

# Bird's-eye View of SaaS Architecture (1)

September 13, 2022

Byung-Gon Chun

(Credits to some slides: Armando Fox, UCB)

# Software as a Service

# Software Targets

- Traditional SW: binary code installed and runs wholly on client device, which users must upgrade repeatedly
  - Must work with many versions of hardware, many versions of OS
  - New versions must go through extensive release cycle to ensure compatibility
- An alternative where develop SW that only needs to work on one HW & OS platform?
  - ➔ Quicker release cycle, no user upgrades?

# Software as a Service: SaaS

- SaaS delivers SW & data as service over Internet via thin program (e.g., browser) running on client device
  - Search, social networking, video
- Now also SaaS version of traditional SW
  - E.g., Microsoft Office 365, TurboTax Online
- SaaS is revolutionary, the future of virtually all software

# Exemplary Companies



- The finance, HR, and planning system for a changing world



- Automating and connecting the entire agreement process



- SaaS CRM (Customer Relationship Management)



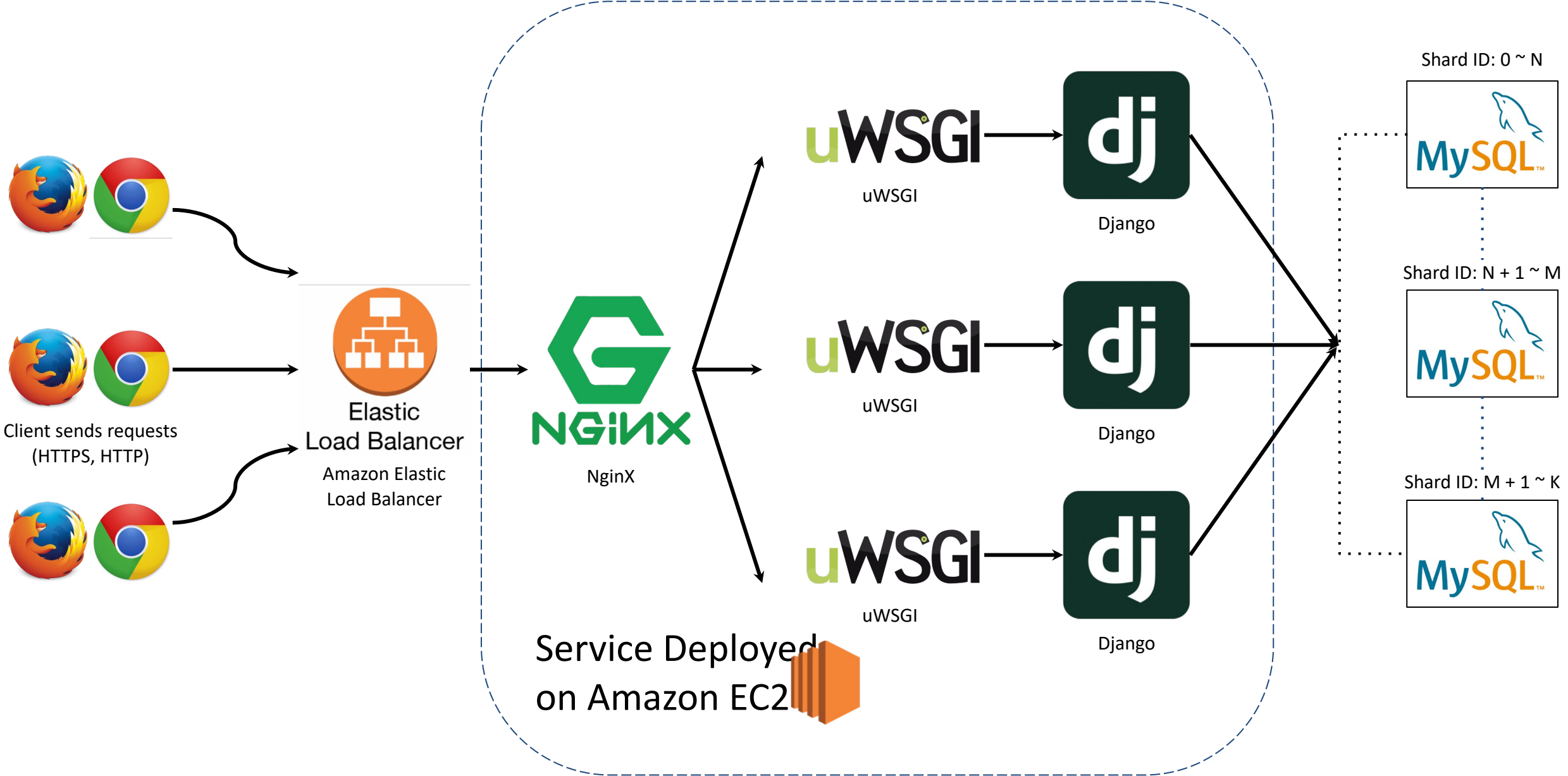
- E-commerce focused on handmade or vintage items and craft supplies

# 6 Reasons for SaaS

1. No install worries about HW capability, OS
2. No worries about data loss (data is remote)
3. Easy for groups to interact with same data
4. If data is large or changed frequently, simpler to keep 1 copy at central site
5. 1 copy of SW, single HW/OS environment  
=> no compatibility hassles for developers  
=> beta test new features on 1% of users
6. 1 copy => simplifies upgrades for developers  
*and* no user upgrade requests

# The Web as a Client-Server System

# Deployment with multiple Backend instances





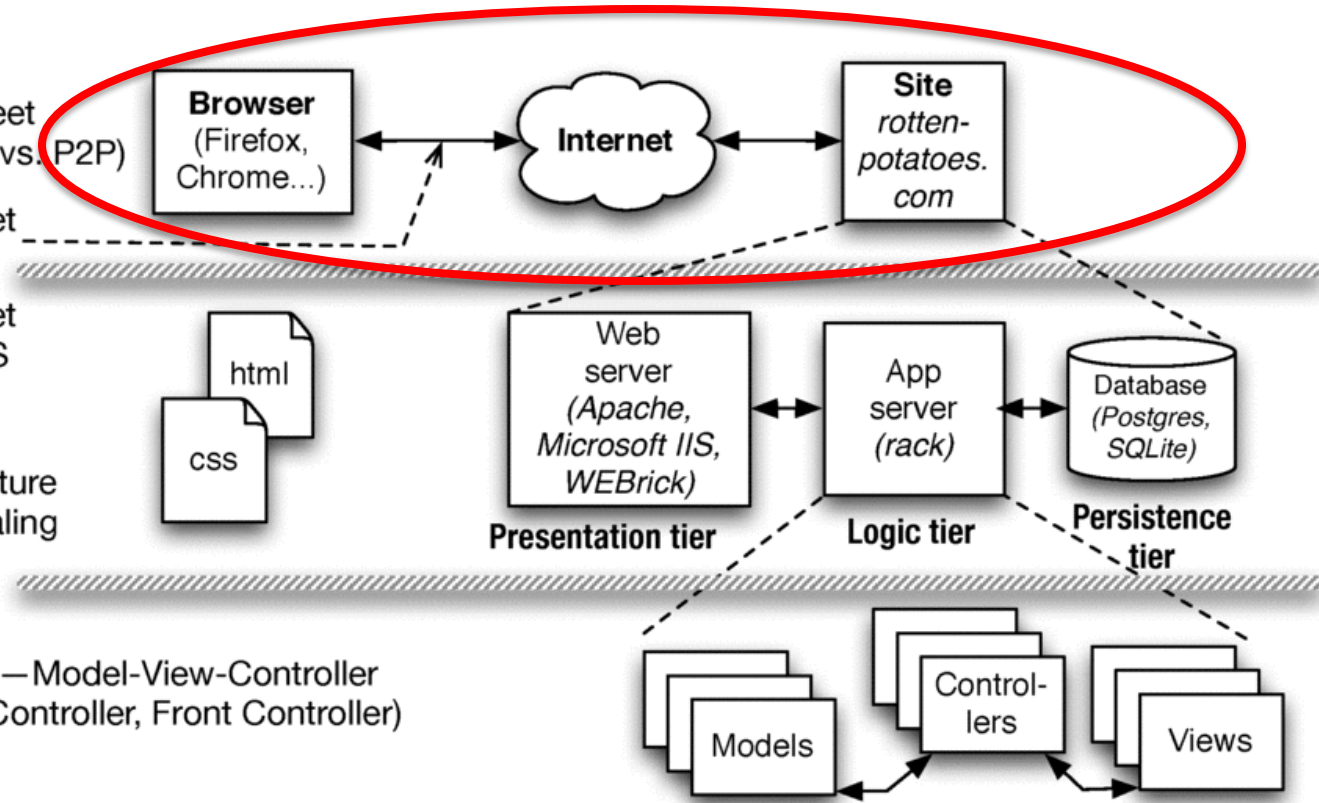
§2.1 100,000 feet  
• Client-server (vs. P2P)

