William Ortiz
shape up
2:30

SQUIRE


7:36
49%
Book an appointment
Choose a client
Select a service
Choose a time
Add appointment notes
Does not repeat
Notify customer

GREENFIELD PROJECT

WHAT DO YOU WANT FROM YOUR CODE?

- Easy to reason about
  - Cohesive
  - Testable
- Easy to find bugs

WHAT IS A BUG?

When a user sees something they aren't supposed to see or doesn't see something they are expected to see - it's a bug

When a user sees something they aren't supposed to see or doesn't see something they are expected to see - it's a bug

– Artur Badretdinov, while preparing this talk

HOW CAN WE SEE WHAT A USER SEES?

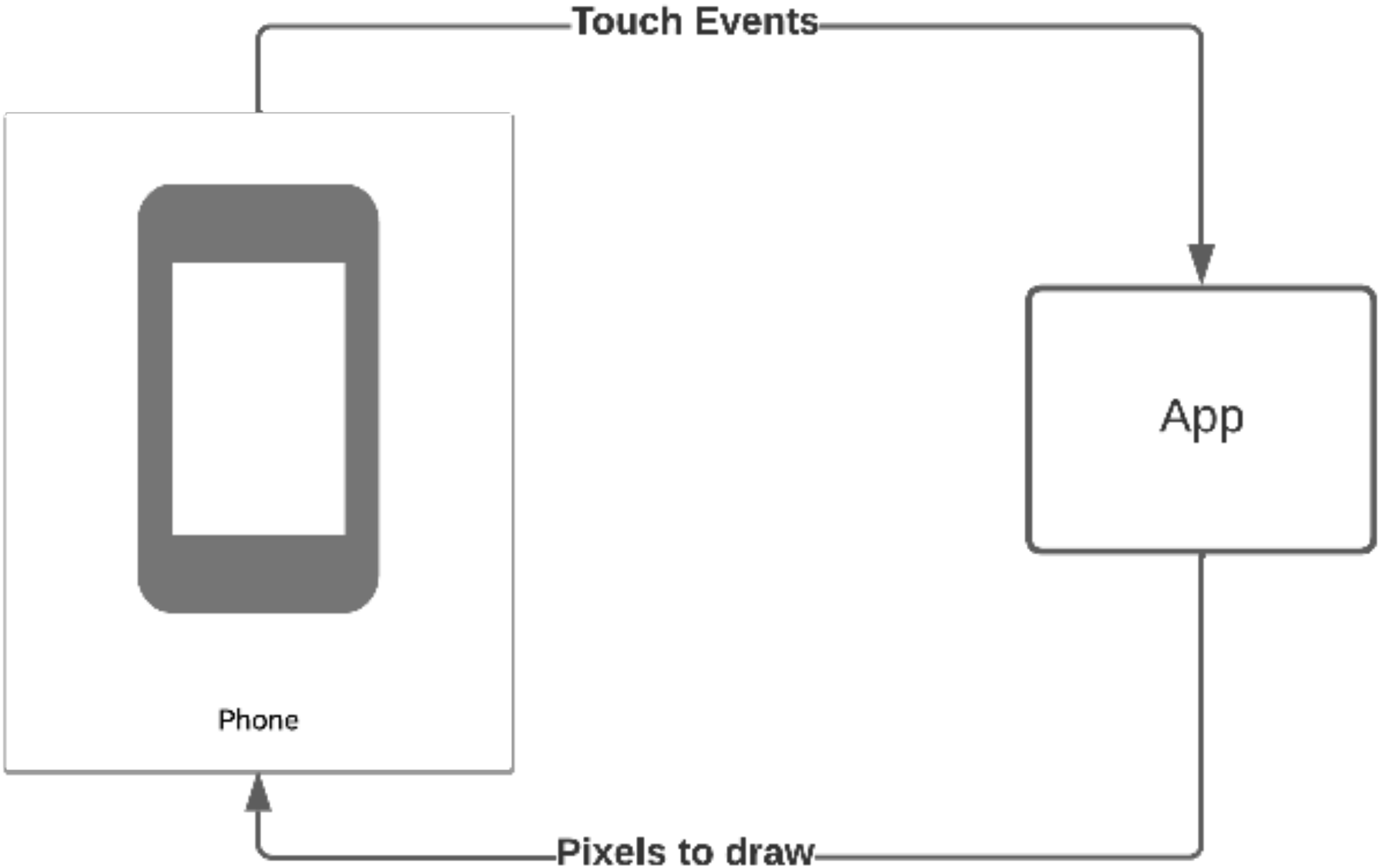
1. Look over their shoulder
2. Use session replay tools
3. Get the current screen state

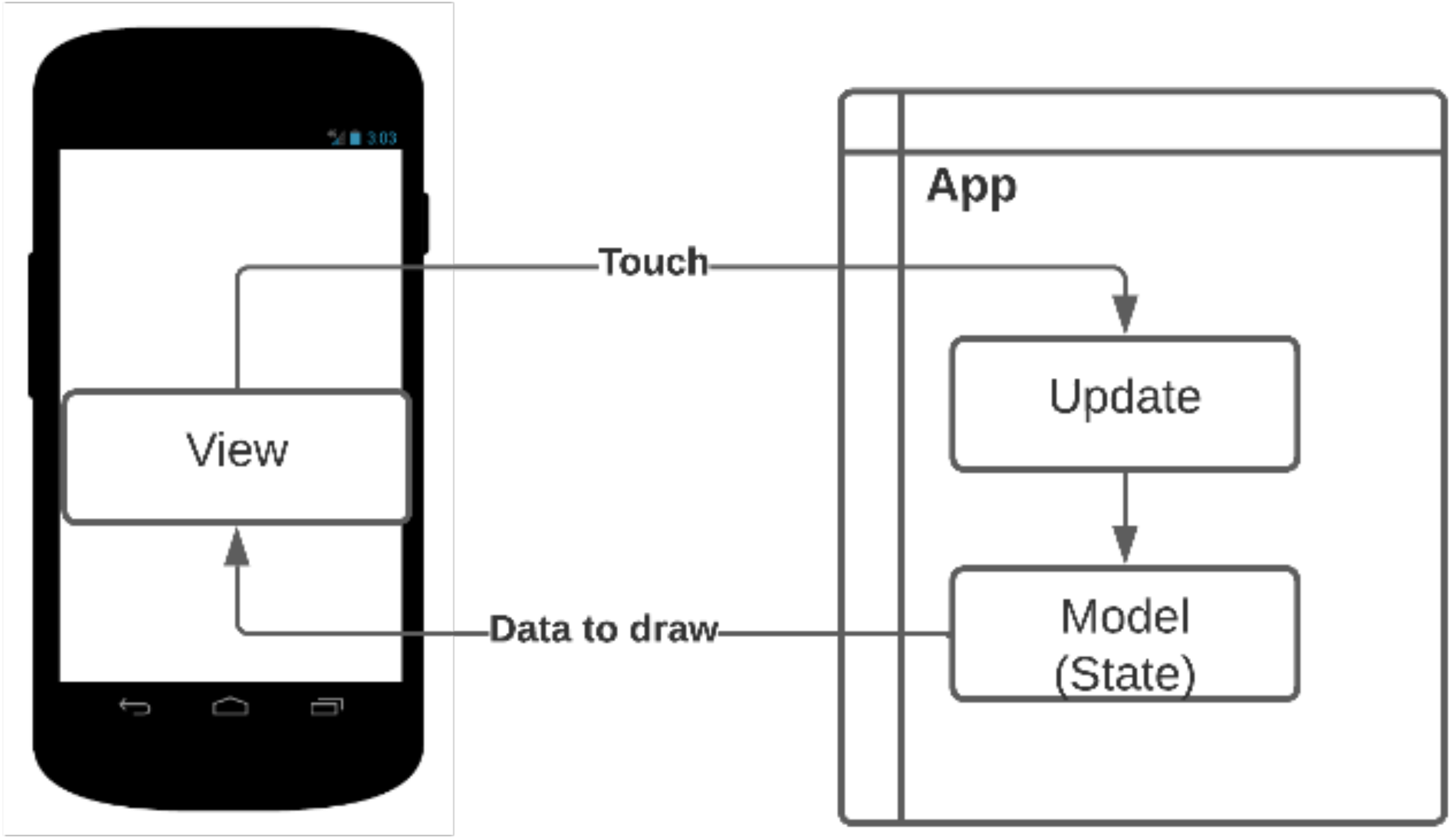
HOW TO GET THE CURRENT SCREEN STATE?

