TEA in Prod

Artur Badretdinov, Squire

Artur Badretdinov

Digital Nomad, Lead Android Developer at Squire

https://t.me/travelernote

https://t.me/ohmyeventbot

The Elm Architecture in prod

Live demo ahead
Real app approach, 160 Fragments

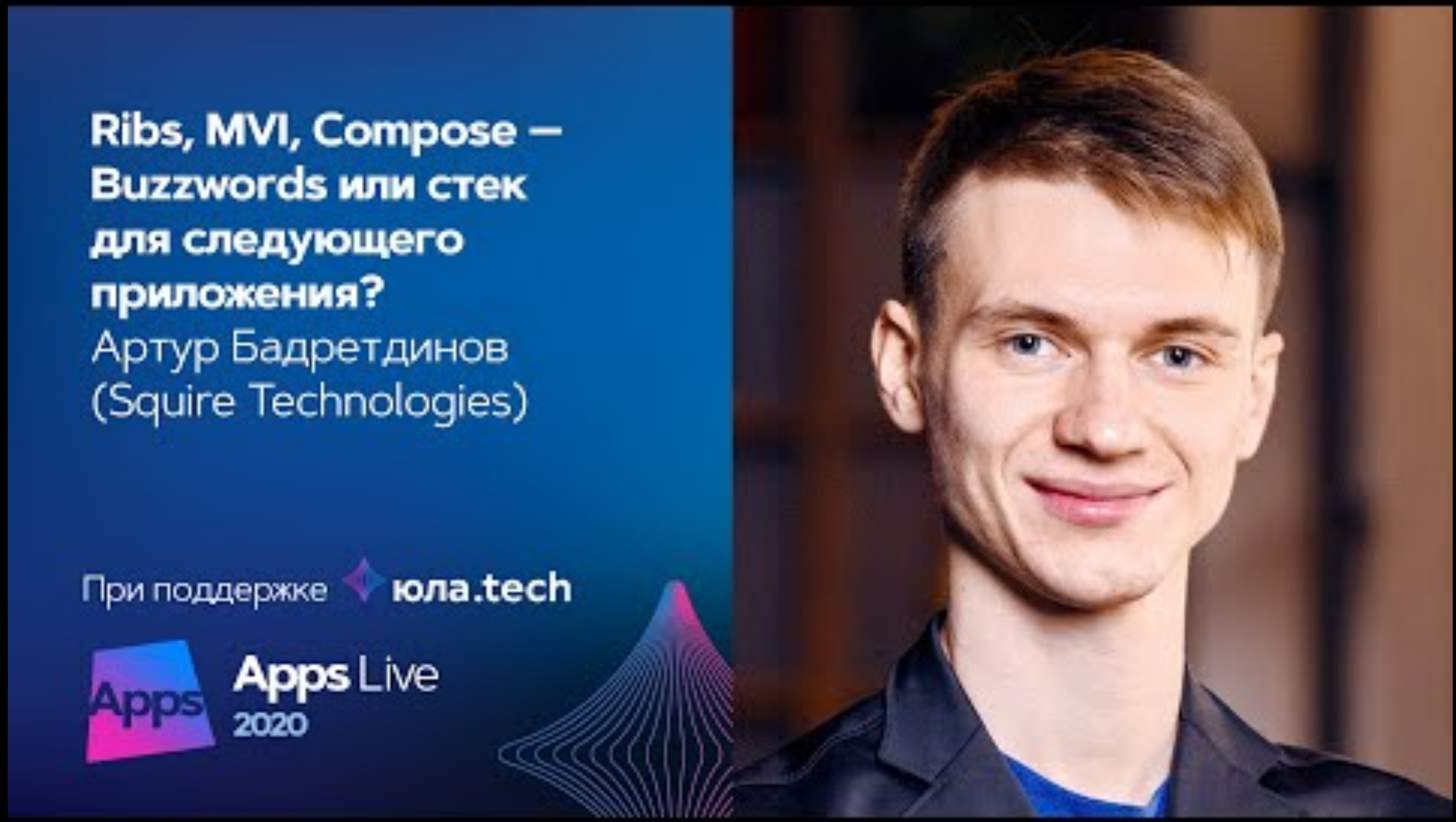
Demo Repo:

https://github.com/Gaket/MvuMovieDb

The goal of this talk

» Not about theory

» But about a real example


Ribs, MVI, Compose —
Buzzwords или стек
для следующего
приложения?
Артур Бадретдинов
(Squire Technologies)
При поддержке
юла.tech
Apps
Apps Live
2020

