

# Scaling WebSockets

*With HAProxy, Pub - Sub and Distributed Caches*



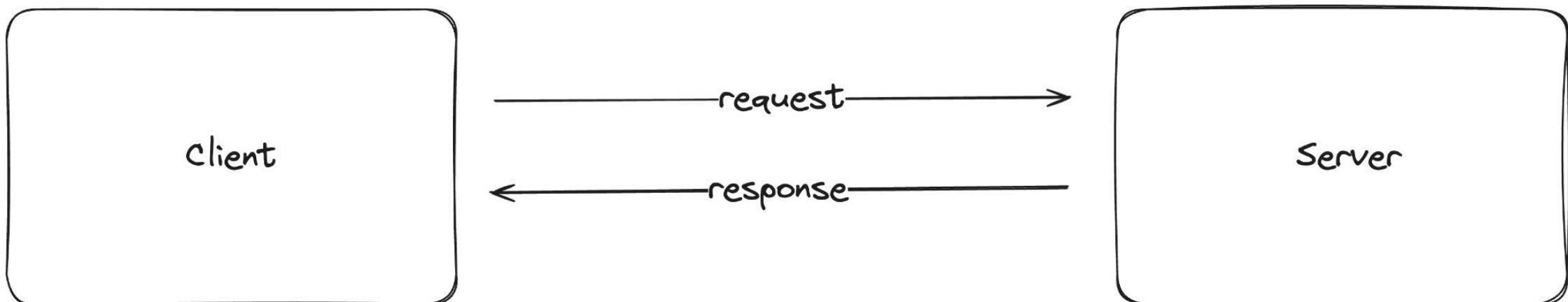
# Introduction





# Exchanging information over the WWW

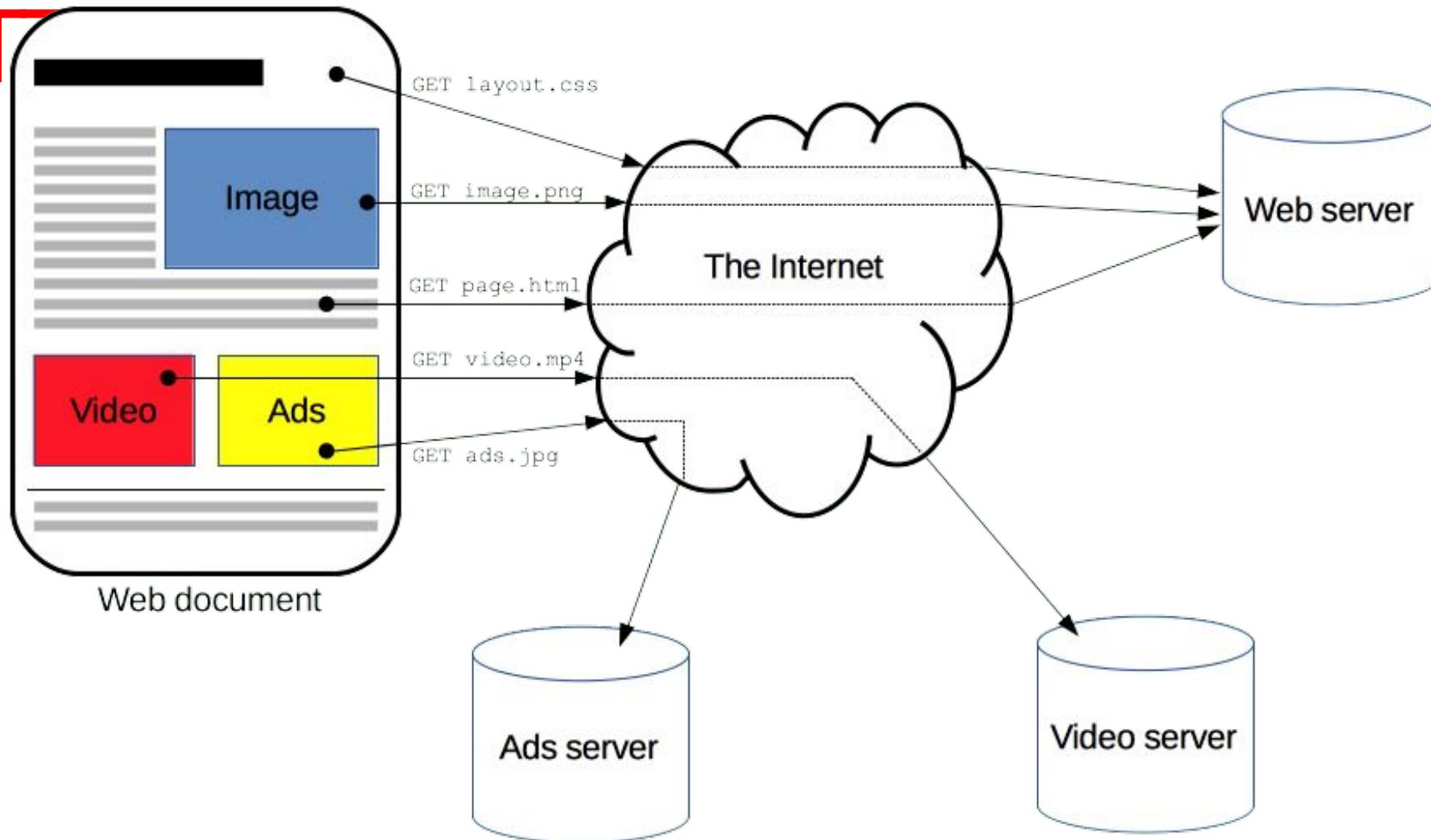
## Client - Server architecture





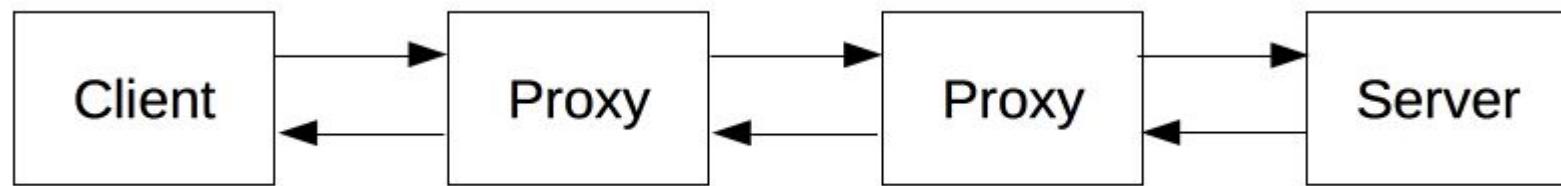
# Communication happens over

HT





# Hypertext Transfer Protocol



- *It is a stateless protocol*
- *Uses TCP as the transport layer protocol*

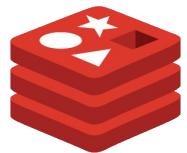




# HTTP Protocol

## TCP Connection Establishment in HTTP

- Before HTTP communication begins, a TCP connection must be established, requiring several round-trips
- HTTP/1.0 defaults to opening separate TCP connections for each request/response pair, which can be inefficient

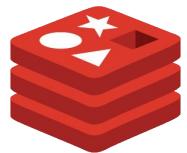




# HTTP Protocol

## Improvements in HTTP/1.1

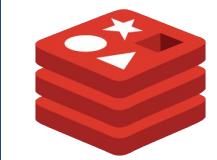
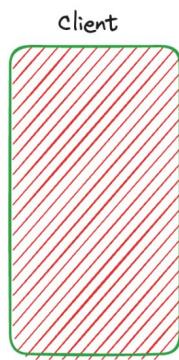
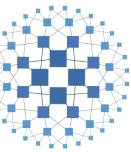
- HTTP/1.1 introduced pipelining and persistent connections to optimize TCP usage, although pipelining proved challenging

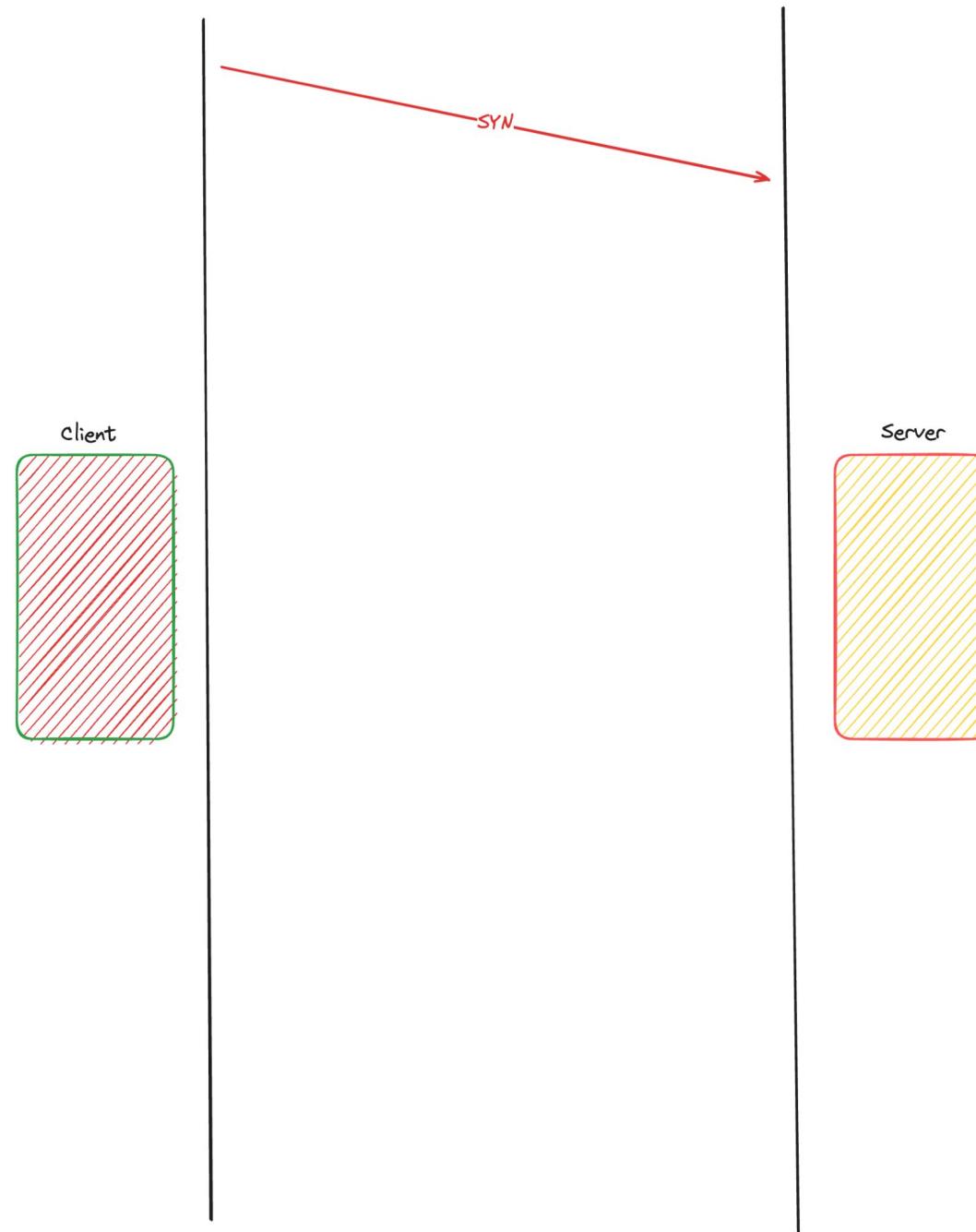


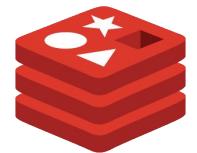
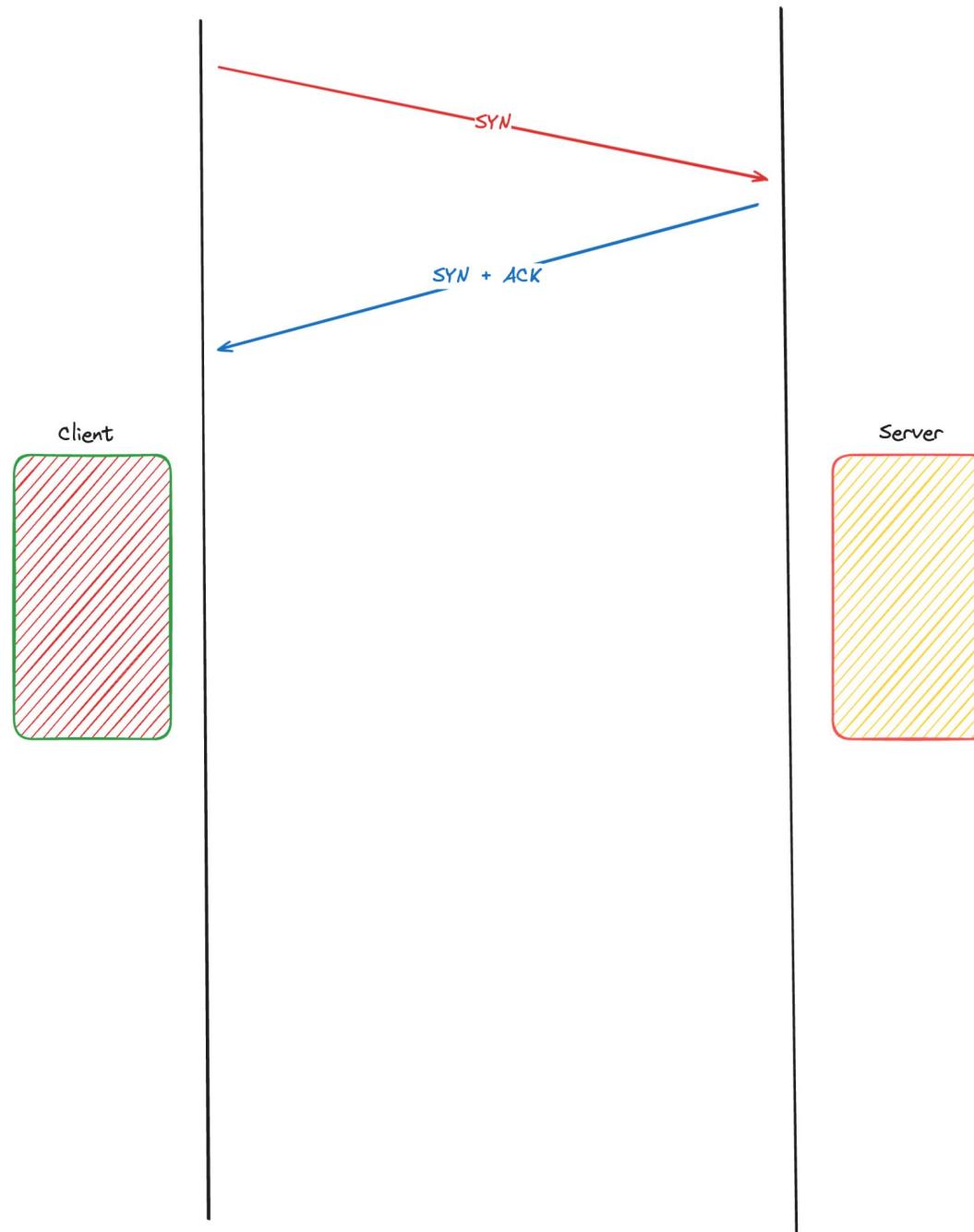
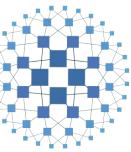


