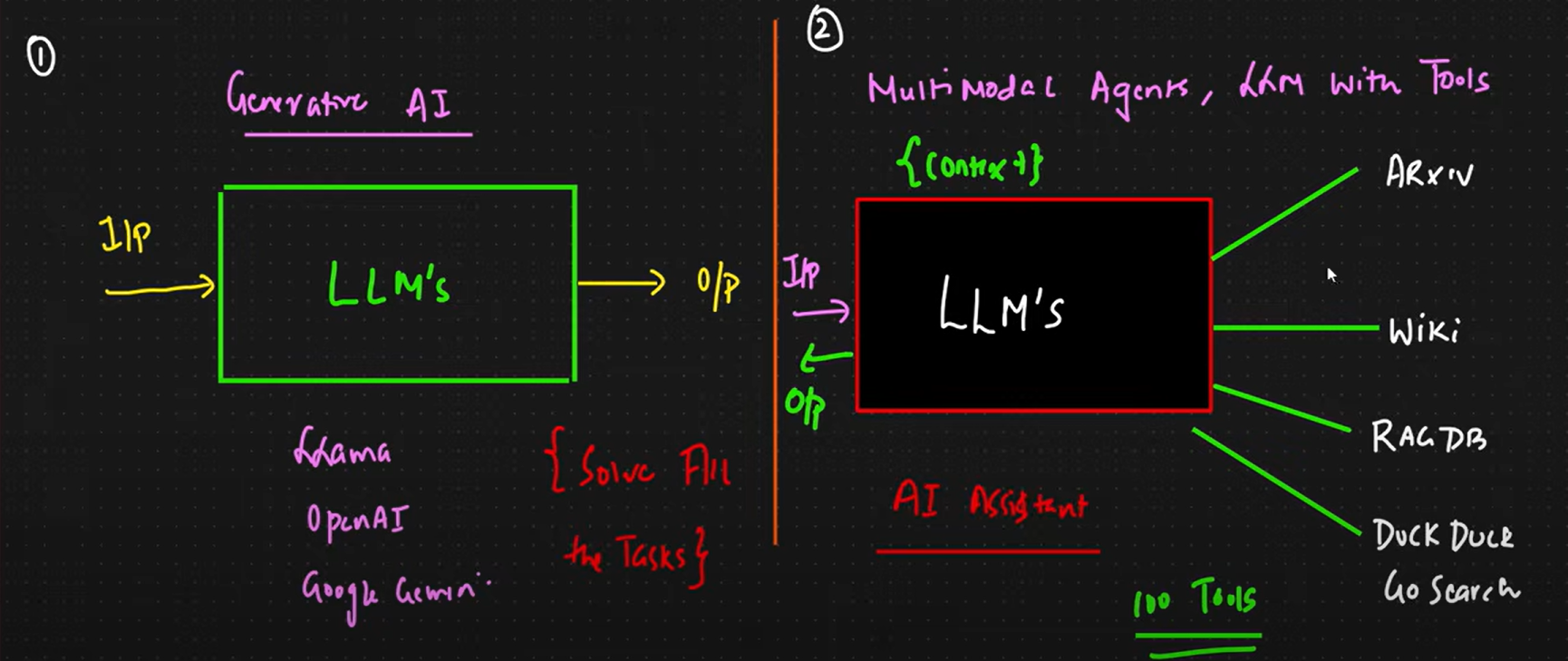
**MODEL CONTEXT PROTOCOL**

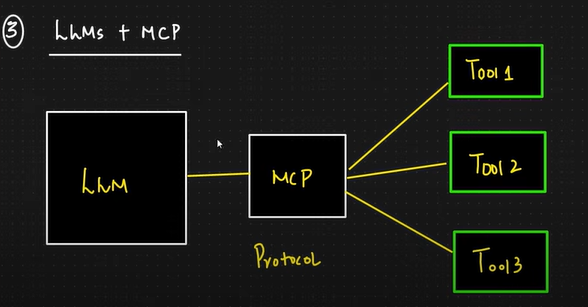
1. **What is Model Context Protocol ( MCP )**

