LEZIONE 33 e 34

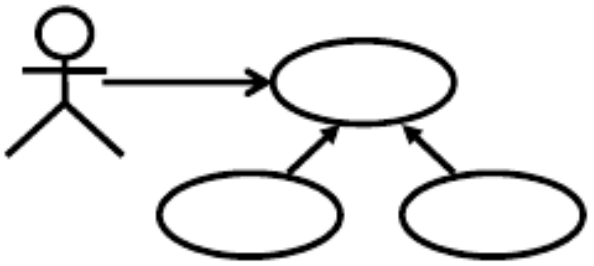
MVC, MVP, Architetture per applicazioni stand-alone e web

Ingegneria del Software e Progettazione Web
Università degli Studi di Tor Vergata - Roma

Guglielmo De Angelis
guglielmo.deangelis@isti.cnr.it

design guidato dalle responsabilità

Required Functionality



Use-Case Model

"realized by client classes"

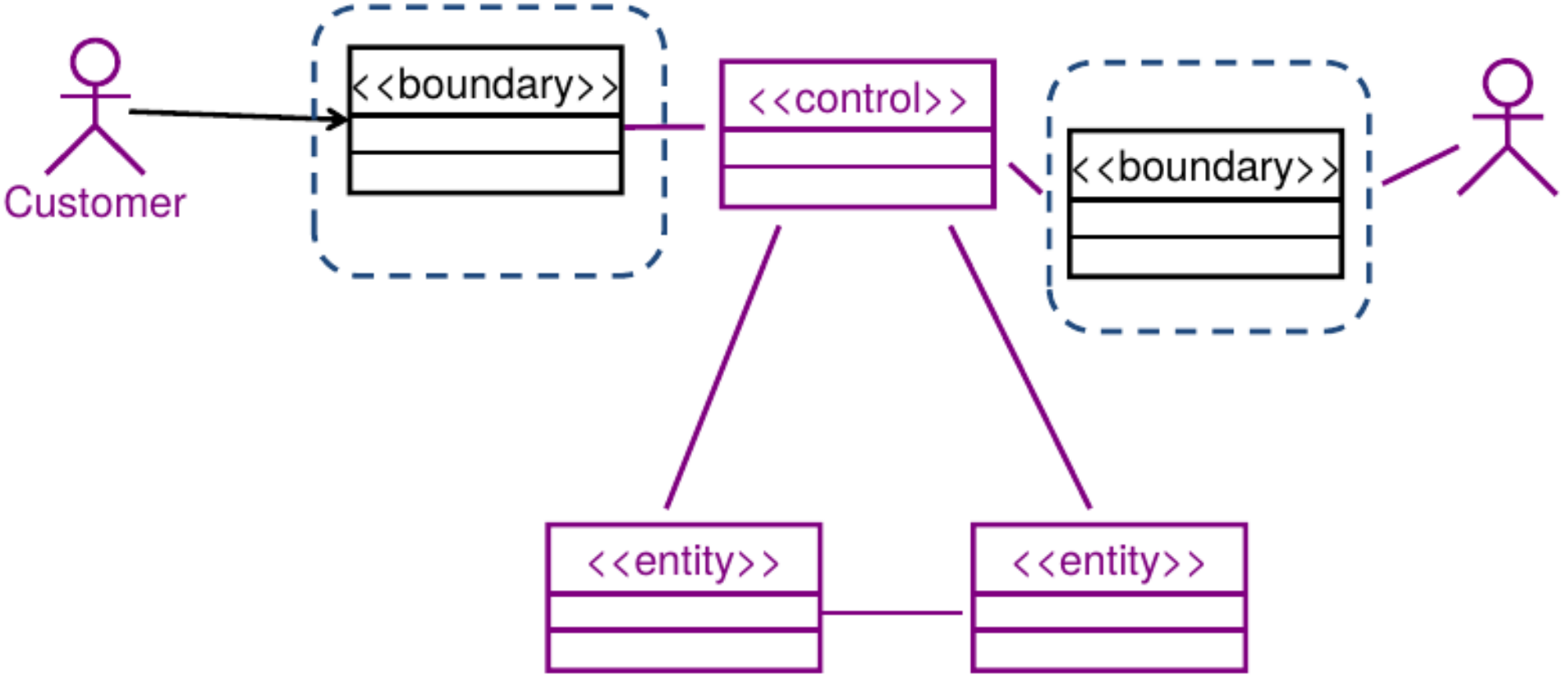
Analysis Classes

- **Entity Classes**
- **Control Classes**
- **Boundary Classes**

BCE : quali responsabilità?

- **boundary**: mediano l'interazione tra il sistema e l'ambiente
  - rappresentano elementi al confine del sistema
  - regolano l'interazione con gli attori
- **control**: coordinano il comportamento durante un caso d'uso del sistema
  - rappresentano comportamenti dipendenti dall'interazione attesa con il sistema (i.e. comportamento descritto dal caso d'uso)
  - sono indipendenti dal modo di attivazione dell'interazione
  - queste classi dovrebbero scaturire dall'analisi dominio del problema
- **entity**: rappresentano le astrazioni chiave del sistema
  - sono indipendenti dall'ambiente di esecuzione
  - dipendono dall'analisi dominio del problema
    - glossario, use case, business model, etc.
  - modellano il comportamento di una entità di dominio incapsulando un insieme coeso di dati

