reacting to code sprawl

Reactive Workflows

Ray Ryan

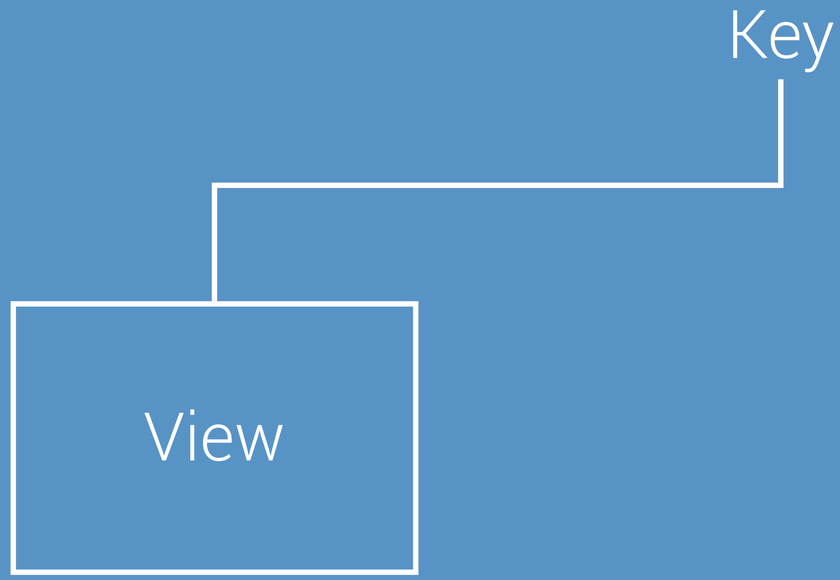
the legacy

Key-based MVP

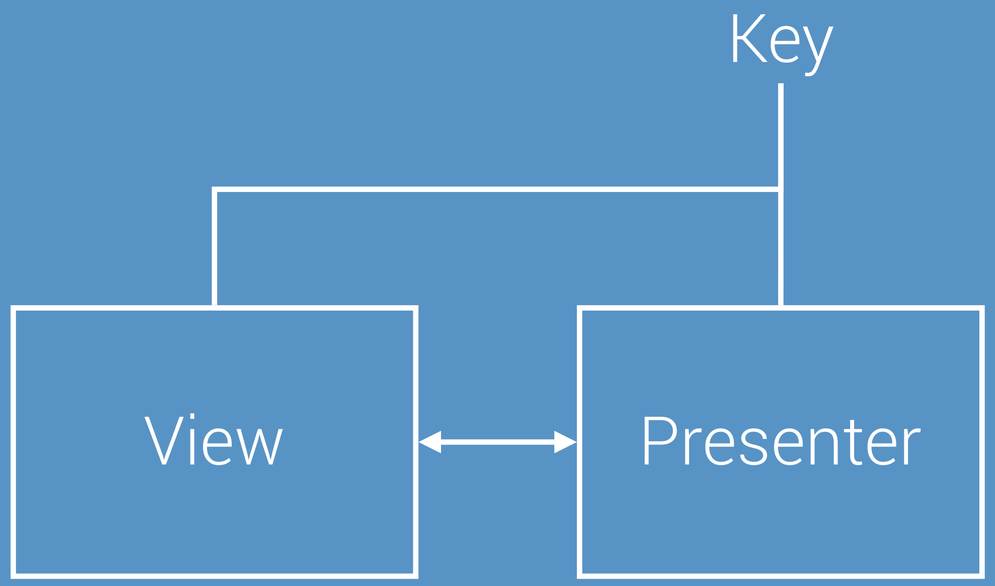
Key-based MVP

flow.set(Key)

