KAIST

Spring 2025

CS30500:
Introduction to Software Engineering

Lecture #16

Prof. In-Young Ko

School of Computing

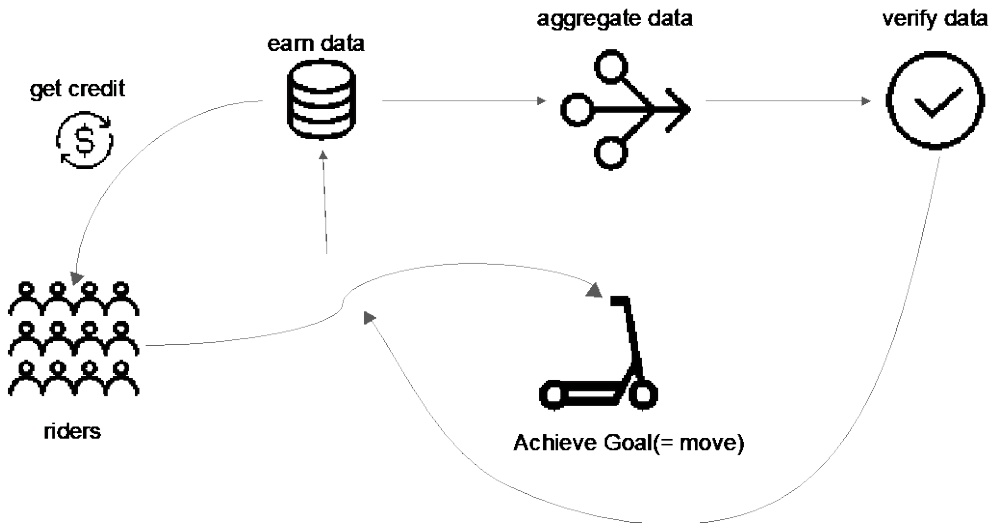
Software Design

- Transforms requirements into the form that can help developers to build a software system effectively
- Design classification:
  - **Architectural** design – defines relationships among the major software structural elements
  - **Component-level** design – transforms structural elements into procedural descriptions of software components
  - **Data/Class** design – transforms analysis classes into implementation classes and data structures
  - **Interface** design – defines how software elements, hardware elements, and end-users communicate

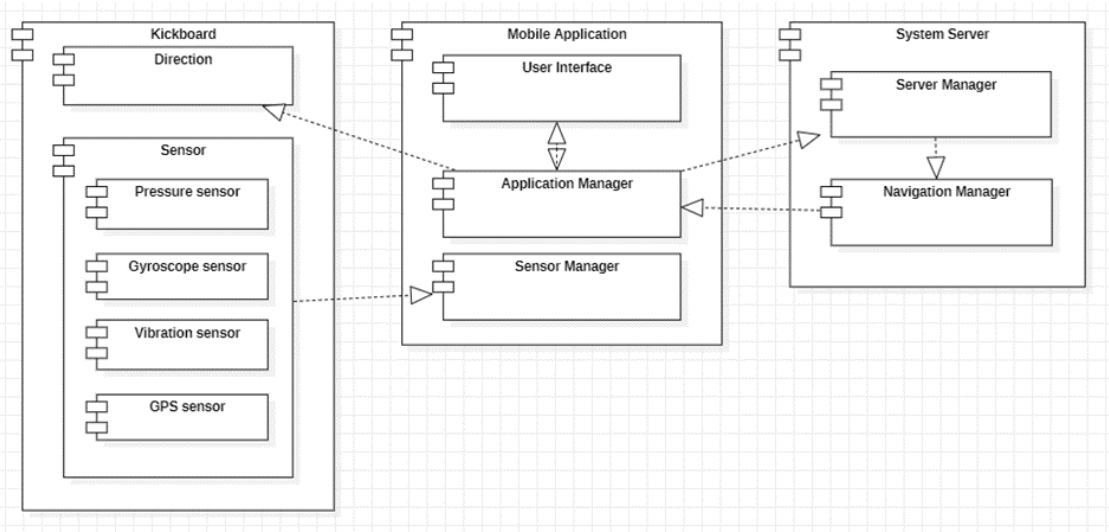
[PrMa20]

What are the things to design? (1/2)

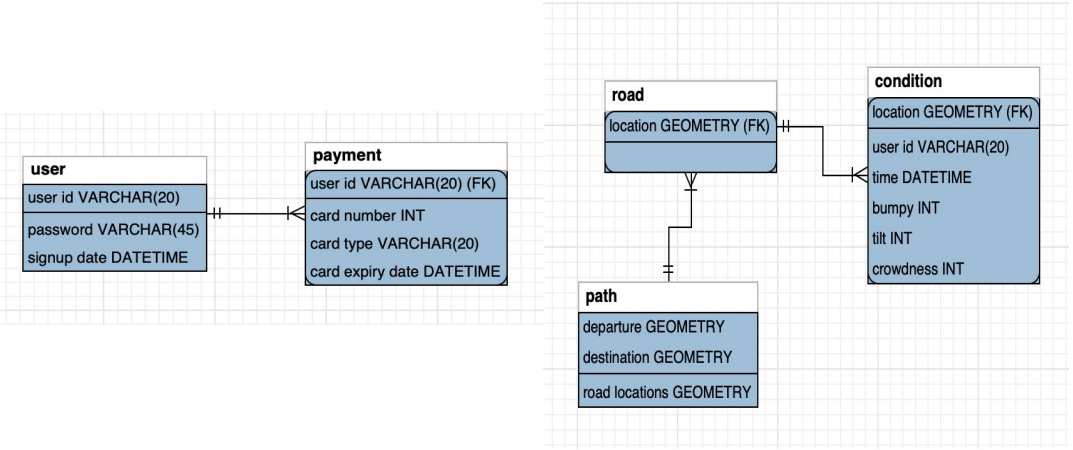
Conceptual Architecture



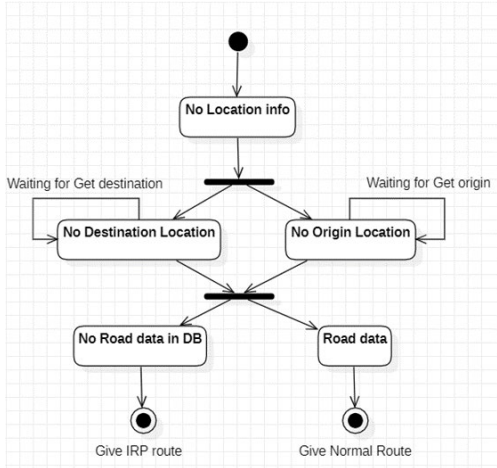
System Architecture (Components)

