CSCI-1680
DNS

Nick DeMarinis

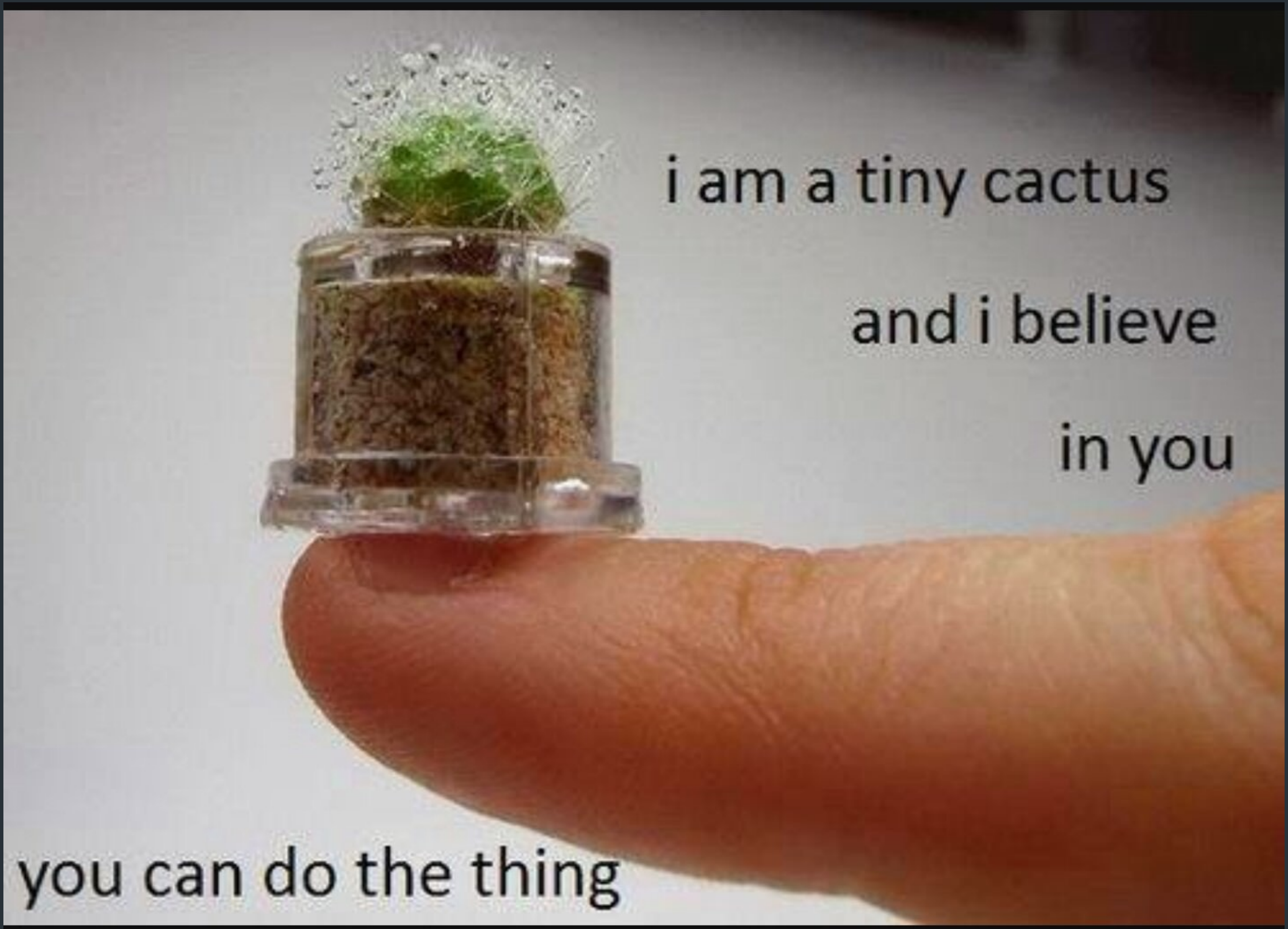
Based partly on lecture notes by Rodrigo Fonseca, Scott Shenker and John Jannotti

Administrivia

- TCP due this Friday (11/22)
  - See Ed for latest on bugs with reference
  - Look for an update on testing resources, SRC component (due after Thanksgiving)
  - It's going to be okay.

- Grading update, final project info out later this week

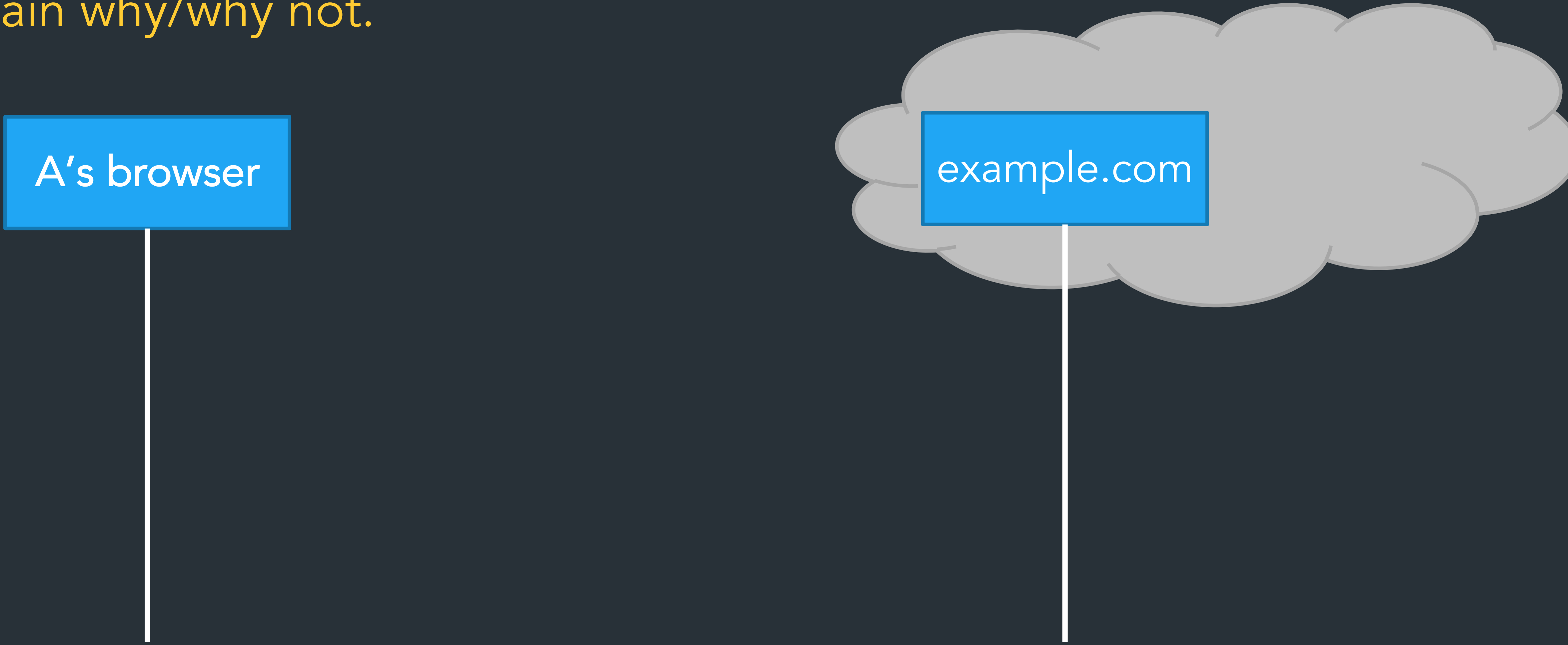
Breathe



Warmup

If a client A makes two separate HTTP requests to example.com, does the server know both requests came from A?

Explain why/why not.



Reverse proxy: proxy server that lives somewhere in the network, transparent to the client

A simple reverse proxy

```
<VirtualHost *:443>
    ServerName test.cs1680.systems
    ErrorLog "/var/log/httpd/test-error_log"
    CustomLog "/var/log/httpd/test-access_log" combined

    ProxyPass "/" "http://127.0.0.1:9999/"
    ProxyPassReverse "/" "http://127.0.0.1:9999/"


  SSLCertificateFile /etc/letsencrypt/live/test.cs1680.systems/fullchain.pem
  SSLCertificateKeyFile /etc/letsencrypt/live/test.cs1680.systems/privkey.pem
  Include /etc/letsencrypt/options-ssl-apache.conf
</VirtualHost>
```

Content Distribution Networks (CDNs)

Companies that specialize in providing caching services (among other things)

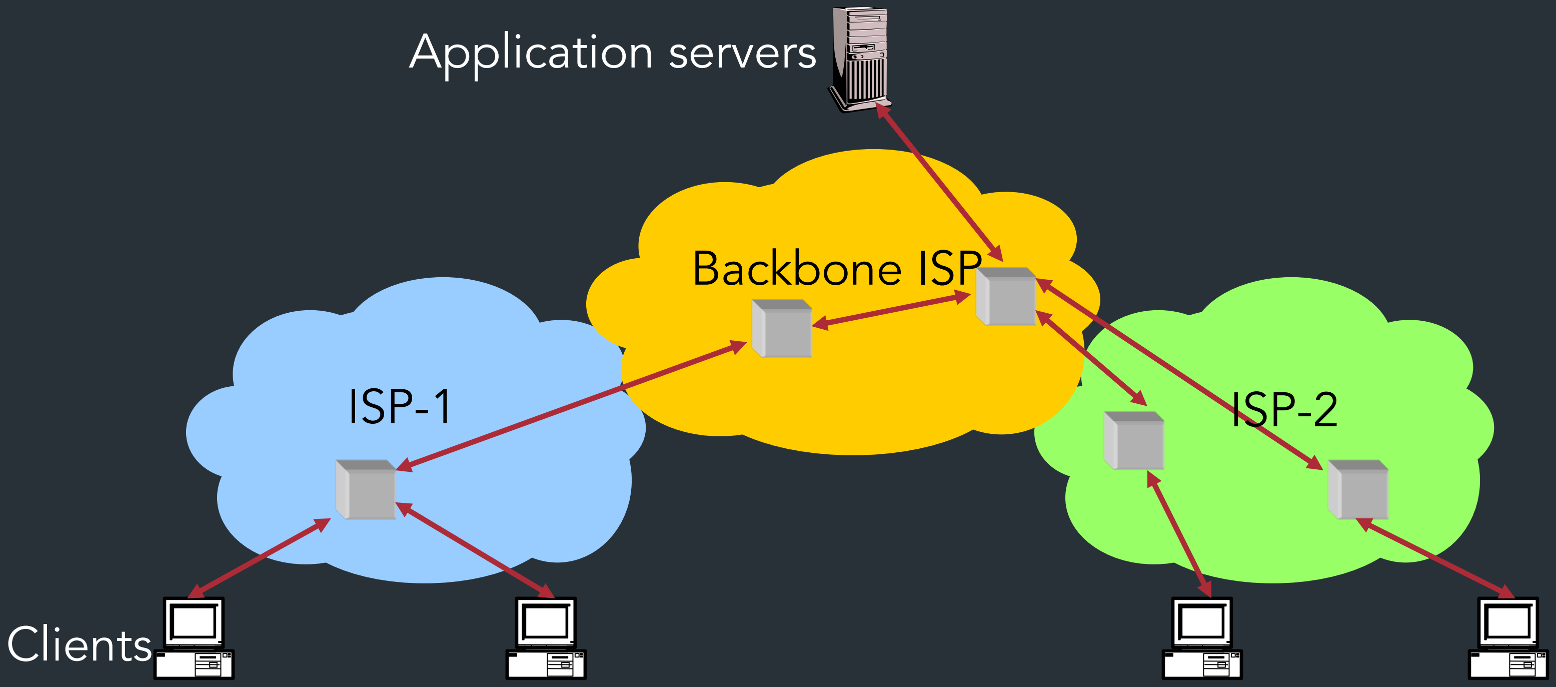
=> Akamai, Cloudflare, …

Content Distribution Networks (CDNs)

