Software Systems Architecture

FEUP-M.EIC-ASSO-2023-2024

Ademar Aguiar, Neil Harrison

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Patterns, Styles, and DSSAs

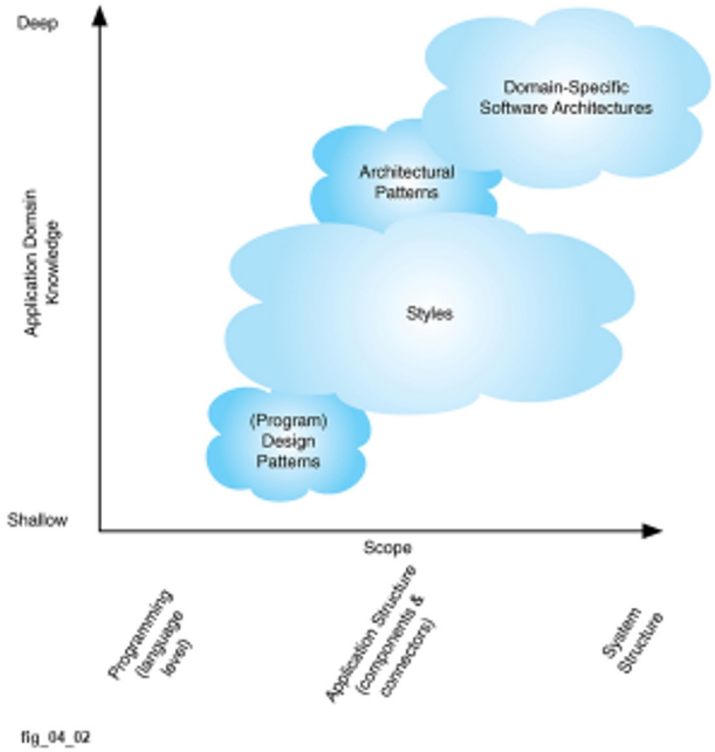


fig_04_02

A Gallery of Patterns

Some of the most common architecture patterns

Three Tier

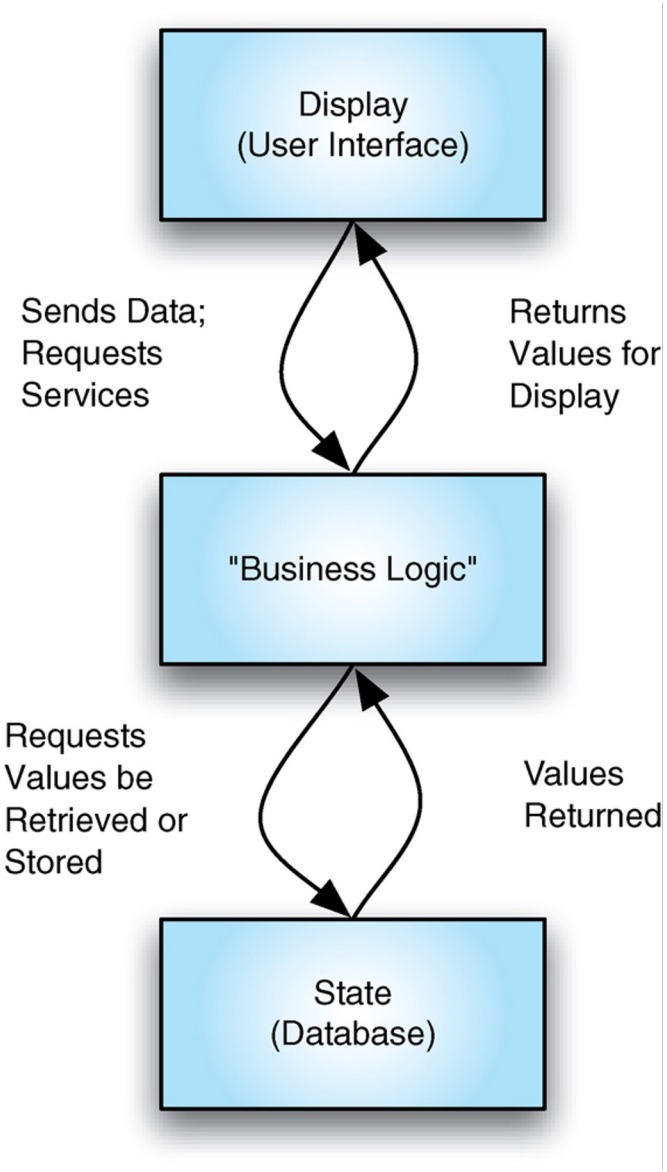
AKA: state-logic-display

User Interface

Business Logic

Data store (state)

Special case of the Layers Pattern



Batch Sequential

The whole task is subdivided into small processing steps.

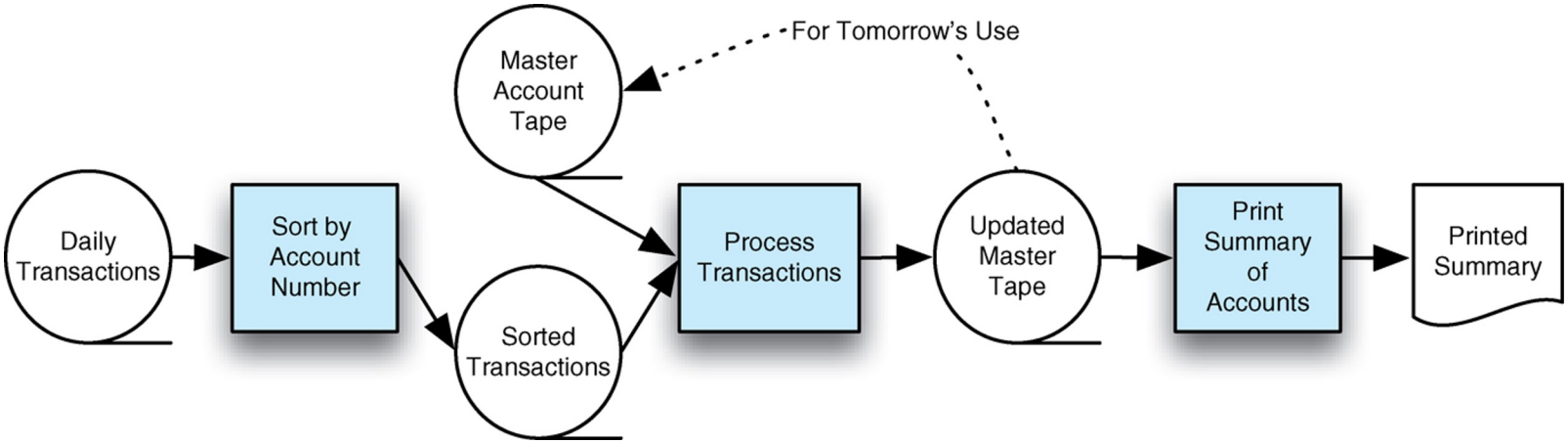
Each step is realized as a separate independent component.

Each step runs to completion and calls the next sequential step until the whole task is fulfilled.

During each step a batch of data is processed and sent as a whole to the next step.

Early computers were batch machines.

Batch Sequential



Financial records processed in batch-sequential architecture

When to use Pipes and Filters (From MS Azure)

Use this pattern when:

The processing required by an application can easily be broken down into a set of independent steps.

The processing steps performed by an application have different scalability requirements.

It's possible to group filters that should scale together in the same process. For more information, see the Compute Resource Consolidation pattern.

Flexibility is required to allow reordering of the processing steps performed by an application, or the capability to add and remove steps.

The system can benefit from distributing the processing for steps across different servers.

A reliable solution is required that minimizes the effects of failure in a step while data is being processed.

This pattern might not be useful when:

The processing steps performed by an application aren't independent, or they must be performed together as part of the same transaction.

The amount of context or state information required by a step makes this approach inefficient. It might be possible to persist state information to a database instead, but don't use this strategy if the additional load on the database causes excessive contention.

What is the difference between Batch-Sequential and Pipes and Filters?

In Batch-Sequential, each step finishes processing the set of data before the next step begins

In Pipes and Filters, the next step may begin as soon as the previous step begins outputting data.

Can you think of applications which are appropriate for either?

Compiler: which is it?

Processing a data stream: which is it?

Sense-Compute-Control

Domain: embedded control applications

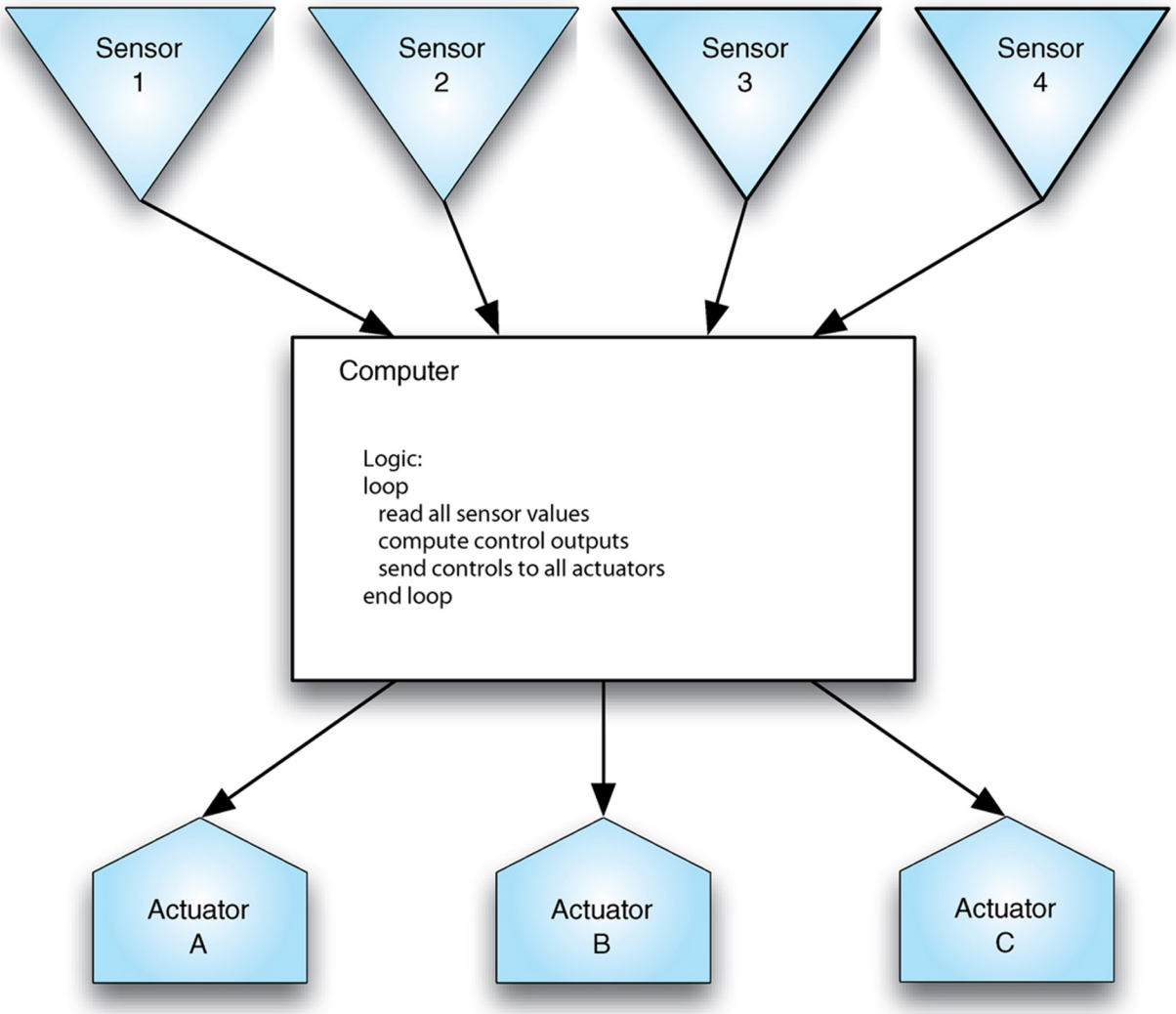
Read sensors, send actions to actuators

Similar to Event-Driven pattern

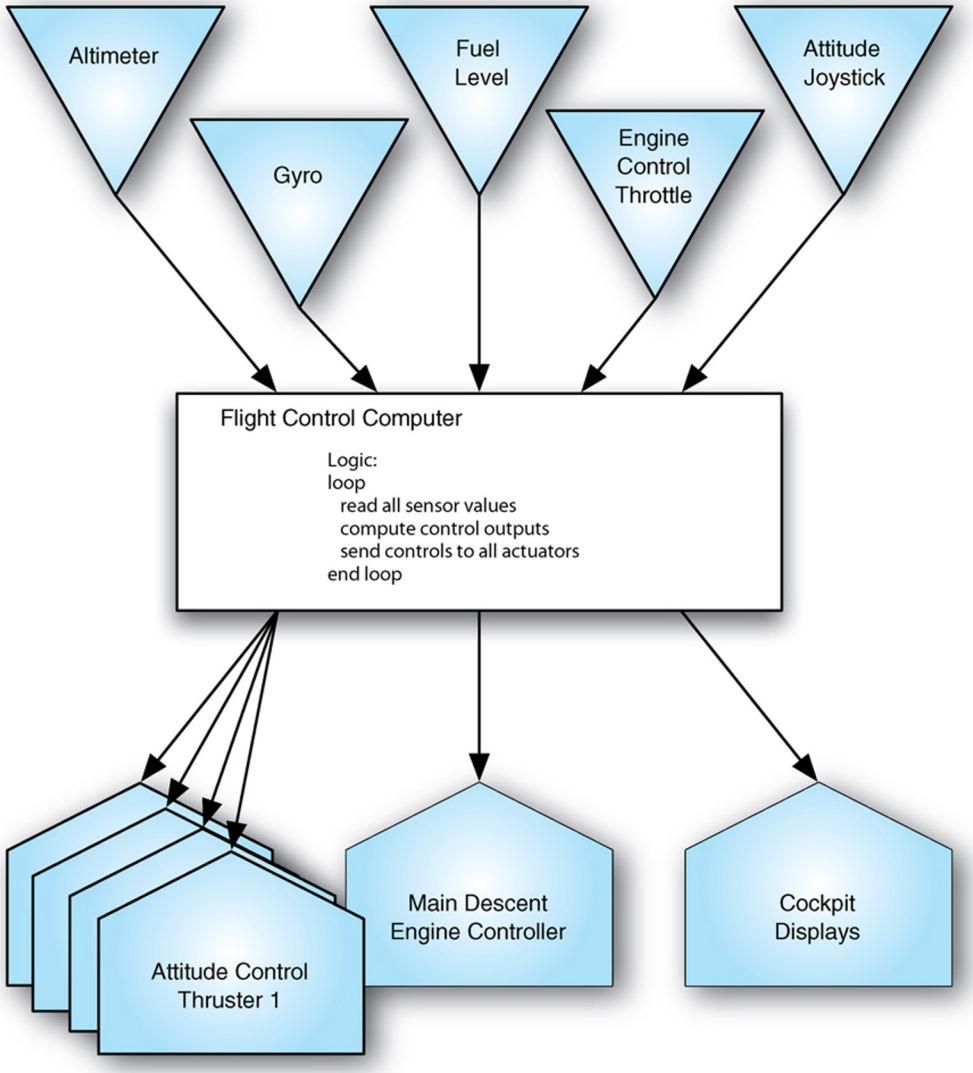
Difference:

- Event driven: handling events, rather than polling

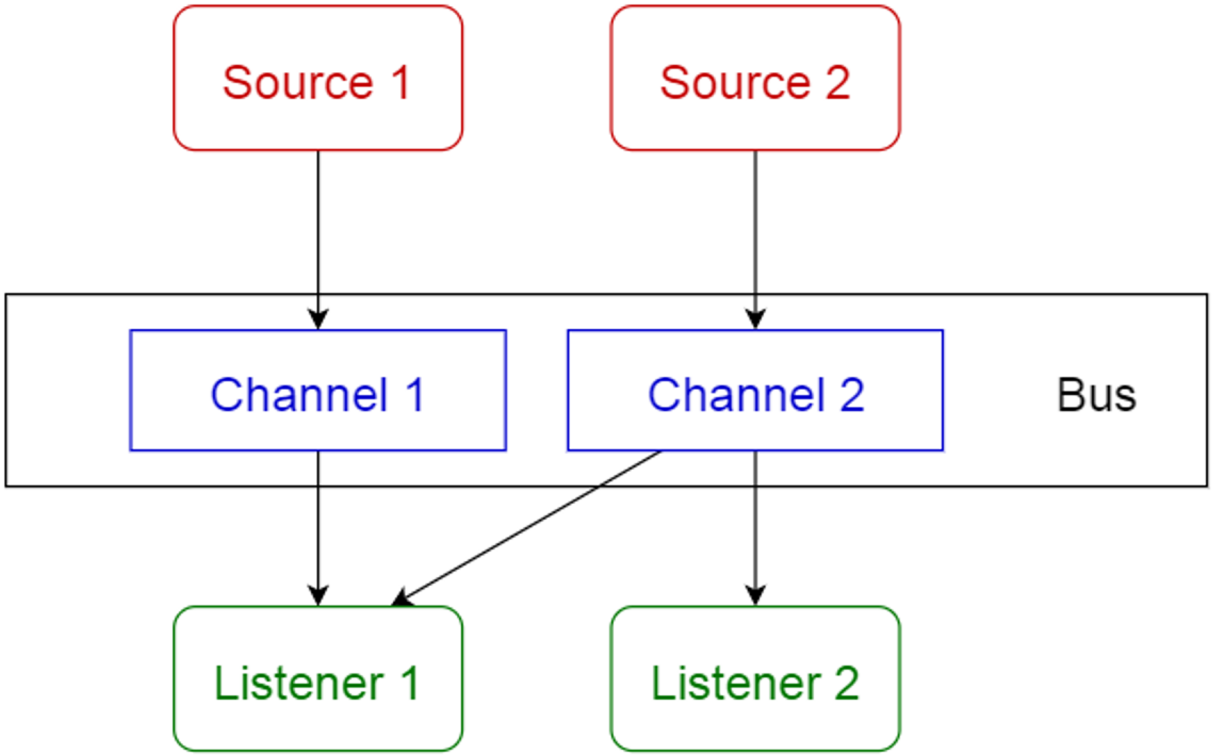
Sense-Compute-Control



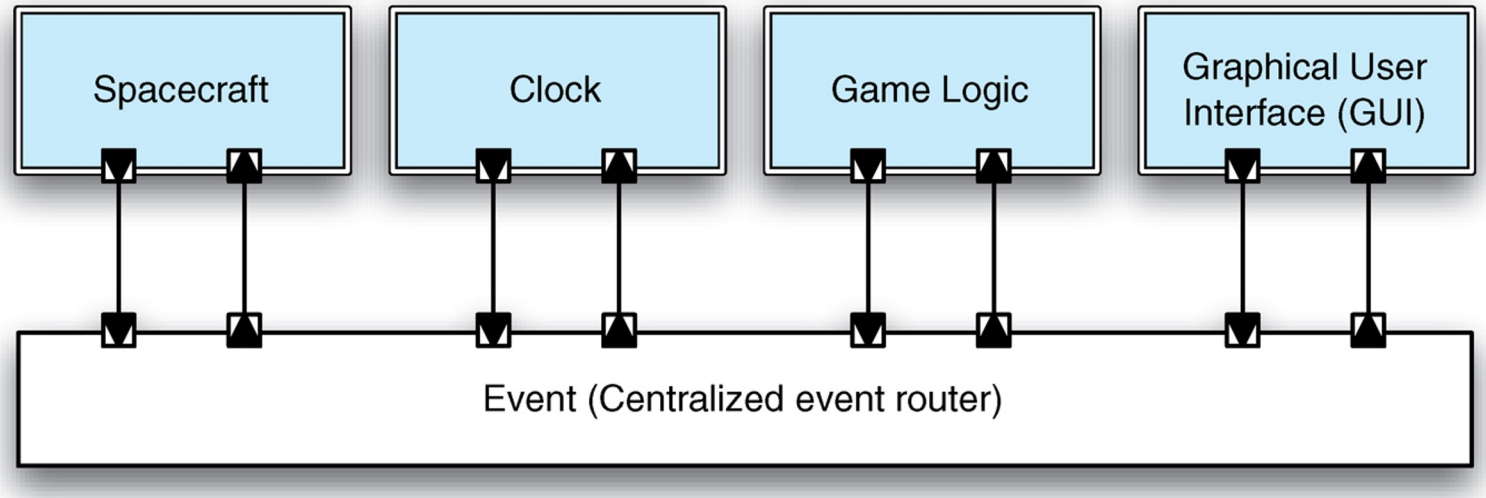
Sense-Compute-Control, Lunar Lander



Event-Driven



Event-based System



What is an Event-Driven Architecture? (From AWS)

An event-driven architecture uses events to trigger and communicate between decoupled services and is common in modern applications built with microservices. An event is a change in state, or an update, like an item being placed in a shopping cart on an e-commerce website. Events can either carry the state (the item purchased, its price, and a delivery address) or events can be identifiers (a notification that an order was shipped).

