How does the Internet Works

- The internet is the global network of computers that communicate using standard protocols like TCP/IP

  - **IANA allocates large blocks of IP addresses to Regional Internet Registries (RIRs)**
    The Internet Assigned Numbers Authority (IANA) oversees global IP address management and delegates IP blocks and Autonomous System Numbers (ASNs) to RIRs such as ARIN, RIPE NCC, and APNIC, which each serve different geographic regions.

  - **RIRs distribute IP blocks to National Internet Registries (NIRs) or directly to Local Internet Registries (LIRs)**
    In countries with an NIR (like India or Japan), the RIR delegates IP resources to them. Where NIRs do not exist, RIRs allocate directly to LIRs such as internet service providers, universities, or large companies.

  - **NIRs coordinate IP allocations within their country**
    A National Internet Registry receives IP blocks from the RIR and is responsible for assigning IP addresses to LIRs and ISPs in that country. It maintains the national IP address database and ensures fair distribution and policy compliance.

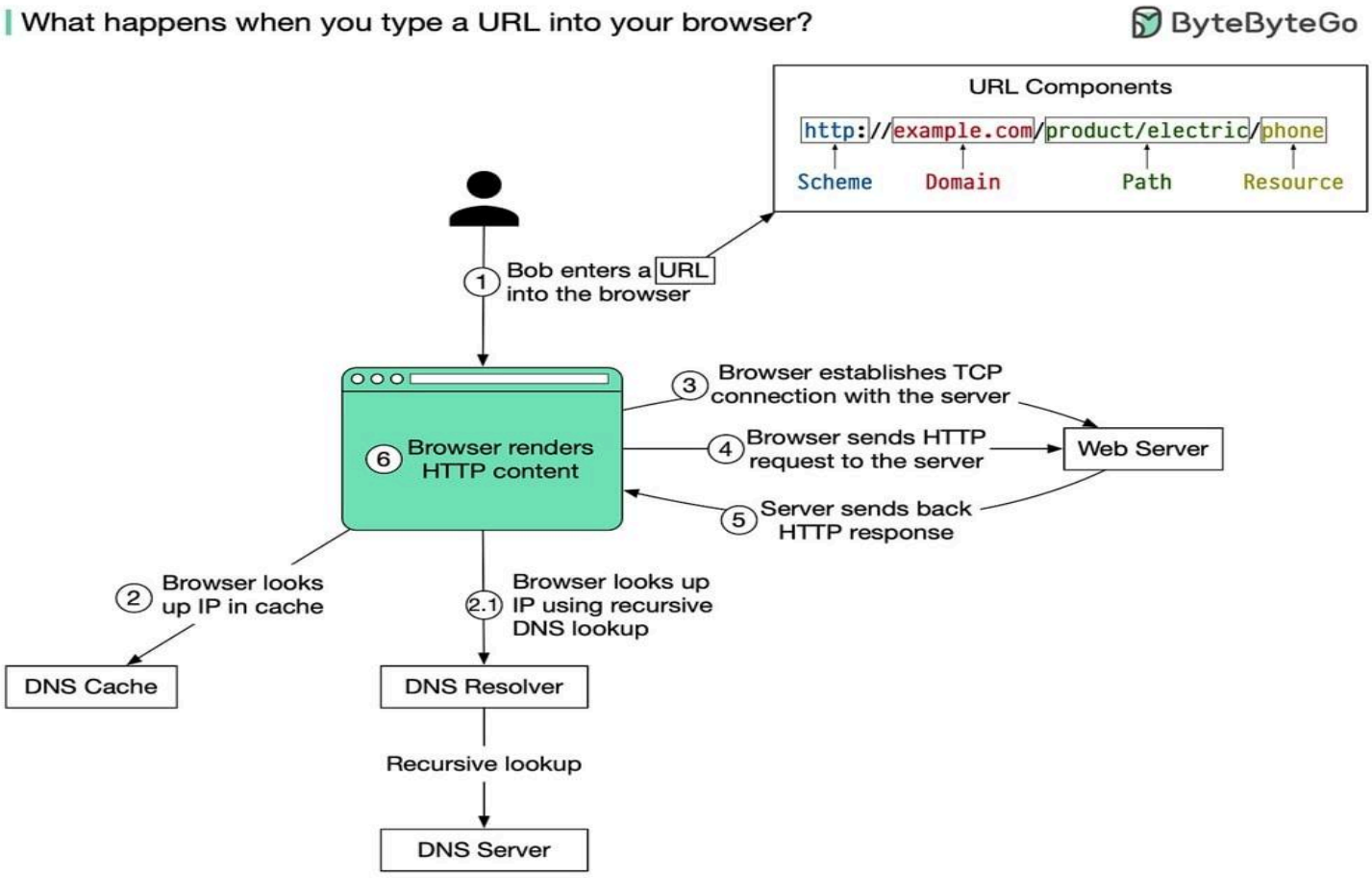
  - **LIRs or ISPs receive IP address blocks from RIRs or NIRs**
    Local Internet Registries (which are often ISPs or telecom operators) are given smaller ranges of IP addresses that they assign to their customers—either for residential broadband, business connections, or mobile data.

