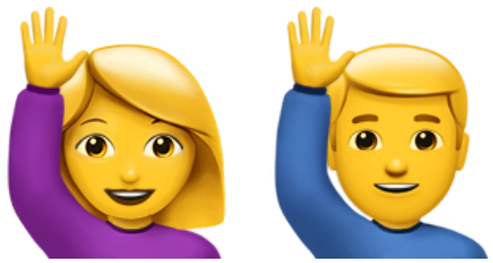
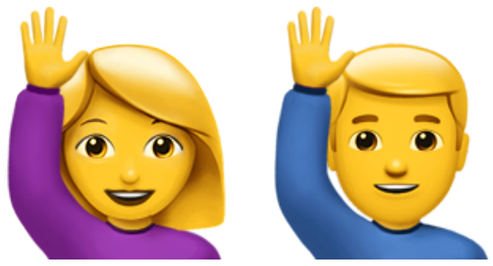
Flux for Android

@shaunkawano

CyberAgent

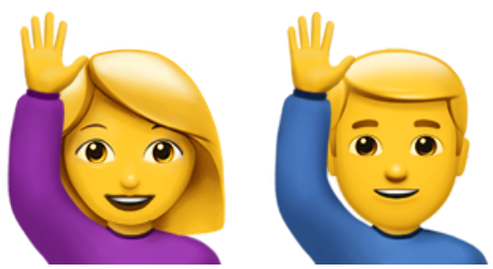




Web Android

Flux for

?



Flux for Android ?

What is Flux?

Flux

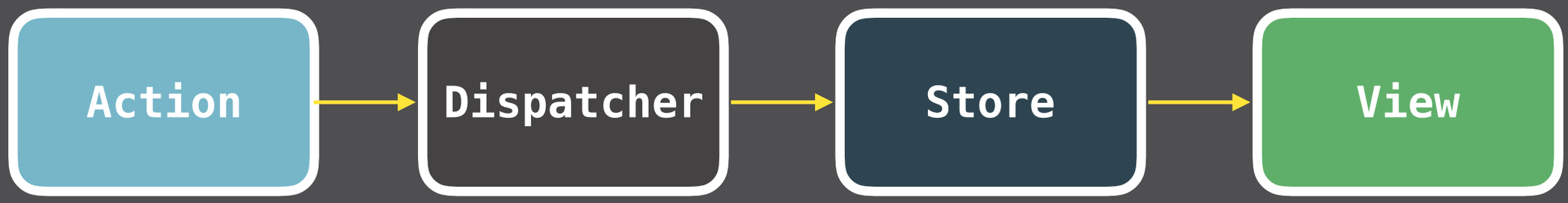
- Introduced by Facebook(@F8 in 2014)
- Originally for building client-side web applications
- Data flows in one direction

Unidirectional Data Flow

"The most important concept is that data flows **in one direction.**"

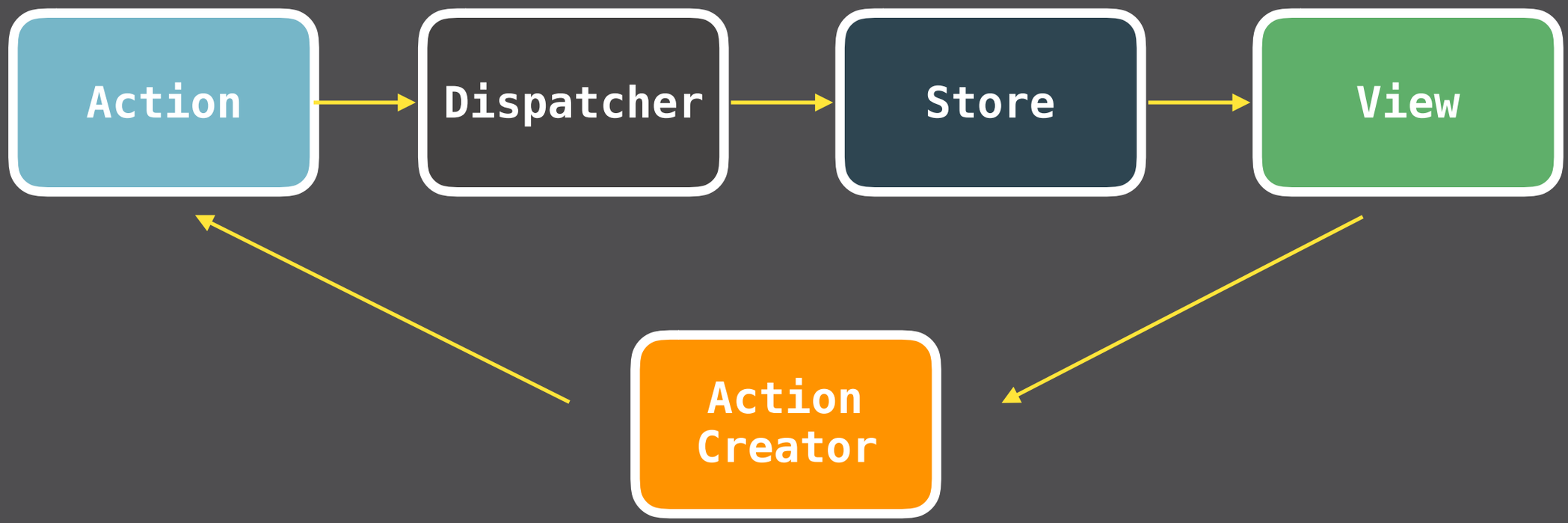
https://github.com/facebook/flux/tree/master/examples/flux-concepts#overview

Flow of Data

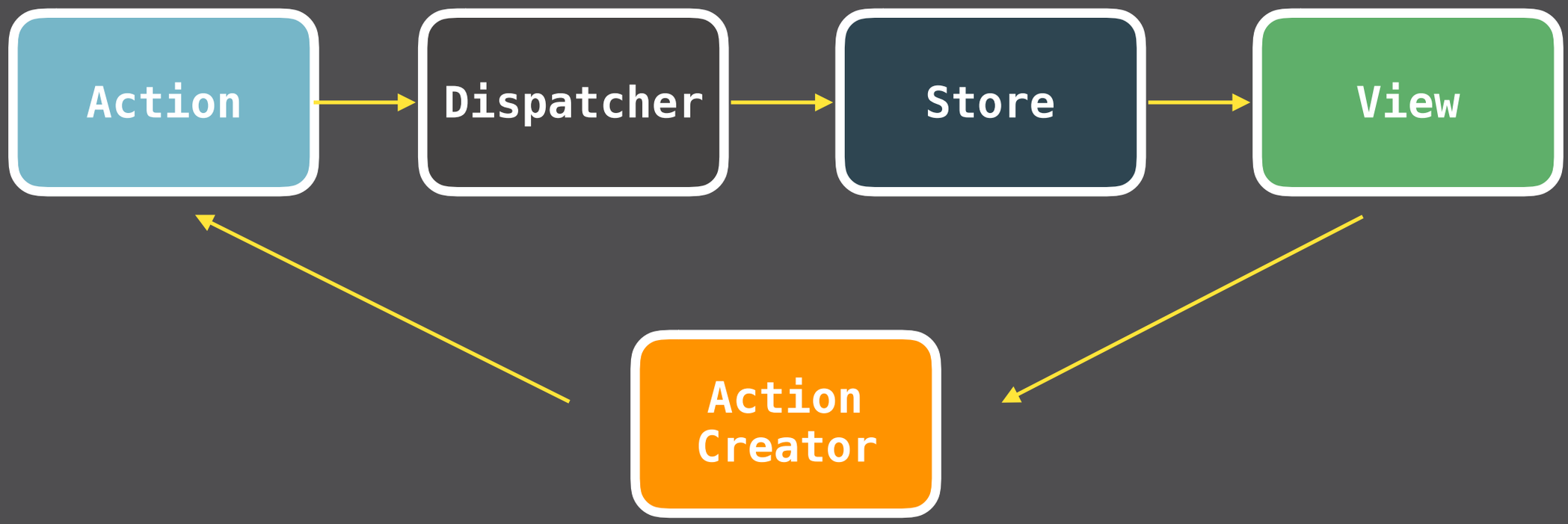


http://facebook.github.io/flux/docs/in-depth-overview.html#structure-and-data-flow

Flow of Data



Components


Action
Dispatcher
Store
View
Action Creator

View

Action Creator

Action

Dispatcher

Store

View

Action Creator

Action

Dispatcher

Store

- Render UI based on Store states/values
- Store subscriber
- Has no states

View

Action Creator

Action

Dispatcher

Store

- Helper
- Creates Actions
- Passes Actions to Dispatcher
- Has no states

View

Action Creator

Action

Dispatcher

Store

- Container
- Created by ActionCreator
- Dispatched by Dispatcher
- Delivered to Store

View

Action Creator

Action

Dispatcher

Store

- Central hub for all of the data flows
- Dispatches Actions to Stores
- Has no states

View

Action Creator

Action

Dispatcher

Store

- State manager
- Changes its states and data only by receiving Actions
- **Has states / internal logic**

Store

The ONLY

state manager

View

Action Creator

Action

Dispatcher

Store

- State manager
- Changes its states and data only by receiving Actions
- **Has states / internal logic**

View

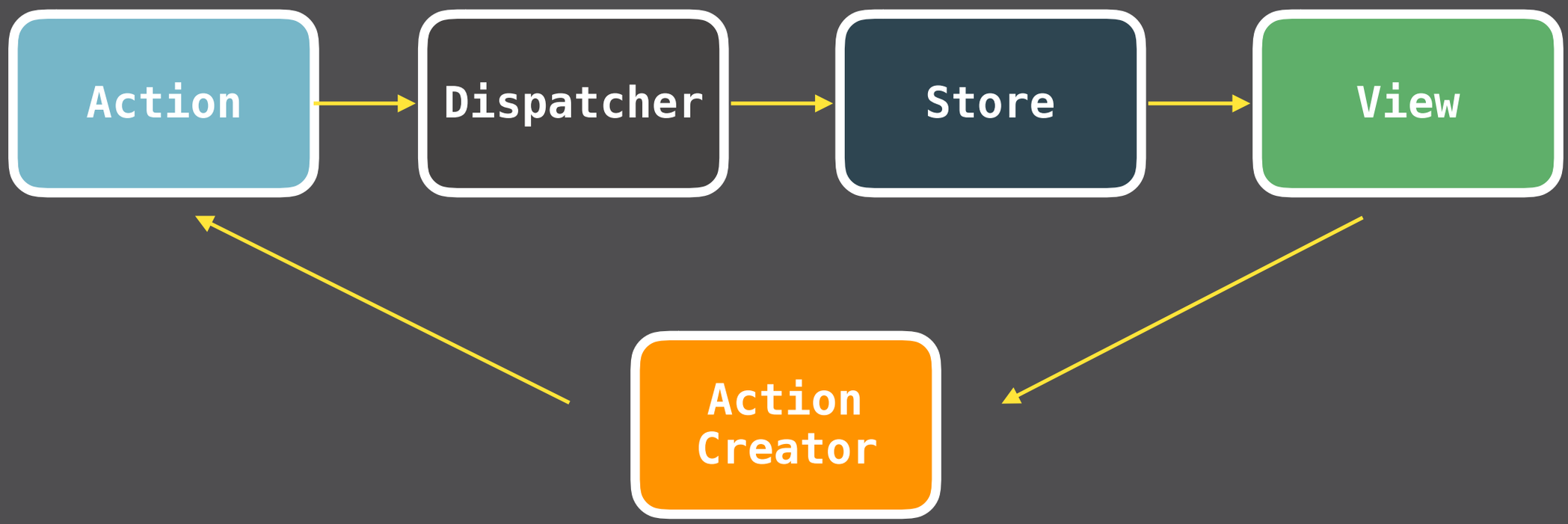
Action Creator

Action

Dispatcher

Store

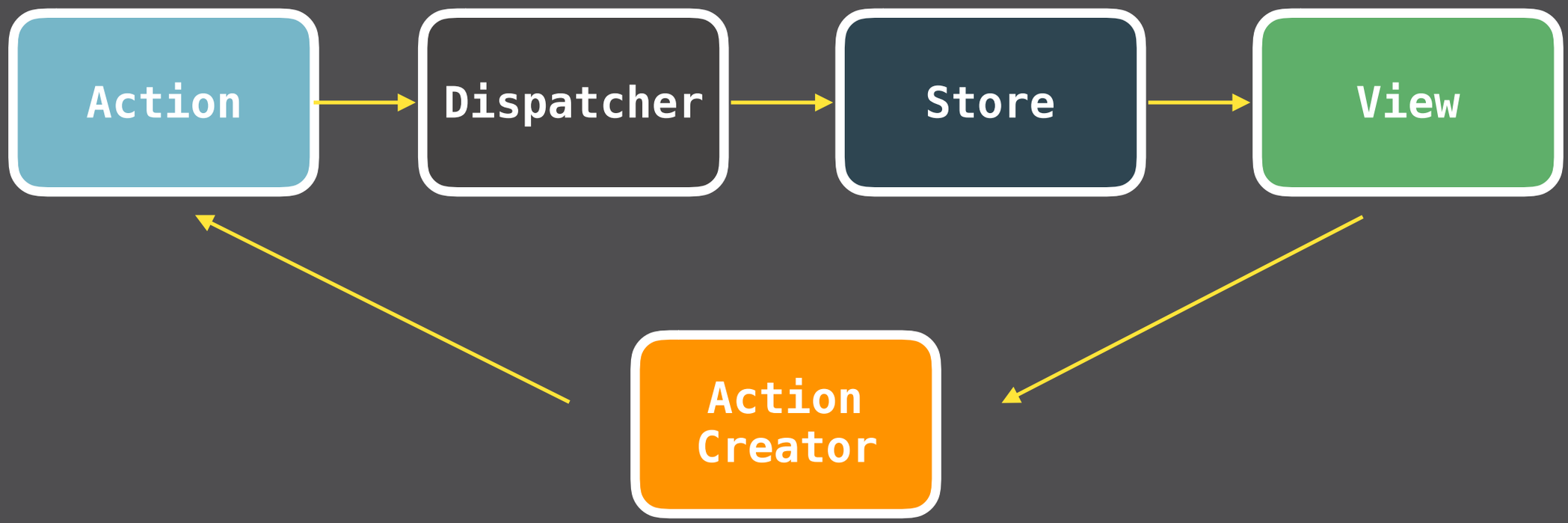
Why?



