Using ViewModel with the LiveData Framework

Omri Erez

SOFTWARE ENGINEER

@innovationMaze | https://www.linkedin.com/in/omrierez/

MVVM and the View Model

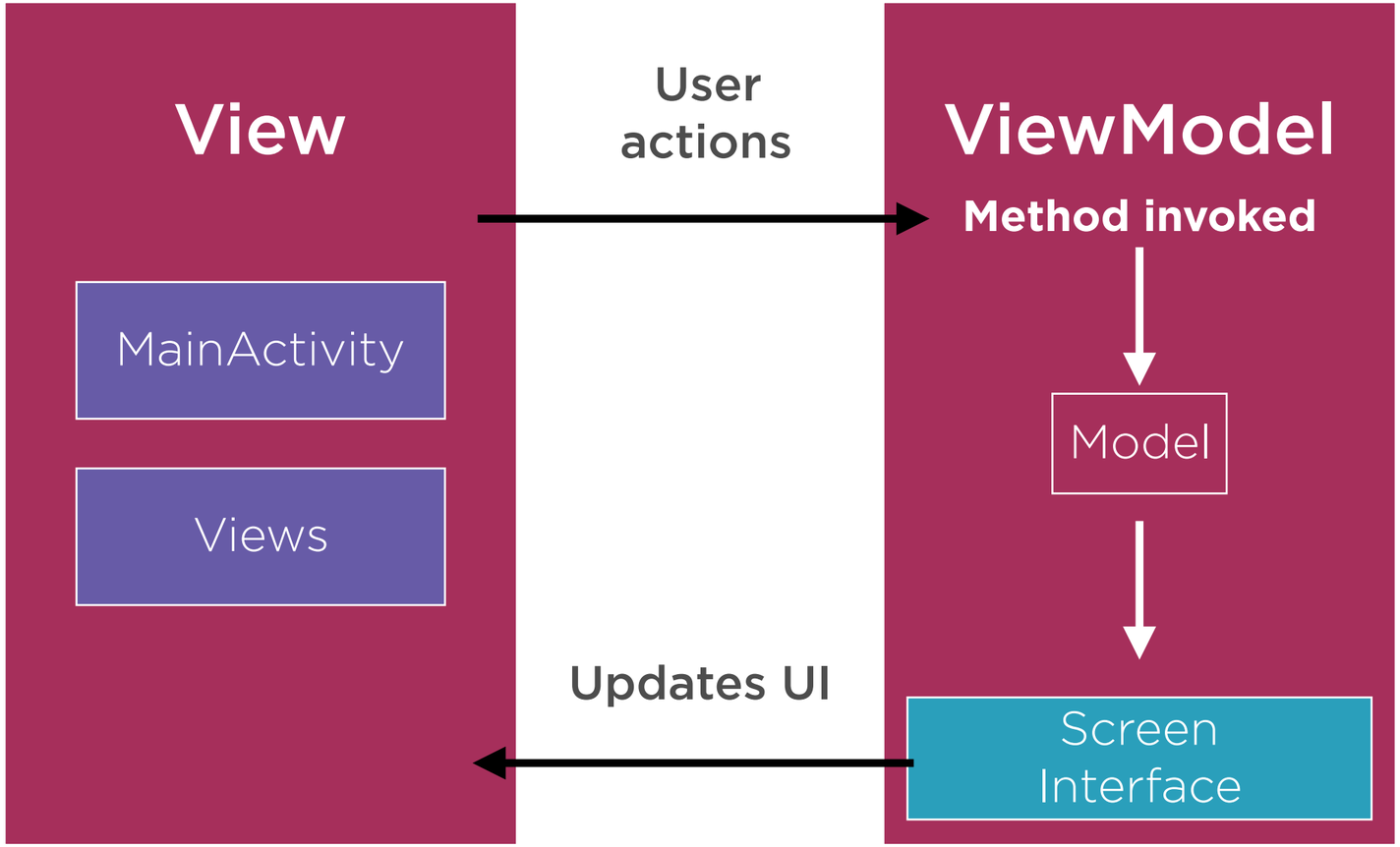
MVVM

- Includes a ViewModel component

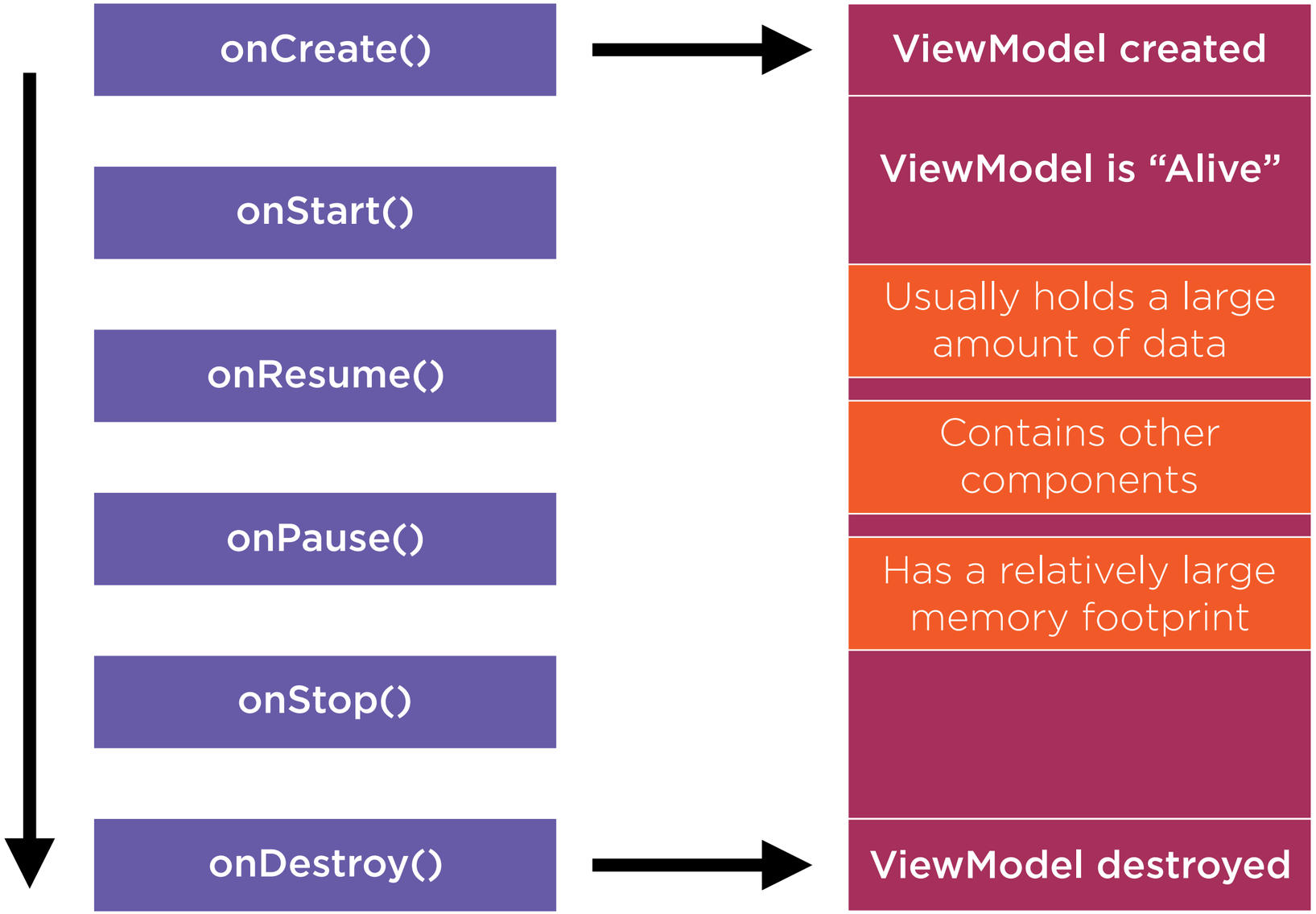
- Self implemented without the data binding support library

- The activity will be in-charge of updating the views

Our MVVM Implementation



Simplified Activity Life Cycle



- On configuration changes, the activity is destroyed are recreated by the OS

- There is a native mechanism for saving the data state using:
  - onSaveInstanceState()
  - onRestoreInstanceState()
  - Doesn’t meant to be used with large data sets

- On activity recreation, our ViewModel is recreated

- It’s wasteful in terms of memory

Solution
android.arch.lifecycle.ViewModel

- Initiated with a scope, initiated once

- the ViewModel instance will survive configuration changes such as screen rotations

- After activity’s recreation, it will use the old ViewModel instance

- UI related data will be saved

- the ViewModel will be bound to the lifecycle scope passed to it

Demo

Crypto Boom App

- Create a ViewModel without ARCH
- Examine ViewModel recreation on configuration changes
- Implement arch.lifecycle.ViewModel
- Compare

Is there a different way of implementing the same thing but in a more memory efficient way?