Event-driven architectures have three key components: event producers, event routers, and event consumers. A producer publishes an event to the router, which filters and pushes the events to consumers. Producer services and consumer services are decoupled, which allows them to be scaled, updated, and deployed independently.

Benefits of an event-driven architecture (from AWS)

Scale and fail independently

By decoupling your services, they are only aware of the event router, not each other. This means that your services are interoperable, but if one service has a failure, the rest will keep running. The event router acts as an elastic buffer that will accommodate surges in workloads.

Develop with agility

You no longer need to write custom code to poll, filter, and route events; the event router will automatically filter and push events to consumers. The router also removes the need for heavy coordination between producer and consumer services, speeding up your development process.

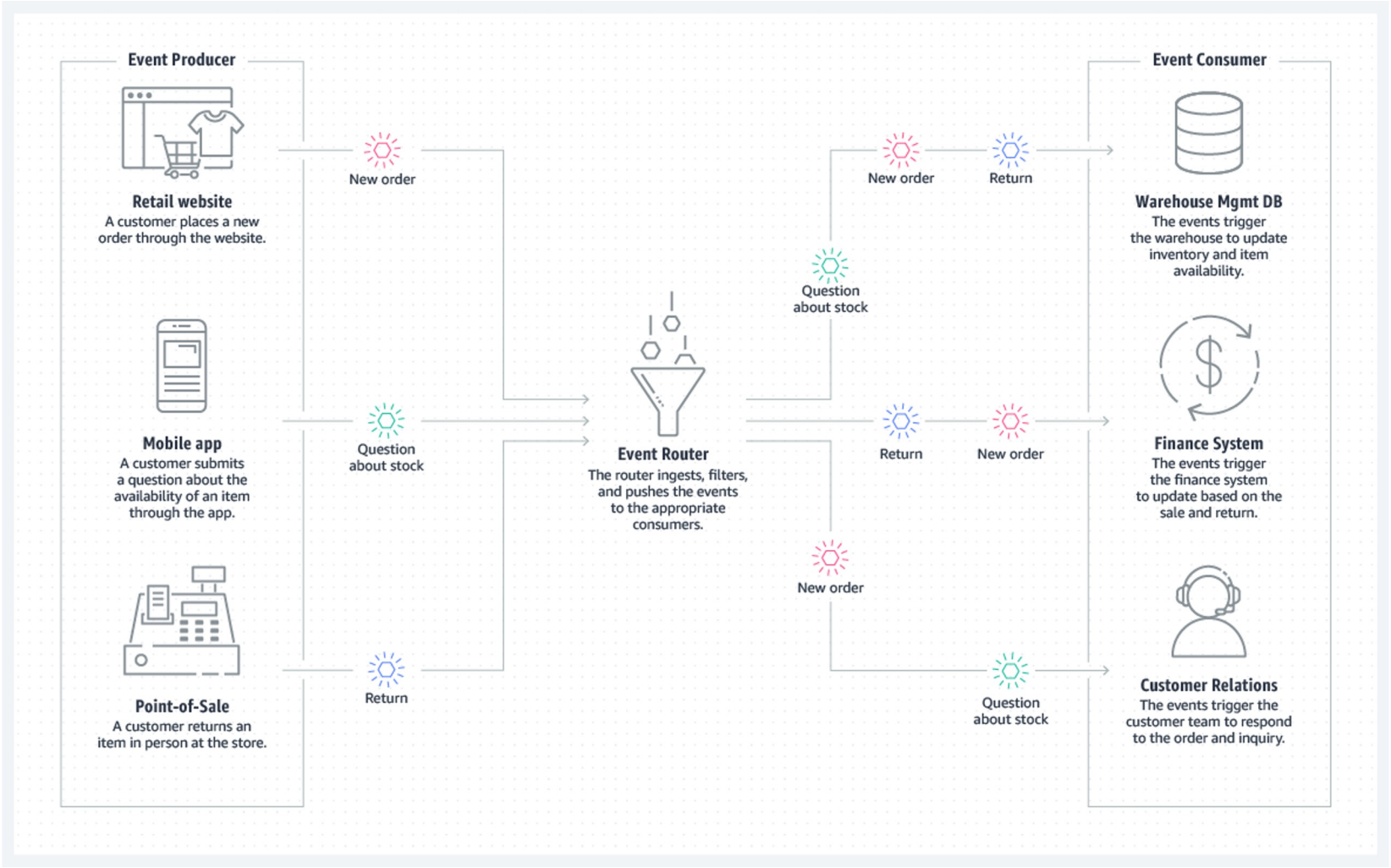
Audit with ease

An event router acts as a centralized location to audit your application and define policies. These policies can restrict who can publish and subscribe to a router and control which users and resources have permission to access your data. You can also encrypt your events both in transit and at rest.

Cut costs

Event-driven architectures are push-based, so everything happens on-demand as the event presents itself in the router. This way, you're not paying for continuous polling to check for an event. This means less network bandwidth consumption, less CPU utilization, less idle fleet capacity, and less SSL/TLS handshakes.

Event driven architecture from AWS (caveat: also a Broker)



Java Swing is event-driven

```
public class FooPanel extends JPanel implements ActionListener {
  public FooPanel() {
    super();

    JButton btn = new JButton("Click Me!");
    btn.addActionListener(this);

    this.add(btn);
  }

  @Override
  public void actionPerformed(ActionEvent ae) {
    System.out.println("Button has been clicked!");
  }
}
```

Microkernel

Provide a common plug-and-play interface for (usually) low level operations

Implements services that all systems in the “family” need

Accessed through APIs provided

Often a layer

Example: the JVM

Highlights:

- Portability!
- Also provides a common point for enhancements
- But performance may suffer
  - Example: common virtual machine for programming apps for mobile devices

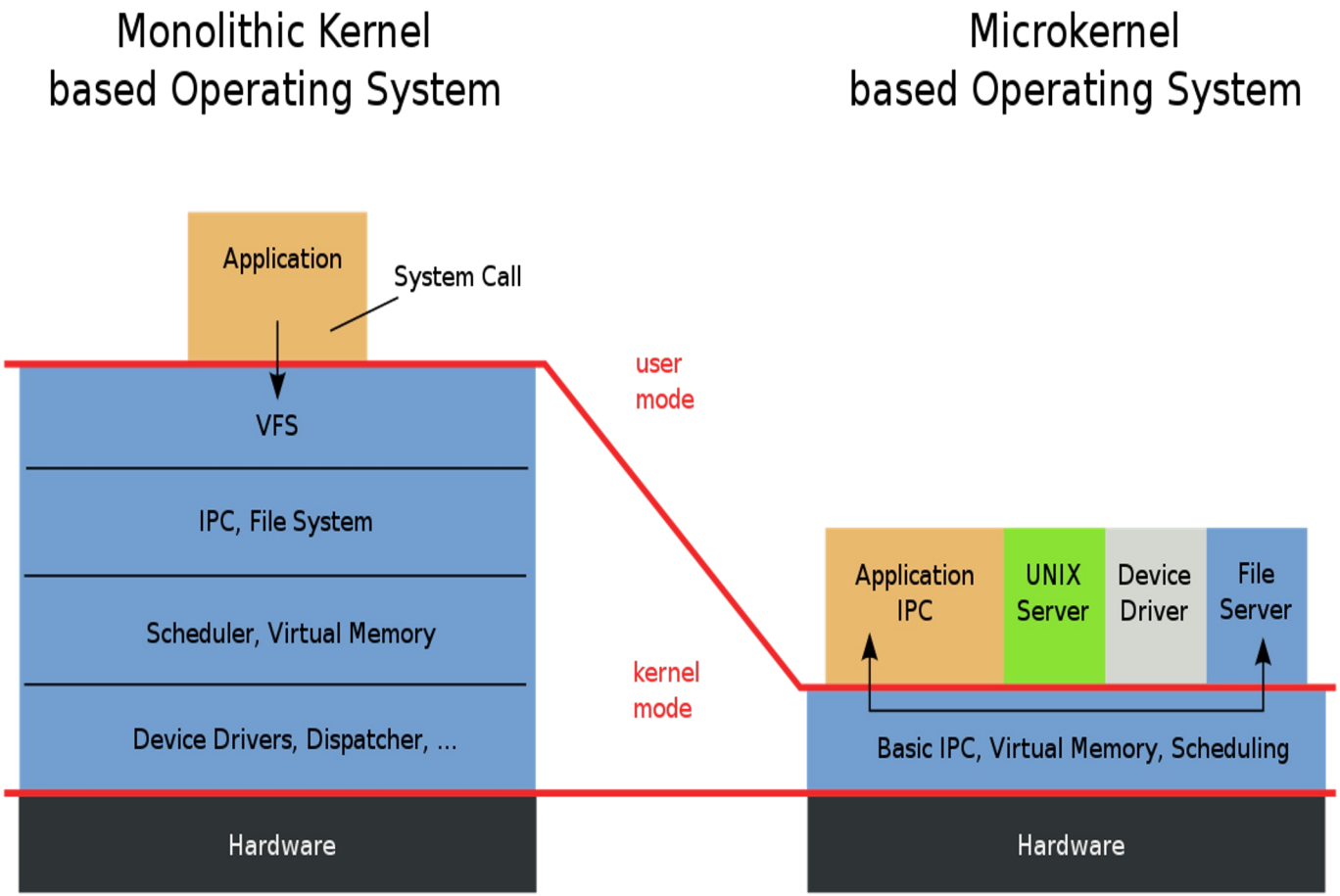
Microkernel: Another definition

The microkernel architecture pattern consists of two types of architecture components: a *core system* and *plug-in modules*. Application logic is divided between independent plug-in modules and the basic core system, providing extensibility, flexibility, and isolation of application features and custom processing logic.

The core system of the microkernel architecture pattern traditionally contains only the minimal functionality required to make the system operational. Many operating systems implement the microkernel architecture pattern, hence the origin of this pattern's name. From a business-application perspective, the core system is often defined as the general business logic sans custom code for special cases, special rules, or complex conditional processing.

https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch03.html#:~:text=The%20microkernel%20architecture%20pattern%20(sometimes,a%20typical%20third%2Dparty%20product.

Microkernel vs. Monolithic Kernel



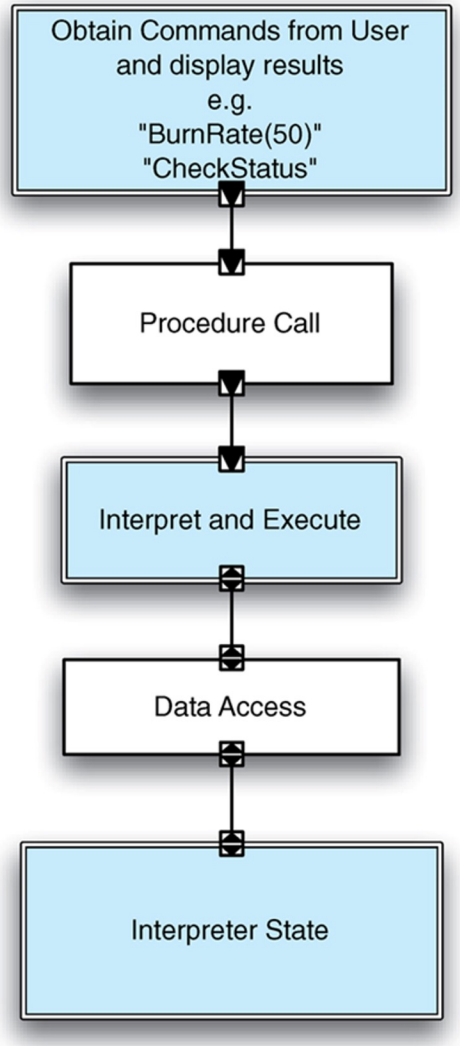
Interpreter

A language syntax and grammar needs to be processed at runtime

An interpreter provides parsing facilities and an execution environment.

Note: there is a GoF pattern called Interpreter. It's a class structure to implement interpretation.

Lunar Lander as an Interpreter



Interpreter: Architecture vs. Design

What is the difference between

- The Interpreter Architecture Pattern
- The Interpreter Design Pattern

C++ Compiler: uses the Interpreter Design Pattern to build an abstract syntax tree

Python: it IS an interpreter (the Interpreter Architecture pattern)

- (of course, it also uses the Interpreter Design pattern)

Publish-Subscribe

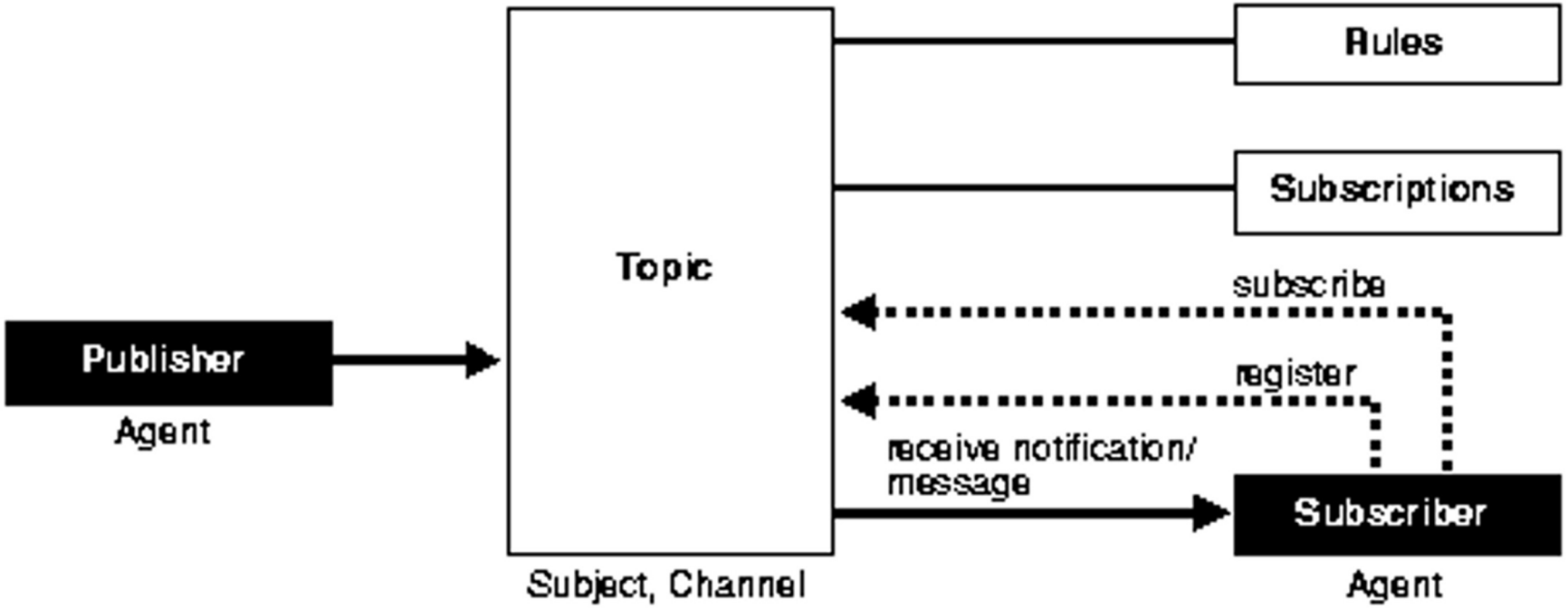
Allows event consumers (subscribers) to register for specific events

Producers publish (raise) specific events that reach a specified set of consumers

