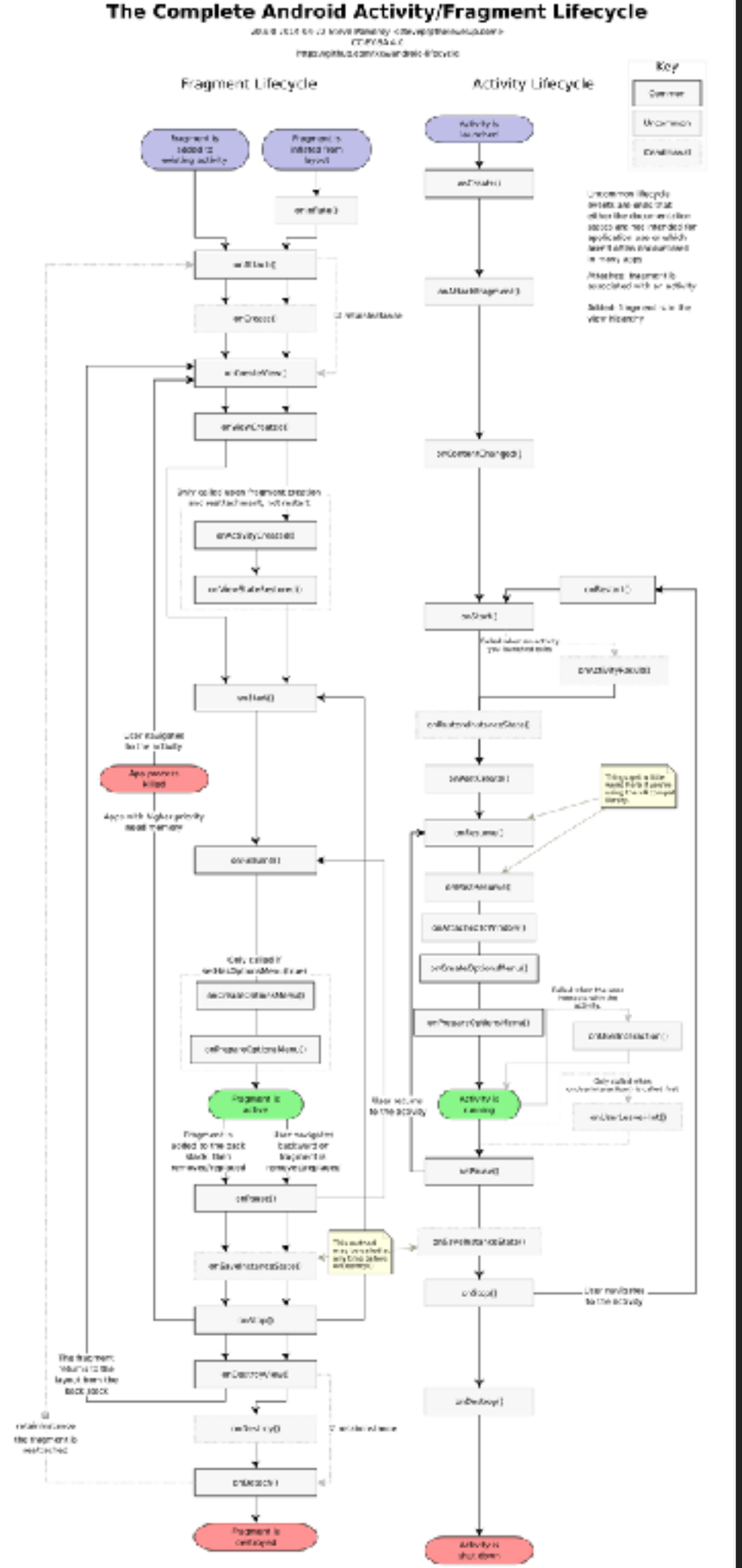
DROID DEVS CPH, JUNE 2017

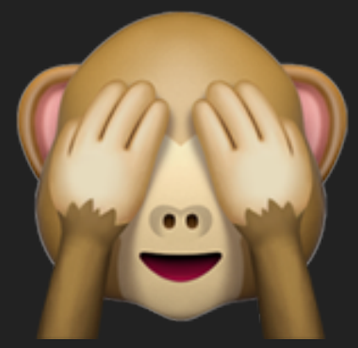
ARCHITECTURE COMPONENTS

Jose Luis Pintado Barbero
Danske Bank MobileLife

PROBLEMS...

