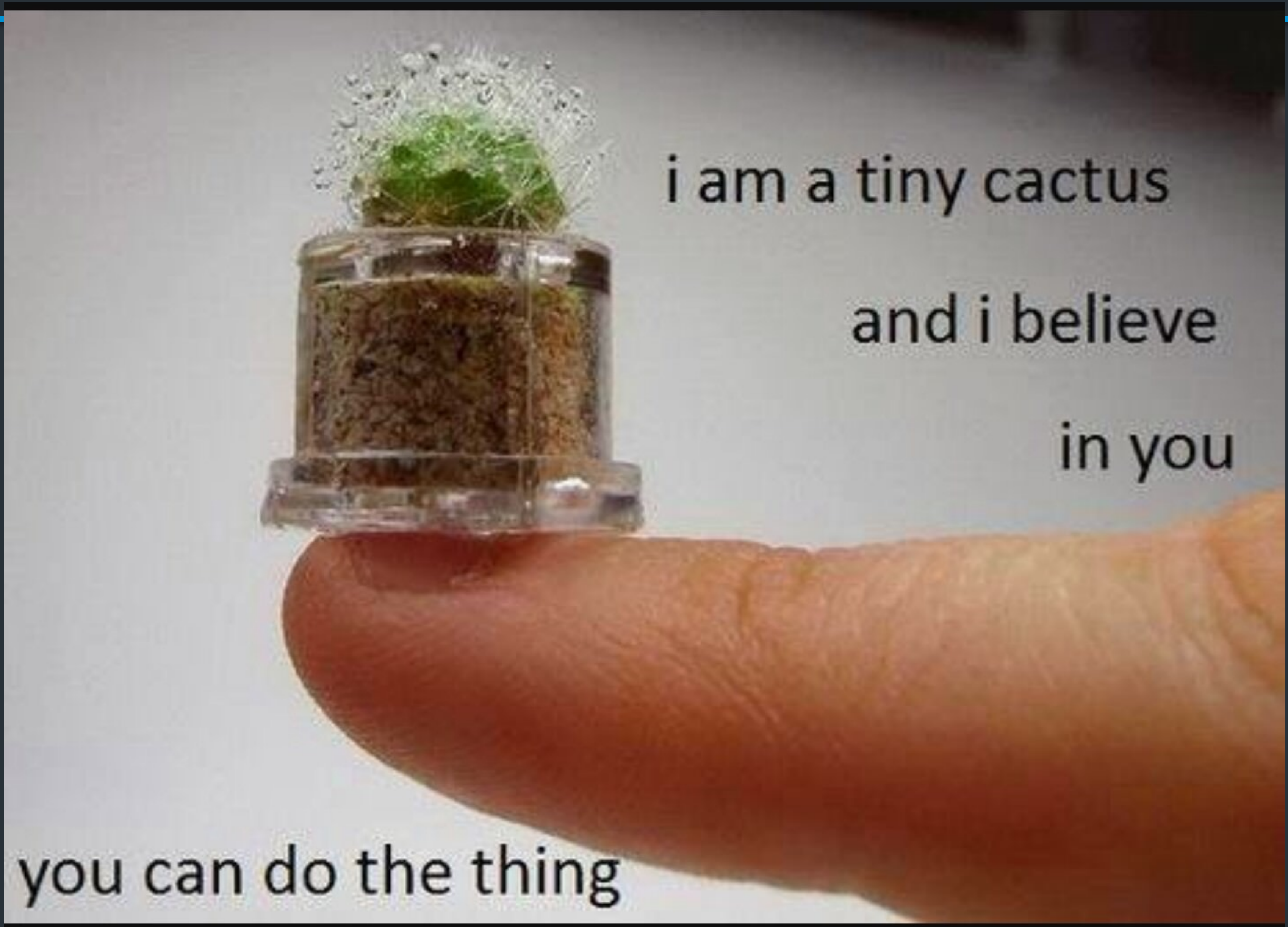
CSCI-1680
DNS

Nick DeMarinis

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

Breathe



Administrivia

TCP officially due tonight (Tuesday, Nov 21)

- Office hours 2-5pm (Me); 5-7pm (Alex); 7-9pm (Rhea)

- Like with IP: you can continue to make *small* bugfixes after the deadline
  - OK: Fixing *small* bugs, README, capture files, code cleanup
  - Not OK: eg. implementing sendfile/recvfile, teardown, submitting untested code

- Grading meetings: after break

Administrivia

TCP officially due tonight (Tuesday, Nov 21)

- Office hours 2-5pm (Me); 5-7pm (Alex); 7-9pm (Rhea)
- Like with IP: you can continue to make *small* bugfixes after the deadline
  - OK: Fixing *small* bugs, README, capture files, code cleanup
  - Not OK: eg. implementing sendfile/recvfile, teardown, submitting untested code
- Grading meetings: after break

If you want to submit late
Monday 11/27 by 11:59pm EST => one day late

The final project

Out after break, handout online after class

…maybe skim it before break?

The final project

Out after break, handout online after class

…maybe skim it before break?

What it is

- Open-ended: build something new related to class topics
- List of ideas in document… or propose your own!