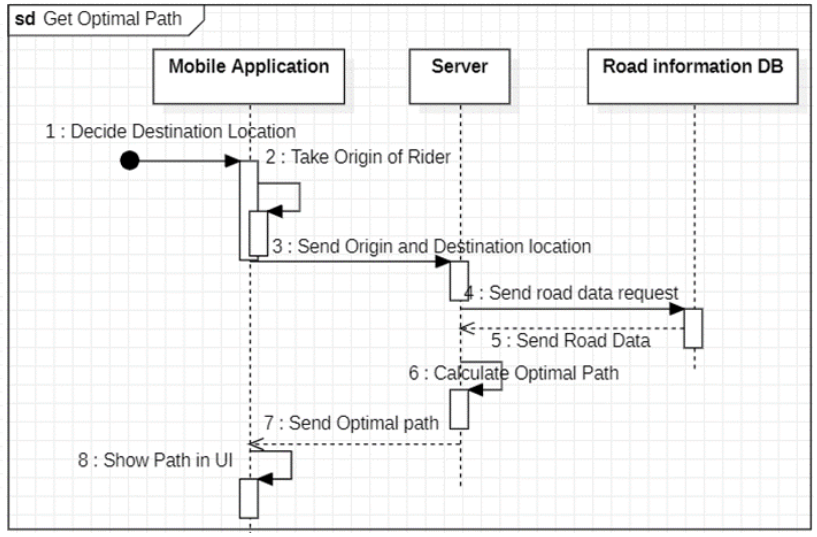


Data



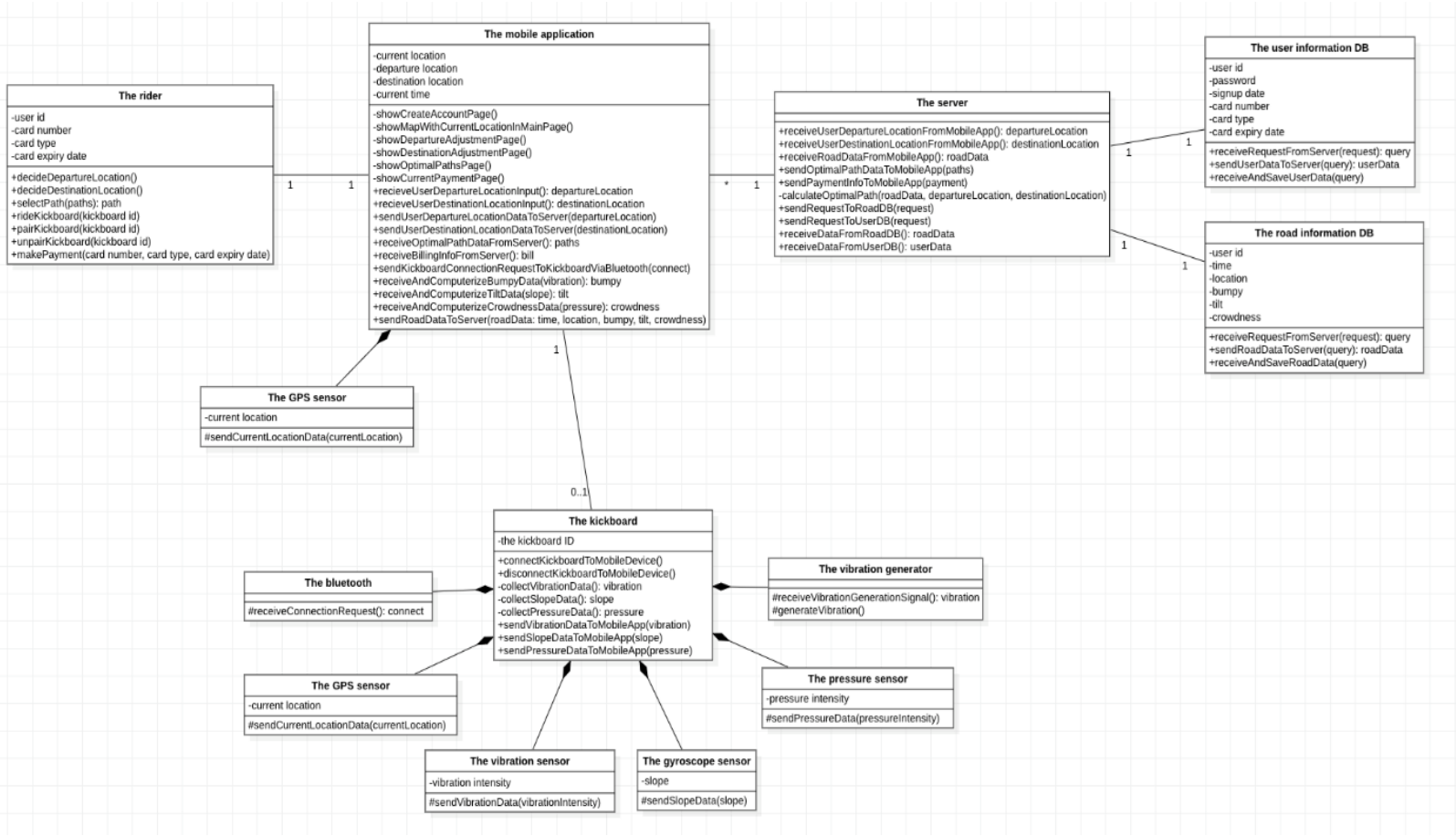
Flows & Interactions



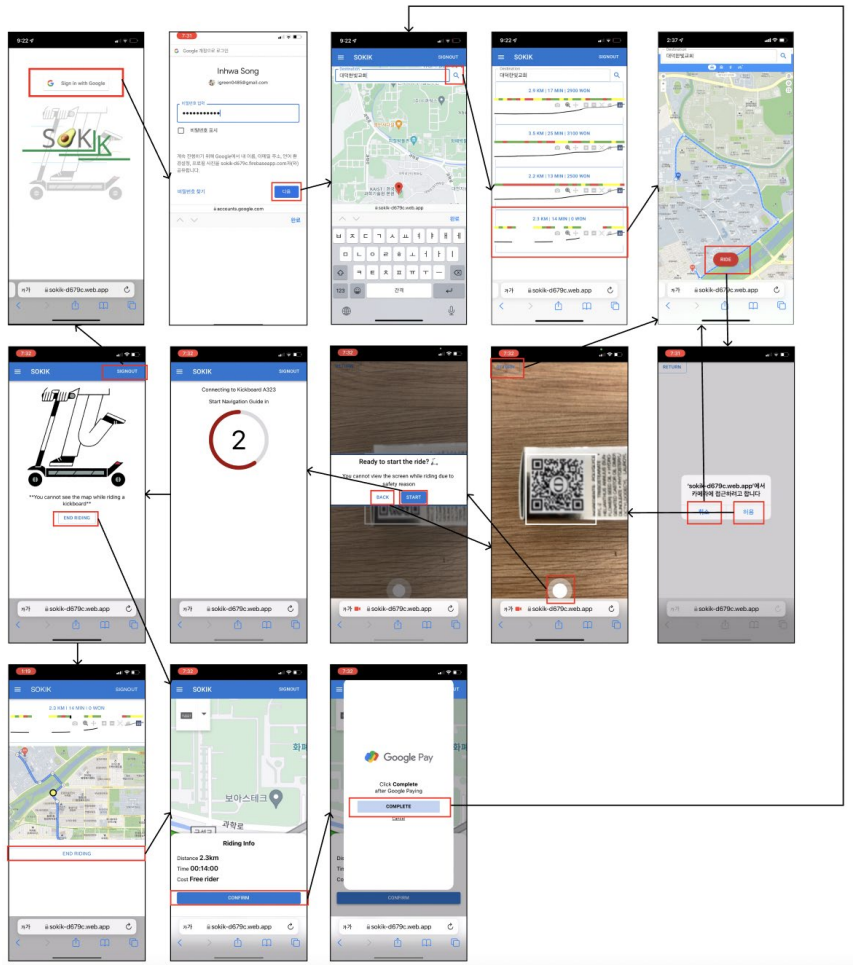


What are the things to design? (2/2)

