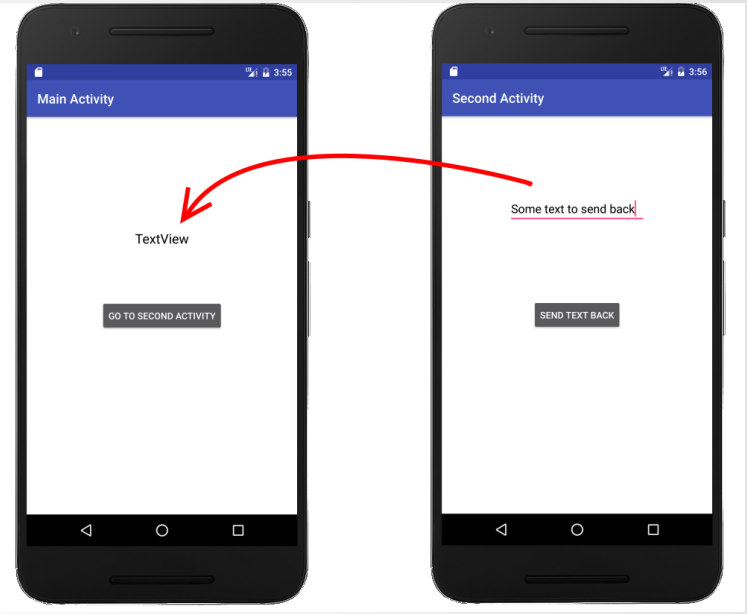
Android

ViewModel

Android

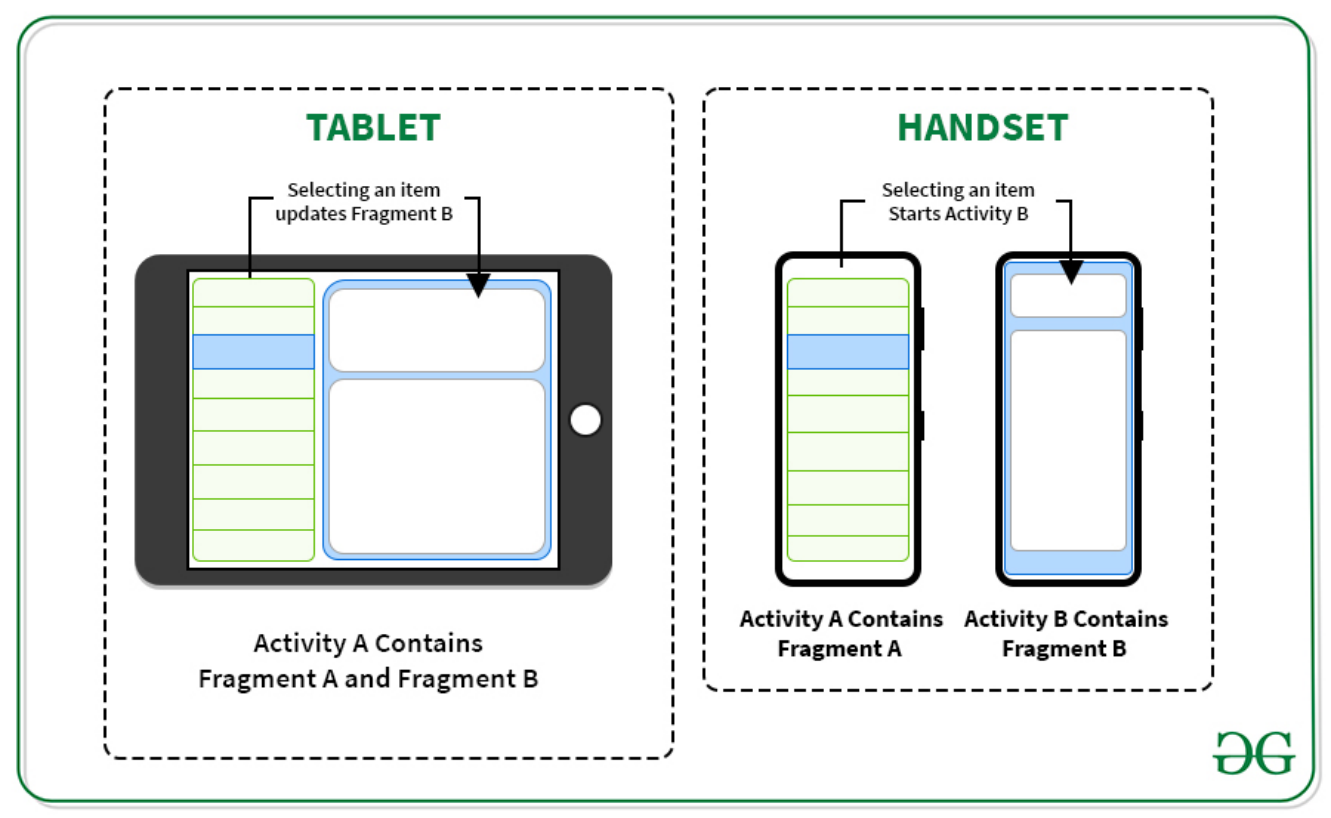
UI-Controllers

Android Activities



- Android Applikationen bestehen aus Activities
- Sie konzentrieren sich auf eine einzelne Handlung des Users

Android Fragments



- Sind Teile einer Activity

Android und UI-Controller

- Activities und Fragments sind UI-Controller
