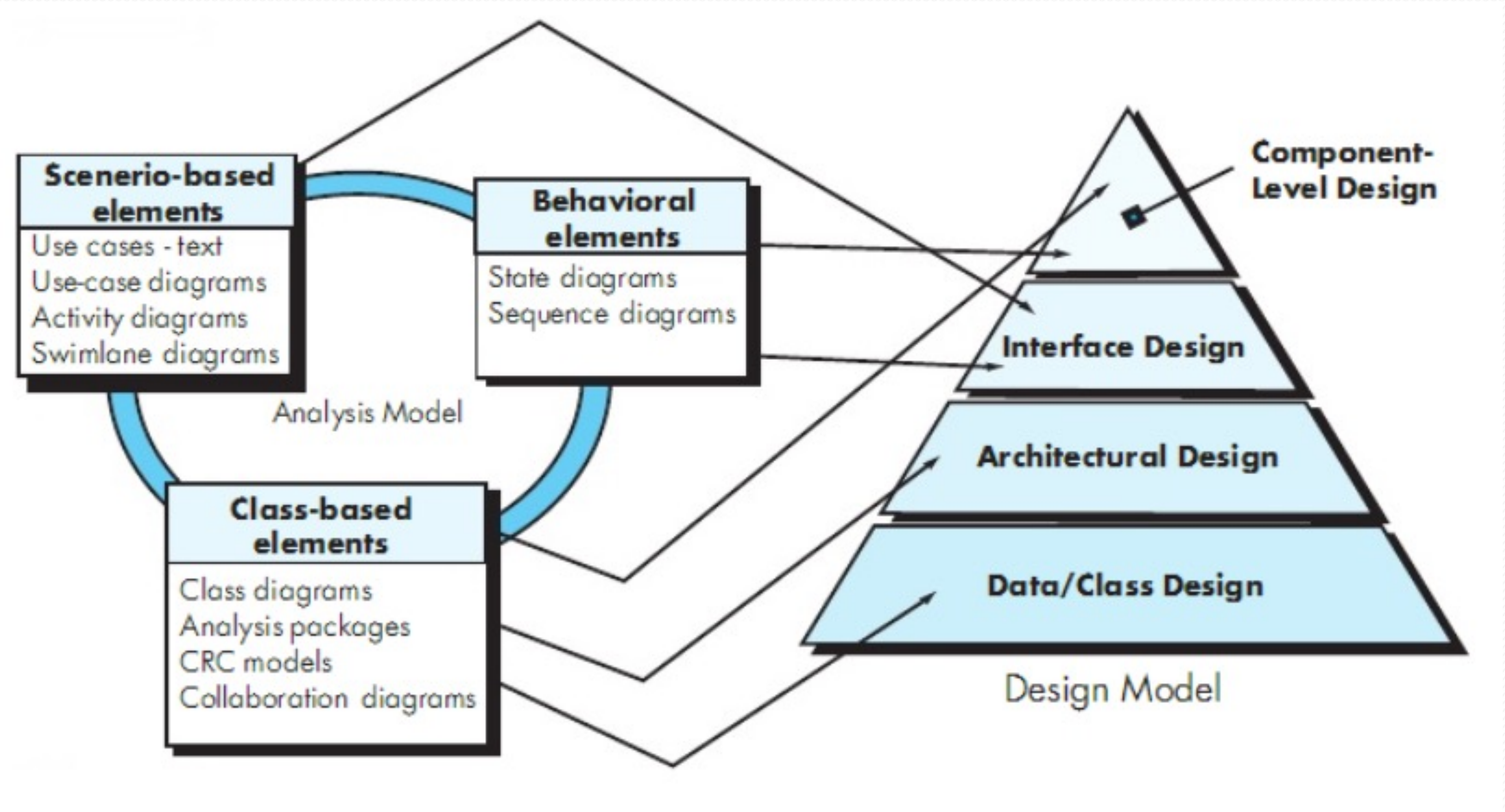
Designing System Architecture

The Design Process

- **Design** is the creative process of figuring out how to implement all of the customer's requirements; the resulting plan is also called the design
- Early design decisions address the system's architecture
- Later design decisions address how to implement the individual units

Translating Requirements to Design



Design or Requirements?

- A room for three children to play and a separate place for them to sleep
- A room for Father and Mother to sleep
- A room for cooking
- Heating for the winter and cooling for the summer
- Indoor water and electricity

Multiple Designs?

- Maximize playing area
- Minimize playing area
- Large bedrooms
- Two storey house
- Single storey house

Which is the best design?

Will a proposed solution result in modified requirements?

5.1 The Design Process

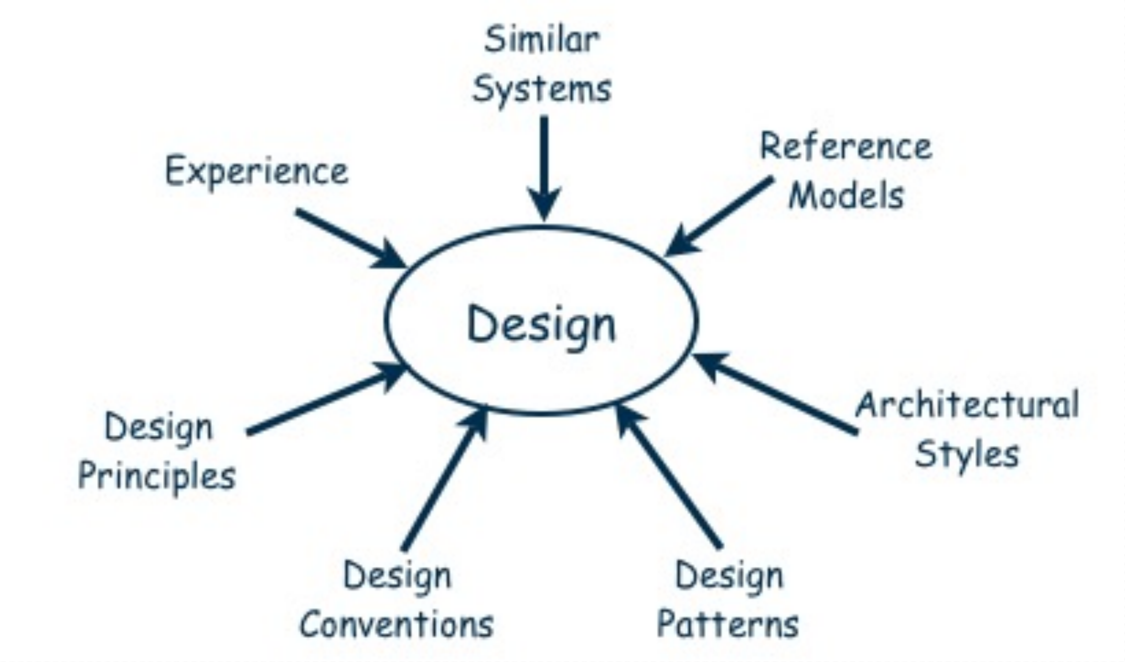
Design is a Creative Process

- Design is an intellectually challenging task
  - Numerous possibilities the system must accommodate
  - Nonfunctional design goals (e.g., ease of use, ease to maintain)
  - External factors (e.g., standard data formats, government regulations)
- We can improve our design by studying examples of good design
- Most design work is **routine design**, solve problem by reusing and adapting solutions from similar problems

5.1 The Design Process

Design is a Creative Process (continued)

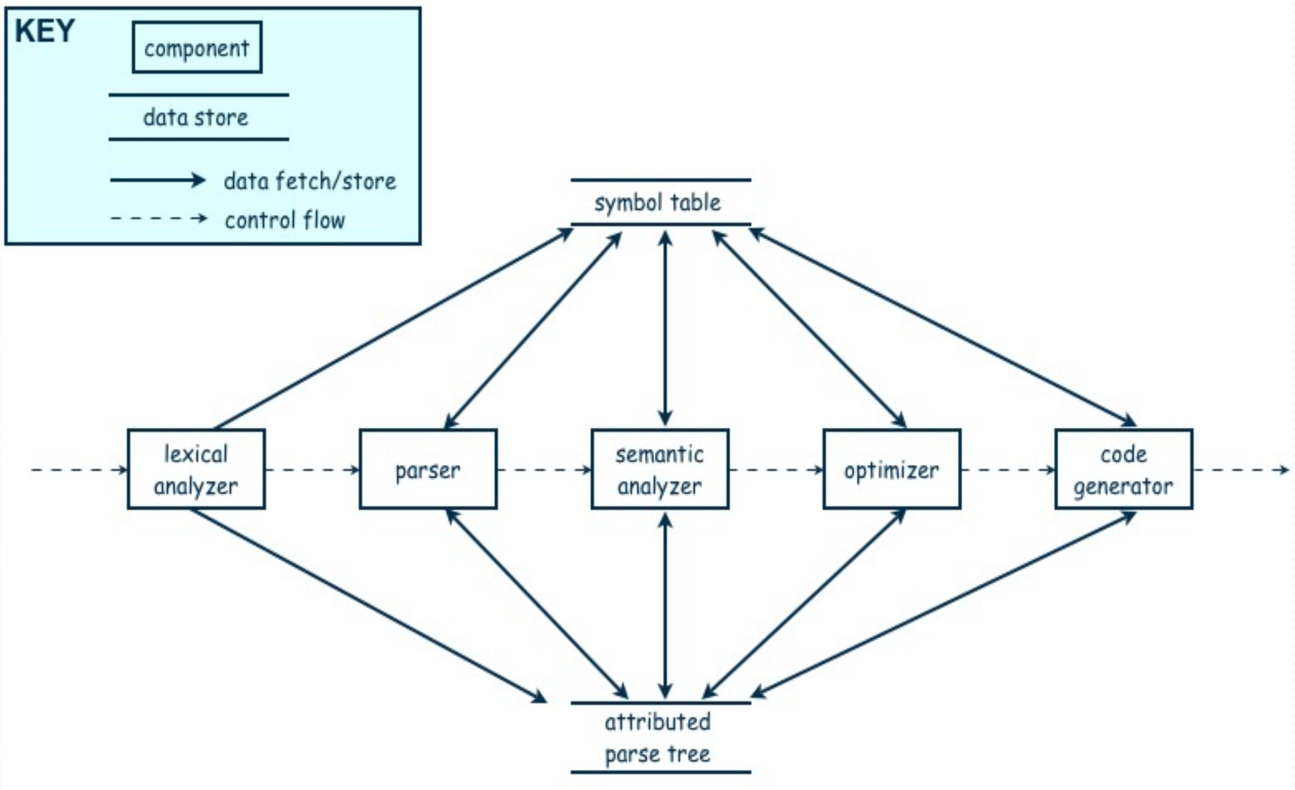
- Many ways to leverage existing solutions
  - Cloning: Borrow design/code in its entirety, with minor adjustments
  - Reference models: Generic architecture that suggests how to decompose the system



