

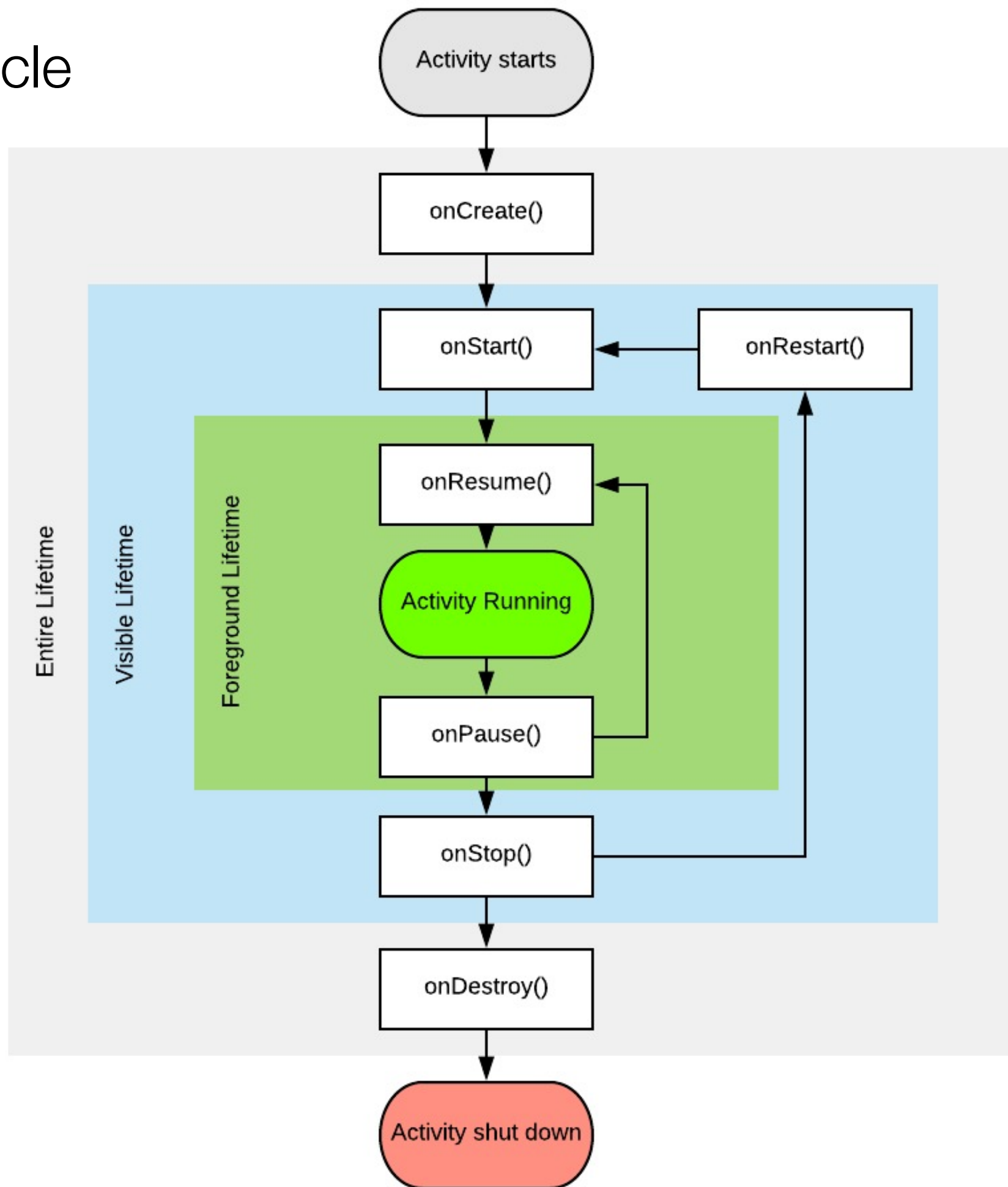
MOBG56 - Développement mobile

MVVM

Etat de l'activité

- Une activité peut avoir différents états
 - **Starting** : en cours de chargement.
 - **Running** : chargement terminé, entièrement visible à l'écran.
 - **Paused** : partiellement visible à l'écran (exemple : une autre activité la cache partiellement).
 - **Stopped** : l'activité n'est plus visible, mais reste en mémoire.
 - **Destroyed** : effacée de la mémoire.
- Les transitions entre ces états sont représentées par des événements qui peuvent être interceptés dans le code de l'activité
 - `onCreate()`, `onPause()`, `onResume()`, `onStop()`, `onDestroy()`