schema di interazioni in BCE



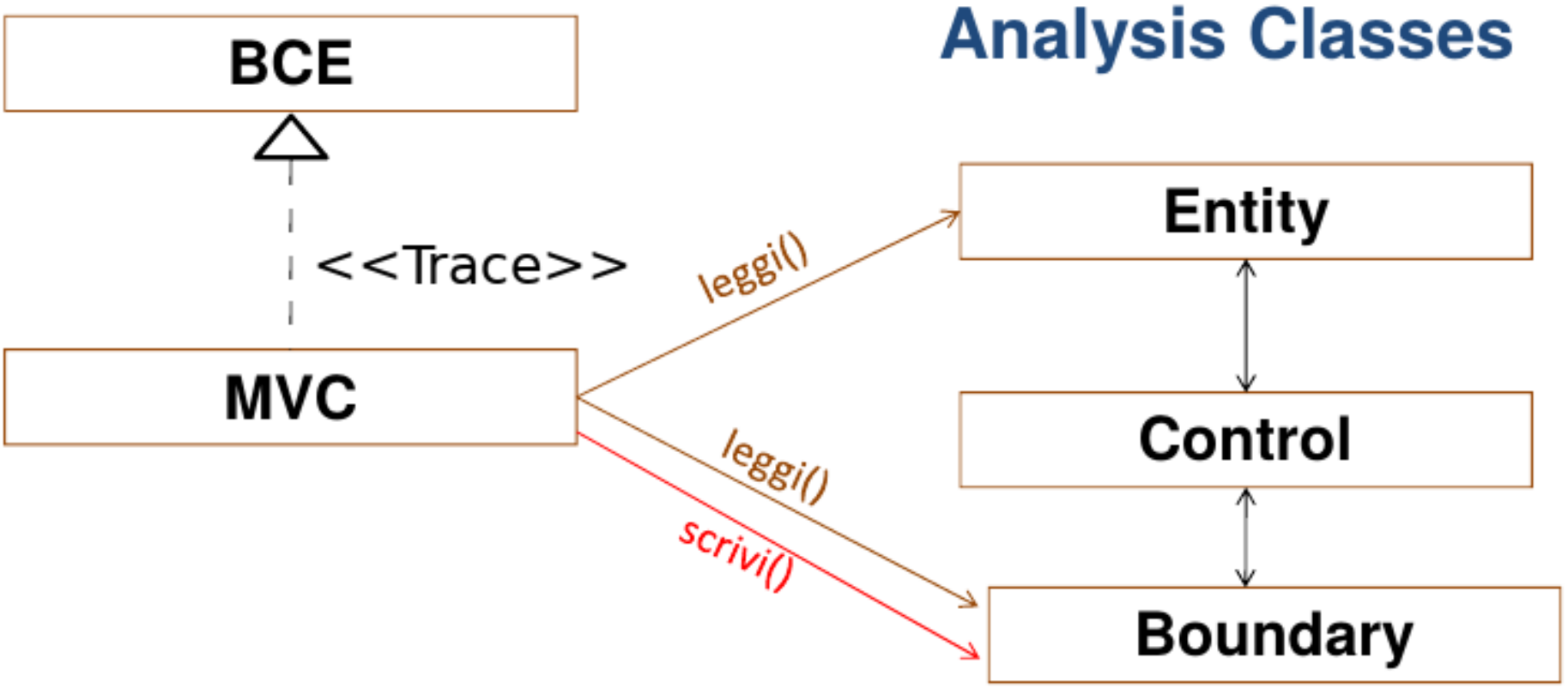
… oltre BCE

- BCE è un pattern di analisi per sistemi O.O.
  - lo scopo principale è identificare le macro classi del sistema
  - le classi derivano dall'osservazione:
    - del dominio
    - dai modi in cui il sistema è utilizzato dagli attori (i.e. casi d'uso)
- come raffinare un modello di analisi BCE?
- come gestire/introdurre aspetti:
  - nel dominio della soluzione
  - che tengano conto di aspetti di ingegnerizzazione del sistema (e.g. riuso, ottimizzazione delle soluzioni, manutenibilità, scalabilità, etc.)

