Software Engineering

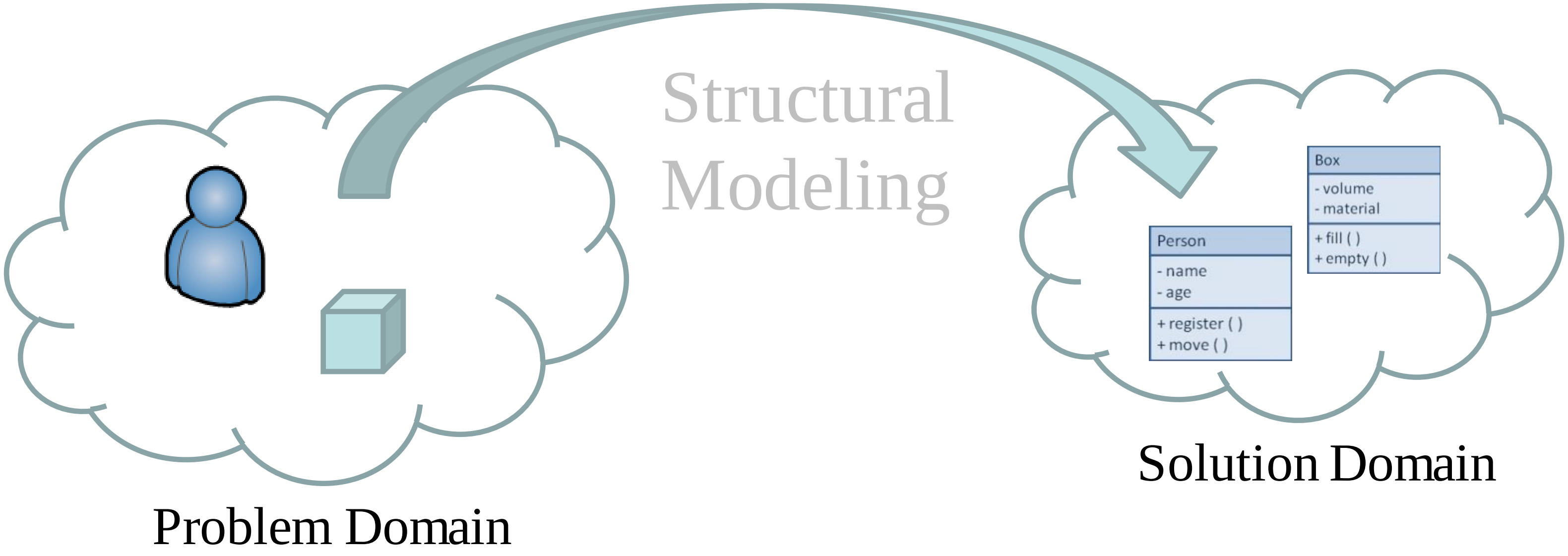
Dr. Young-Woo Kwon

Structural Models

- Drawn using an iterative process
  - First drawn in a conceptual, business-centric way
  - Then refined in a technology-centric way describing the actual databases and files
  - More and more detail is added in each iteration
- Create a vocabulary for analysts & users
  - Allows effective communication between analysts & users

Structural Models

- Main goal: to discover the key data contained in the problem domain and to build a structural model of the objects

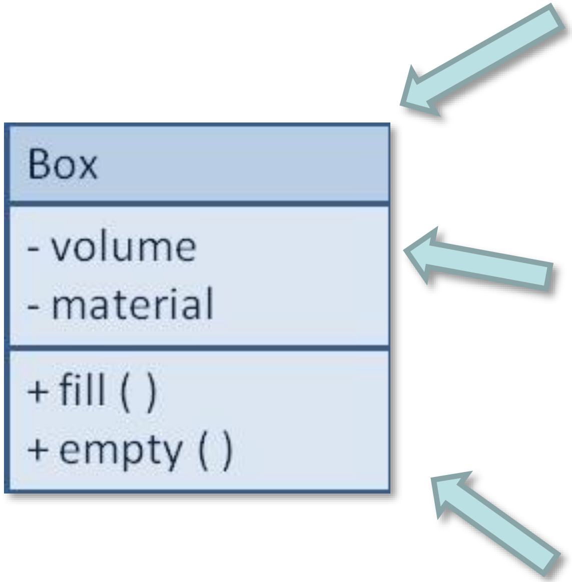


Structural Modeling

Problem Domain

Solution Domain

Classes, Attributes, & Operations



- Classes
  - Templates for instances of people, places, or things
- Attributes
  - Properties that describe the state of an instance of a class (an object)
- Operations
  - Actions or functions that a class can perform

Relationships

- Describe how classes relate to one another
- Three basic types in UML
  - Generalization
    - Enables inheritance of attributes and operations
    - Represents relationships that are “a-kind-of”
  - Aggregation
    - Relates parts to wholes or assemblies
    - Represents relationships that are “a-part-of” or “has-parts”
  - Association
    - Miscellaneous relationships between classes
    - Usually a weaker form of aggregation

Object Identification

- Textual analysis of use-case information
  - Nouns suggest classes
  - Verbs suggest operations
  - Creates a rough first cut to provide an object list
- Brainstorming—people offering ideas
  - Initial list of classes (objects) is developed
  - Attributes, operations and relationships to other classes can be assigned in a second round

Object Identification (cont.)