Activity lifecycle



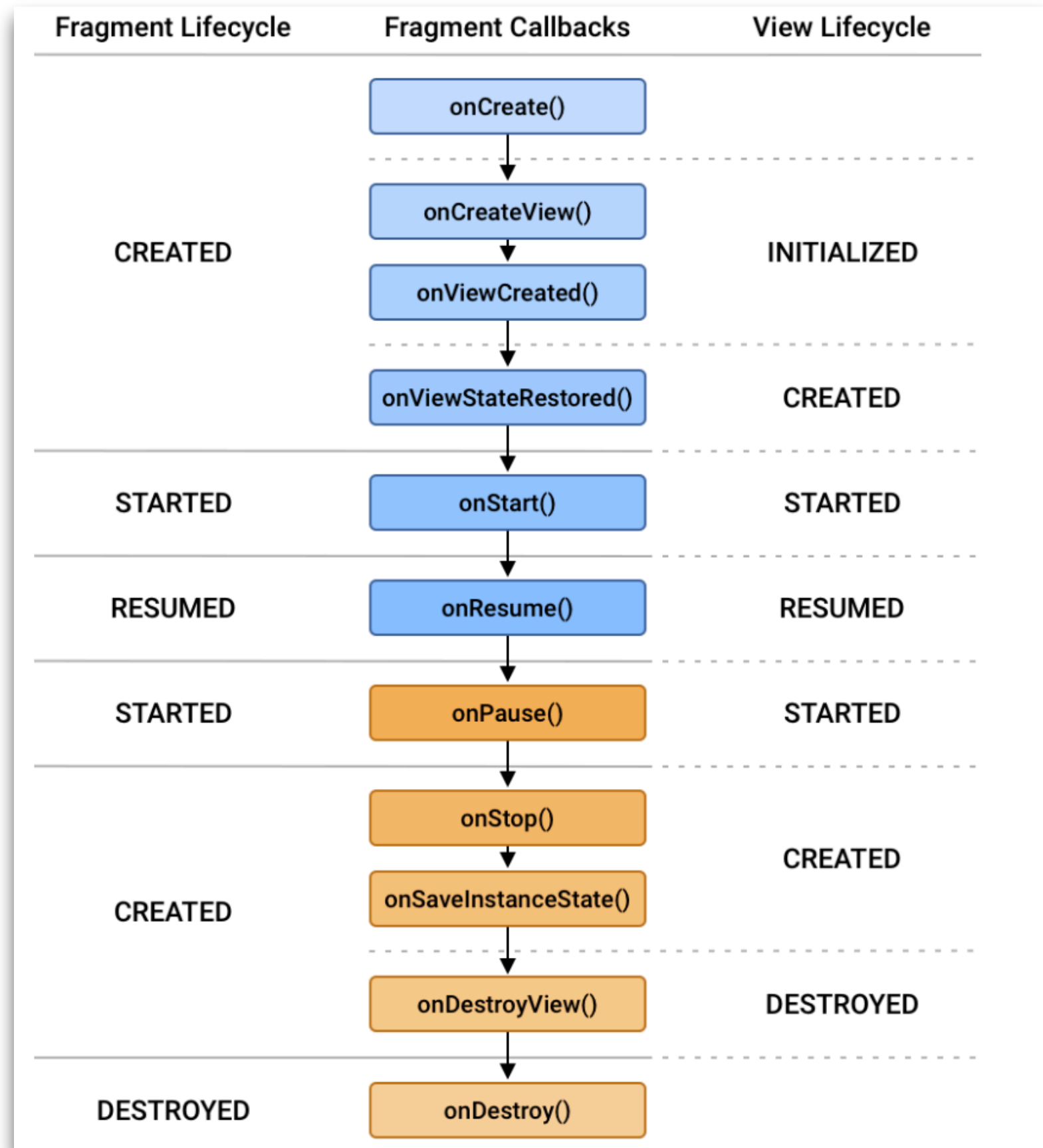
Quand l'activité est-elle détruite ?