Companies that specialize in providing caching services (among other things)

⇒Akamai, Cloudflare, …

- Provides caching throughout network
- Can also do some processing
- Useful for security


Application servers
Backbone ISP
ISP-1
ISP-2
Clients

CDNs for securing traffic

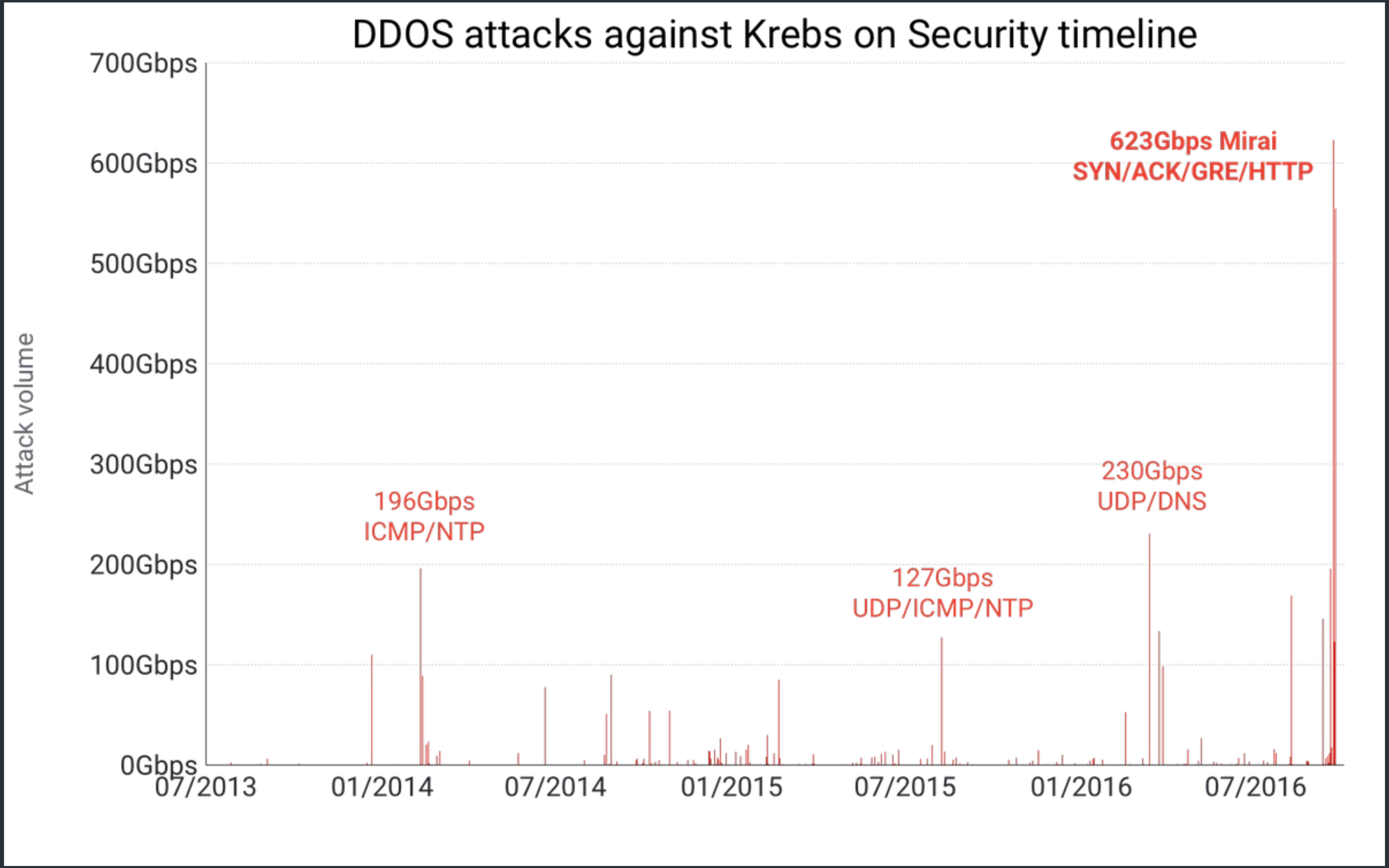
DDoS attacks: overwhelm a target host/network with packets, denying resources for legitimate traffic

DDoS attacks: overwhelm a target host/network with packets, denying resources for legitimate traffic

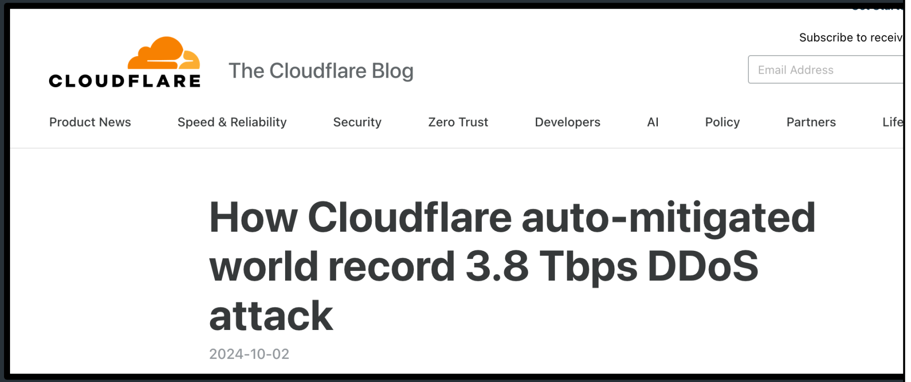
=> Often performed by "botnets" of compromised devices

=> Attack traffic can take many forms: lots of SYNs, DNS requests, exploiting bugs in protocols, …

$\Rightarrow$ Want to learn more? CS 1660.


DDOS attacks against Krebs on Security timeline
700Gbps
600Gbps
500Gbps
400Gbps
300Gbps
200Gbps
100Gbps
0Gbps
Attack volume
07/2013
01/2014
07/2014
01/2015
07/2015
01/2016
07/2016
196Gbps ICMP/NTP
127Gbps UDP/ICMP/NTP
230Gbps UDP/DNS
623Gbps Mirai SYN/ACK/GRE/HTTP

DDoS mitigation via CDN



Subscribe to receiv

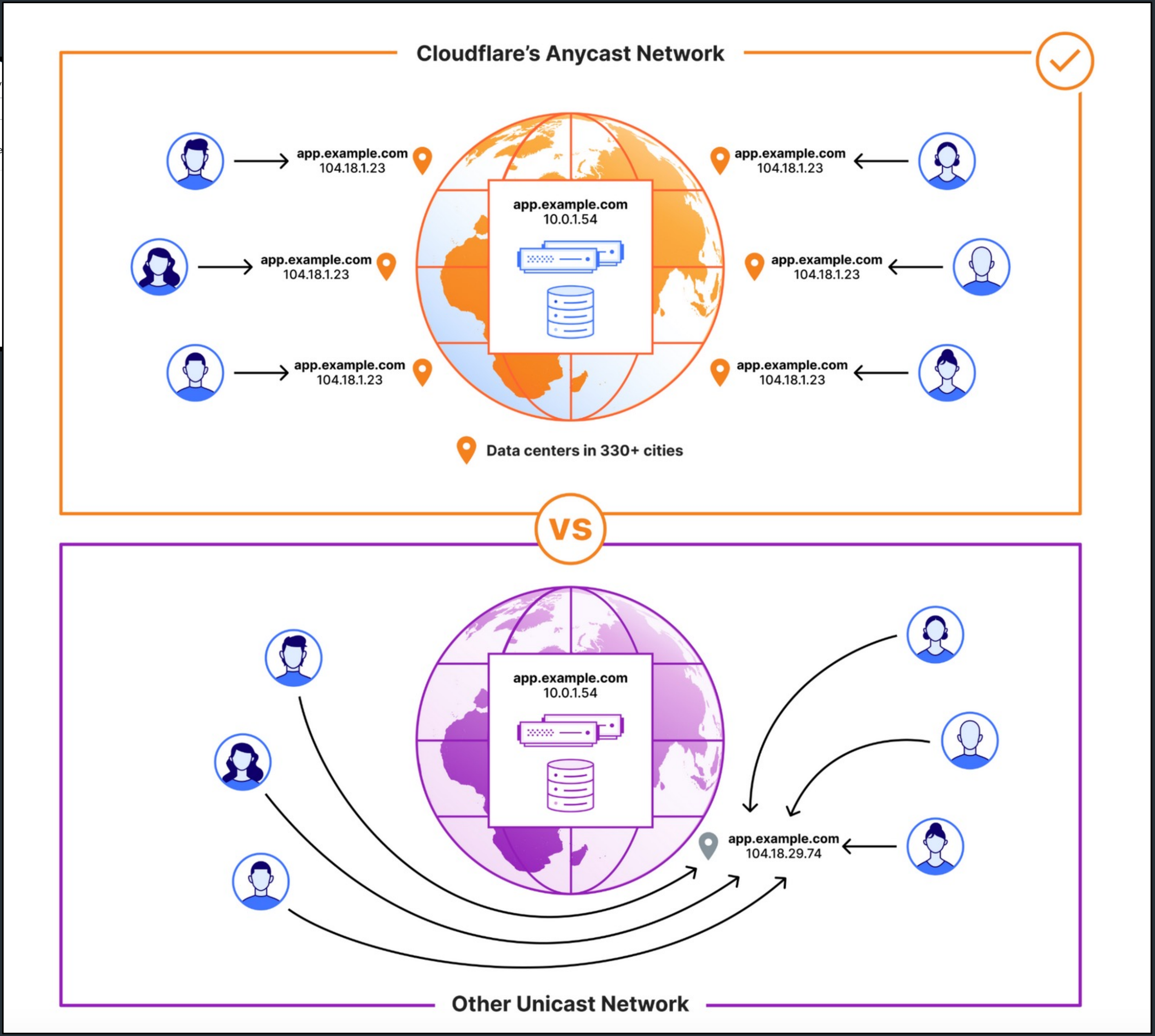
CLOUDFLARE The Cloudflare Blog

Email Address

Product News Speed & Reliability Security Zero Trust Developers AI Policy Partners Life

