Testowalna aplikacja na Androida?

Spróbujmy z Clean Architecture.

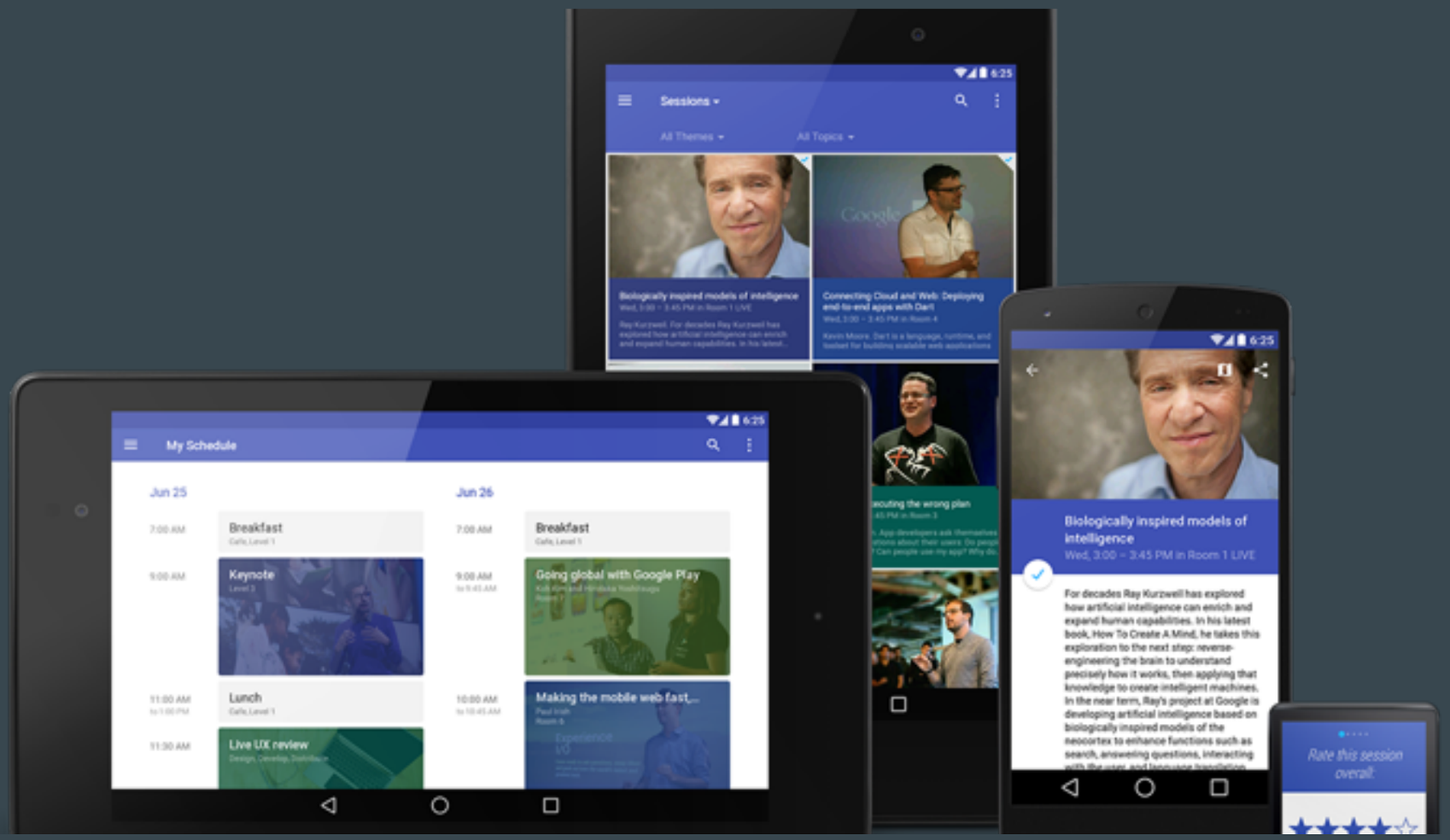
Michał Charmas

O mnie

- Developer aplikacji mobilnych na Androida
- Trener w firmie Bottega IT Solutions
- Konsultant / Freelancer

- Kwestie techniczne / optymalizacyjne - raczej łatwe i dobrze opisane

- Kwestie techniczne / optymalizacyjne - raczej łatwe i dobrze opisane
- **Jak żyć?**
  - jak nie psuć wcześniej działających ficzerów wprowadzonymi zmianami?
  - jak wykrywać takie sytuacje?
  - jak dzielić odpowiedzialność?
  - jak osiągnąć mniejszy coupling?
  - jak testować?


Sessions
All Themes
All Topics
Biologically inspired models of intelligence
Connecting Cloud and Web: Deploying end-to-end apps with Dart
My Schedule
Jun 25
Jun 26
Breakfast
Keynote
Going global with Google Play
Lunch
Making the mobile web fast,...
Live UX review
Biologically inspired models of intelligence
Wed, 3:00 – 3:45 PM in Room 1 LIVE
Rate this session overall.

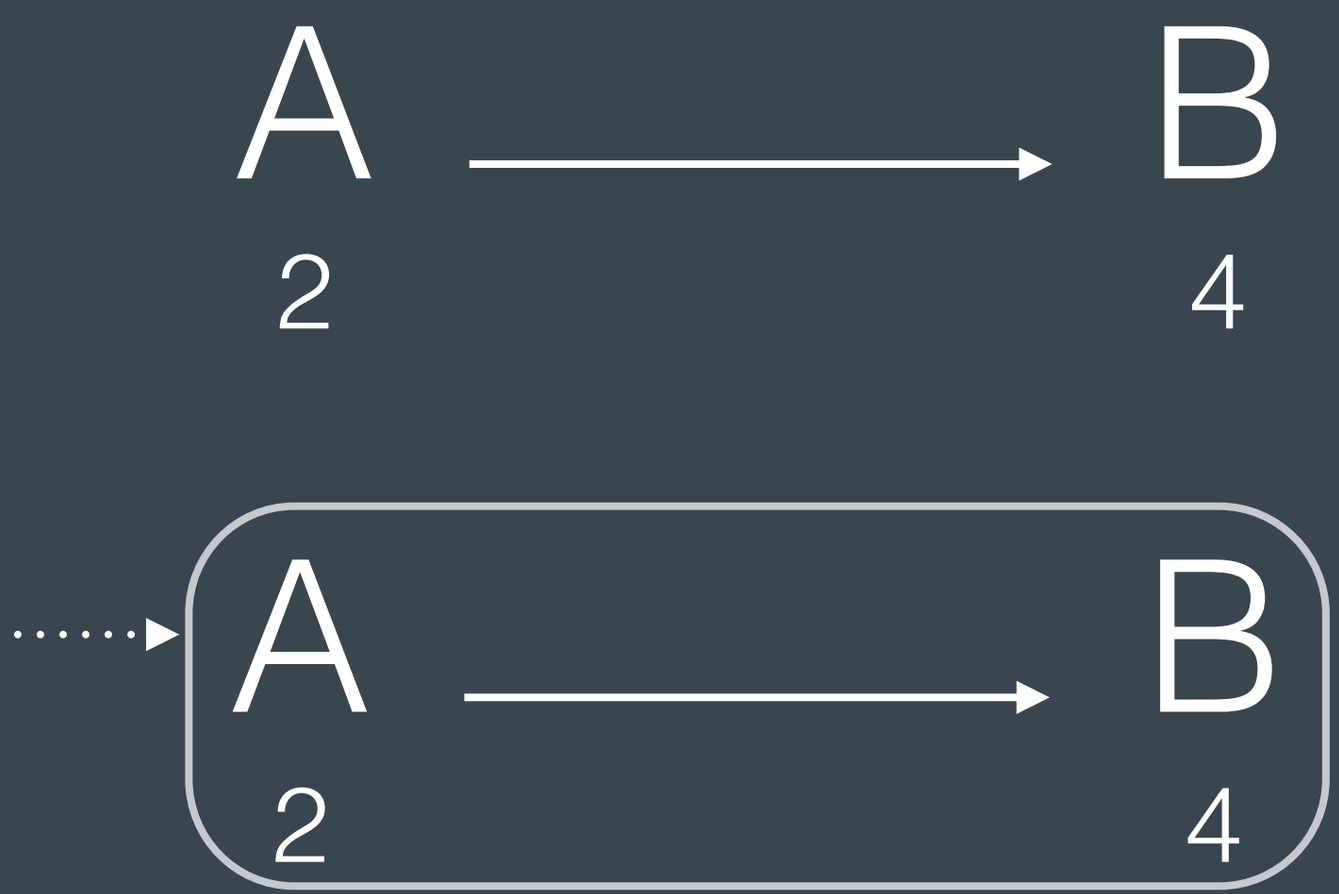
ioChed 2014

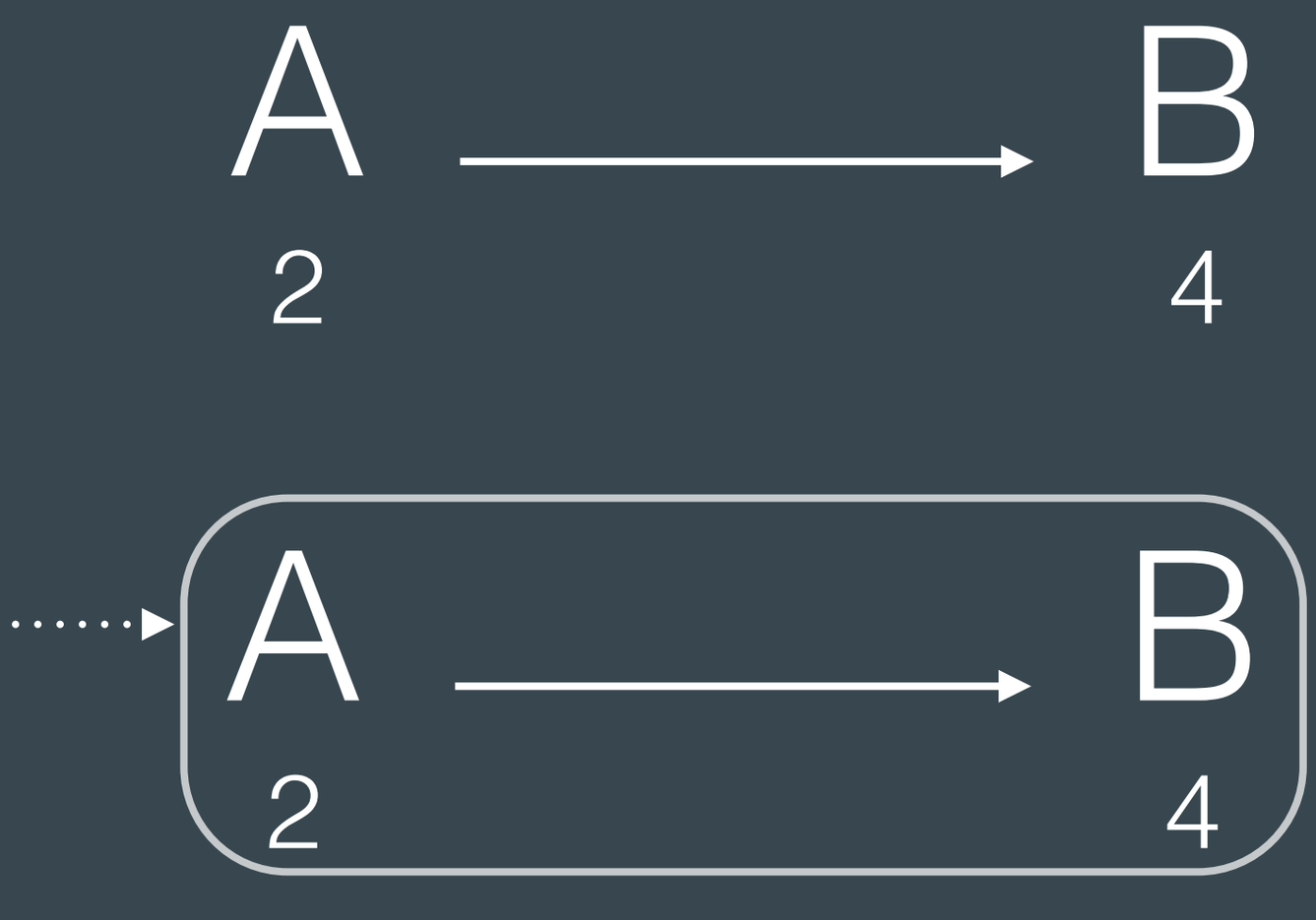
„...the other primary goal is to serve as a practical example of **best practices** for Android app design and **development**."

—Android Developers Blog

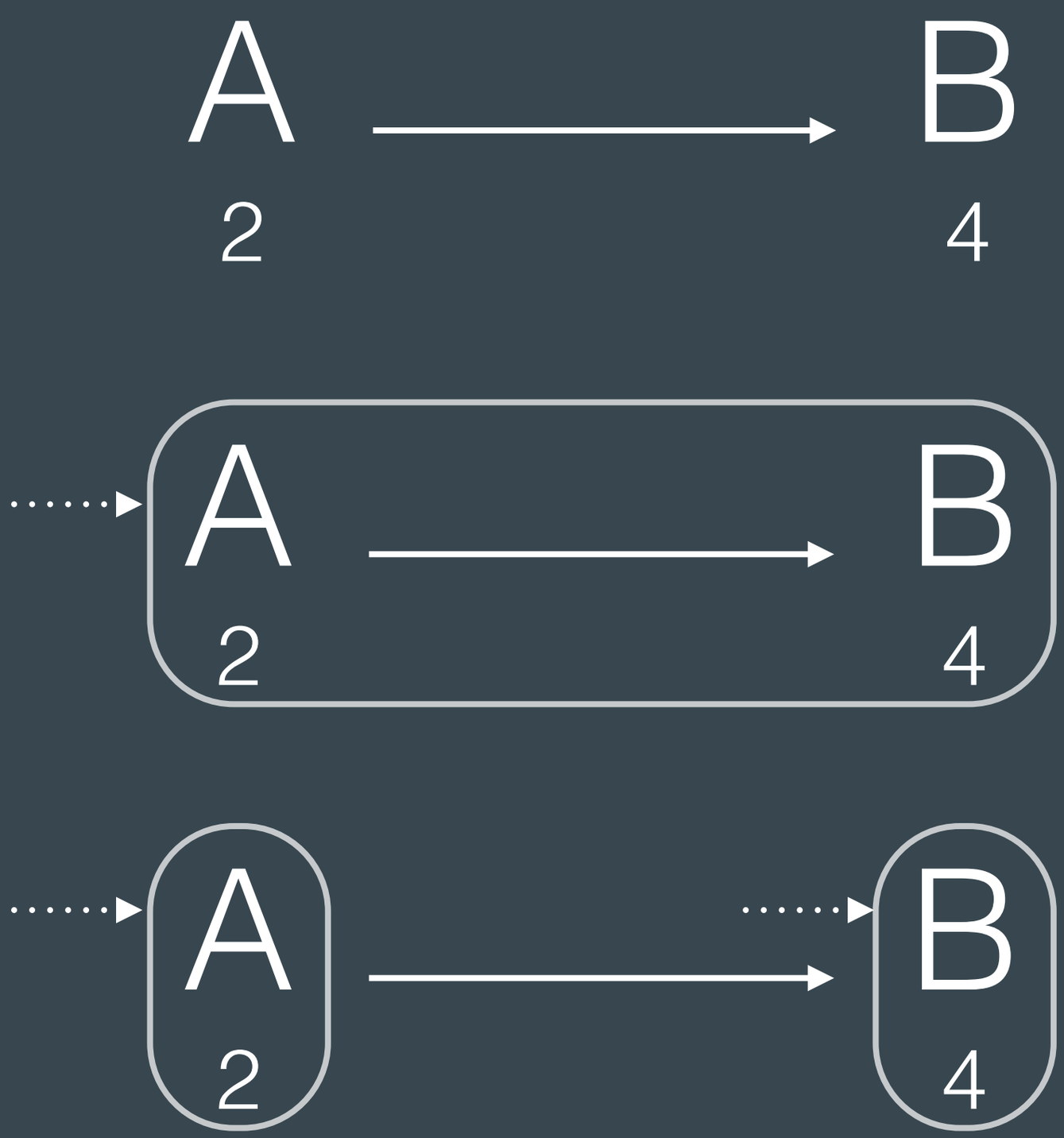
- Gdzie zasada pojedynczej odpowiedzialności?
- Logika domenowa w UI
- Logika UI pomieszana z asynchronicznym pobieraniem danych
- Callbacks everywhere
- Mapowanie kursorów na obiekty biznesowe w UI
- Activity i Fragmenty po 1000+ linii
- Całkowite uzależnienie od frameworka (import android.*)
- Testy?


