

Software architecture patterns

Distributed software systems
CdLM Informatica - Università di Bologna

Agenda

Layered architecture

Event-driven architecture

Microkernel architecture

Microservice architecture

Cloud-based architecture

Source: Richards, *Sw architecture patterns*, O'Reilly 2015

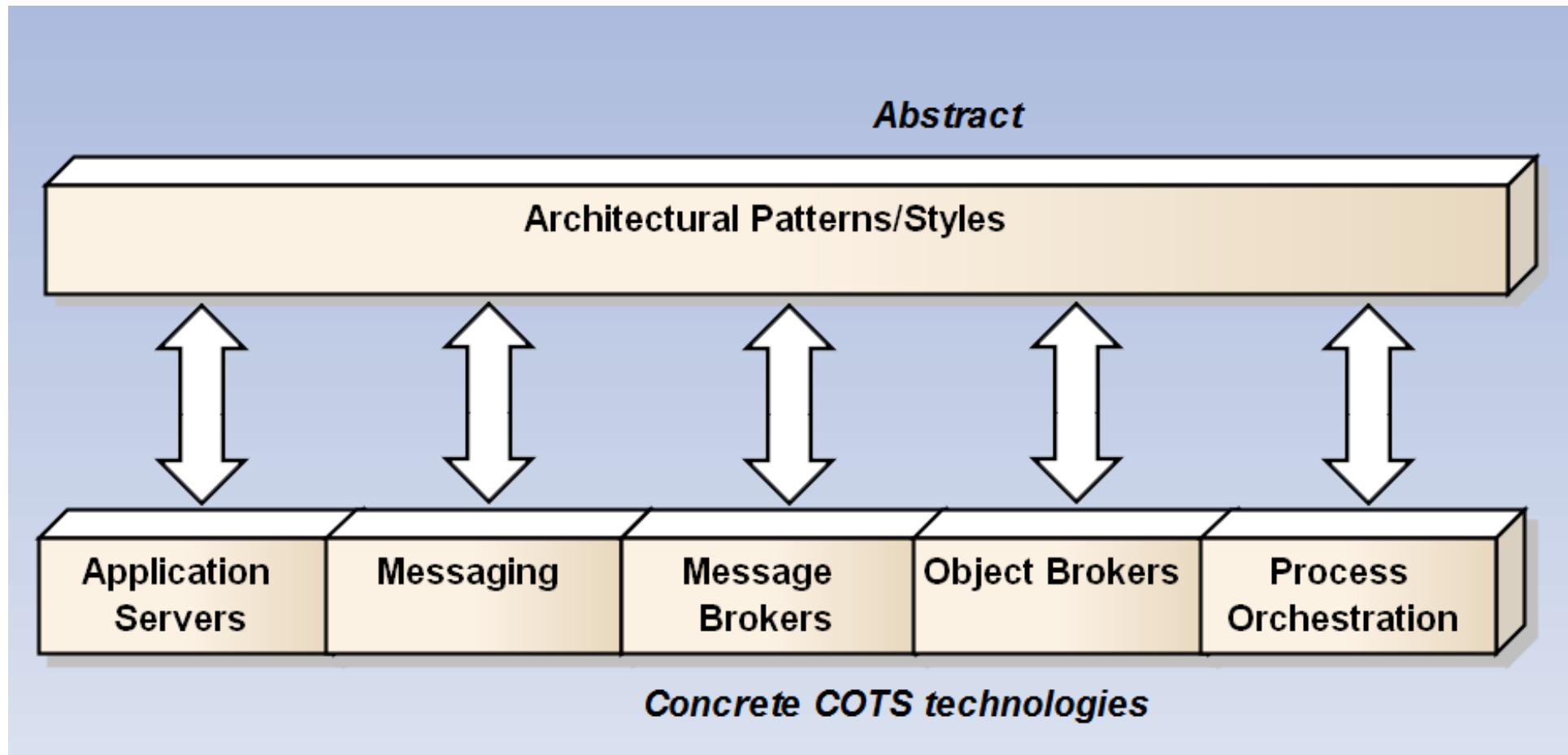
Motivation

Some basic questions about software system deployment and maintenance are hard to answer:

- Does the system scale?
- How responsive is the system?
- What are its performance metrics ?
- How easily does the system respond to change?
- What are its deployment issues?

Some concepts useful to answer to these questions are the «software architecture patterns»

Patterns and technologies

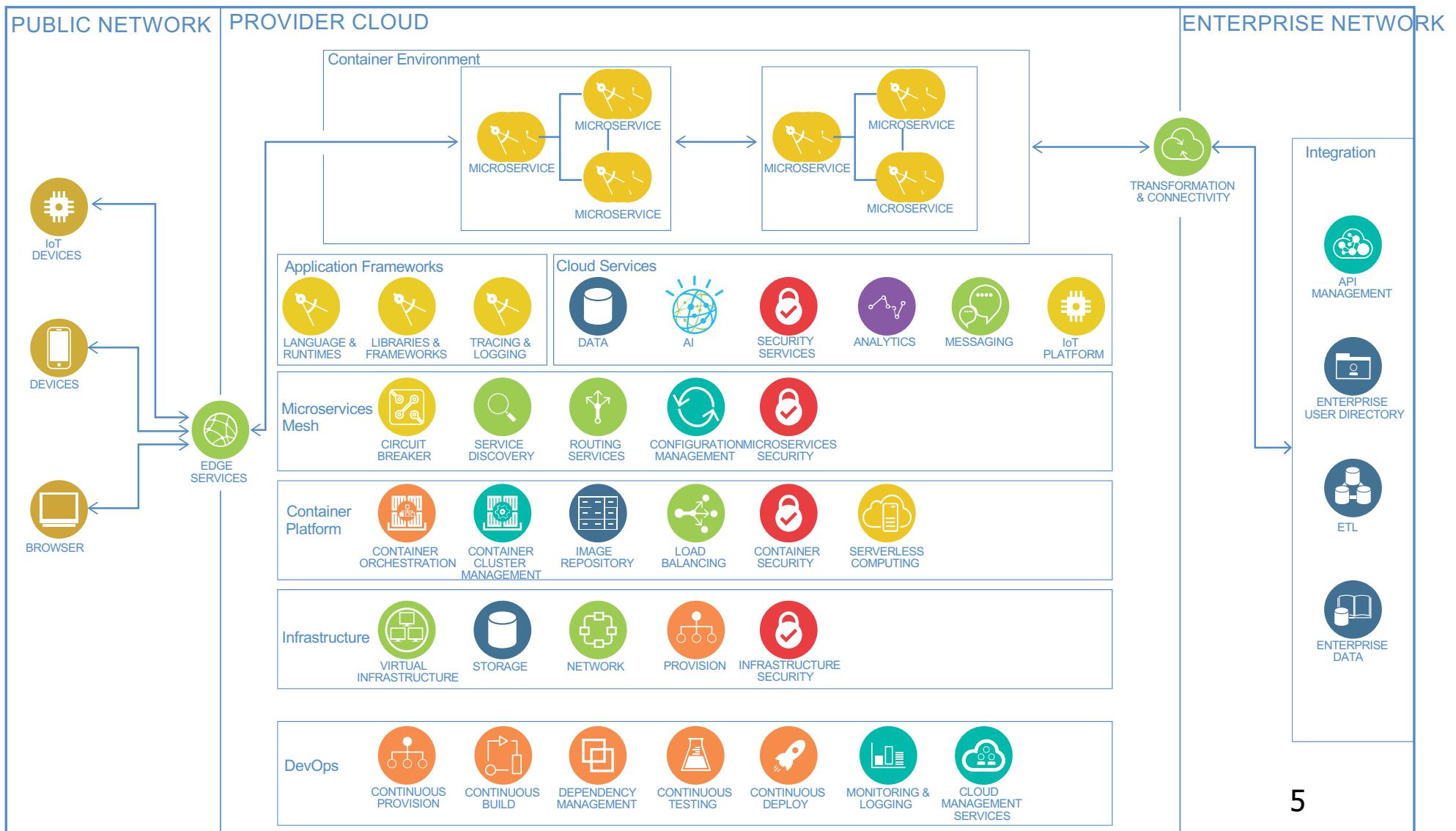


Technologies change all the time, while patterns endure!

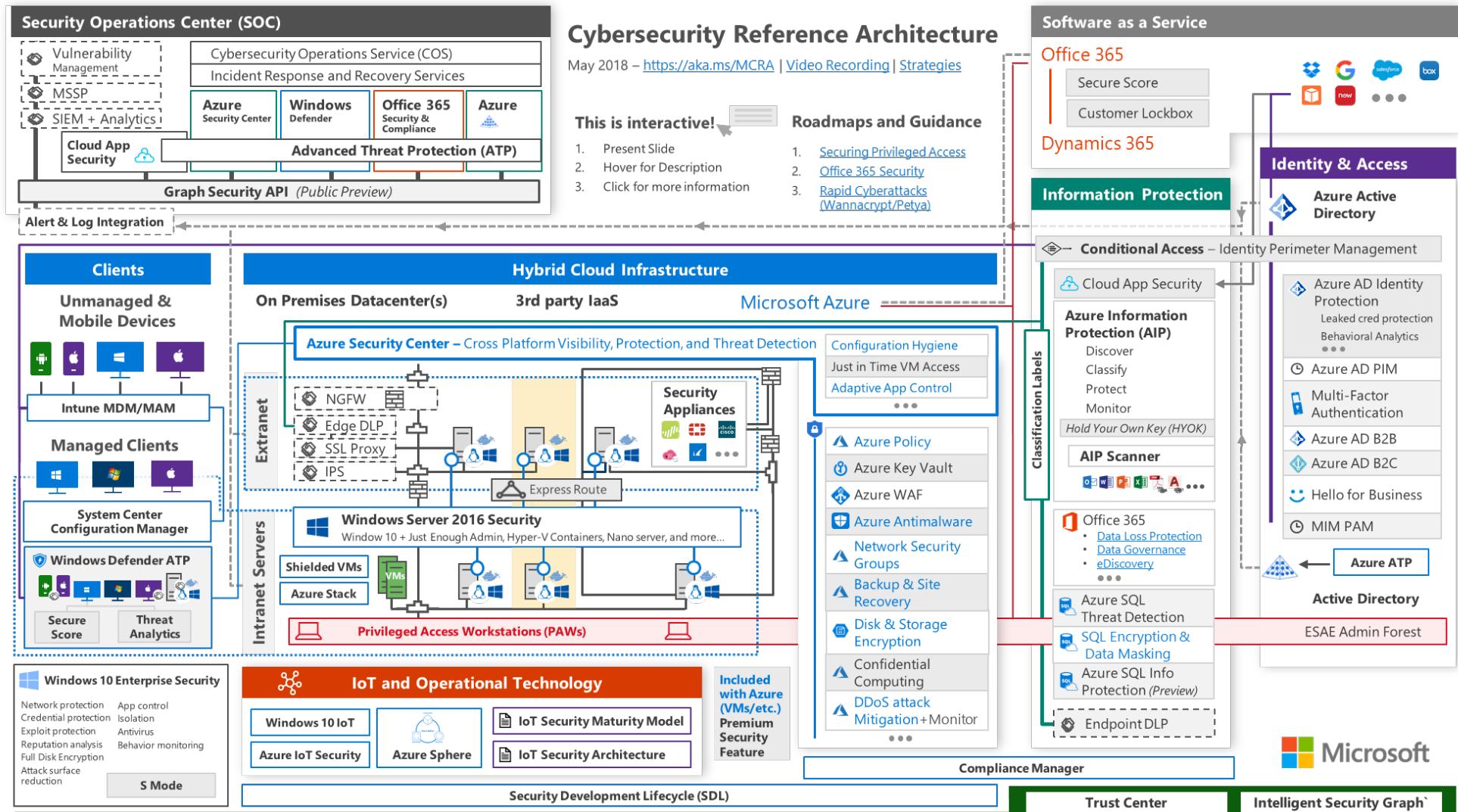
Reference architectures and patterns

A Reference Architecture is a set of **patterns**, partially or completely instantiated

<https://www.ibm.com/cloud/architecture/architectures/microservices/reference-architecture/>

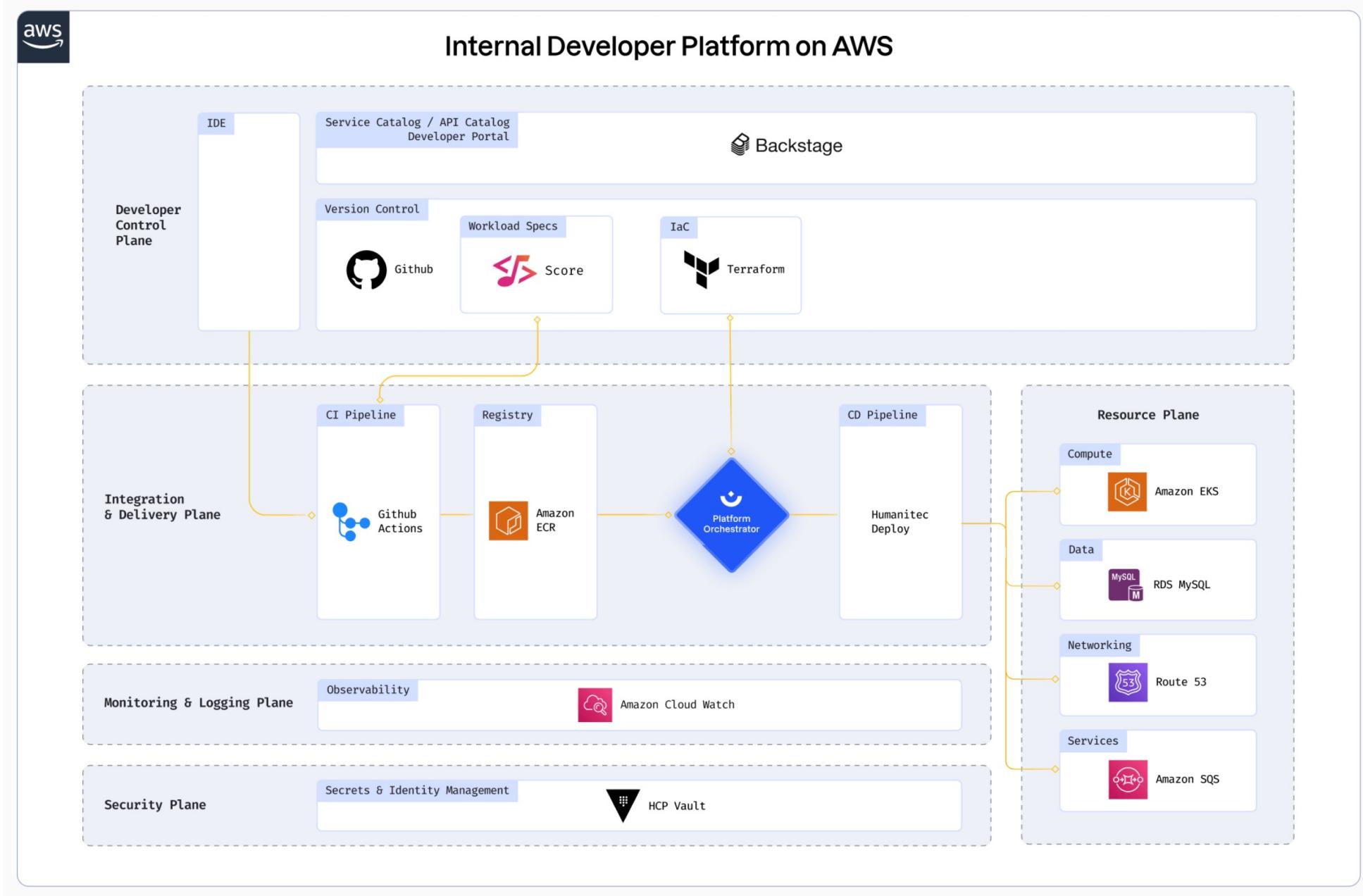


Example: Microsoft Cybersecurity Reference Architecture



<https://www.microsoft.com/security/blog/2018/06/06/cybersecurity-reference-architecture-security-for-a-hybrid-enterprise/>

Example: AWS platform for internal developers



Patterns: Layers and tiers (1)

