Repository Pattern for xAPI & Beyond

Contributors:
Arda Kucukoz

Agenda

- What is the Repository Pattern
- Why we need Repository Pattern
- How to implement a Repository Pattern

Android Caching Library

- Introduced the idea of Repository Pattern
- Covers various different scenarios
  - CacheThenRemoteStore
  - CacheThenRemoteWithFallbackStore
  - RemoteWithFallbackStore

GitHub Repo: https://github.ford.com/fordpass/AndroidCacheLibrary

What is Repository Pattern?

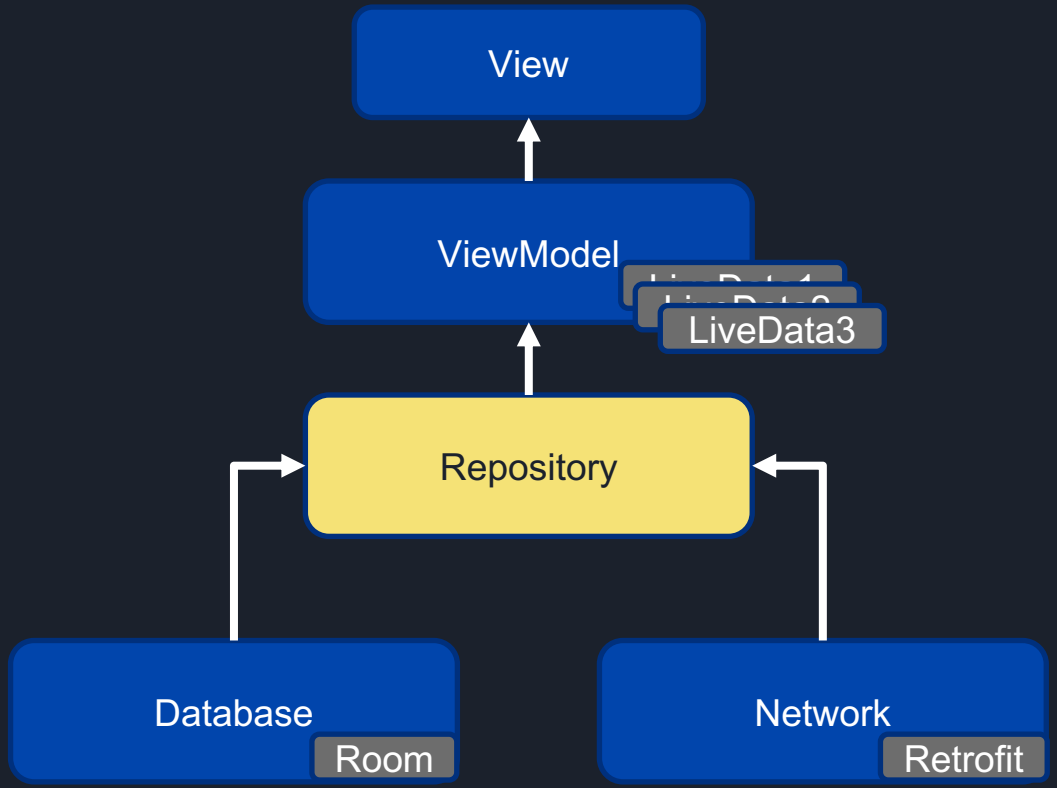
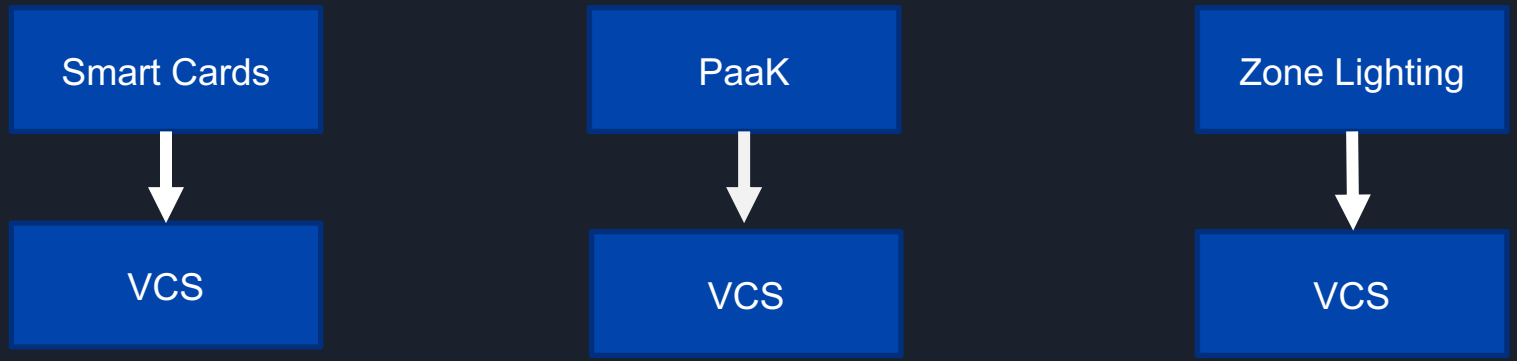


Image source: https://developer.android.com/topic/libraries/architecture/images/final-architecture.png

Why do we need Repository Pattern?

- Avoid multiple network calls
- Cache the last success response
- Update UI faster



