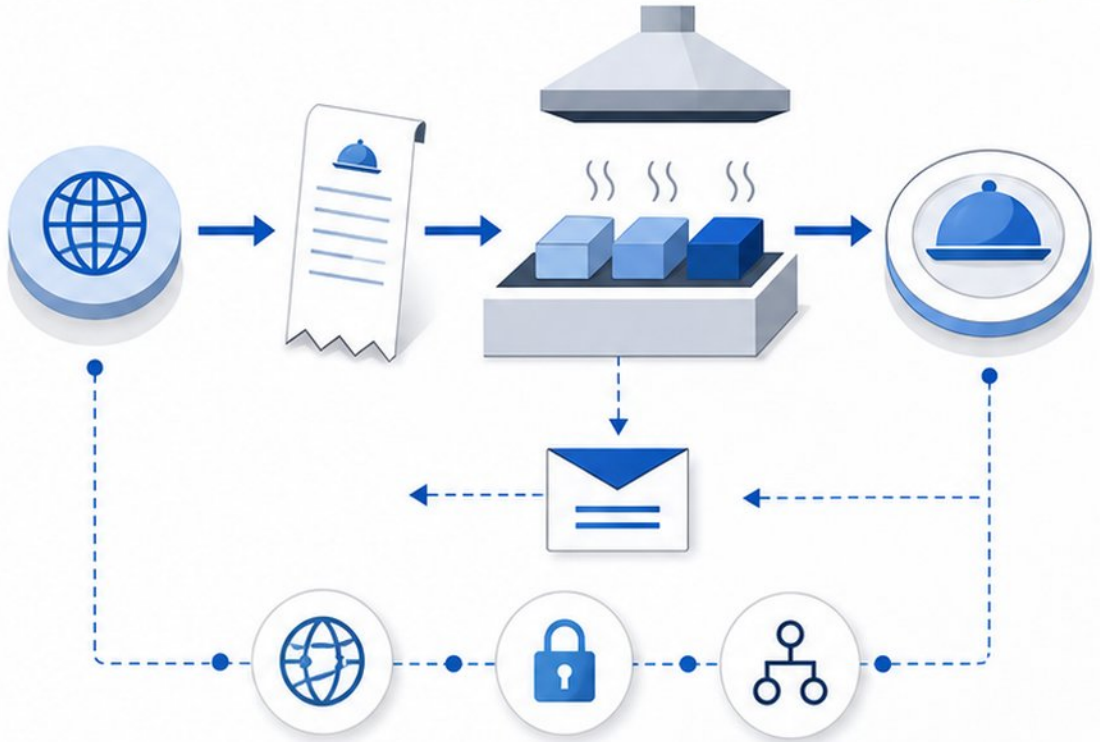
How Web Servers Work: From URL to Response

- A web server receives, processes, and responds to HTTP requests
- Like a restaurant kitchen fulfilling orders and delivering dishes
- Browser = customer; server = kitchen in the client-server model
- Journey: typing a URL to the page fully rendered on screen
- Key helpers: HTTP, DNS, TLS, Load Balancers streamline delivery



The Internet Foundation: HTTP, URLs, and DNS

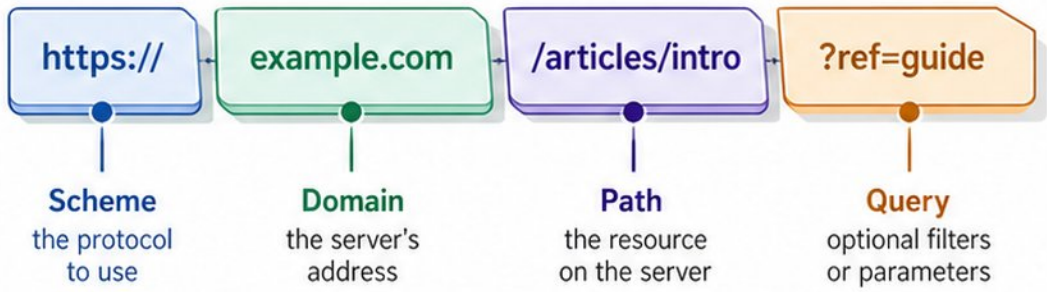
A URL's parts — scheme, domain, path, query — route traffic to a specific server resource

