BITS Pilani

# Software Architecture
# Architecture Patterns #1

Vijayarajan A

# Contents

- Recap
- Architecture Patterns - 01

# Assignment & Quiz 1

**A1 - Build an Architecture for an App**

- The App should at minimum include the technologies Web, Mobile, IOT, cloud and analytics.
- Each team will select an application and get the approval of the TA.
- Duplication May not be allowed …
- The submitted architecture will be evaluated by the TA
- The team will be evaluated for their developed architecture

**A2 – Research paper on real life architecture / latest trends etc**

- Each team has to select a topic and get the approval of the faculty.
- Duplication may not be allowe
- Final Paper should be submitt greed upon template and schedule

Quiz 1 **Ignore this Quiz related information… check the portals**

Sunday 14th Feb - late evening after 6 PM; Block 45 minutes; 25 questions

# Evolution of SW Architecture

We design and implement information systems to solve problems and process data.

As problems become larger and more complex and data becomes more voluminous, so do the associated information systems

- Structured programming, Data Structure, Higher Level languages, software engineering, Object Oriented etc

Computing become Distributed, on the cloud, Mobile as a front end

As the problem size and complexity increase, algorithms and data structures become less important than getting the right structure for the information system.

Specifying the right structure of the information system becomes a critical design problem itself

< Example from Construction Industry>

# Importance of Quality attributes & Tactics

- Functional requirements help us to define the modules
- Quality attributes help us to **structure** the system

Availability

Modifiability

Performance

Security

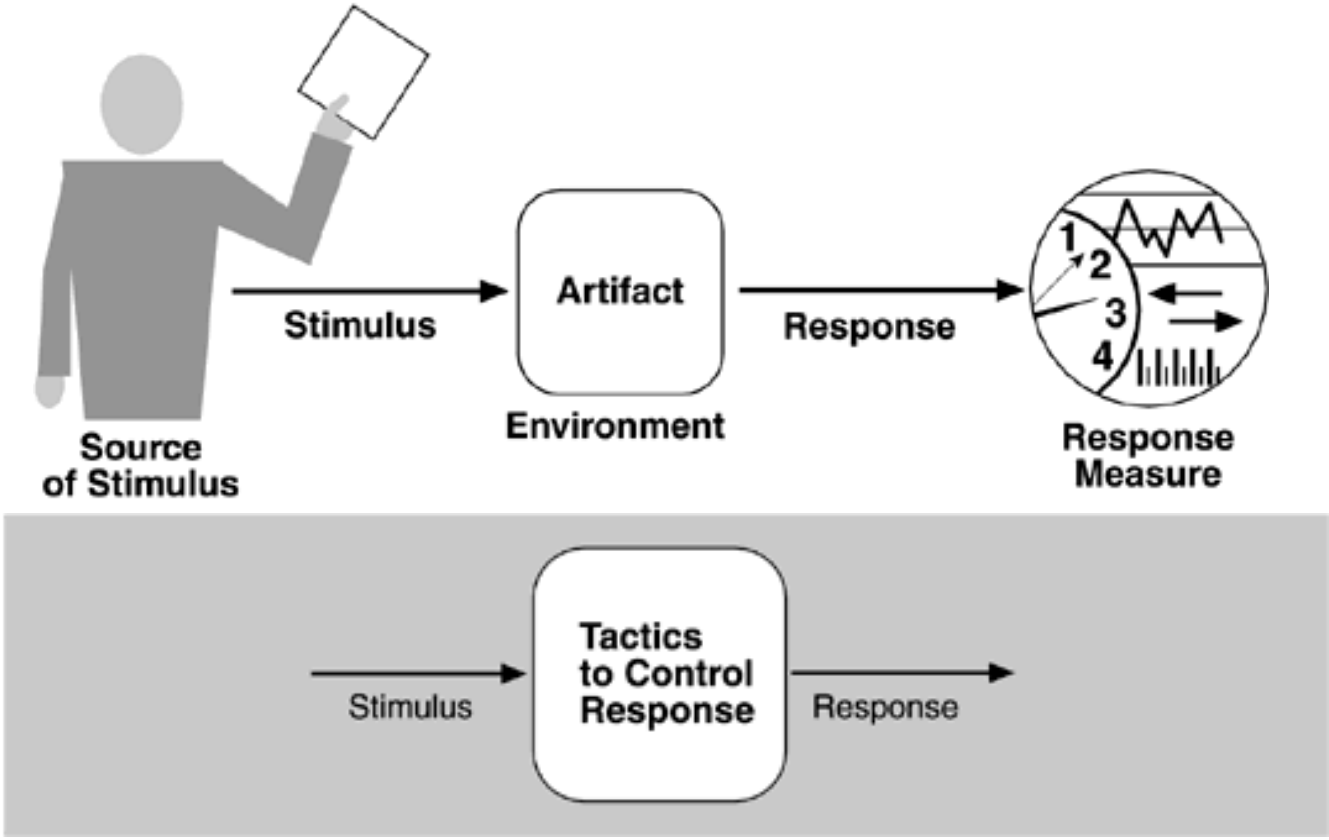
Usability

Interoperability

Scalability

Testability

Architecturally significant requirements



# Power Screwdrivers

Slotted


**Context:** power screwdrivers

**Problem:** don't work well with ordinary slot screws

- time wasted in trying to fit the screwdriver into the slot;
- once you succeed, centrifugal force tends to make the bit slide off the screw and into the workbench

**Solution:** Phillips Screwdrivers

Phillips


Building Construction Example …

# What is a Pattern

1. Addresses a recurring design problem
2. Documents existing, well proven design experience
3. Pattern identify and specify abstractions that are above the level of single classes and instances or of components

   Typically, a pattern describes several components, classes or objects, and details their **responsibilities** and relationships, as well as their **cooperation**.

   All components together solve the problem more effectively that the pattern addresses

Because patterns are (by definition) found repeatedly in practice, one does not invent them; one discovers them.

# What is a Pattern? Formal Definition

An architectural pattern establishes a relationship between:

*A context*. A recurring, common situation in the world that gives rise to a problem.

*A problem.* The problem, appropriately generalized, that arises in the given context.

*A solution.* A successful architectural resolution to the problem, appropriately abstracted. The solution for a pattern is determined and described by:

- A set of element types (for example, data repositories, processes, and objects)
- A set of interaction mechanisms (ex: method calls, events, or message bus)
- A topological layout of the components
- A set of semantic constraints covering topology, element behavior, and interaction mechanisms

Typically Represented as

- Overview
- Elements
- Relationship
- Constraints
- Weakness

# Scope of the Lectures

- We will not discuss the Patterns in the formal definition / representation format
- That are covered in the book

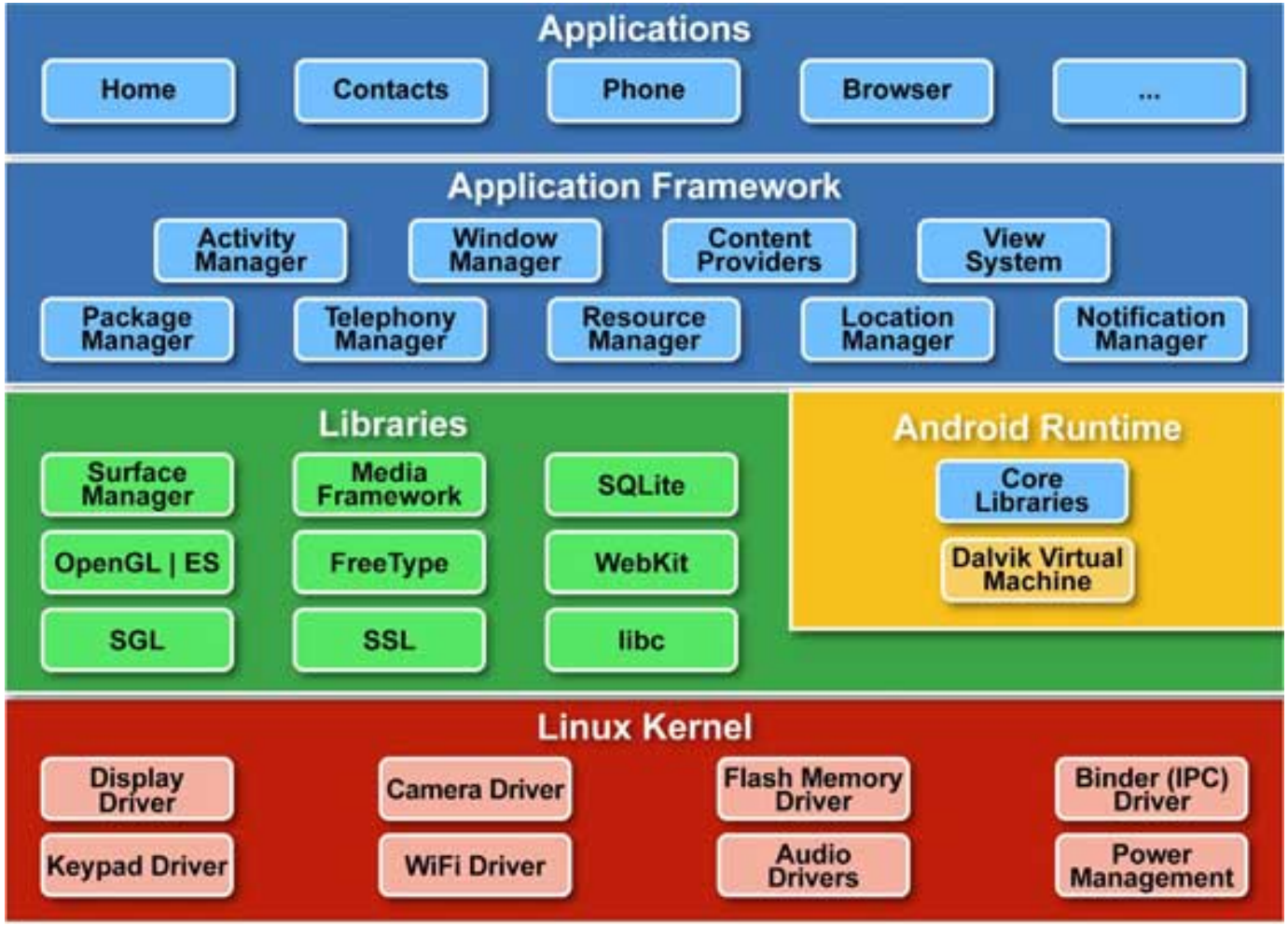
# Patterns & Tactics

- Tactics typically use just a single structure or computational mechanism, and they are meant to address a single architectural force.
- Patterns typically combine multiple design decisions into a package. Tactics are the “building blocks” of design, from which architectural patterns are created.
- Tactics are atoms and patterns are molecules.
- Most patterns consist of (are constructed from) several different tactics.
- Patterns package tactics.

