ILLINOIS INSTITUTE
OF TECHNOLOGY

Transforming Lives. Inventing the Future.
www.iit.edu

Software Engineering
CS 487

Dennis Hood

Computer Science

Summer ‘19

Lesson Overview

- Design: Modeling and Architecture
- Reading
  - Ch. 5 - System Modeling
  - Ch. 6 - Architectural Design
  - Ch. 7 – Design and Implementation
- Objectives
  - Explore the design phase
  - Understand the role of modeling in creating systems – a picture says a thousand words
  - Examine State-Transition Diagrams which can be used to model system behavior and plan the interface to the user
  - Discuss the concept of architecture in the context of software systems
  - Examine architectures to meet the demands of various structural challenges
  - Analyze the concept of design “patterns” – common solutions to common problems

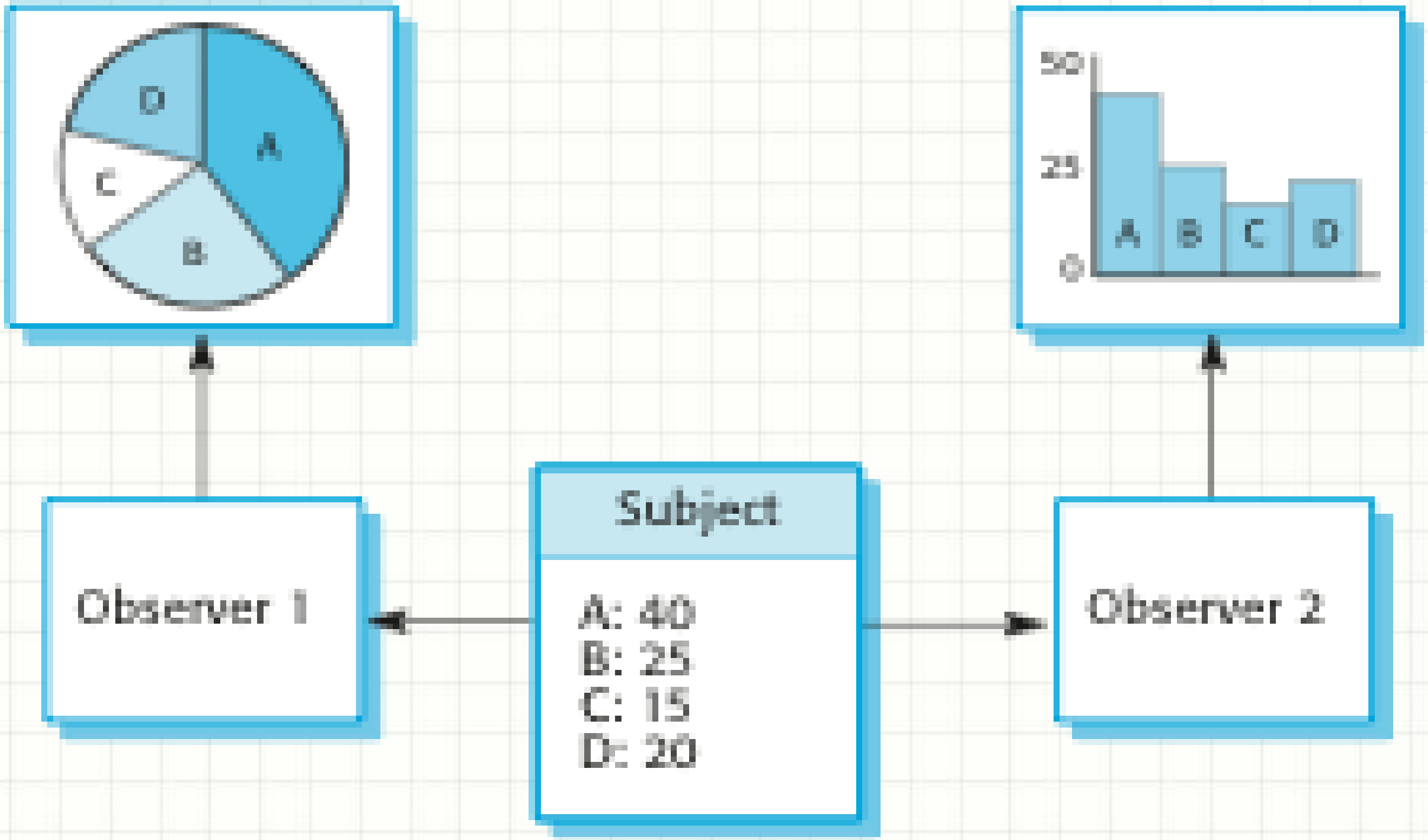
Lecture 5 Modeling and Architecture

Design Patterns

- Adopted from the (building) architecture community
- Common solutions to common problems
- “Why reinvent the wheel?”
  - Similar benefits as in component reuse
  - Already proven, already tested
  - Significant up-front time savings,
  - even greater potential benefit by avoiding the fix-retest cycle

Ex. Observer Pattern

- Separate the display of an object's state from the object itself



From Design to Implementation

- Reuse
  - Abstract reuse via design patterns
  - Object-oriented design and development
  - Reusable components
  - Reusable systems (tailored COTS)
- Configuration management
  - Version control
  - System build management
  - Issue management
- Host-target development
  - Configure development host to match target
  - Simulate target for testing

Architecture

- Levels of abstraction
  - Program-level architecture (“small”)
  - System- or enterprise-level (“large”)
- Designing the building
  - Solid foundation
  - Structure that meets basic needs
  - Support for aesthetic elements
- Additional benefits of defined architecture
  - Means of communicating with stakeholders
  - Helps to complete the analysis
  - Facilitates large-scale reuse

Requirements Satisfaction

- Non-functional system requirements are largely met through architecture design
  - Performance optimization
  - Security
  - Safety
  - Availability
  - Maintainability
- Trade-offs are likely necessary due to conflicting priorities and/or overlaps in design elements

Design Decisions

- Can an existing generic application architecture be reused?
- How will the system be distributed across multiple processors?
- What are the appropriate architectural styles or patterns?
- How will structural elements be decomposed into modules?
- What is the strategy for controlling the operation of system units?
- How will the architecture be evaluated?, documented?

