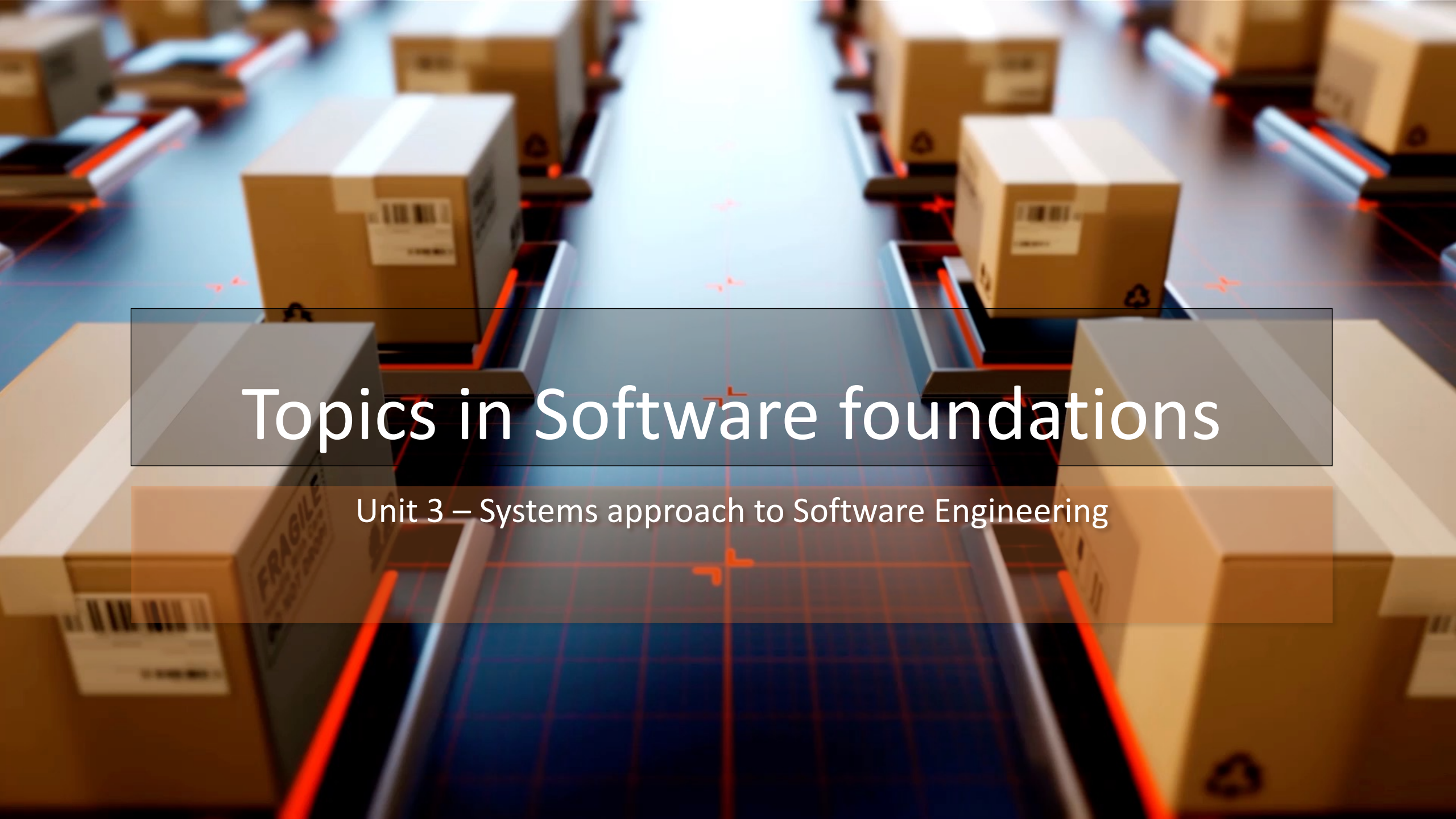


Topics in Software foundations

Unit 3 – Systems approach to Software Engineering

Topics we cover

- Introduction to systems modeling
- Modeling System of Systems (SoS)
- Modeling communications
- Modeling UI systems
- Modeling data access
- Formalizing TS as a modeling language

Modeling UI Systems

Unit 3 – TiSF S'23

Session 5 (2023-04-10)

Recap/continuation from the last class

Semantic of communication model

Outbound ports	A list of (dst, msg) tuples represent N outbound ports in a system A, dst is destination system to which the message type msg is to be sent.
Inbound ports	A list of (src, msg) tuples represent M inbound ports in a system B. The system decides which one to read when, we do not assume any sequencing across these ports.
Ether	We assume the presence of an undefined system which holds the messages in transit for as long as needed – between the time it leaves the outbound port and is received by the inbound port.
Send and receive primitives	We assume the presence of two primitives - sendmsg() and recvmsg() - that enables sending an receiving through the respective ports. Think of (unmodeled) components that support sendmsg() and recvmsg() actions and provide appropriate observables*

*See Forwarder-receiver pattern, *"Pattern-oriented Software Architecture (PoSA)"*

