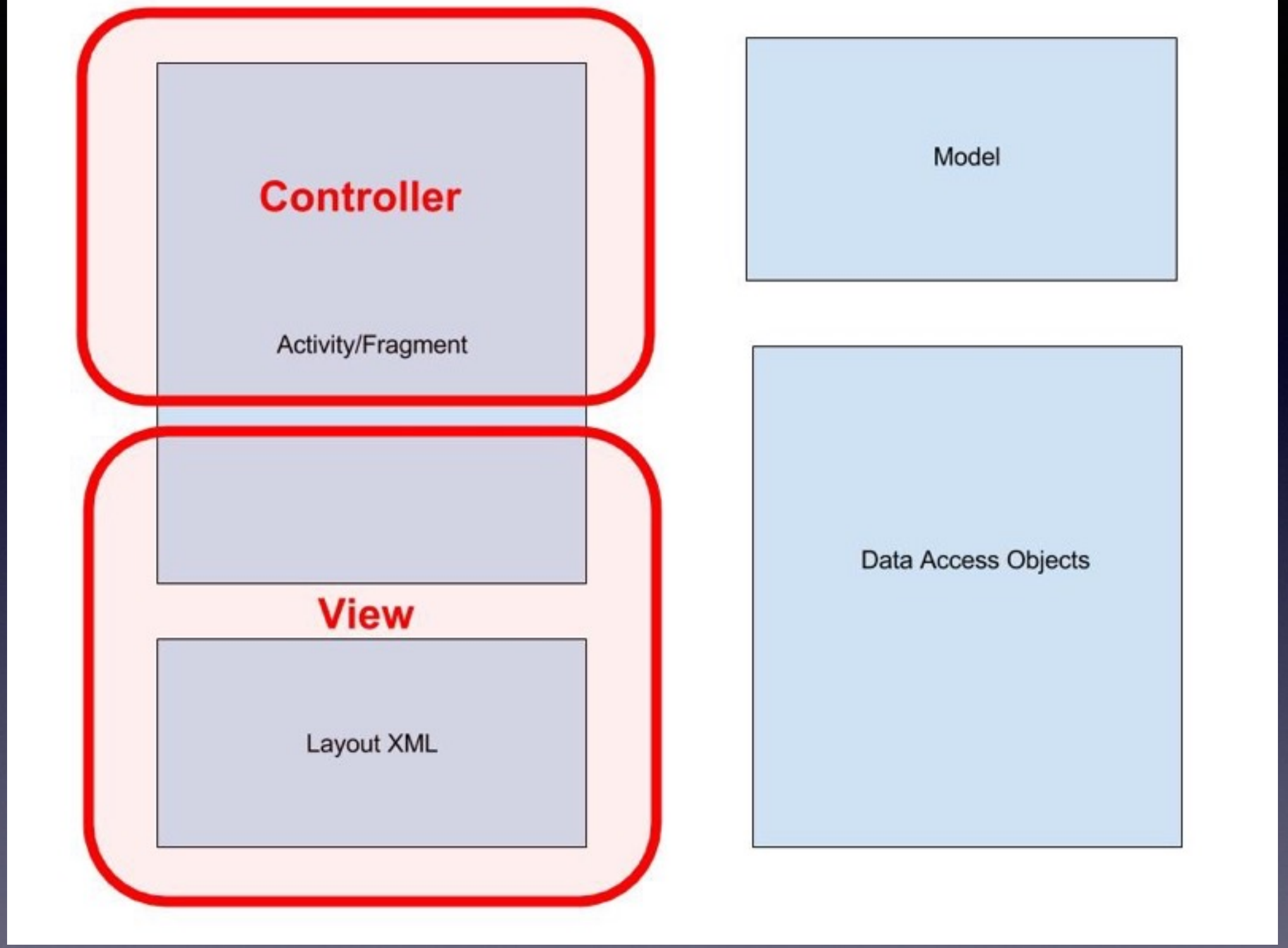
A Better Android Architecture

https://github.com/alphonzo79/