Architectural Styles

Ergude Bao

Beijing Jiaotong University

Copy right of all the slides is from Ergude Bao, School of Software Engineering, Beijing Jiaotong University
Any distribution MUST be approved by Ergude Bao, otherwise is illegal

Contents

- Definition of Architectural Style
- Architectural Styles

Definition of Architectural Style

Definition of Architectural Style

- 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 for high level design; design pattern is for detailed design
  - Styles are open-ended, so new styles will emerge
  - One architecture usually uses only one style

Benefits of Architectural Style

- Core is 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
  - Documentation reuse

Benefits of Architectural Style

- Improves development efficiency and productivity
- Provides a starting point for additional and new design ideas
- Helps to make trade-offs and pre-evaluate the design
- Diminishes risks of the design
- Promotes communications among the designers
  - Phrase such as “client-server” conveys lot of information

Architectural Styles

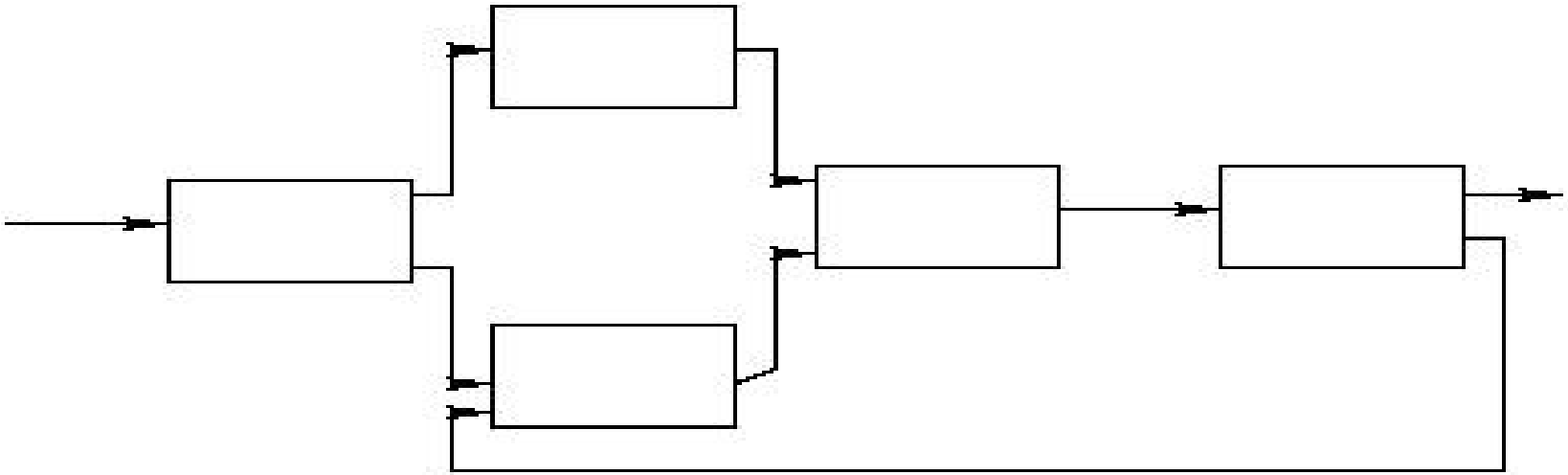
Architectural Styles

- Pipe-Filter
- Repository Model (Shared Data Store)
- Client-Server
- Model-View-Controller (MVC)
- Layered System
- Peer-to-Peer
- Event Driven
- Service-Oriented Architecture (SOA)

Pipe-Filter Style

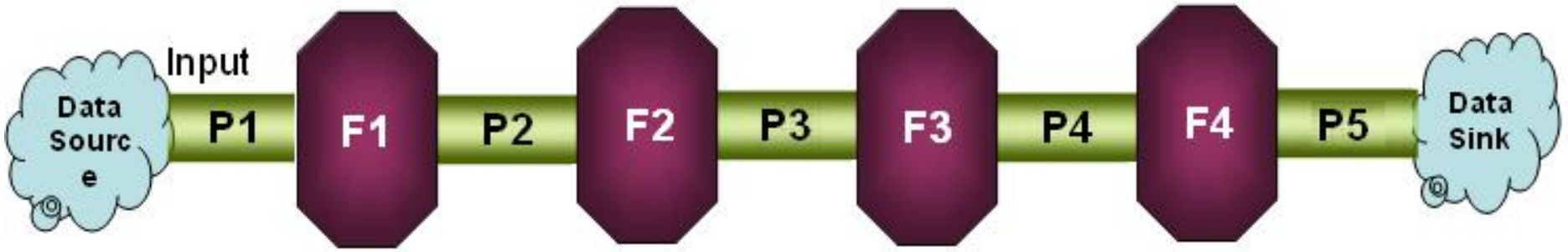
Pipe-Filter

- 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



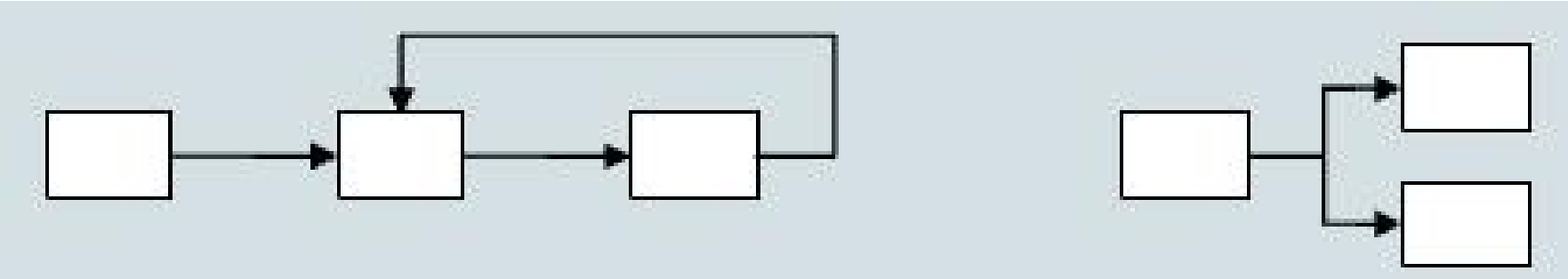
Pipe-Filter

- Component: filter for data handling
  - Reads a stream of data on its inputs and produce a stream of data on its outputs
  - Independent on each other
- Connector: pipe for data translation and transportation
  - Transmits output produced by filters to other filters



