

# System Architectures

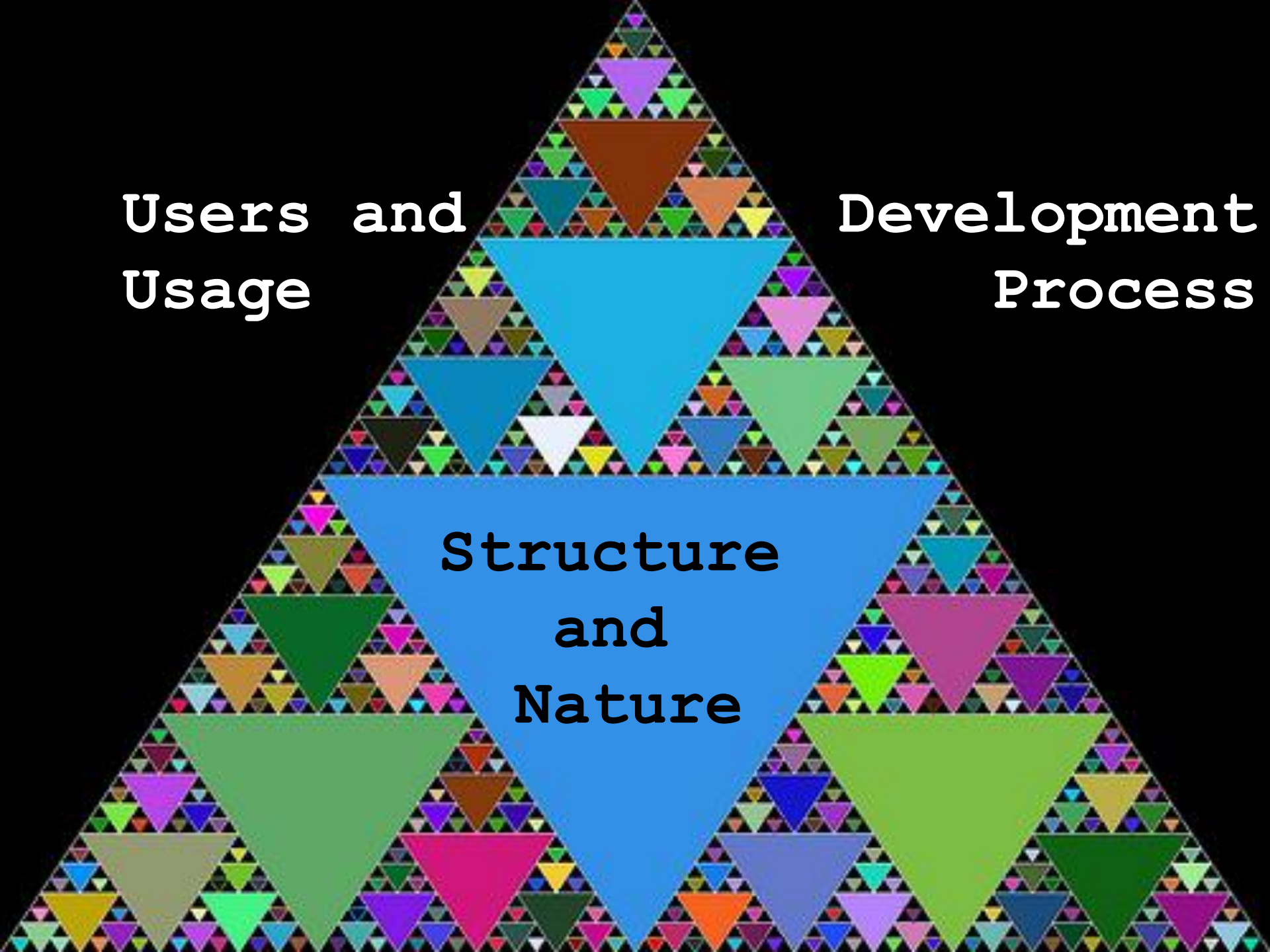
Internet Technology

ITECH

**Users and  
Usage**

**Development  
Process**

**Structure  
and  
Nature**



# Web Complexities

- **The nature and structure of the application**
  - The hypertext structure
  - The presentation and layout
  - Content usage and management
  - Service usage
- **The usage of the application**
  - Instant online access, with permanent availability
  - A wide variety of usually anonymous users
  - On a variety of devices
- **The development process behind the application**
  - by a multidisciplinary team including newbies and noobs
  - In a state of continuous development
  - A mixture of legacy technologies and immature technologies
- **Understanding Website Complexity?**
  - <https://web.eecs.umich.edu/~harshavm/papers/imc11.pdf>

# Structure & Nature

- The **hypertext structure** means there is a need to
  - avoid user disorientation and cognitive overload
  - provide multiple paths to support users with different requirements
  - provide a good superstructure including site maps, search facilities and guided tours
- The **presentation or user interface** must be
  - self-explanatory with no user manuals for web sites!
  - aesthetically pleasing and adaptable to different contexts
    - ideally self-adapting, but somehow there must be a version of the user interface which works in each context
- **Content delivery** must
  - be fast, up-to-date, consistent and reliable
  - be secure - particularly for financial transactions
  - adapt to different contexts in terms of how much can be delivered