- Common Object Lists
  - Physical things
  - Incidents
  - Roles
  - Interactions
- Patterns
  - Useful groupings of collaborating classes that provide solutions to common problems (are reusable)
  - Developed patterns provide a starting point for work in similar domains

Class Diagrams

- A static model that shows classes and their relationships to one another
- Elements
  - Classes
    - Objects within the system (a person, place or thing)
    - Stores and manages information in the system and contains:
      - Attributes—characteristics of the class
      - Operations—activities the class can perform
  - Relationships—the associations between classes
    - Depicted as lines between classes
    - Multiplicity indicates how many of one object is/are associated with other objects

Attributes

- Properties of a class
  - Person: last name, first name, address, etc.
  - Attributes can be derived
    - Preceded with a slash (/)
    - e.g., age is derived from date of birth
- Visibility of an attribute:
  - Restricts access to attributes to ensure consistency
  - Public attributes (+): visible to all classes
  - Private attributes (-): visible only to an instance of the class in which they are defined
  - Protected attributes (#): visible only to an instance of the class in which they are defined and its descendants

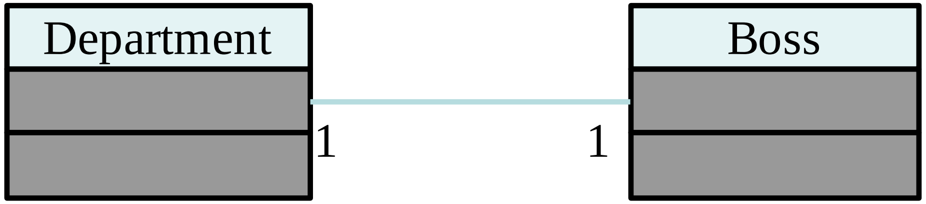
Operations

- Common operations are not shown
  - Create or delete an instance
  - Return or set a value
- Types of operations:
  - Constructor—creates an object
  - Query—makes information about the state of an object available
  - Update—changes values of some or all of an object's attributes
  - Destructor—deletes or removes an object

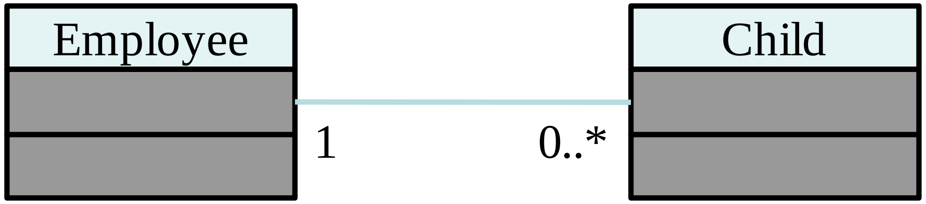
Relationships

- Denotes associations between classes
  - Depicted with a line labeled with the name of the relationship
  - May be directional (depicted with a triangle; e.g., a patient schedules an appointment)
- Classes may be related to themselves (e.g., employees and managers who may be members of the same class)
- Multiplicity indicates how many of one class are related to another class

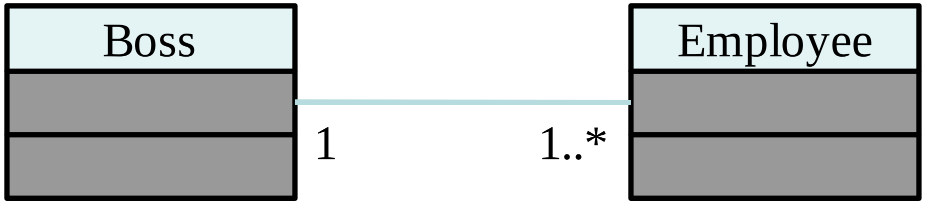
Multiplicities



Exactly one:
A department has one and only one boss



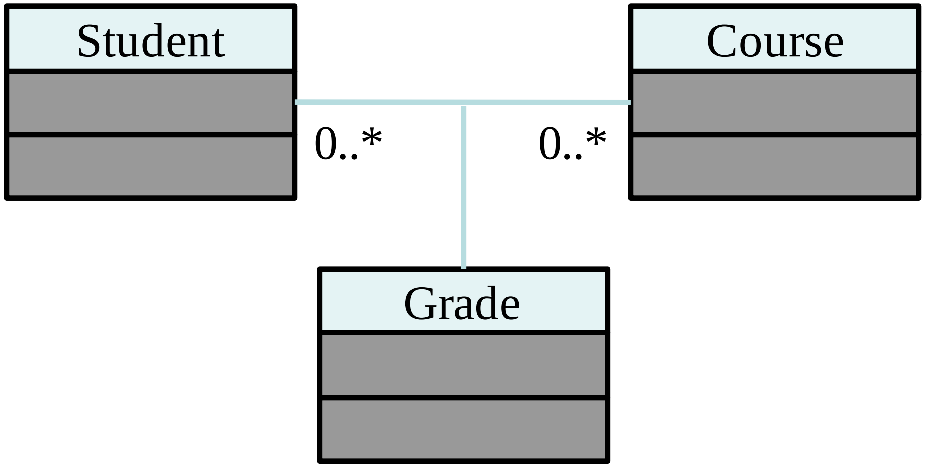
Zero or more:
An employee has zero to many children



One or more:
A boss is responsible for one or more employees

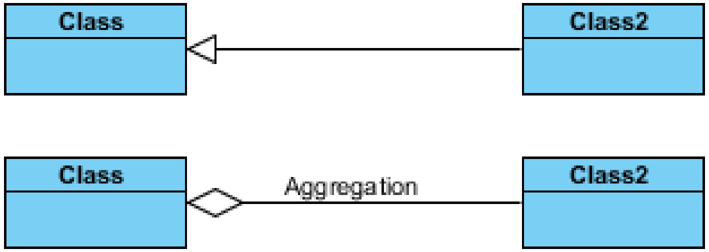
Association Classes

- Common in many-to-many relationships
- Used when attributes about the relationship between two classes needs to be recorded
  - Students are related to courses; a Grade class provides an attribute to describe this relationship

