Engineering Wunderlist for Android

Cesar Valiente

+CesarValiente

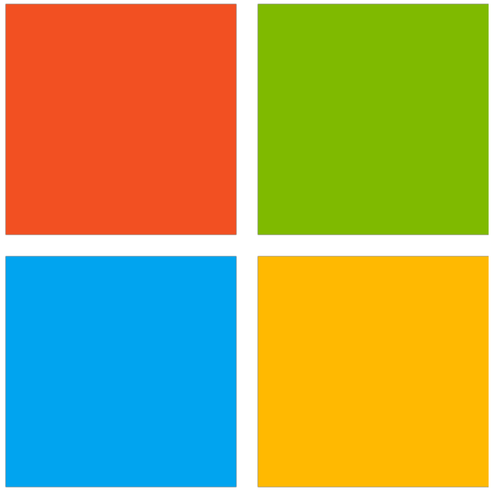
wunderkinder

@CesarValiente

Wunderlist, 6Wunderkinder and me

Together we'll be better!!

+

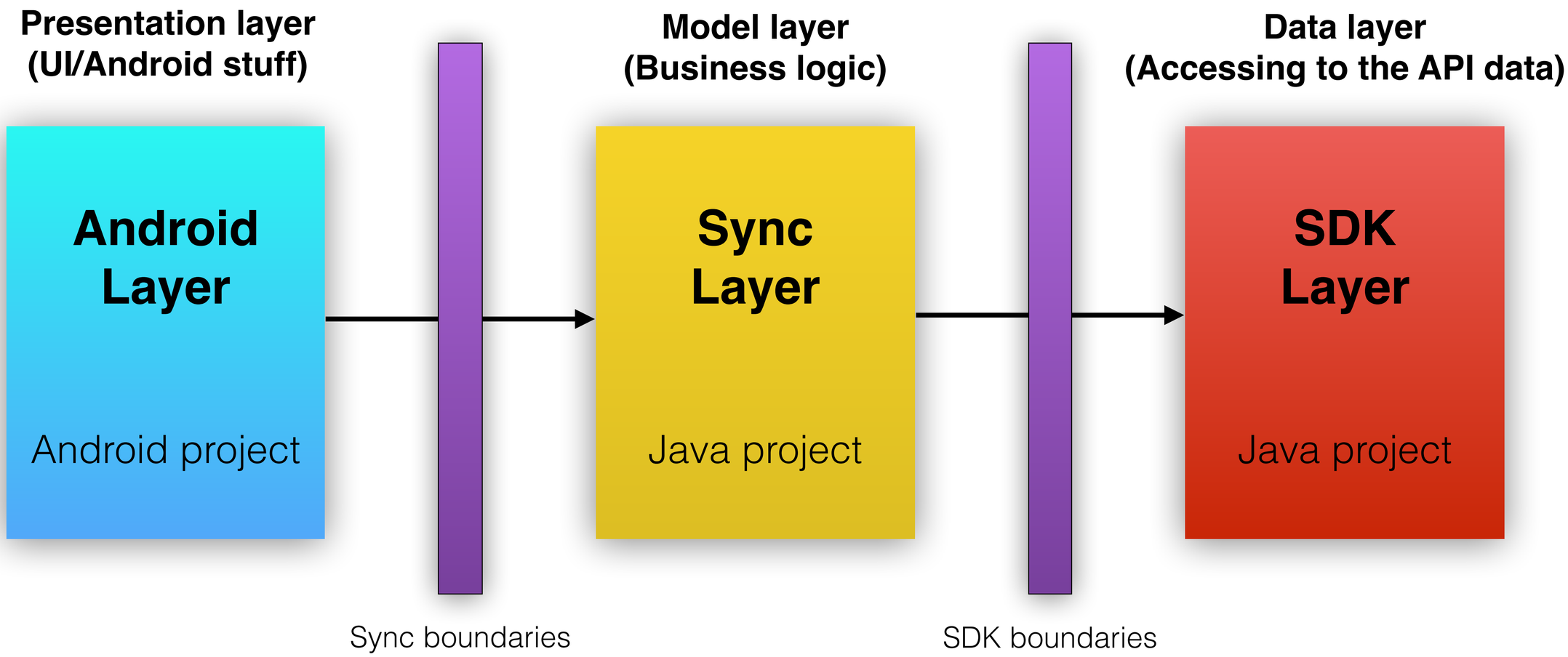


First of all...