Wiring the components for communication

We extend Y and U for modeling communication

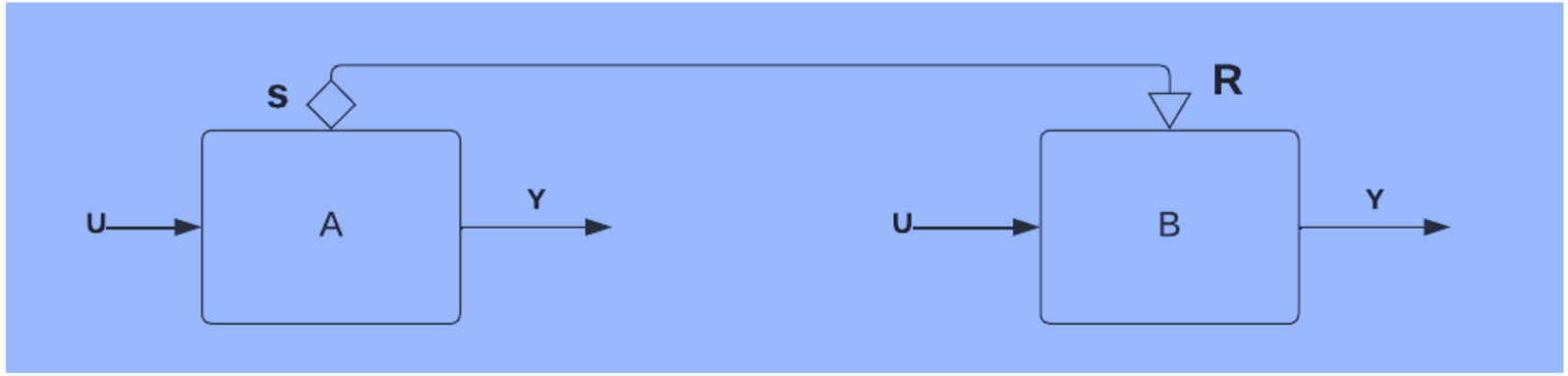
$$Y_{comm} = Y \times P_{out} \qquad P_{out} = \{(dst_1, msg_1), (dst_2, msg_2), \ldots, (dst_m, msg_m)\}$$

$$U_{comm} = U \times P_{in} \qquad P_{in} = \{(src_1, msg_1), (src_2, msg_2), \ldots, (src_n, msg_n)\}$$

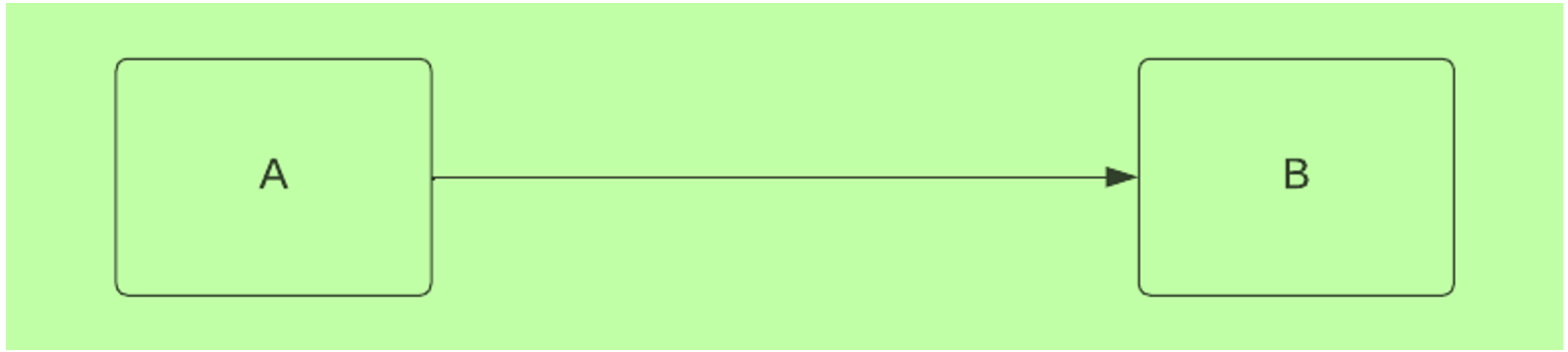
Note that A or B can be a message handler (or message queue) and not the ultimate destination (or source).