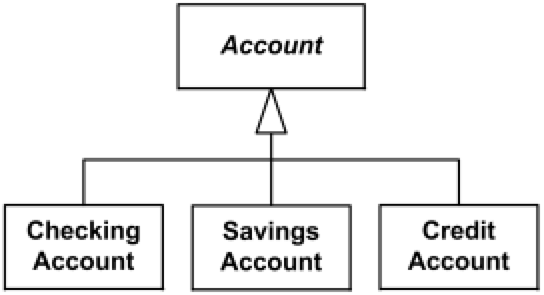


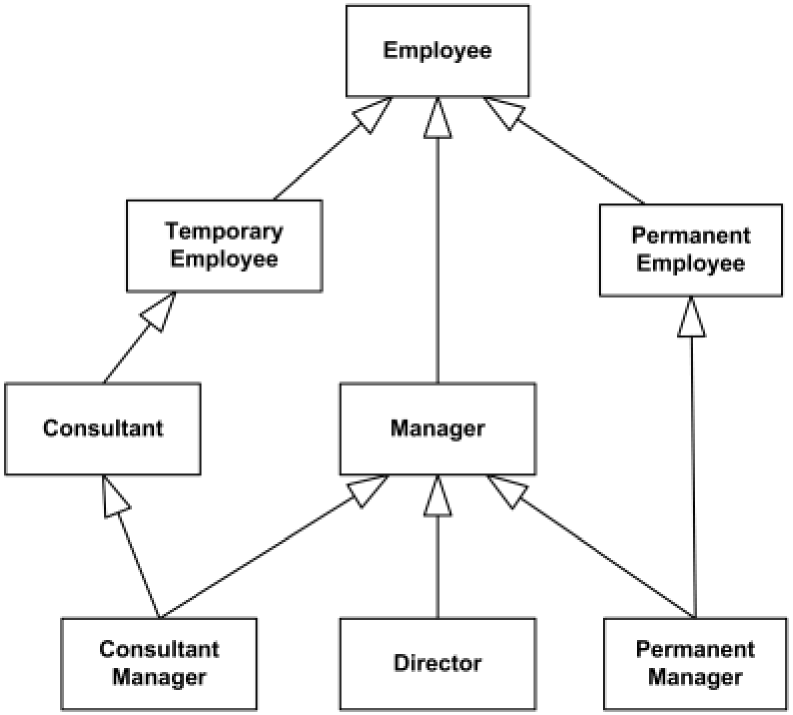
Generalization & Aggregation Associations

- Generalization denotes inheritance
  - Properties and operations of the superclass are valid for the sub-class
  - Depicted as a solid line with a hollow arrow pointing at the superclass
- Aggregation denotes a logical “a-part-of” relationship

