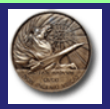

Module 15

# From Stand-alone/Lap-top to Web Applications: Structuring Concepts





G. Cantone, Alessandro D'Angiò, A. Lomartire, G. Pesce, Stefano Scarrone

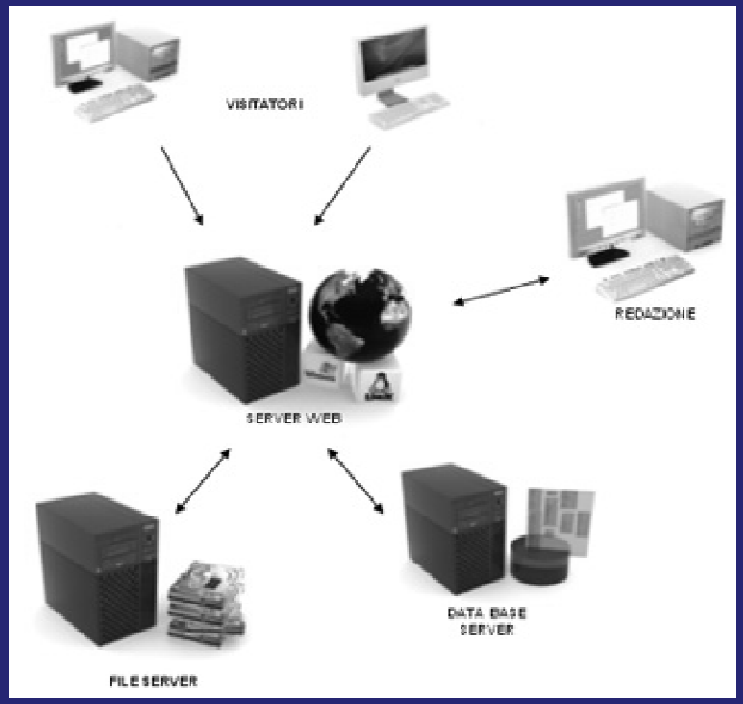
# Sommario

- Stand-alone, Web Application e loro rispettive caratterisriche
- Boundary Control Entity (BCE)
  - Model View Controller (MVC) o Supervising Presenter (SP)
  - View Presentation Model (VPM)
    - Definizione
    - Descrizione componenti
- Architetture BCE per una Stand-alone e una Web Application, rispettivamente
  - Definizione
  - Architettura
  - Descrizione componenti

# What is a Stand-alone | Lap-top Application?

Sa-A è una qualsiasi applicazione software la quale può stare su un unico calcolatore elettronico, anche di limitate dimensioni.

# What is a Web Application?




WA è una qualsiasi applicazione software a cui è possibile accedere attraverso una rete - internet o intranet - a mezzo di un *web browser*.

# Web Application. Motivazioni

Possibilità di aggiornare e mantenere le WA senza distribuire e installare il software su potenzialmente migliaia di macchine.

## Storia

*Origini (Stand alone) -* Ogni modifica software si rifletteva su migliaia di aggiornamenti presso i client … o spediti ai clienti.

*Prima (Client-Server).* Ogni applicazione server aveva il suo programma client (interfaccia remota). Ogni modifica sul server si rifletteva su migliaia di aggiornamenti sui client.

*Dopo.* Basandosi su standard aperti, ogni client può possedere la sua chiave di accesso al programma software (web browser), capace di comunicare con il programma server (web server) dove risiede ed è interpretata l'applicazione.

# Web Application. Storia

6 agosto 1991

Tim Bernes-Lee pubblica il primo sito nella rete

1995

Sun lancia Java (SDK 1.0) sul mercato

1995

Rasmus Lerdorf presenta PHP alla comunità

2002

Microsoft lancia .NET (SDK 1.0) sul mercato

# Web Application. Web Browser, standard e trasferimento informazioni

Un *web browser* è un cliente ***leggero*** (“*thin*”) ; esso può essere installato praticamente su qualsiasi materiale hi-tech (dai frigoriferi ai palmari).

Le applicazioni web si basano su standard ‘*open*’ , fra cui:

HTML: HyperText Markup Language
XML: eXtensible Markup Language
HTTP: Hypertext Transfer Protocol

Il cliente si connette al server. Ogni documento è spedito dal server al client come un ***documento statico***, la sequenza delle pagine e la possibilità di inserire input in ognuna di esse dona un *esperienza dinamica e interattiva*.

# Web Application. **Standard**

A tutt'oggi esistono moltissimi standard, linguaggi, *framework* e piattaforme per la realizzazione di applicazioni web.

In generale le tecnologie web si possono suddividere in Lato Server e Lato Client:

*Lato Client*

Linguaggi integrati nei web browser (**DHTML**, **AJAX**, **JavaScript**), interpretati: minore espressività, sicurezza e scalabilità rispetto a →