- Änderungen am Android-System führen im Allgemeinen dazu, dass UI-Controller zerstört und wieder neu erstellt werden
- Man hat im allgemeinen keinen Einfluss auf diese Vorgänge... Das passiert einfach
  -> User Aktionen
  -> System Events

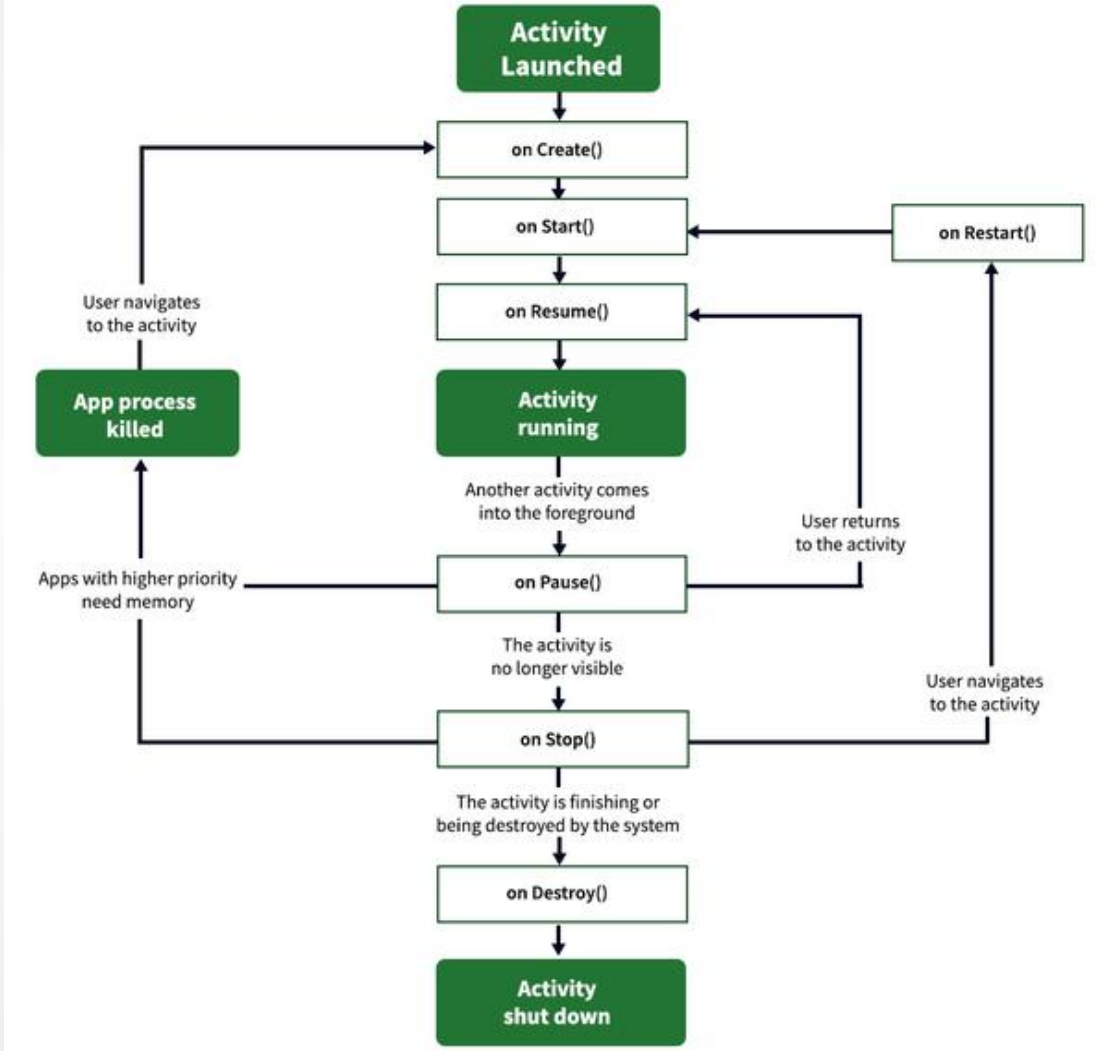
Daten sollten daher NICHT direkt in Activities oder Fragments gespeichert werden!

