Uniwersytet Wrocławski

Rafał Lewandków

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

WYKŁAD 10

- MVx

MVx

Struktura MVx

Model:

Stan i/lub logika biznesowa i/lub struktura danych

View:

Interfejs użytkownika

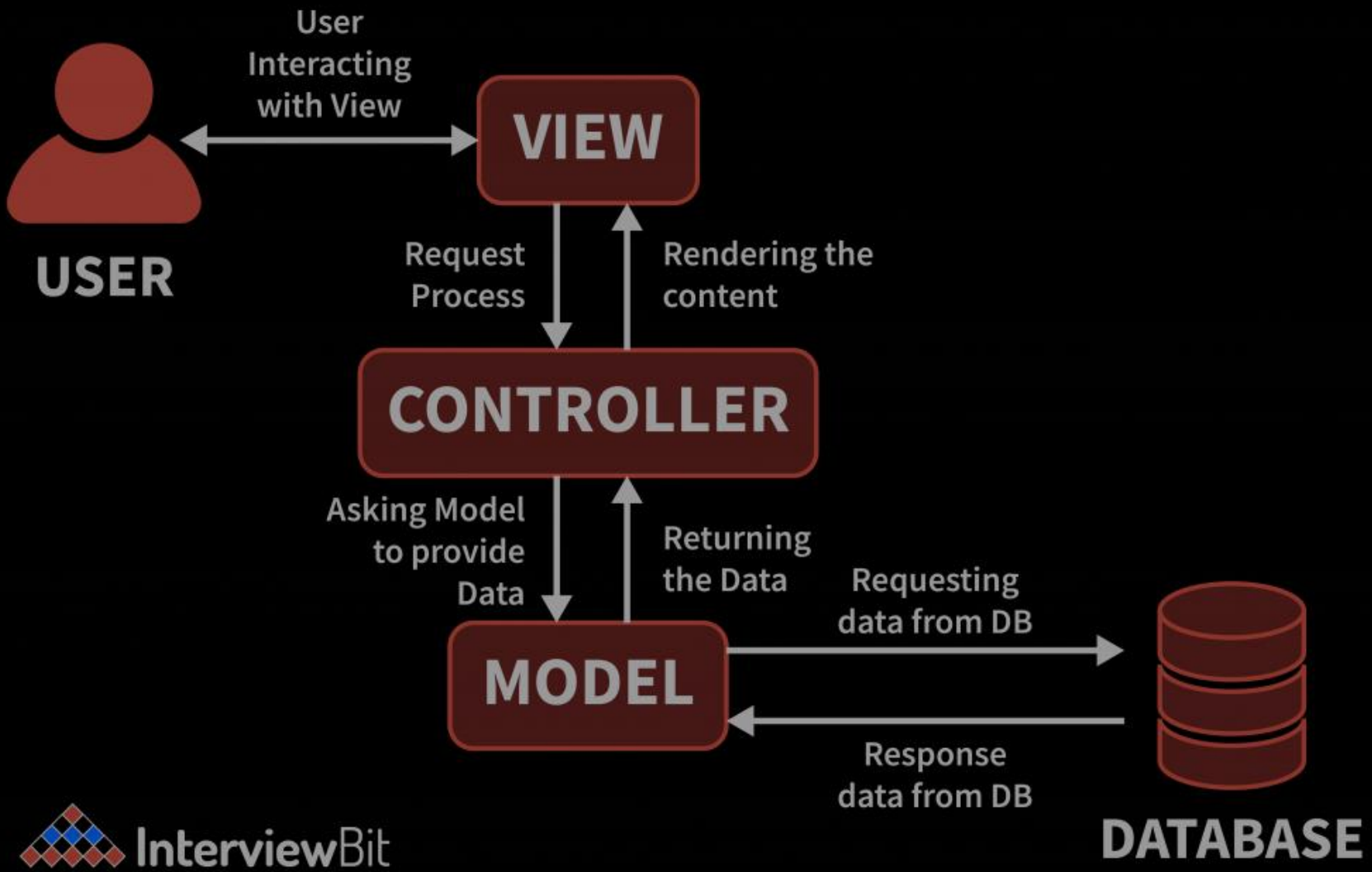
X:

Logika biznesowa i/lub flow control i/lub stan i/lub struktura danych

Zalety

- Testowalność
- Element niezmienny, cała logika interfejsu użytkownika zostaje odłączona, staje się samodzielnym elementem aplikacji

MVC


User
Interacting
with View
VIEW
USER
Request
Process
Rendering the
content
CONTROLLER
Asking Model
to provide
Data
Returning
the Data
Requesting
data from DB
MODEL
Response
data from DB
DATABASE
InterviewBit

MVC

Nowadays, the question of how to apply the MVC patterns has an answer that is easier to find. The Activities, Fragments and Views should be the Views in the MVC world. The Controllers should be separate classes that don't extend or use any Android class, and same for the Models.

