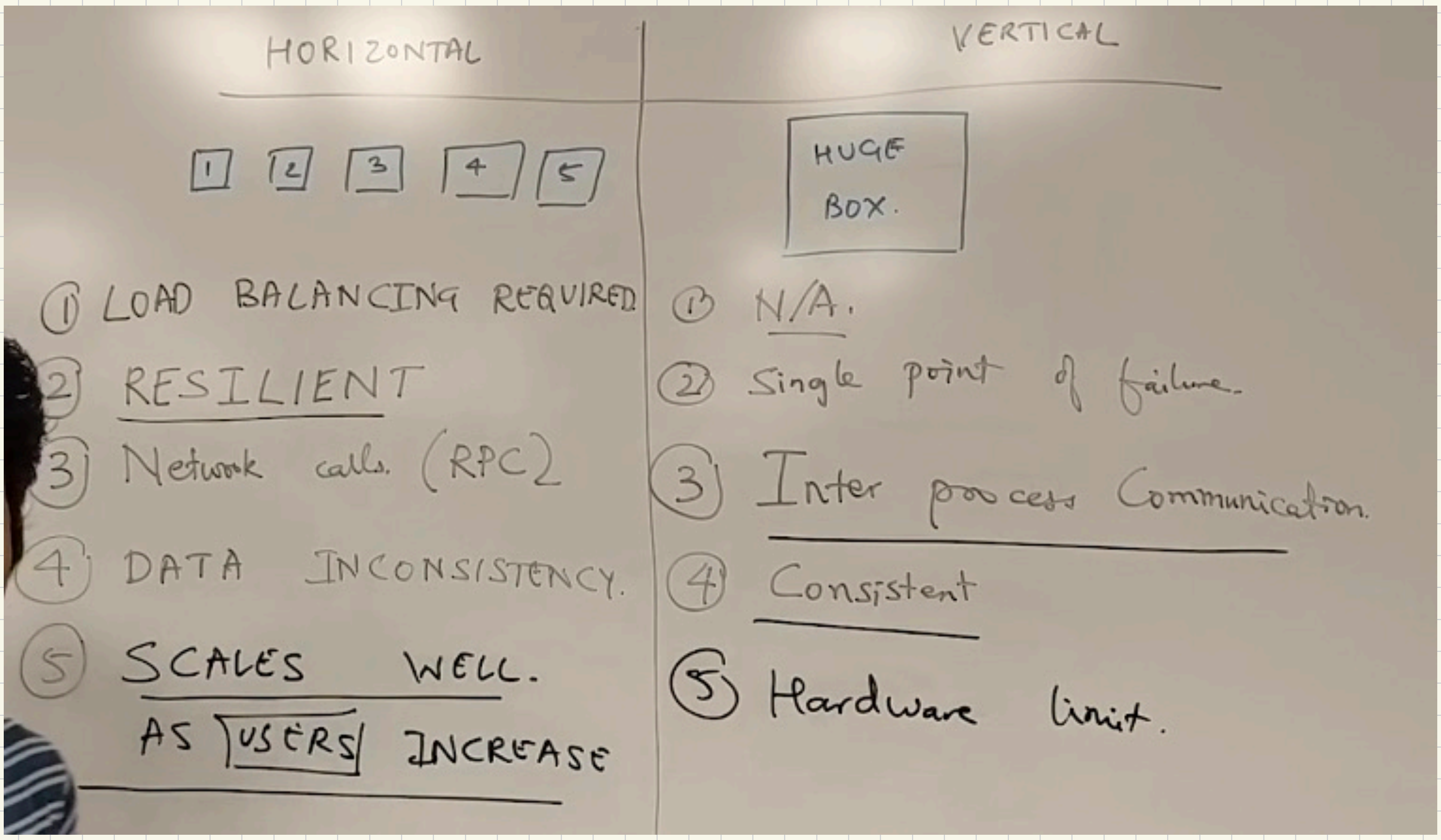
Systems design a procedure by which we define the architecture of a system to satisfy given requirements. It is a technique by which the required amounts of scalability, reliability, performance and consistency are satisfied in real world systems.

We discuss how to start with system design and why it is required. The first concept in designing a system is scalability. We discuss the two main approaches to solve this problem: Horizontal scaling and vertical scaling.

Horizontal scaling is adding more machines to deal with increasing requirements. These machines handle requests in parallel to improve user experience.

Vertical scaling is replacing the current machines with more advanced machines to improve throughput and hence response time. The techniques are used in conjunction in real world systems.