§2.2 50,000 feet  
• HTTP & URIs

§2.3 10,000 feet  
• XHTML & CSS

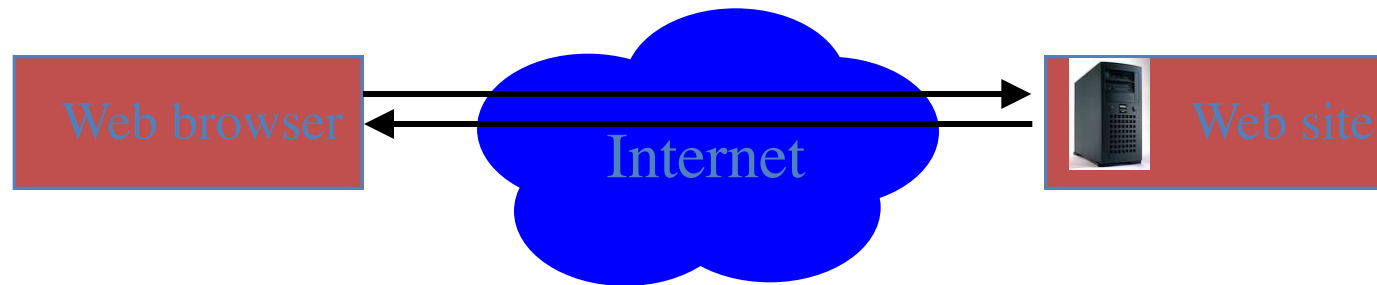
§2.4 5,000 feet  
• 3-tier architecture  
• Horizontal scaling

§2.5 1,000 feet—Model-View-Controller  
(vs. Page Controller, Front Controller)

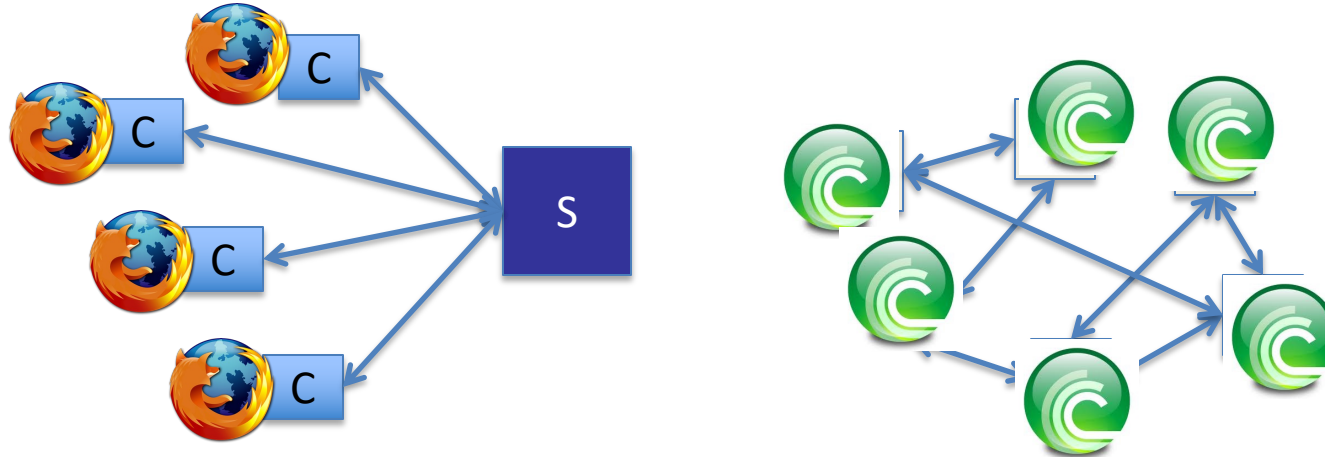


# Web at 100,000 feet

- The web is a *client/server* architecture
- It is fundamentally *request/reply oriented*