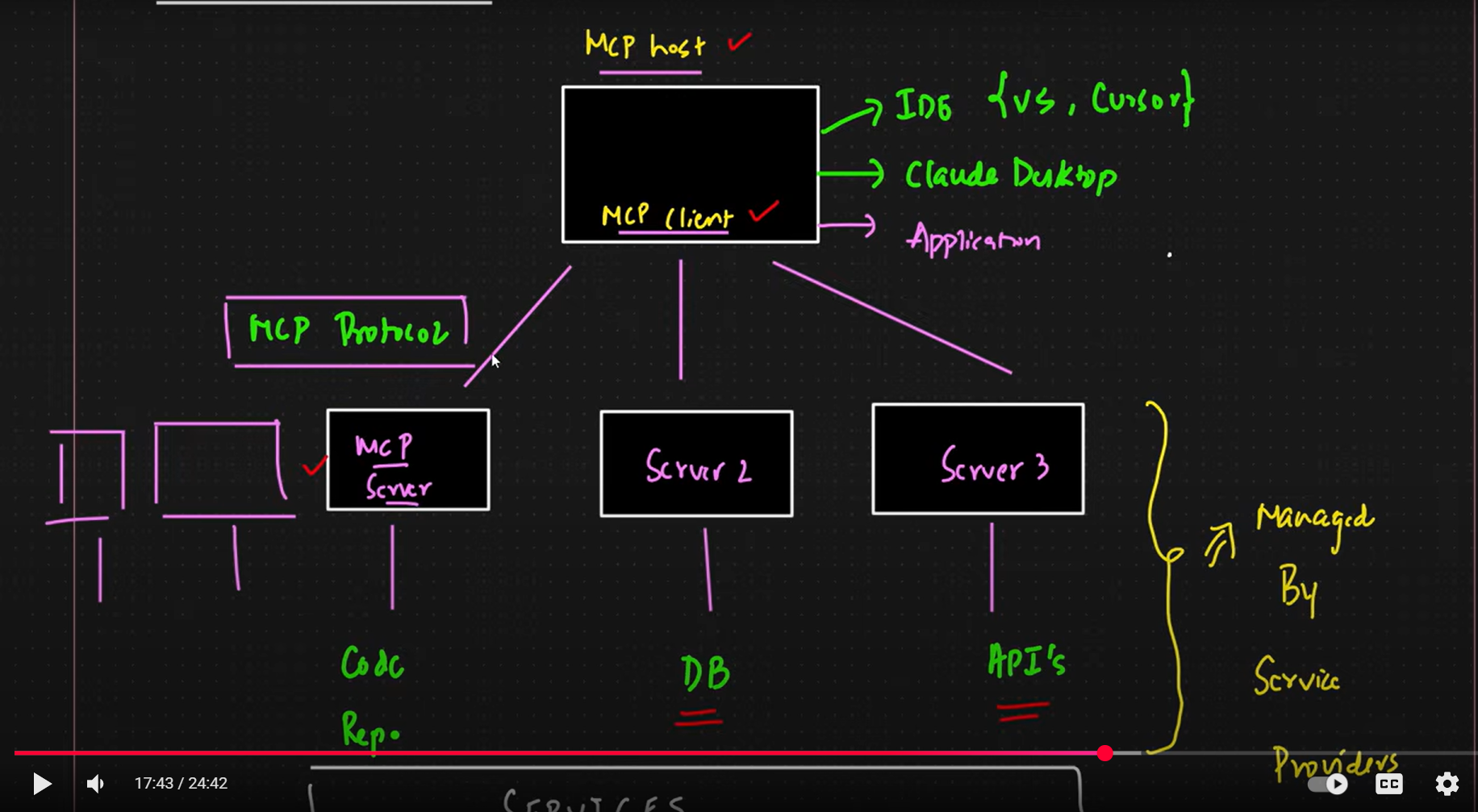
* MCP is an open-source standard that allows ai models to interact with external systems like db, files , API etc
* Think of it like a **USB-C port for AI**: it standardizes how AI applications connect to the outside world.

1. **Why is it needed?**

* Currently lets say there are a lot of tools your llm is interacting with, what you need to do is write different set of codes for different tool ( Ex – a different code for serper tool ( google search) , a different code for wiki search tool.
* Generally its not a problem but lets say you have got n number of tools, or lets say there is a change in any of the tools ( configuration wise or syntax wise) . You need to update your code again and again.



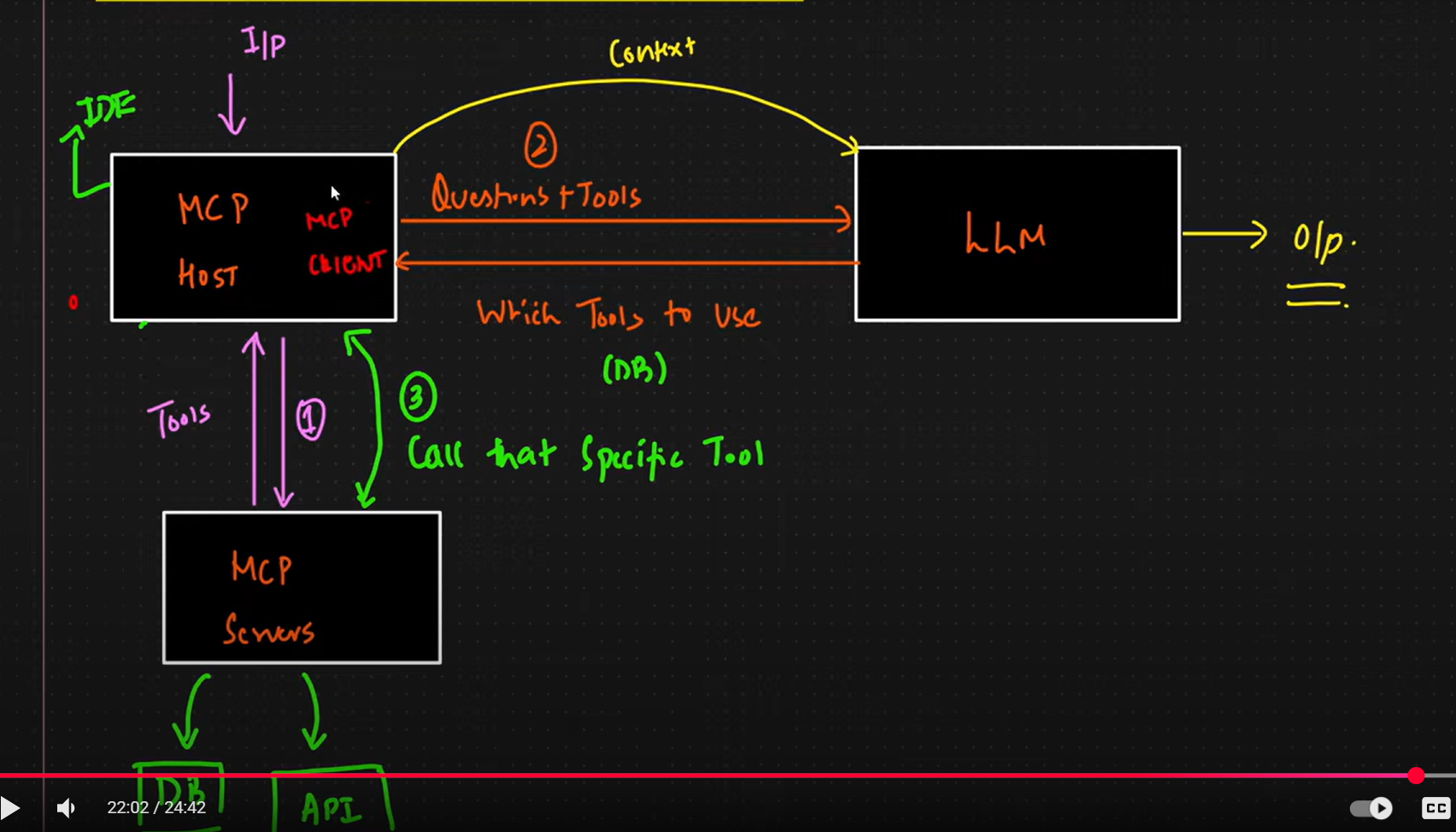
* Therefore, to fix this we use MCP
* Ok so now lets study what does it do
* Jaise ki hamne dekha ki jaise jaise app scale hoga dikkat to aegi hi
* So kya hota hai, Tools aur llm ke beech ek MCP ki layer a jati hai
* Aur sabhi tools ko wo standard ya protocol follow karna padta hai
* **Important Components**