  - **Your ISP assigns a public IP address to your device**
    When you connect to the internet through your ISP, you are dynamically or statically assigned one of their public IPs. This address is part of a block originally assigned by a RIR and routed through ASN-based systems.

  - **Your requests are routed across the internet using your ISP-assigned IP**
    When you enter a URL, your request is sent using the assigned IP. Routers across networks (autonomous systems) forward your data based on ASN and IP routing tables, allowing servers to know where to return the requested information.

- What happens when you visit a website:

    - ➢ When we type our URL (Uniform Resource Locator) into our browser (client) in a computer (https://www.example.com)
    - ➢ Browser looks up IP in cache ⇒ (DNS Cache)
        - Else Browser looks up IP using recursive DNS lookup ⇒ **(DNS Resolver)** ⇒ Recursive lookup ⇒ **(DNS Server**
    - ➢ Once an IP address endpoint is found, the browser establishes TCP connection with the server.
    - ➢ Browser then sends HTTP request to the server,
    - ➢ Server sends back HTTP response.



What Browser Sends to the server

What's included:

- Request Line
  Example: `GET /about.html HTTP/1.1` → method, path, and protocol version

- Headers
  Key-value pairs that give metadata:

  - `Host`: Domain name
  - `User-Agent`: Info about the browser/device
  - `Accept`: What formats it can handle
  - `Cookie`: Any cookies for the site
  - `Referrer`: Where the request came from (if clicked from a page)
  - `Authorization`: (if logged in, includes token or credentials)

- Body (only in POST/PUT requests)
  Used to send data (like form data, JSON, file uploads)

```
GET /about.html HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/xhtml+xml
Accept-Language: en-US
Connection: keep-alive
```

What Response do we get from server

What's included:

- **Status Line**
    - Protocol + status code: `200 OK`, `404 Not Found`, `500 Internal Server Error`, etc.
- **Response Headers**
    - `Content-Type`: What's in the body (HTML, CSS, JSON, etc.)
    - `Content-Length`: Size of the body
    - `Set-Cookie`: For storing data like login tokens
    - `Cache-Control`, `ETag`, etc.: for performance/caching
- **Body**
    - This is the actual content: HTML, CSS, JS, JSON, image, etc.
    - The browser parses and renders it (for HTML), or uses it (e.g., displays an image)

What Happens After Receiving the Response?

1. If the response is HTML, the browser:
    - Parses HTML → builds the DOM
    - Loads external resources (CSS, JS, images) → sends more HTTP requests
2. If the response is JSON (API), JavaScript might update the page dynamically.
3. If it's a file (e.g., image), browser renders or prompts download.