SCREEN ROTATION, CONFIG CHANGES





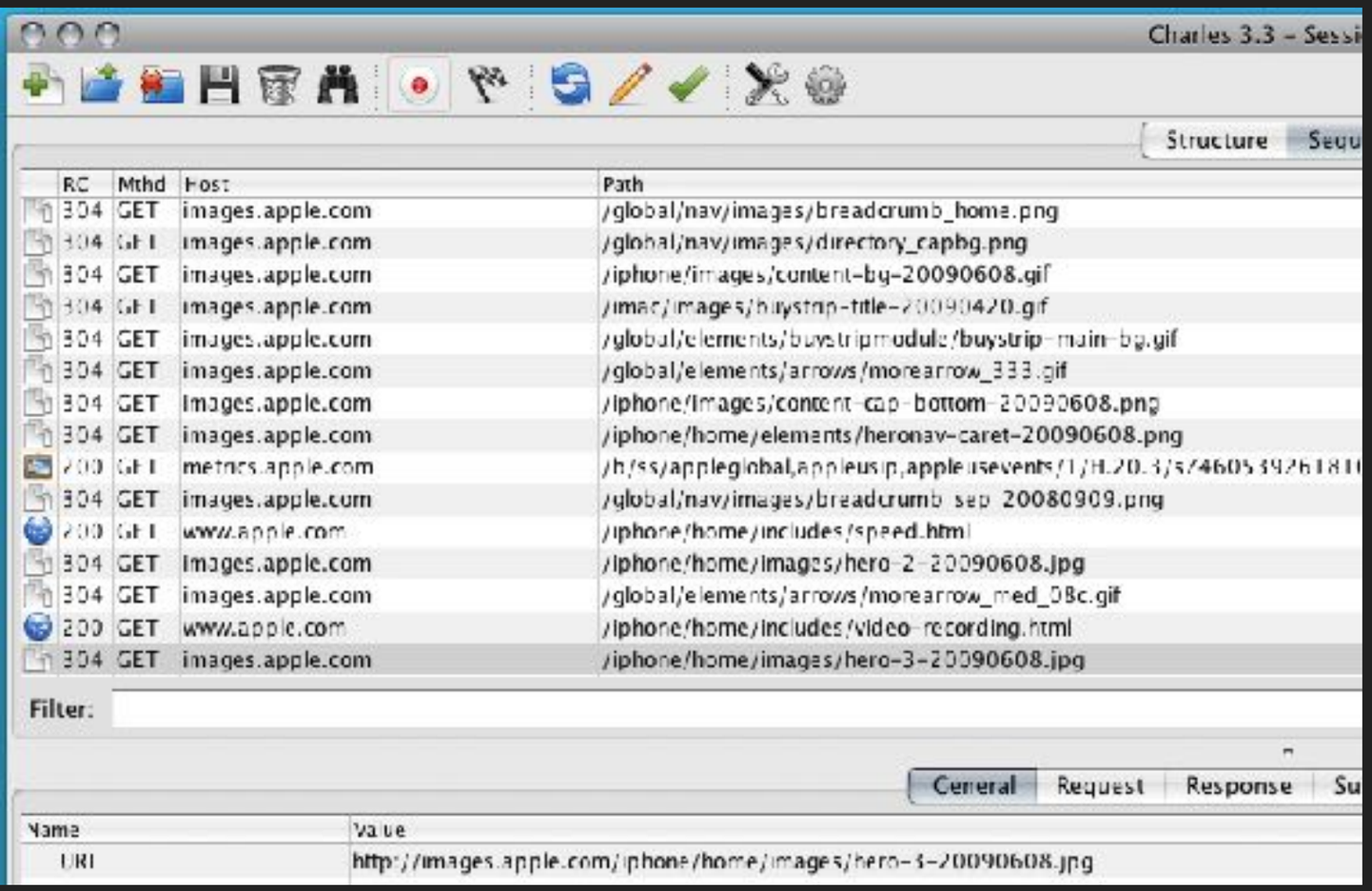
MORE PROBLEMS...