A
B
2
4


A
B
2
4
A
B
2
4



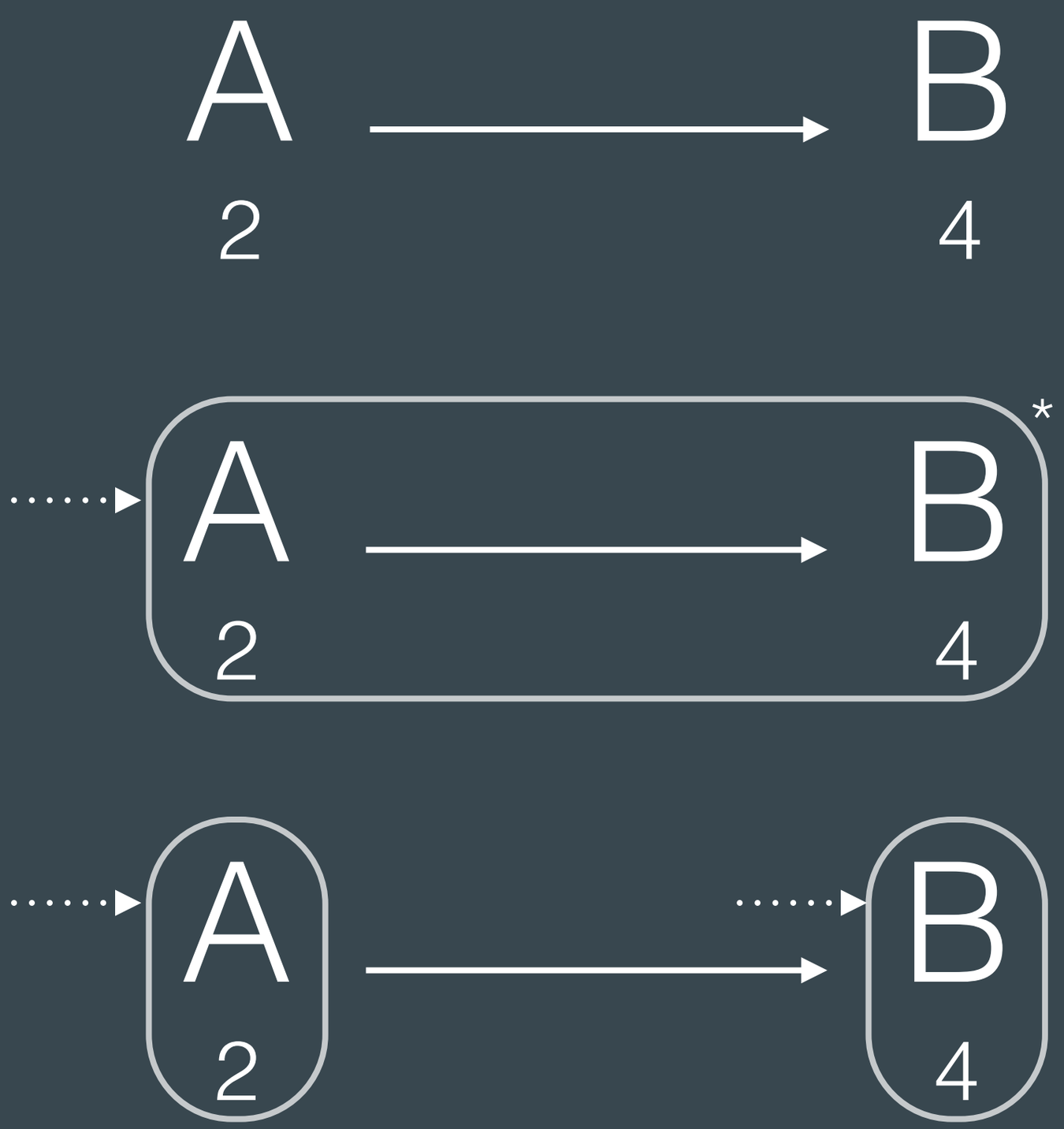
$2 * 4 = 8$



$2 * 4 = 8$

$2 * 4 = 8$

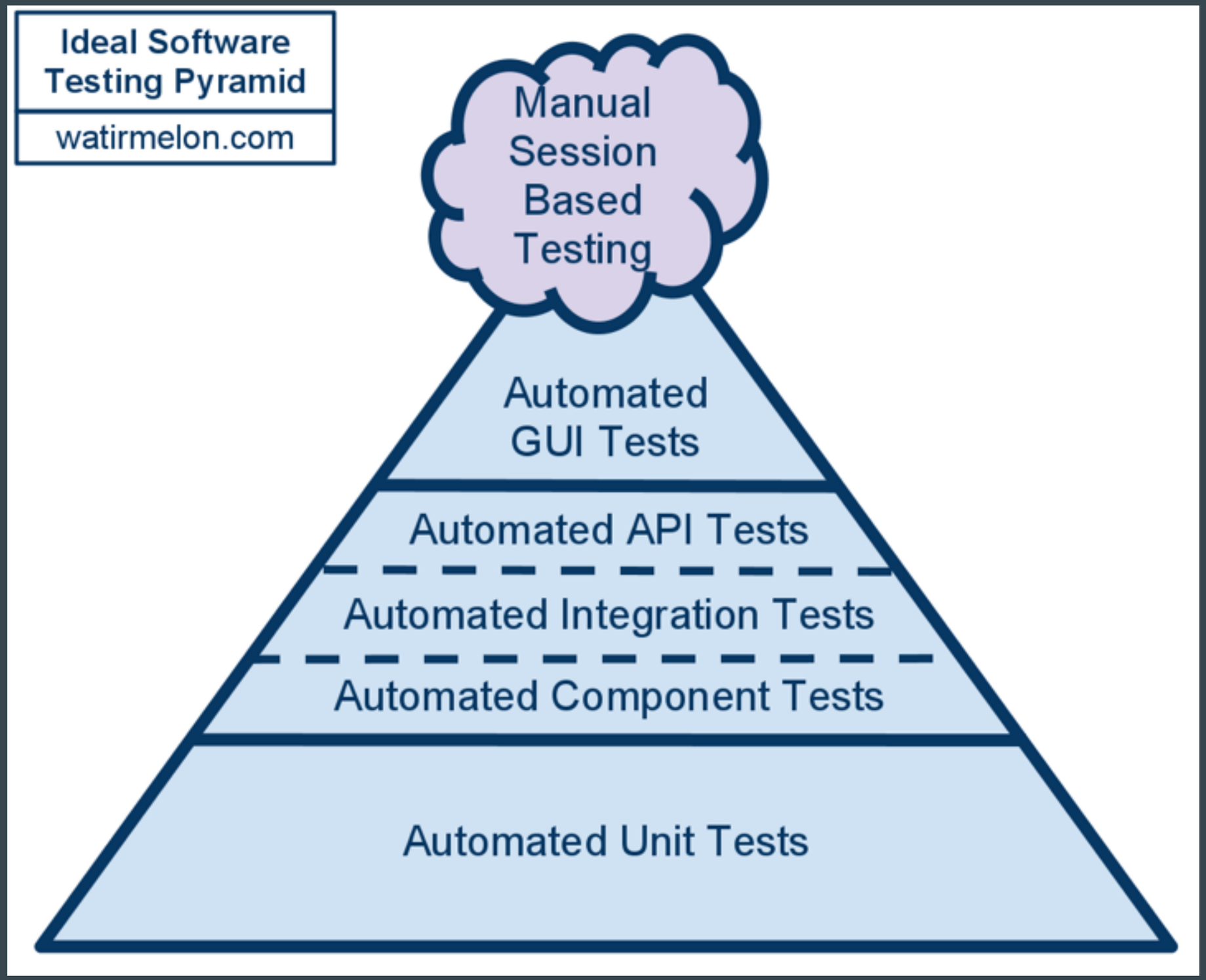
$2 + 4 = 6$

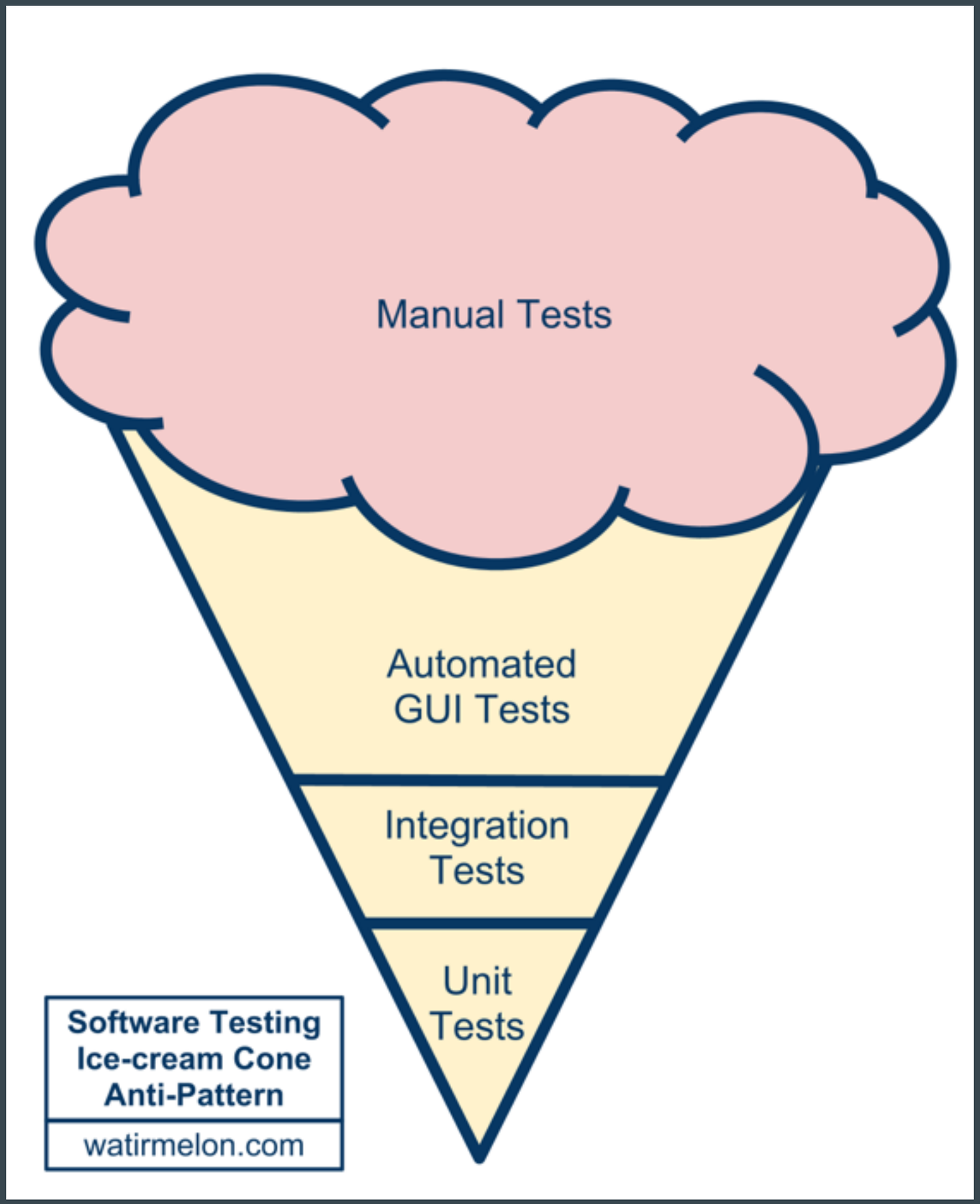


$2 * 4 = 8$

$2 + 4 = 6$

* J.B. Rainsberger - Integrated Tests Are A Scam (https://vimeo.com/80533536)

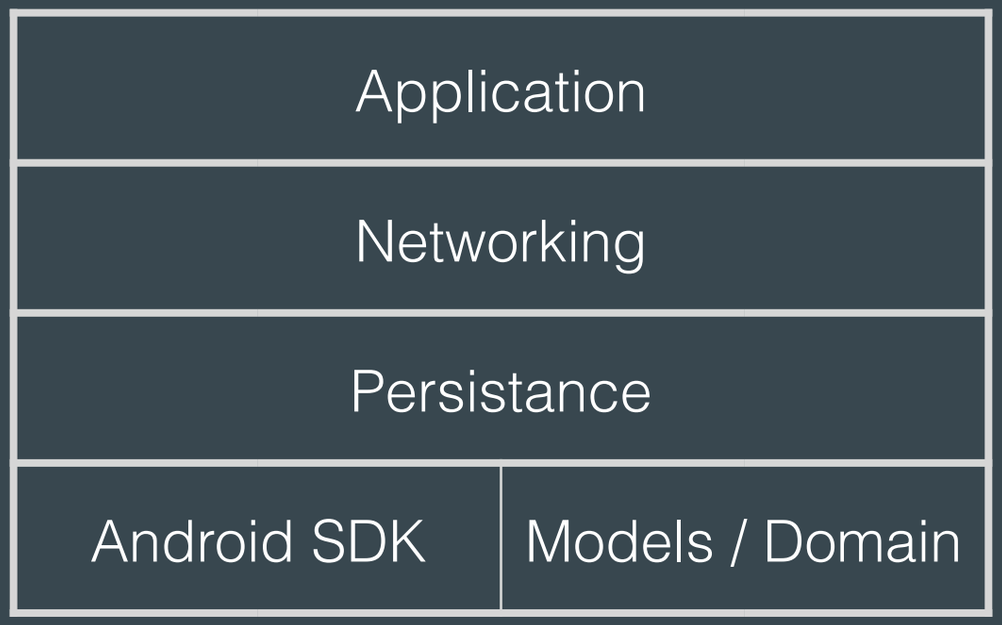

Ideal Software Testing Pyramid
watirmelon.com
Manual Session Based Testing
Automated GUI Tests
Automated API Tests
Automated Integration Tests
Automated Component Tests
Automated Unit Tests

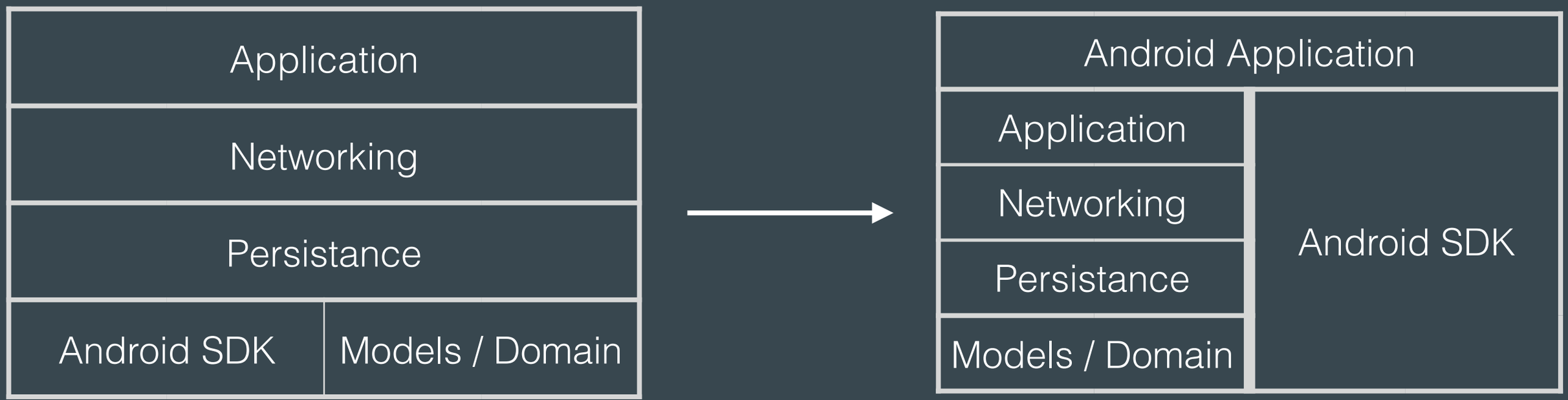

Manual Tests
Automated
GUI Tests
Integration
Tests
Unit
Tests
Software Testing
Ice-cream Cone
Anti-Pattern
watirmelon.com

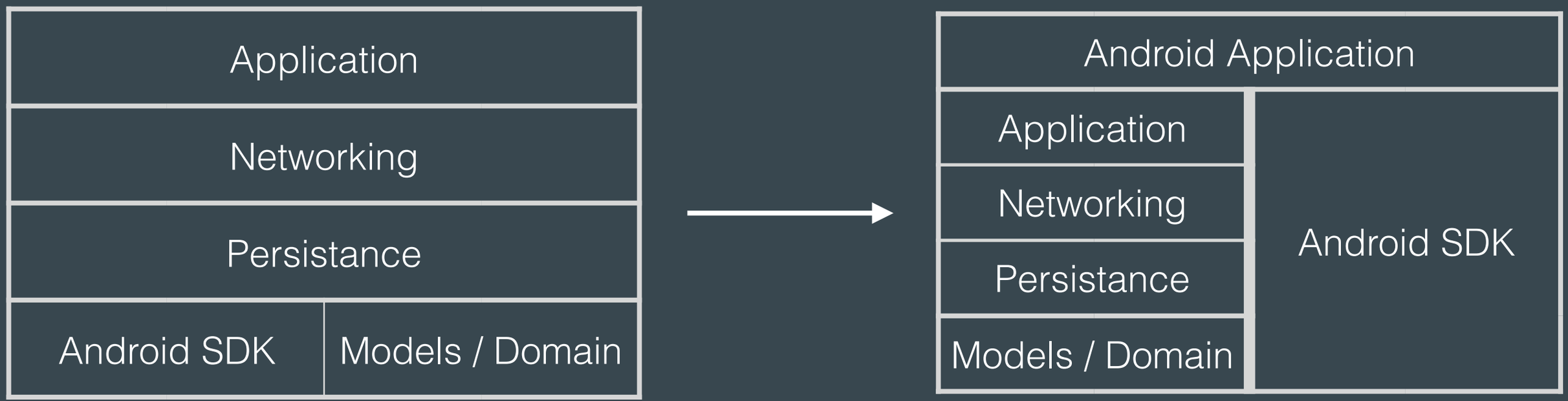

MENU

„Application components (…) are interfaces for your application to interact with the operating system; **don’t take them as a recommendation of the facilities you should architect your entire application around**.”

–Chet Haase, Developing for Android VII

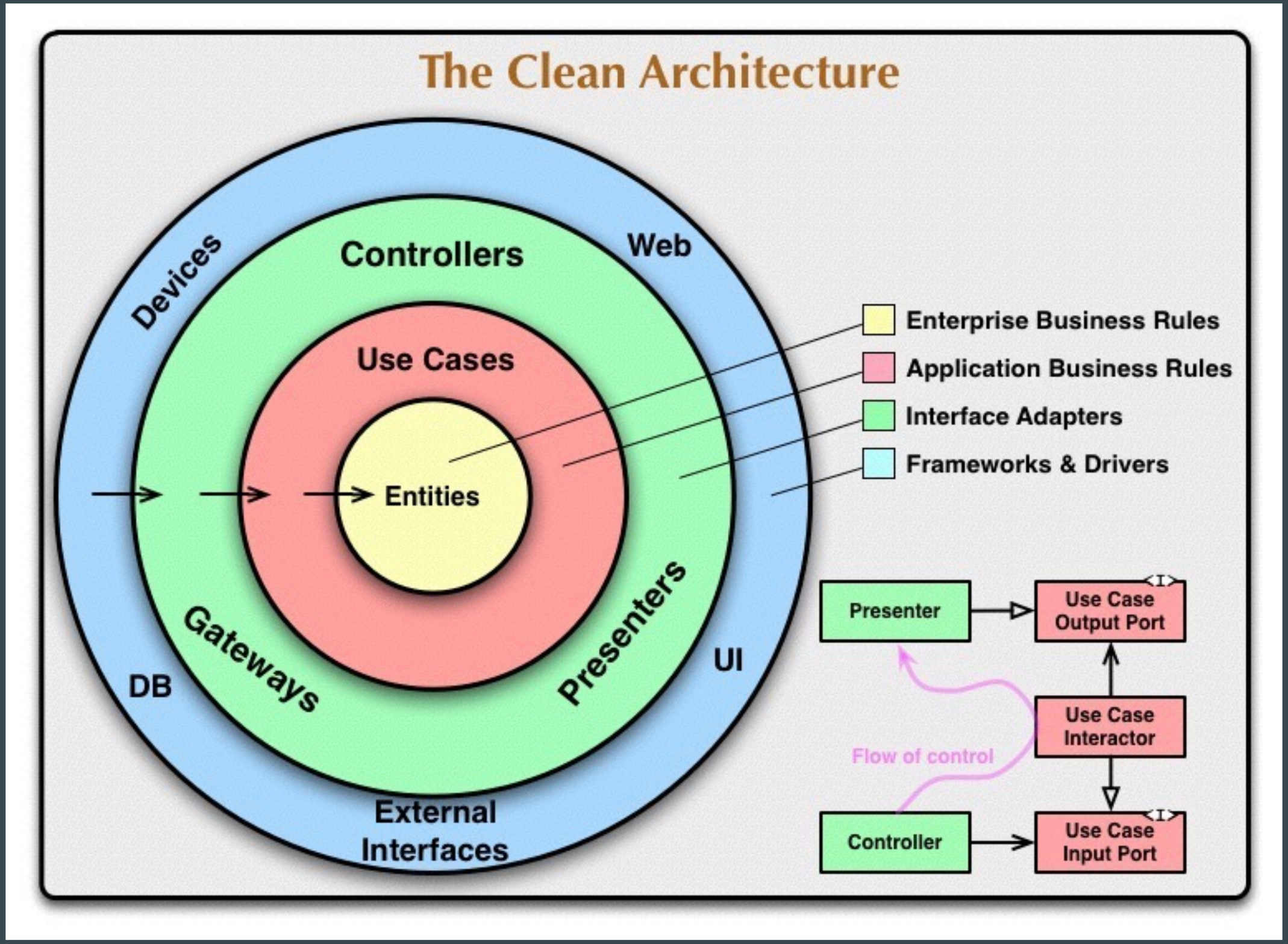

Application
Networking
Persistance
Android SDK
Models / Domain


Application
Networking
Persistance
Android SDK
Models / Domain
Android Application
Application
Networking
Persistance
Models / Domain
Android SDK