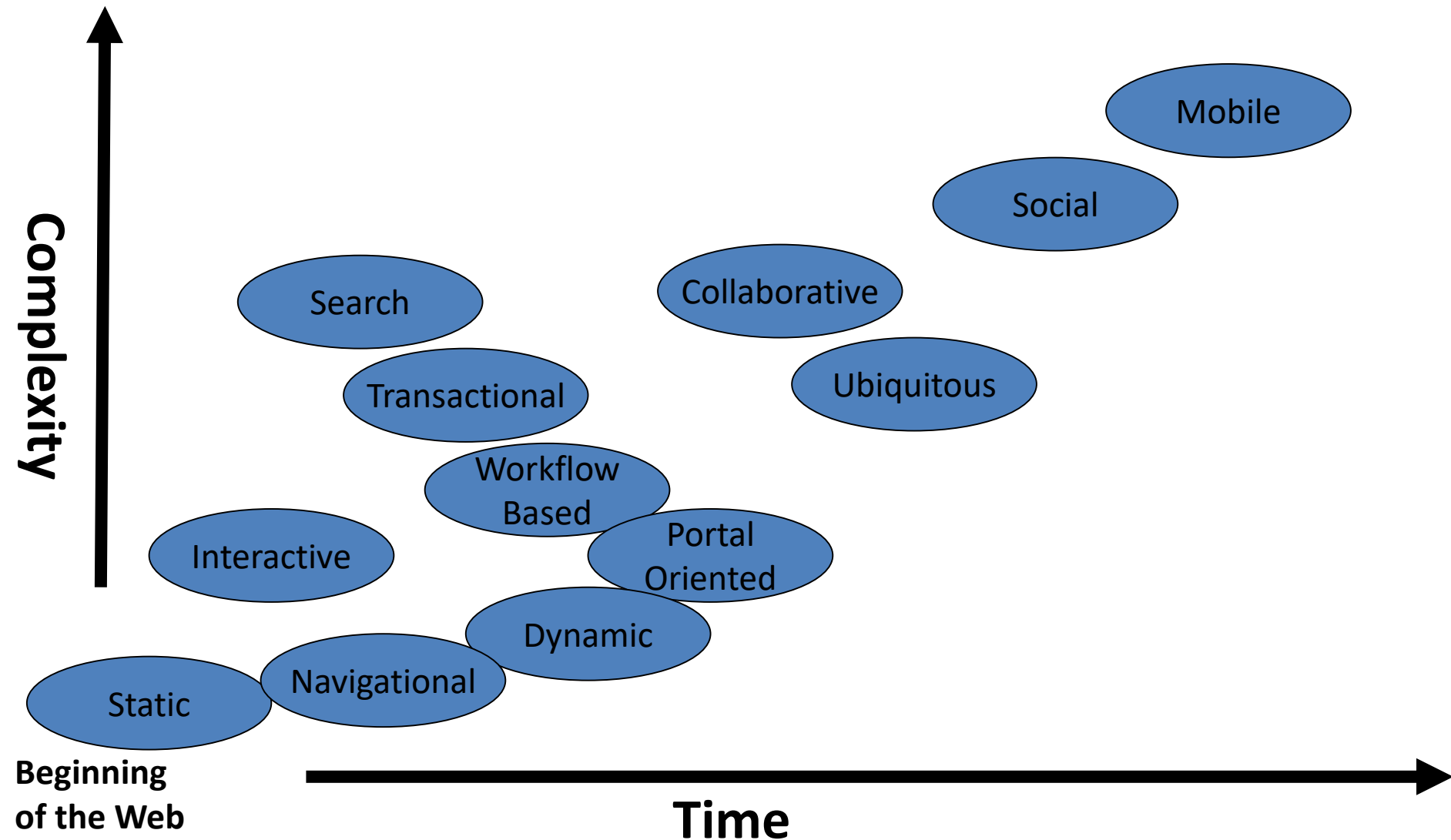
# Users and Usage

- Users expect **immediate** and **fast** availability
- They also expect **permanent** availability
- They also expect access on a **variety of devices**
  - quality of service becomes an issue
- They come from a very **diverse background** culturally and linguistically
- They have a low tolerance threshold for slow or hard to use sites
- There may be lots of them!

# Development Process

- **The nature of the developers**
  - Professional development is by a multidisciplinary team
    - programmers, hypertext designers, graphic designers, domain experts
  - Much development is by amateurs or inexperienced programmers
  - Development can be communal by a geographically distributed group
- **The development environment consists of**
  - a wide variety of technologies each of which is programmed differently
  - many of the technologies are immature but some are legacy technologies
  - each of the technologies has multiple competing products and upgrades to the site often mean a change of product
- **The development process**
  - follows no accepted internet application development methodologies
  - must be flexible and not rigid
  - Parallel development of components (and even of versions) is necessary
  - Agile processes seem valuable here

# Web App Evolution



# Types of Web App

- **Static**

- Originally web applications were just collections of hand built HTML pages
- These required manual update and introduced the possibility of inconsistency

- **Interactive**

- Forms, selection menus and radio buttons on web pages offer the possibility of selectively chosen pages generated by server-side programs
- CGI was the first technology but has been superseded by others (ASP, JSP, PHP, Cold Fusion, etc.)

- **Transactional**

- Forms also offer the capture of data and database storage at the server
- Although the data management may be separated from the internet server

- **Workflow-based**

- Support for functionality express as a sequence of pages reflecting a business process
- For instance, booking a flight



# Types of Web Apps

- **Portal Oriented**

- One point of access is given to multiple web sites
  - e.g. Virtual Shopping malls

- **Collaborative**

- Web sites which permit multiple users to share information management
  - e.g. Wikis such as Wikipedia

- **Social Web**

- Many sites provide a focal point for communities
  - e.g. Photo sharing sites, Facebook, etc.