5.1 The Design Process

Design is a Creative Process (continued)

- Reference model for a compiler



5.1 The Design Process

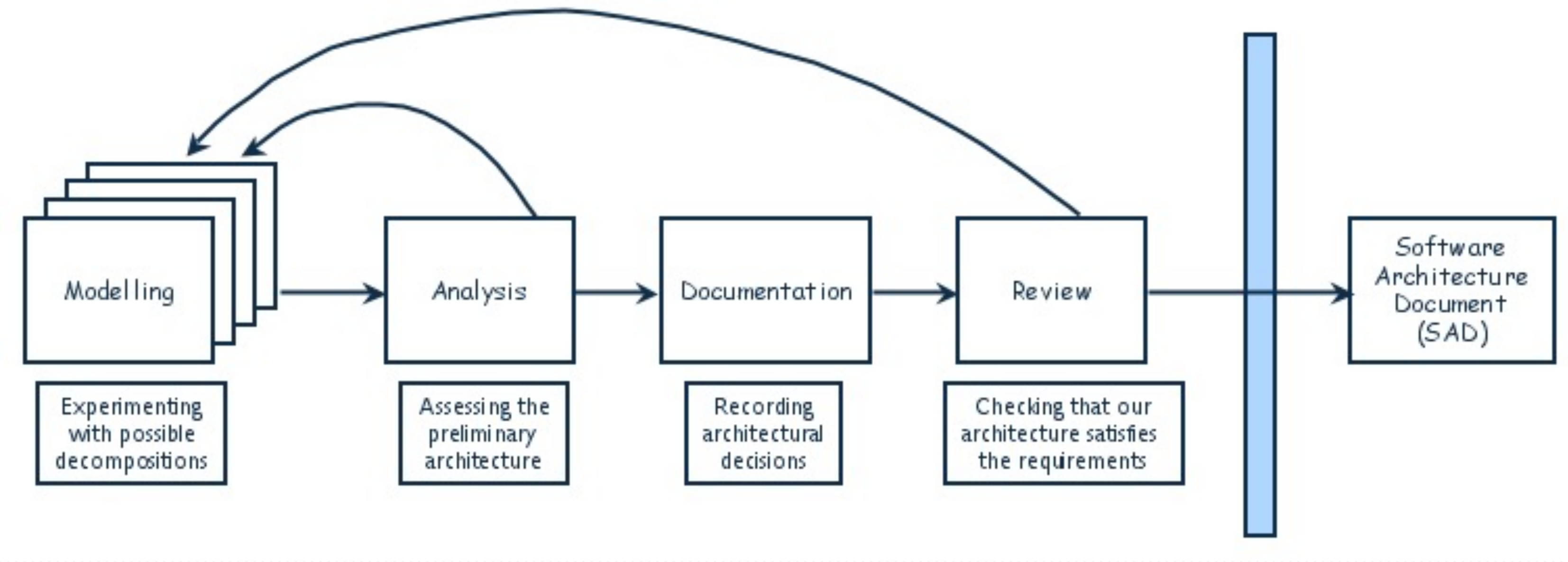
Design is a Creative Process (continued)

- More typically, a reference model will not exist for the problem
- Software architectures have generic solutions too, referred to as **architectural styles**
  - Focusing on one architectural style can create problems
  - Good design is about selecting, adapting, and integrating several architectural design styles to produce the desired result

5.1 The Design Process

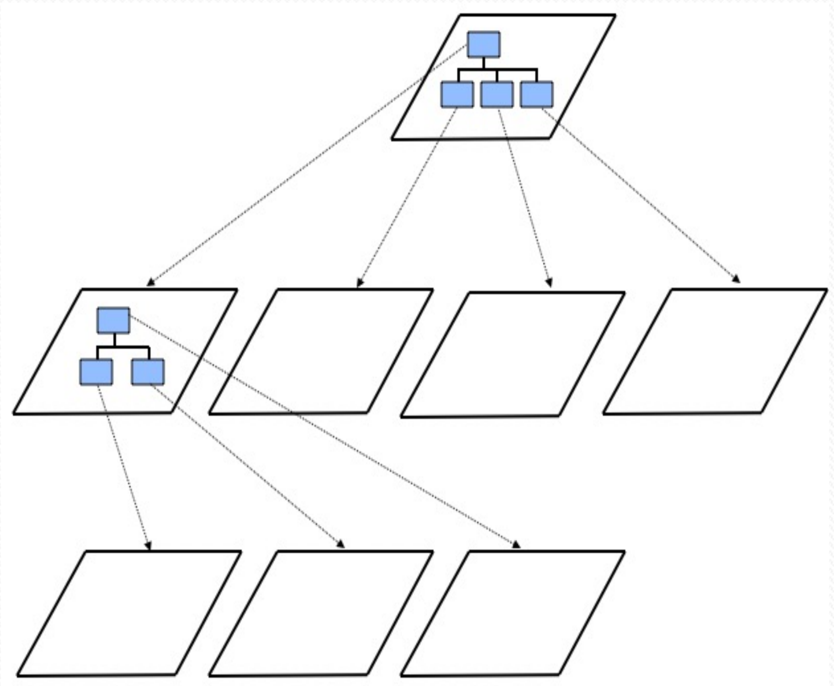
Design is an Iterative Process (continued)

- The designers move back and forth among activities involving:
  - Understanding requirements, proposing solutions, testing feasibility of a solution, presenting possibilities to the customers, documenting the design for programmers
- The final outcome is the software architecture document (SAD)


Modelling
Analysis
Documentation
Review
Software Architecture Document (SAD)
Experimenting with possible decompositions
Assessing the preliminary architecture
Recording architectural decisions
Checking that our architecture satisfies the requirements

5.3 Decomposition and Views

- High-level description of system’s key elements
- Creating a hierarchy of information with increasing details



Top level

First level of decomposition

Second level of decomposition

5.3 Decomposition and Views

Popular Design Methods

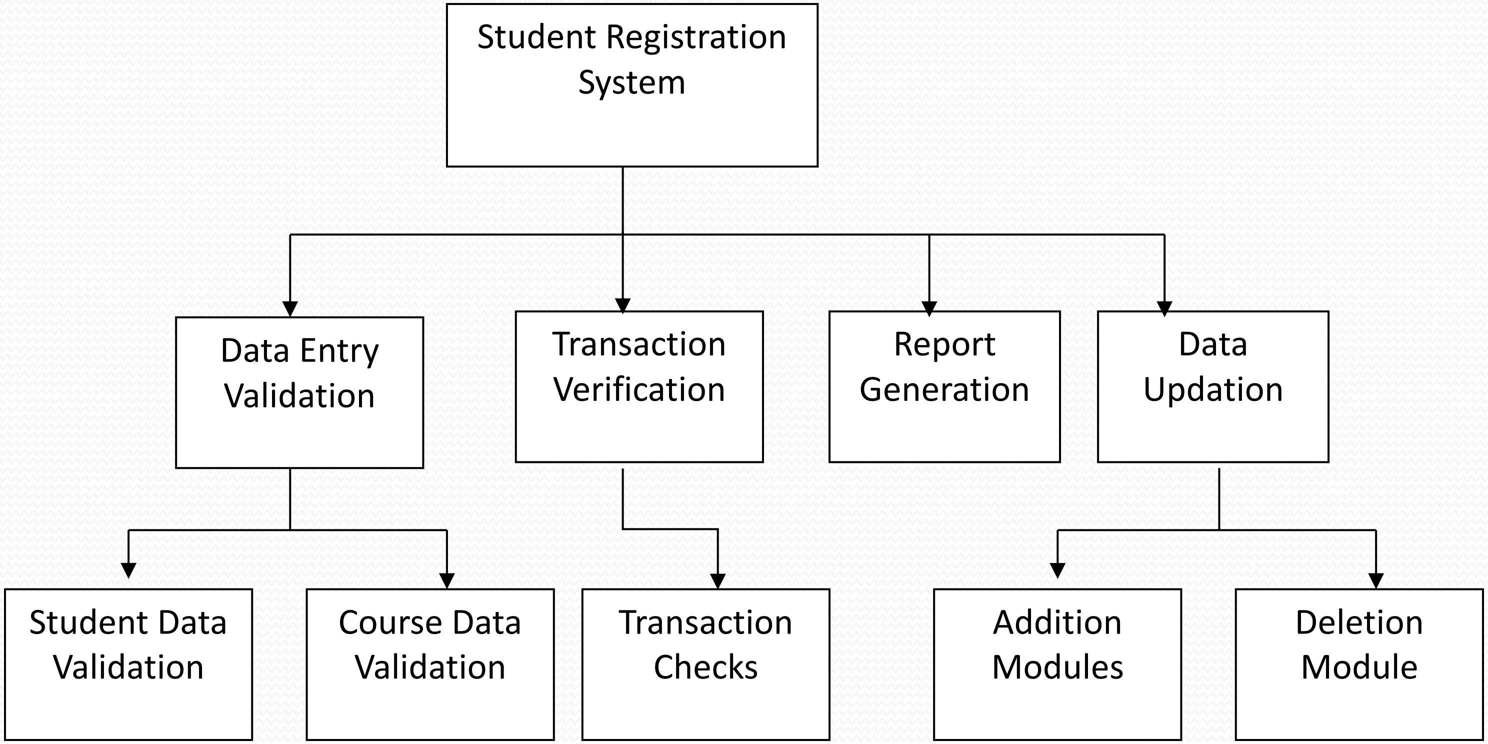
- Some design problems have no existing solutions
  - Designers must decompose to isolate key problems
- Some popular design methods:
  - Functional decomposition
  - Feature-oriented decomposition
  - Data-oriented decomposition
  - Event-oriented decomposition
  - Object-oriented design