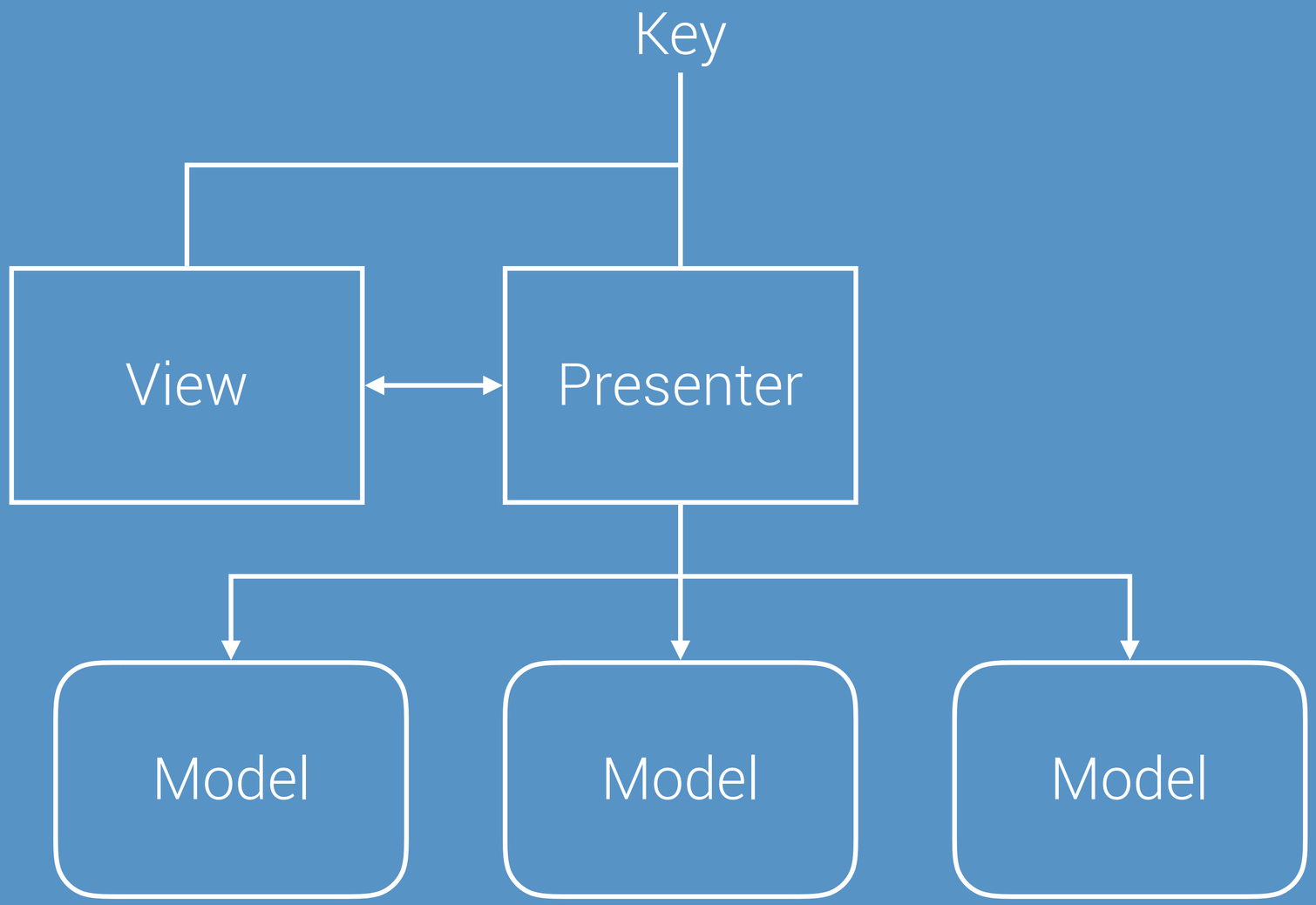
Key-based MVP



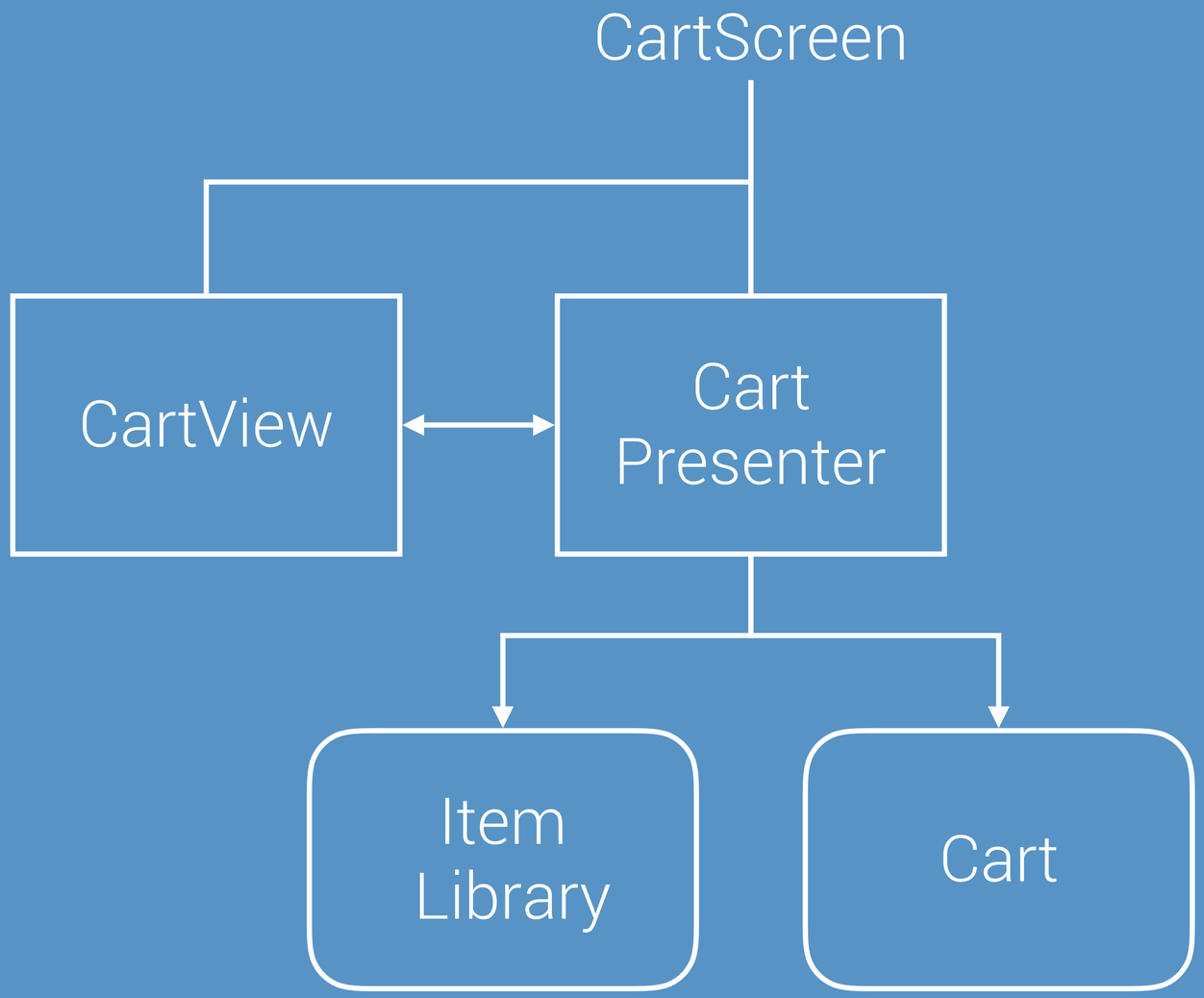
Key-based MVP



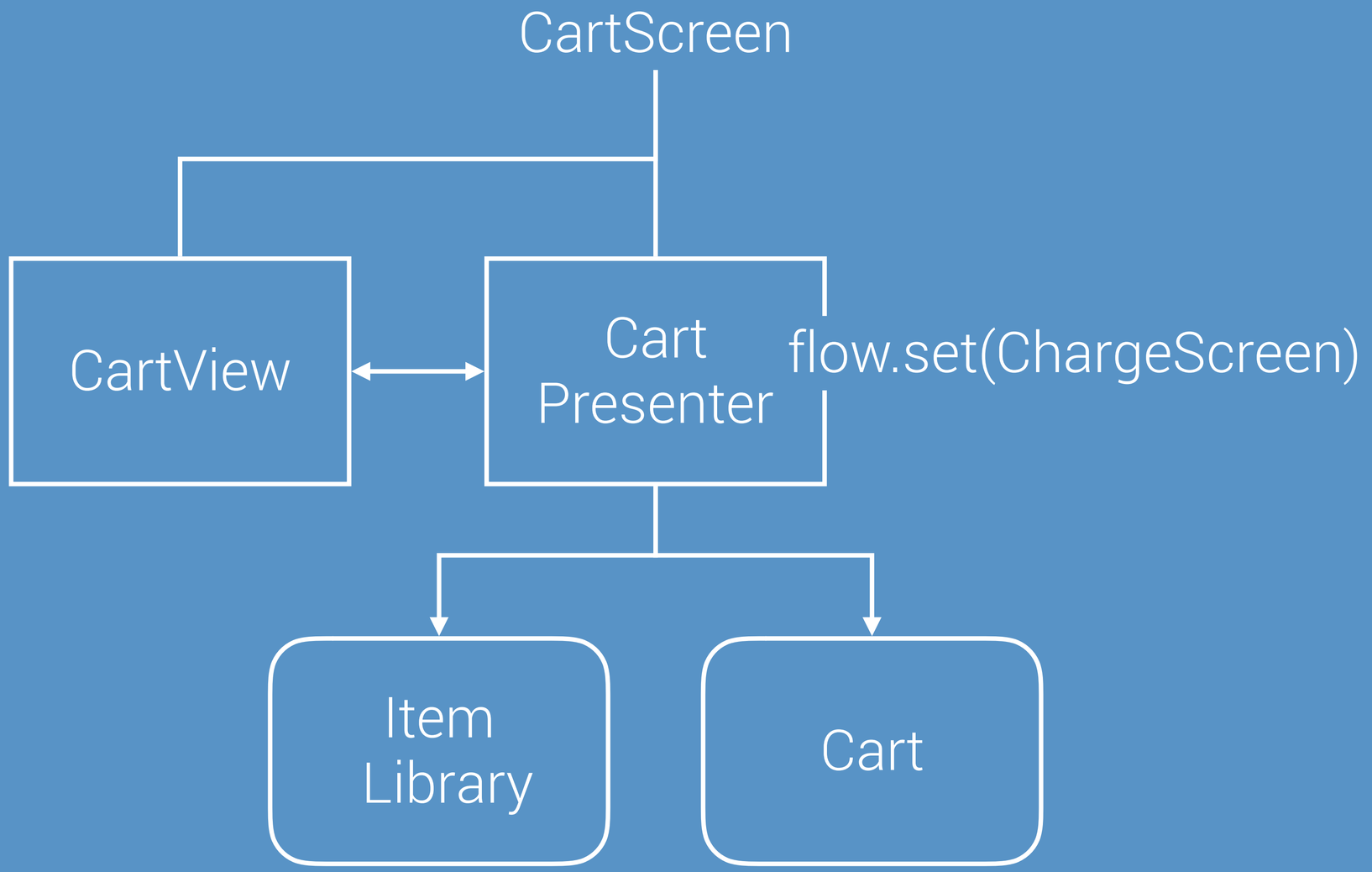
Key-based MVP



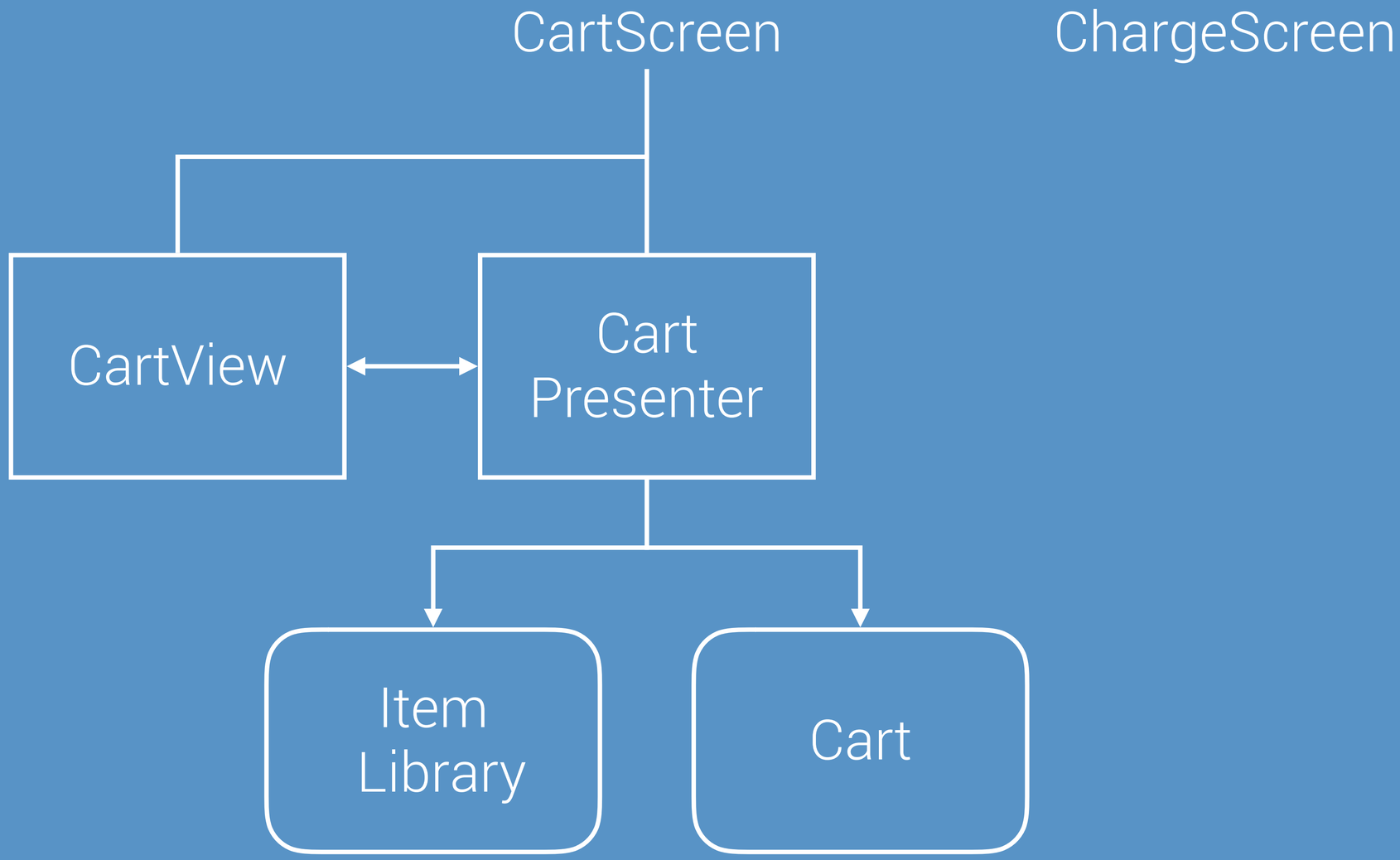
Key-based MVP



Key-based MVP



Key-based MVP

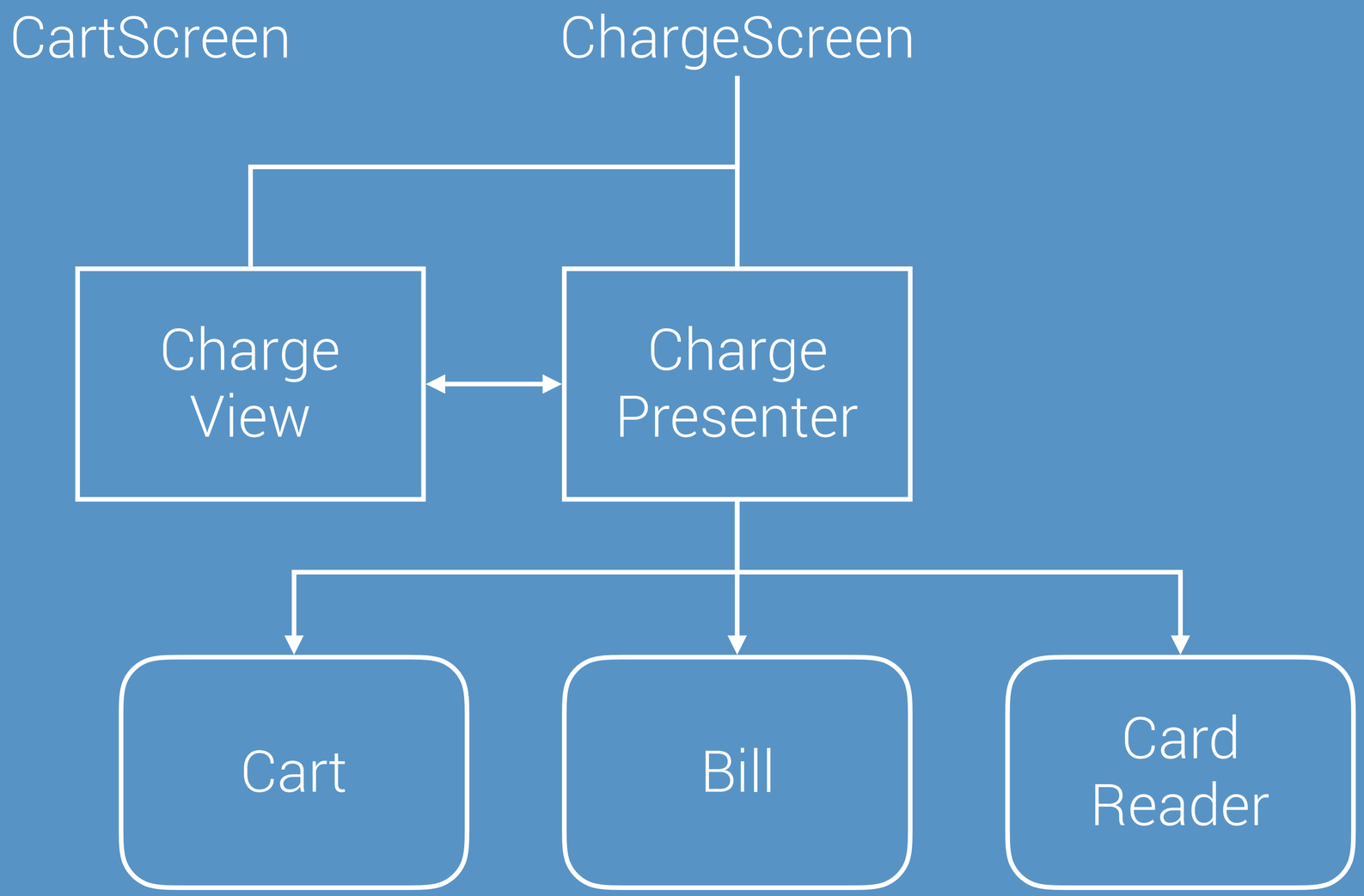


Key-based MVP

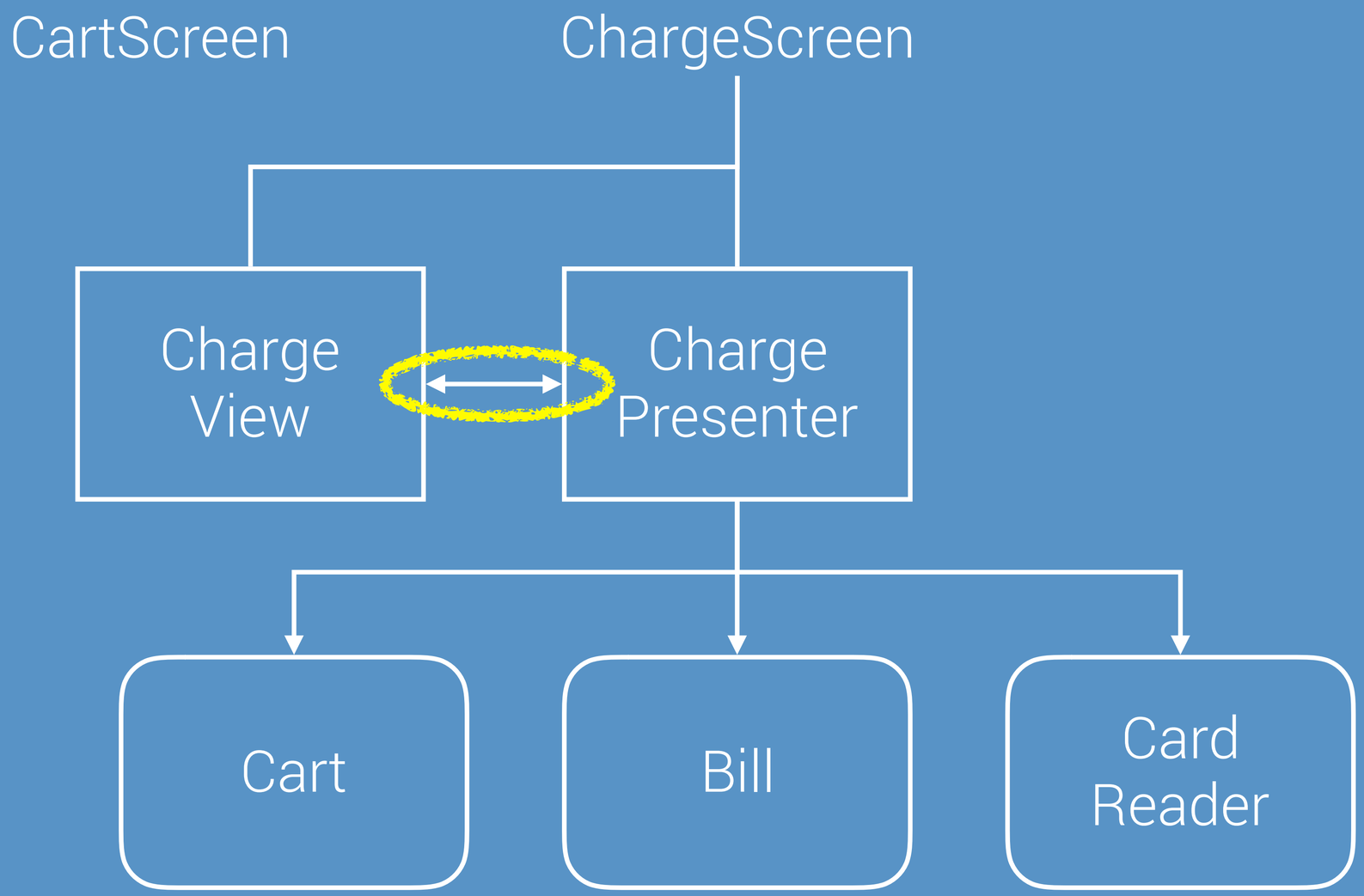
CartScreen

ChargeScreen

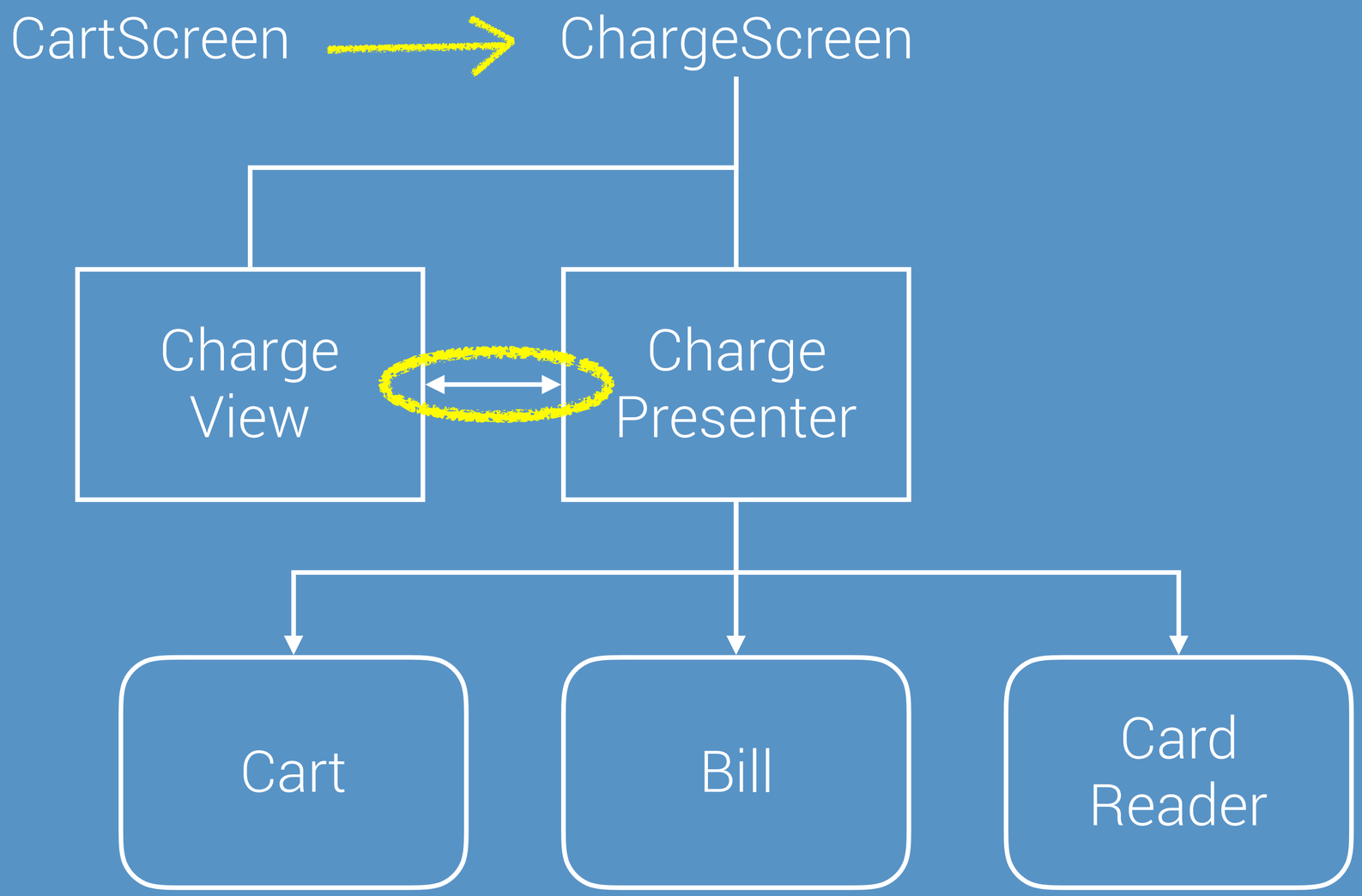
Key-based MVP



Key-based MVP



Key-based MVP

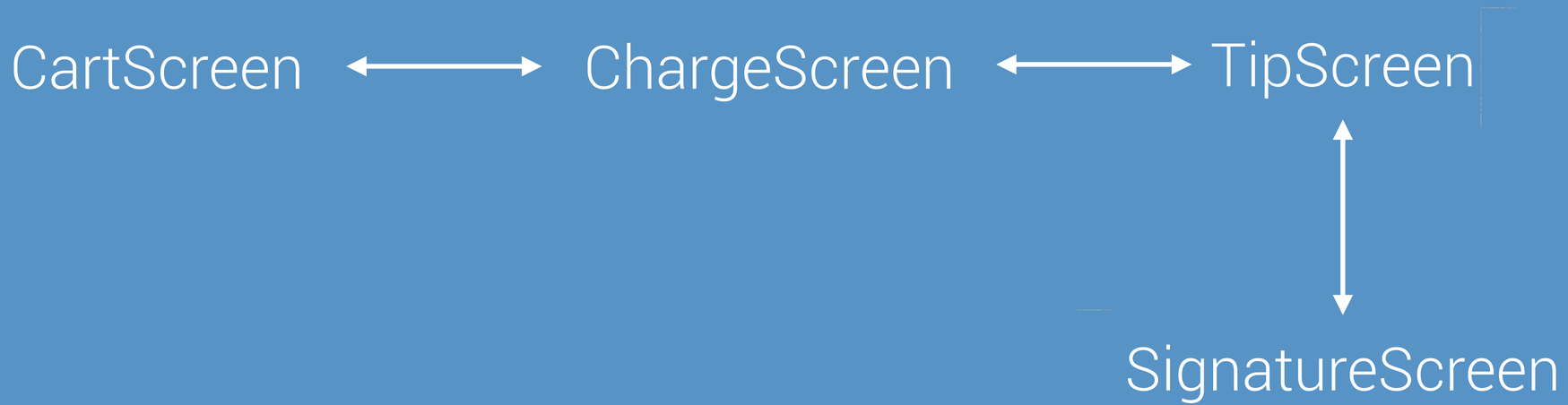


Nav logic sprawl

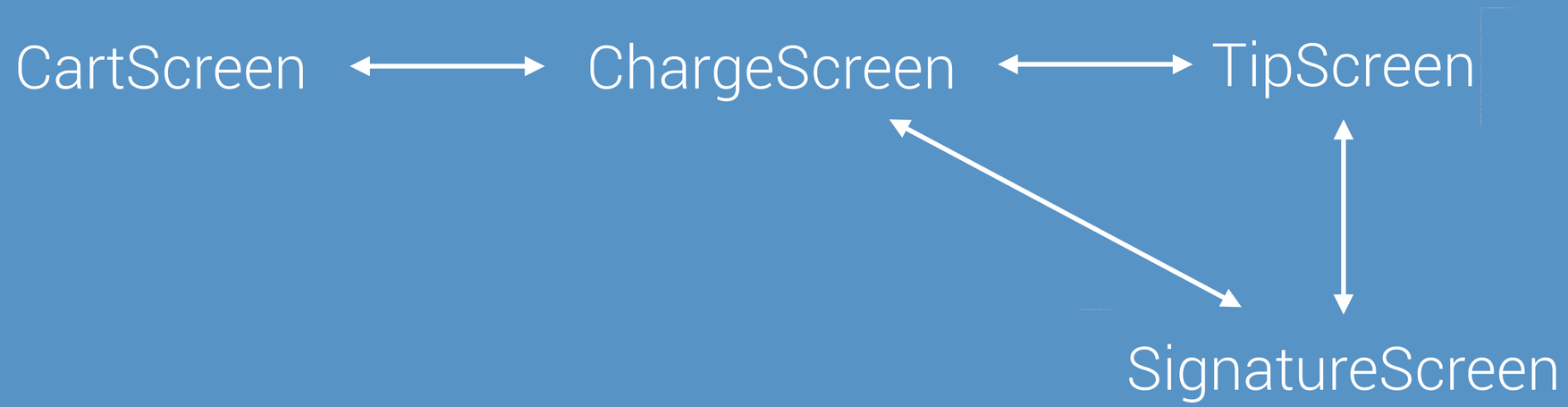
CartScreen ⟷ ChargeScreen

Nav logic sprawl

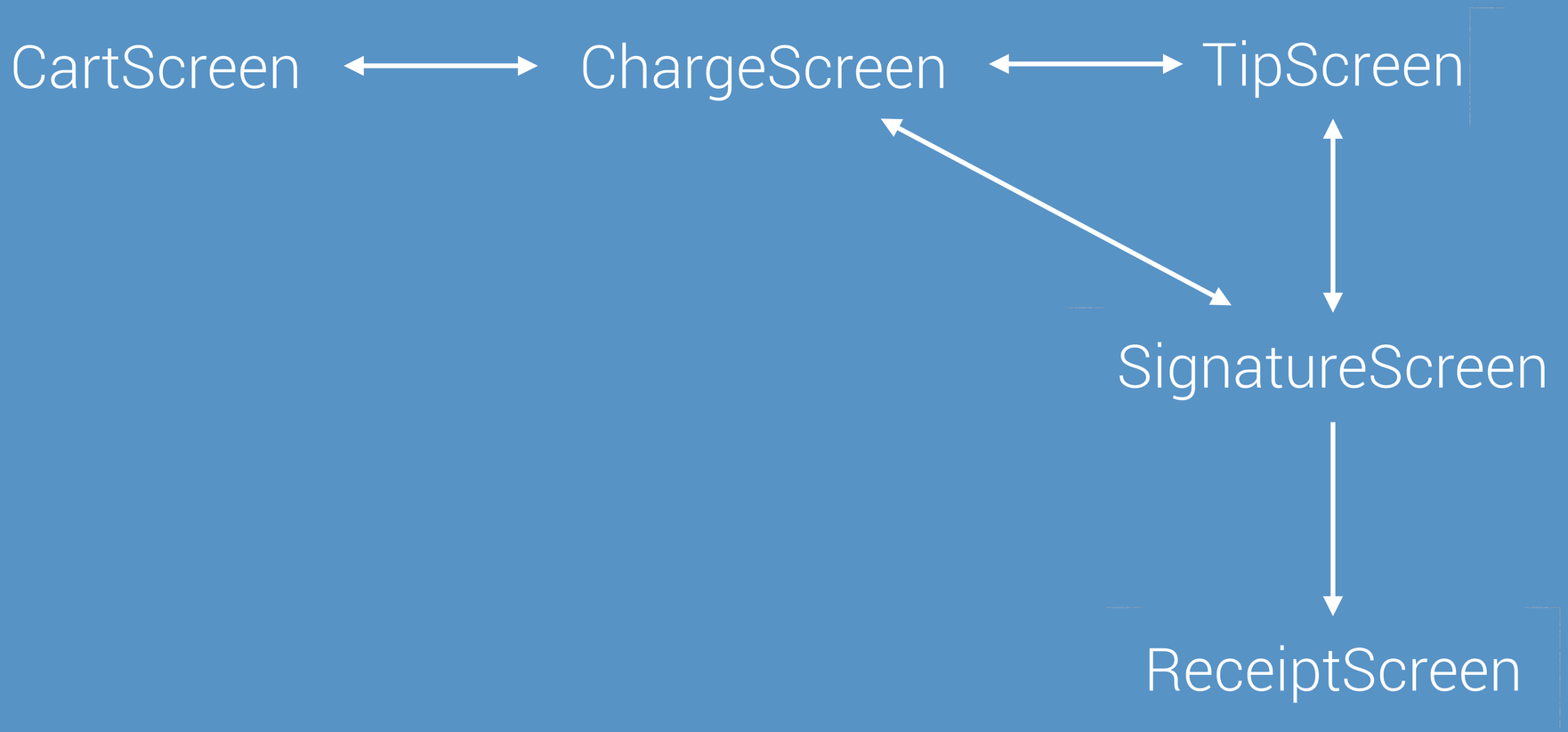
Nav logic sprawl



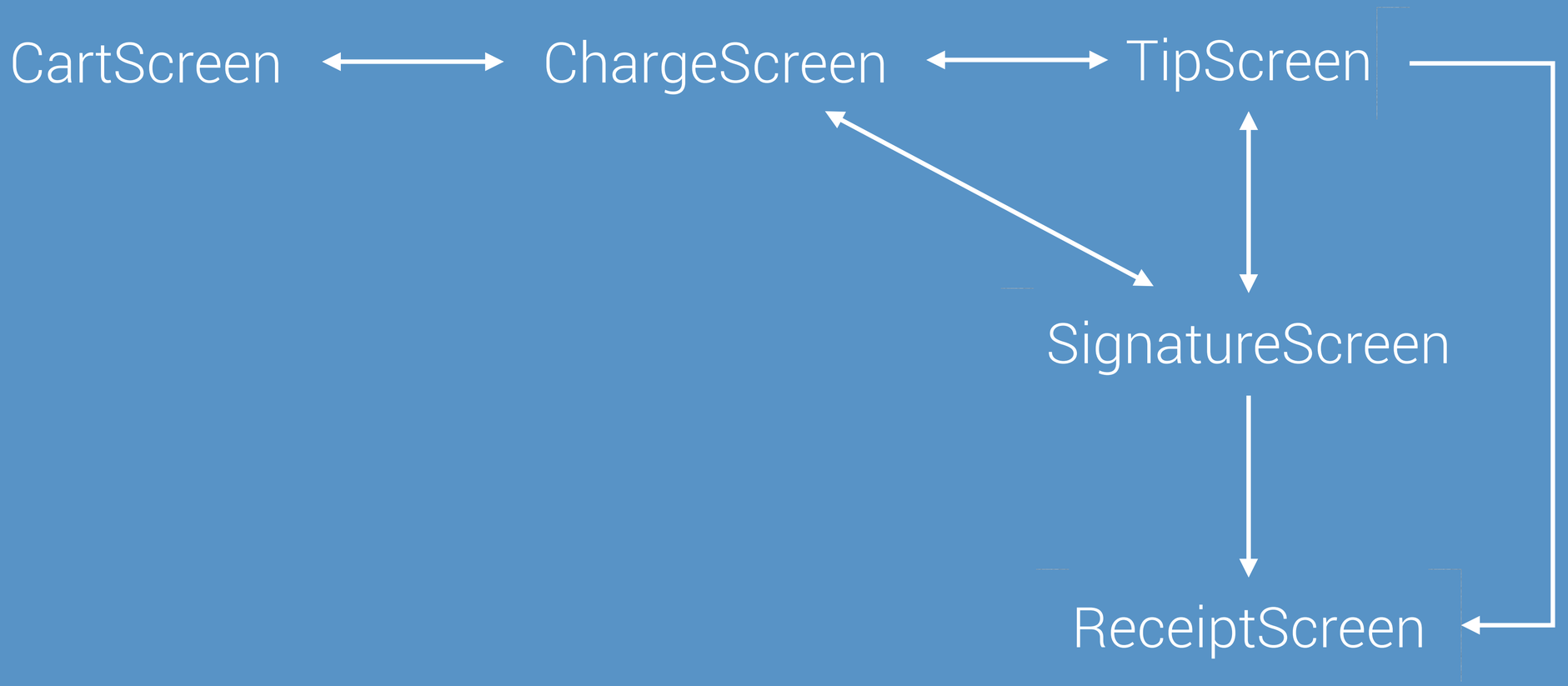
Nav logic sprawl



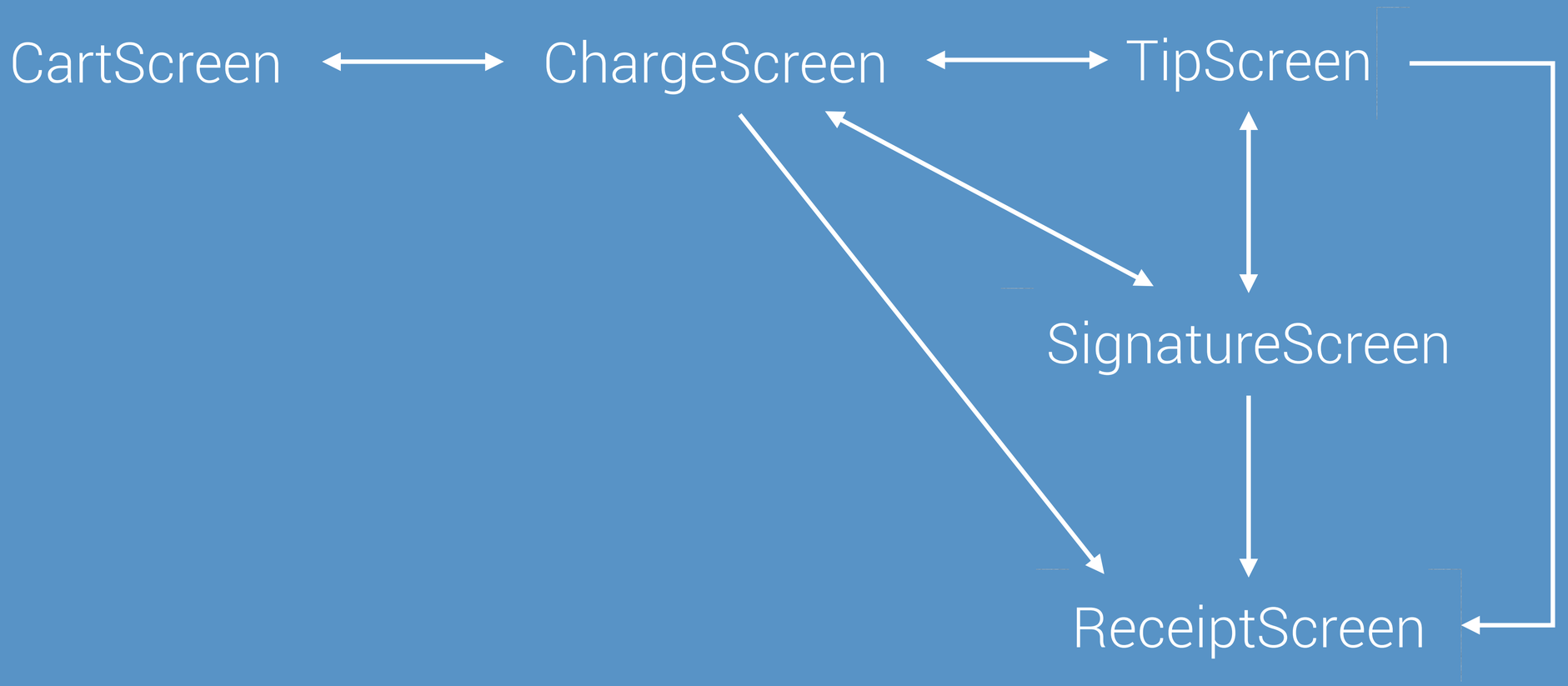
Nav logic sprawl



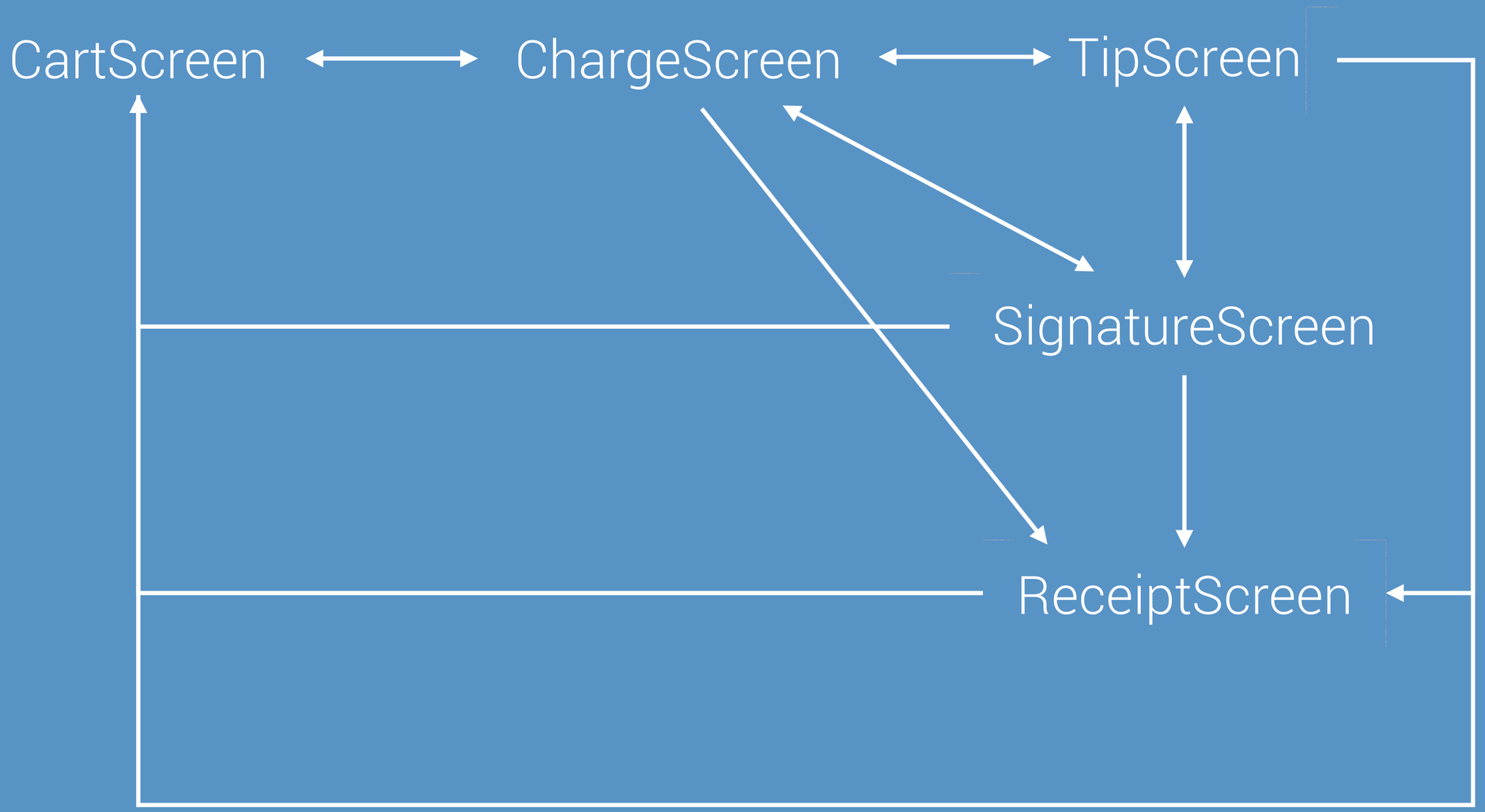
Nav logic sprawl



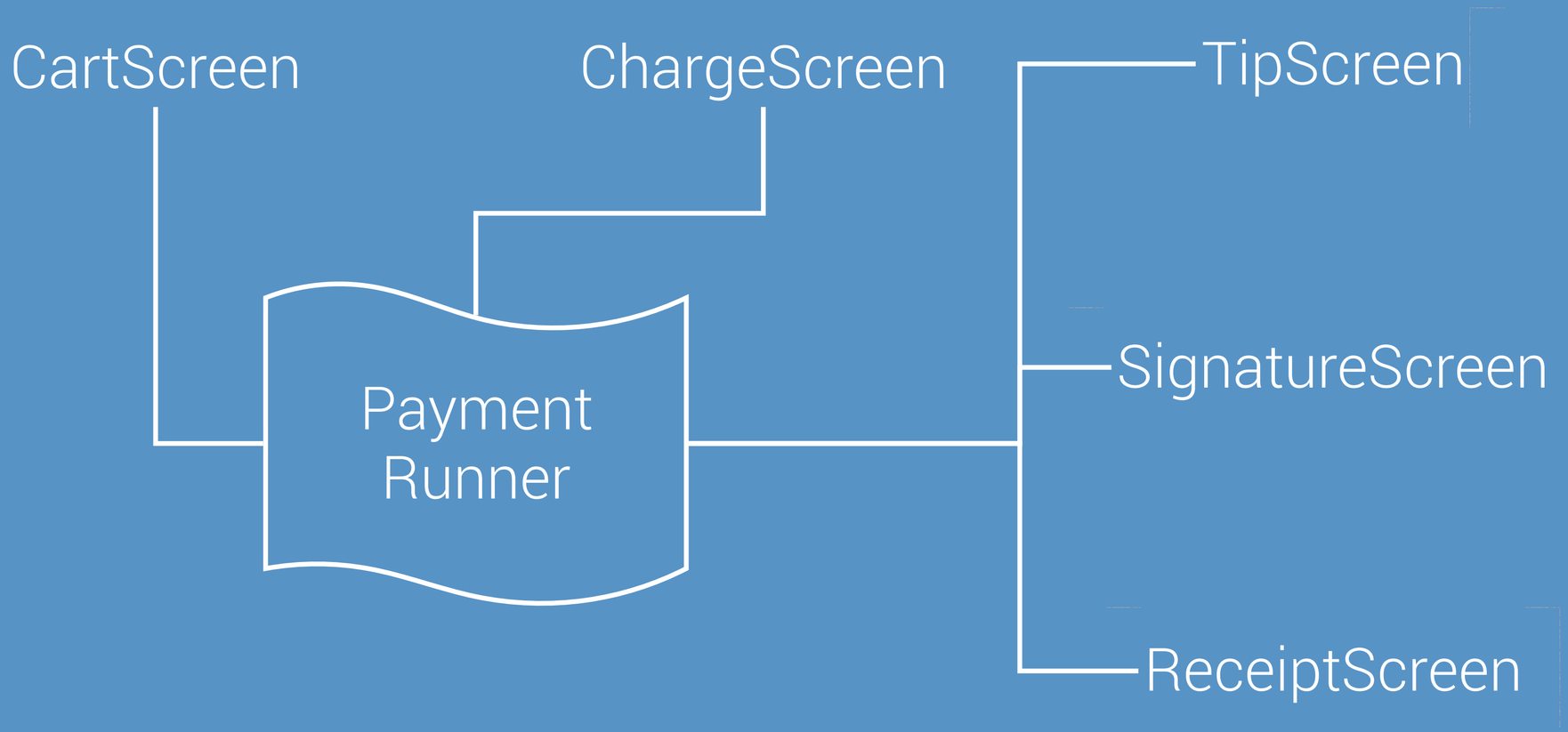
Nav logic sprawl



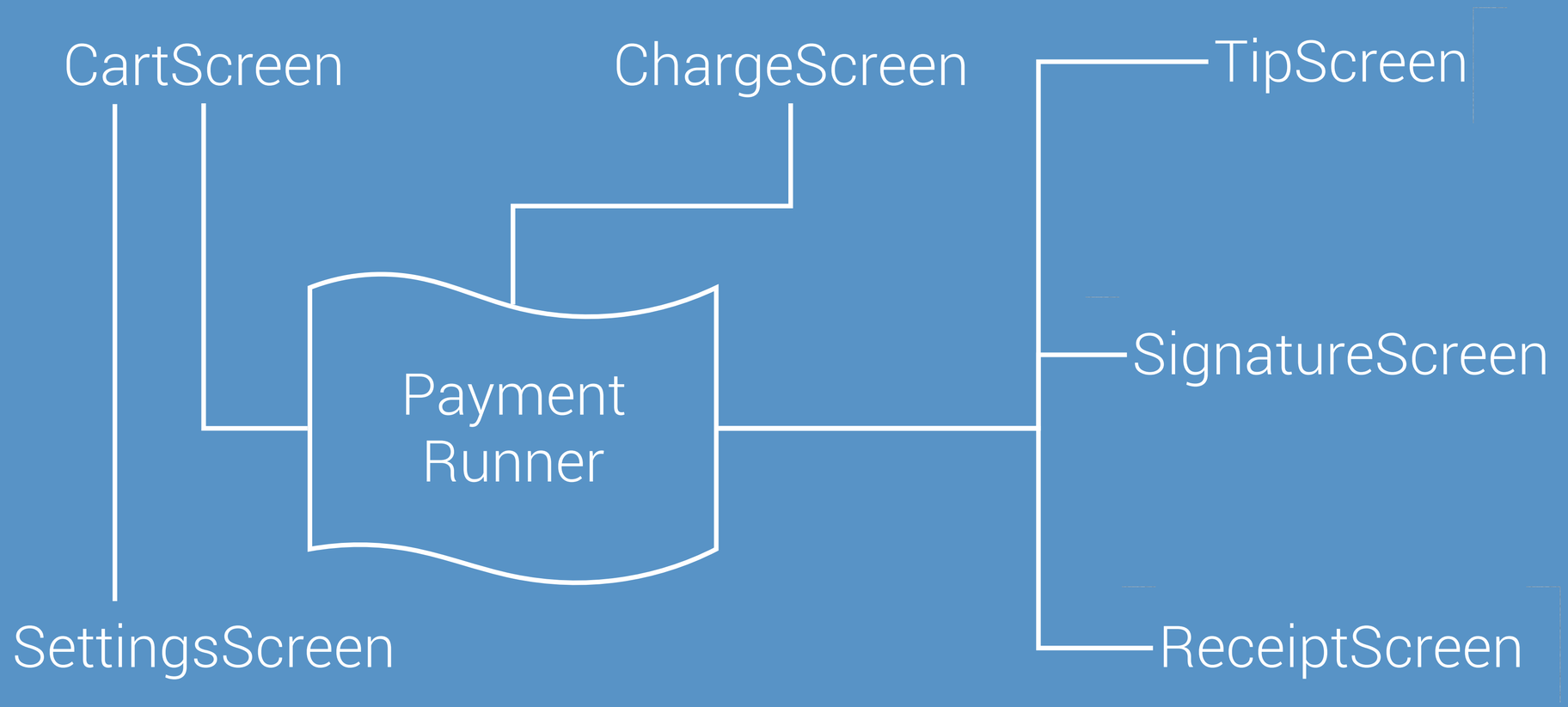
Nav logic sprawl



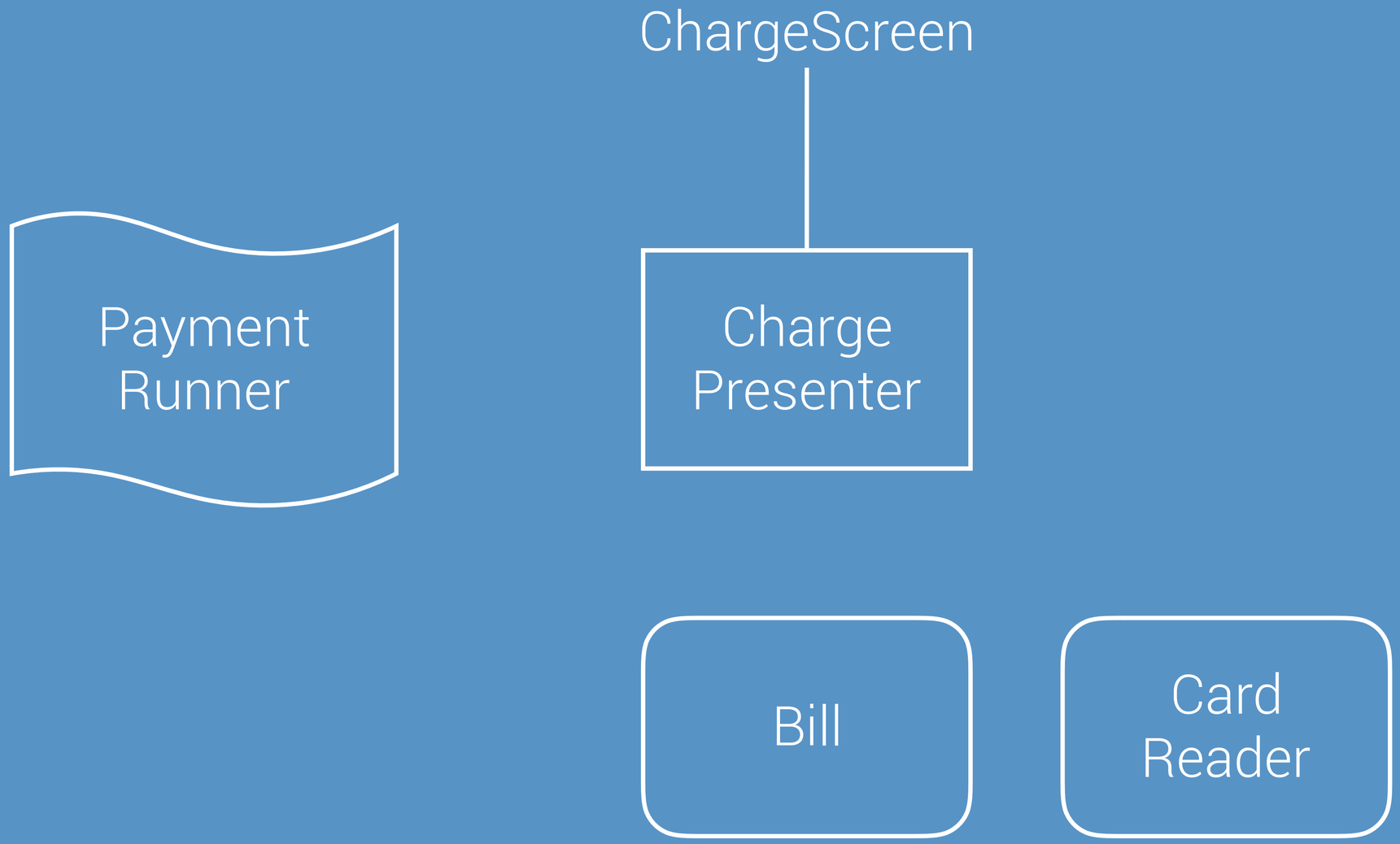
Nav logic sprawl



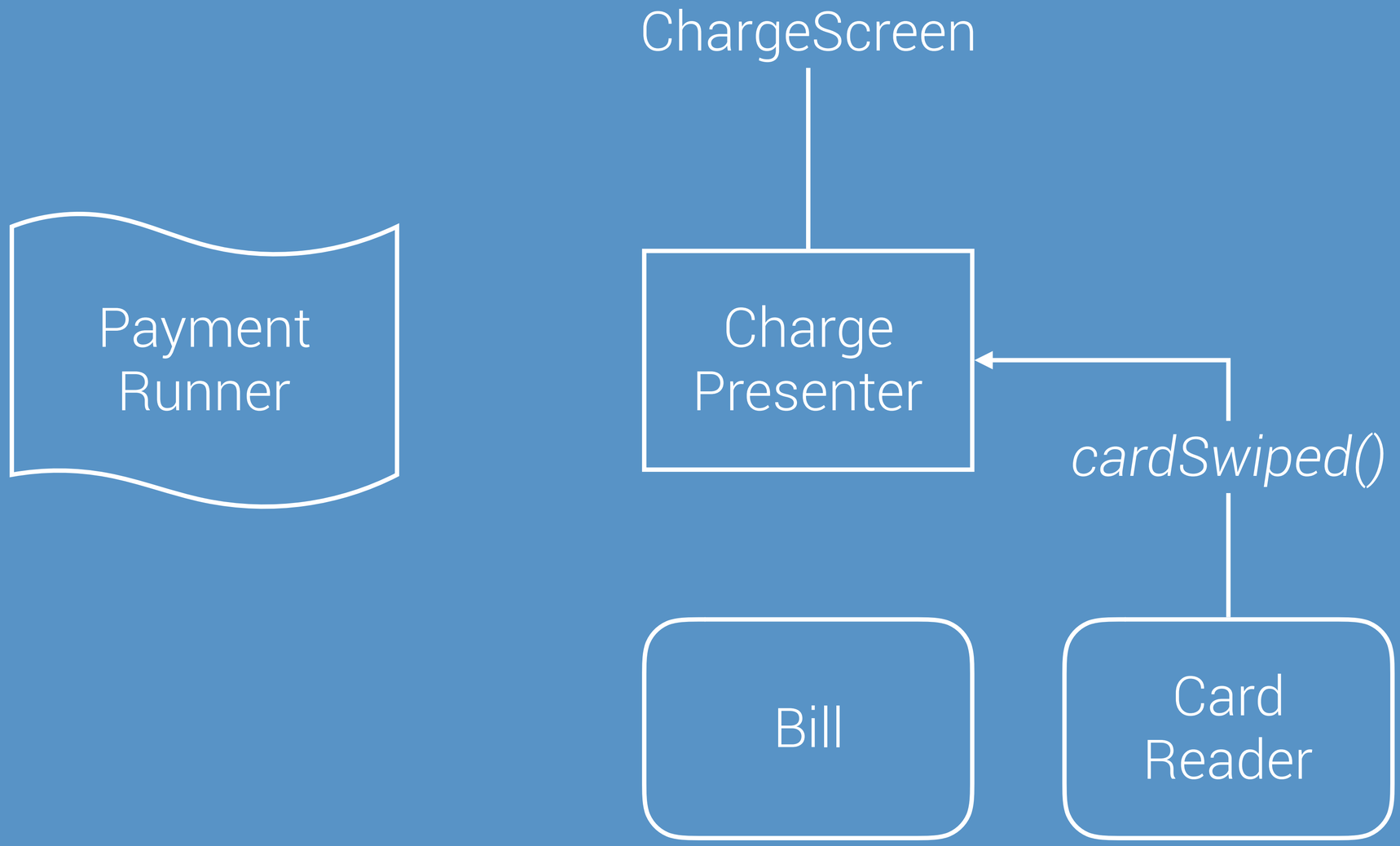
Nav logic sprawl



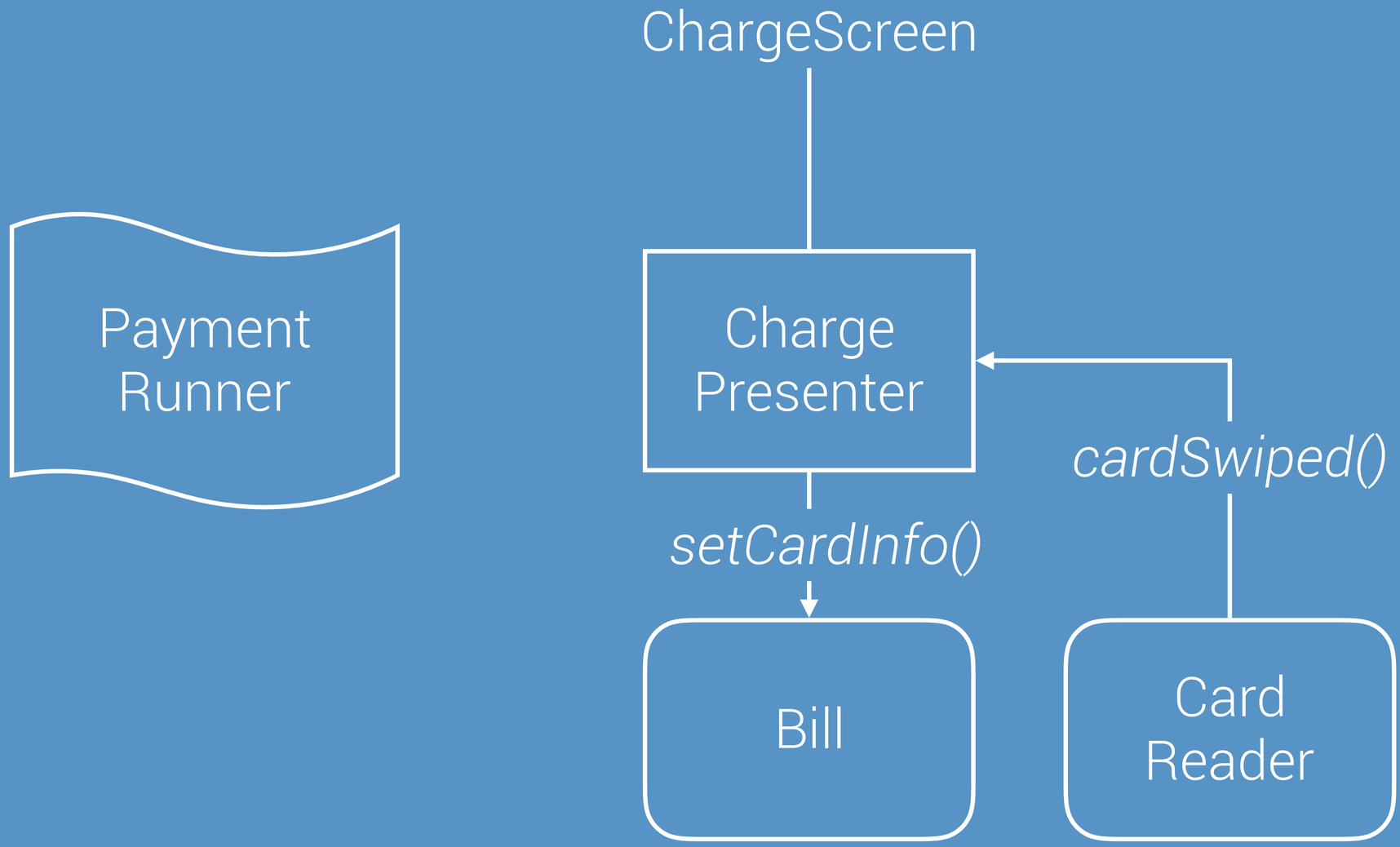
Nav & biz logic entangled



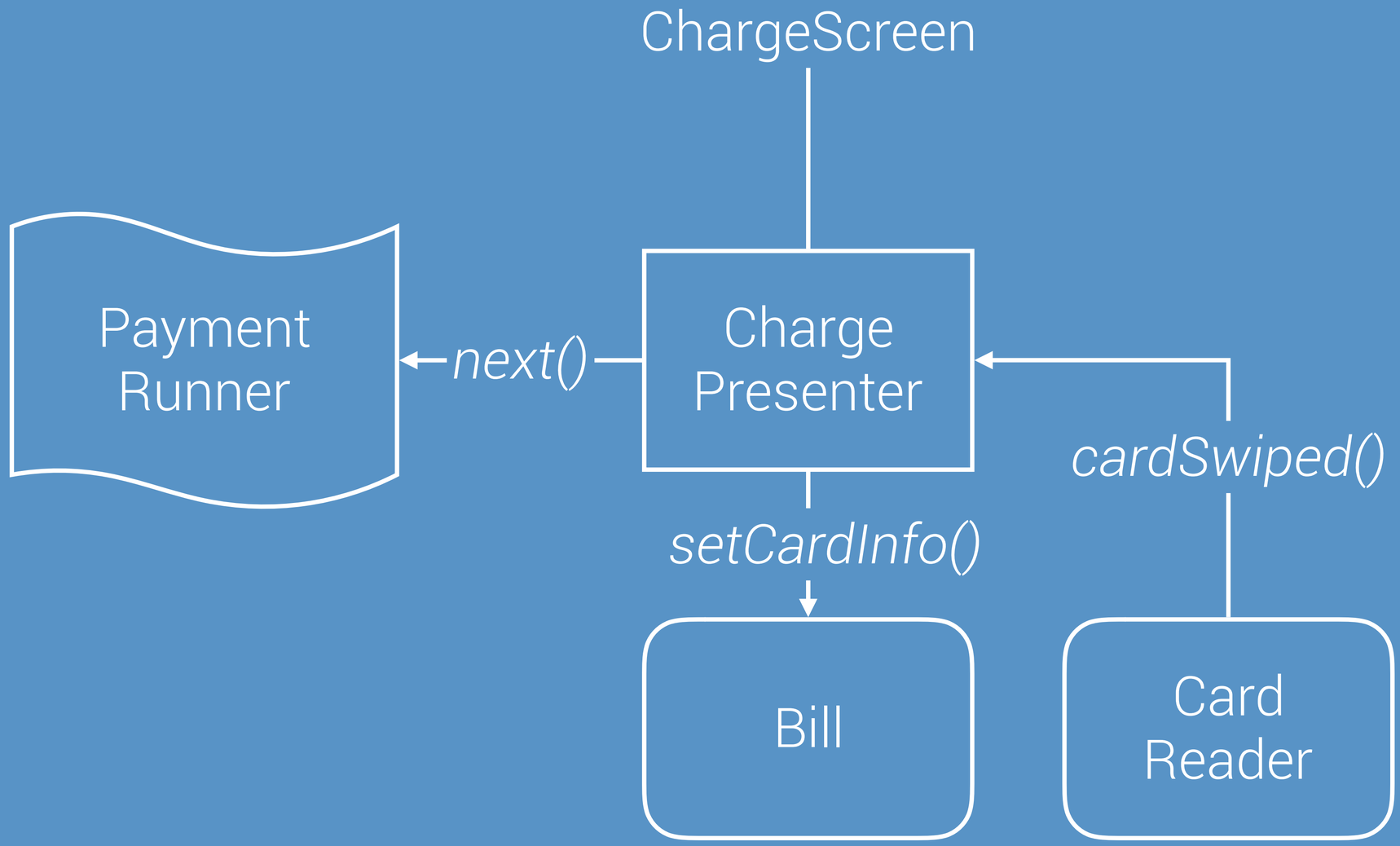
Nav & biz logic entangled



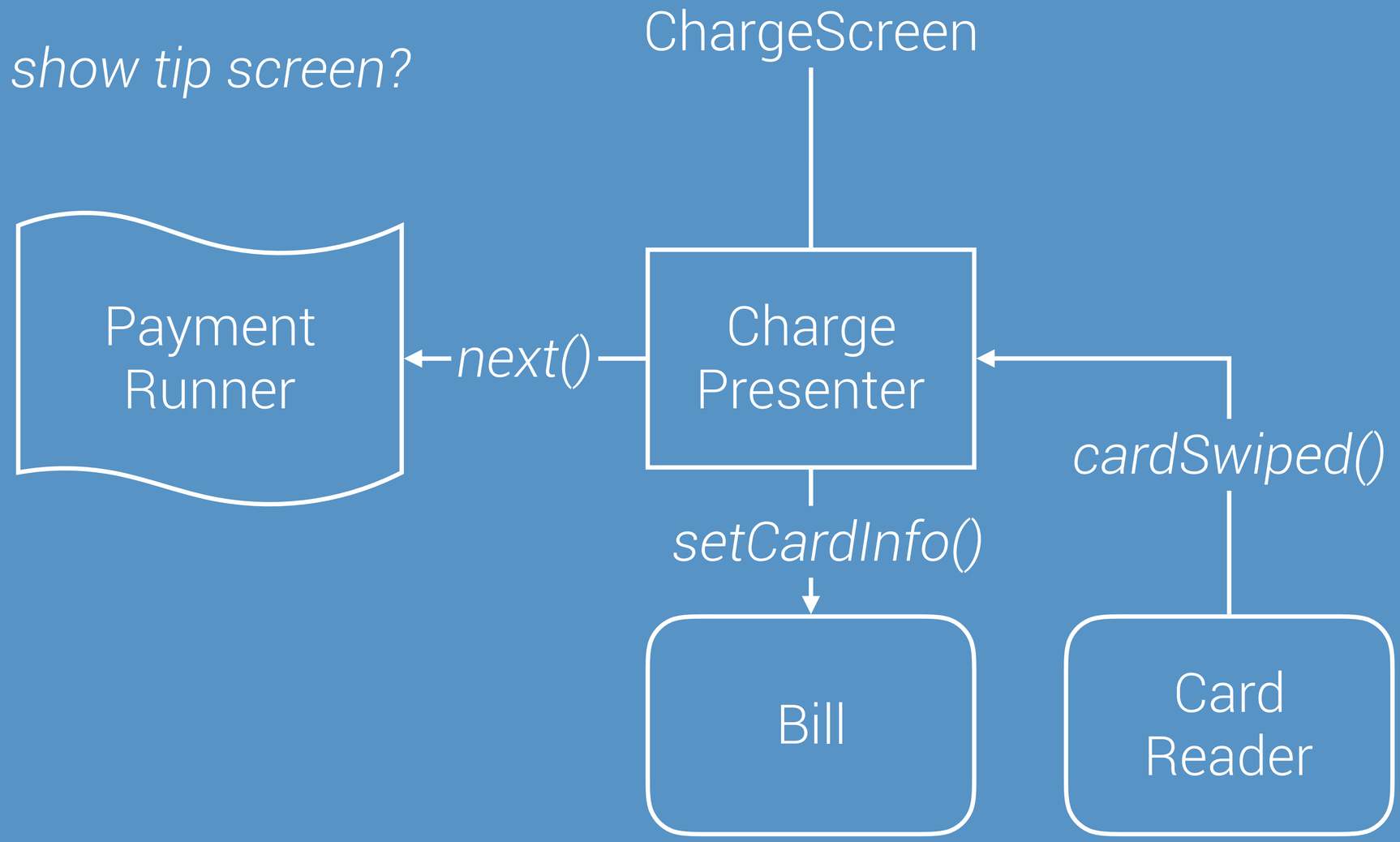
Nav & biz logic entangled



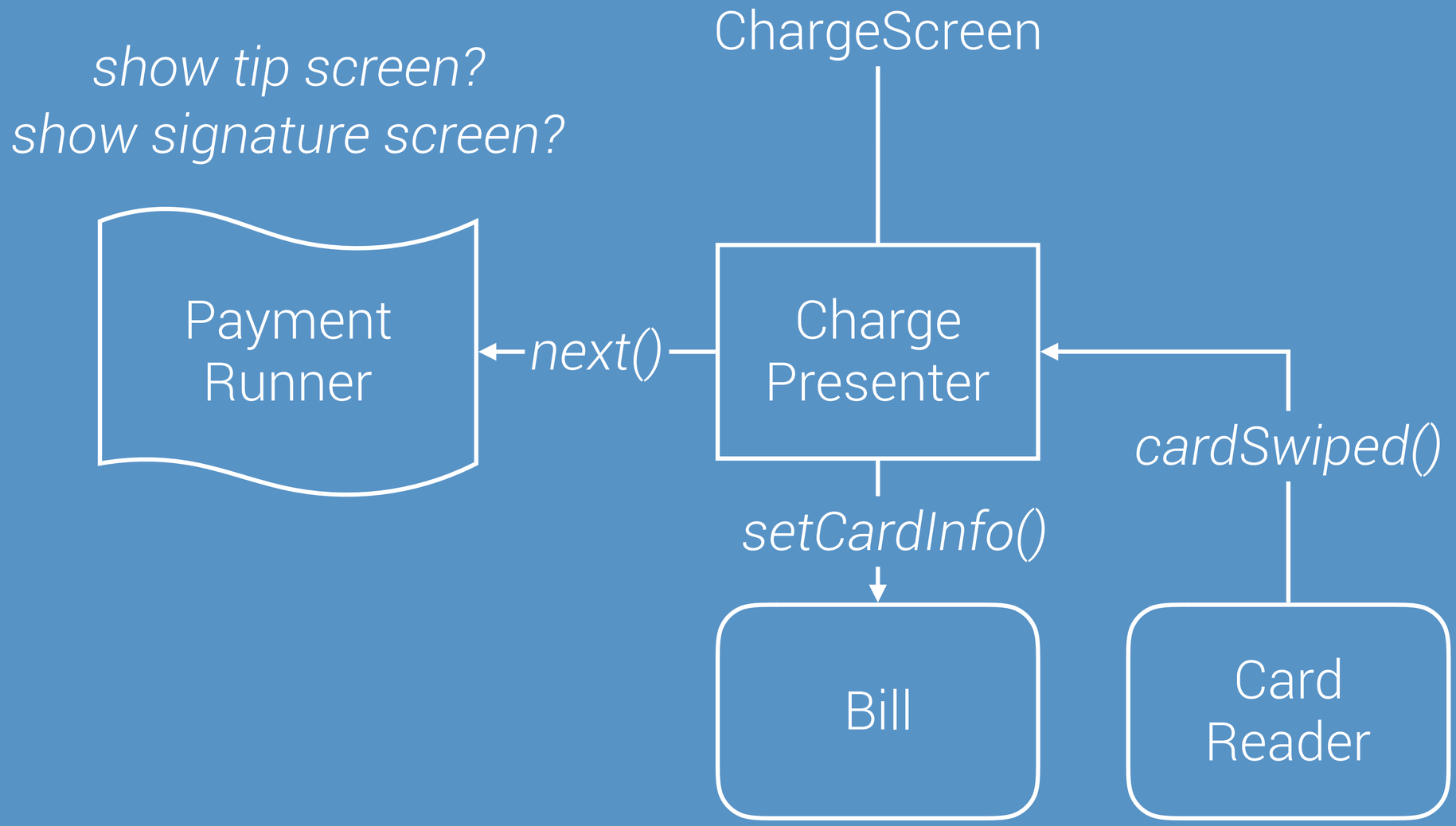
Nav & biz logic entangled



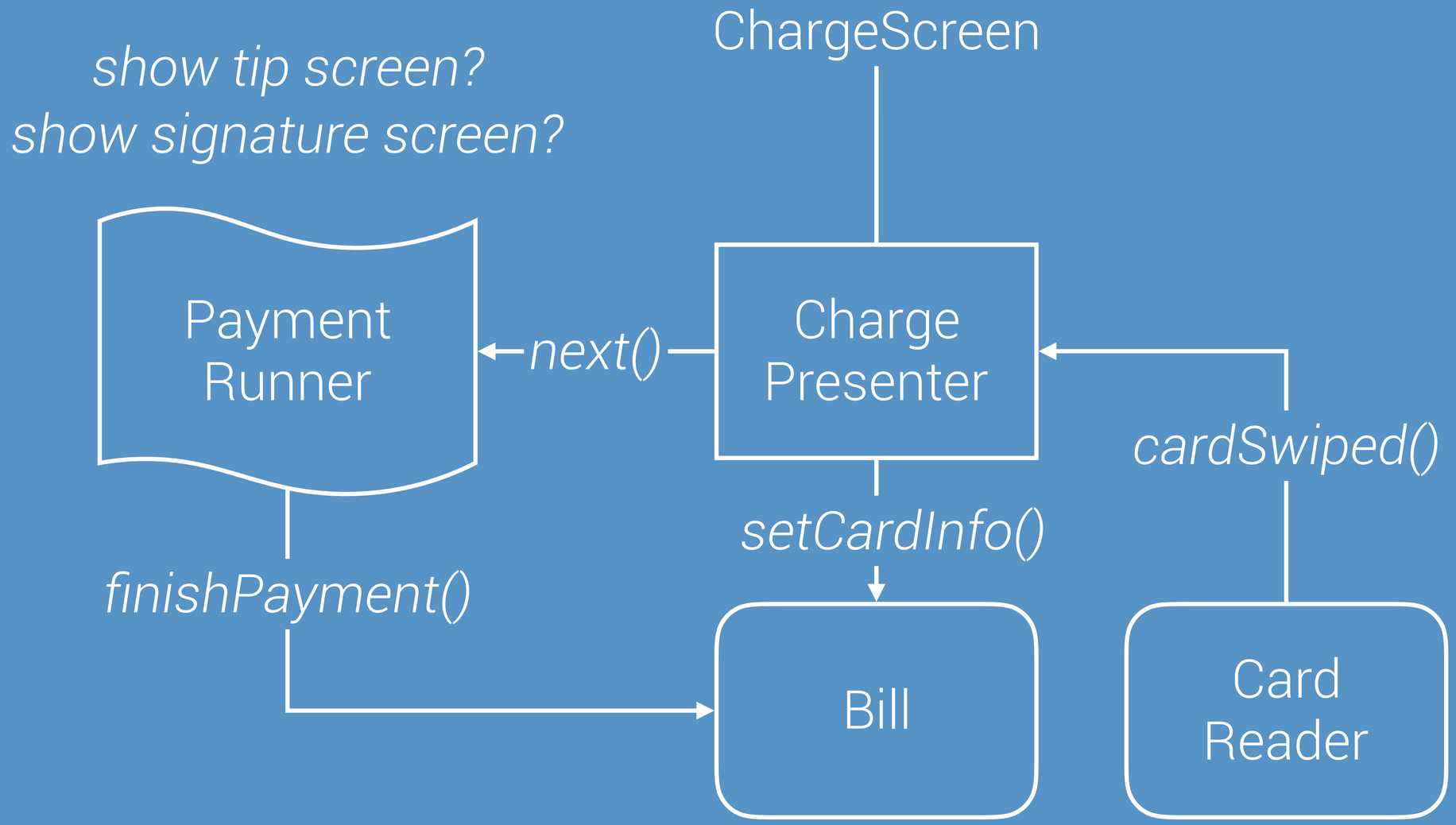
Nav & biz logic entangled



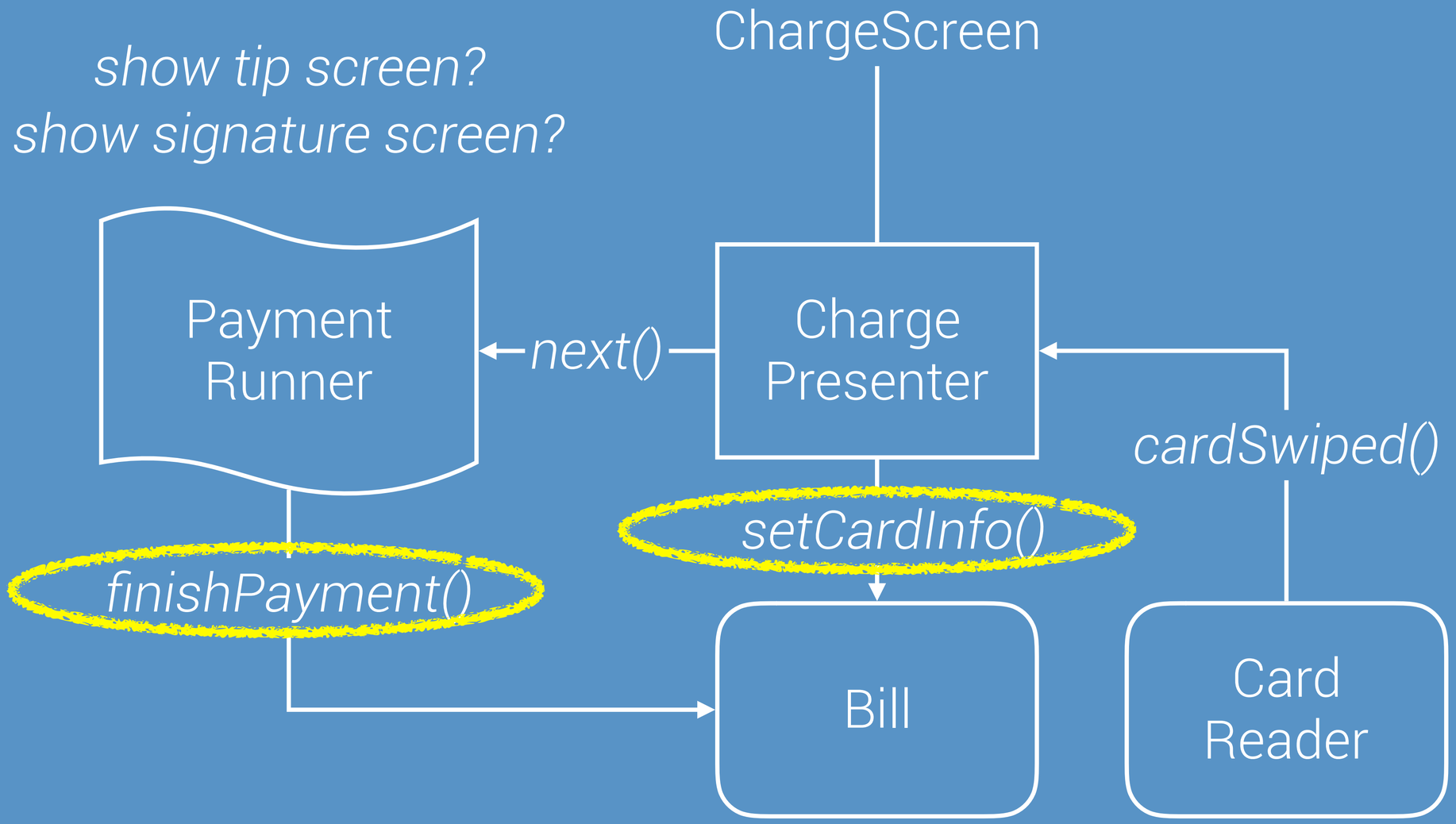
Nav & biz logic entangled



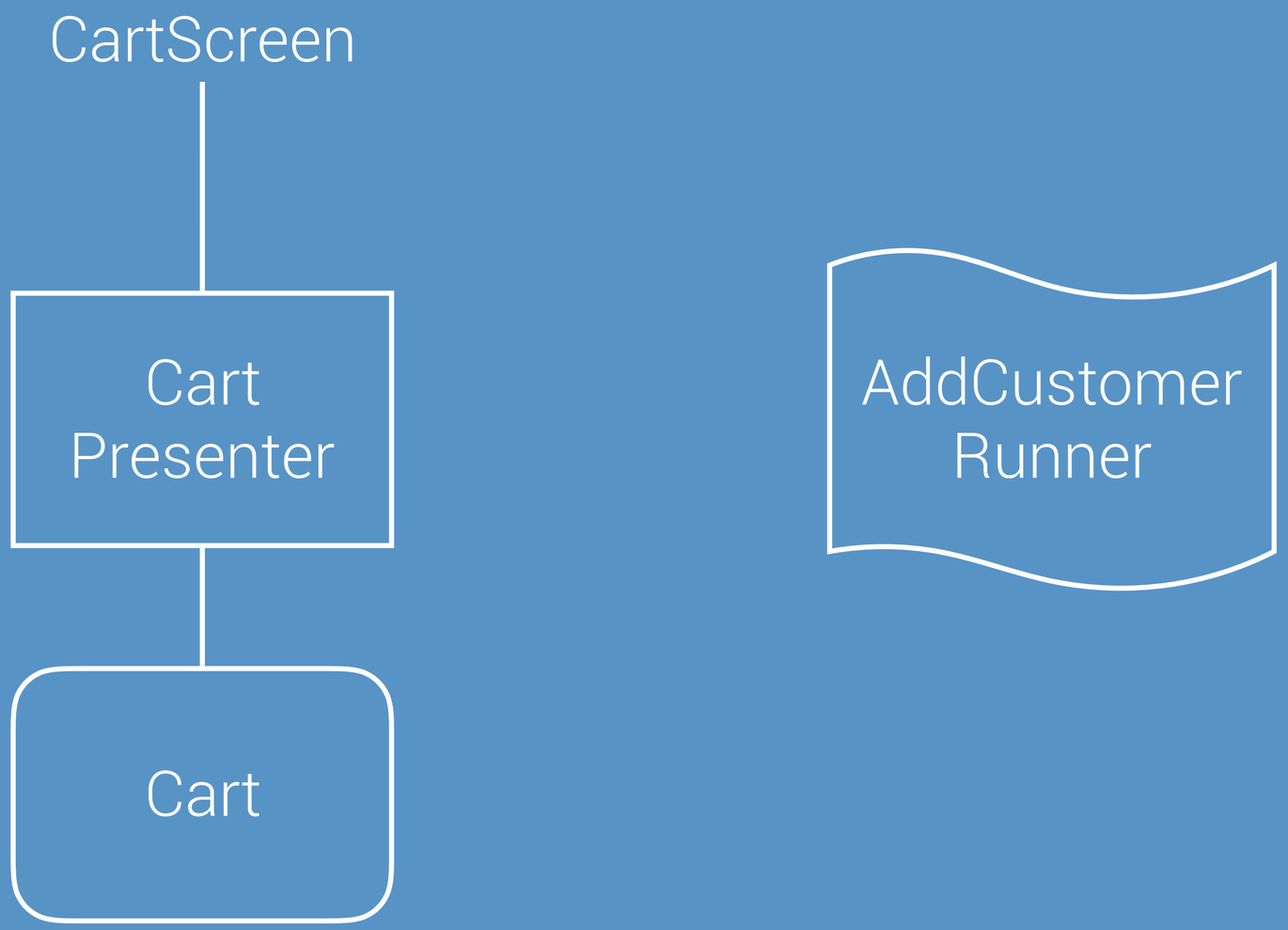
Nav & biz logic entangled