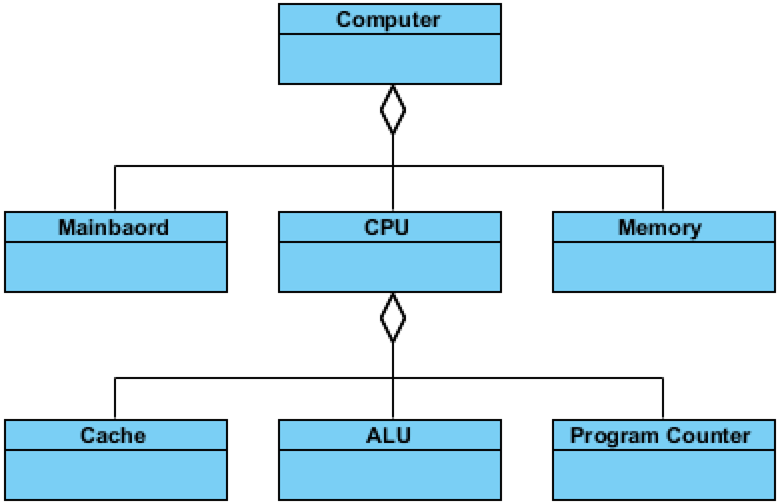


Sample Generalizations



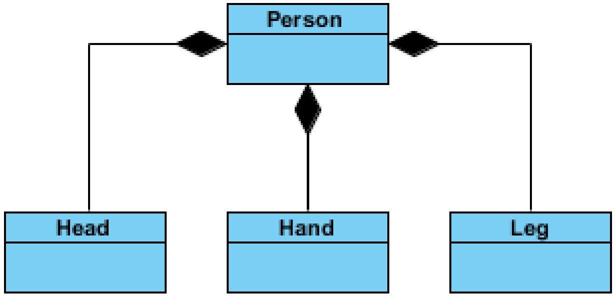


Sample Aggregations

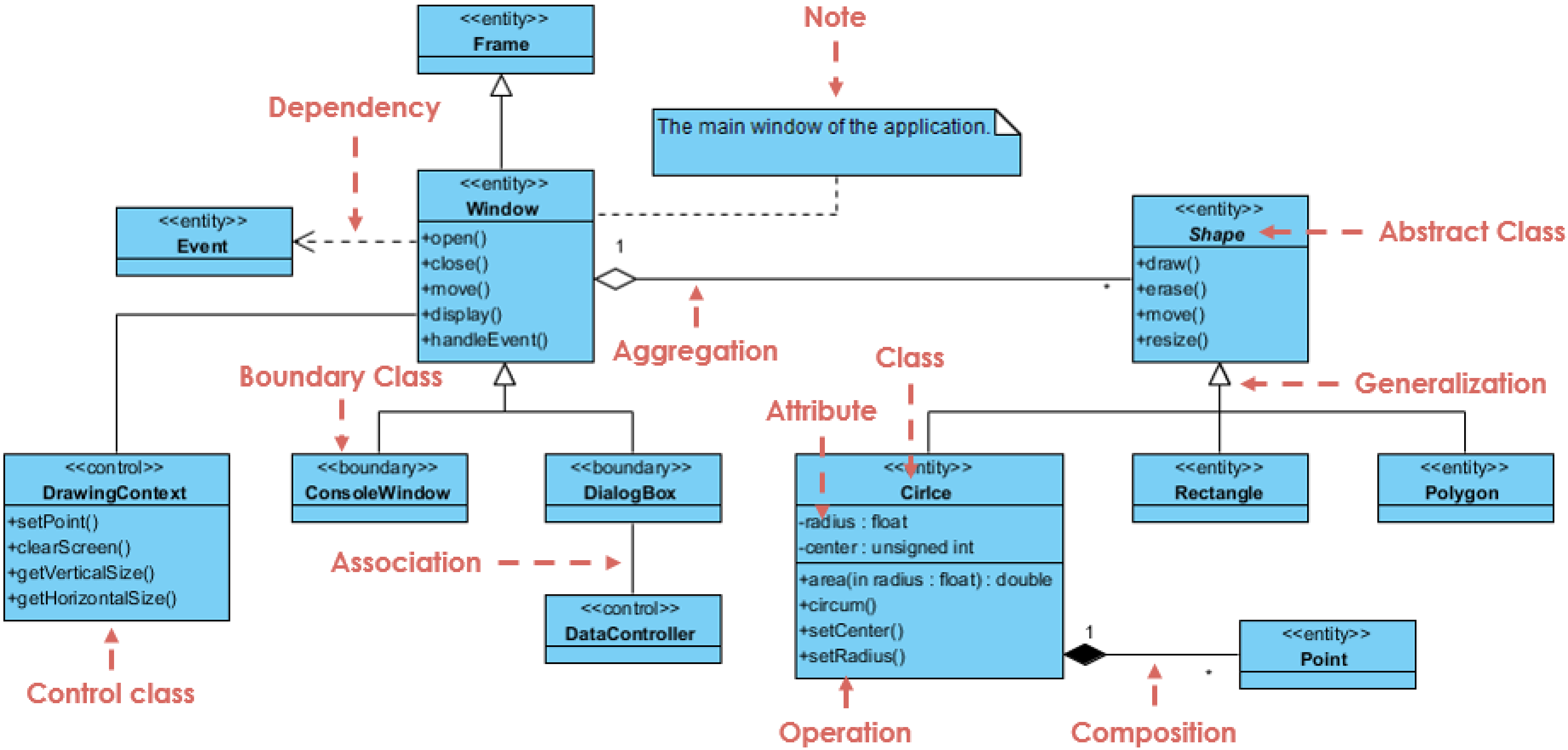


Composition

- A special type of aggregation where parts are destroyed when the whole is destroyed.
  - Objects of Class2 live and die with Class1.
  - Class2 cannot stand by itself.



Sample Class Diagram



Note

Dependency

Abstract Class

Aggregation

Boundary Class

Class

Generalization

Attribute

Association

Control class

Operation

Composition

Simplifying Class Diagrams

- Fully populated class diagrams of real-world system can be difficult to understand
- Common ways of simplifying class diagrams:
  - Show only concrete classes
  - The view mechanism shows a subset of classes
  - Packages show aggregations of classes (or any elements in UML)

SOFTWARE DESIGN PROCESS

The Process of Design

- Definition:
  - Design is a problem-solving process whose objective is to find and describe a way:
    - To implement the system’s functional requirements...
    - While respecting the constraints imposed by the quality, platform and process requirements...
      - including the budget
    - And while adhering to general principles of good quality

Top-down Design

- Top-down design
  - First design the very high level structure of the system.
  - Then gradually work down to detailed decisions about low-level constructs.
  - Finally arrive at detailed decisions such as:
    - the format of particular data items;
    - the individual algorithms that will be used.

Bottom-up Design

- Bottom-up design
  - Make decisions about reusable low-level utilities.
  - Then decide how these will be put together to create high-level constructs.
- A mix of top-down and bottom-up approaches are normally used:
  - Top-down design is almost always needed to give the system a good structure.
  - Bottom-up design is normally useful when reusable components can be created.

Different aspects of design

- *Architecture design*:
  - The division into subsystems and components,
    - How these will be connected.
    - How they will interact.
    - Their interfaces.
- *Class design*:
  - The various features of classes.
- *User interface design*
- *Algorithm design*:
  - The design of computational mechanisms.

SOFTWARE ARCHITECTURE

“What”, “Who”, “How” of Software Development

- “What” → Requirements Analysis
- “Who” → Stakeholders
- “How” → Software Architecture

Software Architecture

The software architecture of a system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

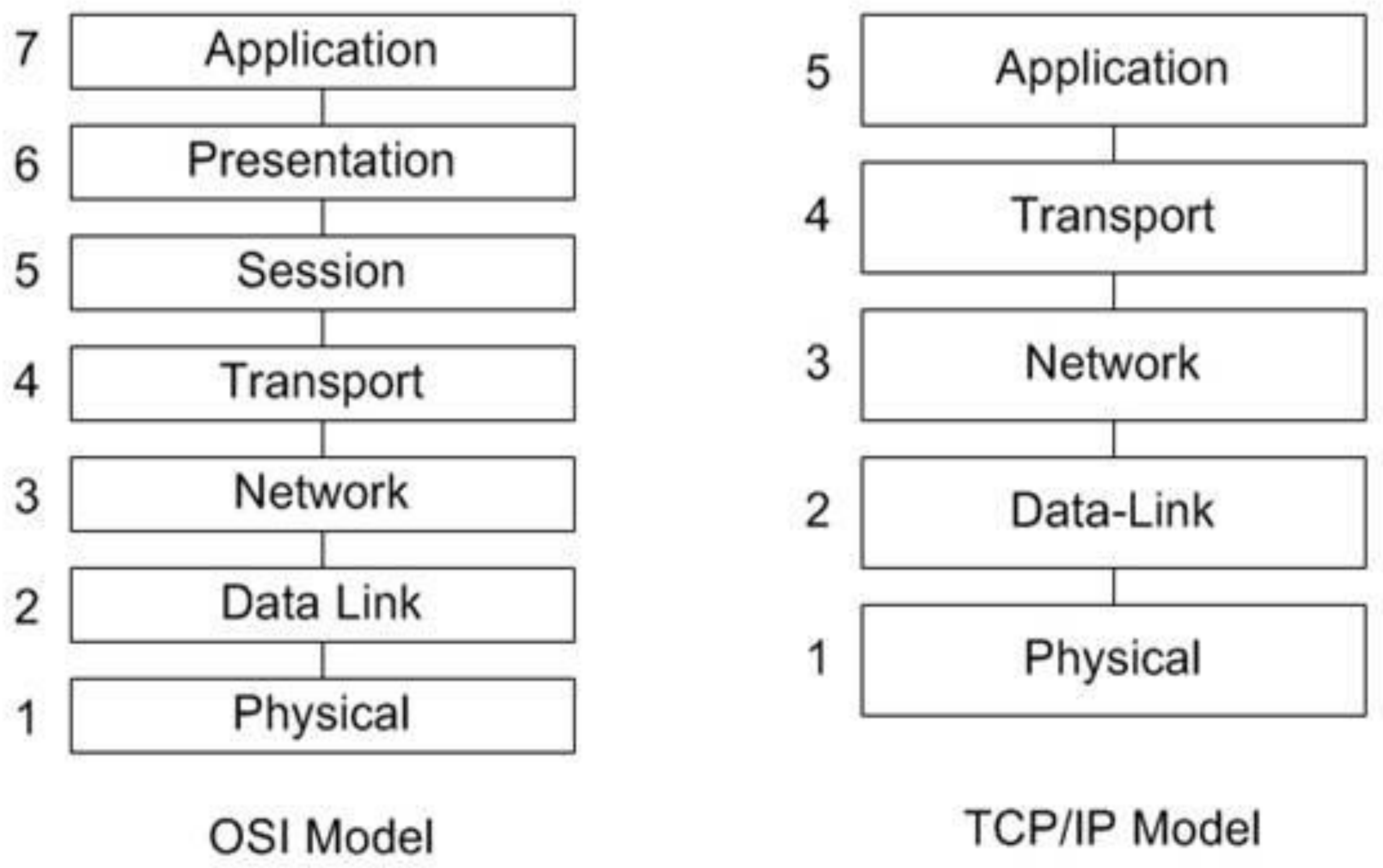
(from Bass, Clements, and Kazman, *Software Architecture in Practice*, SEI Series in Software Engineering. Addison-Wesley, 2003.)

