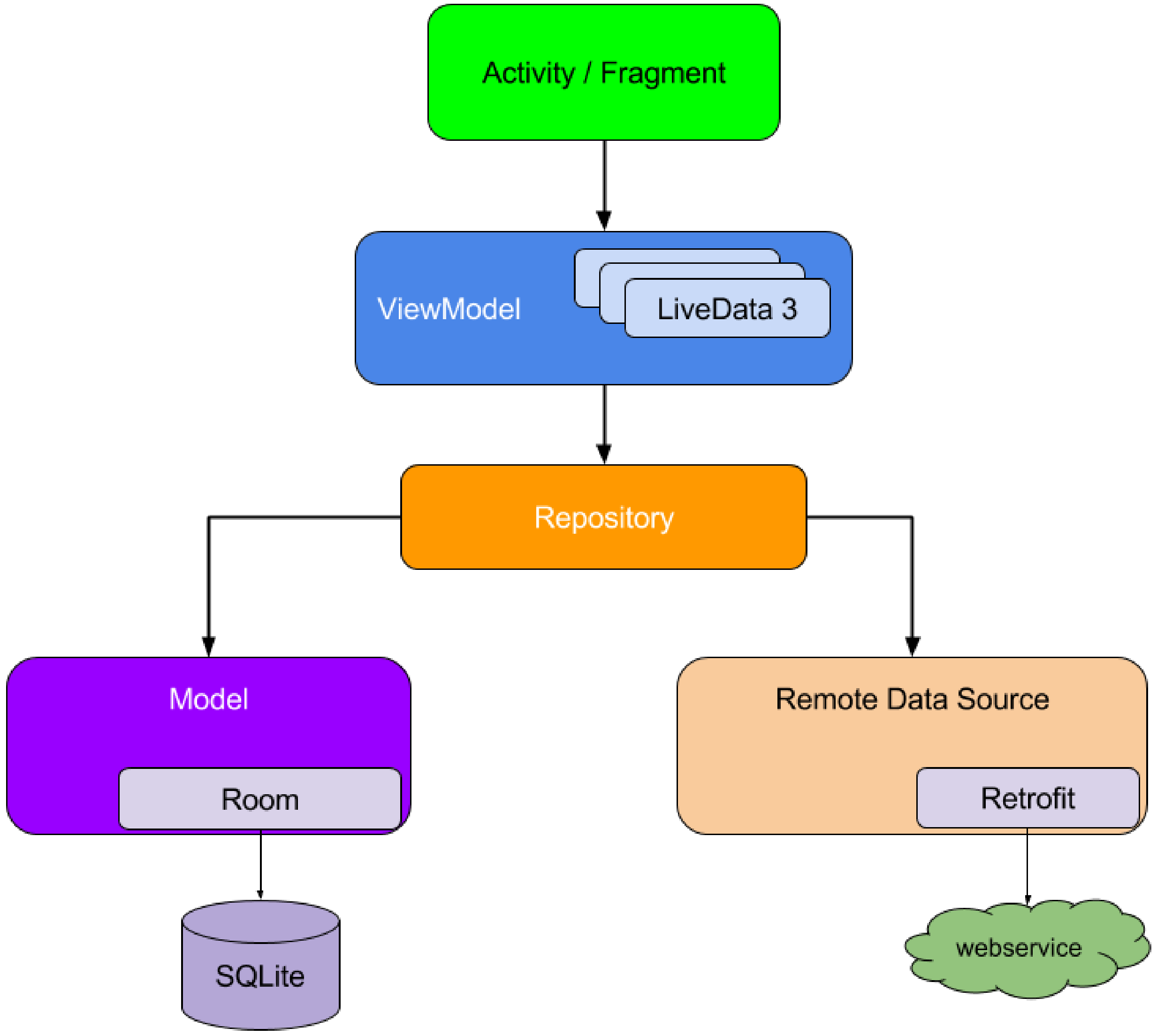
Architecture Components 2: Room

Recap

- We saw how to use ViewModel and LiveData to obtain persistence over the activity lifecycle.
- We saw the repository pattern, moving data out of the ViewModel into a separate repository class.
- We hooked the repository up to the internet, fetched weather data from there.
- We used LiveData to listen to changes in our data array, triggering an observer when the data is actually populated from the internet.
- We saw how ViewModel can help us use the MVVM architecture for our apps.

