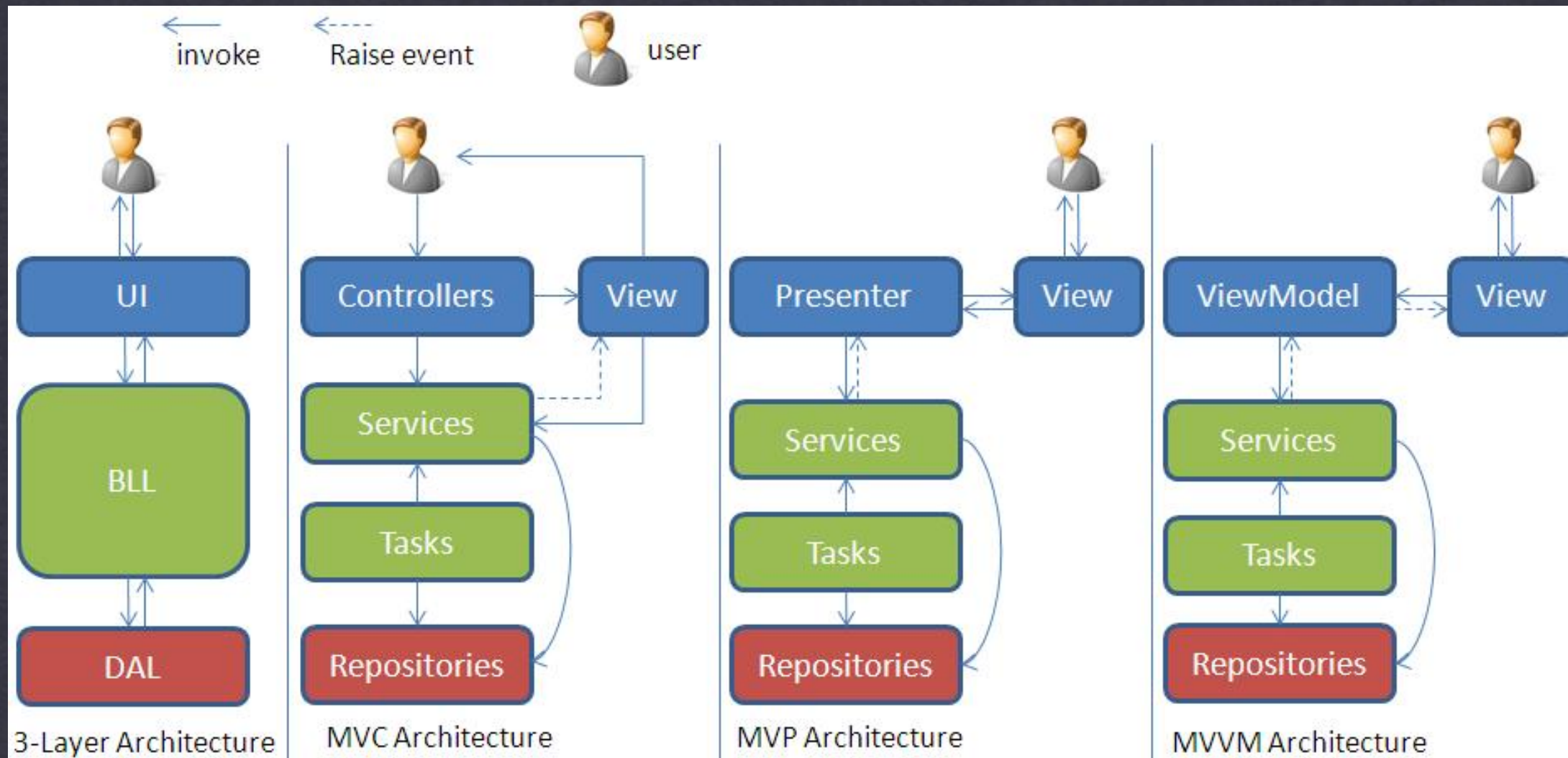


Material and some slide content from:
- Krzysztof Czarnecki
- Ian Sommerville
- Reid Holmes
- Head First Design Patterns