Diese wären dann nämlich einfach weg.

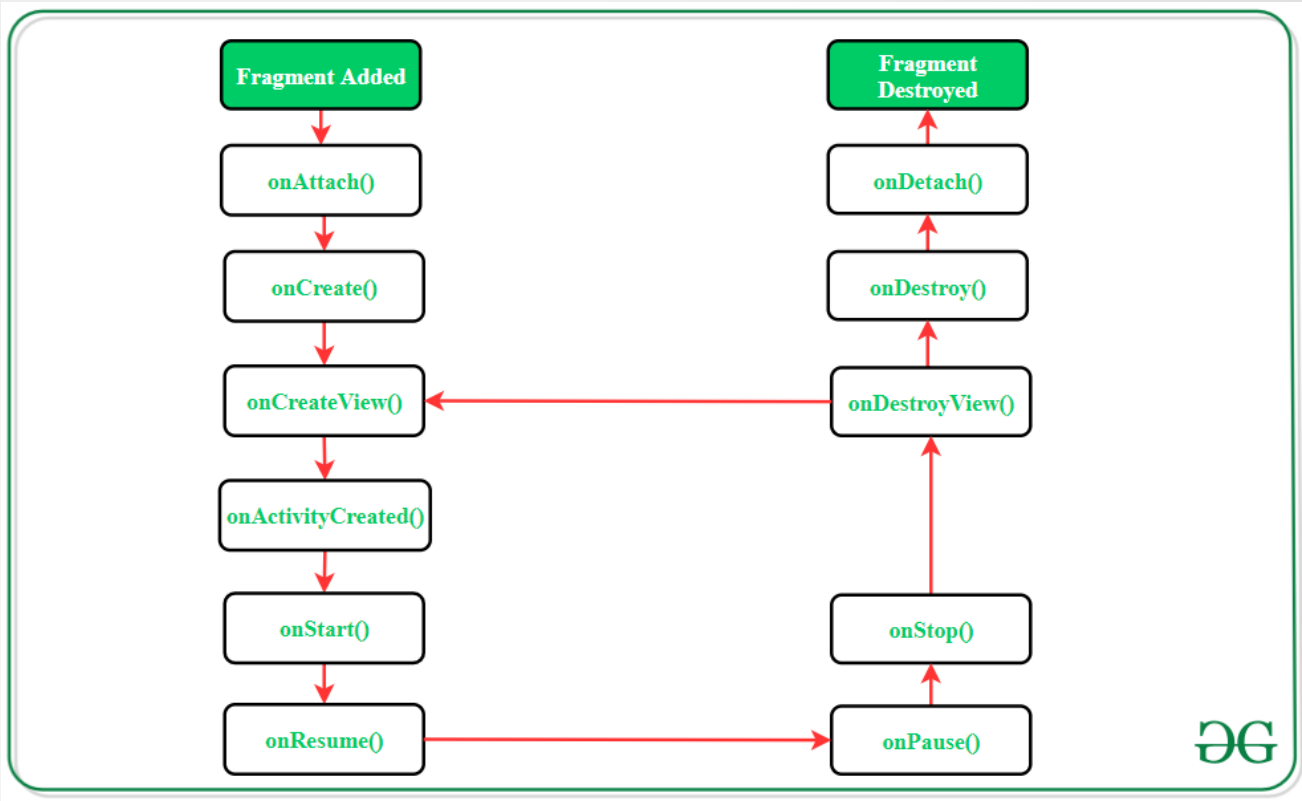
Lifecycles

Activity Lifecycle

Benötige eine Applikation mit höherer Priorität mehr Speicher, werden pausierte Activities einfach gelöscht.



Fragment Lifecycle



destroy / re-create

- Für einfache Daten kann man **onSaveInstance()** verwenden, um sie dann bei **onCreate()** wieder herzustellen
- Für große Datenmengen (oder schwer zu serialisierbare Daten) ist das kein guter Ansatz
- Außerdem gibt es noch asynchrone Calls, die gerade laufen könnten, obwohl der UI-Controller gerade zerstört wurde...

