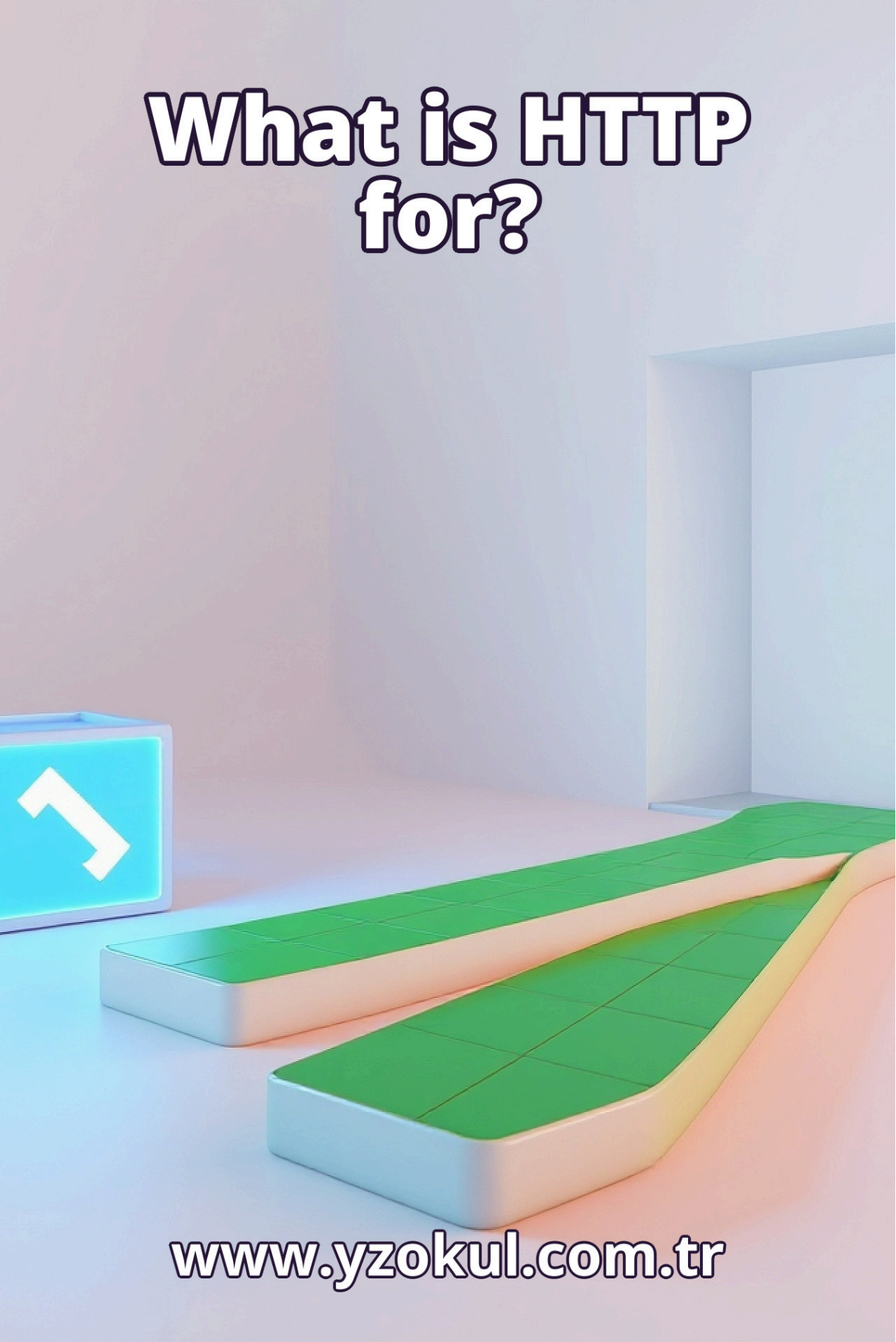
