Our code should

- be bugs free
- be agile
- be readable
- write fast
- ... just work

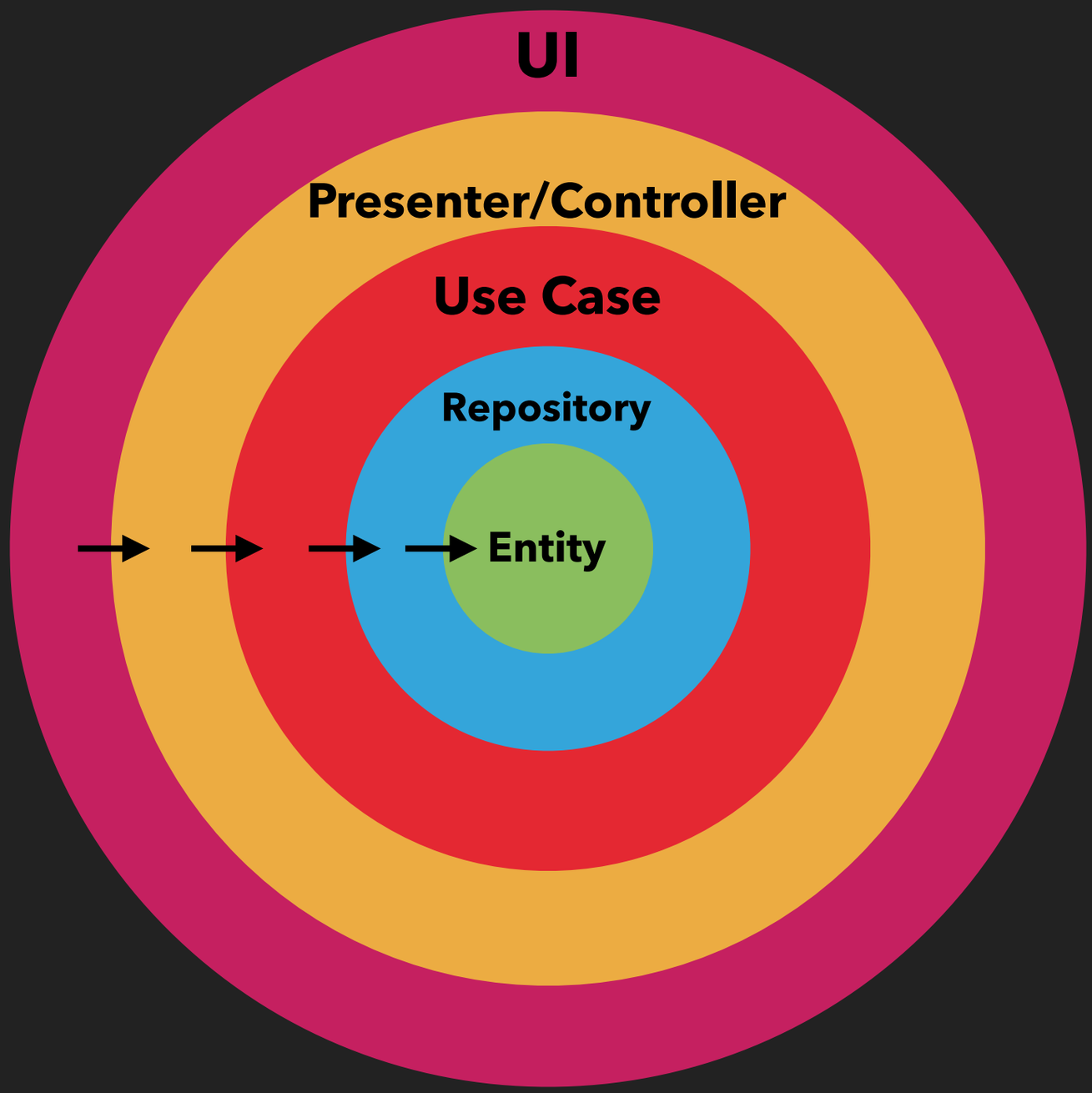
Legacy Code

Clean Architecture

Clean Architecture in Android

- implementation problems
- how to resolve them
- pros of Clean Architecture

Clean Architecture



Clean Architecture

+

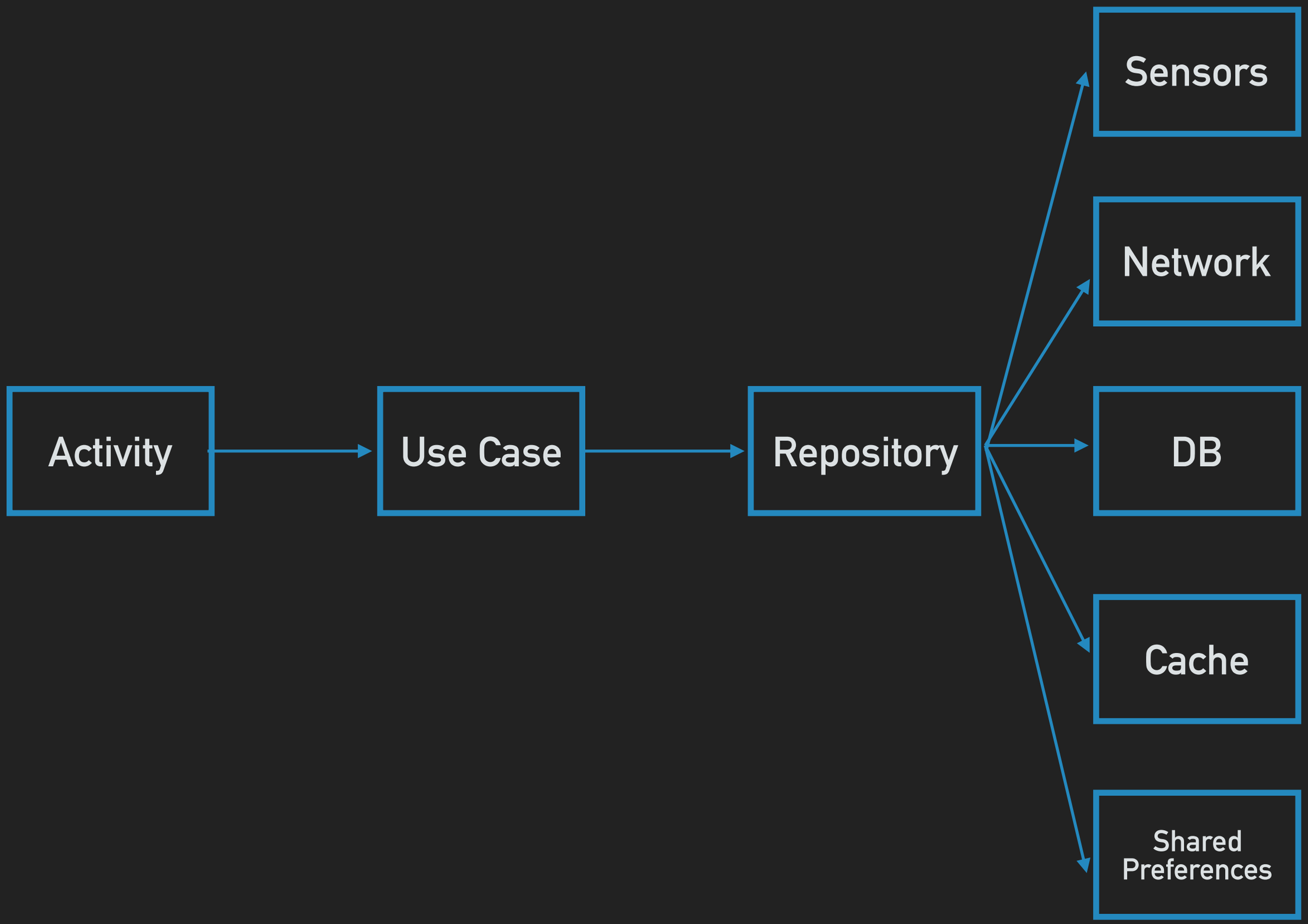
Android

UI

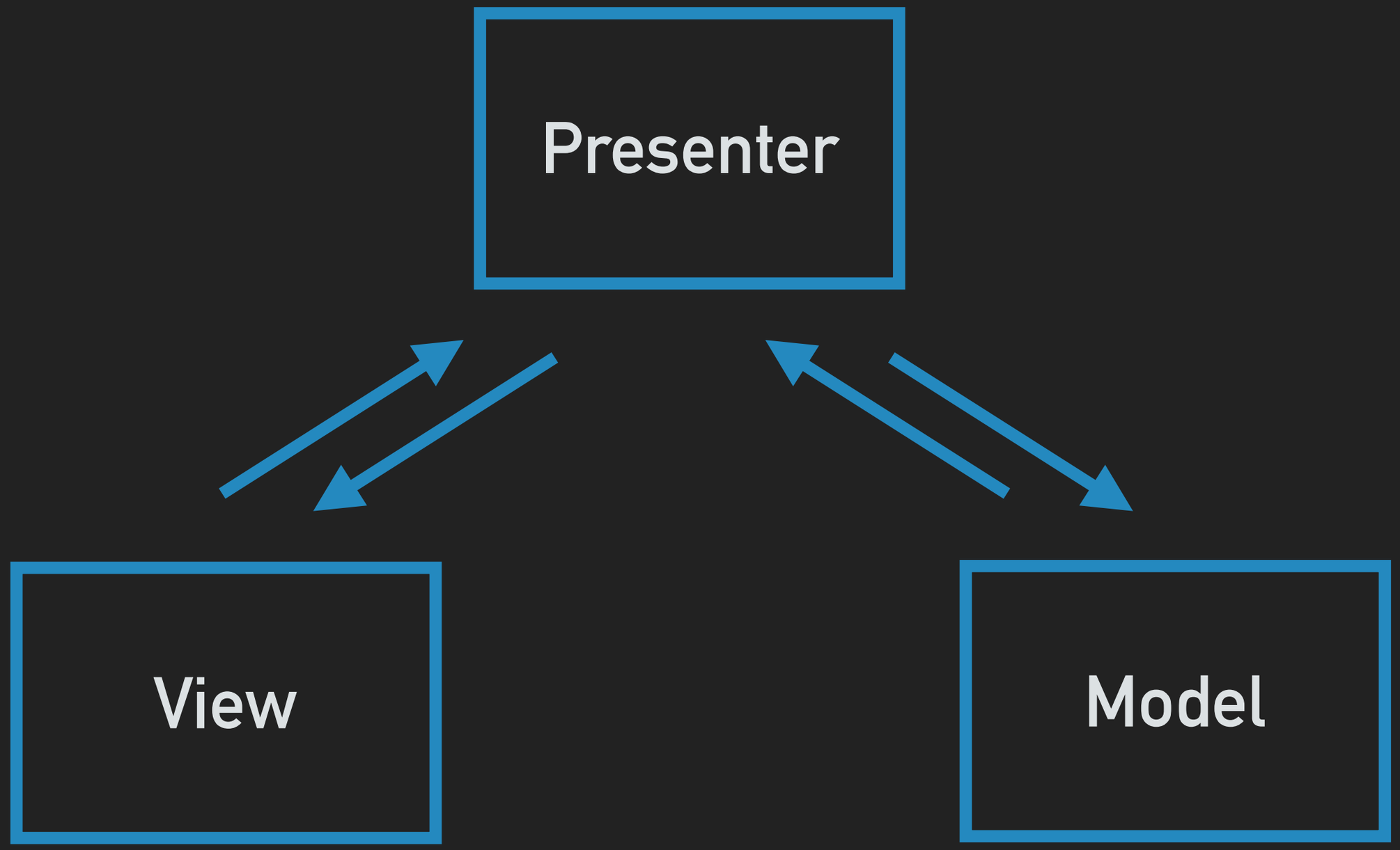
=

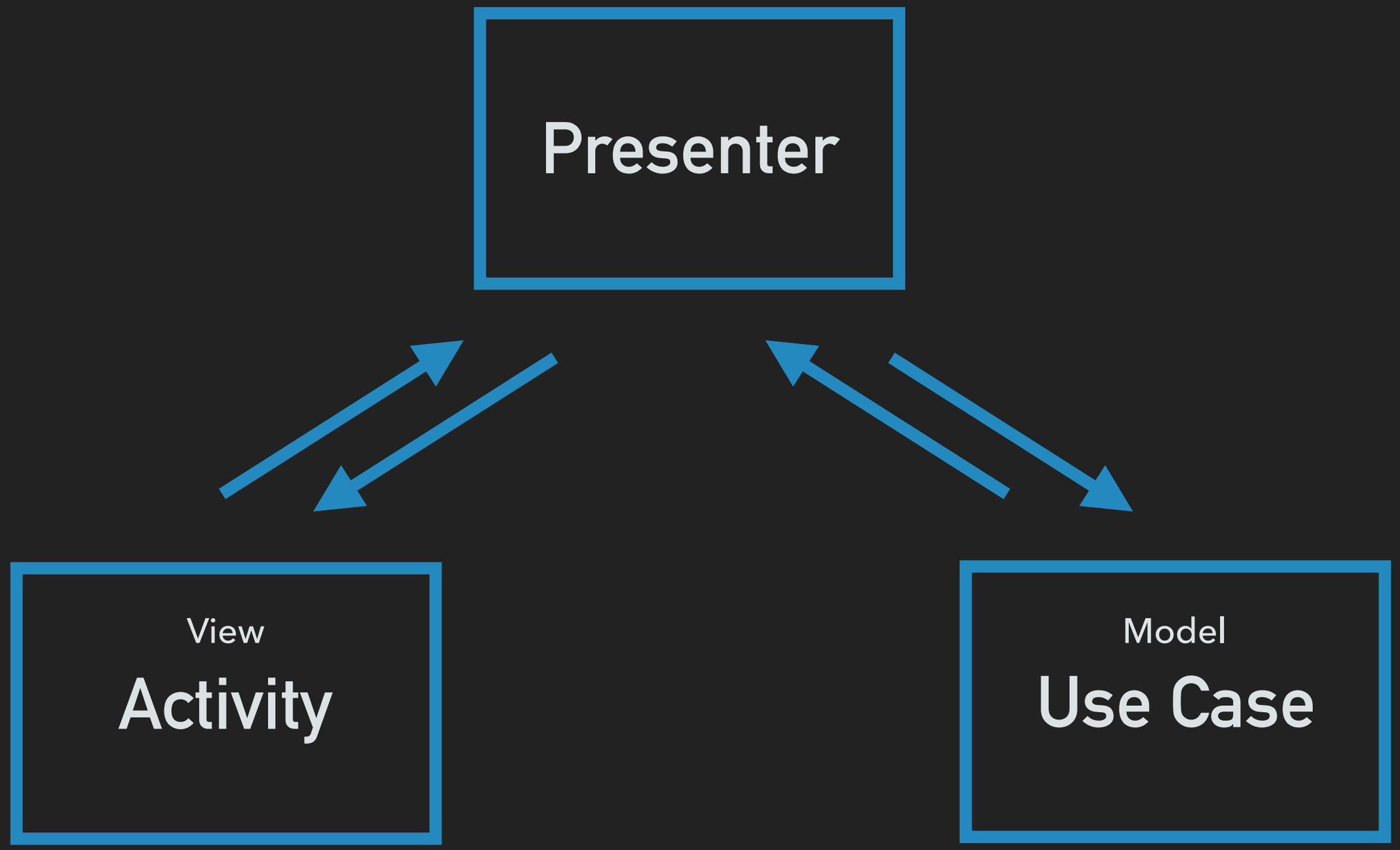
Activity

Android Clean Architecture



God Activity


Presenter
View
Model


Presenter