MVC

```
data class Model (
    private val text: String,
    private val score: Int,
    private val ratio: Int)
{
    fun getData(): String{
        return "$text $score $ratio"
    }
}
```

MVC

```
data class Model (
    private val text: String,
    private val score: Int,
    private val ratio: Int)
{
    fun getData(): String{
        return "$text $score $ratio"
    }
}
```

```
class MyView (private val textView: TextView, private val model: Model){
    fun display(){
        textView.text = model.getData()
    }
}
```

MVC

```
data class Model (
    private val text: String,
    private val score: Int,
    private val ratio: Int)
{
    fun getData(): String{
        return "$text $score $ratio"
    }
}
```

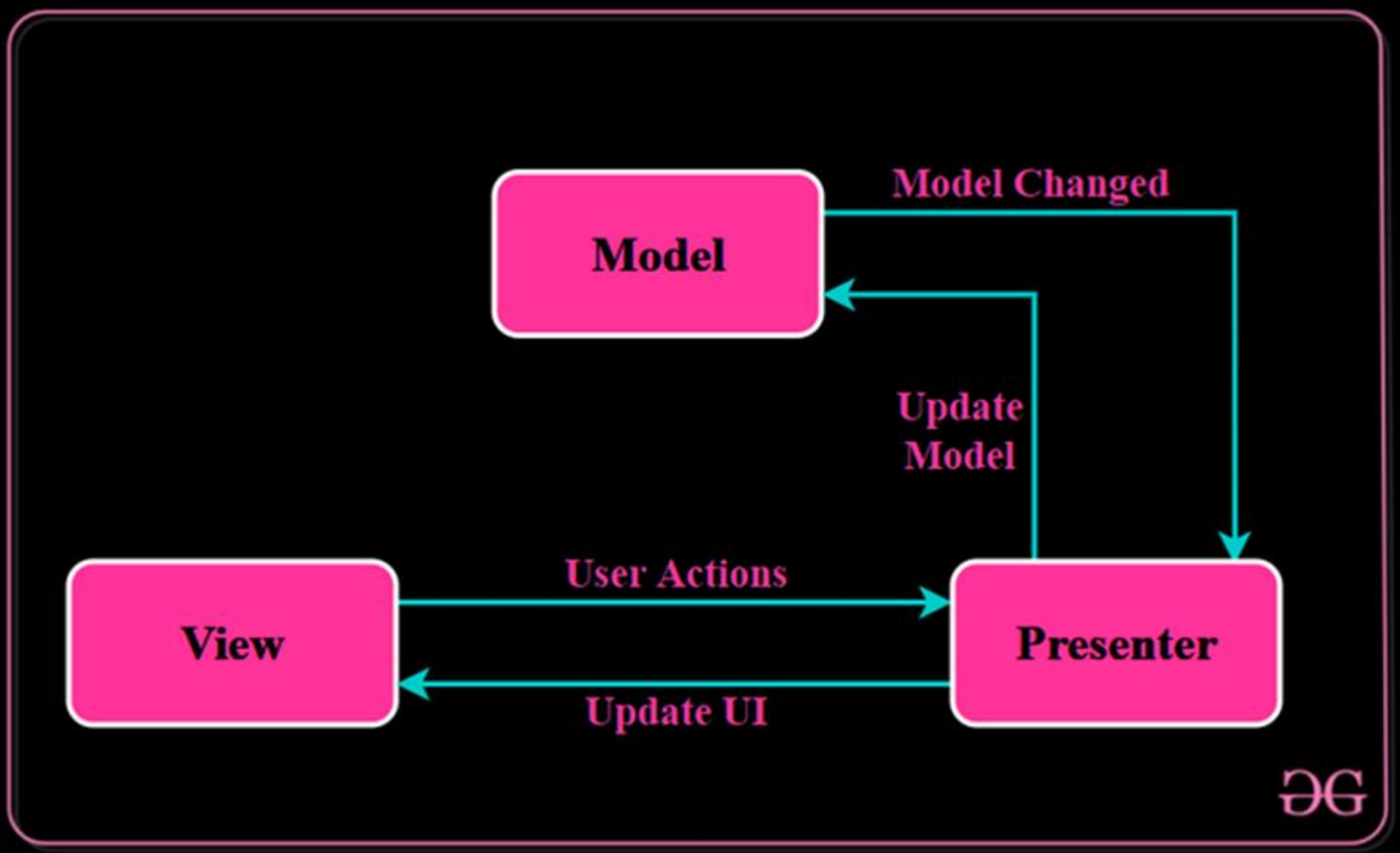
```
class MyView (private val textView: TextView, private val model: Model){
    fun display(){
        textView.text = model.getData()
    }
}
```

```
private val textView: TextView by lazy { findViewById(R.id.textView) }
    private val button: Button by lazy { findViewById(R.id.button) }
```

```
private val model: Model by lazy { Model("text", 1, 2) }
```

```
private val myView: MyView by lazy { MyView(textView, model) }
```