Notational convenience for 1-way communication

Instead of

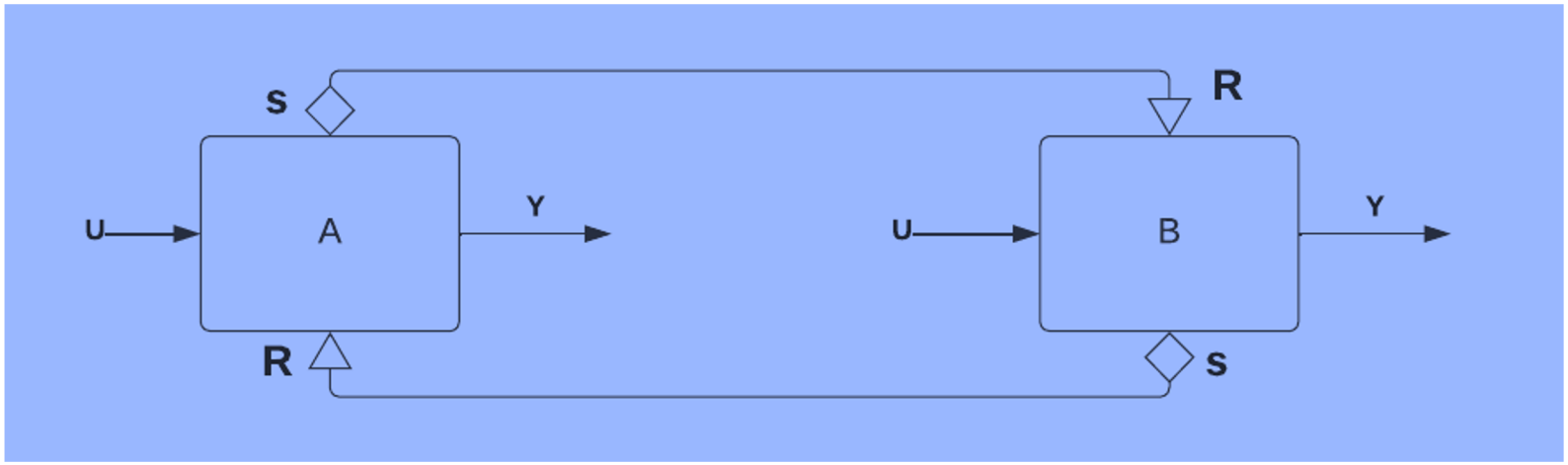


We will use

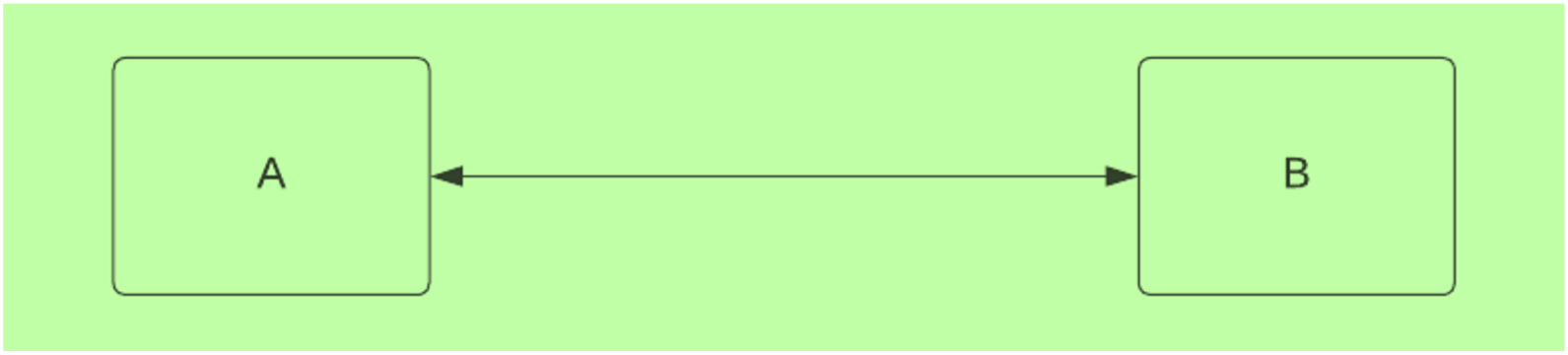


Notational convenience for 2-way communication

Instead of



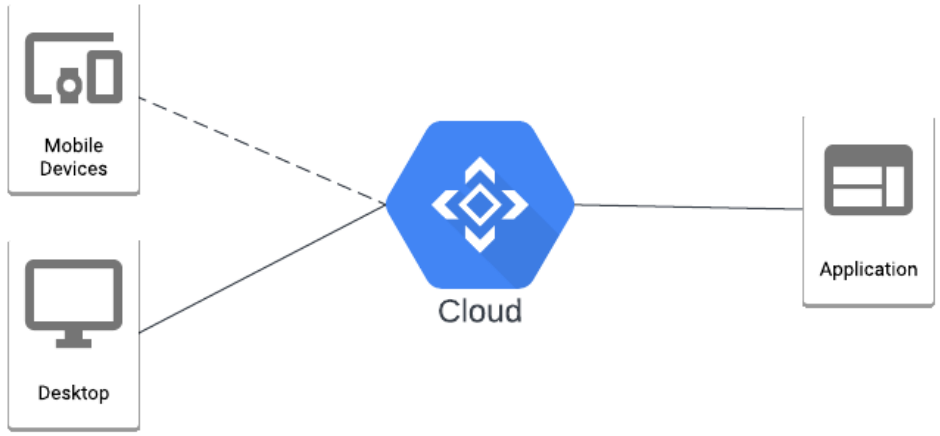
We will use



UI Systems

User interaction needs are diverse

- Users need different things from the same application
- Client capabilities are different – processing, display
- Network capabilities between client and backend vary greatly



Why special attention to UI Systems?

Most applications have some kind of user interface, a large number have rich user interfaces

Applications with user interfaces bring in multiple common concerns that models need to account for

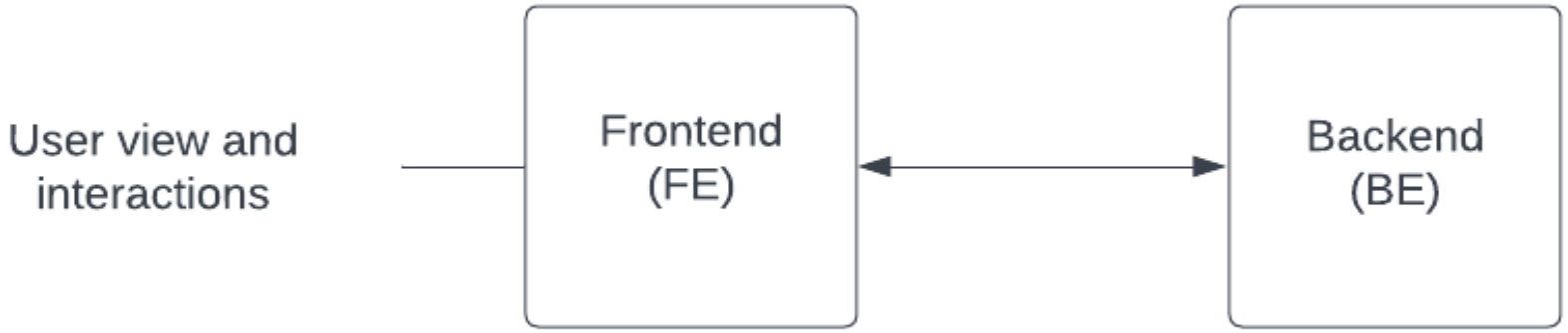
We want to discover common modeling patterns that are applicable across class of UI systems

