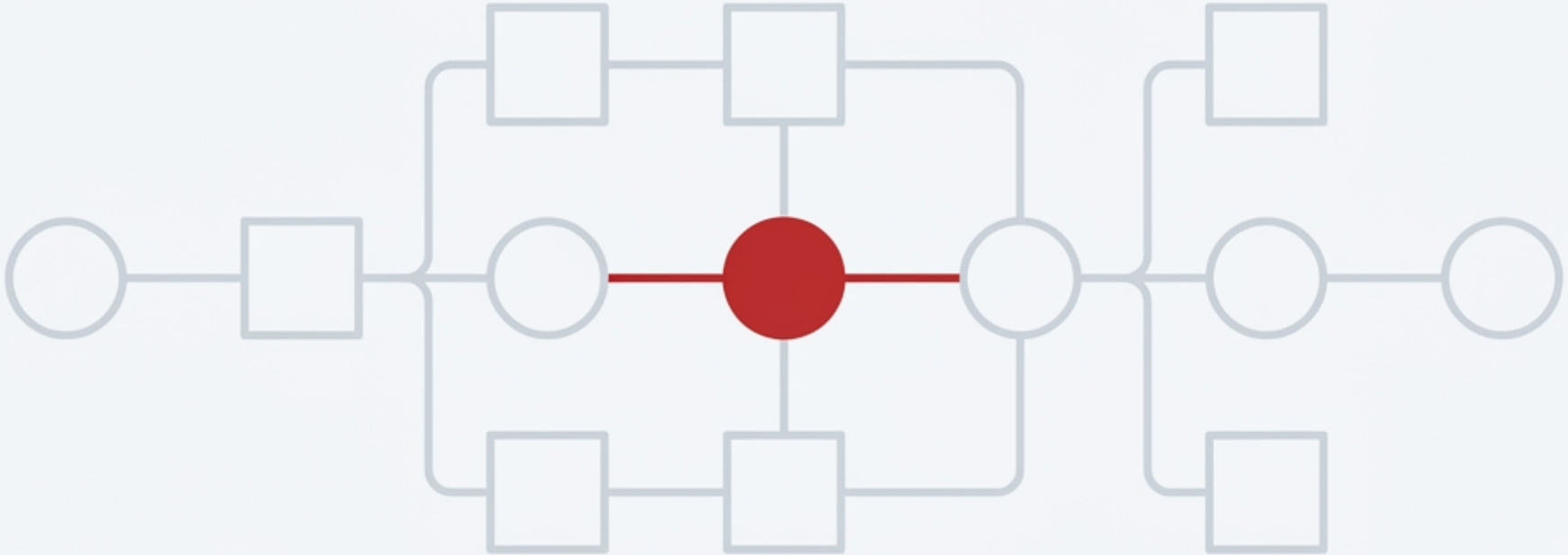
The 5xx Handbook

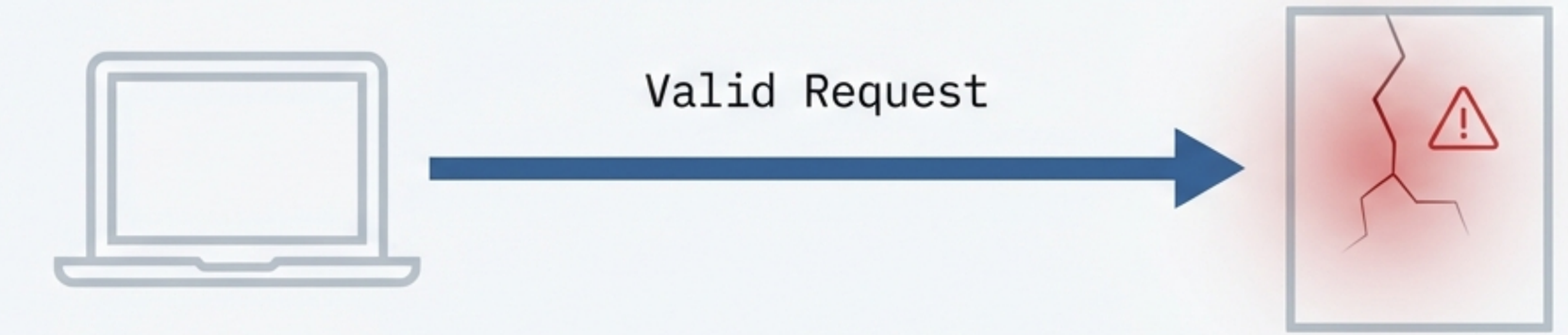
A Visual Guide to Diagnosing Server-Side Failures