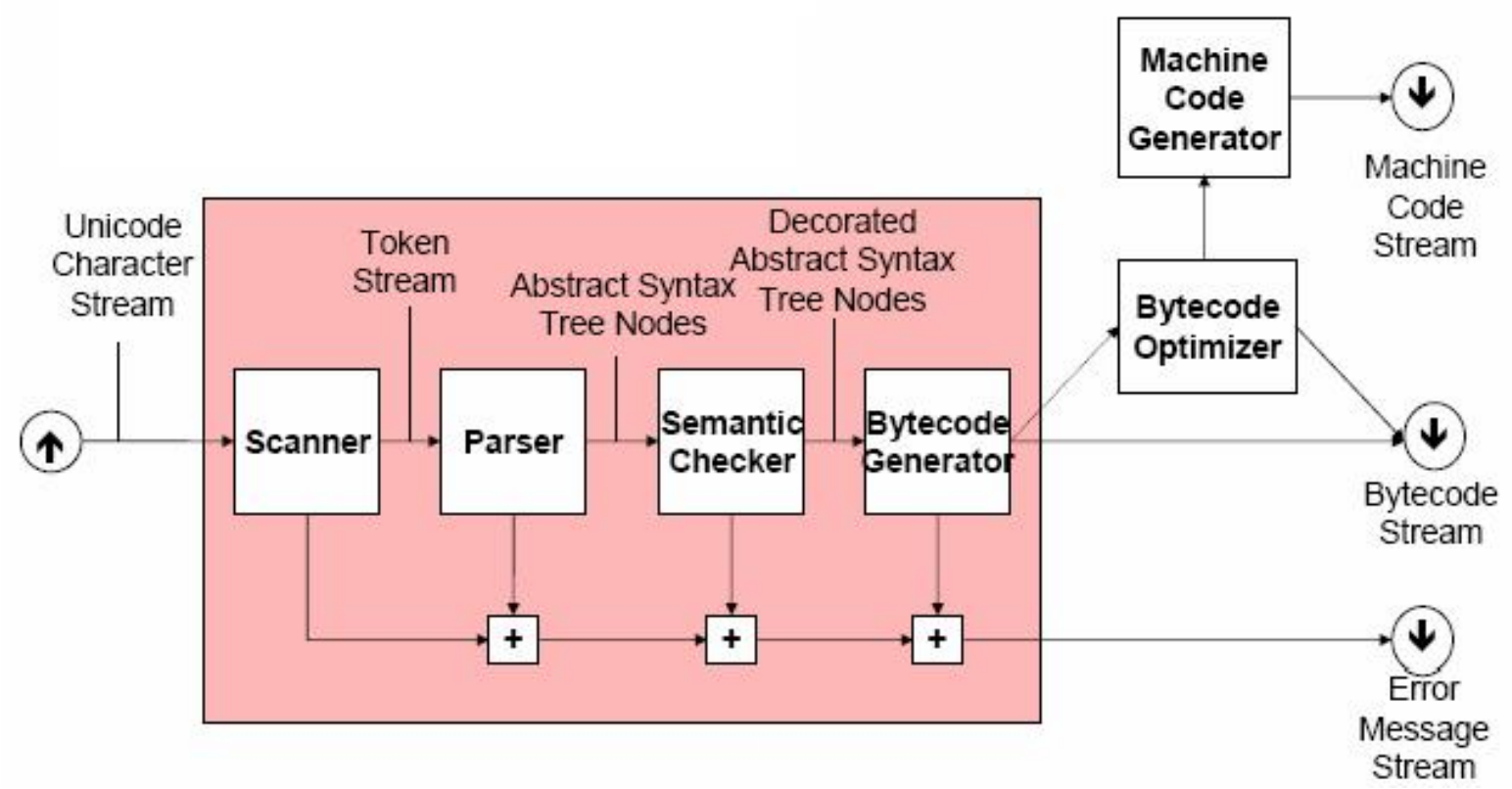
Pipe-Filter

- Topology: linear with feedback-loops or splitting pipes



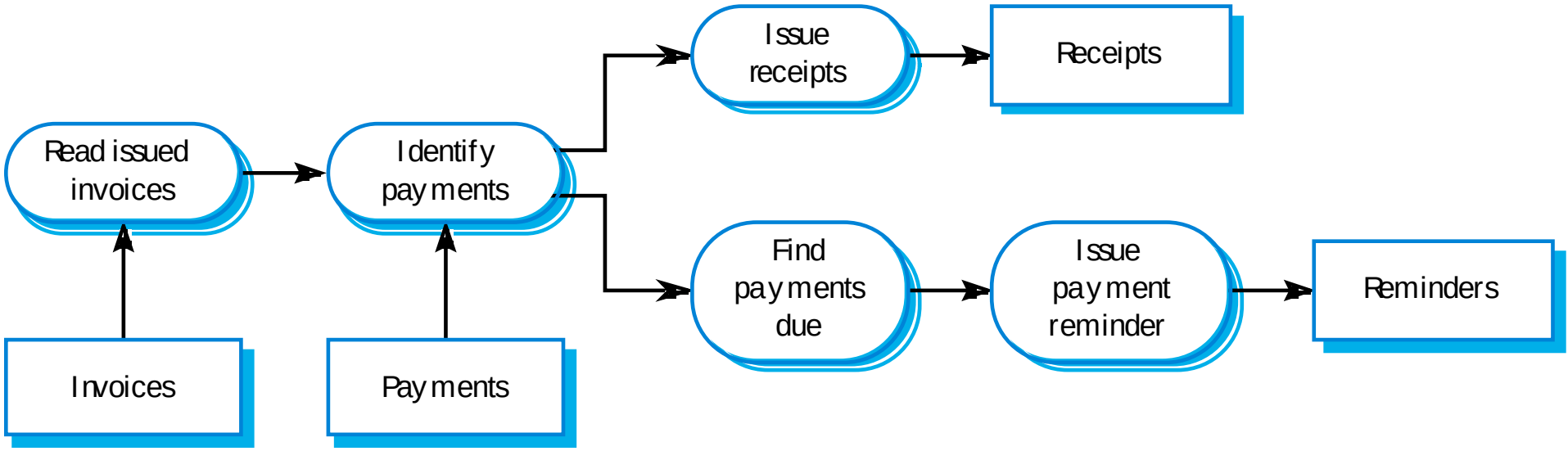
Example I

- Compiler



Example II

- Invoice processing system



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
- Simplicity: filters can be easily used in the design of either a concurrent or sequential system
- Reusability: existing filters can be reused without considering other filters
- Flexibility: existing filters can be easily redefined and/or re-routed
- Extendibility: new filters can be added easily without considering other filters

Disadvantages

- Requires a common format for data transfer along the pipeline
- Has difficulty in dispatching various inputs to corresponding filters (in supporting event-based interaction)

Repository Model Style

Repository Model

- Data exchanging among subsystems can 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 first choice is most commonly used

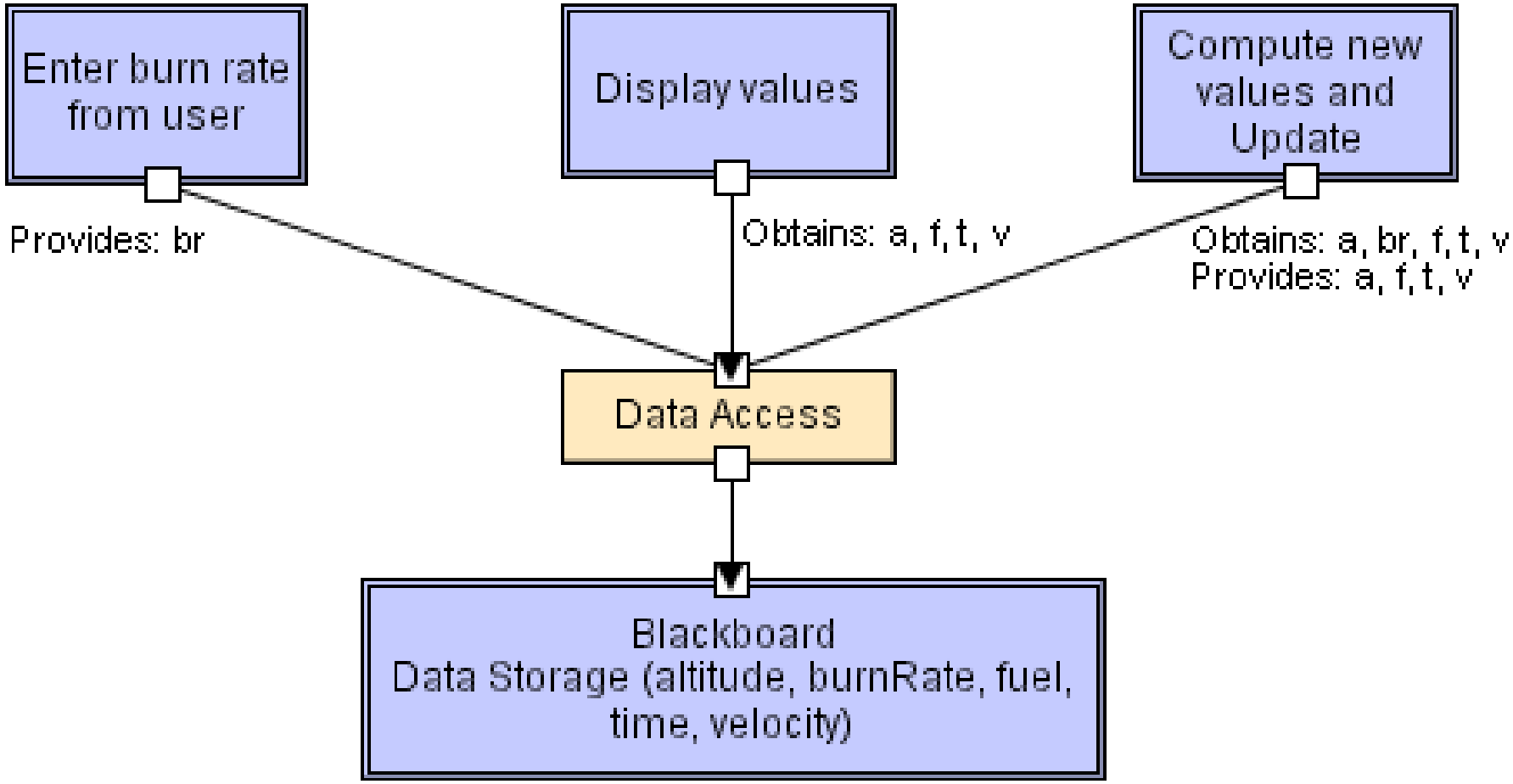
Repository Model

- Component: data store and subsystems processing data
- Connector: interaction protocols
  - All the subsystems work on a single data store
  - Any change to the data store may affect all or some of the subsystems
- Patient processing system, tax processing system, inventory control system; etc.

Two Variations

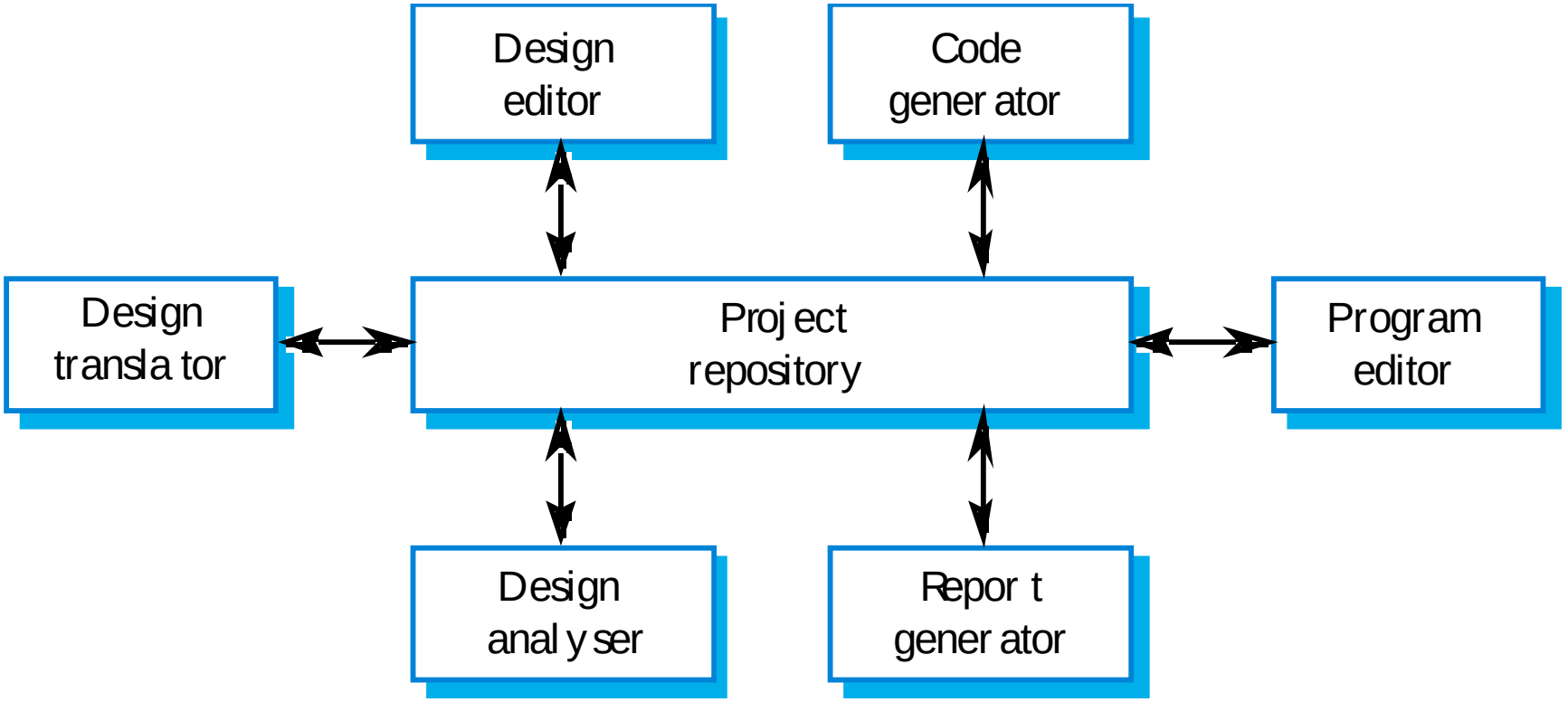
- Blackboard style: the data store alerts the participating subsystems whenever there is an event for changes
- Repository style: the participating subsystems check the changes in data store

Blackboard Style: Example



Repository Style: Example

- Computer Aided Software Engineering (CASE) toolset architecture



Advantages

- Independent subsystems are cohesive within themselves and coupling is restricted to the shared data
- Single data store makes it efficient to share the data

Disadvantages

- Any change in the shared data requires agreement and, potentially, changes in all or some the subsystems
  - Data evolution is difficult and expensive
- If the data store fails, all subsystems are affected and possibly have to stop
  - Redundant database and/or good backup and recovery procedures are required