- **Wunderlist 2** was created as a monolithic app.
- **Wunderlist 3** (from now on, just Wunderlist) has been built to be *highly independent* between layers.
- *Real time sync*.

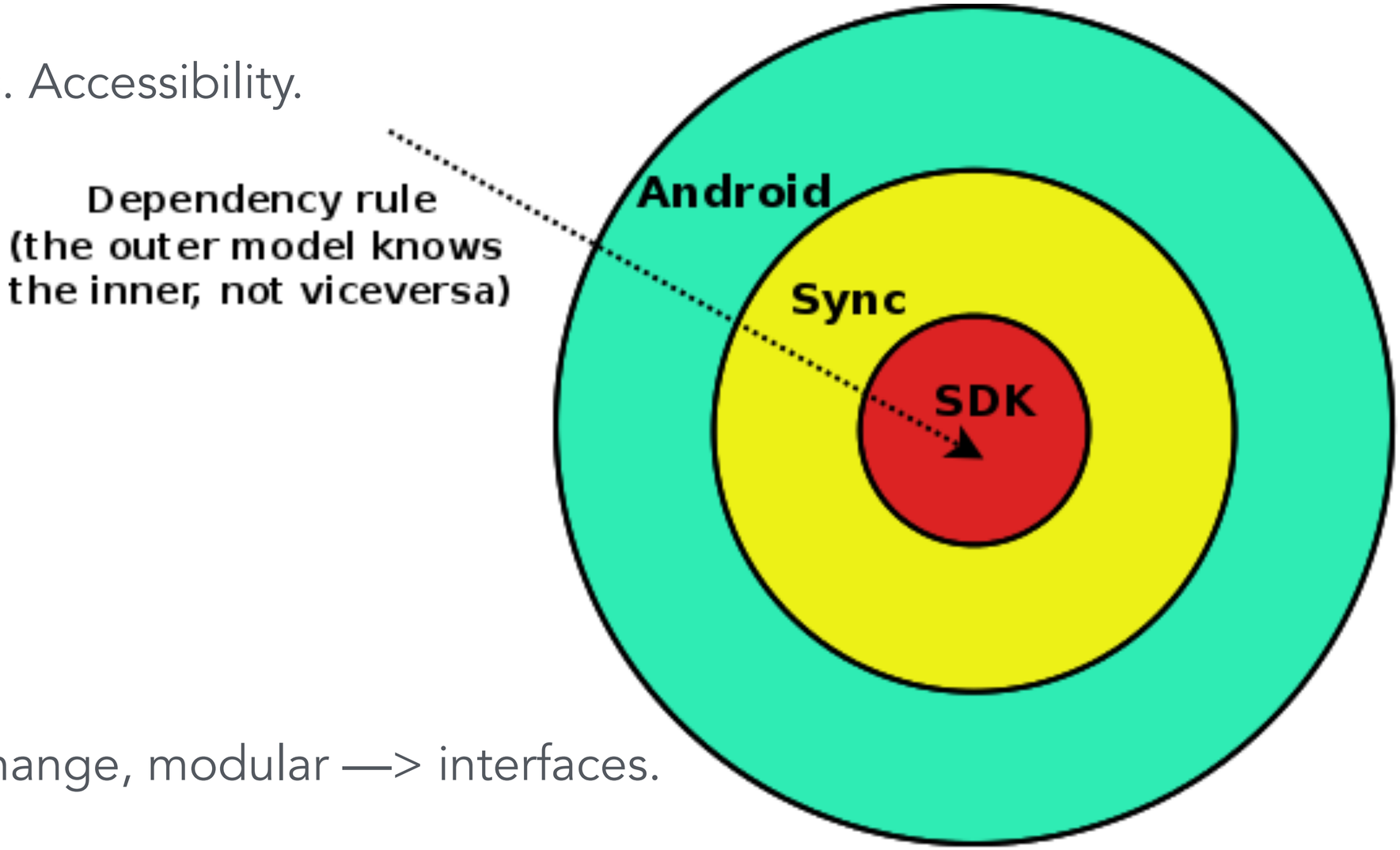
1. General architecture

1.2 Three layers



1.3 What does it mean?

- Independent layers. Accessibility.



- Clean architecture.
- Abstract, easy to change, modular —> interfaces.

```
package com.wunderlist.sdk;
```

2. Sdk

Network

- Websocket (real time).
- REST.