MVP


Model
Model Changed
Update Model
View
User Actions
Update UI
Presenter

MVP

```
data class Model (
    private val text: String,
    private val score: Int,
    private val ratio: Int)
{
    fun getData(): String{
        return "$text $score $ratio"
    }
}
```

MVP

```
data class Model (
    private val text: String,
    private val score: Int,
    private val ratio: Int)
{
    fun getData(): String{
        return "$text $score $ratio"
    }
}
```

```
interface MyView {
    fun onDisplay(text: String)
}
```

Uniwersytet Wrocławski

MVP

```
data class Model (
    private val text: String,
    private val score: Int,
    private val ratio: Int)
{
    fun getData(): String{
        return "$text $score $ratio"
    }
}
```

```
interface MyView {
    fun onDisplay(text: String)
}
```

```
class Presenter(private val myView: MyView) {
```

MVP

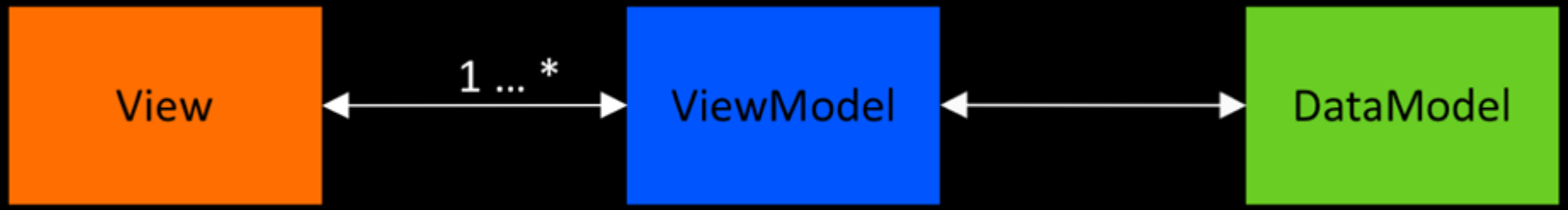
```
data class Model (
    private val text: String,
    private val score: Int,
    private val ratio: Int)
{
    fun getData(): String{
        return "$text $score $ratio"
    }
}
```