Client-Server Style

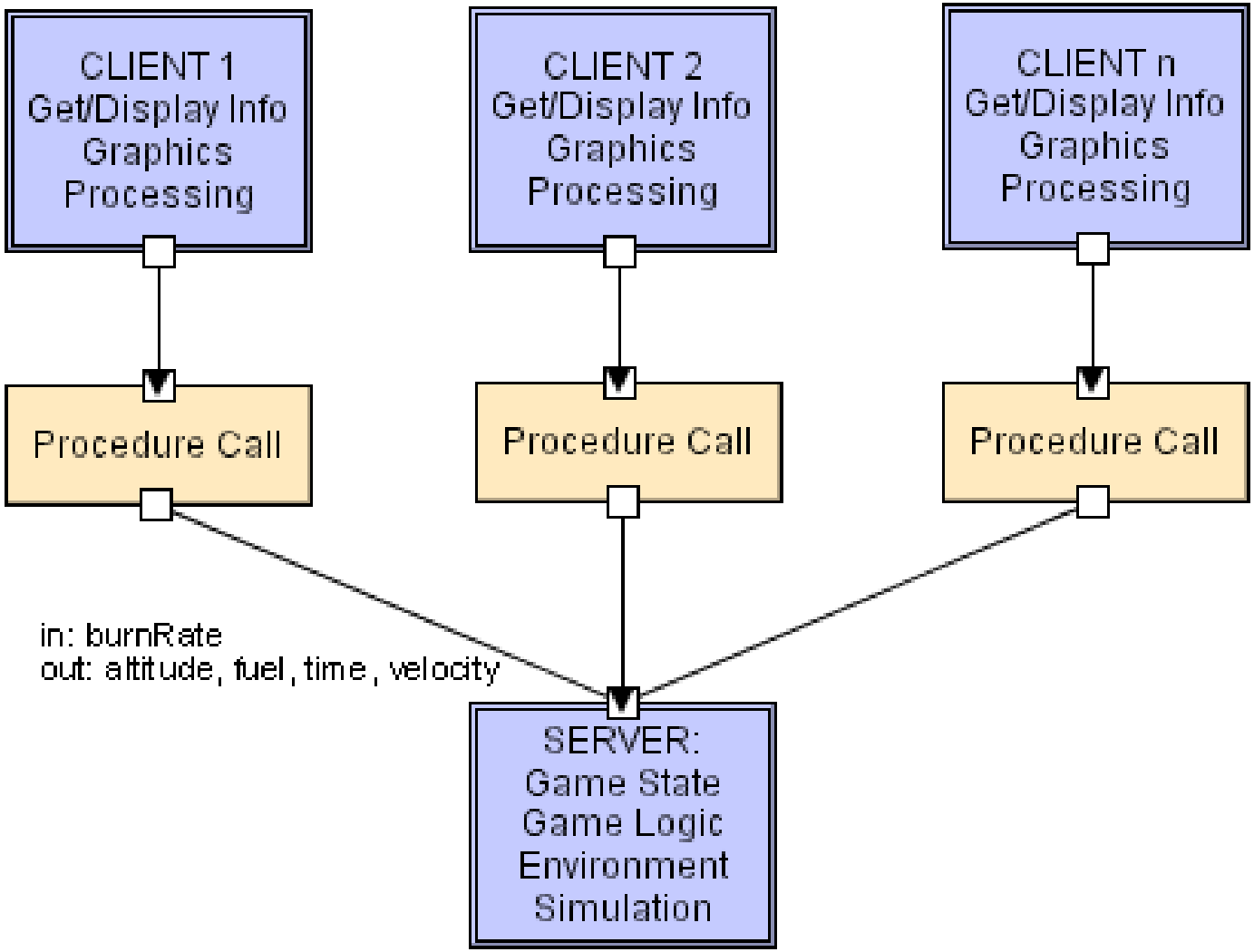
Client-Server

- Component: client and server
  - Client: subsystem that makes requests (to the servers) and handles input/output with the system environment
  - Server: subsystem that responses requests from clients
- Connector: network interaction protocols
  - Client depends on servers
  - Client knows servers' identities, while server does not know number or identities of clients

Client-Server

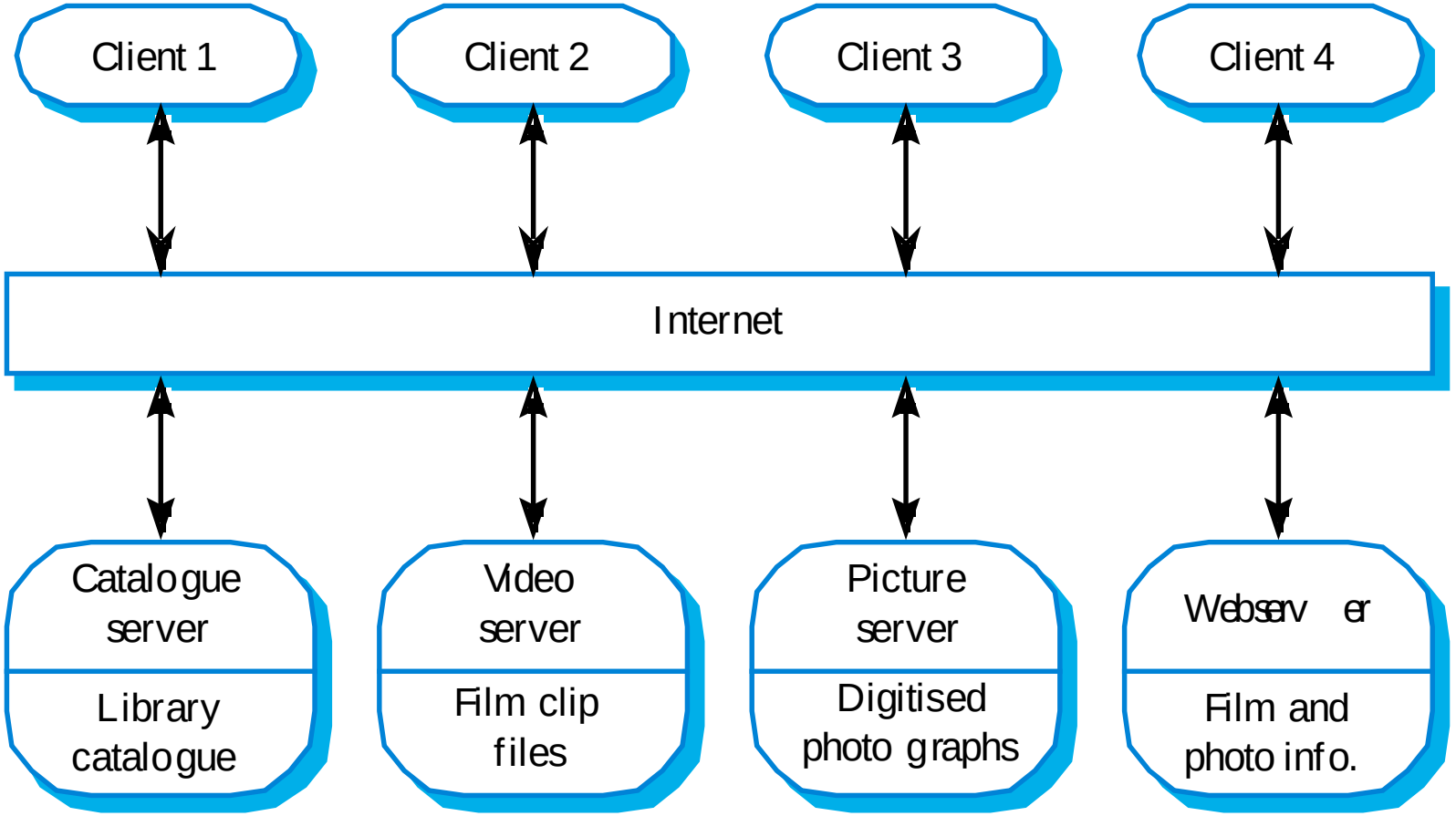
- Topology: many clients to many servers
  - There is no connection between clients
- Synchronicity: synchronous or asynchronous
- Security: typically controlled by server

Example I


CLIENT 1
Get/Display Info
Graphics
Processing
CLIENT 2
Get/Display Info
Graphics
Processing
CLIENT n
Get/Display Info
Graphics
Processing
Procedure Call
Procedure Call
Procedure Call
in: burnRate
out: altitude, fuel, time, velocity
SERVER:
Game State
Game Logic
Environment
Simulation

Example II

- Film and Picture Library



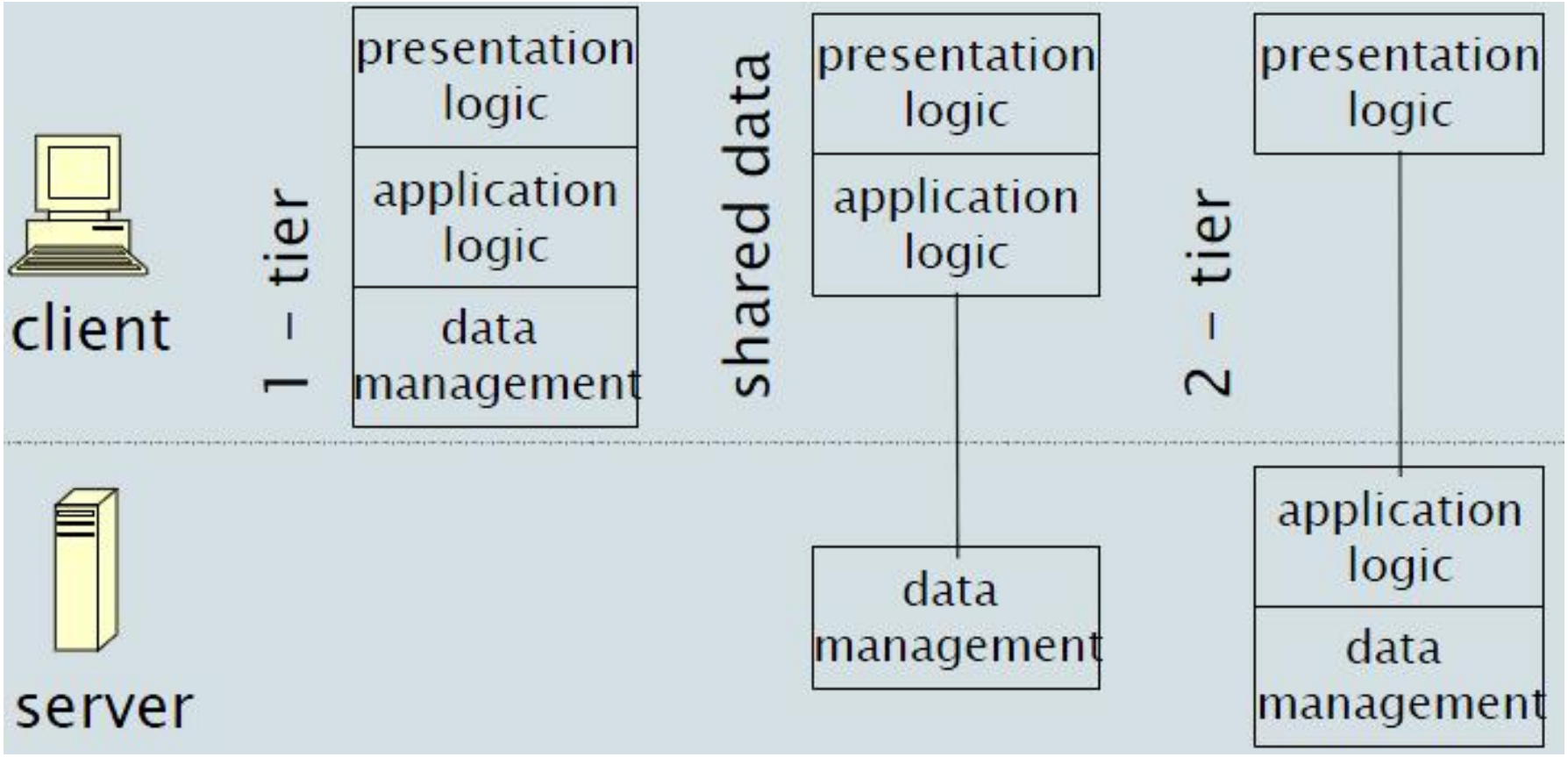
Two Variations

- Two tiers
  - Thin client
  - Thick client
- Three tiers
  - Thin client
  - Thick client