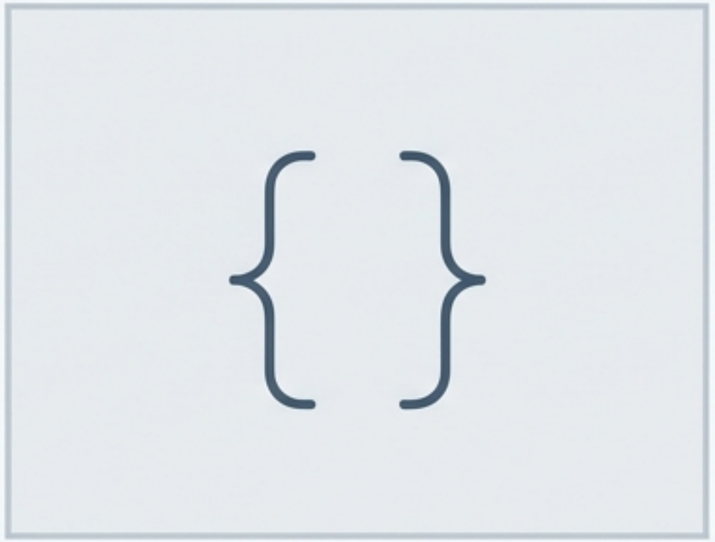
From Chaos to Clarity

It’s Not Them, It’s Us.

“The client sent a valid request, but the server failed to process it. The ‘fault’ lies entirely on the server.”



Three Families of Failure



1. Application Failures

The code itself breaks.

2. Infrastructure & Gateway Failures

The servers can’t communicate.

3. Capacity & Availability Failures

The system is overwhelmed.

{ } 500: ‘Something Broke Inside’

Plain English

Something broke inside, and I don’t have a specific name for it.

Root Cause

A generic catch-all for **Unhandled Exceptions**. This usually means the code crashed (e.g., Null Pointer, Database Connection failure, or Syntax error in a script).

Scenario

A user submits a form, but the backend code tries to save it to a database that is currently down. The code crashes, and the server returns a 500.

Where to Debug

- **Server Logs**: This is the ***only*** place to find the truth. Look for “***Stack Traces***”.
- **Permissions**: Check if the web server has permission to execute the script or read the `.env` file.

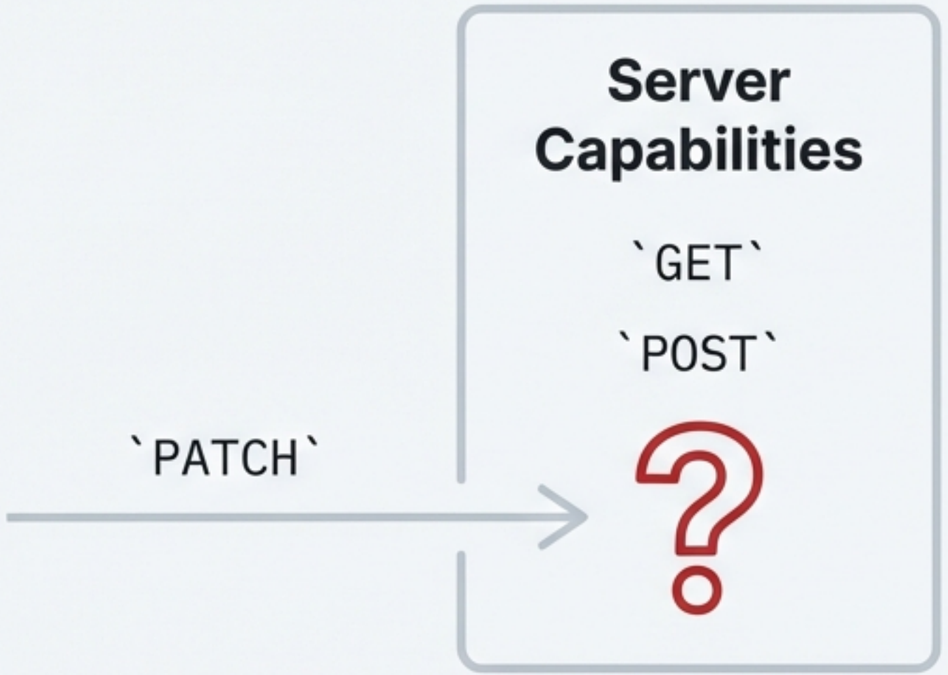
501: ‘I Don’t Know How To Do That’

Plain English

I don’t know how to do that yet.