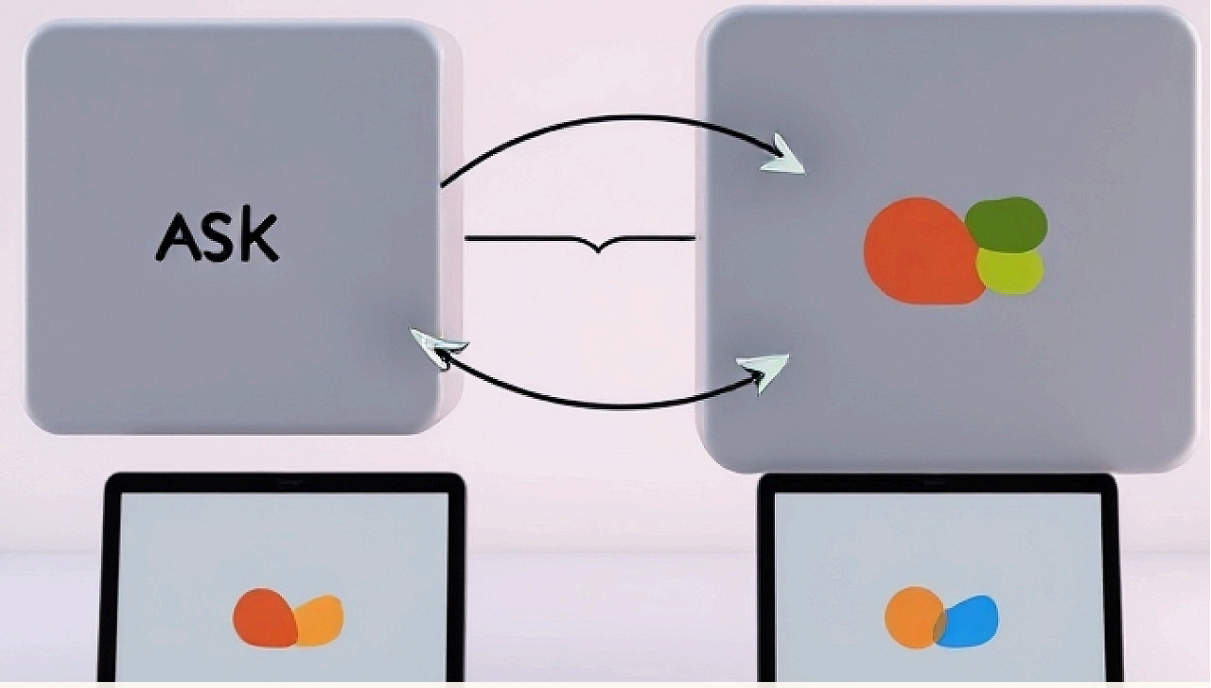
What is HTTP for?