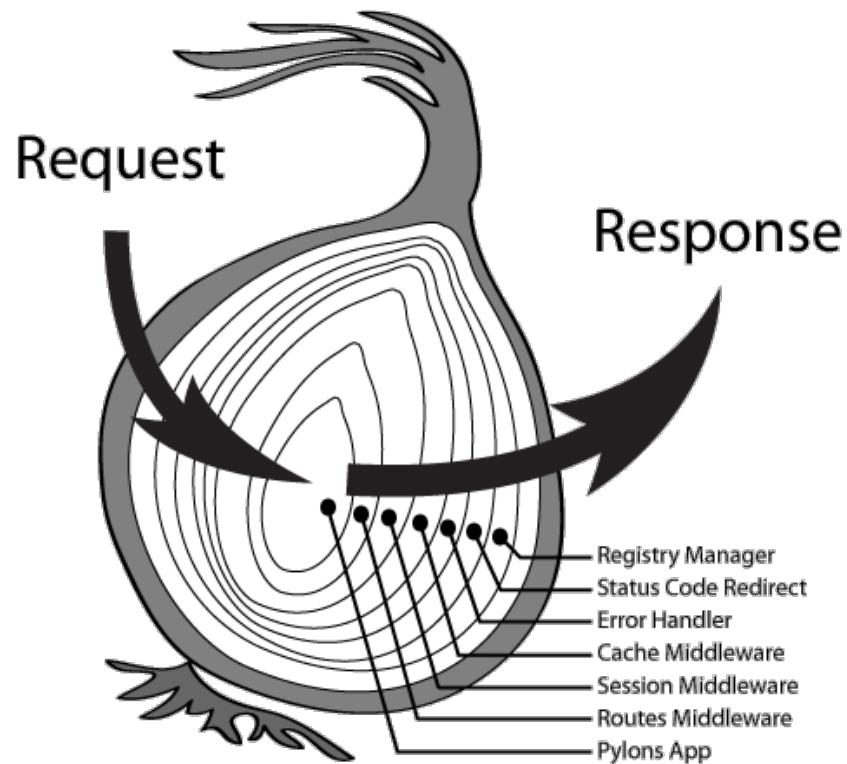
The architecture of a system is generally achieved by decomposing it into subsystems, following a ***layered*** and/or a **partition** based approach

These are orthogonal approaches:

A **layer** is a logical structuring mechanism for the elements that make up a software solution (e.g., a kernel is a layer)

A **partition** (or **tier**) is a physical structuring mechanism for the system infrastructure (e.g., a user interface is a tier)

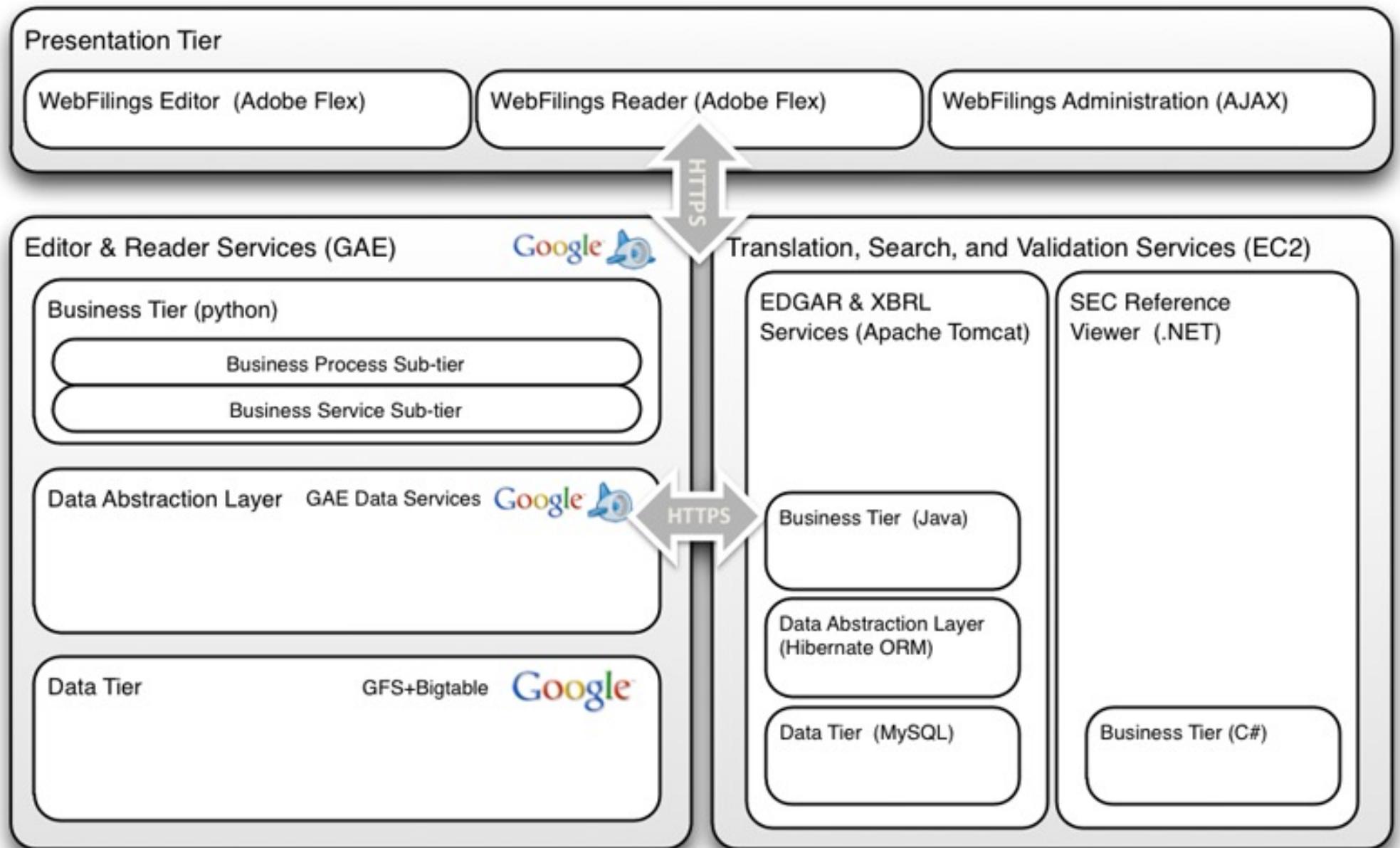
Architectural vegetables



Onion: layers



Garlic: partitions



Layers and tiers (2)

A complete decomposition of a given system comes from both layering and partitioning:

First, the system is divided into top level subsystems which are responsible for certain functionalities (*partitioning*);

Then, if necessary, each subsystem is organized into several layers, up to the definition of *simple enough* layers

Layered approach

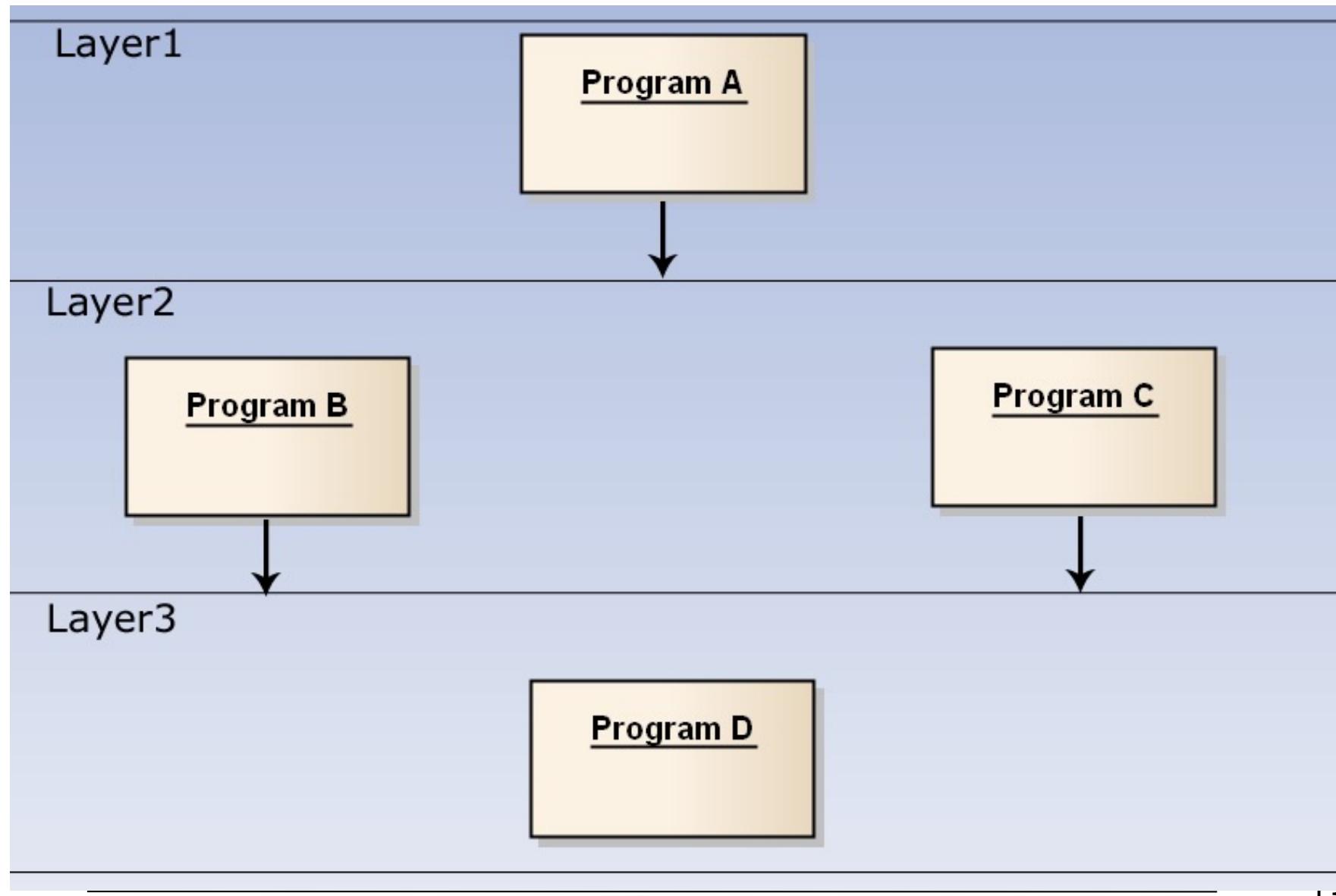
An architecture which has a hierarchical structure, consisting of an ordered set of layers, is *layered*

A **layer** is a subsystem able to provide some related services, that can be realized by exploiting services from other layers

A layer depends only from its lower layers (i.e., layers which are located at a lower level into the architecture)

A layer is only aware of lower layers

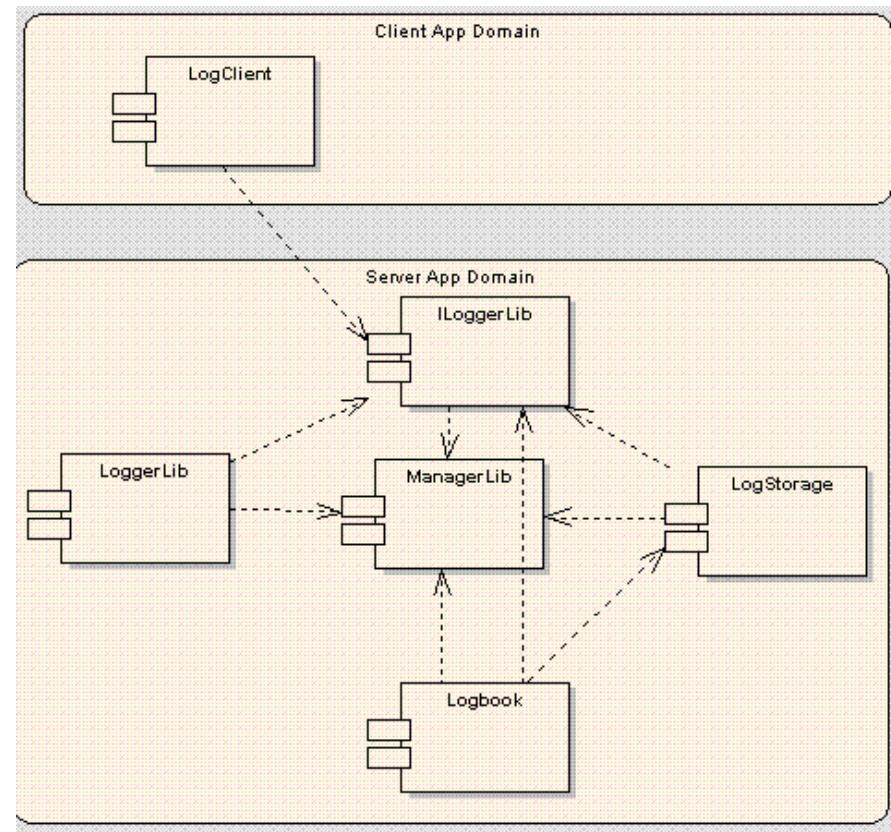
Example: layers of Virtual Machines



Layers: component diagram

Connectors for layered systems are often procedure calls

Each level implements a different virtual machine with a specific input language



Closed vs. open layers

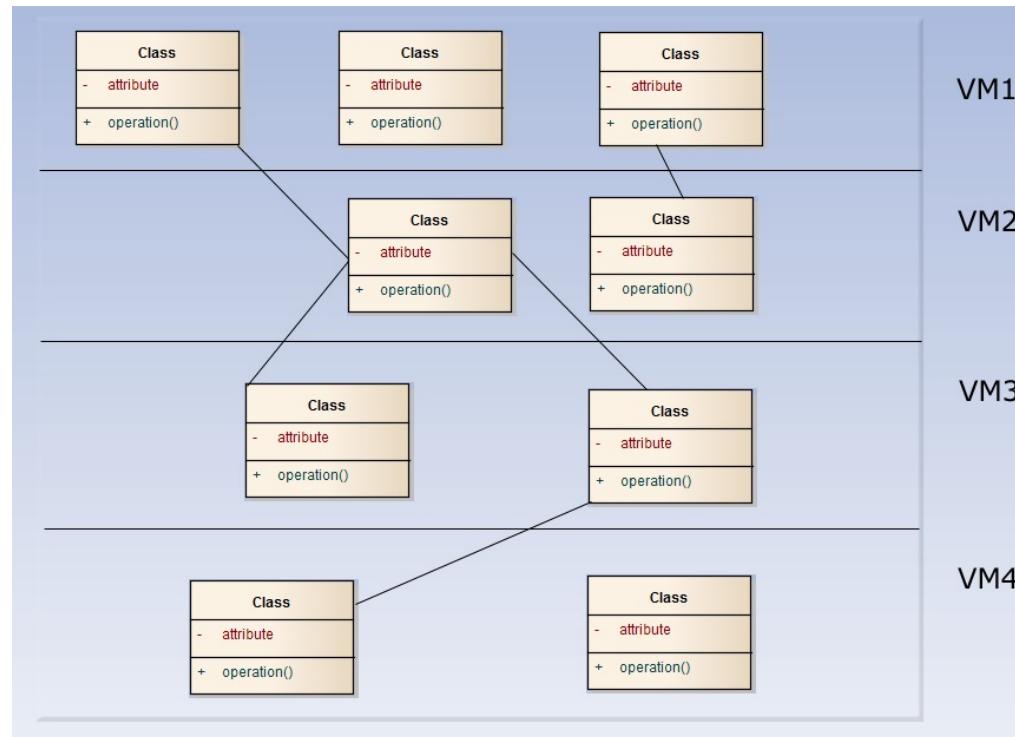
Closed Layers Architecture (CLA): the i -th layer can only have access to the layer $(i-1)$ -th

Open Layers Architecture (OLA): the i -th layer can have access to all the underlying layers (i.e., the layers lower than i)

Closed Layers Architecture

The i -th layer can only invoke the services and the operations which are provided by the layer $(i-1)$ -th

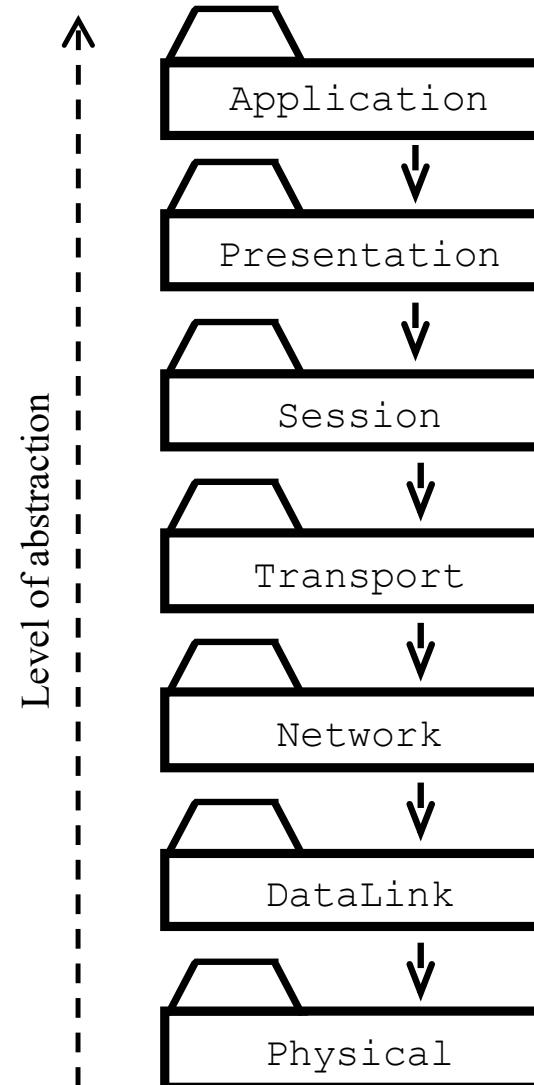
The main goals are the system maintainability and high portability (eg. Virtualization: each layer i defines a Virtual Machine - VM_i)



Example of CLA: ISO/OSI stack

The ISO/OSI reference model defines 7 network layers, characterized by an increasing level of abstraction