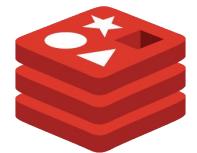
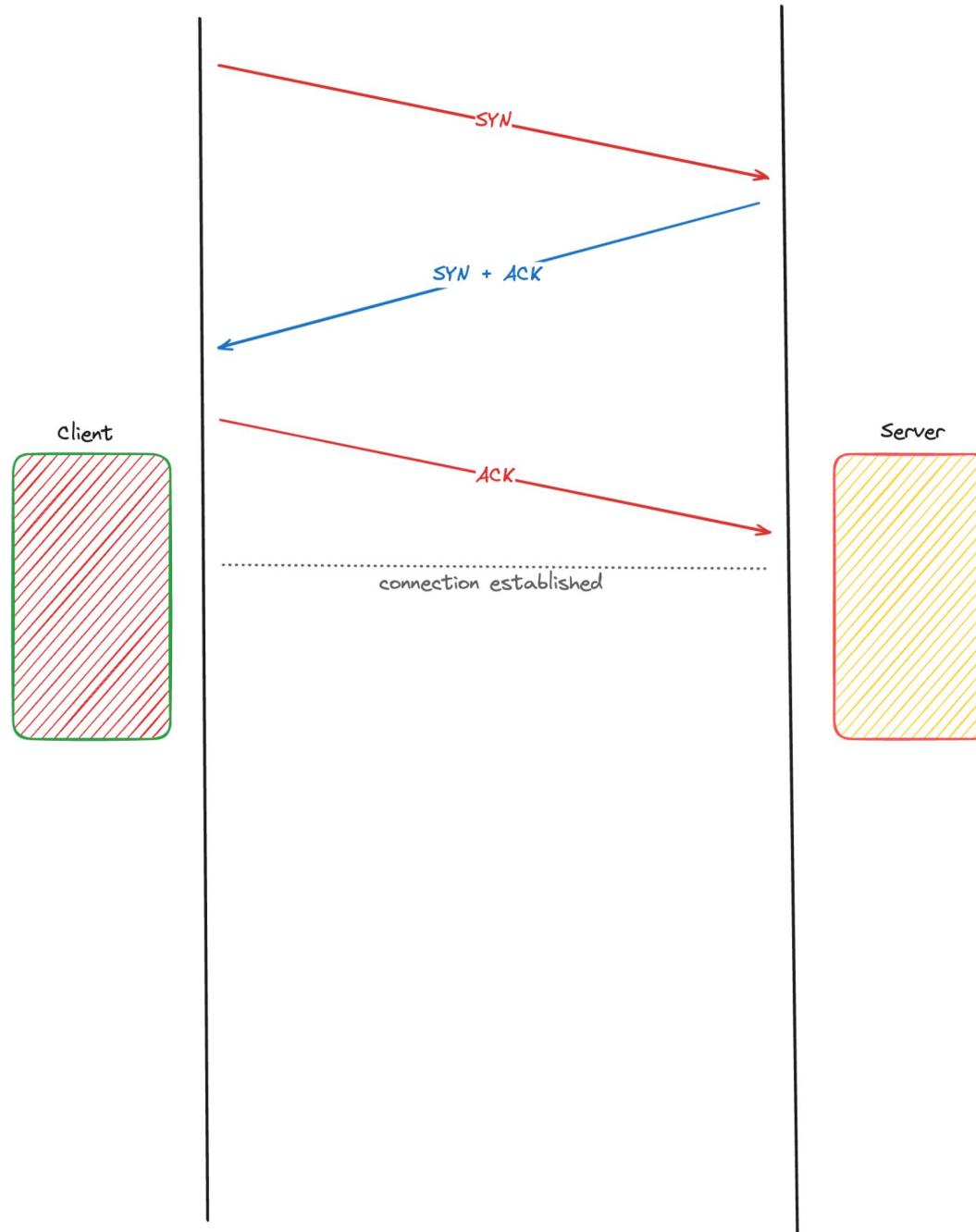
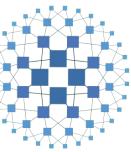
# Demonstration

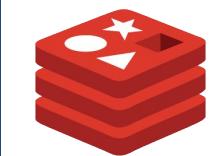
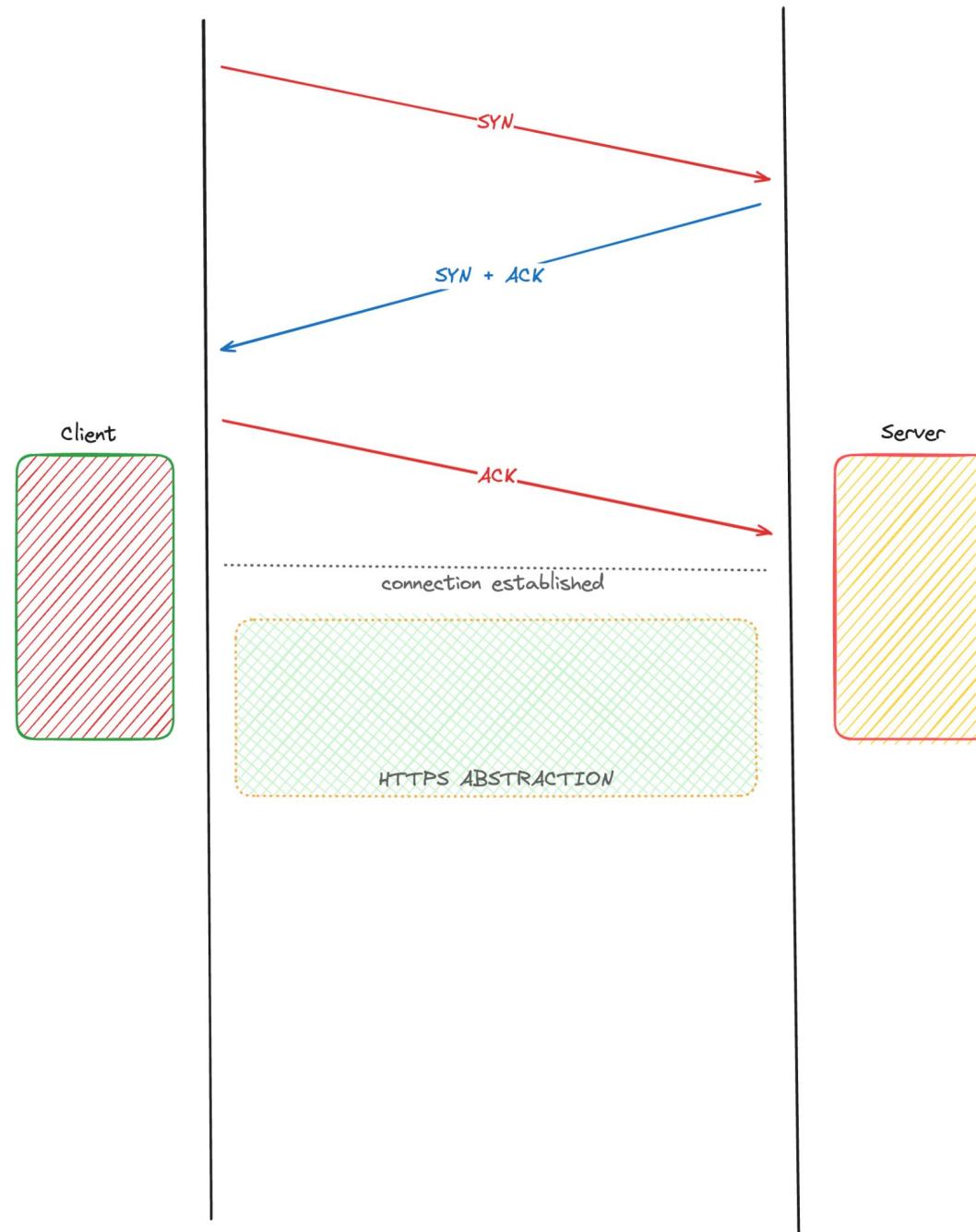


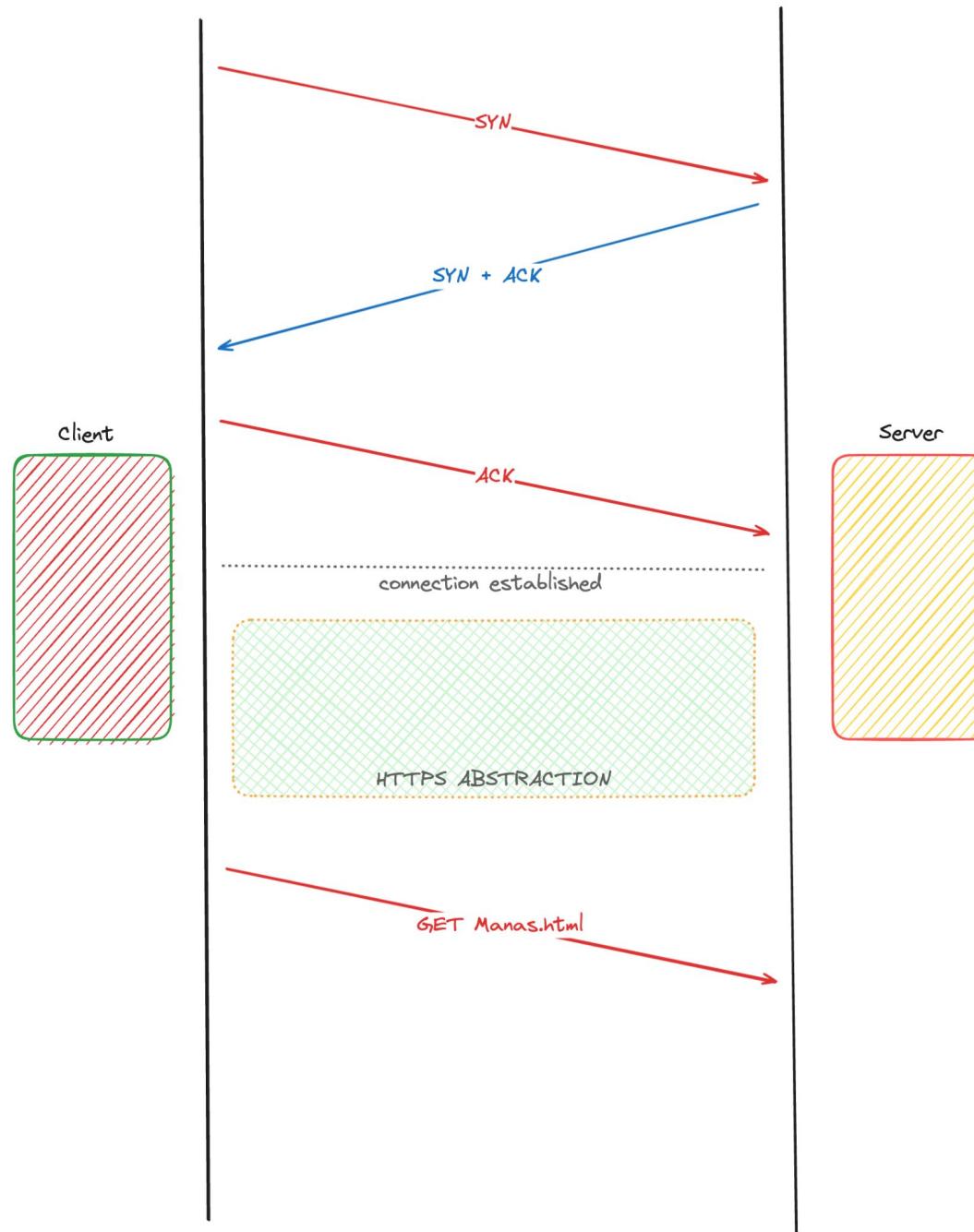
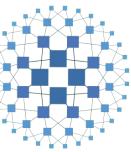


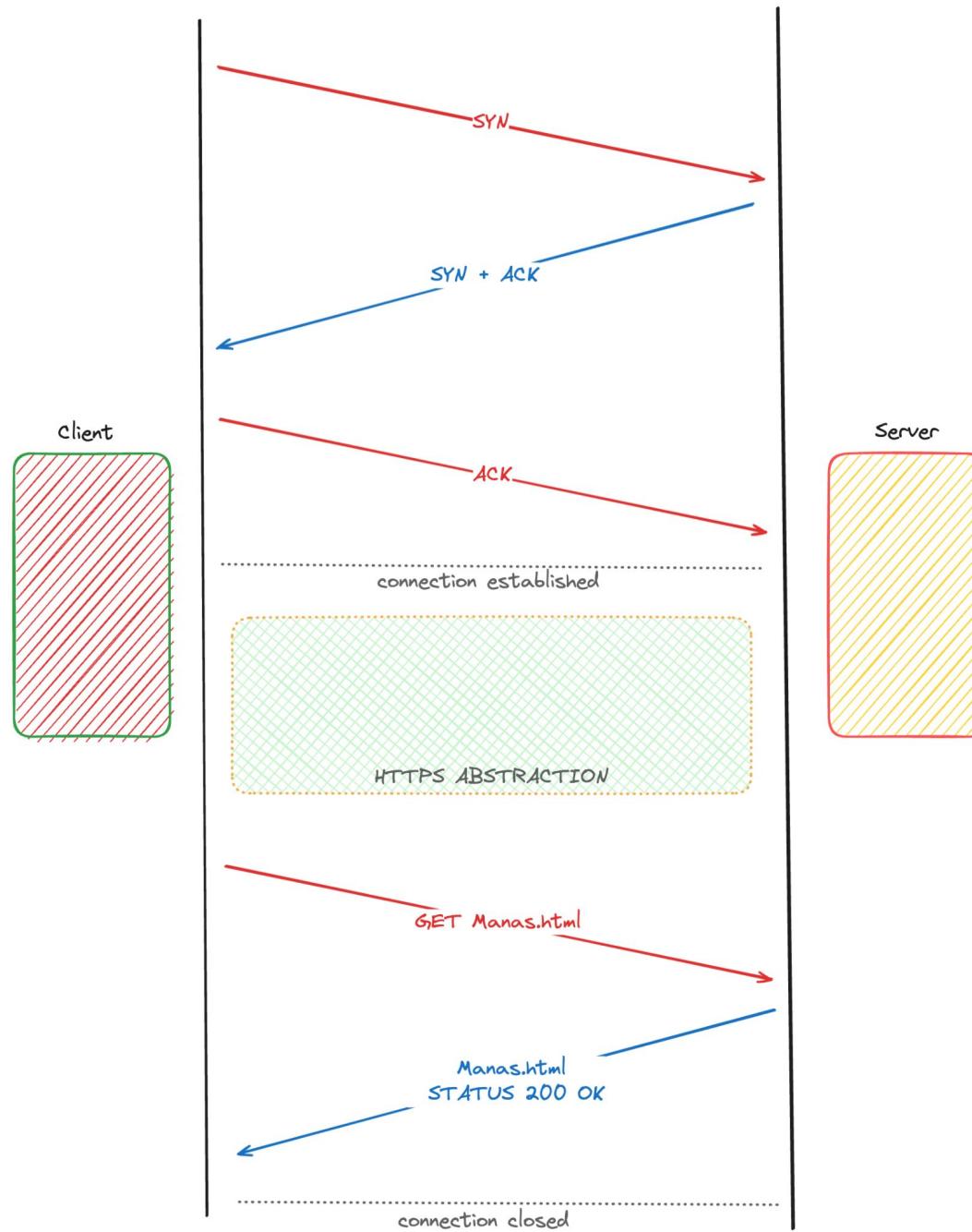






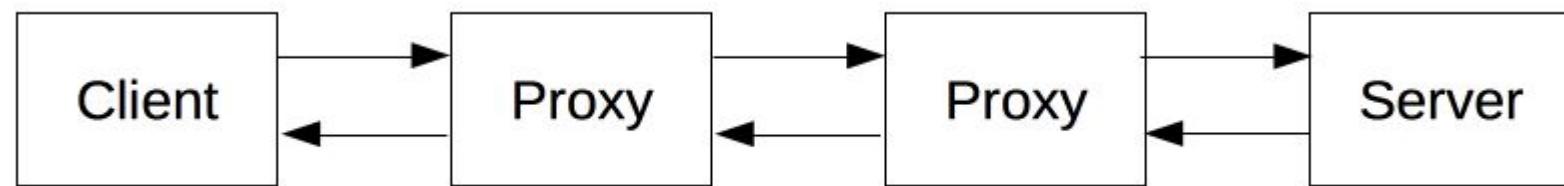








# Hypertext Transfer Protocol



- *The browser is always the entity initiating the request. It is never the server that sends a response automatically*
- *The connection is cut-off after a request-response pair*