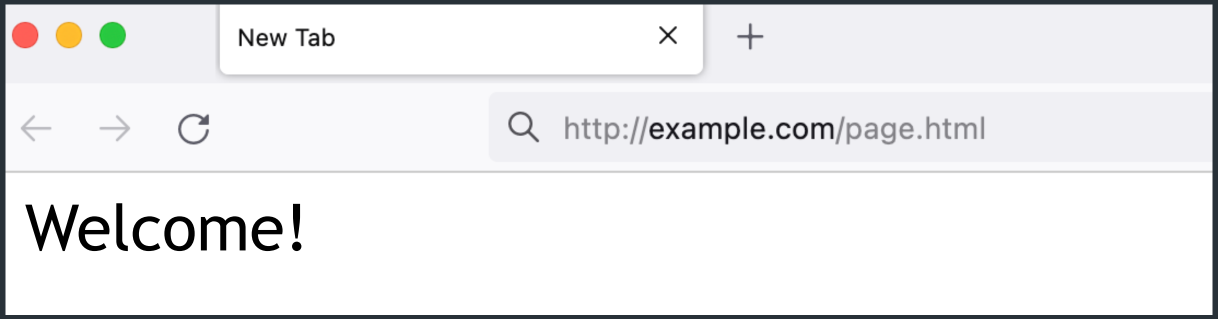
How Cloudflare auto-mitigated world record 3.8 Tbps DDoS attack

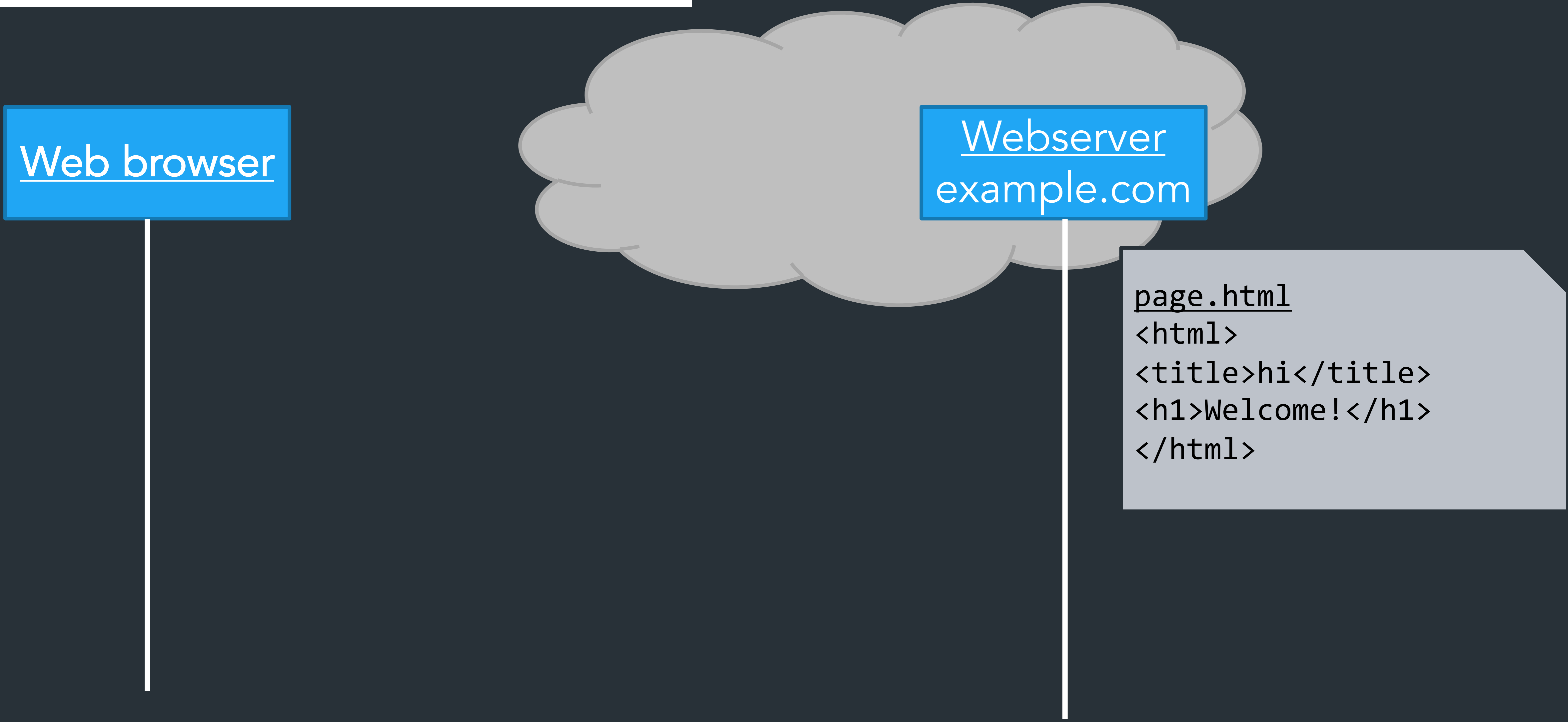
2024-10-02


Cloudflare's Anycast Network
app.example.com 104.18.1.23
app.example.com 104.18.1.23
app.example.com 104.18.1.23
app.example.com 10.0.1.54
app.example.com 104.18.1.23
app.example.com 104.18.1.23
app.example.com 104.18.1.23
Data centers in 330+ cities
VS
app.example.com 10.0.1.54
app.example.com 104.18.29.74
Other Unicast Network

Link

HTTP: what more do we need?

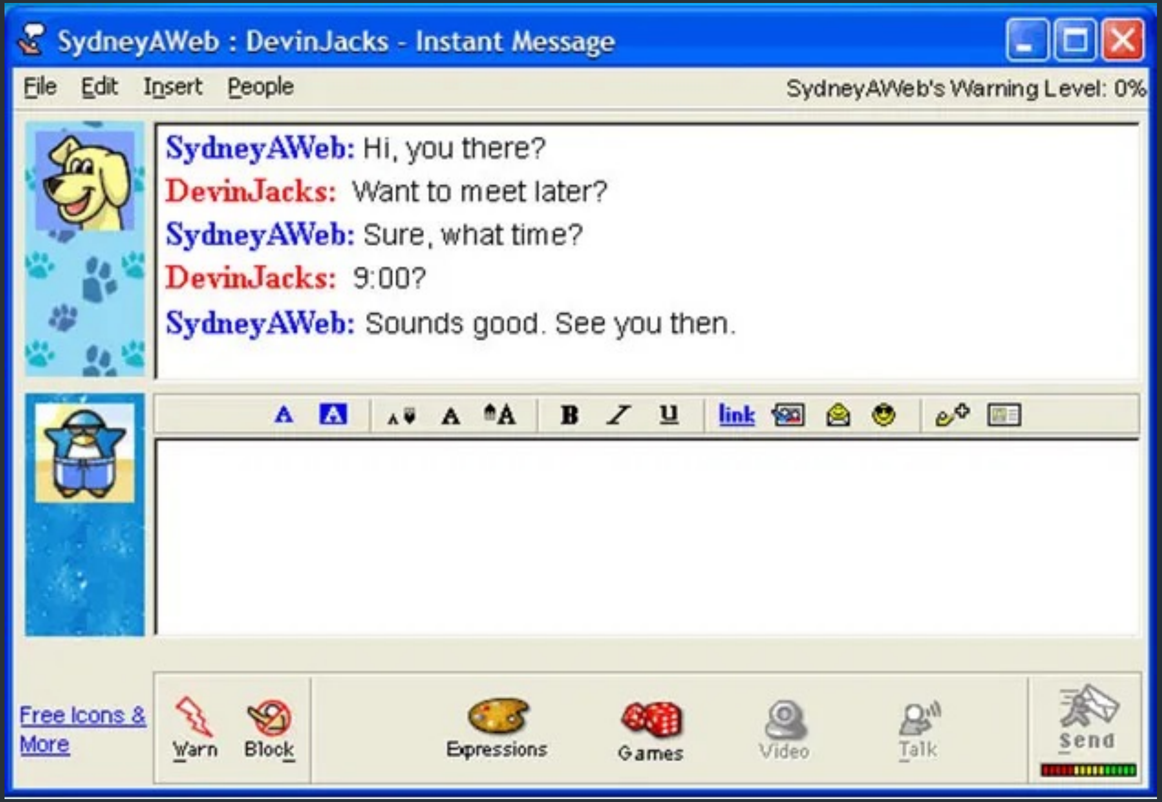




page.html

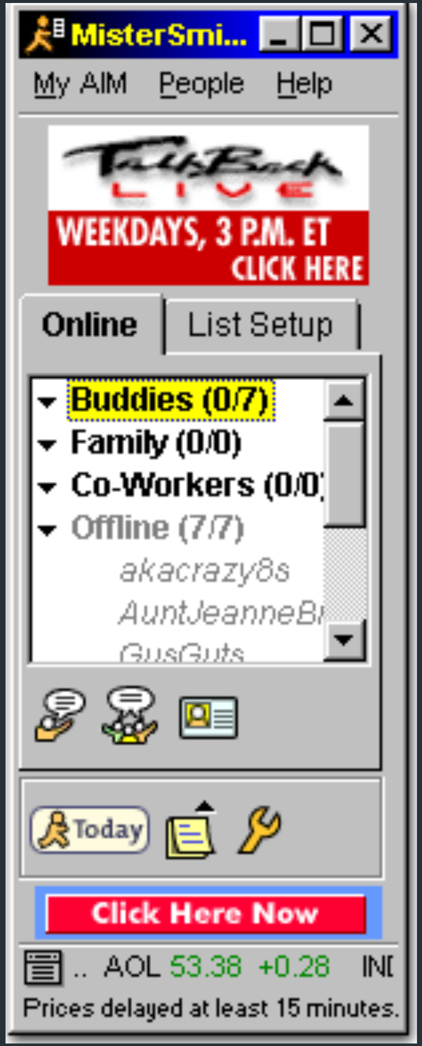
```
<html>
<title>hi</title>
<h1>Welcome!</h1>
</html>
```

Example: Instant Messaging (~2005)