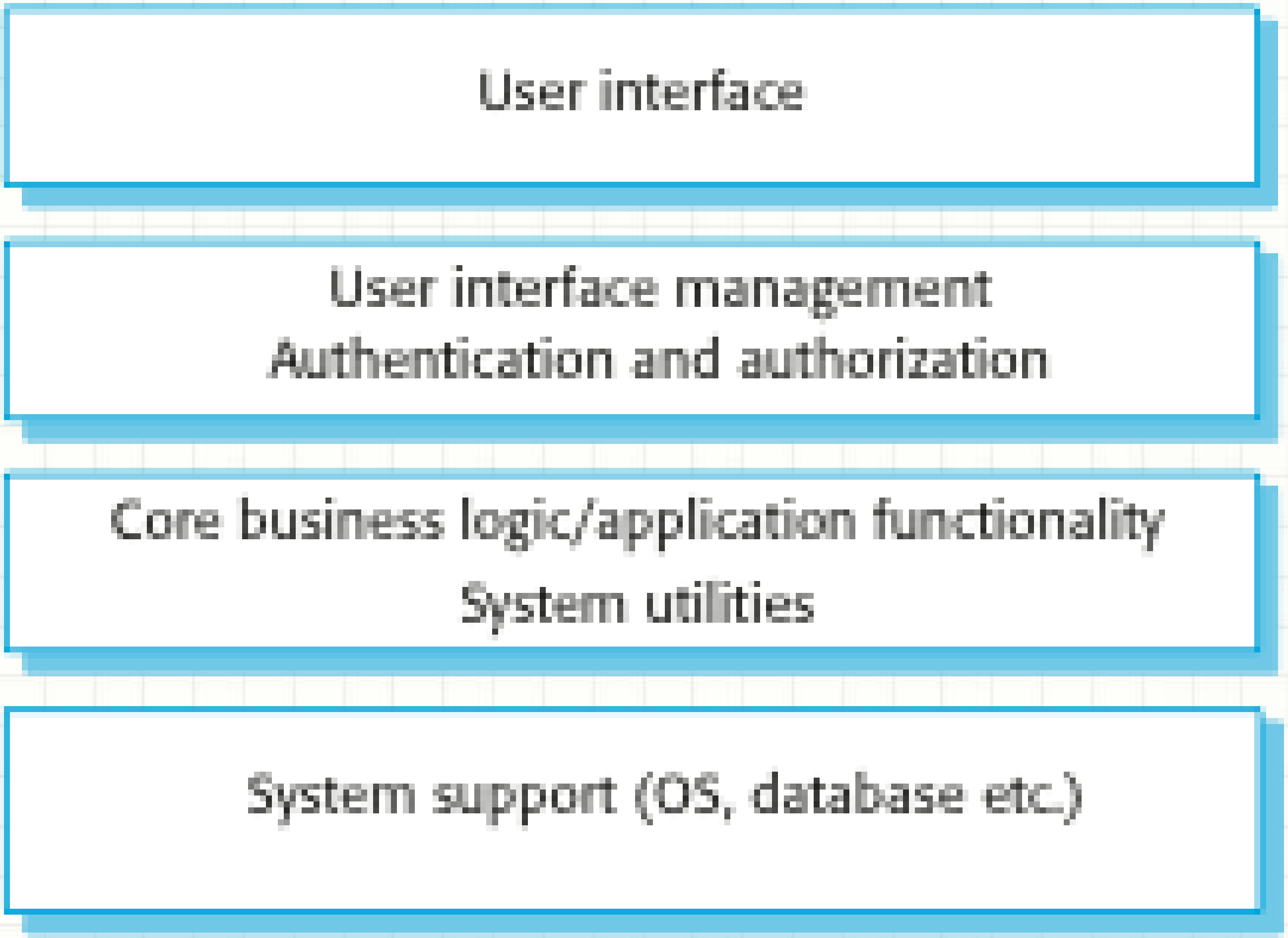
Architectural Views

- Multiple perspectives help bring complex into focus
  - Logical – the system as interacting objects
  - Process – interacting processes
  - Development – components to be developed
  - Physical – interacting hardware and software
  - Conceptual – the basis for decomposing high-level requirements

Architectural Patterns

- Layered architecture
  - Achieve separation and independence through layering
  - Hierarchical organization
  - Supports incremental development
- Repository architecture
  - Support the exchange of information between sub-systems
  - Use a central repository to manage shared data
  - Establish and maintain a separate database for each sub-system
- Client-server architecture
  - Organized as a set of services and associated servers, accessed by clients “calling” the services
- Pipe-and-Filter architecture
  - Workflow
  - Information is transformed as it flows through the system

