From Startup to Scale

The Engineering Playbook for Growing Android Teams

Eric Maxwell : Block

Focus

High Level

- How to work effectively in a large codebase
- Share code
- Maintain consistent architecture
- Tips for tackling large refactors

Focus

High Level

- How to work effectively in a large codebase
- Share code
- Maintain consistent architecture
- Tips for tackling large refactors

Non Focus

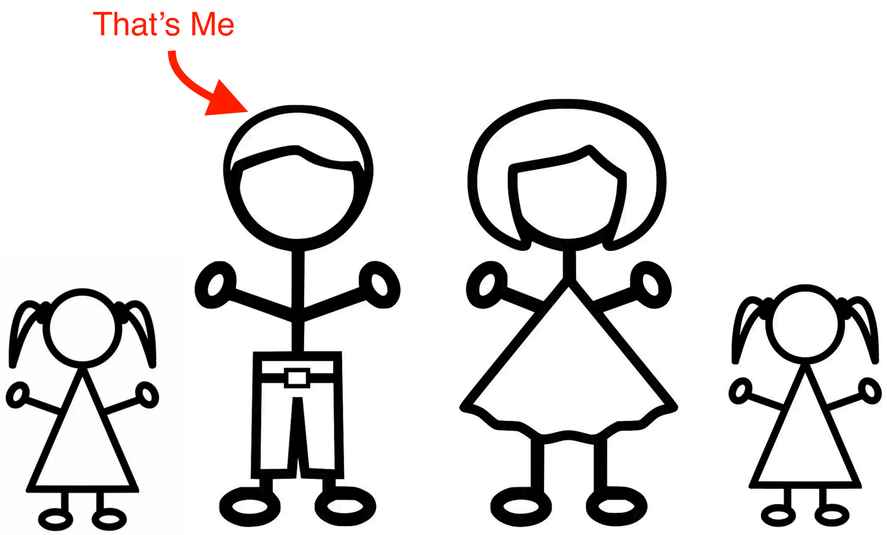
Low Level

- Implementation Specifics

Hello

About me

- Columbus, OH
- 20 Years Engineering
- Block 4 years
- Square Mobile Android Platform
- Previously Big Nerd Ranch, Realm, Consulting
- 2 Kids, 1 Partner, 1 Dog



Scaling Problem

Block

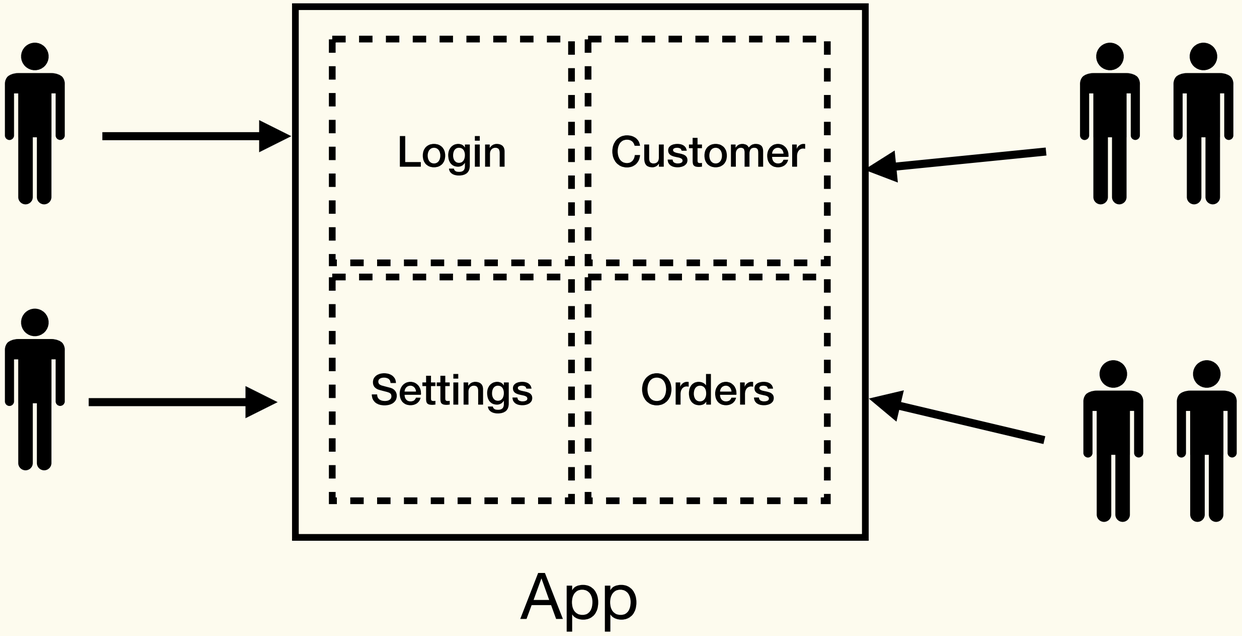
- ~200 Android Developers
- ~2M Lines of code
- ~5400 Gradle modules
- ~10 Apps
- ~4K Modules per app

Scaling Problem

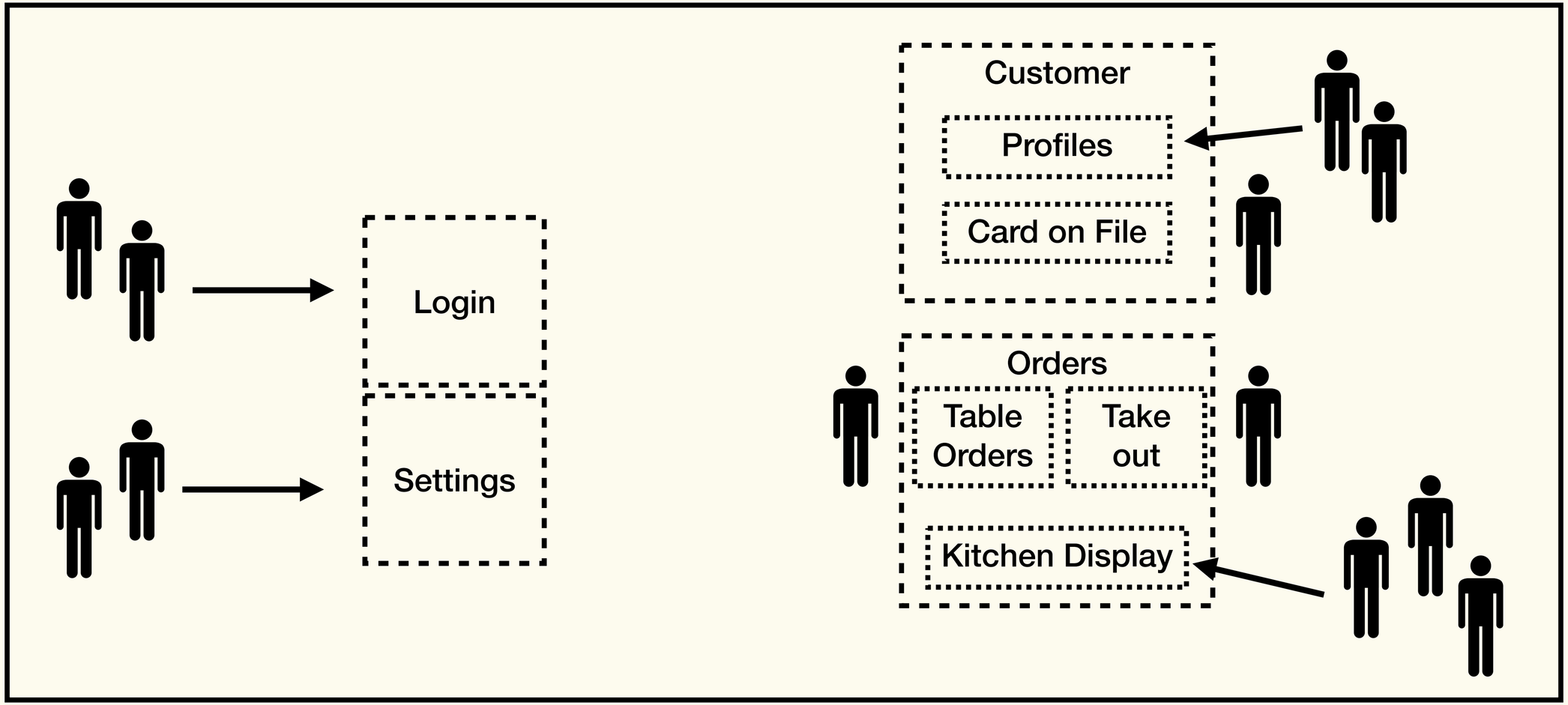
Your Team

- ~6 Android Developers
- ~10K Lines of code
- 1 Gradle modules
- 1 Apps

Small Team

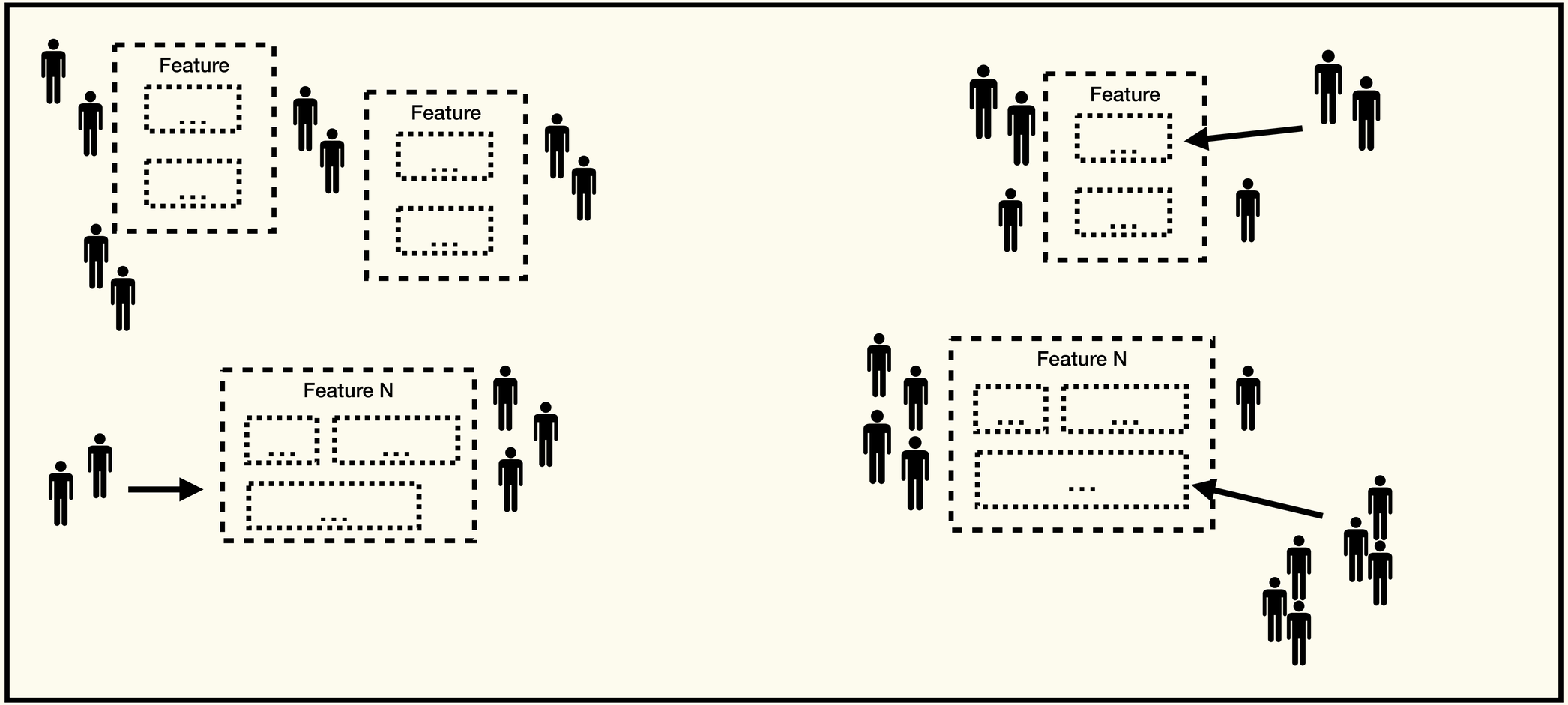


Medium Team



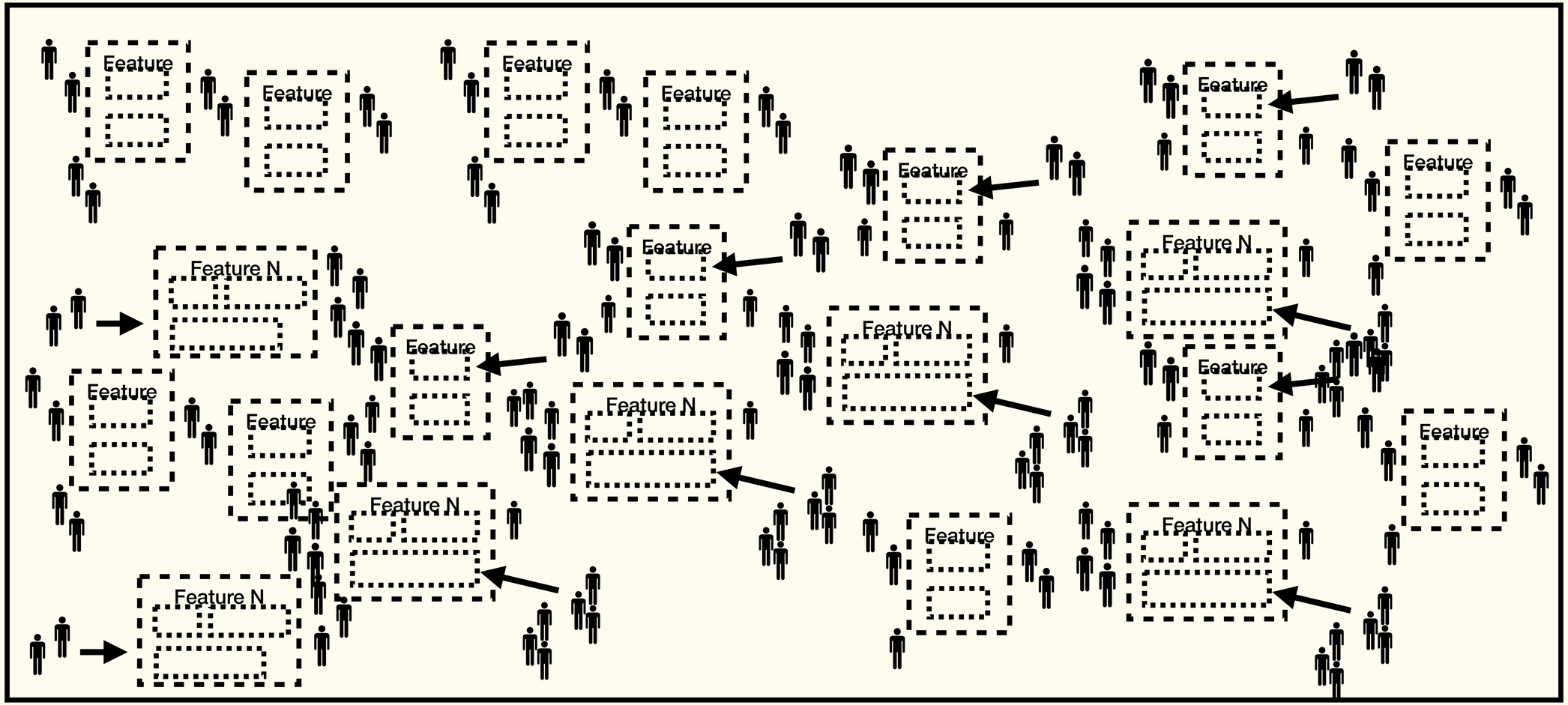
App

Larger Team



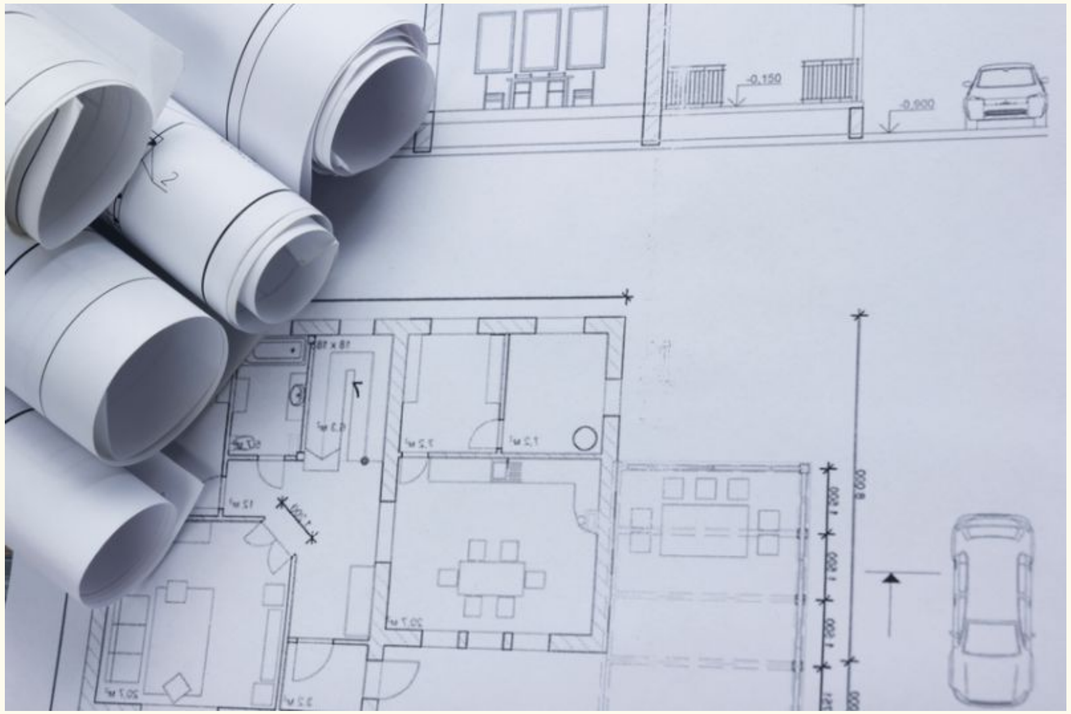
App

Even Larger Team



App

Architecture + Processes



Architectural Pillars

Modularity

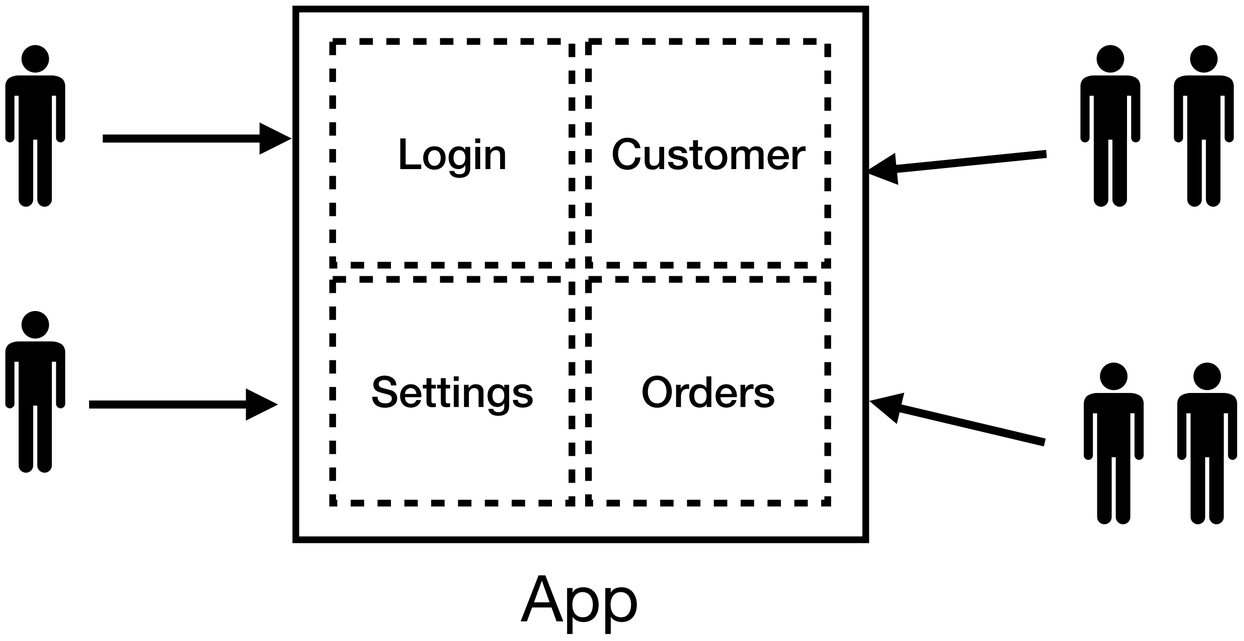
Dependency Injection

UI Framework

Dependency Injection

Dependency Injection

- **Why**
  - Easy to share code
  - Depend on API not the impl
- **Scale**
  - Small



Architecture

Dependency Injection

```
class RealCheckoutFlow
@Inject constructor(): CheckoutFlow { ... }
```

Architecture

Code to Interface

```
class RealCheckoutFlow
@Inject constructor(): CheckoutFlow { ... }
```

Architecture

Code to Interface

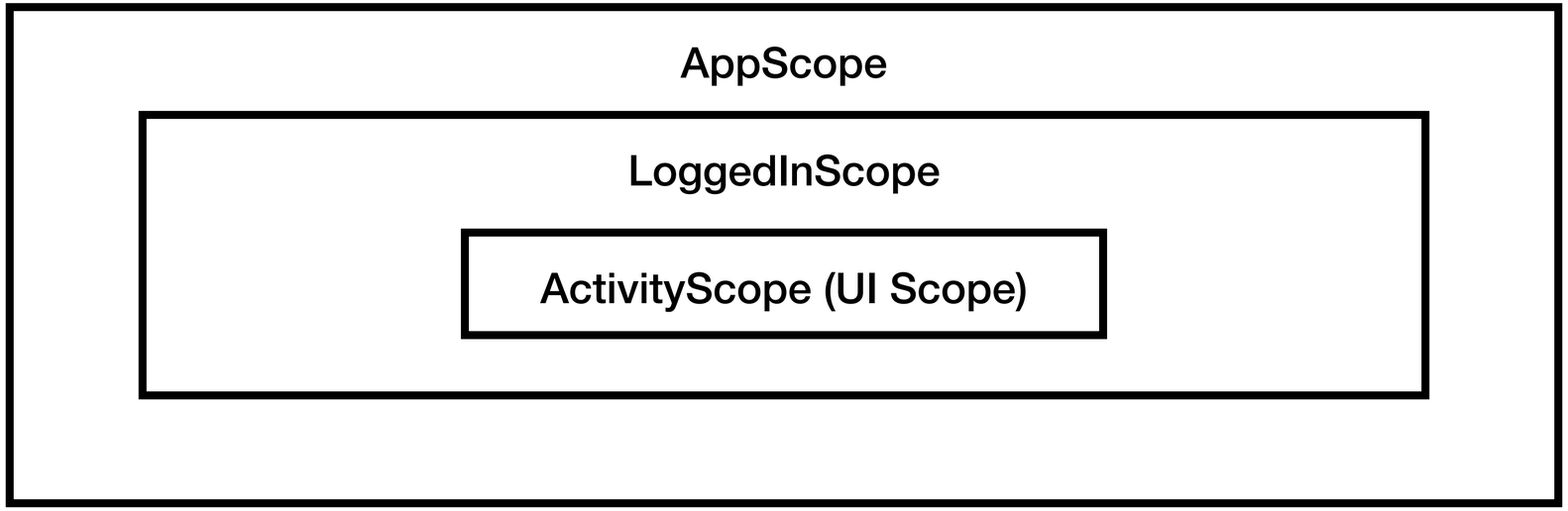
```
class RealCheckoutFlow
@Inject constructor(
  private val paymentEngine: PaymentEngine
): CheckoutFlow { ... }
```

- Depend on other APIs without committing to the implementation
- Swap implementations based on platform, app flavor
- Allows us to inject fakes for tests or small demo apps.