Multiple Network Calls

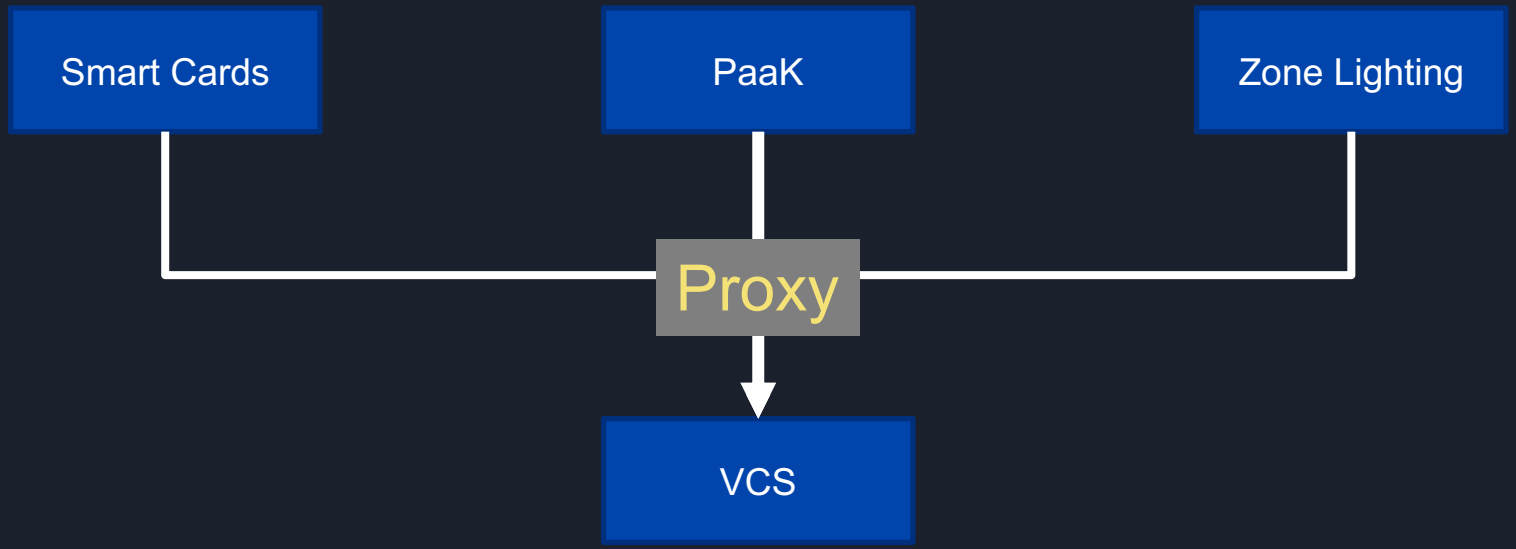
```
//Launches the network call when subscribed
val vcsCall = getVehicleCapabilities(vin)


val SmartCards = vcsCall.subscribe()      //network call 1
val PaaK   = vcsCall.subscribe()          //network call 2
val TPMS = vcsCall.subscribe()            //network call 3
```

	CONNECT	slack.com			09:25:08	201 ms	166 bytes	Failed
200	GET	usapi.cv.ford.com	/api/vehicles/1FTEW1EPXKFD32088/authstatus?lrdt=05-15-2015		09:23:36	511 ms	1.36 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/1FTEW1EPXKFD32088/authstatus?lrdt=5-15-2015		09:23:37	118 ms	1.36 KB	Complete
200	GET	usapi.cv.ford.com	/api/users/vehicles/1FTEW1EPXKFD32088/detail?lrdt=05-15-2015		09:23:39	121 ms	2.91 KB	Complete
200	GET	usapi.cv.ford.com	/api/users/vehicles/1FTEW1EPXKFD32088/detail?lrdt=05-15-2015		09:23:40	126 ms	2.91 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/v3/1FTEW1EPXKFD32088/status	←	09:23:43	192 ms	5.81 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/v3/1FTEW1EPXKFD32088/status	←	09:23:43	170 ms	5.81 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/1FTEW1EPXKFD32088/authstatus?lrdt=05-15-2015		09:23:43	171 ms	1.36 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/v3/1FTEW1EPXKFD32088/status	←	09:23:44	159 ms	5.81 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/1FTEW1EPXKFD32088/authstatus?lrdt=05-15-2015		09:23:44	151 ms	1.36 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/1FTEW1EPXKFD32088/authstatus?lrdt=05-15-2015		09:23:45	181 ms	1.36 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/1FTEW1EPXKFD32088/authstatus?lrdt=05-15-2015		09:23:45	179 ms	1.36 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/v3/1FTEW1EPXKFD32088/status	←	09:23:45	151 ms	5.81 KB	Complete
200	POST	usapi.cv.ford.com	/api/v1/activealertlist		09:23:46	81 ms	1.72 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/v3/1FTEW1EPXKFD32088/status	←	09:23:46	187 ms	5.81 KB	Complete
200	POST	usapi.cv.ford.com	/api/v1/activealertlist		09:23:47	84 ms	1.72 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/1FTEW1EPXKFD32088/authstatus?lrdt=05-15-2015		09:23:50	69 ms	1.36 KB	Complete
200	POST	usapi.cv.ford.com	/api/v1/activealertlist		09:23:50	86 ms	1.72 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/1FTEW1EPXKFD32088/authstatus?lrdt=05-15-2015		09:23:50	73 ms	1.36 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/v3/1FTEW1EPXKFD32088/status	←	09:23:50	162 ms	5.81 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/v3/1FTEW1EPXKFD32088/status	←	09:23:51	151 ms	5.81 KB	Complete
200	GET	usapi.cv.ford.com	/api/vehicles/v3/1FTEW1EPXKFD32088/status	←	09:23:51	197 ms	5.81 KB	Complete
	CONNECT	userlocation.googleapis.com			09:23:23	163 ms	137 bytes	Failed
	CONNECT	userlocation.googleapis.com			09:23:28	134 ms	137 bytes	Failed

8x

Implementation Steps for Repository Pattern

- Avoid multiple network calls → Share the network with all the subscribers
- Cache the last success response
- Update UI faster



How to avoid multiple network calls?

