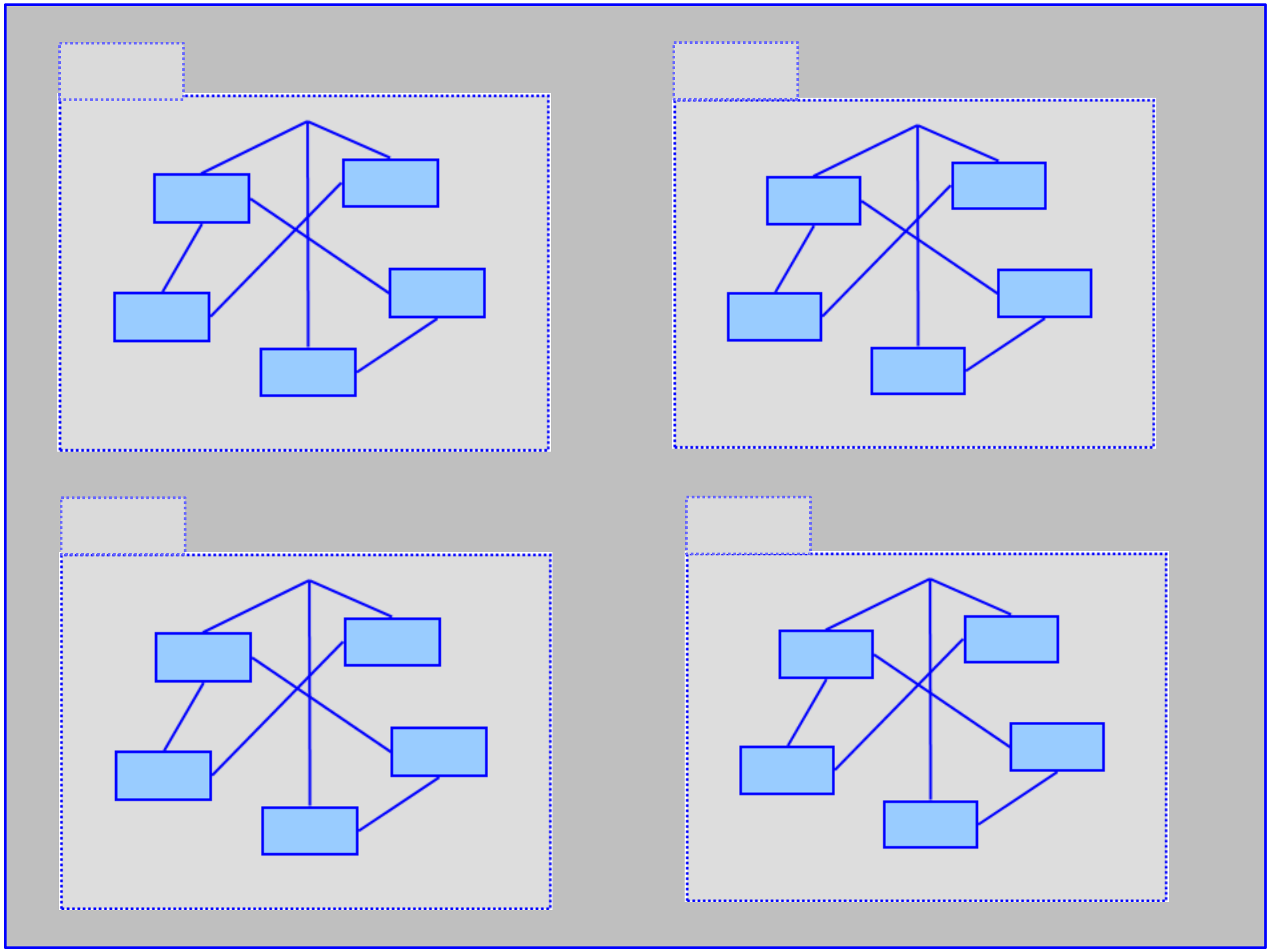
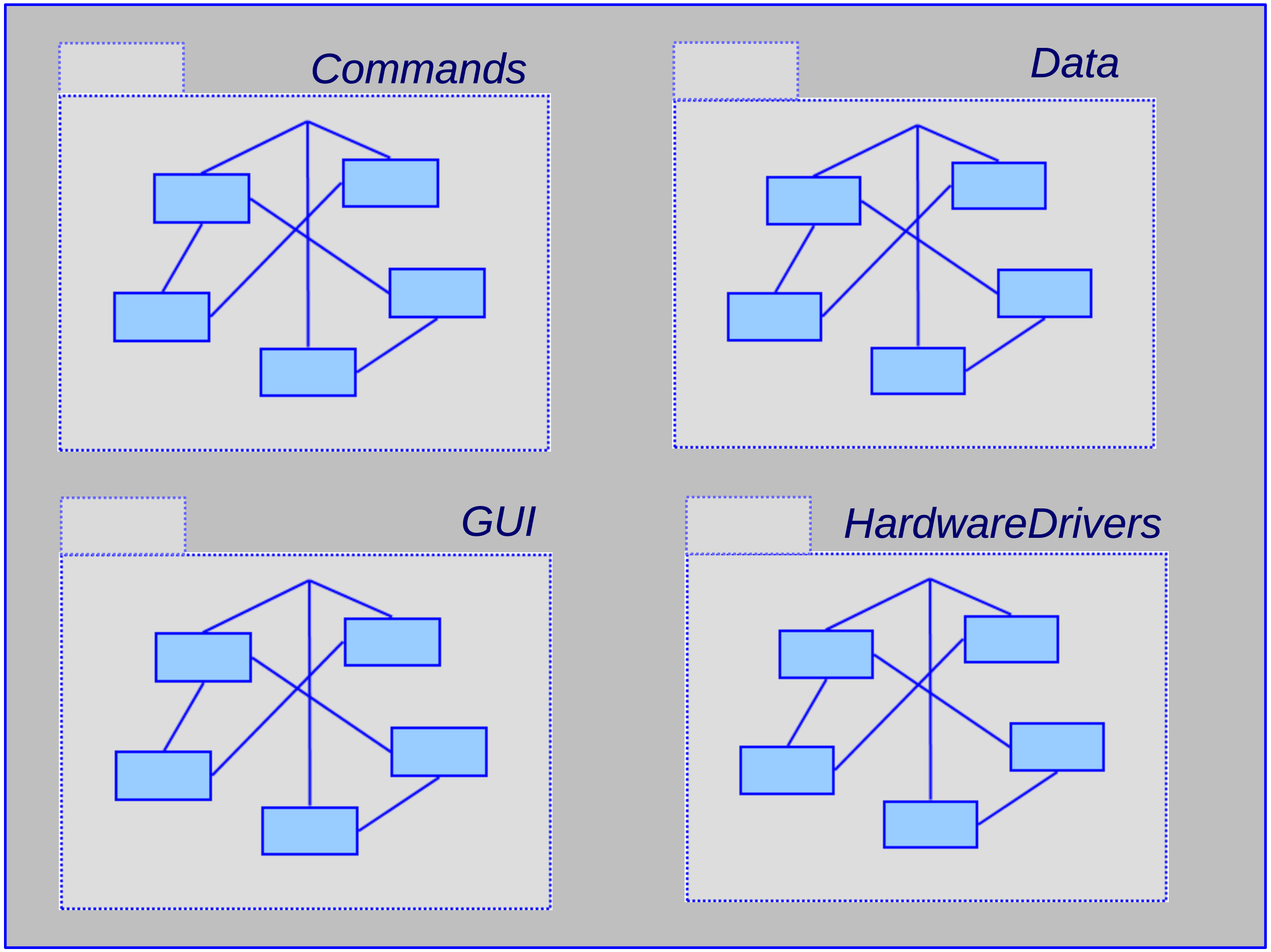
Design Patterns (3)

Acknowledgment: these slides are based on Prof. John Mylopoulos slides
which are used to teach a similar course in the University of Toronto
– St. George campus. Used with Permission.