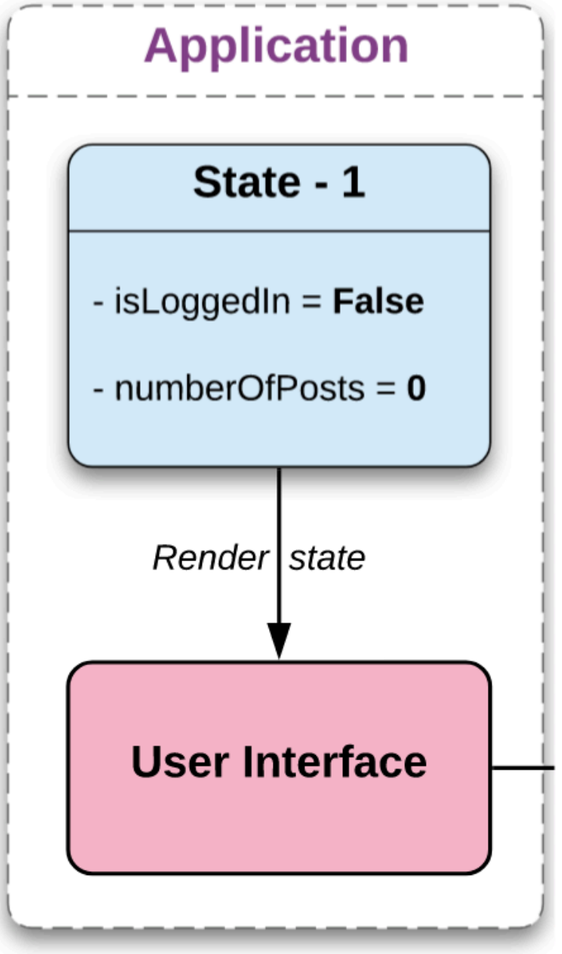
1. MVP - try to extract state from the Android.Views
2. MVVM - find all LiveData's / Flows that the View is subscribed to
3. UDF - have an explicit state object for the View

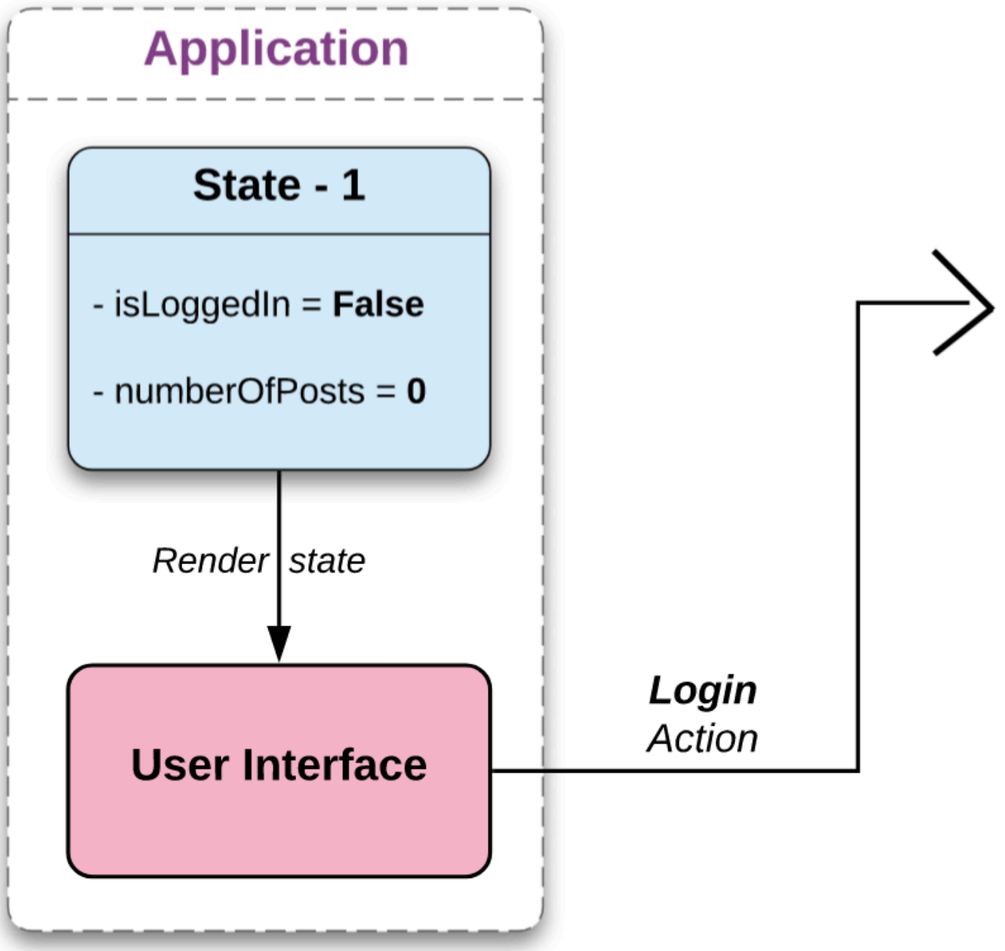
UDF?

