Software Architecture

Architectural Styles (Patterns)

Lecturer: Zhenyan Ji

Architectural design

- An early stage of the system design process.
- Architectural design
  - The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication.
- Software architecture is the output of architectural design process.

Definitions

- An Architectural Style defines a set of rules that describe the properties of and constraints on its components and the way in which the components interact.
  - Architectural Style is a high level design Pattern
  - Design Pattern is a programming level design pattern
- A set of architectural design solutions that have been used successfully before.
- Mark of mature engineering field.
- An architecture can use several architectural styles.
- Styles are open-ended; new styles will emerge

Benefits

- Promote reusability
  - Design reuse
    - Systems in the same domain often have similar architectures that reflect domain concepts.
    - Application product lines are built around a core architecture with variants that satisfy particular customer requirements.
  - Code reuse
    - Shared implementations of invariant aspects of style
  - Documentation reuse

Benefits

- Improves development efficiency and productivity
- Provide a starting point for additional and new design ideas.
- Promotes communications among the designers
  - Phrase such as “client-server” conveys lot of information
- quickly find a applicable SA design solution for a software system
- make trade-offs and pre-evaluate the SA of system
- diminish the risks of SA design

Content of Software Architecture Pattern

- **Name**
  - *Each architecture pattern has a unique, short descriptive name.*
- **Problem**
  - *Each architecture pattern contains a description of the problem to be solved. The problem statement may describe a class of problems or a specific problem.*

The Content of Software Architecture Pattern

- **Context**
  - *The assumptions are conditions that must be satisfied in order for the architecture pattern to be usable in solving the problem. They include constraints on the solution and optional requirements that may make the solution more easy to use.*
- **Models**
  - *The models is to describe the software architecture pattern.*

The Content of Software Architecture Pattern

- **Consequences**
  - *The advantages and disadvantages of using this pattern.*
- **Implementation**
  - *Additional implementation advice that can assist designers in customizing this architectural design pattern for the best results.*
- **Known Uses**
  - *Known applications of the pattern within existing systems*.

Architectural Styles (Patterns)

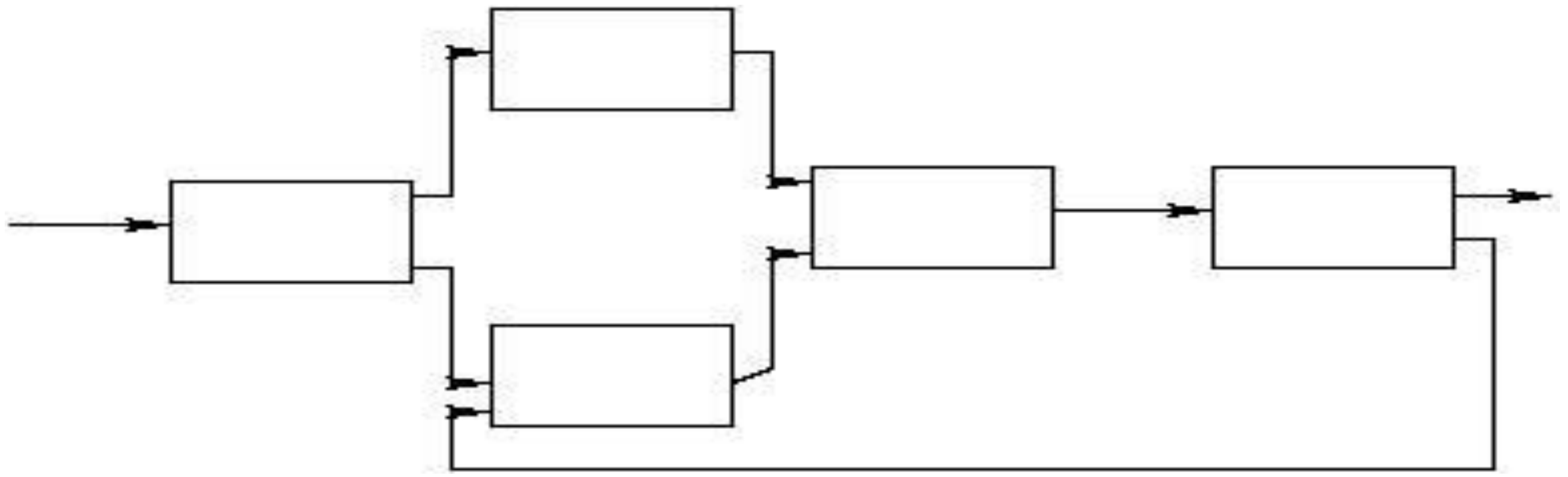
- Pipe and Filter
- The repository model (Shared Data Store)
- Client-Server Style
  - One or two tier C/S
  - Three tier C/S
- Model-View-Controller (MVC)
- Layered Architecture
- Peer-to-Peer Style
- Event Driven Style

Architectural Styles:

Pipes and Filters

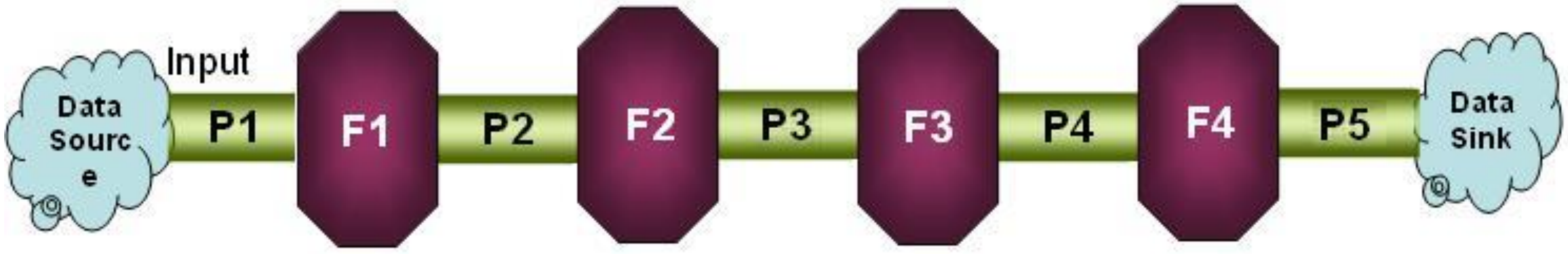
Pipes and Filters

- **A pipeline consists of a** chain **of processing elements, and the** output **of each element is the** input **of the next. Usually some amount of buffering is provided between consecutive elements.**



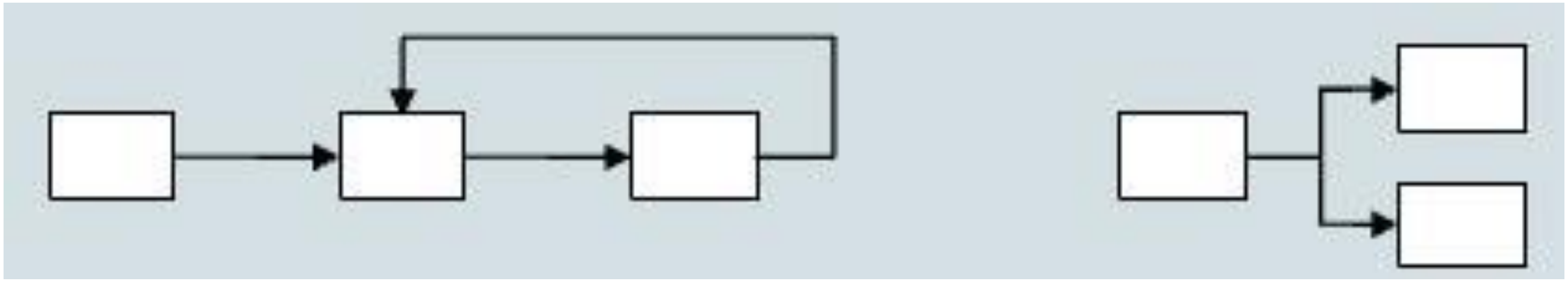
Pipes and Filters

- **Pattern Name: Pipe/Filter**
  - **Components:** filters -- Data Handling
    Read a stream of data on its inputs and produce a stream of data on its outputs.
  - **Connectors:** pipes -- Data Translation and Transportation



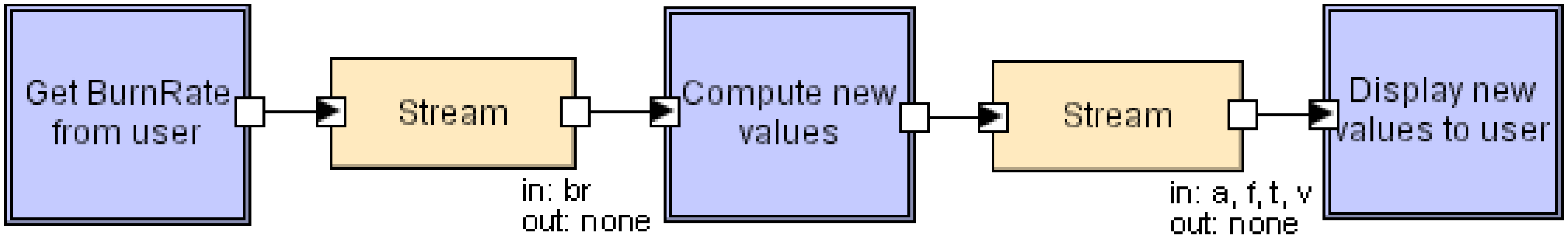
Pipes and Filters

- Topology: linear;
  - variations:feedback-loops, splitting pipes

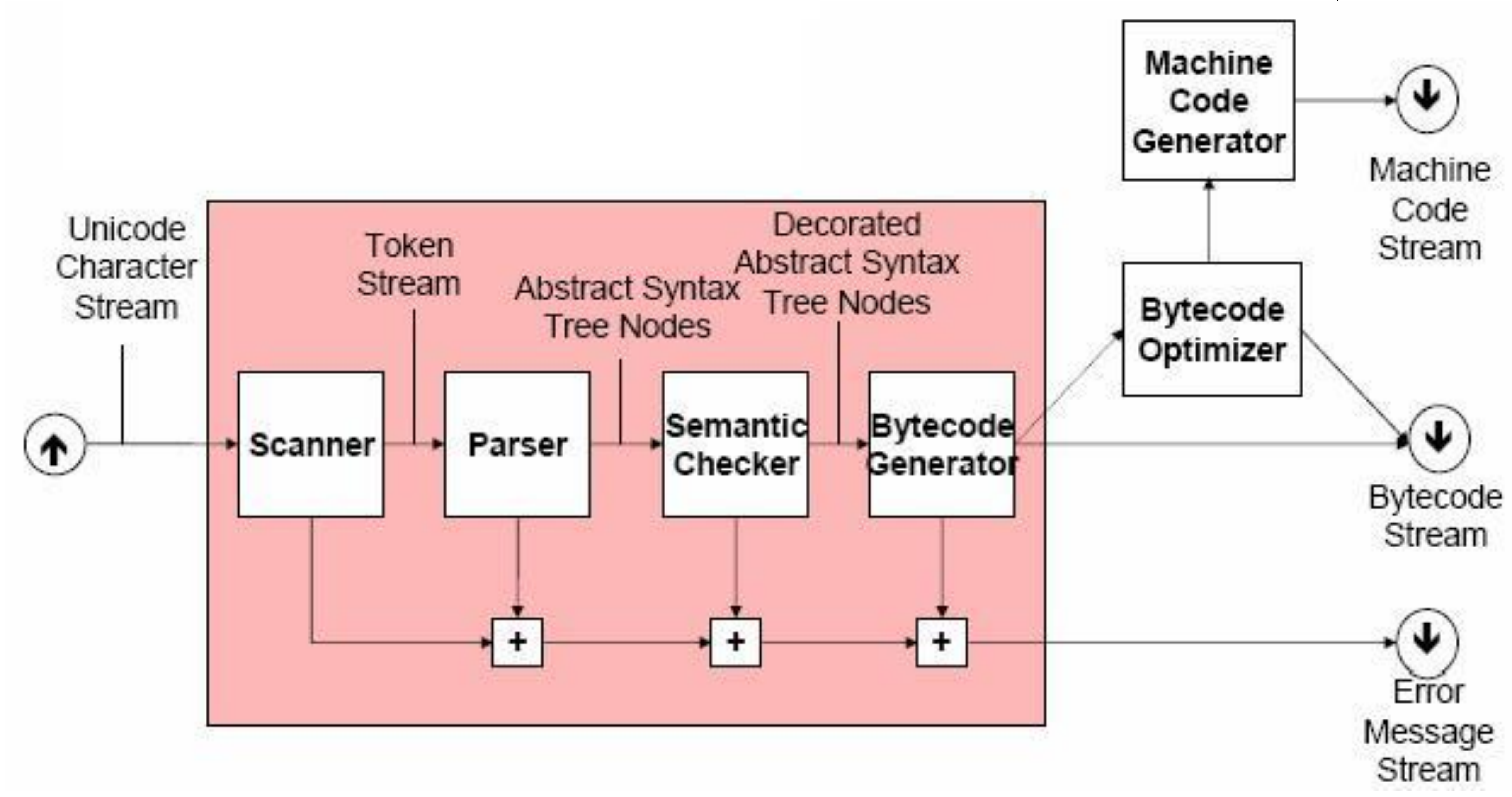


- Semantic Constraints
  - Filters are independent entities
    - they do not share state
    - they do not know their predecessor/successor

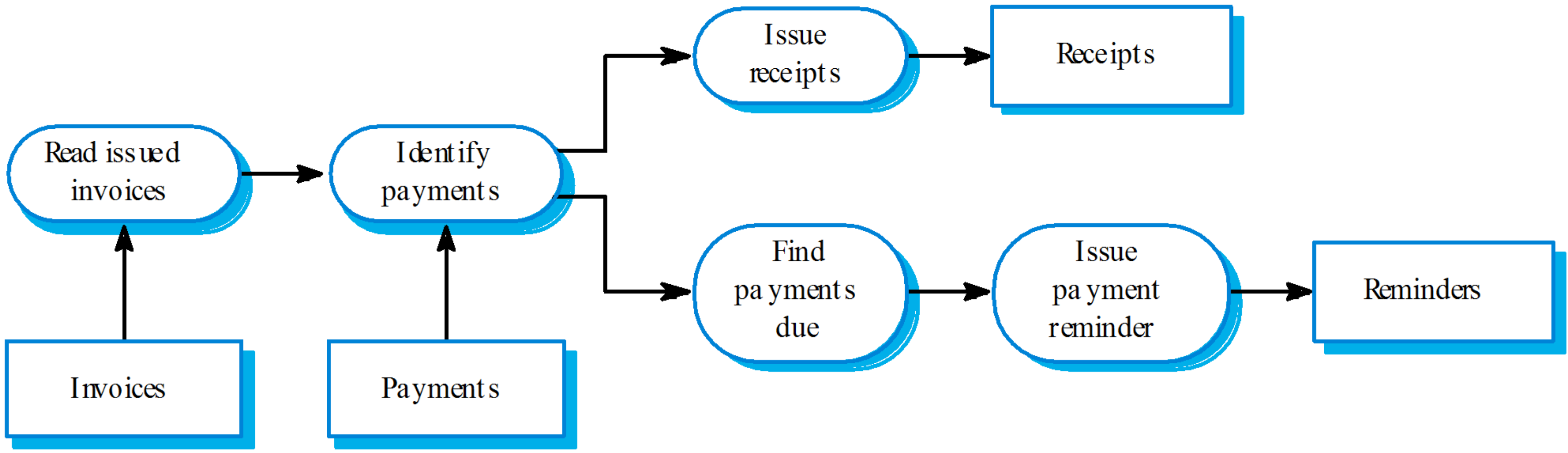
Pipe and Filter



Pipes and Filters: Compiler



Invoice processing system



Advantages and Disadvantages of Pipe-Filter

- **Advantages:**
  - High cohesive: Filters are self containing processing service that performs a specific function thus it is fairly cohesive
  - Low coupling: Filters communicate through pipes only, thus it is "somewhat" constrained in coupling
  - Reusability: Supports to reuse filters.
  - Simple to implement as either a concurrent or sequential system.