- Traditional Solution: Subjects can be used as upper stream
- Better Solution: The call can be shared if it is an Observable

Avoid Multiple Calls Using Subject

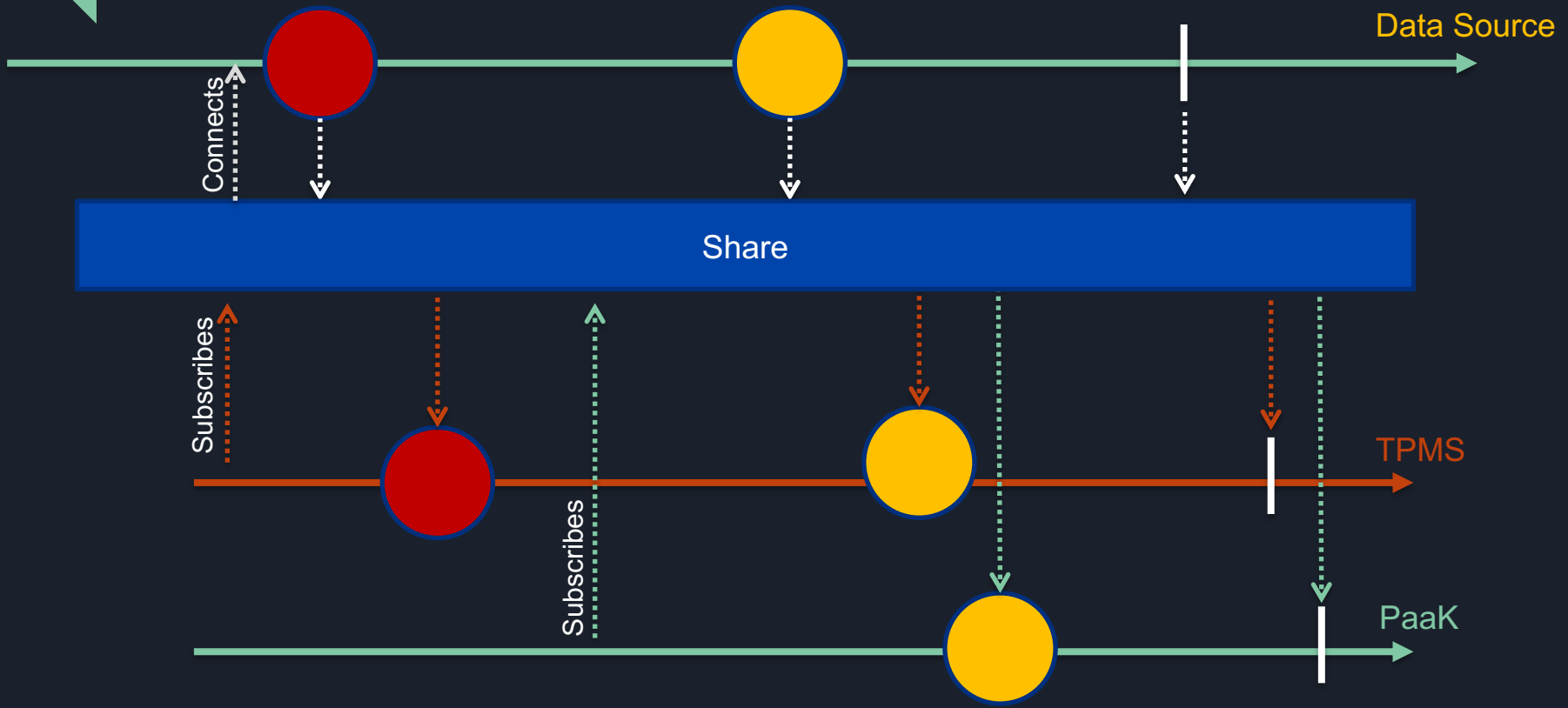
```
val subject = BehaviorSubject.create()

val vcsCall = getVehicleCapabilities(vin).doOnNext { subject.onNext(it) }

val subscriber1 = vcsCall.subscribe()  //emits the same item
val subscriber2 = subject.subscribe() //emits the same item
val subscriber3 = subject.subscribe() //emits the same item
```

DON'T!

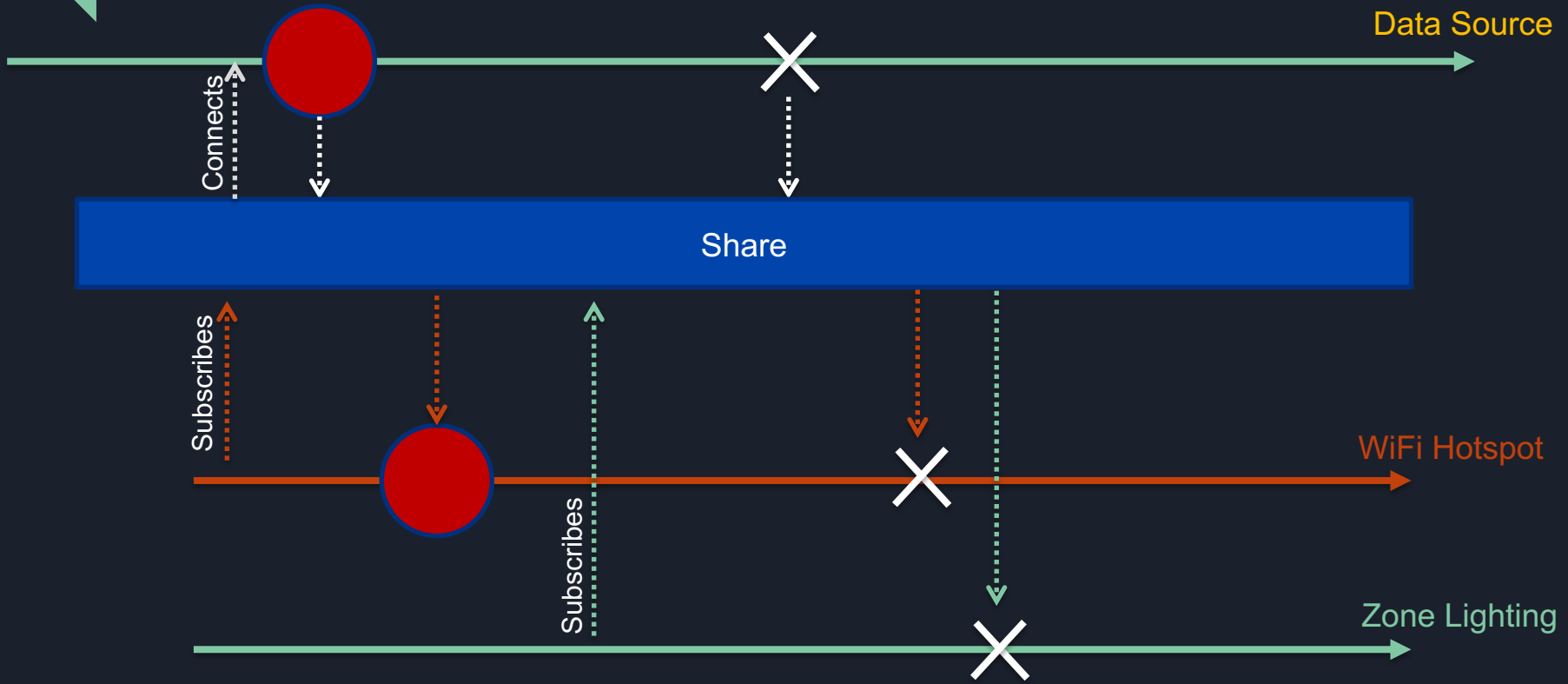
Avoid Multiple Network Call

```
//Shares the emitted item from network call
val vcsCallShare = getVehicleCapabilities(vin).share()


val subscriber1 = vcsCallShare.subscribe() //emits the same item
val subscriber2 = vcsCallShare.subscribe() //emits the same item
val subscriber3 = vcsCallShare.subscribe() //emits the same item
```