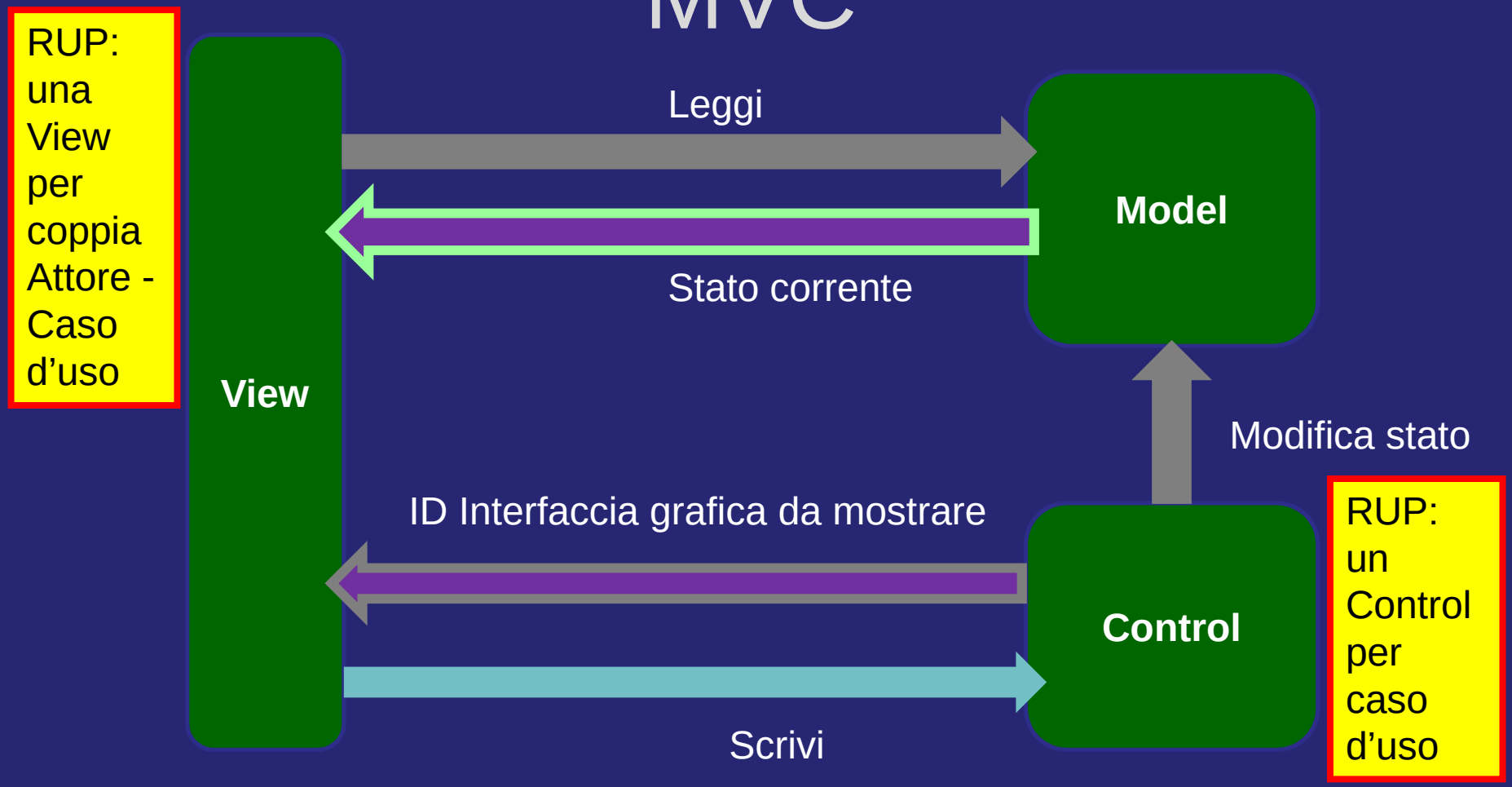
*Lato Server*

Linguaggi integrati nei web server (**ASP, NET**, **PHP**, **JSP**), interpretati o compilati: maggiore potenza espressiva, sicurezza e scalabilità.

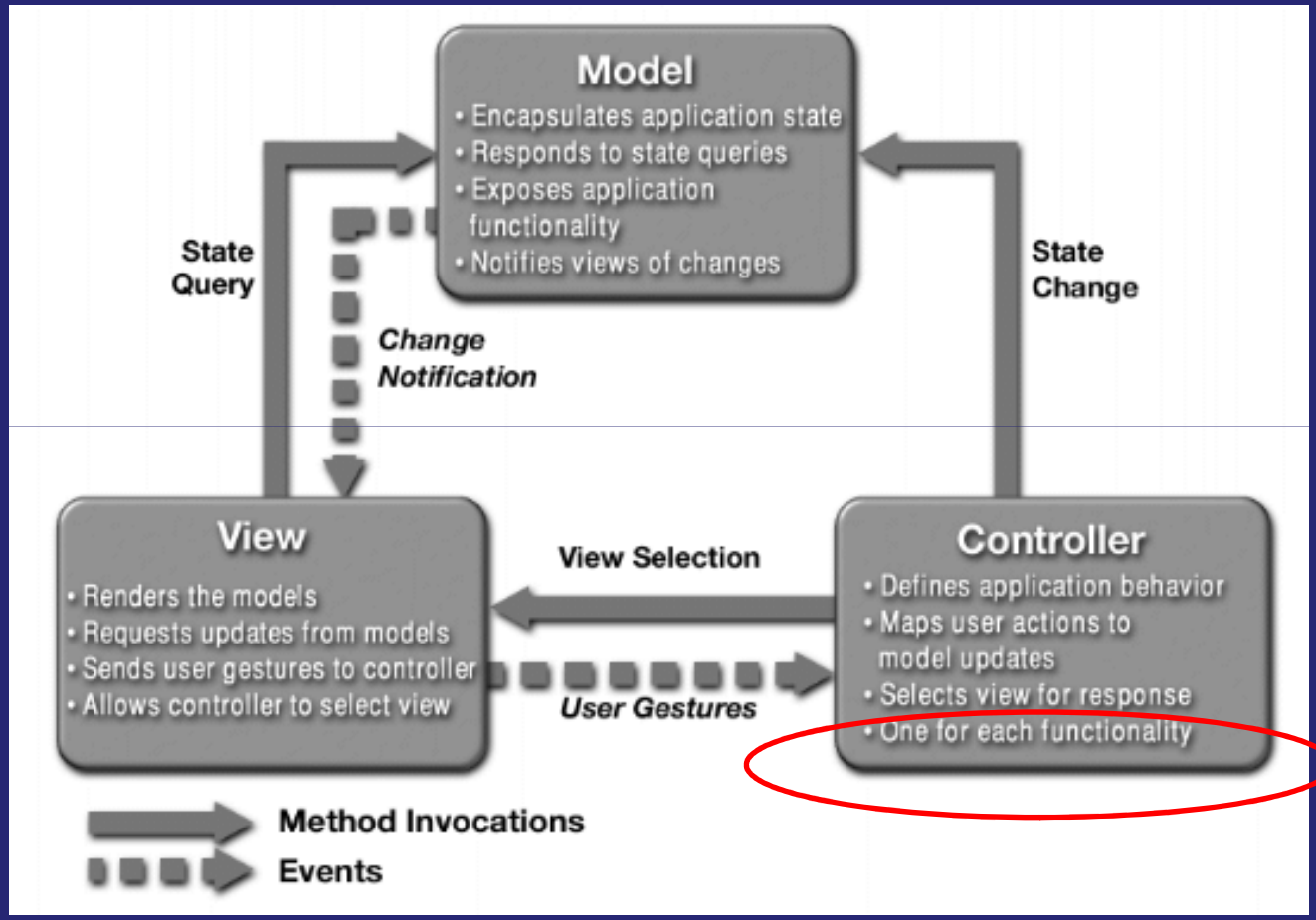
# Model View Controller Pattern

- Possibilità di accedere alla gestione dei dati con diverse tipologie di GUI (magari sviluppate con tecnologie diverse)
- I dati dell'applicazione possono essere aggiornati tramite diverse interazioni da parte dei client (messaggi SOAP, richieste HTTP...)
- Il supporto di varie GUI e tipologie di interazione non dipende dalle, e non influisce sulle, funzionalità di base dell'applicazione.
- Strutturazione di software e documentazione in *tre strati:* Model, View, Controller.

