

Lecture 1: Introduction to HLD (High-Level Design)

◆ System Design

System Design do major parts me divide hota hai:

- HLD (High-Level Design) 🏗️ → Application ka **Architecture design**.
 - LLD (Low-Level Design) 🖥️ → Application ka **Coding design**.
-

🏗️ HLD (High-Level Design)

👉 HLD ka focus hota hai **Application ka Architecture banane me**.

📌 Example: Home Building Analogy

- Cement kaunsa hoga?
- Bricks kaise hongi?
- Floor design kaisa hoga?
- Internal structure kaise banega?

Same tarah Application ke liye bhi planning hoti hai.

🔑 Main Considerations in HLD

1. Tech Stack Selection ⚙️

- Example: MERN, Spring Boot, Django.
- Reasoning clear honi chahiye: *Kyon Spring Boot? Kyon Django?*
- Sirf comfort basis par tech stack choose nahi karna.

2. Cost Optimization 💰

- Application banane ka cost.
- Deployment ke baad server cost (monthly).

- Ensure karo ki Application profitable ho.

3. Database Selection

- Options: SQL, MySQL, NoSQL, GraphQL, Column DB etc.
 - Decide karo → Kaunsa DB application ke liye suitable hai.
 - **Scalability:**
 - 10M users (Flipkart) vs 100M users (CBSE results).
 - Agar users badh jaate hain toh **Scaling, DB migration, Tech stack change** karna padta hai.
 - Ye ek **complicated task** hota hai → Isiliye HLD me pehle se architecture design kiya jata hai.
-





LLD (Low-Level Design)

 Focus hota hai **Coding level design** pe:

- OOPs concepts kaise follow karenge.
- Design Principles (SOLID etc.).
- Design Patterns ka use.

 **Goal of LLD:**




- Clean code 
- Loosely Coupled code  (ek component dusre se tightly dependent na ho).



HLD → Architecture (Application)



Application Types

- Software/Application हो सकते हैं:
 -  Website
 -  Mobile Application
 -  Software (e.g., Photoshop)

👉 Photoshop जैसे software server पर depend नहीं करते → इसलिए वहाँ ज़्यादा **LLD (Low-Level Design)** पर focus होता है।

👉 लेकिन **Website और Mobile Applications** में server-client model होता है → इसलिए यहाँ **HLD** का importance ज़्यादा है।

⚡ HLD का Primary Focus

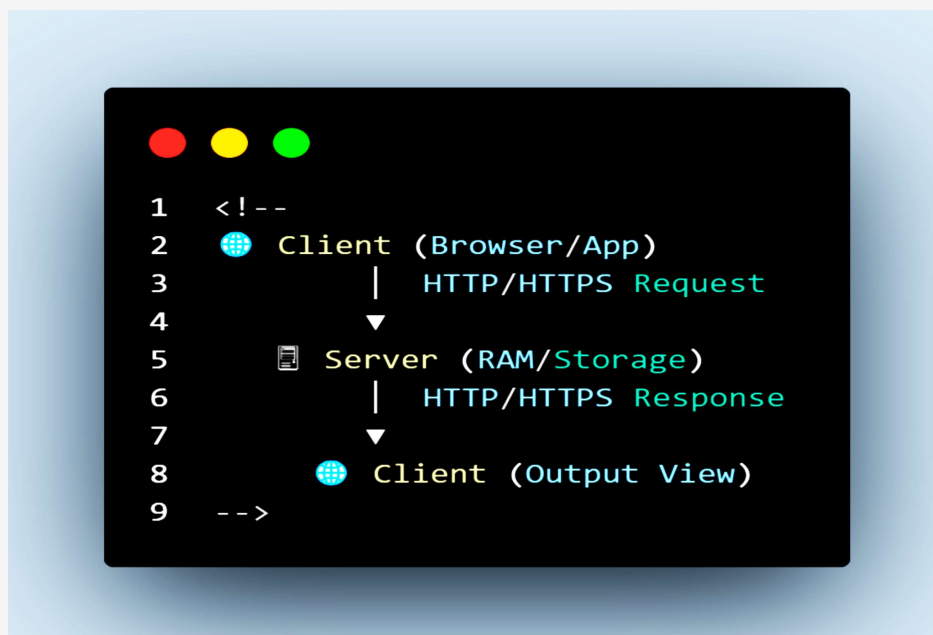
- HLD का focus होता है **internet-based applications** पर।
 - Example → YouTube App, Flipkart Website
-