1. **MCP Host** - The AI model or app making requests (e.g., Claude, ChatGPT).
2. **MCP Client** – This will be created by the MCP host to interact with the **MCP server**
3. **MCP Server –** MCP server will be connected to different services as required by the app. It is the backend that will perform the task
4. **MCP protocol-** The communication standard between host and server (like HTTP for AI).

* **Benefits of this Architecture**

1. One of major benefits is that the server and the services part is purely managed by the service provider
2. Because of this the user need not to care about the changes in the service he just need to create a client and interact with the server ( No change needed unless the server introduces breaking changes or new capabilities that the client wants to use.
3. **How is the communication like in this architecture?**



* Ek client sirf ek service ya tool se hi baat karta hai ( For multiple tools we create different clients)

Here is the flow

* MCP client input lega
* Client mcp servers ke paas jake tools jo bhi available hai le aega
* Ab LLM ko bhejenge tools + question
* LLM analyse karke bataega ki kaunsa tool chaiye kaunsa ni
* LLM kahega client se ki mujhe ye tools chahiye
* Client fir se MCP servers pe jake wale tools ko call Karega
* Ab LLM ke pass Context hai from the tools, question hai ab wo answer kar dega
* Jaise if I ask ki abhi particular location mein kya temp hai ( wo api call kaega weather wali with meri location usko jo data mile wo context ho gaya ab wo ouput kar dega)

1. **User Input:**The user gives a question or task (e.g., "What's the temperature in Bangalore?").
2. **Client Initialization:**The MCP client receives the input and connects to the MCP server to discover available tools (like weather API, calendar, file reader, etc.).
3. **Tool Discovery:**The client fetches the list of available tools and sends them, along with the user's question, to the LLM (Large Language Model).
4. **LLM Tool Selection:**  
   The LLM analyzes the question and the available tools, then decides which tool(s) are needed to answer the question.
5. **Tool Invocation:**  
   The LLM instructs the client to call the selected tool(s) via the MCP server.
6. **Tool Execution:**  
   The MCP server executes the tool (e.g., calls the weather API with the user's location) and returns the result to the client.
7. **Context Formation:**  
   The client sends the tool output + original question back to the LLM.
8. **Final Answer:**  
   The LLM now has both the context (tool output) and the question, and generates the final answer (e.g., "The temperature in Bangalore is 28°C").
9. **MCP Primitives**

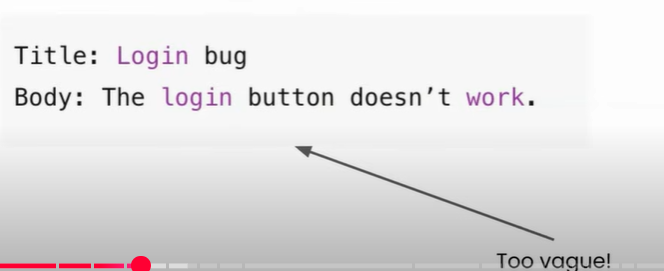
* Things which a server provides to the host. Normally three things it provides:

1. **Tools** – Actions the AI ask the server to perform
2. **Resources** – Structured data sources that the AI can read ( fixed chezein like readme on the github )
3. **Prompts –** Predefined prompt templates or instructions that the server offers to help shape the AI’s behaviour

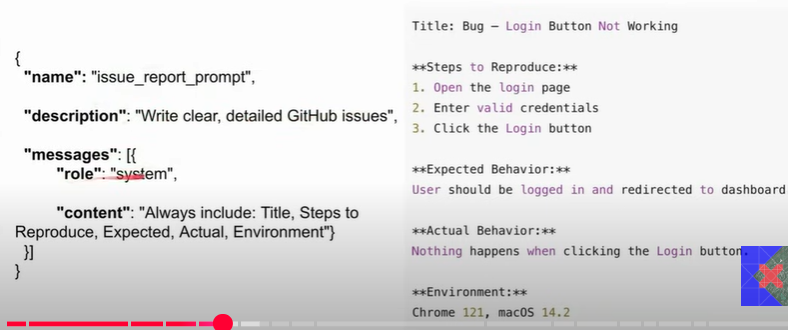
Thora samjhte hai prompt wale point ko

* + Manlo tumhe koi issue create karwana hai login button ni kaam kar rha karke
  + To bahut chances hai LLM tools ki madad se kar to dega tumhara kaam lekin kaafi generalize statement likh sakta, ( Issues to detailed hote hai na )
  + To ham server pe predefined code ya template likh deta hai jaise is example mein name: ‘create-issue’ karke to llm dekhega ki acha ye template follow karna hai to wo follow kar lega

Without prompt primitive



With prompt primitive



Some standard operations the Primitives provide

**1. Tools**

* Represent functions or actions that interact with external systems.
* **Operations:**
  + tools/list: List all available tools.
  + tools/call: Invoke a specific tool with parameters.
  + Example: Call a weather API, query a database, send an email.

**2. Resources**

* Refer to structured data like files, schemas, or application-specific info.
* **Operations:**
  + resources/list: List available resources.
  + resources/get: Retrieve the content of a resource.
  + Example: Access a document, read a config file, fetch a dataset.

**3. Prompts**

* Reusable instructions or message templates that can be invoked with parameters.
* **Operations:**
  + prompts/list: List available prompts.
  + prompts/get: Retrieve a specific prompt.
  + prompts/invoke: Execute a prompt with parameters.
  + Example: Generate a summary, echo a message, format a response.