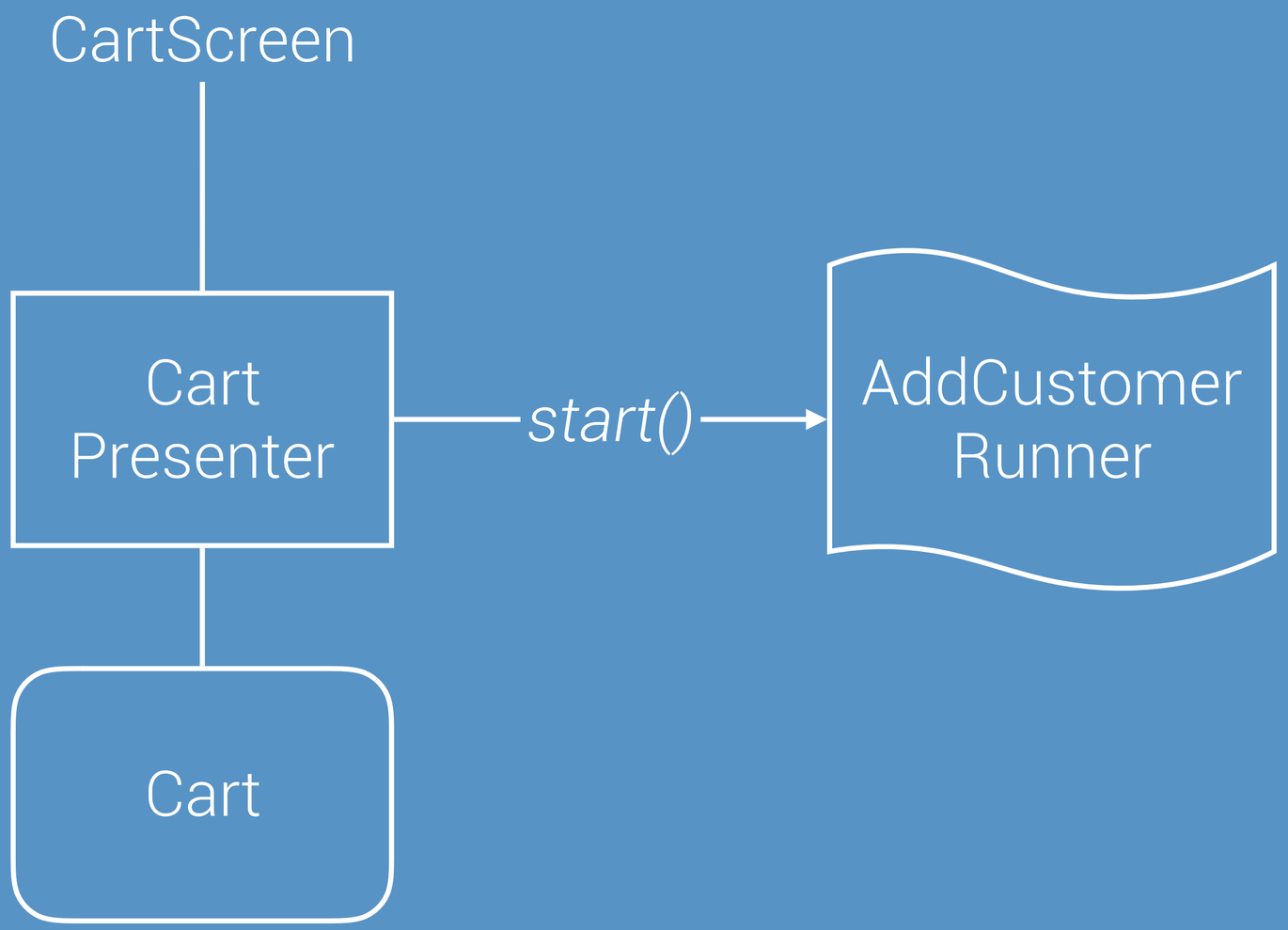
Nav & biz logic entangled



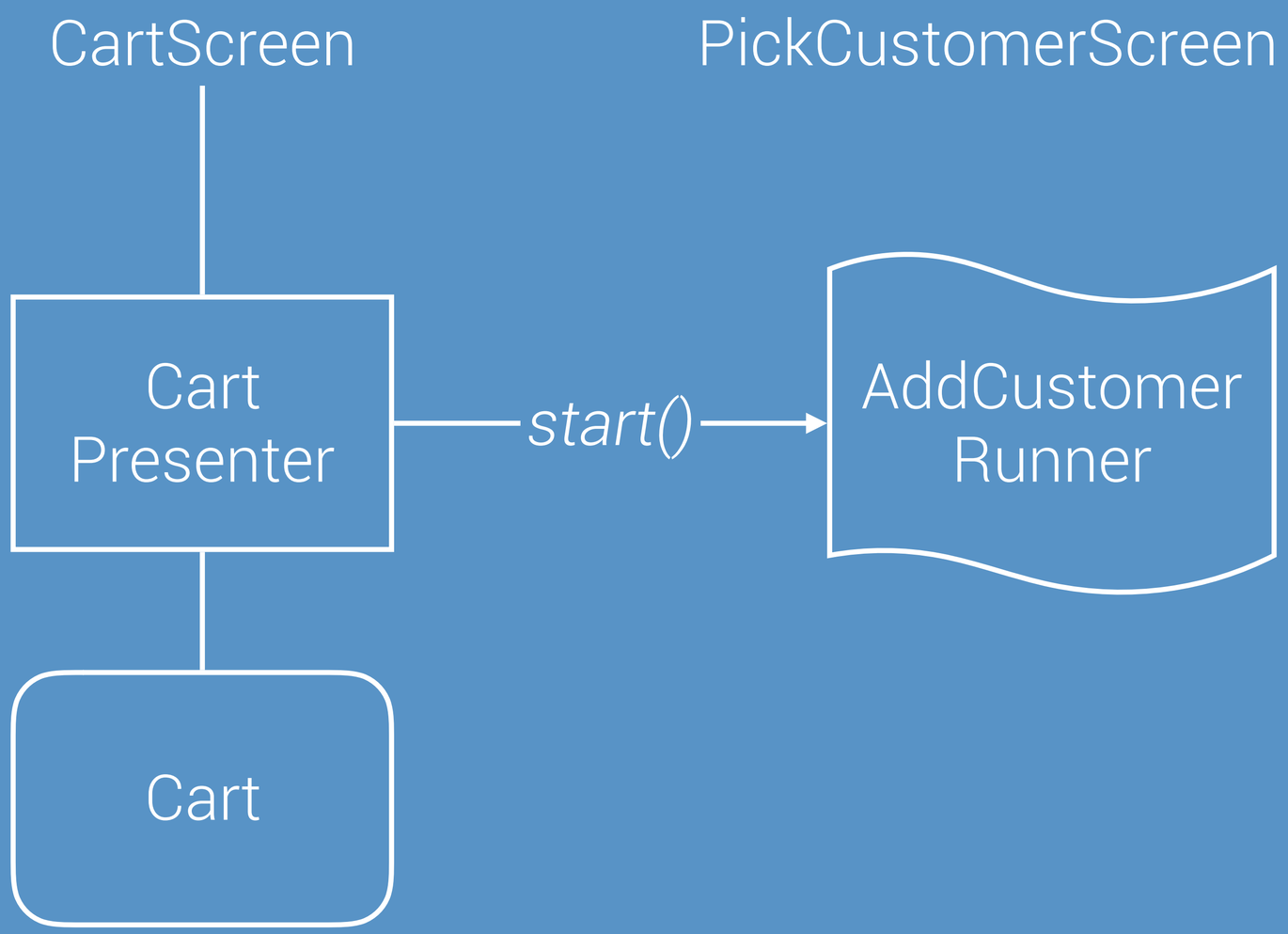
Awkward reuse



Awkward reuse



Awkward reuse



Awkward reuse

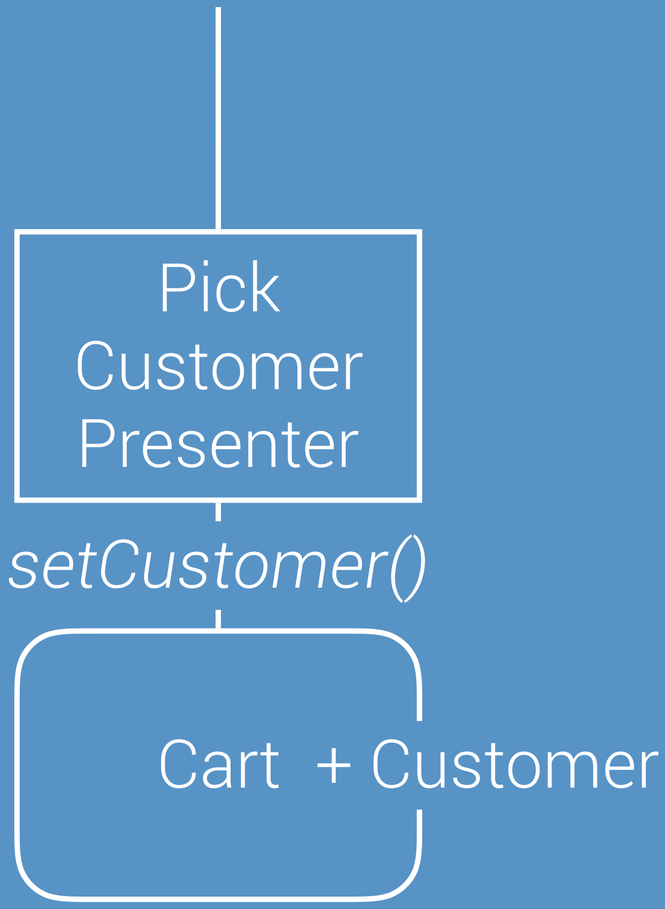
CartScreen

PickCustomerScreen

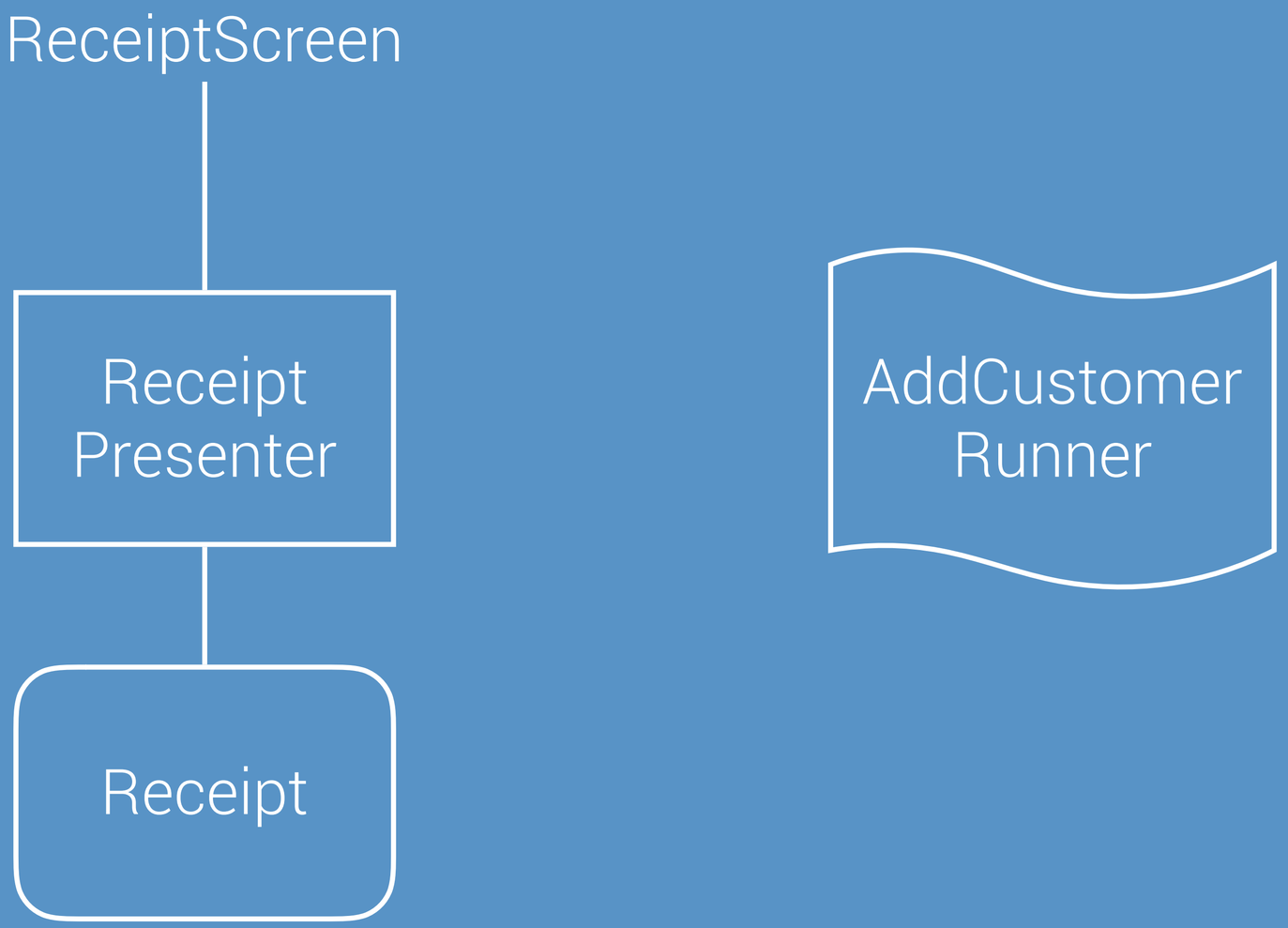
Awkward reuse

CartScreen

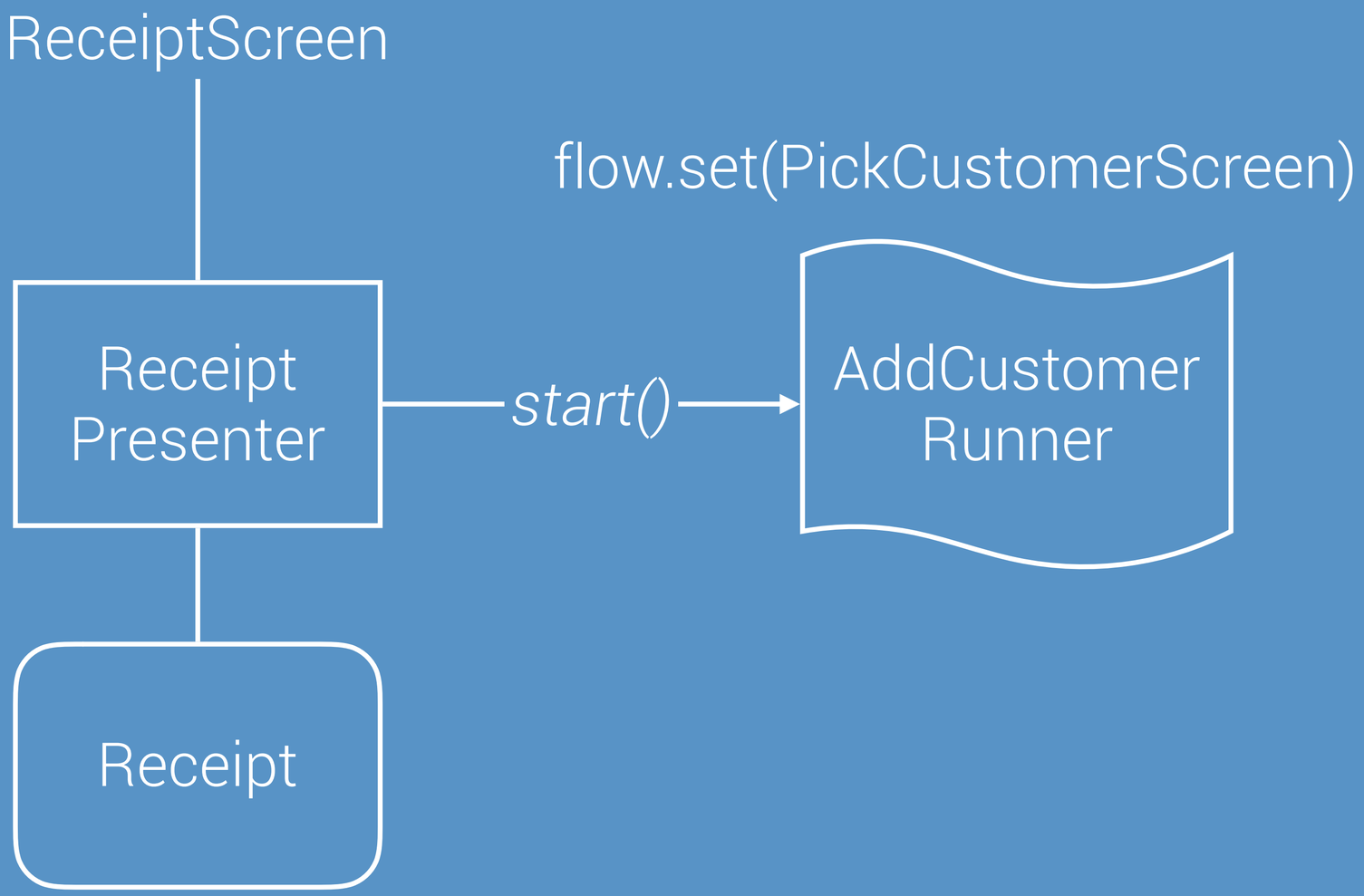
PickCustomerScreen



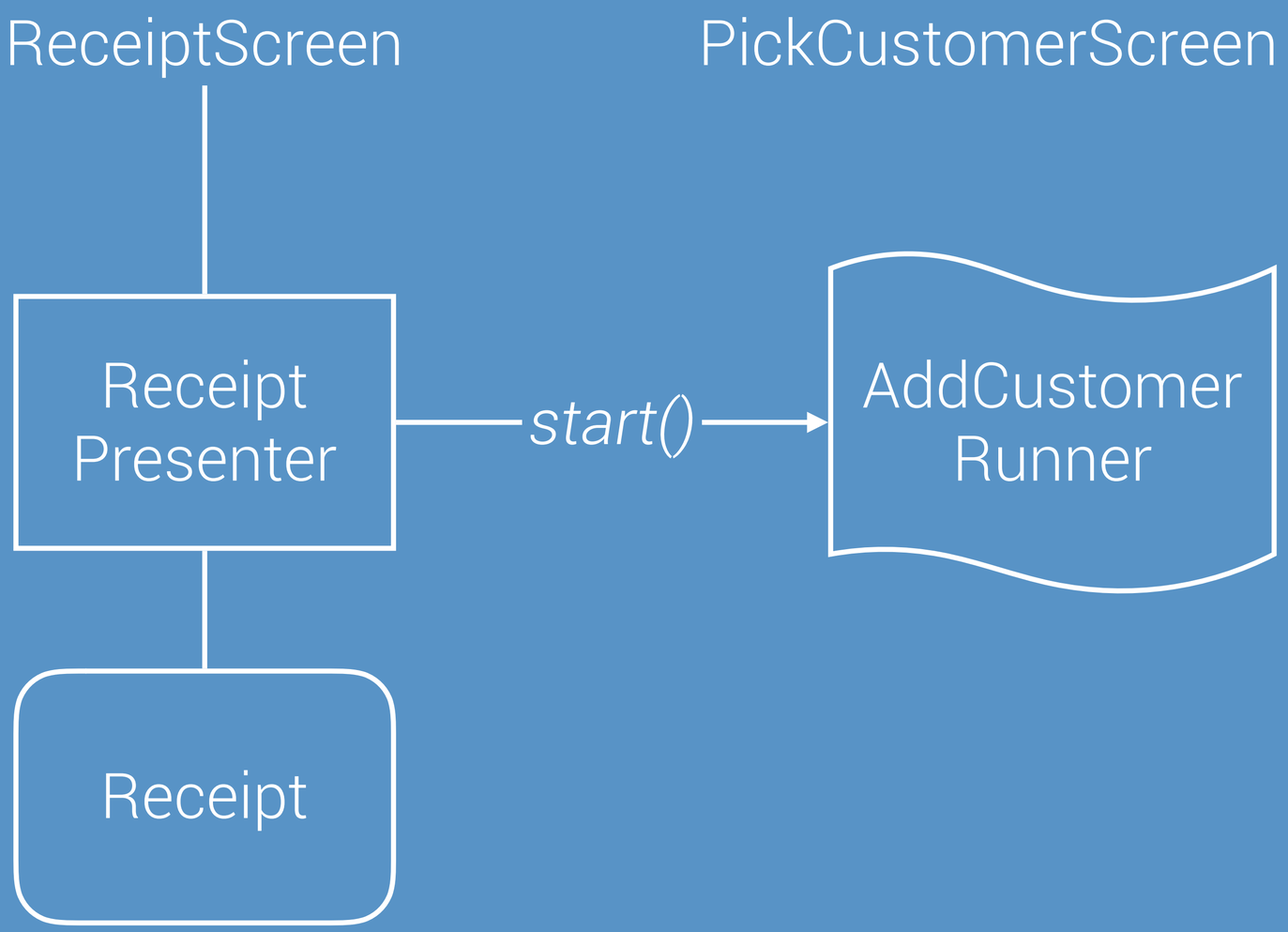
Awkward reuse



Awkward reuse



Awkward reuse



Awkward reuse

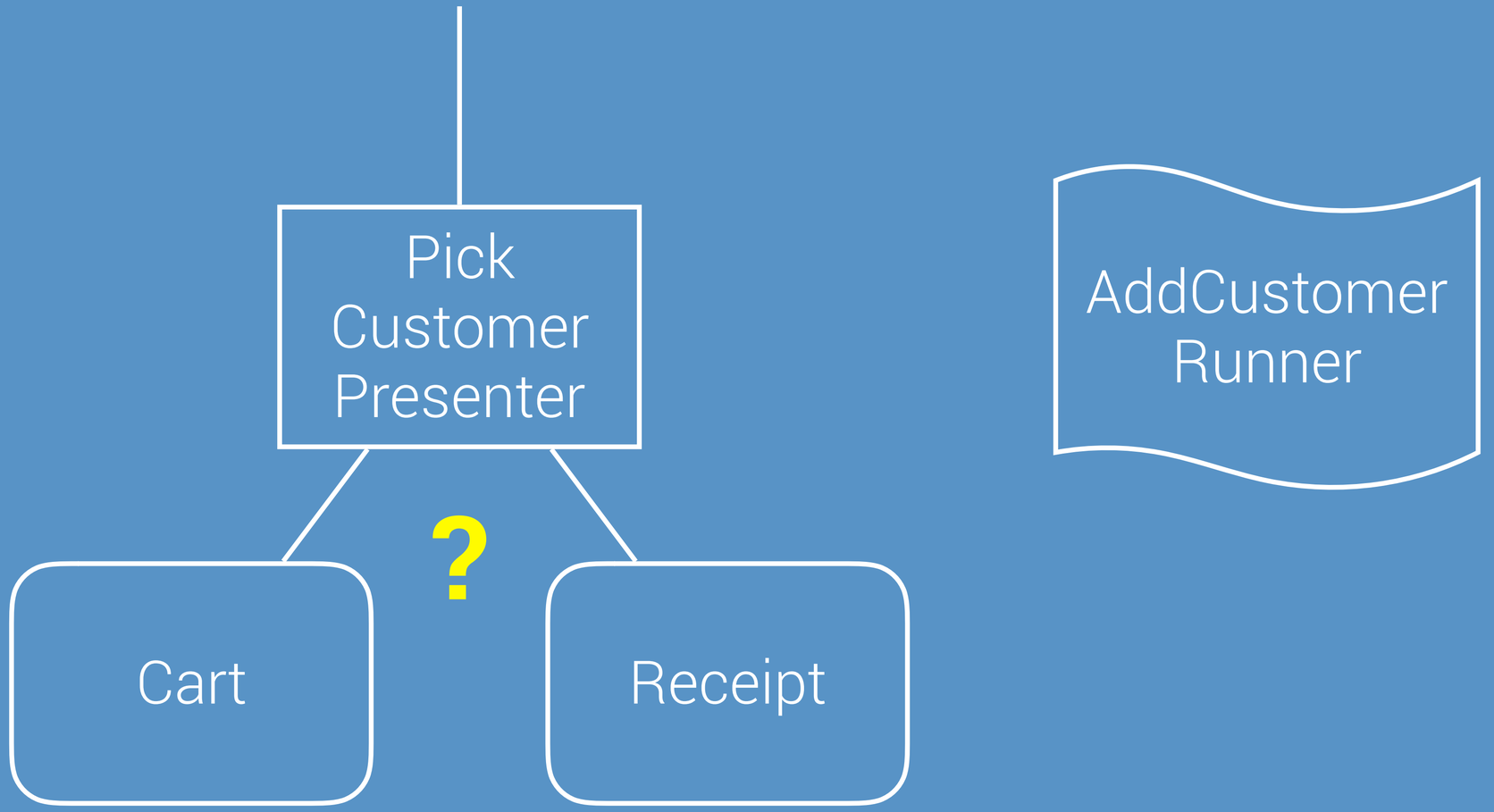
ReceiptScreen

PickCustomerScreen

Awkward reuse

ReceiptScreen

PickCustomerScreen



Awkward reuse

ReceiptScreen
+ Customer ?

PickCustomerScreen

Awkward reuse

ReceiptScreen

PickCustomerScreen

?

Still, pretty good enough

Nearly two hundred contributors

Around three hundred screens

Ships every two weeks

four years later

Good enough don’t scale

Good enough don’t scale

12 minute clean build

2-4 minute rebuild

Spaghetti of the commons

iOS even worse

ViewControllers full of business logic

WWAD syndrome

- Static singletons
- CoreData all the things
- Undiffable XIB, Storyboard files

Learned helplessness

Each platform > 750,000 LOC

Remember, THREE HUNDRED SCREENS

Nearly half just in payment flow

Grassroots heroics no longer practical

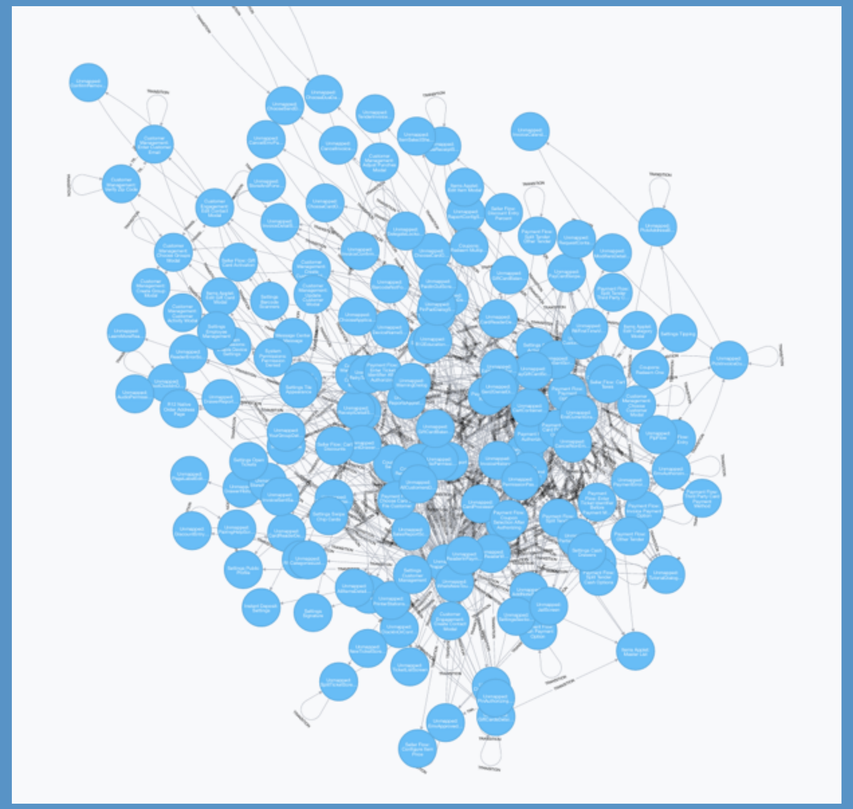
Learned helplessness

Each platform > 750,000 LOC

Remember, THREE HUNDRED SCREENS

Nearly half just in payment flow

Grassroots heroics no longer practical



we bring glad tidings from the future

Reclaiming the commons

Reclaiming the commons

Make fixing things someone's job