View
Activity
Model
Use Case

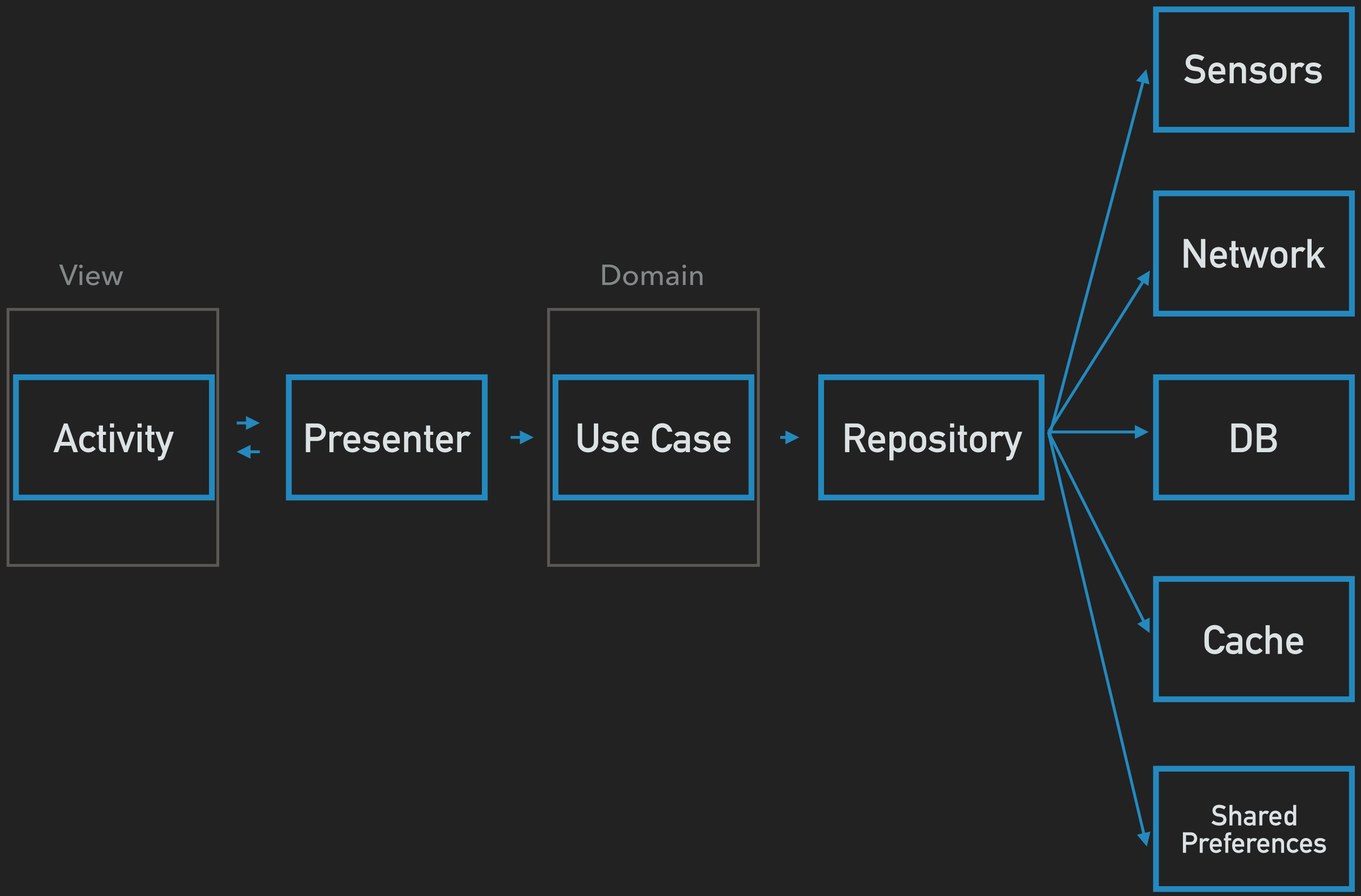
```
public class EventsActivity … implements EventsView {
    EventsPresenter eventsPresenter;

    public void onClick(View v) {
        eventsPresenter.getEvent();
    }
}

public class EventsPresenter {
    …

    public void getEvents() {
        eventsView.showProgress();

        GetEventsUseCase useCase = new GetEventsUseCase();
        EventsEntity events = useCase.execute();

        eventsView.hideProgress();
        eventsView.setEvents(events);
    }

}
```

