UNIVERSITÀ DEGLI STUDI DI PALERMO

dipartimento di ingegneria unipa

Cenni di progettazione software per Web Application

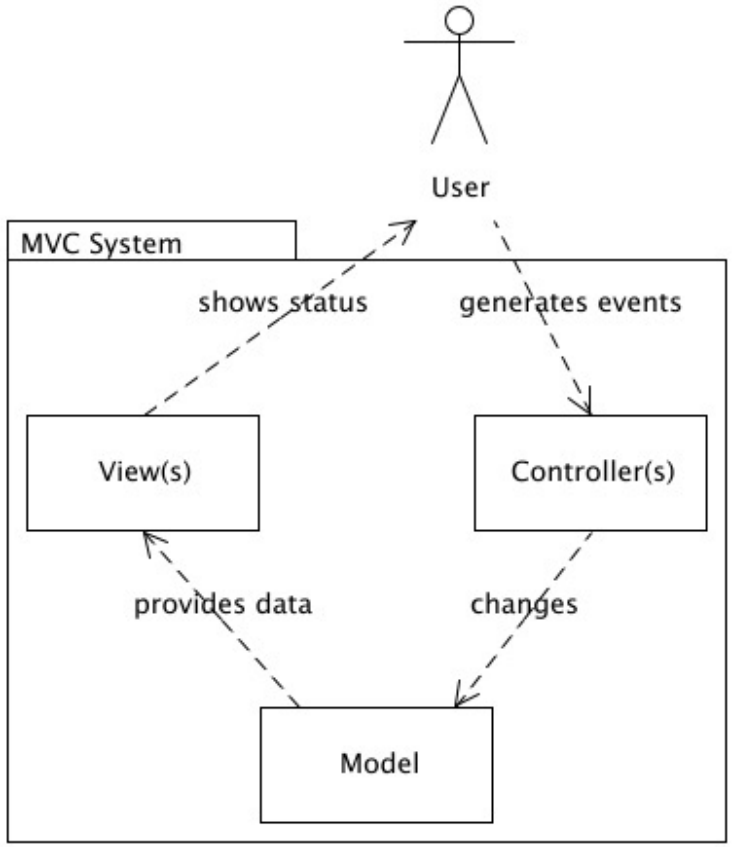
Corso di Programmazione Web e Mobile

a.a. 2021/2022

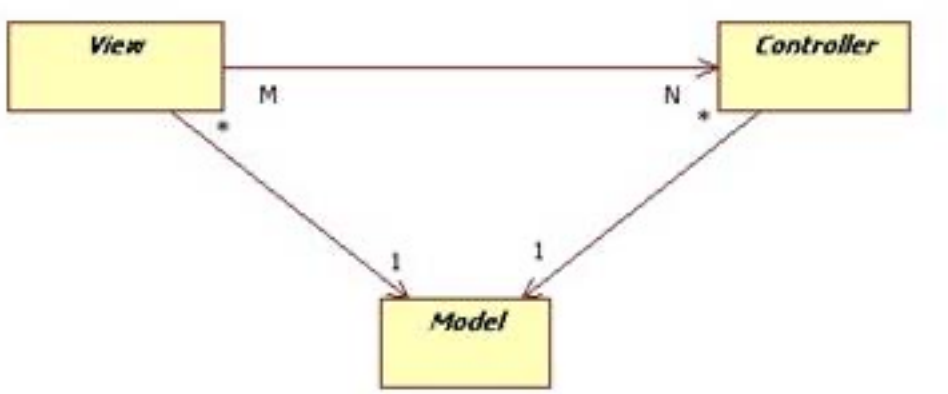
Prof. Roberto Pirrone

LABORATORIO DI INTERAZIONE UOMO-MACCHINA
CHILAB

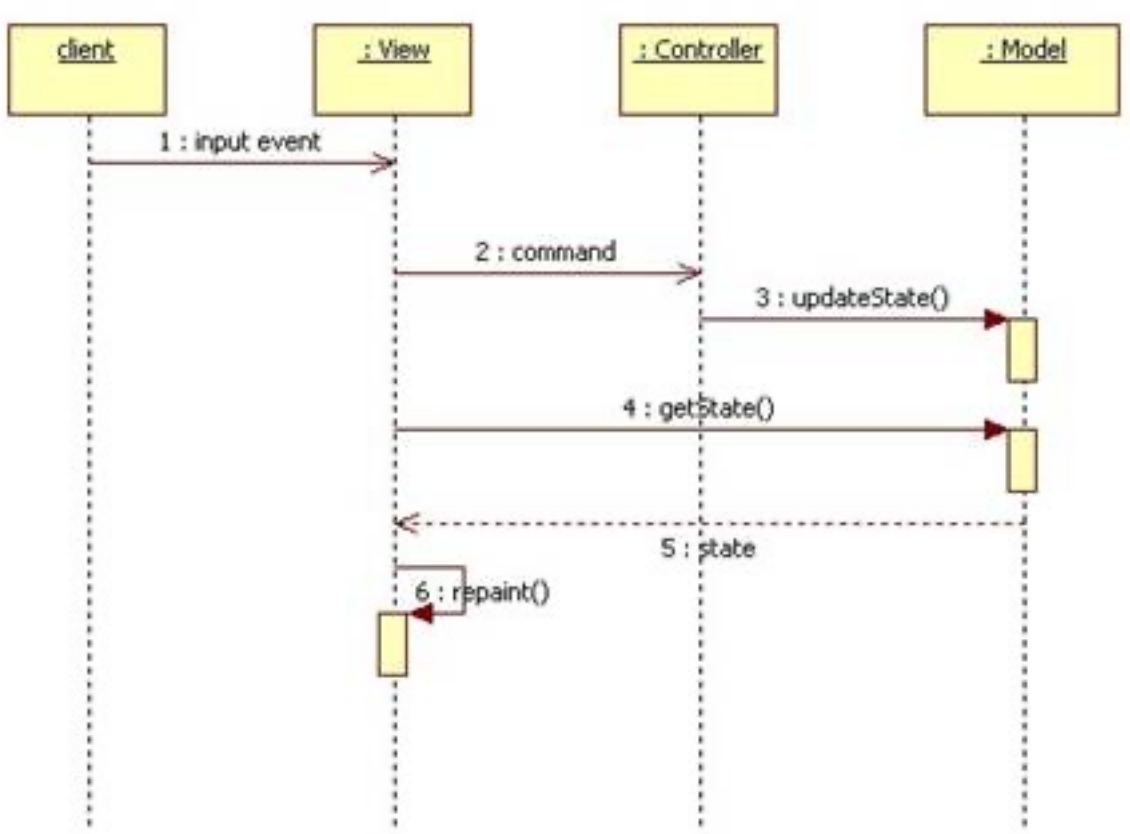
Pattern MVC


User
MVC System
shows status
generates events
View(s)
Controller(s)
provides data
changes
Model

Pattern MVC



Pattern MVC


client
: View
: Controller
: Model
1 : input event
2 : command
3 : updateState()
4 : getState()
5 : state
6 : repaint()

Pattern MVC

- Non molto adatto al modello three-tier delle applicazioni web che sono incentrate sugli eventi all'interfaccia utente che fanno da trigger per tutti i casi d'uso

- Richiede profonda capacità di analisi perché è pensato per il completo disaccoppiamento tra Model, View e Controller

    - Non ben adatto per applicazioni piccole

Pattern MVP – Model, View, Presenter

- Variante di MVC

- Model: contiene i dati dell'applicazione, *recuperati dalla rete o da un db* e poi organizzati in semplici classi che possano essere usate dagli altri componenti

Pattern MVP – Model, View, Presenter

- Variante di MVC

- View: Contiene l'interfaccia utente e nessun componente di logica di business né tantomeno informazioni sui dati. Implementa un'interfaccia usata dal Presenter.