Repository Design Pattern



Today

- We’ll cover one more important architecture component: **Room**.
- We’ll see how to use Room to write data to an embedded SQLite database.

Room

Why a local database?

- Your app may need to fetch data regularly from the internet.
- Imagine if internet access breaks down.
- The app should still be responsive, and give the user some kind of access to the data.
- A local database helps you cache downloaded data.
- Workflow could be:
  - Get data from internet.
  - Cache in database.
  - Show in app.

Room: Introduction

- Writing SQLite code in Android was a pain.
- You’d use the android.database.sqlite package.
- This would expose a low-level API that lets you read/write to SQLite databases.
- Drawbacks:
  - No compile-time verification of SQL queries.
  - Must update SQL queries manually if database graph changes.
  - Must write lots of code to convert between SQL queries and data objects.

Room: Introduction

- Room was introduced to fix this.
- Provides an abstraction layer over SQLite.
- This layer allows easy and automagic conversion of SQLite queries to Java objects (a la linq scaffolding in C#).
- Key idea: Room allows you to annotate Java objects as SQLite entities.
- Let’s dive in.

Room: Components

- To get Room into your app, you need 3 components:
  - Database: a class that is the access point for the app’s relational DB. Must be annotated with @Database.
  - Entity: a table within the Database.
  - DAO: data access objects. These let you annotate Java methods with SQLite queries, or let you annotate methods as insert/delete methods.
- Note that the queries in the DAO are verified at compile time.

Room: Database

- You annotate a class with @Database.
  - This class must be an abstract class that extends RoomDatabase.
  - Must include all the entities associated with the database within the annotation.
  - Must contain an abstract method with no arguments and returns a class annotated with @Dao.
- It’s common to have only one database for your app.
- You enforce this using the Singleton pattern.

Room: Entity

- A class containing data.
- This is the data that gets stored in your database.
- You have to denote one of the variables as a primary key.
- You can annotate with your own column name, or use the variable name itself as a column name.

Room: DAO

- You give function signatures for executing SQL queries.
- You annotate each function with the appropriate SQL statement.
- Typical signatures are insert, delete, get all data, etc.
- The interface is implemented by Room! We don’t need to define the functions inside the DAO.
- Let’s see an example.

Room: Weather data?

- Let’s try and adapt the database architecture for our Openweathermap examples.
- We want to add a little database to our app as a local cache.
- There, the repository fetches data from the internet, and passes it back up to the UI.
- Now, we’ll also cache the location in the database.
- Remember: the ViewModel shouldn’t know about the database.
- The repository therefore needs to handle all database stuff.

Room: Example 35

- Recall: we hand the location string down to the ViewModel from the UI.
- Now, we’ll just store that in a database.
- This change requires a change to the repository class.
- The other classes don’t change.
- We will need to write a DAO, an Entity, and a Database.

Room: Weather example architecture

- This is a little contrived, of course.
- In practice, your architecture would be:
  - Hand location down to repository.
  - Repository checks if location already has been searched for (maintain local list of locations).
    - If it is, repository hands that up to ViewModel.
    - If not, it gets that data from the internet.
      - Puts it into database.
      - Hands it also to ViewModel.
  - ViewModel hands data to Activity/Fragment.
- I’ll leave this to you. This is mostly an exercise in SQL.

Summary

- We saw how to create databases using Room.
- We discussed ViewModel, LiveData in more detail.

Lab

- Discuss refactoring your project 1 code into the MVVM architecture following the repository pattern.
- Use Room to store weather data to a local database, either as a standalone project or as part of your Lifestyle app.
- Try the “Android Room with a View” Google codelab.