Advantages and Disadvantages of Pipe-Filter

- Extendibility: easy to add new filters.
- Flexibility:
  - functionality of filters can be easily redefined
  - control can be re-routed
- **Disadvantages:**
  - requires a common format for data transfer along the pipeline.
  - difficult to support event-based interaction

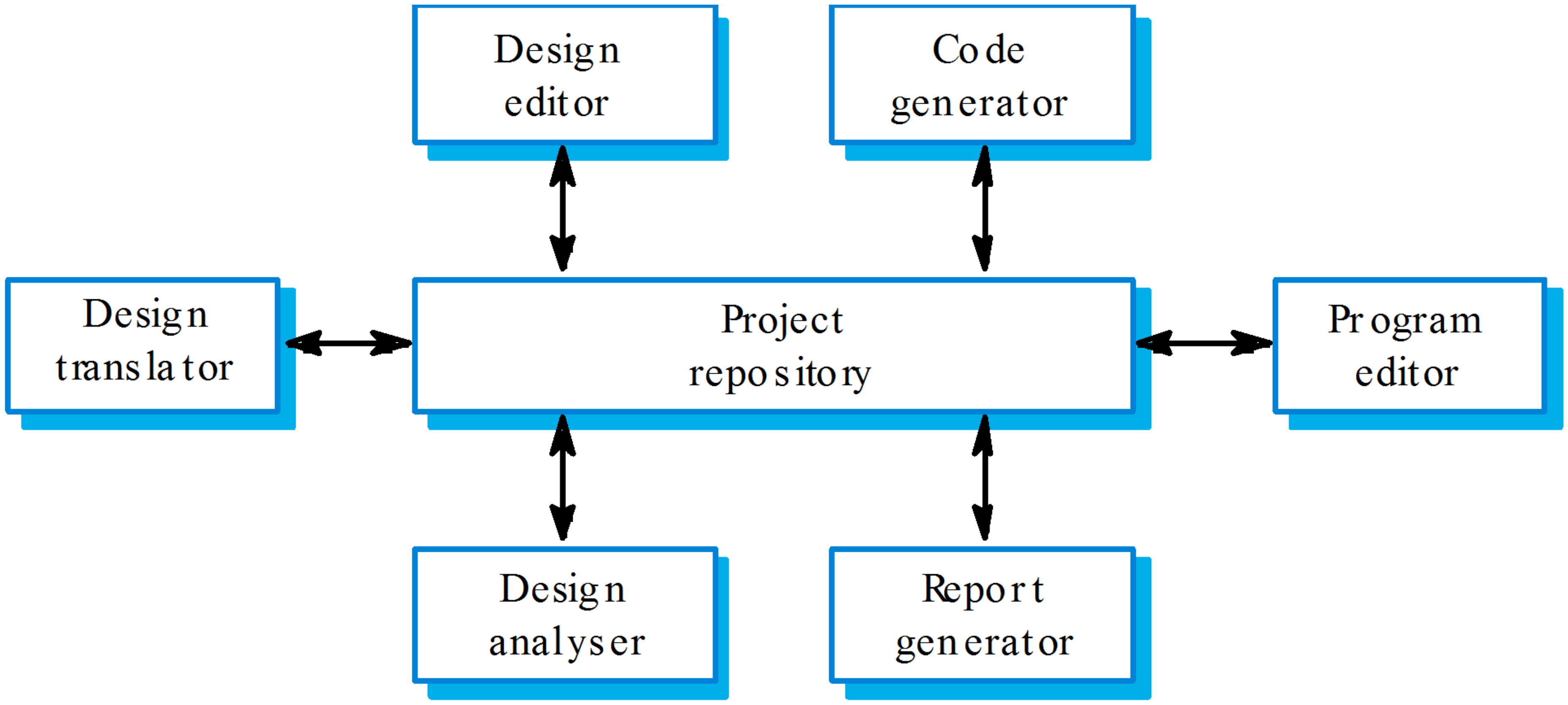
Architectural Styles:

The Repository Model

The Repository Model

- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems;
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.
- When large amounts of data are to be shared, the repository model of sharing is most commonly used.

CASE toolset architecture



The Repository Model

- **Two variations:**
  - Blackboard style: the data-store alerts the participating parties whenever there is a data-store change (trigger)
  - Repository style: the participating parties check the data-store for changes

The Repository Model

- Problem domains that fit this style such as patient processing, tax processing system, inventory control system; etc. have the following properties:
  - All the functionalities work off a single data-store.
  - Any change to the data-store may affect all or some of the functions
  - All the functionalities need the information from the data-store

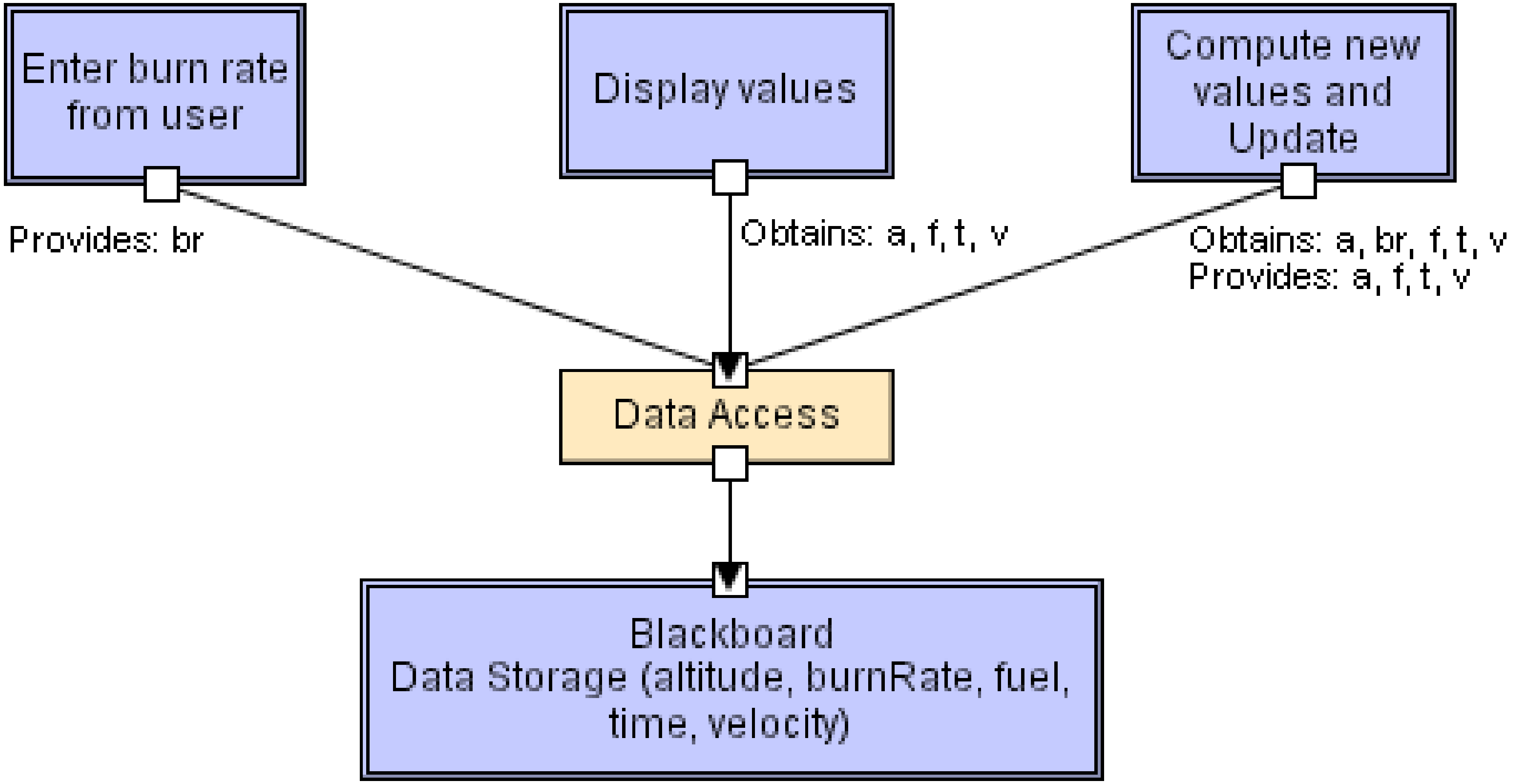
Blackboard Style

- Two kinds of components
  - Central data structure — blackboard
  - Components operating on the blackboard
- System control is entirely driven by the blackboard state
- Applicability:
  - Typically used for AI systems

Blackboard Style: DB triggers

- **a database management trigger has 3 parts:**
  - Event : change to the database that alerts or activates the trigger
  - Condition: a test that is true when the trigger is activated
  - Action: a procedure which is executed when the trigger is activated and the condition is true.

Blackboard


Enter burn rate from user
Display values
Compute new values and Update
Provides: br
Obtains: a, f, t, v
Obtains: a, br, f, t, v
Provides: a, f, t, v
Data Access
Blackboard
Data Storage (altitude, burnRate, fuel, time, velocity)

Repository model

- Advantages:
  - The independent functions are cohesive within itself and the coupling is restricted to the shared data
  - Single data-store makes the maintenance of data in terms of back-up recovery and security easier to manage
  - Efficient way to share large amounts of data.

Repository model

- Disadvantages:
  - Difficult to manage data
    - Any data format change in the shared data requires agreement and, potentially, changes in all or some the functional areas - - this becomes a bigger problem as more functionalities are introduced that have dependency on the shared data.

Repository model

- Data evolution is difficult and expensive;
- If the data-store fails, all parties are affected and possibly all functions have to stop
  - redundant db
  - good back up- and recovery procedures

Architectural Styles:

Client-server Style

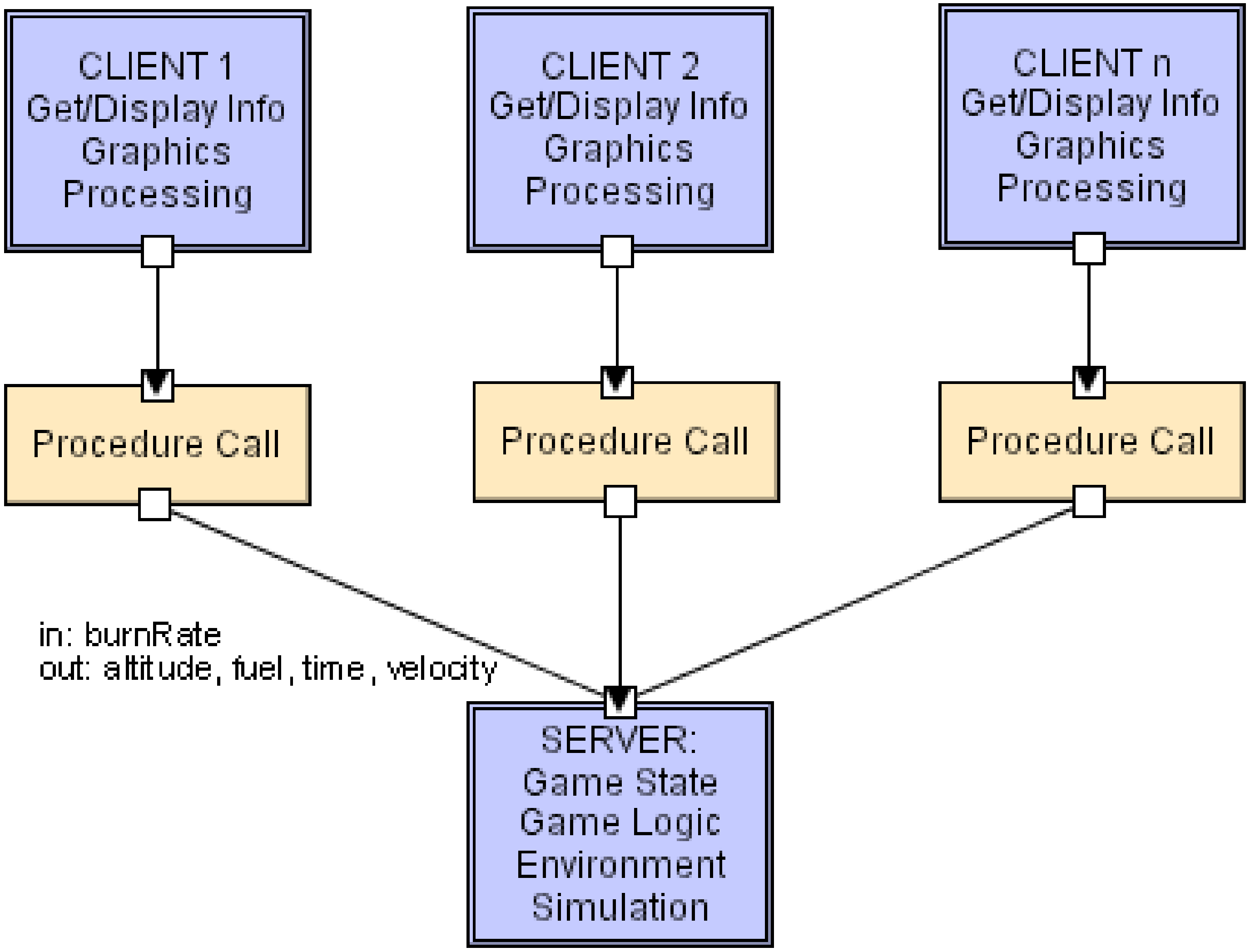
Client-server Style

- Components are clients and servers
  - Client: an application that makes requests (to the servers) and handles input/output with the system environment
  - Server: an application that responses requests from clients.
  - Servers do not know number or identities of clients
  - Clients know server's identity