Pattern MVP – Model, View, Presenter

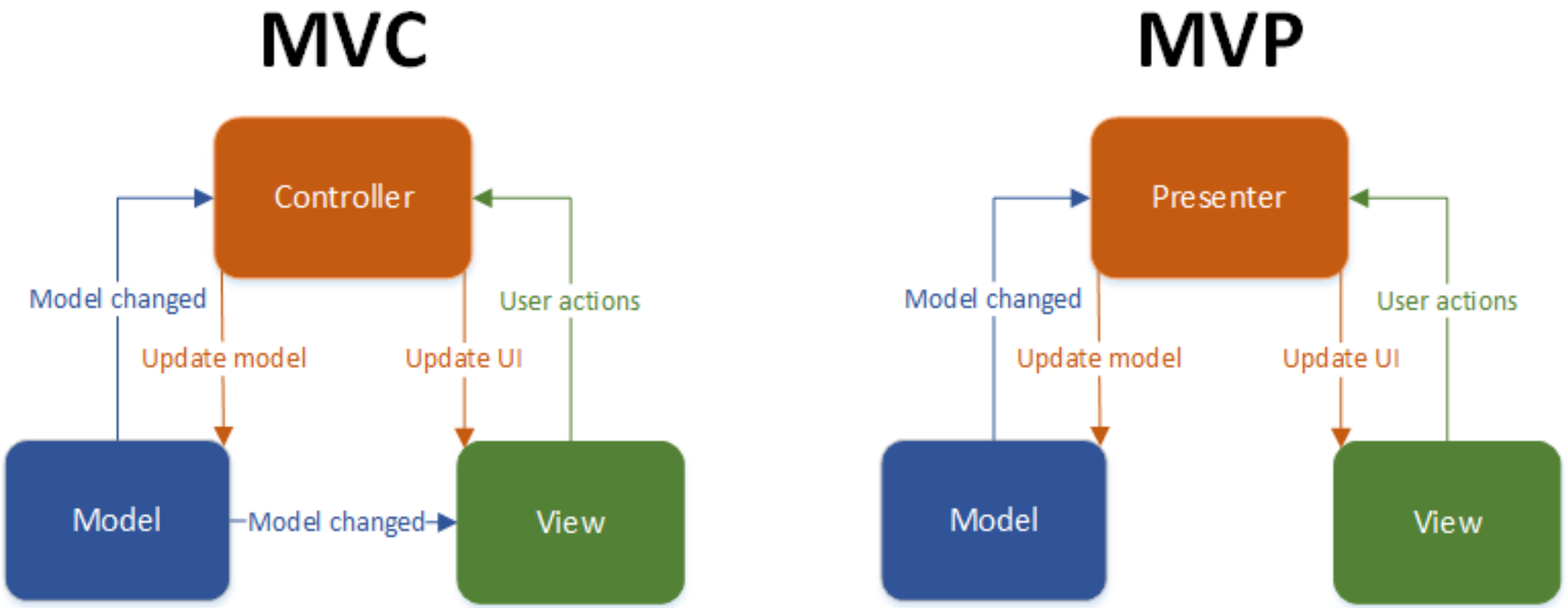
- Variante di MVC

- Presenter: contiene la logica di business, resta in attesa delle azioni dell'utente attraverso la View e aggiorna sia il Model sia la View. Recupera i ati dal Model e li formatta per la View. Possiede dei metodi di intefaccia (ad es. showProgressBar, updateData, …) che gli consentono di manipolare i dati e generare azioni conseguenti per manipolare la View.

  - Una buona prassi di sviluppo del Presenter è quella di *farlo dipendere il meno possibile dall'architettura di SO sottostante* e quindi evitare di usare il core SDK per quanto possibile.

