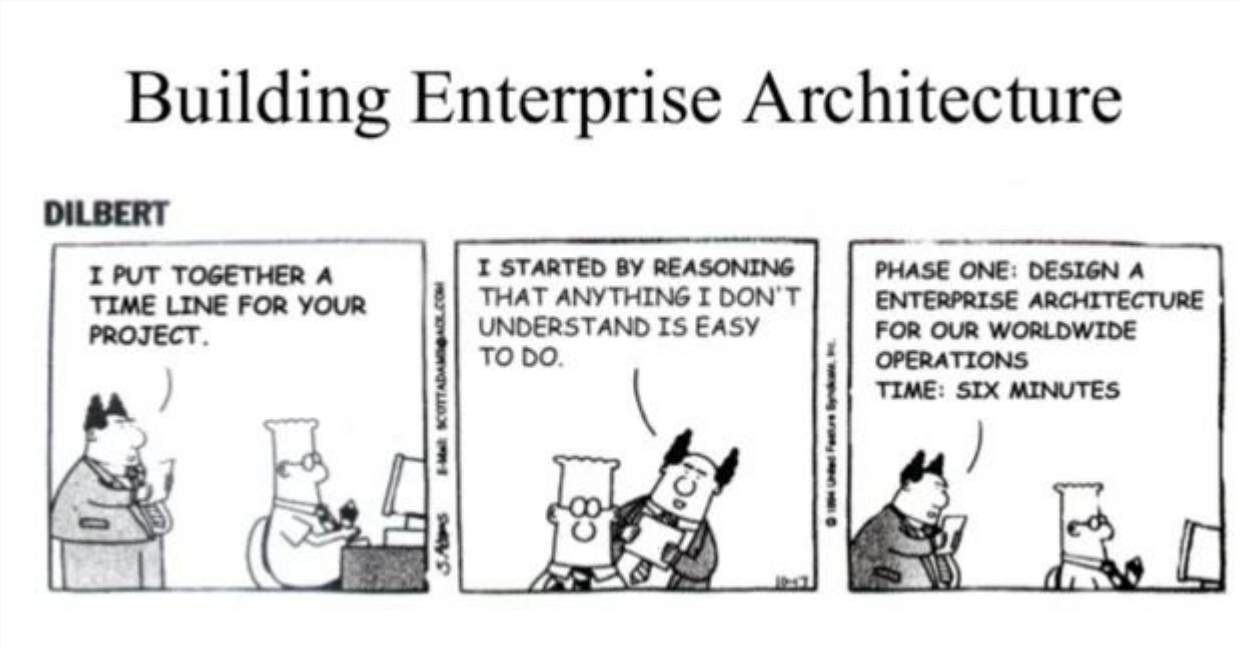
Lesson-8:Software Architecture Fundamentals