Implementation Classes



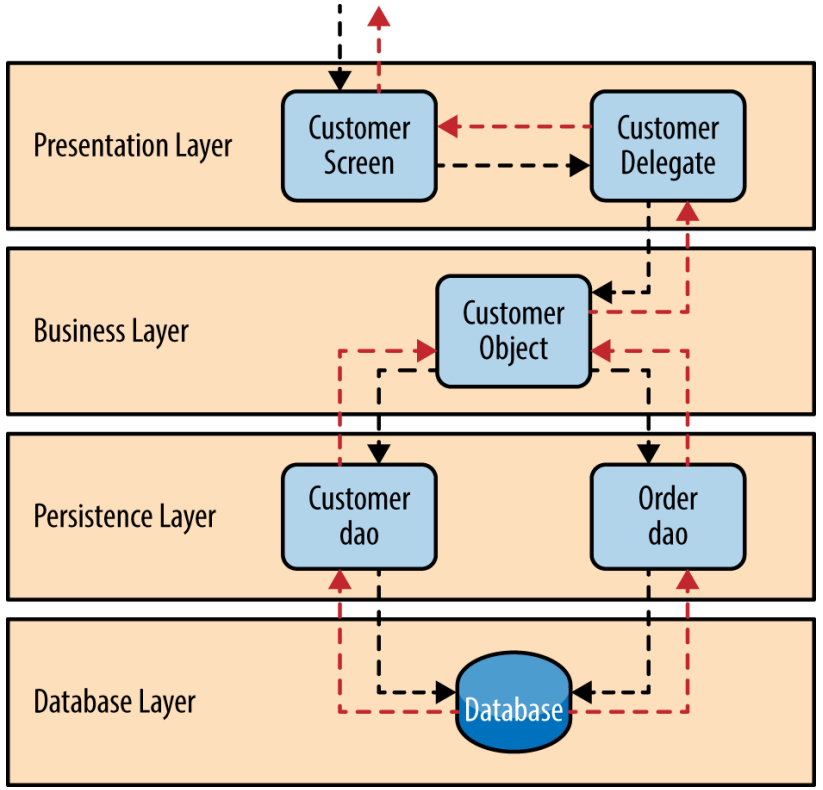
User Interface



ARCHITECTURE PATTERNS

Architectural Design

- An architectural model is derived from three sources: [PrMa20]
  - Information about the application domain for the software to be built
  - Specific requirements model elements such as data flow analysis classes and their relationships (collaborations) for the problem at hand
  - Availability of architectural patterns and styles
- Goal
  - To develop a modular program structure
  - To represent the control relationships between modules
  - To integrate the program structure and data structures
  - To define interfaces

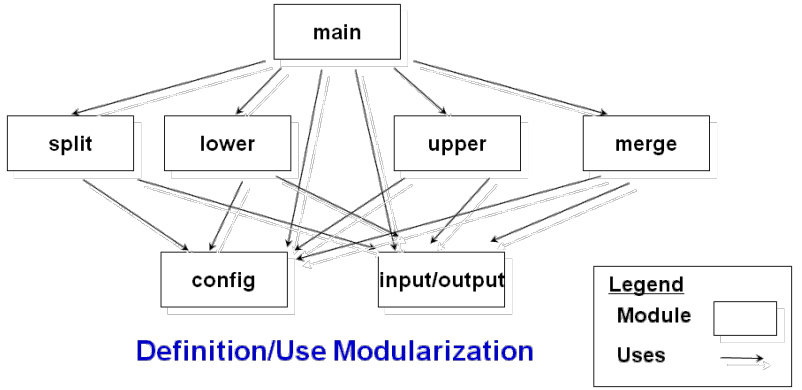


https://towardsdatascience.com/software-architecture-patterns-98043af8028

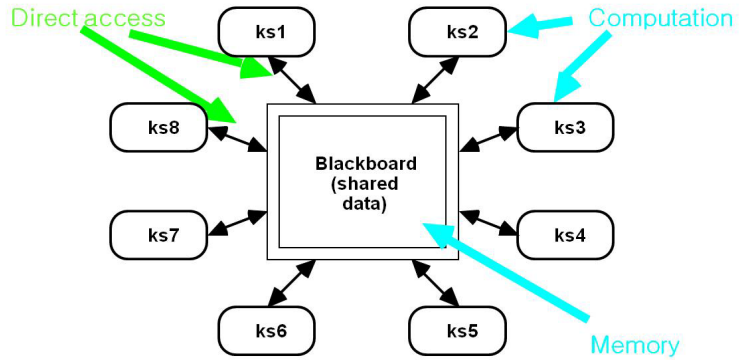
Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

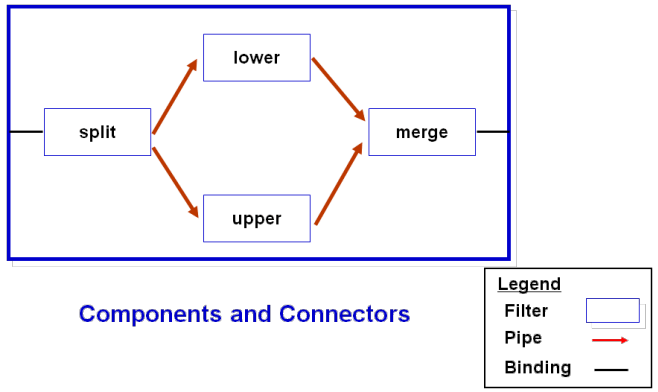
Architectural Styles (Patterns)

- Common and reusable patterns of software architecture observed in different systems
- Specialization of element and relation types, together with a set of constraints
- Represents common properties
- Can be associated with different architectural views
- Most widely used architectural patterns
  - Layered
  - Client-server
  - Pipe-filter
  - Broker
  - Event-bus



