A Journey Through MV Wonderland

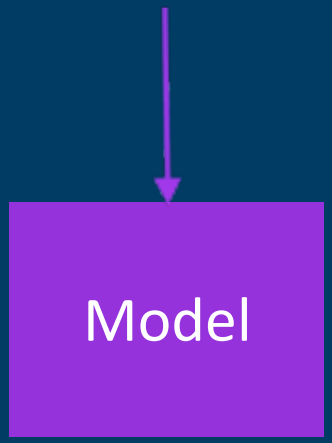
@FMuntenescu

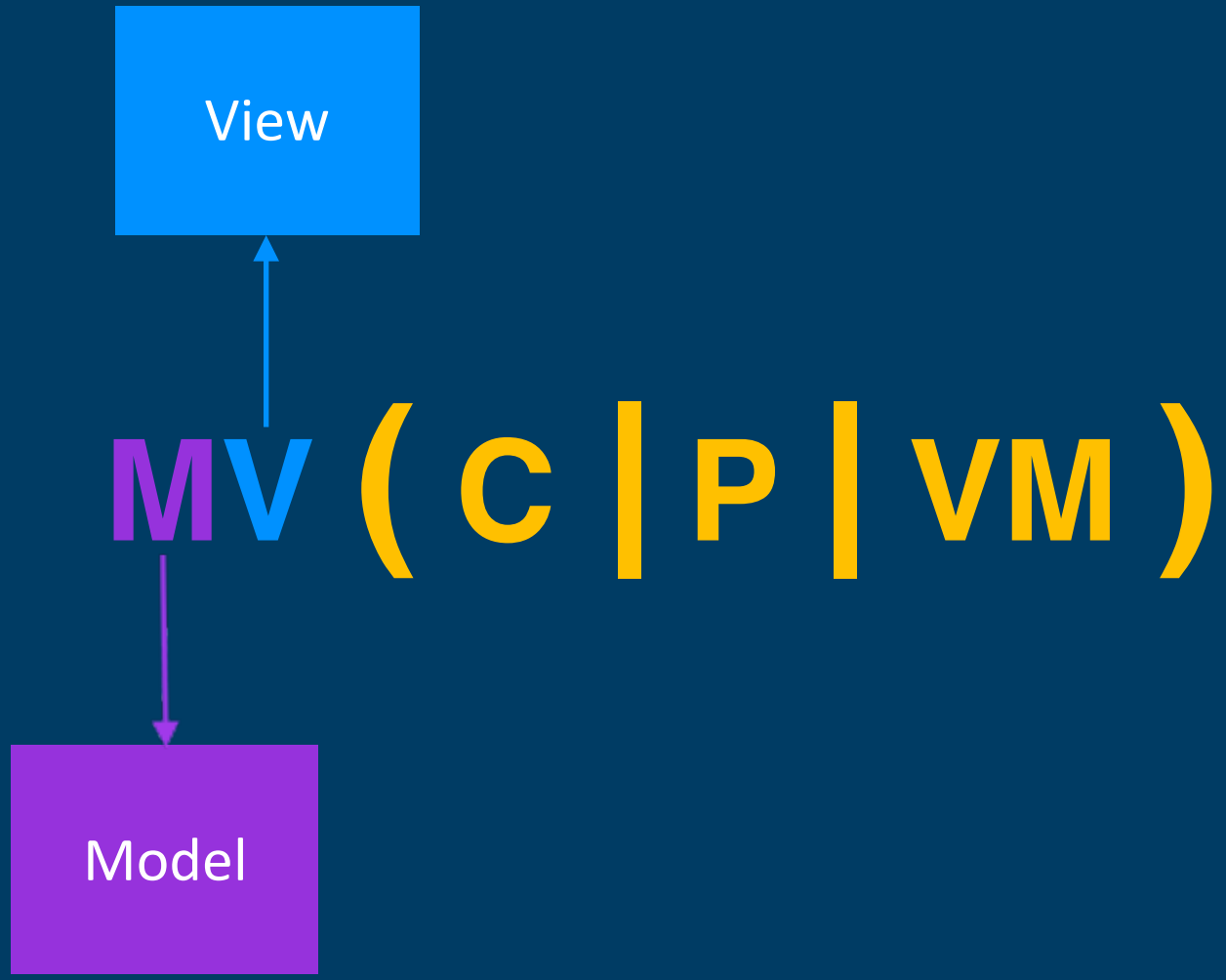
Robust, stable, testable, modular, easy to extend

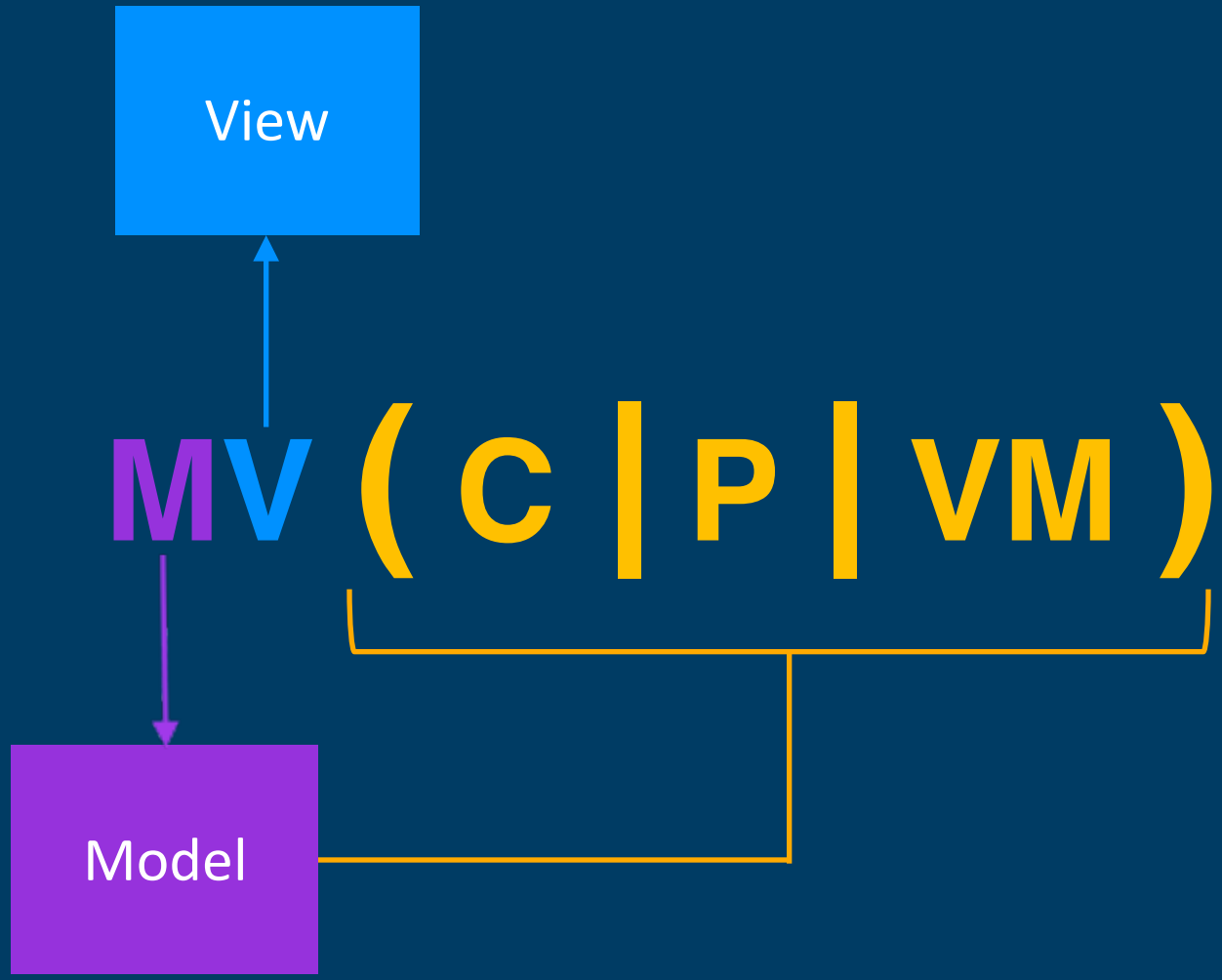
MV (C | P | VM)

@FMuntenescu

MV(C|P|VM)