- **Mobile and Ubiquitous**

- “Web” applications increasingly provide access by small, mobile and non-visual devices (i.e. phones) as well as data capture by sensors



Complexity

# Quiz Question (ID: 1164)

Building web applications involves an array of challenges and complexities; these include:

- A. The hypertext structure, the demand by users for instant availability, the variety of devices and the different web frameworks.
- B. The variety of devices, the simplicity of the HTTP protocol and the composition of the development team.
- C. The lack of an accepted development methodology, the hypertext structure and the homogeneous user populations.
- D. The composition of the development team, the immature and legacy technologies and the low expectations of users.

# DIFFERENT SYSTEM ARCHITECTURES

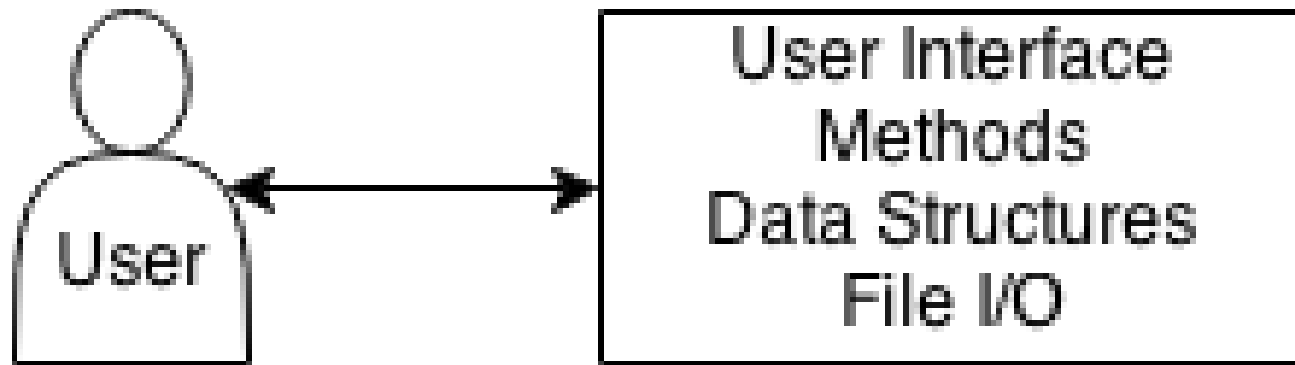
# System Architectures

- Every **component** of a system must be **designed**
- The **architecture** of the **system** shows the **blueprint** or plans of **how each component fits together**.
- Architectural diagrams can be at various levels
  - Data structure / algorithm / object level
  - Component / library level
  - Application level
- Various diagrams are needed
- Here we focus on the application level or the **high level system architecture**

# Monolithic Programs

- The main tasks that any application must support:
  - **user interface** management
  - the implementation of **algorithms** – the business logic
  - **information manipulation**
  - **data storage**
- A monolithic program in JAVA does all of these:
  - user interface with SWING
  - algorithms in methods
  - information manipulation in methods, e.g. sort methods
  - data storage using File I/O

# Single Tier Architecture



# Tiered Architectures

- The **structure of applications** have changed from:
  - **Monolithic** programs in which every aspect of the application is coded
  - To **tiered** structures in which different aspects are separated into different levels
- The **advantage** of the **tiered architectures** are
  - Each tier can be coded **separately** without the programmer having to deal with everything
  - The different tiers can be **distributed over the network** leading to increased efficiency
- But the **tiers** must **interact effectively**
  - There must be well defined interfaces between adjacent tiers
  - Internet protocols and database connection software do this well



# TWO TIER ARCHITECTURES

# Data Management

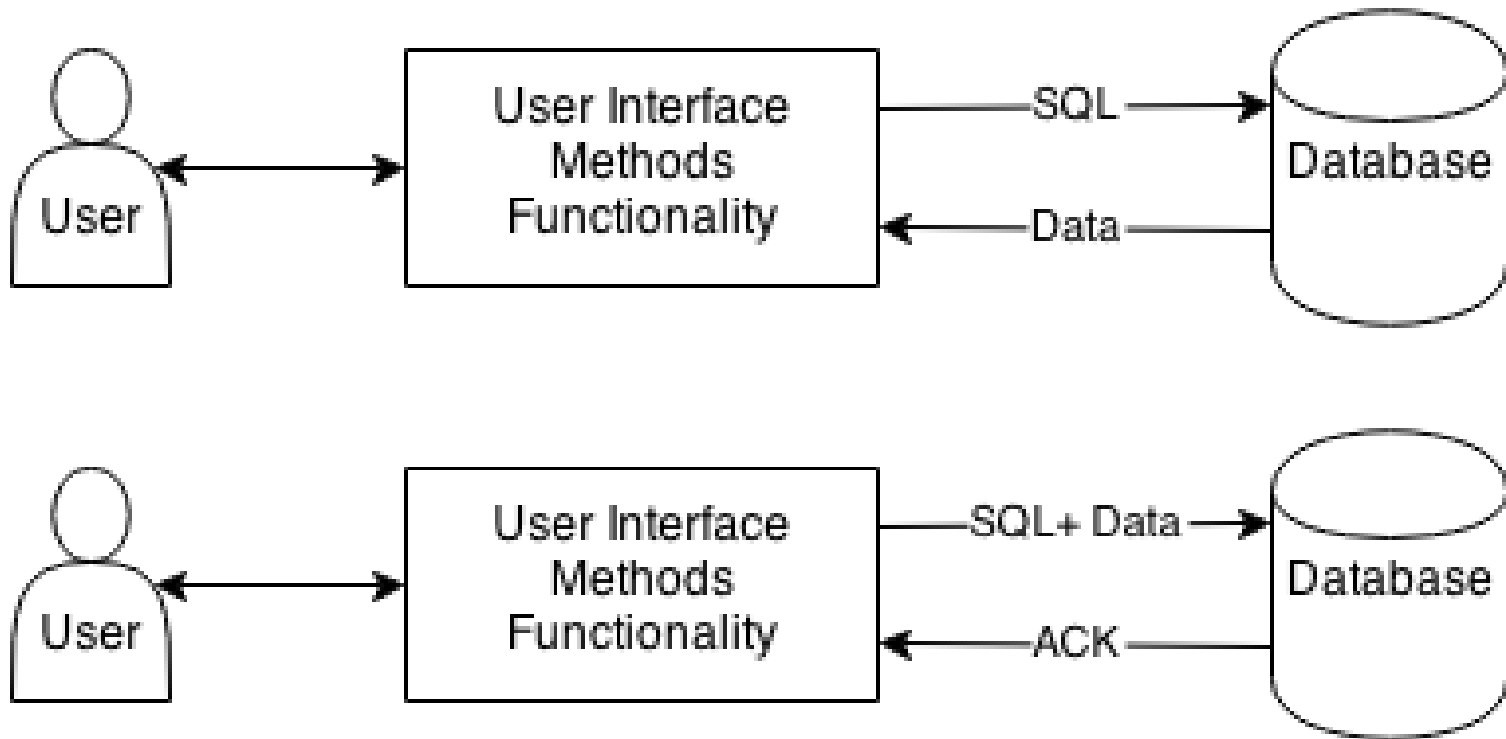
- Data and Information Management is common to many applications
  - Handling and dealing with large amounts of data is required by many applications
  - Access to a shared repository of data is highly beneficial to facilitate information flows between actors
  - Storing, retrieving, modifying and securing are common to many application
  - Databases Systems and Information Retrieval Systems provide ways to manage this data and information (efficiently)