AndroidArchitectureExample

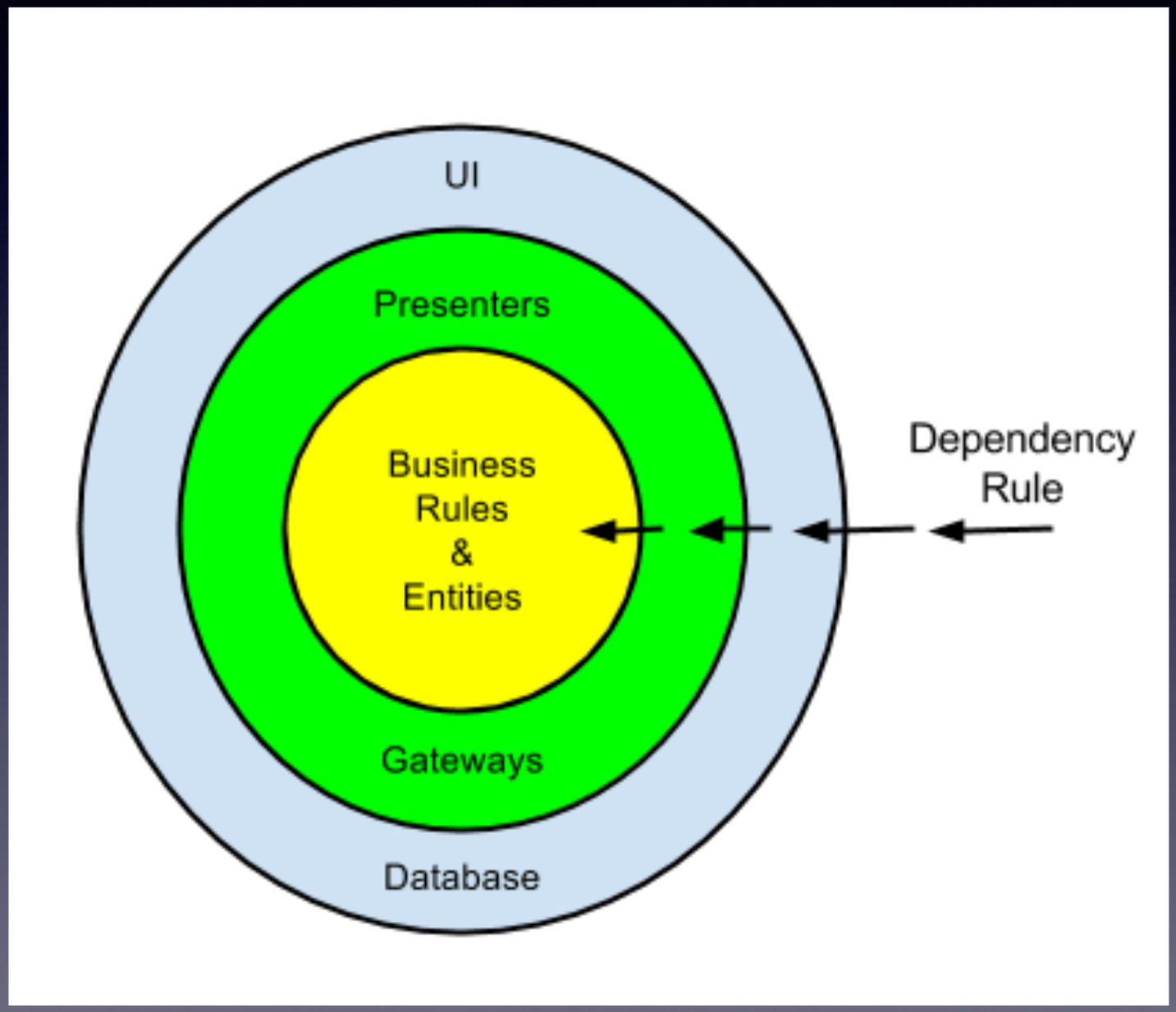


Traditional Android - MVC(ish)