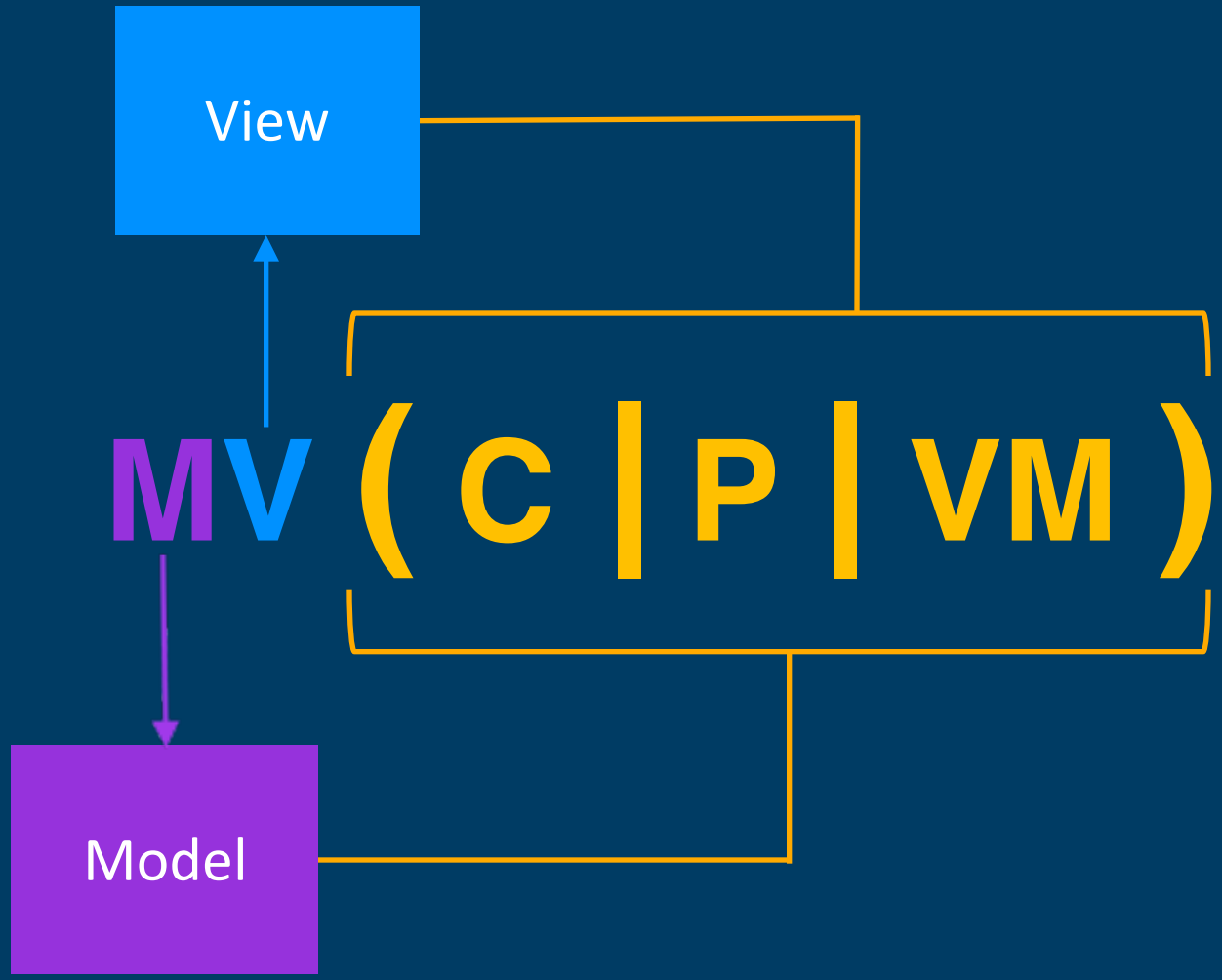



View
MV(C|P|VM)
Model



upday

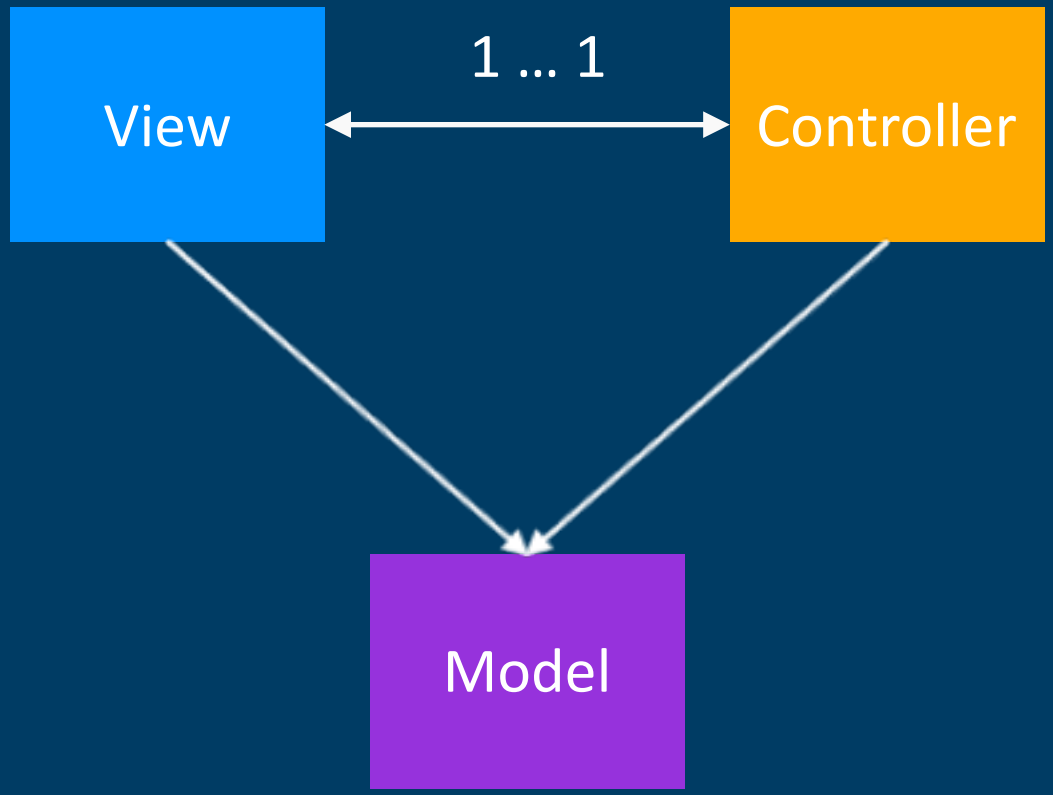
@FMuntenescu



upday

@FMuntenescu

Model – View – Controller

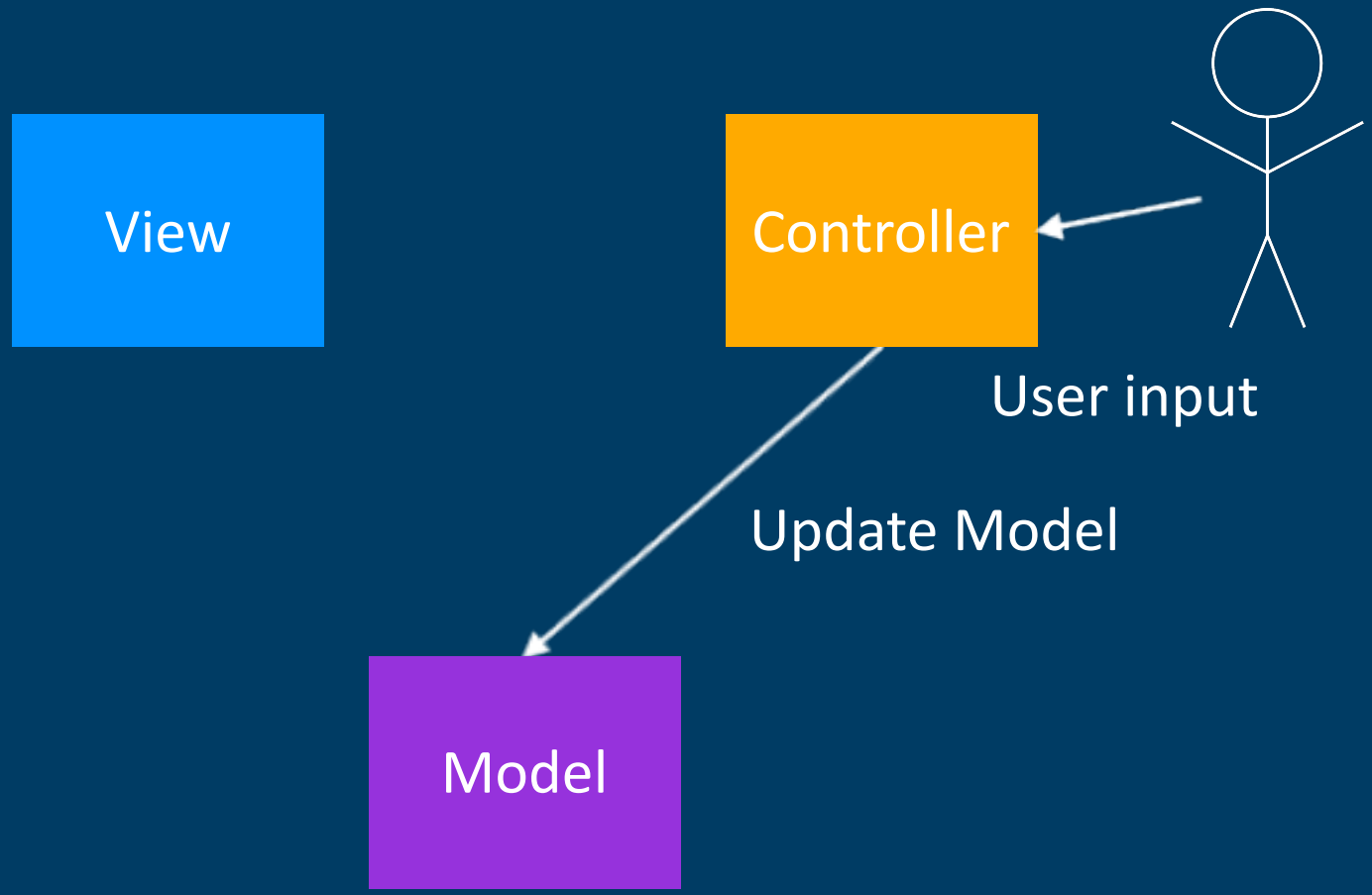

View
1 … 1
Controller
Model

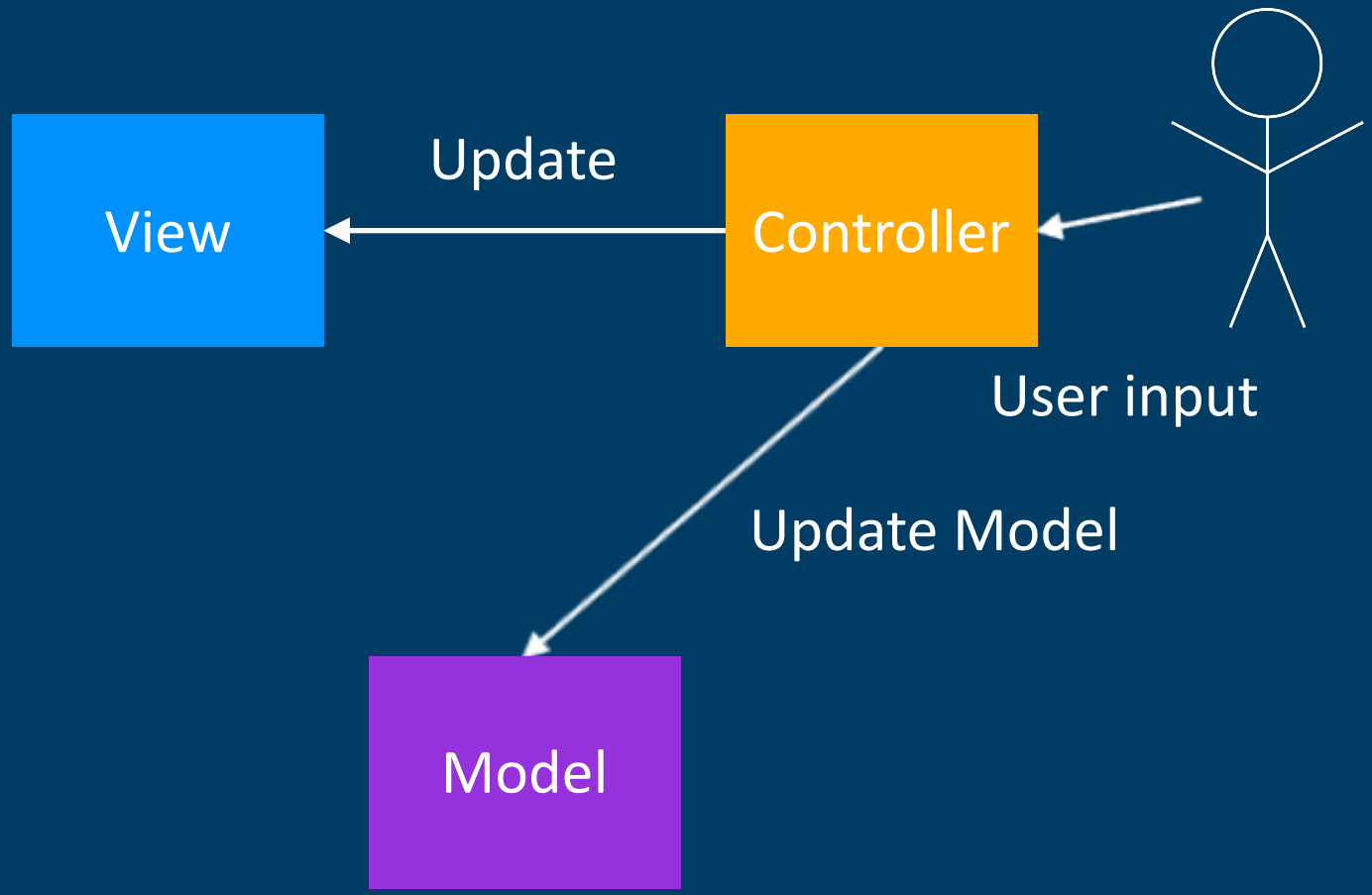
View

Controller

User input

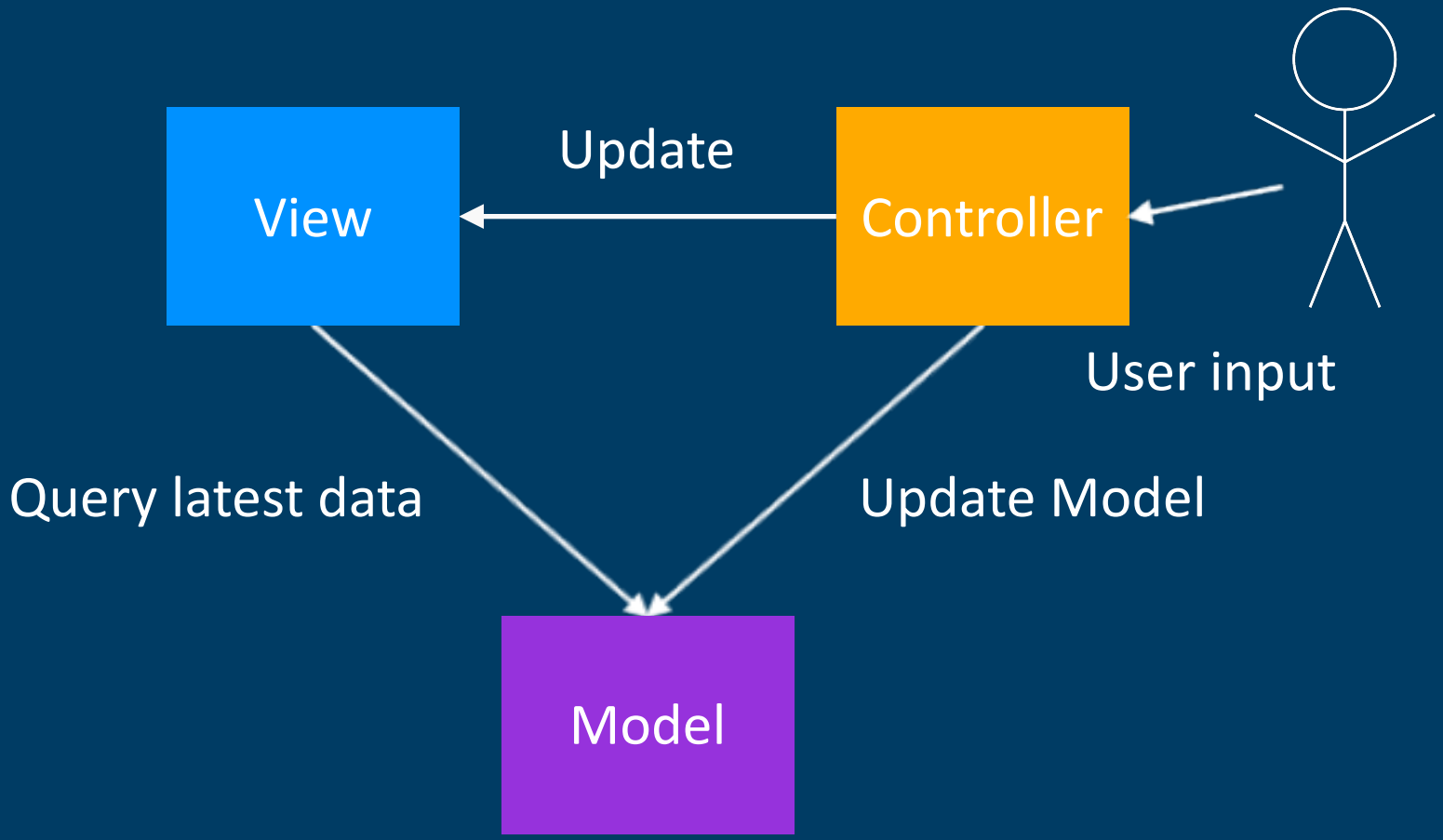
Model


View
Controller
User input
Update Model
Model



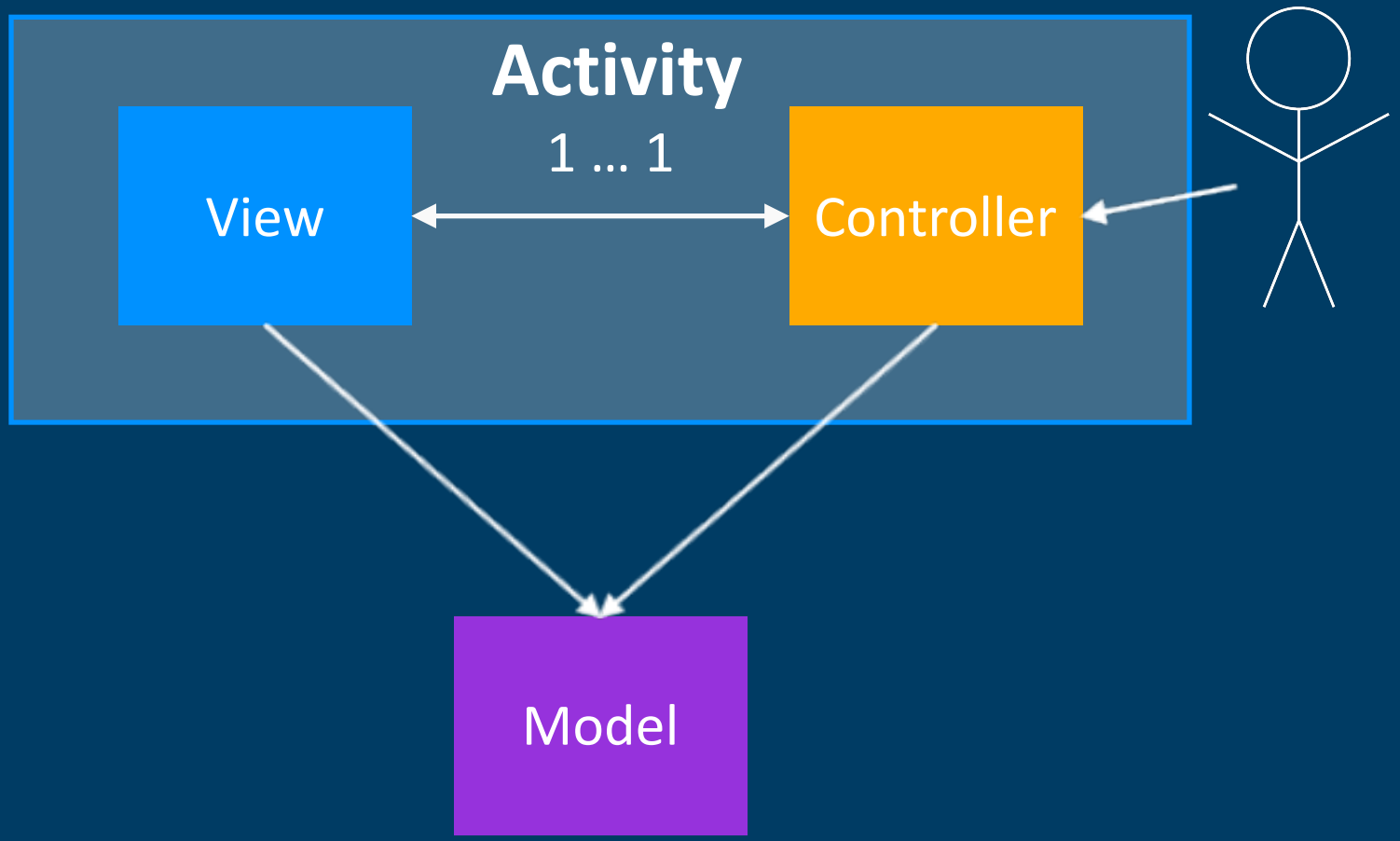
upday

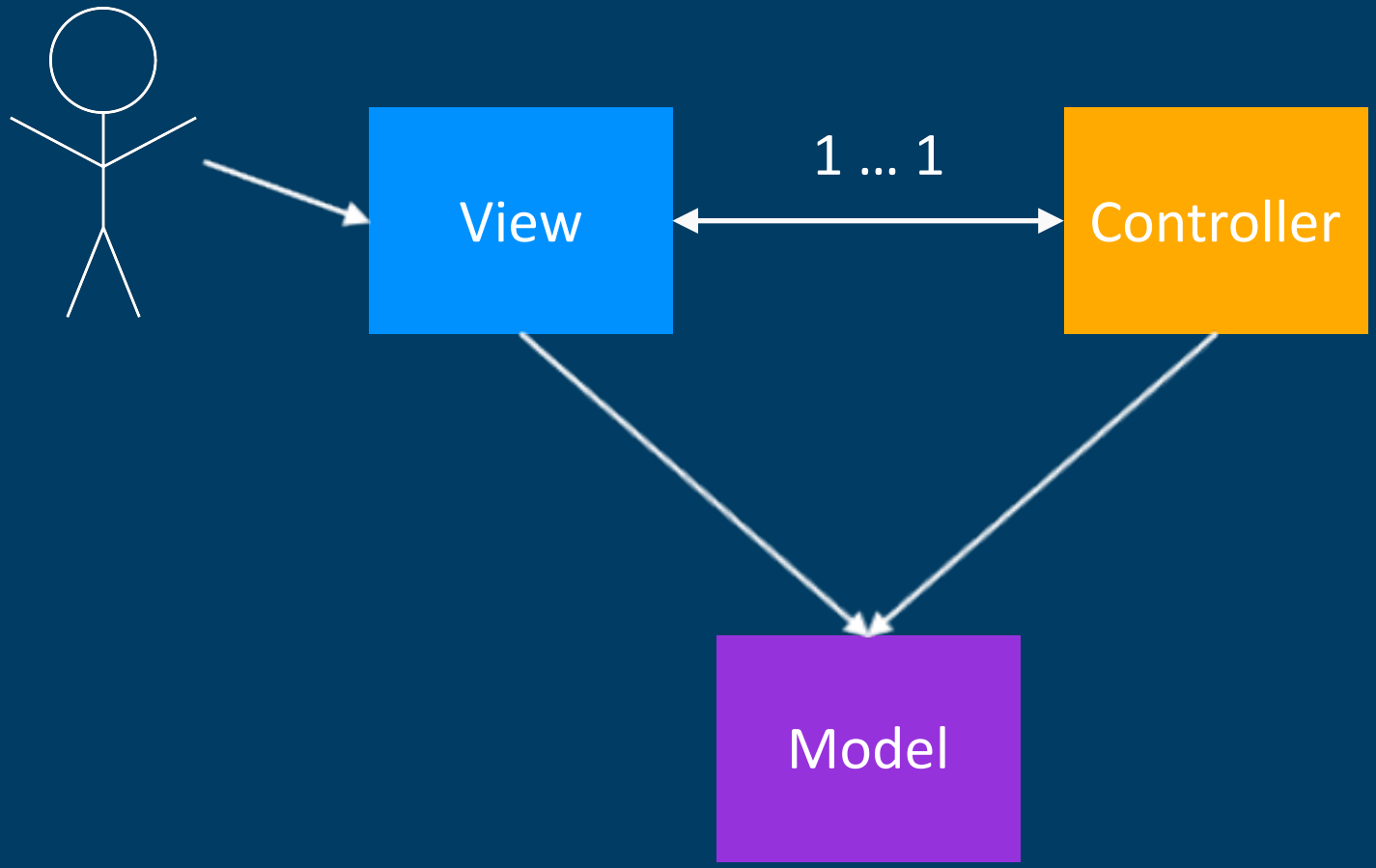
@FMuntenescu


View
Update
Controller
User input
Query latest data
Update Model
Model

Model – View – Controller in Android