ViewModel

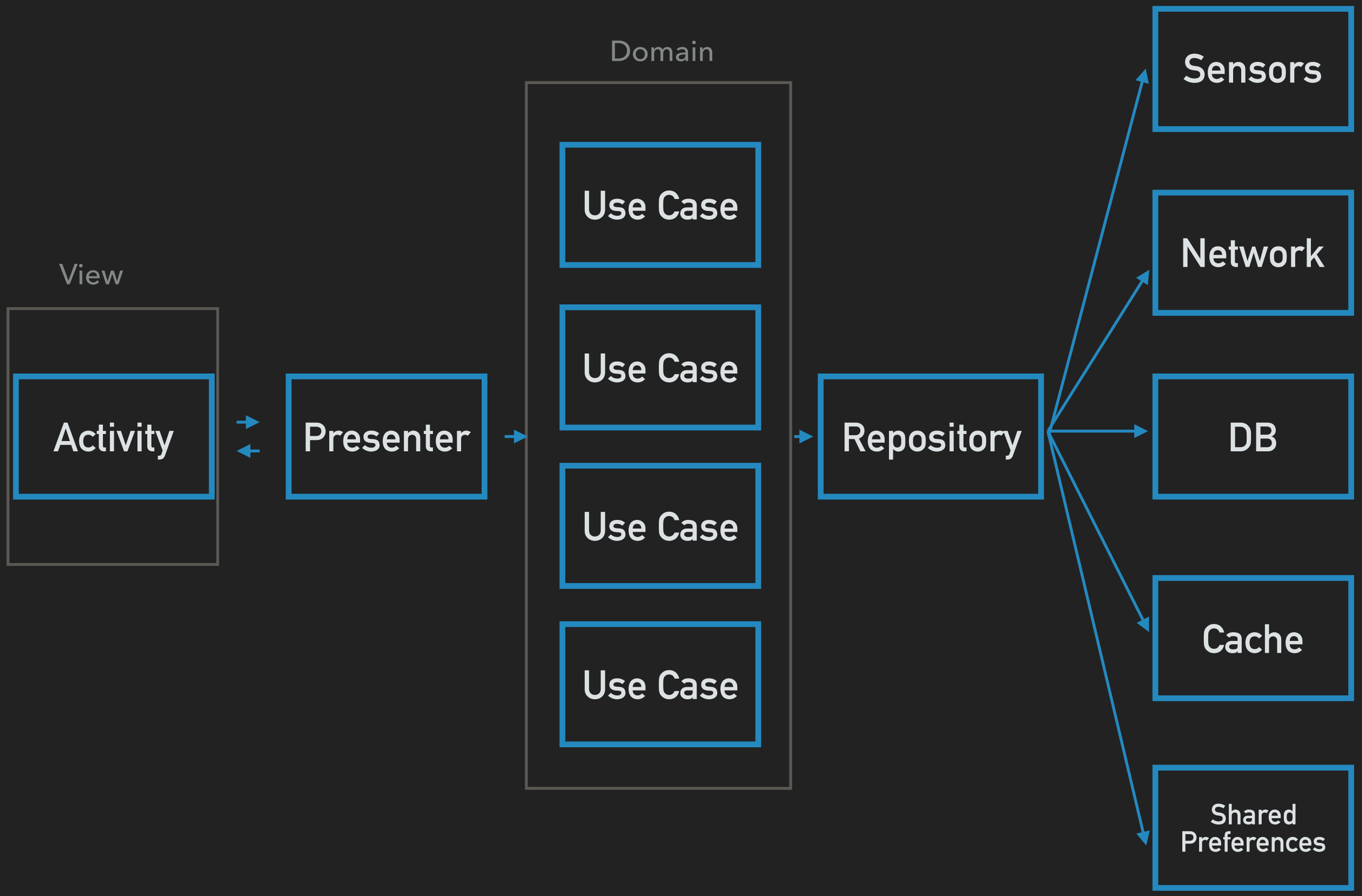
```
public class EventsPresenter {
    public void getEvents() {
        …

        EventsMapper mapper = new EventsMapper();
        EventsViewModel viewModel = mapper.transform(eventsEntity);

        eventsView.setEventsViewModel(viewModel);
    }
}
```

```
public class EventsMapper {

    public EventViewModel transform(EventEntity event) {

        EventViewModel vm = new EventViewModel();

        vm.setEventDate(
                new SimpleDateFormat("dd-MM-yyyy")
                    .format(event.getDate()));

        return vm;
    }

}
```