Make success possible

- Define the problems
- Describe utopia
- Declare and execute two year plan

Define the problems

Edit, compile, debug loop is too slow

The code is too complex

Dependencies are too interconnected

Define the problems

Edit, compile, debug loop is too slow

Modularize: small code blocks build faster

The code is too complex

Modularize: big features from simpler parts

Dependencies are too interconnected

Modularize: dependencies are a DAG

Describe Utopia

Describe Utopia

Utopia's core principals

Immutability is assumed

Be reactive: push, don't pull

Natural separation of UI and business concerns

Uniformity of API across platforms

“Uniformity?” Code sharing?

Maybe

Writing is the easy part, maintenance is forever

Shared code is foreign code

Have you considered...

- Shared tests
- Shared API definitions (protocol buffers)

“Uniformity?” Code sharing?

Maybe

Writing is the easy part, maintenance is forever

Shared code is foreign code

Have you considered...

- Shared tests
- Shared API definitions (protocol buffers)

Workflows and Bricks

Workflows and Bricks

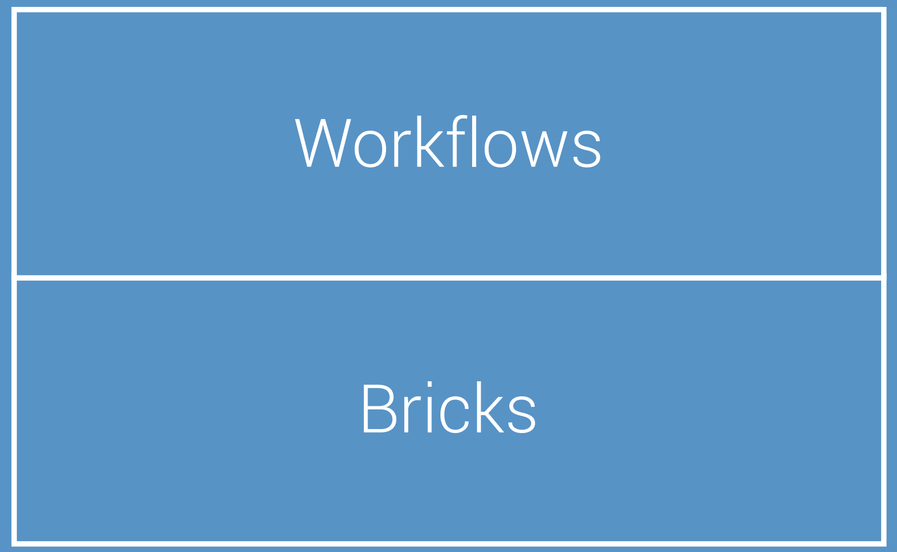
Business model
i/o

Bricks

Workflows and Bricks

App logic
UI modeling

Business model
i/o

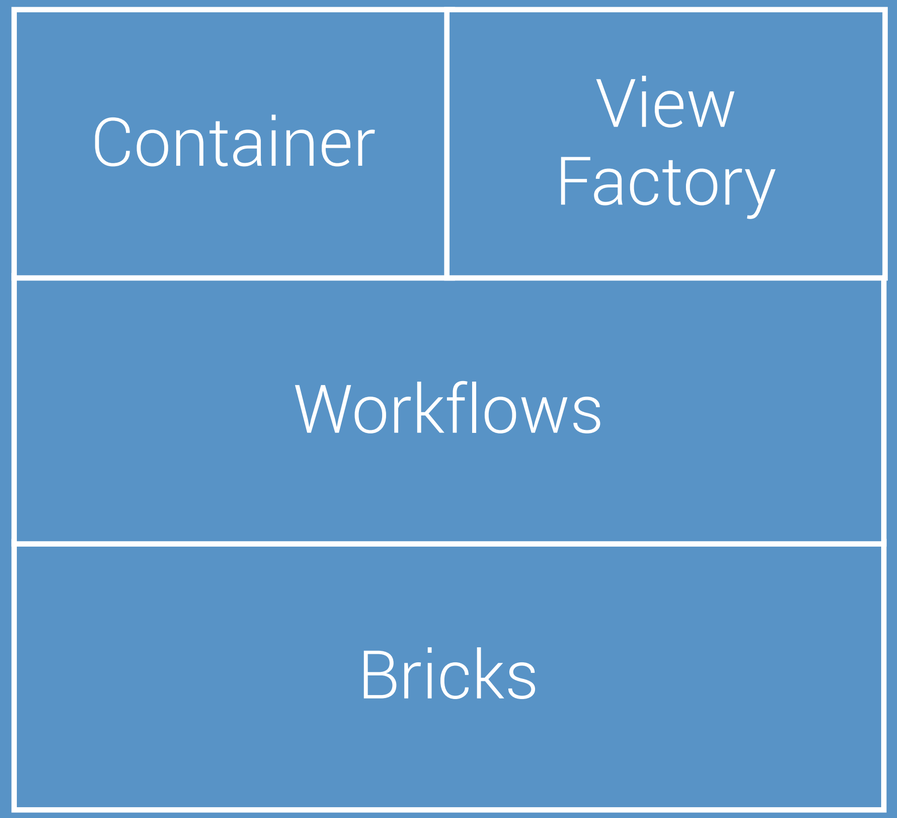


Workflows and Bricks

Rendering
UI event handling

App logic
UI modeling

Business model
i/o

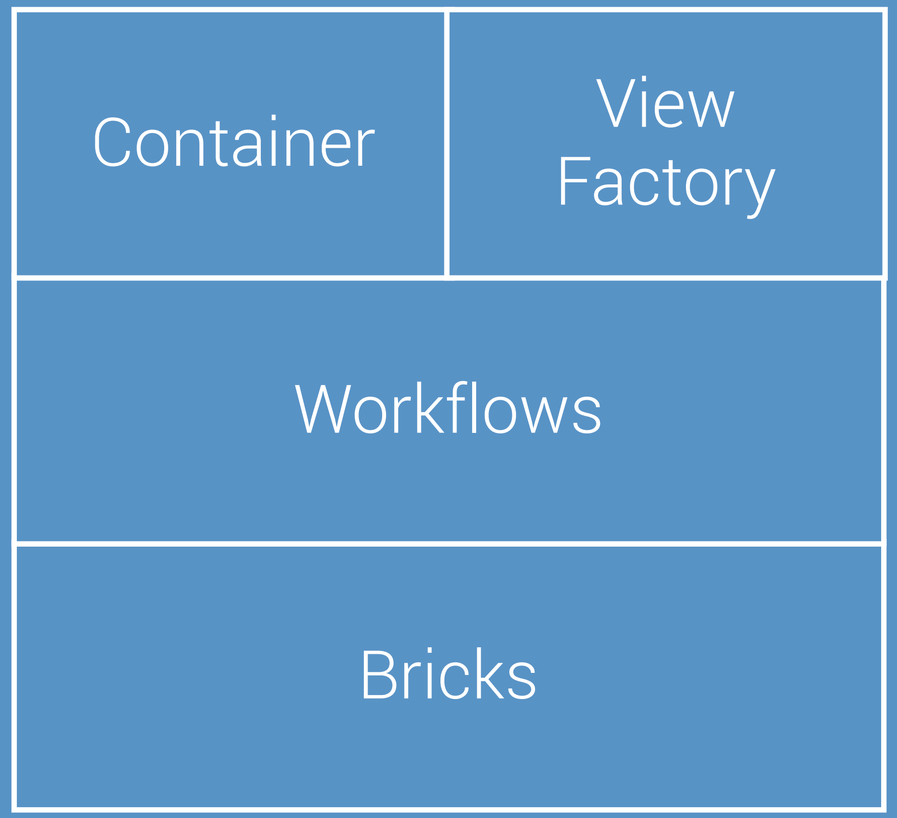


Workflows and Bricks

Rendering
UI event handling

App logic
UI modeling

Business model
i/o



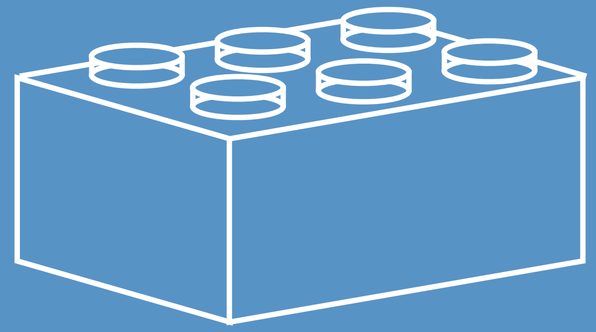
View code

Platform neutral*

“Brick” rhymes with “schmego”