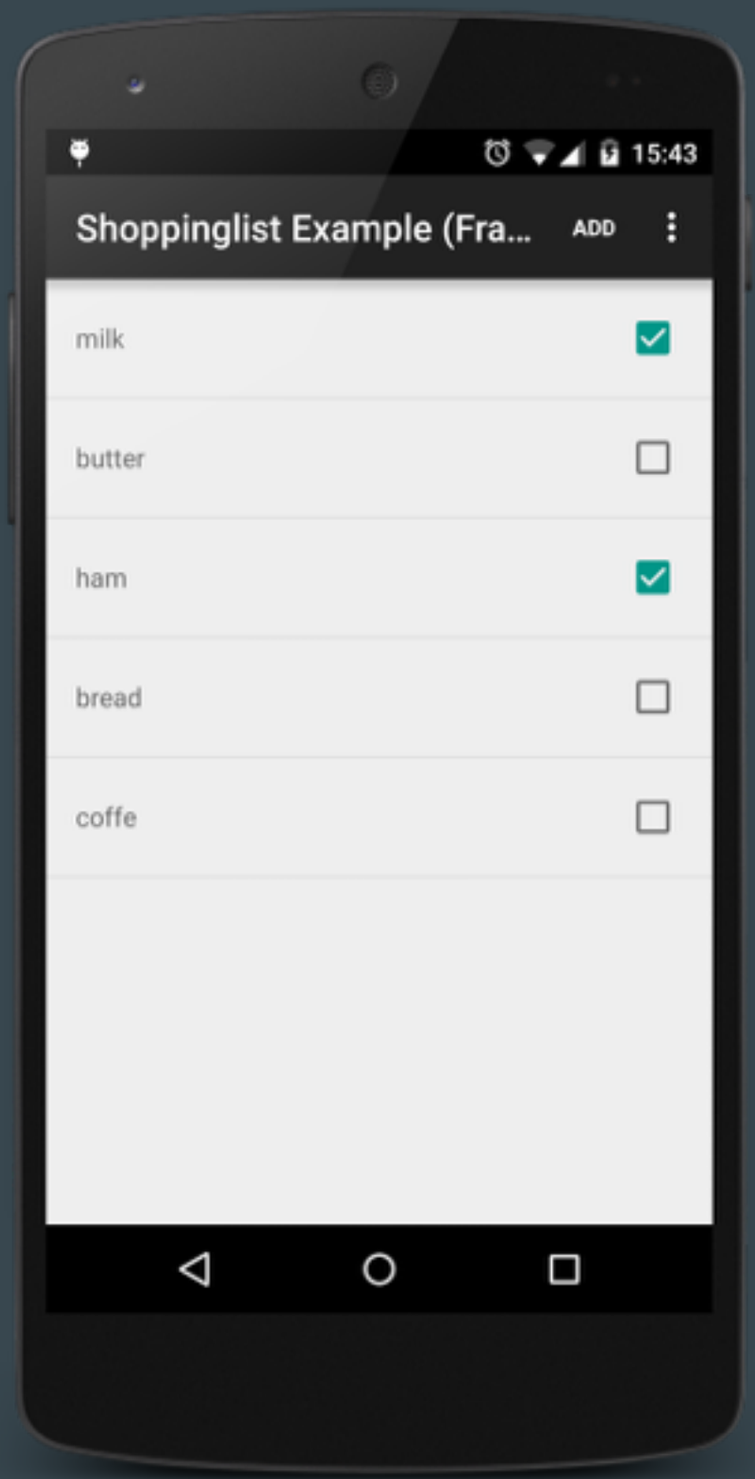

```
import android.*
```

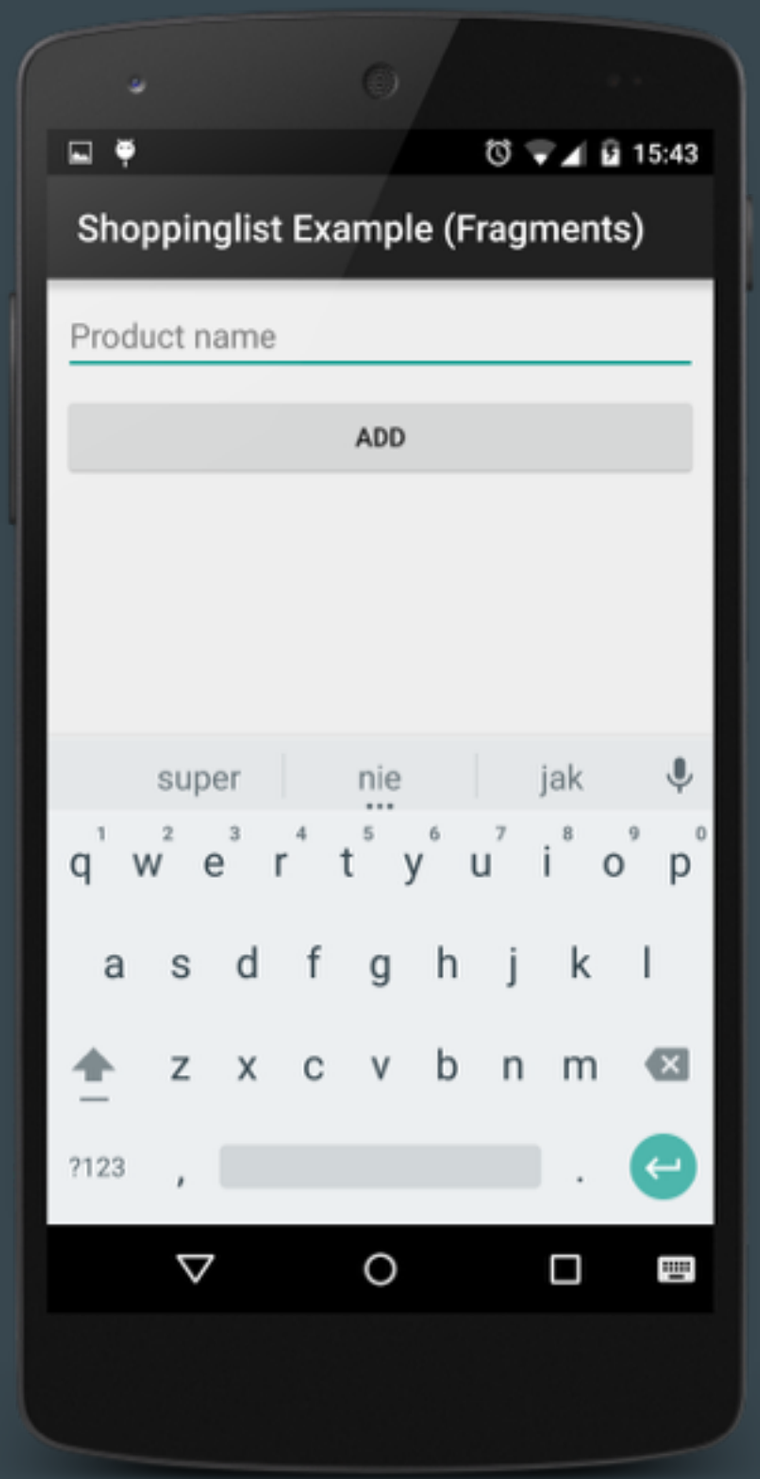
„szczegół techniczny”



Źródło: http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html

- Niezależna od frameworka - zasda zależności.
- Niezależna od interfejsu użytkownika.
- Niezależna od bazy danych.
- Testowalna - w oderwaniu od frameworka / bazy danych / serwera.


15:43
Shoppinglist Example (Fra...
ADD
milk
butter
ham
bread
coffe


15:43
Shoppinglist Example (Fragments)
Product name
ADD
super
nie
jak
?123

```java
public class Product {
  private final long id;
  private final String name;
  private boolean isBought;

  public Product(long id, String name, boolean isBought){
    this.id = id;
    this.name = name;
    this.isBought = isBought;
  }

  public void markBought() {
    this.isBought = true;
  }

  public void markNotBought() {
    this.isBought = false;
  }

  //getters
}
```

```java
@Test public void testShouldBeBoughtWhenMarkedAsBought() throws Exception {
  Product product = new Product(0, "sample name", false);
  product.markBought();
  assertEquals(true, product.isBought());
}
```

```java
public class ProductList implements Iterable<Product> {

  public Product addProduct(String productName) {
  }

  public int removeBoughtProducts() {
  }

  public Product getProduct(long id) {
  }

  public int size() {
  }

  @Override public Iterator<Product> iterator() {
  }
}
```

```java
public class ProductListTest {
  private static final String SAMPLE_PRODUCT_NAME = "sample product";

  @Test public void testShouldCreateProduct() throws Exception {
    ProductList products = new ProductList(null);
    Product product = products.addProduct(SAMPLE_PRODUCT_NAME);
    assertNotNull(product);
  }

  @Test public void testShouldCreateEmptyProductList() throws Exception {
  }

  @Test public void testShouldRemoveOnlyBoughtProducts() throws Exception {
  }

  @Test public void testShouldSetLowestAvailableId() throws Exception {
  }
}
```

„A good architecture emphasizes the use-cases and decouples them from peripheral concerns."

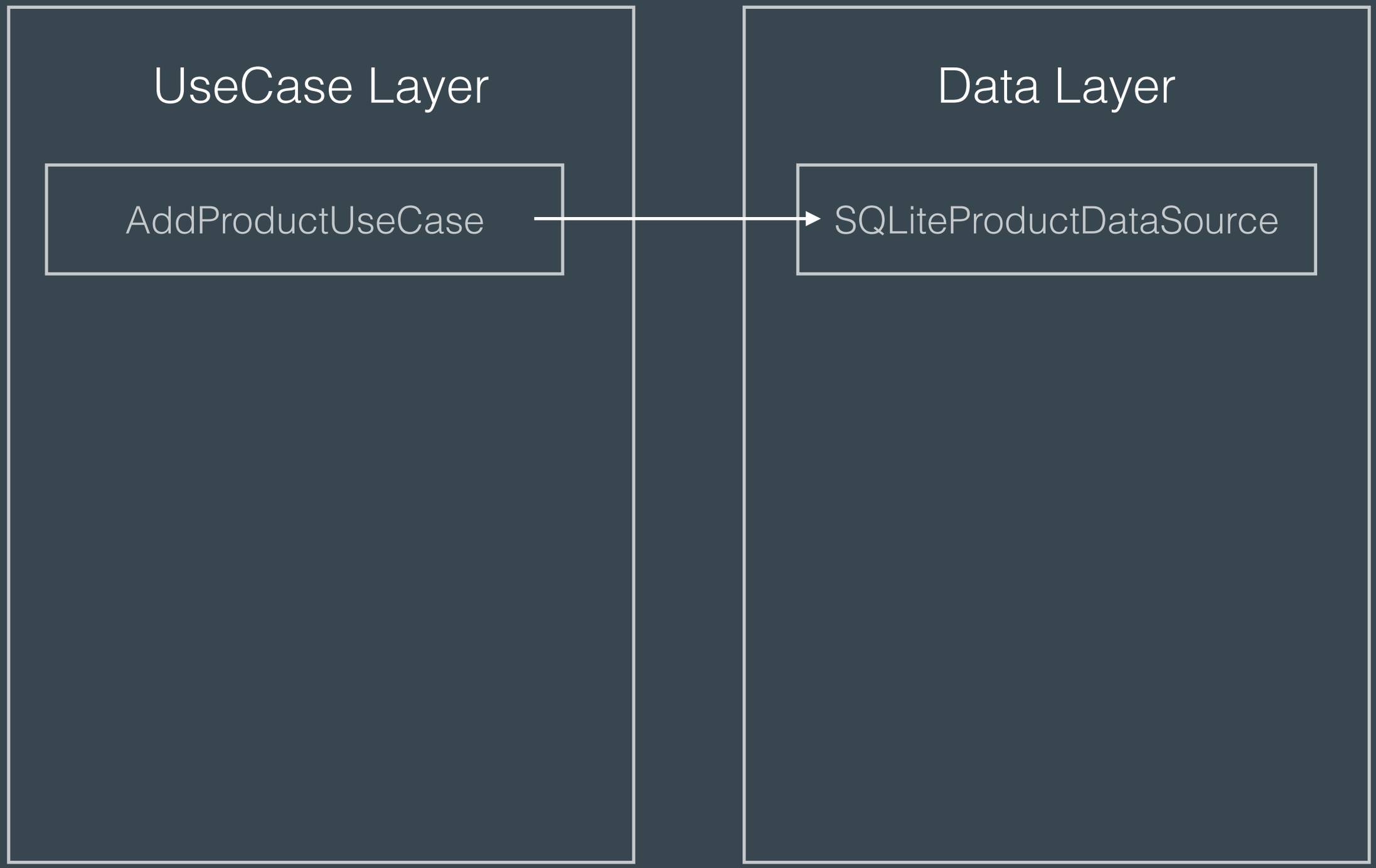
—Robert C. Martin

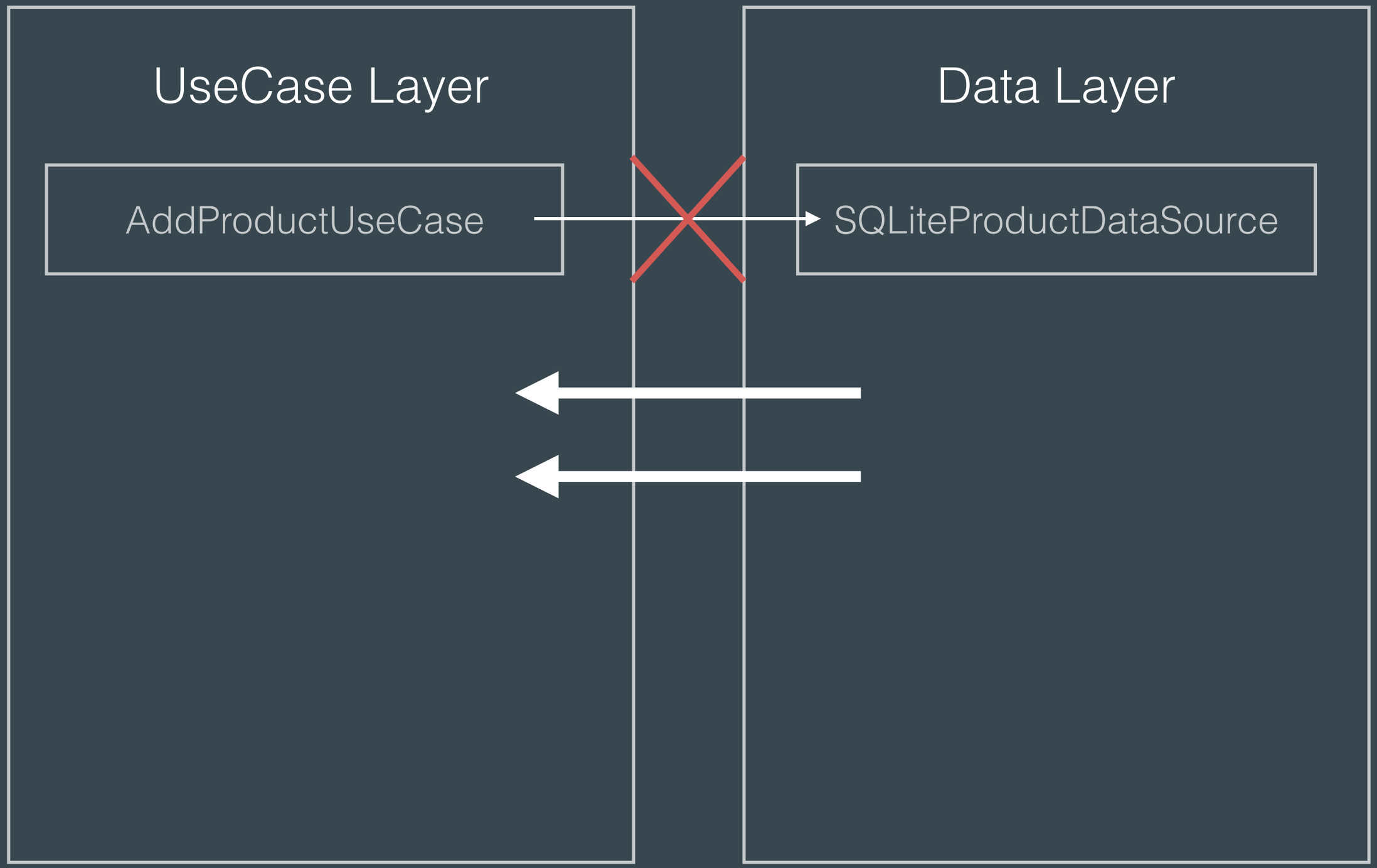
- UseCases:
  - AddProduct
  - ListProducts
  - ChangeProductBoughtStatus
  - RemoveAllBoughtProducts

- UseCases:
  - AddProduct
  - ListProducts
  - ChangeProductBoughtStatus
  - RemoveAllBoughtProducts

```
public interface UseCase<Result, Argument> {
  Result execute(Argument arg) throws Exception;
}

public interface UseCaseArgumentless<Result> {
  Result execute() throws Exception;
}
```

```java
public class AddProductUseCaseTest {
  private AddProductUseCase useCase;

  @Before public void setUp() throws Exception {
    useCase = new AddProductUseCase();
  }

  @Test public void testShouldAddProduct() throws Exception {
    useCase.execute("Sample product");

    //TODO: verify saved
  }
}
```

