Architecture: Bento vs Burrito

Keishin Yokomaku @ Drivemode, Inc.
umeda.apk #1

@KeithYokoma

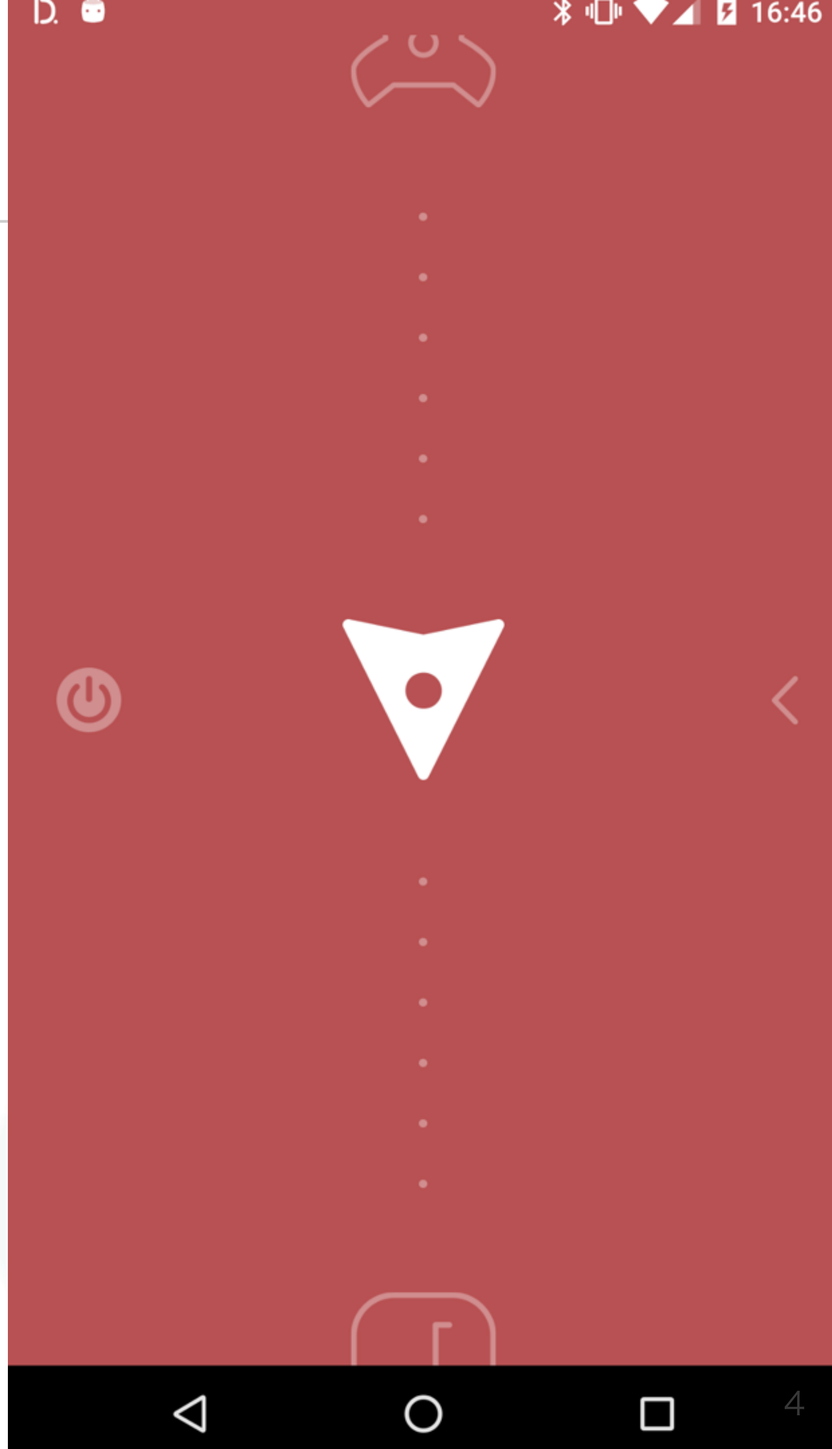
- Keishin Yokomaku at Drivemode, Inc.
- Work
  - Android apps
  - Android Training and its publication
- Like
  - Bicycle, Photography, Tumblr and Motorsport
- hoge

About “Drivemode”

- “No Look” driving interface running on the phone.
  - Simplify the way drivers user their phone while driving.
  - Allows them to focus on the road.

About “Drivemode”

- Available features
  - Contacts: Call and Message
  - Navigation control
  - Music playback control and playlist access
  - Application launcher



About “Drivemode”

- Overlays on any applications
  - You can see navigation while playback controlling
- Help users understanding what they are doing
  - Sound & voice feedback
  - Big UI and touch feedback effect

Available on Play Store!

http://bit.ly/1LYdxAg

Application Requirements

- Login and save user profile
- Choose favorite contacts
- Choose favorite navigation app
- Control music playback

…

Application Requirements

- **Login and save user profile**
- **Choose favorite contacts**
- **Choose favorite navigation app**
- **Control music playback**

Internet Access

Database

Preferences

IPC

Application Requirements

Internet Access	Protocol, network state, cache strategy, etc…
Database	Content Uri, projection, selection, order, etc…
Preferences	Data type, preference name, etc…
IPC	AIDL, media session and controller, etc…

CUSTOMER RESPONSIBILITY

1. DAILY LUBRICATION
2. EQUIPMENT DAMAGE
3. ALL FLUID LEVELS DAILY
4. BATTERY LEVEL
5. REFUELING TANK
6. EXCESSIVE CLEAN UP
7. INSURANCE
8. FLAT TIRES
9. TIRE DAMAGE

“Who is responsible for each features/requirements?”