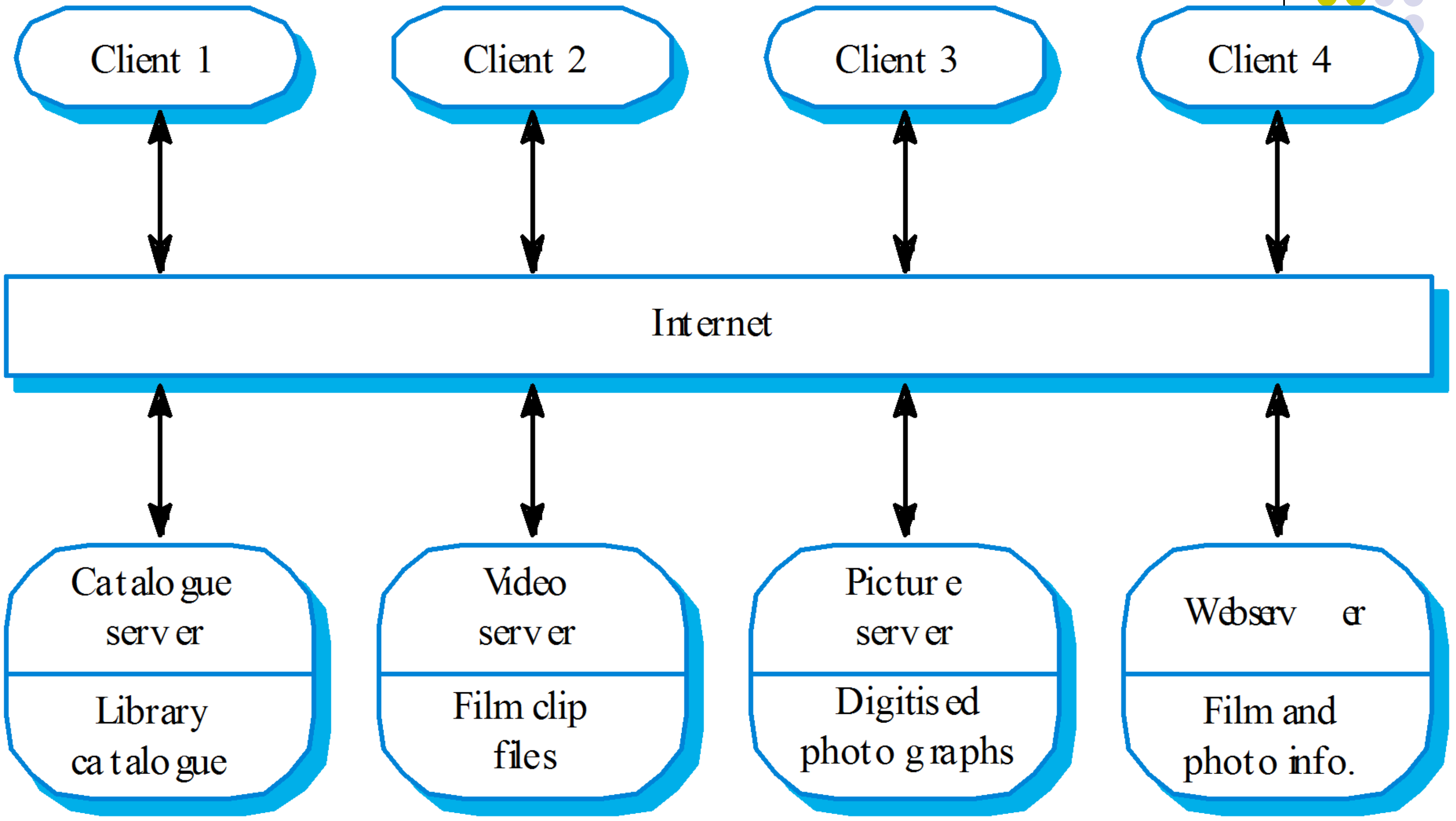
Client-server Style

- Connectors are RPC-based network interaction protocols
- Why Client / Server?
  - multiple users want to share and exchange data
- Typical application area:
  - distributed multi-user (business) information systems

Client-Server Style



Film and picture library



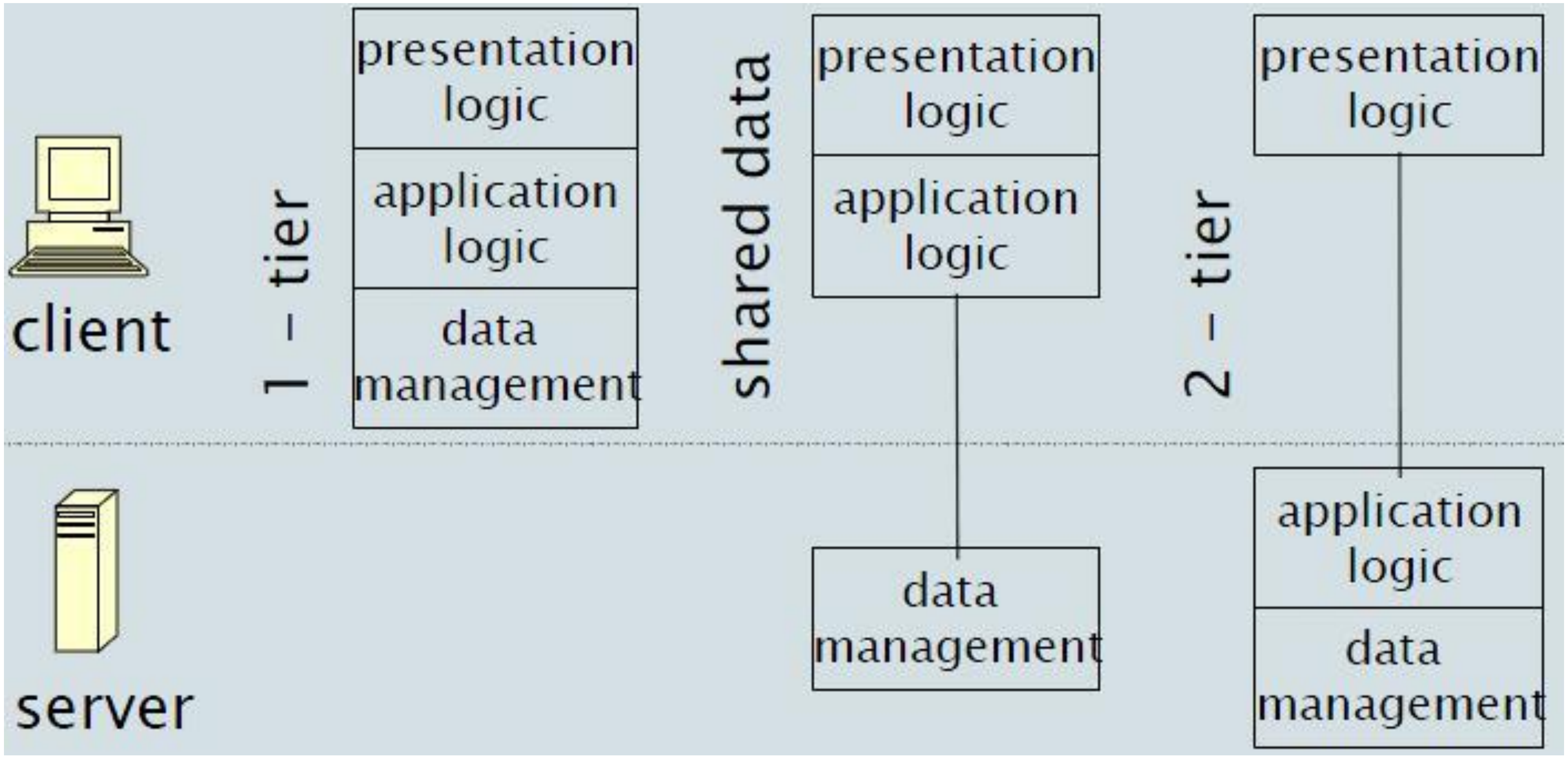
Client/Server Characteristics

- Dependencies: client depends on the server
- Topology:
  - one or more clients may be connected to a server.
  - there are no connections between clients
- Synchronicity: synchronous or asynchronous
- Mobility: easily supports client mobiliy
- Security: typically controlled at server, also possible at application/business layer

One or Two Tier Client/Server Architectures

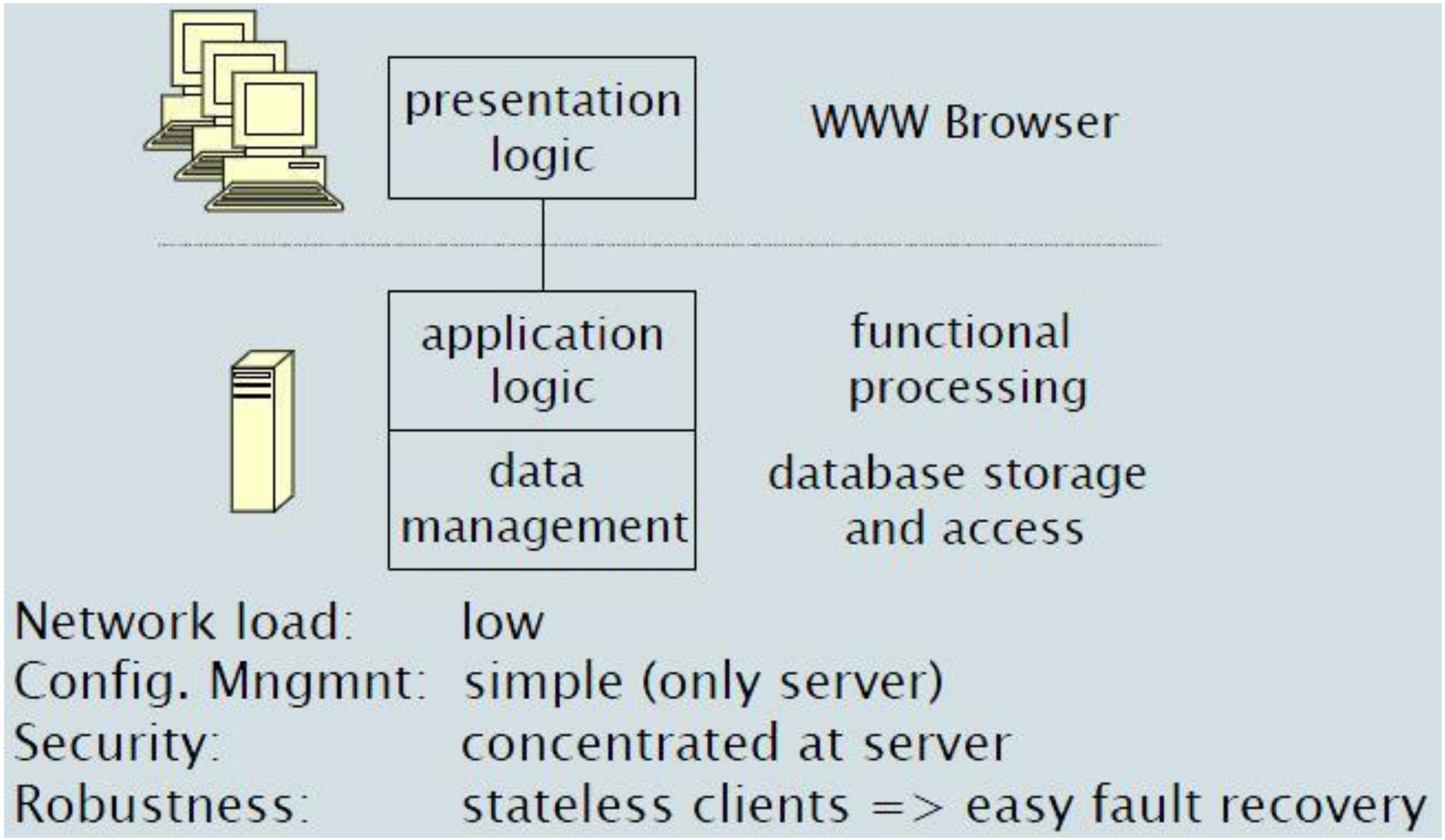
- Processing management split between user system interface environment and database management server environment.
  - Tier 1: user system interface: In user's desktop environment
  - Tier 2: database management services: In a server
- Limitations
  - Performance deteriorate when number of clients is large.

One or Two Tier Client/Server Architectures



C/S Example: Thin Client

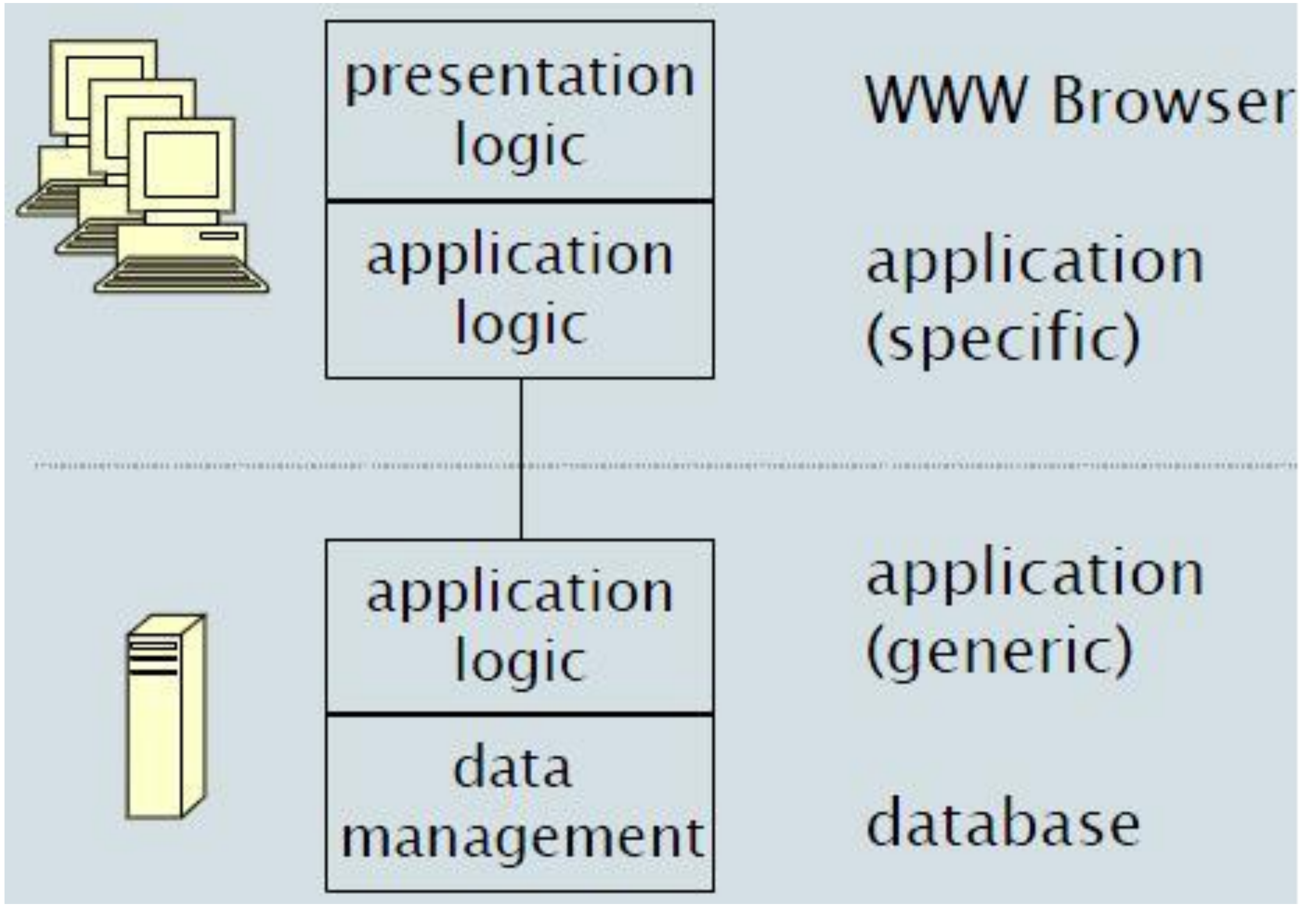
- largest part of processing at the server-side



Network load:	low
Config. Mngmnt:	simple (only server)
Security:	concentrated at server
Robustness:	stateless clients => easy fault recovery

C/S Example: Thick Client

- Thick Client: significant processing at the client-side



C/S Example: Thick Client

- Network load: high
- Config. Mngmnt: complex (both client & server)
- Security: complex (both client & server)
- Robustness: clients have state => complex fault recovery

Client-server characteristics

- Advantages
  - Makes effective use of networked systems. May require cheaper hardware;
  - Easy to add new servers or upgrade existing servers.
  - Allows sharing of data between multiple users
  - Scalable: add new client

Client-server characteristics

- Disadvantages
  - Redundant management in each server;
  - Hard to find out what servers and services are available. No central register of names and services
  - Difficult to change functionalities of server and client
    - Changing application logic is difficult if it is distributed over C&S
  - Scalability of applications is limited by server & network capacity

Three Tier Client/Server Architecture

- **3 Tier architecture:**
  - **Presentation Layer**
    Presentation Layer is the layer responsible for displaying user interface.
  - **Business Tier**
    Business Tier is the layer responsible for accessing the data tier to retrieve, modify and delete data to and from the data tier and send the results to the presentation tier. This layer is also responsible for processing the data retrieved and sent to the presentation layer.

