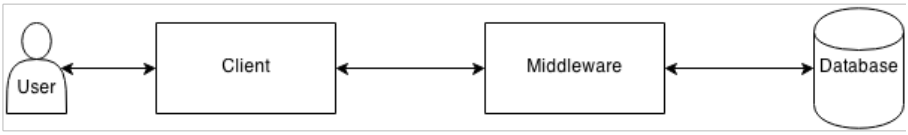
# THREE TO N TIER ARCHITECTURES

## 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 Tier Architecture

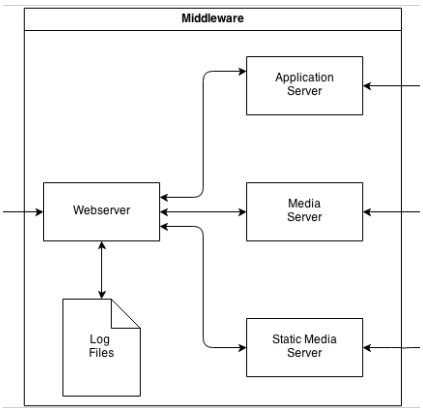
- **Basic Three Tier Architecture:** where the client handles the interaction with the user, the middleware handles the application logic, and the database stores the data




- **Database:** persists and manages the data associated with the application
- **Separation of Concerns:** This architecture further separates out responsibilities (presentation, logic, data)

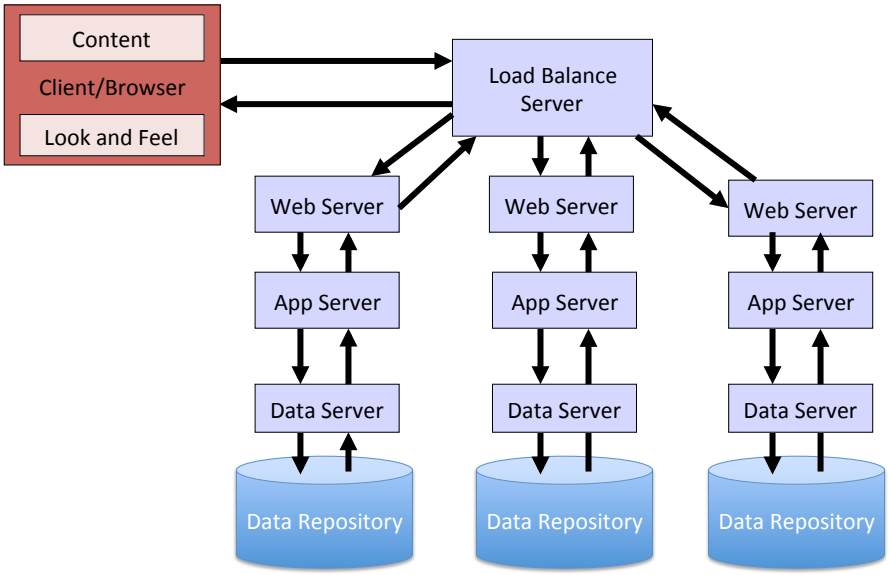
# Middleware (Zoom in)

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

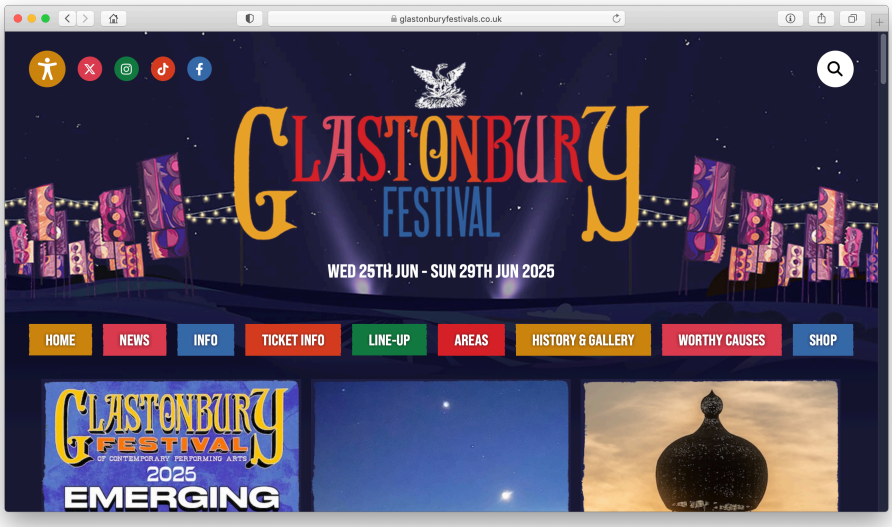



- **Webserver**: handles incoming requests, directing them to the appropriate server
- Servers could be on the same machine or different machines

# N-Tier Architecture




# Load Balancing



# 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 LBServer 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
  - Providing more security (as clients do not interact directly with the database)
- Tiers can be replicated across a number of machines
  - Providing scalability

## System Architecture Diagrams

## Top Down Design

- Starting from a **high level design** is useful because:
  - it is helps 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

## Bottom Up Design

- Piecing together components to give rise to more complex systems, thus making the original elements sub-systems of the emergent system
- The most 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
- What are the advantages and disadvantages?