SydneyAWeb : DevinJacks - Instant Message

SydneyAWeb's Warning Level: 0%

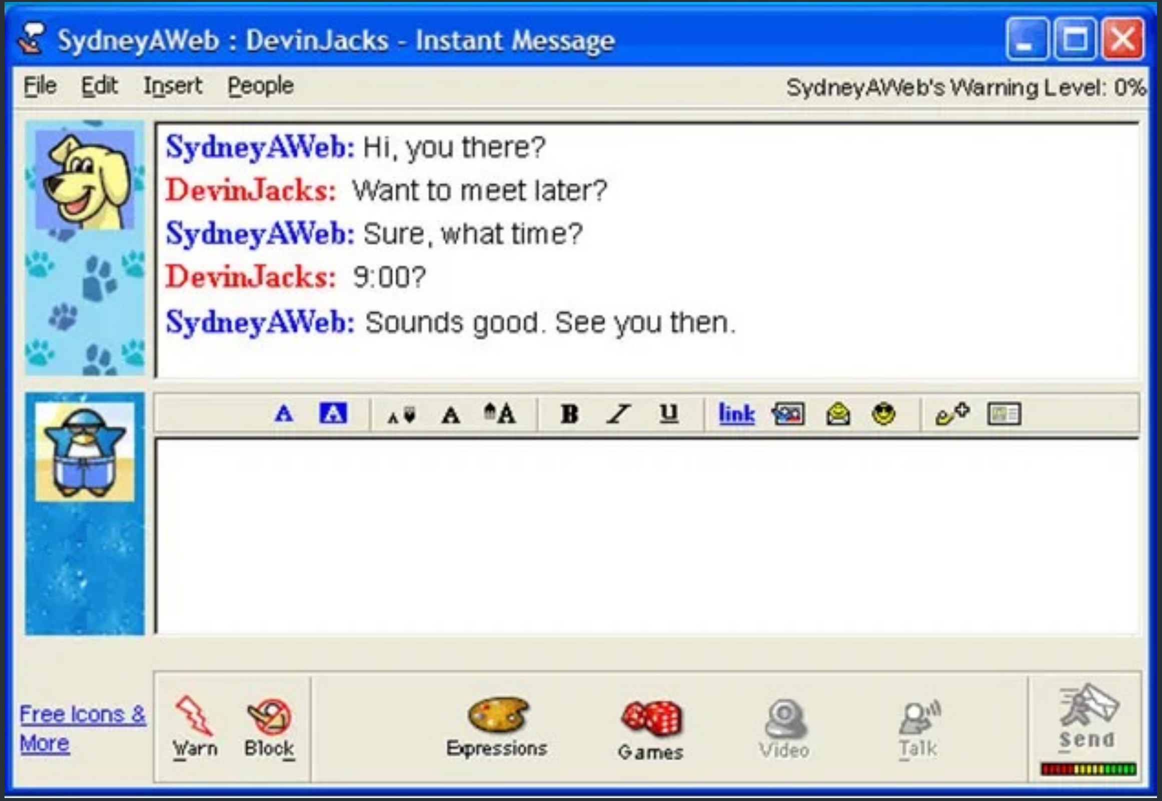
SydneyAWeb: Hi, you there?
DevinJacks: Want to meet later?
SydneyAWeb: Sure, what time?
DevinJacks: 9:00?
SydneyAWeb: Sounds good. See you then.


MisterSmi...

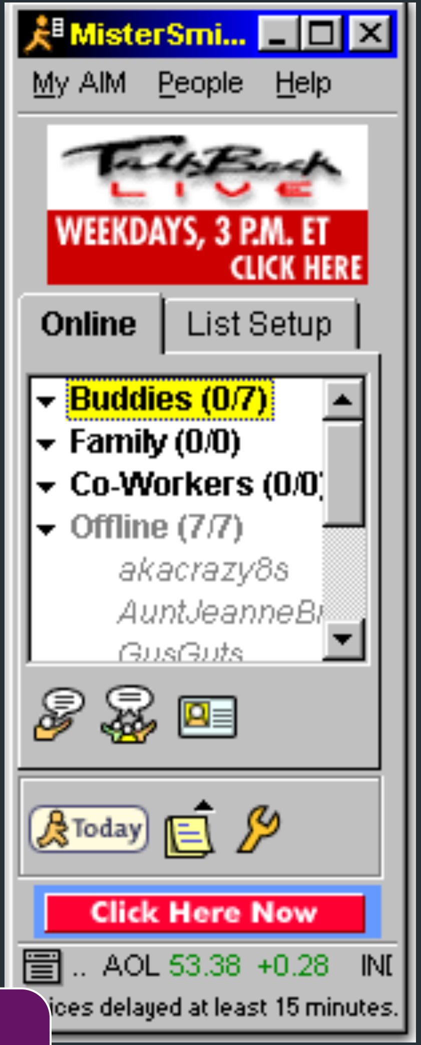
Buddies (0/7)
Family (0/0)
Co-Workers (0/0)
Offline (7/7)
akacrazy8s
AuntJeanneBr
GusGuts

AOL 53.38 +0.28
Prices delayed at least 15 minutes.

Example: Instant Messaging (~2005)

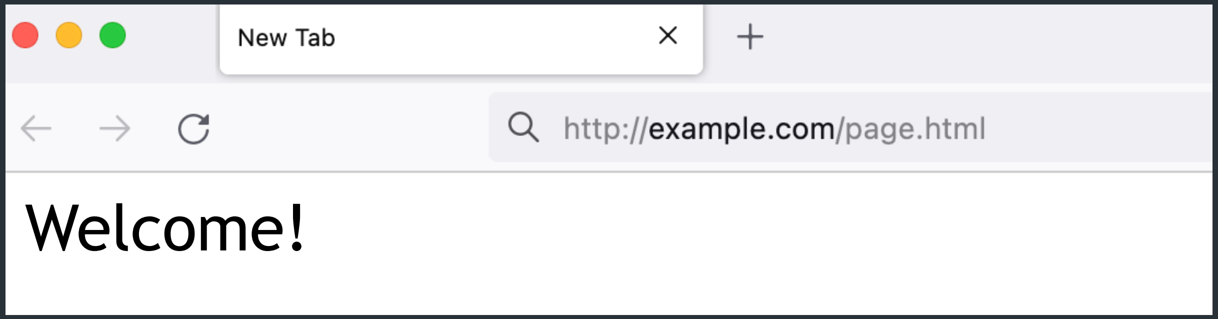


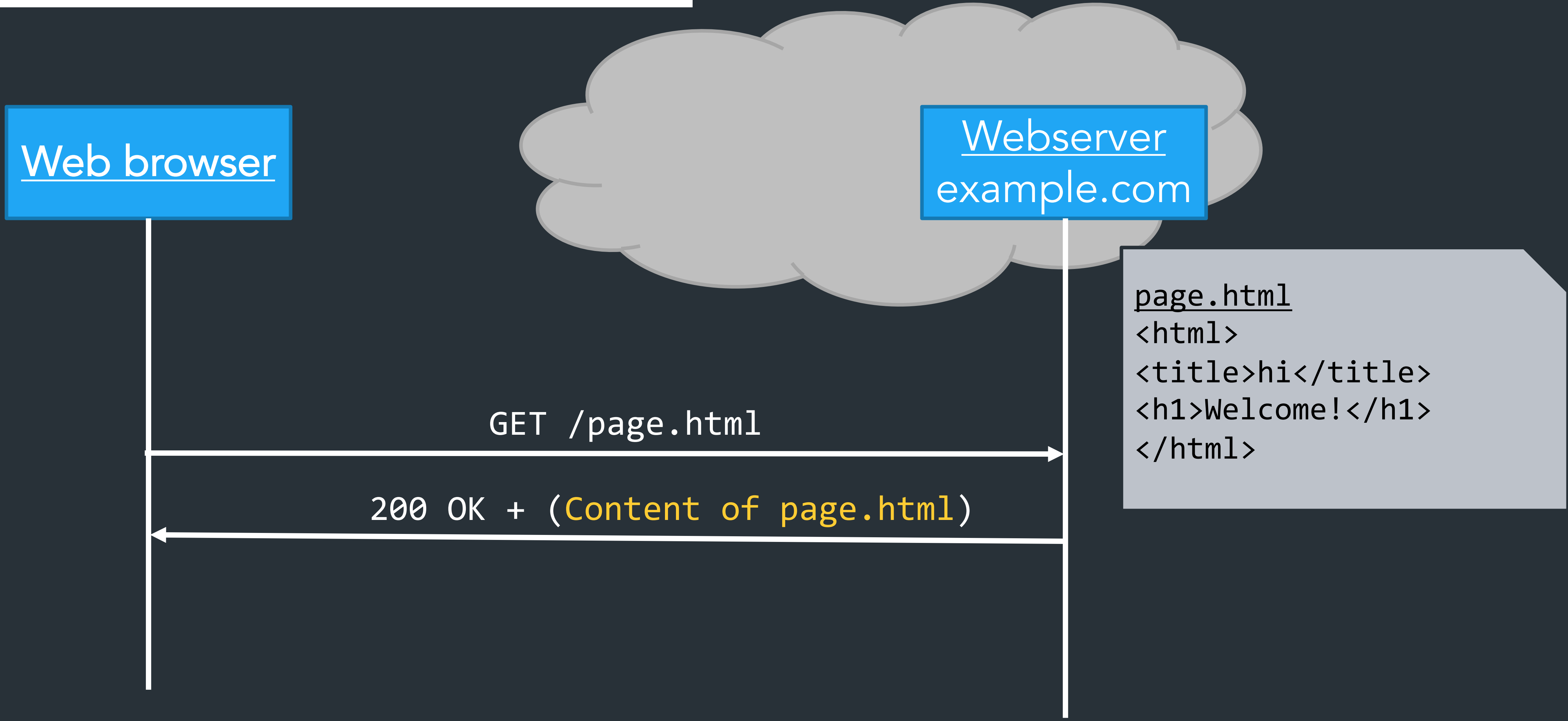
SydneyAWeb: Hi, you there?
DevinJacks: Want to meet later?
SydneyAWeb: Sure, what time?
DevinJacks: 9:00?
SydneyAWeb: Sounds good. See you then.