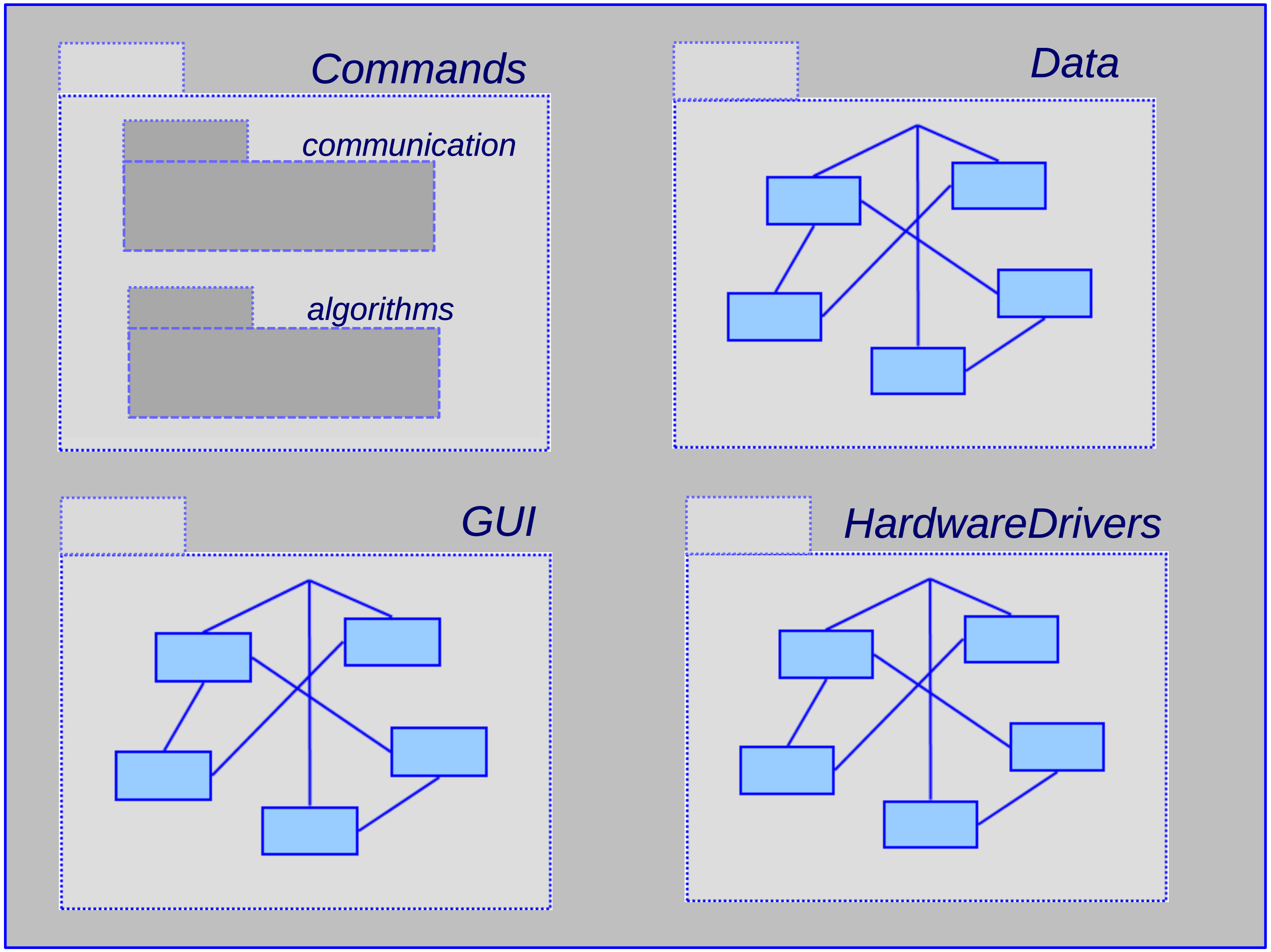
Application Subsystems



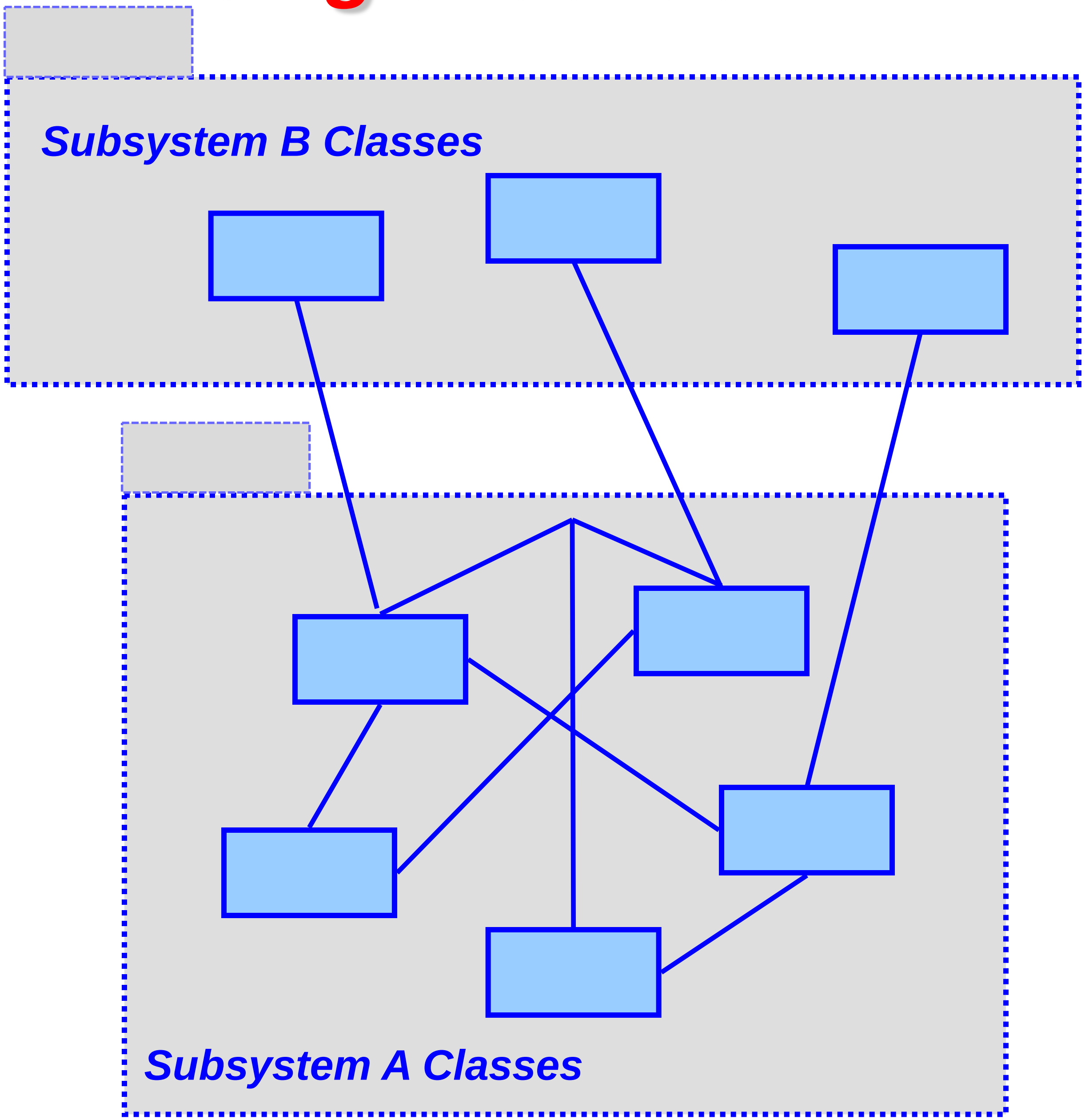
Application Subsystems - example



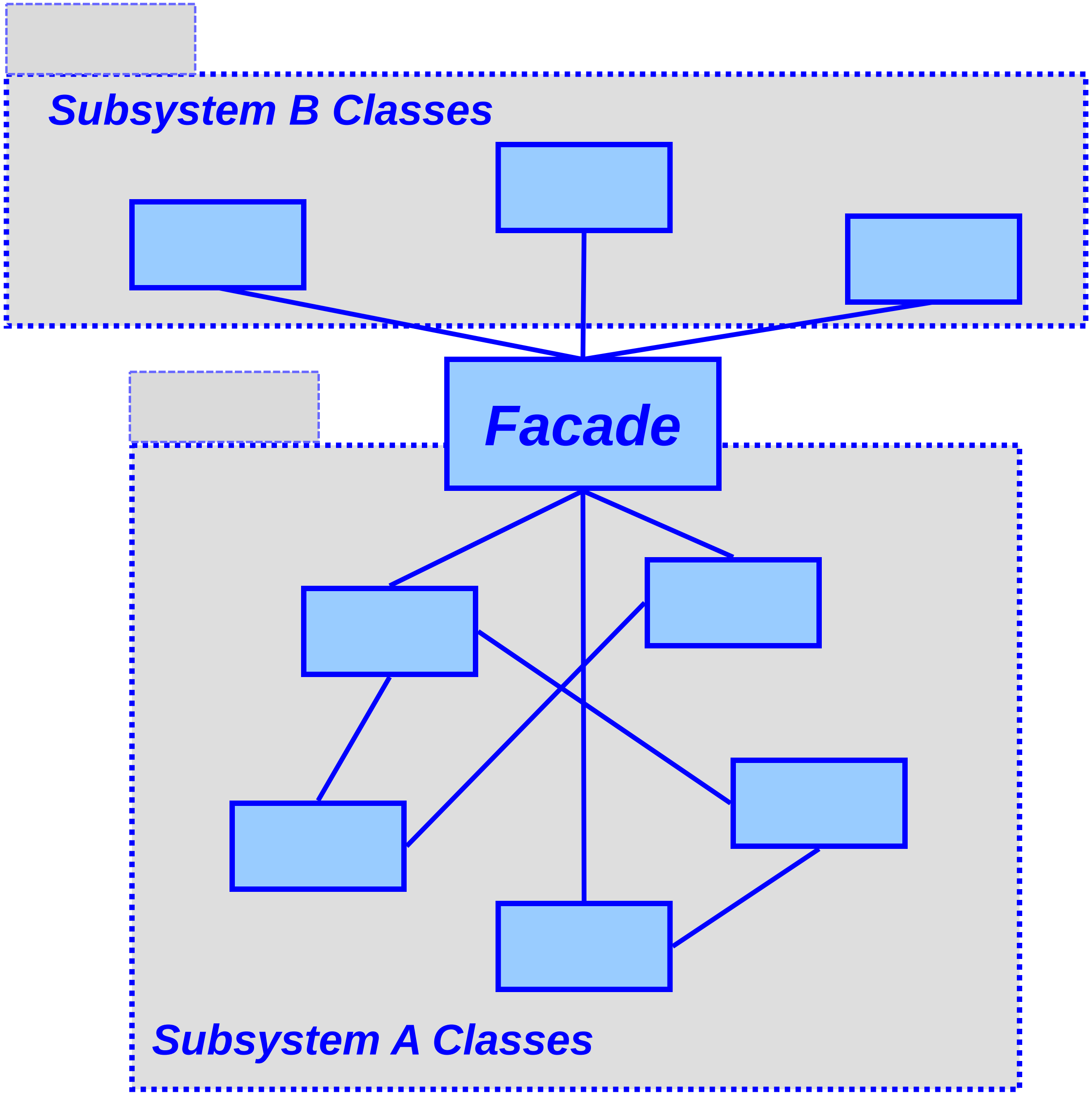
Application Subsystems - example



Before using a facade



Using Facade Pattern



Facade Pattern: Why and What?

- *Subsystems often get complex as they evolve.*
- **Need to provide a simple interface to many, often small, classes. *But not necessarily to ALL classes of the subsystem.***
- *Façade provides a simple default view good enough for most clients.*
- *Facade decouples a subsystem from its clients.*
- *A façade can be a single entry point to each subsystem level. This allows layering.*

Facade Pattern: Participants and Communication

- *Participants: Façade and subsystem classes*
- **Clients communicate with subsystem classes by sending requests to façade.**
- *Façade forwards requests to the appropriate subsystem classes.*
- *Clients do not have direct access to subsystem classes.*

Facade Pattern: Benefits

- *Shields clients from subsystem classes; reduces the number of objects that clients deal with.*

