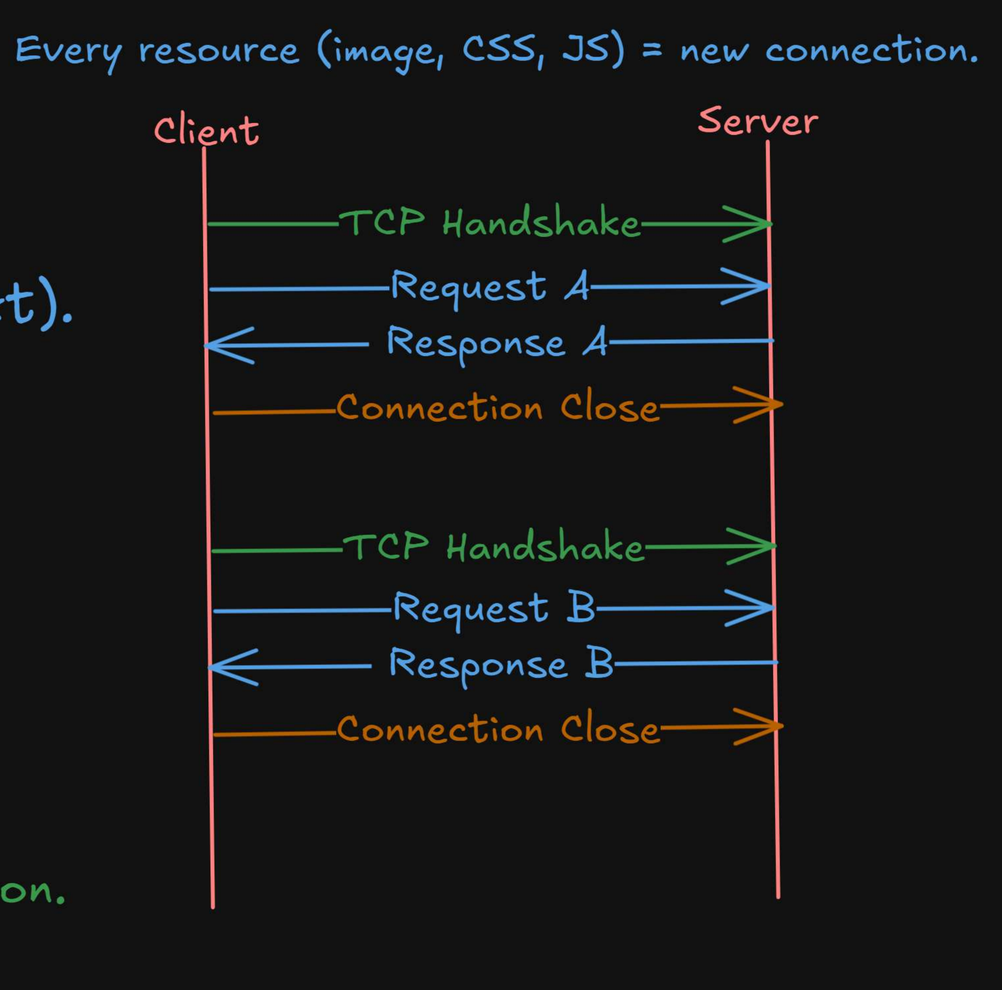
Imagine ordering 100 items from Amazon...

HTTP/1.0 → 1 Truck per Item 🚚🚚🚚
HTTP/1.1 → 1 Truck, Sequential 📦📦📦
HTTP/2 → 1 Truck, Parallel 📦📦📦📦
HTTP/3 → Drones, Independent 🚁🚁🚁

This is the story of HTTP —
the invisible hero that powers the entire web.

HTTP/1.0 (1996)

-> Text-based protocol (headers + body in plain text).
-> Each request = new TCP connection.
-> No persistent connections (no keep-alive).
-> Very inefficient for modern websites
with many resources (images, CSS, JS).



Real-life analogy:
-> Imagine every time you ask your friend a question,
you hang up the phone and call again for the next question.
Extremely slow.