Project examples

- Make your own iterative DNS resolver
- Build a simple HTTP server
- Make your own web API (more next week)
- Implement Snowcast, etc. using RPCs (more next week)
- Extend your IP/TCP in some way…

These are only a few ideas!

Final project Logistics

Out after break, document online after class

…maybe skim it before break?

<u>Deadlines</u>

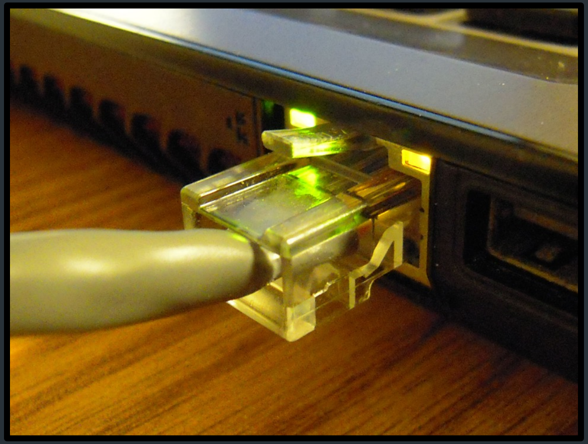
- Team assignment form: Due Tuesday, 11/28
  - Keep your current groups, or form new ones, or work solo
- Project proposal: Due Friday, 12/1
- Final submission: Due Thursday, 12/14

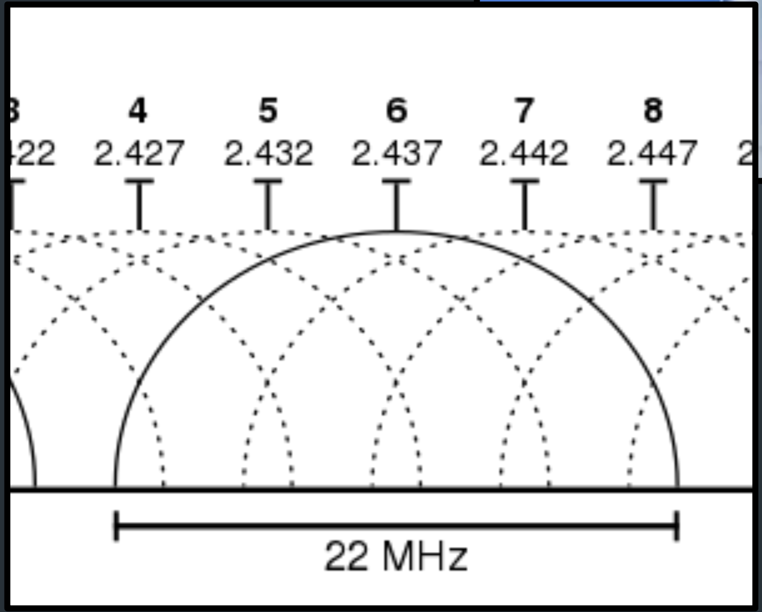
IPoAC

How can we improve the physical layer?

Traditional links have fixed bandwidth

- Media limits what frequencies can be used for signal
- Places upper bound on channel capacity





What if we weren’t constrained by the EM spectrum?

How else can we transmit data?

Network Working Group
Request for Comments: 1149
D. Waitzman
BBN STC
1 April 1990

A Standard for the Transmission of IP Datagrams on Avian Carriers

Status of this Memo

This memo describes an experimental method for the encapsulation of IP datagrams in avian carriers. This specification is primarily useful in Metropolitan Area Networks. This is an experimental, not recommended standard. Distribution of this memo is unlimited.

Overview and Rational

Avian carriers can provide high delay, low throughput, and low altitude service. The connection topology is limited to a single point-to-point path for each carrier, used with standard carriers, but many carriers can be used without significant interference with each other, outside of early spring. This is because of the 3D ether space available to the carriers, in contrast to the 1D ether used by

RFC1149: IPoAC

IP over Avian Carriers (1 April 1990)

- High delay, low throughput, low altitude datagram service
- Nearly unlimited movement in 3D etherspace
- Intrinsic collision avoidance
- Typical MTU: 256 milligrams

```
Network Working Group                                    D. Waitzman
Request for Comments: 1149                                   BBN STC
                                                        1 April 1990
```

A Standard for the Transmission of IP Datagrams on Avian Carriers

Status of this Memo

This memo describes an experimental method for the encapsulation of IP datagrams in avian carriers. This specification is primarily

IPoAC: Design

IPoAC: Implementation

Proof of concept: 28 April 2001
Bergen, Norway
https://web.archive.org/web/20140215072548/http://www.blug.linux.no/rfc1149/

IPoAC in practice

```
$ ping -c 9 -i 900 10.0.3.1
PING 10.0.3.1 (10.0.3.1): 56 data bytes
64 bytes from 10.0.3.1: icmp_seq=0 ttl=255 time=6165731.1 ms
64 bytes from 10.0.3.1: icmp_seq=4 ttl=255 time=3211900.8 ms
64 bytes from 10.0.3.1: icmp_seq=2 ttl=255 time=5124922.8 ms
64 bytes from 10.0.3.1: icmp_seq=1 ttl=255 time=6388671.9 ms

--- 10.0.3.1 ping statistics ---
9 packets transmitted, 4 packets received, 55% packet loss round-trip
min/avg/max = 3211900.8/5222806.6/6388671.9 ms
```

IPoAC: (more) Modern implementations

Pigeon-powered Internet takes flight

One of the Internet's ... to life: transmitting n...