Root Cause

The server does not support the HTTP method used in the request. This is rare on modern servers but common in early-stage API development.



Scenario

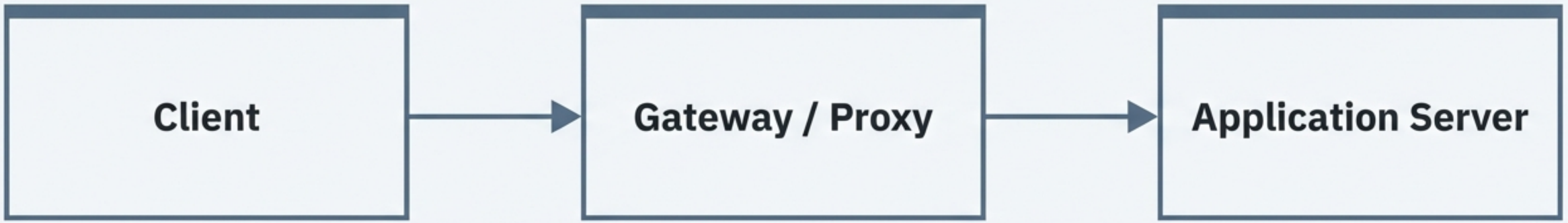
You send a `PATCH` request to a legacy server that only understands `GET` and `POST`.

Where to Debug

Check your HTTP Verb. If you are building the API, you need to add a handler for that specific method.

Gateway Failures: When Servers Miscommunicate

Briefly explain the role of a gateway or proxy server (like Nginx or a load balancer) as an essential intermediary that routes traffic from the client to the correct backend application server.



502: “The Backend Sent Me Garbage”

Plain English

The guy I’m talking to sent me garbage.

Root Cause

Communication Failure between servers. The server acting as a gateway received an invalid response from the backend application.

Scenario

Your Nginx proxy is running fine, but the PHP-FPM process it’s trying to talk to has crashed or returned a malformed header.

Where to Debug

Check if the **Backend Service** is actually running.

Check the **Proxy Configuration** (IP/Port) to ensure the gateway is pointing to the right place.

504: ‘The Backend Took Too Long’

Plain English

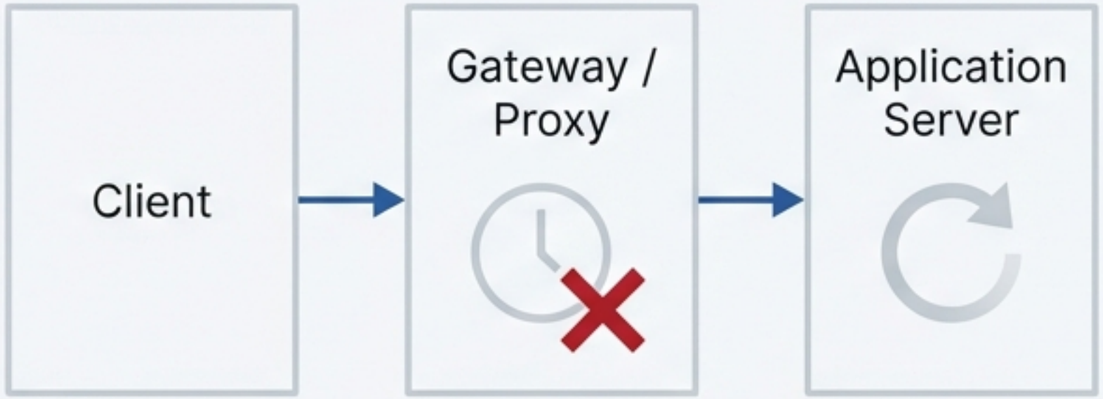
I've been waiting for the backend to answer, but I've given up.

Root Cause

The gateway waited for a response from the upstream server, but the **Timeout Limit** was reached. The backend is likely still working, but it’s too slow.

Scenario

A complex SQL query takes 60 seconds to run, but your Nginx proxy is configured to ‘timeout’ after 30 seconds.



