System Design Study Notes

I. System Design Fundamentals

System Design is the art of defining the architecture, modules, and data flow of a system to satisfy specific requirements. It is rarely about finding a "perfect" solution; it is about managing **trade-offs**.

1. HLD vs. LLD

- **High-Level Design (HLD):** The "Bird's Eye View." It focuses on the macro-architecture, including load balancers, microservices, databases, and how they connect.
- **Low-Level Design (LLD):** The "Micro View." It focuses on class diagrams, method signatures, design patterns (like Singleton or Factory), and specific logic implementation.

> **Interview Talk Track:** > "When approaching a problem, I start with HLD to establish the blueprint—identifying where data lives and how services talk. Once the infrastructure is solid, I move to LLD to define the internal logic, ensuring the code itself is maintainable and follows SOLID principles."

2. Requirements Gathering

- **Functional:** The features. If it's a banking app, "transferring money" is functional.
- **Non-Functional (NFRs):** The "ilities." Scalability, Availability, Reliability, and Maintainability.

> **Interview Talk Track:** > "Before I draw a single box, I clarify the requirements. Functional requirements tell me what the system *does*, but Non-Functional requirements tell me if the system can *survive* 10 million users. I always ask about the expected scale and acceptable downtime first."

II. Core Performance Metrics

1. Scalability: Vertical vs. Horizontal

- **Vertical (Scaling Up):** Adding more power (CPU, RAM) to an existing machine. It's easy but has a "glass ceiling" (hardware limits) and creates a single point of failure.
- **Horizontal (Scaling Out):** Adding more machines to the pool. This is the industry standard for large systems because it allows for infinite growth and better fault tolerance.

2. Latency vs. Throughput

- **Latency:** The time it takes for a single packet to travel from point A to point B (measured in milliseconds).
- **Throughput:** The number of requests the system can handle in a given time frame (e.g., Requests Per Second).

> **Interview Talk Track:** > "I think of latency as the speed of a single car on a highway, while throughput is how many lanes the highway has. You can have a fast car (low latency), but if there's only one lane, the highway can't handle much traffic (low throughput). In system design, we aim for the 'sweet spot' where we handle high volume without making the user wait."

III. Networking & Request Flow

1. The "Google.com" Flow

When a user hits "Enter" in a browser:

1. **DNS Resolution:** The browser checks cache, then asks a DNS server to resolve `google.com` to an IP address (like an operator looking up a phone book).
2. **TCP Handshake:** The browser establishes a connection with the server (TCP for reliability).
3. **HTTP Request:** The browser sends a GET request for the webpage.
4. **Processing:** The server (and potentially a Load Balancer/Database) processes the request and sends back a response (HTML/JSON).
5. **Rendering:** The browser renders the UI for the user.

2. TCP vs. UDP

Feature	TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)
Connection	Connection-oriented (Handshake)	Connectionless
Reliability	Guarantees delivery & order	No guarantee (Fire and forget)
Speed	Slower (due to overhead)	Faster

Use Case	Web browsing, Email, Database	Video streaming, Gaming, VoIP

Interview Talk Track: > "If data integrity is non-negotiable—like in a bank transaction—I'd always choose TCP. However, if we're building a live streaming service where a lost frame is better than a lagging video, UDP is the better choice because of its low overhead."

IV. APIs & Communication

APIs are the "contracts" that allow two pieces of software to talk to each other.

1. Common HTTP Methods

- **GET:** Read data.
- **POST:** Create data.
- **PUT:** Update/Replace data.
- **DELETE:** Remove data.

2. Status Codes to Remember

- **2xx (Success):** 200 OK, 201 Created.
- **4xx (Client Error):** 400 Bad Request, 401 Unauthorized, 404 Not Found.
- **5xx (Server Error):** 500 Internal Server Error, 503 Service Unavailable.

Interview Talk Track: > "A well-designed API should be RESTful and stateless. It should return the correct status codes so the frontend knows exactly what went wrong—for instance, returning a 429 if the user is hitting our rate limits, rather than a generic 500 error."