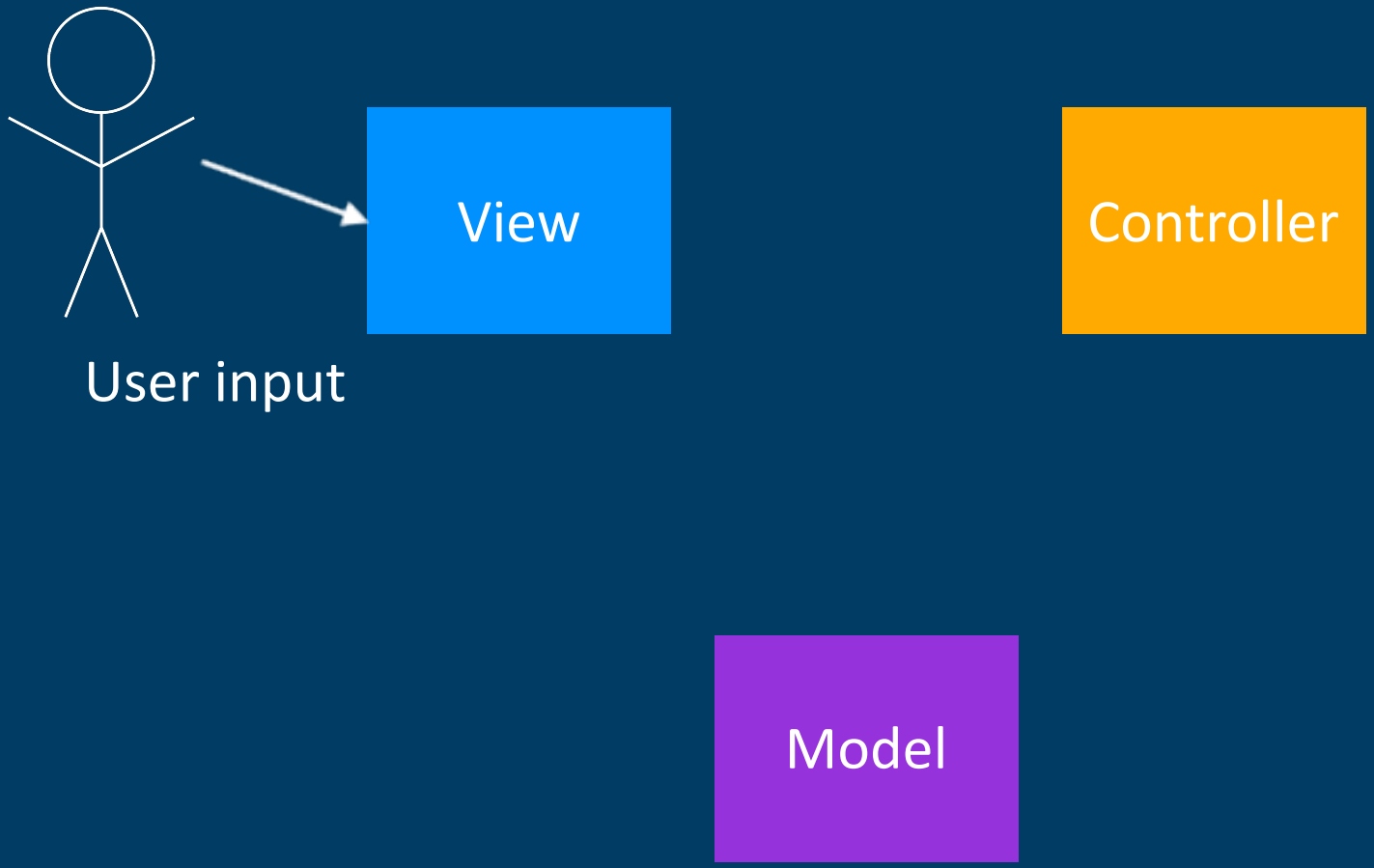
@FMuntenescu

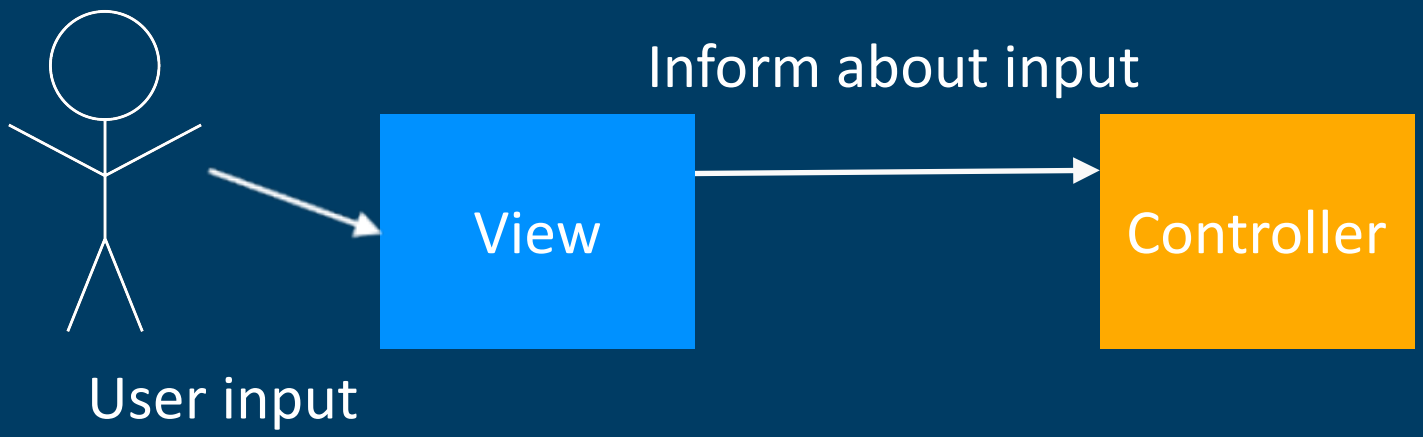

Activity
1 … 1
View
Controller
Model



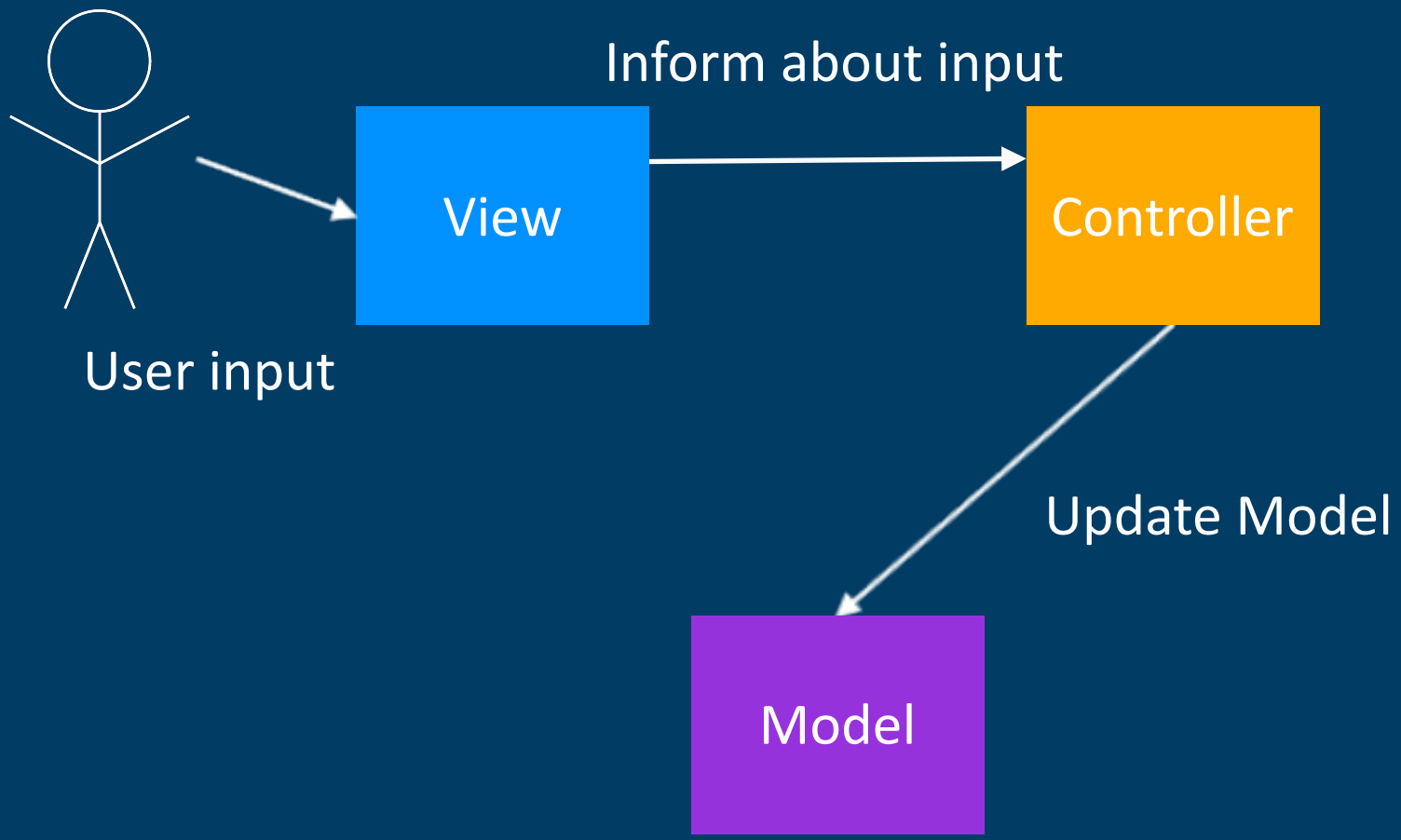
upday

@FMuntenescu


View
Controller
User input
Model

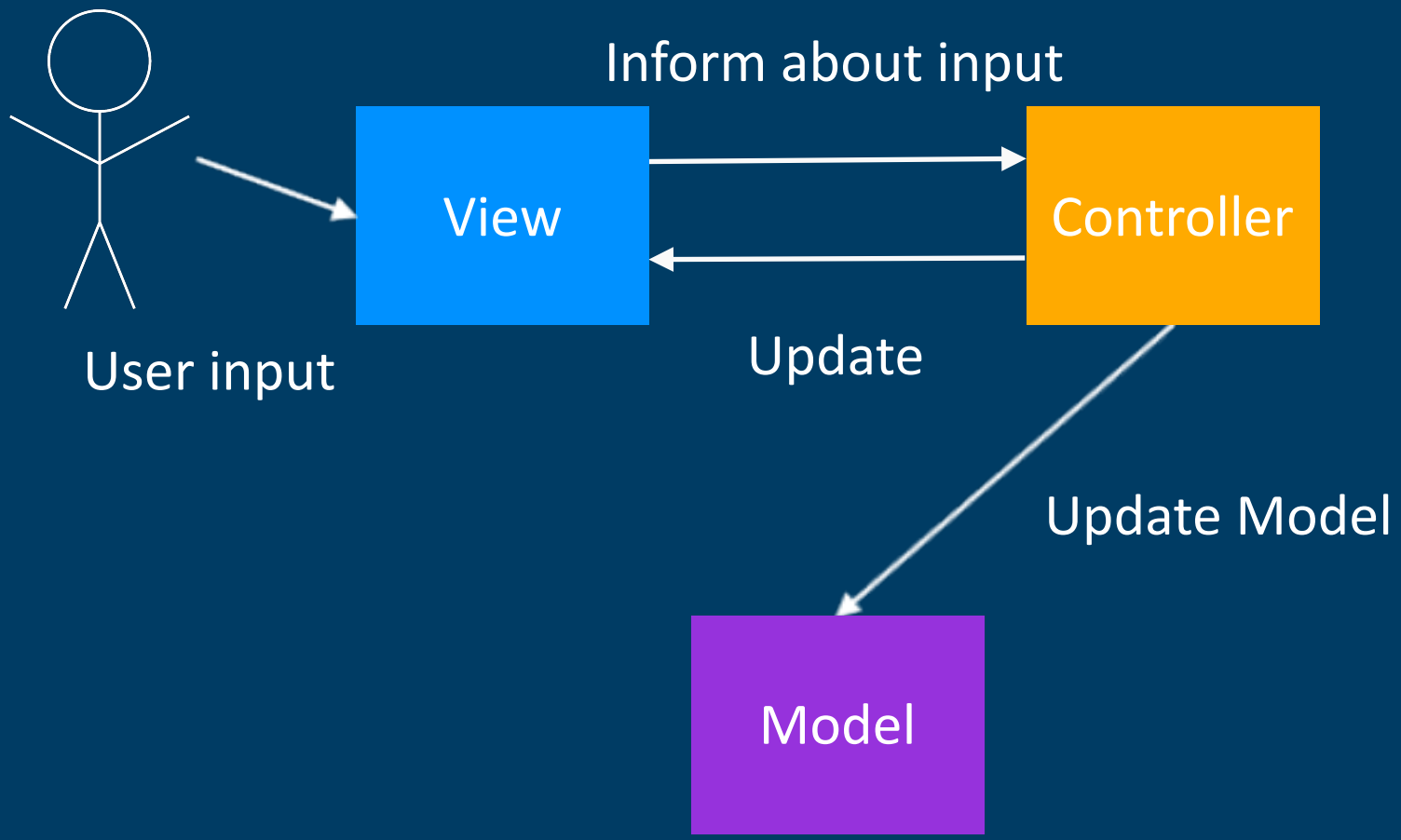


Model



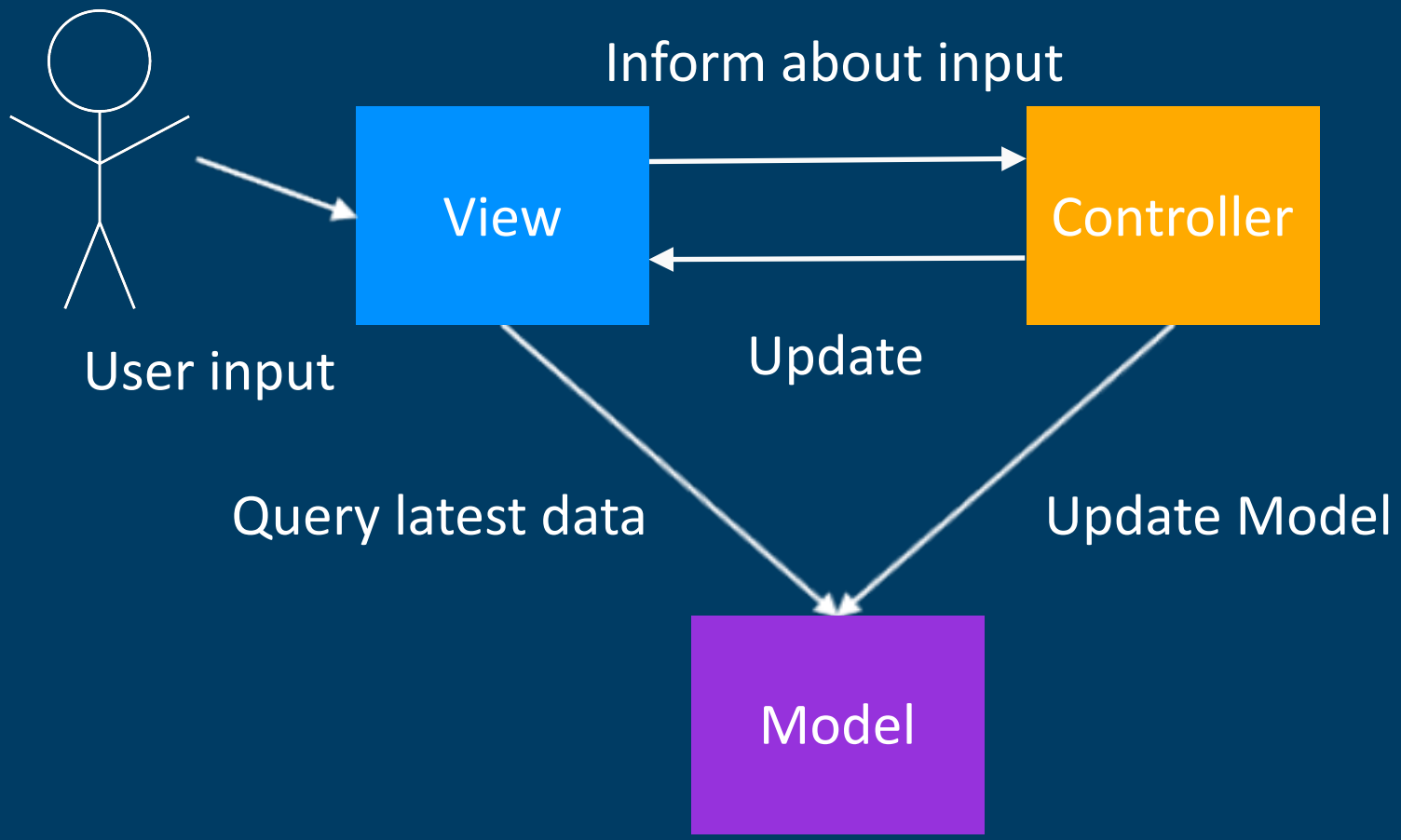
upday

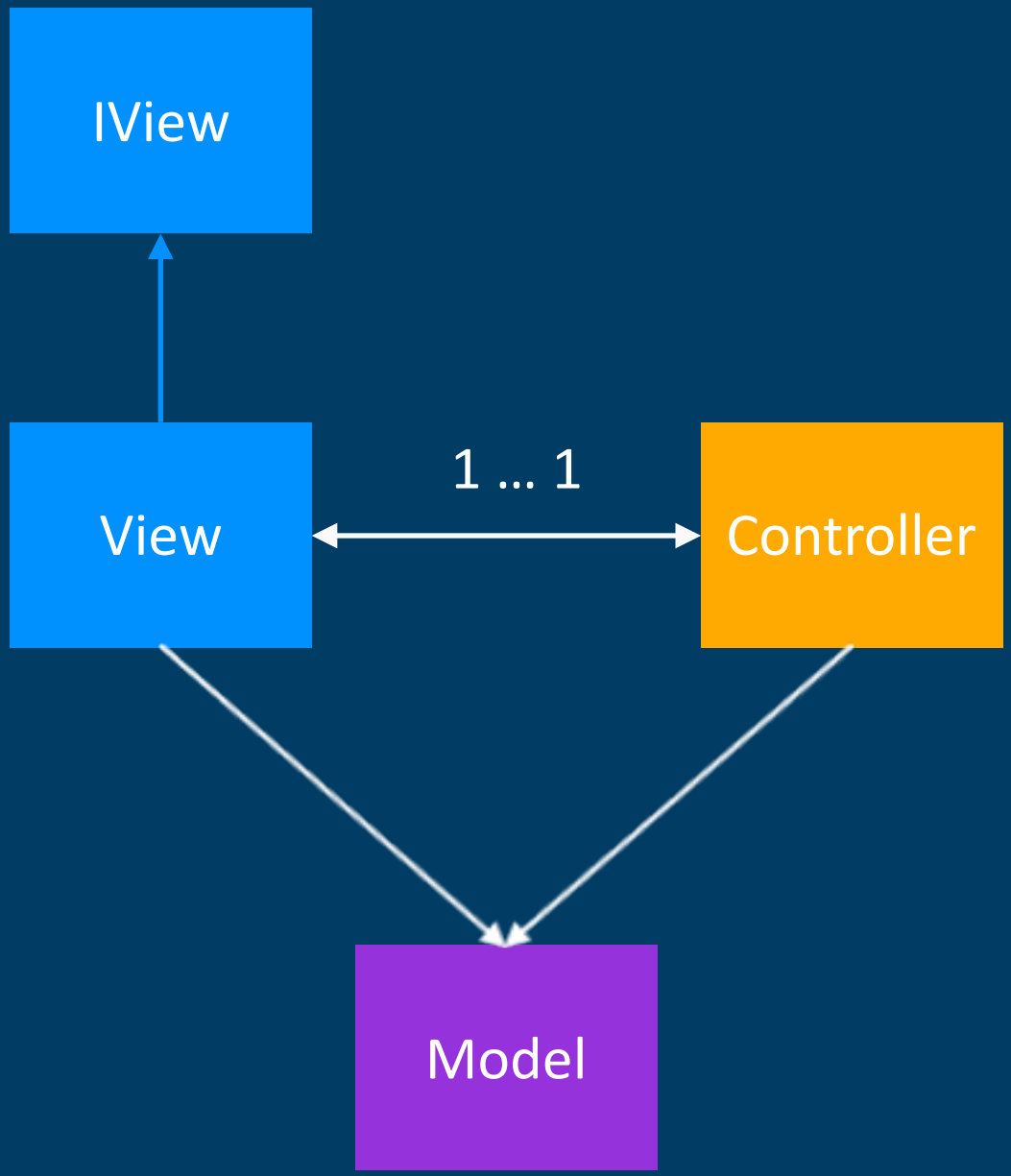
@FMuntenescu



upday

@FMuntenescu


Inform about input
View
Controller
User input
Update
Query latest data
Update Model
Model



upday

@FMuntenescu

Who handles UI logic?

```
class User {
     String firstName;
     String lastName;
     …
}
```

```
class User {