Definition/Use Modularization



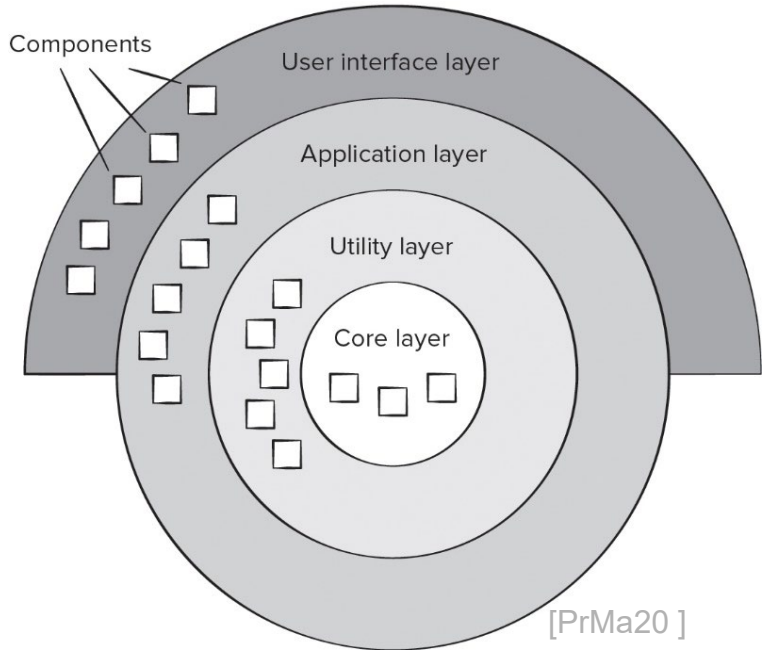


Components and Connectors

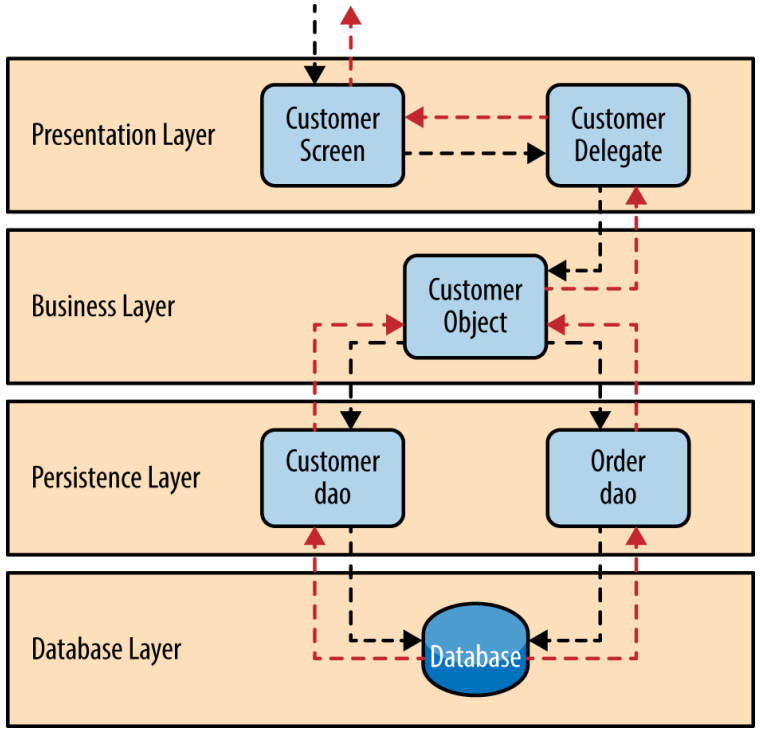
Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Layered Pattern (1/2)

- Decomposed into groups of subtasks, each of which is at a particular level of abstraction
- Each layer provides services to the next higher level
- Common examples:
  - Network systems
  - E-commerce



[PrMa20]



https://towardsdatascience.com/software-architecture-patterns-98043af8028

Layered Pattern (2/2)

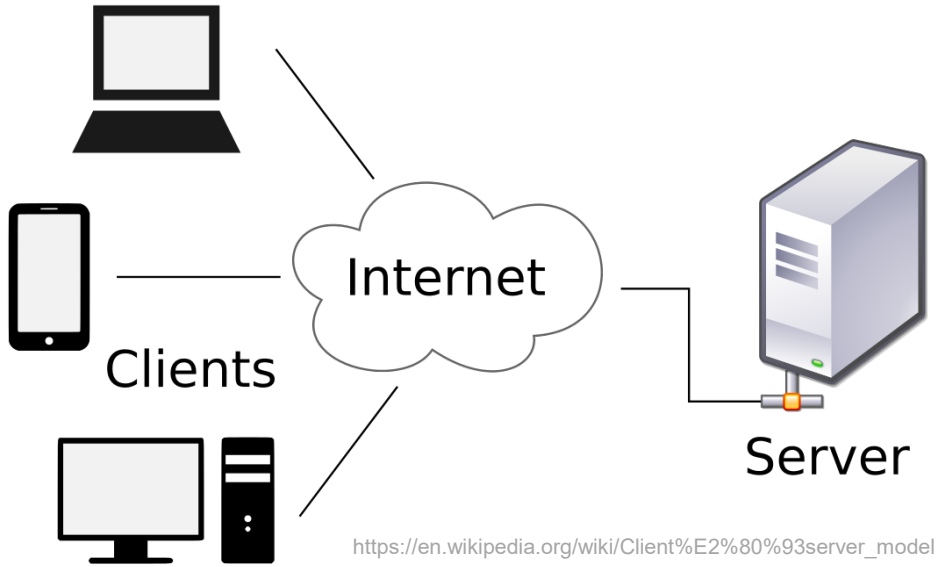
- Pros
  - Separate of concern – we only need to consider a smaller scope in each layer, which makes the problem much more straightforward
  - More testable – each layer has less case to test and thus more testable
  - Isolation – changes in one layer will not affect downstream layers
  - Changeability – if you are not satisfied with the implementation of one layer, you can replace it with another layer, as long as they implements the same interface
- Cons
  - Management cost if there are too many layers
  - The performance is getting slower as more and more layers added

https://enqueuezero.com/layered-architecture.html

Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Client-Server Pattern (1/2)

- The *server* component listens *client* requests, provides services to multiple client component
- Server does not need to know clients
- Online applications such as Web, email, banking, etc.

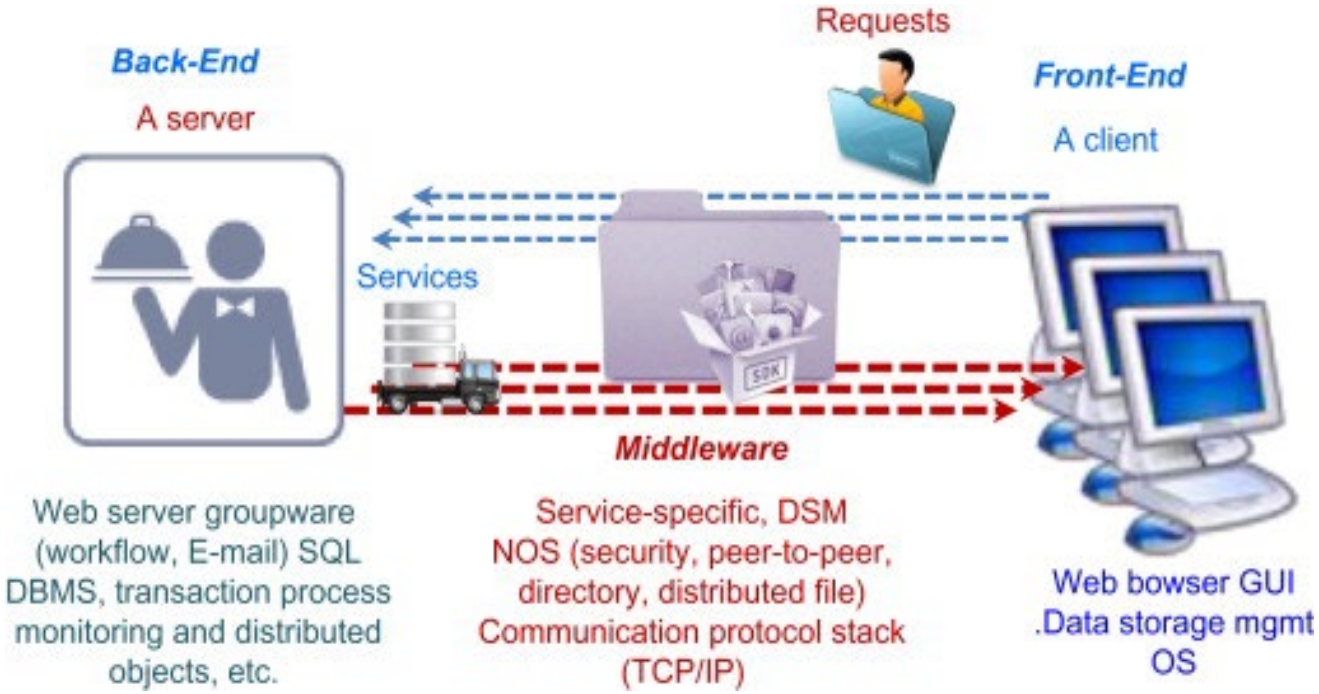


https://en.wikipedia.org/wiki/Client%E2%80%93server_model

Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Client-Server Pattern (2/2)

- A producer/consumer computing architecture
  - A server acts as the *producer*, providing services such as application access, storage, file sharing, printer access, …
  - A client acts as the *consumer*
- Pros and Cons
  - Scalable?
  - Centralized control
  - Cost? – high maintenance
  - In case of failure?