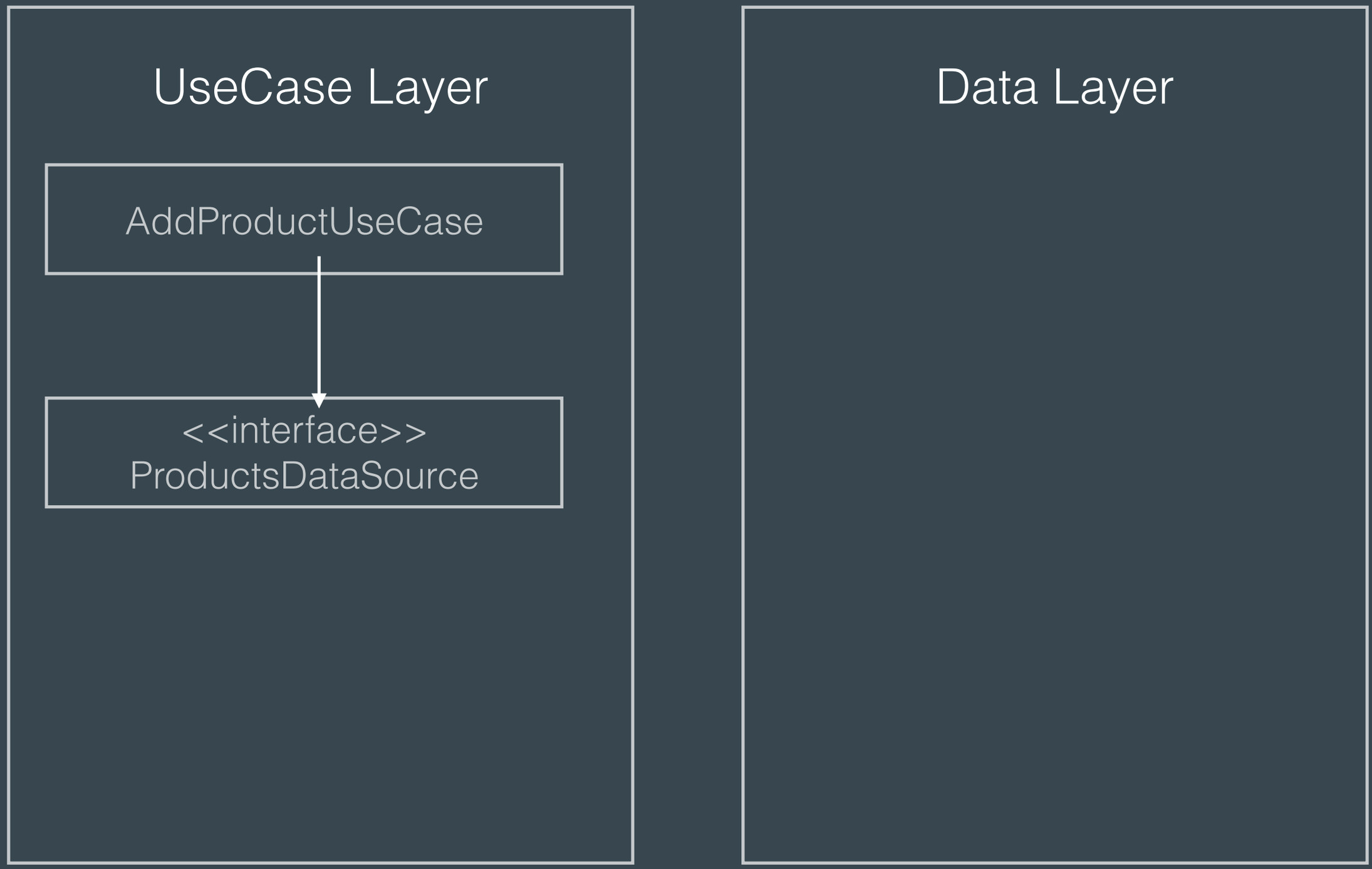

UseCase Layer
AddProductUseCase
Data Layer
SQLiteProductDataSource

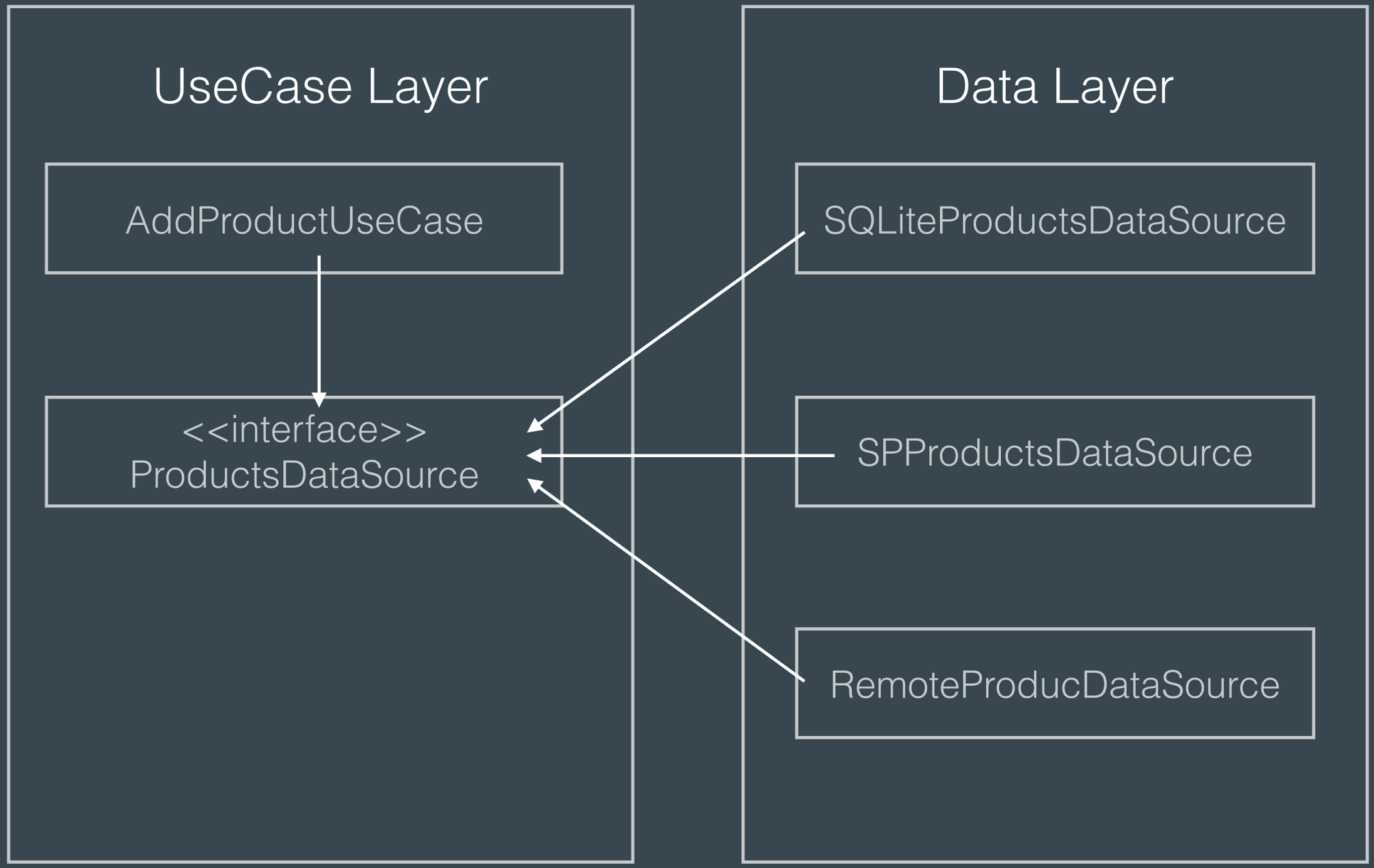

UseCase Layer
Data Layer
AddProductUseCase
SQLiteProductDataSource

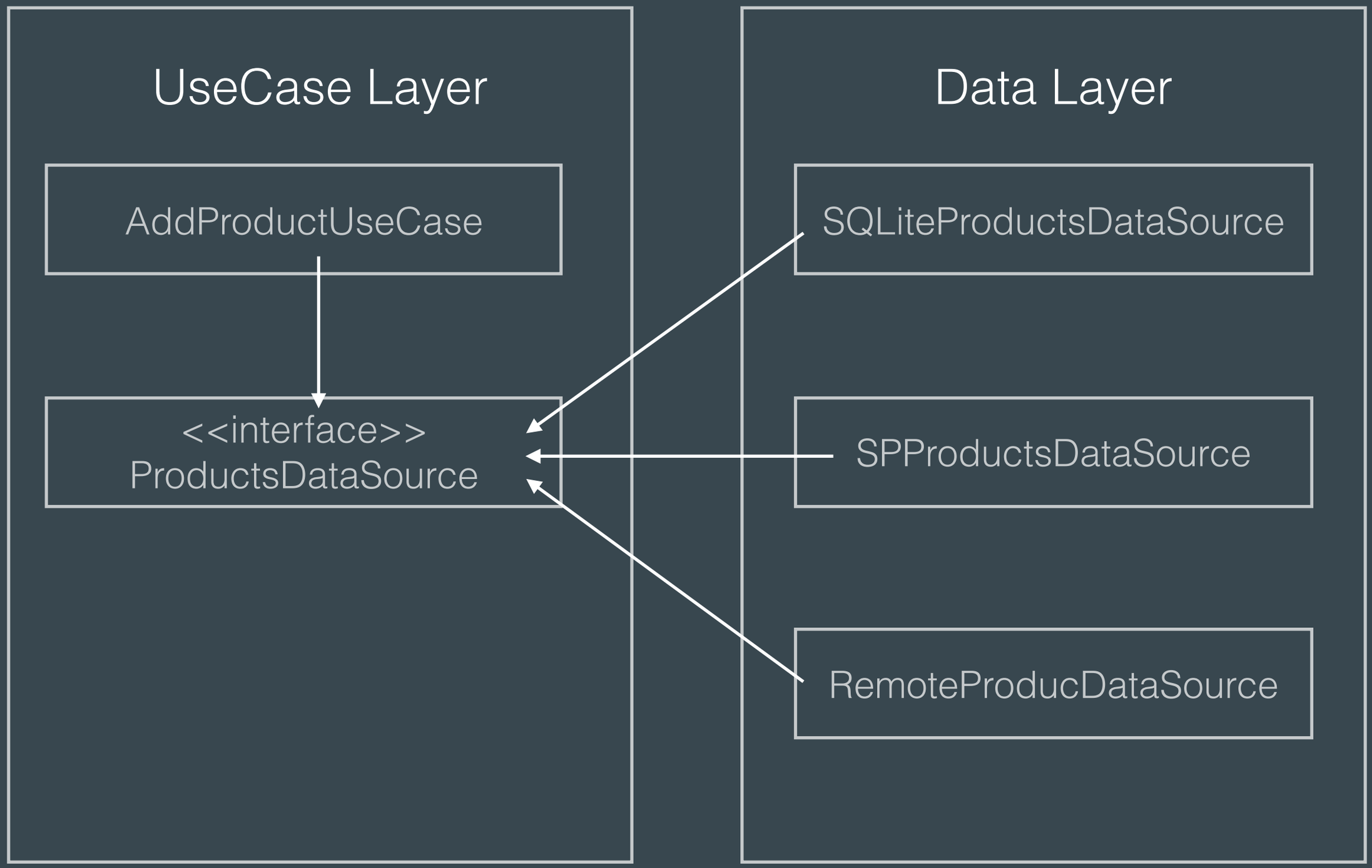
UseCase Layer

AddProductUseCase

Data Layer


UseCase Layer
AddProductUseCase
<<interface>>
ProductsDataSource
Data Layer


UseCase Layer
AddProductUseCase
<<interface>>
ProductsDataSource
Data Layer
SQLiteProductsDataSource
SPProductsDataSource
RemoteProducDataSource



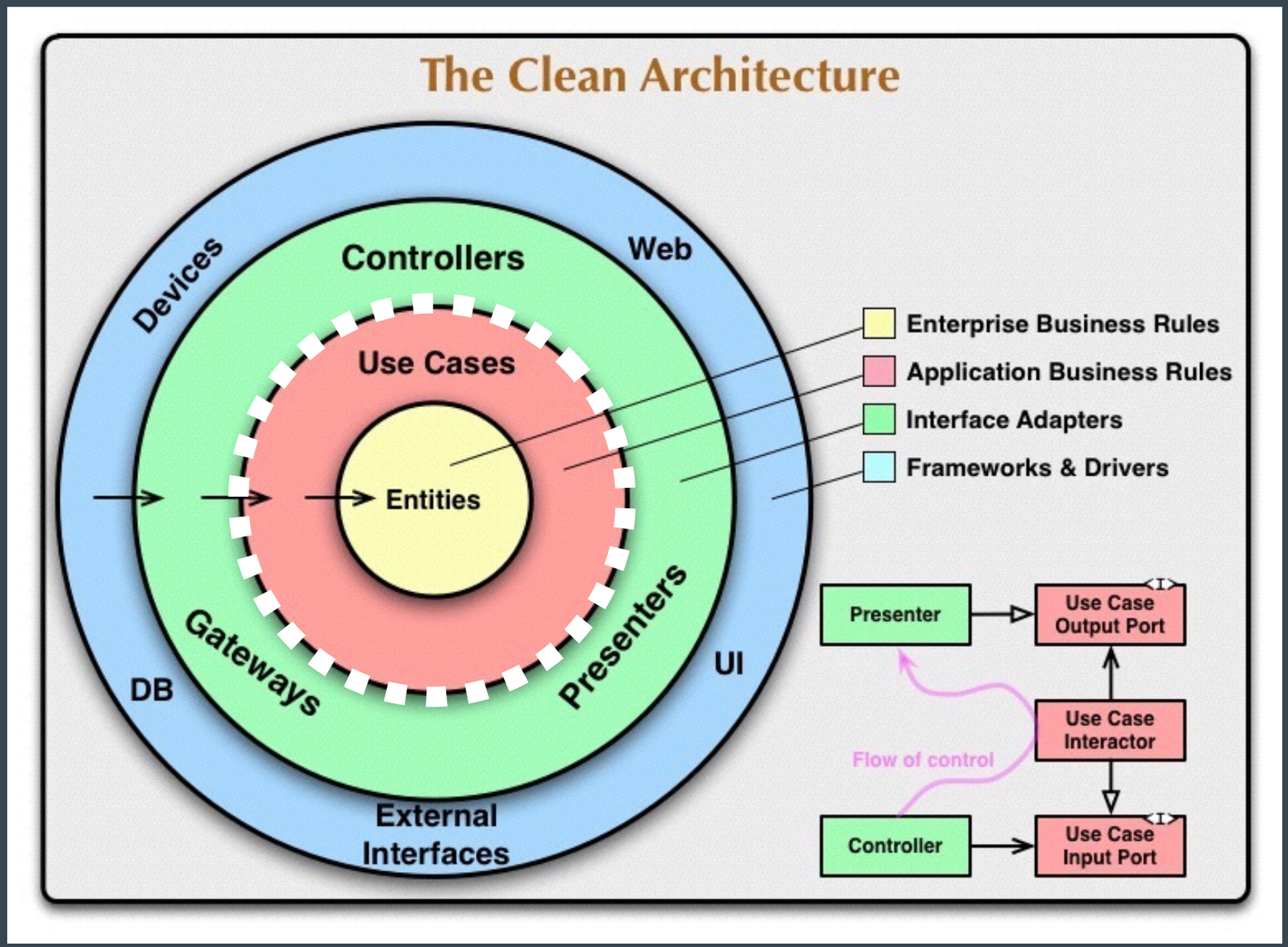
Dependency Inversion Principle

```
public interface ProductsDataSource {

  ProductList getProductList();

  void saveProductList(ProductList products);
}
```

```java
public interface ProductsDataSource {

  ProductList getProductList();

  void saveProductList(ProductList products);
}
```

```java
public class AddProductUseCaseTest {

  // setup

  @Test public void testShouldAddProduct() throws Exception {
    when(productsDataSourceMock.getProductList()).thenReturn(mockProducts);

    useCase.execute(PRODUCT_NAME);

    verify(mockProducts, times(1)).addProduct(PRODUCT_NAME);
    verify(productsDataSourceMock).saveProductList(mockProducts);
  }
}
```

```java
public class AddProductUseCase implements UseCase<Product, String> {
  private final ProductsDataSource productsDataSource;

  @Inject
  public AddProductUseCase(ProductsDataSource productsDataSource) {
    this.productsDataSource = productsDataSource;
  }

  @Override
  public Product execute(final String productName) {
    if (productName == null || productName.trim().isEmpty()) {
      throw new ValidationException("Product name cannot be empty");
    }
    ProductList productList = productsDataSource.getProductList();
    Product product = productList.addProduct(productName);
    productsDataSource.saveProductList(productList);
    return product;
  }
}
```



- całkowicie niezależne od frameworka
- pure Java
- może zostać wyciągnięte do oddzielnego modułu - czysto javowego