DNS acts as the internet's phonebook, converting domain names to IP addresses

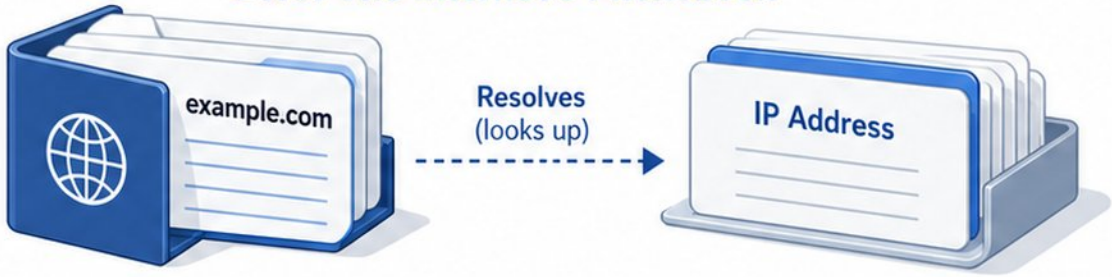
HTTP is the universal request-response language spoken between clients and servers

GET fetches data safely without side effects; POST sends data to create or trigger actions

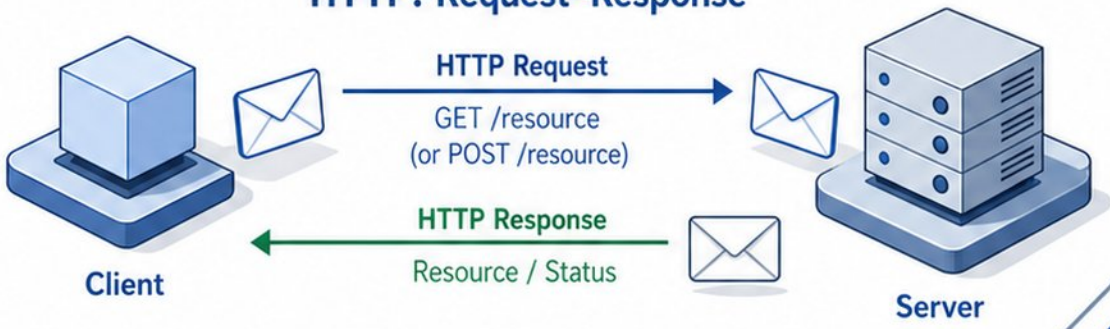
URL Anatomy



DNS: The Internet's Phonebook



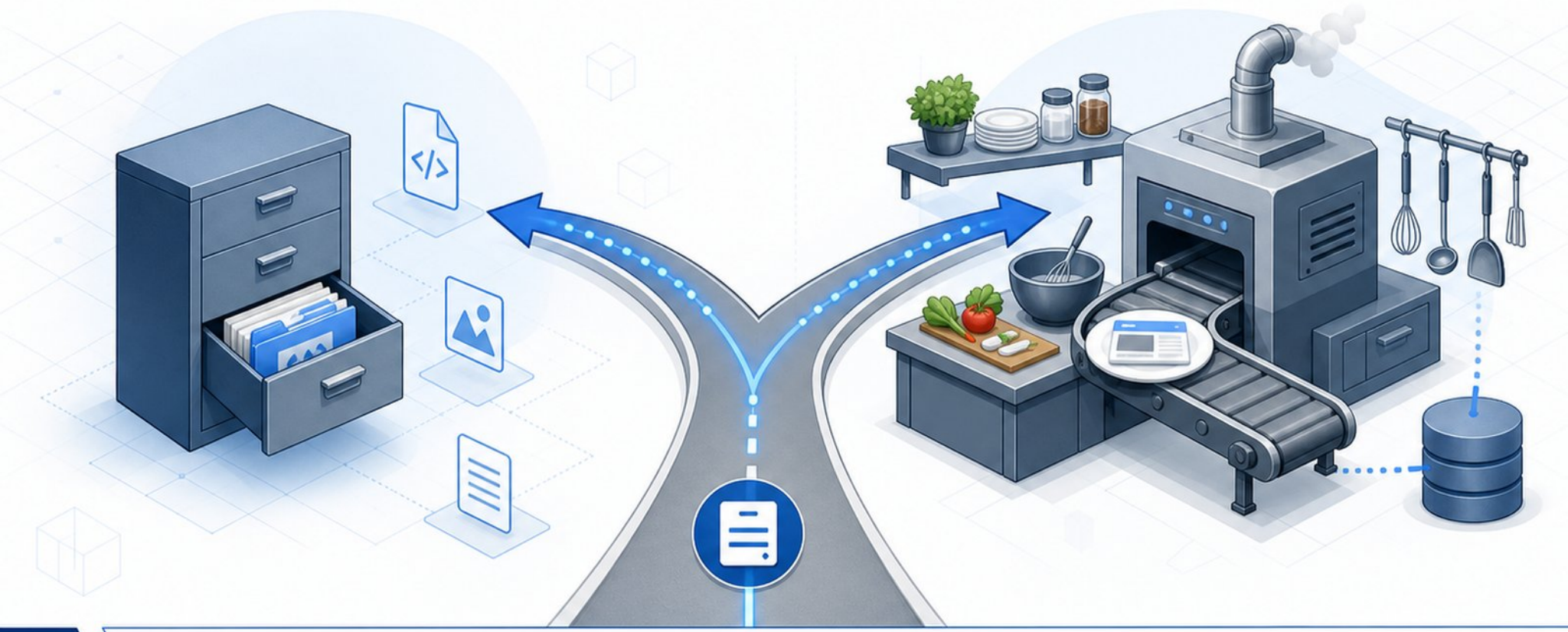
HTTP: Request–Response



Behind the Counter: Server Hardware and Software

- Physical servers, virtual instances, and cloud platforms (AWS, Azure, GCP) provide scalable capacity
