COSC 499: Capstone Software Engineering Project

System Architecture

- A conceptual model the structure and behavior of a system
  - Identifies the major components and subcomponents
  - Identifies how these components interact with each other
  - Can involve hardware and software components
- Software engineers use formal languages to describe and document system architectures
  - E.g., data flow diagrams (DFDs) describe how data moves from one process to another
- Purpose
  - Tool for communication among different stakeholders
  - Ensures business goals and stakeholder requirements are met
  - Documents changes to the system

Examples of Machine Translation (MT) Approaches

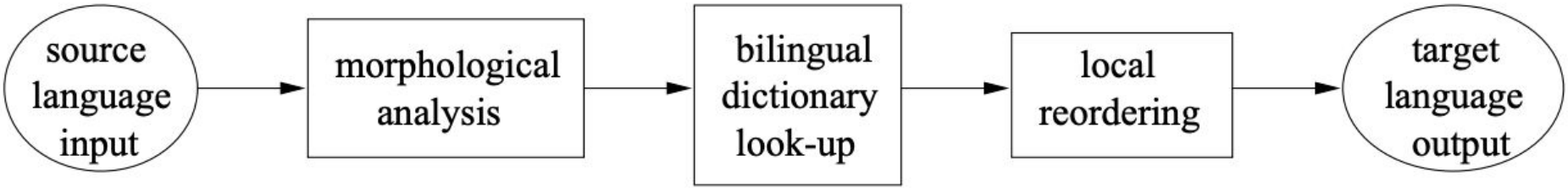


Figure 1: Direct MT System

Examples of Machine Translation (MT) Approaches cont.

Overcomes language-specific idioms

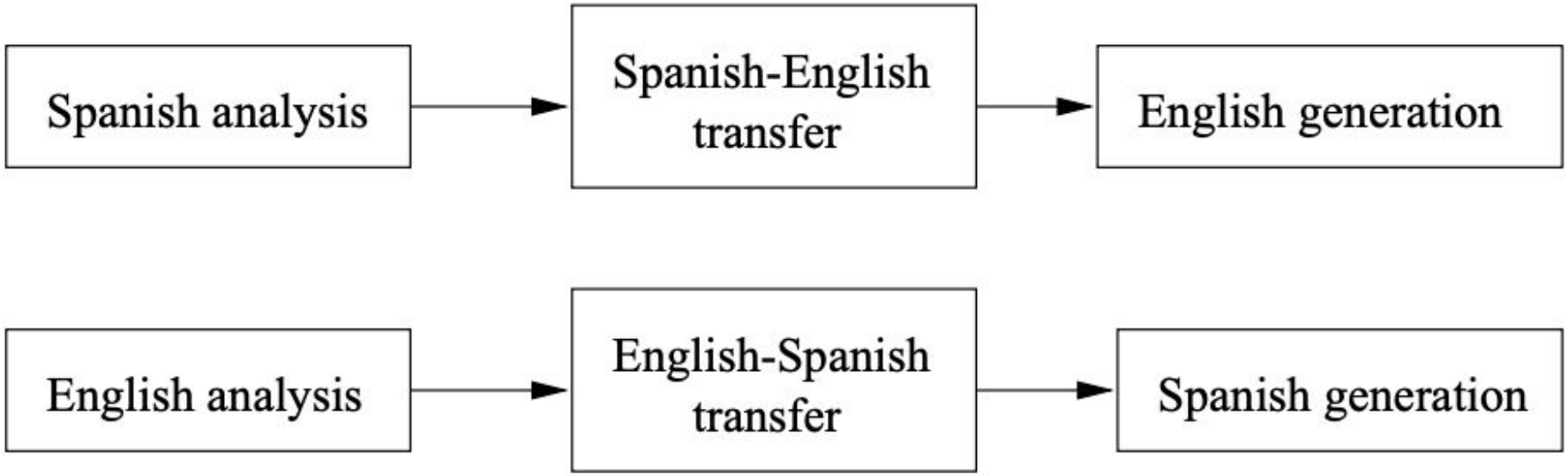
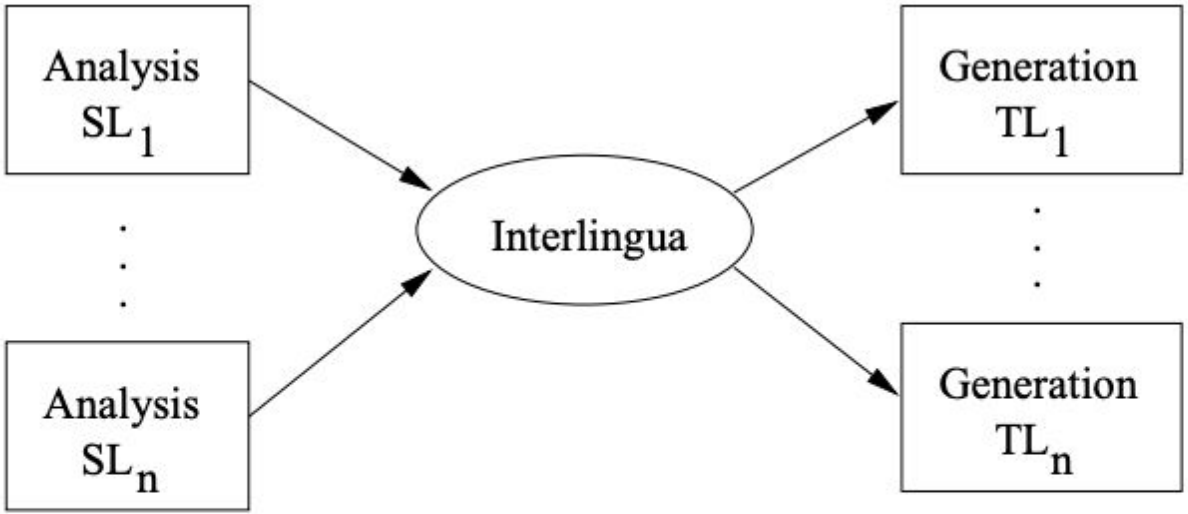