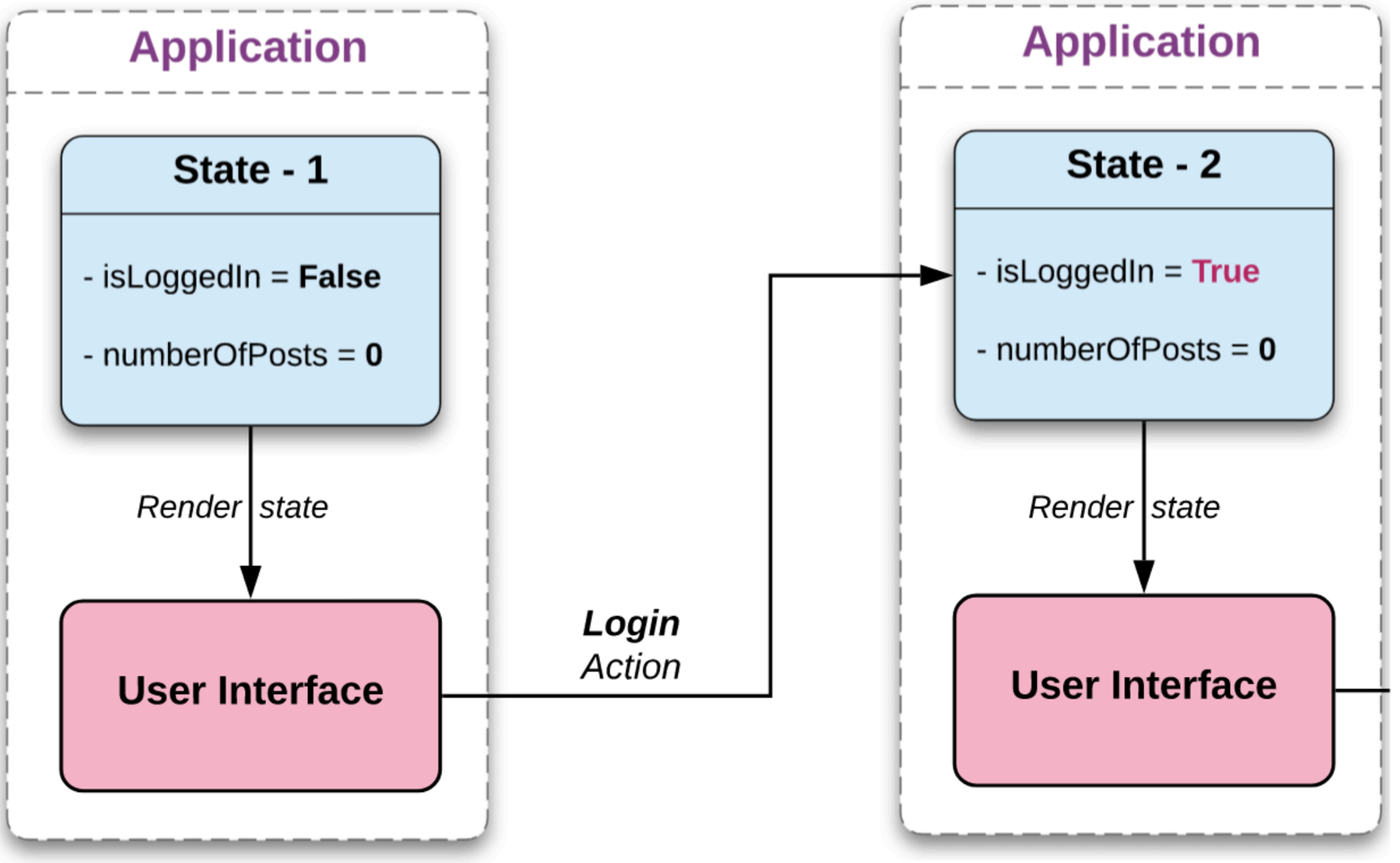
- Unidirectional Data Flow
- Back to at least 2014 with Facebook Flux
- Different patterns
  - Redux
  - MVI
  - MVU (TEA)
- Composable Architecture