- Lorsque le système a besoin de mémoire et que l'activité n'est pas visible.
- **Lorsque l'orientation du device change (portrait-landscape)**
 - l'activité est détruite et recrée instantanément.
 - toutes les ressources contenues dans l'activité sont détruites.
 - L'orientation du device est un cas particulier de **configuration change**
 - L'activité se recrée pour recharger une ressource qui a changée
 - Layout
 - Fichier strings (choix de langue)
 - Taille de police de caractère
- Activité = fondation instable de l'application



Fragment lifecycle

- Lié au lifecycle de l'activité
- Activité se recrée (config change) → le fragment se recrée
- onStart(), onResume(), onPause(), onStop() appellent méthodes équivalentes sur Fragment
- Lifecycle des Fragments = source de beaucoup de bugs



Fragments / Activités :


**TOUTES LEURS PROPRIETES
SERONT DETRUITES
AU MOINDRE CONFIG-CHANGE**



Solution pour l'interface

- Utilisation de onSaveInstanceState()

```
class MainActivity : AppCompatActivity() {  
  
    private var someIntVariable : Int = 0  
    private var someBooleanVariable : Boolean = false  
  
    ...  
    override fun onSaveInstanceState(outState: Bundle) {  
        super.onSaveInstanceState(outState)  
  
        // save ours private vars in the Bundle  
        outState.putInt("intVariable", someIntVariable)  
        outState.putBoolean("boolVariable", someBooleanVariable)  
    }  
    ...  
}
```



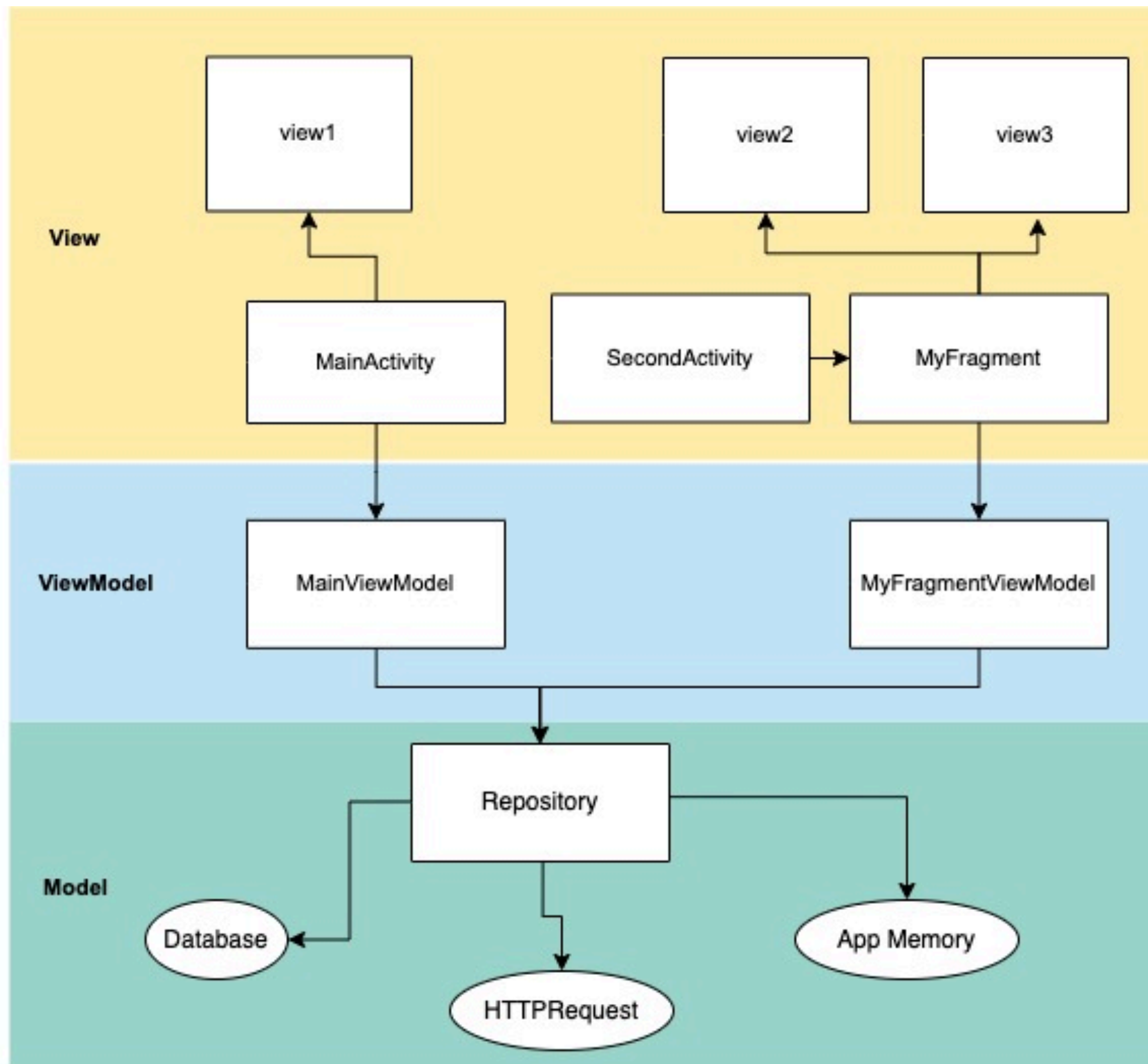
Clé sous laquelle la
variable est enregistrée