- Web server software (Nginx 40.7%, Apache 35.6%) listens for and manages connections
- Server-side runtimes (Node.js, Python, PHP) generate dynamic content before responding
- Ports are numbered channels: HTTP listens on 80, HTTPS on 443

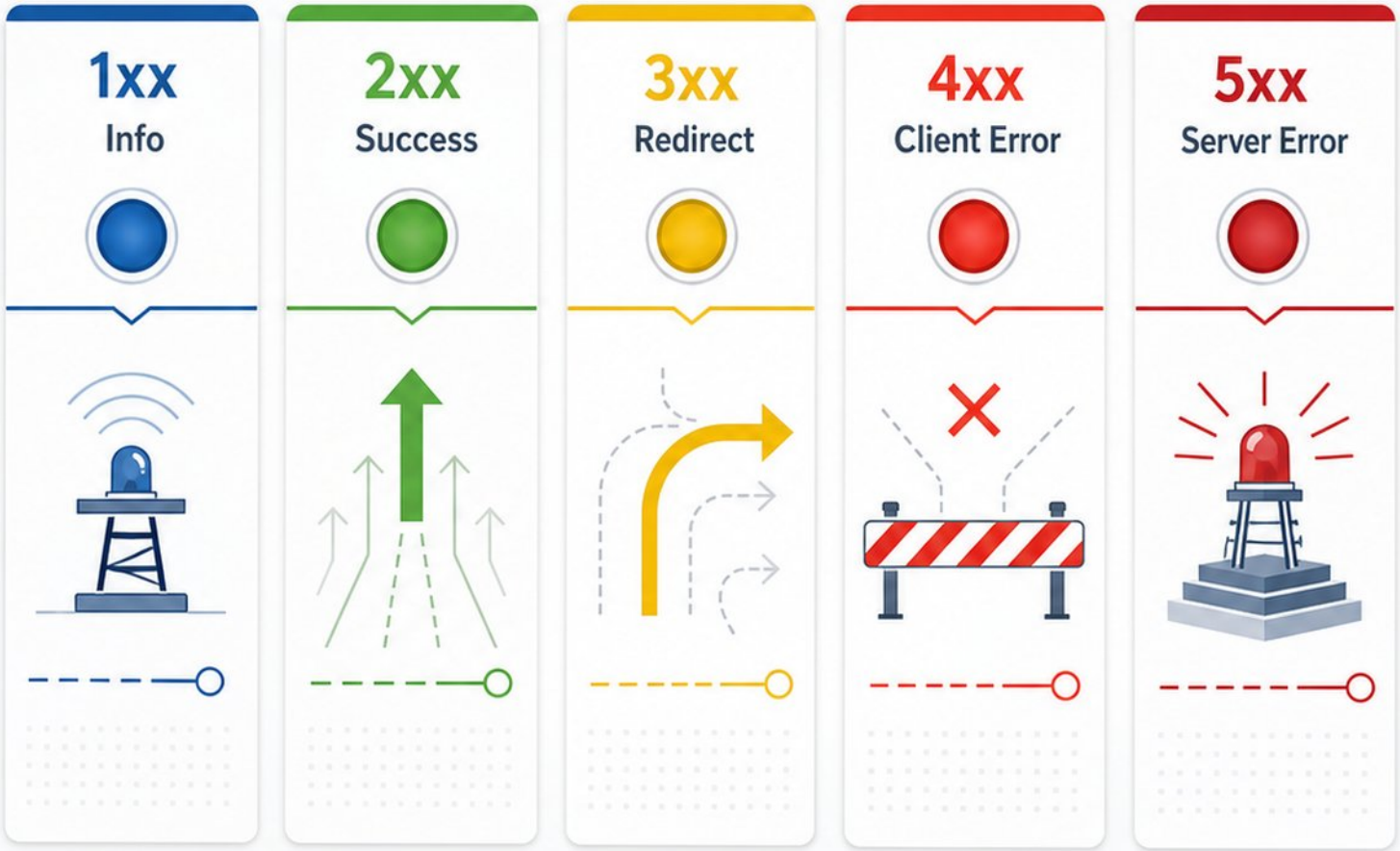
Request Handlers: How the Server Finds Your Content