LIFECYCLES ARE COMPLEX

```
protected void onStart() {
    Sensors.observe(...);
    requests.observe(...);
    expensiveFoo.start(...);
    whenDoesThisEnd.pleaseSaveMe();
}

protected void onStop() {
    Sensors.removeObserver(...);
    requests.cancel(...);
    expensiveFoo.stop(...);
        whenDoesThisEnd.iForgotThisCall();
}
```

MORE PROBLEMS…

MESSING UP THE CODE

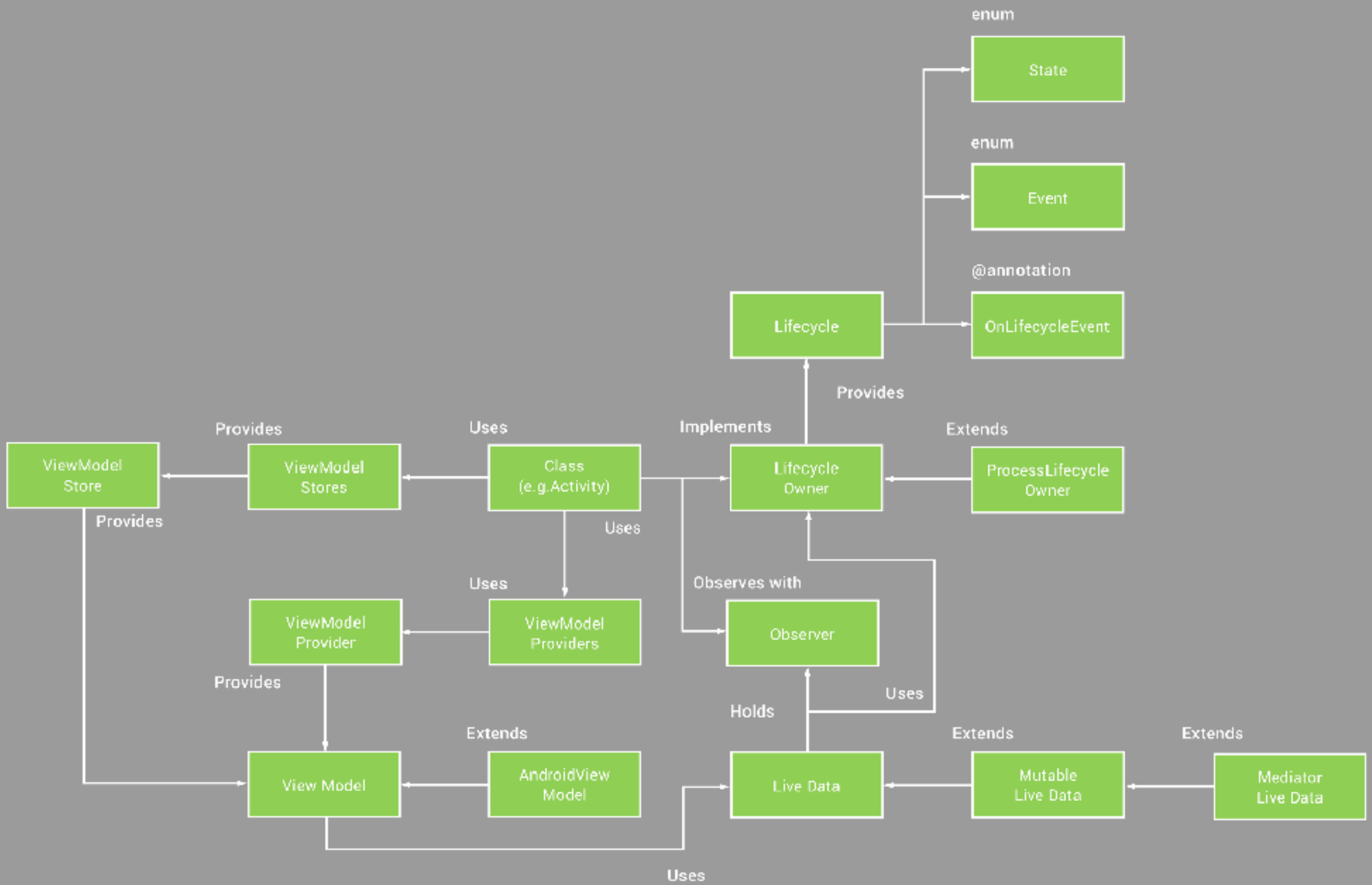


MORE PROBLEMS…

UNNECESSARY NETWORK CALLS

INTRODUCING ARCHITECTURE COMPONENTS

- Lifecycle aware components
- Lifecycle aware data
- Mechanism to survive configuration changes
- An SQL Lite ORM