Three Tier Client/Server Architecture

BLL and DAL
Often this layer is divided into two sub layers: the Business Logic Layer (BLL), and the Data Access Layers (DAL). Business Logic Layers are above Data Access Layers, meaning BLL uses DAL classes and objects. DAL is responsible for accessing data and forwarding it to BLL.

- **Data Tier**
  Data tier is the database or the source of the data itself.

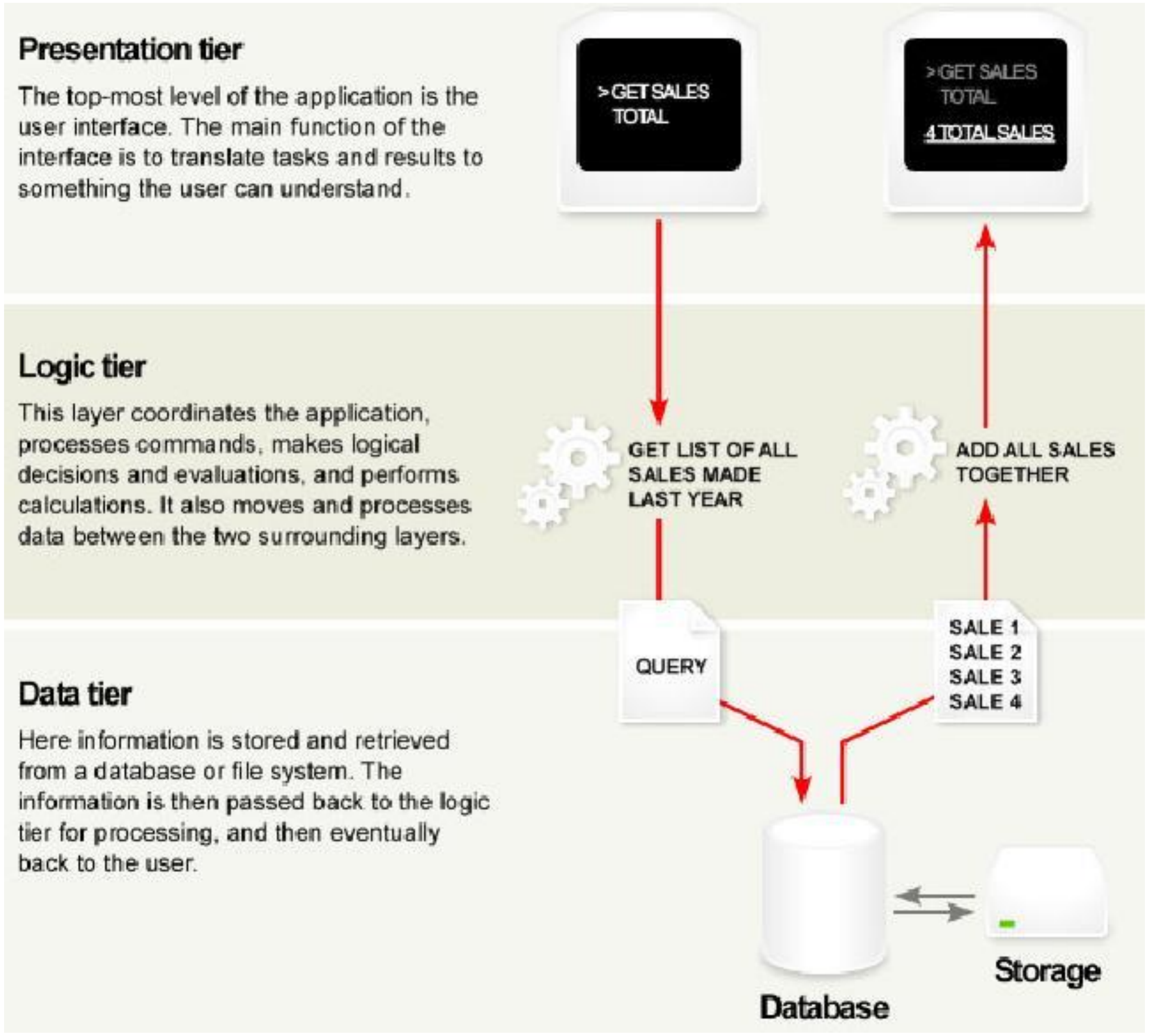
Three Tier Client/Server Architecture

- presentation logic ([G]UI):
  - anything that involves system/user interaction e.g. dialogs (management), forms, reports
- application logic (data processing):
  - where the functionality of the application resides / where the actual computation of the system takes place
- data management:
  - storing, retrieving and updating data

Three Tier Client/Server Architecture

- Common mistakes
  - tightly coupling layers in technology
  - writing business logic in presentation tier

3-tier System



Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

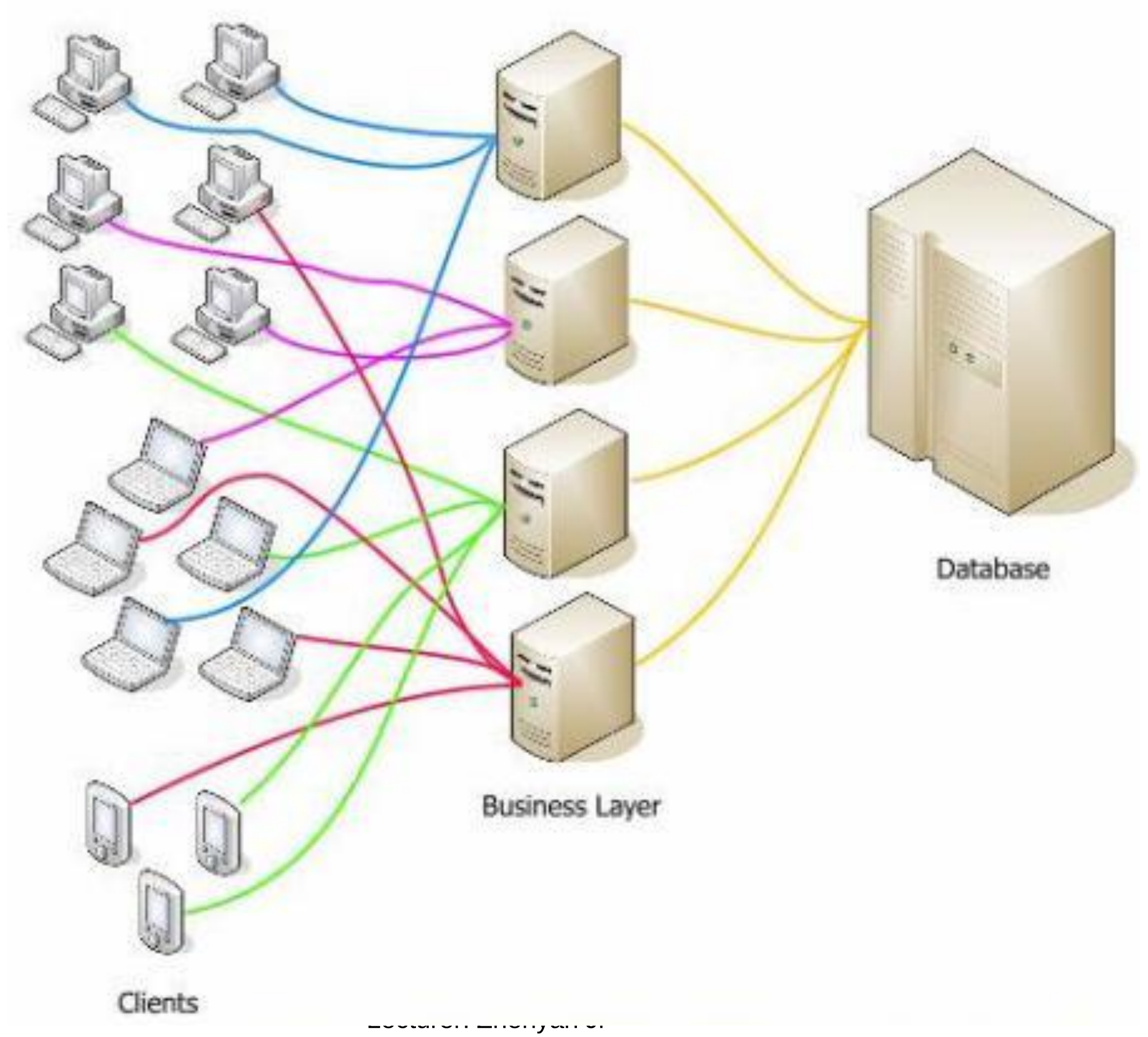
Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

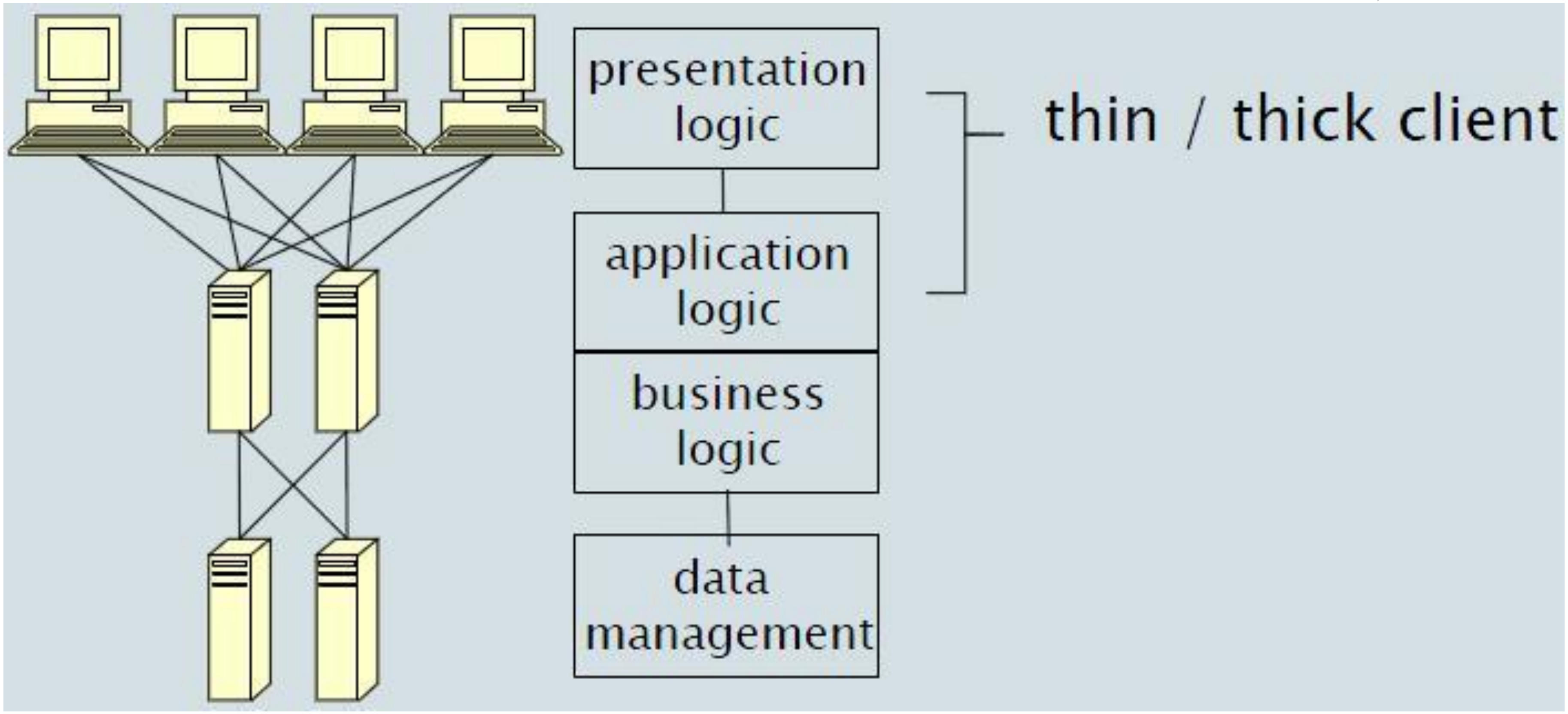
Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

Deployment: Many physical clients and servers



3-tier C/S Implementation Scenarios



Three Tier Client/Server Architecture

- Advantages comparing with two tier CS style:
  - Better performance.
  - Better scalability, reusability and maintainability.
  - Security measures can be centrally allocated.
  - Parallel development of different layers