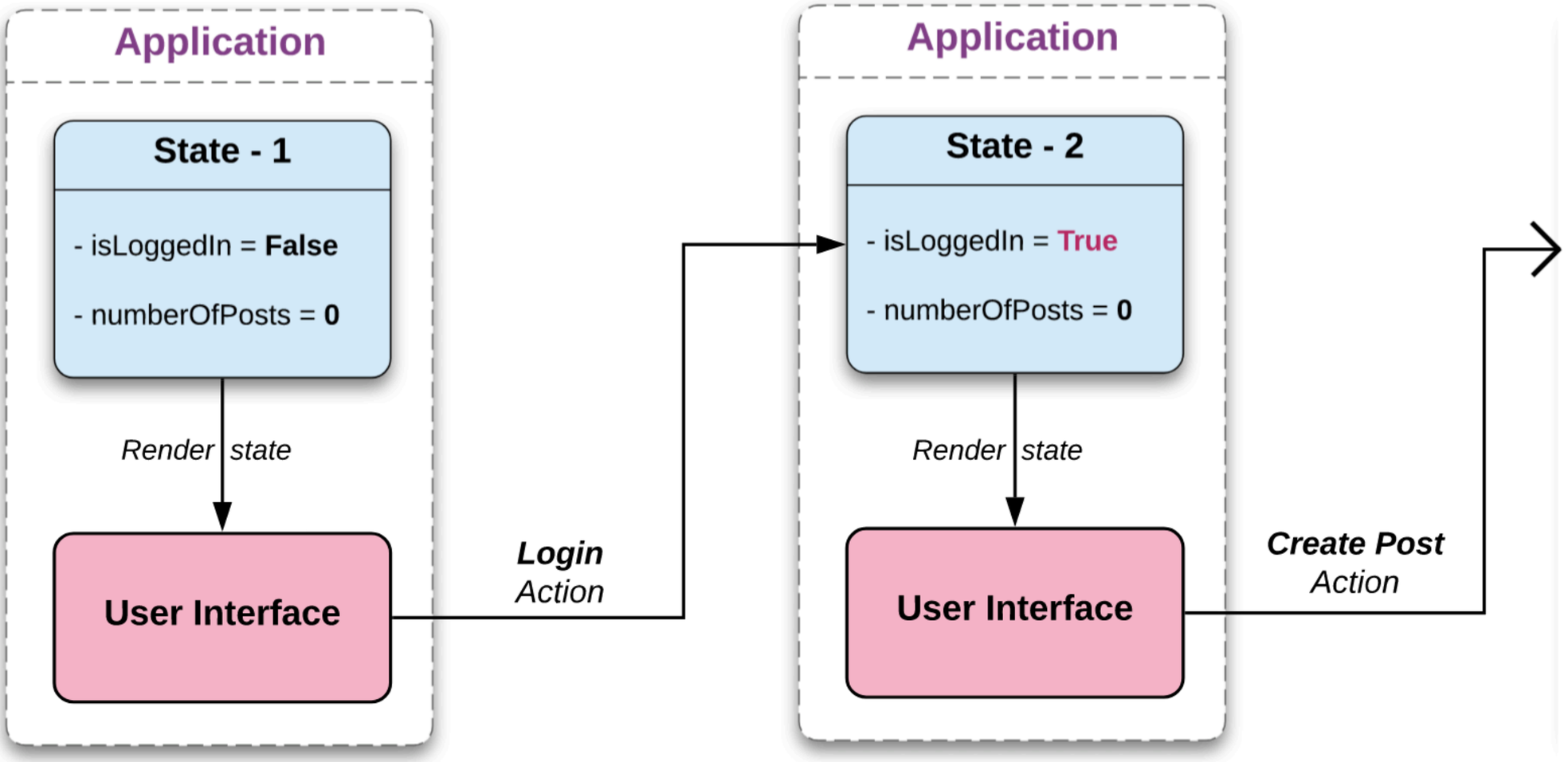

Touch Events
Phone
App
Pixels to draw

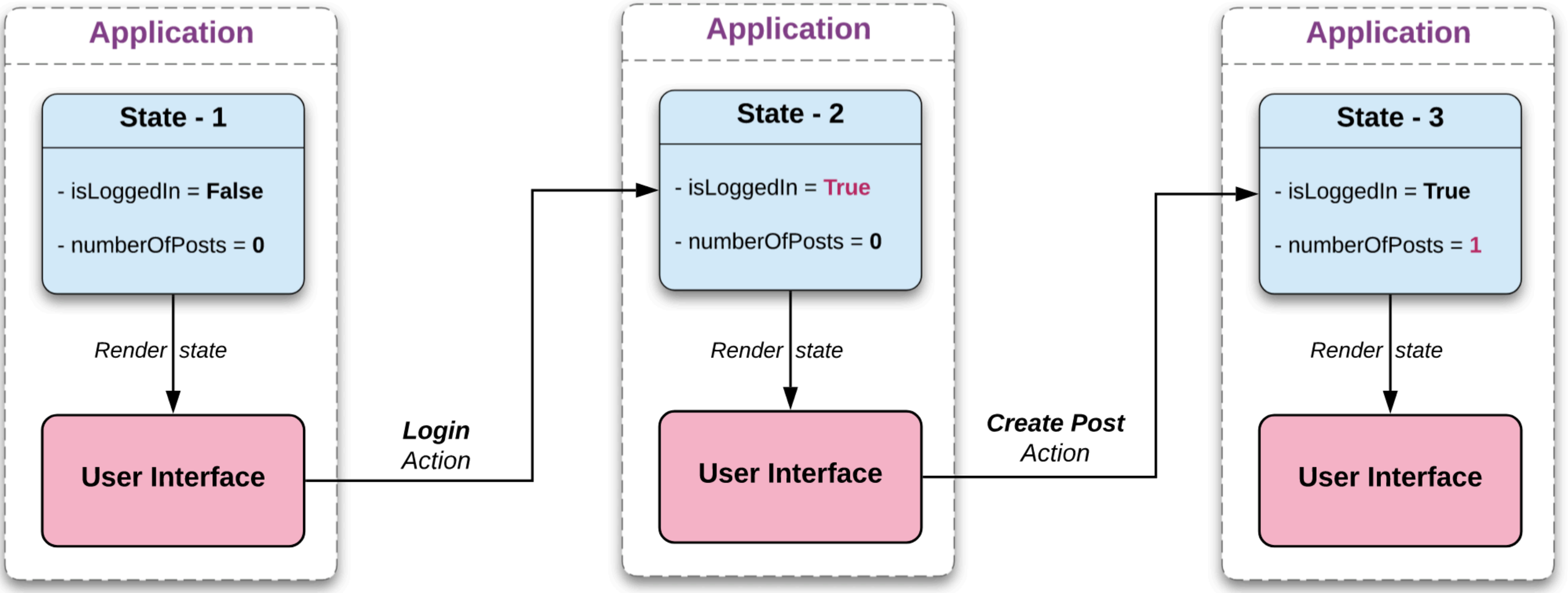

App
Touch
View
Update
Model
(State)
Data to draw


Application
State - 1
- isLoggedIn = False
- numberOfPosts = 0
Render state
User Interface

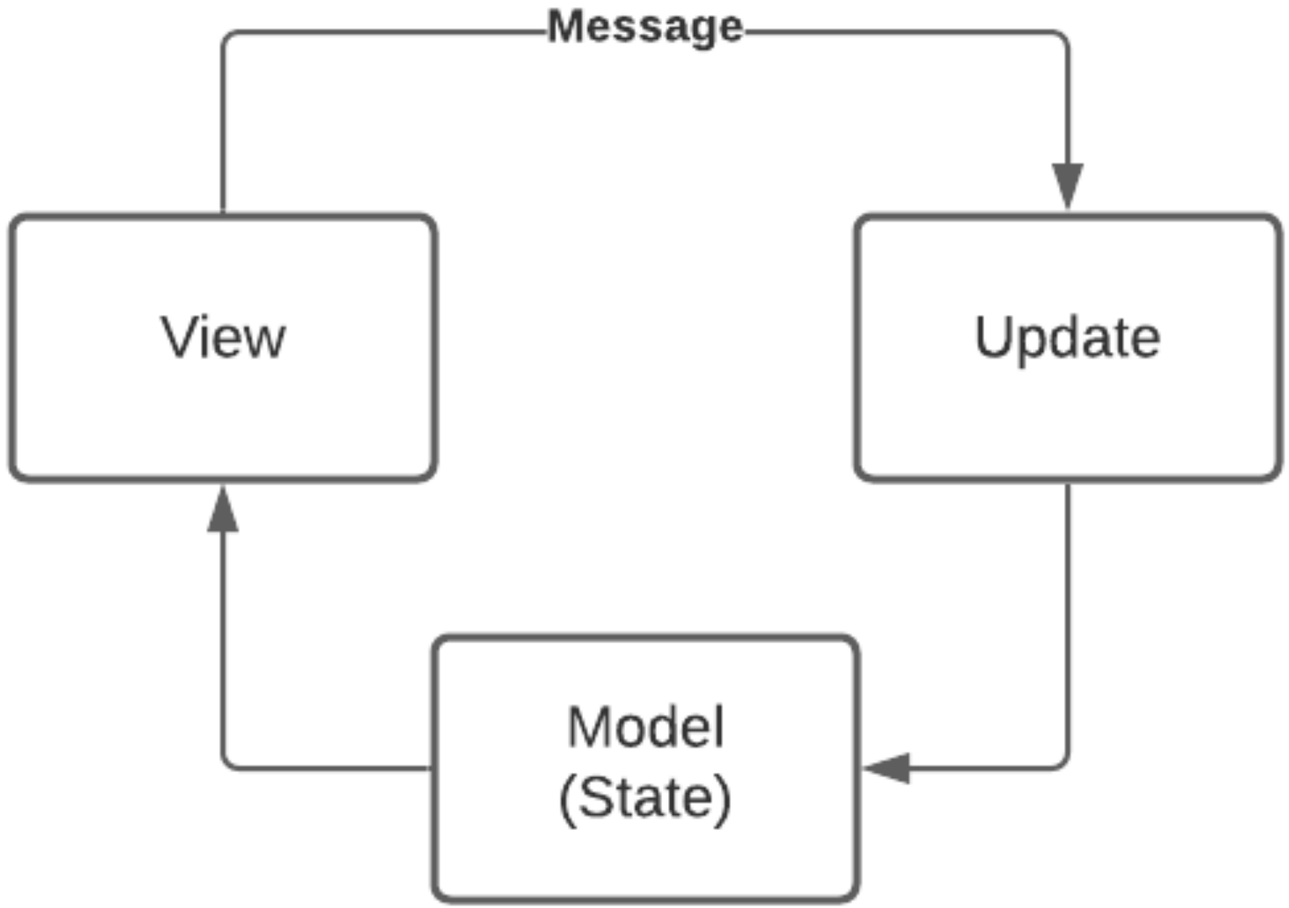

Application
State - 1
- isLoggedIn = False
- numberOfPosts = 0
Render state
User Interface
Login
Action


Application
State - 1
- isLoggedIn = False
- numberOfPosts = 0
Render state
User Interface
Login
Action
Application
State - 2
- isLoggedIn = True
- numberOfPosts = 0
Render state
User Interface


Application
State - 1
- isLoggedIn = False
- numberOfPosts = 0
Render state
User Interface
Login
Action
Application
State - 2
- isLoggedIn = True
- numberOfPosts = 0
Render state
User Interface
Create Post
Action


Application
State - 1
- isLoggedIn = False
- numberOfPosts = 0
Render state
User Interface
Login
Action
Application
State - 2
- isLoggedIn = True
- numberOfPosts = 0
Render state
User Interface
Create Post
Action
Application
State - 3
- isLoggedIn = True
- numberOfPosts = 1
Render state
User Interface

UNIDIRECTIONAL FLOW (UDF)



REALLY QUICK RECAP

- Pure functions
- Immutability
- Higher order functions
- State machine
- Side effects

VIEW = F(STATE)

UPDATED STATE = F(STATE, MESSAGE)

MVI

MVVM with additional constraints:

1. Single state
2. Stricter rules prohibiting logic in View classes
3. Usually, based on reactive streams

OUR MVI EXPERIENCE

- RIBs and MVICore
- 6 months of development
- Great separation of concerns
- A lot of features
- A LOT of code
- Mocks and RxJava in tests

MVU (TEA)