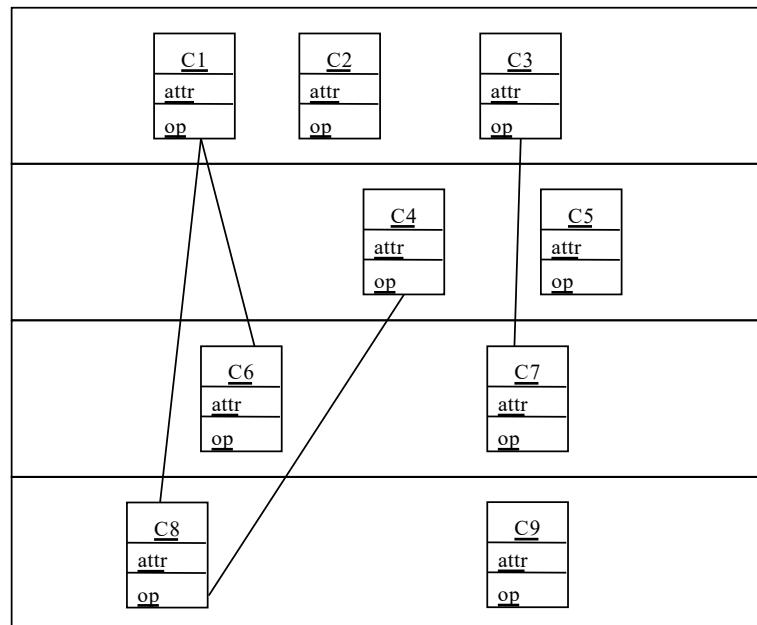
Eg: the Internet Protocol (IP) defines a VM at the network level able to identify net hosts and transmit single packets from host to host



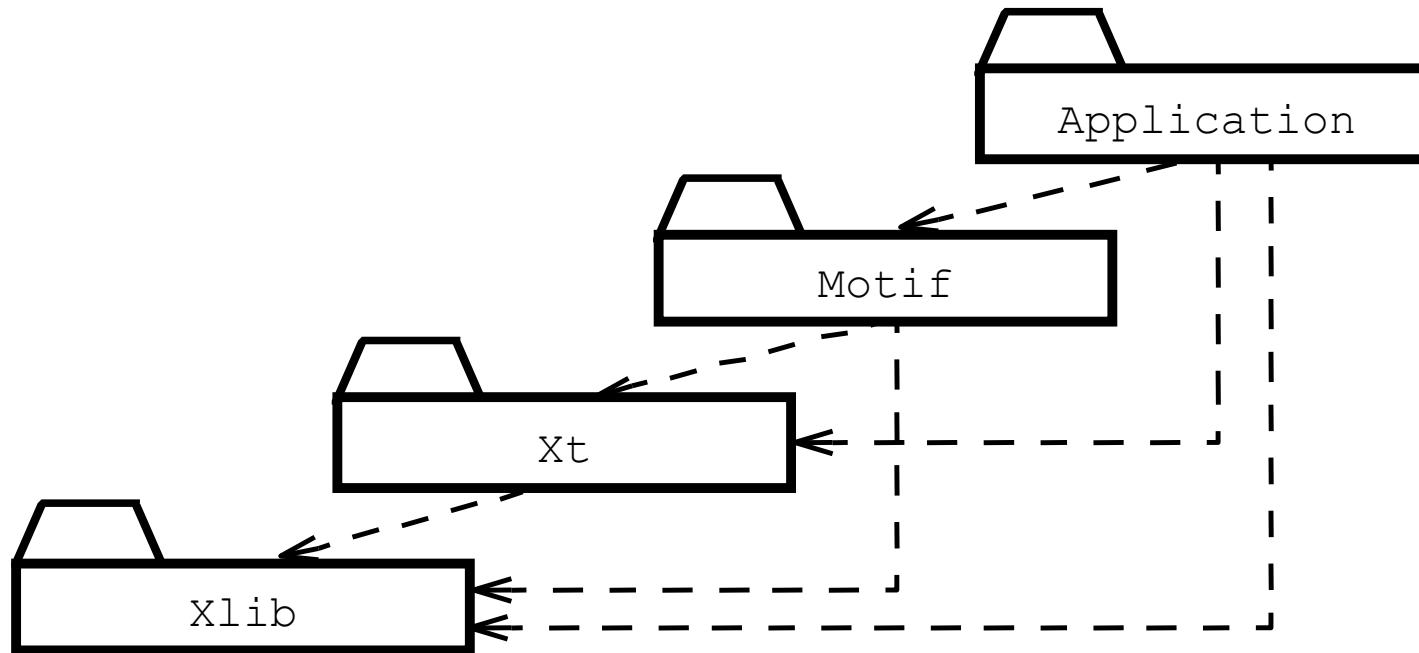
Open Layers Architecture (OLA)

The i -th layer can invoke the services and the operations which are provided by all the lower layers (the layers lower than i)

The main goals are the execution time and efficiency



Example of OLA: OSF/Motif



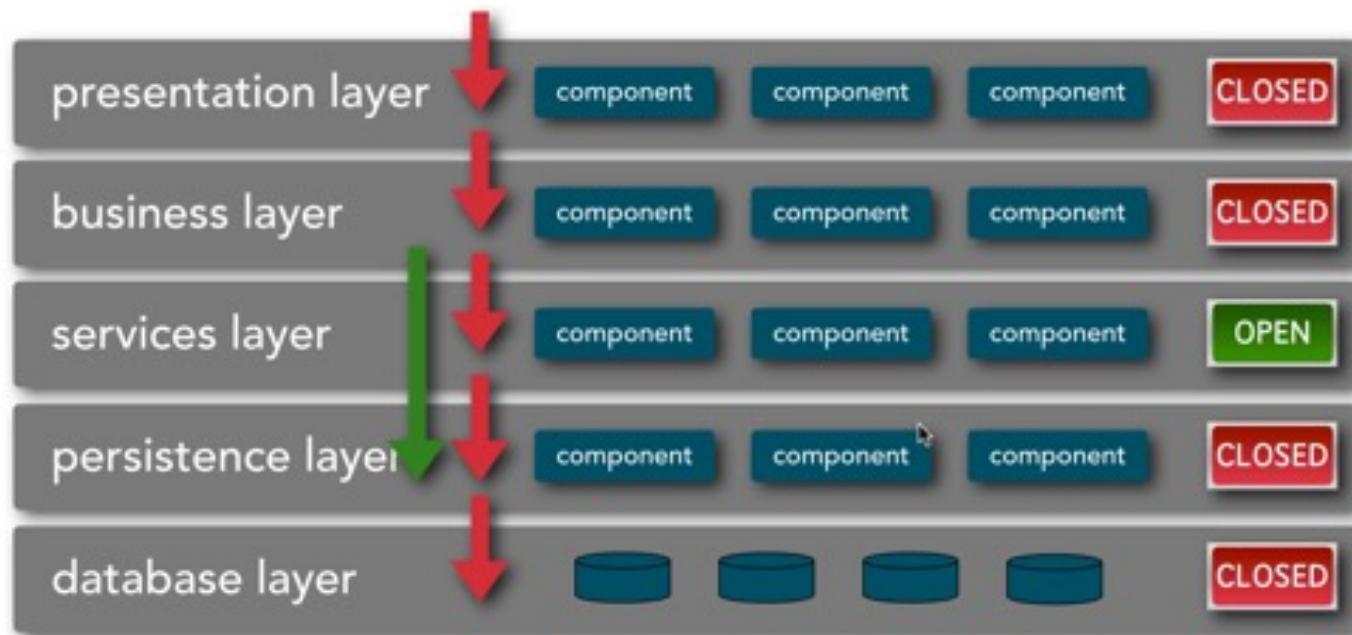
Xlib: provides low-level drawing facilities;

Xt: provides basic user interface widget management;

Motif: provides a large number of sophisticated widgets;

Application: can access each layer independently.

Mixed open-closed layers



<https://www.oreilly.com/ideas/contrasting-architecture-patterns-with-design-patterns>

Creating layers: Façade (1)

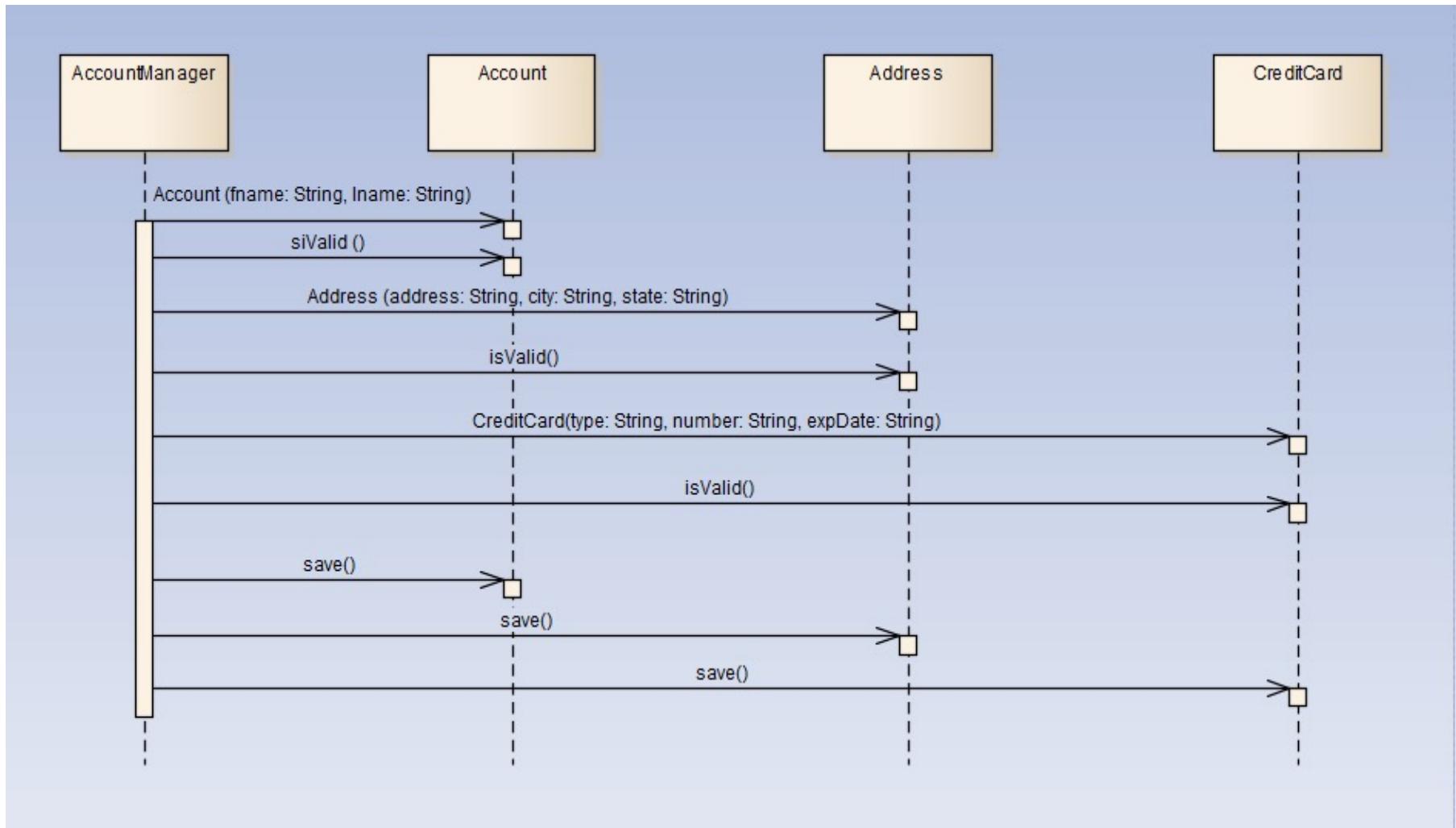


Figure 22.5 How a Client Would Normally Interact (Directly) with Subsystem Classes to Validate and Save the Customer Data

Creating layers: Façade (2)

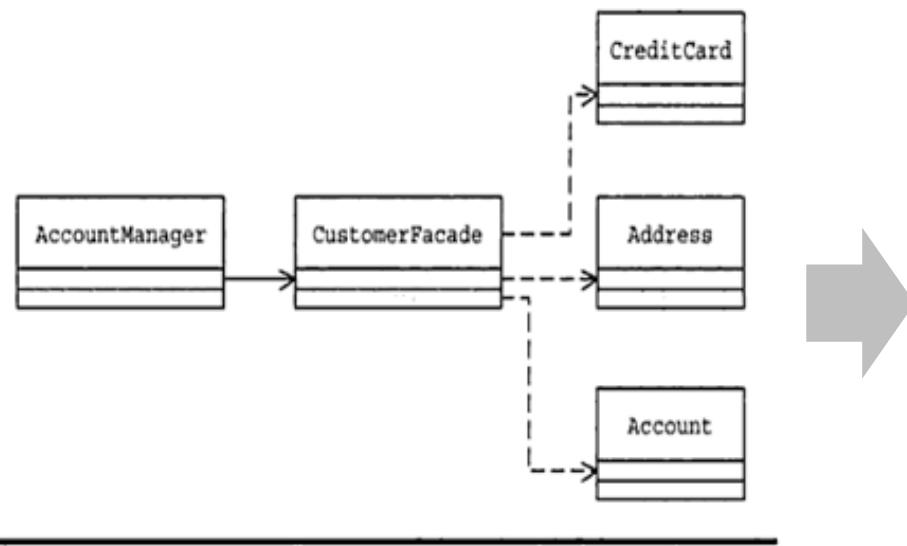


Figure 22.7 Class Association with the Façade Class in Place

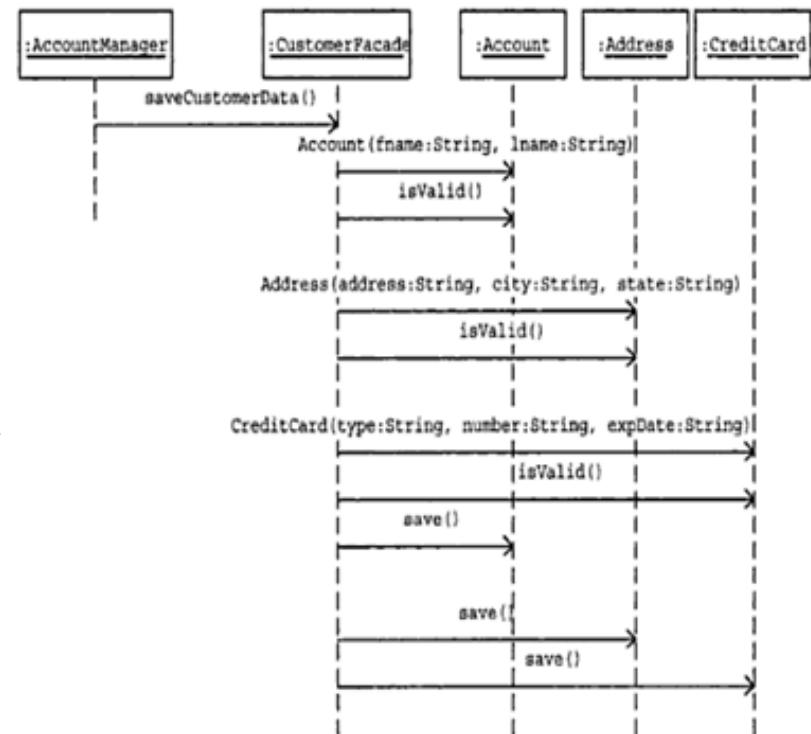


Figure 22.8 In the Revised Design, Clients Interact with the Façade Instance to Interface with the Subsystem

Tiered approach

Another approach consists of *partitioning* a system into independent sub-systems (*tiers*), which are responsible for a class of services maintained as independent modules