# List of patterns

1. Layer
2. Pipe & Filter
3. MVC
4. Publish & Subscribe
5. Client & Server
6. P2P
7. Shared Data
8. Broker
9. Map-Reduce
10. Multi-tier
11. SOA

# Layered pattern
# Example: Architecture of Android




http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture

# Layer Pattern

**Context:**

Complex systems, needs decomposition & independent evolution of parts.

**Problem:**

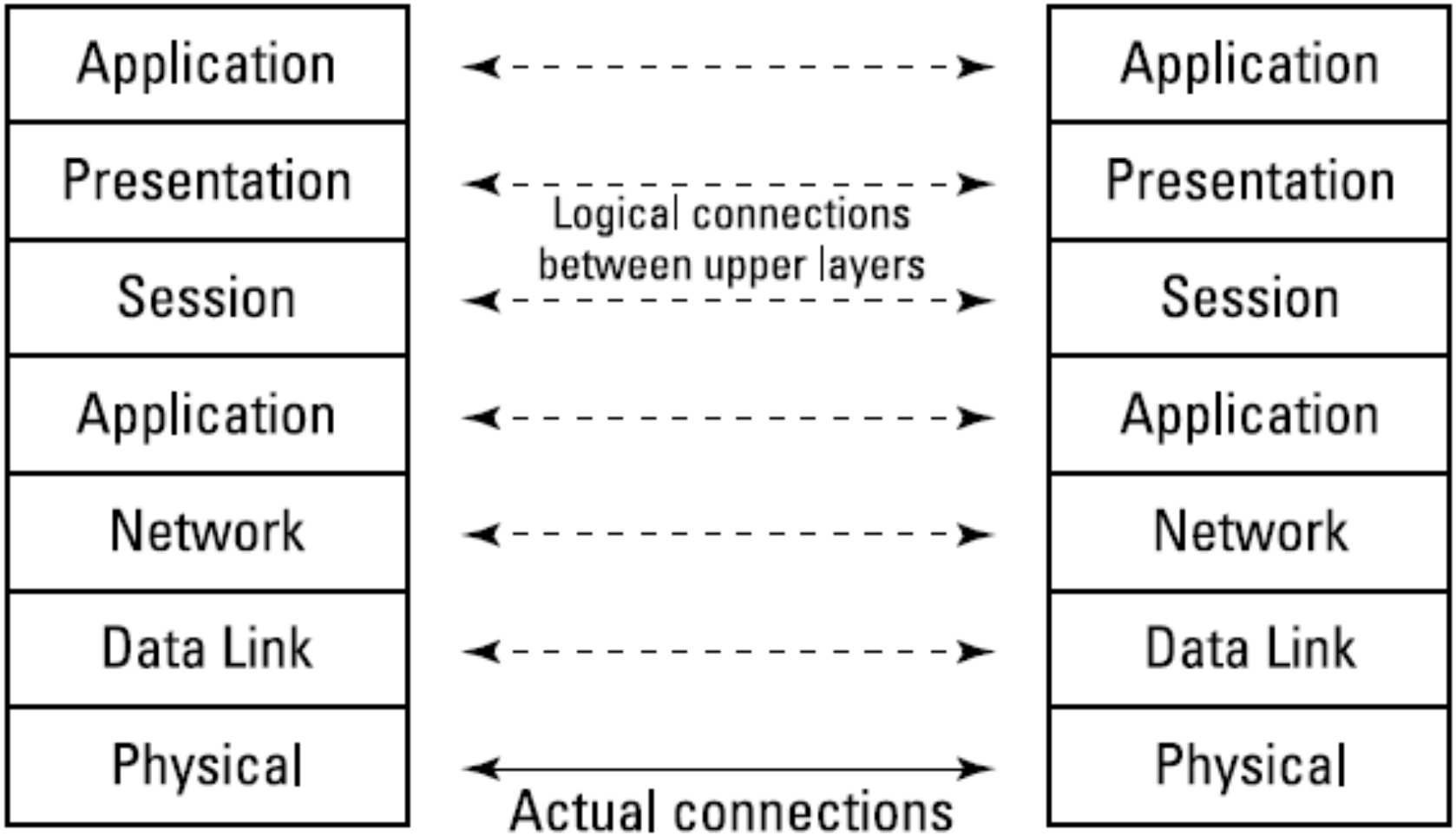
Developers need a clear & well-documented way to implement separation of concerns (focus)

Software needs to be segmented such that modules can be developed and evolved separately with little interaction among the parts, supporting portability, modifiability, and reuse.

Part of the system can be extended (evolution)