Types of HTTP Methods Used:

- `GET`: Retrieve a resource (e.g., a web page)
- `POST`: Send data (e.g., form submission)
- `PUT`: Update a resource
- `DELETE`: Remove a resource
- `OPTIONS`: Ask what methods are supported

What is HTTP?

- **HTTP enables communication between browsers and web servers**
  It follows a request–response model where the browser sends a request, and the server replies with the appropriate content or data.
- **HTTP methods define the type of action being performed**
  Common methods include `GET` (retrieve), `POST` (submit), `PUT` (update), and `DELETE` (remove), each serving different use cases in web apps.
- **HTTP is stateless and relies on headers for context**
  Each request is treated independently. Headers carry metadata like content type, cookies, authentication tokens, and more.
- **Modern HTTP versions improve speed and efficiency**
  HTTP/1.1 allows persistent TCP connections, HTTP/2 supports multiplexing, and HTTP/3 uses QUIC over UDP for faster, more reliable delivery.
- **HTTPS is the secure version of HTTP**
  It encrypts data using TLS to protect users from man-in-the-middle attacks, and it's now essential for privacy, trust, and SEO ranking.
- **Status codes and ports define behavior and communication**
  Servers respond with status codes like `200 OK`, `404 Not Found`, etc. By default, HTTP uses **port 80** (and **443** for HTTPS).

What is a Domain Name?

- **A domain name is a user-friendly web address**
  It acts as a readable shortcut to a website's IP address. Instead of typing `192.0.2.1`, users can just type `example.com`.

- **Domains are part of URLs and help access websites**
  A full URL includes protocol and path, but the domain (like `example.com`) is the core part that connects to a website.

- **Domain names follow a hierarchical structure**
  They are divided into levels: Top-Level Domain (TLD) like `.com`, Second-Level Domain (SLD) like `example`, and optional subdomains like `www`.

- **DNS (Domain Name System) resolves domains to IPs**
  DNS acts like the internet's phonebook, converting domains into IP addresses so browsers can locate the correct server.

- **Domains must be registered and uniquely owned**
  You register domains through accredited registrars. Once claimed, they must be renewed periodically to retain ownership.

- **Domains link to hosting using DNS records**
  Records like A (IPv4), AAAA (IPv6), or CNAME point domains to web servers. Hosting and domain services can be from different providers.

What is Web Hosting?

1. **Web hosting provides an online space to store your website files**
   When you build a site, all its assets—HTML, CSS, JavaScript, images, videos—need to live on a server that's **always connected to the internet**. Hosting companies offer this service by renting out **server space**, so your site can be accessed globally 24/7.

2. **When users visit your domain, the hosting server delivers content**
   After a user types in your domain (like `yourwebsite.com`), the **DNS system** points the browser to your hosting provider's server. The server then returns the requested files using the **HTTP/HTTPS protocol**, enabling the browser to render your site.

3. **Shared hosting is a budget-friendly option with limited resources**
   In **shared hosting**, one physical server hosts multiple websites. All sites share CPU, RAM, and bandwidth. It's cheap and good for small sites, but **performance can suffer** if one site on the server uses too many resources (a "noisy neighbor" issue).

4. **VPS and dedicated hosting offer better performance and control**
   A **VPS (Virtual Private Server)** simulates multiple private servers on one machine, giving you more **isolation and control**. A **dedicated server** gives you full hardware access, perfect for high-traffic or custom-configured apps—but it's more expensive and requires server management skills.

5. **Cloud hosting offers scalability and reliability through virtualization**
   Cloud hosting (e.g., AWS, Google Cloud, Azure) runs your site across multiple virtual servers. If one server fails, another picks up the slack. It scales automatically during high traffic, supports modern deployment workflows, and is ideal for performance and uptime.