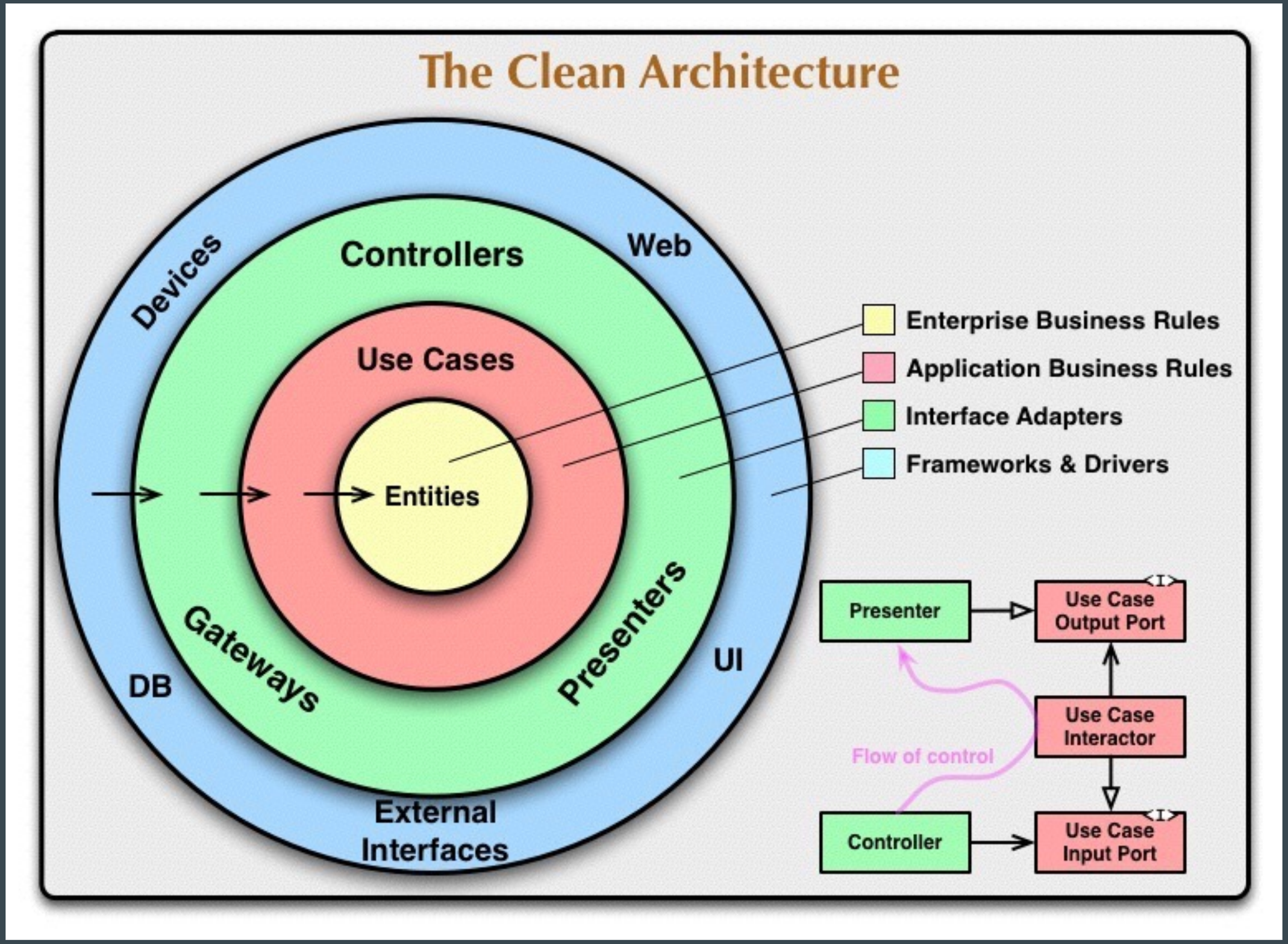
Persistence / Data

http://upload.wikimedia.org/wikipedia/commons/1/10/20090529_Great_Wall_8185.jpg


Entities
UseCases
DataSource
SharedPreferencesDataSource

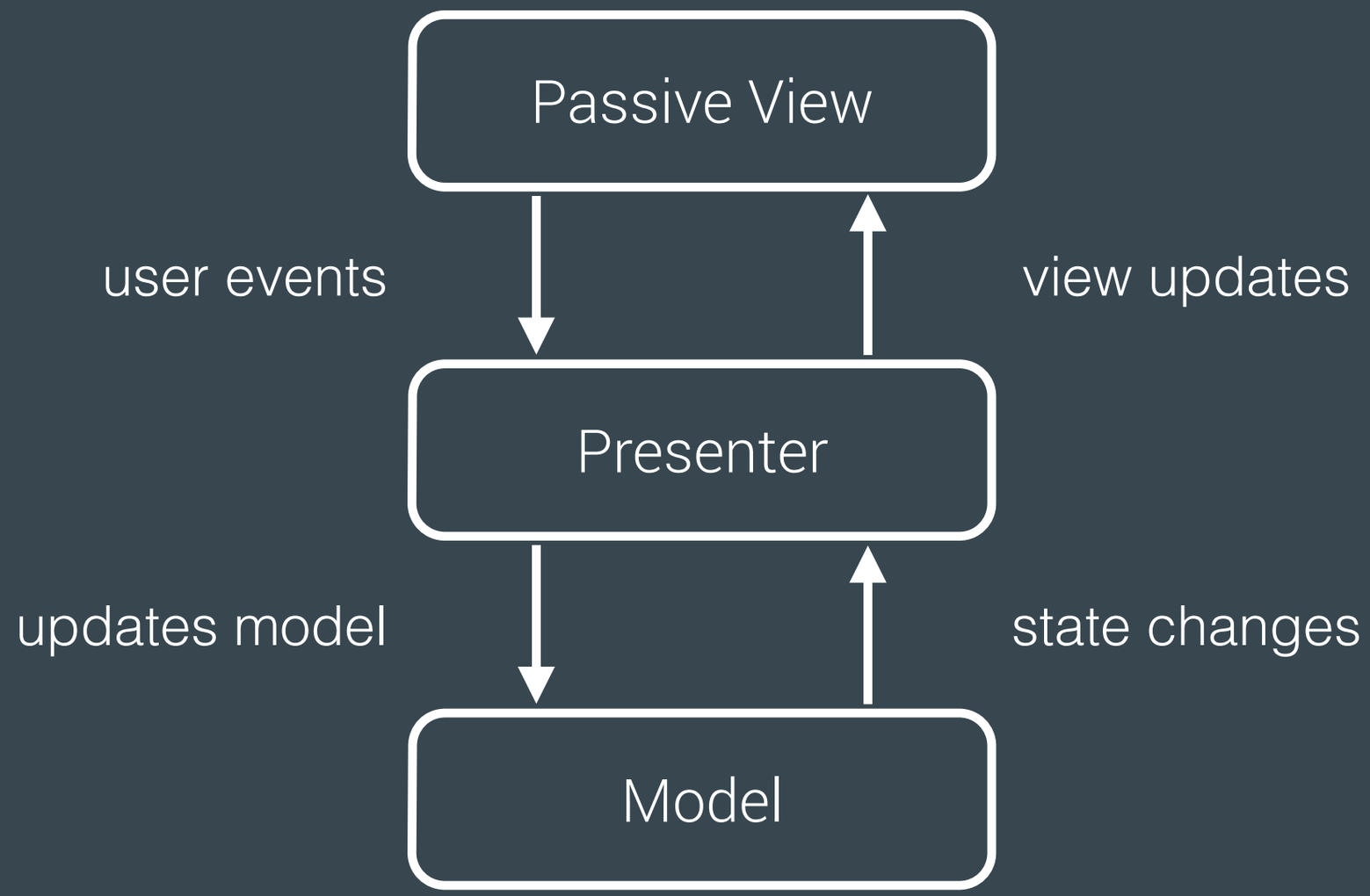
```java
public class SharedPreferencesProductsDataSource implements ProductsDataSource {
  private static final String SP_PRODUCT_ENTITIES = "PRODUCT_ENTITIES";
  private final SharedPreferences sharedPreferences;
  private final ProductListJsonMapper productListJsonMapper;

  @Inject
  public SharedPreferencesProductsDataSource(SharedPreferences sharedPreferences,
      ProductListJsonMapper productListJsonMapper) {
    this.sharedPreferences = sharedPreferences;
    this.productListJsonMapper = productListJsonMapper;
  }

  @Override public ProductList getProductList() {
    // use SP to get product list
  }

  @Override public void saveProductList(ProductList products) {
    // use SP to store product list mapping with JSON
  }
```

UI

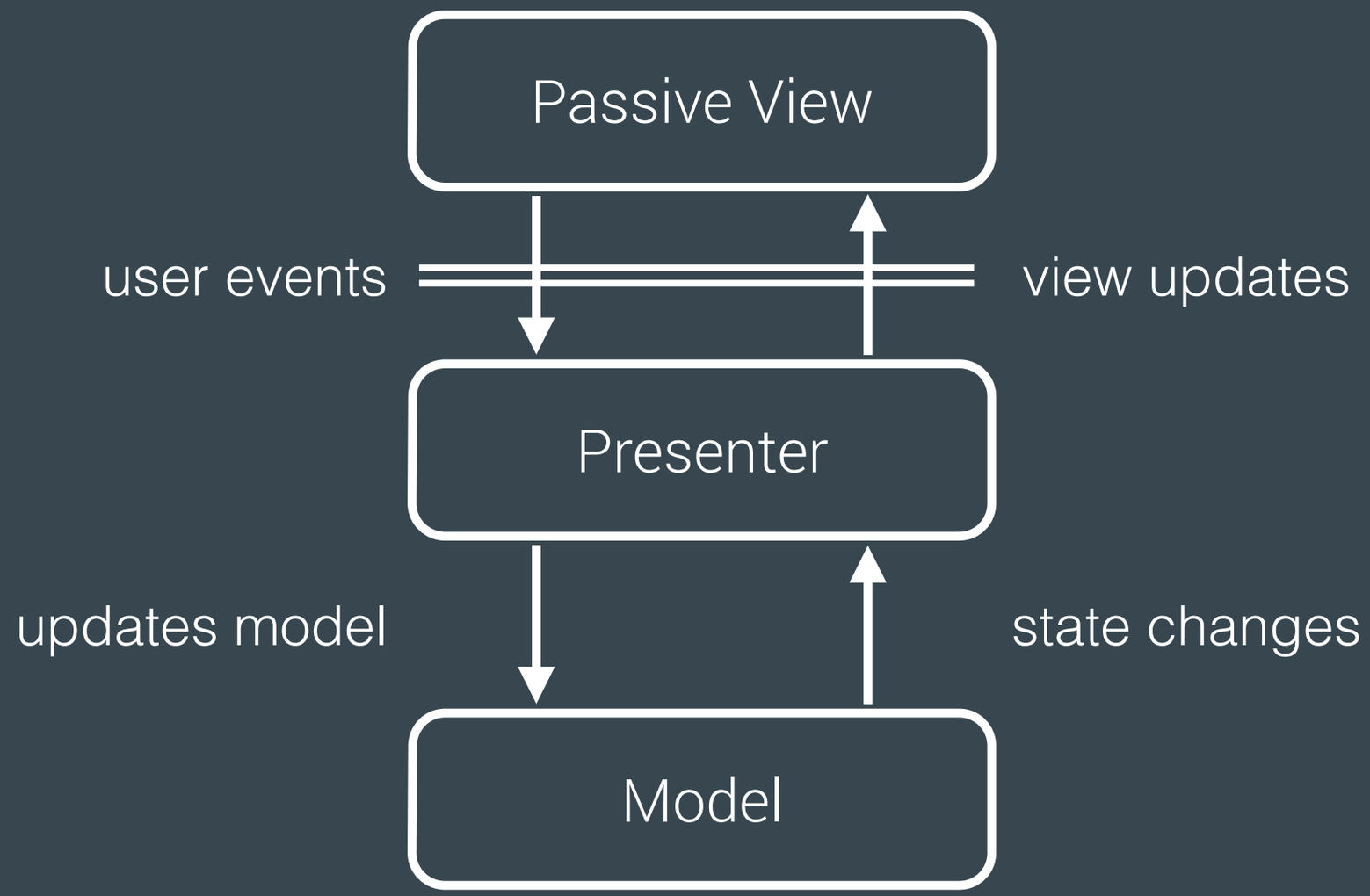


Źródło: http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html

Model View Presenter

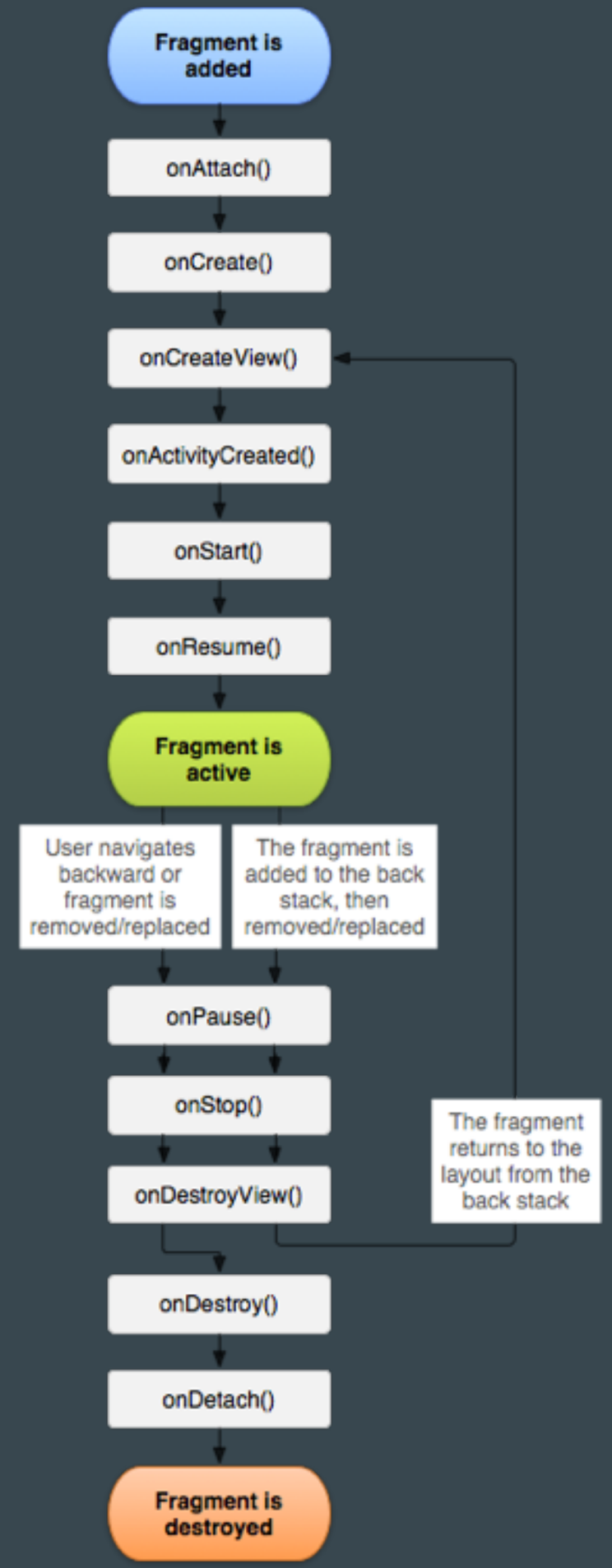


Model View Presenter



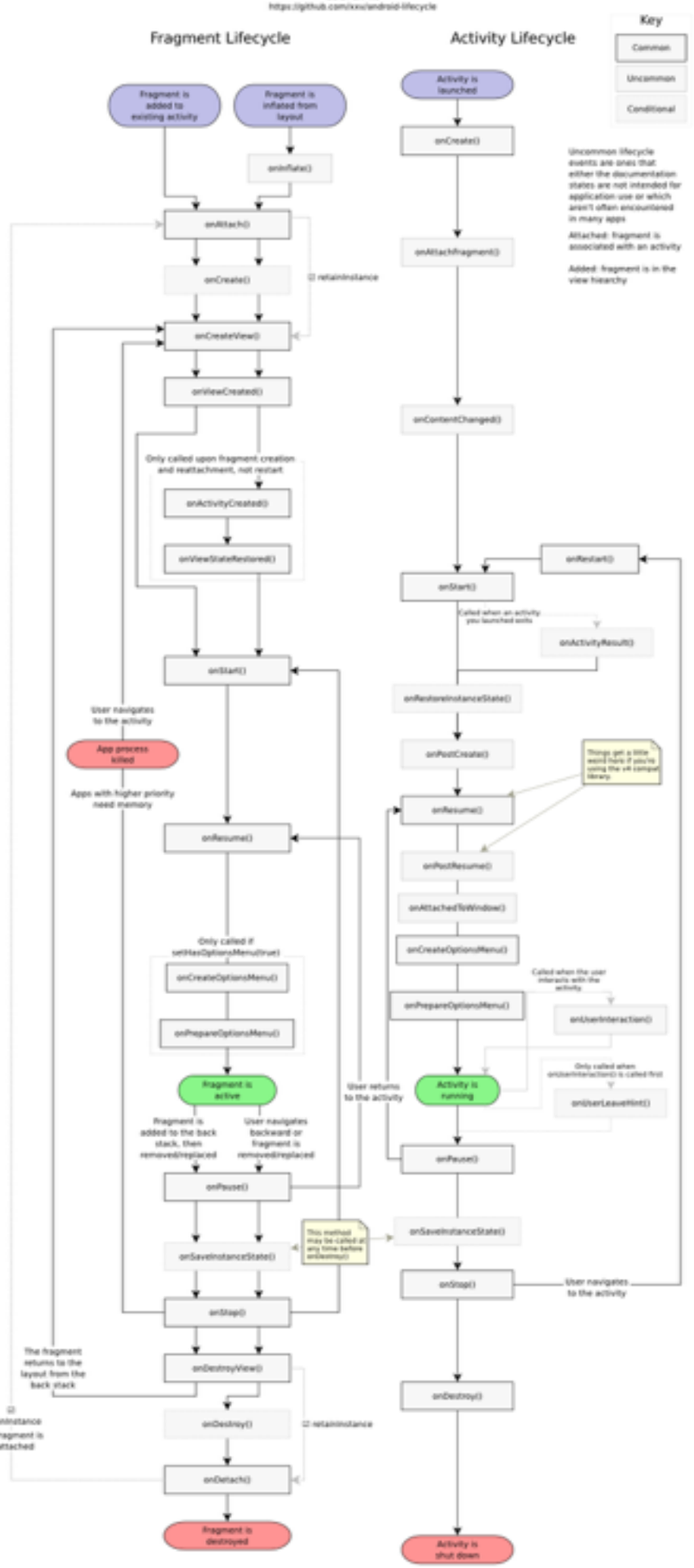

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

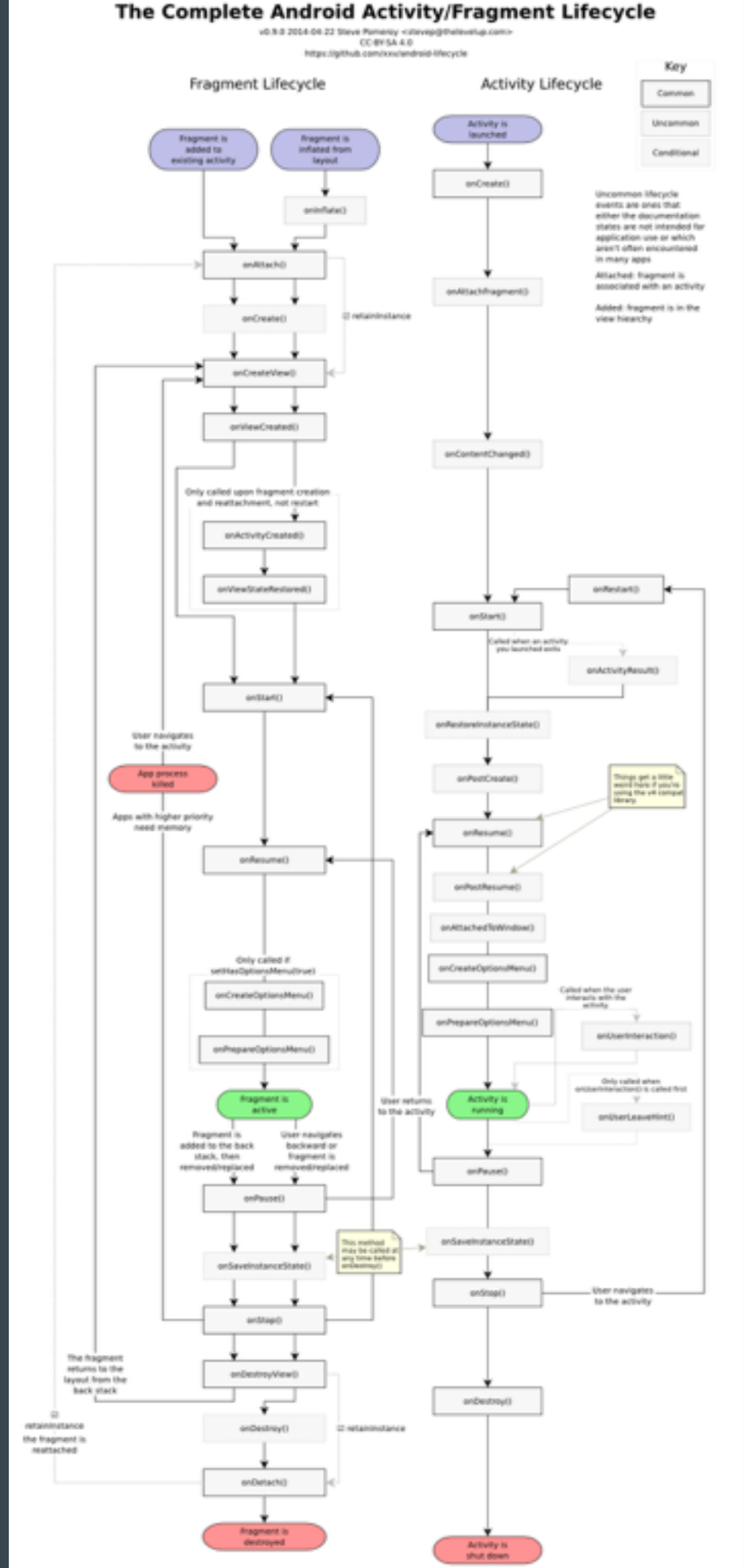

Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down


Fragment is added
onAttach()
onCreate()
onCreateView()
onActivityCreated()
onStart()
onResume()
Fragment is active
User navigates backward or fragment is removed/replaced
The fragment is added to the back stack, then removed/replaced
onPause()
onStop()
onDestroyView()
The fragment returns to the layout from the back stack
onDestroy()
onDetach()
Fragment is destroyed

The Complete Android Activity/Fragment Lifecycle

v0.9.0 2014-04-22 Steve Pomeroy <stevep@thelevelup.com>
CC-BY-SA 4.0
https://github.com/xxv/android-lifecycle