Stable interface among modules – defined by standards

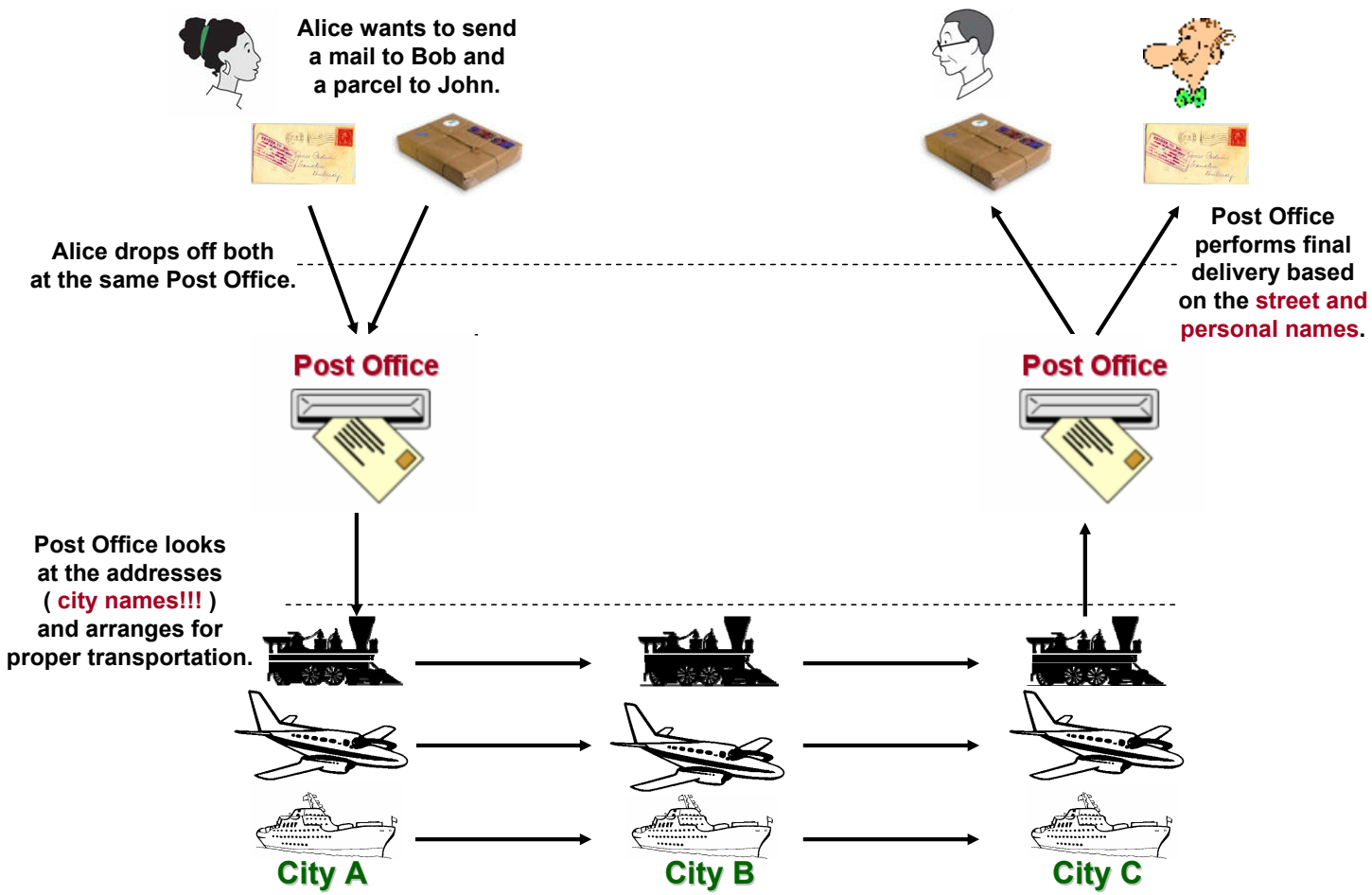
# OSI 7 layer model



# Network Layering

## Why Layering?!

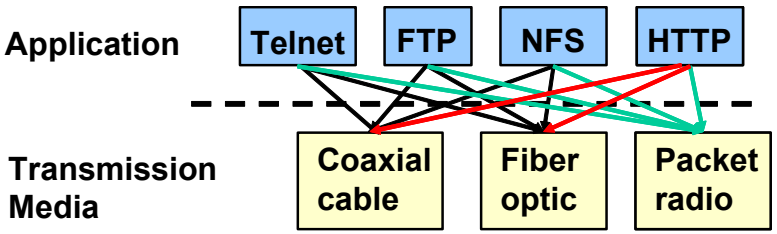
3



# Network Layering

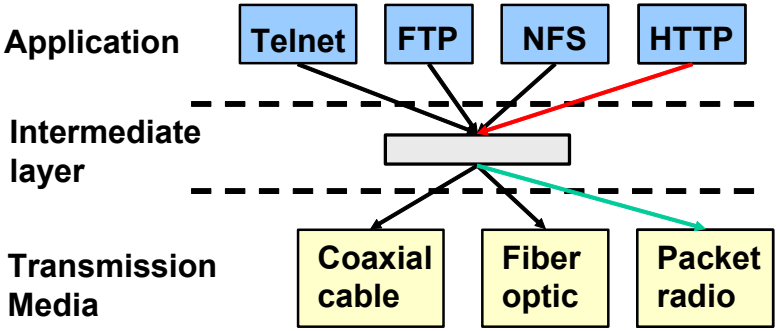
**No Layering**

- each new application has to be *re*-implemented for every network technology!



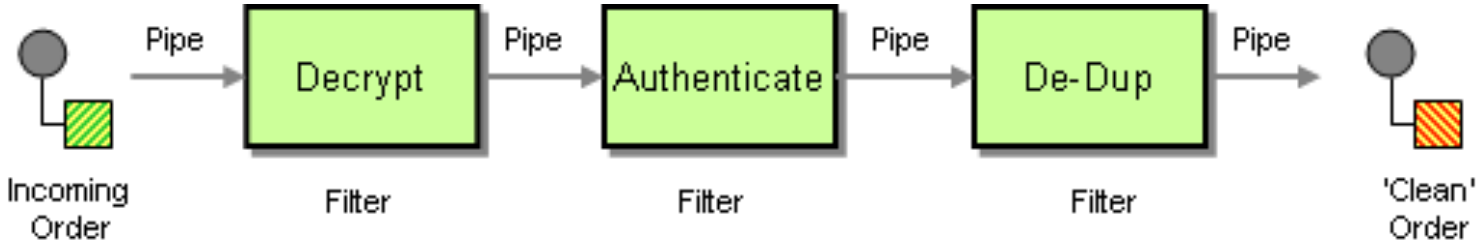

**Layering**

- intermediate layer(s) provide a unique abstraction for various network technologies



# Pipe and Filter Pattern

- Useful when we need to perform a number of transformations in a sequence
- Data arrives at a filter's input port(s), is transformed, and then is passed via its output port(s) through a pipe to the next filter.
- A single filter can consume data from, or produce data to, one or more ports.



# Pipe and Filter Pattern

This architecture is useful when we need to perform a number of transformations in a sequence

**Context:** Many systems are required to transform streams of discrete data items, from input to output.

**Problem:** Such systems need to be divided into reusable, loosely coupled components with simple, generic interaction mechanisms.

In this way they can be flexibly combined with each other. The components, being generic and loosely coupled, are easily reused. The components, being independent, can execute in parallel.

# Example: Monitoring real time events by cab operator

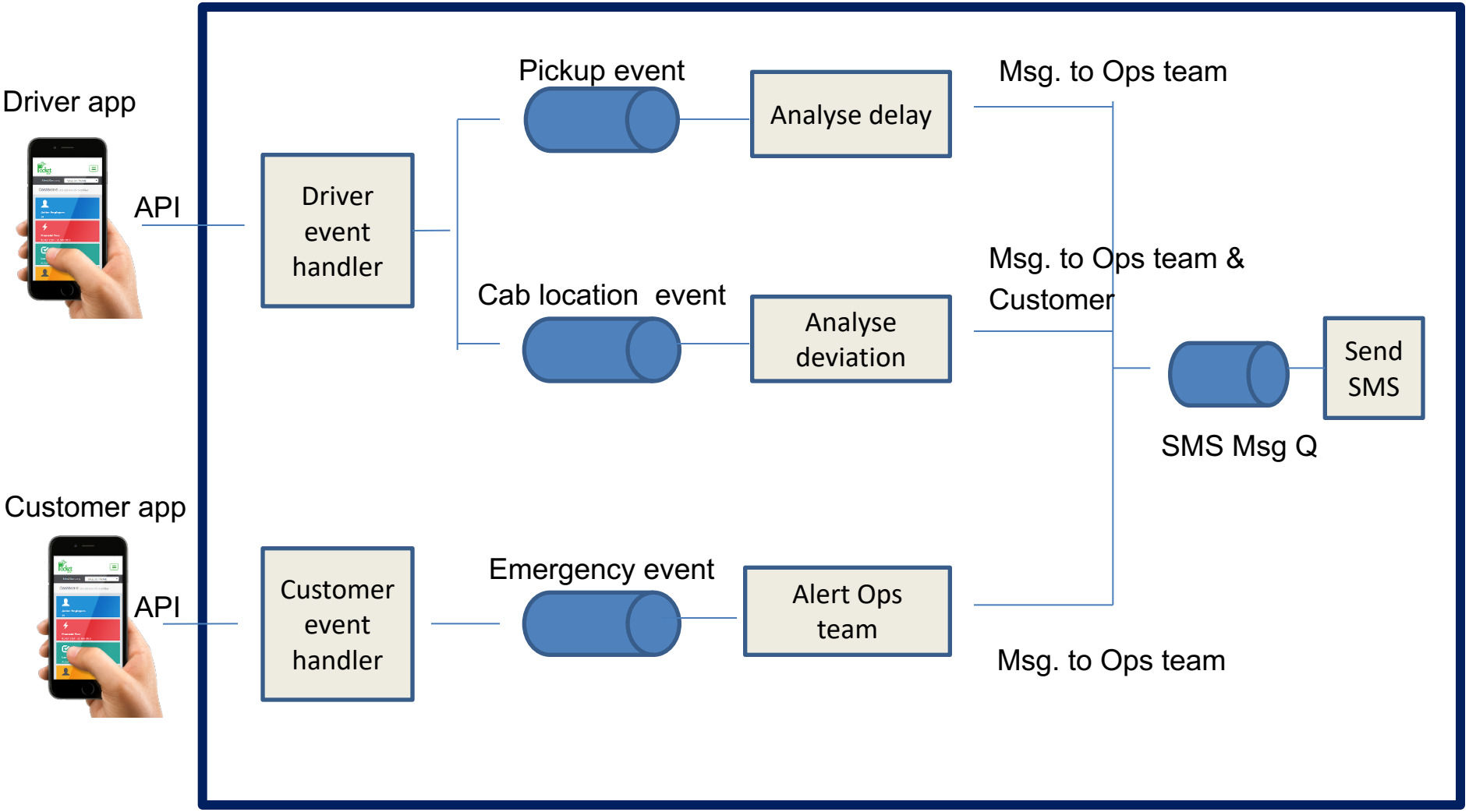
An App based cab operator (like Ola / Uber) wants to **monitor real time events** and take appropriate action as follows:

a) If the cab **deviates from the designated route** and if the deviation is significant, then we want to alert the passenger as well as the ops team (inform via SMS)

b) If the customer presses the **emergency button**, we want to inform the police and the ops team immediately (inform via SMS)

**Using Pipe & filter pattern, draw an architecture diagram to implement the above requirements. Indicate how the pipes and filters are used.**

# Example: Monitoring real time events by cab operator



# Pipes & Filters: Applications

Applications where Pipes & Filters are useful

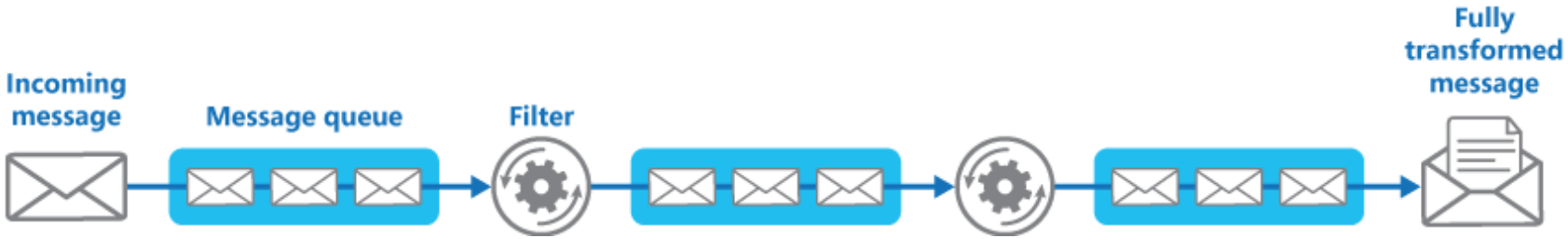
- Satellite signal processing – noise reduction, transformations
- Extract – Transform – Load (ETL) is a good fit for Pipes & filters architecture
- Real time recommendations to customer in ecommerce