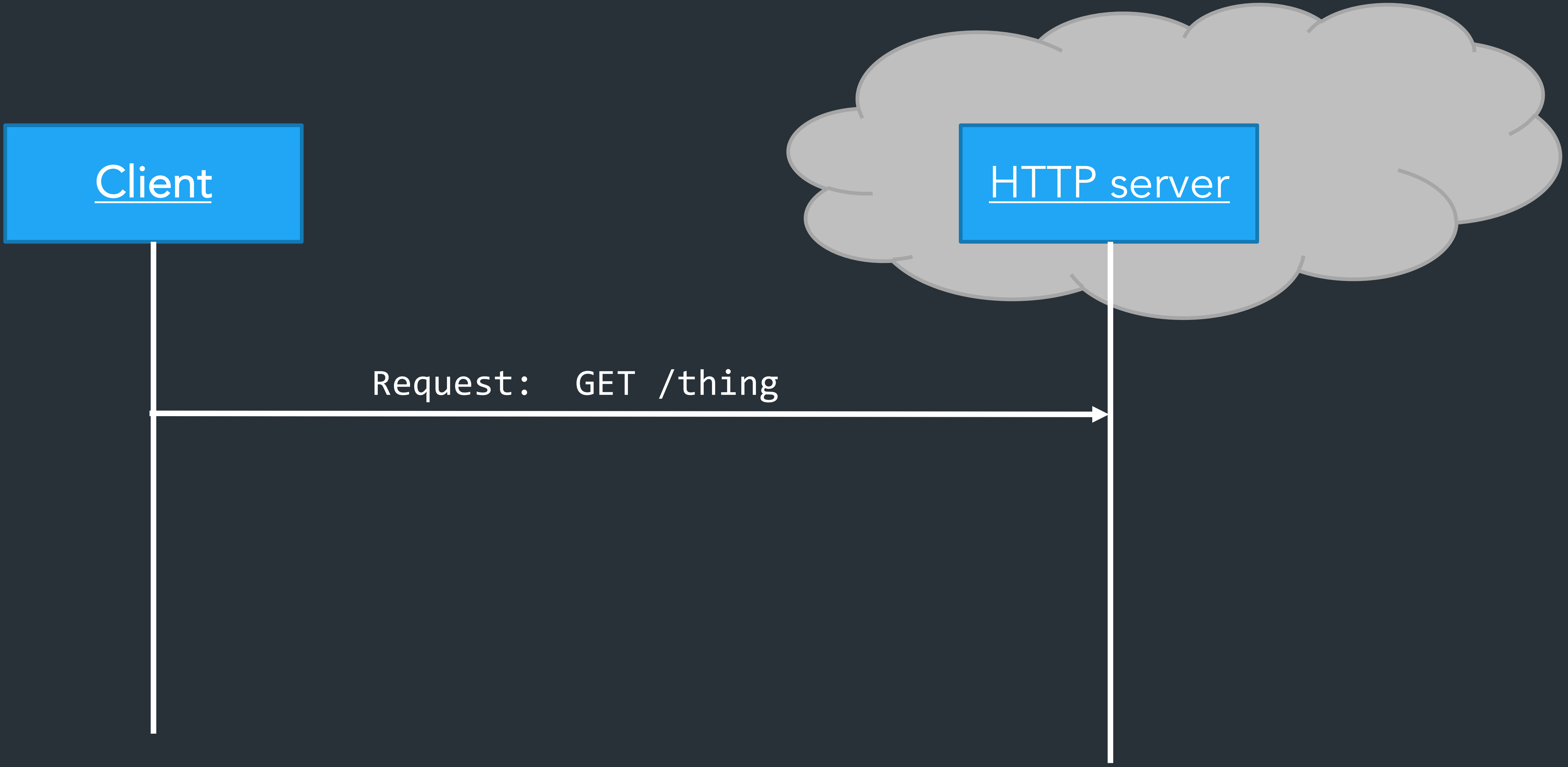
Old chat/IM applications: one TCP connection
=> Can we still do that?

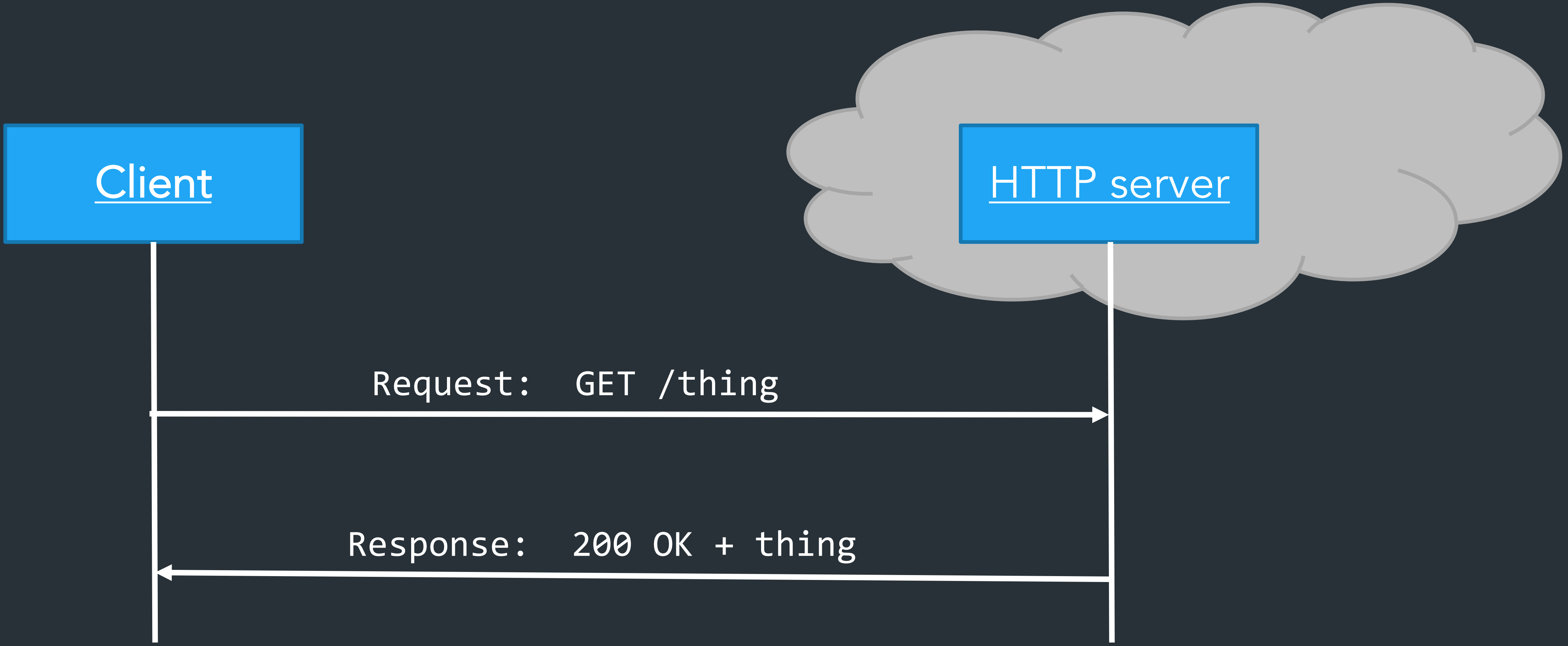
Can we do this with HTTP?