    String firstName;
    String lastName;
    …
}
```

```
displayName =
        lastName + “, “ + firstName
```

```
class User {
    String firstName; Jane
    String lastName; Doe
    …
}
```

```
displayName =  Doe, Jane
        lastName + “, “ + firstName
```

View?

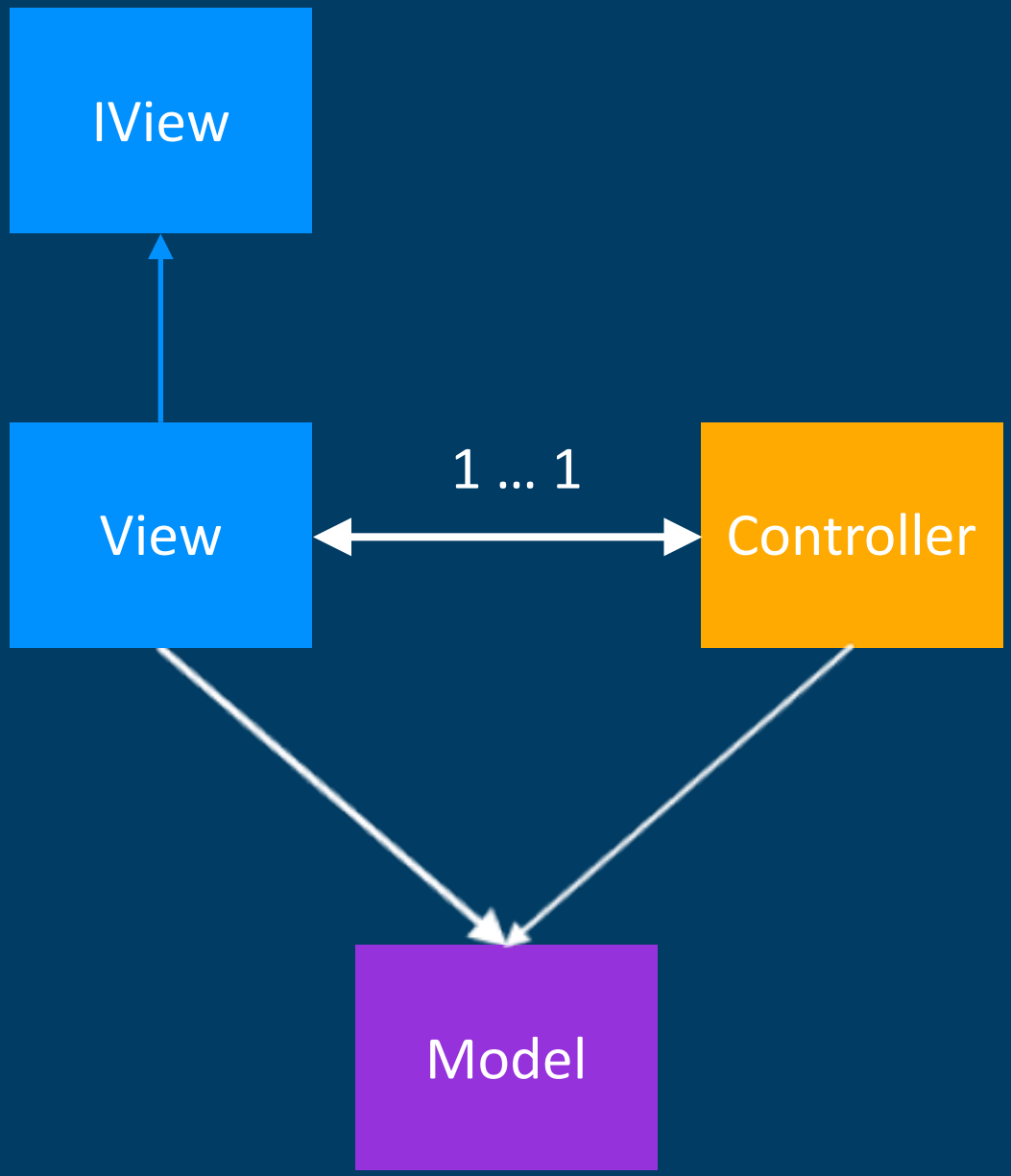
```
User user = userModel.getUser();

nameTextView.setText(
                user.getLastName()
                + ", "
                + user.getFirstName())
```

Model?

```
String name = userModel.getDisplayName();

nameTextView.setText(name);
```

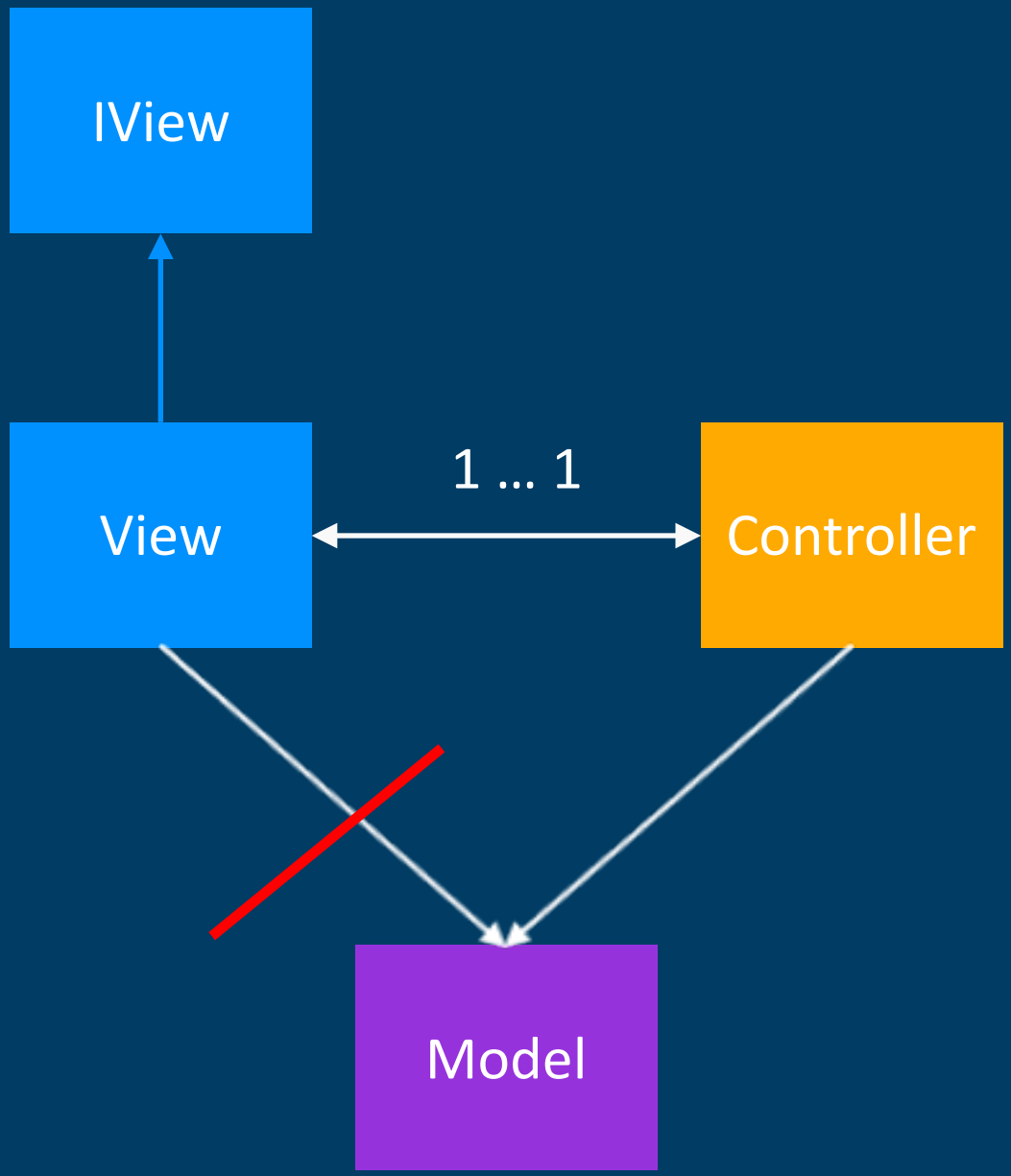
The View knows too much!