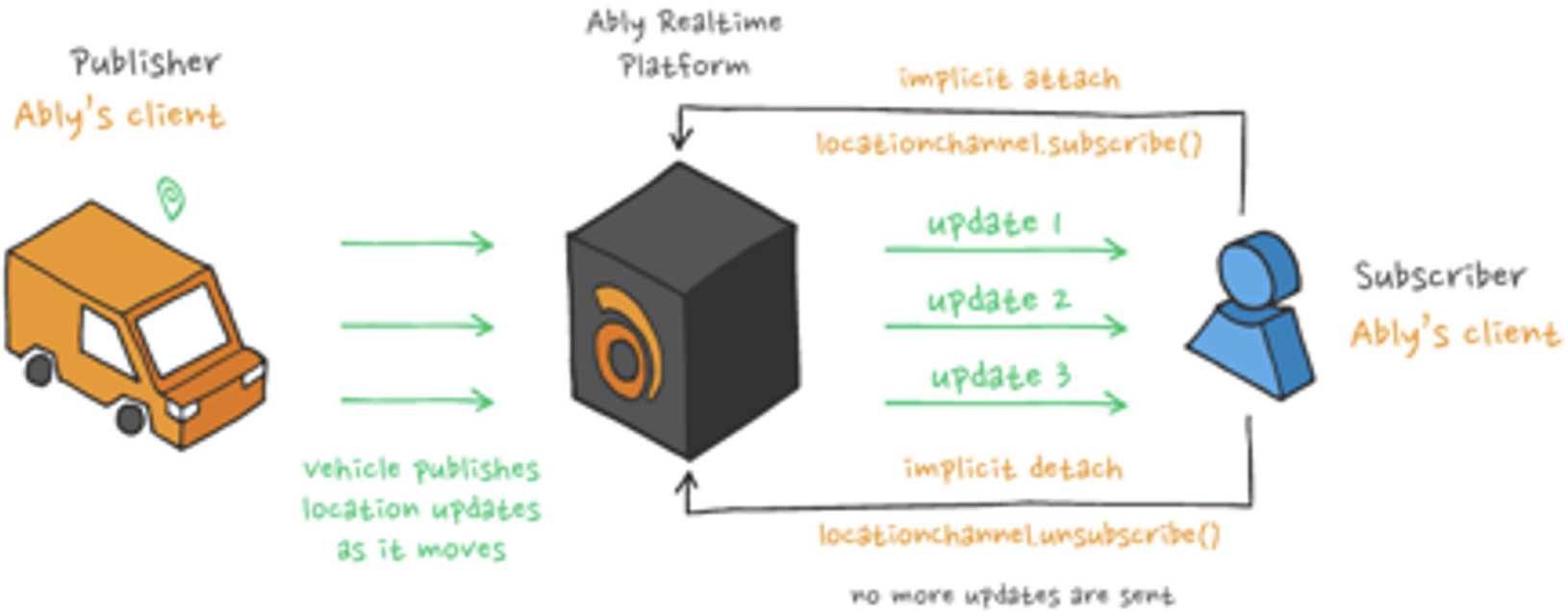
Executes a callback-operation to the event consumers

The Observer GoF pattern at the architectural level

Publish-Subscribe



Publish-Subscribe



Client Server

A central server provides services to multiple clients; usually distributed

Two kinds of components: clients and servers

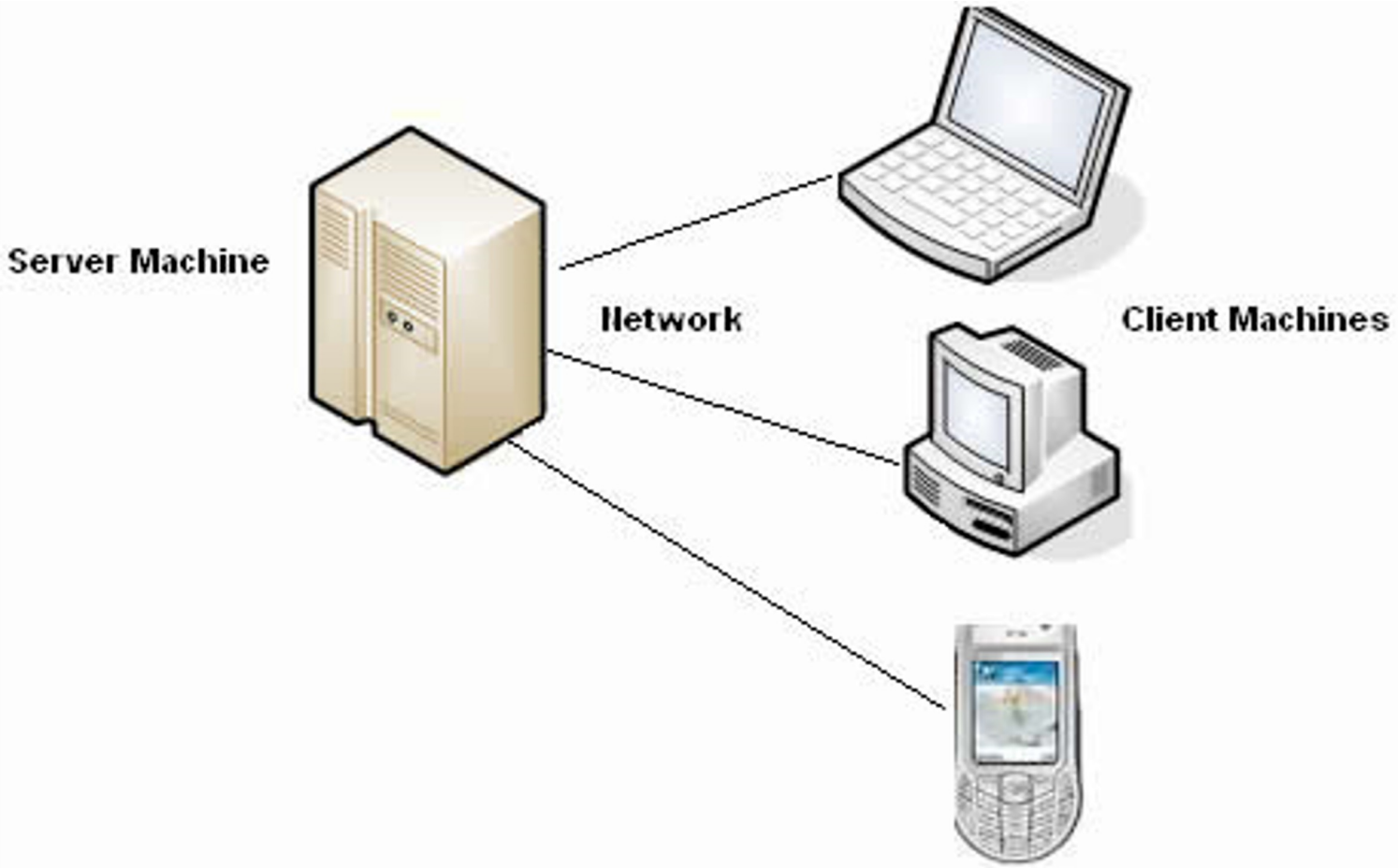
- The client requests information or services from a server.
- It needs to know an ID or address of the server.
- The server responds to requests, and processes each independently
- The server does not know the address of the client beforehand
- Clients are optimized for their application task; servers are optimized for serving multiple clients.

Extremely common, especially among distributed applications

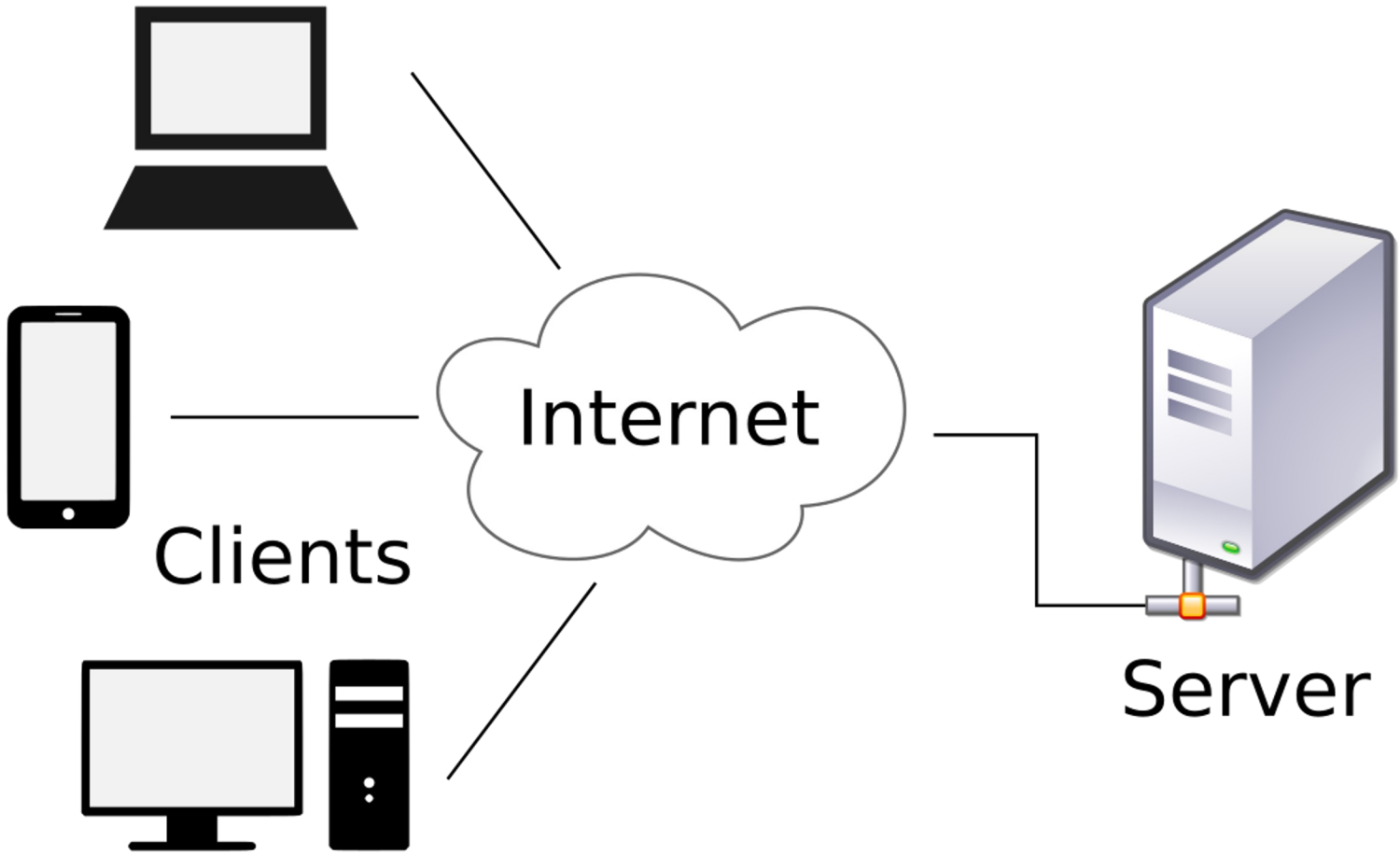
Highlights

- Usually serves a variable number of clients; may be heterogeneous
- Systems often scale well by adding more servers
- Can use multiple servers to increase availability (with some caveats)

Client Server



Client Server



Peer-to-Peer

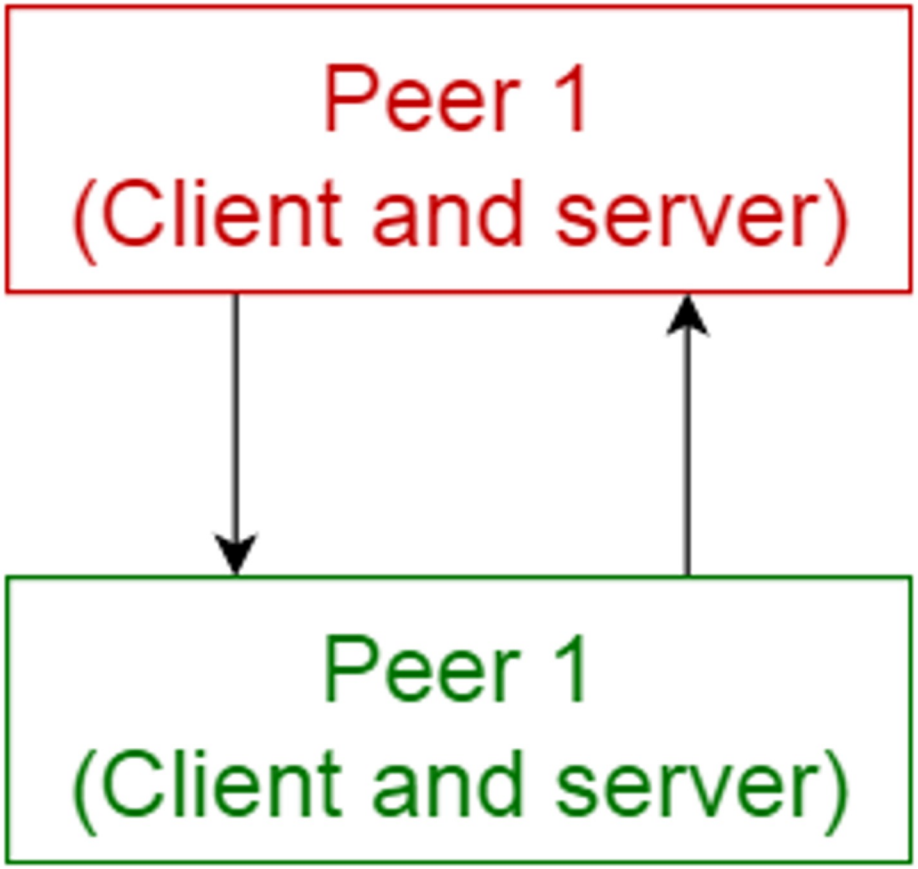
Distributed components

Each component has equal responsibilities (i.e., can act as both a client and a server)

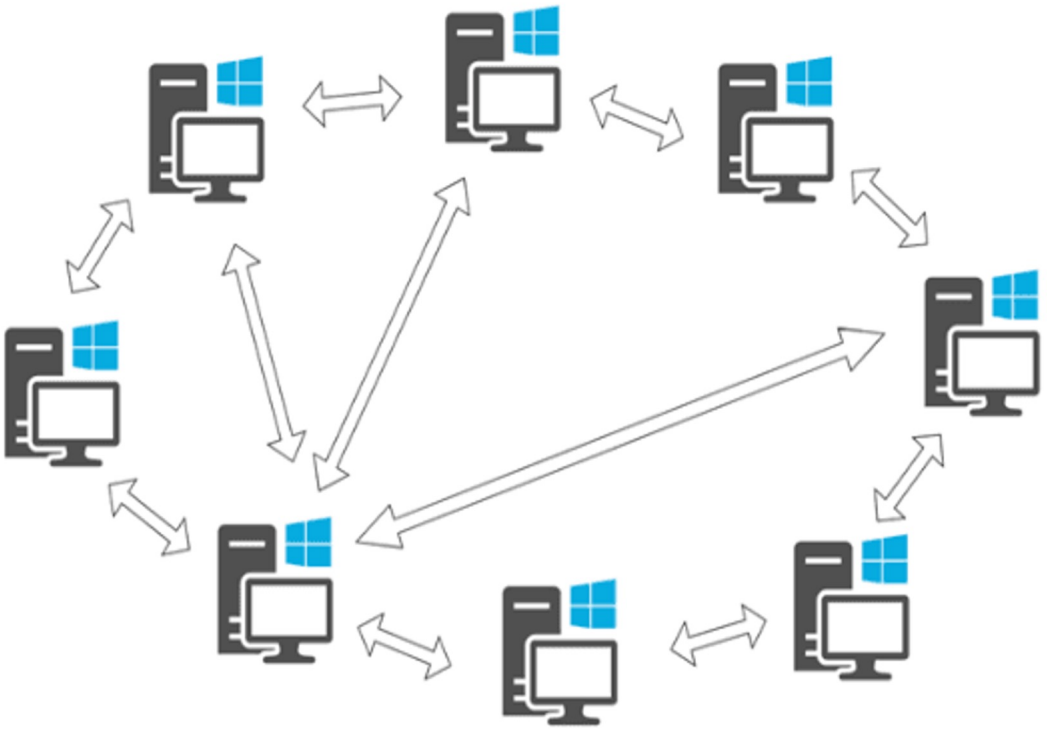
Each component offers its own services or data

A peer-to-peer network often consists of a dynamic number of components

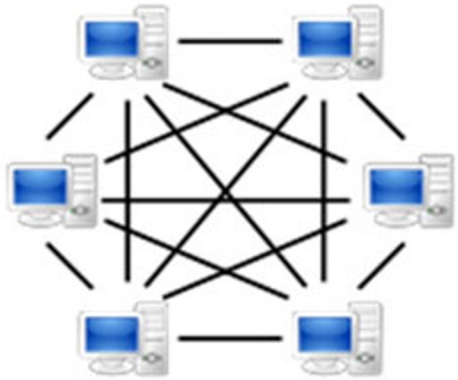
Peer to Peer



Peers: not every peer needs to connect with every other one



Server and Peer-to-Peer

A **server** based network	A **peer-to-peer** based network.
	

Client-Server vs. Peer-to-Peer

Advantages and disadvantages of each?

Peer-to-Peer

Advantages

Easy and simple to set up only requiring a hub or a switch to connect all computers together.

You can access any file on the computer as-long as it is set to a shared folder.

If one computer fails to work all the other computers connected to it still continue to work.

Disadvantages

Security is not good other than setting passwords for files that you don't want people to access.

If the connections are not connected to the computers properly then there can be problems accessing certain files.

It does not run efficiently if you have many computers, it is best to use two to eight computers. (But you can have overcome this in various ways!)

Client-Server

Advantages

A client server can be scaled up to many services that can also be used by multiple users.

Security is more advanced than a peer-to-peer network, you can have passwords to own individual profiles so that nobody can access anything when they want.

All the data is stored on the servers which generally have better security than most clients.

The server can control the access and resources better to guarantee that only those clients with the appropriate permissions may access and change data.

Disadvantages

More expensive than a peer-to-peer network: you must pay for the start up cost.

When the server goes down or crashes all the computers connected to it become unavailable to use.

When you load the server, it starts to slow down due to the bit rate per second.

When everyone tries to do the same thing, it takes a little while for the server to do certain tasks.

Broker

Separates the communication functionality of a distributed system from its application functionality

The Broker hides and mediates all communication between the components of a system (often clients and server)

The Broker consists of a client-side requestor to construct and forward invocations, and a server-side invoker to invoke the operations on the target object.