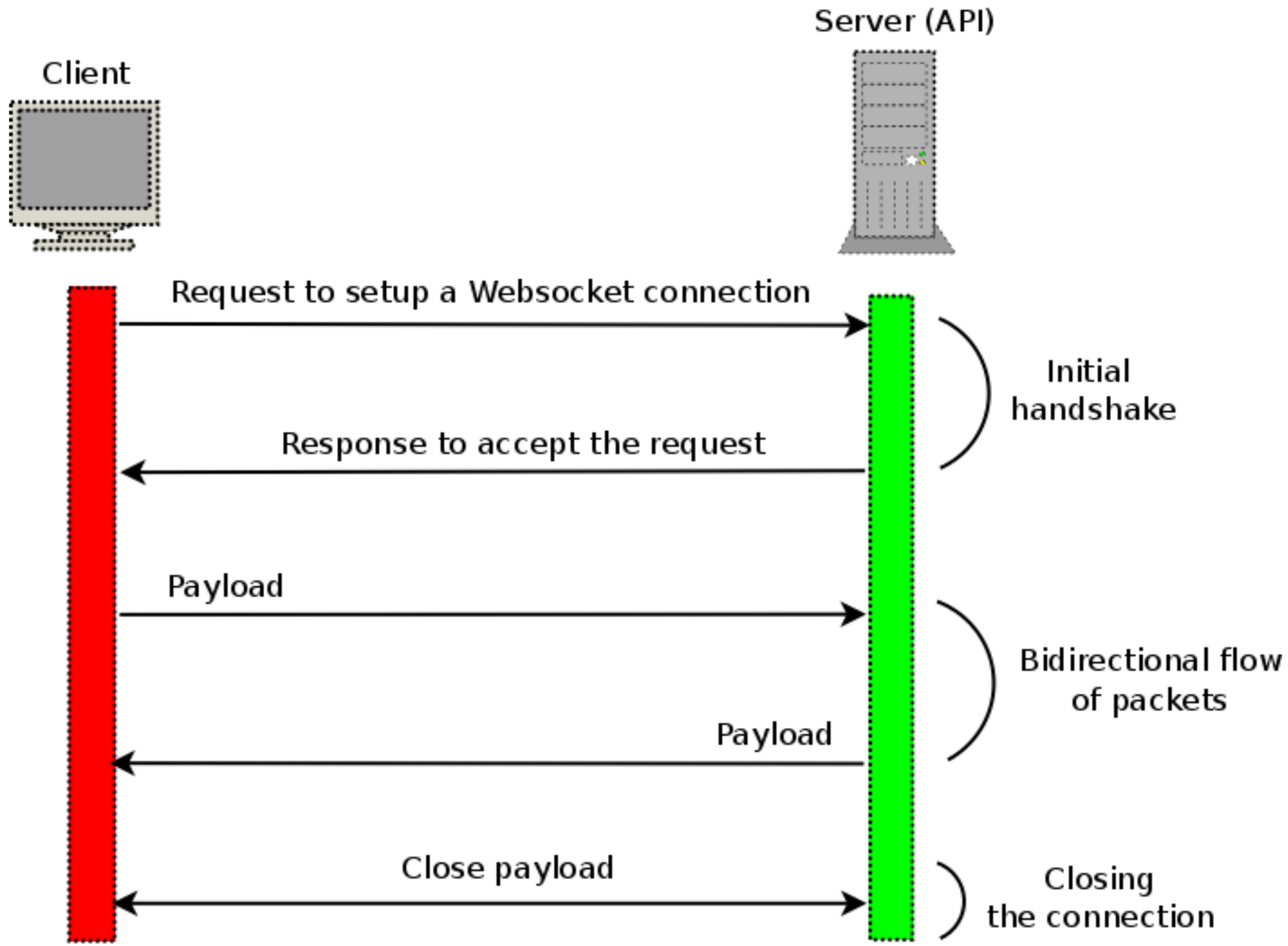
API model

- Services
- API data models.
- Serializers/deserializers.
- Interfaces/callbacks.
- Sync and async tests.

2.1 Websocket (real time)

" WebSocket is a protocol providing full-duplex communications channels over a single TCP connection. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011, and the WebSocket API in Web IDL is being standardized by the W3C."

2.1.1 Websocket (real time)



2.1.2 Websocket. Details

- Main way to sync.
- java-websocket, netty,…. OkHttp ws!