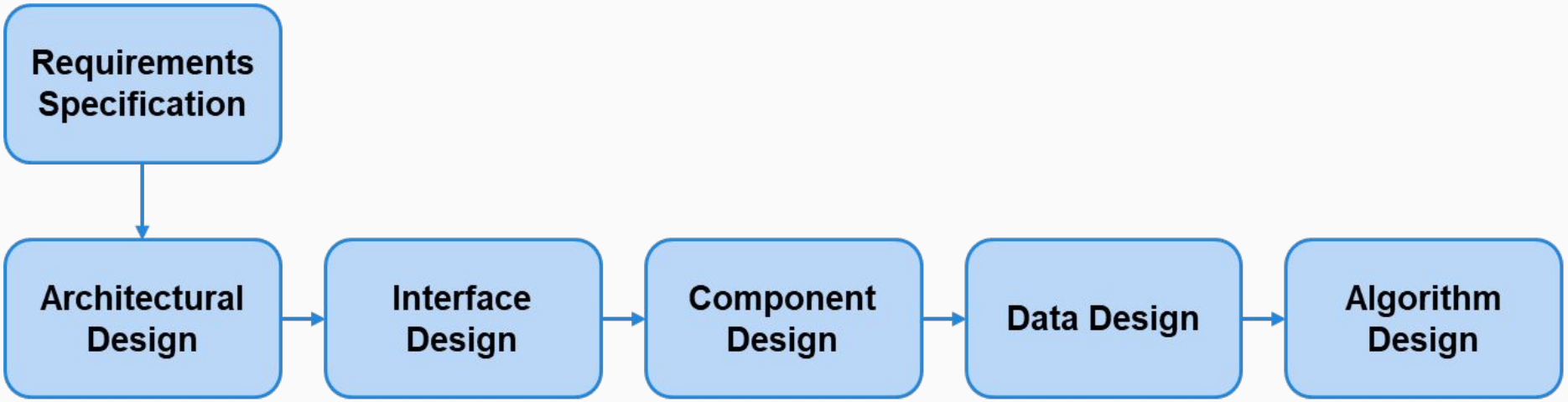
CS 438
Ali Aburas PhD

Today's Goals

- Define Software Architecture
- Common software architecture patterns



Design Activities



System architecture (high-level breakdown of system)

How subsystems interact with each other and how external users and systems interact with your system

A listing of the individual classes within your system

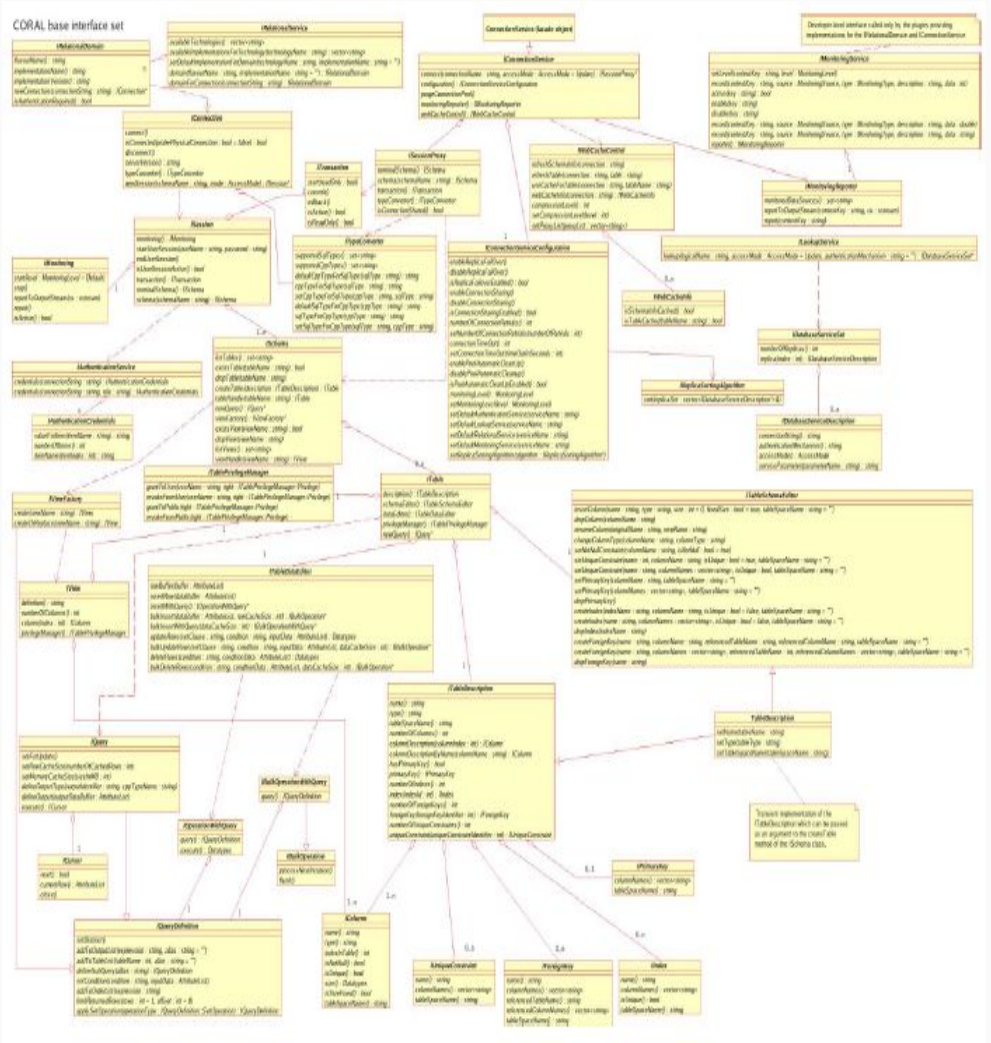
The format of data that is produced and consumed by your system