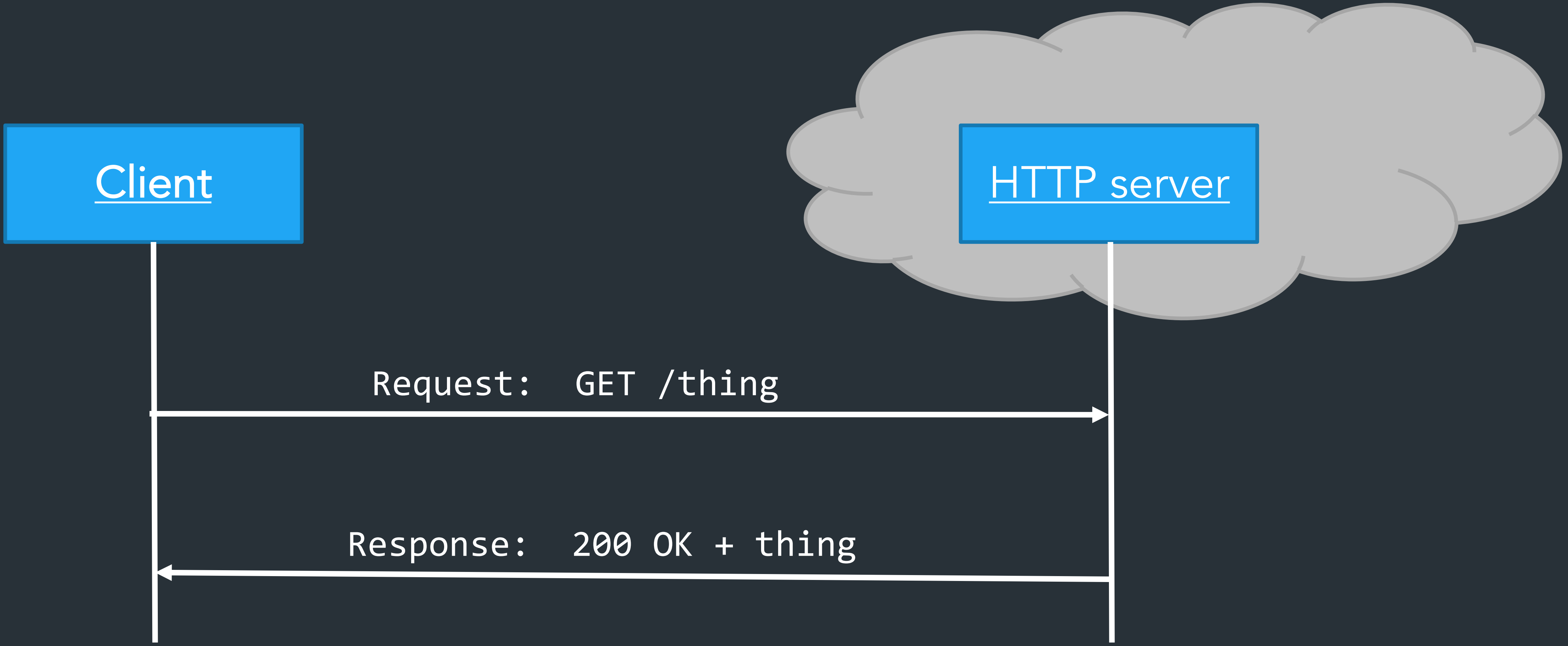




page.html
```
<html>
<title>hi</title>
<h1>Welcome!</h1>
</html>
```


Client
HTTP server
Request: GET /thing

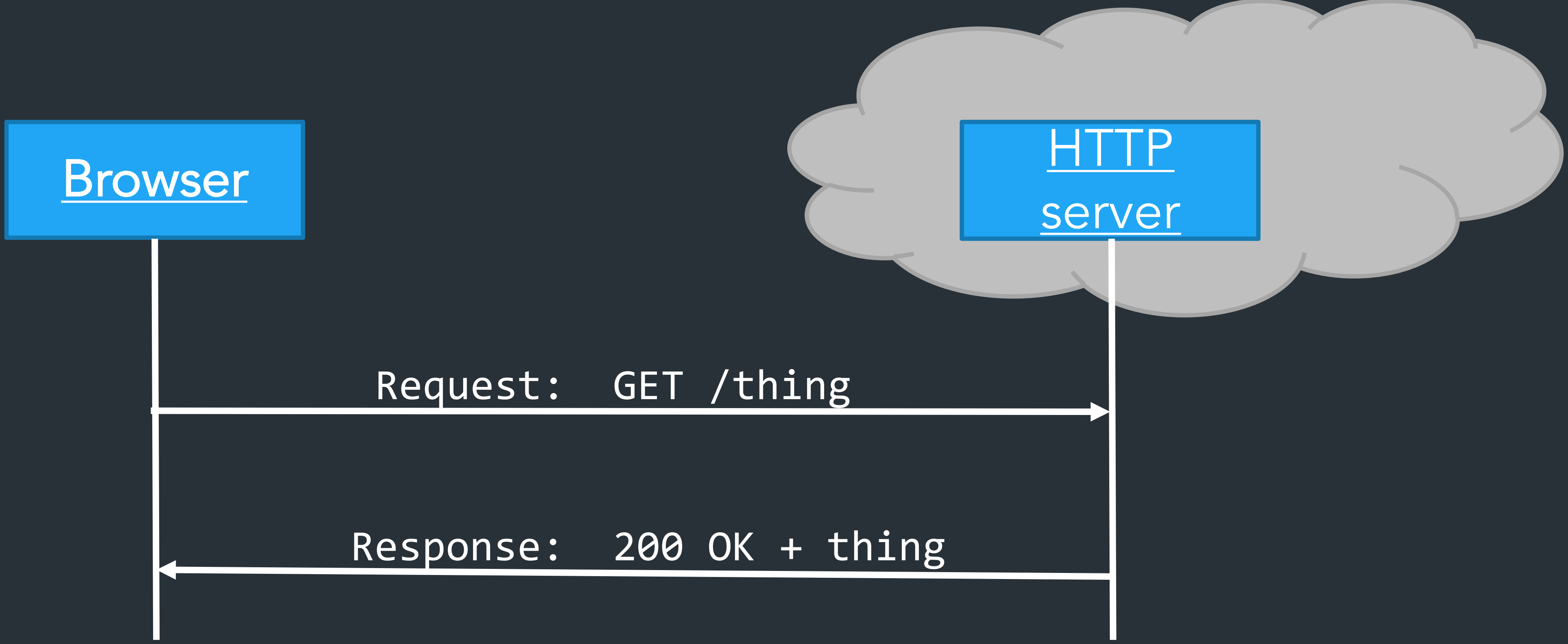

Client
HTTP server
Request: GET /thing
Response: 200 OK + thing



HTTP request: a way to fetch (**GET**) or send (**POST**) some object

- Doesn’t need to be a web page
- Doesn’t need to be from a browser

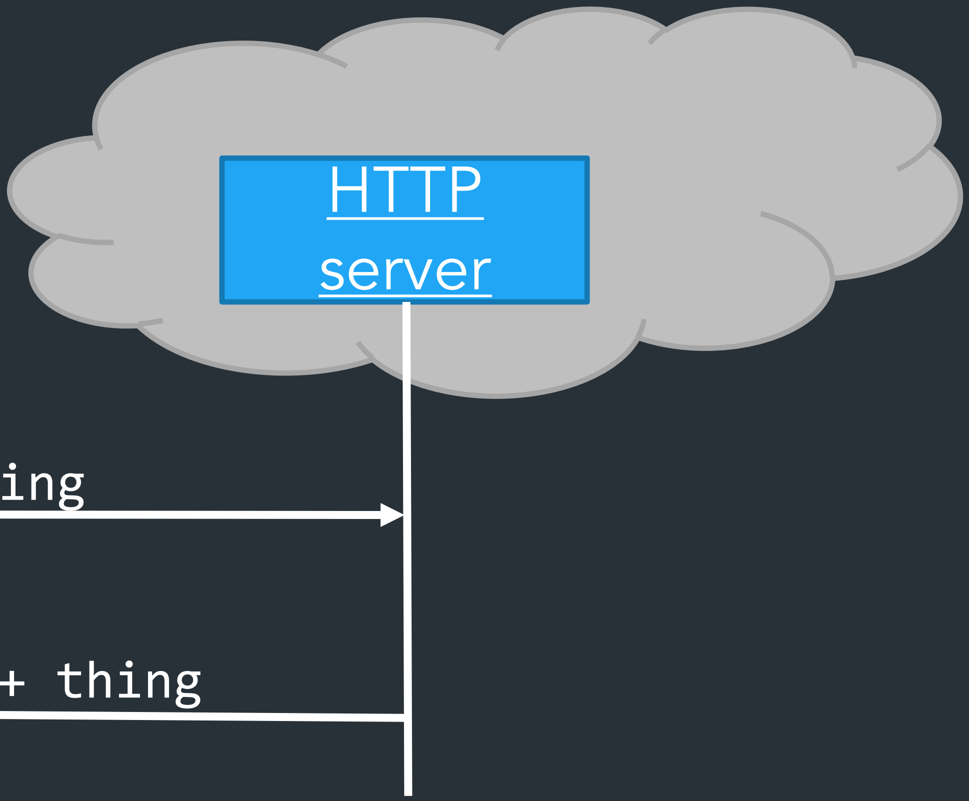
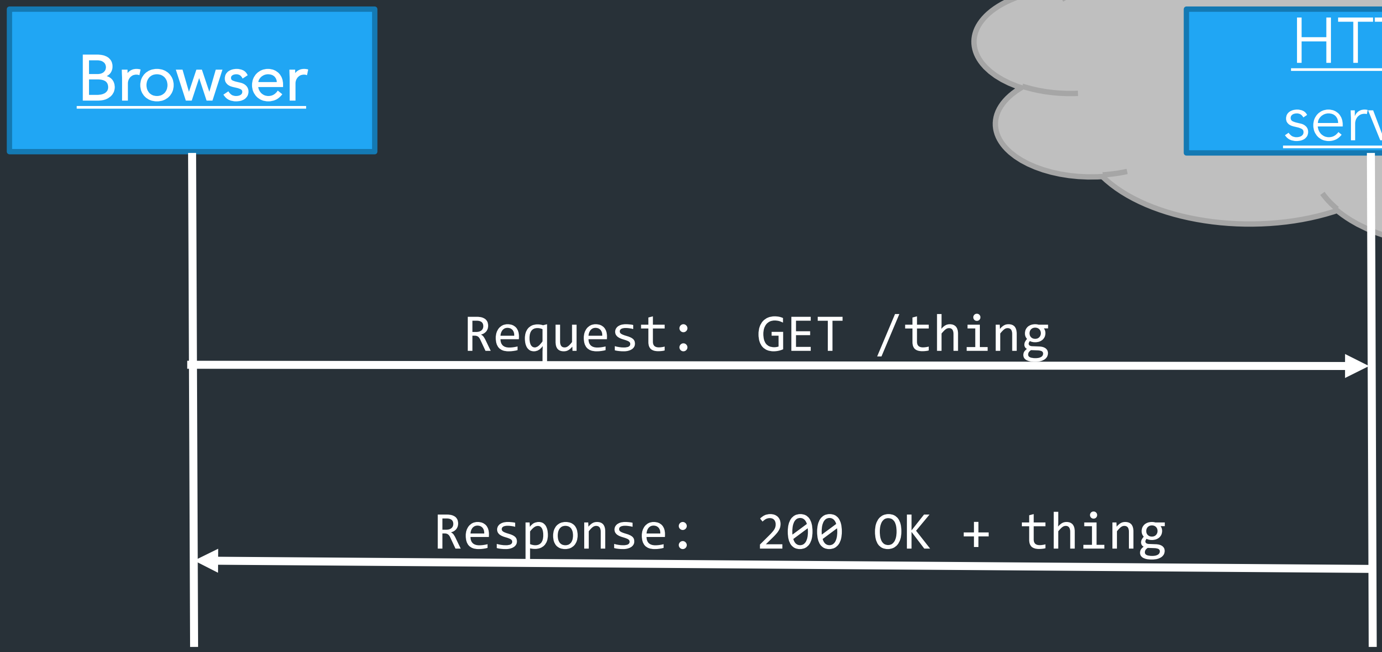
When does this not work?



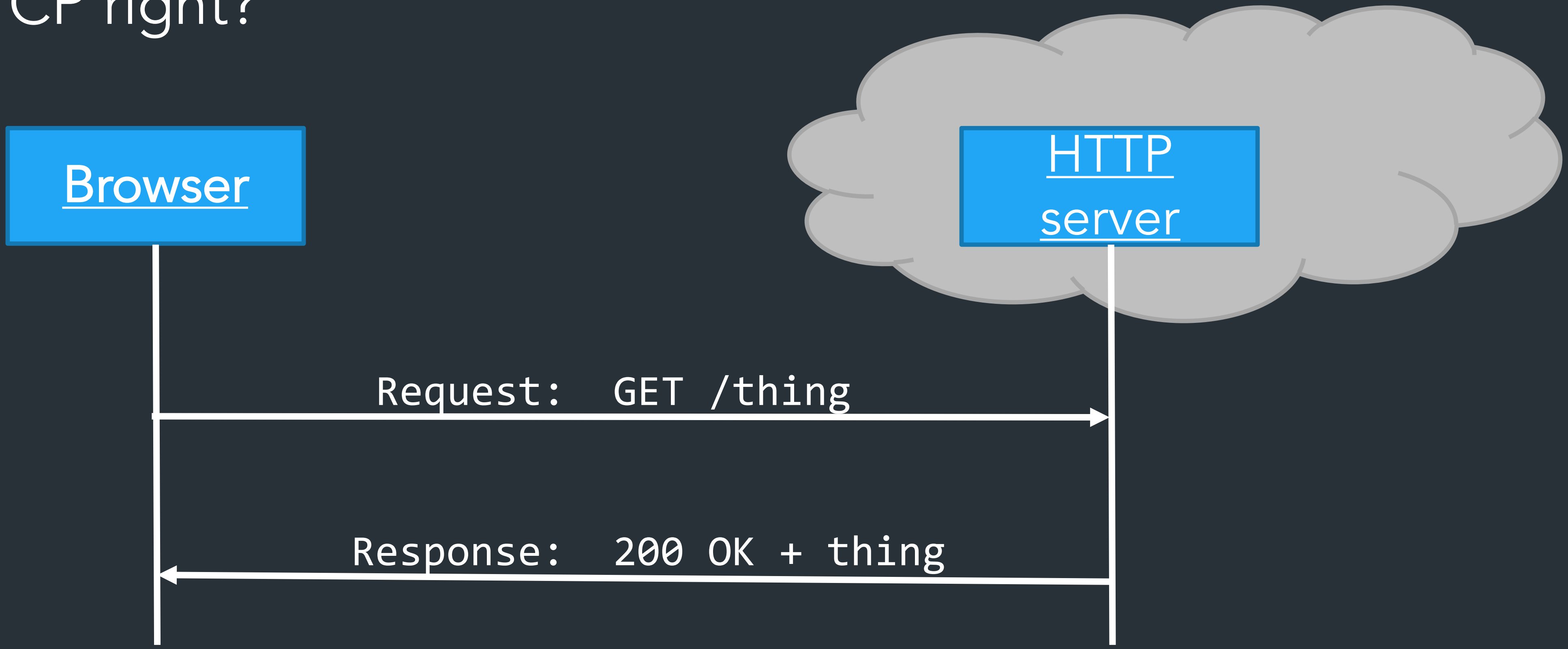
Request, response model doesn't always fit…

=> Server may need to send data asynchronously!

But it’s TCP right?