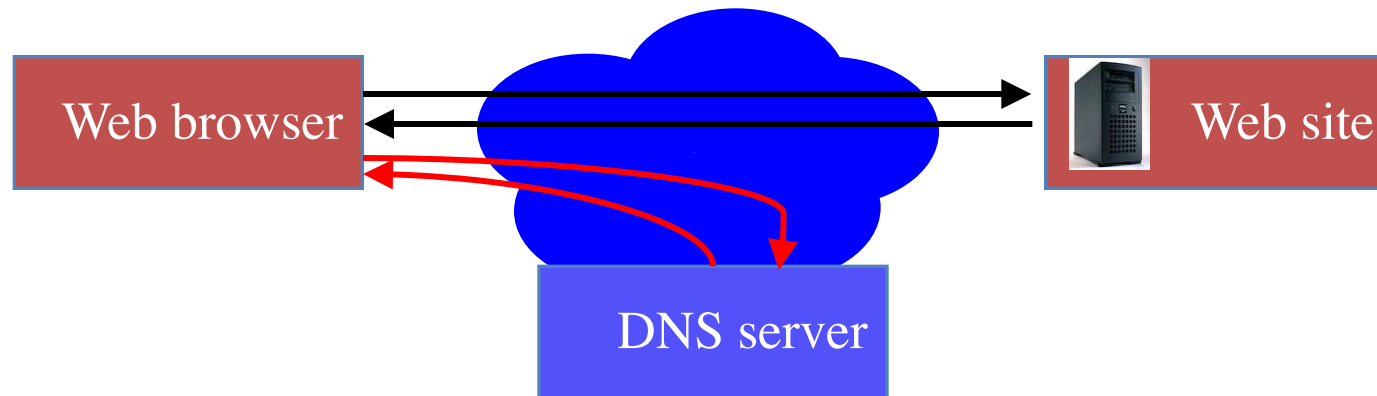
# Client-Server vs. Peer-to-Peer



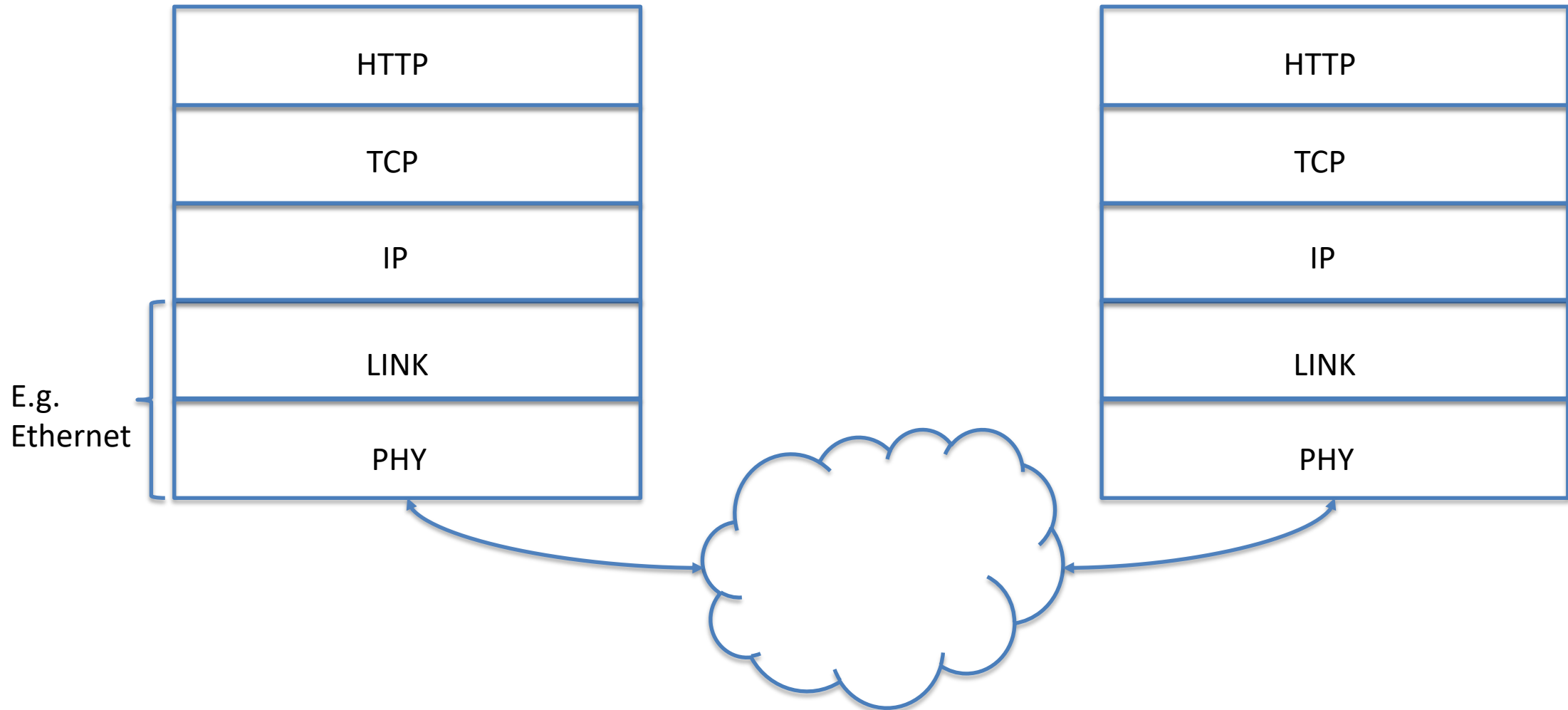
- High-level architecture of the overall system
  - Soon we'll talk about architecture "inside" boxes
- Client & server each *specialized* for their tasks
  - Client: ask questions on behalf of users
  - Server: wait for & respond to questions, serve many clients
- Design Patterns capture common structural solutions to recurring problems
  - Client-Server is an *architectural pattern*

# Web at 100,000 feet

- The web is a *client/server* architecture (distributed system)
  - Domain Name System (DNS) is another kind of server that maps *names* to *IP addresses* (e.g., [www.snu.ac.kr](http://www.snu.ac.kr) => 147.46.10.129)
- It is fundamentally *request/reply oriented*: *HTTP protocol*





# Protocol Stack for Web



# Nuts and bolts: TCP/IP protocols

- IP (Internet Protocol)v4 *address* identifies a physical network interface with four *octets*, e.g. **128.32.244.172**
  - Special address **127.0.0.1** is “this computer”, named **localhost**, even if not connected to the Internet!
- TCP/IP (Transmission Control Protocol/Internet Protocol)
- IP: no-guarantee, best-effort service that delivers *packets* from one IP address to another
- TCP: make IP reliable by detecting “dropped” packets, data arriving out of order, transmission errors, slow networks, etc., and respond appropriately
  - TCP *ports* allow multiple TCP apps on same computer

**GET /snu/**      **GET /snu/**  
**HTTP/0.9 200 OK**      **HTTP/0.9 200 OK**



Vint Cerf & Bob Kahn: 2004 Turing Award

# Now that we're talking, what do we say?

## Hypertext Transfer Protocol (HTTP)

- An *ASCII-based request/reply protocol* for transferring information on the Web
- *HTTP request* includes:
  - *request method* (**GET**, **POST**, etc.)
  - Uniform Resource Identifier (URI)
  - HTTP protocol version understood by the client
  - *headers*—extra info regarding transfer request
- *HTTP response* from server
  - Protocol version & Status code =>
  - Response headers
  - Response body

### **HTTP status codes:**

2xx — *all is well*

3xx — *resource moved*

4xx — *access problem*

5xx — *server error*

# Cookies

- Observation: *HTTP is stateless*
- Early Web 1.0 problem: how to guide a user “through” a flow of pages?
  - use IP address to identify returning user?
    - ✗ public computers, users sharing single IP
  - embed per-user junk into URI query string?
    - ✗ breaks caching
- Quickly superseded by *cookies*



```
(base) Byung-Gons-MacBook-Pro:iloveswpp bgchun$ curl -D - http://www.google.com -o /dev/null
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total      Spent    Left     Speed

  0     0    0     0    0     0      0      0  --:--:--  --:--:--  --:--:--    0HTTP/1.1 200 OK
Date: Sat, 12 Sep 2020 01:41:21 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2020-09-12-01; expires=Mon, 12-Oct-2020 01:41:21 GMT; path=/; domain=.google.c
om; Secure
Set-Cookie: NID=204=wVsYHVLPqcv8lQC_GY0EGHv3M34nWp9XGFLWanNhidg6aA3BAXMQMRiN1CQKTP_hqElN_nmKj34IJ
D0g6WI9yolEbZsgvwExSaxSUFX0TRBB8toz8axvBYdxR2Lbsra1X5xfMuY_IsJLaeZhwHDPaXiueJVLi1vRyocYVc7Fc0o; e
xpires=Sun, 14-Mar-2021 01:41:21 GMT; path=/; domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

100 13143    0 13143    0     0 82143    0  --:--:--  --:--:--  --:--:-- 82660
```

# Real-time Comm.: Server Push vs. Client Pull

- Real-time components: chat, multi-player games, push notification, event feed, etc.
- Client pull: periodic polling (don't use this technique!)
  - Polling creates a new connection for every pull