Application Components

- Activity
- Fragment
- View

- BroadcastReceiver
- Service
- ContentProvider

Activity responsibility

Activity

MainActivity

- UI Event handler
- System service calls
- AsyncTaskLoader
- UI State management

…

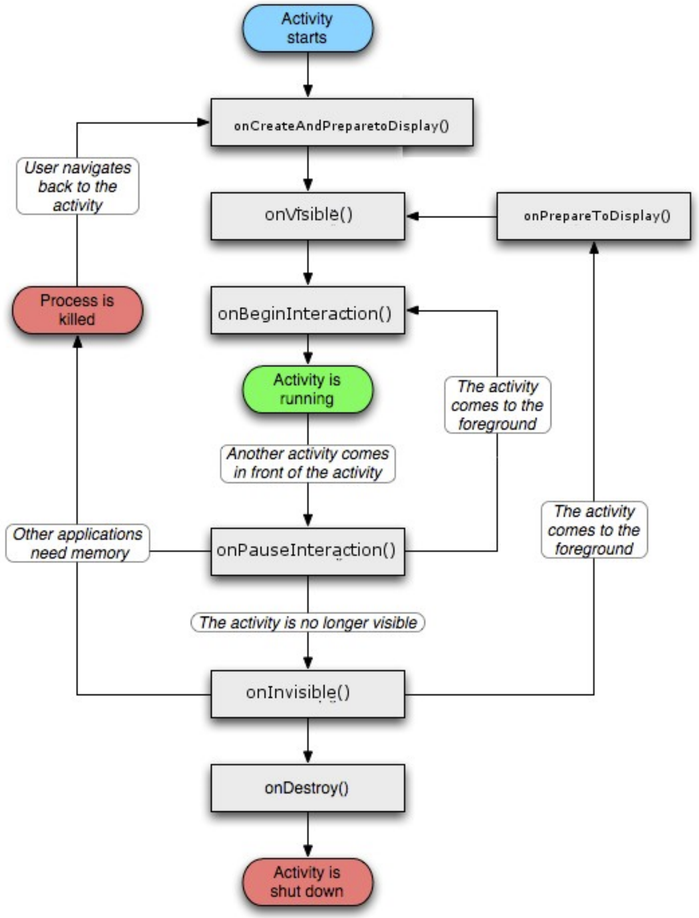
Activity responsibility

Activity

MainActivity

- UI Event handler
- System service calls
- AsyncTaskLoader
- UI State management

...



Burrito architecture

- Everything is mixed into one activity.
- You cannot get rid of cilantro even if you hate it.

Better Architecture

Practices

MVC

MVP

MVVM

...

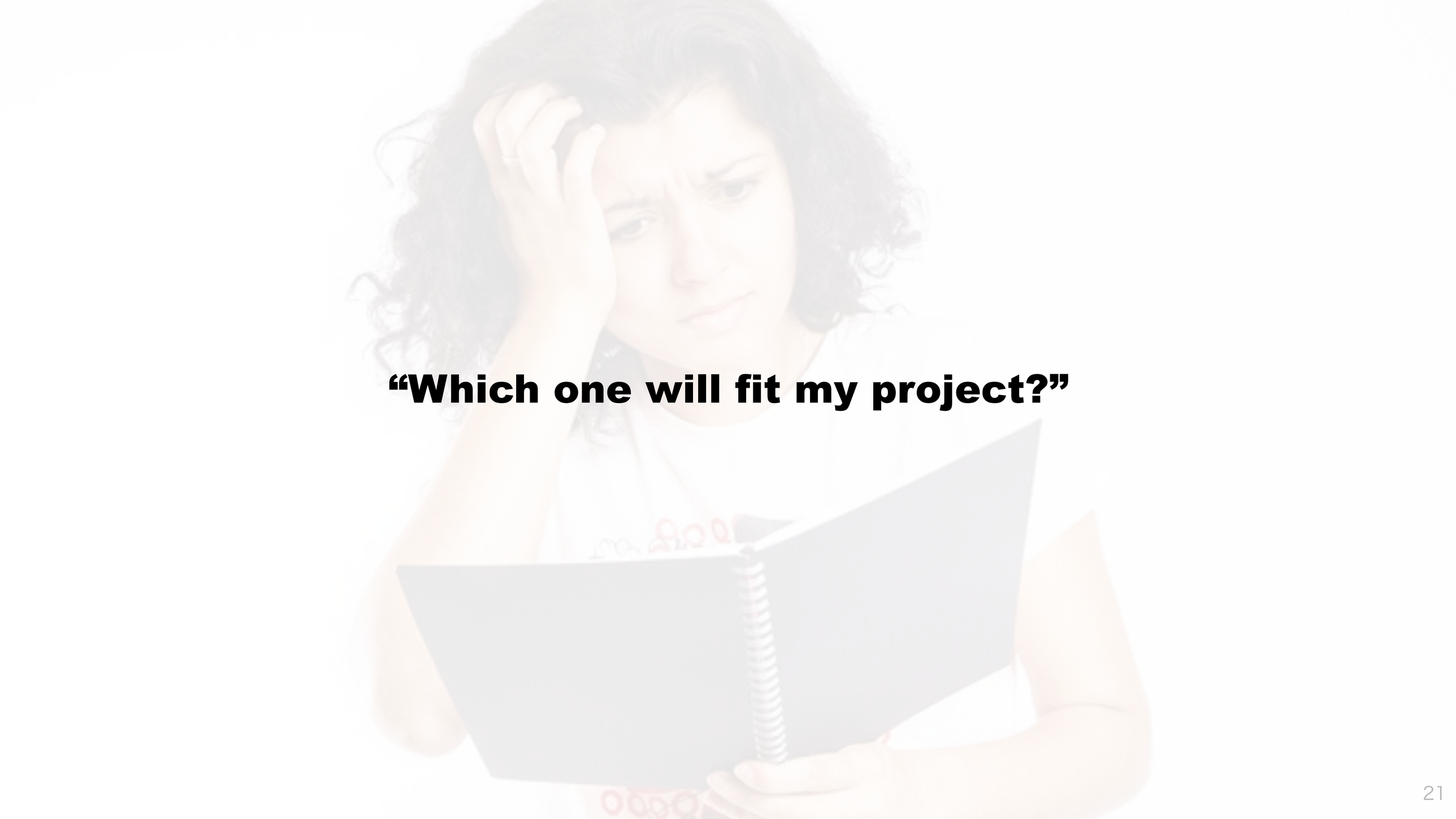
Practices

ModelViewController

ModelViewPresenter

ModelViewViewModel

...



“Which one will fit my project?”