Figure 2: Transfer MT System

Examples of Machine Translation (MT) Approaches cont.

Adopts the idea of universal grammar

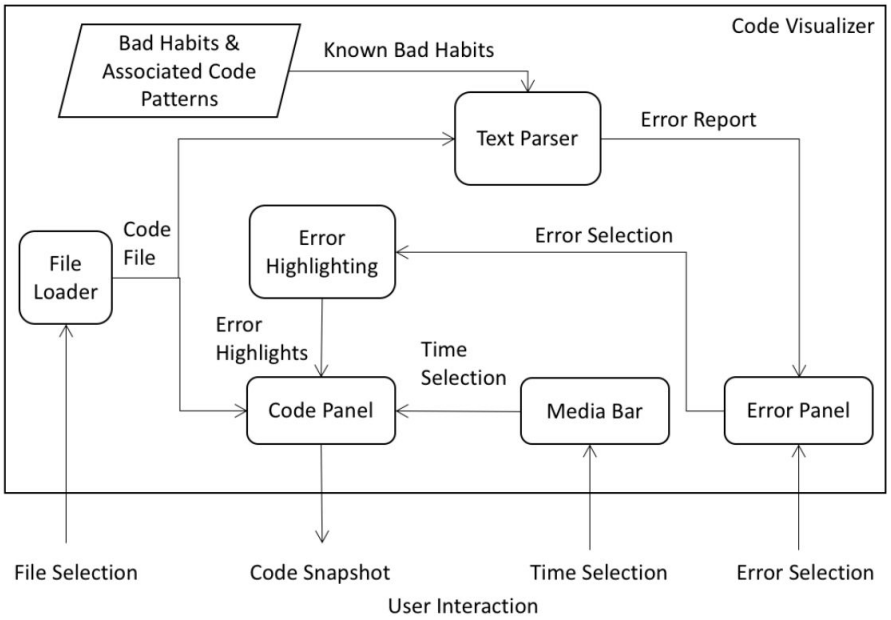


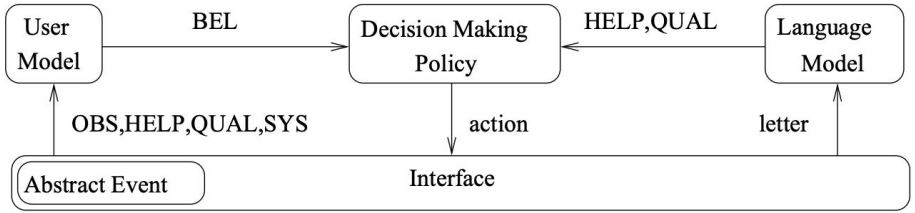
SL = source language

TL = target language

Figure 3: Interlingua MT System

Examples of Specific Systems

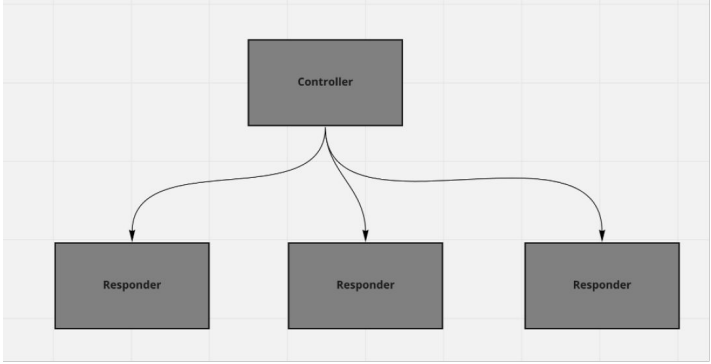


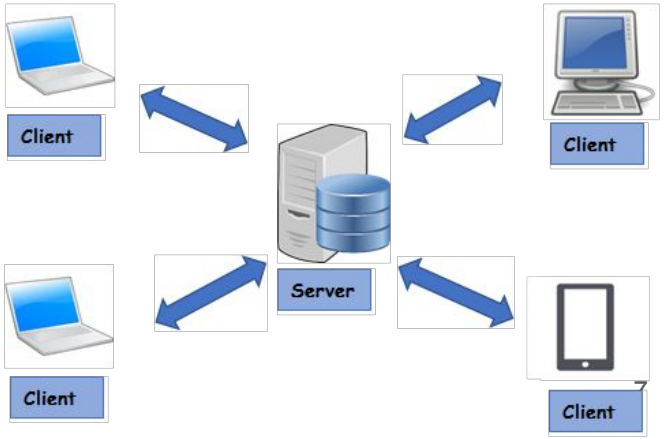


- labeled data flow
- major components
- component relationships
- interactions with other components