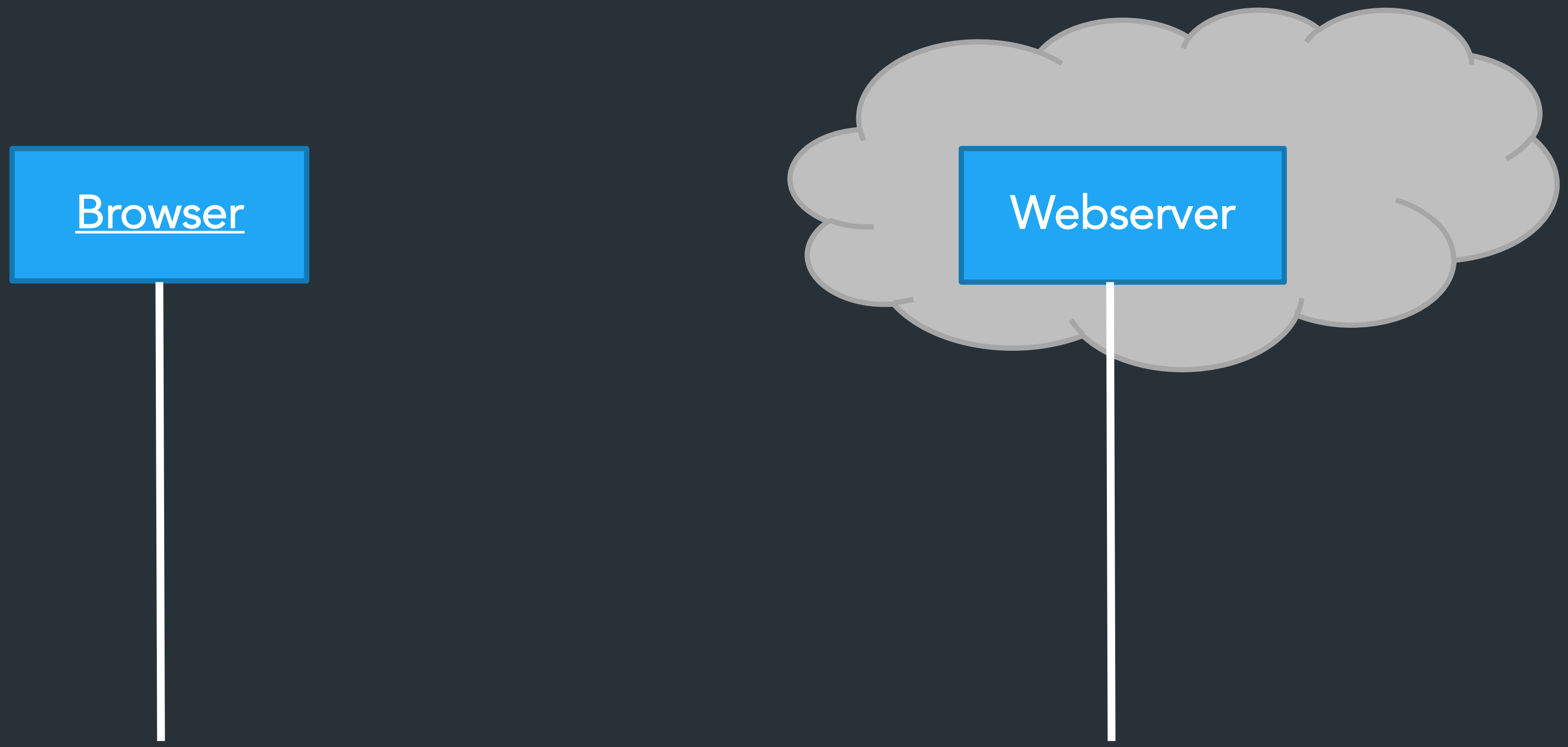



But it’s TCP right?



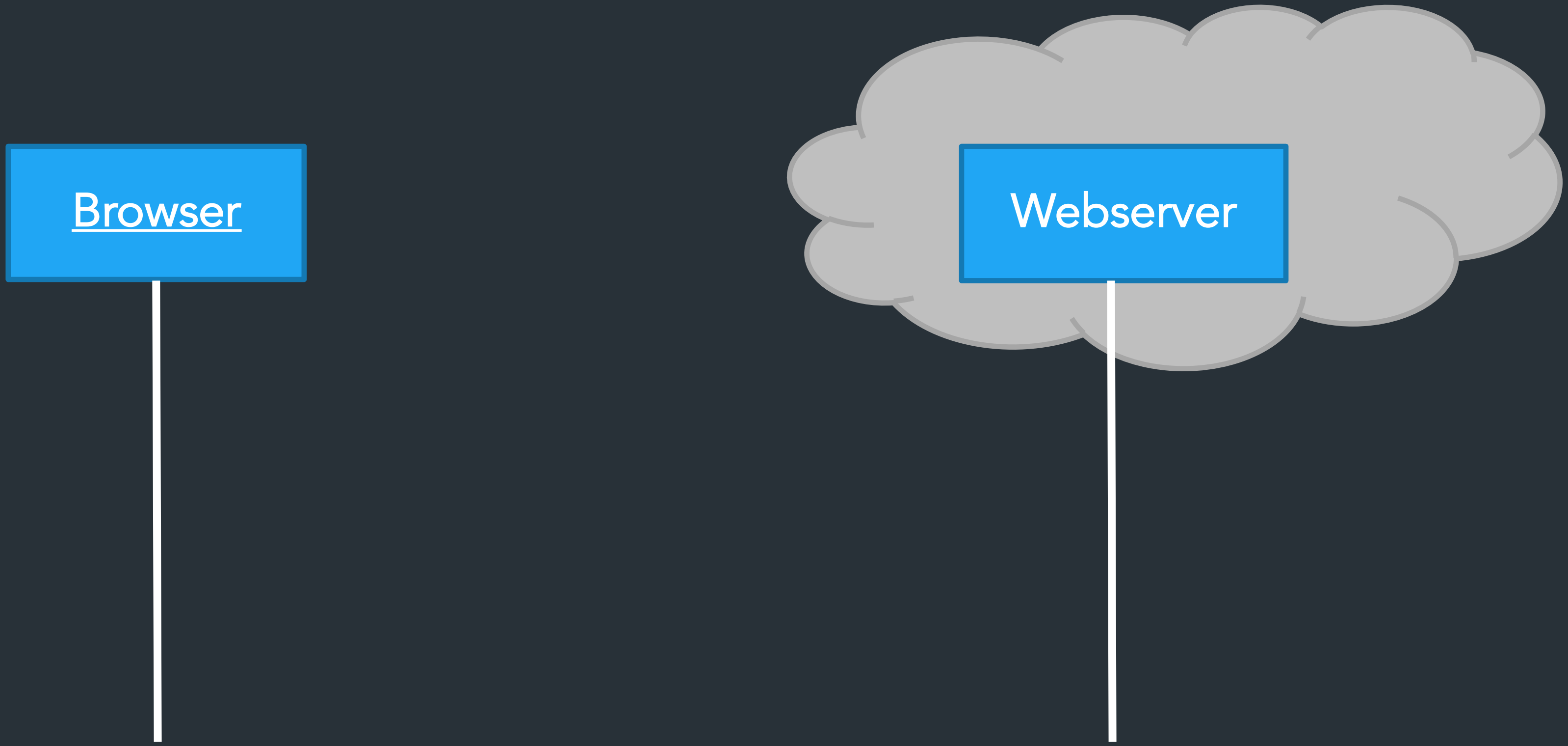
TCP is bidirectional, but the HTTP protocol is not.

What can be done?



Can the server connect to the client?

What can be done?



Can the server connect to the client?

Almost always no.
⇒ NAT, Firewalls, security policies are in the way
⇒ Don't want to allow browser to open a listen port => security risk!

How to wait for the server’s response?

One way: Polling

```
for {
     resp, err := doRequest(“http://example.com/do-you-have-my-data”)
     if resp != nil {.
      doThing(resp)
     }
     time.Sleep(1 * time.Second)
}
```

How to wait for the server’s response?

Another way: long polling

```
for {
     resp, err := doRequest(“http://example.com/do-you-have-my-data”)
     // ^ Assume this will block for very long time

     doThing(resp)
}
```

How to wait for the server’s response?

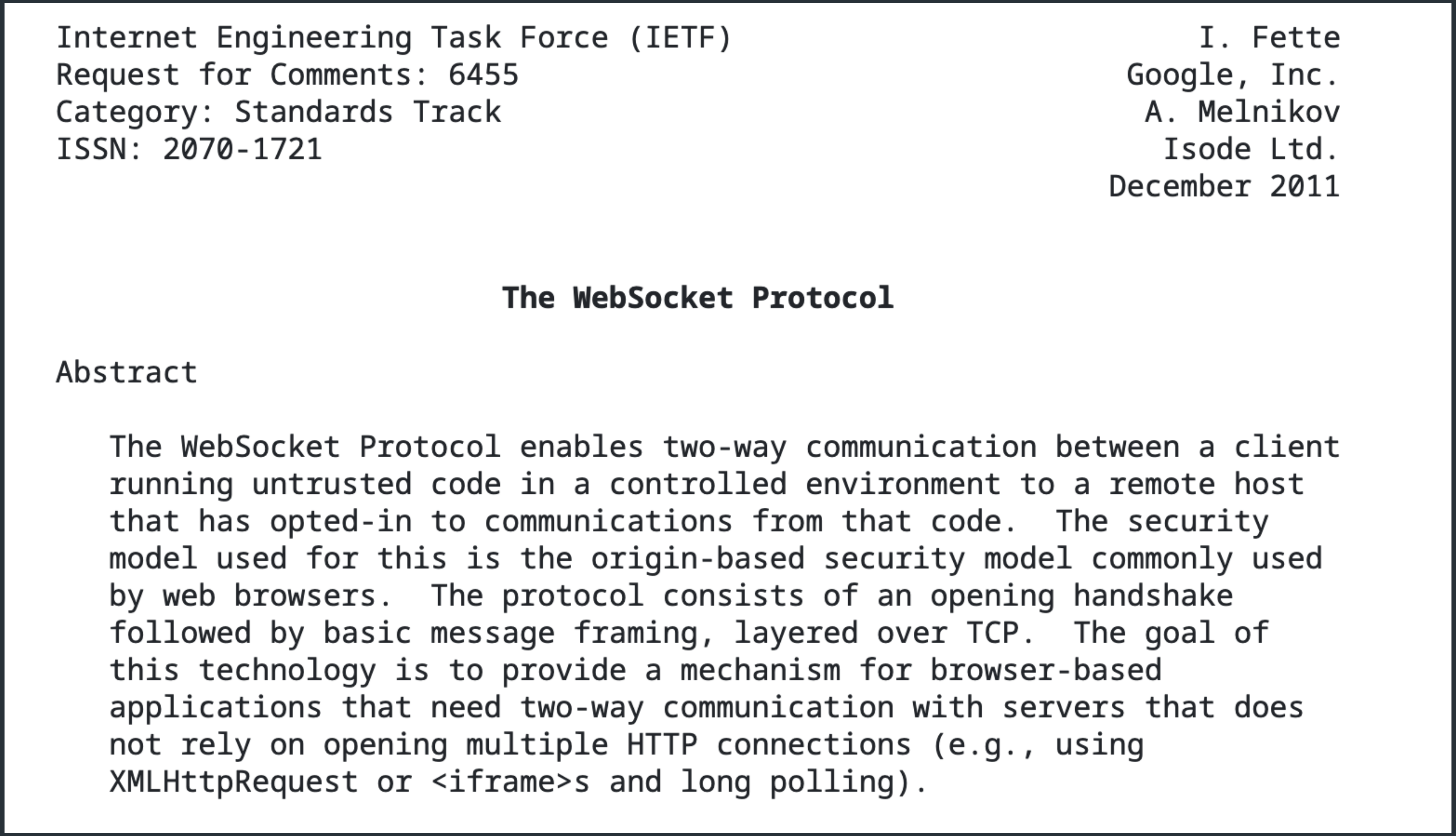
Another way: long polling
⇒ Require server to hold connection open with long timeout, respond when data is ready

```
for {
     resp, err := doRequest(“http://example.com/do-you-have-my-data”)
     // ^ Assume this will block for very long time

     doThing(resp)
}
```

Problems?