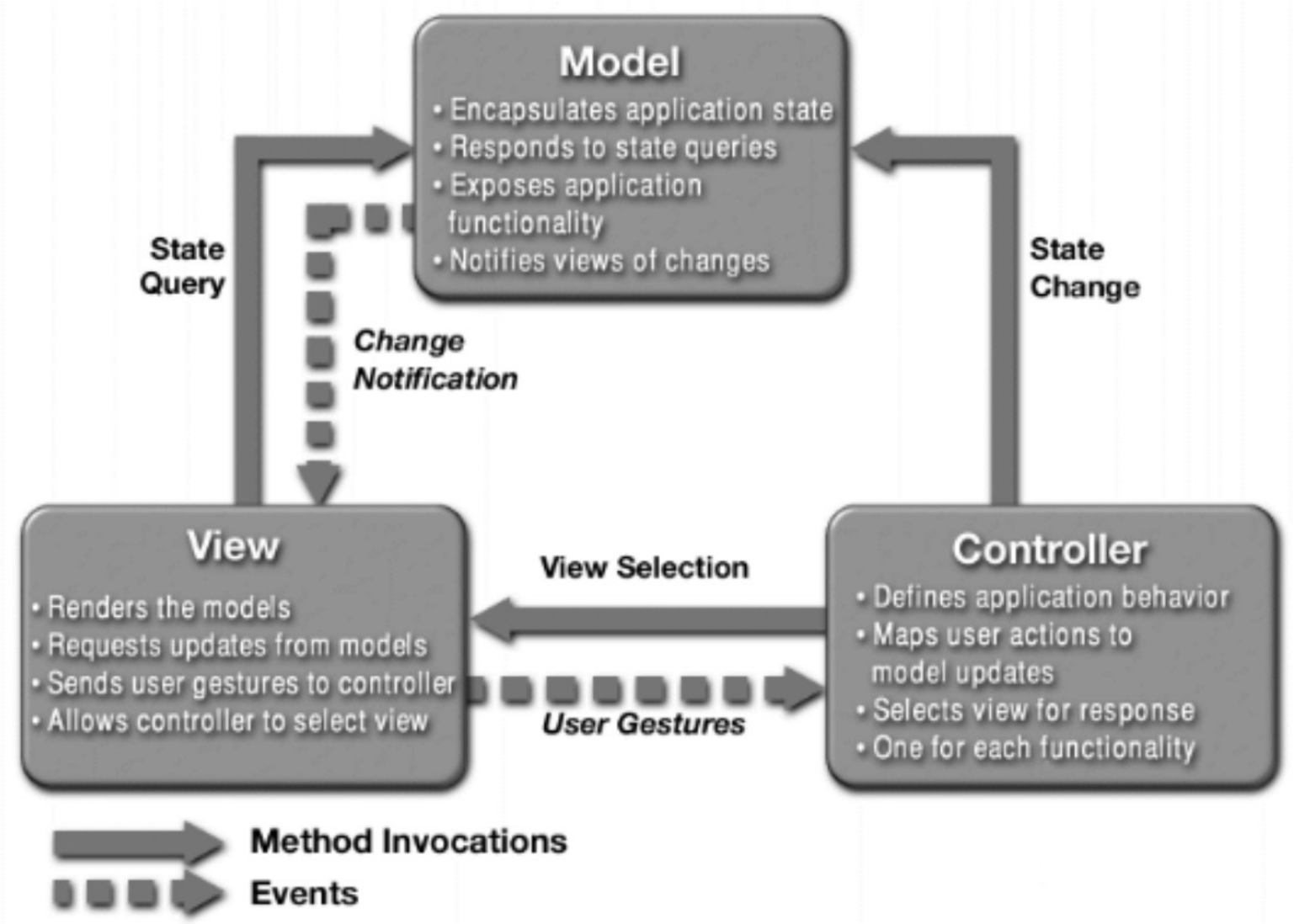
… oltre BCE

- BCE è un pattern di analisi per sistemi O.O.
  - lo scopo principale è identificare le macro classi del sistema
  - le classi derivano dall'osservazione:
    - del dominio
    - dai modi in cui il sistema è utilizzato dagli attori (i.e. casi d'uso)
- come raffinare un modello di analisi BCE?
- come gestire/introdurre aspetti:
  - nel dominio della soluzione
  - che tengano conto di aspetti di ingegnerizzazione del sistema (e.g. riuso, ottimizzazione delle soluzioni, manutenibilità, scalabilità, etc.)
- il pattern di analisi BCE può essere raffinato secondo diverse varianti:
  - Model-View-Controller (MVC)
  - Model-View-Presenter (MVP)

model-view-controller



model-view-controller



MVC – view

- responsabile per la
  - la logica di presentazione dei dati
  - la costruzione/gestione dell'interfaccia grafica (GUI)
  - acquisizione dei dati dell'applicazione
    - (se non ne è responsabile il del controller) può gestire la conversione dei dati da formato esterno a interno (e viceversa)
    - “12 Ottobre 1942” → {1492; 10; 12}
  - accedere ai dati in SOLA lettura direttamente al Model (attributi)
- mantenere aggiornati i dati presentati
  - push model: applicazione del GoF Observer
    - le View possono registrarsi come osservatori di Model
    - le View ricevono aggiornamenti di Model in “tempo reale”
  - pull model: utilizzato nel caso in cui la View intende richiedere gli aggiornamenti solo quando "opportuno“

