들어가기 전에

- 플랫폼
  - Android
- 언어
  - Java
  - Kotlin
- 라이브러리
  - RxJava
  - Dagger
- Presenter 레이어 아키텍쳐
  - MVP

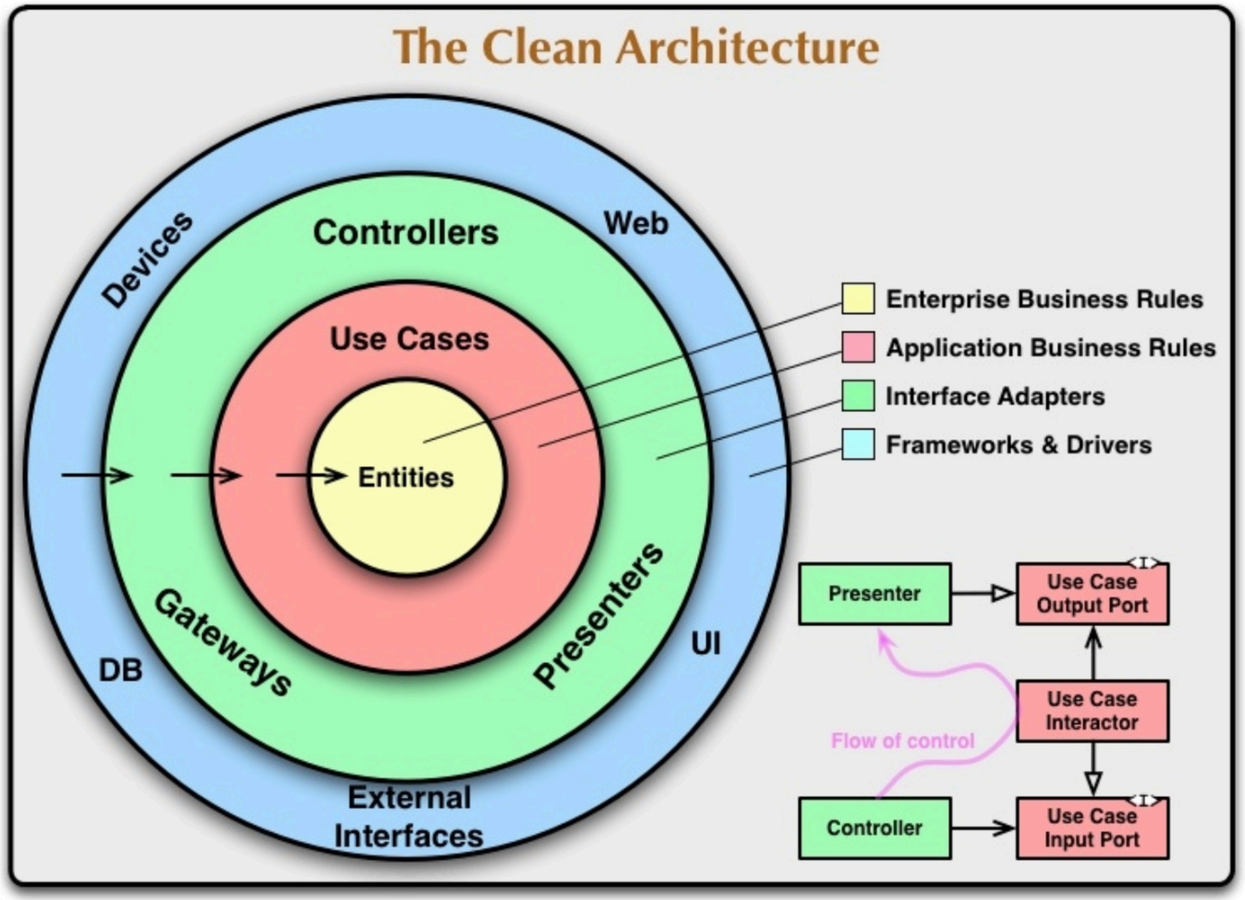
- MVVM?
- MVP?
- MVC?