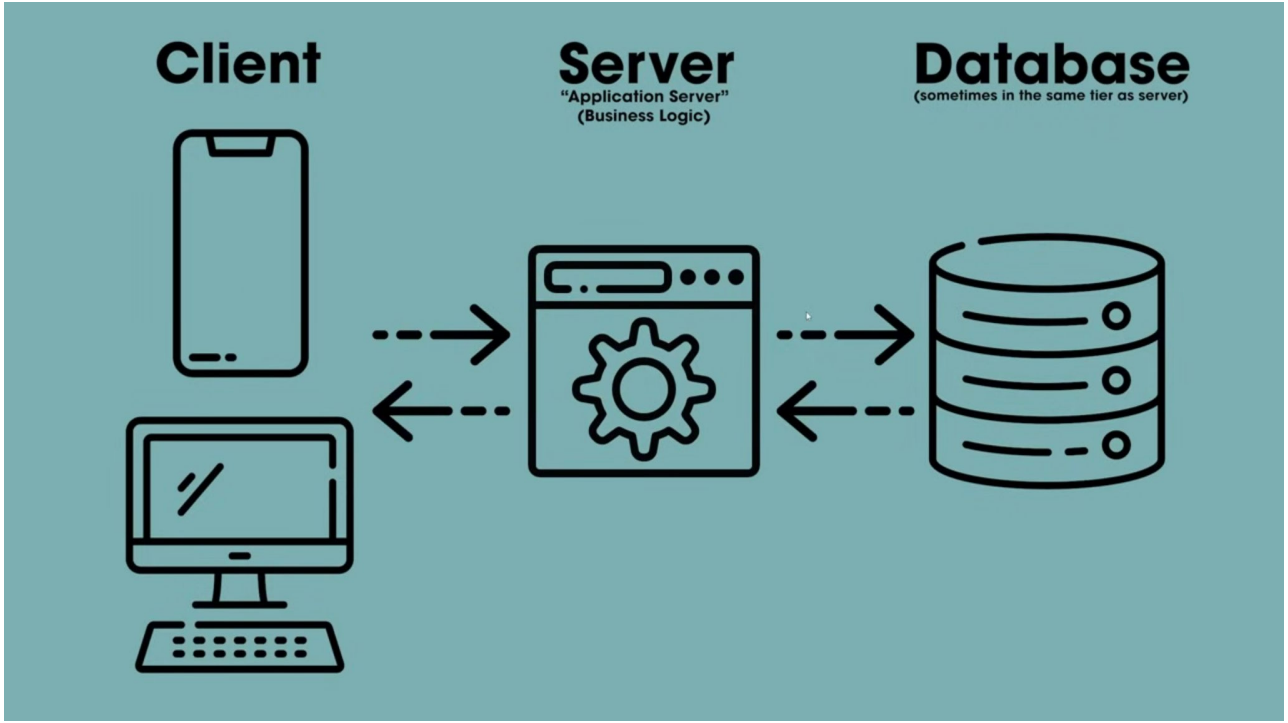
Traditional Examples of System Architectures

- Controller-responder architecture
  - Initially called Master-Slave
  - Controller distributes work to identical responders
  - Results are then compiled by controller
- Client-server architecture
  - Control rests with clients
  - Server handles central computing and data storage
  - Decentralized variation of this is peer-to-peer architecture (e.g. napster)