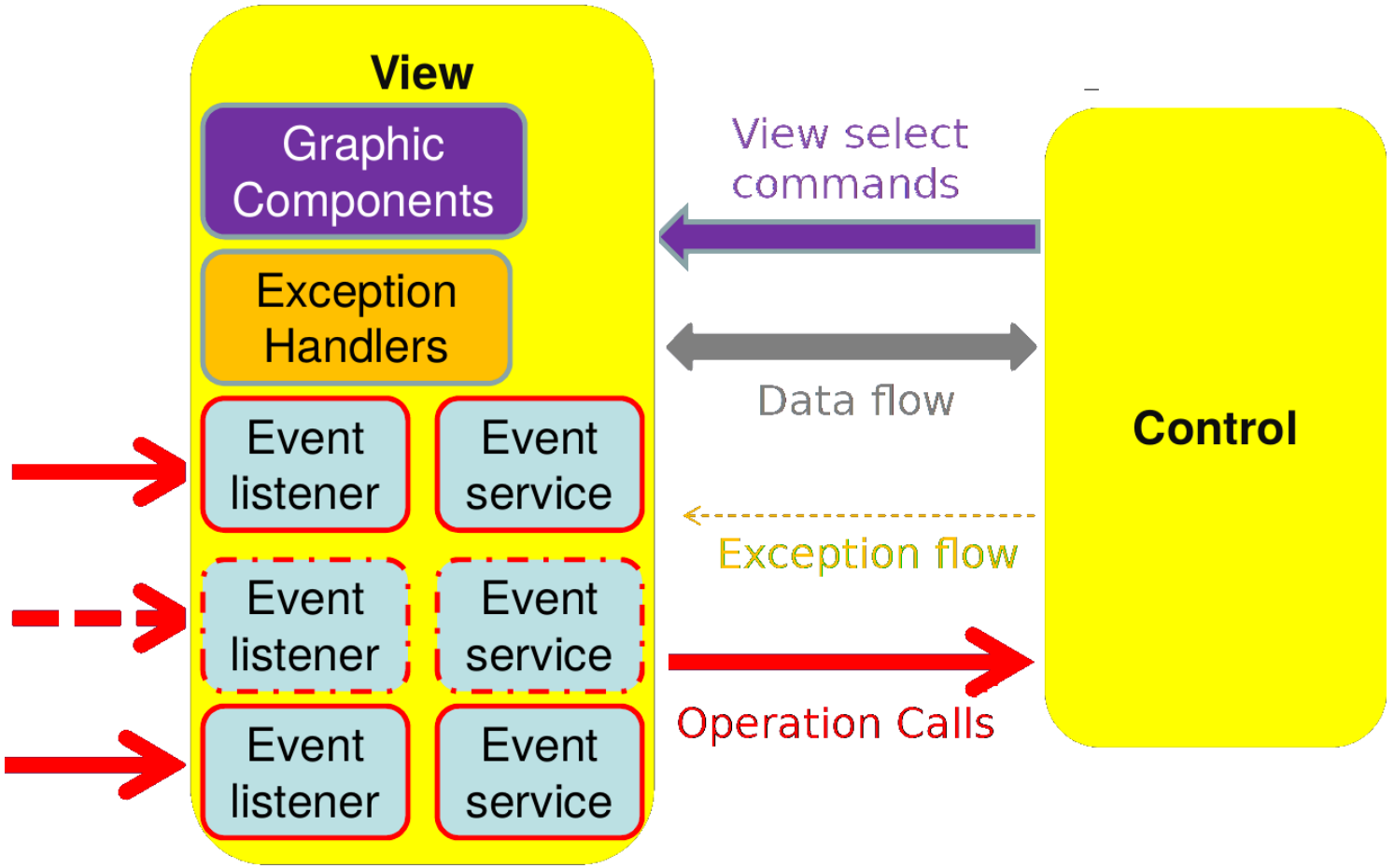
MVC – controller

- controller grafico dell'applicazione: coordina l'interazione tra View e Model per la realizzazione di una funzionalità
  - converte formati esterno in interno e viceversa (se non a carico della View)
  - realizza la mappa tra input dell'utente e processi eseguiti dal Model
  - crea/seleziona le istanze di View richieste
- controller applicativo: implementa la logica di controllo dell'applicazione
  - raffina il concetto di Controller in BCE
  - ha la responsabilità di gestire una azione dell'utente (sulla View) in un o più azioni eseguite sulle istanze nel Model
  - per quanto possibile data l'applicazione dovrebbe avere un comportamento state-less
- progettazione ed implementazione:
  - controller grafico ed applicativo aggregati in un unico "bundle"
    - il controller grafico implementa direttamente la logica di controllo
  - controller grafico ed applicativo disaccoppiati
    - il controller grafico invoca operazioni sul controller applicativo