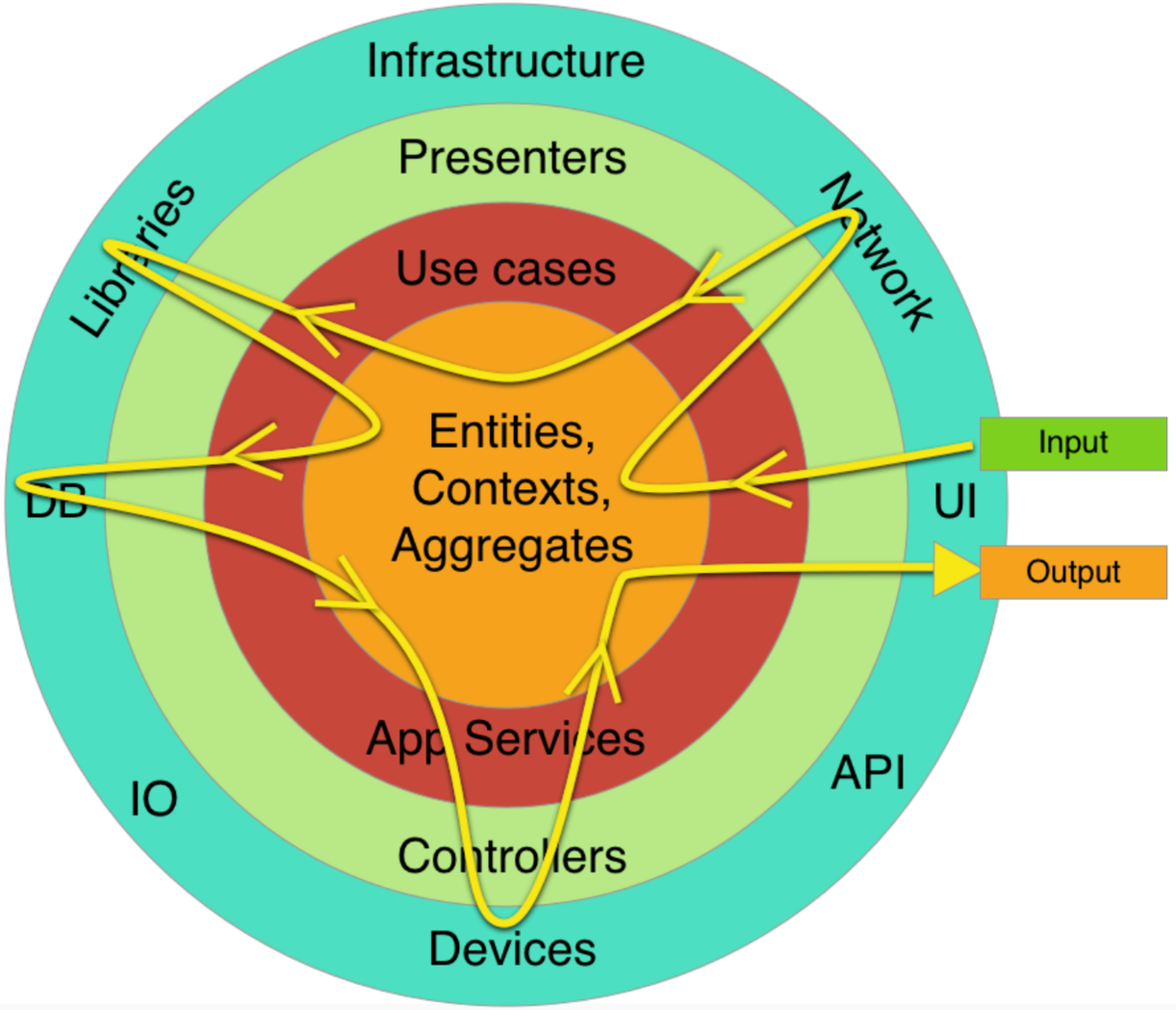
아키텍쳐의 필요성

- 유지보수가 용이한 코드 (변화에 잘 대응할 수 있는 코드)
- (의문점) 클린 아키텍쳐가 왜 유지보수가 용이할까??
- (의문점) 클린 아키텍쳐는 정말 클린한가??