5.3 Decomposition and Views

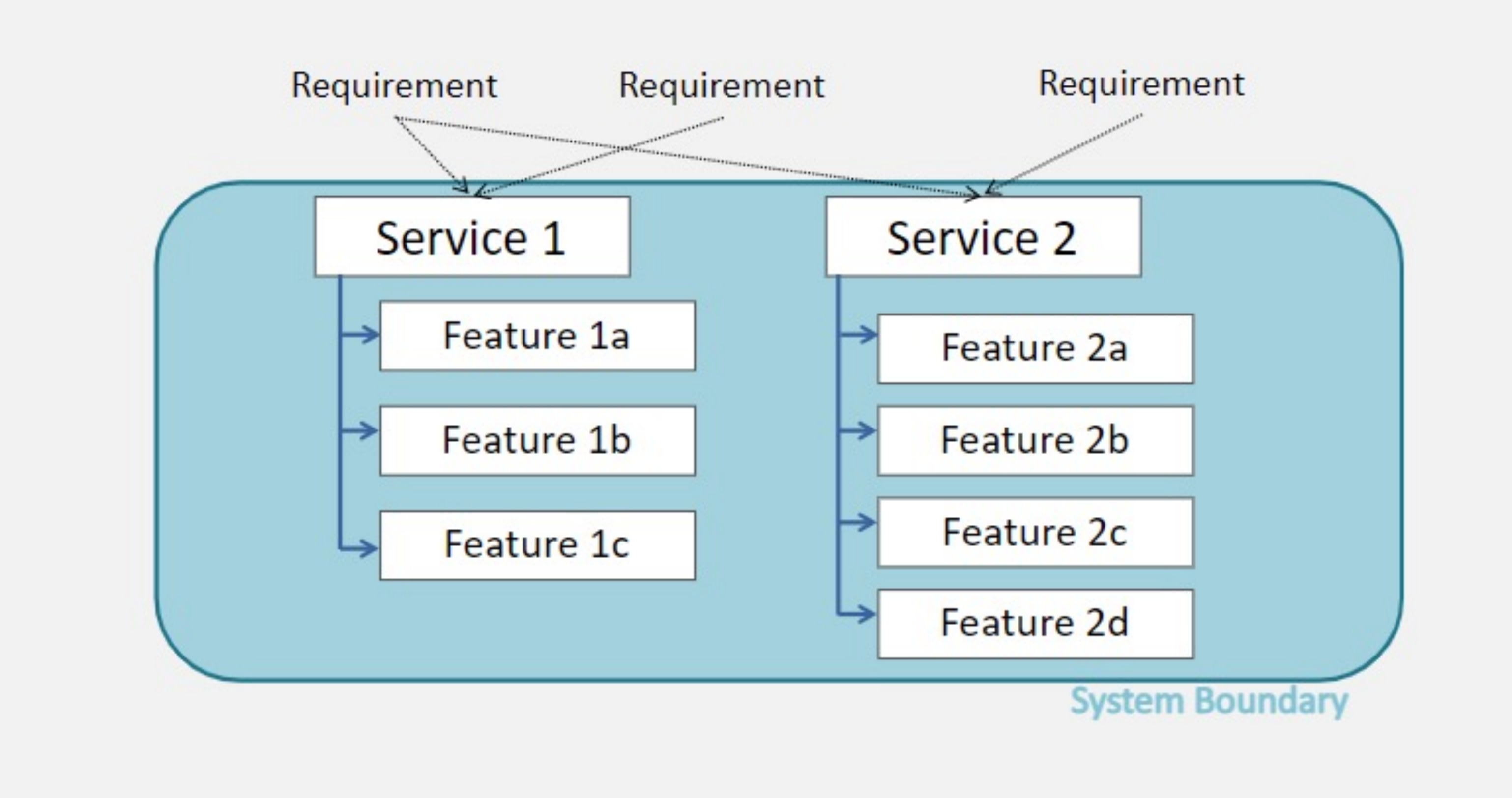
Popular Design Methods

- Functional decomposition
  - partitions functions or requirements into modules
  - begins with the functions that are listed in the requirements specification
  - lower-level designs divide these functions into subfunctions, which are then assigned to smaller modules
  - describes which modules (subfunctions) call each other

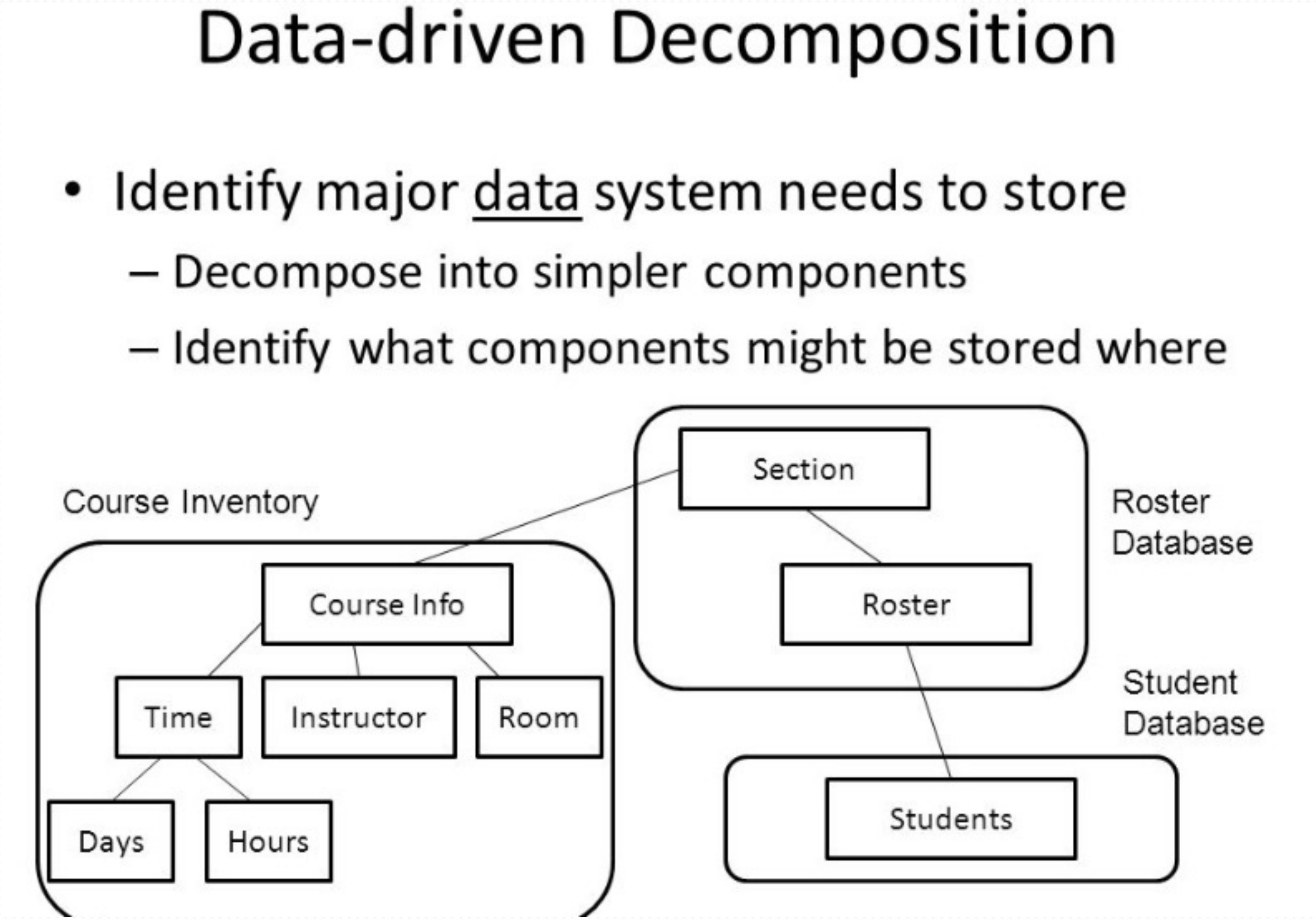
Functional Decomposition Example



Feature Oriented Decomposition Example



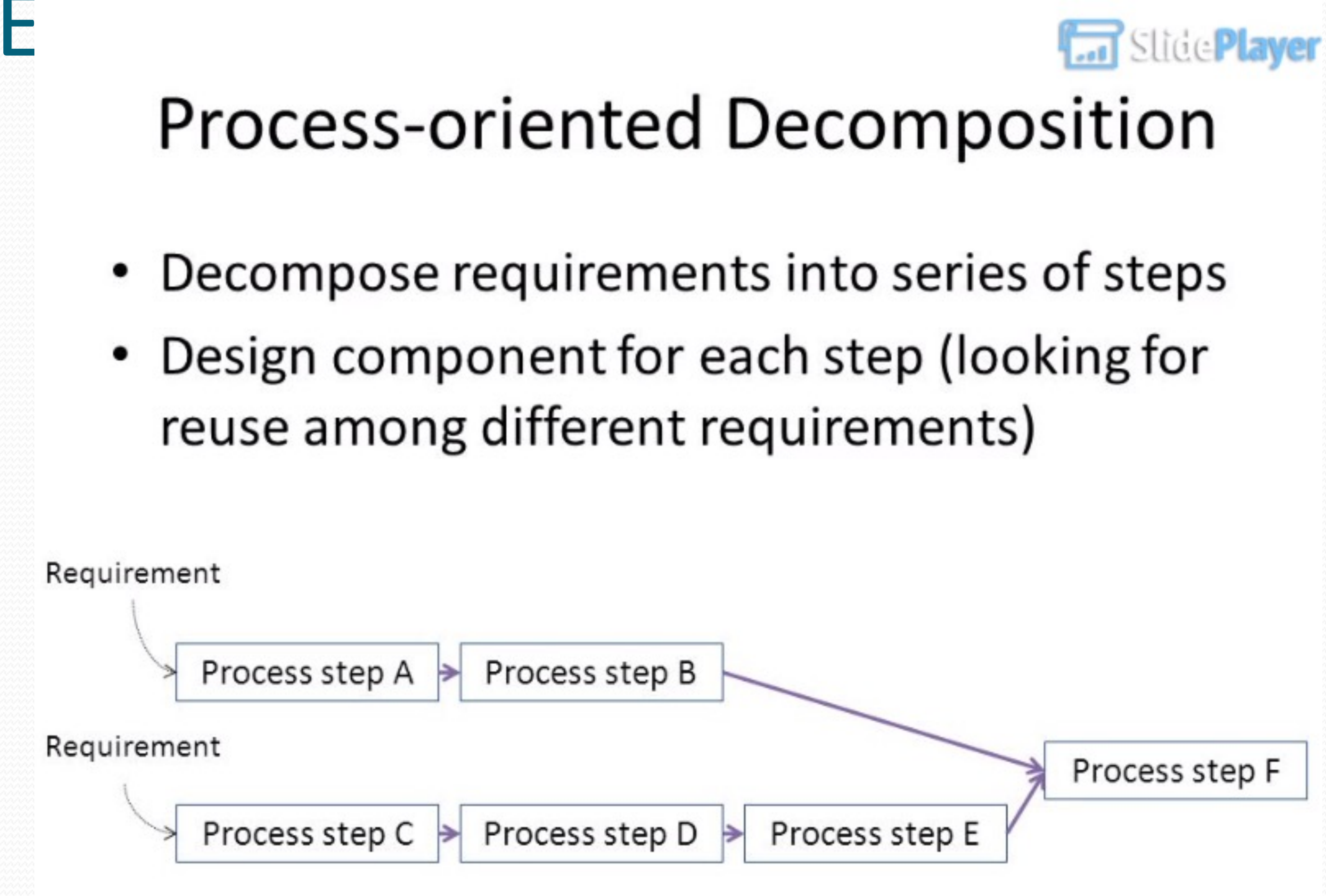
Data Driven Decomposition Example



Data-driven Decomposition

- Identify major data system needs to store
  - Decompose into simpler components
  - Identify what components might be stored where