Most Brokers are extensions of Client-Server systems

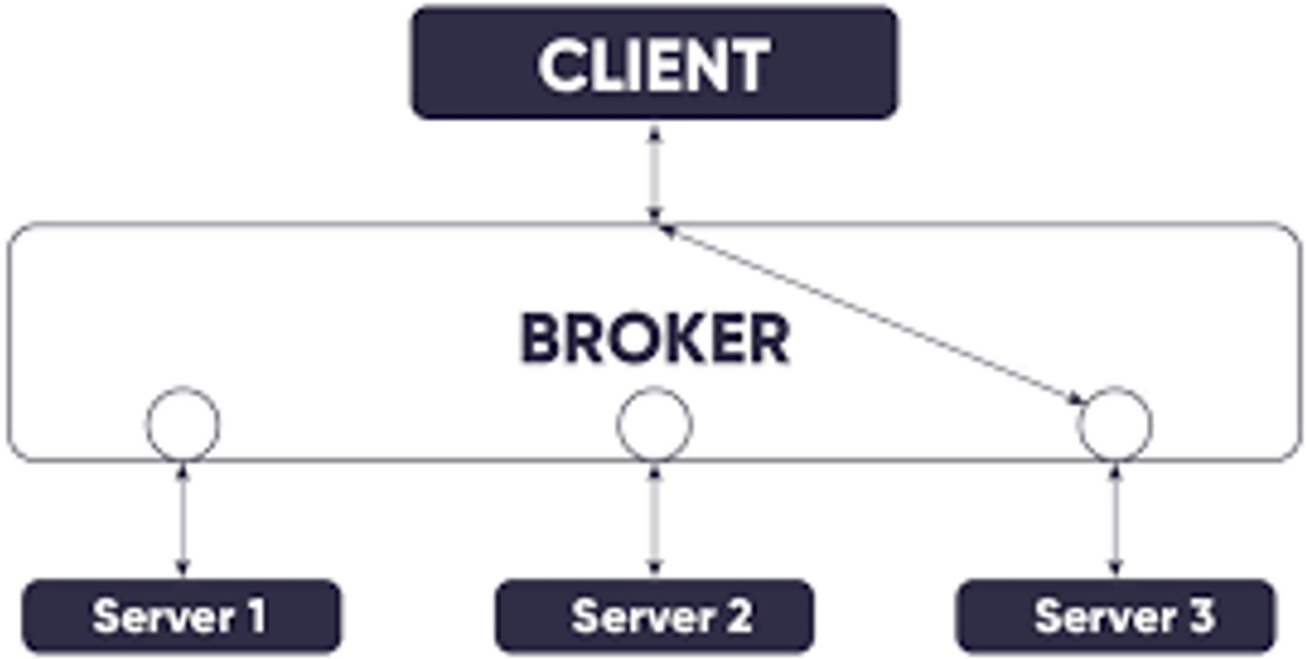
Highlights:

- Distributes them to servers for optimal performance
- Provides security, e.g., acts as a firewall or an authenticator
- Makes duplicating servers for load or availability easy

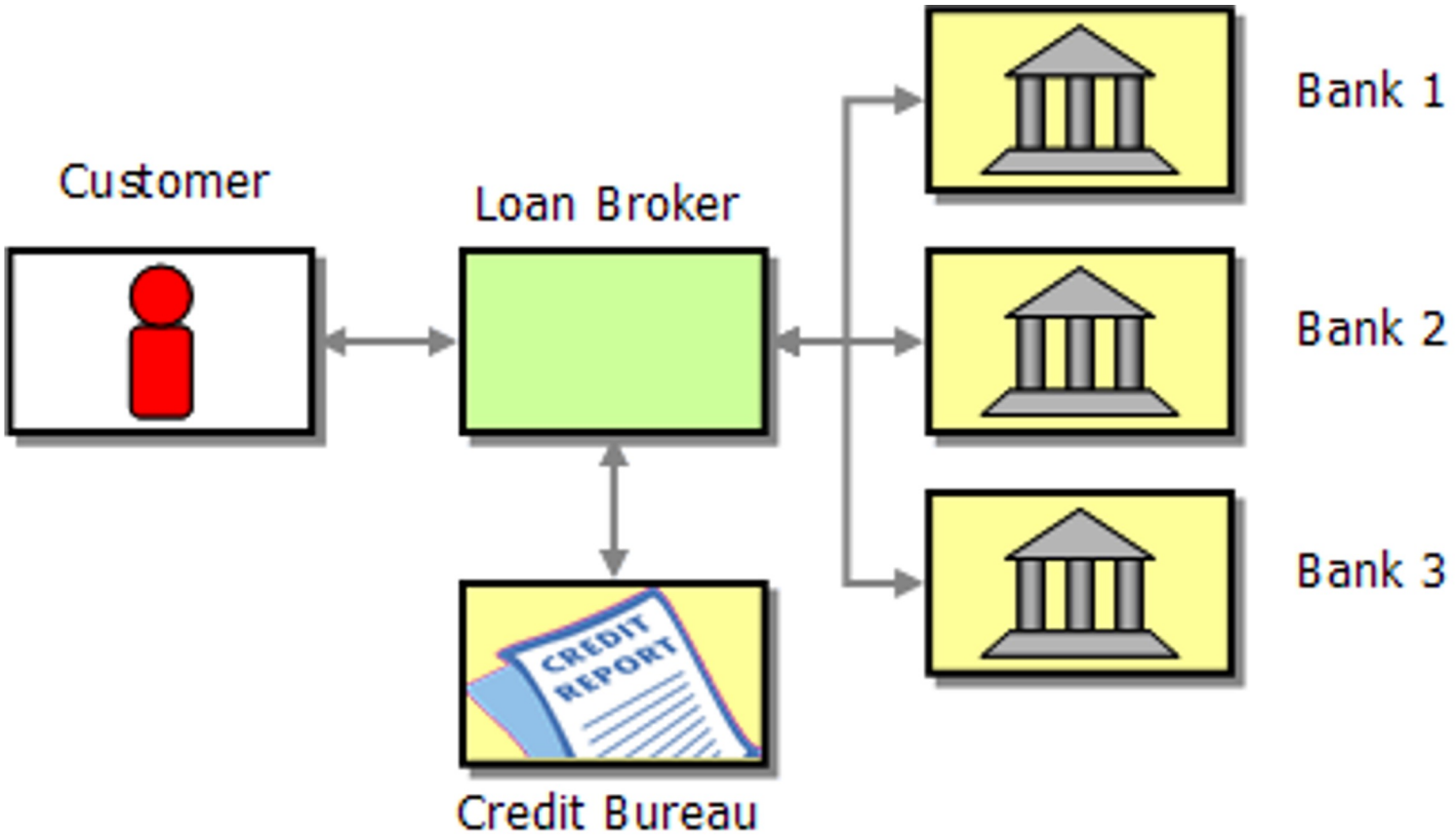
But

- Can the broker component be a bottleneck or a single point of failure?

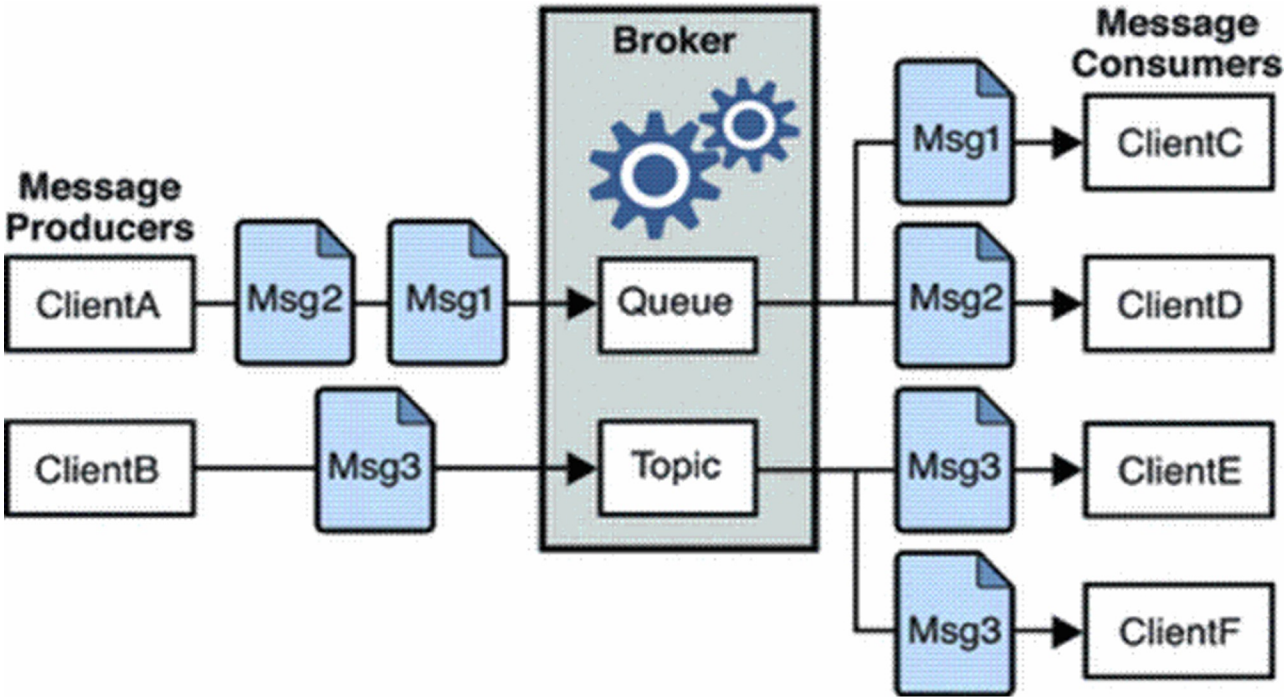
Broker



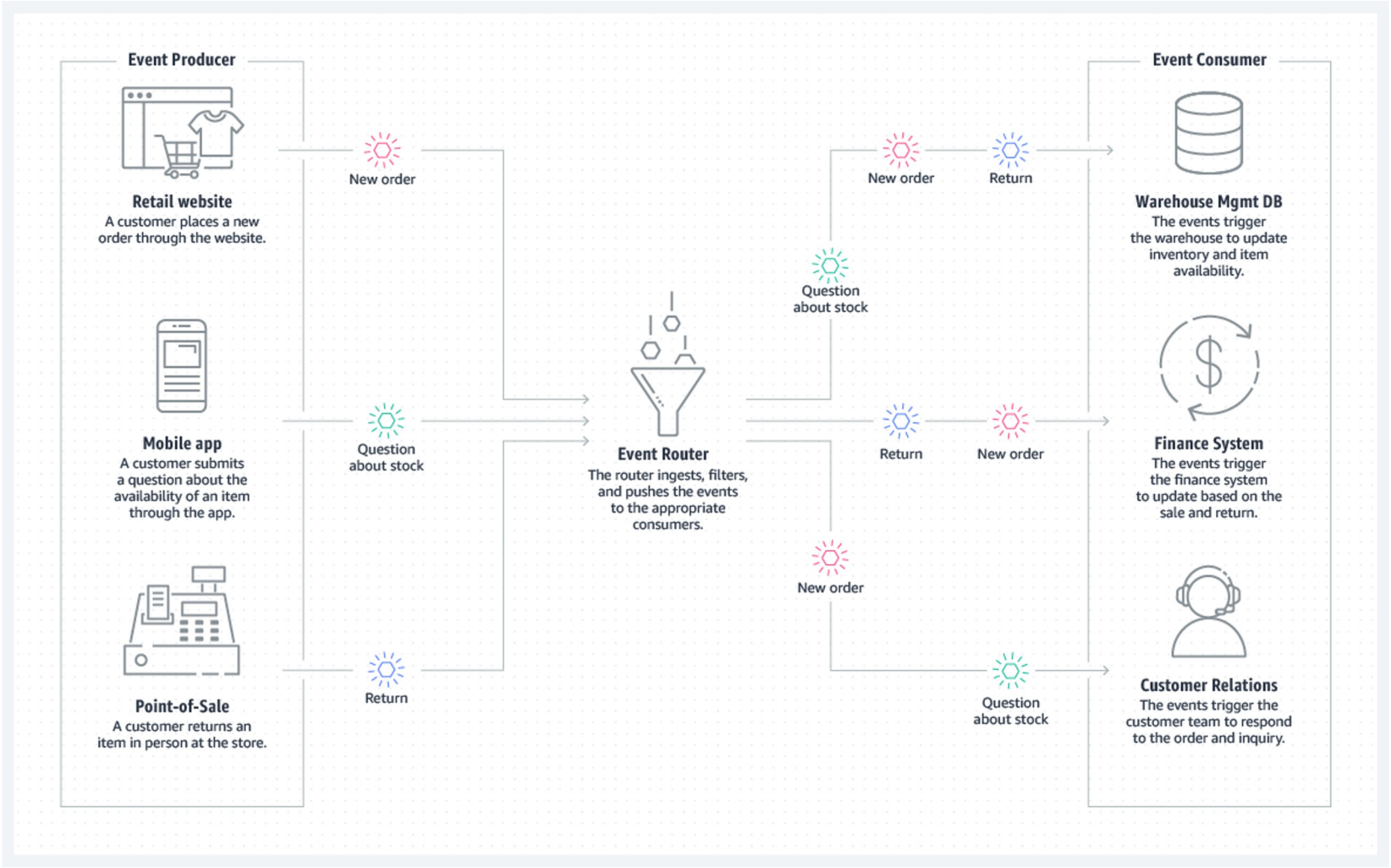
(From Enterprise Integration Patterns)



Broker


Broker
Message Consumers
Message Producers
ClientA
Msg2
Msg1
Queue
ClientB
Msg3
Topic
Msg1
ClientC
Msg2
ClientD
Msg3
ClientE
Msg3
ClientF

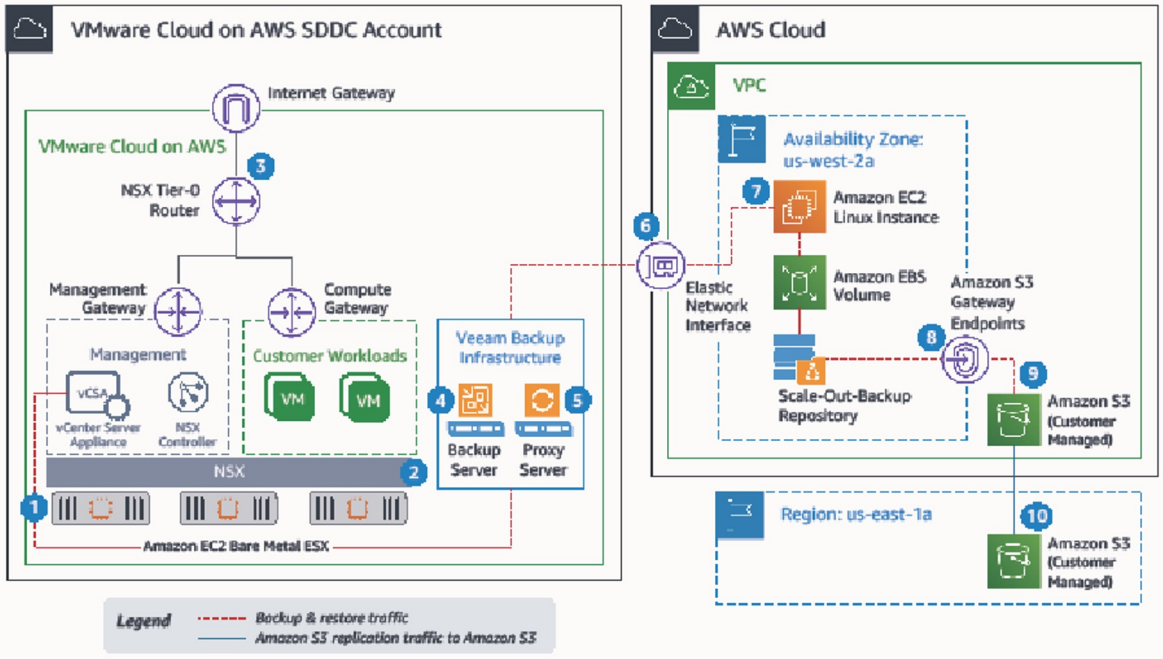
Event driven architecture from AWS, It's also a Broker



Broker, Complex example

Veeam Backup on VMware Cloud on AWS

Repository options for storing backup data with Veeam using native AWS Cloud storage (Amazon EC2, Amazon EBS & Amazon S3) on the VMware Cloud platform.



1. Bare metal Amazon Elastic Compute Cloud (Amazon EC2) instances running vSphere ESXI provide compute and VSAN flash storage for the workloads running on VMware Cloud on AWS.
2. NSX is the overlay network for VMware Cloud on AWS. It provides compute and management connectivity for workloads that run in the configured platform.
3. NSX Tier-0 router sends traffic from the compute & management gateways through the internet gateway for external connectivity.
4. Veeam backup server is deployed as a VM in the VMC cluster. It manages backups, backup job scheduling, resource allocation, recovery verification, restore tasks, and the backup infrastructure.
5. Veeam proxy server processes backup and restore jobs and delivers the backup traffic through the ENI to AWS storage repositories.
6. Elastic Network Interface (ENI) provides fast, low-latency connections between the SDDC and the Amazon Virtual Private Cloud (Amazon VPC). Backup traffic goes through the ENI to the backup repositories in the AWS Cloud.
7. Daily Veeam server backups are stored on Linux-based repository servers with Amazon Elastic Block Store (Amazon EBS) storage attachments in one Availability Zone in the VPC in the Region. Repositories are configured as *scale-out-backup* to allow data offload from the attached Amazon Elastic Block Storage (EBS) to object storage (S3) to optimize costs.
8. Amazon Simple Storage Service (Amazon S3) gateway endpoints provide private access to the storage gateway service and S3 buckets.
9. Send backups to customer-managed S3 buckets. Configure by setting policies on the scale-out-backup repository to move data to the S3 in the Region.
10. Replicate backup files offsite to an S3 bucket in another Region to store data in a different Region than your workloads (when required).