MVC / MVP

Mei Nagappan

Background

- ▶ MVC started w/ Smalltalk-80
- ▶ Java UI frameworks & EJBs reignited interest
- ▶ Also prevalent in GWT and .NET development



MVC Motivation

- ▶ UI changes more frequently than business logic
 - ▶ e.g., layout changes (esp. in web applications)
- ▶ The same data is often displayed in different ways
 - ▶ e.g., table view vs chart view
 - ▶ The same business logic can drive both
- ▶ Designers and developers are different people
- ▶ Testing UI code is difficult and expensive
- ▶ Main Goal: Decouple models and views
 - ▶ Increase maintainability/testability of system
 - ▶ Permit new views to be developed

Model

- ▶ Contains application data
 - ▶ This is often persisted to a backing store
- ▶ Does not know how to present itself
- ▶ Is domain independent
- ▶ Are often Subjects in the Observer pattern



View

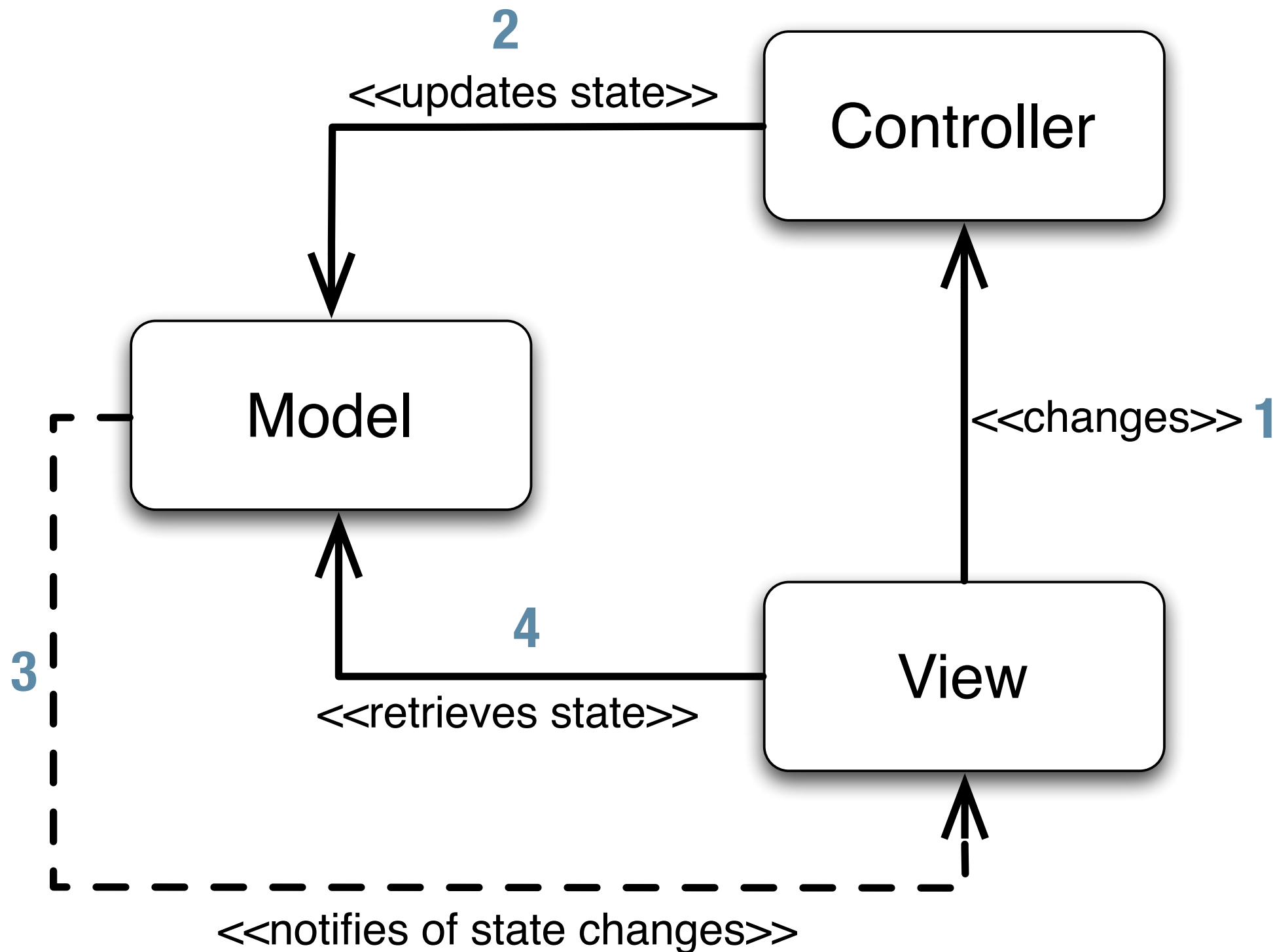
- ▶ Presents the model to the user
- ▶ Allows the user to manipulate the data
- ▶ Does not store data
- ▶ Is configurable to display different data