2.1.3 Websocket. Mutations

```
void handleMessage(String message) {
  try {
     JsonObject jsonObject = Json.parseJson(message);
     if (HealthResponse.isHealthResponse(jsonObject)) {
        ……
        } else if (Response.isValidJsonForObject(jsonObject)) {
        ……
        } else if (Mutation.isValidJsonForObject(jsonObject)) {
        receiveMutation((Mutation)Json.fromJson(
           jsonObject, Mutation.class));
        } else {
        Log.warn("Oh oh! We got a corrupted object on websocket: "
           + message);
        }
      } catch (JsonSyntaxException e) {
           Log.warn("Discarding a malformed json object on websocket: "
           + message);
      }
}
```

2.1.4 Websocket. Real-time example

2.2 REST

- Conventional REST API services for synchronous requests.
- Secondary (fallback) way to sync.
- Using Square OkHttp.
- To encourage hunting bugs and API stress tests, we always use production API.

2.3 Others

- **Services**, manages an API endpoint (Users, Lists, Tasks, etc.)
- **API data models**, same structure as API scheme data objects (basic models, just data).
- **Json serializers/deserializers**. Using Gson. Integrity (health) checks.
- Callbacks.
- Sync/Async unit **tests** (Junit and mockito) to test requests/responses.

```
package com.wunderlist.sync;
```

3. Sync

This layer manages all business logic of Wunderlist, where we make all sync operations.

Data

- Data models.
- Deserializers from the basic model.
- Cache.

- Services that manage data.

Sync logic

- Matryoshka (aka Russian doll).
- Conflict resolver (included in Matryoshka).

3.1 Data models

Sync data models in this layer manage the raw data that we already have from the SDK data models.

```
public class WLUser extends WLApiObject<User> {

    public static Type collectionType =
            new TypeToken<List<WLUser>>() {}.getType();

    @Persist
    private boolean isMe;
    private String pictureUrl;

    private WLUser(User user) {
        super(user);
    }

    public static WLUser buildFromBaseModel(User user) {
        return new WLUser(user);
    }

    public static WLUser buildFromEmail(String email) {
            ..............
    }

    ..........................................
}
```

3.2 Deserializers

- Deserializers make the parsing stuff from the raw API model, to the model we are going to use within the app.
- Our deserializers are used by Gson to work with the correct scheme.

```
public class WLTaskDeserializer
    extends WLApiObjectDeserializer<Task, WLTask> {

    @Override
    protected Type getType() {
        return Task.class;
    }

    @Override
    protected WLTask newInstance(Task baseModel) {
        return new WLTask(baseModel);
    }
}
```

3.3 Cache

- Caches for different models (Tasks, Lists, Memberships, etc.)
- DataStore interface which our database model (in Android layer) and Cache implement to work on the same way.
- Cache has to be filled in a background thread (not UI).
- Two dynamic data structures, List and Map.

3.4 Services

- Used to communicate the Android layer with the SDK (AppDataController, retrieves API data).

```
public class WLTaskService extends WLService<WLTask, TaskService> {

    public WLTaskService(Client client) {
        super(new TaskService(client));
    }
    ……….

    public void getCompletedTasks(final WLList list, final SyncCallback uiCallbacks) {
        ResponseCallback responseCallback = new ResponseCallback() {
            @Override
            public void onSuccess(Response response) {
                ……….
                uiCallbacks.onSuccess(tasks);
            }
            @Override
            public void onFailure(Response response) {
                ……….
                uiCallbacks.onFailure(response);
            }
        };
        service.getCompletedTasksForList(list.getId(), responseCallback);
    }
    ……….
}
```

3.5. Matryoshka (aka Russian Doll)

3.5.1 What is this?

- Set of wooden dolls of decreasing size placed one inside the other.