Summary Table for Quick Revision

Concept	Key takeaway
Availability	Is the system "up"? (Measured in 9s, e.g., 99.9%)

Reliability	Does the system do what it’s supposed to do without errors?
DNS	Mapping human-readable names to machine-readable IPs.
Horizontal Scaling	Best for high availability; uses Load Balancers.

Would you like me to create a set of practice interview questions based on these Day 1–4 notes to test your knowledge?

That’s the spirit! Testing your knowledge is the best way to move from "I read it" to "I can explain it to a Senior Engineer."

Here are **10 practice interview questions** based on your Day 1–4 notes, ranging from basic definitions to analytical "trade-off" questions.

Category 1: Fundamentals & Requirements

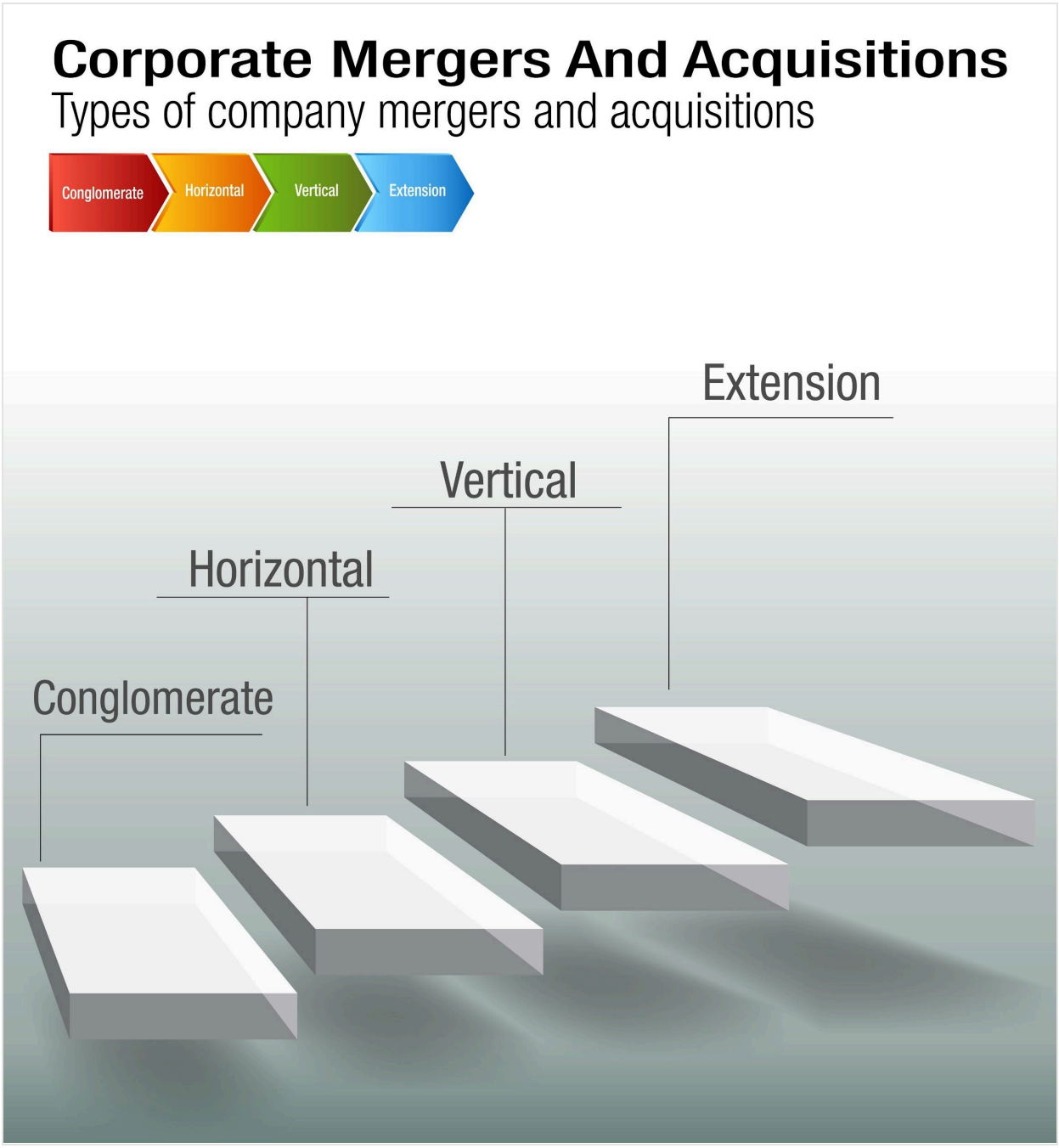
1. "A stakeholder says the system must be 'fast.' How do you translate this into technical requirements?"