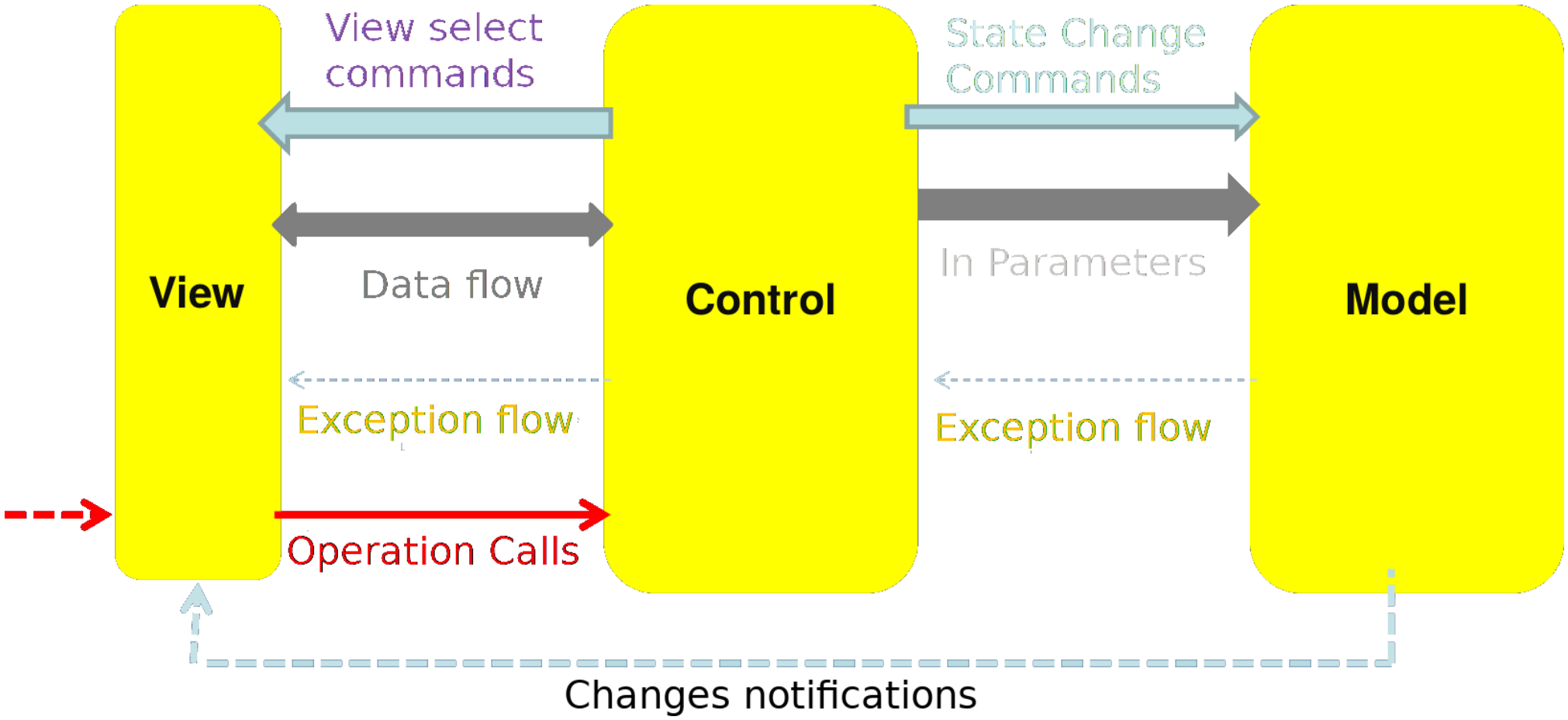
MVC – model

- costituisce la rete di entità partecipanti alla logica di applicazione
  - per ogni entità
    - descrive i comportamenti esposti in forma di operazioni
    - gestisce/realizza le relazioni tra le entità
- come nel caso Model BCE
  - incapsulare lo stato (i.e. attributi) delle entità dell'applicazione enfatizzando il ruolo delle operazioni offerte
  - un buon modello di entità non dovrebbe esporre lo stato di un oggetto
- può avere la responsabilità di notificare ai componenti View eventuali aggiornamenti verificatisi (in seguito a richieste del Controller)