Let’s simplify it

Keywords

- **Single Responsibility Principle**
  - “A class should have only one reason to change.”
- **Clean Architecture**
  - “Independent of Frameworks”, “Testable”, “Independent of UI”, “Independent of Database”, “Independend of any external agency”.
- For this time, let’s talk about **“Model”** layer

–@rallat

Presentation

Bento

Presentation

Model

The so-called “Model”

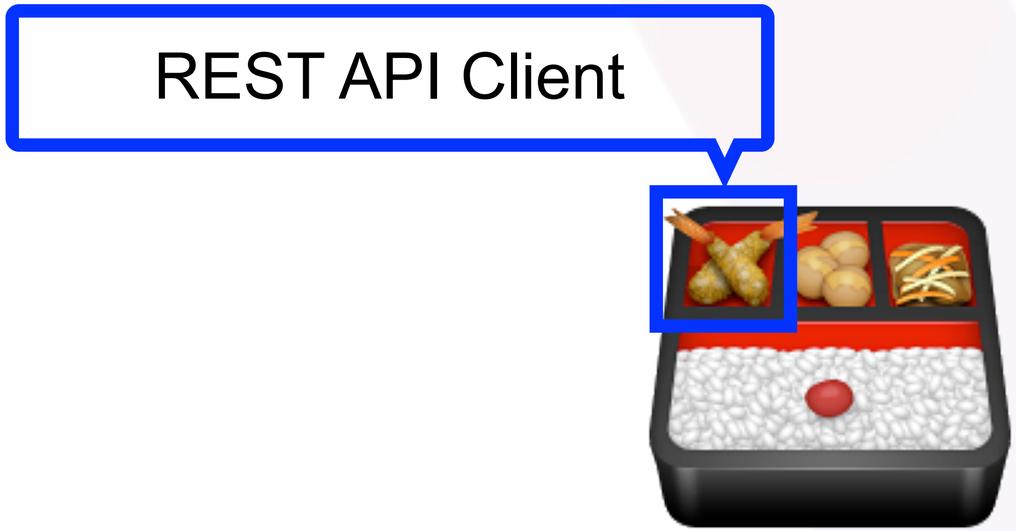
- Model has “Business Logic” and “Data”.
- One model contains various kind of “Business Logic” and “Data”.
  - How to retrieve and store data(REST API, Database Cache, etc…)
  - How to convert serialized data to value object(entity)
  - How to order collection data
  - etc…
- Model may have blocking procedure(Network I/O, File I/O, etc…).

“Model can be Burrito”

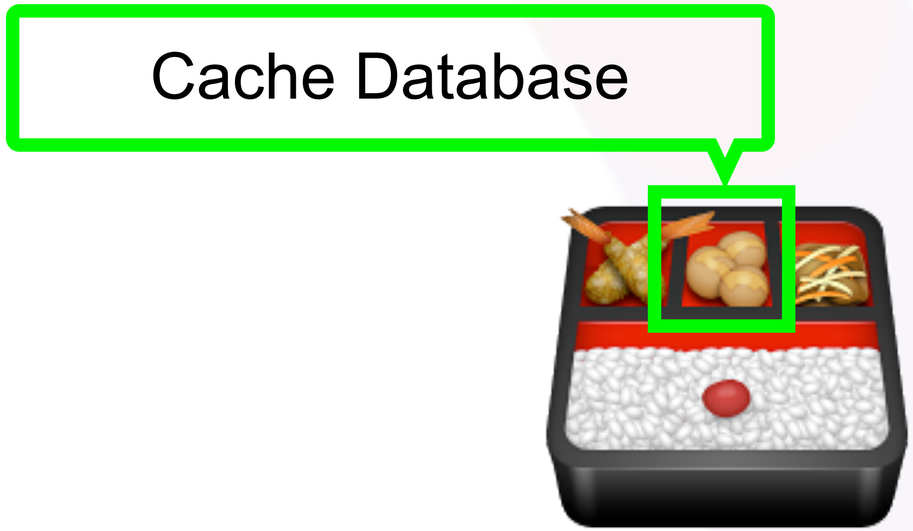
–Someone

Model as a Bento

Model as a Bento

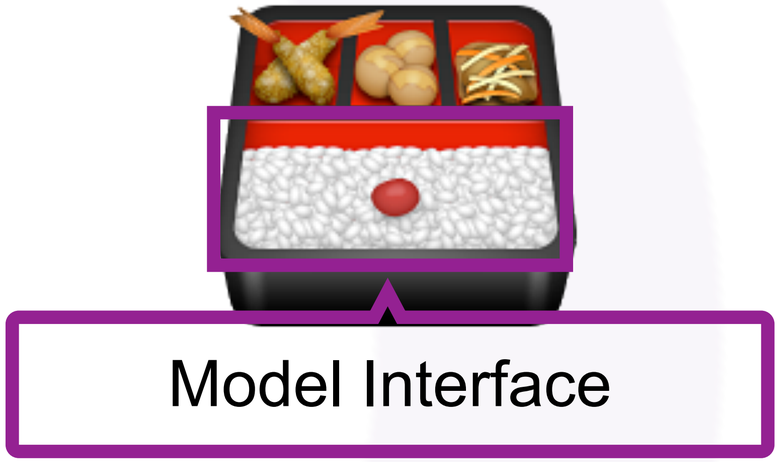


Model as a Bento



Model as a Bento

Model as a Bento



Model as a Bento

Worker Thread

Main Thread

Model as a Bento

Model as a Bento

- Build classes for each purpose(SRP)
  - REST Client class
  - Cache Database manager class
  - Preferences class
- Combine all asynchronous tasks in Model Interface class using Rx/Promise

Model as a Bento

```
// REST Client interface
public interface FooBarRestGateway {
  @GET(“foo/bar”)
  Observable<List<FooBarData>> requestFooBar();
}

// Database Manager interface
public interface FooBarDatabaseClient {
  Observable<List<FooBarData>> fetchFooBar();
}
```

Model as a Bento

```
// REST Client interface
public interface FooBarRestGateway {
  @GET("foo/bar")
  Observable<List<FooBarData>> requestFooBar();
}

// Database Manager interface
public interface FooBarDatabaseClient {
  Observable<List<FooBarData>> fetchFooBar();
}
```