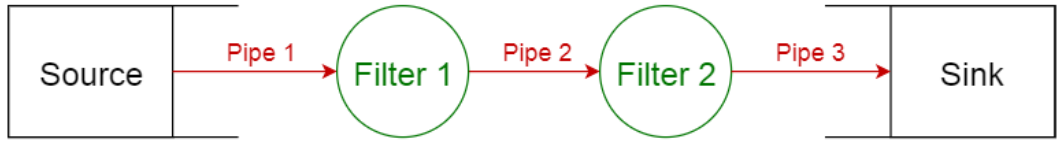


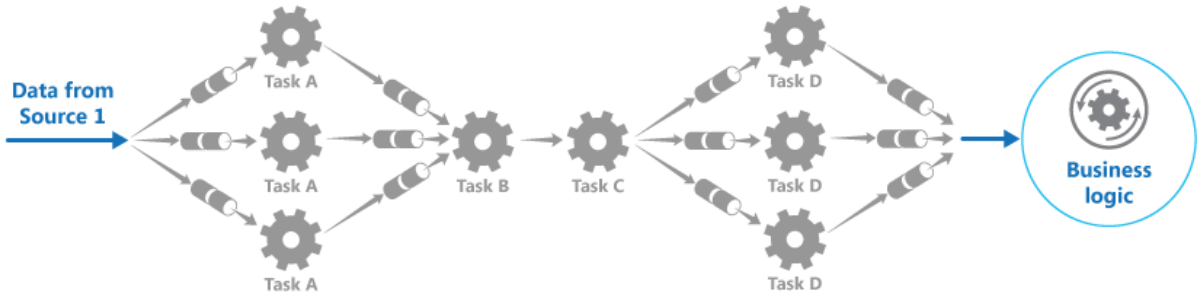
https://www.sciencedirect.com/topics/computer-science/client-server-architecture

Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Pipe and Filter (Data Flow) Pattern

- The structure of a system that produces and processes a stream of data
- Each processing unit is enclosed within a *filter* component
- Data to be processed is passed through *pipes*
- e.g., Compiler, workflow machine,.. IoT, Digital Twins in CPS,..



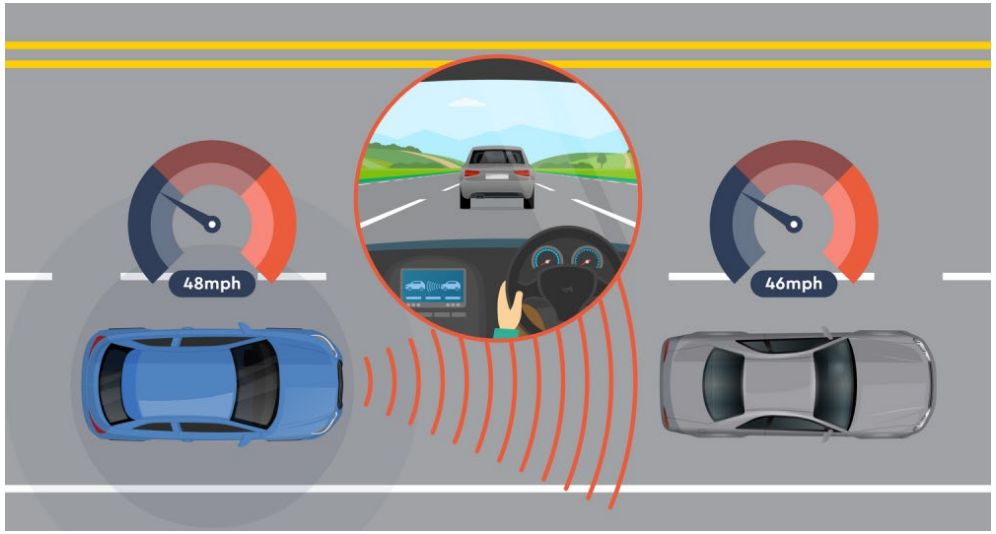


Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

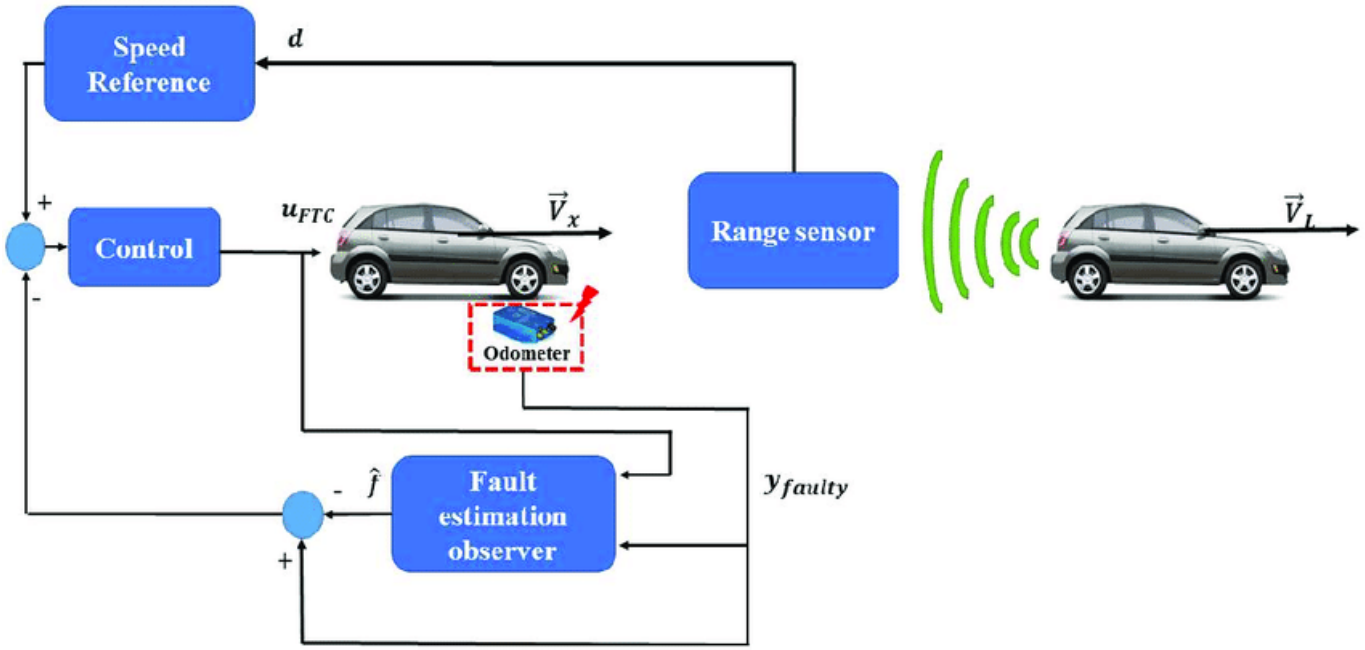
Pipe and Filter Example

Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

- Automated Cruise Control System
  - Maintain a speed set by the driver
    - Sensor, processing unit, actuator
  - How to measure the current speed?