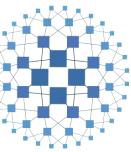




# How to achieve a stateful connection ?

## WebSocket Protocol





# Why WebSocket Protocol ?

- *Full-duplex, bi-directional communication*
- *Persistent Stateful Connection*
- *Server can send resources even without client initiating a request*



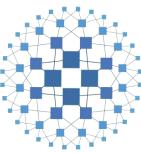


# WebSocket Protocol

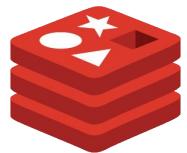
*Initial GET request over a HTTP / 1.1 with an **upgrade** flag*

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade ←
Sec-WebSocket-Key: dGhlIHNbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```





# WebSocket Protocol





# Proof of Work





```
// WebSocketServer.ts

import { WebSocketServer, WebSocket } from 'ws'
import express from 'express'

function onSocketPreError(e: Error) {
  console.log(e)
}

function onSocketPostError(e: Error) {
  console.log(e)
}

const app = express()
const PORT: string | number = process.env.PORT || 8080

const httpServer = app.listen(PORT, () => {
  console.log(`HTTP Server is running on PORT ${PORT}`)
})

const wss = new WebSocketServer({ noServer: true })
```





```
// WebSocketServer.ts
```

```
httpServer.on('upgrade', (req, socket, head) => {
  socket.on('error', onSocketPreError)

  wss.handleUpgrade(req, socket, head, (ws) => {
    socket.removeListener('error', onSocketPreError)
    wss.emit('connection', ws, req)
  })
})
```





<https://github.com/akarmanya/wscale>



```
// WebSocketServer.ts

wss.on('connection', function connection(ws: WebSocket, req) {
  console.log(
    `HTTP Server upgraded to WSS Server and is running on PORT ${PORT}`
  )

  ws.on('error', onSocketPostError)

  ws.on('message', function message(data: any, isBinary: boolean) {
    try {
      wss.clients.forEach(function each(client) {
        if (client.readyState === WebSocket.OPEN) {
          client.send(data, { binary: isBinary })
        }
      })
    } catch (error) {
      console.error(error)
    }
  })

  ws.on('close', () => {
    console.log('Connection closed from ws server')
  })
})
```



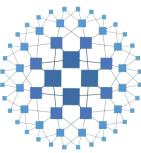


Wi-Fi: en0

(ip.addr eq 192.168.254.10 and ip.addr eq 192.168.254.81) and (tcp.port eq 8080)

No.	Time	Source	Destination	Protocol	Length	Info
737...	361.019315	192.168.254.10	192.168.254.81	TCP	78	49911 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=269310857 TSecr=0 SACK_PERM=1
737...	361.022139	192.168.254.81	192.168.254.10	TCP	74	8080 → 49911 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=4120177935 TSecr=0
737...	361.022260	192.168.254.10	192.168.254.81	TCP	66	49911 → 8080 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=269310859 TSecr=4120177935
737...	361.022588	192.168.254.10	192.168.254.81	HTTP	590	GET / HTTP/1.1
737...	361.025558	192.168.254.81	192.168.254.10	TCP	66	8080 → 49911 [ACK] Seq=1 Ack=525 Win=30080 Len=0 TSval=4120177939 TSecr=269310859
737...	361.585718	192.168.254.81	192.168.254.10	HTTP	209	HTTP/1.1 101 Switching Protocols
737...	361.585822	192.168.254.10	192.168.254.81	TCP	66	49911 → 8080 [ACK] Seq=525 Ack=144 Win=131584 Len=0 TSval=269311417 TSecr=4120178364
738...	381.479311	192.168.254.81	192.168.254.10	WebSocket	68	WebSocket Ping [FIN]
738...	381.479396	192.168.254.10	192.168.254.81	TCP	66	49911 → 8080 [ACK] Seq=525 Ack=146 Win=131584 Len=0 TSval=269331255 TSecr=4120198393
738...	381.479532	192.168.254.10	192.168.254.81	WebSocket	72	WebSocket Pong [FIN] [MASKED]
738...	381.482689	192.168.254.81	192.168.254.10	TCP	66	8080 → 49911 [ACK] Seq=146 Ack=531 Win=30080 Len=0 TSval=4120198396 TSecr=269331255
738...	391.720247	192.168.254.10	192.168.254.81	WebSocket	75	WebSocket Text [FIN] [MASKED]
738...	391.723511	192.168.254.81	192.168.254.10	TCP	66	8080 → 49911 [ACK] Seq=146 Ack=540 Win=30080 Len=0 TSval=4120208638 TSecr=269341461
738...	391.777315	192.168.254.81	192.168.254.10	WebSocket	89	WebSocket Text [FIN]
738...	391.777398	192.168.254.10	192.168.254.81	TCP	66	49911 → 8080 [ACK] Seq=540 Ack=169 Win=131584 Len=0 TSval=269341517 TSecr=4120208691
738...	405.199381	192.168.254.10	192.168.254.81	WebSocket	72	WebSocket Connection Close [FIN] [MASKED]
738...	405.203350	192.168.254.81	192.168.254.10	TCP	66	8080 → 49911 [ACK] Seq=169 Ack=546 Win=30080 Len=0 TSval=4120222118 TSecr=269354904
738...	405.222155	192.168.254.81	192.168.254.10	WebSocket	70	WebSocket Connection Close [FIN]
738...	405.222218	192.168.254.10	192.168.254.81	TCP	66	49911 → 8080 [ACK] Seq=546 Ack=173 Win=131584 Len=0 TSval=269354926 TSecr=4120222137
738...	405.240117	192.168.254.81	192.168.254.10	TCP	66	8080 → 49911 [FIN ACK] Seq=172 Ack=546 Win=30080 Len=0 TSval=4120222154 TSecr=269354926
▶ Frame 73729: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface en0, id 0						
▶ Ethernet II, Src: Apple_7f:c0:07 (ac:bc:32:7f:c0:07), Dst: Raspberr_02:cb:df (b8:27:eb:02:cb:df)						
▶ Internet Protocol Version 4, Src: 192.168.254.10, Dst: 192.168.254.81						
▼ Transmission Control Protocol, Src Port: 49911, Dst Port: 8080, Seq: 0, Len: 0						
Source Port: 49911						
Destination Port: 8080						
[Stream index: 27]						
[TCP Segment Len: 0]						
Sequence number: 0 (relative sequence number)						
Sequence number (raw): 2402399483						
[Next sequence number: 1 (relative sequence number)]						
Acknowledgment number: 0						
Acknowledgment number (raw): 0						
1011 .... = Header Length: 44 bytes (11)						
▼ Flags: 0x002 (SYN)						
000. .... .... = Reserved: Not set						
...0 .... .... = Nonce: Not set						
.... 0.... .... = Congestion Window Reduced (CWR): Not set						
.... .0.... .... = ECN-Echo: Not set						
.... .... 0... = Urgent: Not set						





# Why WebSockets are difficult to *scale*

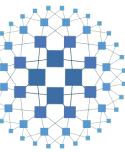
- *HTTP works on a duplex communication model*
- *The client initiates a request to the server for a resource*
- *The server responds back with a response*
- *Once the client receives the response, the connection is closed*



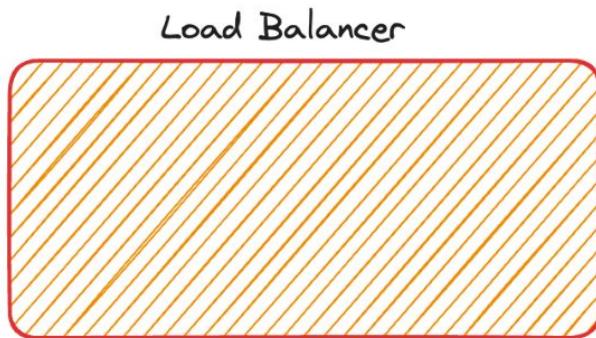


# Demonstration





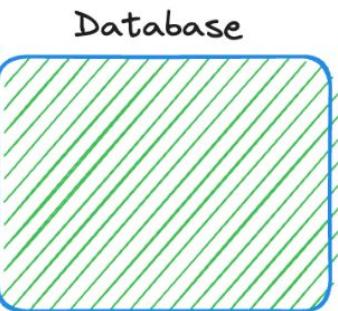
client

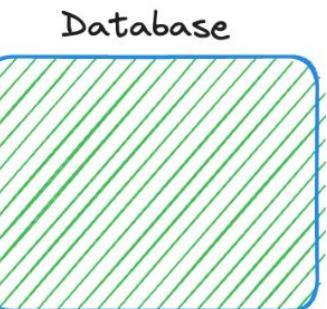
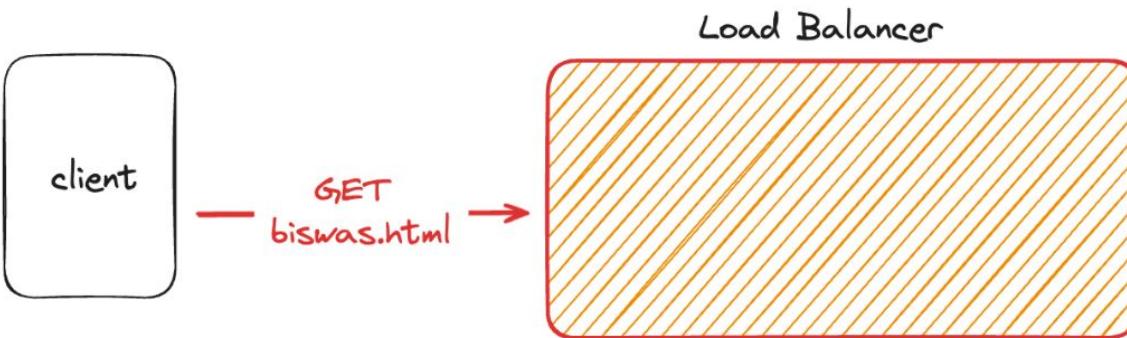