Solution
android.arch.lifecycle.ViewModel

Share Data with a Fragment

```
public static class Fragment extends android.support.v4.app.Fragment
{
    private CryptoViewModel mViewModel;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mViewModel =
        ViewModelProviders.of(this).get(CryptoViewModel.class);
    }
}
```

LiveData

- Android apps often need to fetch data asynchronously

- In our demo app: fetching the cryptocurrency market data

- We need to handle resources manually

- Involves maintenance code to clear resources

- Our components need to to be life cycle aware

- Resources are newly allocated on configuration changes

Solution
android.arch.lifecycle.LiveData

- It’s an observable data holder

- Utilizes the Observer pattern

- It’s a life cycle aware component

- No more manual life cycle handling

- The data is always up to date

- Sharing resources and not wasteful to memory

- Like a light RxJava implementation

- Survives configuration changes

- Only active in foreground life cycle states

LiveData **allow async data** access and **decouples** the access to our data from the component who consume it.

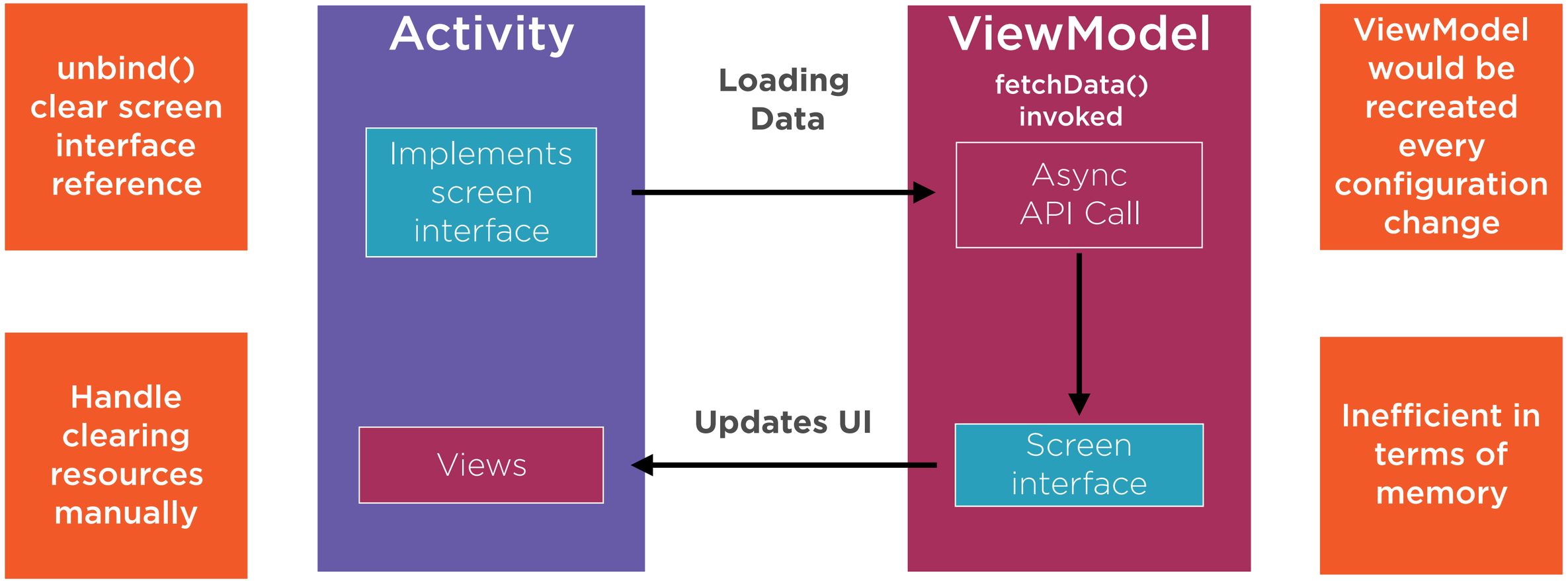
Demo

Crypto Boom App

- Wrap our data fetching with a LiveData object
- Add an update interval logic
- Use the Transformations to map our data
- Use our ViewModel and LiveData objects to share data between our activity and a fragment