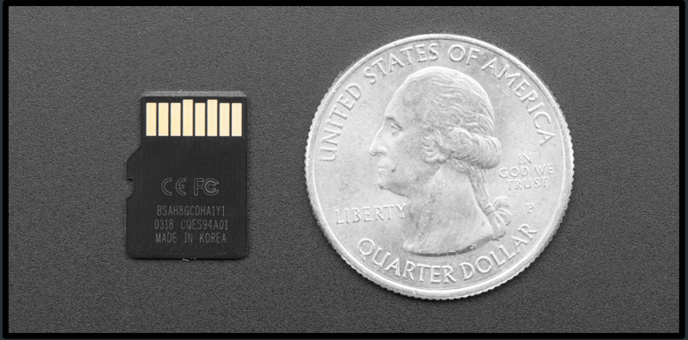
Stephen Shankland
Jan. 2, 2002 4:43 p.m. PT

BUSINESS

Pigeon carries data bundles faster than Telkom

Staff Reporter 10 Sep 2009

Today: microSD card: ~250mg, 1TB



+

= ???

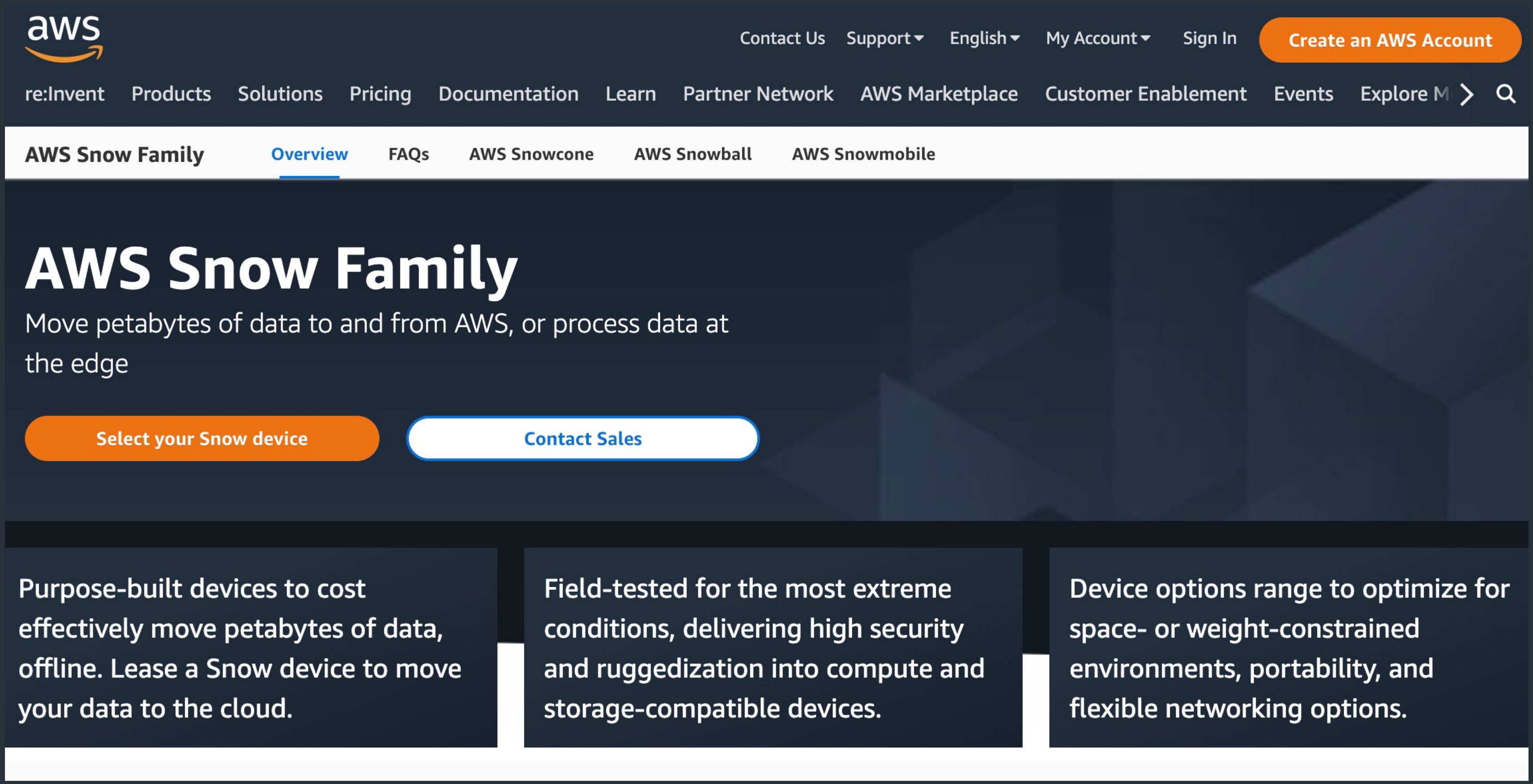
But actually

What happens if you have a LOT of data to move into the cloud?