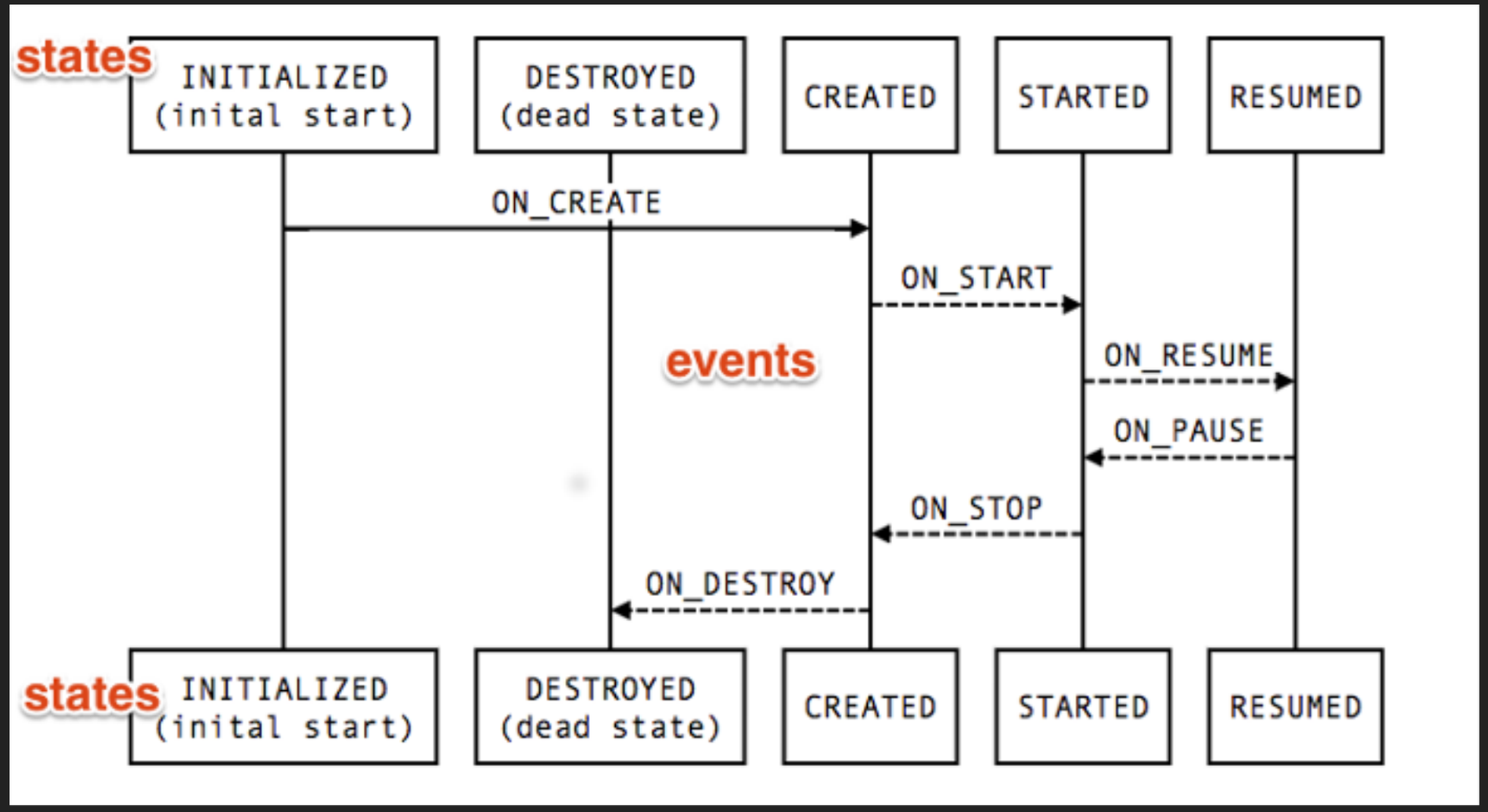

enum
State
enum
Event
@annotation
OnLifecycleEvent
Lifecycle
Provides
Implements
Extends
Provides
Uses
ViewModel Store
ViewModel Stores
Class (e.g.Activity)
Lifecycle Owner
ProcessLifecycle Owner
Provides
Uses
Uses
Observes with
ViewModel Provider
ViewModel Providers
Observer
Provides
Uses
Holds
Extends
Extends
Extends
View Model
AndroidView Model
Live Data
Mutable Live Data
Mediator Live Data
Uses