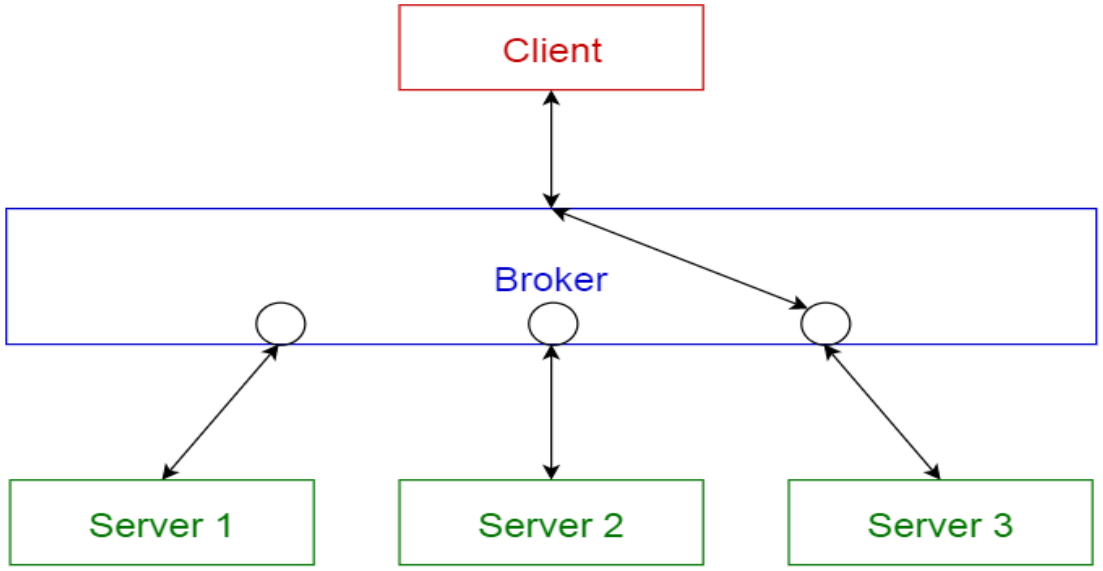
https://www.thewindscreenco.co.uk/adas-guide/adaptive-cruise-control-acc/



https://www.researchgate.net/figure/Adaptive-Cruise-Control-ACC-fault-tolerant-paradigm_fig1_325702711

Broker Pattern

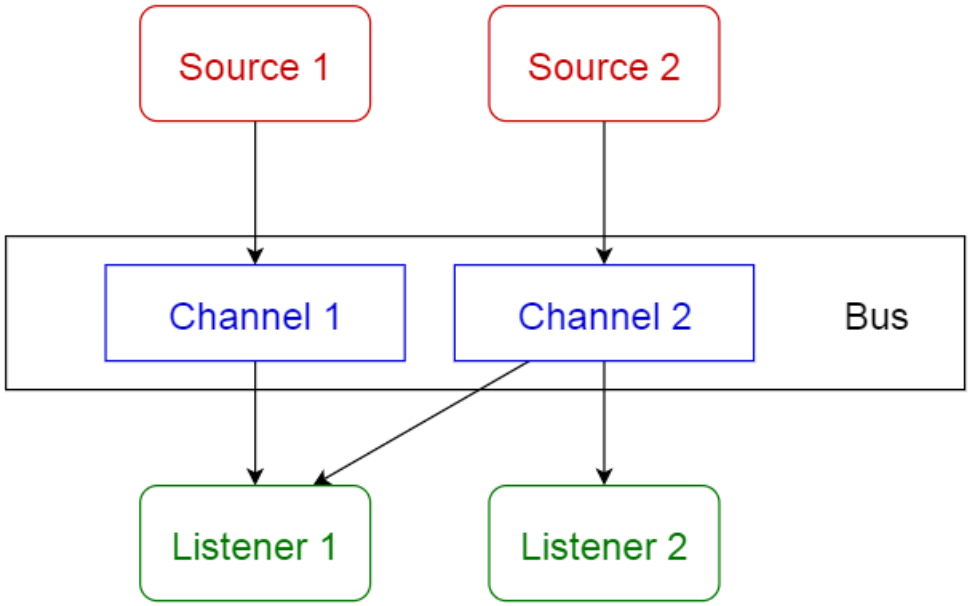
- Structure of a distributed system with decoupled components
- *Broker* component is responsible for the coordination and communication among components
- *Servers* publish their capabilities to a broker
- *Clients* request a service from the broker, and the broker redirects the client to a suitable service
- Message brokers:
  - Apache ActiveMQ
  - JBoss Messaging