Process –oriented Decomposition E



Process-oriented Decomposition

- Decompose requirements into series of steps
- Design component for each step (looking for reuse among different requirements)

Requirement

Process step A

Process step B

Requirement

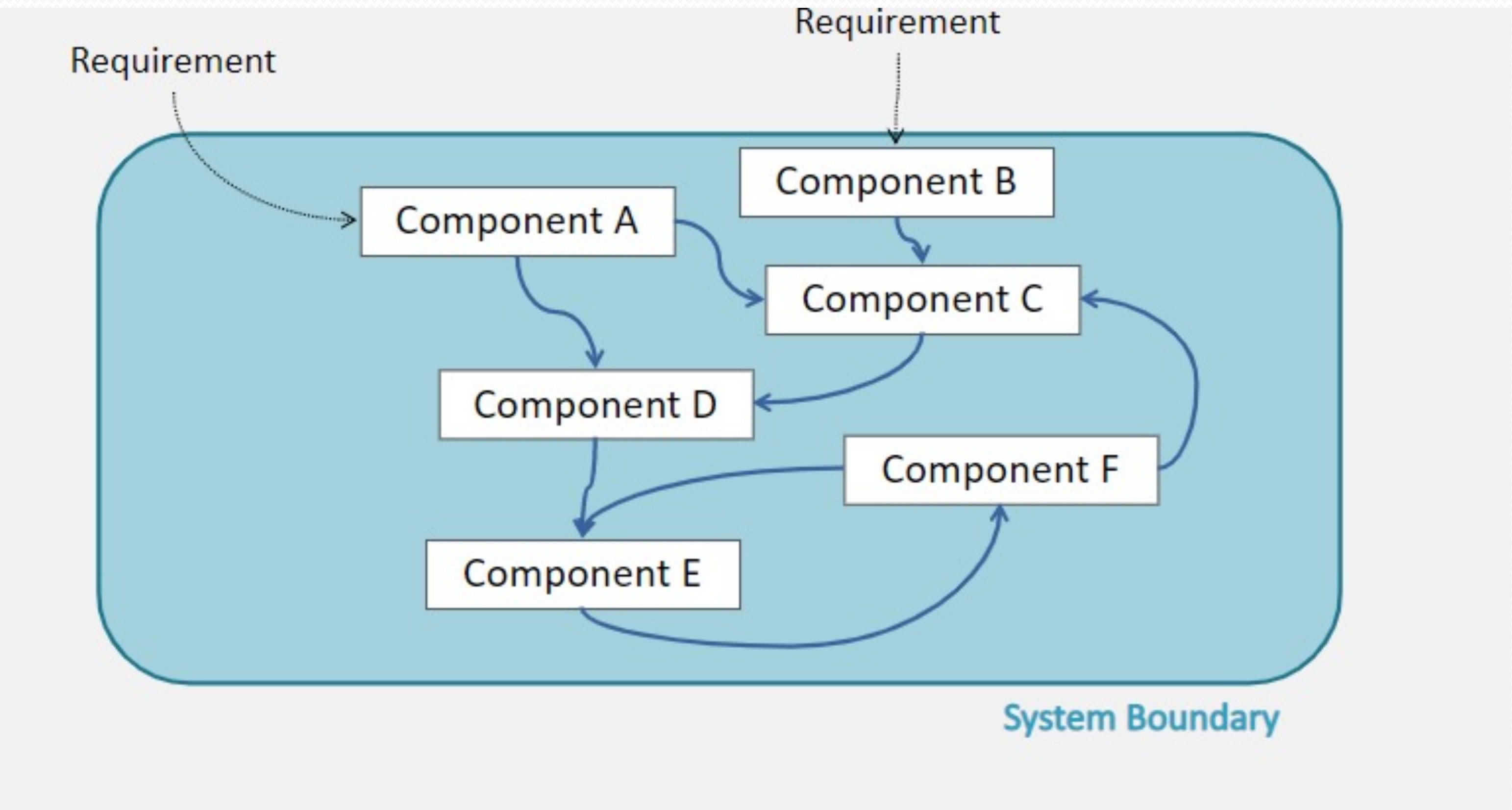
Process step C

Process step D

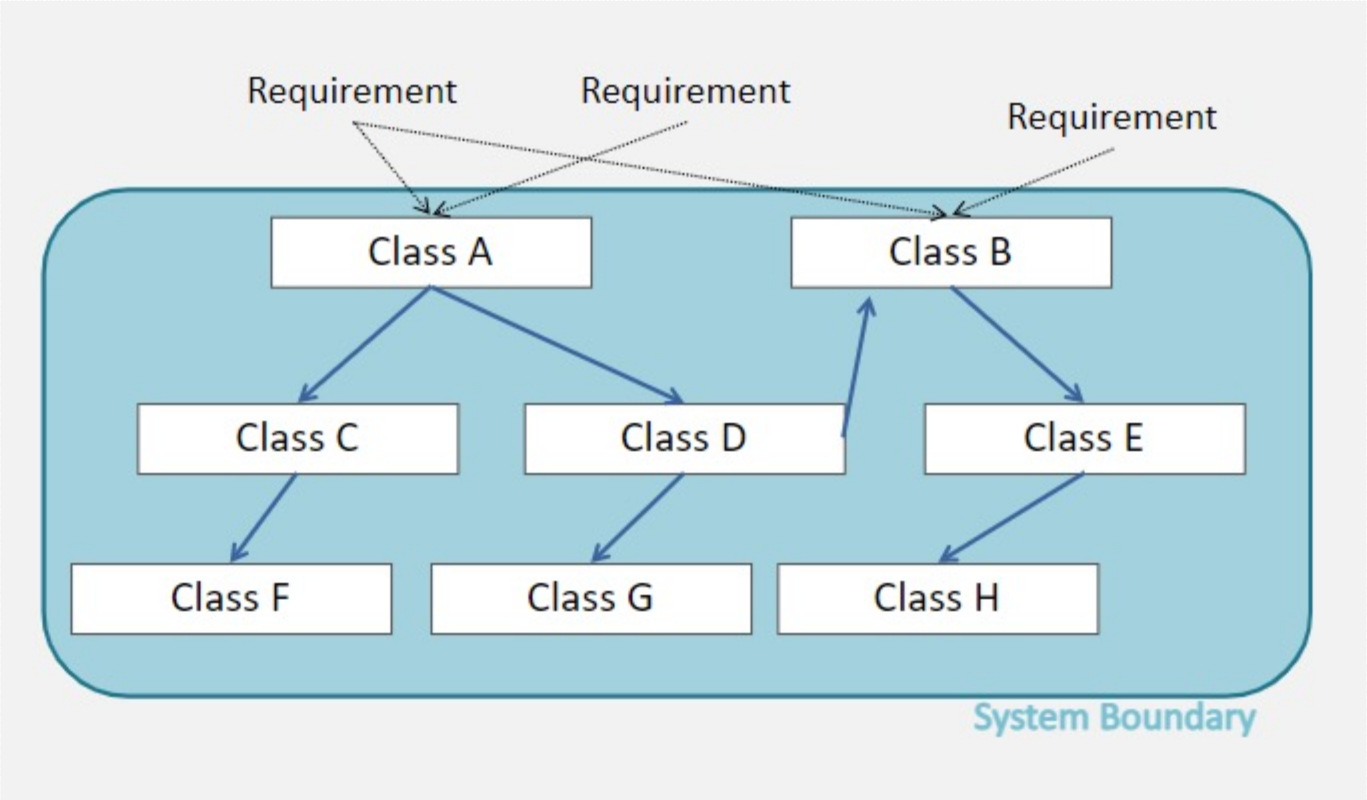
Process step E

Process step F

Event –oriented Decomposition Example



Object –oriented Decomposition Example



5.3 Decomposition and Views

Popular Design Methods (continued)

- A design is **modular** when each activity of the system is performed by exactly one software unit, and when the inputs and outputs of each software unit are well-defined

5.3 Decomposition and Views

Sidebar 5.2 Component-based Software Engineering

- **Component-based software engineering (CBSE)** is a method of software development whereby systems are created by assembling together preexisting components
- A **component** is “a self-contained piece of software with a well-defined set of interfaces” that can be developed, bought, and sold as a distinct entity
- The goal of CBSE is to support the rapid development of new systems, by reducing development to component integration, and to ease the maintenance of such systems by reducing maintenance to component replacement
- At this point, CBSE is still more of a goal than a reality with considerable on-going research