1. **MCP Data Layer**

**Data Layer-** The data layer is the language and grammar of the MCP ecosystem that everyone agrees upon to communicate. In MCP , **JSON RPC 2.0** serves as the foundation of the data layer

**What is RPC?**

Remote Procedure Call (RPC) is a protocol that allows a program to execute a procedure (function or method) on a remote system as if it were a local call, abstracting away the details of the network communication.

**JSON RPC 2.0 ( JavaScript Object Notation – Remote Procedure Call )**

**JSON-RPC** combines the concept of RPC and simplicity of JSON, allowing developers to structure RPC requests and responses in a standardized JSON format.

**Ex-** Lets say I need to call the function add(2,3) on a different server/computer

**1. Request Structure**

{

“jsonrpc”: “2.0”,

“method”:”add”,

“params”:[2,3],

“id”:1

}

**2. Response structure**

{

“jsonrpc”: “2.0”,

“result”: “5”,

“id”:1

}

Lets say method doesn’t exist

{

“jsonrpc”: “2.0”,

“error”: {“code”:-32103 , “message”: “Method not found:”},

“id”:1

}

**3. Notification structure**

One-way messages that don’t require a response. Typically sent from Server to Client to provide updates or notifications about events.

**Example Notification:**

**{**

"jsonrpc": "2.0",

"method": "progress",

"params": {

"message": "Processing data...",

"percent": 50

**}**

**}**

**Why JSON RPC for Data Layer?**

* It’s lightweight
* Supports bi-directional communication
* Its transport – agnostic
* Its supports bathcing ( multiple request at the same time)
* Supports notifications

**Discussion on why it is transport-agnostic**

Transport-agnostic means the protocol (like JSON-RPC or MCP) does not depend on a specific way of sending data. It defines how messages should look, but not how they should be delivered.

So, whether you're using:

* HTTP
* WebSockets
* Stdio (standard input/output)
* Server-Sent Events (SSE)
* Named pipes
* or even custom transports

Matlab yahi hai bhaiya main simply apna request bana dunga ba tum use jise marzi chaho bhejo

**🔄 You can use it over:**

* **HTTP POST** (most common)
* **WebSockets** (for real-time apps)
* **Stdio** (for CLI tools or local agents)
* **Message queues** (like RabbitMQ or Kafka)

1. **MCP Transport Layer**

* The transport layer is the mechanism that moves JSON-RPC messages between the Client and Server.
* The choice of transport depends on the type of server

**Two types of server**

* 1. Local Server – Host and the server are on the same machine
  2. Remote Server – Host and the server are on the different machine

**Two modes of transport**

* If the server is local then the mode of transport used is **STDIO ( Standard input and output )**
* If the server is remote then the mode of transport used is **HTTP+SSE**

1. **STDIO**

It refers to the built-in streams every program has.

**Stdin** ( input the program reads)

**Stdout** ( output the program writes)

In MCP, these streams are used as transport layer between the client and the server

Process:

Host server ka sub process banata hai in the same system ( parent child relationship established ho gayi)