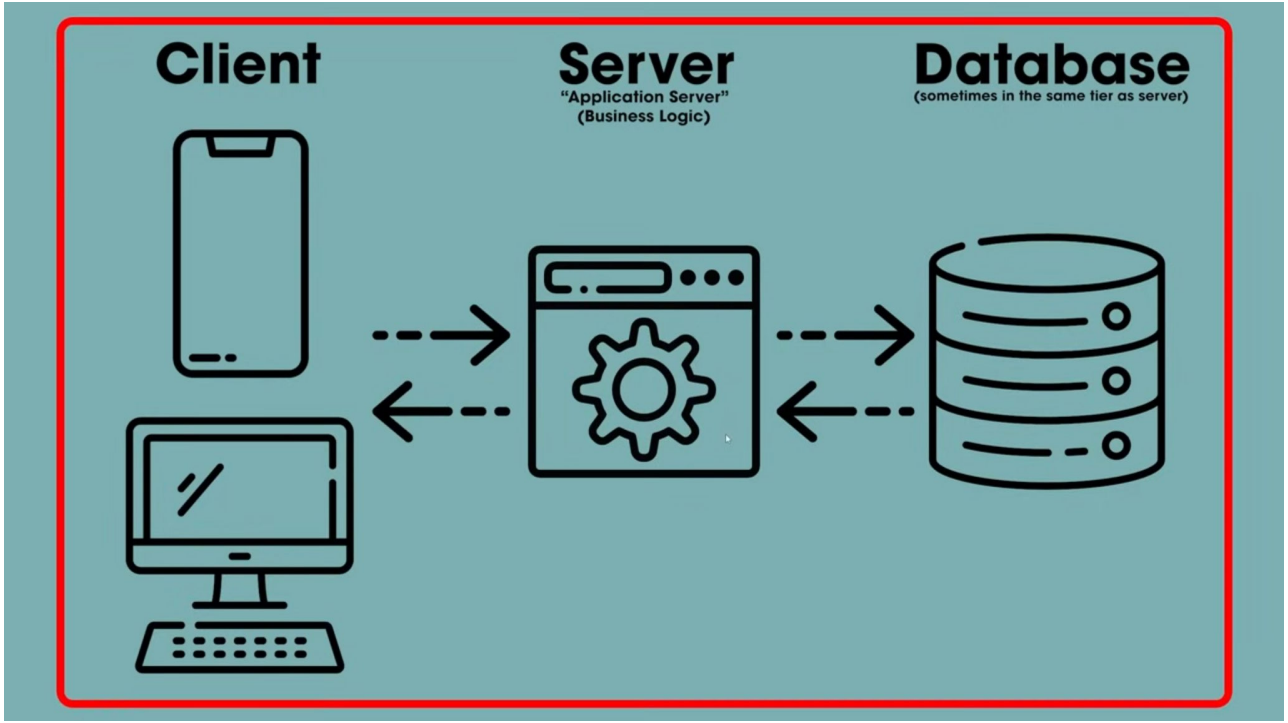


Client-Server Architecture in Web Apps



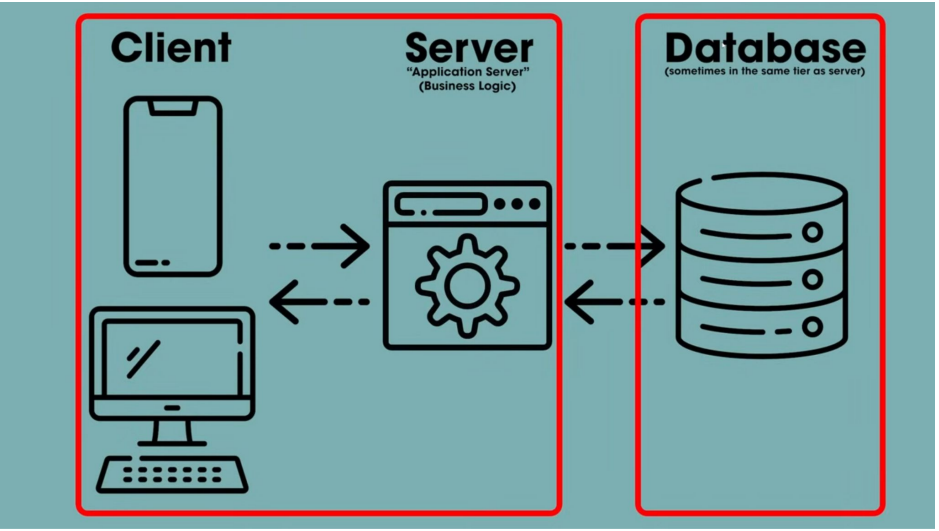
Front-end Back-end Database

1-Tier Client-Server Architecture in Web Apps

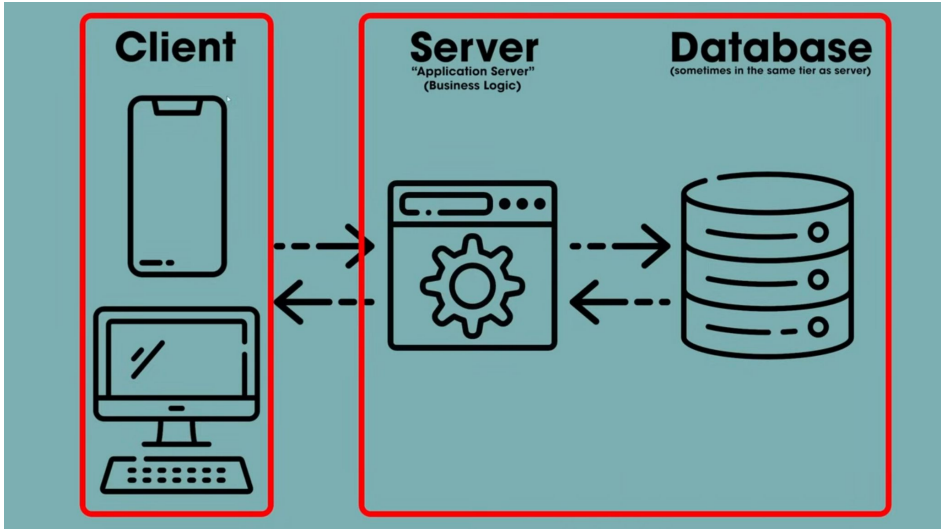


Front-end Back-end Database