- MVI with explicit side effects
- MVI with hidden event flows

Elm is a functional language for web apps

ONE WAY TO IMPLEMENT MODEL

ONE WAY TO IMPLEMENT ~~MODEL~~ FEATURE

ONE WAY TO IMPLEMENT FEATURE

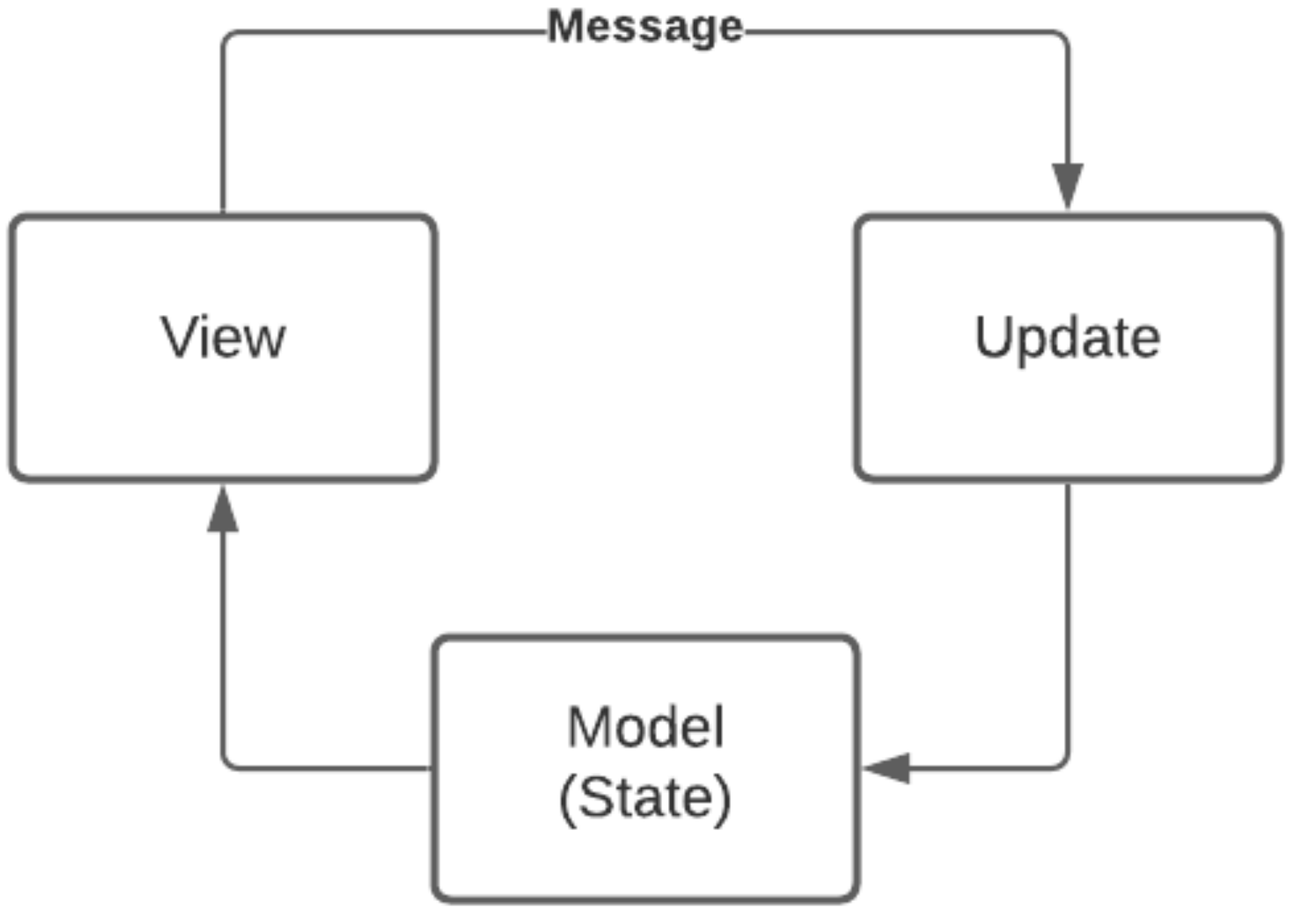
- State
- Messages
- Update
- Effects
- Dependencies

FIRST APPROACH, MVICORE

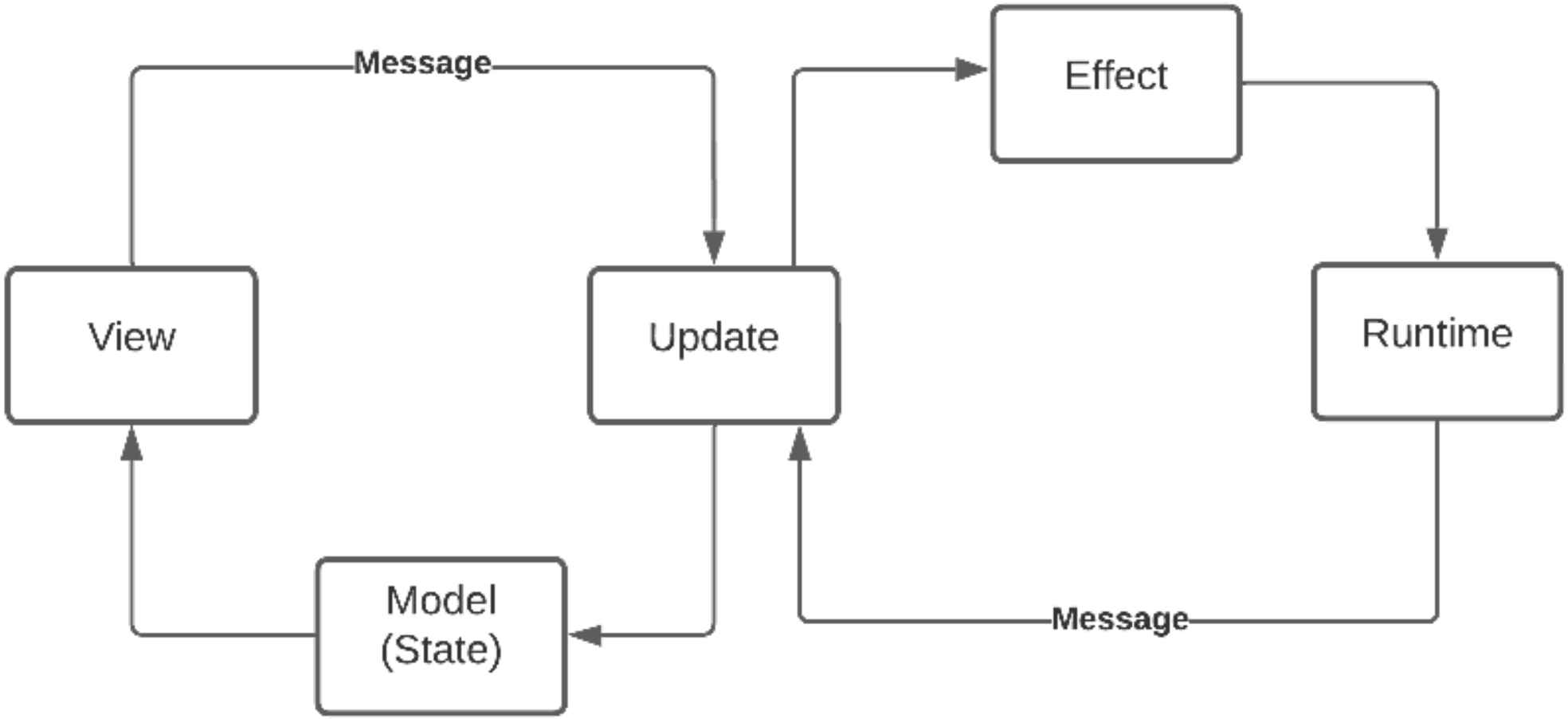
1. Feature
    1. State
    2. Wish
    3. Effect
    4. News
5. Bootstrapper
    6. Actor
    7. Reducer
8. Post Processor

1. View
    1. ViewModel
    2. ViewEvent
    2. Binder
    3. Middleware
    4. Mappers
        1. StateToViewModel
        2. ViewEventToWish
    3. ViewEventToAnalytics