- **Promotes weak coupling between subsystem and its clients.**

- *Helps in layering the system. Helps eliminate circular dependencies.*

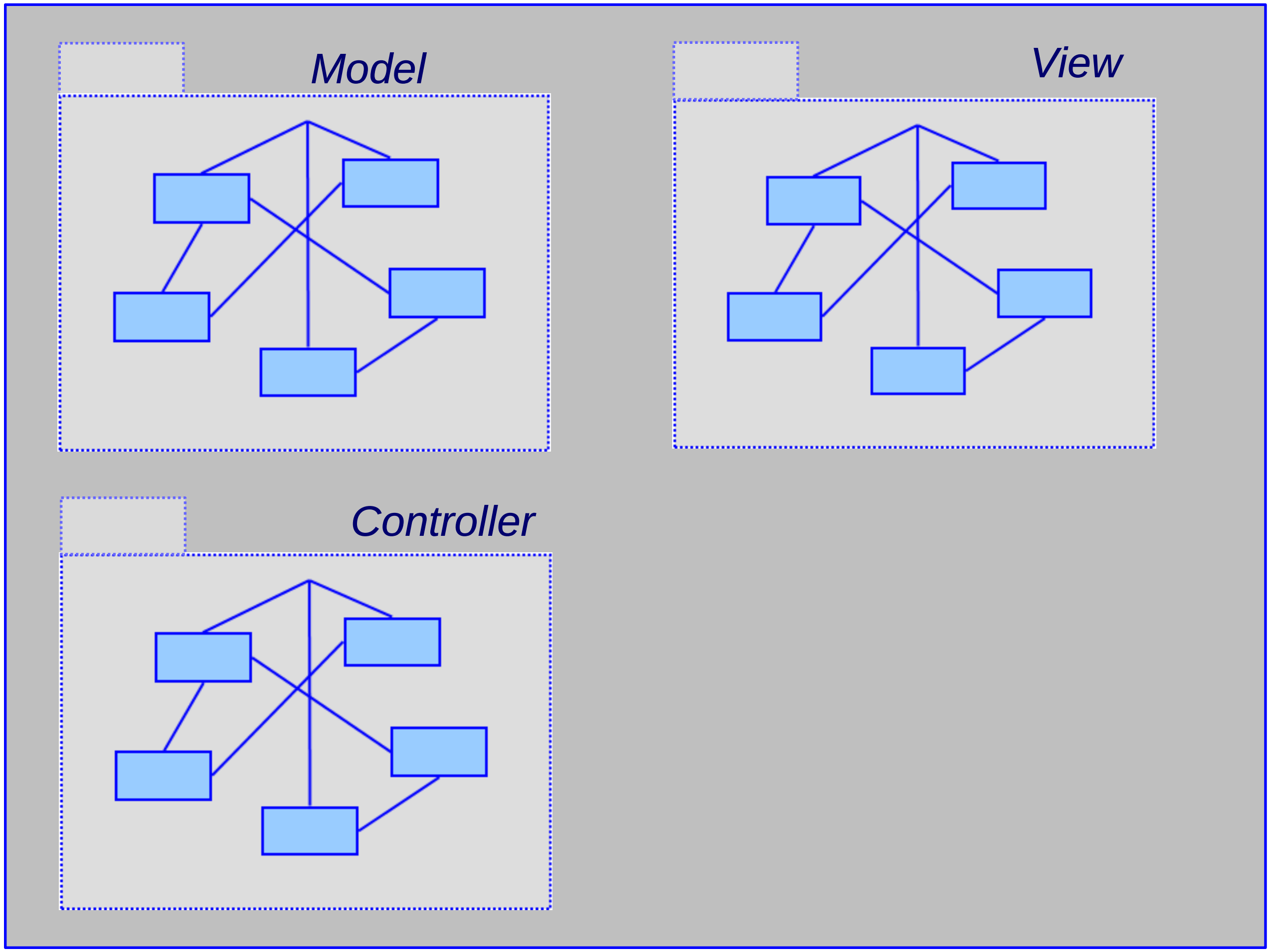
Model View Controller

- **Context (where does this problem occur?)**
  - MVC is an architectural pattern that is used when developing interactive application!
- **Problem (definition of the reoccurring difficulty)**
  - User interfaces change often, especially on the internet where look-and-feel is a competitive issue. Also, the same information is presented in different ways. The core business logic and data is stable.

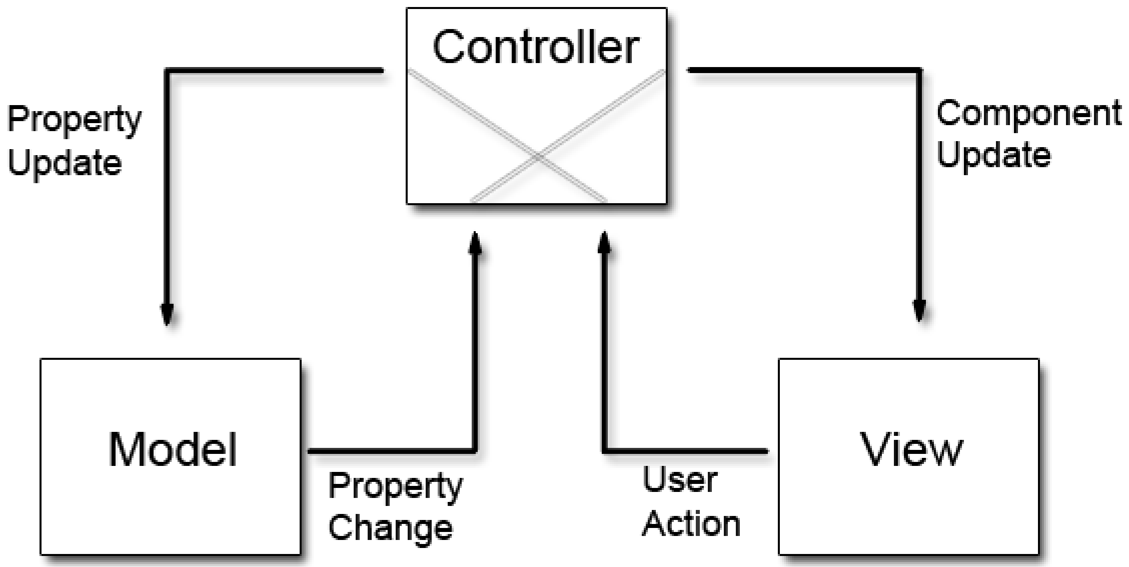
MVC continued

- **Solution (how do you solve the problem?)**
  - Use the software engineering principle of “separation of concerns” to divide the application into three areas:
    - **Model** encapsulates the core data and functionality
    - **View** encapsulates the presentation of the data there can be many views of the common data
    - **Controller** process user input and makes request from the model for the data to produce a new view.