UI Systems – modeling goals

- User experience concerns should be isolated to a few systems
- Support multiple UI systems for the same application
- Devise a few reference models that address common UI system concerns

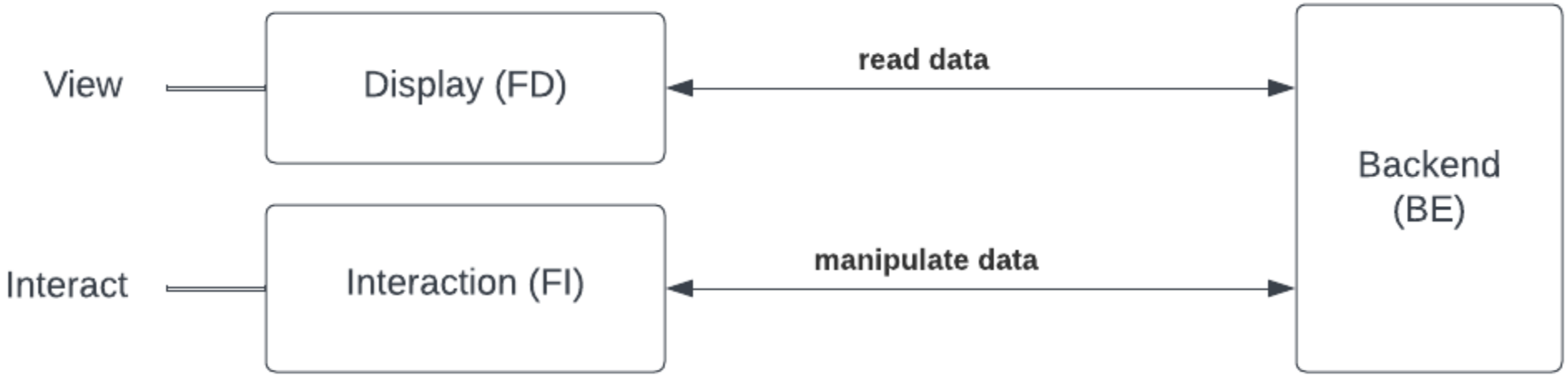
Refinement #1 (separate user facing system)

- All user-facing behavior is handled by frontend (FE) system
- FE sends commands to backend (BE), BE processes the command and responds to FE
- Systems communicate over the underlying communication protocol (HTTPS, RTP, WebSocket, etc.)



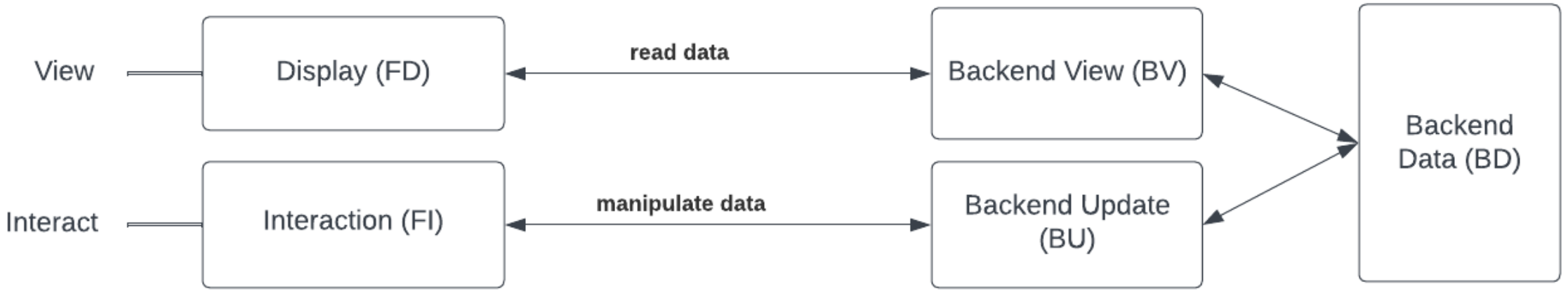
Refinement #2 (separation of concerns in FE system)

- Display system handles data views
- Interaction system handles user inputs
- Each system works with the common backend system through separate interfaces



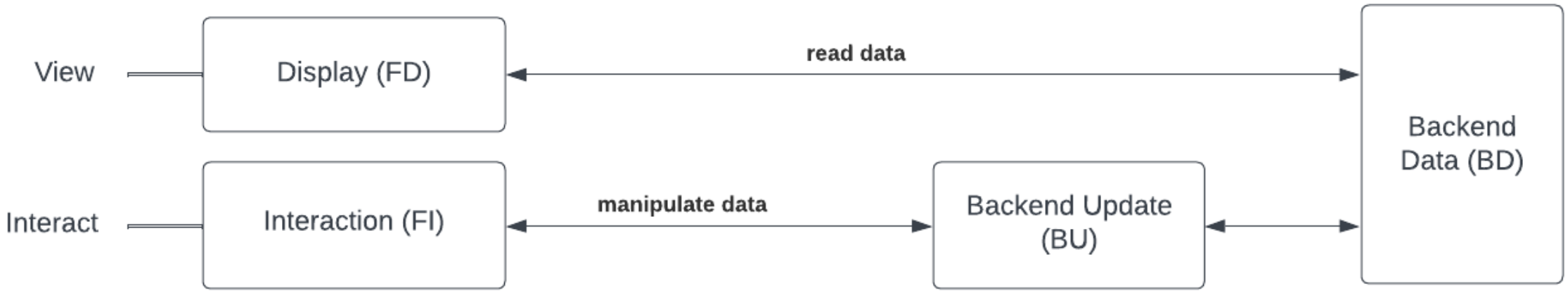
Refinement #3 (separation of concerns in BE system)