These modules are separated from each other by physical boundaries:

machine boundaries

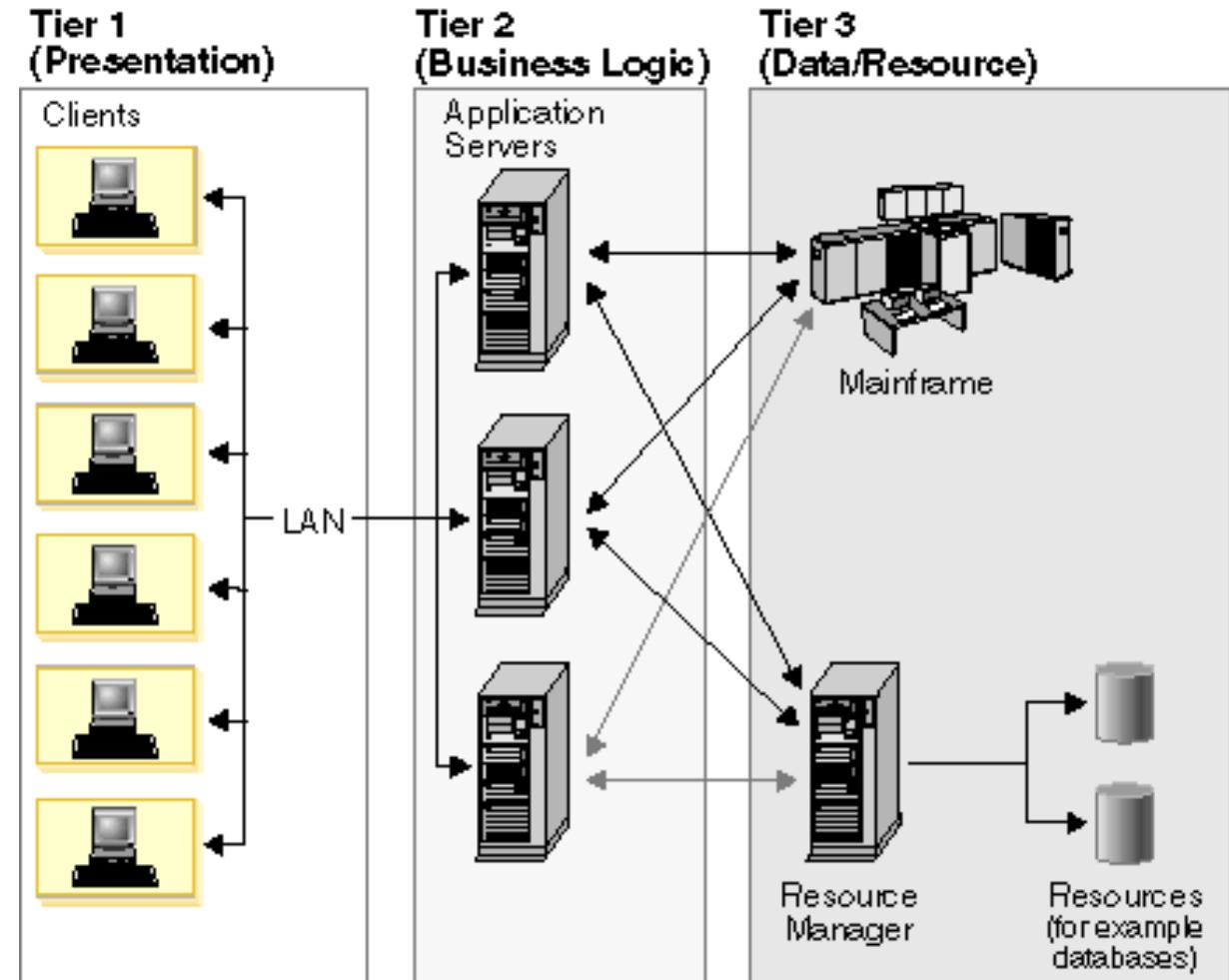
process boundaries

corporate boundaries

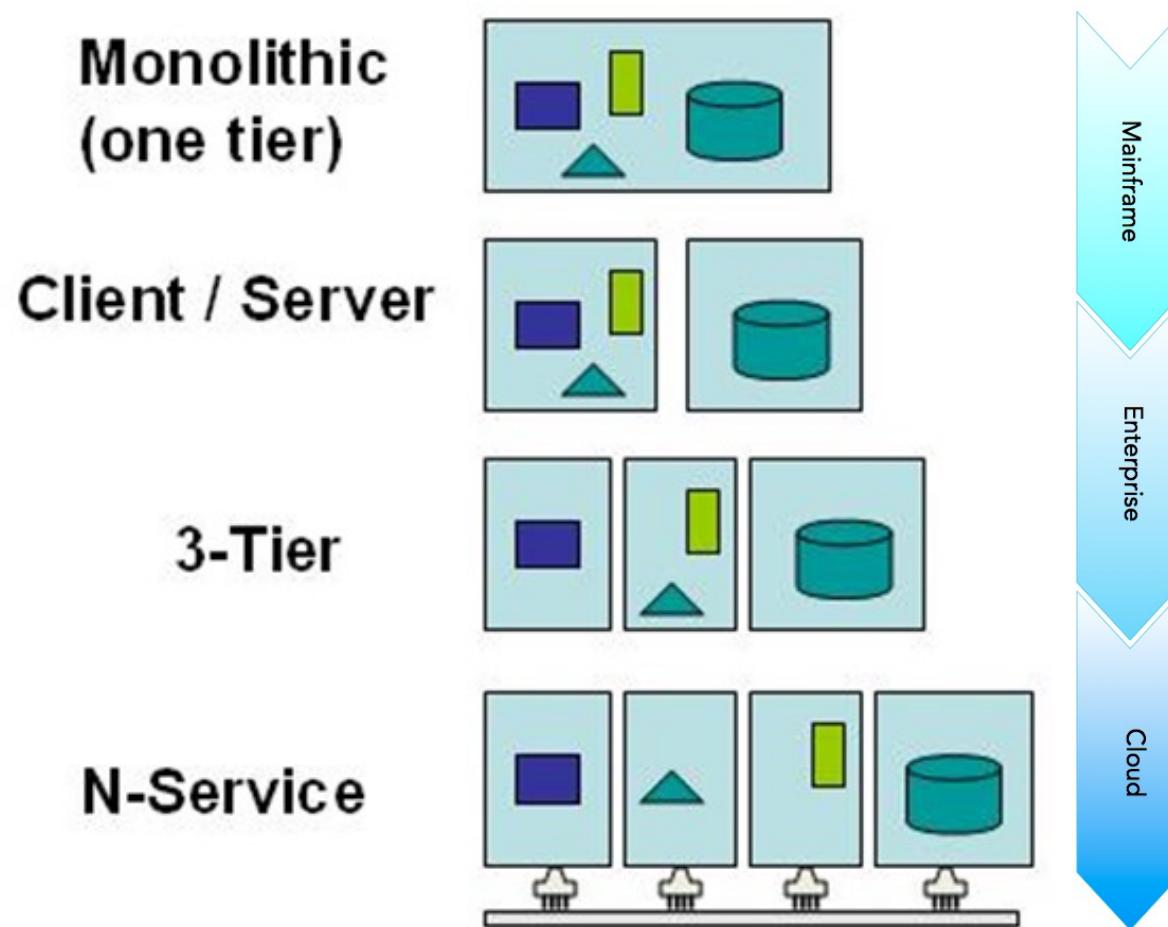
Example: tiered system

A common design of client/server systems uses three tiers:

1. A client that interacts with the user
2. An application server that contains the business logic of the application
3. A resource manager that stores data



Tiers and computing evolution



1-tier, 2-tiers, 3-tiers

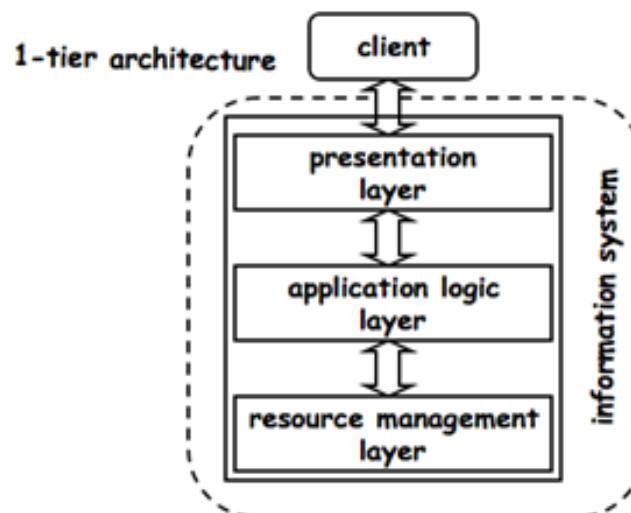
- A **1-tier model (monolithic)** describes a single-tiered application in which the user interface and data access code are combined into a single program from a single platform
- A **2-tiers model (client/server)** represents a split monolithic model composed by a *client* tier that interacts directly with a *server* tier
- A **3-tiers model (n-tiers)** is a client/server model in which the presentation, the application processing, and the data management are logically (and often physically) separate processes

1-tier model: Monolithic

The presentation, application logic and resource manager are built as a monolithic entity (no modularity)

Users/programs access the system through “dumb” terminals managed by an *information system*

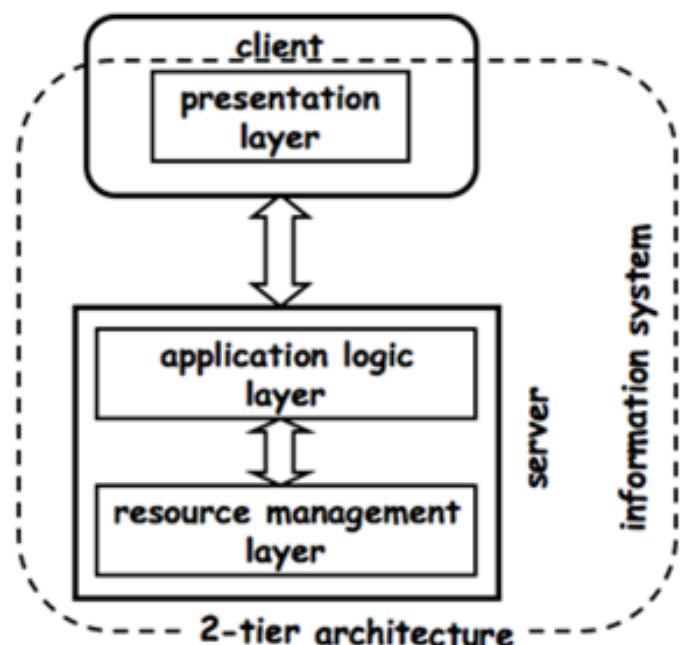
Typical model for mainframes



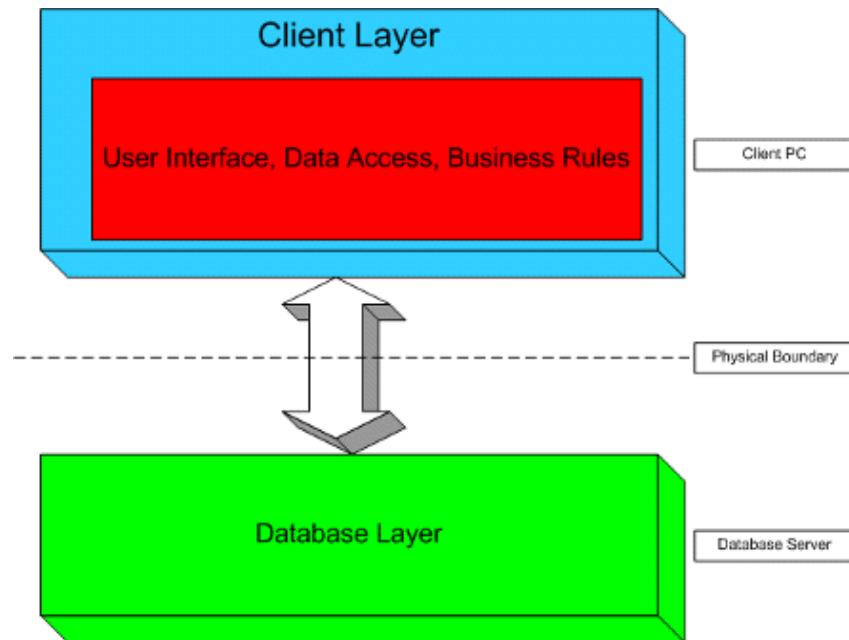
2-tiers model: Client/Server

This model allows to move the presentation tier to the client (*fat client*)

It introduces the concept of API, an interface to invoke the system from the outside



[Copyright © 2004 Springer-Verlag Berlin Heidelberg](#)



[Copyrigth © 2012 Microsoft Corporation](#)

3-tiers model: Multitiered (1)

In this approach, the user interface runs on a desktop PC or workstation and uses a standard graphical user interface (*thin client*)

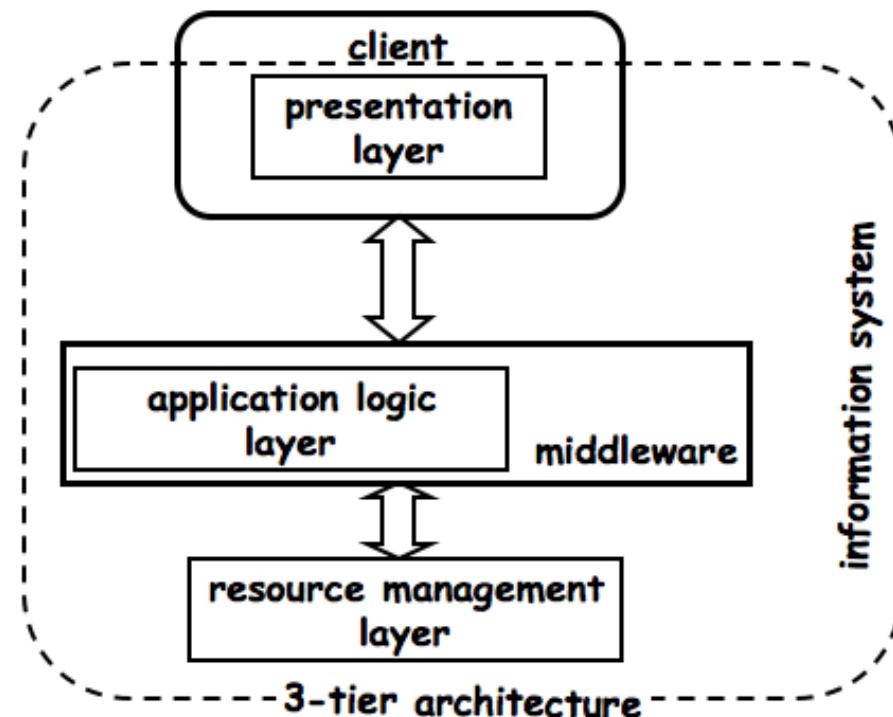
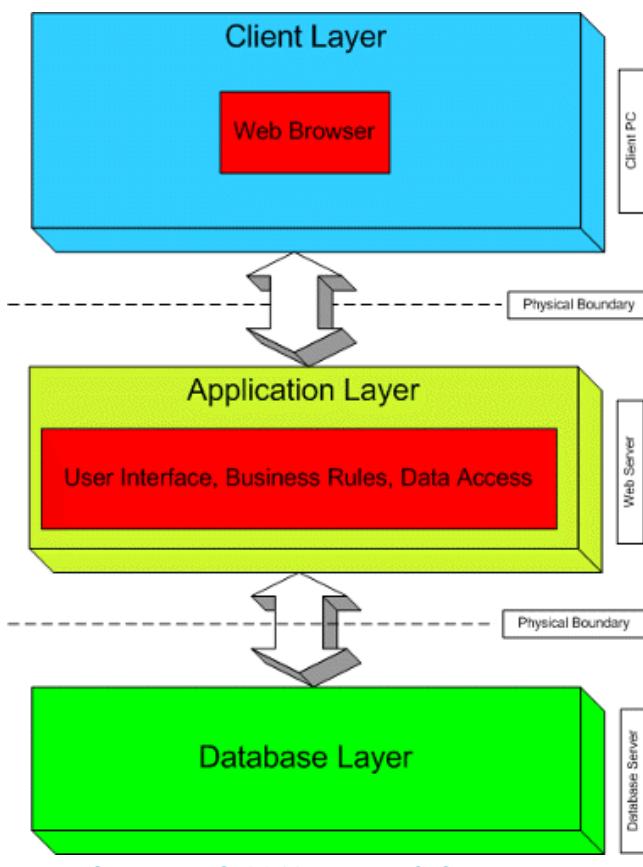
The functional process logic may consist of one or more separate modules running on a workstation or application server

An RDBMS on a database server or mainframe contains the computer data storage logic

This is a specialization (the most widespread) of the n-tiers model

3-tiers model: Multitiered (2)

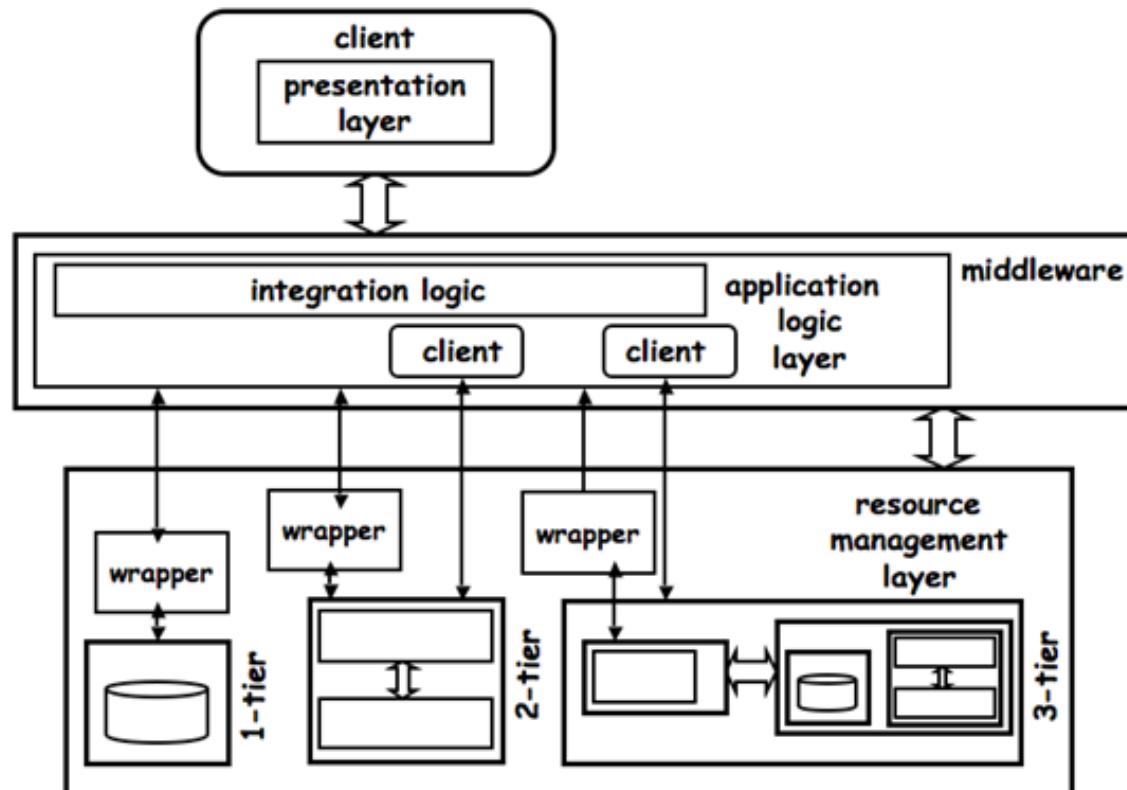
Concept of *middleware*, that introduces an additional tier of business logic encompassing all the underlying systems