Android Clean Architecture

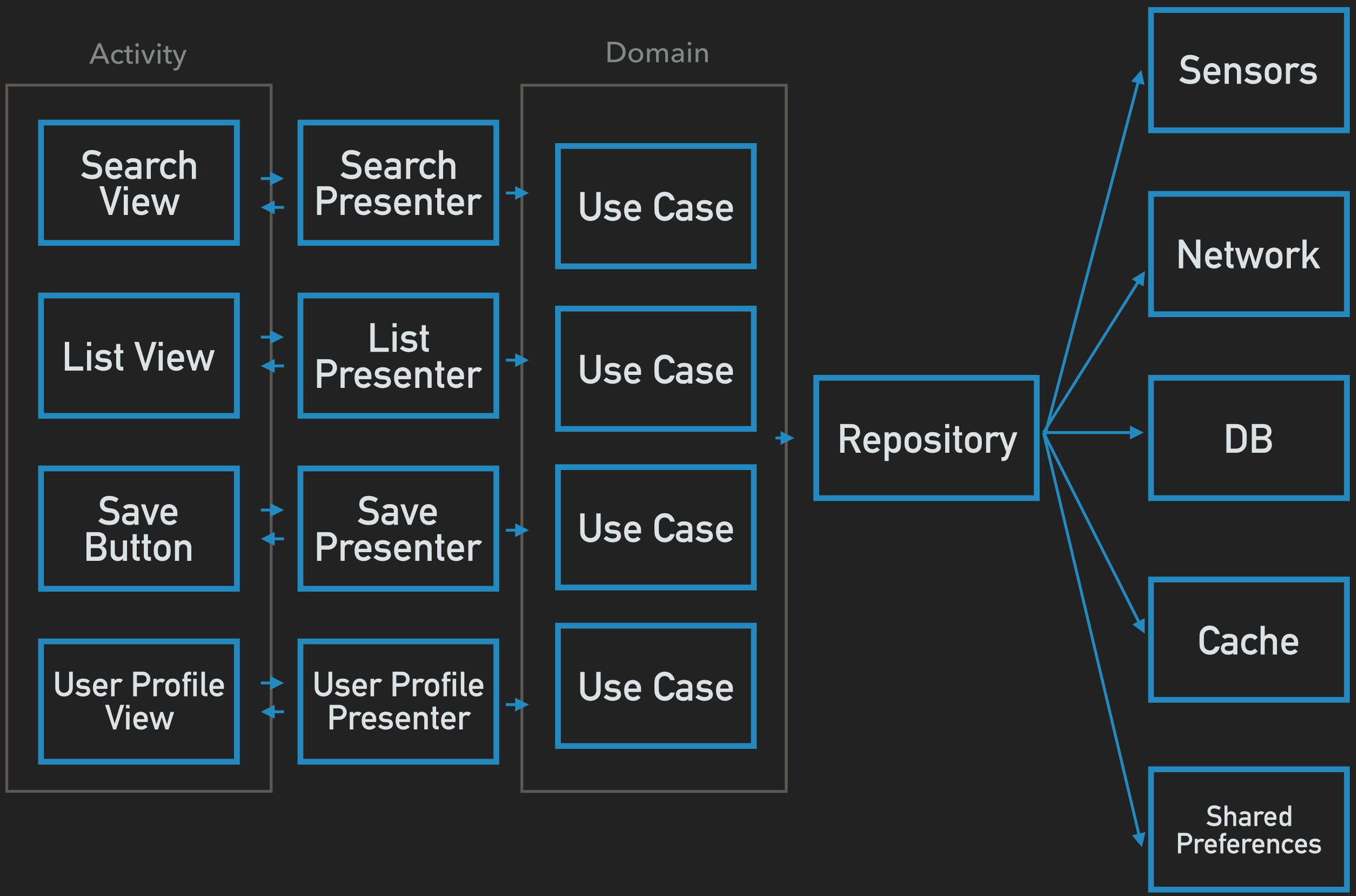


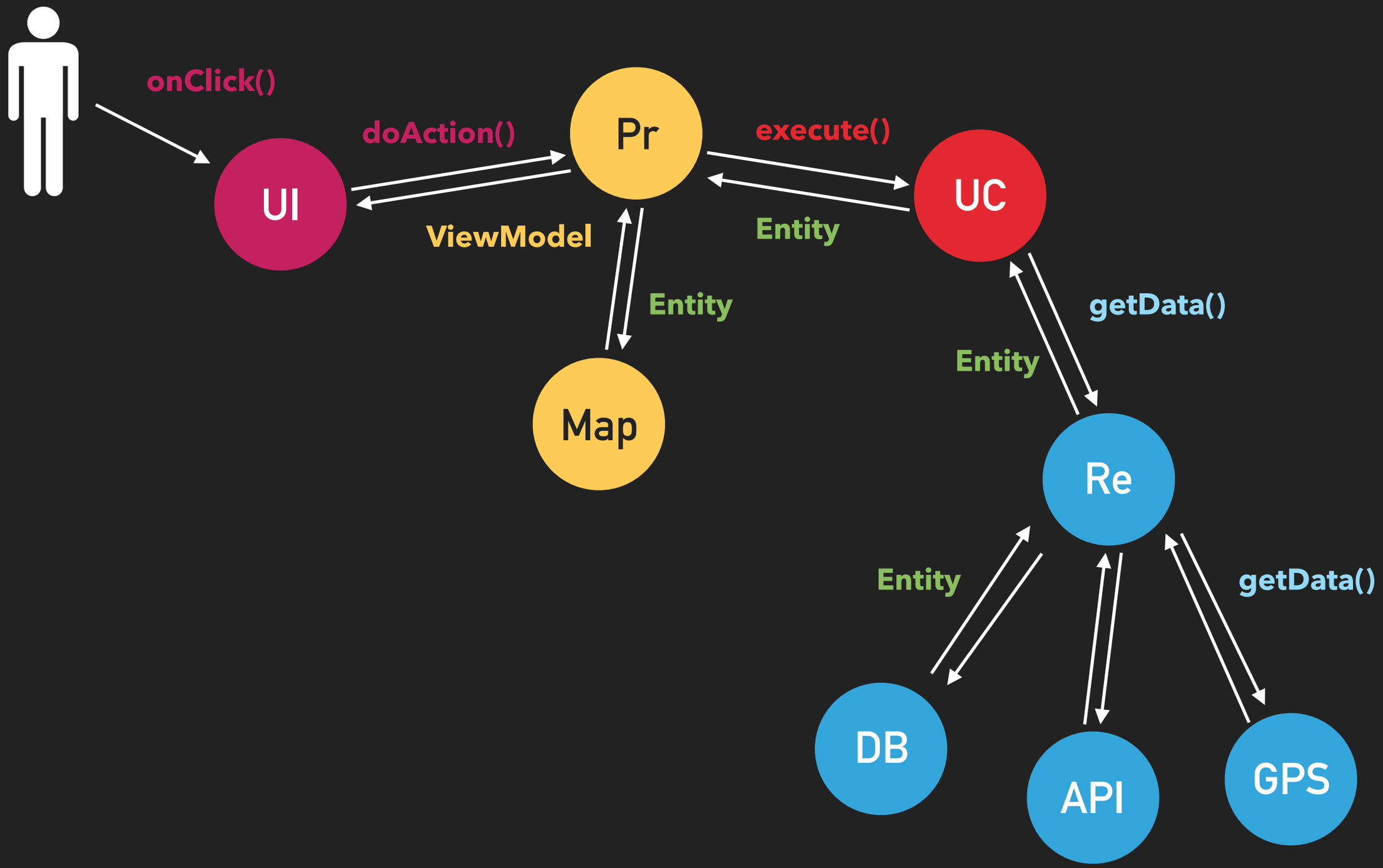
Android Clean Architecture



God Presenter

Android Clean Architecture




onClick()
UI
doAction()
ViewModel
Pr
execute()
Entity
UC
Entity
Map
Entity
getData()
Re
Entity
DB
API
getData()
GPS

Clean Architecture

—

SRP

Asynchronous Calls

Callback

```
useCase.execute(requestData, new Callback<JobResponse>() {

    @Override
    public void success(JobResponse response) {
        useCase2.execute(requestData, new Callback<JobResponse2>() {
            @Override
            public void success(JobResponse2 response2) {

            }
            @Override
            public void failure(Error error) {}
        });
    }

    @Override
    public void failure(Error error) {}

});
```

Callback Hell

RxJava

```
public class EventsPresenter {
    public void getPosts() {
        GetPostsUseCase useCase = new GetPostsUseCase();
        useCase.execute(new PostsObserver());
    }
}
```

```
public class PostsObserver
        extends DisposableObserver<UserPosts> {

    @Override
    public void onNext(@NonNull UserPosts userPosts) {
        postsView.showPosts(userPosts);
    }

    @Override
    public void onError(@NonNull Throwable e) {
        postsView.hideProgress();
        postsView.showError(e.getMessage());
        postsView.showRetry();
    }

    @Override
    public void onComplete() {
        postsView.hideProgress();
    }
}
```

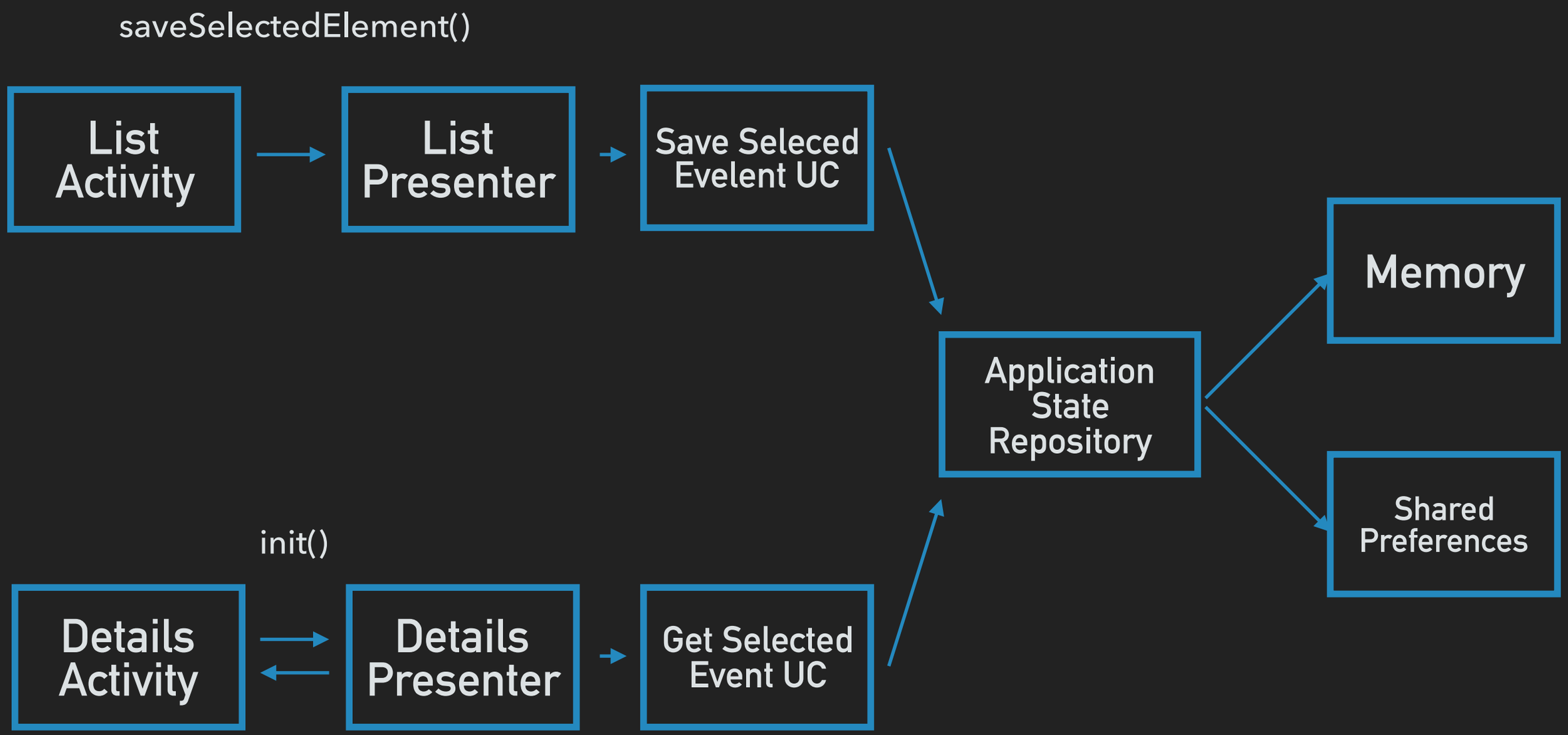
```
public class GetPostsUseCase {

    Observable<UserPosts> createUseCaseObservable() {
        return postsRepository.getPosts();
    }

    public void execute(DisposableObserver<UserPosts>
                                            postsObserver) {
        Observable<UserPosts> observable =
                createUseCaseObservable()
                    .subscribeOn(Schedulers.io())
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribeWith(postsObserver);
    }

}
```