Comparison

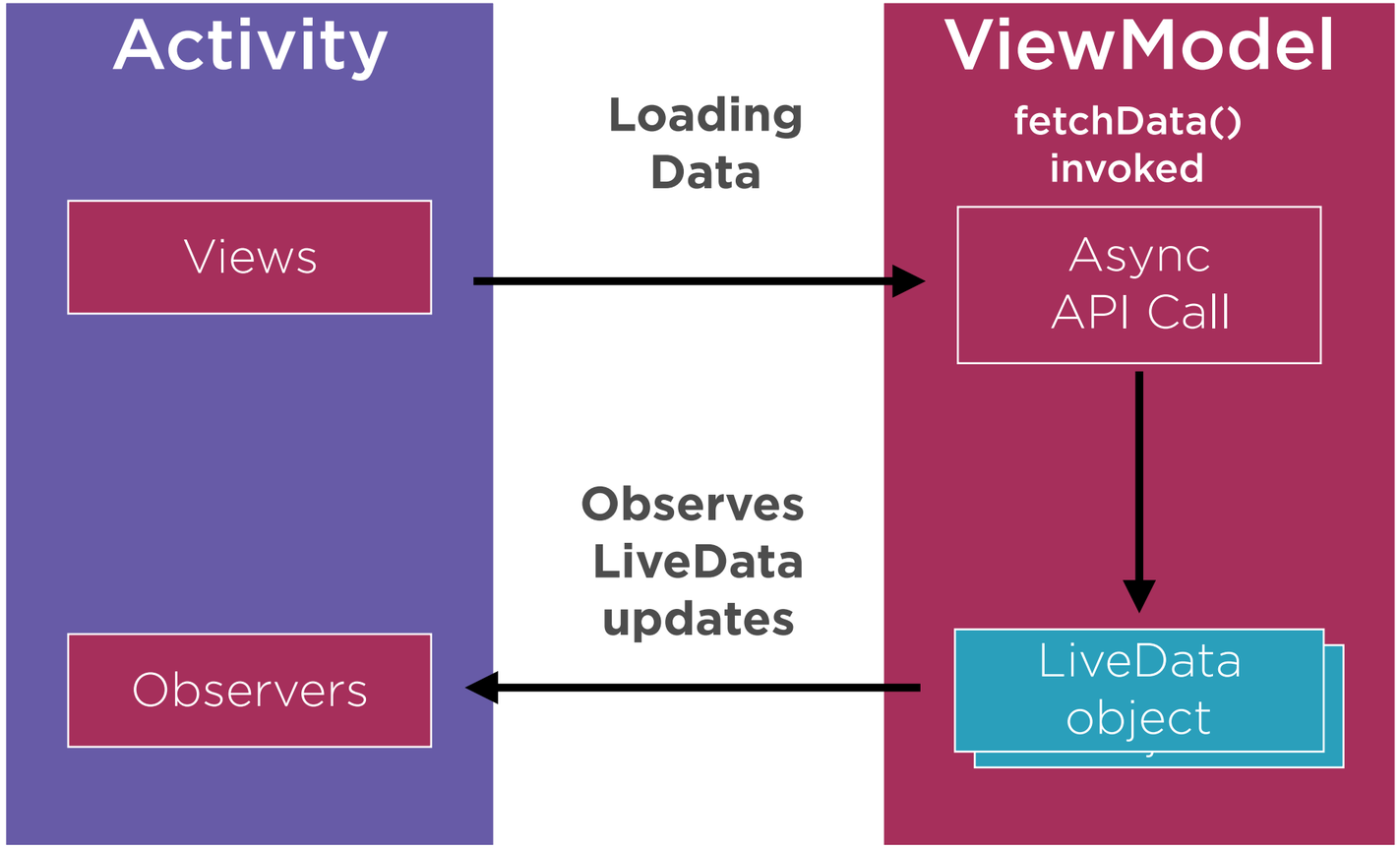
Architecture Before



Architecture After

No need for unbinding code

Life cycle aware component



Doesn't contain a reference to our activity

Efficient in terms of memory

Summary

We changed our app's architecture by introducing and implementing:

- arch.lifecycle.ViewModel
- arch.lifecycle.LiveData

Summary

ViewModel

- It’s a lifecycle aware component
- Created only once per lifecycle owner (Our MainActivity)
- Contains our data access logic and other application related logic

Summary

ViewModel's Advantages

- Survives configuration changes
- Efficient in terms of memory
- Sharable between components

Summary

LiveData

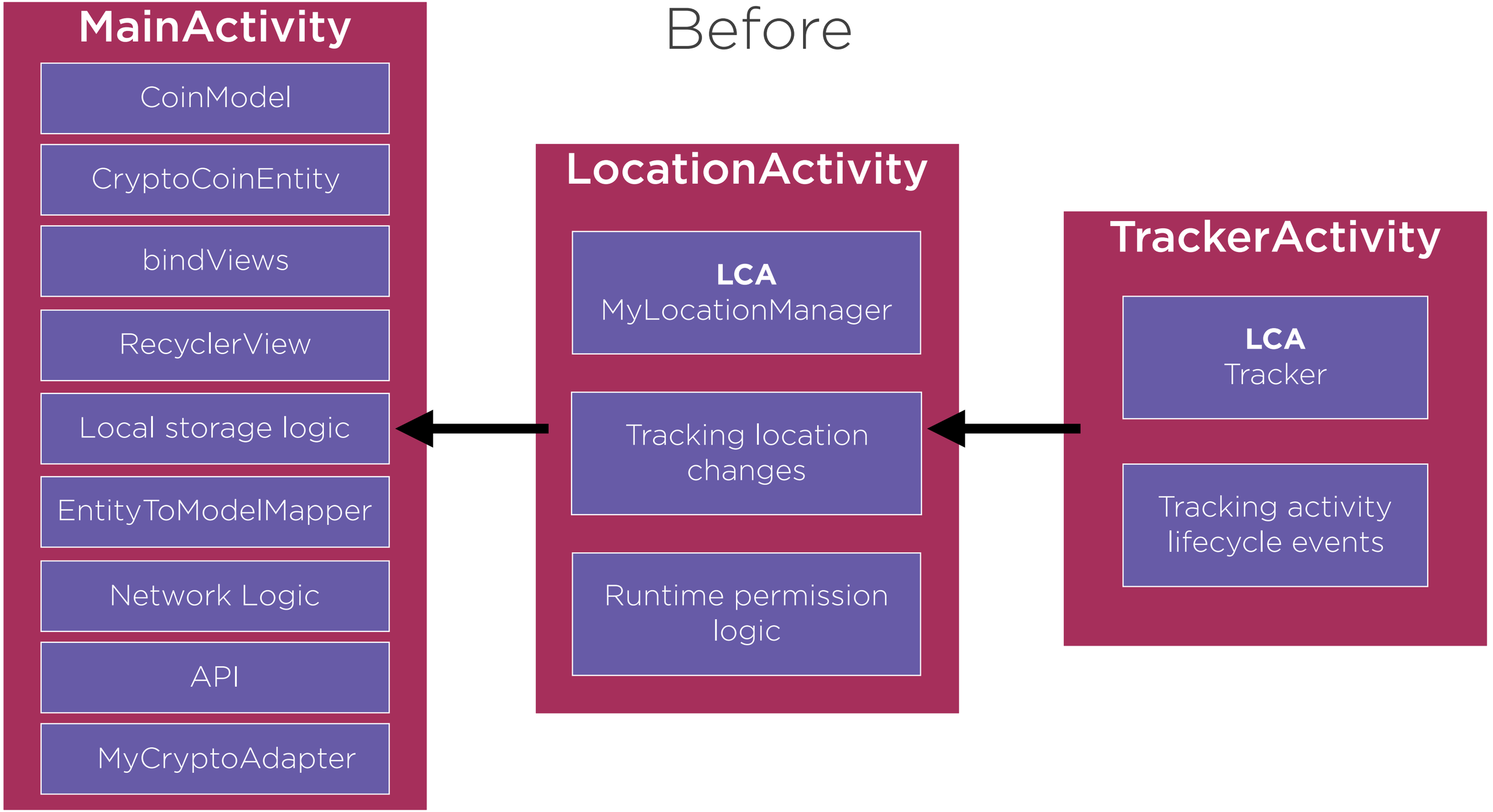
- It’s a lifecycle aware component
- Get data updates via observers
- How to update LiveData objects from different threads
- How to share data between multiple components