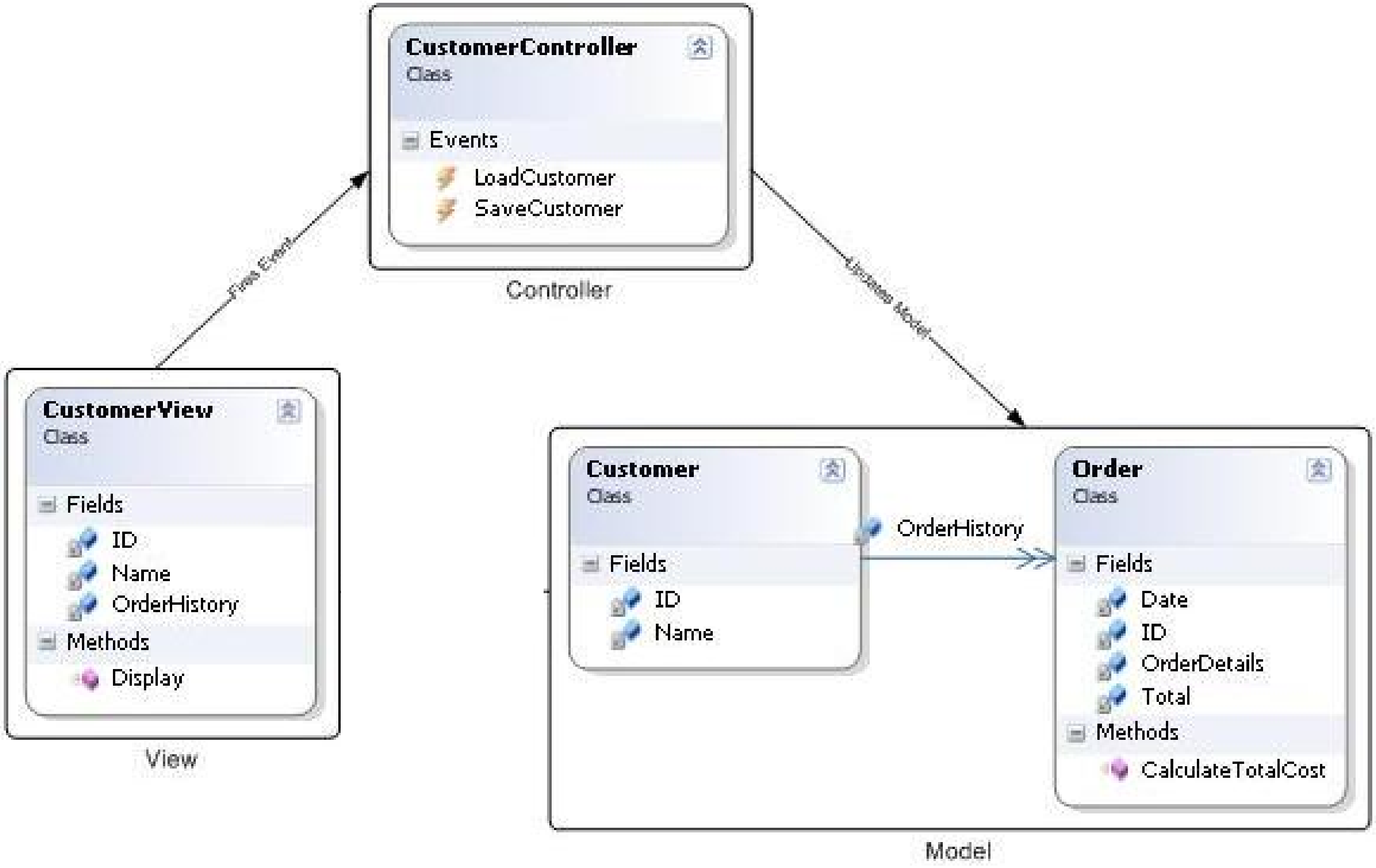
Model View Controller



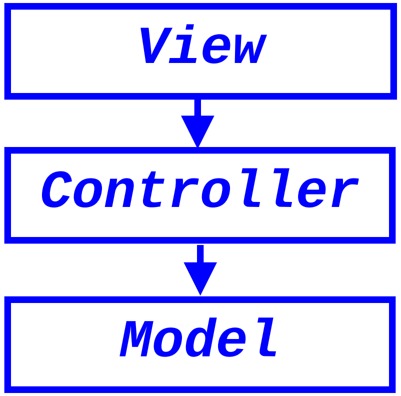
Model View Controller



Model View Controller



Model / view / controller (MVC)



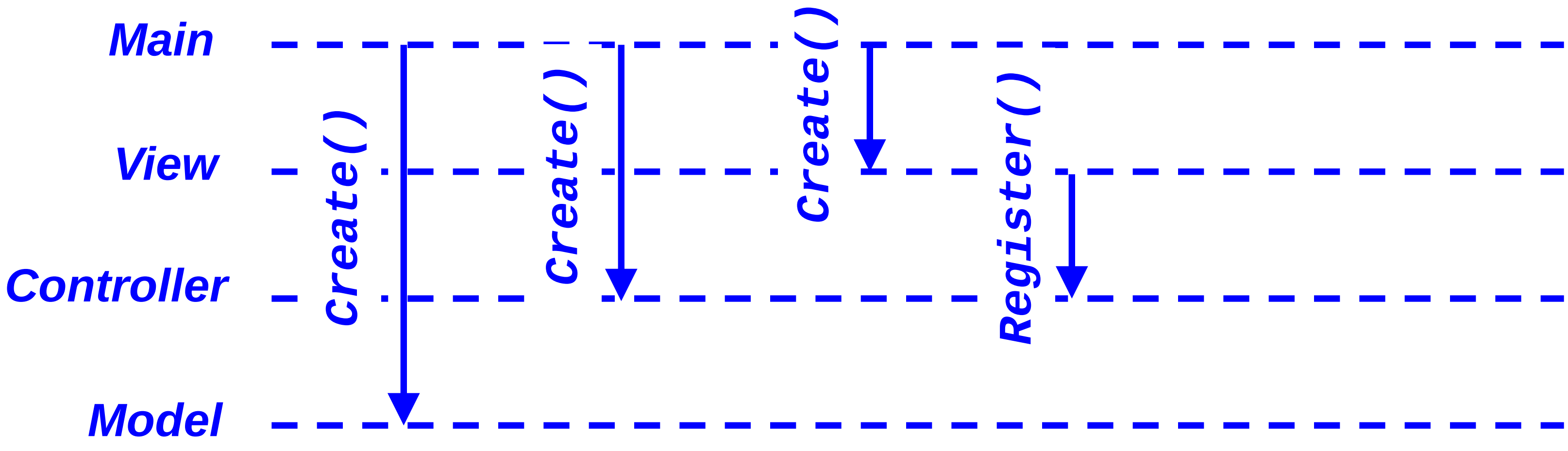
View (displays data)

Controller (mediates)