variable

Solutions pour les données de l'application

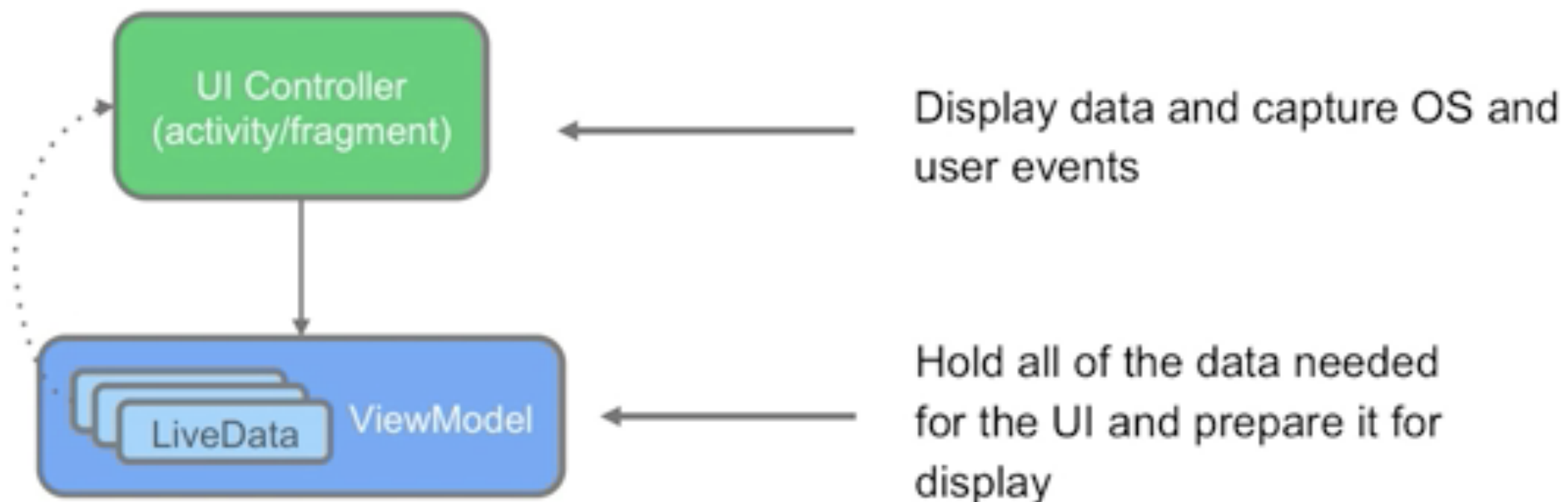
- Séparation des données (modèle) et de l'UI
- Architecture Model-View-ViewModel (**MVVM**)
 - **Model** : données de l'application, provenant d'un fichier JSON, d'une requête vers un service web, d'une base de données locale, etc ... Ne connaît rien de l'interface (**aucune référence de composants graphiques dans ces classes**)
 - **View** : classes qui gèrent ce qui est affiché à l'écran (les composants graphiques). Les Activités font également partie de cette catégorie.
 - **ViewModel** : lien entre le Model et la View. Ces classes contiennent les données à afficher dans les Views.