Generic Layered Architecture



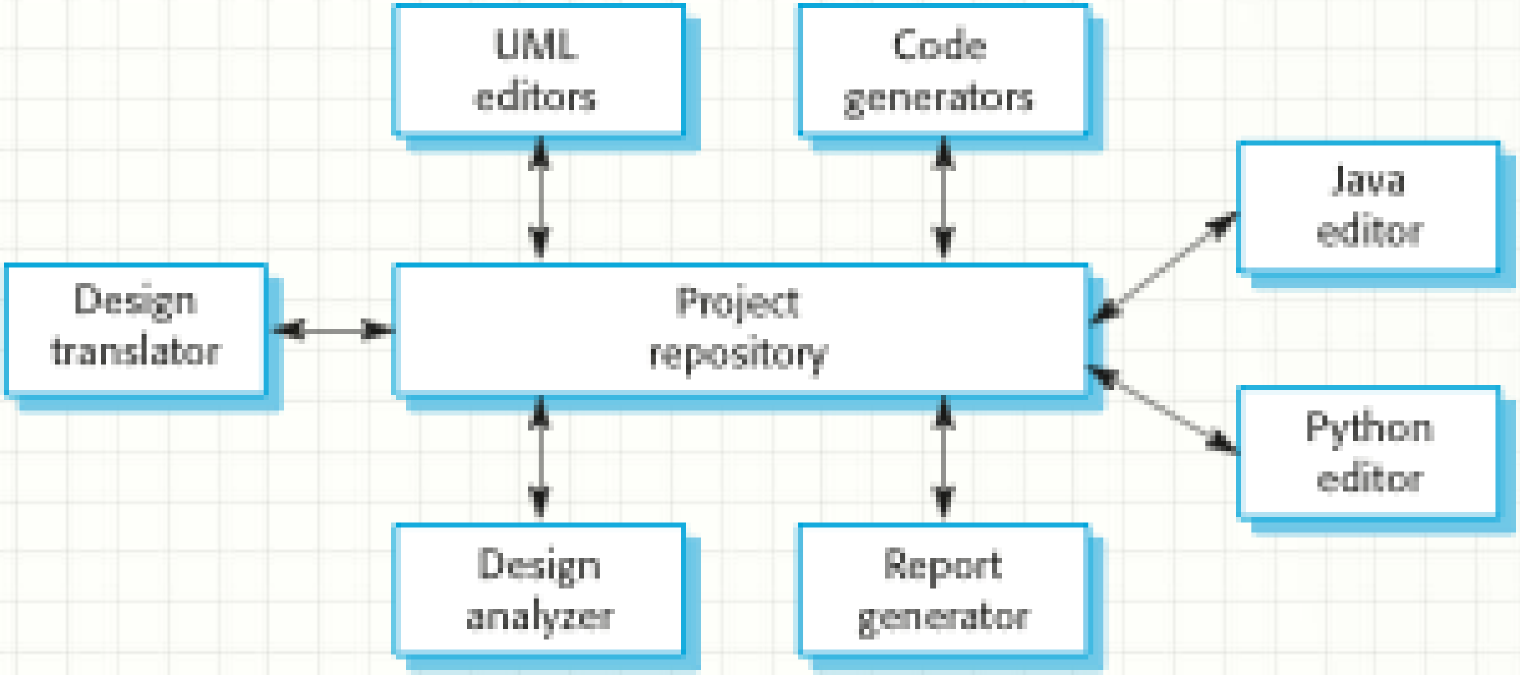
User interface

User interface management
Authentication and authorization

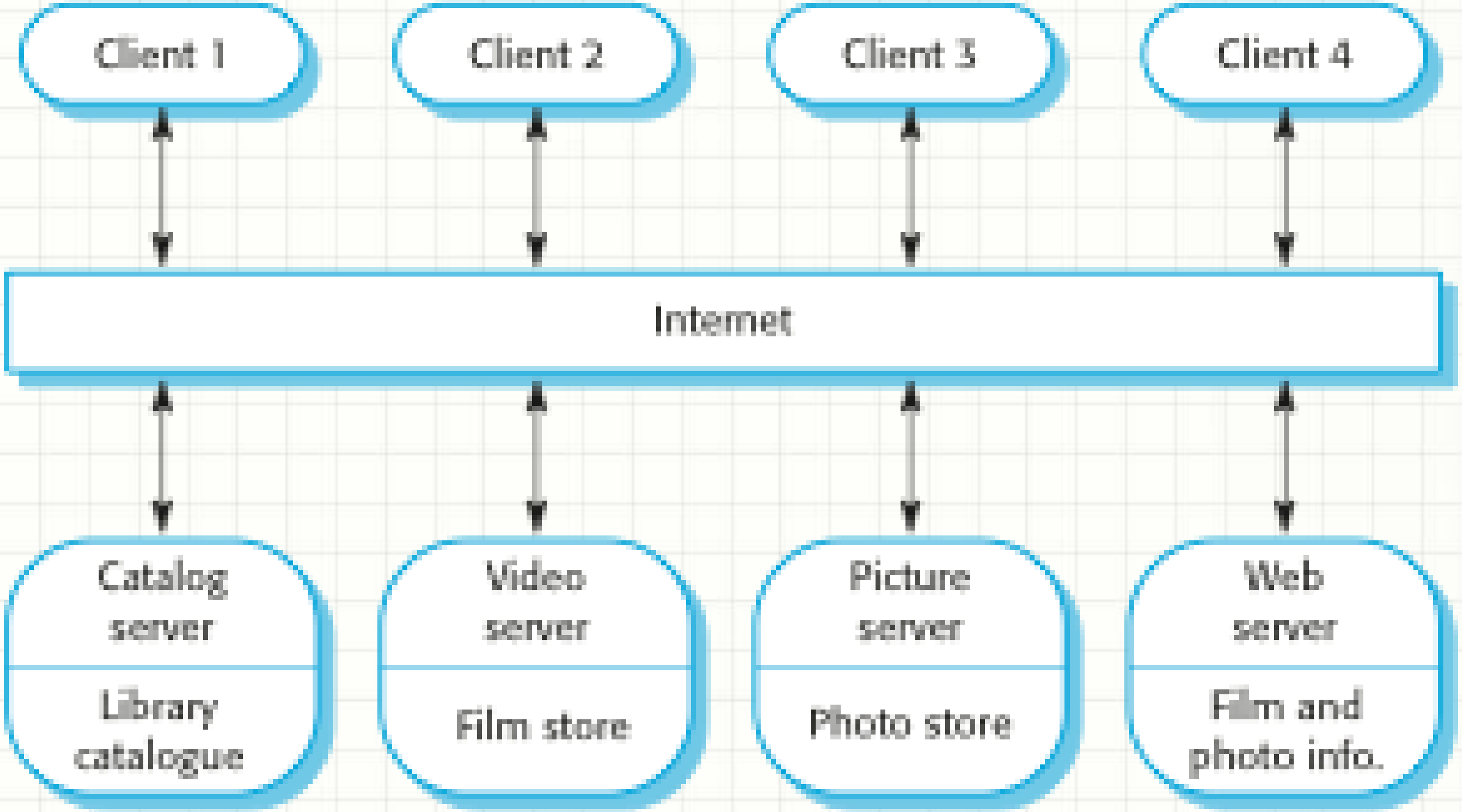
Core business logic/application functionality
System utilities

System support (OS, database etc.)