@FMuntenescu



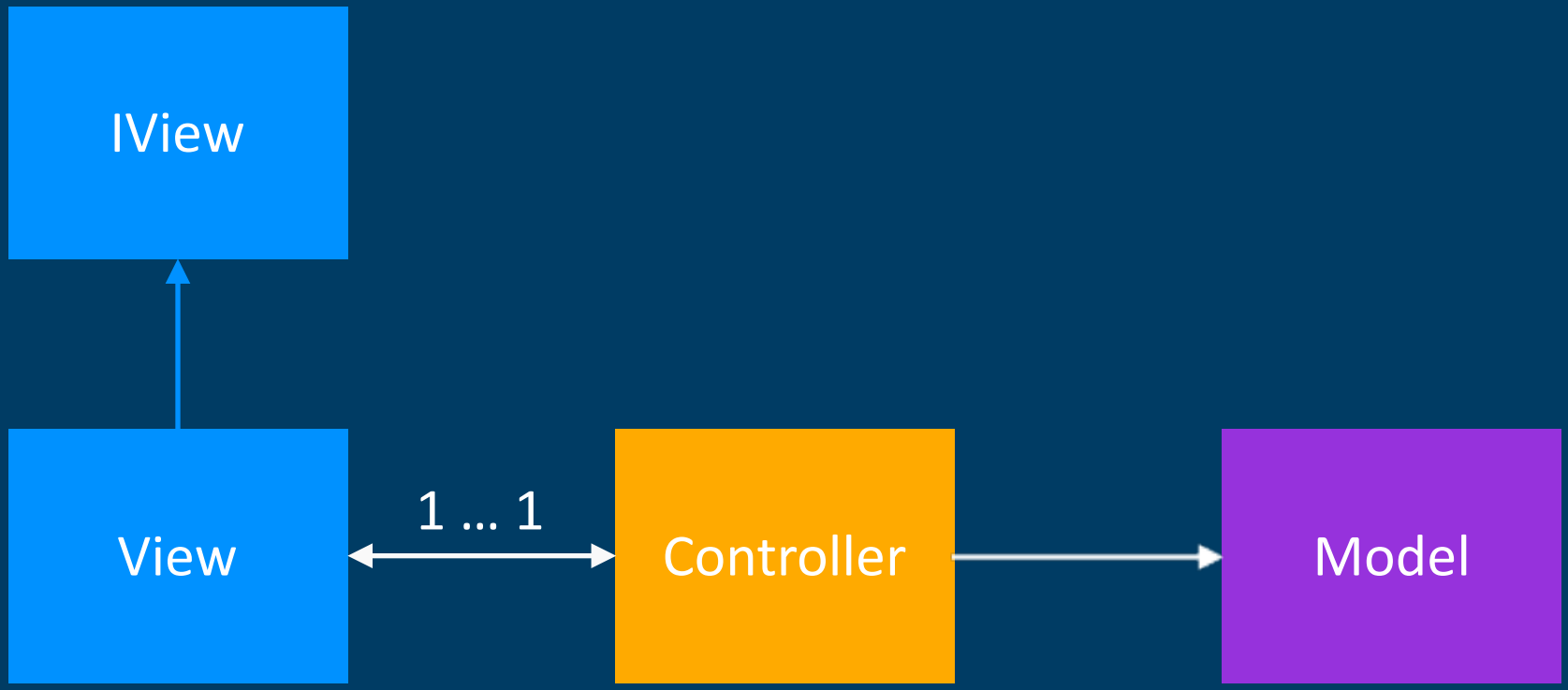
upday

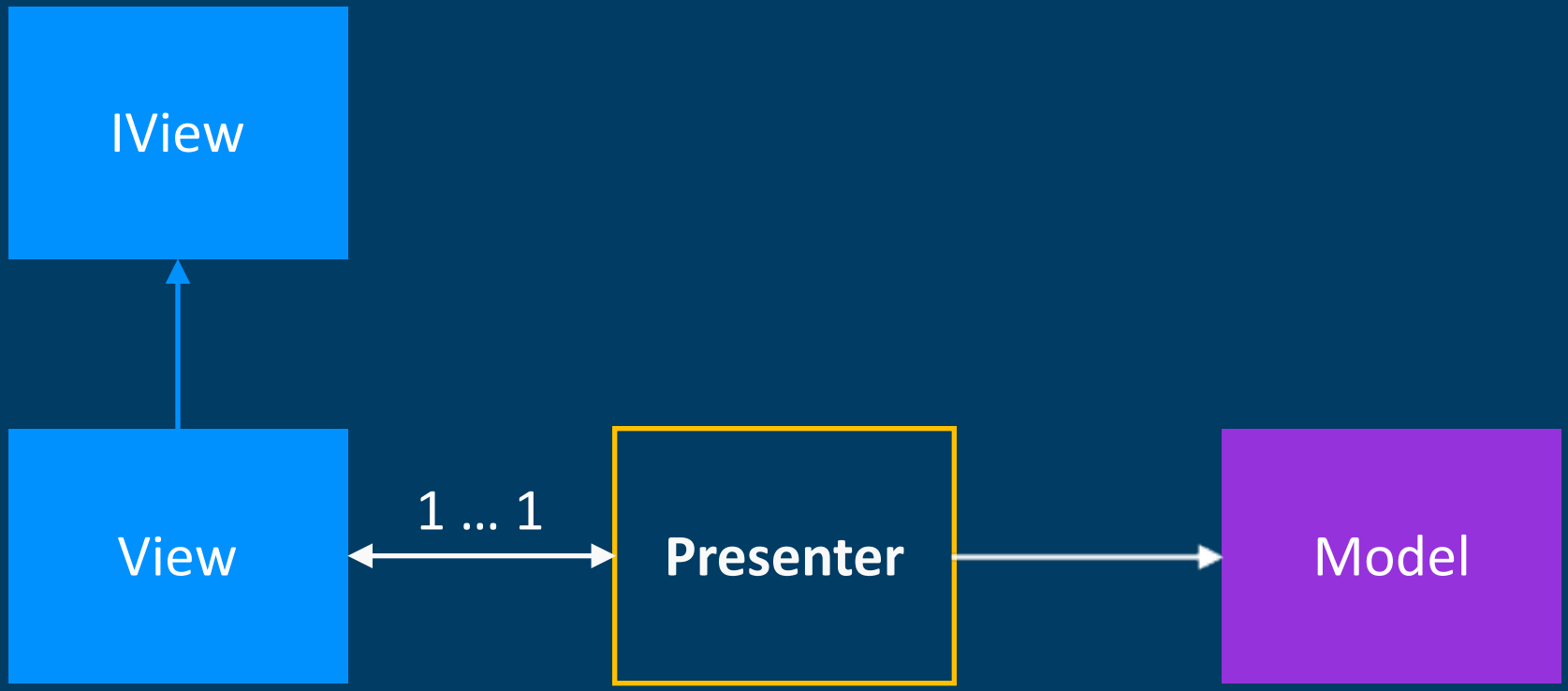
@FMuntenescu



updday

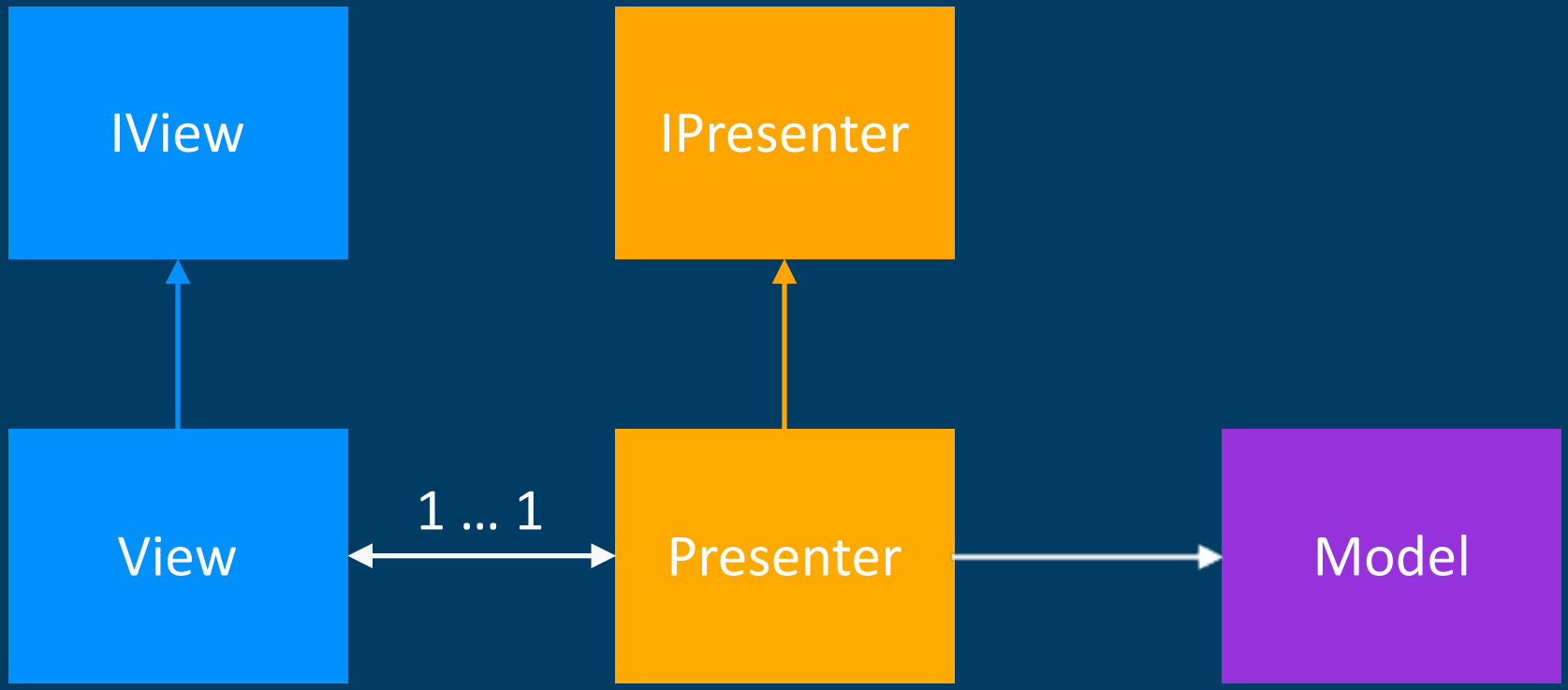
@FMuntenescu


IView
View
1 … 1
Controller
Model

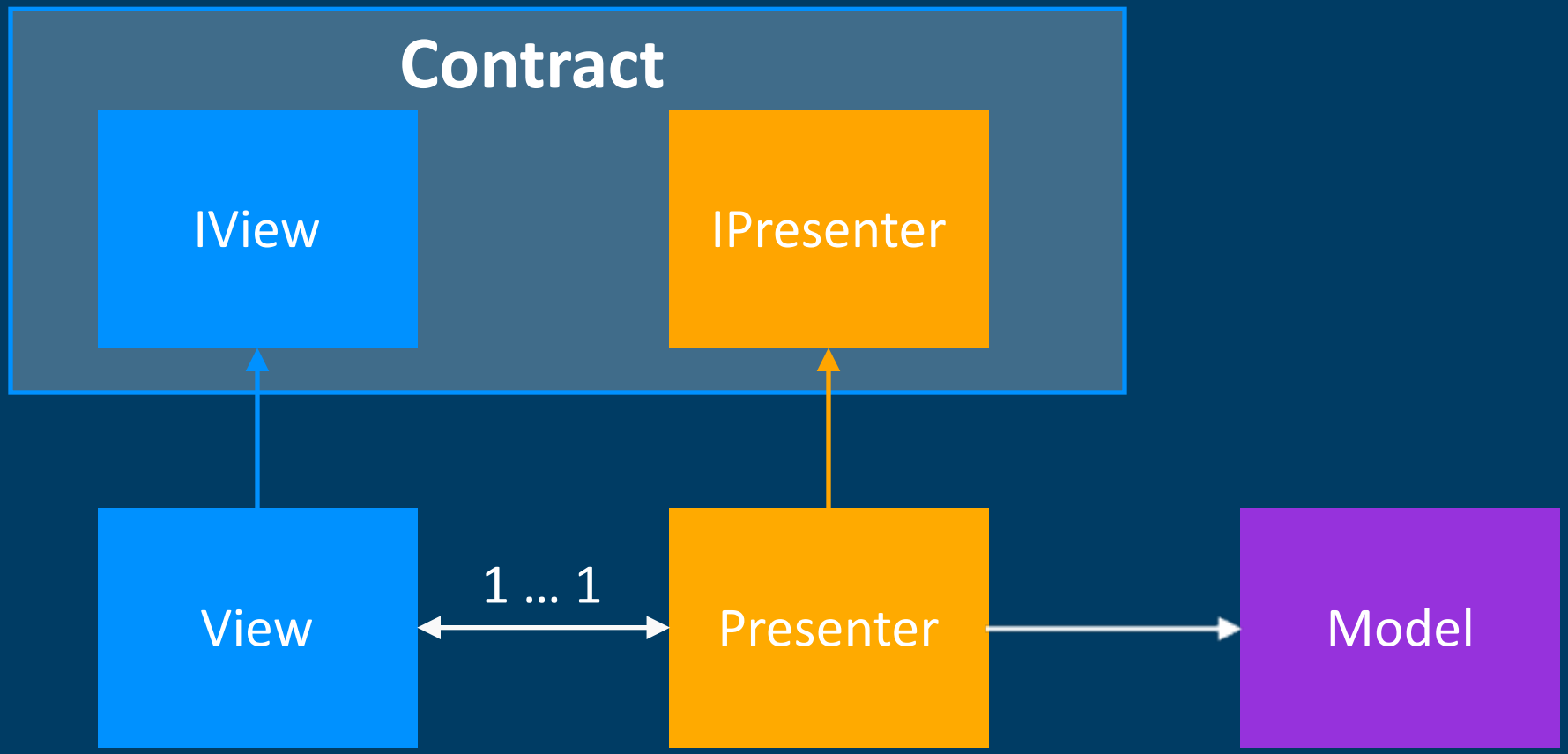

IView
View
1 ... 1
Presenter
Model

Model – View – Presenter

@FMuntenescu



updaY


Contract
IView
IPresenter
View
1 … 1
Presenter
Model

```
interface IView {
    void setName(String name);
}
```

```
IUserModel userModel;

IView view;

Presenter(IUserModel userModel,
          IView view) {
    this.userModel = userModel;
    this.view = view;
}
```

```
IUserModel userModel;

IView view;

Presenter(IUserModel userModel,
          IView view) {
    this.userModel = userModel;
    this.view = view;
}
```

```
interface IPresenter {
    void onLoad();
}
```

```
@Override
public void onLoad() {

User user = userModel.getUser();
String displayName =
                user.getFirstName()
                + ", "
                + user.getLastName();

view.setName(displayName);
}
```

```
@Override
public void onLoad() {

User user = userModel.getUser();
String displayName =
                user.getFirstName()
                + ", "
                + user.getLastName();

view.setName(displayName);
}
```

```
@Override
public void onLoad() {

User user = userModel.getUser();
String displayName =
                user.getFirstName()
                + ", "
                + user.getLastName();

view.setName(displayName);
}
```

```
@Mock
IUserModel userModel;

@Mock
IView view;

Presenter presenter;

@Before
public void setup() {
     …
     presenter =
          new Presenter(userModel, view);
}
```

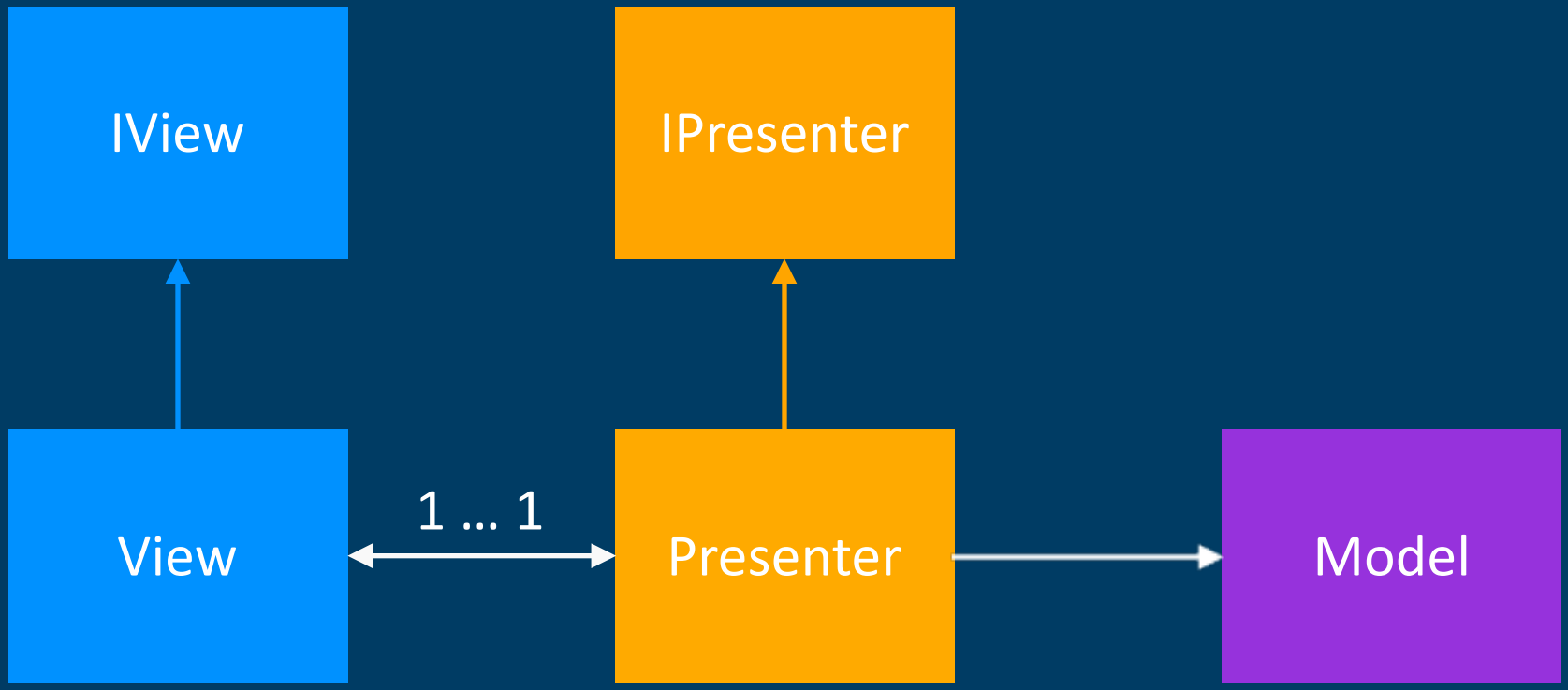
```
@Mock
IUserModel userModel;

@Mock
IView view;

Presenter presenter;

@Before
public void setup() {
     …
     presenter =
          new Presenter(userModel, view);
}
```

```
@Test
public void displayName_setOnLoad() {
when(userModel.getUser())
     .thenReturn(new User(“Jane”, “Doe”);

presenter.onLoad();

verify(view).setName(“Doe, Jane”);
}
```

```
@Test
public void displayName_setOnLoad() {
when(userModel.getUser())
      .thenReturn(new User(“Jane”, “Doe”);

presenter.onLoad();

verify(view).setName(“Doe, Jane”);
}
```

```
@Test
public void displayName_setOnLoad() {
when(userModel.getUser())
     .thenReturn(new User(“Jane”, “Doe”);

presenter.onLoad();

verify(view).setName(“Doe, Jane”);
}
```

```
@Test
public void displayName_setOnLoad() {
when(userModel.getUser())
     .thenReturn(new User(“Jane”, “Doe”);

presenter.onLoad();

verify(view).setName(“Doe, Jane”);
}
```



updaY

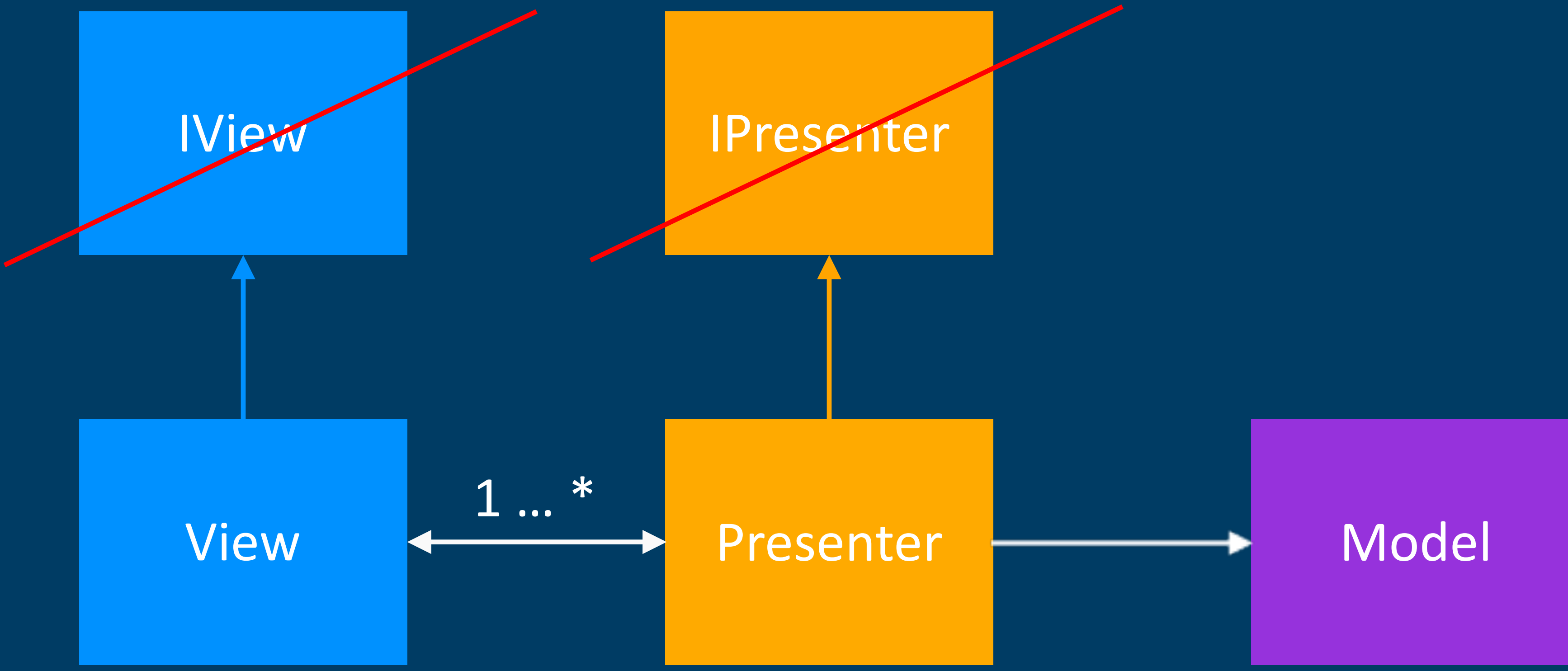
```
IUserModel userModel;
IView view;
Presenter(IUserModel userModel, IView view) {
    this.userModel = userModel;
    this.view = view;
}

void onLoad() {
      …
     view.setName(displayName);
}
```

```
IUserModel userModel;
IView view;
Presenter(IUserModel userModel, IView view) {
    this.userModel = userModel;
    this.view = view;
}

void onLoad() {
    …
    view.setName(displayName);
}

Observable<String> getName() {
    …
}
```

```
class View {

…
viewModel.getName()
     .subscribe(name-> setName(name));

…
private void setName(String name){
     nameTextView.setText(name);

}
```