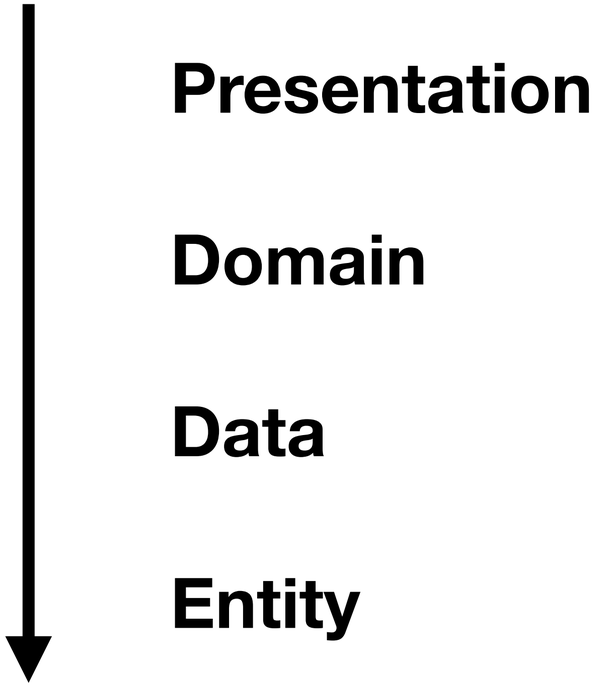
클린 아키텍처

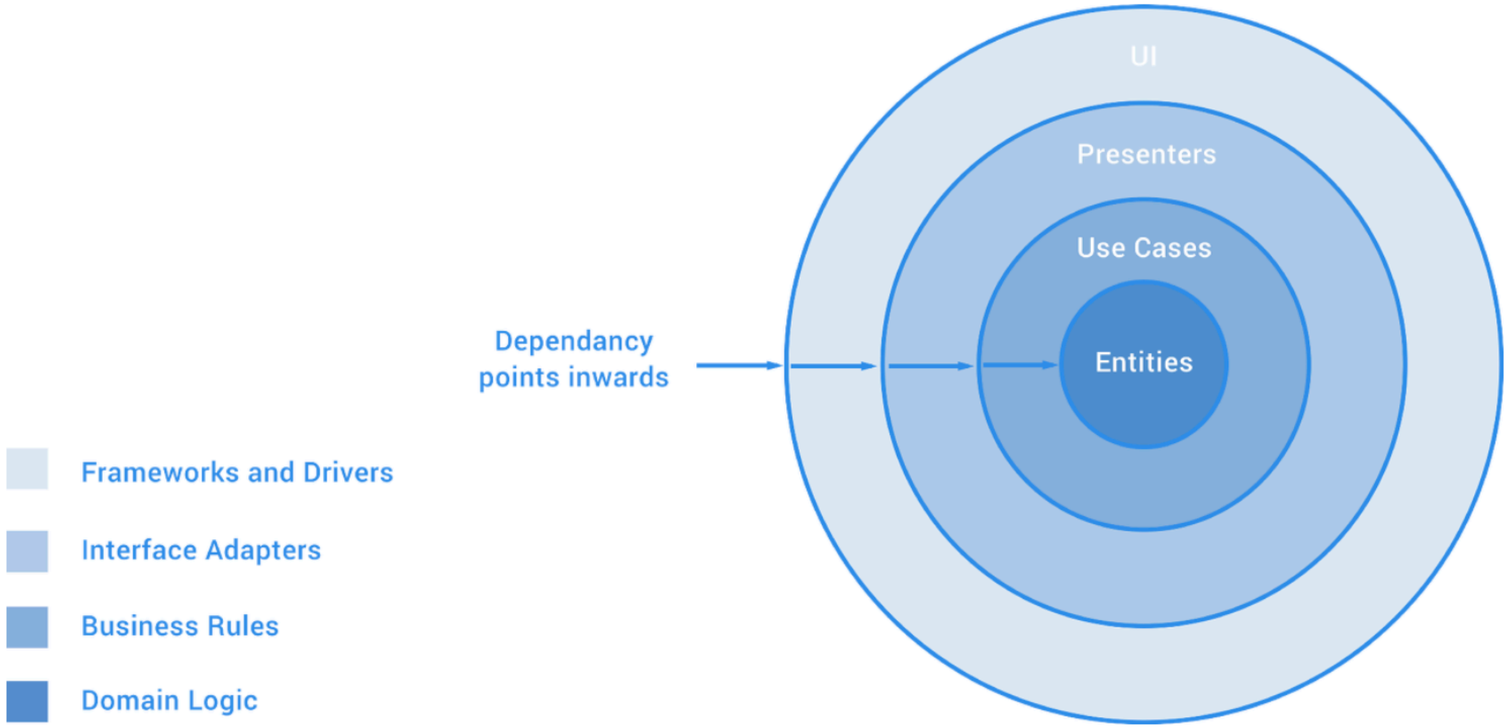
- 프레임워크로부터 독립되어라!
- 세부 구현으로부터 독립되어라!
  - UI, DB, Framework, Library 등 변경에 영향받지 않아야함
- 내부 레이어는 상위 레이어를 알면 안된다.
- 경계 간 데이터 전달은


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port


Infrastructure
Presenters
Use cases
Entities,
Contexts,
Aggregates
App Services
Controllers
Devices
Libraries
Network
DB
UI
IO
API
Input
Output

의존성은 아래로




Frameworks and Drivers
Interface Adapters
Business Rules
Domain Logic
Dependancy
points inwards
UI
Presenters
Use Cases
Entities

Entity Layer

- 순수한 모듈
- 다른 레이어와 의존성이 없음
- 비즈니스 로직에서 파생된 개념
- 서버와 클라이언트가 맞추면 좋은 레이어
  - Ex) 클라이언트의 경우 서버 API 와 맞춘다.
  - Ex) 서버의 경우 API 로 나갈 모델

Data Layer

- Repository 의 구현체를 담고 있다.
- 안드로이드의 경우 DataSource 쪽에서 디바이스 종속성이 생길 수 있음 (Network 호출, Local DB 등)
- Repository 는 Domain 레이어에서 필요한 소스(Model or Data) 들을 어떻게 가져올지 (DataSource 를 선택하여) 를 판단하여 구현