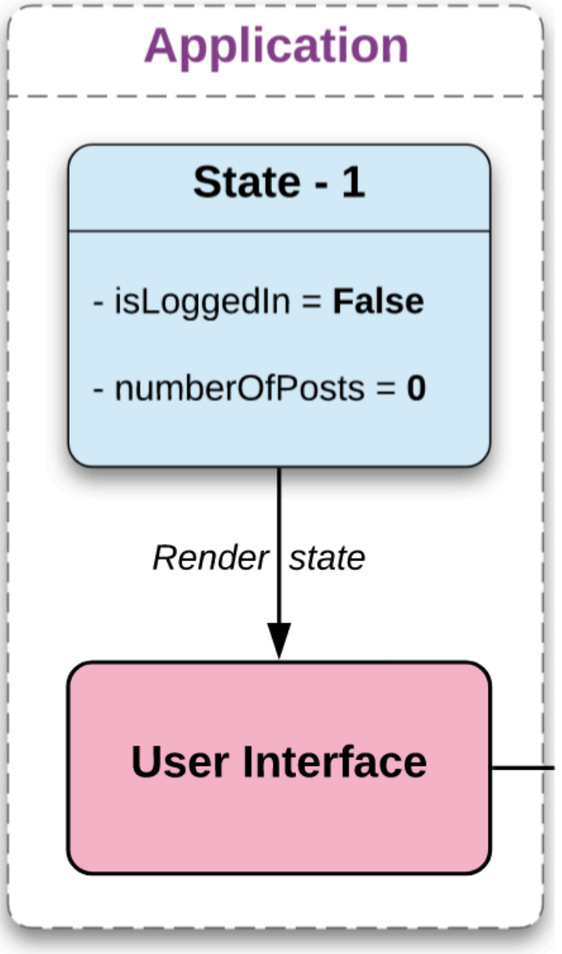
*Elm is a **functional** language for web apps*