🔄 Request–Response Model (Client–Server)

- Internet पर हर website/app दो parts में divide होती है:
 - **Client (Web Browser / Mobile App)**
 - **Server (जिसके पास RAM, Storage, Processing power होता है)**

📌 Working:

1. Client → Server को **HTTP/HTTPS Request** भेजता है
2. Server → Request process करके **HTTP/HTTPS Response** भेजता है



👉 इसे ही कहते हैं **Request–Response Model** या **Client–Server Model**

⚙️ HLD & Distributed Systems

- HLD हमेशा **Distributed System** पर based होता है।
 - मतलब → अलग-अलग servers को **एक साथ connect करके** use किया जाता है।
-

📊 Scaling Approaches

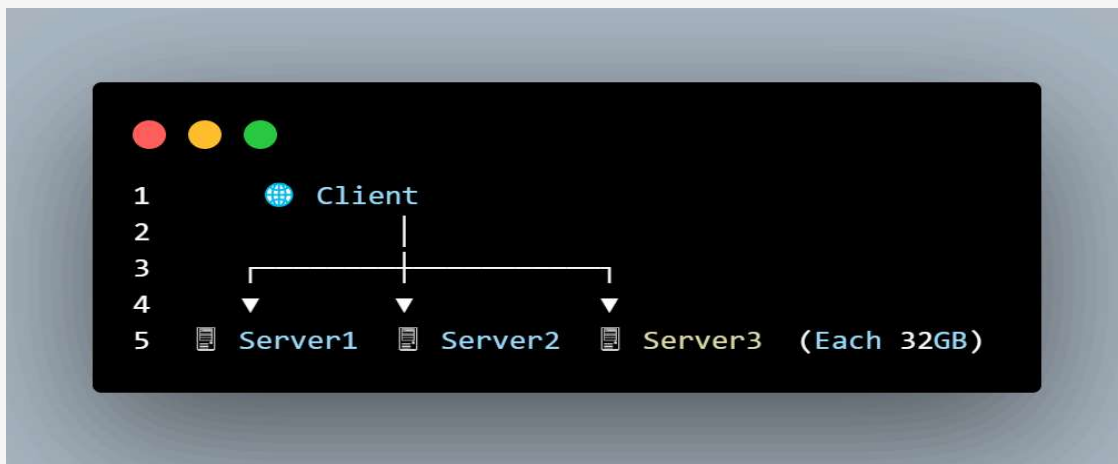
① Vertical Scaling ⚠️

- एक server की RAM/CPU बढ़ाकर ज्यादा users handle करना।
- Example: 32GB RAM server → 64GB RAM add करना।
- Limitation → Real projects में practical नहीं होता (क्योंकि hardware की limits होती हैं)।



② Horizontal Scaling ✅

- Server पर load बढ़े तो **नया server add** कर दो।
- Multiple servers मिलकर requests handle करते हैं।
- यह scalable और practical approach है → Real-world systems में यही use होता है।



👉 इस model में हर नया server एक **Node** कहलाता है।

📄 Horizontal Scaling → Nodes

- जब हम **servers** को **horizontally scale** करते हैं → उन्हें **Nodes** कहते हैं।
- Example: **S1, S2, S3, S4** → अलग-अलग servers workload share करते हैं।




⚠️ Problem:

- ज्यादा servers → structure **complex** हो जाता है।
- Challenge → ये servers आपस में कैसे **communicate** करें?
- Communication → **HTTP Calls** के through होता है।
- Important → **Data Integrity** maintain रहनी चाहिए।

Network Communication (Protocols)

- जब client और server communicate करते हैं → वो **Rules (Protocols)** follow करते हैं।
 - ये rules **ISO–OSI Model (7 Layers)** पर based हैं।
 - Decide करते हैं कि **2 systems network के through** कैसे बात करेंगे।
-