2-Tier Client-Server Architecture in Web Apps

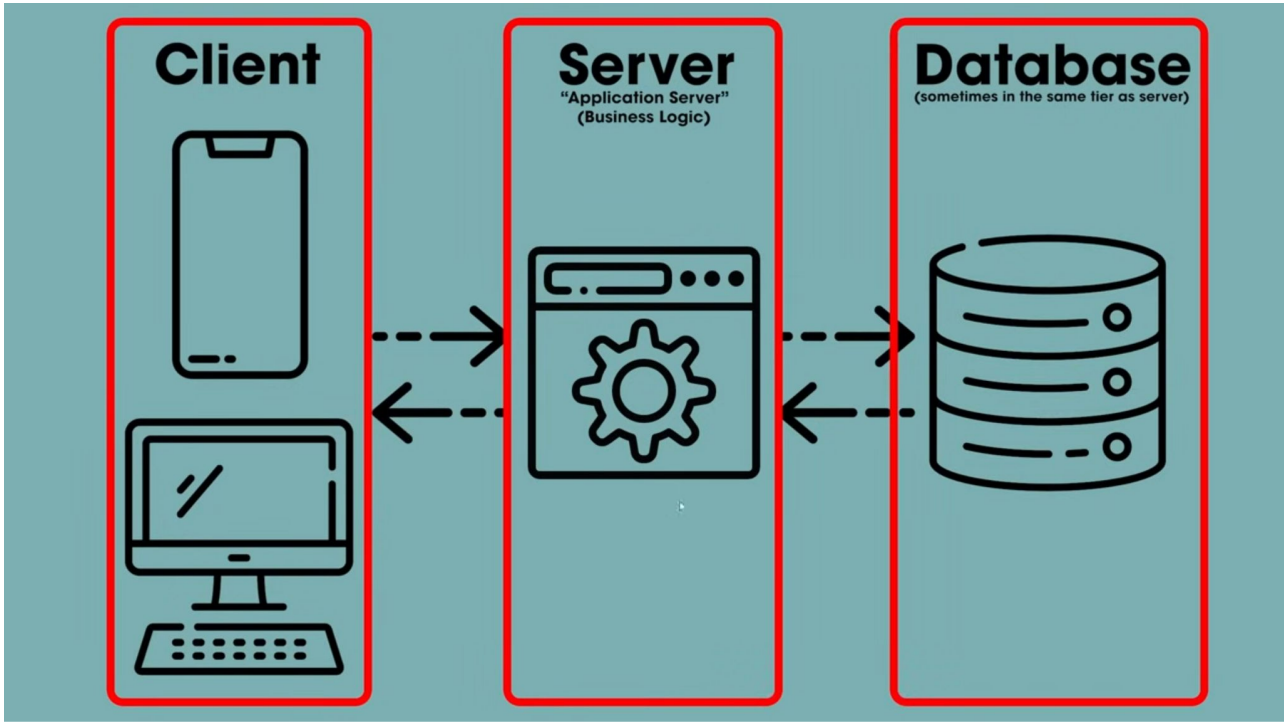


All the code lives on one machine, the database lives on a separate machine



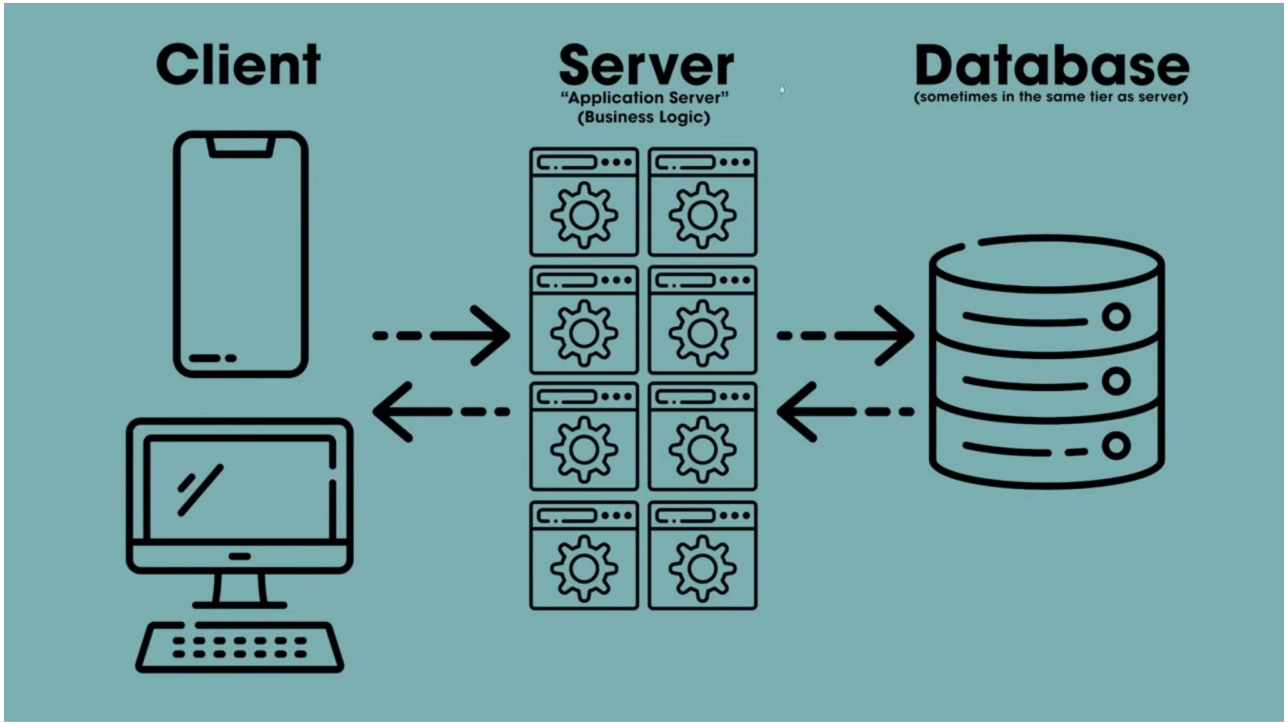
The client side lives on one machine, the server and database live on a separate machine