Model as a Bento

```
// Model Interface class
public class FooBarRepository {
  private final FooBarRestGateway mGateway;
  private final FooBarDatabaseClient mDbClient;

  public Observable<List<FooBarData>> fooBarList() {
    return Observable.<List<FooBarData>>concat(
        mDbClient.fetchFooBar(),
        mGateway.requestFooBar()
      ).first(list -> list != null)
      .subscribeOn(Schedulers.io());
  }
}
```

https://speakerdeck.com/hkurokawa/5-rxjava-tips-you-might-not-know

Model as a Bento

```
// Model Interface class
public class FooBarRepository {
  private final FooBarRestGateway mGateway;
  private final FooBarDatabaseClient mDbClient;

  public Observable<List<FooBarData>> fooBarList() {
    return Observable.<List<FooBarData>>concat(
        mDbClient.fetchFooBar(),
        mGateway.requestFooBar()
      ).first(list -> list != null)
      .subscribeOn(Schedulers.io());
  }
}
```

https://speakerdeck.com/hkurokawa/5-rxjava-tips-you-might-not-know

Model as a Bento

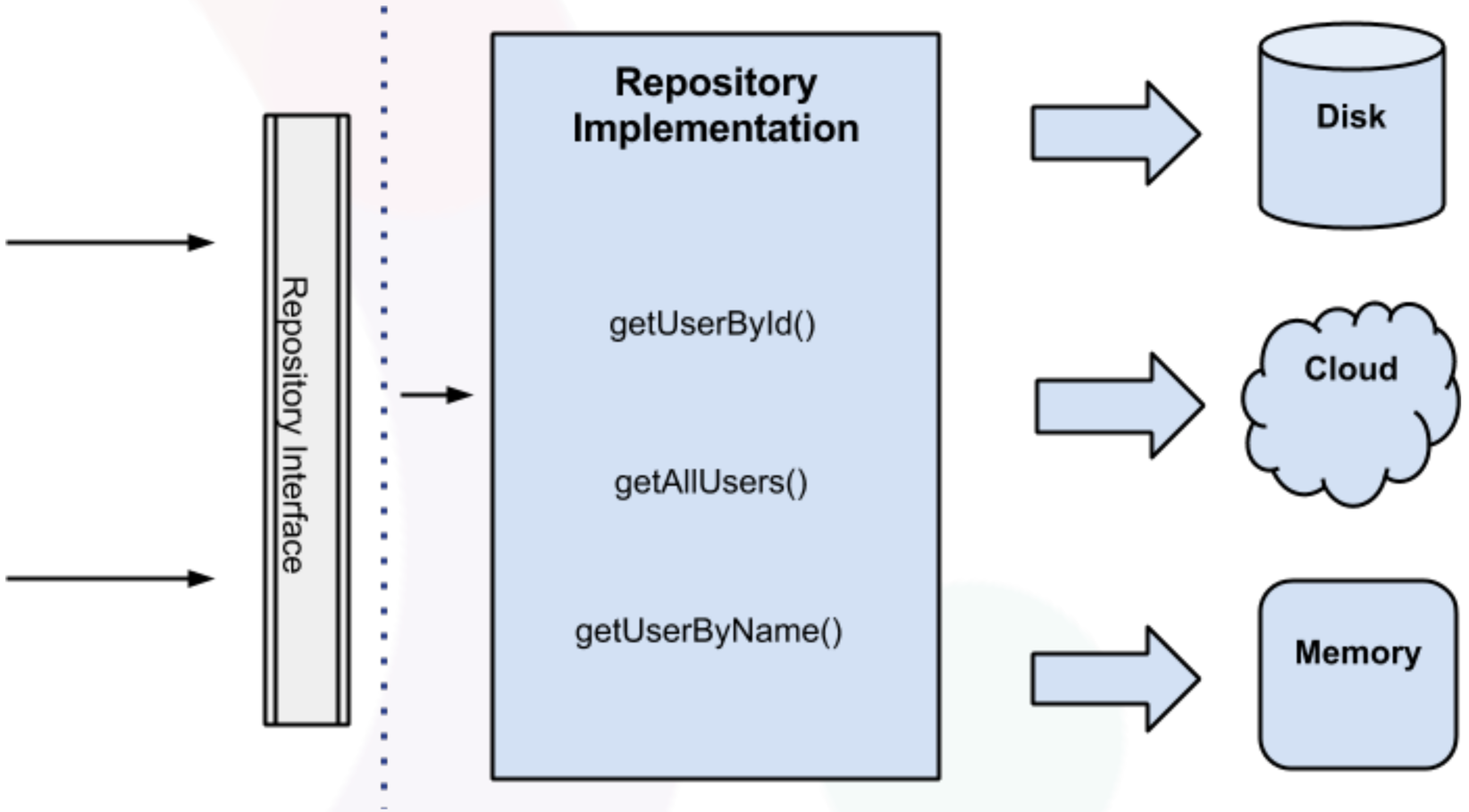
```
// Model Interface class
public class FooBarRepository {
  private final FooBarRestGateway mGateway;
  private final FooBarDatabaseClient mDbClient;

  public Observable<List<FooBarData>> fooBarList() {
    return Observable.<List<FooBarData>>concat(
        mDbClient.fetchFooBar(),
        mGateway.requestFooBar()
      ).first(list -> list != null)
      .subscribeOn(Schedulers.io());
  }
}
```