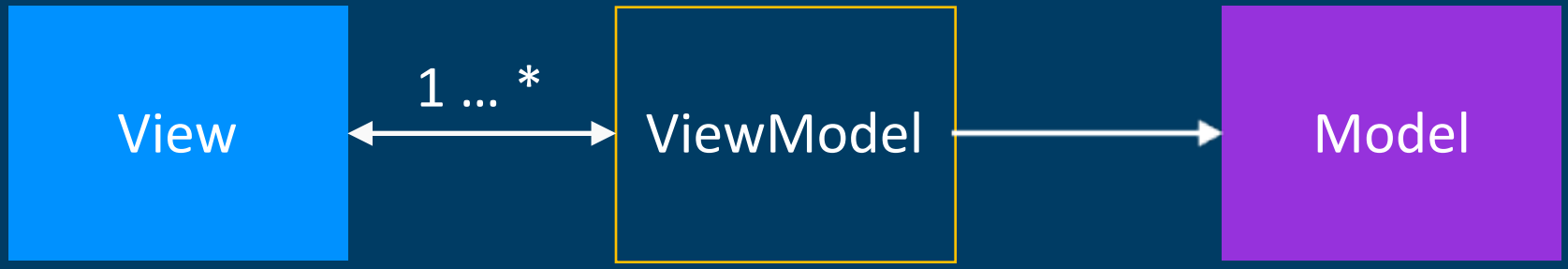
```
IUserModel userModel;
IView view;

Presenter(IUserModel userModel,
          IView view) {
    userModel = userModel;
    view = view;
}
```

```
interface IView {
    void setName(String name);
}
```



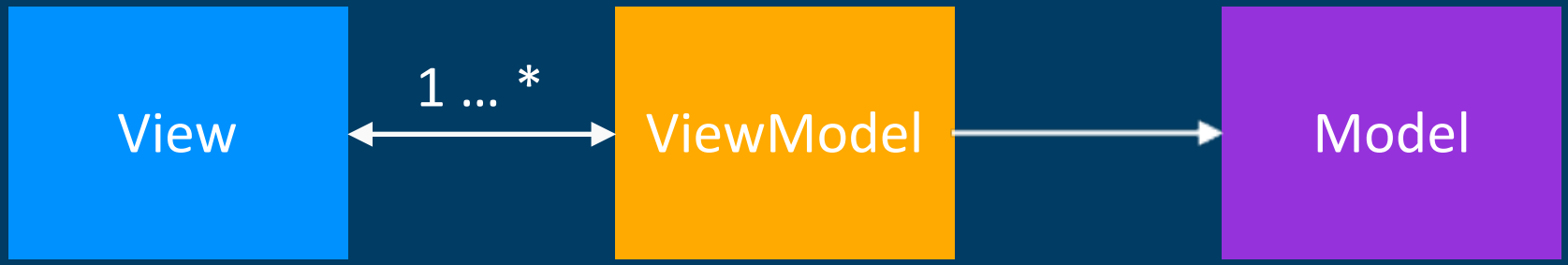
upday

@FMuntenescu


View
1 ... *
ViewModel
Model

Model – View – ViewModel

@FMuntenescu


View
1 … *
ViewModel
Model

```
class User {
    String firstName;
    String lastName;

    Date dateOfBirth;
}
```

```
class User {
    String firstName; Jane
    String lastName; Doe

    Date dateOfBirth; 04.04.1987
}
```

```
class User {
    String firstName; Jane
    String lastName; Doe

    Date dateOfBirth; 04.04.1987
}
```

```
Name: Doe, Jane
Age: 29
```

```
class ViewModel {

    Observable<String> getName() {
    …
    }

    Observable<Integer> getAge() {
    …
    }
}
```

```
class ViewModel {

Observable<DisplayableUser> getUser() {
    …
}
}
```

```
class ViewModel {

Observable<DisplayableUser> getUser() {
    …
}
}

 class DisplayableUser {
     String mName;
     int mAge;
 }
```

```
@Mock
IUserModel userModel;

ViewModel viewModel;

@Before
void setup() {
     …
     viewModel = new ViewModel(userModel);
}
```

```
@Mock
IUserModel userModel;

ViewModel viewModel;

@Before
public void setup() {
      …
      viewModel = new ViewModel(userModel);
}
```

```
@Test
public void getUser_emitsCorrectValue() {
when(userModel.getUser())
          .thenReturn(new User(“Jane”,”Doe”,
                        new Date(4,4,1987));

viewModel.getUser()
          .subscribe(testSubscriber);

testSubscriber.assertValue(expectedUser);
}
```

```
@Test
public void getUser_emitsCorrectValue() {
when(userModel.getUser())
          .thenReturn(new User(“Jane”,”Doe”,
                              new Date(4,4,1987));

viewModel.getUser()
          .subscribe(testSubscriber);

testSubscriber.assertValue(expectedUser);
}
```

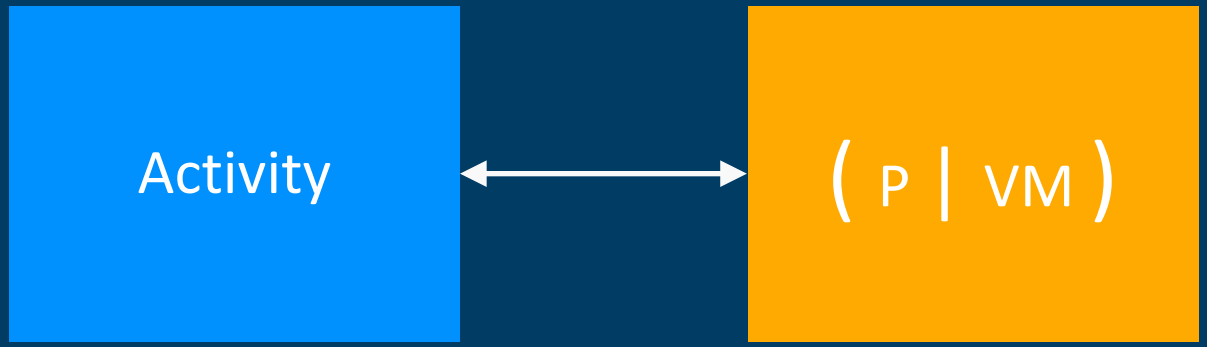
```
@Test
public void getUser_emitsCorrectValue() {
when(userModel.getUser())
          .thenReturn(new User(“Jane”,”Doe”,
                              new Date(4,4,1987));

viewModel.getUser()
          .subscribe(testSubscriber);

testSubscriber.assertValue(expectedUser);
}
```

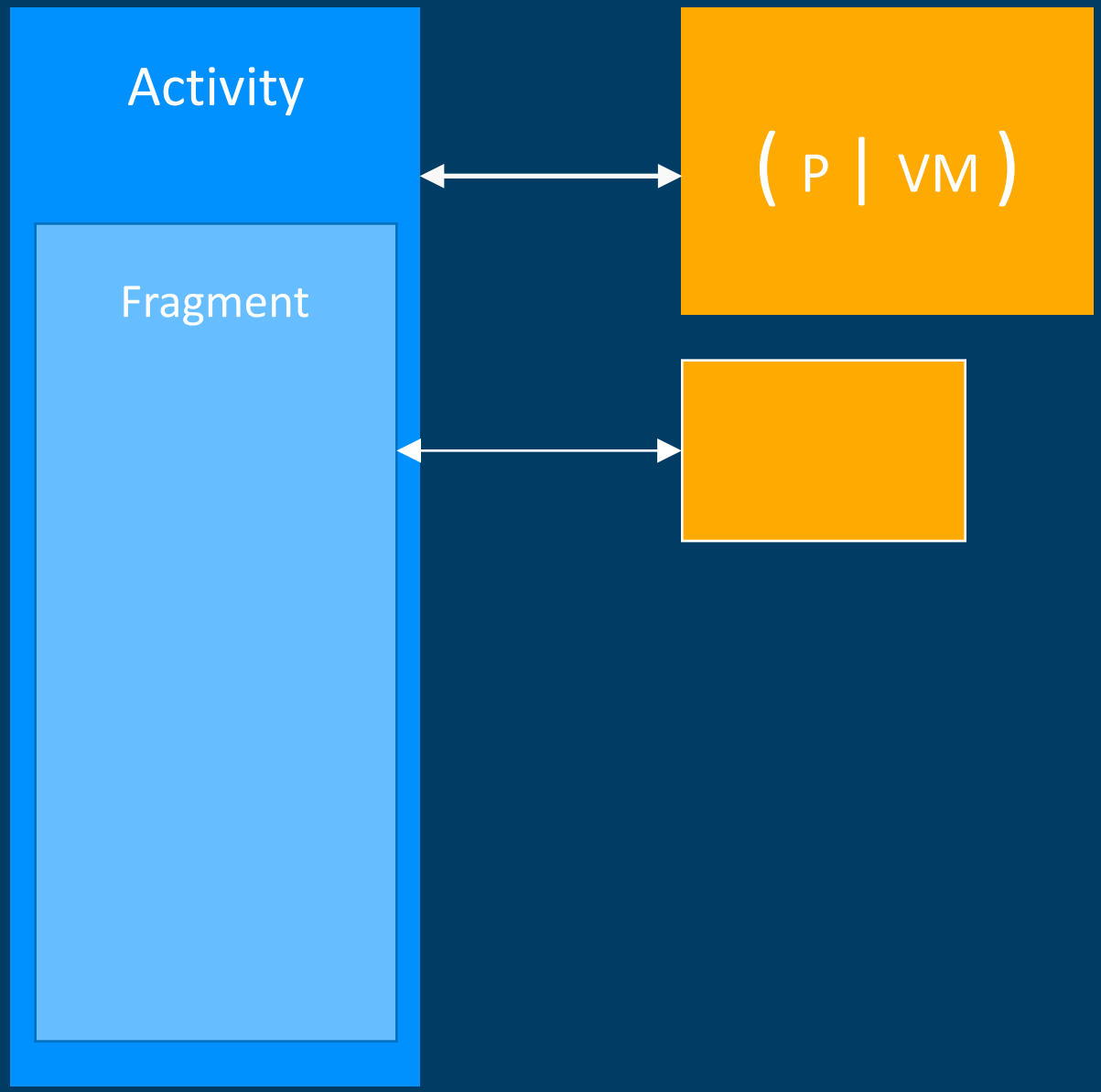
```
@Test
public void getUser_emitsCorrectValue() {
when(userModel.getUser())
          .thenReturn(new User(“Jane”,”Doe”,
                        new Date(4,4,1987));

viewModel.getUser()
          .subscribe(testSubscriber);

testSubscriber.assertValue(expectedUser);
}
```

Model – View – Presenter
&
Model – View – ViewModel

```
class Presenter {
     void onLoad() {
          …
          view.setName(displayName);
     }
}

class ViewModel {
     Observable<String> getName() {
          …
     }
}
```

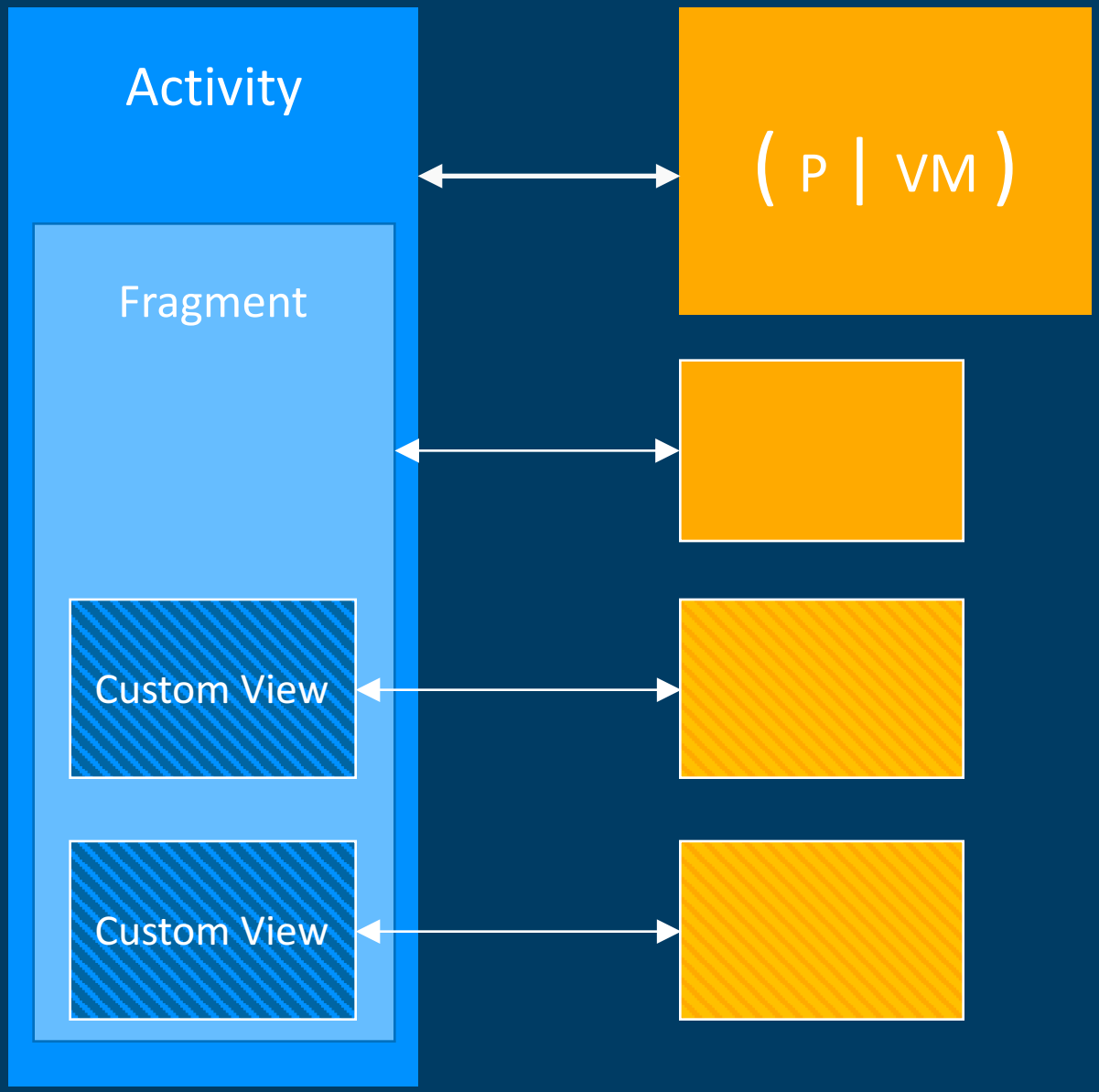
Who is the View?