Architecture

Establish Standard Scopes & Lifecycles

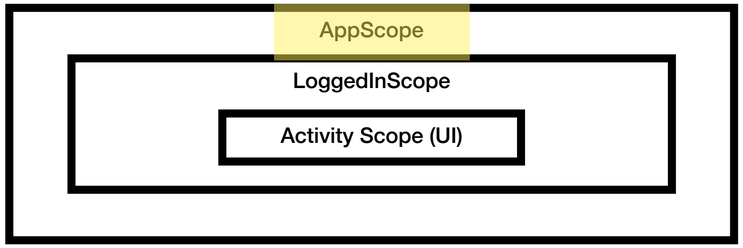
```
class RealCheckoutFlow
@Inject constructor(
  private val paymentEngine: PaymentEngine
): CheckoutFlow { ... }
```



Architecture

AppScope

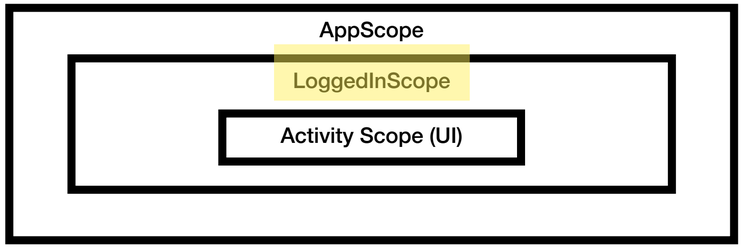
```
class RealCheckoutFlow
@Inject constructor(
  private val paymentEngine: PaymentEngine,
  private val featureFlagsClient: FeatureFlagsClient
  @AuthenticatedConnectClient client: OkHttpClient,
  @Main mainContext: CoroutineContext,
  @Computation computationContext: CoroutineContext,
): CheckoutFlow { ... }
```



Architecture

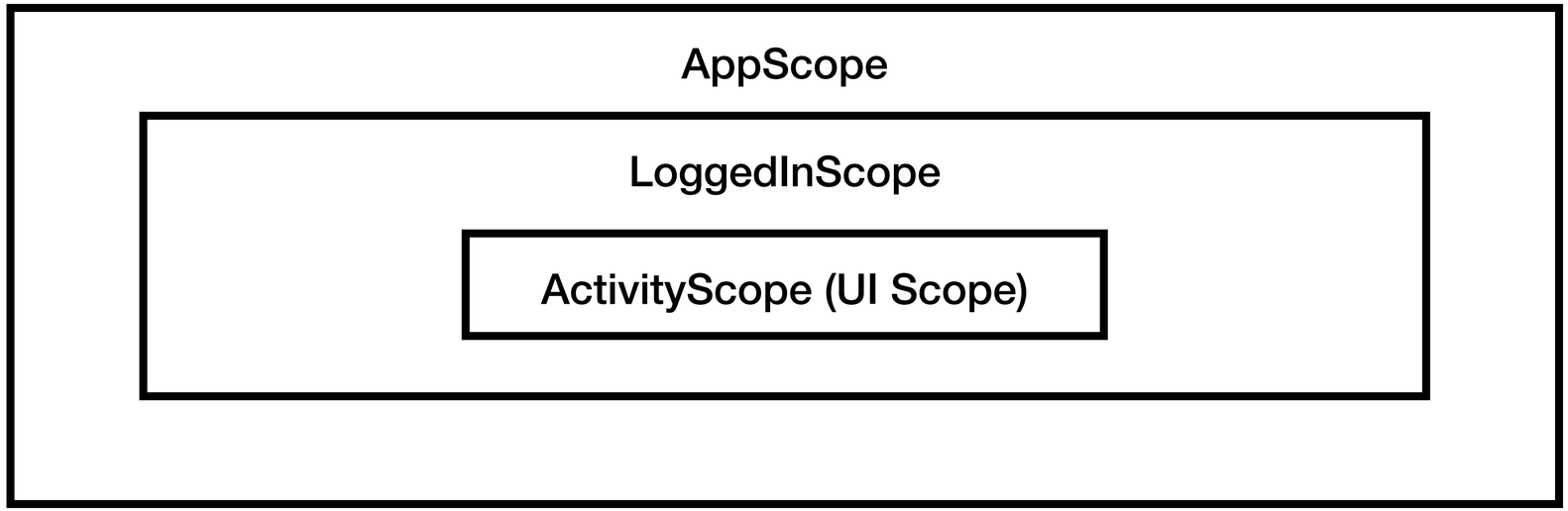
LoggedInScope

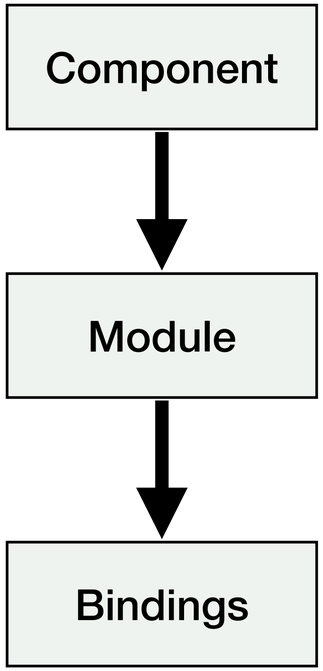
```
class RealCheckoutFlow
@Inject constructor(
  private val paymentEngine: PaymentEngine,
  private val featureFlagsClient: FeatureFlagsClient
  @AuthenticatedConnectClient client: OkHttpClient,
  @Main mainContext: CoroutineContext,
  @Computation computationContext: CoroutineContext,
): CheckoutFlow { ... }
```



Architecture

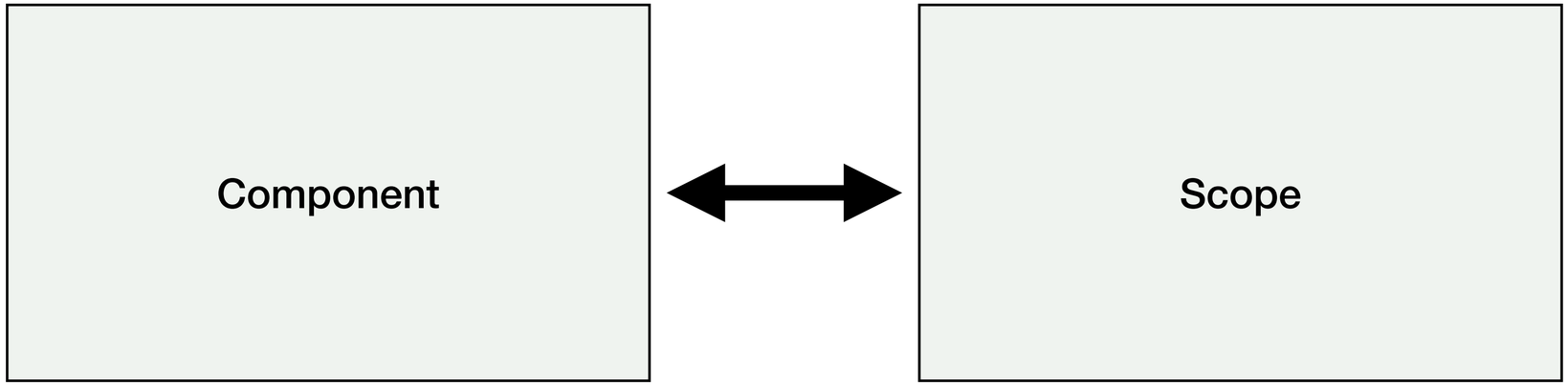
Dagger





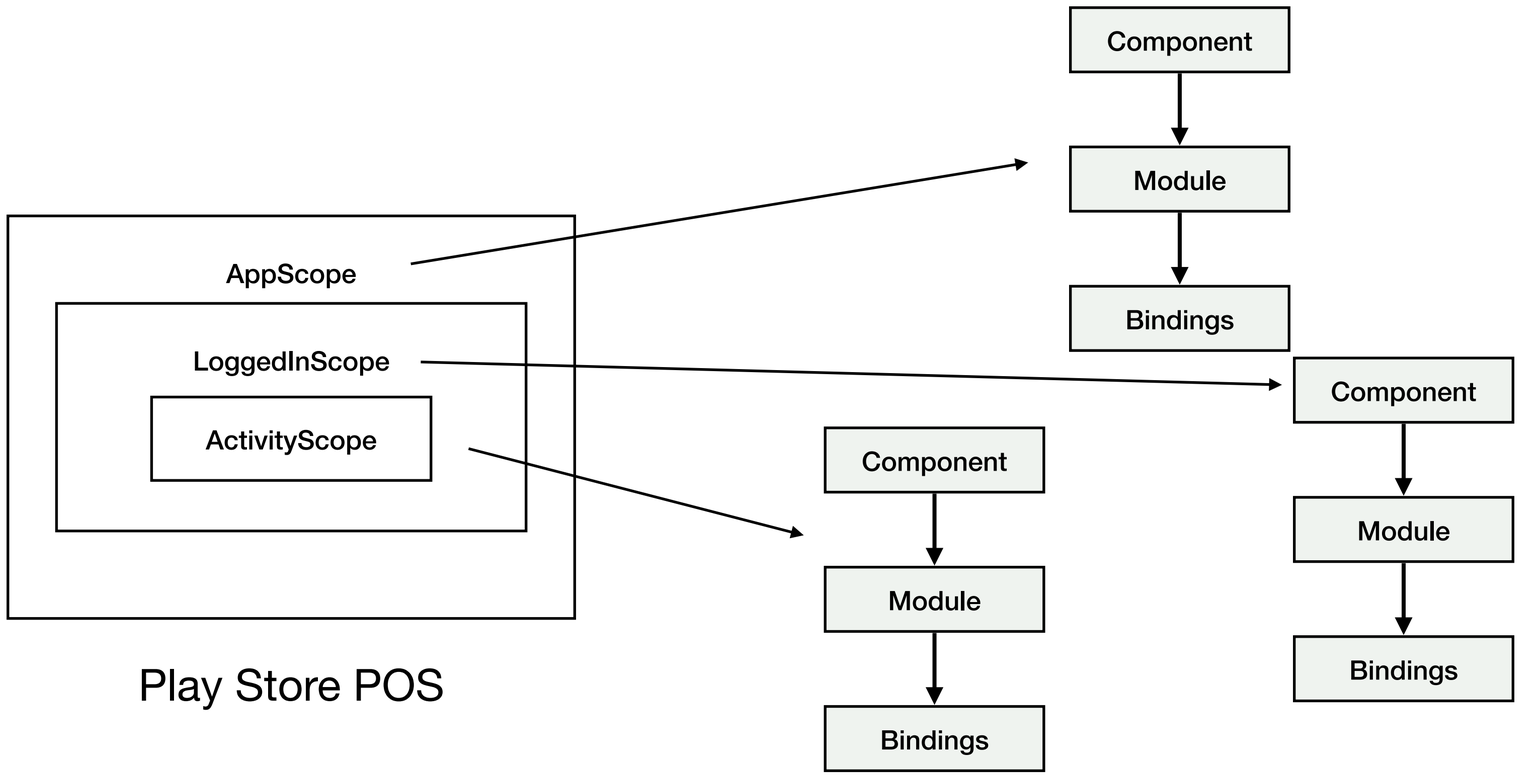
Architecture

Dagger



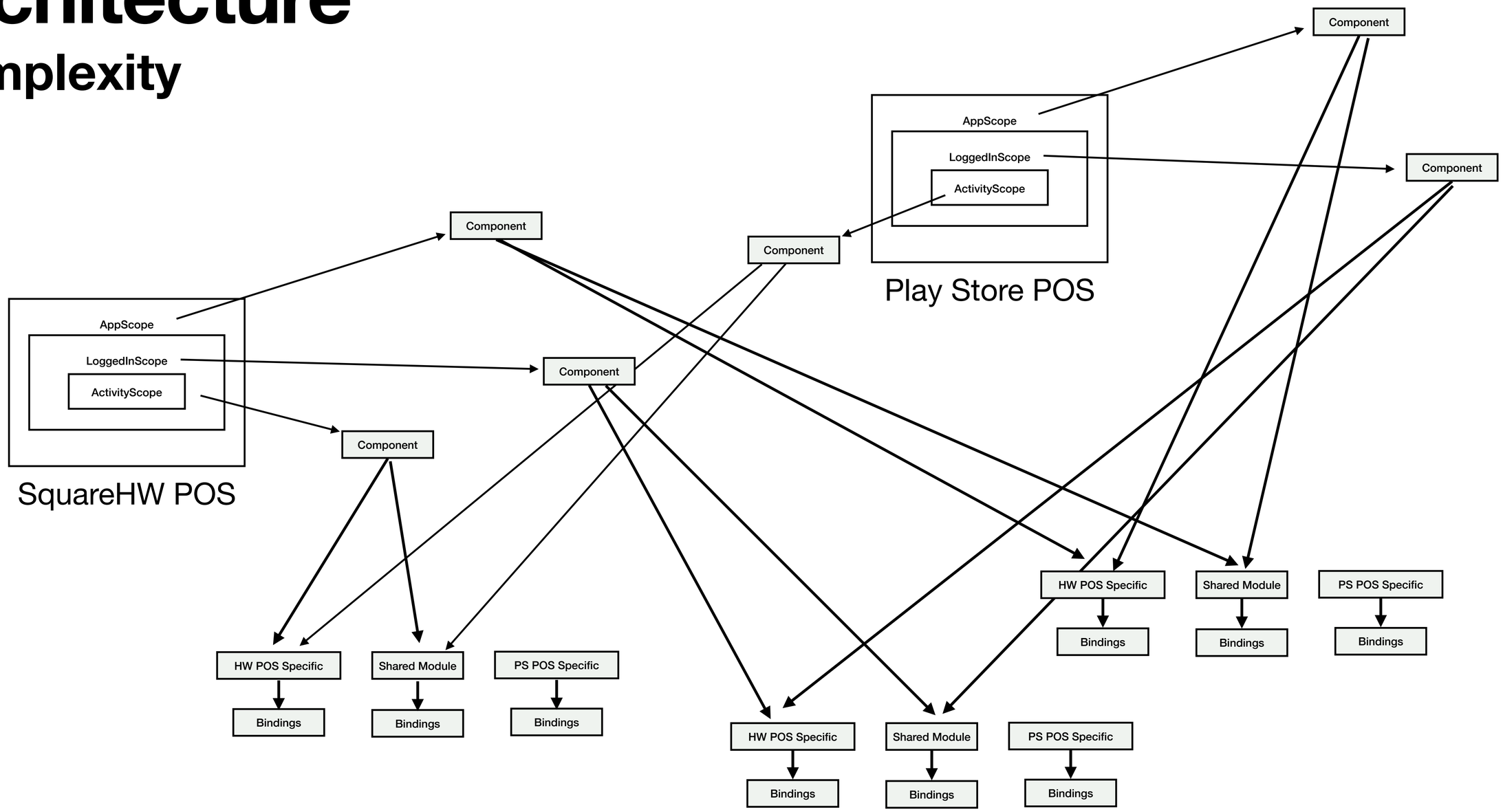
Architecture

Complexity



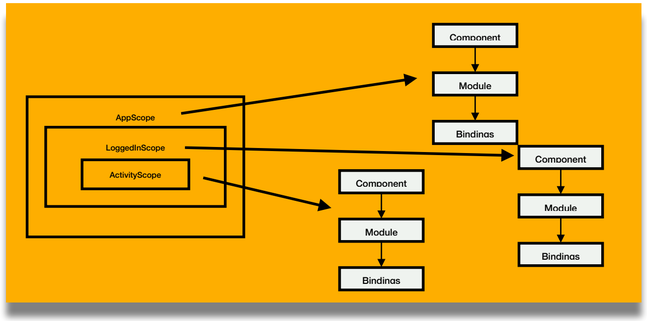
Architecture

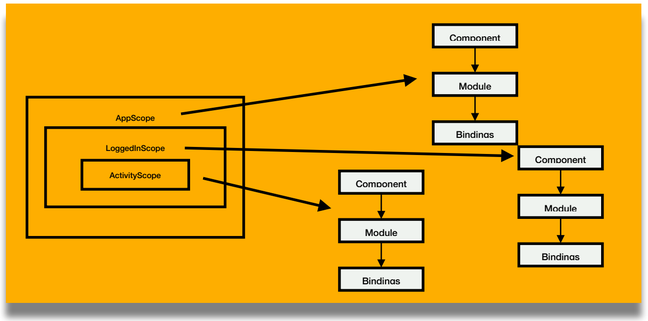
Complexity

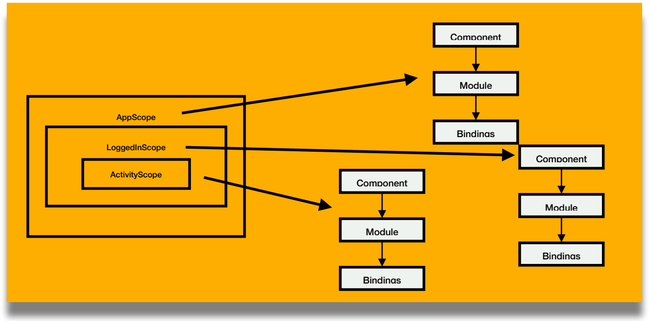


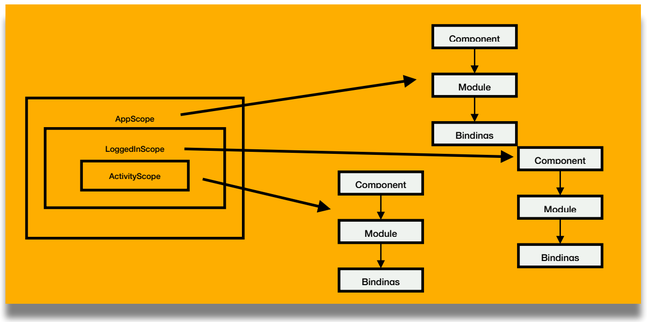
Architecture

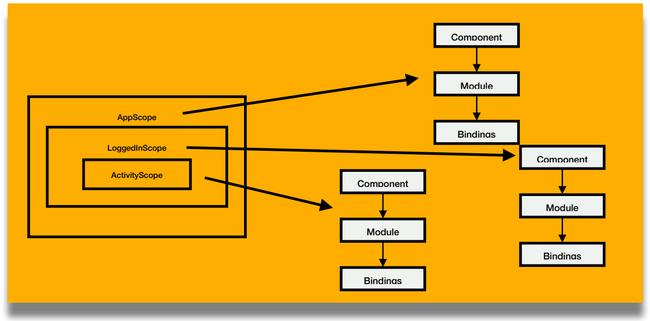
Complexity x 10

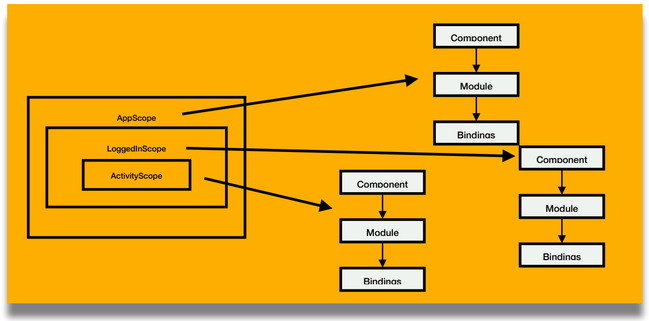








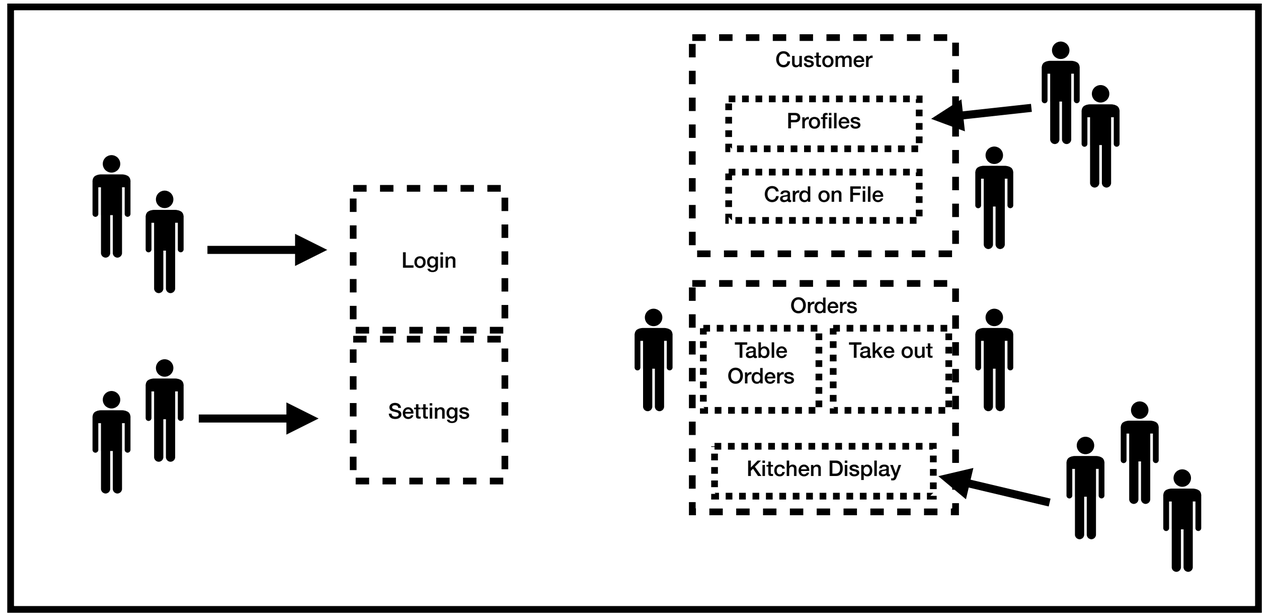




. . .

Dependency Injection + Code Generation

- **Why**
  - Reduce boiler plate
  - Keeps things sane
- **Scale**
  - Small-Medium

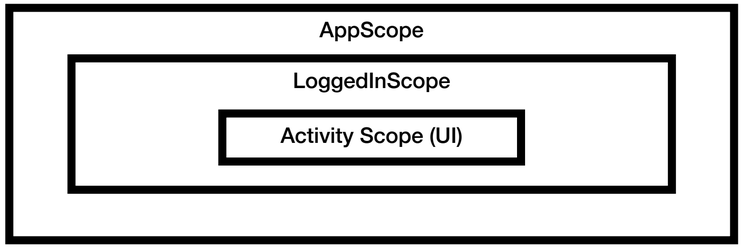


App

Code Generation

Anvil

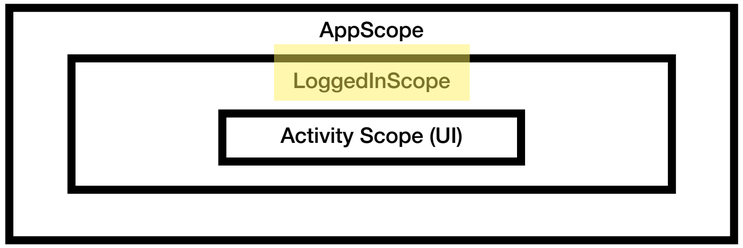
```
class RealCheckoutFlow
@Inject constructor(
  …
): CheckoutFlow { ... }
```



Architecture

Anvil

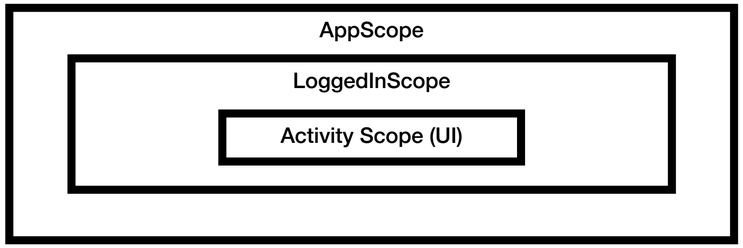
```
@SingleIn(LoggedInScope::class)
@ContributesBinding(scope = LoggedInScope::class, boundType = CheckoutFlow::class)
class RealCheckoutFlow
@Inject constructor(
  ...
): CheckoutFlow { ... }
```



Architecture

Avoid Manual Maintenance

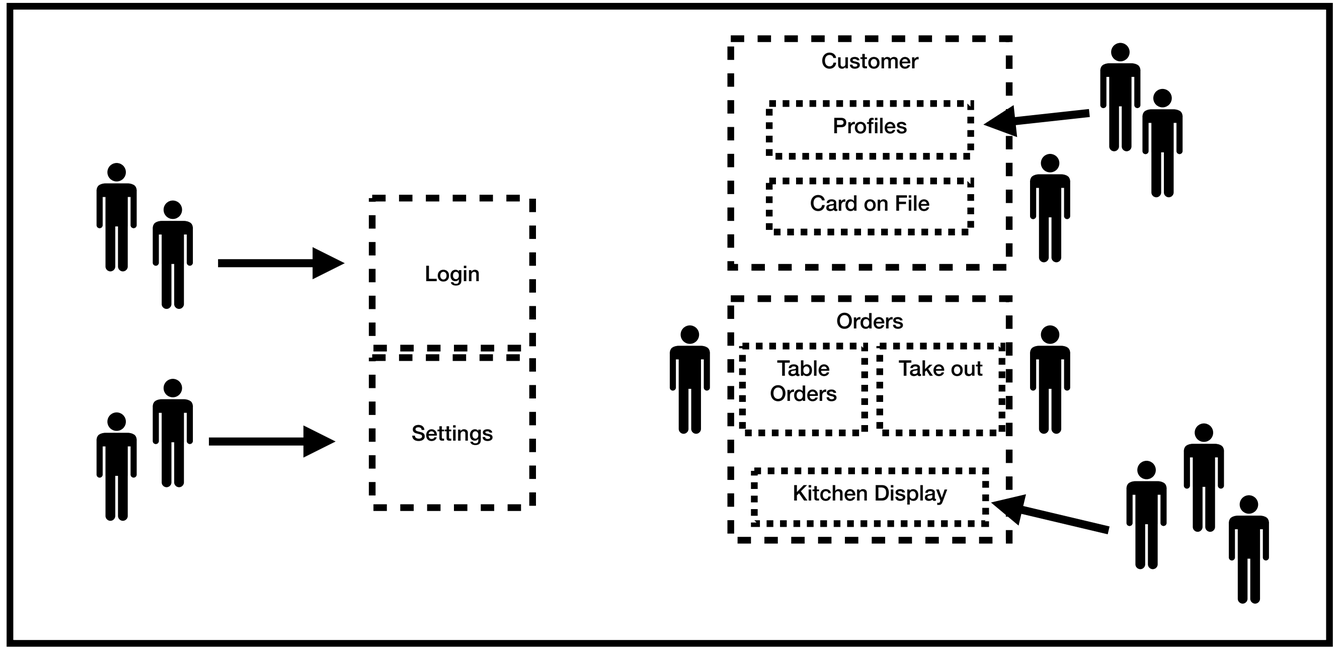
```
@LoggedInScope
class RealCheckoutFlow
@Inject constructor(
  ...
): CheckoutFlow { ... }


@Module
@InstallIn(LoggedInComponent::class)
abstract class CheckoutFlowModule {

  @Binds
  abstract fun bindCheckoutFlow(
    impl: RealCheckoutFlow
  ): CheckoutFlow
}
```



Modularization

Modularization

- **Why**
  - Clean separation of concerns
  - Teams working independently
  - Keep build times low
  - Easy compile time implementation swapping
- **Scale**
  - Small-Medium



App

Architecture

Module Structure

public

```
interface PaymentEngine { }
```

impl

```
@ContributesBinding(boundType = PaymentEngine::class)
class RealPaymentEngine @Inject constructor(…) { }
```

Architecture: Module Structure

Feature Module Structure

Example: common/my-feature/…

- public
- impl
- impl-robots
- demo

Architecture: Module Structure

Feature Module Structure

Example: common/my-feature/…

public	Contracts - this is what your feature provides and expects
impl	Implementation
impl-robots	Driver for your Implementation
demo	Android App which can run just impl in isolation