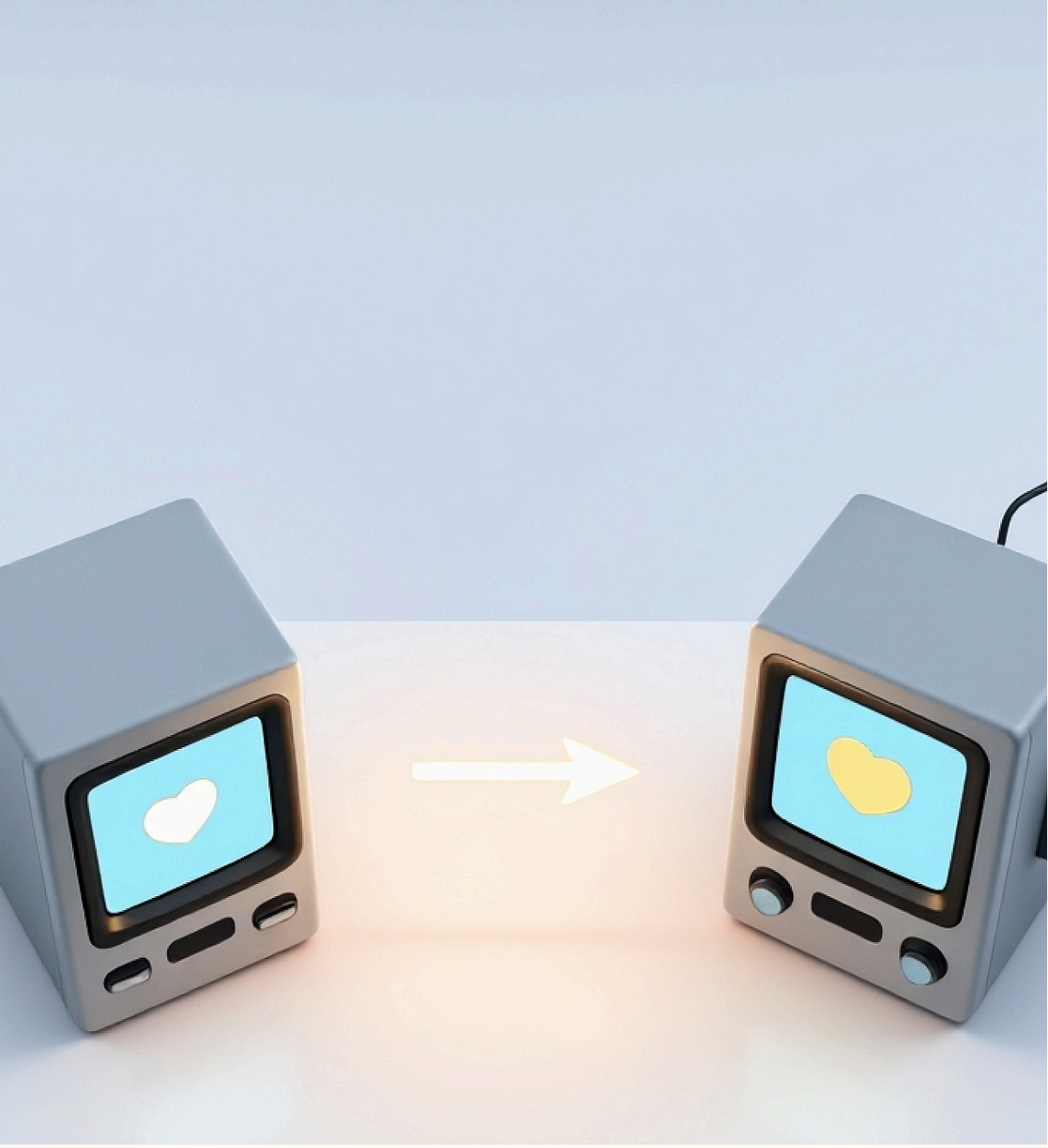
www.yzokul.com.tr

What is HTTP for?

www.yzokul.com.tr



HTTP is a protocol for exchanging requests and responses between a client and a server. That one sentence is the whole book in miniature: a client asks, a server answers, and HTTP sets the rules for that exchange. This book explains each part step by step so you can see what HTTP is for. By the end, you will understand how requests, responses, and status codes work together.