- Not complicated diagram or boilerplates
- You know where to look
- Code consistency
- Easy to test
- Easy to debug
- ...

DRY

Construct Flux Dataflow



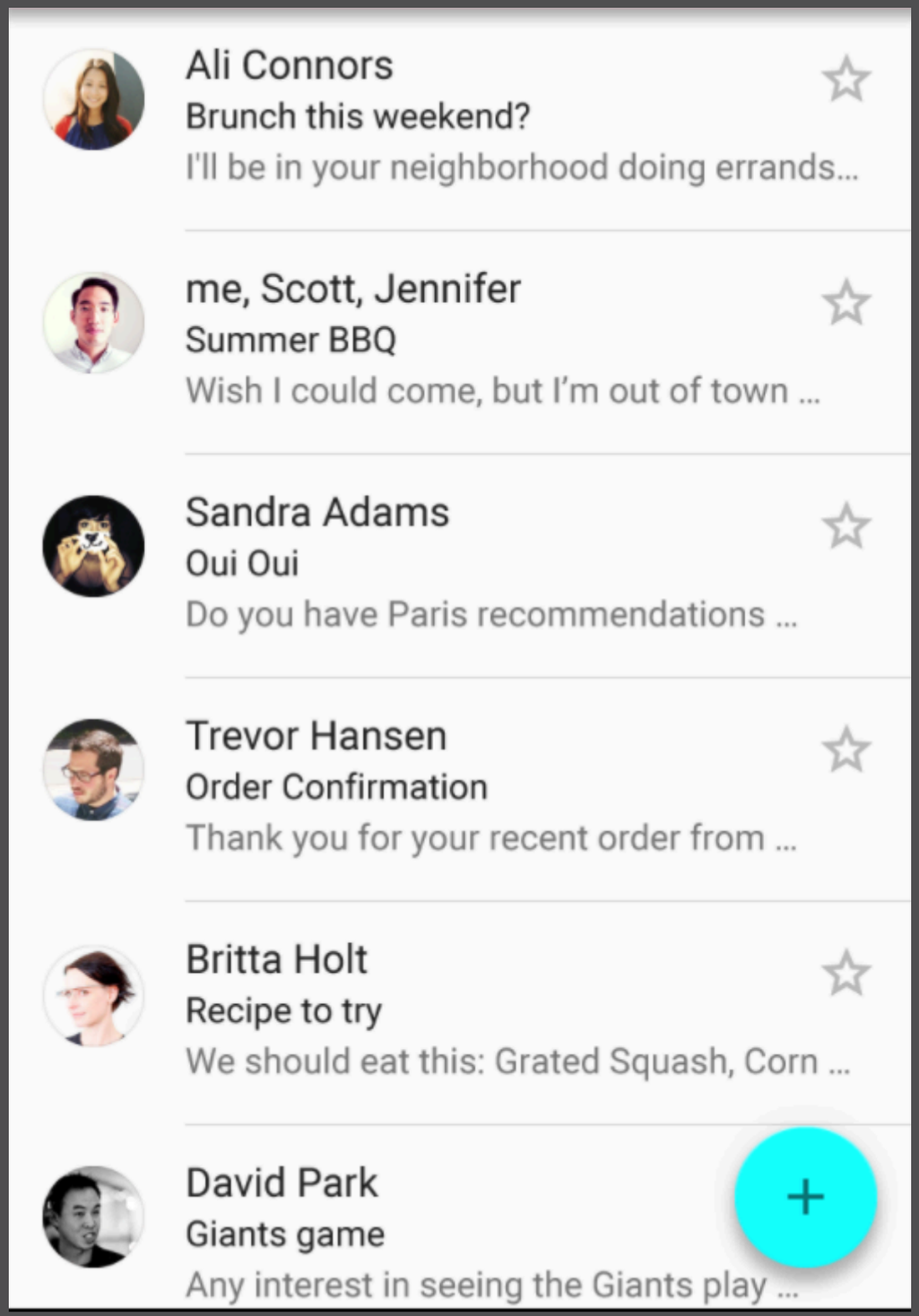


image: https://material.io/guidelines/components/lists.html#lists-specs

Libraries in this example

- RxJava
- Dagger
- Data Binding
- greenrobot/EventBus
- uber/AutoDispose
- trello/navi

View

View

Activity
Fragment
Views(e.g. RecyclerView)
Custom View

View

onCreate(savedInstanceState: Bundle?)

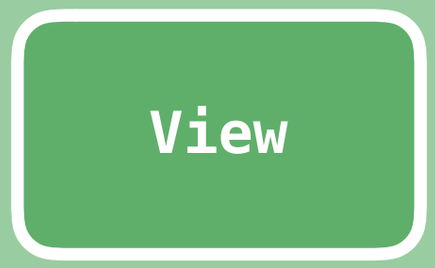
```
class UserListActivity : BaseActivity() {

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

}
```

```
class UserListActivity : BaseActivity() {

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

}
```

```
<!-- activity_user_list.xml -->

<?xml version="1.0" encoding="utf-8"?>
<layout … >

<data>
  <import type="android.view.View"/>
  <variable
      name="isLoading"
      type="boolean"
      />
</data>
  …
  <android.support.v7.widget.RecyclerView
    android:id=“@+id/user_list"
    …
    />

  <ProgressBar
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:visibility="@{isLoading ? View.VISIBLE : View.GONE}"
      …
      />
…
```

```
class UserListActivity : BaseActivity() {

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

}
```

```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

}
```



3 Steps in Activity / Fragment

1. Initialize View
2. Subscribe State
3. Call ActionCreator’s functions

View

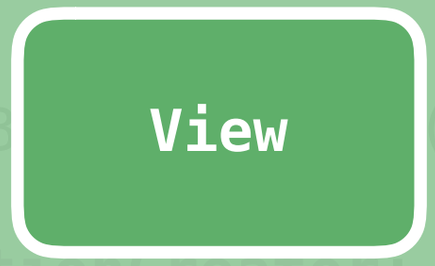
```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var userListStore: UserListStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

  }
```

1.

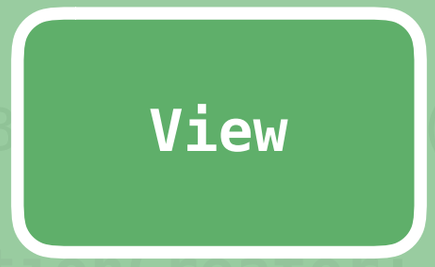
Initialize View

```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

    // Initialize View
    binding.fab.setOnClickListener { doSomething() }
    binding.userList.adapter = userListAdapter
    binding.userList.layoutManager = LinearLayoutManager(this)

}
```

```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

    // Initialize View
    binding.fab.setOnClickListener { doSomething() }
    binding.userList.adapter = userListAdapter
    binding.userList.layoutManager = LinearLayoutManager(this)

}
```



2. Subscribe State



2. Subscribe “Loading” State

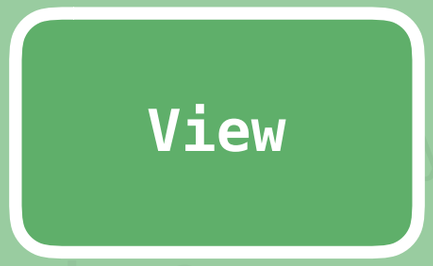
```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

    // Initialize View
    binding.fab.setOnClickListener { doSomething() }
    binding.userList.adapter = userListAdapter
    binding.userList.layoutManager = LinearLayoutManager(this)

    // Subscribe State
    loadingStore.loadingState.autoDisposable(scope)
      .subscribe { binding.isLoading = it }

  }
```

```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

    // Initialize View
    binding.fab.setOnClickListener { doSomething() }
    binding.userList.adapter = userListAdapter
    binding.userList.layoutManager = LinearLayoutManager(this)

    // Subscribe State
    loadingStore.loadingState.autoDisposable(scope)
      .subscribe { binding.isLoading = it }

  }
```

```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

    // Initialize View
    binding.fab.setOnClickListener { doSomething() }
    binding.userList.adapter = userListAdapter
    binding.userList.layoutManager = LinearLayoutManager(this)

    // Subscribe State
    loadingStore.loadingState.autoDisposable(scope)
      .subscribe { binding.isLoading = it }

  }
```

```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

    // initView()
    binding.fab.setOnClickListener { doSomething() }
    binding.userList.adapter = userListAdapter
    binding.userList.layoutManager = LinearLayoutManager(this)

    // subscribeState()
    loadingStore.loadingState.autoDisposable(scope)
      .subscribe { binding.isLoading = it }

  }
```

```
class UserListActivity : Activity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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


    initView()
    subscribeState()

  }
…
```



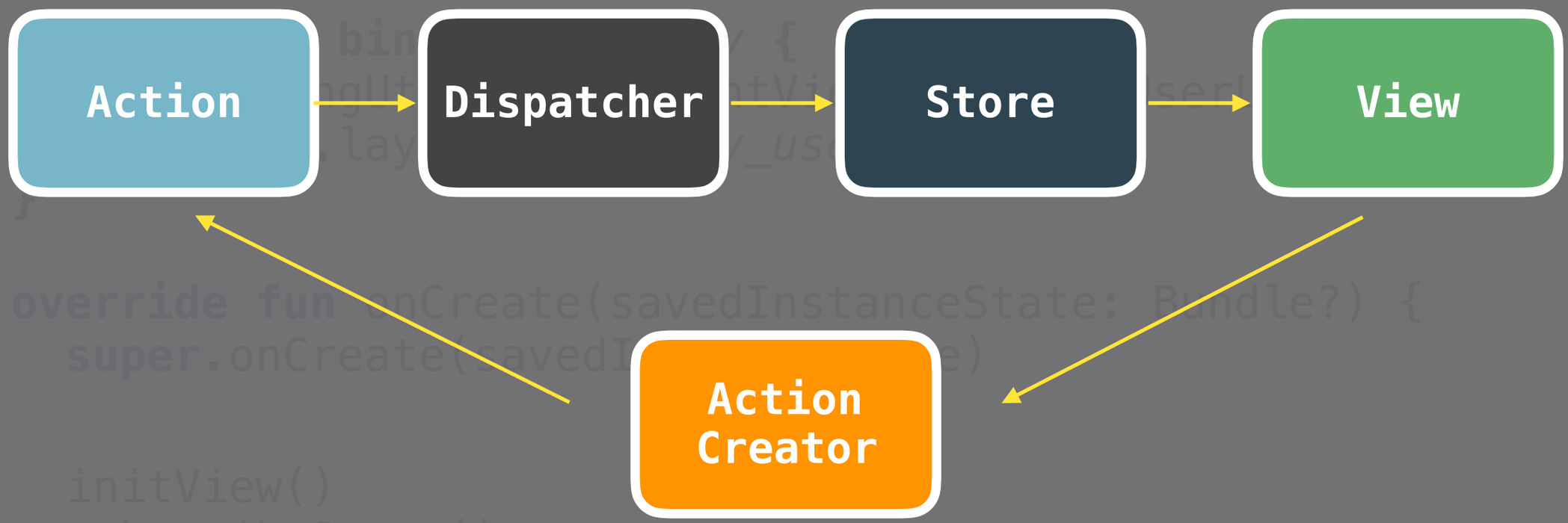
3.

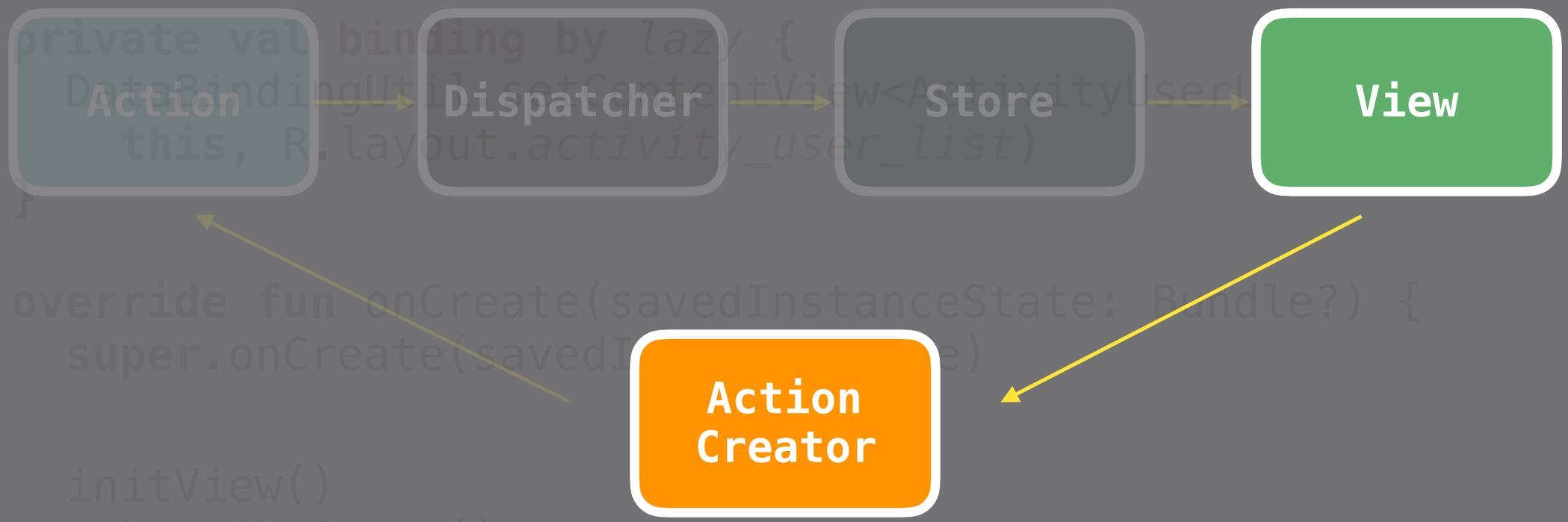
Call ActionCreators’ functions

```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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


    initView()
    subscribeState()

    // Create & Dispatch Action
    actionCreator.showUserList()
  }
…
```

```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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

    initView()
    subscribeState()

    // Create & Dispatch Action
    actionCreator.showUserList()
  }
…
```