Architecture: Module Structure

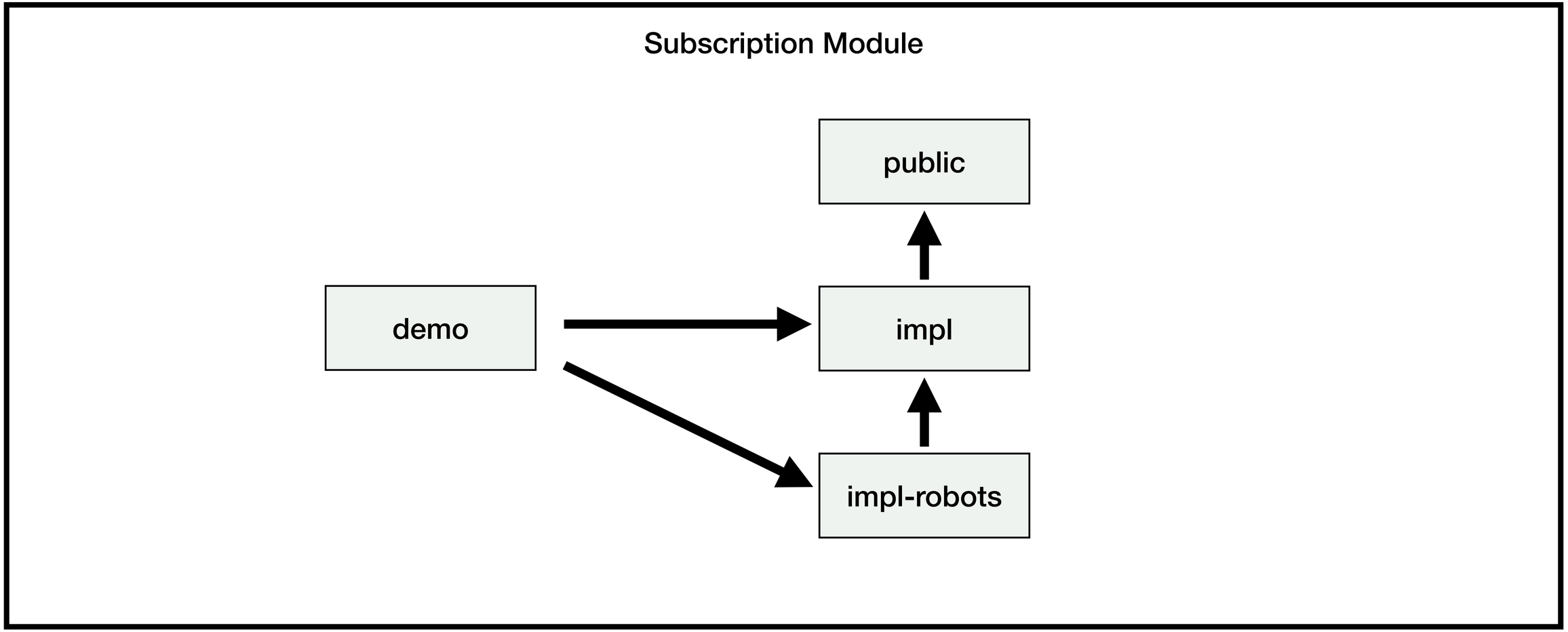
Feature Module Structure

Example: common/my-feature/…

- public
- impl
  - impl-retail
- impl-robots
- demo

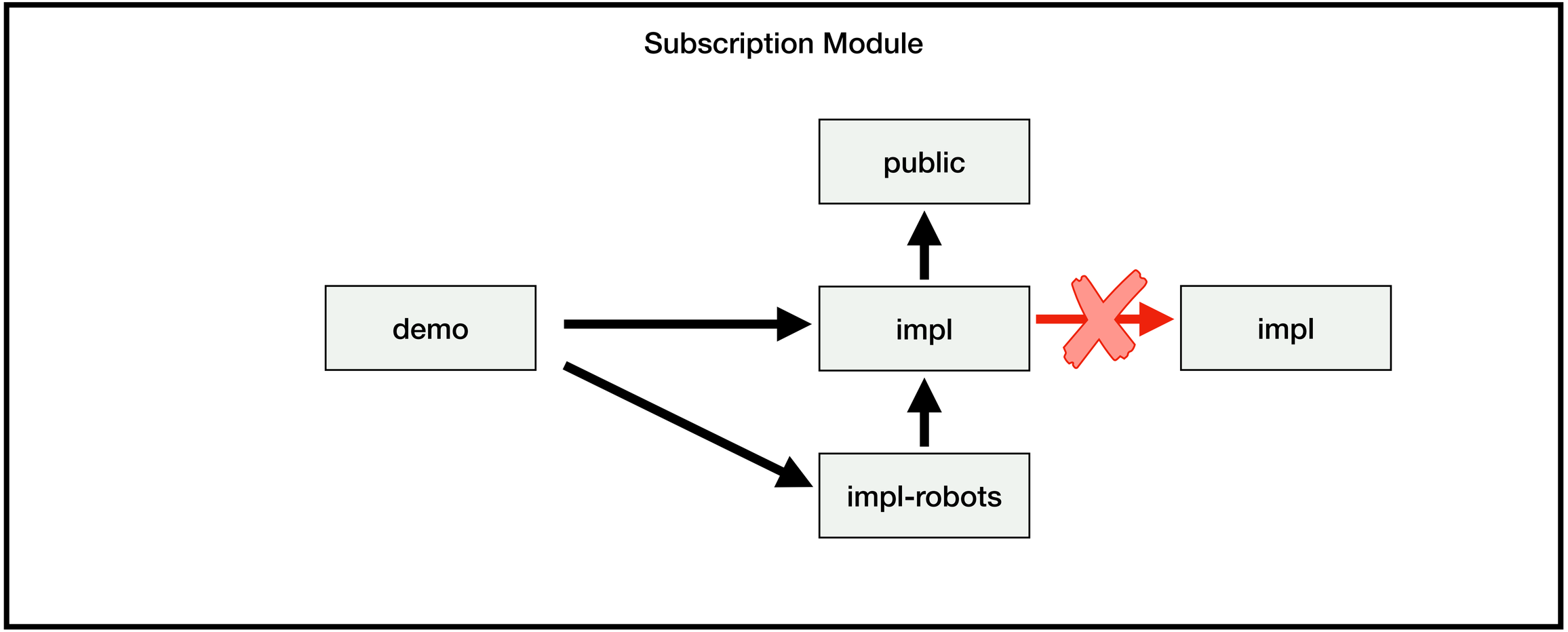
Architecture: Module Structure

Feature Module Structure



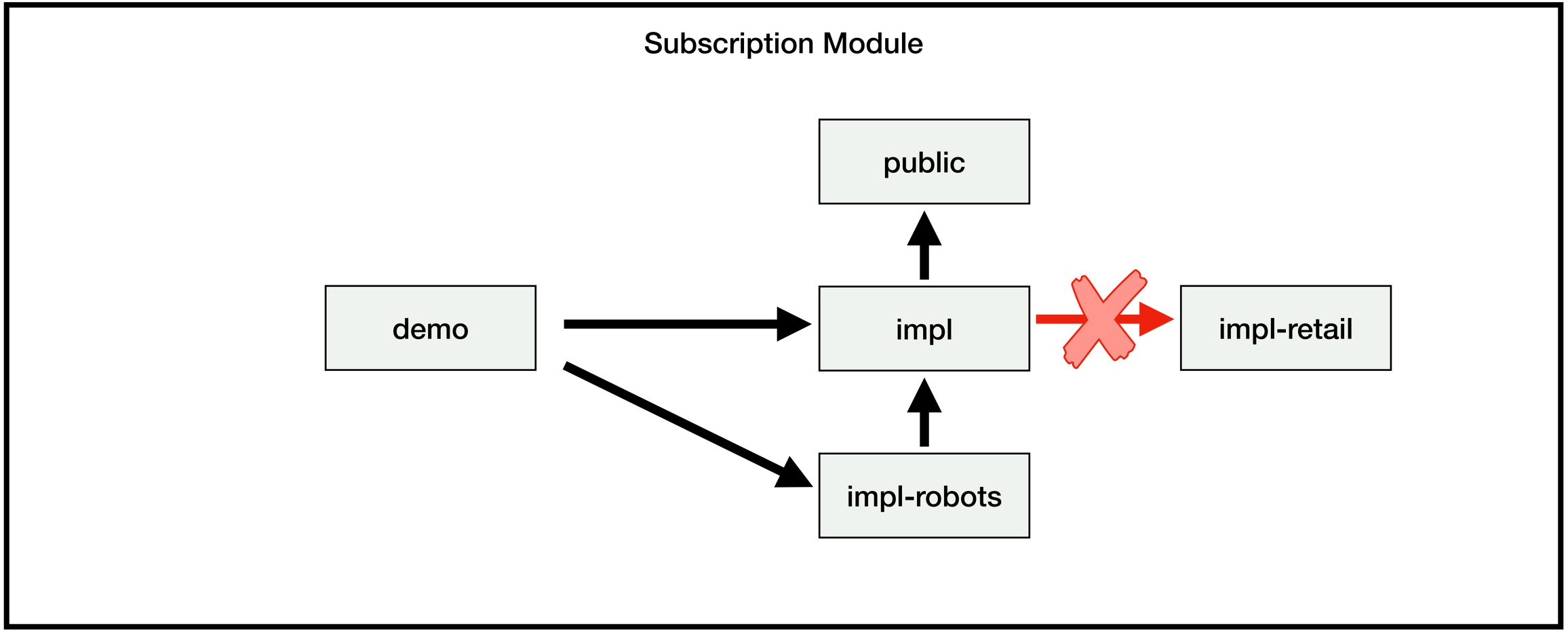
Architecture: Module Structure

Feature Module Structure



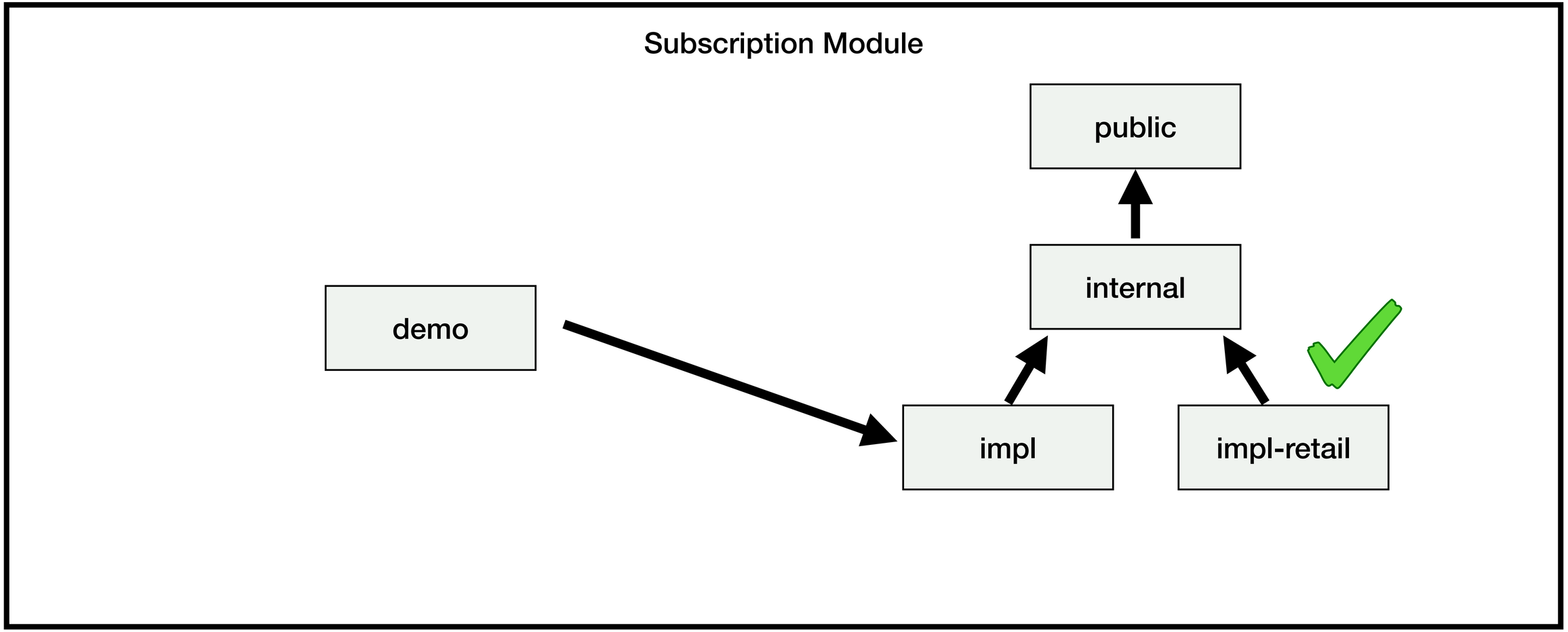
Architecture: Module Structure

Feature Module Structure



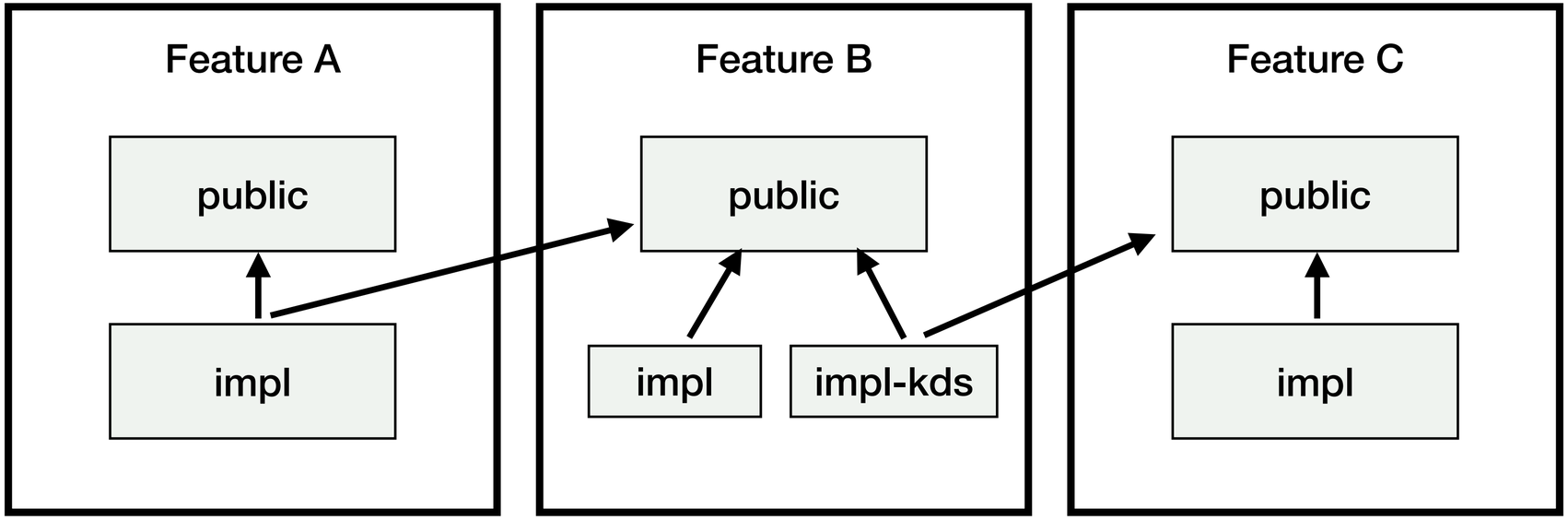
Architecture: Module Structure

Feature Module Structure



Architecture: Dagger + Anvil

Wiring Together

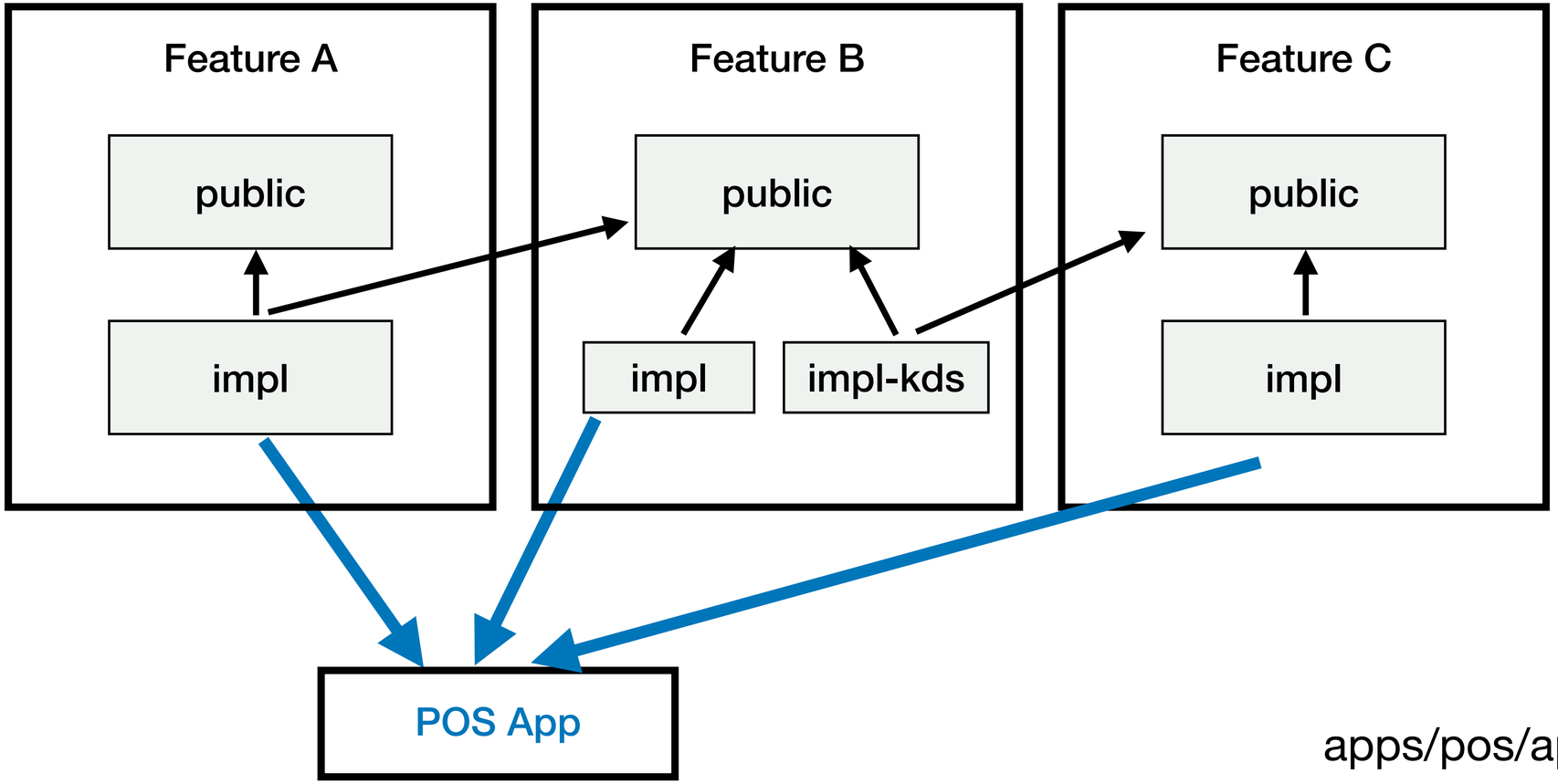


common/featureA/impl

```
api project(':common:featureA:public')
api project(':common:featureB:public')
```

Architecture: Dagger + Anvil

Wiring Together

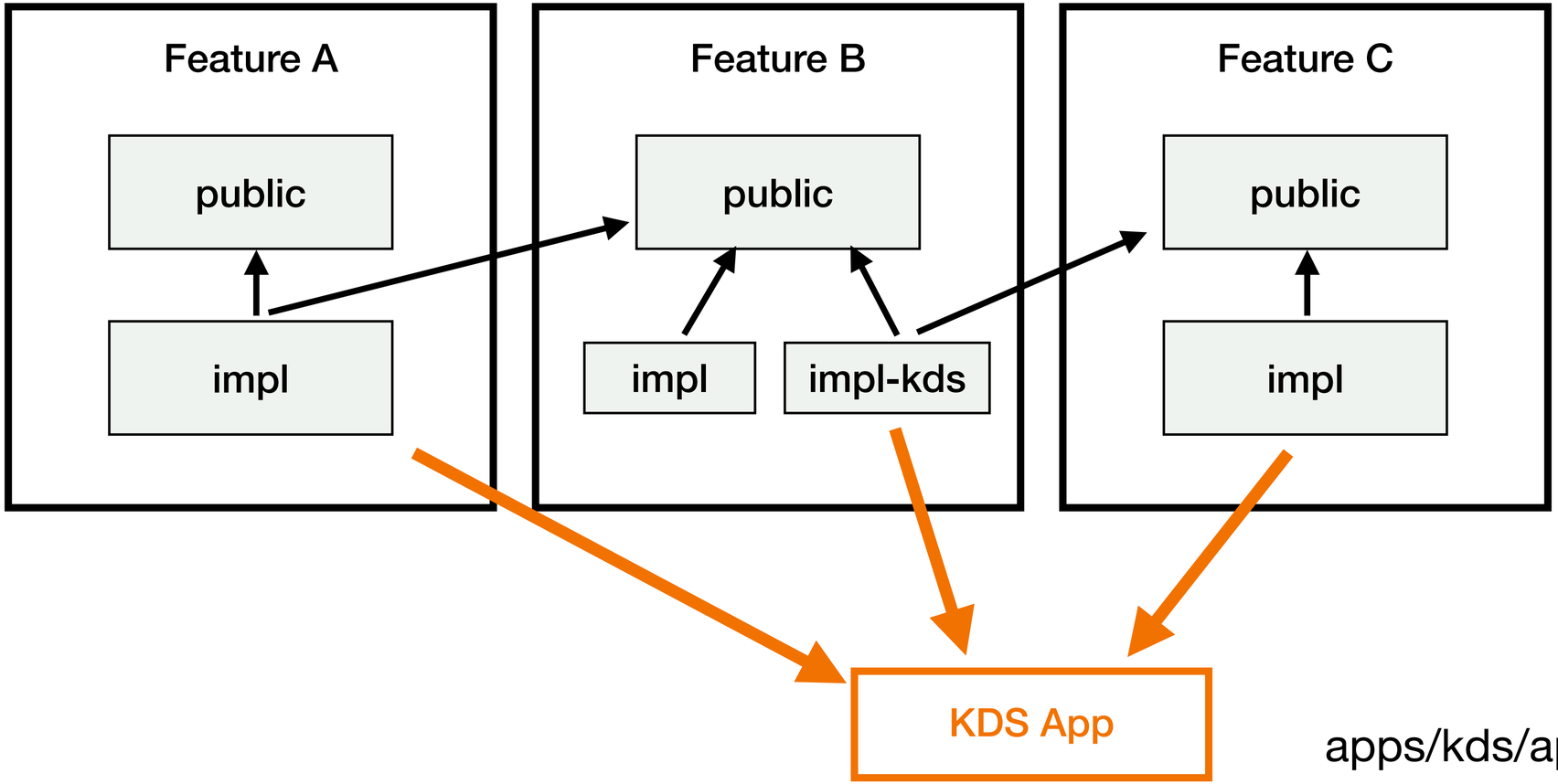


apps/pos/app/build.gradle

```
api project(':common:featureA:impl')
api project(':common:featureB:impl')
api project(':common:featureC:impl')
```

Architecture: Dagger + Anvil

Wiring Together

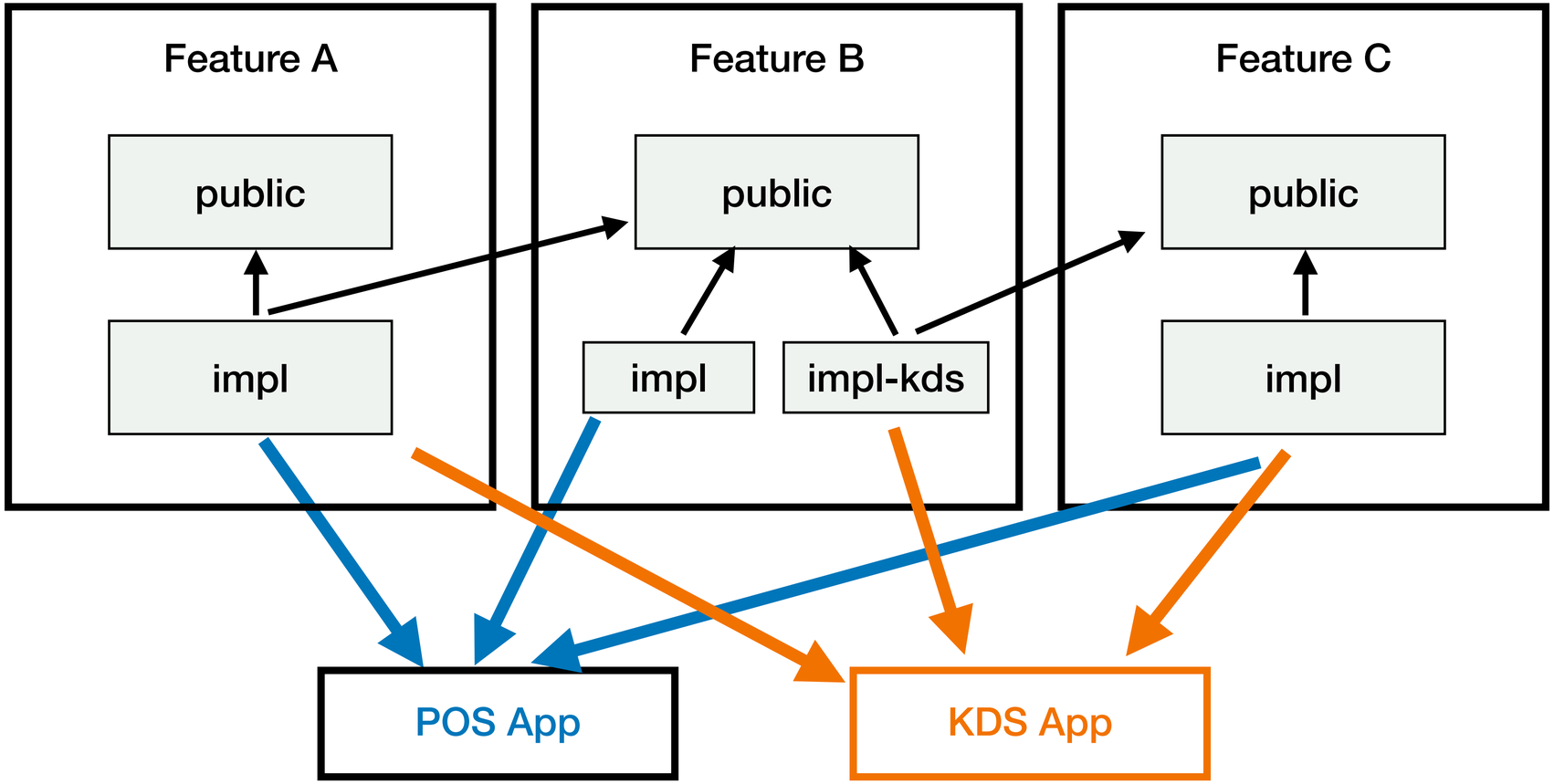


apps/kds/app/build.gradle

```
api project(':common:featureA:impl')
api project(':common:featureB:impl-kds')
api project(':common:featureC:impl')
```

Architecture: Dagger + Anvil

Wiring Together



Modularization

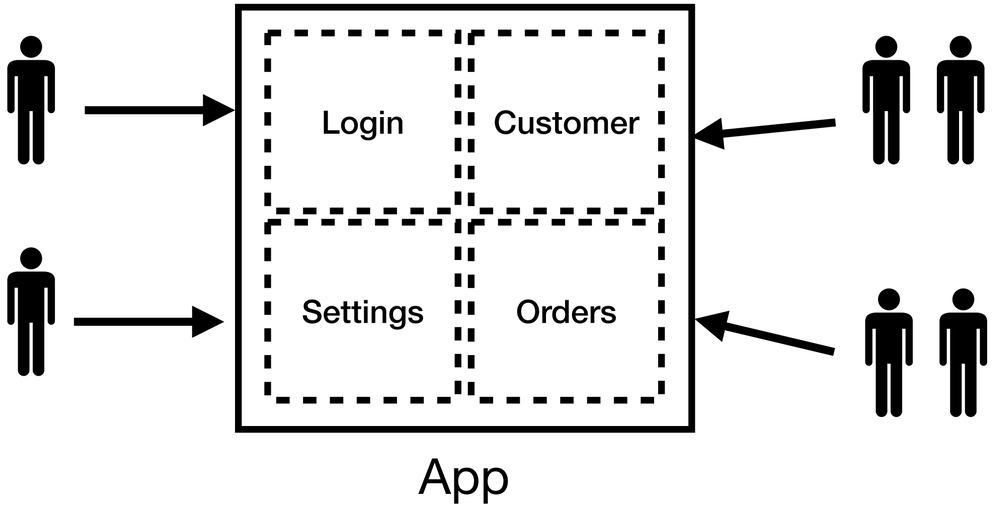
Review

- Easy / Compile time implementation swapping
- Keep teams from stepping on each others toes
- Keep build times low even with scale
- Clean separation of concerns