- - Static content: pre-made files delivered directly from disk
- - Dynamic content: personalized pages built by server scripts in real time
- - Server maps requests using URL patterns and file extensions
- - MIME types (text/html, image/png) tell browsers how to interpret data
- - Logo request = static file; shopping cart = database-generated dynamic page

The Response Explained: Status Codes and Their Meanings

- Format: status line with code, response headers, and optional body
- Five families: 1xx Info, 2xx Success, 3xx Redirect, 4xx Client Error, 5xx Server Error
- Common codes: 200 OK, 301 Moved, 302 Found, 304 Not Modified, 404 Not Found, 500 Server Error
- 4xx errors mean client mistake (bad request); 5xx errors mean server fault (internal crash)

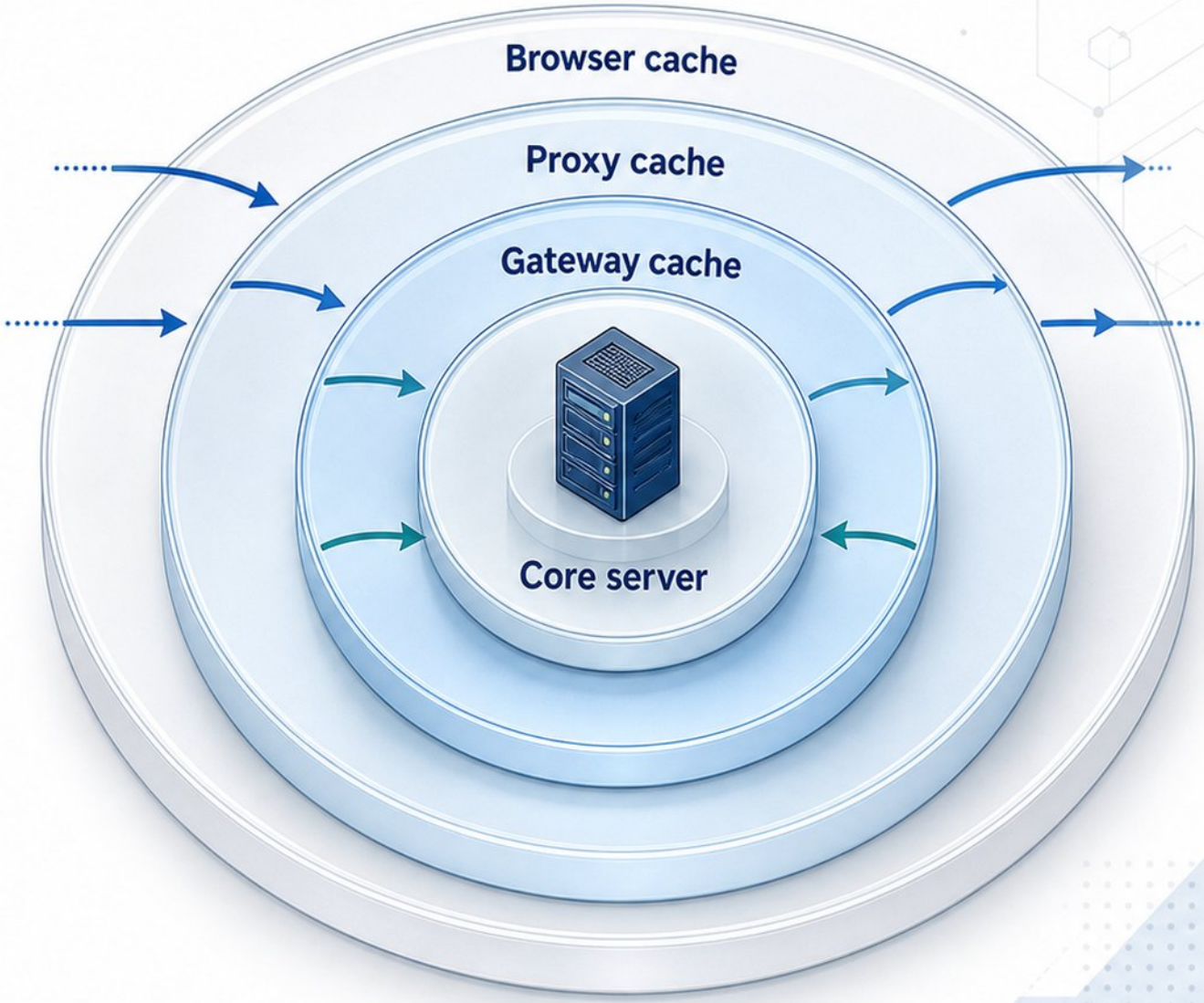


Response Headers:
The Server's Deepest Secrets

- Response headers carry metadata that controls caching, security, and rendering behavior.
- **`Content-Type`** tells the browser whether it's receiving HTML, an image, or a PDF.
- **`Content-Length`** announces the file size so the browser knows when the download finishes.
- The **`Cache-Control`** family commands browsers and CDNs to keep local copies for a set time.
- **`ETag`** headers enable fast validation so unchanged files aren't re-downloaded.

Caching: The Art of Not Repeating Yourself

- Storing responses near users to avoid re-generation
- Browser, proxy, and gateway caches form layered defense
- ETag/If-None-Match validates freshness, returns 304 Not Modified
- max-age sets TTL, the expiration timer for cached content



Making It Secure: HTTPS and the TLS Handshake

- HTTPS wraps HTTP in a TLS encryption layer, ensuring privacy and integrity.
- TLS certificates act as digital passports, verified by trusted Certificate Authorities.
- The TLS handshake is a rapid secret greeting to agree on ciphers and session keys.
- HTTPS is now mandatory for all modern websites, not just banking.