6. **Platforms like Netlify and Vercel streamline front-end hosting**
   These services are designed for **static sites** and **JAMstack** apps (JavaScript, APIs, Markup). They offer CI/CD (auto-deploy from GitHub), **serverless functions**, free **SSL**, custom domains, and **CDNs (Content Delivery Networks)** for fast global delivery.

7. **Hosting services often include DNS tools and SSL certificates**
   DNS (Domain Name System) tools let you **connect your domain** to your server using **A records**, **CNAMEs**, etc. SSL/TLS certificates (enabled via **HTTPS**) encrypt user data in transit and are essential for **security** and **SEO rankings**. Many hosts offer **free SSL via Let's Encrypt**.

8. **A fast, secure host improves performance, UX, and SEO**
   Google ranks sites better when they load quickly and use HTTPS. Hosting affects **TTFB (Time To First Byte)**, caching, and overall site reliability. Using a **CDN**, enabling **GZIP compression**, and choosing nearby data centers all help improve front-end performance.

What is DNS?

1. **DNS (Domain Name System) translates domain names into IP addresses**
   When you enter a website like `example.com`, DNS finds the matching **IP address** (like `93.184.216.34`) so your browser knows where to send the request. Without DNS, you'd have to remember numeric IPs for every site.

2. **A DNS lookup follows a multi-step resolution process**
   Your browser starts by checking the local cache. If not found, it asks a **recursive resolver** (often from your ISP), which contacts a **root server**, then a **TLD server** (like `.com`), and finally the **authoritative server** that holds the actual IP.

3. **DNS records define how domains behave**

   - **A record** points a domain to an IPv4 address.
   - **AAAA** does the same for IPv6.
   - **CNAME** creates an alias to another domain.
   - **MX** routes email.
   - **TXT** adds metadata (often for domain verification).
   - **NS** defines the authoritative name servers.

4. **Caching and TTL improve speed and reduce network load**
   DNS results are cached in browsers, operating systems, and recursive resolvers. **TTL (Time to Live)** defines how long a record stays cached before a new lookup is needed, reducing unnecessary traffic and improving performance.

5. **DNSSEC enhances DNS security against spoofing**
   Traditional DNS is vulnerable to attacks like **DNS spoofing** or **cache poisoning**. **DNSSEC** (Domain Name System Security Extensions) adds cryptographic signatures to DNS records, ensuring they haven't been tampered with.

6. **Modern DNS services offer speed, security, and control**
   Services like **Cloudflare**, **Google DNS (8.8.8.8)**, and **Quad9** provide fast and secure DNS with features like DDoS protection, built-in DNSSEC, analytics, and easy domain management—great tools for developers and site owners.

How Do Browsers Work?

1. **Browsers fetch HTML, CSS, and JavaScript files to build web pages**
   The **rendering engine** parses HTML into a **DOM tree**, styles into a **CSSOM**, and combines both to create a **render tree**, which is then laid out and painted to the screen.

2. **JavaScript is executed by the browser's JavaScript engine**
   JavaScript runs in the **main thread** and can interact with the DOM using browser-provided APIs (like `document.querySelector`). Chrome uses **V8**, Firefox uses **SpiderMonkey**, and both aim for high performance.