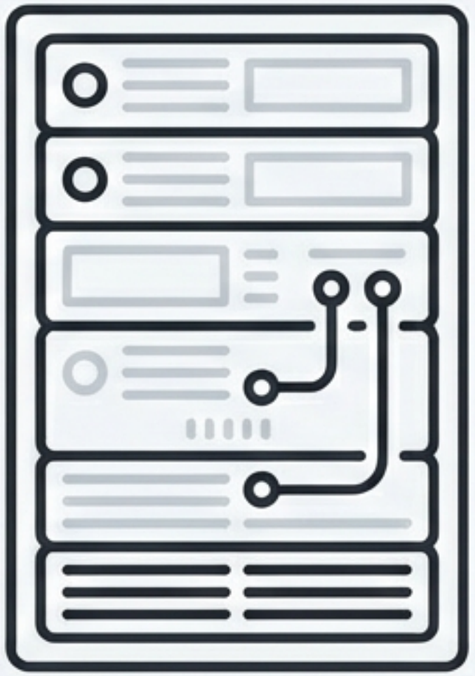
Where to Debug

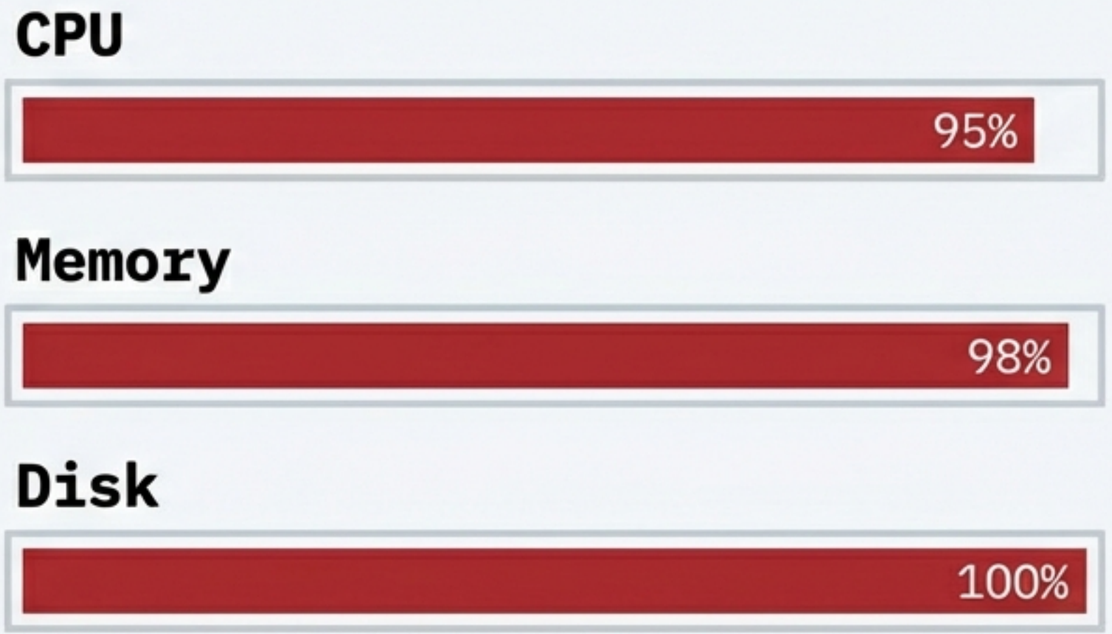
Slow Queries: Check for unoptimized database calls.

Timeout Settings: Increase `proxy_read_timeout` (Nginx) or equivalent.

Capacity Failures: When The Server is Overwhelmed

Sometimes the server's code and its connections are fine, but it has simply run out of fundamental resources like CPU, RAM, or disk space, making it unable to handle new requests.