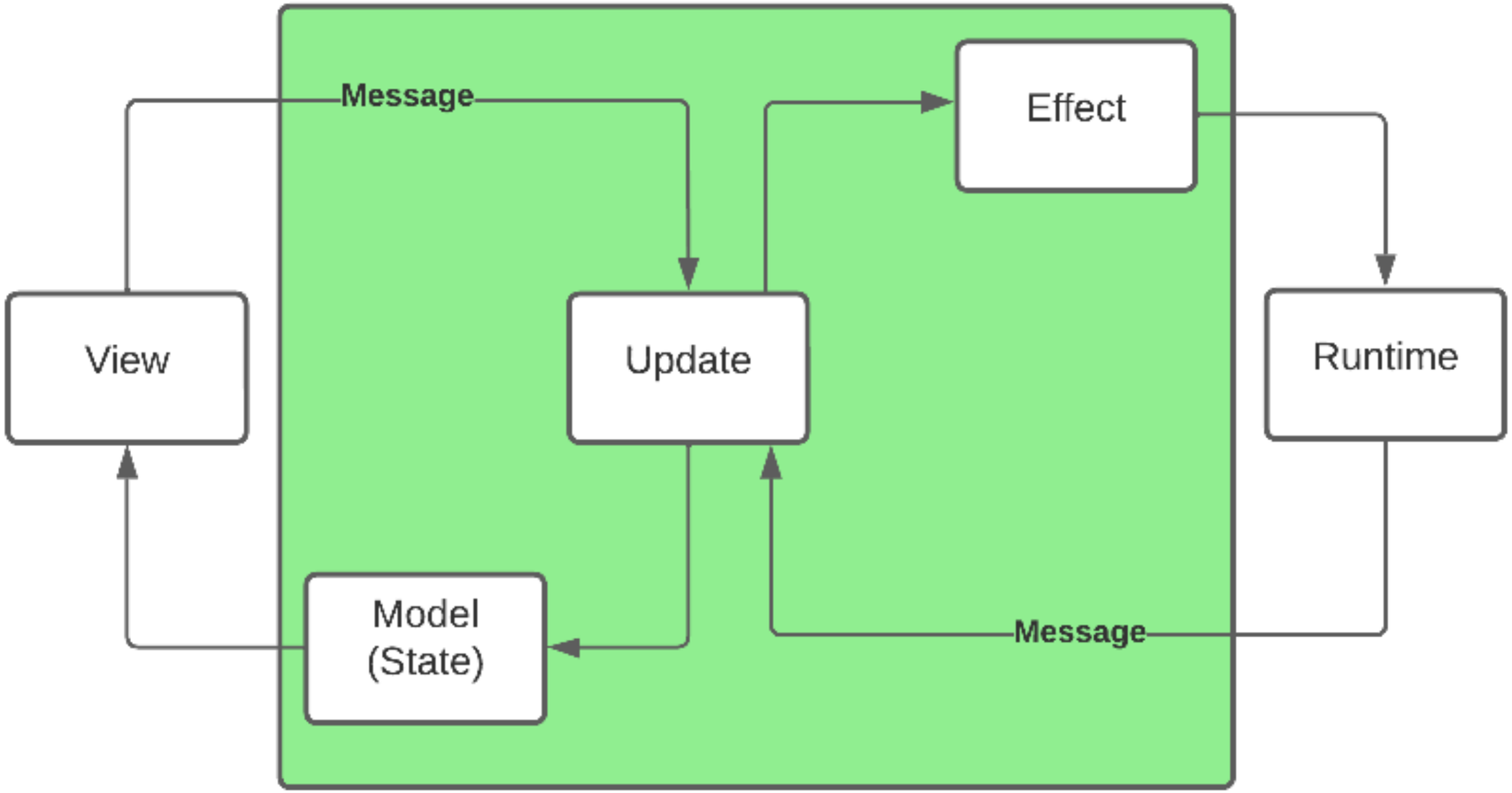
MVU (MODEL - VIEW - UPDATE)

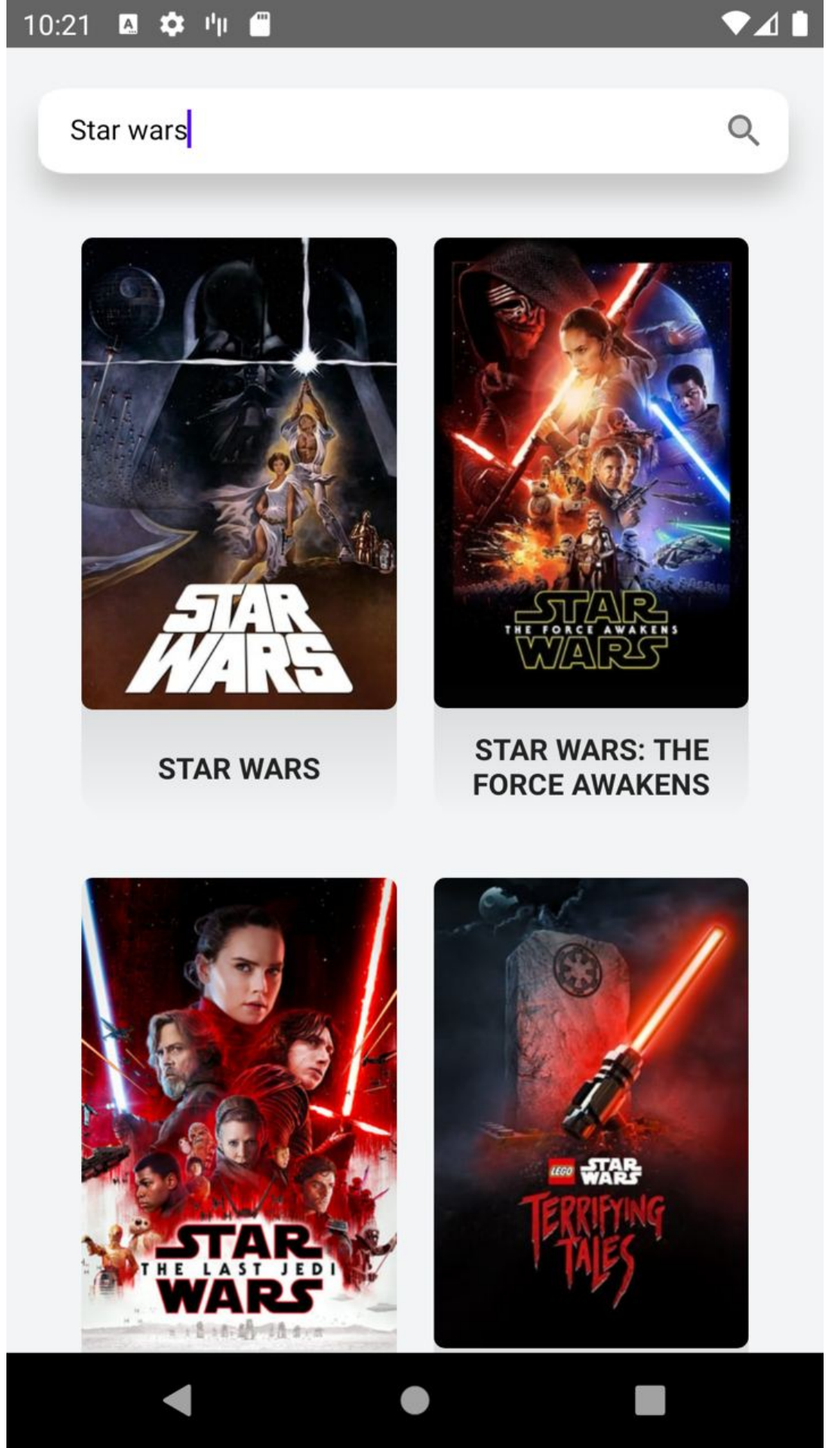


REAL-WORLD MVU



REAL-WORLD MVU





STATE

- Contains everything needed to render the View
- Never updated manually

```
data class State(
  val loading: Boolean,
  val movies: List<Movie>,
  val message: Text,
)
```

MESSAGES

Whatever can lead to state change

```
sealed class Message {
  // user
  data class SearchUpdated(val query: String, val time: LocalTime) : Message()
  data class MovieClicked(val movie: Movie) : Message()

  // system
  data class MoviesResponse(val response: Try<List<Movie>>) : Message()
}
```

UPDATE

Pure boy
"Old state in, new state out"

```
fun update(message: Message, state: State)
                    : Pair<State, Set<Effect>> =
    when (message) {
        is Message.MovieClicked -> handleMovieClick(message.movie, state)
        is Message.SearchUpdated -> handleSearchUpdate(message.query, state)
        is Message.MoviesResponse -> handleMoviesResponse(message.response, state)
      }
```

Keep View stupid!