“Brick” rhymes with “schmego”

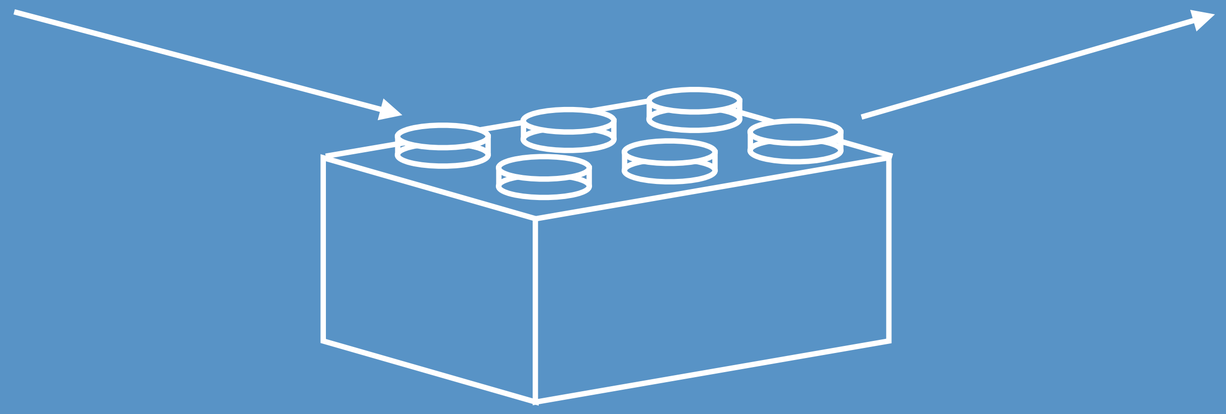
“Brick” rhymes with “schmego”



“Brick” rhymes with “schmego”

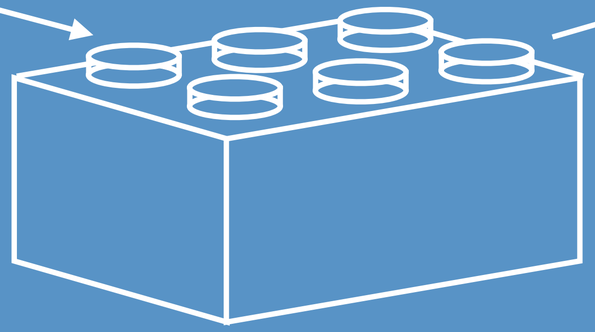
setFoo(FooValue f)

Observable<BarValue> bar();



“Brick” rhymes with “schmego”

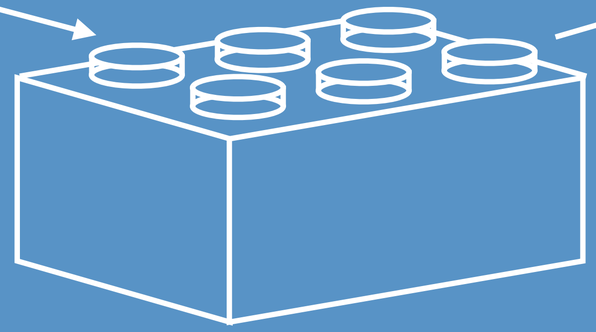
setFoo(FooValue f)



Observable<BarValue> bar();

“Brick” rhymes with “schmego”

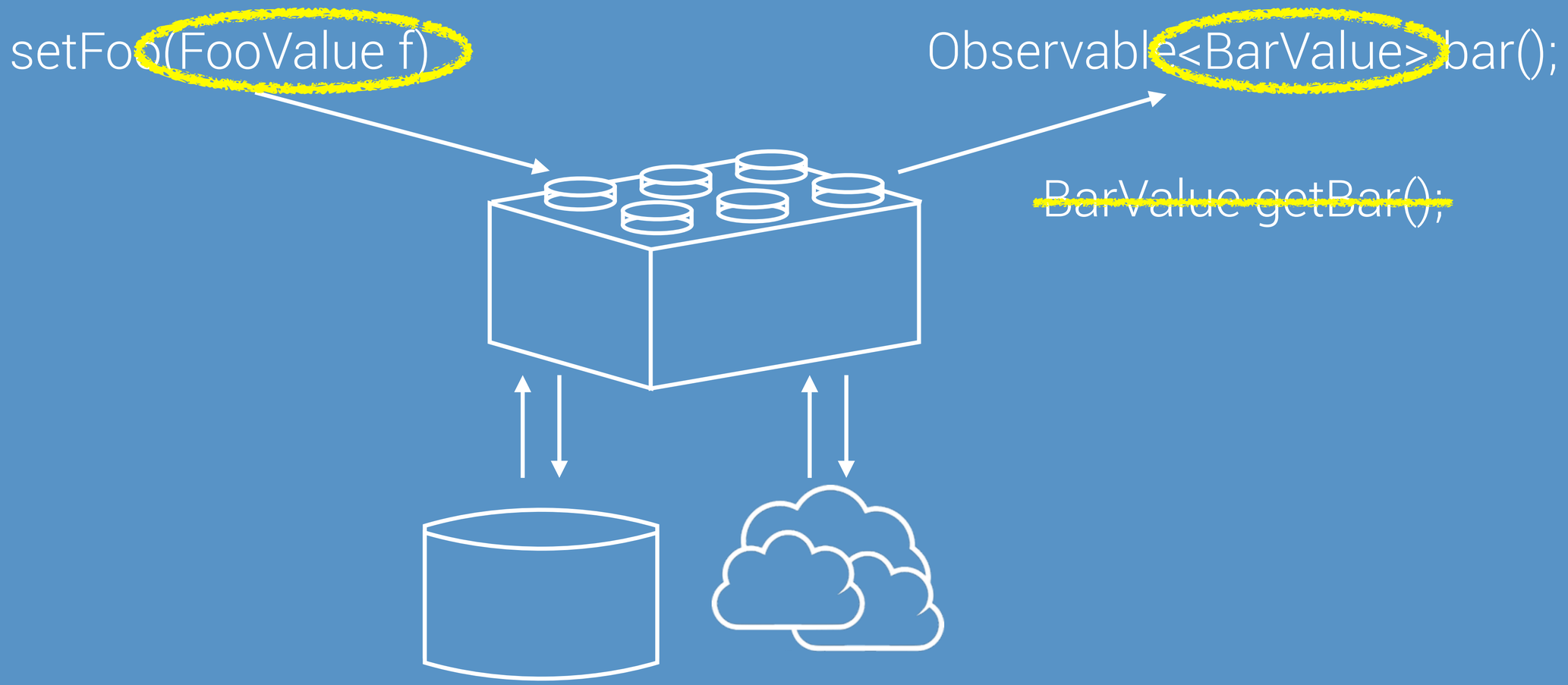
setFoo(FooValue f)



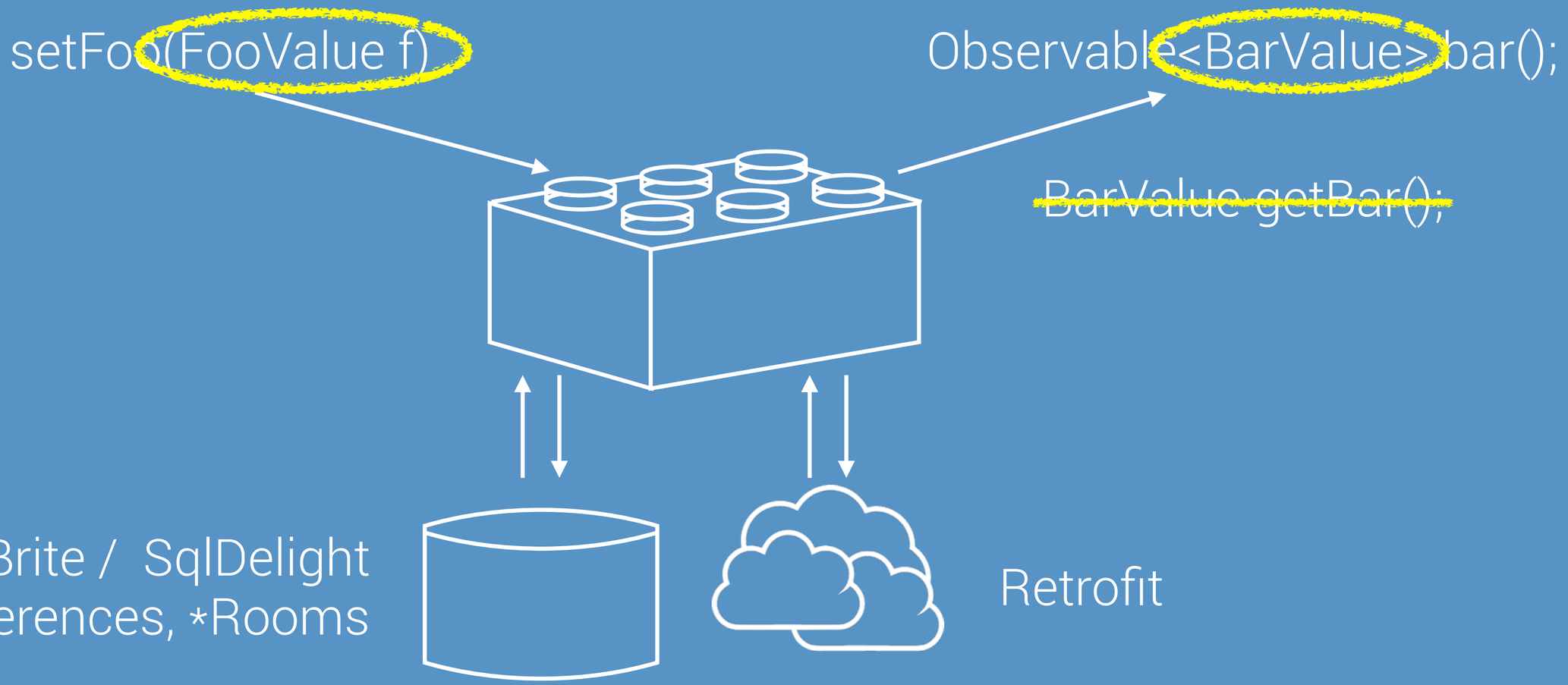
Observable<BarValue> bar();

~~BarValue getBar();~~

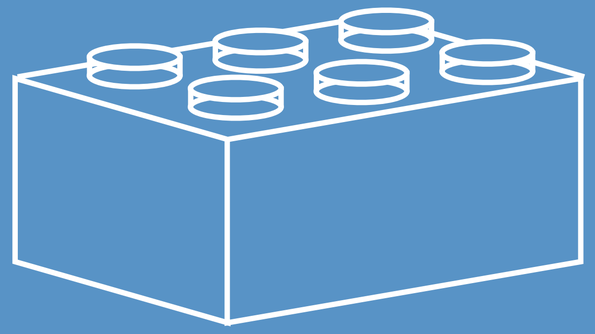
“Brick” rhymes with “schmego”



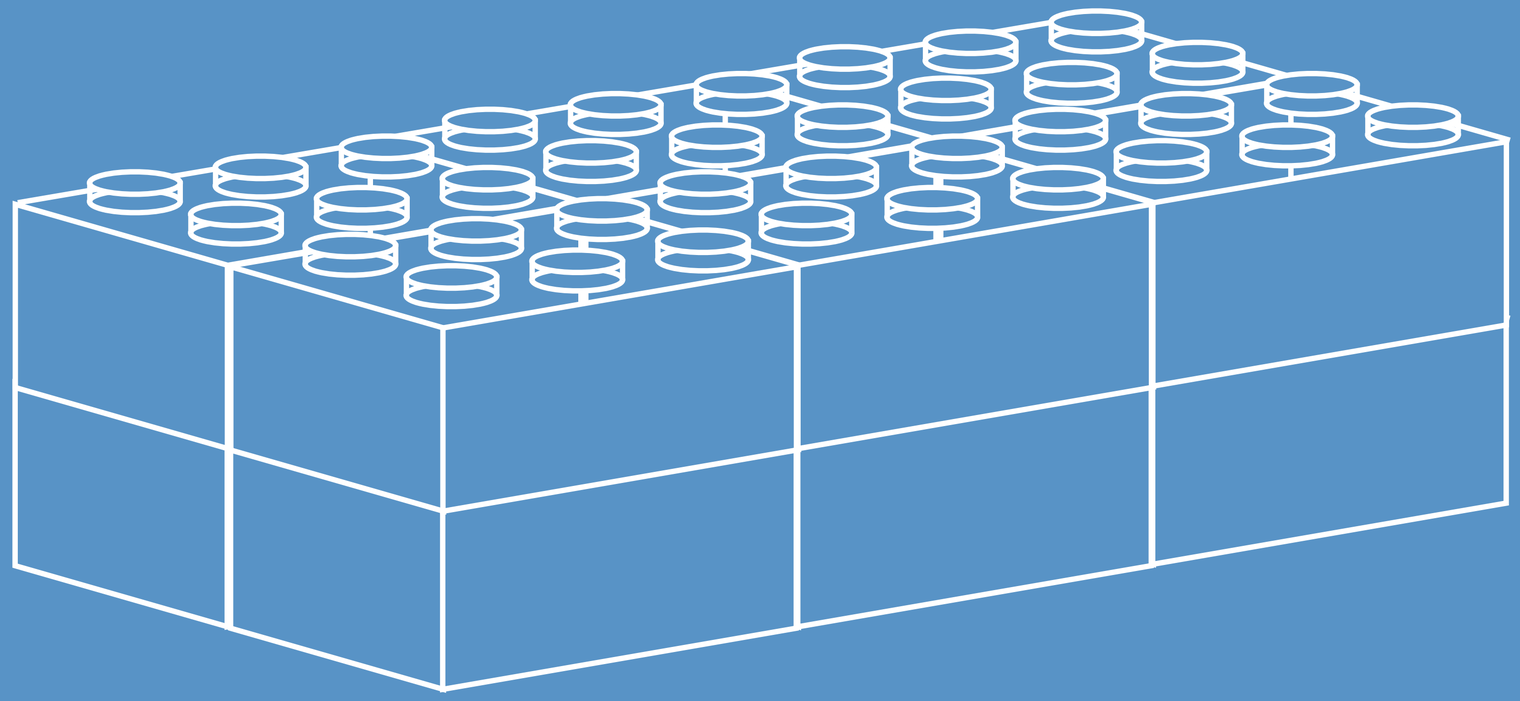
“Brick” rhymes with “schmego”



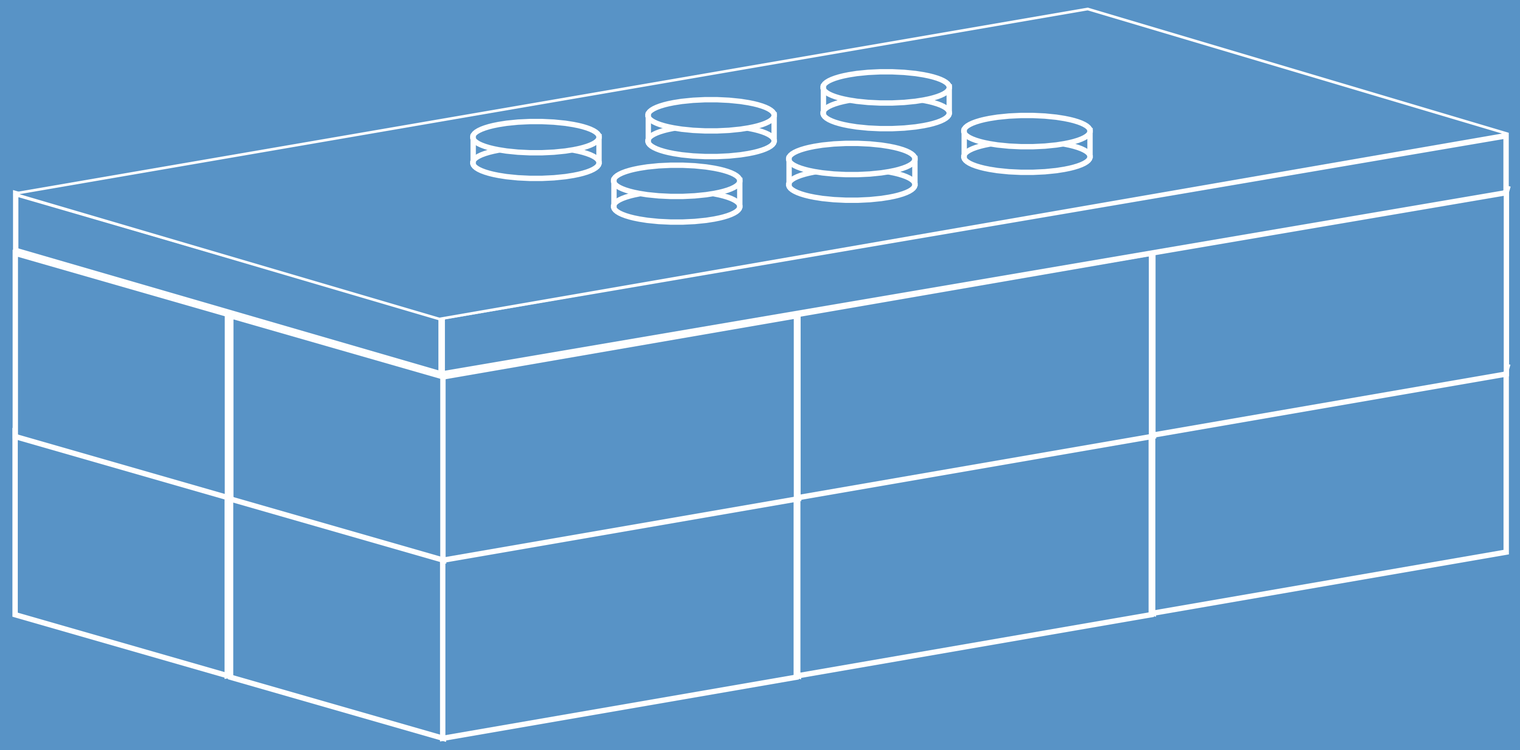
Build big bricks from small bricks



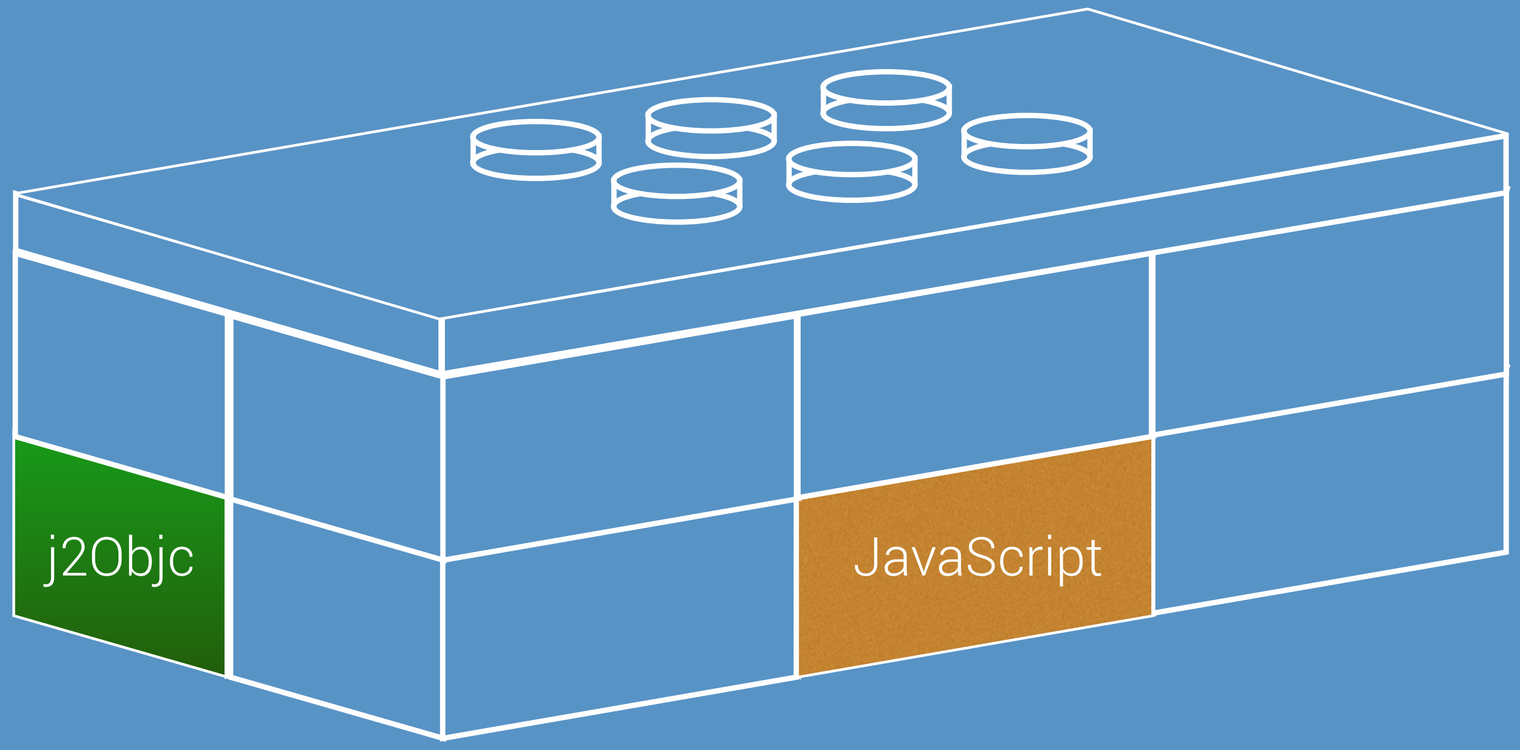
Build big bricks from small bricks



Build big bricks from small bricks



Build big bricks from small bricks



```
enum class StateOfPlay {
  PLAYING, VICTORY, DRAW
}
```

```
enum class StateOfPlay {
  PLAYING, VICTORY, DRAW
}

enum class MARK {
  EMPTY, X, O
}
```

```
enum class StateOfPlay {
  PLAYING, VICTORY, DRAW
}

enum class MARK {
  EMPTY, X, O
}

data class Player(
  val id: String,
  val name: String
)
```

```
enum class StateOfPlay {
  PLAYING, VICTORY, DRAW
}

enum class MARK {
  EMPTY, X, O
}

data class Player(
  val id: String,
  val name: String
)
```

```
data class GameState(
    val id: String,
    val playerX: Player,
    val playerO: Player,
    val stateOfPlay: StateOfPlay,
    val grid: List<List<MARK>>,
    val activePlayerId: String
)
```

```
class GameRunner {

  fun newGame(xPlayerName: String, oPlayerName: String) {}

  fun restoreGame(clientId: String) {}

  fun takeSquare(row: Int, col: Int) {}

  fun end() {}

  fun gameState(): Observable<GameState> {}
}
```

```
class GameRunner {
  /** @throws AssertionError if not called from main thread */
  fun newGame(xPlayerName: String, oPlayerName: String) {}

  /** @throws AssertionError if not called from main thread */
  fun restoreGame(clientId: String) {}

  /** @throws AssertionError if not called from main thread */
  fun takeSquare(row: Int, col: Int) {}

  /** @throws AssertionError if not called from main thread */
  fun end() {}

  fun gameState(): Observable<GameState> {}
}
```

```
class GameRunner {
  /** @throws AssertionError if not called from main thread */
  fun ne

  /** @t
  fun re

  /** @
  fun ta

  /** @thr
  fun end() {}

  fun gameState(): Observable<GameState> {}
}
```

~~observeOn(mainThread())~~

speakerdeck.com/rjrjr/
where-the-reactive-rubber-meets-the-road

```
class GameRunner {
  /** @throws AssertionError if not called from main thread */
  fun newGame(xPlayerName: String, oPlayerName: String) {}

  /** @throws AssertionError if not called from main thread */
  fun restoreGame(clientId: String) {}

  /** @throws AssertionError if not called from main thread */
  fun takeSquare(row: Int, col: Int) {}

  /** @throws AssertionError if not called from main thread */
  fun end() {}

  fun gameState(): Observable<GameState> {}
}
```

```
class GameRunner {
```

```
class GameRunner {
  sealed class Command {

    data class NewGame(val xPlayer: String, val yPlayer: String): Command()

    data class RestoreGame(val id: String): Command()

    data class TakeSquare(val row: Int, val col: Int): Command()

    object End: Command()
  }

  fun asTransformer():
      Observable.Transformer<Command, GameState> {
  }
}
```

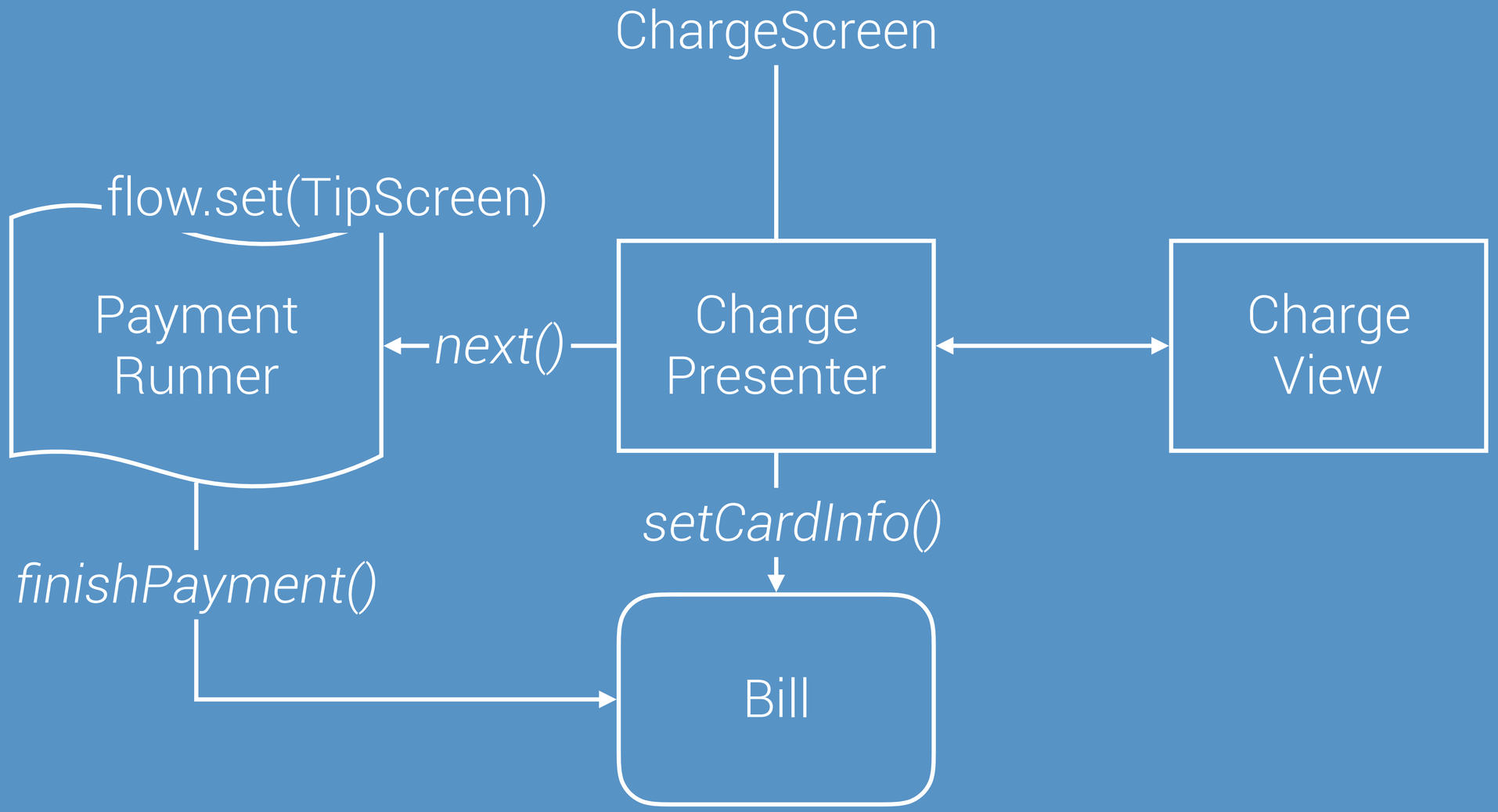
```
class GameRunner {
  sealed class Comm

    data class NewC                                    : String): Command()

    data class Rest

    data class Take                                    Command()

    object End: Co
  }

  fun asTransformer(
      Observable.Transformer<Command, GameState> {
  }
}
```

search for:

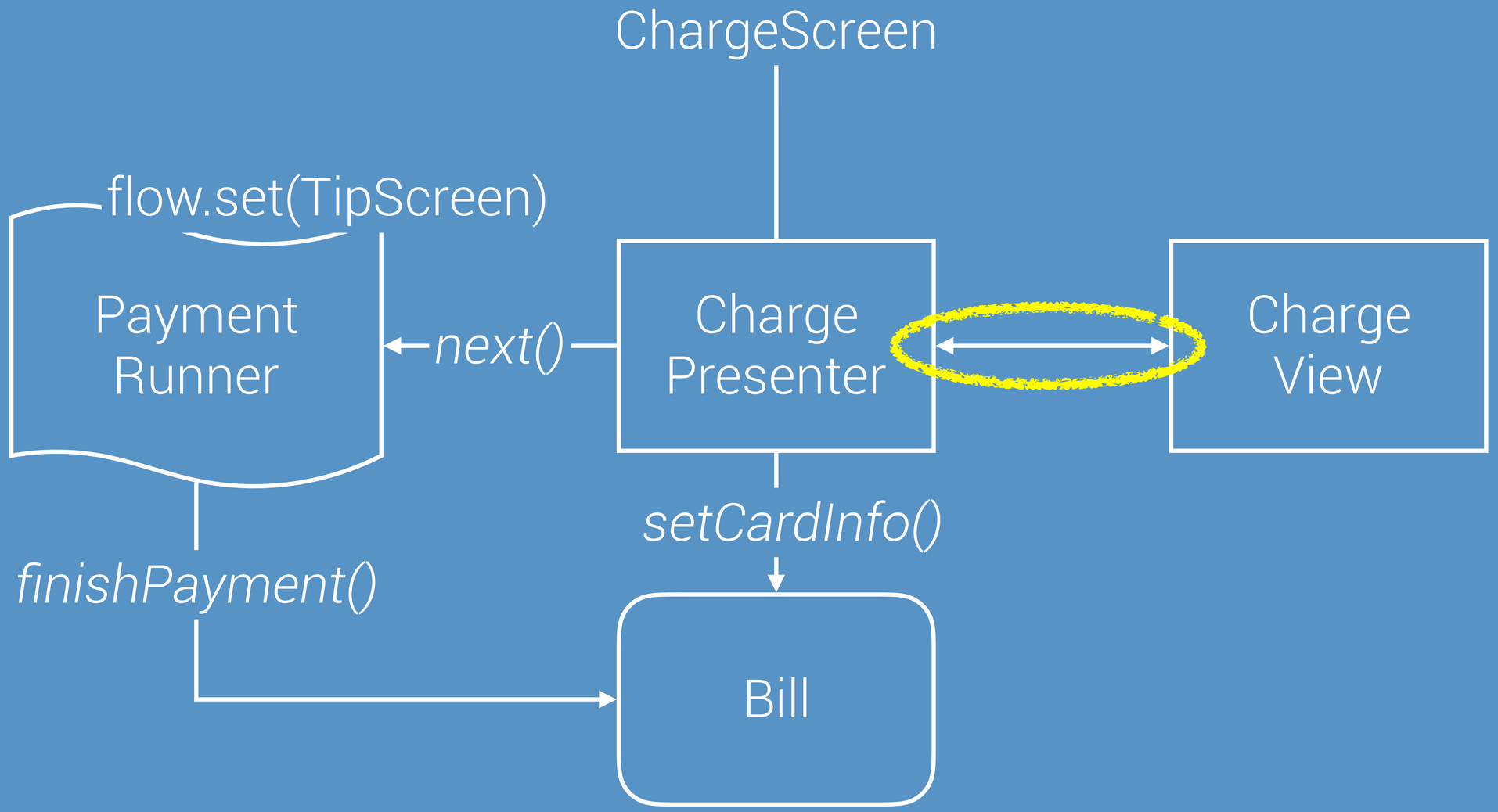
Dan Lew transformer

Workflow through the brickyard

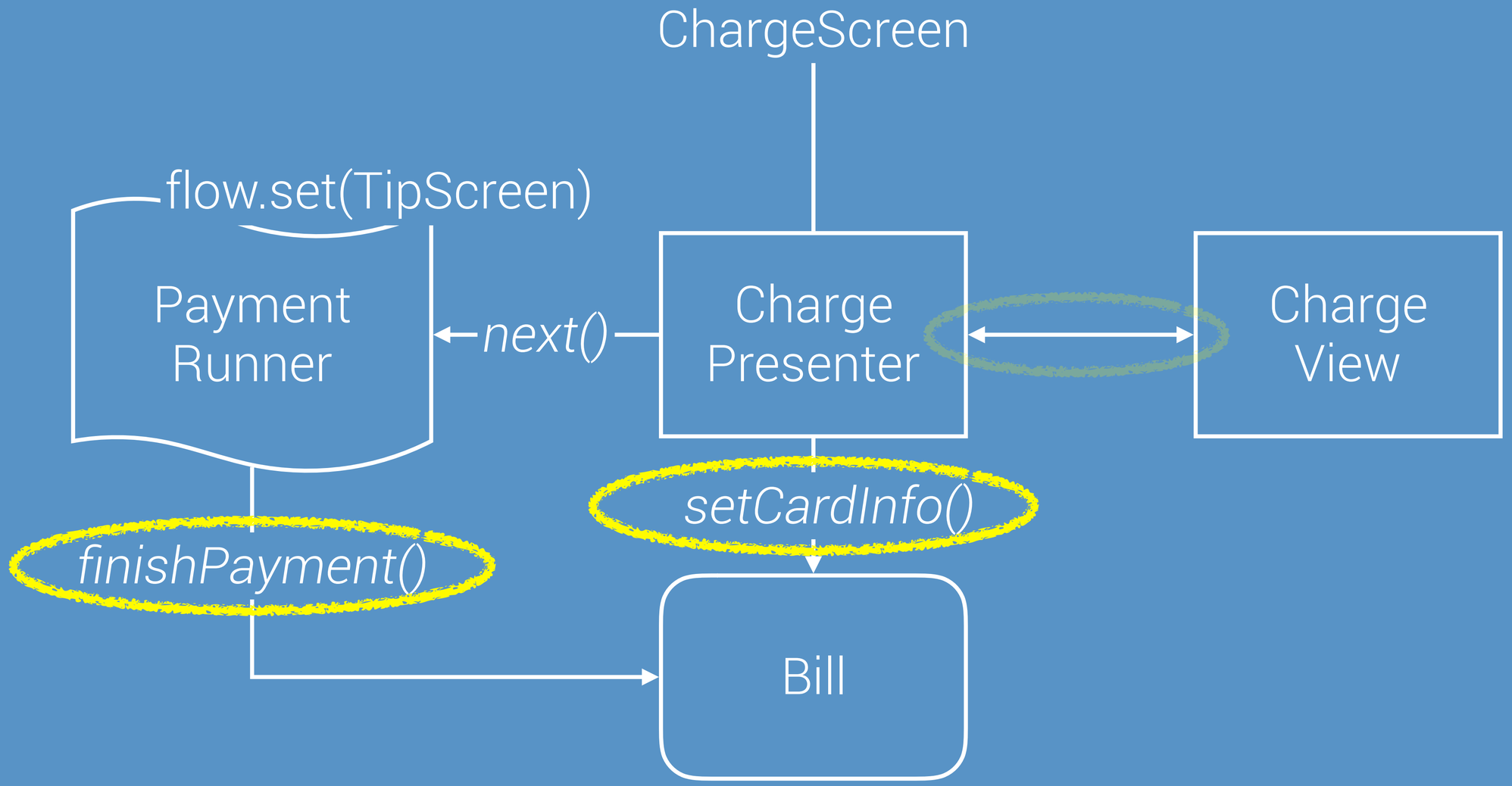
Remember these things?



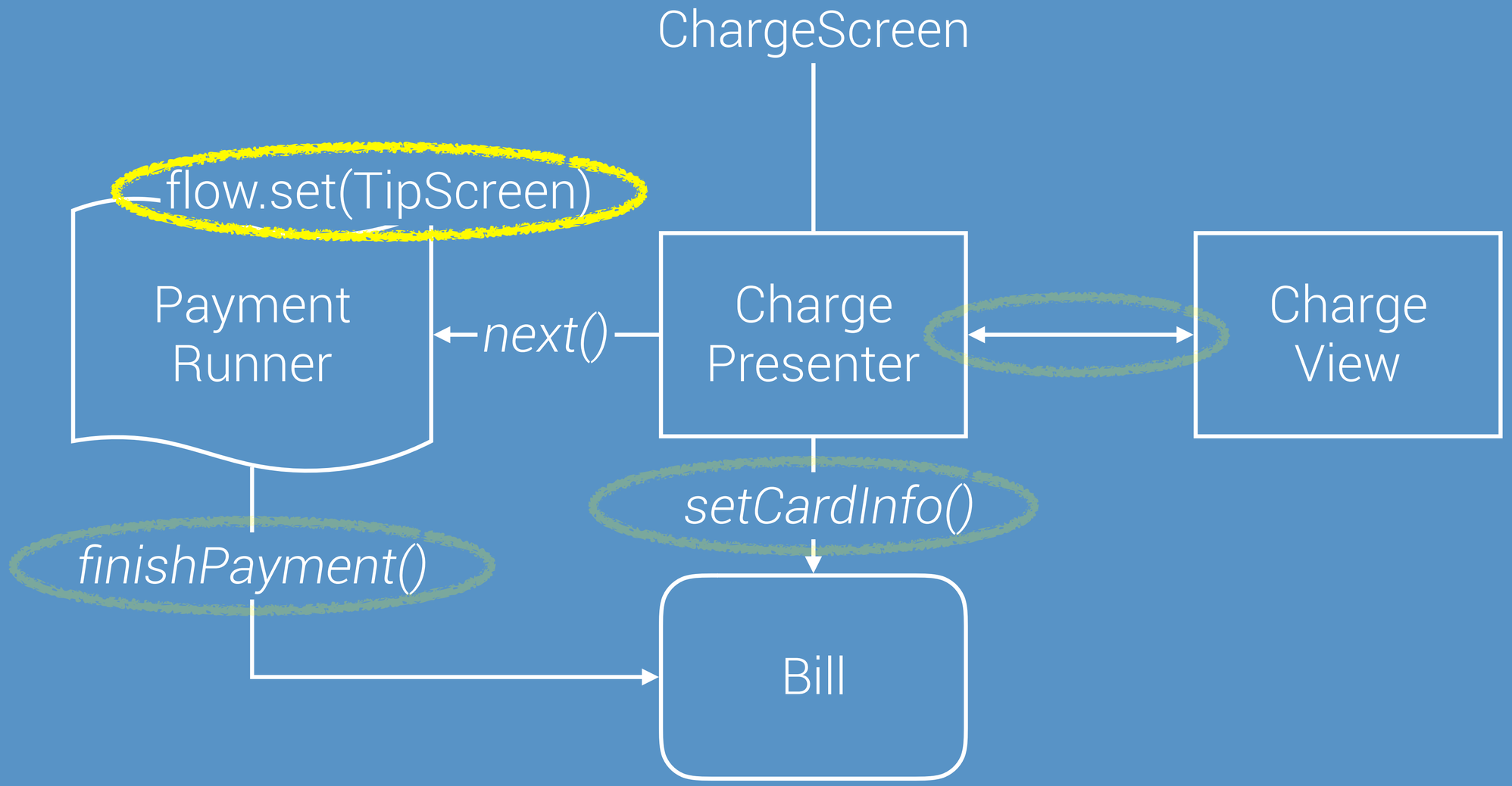
Remember these things?



Remember these things?



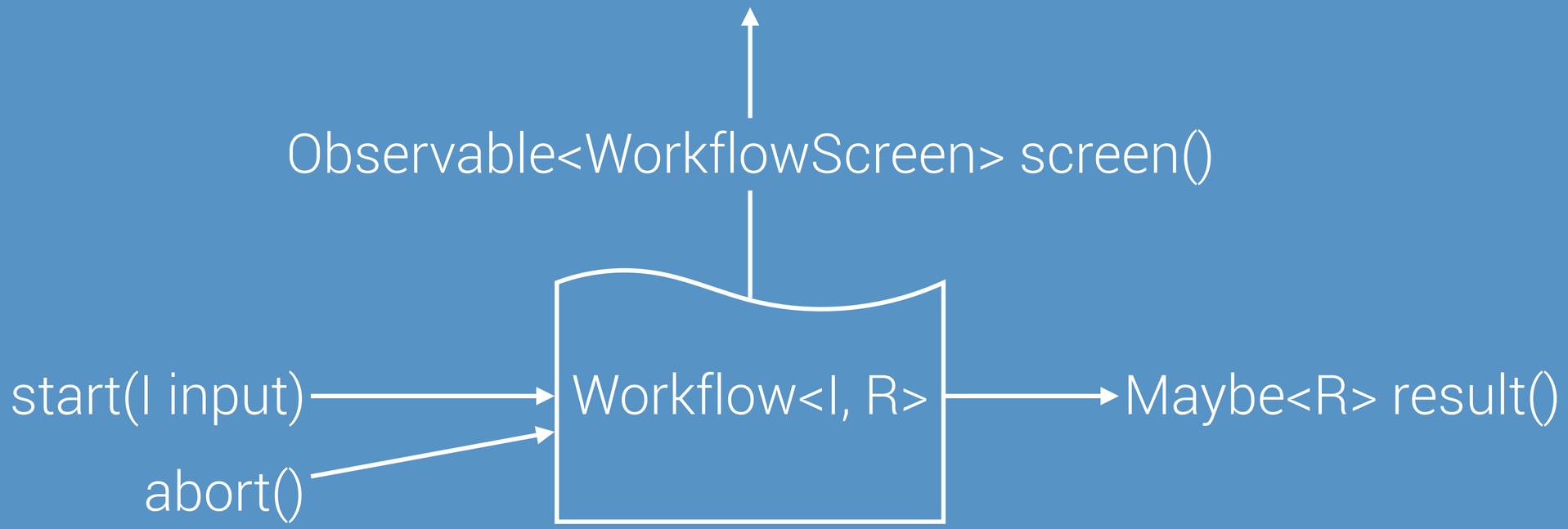
Remember these things?




Runner
+
Presenter
=
Workflow<I, R>

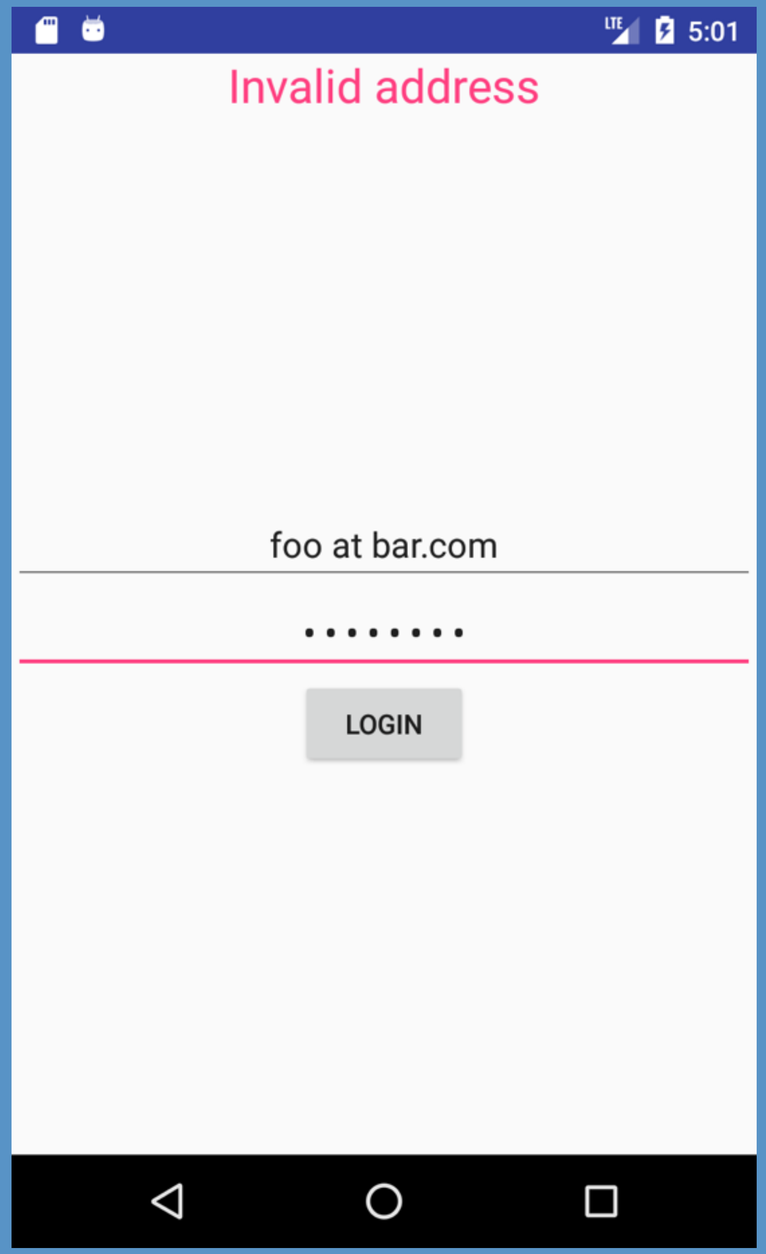
Workflow as pipeline

Workflow as ui driver

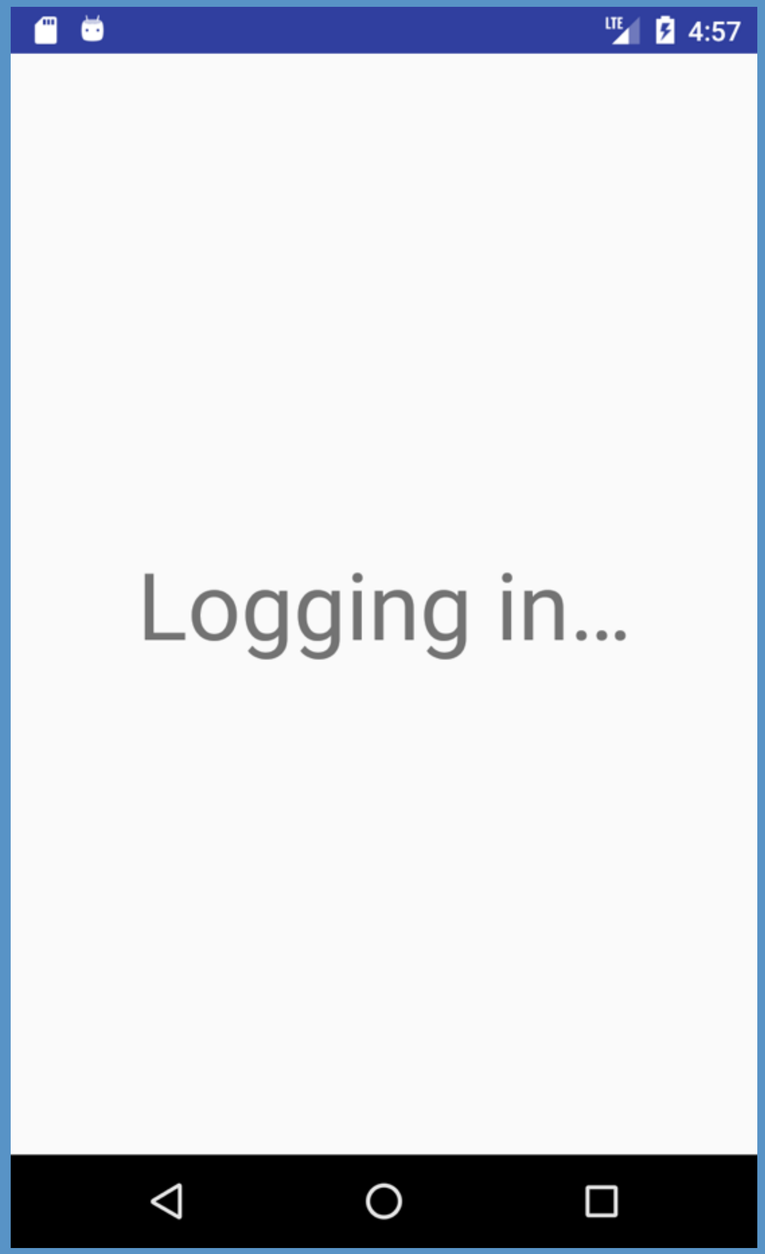


Rhymes with “view model”

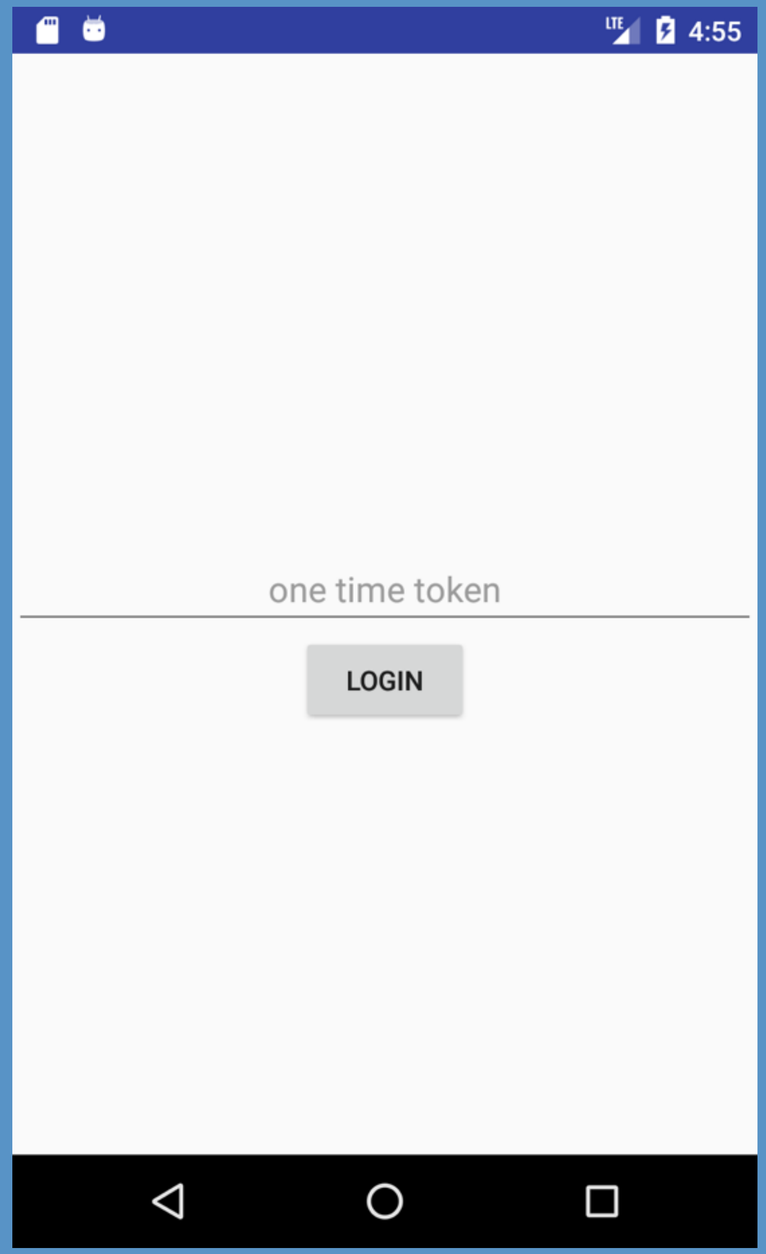
```
/** Allows interaction with a [Workflow] in a particular state. */
abstract class WorkflowScreen<D, out E> protected constructor(

    /** Uniquely identifies this screen. */
    val key: String,

    /** Stream of data to render this screen. */
    val screenData: Observable<D>,

    /** Callback methods (click handlers, etc.) handled by this screen. */
    val eventHandler: E
)
```



LoginScreen



AuthorizingScreen



SecondFactorScreen

```
class LoginScreen(
    errorMessage: Observable<String>,
    eventHandler: Events
) : WorkflowScreen<String, Events>(KEY, errorMessage, eventSink) {

  companion object {
    val KEY = LoginScreen::class.name
  }

  interface Events {
    fun onLogin(event: SubmitLogin)
  }

  data class SubmitLogin(
    val email: String,
    val password: String
  )
}
```

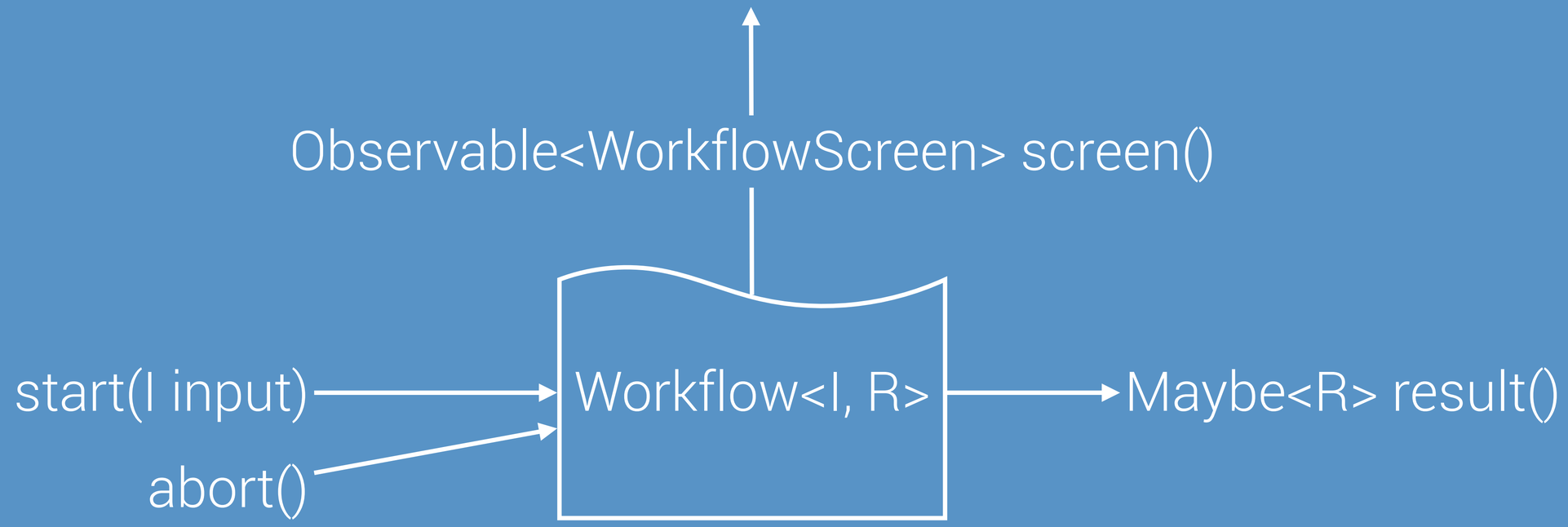
```
class LoginScreen(
    errorMessage: Observable<String>,
    eventHandler: Events
) : WorkflowScreen<String, Events>(KEY, errorMessage, eventSink) {

  companion object {
    val KEY = LoginScreen::class.name
  }

  interface Events {
    fun onLogin(event: SubmitLogin)
  }

  data class SubmitLogin(
    val email: String,
    val password: String
  )
}
```

```
class LoginScreen(
    errorMessage: Observable<String>,
    eventHandler: Events
) : WorkflowScreen<String, Events>(KEY, errorMessage, eventSink) {

  companion object {
    val KEY = LoginScreen::class.name
  }

  interface Events {
    fun onLogin(event: SubmitLogin)
  }

  data class SubmitLogin(
    val email: String,
    val password: String
  )
}
```

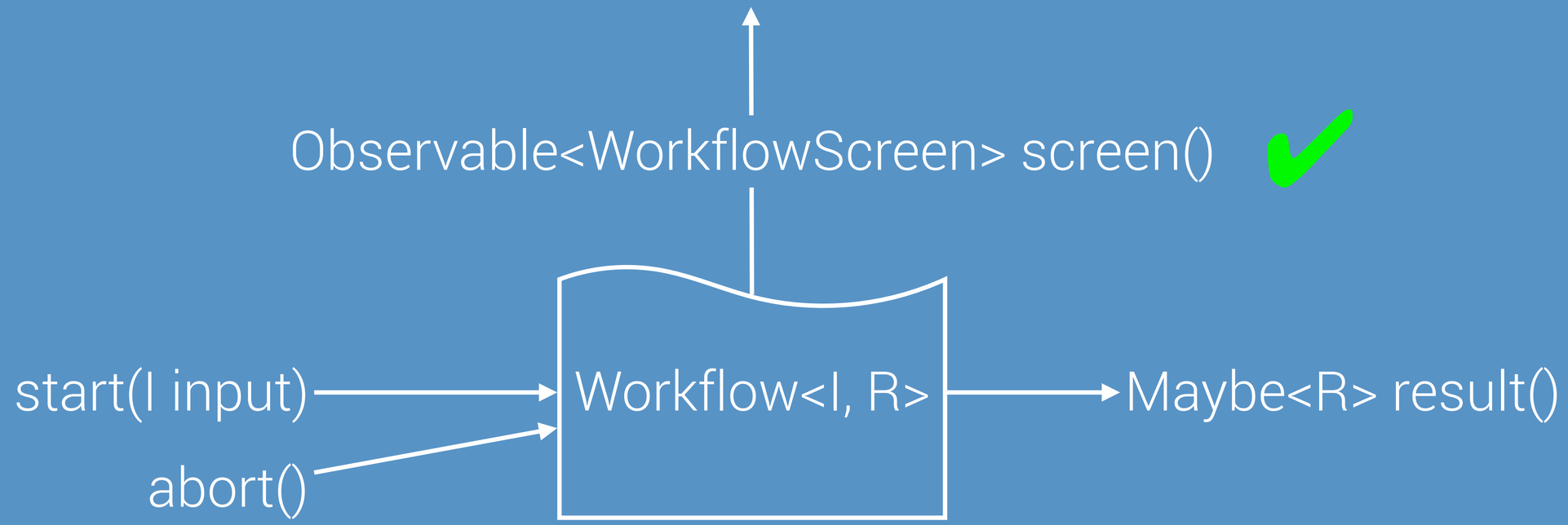
```
class LoginScreen(
    errorMessage: Observable<String>,
    eventHandler: Events
) : WorkflowScreen<String, Events>(KEY, errorMessage, eventSink) {

  companion object {
    val KEY = LoginScreen::class.name
  }

  interface Events {
    fun onLogin(event: SubmitLogin)
  }

  data class SubmitLogin(
    val email: String,
    val password: String
  )
}
```

```
class ConfirmChargeCardOnFileScreen(
    screenData: Observable<ScreenData>,
    eventHandler: Events
) : WorkflowScreen<ScreenData, Events>(KEY, screenData, eventHandler) {

  companion object {
    val KEY = ConfirmChargeCardOnFileScreen::class.name
  }

  data class ScreenData(
      val amountDue: Money,
      val customerName: String,
      val cardNameAndNumber: String,
      val instrumentIndex: Int)

  interface Events {
    fun doNotChargeCardOnFile()
    fun chargeCardOnFile(tenderedAmount: Money, instrumentIndex: Int)
  }
```