- View system is designed to deal with large data sets
- Data modification system is designed to deal with complex business rules for access control and data consistency
- Both need access to data which is handled by the Data access system



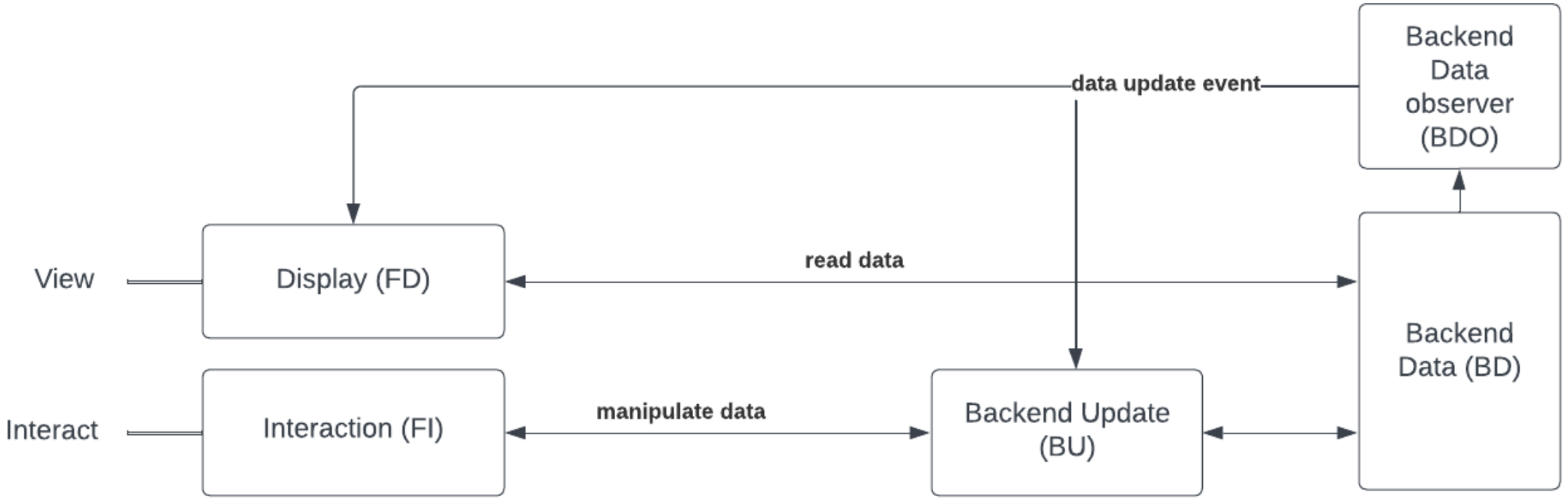
Refinement #4 (simplify backend systems)

- Read requests can be directly handled by the data access system if these are straightforward queries for data



Refinement #5 (handle data refresh)

- The data modified through data modification system or any external system may cause the view to become outdated.
- Observer system monitors for data changes and lets the display handling system know



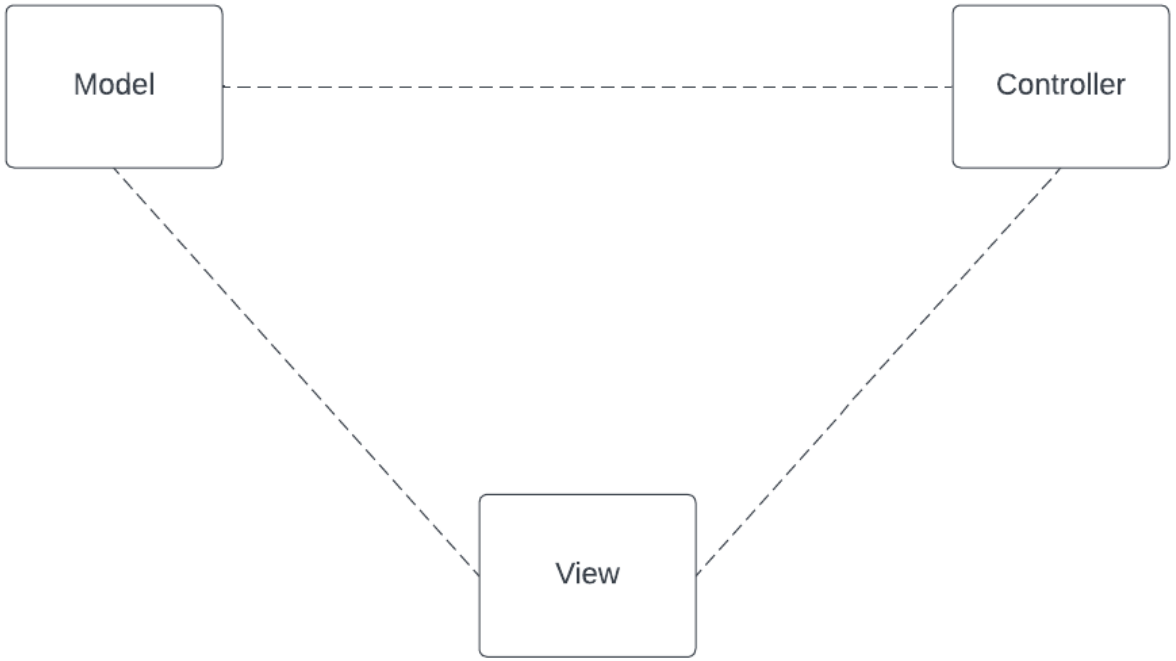
MVC: Model-View-Controller (A popular architectural/GUI* pattern)