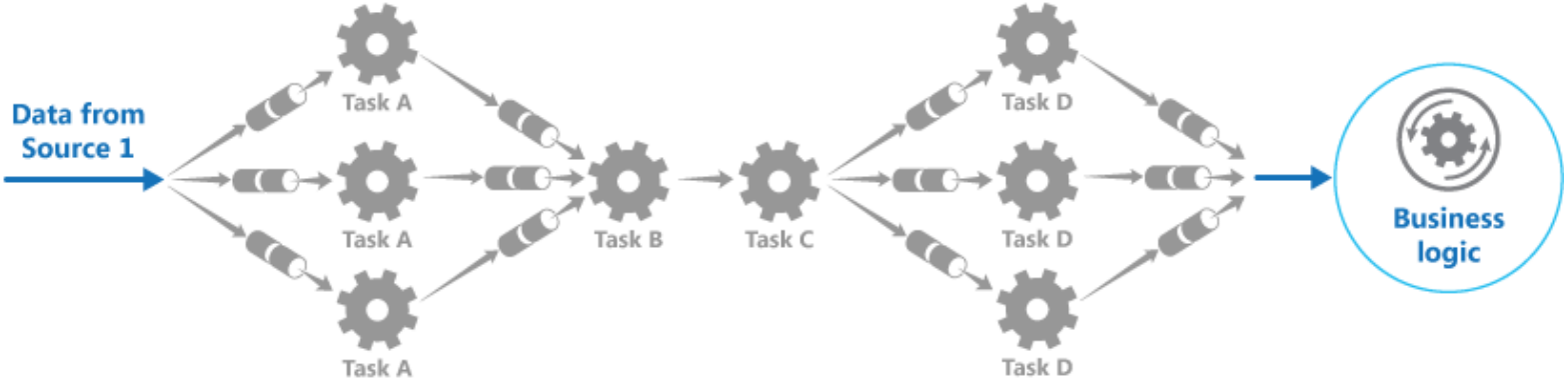
Weakness

- Not good for interactive system
- More filters - Computational overhead
- Any failure can cause the entire pipeline to fail

**Think about availability and Scalability !**

# Pipes & Filters



# Model View Controller (MVC)

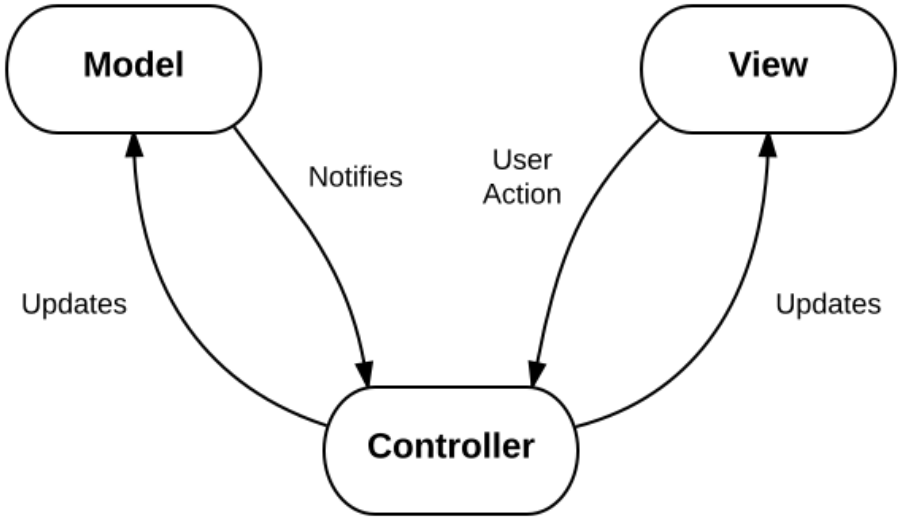
MVC framework utilizes component-based design approach where components belong to one of 3 types.

Each component supports an interface

Date Format

- Handles data & business logic
- independent of the GUI
- Does not know How to present data

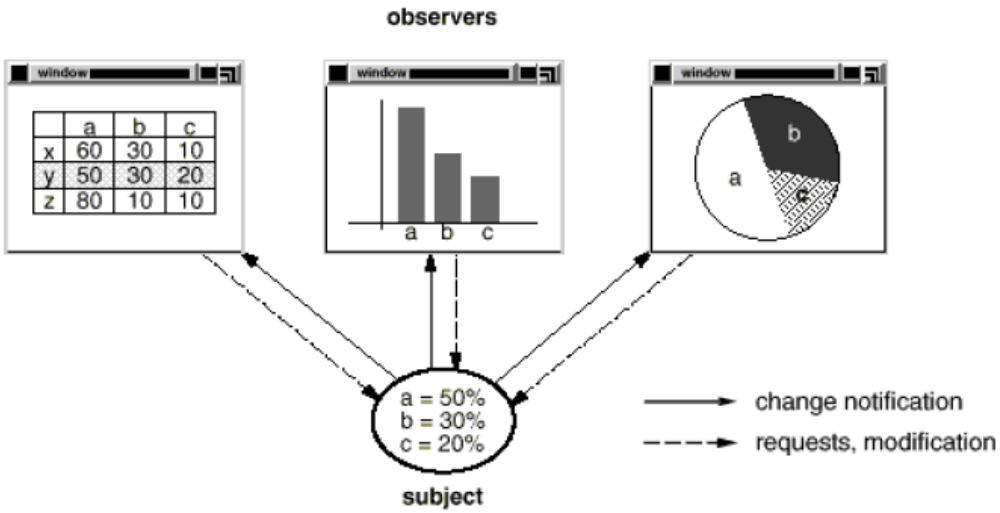
- Presents data to the user
- Knows how to display
- Not responsible to store data
- when the model changes, the view automatically redraws




- Receives user requests, calls appropriate modules and performs business logic
- Specific to Application, least reusable

# Model View Controller (MVC)

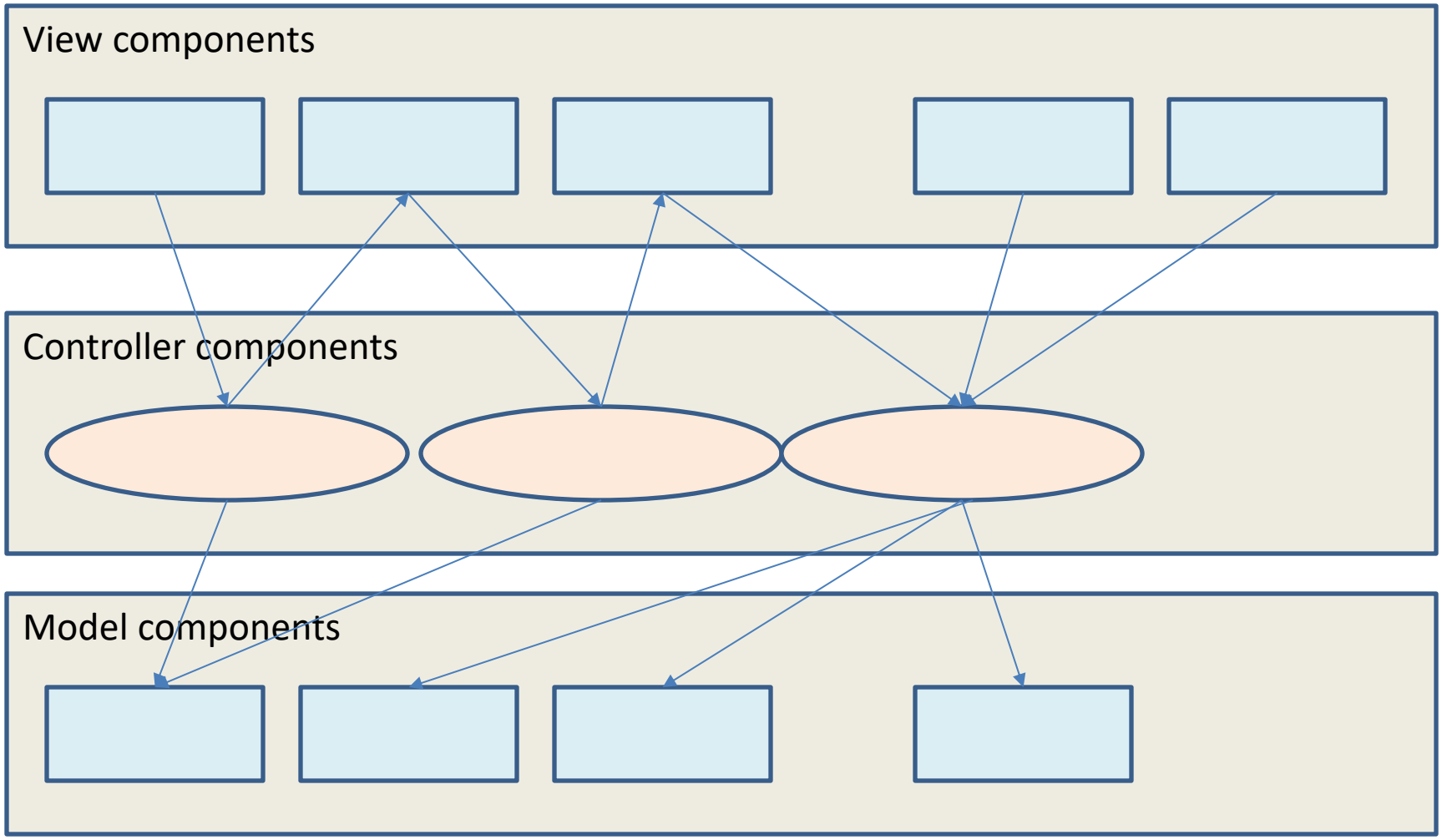
- Useful when we need different views of data
- Eg. Bar chart & Pie chart of data