Algorithms used to implement system functionality.

Software Starts to Grow Up

Under the Hood:

- Systems have millions of lines of code. Example: Linux Kernel 16 million Lines of Code
- No person can understand an entire million-line codebase.
- Divided into hundreds of classes.
- Classes organize code, but how can you find the right clases when there are thousands?

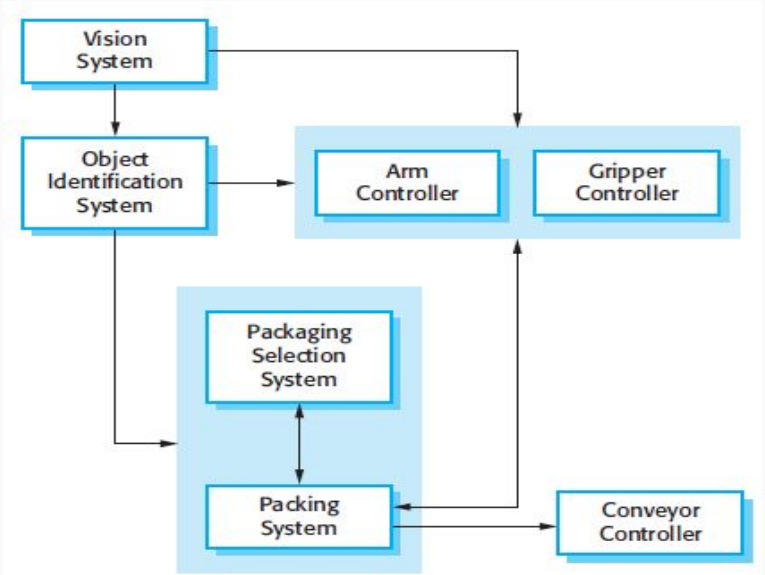


Architectural design

- **Architectural design** is concerned with understanding how a system should be organized and designing the overall structure of that system.
- The output of the architectural design process is an architectural model that describes **how the system is organized** as a set of communicating components.
- The nonfunctional requirements depend on the system architecture—the way in which these components are organized and communicate.

- This shows an abstract model of the architecture for a **packing robot system** that shows the components that have to be developed.
- It uses a vision component to pick out objects on a conveyor, identify the type of object, and select the right kind of packaging.
- The system then moves objects from the delivery conveyor to be packaged.
- It places packaged objects on another conveyor.
- The architectural model shows these components and the links between them.

architecture of a packing robot control system



What is Software Architecture?

- **Architecting software** is the practice of partitioning a large system into smaller ones.
  - That can be created separately
  - That individually have business value
  - That can be straightforwardly integrated with one another and with existing systems.
- **First stage of design.**
  - **Partitions** the requirements into self-contained subsystems.
  - Later, each subsystem will be decomposed into one or more classes.
  - Plan how those subsystems cooperate and communicate.