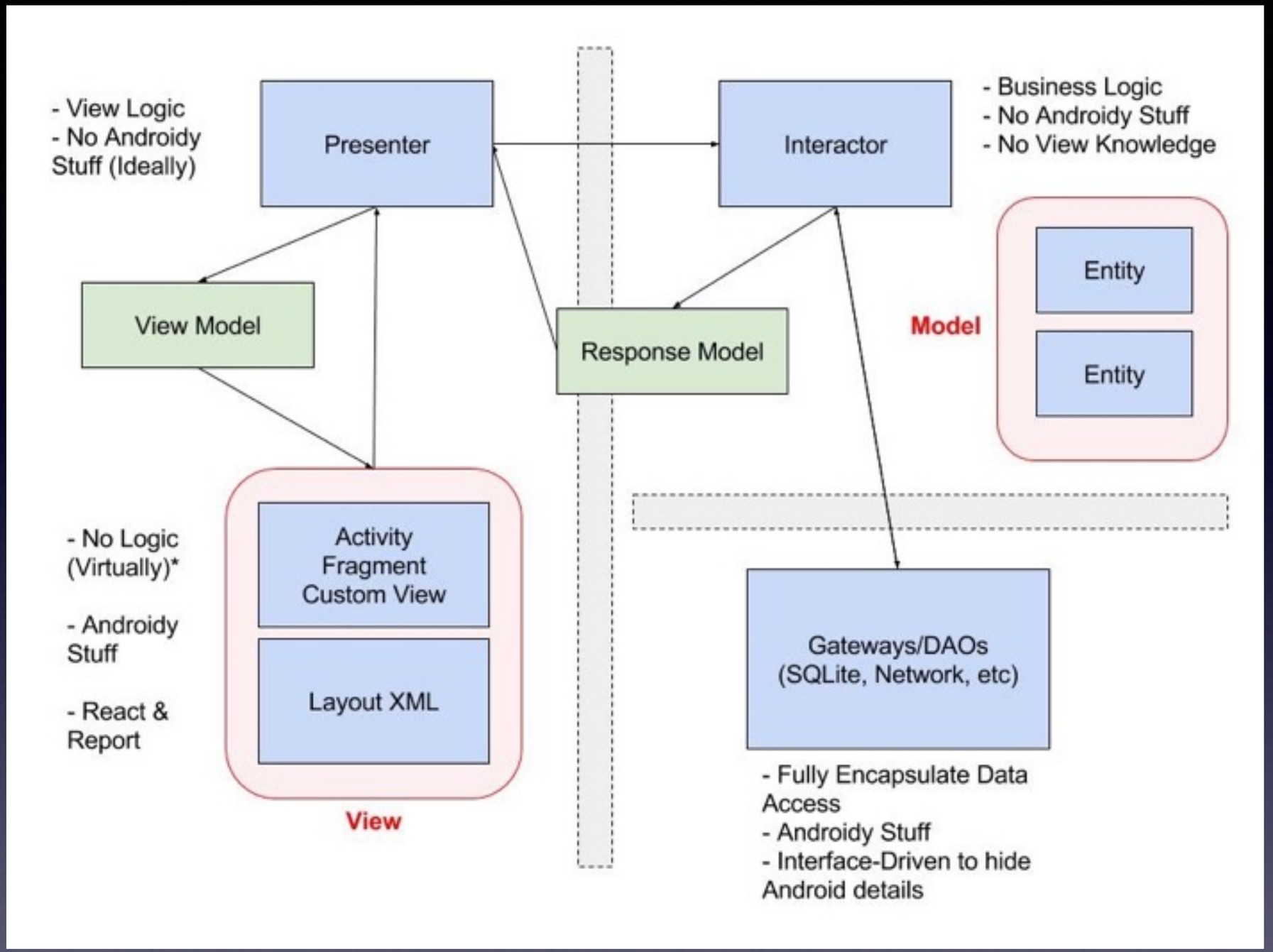
- Distinction between View and Controller is not clear
- Has no clear division between logic layers
- Lends itself to very large Activity/Fragment classes
- Pollutes the View/Controller with data access duties
- Changes to the view structure tend to become large projects
- Virtually all testing requires instrumentation or other “non-unit” support (Robolectric)

Architecture: Driving Forces

- Separate Responsibilities
- Hide Internal Details
- Encapsulate Change
- Manage Dependency Flow