3-Tier Client-Server Architecture in Web Apps



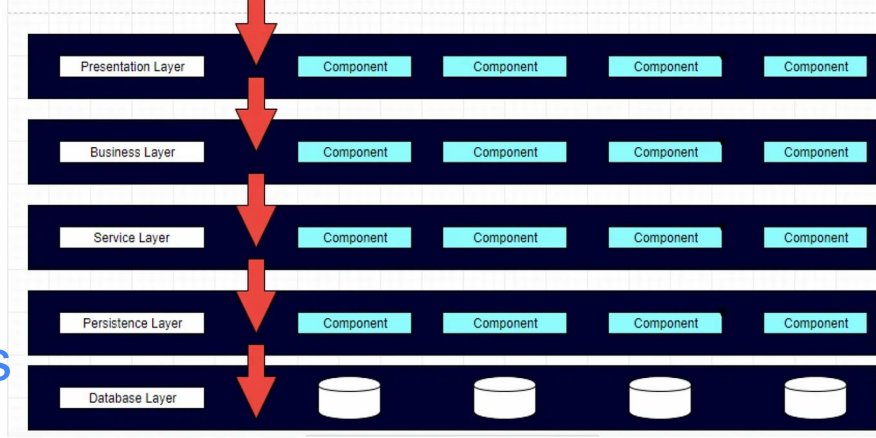
Can we have more tiers? Why?

N-Tier Client-Server Architecture in Web Apps



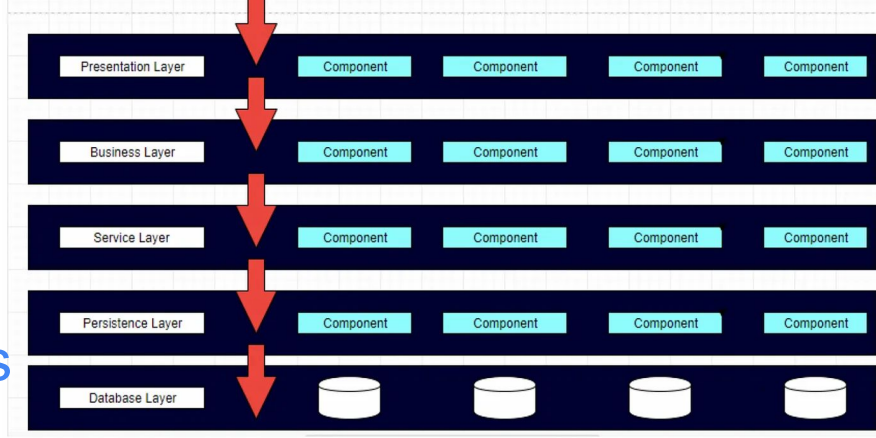
Each business logic component could live on its own machine

Layered Architecture



- Popular in E-Commerce apps
- Components are defined in layers
  - Apps may have different layers
  - Calls and data propagation flow downwards
  - Hide details within layers
  - Most common layer separation: Presentation, Business/Domain, Data
- Characteristics:
  - Easy to test components within layers
  - Easy to implement and conceptualize
  - Changes can be messy due to coupling of neighbouring layers
  - Changes in a given layer can impact the entire system

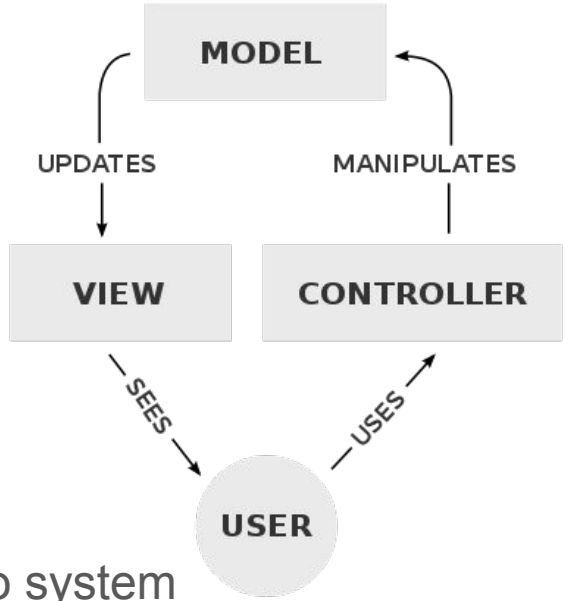
Layered Architecture



- Popular in E-Commerce apps
- Components are defined in layers
  - Apps may have different layers
  - Calls and data propagation flow downwards
  - Hide details within layers
  - Most common layer separation: Presentation, Business/Domain, Data
- Characteristics:
  - Easy to test components within layers
  - Easy to implement and conceptualize
  - Changes can be messy due to coupling of neighbouring layers
  - Changes in a given layer can impact the entire system

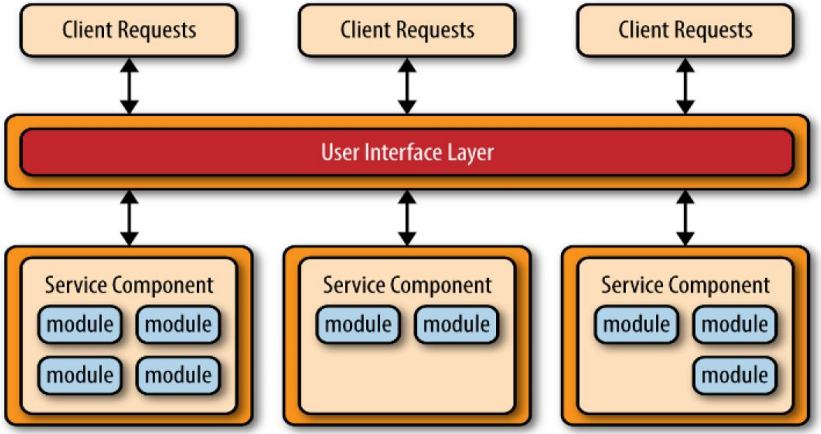
Important terms: *separation, de-coupling, modularity*