# Data Management

- It is **hard/difficult/time-consuming** to write the code which accesses large amounts of data efficiently
- **Solution: Separate the concerns**
  - Let **Database Systems** to handle the data management
  - And then use a **Client application** to interact with the database:
    - The client acts like a database user sending in queries and updates and making use of the result
    - The standard mechanism is to code **SQL statements** as strings inside the program

# Client-Server Architecture

**Fat Client:** This then is the standard architecture for getting an application (e.g. Java) to work on top of a database.

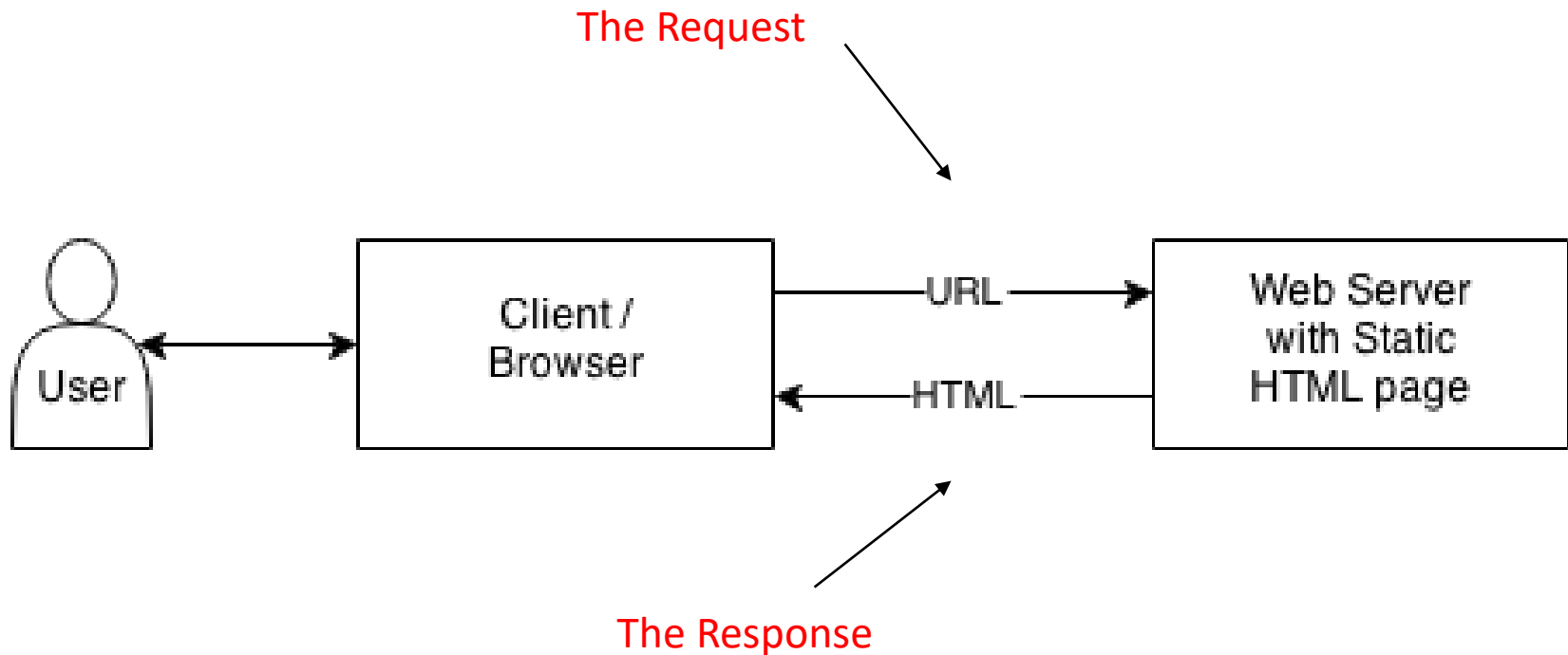


# Static Web Application

- Static web sites consist of a set of hyper-linked HTML files
- Each HTML file describes one page of the web site
- Each page includes one or more links to other pages
- If a user clicks a links or enters a URL then a request is sent to the web server identifying the next page to load