# 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 new module to existing system

# Diagram and Design

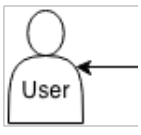
- Architects need to communicate to developers how the application and its components fit together and who is responsible for what
- The designs also serve as a communication tool with the client
- It is important to be able to draw and read such diagrams especially when projects become large and complex

# 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

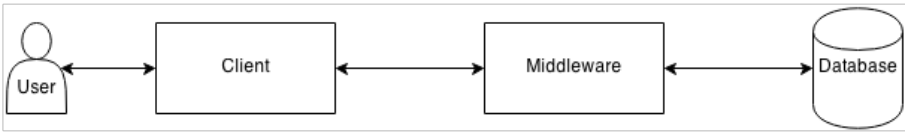
# User

- The user or customer instigates and interacts with the services or applications provided
- There are various types of users including:
  - end users (of varying abilities)
  - administrators
  - developers
  - 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

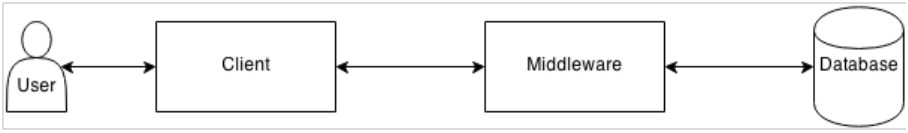



# Middleware

- 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 brokers requests between the client and database.
  - Though encapsulates a number of other components



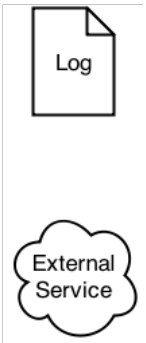

# Databases

- 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 can be used to interact with the service

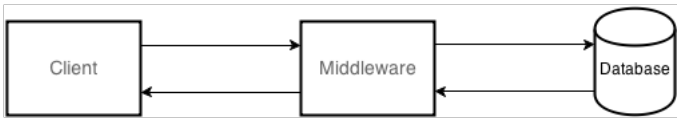



## Technology and Devices

- For each box, we could choose to specify the technology/device used, e.g
  - 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
  - This shows how the entities are related




# Information Architecture

Web Application Development 2

# Types of Architecture

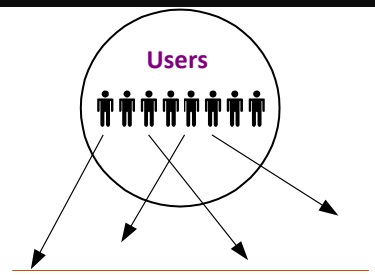
**System Architecture**

**Information Architecture**

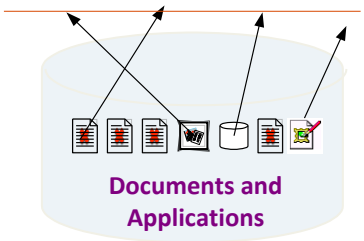
# Information Architecture

1. The **structural design of shared information** environments
2. The combination of **organisation**, **labelling**, **search** and **navigation systems** within web sites and intranets
3. The **art** and **science** of shaping information products and experiences to support **usability** and **findability**
4. An emerging discipline and community of practice, focused on bringing **principles of design and architecture** to the **digital landscape**