Two Tier Client-Server

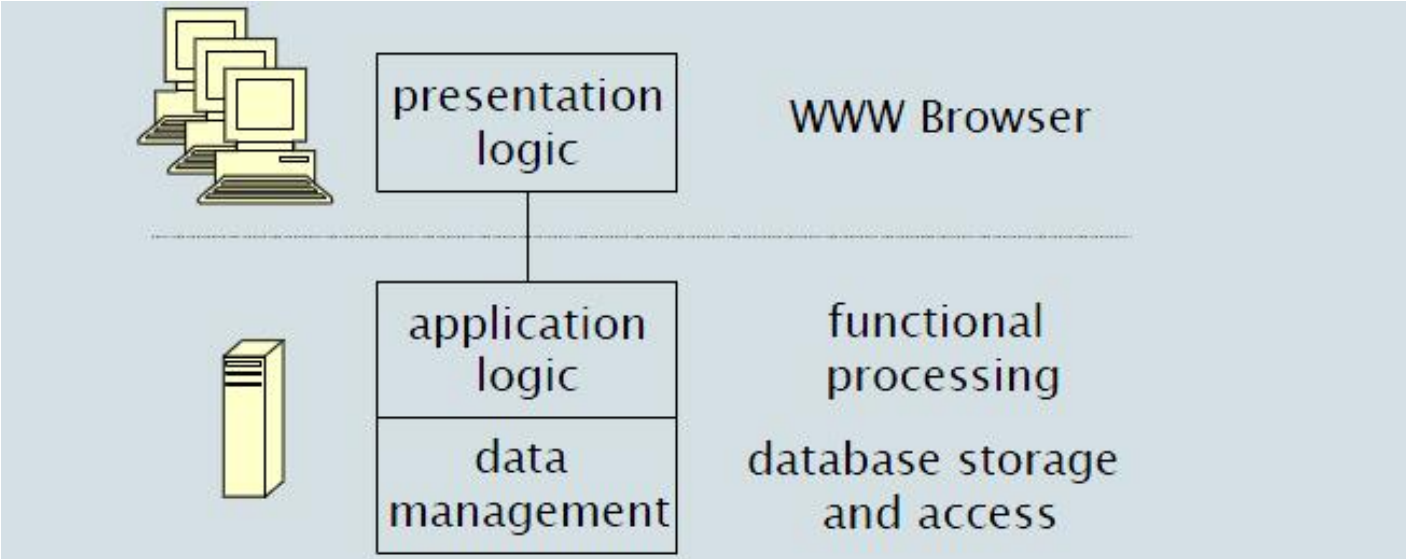
- Tier 1: user interface of local computer environment in clients
- Tier 2: application and database management in servers

Two Tier Client-Server



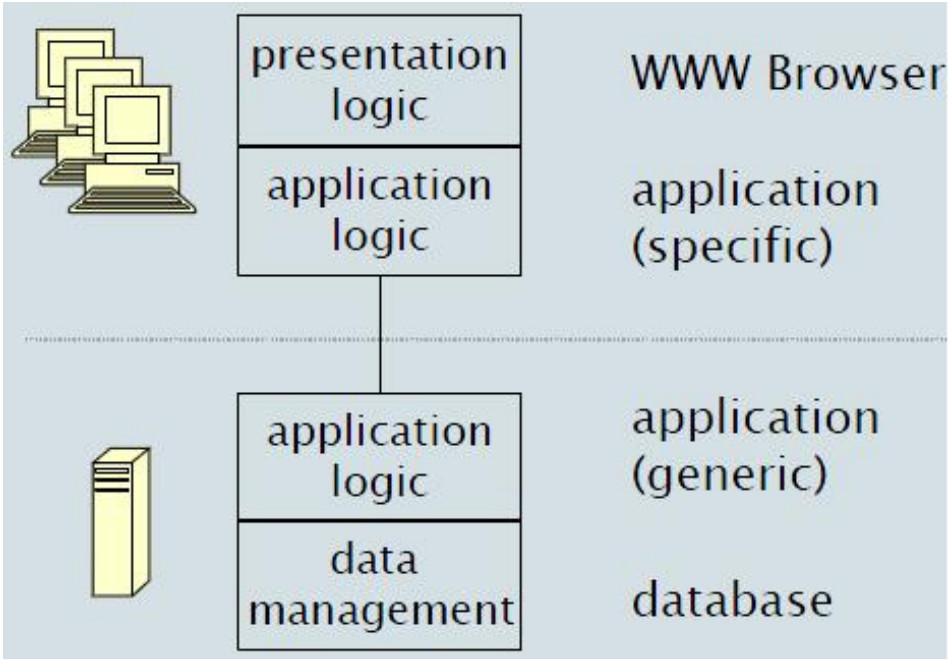
Two Tier Client-Server: Thin Client

- Largest part of processing at the server-side



Two Tier Client-Server: Thick Client

- Significant processing at the client-side



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

Two Tier Client-Server: Advantages

- Makes effective use of network resources
- Easy to add new clients/servers or upgrade existing clients/servers
- Allows sharing of data between multiple clients

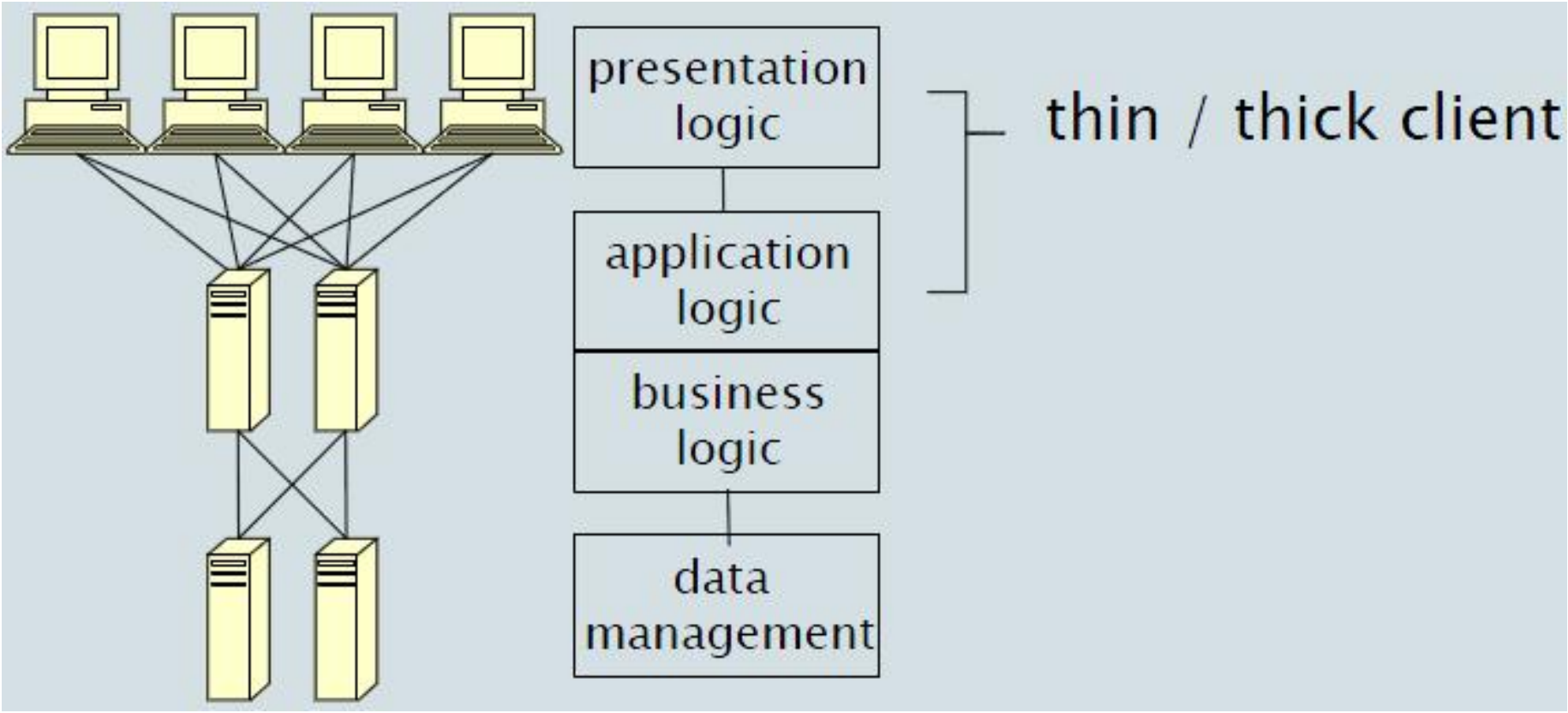
Two Tier Client-Server: Disadvantages

- Requires quality hardware and redundant management in each server
- Hard to find out what servers and services are available
  - No central registration of names and services
- Performance and scalability is limited by server and network capacity