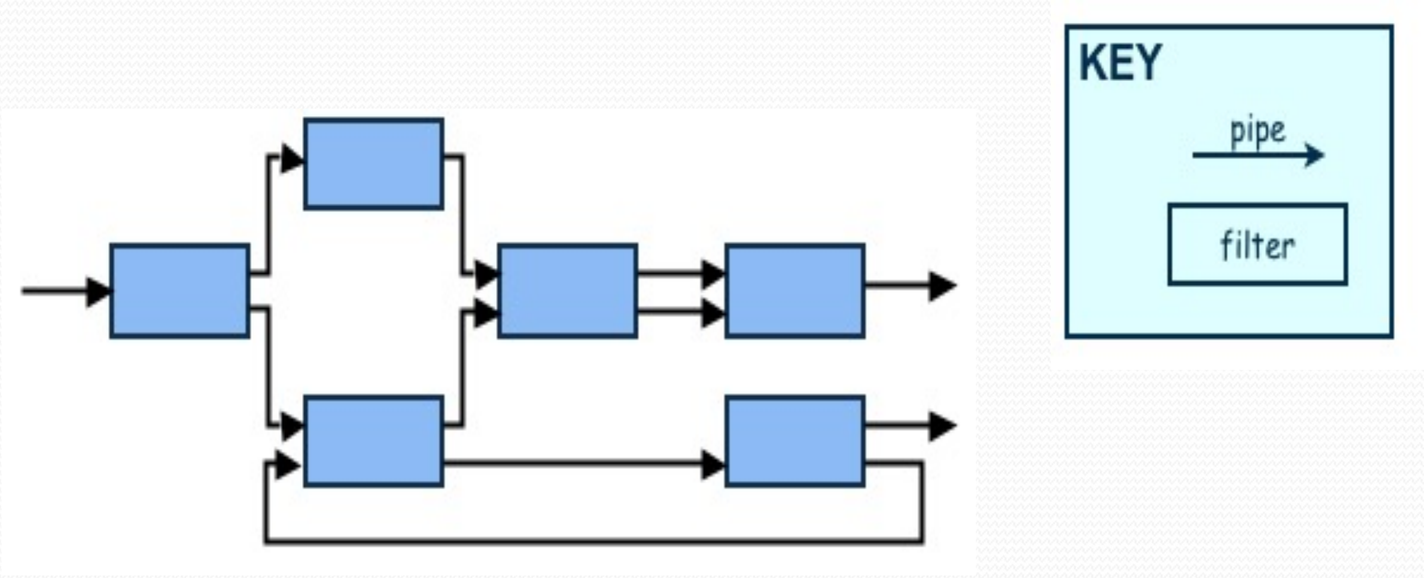
5.4 Architectural Styles and Strategies

- Pipe-and-Filter
- Client-Server
- Peer-to-Peer
- Publish-Subscribe
- Repositories
- Layering

5.4 Architectural Styles and Strategies

Pipe-and-Filter

- The system has
  - Streams of data (pipe) for input and output
  - Transformation of the data (filter)



5.4 Architectural Styles and Strategies

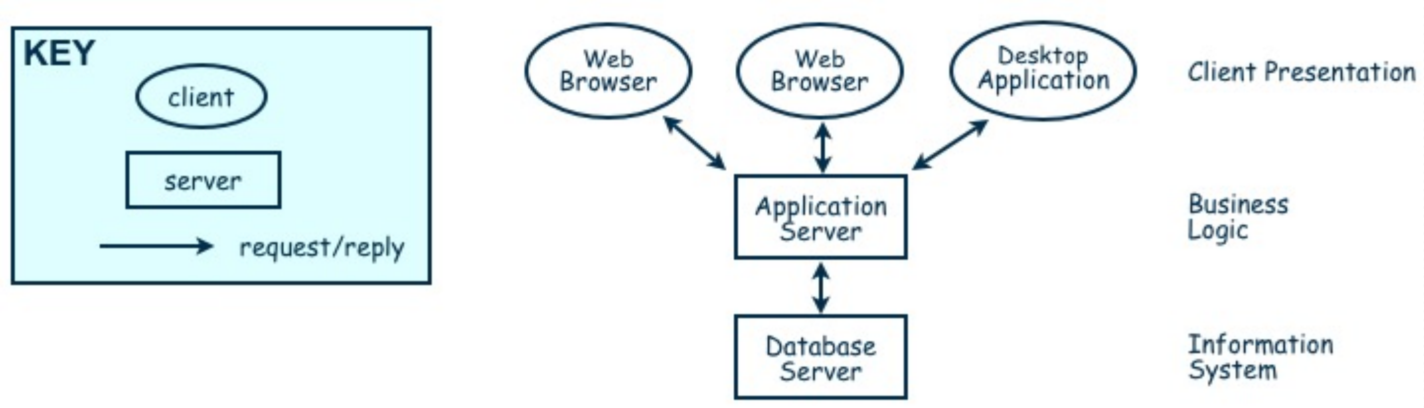
Pipe-and-Filter (continued)

- Several important properties
  - The designer can understand the entire system's effect on input and output as the composition of the filters
  - The filters can be reused easily on other systems
  - System evolution is simple
  - Allow concurrent execution of filters
- Drawbacks
  - Not good for handling interactive application

5.4 Architectural Styles and Strategies

Client-Server

- Two types of components:
  - Server components offer services
  - Clients access them using a request/reply protocol
- Client may send the server an executable function, called a callback
  - The server subsequently calls under specific circumstances

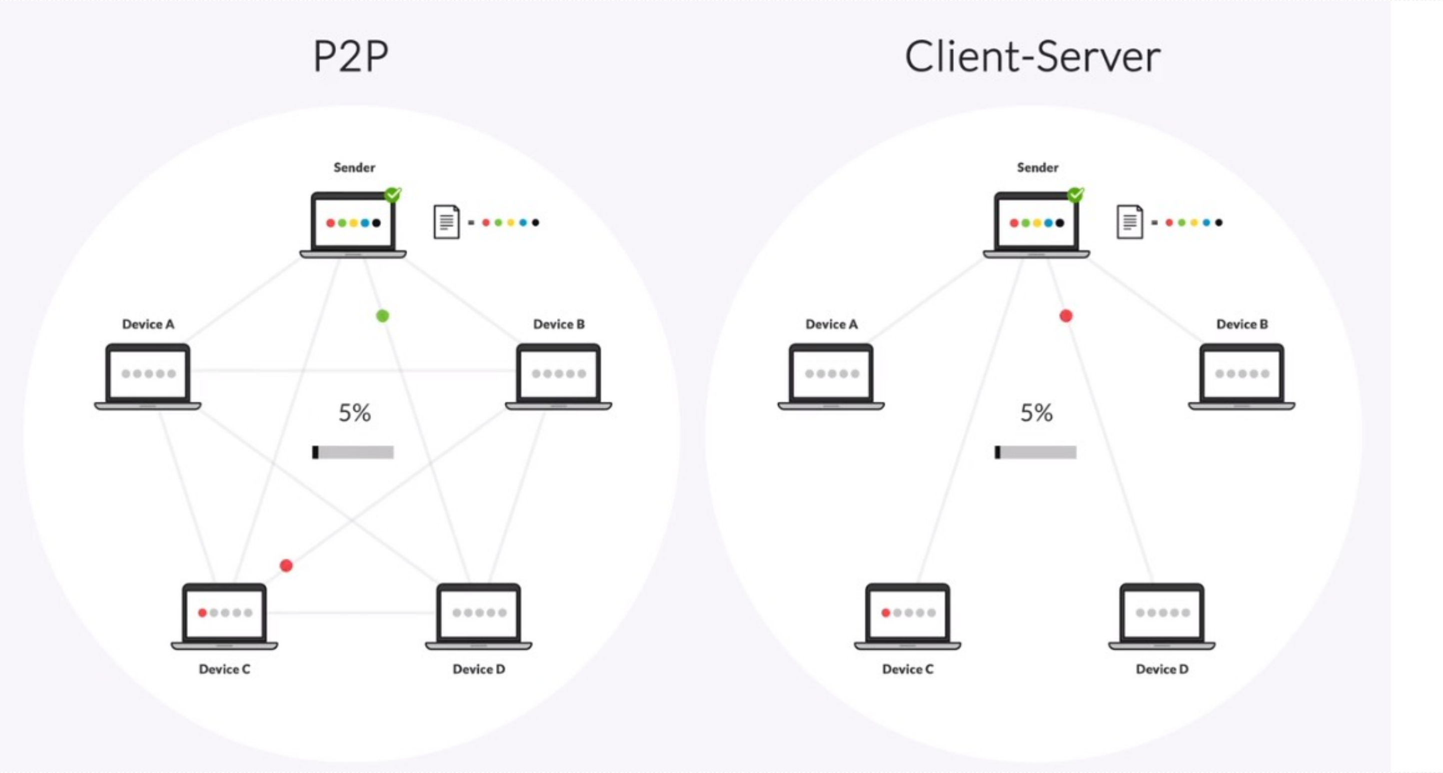


5.4 Architectural Styles and Strategies

Peer-to-Peer (P2P)

- Each component acts as its own process and acts as both a client and a server to other peer components.
- Any component can initiate a request to any other peer component.
- Characteristics
  - Scale up well
  - Increased system capabilities
  - Highly tolerant of failures
- Examples: Napster, Kaaza, BitTorrent

5.4 Architectural Styles and Strategies



5.4 Architectural Styles and Strategies

Peer-to-Peer (P2P)

- When NOT to use:
  - When file contents change frequently (e.g. prices)
  - When sharing speed has importance (e.g. large files are needed quickly)
  - File quality is critical
  - When trust between peers is required (e.g. the content is protected)

