Software Architectures + Project Teams

Lab 02 - CS 411 @ Boston University

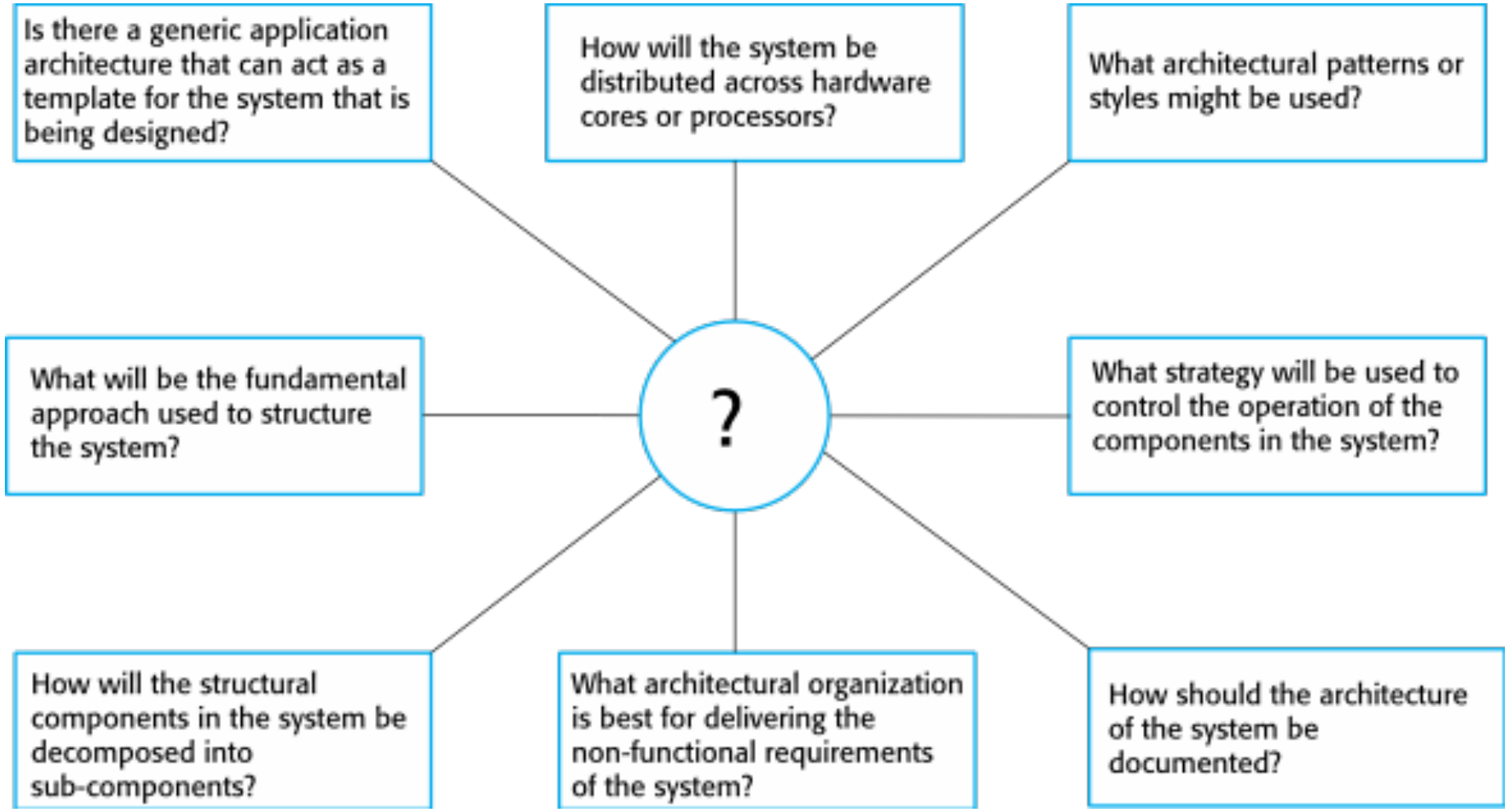
Anyone having trouble with git?