Architectural Patterns

- The notion of patterns can be applied to software architecture.
  - These are called architectural patterns or architectural styles.
- Each pattern has a:
  - Context, a recurring situation in the world that gives rise to a problem
  - Problem, appropriately generalized
  - Solution, a successful architectural solution to a problem

1. Multi-Layer Pattern

- Problem: Pieces of the system need to be built and tested independently of one another
- Solution: The layered pattern defines layers (groupings of modules that offer a cohesive set of services) and a unidirectional allowed-to-use relation among the layers
- The pattern is usually shown graphically by stacking boxes representing layers on top of each other



7 Application
6 Presentation
5 Session
4 Transport
3 Network
2 Data Link
1 Physical

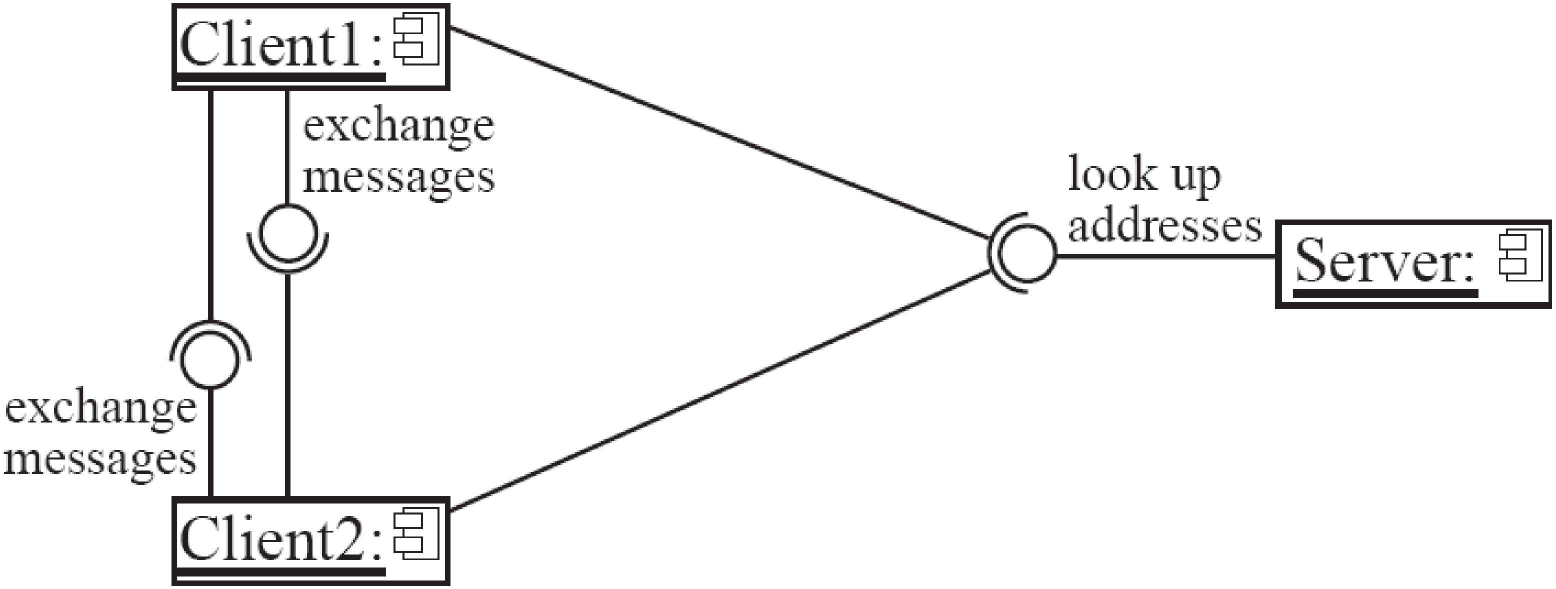
OSI Model

5 Application
4 Transport
3 Network
2 Data-Link
1 Physical

TCP/IP Model

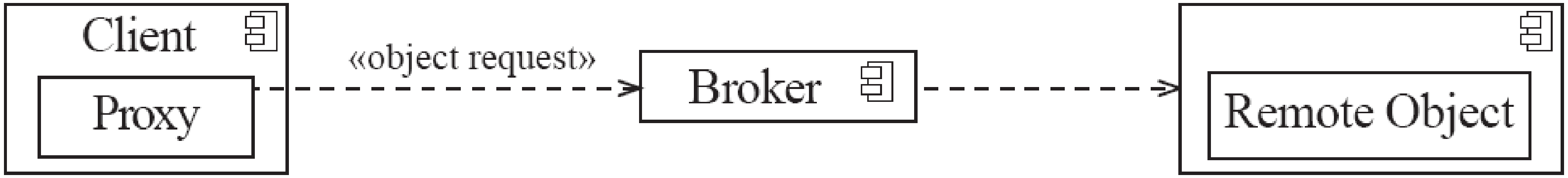
2. The Client-Server and other distributed architectural patterns

- Problem: A large number of distributed clients need to access shared resources or services
- Solution: Client components initiate interactions with server components, invoking services as needed from those servers and waiting for the results of those requests.