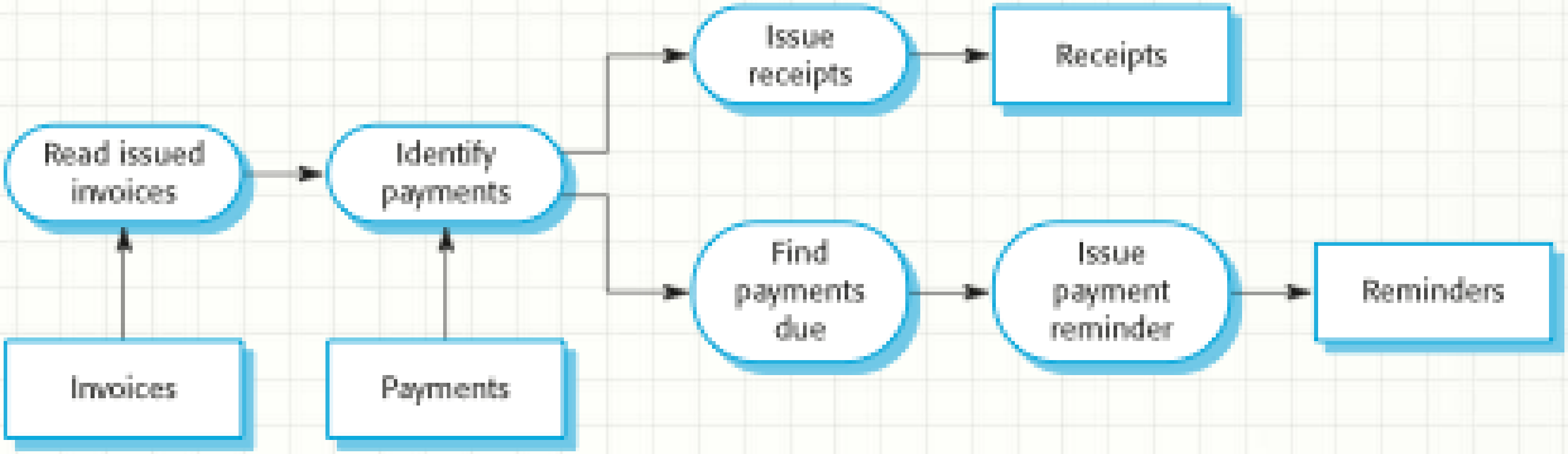
Ex. Repository Architecture



Ex. Client-Server Architecture



Ex. Pipe-and-Filter Architecture



Application Architectures

- Alternatives to system architectures that focus on the needs of the application
- Data-processing applications
- Transaction-processing applications
- Event-processing systems
- Language-processing systems

Modular Decomposition Styles

- Deciding how best to decompose sub-systems down to modules
- Object-oriented decomposition
  - Loosely coupled objects with well-defined interfaces
  - Classes are templates with attributes and operations
  - During execution, objects are instantiated from classes
- Function-oriented pipelining
  - Data flow
  - Inputs are processed by transformational functions to produce outputs
- The "real" world looks more like objects
  - Therefore, OO works best for reuse
- However, work looks more like functions
  - Therefore, FO provides the best immediate fit

Control Styles

- Deciding how best to control modules in operation
- Centralized control
  - Design a sub-system whose primary function is to control the other sub-systems
  - The vast majority of control is handled by this sub-system
- Event-based control
  - Design each sub-system to “react” to events
  - Events can come from other sub-systems or the environment
- Centralizing provides a single-point of design, implementation, etc. system focus
- De-centralizing will often be a more logical fit for modeling the “real” world

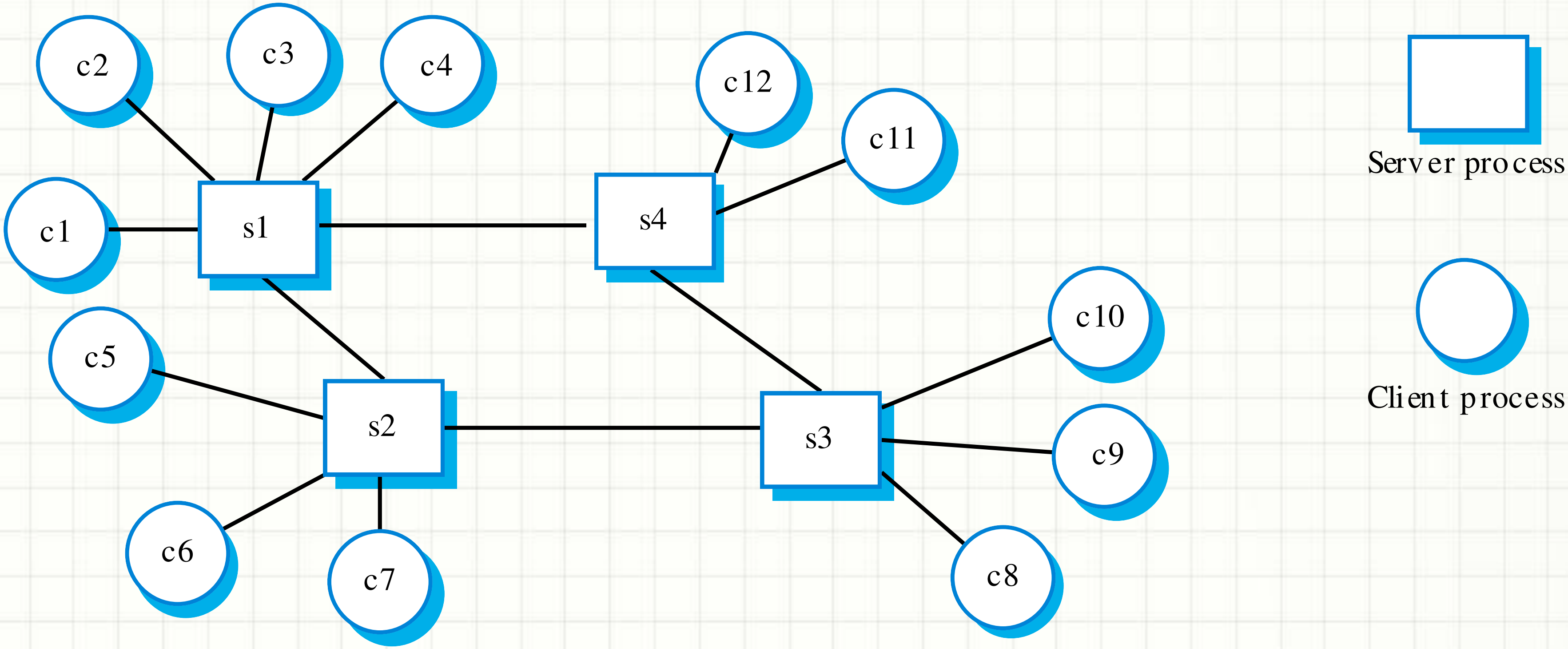
Designing Distributed Systems

- Spreading the processing load across multiple machines
- Benefits
  - Shared and therefore better utilized resources
  - Open and therefore more standard-driven systems
  - Concurrency
  - Scalability
  - Fault tolerance
- Disadvantages
  - Complexity
  - Vulnerable to security breaches
  - Difficult to manage
  - Unpredictability