# Real-time Comm.: Server Push vs. Client Pull

- WebSocket/HTML5: The server pushes updated content to the client (without an explicit request from the client). **Full-duplex connection between server and client.**
  - Supported by all modern browsers (there is a JS API to use them natively in the browser)

```
// Create WebSocket connection.  
const socket = new WebSocket('ws://localhost:8080');  
// Listen for messages  
socket.addEventListener('message', function (event) {  
    console.log('Message from server ', event.data);  
});
```

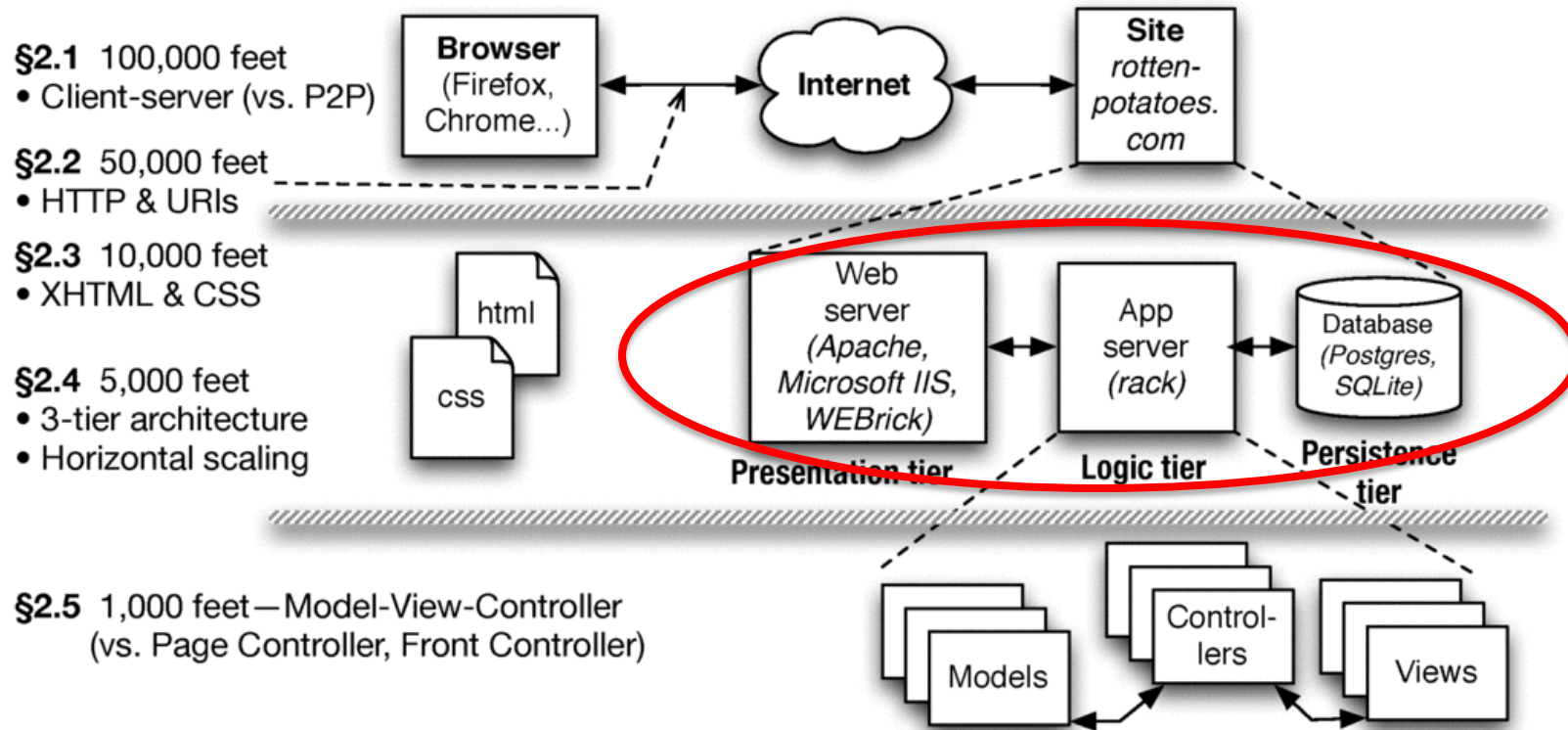
# Real-time Comm.: Server-side Events

- Server-side events: push notifications, event feeds, etc.
  - We get only one connection and keep an event stream going through it.
  - Server-to-client only: unidirectional

```
// subscribe for messages
var source = new EventSource('URL');

// handle messages
source.onmessage = function(event) {
    // Do something with the data:
    event.data;
};
```

# Backend: 3-tier Shared-Nothing Architecture & Scaling

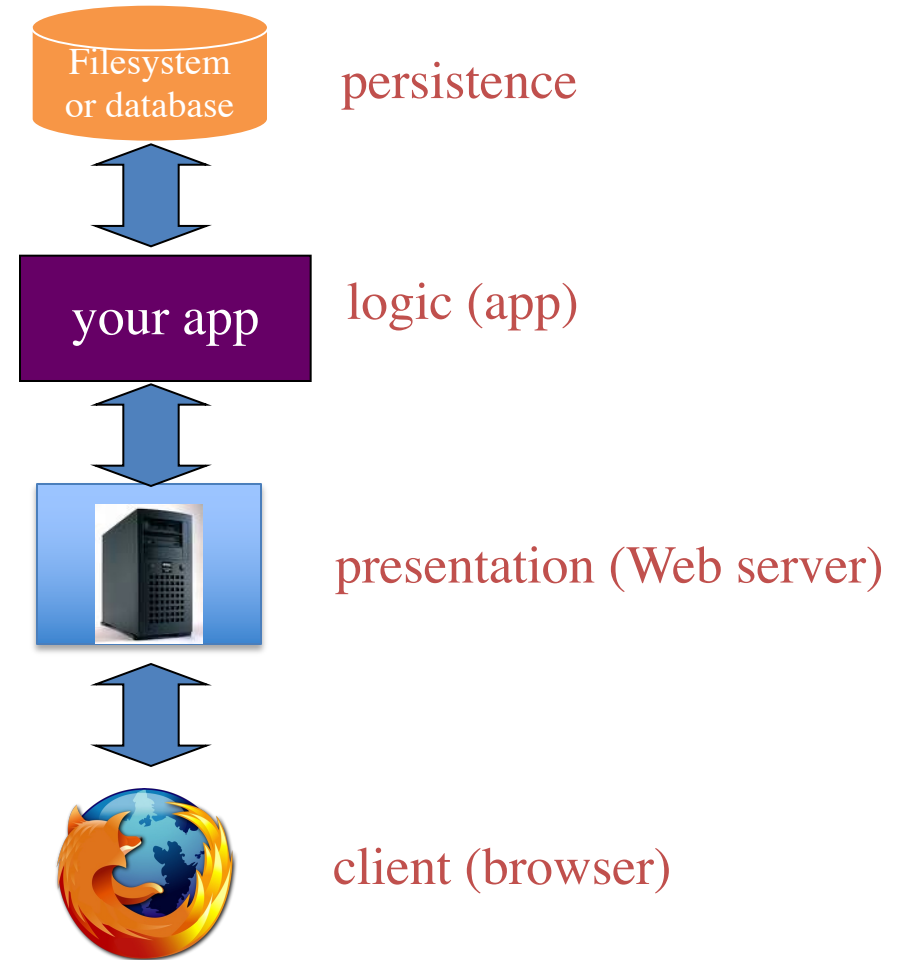


# Dynamic content generation

- In the Elder Days, most web pages were (collections of) plain old files
- But most interesting Web 1.0/e-commerce sites *run a program* to generate each “page”
- Originally: templates with embedded code “snippets”
- Eventually, **code** became “tail that wagged the dog” and moved out of the Web server