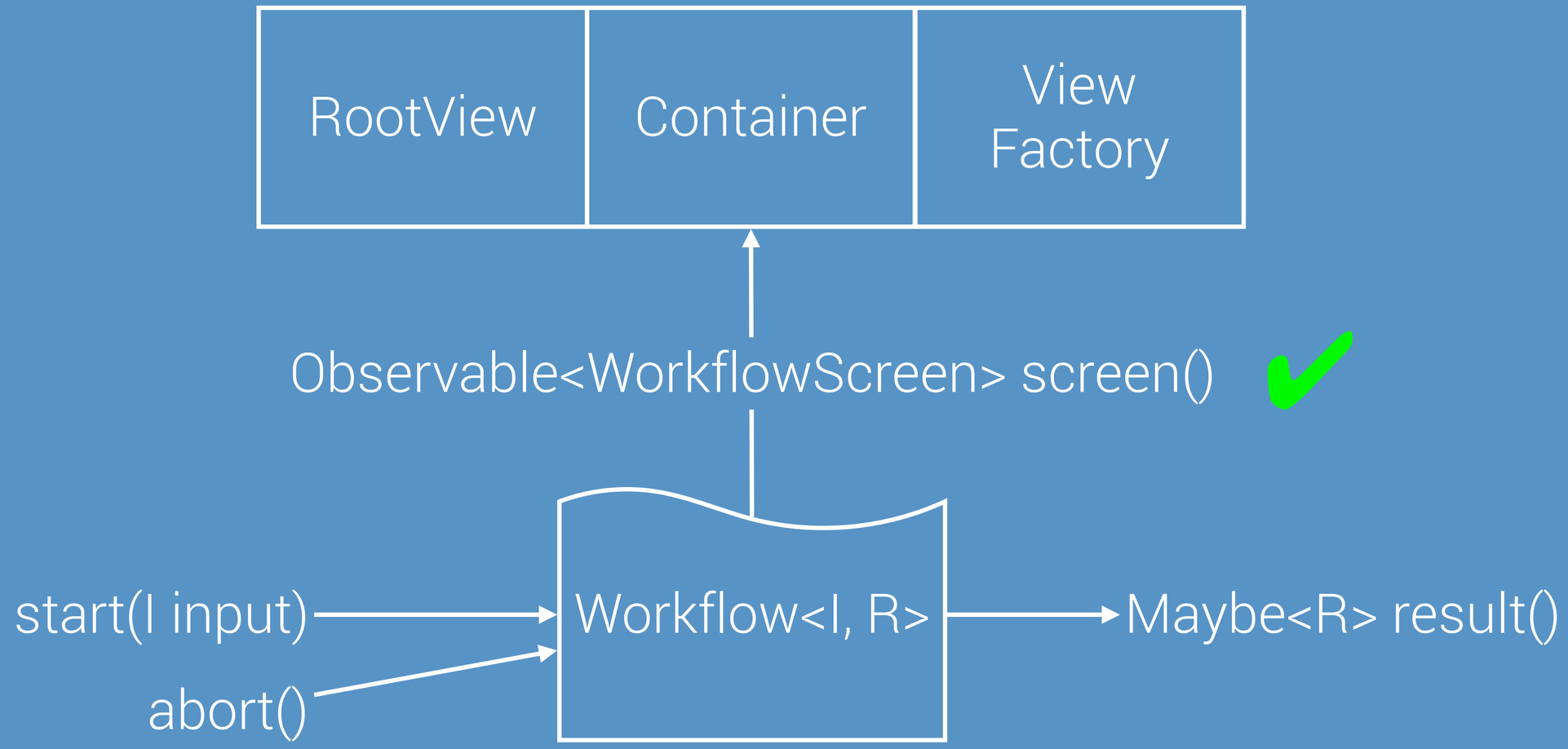
Workflow as view model source



Workflow as view model source



Workflow as view model source



```
class AuthViewFactory : AbstractViewFactory(asList(

    bindLayout(LoginScreen.KEY, R.layout.login) { screen ->
      LoginCoordinator(screen as LoginScreen)
    },

    bindLayout(AuthorizingScreen.KEY, R.layout.authorizing) { screen ->
      AuthorizingCoordinator(screen as AuthorizingScreen)
    },

    bindLayout(SecondFactorScreen.KEY, R.layout.second_factor) { screen ->
      SecondFactorCoordinator(screen as SecondFactorScreen)
    }

))
```

```
class LoginCoordinator(private val screen: LoginScreen) : Coordinator() {
  private var subscription: Subscription = Subscriptions.unsubscribed()

  override fun attach(view: View) {
```

```
class LoginCoordinator(private val screen: LoginScreen) : Coordinator() {
  private var subscription: Subscription = Subscriptions.unsubscribed()

  override fun attach(view: View) {
    val error = view.findViewById<View>(R.id.login_error_message) as TextView
    val email = view.findViewById<View>(R.id.login_email) as EditText
    val password = view.findViewById<View>(R.id.login_password) as EditText
    val button = view.findViewById<View>(R.id.login_button) as Button
```

```
class LoginCoordinator(private val screen: LoginScreen) : Coordinator() {
  private var subscription: Subscription = Subscriptions.unsubscribed()

  override fun attach(view: View) {
    val error = view.findViewById<View>(R.id.login_error_message) as TextView
    val email = view.findViewById<View>(R.id.login_email) as EditText
    val password = view.findViewById<View>(R.id.login_password) as EditText
    val button = view.findViewById<View>(R.id.login_button) as Button

    button.setOnClickListener { _ ->
      val event = SubmitLogin(email.text.toString(),
          password.text.toString())
      screen.eventHandler.login(event)
    }
```

```
class LoginCoordinator(private val screen: LoginScreen) : Coordinator() {
  private var subscription: Subscription = Subscriptions.unsubscribed()

  override fun attach(view: View) {
    val error = view.findViewById<View>(R.id.login_error_message) as TextView
    val email = view.findViewById<View>(R.id.login_email) as EditText
    val password = view.findViewById<View>(R.id.login_password) as EditText
    val button = view.findViewById<View>(R.id.login_button) as Button

    button.setOnClickListener { _ ->
      val event = SubmitLogin(email.text.toString(),
          password.text.toString())
      screen.eventHandler.login(event)
    }
    subscription = screen.screenData.subscribe { error.text = it }
  }
```

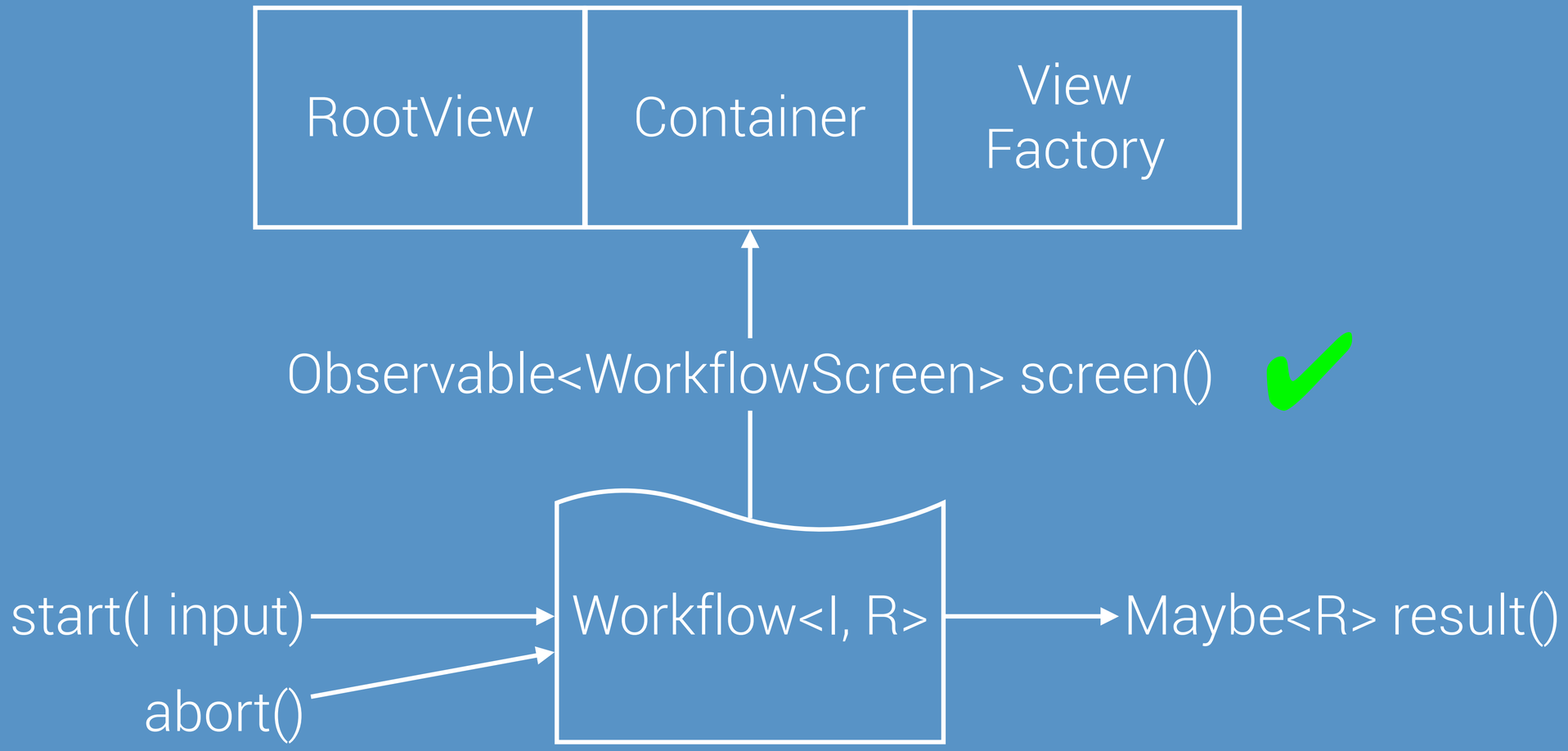
```
class LoginCoordinator(private val screen: LoginScreen) : Coordinator() {
  private var subscription: Subscription = Subscriptions.unsubscribed()

  override fun attach(view: View) {
    …
  }

  override fun detach(view: View?) {
    subscription.unsubscribe()
  }
```

```
class LoginCoordinator(private val screen: LoginScreen) : Coordinator() {
  private var subscription: Subscription = Subscriptions.unsubscribed()

  override fun atta
    …
  }

  override fun deta
    subscription.un
  }
```

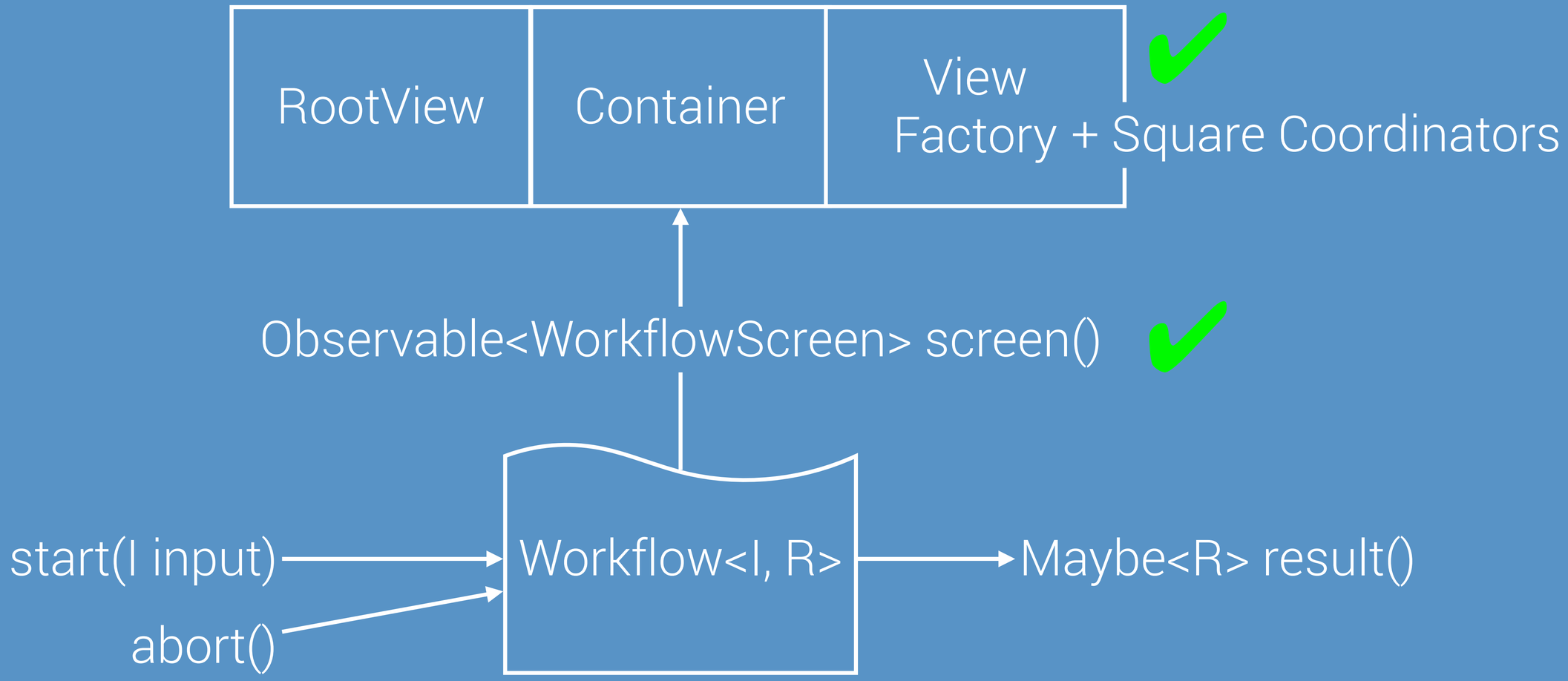
search for:

square coordinators

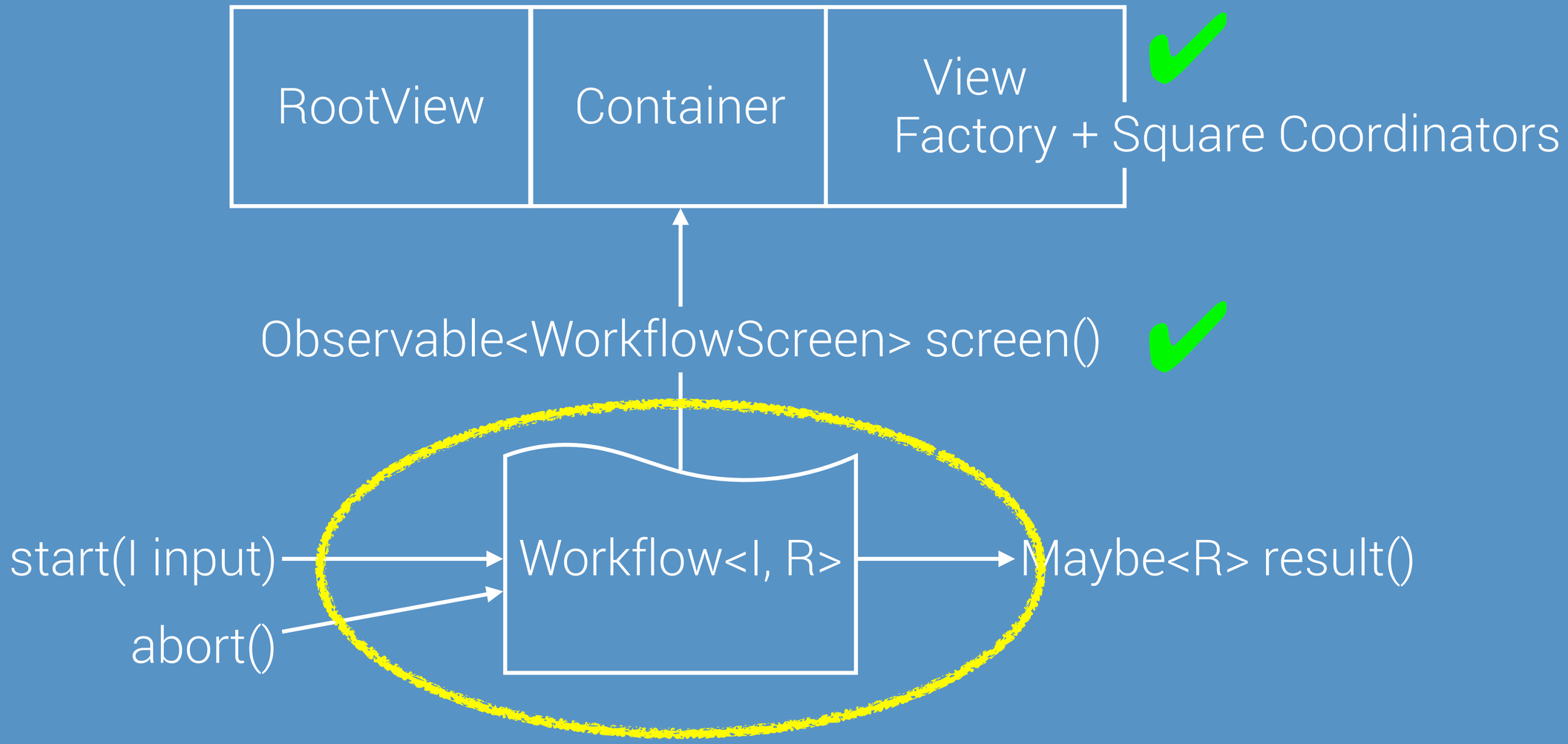
Workflow as state machine



Workflow as state machine



Workflow as state machine



```
class AuthWorkflow(): Workflow<Unit, String>
```

```
class AuthWorkflow(): Workflow<Unit, String>,
    LoginScreen.Events, SecondFactorScreen.Events
```

```
class AuthWorkflow(): Workflow<Unit, String>,
    LoginScreen.Events, SecondFactorScreen.Events {

  private val currentScreen = BehaviorSubject.create<String>()
```

```
class AuthWorkflow(): Workflow<Unit, String>,
    LoginScreen.Events, SecondFactorScreen.Events {

  private val currentScreen = BehaviorSubject.create<String>()

  private val loginMessage = BehaviorSubject.create("")
  private val authorizingMessage = BehaviorSubject.create<String>()
  private val secondFactorMessage = BehaviorSubject.create<String>()
```

```
class AuthWorkflow(): Workflow<Unit, String>,
    LoginScreen.Events, SecondFactorScreen.Events {

  private val currentScreen = BehaviorSubject.create<String>()

  private val loginMessage = BehaviorSubject.create("")
  private val authorizingMessage = BehaviorSubject.create<String>()
  private val secondFactorMessage = BehaviorSubject.create<String>()

  override fun screen(): Observable<WorkflowScreen<*,*> =
   currentScreen.map { it ->
    when (it) {
      LoginScreen.KEY -> LoginScreen(loginMessage, this)
      AuthorizingScreen.KEY -> AuthorizingScreen(authorizingMessage)
      SecondFactorScreen.KEY -> SecondFactorScreen(secondFactorMessage, this)
      else -> throw IllegalArgumentException("Unknown key " + it)
    }
  }
```

```
class AuthWorkflow(): Workflow<Unit, String>,
    LoginScreen.Events, SecondFactorScreen.Events {

  …
```

```
class AuthWorkflow(): Workflow<Unit, String>,
    LoginScreen.Events, SecondFactorScreen.Events {

  …

  override fun onLogin(event: LoginScreen.SubmitLogin) {
    stateMachine.onEvent(event)
  }

  override fun onSecondFactor(event: SecondFactorScreen.SecondFactor) {
    stateMachine.onEvent(event)
  }
```

```
internal enum class State {
  LOGIN_PROMPT, AUTHORIZING, SECOND_FACTOR_PROMPT, DONE
}
```

```
internal enum class State {
  LOGIN_PROMPT, AUTHORIZING, SECOND_FACTOR_PROMPT, DONE
}
init {
  stateMachine = FiniteStateMachine(

      onEntry(AUTHORIZING) { currentScreen.onNext(AuthorizingScreen.KEY) },
      onEntry(SECOND_FACTOR_PROMPT) {
        currentScreen.onNext(SecondFactorScreen.KEY)
      },
```

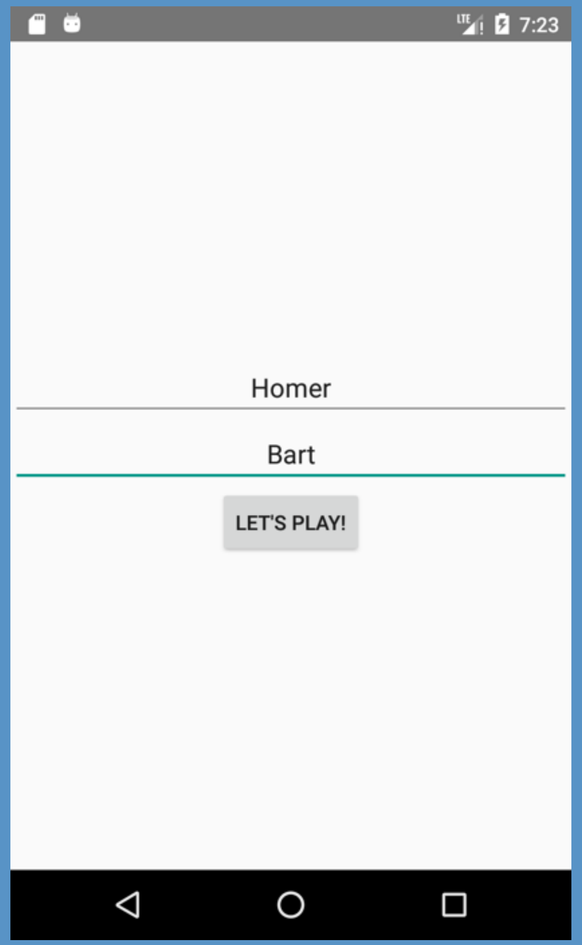
```
internal enum class State {
  LOGIN_PROMPT, AUTHORIZING, SECOND_FACTOR_PROMPT, DONE
}
init {
  stateMachine = FiniteStateMachine(
      …
```

```
internal enum class State {
  LOGIN_PROMPT, AUTHORIZING, SECOND_FACTOR_PROMPT, DONE
}
init {
  stateMachine = FiniteStateMachine(
      …

      transition(LOGIN_PROMPT, SubmitLogin::class, AUTHORIZING)
          .doAction { doLogin(it) },

      transition(AUTHORIZING, AuthResponse::class, LOGIN_PROMPT)
          .onlyIf { isLoginFailure(it) }
          .doAction { response ->
            val errorMessage = response.errorMessage
            loginMessage.onNext(errorMessage)
          },
```

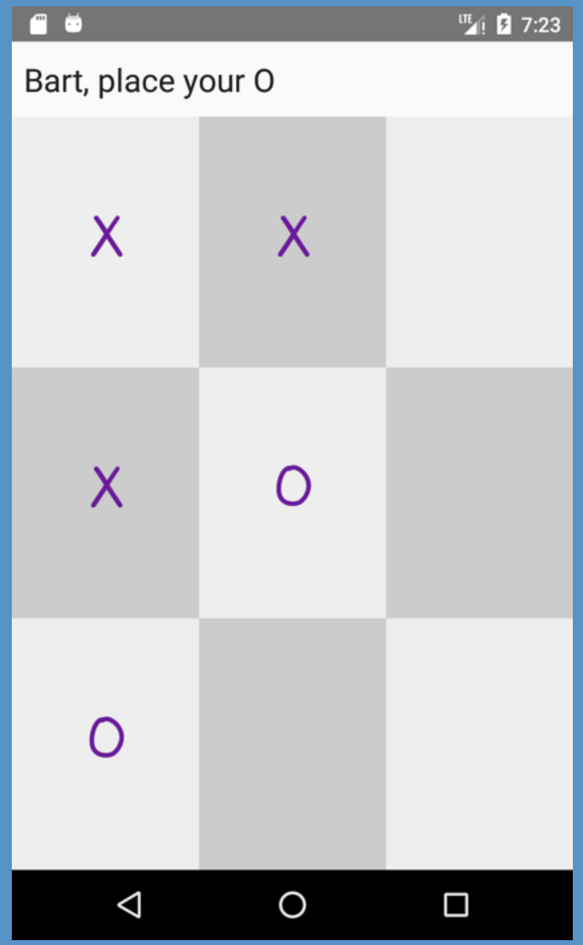
```
internal enum class State {
  LOGIN_PROMPT, AUTHORIZING, SECOND_FACTOR_PROMPT, DONE
}
init {
  stateMachine = Fi
      …

      transition(LO                    ORIZING)
          .doAction

      transition(AU                    N_PROMPT)
          .onlyIf {
          .doAction
            val err
            loginMes
          },
```

search for:

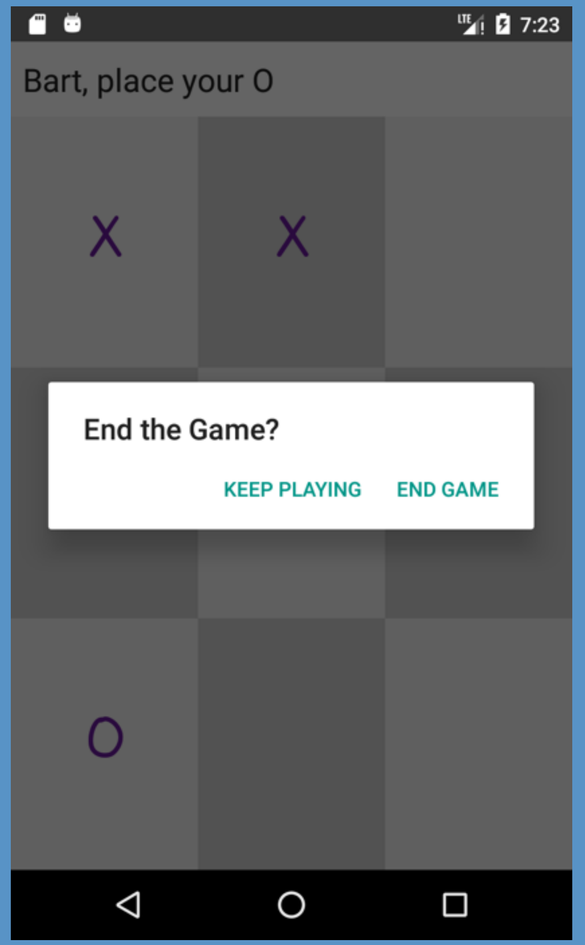
Andy Matuschak states



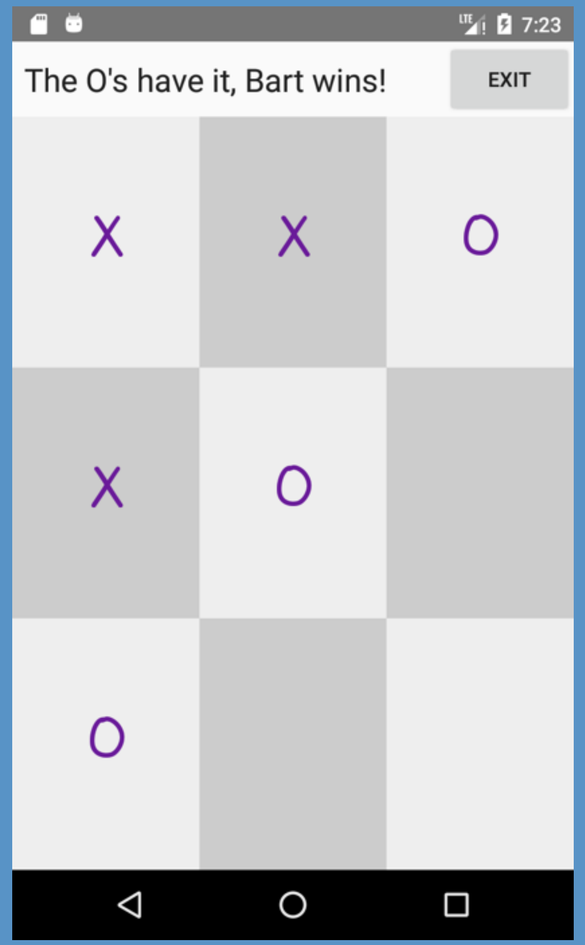
NewGameScreen



GamePlayScreen



ConfirmQuitScreen



GameOverScreen

```
class TicTacToeViewFactory private constructor()
  : AbstractViewFactory(asList(

    bindLayout(NewGameScreen.KEY, layout.new_game_layout
    ) { screen -> NewGameCoordinator(screen as NewGameScreen) },

    bindLayout(GamePlayScreen.KEY, layout.game_play_layout
    ) { screen -> GamePlayCoordinator(screen as GamePlayScreen) },

    bindLayout(GameOverScreen.KEY, layout.game_play_layout
    ) { screen -> GameOverCoordinator(screen as GameOverScreen) },

    bindDialog(ConfirmQuitScreen.KEY
    ) { screen -> ConfirmQuitDialogFactory(screen as ConfirmQuitScreen) }

))
```

```
class TicTacToeViewFactory private constructor()
  : AbstractViewFactory(asList(

    bindLayout(NewGameScreen.KEY, layout.new_game_layout
    ) { screen -> NewGameCoordinator(screen as NewGameScreen) },

    bindLayout(GamePlayScreen.KEY, layout.game_play_layout
    ) { screen -> GamePlayCoordinator(screen as GamePlayScreen) },

    bindLayout(GameOverScreen.KEY, layout.game_play_layout
    ) { screen -> GameOverCoordinator(screen as GameOverScreen) },

    bindDialog(ConfirmQuitScreen.KEY
    ) { screen -> ConfirmQuitDialogFactory(screen as ConfirmQuitScreen) }

))
```

```
class TicTacToeViewFactory private constructor()
  : AbstractViewFactory(asList(

    bindLayout(NewGameScreen.KEY, layout.new_game_layout
    ) { screen -> NewGameCoordinator(screen as NewGameScreen) },

    bindLayout(GamePlayScreen.KEY, layout.game_play_layout
    ) { screen -> GamePlayCoordinator(screen as GamePlayScreen) },

    bindLayout(GameOverScreen.KEY, layout.game_play_layout
    ) { screen -> GameOverCoordinator(screen as GameOverScreen) },

    bindDialog(ConfirmQuitScreen.KEY
    ) { screen -> ConfirmQuitDialogFactory(screen as ConfirmQuitScreen) }

))
```

```
class TicTacToeWorkflow(
  private val gameRunner: GameRunner
): Workflow<Unit, TicTacToeGameState>
```

```
class TicTacToeWorkflow(
  private val gameRunner: GameRunner
): Workflow<Unit, TicTacToeGameState>,
    NewGameScreen.Events, GamePlayScreen.Events,
    ConfirmQuitScreen.Events, GameOverScreen.Events {
```

```
class TicTacToeWorkflow(
  private val gameRunner: GameRunner
): Workflow<Unit, TicTacToeGameState>,
    NewGameScreen.Events, GamePlayScreen.Events,
    ConfirmQuitScreen.Events, GameOverScreen.Events {

  private val gameStates: Observable<TicTacToeGameState> =
      gameRunner.gameState().startWith(NO_GAME)
```

```
class TicTacToeWorkflow(
  private val gameRunner: GameRunner
): Workflow<Unit, TicTacToeGameState>,
    NewGameScreen.Events, GamePlayScreen.Events,
    ConfirmQuitScreen.Events, GameOverScreen.Events {

  private val gameStates: Observable<TicTacToeGameState> =
      gameRunner.gameState().startWith(NO_GAME)

  companion object {
    private val FAKE_ID = UUID.randomUUID().toString()

    private val NO_GAME = TicTacToeGameState.newGame(FAKE_ID,
        Player(FAKE_ID, "X"), Player(FAKE_ID, "O"))
  }
```

```
class TicTacToeWorkflow(
  private val gameRunner: GameRunner
): Workflow<Unit, TicTacToeGameState>,
    NewGameScreen.Events, GamePlayScreen.Events,
    ConfirmQuitScreen.Events, GameOverScreen.Events {

  private val gameStates: Observable<TicTacToeGameState> =
      gameRunner.gameState().startWith(NO_GAME)

  …
```

```
class TicTacToeWorkflow(
  private val gameRunner: GameRunner
): Workflow<Unit, TicTacToeGameState>,
    NewGameScreen.Events, GamePlayScreen.Events,
    ConfirmQuitScreen.Events, GameOverScreen.Events {

  private val gameStates: Observable<TicTacToeGameState> =
      gameRunner.gameState().startWith(NO_GAME)

  private val quitting = BehaviorSubject.create(false)

  …
```

```
class TicTacToeWorkflow(
  private val gameRunner: GameRunner
): Workflow<Unit, TicTacToeGameState>,
    NewGameScreen.Events, GamePlayScreen.Events,
    ConfirmQuitScreen.Events, GameOverScreen.Events {

  private val gameStates: Observable<TicTacToeGameState> =
      gameRunner.gameState().startWith(NO_GAME)

  private val quitting = BehaviorSubject.create(false)

  private val screen = combineLatest(gameStates, quitting,
      { gameState, quitting -> update(gameState, quitting) })
     .replay(1)

  override fun screen(): Observable<WorkflowScreen<*,*>> = screen

  …
```

```
class TicTacToeWorkflow(
  private val gameRunner: GameRunner
): Workflow<Unit, TicTacToeGameState>,
    NewGameScreen.Events, GamePlayScreen.Events,
    ConfirmQuitScreen.Events, GameOverScreen.Events {
  …
```

```
class TicTacToeWorkflow(
  private val gameRunner: GameRunner
): Workflow<Unit, TicTacToeGameState>,
    NewGameScreen.Events, GamePlayScreen.Events,
    ConfirmQuitScreen.Events, GameOverScreen.Events {
  …

  private fun update(gameState: GameState, quitting: Boolean):
      WorkflowScreen<*,*> {
    if (quitting) return ConfirmQuitScreen(this)
    if (gameState == NO_GAME) NewGameScreen(this)

    return when (gameState.stateOfPlay) {
      PLAYING -> GamePlayScreen(gameStates, this)
      VICTORY, DRAW -> GameOverScreen(gameStates, this)
    }
  }

  …
```