But actually

What happens if you have a LOT of data to move into the cloud?

Example: AWS



Feature comparison matrix

	AWS SNOWCONE	AWS SNOWBALL EDGE STORAGE OPTIMIZED	AWS SNOWBALL EDGE COMPUTE OPTIMIZED	AWS SNOWMOBILE
Usable HDD Storage	8 TB	80 TB	N/A	100 PB
Usable SSD Storage	14 TB	1 TB	28 TB	No
Usable vCPUs	4 vCPUs	40 vCPUs	104 vCPUs	N/A
Usable Memory	4 GB	80 GB	416 GB	N/A
Device Size	9in x 6in x 3in 227 mm x 148.6 mm x 82.65 mm	548 mm x 320 mm x 501 mm	548 mm x 320 mm x 501 mm	45 ft. shipping container
Device Weight	4.5 lbs. (2.1 kg)	49.7 lbs. (22.3 kg)	49.7 lbs. (22.3 kg)	N/A
Storage Clustering	No	Yes, 5-10 nodes	Yes, 5-10 nodes	N/A
256-bit Encryption	Yes	Yes	Yes	Yes
HIPAA Compliant	No	Yes, eligible	Yes, eligible	Yes, eligible

RFC791: IPv4 Header

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               Data                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Internet Header Format [RFC-791]

IP over Burrito Carriers

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Obvious|  Onion      |  Jalapenos  |     Physical Length (mm)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Number Written on Foil        |Bean Type|  Number of Beans  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Given Delivery Time | Guacamole |         Receipt               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Lettuce                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              Rice                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              Beef                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