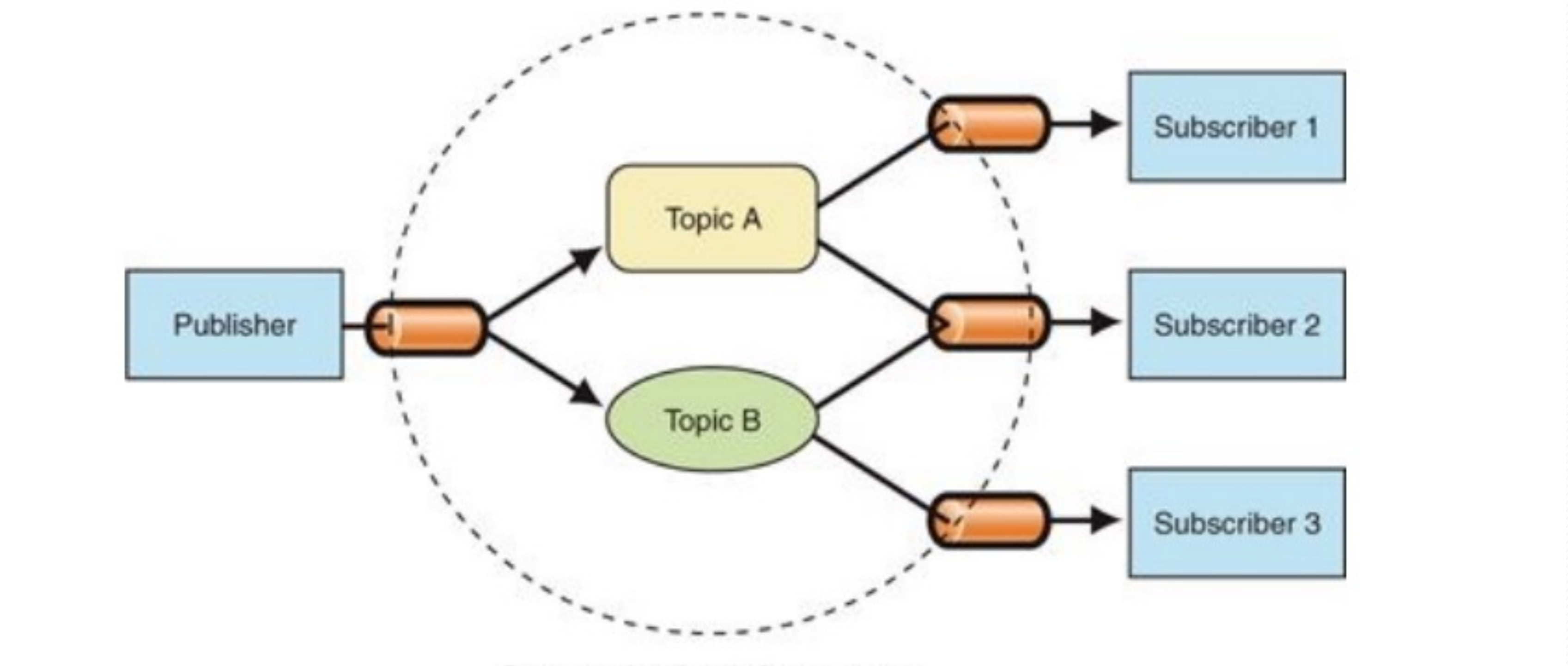
5.4 Architectural Styles and Strategies

Publish-Subscribe

- interact by broadcasting and reacting to events Components
  - Component expresses interest in an event by subscribing to it
  - When another component announces (publishes) that event has taken place, subscribing components are notified
  - Implicit invocation is a common form of publish-subscribe architecture
    - Registering: subscribing component associates one of its procedures with each event of interest (called the procedure)
- Characteristics
  - Strong support for evolution and customization
  - Easy to reuse components in other event-driven systems
  - Need shared repository for components to share persistent data
  - Difficult to test

5.4 Architectural Styles and Strategies

Publish-Subscribe



5.4 Architectural Styles and Strategies

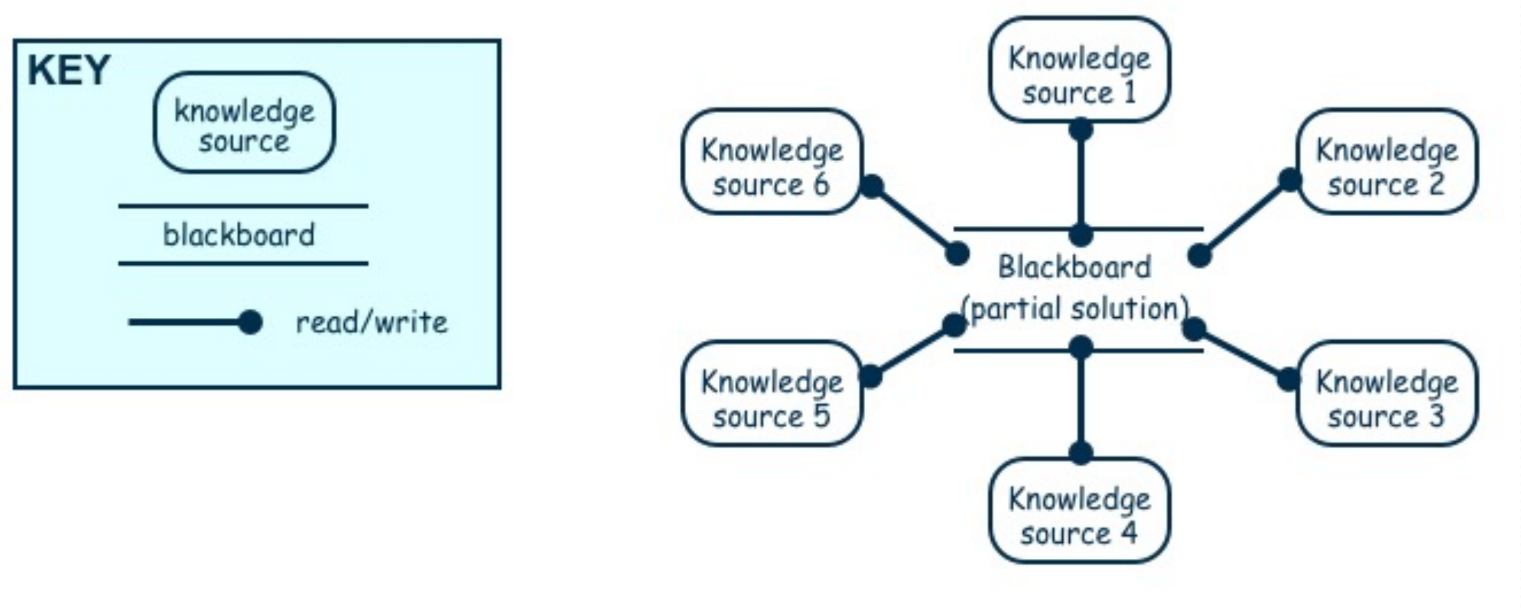
Repositories

- Two components
  - A central data store
  - A collection of components that operate on it to store, retrieve, and update information (data accessors)
- The challenge is deciding how the components will interact
  - A traditional database: transactions trigger process execution
  - A blackboard: the central store controls the triggering process
  - Knowledge sources: information about the current state of the system’s execution that triggers the execution of individual data accessors

5.4 Architectural Styles and Strategies

Repositories (continued)

- Openness?
  - Data representation is made available to various programmers (vendors) so they can build tools to access the repository
  - But also a disadvantage: the data format must be acceptable to all components