– MVP

Keep View and Side Effects stupid!

– MVU

EFFECTS

- DB
- Network
- Random

```
class GetMovies(query: String) : Message ({ deps ->
    val movies = deps.repository.searchMovies(query)
    return@single Message.MoviesResponse(movies)
})
```

LIVE DEMO

REAL APP APPROACH, 160 FRAGMENTS

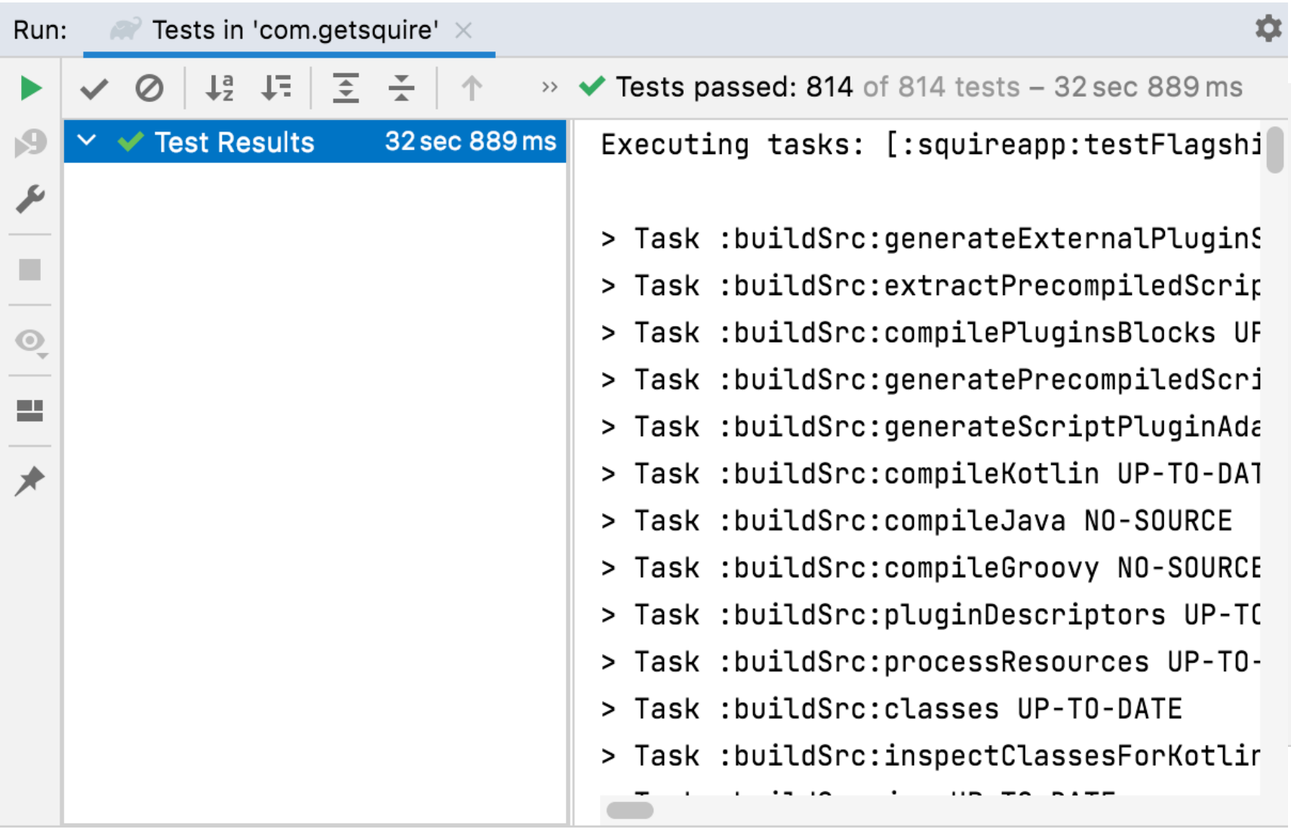
Demo Repo:
https://github.com/Gaket/GreenTea

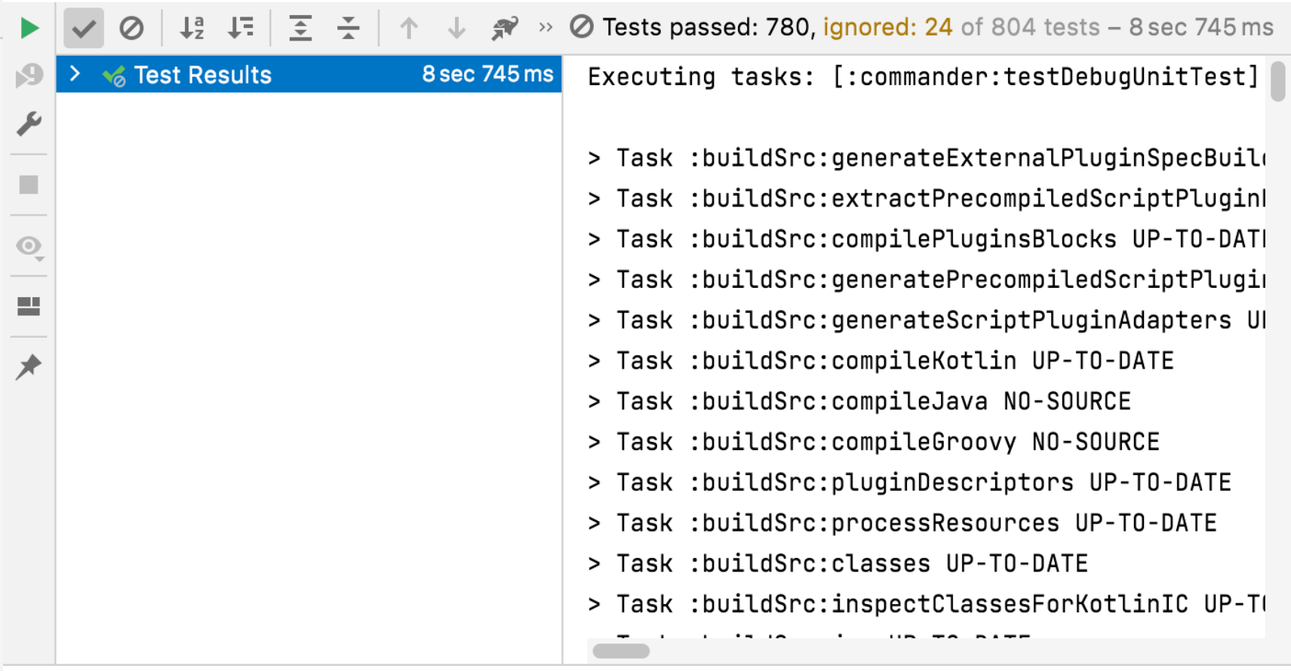
HOW TO DEBUG

1. Check what state gets to View
2. Check the `update` function calls
3. Roll back a few messages if needed

LOGS

```
18:00:00 Init: State {...}
18:00:03 Message: OnMovieClick(Movie(...))
18:00:03 Render: State {...}
18:00:04 Message: OnMoviesResult(Movies(...))
18:00:04 Render: State {...}
```