Stdin se server ko input jata hai

Stdout se server wapis response bhejta hai

**Benefits:**

* The data passed is very fast
* No open network port hence secured
* Its simple

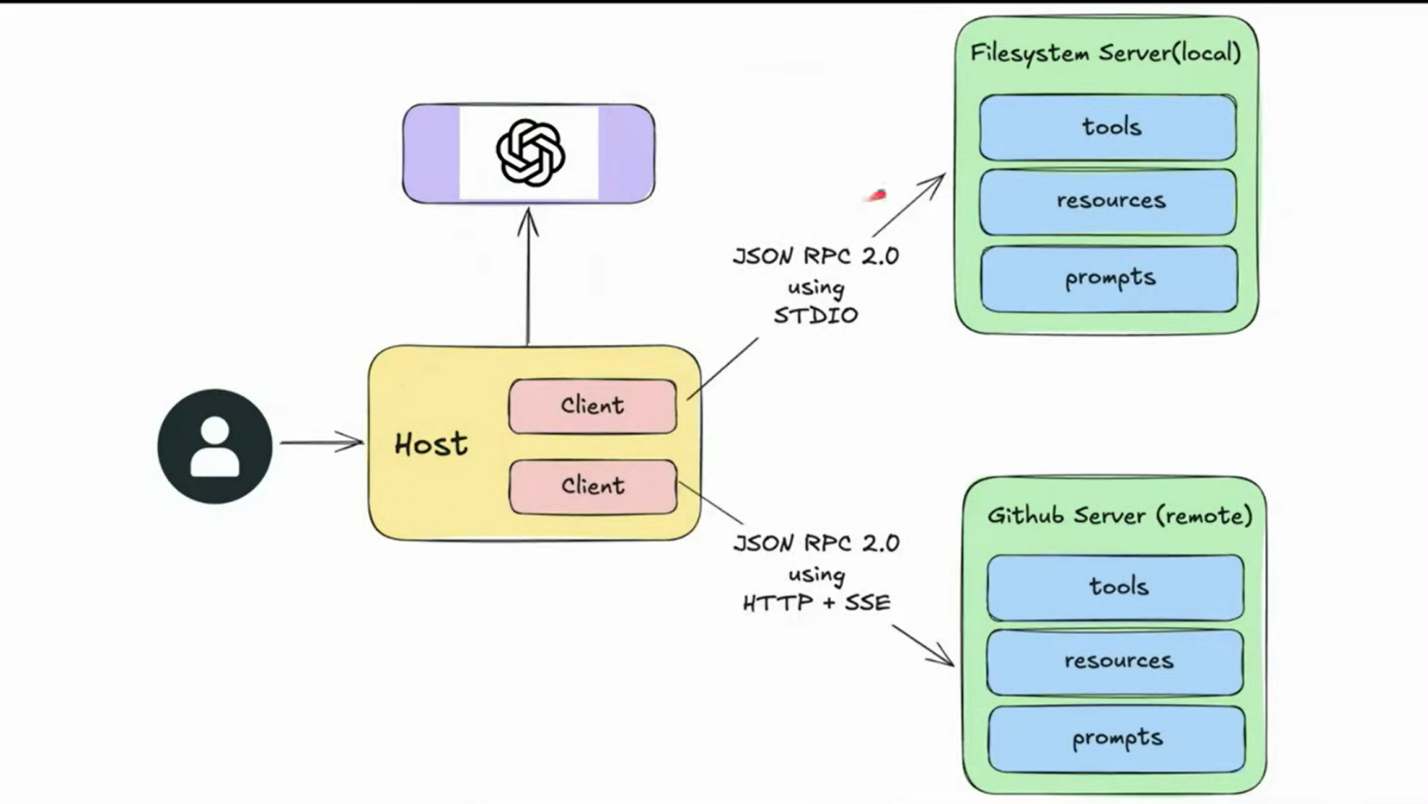
1. **HTTP+SSE**

**HTTP**

* Using HTTP allows the host to reach Servers running anywhere
* Hosts sends JSON RPC requests using POST requests with a JSON payload
* This transport system supports standard HTTP auth methods.

**SSE**

* Server sent events and it’s an extension of HTTP
* Using SSE the server sends multiple messages to client over a single open connection
* Instead of sending one large JSON blob, the server can stream chunks of data as they are ready
* Ideal for long running tasks or incremental updates.



**MCP Architecture**

1. **MCP Lifecycle**

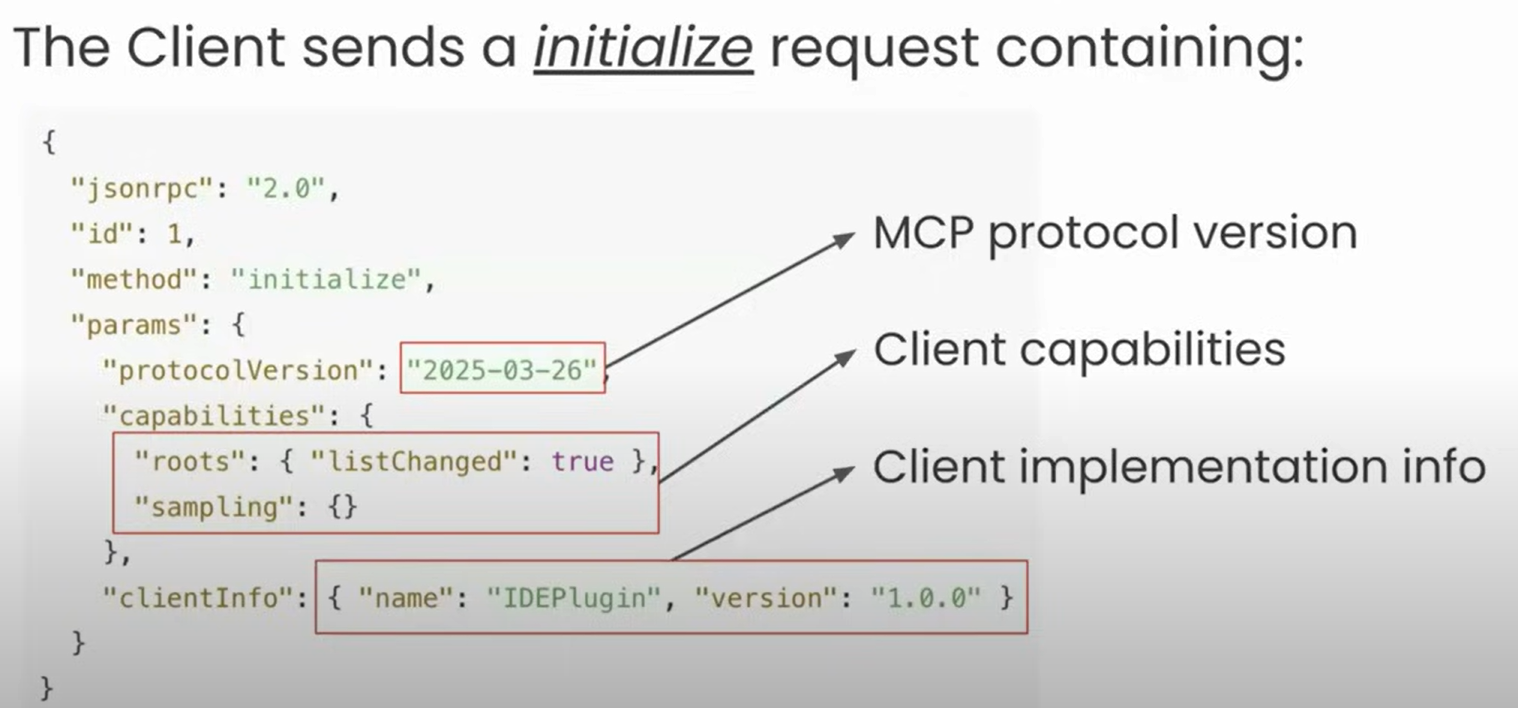
* The MCP life cycle describes the complete sequence of steps that govern how a Host ( client ) and a Server establish, use and end a connection during a session
* **Session** – One continuous connection between the client and the server