Activity
(P | VM)



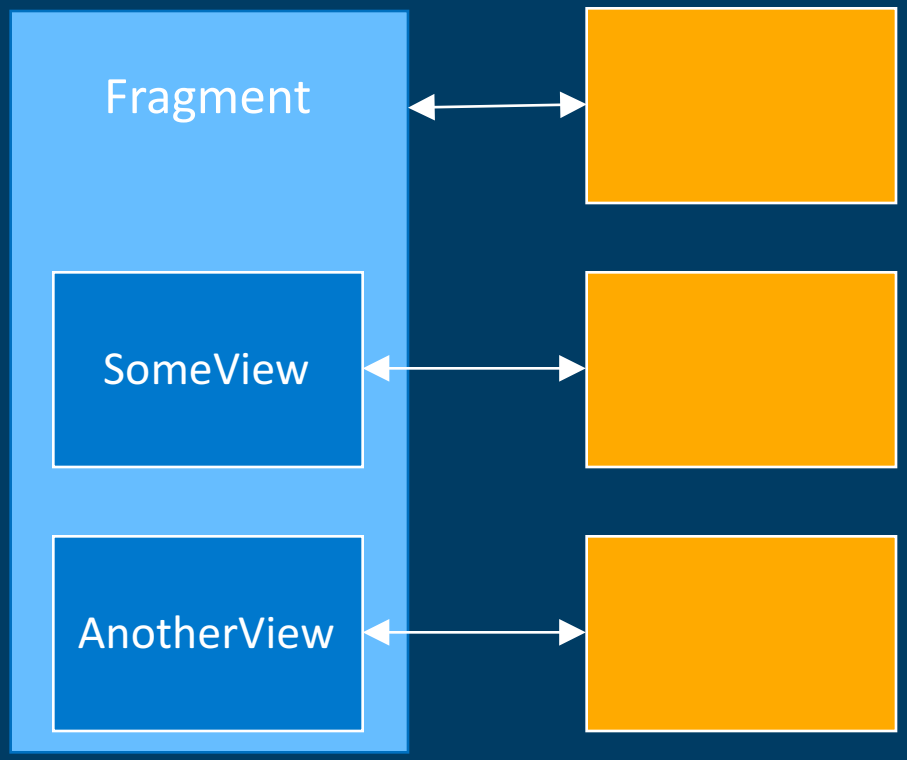
updday

@FMuntenescu


Activity
Fragment
Custom View
Custom View
(P | VM)

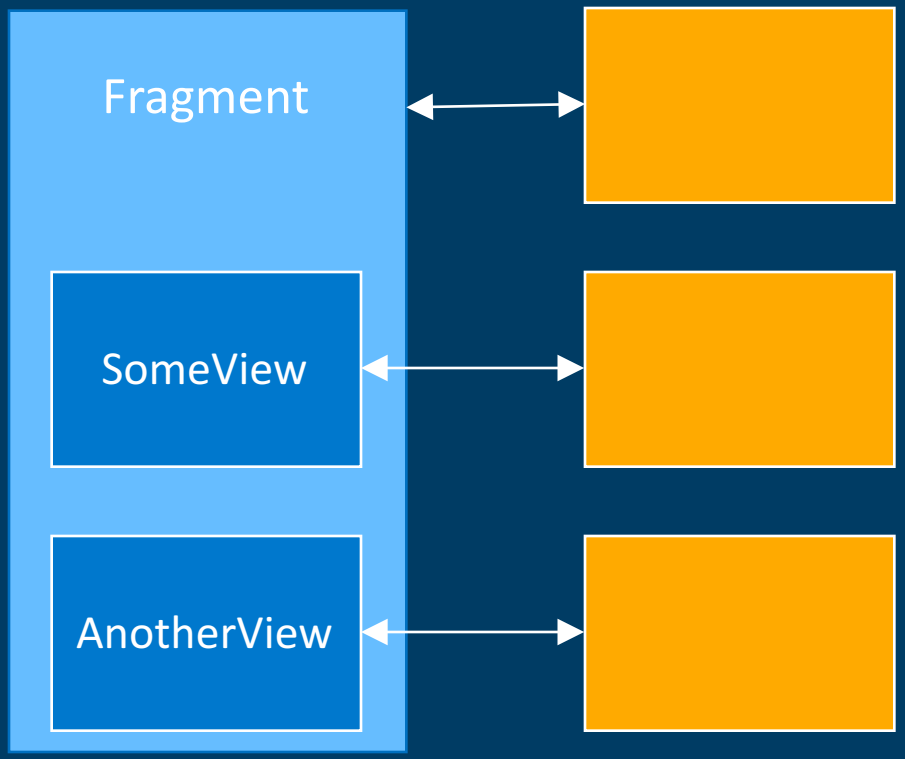
```
class View implements IView {

@Inject
Presenter presenter;

…

presenter.onLoad(this);
```

```
class View {

@Inject
ViewModel viewModel;
```



updday

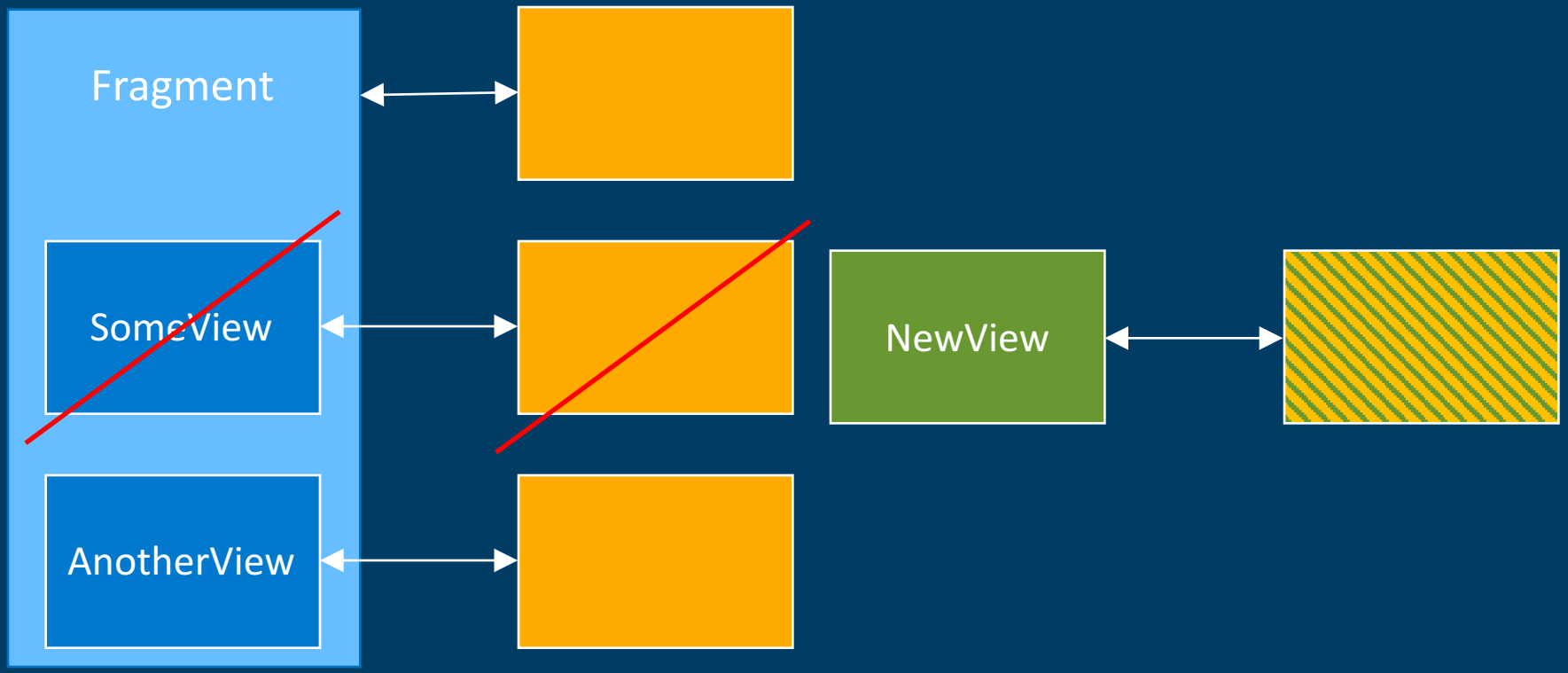
@FMuntenescu




```
<RelativeLayout>
    <my.custom.SomeView />
    <my.custom.AnotherView />
</RelativeLayout>
```

updaY

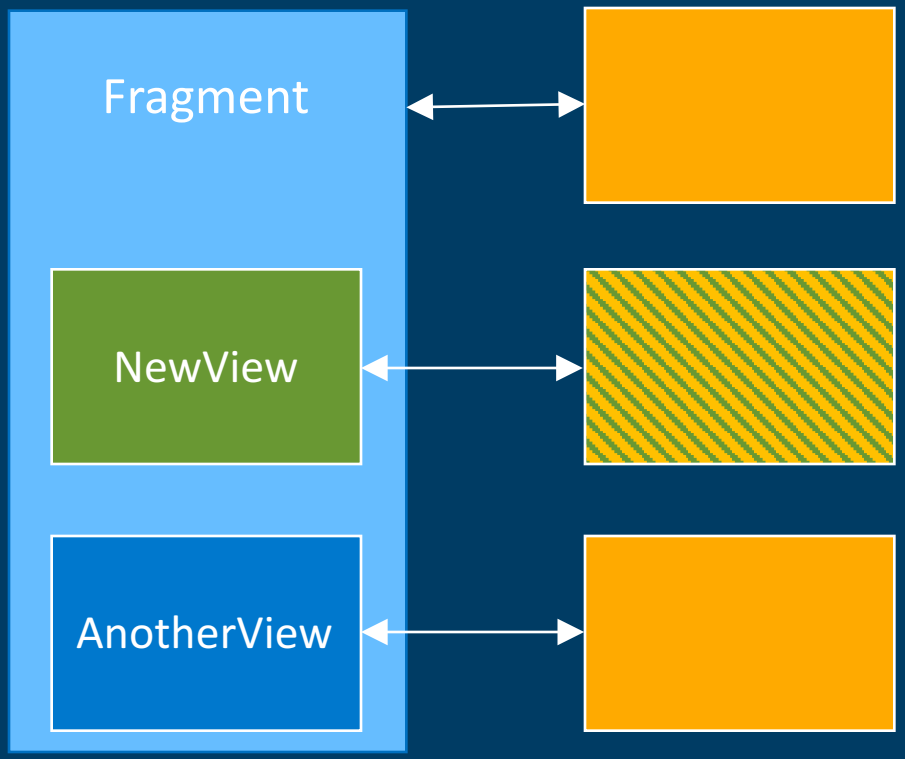
@FMuntenescu




```
<RelativeLayout>
    <my.custom.SomeView />
    <my.custom.AnotherView />
</RelativeLayout>
```

upday

@FMuntenescu




```
<RelativeLayout>
    <my.custom.SomeView />
    <my.custom.NewView />
    <my.custom.AnotherView />
</RelativeLayout>
```

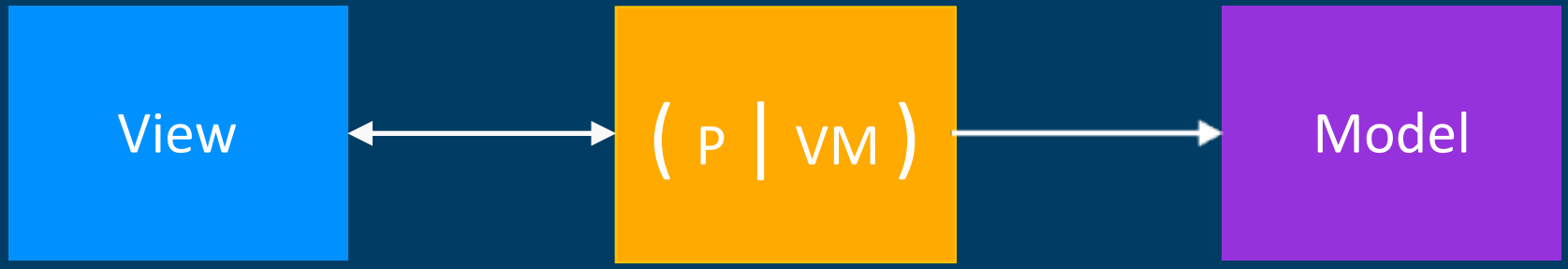
How to split Views?

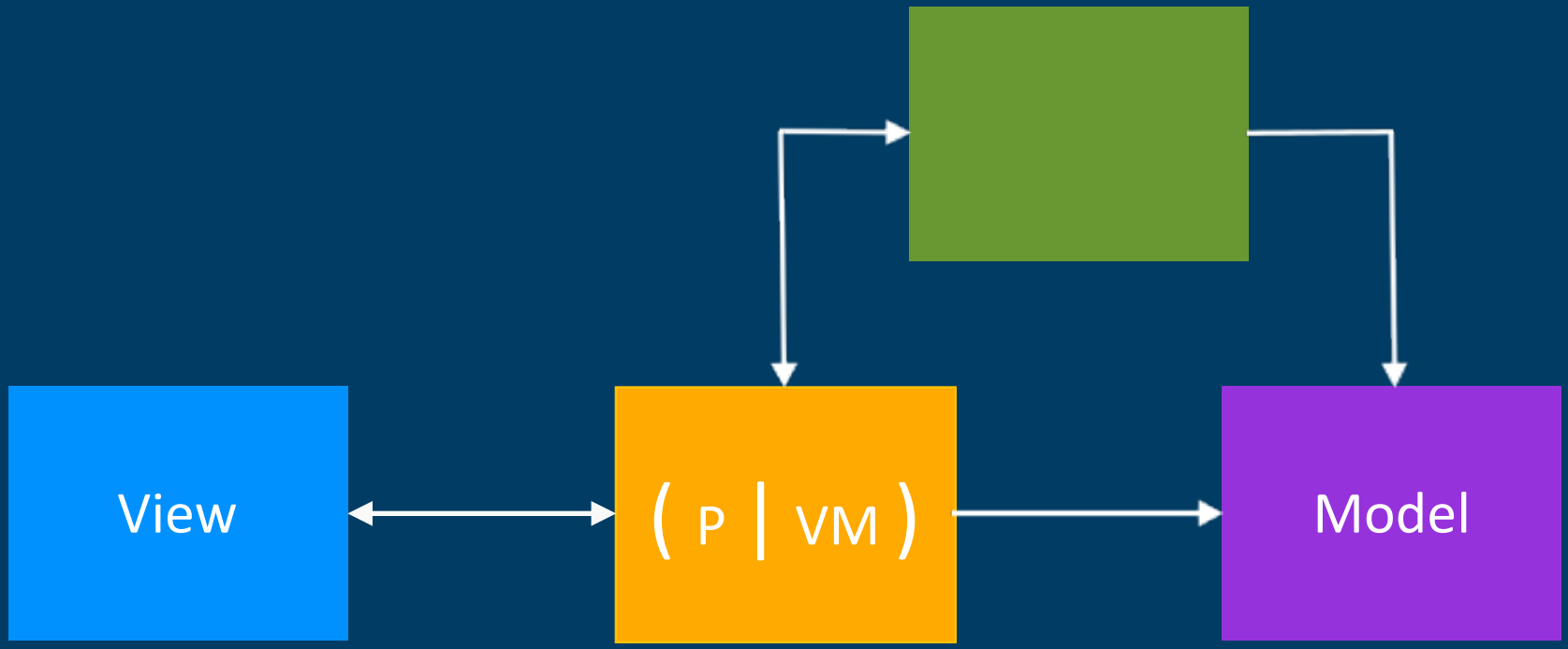
Complexity
Responsibility
Reusability

What goes where?

V ? (P | VM)

Does it contain UI logic?