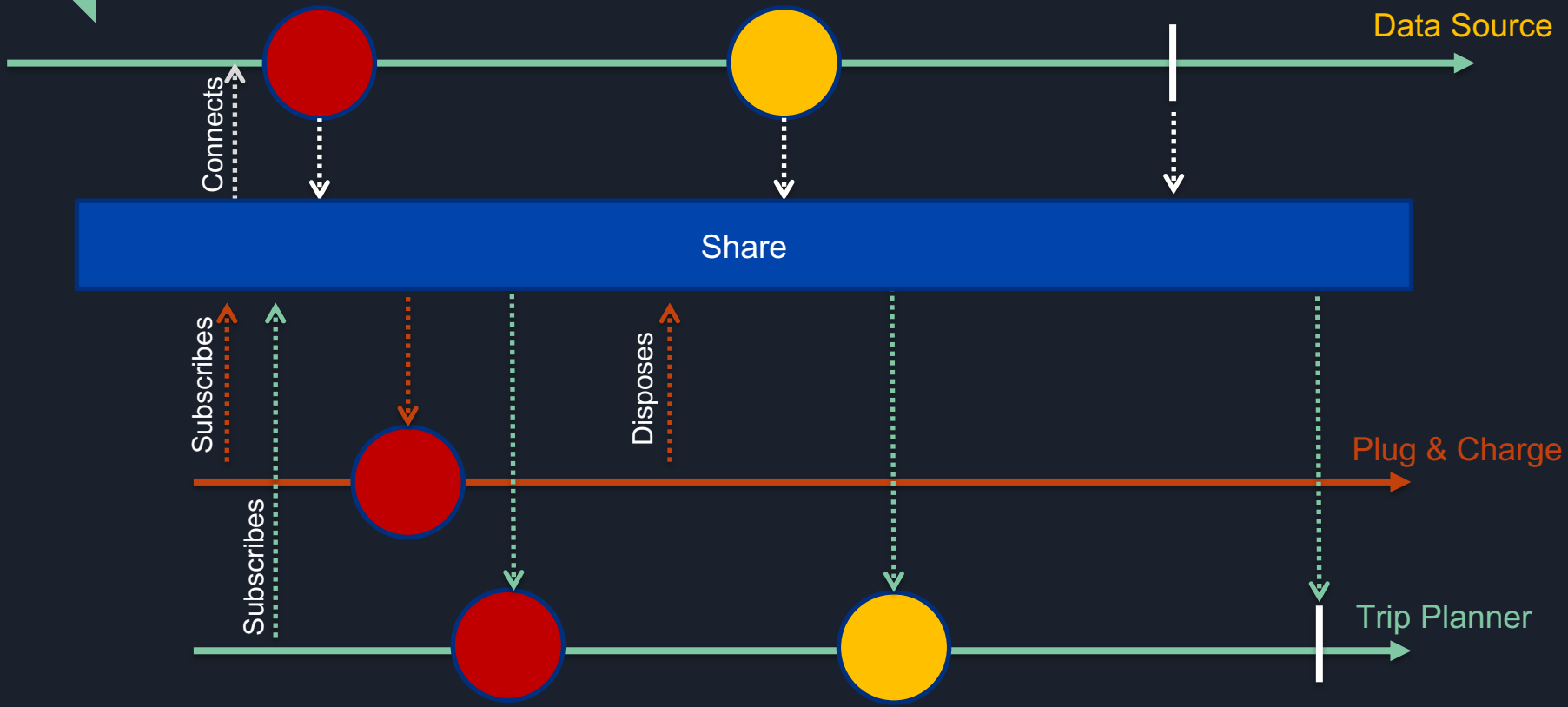
How ‘Share’ Works?



How ‘Share’ Works?



How ‘Share’ Works?

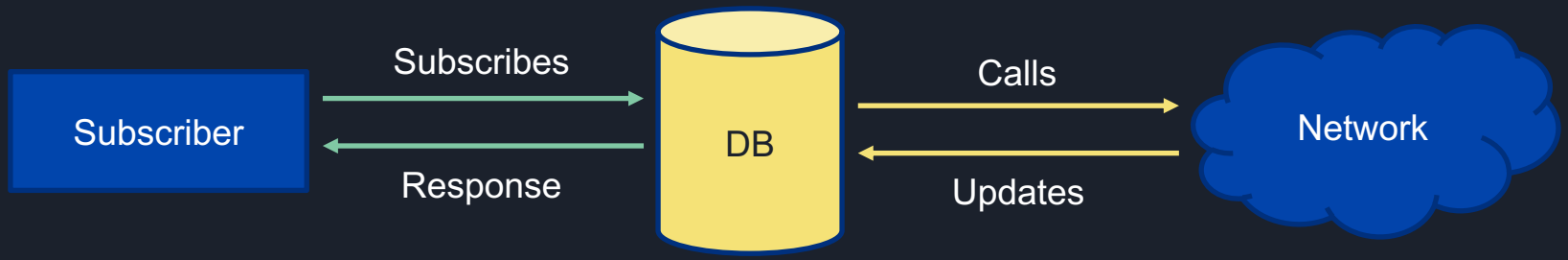


Where We Are?

```
val vcsCallShare = getVehicleCapabilities(vin).share()


val subscriber1 = vcsCallShare.subscribe()
val subscriber2 = vcsCallShare.subscribe()
val subscriber3 = vcsCallShare.subscribe()
```