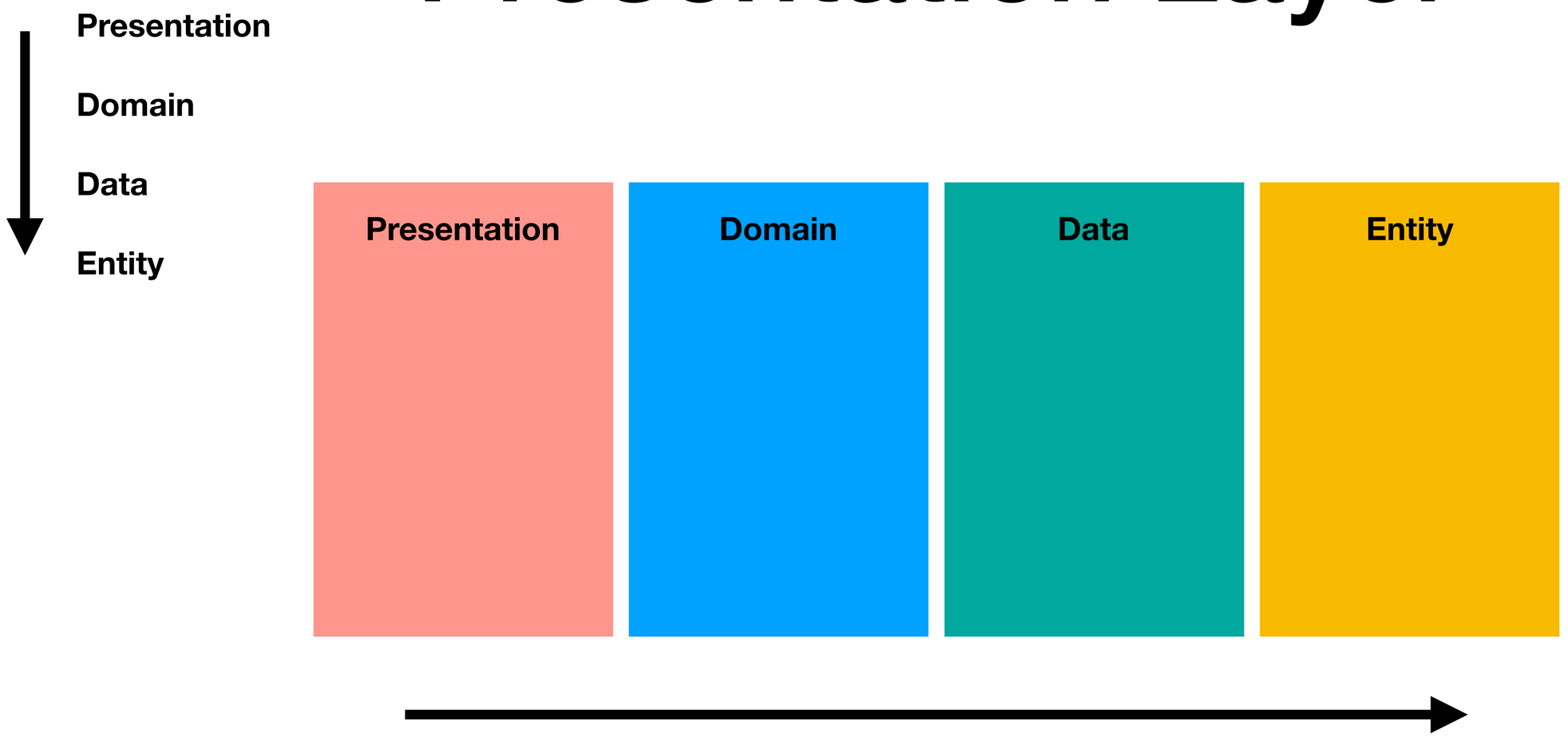
Domain Layer

- 순수 언어 모듈, 안드로이드와 종속성 없음.
- UseCase 를 적용하는 부분 (실제 사용자의 행동)
- Data Layer 에서 Repository 가 적절한 DataSource 를 선택하므로 Domain Layer 는 그런 고민 없이 Repository 를 잘 이용하여 로직을 구축

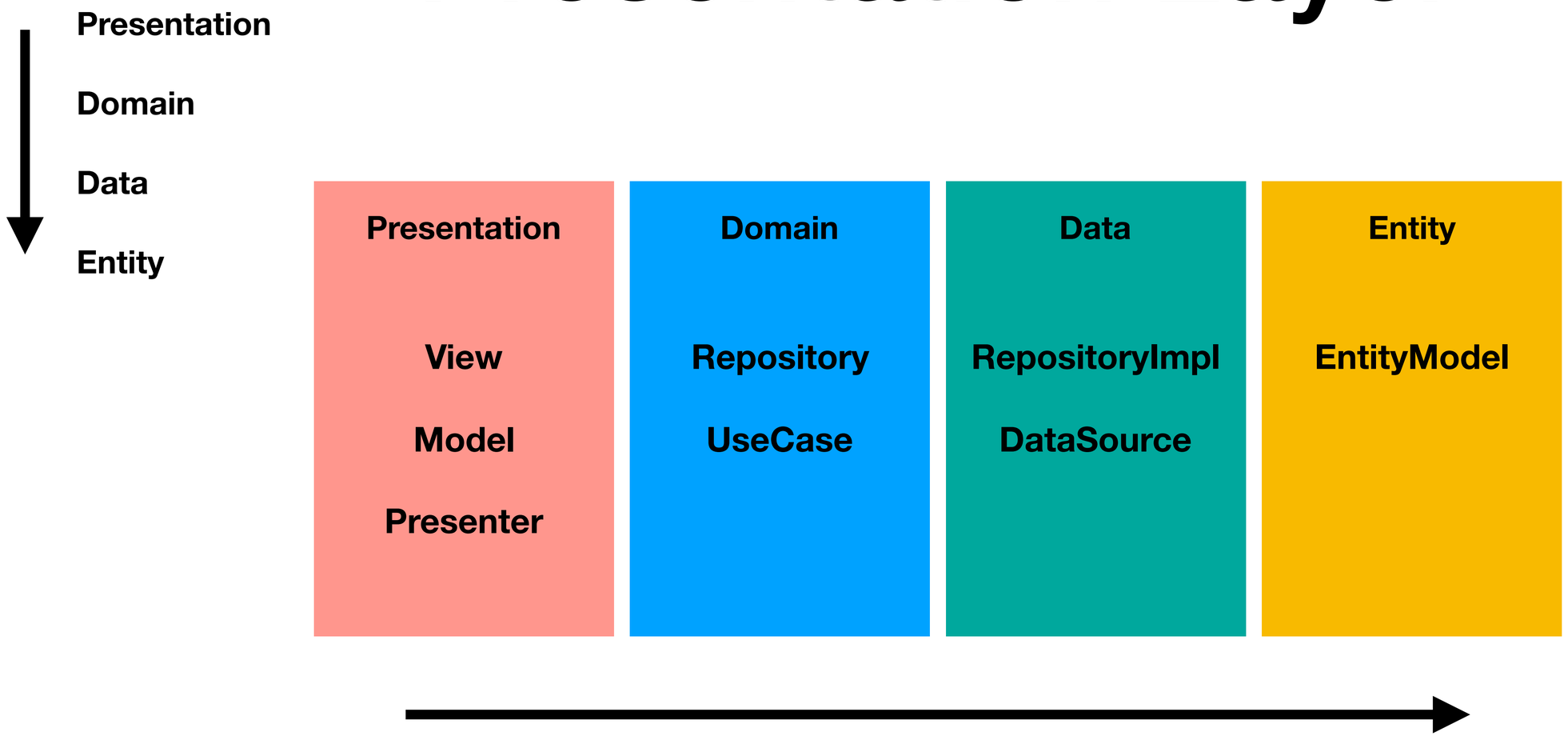
Presentation Layer

- 실질적인 UI 부분
- UseCase를 통해 데이터를 주입받아 사용.
- 필요에 따라 MVP, MVC, MVVM 모델 등을 활용할 수 있다.