- Follow my slides from last time
- Check out Dominic's videos on Piazza (they are awesome resources)
- Come see me after class, or go to office hours for help
- Ask people in your groups

Let's talk about Software Architectures

TLDR from lecture: There are different design patterns you can use when creating software.

Architectural Design Decisions



Architecture and System Characteristics

- Performance
  - Localize critical operations and minimize communications. Use large rather than fine-grain components.
- Security
  - Use a layered architecture with critical assets in the inner layers.
- Safety
  - Localize safety-critical features in a small number of sub-systems.
- Availability
  - Include redundant components and mechanisms for fault tolerance.
- Maintainability
  - Use fine-grain, replaceable components.

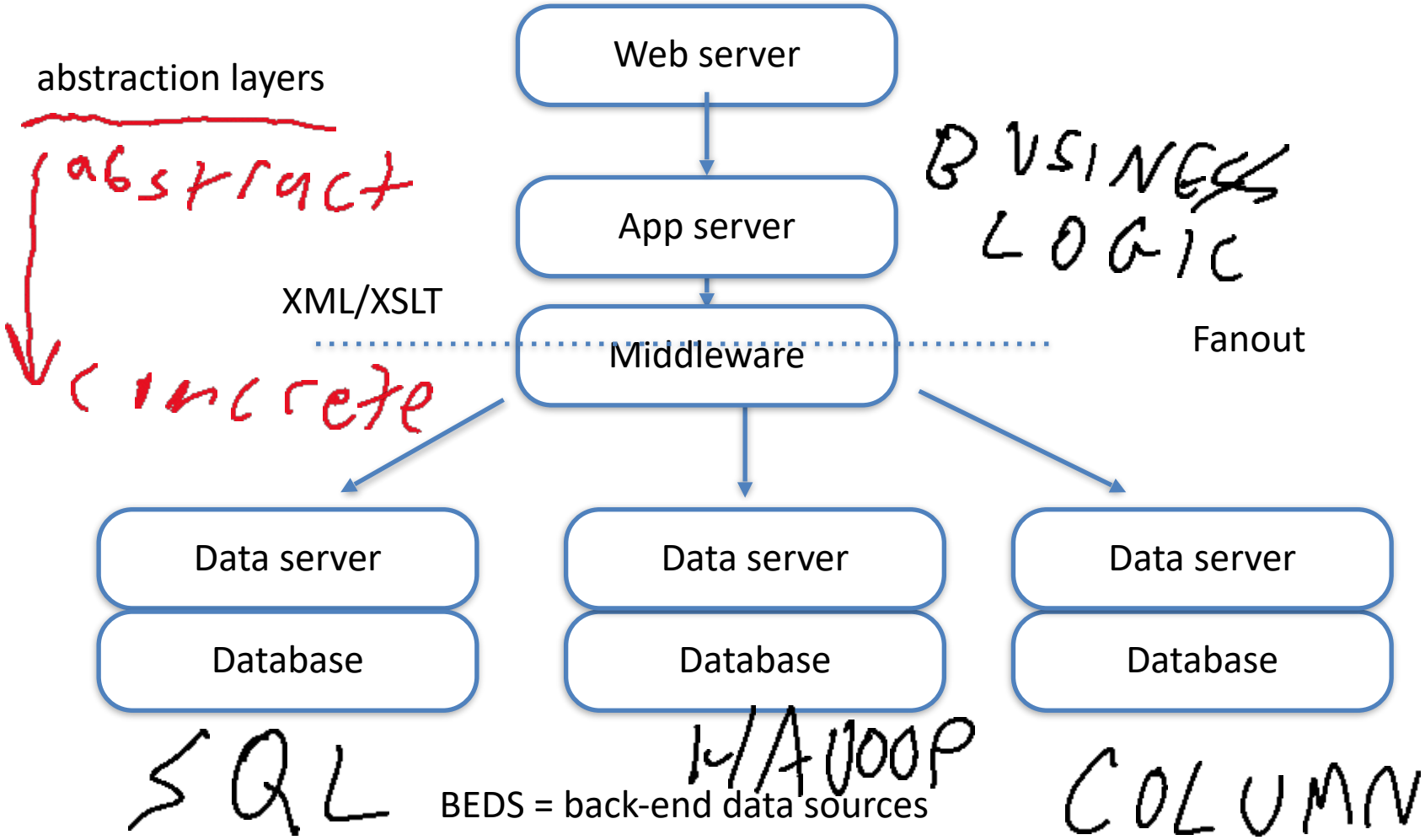
Client-Server Architecture

- Each component of a client-server system has the role of either client or server (request / response)

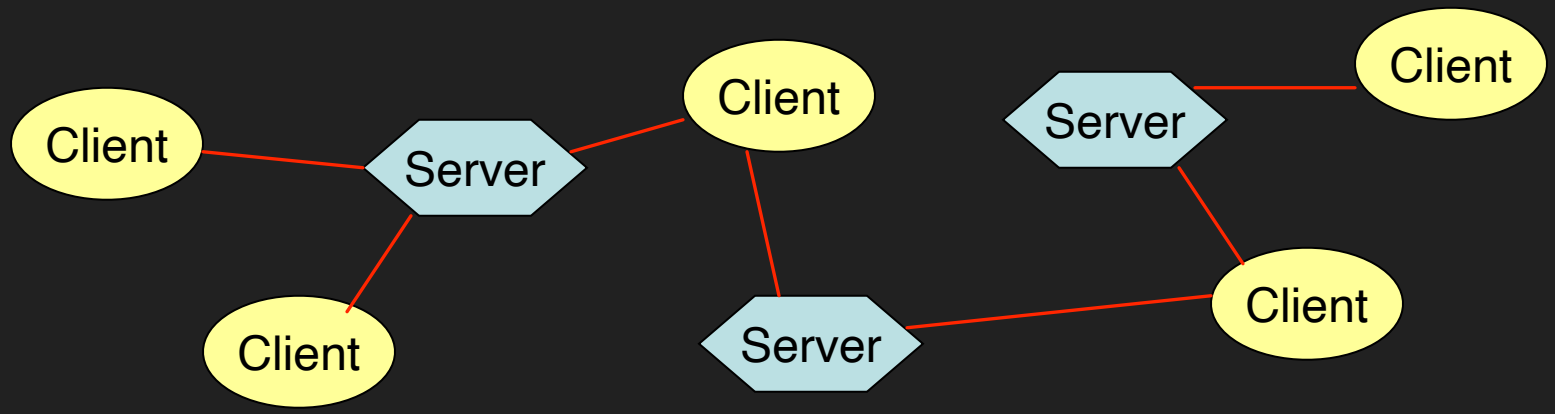
  - Client: a component that makes requests
    clients are active initiators of transactions

  - Server: a component that satisfies requests
    servers are passive and react to client requests

Call and Return Architecture - more detailed



The web is a client-server system:

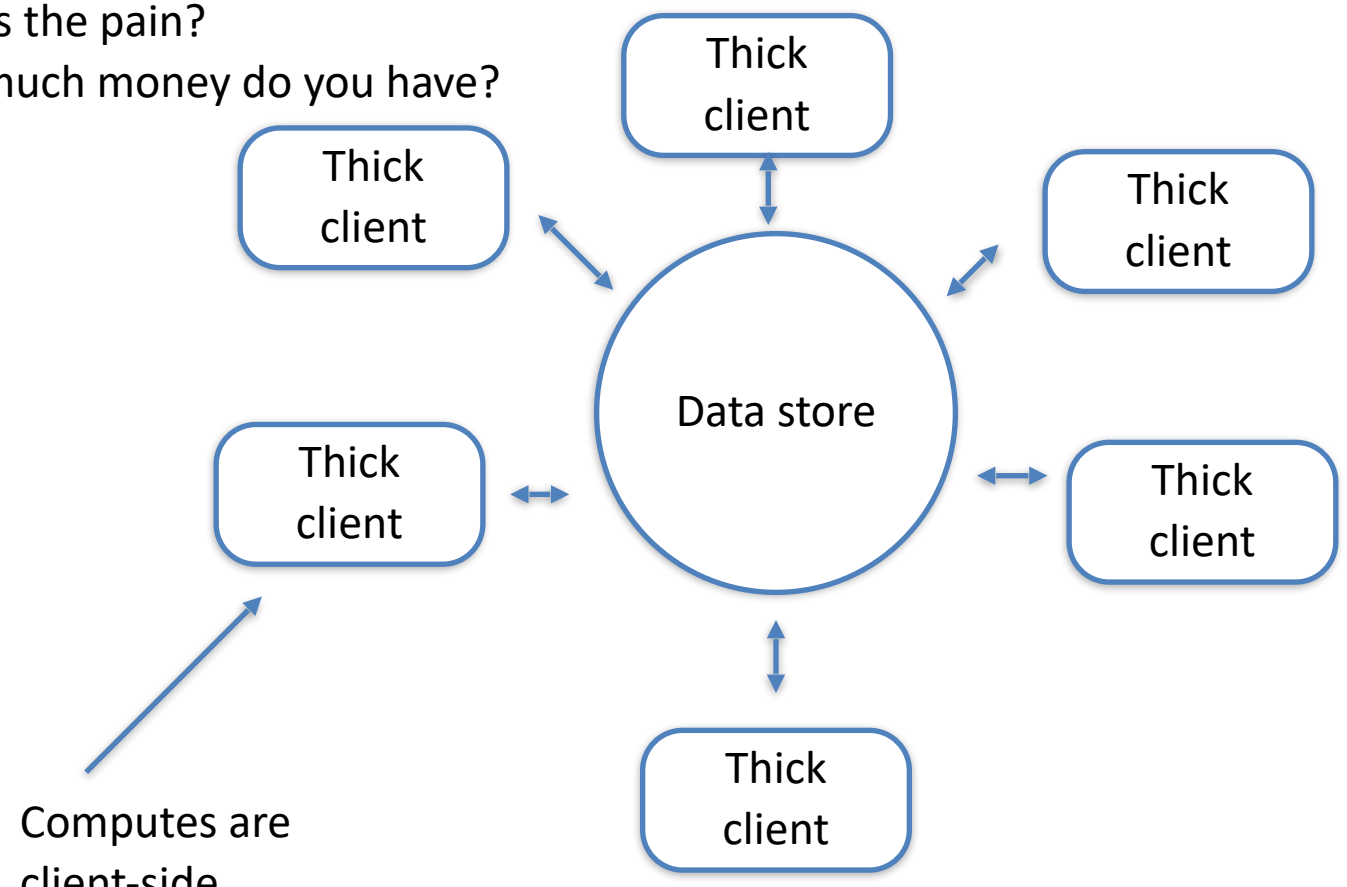


Thick vs Thin Client

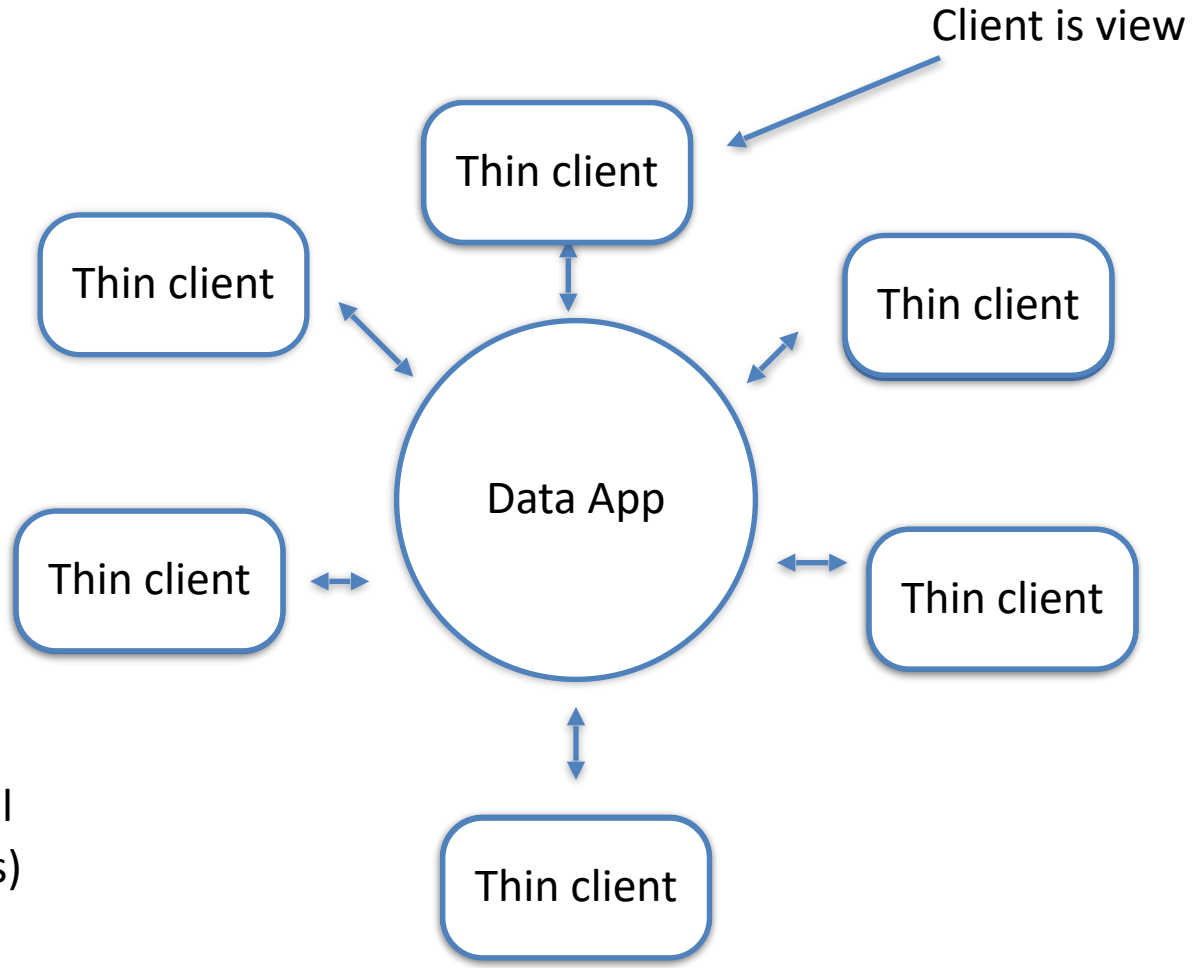
- Thick client: Work is done on both client and server
- Thin client: Work is done on almost exclusively on the server

Data-Centered Architecture

What's the pain?
How much money do you have?