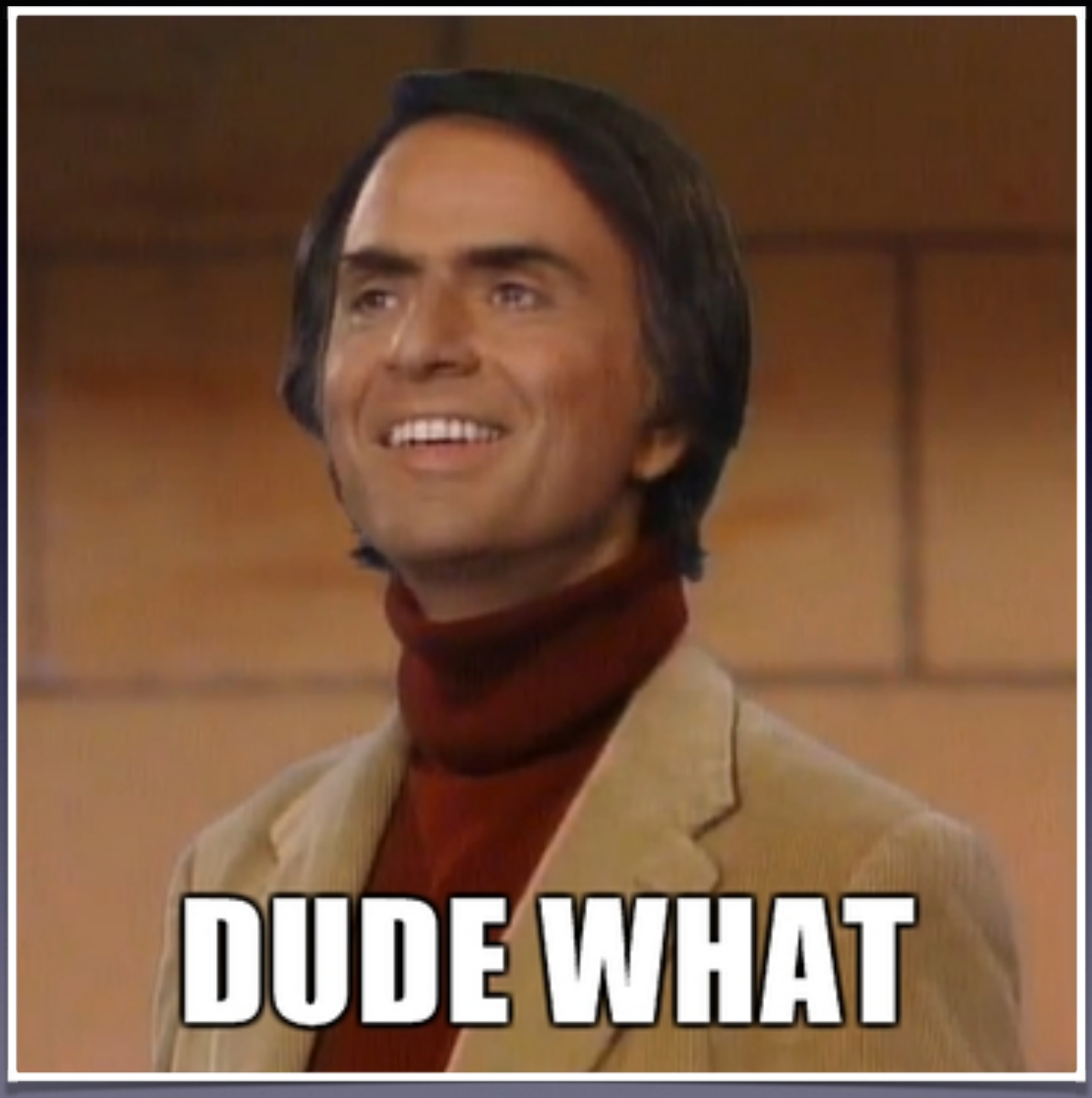


Adapted from https://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html



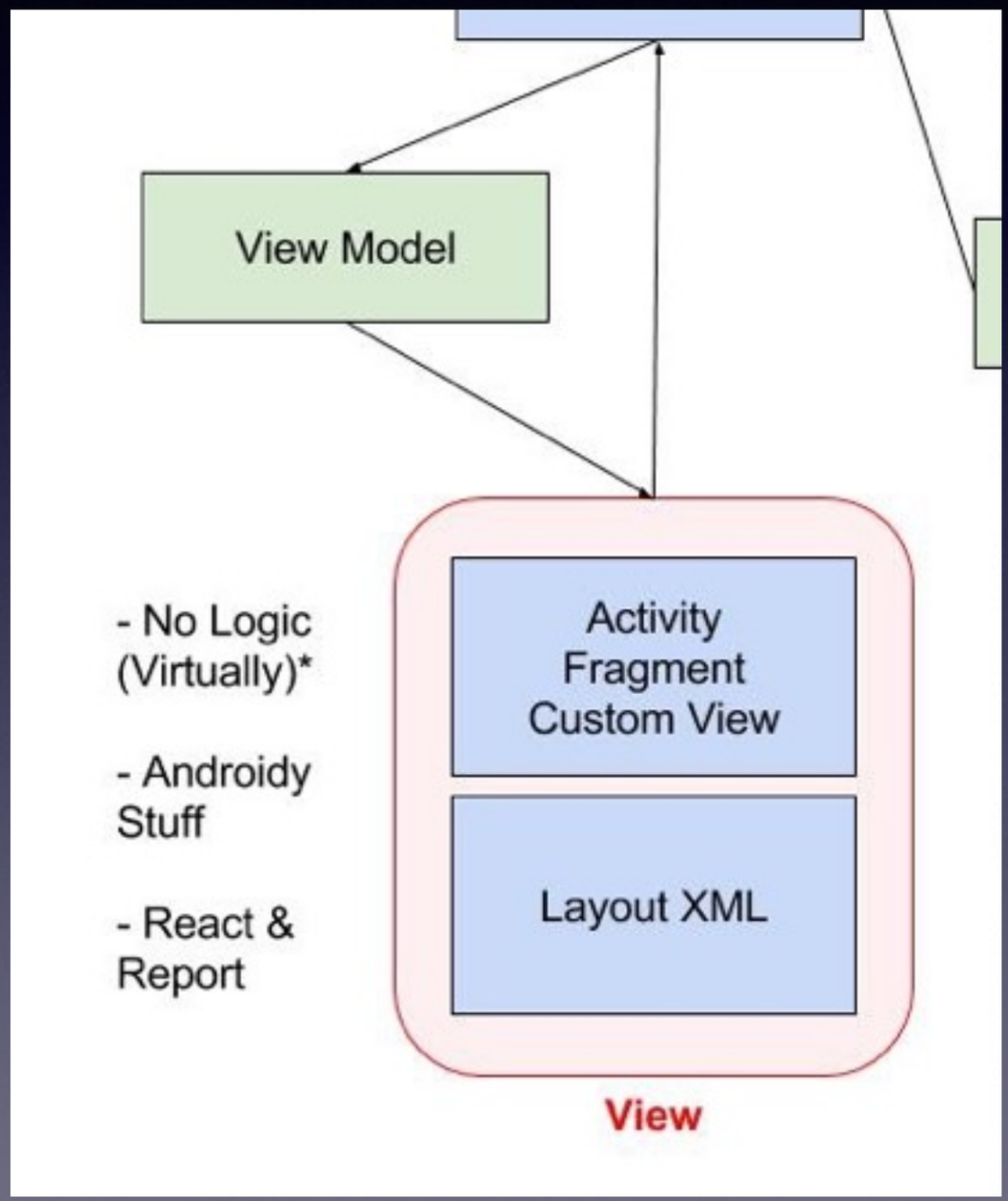
Model-View-Presenter