Scaling for Millions: Load Balancers

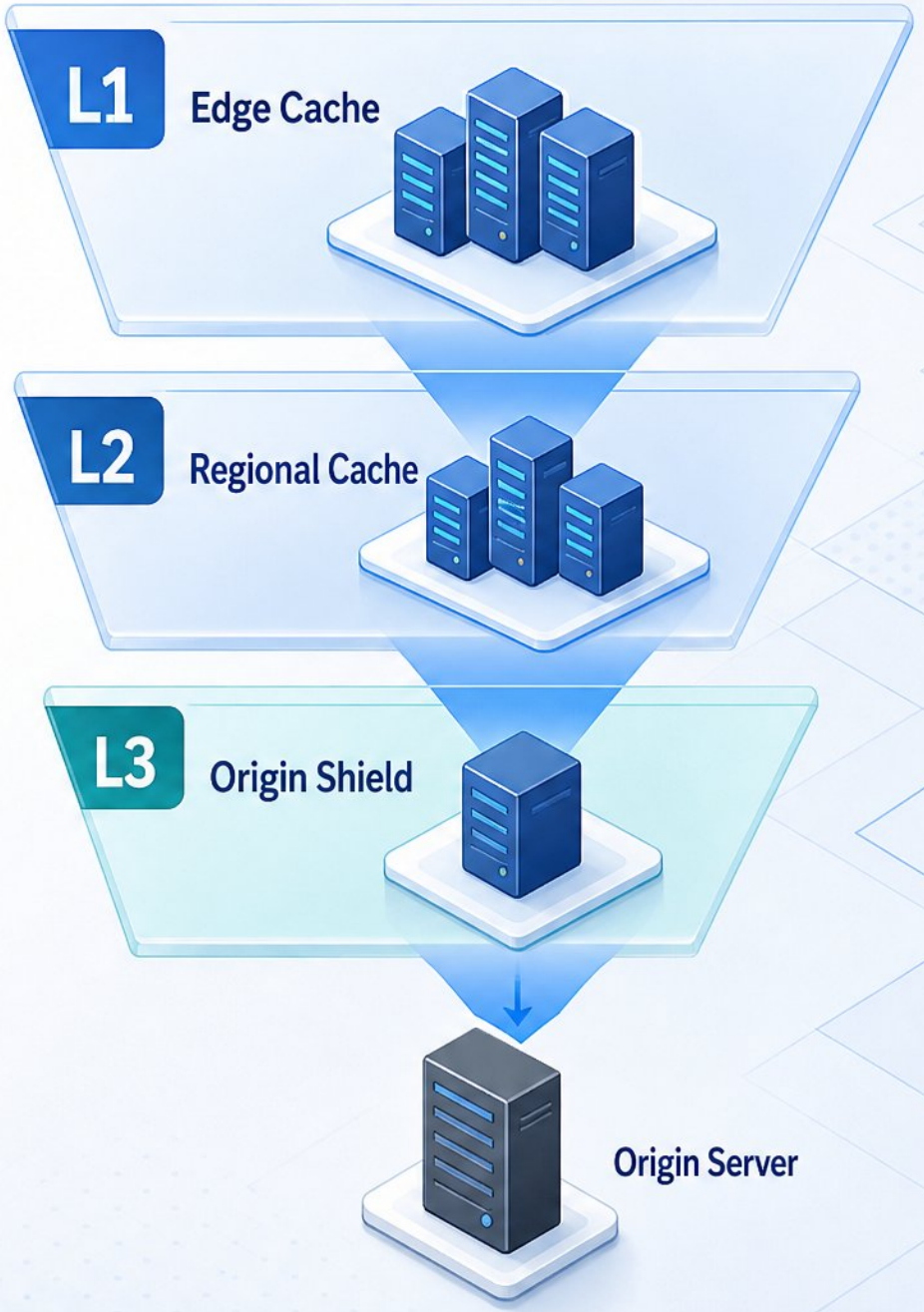
- Single server hits a traffic bottleneck, like one cashier facing a crowd
- Load balancer acts as "traffic director" across a server pool
- Algorithms: Round Robin, Least Connections, and IP Hash
- Sticky sessions keep a user's cart intact across refreshes

Global Delivery: Content Delivery Networks (CDNs)

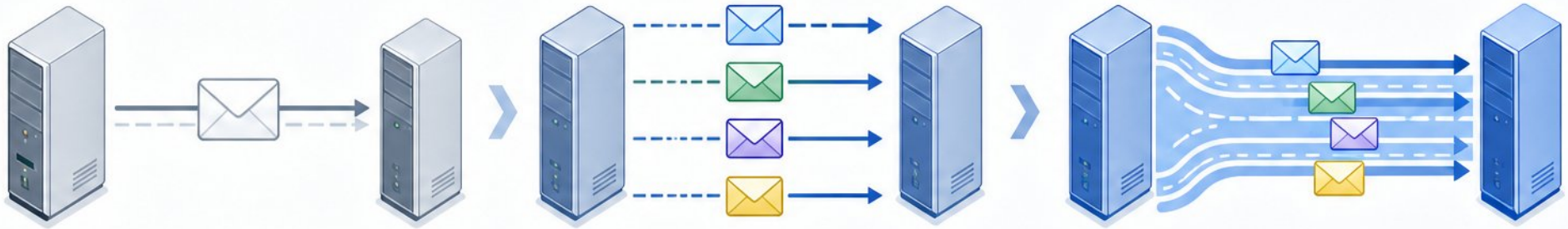
- A CDN is a global network of edge servers caching content near users for fast delivery
- A Tokyo user fetches from a local edge, not an origin server in Virginia
- DNS routing (Anycast or GeoDNS) sends each user to their nearest Point of Presence
- Beyond speed: CDNs absorb DDoS attacks and shield your origin infrastructure