Implementation Steps for Repository Pattern

- Avoid multiple network calls
- Cache the last success response → Network call updates database
- Update UI faster

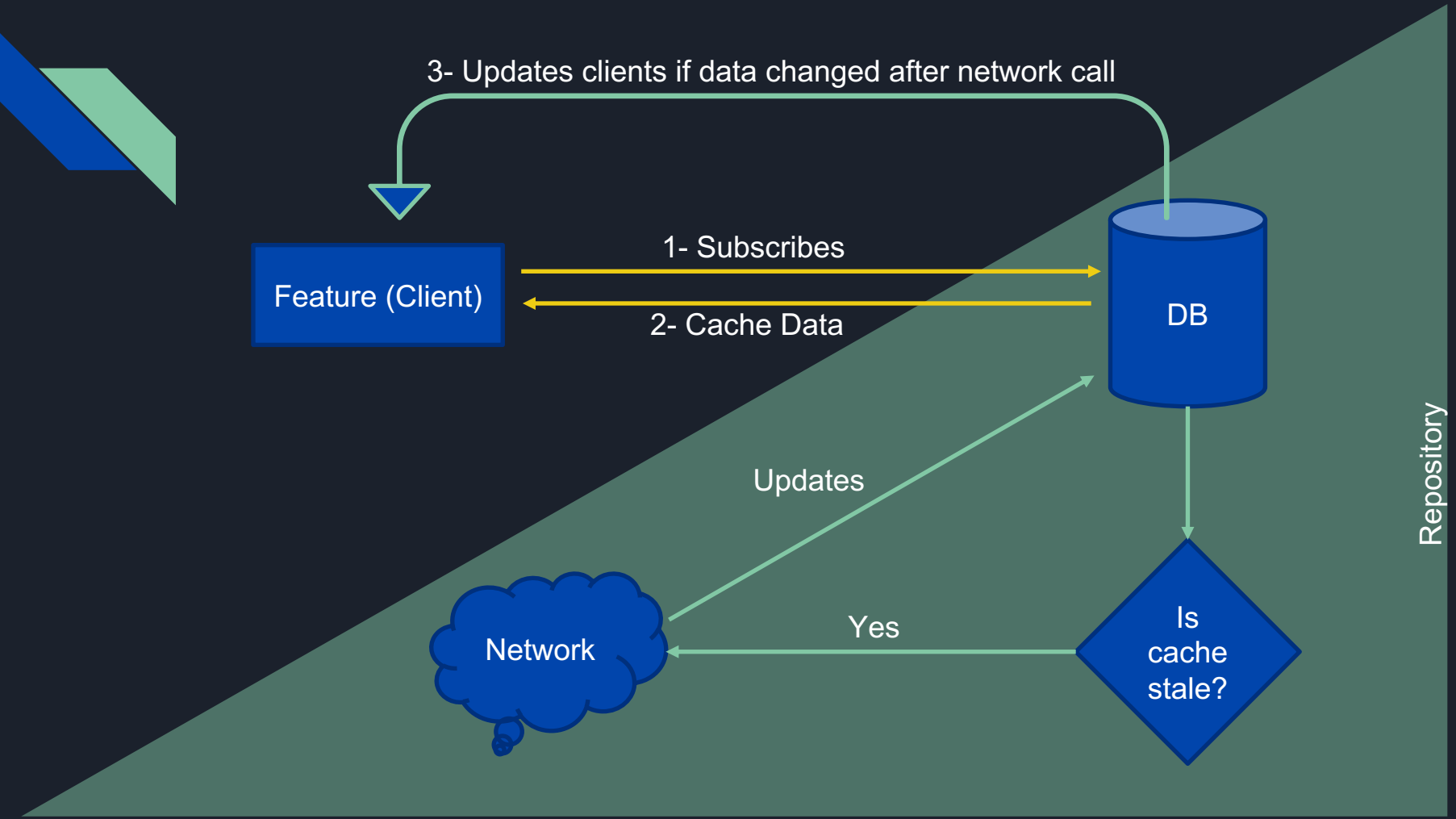


Why do we need DB?

- Support offline mode for Bluetooth eligible features
- Build UI faster with cached values

Cache The Last Success Response

- Save the result to the database
- Retrieve the cache data first
- Update UI if necessary


3- Updates clients if data changed after network call
Feature (Client)
1- Subscribes
2- Cache Data
DB
Updates
Network
Yes
Is cache stale?
Repository

Cache The Last Success Response

```
val vcsCallShare = getVehicleCapabilities(vin)
        .doOnNext { response -> saveToDatabase(response) }
        .share()


val subscriber1 = vcsCallShare.subscribe()
val subscriber2 = vcsCallShare.subscribe()
val subscriber3 = vcsCallShare.subscribe()
```

Subscribe To Database Observable

```
val vcsCallShare = getVehicleCapabilities(vin)
                .doOnNext { response -> saveToDatabase(response) }
                .share()

val databaseCall = databaseItemFlowable(vin) // db observable


val subscriber1 = databaseCall.subscribe()
val subscriber2 = databaseCall.subscribe()
val subscriber3 = databaseCall.subscribe()
```