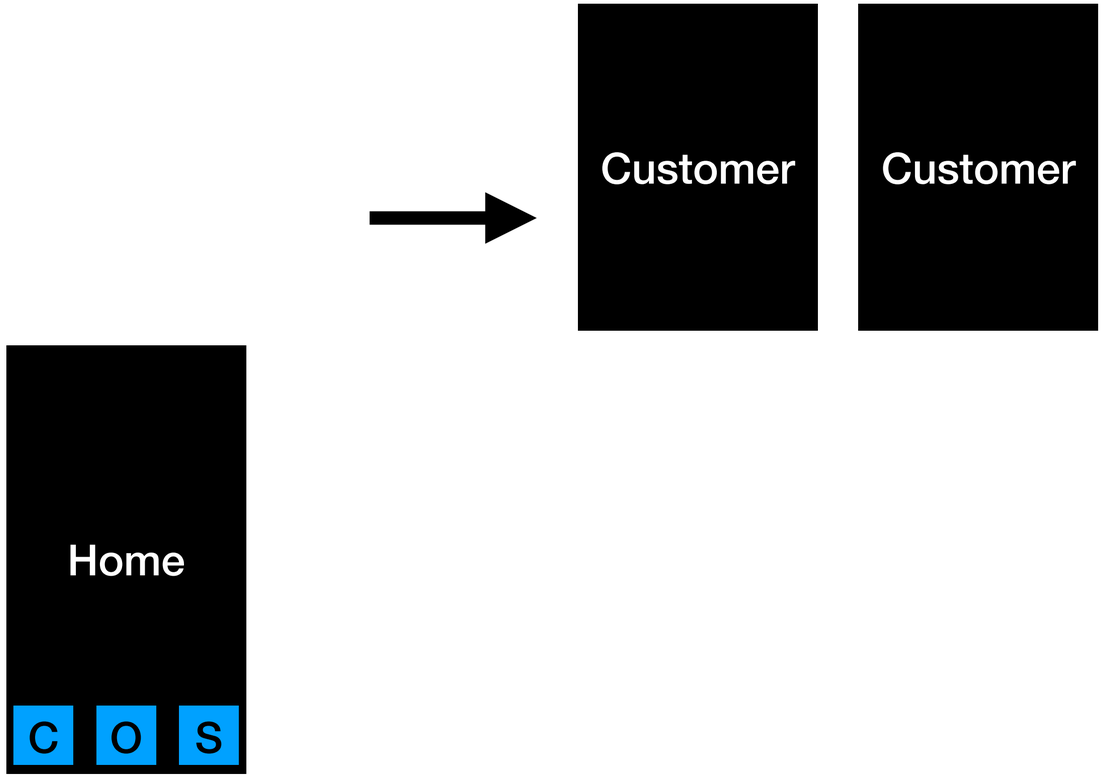
UI Framework

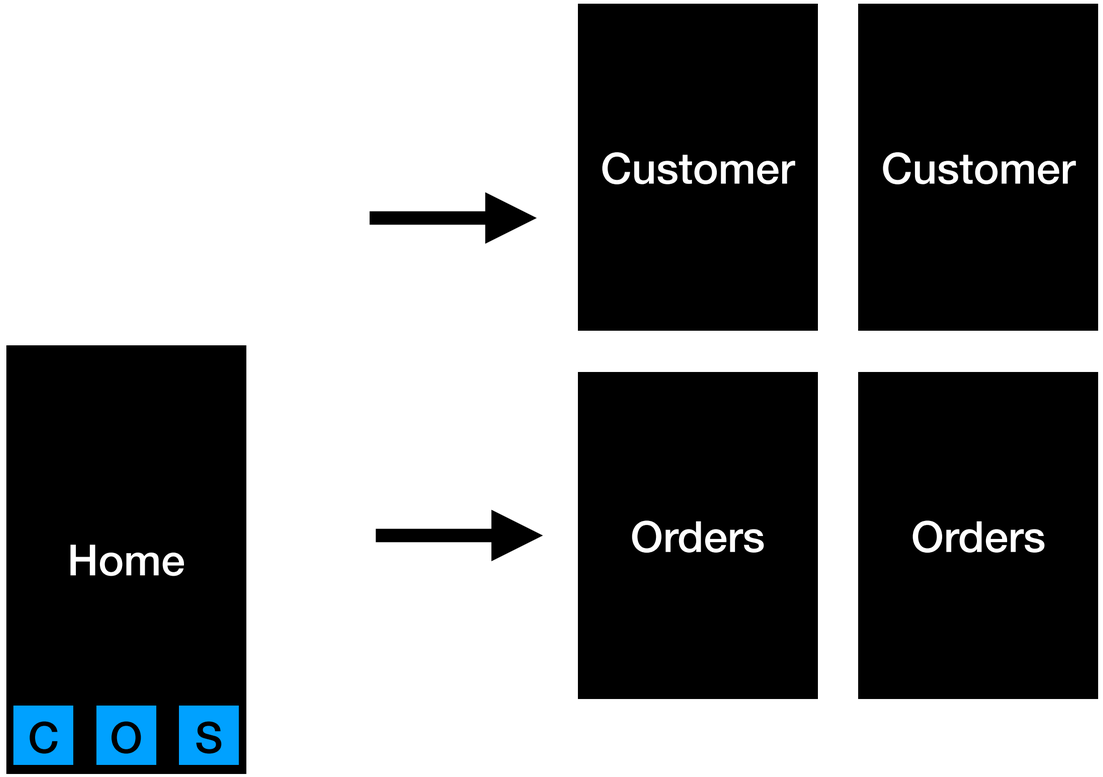
UI Framework

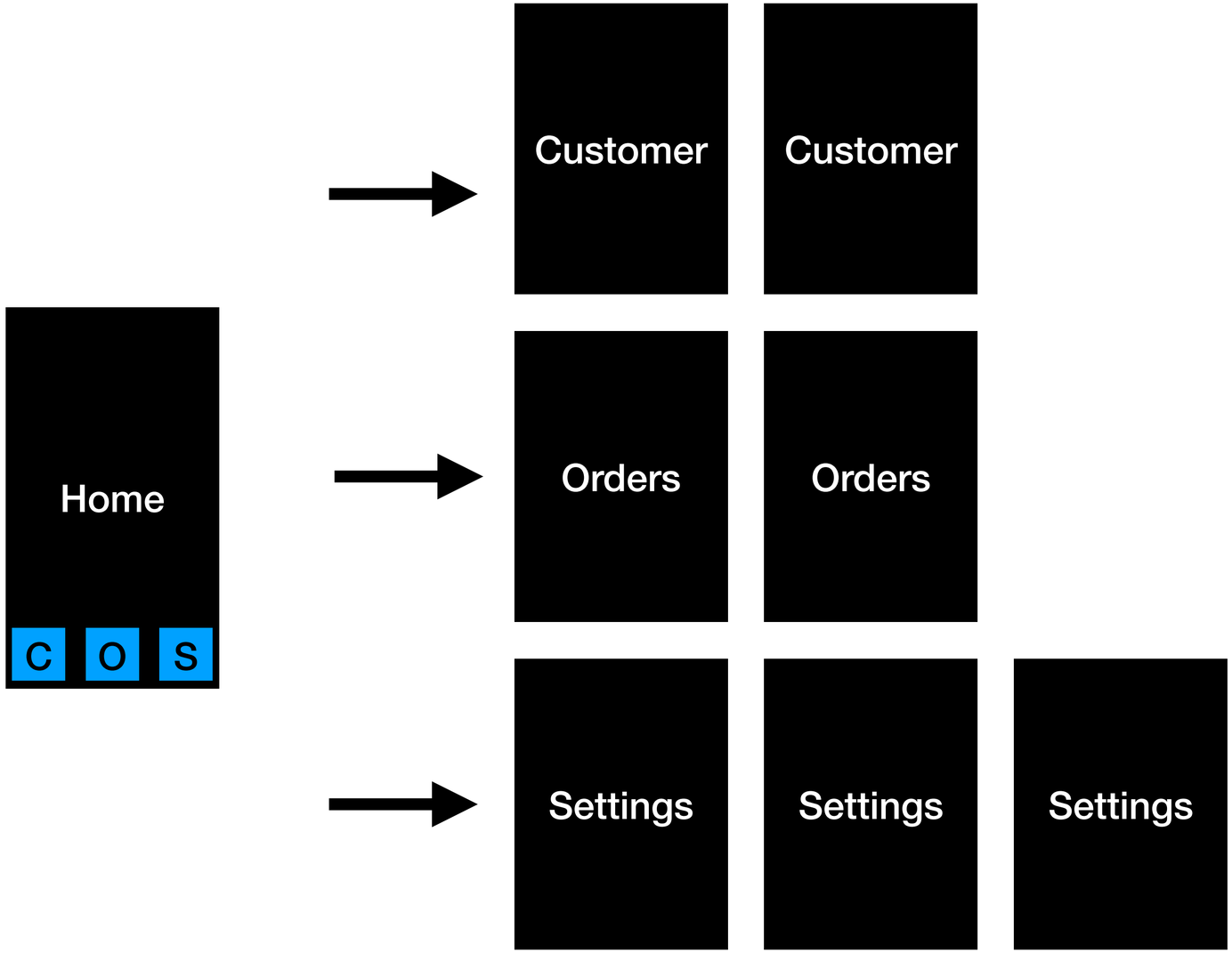
- **What**
- Unidirectional Data Flow (UDF)
- Strong Contracts
- Strong Coupling between
  (Navigation, Model and UI State)
  - State transitions drive navigation
  - Navigation is pure function of state
- **Scale**
  - Small