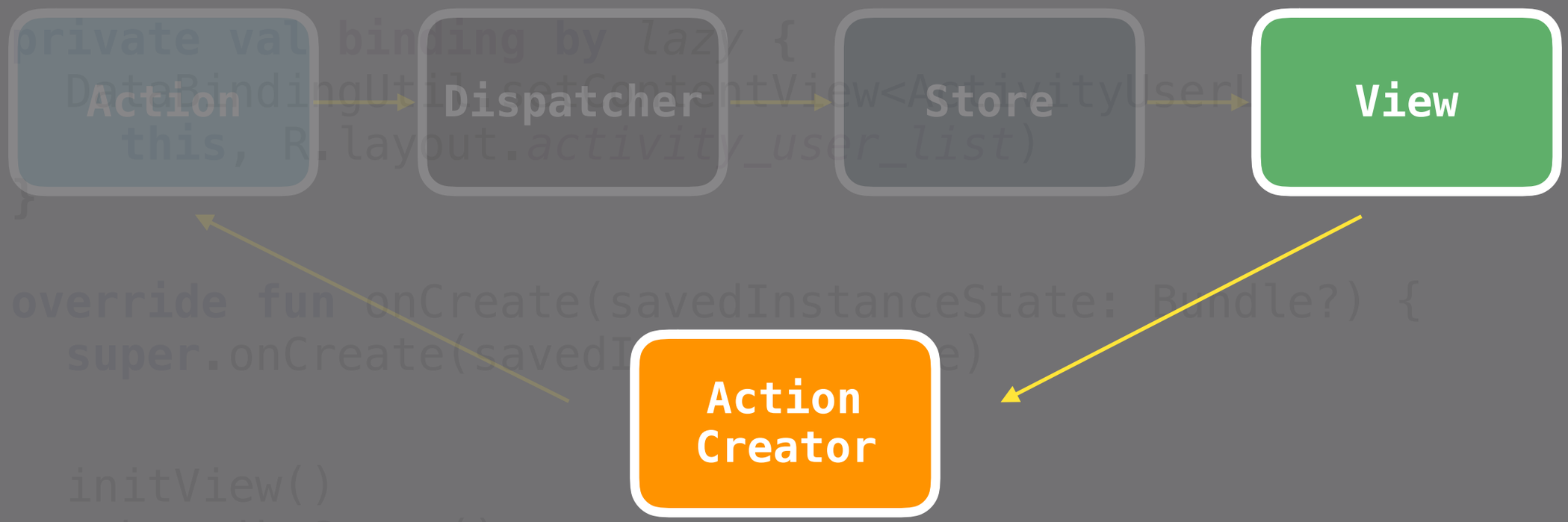
View

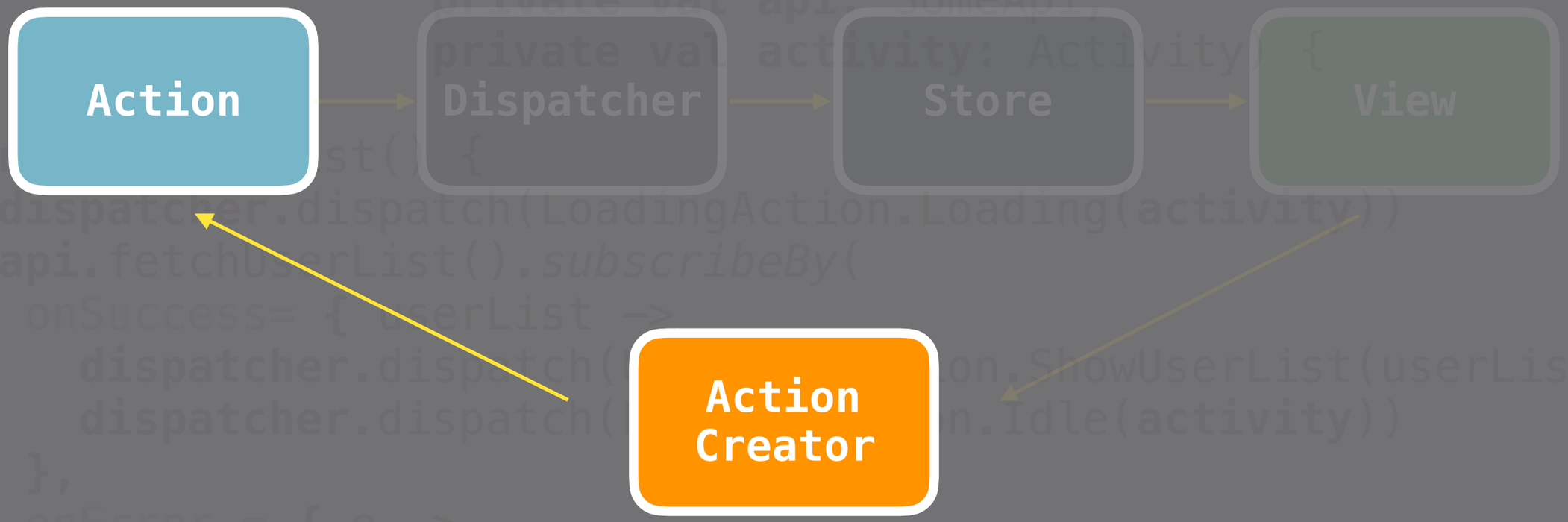

Action
Dispatcher
Store
View
Action
Creator


Action
Dispatcher
Store
View
Action Creator

```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

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


    initView()
    subscribeState()

    // Create & Dispatch Action
    actionCreator.showUserList()
  }
…
```

Action Creator


Action
Dispatcher
Store
View
Action Creator


Action
Dispatcher
Store
View
Action Creator

Action Creator

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
     onSuccess= { userList ->
       dispatcher.dispatch(UserListAction.ShowUserList(userList))
       dispatcher.dispatch(LoadingAction.Idle(activity))
     },
     onError = { e ->
       dispatcher.dispatch(UserListAction.Error(e))
       dispatcher.dispatch(LoadingAction.Idle(activity))
     })
  }
}
```

Action Creator

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
      onSuccess= { userList ->
        dispatcher.dispatch(UserListAction.ShowUserList(userList))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      },
      onError = { e ->
        dispatcher.dispatch(UserListAction.Error(e))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      })
  }
}
```

Action Creator

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
      onSuccess= { userList ->
        dispatcher.dispatch(UserListAction.ShowUserList(userList))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      },
      onError = { e ->
        dispatcher.dispatch(UserListAction.Error(e))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      })
  }
}
```

Action Creator

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
     onSuccess= { userList ->
       dispatcher.dispatch(UserListAction.ShowUserList(userList))
       dispatcher.dispatch(LoadingAction.Idle(activity))
     },
     onError = { e ->
       dispatcher.dispatch(UserListAction.Error(e))
       dispatcher.dispatch(LoadingAction.Idle(activity))
     })
  }
}
```

Action

LoadingAction

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribe(
      onSuccess = { userList ->
        dispatcher.dispatch(UserListAction.ShowUserList(userList))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      },
      onError = { e ->
        dispatcher.dispatch(UserListAction.Error(e))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      })
  }
}
```

Action

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
      onSuccess= { userList ->
        dispatcher.dispatch(UserListAction.ShowUserList(userList))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      },
      onError = { e ->
        dispatcher.dispatch(UserListAction.Error(e))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      })
  }
}
```

Action

```
sealed class LoadingAction(activity: Activity) {

    val activityKey: String = activity.componentName.className

    class Idle(activity: Activity) : LoadingAction(activity)

    class Loading(activity: Activity) : LoadingAction(activity)
}
```


```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribe(
        onSuccess= { userList ->
          dispatcher.dispatch(UserListAction.ShowUserList(userList))
          dispatcher.dispatch(LoadingAction.Idle(activity))
        },
        onError = { e ->
          dispatcher.dispatch(UserListAction.Error(e))
          dispatcher.dispatch(LoadingAction.Idle(activity))
        })
  }
}
```

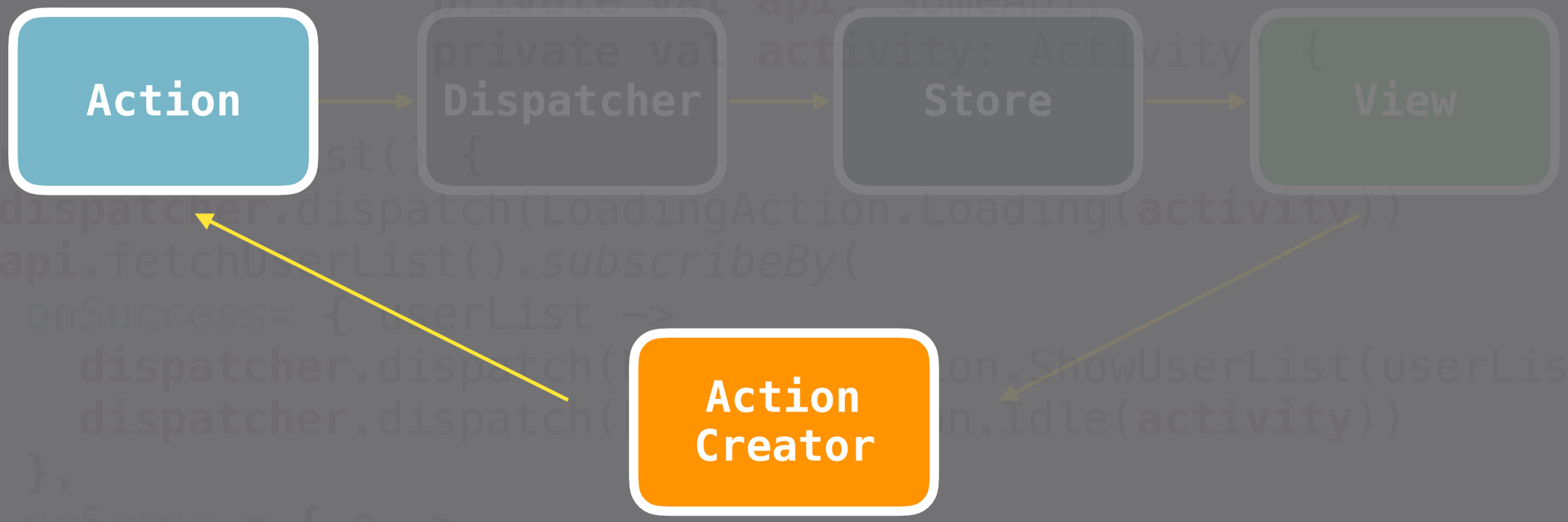
UserListAction

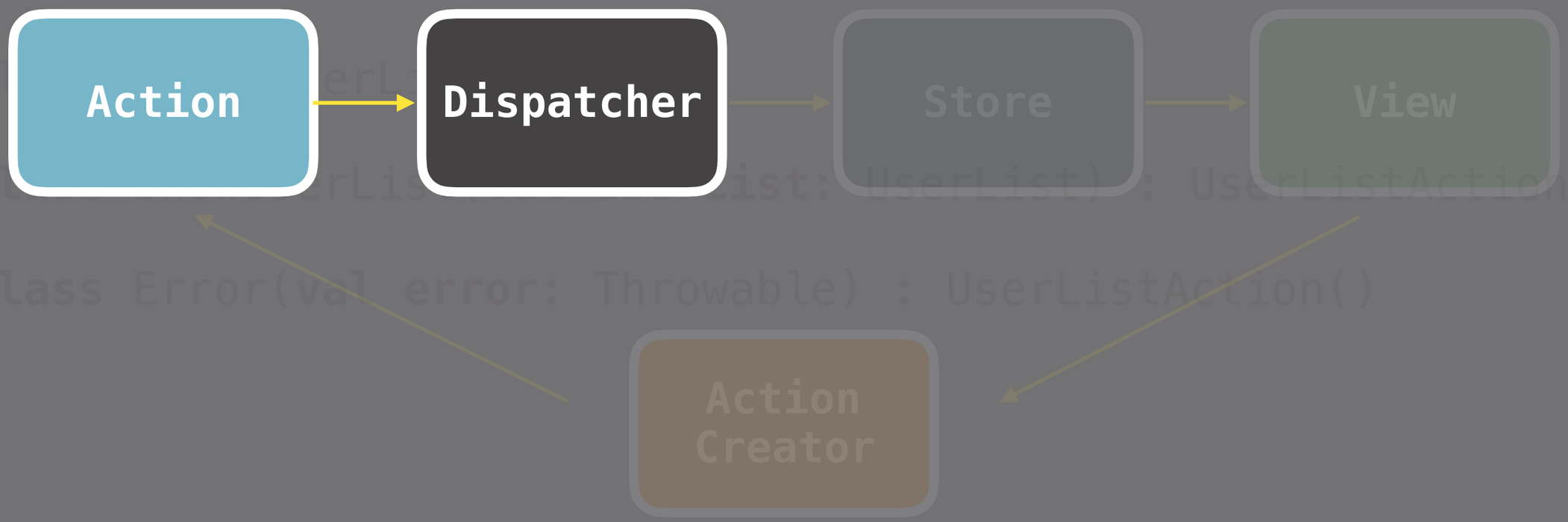
Action

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
      onSuccess= { userList ->
        dispatcher.dispatch(UserListAction.ShowUserList(userList))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      },
      onError = { e ->
        dispatcher.dispatch(UserListAction.Error(e))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      })
  }
}
```

Action

```
sealed class UserListAction {

  class ShowUserList(val userList: UserList) : UserListAction()
  class Error(val error: Throwable) : UserListAction()
}
```


Action
Dispatcher
Store
View
Action Creator


Action
Dispatcher
Store
View
Action Creator

Action Creator

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
     onSuccess= { userList ->
       dispatcher.dispatch(UserListAction.ShowUserList(userList))
       dispatcher.dispatch(LoadingAction.Idle(activity))
     },
     onError = { e ->
       dispatcher.dispatch(UserListAction.Error(e))
       dispatcher.dispatch(LoadingAction.Idle(activity))
     })
  }
}
```

Dispatcher

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
      onSuccess= { userList ->
        dispatcher.dispatch(UserListAction.ShowUserList(userList))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      },
      onError = { e ->
        dispatcher.dispatch(UserListAction.Error(e))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      })
  }
}
```

Dispatcher

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
      onSuccess= { userList ->
        dispatcher.dispatch(UserListAction.ShowUserList(userList))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      },
      onError = { e ->
        dispatcher.dispatch(UserListAction.Error(e))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      })
  }
}
```