# Sites that are really programs (SaaS)

- How do you:
  - “map” URI to correct program & function?
  - pass arguments?
  - invoke program on server?
  - handle persistent storage?
  - handle cookies?
  - handle errors?
  - package output back to user?
- *Frameworks* support these common tasks
  - *Python Django*
  - *Node.js*

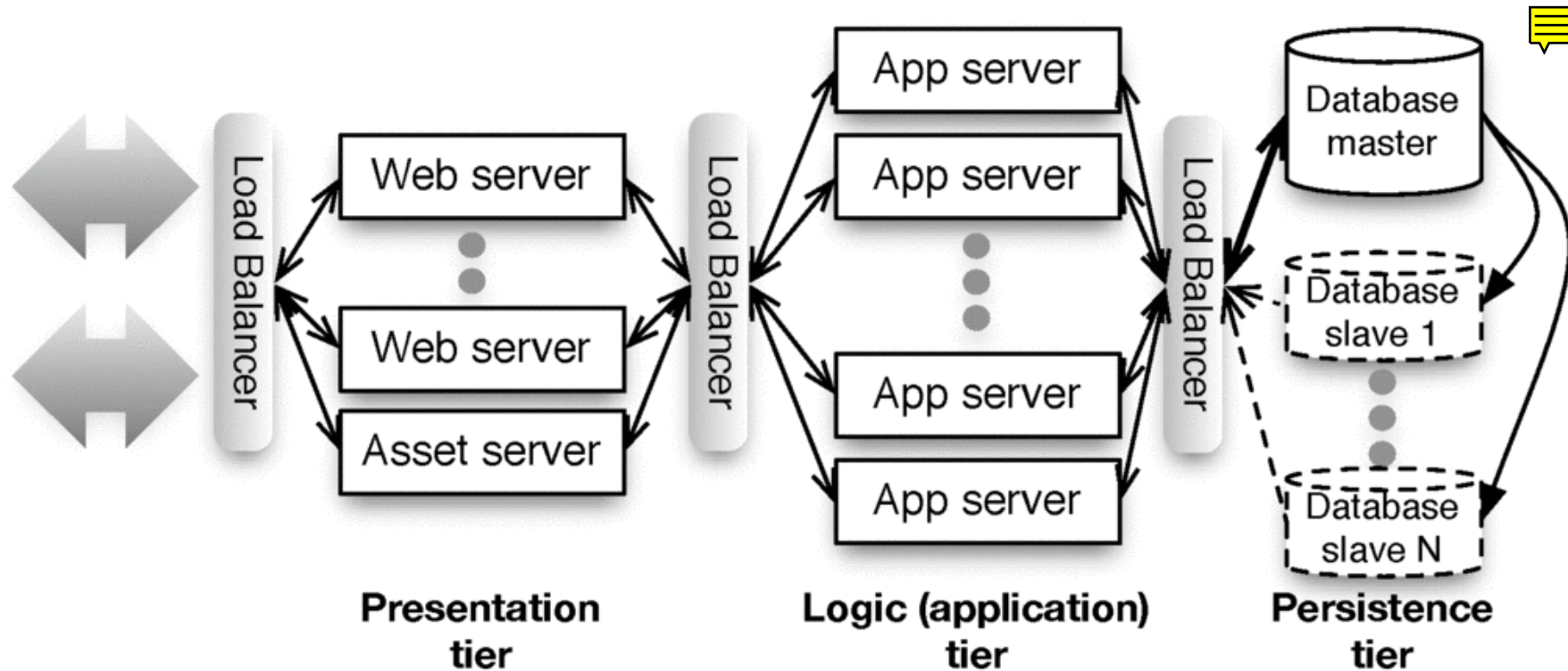




# Web Application Framework

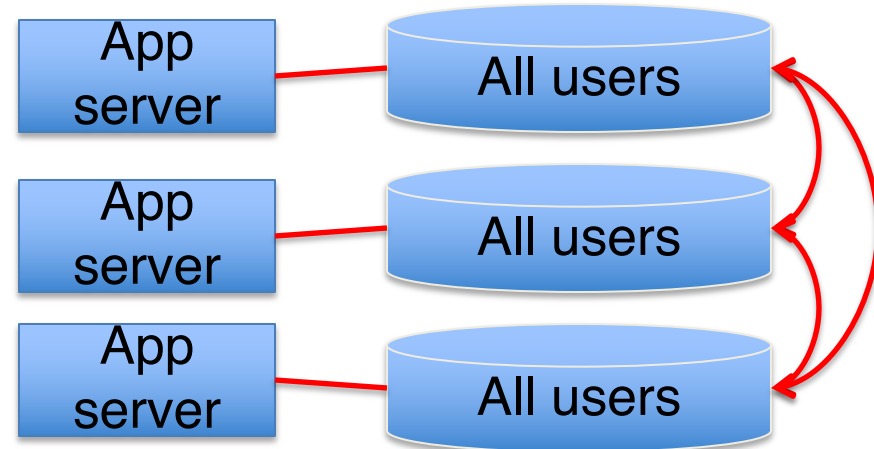
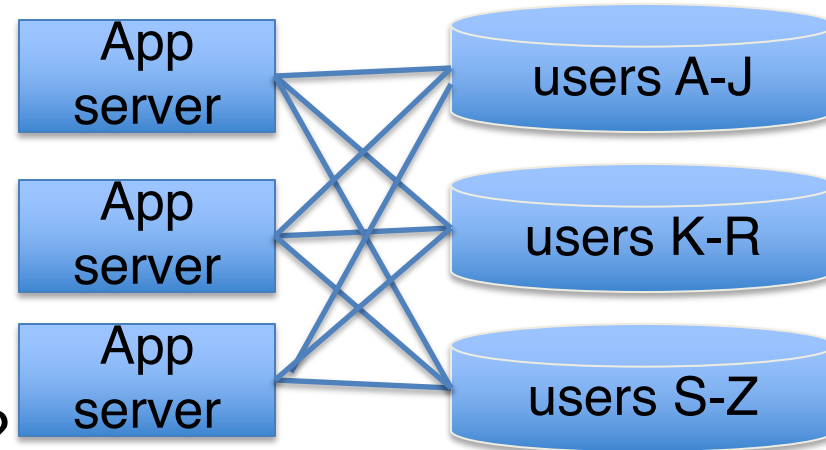
- URL routing
- HTML, XML, JSON, and other output format templating
- Database manipulation
- Security against Cross-site request forgery (CSRF) and other attacks
- Session storage and retrieval