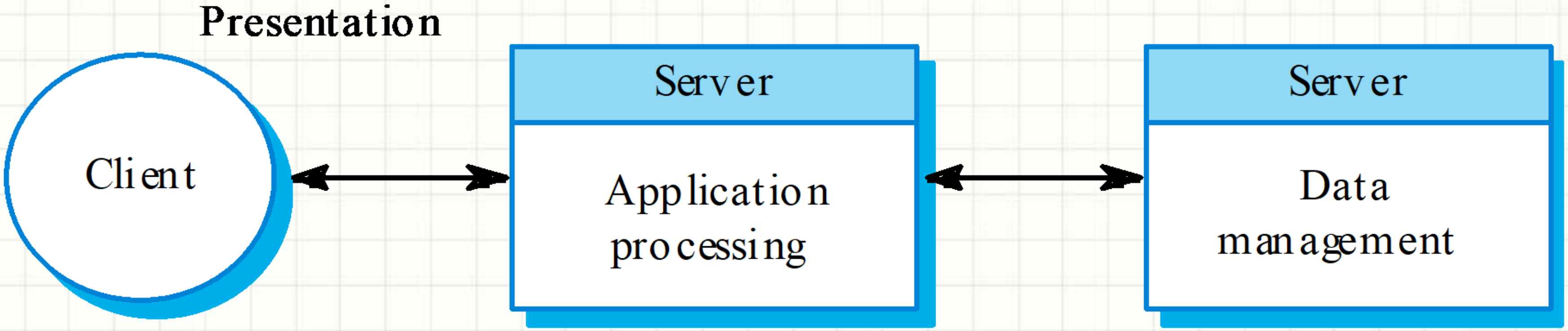
Types of Distributed Systems

- Multiprocessor architectures
  - The operating system *can* distribute the processes of a software system across multiple processors
  - The processes must be capable of running independent of each other
- Client-server architectures
  - A centralized server system "offers" services to
  - de-centralized client processes
    - Thin-client systems are designed such that all but the presentation is housed at the server
    - Fat-client systems are designed such that all but the data management is housed at the clients

Client-Server Architecture



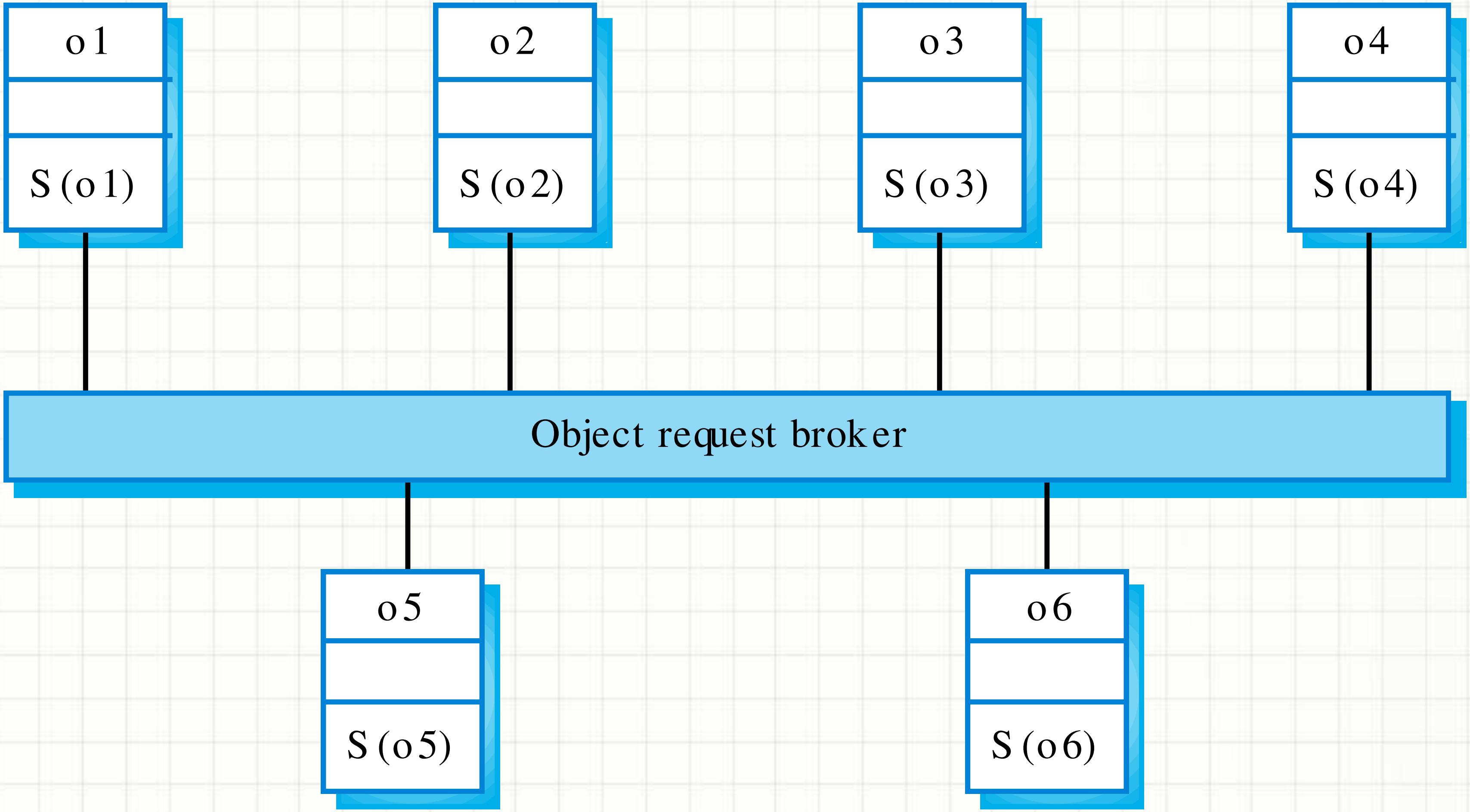
3-tier C/S Architecture



Distributed Object Architectures

- Less restrictive than client-server in that all objects can offer services to all other objects
- Middleware, known as an object request broker, allow distributed objects to communicate across networked computers