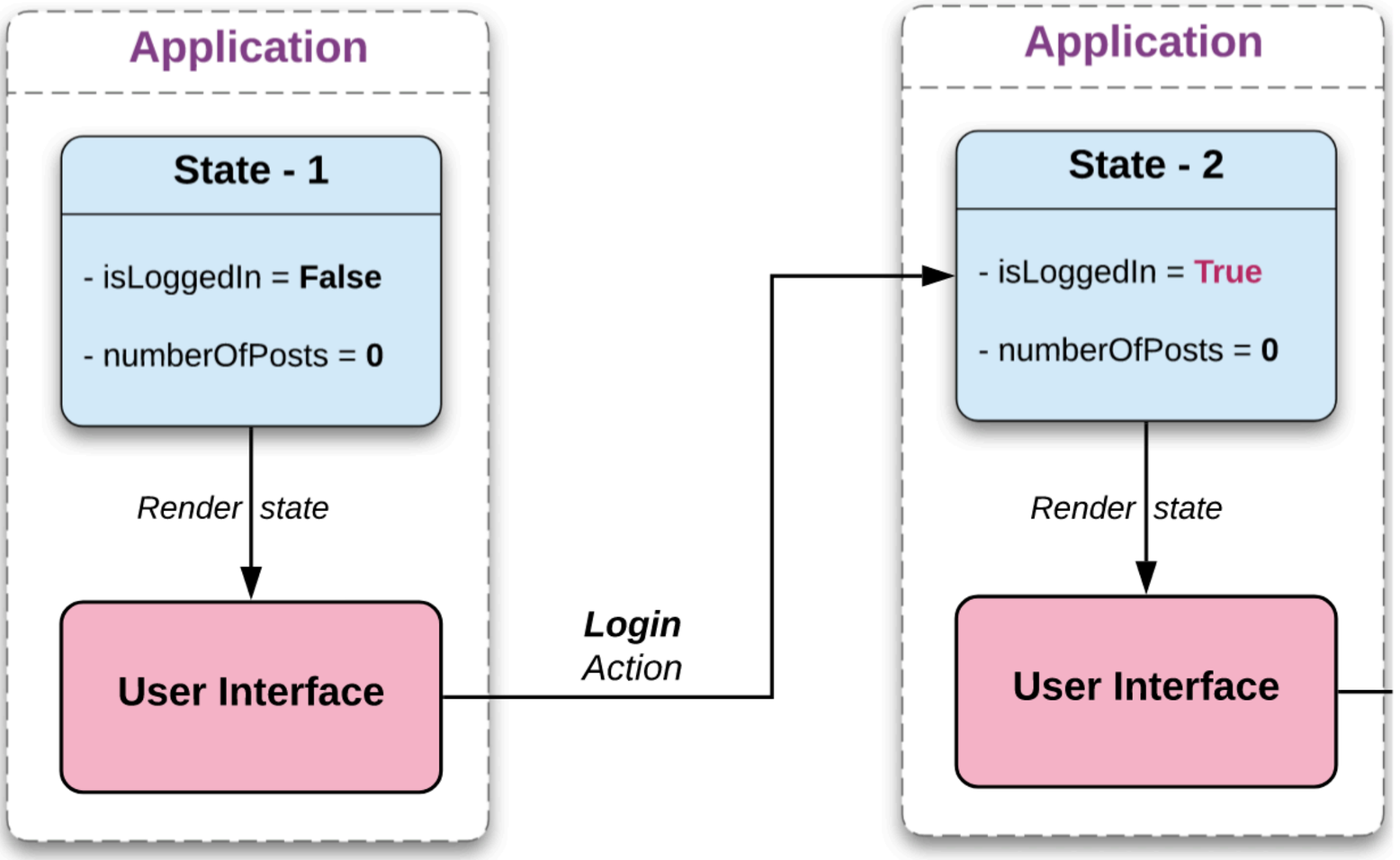
Elm Architecture is one of the UDF band

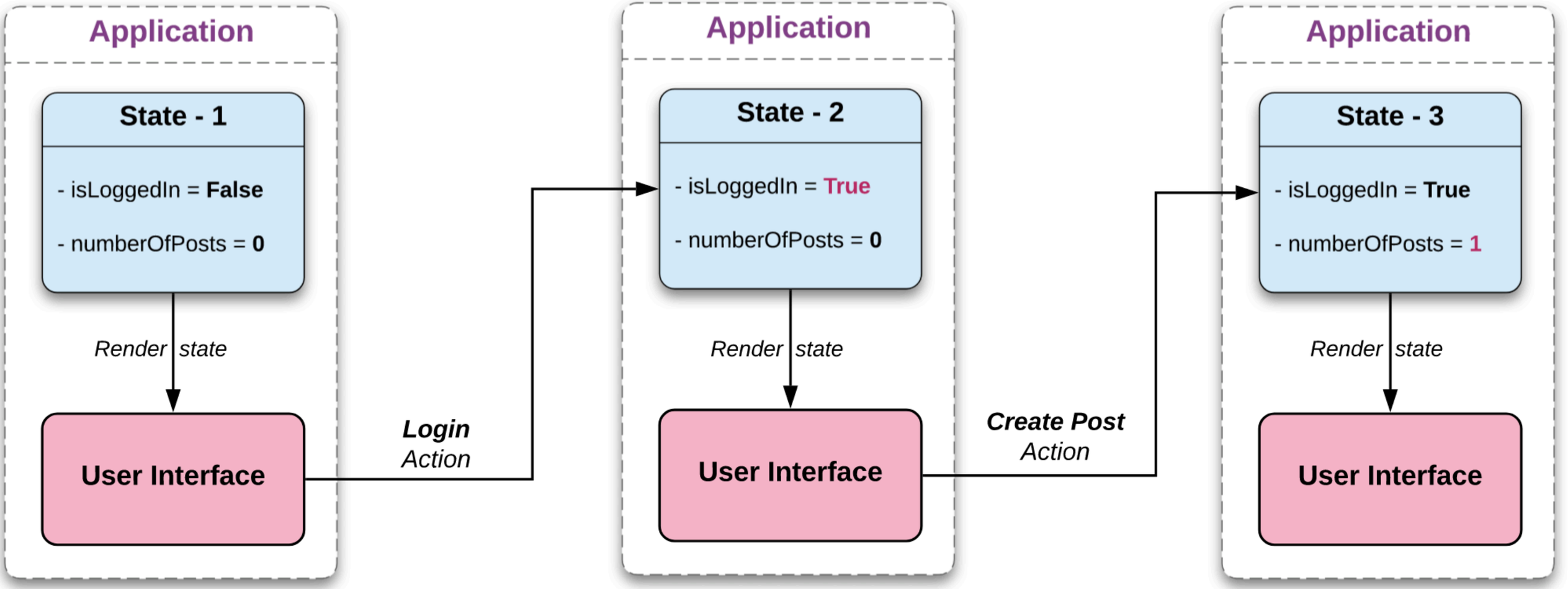
Really Quick Recap [1]