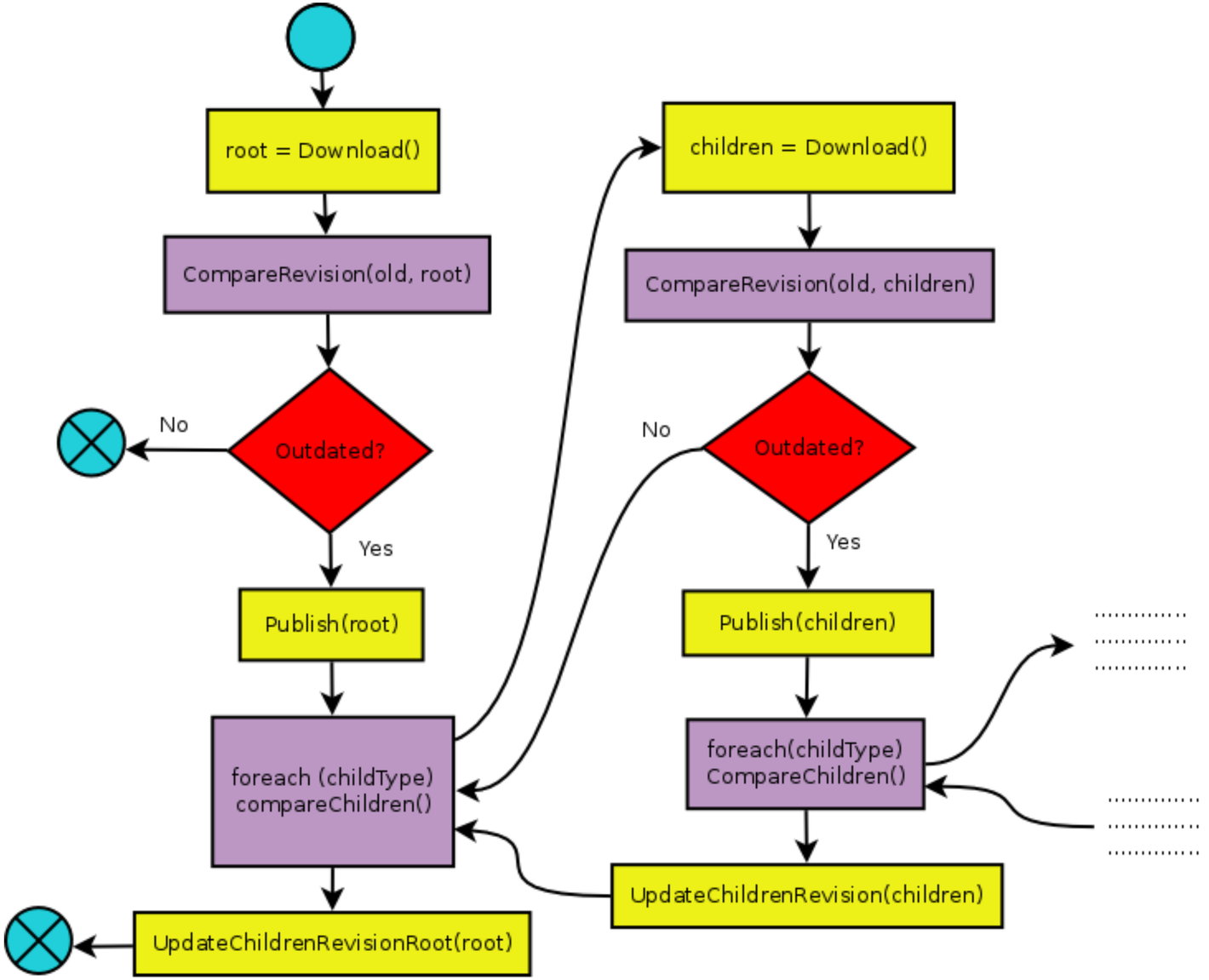
- Mechanism to **properly sync** entire model.

3.5.2 How does it work?

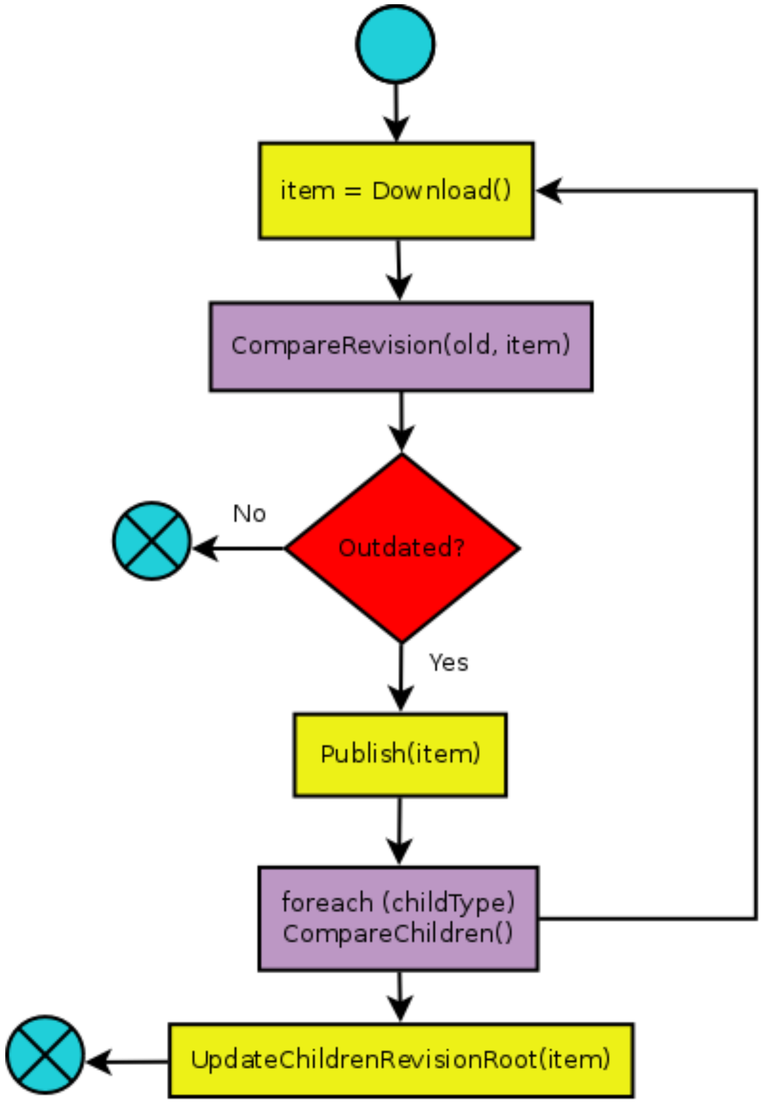
- Revision based.