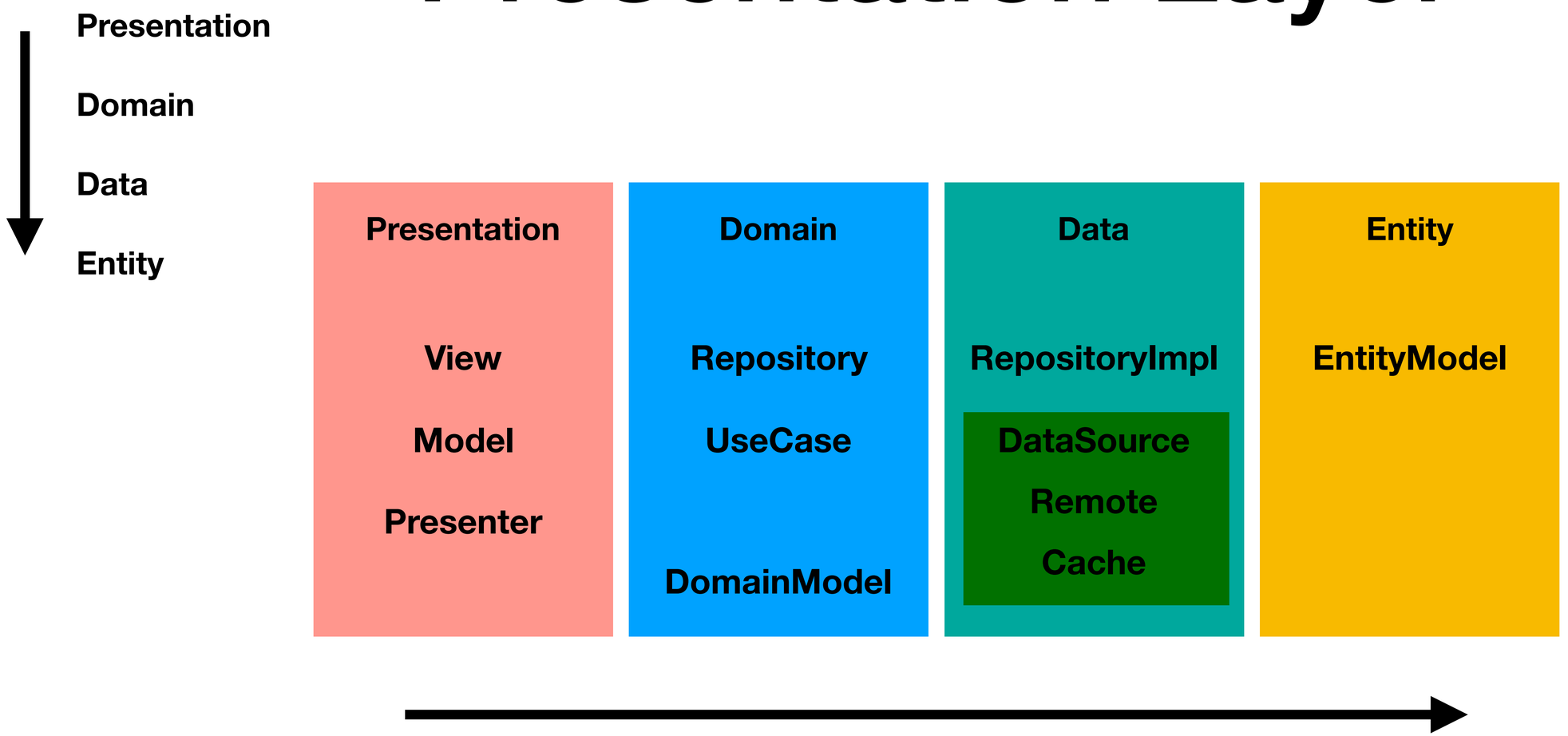
Presentation Layer



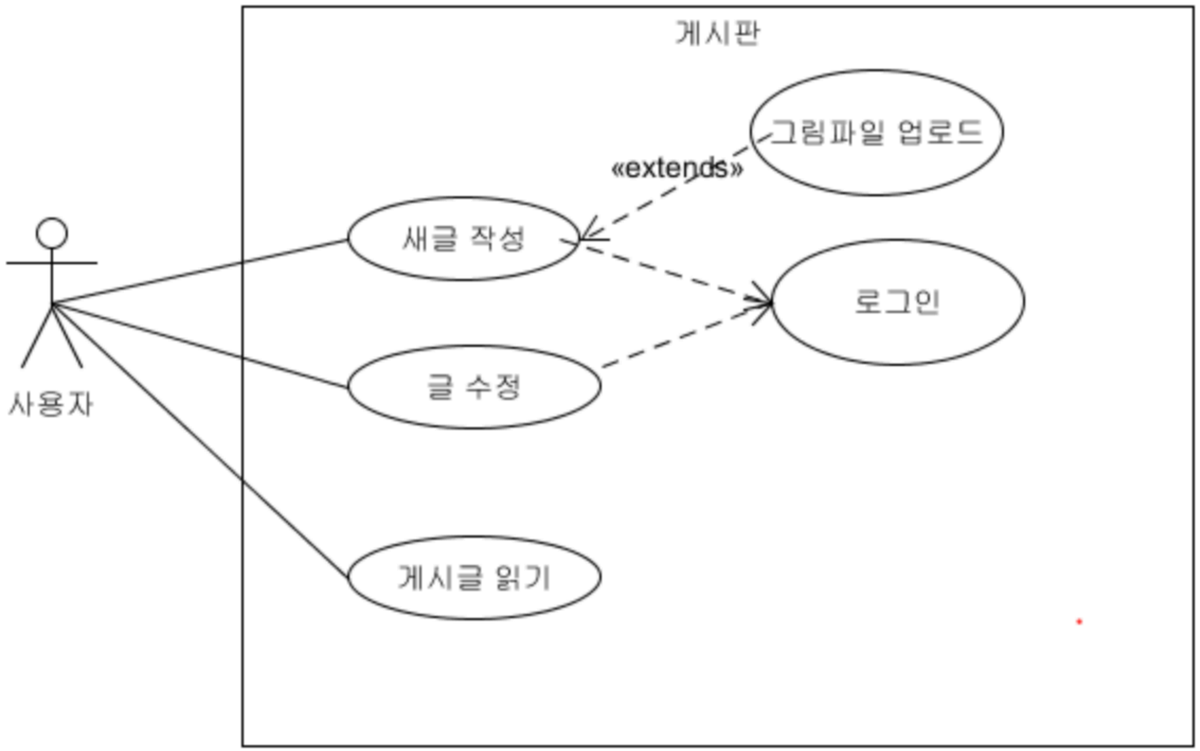
Presentation Layer



Presentation Layer



UseCase



Entity Layer

```
data class EventEntity(
    @SerializedName("eventId") var eventId: Int = 0,
    @SerializedName("name") var name: String? = null,
    @SerializedName("startDate") var startDate: String? = null,
    @SerializedName("location") var location: EventLocation? = null,
    @SerializedName("metadata") var metadata: EventMetaData? = null,
    @SerializedName("hostOrganization") var hostOrganization: EventOrganization? = null
)
```

```
data class EventOrganization(
    @SerializedName("organizationId") var organizationId: Int = 0,
    @SerializedName("name") var name: String? = null,
    @SerializedName("description") var description: String? = null
)
```

Data Layer

```
data class EventListDTO(
    @SerializedName("page") var page: String? = null,
    @SerializedName("pageSize") var pageSize: String? = null,