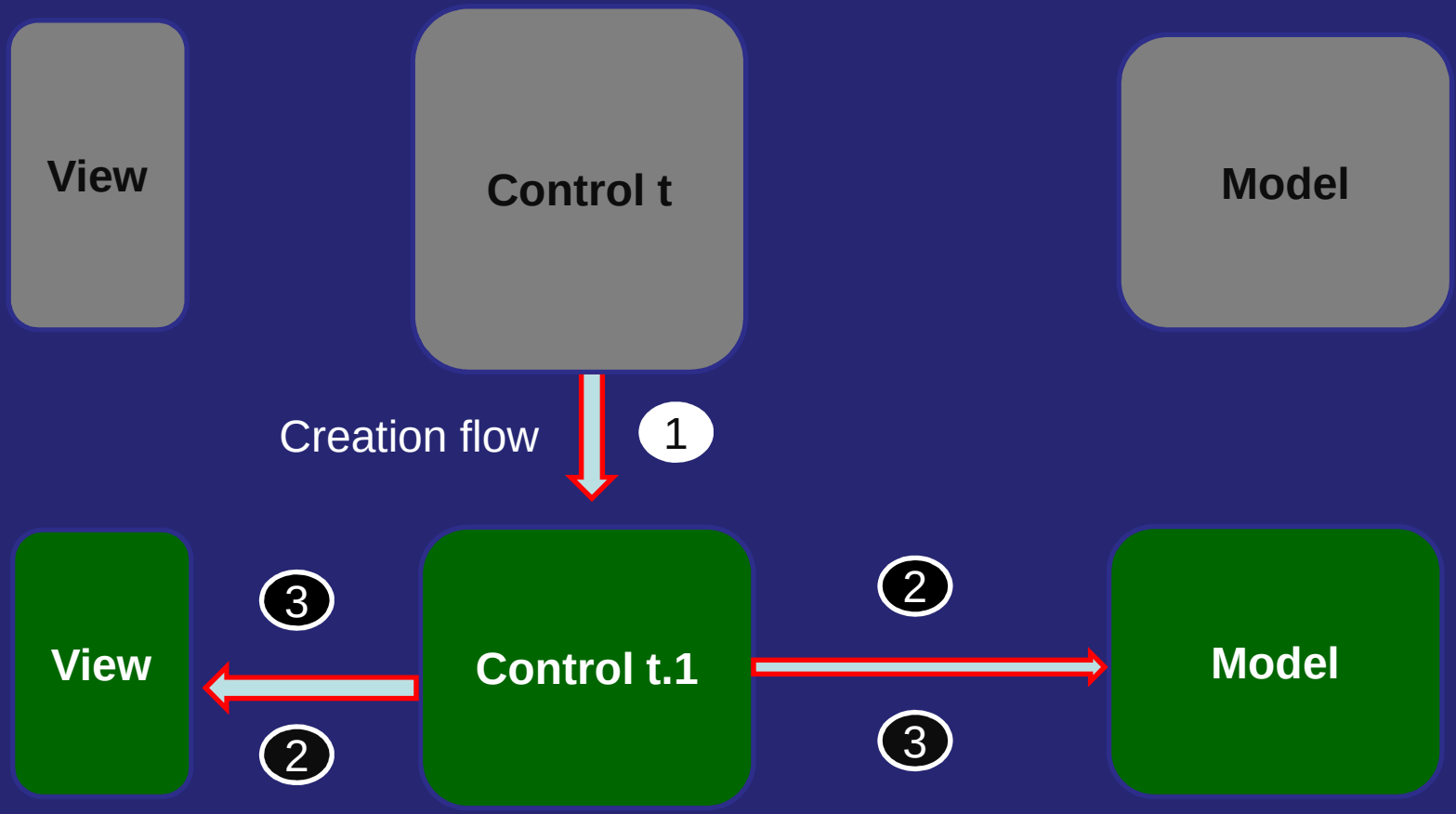
# Model View Control Pattern, MVC



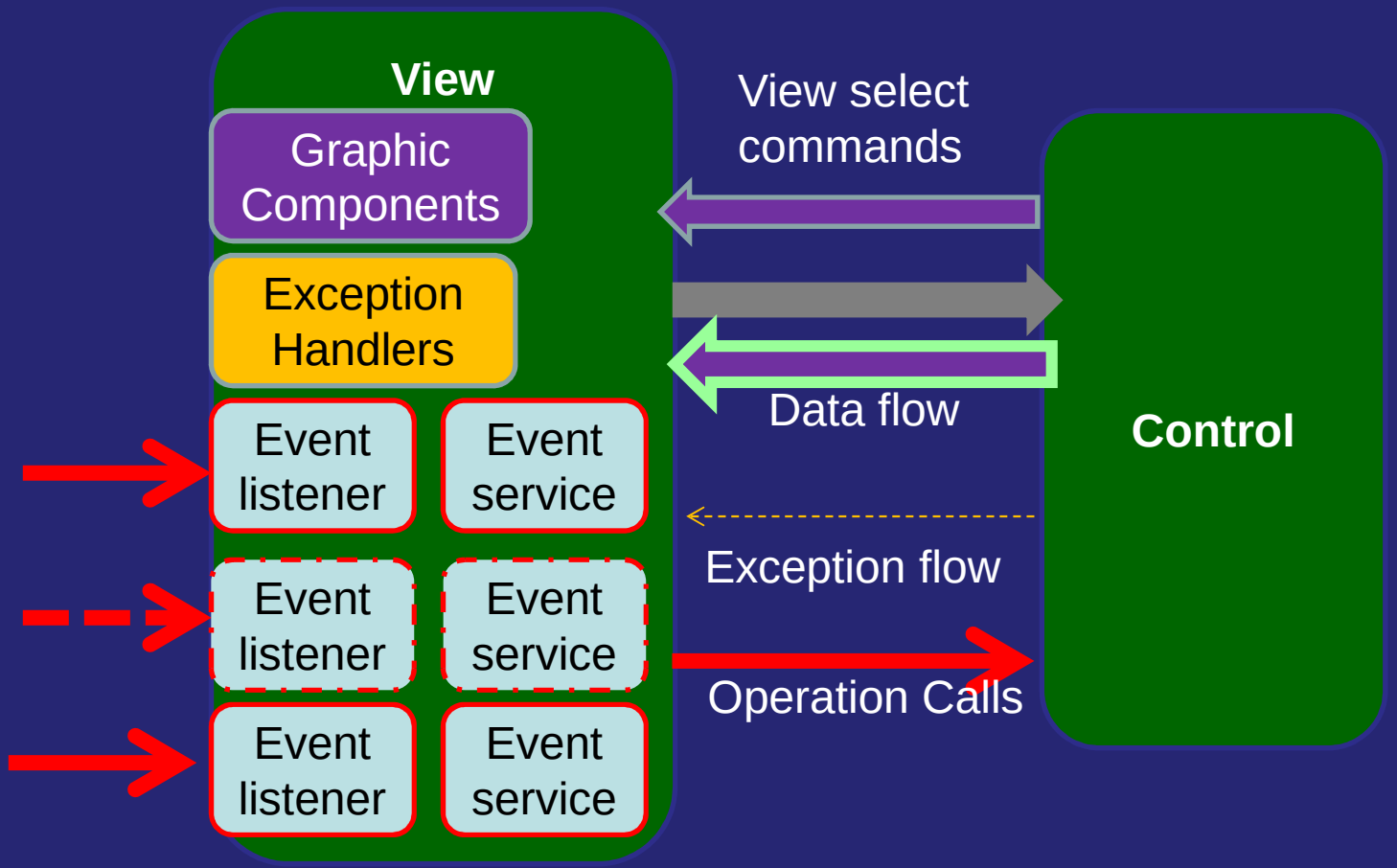
# MVC by Microsoft



# Model View Control. Lap-top/Stand-alone applications: Creation flow