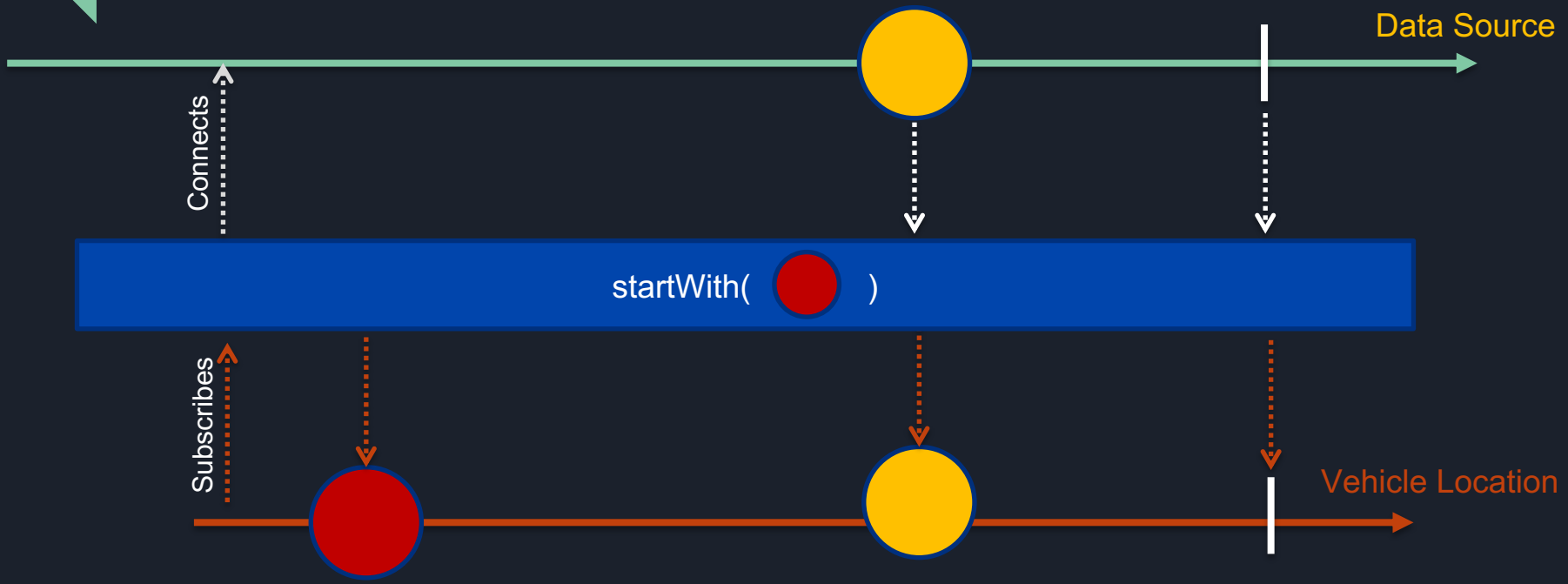
Subscribers Should Trigger The Network Call

```
val vcsCallShare = getVehicleCapabilities(vin)
        .doOnNext { response -> saveToDatabase(response) }
        .share()

val databaseCall = databaseItemFlowable(vin)
        .flatMap { vcsCallShare }

val subscriber1 = databaseCall.subscribe()
val subscriber2 = databaseCall.subscribe()
val subscriber3 = databaseCall.subscribe()
```

Respond With Cached Value First

```
val databaseCall = databaseItemFlowable(vin)
                .flatMap { cacheValue -> vcsCallShare.startWith(cacheValue) }
```

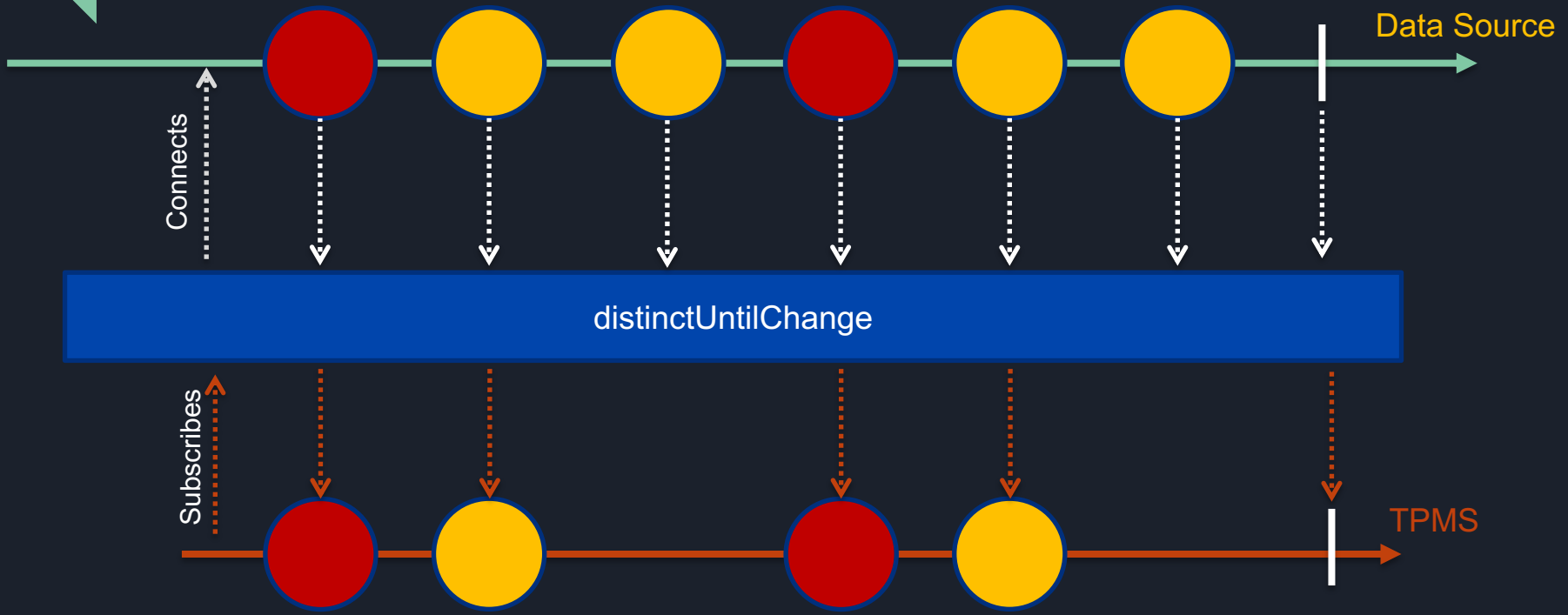
What is `startWith`?



Distinct Response Until Data Changes

```
val databaseCall = databaseItemFlowable(vin)
        .flatMap { cacheValue -> vcsCallShare.startWith(cacheValue) }
        .distinctUntilChanged()
```

What is `distinctUntilChange()`?