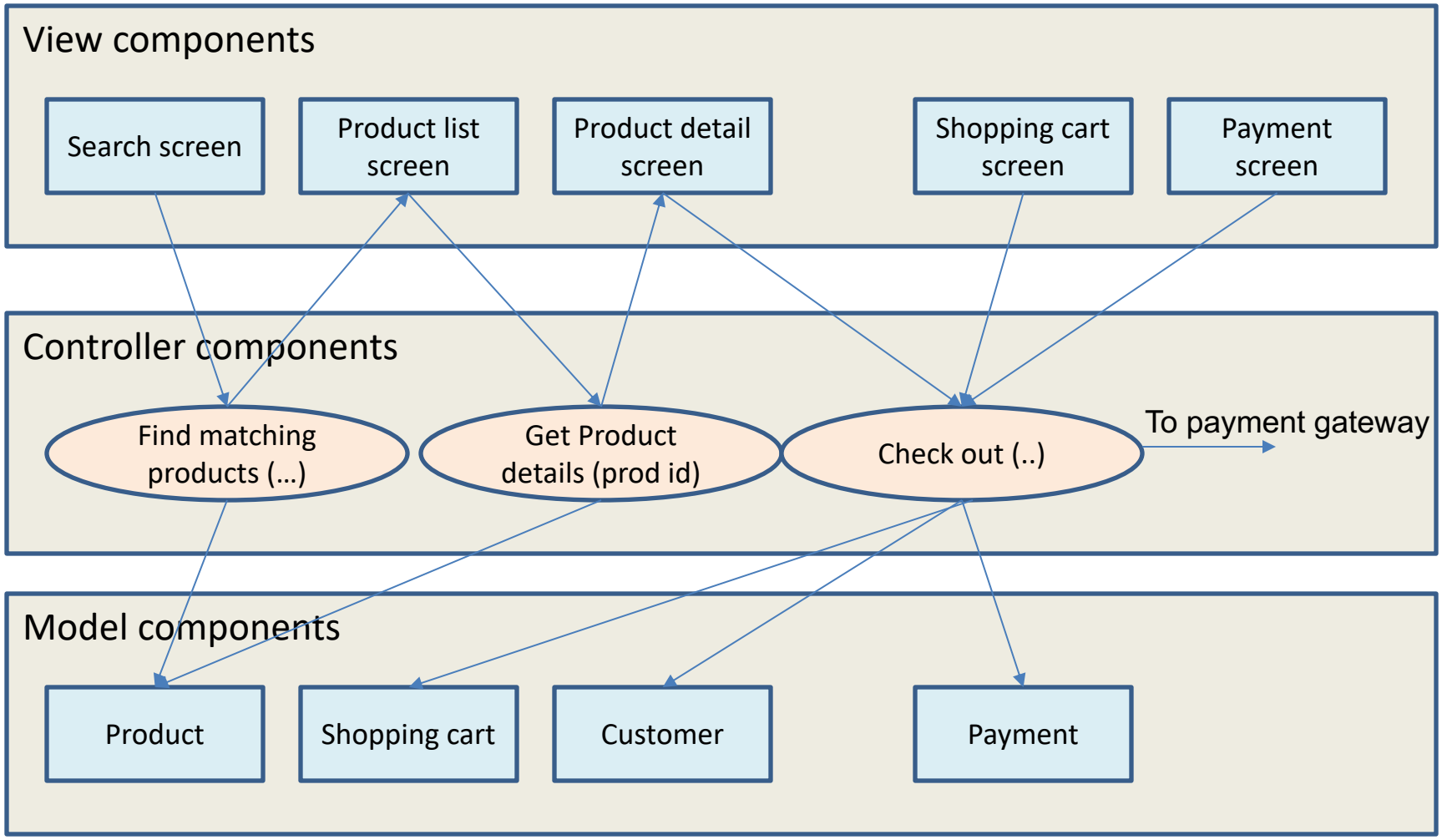
# Drivers for MVC

- User Interface Changes frequently
- the application displays the same data in different ways
- Skills Required to develop UI .vs. Business Logic
- UI is device dependent
- Test Automation UI vs Business Logic

# MVC Exercise: For an eComm system, identify 1 component of each type – Model, View & Controller



# MVC architecture of eComm system

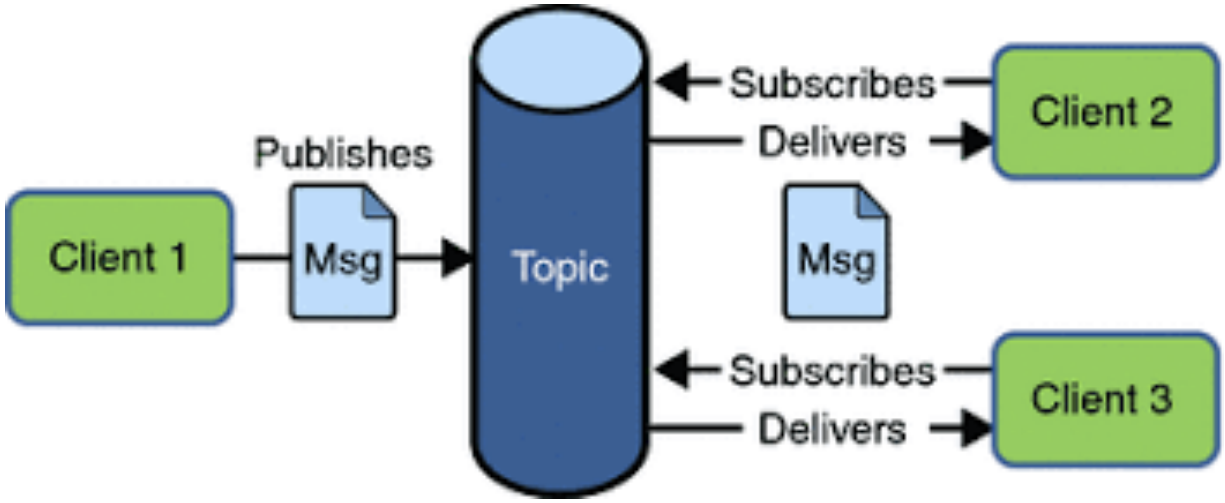


# Publish-Subscribe Pattern

- Number of independent producers and consumers of data that must interact.
- The precise number of producers, consumers, amount of data are not predetermined or fixed
- integration mechanisms to transmit messages among the producers and consumers who are unaware of each other's identity
  - Consumers subscribe to a set of events.
  - Publisher place events on the bus by announcing them
  - Connector delivers those events to the subscriber that have registered for those events

# Publish - Subscribe

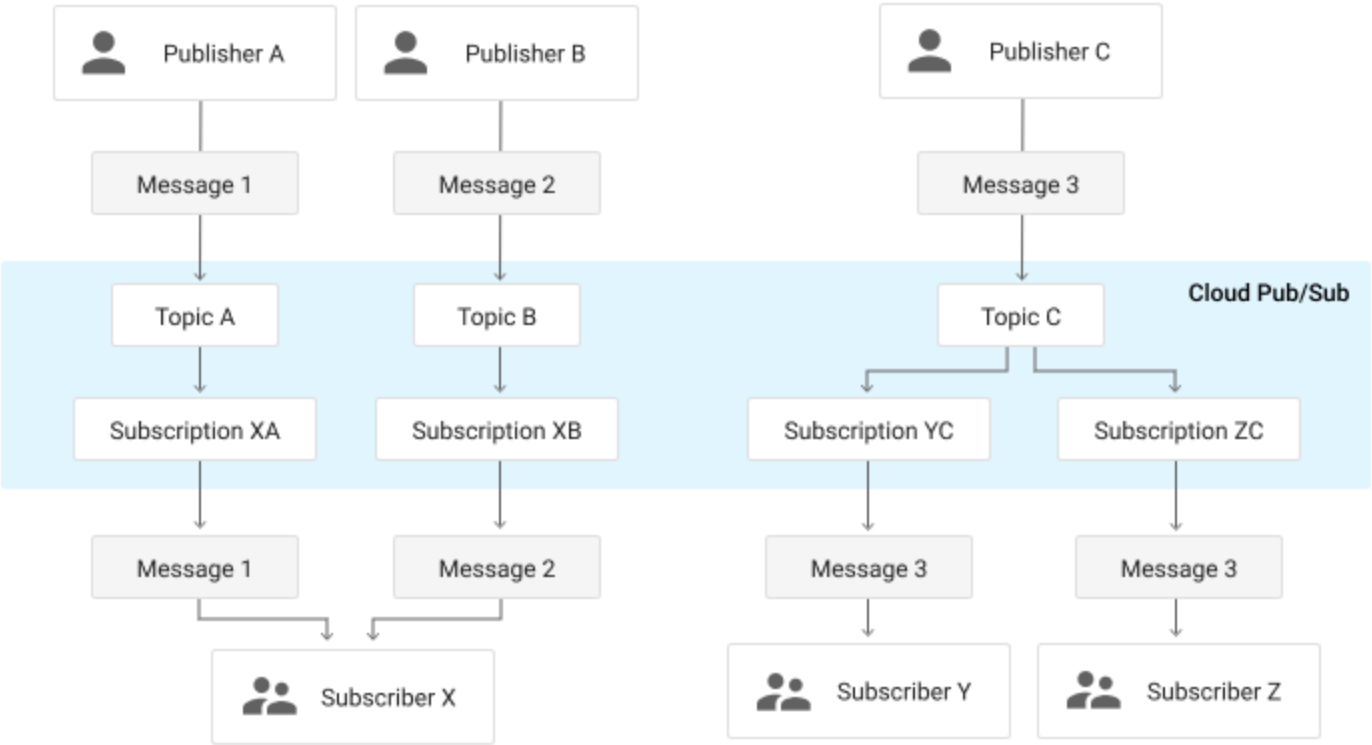
- Components interested in knowing about events, subscribe to relevant events
- Publisher place events on an Event bus
- Event bus delivers events to subscribers who have registered for those events