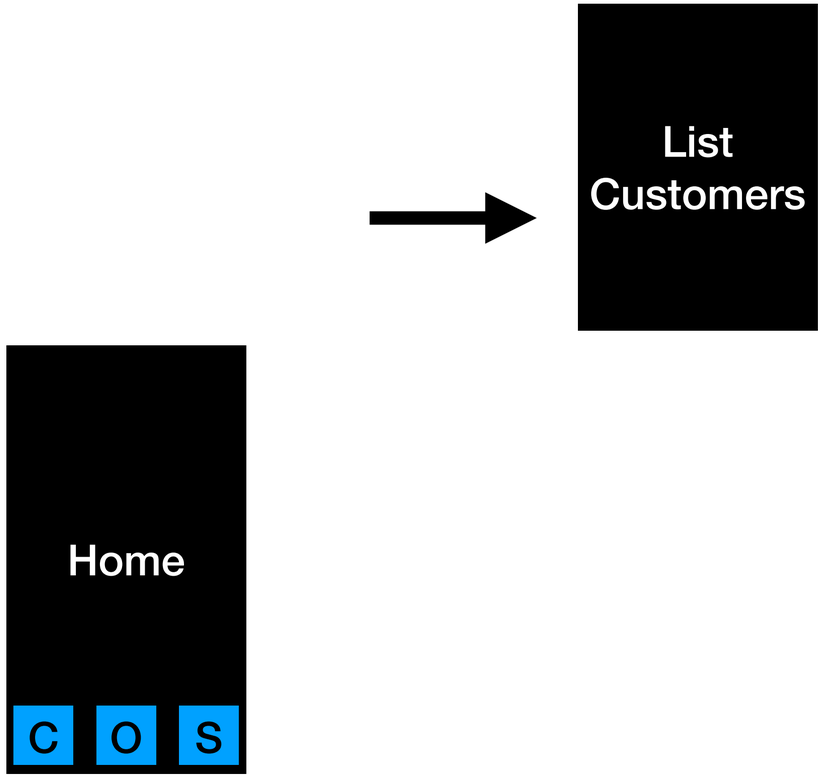


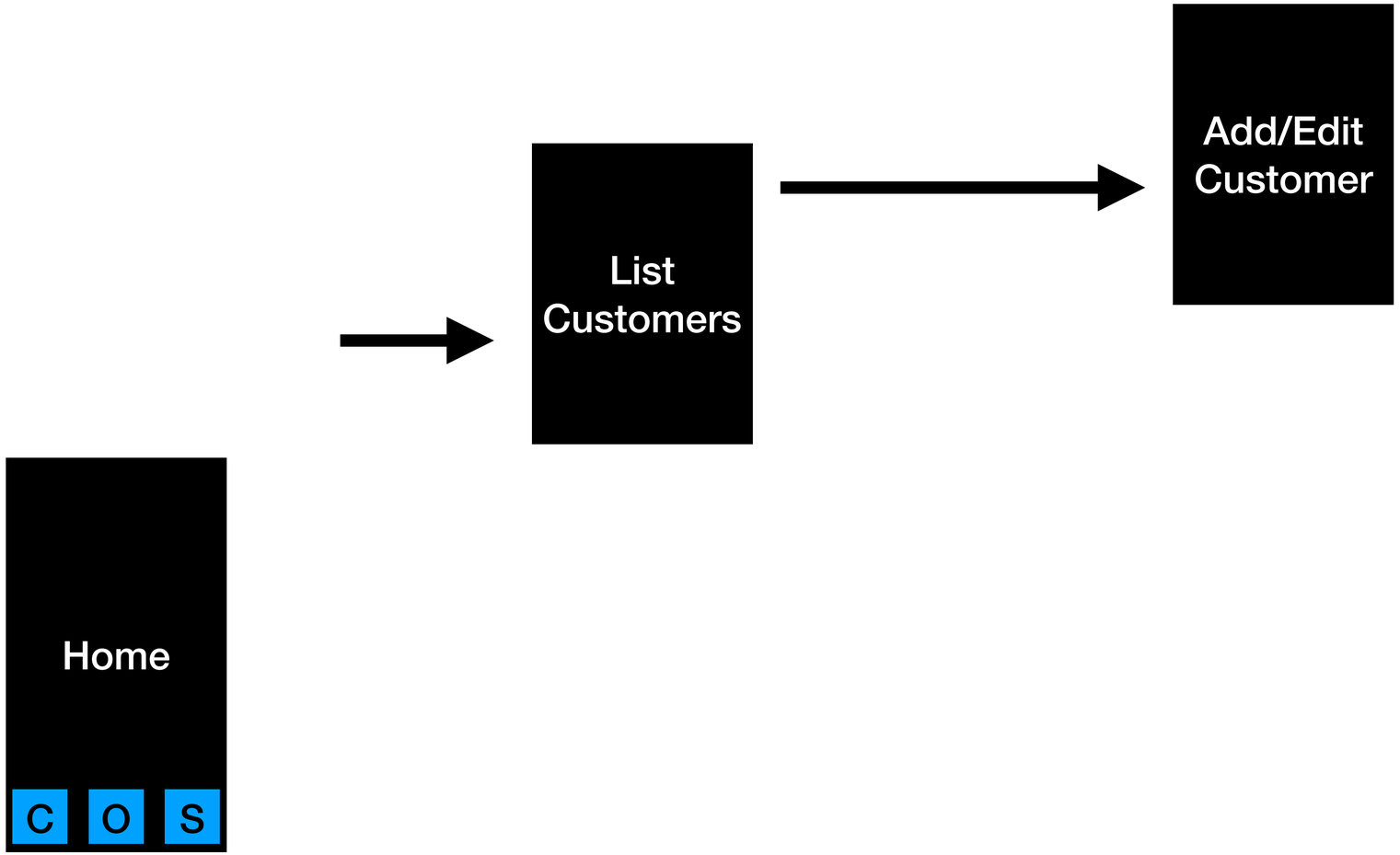
Home
C
O
S

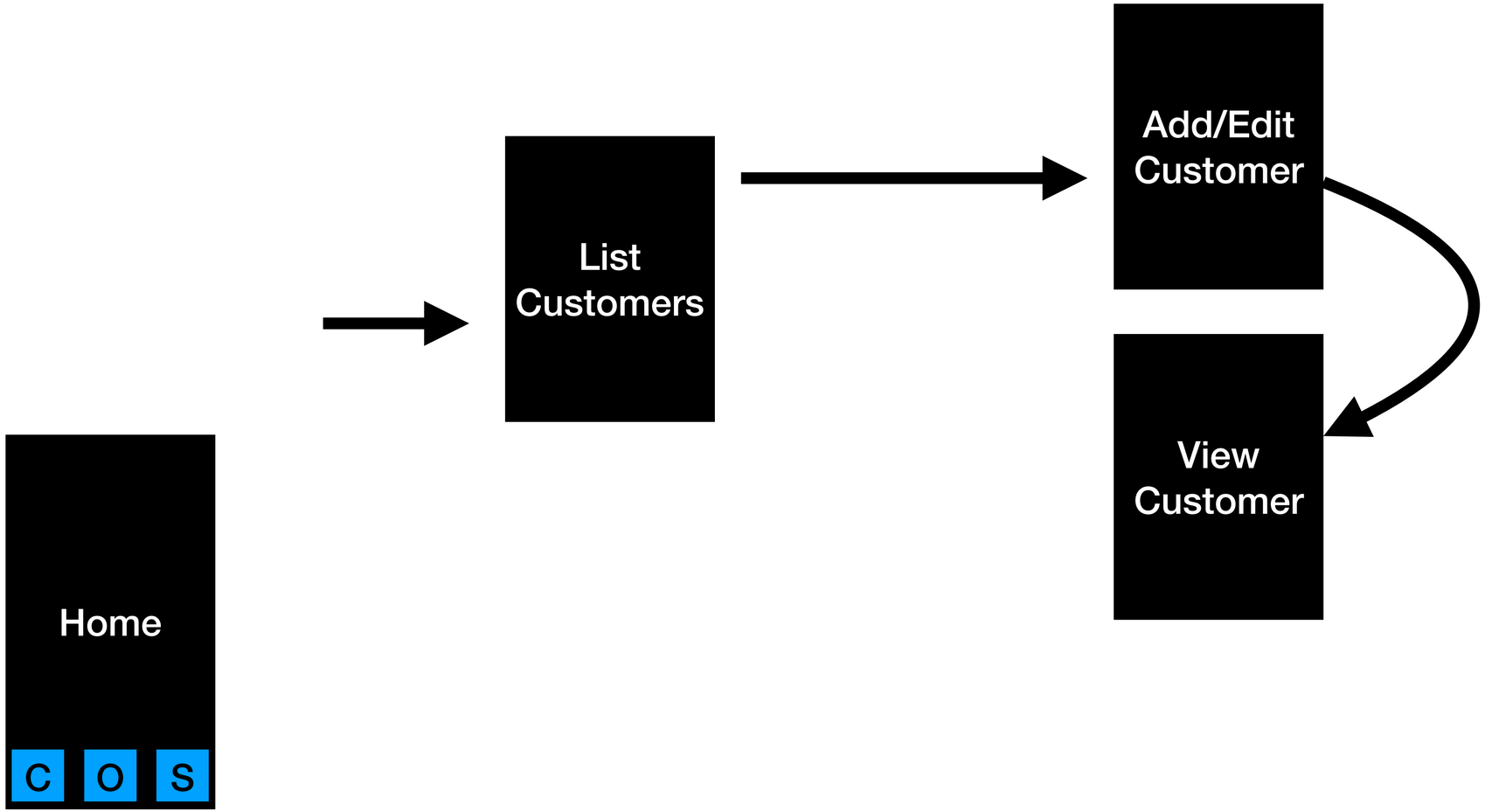

Customer
Customer
Home
C
O
S

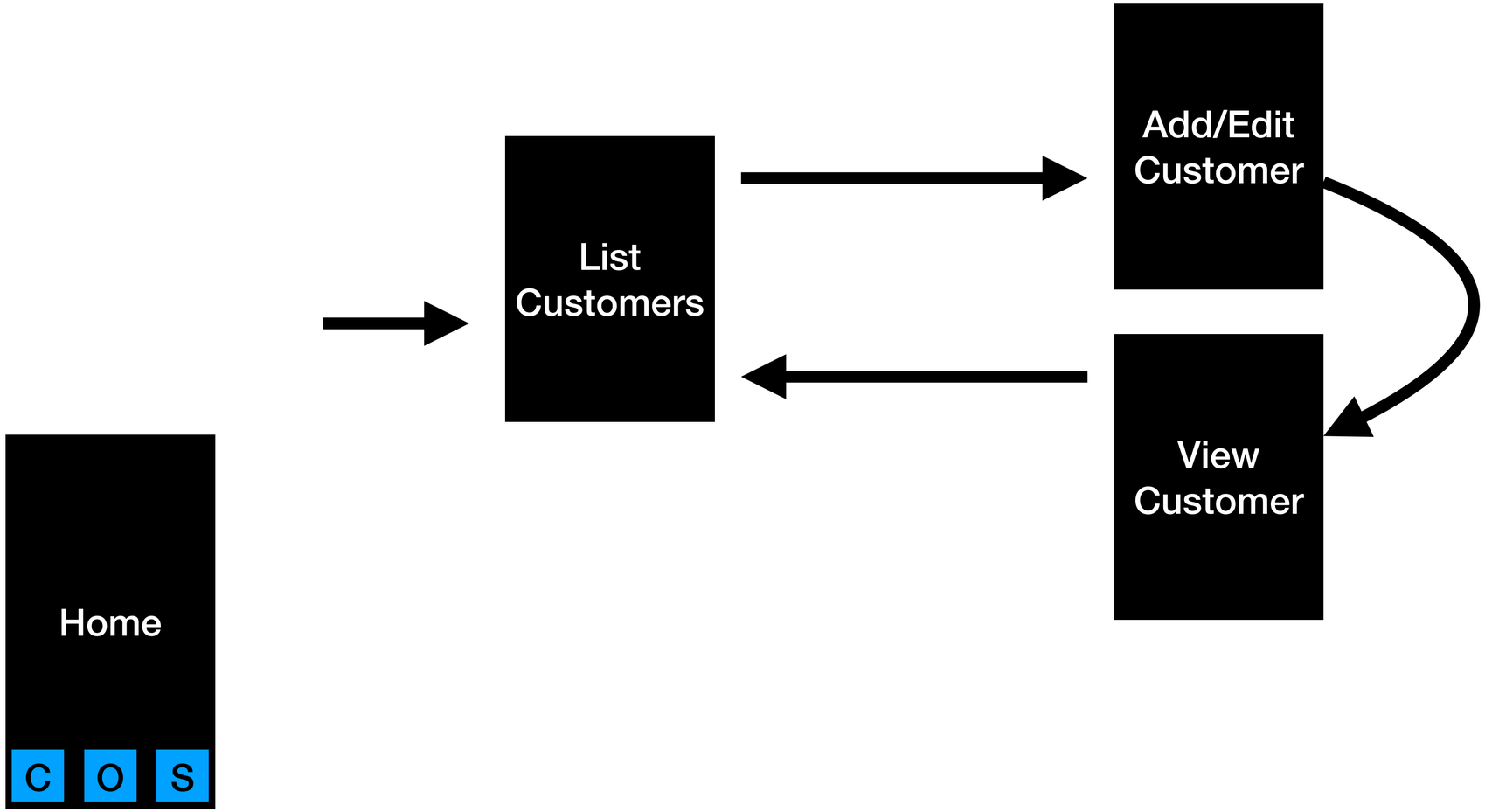

Customer
Customer
Home
C
O
S
Orders
Orders

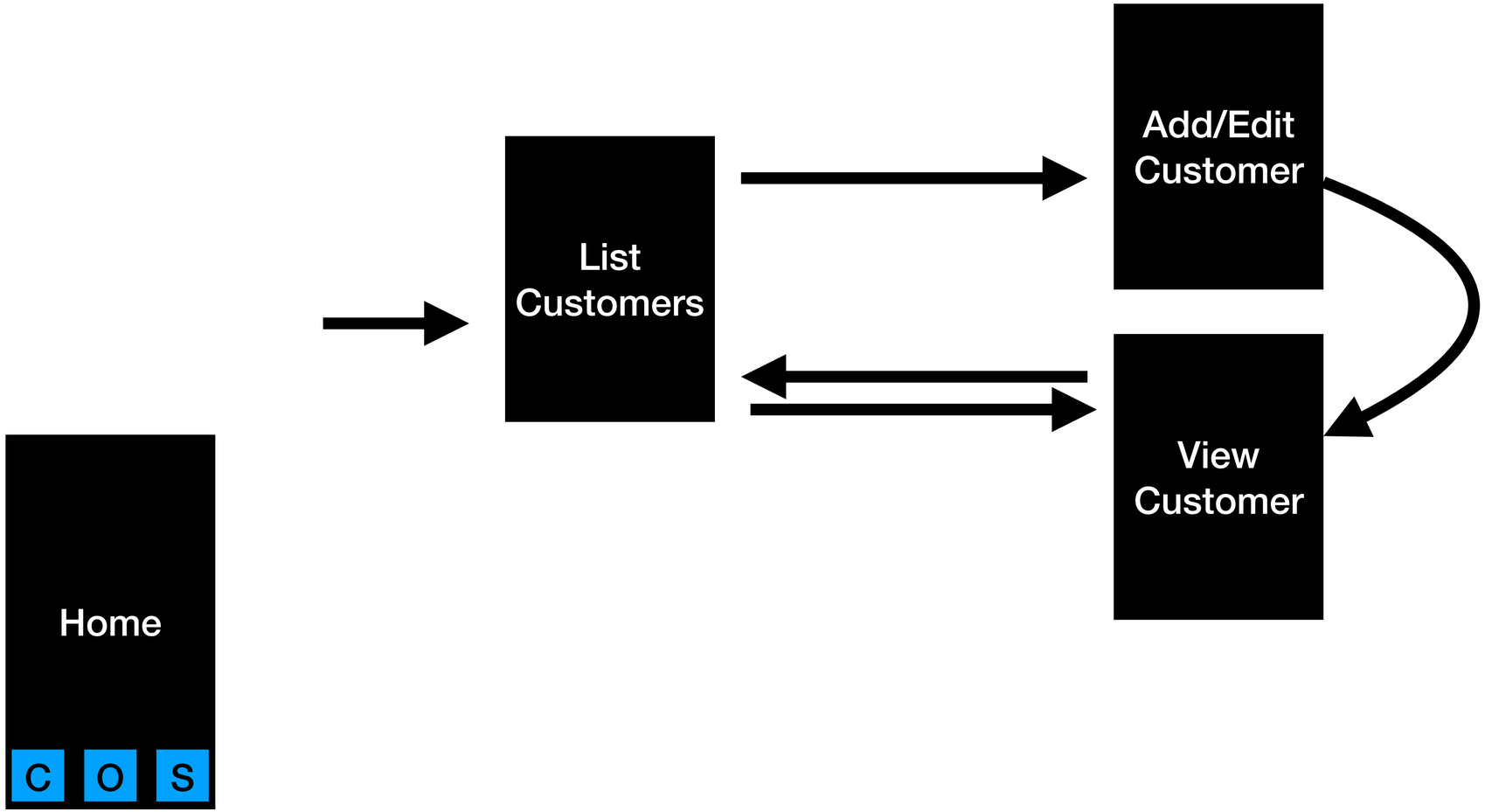

Home
C
O
S
Customer
Customer
Orders
Orders
Settings
Settings
Settings

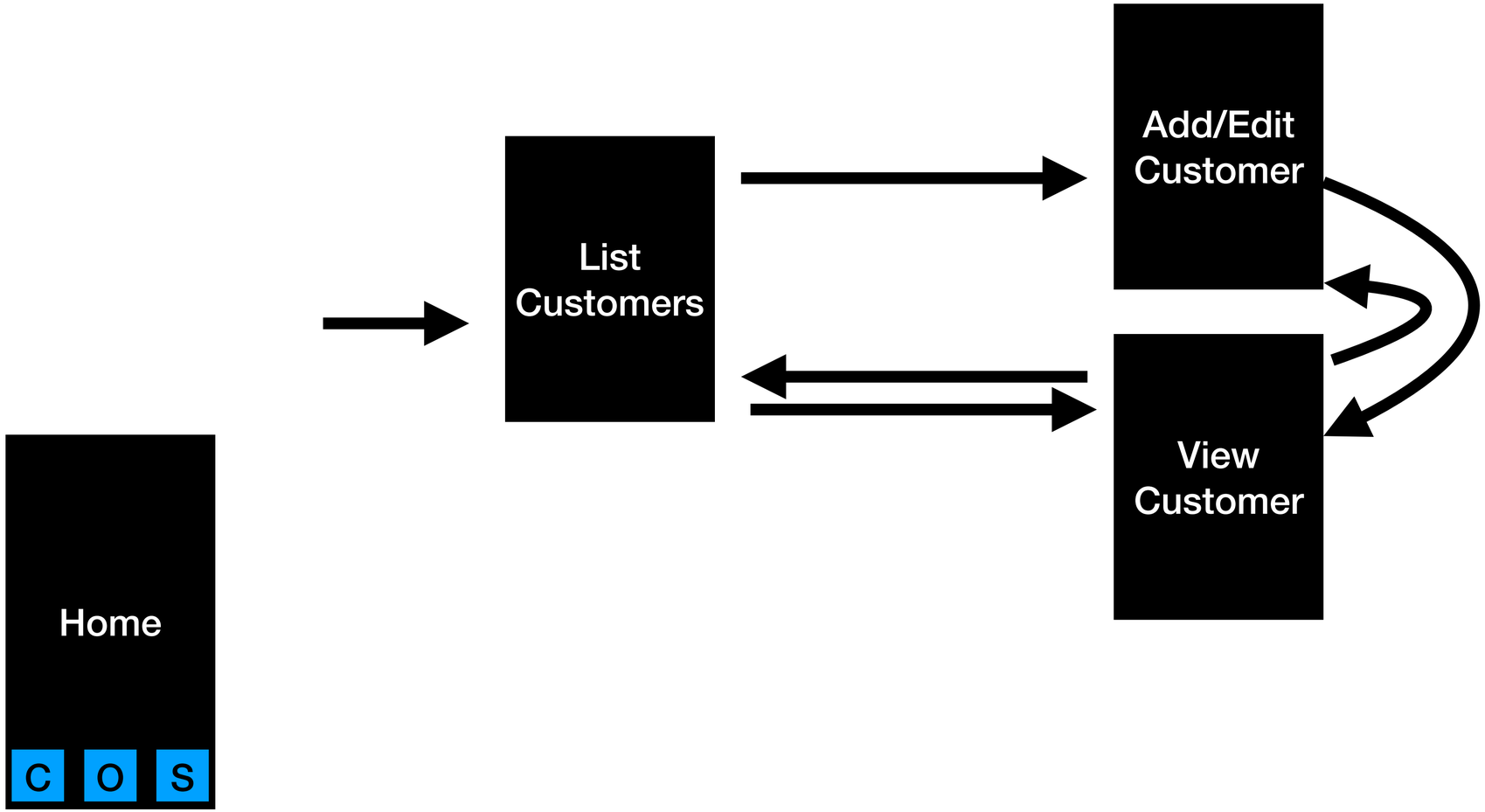

List
Customers
Home
C
O
S

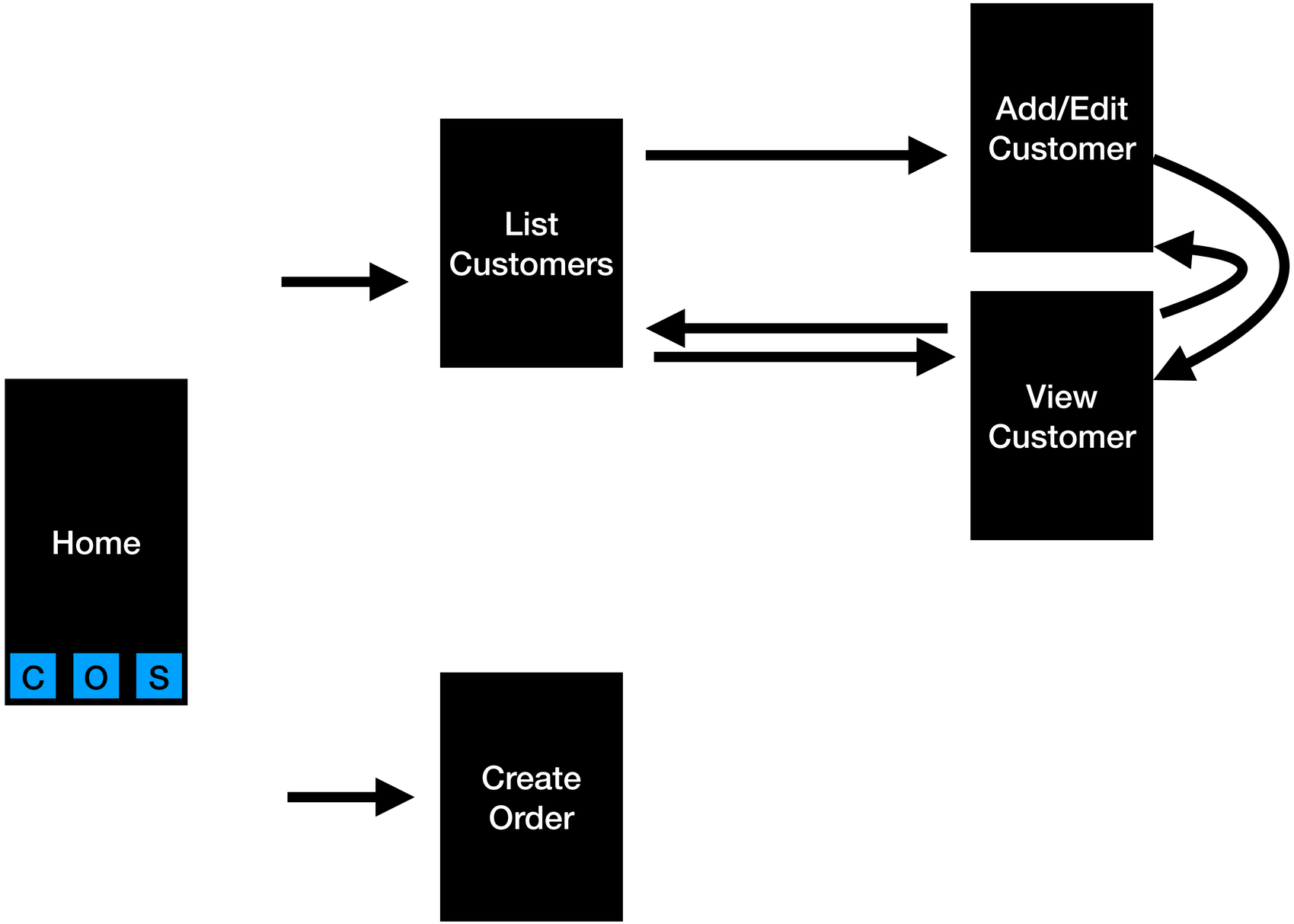

Add/Edit
Customer
List
Customers
Home
C
O
S

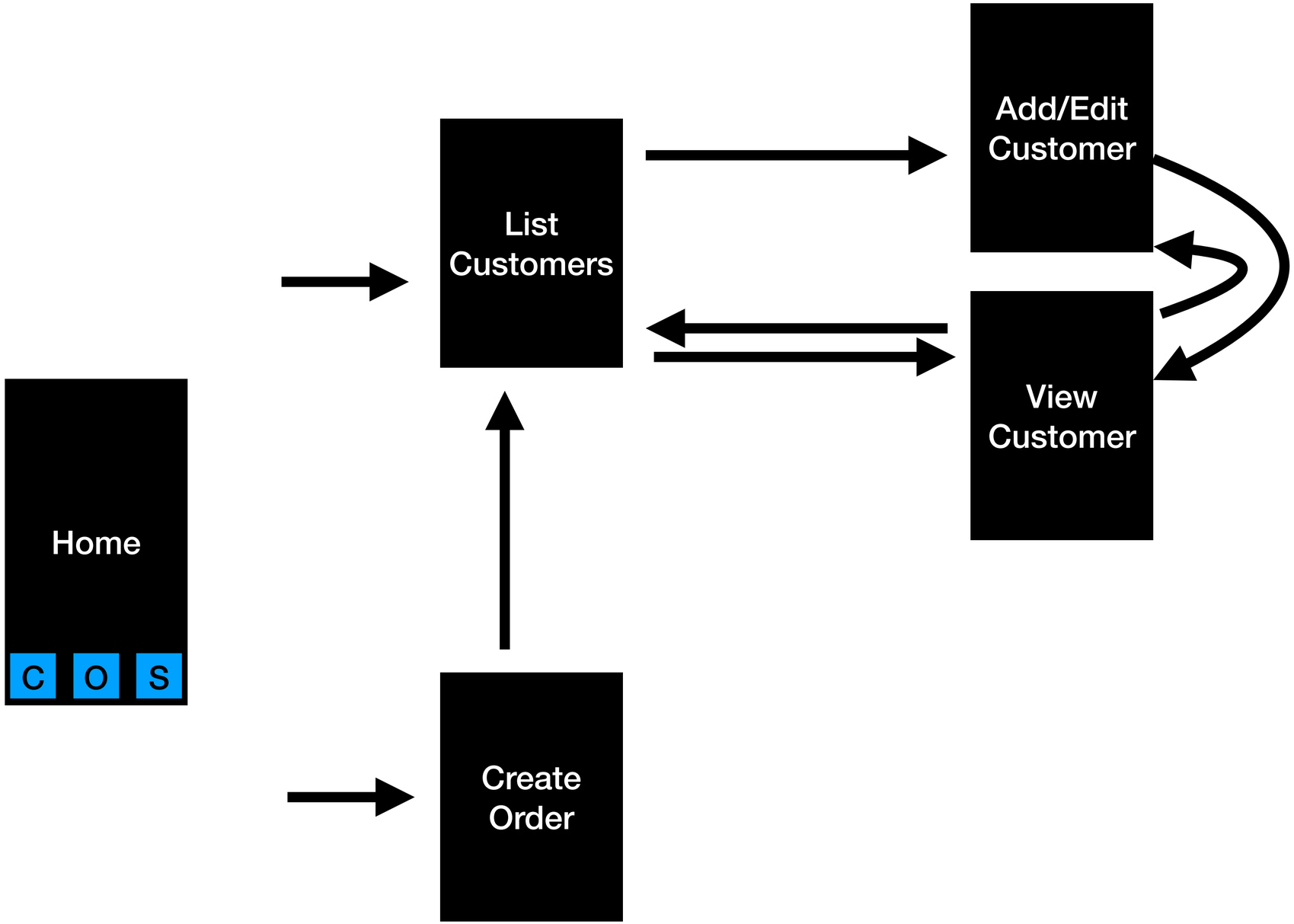

Home
C
O
S
List Customers
Add/Edit Customer
View Customer

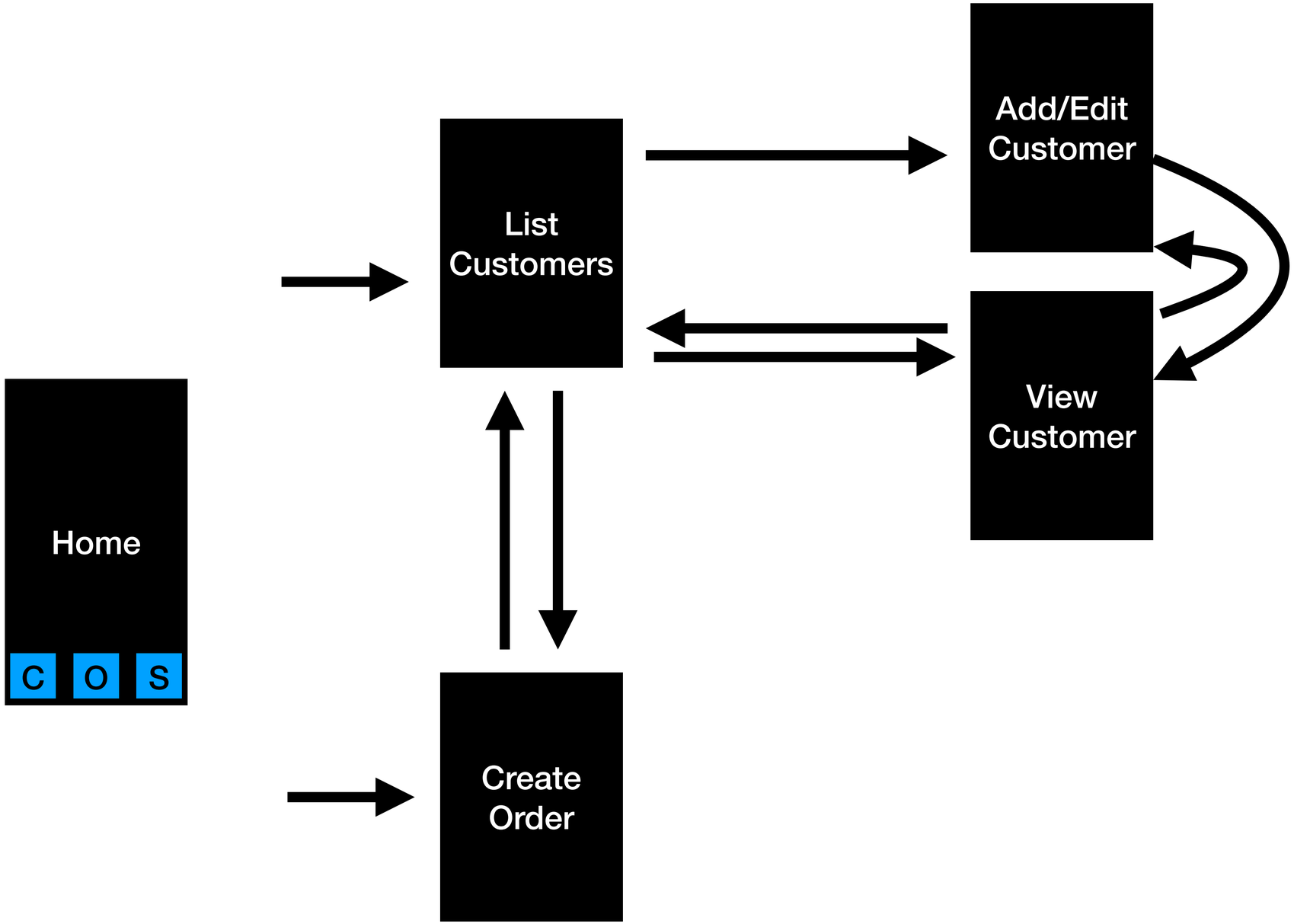

Add/Edit
Customer
List
Customers
View
Customer
Home
C
O
S

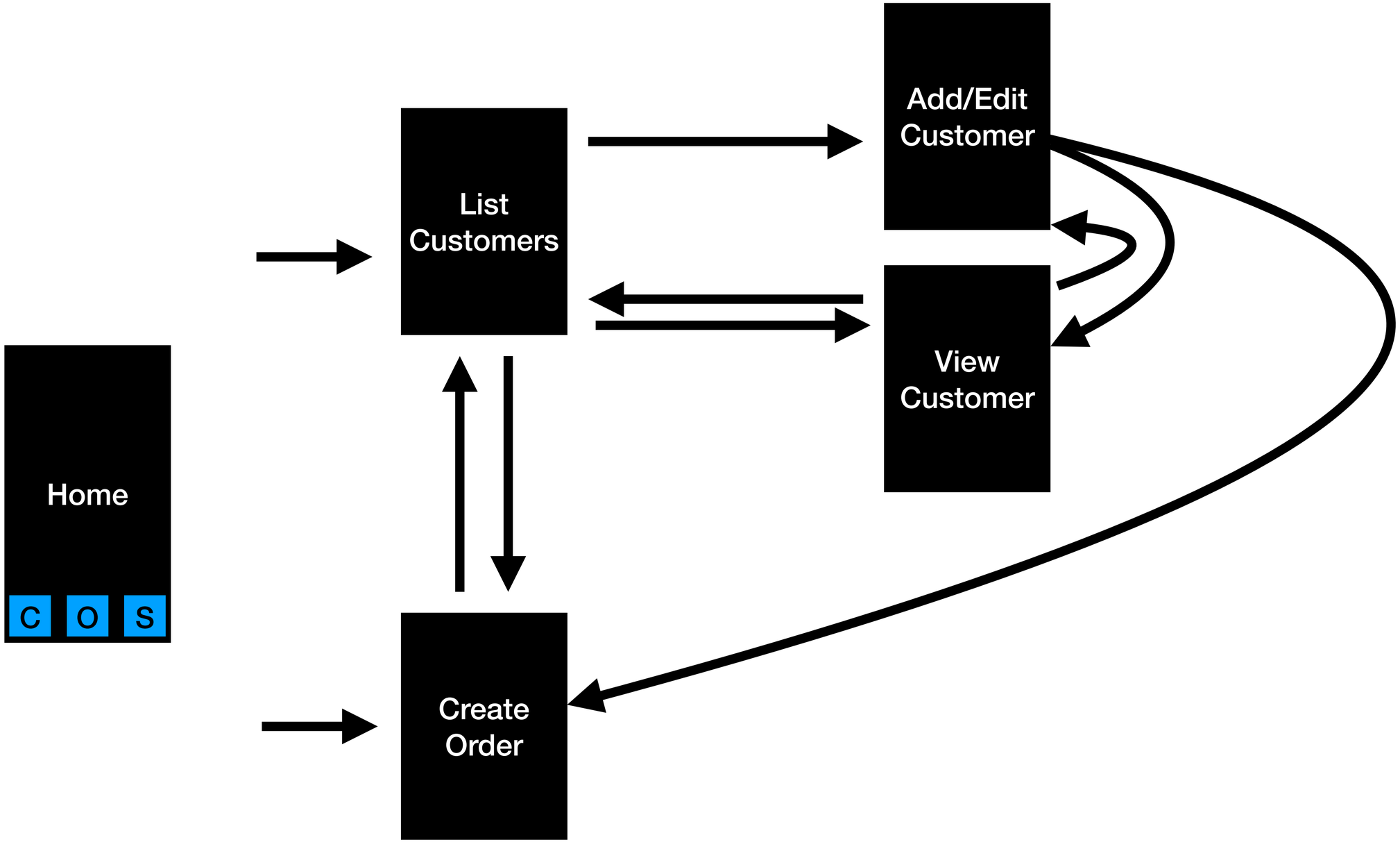

Add/Edit
Customer
List
Customers
View
Customer
Home
C
O
S


Add/Edit
Customer
List
Customers
View
Customer
Home
C
O
S

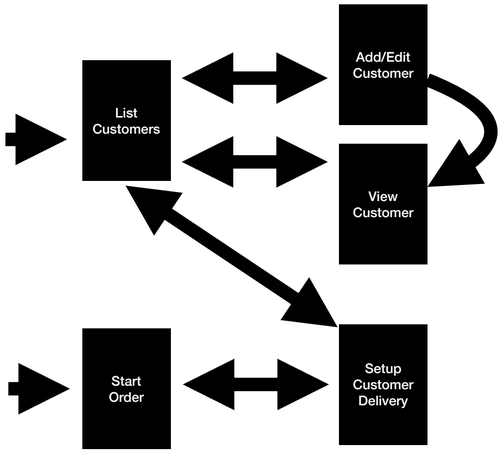

Add/Edit
Customer
List
Customers
View
Customer
Home
C
O
S
Create
Order


Add/Edit
Customer
List
Customers
View
Customer
Home
C
O
S
Create
Order

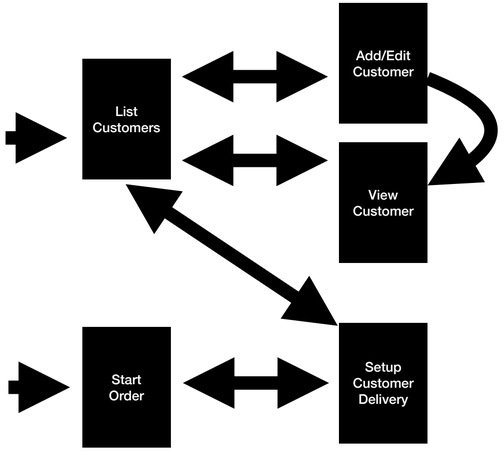

Add/Edit
Customer
List
Customers
View
Customer
Home
C
O
S
Create
Order

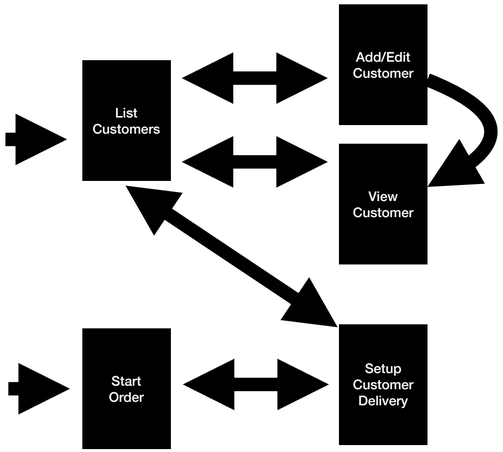

Add/Edit
Customer
List
Customers
View
Customer
Home
C
O
S
Create
Order

Home
C
O
S

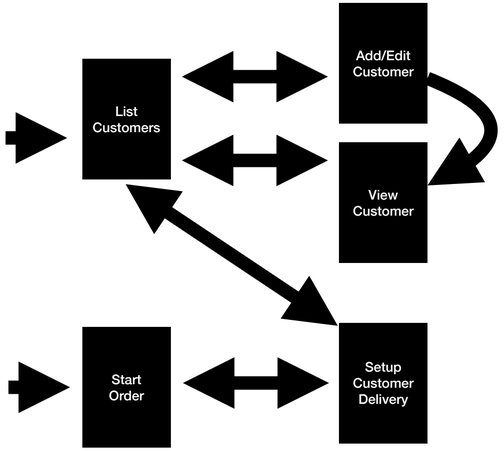

List Customers
Add/Edit Customer
View Customer
Start Order
Setup Customer Delivery

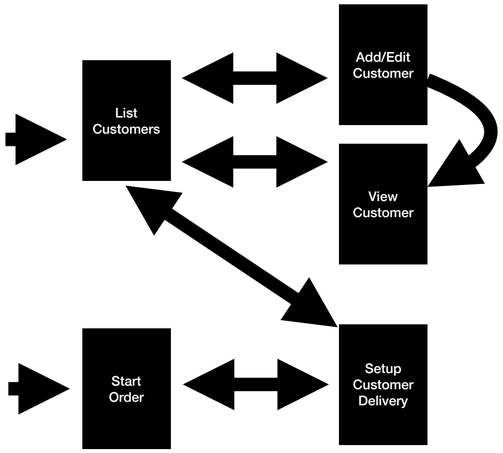
Home
C
O
S

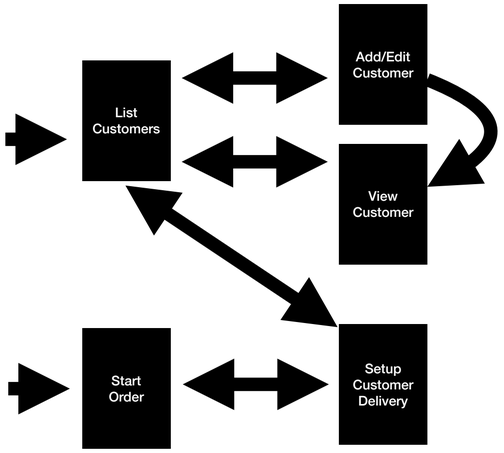

List Customers
Add/Edit Customer
View Customer
Start Order
Setup Customer Delivery


List Customers
Add/Edit Customer
View Customer
Start Order
Setup Customer Delivery

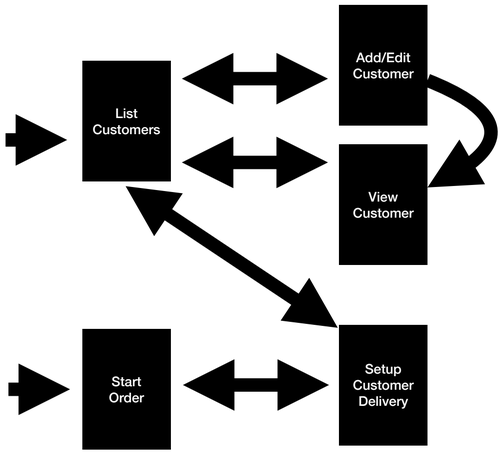
Home
C
O
S

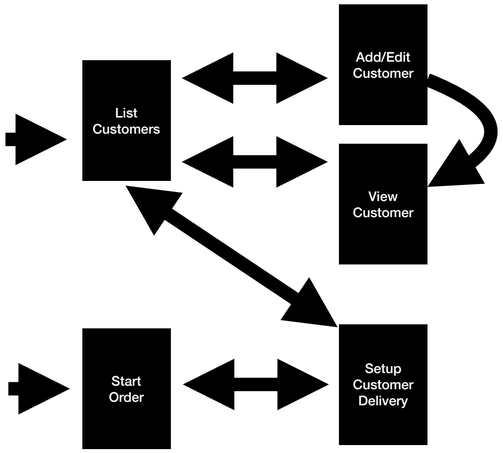

List Customers
Add/Edit Customer
View Customer
Start Order
Setup Customer Delivery

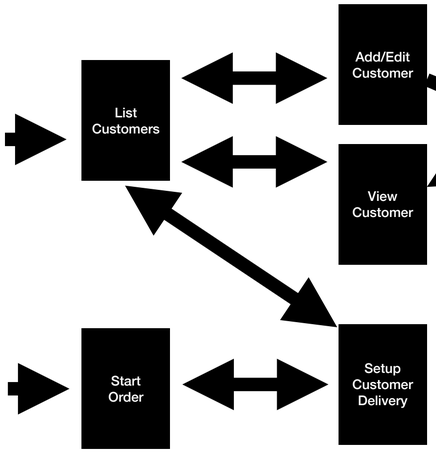

List Customers
Add/Edit Customer
View Customer
Start Order
Setup Customer Delivery

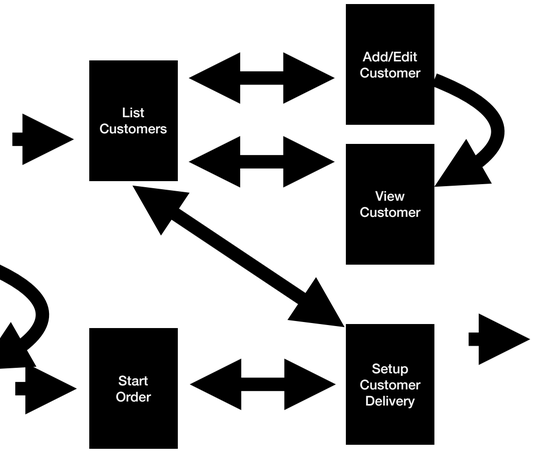

List Customers
Add/Edit Customer
View Customer
Start Order
Setup Customer Delivery

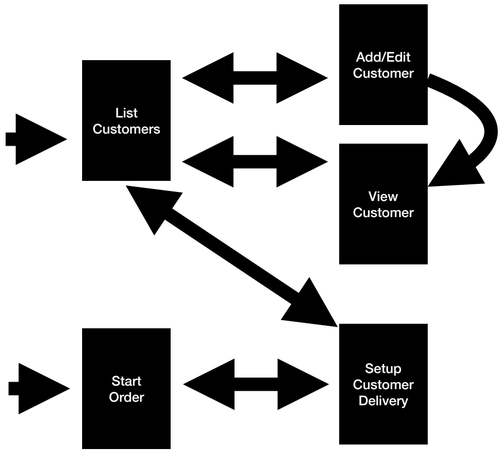
Home
C
O
S


List Customers
Add/Edit Customer
View Customer
Start Order
Setup Customer Delivery


List Customers
Add/Edit Customer
View Customer
Start Order
Setup Customer Delivery


List Customers
Add/Edit Customer
View Customer
Start Order
Setup Customer Delivery


List Customers
Add/Edit Customer
View Customer
Start Order
Setup Customer Delivery


List Customers
Add/Edit Customer
View Customer
Start Order
Setup Customer Delivery

Navigation

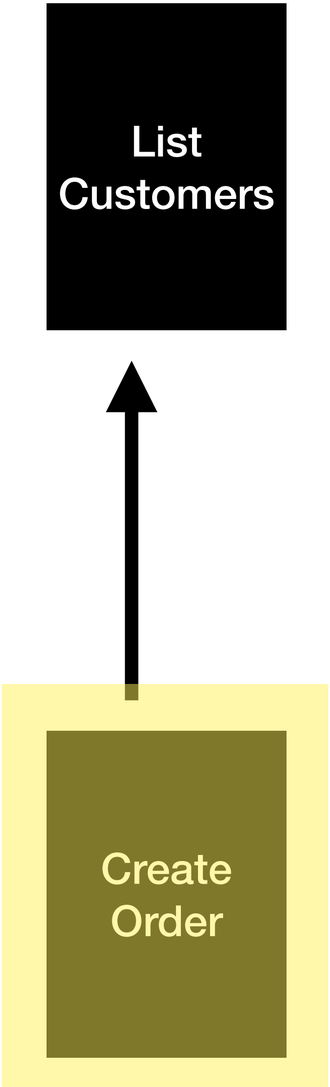
Nav Graph Definition

```
NavHost(navController, startDestination = "home") {
  composable("create_order") {
    CreateOrderScreen(navController)
  }
  composable("list_customers") {
    ListCustomersScreen(navController)
  }
  composable("view_customer/{id}") { ... }
  composable("edit_customer/{id}") { ... }
  composable("add_customer") { ... }
}
```