- Unidirectional data flow
- Pure functions
- Immutability
- Higher order functions
- State machine
- MVI
- TEA (MVU)
- Redux
- The Composable Architecture

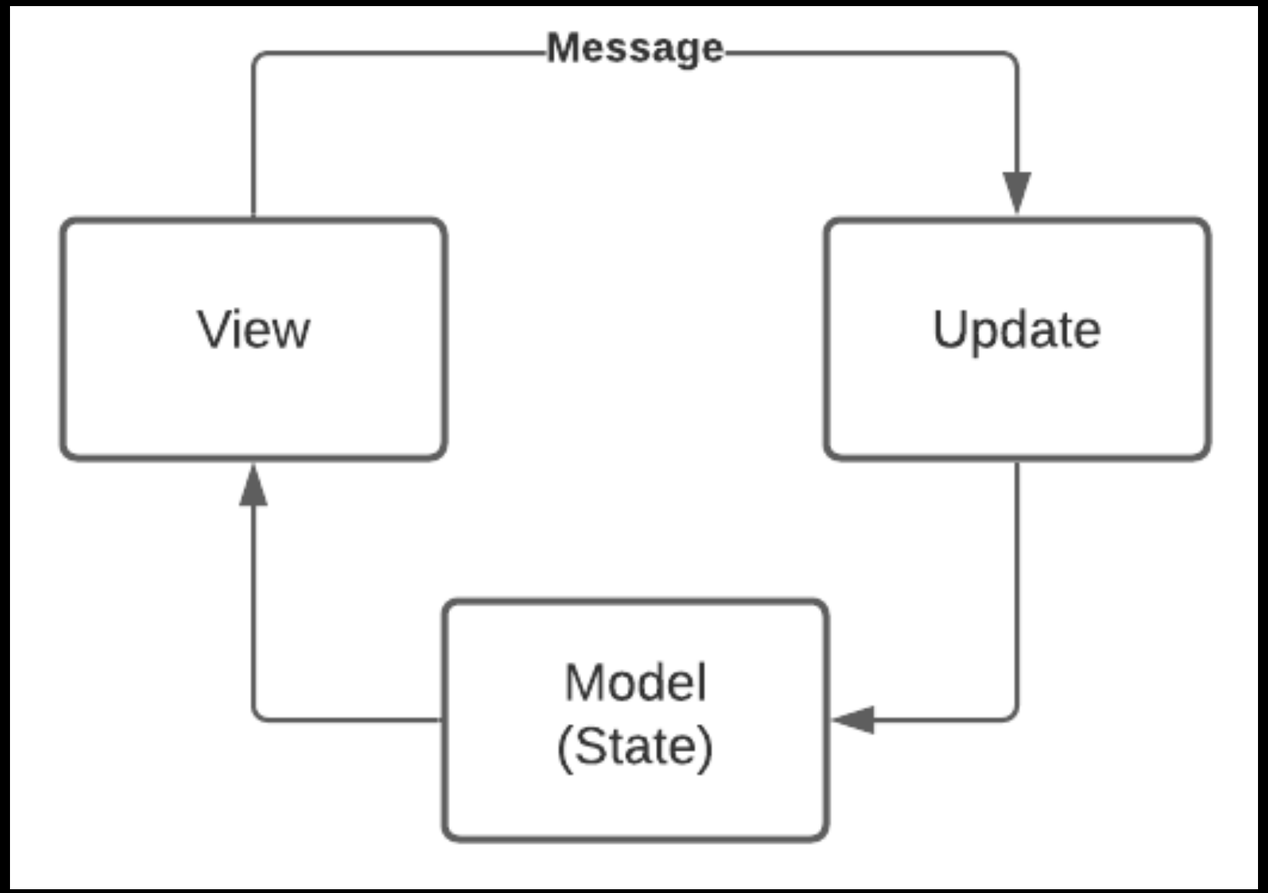
[1] https://www.youtube.com/watch?v=FPbUAgyMPAc


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 - 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 - 3
- isLoggedIn = True
- numberOfPosts = 1
Render state
User Interface

Unidirectional Flow (UDF)



view = f(state)

updated state = f(state, message)

What do we want?