Advantages of explicit architecture

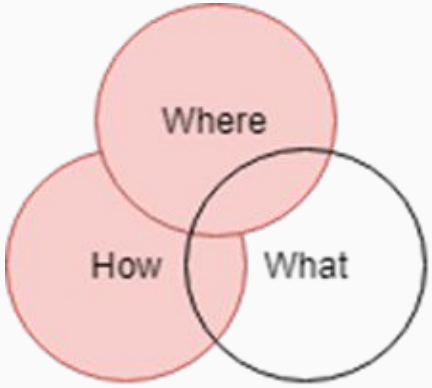
1. ***Stakeholder communication***
   - Architecture may be used as a **focus of discussion** by system stakeholders, for the negotiation of system requirements.
   - It is an essential **tool for complexity management**. **It hides details** and allows the designers to **focus on the key system abstractions**.
2. ***System analysis***
   - Means that analysis of whether the system can **meet its nonfunctional requirements is possible**.
   - Architectural design decisions have a **profound effect** on whether or not **the system can meet critical requirements such as performance, reliability, and maintainability**.
3. ***Large-scale reuse***
   - **The architecture may be reusable** across a range of systems with similar requirements and so **can support large-scale software reuse**

Why is Software architecture Important

- It affects the **performance, robustness, distributability, and maintainability** of a system.
- Individual components implement **the functional system requirements**.
- The **non-functional** requirements depend on the system architecture—the way in which these components are **organized and communicate**.
- In many systems, non-functional requirements are **also influenced by individual components**, but there is no doubt that **the architecture of the system is the dominant influence**.

Difference between Architecture and Design

- **software design** is focused on the details of the system, while **software architecture** is focused on the overall structure and high-level design.
- **Software design** typically involves making decisions about the algorithms, data structures, and other low-level details of the system, while **software architecture** focuses on the overall shape of the system and how it will meet the requirements.



Design

Architecture

Architecture and system characteristics

The particular architectural style and structure that you choose should depend on the non-functional system requirements::

1. **Performance** If performance is a critical requirement, the architecture should be designed to **localize critical operations within a small number of components**, with these components all deployed on the same computer rather than distributed across the network.
    a. This may mean using a few relatively large components rather than small, fine-grain components, which reduces the number of component communications.
2. **Security** If security is a critical requirement, a layered structure for the architecture should be used, with the most critical assets protected in the inner layers, with a high level of security validation applied to these layers.
3. **Safety** If safety is a critical requirement, the architecture should be designed so that safety-related operations are all located in either a single component or in a small number of components.
    a. This reduces the costs and problems of safety validation and makes it possible to provide related protection systems that can safely shut down the system in the event of failure.

Architecture and system characteristics

4. **Availability** If availability is a critical requirement, the architecture should be designed to include redundant components so that it is possible to replace and update components without stopping the system.
5. **Maintainability** If maintainability is a critical requirement, the system architecture should be designed using fine-grain, self-contained components that may readily be changed. Producers of data should be separated from consumers and shared data structures should be avoided.

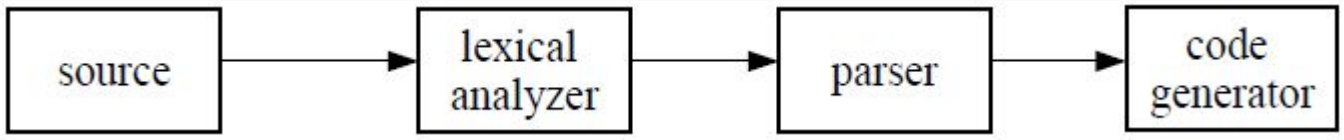
- **Note**: Obviously there is potential **conflict between some of these architectures**. For example, using large components improves performance and using small, fine-grain components improves maintainability. If both performance and maintainability are important system requirements, then **some compromise must be found**.

What is an Architectural Pattern?

- The idea of patterns as a way of presenting, sharing, and reusing knowledge about software systems is now widely used..
  - Use an architecture you've seen before
- You can think of an **architectural pattern** as a stylized, abstract description of good practice, which has been tried and tested in different systems and environments.
- An architectural pattern should describe a system organization that has been successful in previous systems.