Check If the Cache is Valid or Not

```
val databaseCall = databaseItemFlowable(vin)
        .flatMap { cacheValue ->
            if (cacheValue.cacheTime < TTL) vcsCallShare.startWith(cacheValue)
            else Completable.complete().startWith(cacheValue)
        }
        .distinctUntilChanged()
```

Syncronized TTL Check

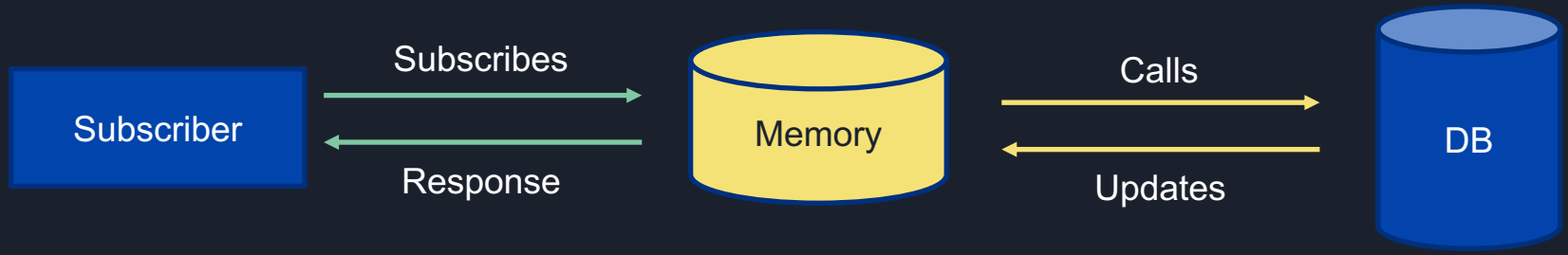
```
val databaseCall = databaseItemFlowable(vin)
        .observeOn (Schedulers.single())
        .flatMap { cacheValue ->
            if (cacheValue.cacheTime < TTL) vcsCallShare.startWith(cacheValue)
            else Completable.complete().startWith(cacheValue)
        }
        .observeOn (Schedulers.computation())
        .distinctUntilChanged()
```

Where We Are?

```
val vcsCallShare = getVehicleCapabilities(vin)
        .doOnNext { response -> saveToDatabase(response) }
        .share()

val databaseCall = databaseItemFlowable(vin)
        .observeOn (Schedulers.single())
        .flatMap { cacheValue ->
            if (cacheValue.cacheTime < TTL) vcsCallShare.startWith(cacheValue)
            else Completable.complete().startWith(cacheValue)
         }
        .observeOn (Schedulers.computation())
        .distinctUntilChanged()
```

Implementation Steps for Repository Pattern

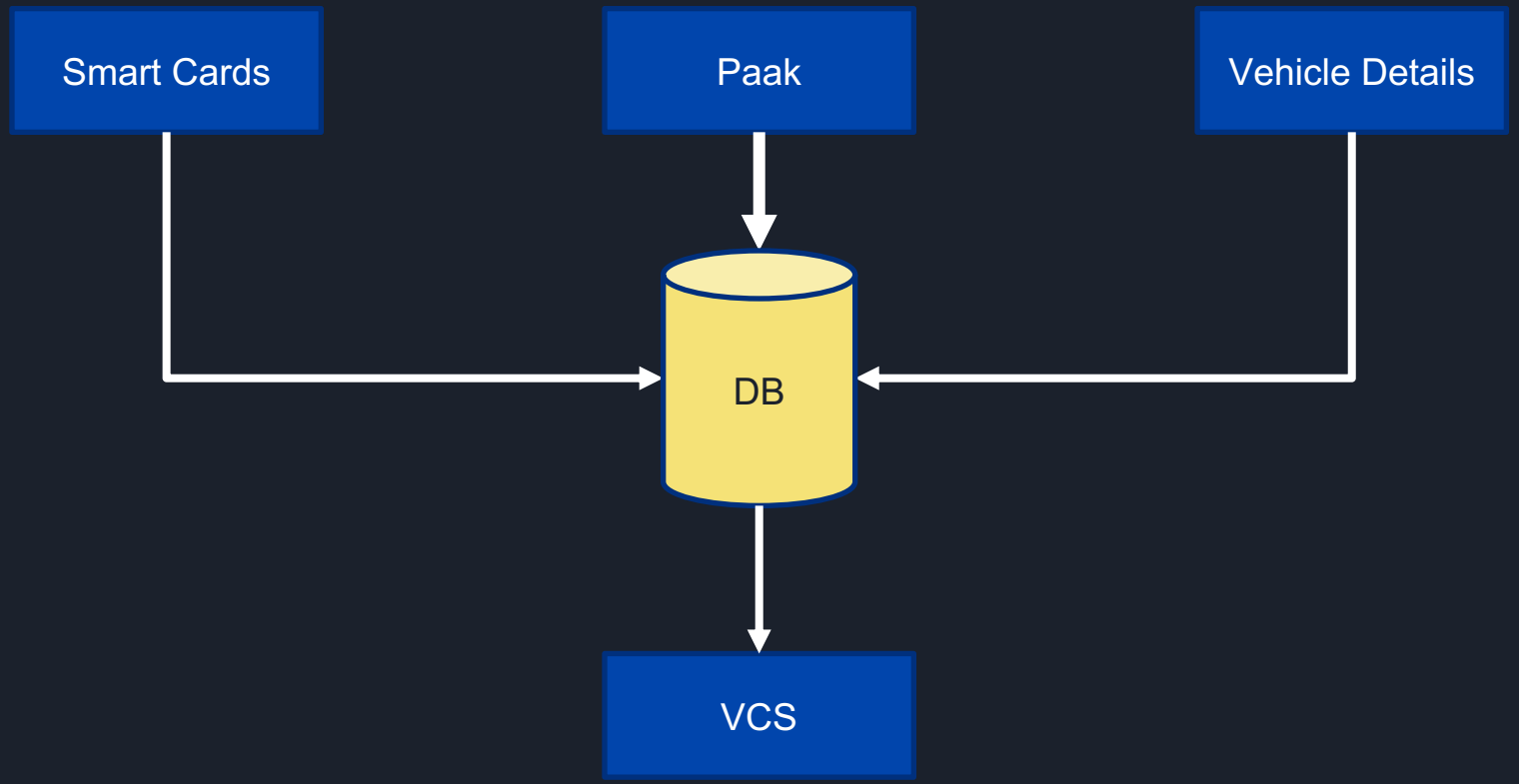
- Avoid multiple network calls
- Cache the last success response
- Update UI faster → Memory Caching Strategy