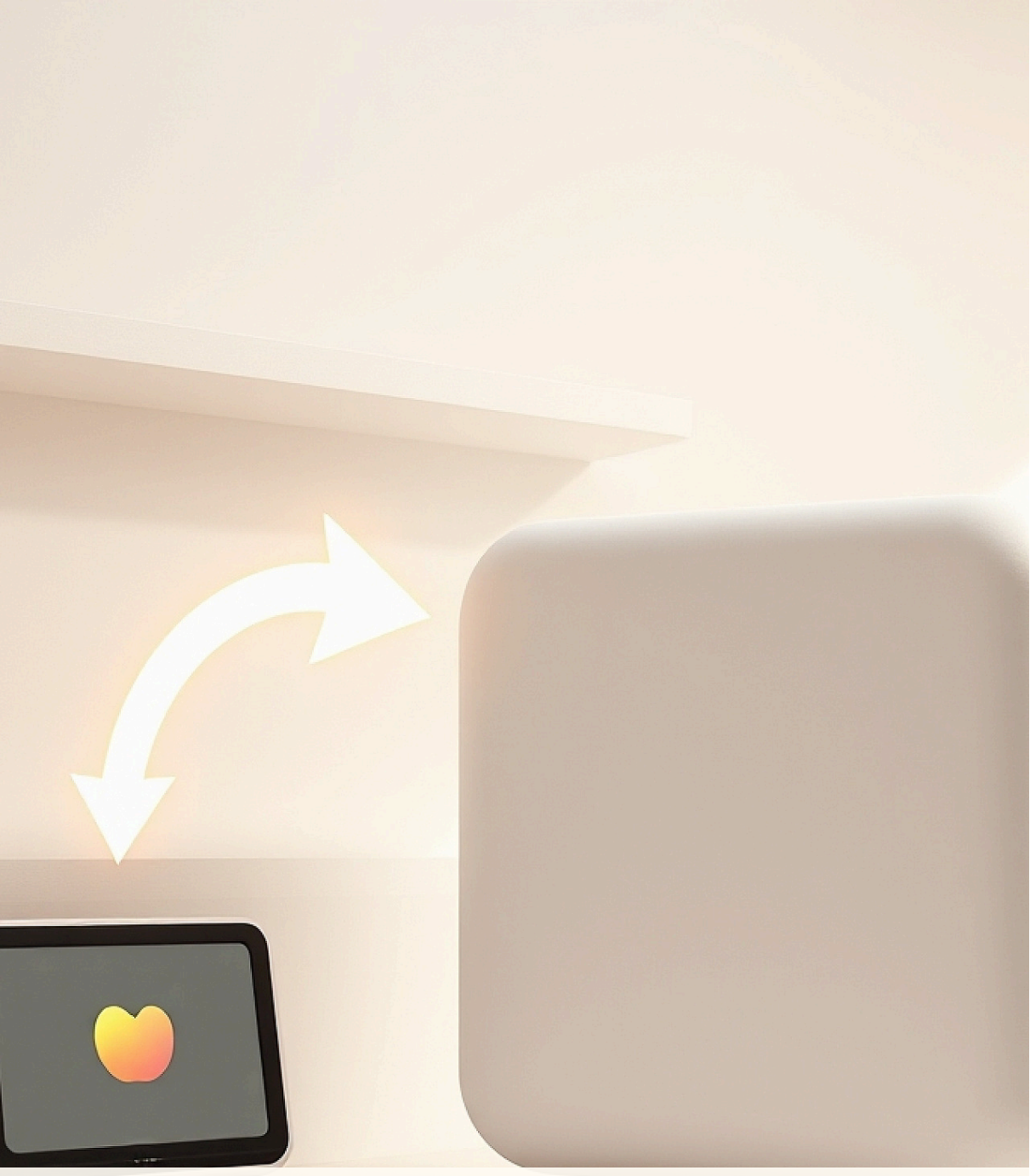
Picture two computers: one asks a question, and the other answers. This simple back-and-forth is the heart of every HTTP exchange. The asking computer sends a message, and the answering computer replies. HTTP makes sure both sides send and read messages in a way they both understand. This pattern repeats every time you load a page or an app fetches data.