[Copyright © 2004 Springer-Verlag Berlin Heidelberg](#)

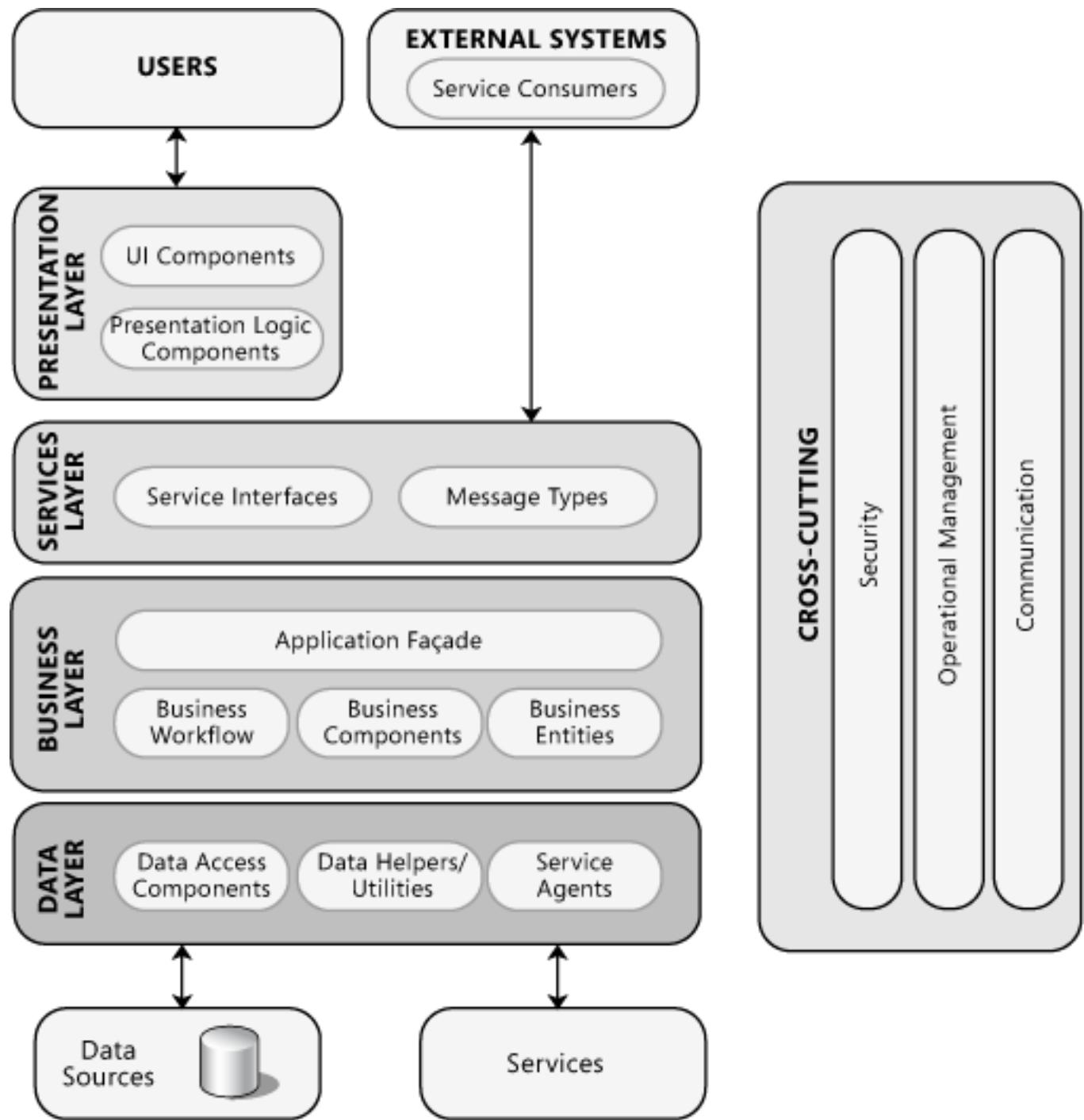
Middleware as a glue

Middleware systems also enable the integration of systems built using other architecture models