https://speakerdeck.com/hkurokawa/5-rxjava-tips-you-might-not-know

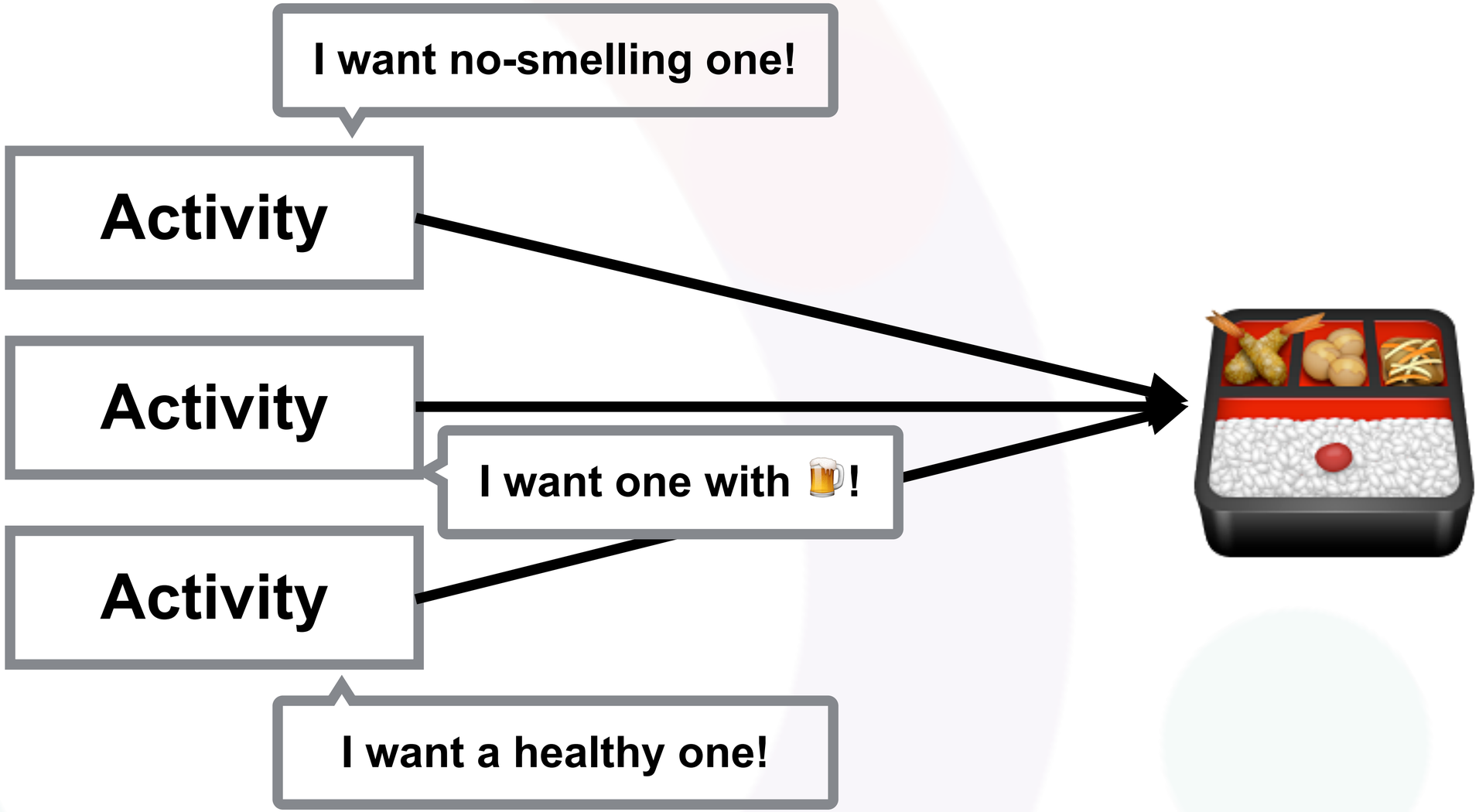
Bento and Clean Architecture

Bento



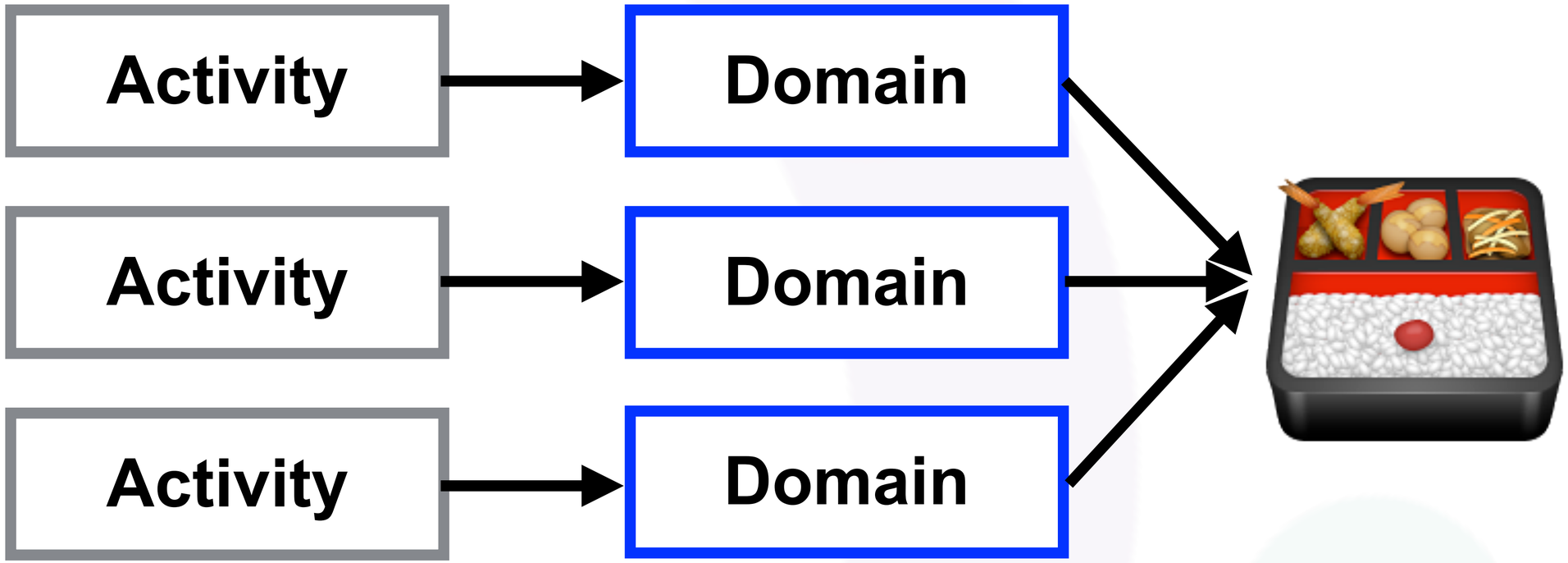
Clean Architecture: Data Layer using Repository Pattern