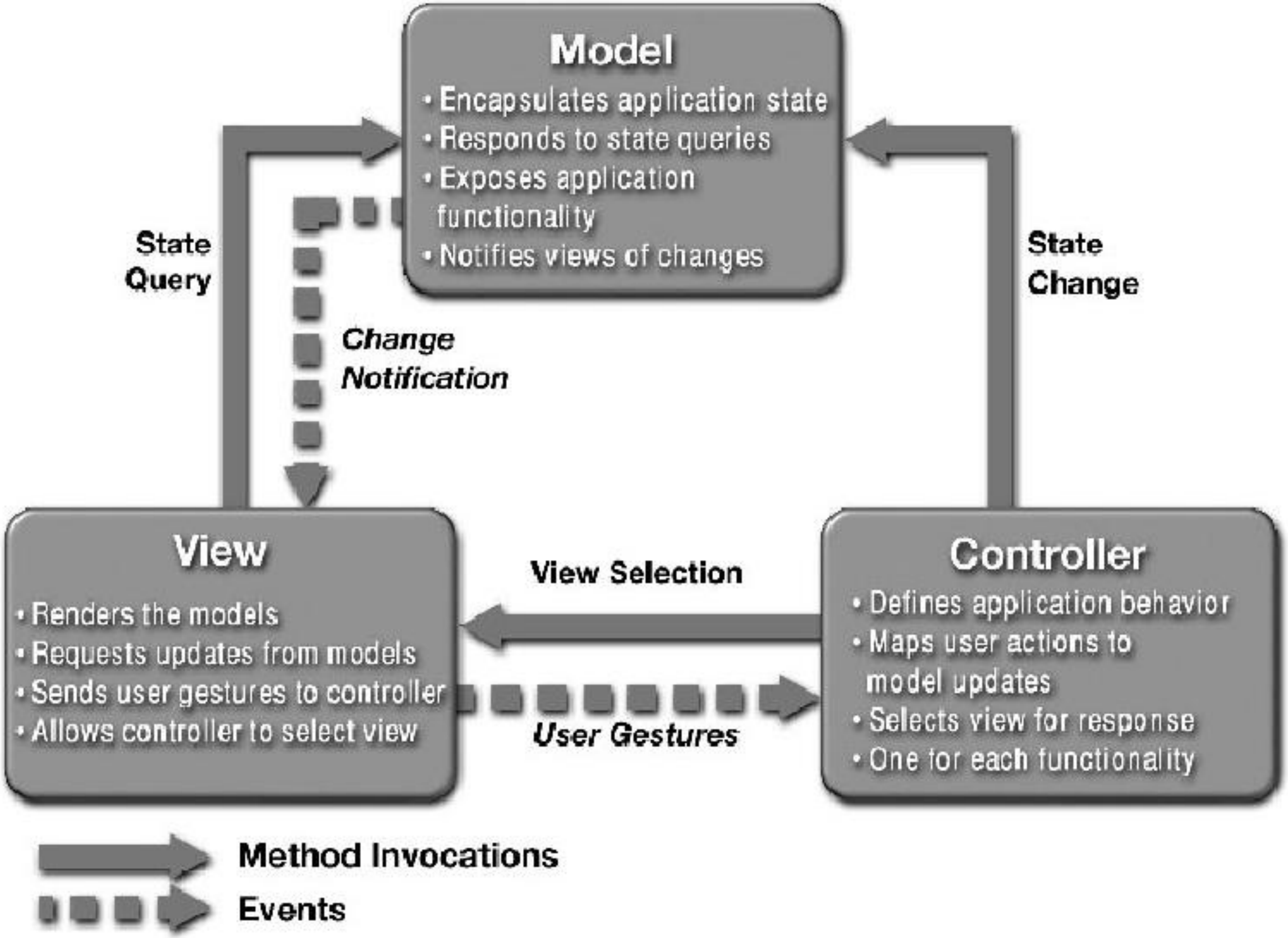
Architectural Styles:

Model View Controller

Model View Controller

- The Model-View-Controller (MVC) pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes
- The controller changes the model
- The View Listens to Model Changed events and update itself

MVC Pattern


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Advantages and Disadvantages

- **Advantages:**
  - **Views, controller, and model are separate components that allow modification and change in each “layer” without significantly disturbing the other**
    - **The view component, which often needs changes (UI technology improvement) and updates to keep the users continued interests, is separate**

Advantages and Disadvantages

- The View-Controller can keep on partially functioning even if the model component is down.

- **Disadvantages:**
  - Heavily dependent on the development and production system environment and tools that match the MVC architecture (e.g. TomCat, .Net, Rail, etc.)

Architectural Styles:

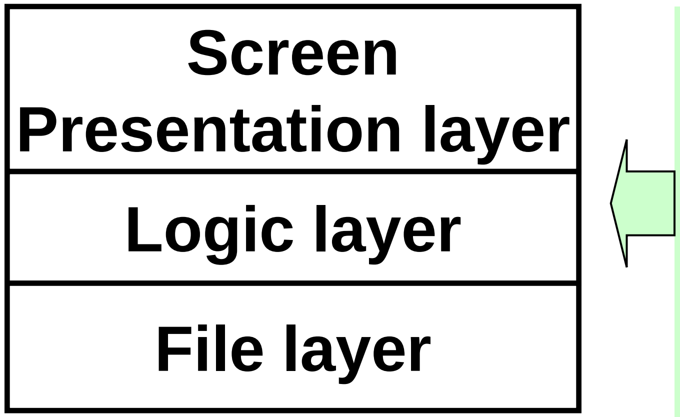
Layered System

Layered System

- **Pattern Name: Layered System**
  - Component: Layer
  - Connector: Interaction Protocol between layers

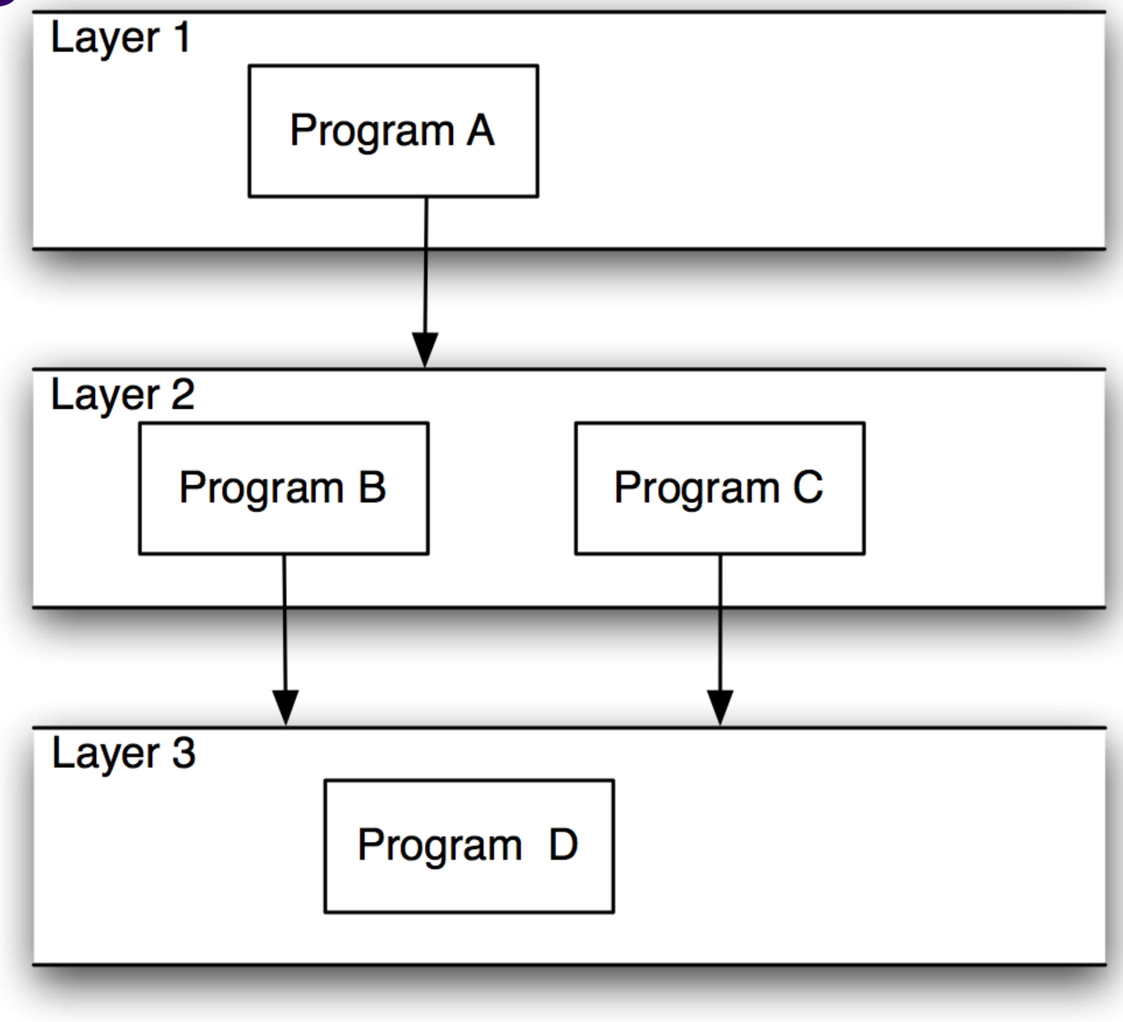
Layered architecture

- **The high level design solution is decomposed into Layers:**
  - **Structurally, each layer provides a related set of services**
  - **Dynamically, each layer may only use the layers below it**



1. If any layer only uses the layer directly below it, then it is a Strict Layered Style.
2. If a layer may use any of the layers below it, then it is a Relaxed Layer Style

Layered Systems/Virtual Machines

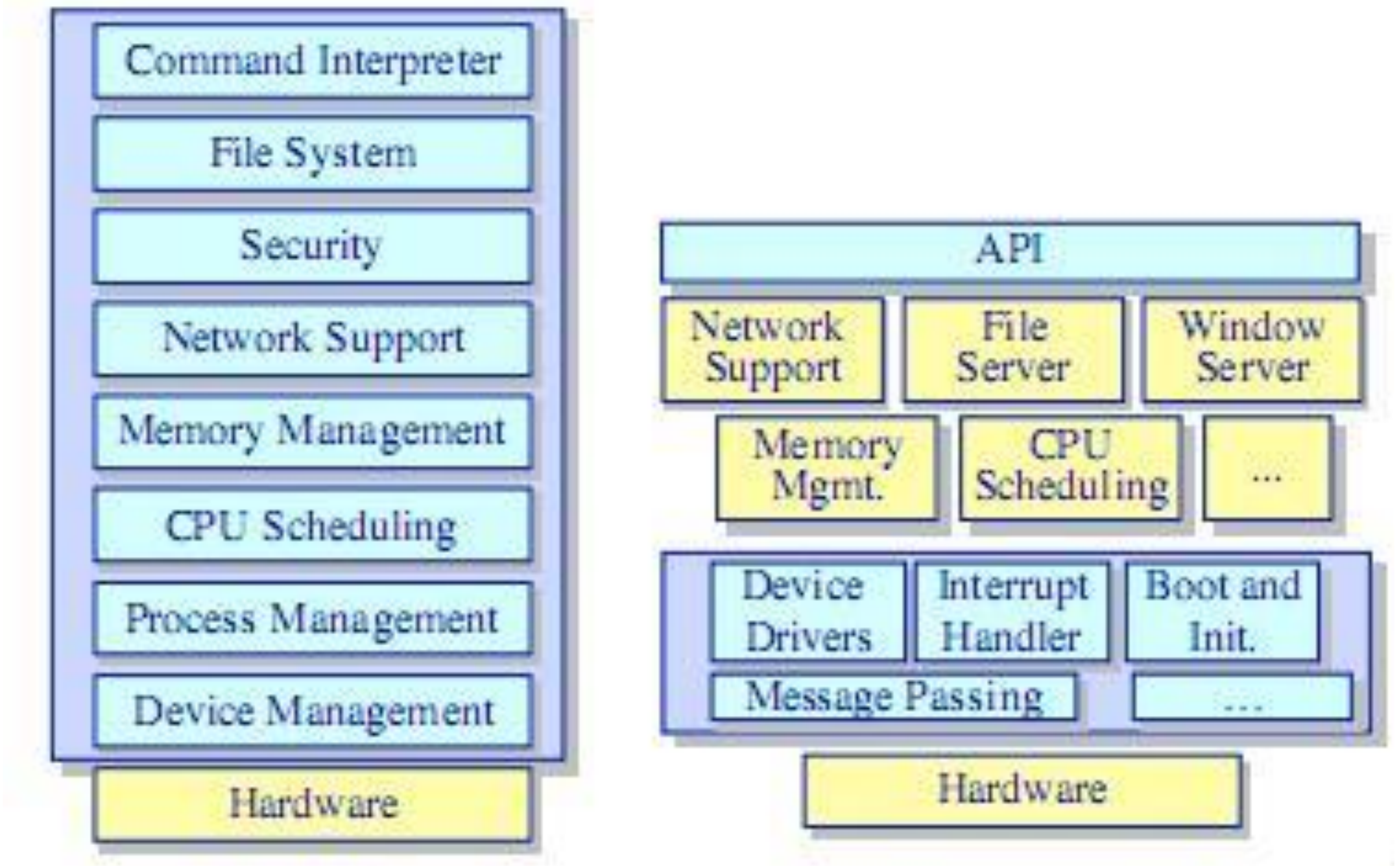


Layered Style

- Hierarchical system organization
  - “Multi-level client-server”
  - Each layer exposes an interface (API) to be used by above layers
- Each layer acts as a
  - *Server:* service provider to layers “above”
  - *Client:* service consumer of layer(s) “below”
- Connectors are protocols of layer interaction
- Example: operating systems