Synchronous Calls

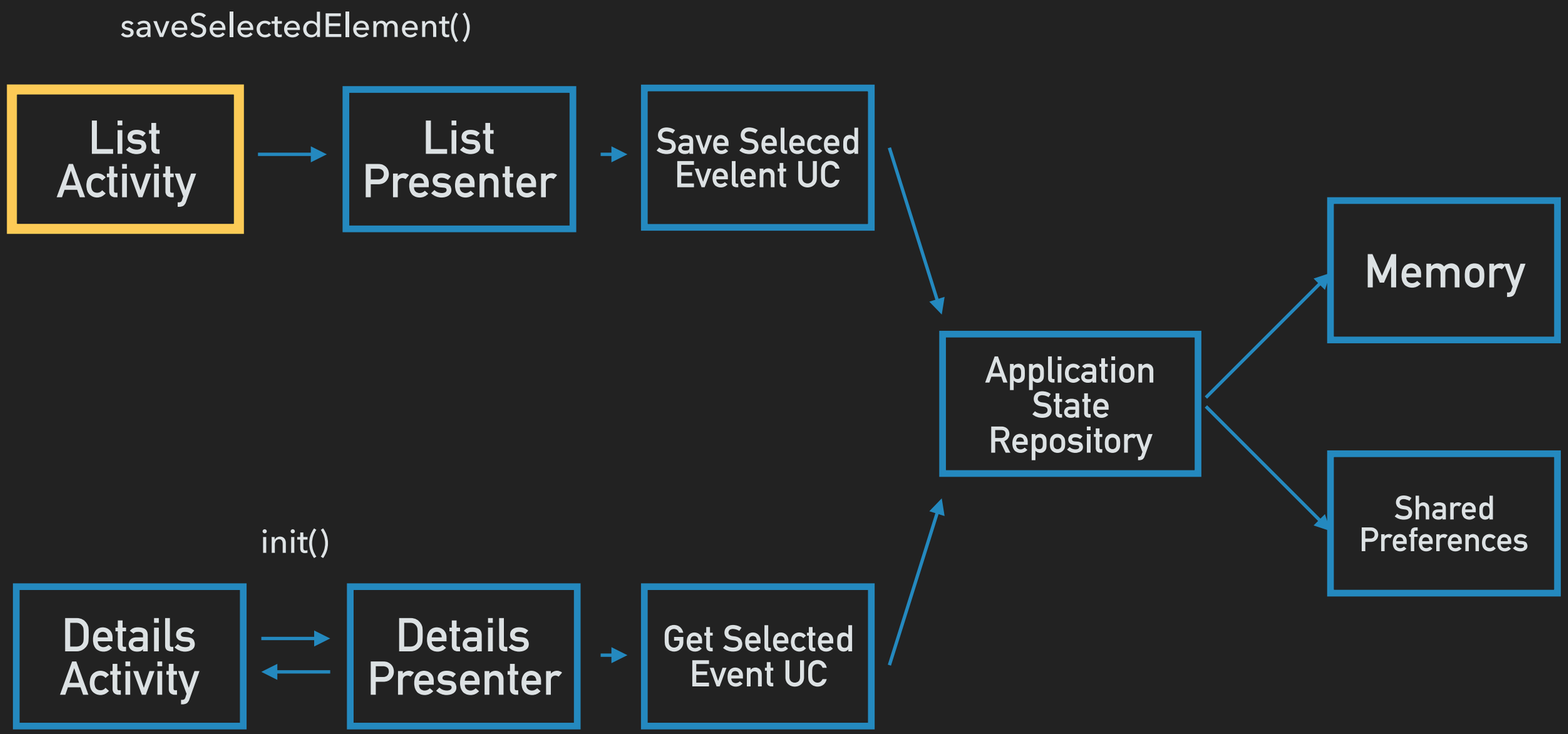
Activity Lifecycle

Stateless Activity

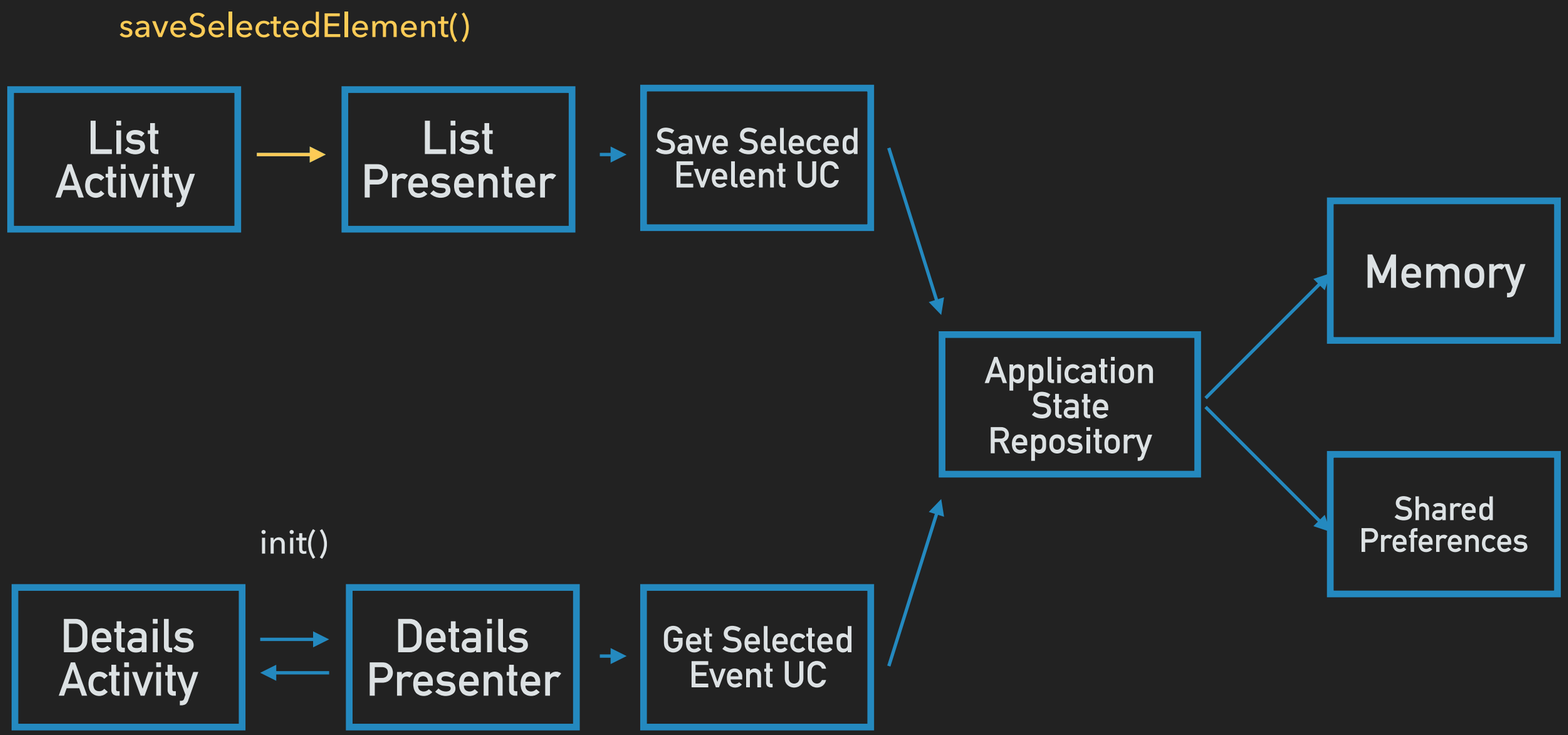
Stateless Activity



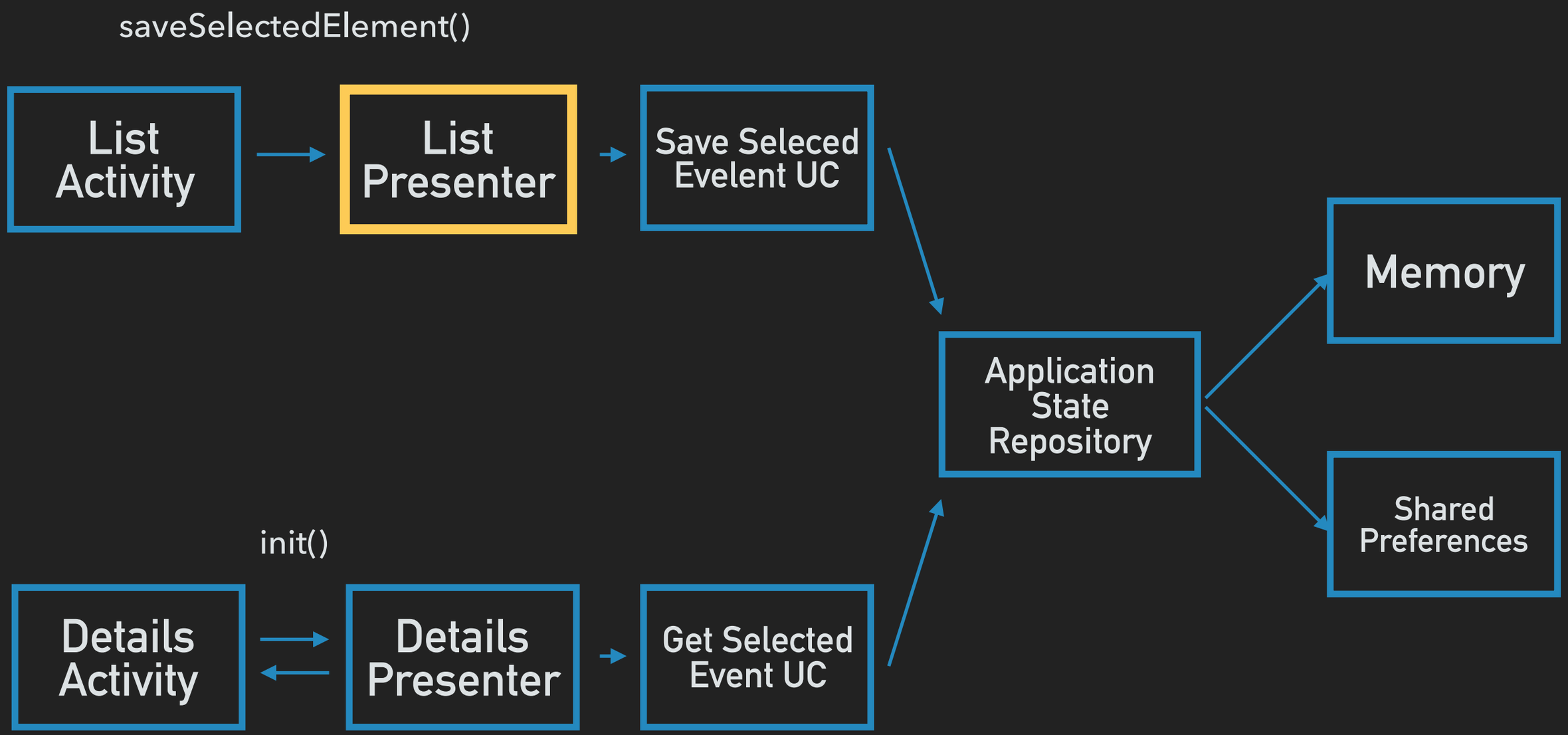
Stateless Activity



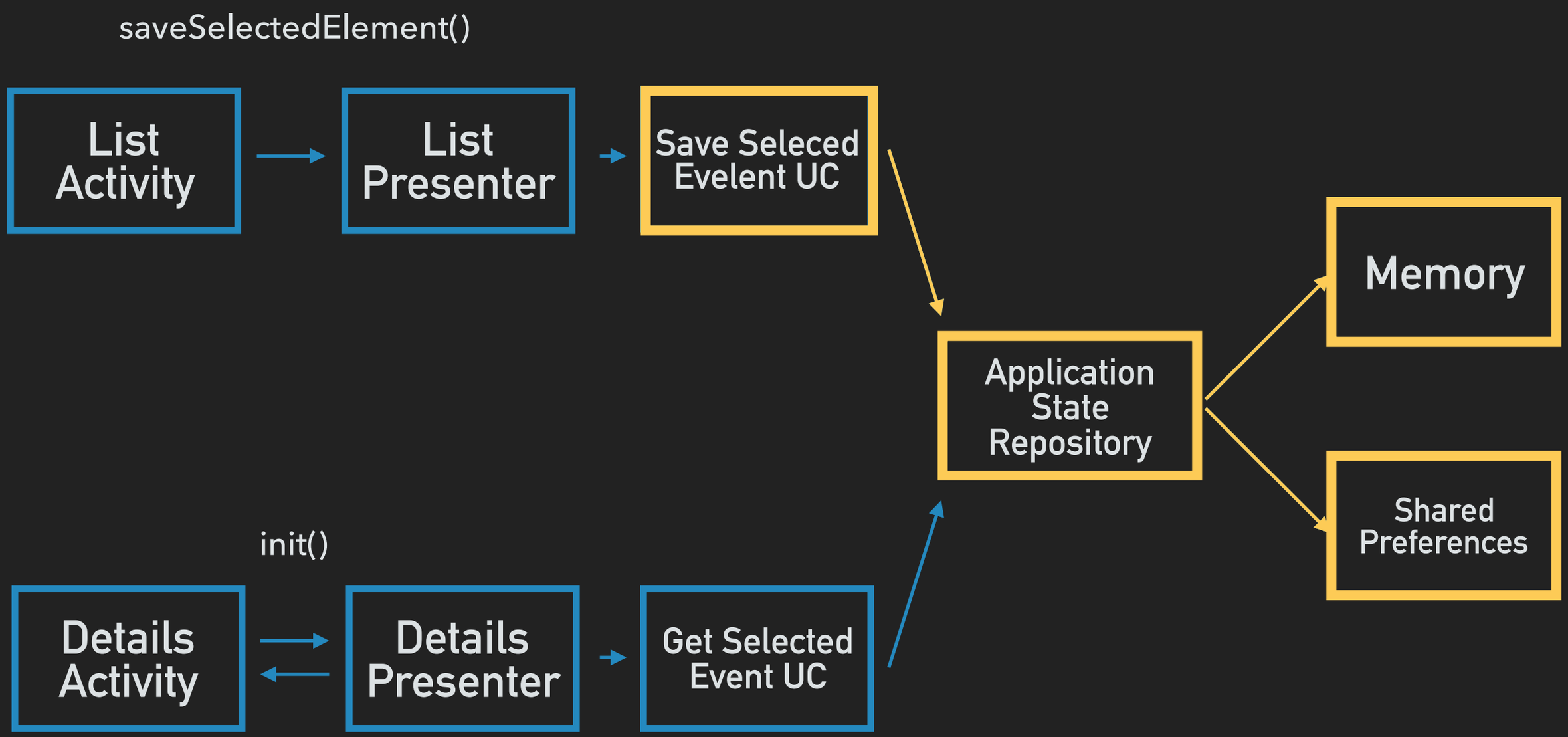
Stateless Activity



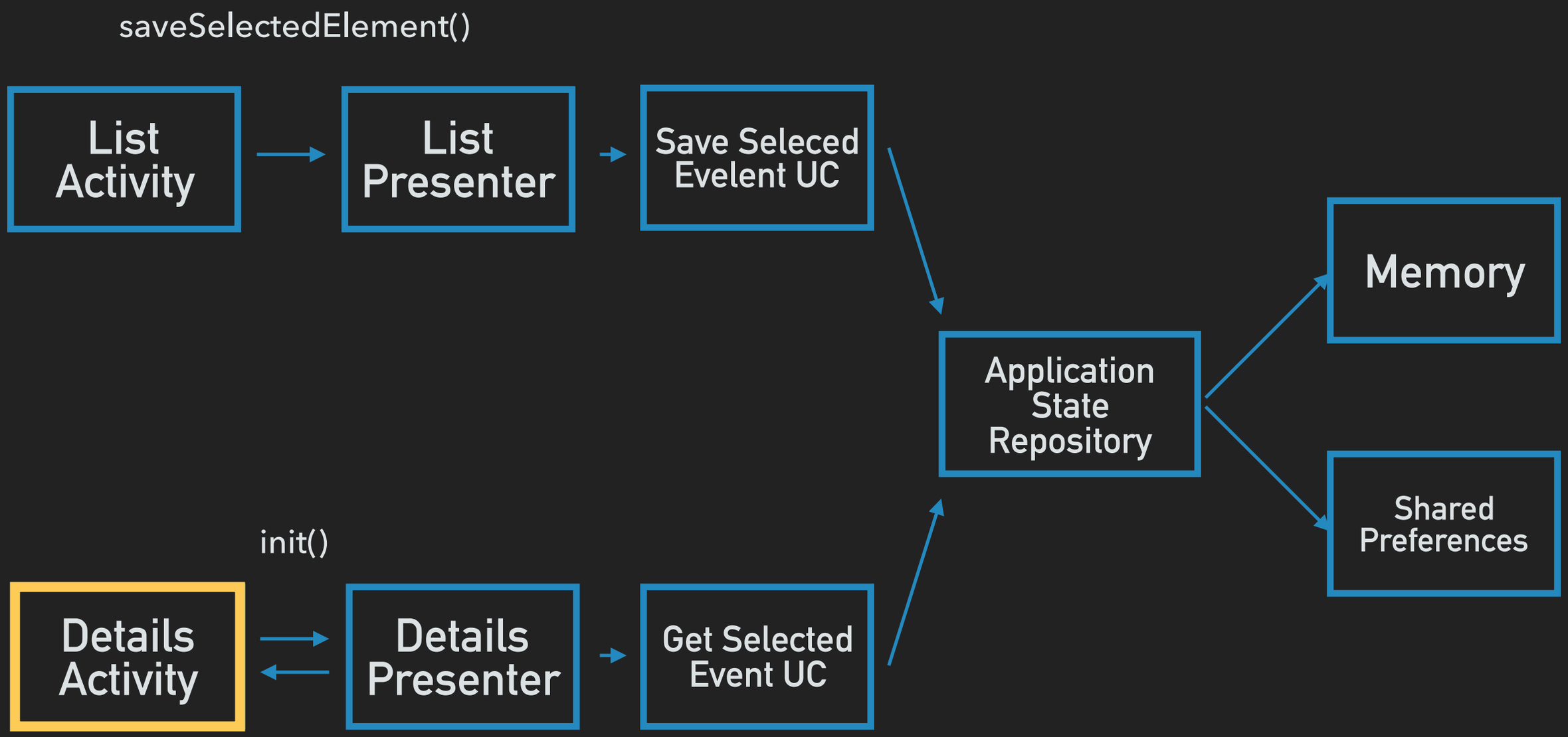
Stateless Activity



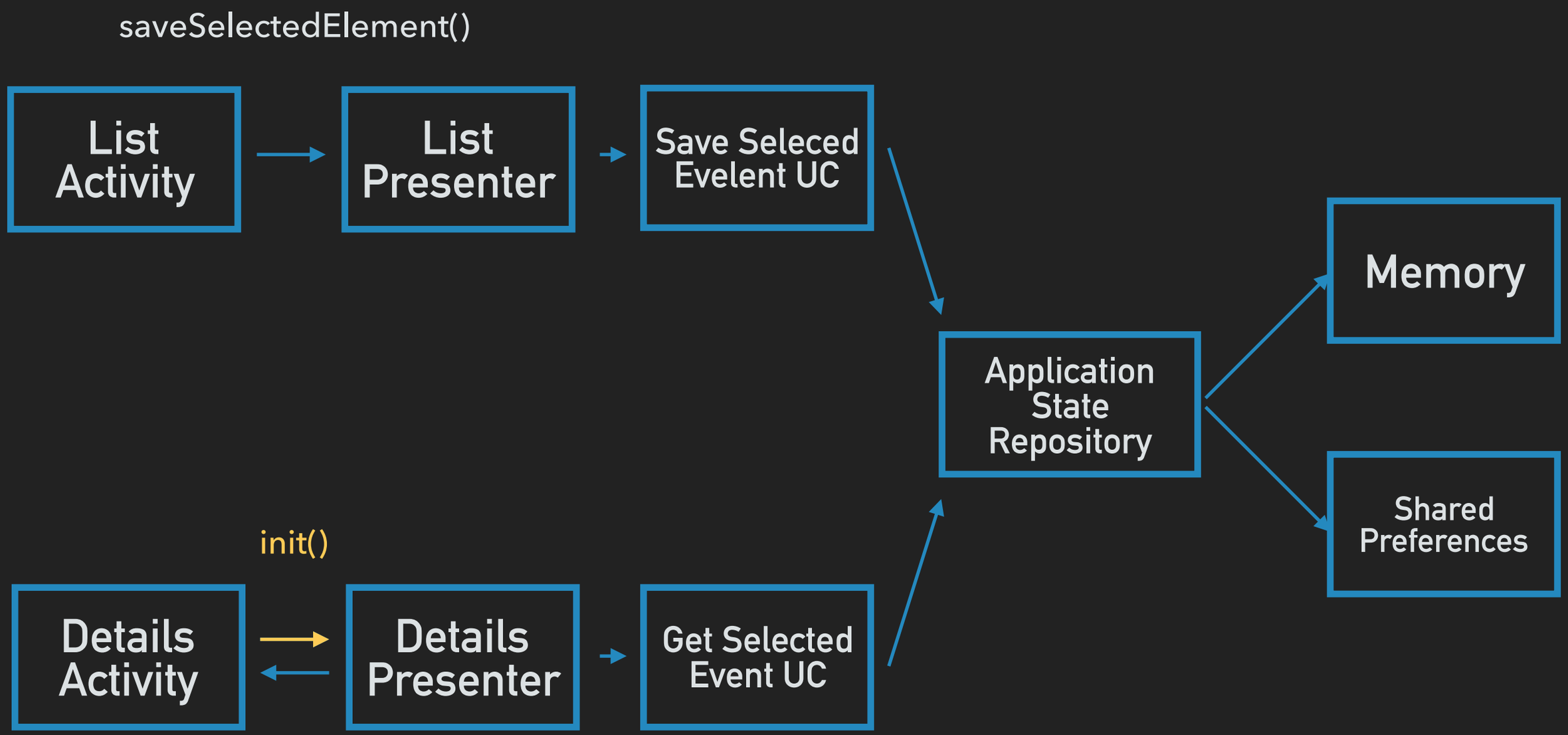
Stateless Activity



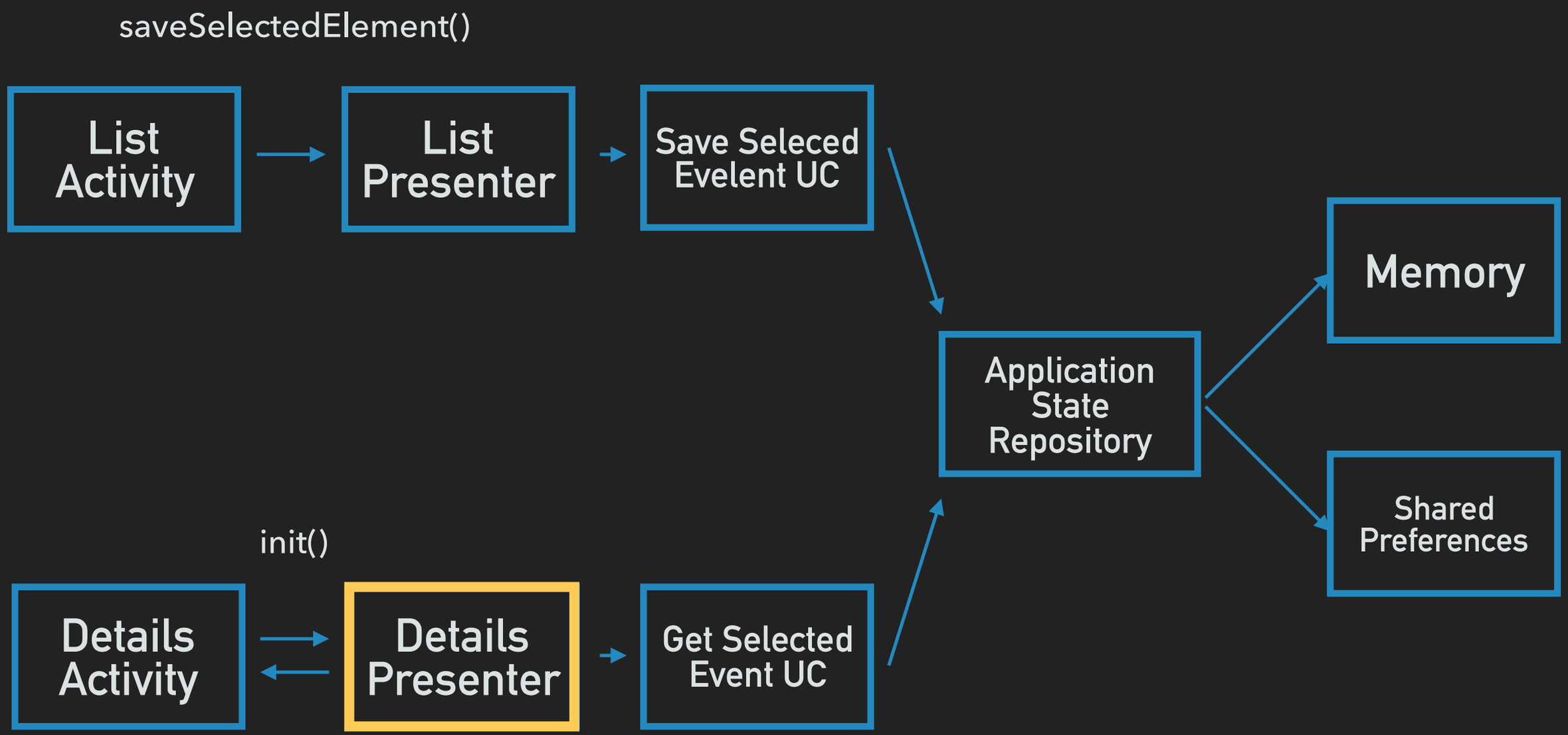
Stateless Activity



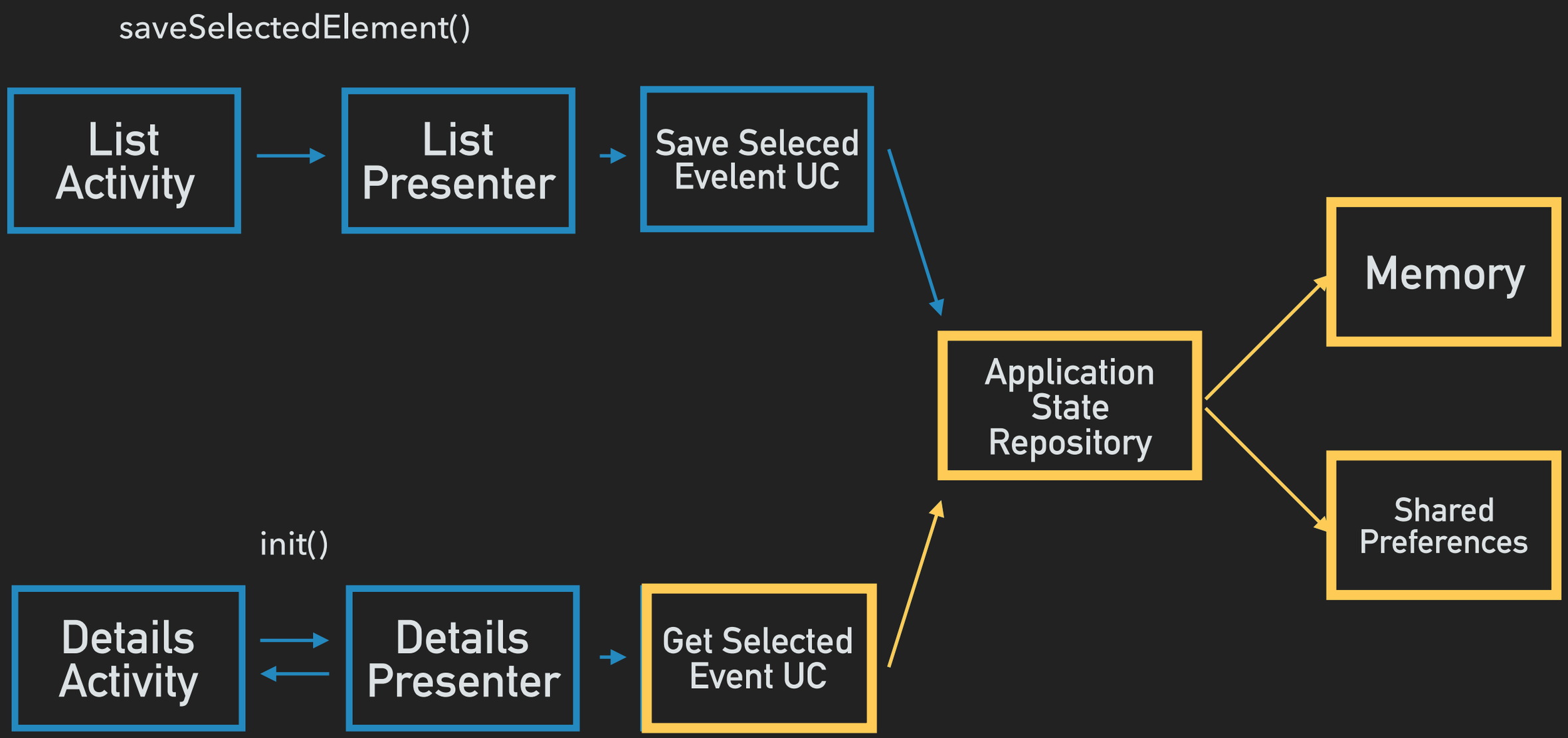
Stateless Activity



Stateless Activity



Stateless Activity



Stateless Activity

- don’t loose View state