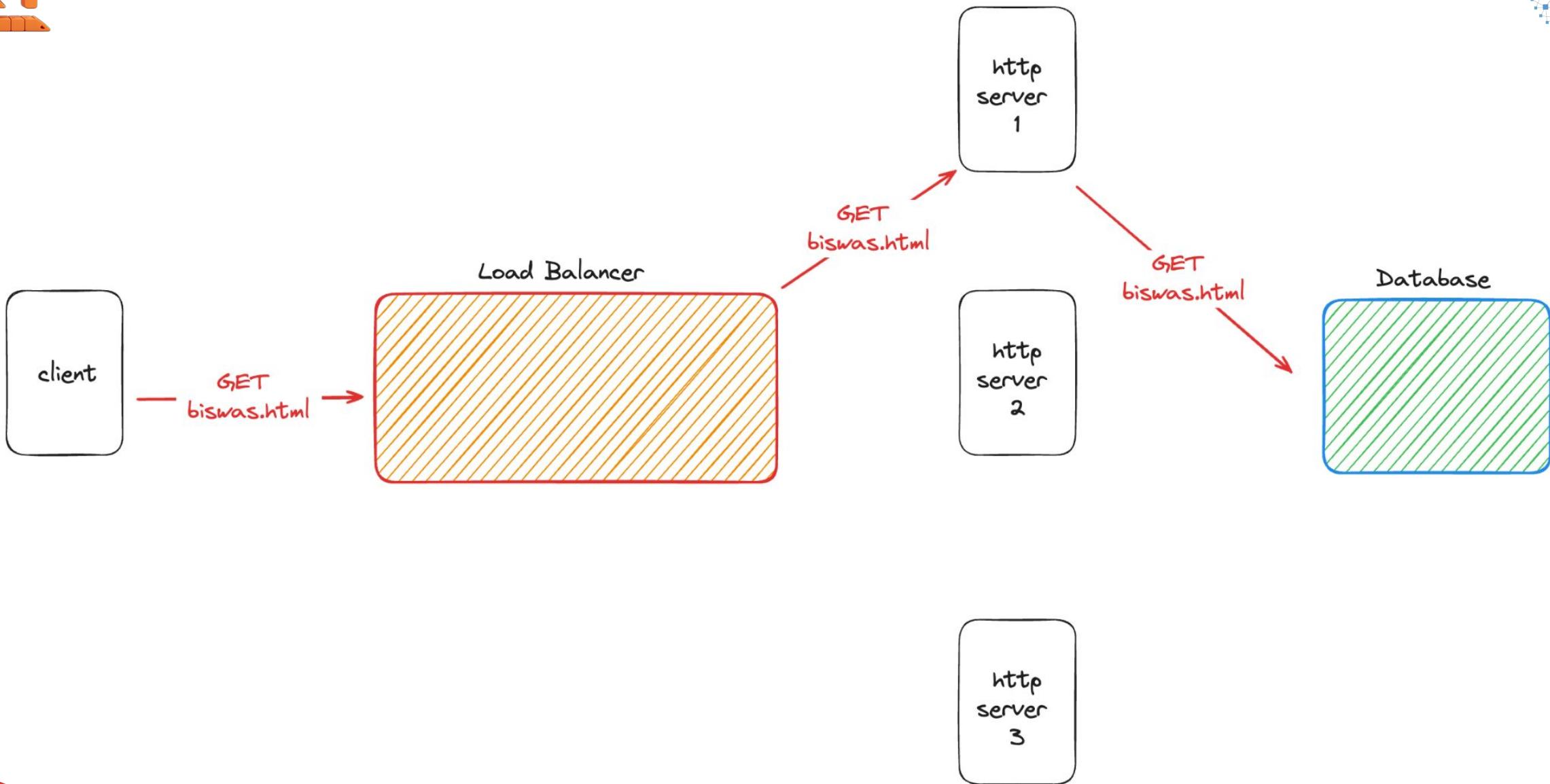
http  
server  
1

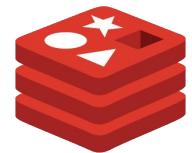
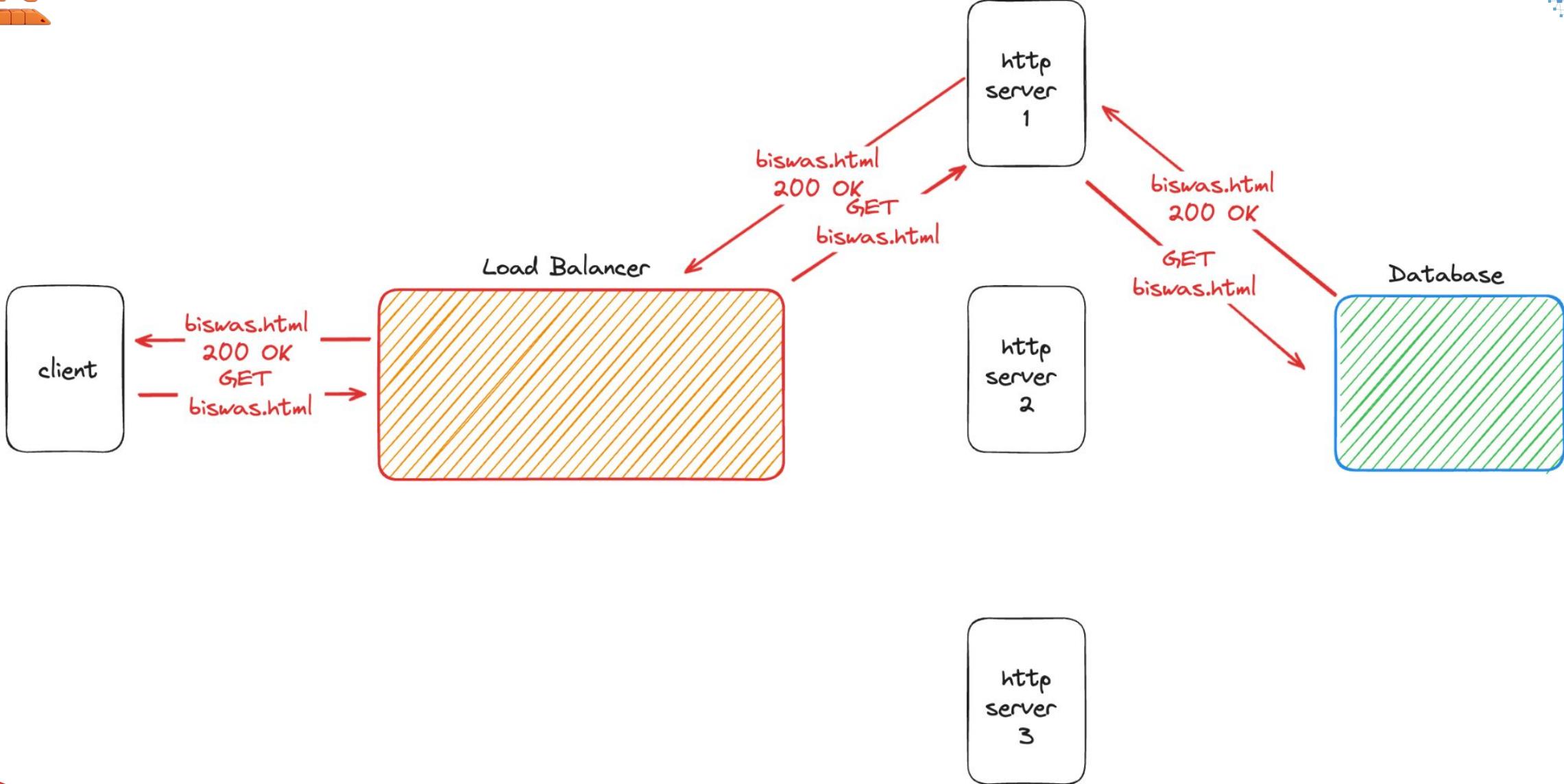
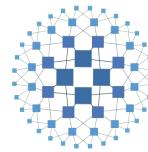
http  
server  
2

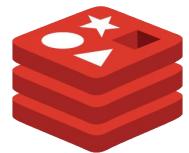
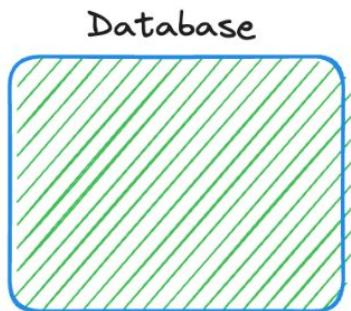
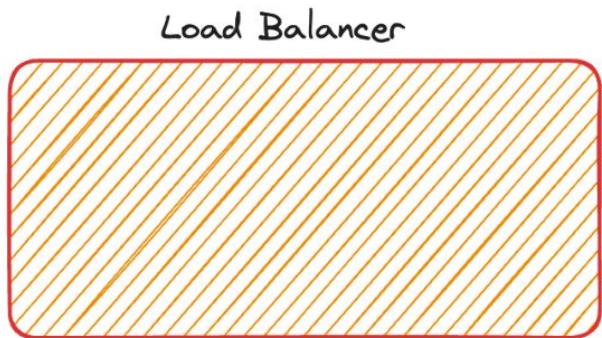
http  
server  
3

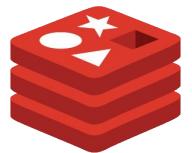
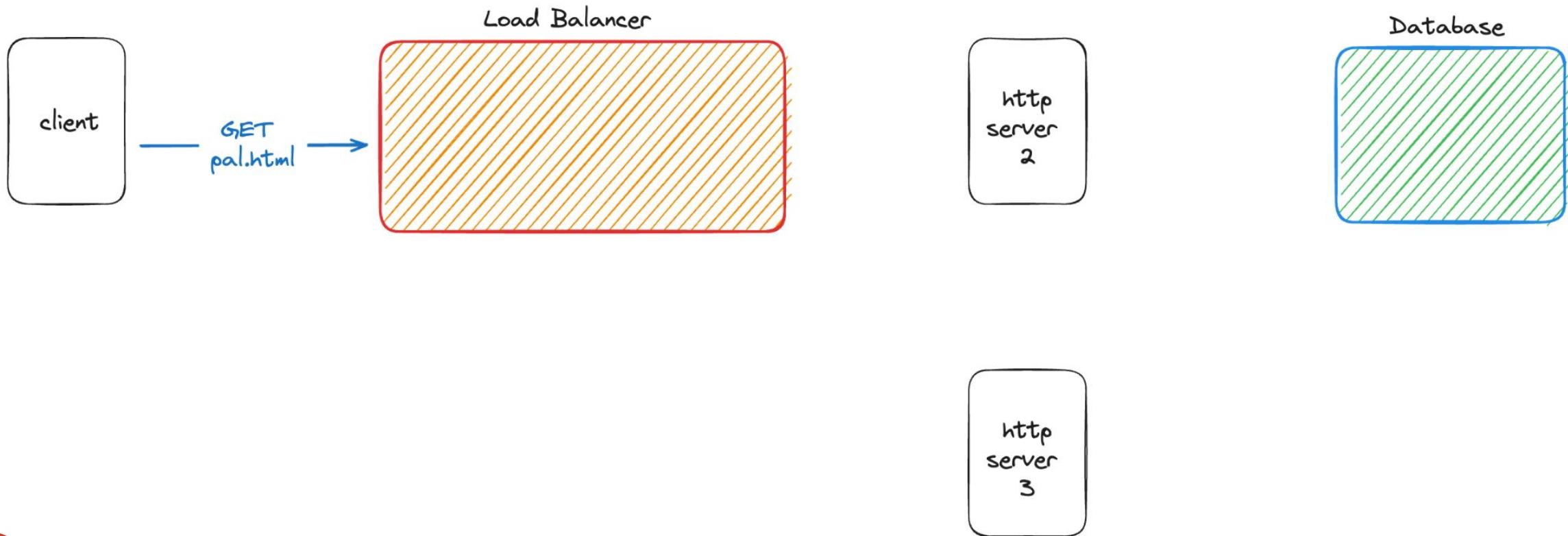


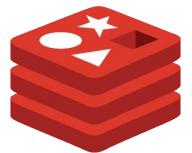
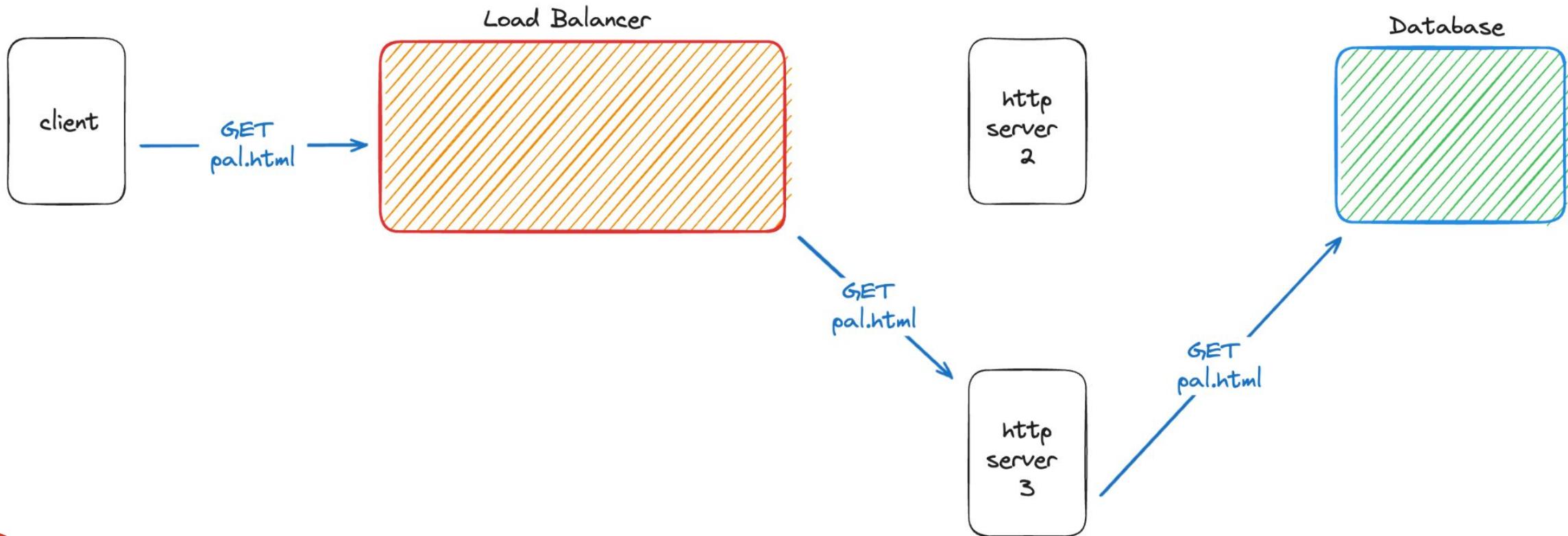


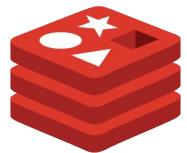
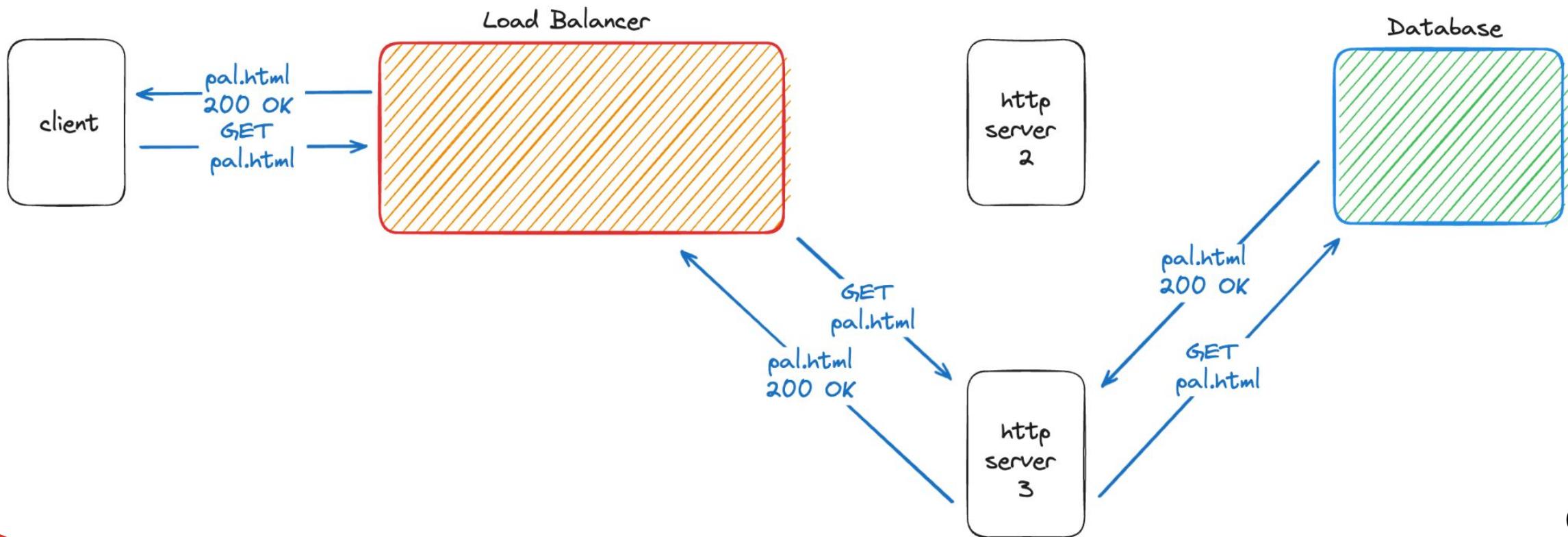


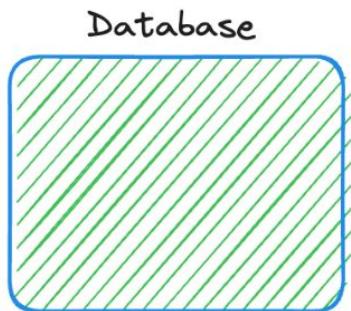
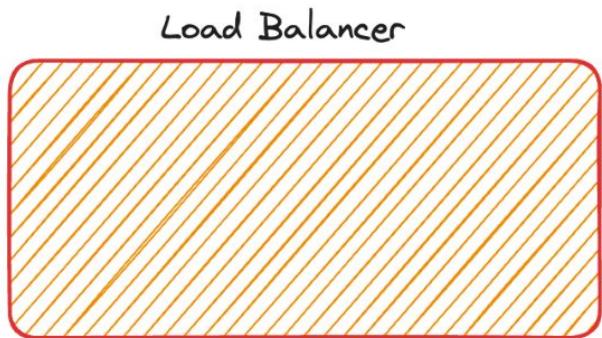










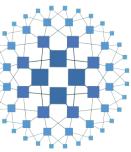




# Why WebSockets are difficult to scale

- *Hence, one client can request to multiple HTTP servers and the client state need not be stored*
- *Millions of clients can be served independently across multiple HTTP servers*
- *Therefore, HTTP servers are horizontally scalable*





# Why WebSockets are difficult to scale

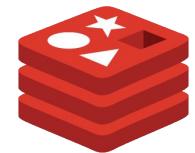
- *However, in case of WebSocket connection, one client is binded to a single WebSocket server*
- *Loss of connection implies loss of state*
- *Therefore, one client ideally binds to one WebSocket server and scaling horizontally cannot replicate states across the servers*





# How to achieve a *synchronised* stateful connection ?

Load Balancer  
Publisher - Subscriber

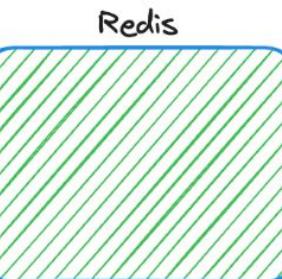
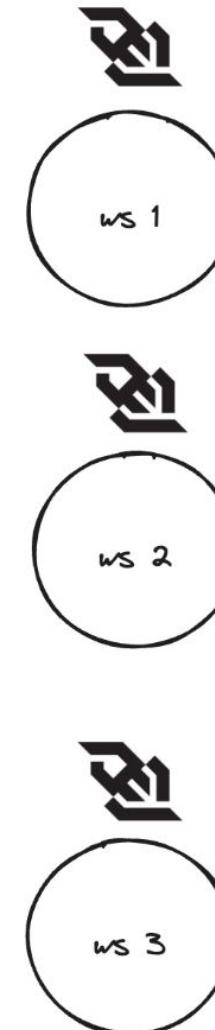
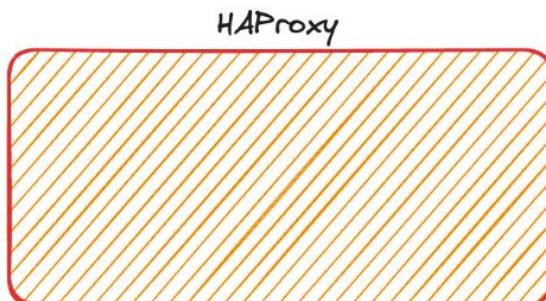