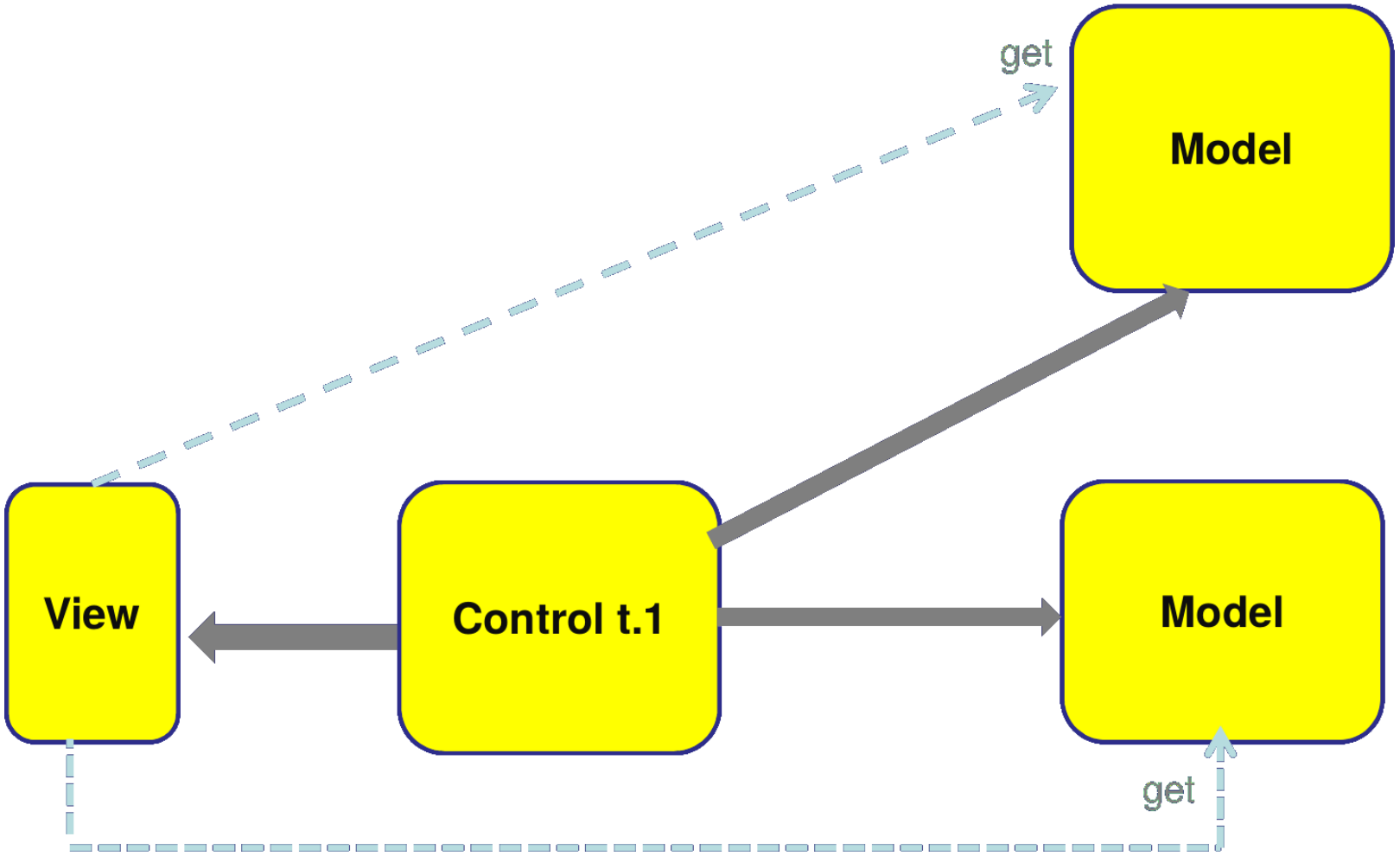
MVC – interazioni utente



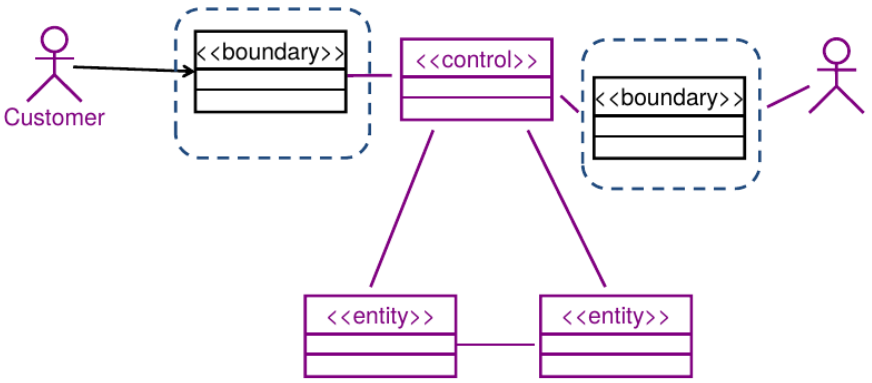
MVC – interazioni utente && push-model

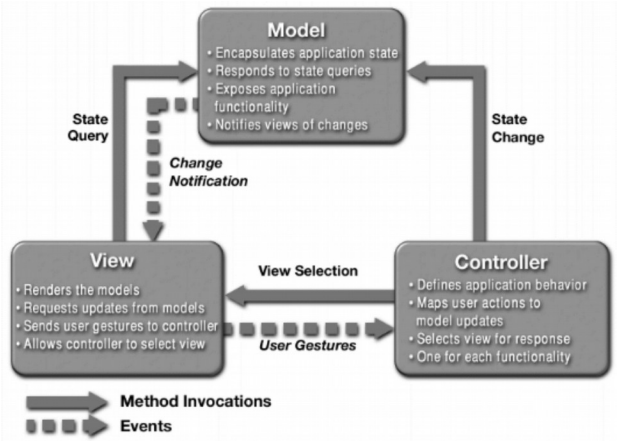


MVC – interazioni utente && pull-model



gestione delle evoluzioni in BCE e MVC



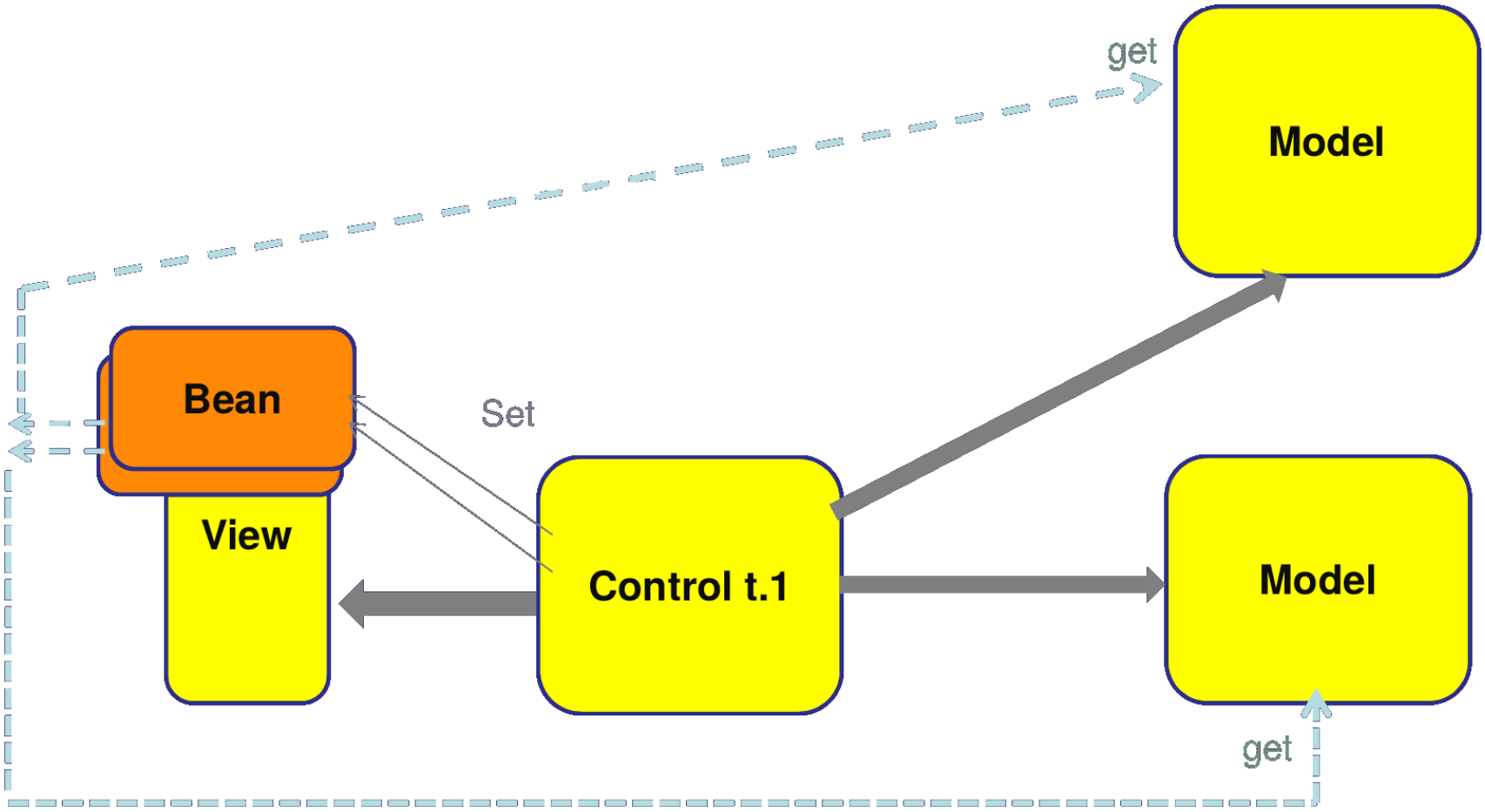


- evoluzione del Model → quali impatti a seguito di:
  - evoluzione dei formati di rappresentazione delle informazioni