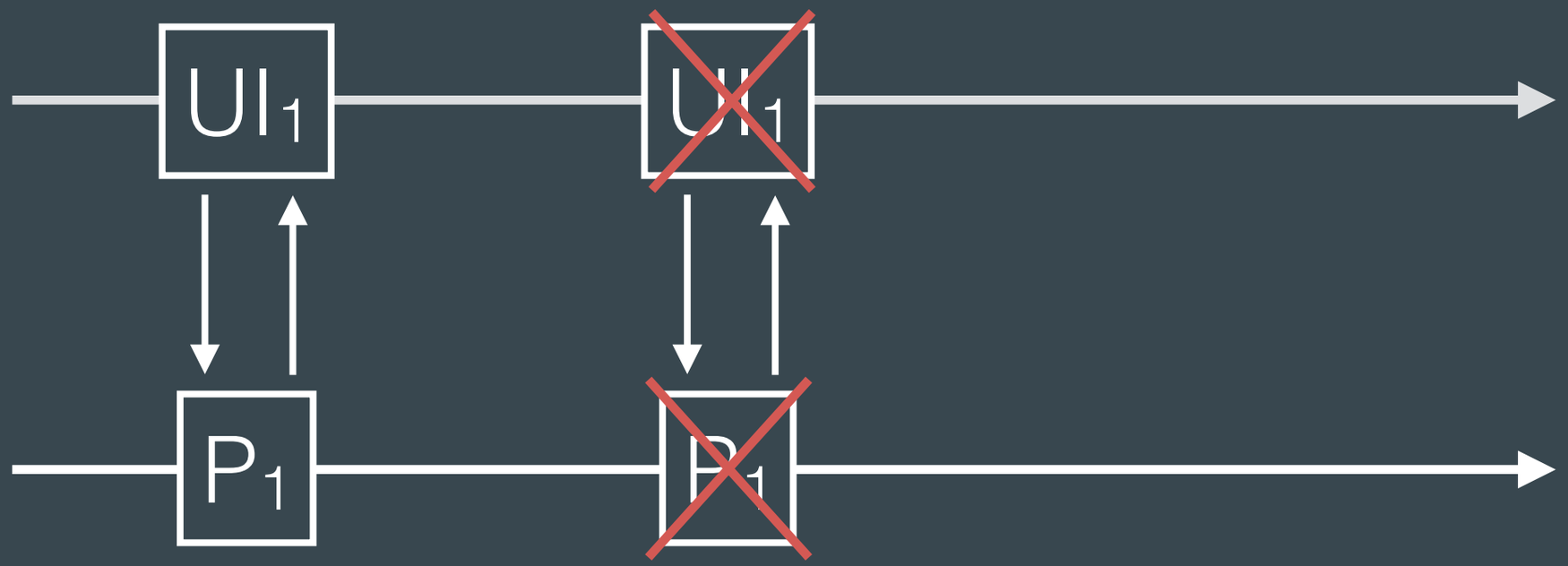


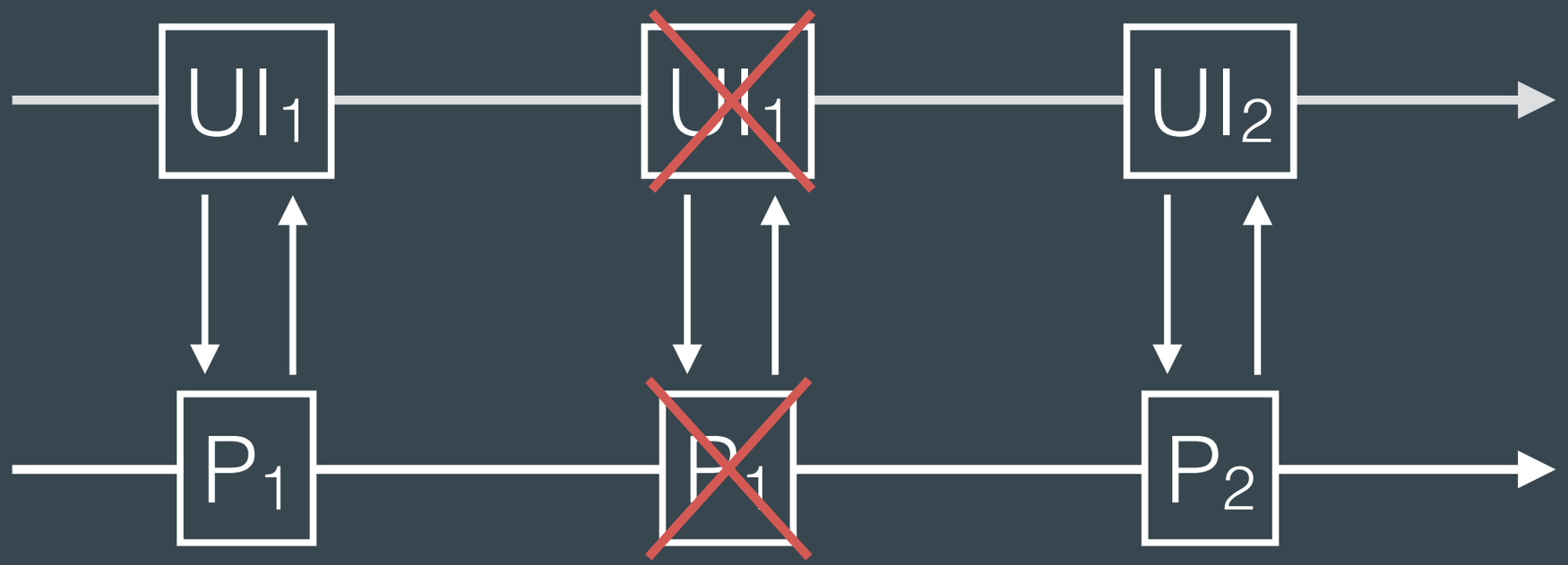

The Complete Android Activity/Fragment Lifecycle
v0.9.0 2014-04-22 Steve Pomeroy <stevep@thelevelup.com>
CC-BY-SA 4.0
https://github.com/xxv/android-lifecycle
Fragment Lifecycle
Activity Lifecycle
Key
Common
Uncommon
Conditional
Fragment is added to existing activity
Fragment is inflated from layout
onInflate()
onAttach()
onCreate()
retainInstance
onCreateView()
onViewCreated()
Only called upon fragment creation and reattachment, not restart
onActivityCreated()
onViewStateRestored()
onStart()
User navigates to the activity
App process killed
Apps with higher priority need memory
onResume()
Only called if setHasOptionsMenu(true)
onCreateOptionsMenu()
onPrepareOptionsMenu()
Fragment is active
Fragment is added to the back stack, then removed/replaced
User navigates backward or Fragment is removed/replaced
onPause()
onSaveInstanceState()
onStop()
The fragment returns to the layout from the back stack
onDestroyView()
onDestroy()
retainInstance
the fragment is reattached
onDetach()
Fragment is destroyed
Activity is launched
onCreate()
Uncommon lifecycle events are ones that either the documentation states are not intended for application use or which aren't often encountered in many apps
Attached: fragment is associated with an activity
Added: fragment is in the view hierarchy
onAttachFragment()
onContentChanged()
onRestart()
onStart()
Called when an activity you launched exits
onActivityResult()
onRestoreInstanceState()
onPostCreate()
Things get a little weird here if you're using the v4 compat library.
onResume()
onPostResume()
onAttachedToWindow()
onCreateOptionsMenu()
Called when the user interacts with the activity
onPrepareOptionsMenu()
onUserInteraction()
Only called when onUserInteraction() is called first
onUserLeaveHint()
Activity is running
User returns to the activity
onPause()
This method may be called at any time before onDestroy()
onSaveInstanceState()
onStop()
User navigates to the activity
onDestroy()
Activity is shut down

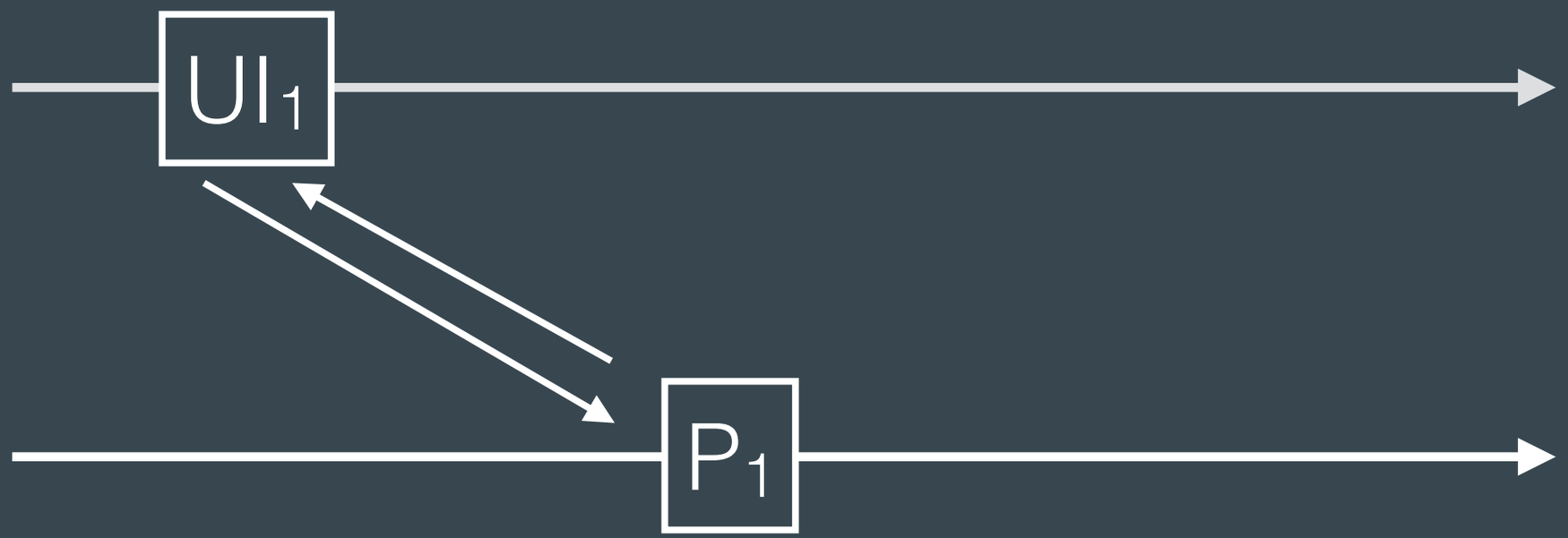
LOLCycle

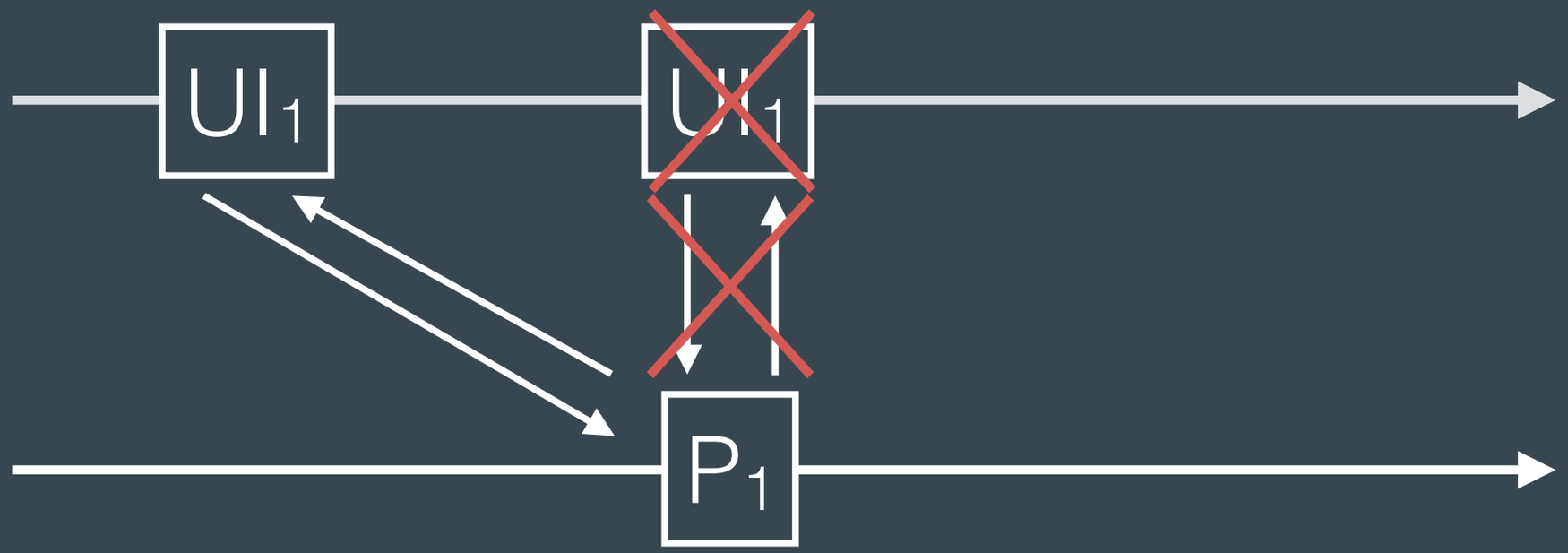
```
public interface Presenter<T extends UI> {
  void attachUI(T ui);
}
```