# The Architecture



HAProxy acts as a reverse proxy, load balancer that supports ws:// distributing the incoming load across multiple WebSocket servers that can be independently scaled horizontally

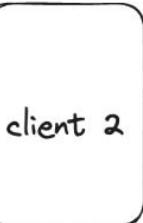


Redis acts as a Publisher - Subscriber service. Each WebSocket sever subscribe to a common channel. Any new updates by a publisher is fired to all the subscribers

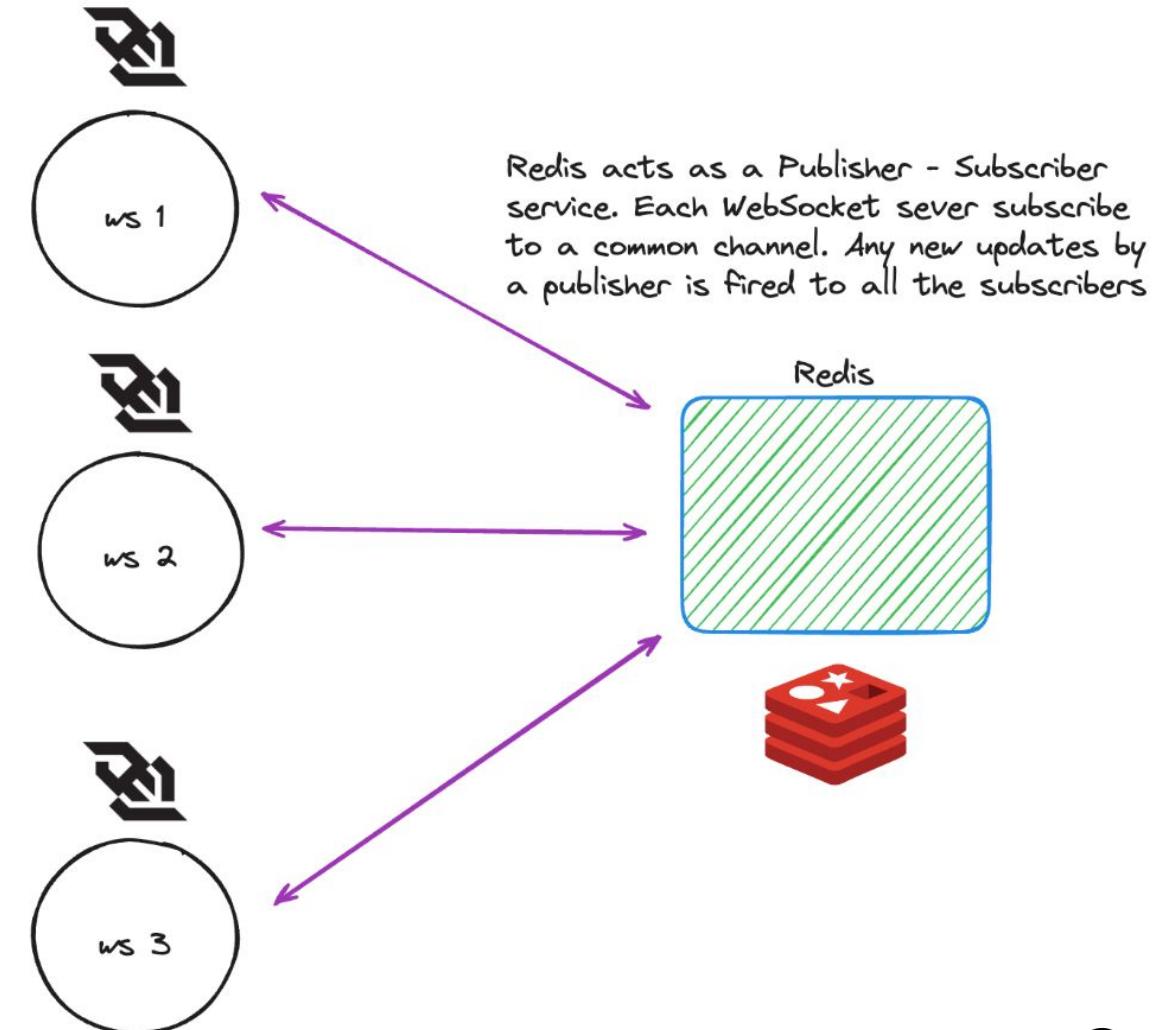
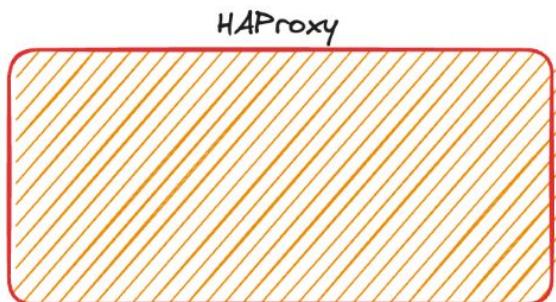




# The Architecture

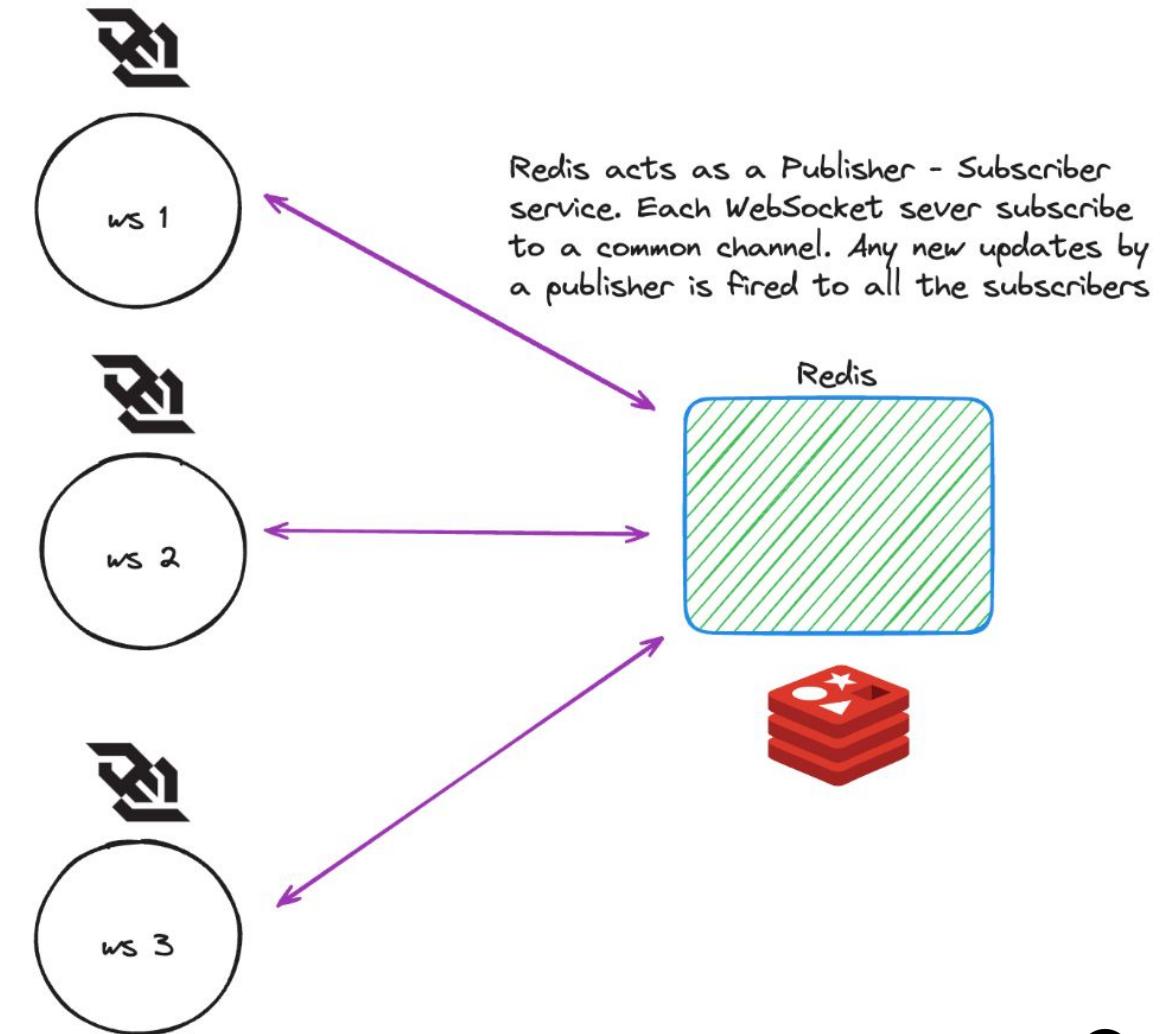
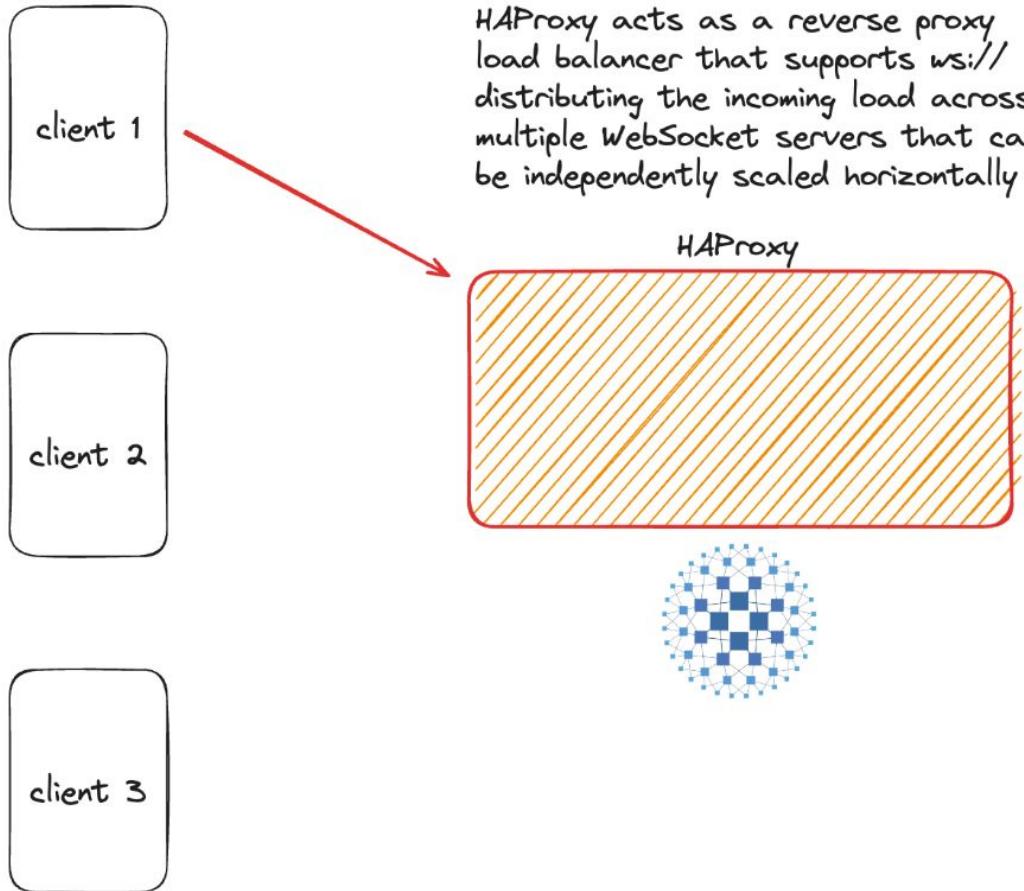


HAProxy acts as a reverse proxy, load balancer that supports ws:// distributing the incoming load across multiple WebSocket servers that can be independently scaled horizontally





# The Architecture

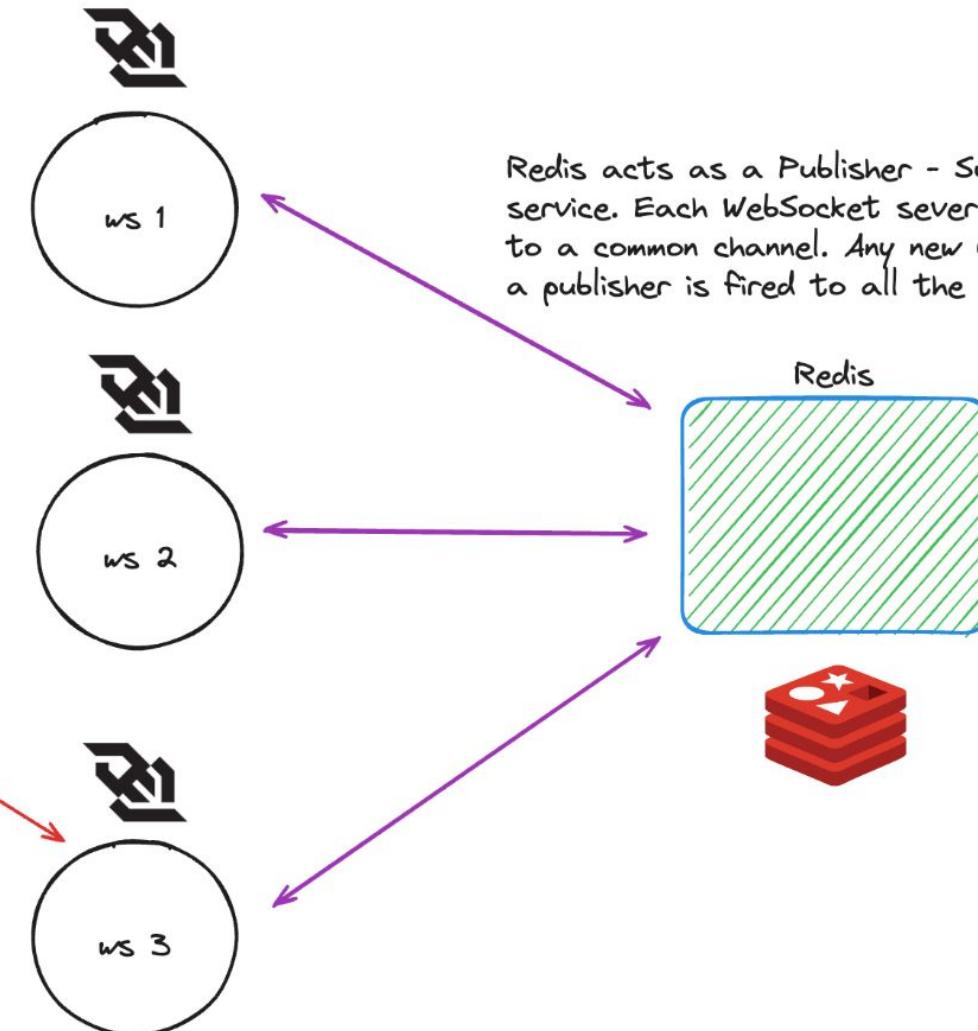
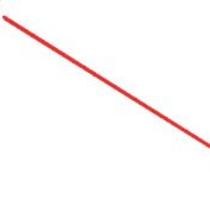
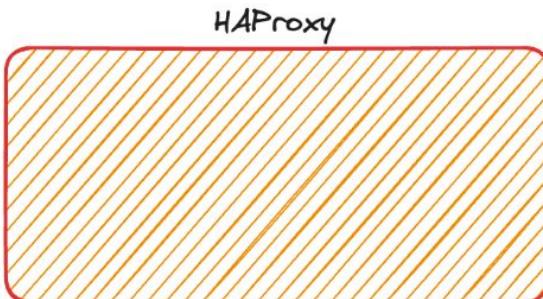




