(initial slide)

MOBIUS.SWIFT

Erik Price
iOS Developer
Personalization, Spotify

SCALE

» Modularized code, but large number of teams working in the same codebase

» Cross-platform consistency

» Teams of "T-shaped" developers

» Test maintenance

BUT REALLY

ARCHITECTURES

» "MVC" with UIViewControllers

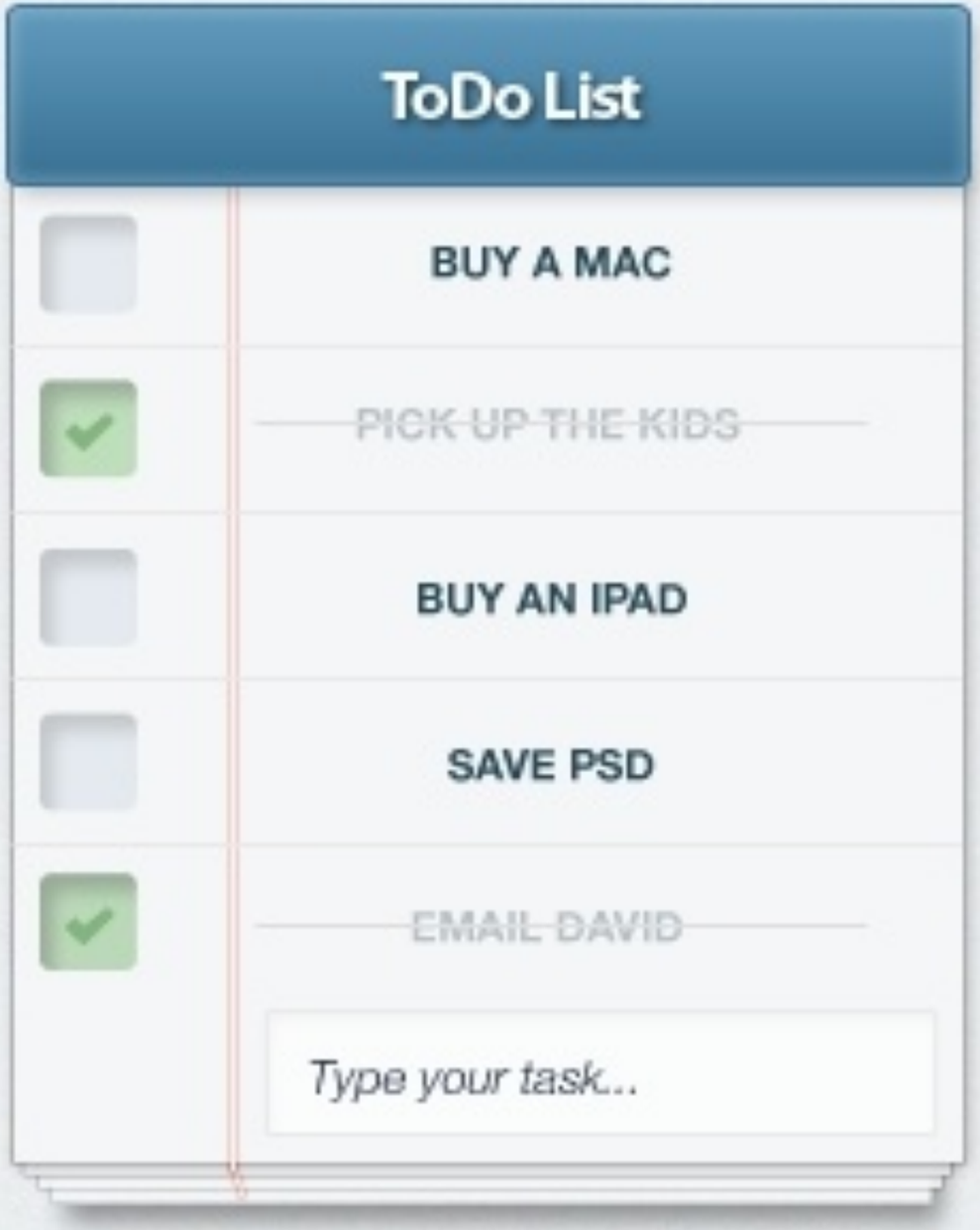
» Presentation Coordinators

» MVVM

» RIBs etc

» React Native/Flutter etc

"BUSINESS LOGIC"


ToDo List
BUY A MAC
PICK UP THE KIDS
BUY AN IPAD
SAVE PSD
EMAIL DAVID
Type your task...