View
(P | VM)
Model

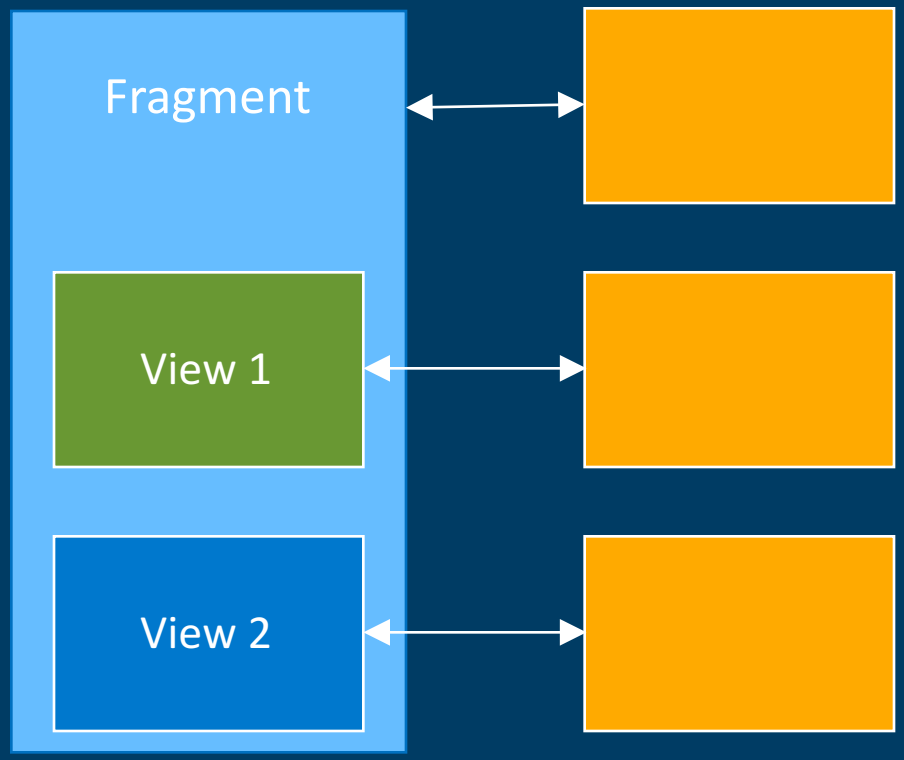


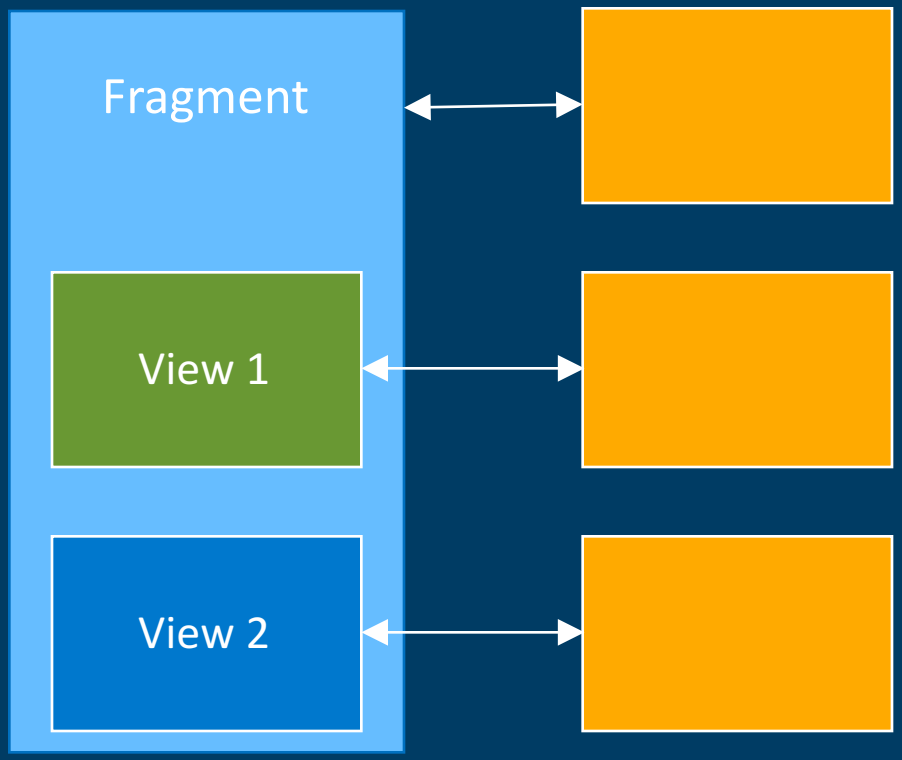
updday

@FMuntenescu

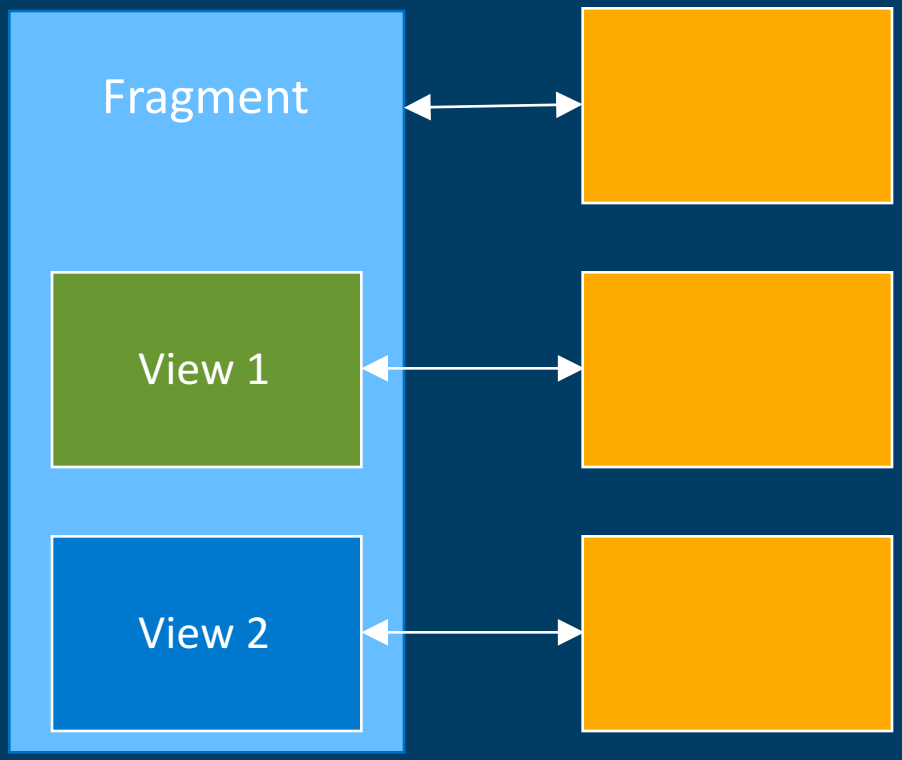
How to handle dependencies between Views?

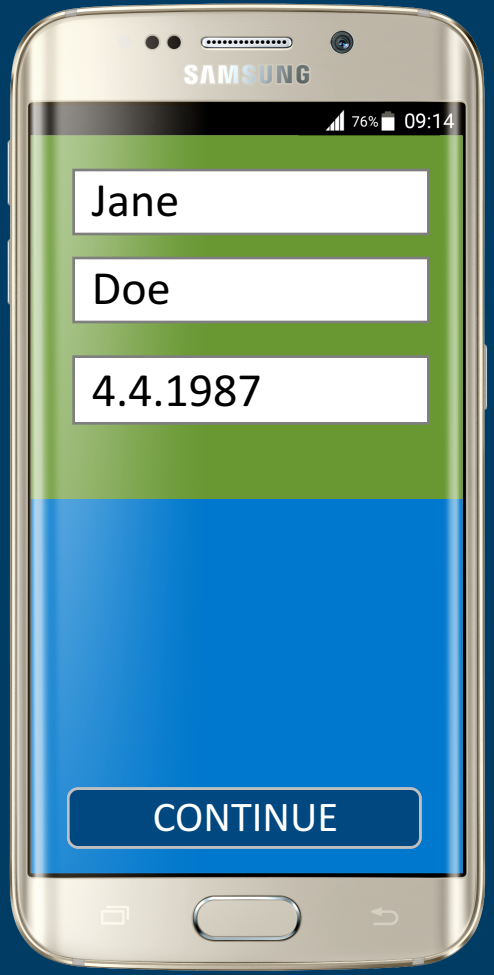
@FMuntenescu

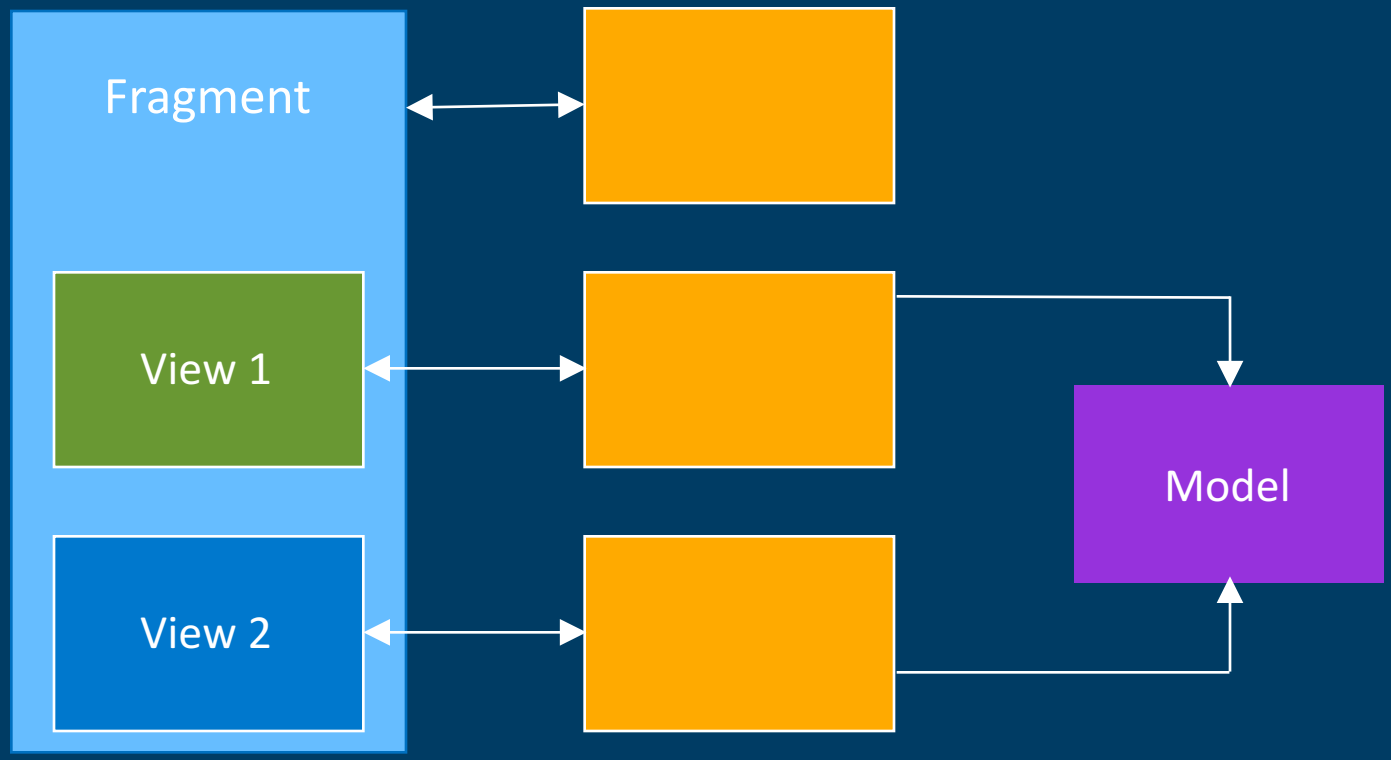

Fragment
View 1
View 2

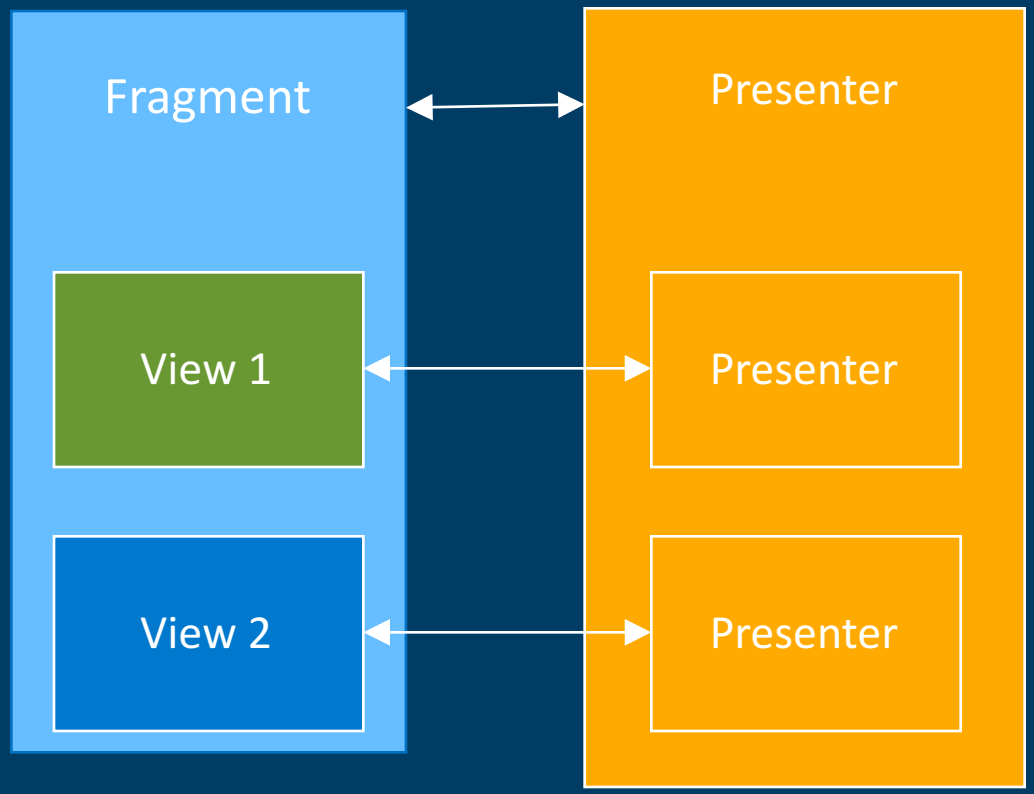

Fragment
View 1
View 2

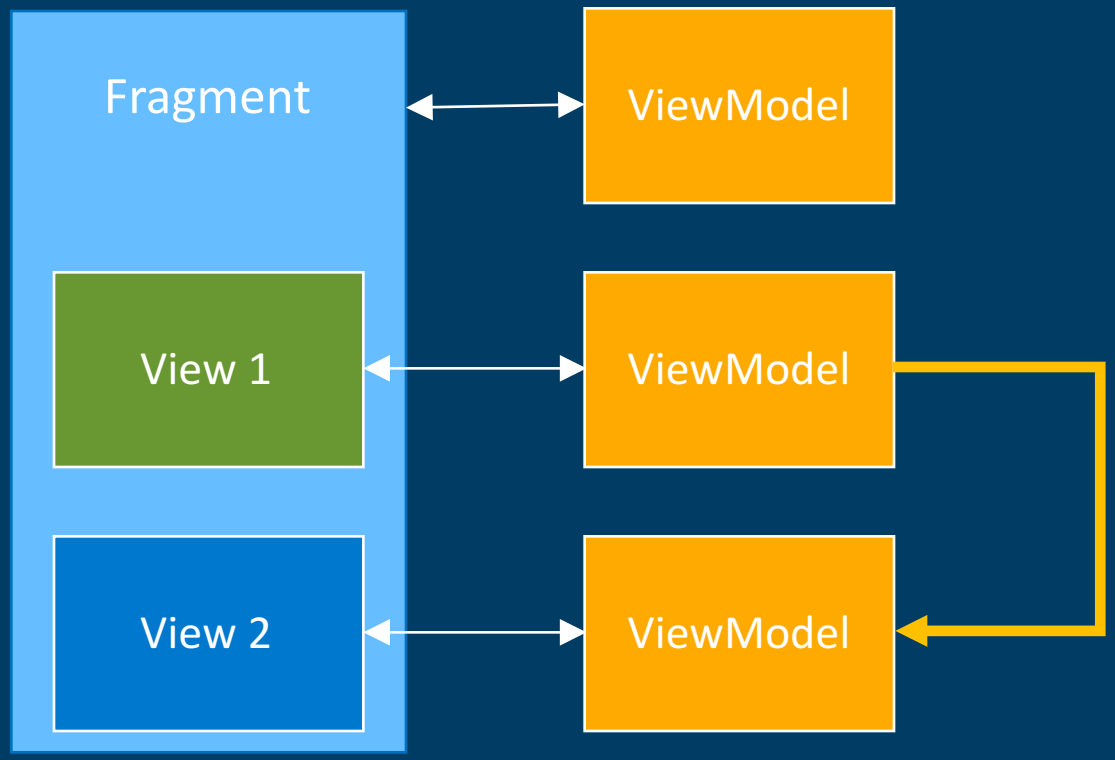

SAMSUNG
76%
09:14
View 1
View 2


Fragment
View 1
View 2


SAMSUNG
76%
09:14
Jane
Doe
4.4.1987
CONTINUE


Fragment
View 1
View 2
Model


Fragment
Presenter
View 1
Presenter
View 2
Presenter


Fragment
ViewModel
View 1
ViewModel
View 2
ViewModel

How to handle Android lifecycle?

@FMuntenescu

onResume / onPause

```
@Override
protected void onResume() {
    …
    presenter.onLoad();
}

@Override
protected void onPause() {
    presenter.onStopLoad();
    …
}
```

```
@Override
protected void onResume() {
    …
    presenter.onLoad();
}

@Override
protected void onPause() {
    presenter.onStopLoad();
    …
}
```

```
@Override
protected void onResume() {
    …
    viewModel.getDataStream()
             .subscribe(…);
}

@Override
protected void onPause() {
    // unsubscribe
    …
}
```

```
@Override
protected void onResume() {
    …
    viewModel.getDataStream()
             .subscribe(…);
}

@Override
protected void onPause() {
    // unsubscribe
    …
}
```

onAttachedToWindow / onDetachedFromWindow

```
@Override
void onAttachedToWindow() {
    presenter.onLoad();
    …
}

@Override
void onDetachedFromWindow() {
    …
    presenter.onStopLoad();
}
```

```
@Override
void onAttachedToWindow() {
    presenter.onLoad();
    …
}

@Override
void onDetachedFromWindow() {
    …
    presenter.onStopLoad();
}
```

```
@Override
void onAttachedToWindow() {
    viewModel.getDataStream()
             .subscribe(…);
    …
}

@Override
void onDetachedFromWindow() {
    …
    // unsubscribe
}
```

```
@Override
void onAttachedToWindow() {
    viewModel.getDataStream()
             .subscribe(…);
    …
}

@Override
void onDetachedFromWindow() {
    …
    // unsubscribe
}
```

onSaveInstanceState / onRestoreInstanceState

```
@Override
void onSaveInstanceState(Bundle state) {

state.putAll(
     presenter/viewModel.getState());

}

@Override
void onRestoreInstanceState(Bundle state) {

     presenter/viewModel
                    .restoreState(state);
}
```

```
@Override
void onSaveInstanceState(Bundle state) {

state.putAll(
     presenter/viewModel.getState());

}

@Override
void onRestoreInstanceState(Bundle state) {

     presenter/viewModel
                    .restoreState(state);
}
```

```
@Override
Parcelable onSaveInstanceState() {
     // save state
}

@Override
void onRestoreInstanceState(
                        Parcelable state) {
     // restore state
}
```

MVC vs MVP vs MVVM

@FMuntenescu

Are your Android classes logic free?

@FMuntenescu

Can you unit test everything?

Do your classes do one thing and one thing only?

Robust, stable, testable, modular, easy to extend

A Journey Through MV Wonderland

jobs@upday.com

https://upday.github.io/

Google Architecture Blueprints
https://github.com/googlesamples/android-architecture

MVP vs MVVM example
https://github.com/florina-muntenescu/MVPvsMVVM

upday

@FMuntenescu