**Stages**

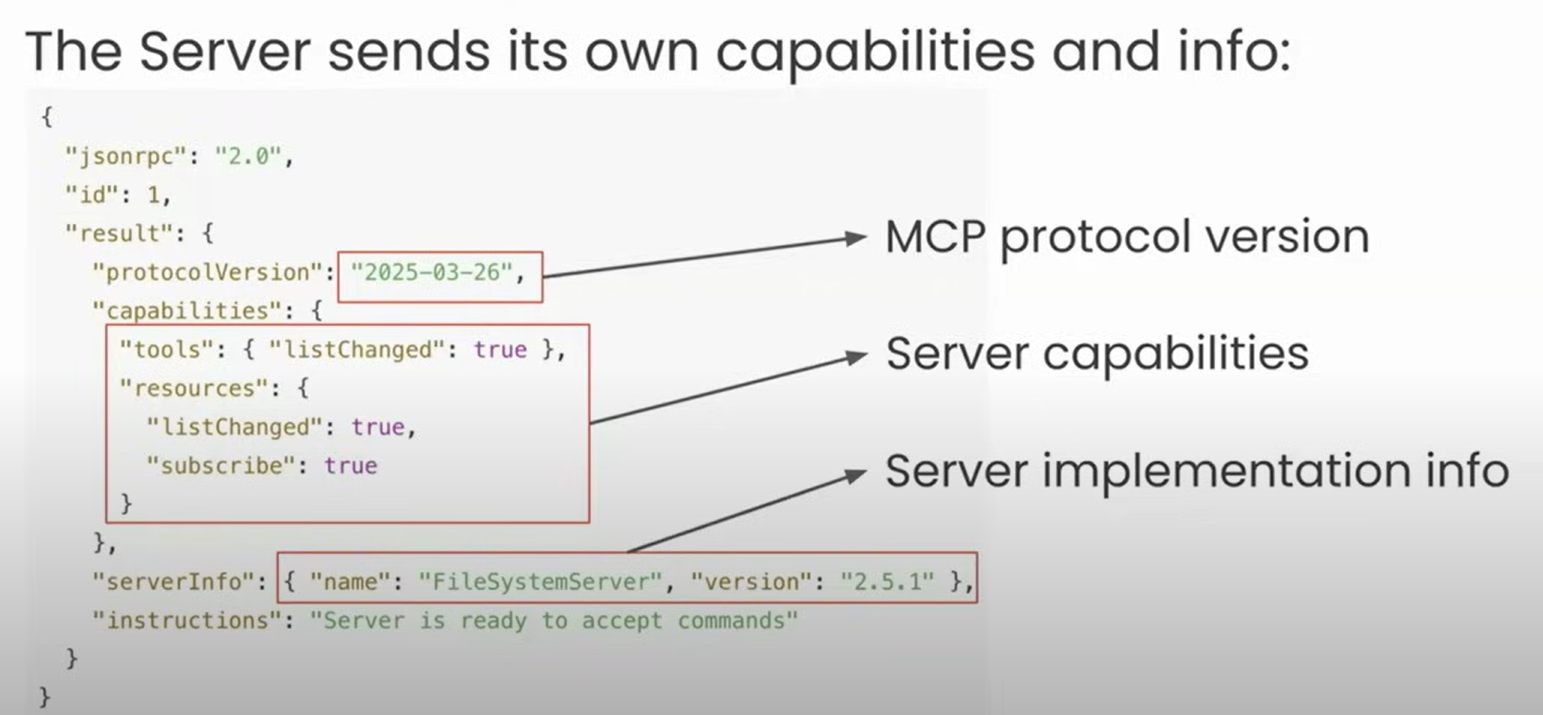
1. **Initialization**
   * The initialization phase **must** be the first interaction between the client and server.
   * Establish protocol version compatibility
   * Exchange and negotiate capabilities ( Basically is baat pe discussion ki tum kya kar sakte aur main kya kar sakta – Handshake kind of thing )

**Steps of this phase**

* + Step 1: The client sends a ***initialize*** request to the server



* + Step 2: The server sends its capabilities and info:



* + Step 3: After successful initialization, the client **MUST** send an initialized notification to indicate it is ready to begin normal operations