               The Burrito Internet Header Format
```

April Fool’s Day RFCs

April Fools' Day Request for Comments

From Wikipedia, the free encyclopedia
(Redirected from Peg DHCP)

A Request for Comments (RFC), in the context of Internet governance, is a type of publication from the Internet Engineering Task Force (IETF) and the Interne
behaviors, research, or innovations applicable to the working of the Internet and Internet-connected systems.

Almost every April Fools' Day (1 April) since 1989, the Internet RFC Editor has published one or more humorous Request for Comments (RFC) documents, fol
RFC 527 called ARPAWOCKY, a parody of Lewis Carroll's nonsense poem "Jabberwocky". The following list also includes humorous RFCs published on othe

Contents [hide]

List of April Fools' RFCs [edit]

1978

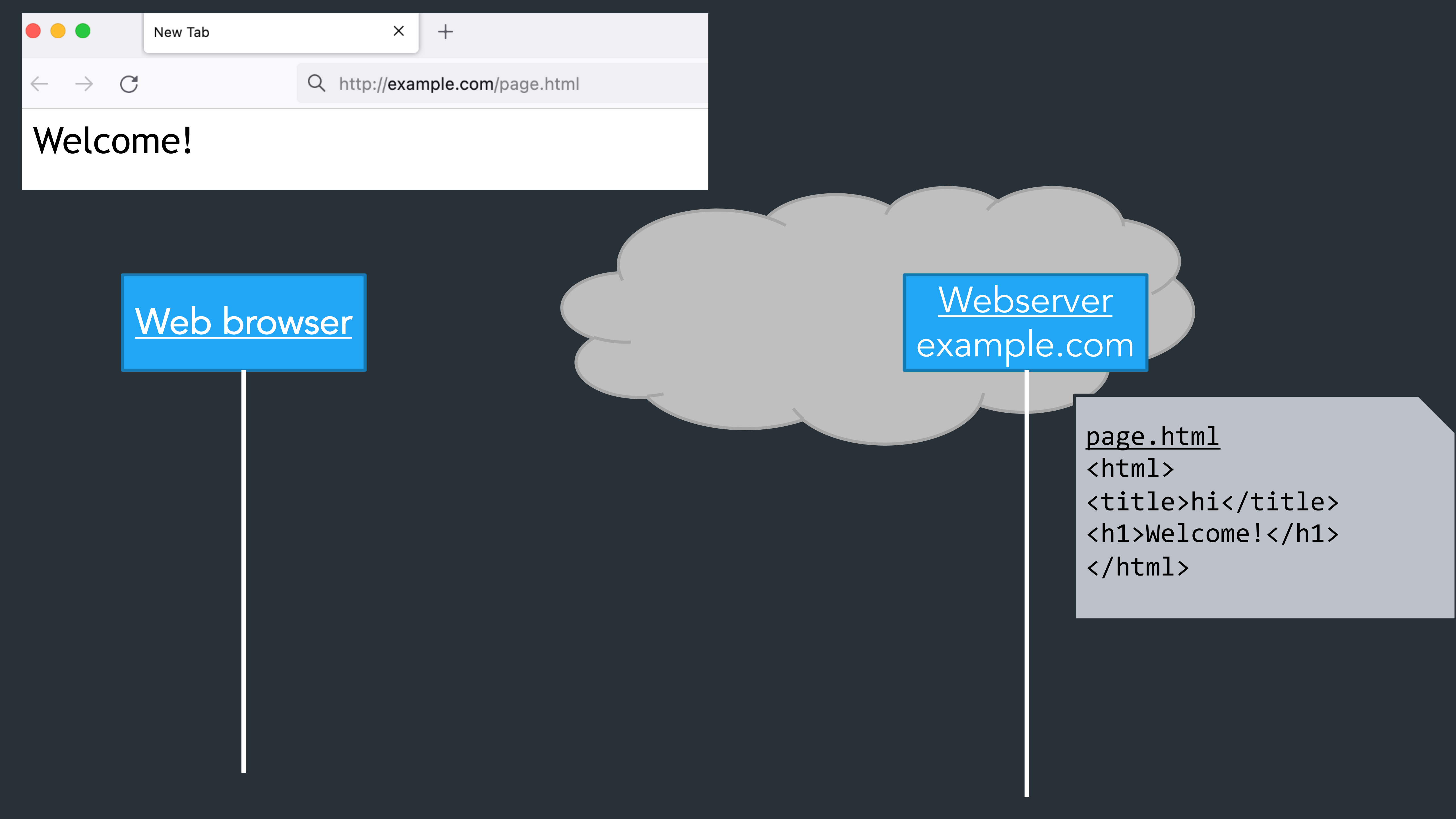
M. R. Crispin (1 April 1978). *TELNET RANDOMLY-LOSE option*. IETF. doi:10.17487/RFC0748. RFC 748.