- Many variations have been proposed and used:
  - Presentation–abstraction–control
  - Model–view–adapter
  - Action–domain–responder

- Similar approach applied for designing user interfaces:
  - Model–view–view model
  - Model–view–presenter

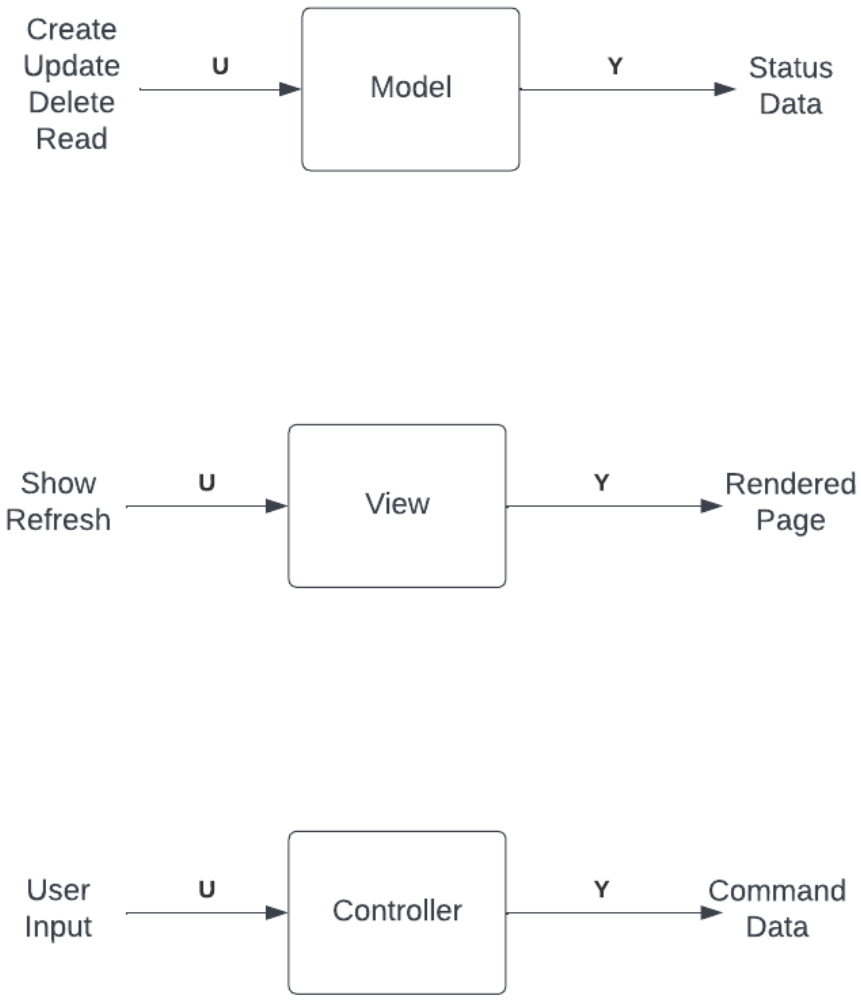
*It is debatable whether MVC is an application architecture pattern or just a GUI pattern. See this.