Dispatcher

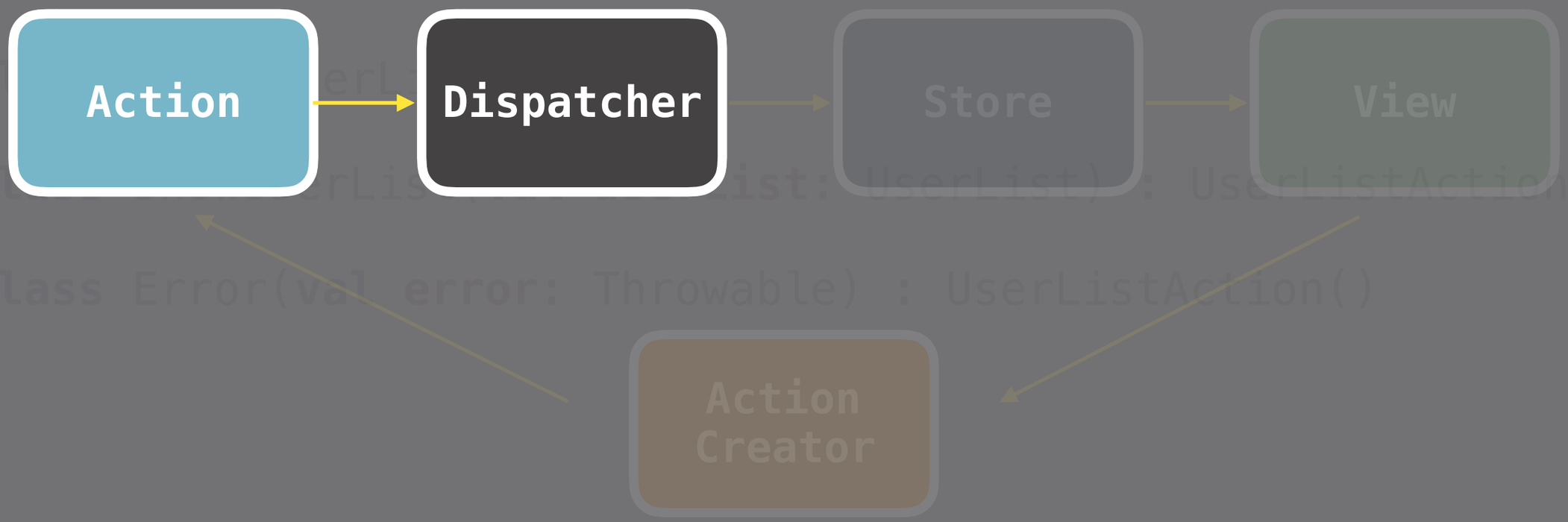
```
class Dispatcher {

 private val eventBus = EventBus.builder()
         .addIndex(DispatcherIndex())
         .throwSubscriberException(BuildConfig.DEBUG)
         .build()

fun dispatch(event: Any) {
  eventBus.post(event)
}

fun register(observer: Any) {
  eventBus.register(observer)
}

fun unregister(observer: Any) {
  eventBus.unregister(observer)
}
```

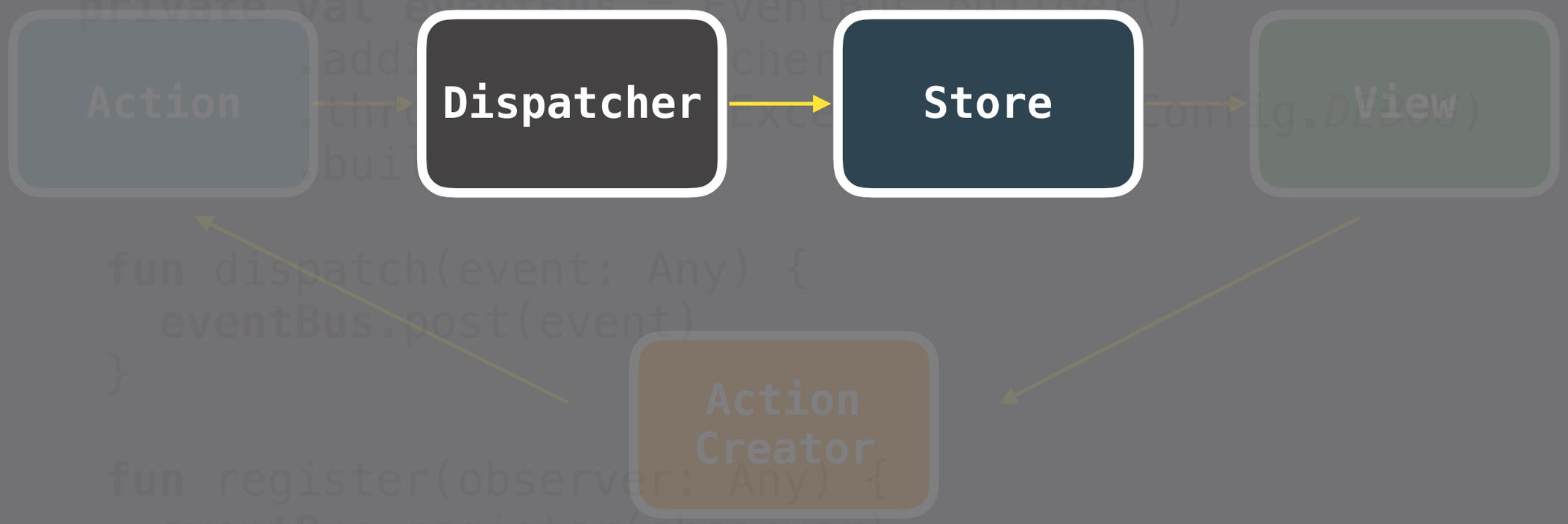
Dispatcher

For Broadcasting Actions:

- **RxJava**
- **EventBus**

... and more?


Action
Dispatcher
Store
View
Action Creator


Action
Dispatcher
Store
View
Action Creator

Dispatcher

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
      onSuccess= { userList ->
        dispatcher.dispatch(UserListAction.ShowUserList(userList))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      },
      onError = { e ->
        dispatcher.dispatch(UserListAction.Error(e))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      })
  }
}
```

Dispatch
LoadingAction

Dispatcher

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
      onSuccess= { userList ->
        dispatcher.dispatch(UserListAction.ShowUserList(userList))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      },
      onError = { e ->
        dispatcher.dispatch(UserListAction.Error(e))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      })
  }
}
```

Store

```
@PerActivity class LoadingStore
@Inject constructor(private val activityKey: String,
dispatcher: Dispatcher, navi: NaviComponent) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: LoadingAction) {
    …
  }
```

Store

```
@PerActivity class LoadingStore
@Inject constructor(private val activityKey: String,
dispatcher: Dispatcher, navi: NaviComponent) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: LoadingAction) {
    …
  }
```

Store

```
@PerActivity class LoadingStore
@Inject constructor(private val activityKey: String,
dispatcher: Dispatcher, navi: NaviComponent) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: LoadingAction) {
    …
  }
```

For Observing Lifecycle Events:

- **AAC: Lifecycle**
- **RxLifecycle: Navi**

... and more?

Store

```
@PerActivity class LoadingStore
@Inject constructor(private val activityKey: String,
dispatcher: Dispatcher, navi: NaviComponent) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: LoadingAction) {
    …
  }
```

Store

```
@PerActivity class LoadingStore
@Inject constructor(private val activityKey: String,
dispatcher: Dispatcher, navi: NaviComponent) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

@Subscribe(threadMode = ThreadMode.MAIN)
fun on(action: LoadingAction) {
  …
}
```

Store

```
@PerActivity class LoadingStore
@Inject constructor(private val activityKey: String,
dispatcher: Dispatcher, navi: NaviComponent) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: LoadingAction) {
    …
  }
```

Store

```
@PerActivity class LoadingStore
@Inject constructor( … ) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init { … }
```

```
@Subscribe(threadMode = ThreadMode.MAIN)
fun on(action: LoadingAction) {
  if (action.activityKey == activityKey) {
    when (action) {
      is LoadingAction.Loading -> {
        loadingSubject.onNext(true)
      }

      is LoadingAction.Idle -> {
        loadingSubject.onNext(false)
      }
  …
```

Store

```
@PerActivity class LoadingStore
@Inject constructor( … ) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init { … }

@Subscribe(threadMode = ThreadMode.MAIN)
fun on(action: LoadingAction) {
  if (action.activityKey == activityKey) {
    when (action) {
      is LoadingAction.Loading -> {
        loadingSubject.onNext(true)
      }

      is LoadingAction.Idle -> {
        loadingSubject.onNext(false)
      }

  …
```

Store

```
@PerActivity class LoadingStore
@Inject constructor( … ) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init { … }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: LoadingAction) {
    if (action.activityKey == activityKey) {
      when (action) {
        is LoadingAction.Loading -> {
          loadingSubject.onNext(true)
        }

        is LoadingAction.Idle -> {
          loadingSubject.onNext(false)
        }

    …
```

Store

```
@PerActivity class LoadingStore
@Inject constructor( … ) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init { … }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: LoadingAction) {
    if (action.activityKey == activityKey) {
      when (action) {
        is LoadingAction.Loading -> {
          loadingSubject.onNext(true)
        }

        is LoadingAction.Idle -> {
          loadingSubject.onNext(false)
        }
    …
```

Dispatch
UserListAction

Dispatcher

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
      onSuccess= { userList ->
        dispatcher.dispatch(UserListAction.ShowUserList(userList))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      },
      onError = { e ->
        dispatcher.dispatch(UserListAction.Error(e))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      })
  }
}
```

Dispatcher

```
@PerActivity class UserListActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi,
                    private val activity: Activity) {

  fun showUserList() {
    dispatcher.dispatch(LoadingAction.Loading(activity))
    api.fetchUserList().subscribeBy(
      onSuccess= { userList ->
        dispatcher.dispatch(UserListAction.ShowUserList(userList))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      },
      onError = { e ->
        dispatcher.dispatch(UserListAction.Error(e))
        dispatcher.dispatch(LoadingAction.Idle(activity))
      })
  }
}
```

Store

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val userList = ObservableArrayList<User>()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: UserListAction) {
    when (action) {
      is UserListAction.ShowUserList -> {
        userList.addAll(action.userList.data)
      }
      is UserListAction.Error -> {
        …
      }
    }
   …
```

Store

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val userList = ObservableArrayList<User>()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: UserListAction) {
    when (action) {
      is UserListAction.ShowUserList -> {
        userList.addAll(action.userList.data)
      }
      is UserListAction.Error -> {
        …
      }
    }
  …
```

Store

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val userList = ObservableArrayList<User>()
  …

  fun addOnListChangedCallback(callback:
  ObservableList.OnListChangedCallback<ObservableList<User>>) {
    userList.addOnListChangedCallback(callback)
}

  fun removeOnListChangedCallback(callback:
  ObservableList.OnListChangedCallback<ObservableList<User>>) {
    userList.removeOnListChangedCallback(callback)
}
```

Store

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val userList = ObservableArrayList<User>()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: UserListAction) {
    when (action) {
      is UserListAction.ShowUserList -> {
        userList.addAll(action.userList.data)
      }
      is UserListAction.Error -> {
        …
      }
    }
   …
```

Store

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val userList = ObservableArrayList<User>()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: UserListAction) {
    when (action) {
      is UserListAction.ShowUserList -> {
        userList.addAll(action.userList.data)
      }
      is UserListAction.Error -> {
        …
      }
    }
  …
```

Store

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val userList = ObservableArrayList<User>()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: UserListAction) {
    when (action) {
      is UserListAction.ShowUserList -> {
        userList.addAll(action.userList.data)
      }
      is UserListAction.Error -> {
        …
      }
    }
  …
```


Action
Dispatcher
Store
View
Action
Creator

Streaming Loading State Change Using RxJava

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val listSubject = PublishSubject.create<List<User>>()
  val listState: Observable<List<User>> = listSubject.hide()

  init {
```

Store

```
@PerActivity class LoadingStore
@Inject constructor( … ) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init { … }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: LoadingAction) {
    if (action.activityKey == activityKey) {
      when (action) {
        is LoadingAction.Loading -> {
          loadingSubject.onNext(true)
        }

        is LoadingAction.Idle -> {
          loadingSubject.onNext(false)
        }
    …
```

Store

```
@PerActivity class LoadingStore
@Inject constructor( … ) {

  private val loadingSubject = PublishSubject.create<Boolean>()
  val loadingState: Observable<Boolean> = loadingSubject.hide()

  init { … }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: LoadingAction) {
    if (action.activityKey == activityKey) {
      when (action) {
        is LoadingAction.Loading -> {
          loadingSubject.onNext(true)
        }

        is LoadingAction.Idle -> {
          loadingSubject.onNext(false)
        }
    …
```

```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

  override fun onCreate(sa                State: Bundle?) {
    super.onCreate(savedIn                )

    // initView()
    binding.fab.setOnClickListener { doSomething() }
    binding.userList.adapter = userListAdapter
    binding.userList.layoutManager = LinearLayoutManager(this)

    // subscribeState()
    loadingStore.loadingState.autoDisposable(scope)
      .subscribe { binding.isLoading = it }

}
```

View

```
class UserListActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserListActionCreator
  @Inject lateinit var loadingStore: LoadingStore
  @Inject lateinit var userListAdapter: UserListAdapter

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserListBinding>(
      this, R.layout.activity_user_list)
  }

  override fun onCreate(sa                State: Bundle?) {
    super.onCreate(savedIn                )

    // initView()
    binding.fab.setOnClickListener { doSomething() }
    binding.userList.adapter = userListAdapter
    binding.userList.layoutManager = LinearLayoutManager(this)

    // subscribeState()
    loadingStore.loadingState.autoDisposable(scope)
      .subscribe { binding.isLoading = it }

}
```