Data-Centered Architectures

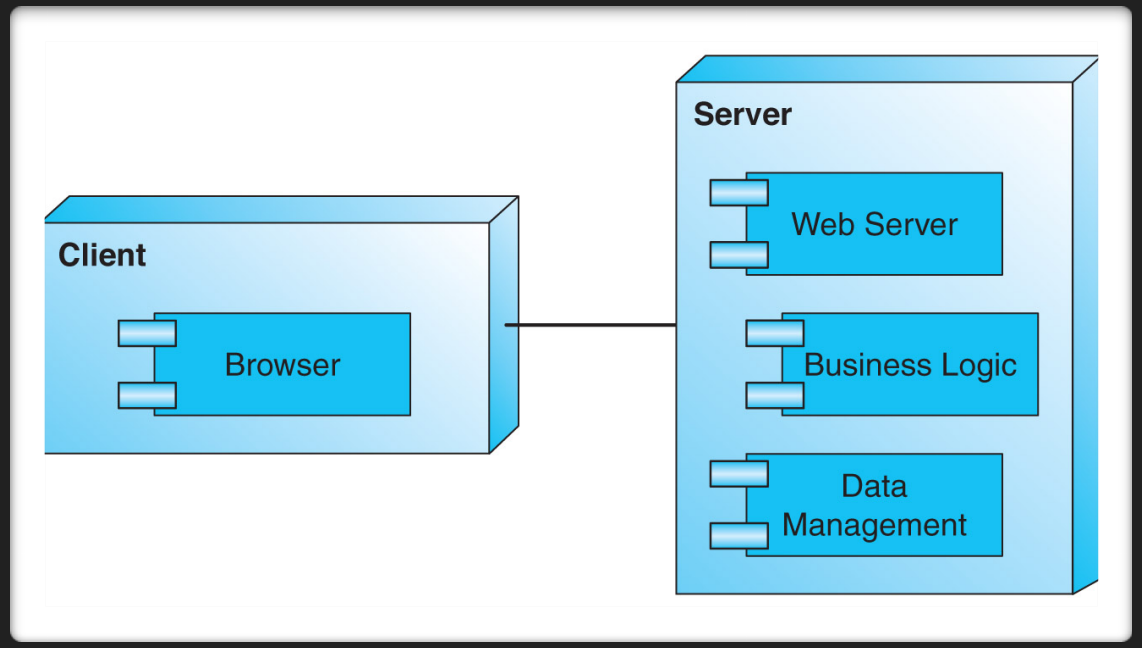


dumb terminal
(green screens)

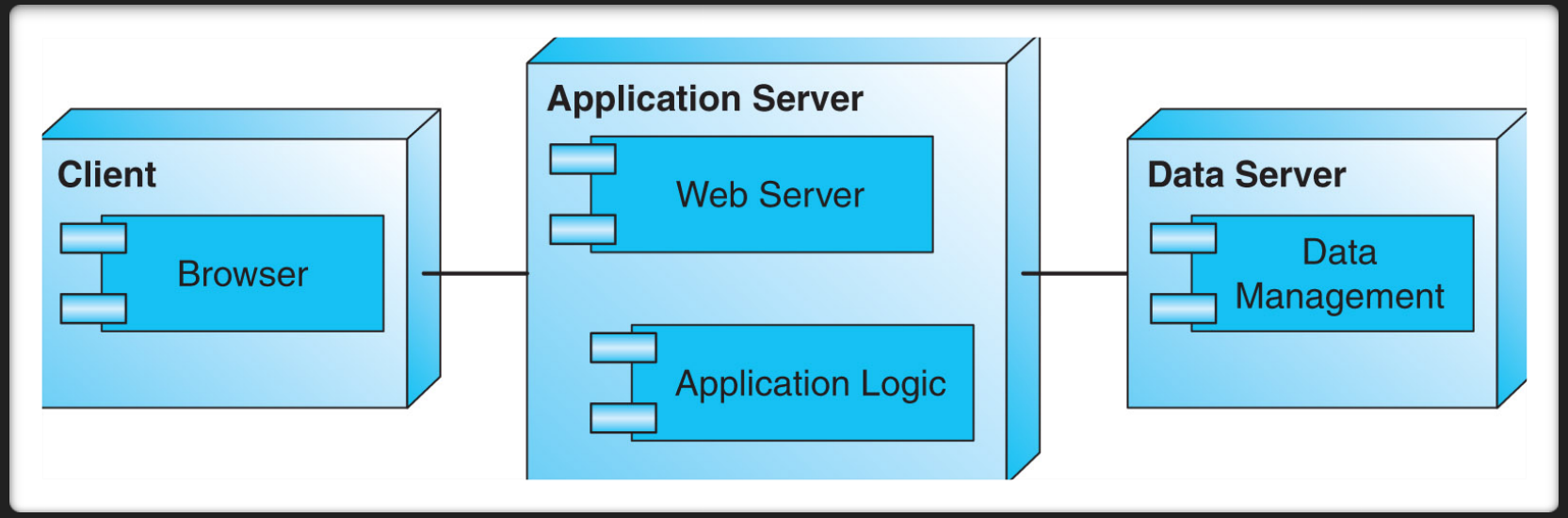
Tiered Web Architectures

- Web applications are usually implemented with 2-tier, 3-tier, or multitier (N-tier) architectures
- Each tier is a platform (client or server) with a unique responsibility