Another way: Websockets (RFC6455, 2011)



```
Internet Engineering Task Force (IETF)                          I. Fette
Request for Comments: 6455                                  Google, Inc.
Category: Standards Track                                    A. Melnikov
ISSN: 2070-1721                                               Isode Ltd.
                                                           December 2011
```

The WebSocket Protocol

Abstract

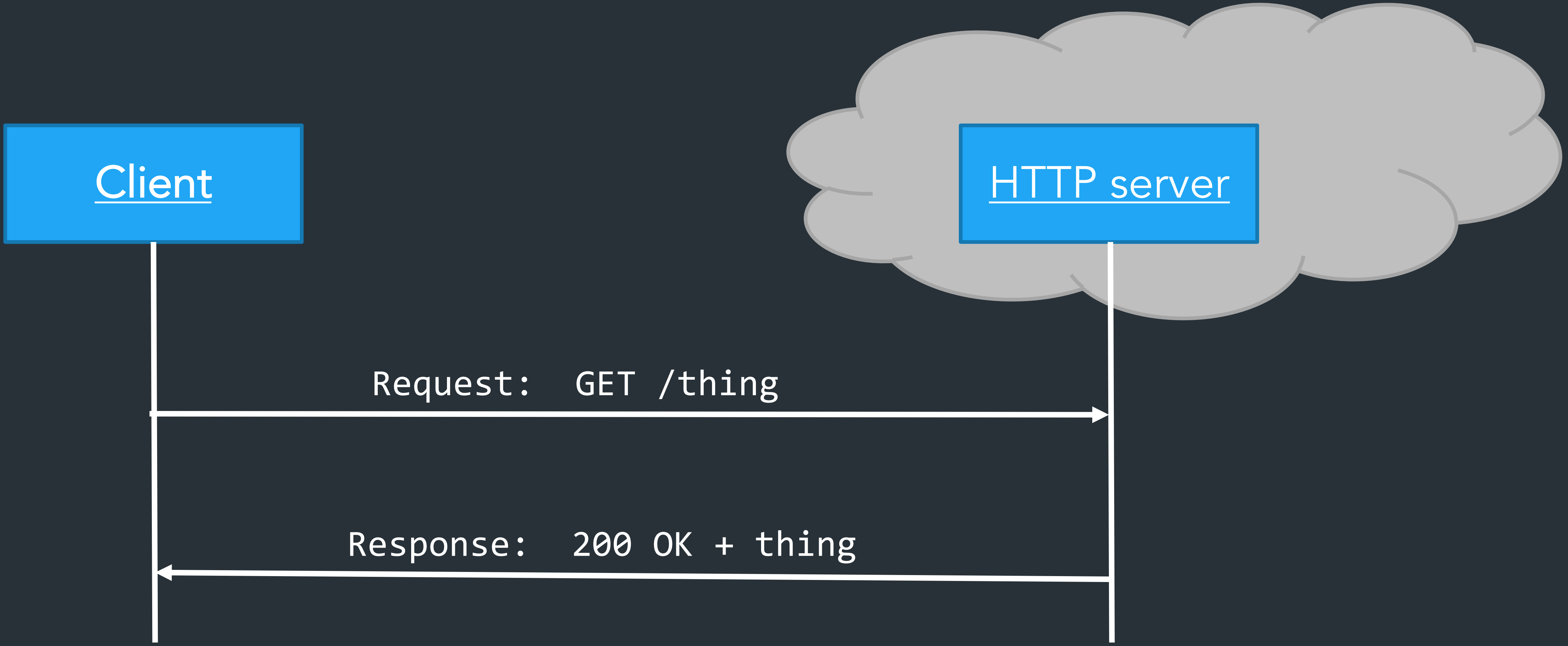
The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code. The security model used for this is the origin-based security model commonly used by web browsers. The protocol consists of an opening handshake followed by basic message framing, layered over TCP. The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections (e.g., using XMLHttpRequest or <iframe>s and long polling).

Another way: Websockets (RFC6455, 2011)

Persistent, bidirectional transport layer between browser and server
=> Can start with an HTTP request!

```
GET /chat
Host: javascript.info
Origin: https://javascript.info
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Key: Iv8io/9s+lYFgZWcXczP8Q==
Sec-WebSocket-Version: 13
```

Push notifications



HTTP request: a way to fetch (**GET**) or send (**POST**) some object

- Doesn't need to be a web page
- Doesn't need to be from a browser

⇒ Generic way to ask the server to do something => an API over the network!

Modern websites don’t just load pages when you click links:

Every modern webpage is filled with arbitrary code, usually Javascript, which can make more requests:

```
async function doRequest() {
      const response = await fetch("http://example.com/thing.json");
      const data = await response.json();
      console.log(data);
}
```

Can make requests when….

- User does something (click button, scroll, ...)
- Periodic events, timers, etc
- …

Modern websites don’t just load pages when you click links:

Every modern webpage is filled with arbitrary code, usually Javascript, which can make more requests:

```
async function doRequest() {
      const response = await fetch("http://example.com/thing.json");
      const data = await response.json();
      console.log(data);
}
```

Modern websites don’t just load pages when you click links:

Every modern webpage is filled with arbitrary code, usually Javascript, which can make more requests:

```
async function doRequest() {
      const response = await fetch("http://example.com/thing.json");
      const data = await response.json();
      console.log(data);
}
```

Can make requests when….

- User does certain action
- Periodic events, timers, etc
- …

Modern websites don’t just load pages when you click links:

Every modern webpage is filled with arbitrary code, usually Javascript, which can make more requests:

```
async function doRequest() {
	const response = await fetch("http://example.com/thing.json");
	const data = await response.json();
	console.log(data);
}
```

Can make requests when….

- User does certain action
- Periodic events, timers, etc
- …

“Arbitrary code”… from a web page?
Sound sketchy? It can be. Take CS1660.

HTTP

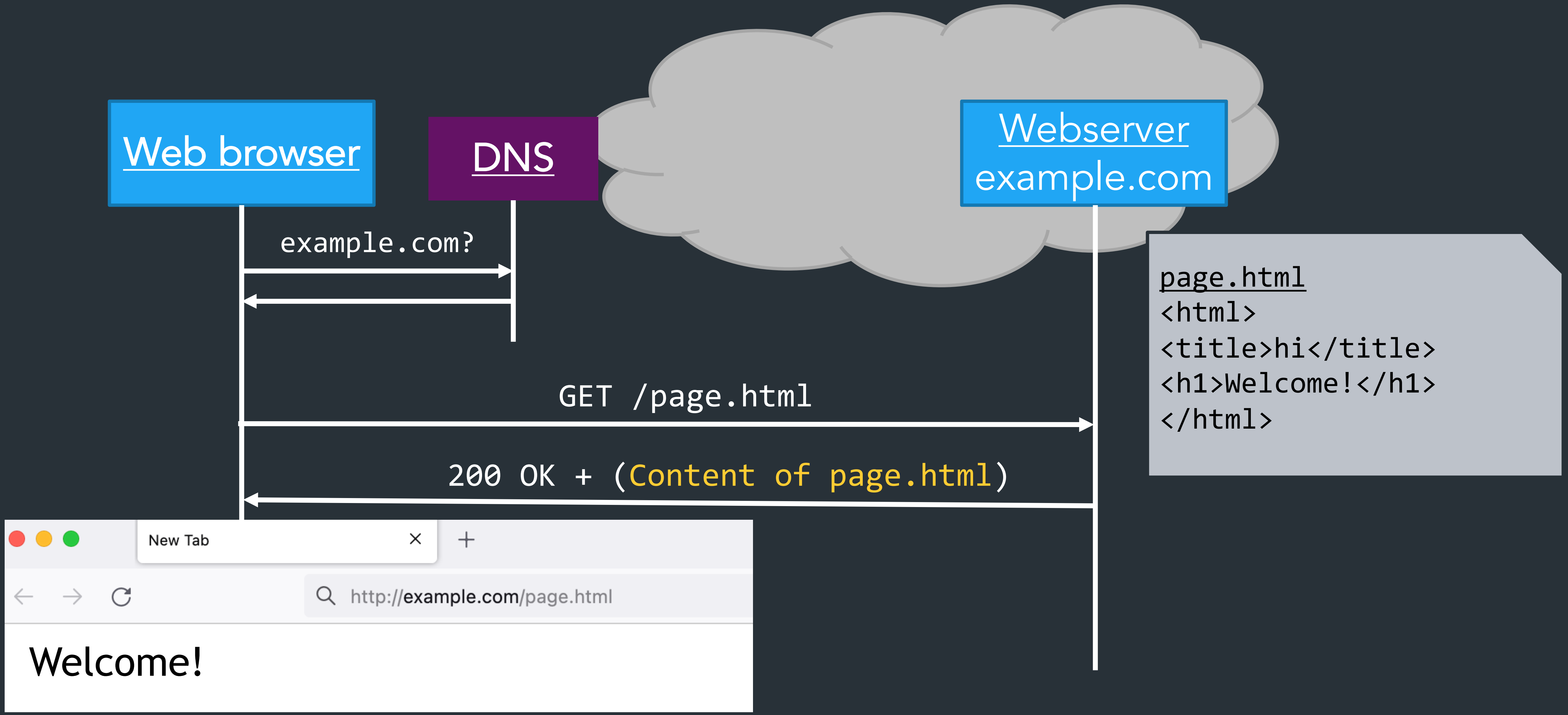
```
> telnet www.cs.brown.edu 80
Trying 128.148.32.110...
Connected to www.cs.brown.edu.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 24 Mar 2011 12:58:46 GMT
Server: Apache/2.2.9 (Debian) mod_ssl/2.2.9 OpenSSL/0.9.8g
Last-Modified: Thu, 24 Mar 2011 12:25:27 GMT
ETag: "840a88b-236c-49f3992853bc0"
Accept-Ranges: bytes
Content-Length: 9068
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Example: Github public API

```
$ curl https://api.github.com/users/ndemarinis
{
  "login": "ndemarinis",
  "id": 1191319,
  "node_id": "MDQ6VXNlcjExOTEzMTk=",
  "avatar_url": "https://avatars.githubusercontent.com/u/1191319?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/ndemarinis",
  "type": "User",
  "site_admin": false,
  "name": "Nick DeMarinis",
  "blog": "https://vty.sh",
  "twitter_username": null,
  "public_repos": 10,
 . . .
}
```



page.html
```
<html>
<title>hi</title>
<h1>Welcome!</h1>
</html>
```

Server returns response (in this case, with HTML)