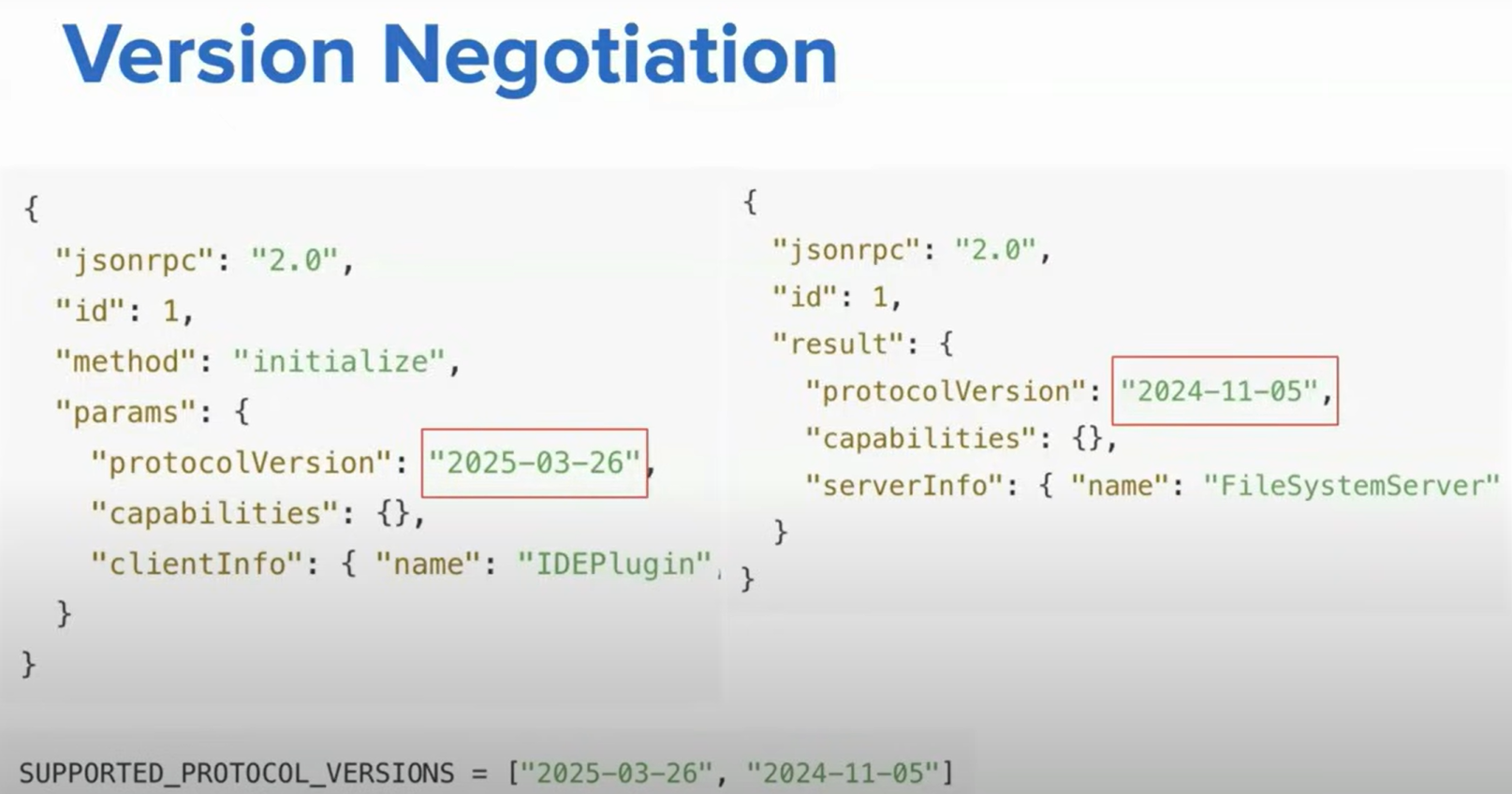


**Important Rules**

* + The client should not send requests other than pings before the server has responded to the initialize request ( Matlab sirf initialize wali hi ping jaegi , koi tool karna wagerah ni hoga yahan pe)
  + The server should not send requests other than pings and logging before receiving the initialized request

**Important Negotiations**

* + 1. **Version Negotiation –** Jab client aur server ke protocol versions match na karein to client apne config mein jake supported versions kaunse hai , agar unmein se koi server se match kar gaya to thik warna wahin connection closed

****

* + 1. **Capability negotiations –** Client and server capabilities establish which protocol features will be available during the session ( Apni apni capabilities share karga)

**Major capabilities of client**

* + **Roots –** root ka access dedeta hai server ko
  + **Sampling –** server can ask for help from client ( basically server request karta hai )
  + **Elicitation –** server kuch incomplete info demand kar sakta hai ( jaise github se connect karne ko kahan client ne lekin api key ni di, to server client ko bolega bhai api key de pehle

**Major capabilities of server**

* + **Prompts**
  + **Resources**
  + **Tools**
  + **Logging –** ye to simple hai

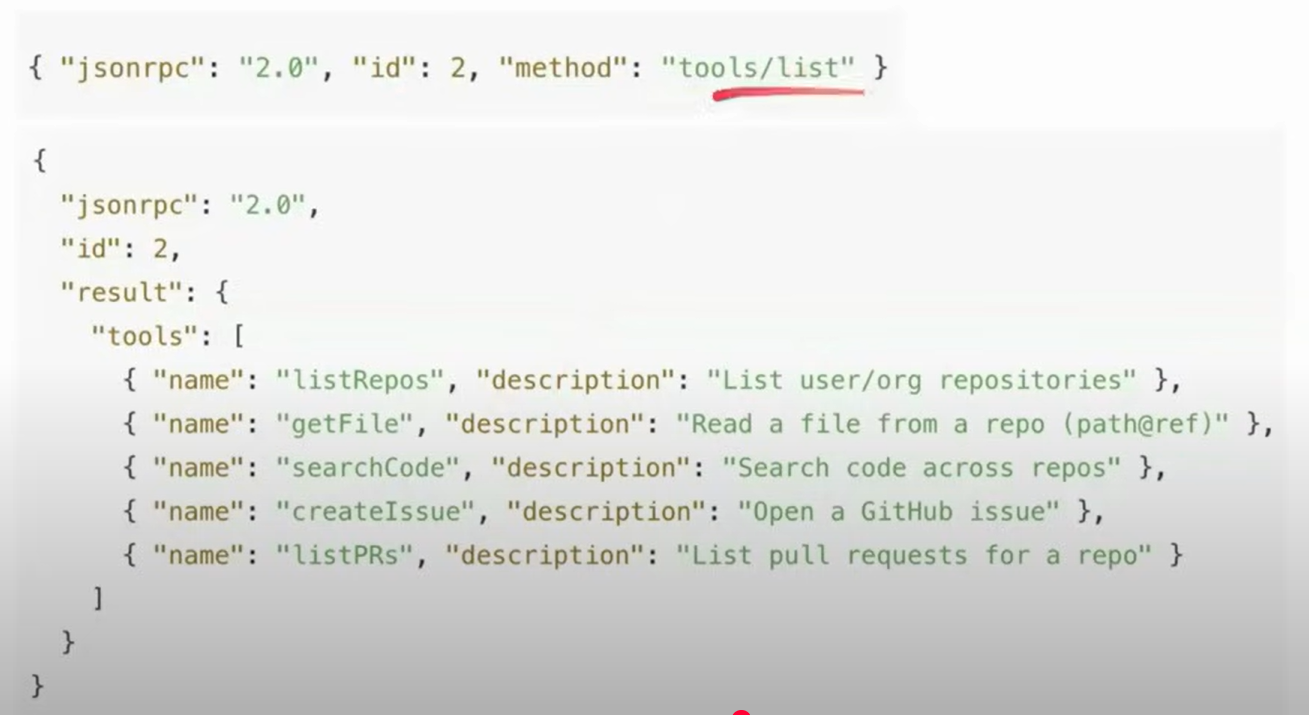
**Sub- capabilities of server**

* + **listChanged :** man lo during a session kuch change ho jae in tools, resources ( add ya subtract ) to server notification marega client ko dekh bhai ye tool a gaya ya ye tool chalaa gya
  + **subscribe:** jab kisi resources mein kuch change ho ( jaise github readme wala data mein kuch add kardun to server fir bataega client ko dekh bhai kuch change hua hai)

1. **Operation Phase**
   * During the operation phase, the client and server exachange messages according to the negotiated capabilities
   * Respect the negotitated protocol version
   * Only use the capabilities that were successfully negotiated