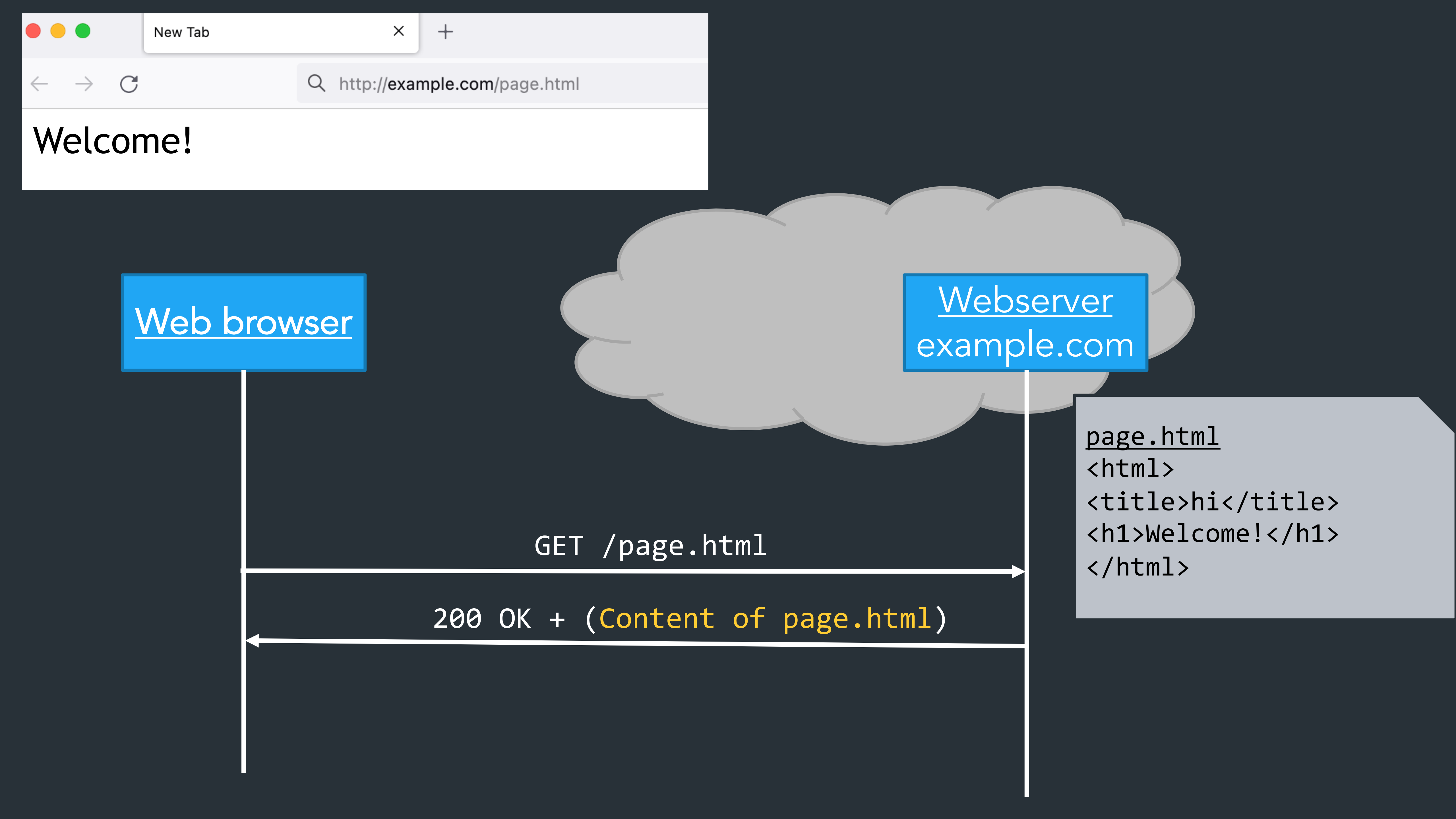
A parody of the TCP/IP documentation style. For a long time it was specially marked in the RFC index with "note date of issue".

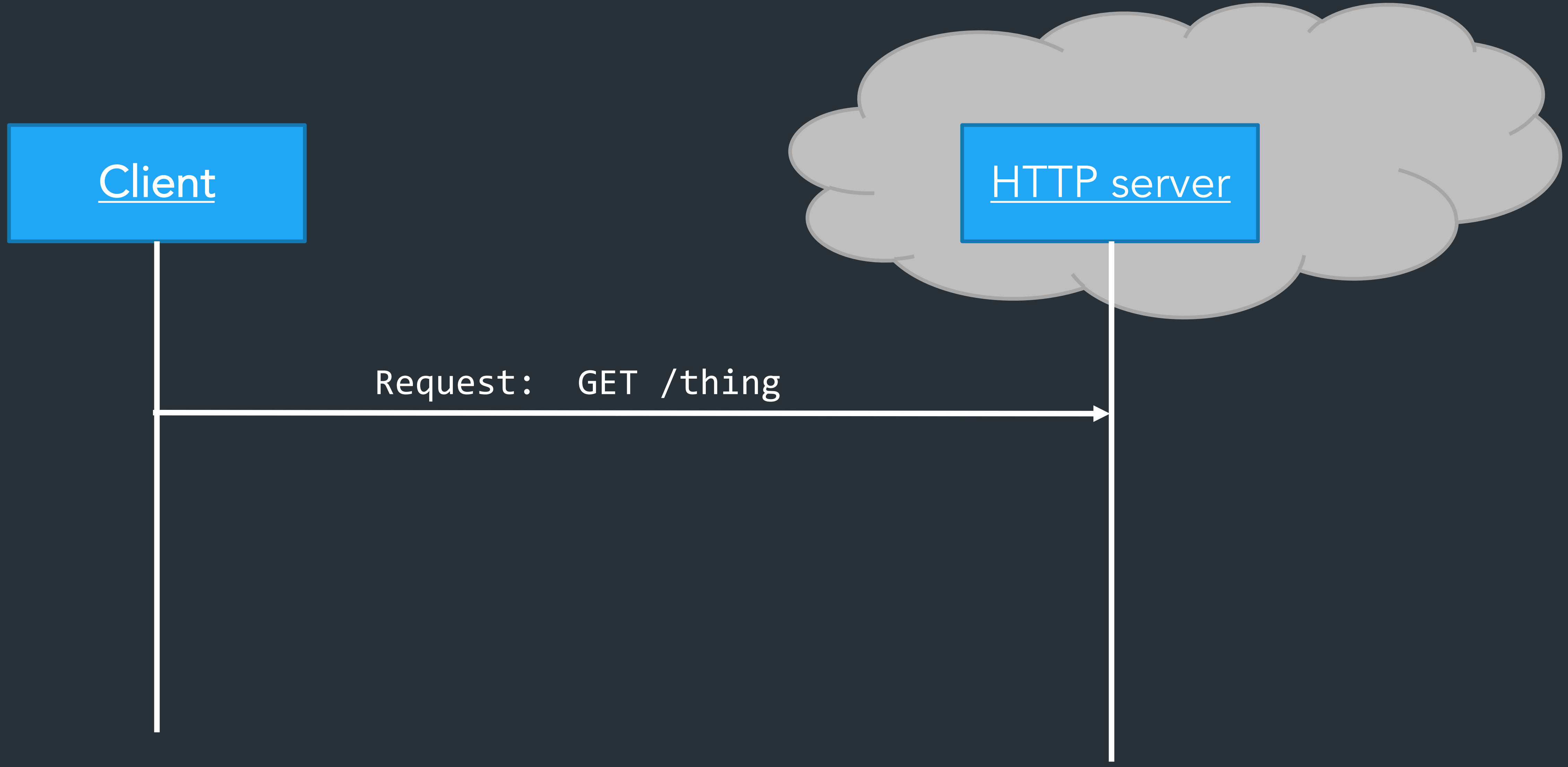
1989

https://en.wikipedia.org/wiki/April_Fools%27_Day_Request_for_Comments Enjoy!

Back to HTTP


New Tab
http://example.com/page.html
Welcome!
Web browser
Webserver
example.com
page.html
<html>
<title>hi</title>
<h1>Welcome!</h1>
</html>


New Tab
http://example.com/page.html
Welcome!
Web browser
Webserver
example.com
page.html
<html>
<title>hi</title>
<h1>Welcome!</h1>
</html>
GET /page.html
200 OK + (Content of page.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

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:

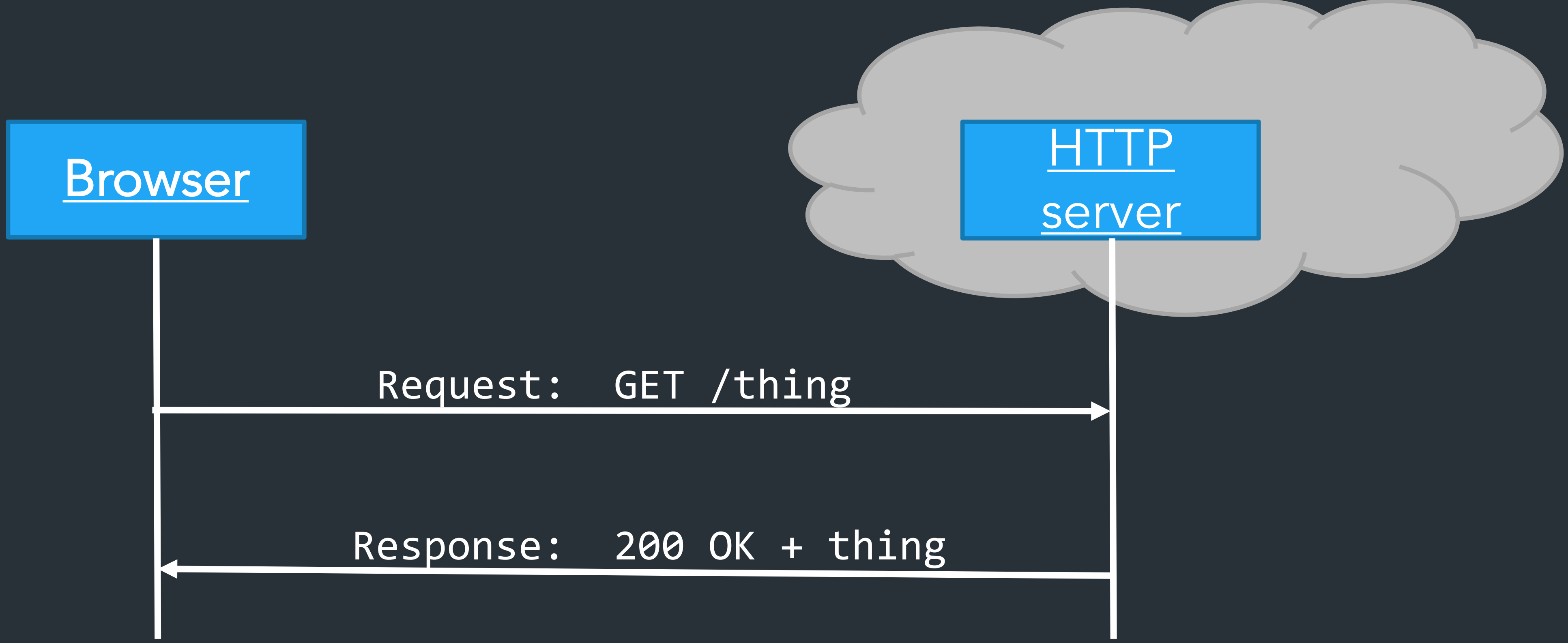
Every modern webpage is filled with arbitrary code, usually Javascript, which can make more requests:

```
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.

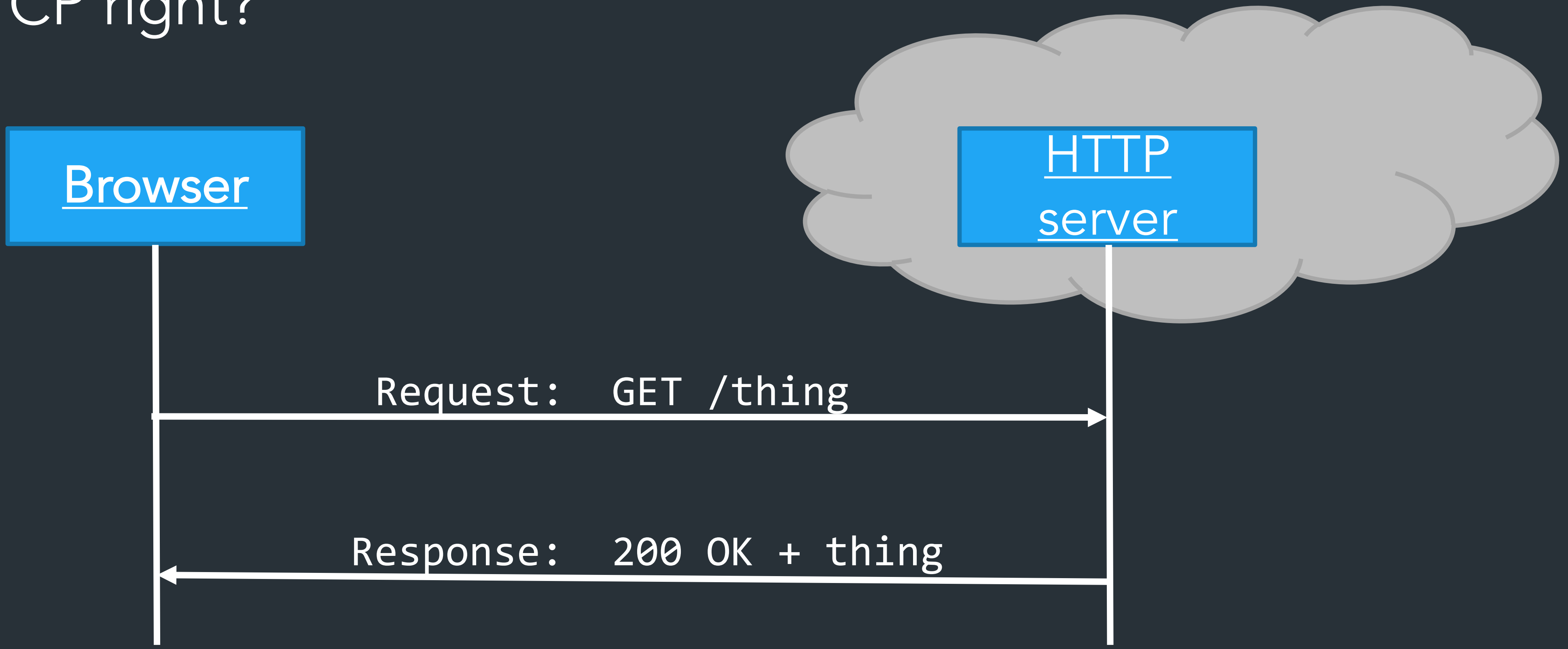
When does this not work?