RootView

Container

View Factory

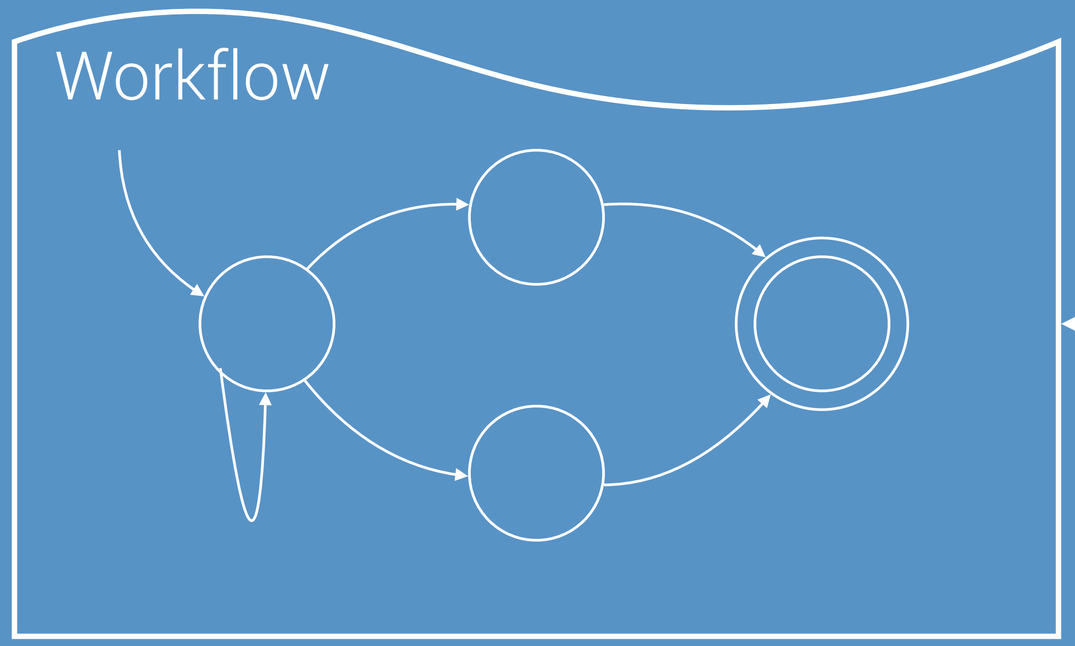
Workflow

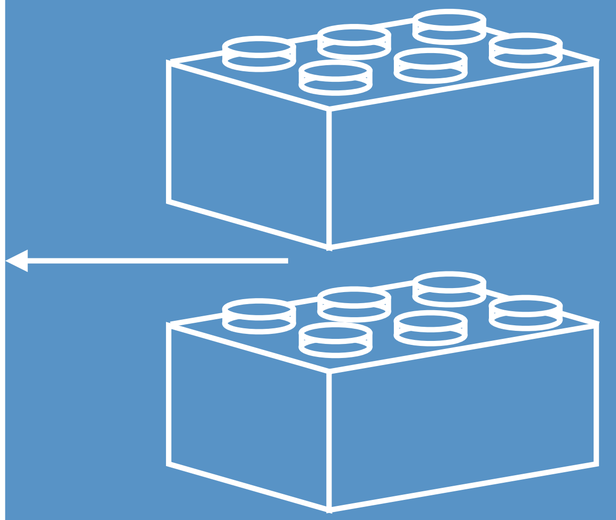
RootView

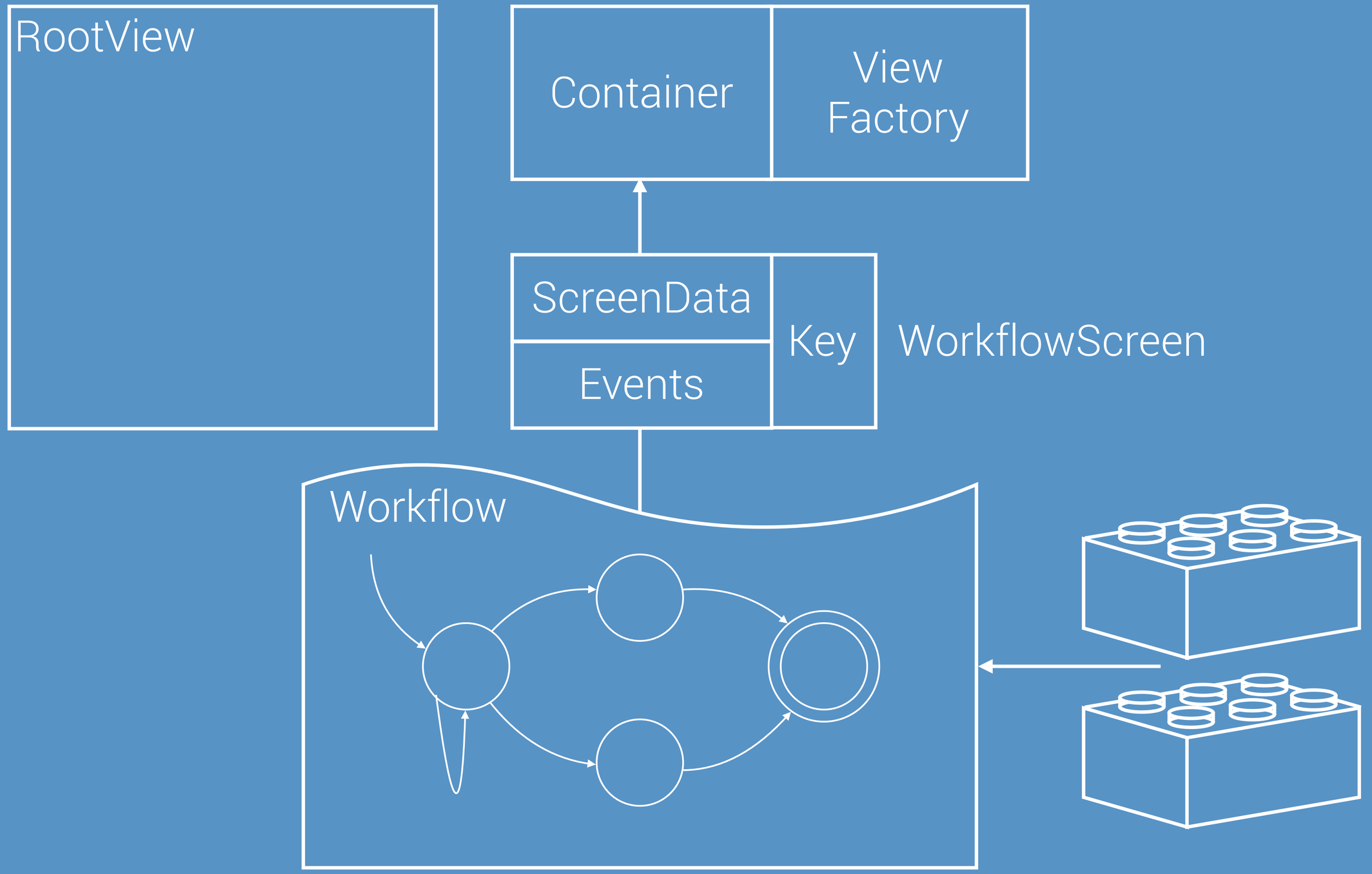
Container

View Factory

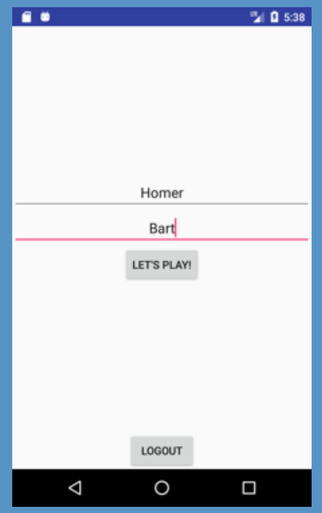
Workflow






RootView
Container
View Factory
ScreenData
Events
Key
WorkflowScreen
Workflow

RootView



Container

View Factory

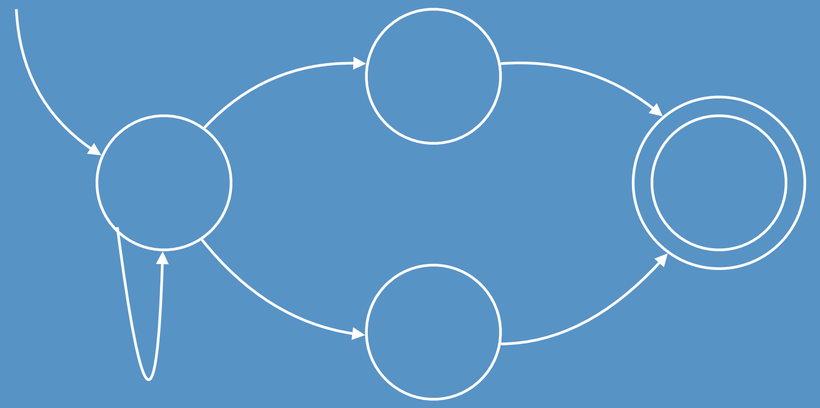
ScreenData

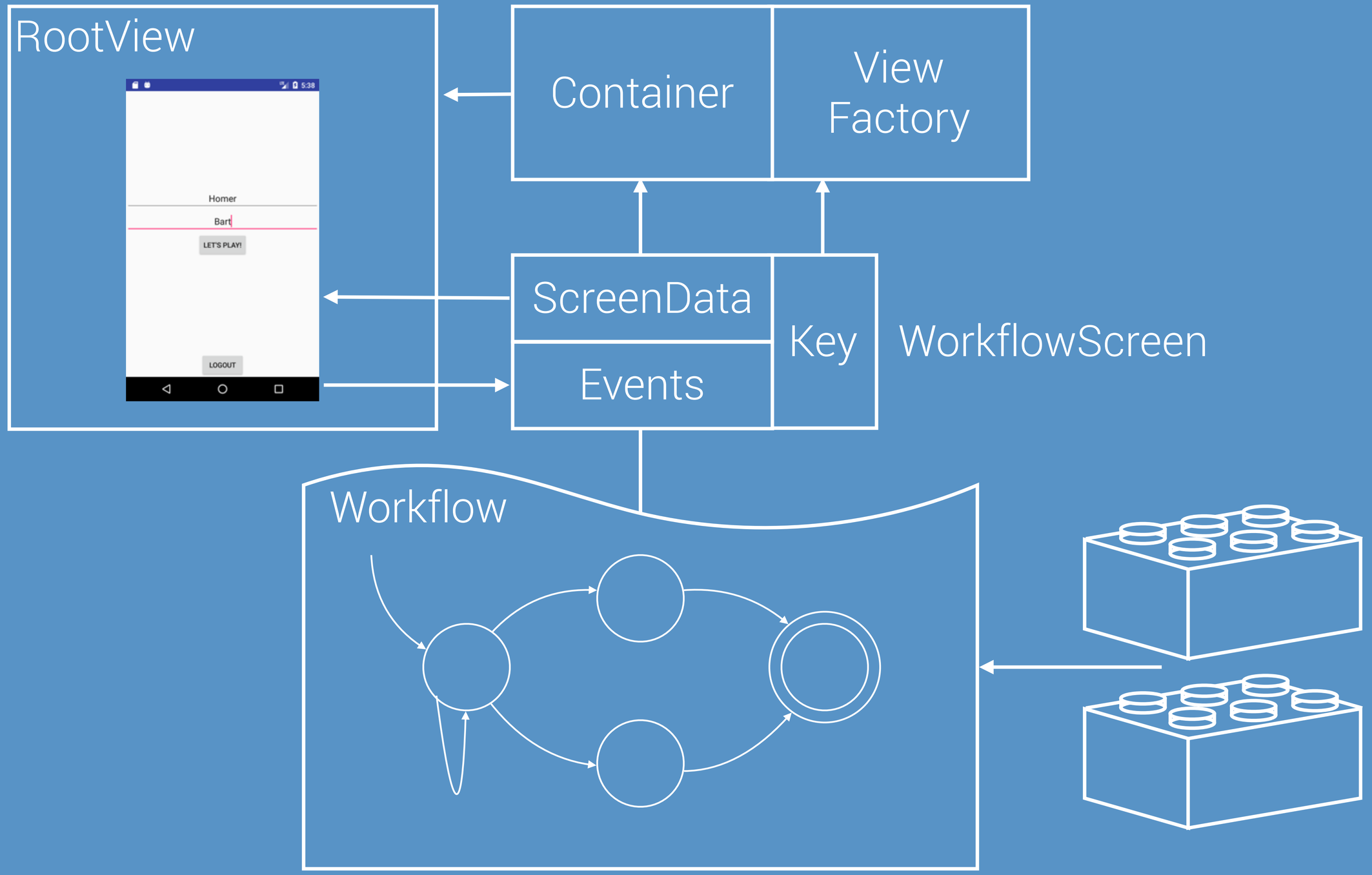
Events

Key

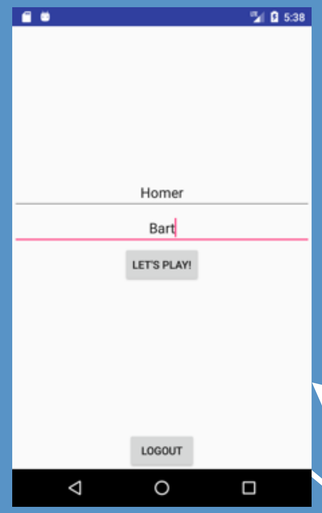
WorkflowScreen

Workflow




RootView
5:38
Homer
Bart
LET'S PLAY!
LOGOUT
Container
View Factory
ScreenData
Events
Key
WorkflowScreen
Workflow

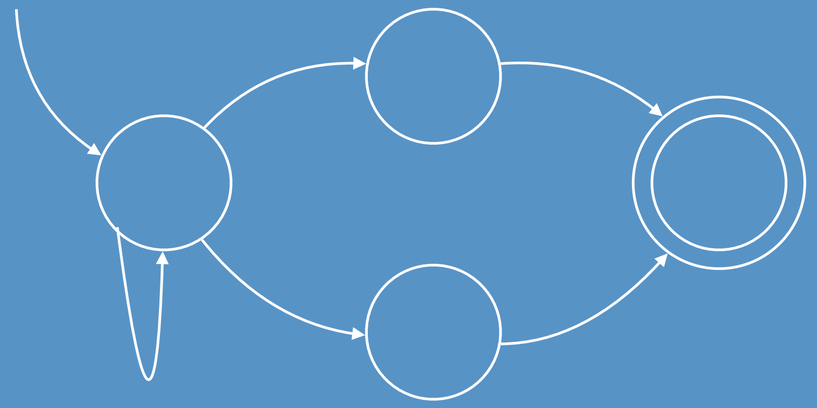
RootView



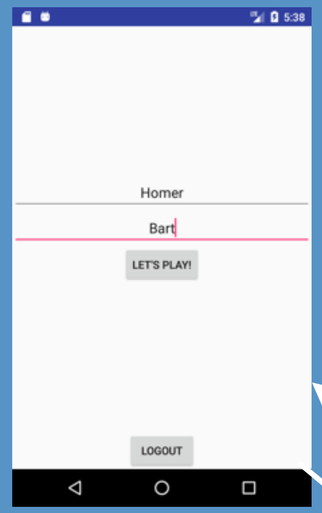
Container

View Factory

Workflow



RootView



Container

View Factory

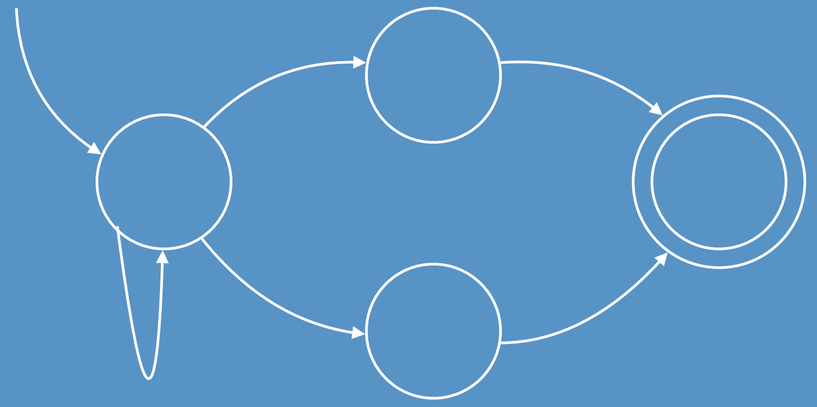
ScreenData

Events

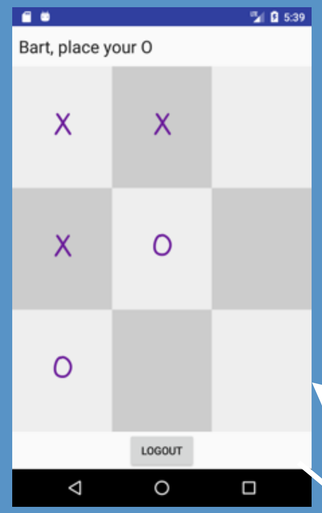
Key

WorkflowScreen

Workflow



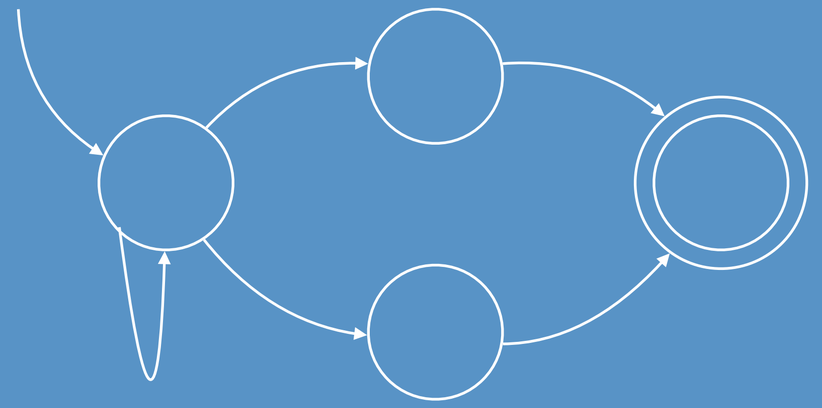
RootView



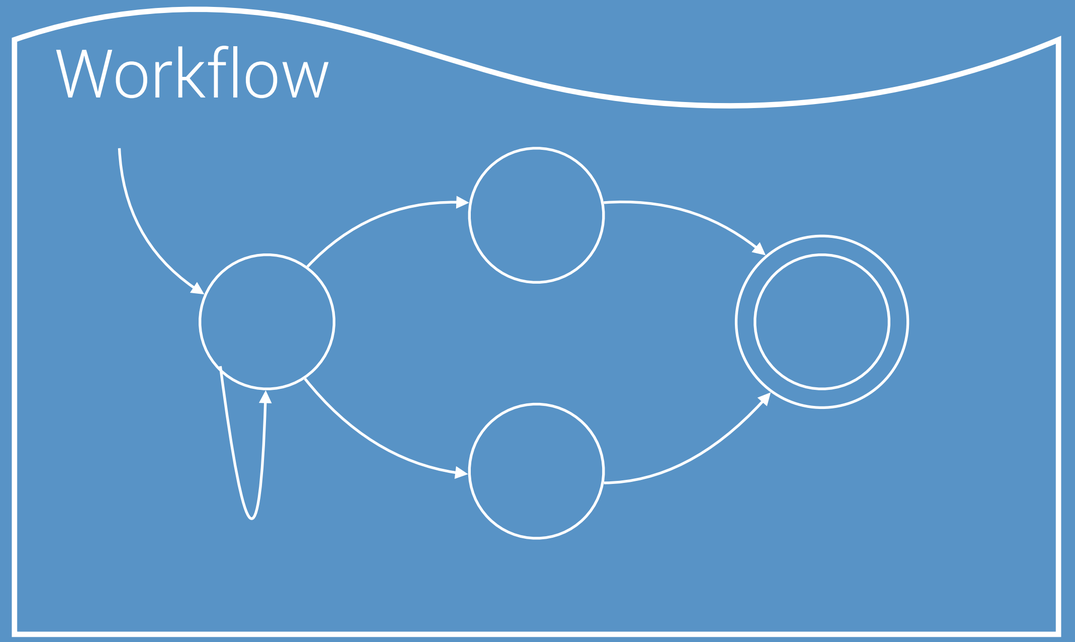
Container

View Factory

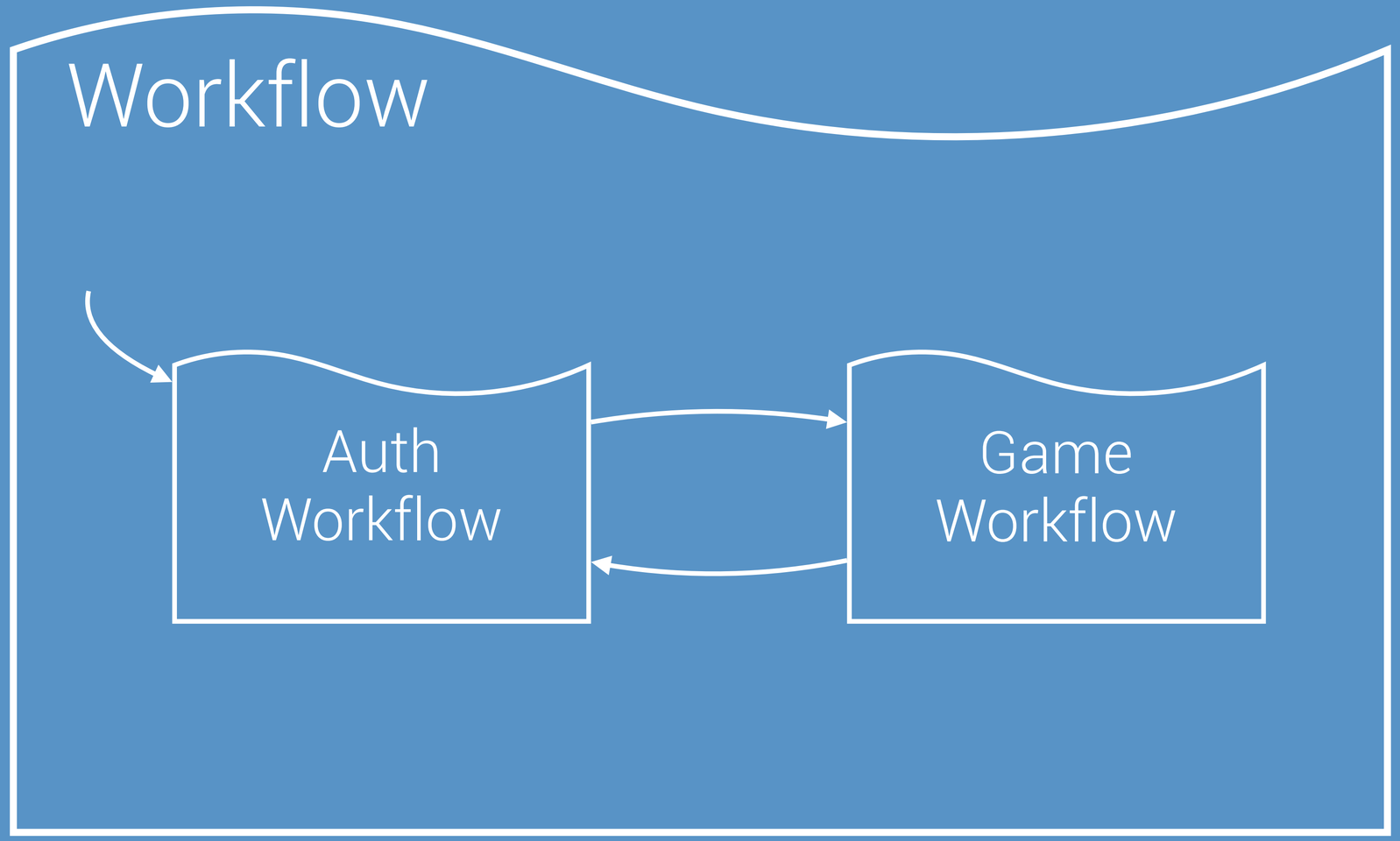
Workflow



You said these things compose



You said these things compose



```
new CompositeWorkflow<>(
    // Start in the AuthWorkflow. When it finishes, kick off
    // a TicTacToe game.
    new WorkflowBinding<>(AuthWorkflow.class,
        () -> ignoreStartArg(authWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT))),

    // When a TicTacToe game ends, start another one.
    new WorkflowBinding<>(TicTacToeWorkflow.class,
        () -> ignoreStartArg(ticTacToeWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT)))
)
```

```
new CompositeWorkflow<>(
    // Start in the AuthWorkflow. When it finishes, kick off
    // a TicTacToe game.
    new WorkflowBinding<>(AuthWorkflow.class,
        () -> ignoreStartArg(authWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT))),

    // When a TicTacToe game ends, start another one.
    new WorkflowBinding<>(TicTacToeWorkflow.class,
        () -> ignoreStartArg(ticTacToeWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT)))
)
```

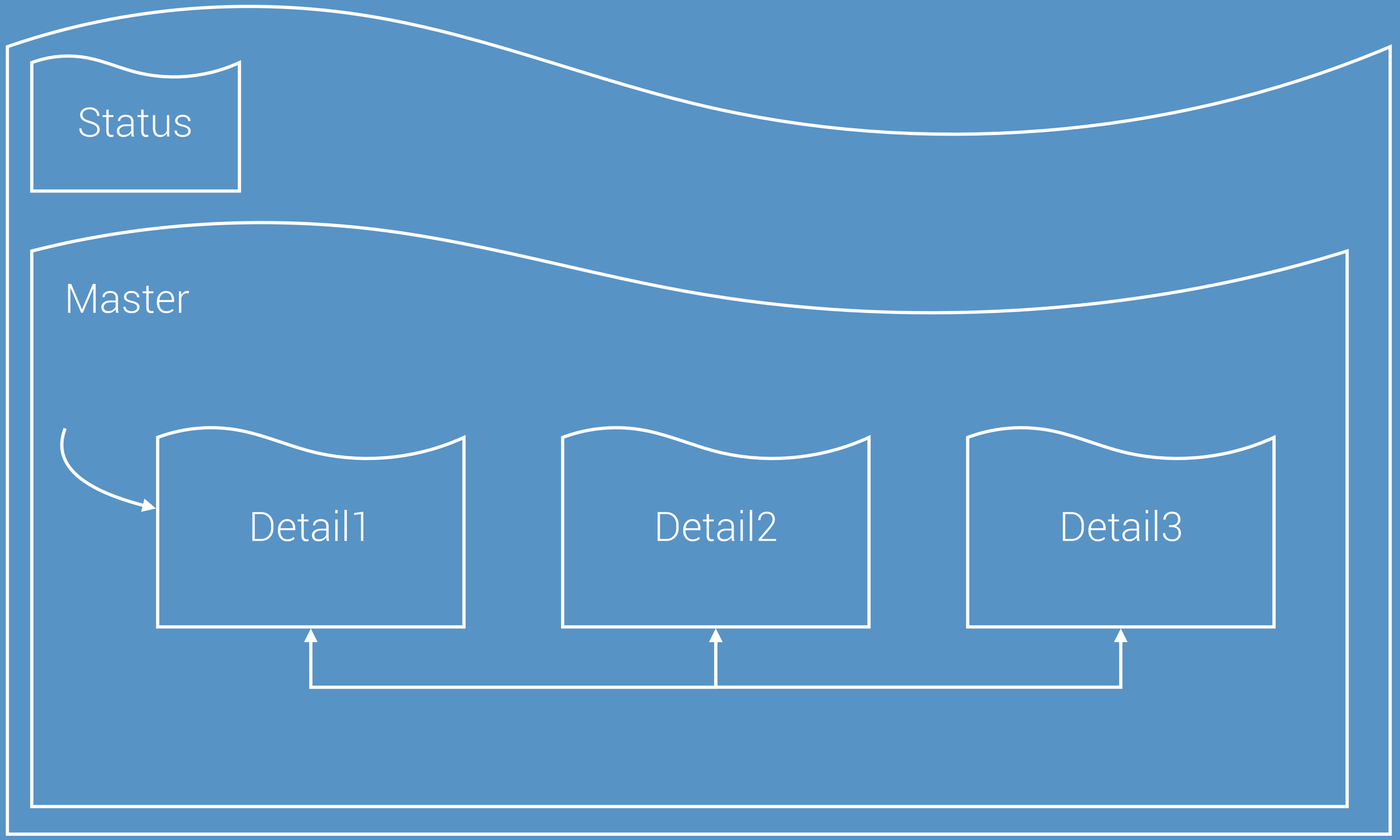
```
new CompositeWorkflow<>(
    // Start in the AuthWorkflow. When it finishes, kick off
    // a TicTacToe game.
    new WorkflowBinding<>(AuthWorkflow.class,
        () -> ignoreStartArg(authWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT))),

    // When a TicTacToe game ends, start another one.
    new WorkflowBinding<>(TicTacToeWorkflow.class,
        () -> ignoreStartArg(ticTacToeWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT)))
)
```

```
new CompositeWorkflow<>(
    // Start in the AuthWorkflow. When it finishes, kick off
    // a TicTacToe game.
    new WorkflowBinding<>(AuthWorkflow.class,
        () -> ignoreStartArg(authWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT))),

    // When a TicTacToe game ends, start another one.
    new WorkflowBinding<>(TicTacToeWorkflow.class,
        () -> ignoreStartArg(ticTacToeWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT)))
)
```

```
new CompositeWorkflow<>(
    // Start in the AuthWorkflow. When it finishes, kick off
    // a TicTacToe game.
    new WorkflowBinding<>(AuthWorkflow.class,
        () -> ignoreStartArg(authWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT))),

    // When a TicTacToe game ends, start another one.
    new WorkflowBinding<>(TicTacToeWorkflow.class,
        () -> ignoreStartArg(ticTacToeWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT)))
)
```

```
new CompositeWorkflow<>(
    // Start in the AuthWorkflow. When it finishes, kick off
    // a TicTacToe game.
    new WorkflowBinding<>(AuthWorkflow.class,
        () -> ignoreStartArg(authWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT))),

    // When a TicTacToe game ends, start another one.
    new WorkflowBinding<>(TicTacToeWorkflow.class,
        () -> ignoreStartArg(ticTacToeWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT)))
)
```

```
new CompositeWorkflow<>(
    // Start in the AuthWorkflow. When it finishes, kick off
    // a TicTacToe game.
    new WorkflowBinding<>(AuthWorkflow.class,
        () -> ignoreStartArg(authWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT))),

    // When a TicTacToe game ends, start another one.
    new WorkflowBinding<>(TicTacToeWorkflow.class,
        () -> ignoreStartArg(ticTacToeWorkflowProvider.get()),
        (composite, result) -> composite.startWorkflow(
            forArg(TicTacToeWorkflow.class, (Unit) UNIT)))
)
```


Status
Master
Detail1
Detail2
Detail3

Tactics

Step zero: make refactoring...

...possible

- Mock service layer
- Robot-based UI tests (espresso, KIF)

...tolerable

- OkBuck for Android
- CocoaPods for iOS

Divide first, conquer later

Always be writing

PSAs

Design docs (everyone, all the time)

Policy docs (a few, mostly: “write a damn design doc”)

How-to guides and sample code

Full disclosure

Brick pattern established on both

First j2Objc brick entering production

Non-toy workflows in development on both

Composite Workflow on Android

Composite WorkflowScreen TBD

@rjrjr

speakerdeck.com/rjrjr/where-the-reactive-rubber-meets-the-road

Dan Lew transformer

github.com/square/coordinators

Andy Matuschak states (http://bfy.tw/E969)

square.com/jobs