- Explicit state
- Making state handling visible
- Clear responsibilities
- Increased testability
- No mocks, No flakiness
- Easy to reason about

Explicit state requirements

- Same way as no platform dependencies
- No side effects in Update
- Otherwise, we can't guarantee repeatability

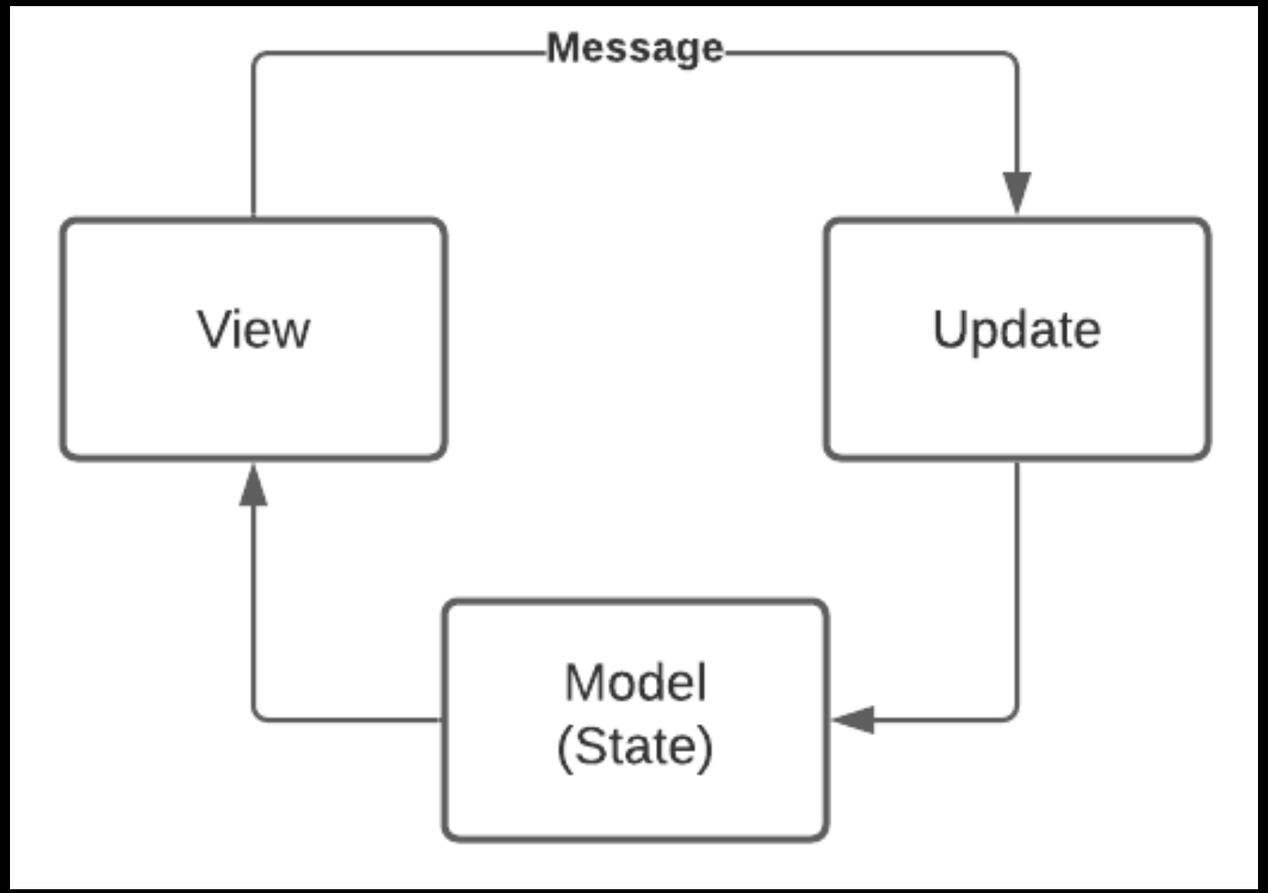
One way to implement Model

One way to implement ~~Model~~ Feature

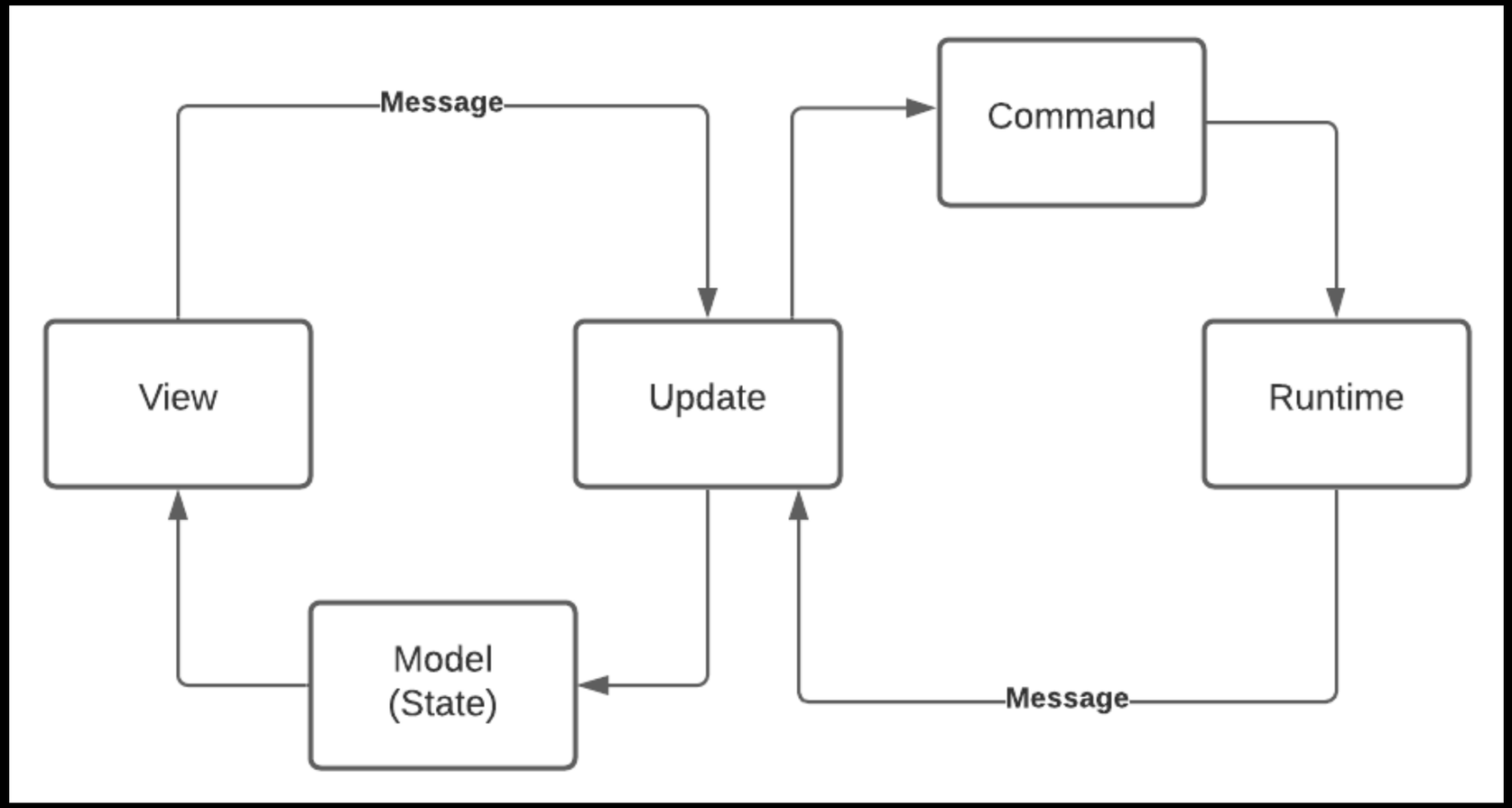
One way to implement Feature

- State
- Messages
- Update
- Commands
- Dependencies

Unidirectional Flow (UDF)



Real Unidirectional Flow (UDF)



State

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

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

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<Command>> =
    when (message) {
        is Message.MovieClicked -> handleMovieClick(message.movie, state)
        is Message.SearchUpdated -> handleSearchUpdate(message.query, state)
        is Message.MoviesResponse -> handleMoviesResponse(message.response, state)
      }
```

Commands

We live in a real world

* DB
* Network
* Random

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

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
    }
   }
}
```

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 {...}

Live demo
Real app approach, 160 Fragments

Demo Repo:

https://github.com/Gaket/MvuMovieDb

Other cases

- One-time events
- Navigation
- Seperate classes for UI models

Imperfect world

» View is expected to be stateless

» Still we have scroll position, animations

Conclusion

Not a silver bullet, but worth trying!

» Clear responsibilities

» Increased testability

» No flakiness

» Easy to reason about

» A bit more code

Acknowledgements

- Tim Plotnikov, https://twitter.com/timofeipl
- Sergey Opivalov
- Evgeniy Ekgardt

Questions?

Artur Badretdinov @ Squire

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

https://twitter.com/ArtursTwit

https://t.me/gaket