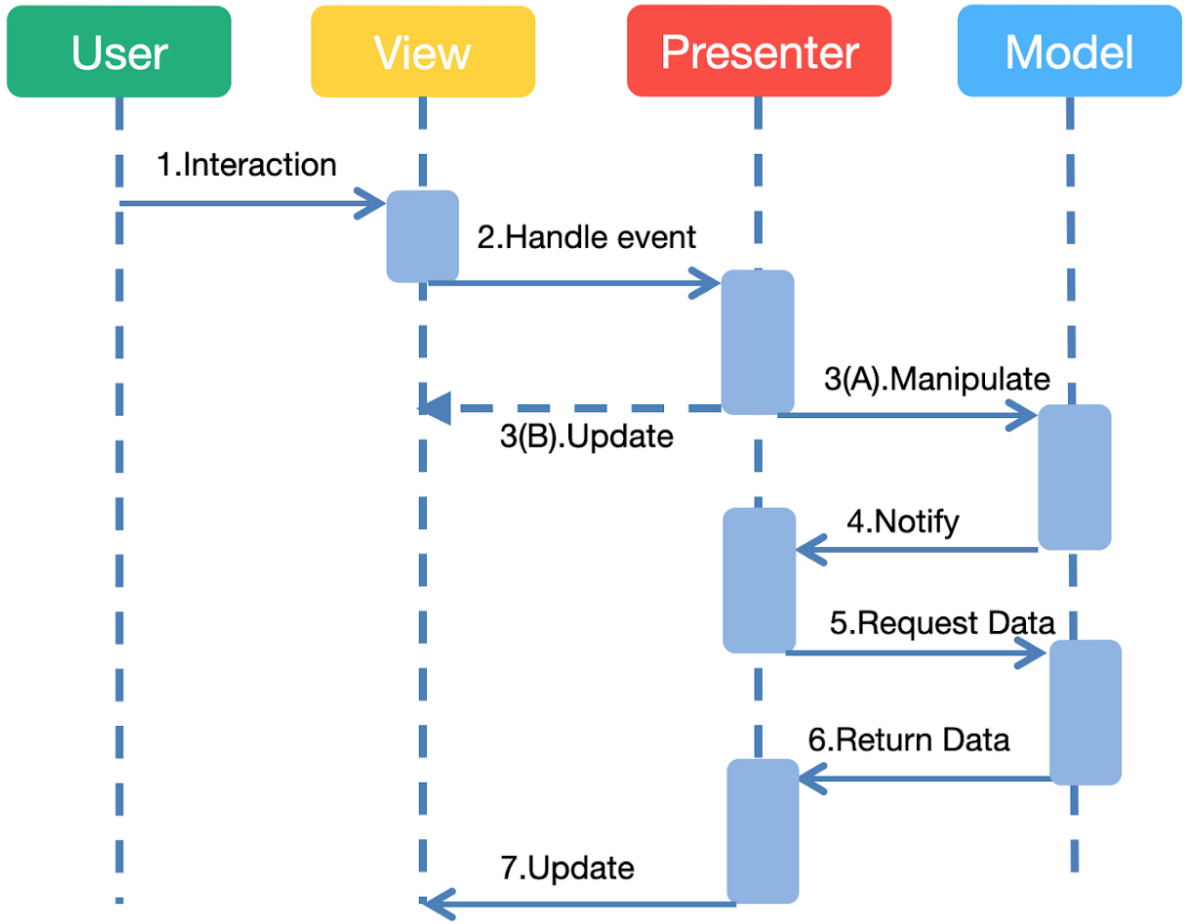
  - Il Presenter cerca di separare la View quanto più possibile dagli altri componenti *per consentirne la modifica con il minimo impatto sulla logica sottostante*.

Pattern MVP – Model, View, Presenter



Fonte: https://bit.ly/3lQqdbD

Pattern MVP – Model, View, Presenter



Fonte: https://bit.ly/3wLidXH

Pattern MVP – Model, View, Presenter

- Ricalca il modello Three-tier
- Può essere semplicemente implementato attraverso il metodo `res.render(view, [locals], callback)` in Express.js
- La View diventa *l'intera applicazione di front-end* nell'architettura MERN
  - Non c'è più un rendering esplicito nel server Node/Express in questo caso

Pattern MVVM – Model, View, ViewModel

- Orientato al *binding dinamico dei dati* dell'interfaccia
- Tipico delle implementazioni mobili
- Ad alto livello, definisce il funzionamento dei framework di front-end come Angular o React
  - Ognuno di questi possiede implementazioni particolari del pattern

Pattern MVVM – Model, View, ViewModel

- Model: contiene i dati dell'applicazione, *recuperati dalla rete o da un db* e poi organizzati in semplici classi che possano essere usate dagli altri componenti. Espone i dati all'applicazione usando l'*observer design pattern*. In tal modo si disaccoppia dalla View e dal ViewModel.