- When a child is updated the **revision changes the whole tree up** (eg.: when a **Task** is created the **List's** revision is incremented, as well the **root**).

3.5.2.1 How does it work?



3.5.2.2 How does it work?



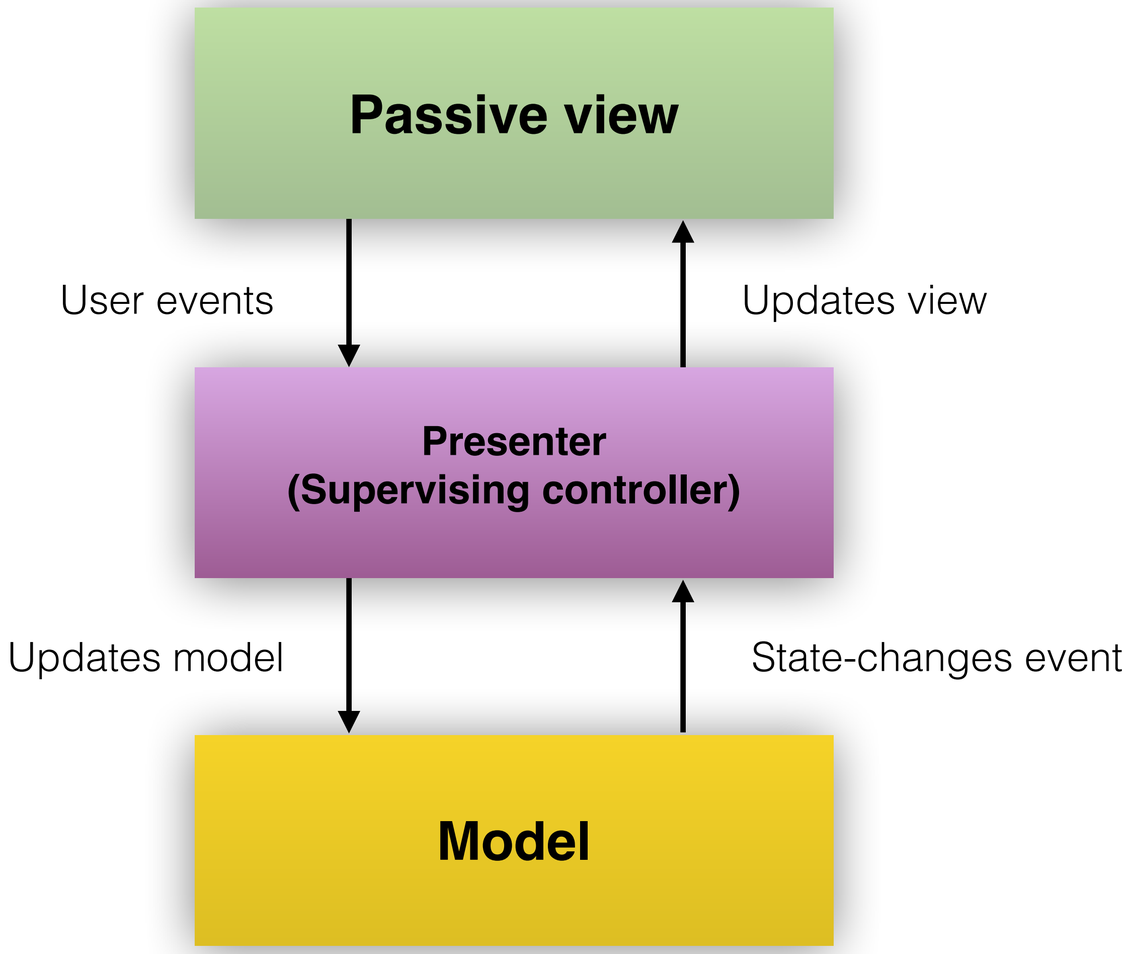
Yay!!! is like a Matryoshka!!!
so… is recursive ;-)

```
package com.wunderlist.wunderlistandroid;
```