# The Architecture



HAProxy acts as a reverse proxy, load balancer that supports ws:// distributing the incoming load across multiple WebSocket servers that can be independently scaled horizontally

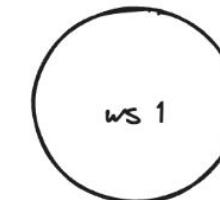
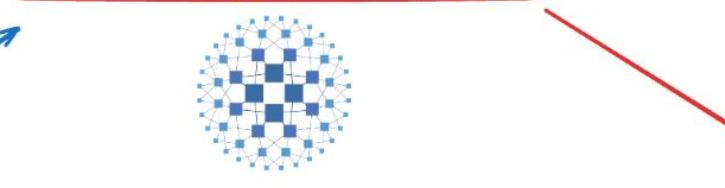




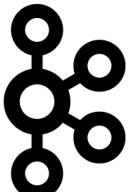
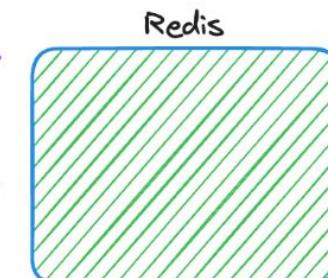
# The Architecture



HAProxy acts as a reverse proxy, load balancer that supports ws:// distributing the incoming load across multiple WebSocket servers that can be independently scaled horizontally

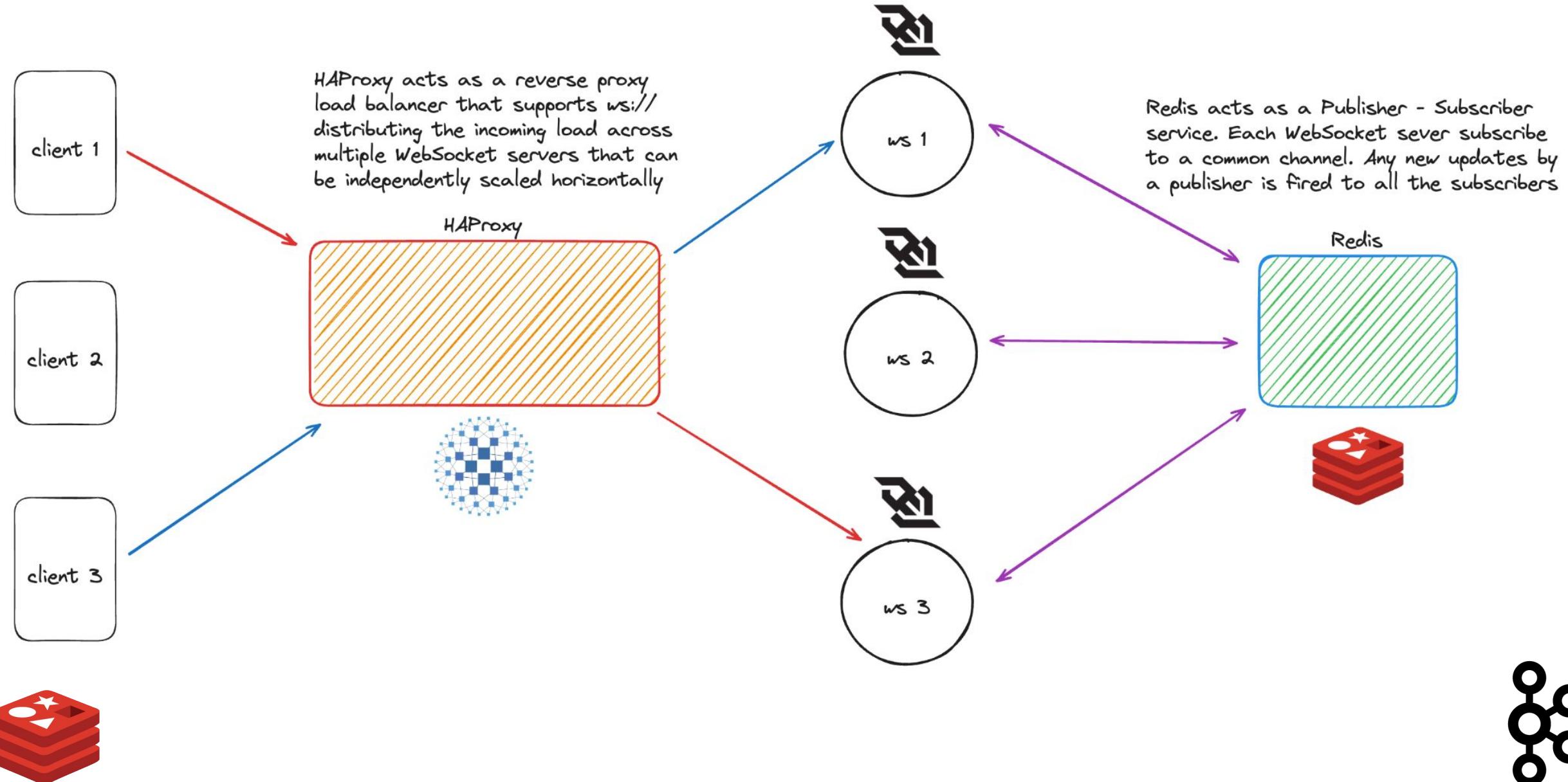
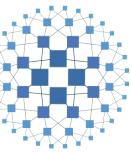


Redis acts as a Publisher - Subscriber service. Each WebSocket sever subscribe to a common channel. Any new updates by a publisher is fired to all the subscribers



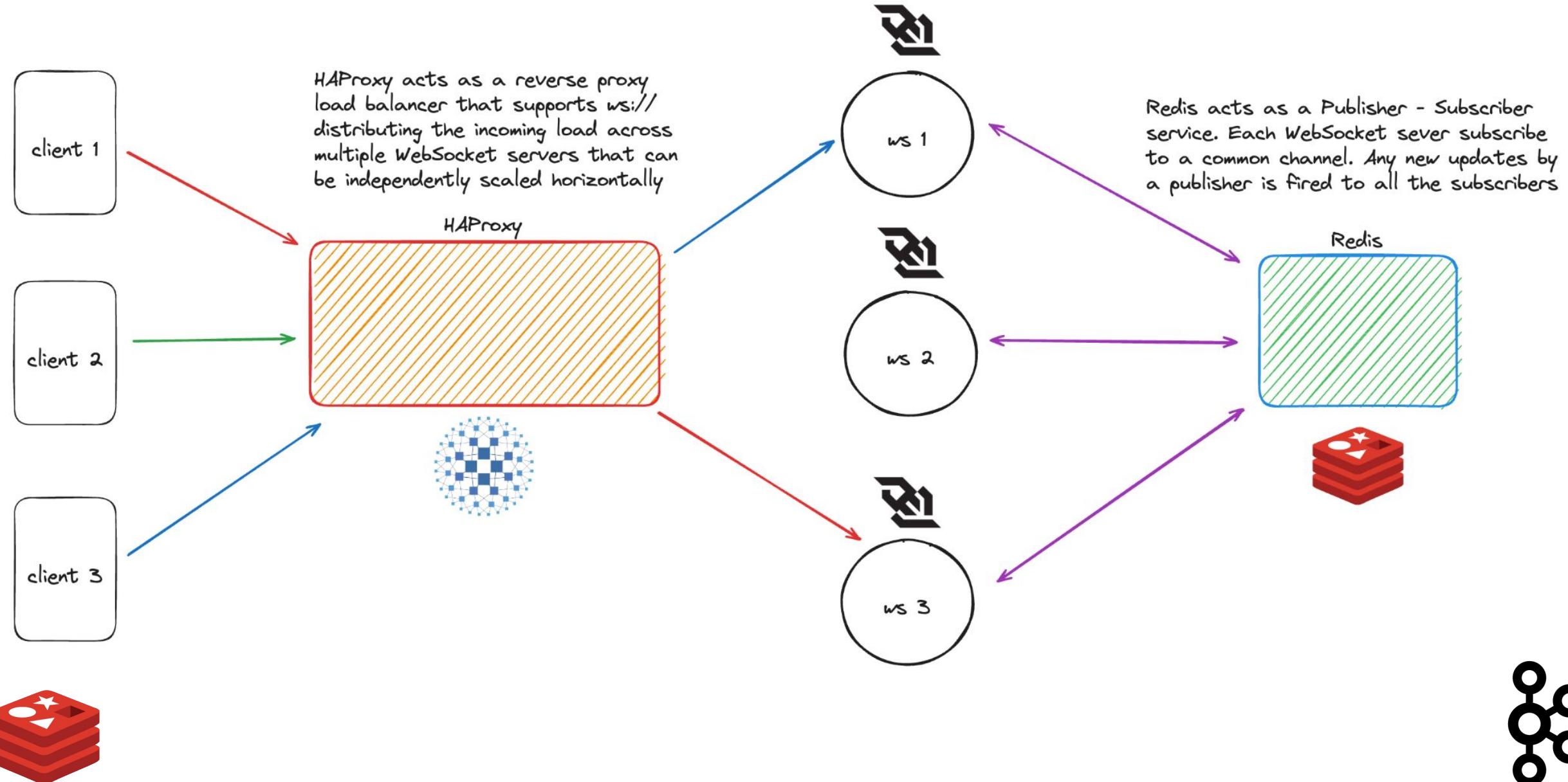
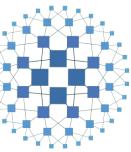