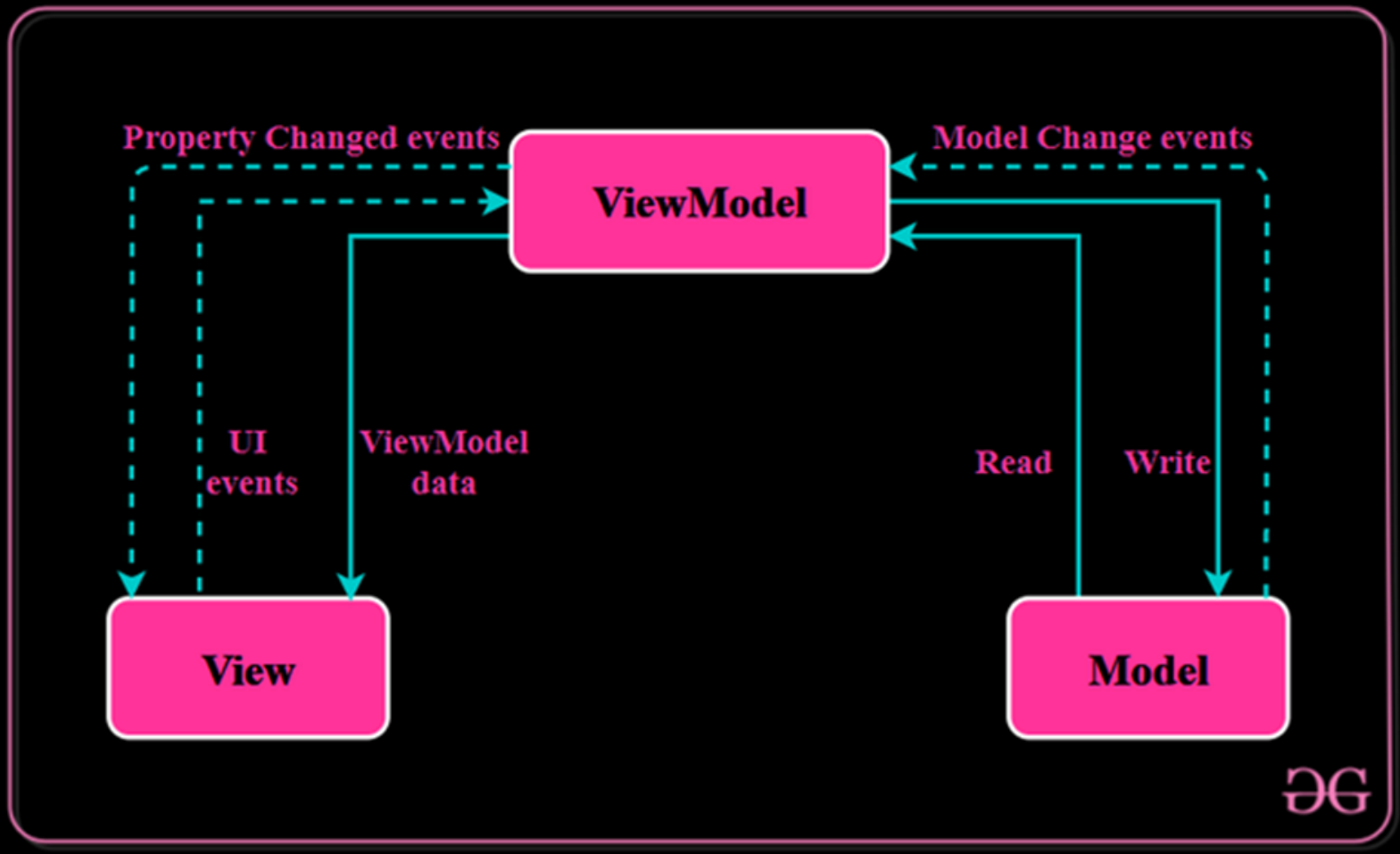
```
interface MyView {
    fun onDisplay(text: String)
}
```

```
class Presenter(private val myView: MyView) {
```

```
class MainActivity : AppCompatActivity(), MyView {
  private val textView: TextView by lazy { findViewById(R.id.textView) }
    private val button: Button by lazy { findViewById(R.id.button) }

    private val presenter: Presenter by lazy { Presenter(this) }
```

MVVM


View
1 ... *
ViewModel
DataModel


Property Changed events
Model Change events
ViewModel
UI events
ViewModel data
Read
Write
View
Model

MVVM

```
data class Model (
    private val text: String,
    private val score: Int,
    private val ratio: Int)
{
    fun getData(): String{
        return "$text $score $ratio"
    }
}
```