Don’t touch Bento directly

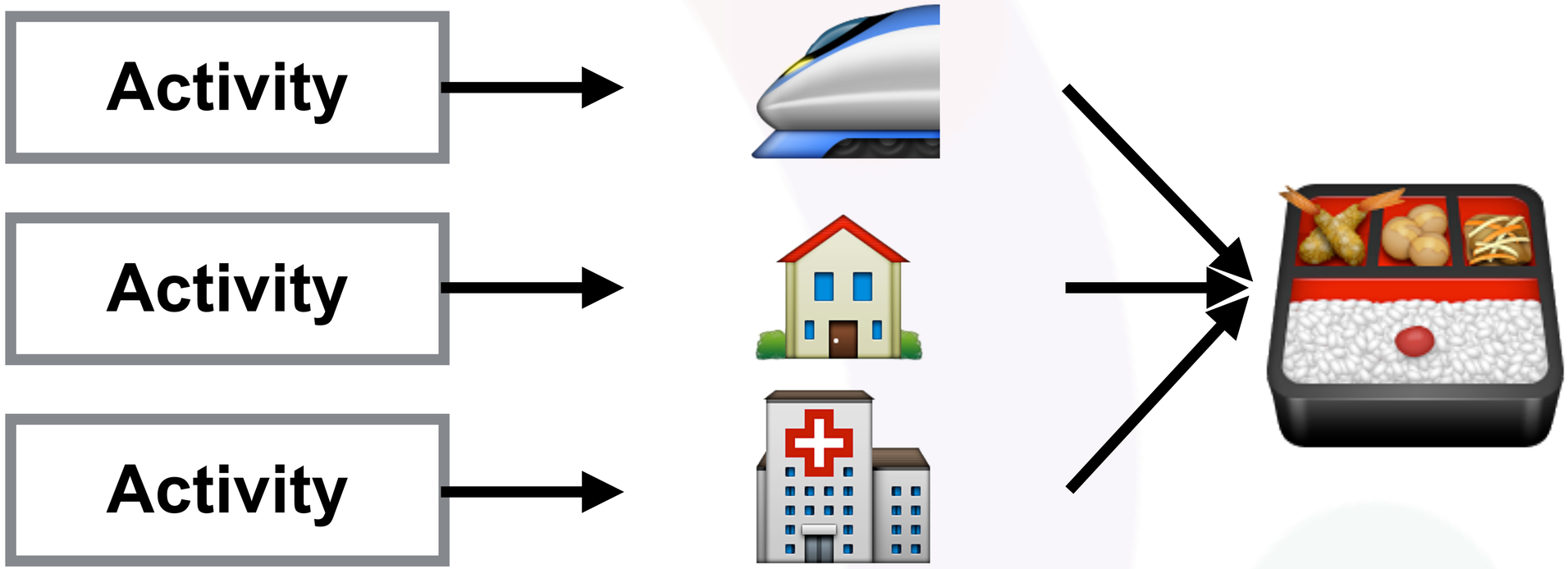


(Photo by SAKURAKO - Do not get mad !/ MJ/TR (´・ω・))