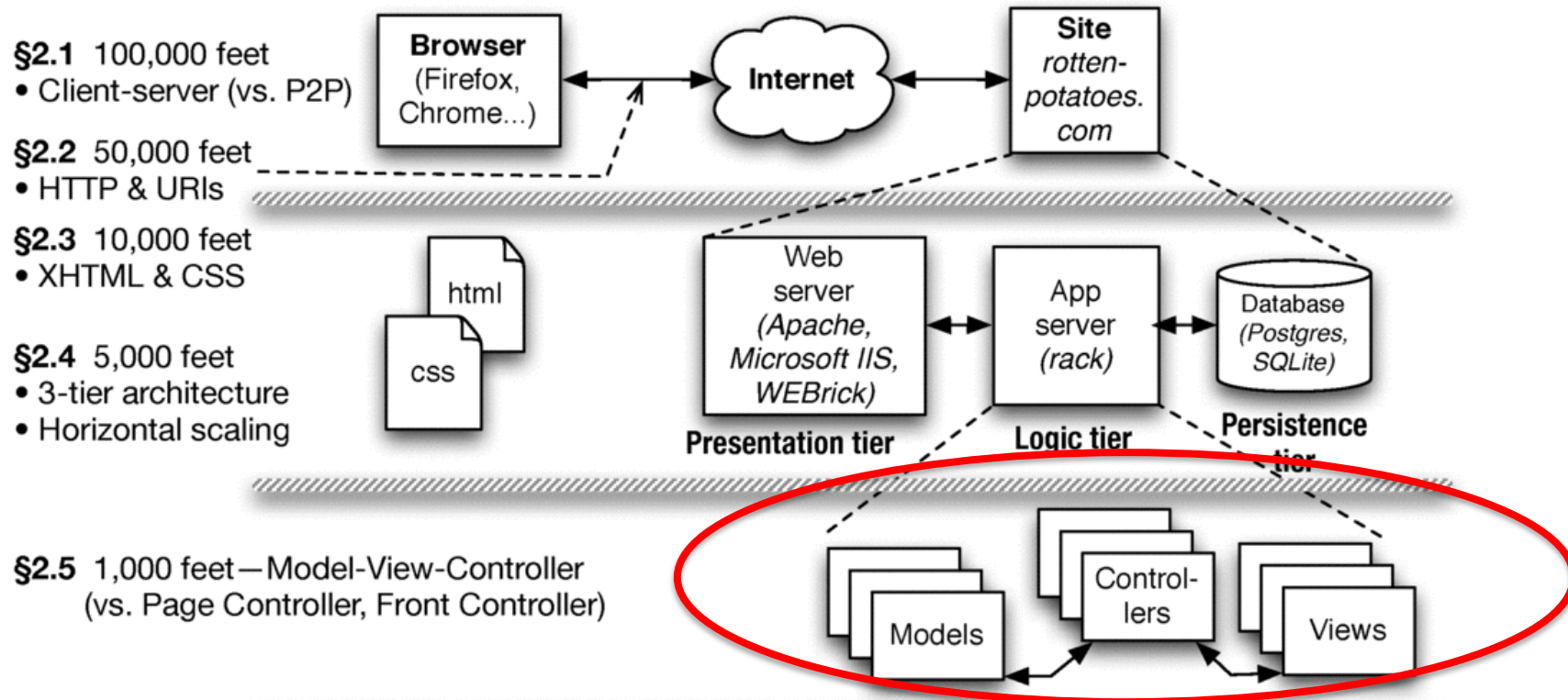
# “Shared nothing”



# Sharding vs. Replication

- Partition data across independent “shards”?
  - + Scales great
  - Bad when operations touch >1 table
  - Example use: user profile
- Replicate all data everywhere?
  - + Multi-table queries fast
  - Hard to scale: writes must propagate to all copies => temporary *inconsistency* in data values
  - Example: Facebook wall posts/“likes”





# Model-View-Controller

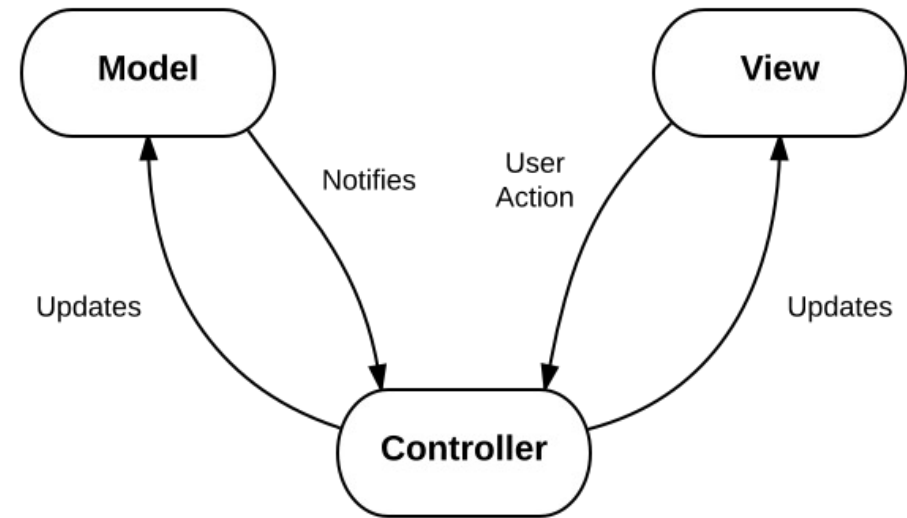
- The MVC paradigm is a way of breaking an application into three parts: the model, the view, and the controller.
- MVC was originally developed to map the traditional input, processing, output roles into the GUI realm:
- Input -> Processing -> Output
- Controller -> Model -> View

# Little Bit of MVC History

- The formal separation of these three tasks is an important notion that is particularly suited to MVC Smalltalk-80 where the basic behavior can be embodied in abstract objects: *View*, *Controller*, *Model*, and *Object*
- MVC was discovered by Trygve Reenskaug in 1979

# Model-View-Controller

- The pattern isolates business logic from input and presentation, permitting independent development, testing and maintenance of each.
- An MVC application may be a collection of MVCs, each responsible for a different UI element.



# Model, View, Controller

- Model
  - Domain-specific representation of the data on which the application operates
  - When a model changes its state, it notifies its associated views so they can refresh through the controller
  - Many applications use a persistent storage mechanism (such as a database) to store data. MVC does not specifically mention the data access layer because it is understood to be underneath or encapsulated by the model



# Model, View, Controller

- View
  - Renders the model into a form suitable for interaction, typically a user interface element.
  - Multiple views can exist for a single model for different purposes.
- Controller
  - Receives input and initiates a response by making calls on model objects.
  - Allows data to flow between the view and the model
  - The controller mediates between the view and model

# MVC Benefits

- Clarity of design
  - easier to implement and maintain
- Modularity
  - changes to one doesn't affect other modules
  - can develop in parallel once you have the interfaces
- Multiple views
  - spreadsheets, powerpoint, file browsers, games, Eclipse, UML reverse engineering, ....

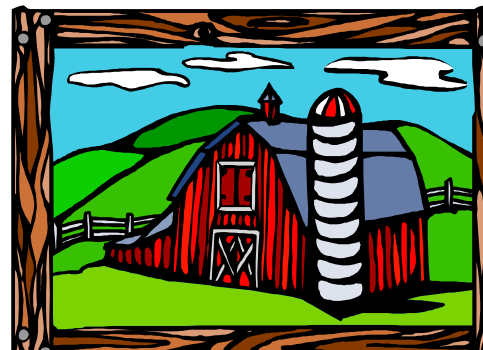
# Service Oriented Architecture (SOA)

# Software Architecture Question

- Can you design software so that you can **recombine independent modules to offer many different apps** without a lot of programming?