Ohne System wird das alles ziemlich schnell unübersichtlich

ViewModel

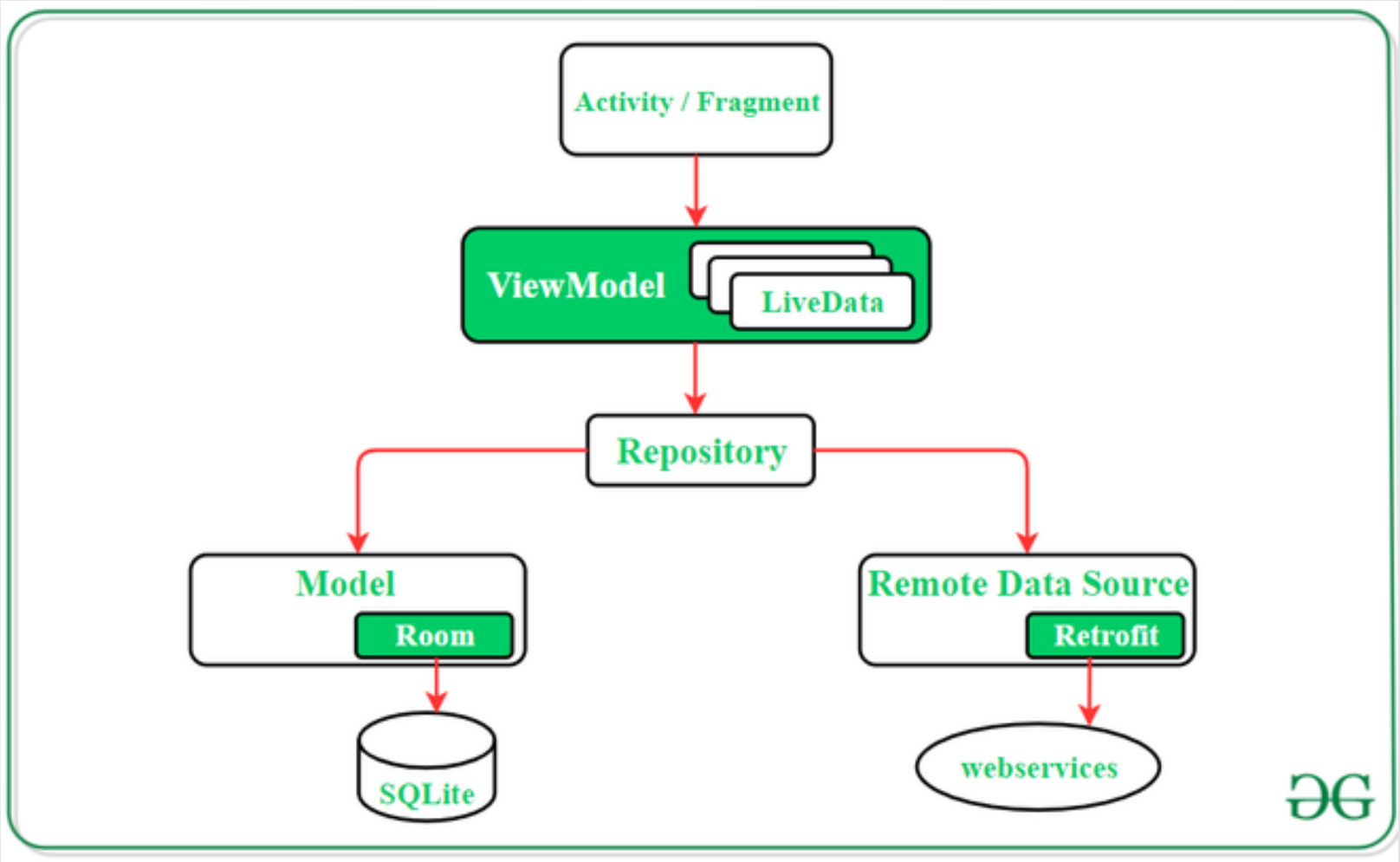
Erklärung

ViewModel

- Teil der Android Architektur
- Speichert und managed UI-Daten unter Berücksichtigung des Activity-Lifecycles
- Wird bei Konfigurationsänderungen NICHT zerstört!
- Erlaubt dadurch der Applikation Konfigurationsänderungen zu überstehen