Navigation

Create Order Screen

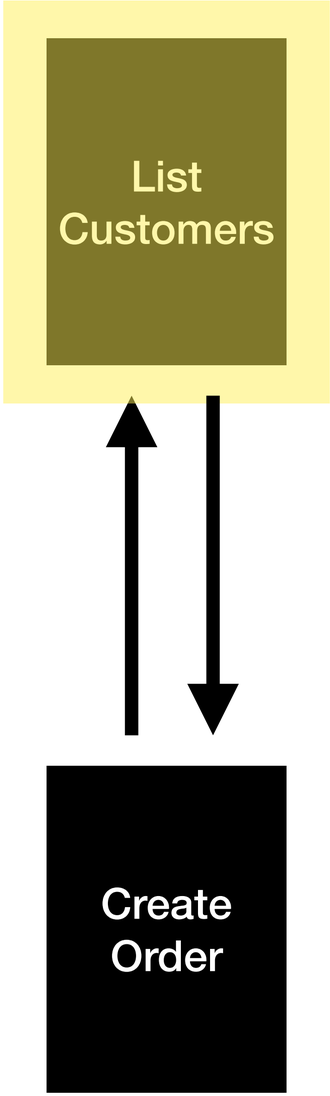
```
Button(onClick = {
  navController.navigate("list_customers")
}) { Text("Select Customer") }

// Handle result
val selectedCustomer = navController
  .currentBackStackEntry
  ?.savedStateHandle
  ?.getLiveData<Customer>("selected_customer")
  ?.observeAsState()
```



Navigation

Add Customer Screen

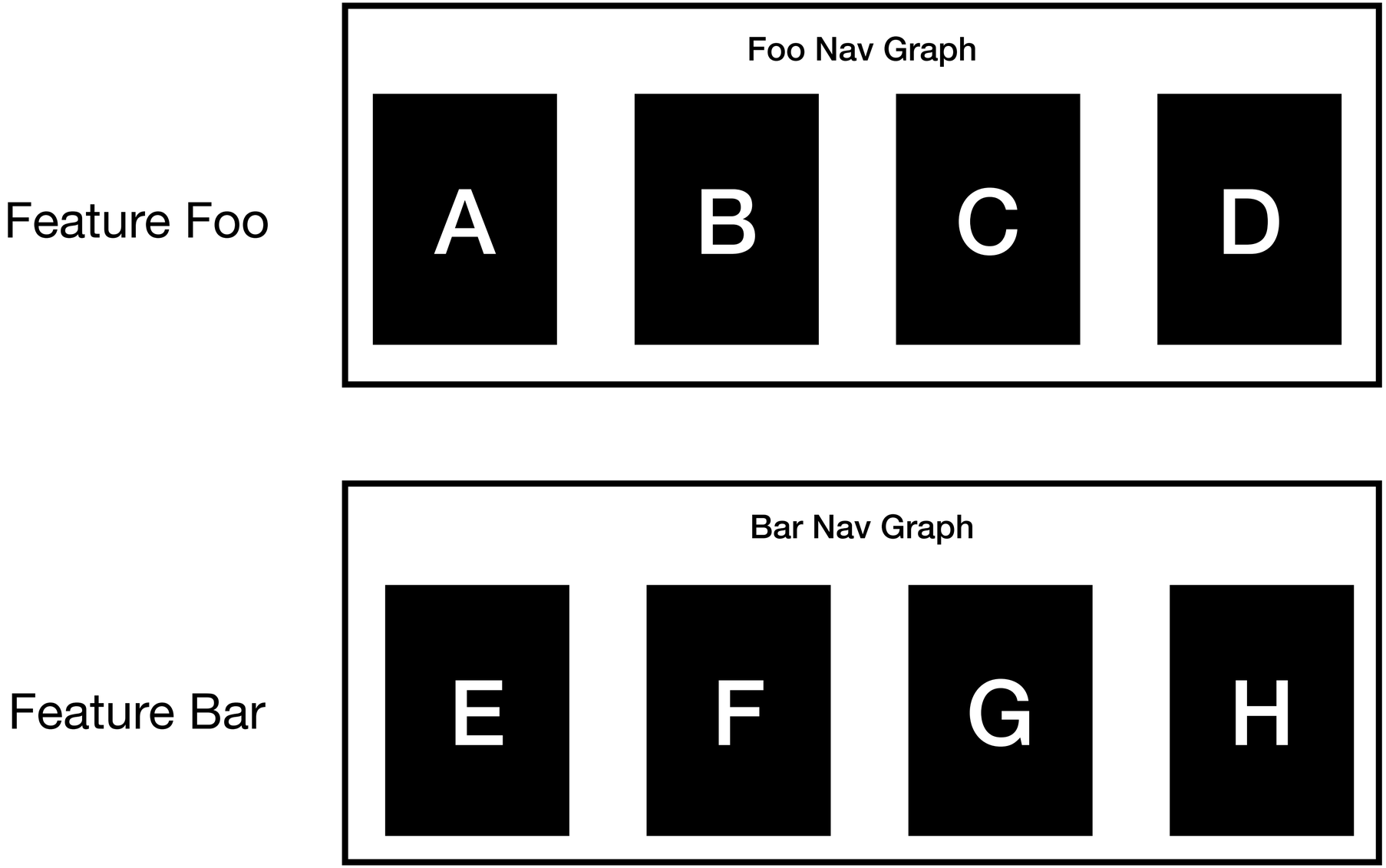
```
onCustomerSelected = { customer ->
  navController.previousBackStackEntry
    ?.savedStateHandle
    ?.set("selected_customer", customer)
  navController.popBackStack()
}
```



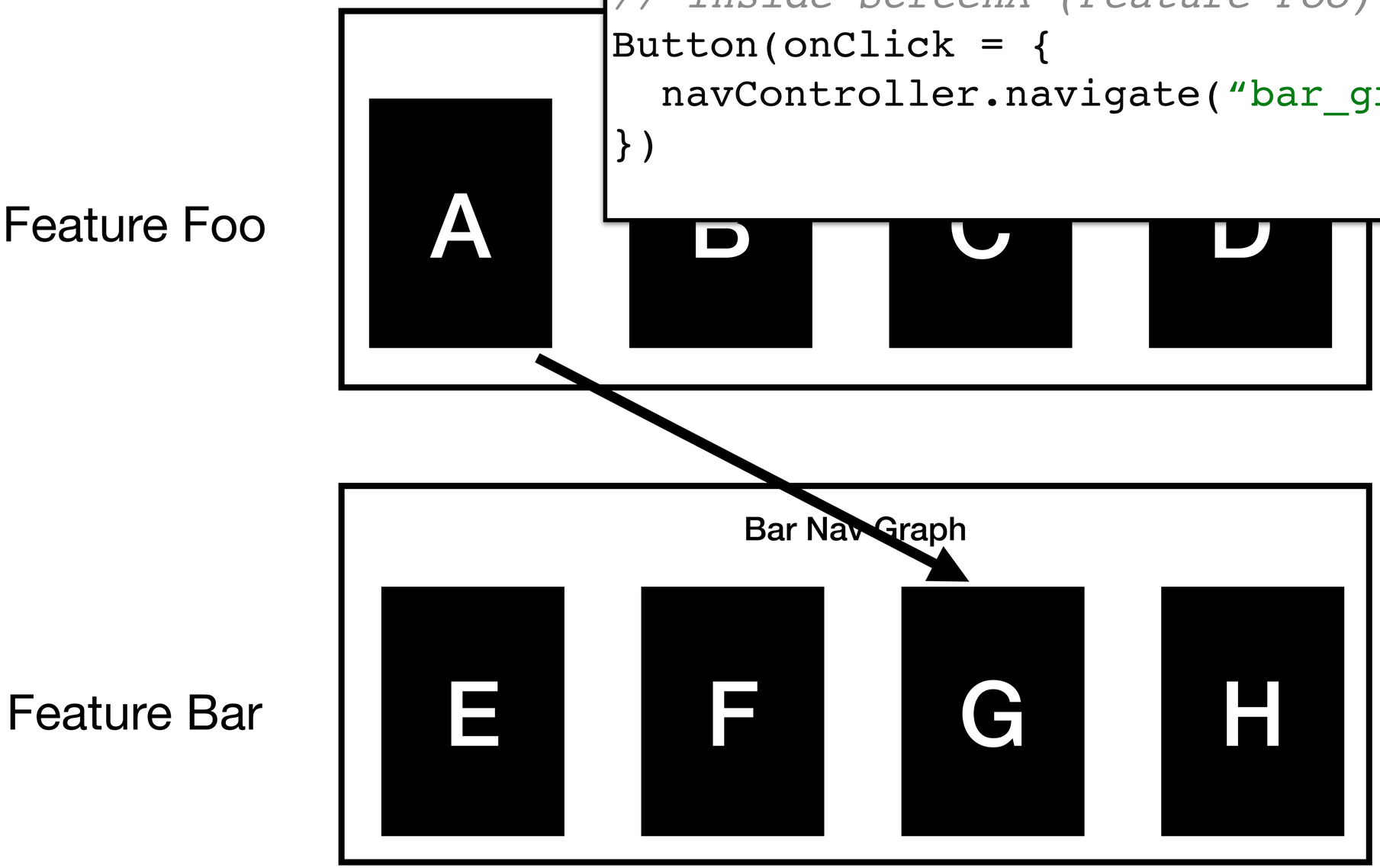
Navigation

❌ Complexity

- Nav Graph logic is spread across screen
- Data passing is clumsy and will start to become inconsistent
- Refactoring later will become hard
- Requires more coordination between teams


Foo Nav Graph
Feature Foo
A
B
C
D
Bar Nav Graph
Feature Bar
E
F
G
H

```
// Inside ScreenA (Feature Foo)
Button(onClick = {
  navController.navigate("bar_graph/G/{id}")
})
```



Navigation

❌ Complexity

- Foo must know Bar’s route structure
- Bar must expose “G” as a deep-linkable route
- You need to bypass Bar’s startDestination ("E")
- You might skip initializing context/state Bar expects from E → F → G
- If G communicates back via SavedStateHandles, the times it’s rendered by F may be add or mutate data in a way that F doesn’t expect.

UI Framework

✅ Recommend

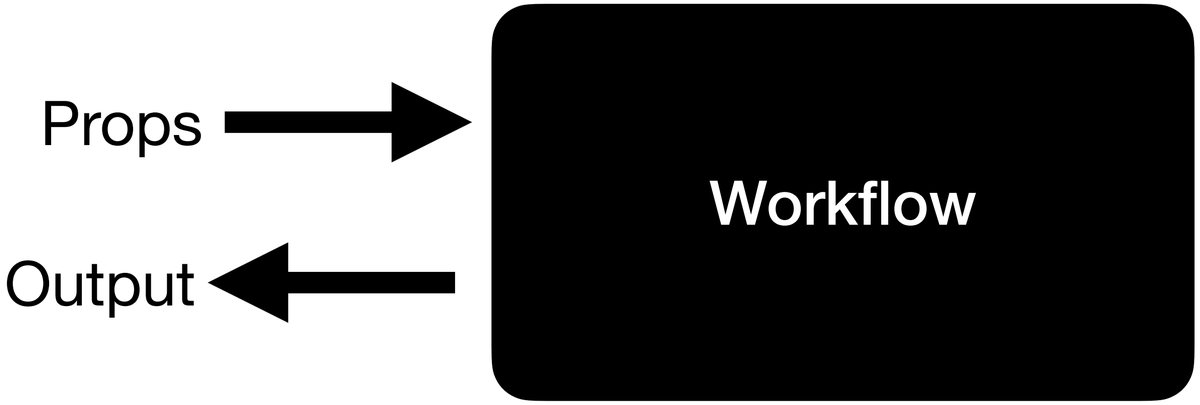
- Strong Contracts
- Strong Coupling between (Navigation, Model and UI State)
  - State transitions drive navigation, not the reverse
  - Navigation is pure function of state