Power at the Edge: Tiered Caching and Edge Computing

- Tiered caching: L1 edge → L2 regional → L3 origin shield before hitting your server
- Solves the 'thundering herd' by collapsing thousands of expired requests into one origin fetch
- Edge computing runs custom code directly on CDN servers without contacting your origin
- Use cases: A/B testing, geo-redirection, and authentication with zero added latency



Modern Voice: HTTP/2 and HTTP/3



HTTP/1.1

One connection.
One file at a time.

HTTP/2

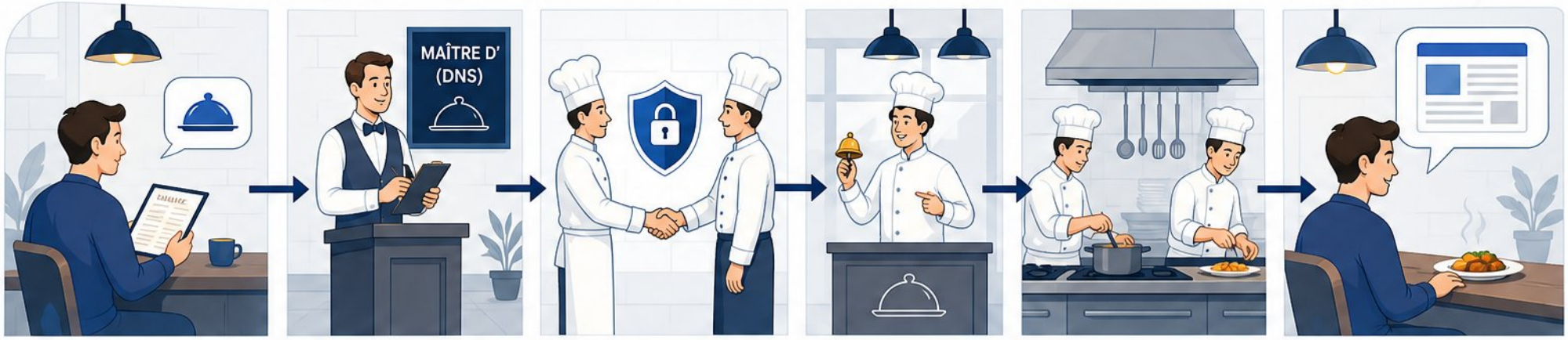
Many files in parallel
over one TCP connection.

HTTP/3

QUIC (UDP) highway.
No head-of-line blocking.

- - HTTP/2 multiplexes many files in parallel over one TCP connection
- - HTTP/3 switches to QUIC (UDP), eliminating head-of-line blocking
- - This evolution powers fluid page loads, smooth video, and VoIP
- - Modern servers and CDNs drive protocol upgrades transparently

The Full Journey: End-to-End Request Walkthrough



BROWSER
You place an order (enter the URL)

DNS
Finds the best table (optimal IP address)

TLS / QUIC
Securely greet and open the line to the kitchen

LOAD BALANCER
Host routes your order to the right station

SERVER
Kitchen prepares your meal and adds a status

BROWSER
You enjoy the meal (page is painted)

- Browser resolves domain via DNS to optimal IP address
- QUIC connection and 0-RTT TLS handshake established
- Load-balanced server handles request, returns response with status
- HTML parsed; parallel CDN requests for assets paint the final page

Course Summary and Your Path Forward

- Web server receives, processes, and responds to HTTP requests
- DNS, TLS, load balancers, and CDNs provide security and scale
- All web apps use the same request-response foundation
- Try Nginx locally, inspect DevTools, and deploy on free cloud tiers

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/ea807f39/public