2-Tier C-S Architecture



3-Tier C-S Architecture



2-Tier Characteristics

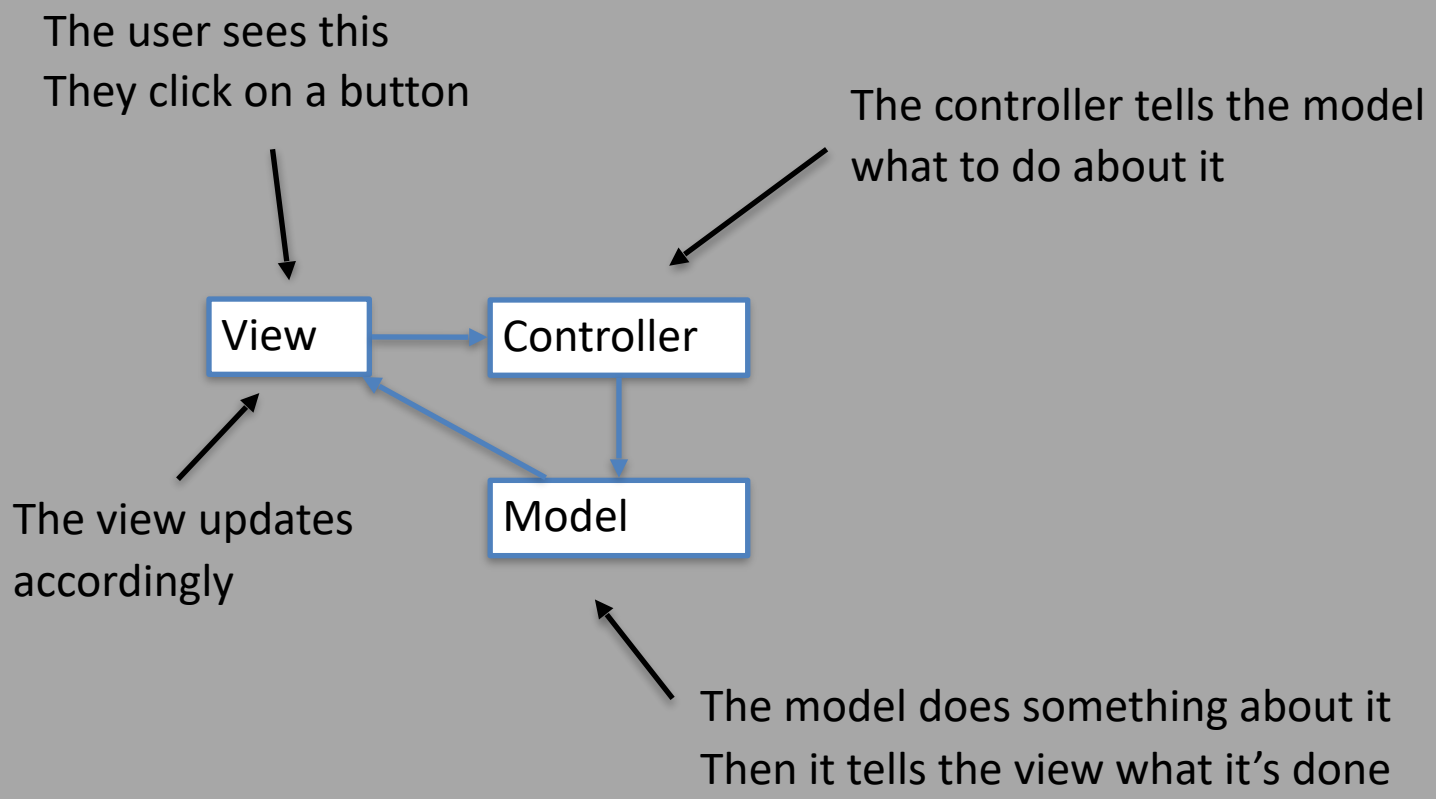
- Advantage:
  - Inexpensive (single platform)
- Disadvantages
  - Interdependency (coupling) of components
  - No redundancy
  - Limited scalability
- Typical application
  - 10-100 users
  - Small company or organization, e.g., law office, medical practice, local non-profit