Model-View-Controller (MVC) Architecture



- Very popular layered architecture
- Used in web apps to divide responsibilities between the client and server
  - Most of the work is done on server side
  - Client sends requests through form submissions to system
  - Controller handles app logic and manipulates Model as needed
  - View retrieves data from the Model as needed
  - View sends a new page to the client
- Variations: MVP (presenter), MVA (adapter), MVVM, etc.

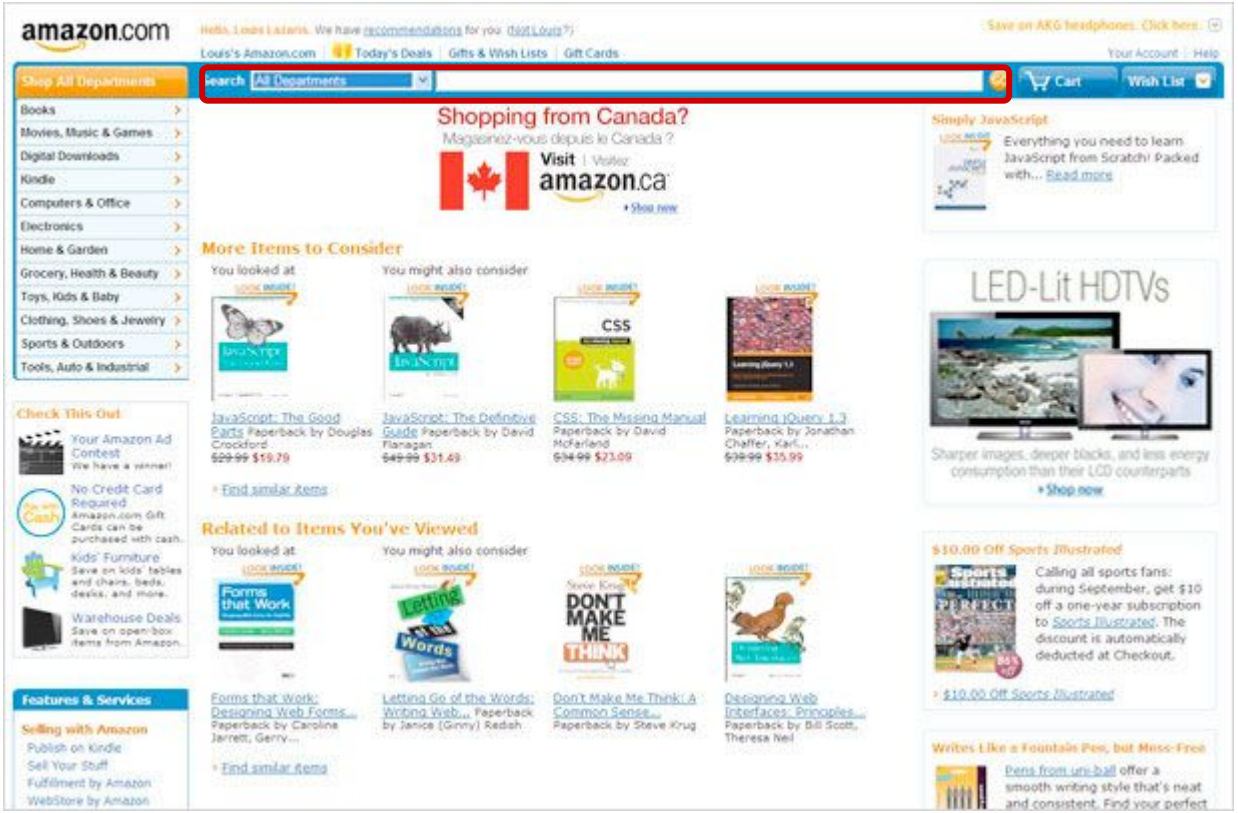
Microservices Architecture



- Involves creating multiple services that work together but can be deployed independently

- Characteristics:
  - Creates streamlined delivery pipeline
  - Its distributed nature allows component decoupling
  - Increases scalability and maintainability
  - Designing decoupled services can be tricky (for experienced architects)