- **What they are looking for:** Your ability to distinguish between Functional and Non-Functional requirements.
- **Sample Answer:** "I’d clarify if 'fast' means low **Latency** (the time for one request) or high **Throughput** (handling many users at once). I’d also define a target, like '95% of requests must complete under 200ms'."

2. "Can a system be Relevant but not Reliable? Give an example."

- **What they are looking for:** Understanding that features (functional) are useless if the system is down (non-functional).
- **Sample Answer:** "Yes. An app like WhatsApp might have a 'relevant' feature like Video Calling, but if the servers crash every time someone starts a call, it isn't 'Reliable.' Reliability ensures the feature works consistently under stress."

Category 2: Scalability & Performance



Shutterstock

3. "We have a single-server database that is hitting 90% CPU usage. Should we Scale Up or Scale Out?"

- **What they are looking for:** Your understanding of the trade-offs between Vertical and Horizontal scaling.

- **Sample Answer:** "Scaling Up (Vertical) is a quick fix but has a limit. Scaling Out (Horizontal) is better for long-term growth and high availability. However, Horizontal scaling for databases is complex because you have to handle data consistency across nodes."

4. "If I decrease Latency, does Throughput always increase?"

- **What they are looking for:** Understanding the relationship (they are related but not identical).
- **Sample Answer:** "Generally, yes, because faster requests free up resources sooner. However, if your system is bottlenecked by a limited number of worker threads, decreasing latency won't help throughput if those threads are already at 100% capacity."

Category 3: Networking & APIs

5. "Explain the 'DNS Lookup' process like I'm a non-technical manager."

- **What they are looking for:** Simple analogies.
- **Sample Answer:** "It's like the Contacts app on your phone. You know the name 'Piyush,' but your phone needs the 'Phone Number' to make the call. DNS takes the name 'google.com' and finds the 'IP Address' (the number) so the computer knows where to send the data."

6. "Why do we use HTTPS instead of HTTP for a login page?"

- **What they are looking for:** Security basics (Encryption).
- **Sample Answer:** "HTTP sends data in 'Plain Text.' If someone intercepts the packet, they can see the password. HTTPS encrypts that data, so even if it's intercepted, it looks like gibberish to the attacker."

7. "What is the difference between a 401 and a 403 error?"

- **What they are looking for:** Precision in status codes.
- **Sample Answer:** "401 (Unauthorized) means I don't know who you are—please log in. 403 (Forbidden) means I know who you are, but you don't have permission to see this specific page."

Category 4: The "Deep Dive" (Scenario Based)

8. "You are designing a Multiplayer Game. Which protocol would you use for player movement: TCP or UDP?"

- **Sample Answer:** "UDP. In gaming, speed is more important than 100% accuracy. If one 'movement packet' is lost, the next one will arrive 10ms later anyway. TCP's 're-transmission' logic would cause the game to lag or stutter."

9. "What are the 3 most important Non-Functional Requirements for a Banking App?"

- **Sample Answer:** "**Reliability** (transactions must never fail), **Security** (data must be encrypted), and **Consistency** (the balance must be the same whether I check it on my phone or a laptop)."

10. "Why is Horizontal Scaling considered 'Fault Tolerant' while Vertical Scaling is a 'Single Point of Failure'?"

- **Sample Answer:** "In Vertical Scaling, if that one big server dies, the whole app dies. In Horizontal Scaling, if one of our 10 small servers dies, the Load Balancer just sends traffic to the other 9. The user never notices."