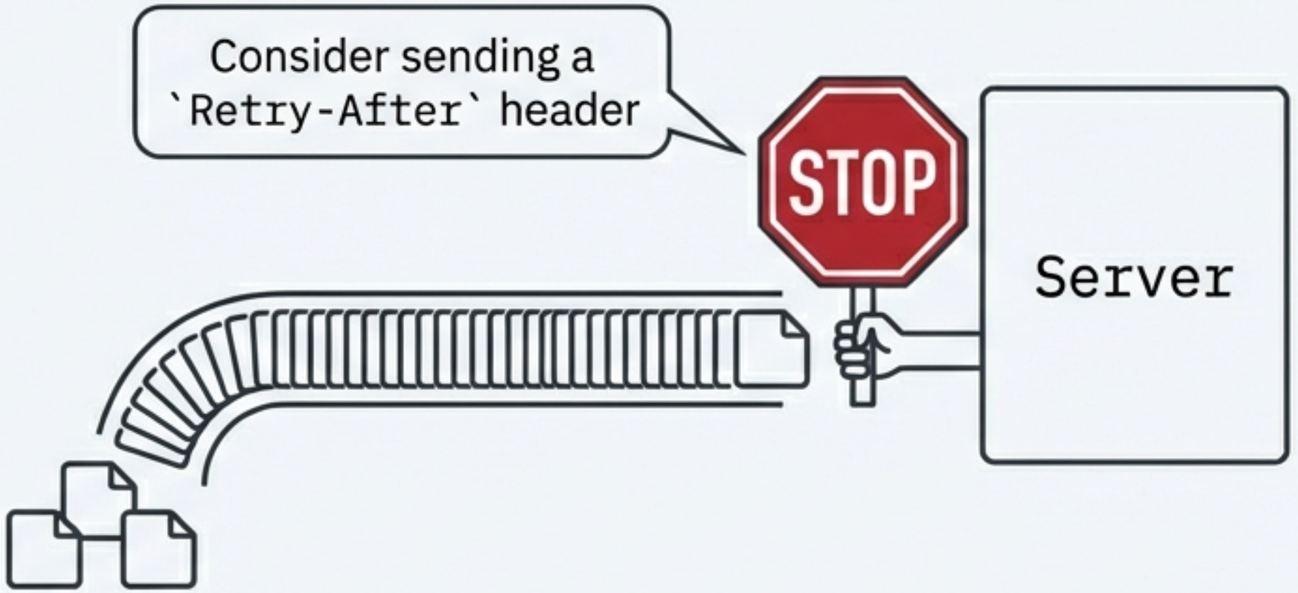
503: “I’m Too Busy Right Now”

Plain English

I’m too busy or closed for cleaning.

Root Cause

The server is physically up but cannot handle the request, usually due to **Overload** (CPU/RAM spikes) or **Scheduled Maintenance.**



Scenario

A viral tweet sends 1 million people to a small server at once. The server refuses new connections to keep from crashing.

Where to Debug

- **Resources:** Check CPU and Memory usage.
- **Retry-After Header:** Good APIs send this header to tell you when to try again.

507: “My Hard Drive Is Full”

Plain English

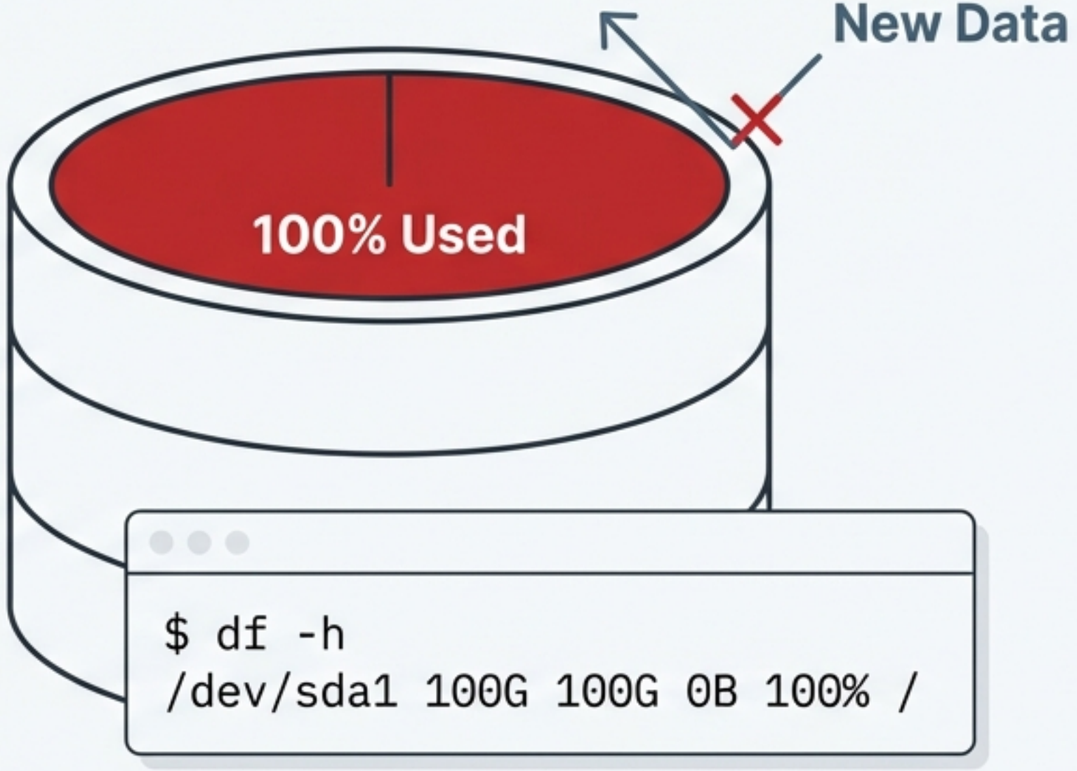
My hard drive is full.