MVC components

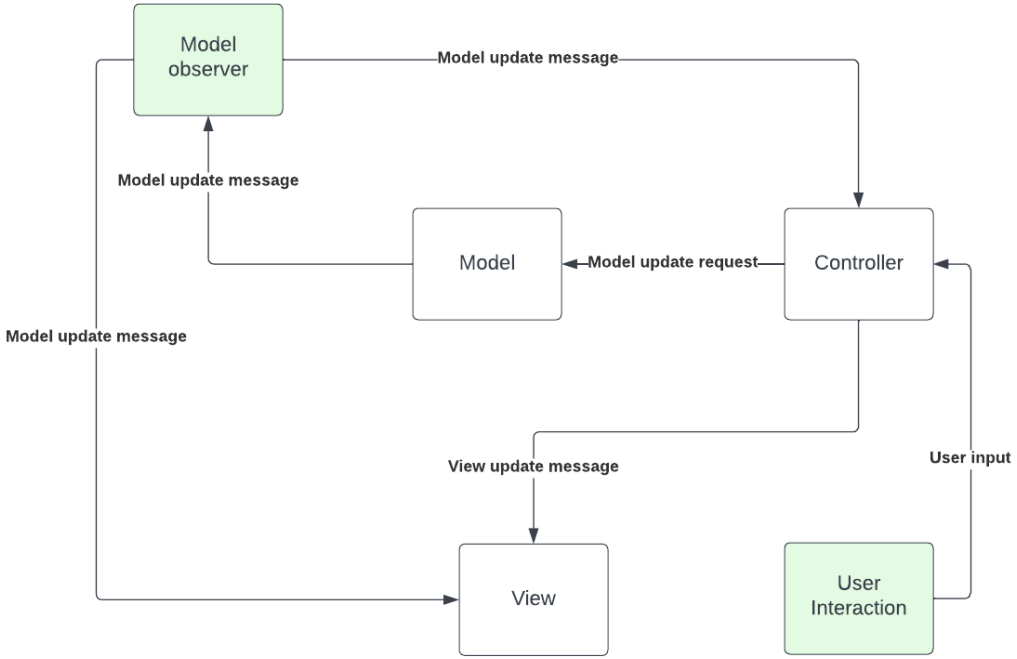


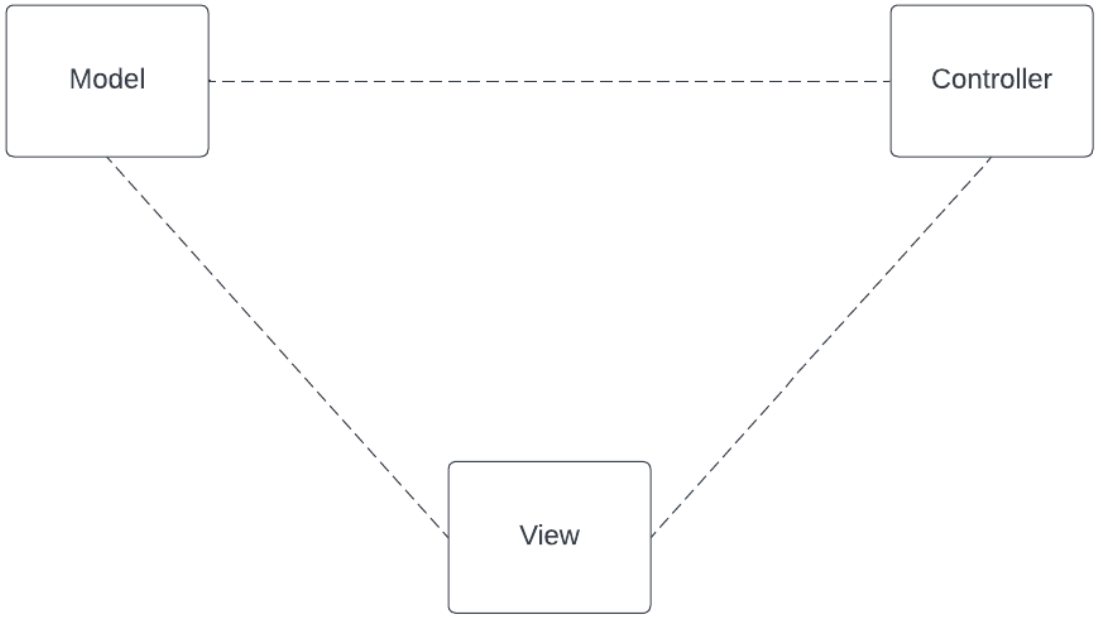
- **Model** – Responsible for representing the domain data that the user wishes to see and manipulate
- **View** - Responsible for display of the model in user-friendly manner.
- **Controller** - Responsible for updating the view or the model based on user interactions