Run:
Tests in 'com.getsquire'
Tests passed: 814 of 814 tests – 32 sec 889 ms
Test Results 32 sec 889 ms
Executing tasks: [:squireapp:testFlagshi
> Task :buildSrc:generateExternalPluginS
> Task :buildSrc:extractPrecompiledScrip
> Task :buildSrc:compilePluginsBlocks UP
> Task :buildSrc:generatePrecompiledScri
> Task :buildSrc:generateScriptPluginAda
> Task :buildSrc:compileKotlin UP-TO-DAT
> Task :buildSrc:compileJava NO-SOURCE
> Task :buildSrc:compileGroovy NO-SOURCE
> Task :buildSrc:pluginDescriptors UP-TO
> Task :buildSrc:processResources UP-TO-
> Task :buildSrc:classes UP-TO-DATE
> Task :buildSrc:inspectClassesForKotlin

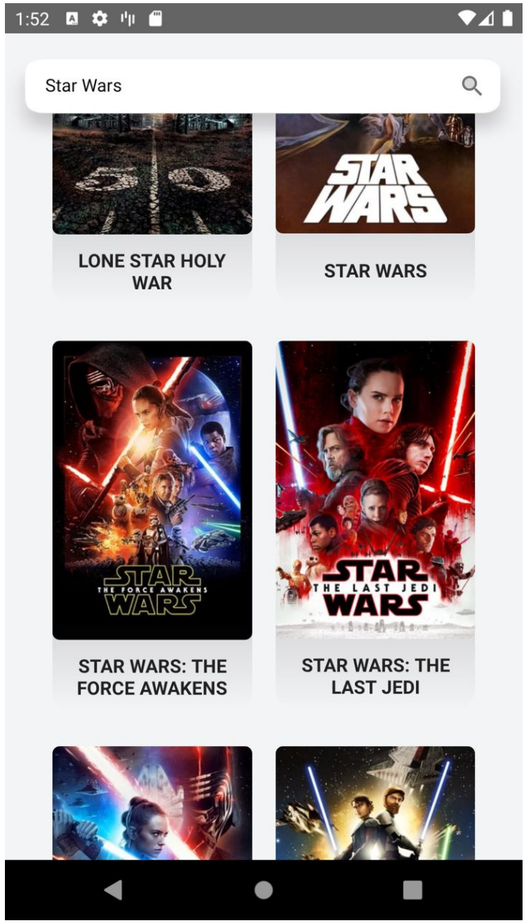

Tests passed: 780, ignored: 24 of 804 tests – 8 sec 745 ms
Test Results 8 sec 745 ms
Executing tasks: [:commander:testDebugUnitTest]
> Task :buildSrc:generateExternalPluginSpecBuild
> Task :buildSrc:extractPrecompiledScriptPluginI
> Task :buildSrc:compilePluginsBlocks UP-TO-DATE
> Task :buildSrc:generatePrecompiledScriptPlugin
> Task :buildSrc:generateScriptPluginAdapters UP
> Task :buildSrc:compileKotlin UP-TO-DATE
> Task :buildSrc:compileJava NO-SOURCE
> Task :buildSrc:compileGroovy NO-SOURCE
> Task :buildSrc:pluginDescriptors UP-TO-DATE
> Task :buildSrc:processResources UP-TO-DATE
> Task :buildSrc:classes UP-TO-DATE
> Task :buildSrc:inspectClassesForKotlinIC UP-TO

OTHER CASES

- One-time events
- Navigation
- Separate classes for UI models
- Compose for View

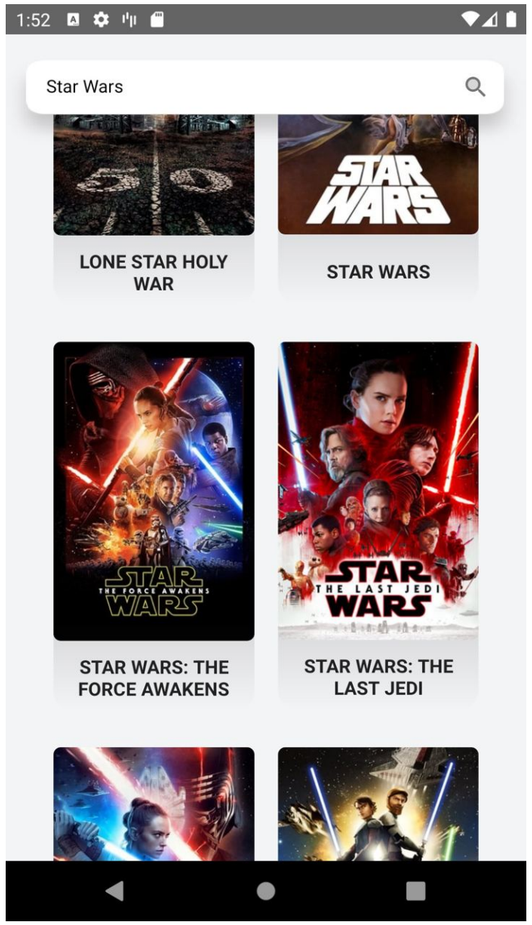
IMPERFECT WORLD

```
data class State(
  val loading: Boolean,
  val movies: List<Movie>,
  val message: Text,
)
```



IMPERFECT WORLD

```
data class State(
  val loading: Boolean,
  val movies: List<Movie>,
  val message: Text,
)
```



- View is expected to be stateless
- We still have scroll position, animations, Toasts

CONCLUSION

Not a silver bullet, but worth trying!

- Clear responsibilities
- Increased testability
- Ability to run integration tests with backend but without UI
- No flakiness
- Easy to debug / reason about - one input, one output
- Perks like Time Travel / State Restore (exist in MVI too)

MAIN DRAWBACKS

- A bit more code
- Action handling code is split into different chunks
- Learning curve for new devs

ACKNOWLEDGEMENTS

- Tim Plotnikov
- Sergey Opivalov
- Evgeniy Ekgardt
- Mikhail Gurevich
- Nicolas Mottin

QUESTIONS?

Artur Badretdinov @ Squire

Repo:
https://github.com/Gaket/GreenTea

https://twitter.com/ArtursTwit
https://t.me/gaket

If we had more time... Testing!

What matters the most?

1. App's data and logic
2. Elements on Screens
3. Data on Screens
4. Navigation between Screens

User experience!

Don't fix bugs later; fix them now

– Steve Maguire

DOUBLE-ENTRY BOOKKEEPING

TESTING MVP

```
presenter.onAction()
verify(testView.doA())
verify(testView.doB())
verify(someRepo.wasCalled())
```

TESTING MVVM

```
///
/// a lot of prep code to test LiveData / RxJava / Coroutines
///
viewModel.onAction()
assertThat(firstLiveData.value).isTrue
assertThat(secondLiveData.value).isNull
verify(someRepo.wasCalled())
```

TESTING MVU

```
val (viewState, effects) = feature.update(prevState, message)
assertThat(viewState.isLoading).isTrue
assertThat(viewState.error).isNull
assertThat(effects).contain()
```

Backup Slide:

VIEW

```
class MoviesFragment {
 override fun initDispatchers() {
    binding.searchInput.afterTextChanged { query ->
      dispatch(MoviesFeature.Message.SearchUpdated(query, LocalTime.now()))
    }
  }
   override fun render(state: MoviesFeature.State) {
    if (state.loading) {
      binding.searchIcon.visibility = View.GONE
      binding.searchProgress.visibility = View.VISIBLE
    } else {
      binding.searchIcon.visibility = View.VISIBLE
      binding.searchProgress.visibility = View.GONE
    }
   }
}
```