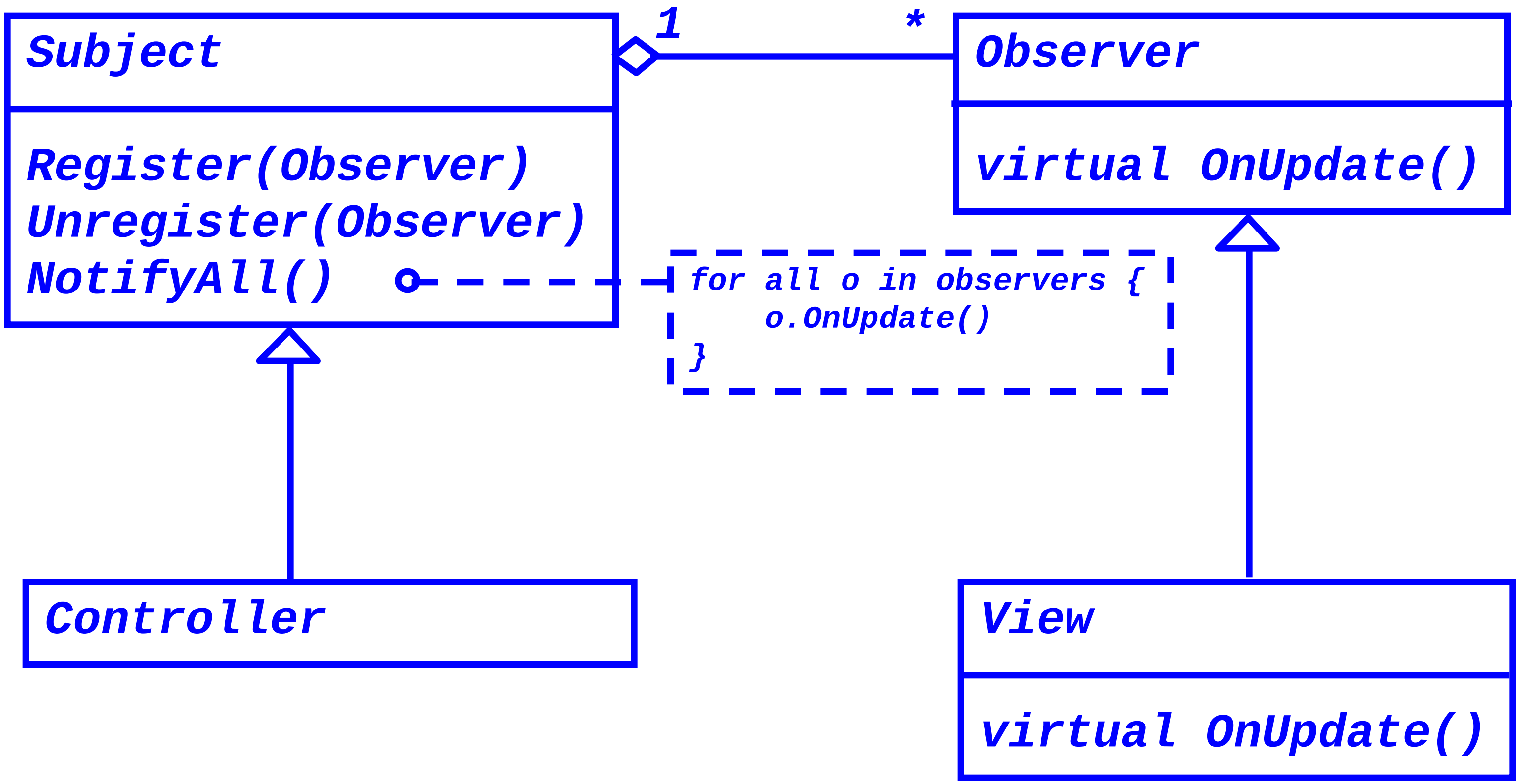
Model (holds data)

```
{
   Model m;
   Controller c(m);
   View v(c);
}
```

calls Register()



MVC uses Observer pattern (cont.)

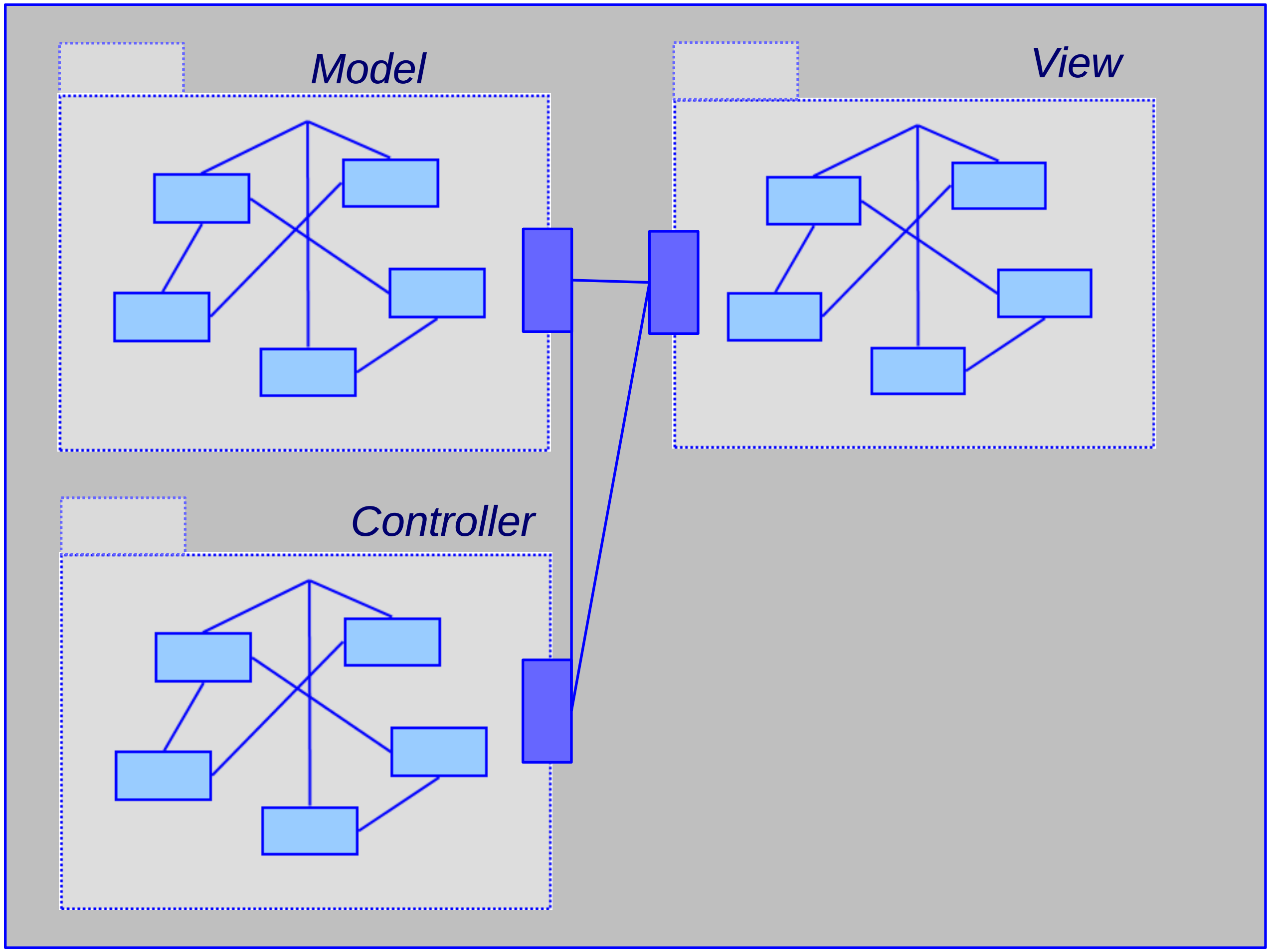


```
for all o in observers {
    o.OnUpdate()
}
```