AWS Reference Architecture

Shared Repository

Common component for keeping data

Accessed by multiple other components

Most databases are shared repositories these days

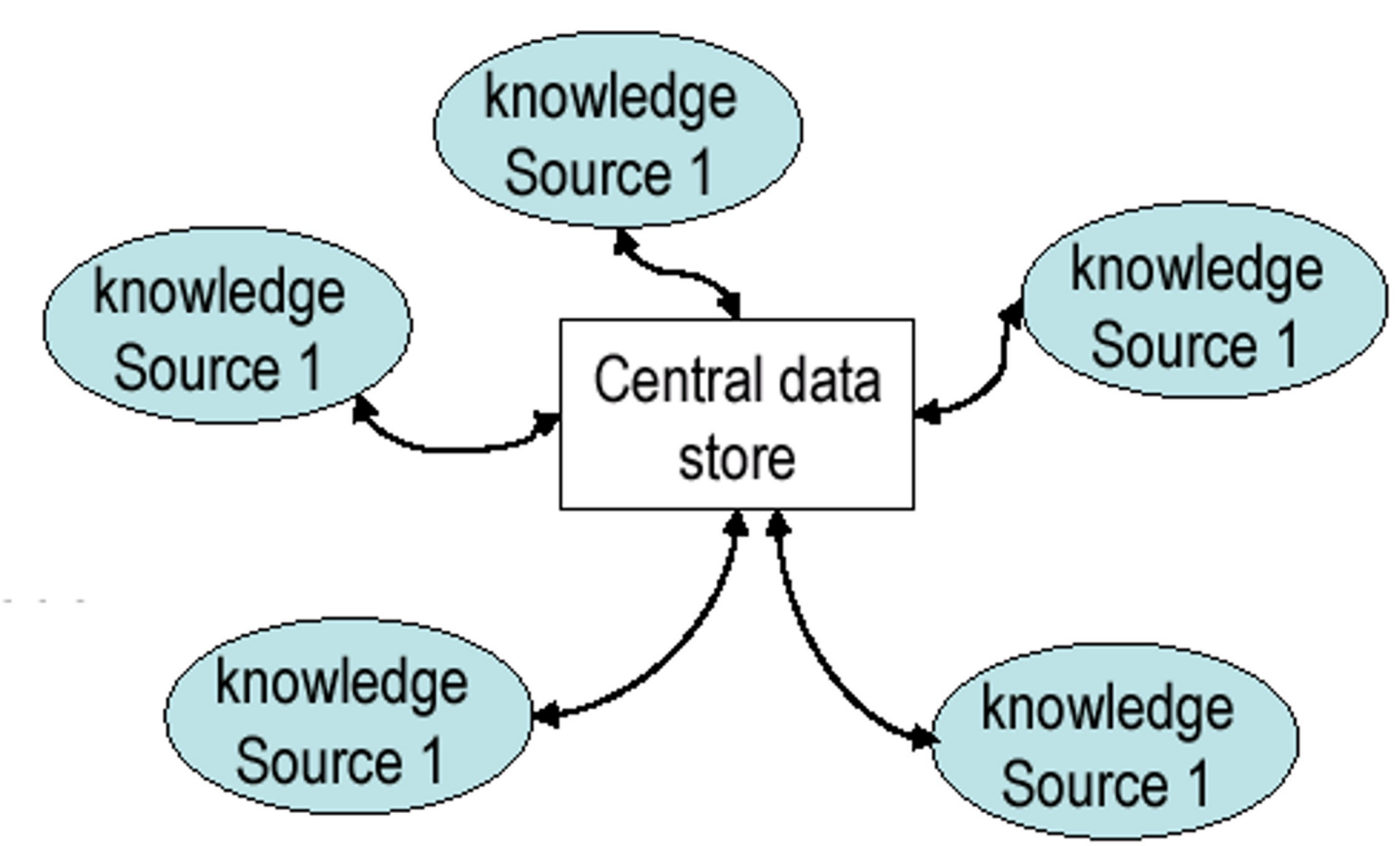
Considerations:

- Performance
- Concurrent access/update issues
- Reliability and availability (backups)

Variations:

- Active Repository
- Blackboard

Shared Repository



Active Repository

A repository (shared or not)

The repository does something other than just collect the data

- Maybe does some processing of the data it stores
- Maybe does some action based on the data

Blackboard

A Blackboard is a shared repository that uses the results of its clients for heuristic computation and step-wise improvement of the solution.

Consolidates/interprets independent inputs

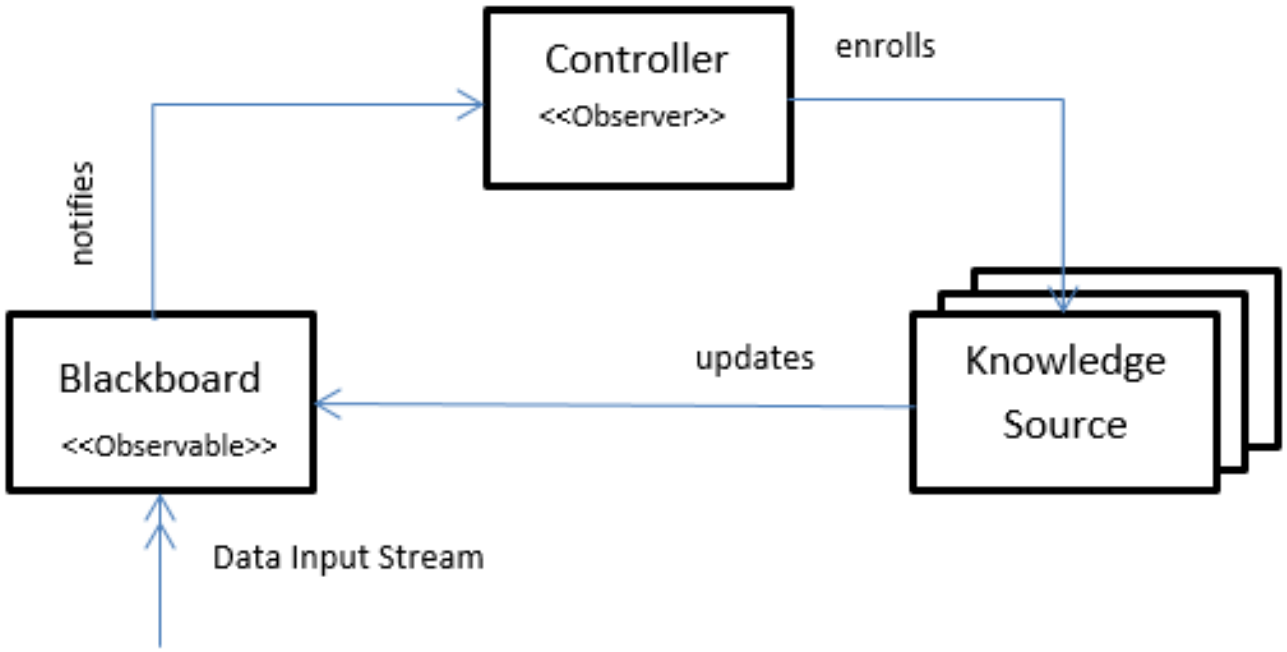
Structure:

- Shared repository with data processing capabilities
- Multiple sources of data; each can access the blackboard to check state.
- A controller/supervisor component monitors the blackboard and coordinates the clients according to the state of the blackboard.

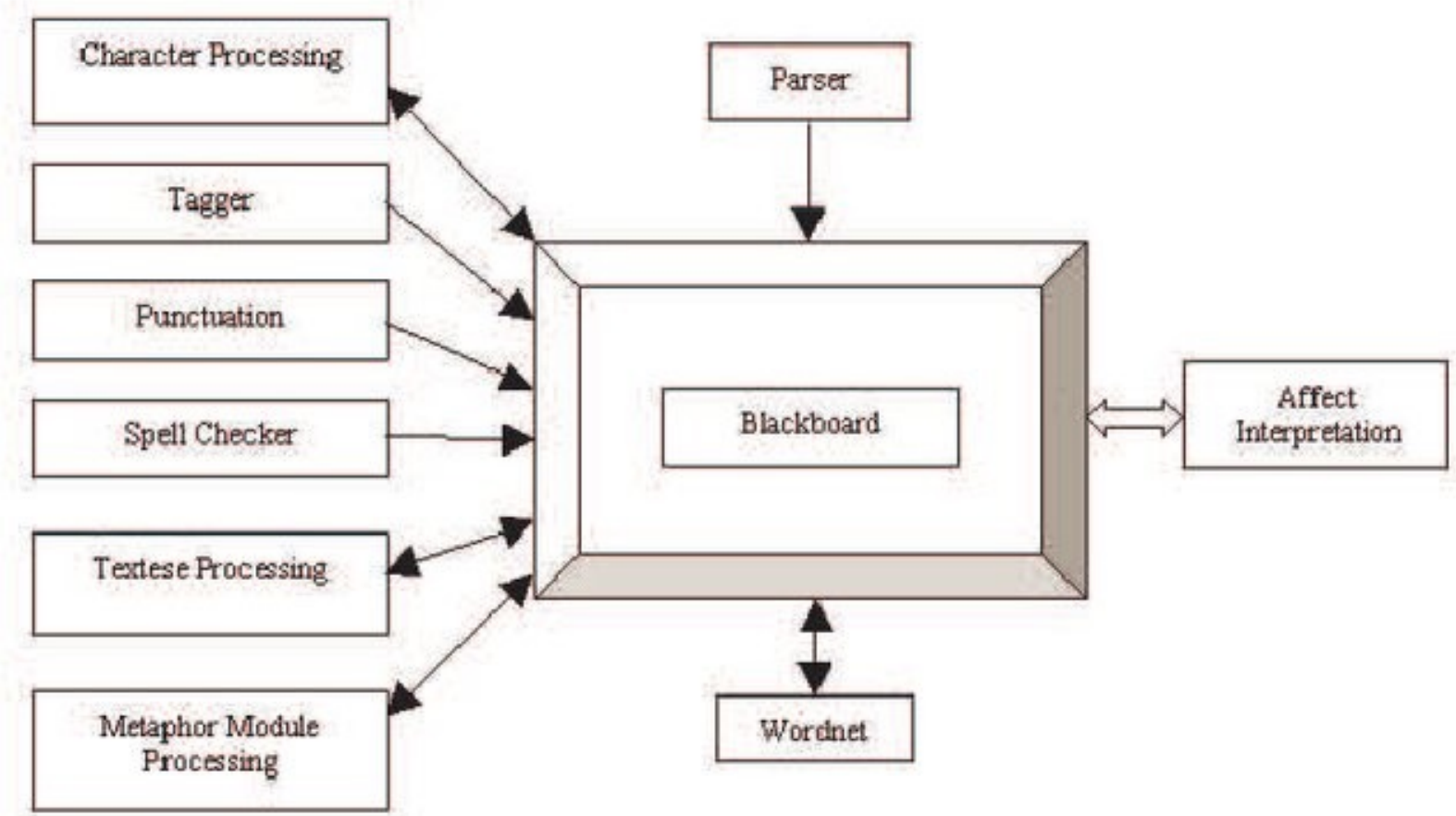
Highlights:

- Special purpose use: shared data is needed, but no deterministic solution strategies are known
- Examples: image or speech recognition
- May have challenges for debugging, enhancements

The Blackboard Pattern



The Blackboard Pattern



Rule-Based System

System consists of three things: facts, rules, and an engine that acts on them.

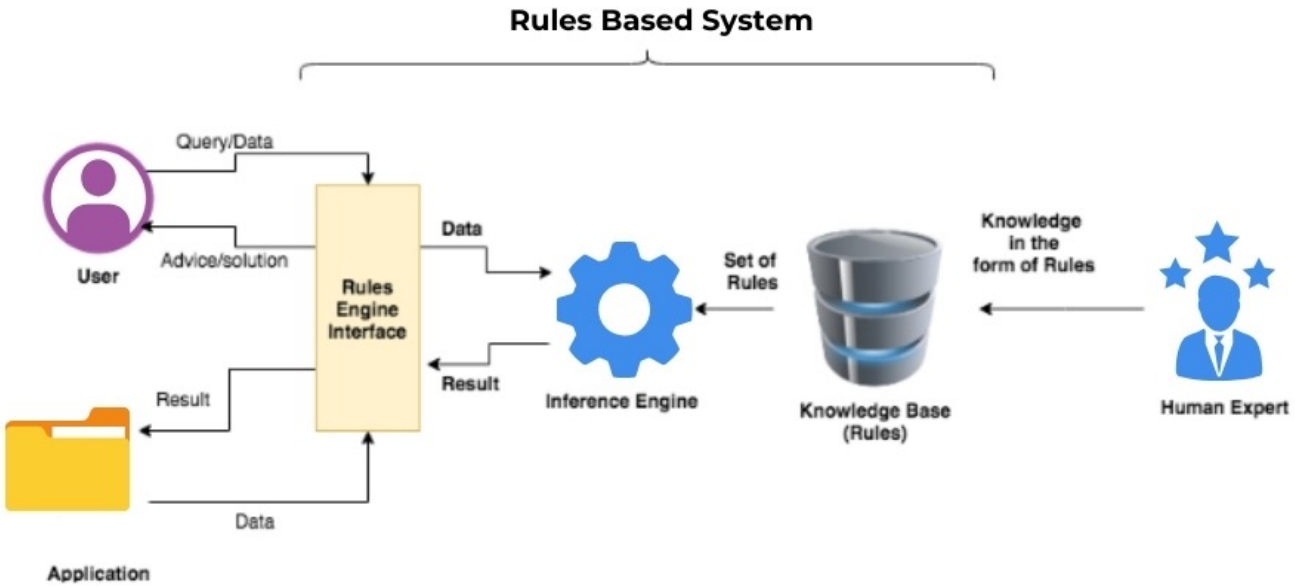
Rules are a condition and associated actions

Facts represent data

A rule-based system applies its rules to the known facts

The actions of a rule might assert new facts, which in turn trigger other rules

Rule-Based System Example



Model View Controller (MVC)

Managing the User Interface with 3 Components

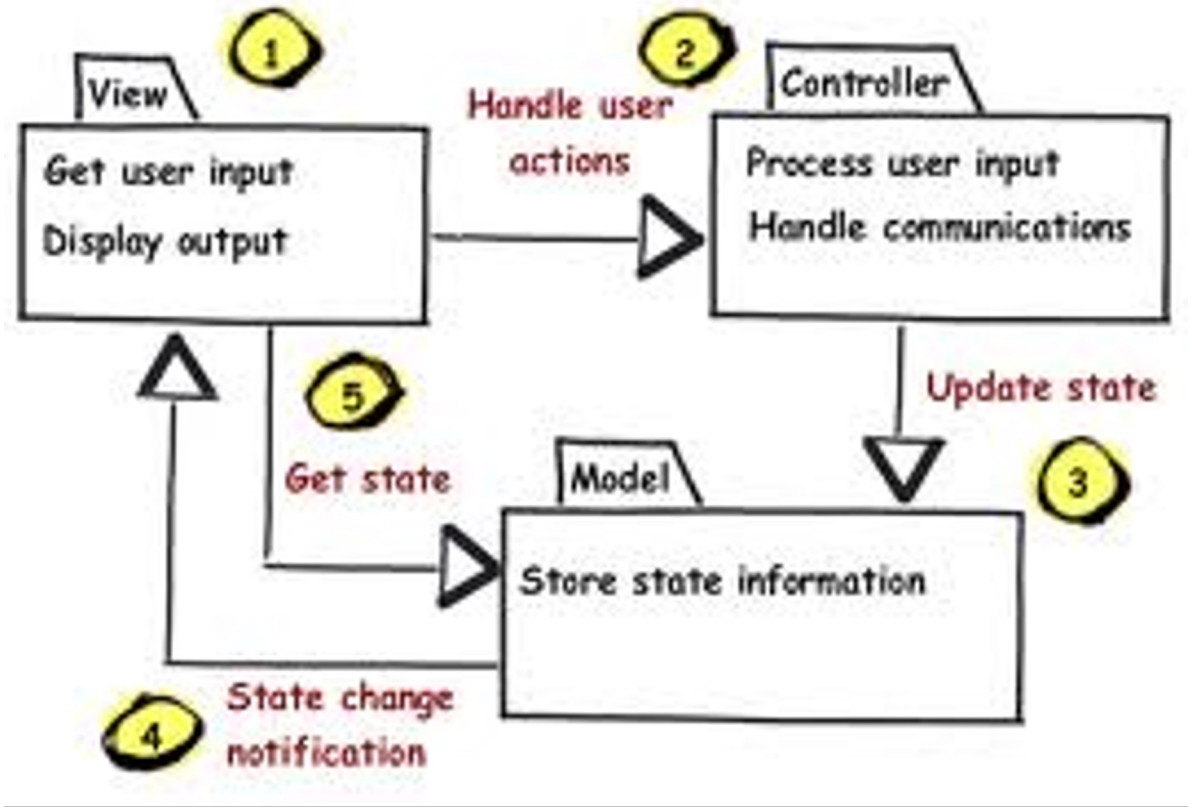
- Model: the main processing, e.g., implementation of the business rules
- View: the presentation to the user
- Controller: handles user actions and forwards them to the model

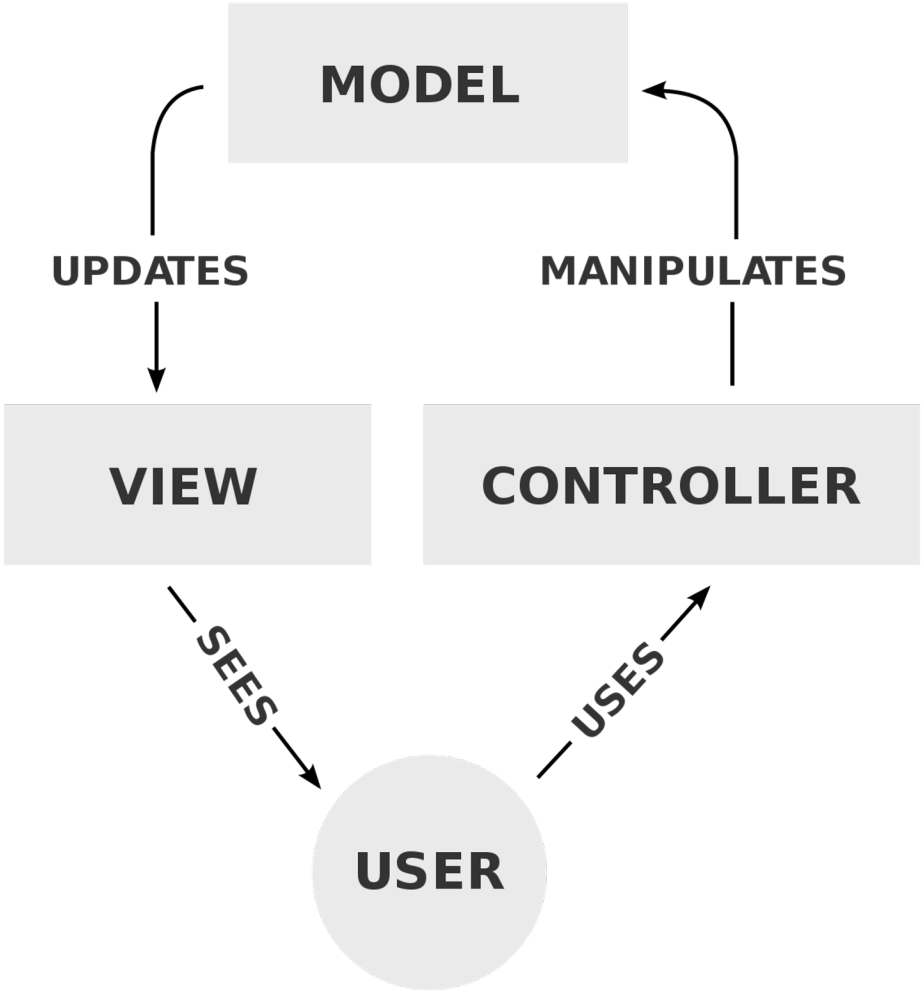
Separates the presentation from the processing