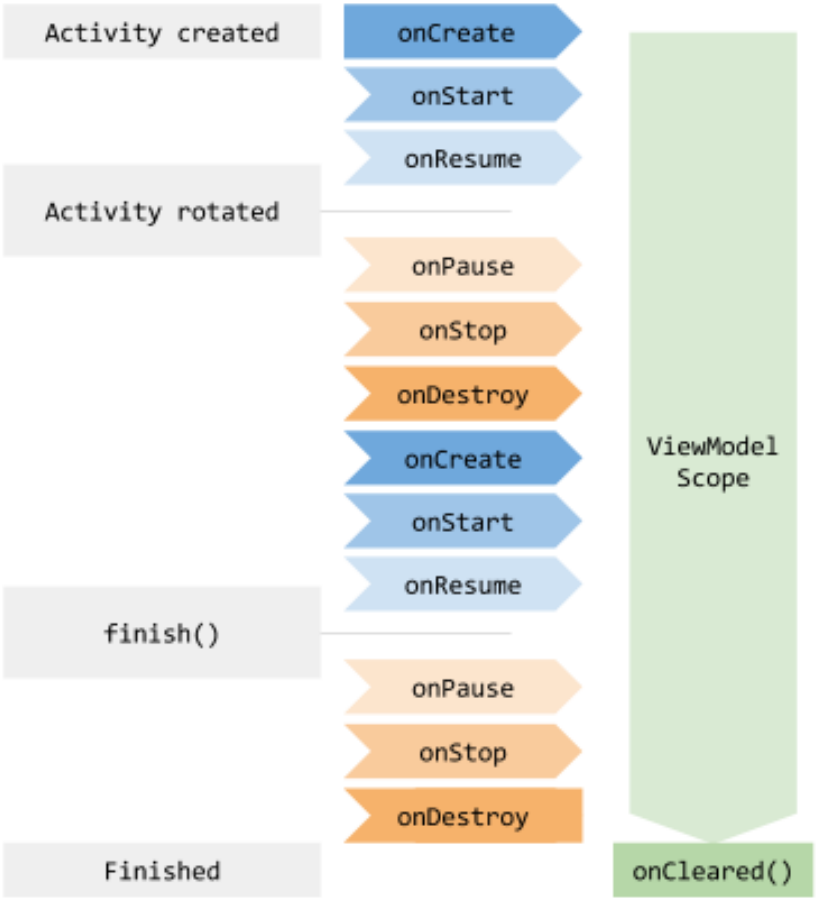
Im ViewModel werden auch externe Datenquellen angebunden (Datenbank, Webservices...)

ViewModel


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

ViewModel Lifecycle

Das ViewModel bleibt erhalten, obwohl die Activity wegen einer Rotation zerstört wird...



ViewModel

Beispiel

ViewModel und Zugriff

```kotlin
class MyViewModel : ViewModel() {
    private val users: MutableLiveData<List<User>> by lazy {
        MutableLiveData<List<User>>().also {
            loadUsers()
        }
    }

    fun getUsers(): LiveData<List<User>> {
        return users
    }

    private fun loadUsers() {
        // Do an asynchronous operation to fetch users.
    }
}
```

```kotlin
class MyActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        // Create a ViewModel the first time the system calls an
        // activity's onCreate() method.
        // Re-created activities receive the same MyViewModel instance
        // created by the first activity.

        // Use the 'by viewModels()' Kotlin property delegate
        // from the activity-ktx artifact
        val model: MyViewModel by viewModels()
        model.getUsers().observe(this, Observer<List<User>>{ users ->
            // update UI
        })
    }
}
```

Shared ViewModels

- Ein gängiges Problem ist, dass sich zwei Fragments Daten teilen müssen
- Das erste selektiert zum Beispiel ein List-Item
- Das zweite zeigt Daten zum vom ersten ausgewählten Item an

Hier ist ein ViewModel sehr hilfreich, auf das beide Fragments Zugriff haben

Shared ViewModels

```
class SharedViewModel : ViewModel() {
    val selected = MutableLiveData<Item>()

    fun select(item: Item) {
        selected.value = item
    }
}

class ListFragment : Fragment() {
    private lateinit var itemSelector: Selector
    private val model: SharedViewModel by activityViewModels()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        itemSelector.setOnClickListener { item ->
            // Update the UI
        }
    }
}

class DetailFragment : Fragment() {
    private val model: SharedViewModel by activityViewModels()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        model.selected.observe(viewLifecycleOwner, Observer<Item> { item ->
            // Update the UI
        })
    }
}
```

Referenzen

- https://developer.android.com/topic/libraries/architecture/viewmodel
- https://www.geeksforgeeks.org/viewmodel-in-android-architecture-components/