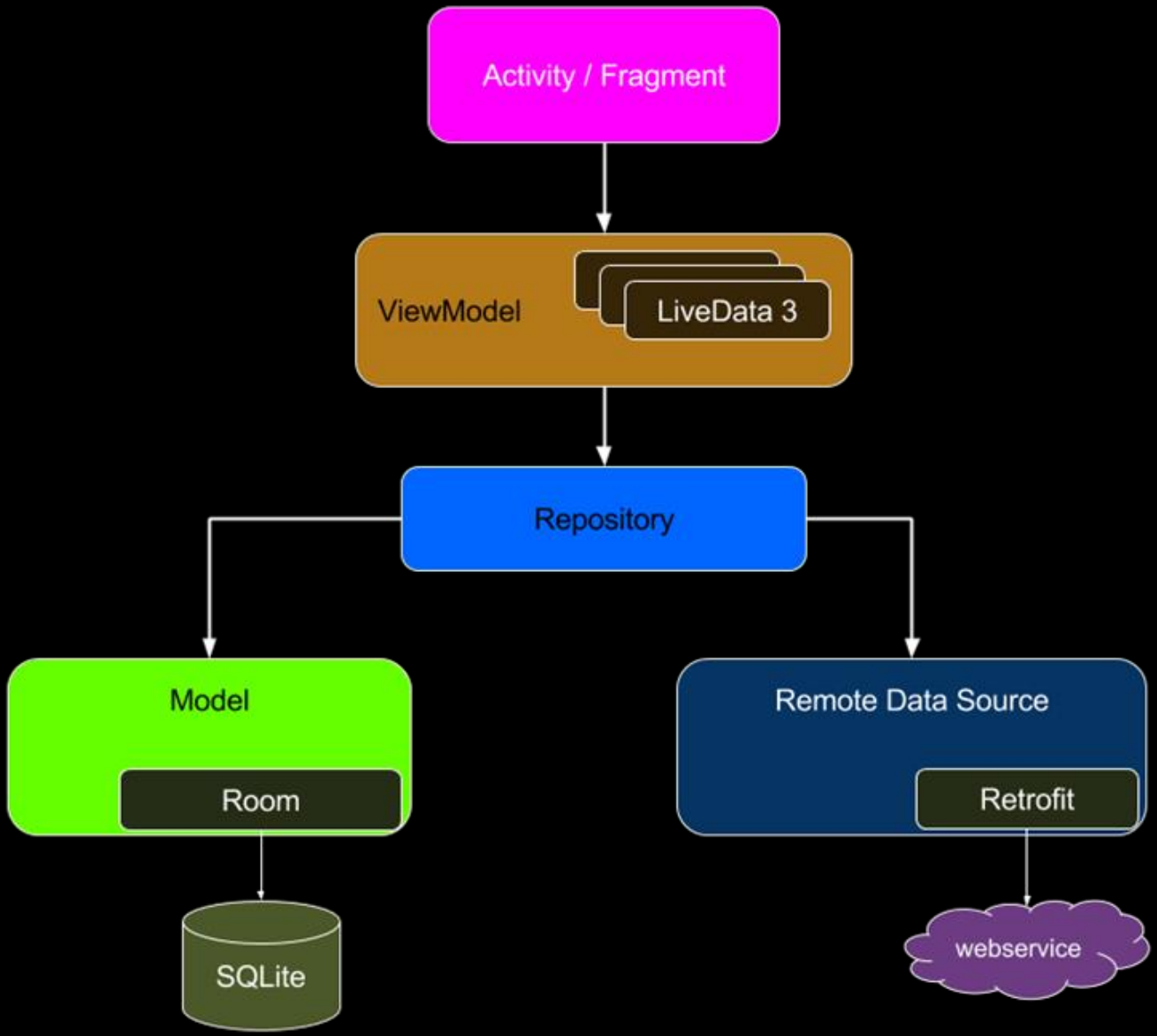
MVVM

```
data class Model (
    private val text: String,
    private val score: Int,
    private val ratio: Int)
{
    fun getData(): String{
        return "$text $score $ratio"
    }
}
class MyViewModel : ViewModel() {

    private var _myData: MutableLiveData<String> = MutableLiveData()
    val myData: LiveData<String> = _myData

    private fun getModel(): Model {
        return Model("text", 1, 2)
    }

    init {
        _myData.value = getModel().getData()
    }
}
```