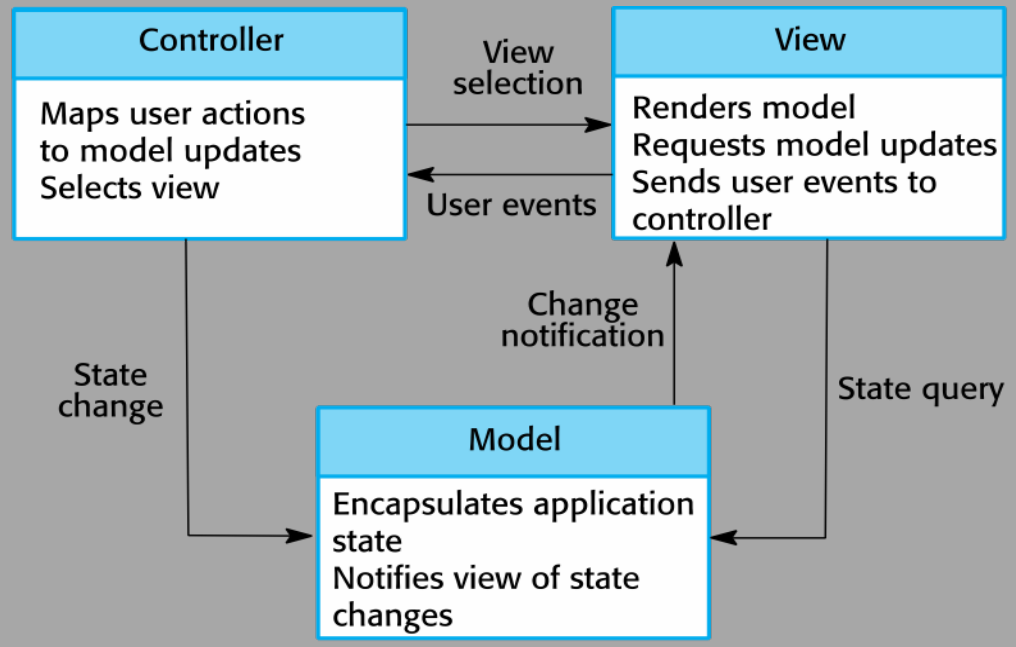
3-Tier Characteristics

- Advantages
  - Improved performance, from specialized hardware
  - Decreased coupling of software components
  - Improved scalability
- Disadvantages
  - No redundancy
- Typical Application
  - 100-1000 users
  - Small business or regional organization, e.g., specialty retailer, small college

Model-View-Controller (MVC) pattern

- Before discussing MVC we need to understand the *state* of an object or group of objects
- When we talk about object we are interested in their attributes (their internal variables) and their behaviors (the methods on the object)
- State is the current instantaneous value of all of an object's attributes… what values do the variables hold
- The MVC pattern gives us a way to isolate state from other parts of the architecture
- It's all about **encapsulation of concerns**


The user sees this
They click on a button
The controller tells the model
what to do about it
View
Controller
Model
The view updates
accordingly
The model does something about it
Then it tells the view what it's done



State = the values of variables at a given moment

Think about how you're going to balance your teams!

Having separate roles for:

- Front-end HTML/CSS/Bootstrap
- Database (mongoDB, mySQL)
- Back-end (Node, PHP (gross imho), Flask)
- Views (Angular, etc.)
- Project manager

All can work on test design and execution

Usually teams in industry have four to six members, so this balance is easier to achieve. However, given that you're in teams of 3 mostly, it's alright to share roles among each other and develop fluidly.

Find your teams! Talk for the rest of the session

Link's on piazza or here if you want