Client1:
exchange messages
look up addresses
Server:
exchange messages
Client2:

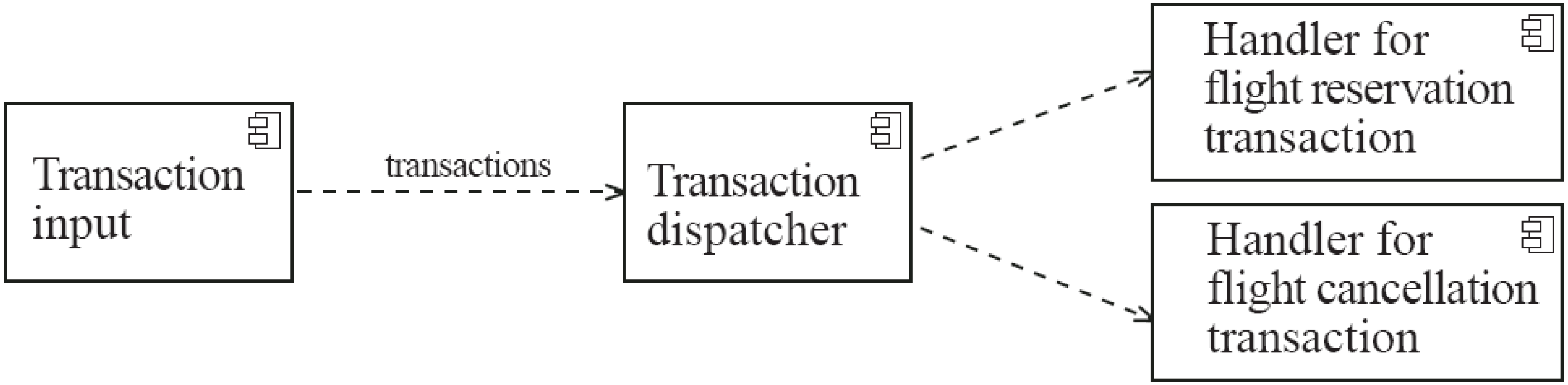
The Broker architectural pattern

- Transparently distribute aspects of the software system to different nodes
  - An object can call methods of another object without knowing that this object is remotely located.
  - CORBA is a well-known open standard that allows you to build this kind of architecture.


Client
Proxy
«object request»
Broker
Remote Object

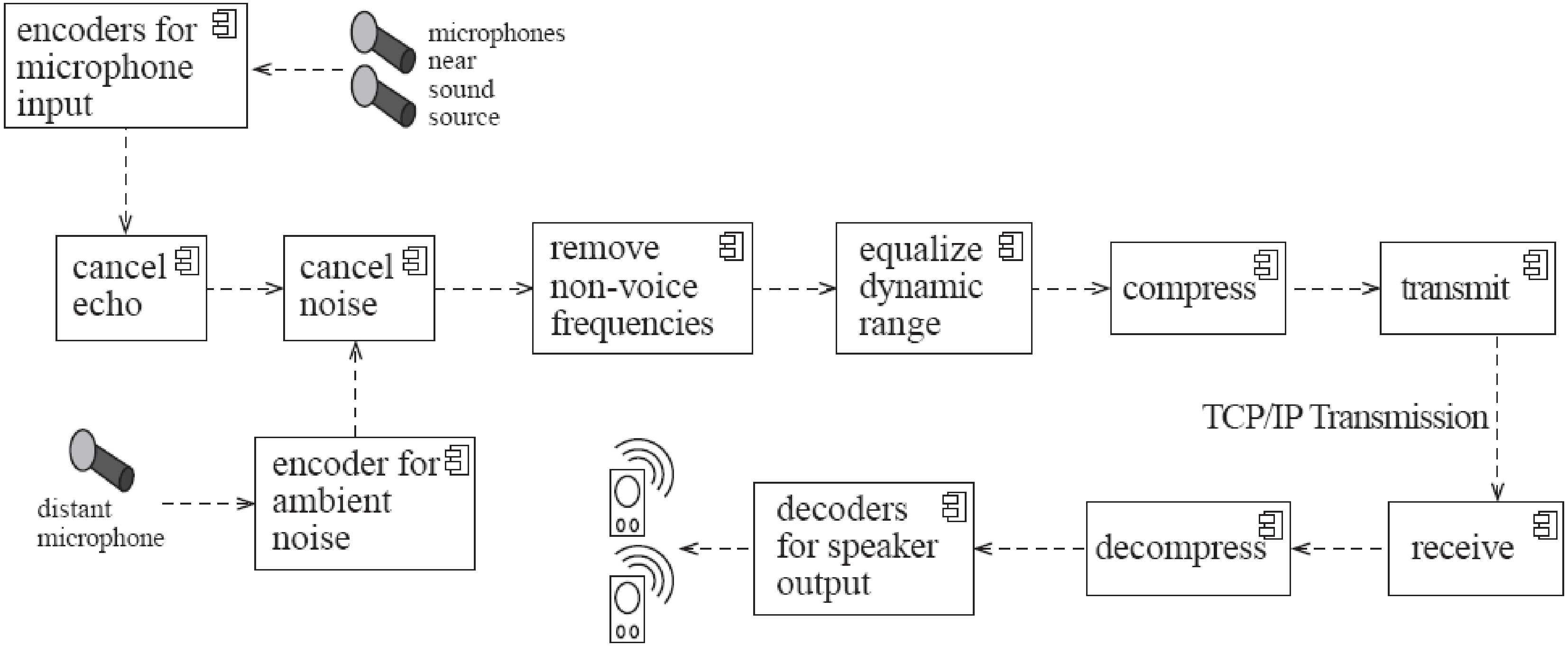
The Transaction-Processing architectural pattern

- Problem: The system must read and handle a series of inputs that change stored data
- Solution: A dispatcher component decides what to do with each transaction (input) and calls a procedure or messages a component to handle it


Transaction input
transactions
Transaction dispatcher
Handler for flight reservation transaction
Handler for flight cancellation transaction