  - Observer design pattern: un oggetto mantiene una lista di *osservatori* che vengono notificati immediatamente di ogni suo cambiamento

Pattern MVVM – Model, View, ViewModel

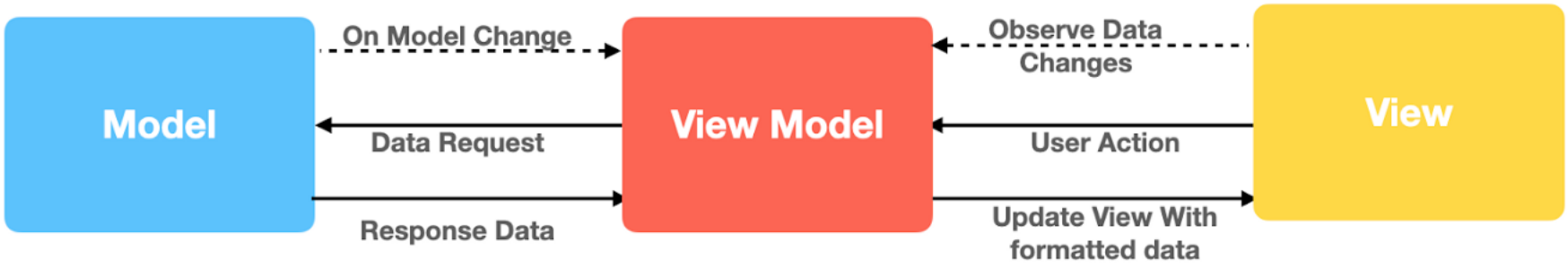
- View: Simile al caso MVC e MVP. Aggiorna l'interfaccia utente ogni volta che il ViewModel viene modificato. Qui possono essere utilizzati approcci differenti: un possibile esempi è sempre l'observer design pattern. La View, passa tutte le azioni dell'utente al ViewModel.

Pattern MVVM – Model, View, ViewModel

- ViewModel: Sostituisce il Presenter. E' responsabile per le funzioni ed i metodi di presentazione per manipolare lo stato della View. Richiede i corretti metodi del Model per servire le richieste mirate ad aggiornare lo stato della View.