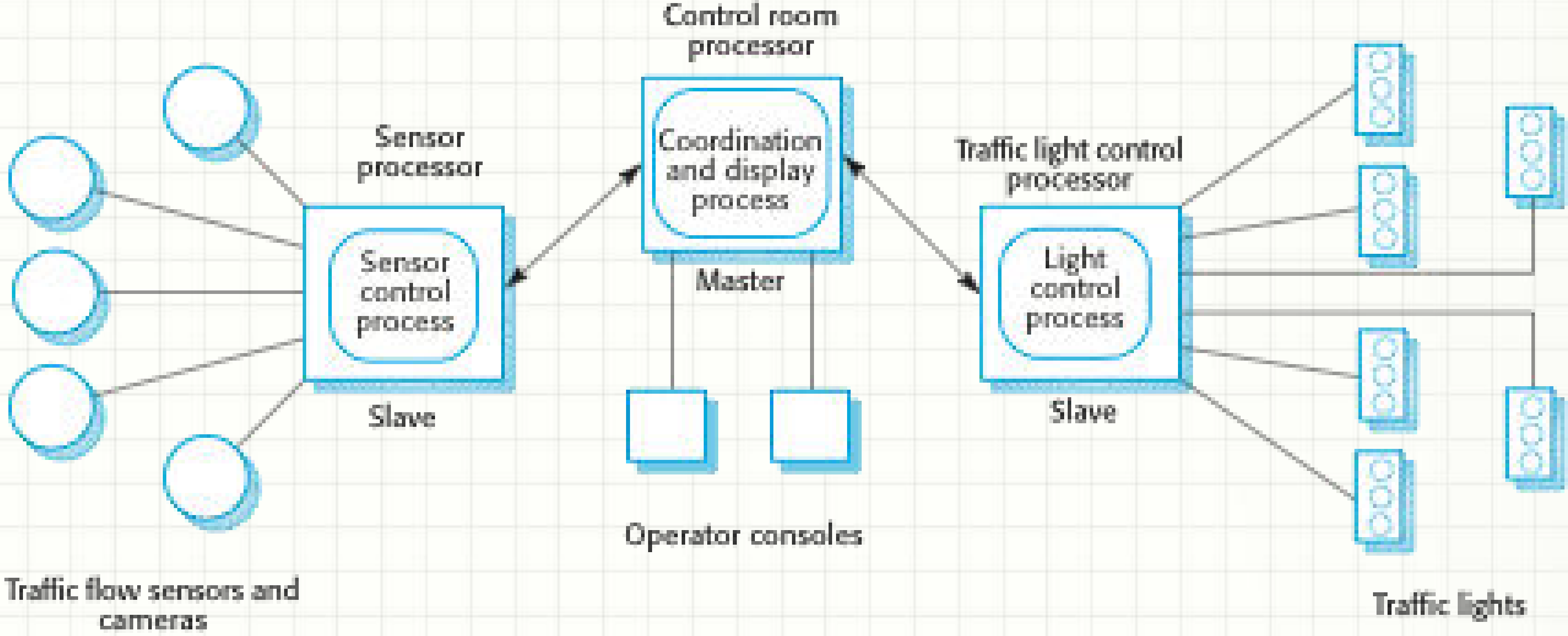
Distributed Object Architecture



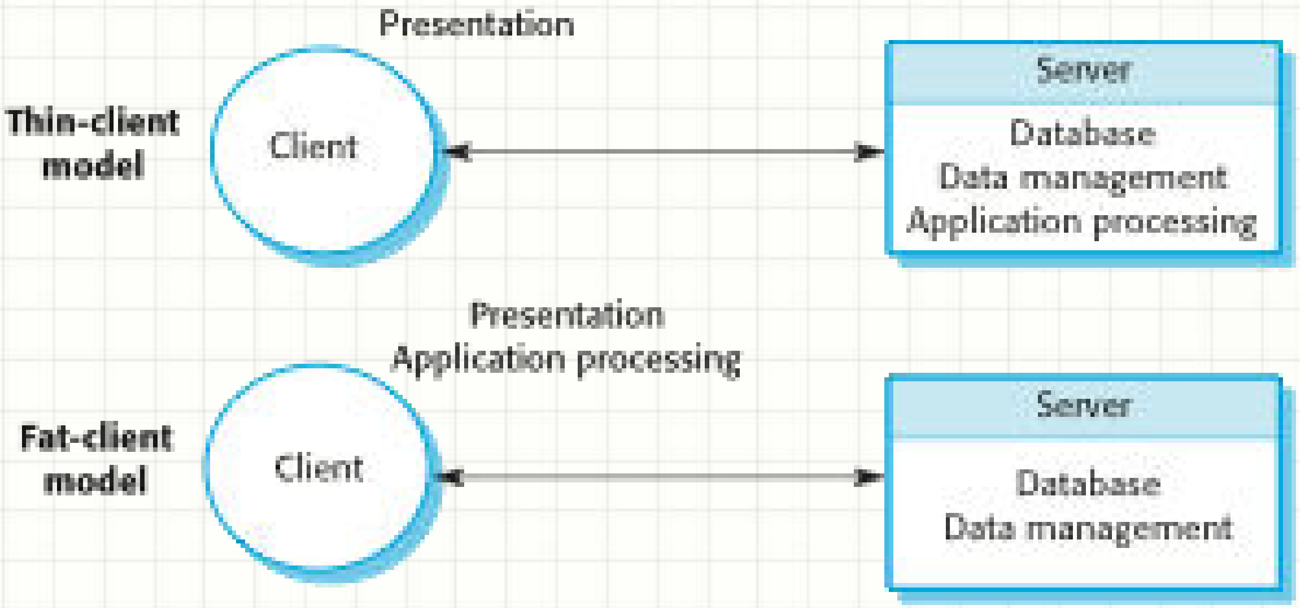
Distributed Systems Patterns

- Master-slave architecture
  - Real-time systems requiring guaranteed response times
- 2-tier client-server architecture
  - Centralized systems for security reasons
- Multi-tier C/S architecture
  - To support high-volume transaction processing
- Distributed component architecture
  - Supports combining resources from different systems
- Peer-to-peer architecture
  - Servers “introduce” peers who then work together locally