Application Layer Protocols

 Client और Server के बीच communication establish करने के लिए use होते हैं।

- **HTTP / HTTPS** → Web communication
- **FTP / SFTP** → File transfer (Server पर upload/download)
- **SMTP** → Mail भेजने के लिए (e.g., Gmail sending mail)
- **POP3 / IMAP** → Mail receive करने के लिए
- **WebRTC / WPSRTC** → Real-time communication
 - Example → WhatsApp, Jio
 - Client server को msg भेजता है
 - Server बिना HTTP के directly client को msg वापस भेज सकता है

Client ⇌ HTTP/HTTPS ⇌ Server


Client ⇌ WebRTC ⇌ Server ⇌ Client (Realtime Msg)

WebSockets (Peer-to-Peer)

- WebSocket का use **real-time peer-to-peer communication** के लिए होता है।
- यहाँ पर clients आपस में **direct communicate** करते हैं → server की ज़रूरत नहीं होती।

Client 1 ⇌ Client 2

(Direct Connection using WebSockets)

 Example: Chat apps (कुछ जगह पर server सिर्फ initial connection setup करता है, बाद में direct client-to-client communication possible होता है)।

⚡ TCP vs UDP

Protocol	Working	Speed	Reliability	Use Cases
TCP (Transmission Control Protocol)	Data को छोटे packets में divide करता है और हर packet का ACK (Acknowledgement) लेता है।	Slow ⌚	Reliable ✅ (100% delivery confirm)	Web apps, File transfer, Banking
UDP (User Datagram Protocol)	Packets directly भेजता है, ACK नहीं लेता।	Fast ⚡	Non-Reliable ❌ (कुछ packets lost हो सकते हैं)	Live Streaming, Online Gaming, Google Meet, Live News

📌 Diagram Representation:

TCP:

```
MSG → [P1][P2][P3] → Server
      ↑
      ACK (Confirm each packet)
```

UDP:

```
MSG → [P1][P2][P3] → Server
      (No ACK, Fast but Unreliable)
```

🌐 DNS (Domain Name System)

- Client और Server communication के लिए **IP Address** की जरूरत होती है।
- लेकिन user हमेशा domain name (जैसे **youtube.com**) type करता है → IP याद रखना मुश्किल होता है।
- **DNS का काम:** Domain Name को IP Address में convert करना।

📌 Flow:

```
Client → DNS Server → Finds IP of youtube.com → Returns IP
Client → Connects to YouTube Server using that IP
```

👉 Example: YouTube से connect होने के लिए पहले उसका IP जानना ज़रूरी है।

🌟 Quick Recap

- **WebSockets** → Peer-to-peer direct communication, server involvement कम।
- **TCP** → Reliable, Slow, ACK based.
- **UDP** → Fast, Unreliable, कोई ACK नहीं।
- **DNS** → Domain को IP में convert करता है ताकि

Short Summary :

◆ **System Design** → 2 parts:

- **HLD** → Architecture design (Tech stack, DB, Scaling, Cost).
- **LLD** → Code-level design (OOPs, SOLID, Patterns).

◆ **HLD Focus** → Websites & Apps (Client–Server Model).

Request–Response Model

Client → Request → Server → Response → Client

◆ **Scaling**

- **Vertical** → Single server upgrade (limited).
- **Horizontal** → Multiple servers = Nodes (practical).

◆ **Protocols (Communication Rules)**

- **HTTP/HTTPS** → Web
- **FTP/SFTP** → File transfer
- **SMTP/IMAP/POP3** → Mail
- **WebRTC/WebSockets** → Real-time

◆ **WebSockets (P2P)** → Direct client-to-client communication (server not always needed).

◆ **TCP vs UDP**

- **TCP** → Reliable, Slow, ACK (Web apps, Banking).
- **UDP** → Fast, Unreliable, No ACK (Streaming, Gaming).

◆ **DNS** → Converts Domain → IP (youtube.com → IP).

✨ **Quick Keywords to Remember:**

HLD = Architecture | LLD = Code | Scaling = Vertical/Horizontal | Protocols = Rules |
WebSockets = P2P | TCP = Reliable | UDP = Fast | DNS = Domain → IP