Very common these days

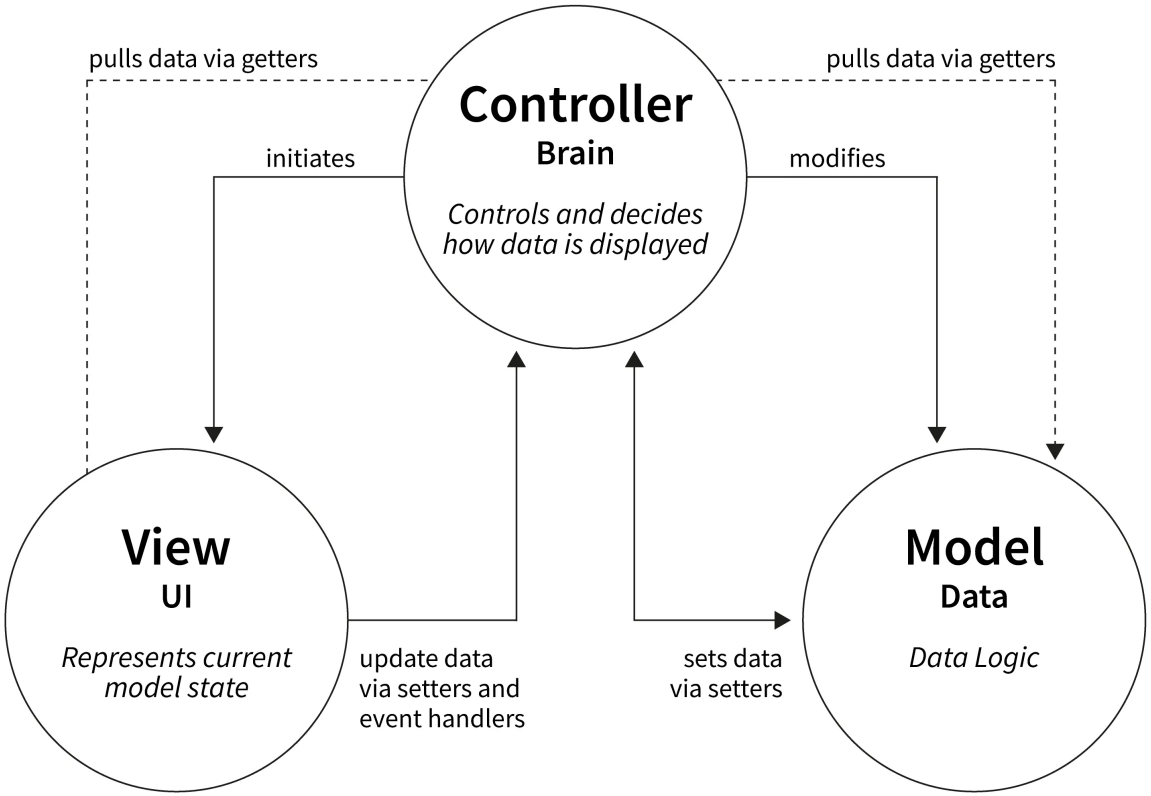
Some variants combine view and controller

Model View Controller




MODEL
UPDATES
MANIPULATES
VIEW
CONTROLLER
SEES
USES
USER

MVC Architecture Pattern



SCALER Topics

C2 (Components and Connectors)

Multiple components that are somewhat independent and heterogeneous

A top-to-bottom hierarchy of concurrent components that interact asynchronously by sending messages through explicit connectors

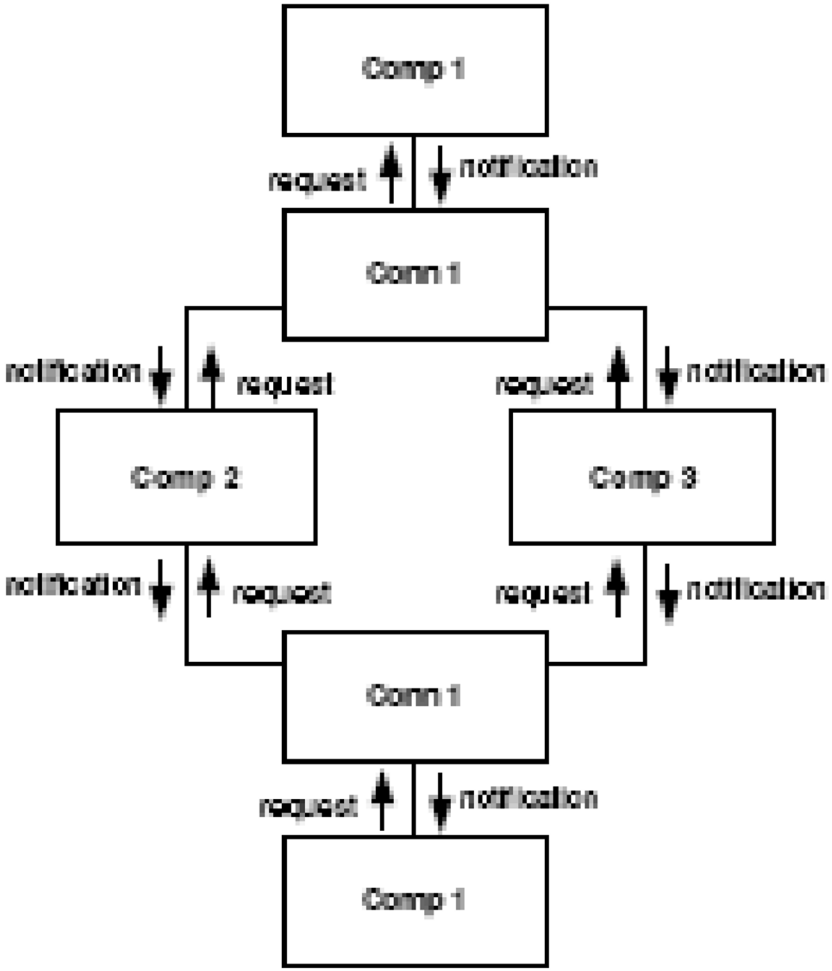
Components submit request messages upwards in the hierarchy, knowing the components above

They send notification messages down the hierarchy, without knowing the components below

Components are connected only with connectors

The connectors broadcast, route, and filter messages

C2


Comp 1
request
notification
Conn 1
notification
request
request
notification
Comp 2
Comp 3
notification
request
request
notification
Conn 1
request
notification
Comp 1

Example: video game

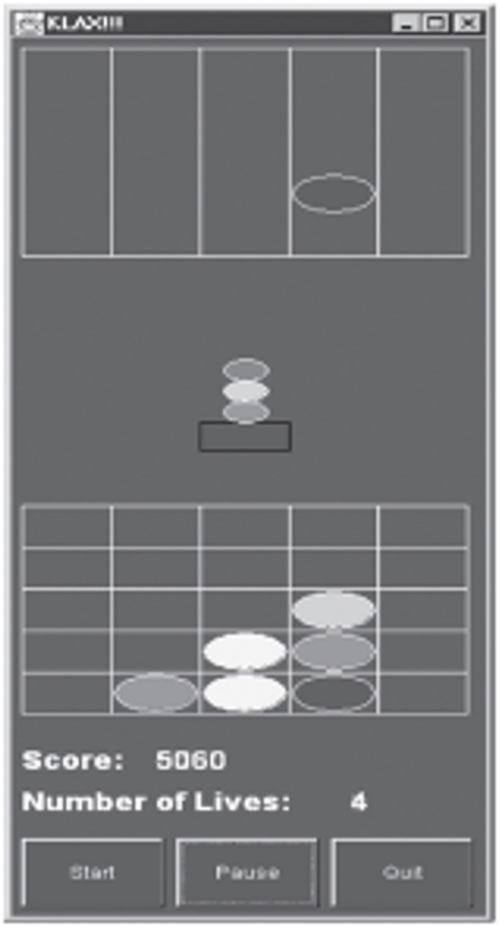
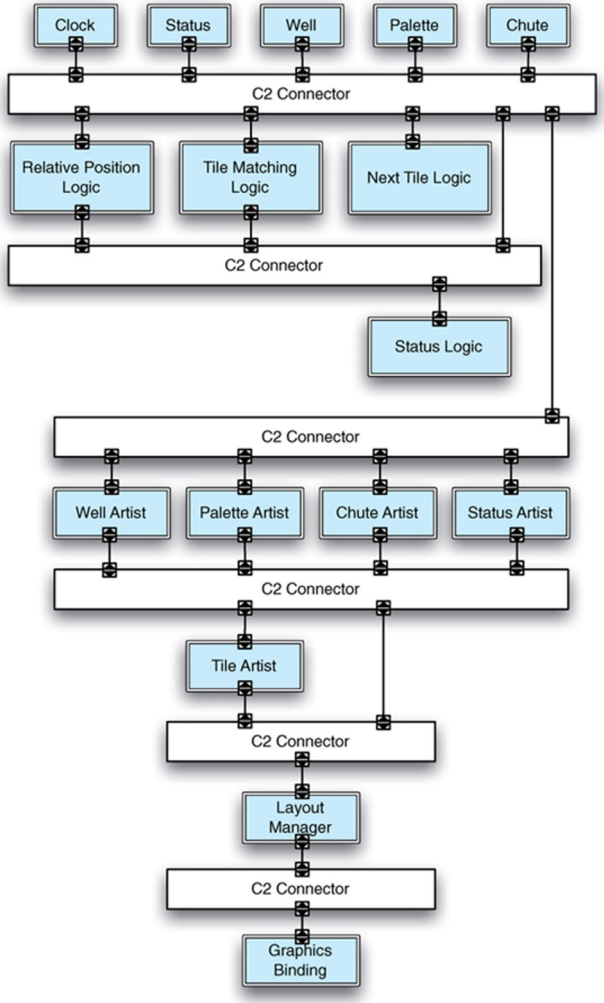
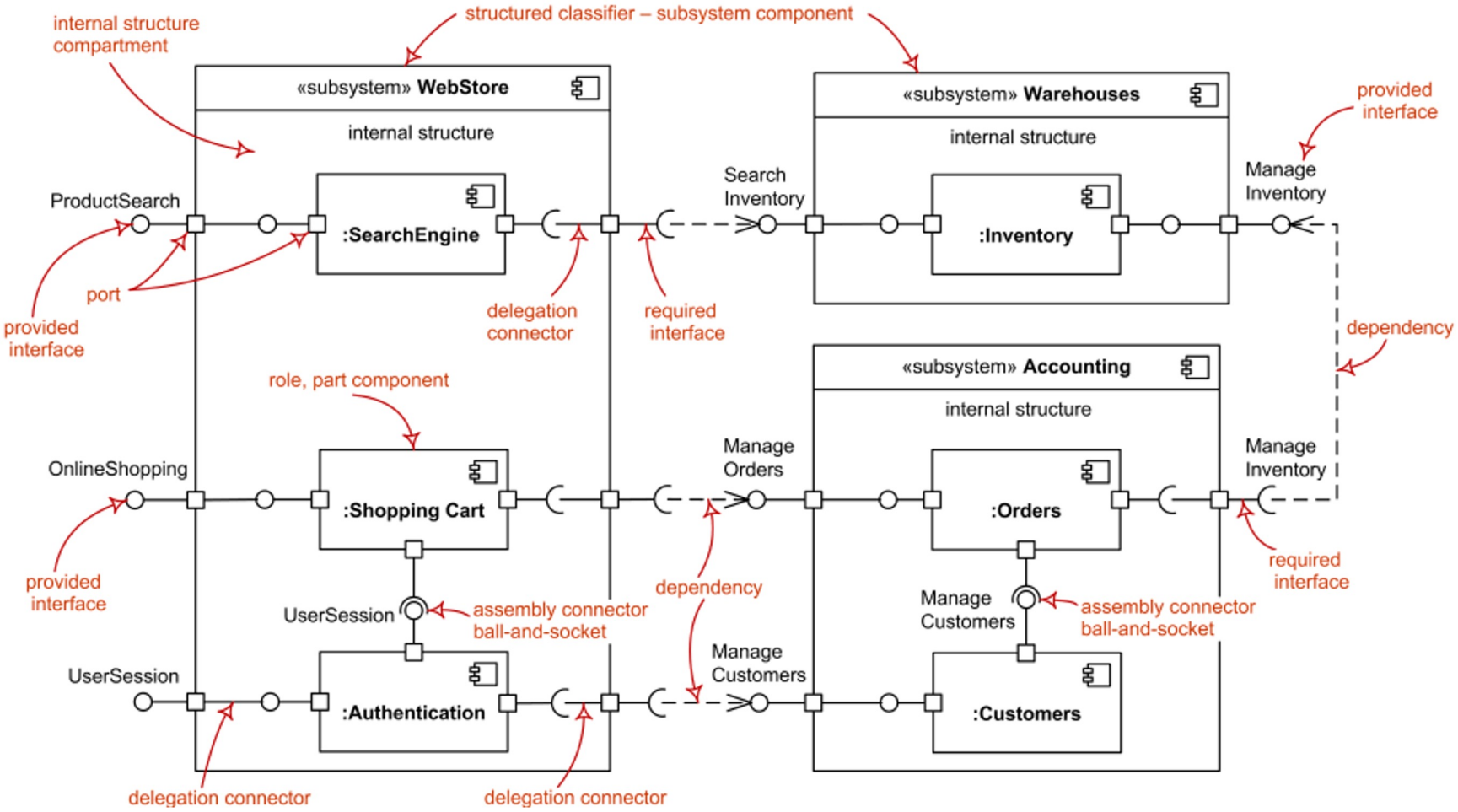


fig_04_24

Video Game: C2



C2


internal structure compartment
structured classifier – subsystem component
«subsystem» WebStore
internal structure
«subsystem» Warehouses
internal structure
provided interface
ProductSearch
:SearchEngine
Search Inventory
:Inventory
Manage Inventory
port
provided interface
delegation connector
required interface
dependency
«subsystem» Accounting
internal structure
role, part component
OnlineShopping
:Shopping Cart
Manage Orders
:Orders
Manage Inventory
provided interface
dependency
required interface
UserSession
assembly connector ball-and-socket
Manage Customers
assembly connector ball-and-socket
UserSession
:Authentication
Manage Customers
:Customers
delegation connector
delegation connector

Presentation Abstraction Control

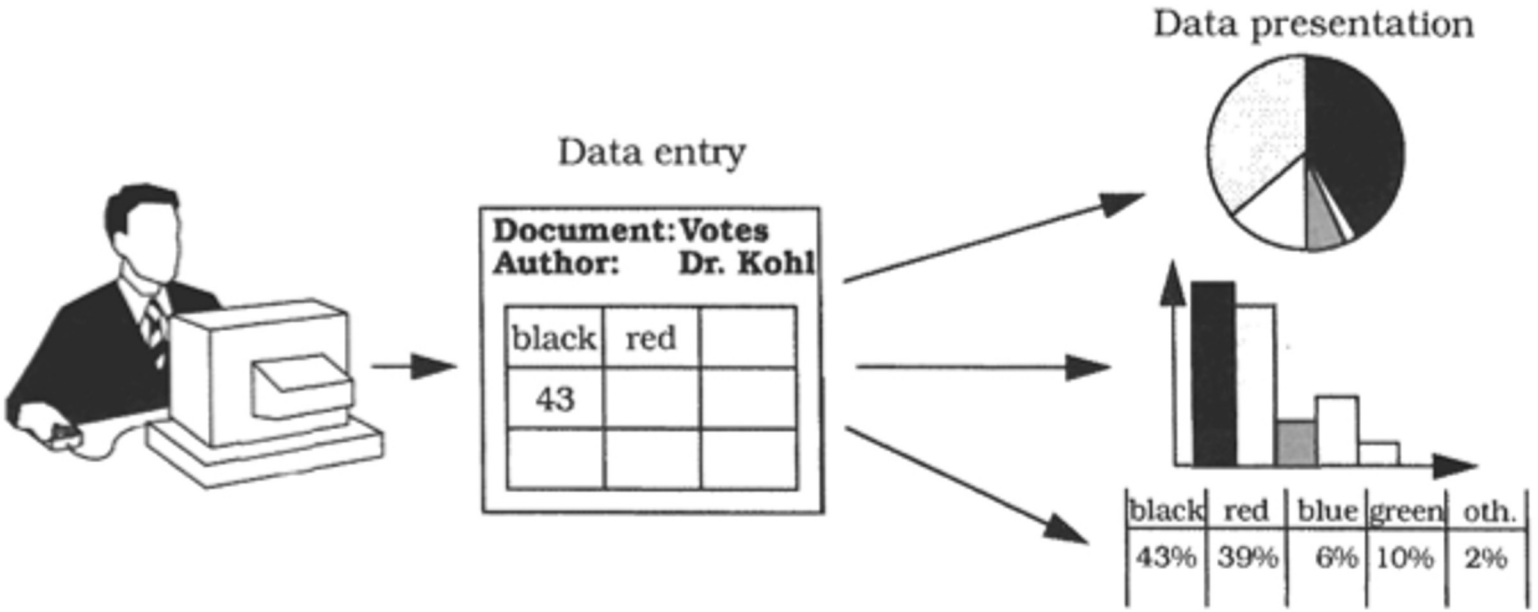
Alternative to MVC

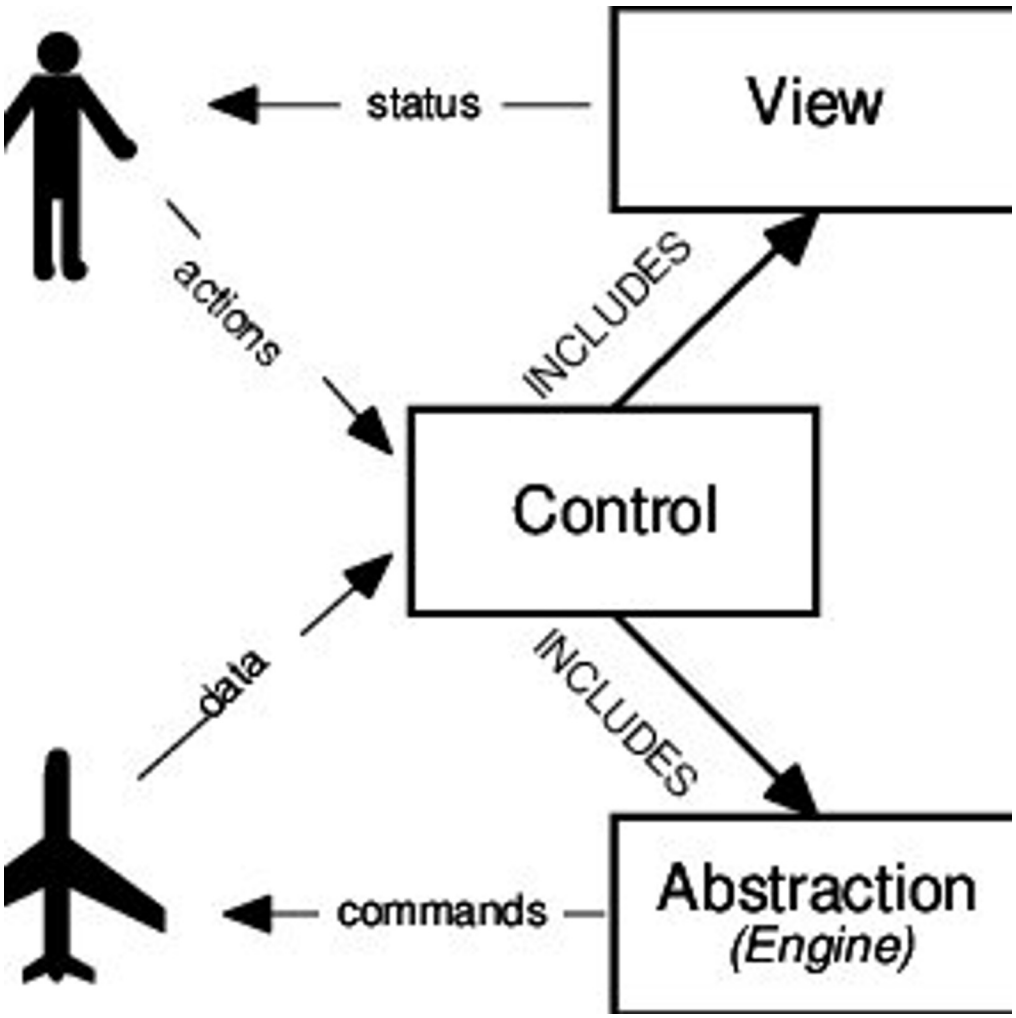
Modules with different abstractions and presentations of the data