The Pipe-and-Filter architectural pattern

- A stream of data, in a relatively simple format, is passed through a series of processes
  - Each of which transforms it in some way.
  - Data is constantly fed into the pipeline.
  - The processes work concurrently.
  - The architecture is very flexible.
    - Almost all the components could be removed.
    - Components could be replaced.
    - New components could be inserted.
    - Certain components could be reordered.


encoders for microphone input
microphones near sound source
cancel echo
cancel noise
remove non-voice frequencies
equalize dynamic range
compress
transmit
TCP/IP Transmission
distant microphone
encoder for ambient noise
decoders for speaker output
decompress
receive

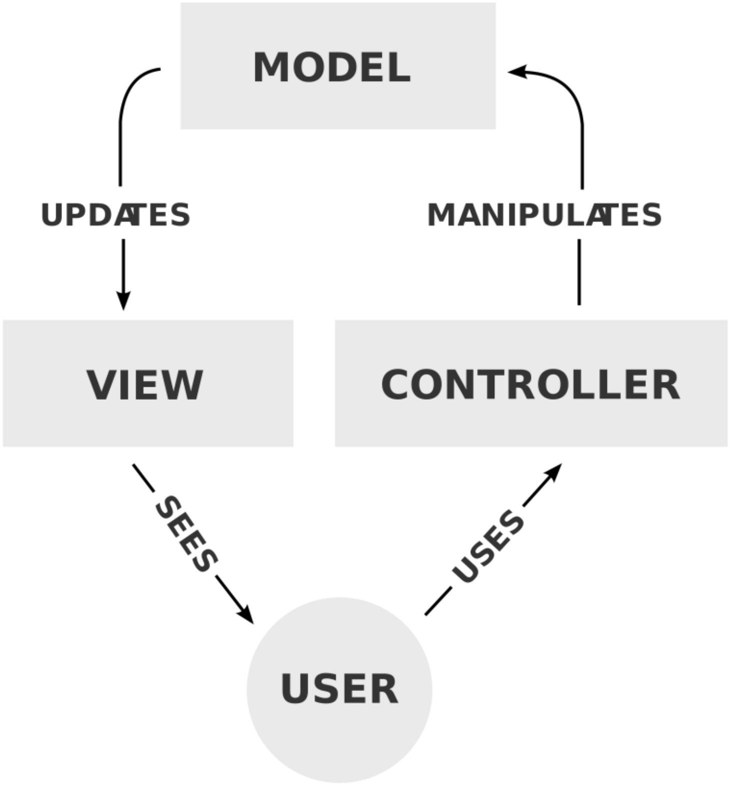
The Model-View-Controller (MVC) architectural pattern

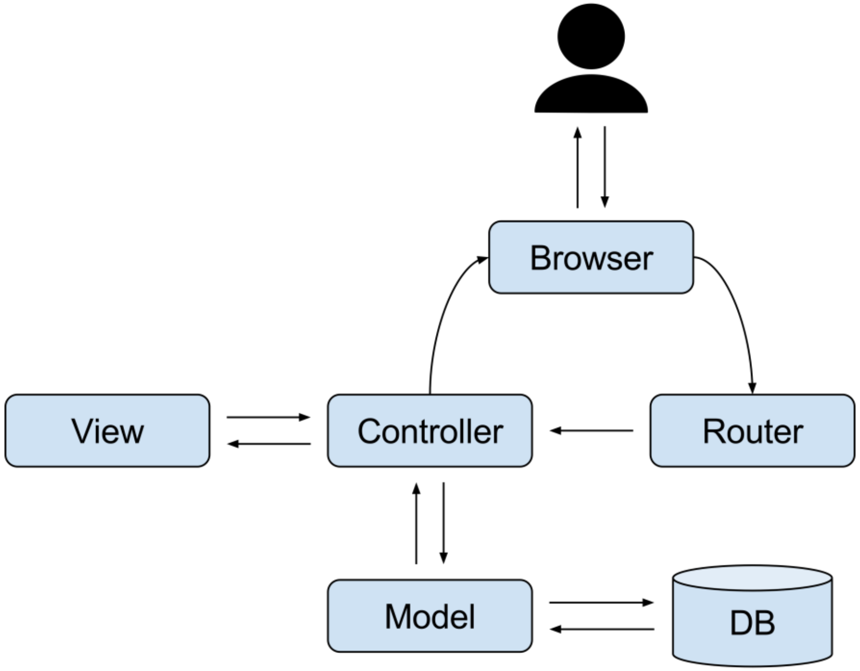
- Problem: The user interface needs to be modified frequently without impacting the rest of the system's functionality
- Solution: The MVC pattern breaks system functionality into three components: a model, a view, and a controller that mediates between the model the view