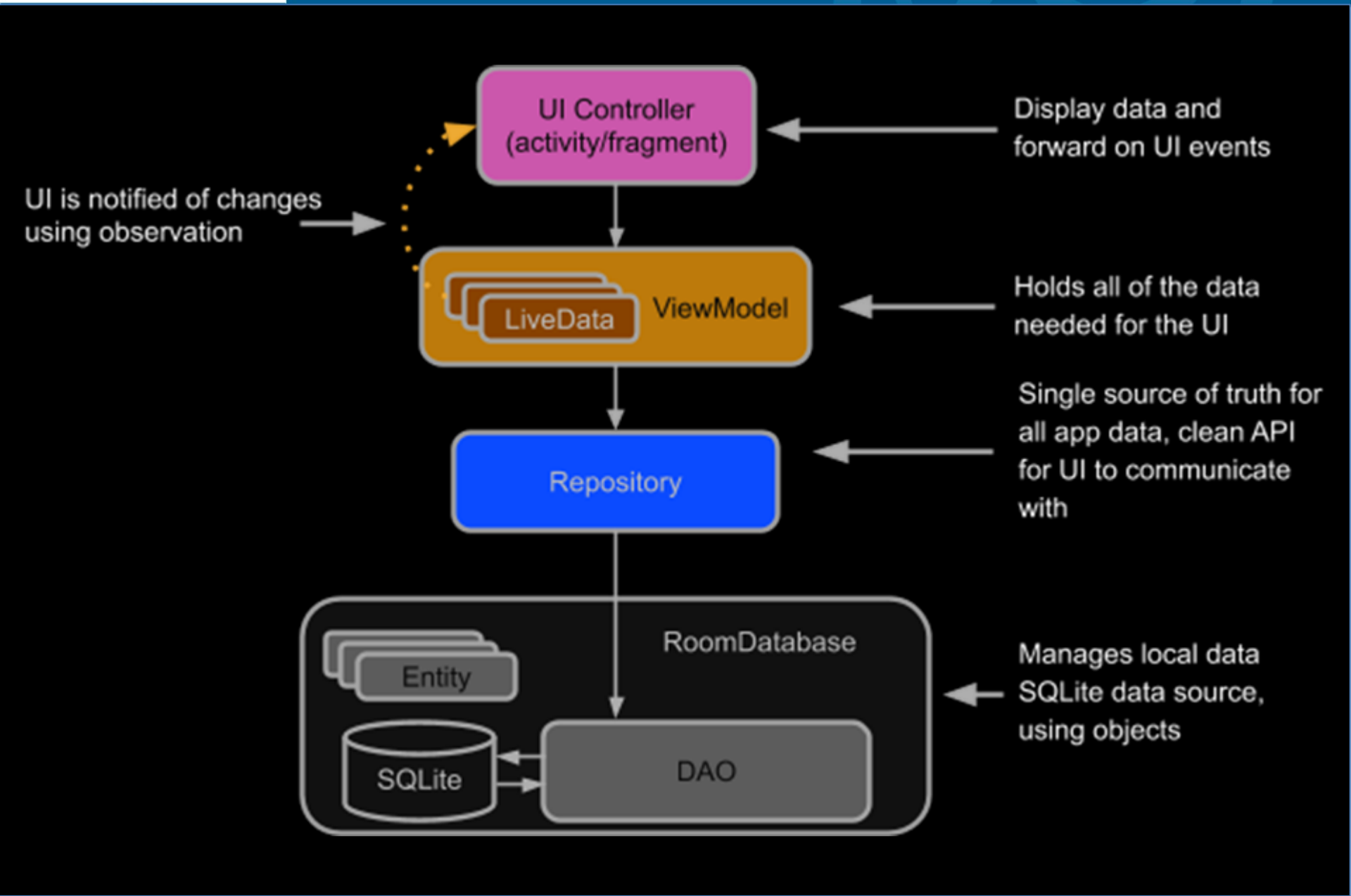
MVVM

```
class MainActivity : AppCompatActivity() {

    private val textView: TextView by lazy { findViewById(R.id.textView) }
    private val button: Button by lazy { findViewById(R.id.button) }

    private val myViewModel: MyViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        myViewModel.myData.observe(this){
            textView.text = it
        }
    }
}
```

MVVM


Activity / Fragment
ViewModel
LiveData 3
Repository
Model
Room
SQLite
Remote Data Source
Retrofit
webservice

MVVM


UI Controller
(activity/fragment)
Display data and
forward on UI events
UI is notified of changes
using observation
LiveData
ViewModel
Holds all of the data
needed for the UI
Repository
Single source of truth for
all app data, clean API
for UI to communicate
with
RoomDatabase
Entity
SQLite
DAO
Manages local data
SQLite data source,
using objects