4. Android layer

- This is the UI + Android libs and code.
- Maximized **decoupling** among **UI** and **business logic.**

4.1 Model View Presenter

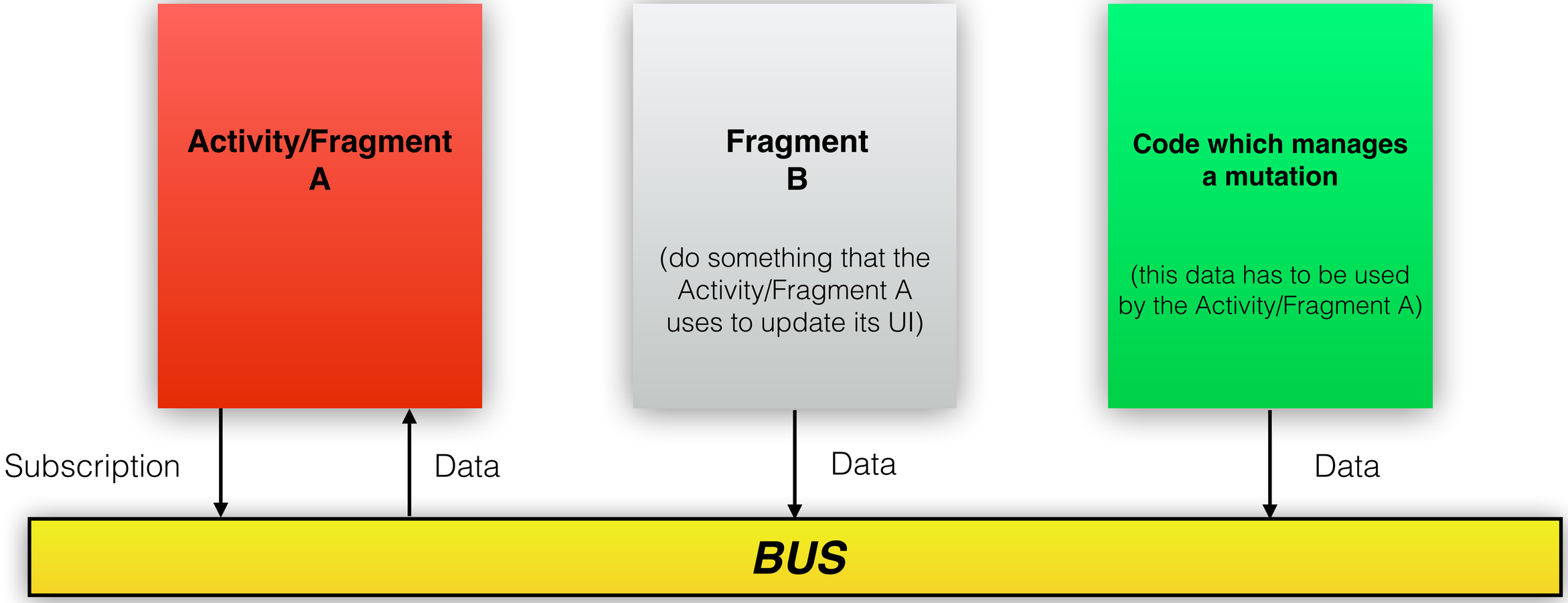


4.2 EventBus

EventBus is an Android optimised publish/subscribe event bus. A typical use case for Android apps is gluing Activities, Fragments, and background threads together.

- We use **EventBus** to update the UI when we have loaded asynchronously our requested data…
- … but also when something that we are listen for happens, like a mutation.