  - evoluzione delle key abstractions di una applicazione
  - evoluzione della rappresentazione delle key abstractions di una applicazione
  - riorganizzazione delle relazioni tra key abstractions
- evoluzioni sulla View/Boundary → quali impatti nel caso di:
  - evoluzioni strettamente legate alla variazione dei meccanismi di IO
  - gestione simultanea di più versioni/varianti della stessa View

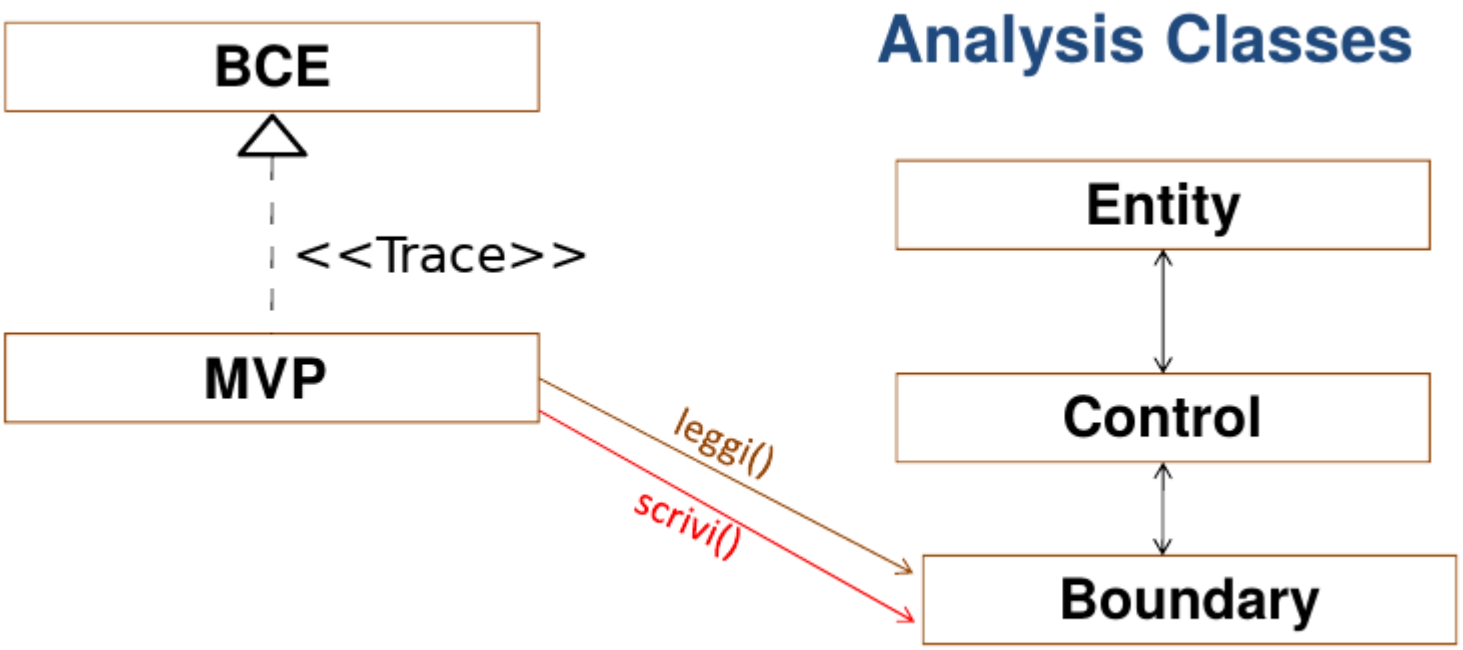
classi “bean”