"MOBIUS FLOW"

» What do we show

» What can users do

» What happens

"MOBIUS FLOW"

Starting from a UI design or sketch

» What do we show - Model

» What can users do - Events

» What happens - Effects

- Model
  - List of items

» Model

  » List of items

» Events

  » Checked an item's checkbox

- Model
  - List of items
- Events
  - Checked an item's checkbox
- Effects
  - Update item to "completed" state on server

» Model

  » List of items

» Events

  » Checked an item's checkbox

  » Requested creation of a new item

» Effects

  » Update item to "completed" state on server

- Model
  - List of items
- Events
  - Checked an item's checkbox
  - Requested creation of a new item
- Effects
  - Update item to "completed" state on server
  - Create a new item in "uncommitted" state locally

» Model

  » List of items

  » Uncommitted item

» Events

  » Checked an item's checkbox

  » Requested creation of a new item

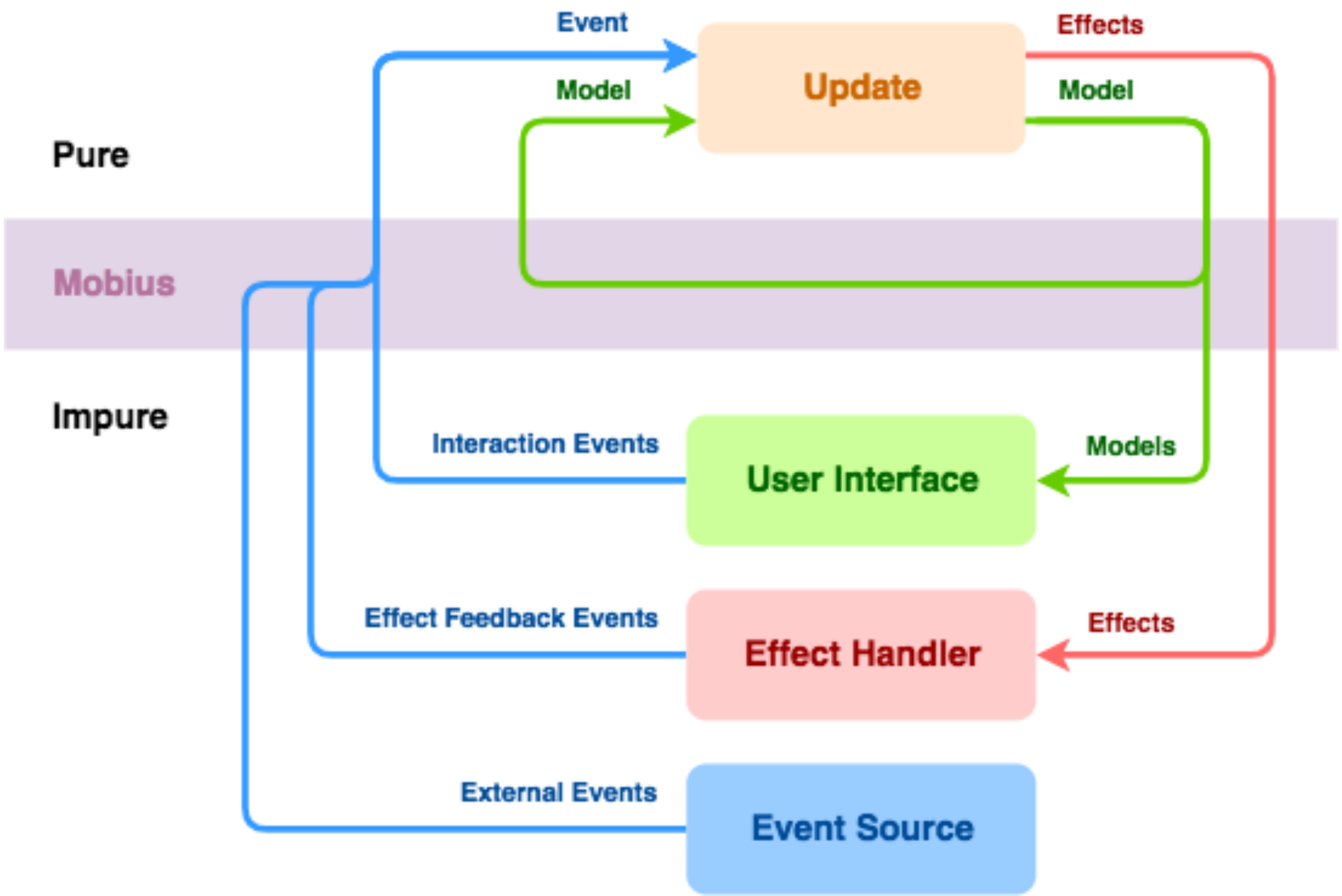
» Effects