LIFECYCLE / LIFECYCLEOWNER

- Classes that have lifecycle: Activity / Fragment
- LifecycleOwner classes:
  - LifecycleActivity
  - LifecycleFragment
- Others:
  - LifecycleService
  - ProcessLifecycleOwner
  - Make your own LifecycleOwner

LIFECYCLE OBSERVER

- Observes LifecycleOwner class
- Lifecycle events / states



```
abstract class BaseActivity : LifecycleActivity() {

    @Inject
    lateinit var debugLifecycleObserver: DebugLifecycleObserver;

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        AndroidInjection.inject(act)
        lifecycle.addObserver(debugLifecycleObserver)
        debugLifecycleObserver.activityName = localClassName.toString()
    }
}
```

```
class DebugLifecycleObserver : LifecycleObserver {

    var activityName: String = "";

    @OnLifecycleEvent(Lifecycle.Event.ON_CREATE)
    fun onCreate() {
        Timber.d(activityName + ".onCreate() has been called")
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_DESTROY)
    fun onDestroy() {
        Timber.d(activityName + ".onDestroyed() has been called")
    }

}
```

LIVE DATA

- Observable…
  - With lifecycle awareness
    - *LiveData.observe(Lifecycle owner, Observer<T> observer)*
- RxJava Flowable<T> support (LiveDataReactiveStreams)
  - (*Implementers of org.reactivestreams.Publisher<T>*)
- Transformations
- LiveData vs MutableLiveData vs MediatorLiveData

Enriched data from the network

```
class Resource<T>(val resourceStatus: ResourceStatus, val data: T?, val message: String?) {
    companion object {

        fun <T> loading(msg: String? = null): Resource<T> {
            return Resource(ResourceStatus.LOADING, null, msg)
        }

        fun <T> success(data: T): Resource<T> {
            return Resource(ResourceStatus.SUCCESS, data, null)
        }

        fun <T> error(msg: String?): Resource<T> {
            return Resource(ResourceStatus.ERROR, null, msg)
        }

    }
}
```

```
enum class ResourceStatus {
    LOADING,
    SUCCESS,
    ERROR
}
```

Live Data to be observed...

```
class ForecastViewModel
@Inject
constructor(private val darkSkyRepository: DarkSkyRepository) : ViewModel() {

    fun getForecast(latitude: Double, longitude: Double): LiveData<Resource<Forecast>>
```

Observing the data in the activity...

```
forecastViewModel.getForecast(location.lat!!, location.lng!!)
        .observe(this, Observer<Resource<Forecast>> { resource ->
            when (resource!!.resourceStatus) {
                ResourceStatus.SUCCESS -> updateUI(resource.data!!)
                ResourceStatus.ERROR -> Toast.makeText(this@ForecastActivity, "No forecast!!! :(",
                ResourceStatus.LOADING -> Toast.makeText(this@ForecastActivity, "Loading!", Toast.
            }
        })
```

API definition...using Flowable<T>


```
interface GoogleMapsApi {

    companion object {
        const val BASE_URL: String = "https://maps.googleapis.com/maps/api/";
    }

    @GET("place/textsearch/json")
    fun getPlace(@Query("query") query: String,
                 @Query("type") type: String,
                 @Query("key") key: String): Flowable<PlaceResponse>

}
```

RxJava & LiveData

```
fun <T> Flowable<T>.enqueueToResource
        (errorMessage: String?, loadingMessage: String?): MutableLiveData<Resource<T>> {
    val result = MutableLiveData<Resource<T>>()
    result.value = Resource.loading(loadingMessage)
    this.subscribeOn(Schedulers.newThread()).observeOn(AndroidSchedulers.mainThread()).subscribe(
            { response -> result.value = Resource.success(response); },
            { e -> result.value = Resource.error(errorMessage ?: e.toString()) })
    return result
}
```