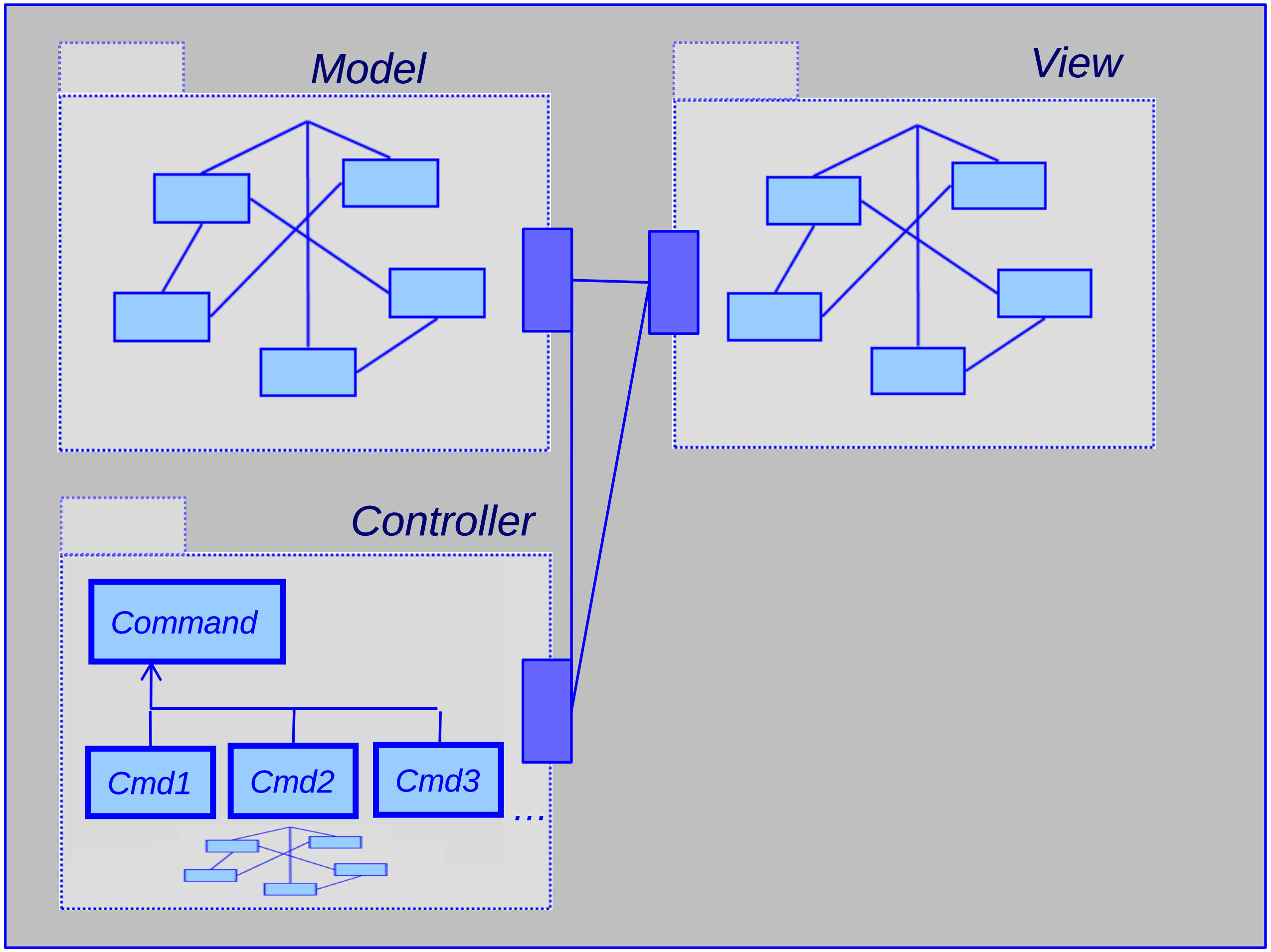
MVC Benefits

- **Clarity of design**
  - easier to implement and maintain
- **Modularity**
  - changes to one don't affect the others
  - can develop in parallel once you have the interfaces between subsystems
- **Multiple views**
  - games, spreadsheets, powerpoint, Eclipse, UML reverse engineering, ….