# Two Tier Architecture

**Thin Client:** This is the standard architecture for serving up static content on the web.

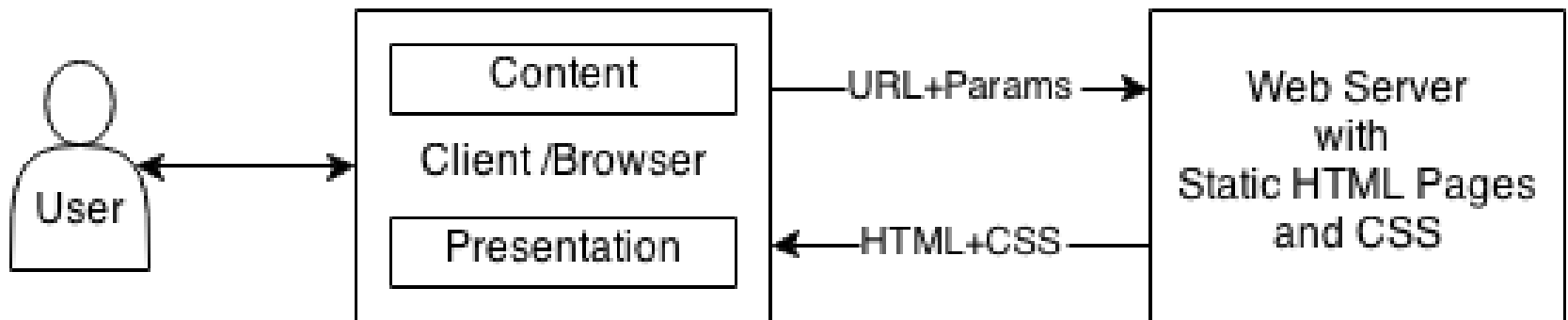


# Style and Substance

- Initially, the HTML files contained all the information on how to render the page
  - The text/data (html) and the style (css),
- With lots of pages there is lots of overhead in creating and maintaining the site.
- **Solution: Separate the concerns**
  - Use external Cascading Style Sheets which act upon the different elements in the html.
  - The web server/application returns a html file and css file.
  - i.e. one to hold the Content and one to represent the Style

# Two Tier with Layers

**Extended Thin Client:** Layers within the client tier handles the content and look/feel.



The **content layer** represents/stores the data

The **presentation layer** renders the data



# Repetition of Structure

- On a site, many of the pages have a similar structure and a lot of repeated content
  - i.e. headers, footers, navigation, etc
- This leads to a maintenance nightmare
  - (or a job for life)
- What should we do when we encounter such a situation?

# Repetition of Structure

- **Solution: use a template and decompose repeated elements i.e. separate the concerns**
  - It is better to have a program which generates pages by merging a template with the specific page data
  - The appropriate data depends on the URL and the parameters in the URL
  - This enables the provision of dynamic content
    - but requires a more sophisticated architecture

# Look for Commonalities

- **Anywhere, anytime, anyplace you see repeated code look to refactor**
  - Through methods, objects, templates, applications
- **Train of Thought**
  - **Extract** the common parts
  - **Parameterise** the parts which change
  - Create a **template**
  - Use a program to use the template given a set of parameters

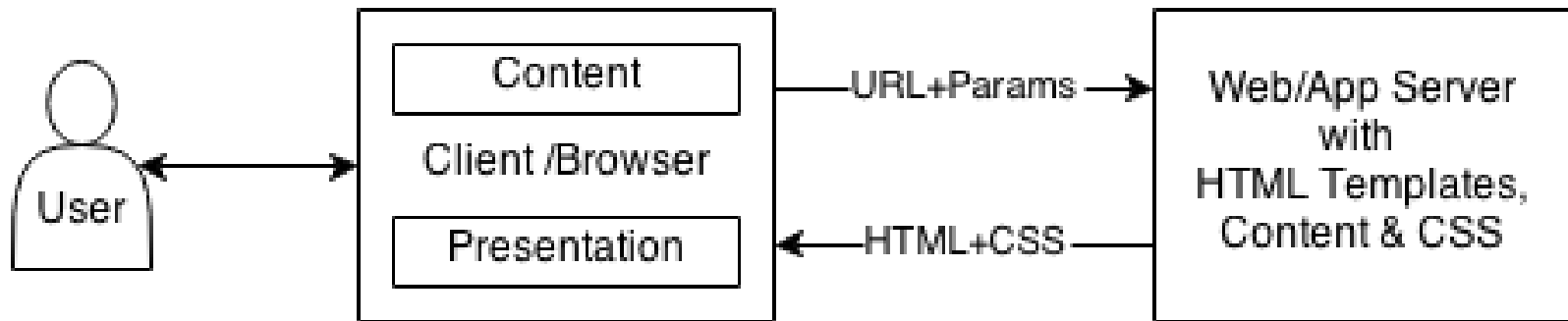
# Handling User Interaction

- Clicking links only provides the link information
  - Forms take input but what happens when you hit the submit button?
  - This enables the provision of Interaction
  - But this requires the server side program to be able to deal with the data from the form data
- HTTP provides methods for sending the data entered by the user to the program
  - GET and POST

# Two Tier with Layers & Servers

## Extended Thin Client and Extended Server:

Layers within the client tier handles the content and look/feel. The server now houses a web server and an application server.