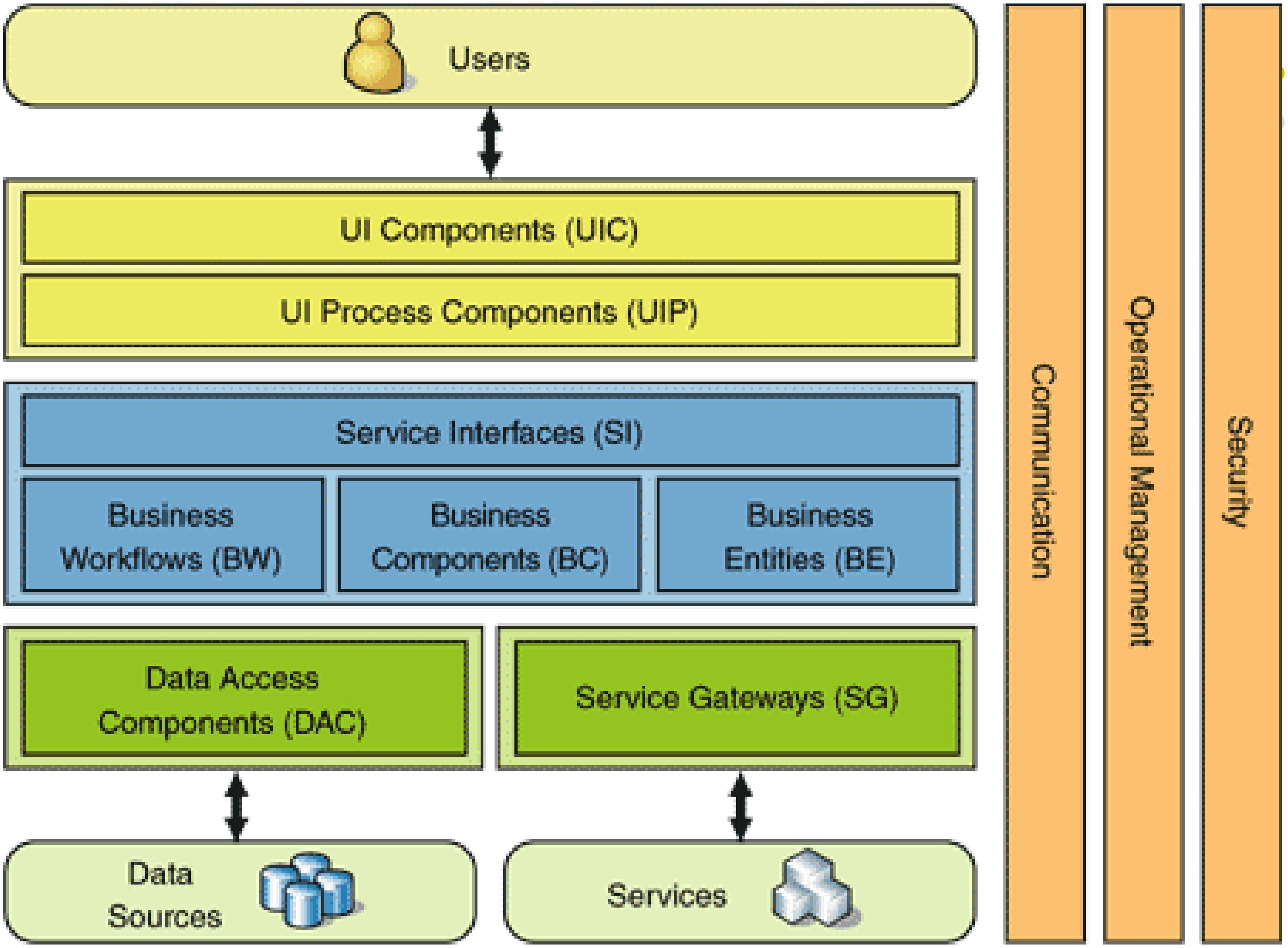
Example 1

- Monolithic OS& Micro-kernel OS



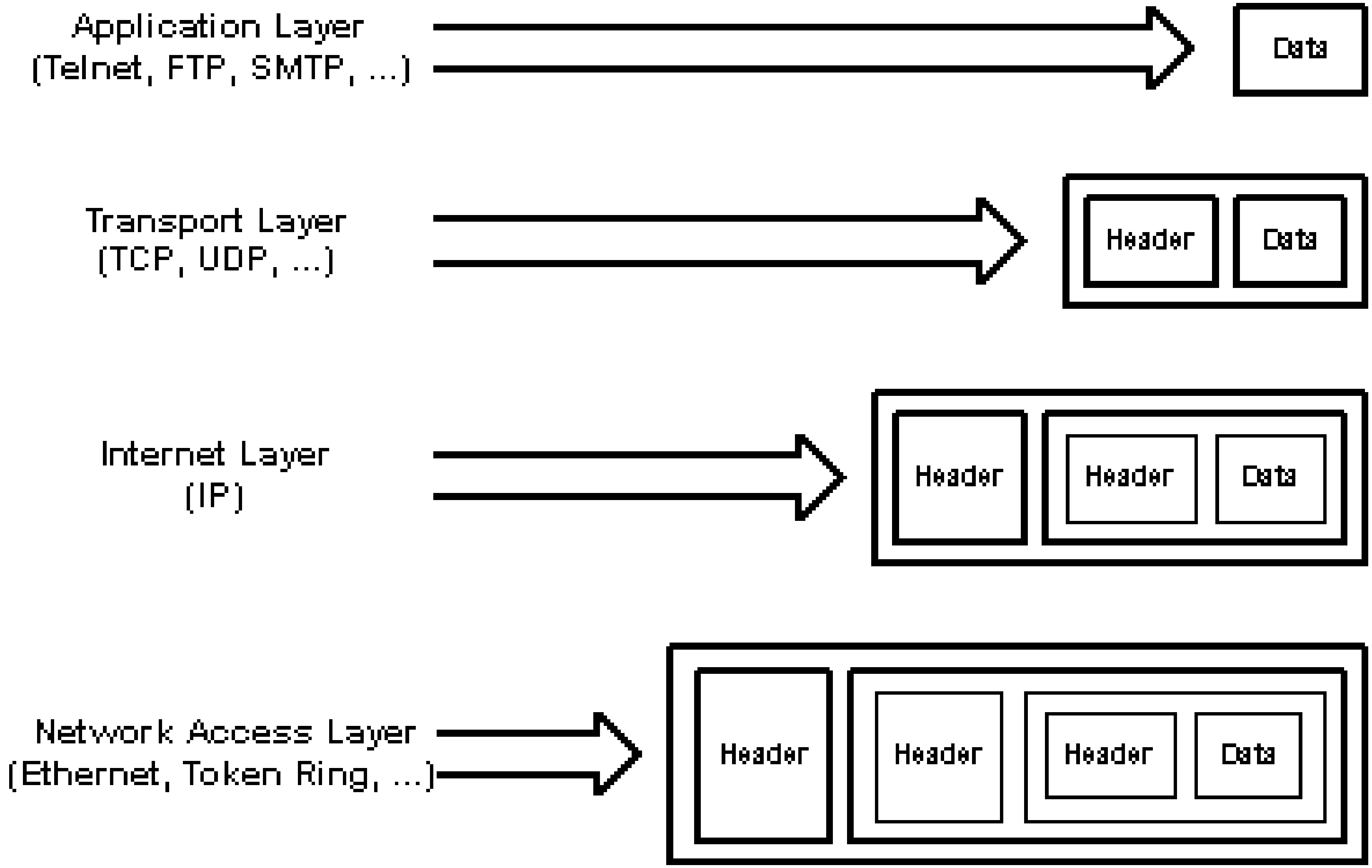
Example 2

- It is quite common for **enterprise application architects** to compose their solutions into the following three layers:
  - **Presentation**. This layer is responsible for interacting with the user.
  - **Business**. This layer implements the business logic of the solution.
  - **Data**. This layer encapsulates the code that accesses the persistent data stores such as a relational database.

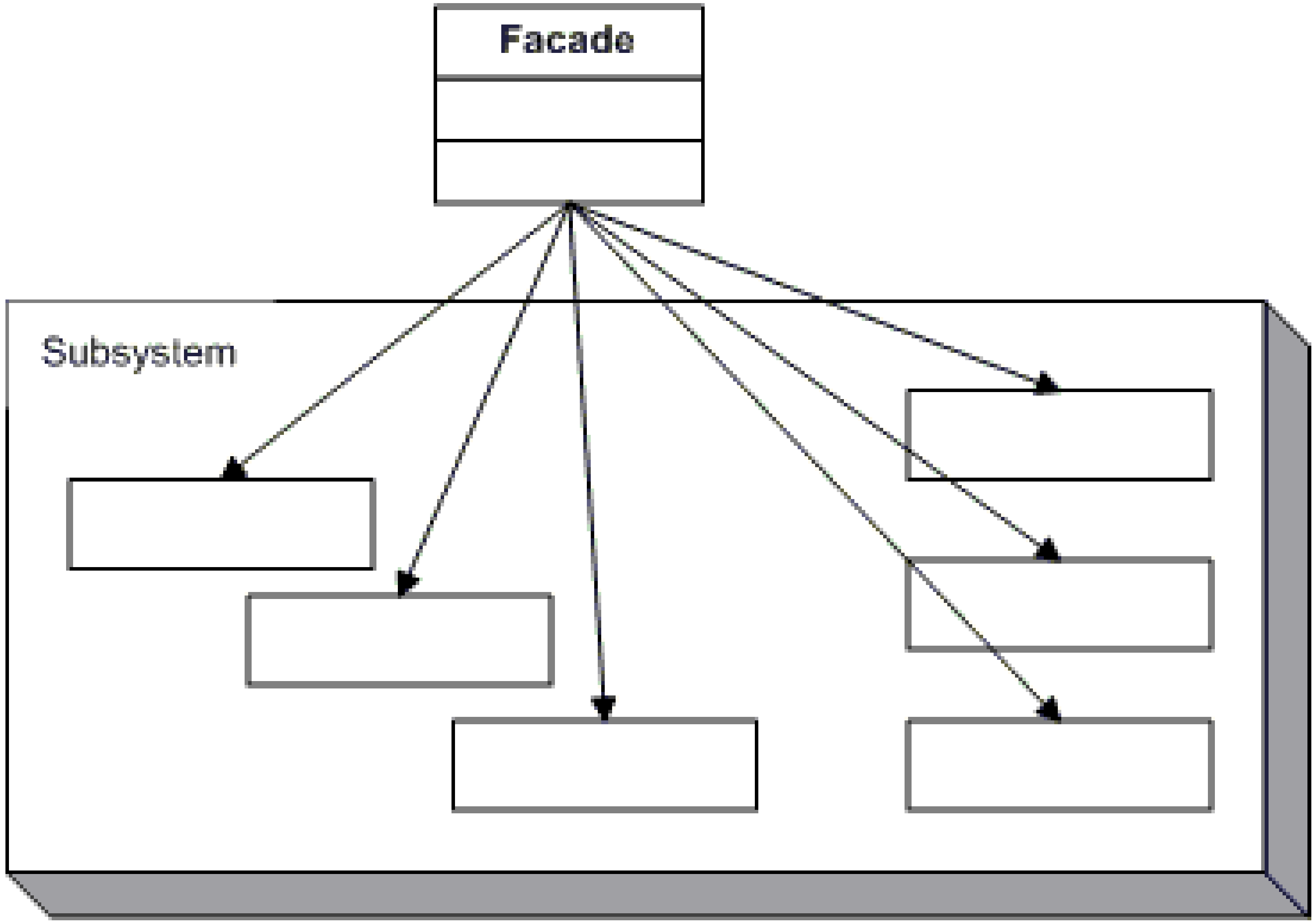

Users
UI Components (UIC)
UI Process Components (UIP)
Service Interfaces (SI)
Business Workflows (BW)
Business Components (BC)
Business Entities (BE)
Data Access Components (DAC)
Service Gateways (SG)
Data Sources
Services
Communication
Operational Management
Security

Example 3

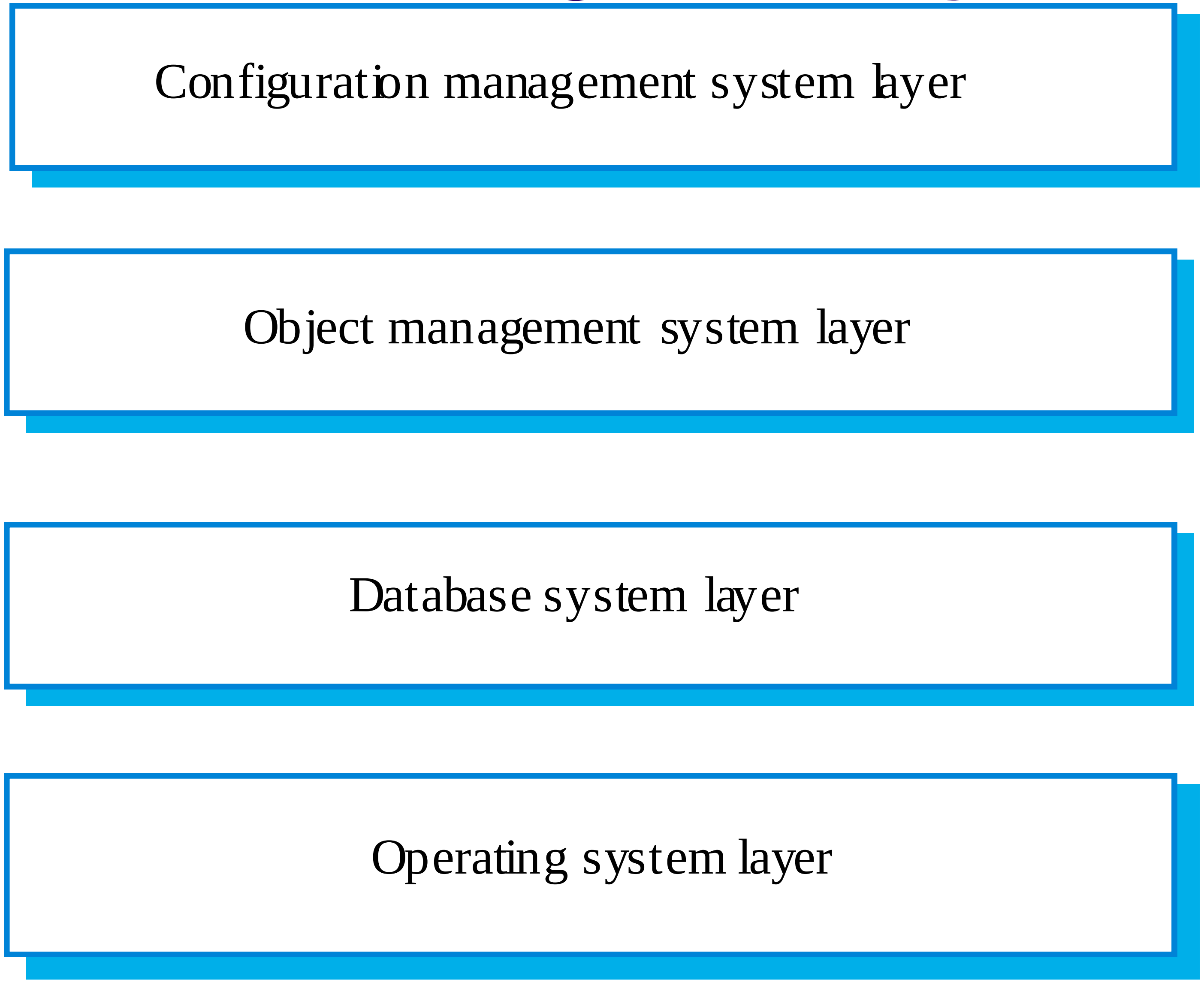
- The Tcp/IP protocol also is a layered system.



Layered System



Version management system


Configuration management system layer
Object management system layer
Database system layer
Operating system layer

Advantages and Disadvantages

- Advantages:
  - Each layer is selected to be a set of related services; thus the architecture provides high degree of cohesion within the layer.
  - Each layer may hide private information from other layers
  - Layers may use only lower layers, constraining the amount of coupling.
    - Easy to add and/or modify a current layer

Advantages and Disadvantages

  - Changes in a layer affect at most the adjacent two layers
- Each layer, being cohesive and is coupled only to lower layers, makes it easier for reuse by others and easier to be replaced or interchanged
  - change of DB touches only the data store/access layer, change of browser only changes the presentation layer.

Advantages and Disadvantages

- Different implementations of layer are allowed as long as interface is preserved
- Supports the incremental development of sub-systems in different layers.
- Supports design by abstraction levels.