Transforming flowable directly into LiveData...


```
fun <T> Flowable<T>.enqueueToResource
        (errorMessage: String?, loadingMessage: String?): MutableLiveData<Resource<T>> {
    val safeFlowable =
            this.subscribeOn(Schedulers.newThread())
                    .observeOn(AndroidSchedulers.mainThread())
                    .doOnError {}
    val liveData = LiveDataReactiveStreams.fromPublisher(safeFlowable)
    val result = MediatorLiveData<Resource<T>>()
    result.value = Resource.loading(loadingMessage)
    result.addSource(liveData) { emittedValue ->
        result.value =
                if (emittedValue is T) Resource.success(emittedValue)
                else Resource.error(errorMessage ?: emittedValue.toString());
    }
    return result
}
```

Using the extension method...

```
class GoogleMapsRepository
@Inject
constructor(private val googleMapsApi: GoogleMapsApi) {


    fun getLocality(query: String): MutableLiveData<Resource<PlaceResponse>> =
            googleMapsApi
                    .getPlace(query, "locality", BuildConfig.GOOGLE_PLACES_API_KEY)
                    .enqueueToResource()

}
```

Transformations

```
@PaperParcel
data class ForecastData(
        val latitude: Double,
        val longitude: Double,
        val timezone: String?,
        val offset: Int?,
        val currently: DataPoint?,
        val minutely: DataBlock?,
        val hourly: DataBlock?,
        val daily: DataBlock?,
        val flags: Flags?) : PaperParcelable {
    companion object {
        @JvmField val CREATOR = PaperParcelForecastData.CREATOR
    }
}

@PaperParcel
data class DataBlock(
        val summary: String?,
        val icon: String?,
        val data: List<DataPoint>?): PaperParcelable {
    companion object {
        @JvmField val CREATOR = PaperParcelDataBlock.CREATOR
    }
}

@PaperParcel
data class DataPoint(
        val time: Int,
        val summary: String?,
        val icon: String?,
        val sunriseTime: Int?,
        val sunsetTime: Int?,
        val moonPhase: Double?,
        val precipAccumulation: Double?,
        val precipIntensity: Double?,
        val precipIntensityMax: Double?,
        val precipIntensityMaxTime: Int?,
        val precipProbability: Double?,
        val precipType: String?,
        val temperature: Double?,
        val temperatureMin: Double?,
        val temperatureMinTime: Int?,
        val temperatureMax: Double?,
        val temperatureMaxTime: Int?,
        val apparentTemperature: Double?,
        val apparentTemperatureMin: Double?,
        val apparentTemperatureMinTime: Int?,
        val apparentTemperatureMax: Double?,
        val apparentTemperatureMaxTime: Int?,
        val dewPoint: Double?,
        val humidity: Double?,
        val windSpeed: Double?,
        val windBearing: Int?,
        val visibility: Double?,
        val cloudCover: Double?,
        val pressure: Double?,
        val ozone: Double?,
        val nearestStormBearing: Int?,
        val nearestStormDistance: Double?): PaperParcelable {
    companion object {
        @JvmField val CREATOR = PaperParcelDataPoint.CREATOR
    }
}
```

->

```
data class Forecast(
        val timestamp: Double,
        val iconId: String,
        val description: String,
        val temperature: Double,
        val windSpeed: Double,
        val prediction: String)
```

Another extension function...

```
fun ForecastData.toForecast(): Forecast =
        Forecast(
                timestamp = this.currently?.time?.toDouble()!!,
                iconId = this.currently.icon!!,
                description = this.currently.summary!!,
                temperature = this.currently.temperature!!,
                windSpeed = this.currently.windSpeed!!,
                prediction = this.daily?.summary!!)
```