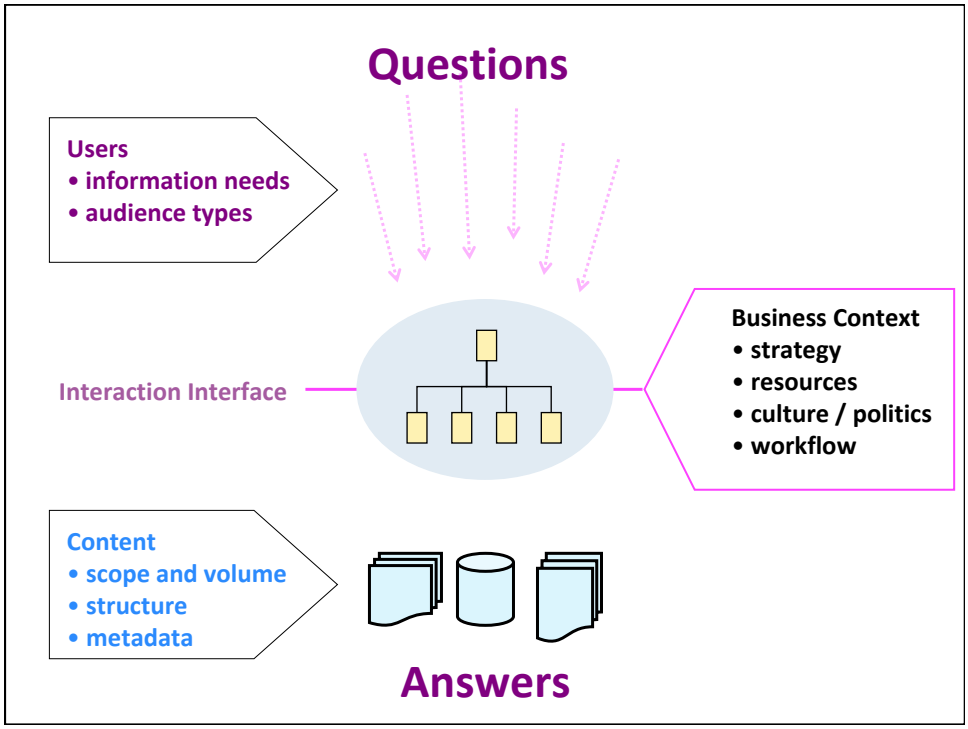

# Communication Chasm

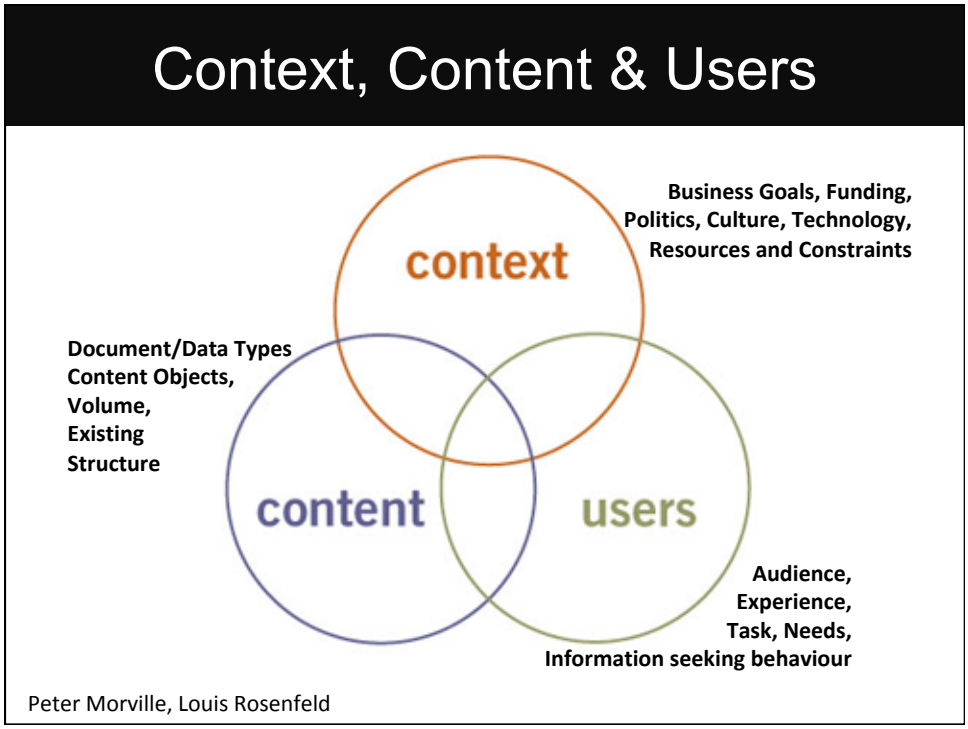



Communication Chasm



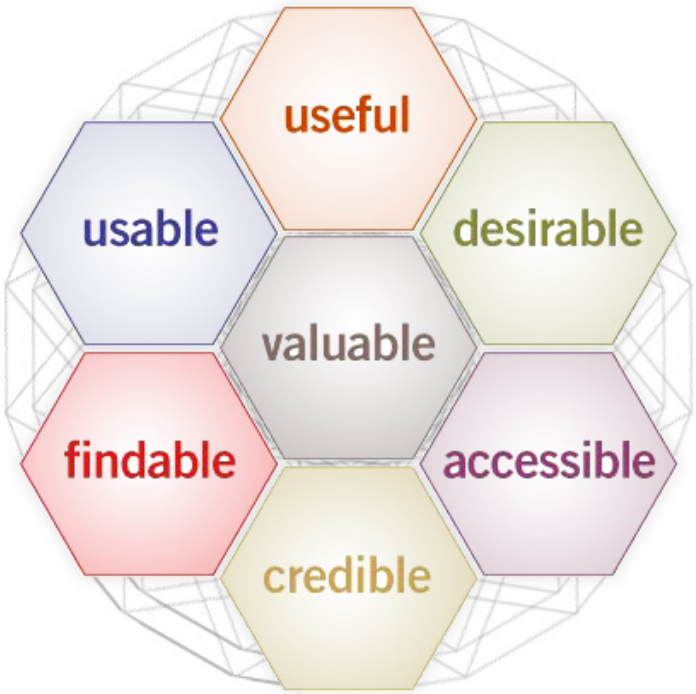

- Key Problem:
  - Users have particular needs, goals, aims and objectives
  - The organisation may/may not provide answers, suggestions, solutions, options, etc
- How do we bridge the gap between user and organisation?


Questions
Users
• information needs
• audience types
Interaction Interface
Business Context
• strategy
• resources
• culture / politics
• workflow
Content
• scope and volume
• structure
• metadata
Answers

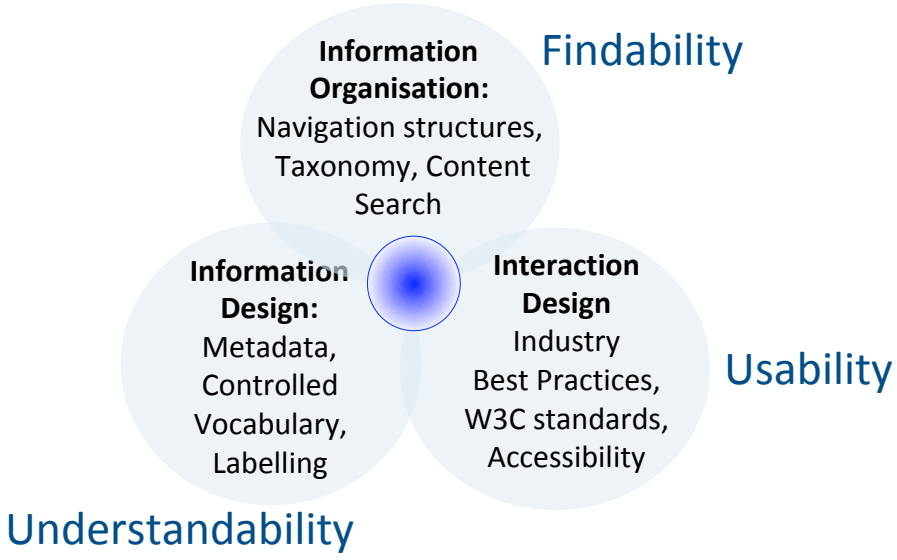

Context, Content & Users
context
content
users
Business Goals, Funding, Politics, Culture, Technology, Resources and Constraints
Document/Data Types Content Objects, Volume, Existing Structure
Audience, Experience, Task, Needs, Information seeking behaviour
Peter Morville, Louis Rosenfeld

# Why is IA Important?

- Cost of **finding**
  - time, frustration
- Cost of **not finding**
  - bad decisions, alternate channels
- Cost of **construction**
  - staff, technology, planning, bugs
- Cost of **maintenance**
  - content management, redesigns
- Cost of **training**
  - employees, turnover
- Value of **brand**
  - identity, reputation, trust



# IA Components




# Top Down Design

- Top Down Design (from an Information Architect's Perspective) is designing for when a user arrives at the main page of the site
- Typical questions the users have in mind when they arrive are:
  - Where am I?
  - I know what I want, how do I search for it?
  - How do I get around this site?
  - What useful, important, unique about this site?
  - What's available, what's happening?
  - How can I get help, contact a human, get their address?

# Bottom-Up Design

- Bottom Up Design is catering for when the user lands somewhere in your site
  - typically via a search engine
- Typical questions are:
  - Where am I?
  - What's here?
  - What else is here?
  - Where can I go from here?

# Information Systems

- Information Retrieval Systems
  - Users query the search engine via the search interface
  - Results are ranked and returned
- Navigation Systems
  - Global Navigation
  - Local Navigation
  - Contextual Navigation