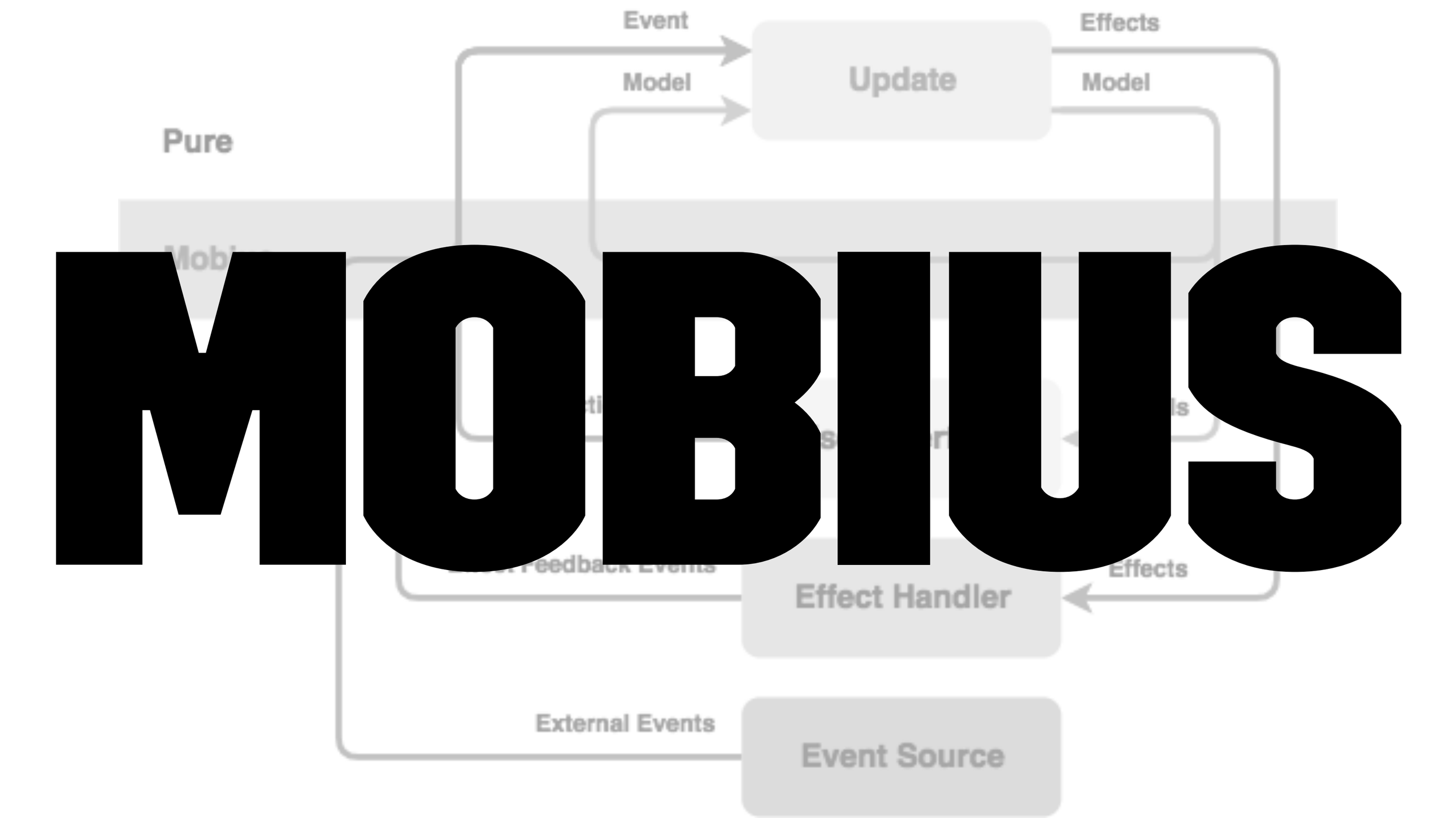
  » Update item to "completed" state on server

  » Create a new item in "uncommitted" state locally

- Events
  - Checked an item's checkbox
  - Requested creation of a new item
  - Finished editing an item
- Effects
  - Update item to "completed" state on server
  - Create a new item in "uncommitted" state locally

- Events
  - Checked an item's checkbox
  - Requested creation of a new item
  - Finished editing an item
- Effects
  - Update item to "completed" state on server
  - Create a new item in "uncommitted" state locally
  - Commit an item being edited to server


Event
Effects
Update
Model
Model
Pure
Mobius
Impure
Interaction Events
User Interface
Models
Effect Feedback Events
Effect Handler
Effects
External Events
Event Source


Event
Effects
Update
Model
Model
Pure
Effect Handler
Effects
External Events
Event Source

MOBIUS

Convert Mobius Flow output into a domain:

```
struct TodoAppModel { }

enum TodoAppEvent { }

enum TodoAppEffect { }
```