4.2.1 EventBus (example)

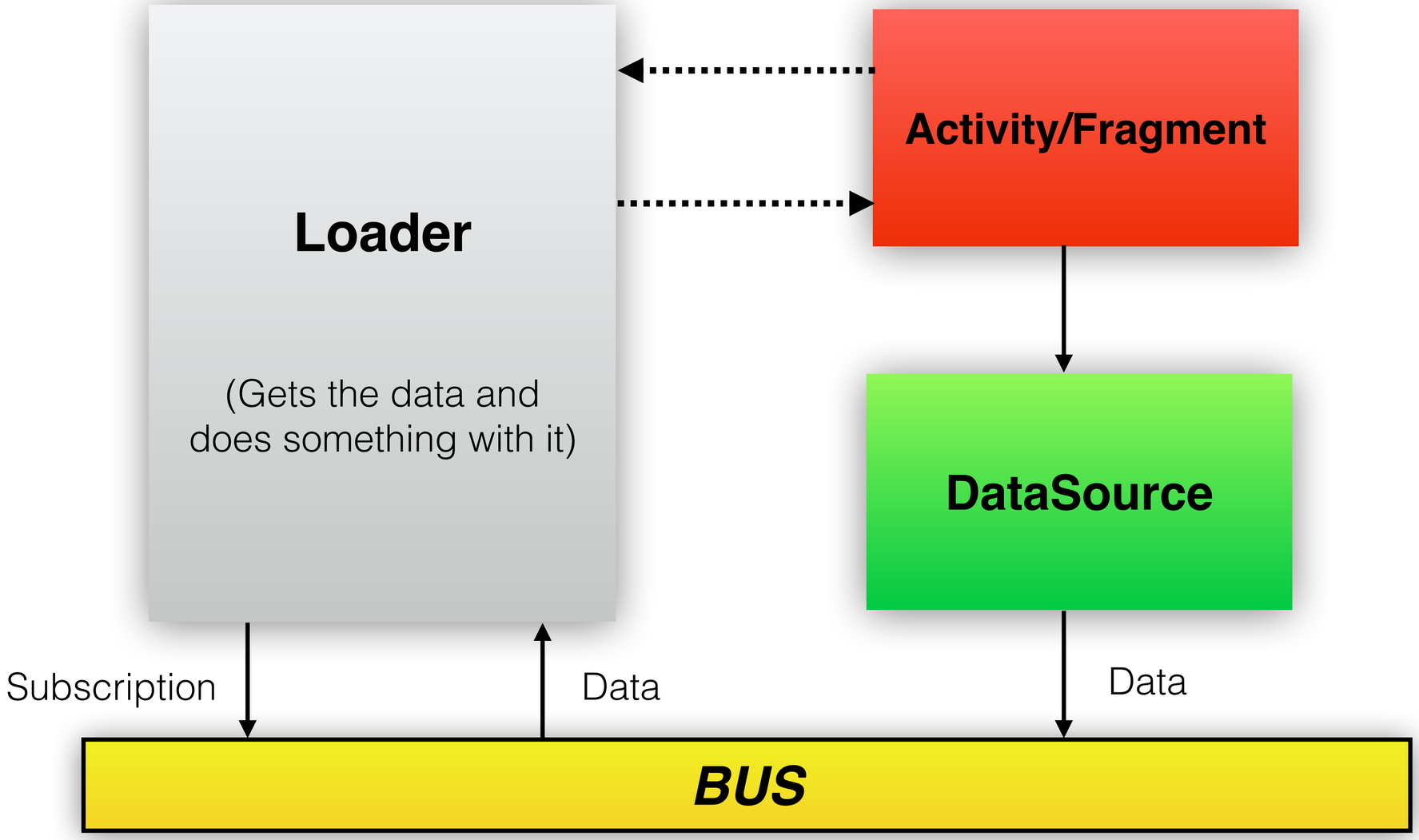


4.3 Loaders

Introduced in Android 3.0, loaders make it easy to asynchronously load data in an activity or fragment.

- We use **loaders** to request data to our database model/cache.
- The loaders, once they finish, return the data to the Activity/Fragment.
- The loaders can also subscribe themselves to the EventBus and listen for events.

4.4 Loaders and EventBus (example)



4.5 Database

- We can not decuple the Android database (SQLite) here …
- … but we try to make as much abstract as we can the work between data persistence, cache, and API data.

4.6 Others

- We use others useful open source libraries to manage different stuff, like:
    - Image management, Square Picasso.
    - Network request queue, Path PriorityJobQueue.

Questions?

תודה!!!

Thank you!!!!

Gracias!!!

+CesarValiente

@CesarValiente

License