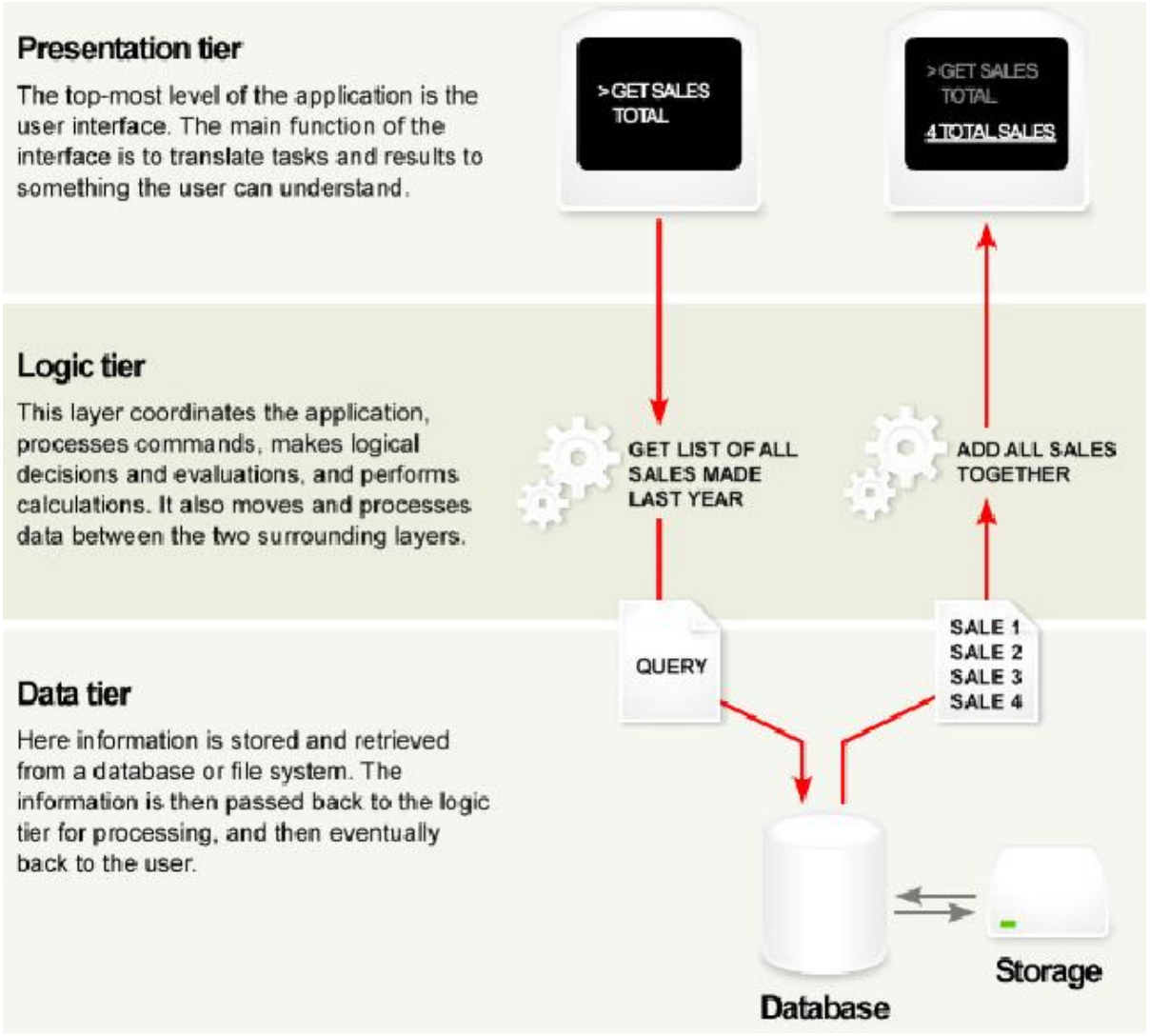
Three Tier Client-Server

- Presentation tier: responsible for displaying user interface
- Data access tier: Responsible for retrieving, modifying and deleting data and sending results to the presentation tier
- Business logic tier: Responsible for processing the data retrieved and sending to the presentation tier

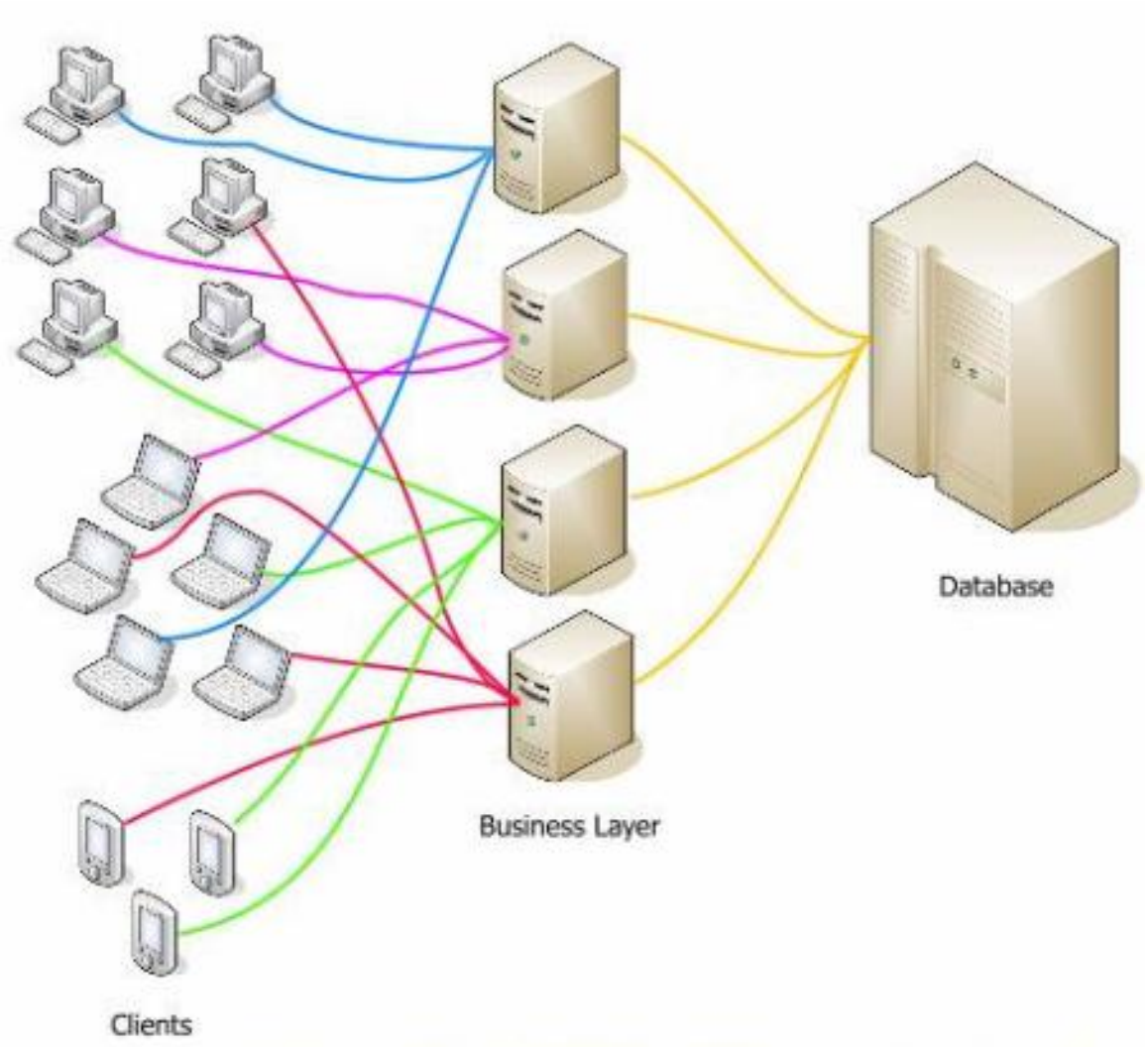
Three Tier Client-Server



Three Tier Client-Server: Example



Three Tier Client-Server: Deployment



Three-Tier Client-Server: Advantages

- Better performance and scalability
- Security measures can be centrally allocated
- Parallel development of different tiers

Three-Tier Client-Server: Disadvantages

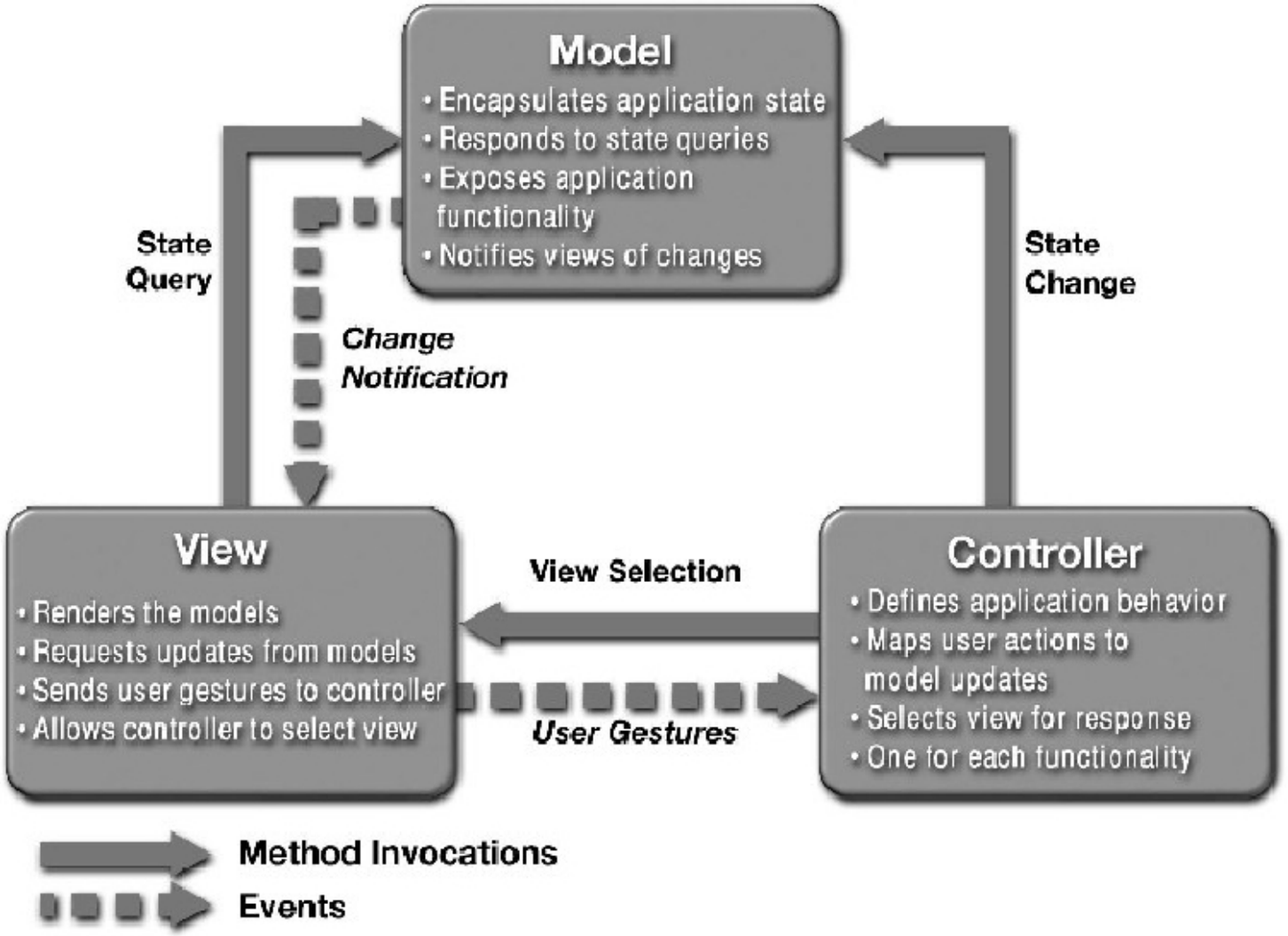
- The same with the two-tier client-server style