**Web Server** handles incoming requests and send outgoing responses, routing them to/from the correct application server

**Application Server** is typically a script that dynamically generates a response given a request

# Two Tier Issues

- Distributed applications with a **two-tiered architecture** distinguish only:
  - **client** processes – control the application logic and the user interface
  - **server** processes – supply resources, such as data, to the clients
  - however this:
    - puts a lot of the load on the client
    - ties the client software to the database software so changes in each affect the other

# Further Tiered Architectures

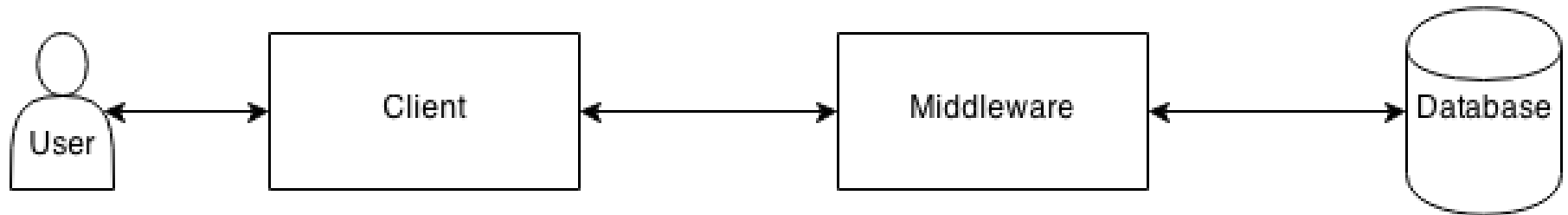
- An improvement is to separate the application into **three (or N-) tiers**:
  - **presentation** processes deal exclusively with the user interface – this might, for instance, be through the use of a browser or other user agent
  - **application** processes deal with the logic of the application, queries, calculations, etc. – there is either one (3-tier) or more (N-tier) of such processes
  - **data source** processes supply the data from a database (binary) or a file (for instance, XML)

# THREE TO N TIER ARCHITECTURES



# Three Tier Architecture

**Basic Three Tier Architecture:** where the client handle the interaction with the user, the middleware handles the application logic, and the database stores the data.



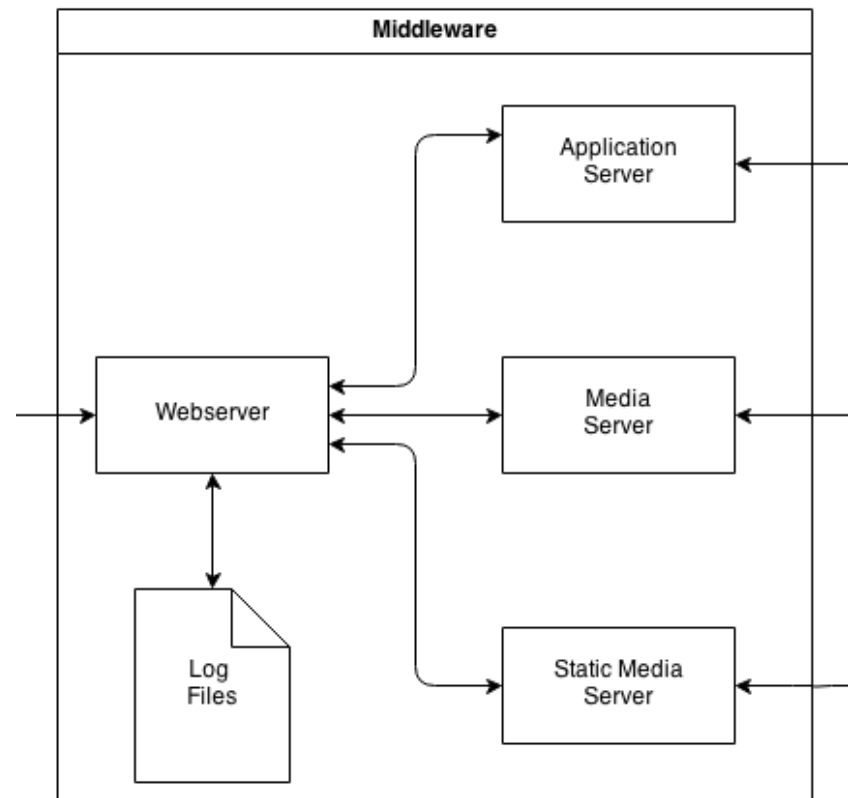
**Database:** persists and manages the data associated with the application.