With the addition of Interactors


DUDE WHAT

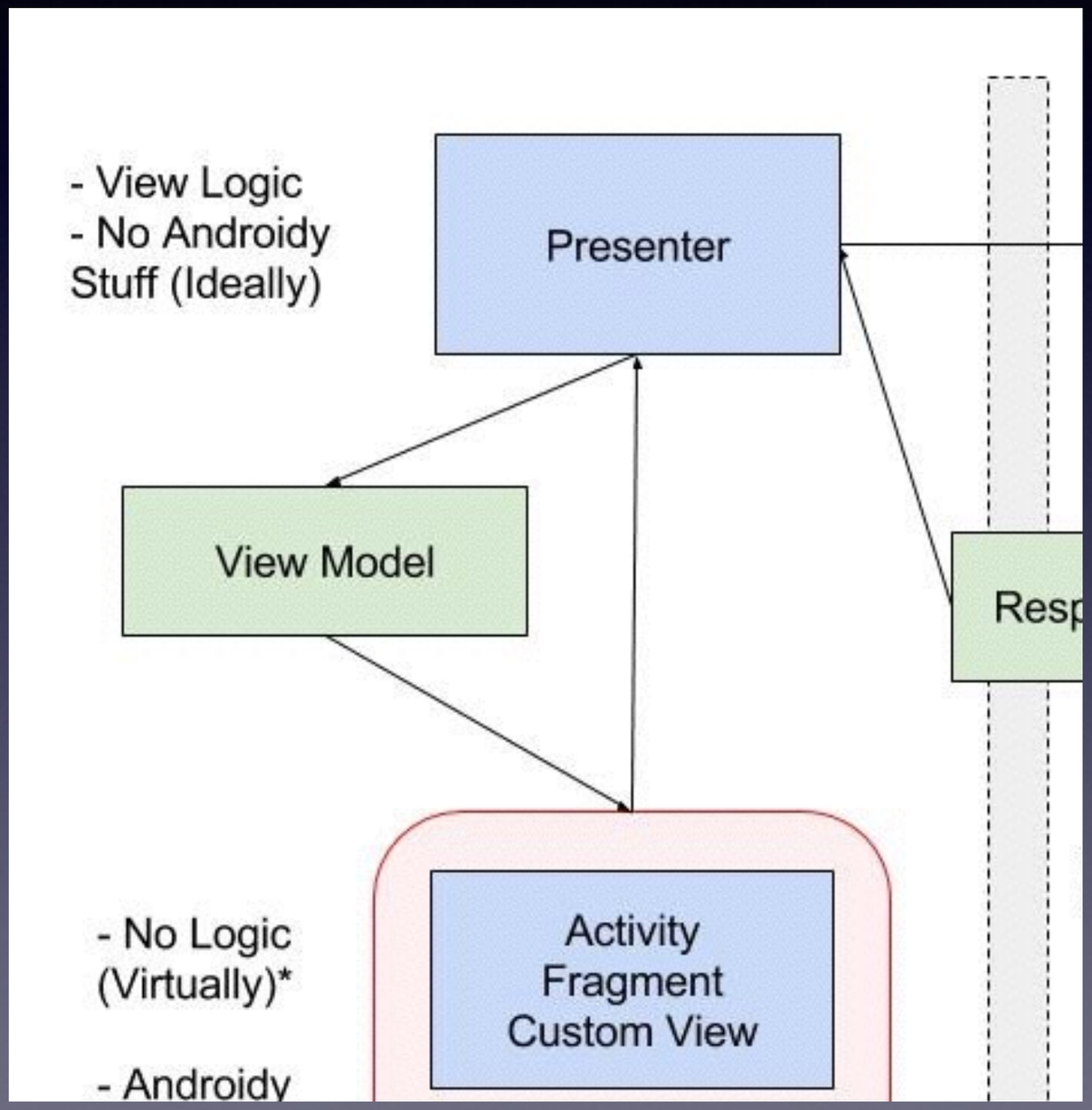
View Layer

- Strictly View duties - Dumb
- Inflate, Input, Output
- Makes as few decisions as possible
  - **Exceptions:**
    - Instantiates Presenter, Interactor, Gateways
    - Handles View components like adapters, etc
- Keep Android in this layer as much as possible
- Should be “swappable” (View Model Interface)