Model-View-Controller (MVC) Style

Model-View-Controller (MVC)

- Component: model, view and controller
  - Separates the modeling of the domain, the presentation, and the actions based on user input into three separate groups
- Connector: interaction protocols

Model-View-Controller (MVC)



Advantage

- View, controller, and model are separate components that allow modifications without significantly disturbing the others
  - The view and controller can keep on partially functioning even if the model is down

Disadvantage

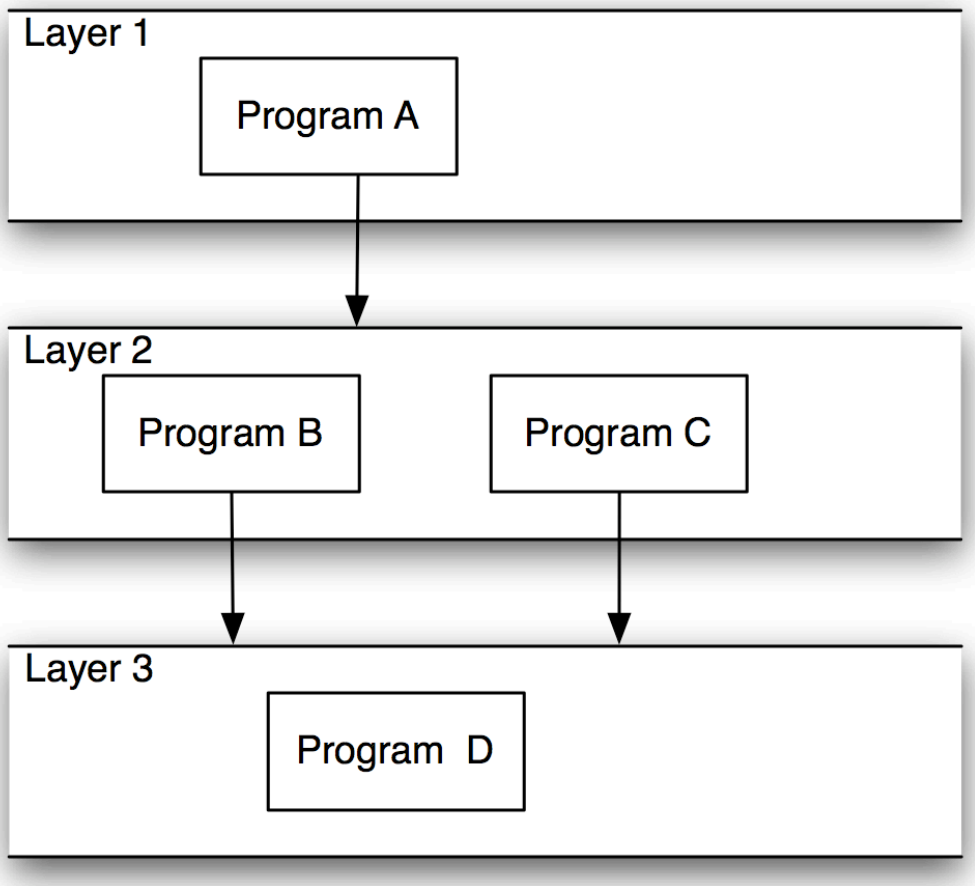
- Heavily dependent on the development and production system environment and tools that match the MVC architecture (e.g. TomCat, .Net, Rail, etc.)

Layered System Style

Layered System

- Component: layer
  - Each layer provides a related set of services
- Connector: interaction protocols between layers
  - Each layer may only use the layer(s) below it

Layered System



Two Variations

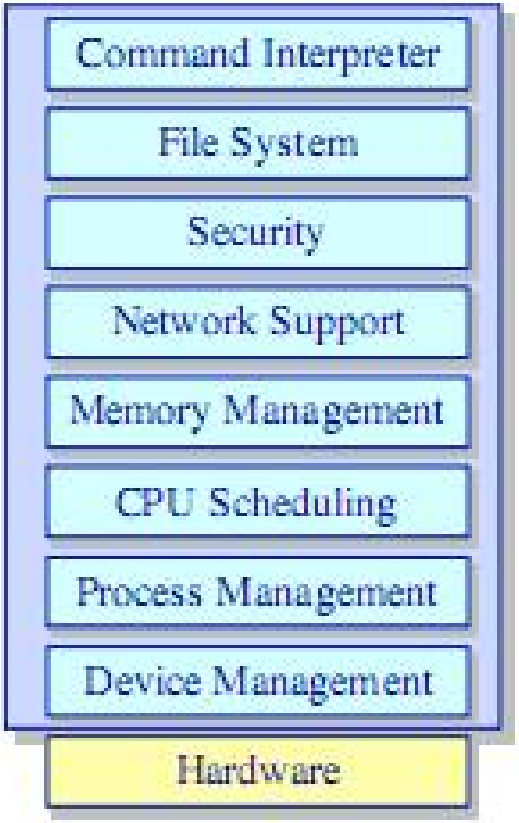
- Strict layered system style: if any layer only uses the layer directly below it
- Relaxed layer system style: if a layer may use any of the layers below it

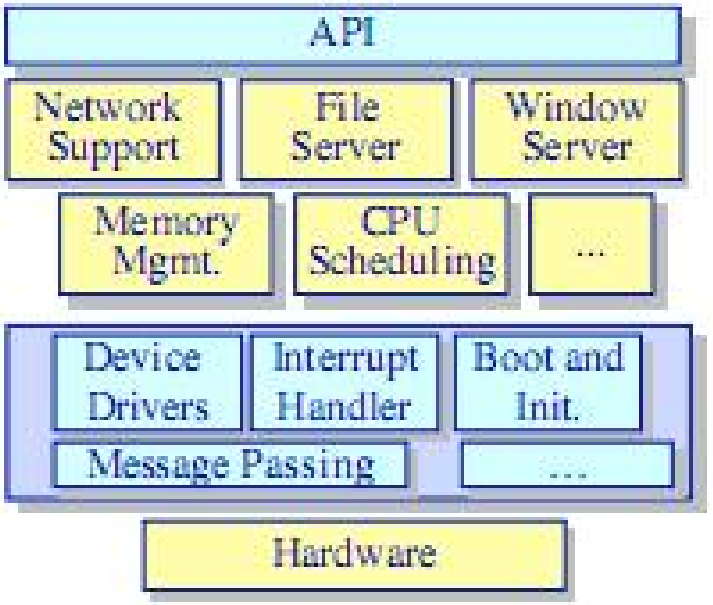
“Multi-Tier Client-Server”

- Each layer acts as a
  - Server: service provider to layer(s) “above”
  - Client: service consumer of layer(s) “below”