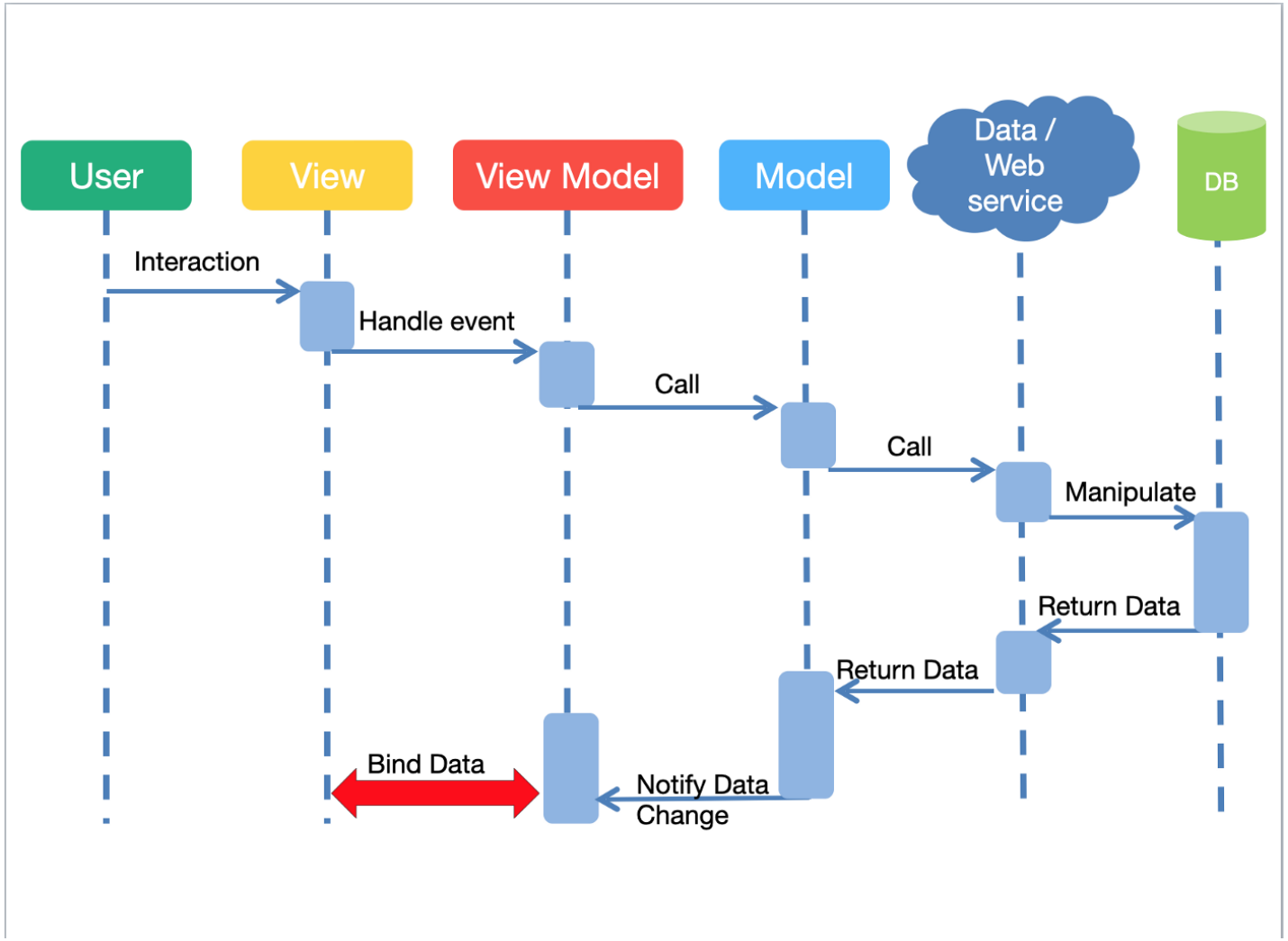
  - Non dovrebbe avere espliciti riferimenti alle istanze degli oggetti della View, ma solamente manipolare le informazioni che devono essere da questa mostrate
  - Mantiene i dati separatamente da possibili cambiamenti di stato della View, per es. modifica e/o rimozione di componenti dell'interfaccia utente