Exemple de MVVM



ViewModel

- Objet qui stocke les données affichés par le Fragment
- N'est pas détruit lors des *config-change*
- Est détruit lorsque le Fragment est détaché
- En général : 1 classe ViewModel par Fragment



ViewModel

Définition :

```
class MyViewModel : ViewModel() {  
    // some properties ...  
    // some methods  
}
```

Instanciation dans le Fragment:

```
class MyFragment : Fragment() {  
    private lateinit var viewModel: MyViewModel  
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,  
                               savedInstanceState: Bundle?): View? {  
        ...  
        viewModel = ViewModelProvider(this).get(MyViewModel::class.java)  
        ...  
    }  
}
```

ViewModel

- Comment initialiser les propriété du ViewModel ?
- Assignment des valeurs hors constructeur

```
viewModel.property1 = "Hello"  
viewModel.property2 = 23
```

ViewModel

- Comment initialiser les propriété du ViewModel ?
 - Utilisation d'un constructeur + classe Factory

```
class MyViewModel(val property1, val property2): ViewModel() {  
    ...  
}
```

CONSTRUCTEUR

```
class MyViewModelFactory(private val property1: String,  
                        private val property2 : Int) : ViewModelProvider.Factory  
{  
    override fun <T : ViewModel?> create(modelClass: Class<T>): T {  
        if (modelClass.isAssignableFrom(MyViewModel::class.java)) {  
            return MyViewModel(property1, property2) as T  
        }  
        throw IllegalArgumentException("Unknown ViewModel class")  
    }  
}
```

FACTORY

ViewModel

- Comment initialiser les propriété du ViewModel ?
 - Utilisation d'un constructeur + classe Factory

```
class MyFragment : Fragment() {  
  
    private lateinit var viewModel: MyViewModel  
    private lateinit var viewModelFactory: ScoreViewModelFactory  
  
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,  
                               savedInstanceState: Bundle?): View? {  
        ...  
        viewModelFactory = MyViewModelFactory("Hello" , 23)  
        viewModel = ViewModelProvider(this, viewModelFactory)  
                               .get(ScoreViewModel::class.java)  
        ...  
    }  
}
```

INSTANCIATION

LiveData

- Observable/Observer design pattern
- Twist : Observer s'active uniquement quand le lifecycle associé est **STARTED** ou **RESUMED**

```
val someObservableValue = MutableLiveData<String>()  
                                n'importe quelle classe
```

...

```
someObservableValue.observe(lifecycleOwner) { theValue ->  
    Log.d(TAG, $theValue)  
}
```