Summary

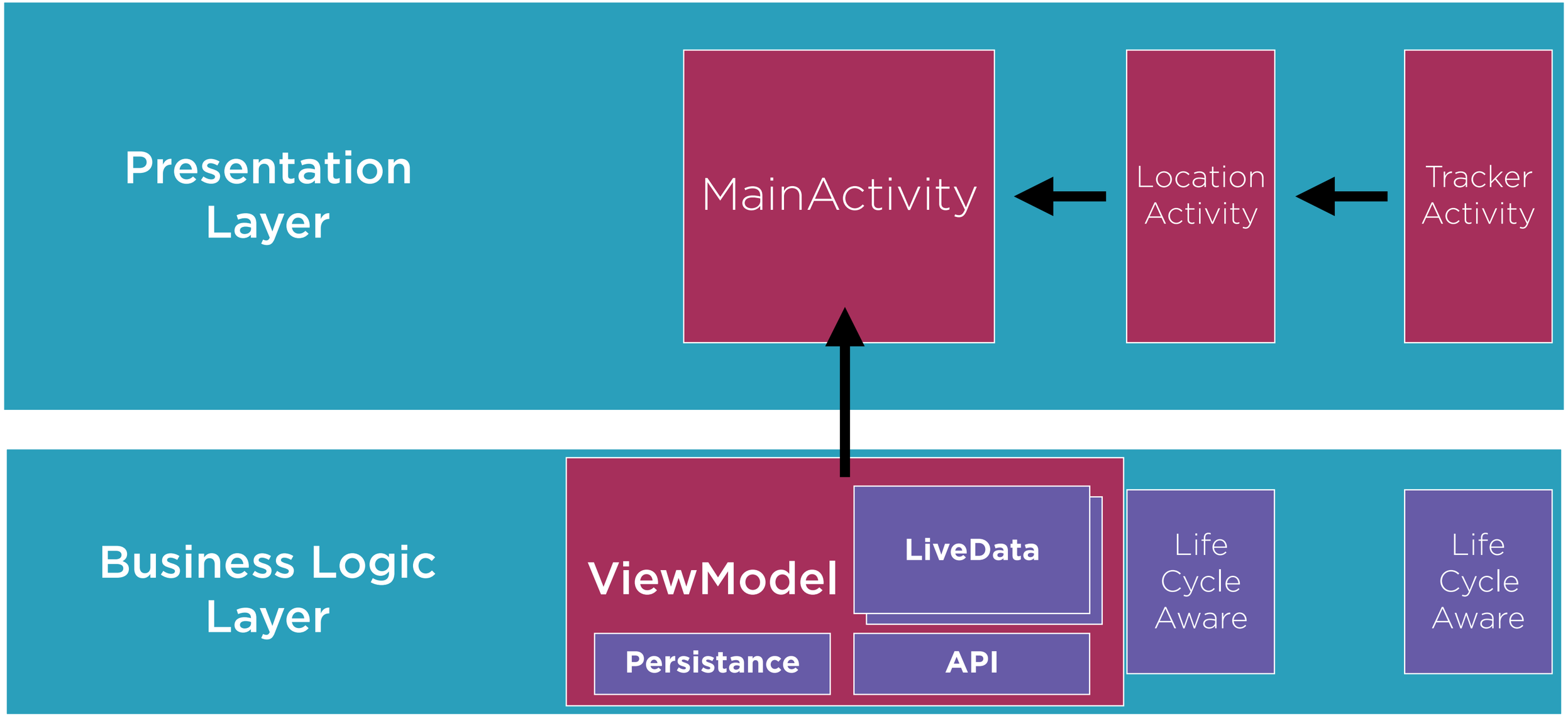
LiveData's Advantages

- It's a lifecycle aware component
- Decouples data access logic from the one who consumes it
- Bound to a life cycle owner

Before


MainActivity
CoinModel
CryptoCoinEntity
bindViews
RecyclerView
Local storage logic
EntityToModelMapper
Network Logic
API
MyCryptoAdapter
LocationActivity
LCA
MyLocationManager
Tracking location changes
Runtime permission logic
TrackerActivity
LCA
Tracker
Tracking activity lifecycle events

After


Presentation Layer
MainActivity
Location Activity
Tracker Activity
Business Logic Layer
ViewModel
LiveData
Persistance
API
Life Cycle Aware
Life Cycle Aware