Root Cause

The server cannot create or modify a resource because it has run out of disk space or has reached a storage quota.

Scenario

An automated logging script fills up the server’s primary partition, leaving 0 bytes for new data.

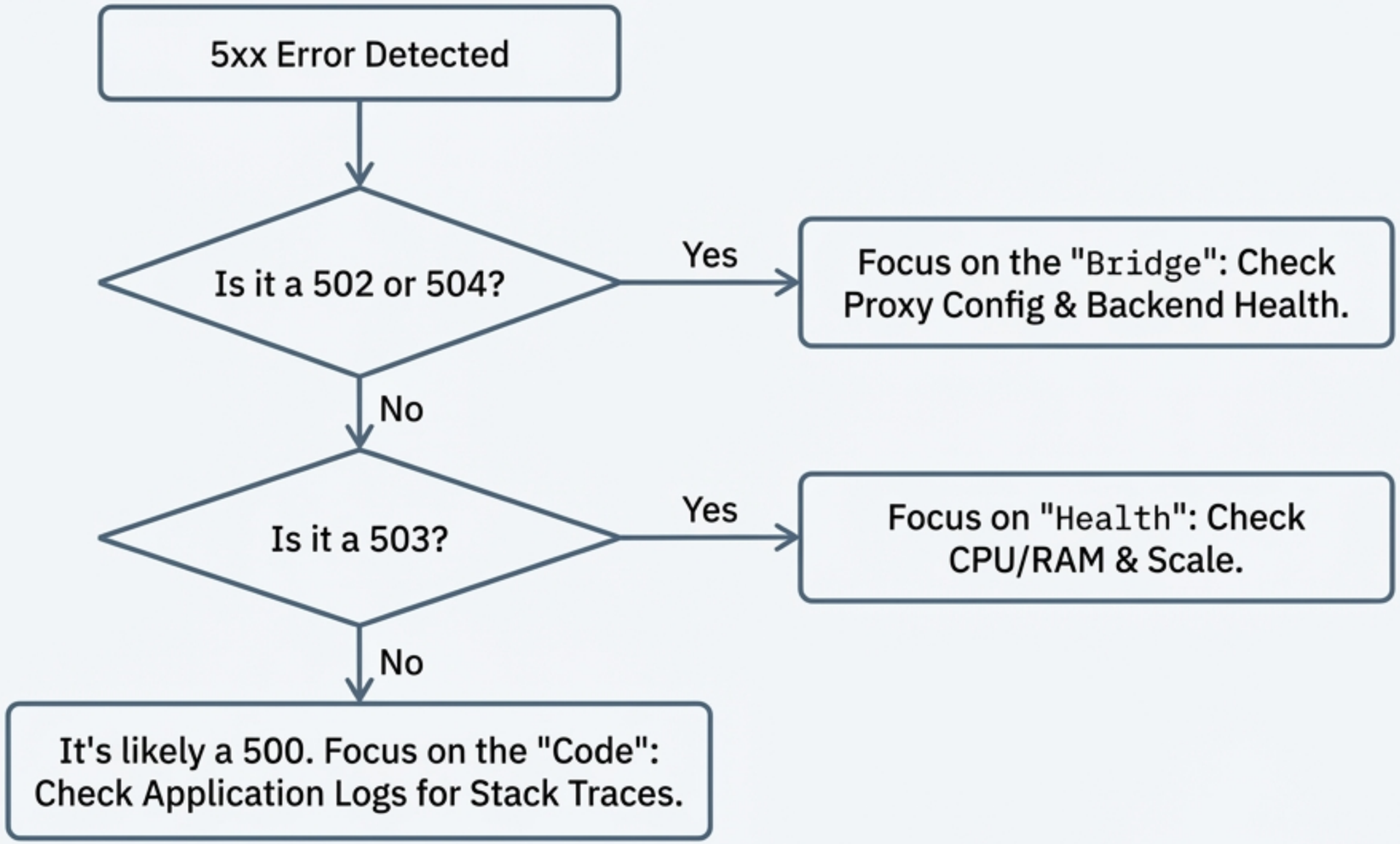


Where to Debug

Run `df -h` on Linux to check disk space.