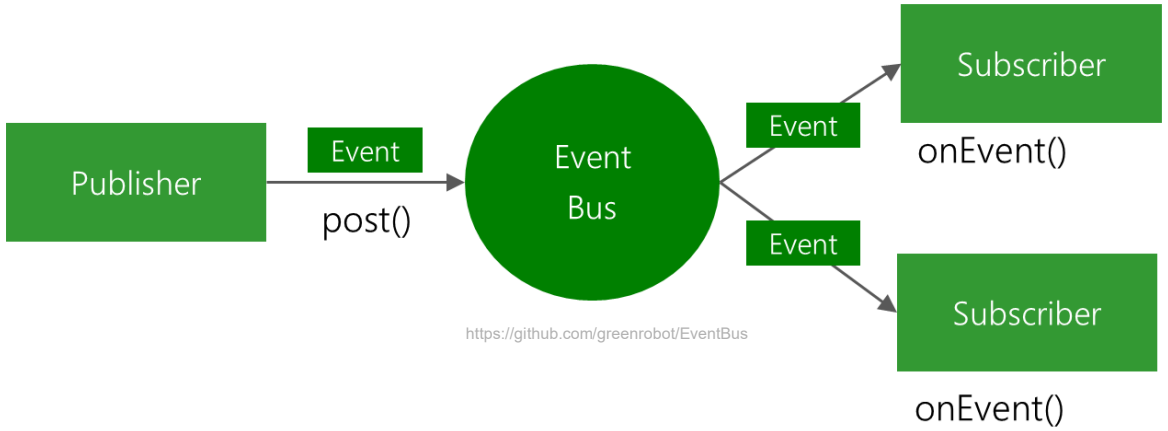


Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Event-bus Pattern

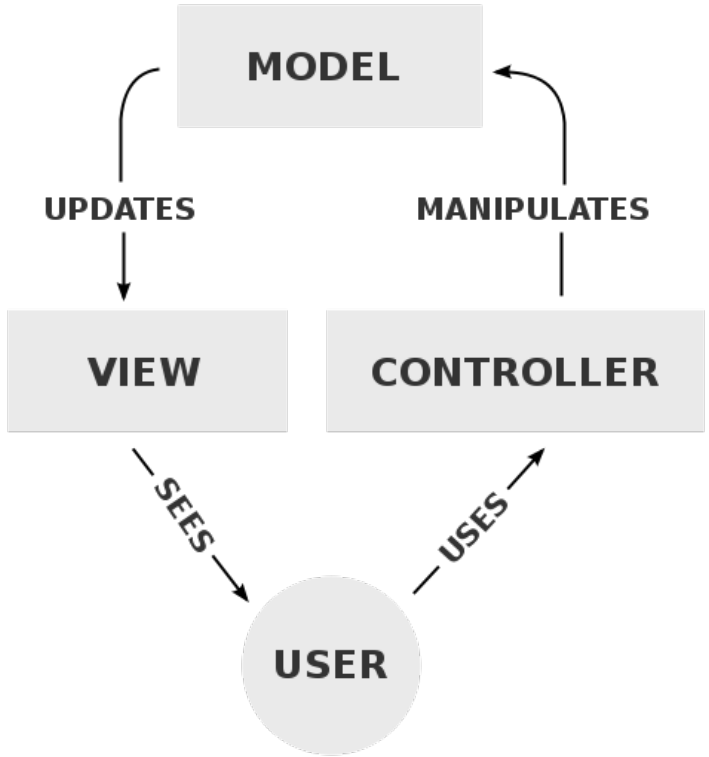
- Has 4 components
  - *Event source* – publishes messages to particular channels
  - *Event listener* – subscribes to particular channels to get notifications of messages
  - *Channel*
  - *Event bus (Common API)*
- Publish–subscribe (pub-sub) pattern
- e.g., Android EventBus





Model-view-controller (MVC) Pattern

- Isolates business logic from user interface
- Makes it easier to modify applications either their visual appearance or the underlying business rules without affecting others



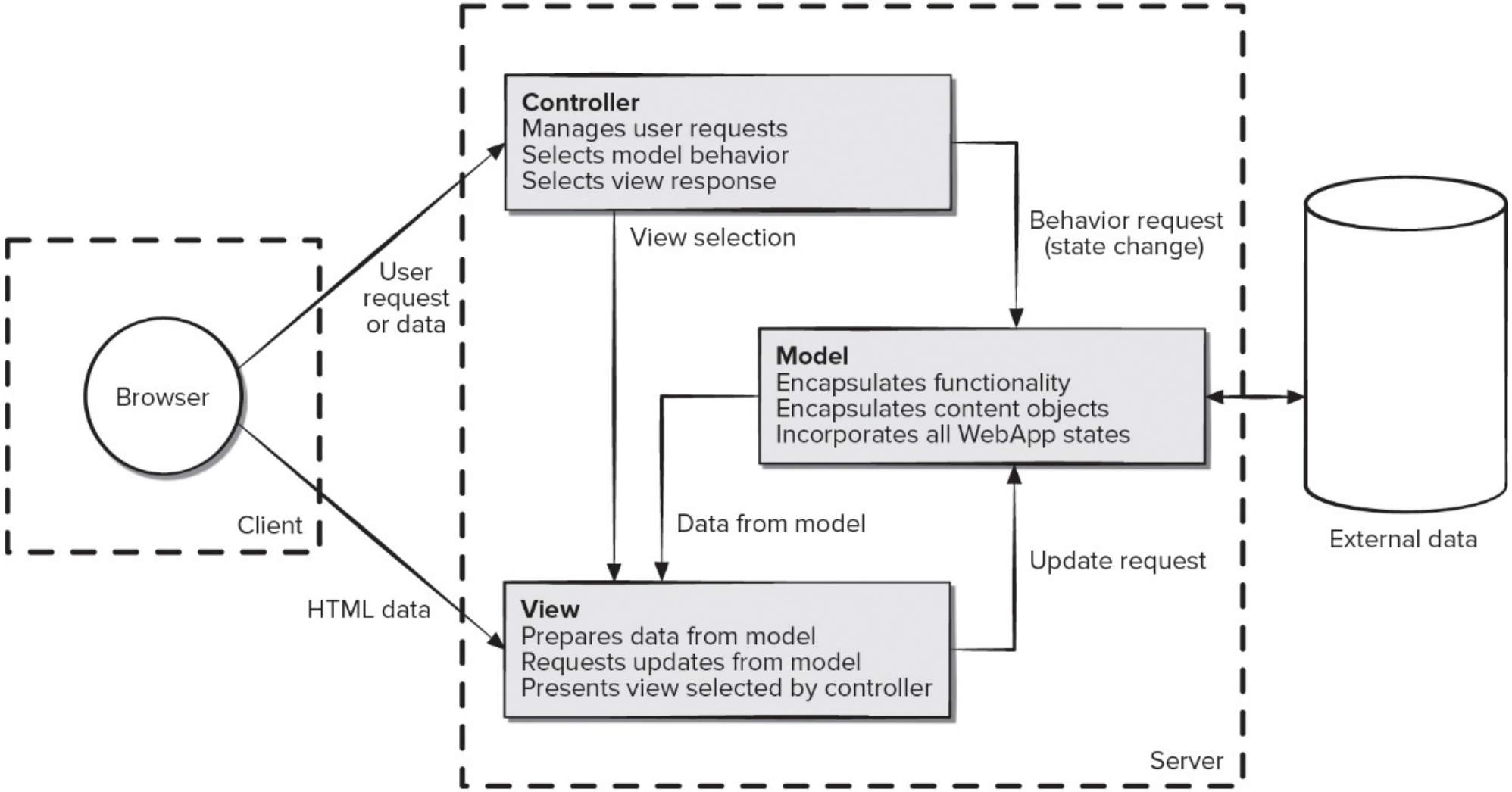
Model: the domain-specific representation of the information on which the application operates

View: renders the model into a form suitable for interaction, typically a user interface element

Controller: processes and responds to events, typically user actions, and may invoke changes on the model

[Wikipedia]