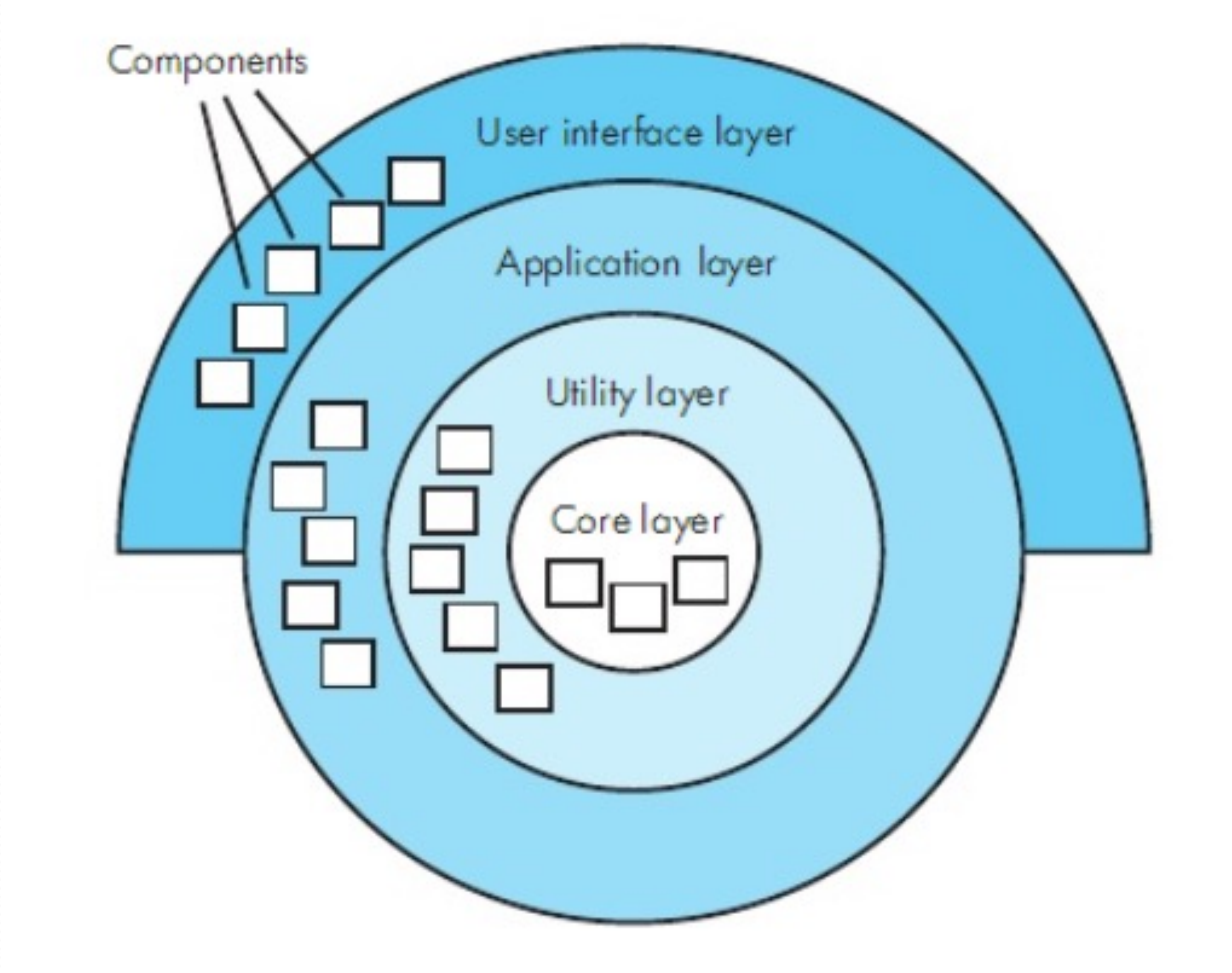
5.4 Architectural Styles and Strategies

Layering

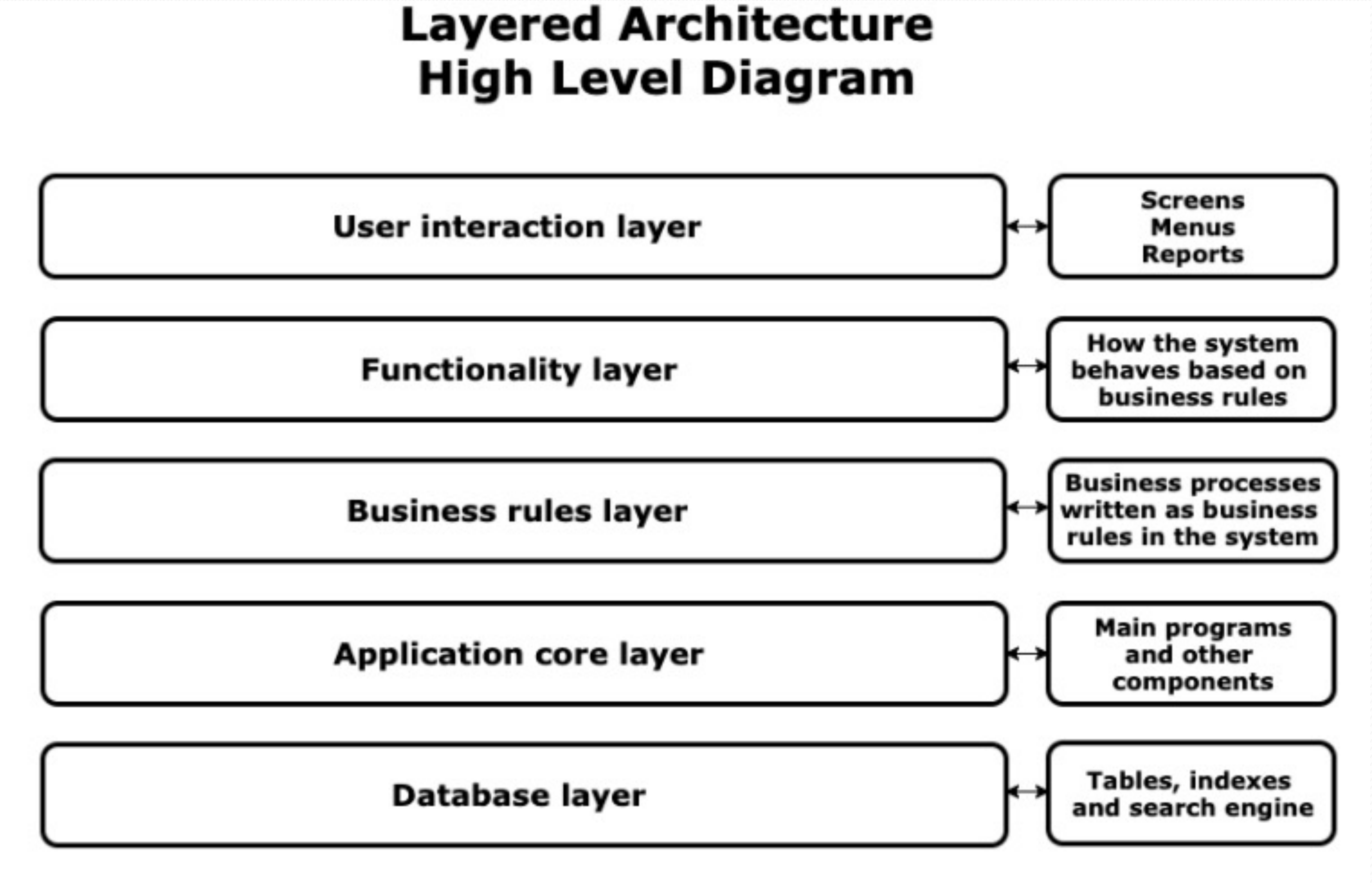
- Layers are hierarchical
  - Each layer provides service to the one outside it and acts as a client to the layer inside it
  - Layer bridging: allowing a layer to access the services of layers below its lower neighbor
- The design includes protocols
  - Explain how each pair of layers will interact
- Advantages
  - High levels of abstraction
  - Relatively easy to add and modify a layer
- Disadvantages
  - Not always easy to structure system layers
  - System performance may suffer from the extra coordination among layers

5.4 Architectural Styles and Strategies

Example of Layering System



5.4 Architectural Styles and Strategies



Layered Architecture High Level Diagram

User interaction layer ↔ Screens Menus Reports

Functionality layer ↔ How the system behaves based on business rules

Business rules layer ↔ Business processes written as business rules in the system

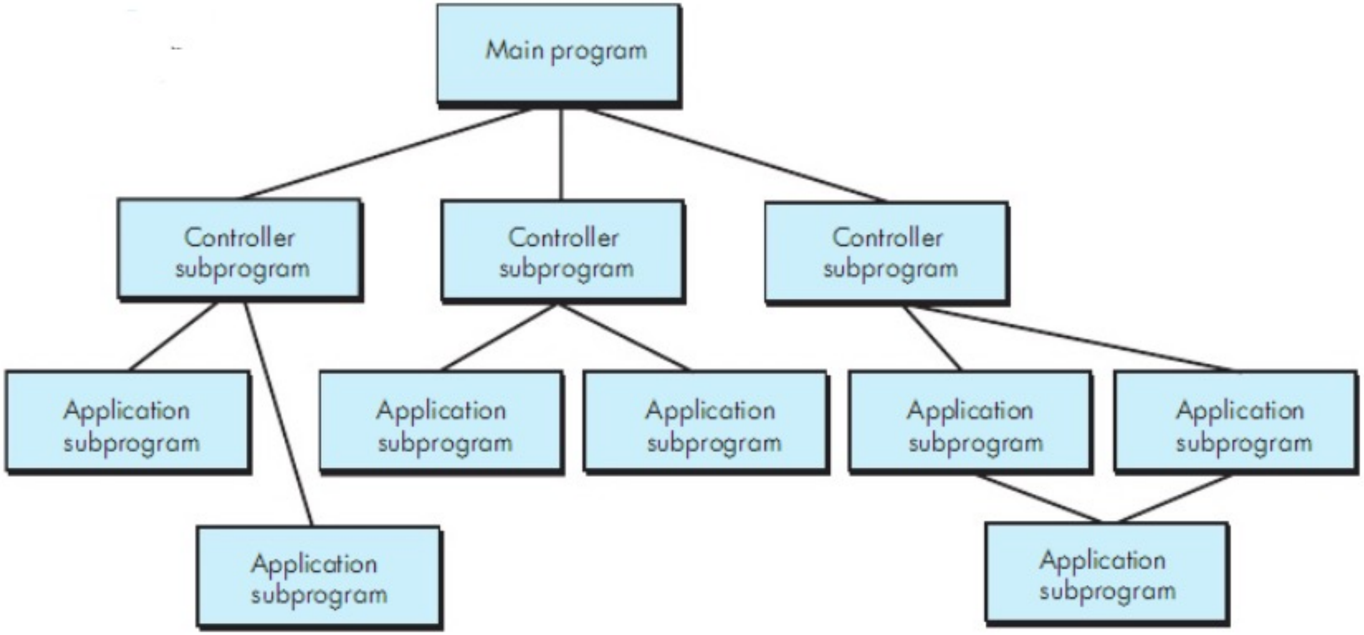
Application core layer ↔ Main programs and other components

Database layer ↔ Tables, indexes and search engine

Architecture Styles and Strategies

Call and Return

- Main program/sub program
- Easy to scale and modify



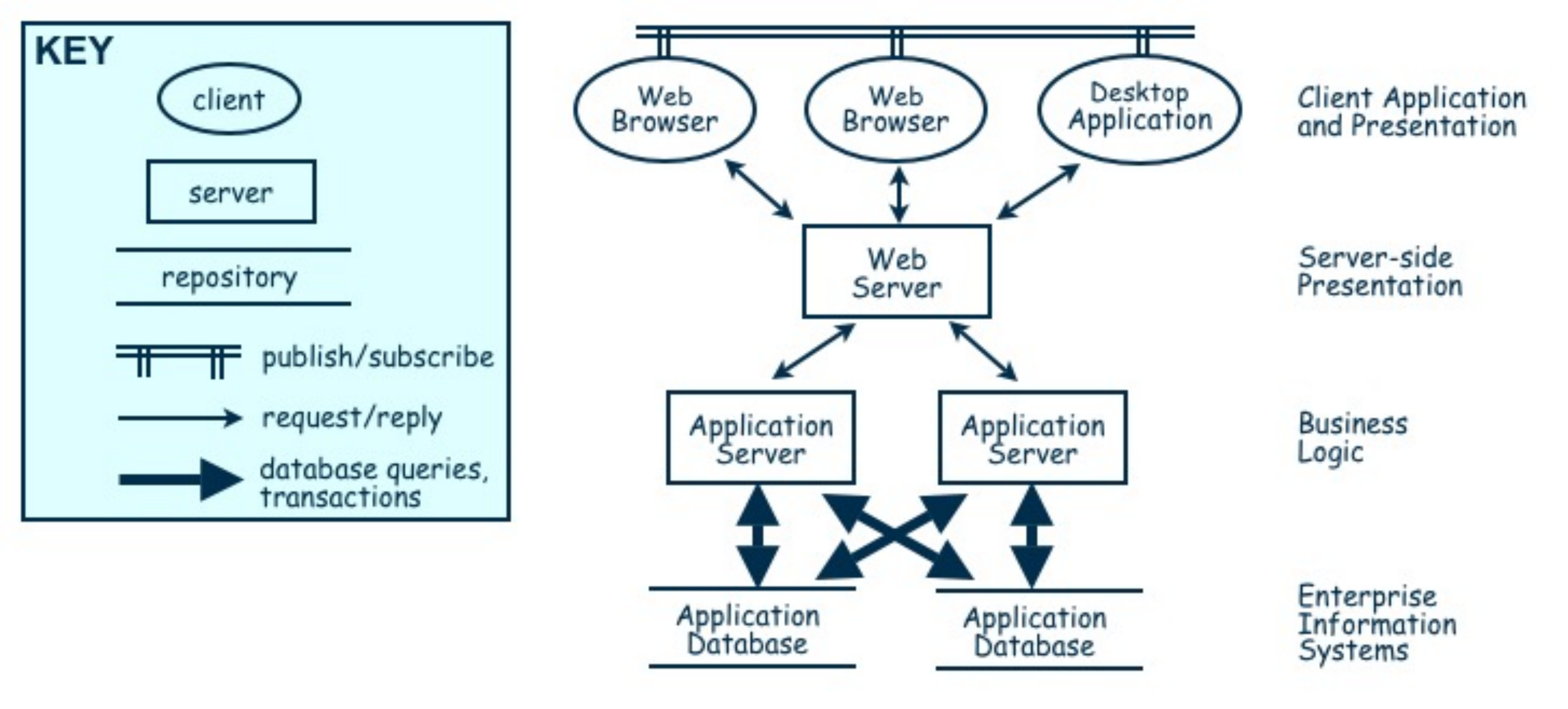
5.4 Architectural Styles and Strategies

Combining Architectural Styles

- Actual software architectures rarely based on purely one style
- Architectural styles can be combined in several ways
  - Use different styles at different layers (e.g., overall client-server architecture with server component decomposed into layers)
  - Use mixture of styles to model different components or types of interaction (e.g., client components interact with one another using publish-subscribe communications
- If architecture is expressed as collection of models, documentation must be created to show relation between models