- in generale:
  - elemento di buffering per la comunicazione fra classi Boundary, Control ed Entity
  - principalmente sono interposte fra le classi Boundary e il resto delle classi del VOPC
  - **MODELLARE LE CLASSI BEAN FIN DAL BCE, PER POI FARLE RAFFINARLE IN MVC**
- principali responsabilità
  - disaccoppiamento tra classi Boundary ed Entity
  - controllo della validazione sintattica sull’inserimento dei dati in input
  - eventuale gestione dei riferimenti per i dati in visualizzazione
    - gestione pull-mode o push-mode
- struttura tipica
  - realizzano un POJO (Plain Old Java Object)
  - attributi privati destinati a contenere dati di IO
  - principalmente metodi pubblici solo di tipo setter e getter
  - accesso agli attributi solo attraverso IO setter e getter
  - metodi privati per il controllo sintattico dei dati

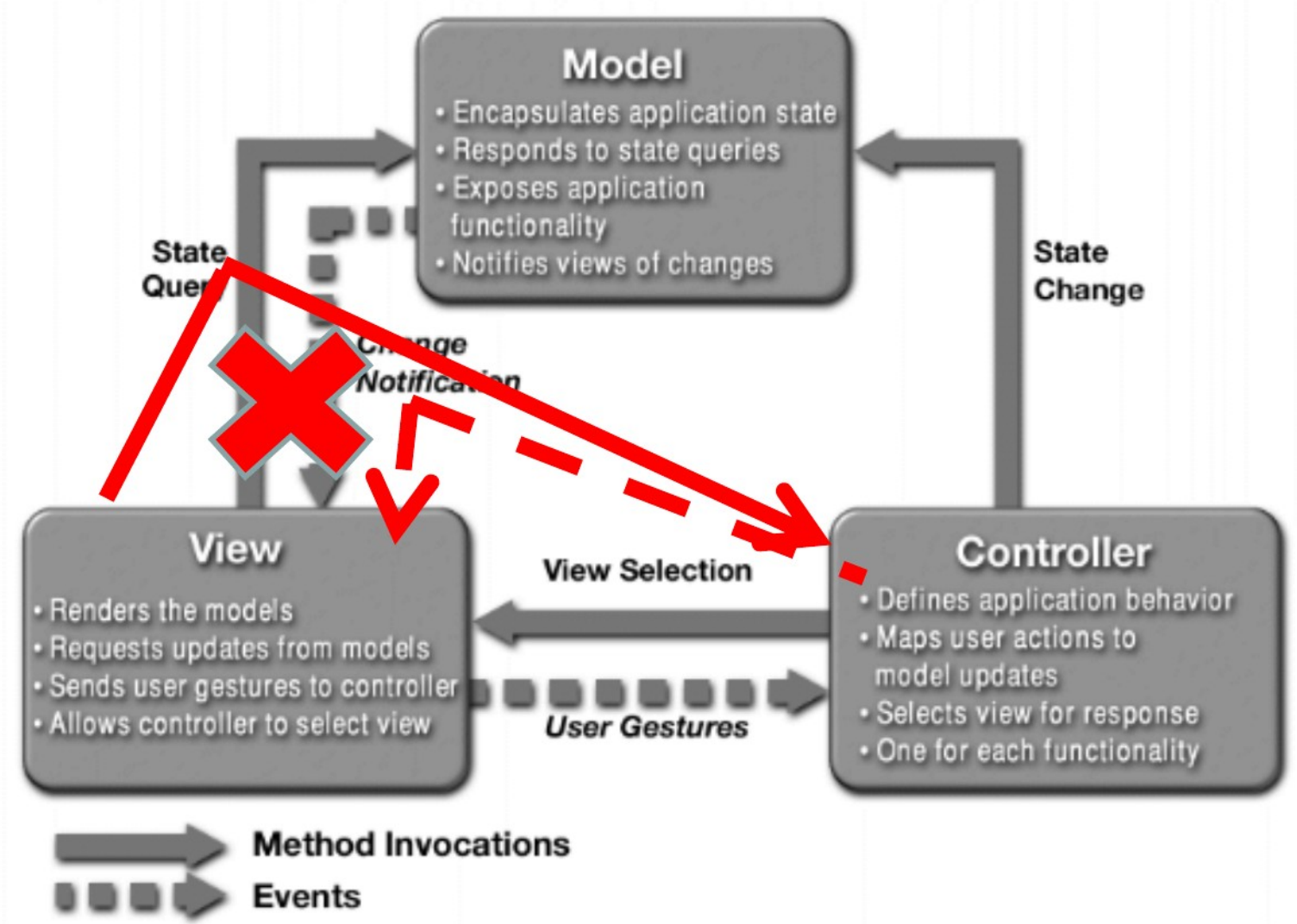
MVC – pull-model con beans



model-view-presenter



model-view-presenter



MVP – dettagli

- View
  - i dati da presentare **NON possono** essere acquisiti direttamente dal Model
  - la View ha accesso ai metodi del Model del tipo Attributo (e.g. getAttributo())
  - la View deve sempre interagire con il Controller
  - le restanti responsabilità sono pari a quelle di View di MCV
    - anche nel caso MVP i dati scambiati tra la parte View e la parte Presentation vanno intesi come classi Bean di appoggio
- Presentation Model
  - come Control di MVC.
- Model
  - come per MVC

- principalmente utilizzato per sviluppare GUI

credits

parte dei contenuti di queste slide sono stati elaborati a partire dalla presentazione del Prof. Giovanni Cantone :

“Boundary-Control-Entity Structuring Concepts for Stand-alone (Lap-top) Applications, and Web Applications”