...

```
someObservableValue.value = "Hello"
```

lifecycle associé

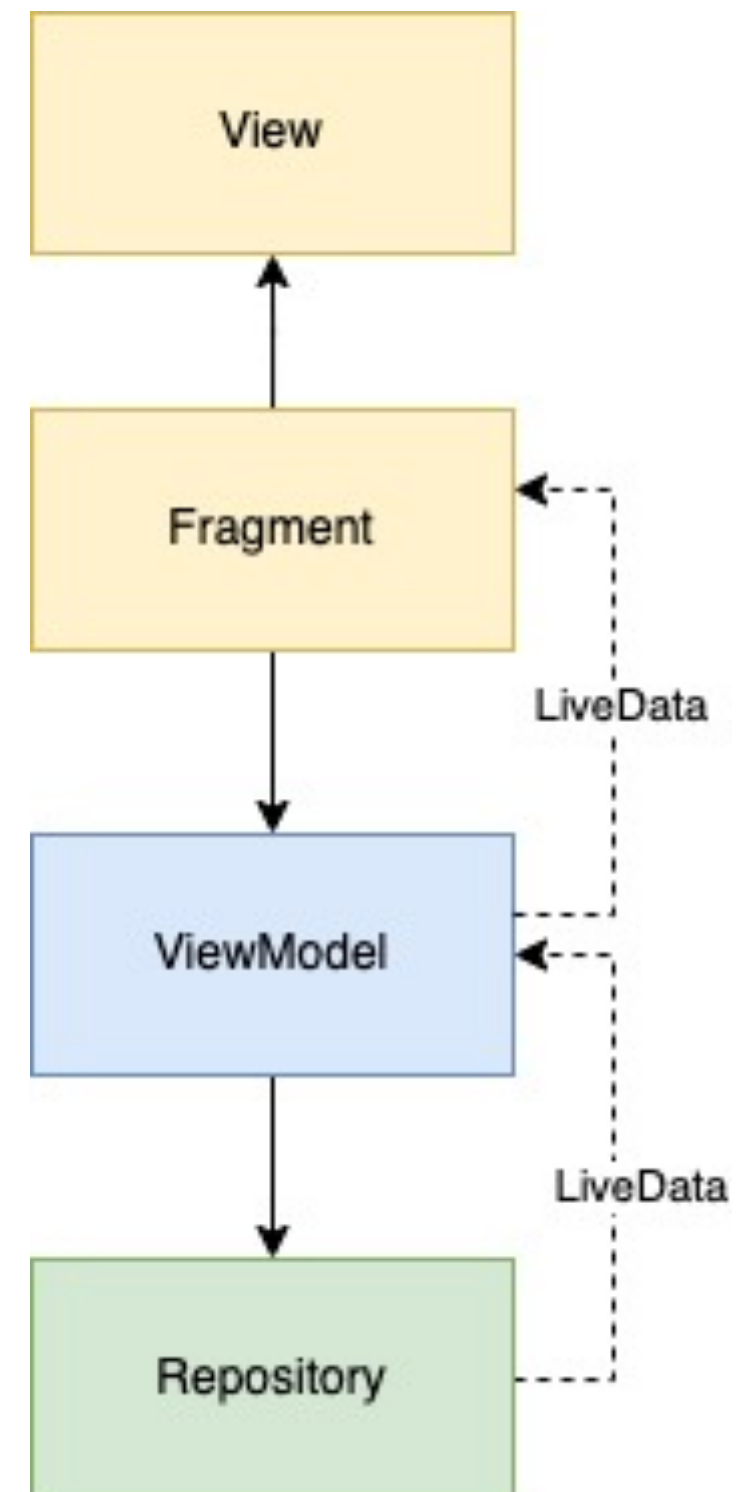


TRIGGERS

uniquement si lifecycle est STARTED ou RESUMED

Utilisation de LiveData

- Le Fragment observe les propriétés LiveData du ViewModel
- Le Fragment update les Views
- Le ViewModel observe les propriétés du Modèle (Repository)