HORIZONTAL	VERTICAL
1 2 3 4 5	HUGE BOX.
1 LOAD BALANCING REQUIRED	1 N/A.
2 RESILIENT	2 Single point of failure.
3 Network calls. (RPC)	3 Inter process Communication.
4 DATA INCONSISTENCY.	4 Consistent
5 SCALES WELL. AS USERS INCREASE	5 Hardware limit.

Horizontal Scaling
Load Balancing Required: This means that a load balancer is needed to distribute workloads across multiple machines, servers, or systems to ensure no single system is overwhelmed.
Resilient: Systems that are horizontally scaled can be more fault-tolerant because if one machine fails, the others can take over the workload.
Network Calls (RPC): Remote Procedure Calls (RPC) are a method for a program to cause a procedure (subroutine) to execute in another address space (commonly on another computer on a shared network). This is a common necessity for horizontally scaled systems.
Data Inconsistency: Since data might be spread across multiple systems, keeping data consistent across all of them can be challenging.
Scales Well as Users Increase: Horizontal scaling can be very effective as the number of users or the workload increases, as you can add more machines or resources as needed.

Vertical Scaling
Huge Box: This term suggests that vertical scaling involves increasing the capacity of a single server or system ("box") by adding more resources such as CPU, memory, or storage.
N/A: Not applicable.
Single Point of Failure: Because everything is on one system, if that system fails, everything goes down with it.
Interprocess Communication: This refers to communication between different processes within the same system, which is more efficient than network calls in a horizontal setup.
Consistent: With all resources centralized, data remains consistent across the system.
Hardware Limit: There is a physical limit to how much you can scale up a single system. Eventually, you can't add more resources due to hardware limitations or it becomes cost-prohibitive.

why would be need RPC in horizontal scaling, I was thinking all the required process would be on same server and then we would eliminate RPC calls. Is my understanding right?

Your understanding is not entirely accurate in the context of horizontal scaling. Horizontal scaling, also known as scaling out, involves adding more nodes to a system, such as additional servers, to distribute the workload more broadly. In a horizontally scaled architecture, the processes are spread out across multiple servers or instances rather than being confined to a single server. Here's why RPC (Remote Procedure Calls) come into play in such a scenario:

Distributed Systems: In a horizontally scaled system, since you have multiple servers, each server may perform different tasks, or they may work together to perform the same tasks. Processes may need to communicate with each other across these different servers, which is where RPC is used.

Communication Between Services: Horizontal scaling is often employed in microservices architectures where different services run on different servers or clusters. To perform a task, a service may need to communicate with other services that don't reside on the same server. RPC allows these services to call functions on each other as if they were local.

Load Distribution: When using a load balancer to distribute incoming requests across multiple servers, the subsequent internal requests to different services or components may still need to be communicated across servers. RPCs facilitate this internal communication.

Scalability and Flexibility: With RPC, you can scale out each service independently. If one particular service requires more resources, you can add more servers to that service without affecting the rest of the system. RPC provides the necessary mechanism for these services to continue interacting with one another, regardless of where they are deployed.

So, in summary, while a single server might eliminate the need for RPC by running all processes locally, horizontal scaling inherently requires network communication mechanisms like RPC because it involves multiple servers that need to communicate with each other to function as a coherent system.

what if I am using monolith architecture then all process are coupled together, in that case we could eliminate RPC calls?

In a monolithic architecture, all processes are indeed coupled together into a single, indivisible unit that typically runs on a single server or instance. In such an architecture, because all components of the application are interconnected and run within the same process space, you can eliminate the need for RPC calls for internal communication. Here's how it works:

Internal Function Calls: Within a monolithic application, components and services communicate with each other through direct function or method calls rather than through network calls.

Shared Memory Access: The different parts of a monolithic application can often share data directly through in-memory data structures, without the need for the serialization and deserialization that RPC requires.

Simplified Deployment: The application is usually deployed as a single entity, which simplifies the deployment process but can lead to challenges in scaling and continuous deployment.

Scaling the Monolith: To scale a monolithic application, you typically perform vertical scaling (scaling up) by adding more resources (CPU, memory, disk I/O, etc.) to the existing server. If horizontal scaling (scaling out) is necessary, the entire application is duplicated across multiple servers, which can lead to redundant processing and does not benefit from the modular scalability that microservices provide.

However, it's important to note that while a monolithic architecture may not require RPC for internal communication, it can still use RPC or other network communication protocols to interact with external services, third-party APIs, or databases that are not part of the monolithic codebase.

As with any architectural decision, there are trade-offs. While monoliths can reduce the complexity of inter-process communication, they can also become unwieldy as they grow, and scaling can become more difficult. This is one reason why many organizations have moved towards microservices architectures, which allow for finer-grained scaling and deployment, despite the added complexity of communication between services using RPC or other communication protocols.