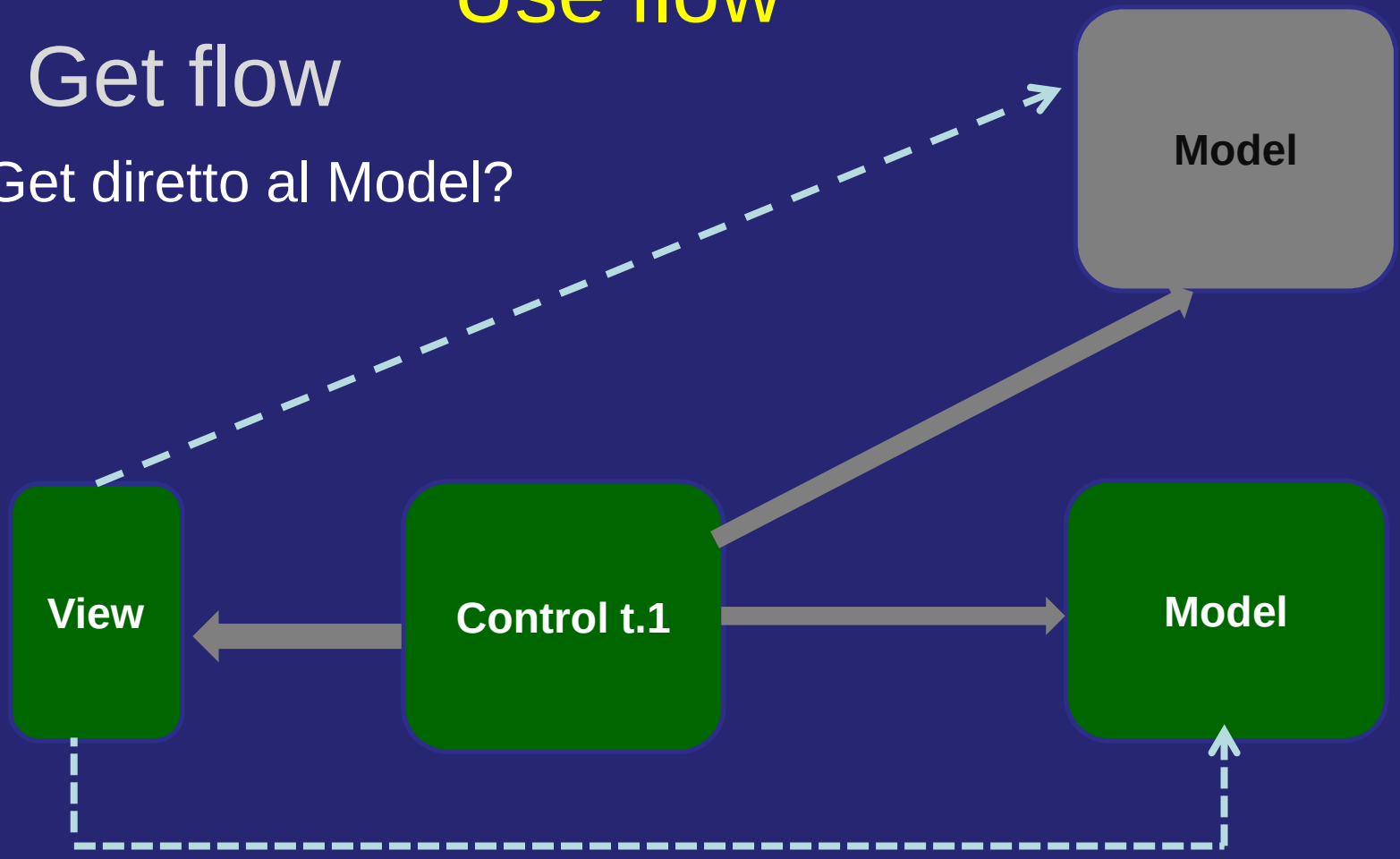
# Lap-top/Stand-alone applications: User events flow



# Lap-top/Stand-alone applications: Use flow

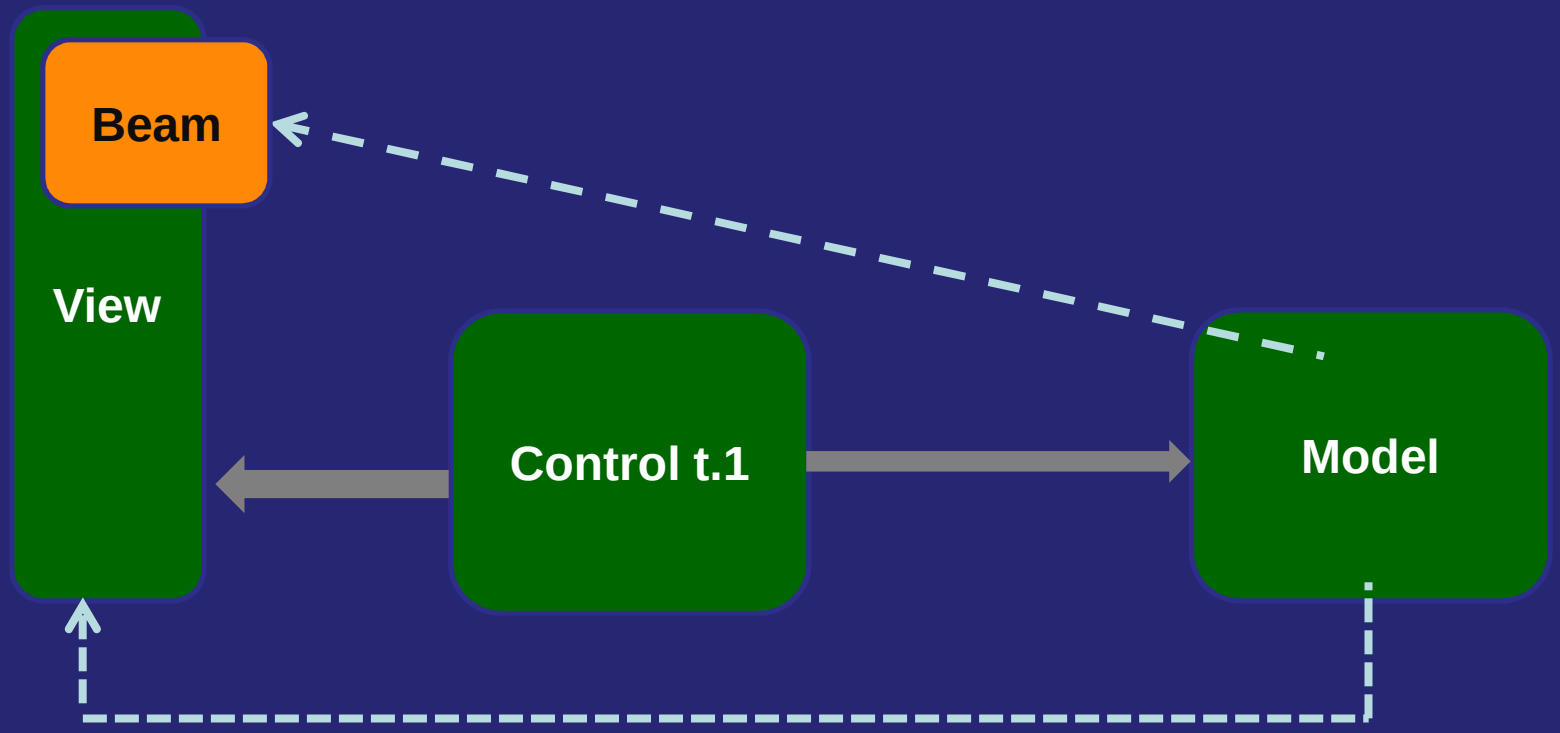
## Get flow

Get diretto al Model?