Architecture: Workflow

Unidirectional Framework for building UX

Architecture: Workflow

Unidirectional Framework for building UX

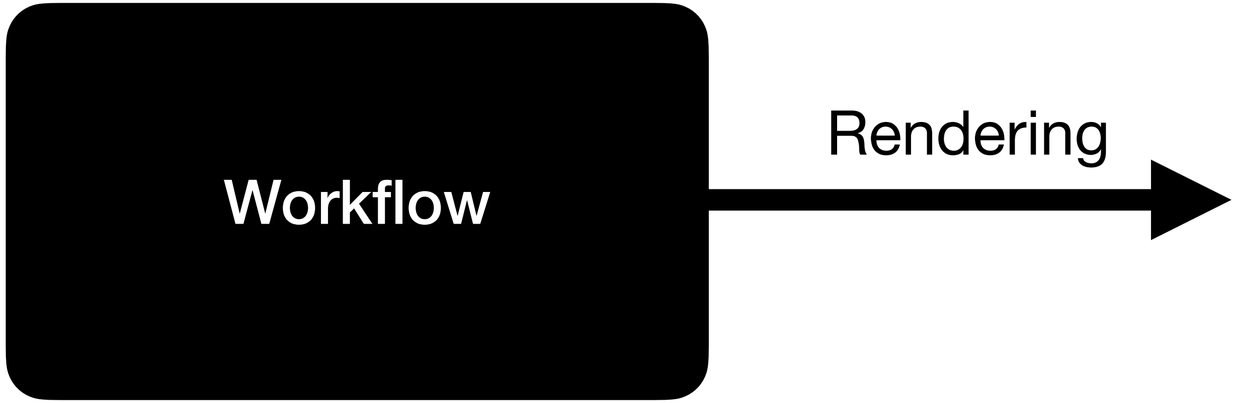


Architecture: Workflow

Unidirectional Framework for building UX

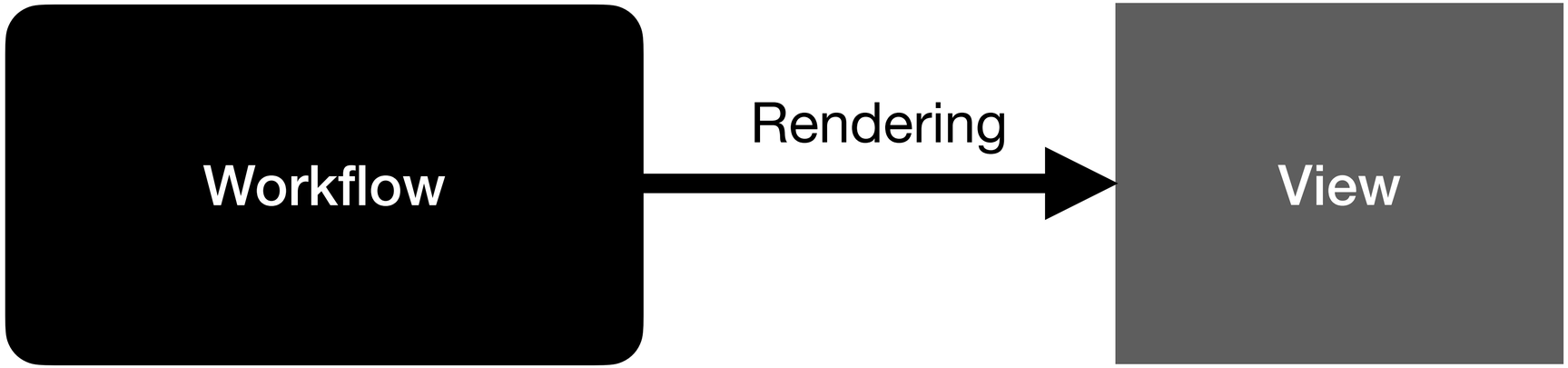
Architecture: Workflow

Workflows Produce Renderings



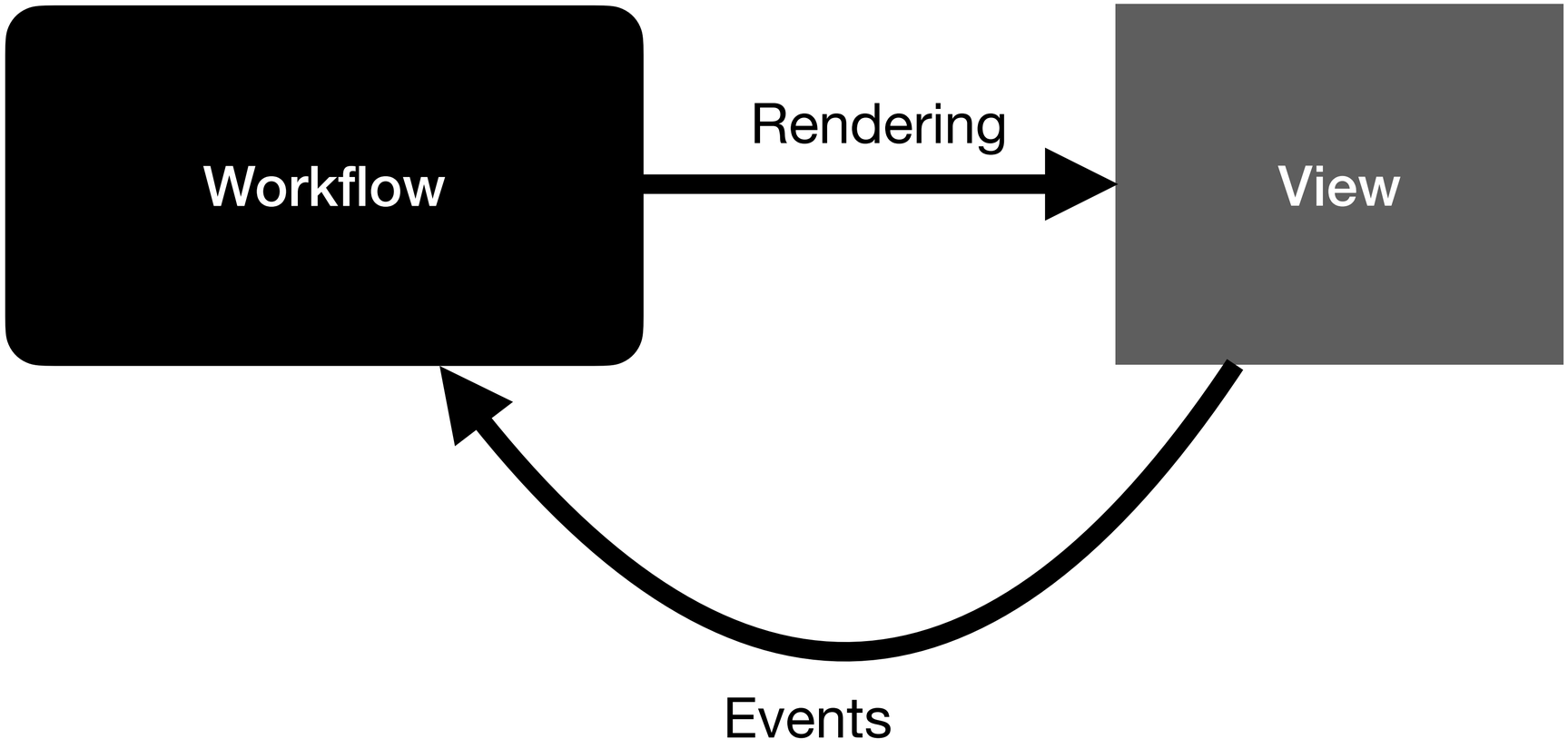
Architecture: Workflow

Renderings used to build Views



Architecture: Workflow

Views Produce Events



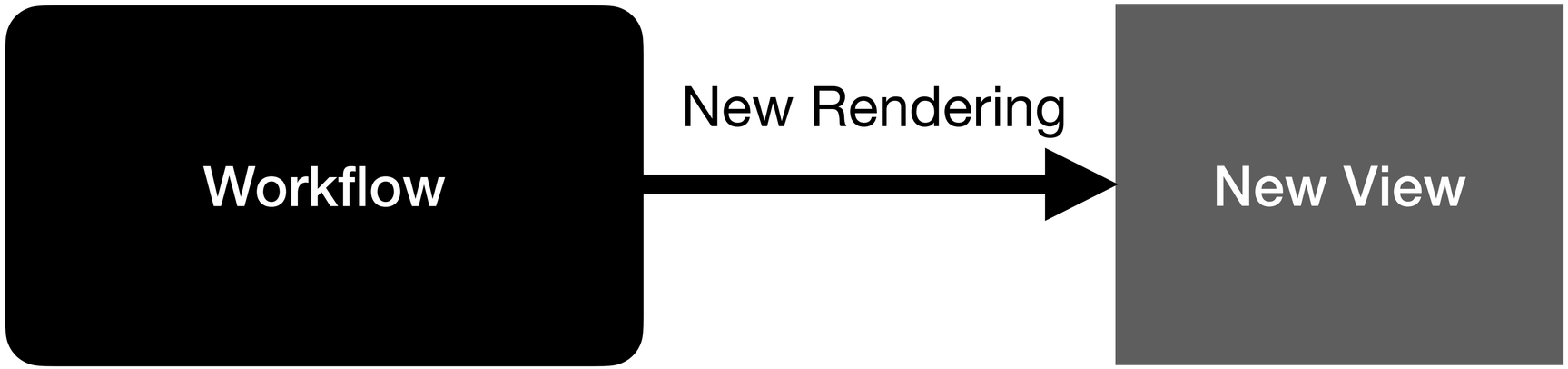
Architecture: Workflow

Queue Actions

Action
{state = newState}

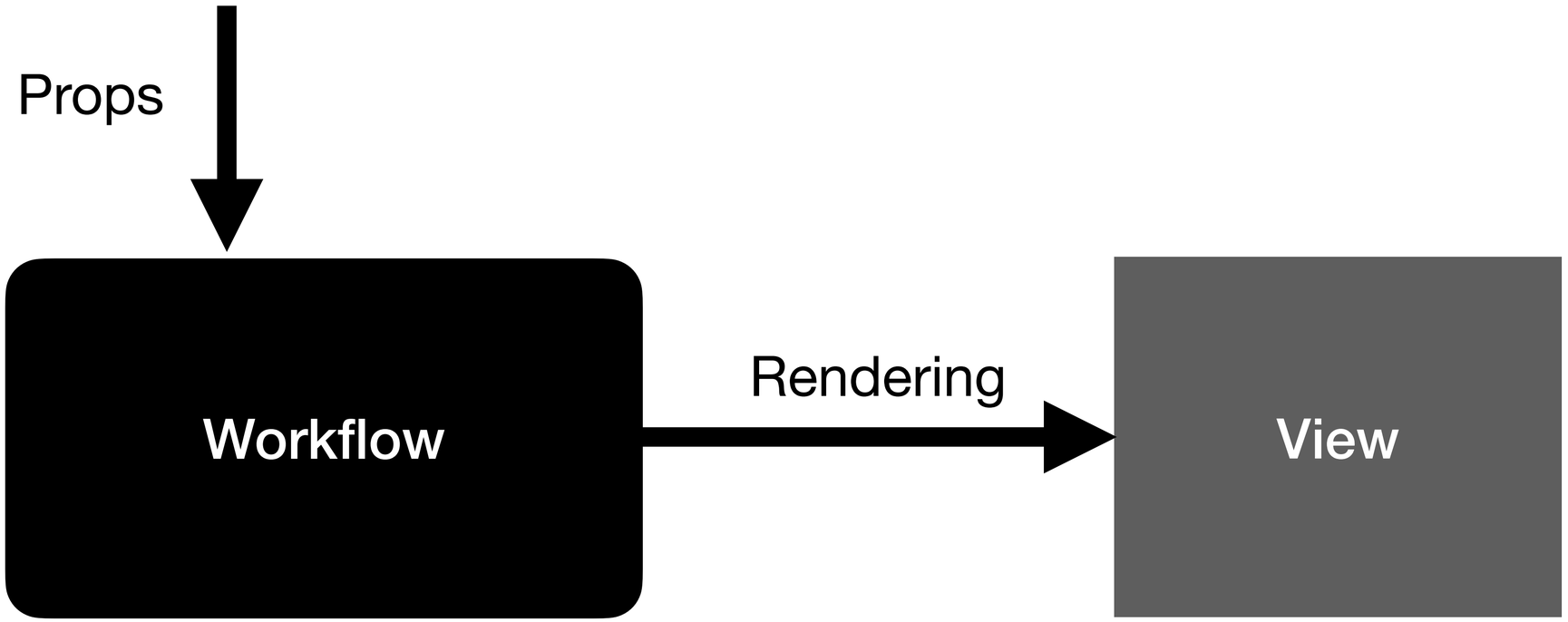
Architecture: Workflow

Produce new renderings and views



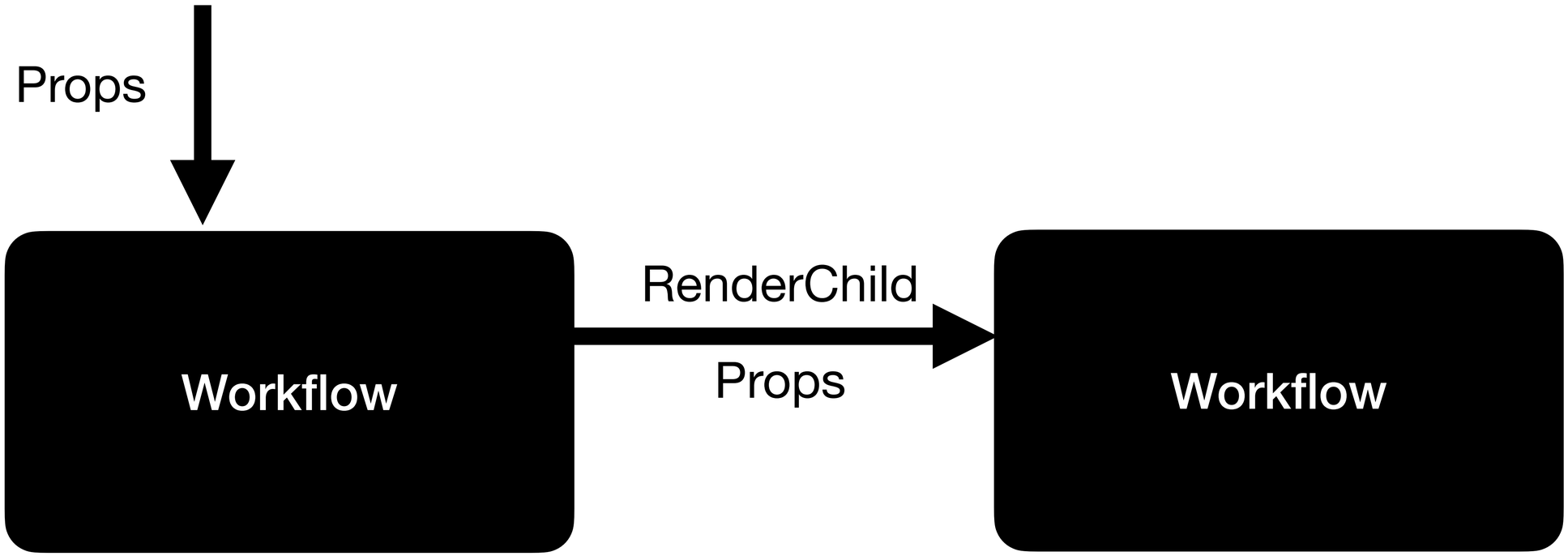
Architecture: Workflow

Props



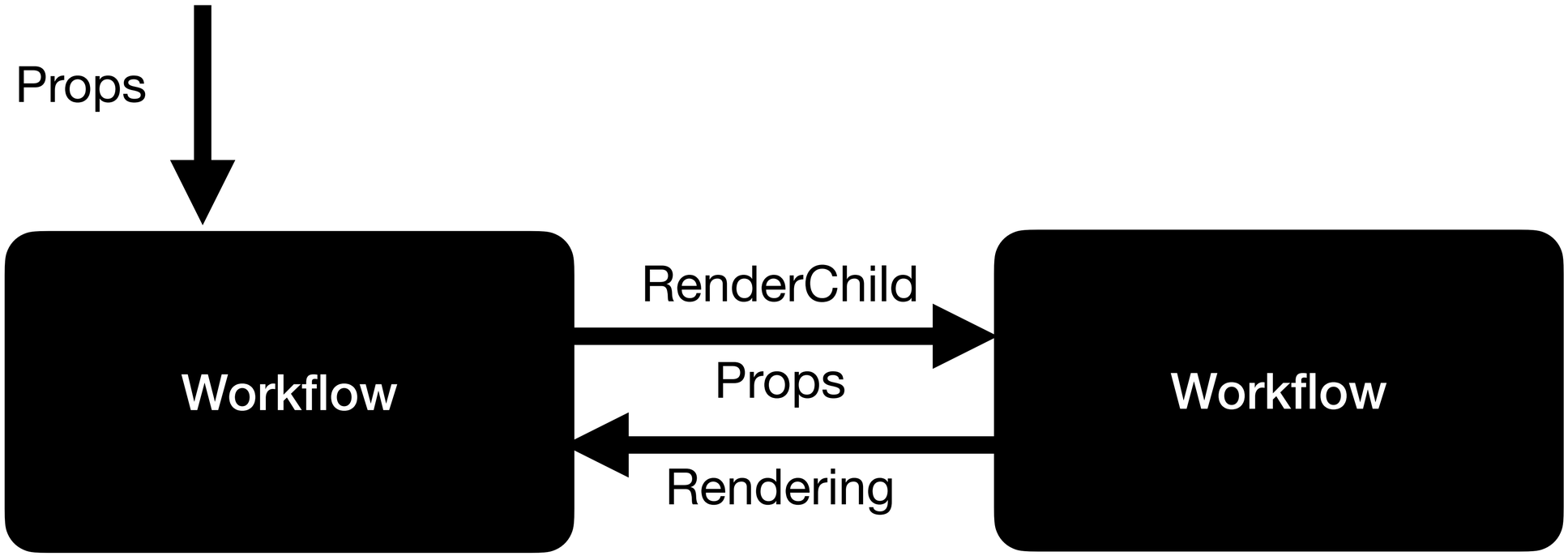
Architecture: Workflow

Render Children Workflows



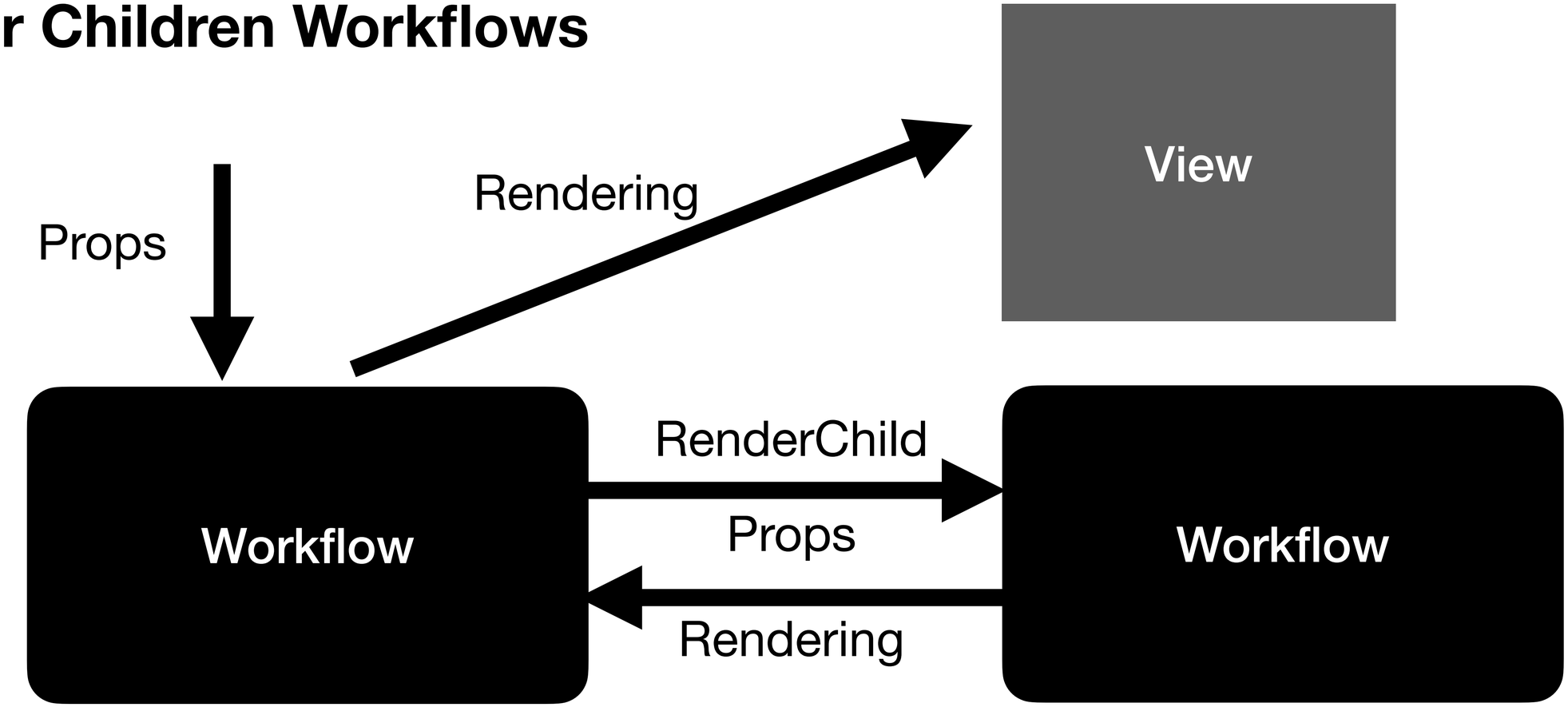
Architecture: Workflow

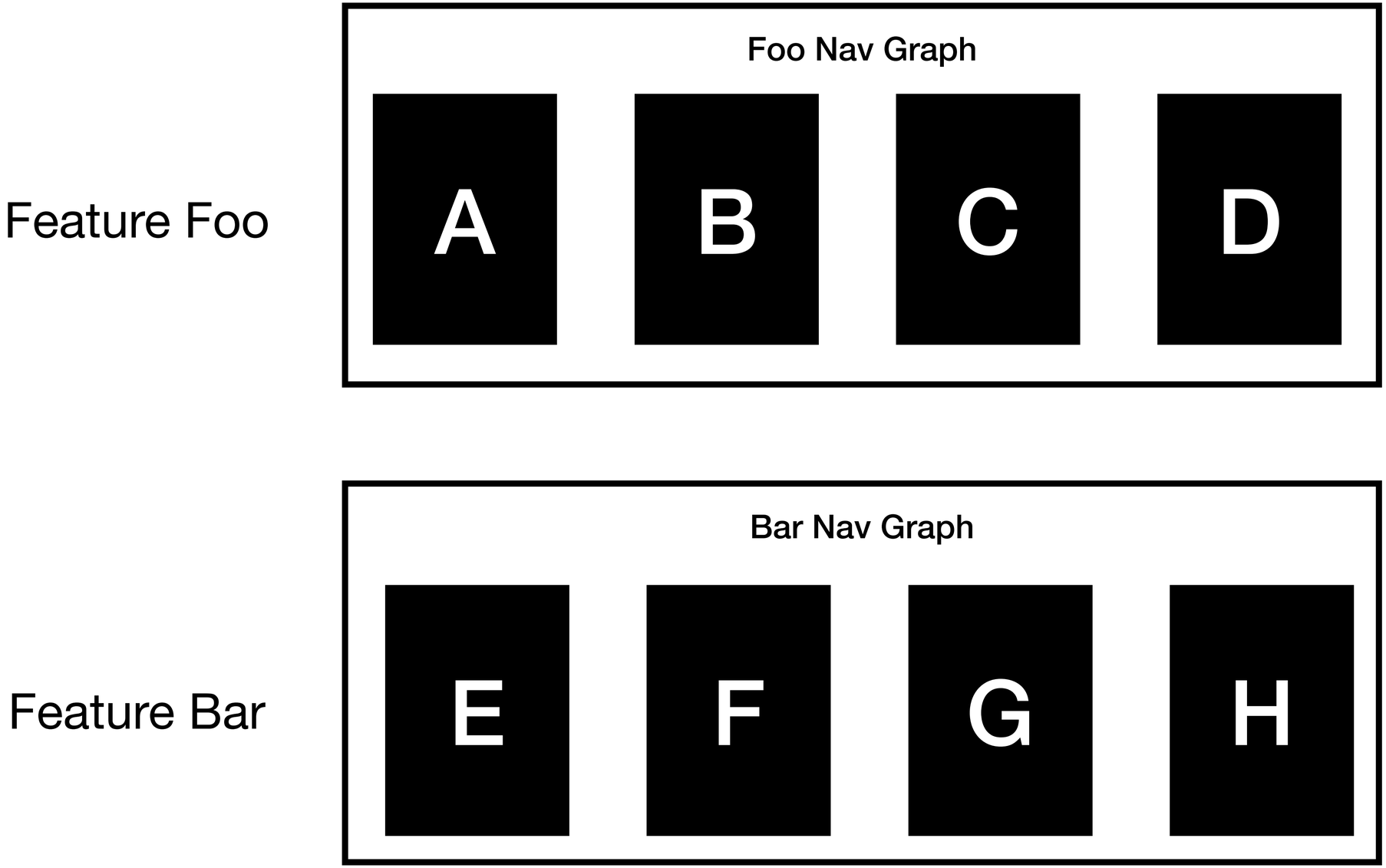
Render Children Workflows



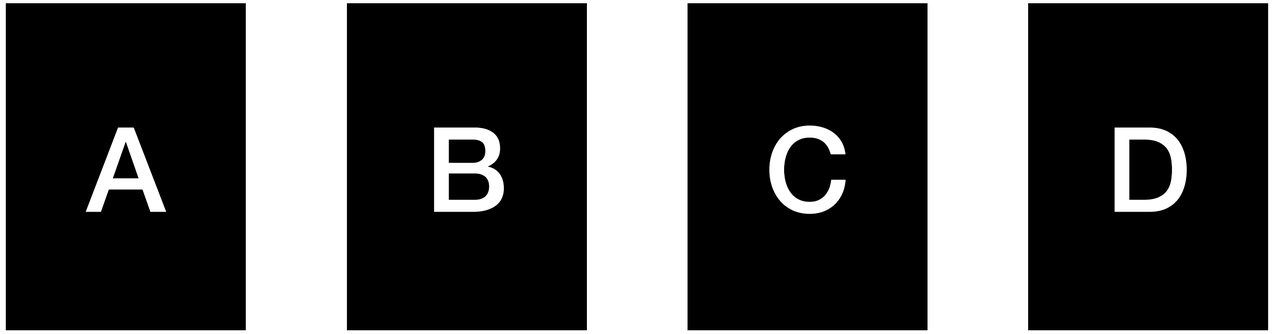
Architecture: Workflow

Render Children Workflows



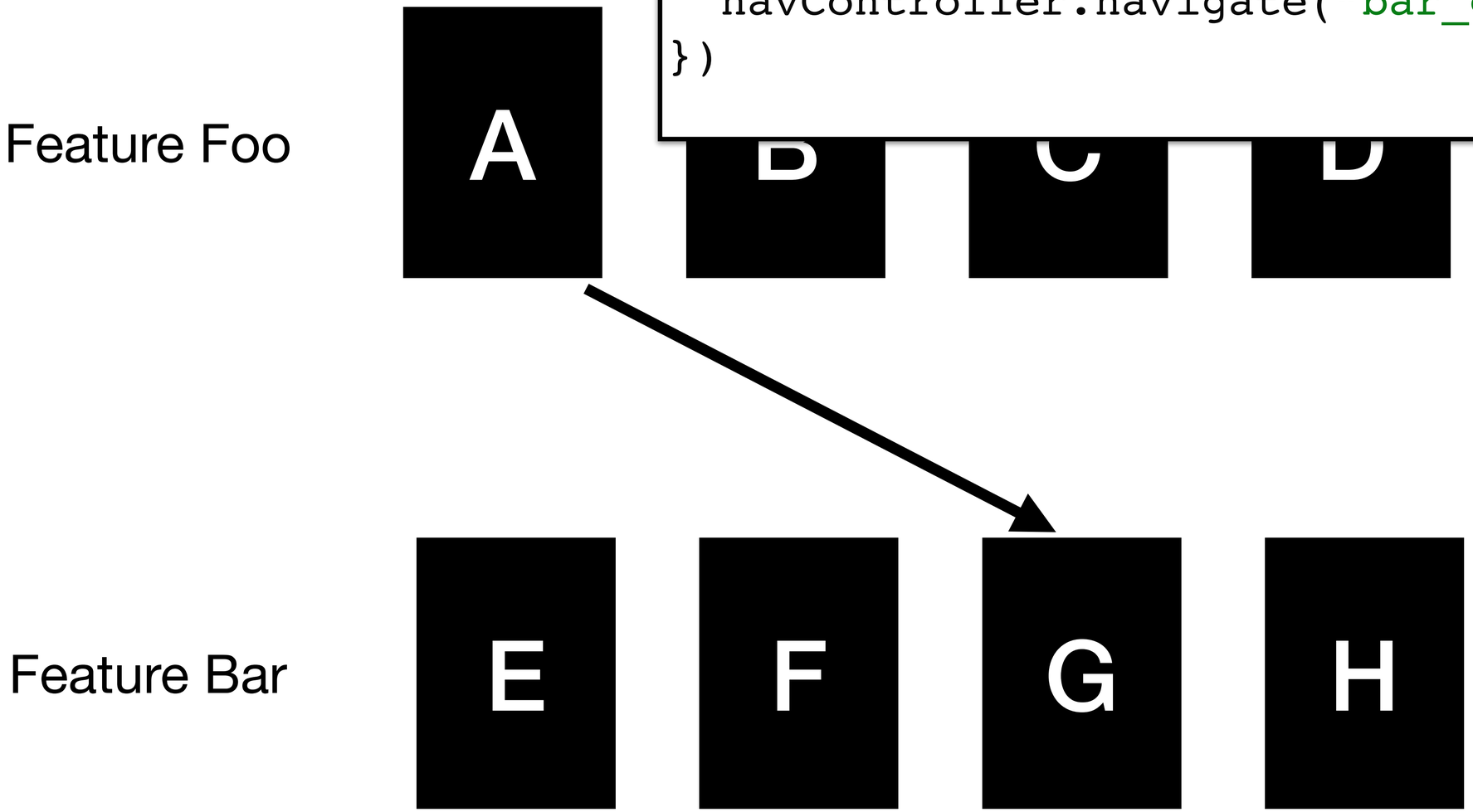

Foo Nav Graph
Feature Foo
A
B
C
D
Bar Nav Graph
Feature Bar
E
F
G
H

Feature Foo

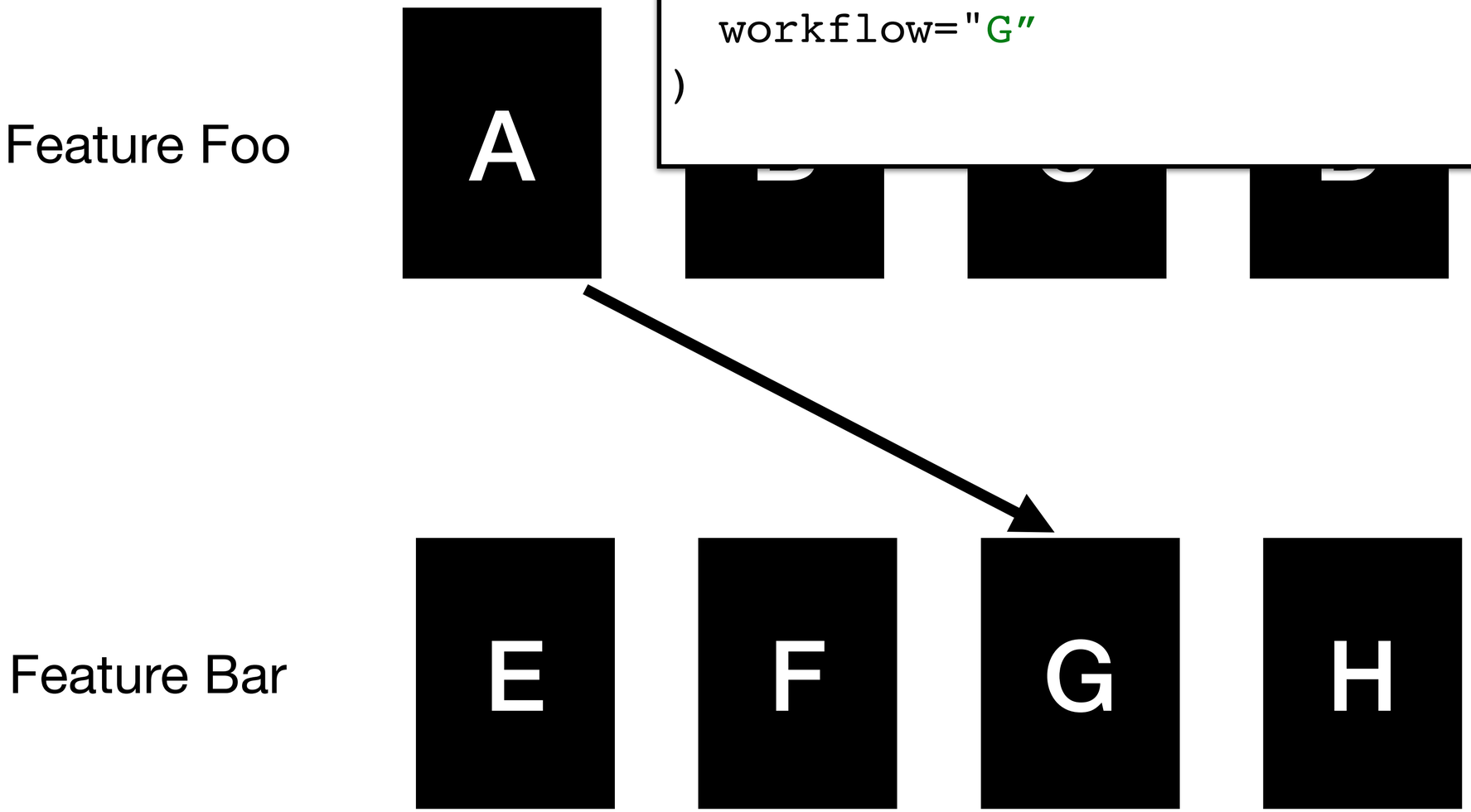


Feature Bar

```
// Inside ScreenA (Feature Foo)
Button(onClick = {
  navController.navigate("bar_graph/G/{id}")
})
```

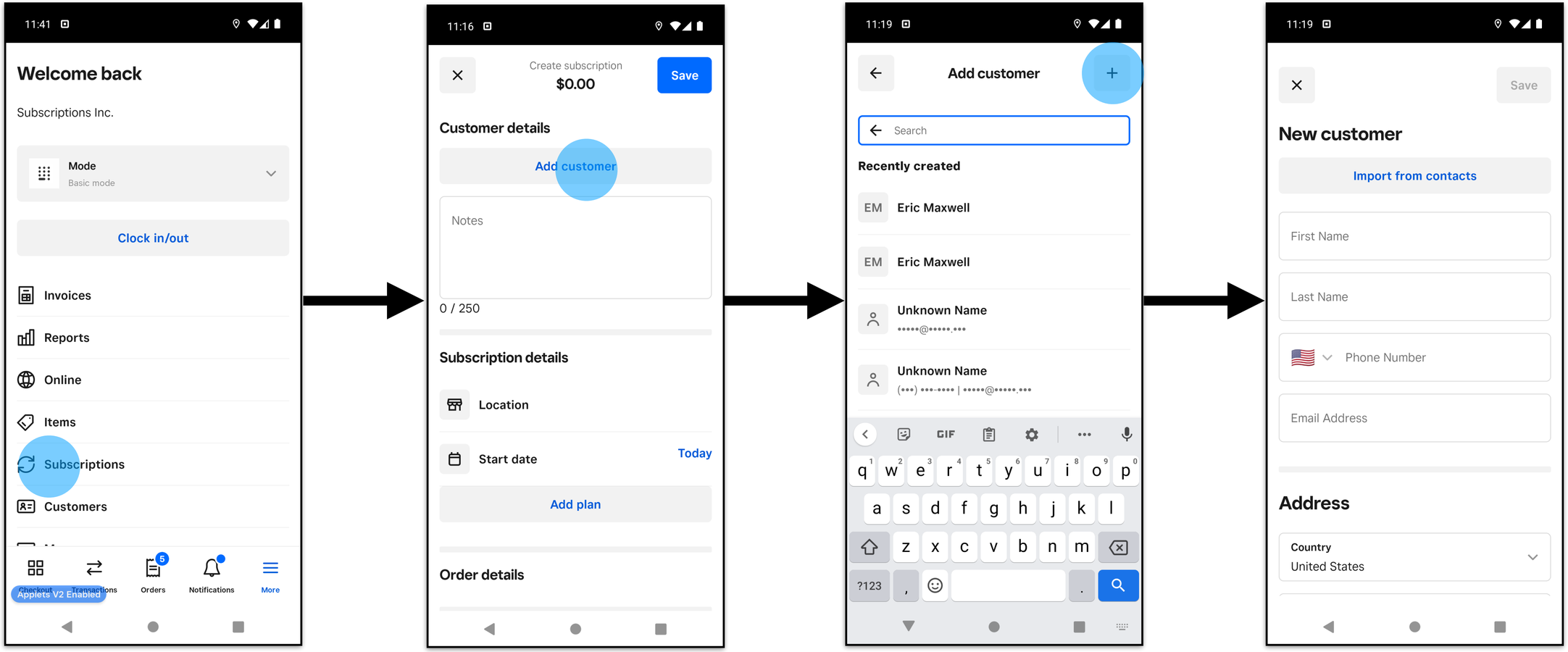


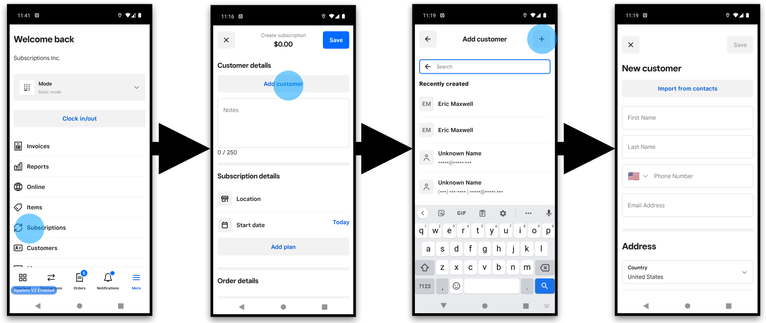
```
// Inside ScreenA (Feature Foo)
rendering = context.renderChild(
  props = GProps(id="…")
  workflow="G"
)
```

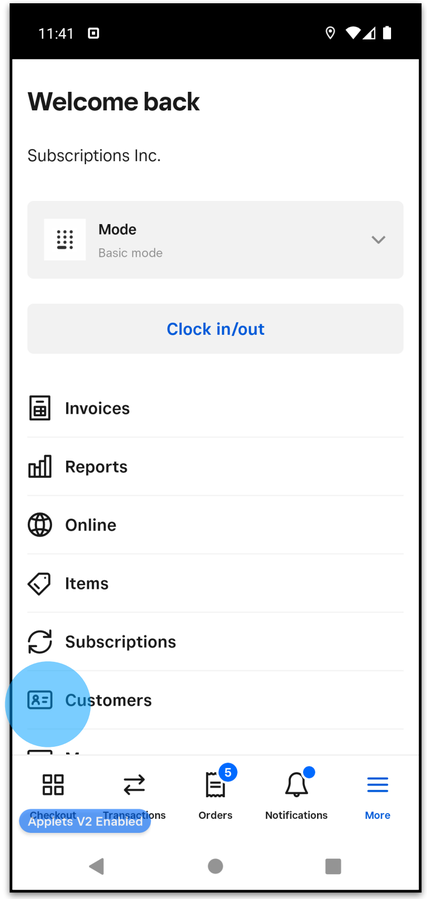


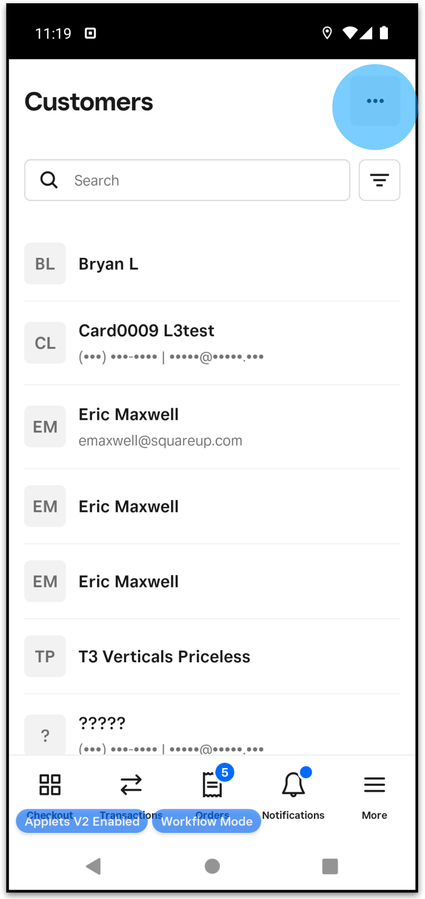
Architecture: Workflow

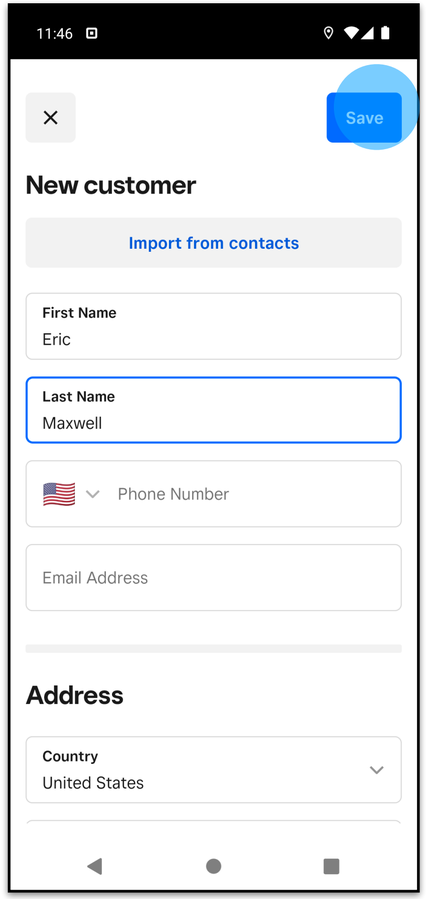
Customer Workflow

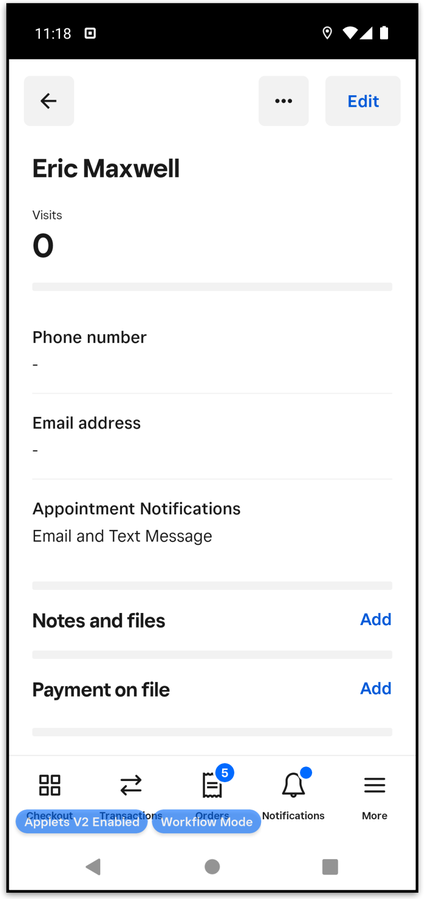


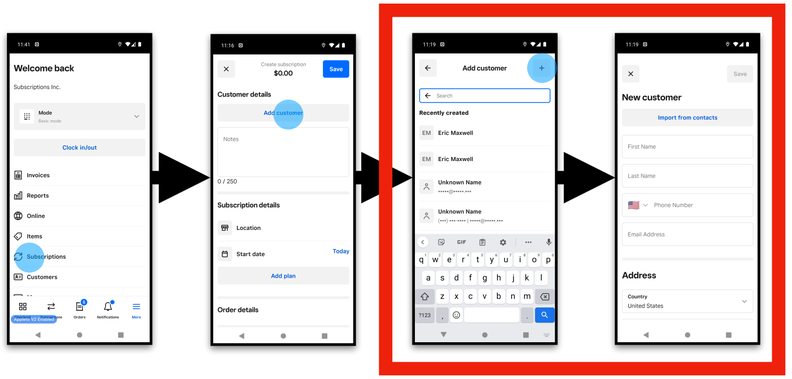


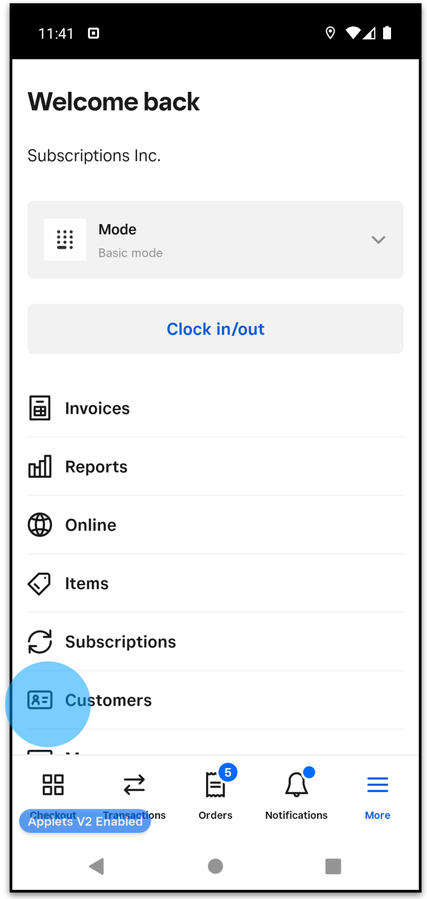

11:41
Welcome back
Subscriptions Inc.
Mode
Basic mode
Clock in/out
Invoices
Reports
Online
Items
Subscriptions
Customers
Checkout
Transactions
Orders
Notifications
More
Applets V2 Enabled