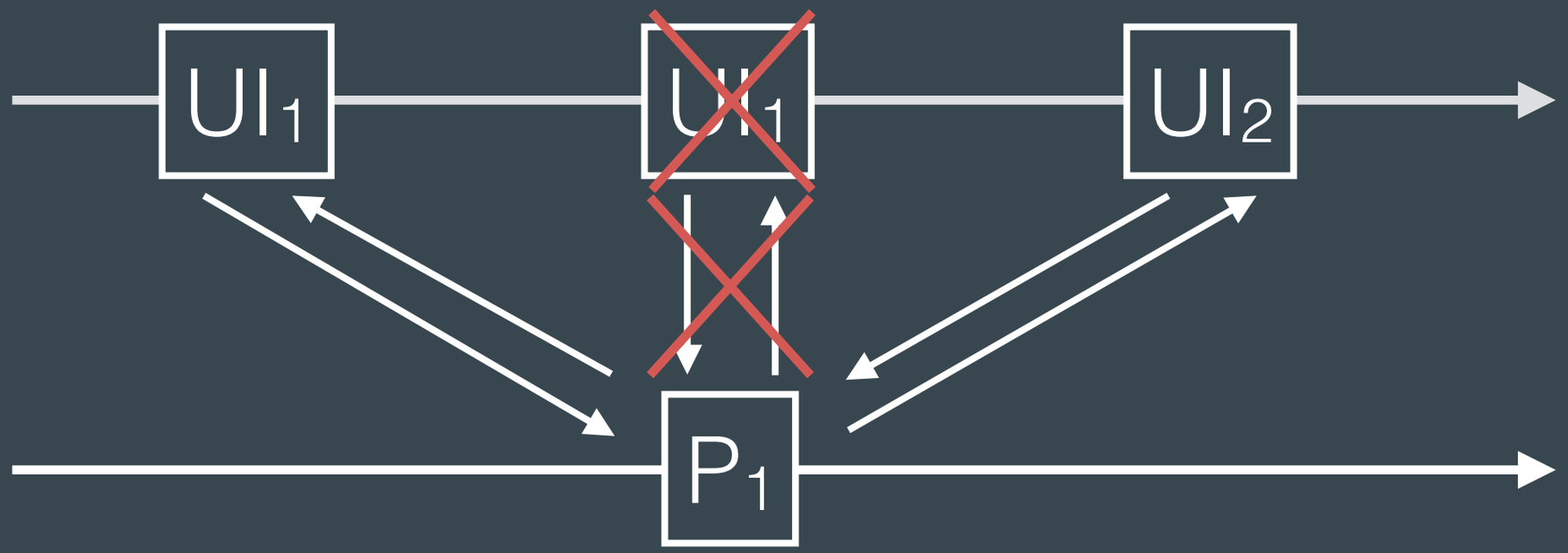

UI1
P1


UI1
UI1
P1
P1


UI1
UI1
UI2
P1
P1
P2

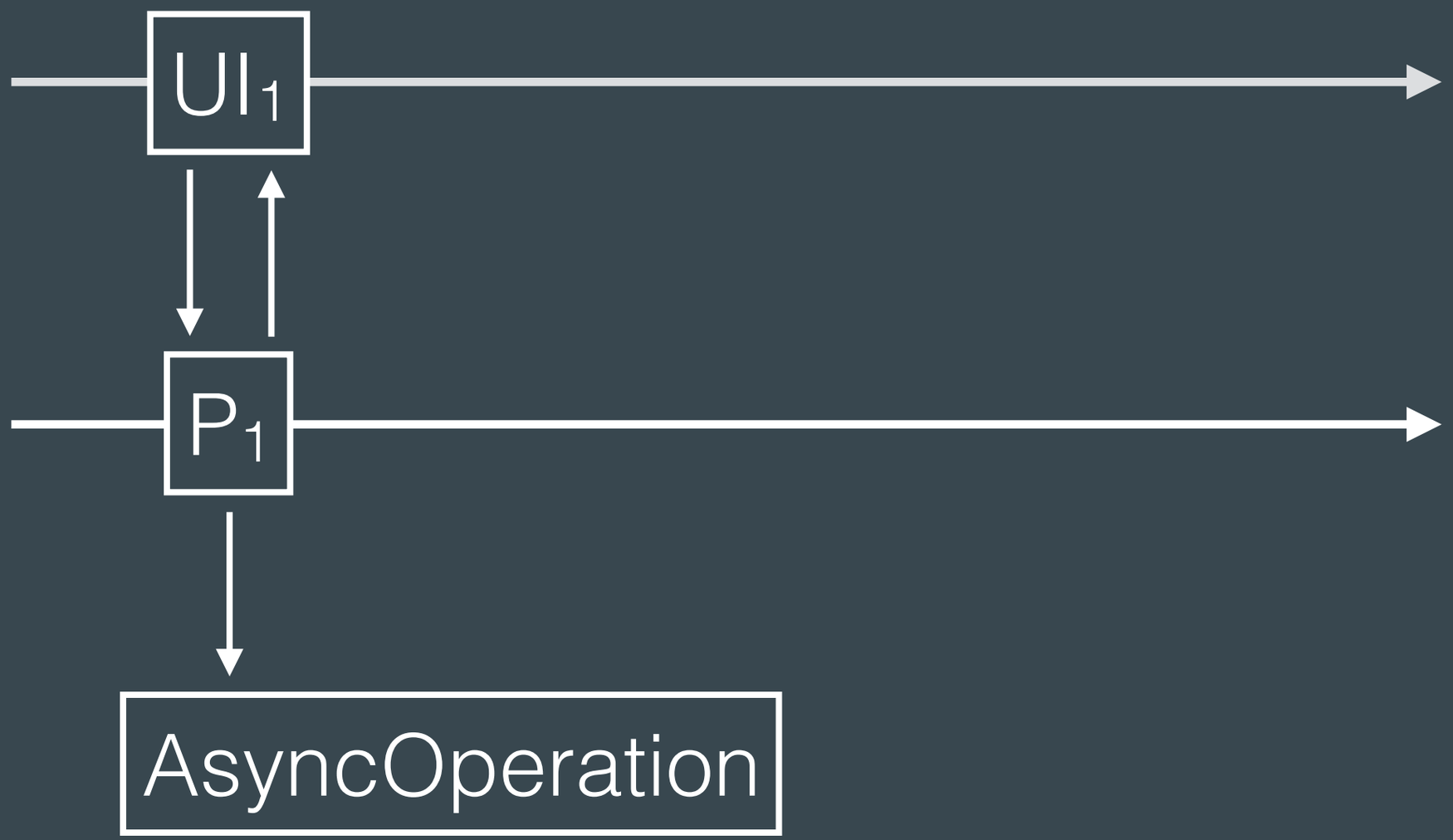

UI1
P1

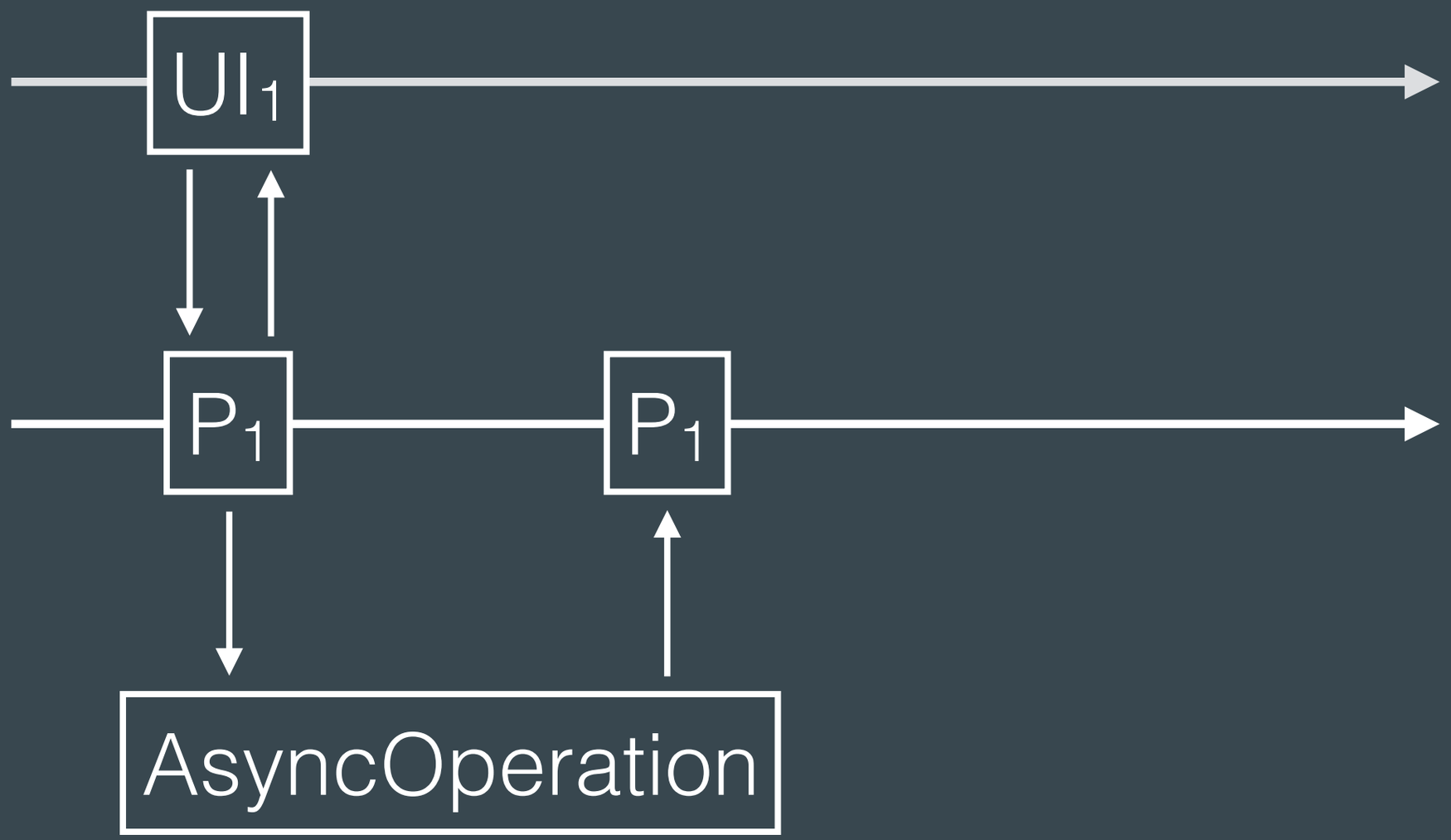

UI1
UI1
P1

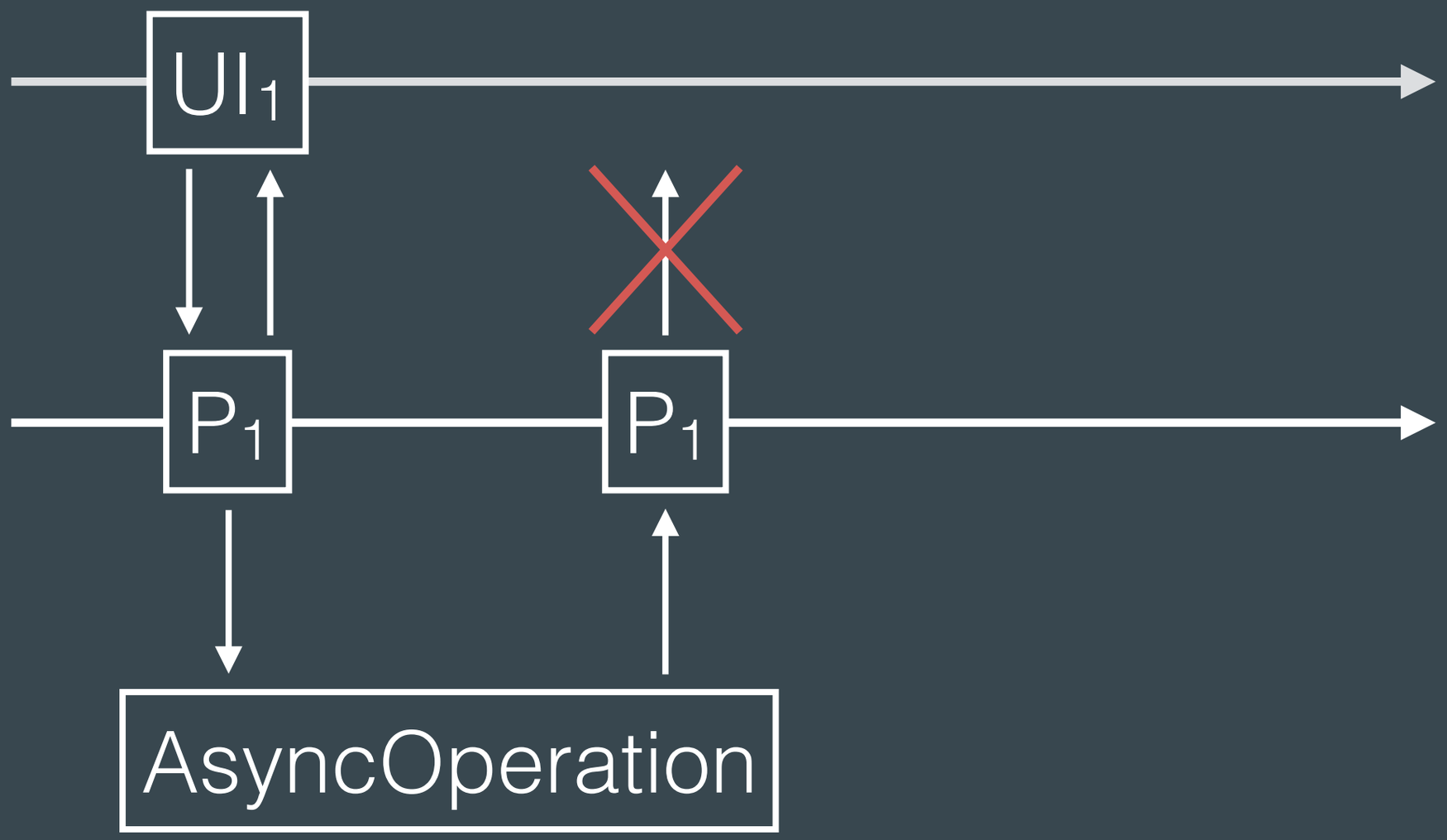

UI1
UI1
UI2
P1

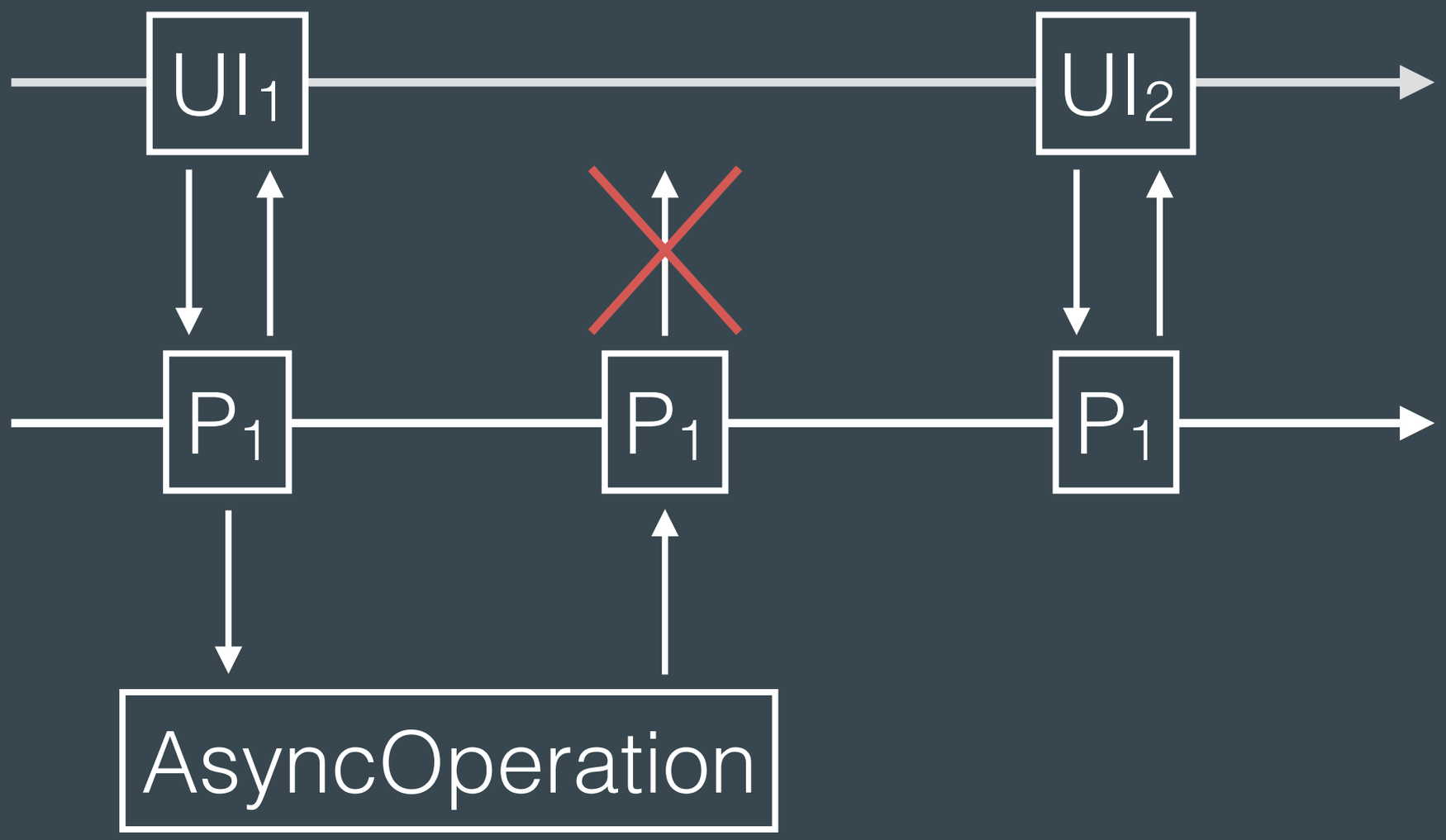
```java
public interface Presenter<T extends UI> {
  void attachUI(T ui);

  void detachUI();
}
```

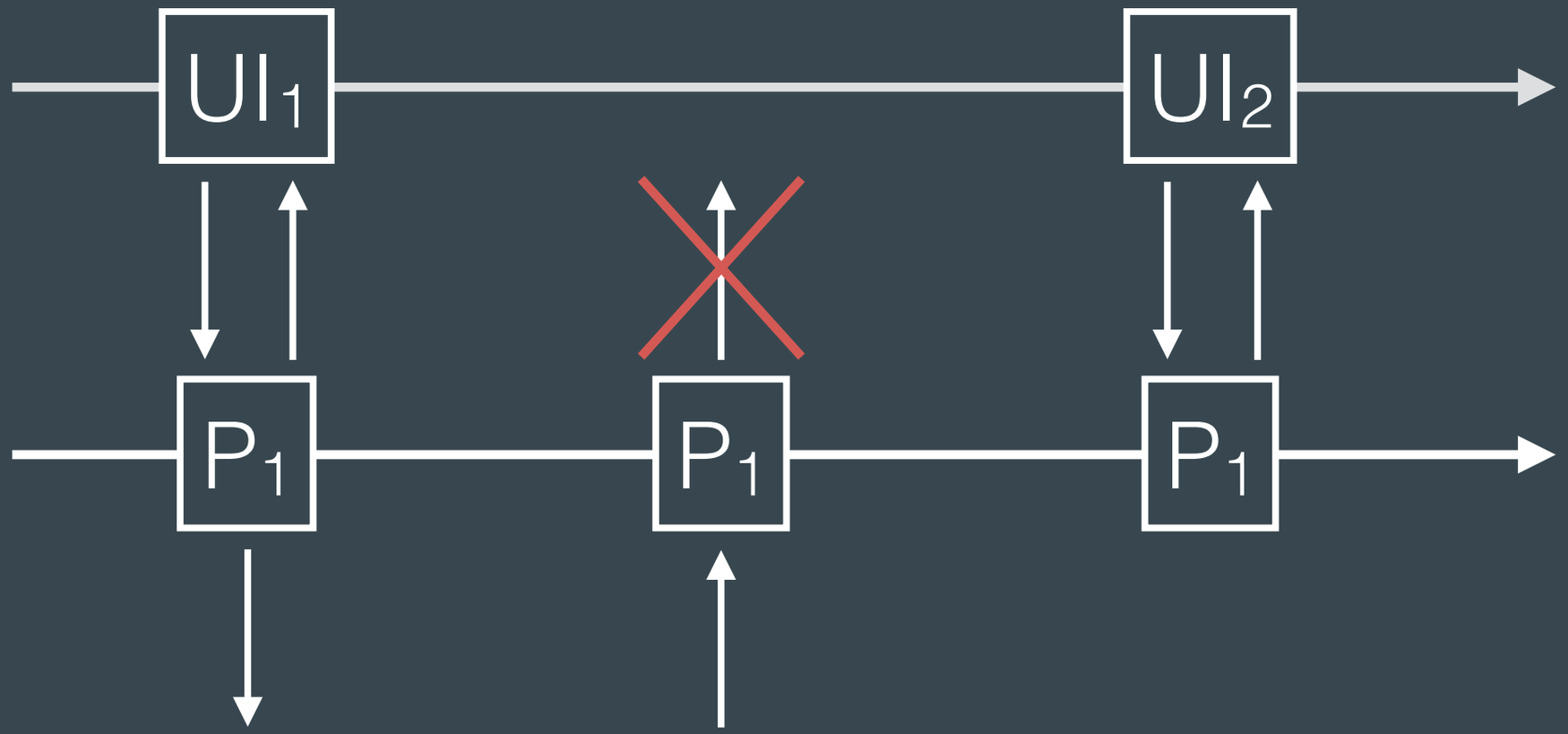

UI1
P1


UI1
P1
AsyncOperation


UI1
P1
P1
AsyncOperation


UI1
P1
P1
AsyncOperation


UI1
UI2
P1
P1
P1
AsyncOperation



AsyncOperation

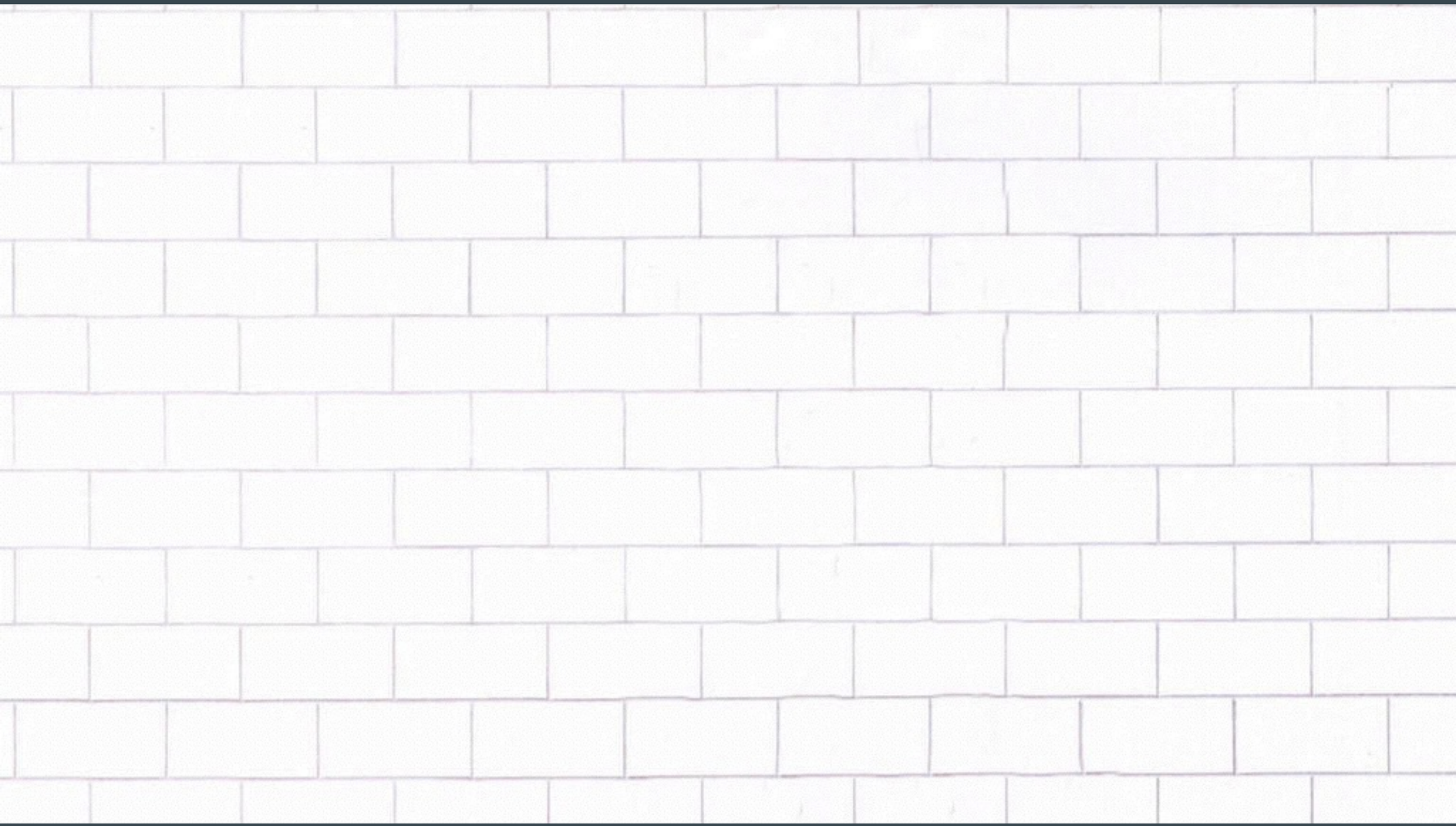
```
public interface UICommand<T> {
  void execute(T ui);
}

private Queue<UICommand<T>> commandQueue = new LinkedList<>();
```

```
public interface ProductListUI extends UI {
  void showProductList(List<ProductViewModel> productViewModels);

  void showProgress();

  void navigateToAddProduct();
}
```

```
public interface ProductListUI extends UI {
  void showProductList(List<ProductViewModel> productViewModels);

  void showProgress();

  void navigateToAddProduct();
}
```