SW Architecture (Option #1): Pipe and Filter

- Input is taken in by one component, processed, and the output serves as input to the next component.
  - Filter = component that computes on the data
  - Pipe = connector that passes data between filters
- Data can be processed as items or batches.
- Example: **A compiler for a programming language**



- Example: **Programs written in the Unix shell**

  Commands are piped together using vertical bar " | " symbol. Syntax:

```
cat Files.txt | grep "new"
```

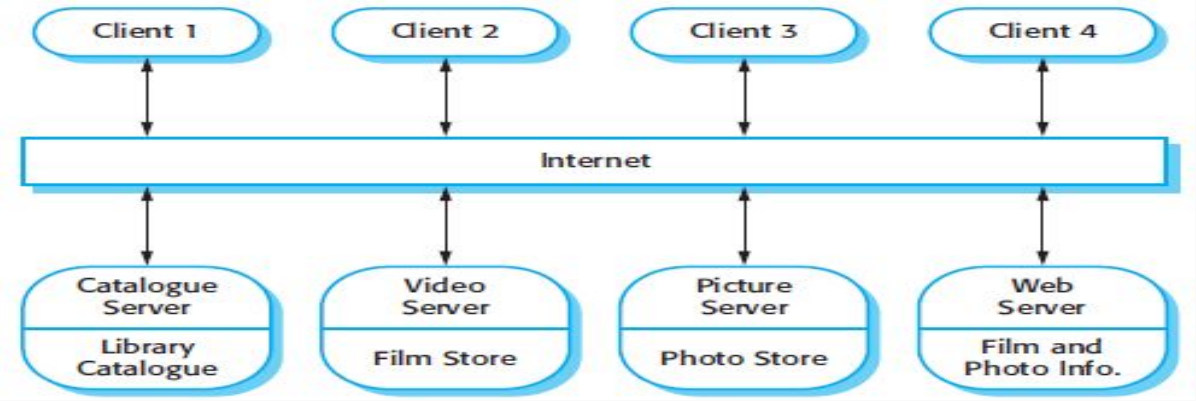
Benefits:

– It is easy to exchange one filter element for another with the same interfaces and functionality (support reuse).

– Systems can be easily maintained and enhanced: new filters can be added to existing systems and old filters can be replaced by improved ones

SW Architecture (Option #2): Client/Server

- Functionality organized into services, distributed across a range of components
  - **Server** = component that provides services and functionality
    - Print server, file server, code compilation server, etc..
  - **Client** = component that interacts with user and calls server
    - Through locally-installed front-end.
  - **Network** that allows clients to access these services.
    - Distributed systems connected across the internet.
- Example: A client—server architecture for a film library



Client-Server Model Characteristics

In a **client–server** architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.

Advantages

- Failure in one server does not impact others (Distributed architecture)
- Easy to add new servers or upgrade existing servers.