View

```
<!-- activity_user_list.xml -->

<?xml version="1.0" encoding="utf-8"?>
<layout … >

<data>
  <import type="android.view.View"/>
  <variable
      name="isLoading"
      type="boolean"
      />
</data>
  …
  <android.support.v7.widget.RecyclerView
    android:id=“@+id/user_list"
    …
    />

  <ProgressBar
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:visibility="@{isLoading ? View.VISIBLE : View.GONE}"
      …
      />
…
```

```
<!-- activity_user_list.xml -->

<?xml version="1.0" encoding="utf-8"?>
<layout … >

<data>
  <import type="android.view.View"/>
  <variable
      name="isLoading"
      type="boolean"
      />
</data>
…
  <android.support.v7.widget.RecyclerView
    android:id="@+id/user_list"
    …
    />

  <ProgressBar
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:visibility="@{isLoading ? View.VISIBLE : View.GONE}"
      …
      />
…
```

Streaming UserList Change Using Data Binding

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val listSubject = PublishSubject.create<List<User>>()
  val listState: Observable<List<User>> = listSubject.hide()

  init {
```

Store

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val userList = ObservableArrayList<User>()

  init {
    navi.addListener(Event.CREATE, { dispatcher.register(this)})
    navi.addListener(Event.DESTROY, { dispatcher.unregister(this)})
  }

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: UserListAction) {
    when (action) {
      is UserListAction.ShowUserList -> {
        userList.addAll(action.userList.data)
      }
      is UserListAction.Error -> {
        …
      }
    }
  …
```

View

```
@PerActivity class UserListAdapter
@Inject constructor(
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {

  private val listChangeCallback = …

  init {
    navi.addListener(Event.CREATE, {
      store.addOnListChangedCallback(listChangeCallback) })
    navi.addListener(Event.DESTROY, {
      store.removeOnListChangedCallback(listChangeCallback) })
  }
…
```

View

```
@PerActivity class UserListAdapter
@Inject constructor(
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {

  private val listChangeCallback = …

  init {
    navi.addListener(Event.CREATE, {
      store.addOnListChangedCallback(listChangeCallback) })
    navi.addListener(Event.DESTROY, {
      store.removeOnListChangedCallback(listChangeCallback) })
  }
…
```

View

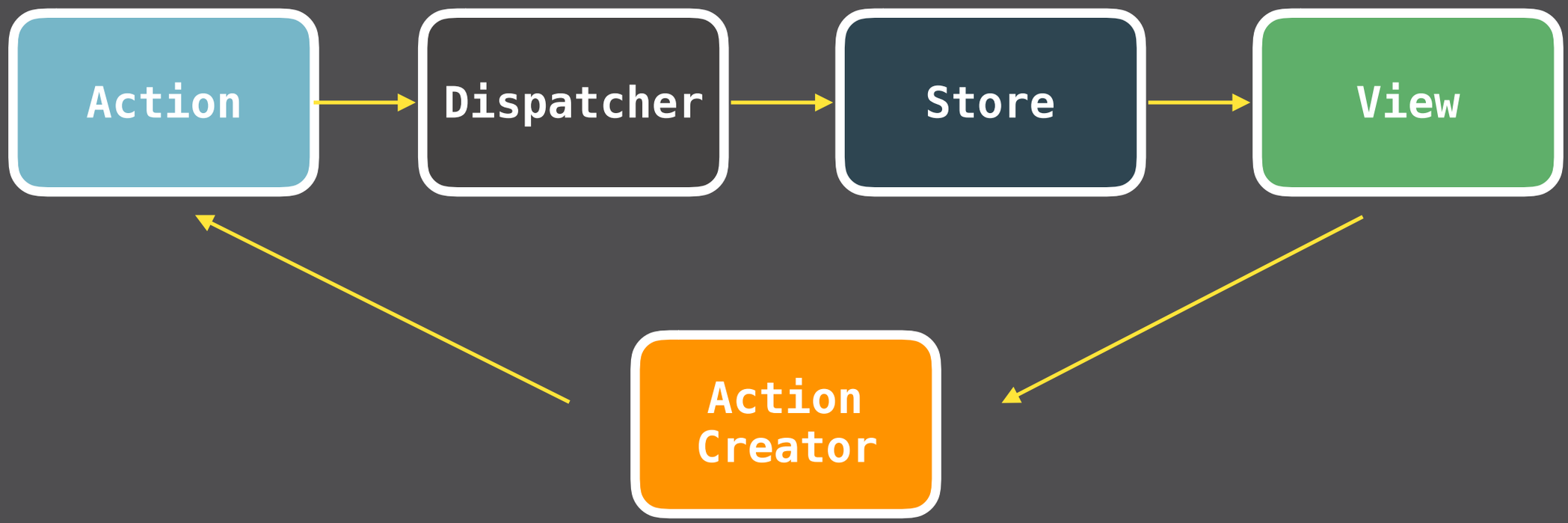
```
@PerActivity class UserListAdapter
@Inject constructor(
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {

  private val listChangeCallback = …

  init {
    navi.addListener(Event.CREATE, {
      store.addOnListChangedCallback(listChangeCallback) })
    navi.addListener(Event.DESTROY, {
      store.removeOnListChangedCallback(listChangeCallback) })
  }

  …
```

View

```
@PerActivity class UserListAdapter
@Inject constructor(
  private val store: UserListStore
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {

  private val listChangeCallback = …

  init { … }

  override fun onBindViewHolder( … ) {
    …
    holder.binding.user = store.getUserAt(position)
    holder.binding.executePendingBindings()
  }

  override fun getItemCount() = store.userListSize

  …
```


Action
Dispatcher
Store
View
Action
Creator

Constructed Flux Dataflow



Ali Connors

Brunch this weekend?

I'll be in your neighborhood doing errands...

me, Scott, Jennifer

Summer BBQ

Wish I could come, but I'm out of town ...

Sandra Adams

Oui Oui

Do you have Paris recommendations ...

Trevor Hansen

Order Confirmation

Thank you for your recent order from ...

Britta Holt

Recipe to try

We should eat this: Grated Squash, Corn ...

David Park

Giants game

Any interest in seeing the Giants play ...

Ali Connors
Brunch this weekend?
I'll be in your neighborhood doing er

me, Scott, Jennifer
Summer BBQ
Wish I could come, but I'm out of to

Sandra Adams
Oui Oui
Do you hav mmendations ...

Trevor Han
Order Confirmation
Thank you for your recent order fro

Britta Holt
Recipe to try
We should eat this: Grate

David Park
Giants game
Any interest in seeing the Giants play ...

‘Favorite’?

User Detail?

Ali Connors

Brunch this weekend?

I'll be in your neighborhood doing errands...

me, Scott, Jennifer

Summer BBQ

Wish I could come, but I'm out of town ...

Sandra Adams

Oui Oui

Do you have Paris recommendations ...

Trevor Hansen

Order Confirmation

Thank you for your recent order from ...

Britta Holt

Recipe to try

We should eat this: Grated Squash, Corn ...

David Park

Giants game

Any interest in seeing the Giants play ...

Ali Connors

Brunch this weekend?

I'll be in your neighborhood doing errands...

me, Scott, Jennifer

Summer BBQ

Wish I could come, but I'm out of town ...

Sandra Adams

Oui Oui

Do you have Paris recommendations ...

Trevor Hansen

Order Confirmation

Thank you for your recent order from ...

Britta Holt

Recipe to try

We should eat this: Grated Squash, Corn ...

David Park

Giants game

Any interest in seeing the Giants play ...

Ali Connors

Brunch this weekend?

I'll be in your neighborhood doing errands...

me, Scott, Jennifer

Summer BBQ

Wish I could come, but I'm out of town ...

Sandra Adams

Oui Oui

Do you have Paris recommendations ...

Trevor Hansen

Order Confirmation

Thank you for your recent order from ...

Britta Holt

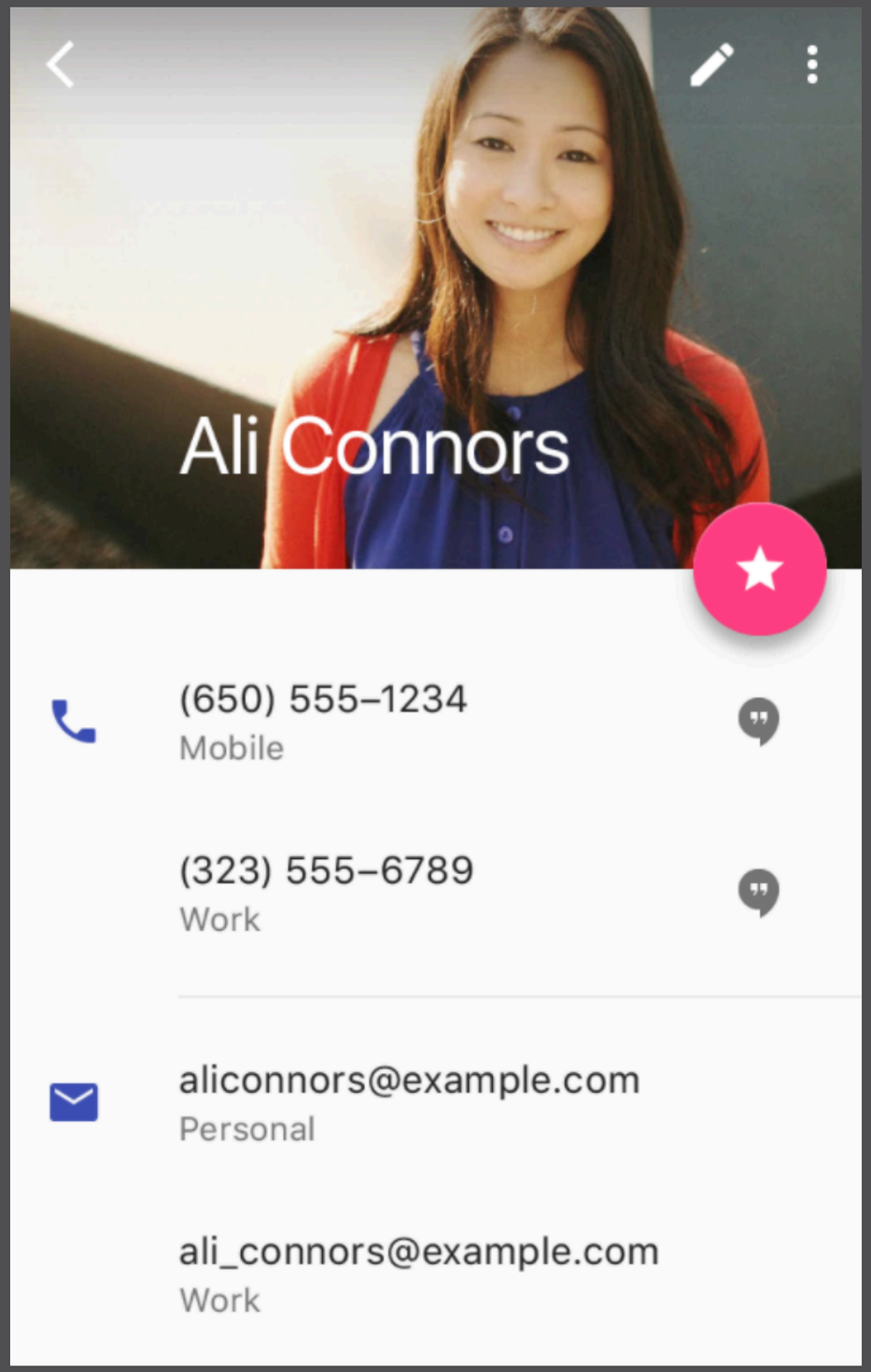
Recipe to try

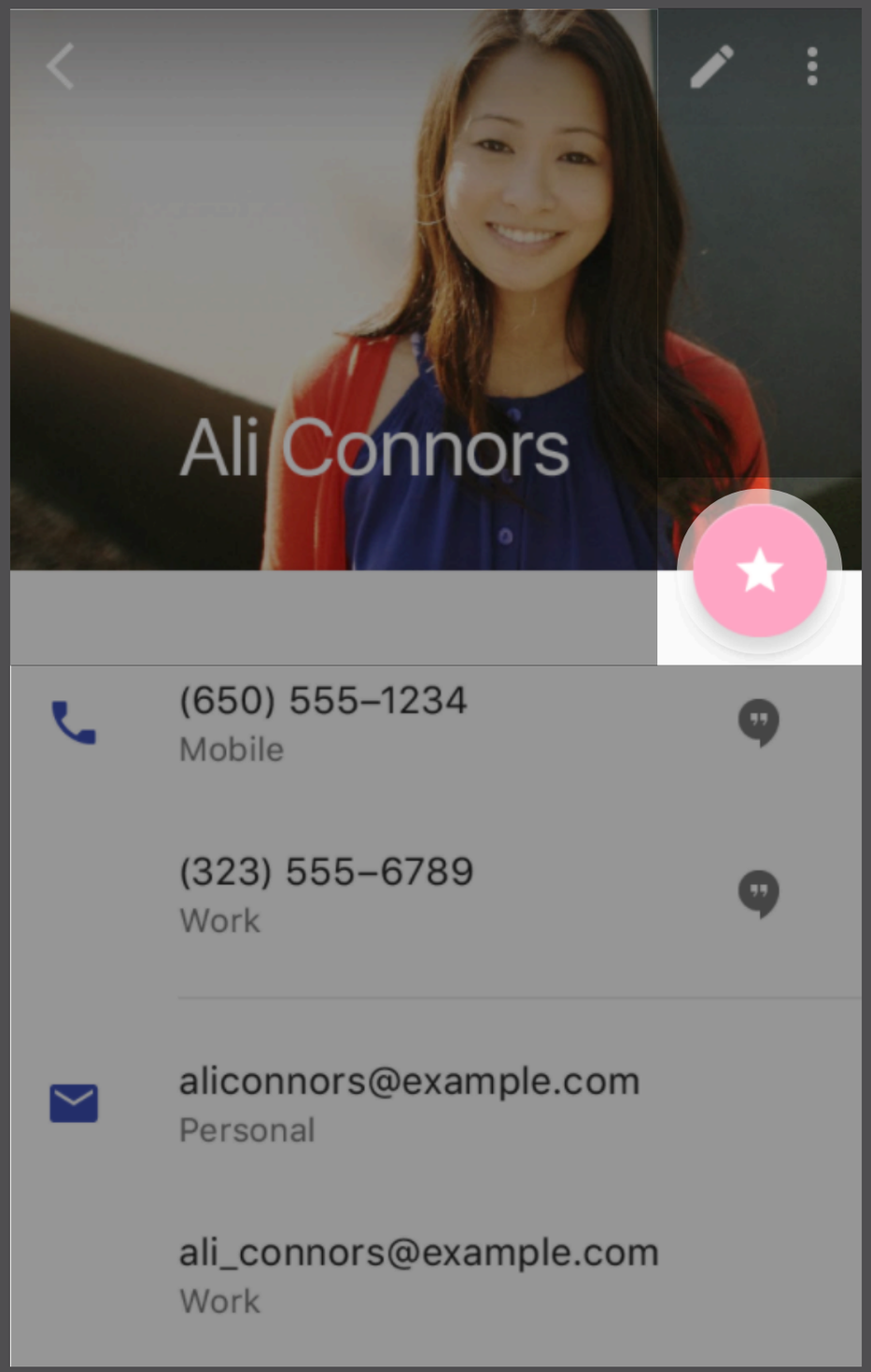
We should eat this: Grated Squash, Corn ...