# Middleware (Zoom in)

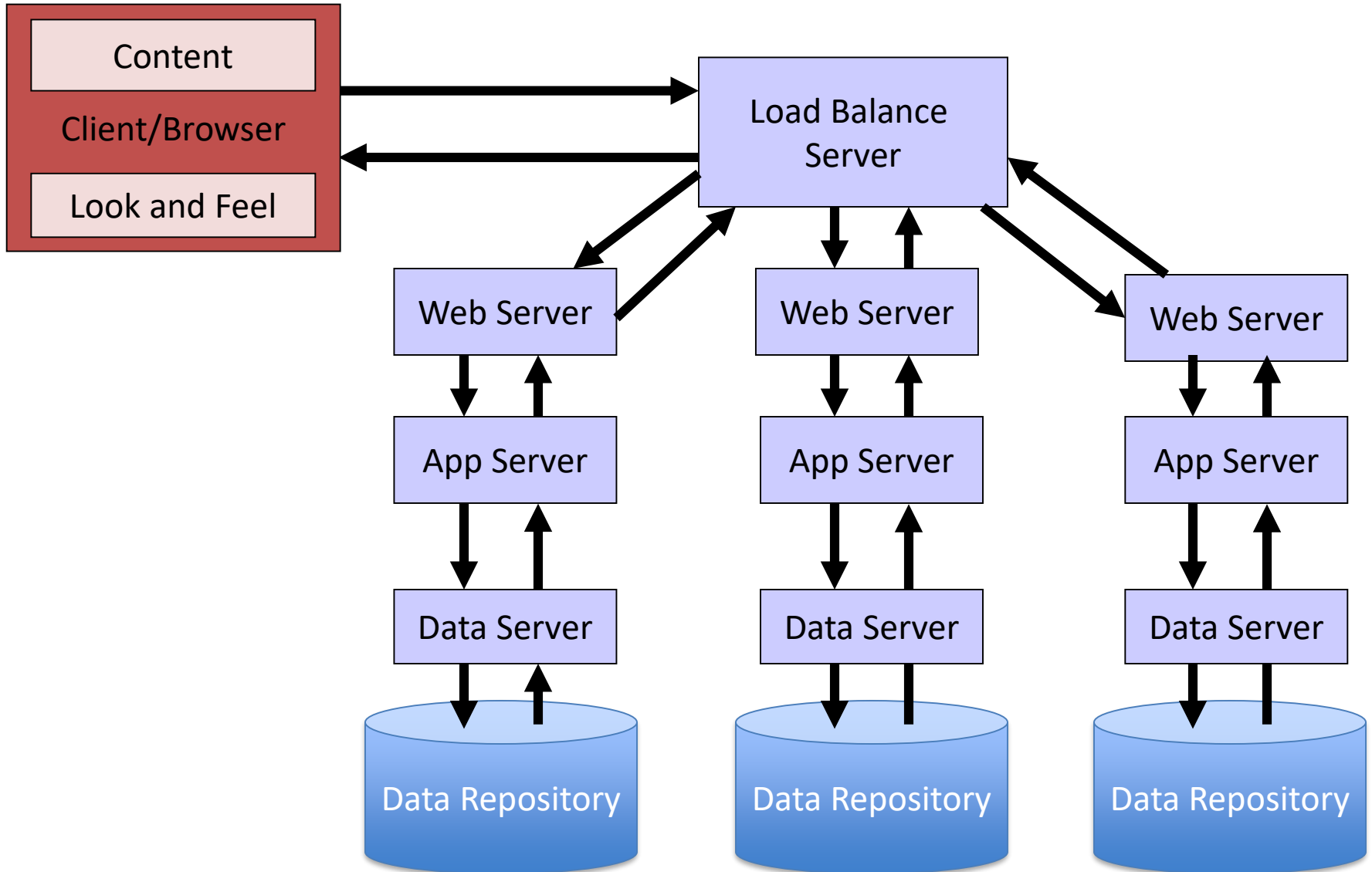
Within the middleware tier, usually there is a webserver, an application server, and potential media servers.

**Webserver:** handles incoming responses, directing them to the appropriate server.

Servers could be on the same machine or different machines.



# N-Tier Architecture



# Load Balancing

- There are two main ways to balance the load
  - Using a **Domain Name Server** such that when a URL is resolved, it rotates through a series of IP addresses that route the message to that machine.
  - Using a **Load Balancing Server**, which farms out the requests to available machines.
    - Machines in the farm inform the LB Server of their load so that the load can be balanced.



# SUMMARY

# Tiered Architecture Benefits

- Tiers enable the separation of concerns
- Tiers encapsulate complexity
  - they can be broken down into layers or into sub-tiers
- Tiers can be distributed across a number of machines
  - Providing flexibility
- Tiers can be replicated across a number of machines
  - Providing scalability

# Quiz Question (ID: 1164)

Which option from the following list best describes a 1-tier architecture?

- A. Difficult to design but easy to maintain
- B. Easy to design and good for web applications
- C. Easy to design but not scalable
- D. Hard to design but scalable





# System Architectures Diagrams

# Top-Down vs. Bottom Up

Separates the low level work from the higher level abstractions

Leads to a modular design

Development can be self-contained (tiered)

Emphasizes planning and system understanding

Coding is late, and Testing is even later

Skeleton code can show how everything integrates



Coding begins early and so Testing can be performed early

Requires really good intuitions to determine functionality of modules

Low level design decisions can have major impact on solutions

Risks integration problems – how do components link together

Often used to add on to existing modules

# Top Down Design

Starting from a **high level design** is useful because:

- it helps to describe the system at a level which makes the **goals, scope** and **responsibilities** clear
- the abstraction provides a tool for communicating the design
- permits the specific technologies to be **chosen late** or to be **changed**
- **maintenance** and **re-usability** also profit
- **accessibility** variants can be considered *a priori*

# Bottom Up Design

- Piecing together components to give rise to more complex systems, thus making the original elements sub-systems of the **emergent system**
- Specific and basic individual components of the system are first developed
- These elements are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed
- This strategy often resembles a "**seed**" model, by which the beginnings are small but eventually grow in complexity and completeness.

# Diagram and Design

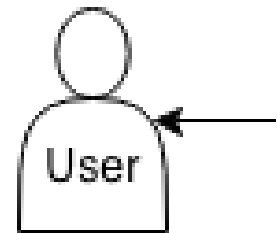
- Architects need to communicate to developers how the application and its components fit together and who is responsible for what
- The designs also serve as a communication tool with the client
- It is important to be able to draw and read such diagrams especially when projects become large and complex.