Control: roughly equivalent to “model” and some of “control” in MVC

Handles different types of data presentations, usually simultaneously

Presentation Abstraction Control




status
View
actions
INCLUDES
Control
data
INCLUDES
commands
Abstraction
(Engine)

Relationships of patterns

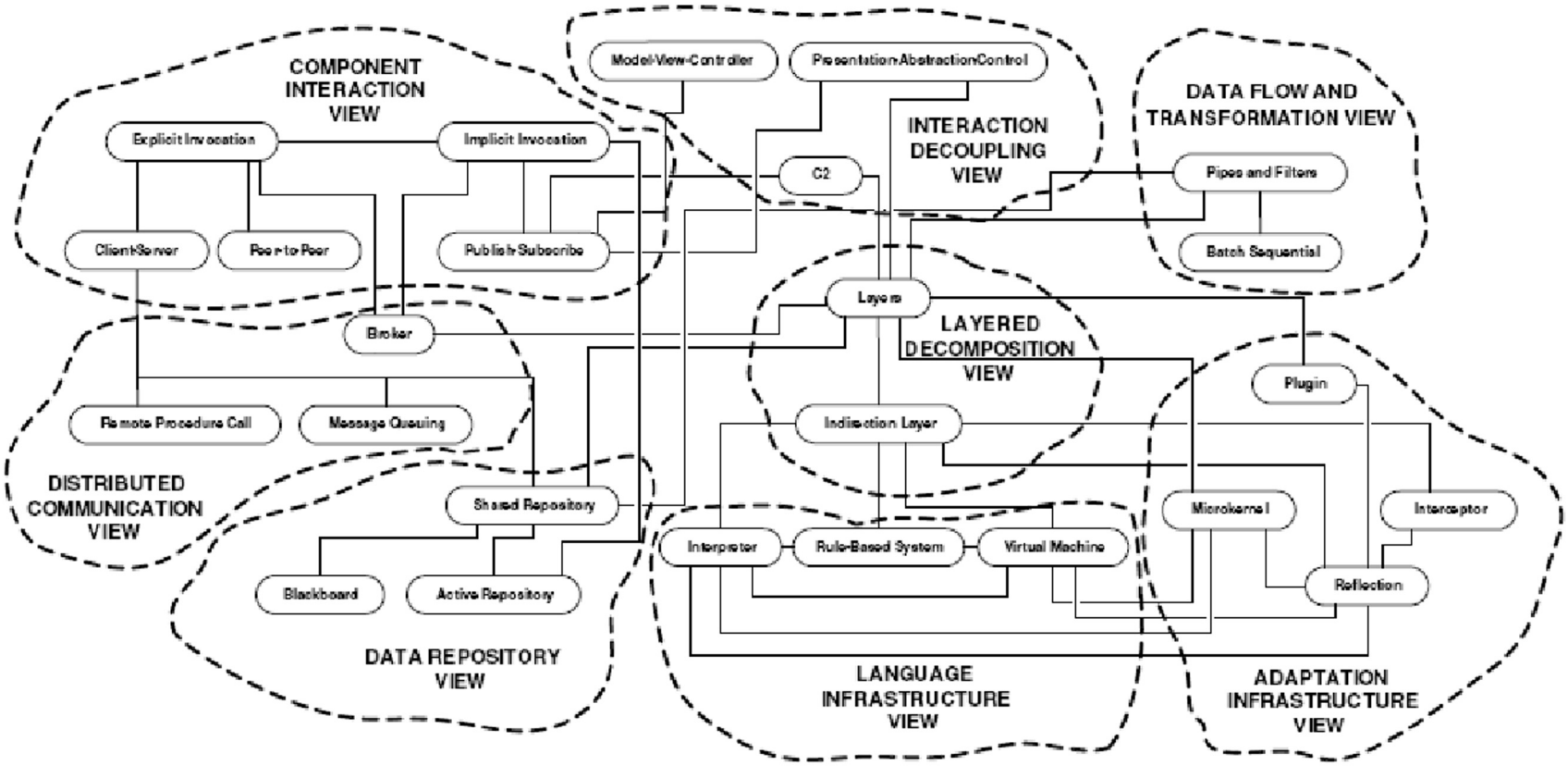


Figure 1: The patterns of the different views and the most significant pattern relationships

Table 4-1
A Comparison Table of Architectural Styles

Style Category & Name	Summary	Use It When . . .	Avoid It When . . .
Language-influenced styles			
Main program and subroutines	Main program controls program execution, calling multiple subroutines.	. . . application is small and simple.	. . . complex data structures needed (because of lack of encapsulation).
			. . . future modifications likely.
Object-oriented	Objects encapsulate state and accessing functions.	. . . close mapping between external entities and internal objects is sensible.	. . . application is distributed in a heterogeneous network.
		. . . many complex and interrelated data structures.	. . . strong independence between components necessary.
			. . . very high performance required.
Layered			
Virtual machines	Virtual machine, or a layer, offers services to layers above it.	. . . many applications can be based upon a single, common layer of services.	. . . many levels are required (causes inefficiency).
		. . . interface service specification resilient when implementation of a layer must change.	. . . data structures must be accessed from multiple layers.
Client-server	Clients request service from a server.	. . . centralization of computation and data at a single location (the server) promotes manageability and scalability.	. . . centrality presents a single-point-of-failure risk.
		. . . end-user processing limited to data entry and presentation.	. . . network bandwidth limited.
			. . . client machine capabilities rival or exceed the server's.
Dataflow Styles			
Batch-sequential	Separate programs executed sequentially, with batched input.	. . . problem easily formulated as a set of sequential, severable steps.	. . . interactivity or concurrency between components necessary or desirable.
			. . . random-access to data required.
Pipe-and-filter	Separate programs, aka filters, executed, potentially concurrently. Pipes route data streams between filters.	[as with batch-sequential] . . . filters are useful in more than one application.	. . . interaction between components required.
		. . . data structures easily serializable.	

Table 4-1
(Continued)

Style Category & Name	*Summary*	*Use It When . . .*	*Avoid It When . . .*
Shared Memory			
Blackboard	Independent programs, access and communicate exclusively through a global repository known as blackboard.	. . . all calculation centers on a common, changing data structure. . . . order of processing dynamically determined and data-driven.	. . . programs deal with independent parts of the common data. . . . interface to common data susceptible to change. . . . interactions between the independent programs require complex regulation.
Rule-based	Use facts or rules entered into the knowledge base to resolve a query.	. . . problem data and queries expressible as simple rules over which inference may be performed.	. . . number of rules is large. . . . interaction between rules present. . . . high-performance required.
Interpreter			
Interpreter	Interpreter parses and executes the input stream, updating the state maintained by the interpreter.	. . . highly dynamic behavior required. High degree of end-user customizability.	. . . high performance required.
Mobile code	Code is mobile, that is, it is executed in a remote host.	. . . it is more efficient to move processing to a data set than the data set to processing. . . . it is desirous to dynamically customize a local processing node through inclusion of external code.	. . . security of mobile code cannot be assured, or sandboxed. . . . tight control of versions of deployed software is required.
Implicit Invocation			
Publish-subscribe	Publishers broadcast messages to subscribers.	. . . components are very loosely coupled. . . . subscription data is small and efficiently transported.	. . . middleware to support high-volume data is unavailable.
Event-based	Independent components asynchronously emit and receive events communicated over event buses.	. . . components are concurrent and independent. . . . components heterogeneous and network-distributed.	. . . guarantees on real-time processing of events is required.

(Continued)

Table 4-1
(Continued)

Style Category & Name	Summary	Use It When . . .	Avoid It When . . .
Peer-to-Peer			
	Peers hold state and behavior and can act as both clients and servers.	. . . peers are distributed in a network, can be heterogeneous, and mutually independent.	. . . trustworthiness of independent peers cannot be assured or managed.
		. . . robustness in face of independent failures and high scalability required.	. . . designated nodes to support resource discovery unavailable.
More Complex Styles			
C2	Layered network of concurrent components communicating by events.	. . . independence from substrate technologies required.	. . . high-performance across many layers required.
		. . . heterogeneous applications.	. . . multiple threads are inefficient.
		. . . support for product-lines desired.	
Distributed objects	Objects instantiated on different hosts.	. . . objective is to preserve illusion of location-transparency.	. . . high overhead of supporting middleware is excessive.
			. . . network properties are unmaskable, in practical terms.

table_04_01c

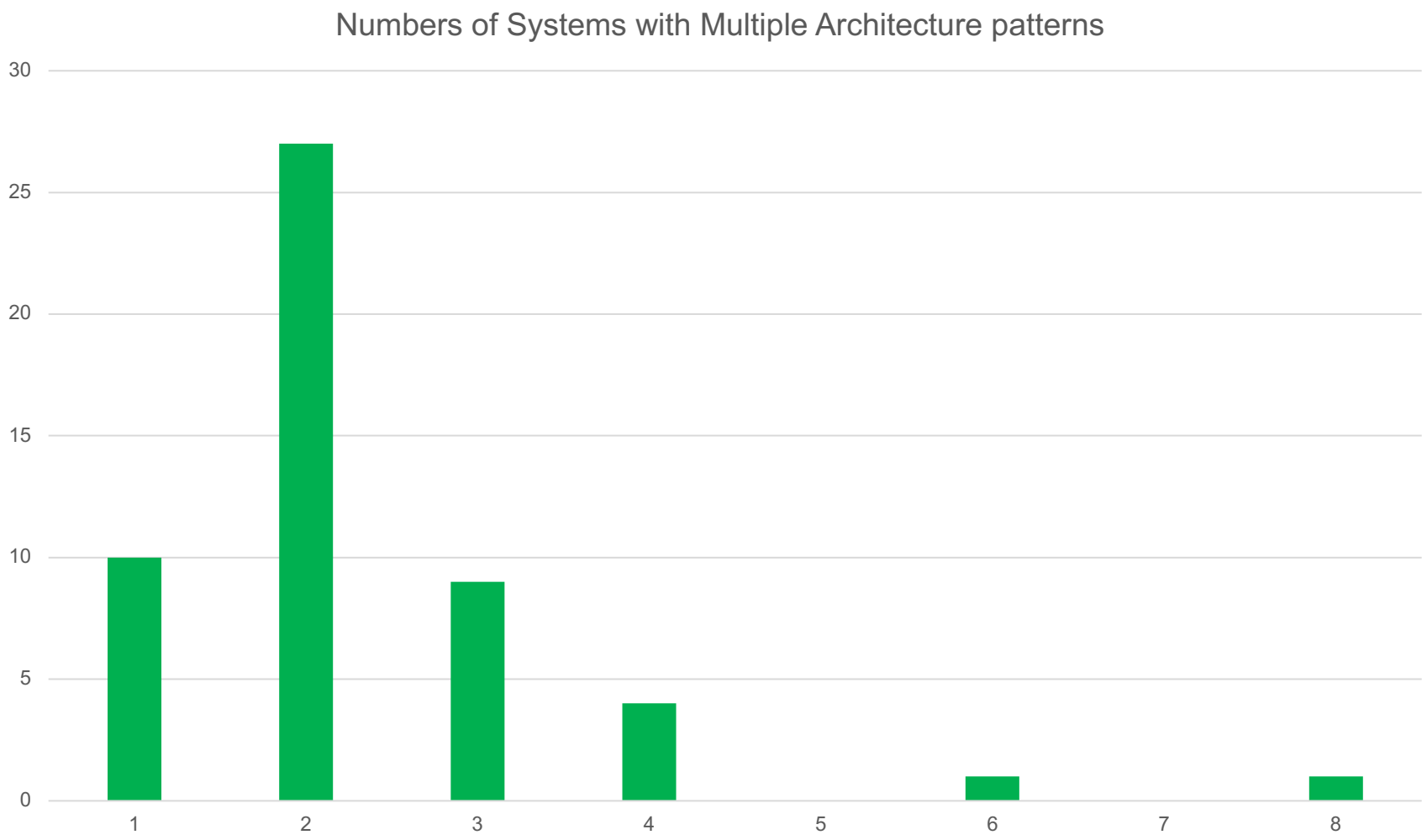
Comparison of some patterns

Name	Advantages	Disadvantages
Layered	A lower layer can be used by different higher layers. Layers make standardization easier as we can clearly define levels. Changes can be made within the layer without affecting other layers.	Not universally applicable. Certain layers may have to be skipped in certain situations.
Client-server	Good to model a set of services where clients can request them.	Requests are typically handled in separate threads on the server. Inter-process communication causes overhead as different clients have different representations.
Master-slave	Accuracy - The execution of a service is delegated to different slaves, with different implementations.	The slaves are isolated: there is no shared state. The latency in the master-slave communication can be an issue, for instance in real-time systems. This pattern can only be applied to a problem that can be decomposed.
Pipe-filter	Exhibits concurrent processing. When input and output consist of streams, and filters start computing when they receive data. Easy to add filters. The system can be extended easily. Filters are reusable. Can build different pipelines by recombining a given set of filters	Efficiency is limited by the slowest filter process. Data-transformation overhead when moving from one filter to another.
Broker	Allows dynamic change, addition, deletion and relocation of objects, and it makes distribution transparent to the developer.	Requires standardization of service descriptions.
Peer-to-peer	Supports decentralized computing. Highly robust in the failure of any given node. Highly scalable in terms of resources and computing power.	There is no guarantee about quality of service, as nodes cooperate voluntarily. Security is difficult to be guaranteed. Performance depends on the number of nodes.
Event-bus	New publishers, subscribers and connections can be added easily. Effective for highly distributed applications.	Scalability may be a problem, as all messages travel through the same event bus
Model-view-controller	Makes it easy to have multiple views of the same model, which can be connected and disconnected at run-time.	Increases complexity. May lead to many unnecessary updates for user actions.
Blackboard	Easy to add new applications. Extending the structure of the data space is easy.	Modifying the structure of the data space is hard, as all applications are affected. May need synchronization and access control.
Interpreter	Highly dynamic behavior is possible. Good for end user programmability. Enhances flexibility, because replacing an interpreted program is easy.	Because an interpreted language is generally slower than a compiled one, performance may be an issue.

Architecture Pattern Usage (legacy)

Pattern Name	Frequency
Layers	18
Shared Repository	13
Pipes and Filters	10
Client-Server	10
Broker	9
Model-View-Controller	7
Presentation-Abstraction- Control	7
Explicit Invocation	4
Plug-in	4
Blackboard	4
Microkernel	3
Peer to Peer	3
C2	3
Publish-Subscribe	3
State Transition	3
Interpreter	2
Half Sync, Half Async	2
Active Repository	2
Interceptor	2
Remote Procedure Call	1
Implicit Invocation	1

Nearly every non-trivial system has multiple architecture patterns



Frequency of pattern pairs

Pattern Pair	Frequency
Layers – Broker	6
Layers – Shared Repository	3
Pipes & Filters – Blackboard	3
Client Server – Presentation Abstraction Control	3
Layers – Presentation Abstraction Control	3
Layers – Model View Controller	3
Broker – Client-Server	2
Shared Repository – Presentation Abstraction Control	2
Layers – Microkernel	2
Shared Repository – Model View Controller	2
Client Server – Peer to Peer	2
Shared Repository – Peer to Peer	2
Shared Repository – C2	2
Peer to Peer – C2	2
Layers – Interpreter	2
Layers – Client Server	2
Pipes & Filters – Client Server	2
Pipes & Filters – Shared Repository	2
Client Server – Blackboard	2
Broker – Shared Repository	2
Broker – Half Sync/Half Async	2
Shared Repository – Half Sync/Half Async	2
Client Server – Half Sync/Half Async	2

Hunting Patterns in the Wild

Patterns as they appear in architectures

Finding Architecture Patterns

Are architecture patterns really used?

Yes!

- In a study of 45 architectures, we found architecture patterns in every architecture
- And in many cases, it is likely the architects didn't even use them on purpose

Studying architectures can help us see how to use architecture patterns

Boxology: a Field Guide

Most architecture diagrams consist of:

- Boxes
- Lines between the boxes
- Words, usually labeling the boxes and lines

They usually follow no standard methodology

- We are generally left to figure out the meanings of the boxes and lines
- They may mean different things – in the same diagram!

But it usually works pretty well

Exercise: Find the Patterns

You will see pictures of real architectures

Can you identify any patterns in them?