- no onSaveInstanceState()
- passive View → SRP

Clean Architecture

Pros

Single Responsibility Principle

Bugs

Test Driven Development

```java
@RunWith(MockitoJUnitRunner.class)
public class GetPostsUseCaseTest {

    @Mock UserPostsRepository repository;

    @InjectMocks GetPostsUseCase useCase;

    @Test
    public void whenExecuteThenCallRepository() {

        when(repository.getPosts())
            .thenReturn(Observable.just(new UserPosts()));

        TestObserver<UserPosts> observer =
                useCase.createUseCaseObservable().test();
        observer.assertComplete();

        verify(repository).getPosts();
        verifyNoMoreInteractions(repository);

    }

}
```

UI Tests

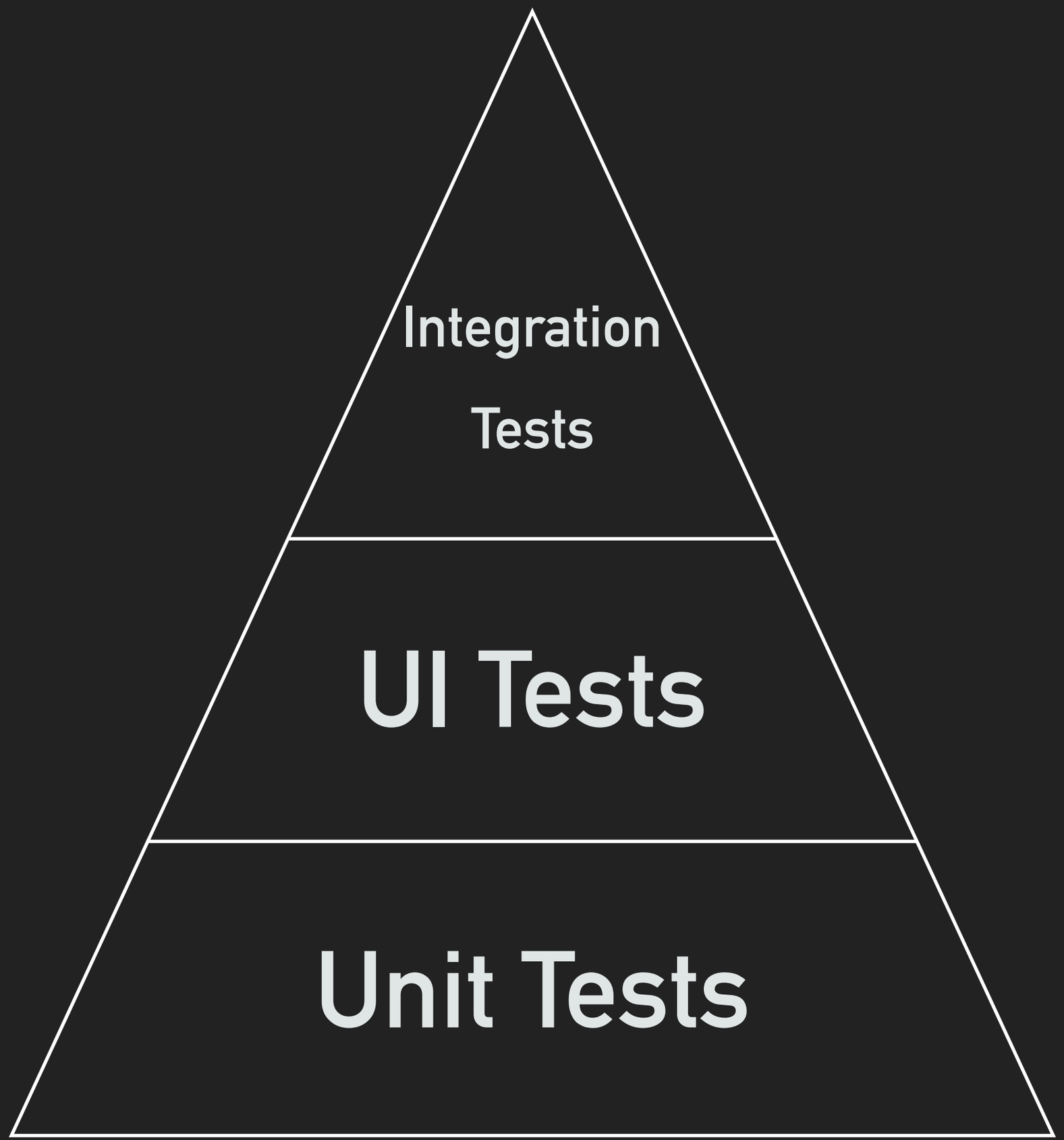
```java
@RunWith(AndroidJUnit4.class)
public class SelectEventsActivityTest {

    @Rule public ActivityTestRule<PostsActivity> rule =
                        new ActivityTestRule<PostsActivity>
                        (PostsActivity.class, true, false);

    @Before
    public void setUp() throws Exception {
        rule.launchActivity(new Intent());
    }

    @Test
    public void showEvents() throws Exception {
        rule.getActivity().runOnUiThread(() -> {
             UserPosts posts = preparePosts("first", "second");
             rule.getActivity().showPosts(posts);
         });

         onView(withText("first")).check(matches(isDisplayed()));
         onView(withText("second")).check(matches(isDisplayed()));

    }
}
```

View Debugging

Integration Tests

Integration Tests

- mock Repository (Api calls)
- simulate user clicks
- verify application behavior


Integration
Tests
UI Tests
Unit Tests

Clean Architecture

- clean code
- SRP → Agile
- reusable code
- async calls and Activity lifecycle
- TDD, UI and Integration Tests
- increase of development speed

Q & A

arek.macudzinski@gmail.com

Last thing

Use Architecture, SRP and TDD