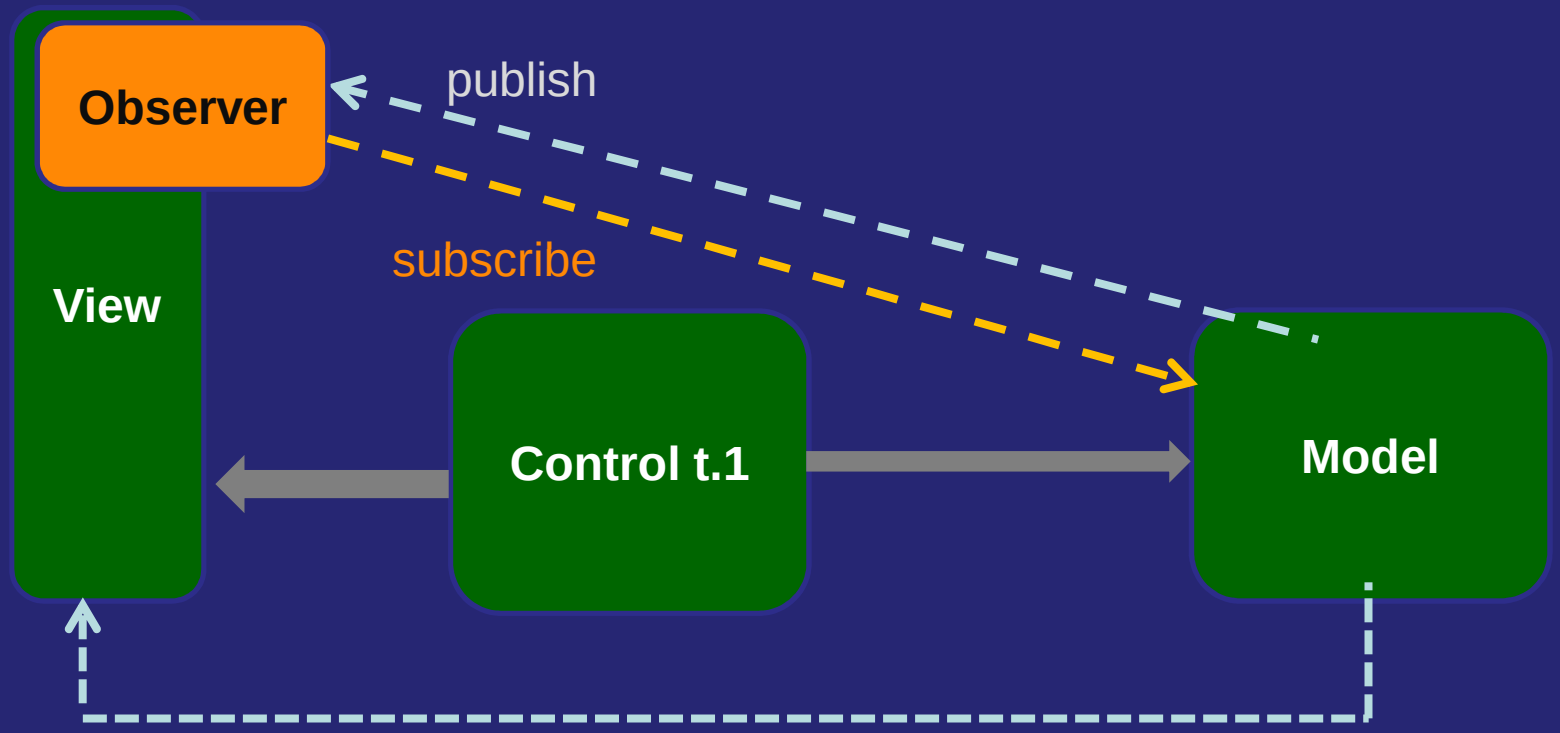
# Lap-top/Stand-alone applications: Use and Notification flow

Notificazione di cambio di stato?