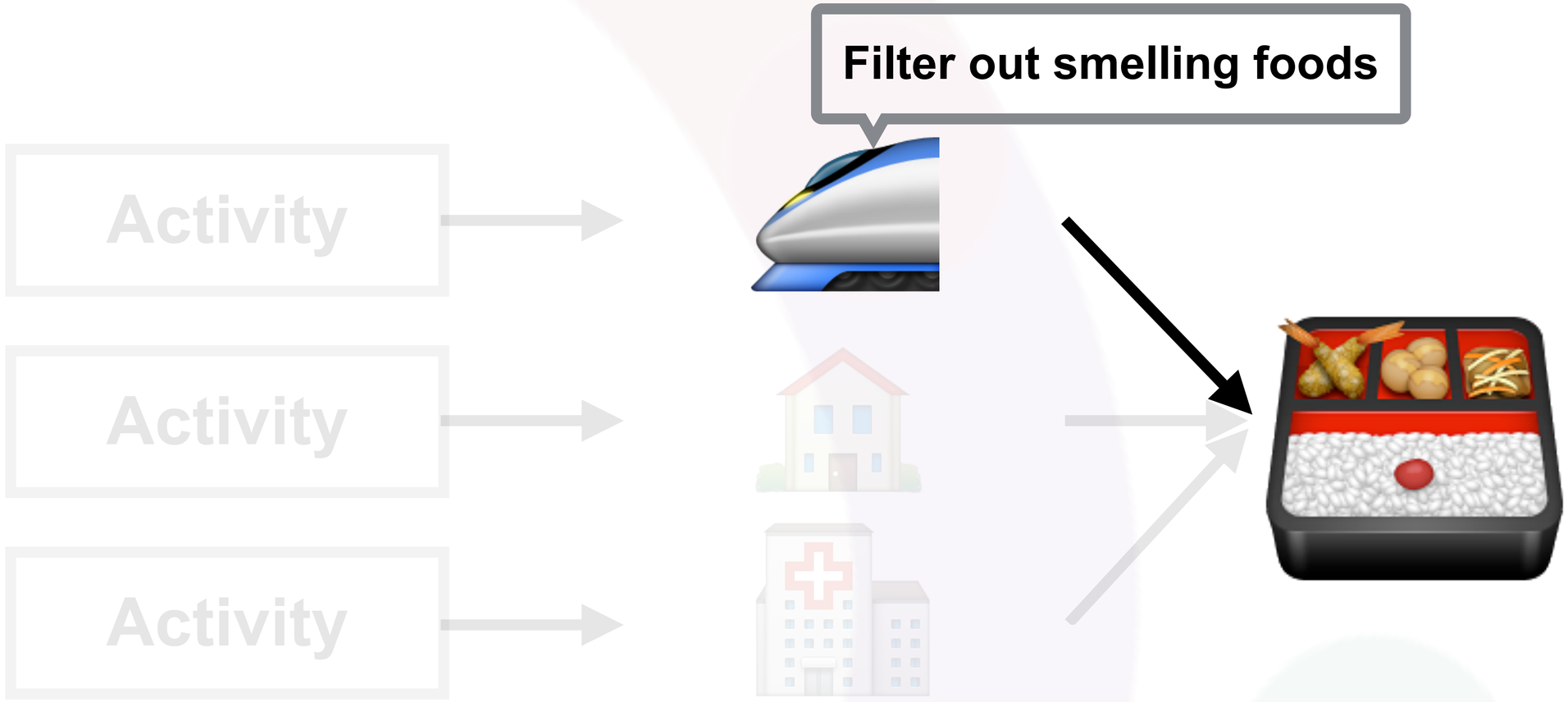
Domain layer



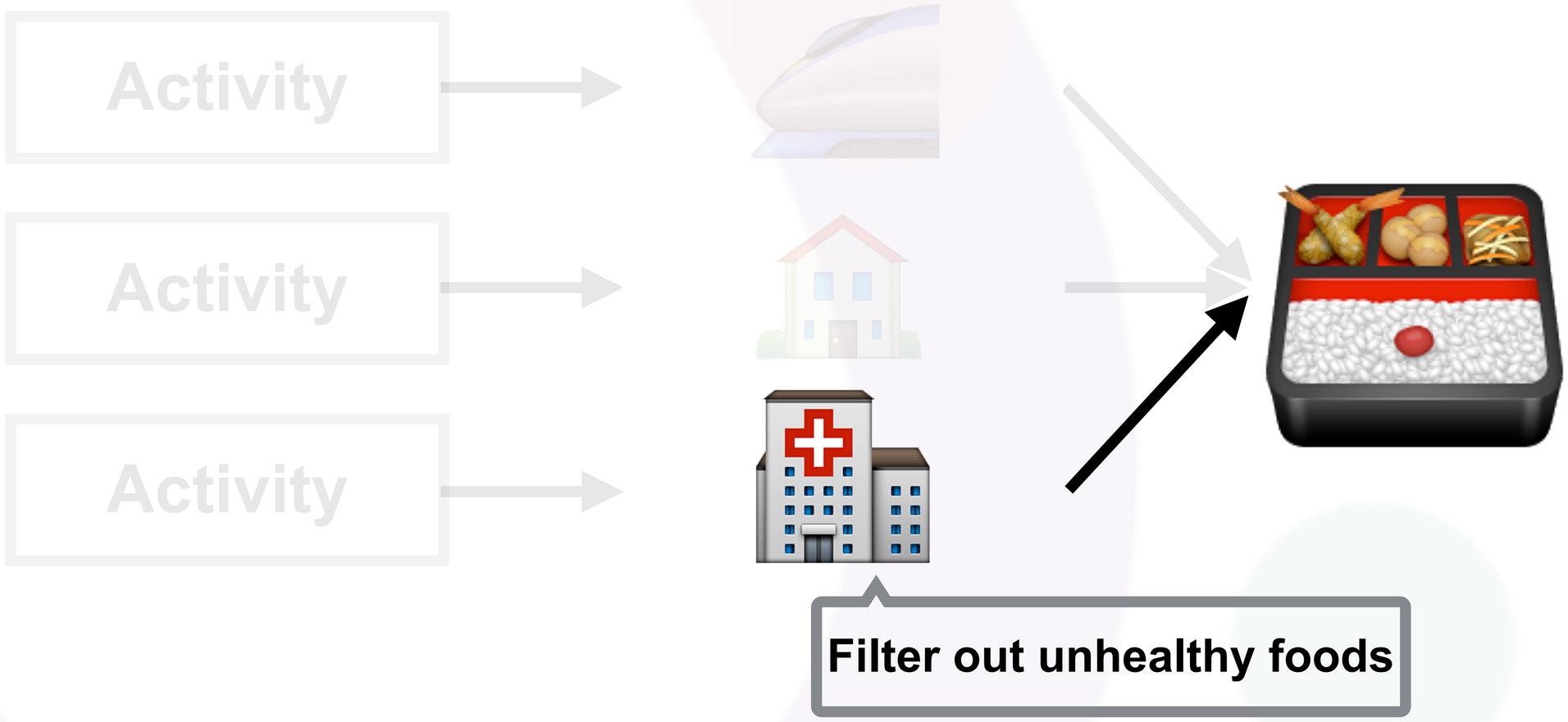
Domain layer



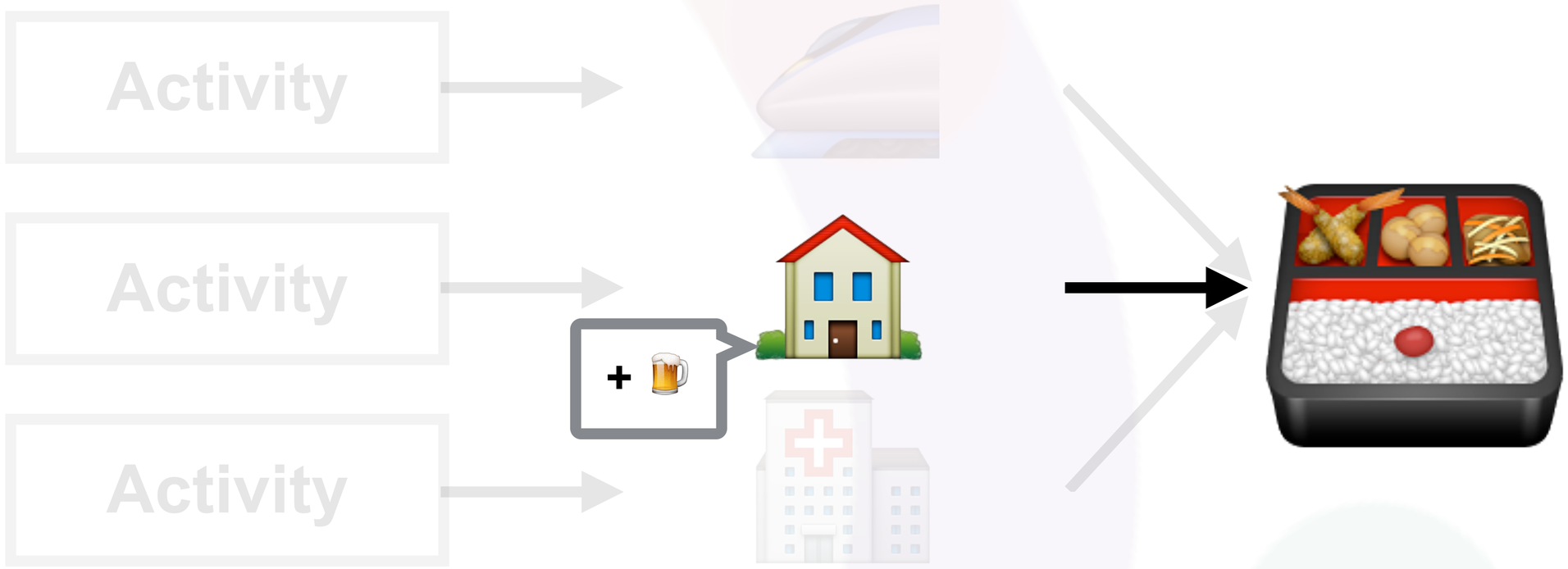
Domain layer



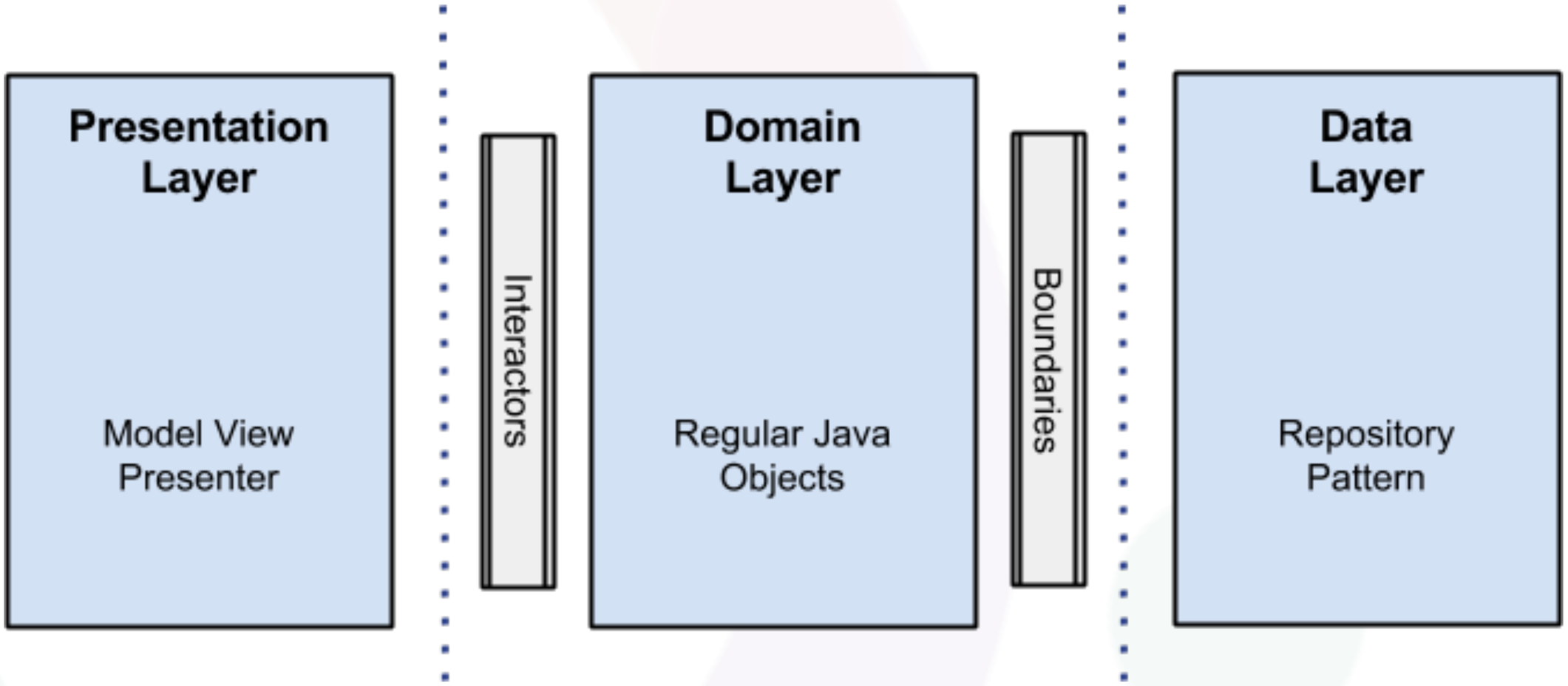
Domain layer



Domain layer

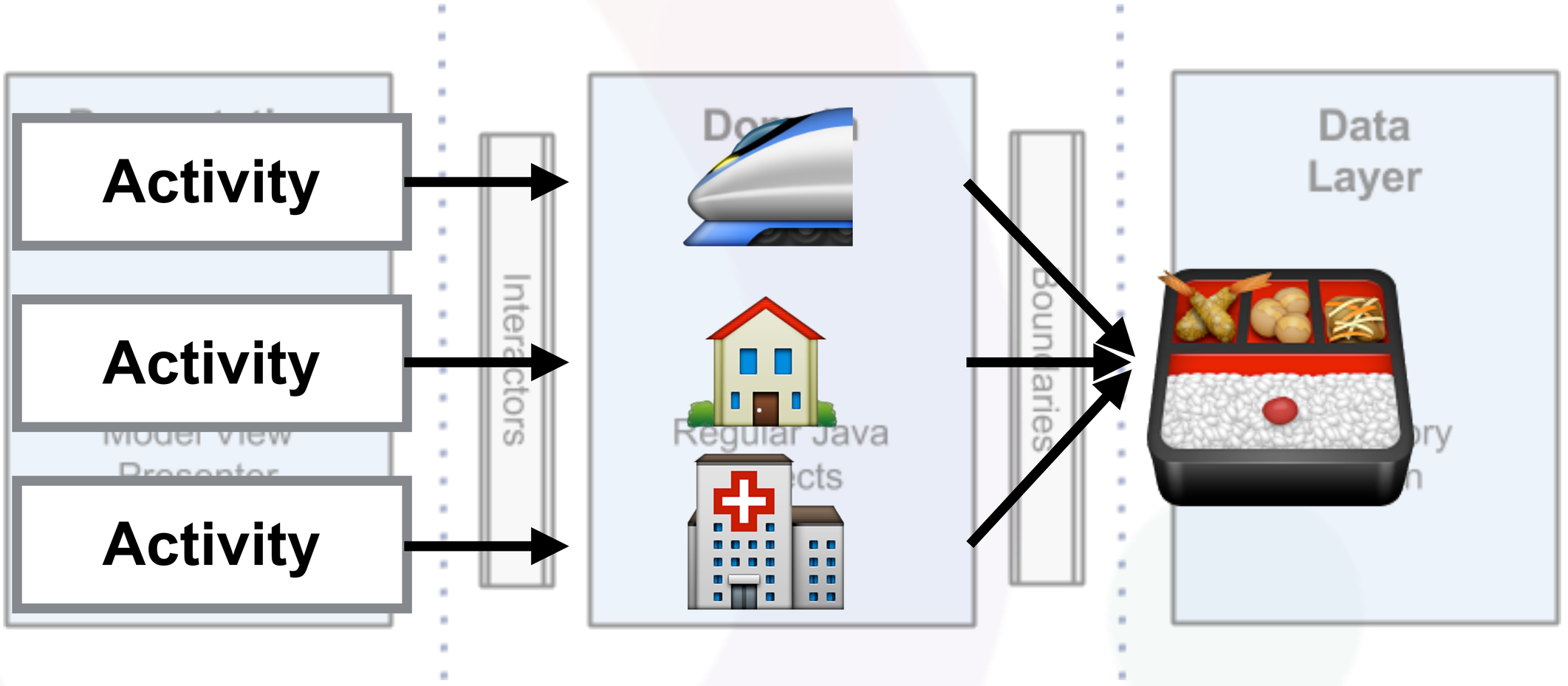


Domain layer



Clean Architecture: 3 layer architecture

Domain layer



Clean Architecture: 3 layer architecture

Wrap up

- Clean Architecture and Bento
  - Class separation according to Single Responsibility Principle
  - Combine several tasks using Rx/Promise
  - Data layer bento is Singleton
  - Modify data in Domain layer
- Don’t Burrito, do Bento

Architecture: Bento vs Burrito

Thank you for listening!

Keishin Yokomaku @ Drivemode, Inc.
umeda.apk #1