Pattern MVP – Model, View, Presenter



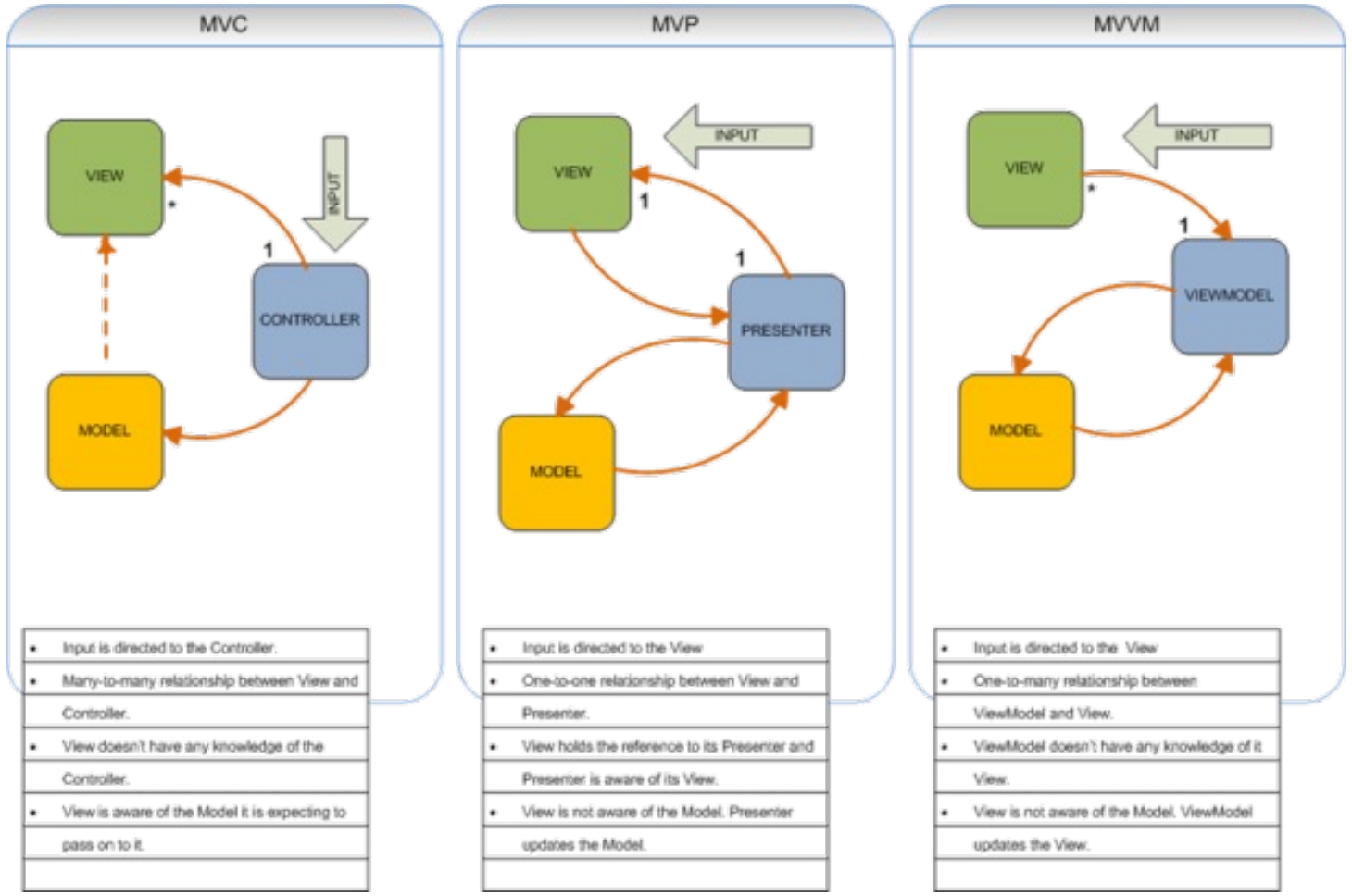
Fonte: https://bit.ly/3wLidXH

Pattern MVP – Model, View, Presenter



Fonte: https://bit.ly/3wLidXH

Pattern a confronto



Fonte: https://bit.ly/3MFg2vh

Navigazione dell'interfaccia web

- Un possibile approccio al disegno della View e, conseguentemente dell'intera applicazione è quello di descrivere i casi d'uso dell'applicazione dal punto di vista dell'utente usando degli Activity Diagram al fine di creare dei veri e propri *diagrammi di navigazione*
  - Nelle attività si potranno descrivere esplicitamente le tipologie di attività da compiere: browsing o processing
  - Le attività di processing sono quelle che richiedono *esplicito processo da parte del back-end*

Navigazione dell'interfaccia web

- Il progetto del database fornirà il modello generale dei dati
- Il diagramma di navigazione vero e proprio sarà un diagramma delle classi di front-end in cui sono esplicitamente stereotipati i tipi di link tra le pagine
- La decomposizione in classi di questi diagrammi e gli stereotipi dei link rifletteranno il modello dei dati in relazione a ciò che deve essere visualizzato e manipolato nei diversi casi d'uso

Navigazione dell'interfaccia web

- Le attività di processing saranno di fatto le richieste al server e ne illustreranno quindi i casi d'uso generali

- Il server potrà essere progettato usando il pattern MVP in cui la View non è altro che una classe boundary verso l'applicazione di front-end

    - Il nostro server essenzialmente copre i layer della logica di business e quello della logica di accesso ai dati