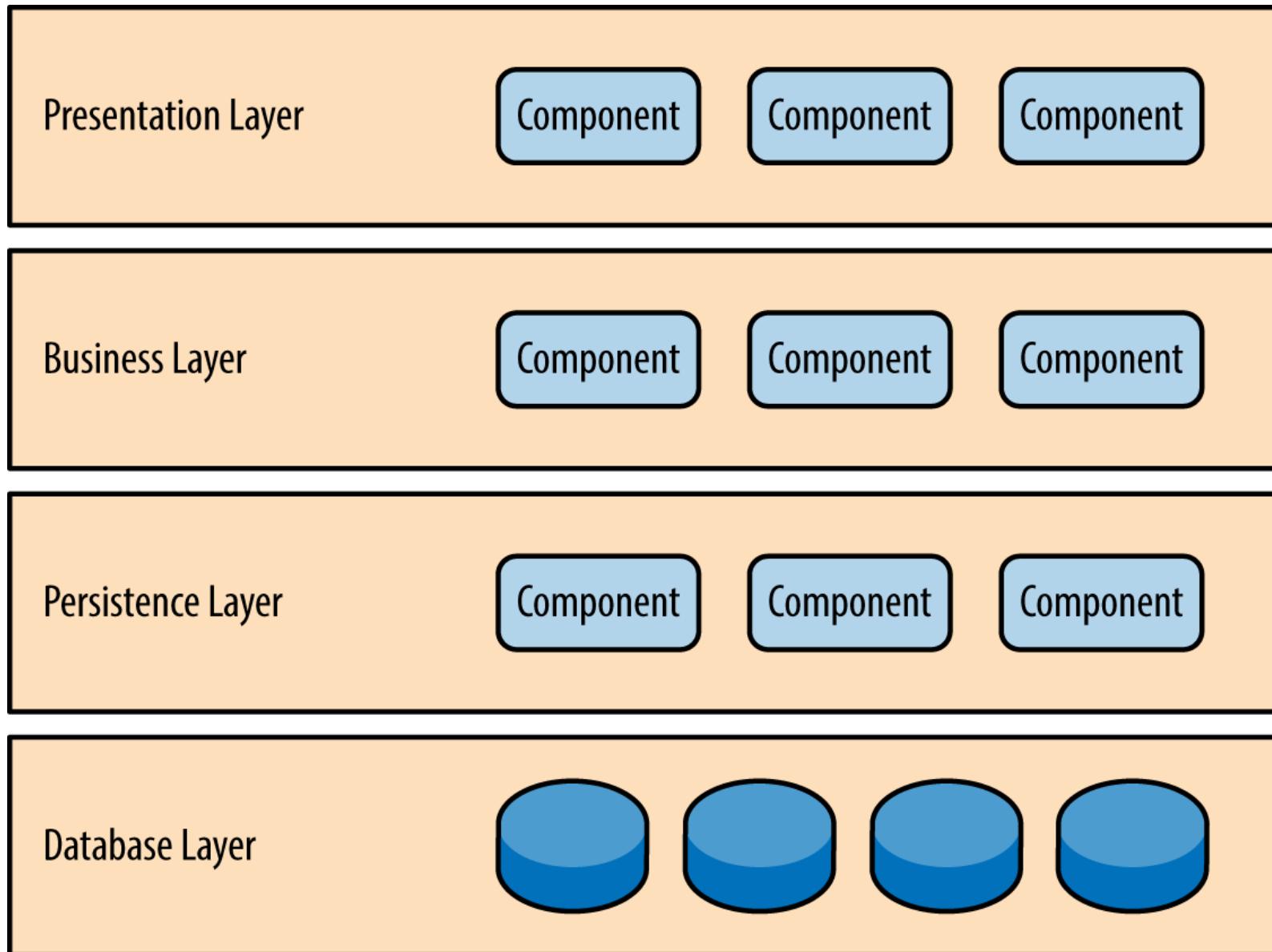


Copyright © 2004 Springer-Verlag Berlin Heidelberg

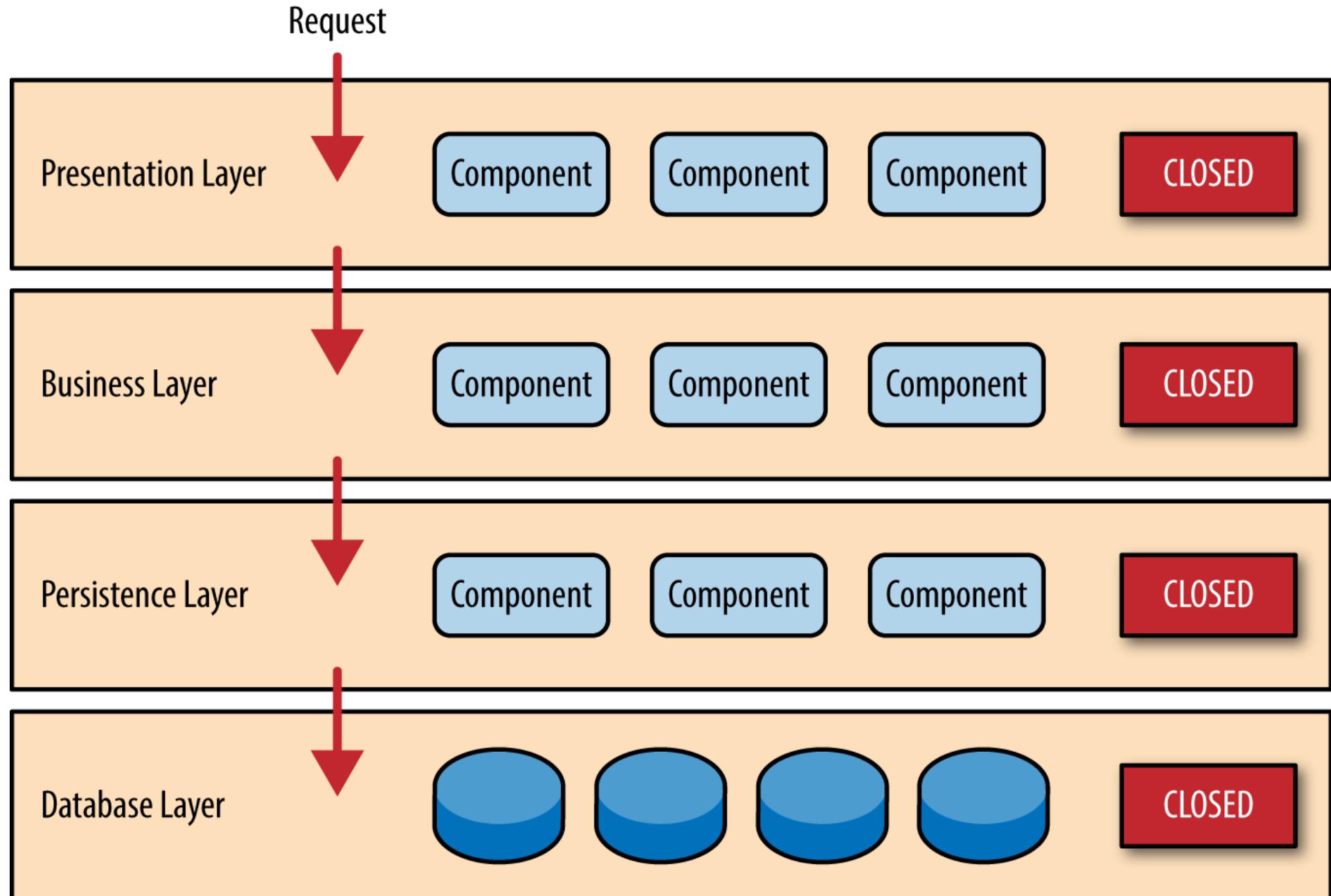
**Do not confuse
layers and tiers!**



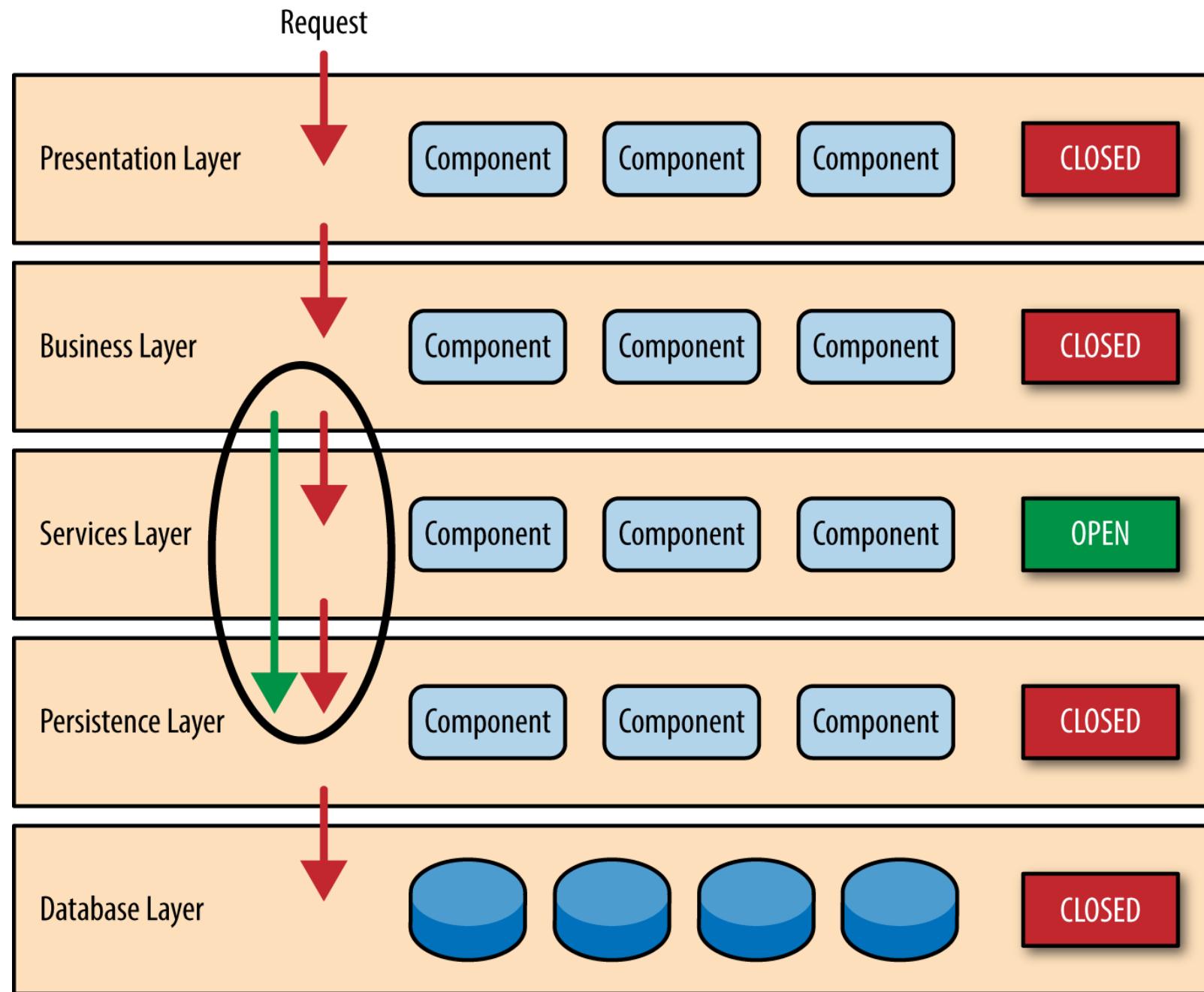
Summary: Layered architecture



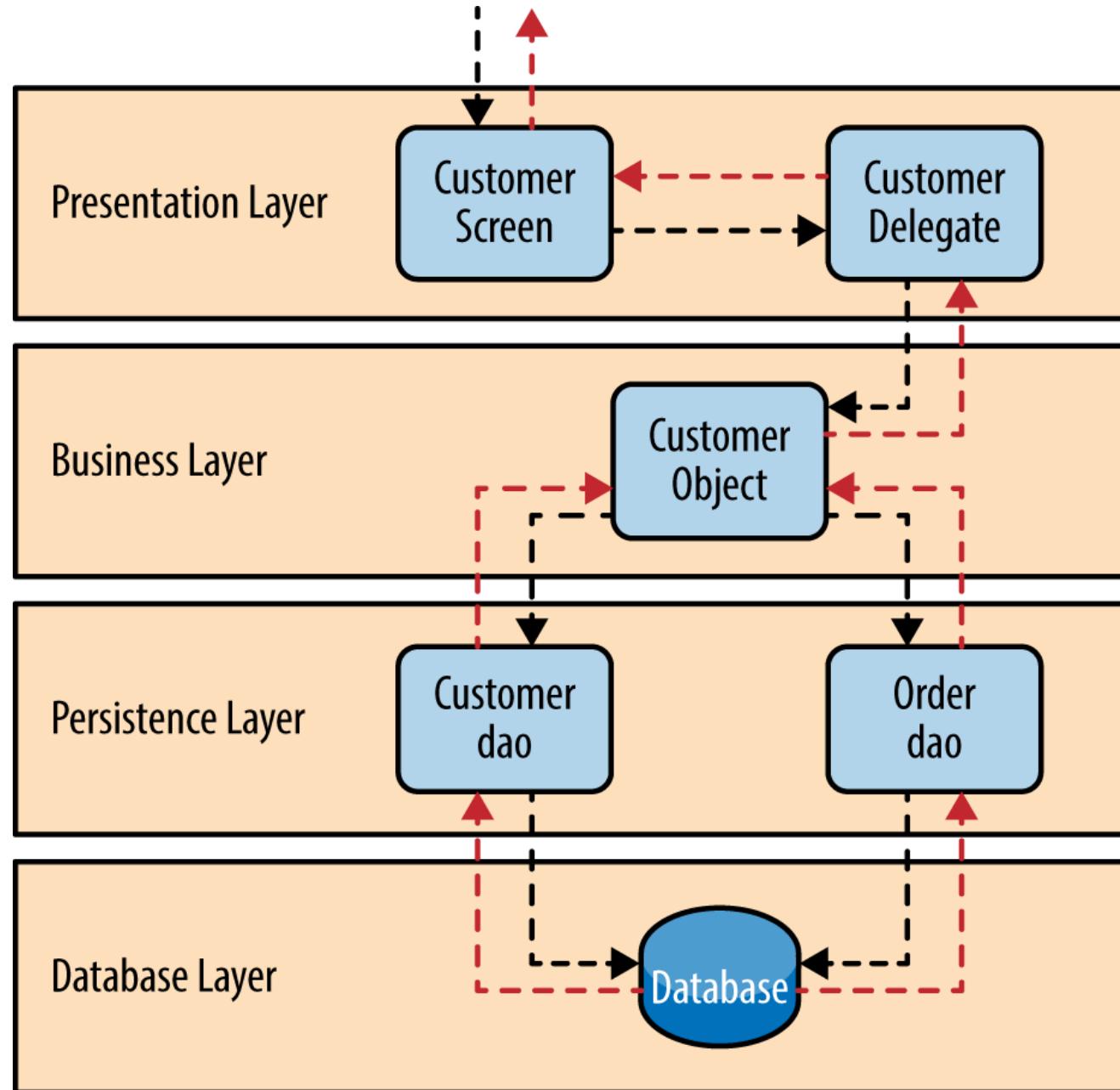
Summary: Closed layers



Summary: Open layers

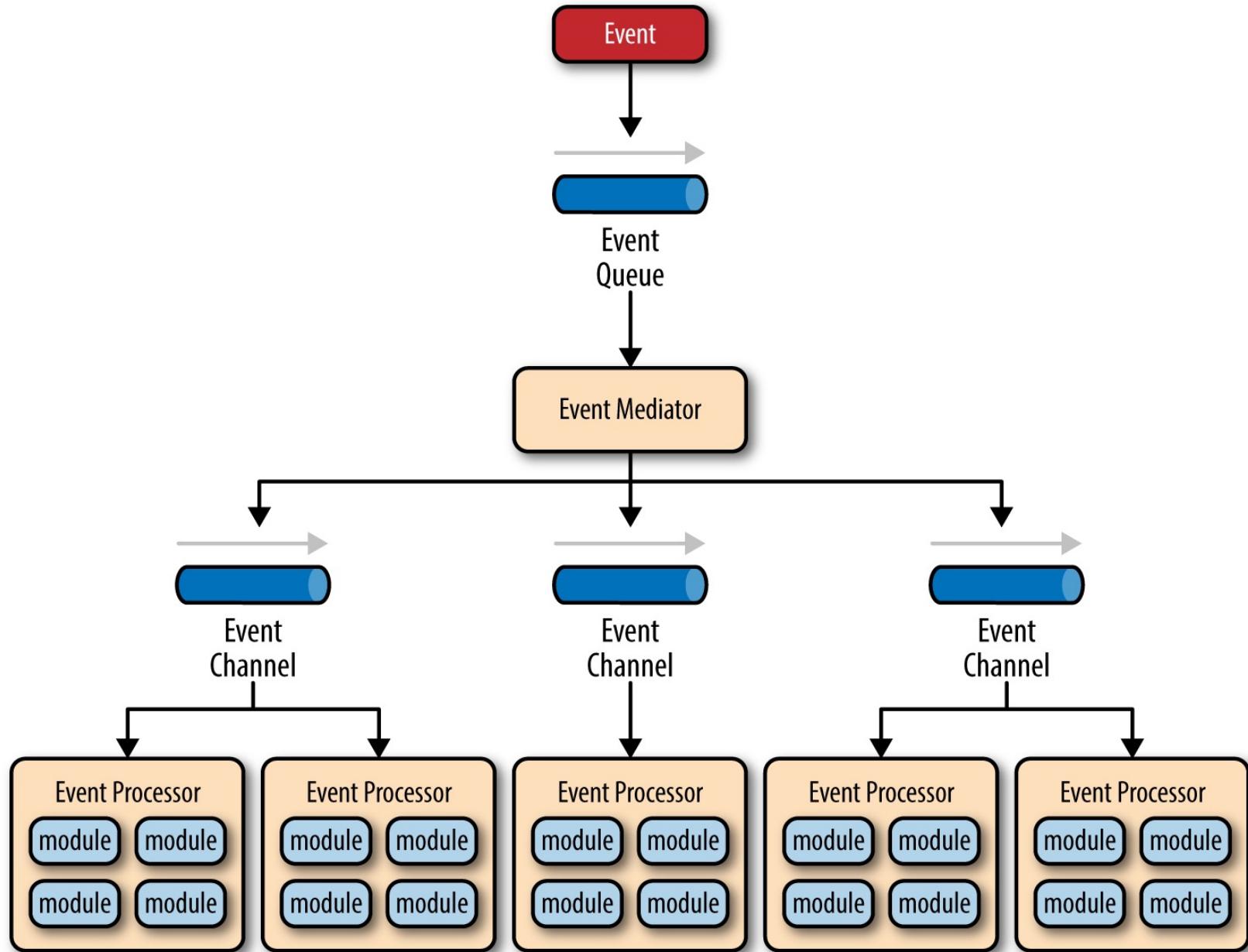


Layered architecture example

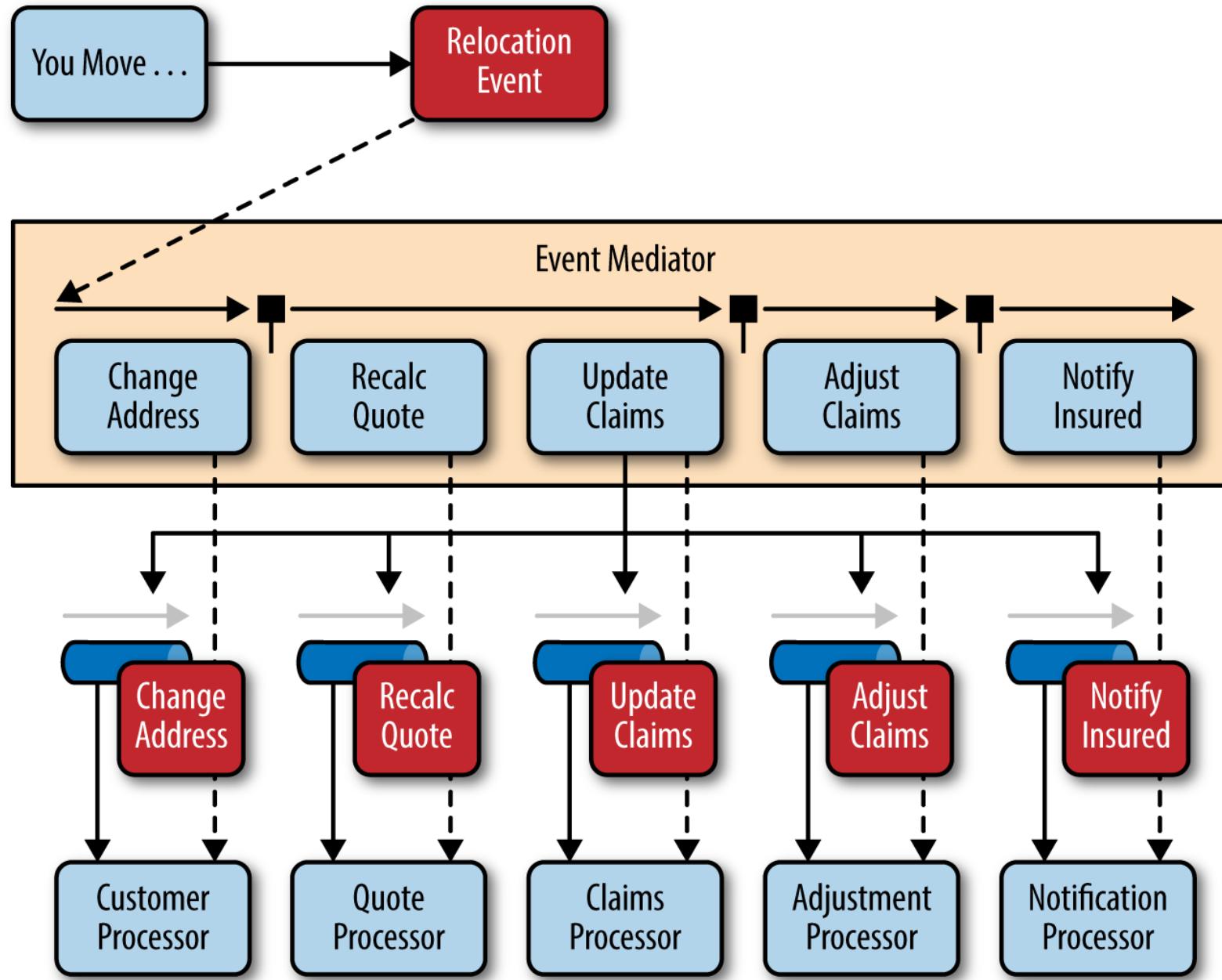


The black arrows show the user request flowing down to the database to retrieve the customer data, and the red arrows show the response flowing back up to the screen to display the data.