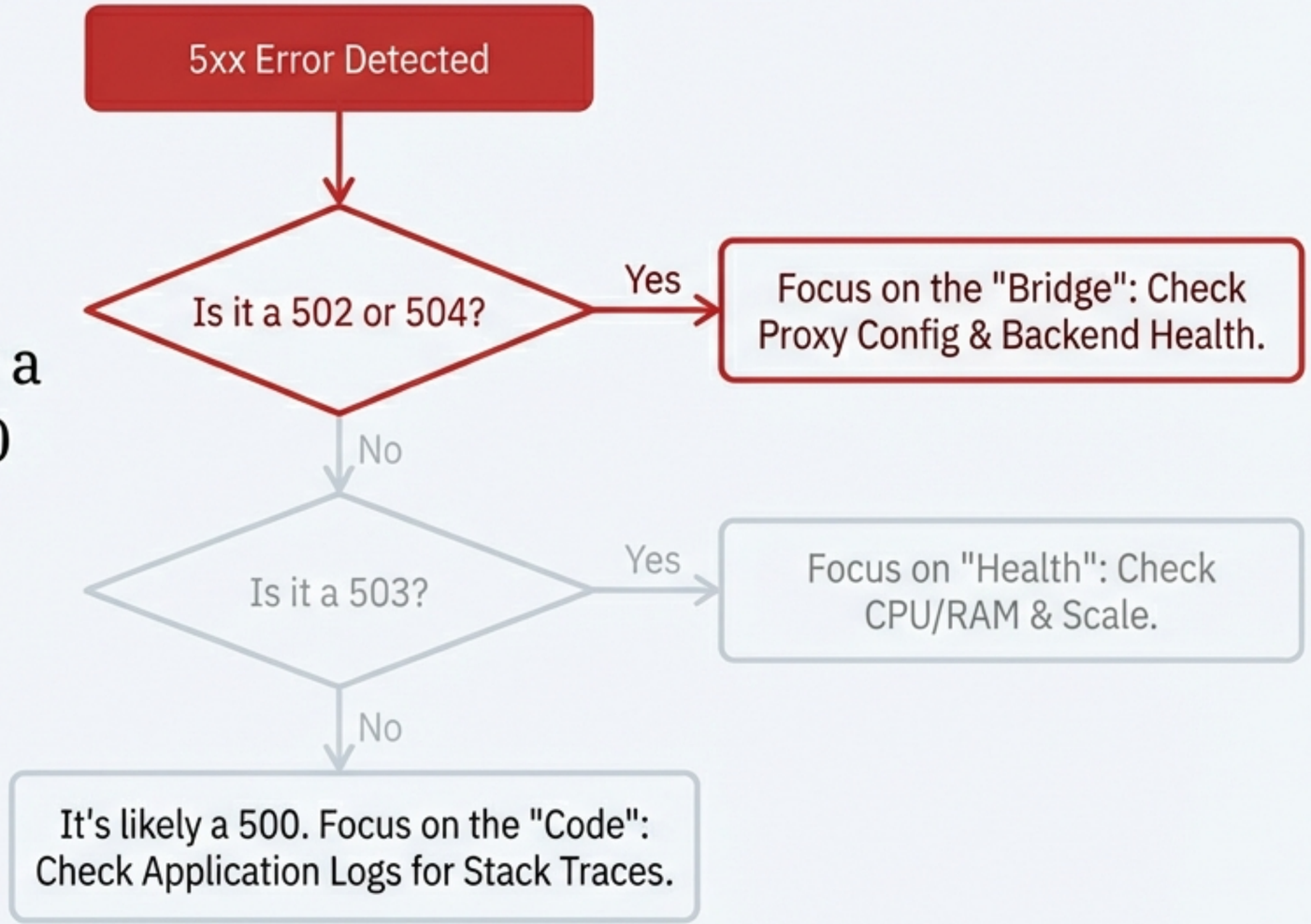
Your Troubleshooting Decision Tree

When you see a 5xx, don't guess. Ask these questions in order.