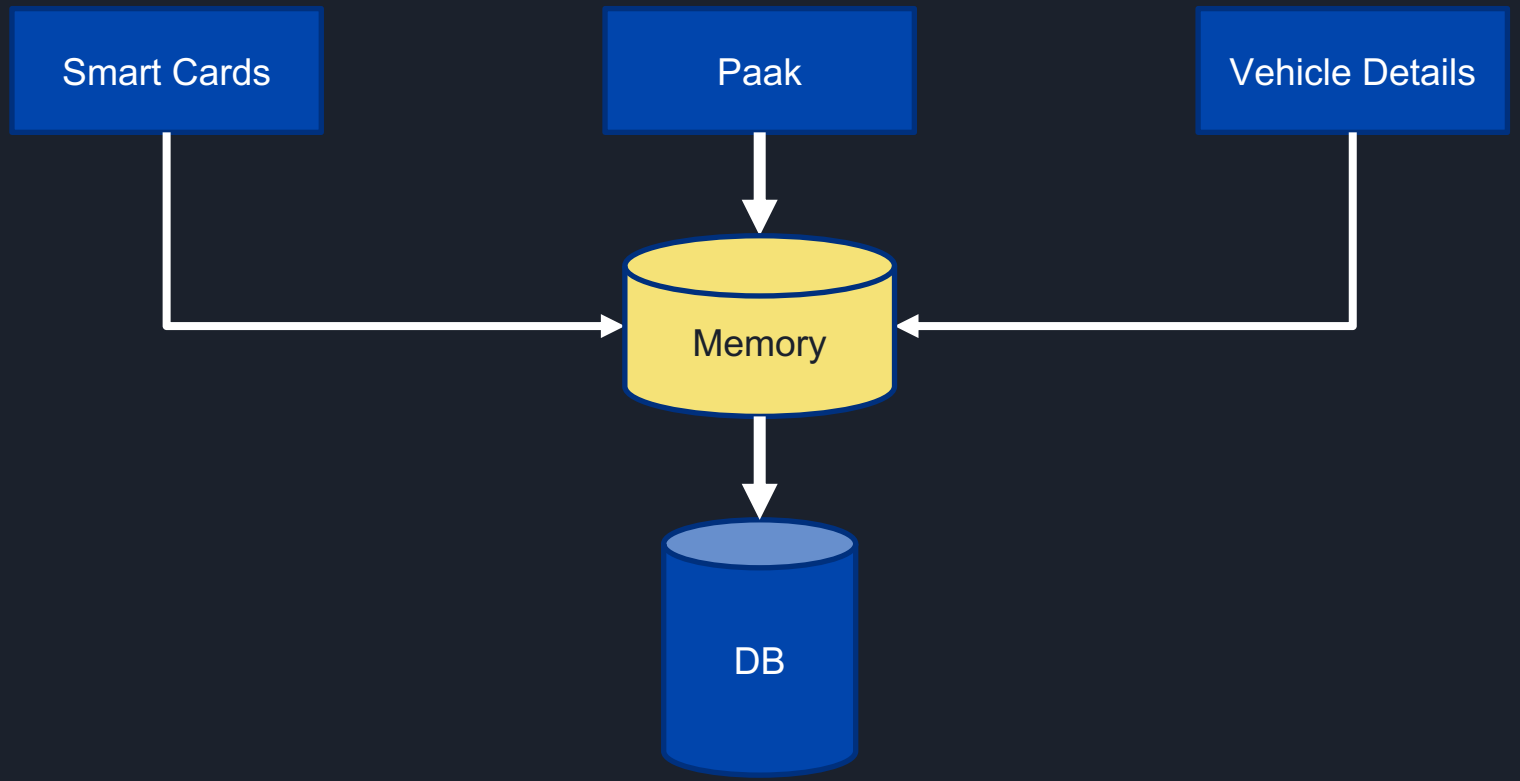
Memory Caching Strategy

- Last emitted item from database can be repeated
- Database can update memory observable when there is a change

Current Repository Implementation SO FAR



After Memory Caching



Implementation of Memory Caching

Before

```
val vcsRepoObservable = databaseItemFlowable(vin)
        .flatMap { cacheValue -> vcsCallShare.startWith(cacheValue) }
        .distinctUntilChanged()
```

After

```
val memoryObservable = databaseItemFlowable(vin).replay(1).refCount()

val vcsRepoObservable = memoryObservable
        .flatMap { cacheValue -> vcsCallShare.startWith(cacheValue) }
        .distinctUntilChanged()
```

Where We Are? (Final)

```
val vcsCallShare = getVehicleCapabilities(vin)
                .doOnNext { response -> saveToDatabase(response) }
                .share()

val memoryObservable = databaseItemFlowable(vin).replay(1).refCount()

val vcsRepoObservable = memoryObservable
                .observeOn (Schedulers.single())
                .flatMap { cacheValue ->
                    if (cacheValue.cacheTime < TTL) vcsCallShare.startWith(cacheValue)
                    else Completable.complete().startWith(cacheValue)
                 }
                .observeOn (Schedulers.computation())
                .distinctUntilChanged()
```

DEMO

https://github.ford.com/AKUCUKO1/RepositoryPatternExample

Insights of Repository Pattern for FP/LW

- Creating single source of truth for all the features
- Decoupling features/viewmodels from each other
- Avoiding to have giant objects like VehicleInfo

REPOSITORIES

- repo-xapi-alerts
- repo-xapi-user-vehicle
- repo-xapi-vehicle-capabilities
- repo-vcs
- repo-telemetry