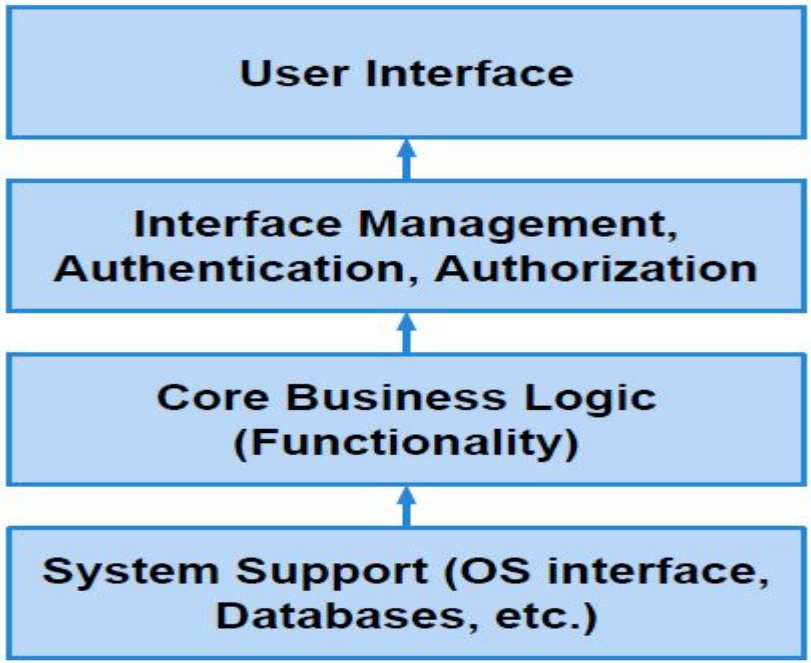
Disadvantages

- Performance is unpredictable (depends on system and network).
- Data exchange may be inefficient (server -> client -> server).
- Management problems if servers owned by others.

SW Architecture (Option #3): Layered

- System functionality organized into layers, with each layer only dependent on the previous layer.
- Allows elements to change independently.
- Supports incremental development.
- Each layer offers a service to the one on the top!

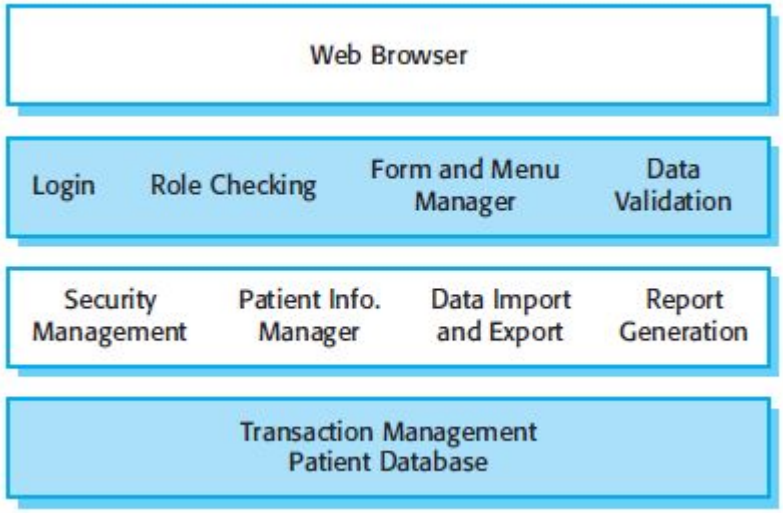
1. The top layer is responsible for implementing the user interface.
2. This layer includes form and menu management components that present information to users, and data validation components that check information consistency.
3. The third layer implements the functionality of the system and provides components that implement system ,e.g., security, patient information creation and updating.
4. Finally, the lowest layer, which is built using a commercial database management system, provides transaction management and persistent data storage.



Layered Model Characteristics

Advantages

- Allows replacement of entire layers as long as interface is maintained.
- There is reduced dependency because the function of each layer is separate from the other layers.
- Cost overheads are fairly low.
- Redundant features (authentication) in each layer can enhance security and dependability.