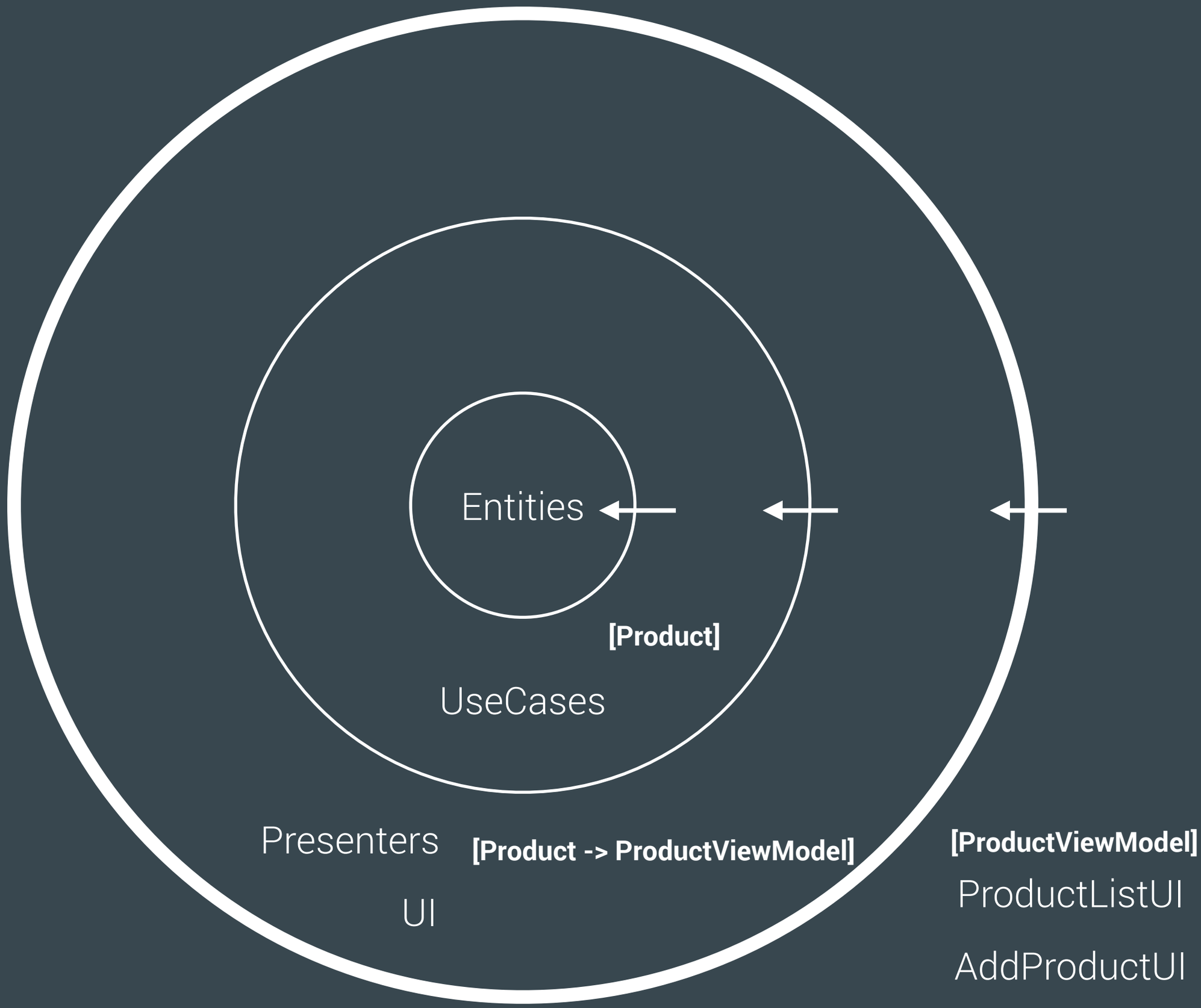
```
public class ShowProgressCommand implements UICommand<ProductListUI> {
  @Override public void execute(ProductListUI ui) {
    ui.showProgress();
  }
}

public class PresentContentCommand implements UICommand<ProductListUI> {
  private final List<ProductViewModel> products;

  public PresentContentCommand(List<ProductViewModel> products) {
    this.products = products;
  }

  @Override public void execute(ProductListUI ui) {
    ui.showProductList(this.products);
  }
}
```

```java
@Singleton
public class ProductListPresenter
  extends BasePresenter<ProductListPresenter.ProductListUI> {

  @Override protected void onFirstUIAttachment() {
    updateProductList(true);
  }

  public void productStatusChanged(long productId, boolean isBought) {
  }

  public void onAddNewProduct() {
  }

  public void onRemoveBoughtProducts() {
  }

  private void updateProductList(boolean withProgress) {
    if (withProgress) {
      execute(new ShowProgressCommand(), true);
    }
    asyncUseCase.wrap(listProductsUseCase).subscribe(
        new Action1<ProductList>() {
          @Override public void call(ProductList products) {
            List<ProductViewModel> viewModels = mapper.toViewModel(products);
            executeRepeat(new PresentContentCommand(viewModels));
          }
        }
    );
  }
}
```



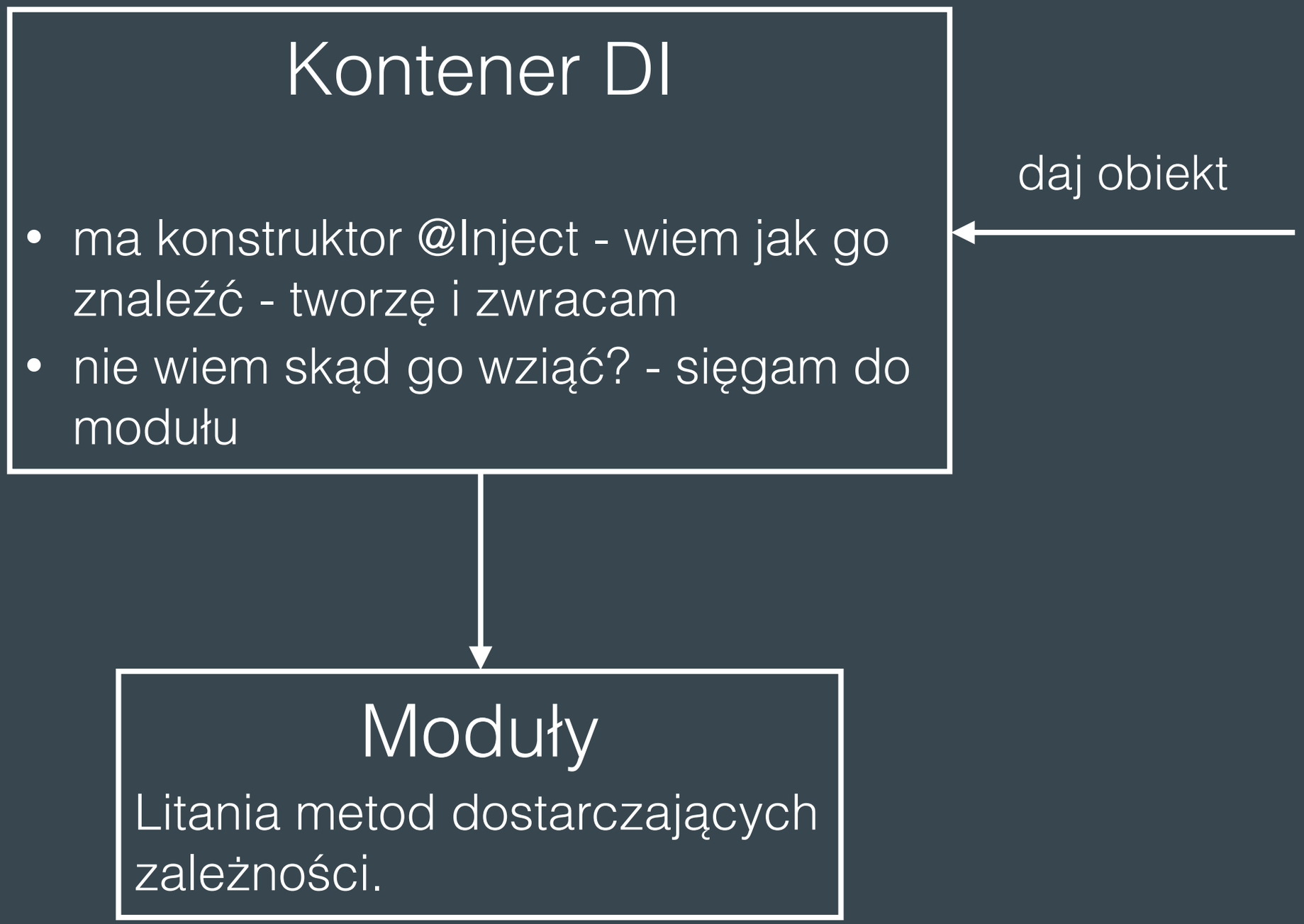
Pink Floyd - The Wall Cover

```java
public class ProductListFragment extends
  PresenterListFragment<ProductListPresenter.ProductListUI>
  implements ProductListPresenter.ProductListUI {

  @Inject ProductListPresenter presenter;

  @Override public void showProductList(List<ProductViewModel> productViewModels) {
    ProductListAdapter adapter = (ProductListAdapter) getListAdapter();
    if (adapter != null) {
      adapter.swapData(productViewModels);
    } else {
      setListAdapter(new ProductListAdapter(
        getActivity(),
        productViewModels,
        onProductStatusChangedListener)
      );
    }
    setListShown(true);
  }

  @Override public void onCreate(Bundle savedInstanceState) {
    // setup
    onProductStatusChangedListener = new OnProductStatusChangedListener() {
      @Override public void onProductStatusChanged(long productId, boolean isBought) {
        presenter.productStatusChanged(productId, isBought);
      }
    };
  }
}
```


Entities
[Product]
UseCases
Presenters
[Product -> ProductViewModel]
UI
[ProductViewModel]
ProductListUI
AddProductUI

DI

Dagger


Kontener DI
• ma konstruktor @Inject - wiem jak go znaleźć - tworzę i zwracam
• nie wiem skąd go wziąć? - sięgam do modułu
daj obiekt
Moduły
Litania metod dostarczających zależności.

Dagger

ObjectGraph

Kontener DI

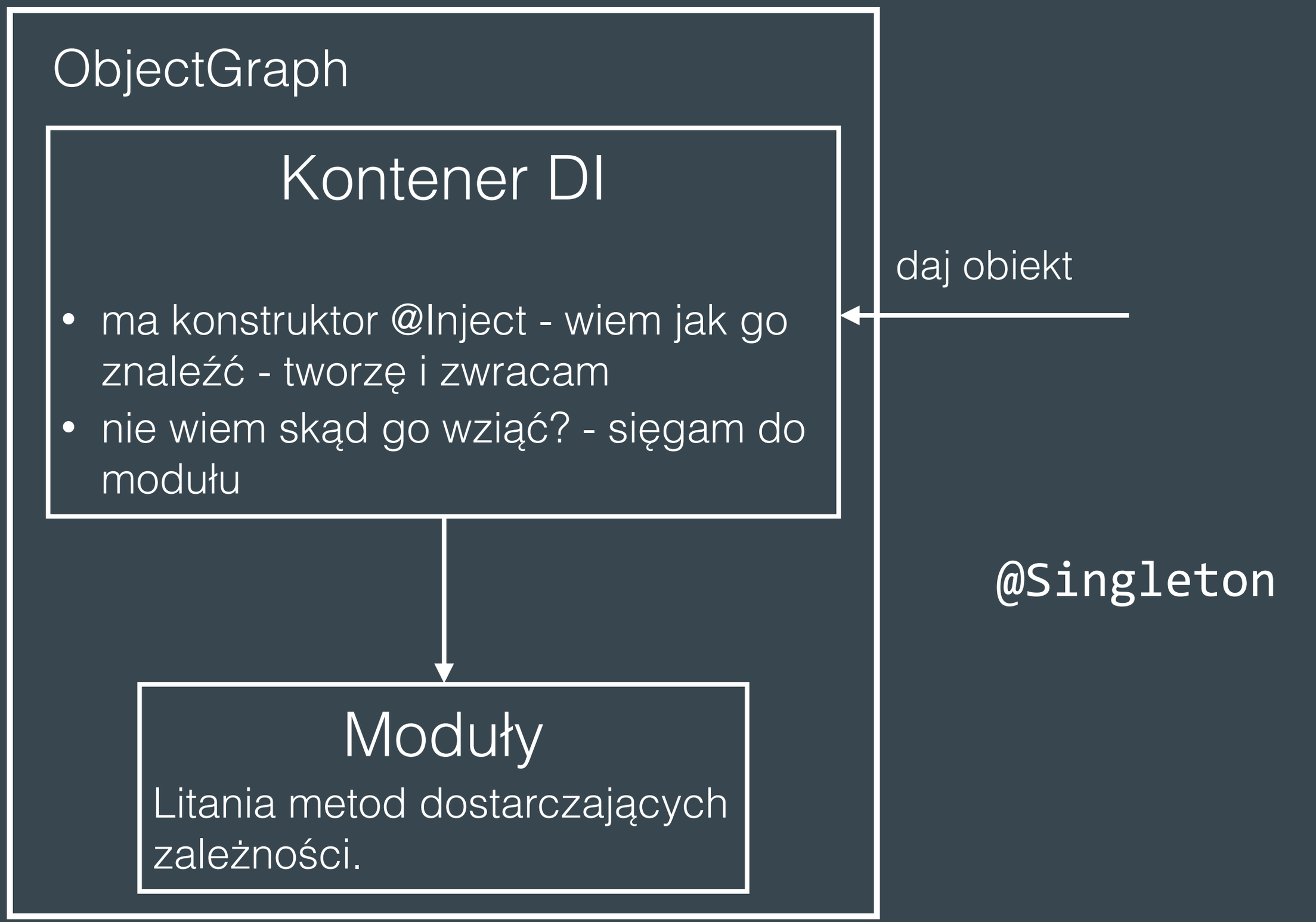
- ma konstruktor @Inject - wiem jak go znaleźć - tworzę i zwracam
- nie wiem skąd go wziąć? - sięgam do modułu

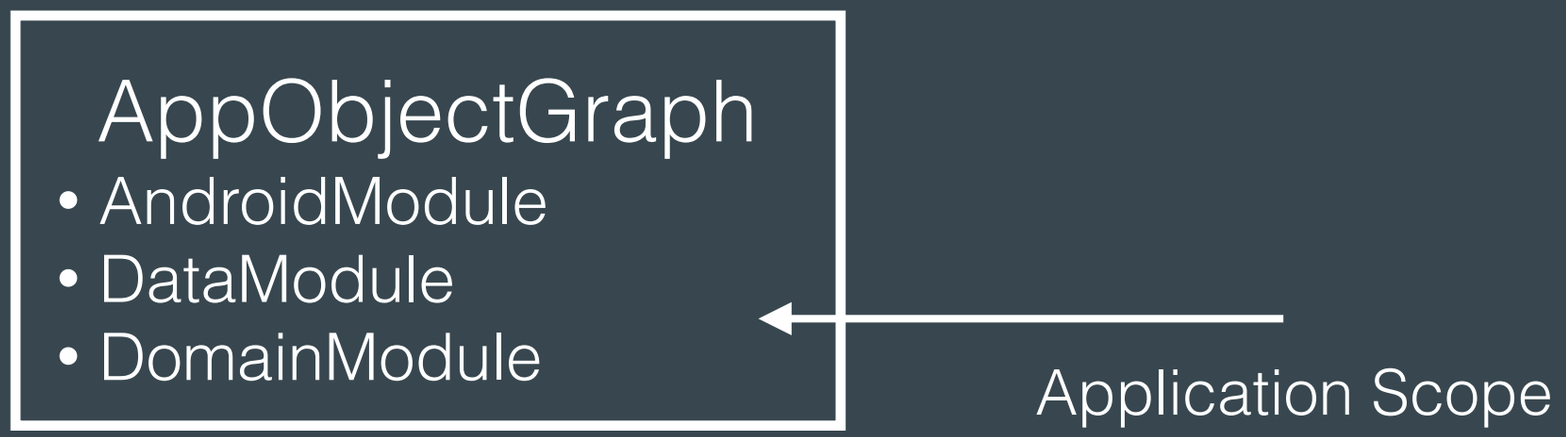
daj obiekt

Moduły

Litania metod dostarczających zależności.

Dagger


ObjectGraph
Kontener DI
• ma konstruktor @Inject - wiem jak go znaleźć - tworzę i zwracam
• nie wiem skąd go wziąć? - sięgam do modułu
Moduły
Litania metod dostarczających zależności.
daj obiekt
@Singleton


AppObjectGraph
• AndroidModule
• DataModule
• DomainModule
Application Scope

PresenterActivityOG

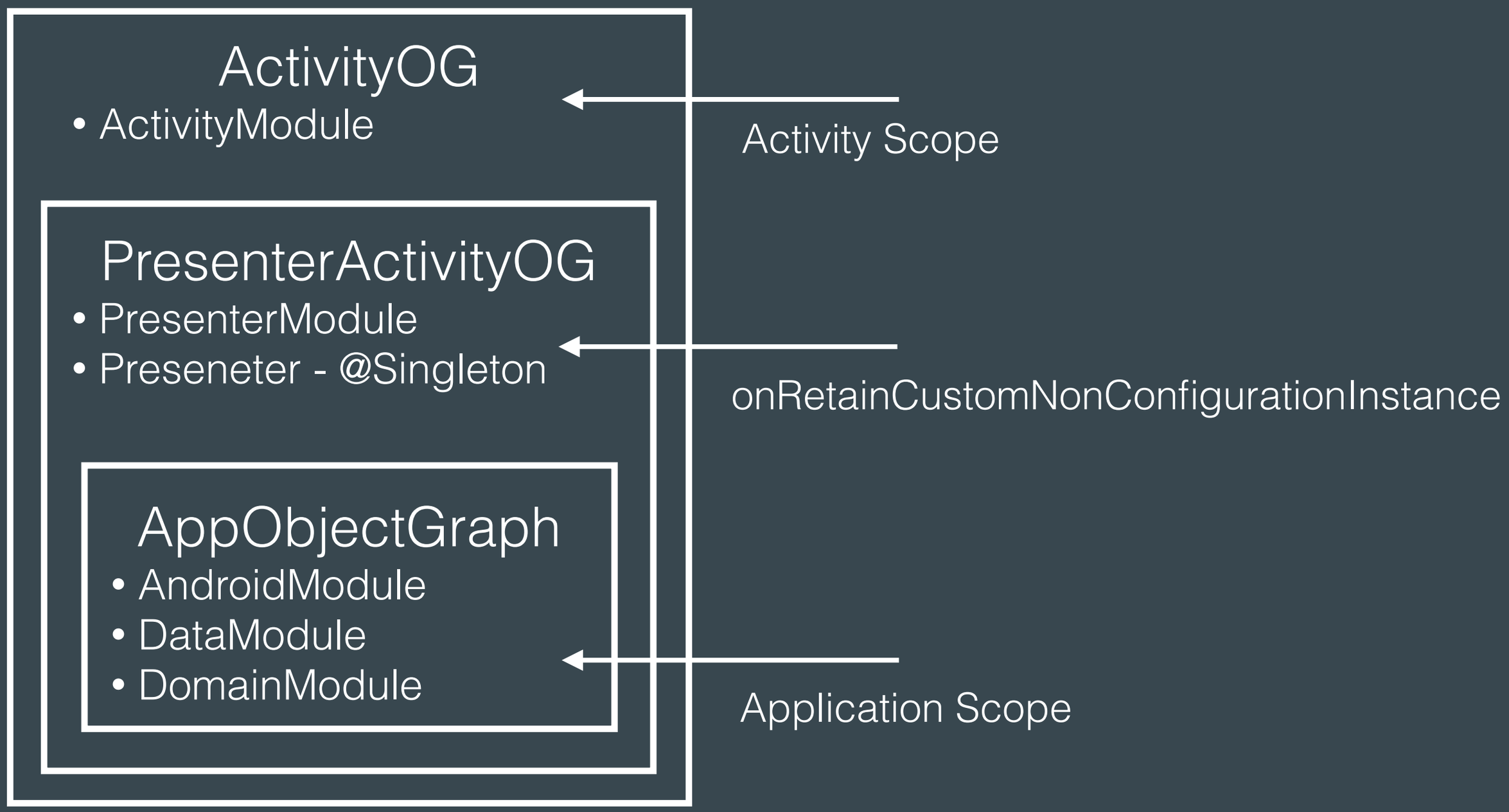
- PresenterModule
- Preseneter - @Singleton

onRetainCustomNonConfigurationInstance

AppObjectGraph

- AndroidModule
- DataModule
- DomainModule

Application Scope


ActivityOG
• ActivityModule
Activity Scope
PresenterActivityOG
• PresenterModule
• Preseneter - @Singleton
onRetainCustomNonConfigurationInstance
AppObjectGraph
• AndroidModule
• DataModule
• DomainModule
Application Scope

Morał?

Morał?

Dependency Inversion Principle FTW!

https://github.com/mcharmas/shoppinglist-clean-architecture-example

Q&A?

http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html

http://blog.8thlight.com/uncle-bob/2011/09/30/Screaming-Architecture.html

https://vimeo.com/43612849

https://vimeo.com/80533536

http://tpierrain.blogspot.com/2013/08/a-zoom-on-hexagonalcleanonion.html