# Notation for System Architecture

- **Modified Dataflow Language** where we have the following entities:
  - User
  - Client
  - Middleware
  - Database/Datastore
  - Logs/Files
  - External Service/Application
  - Communications/DataFlows

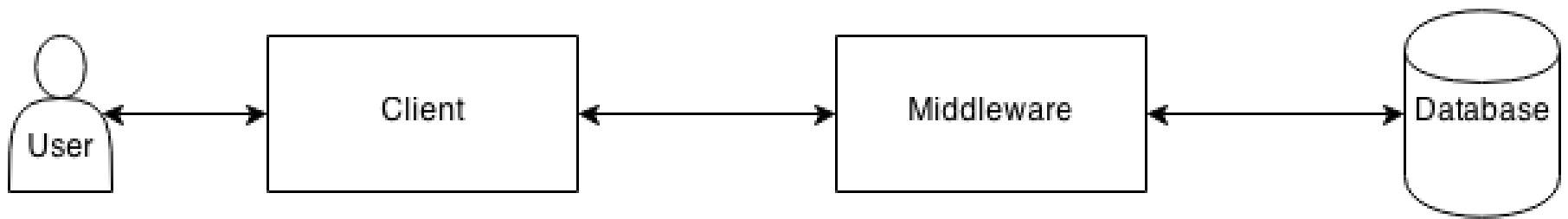
# User

- The user or client instigates and interacts with the services or applications provided
- There are various clients including:
  - End Users (of varying abilities)
  - Administrators,
  - Developers,
  - Agents,
  - Other systems,
  - etc



# Client

- The client and the interface presented takes on many forms and can vary greatly:
  - Web Browser on a PC, Tablet, Mobile, etc
  - An API for other systems, agents, developers, etc
  - Devices and Robots
  - Sensors



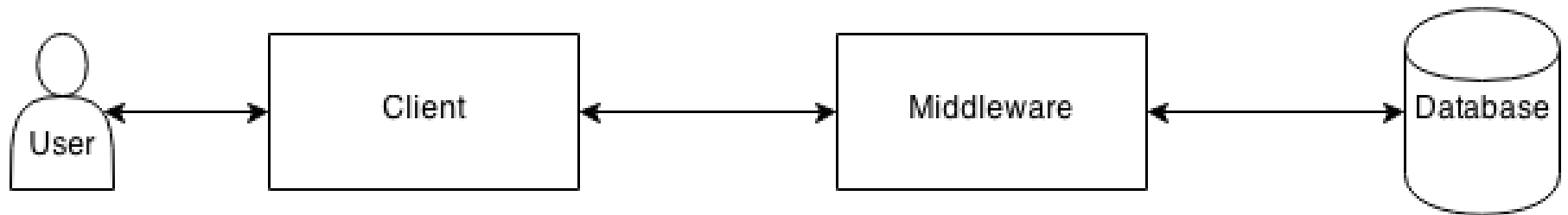


# Middleware

- The middleware houses an array of possible components from:
  - Domain Name Servers
  - Load Balancing Servers
  - Web Servers
  - Application Servers
  - Caching Servers
- Typically the first three are predefined or configured using standard software
- The Application Server is what is mainly of interest i.e. what needs to be developed.

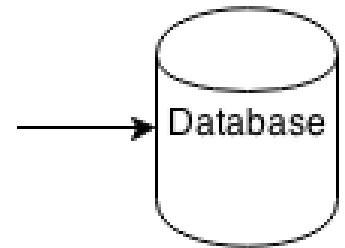
# Middleware

- Often represented as a single component that process requests between the client and database.
  - Though encapsulates a number of other components



# Databases

- A database server is usually employed to handle the data management side of applications.
  - I.e. Postgres, SQLServer, MySQL
- While the system is usually already in existence, it needs to be configured
  - i.e. the database tables have to be defined and populated.
  - To specify this part more precisely ER Diagrams can be used.



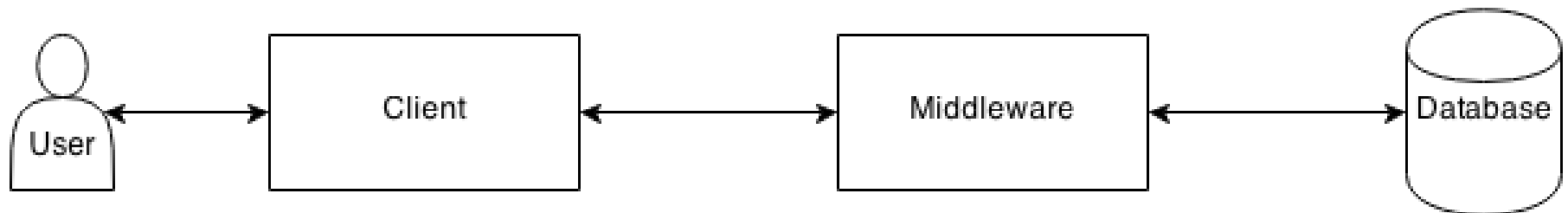
# Logs and External Services

- Logs represent data sinks
  - The application outputs data but does not read it back, directly.
- External Services represent applications and services that are used by the application
  - They provide an API or interface of some kind that which can be used to interact with the service.



# Technology and Devices

- For each box, we can state/specify the technology/device used, i.e.
  - Client: Web browser on a mobile device, using HTML/CSS/JS
  - Middleware: Apache Web Server, with an Application Server built using Django
  - Database: MySql Database Server



# Data flows

- Arrows are used to denote the flow of information
  - The direction of the arrow denotes the direction of the communication.
  - Most communications are both ways, where a request is made, followed by a response.
  - The client makes the requests.
  - This show how the entities are related.