» Model

  » List of items

  » Uncommitted item

```
struct TodoAppModel {
    var items: [TodoItem]
    var uncommittedItem: TodoItem?
}
```

» Events

  » Checked an item's checkbox

  » Requested creation of a new item

  » Finished editing an item

```
enum TodoAppEvent {
    case checked(TodoItem)
    case requestedCreateNewItem
    case finishedEditing(TodoItem)
}
```

» Effects

  » Update item to "completed" state on server

  » Create a new item in "uncommitted" state locally

  » Commit an item being edited to server

```
enum TodoAppEffect {
    case completeOnServer(TodoItem)
    case createNewItem
    case commitEditingToServer(String, TodoItem)
}
```

WRITING DOMAIN TESTS IS PART OF MOBIUS FLOW

```
describe("Checked an item's checkbox") {
  context("item is not yet completed") {
    it("should render as completed in the UI") {
```

WRITING DOMAIN TESTS IS PART OF MOBIUS FLOW

```
describe("Checked an item's checkbox") {
  context("item is not yet completed") {
    it("should render as completed in the UI") { }
    it("should produce a 'complete' TodoAppEffect"
       "for the item") {
```

WRITING TESTS EXPOSES HOLES IN YOUR DOMAIN

```
describe("Checked an item's checkbox") {
  context("item is not yet completed") { ... }
  context("item is already completed") {
    it("should render as uncompleted in the UI") { }
    it("should produce an 'uncomplete' TodoAppEffect"
       "for the item") { }
```

LOGIC (FINISHING THE DOMAIN)

```
update(model: TodoAppModel,
         event: TodoAppEvent)

              -> Next<TodoAppModel, TodoAppEffect>
```

UPDATE FUNCTION RETURNS

Next<M, E>

```
public struct Next<Model, Effect>
        where Effect: Hashable {

    public let model: Model?
    public let effects: Set<Effect>
}
```

```
func update(model: TodoAppModel,
            event: TodoAppEvent)
                -> Next<TodoAppModel, TodoAppEffect> {
  switch event {
    case .checked(let item):
      guard !item.completed else { return .noChange }

      var updatedItem = item
      updatedItem.completed = true

      var updatedModel = model
      updatedModel.updateItem(item: updatedItem)

      let completeEffect = .completeOnServer(updatedItem)
      return .next(updatedModel, effects: [completeEffect])
```

```
func update(model: TodoAppModel,
            event: TodoAppEvent)
                -> Next<TodoAppModel, TodoAppEffect> {
  switch event {
    case .checked(let item):
      ...
    case .requestedCreateNewItem:
      let newItem = TodoItem()

      var updatedModel = model
      updatedModel.addItem(item: newItem)

      return .next(updatedModel, effects: [.createNewItem])
```

HANDLING EFFECTS

» Implement handle(_ input: InputType) -> Void

» Perform whatever side-effects occur as a result

» InputType is an associated type of the Connectable protocol

  » Your effect type is typealiased to it

```
class TodoAppEffectHandler:
    ConnectableClass<TodoAppEffect, TodoAppEvent> {
```

```
override func handle(_ effect: TodoAppEffect) {
  switch effect {
    case .completeOnServer(let completedItem)
      serverHandler.complete(completedItem)
    case .createNewItem:
      localStorageHandler.createNewItem()
    case .commitEditingToServer(let text,
                                let editedItem):
      serverHandler.commitEditing(text: text,
                                  item: editedItem)
}
```

WIRING IT UP TO YOUR APP

```
enum TodoAppLoopTypes: LoopTypes {
   typealias Model = TodoAppModel
   typealias Event = TodoAppEvent
   typealias Effect = TodoAppEffect
}
```

```
override func viewWillAppear(_ animated: Bool) {
  super.viewWillAppear(animated)

  let effectHandler = TodoAppEffectHandler(
    serverHandler: ServerHandler(authService, todoListService),
    localStorageHandler: LocalStorageHandler(userDefaults))

  let initialModel = TodoAppModel(userDefaults.readTodoState())

  let loop: MobiusLoop<TodoAppLoopTypes> =
    Mobius.loop(update: update,
                effectHandler: effectHandler)
          .start(initialModel)

  self.loop = loop
```

COMPOSING MOBIUS LOOPS

- Update function composition
  - "Parent" update function calls a "child" update function and examines the Next<Model, Effect> that it gets back
- Loop composition
  - Loops don't know about each other (domains are more "pure")
  - Wiring code (outside of domain) communicates events between loops

SUMMARY

» Business logic

» Mobius Flow (Model/Events/Effects)

» Keep your update() pure

» Cross-platform

» https://github.com/spotify/mobius.swift