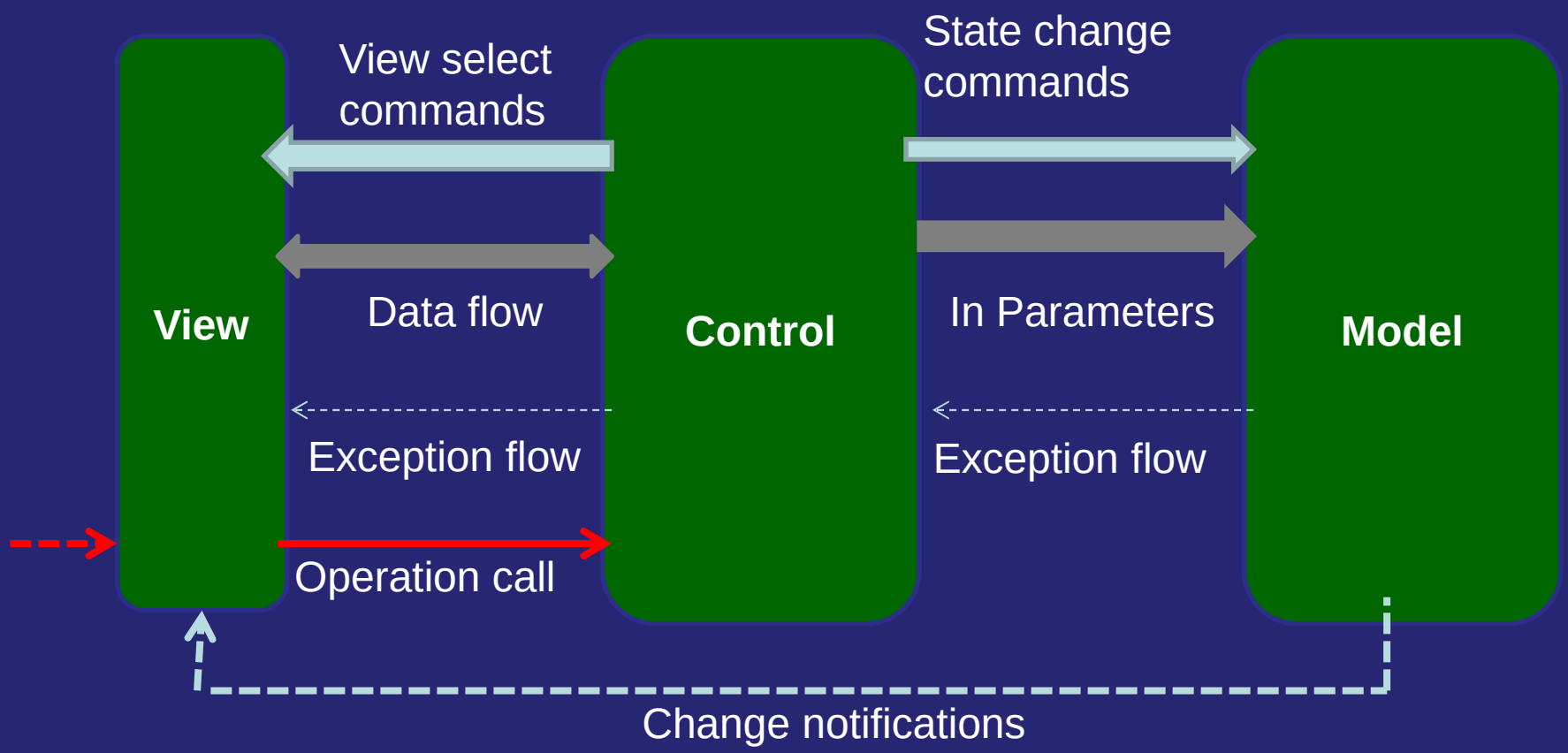


# Lap-top/Stand-alone applications: Use and Notification flow

Notificazione di cambio di stato?




# Model View Control



# Descrizione dei componenti

## View

- Gestisce sia la logica di presentazione dei dati e la costruzione dell' interfaccia grafica (GUI), sia i modi di acquisire i dati dell'applicazione.

  I dati da presentare possono essere acquisiti *in sola lettura accedendo* direttamente dal *Model* (attributi).

- Per far sì che i dati presentati siano sempre aggiornati è possibile adottare due strategie note come *push model* e *pull model*.

  - ***push model***
    Adotta il pattern Observer: le View possono registrarsi come osservatori di Model. onde ricevere gli aggiornamenti di Model in tempo reale.

  - ***pull model***
    Utilizzato nel caso in cui la View intende richiedere gli aggiornamenti quando "*lo ritiene opportuno*".

# Model View Controller

## Descrizione dei componenti

**Controller**

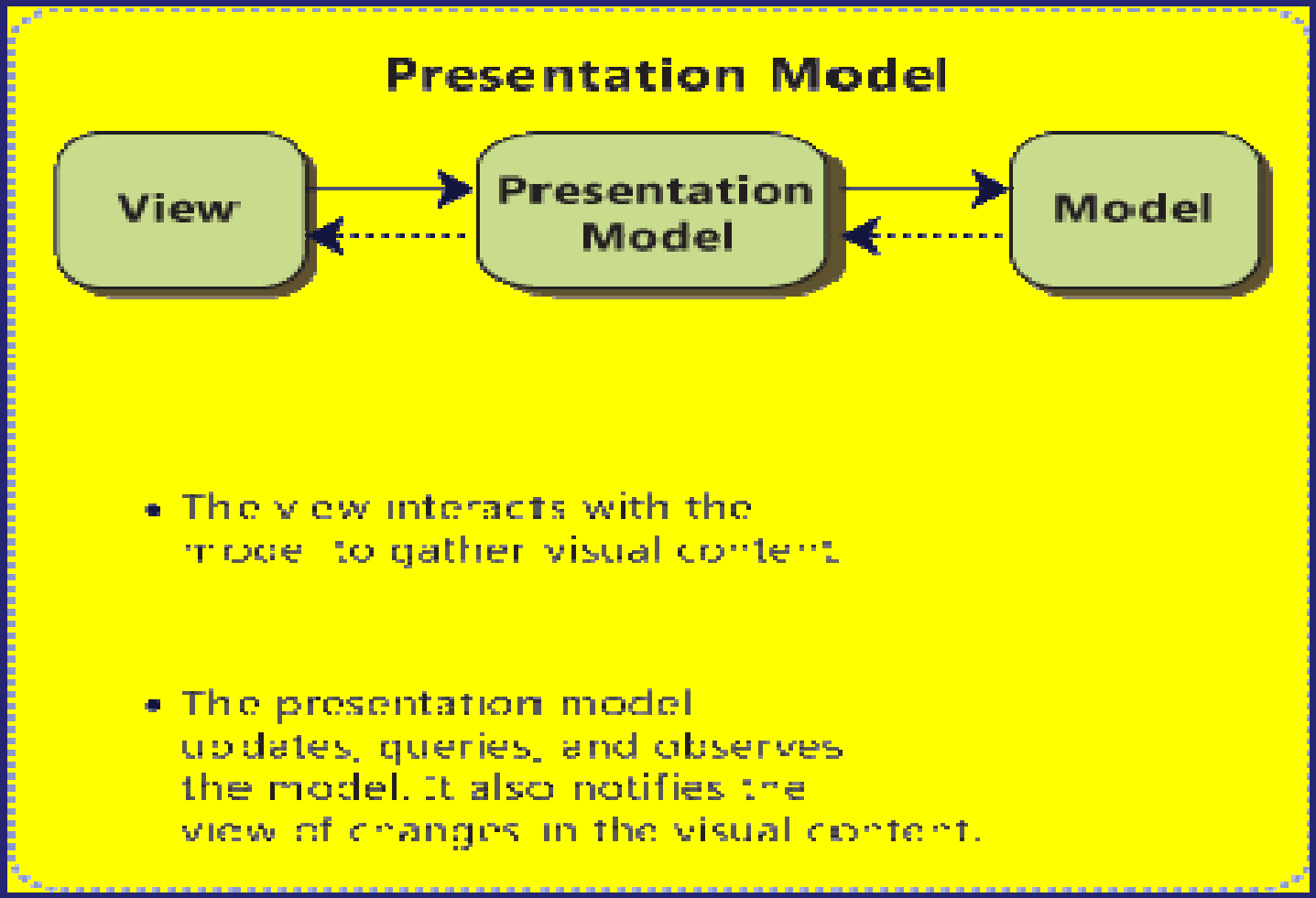
- Non svolge il solo ruolo di “passacarte” tra View e Model.
- Ha la responsabilità di trasformare una azione dell'utente sulla View in una o più azioni eseguite su (una o più istanze del) Model.
  - Realizza la mappatura tra input dell'utente e processi eseguiti dal Model e seleziona le schermate della View richieste.
  - Implementa la logica di controllo dell'applicazione.