Controller

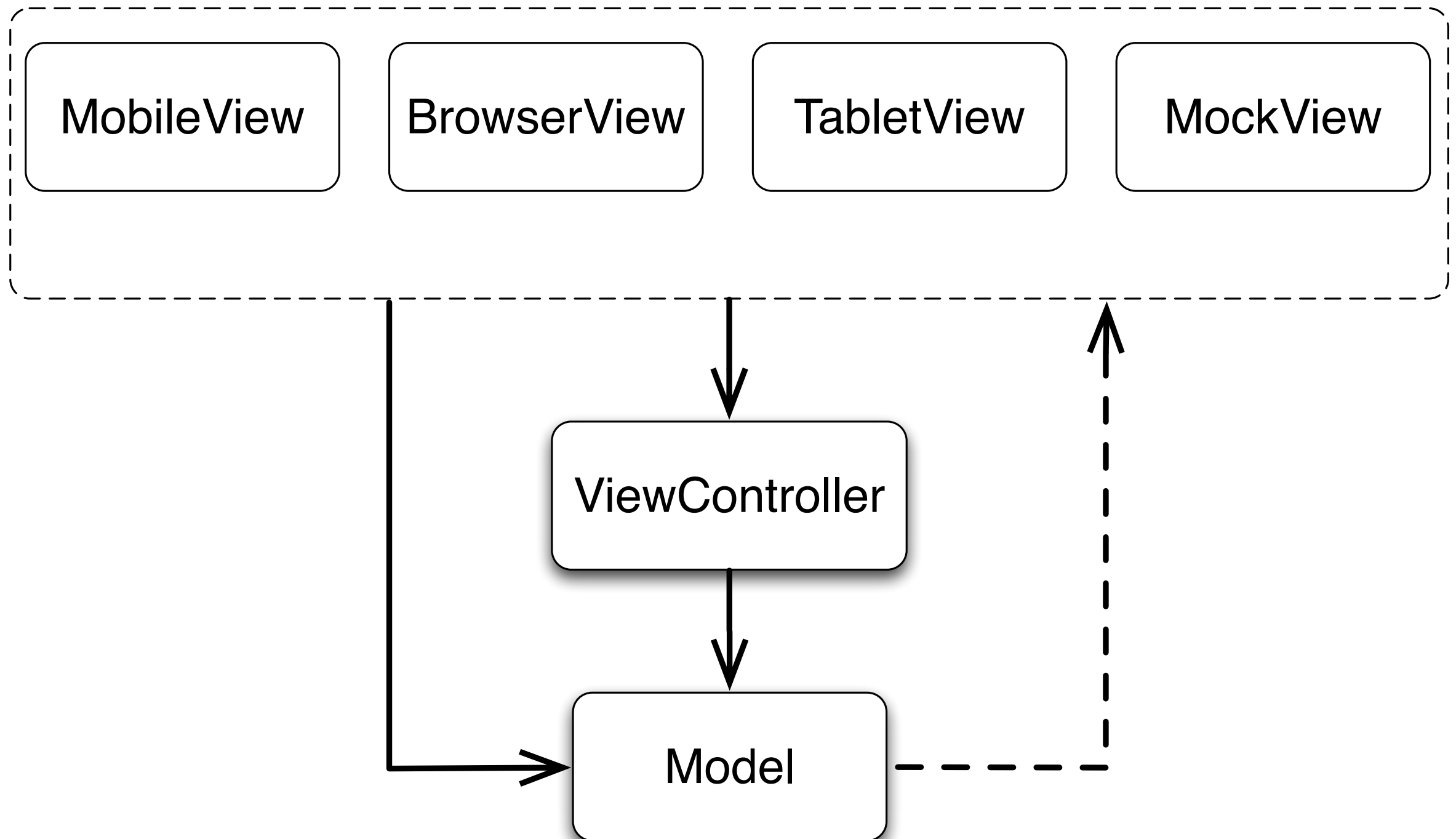
- ▶ Glues Model and View together
- ▶ Updates the view when the Model changes
- ▶ Updates the model when the user manipulates the view
- ▶ Houses the application logic
- ▶ Loose coupling between Model and others
- ▶ View tightly cohesive with its Controller