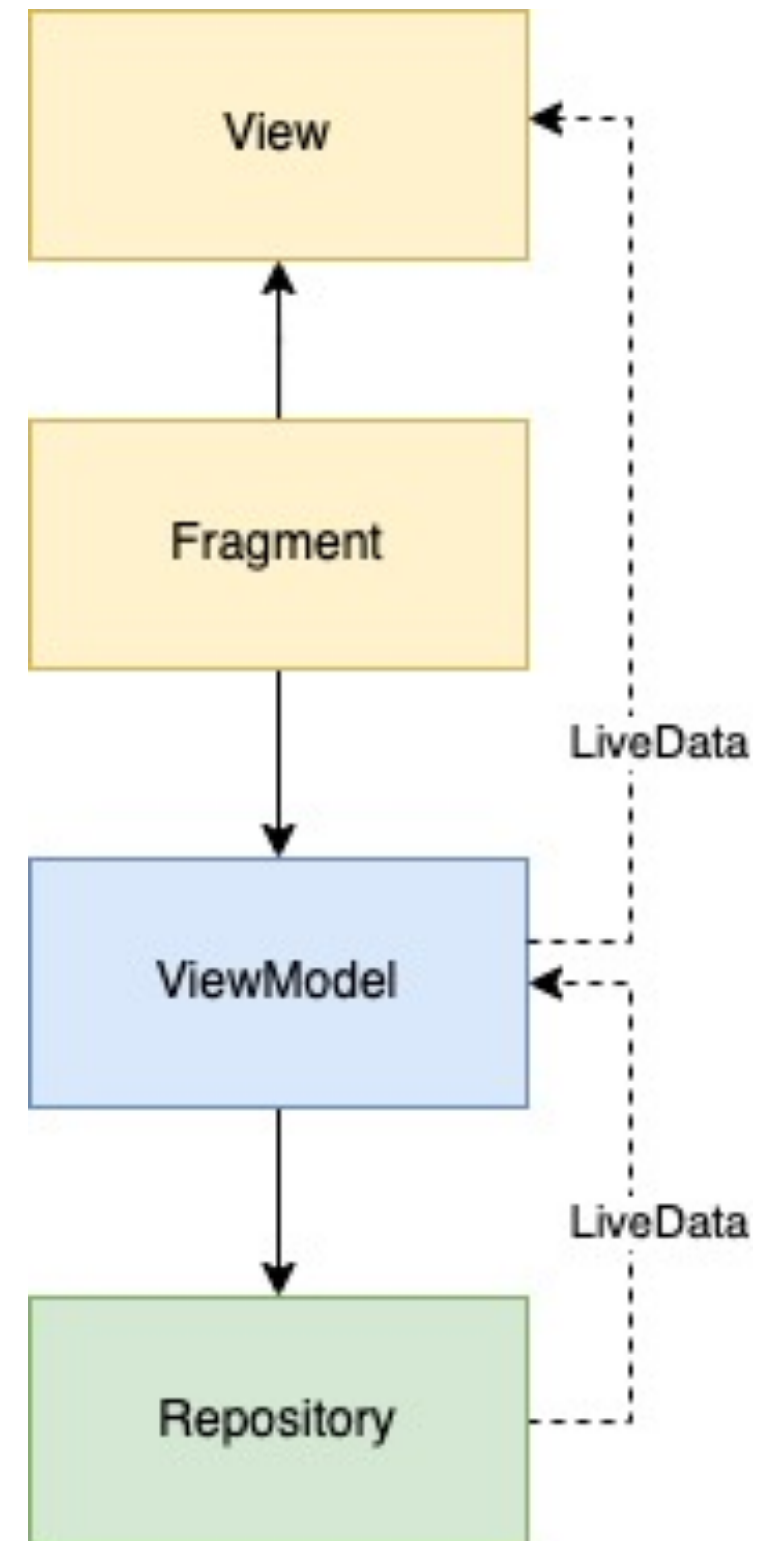


Utilisation de LiveData + DataBinding

- Les Views/Layout observe directement le ViewModel sans passer par le Fragment
- Accès direct au ViewModel depuis layout XML :

```
<data>
  <variable
    name="myViewModel"
    type="com.example.android.myapp.MyViewModel" />
</data>

<TextView
  android:id="@+id/word_text"
  ...
  android:text="@{myViewModel.property1}"
  ... />
```



Utilisation de LiveData + DataBinding

- Configuration dans Fragment :

```
class MyFragment : Fragment() {  
  
    private lateinit var viewModel: MyViewModel  
    private lateinit var viewModelFactory: ScoreViewModelFactory  
  
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,  
                             savedInstanceState: Bundle?): View? {  
        ...  
        viewModelFactory = MyViewModelFactory("Hello" , 23)  
        viewModel = ViewModelProvider(this, viewModelFactory)  
            .get(ScoreViewModel::class.java)  
  
        binding.myViewModel = viewModel  
        // Specify the fragment view as the lifecycle owner of the binding.  
        // This is used so that the binding can observe LiveData updates  
        binding.lifecycleOwner = viewLifecycleOwner  
    }  
}
```