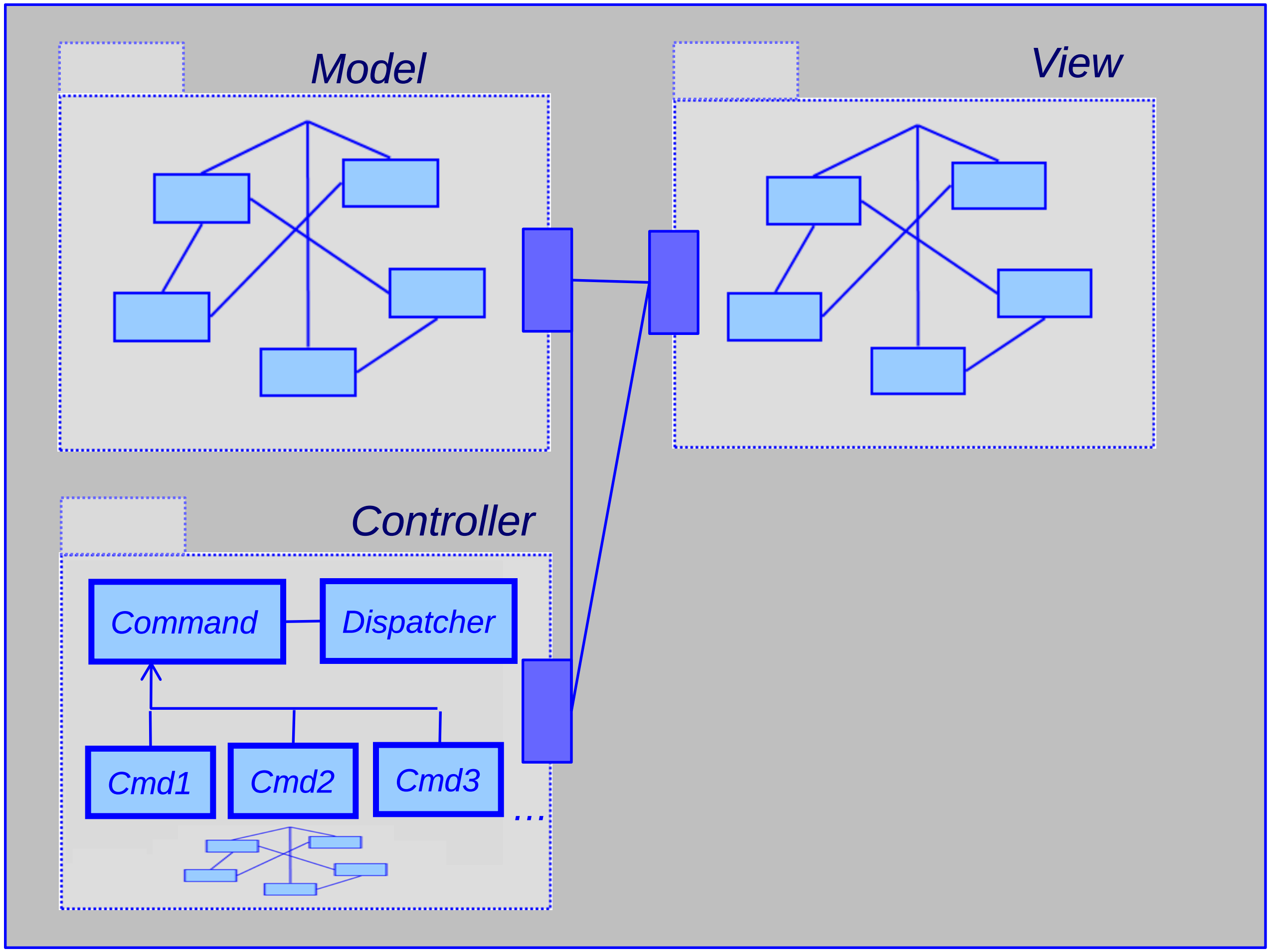
Model View Controller + Facade



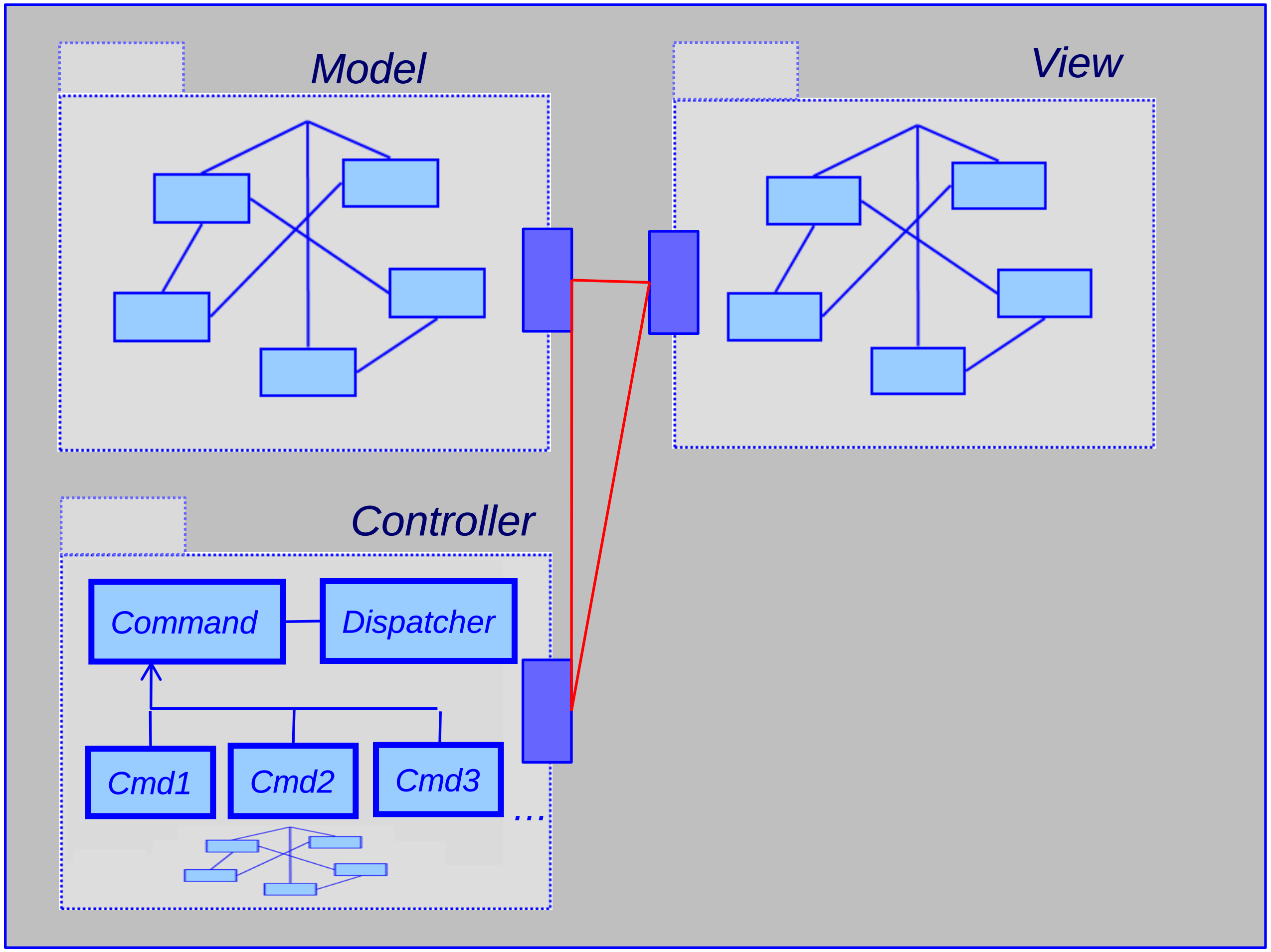
Model View Controller + Façade + Command



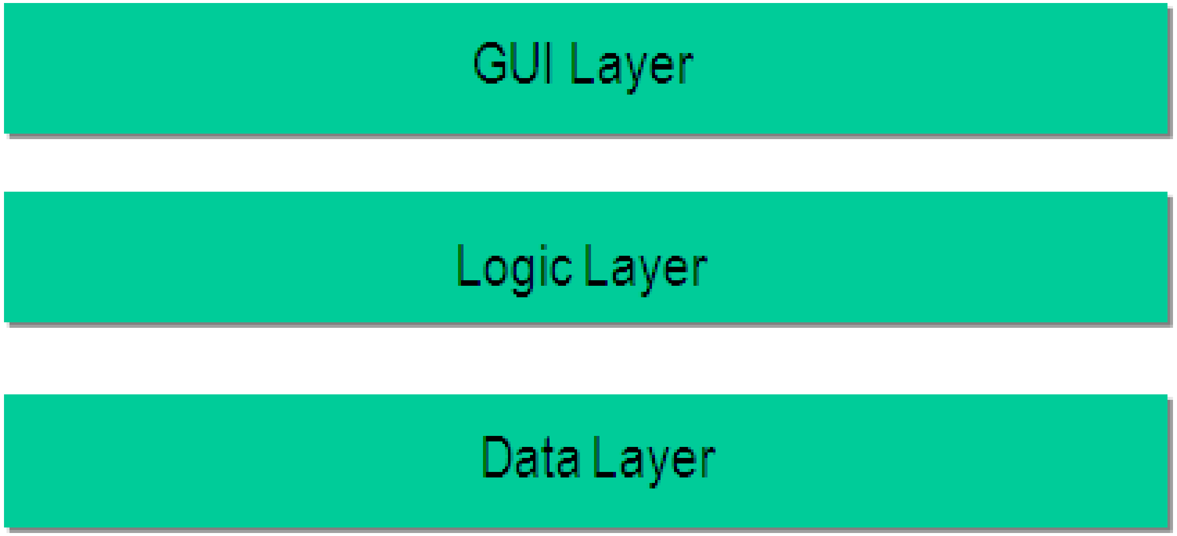
Model View Controller + Façade + Command + Dispatcher



Model View Controller + Façade + Command + Dispatcher + Observer



Program Architecture – 3 tier Architecture



GUI Layer

Logic Layer

Data Layer