Abstract topology



Concrete topology



Interaction mechanism

- ▶ User interacts with the UI (View)
- ▶ UI (View) notifies controller of changes
- ▶ Controller handles notifications, processing them into actions that can be performed on the model
- ▶ Controller modifies the model as required
- ▶ If the model changes, it fires modification events
- ▶ The view responds to the modification events



Benefits and tradeoffs

- ▶ Pro:
 - ▶ Decouple view from model
 - ▶ Support multiple views [collaborative views]
 - ▶ Maintainability [add new views]
 - ▶ Split teams [relieve critical path]
 - ▶ Testability [reduce UI testing]
- ▶ Con:
 - ▶ Complexity [indirection, events]
 - ▶ Efficiency [frequent updates, large models]

Compound Pattern

- ▶ MVC (and other similar patterns) rely upon several more basic design patterns
- ▶ In MVC:
 - ▶ View/Controller leverage the strategy pattern
 - ▶ View is often a composite pattern
 - ▶ View/Model interact through the observer pattern
- ▶ Other meta-patterns rely upon similar lower-level design patterns



MVP Motivation

- ▶ Take MVC a tiny bit further:
 - ▶ Enhance testability
 - ▶ Further separate Designers from Developers
- ▶ Leveraged by both GWT and .NET



Model

- ▶ Contains application data
 - ▶ This is often persisted to a backing store
- ▶ Does not know how to present itself
- ▶ Is domain independent
- ▶ Often fires events to an Event Bus



View

- ▶ Thin UI front-end for controller
- ▶ Does not store data
- ▶ Can be interchanged easily
- ▶ Does not ever see or manipulate Model objects
- ▶ Only interacts with primitives
 - ▶ e.g., (setUser(String) instead of setUser(User))