Decision Tree in Action: A 504 Gateway Timeout

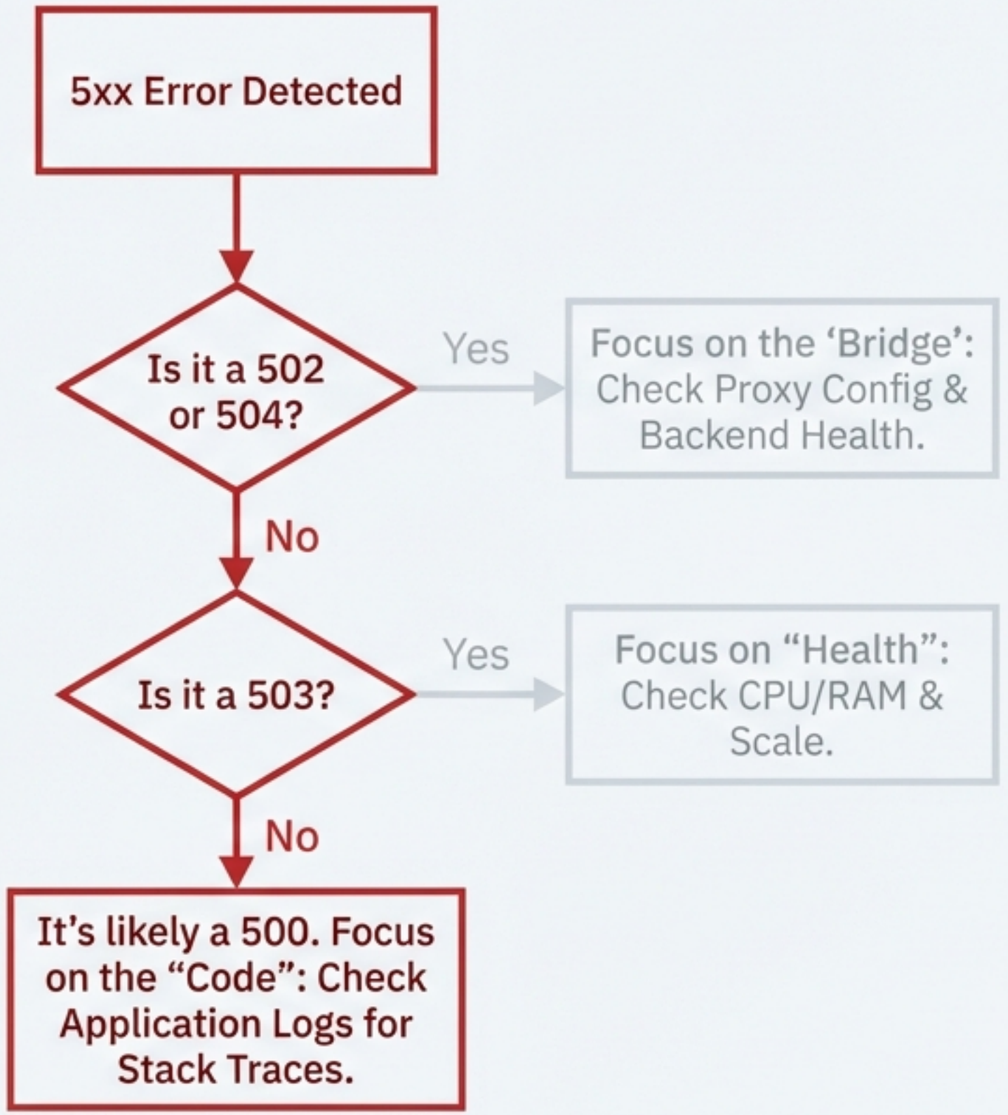
A user reports the checkout page is timing out. You check the logs and see a 504. Following the tree, you immediately suspect the ‘Bridge’. You discover a complex report query is taking 90 seconds to run, but the proxy timeout is only 30 seconds.



Decision Tree in Action: A 500 Internal Server Error

A simplified for fvo previous slides

A new feature deployment is followed by a spike in 5xx errors. The gateway logs look clean. Following the tree, you know to focus on the “Code”. The application logs reveal a “Null Pointer Exception” on a newly added line.



The Path to Mastery: From Debugging to Prevention

Reactive

Proactive

Structured Logging: Treat logs as event streams for easier parsing and analysis.

Error Tracking: Use tools like Sentry to catch and aggregate unhandled exceptions automatically.

Health Checks: Configure load balancers to automatically route traffic away from unhealthy instances.

Resources for Mastery

Sentry.io Blog on 500 Errors

An excellent guide on tracking down the root cause of code crashes.

https://blog.sentry.io/2021/05/06/debugging-500-internal-server-errors-with-sentry/

Nginx Error Log Documentation

Essential reading for debugging 502 and 504 gateway errors.

https://nginx.org/en/docs/http/ngx_http_core_module.html#error_log

The Twelve-Factor App - Logs

Learn why treating logs as event streams is the key to managing server errors proactively.

https://12factor.net/logs

Thank You

Questions?