David Park

Giants game

Any interest in seeing the Giants play ...


Ali Connors
(650) 555–1234
Mobile
(323) 555–6789
Work
aliconnors@example.com
Personal
ali_connors@example.com
Work


Ali Connors
(650) 555–1234
Mobile
(323) 555–6789
Work
aliconnors@example.com
Personal
ali_connors@example.com
Work

Ali Connors

Brunch this weekend?

I'll be in your neighborhood doing errands...

me, Scott, Jennifer

Summer BBQ

Wish I could come, but I'm out of town ...

Sandra Adams

Oui Oui

Do you have Paris recommendations ...

Trevor Hansen

Order Confirmation

Thank you for your recent order from ...

Britta Holt

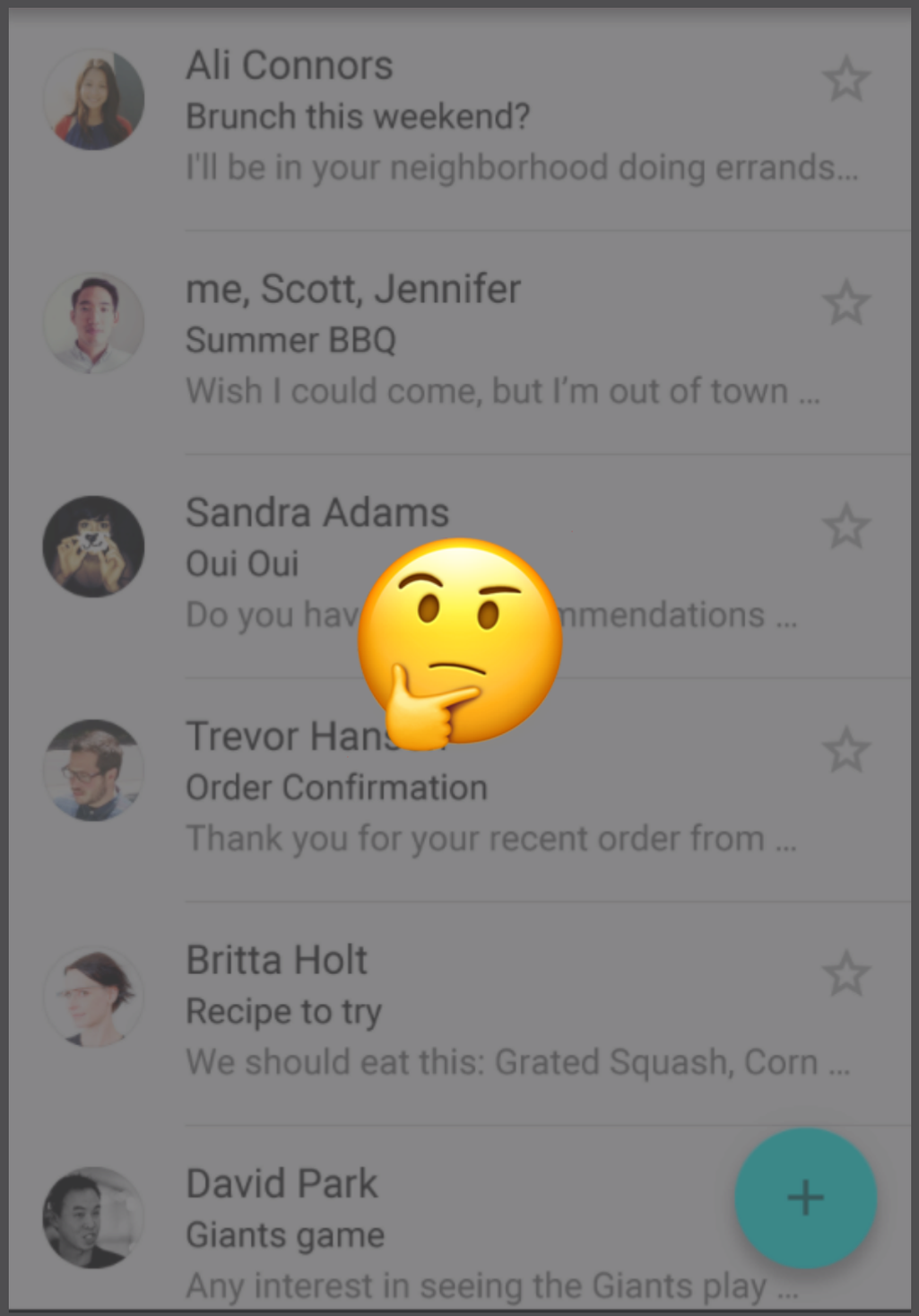
Recipe to try

We should eat this: Grated Squash, Corn ...

David Park

Giants game

Any interest in seeing the Giants play ...


Ali Connors
Brunch this weekend?
I'll be in your neighborhood doing errands...
me, Scott, Jennifer
Summer BBQ
Wish I could come, but I'm out of town ...
Sandra Adams
Oui Oui
Do you hav
mmendations ...
Trevor Hans
Order Confirmation
Thank you for your recent order from ...
Britta Holt
Recipe to try
We should eat this: Grated Squash, Corn ...
David Park
Giants game
Any interest in seeing the Giants play ...


Ali Connors
Brunch this weekend?
I'll be in your neighborhood doing er
me, Scott, Jennifer
Summer BBQ
Wish I could come, but I'm out of to
Sandra Adams
Oui Oui
Do you hav mmendations ...
Trevor Han
Order Confirmation
Thank you for your recent order from ...
Britta Holt
Recipe to try
We should eat this: Grated Squash, Corn ...
David Park
Giants game
Any interest in seeing the Giants play ...
'Favorite'?
User Detail?

View

```
@PerActivity class UserListAdapter
@Inject constructor(
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {

  init { … }

  override fun onBindViewHolder( … ) {
    …
    holder.binding.user = store.getUserAt(position)
    holder.binding.executePendingBindings()
  }
  …
```

View

```
@PerActivity class UserListAdapter
@Inject constructor(
  private val userActionCreator: UserActionCreator,
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {

  init { … }

  override fun onBindViewHolder( … ) {
    …
    holder.binding.user = store.getUserAt(position)
    holder.favIcon.setOnClickListener {
      userActionCreator.favorite(user)
    }
    holder.binding.executePendingBindings()
  }
  …
```

View

```
@PerActivity class UserListAdapter
@Inject constructor(
  private val userActionCreator: UserActionCreator,
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {

  init { … }

override fun onBindViewHolder( … ) {
  …
  holder.binding.user = store.getUserAt(position)
  holder.favIcon.setOnClickListener {
    userActionCreator.favorite(user)
  }
  holder.binding.executePendingBindings()
}
…
```

View

```
@PerActivity class UserListAdapter
@Inject constructor(
  private val userActionCreator: UserActionCreator,
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {

  init { … }

override fun onBindViewHolder( … ) {
  …
  holder.binding.user = store.getUserAt(position)
  holder.favIcon.setOnClickListener {
    userActionCreator.favorite(user)
  }
  holder.binding.executePendingBindings()
}
…
```

View

```
@PerActivity class UserListAdapter
@Inject constructor(
  private val userActionCreator: UserActionCreator,
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {

  init { … }

override fun onBindViewHolder( … ) {
  …
  holder.binding.user = store.getUserAt(position)
  holder.favIcon.setOnClickListener {
    userActionCreator.favorite(user)
  }
  holder.binding.executePendingBindings()
}
…
```

Action Creator

```
@PerActivity class UserActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi) {

  fun favorite(user: User) {
    api.favorite(user.id).subscribeBy(
     onComplete = {
       dispatcher.dispatch(UserAction.Favorite(user.id))
     },
     …
  }
}
```

Action Creator

```
@PerActivity class UserActionCreator
@Inject constructor(private val dispatcher: Dispatcher,
                    private val api: SomeApi) {

  fun favorite(user: User) {
    api.favorite(user.id).subscribeBy(
      onComplete = {
        dispatcher.dispatch(UserAction.Favorite(user.id))
      },
      …
  }
}
```

Store

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val userList = ObservableArrayList<User>()
  init { … }
  …

@Subscribe(threadMode = ThreadMode.MAIN)
fun on(action: UserAction) {
  when (action) {
    is UserAction.Favorite -> {
      userList.find { it.id == action.targetUserId }?.let {
        // Update or replace with new user
      }
    }
}
…
```

Store

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val userList = ObservableArrayList<User>()
  init { … }
  …

  @Subscribe(threadMode = ThreadMode.MAIN)
  fun on(action: UserAction) {
    when (action) {
      is UserAction.Favorite -> {
        userList.find { it.id == action.targetUserId }?.let {
          // Update or replace with new user
        }
      }
  }
  …
```

Ali Connors

Brunch this weekend?

I'll be in your neighborhood doing errands...

me, Scott, Jennifer

Summer BBQ

Wish I could come, but I'm out of town ...

Sandra Adams

Oui Oui

Do you have Paris recommendations ...

Trevor Hansen

Order Confirmation

Thank you for your recent order from ...

Britta Holt

Recipe to try

We should eat this: Grated Squash, Corn ...

David Park

Giants game

Any interest in seeing the Giants play ...

Ali Connors

Brunch this weekend?

I'll be in your neighborhood doing errands...

me, Scott, Jennifer

Summer BBQ

Wish I could come, but I'm out of town ...

Sandra Adams

Oui Oui

Do you have Paris recommendations ...

Trevor Hansen

Order Confirmation

Thank you for your recent order from ...

Britta Holt

Recipe to try

We should eat this: Grated Squash, Corn ...

David Park

Giants game

Any interest in seeing the Giants play ...

Ali Connors
Brunch this weekend?
I'll be in your neighborhood doing errands…

me, Scott, Jennifer
Summer BBQ
Wish I could come, but I'm out of town …

Sandra Adams
Oui Oui
Do you hav mmendations …

Trevor Han
Order Confirmation
Thank you for your recent order fro

Britta Holt
Recipe to try
We should eat this: Grate

David Park
Giants game
Any interest in seeing the Giants play …

‘Favorite’?

User Detail?

View

```
@PerActivity class UserListAdapter
@Inject constructor(
  private val userActionCreator: UserActionCreator,
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {

  init { … }

  override fun onBindViewHolder( … ) {
    …
    holder.binding.user = store.getUserAt(position)
    holder.favIcon.setOnClickListener {
      userActionCreator.favorite(user)
    }
    holder.binding.executePendingBindings()
  }
  …
```

View

```
@PerActivity class UserListAdapter
@Inject constructor(
  private val activityActionCreator: ActivityActionCreator,
  private val userActionCreator: UserActionCreator,
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {
  …
override fun onBindViewHolder( … ) {
  …
  holder.binding.user = store.getUserAt(position)
  holder.favIcon.setOnClickListener {
    userActionCreator.favorite(user)
  }
  holder.binding.root.setOnClickListener {
    activityActionCreator.startUserDetailActivity(user)
  }
  holder.binding.executePendingBindings()
}
…
```

View

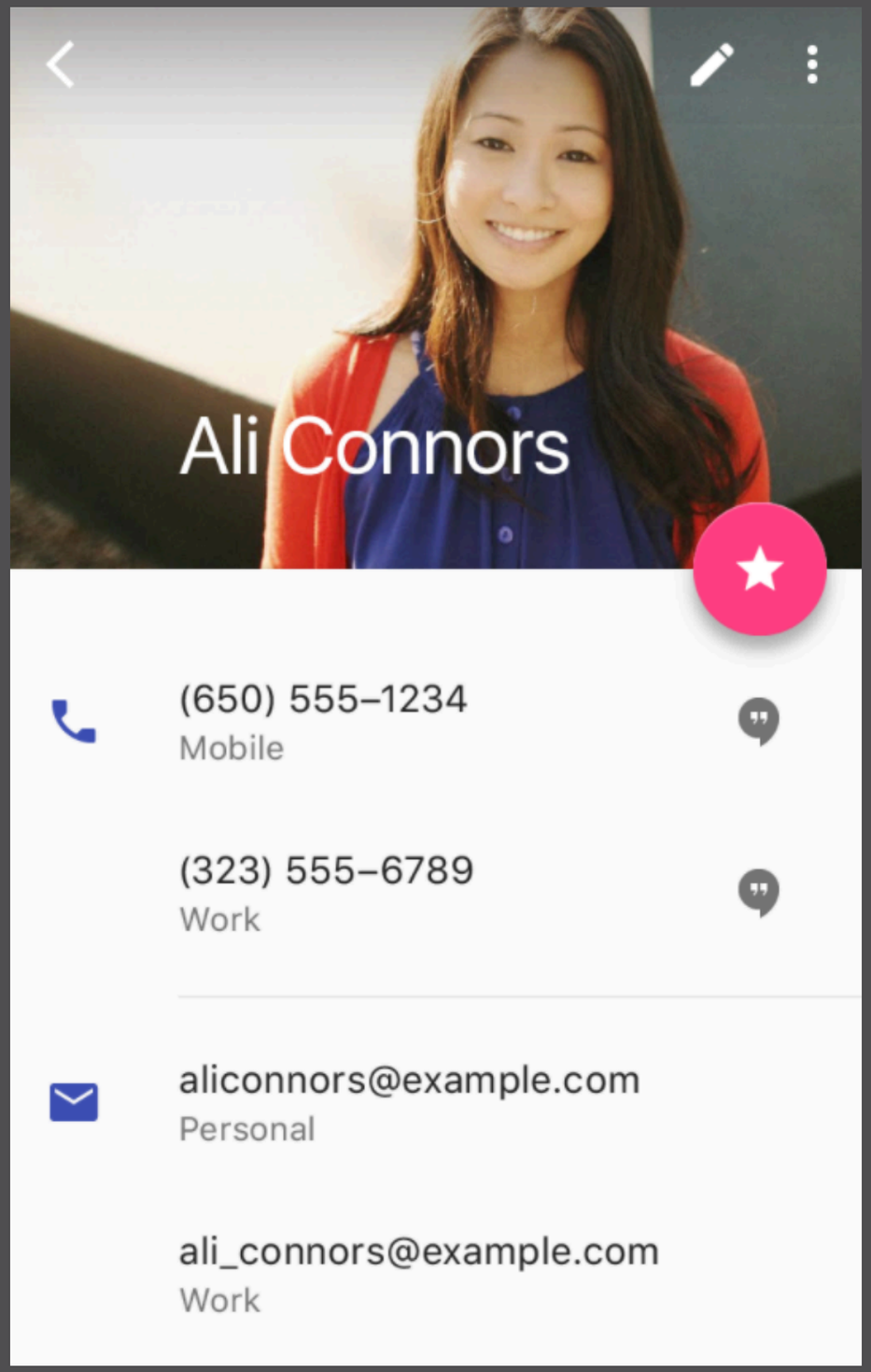
```
@PerActivity class UserListAdapter
@Inject constructor(
  private val activityActionCreator: ActivityActionCreator,
  private val userActionCreator: UserActionCreator,
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {
  …
override fun onBindViewHolder( … ) {
  …
  holder.binding.user = store.getUserAt(position)
  holder.favIcon.setOnClickListener {
    userActionCreator.favorite(user)
  }
  holder.binding.root.setOnClickListener {
    activityActionCreator.startUserDetailActivity(user)
  }
  holder.binding.executePendingBindings()
}
…
```