**Two steps**

1. **Capability Discovery**
   * Ismein tum exactly ye puchte ho server se kya kya tools hai tumhare pass mujhe de do
   * **Note : Ye call Initialization phase complete hone ke baad turant ho jata hai**
   * Client request marega /tools, /resources,/prompts ki server se aur server wo sab provide karega jo uske pass hoga

****

1. **Tools Calling**
   * Ab tool/resources ko call karte ho
2. **Shutdown Phase**
   * One side ( typically the client) initiates shutdown
   * No special JSON-RPC shutdown message is shared
   * Transport layer is responsible for signaling termination

**Shutdown in STDIO**

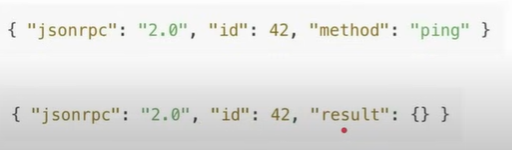
1. Client-initiated shutdown ( SHOULD) :
   * + Close input stream to the child process ( server) ( Aisa kaise kr pa rha kyunki client hi subprocess se server ka instance banata hai)
     + Wait for the server to exit
     + Send SIGTERM if server does not exit on time ( OS ko bolta hai server se kaho shutdown karde)
     + Send SIGKILL if still unresponsive ( Yahan OS ko force karta hai ki server ko band kardo)
2. Server-initiated shutdown ( MAY ):
   * + Close output stream to the client
     + Exit process

**Shutdown in HTTP**

1. Client-initiated shutdown ( Common case ):
   * + The client( Host) close the HTTP connection(s) it opened on the server
2. Server-initiated shutdown ( Possible):
   * + The server may close the connection
     + The client must be prepared to detect such drop in connection and gracefully handle it ( ex : reconnect if needed)

**Special Cases in MCP Lifecycle**

1. **Ping**
   * Ping is a lightweight request/response method defined in MCP.
   * **Purpose:** to check whether the other side ( Host or server ) is still alive and the connection is responsive.
   * Can be from both ways ( the client(host) or the server)

****

**When is ping used?**

* + Useful for checking if the other side is up before full **initialize**
  + If there is long running task between the client and the server, there might be cases where there is no activity performed for a while, so the system might think to close the connection, Therefor to prevent this , a client may send periodic pings so that the connection remains active
  + Prevents the connection from beign dropped silently by the OS, proxies , or firewalls

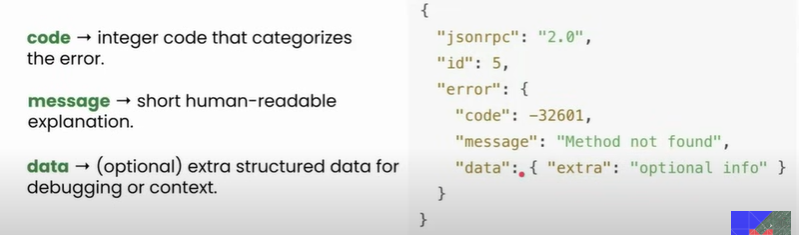
1. **Error handling**
   * Error handling in MCP is how the host ( client ) and Server signal that something went wrong with a request
   * **MCP** inherits JSON RPC’s standard error object format

**Some causes of Error**

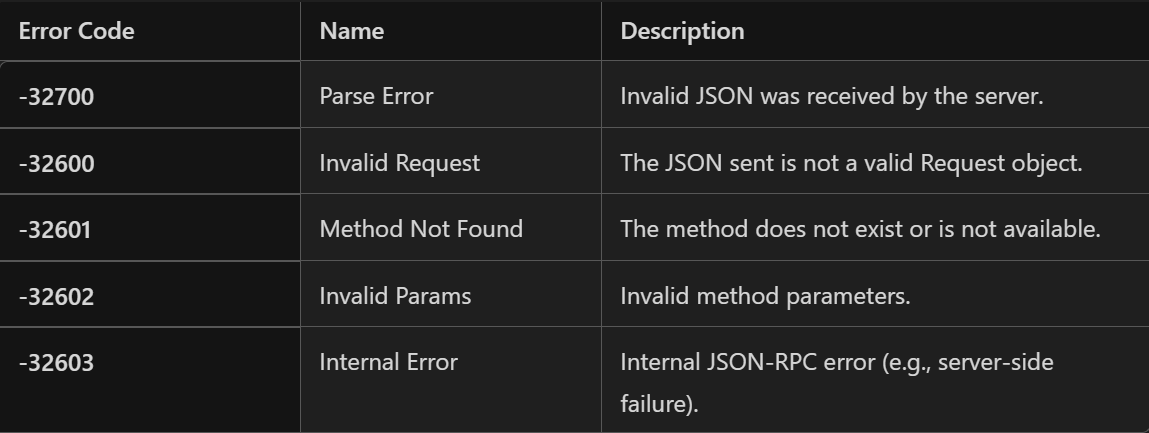
* + Unsupported or mismatched protocol version
  + Calling a method which was not negotiated
  + Invalid argument in a tool
  + Internal server failure while processing a request
  + **Timeout exceeded** – (Client kabhi kabhi timeout lga deta hai ki itna hi wait karunga fir close kar dunga request) client cancels request
  + Malformed JSON-RPC message

**Error Object Structure**

* + Code – integer code that represents the error
  + Message – short human-readable explanation
  + Data – (optional) extra structured data for debugging or context



**Common Error Codes**

****

1. **Timeout**

Timeout is about ensuring requests don’t hang forever

**Purpose:**

* + Protects against unresponsive or overloaded servers
  + Ensures resources ( memory,CPU) aren’t held indefinitely.
  + Gives the user feedback instead of waiting forever

**Working:**

* + SDKs let client set a per-request timeout ( Ex – 30s)
  + If the deadline passes with no result – client triggers a timeout
  + Client then sends a **cancellation** notification to tell the server to stop
  + The server must stop processing that request and not return a result

1. **Progress Notification**
   * Long running task mein periodically info dete raho ki ab kya ho rha hai
   * The purpose of this is to let the user know that the long running task is still executing
   * Client includes a progress token in the request \_meta
   * Server can then send notifications/progress updates while working

****