# Service Oriented Architecture

- SOA: SW architecture where **all components** are designed to be **services**
- Apps composed of interoperable services
  - Easy to tailor new version for subset of users
  - Also easier to recover from mistake in design
- Contrast to “SW silo” without internal APIs



# CEO: Amazon shall use SOA!

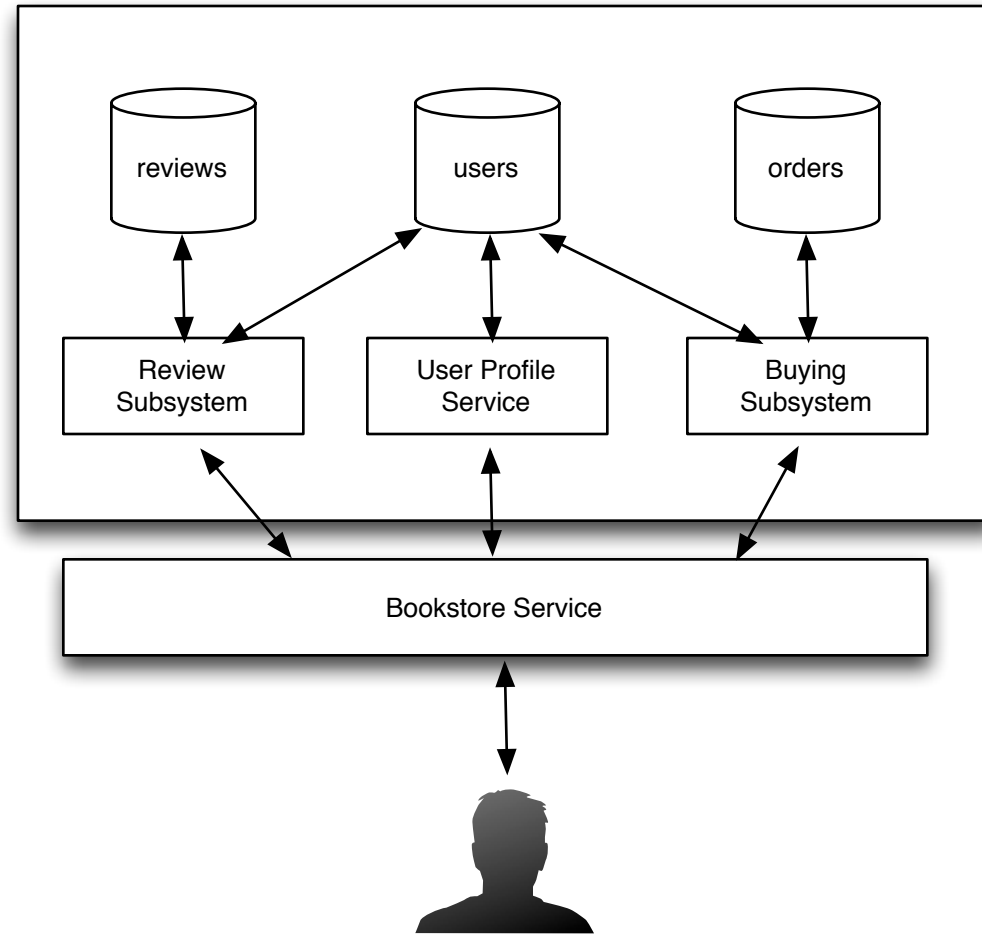
1. “All teams will henceforth expose their data and functionality through service interfaces.”
2. “Teams must communicate with each other through these interfaces.”
3. “There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.”

# CEO: Amazon shall use SOA!

4. “It doesn't matter what [API protocol] technology you use.”
5. “Service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.”
6. “Anyone who doesn't do this will be fired.”
7. “Thank you; have a nice day!”

# Bookstore: Silo

- Internal subsystems can share data directly
  - Review access user profile
- All subsystems inside single API (“Bookstore”)

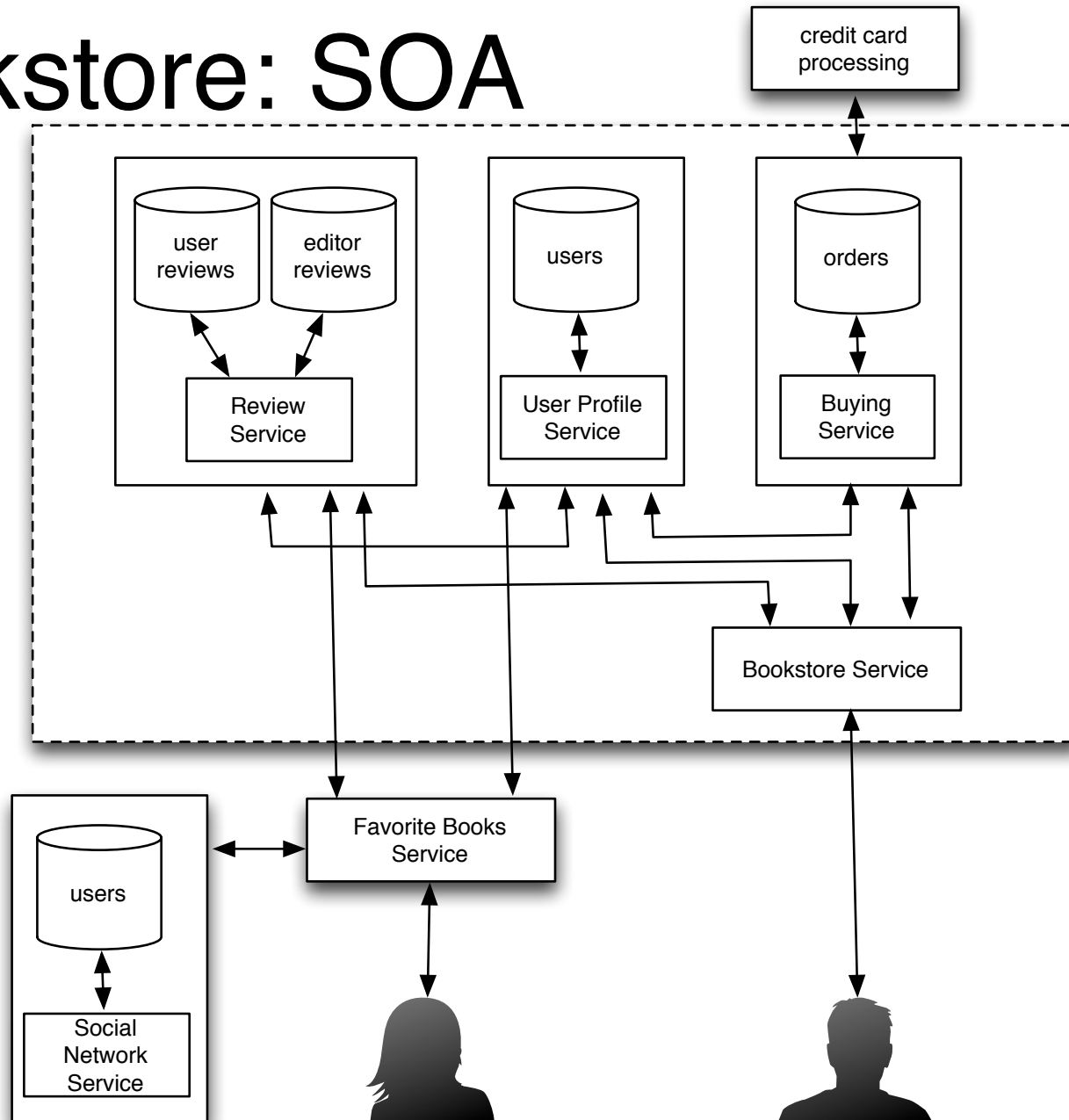


(Figure 1.2, *Engineering Software as a Service* by Armando Fox and David Patterson, 2<sup>nd</sup> Beta edition, 2013.)