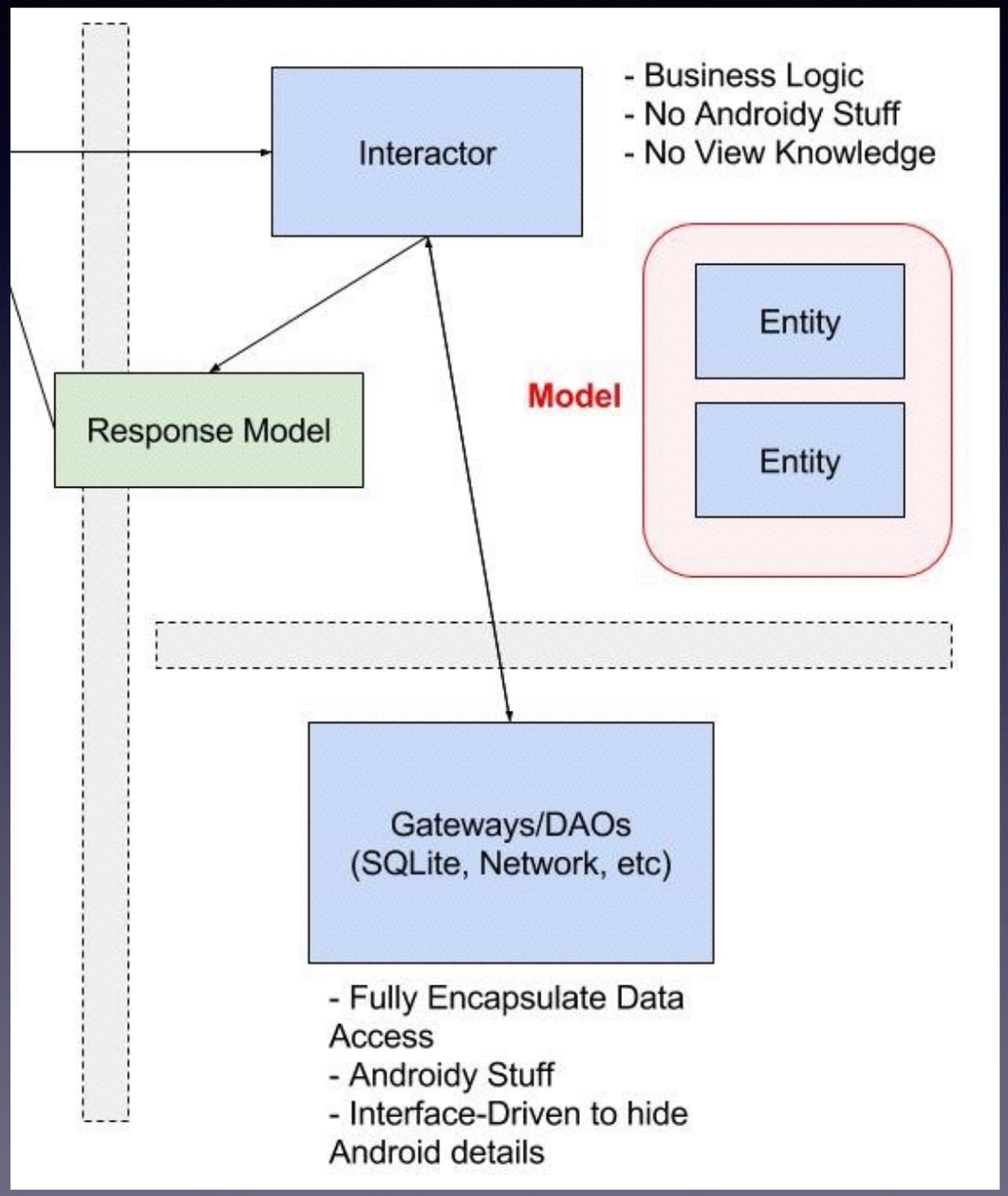
Presenter Layer

- View Logic
    - What to show; What to hide, How to handle user interaction; When to get data; Convert Server Model to View Model
- Communicates with the view through a View Model interface
- Ideally clean of anything “Android”
- Makes no business decisions, Handles no data fetching
- Implements Response Model interface to receive updates from Interactor



Interactor/Gateway Layer

- Business Logic: Source(s) of data, caching/refresh, etc
- Provides models/data back to the Presenter through Response Model interface
- Holds Gateways (DAOs), but ideally is unaware of their internals. Only requests data from them
- Ideally knows nothing of Android
  - (Gateways must, since they handle network and file IO or SQLite access)



Pros / Cons

- Better separation of duties & logic
- Smaller classes
- Clear dependency flow
- More portable - swap views, presenters or interactions for various scenarios

- Class Structure Explosion
- Interfaces implemented exactly once?
- Pass-throughs show up sometimes

- View Logic
- No Androidy Stuff (Ideally)

Presenter

Interactor

- Business Logic
- No Androidy Stuff
- No View Knowledge

View Model

Response Model

Model

Entity

Entity

- No Logic (Virtually)*

- Androidy Stuff

- React & Report

Activity
Fragment
Custom View

Layout XML

View

Gateways/DAOs
(SQLite, Network, etc)

- Fully Encapsulate Data Access
- Androidy Stuff
- Interface-Driven to hide Android details

Testability

- Business logic is (should be) completely independent of instrumentation
- View logic is (should be) completely independent of instrumentation
- With a mocking library you can now do real unit tests on big parts of the app that used to be wrapped up in instrumented components