- **Disadvantages:**
  - **Strict Layered Style may cause performance problem depending on the number of layers**
  - **Not always easy to structure in *clean* layers.**

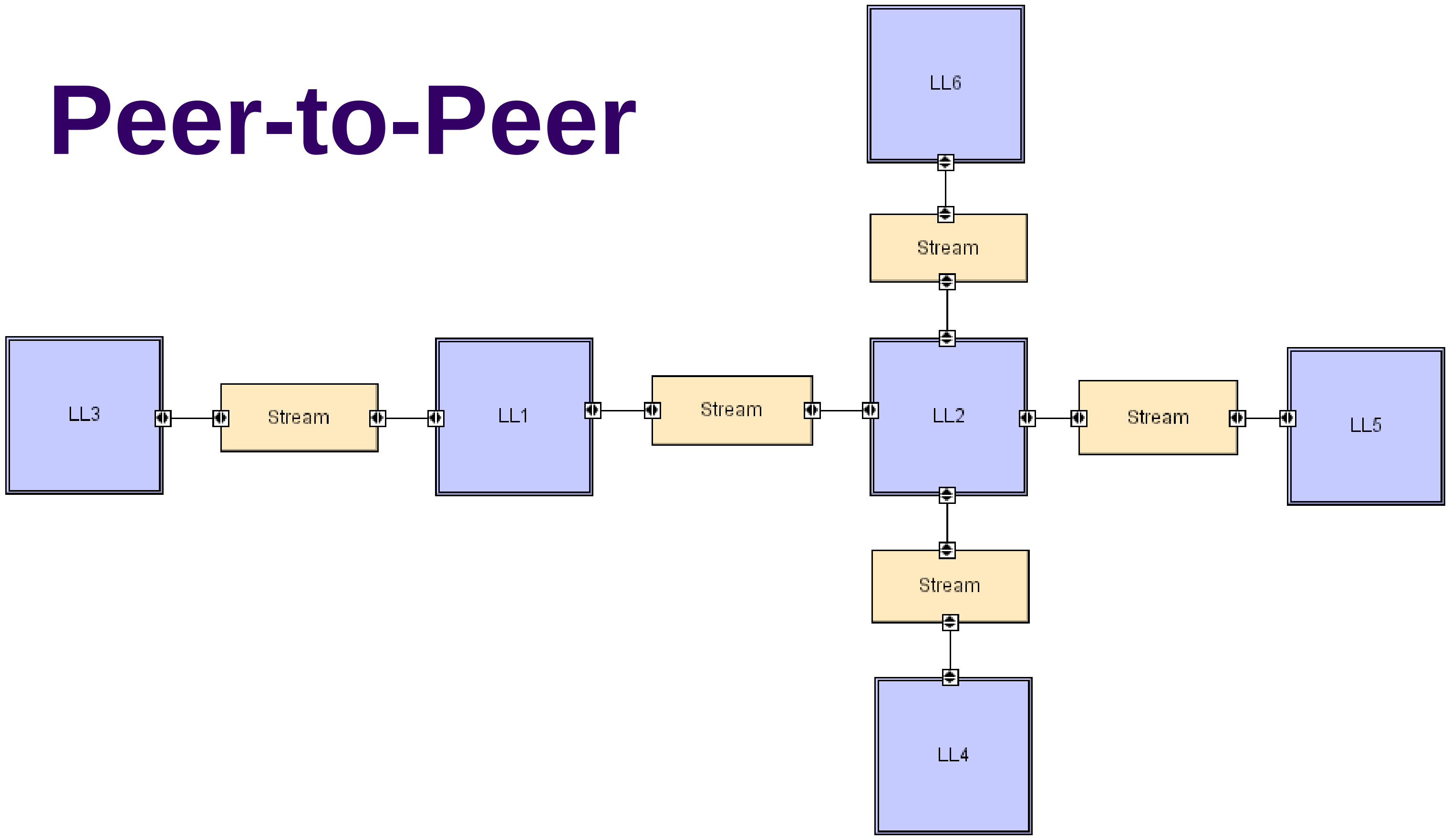
Architectural Styles:

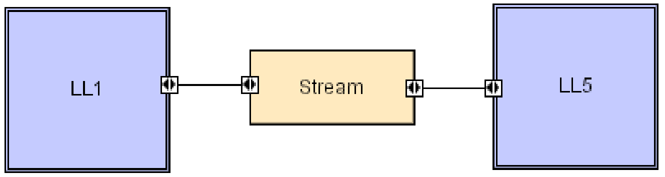
Peer-to-Peer Style

Peer-to-Peer Style

- No distinction between processes (nodes). Each process (node) can act as both a server and a client.
- Peers: Each maintain its own data-store, as well as a dynamic routing table of addresses of other nodes
- Connectors: Network protocols, often custom.

Peer-to-Peer


LL6
Stream
LL3
Stream
LL1
Stream
LL2
Stream
LL5
Stream
LL4


LL1
Stream
LL5

Architectural Styles:

Event-Driven Style

Event-Driven Style

- The high level design solution is based on an *event dispatcher* which manages events and the functionalities which depends on those events. These have the following characteristics:
  - Events may be a simple notification or may include associated data
  - Events may be prioritized based on constraints *such as time*
  - Events may require synchronous or asynchronous processing
  - Events may be “registered” or “unregistered” by components

Event-Driven Style

- Problems that fit this architecture includes real-time systems such as: airplane control; medical equipment monitor; home monitor; embedded device controller; game; etc.

Event-Driven Style

- Two principal event-driven models
    - Broadcast models.

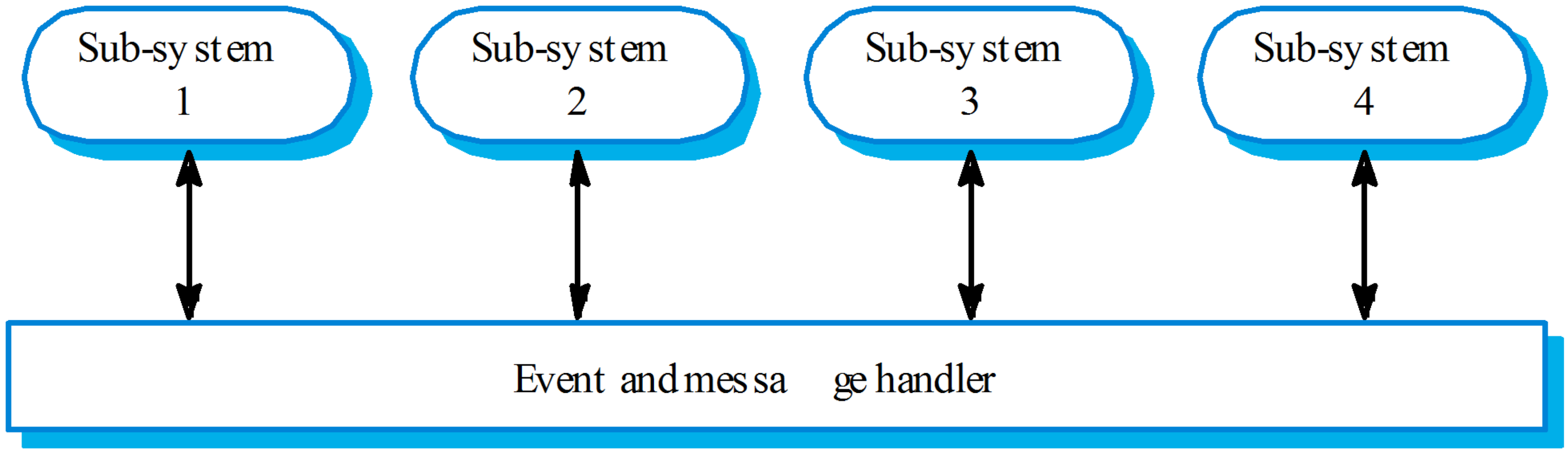
      An event is broadcast to all sub-systems. Any sub-system which can handle the event may do so;
    - Interrupt-driven models.

      Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing.

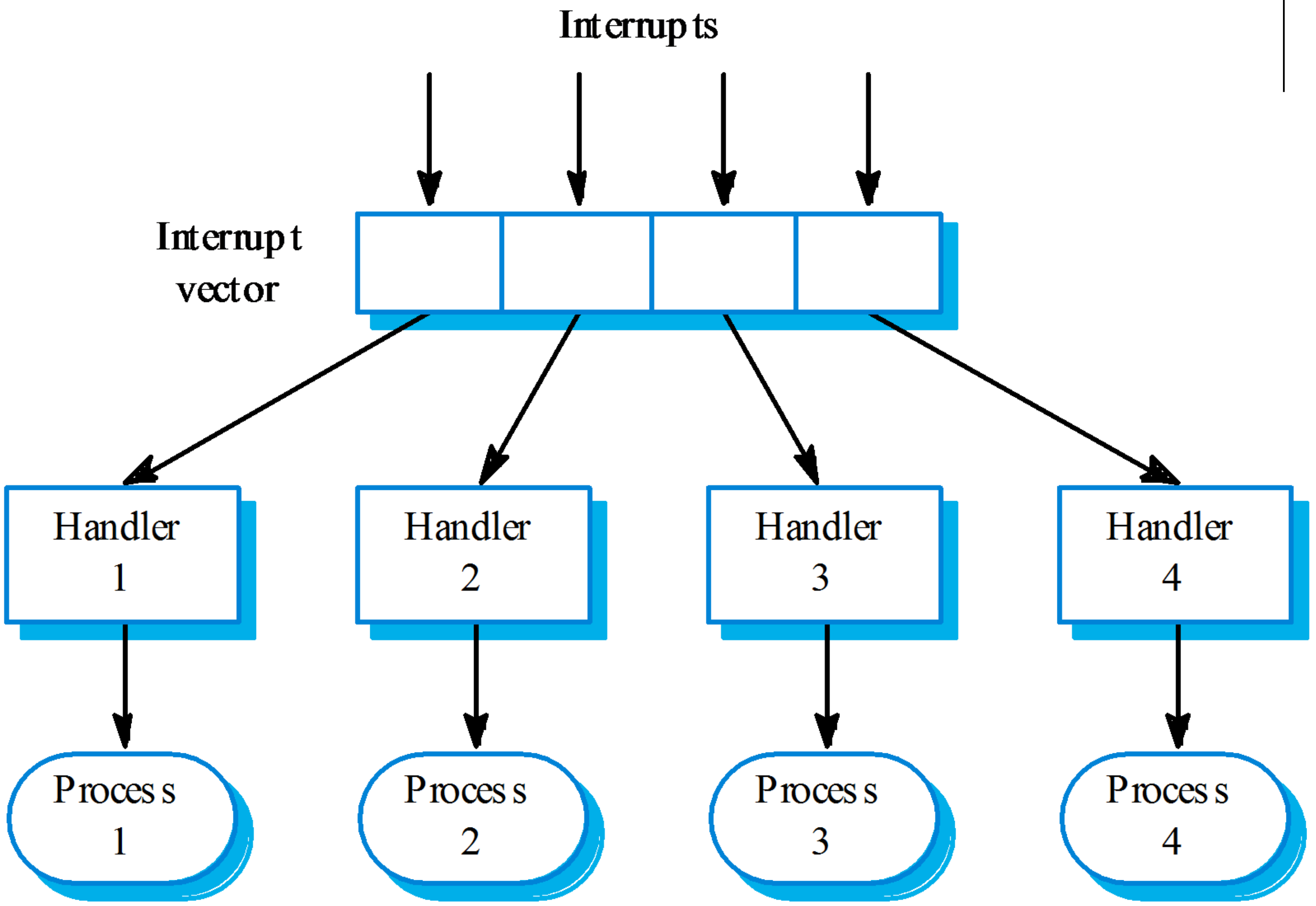
Broadcast model

- Sub-systems register an interest in specific events. When these occur, control is transferred to the sub-system which can handle the event.
- Control policy is not embedded in the event and message handler. Sub-systems decide on events of interest to them.

Selective broadcasting



Interrupt-driven control



Advantages and Disadvantages

- **Advantages:**
  - The event sensors and the event processors are separate, providing decoupled and individual functionalities.
  - The replacement and additions are independent and thus easier to perform
  - Any sensor or processing malfunction will not affect the other sensors and processors
  - High reuse potential

Advantages and Disadvantages

- **Disadvantages:**
  - It is difficult for the dispatcher to react to a myriad of sensor inputs and respond in time (especially on simultaneous inputs)
  - A dispatcher malfunction will bring the whole system down.
  - Dispatcher is performance bottleneck. It must be fast.
  - No insurance that an event will be treated.

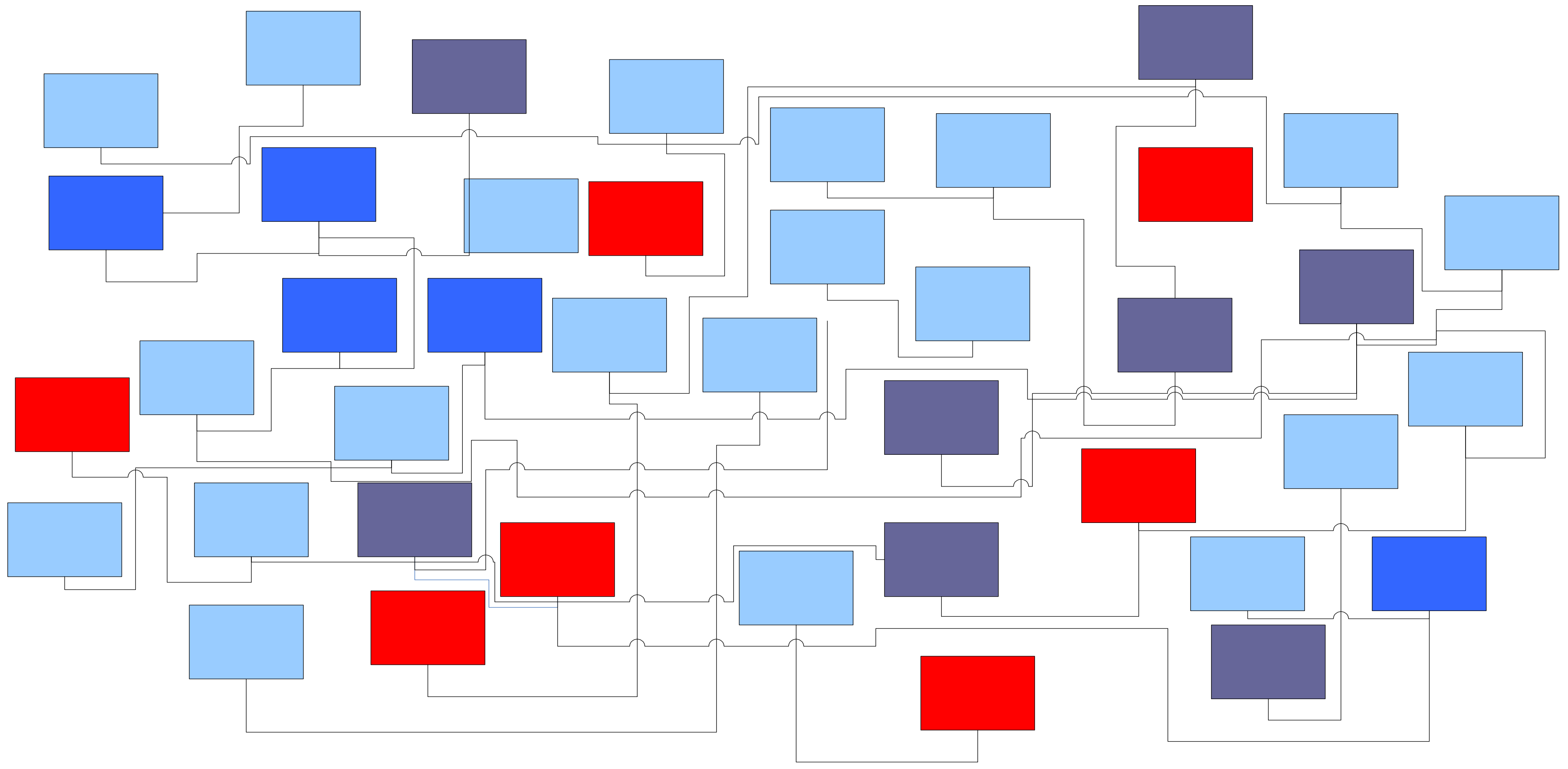
Architectural Styles:

SOA

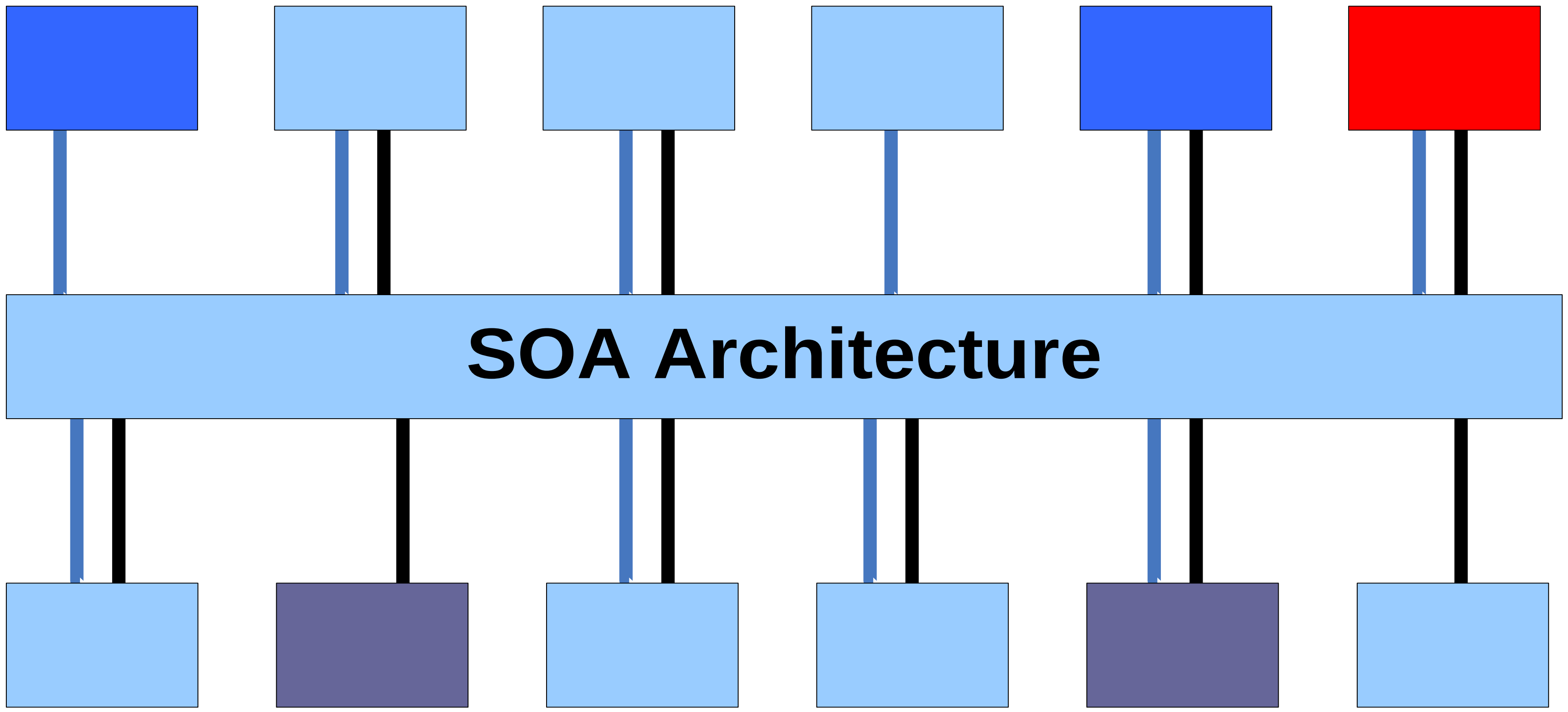
SOA

- Applications built using an SOA style deliver functionality as services that can be used or reused when building applications or integrating within the enterprise or trading partners.
- Goal: loose coupling among interacting software agents.

Legacy Integration



SOA Integration



SOA: Service

- A service
  - a reusable component that can be used as a building block to form larger, more complex business-application functionality.
  - a unit of work done by a service provider to achieve desired end results for a service consumer

SOA: Service

- Characteristics
  - Supports open standards for integration
  - Loose coupling
  - Stateless
    - The service does not maintain state between invocations.
  - Location agnostic:
    - Users of the service do not need to worry about the implementation details for accessing the service.

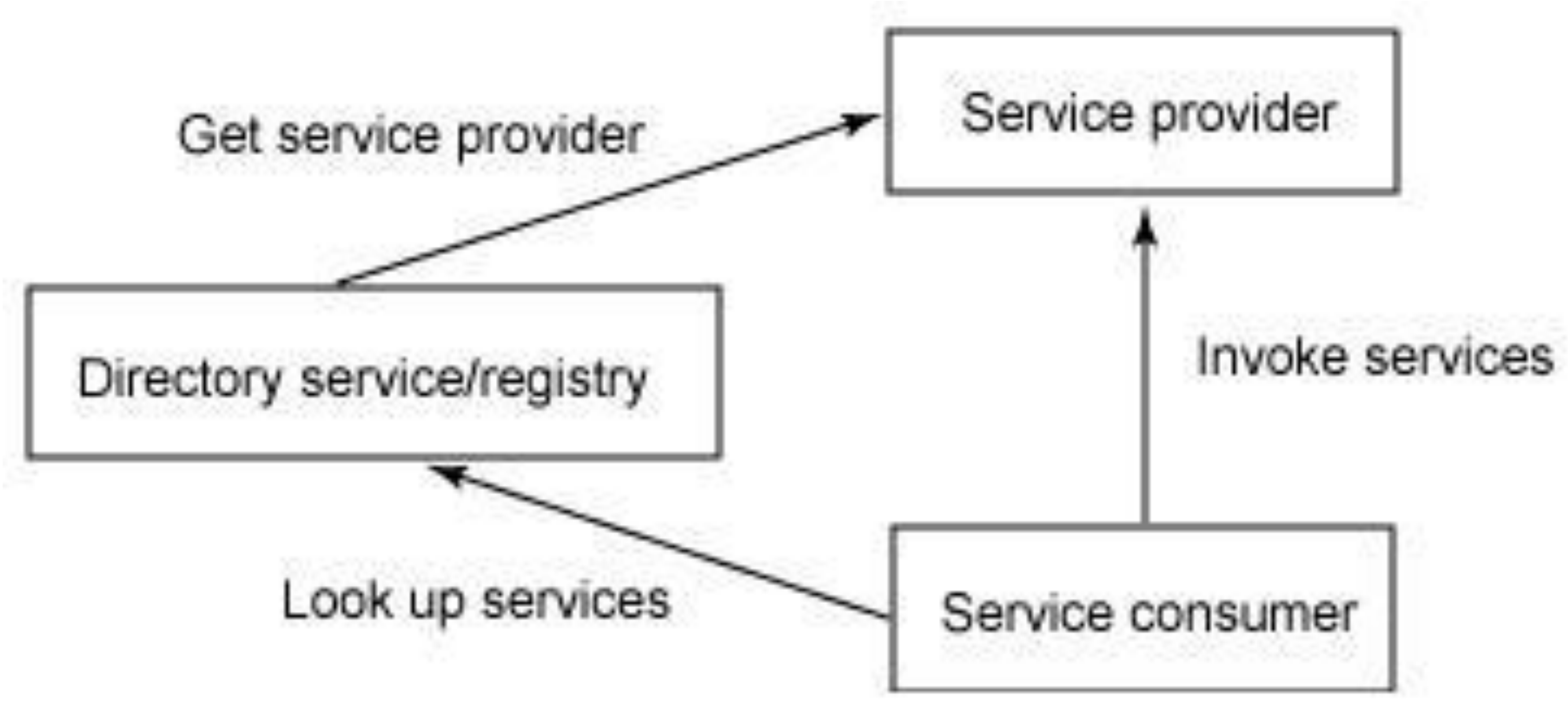
SOA

- The service-oriented architecture (SOA) consists of three roles: requester, provider, and broker.
  - *Service Provider*: A service provider allows access to services, creates a description of a service and publishes it to the service broker.

SOA

- *Service Requestor*: A service requester is responsible for discovering a service by searching through the service descriptions given by the service broker. A requester is also responsible for binding to services provided by the service provider.
- *Service Broker*: A service broker hosts a registry of service descriptions. It is responsible for linking a requestor to a service provider.

SOA


Service provider
Get service provider
Directory service/registry
Invoke services
Look up services
Service consumer

Architectural Styles:

C2

C2

- Network of concurrent components hooked together by message routing devices
- Motivation for C2 style
  - Support component-based software development
  - Support multi-lingual programming
  - Support distributed, heterogeneous environments
  - Support dynamic architectural changes

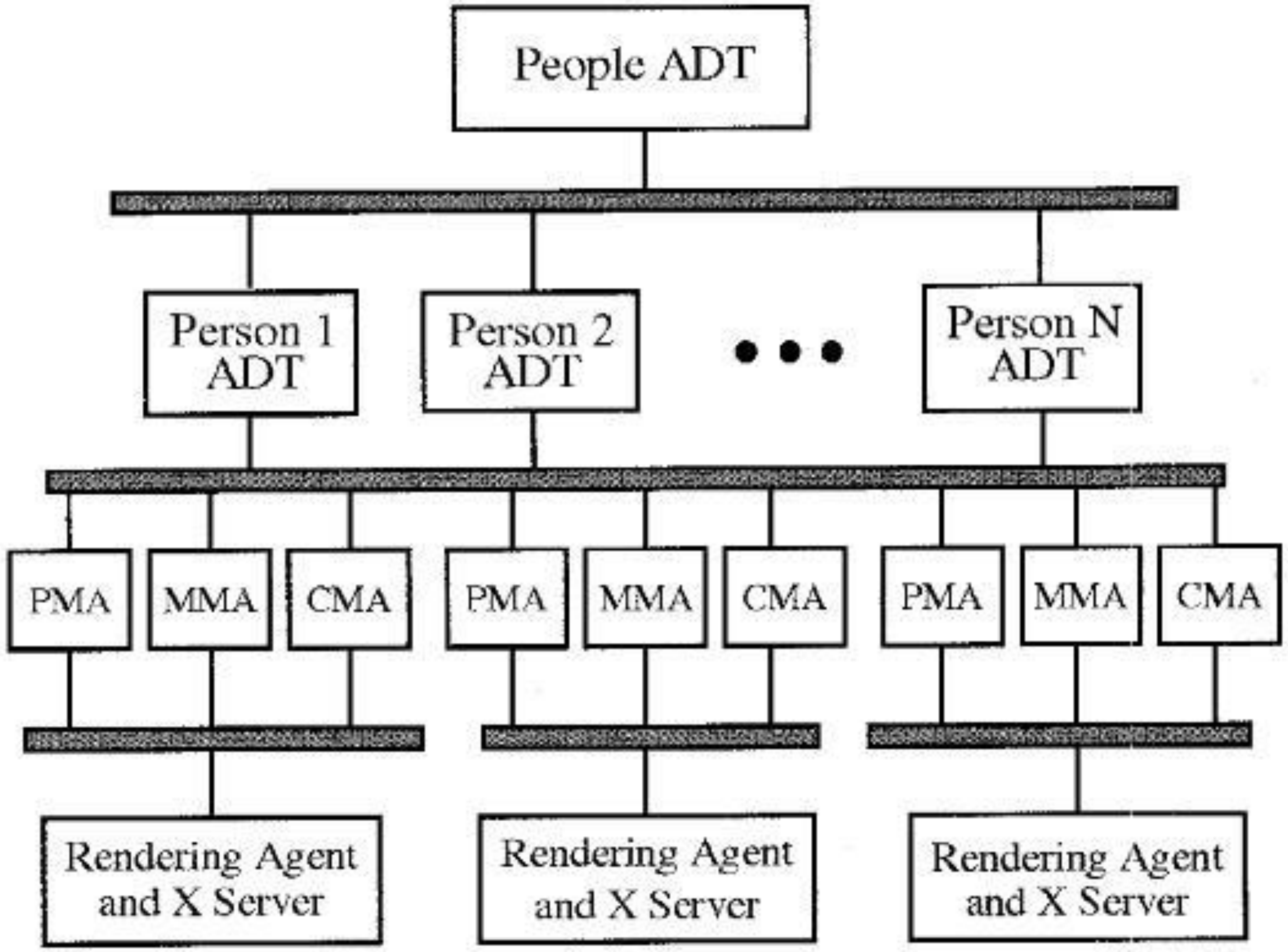
C2

- Components and connectors both have a defined top and bottom.
  - The top of a component may be connected to the bottom of a single connector.
  - The bottom of a component may be connected to the top of a single connector.
  - no bound on the number of components or connectors that may be attached to a single connector.

C2

  - Components can only communicate via connectors; direct communication is disallowed.
- Components communicate by passing messages; notifications travel down an architecture and requests up. Connectors are responsible for the routing and potential multi-cast of the messages.

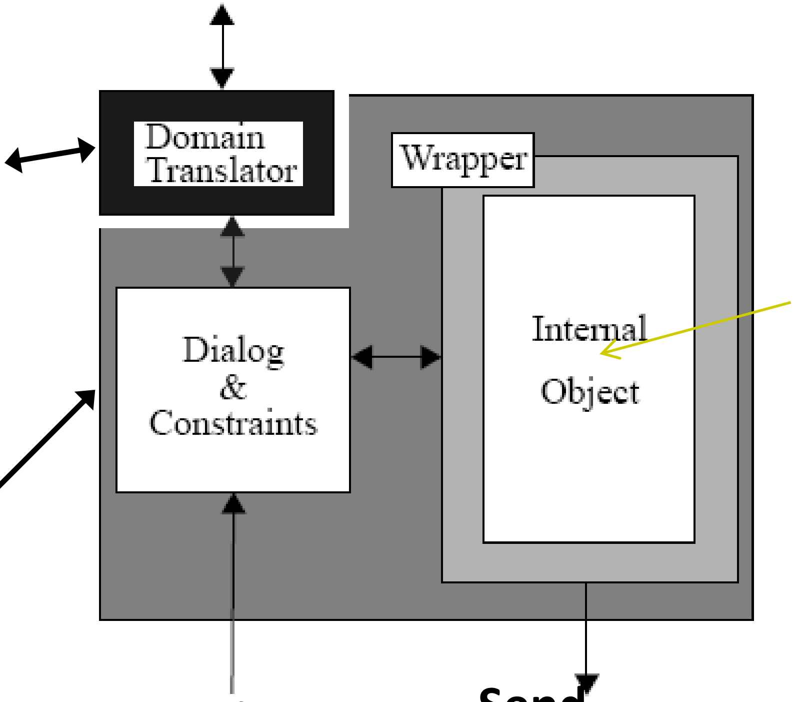
C2


People ADT
Person 1 ADT
Person 2 ADT
• • •
Person N ADT
PMA
MMA
CMA
PMA
MMA
CMA
PMA
MMA
CMA
Rendering Agent and X Server
Rendering Agent and X Server
Rendering Agent and X Server

The internal architecture of a C2 component

Receives notifications and send requests

Mapping between component's internal semantic domain and that of the connector above



Determines requests that can be received and notifications that can be sent

a) react to a notification
b) execute a request received
c) maintain some constraint

Receive requests

Send notifications

The C2 architectural style

- Connectors
  - Bind components together
  - Connected to components and other connectors
  - Routing and broadcast of messages
  - Message filtering

The C2 architectural style

- Principles
  - Substrate independence
  - Message-based communication
  - Multi-threaded
  - Multiple programming languages
  - Heterogeneous environments
  - Implementation separate from architecture