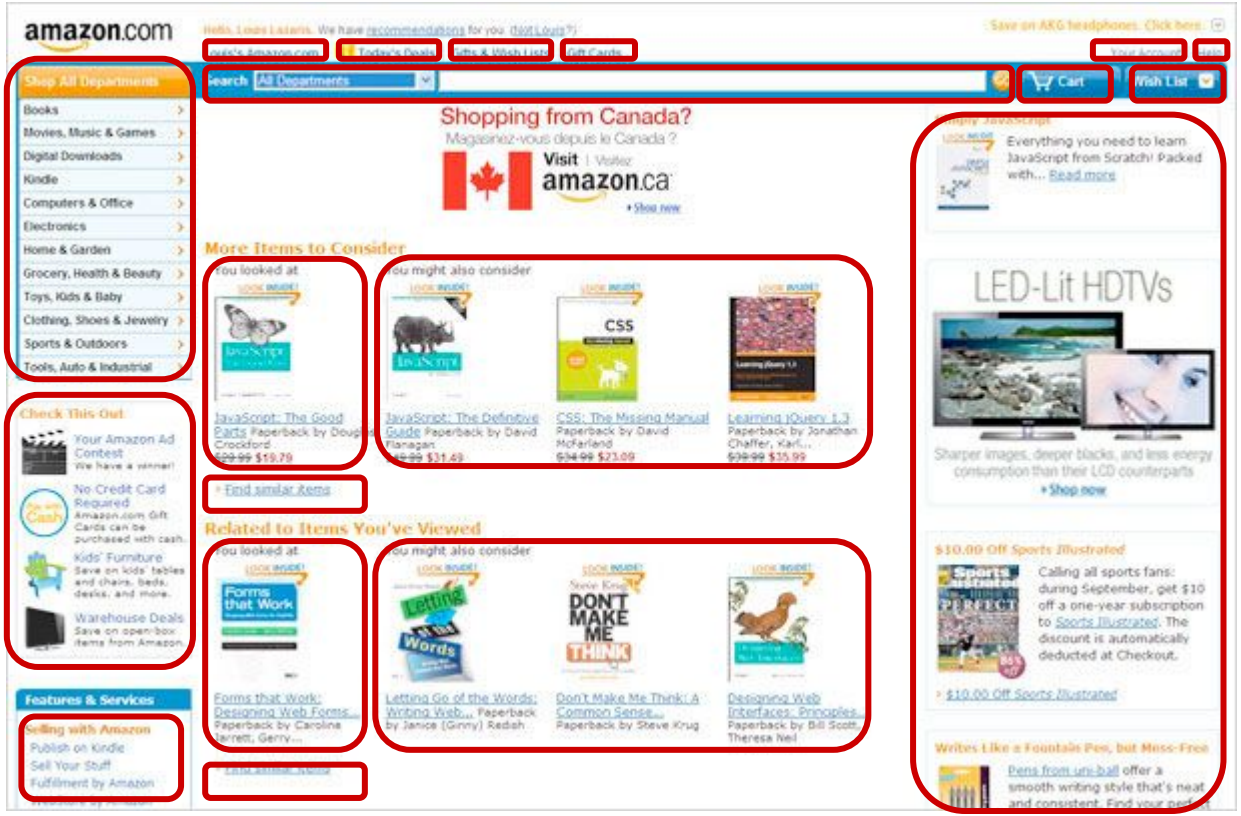
An Amazon Example



I work on search

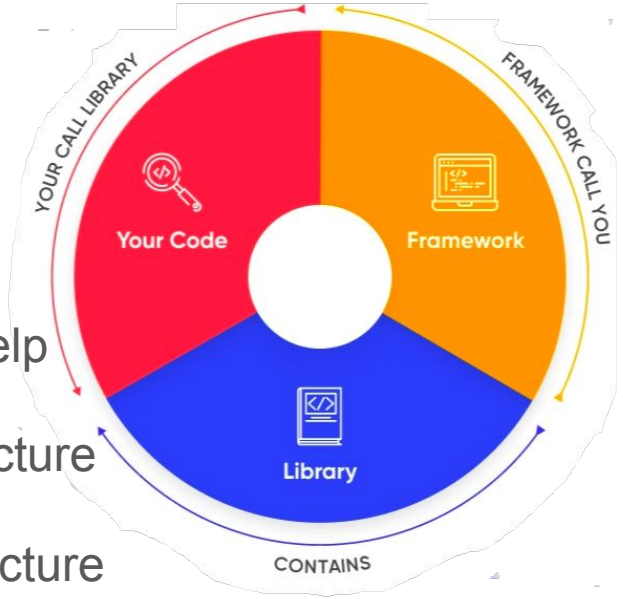
Tyson Olychuck
UBCO
Class of 2013
ex. Amazon

An Amazon Example



Tyson Olychuck
UBCO
Class of 2013
ex. Amazon

Relation to Technology Stack



- Use proven frameworks to get started
  - A framework is a set of programming tools to help build a well-structured app
  - Comes with auto-generated code for basic structure
  - Supports libraries for common functionality
  - Provides code standards and development structure for rest of app
  - Look for an active community that supports the chosen framework
- Many frameworks use a prescribed system architecture
  - E.g. Ruby on Rails is MVC
  - E.g. Flutter uses layered architecture
  - E.g. Django uses model-view-template (MVT)
  - E.g. React JS uses higher-order-component (HOC)

Data Flow Diagrams (DFD)

Follow this notation:

- Formal notation to represent data flow in a system
  - Useful for communication between technical and non-technical members
  - Identifies system scope and boundary
  - Assists in top-down system decomposition
  - Level 0: Context diagram
  - Level 1: Highest-level system processes
- Further details and examples:
  - https://online.visual-paradigm.com/knowledge/software-design/dfd-using-yourdon-and-demarco
  - https://blog.hubspot.com/marketing/data-flow-diagram

Next Steps

- Design your microservices architecture
- Contrast your architecture with other teams (same project option)
- Develop DFD levels 0 and 1

- Next week:
  - No lecture, just team support
  - Design before code
  - Align code to your design