# The Architecture



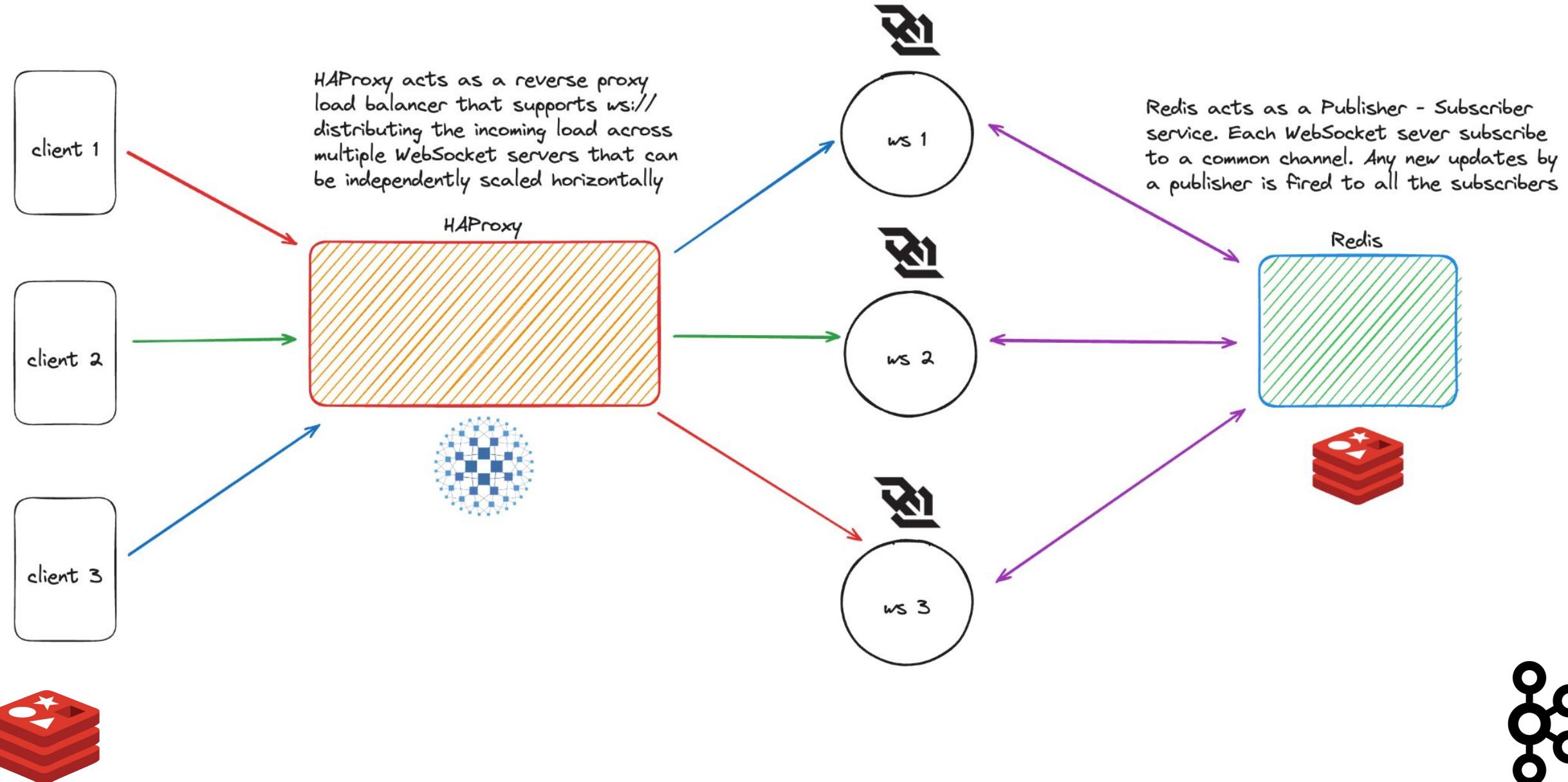


# The Architecture



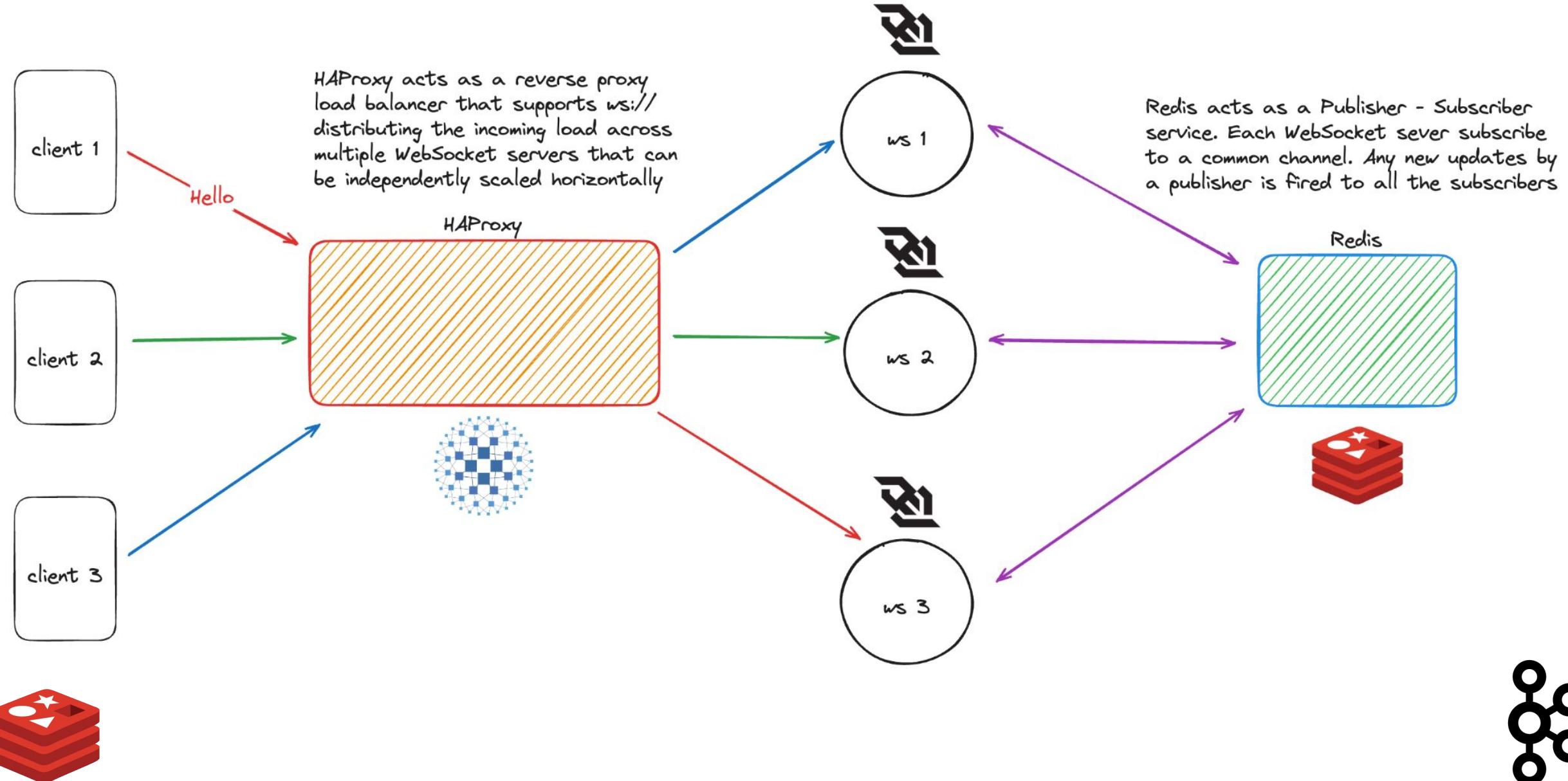


# The Architecture



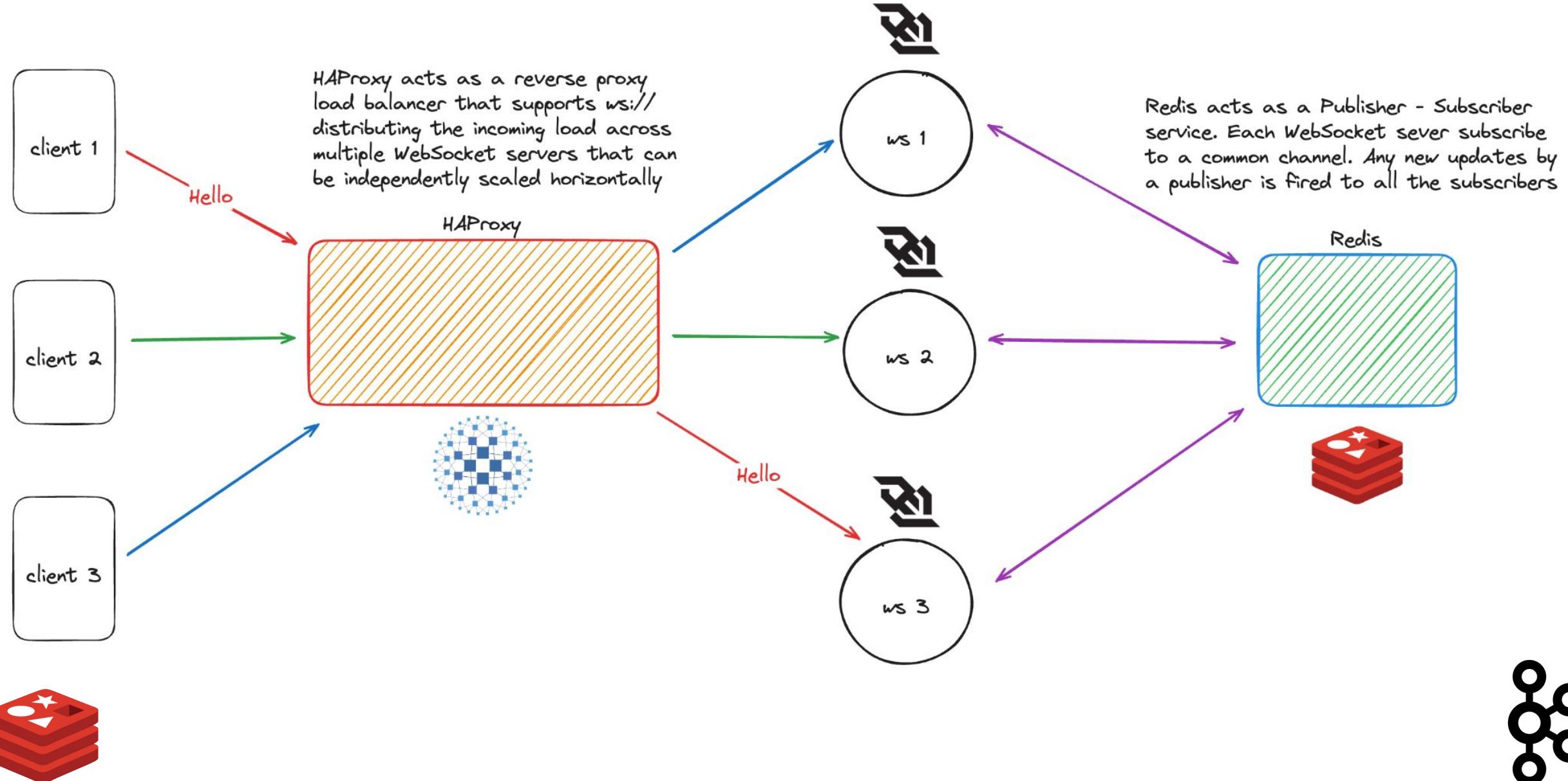


# The Architecture



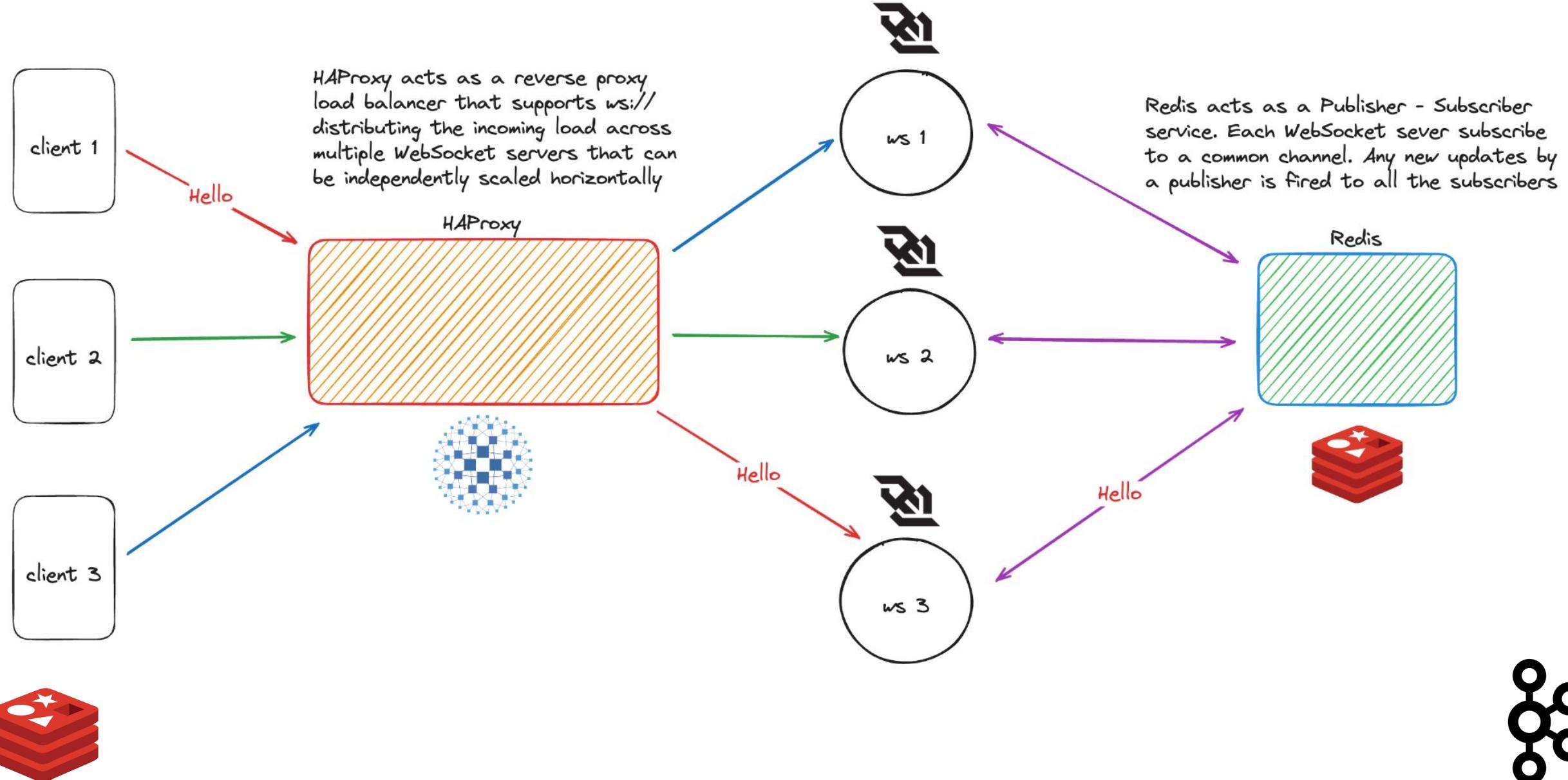


# The Architecture



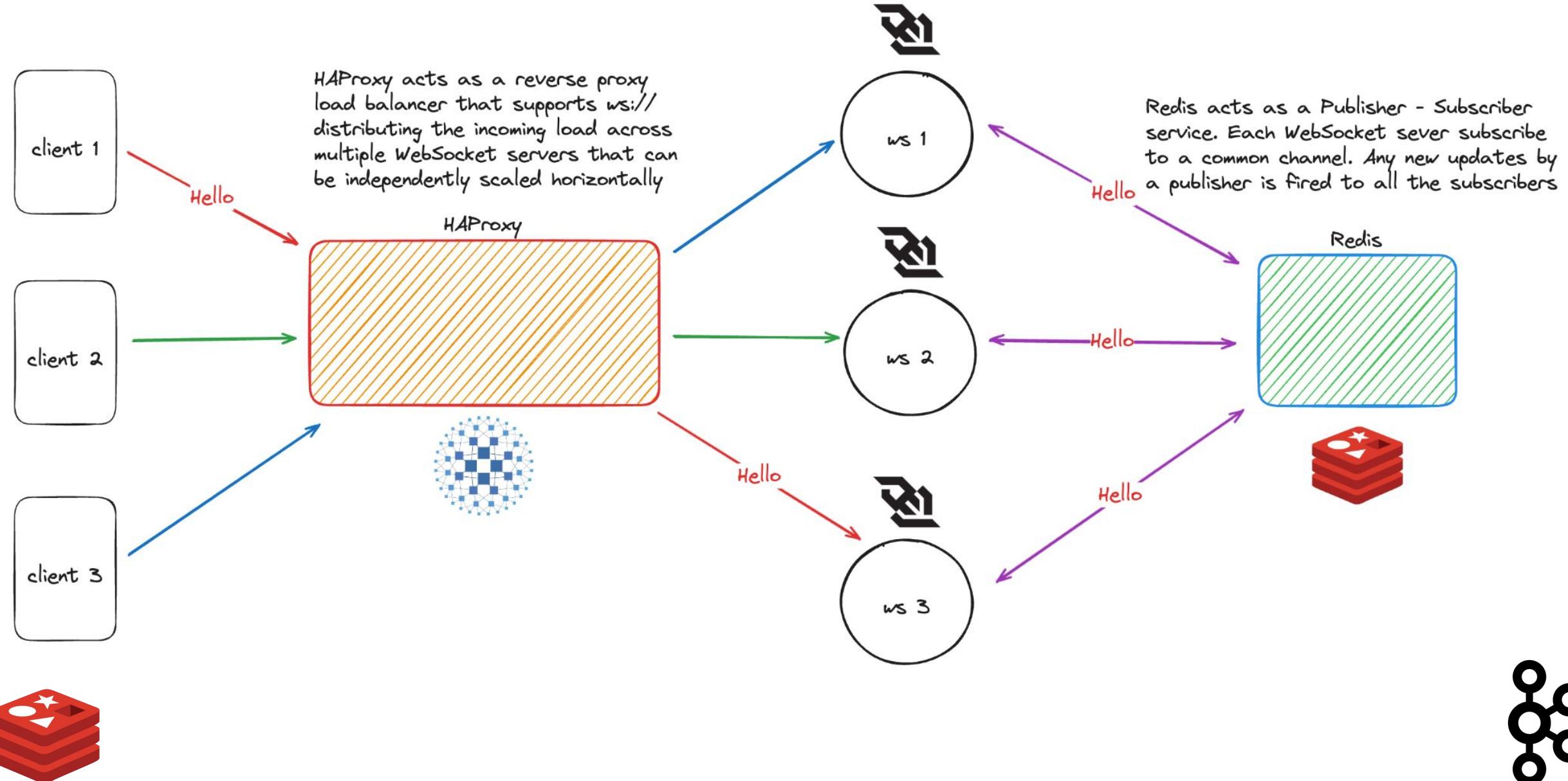


# The Architecture



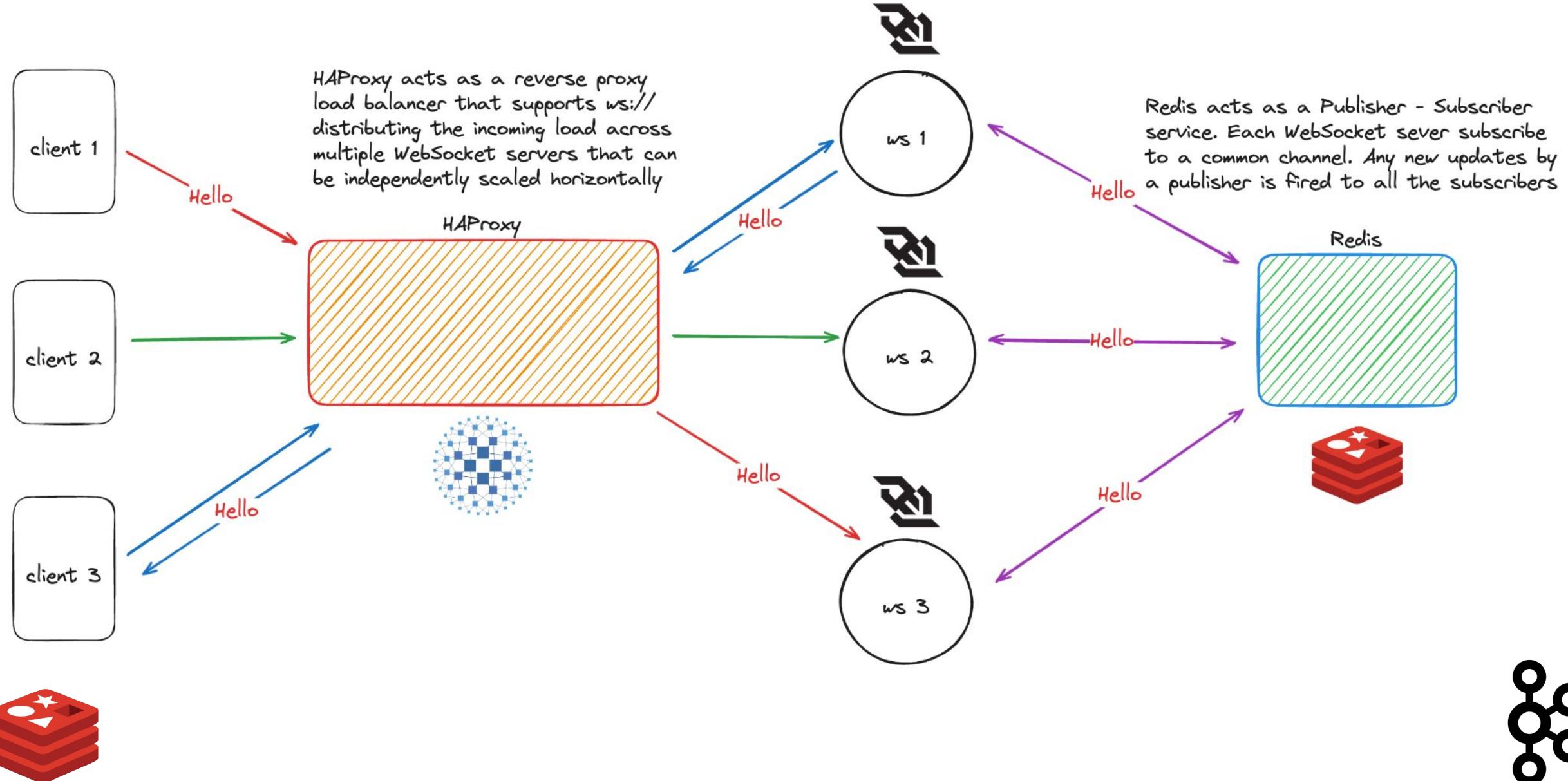


# The Architecture



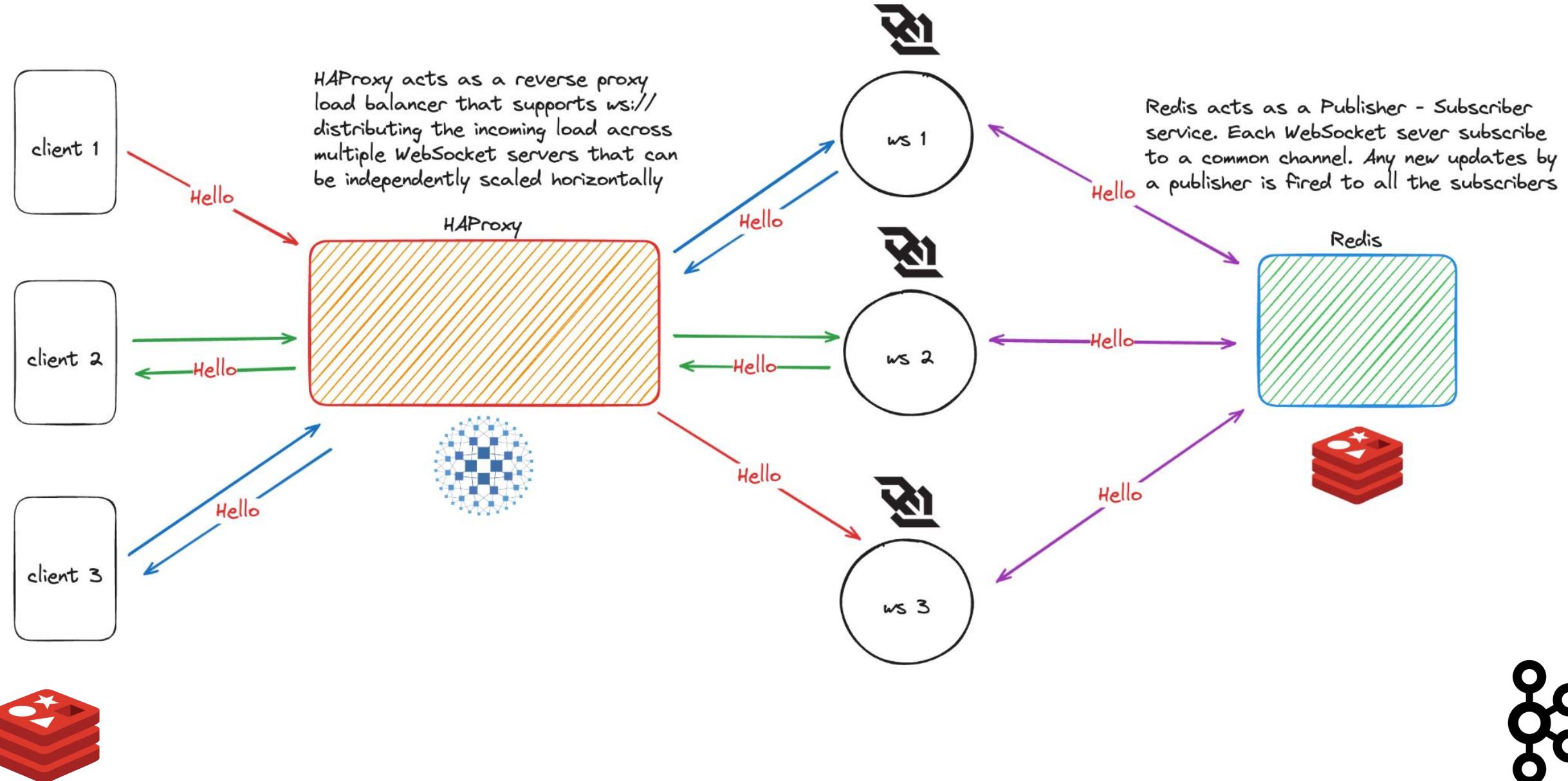
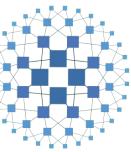


# The Architecture



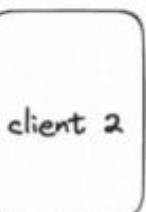


# The Architecture

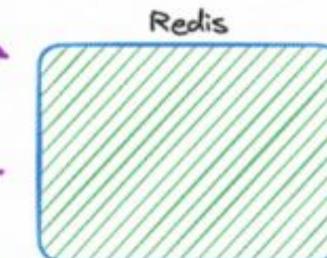
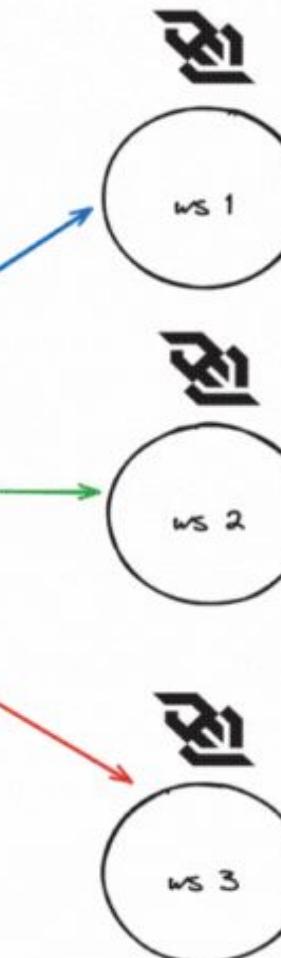
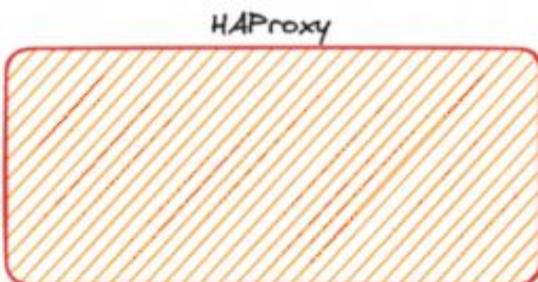




# The Architecture



HAProxy acts as a reverse proxy load balancer that supports ws:// distributing the incoming load across multiple WebSocket servers that can be independently scaled horizontally



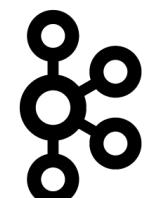
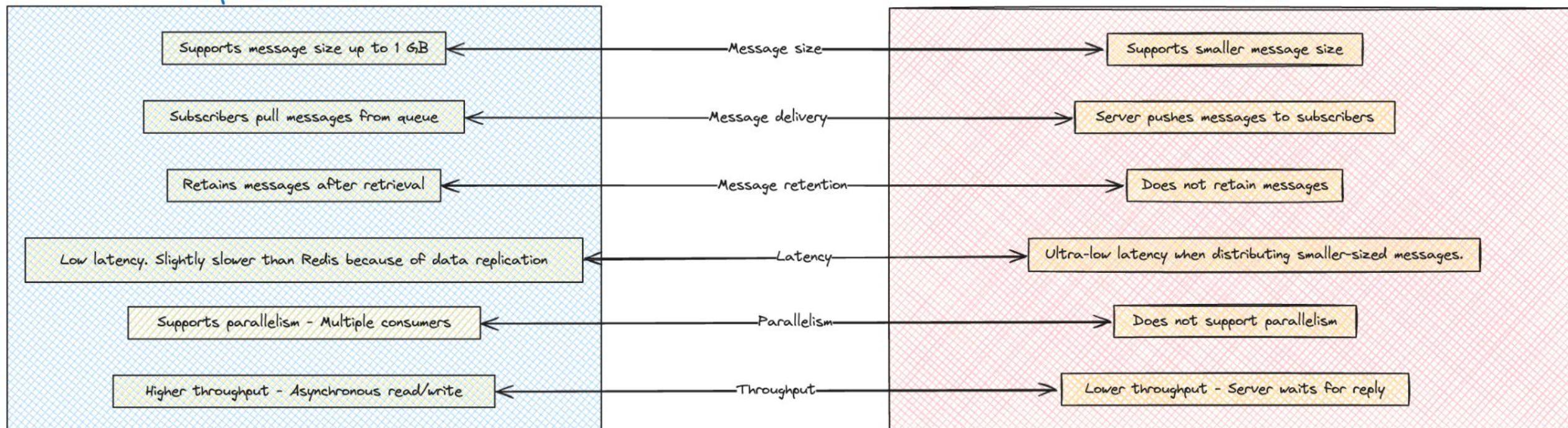
Redis acts as a Publisher - Subscriber service. Each WebSocket sever subscribe to a common channel. Any new updates by a publisher is fired to all the subscribers





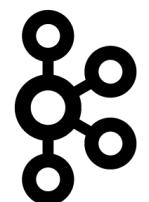
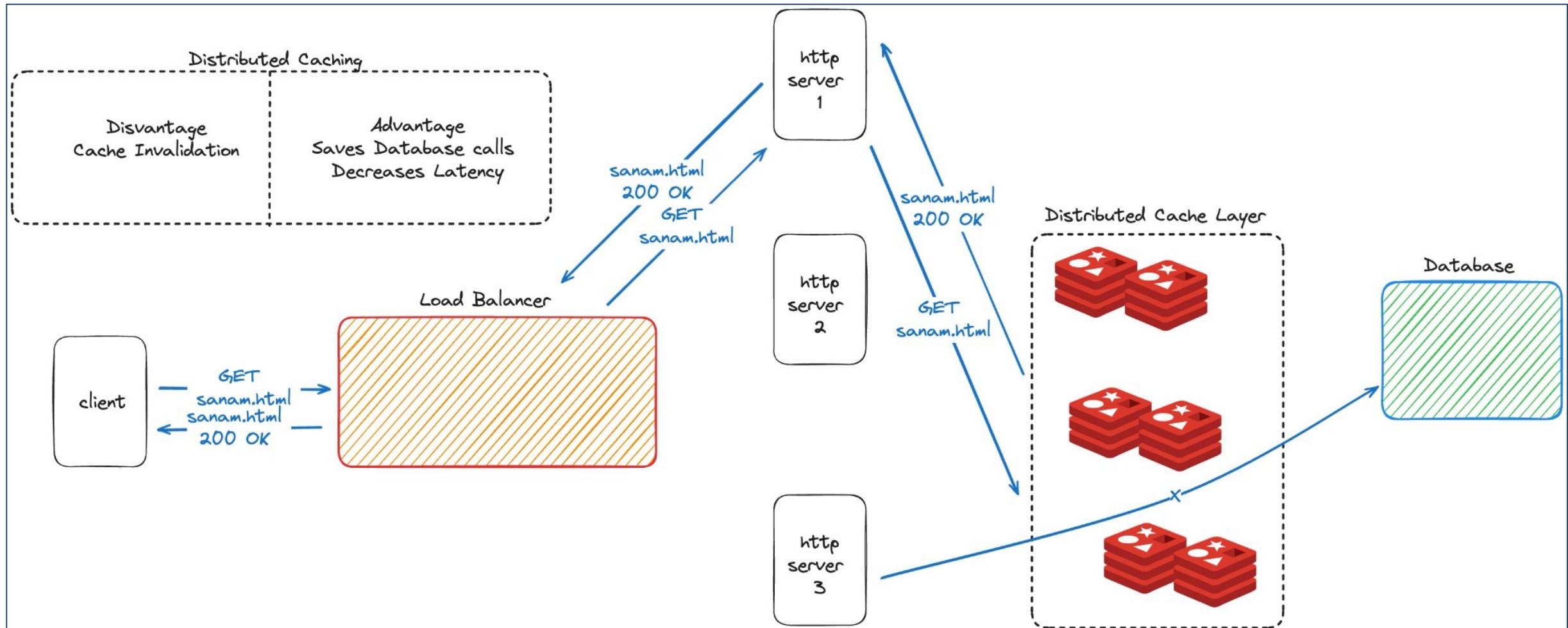
# Redis v/s Kafka

(Deciding which Pub-Sub to use to when ?)





# Distributed Caching





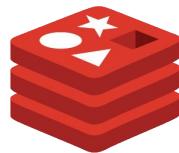