    @SerializedName("total") var total: Int = 0,
    @SerializedName("rows") var data: List<EventEntity>? = emptyList()

)
```

```
interface EventAPI {
    @GET("/api/v1/events")
    fun getEvents(
        @Query("page") page: Int = 1,
        @Query("pageSize") pageSize: Int = 5,
        @Query("order") order: String = "startDate"
    ): Flowable<EventListDTO>
}
```

```
@Singleton
class EventListRemoteDataSourceImpl @Inject constructor(private val api: EventAPI) : EventListRemoteDataSource {

    override fun getEvents(): Flowable<List<EventEntity>> {
        return api.getEvents()
            .map { it.data }
    }
}
```

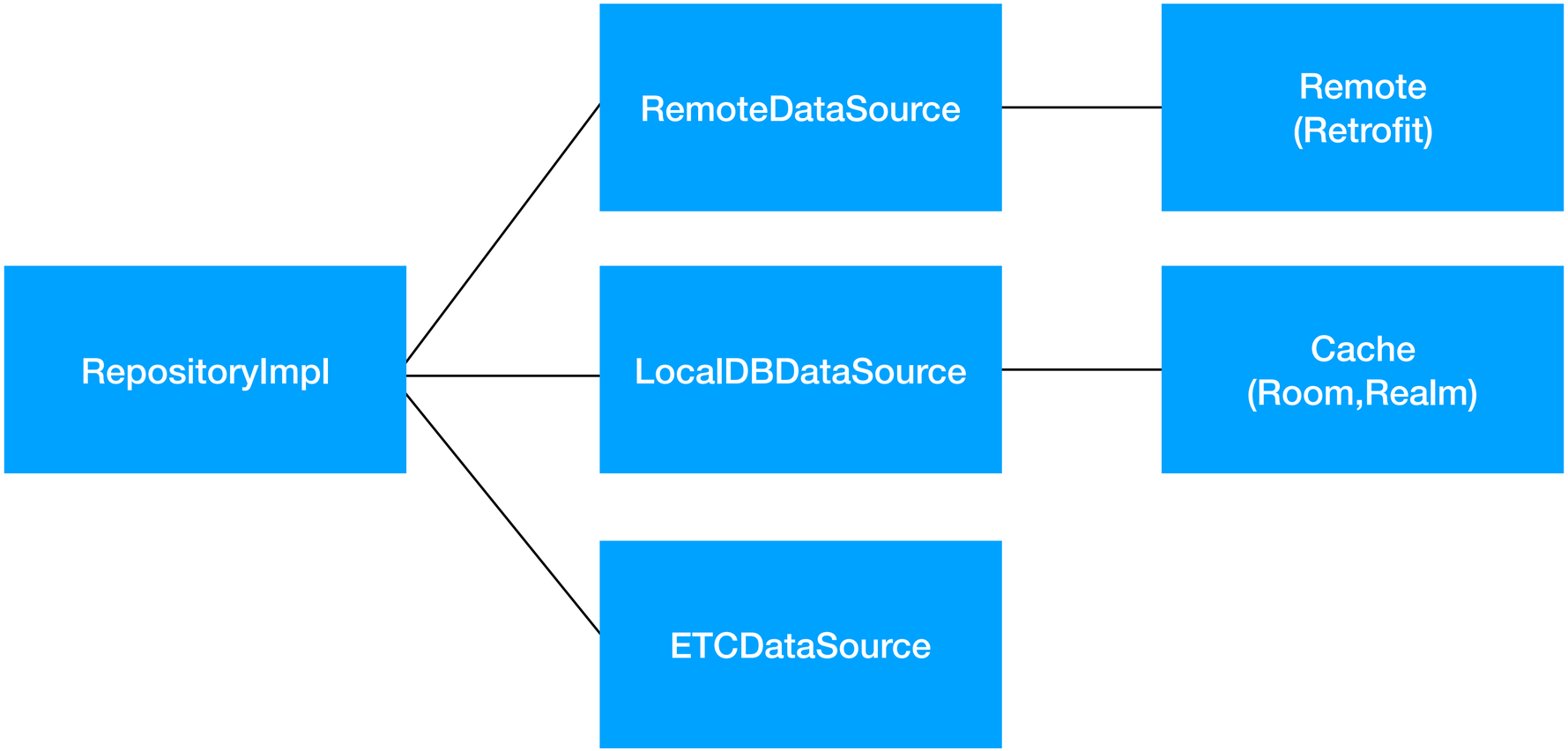
```
fun EventEntity.transform() = Event(
    eventId = eventId,
    name = name.orEmpty(),
    startDate = startDate ?: "1900-01-01",
    location = location?.name.orEmpty(),
    coverImageUrl = metadata?.coverImage.orEmpty(),
    content = metadata?.contents.orEmpty(),
    hostName = hostOrganization?.name.orEmpty()

)
```