# Model View Controller
## Descrizione dei componenti

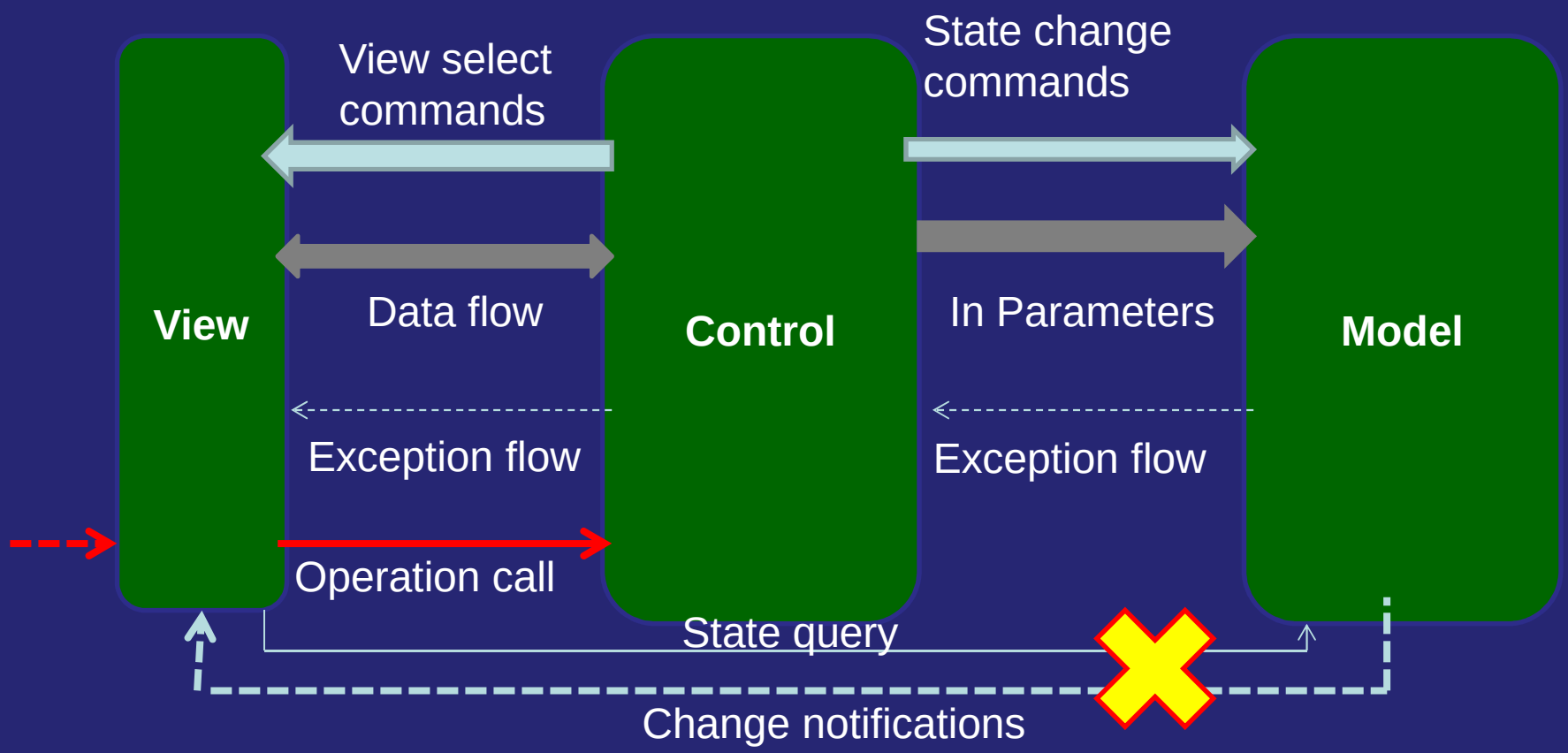
**Model**

- Definisce la rete di entità partecipanti al business e, per ogni entità, realizza le relazioni e i comportamenti che espone in forma di operazioni.
- Incapsula lo stato delle entità dell'applicazione, ne implementa gli attributi e le operazioni che possono essere eseguite su questi
- Un buon modello di entità non dovrebbe esporre lo stato.
- Anche si esclude che ogni controllore possa estrarre dai comportamenti del Model o conservare in proprio tutti i dati di I/O di interesse,
  - Il controllore estrarrebbe tutti i dati di I/O possibili invocando i comportamenti;
  - il Model esporrebbe metodi ClasseAttributo getAttributo() per i restanti attributi destinati ad essere direttamente presentati all'esterno dalla View.
- Responsabilità di notificare ai componenti della View eventuali aggiornamenti verificatisi in seguito a richieste del Controller.

# View Presentation Model



# From Model View Control to View Presentation Model



# View Presentation Model

**View**

- Neanche i dati da presentare possono essere acquisiti direttamente dal Model. La View non deve allora utilizzare metodi del Model del tipo Attributo getAttributo(), ammesso che ve ne siano.
- Per il resto tutto è pari alla View di MCV

**Presentation Model**

- Come Control di MVC.

**Model**

- Come per MVC

# STAND ALONE – MVC

Tutti i tre strati risiedono sulla macchina utente, compreso lo strato View. Sarebbe rischioso implementare MVC in senso stretto, perché ogni utente avrebbe la vista dei riferimenti agli oggetti del Model piuttosto che al solo Controller. Si potrebbe pensare allora a delle *classi locali di scorta* (bean) in cui sistematicamente replicare i dati di output e appoggiare quelli di input del particolare caso d'uso. Tali classi sarebbero accessibili alla View e al Controller del particolare caso d'uso.

Una Stand-alone basata sul un pattern BCE potrebbe allora essere composta dalle seguenti tipologie di componenti:

- Grafica, listener e relazione con controllore di UC
- Classi di controllo
- Classi entità e di relativa interfaccia verso la persistenza
- Persistenza

# Client - Server

Richiami

# Web Application – BCE

Trattasi di applicazioni distribuite web-based che forniscono funzionalità accessibili via web utilizzando come terminali normali web browser.

La View non è istanziata sulle macchine utente, quindi non bisogna proteggere oggetti e loro riferimenti dagli utenti come nelle stand-alone e client-server.

La View può allora conservare in proprio gli attributi di I/O necessari per l'esecuzione del caso d'uso. Attributi di I/O dello UC e accesso al Controller del caso d'uso possono essere riprodotti in una stessa classe della View e di solito lo sono (UCJBean).