Applying a transformation function

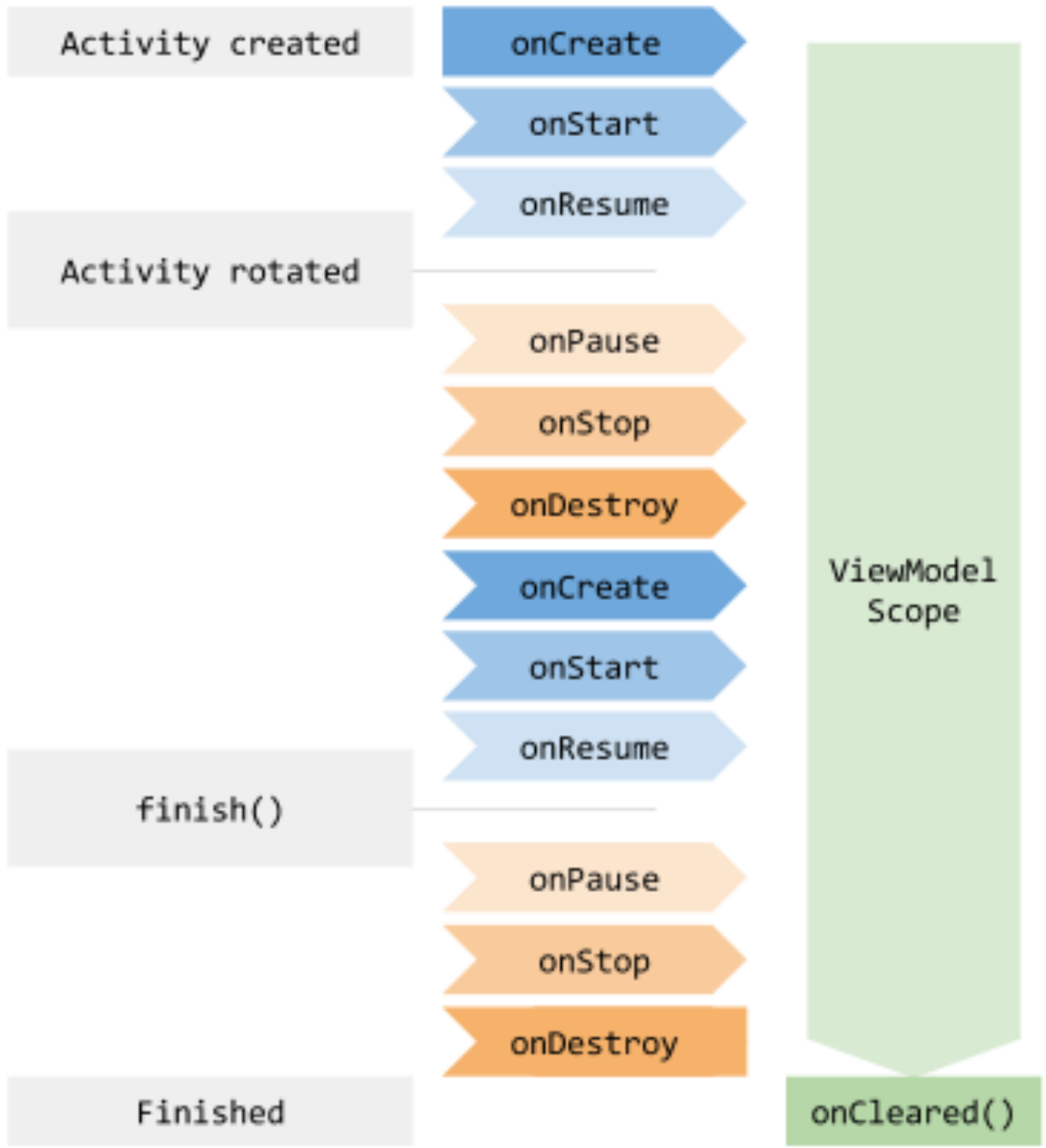
```
class DarkSkyRepository
@Inject
constructor(private val darkSkyApi: DarkSkyApi) {

    companion object {
        private const val UNITS_CA = "ca"
    }

    fun getForecast(latitude: Double, longitude: Double): LiveData<Resource<Forecast>> {
        val remoteData = darkSkyApi
                .getForecast(BuildConfig.DARK_SKY_API_KEY, latitude, longitude, UNITS_CA).enqueueToResource();
        return Transformations.switchMap(remoteData) {
            val forecastLiveData = MutableLiveData<Resource<Forecast>>()
            when (remoteData.value?.resourceStatus) {
                ResourceStatus.SUCCESS -> forecastLiveData.value = Resource.success(remoteData.value?.data?.toForecast()!!)
                ResourceStatus.ERROR -> forecastLiveData.value = Resource.error(remoteData.value?.message!!)
                ResourceStatus.LOADING -> forecastLiveData.value = Resource.loading(remoteData.value?.message)
            }
            return@switchMap forecastLiveData
        }
    }
}
```

SOLVING THE CONFIGURATION CHANGES PROBLEM

- Until now there was no silver bullet for it…
- AsyncTaskLoaders
- Headless fragments
  - Fragment.setRetainInstance(true)
- FragmentActivity.onRetainCustomNonConfigurationInstance
- Singleton in the app
- …

VIEWMODELS

- Class acting as data manager for Activities & Fragments
- Scoping an activity (and its fragments) or just a fragment.
- Expose LiveData objects to be observed from the UI
- NEVER STORE ACTIVITY CONTEXT/VIEWS HERE!