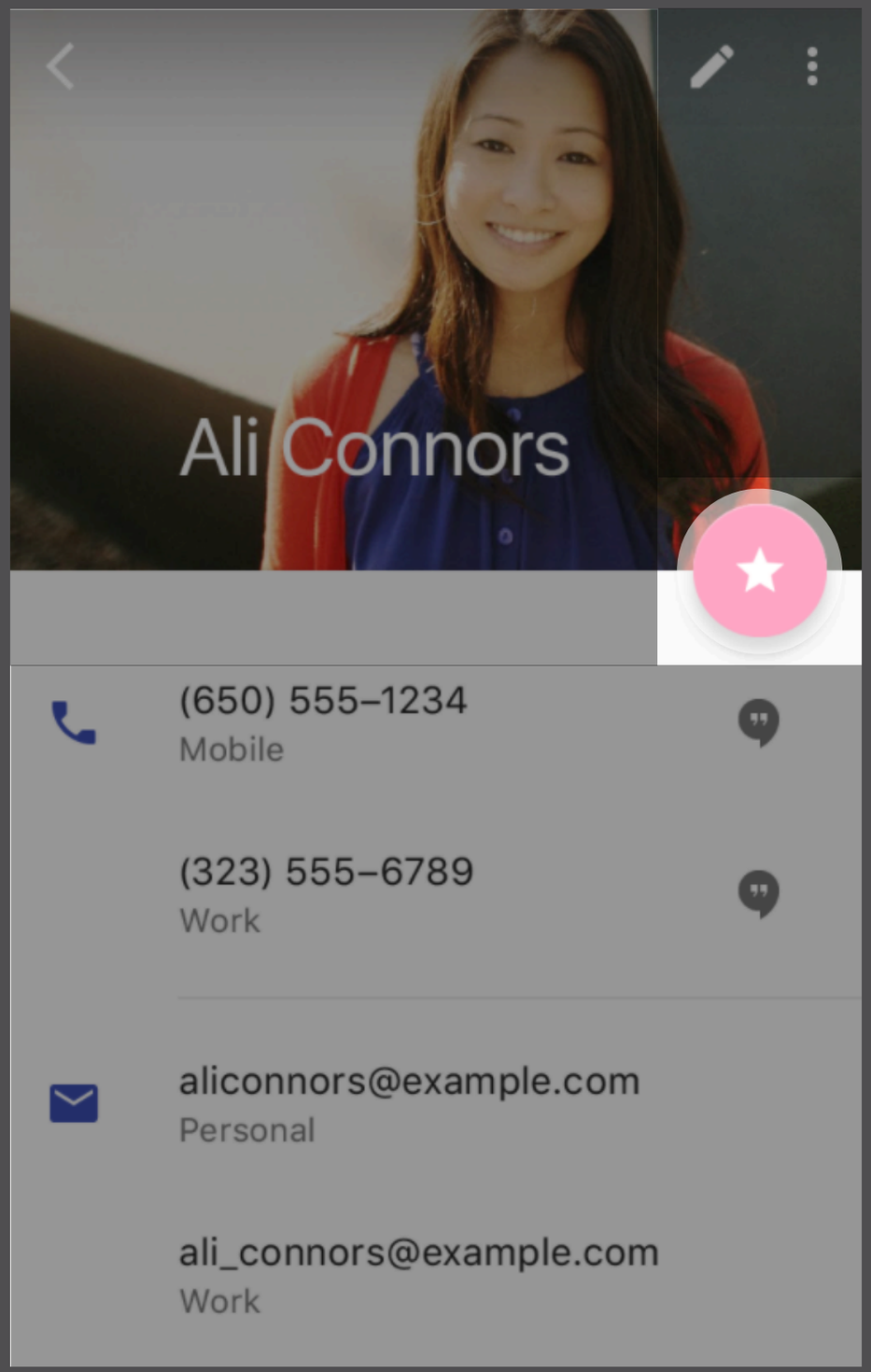
View

```
@PerActivity class UserListAdapter
@Inject constructor(
  private val activityActionCreator: ActivityActionCreator,
  private val userActionCreator: UserActionCreator,
  private val store: UserListStore,
  navi: NaviComponent):
  RecyclerView.Adapter<RecyclerView.ViewHolder>() {
  …
override fun onBindViewHolder( … ) {
  …
  holder.binding.user = store.getUserAt(position)
  holder.favIcon.setOnClickListener {
    userActionCreator.favorite(user)
  }
  holder.binding.root.setOnClickListener {
    activityActionCreator.startUserDetailActivity(user)
  }
  holder.binding.executePendingBindings()
}
…
```

Action Creator

```
@PerActivity class ActivityActionCreator
@Inject constructor(private val activity: Activity) {

  fun startUserDetailActivity(user: User) {
    val intent = UserDetailActivity.createIntent(activity, user)
    activity.startActivity(intent)
  }

  fun start…Activity ( … ) { … }

  fun start…Activity ( … ) { … }
}
```


Ali Connors
(650) 555–1234
Mobile
(323) 555–6789
Work
aliconnors@example.com
Personal
ali_connors@example.com
Work


Ali Connors
(650) 555–1234
Mobile
(323) 555–6789
Work
aliconnors@example.com
Personal
ali_connors@example.com
Work

View

```
class UserDetailActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserActionCreator
  @Inject lateinit var loadingStore: LoadingStore

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserDetailBinding>(
      this, R.layout.activity_user_detail)
  }

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

    initView()
    subscribeState()
  }
…
```

View

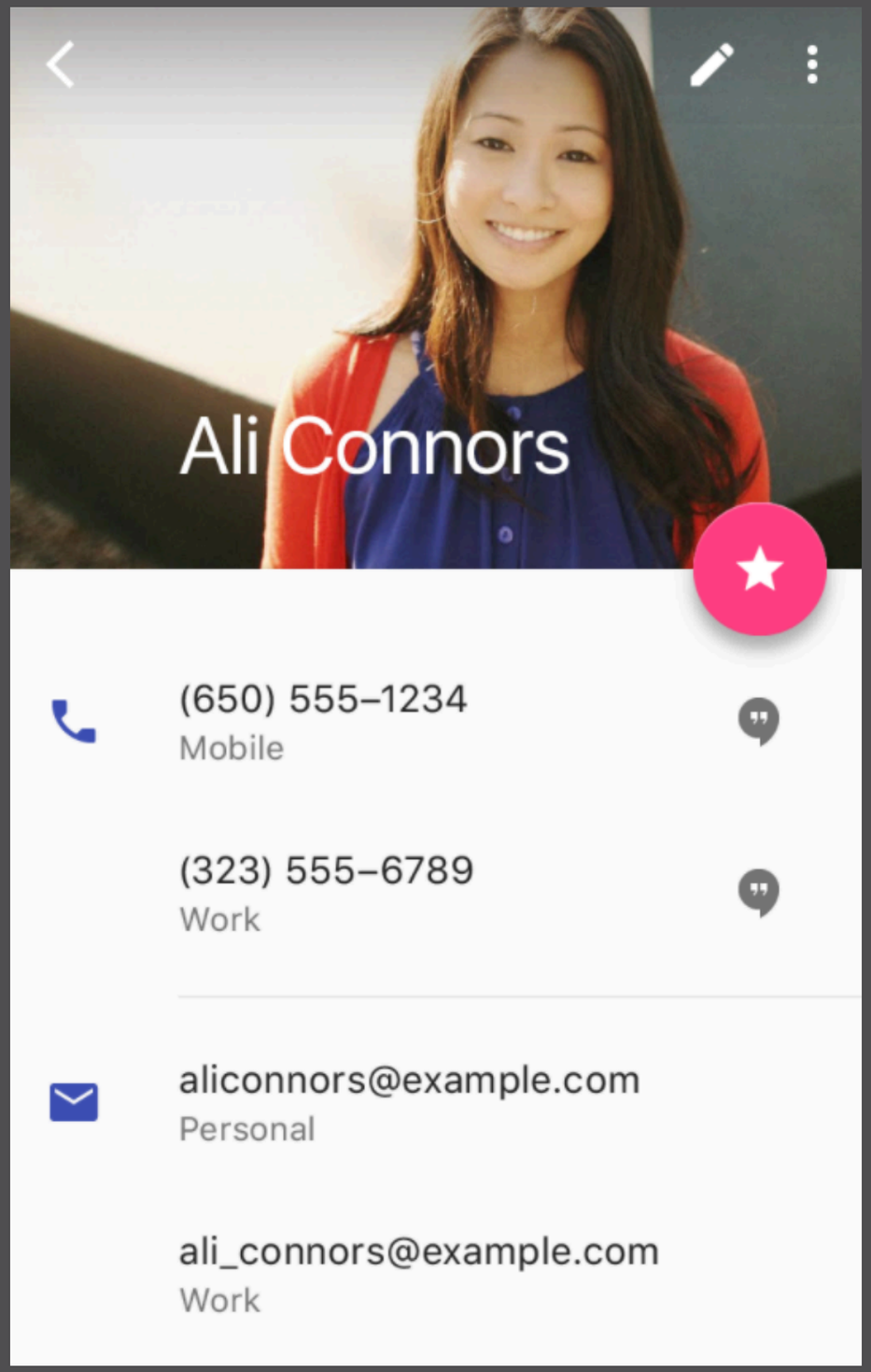
```
class UserDetailActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserActionCreator
  @Inject lateinit var loadingStore: LoadingStore

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserDetailBinding>(
      this, R.layout.activity_user_detail)
  }

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

    initView()
    subscribeState()
  }
…
```

View

```
class UserDetailActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserActionCreator
  @Inject lateinit var loadingStore: LoadingStore

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserDetailBinding>(
      this, R.layout.activity_user_detail)
  }

  override fun onCreate(savedInstanceState: Bundle?) {
    initView()
    …
  }

  private fun initView() {
    binding.fab.setOnClickListener {
      userActionCreator.favorite(user)
    }
  }
…
```

View

```
class UserDetailActivity : BaseActivity() {

  @Inject lateinit var actionCreator: UserActionCreator
  @Inject lateinit var loadingStore: LoadingStore

  private val binding by lazy {
    DataBindingUtil.setContentView<ActivityUserDetailBinding>(
      this, R.layout.activity_user_detail)
  }

  override fun onCreate(savedInstanceState: Bundle?) {
    initView()
    …
  }

  private fun initView() {
    binding.fab.setOnClickListener {
      userActionCreator.favorite(user)
    }
  }
…
```


Ali Connors
(650) 555–1234
Mobile
(323) 555–6789
Work
aliconnors@example.com
Personal
ali_connors@example.com
Work

Ali Connors

Brunch this weekend?

I'll be in your neighborhood doing errands...

me, Scott, Jennifer

Summer BBQ

Wish I could come, but I'm out of town ...

Sandra Adams

Oui Oui

Do you have Paris recommendations ...

Trevor Hansen

Order Confirmation

Thank you for your recent order from ...

Britta Holt

Recipe to try

We should eat this: Grated Squash, Corn ...

David Park

Giants game

Any interest in seeing the Giants play ...