Some architectures may have more than one pattern

- (our experience: nearly all had two or more patterns)

Break into groups and discuss each

- Then we will discuss together

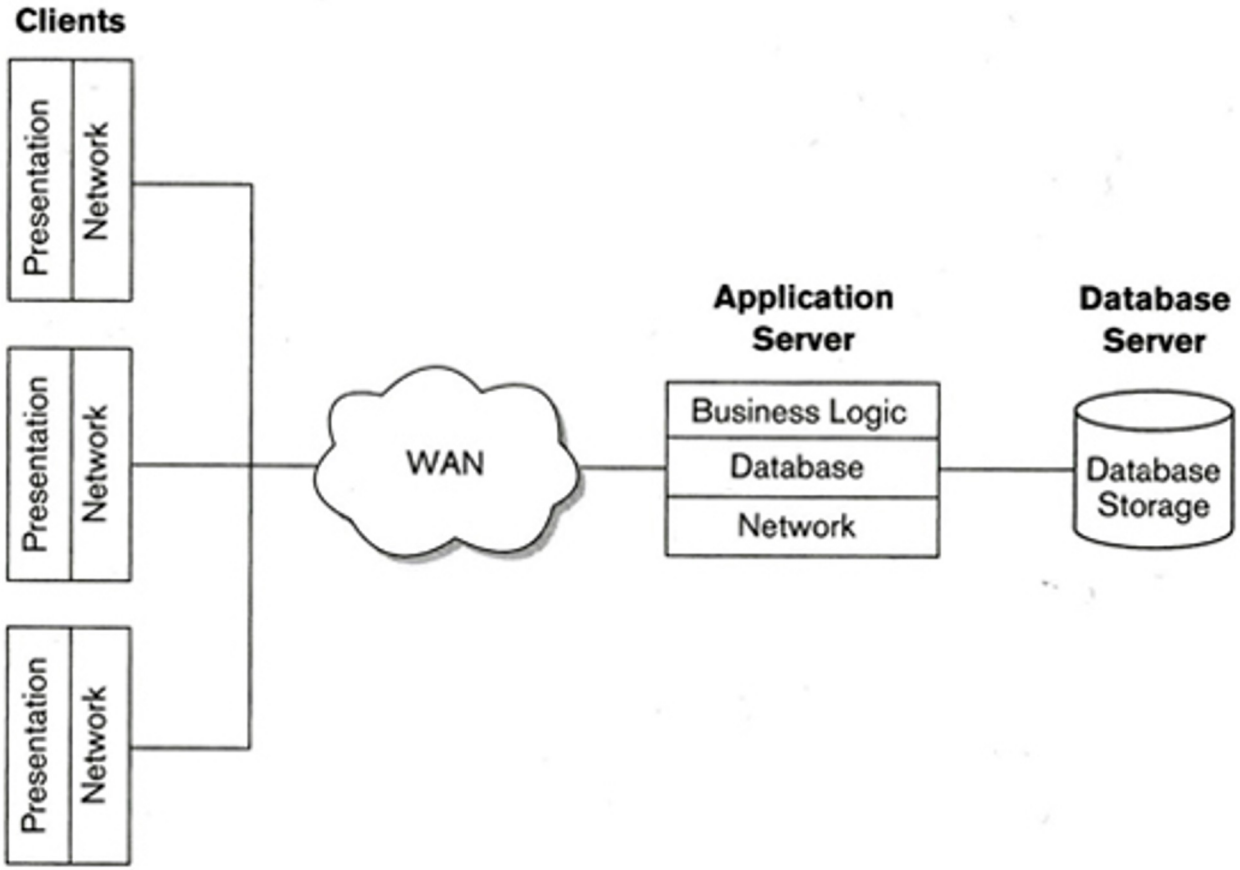
Airline Booking System

Client Server

Layers

Shared Repository (maybe)

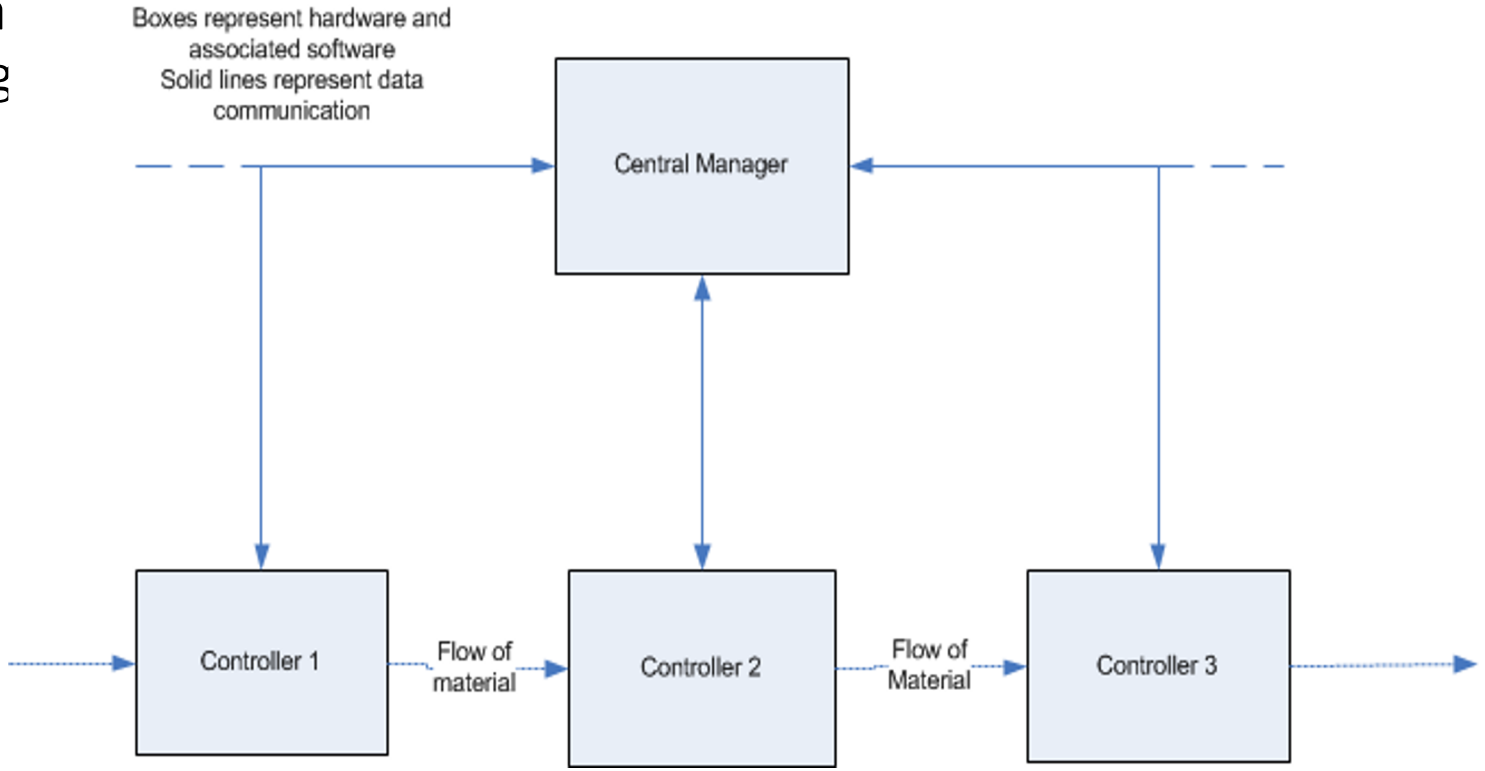
Presentation Abstraction Control (maybe)



Materials Processing System

Pipes and Filters

What is the centra' manag

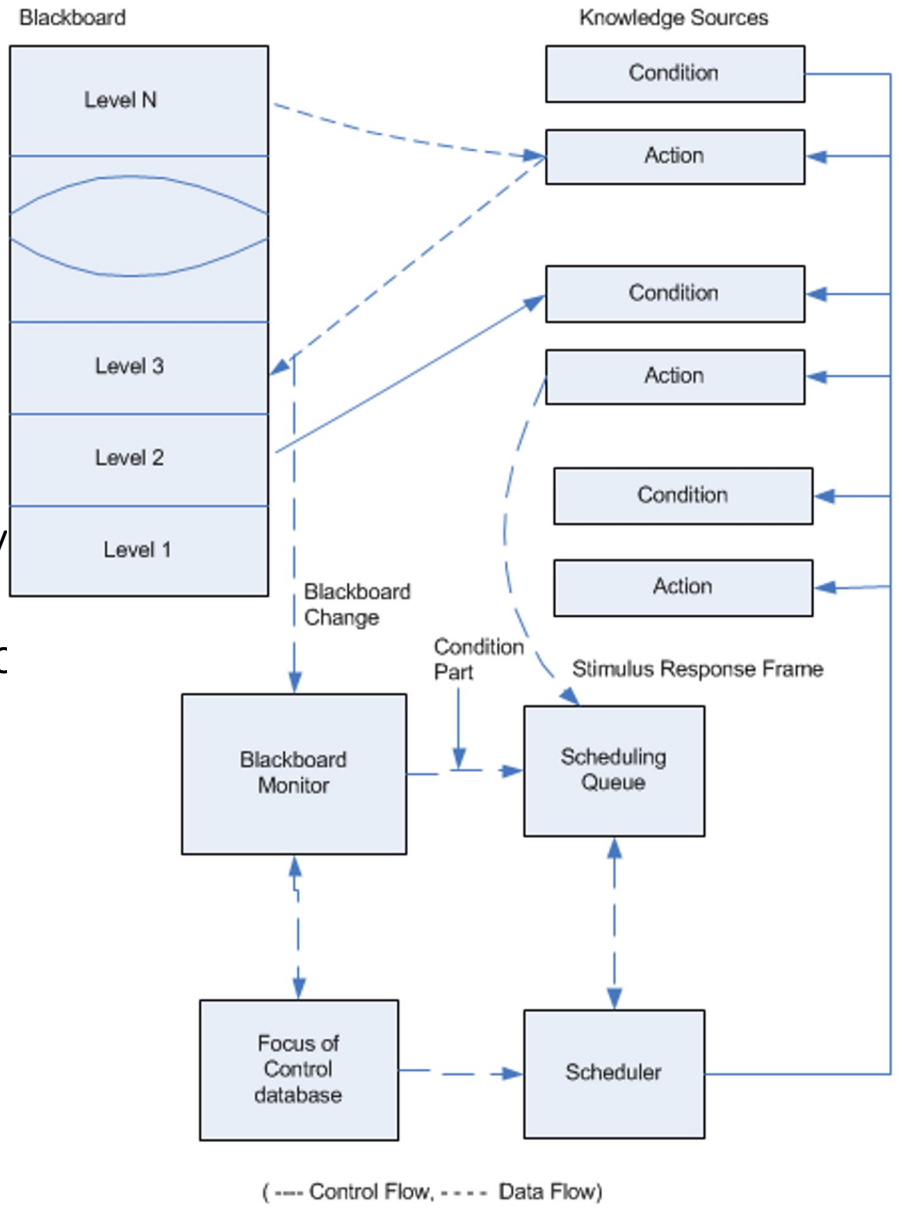


Speech Recognition

Blackboard

Layers?

- It looks like it, but they don't appear to communicate with eac other

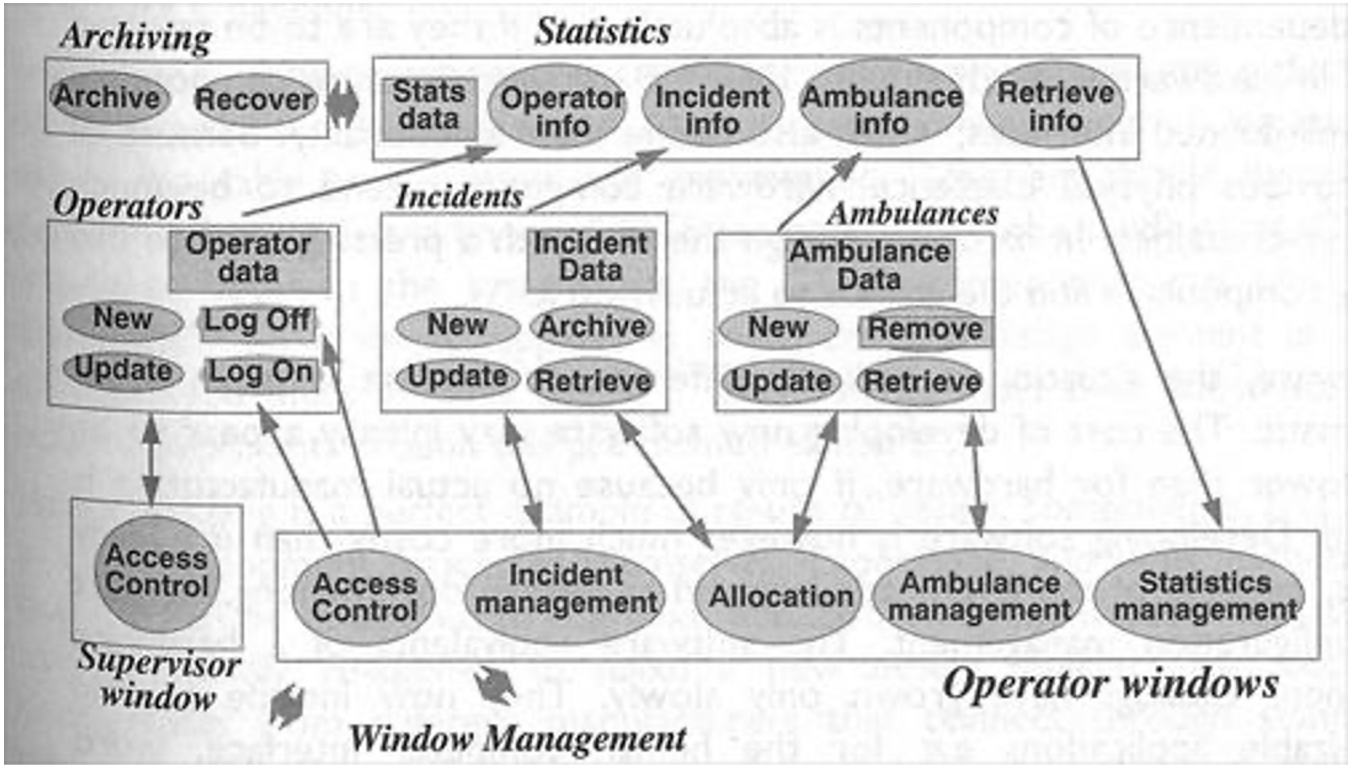


Ambulance Management System

Presentation Abstraction Control (different operator windows)

Shared Repository (statistics box)

- It is also active? Maybe, but probably not.



Google

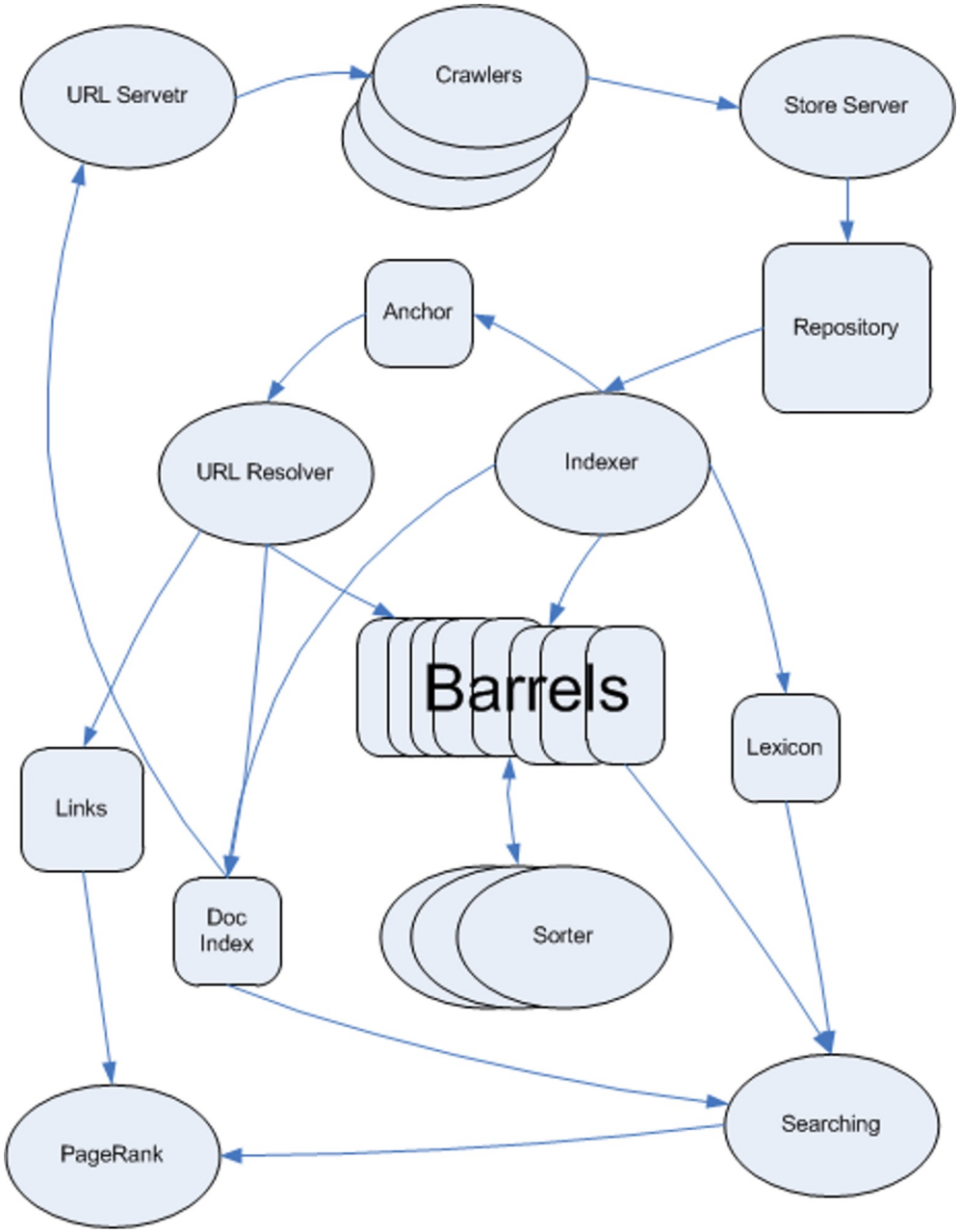
Repository
- Shared?

Client-Server?

Broker?

Any Layers?

What are the Barrels?
- Maybe a kind of Blackboard



Lessons Learned

Nearly all software architectures use architecture patterns

Architects don’t always use the patterns knowingly

Sometimes patterns are obvious

Sometimes not

Patterns, even accidental ones, have consequences

- More about this later

Lessons Learned, cont.

What if we got the patterns “wrong?”

- Not what the architect intended?

One way to think of the patterns is that they describe the partitioning of the system.

- Therefore, if a pattern does describe the system’s partitioning, then it doesn’t really matter if the architect intended it or not.
- In fact, many system architectures predate the notion of architecture patterns – and we still find the patterns