# Bookstore: SOA

- Subsystems independent, as if in separate datacenters
  - Review Service access User Service API
- Can recombine to make new service (“Favorite Books”)

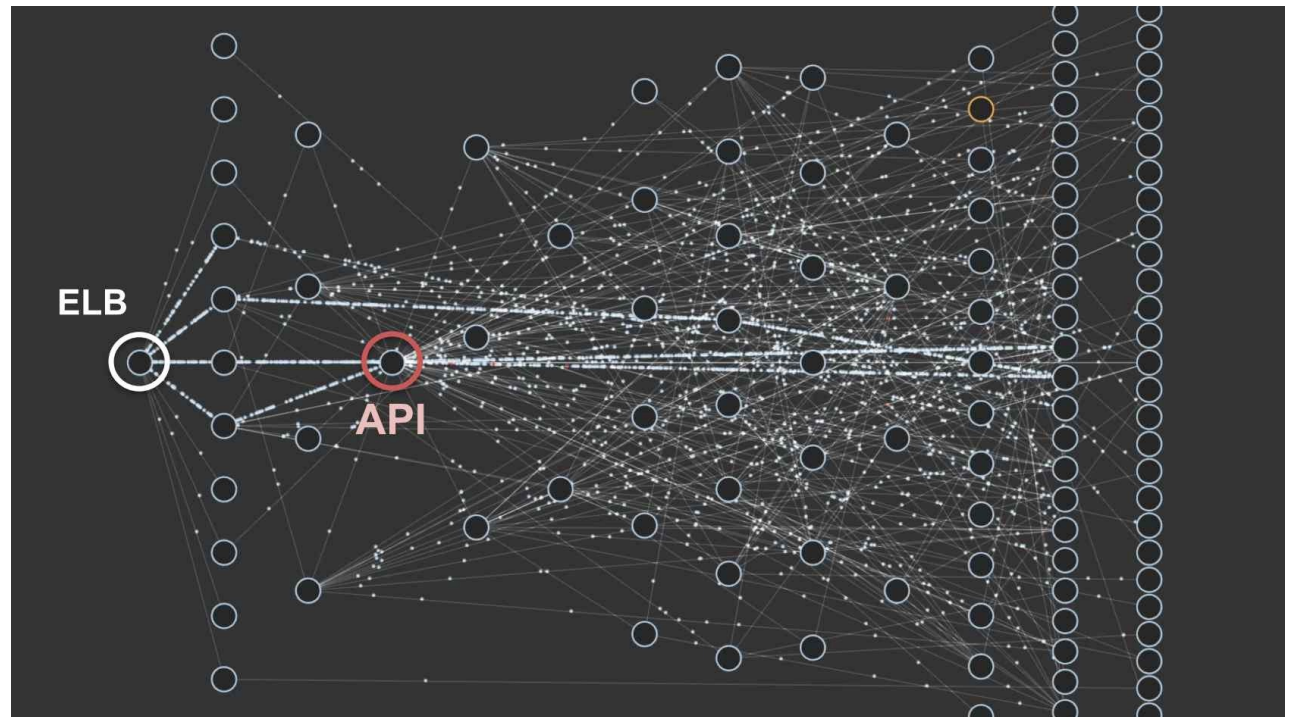
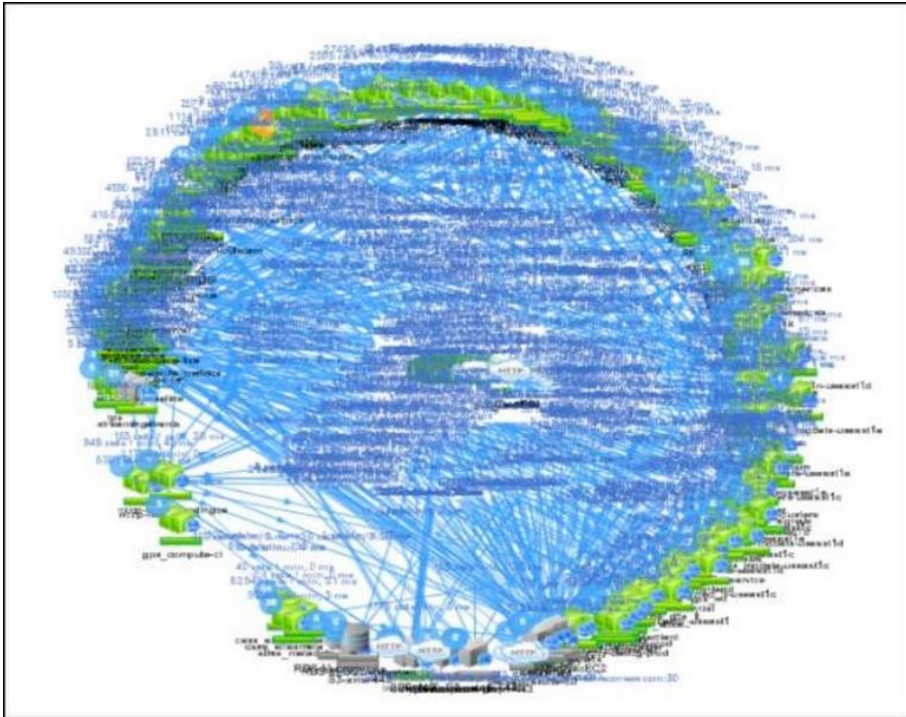


(Figure 1.3, *Engineering Software as a Service* by Armando Fox and David Patterson, 2<sup>nd</sup> Beta edition, 2013.)

# Microservices

- A variant of the SOA architectural style that structures an application as a collection of loosely coupled services
- Services are fine-grained and the protocols are lightweight

## Netflix



# Netflix Microservices

- Netflix uses over 1,000 microservices now.
  - It would spend over [\\$1 billion](#) on “streaming services and cloud computing costs” through 2023 - That would average Netflix’s AWS cloud services costs at over \$27.78 million per month.
- Why? Prevent service delivery outages by using multiple, smaller services that ran independently instead of a single, vulnerable stack.

# Netflix Microservices

- Create a main menu list of movies
- Determine your subscription status to provide content relevant to that subscription tier
- Use your watch history to recommend videos you may like
- Bill your credit card when it is time to renew your Netflix plan
- Keep tabs on the best-performing Content Delivery Appliance (CDA) near you in case the one you are on becomes overcrowded or fails
- Automatically migrate you to a CDA with the strongest internet connection
- Store a digital copy of original cinema-quality files on AWS servers
- Ensure another set of AWS servers convert the original copy into video qualities, formats, sizes, and audio that can play seamlessly on all kinds of devices, from mobile devices to smart TVs and gaming equipment
- Determine what device you are watching Netflix on and provide the relevant video format for it to boost your viewing experience
- Add copyright markers to all files (Digital Rights Management)