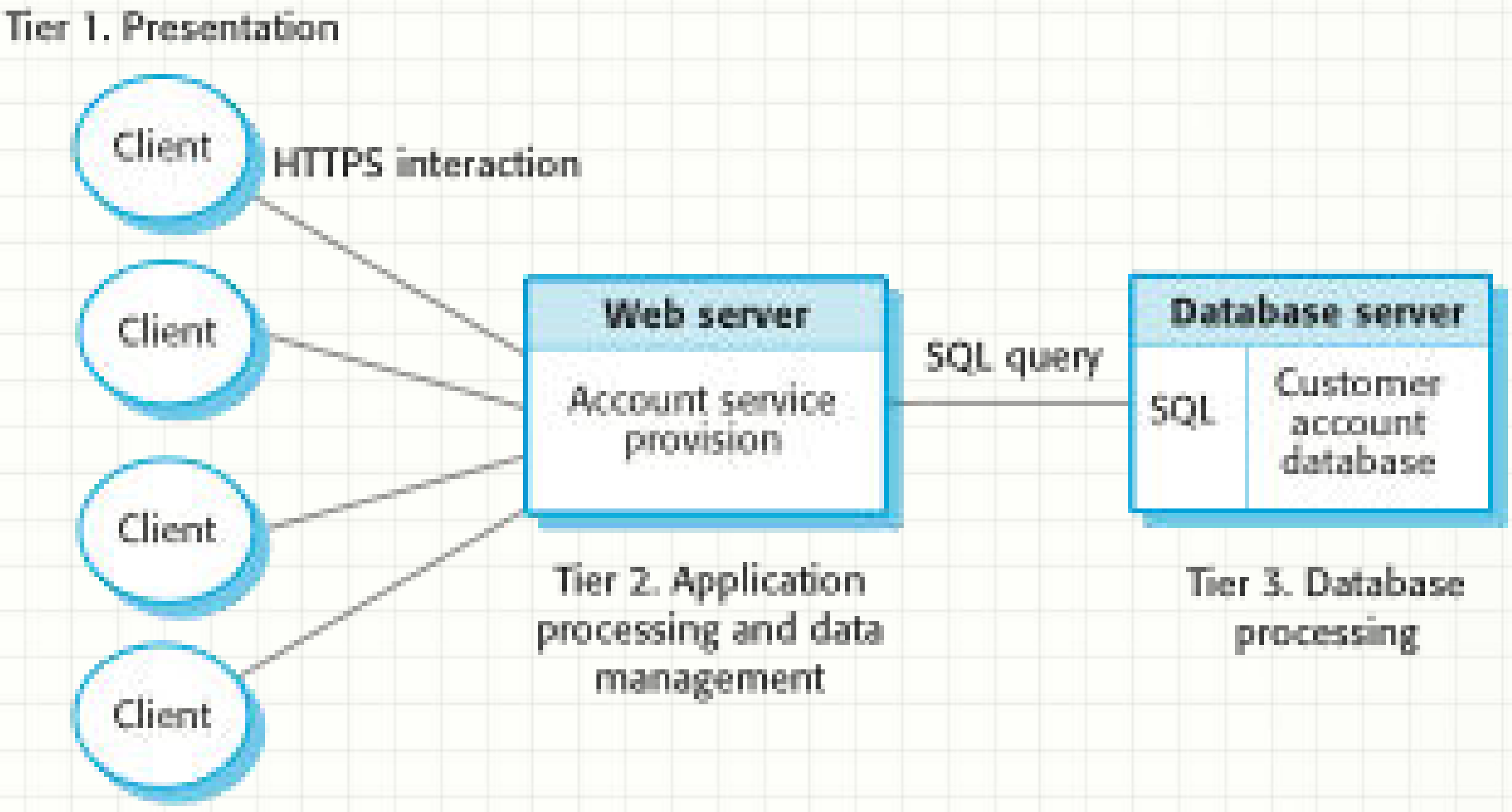
Ex. Master-Slave



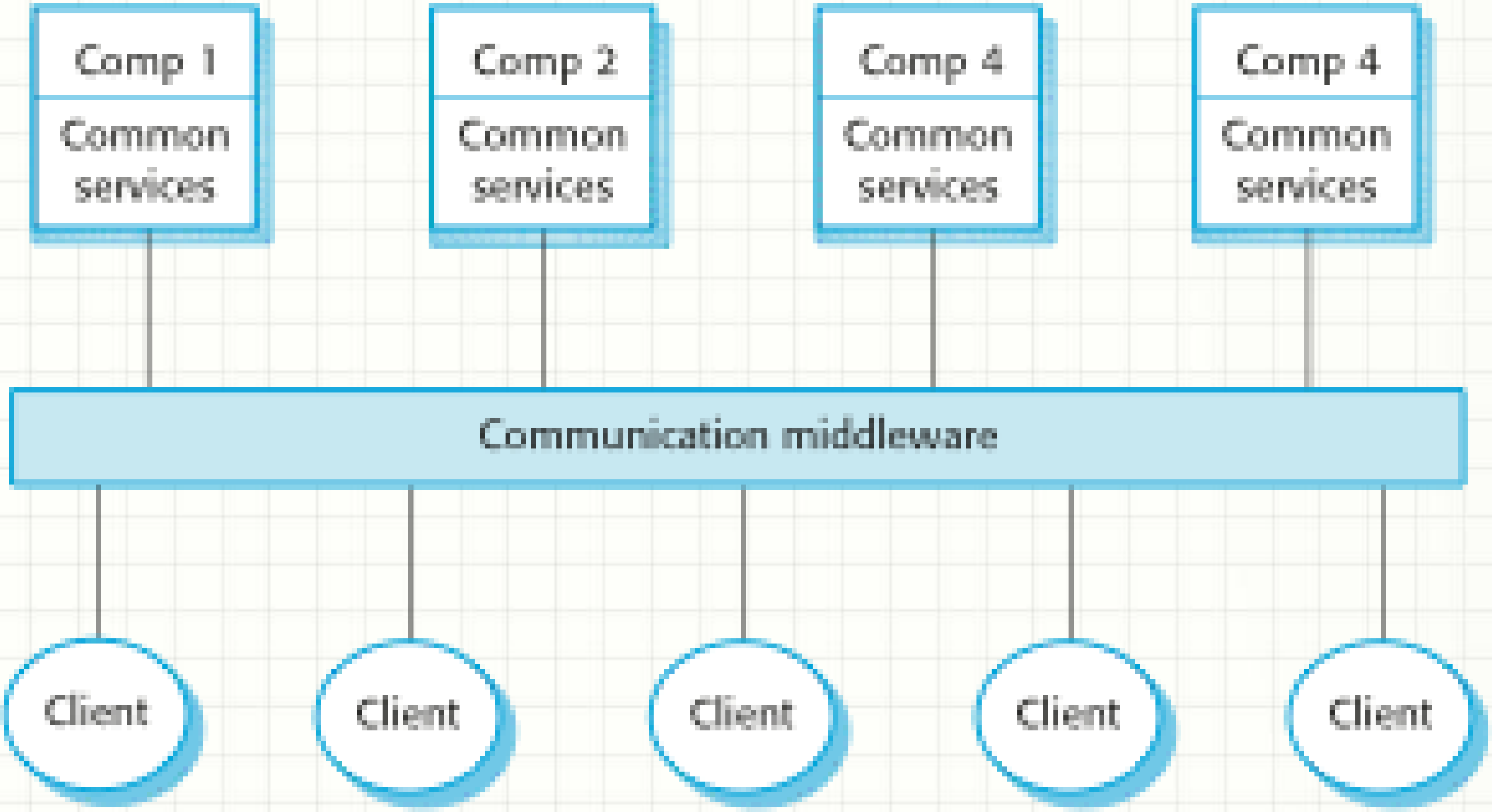
Ex. 2-tier Client-Server



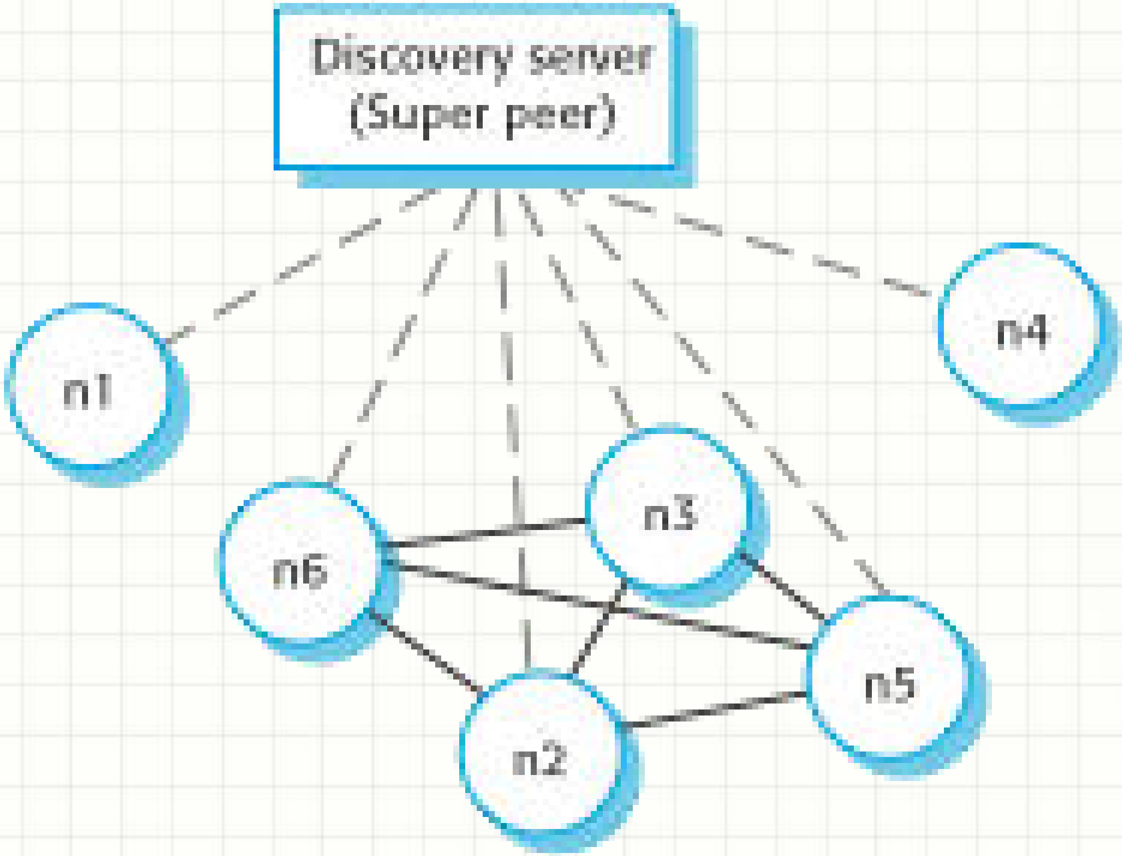
Ex. Multi-tier C/S


Tier 1. Presentation
Client
HTTPS interaction
Client
Client
Client
Web server
Account service provision
SQL query
Database server
SQL
Customer account database
Tier 2. Application processing and data management
Tier 3. Database processing

Ex. Distributed Component



Ex. Peer-to-Peer


Discovery server
(Super peer)
n1
n4
n3
n6
n5
n2