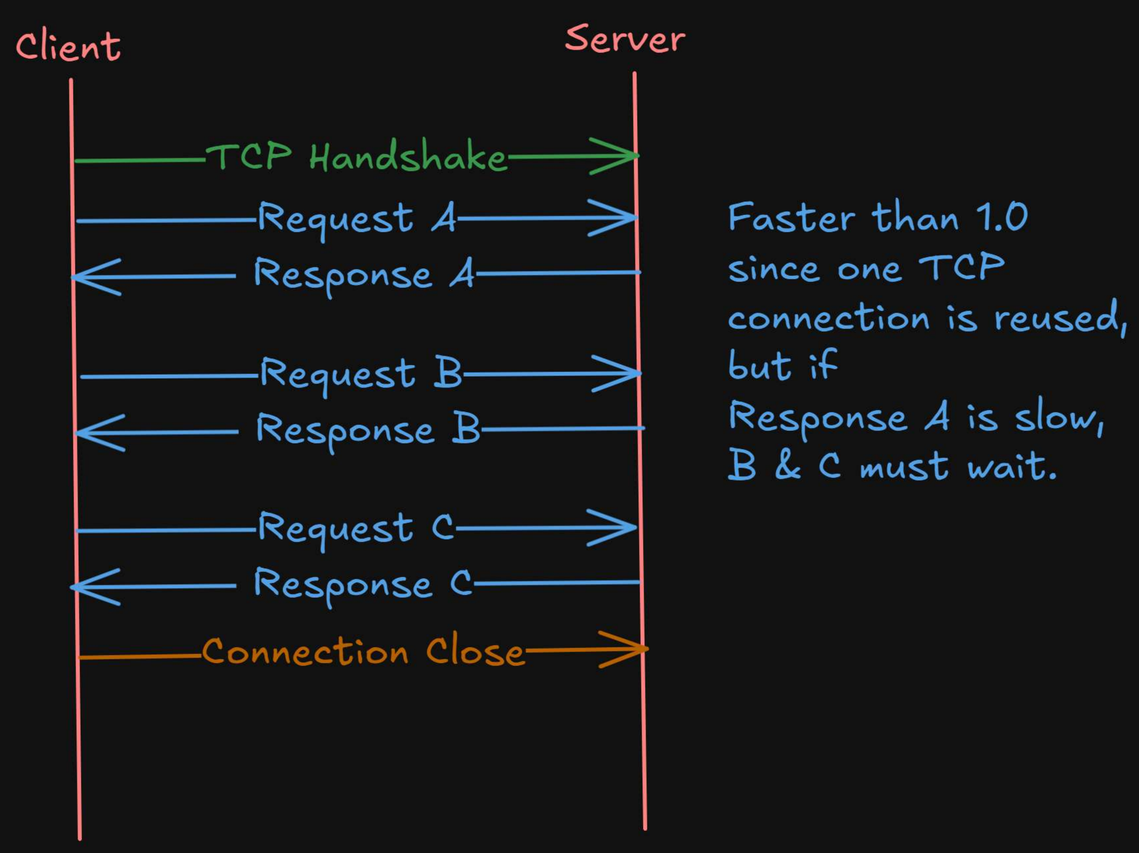
Use case:
Early static websites (mostly HTML + few images)

HTTP/1.1 (1997)

Improvement
-> Persistent connections with Connection: keep-alive.

-> Multiple requests on the same TCP connection.

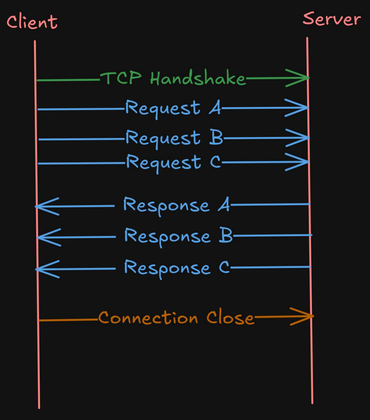
-> Still suffers from Head-of-Line (HOL) blocking →
if one request gets delayed, others behind it wait.



Real-life analogy:
Now you keep the phone line open and
ask multiple questions in the same call.
Still, if your friend delays answering one,
all other answers wait.

Use case:
Dynamic sites with multiple resources
(Amazon product page, blogs).

HTTP/2 (2015)



Major Upgrade:

Binary protocol (not text).

Multiplexing: Many requests/responses simultaneously over a single TCP connection.

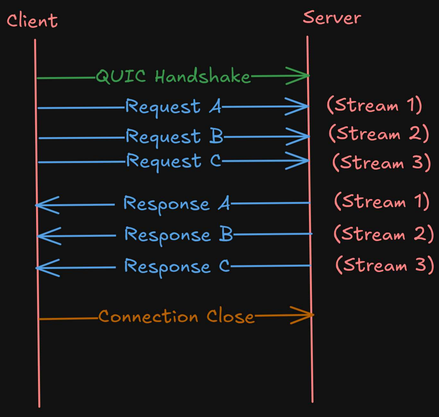
Header compression (HPACK) reduces repeated data like cookies.(using header tables + Huffman encoding)

Server Push: Server can send resources before client requests them.

Still limited by TCP HOL blocking → if one packet is lost, all streams pause until retransmission.

Use case:
Modern websites with lots of images, scripts, and styles (news portals, e-commerce).

HTTP/3 (2020s – QUIC/UDP)



Game Changer:

Based on QUIC (over UDP) → avoids TCP's HOL blocking.

Independent streams → one packet loss doesn't affect others.

TLS 1.3 built-in → always encrypted, no extra handshake.

0-RTT connection setup → instant resume of sessions.

Works great in mobile networks where packet loss + switching between WiFi/MobileData is common.

Use case:
Streaming (YouTube, Netflix), gaming, video conferencing, mobile browsing.

Myth

REST vs gRPC

REST is just an architectural style.
If the server and client support HTTP/2 or even HTTP/3, your REST API will automatically run over it — and take advantage of multiplexing, header compression, and lower latency.

But here's the key difference: REST was designed in the HTTP/1.1 era, so most implementations don't really leverage all the advanced features of HTTP/2.

On the other hand, gRPC was built from day one on top of HTTP/2.

So yes, REST can run on HTTP/2 or even HTTP/3.
But gRPC is engineered to unlock the full power of HTTP/2, which is why it's often the better choice for high-performance microservices and APIs."