# Conclusion





# Conclusions

- A single WebSocket connection is blocking in nature and highly resource intensive
- WebSocket cannot be scaled vertically due to hardware limitations
- Horizontal scaling is only possible across multiple WebSockets if the incoming load can be balanced to multiple servers using a *Reverse Proxy* and all of them maintain a common state through a *Pub-Sub* mechanism





# Limitation





# Limitations

- Maintaining a common state across all the WebSockets is hard to implement
- In case of frequent state updates and concurrent access to shared resources may lead to race conditions
- Employing synchronization mechanisms such as mutexes, semaphores, or atomic operations to coordinate access to shared resources need to be employed when dealing with distributed systems leading to higher latency





# References

- [https://github.com/hnasr/javascript\\_playground/  
tree/master/ws-live-chat-system](https://github.com/hnasr/javascript_playground/tree/master/ws-live-chat-system)
- <https://datatracker.ietf.org/doc/html/rfc6455>
- <https://www.haproxy.org>
- [https://redis.io/docs/latest/develop/interact/  
pubsub](https://redis.io/docs/latest/develop/interact/pubsub)





# Thank you

<https://github.com/akarmanya/wscale>

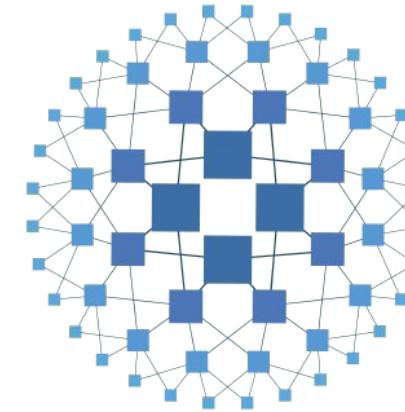
Manas Pratim Biswas (@sanam2405)

Roll - 002011001025      UG - IV

Information Technology

Jadavpur University





# Questions ?