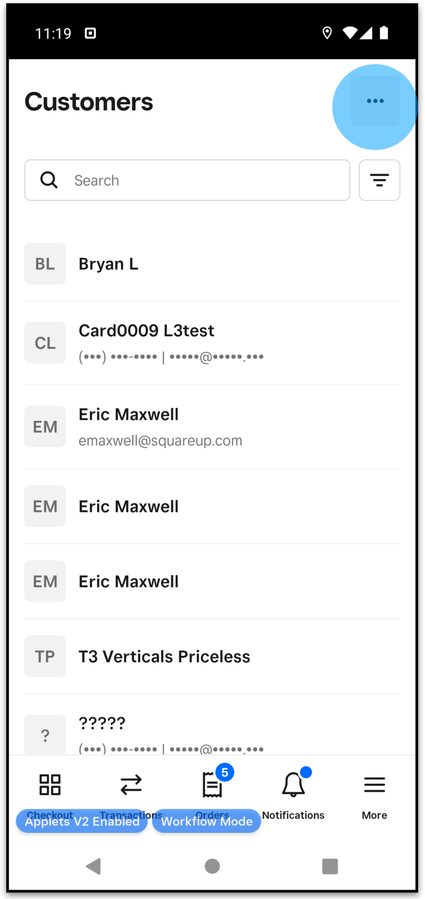

11:19
Customers
Search
BL Bryan L
CL Card0009 L3test
(•••) •••-•••• | •••••@•••••.•••
EM Eric Maxwell
emaxwell@squareup.com
EM Eric Maxwell
EM Eric Maxwell
TP T3 Verticals Priceless
? ?????
(•••) •••-•••• | •••••@•••••.•••
Checkout
Transactions
Orders
Notifications
More
Applets V2 Enabled
Workflow Mode

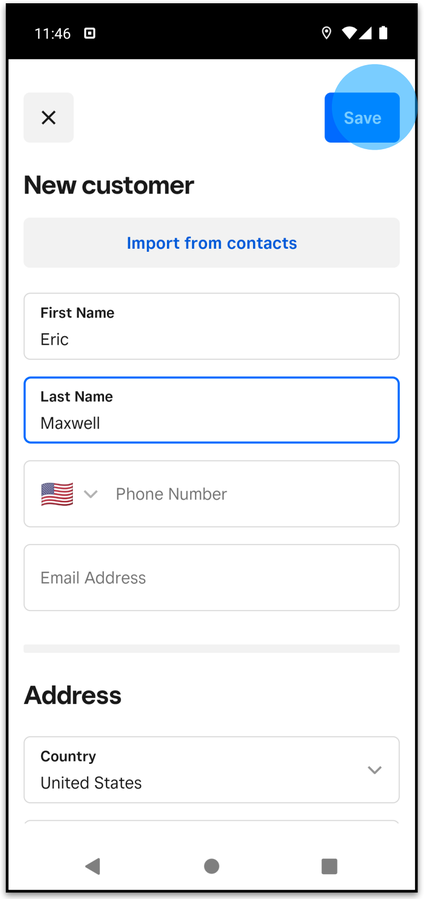

11:46
Save
New customer
Import from contacts
First Name
Eric
Last Name
Maxwell
Phone Number
Email Address
Address
Country
United States

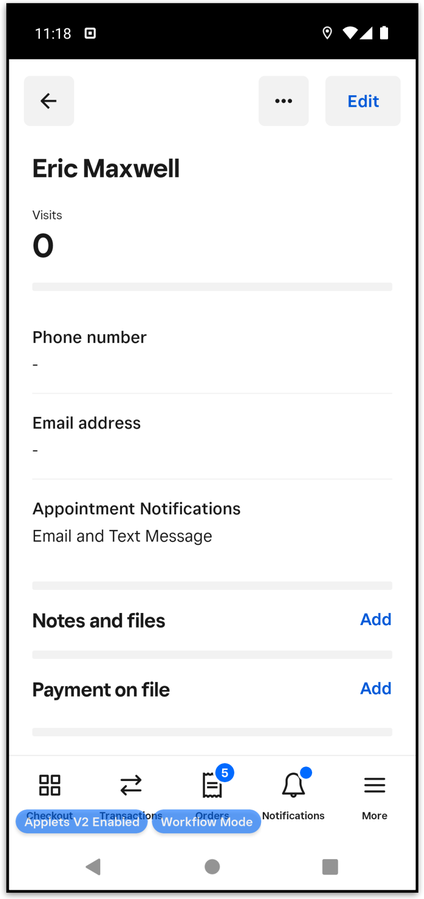

11:18
Edit
Eric Maxwell
Visits
0
Phone number
-
Email address
-
Appointment Notifications
Email and Text Message
Notes and files
Add
Payment on file
Add
Checkout
Transactions
Orders
Notifications
More
Applets V2 Enabled
Workflow Mode


Welcome back
Customer details
Add customer
Recently created
New customer
Import from contacts
Address


11:41
Welcome back
Subscriptions Inc.
Mode
Basic mode
Clock in/out
Invoices
Reports
Online
Items
Subscriptions
Customers
Checkout
Transactions
Orders
Notifications
More
Applets V2 Enabled


11:19
Customers
Search
BL Bryan L
CL Card0009 L3test
(•••) •••-•••• | •••••@•••••.•••
EM Eric Maxwell
emaxwell@squareup.com
EM Eric Maxwell
EM Eric Maxwell
TP T3 Verticals Priceless
? ?????
(•••) •••-•••• | •••••@•••••.•••
Checkout
Transactions
Orders
Notifications
More
Applets V2 Enabled
Workflow Mode


11:46
Save
New customer
Import from contacts
First Name
Eric
Last Name
Maxwell
Phone Number
Email Address
Address
Country
United States


11:18
Edit
Eric Maxwell
Visits
0
Phone number
-
Email address
-
Appointment Notifications
Email and Text Message
Notes and files
Add
Payment on file
Add
Checkout
Transactions
Orders
Notifications
More
Applets V2 Enabled
Workflow Mode

Architecture: Workflow

What does this look like

- Workflow

```
interface ContactSearchWorkflow :
  Workflow<ContactSearchProps, ContactSearchOutput, ContactSearchRendering>
```

- Props

```
data class ContactSearchProps(
  val title: TextData<*> = TextData.empty,
  val initialSearchQuery: String = "",
)
```

Architecture: Workflow

What does this look like

- Output

```
sealed class ContactSearchOutput {

 data object Canceled : ContactSearchOutput()

 data class ContactSelected(val contact: Contact) : ContactSearchOutput()
}
```

Architecture: Workflow

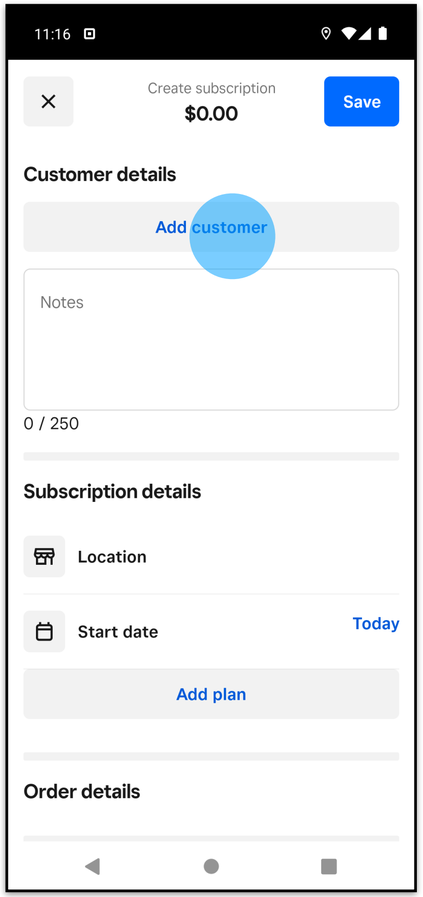
What does this look like

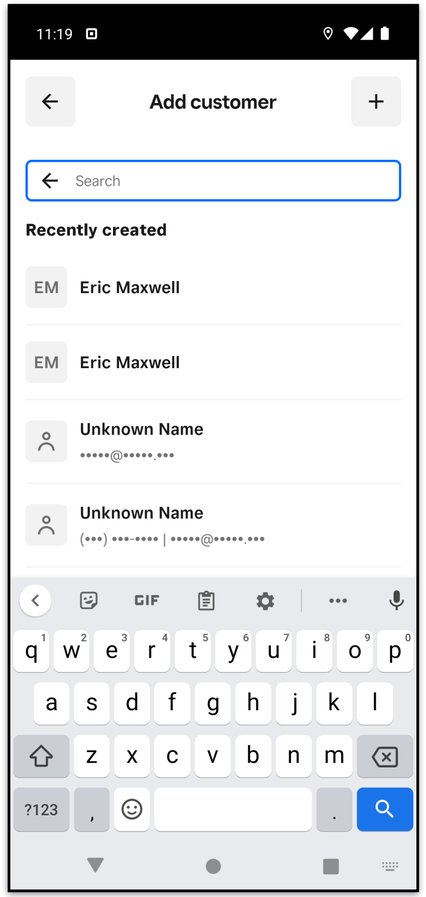
```
class SubscriptionAddWorkflow @Inject constructor(
    private val contactSearchWorkflow: ContactSearchWorkflow
) {
   ...
  override fun render(
    renderProps: SubscriptionAddProps,
    renderState: SubscriptionAddScreenState,
    context: RenderContext
  ): MarketStack {
    return when (renderState.phase) {
      is Idle -> renderIdle(context, renderState, renderProps)
      is ChoosingCustomer -> renderChooseCustomer(context)
      is ChoosingPlan -> renderChoosePlan(context, renderState)
      ...
    }
    }
   }
}
```

Architecture: Workflow

What does this look like

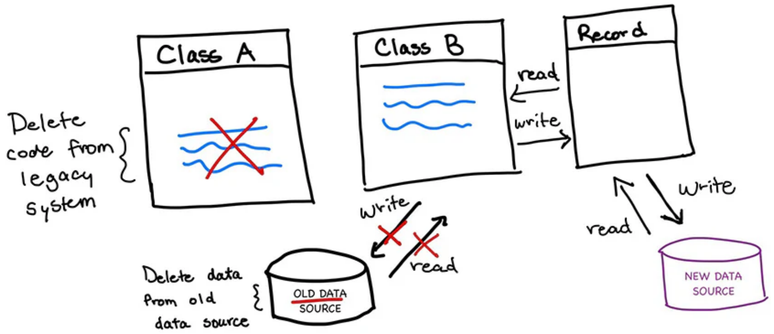
```
fun renderChooseCustomer(
    renderProps: SubscriptionAddProps,
    renderState: SubscriptionAddScreenState,
    context: RenderContext
  ): MarketStack {

      val props = ContactSearchProps(...)

      return context.renderChild(
          workflow = contactSearchWorkflow,
          props = props) { output ->
           action {
             state = when (output) {
                is ContactSelected -> state.copy(
                  phase = Idle,
                  contact = output.contact
             )
           }
          }
        }
      }
}
```





Processes

Guard Rails for Consistency



Approaching Refactoring

Guard Rails

Consistent Architecture

- How can you ensure developers are using
  - Preferred frameworks (for database, ui, logging, concurrency, etc.)
  - Avoiding foot guns
  - Proper code hygiene

Consistent Architecture

Options not so helpful

- Establish code owners and block PRs?
- Documentation?
- Training?

Consistent Architecture

Options

- Establish code owners and block PRs?
  - Huge bottle neck, can wait days or week 😬
- Documentation?
- Training?

Consistent Architecture

Options

- Establish code owners and block PRs?
  - Huge bottle neck, can wait days or week 😬
- Documentation?
  - Nobody will discover / read it
- Training?

Consistent Architecture

Options

- Establish code owners and block PRs?
  - Huge bottle neck, can wait days or week 😬
- Documentation?
  - Nobody will discover / read it
- Training?
  - Will be lost in a quarter

Consistent Architecture

Block Approach

- 1 PR approval from any Android developer
- Owning teams are notified, but not required
- Lean heavily into static analysis tools such as ktlint, checkstyle
- Don’t stop there use Custom Lints => we have 45

Consistent Architecture

Block Approach

```
NoKotlinTestingRule
NoModuleAnnotationOnCompanionObjects
NoNewFeatureFlagFeatures
NoParcelizedFunctionsRule
NoPhoneNumberUtilInstanceCreation
NoRememberAsyncImagePainterRule
NoTimberImportRule
NoVersionCodeImports
NoViewTreeSetRule
NoWorkflowRenderingRule
PreferOnPlacedNodeRule
ProguardReflectionAccessorsRule
RedundantBoundTypeRule
SetBackHandlerImportRule
```

- 1 PR approval from any Android developer
- Owning teams are notified, but not required
- Lean heavily into static analysis tools such as ktlint, checkstyle
- s =

```
AnnotationImportRule
BewareRunCatchingRule
BindsAndProvidesMethodsMustSpecifyReturnType
ContributesFeatureFlagRule
DaggerModulesOrdering
ErrorWithLambda
IncompatibleWithAnvilRule
NoAndroidParcelImportRule
NoBackPressedHandlersRule
NoBlockingGetRule
NoComposeViewRule
NoCoroutineLaunchInTryBlockRule
```

```
NoCoroutineScopeRule
NoGlobalScopeRule
NoHardcodedDispatchers
NohoStandardColorsRule
NoInnerRImport
NoInternalModifierInDaggerModules
NoInternalModifierOnCompanionObjectsInDaggerModules
NoJavaTimeImportRule
NoJunitAssertions
NoJvmAssertionsRule
NoJvmStaticInDaggerModules
NoKotlinCoroutineRx2AsFlowRule
NoKotlinReflectJvmName
```

Consistent Architecture

Block Approach

- 1 PR approval from any Android developer
- Owning teams are notified, but not required
- Lean heavily into static analysis tools such as ktlint, checkstyle
- Don’t stop there use Custom Lints => we have 45
- Module validation rules
- Gradle DSL and rules

Consistent Architecture

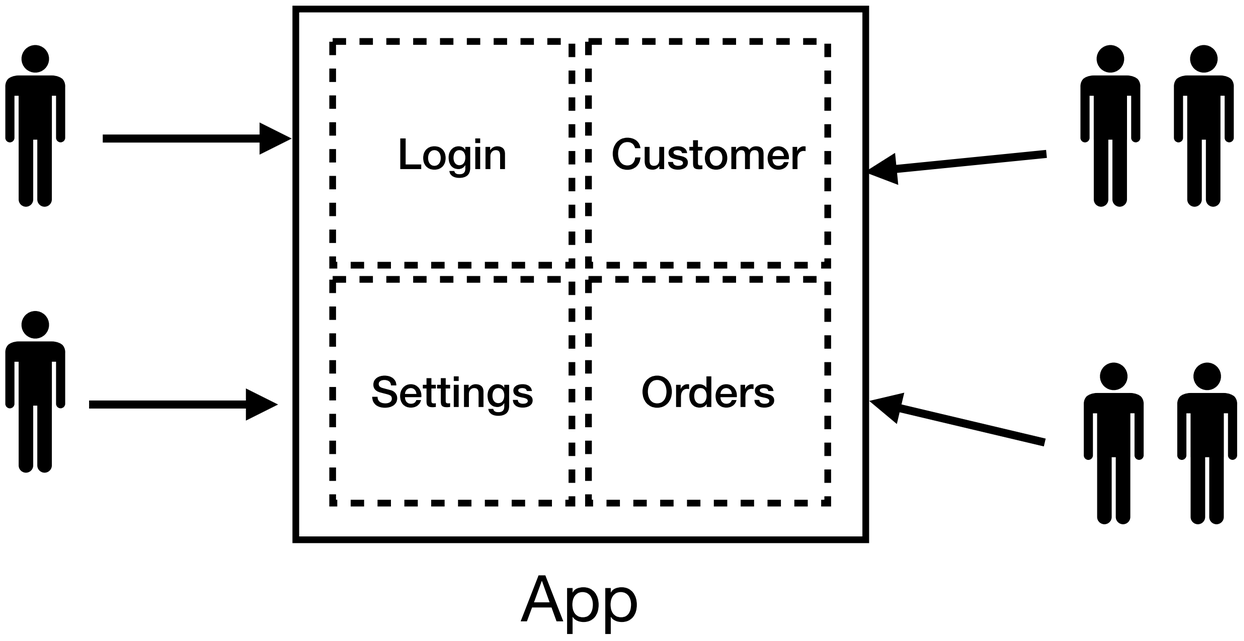
Consider a DSL

```
plugins {  id 'com.squareup.android.wiring'}

square {
  android {
   namespace 'com.squareup.buyerscreenrendersupport.impl.bran.wiring'
   unitTests {
      enabled true
      enableMocking true
    }
  }
  checks { dependencyAnalysis true }
}

dependencies {
  api project(':common:buyer-screen-render-support:public')
  implementation project(':common:x2comms:public')

  testImplementation deps.testparameterinjector
  testImplementation deps.workflow1.testing
}
```

Pit of Success

Guard Rails

- **Why**
  - Adding early saves you time
  - May catch things you wouldn’t have caught manually
  - Promotes good hygiene early
- **Scale**
  - Small



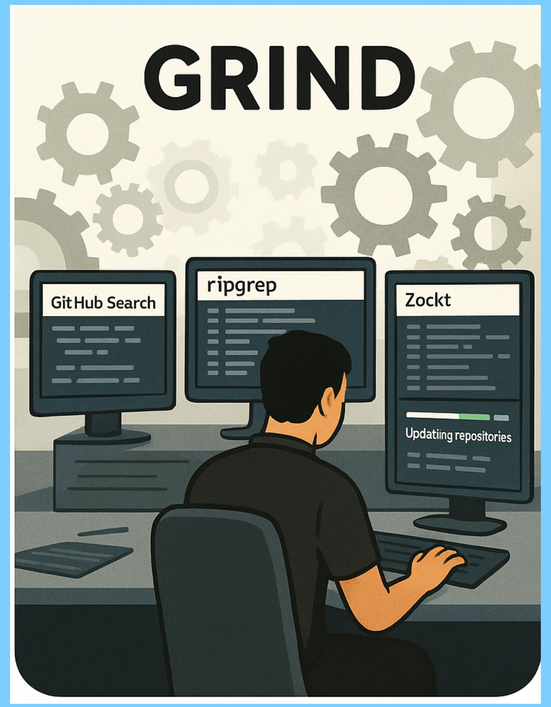
Approaching Refactoring

Tech Debt

We have it too

- Mortar/Flow to Workflow
- Multiple UI Design Systems
- View/Mosaic to Compose
- Async/Multithreading Frameworks (Rx, Coroutiunes, Executors)
- Java to Kotlin
- …

Approaches



https://publicobject.com/2023/06/10/farm-or-grind/

Grind Approach

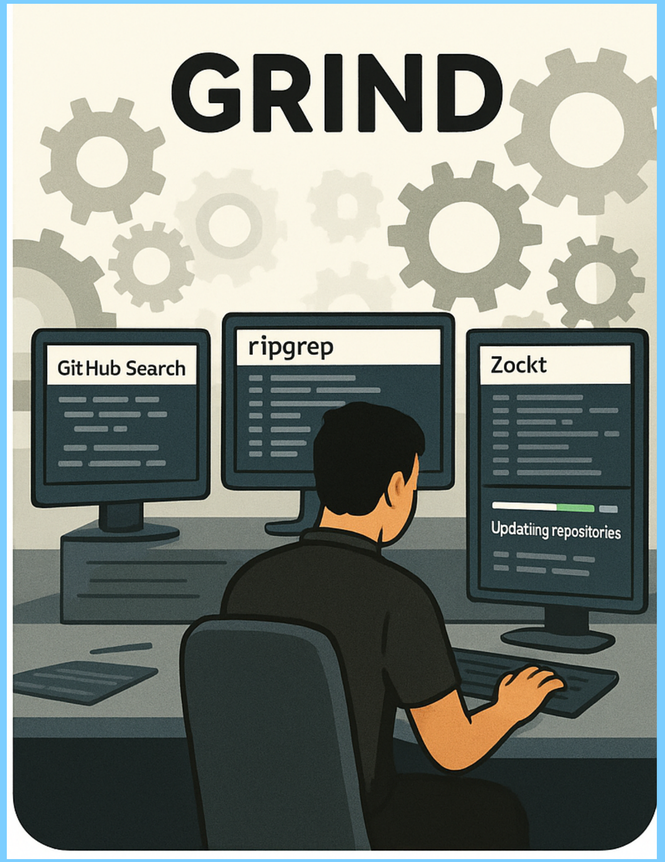
Roll up your sleeves and get scrappy

Motivations:

- Change ***is*** mechanical regex/script/AI
- Hard to get focus from teams

Recommendations:

- Use scripts / AI Agents
- Batch PRs



Farm Approach

It takes a village approach

Motivations:

- Change is not mechanical
- Requires domain expertise

Recommendations:

- Deprecate and offer the alternative
- Create a Runbook
- Dashboards or Jiras

Combination

Tiger Squad / SME Groups

Motivations:

- Very large, heterogenous refactors
- Span large API surface throughout the org
- Core Team helps keep other teams informed / aligned.

Resources

Resources

Tools to accelerate

- Workflow - https://square.github.io/workflow/
  - https://www.droidcon.com/2022/06/28/the-workflow-pattern-composed/
  - https://www.youtube.com/watch?v=cw9ZF9-ilac
- Dependency Injection
  - https://ralf-wondratschek.com/presentation/dagger-plus-anvil.html
  - https://zacsweers.github.io/metro/

Resources

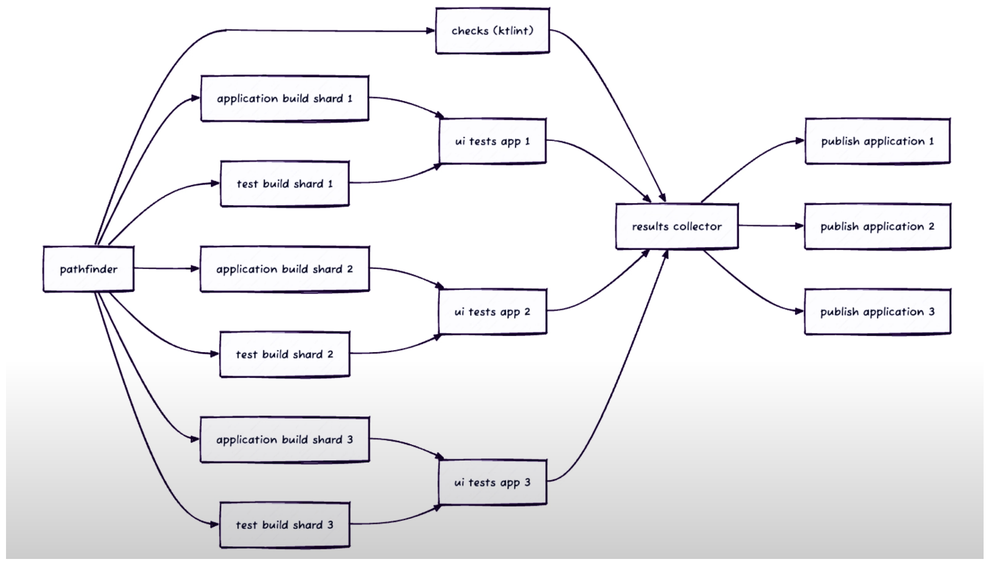
Tools to accelerate

- Module Structure - https://amzn.github.io/app-platform
- And actually, you may check that out for an opinionated launch pad on
  - Dependency Injection / Scopes
  - Module structure
  - Testing / Robots
  - Enforcement
  - Thank you Ralf (https://androiddev.social/@ralf)

Resources

Tools to accelerate

- Continuous Integration
  - 11 Apps
  - 1200 Jobs on each PR
  - 14000 UI Tests
  - 5400 Modules
- https://www.youtube.com/watch?v=rpf-V7eSdoE



Thank you