Example of Topics in a stock exchange
- Change in share price
- Increase in volume of trading

# Publish – Subscribe scenarios

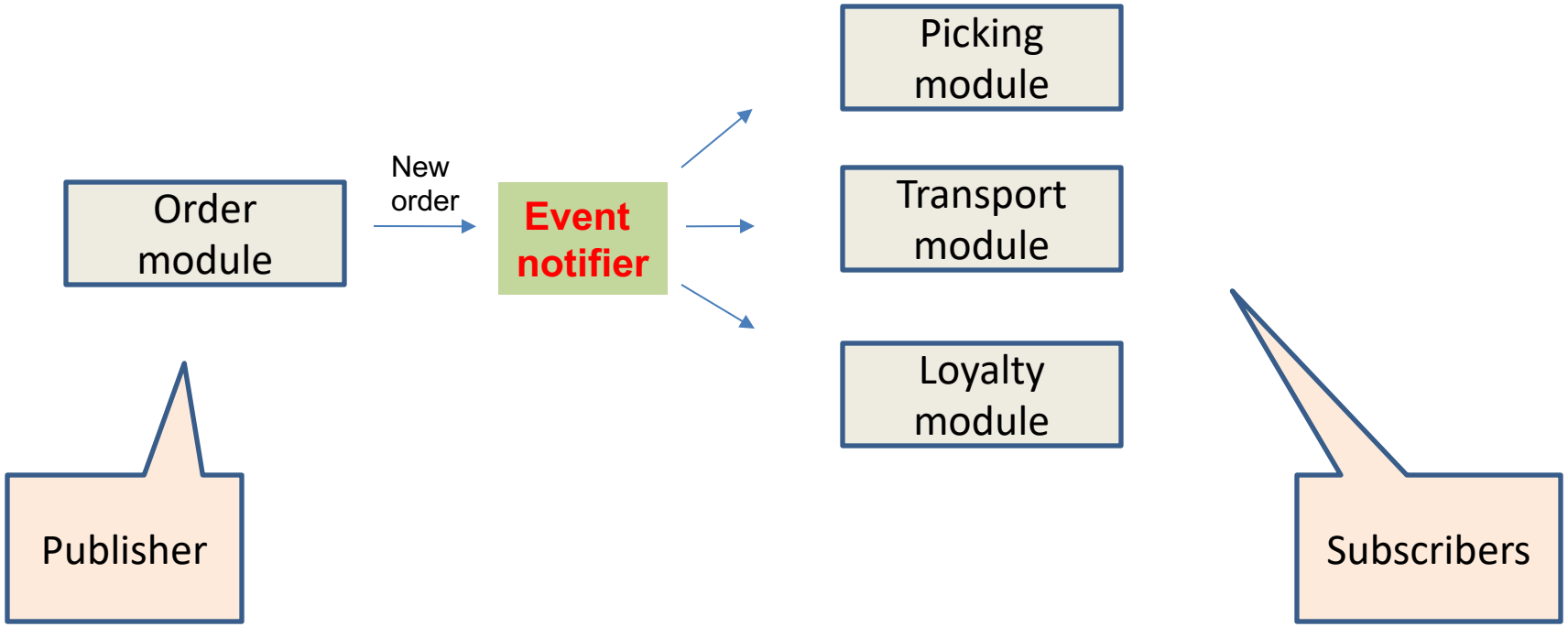
- A process may subscribe to more than one topic / channel
- A topic / channel can be subscribed to by more than 1 process



# Publish - Subscribe

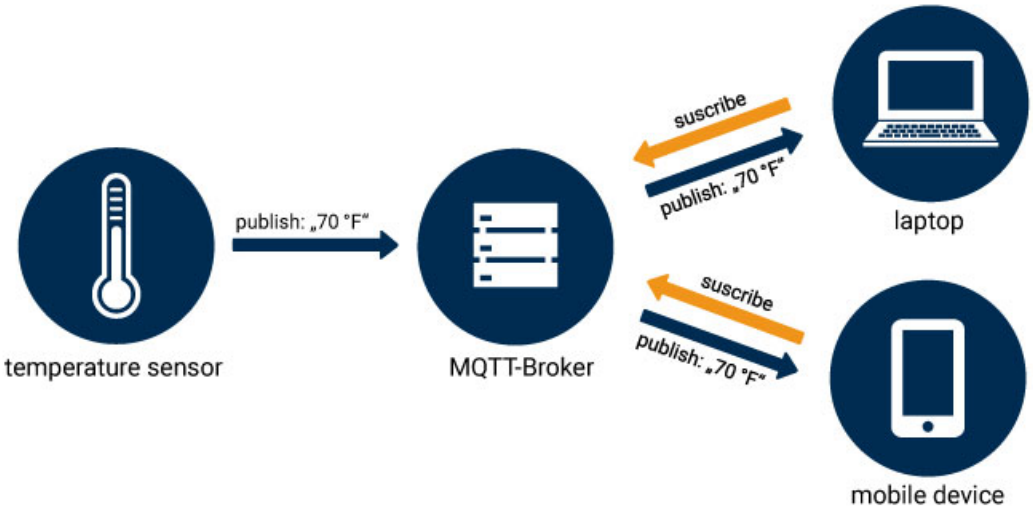
Example

- Order creation event is subscribed to by Picking module, Transport module, Loyalty module



# Publish-Subscribe tools

- Amazon Simple Notification Service (SNS)
- MQTT (MQ Telemetry Transport) for IoT: Extremely lightweight publish/subscribe protocol



# Benefits of Publish-Subscribe

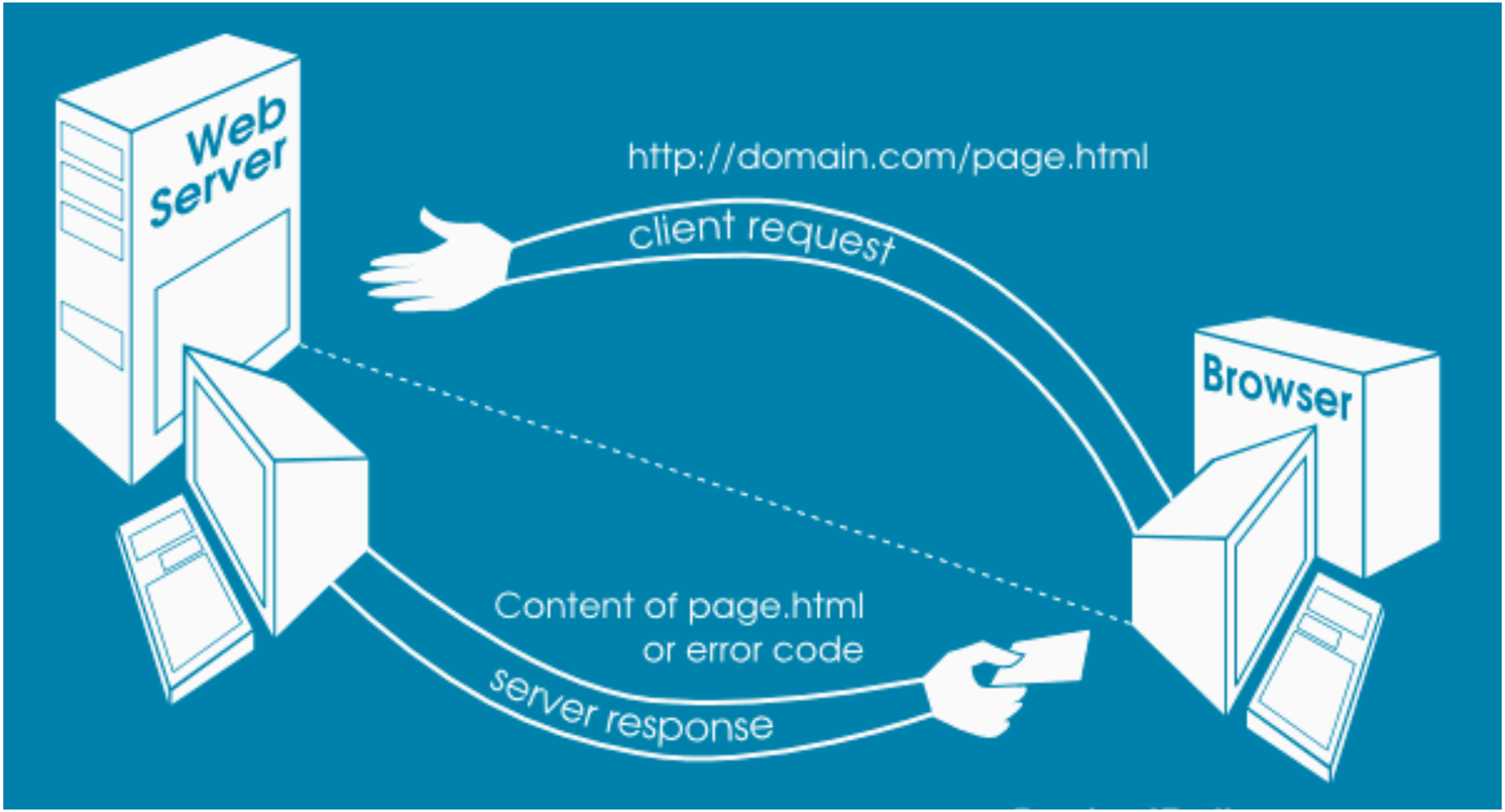
- Loose coupling between producer & consumer
- New listeners can be added without impacting producer
- Eliminates polling

# Publish & Subscribe

Can you describe a scenario where you have used Publish-Subscribe pattern?