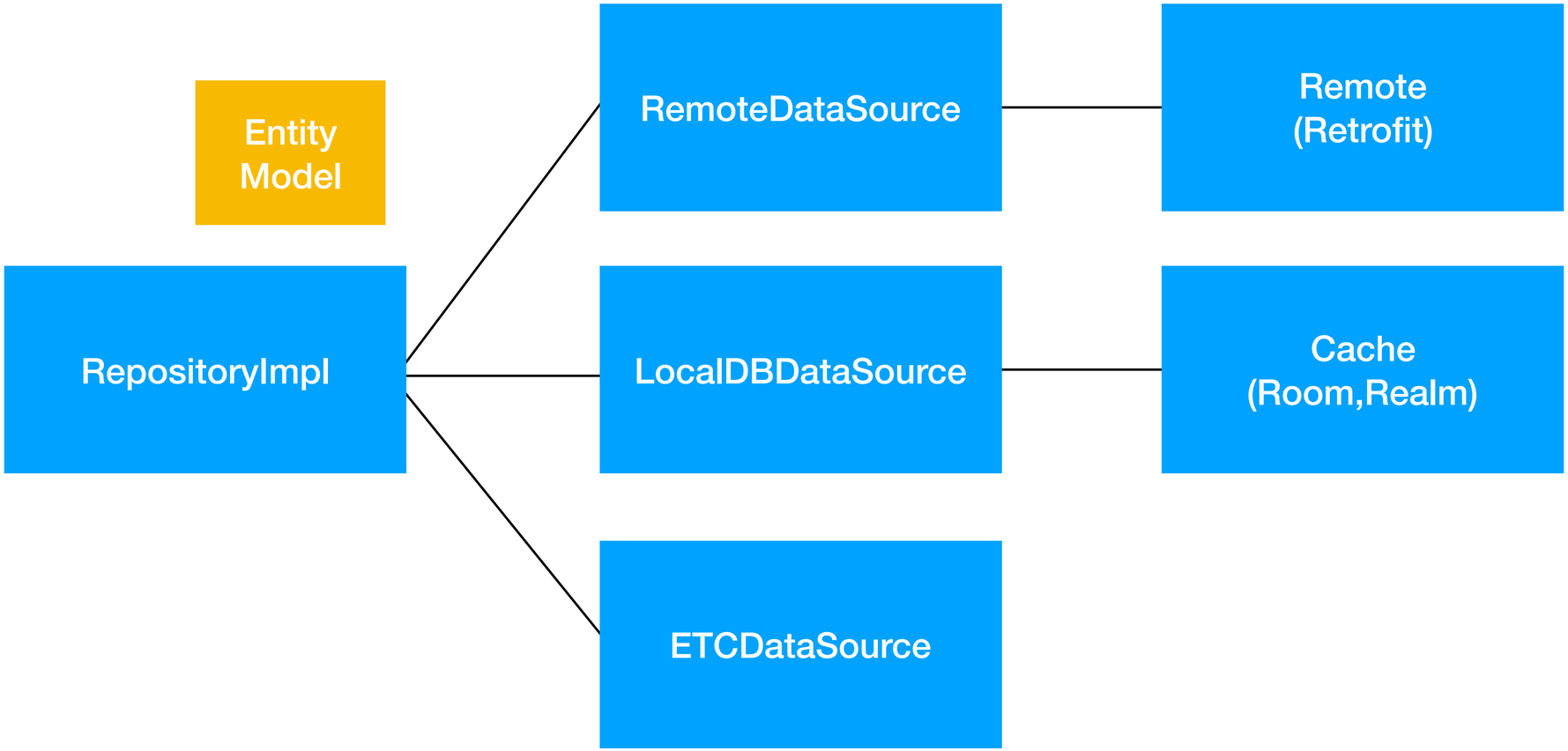
```
@Singleton
class EventListRepositoryImpl @Inject constructor(private val listRemote: EventListRemoteDataSource) :
    EventListRepository {

    override fun getRemoteEvents(): Flowable<List<Event>> {
        return listRemote.getEvents()
            .concatMapIterable { it }
            .map { it.transform() }
            .toList().toFlowable()
    }
}
```