A protocol is just an agreed set of rules for talking. When two computers use HTTP, they both follow the same rules so each understands the other. For example, one rule says how to format a request, and another says how to format a response. Without shared rules, the computers would not know what the other's messages mean. HTTP is that shared rulebook.



The client is the side that starts things by sending a request. The browser on your computer is a common client. When you type a web address, the browser builds an HTTP request and sends it out. The request asks for something specific, such as a page or an image. The client does not wait for the server to speak first; it begins the exchange.



The server is the side that waits, receives the request, and sends back a response. It holds the content the client wants, such as web pages, images, or data. When a request arrives, the server reads it and decides what to return. Then it sends a response back to the client. The server does not start the conversation; it answers when asked.



A client requests a specific resource, such as a web page or an image. The request names what is wanted, so the server knows exactly what to look for. For example, the request might ask for the page at a certain address. The resource could be a file, a picture, or data from an application. Naming the resource is how the client tells the server what to send back.



The server sends back the requested content in its response. This is how a page or file actually reaches the browser. The response carries the content itself, such as the text and images of a web page. When you see a page appear on your screen, the server has just returned that content. The client receives it and displays it for you.



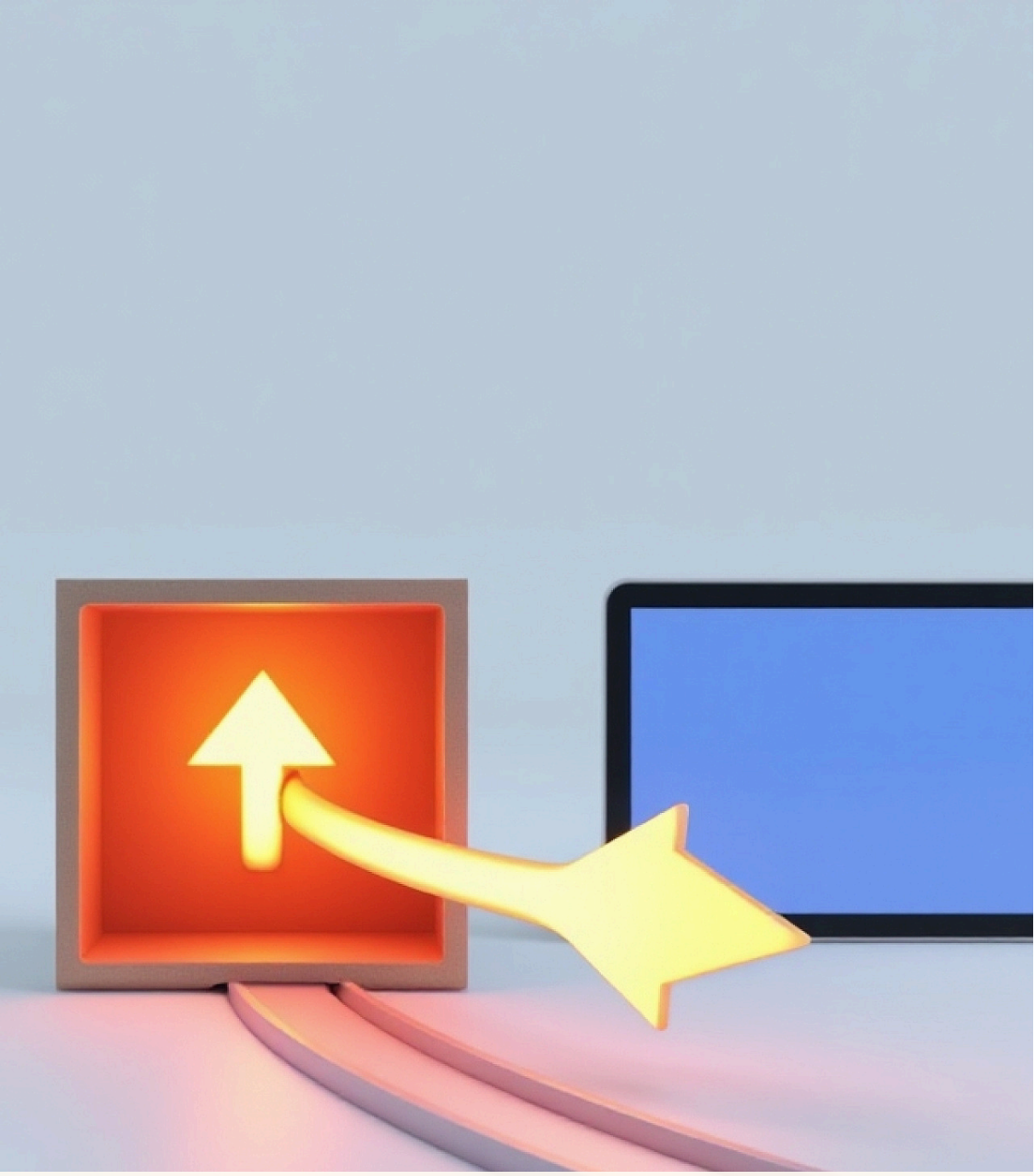
Every response carries a status code saying how it went. The code is a short number that summarizes the outcome. For example, a code might say the request succeeded or that the resource was not found. The client reads the code to know what happened before looking at the content. Status codes are a quick way to report the result of a request.



When the page is found, the server replies with a success status code. The content comes along with it. For example, the server might return the number 200 to mean success, and then send the page itself. The client sees the success code and knows the content is valid. This is the normal case when everything works as expected.