Model-View-Controller Pattern

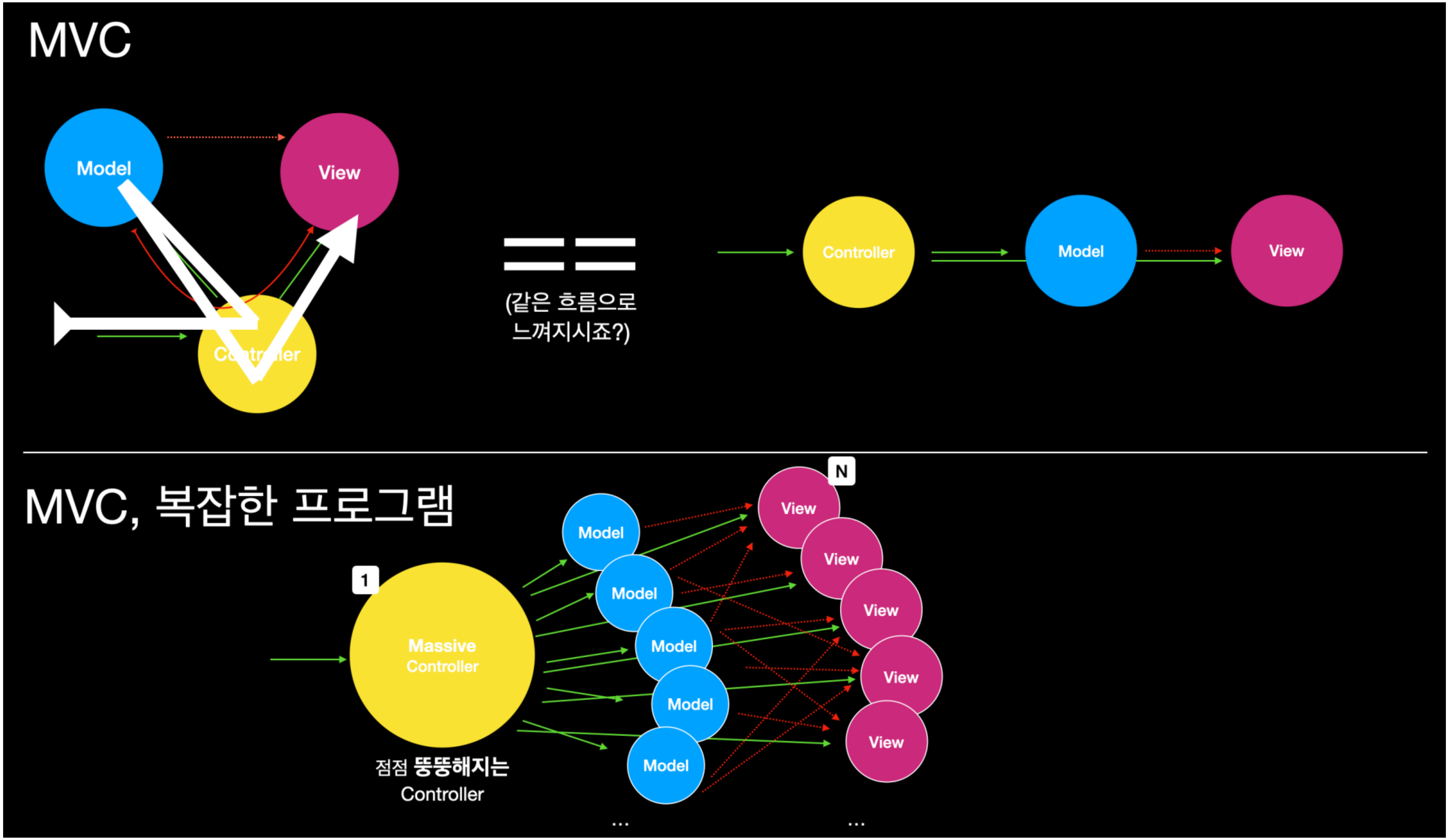
- The model contains the underlying classes whose instances are to be viewed and manipulated
- The view contains objects used to render the appearance of the data from the model in the user interface
- The controller contains the objects that control and handle the user’s interaction with the view and the model

MVC Pattern


MODEL
UPDATES
MANIPULATES
VIEW
CONTROLLER
SEES
USES
USER

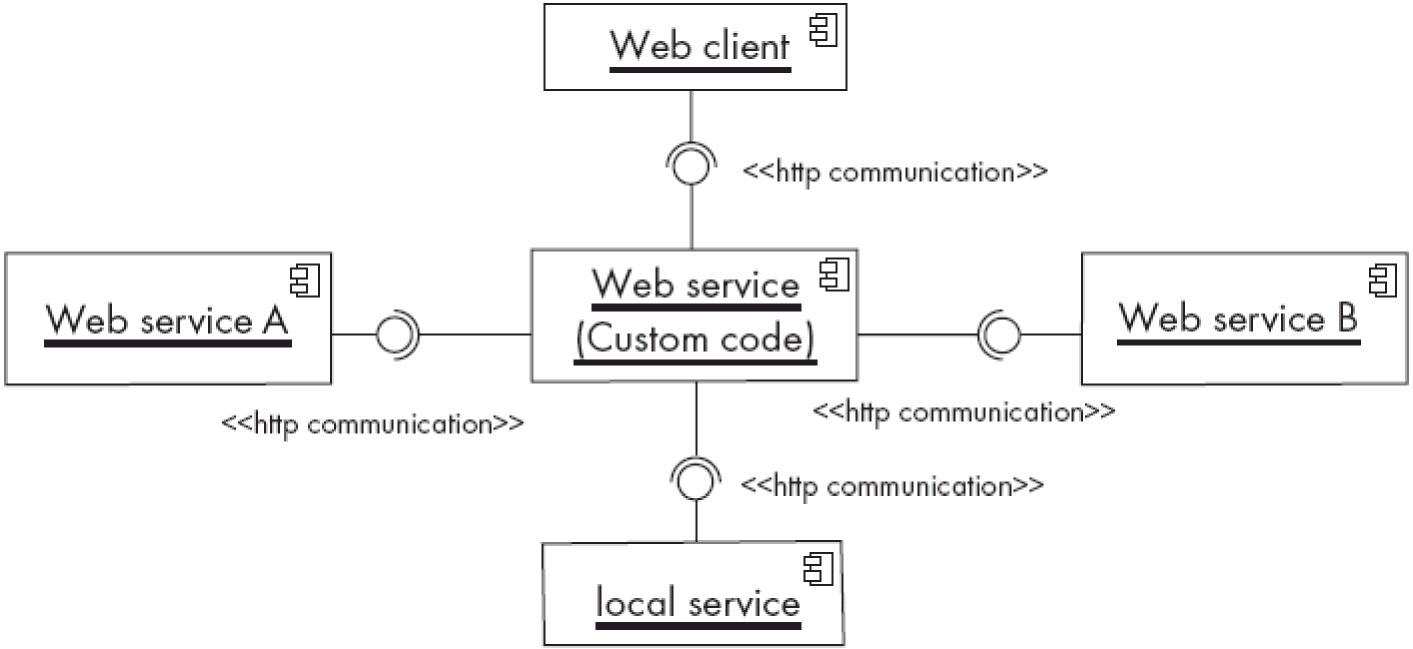

Browser
View
Controller
Router
Model
DB

MVC (Massive View Controller?)



The Service-oriented architectural pattern

- Problem: Service consumers must be able to use/access a number of service providers without understanding their implementation
- Solution: Computation is achieved by cooperating peers that request service from and provide services to one another across a network.
- In the context of the Internet, the services are called Web services


Web client
<<http communication>>
Web service A
Web service
(Custom code)
Web service B
<<http communication>>
<<http communication>>
<<http communication>>
local service