# Web Application – MVC

## Definizione

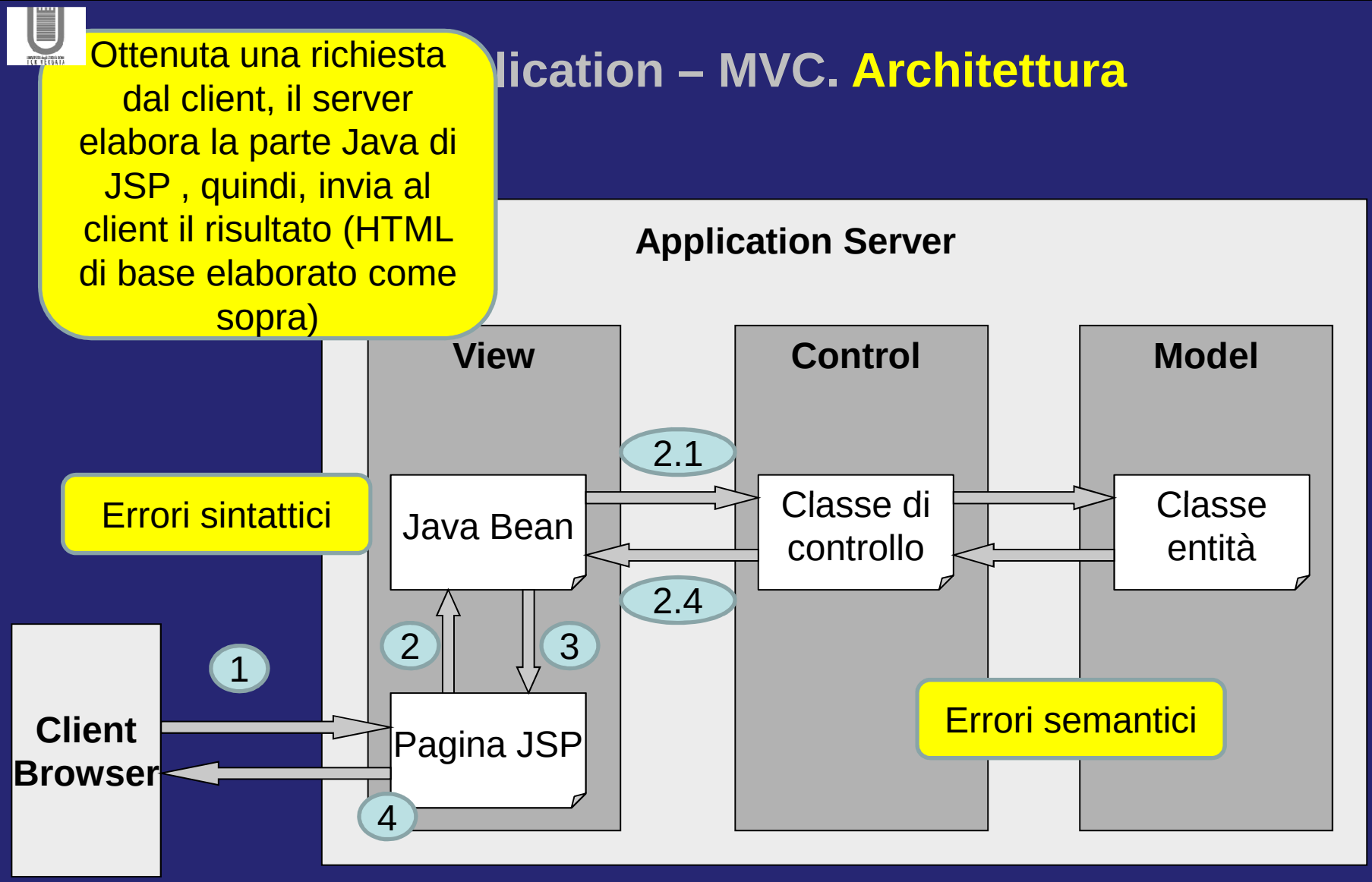
Trattasi di applicazioni distribuite web-based che forniscono funzionalità accessibili via web utilizzando come terminali normali web browser.

Una web application basata sul pattern MVC è tipicamente composta da quattro tipologie di componenti:

- Pagine JSP
- Classi Java Bean
- Classi di controllo
- Classi entità
- Persistenza


Nella **View**, sostituiti:
1) Action listener
2) Servlet dei Client-Server, perché rigide

In pratica, le Java Server Pages (ex Servlet) contengono la **grafica HTML** + minimo di Java per la **attivazione dei Java Bean**; tale separazione facilita sviluppo e manutenibilità


Ottenuta una richiesta dal client, il server elabora la parte Java di JSP , quindi, invia al client il risultato (HTML di base elaborato come sopra)
lication – MVC. Architettura
Application Server
View
Control
Model
2.1
Errori sintattici
Java Bean
Classe di controllo
Classe entità
2.4
2
3
1
Client Browser
Pagina JSP
Errori semantici
4

# Web Application – MVC. Descrizione dei componenti.

## Strato View

### Pagine JSP (*Java Server Pages*)

- Appartengono allo strato di presentazione e permettono all'utente di inserire o visualizzare dati (HTML)
- Contengono frammenti di codice Java per consentire ai Bean di accedere ai dati inseriti dall'utente, gestire i contenuti dinamici e per modificare lo HTML di base per adeguarlo dinamicamente alle esigenze.

### Classi Java Bean

- Rappresentano il ponte che unisce una pagina JSP al resto dell'applicazione (comunicano direttamente con lo strato Control e Model)
- Realizzano il controllo di validazione sintattica sull'inserimento dei dati e tengono traccia dei dati in visualizzazione.

# Web Application – MVC. Descrizione d… enti.

## Strato Vie…

**Pagine JSP (*Java Server Pa…*)**

- Appartengono all… inserire o vi…
- C…

…al resto dell'applicazione …trol e Model)

- …ne sintattica sull'inserimento dei dati e te… …isualizzazione.

N.B. A differenza delle stand-alone, non è il controllore a identificare la GUI successiva ma questa deve essere richiesta dal client al server.

Ad esempio, quando l'utente fa click su un bottone, viene inviata una richiesta al server (1) che la processa. Al ritorno, è solo la pagina JSP che, essendo processata come *servlet*, ha il compito di comunicare al server quale pagina inviare al client (4).

Il controllo di flusso di presentazione ("*View selection*") è ora, dunque, eseguito all'interno della view (*violazione* di MVC).

# Web application – MVC. Descrizione dei componenti

**Strato Control – classi di controllo**

- Racchiudono la logica di esecuzione dell'applicazione
- Modificano lo stato dell'applicazione interagendo sulle classi entità, in base alle richieste effettuate dallo strato View
- *All'interno di una web application il Control non gestisce il flusso di presentazione*

**Strato Model – classi entità**

- Incapsulano la logica di business e applicativa (stato dell'applicazione)
- Espongono i comportamenti e, dunque, permettono la modifica di stato e l'invocazione della storicizzazione
- *All'interno di una web application le classi entità non possono notificare una modifica allo strato View, ma deve essere quest'ultimo ad effettuare il controllo*

# Typical User Scenario

1. Let a user request to visit a URL.
    1. The browser then generates an HTTP request for this URL.
    2. This request is then sent to the appropriate server.
2. The HTTP request is received by the web server and forwarded to the servlet container.
    1. The container maps this request to a particular servlet.
    2. The servlet is dynamically retrieved and loaded into the address space of the container.
3. The container invokes the `init()` method of the servlet.
    1. This method is invoked only when the servlet is first loaded into memory.
    2. It is possible to pass initialization parameters to the servlet so that it may configure itself.



http://en.wikipedia.org/wiki/Servlet

# Typical User Scenario

4. The container invokes the **`service()`** method of the servlet.
   1. This method is called to process the HTTP request.
   2. The servlet may read data that has been provided in the HTTP request.
   3. The servlet may also formulate an HTTP response for the client.
5. The servlet remains in the container's address space and is available to process any other HTTP requests received from clients.
   1. The **`service()`** method is called for each HTTP request.



http://en.wikipedia.org/wiki/Servlet

## Typical User Scenario

4. The container calls the servlet's **`destroy()`** method to relinquish any resources such as file handles that are allocated for the servlet; important data may be saved to a persistent store.
5. The memory allocated for the servlet and its objects can then be garbage collected.



http://en.wikipedia.org/wiki/Servlet

# Web application – Typical User Scenario

JavaServletExample



http://en.wikipedia.org/wiki/Servlet