Systems view of MVC

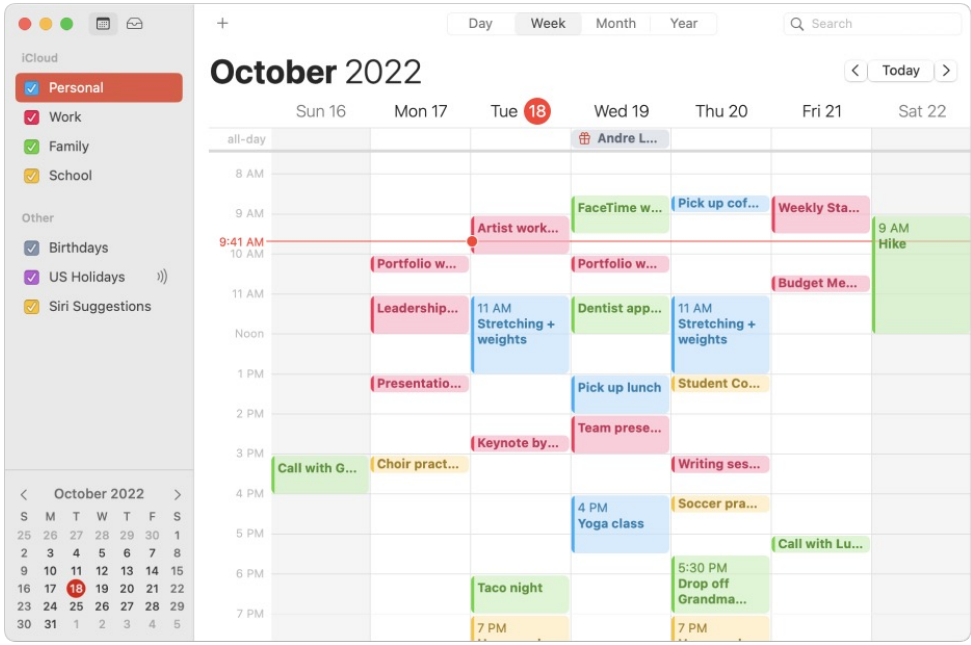


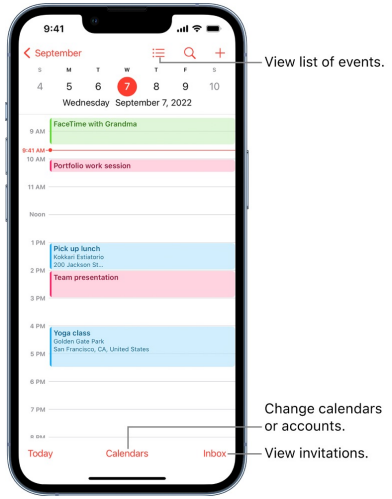
Systems view of MVC

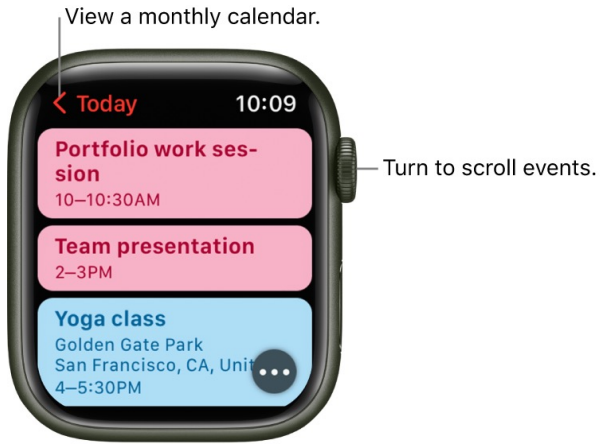




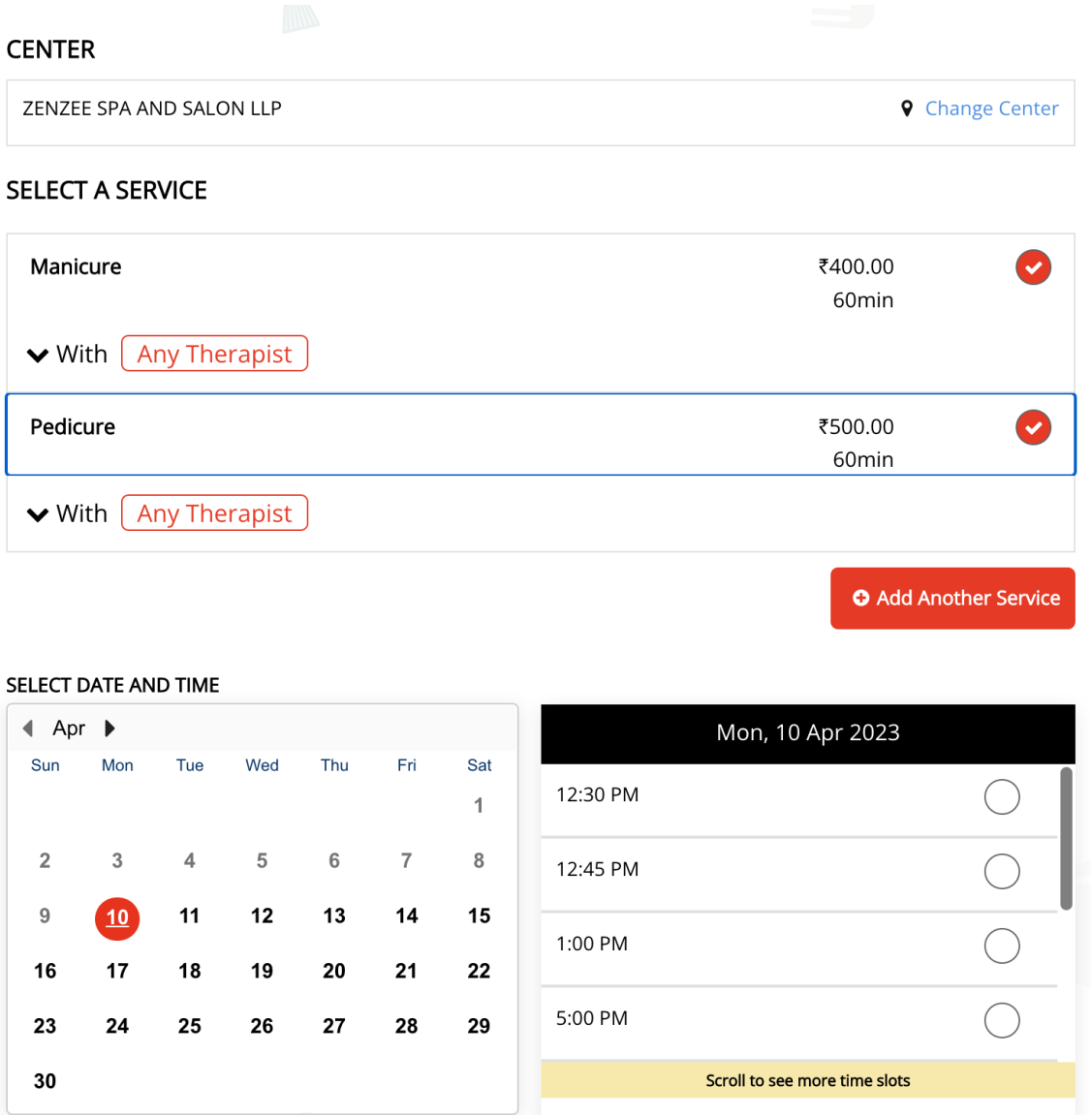
Different types of UI for the same application







Different kinds of data in same UI



Common concerns we need to model for

Richness of UI	Quality of client-server connection	Quality of client device	Patterns of application access
• There may be multiple data sources that compose a page • A page component may include data from multiple sources	• Low bandwidth • High latency • Frequent connection drops	• Low processing power (low-end android phone) • Low display capabilities (watch)	• The user may use multiple devices to access the system simultaneously • A large number of users access the system simultaneously

Classwork

Recommend a model (or changes to MVC model) to handle each of the problems listed in the last slide and give your reasons why the change is required and how the change addresses the problem
• Richness of UI • Quality of client connection • Quality of device • Patterns of application access

Questions?