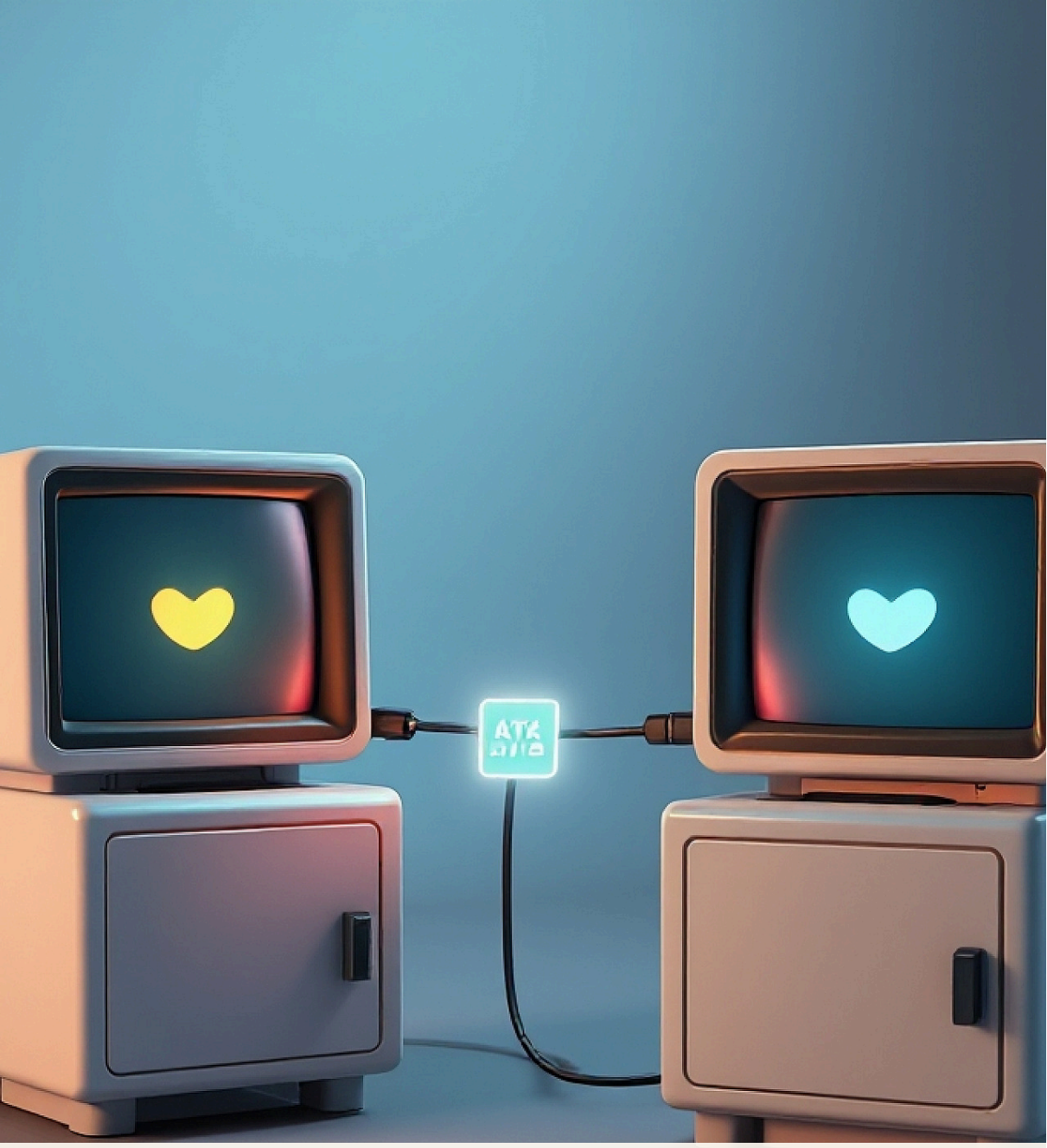
When the page is missing, the server replies with a not-found status code. The code tells the client what happened. For example, the server might return the number 404 to mean not found. The client sees this code and knows the requested resource does not exist. The server may still send a short message, but the code is the key signal.



Each exchange is one request followed by one response. This pairing is the basic unit of HTTP. The client sends exactly one request, and the server sends back exactly one response. Then the exchange is complete. If the client needs more, it sends another request. This simple pattern repeats for every resource you load.



HTTP is not only for displaying pages. It is also commonly used to transfer data between applications. For example, a weather app on your phone might use HTTP to fetch the forecast from a server. The same request-and-response pattern applies, but the content is data rather than a web page. HTTP moves the data just as it moves pages.



Picture two applications, not a person and a browser, exchanging data over HTTP. One application acts as the client and sends a request; the other acts as the server and sends a response. For example, a shopping app might request product information from a store's server. The same rules apply: request, response, and status codes. HTTP works the same way for apps as for browsers.



A common mistake is thinking HTTP is a website or a page. HTTP is the set of rules used to move requests and responses, not the content itself. The website is the content; HTTP is the protocol that carries it. You can use HTTP to move any kind of content, not just web pages. Keep the rules and the content separate in your mind.



Another common mistake is thinking HTTP only works with browsers. Any client and server that follow the rules can use it, including apps sharing data. A mobile app, a desktop program, or a server talking to another server can all use HTTP. The browser is just one common client. The protocol does not care who the client is, as long as it follows the rules.



HTTP organizes requests and responses between a client and a server. The client asks for a resource, and the server returns content with a status code that tells the result. HTTP also carries data between applications, not just web pages. Remember these key points: request, response, status code, and data transfer. That is what HTTP is for.