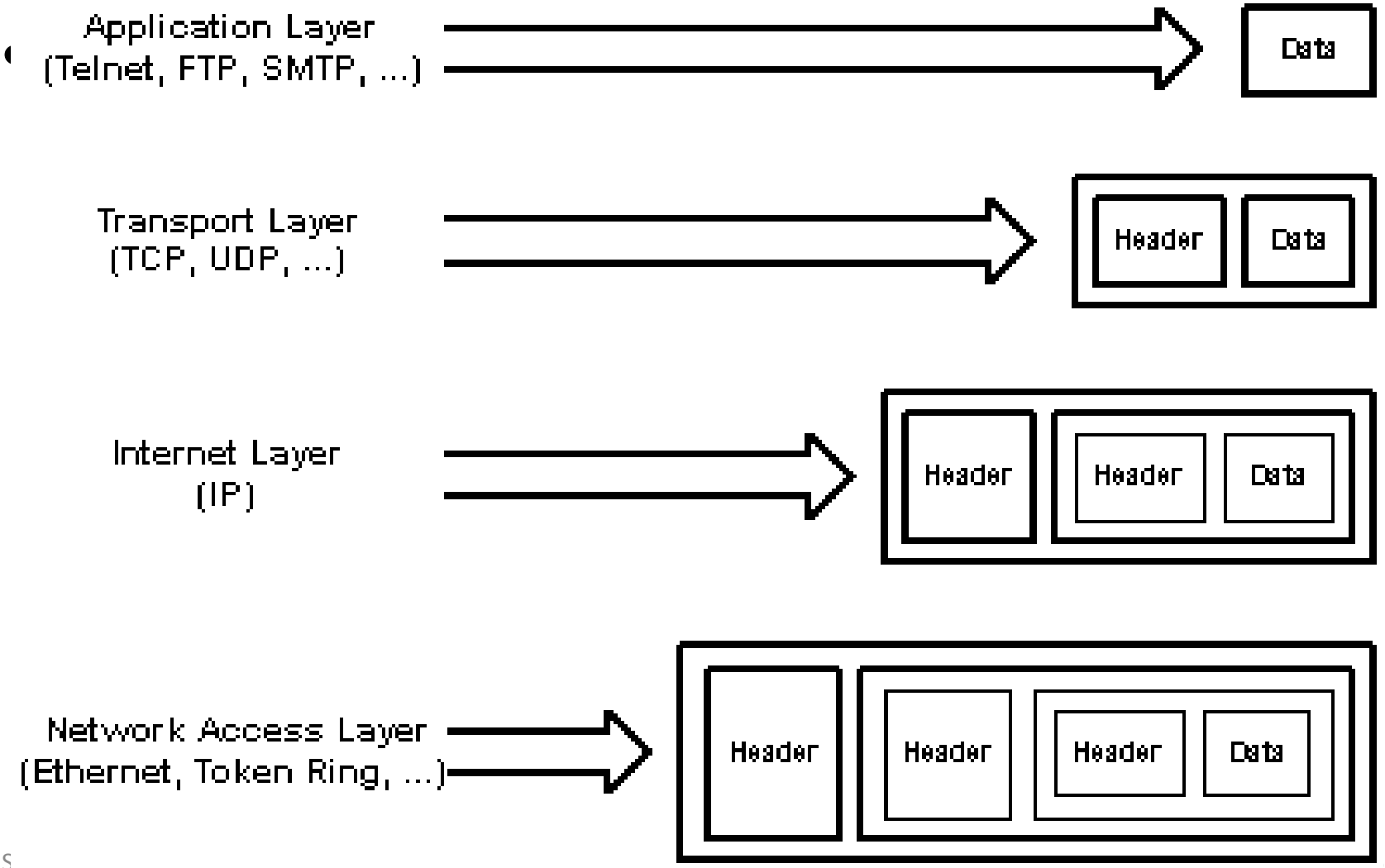
Example I

- Monolithic OS & Micro-kernel OS



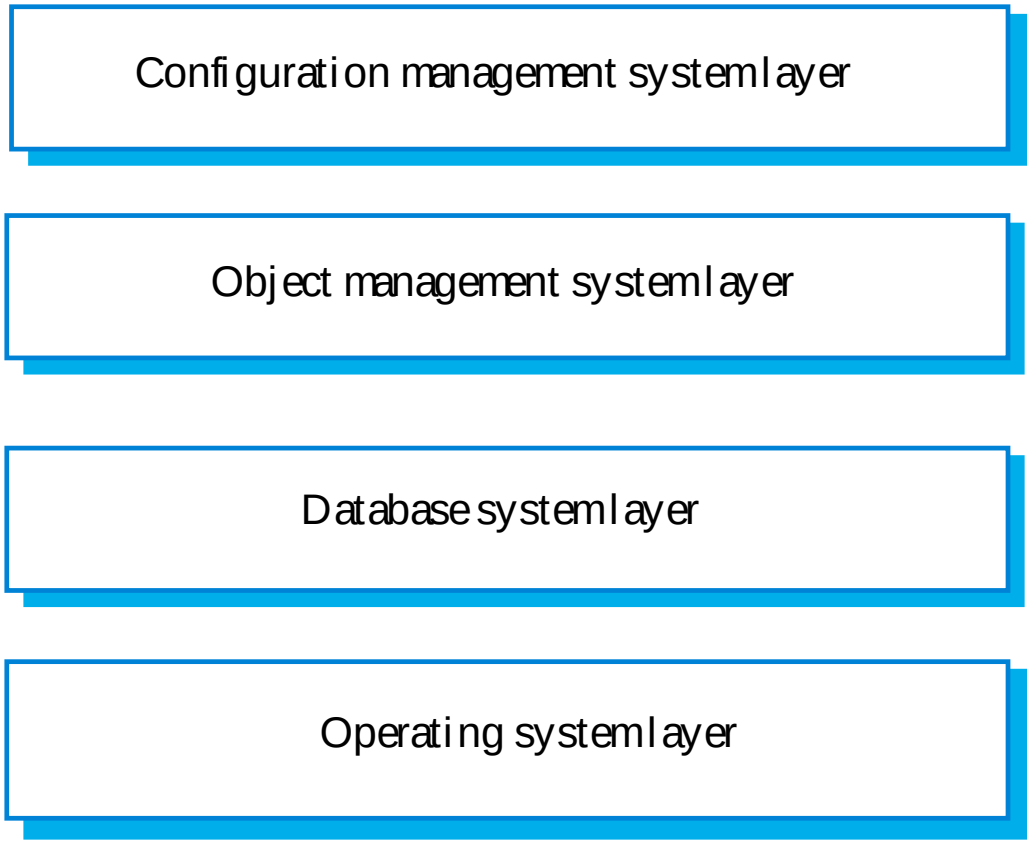


Example II


Application Layer
(Telnet, FTP, SMTP, ...)
Data
Transport Layer
(TCP, UDP, ...)
Header
Data
Internet Layer
(IP)
Header
Header
Data
Network Access Layer
(Ethernet, Token Ring, ...)
Header
Header
Header
Data

Example III

- Version management system



Advantages

- Each layer is selected to be a set of related services, so it provides high degree of cohesion within the layer
- Layers may use only lower layers, so it constrains the amount of coupling
- Each layer, being cohesive and is coupled only to lower layers, makes it easier to add, modify, and/or reuse a layer

Disadvantages

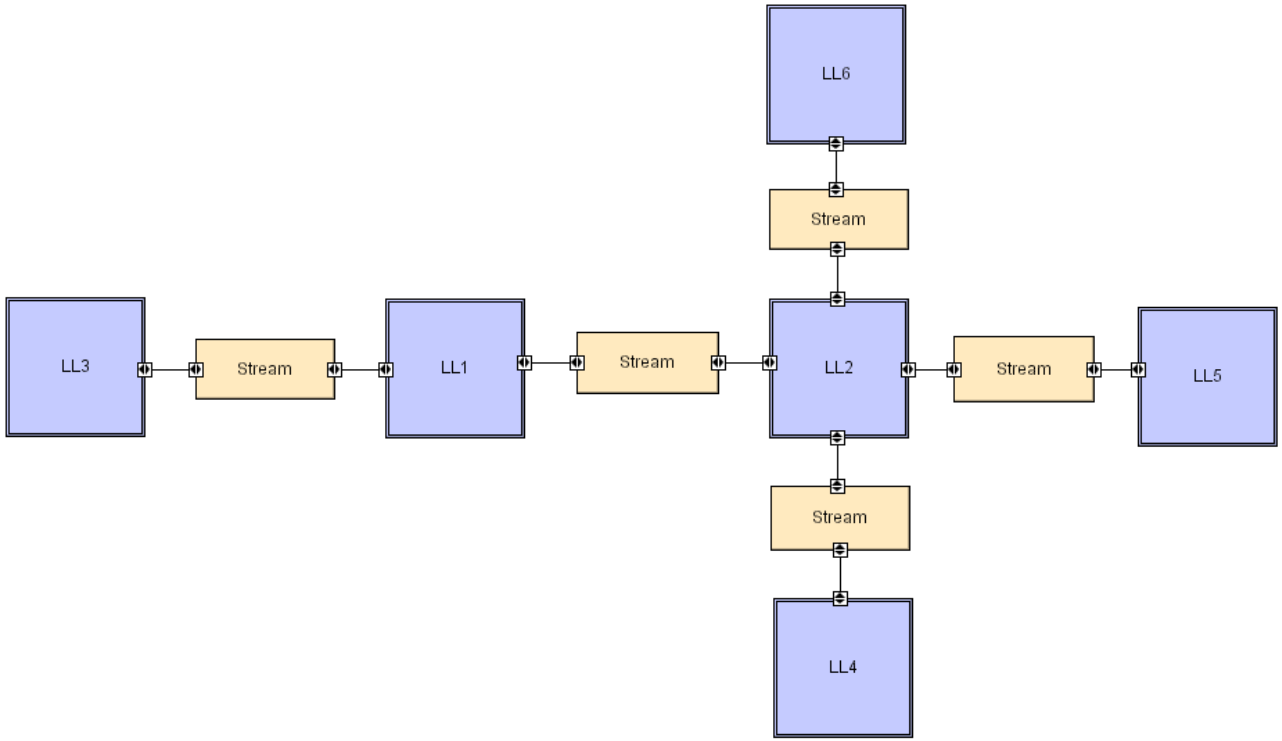
- Strict layered style may cause performance problem depending on the number of layers
- Not always easy to design with clean layers

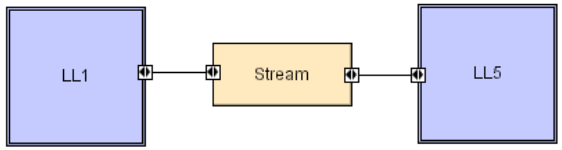
Peer-to-Peer Style

Peer-to-Peer

- Component: peer
  - Each peer maintains its own data store, as well as a dynamic routing table of other peers' addresses
  - No distinction between peers
- Connector: network protocols
  - Each peer can act as both a server and a client
- BitTorrent

Example


LL6
Stream
LL3
Stream
LL1
Stream
LL2
Stream
LL5
Stream
LL4


LL1
Stream
LL5

Event-Driven Style

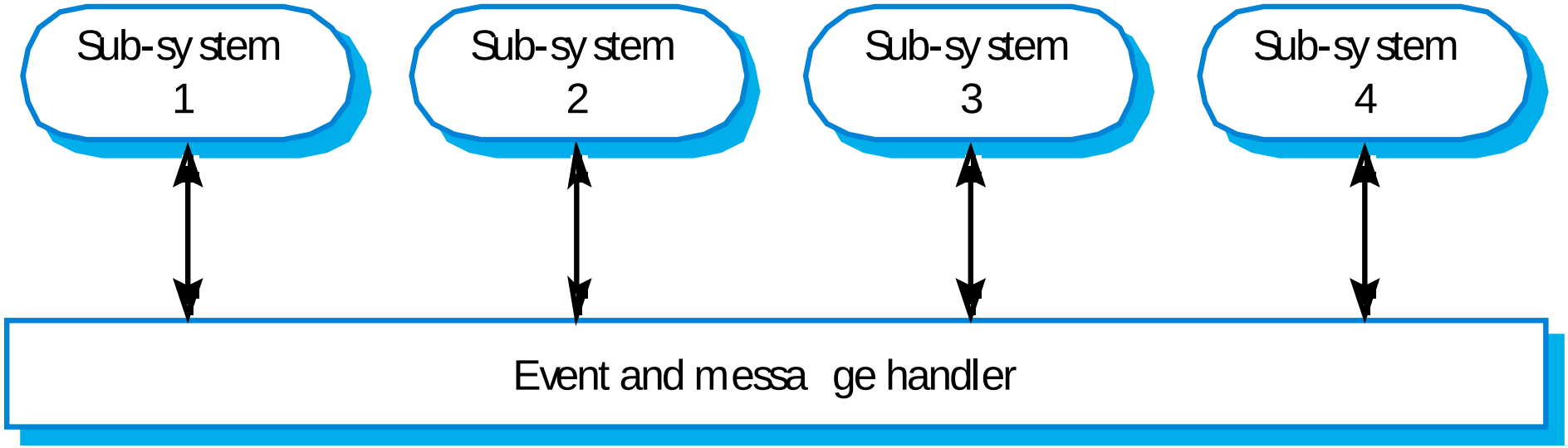
Event-Driven

- Component: event dispatcher and processor
  - Events may be a simple notification or may include associated data
  - Events may be “registered” or “unregistered” by components
- Connector: interaction protocols
  - Events may be prioritized based on constraints such as time
  - Events may require synchronous or asynchronous processing
- Usually useful for real-time systems such as: airplane control; medical equipment monitor; home monitor; embedded device controller; game; etc.