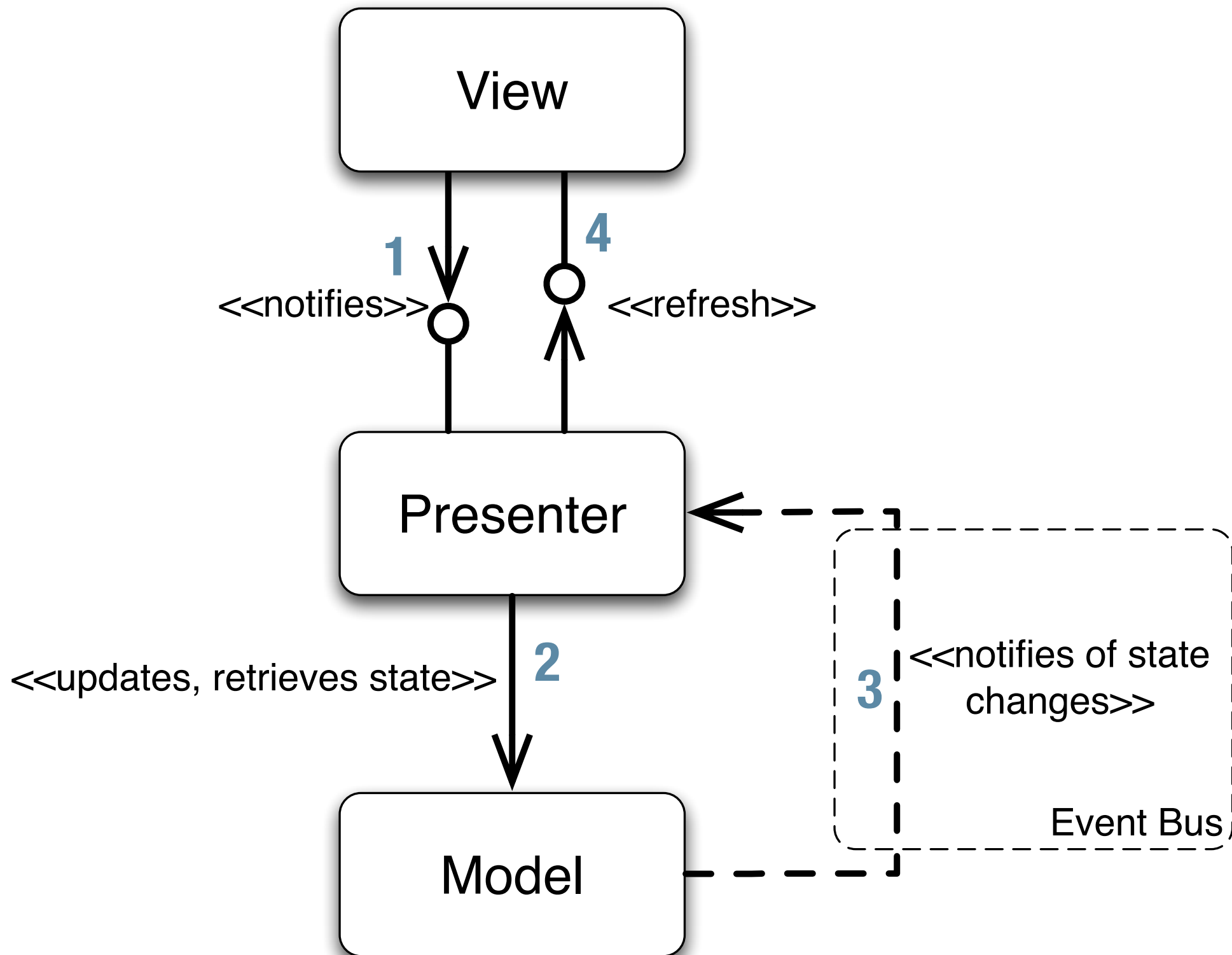


Controller

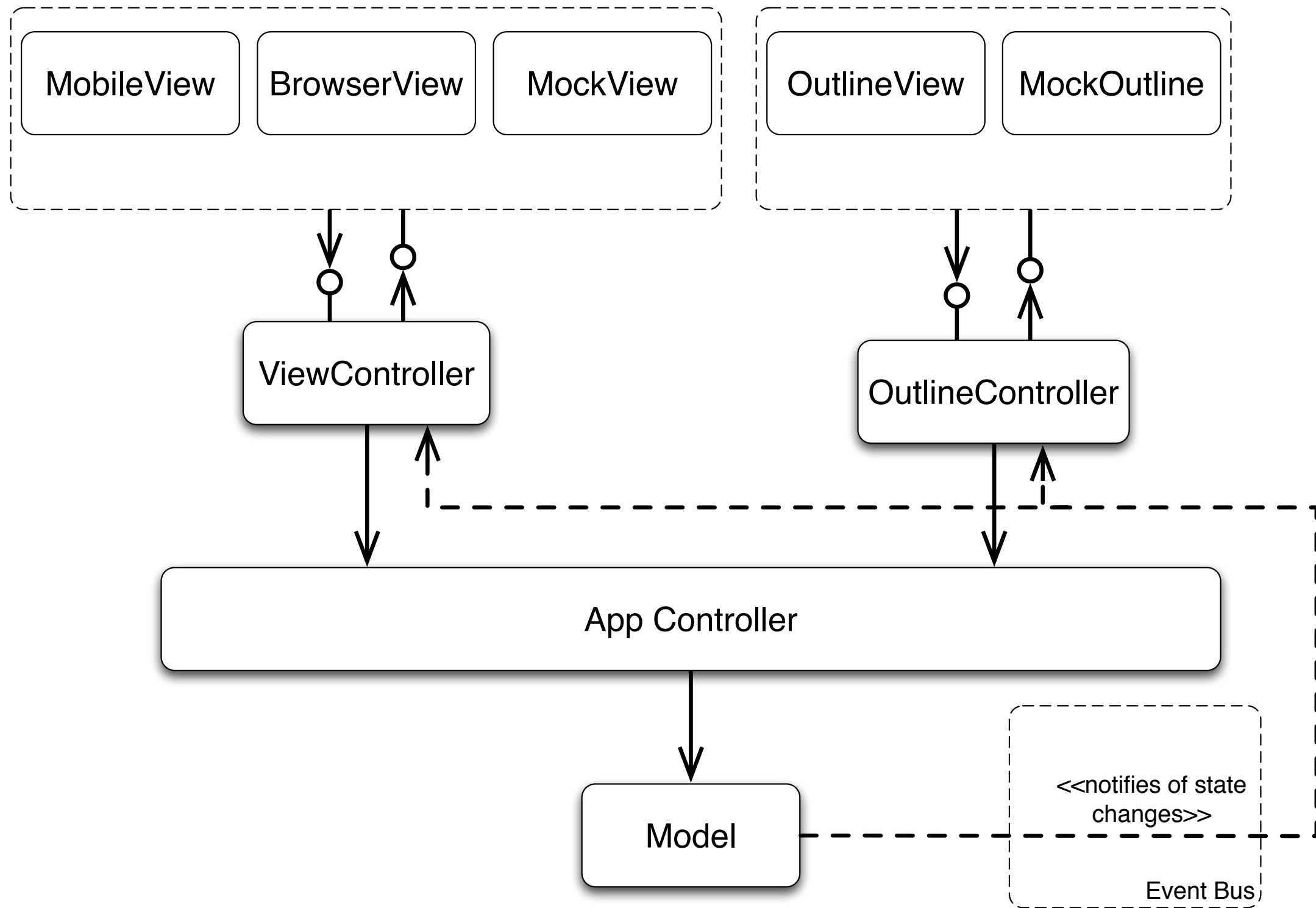
- ▶ Glues Model and View together
- ▶ Updates the view when the Model changes
- ▶ Updates the model when the user manipulates the view
- ▶ Houses the application logic



MVP Topology



Concrete MVP Topology



Benefits and tradeoffs

- ▶ Same as MVC with improved:
 - ▶ Decoupling of views from the model
 - ▶ Split teams [relieve critical path]
 - ▶ Testability [reduce UI testing]
 - ▶ A little less complex than MVC [fewer events]



Architecture/Design Review Meeting

- ▶ Don't think of this as an oral exam
- ▶ Start with 5 minute presentation (board only)
- ▶ Followed by 20 minute discussion
- ▶ Evaluating the product, not the producer
- ▶ Be prepared!
- ▶ Goal:
 - ▶ Ensure system meets proposal
 - ▶ Check consistency of design with architecture
 - ▶ Talk about arch/design decisions/justification
 - ▶ Discuss support for future system evolution