# Client-Server Patterns

# Client-Server Architecture of Internet



# Client-Server Pattern

- Access to shared resources and services
- Clients interact by requesting services of servers (central / Distributed)
- Some components may act as both clients and servers. There may be one central server or multiple distributed ones.
- By managing shared resources and services, we can promote modifiability, reuse, scalability and availability

# Client Server - Weakness

- Server can be a performance bottleneck.
- Server can be a single point of failure.
- Decisions about where to locate functionality (in the client or in the server) are often complex and costly to change after a system has been built.

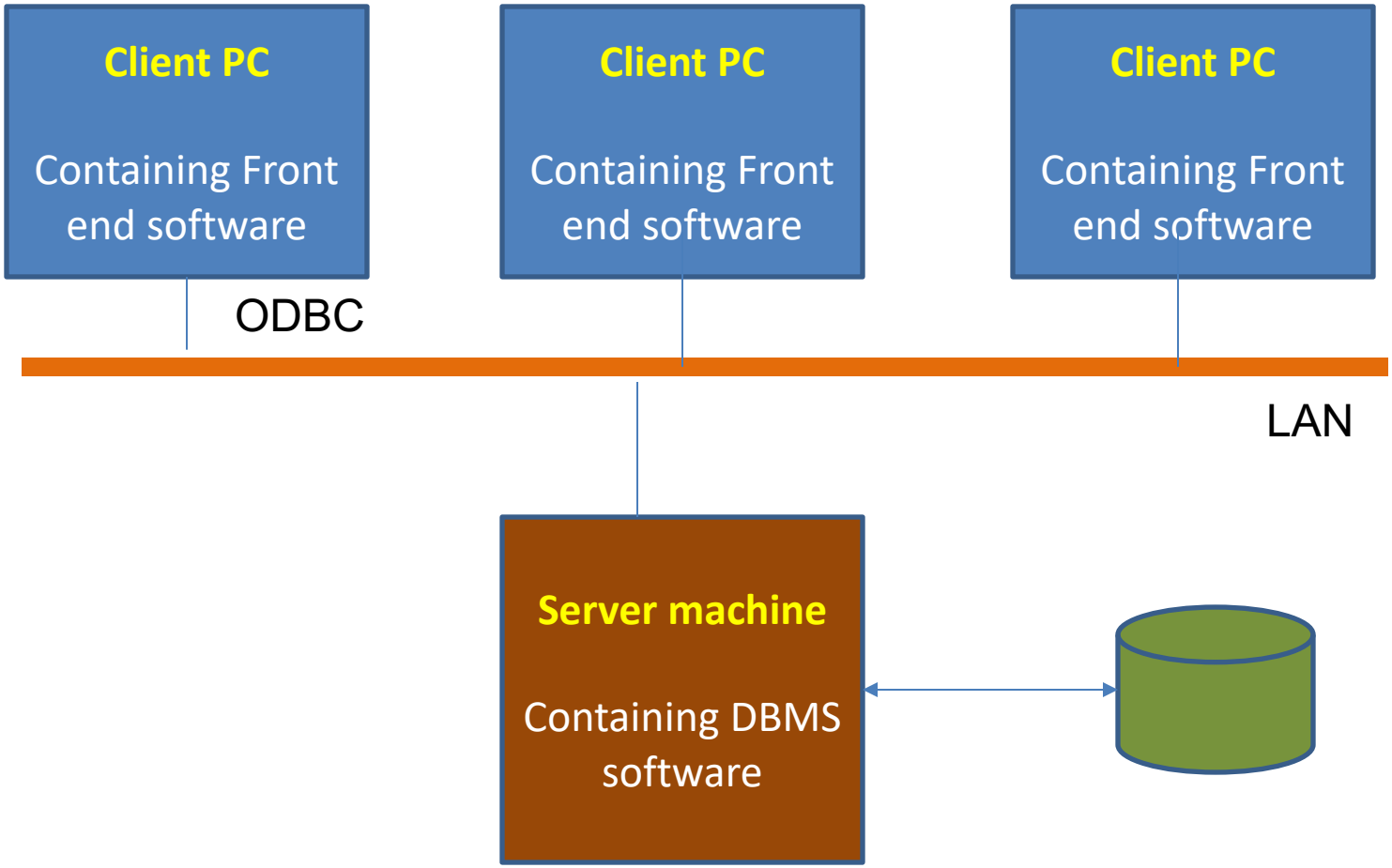
# Client - Server pattern

Useful when several clients want to access the services provided by the server

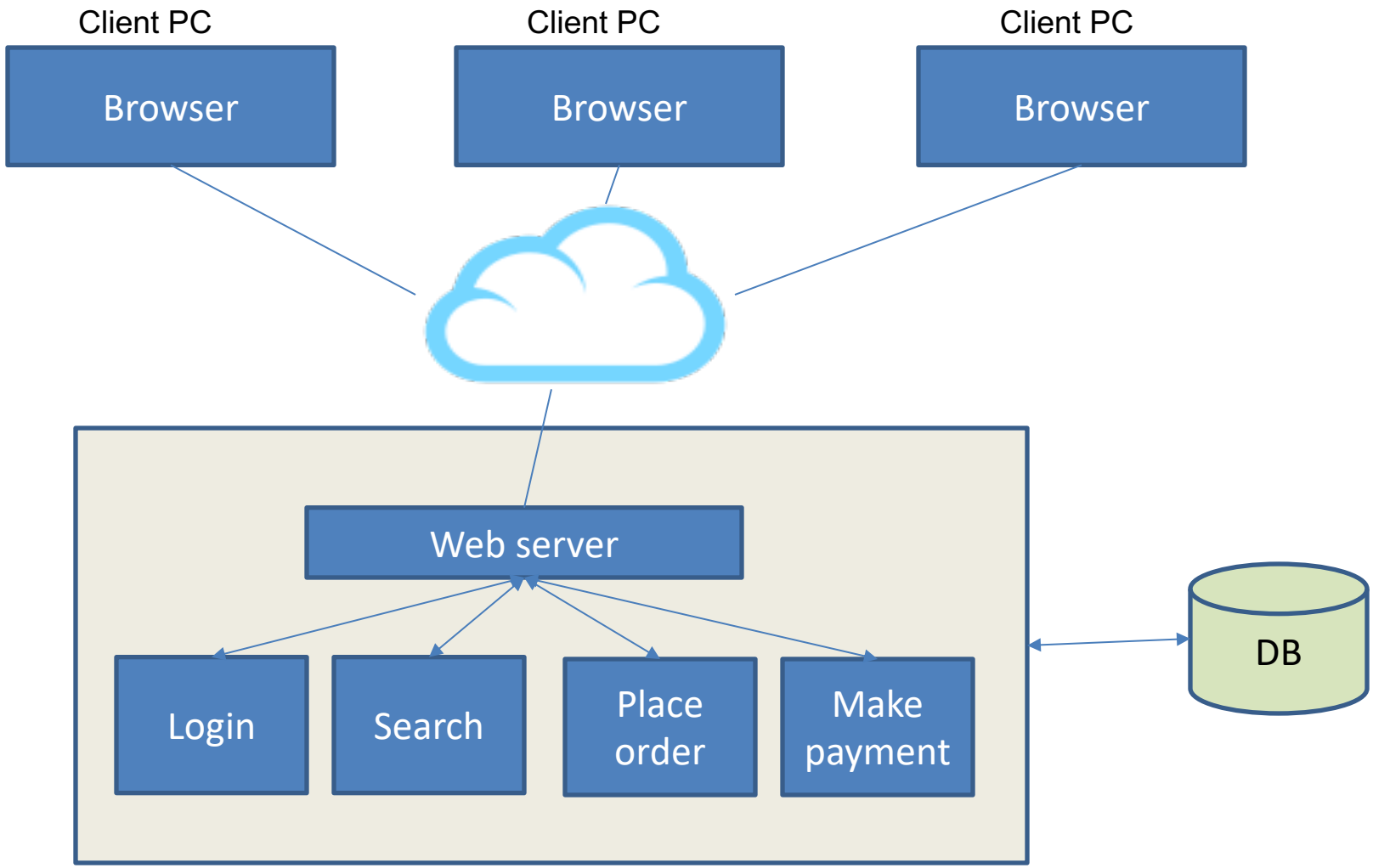
Examples

- Database server (accessed using ODBC)
- Mail server (MS Outlook)
- Web server (accessed using HTTP request)

# Client server pattern: Example



# Client server pattern: Example



# Client-server pattern

Have you come across systems that use this pattern?