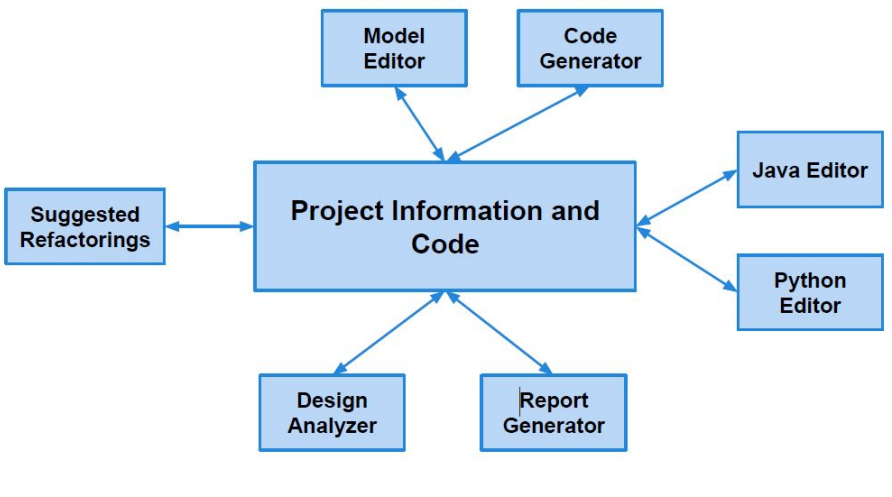
system for mental healthcare

Disadvantages

- Scalability is difficult because the structure of the framework does not allow for growth.
- They can be difficult to maintain. A change in a single layer can affect the entire system because it operates as a single unit.
- Performance can be a problem because of multiple layers of processing between call and return.

SW Architecture (Option 4): Repositories

- Subsystems often exchange and work with the same data. This can be done in two ways:
  - Each subsystem maintains its own database and passes data explicitly to other Subsystems.
  - **Shared** data is held in a central repository and may be accessed by all subsystems.
- Example: IDE



Repository Model Characteristics

Advantages

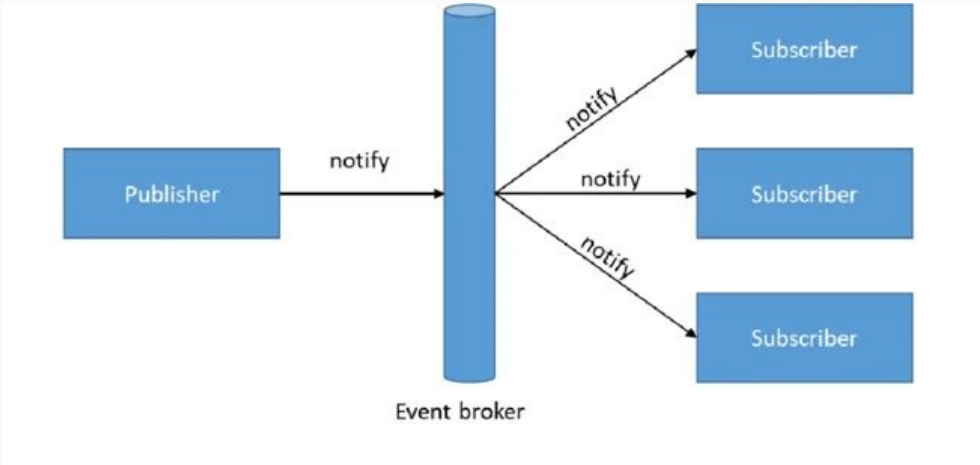
- Efficient way to share large amounts of data.
- Components can be independent. May be more secure.
- All data can be managed consistently (centralized backup, security, etc)

Disadvantages

- Single point of failure.
- Subsystems must agree on a data model (inevitably a compromise).
- Data evolution is difficult and expensive.
- Communication may be inefficient.

SW Architecture (Option 5): Publish/Subscribe

- **Pub/Sub (or Publish/Subscribe)** is an architectural design pattern used in distributed systems for asynchronous communication between different components or services
  - An event is broadcast to all subsystems, and any that can handle it respond.
  - Any subsystem that needs to respond to the event does do.
- Subsystems can register interest in specific events. When these occur, control is transferred to the registered subsystems.



pub/sub (publish/subscribe) Architecture Example

1. **News alerts**: A news organization might use a pub/sub system to distribute breaking news alerts to subscribers.
2. **Stock market updates**: A financial institution might use a pub/sub system to distribute stock market updates to clients.
3. **Social media notifications**: Social media platforms often use pub/sub systems to deliver notifications to users.

Advantages

- **Loose coupling** between components, making your system more modular and flexible.
- **High scalability** (in theory, Pub/Sub allows any number of publishers to communicate with any number of subscribers).

Disadvantages

- **Potential for message overload**: If a pub/sub system experiences a sudden surge in message volume, it can overwhelm subscribers or cause delays in message delivery.
- **Message durability**: In some pub/sub systems, messages are transient and may not be stored persistently.