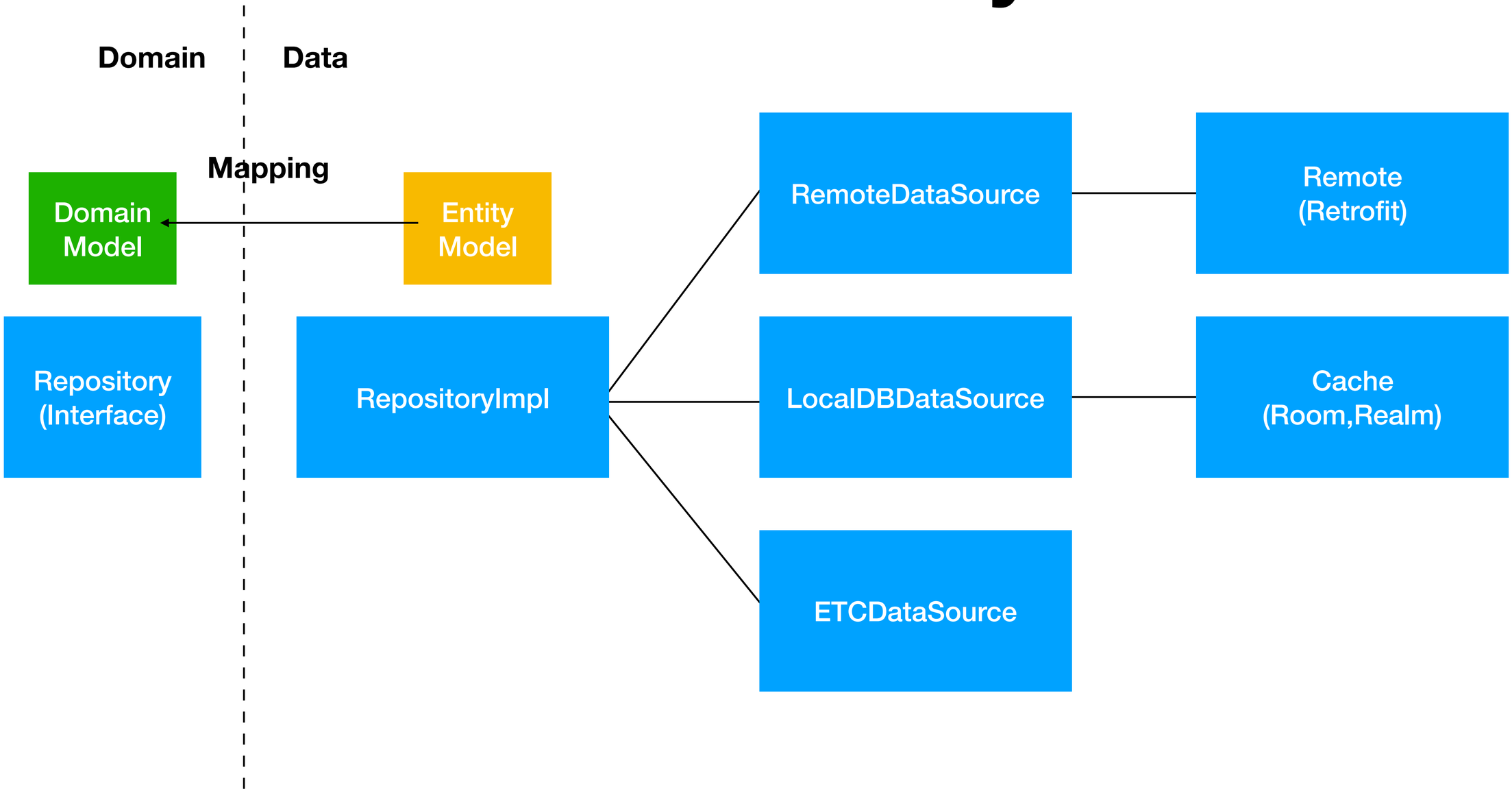
Data Layer


RepositoryImpl
RemoteDataSource
LocalDBDataSource
ETCDataSource
Remote
(Retrofit)
Cache
(Room,Realm)

Data Layer


Entity
Model
RepositoryImpl
RemoteDataSource
LocalDBDataSource
ETCDataSource
Remote
(Retrofit)
Cache
(Room,Realm)

Data Layer


Domain
Data
Mapping
Domain Model
Entity Model
Repository (Interface)
RepositoryImpl
RemoteDataSource
LocalDBDataSource
ETCDataSource
Remote (Retrofit)
Cache (Room,Realm)

Domain Layer

```
interface EventListRepository {
    fun getRemoteEvents(): Flowable<List<Event>>
}
```

```
data class Event(
    val eventId: Int,
    val name: String,
    val startDate: String,
    val location: String,
    val coverImageUrl: String,
    val content: String,
    val hostName: String
)
```

```
class GetEventListUseCase @Inject constructor(private val eventListRepository: EventListRepository) :
    UseCase<EventList, Void>() {

    override fun createFlowable(data: Void?): Flowable<EventList> = eventListRepository
        .getRemoteEvents().map {
            EventList(newEvents = it)
        }
}
```

Domain Layer