# Client – Server pattern

Variants of client – server

- Earlier client server communication was synchronous

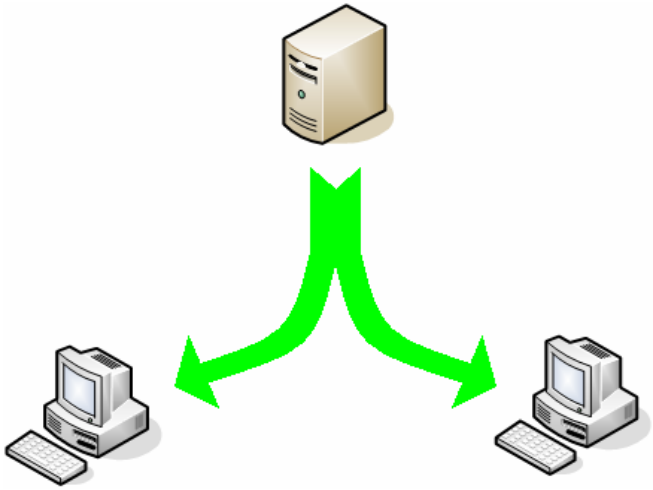
Now we have

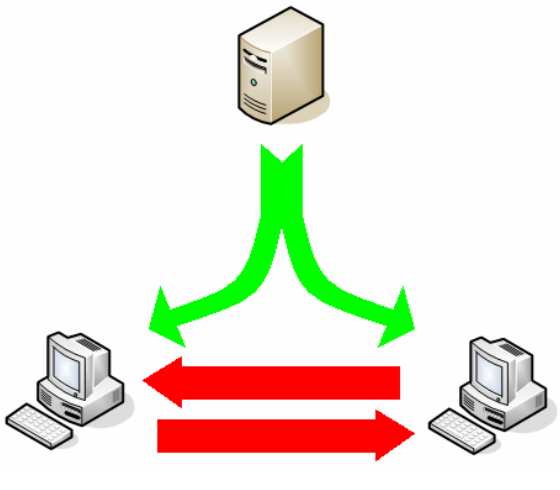
- Asynchronous communication (using AJAX)
- Clients providing call back functions (ex. Javascript). Eg. Updating Live cricket scores on a Cricket website or updating Live stock prices in a stock exchange

Demerits

- Server can be a single point of failure
- Server can be a bottleneck

# BitTorrent





# P2P

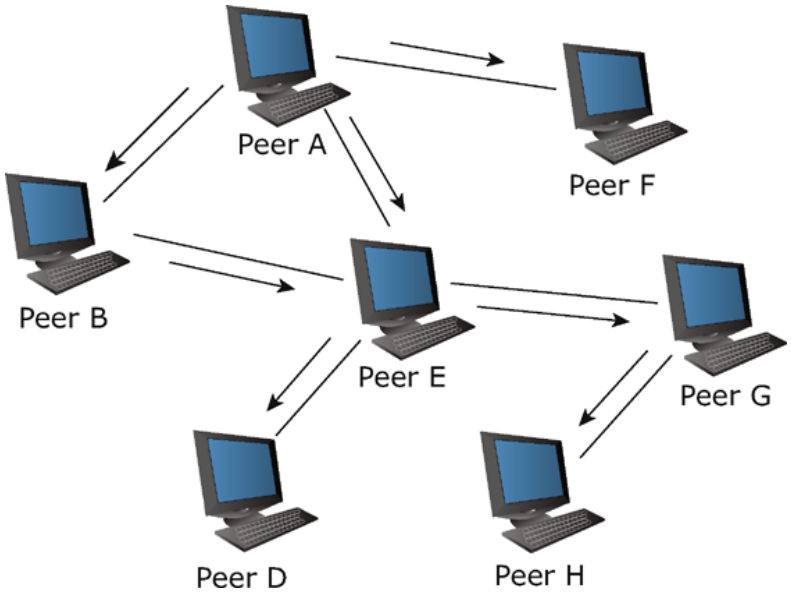
A **peer-to-peer** (**P2P**) **architecture** consists of a decentralized network of **peers** - nodes that are both clients and servers. **P2P** networks distribute the workload between **peers**, and all **peers** contribute and consume resources within the network without the need for a centralized server.

# P2P Network

- Peers first connect to the P2P network
- They discover other peers they can interact with, and then initiate connections
- Often a peer's search for another peer is propagated from one peer to its connected peers.
- A peer-to-peer architecture may have specialized peer nodes (called super nodes) that have indexing or routing capabilities and allow a regular peer's search

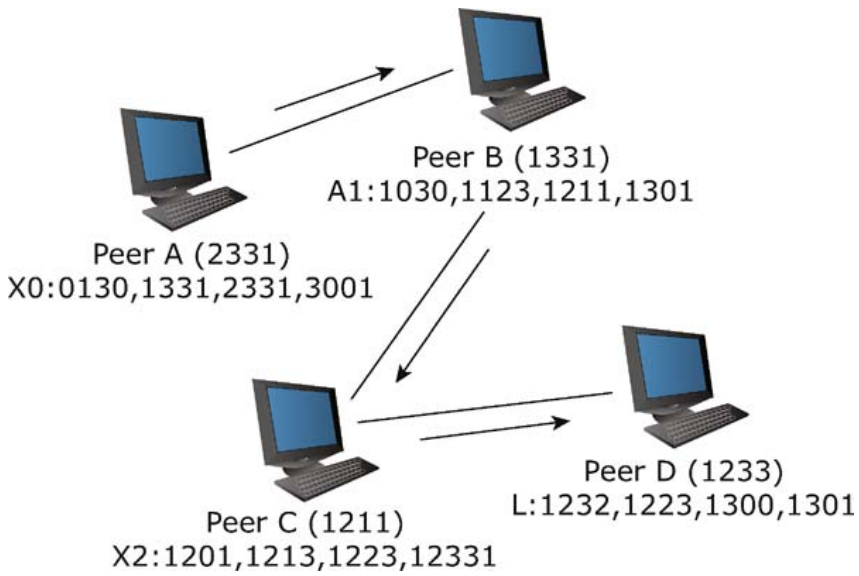
# P2P Search

Peer A requests for some data that Peer D and Peer H have. The query will be broadcasted to the neighbors of Peer A, and gradually, to the other peers in the whole network



# P2P Communication

Peer 2331 issues a query for a file on Peer 1233. Each time, the query is forwarded to a peer closer to the destination. Finally, it will arrive at Peer 1233



# P2P Application

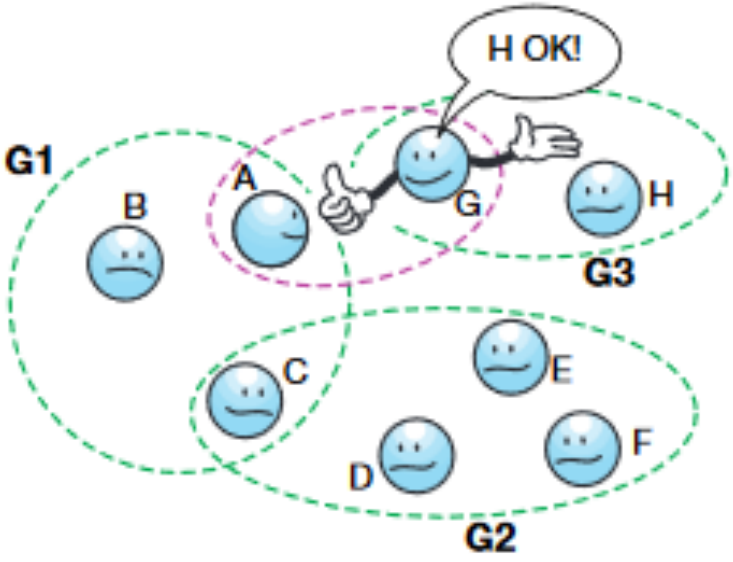
Microsoft in Windows 10 uses a proprietary peer to peer technology called "Delivery Optimization" to deploy operating system updates using end-users PCs either on the local network or other PCs. According to Microsoft's Channel 9 it led to a 30%-50% reduction in Internet bandwidth usage

# P2P & Wireless Adhoc Networks

Army uses Wireless Ad-hoc networks to share information between different entities in the battle field.

The entities are on the move and there is no fixed network consisting of links and routers

Every node gathers information about the enemy and shares it with other nodes



# Peer-to-Peer Challenges

Challenges

- Data consistency – Copies of data in different nodes might be out of sync
- Data & service availability – No central authority to ensure this
- Backup & Recovery – May be more involved, since there is no single source from where to recover

# Shared-Data Pattern

- Various independent computational components need to share and manipulate large amounts of data.
- This data does not belong solely to any one of those components.
- In the shared-data pattern, interaction is dominated by the exchange of persistent data between multiple *data accessors* and at least one *shared-data store*.
- Exchange may be initiated by the accessors or the data store.

# Shared Data Weakness

- The shared-data store may be a performance bottleneck.
- The shared-data store may be a single point of failure.
- Producers and consumers of data may be tightly coupled.

Architectural patterns & tactics: captures proven design structures so that they can be reused; Described as {context, problem, solution }

- *A context*. A recurring, common situation that gives rise to a problem.
- *A problem.* The pattern describes the problem, its variants, any complementary or opposing forces. Also includes quality attributes that must be met.
- *A solution.* The solution describes the architectural structures that solve the problem, including how to balance the many forces at work. The solution for a pattern is described by:
  - A set of element types (for example, data repositories, processes, and objects)
  - A set of interaction mechanisms or connectors (for example, method calls, events, or message bus)
  - A topological layout of the components
  - A set of semantic constraints

| No | Style | Problem | Tactics | QA - Strong | QA - Weak | Application |
|---|---|---|---|---|---|---|
| 1 | Layer | | | | | |
| 2 | Broker | | | | | |
| 3 | MVC | | | | | |
| 4 | Pipe & Filter | | | | | |
| 5 | Client & Server | | | | | |
| 6 | Peer to Peer | | | | | |
| 7 | SOA | | | | | |
| 8 | Publish & Subscribe | | | | | |
| 9 | Shared Data | | | | | |
| 10 | Map-Reduce | | | | | |
| 11 | Multi Tier | | | | | |

# Shared data pattern

Database is the central piece

Different components share the database

The database supports

- Persistence
- Concurrent access
- Fault tolerance
- Access control
- Distribution
- Caching