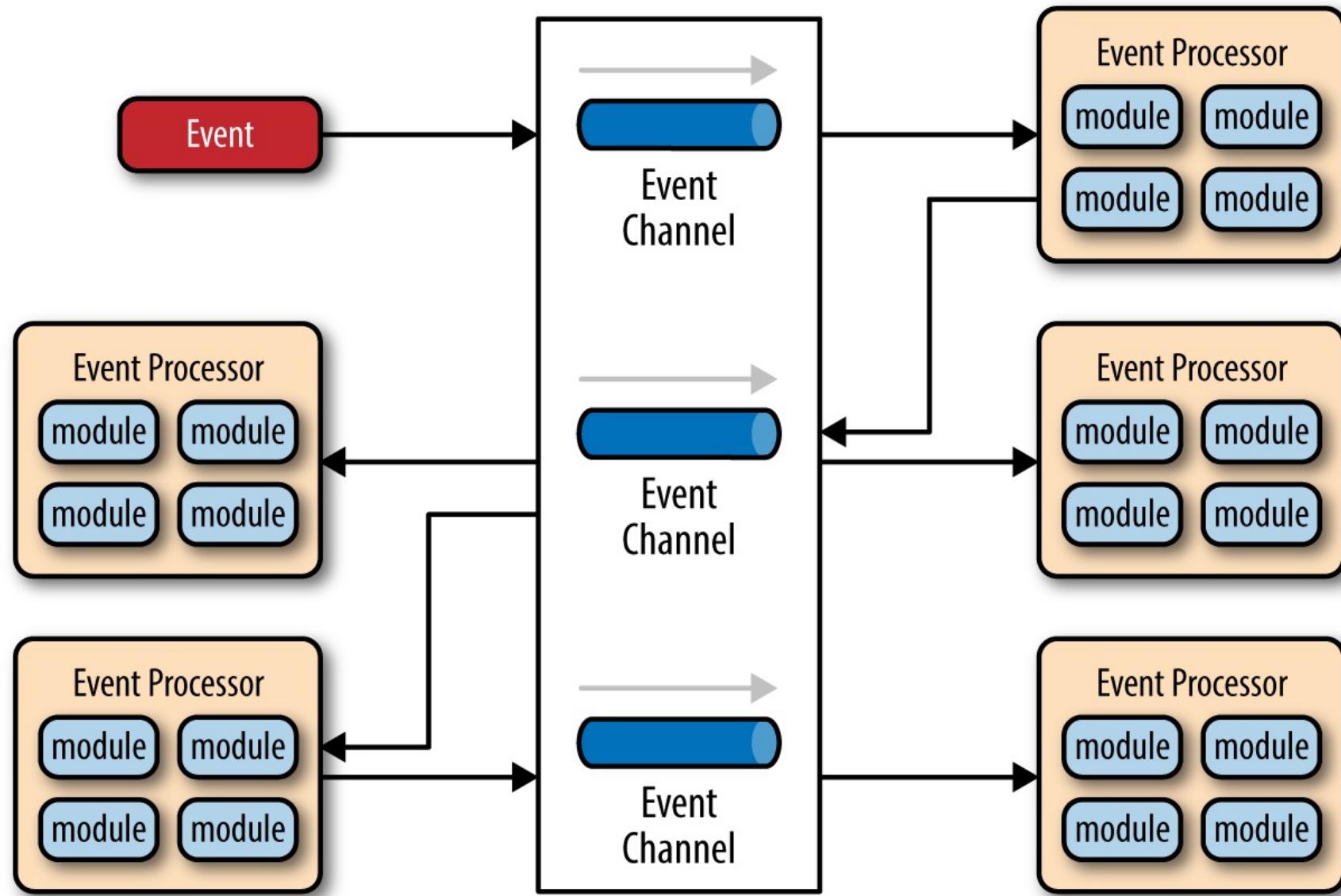
Event driven mediator topology



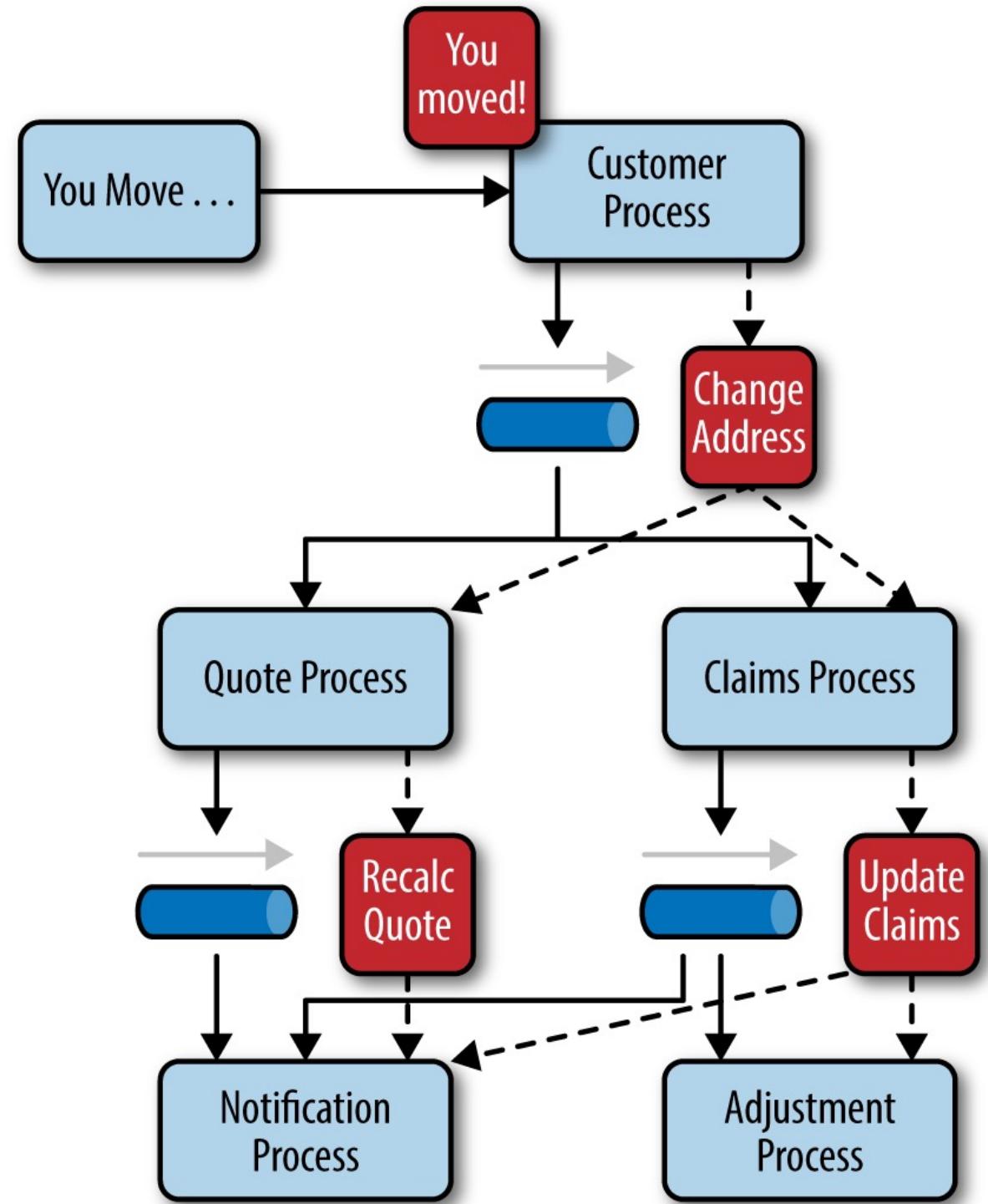
Mediator topology example



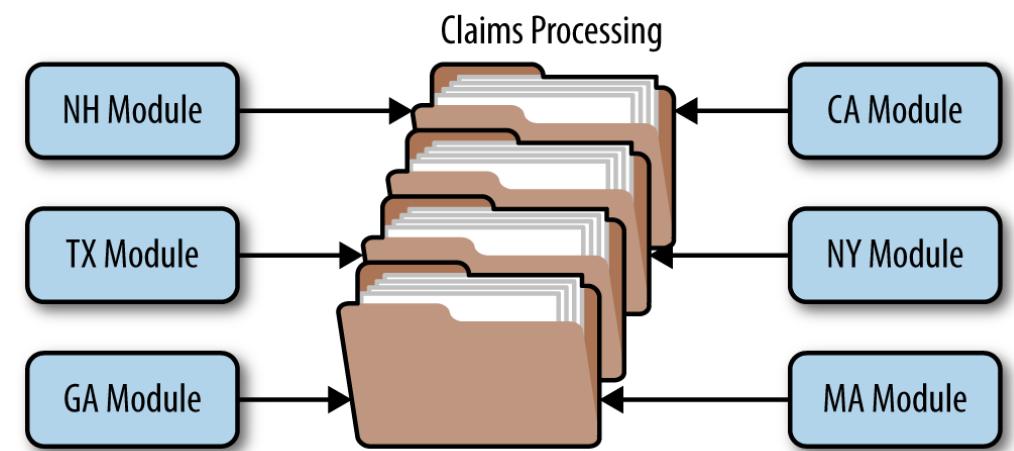
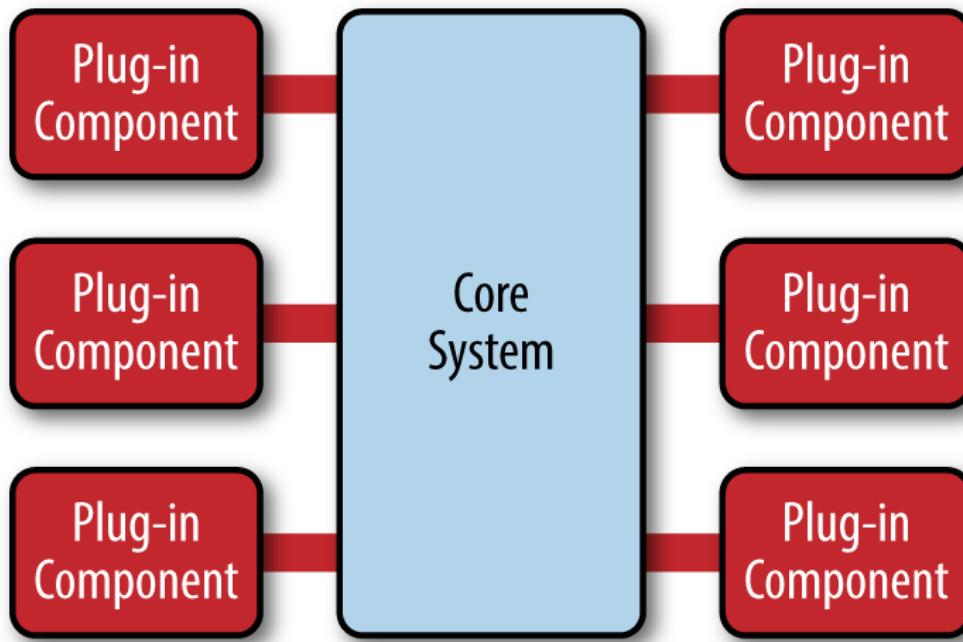
Event driven broker topology