MVC Example – Dynamic Web Applications

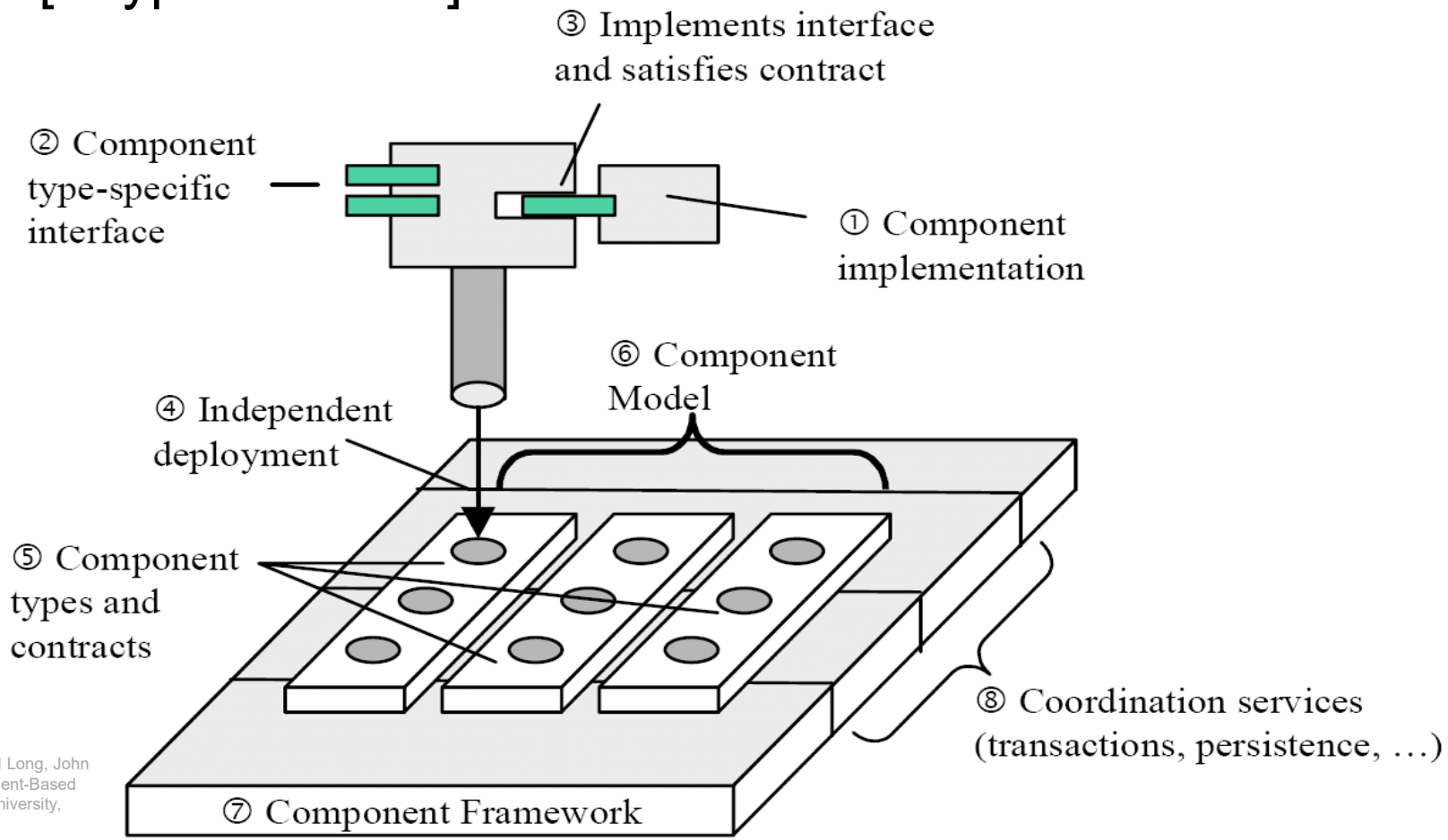


Source: Adapted from Jacyntho, Mark Douglas, Schwabe, Daniel and Rossi, Gustavo, "An Architecture for Structuring Complex Web Applications," 2002, available at http://www-di.inf.puc-rio.br/schwabe/papers/OOHDMJava2%20Report.pdf

[PrMa20]

Component-based Design Pattern

- *Software Components*: “Binary (executable) units that are independently produced, acquired and deployed, and can be connected to each other to form a composite system.” [Szyperski 1998]

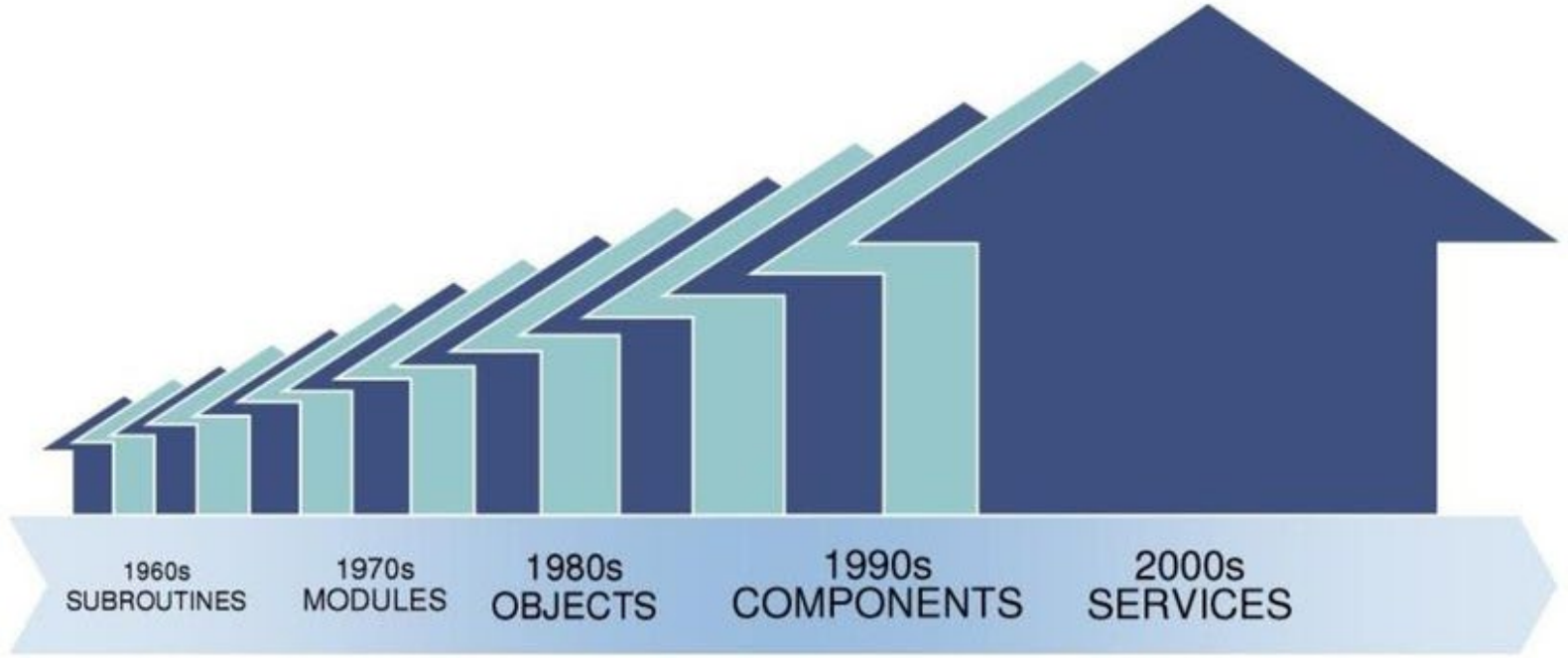


[BBB+00] Felix Bachmann, Len Bass, Charles Buhman, Santiago Comella-Dorda, Fred Long, John Robert, Robert Seacord, and Kurt Wallnau, "Volume II: Technical Concepts of Component-Based Software Engineering, 2nd Edition", Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2000-TR-008, 2000.

Why Component-based?

- Reusability
- Flexibility
  - Easier to customize a software system
  - Easier to satisfy changing user requirements
- Maintainability
  - Independent extensions (shorter upgrade cycle)
  - Easier to maintain state-of-the-art software
- Productivity
  - Reduced time-to-market
  - More outsourcing components (*component markets*)
  - Better cost efficiency
  - Higher quality (improved predictability & reliability)

Software Reuse History



Software Engineering Institute | Carnegie Mellon

24

Software Reuse & Component-Based Development (SE441)