Activity created
onCreate
onStart
onResume
Activity rotated
onPause
onStop
onDestroy
onCreate
onStart
onResume
finish()
onPause
onStop
onDestroy
Finished
ViewModel Scope
onCleared()

VIEWMODELS

- onCleared() if you need to cleanup resources...
- Instantiating ViewModels
  - ViewModelProvider.of(FragmentActivity act)
  - ViewModelProvider.of(Fragment fr)
  - ViewModelProvider.of(FragmentActivity act, Factory ft)
  - ViewModelProvider.of(Fragment fr, Factory ft)
- AndroidViewModel if you need Application access

```
class PlaceViewModel
@Inject
constructor(private val googleMapsRepository: GoogleMapsRepository) : ViewModel() {

    private var currentSearch: String? = null;
    private var places: MutableLiveData<Resource<PlaceResponse>>? = null

    fun getPlaces(place: String): MutableLiveData<Resource<PlaceResponse>> {
        if (places == null || place != currentSearch) {
            currentSearch = place;
            places = googleMapsRepository.getLocality(place);
        }
        return places!!;
    }

}
```

```
lateinit var placeViewModel: PlaceViewModel

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_start)
    placeViewModel = ViewModelProviders.of(this, viewModelFactory).get(PlaceViewModel::class.java)
    setupUI()
}
```

ROOM

- Android team take on ORM
- Major components:
  - Database
  - Entity
  - DAO (Data Access Objects)

ROOM

- Syntax similar to Retrofit (Java Persistence Query Language)
- PrimaryKey, ForeignKey, Index, Nested Objects…
- Insert, Update, Delete, Query…
- LiveData queries
- RxJava (Flowable) queries
- TypeConverters
- Migrations

The Database


```
@Database(entities = arrayOf(Place::class), version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun ForecastDao(): ForecastDao
}
```

The entity, plus a nested object

```
@Entity
data class Place(
        @PrimaryKey val id: String,
        val name: String,
        val latitude: Double,
        val longitude: Double,
        @Embedded var forecast: Forecast? = null)
```

```
data class Forecast(
        val timestamp: Double,
        val iconId: String,
        val description: String,
        val temperature: Double,
        val windSpeed: Double,
        val prediction: String)
```

DAO

```
@Dao
interface ForecastDao {

    @Query("SELECT * FROM place where id = :p0")
    fun getPlace(id:String) : LiveData<Place>

    @Insert(onConflict = REPLACE)
    fun insertPlace(place: Place)

}
```

Usage

```
class ForecastViewModel
@Inject
constructor(private val darkSkyRepository: DarkSkyRepository,
            private val database: AppDatabase) : ViewModel() {

    fun saveForecast(place: Place, forecast: Forecast) {
        place.forecast = forecast
        AsyncTask.execute({ database.ForecastDao().insertPlace(place) })
    }
```

ADDITIONAL NOTES

- Architecture Components is in Alpha
  - Breaking changes expected
  - Not production ready
- Fragment and ActivityCompat in the support library does not yet implement LifecycleOwner (it will by 1.0)

WHERE TO GO FROM HERE...

- Code shown on the presentation https://github.com/jospint/Architecture-Components-DroidDevs
- Google I/O talks (intro, lifecycle, persistence)
- Google samples (3 demos) (link)
- Google Guide to App Architecture (link)
- Exploring the new Android Architecture Components Library by Joe Birch (link)
- Lifecycle Aware Data Loading with Architecture Components by Ian Lake (link)
- And others (link, link, link, link...)

EXTRA: KOTLIN RESOURCES

- Antonio Leiva “Kotlin for Android Developers” (link)
- https://github.com/antoniolg/Bandhook-Kotlin
- kotlinlang Slack group (link)
- Kotlin Weekly (link)
- Google it!

DEMO APP

THANK YOU