Ali Connors
Brunch this weekend?
I'll be in your neighborhood doing errands…

me, Scott, Jennifer
Summer BBQ
Wish I could come, but I'm out of town …

Sandra Adams
Oui Oui
Do you hav mmendations …

Trevor Hansen
Order Confirmation
Thank you for your recent order from …

Britta Holt
Recipe to try
We should eat this: Grated Squash, Corn …

David Park
Giants game
Any interest in seeing the Giants play …

Store

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val userList = ObservableArrayList<User>()
  init { … }
  …

@Subscribe(threadMode = ThreadMode.MAIN)
fun on(action: UserAction) {
  when (action) {
    is UserAction.Favorite -> {
      userList.find { it.id == action.targetUserId }?.let {
        // Update or replace with new user
      }
    }
}
…
```

Store

```
@PerActivity class UserListStore
@Inject constructor(dispatcher: Dispatcher, navi: NaviComponent) {

  private val userList = ObservableArrayList<User>()
  init { … }
  …

@Subscribe(threadMode = ThreadMode.MAIN)
fun on(action: UserAction) {
  when (action) {
    is UserAction.Favorite -> {
      userList.find { it.id == action.targetUserId }?.let {
        // Update or replace with new user
      }
    }
}
  …
```

DRY

Code Consistency

Steps in Activity / Fragment

1. Initialize View
2. Subscribe State
3. Call ActionCreators' functions

View

Steps in Activity / Fragment

1. initView()
2. subscibeState()
3. actionCreator.doSomething()

Action Creator

Steps in ActionCreator

1. (Create & Dispatch LoadingAction)
2. Create & Dispatch Success / Error Action
3. (Create & Dispatch Idle Action)

The logic inside Store becomes the intuitive specification

- Kotlin sealed classes & “when” keyword
- Actions with descriptive names

Store

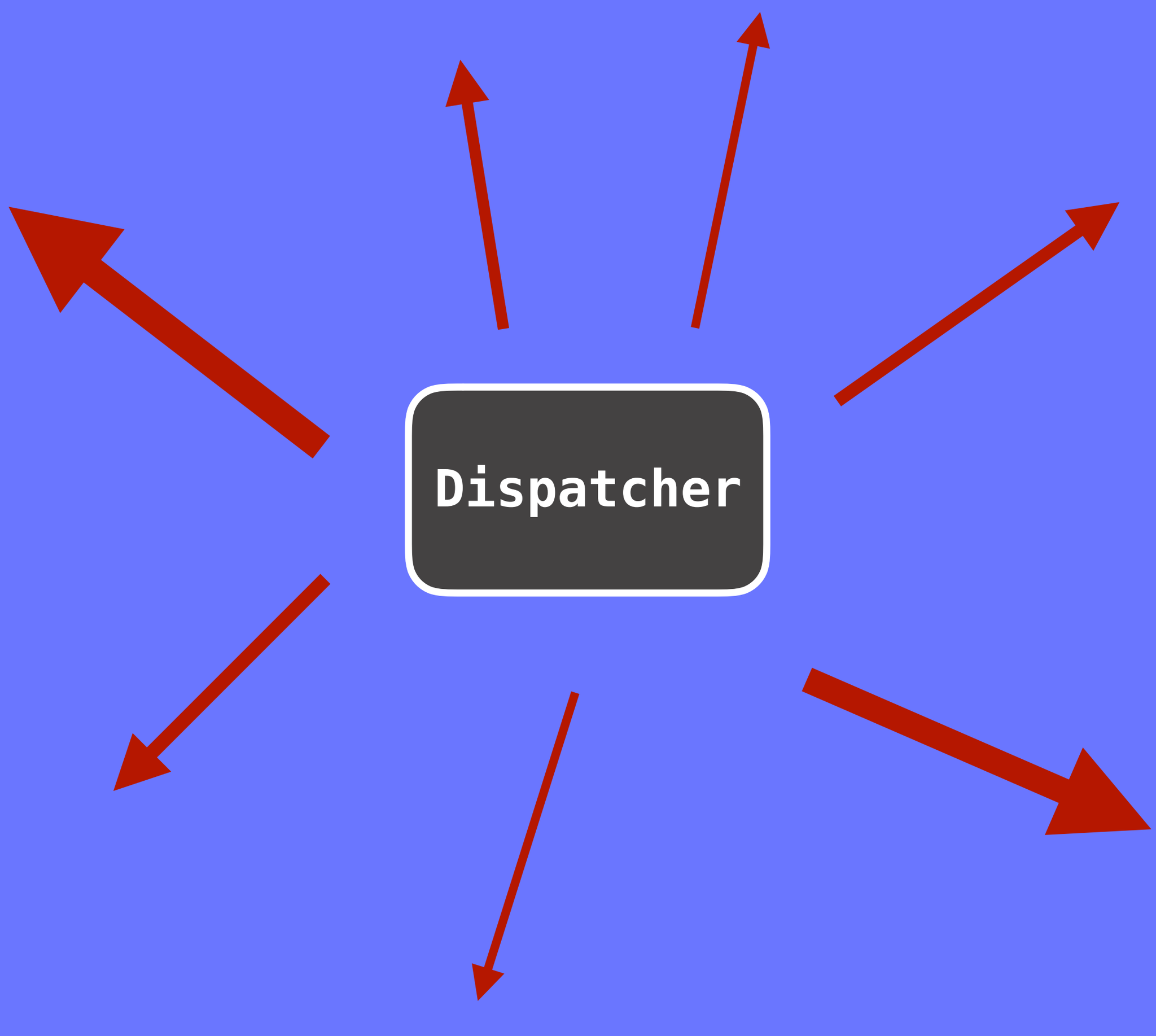
When given action is Loading, then the next loading state is “true”, if the action is Idle, then “false”.

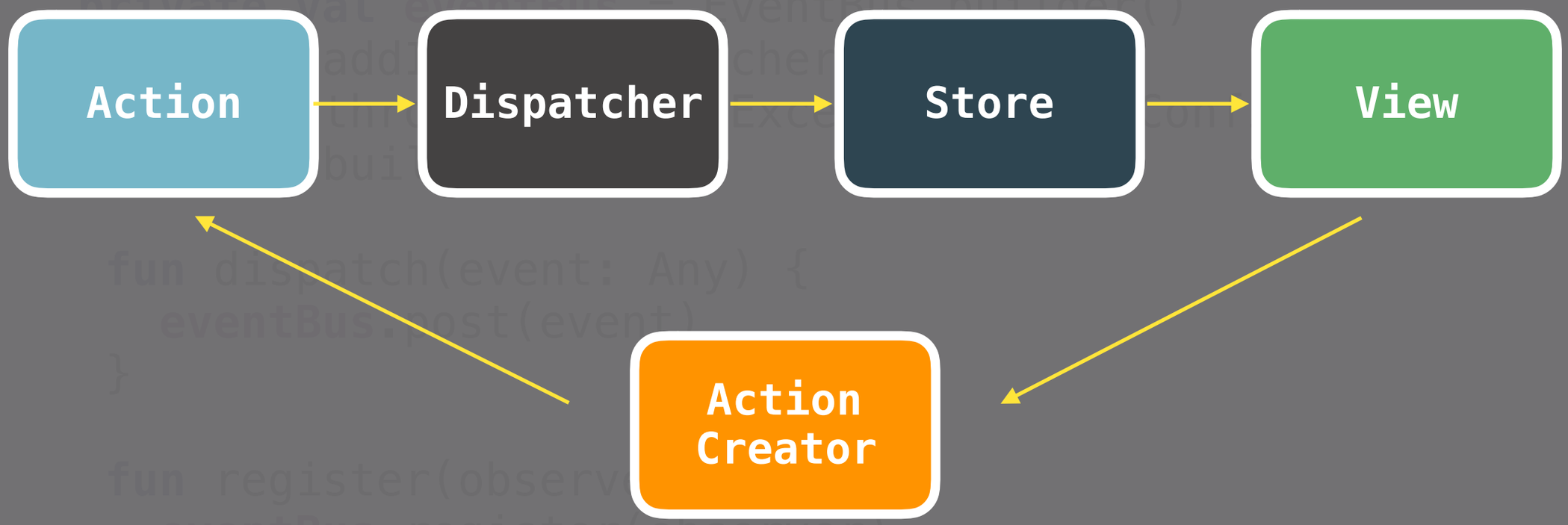
```
@Subscribe(threadMode = ThreadMode.MAIN)
fun on(action: LoadingAction) {
  if (action.activityKey == activityKey) {
    when (action) {
      is LoadingAction.Loading -> {
        loadingSubject.onNext(true)
      }

      is LoadingAction.Idle -> {
        loadingSubject.onNext(false)
      }
  …
```

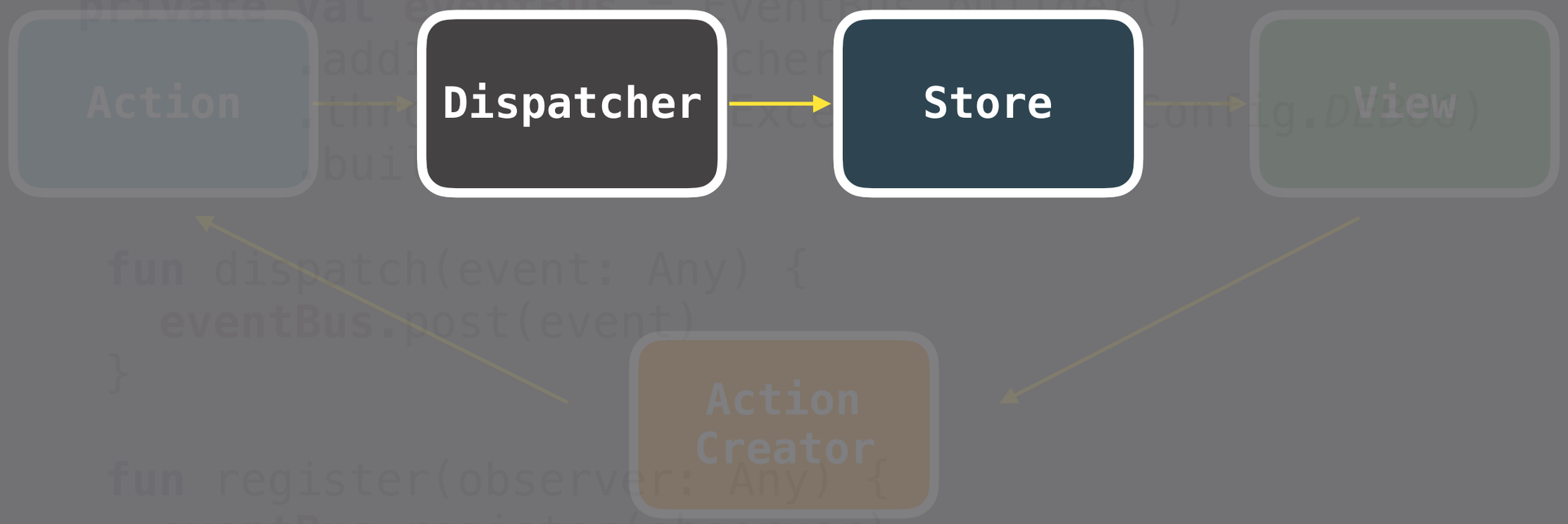
Store

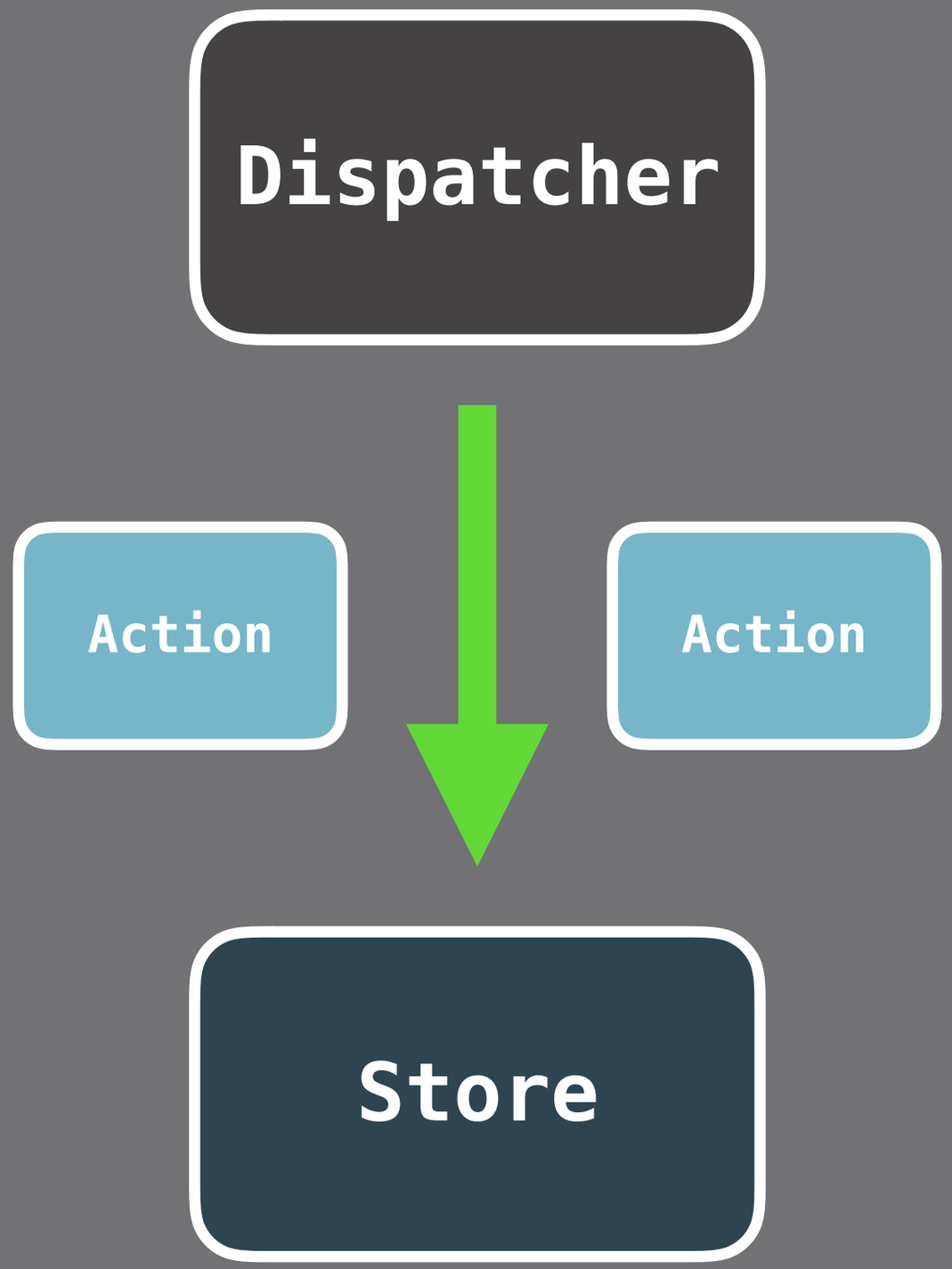
When given action is User's Favorite Action, then find target user from user list and update as favorited (if there is).

```
@Subscribe(threadMode = ThreadMode.MAIN)
fun on(action: UserAction) {
  when (action) {
    is UserAction.Favorite -> {
      userList.find { it.id == action.targetUserId }?.let {
        // Update or replace with new user
      }
    }
}
…
```


Dispatcher


Action
Dispatcher
Store
View
Action
Creator


Action
Dispatcher
Store
View
Action Creator


Dispatcher
Action
Action
Store

Action Creator

Store

Store

The ONLY

state manager

Summary

Flux Summary

- Flux is for utilizing unidirectional data flow by declaring the components
- Store is the only state manager
- Makes your code DRY and consistent with minimal boilerplates

Flux for Android

Thank you!

@shaunkawano