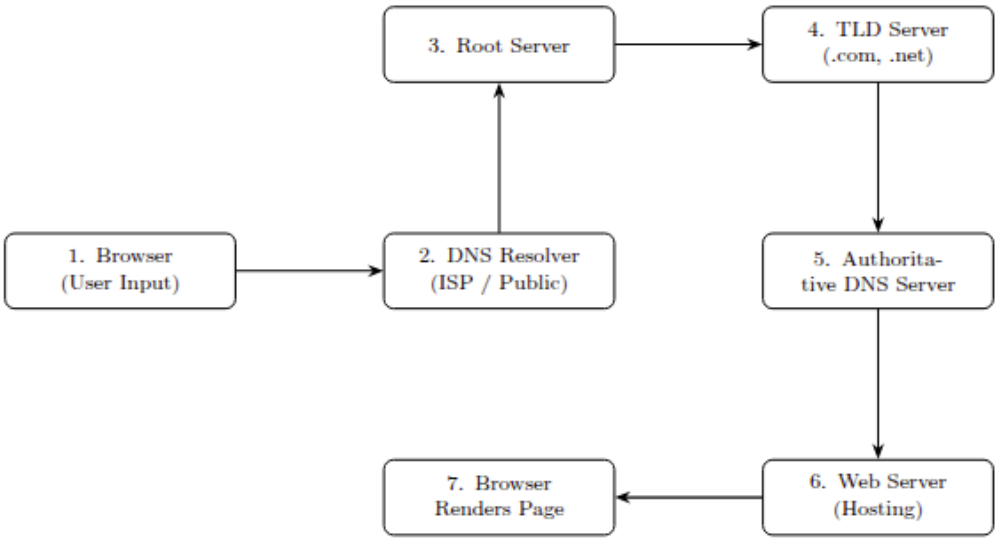
3. **Modern browsers use a multi-process architecture**
   Tabs, extensions, and GPU rendering are often split into isolated **sandboxed processes** to improve **security**, prevent crashes, and enhance **performance**.

4. **JavaScript runs in the event loop to manage asynchronous tasks**
   The **event loop** processes tasks like clicks, network responses (`fetch`), or timers (`setTimeout`) in sequence, ensuring the browser stays responsive while executing async operations.

5. **DevTools are essential for debugging and performance analysis**
   Browser Developer Tools let you inspect the DOM, monitor **network requests**, profile **JavaScript performance**, and detect **layout shifts**, which is vital for Core Web Vitals and optimization.

6. **Good front-end practices improve rendering and UX**
   To help the browser work efficiently, developers should avoid layout thrashing, defer or lazy-load non-critical JS, use **semantic HTML**, and minimize reflows for better performance and accessibility.

DNS to Browser Rendering Flow



Step-by-Step Explanation:

- **1. Browser (User Input)** – The user enters a website URL like `example.com` into the browser.
- **2. DNS Resolver** – The browser asks a DNS resolver (from your ISP or public DNS) to find the matching IP address.
- **3. Root Server** – The resolver queries a root DNS server, which points it to the correct Top-Level Domain (TLD) server.
- **4. TLD Server** – This server (e.g., for .com) tells the resolver where the domain's authoritative name server is located.
- **5. Authoritative DNS Server** – It contains the domain's actual DNS records and responds with the IP address of the web server.
- **6. Web Server (Hosting)** – The browser sends an HTTP request to this server, which hosts the website files.
- **7. Browser Renders Page** – The browser receives HTML, CSS, JS files and renders the final web page visually for the user.

SEO (Search Engine Optimization)

1. **SEO helps websites rank higher in search engine results**
   It improves visibility and drives organic traffic by aligning web content with what search engines look for.

2. **Search engines use bots to crawl and index websites**
   SEO ensures your content is easily understood, fast-loading, and relevant, so it appears in search results.

3. **Semantic HTML enhances content comprehension for crawlers**
   Tags like `<header>`, `<main>`, and `<article>` give structure, which helps bots and users understand your content layout.

4. **HTML tags like `<title>`, `<meta>`, `<h1>`–`<h6>`, and `alt` text are crucial**
   These elements tell crawlers what your page is about, which improves indexing and keyword targeting.

5. **Good form validation and UX reduce bounce rate**
   Search engines factor in user engagement—error-free, accessible forms improve retention and SEO indirectly.

6. **Security, performance, and structured data matter**
   HTTPS, fast loading (Core Web Vitals), and JSON-LD for rich results all influence how search engines rank your site.

HTML

1. **HTML (HyperText Markup Language) is the foundation of the web**
   Created in the early '90s, HTML started as a way to structure documents using simple tags. It's now the universal language of web content.