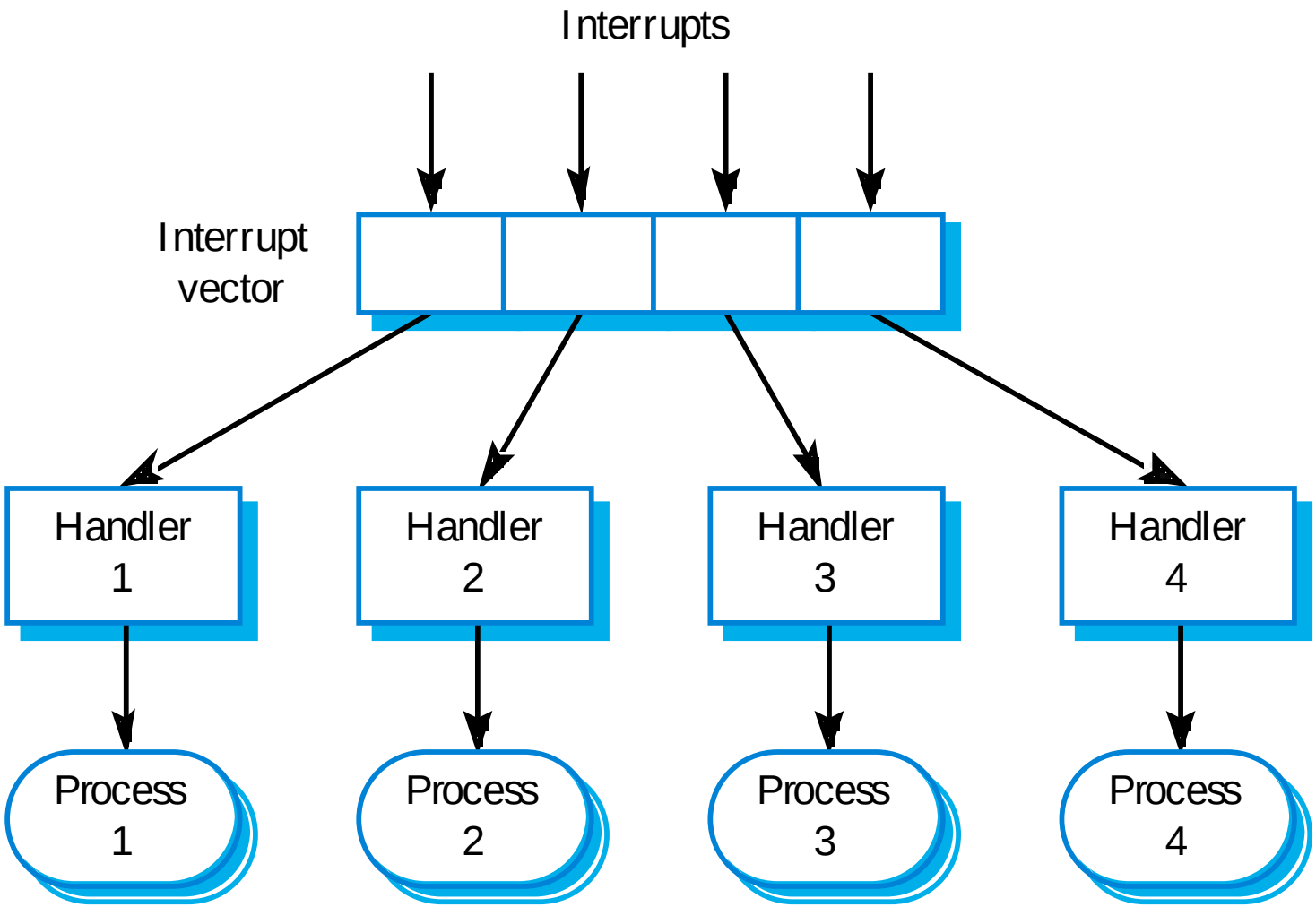
Two Variations

- Broadcast style: an event is dispatched to all processors
- Interrupt-driven style: an interrupt is sent to an event dispatcher (interrupt handler) and passed to some of the processors

Broadcast: Example



Interrupt-Driven: Example



Advantages

- The event dispatcher and the event processors are separate, guaranteeing low decoupling
- Malfunction of any processor will not affect the other processors

Disadvantages

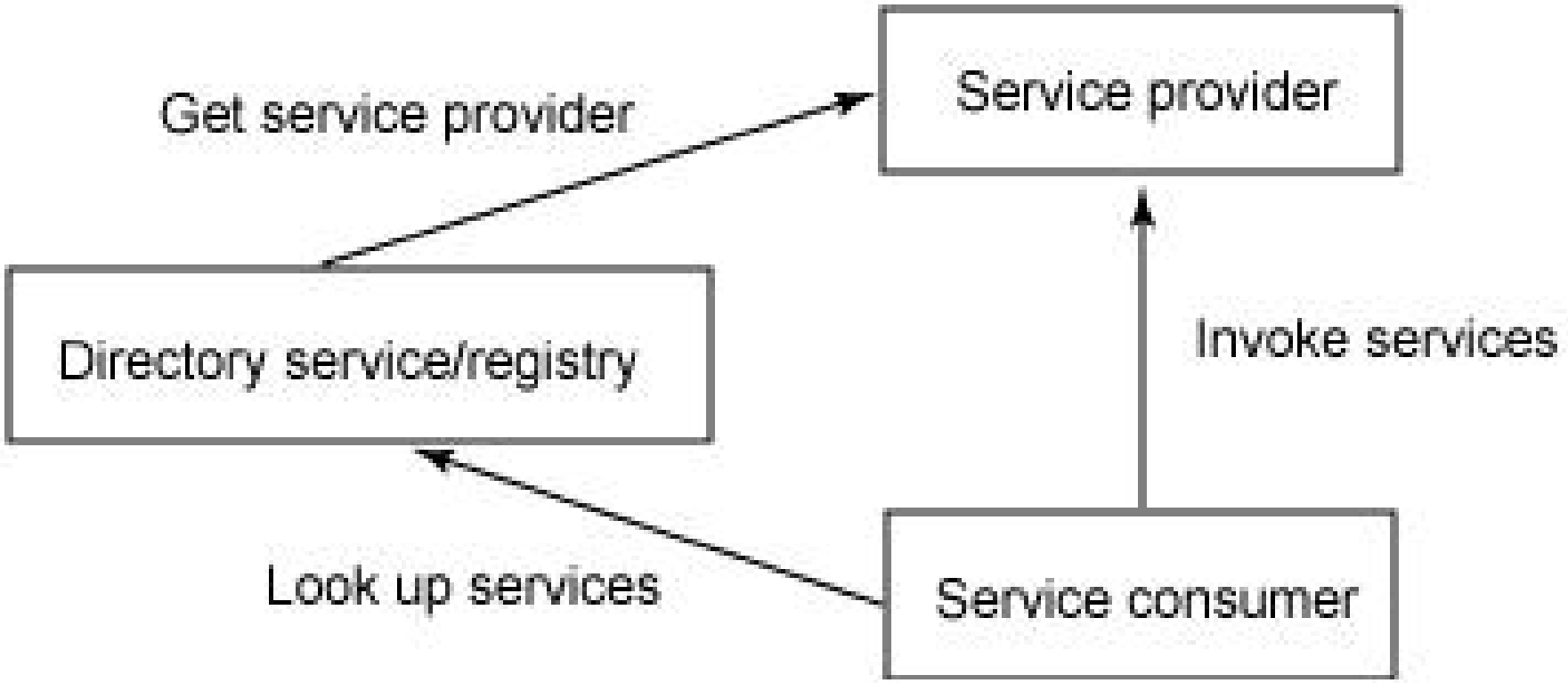
- Dispatcher is performance bottleneck
- Malfunction of the dispatcher will bring the whole system down

Service-Oriented Architecture (SOA) Style

Service-Oriented Architecture (SOA) Style

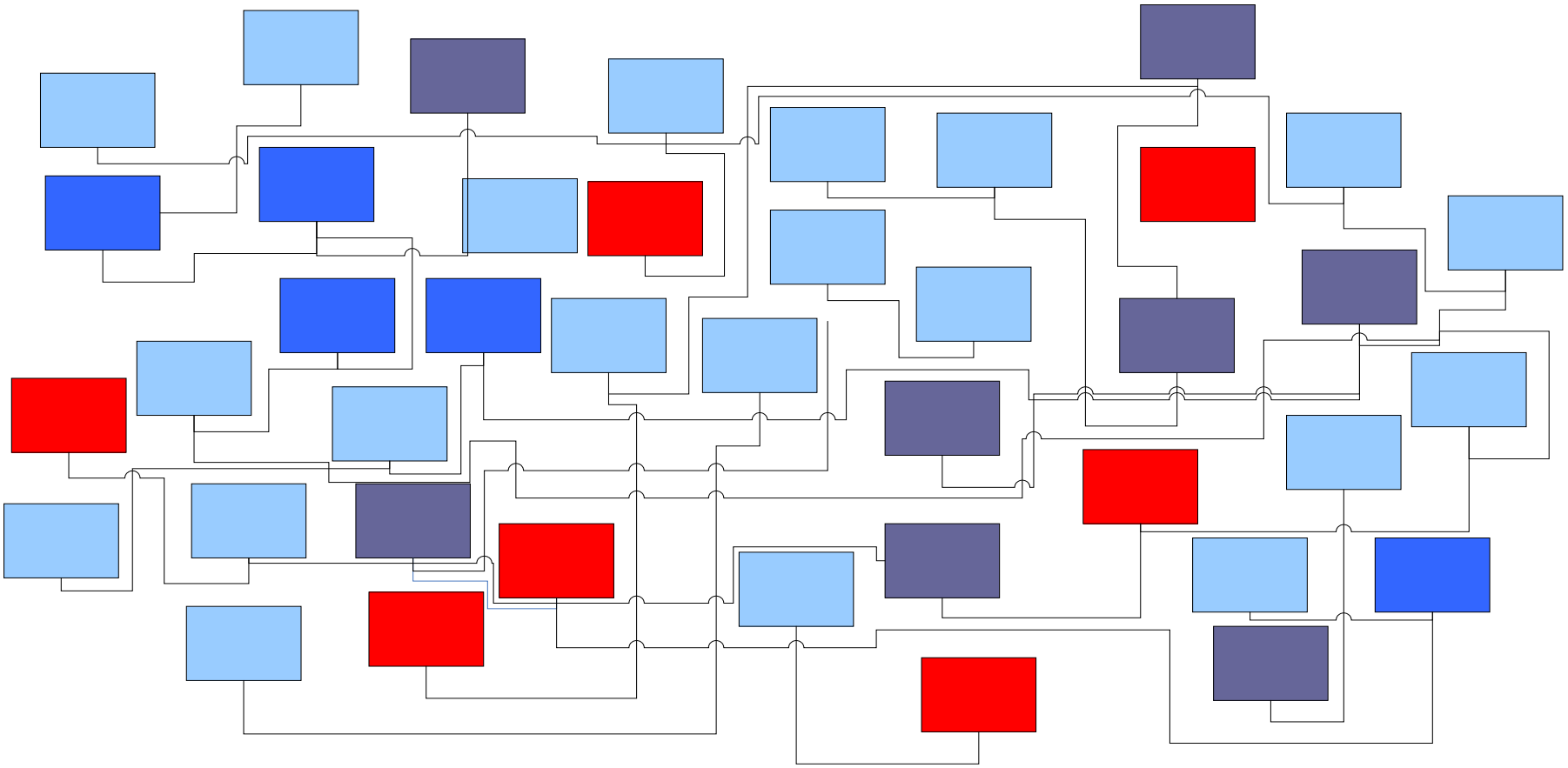
- Component: service provider, service requester and service broker
  - Service provider allows access to services, creates a description of a service and publishes it to the service broker
  - Service broker hosts a registry of service descriptions
  - Service requester discovers a service by searching through the service descriptions given by the service broker and binds to it
- Connector: web service protocols

Service-Oriented Architecture (SOA) Style



Example

- Legacy integration



Example

- SOA integration