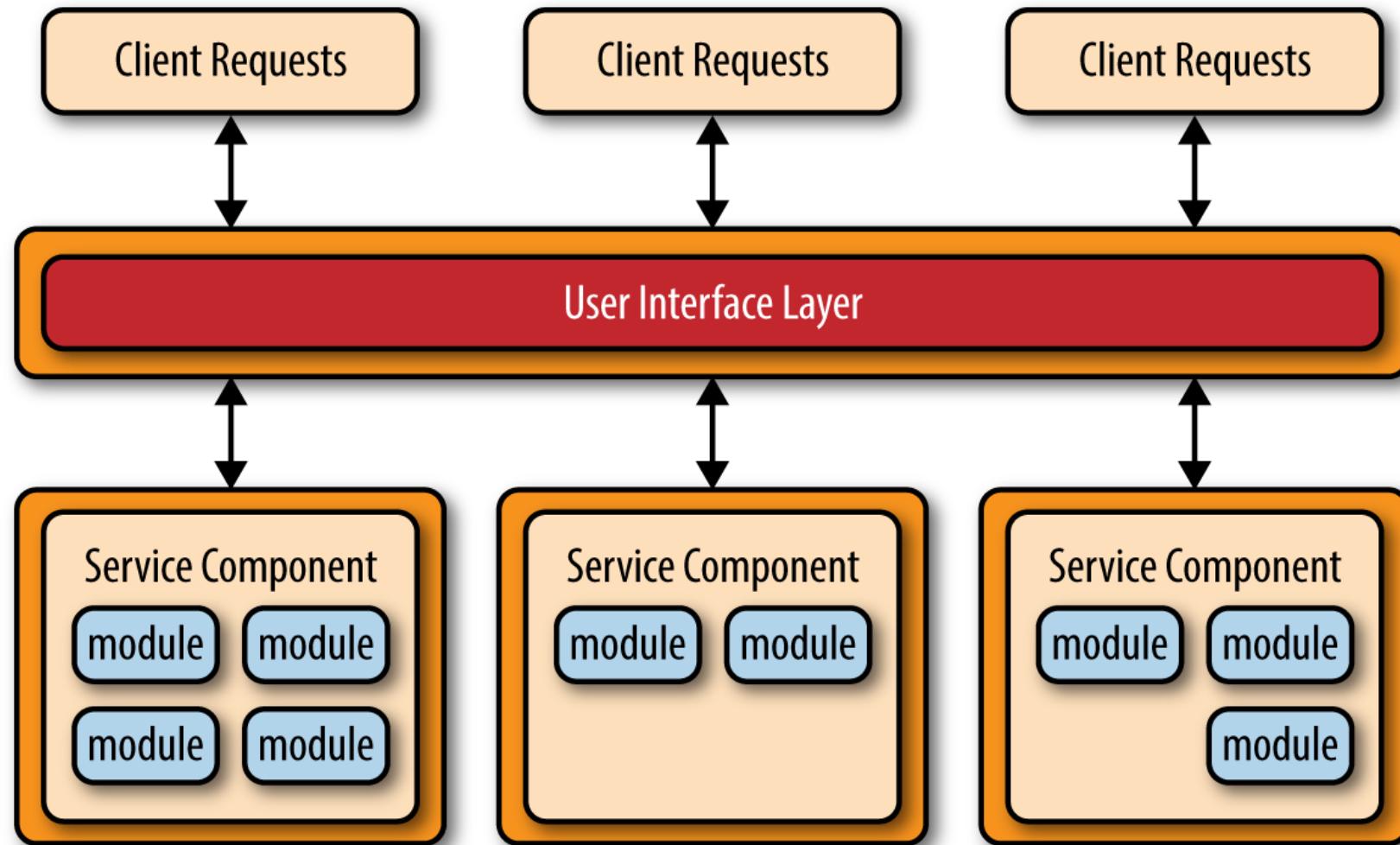
Broker topology example



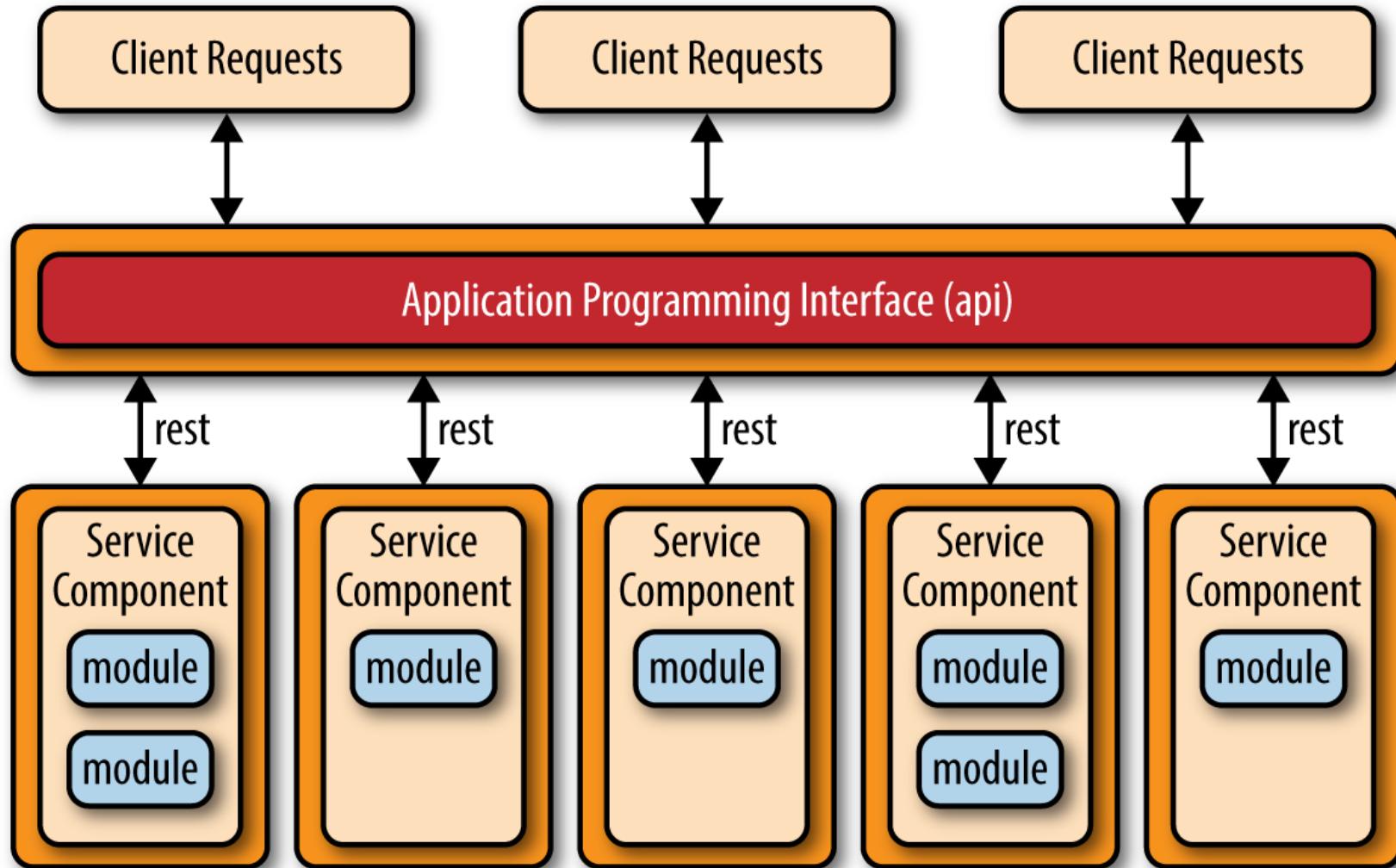
Microkernel pattern and example



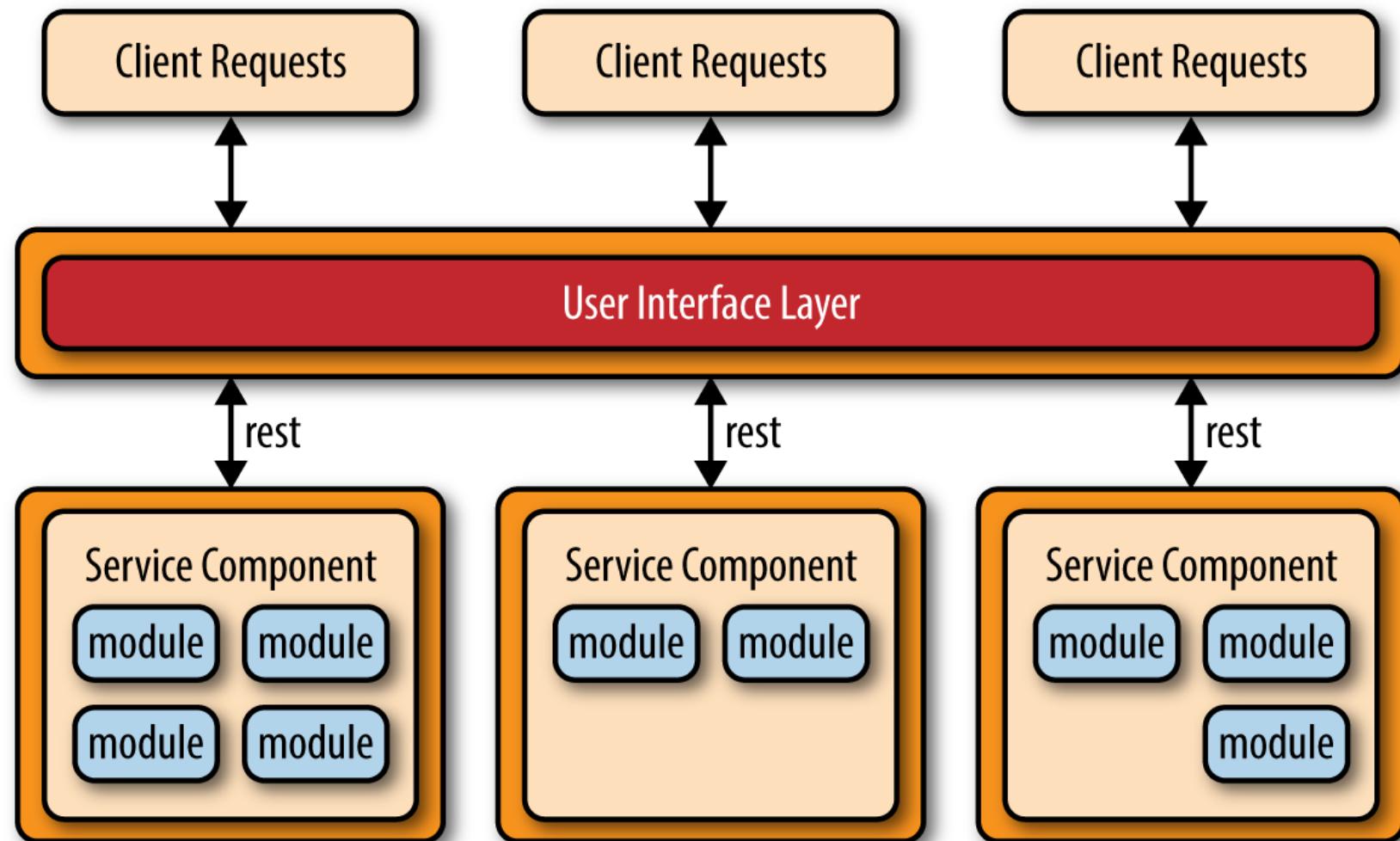
Basic microservice architecture pattern



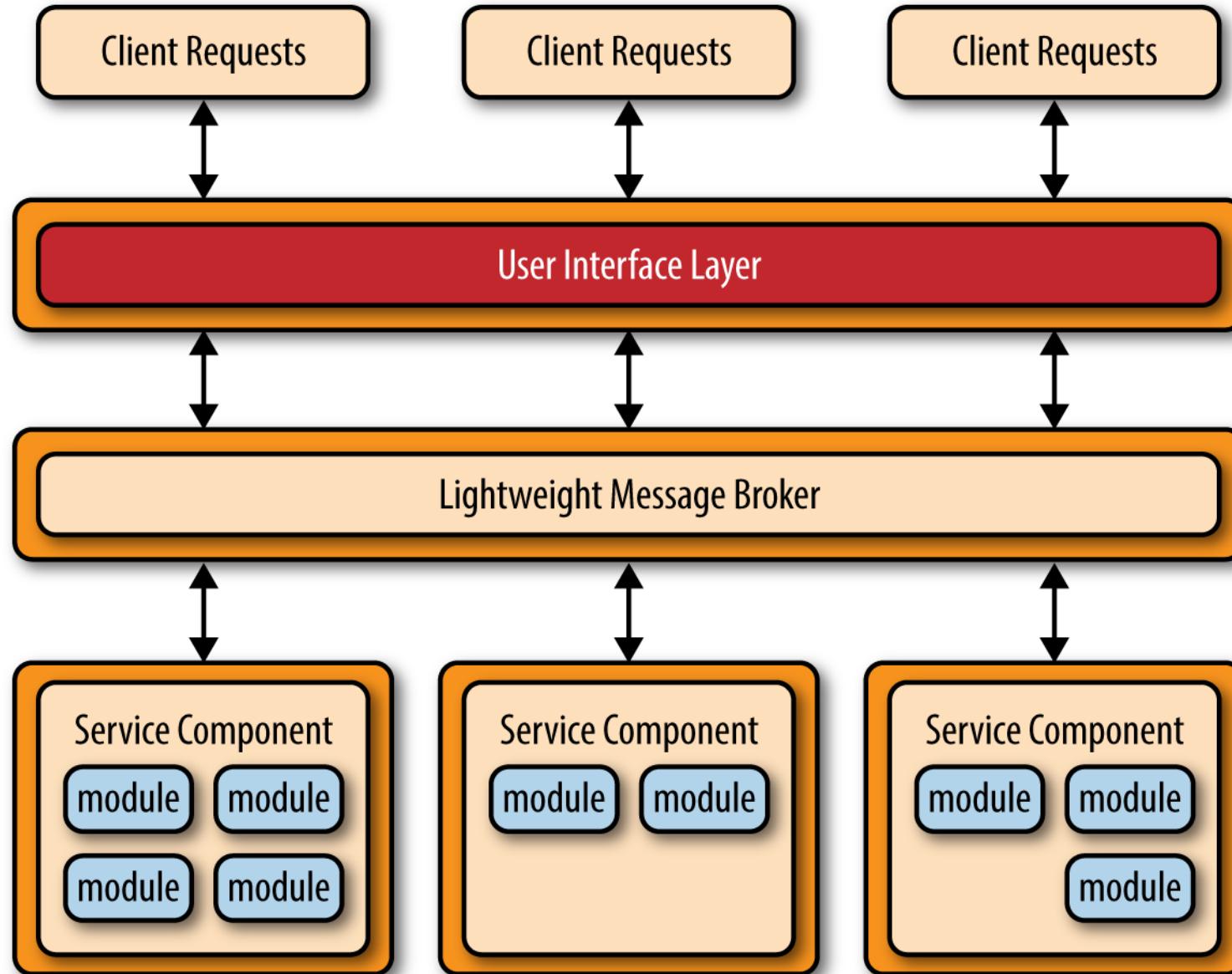
API REST-based topology



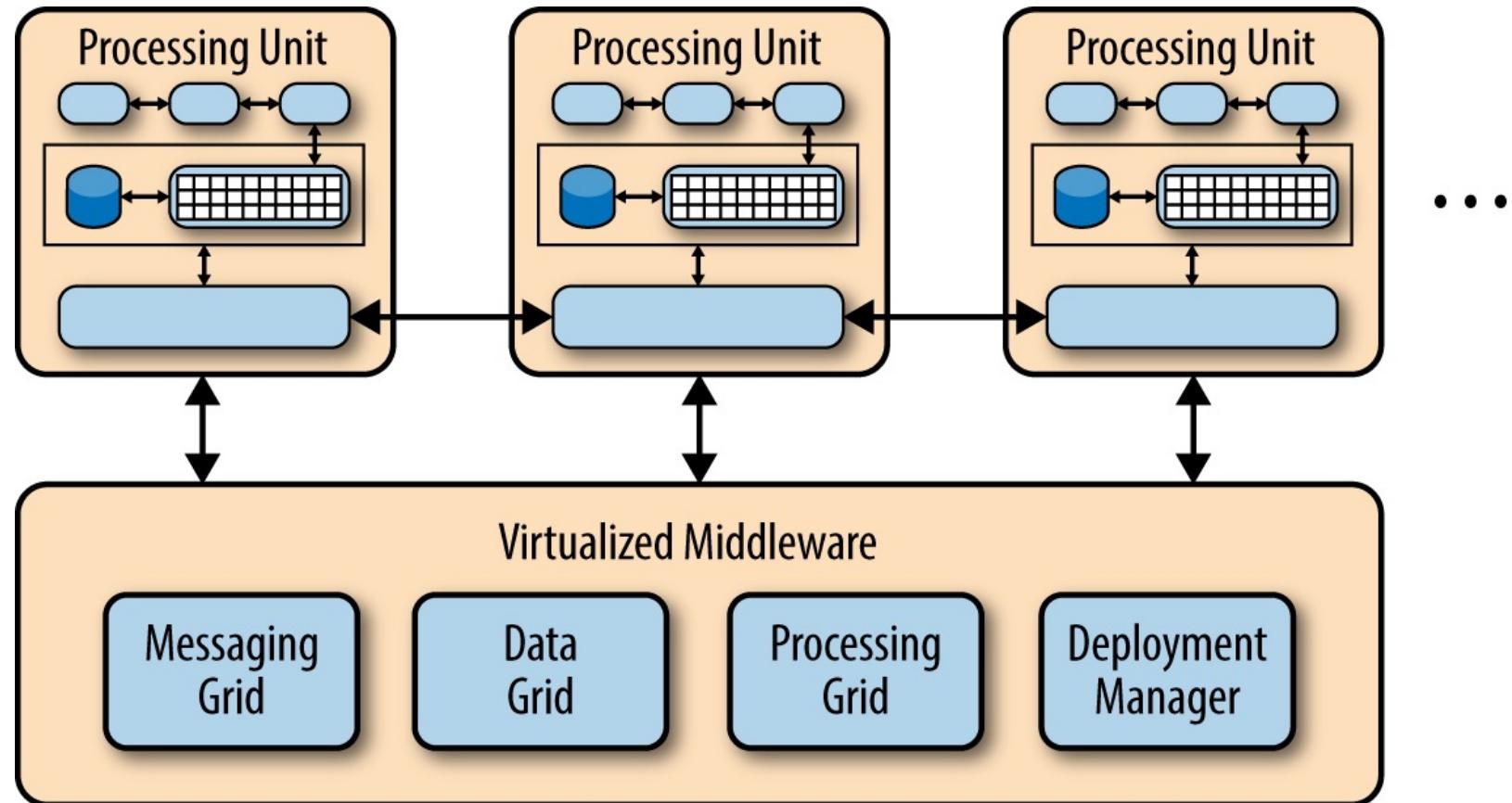
Application REST-based topology

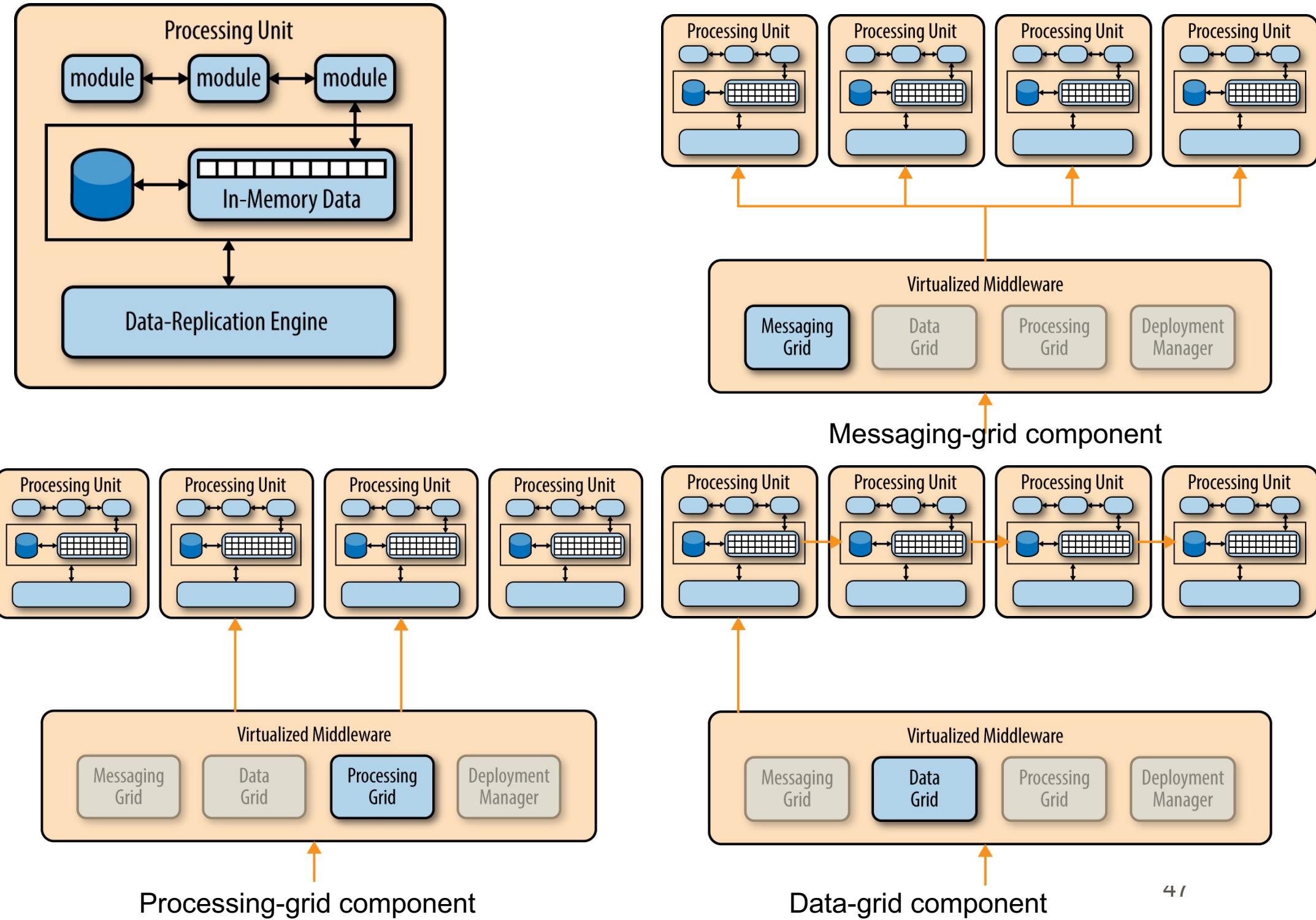


Centralized messaging (SOA-like) topology



Space-based architecture pattern (cloud-like)

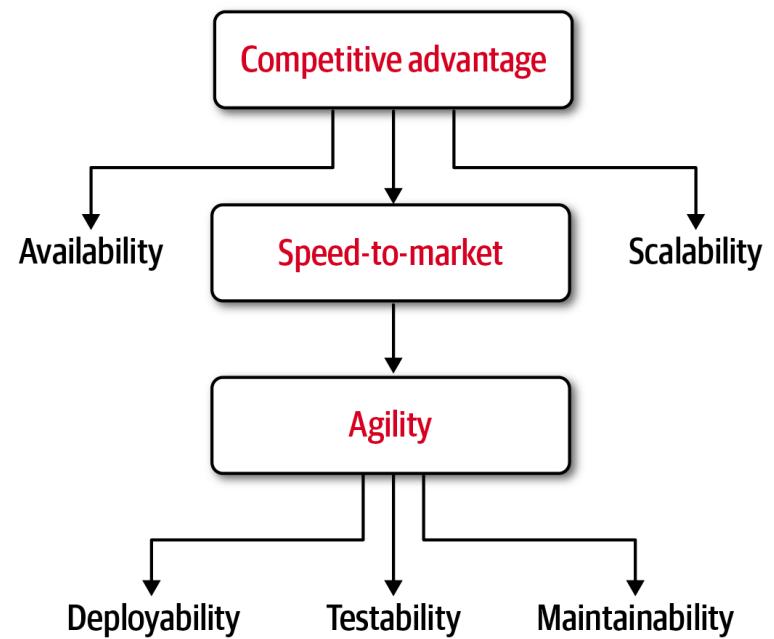




Pattern analysis comparison summary

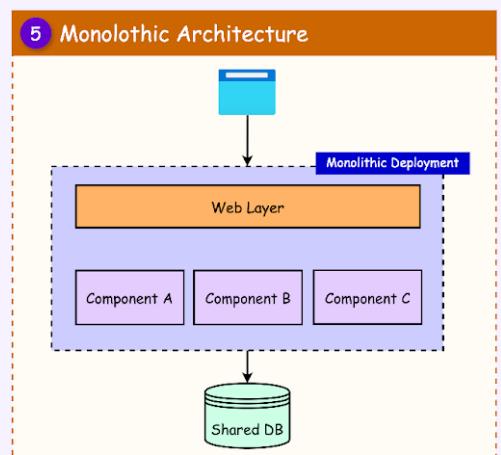
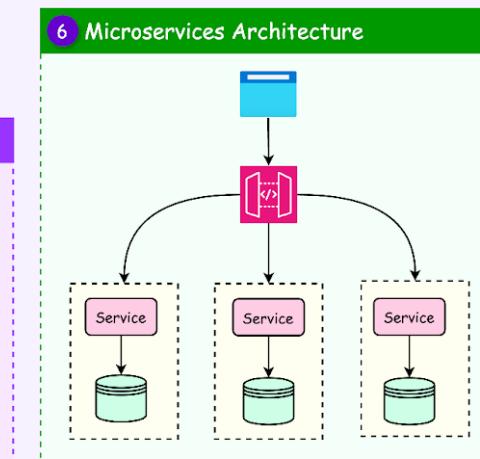
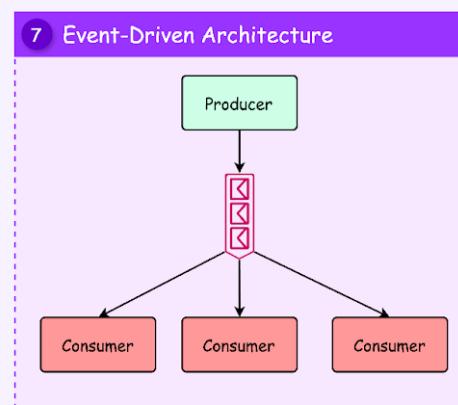
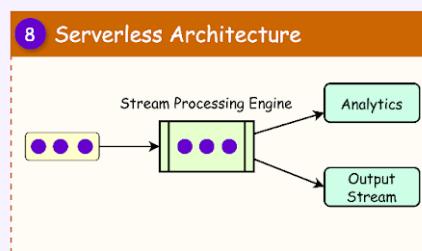
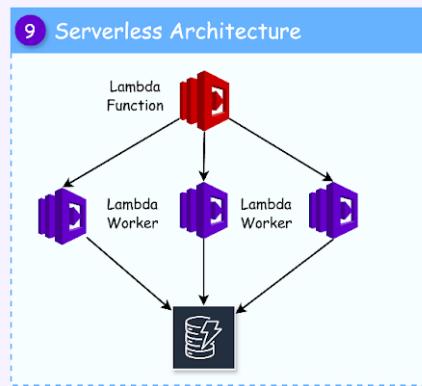
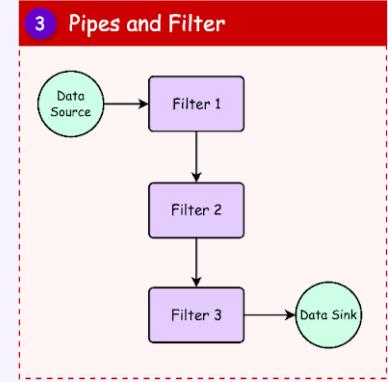
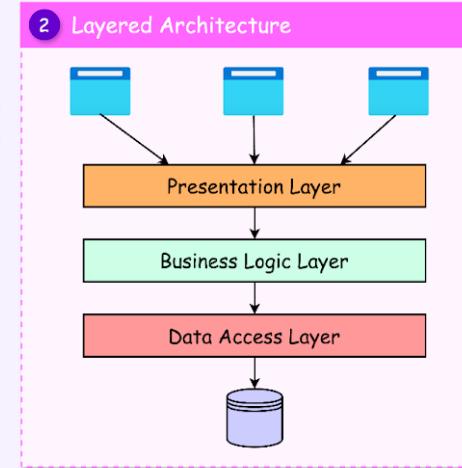
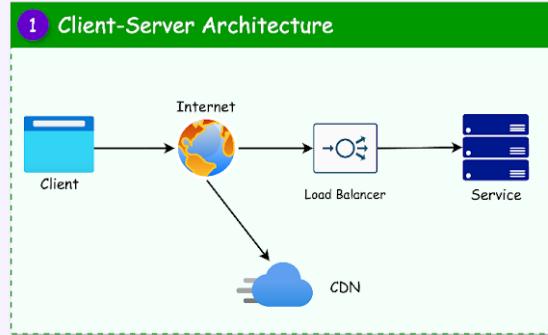
	Layered	Event-driven	Microkernel	Microservices	Space-based
Overall Agility	↓	↑	↑	↑	↑
Deployment	↓	↑	↑	↑	↑
Testability	↑	↓	↑	↑	↓
Performance	↓	↑	↑	↓	↑
Scalability	↓	↑	↓	↑	↑
Development	↑	↓	↓	↑	↓

Overall agility is the ability to respond quickly to a constantly changing environment



An alternative catalog of architectural patterns

Software Architecture Patterns



Conclusions

Describing system architectures is important to understand how systems are designed, built and maintained

Reference architecture, architectural styles, and patterns help in understanding, building and maintaining complex computer systems

Questions