Architectural Styles and Strategies

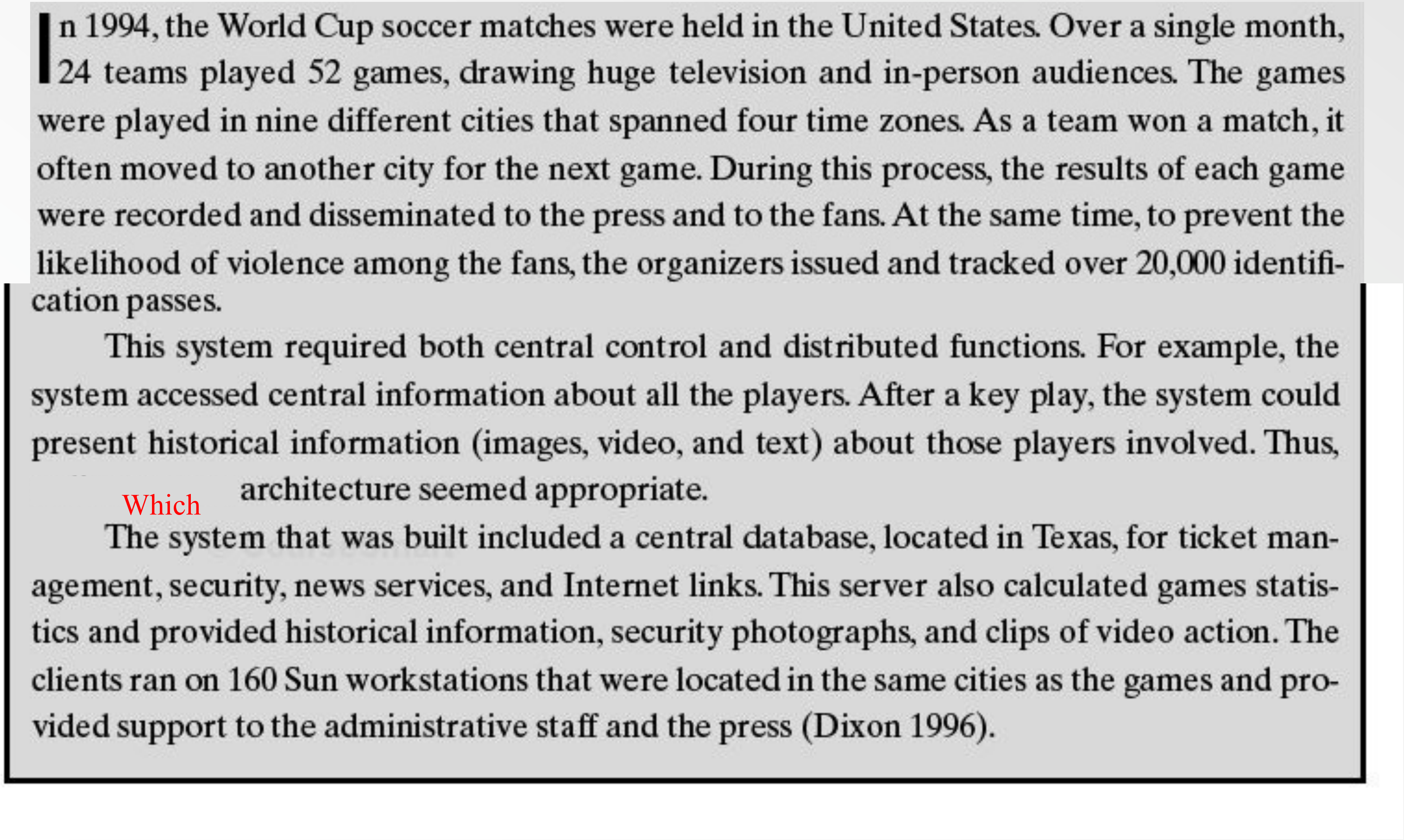
Combination of WHICH Architecture Styles?



Combination of Publish-Subscribe, Client-Server, Layering, Repository

Architectural Styles and Strategies

The World Cup System



In 1994, the World Cup soccer matches were held in the United States. Over a single month, 24 teams played 52 games, drawing huge television and in-person audiences. The games were played in nine different cities that spanned four time zones. As a team won a match, it often moved to another city for the next game. During this process, the results of each game were recorded and disseminated to the press and to the fans. At the same time, to prevent the likelihood of violence among the fans, the organizers issued and tracked over 20,000 identification passes.

This system required both central control and distributed functions. For example, the system accessed central information about all the players. After a key play, the system could present historical information (images, video, and text) about those players involved. Thus, Which architecture seemed appropriate.

The system that was built included a central database, located in Texas, for ticket management, security, news services, and Internet links. This server also calculated games statistics and provided historical information, security photographs, and clips of video action. The clients ran on 160 Sun workstations that were located in the same cities as the games and provided support to the administrative staff and the press (Dixon 1996).

5.4 Architectural Styles and Strategies

Sidebar 5.3 The World Cup Client-Server System

- Over one month in 1994, the World Cup soccer matches were held in the United States. Design system issues:
  - 24 teams played 52 games
  - nine different cities that spanned four time zones
  - results of each game were recorded and disseminated to the press and to the fans
  - To deter violence among the fans, the organizers issued and tracked over 20,000 identification passes
- This system required both central control and distributed functions. Thus, a client-server architecture seemed appropriate.
- The system that was built included a central database, located in Texas, for ticket management, security, news services, and Internet links. This server also calculated games statistics and provided historical information, security photographs, and clips of video action.
- The clients ran on 160 Sun workstations that were located in the same cities as the games and provided support to the administrative staff and the press

Architecture Evaluation

Achieving Quality Attributes

- Architectural styles provide general beneficial properties. Quality attributes also need to be supported:
  - Modifiability
  - Performance
  - Security
  - Reliability
  - Robustness
  - Usability
  - Business goals

Characteristics that users want to see in products that we build

Achieving Quality Attributes

Modifiability

Cost spent after first version?

- Design must be easy to change
- Two classifications of affected software units:
  - Directly affected
  - Indirectly affected
- Directly affected units' responsibilities change to accommodate a system modification
  - Anticipate expected changes
  - Value cohesion
  - Maintain generality e,g servers
- Indirectly affected units' responsibilities do not change, but implementations must be revised
  - Lower coupling
  - Interact through interfaces
  - Employ multiple interfaces

Achieving Quality Attributes

Performance

- Performance attributes describe constraints on system speed and capacity:
  - Response time: How fast does our software respond to requests?
  - Throughput: How many requests can it process per minute?
  - Load: How many users can it support before response time and throughput start to suffer?

Achieving Quality Attributes

Performance

- Tactics for improving performance include:
  - Improve utilization of resources
  - Manage resource allocation more effectively
    - First-come/first-served: Requests are processed in the order in which they are received
    - Explicit priority: Requests are processed in order of their assigned priorities
    - Earliest deadline first: Requests are processed in order of their impending deadlines
  - Reduce demand for resources

Reduce computational overhead?

Increase resources? Multiple copies? Additional processors, memory? Lower sample rate?

Achieving Quality Attributes

Security

- Two key architectural characteristics particularly relevant to security: immunity and resilience
- **Immunity**: ability to thwart an attempted attack
  - The architecture encourages immunity by:
    - Ensuring all security features are included in the design
    - Minimizing exploitable security weaknesses
- **Resilience**: ability to recover quickly and easily from an attack
  - The architecture encourages resilience by:
    - Segmenting functionality to contain attack
    - Enabling the system to quickly restore functionality

Achieving Quality Attributes

Reliability

- A software system is reliable if it correctly performs its required functions under assumed conditions
  - Is the software internally free of errors?
- A **fault** is the result of human error, compared to a **failure**, which is an observable departure from required behavior
  - Software is made more reliable by preventing or tolerating faults

Achieving Quality Attributes

Reliability (continued)

- **Fault recovery**: handling fault immediately to limit damage
- Fault recovery tactics:
  - Undoing transactions: manage a series of actions as a single transaction that are easily undone if a fault occurs midway through the transaction
  - Checkpoint/rollback: software records a checkpoint of current state; rolls back to that point if system gets in trouble
  - Backup: system automatically substitutes faulty unit with backup
  - Degraded service: returns to previous state, offers degraded version of the service
  - Correct and continue: detects the problem and treats the symptoms
  - Report: system returns to its previous state and reports the problem to an exception-handling unit

Achieving Quality Attributes

Robustness

- A system is **robust** if it includes mechanisms for accommodating or recovering from problems in the environment or in other unit
- Mutual suspicion: each software unit assumes that the other units contain faults
- Robustness tactics differ from reliability tactics
- Recovery tactics are similar:
  - Rollback to checkpoint state
  - Abort a transaction
  - Initiate a backup unit
  - Provide reduced service
  - Correct symptoms and continue processing
  - Trigger an exception

Achieving Quality Attributes

Usability

- Usability reflects the ease in which a user is able to operate the system
  - User interface should reside in its own software unit

Achieving Quality Attributes

Business Goals

- Business Goals are quality attributes the system is expected to exhibit (e.g., minimizing the cost of development and time to market)
  - Buy vs. Build
    - Save development time, money
    - More reliable
    - Existing components create constraints; vulnerable to supplier
  - Initial development vs. maintenance costs
    - Save money by making system modifiable
    - Increased complexity may delay release; lose market to competitors
  - New vs. known technologies
    - Acquiring expertise costs money, delays product release
    - Either learn how to use the new technology or hire new personnel
    - Eventually, we must develop the expertise ourselves

References

- UCF slides for SE course (a few slides have been reused)
- SE book by Pressman
- SE book by Pfleeger
- SE book by Ian Sommerville