2. **HTML has evolved through major versions into HTML5**
   HTML5 introduced new features like multimedia support, semantic tags, native form controls, and APIs, enabling modern, responsive, and accessible web apps.

3. **HTML provides the structure for all web pages**
   It uses tags like `<div>`, `<h1>`, `<p>`, and `<img>` to define sections, headings, text, and media. HTML outlines the skeleton of a website.

4. **HTML is parsed into the DOM (Document Object Model)**
   Browsers convert HTML into a DOM tree — a hierarchical structure of elements that JavaScript can manipulate. This allows dynamic content updates and user interaction.

5. **Semantic HTML is a modern approach within HTML5**
   It uses tags like `<main>`, `<nav>`, `<section>`, and `<article>` to describe the client's role. This improves accessibility, SEO, and developer understanding of the layout.

6. **HTML works with CSS and JavaScript to build the full UI**
   HTML handles structure, CSS styles the layout, and JavaScript controls interactivity. These three technologies together form the front-end development stack.

7. **Browsers render HTML visually using the DOM structure**
   After building the DOM, browsers apply styles from the CSSOM, run JavaScript for interactions, and display everything visually — this is the rendering process

Forms and Validation

1. **Forms are used to collect and send user data to the server**
   They use input fields like `<input>`, `<textarea>`, `<select>`, which can be validated on the client and server side.

2. **Native HTML5 attributes handle basic validation**
   Features like `required`, `type="email"`, and `pattern` enforce correct input before submission.

3. **JavaScript offers advanced validation control**
   Custom rules, real-time error feedback, and conditional logic can improve UX, but should never replace server-side checks.

4. **Server-side validation is essential for security**
   Even if the front-end checks data, the back-end must revalidate to prevent manipulation or malicious input.

5. **Accessibility in forms matters for usability and SEO**
   Proper use of `<label>`, ARIA roles, and focus states ensures screen reader compatibility.

6. **Well-validated forms reduce bounce rate and increase engagement**
   A smoother form experience encourages users to complete actions like signups or purchases, improving site performance metrics.

Accessibility (a11y)

1. **Accessibility ensures usability for people with disabilities**
   This includes users with visual, hearing, motor, or cognitive impairments who rely on assistive tech.

2. **Semantic HTML supports screen readers and navigation**
   Structured tags let users understand content context, order, and importance.

3. **Forms need accessible labels and error feedback**
   Use `<label>`, `aria-live`, and clear instructions to ensure all users can interact with forms effectively.

4. **Keyboard navigation must be fully supported**
   Users should be able to navigate with `Tab`, `Enter`, and arrow keys through all interactive elements.

5. **Color contrast and text alternatives are critical**
   Use high-contrast text and `alt` attributes for images to ensure content is perceivable by all users.

6. **Accessibility overlaps with SEO and is a legal requirement**
   A11y best practices improve search visibility and may be mandatory under laws like ADA or WCAG compliance standards.

A Bit About the Back-End

1. **The back-end handles server-side logic and data**
   It processes requests, runs business logic, stores/fetches data from databases, and sends responses to the front-end.

2. **Common back-end languages include Node.js, Python, PHP, Java**
   Frameworks like Express, Django, or Laravel simplify building scalable APIs and web services.

3. **Front-end interacts with the back-end via APIs**
   The browser sends data using `fetch` or AJAX, and the server responds with HTML, JSON, or status codes.

4. **Authentication and session control happen on the back-end**
   It validates users using cookies, JWT tokens, or sessions, and controls who can access what.

5. **Form submissions are processed and stored by the back-end**
   It verifies inputs, saves data in databases like MySQL or MongoDB, and returns success/error responses.

6. **SEO-related tasks like rendering dynamic content or generating sitemaps are often back-end driven**
   Servers may insert meta tags or structured data before sending the final HTML, which crawlers rely on for indexing.