Request, response model doesn’t always fit…

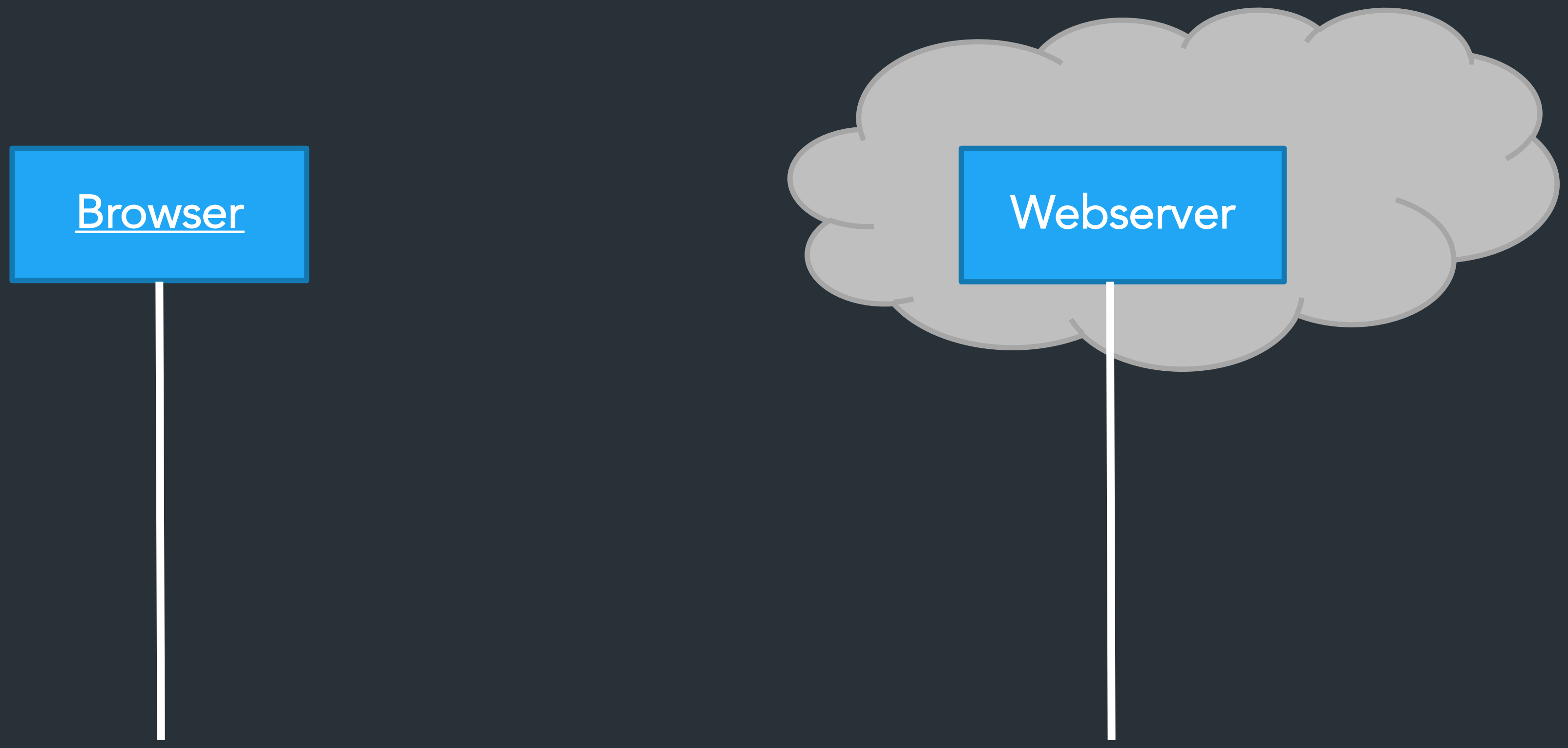
=> Server may need to send data asynchronously!

But it's TCP right?



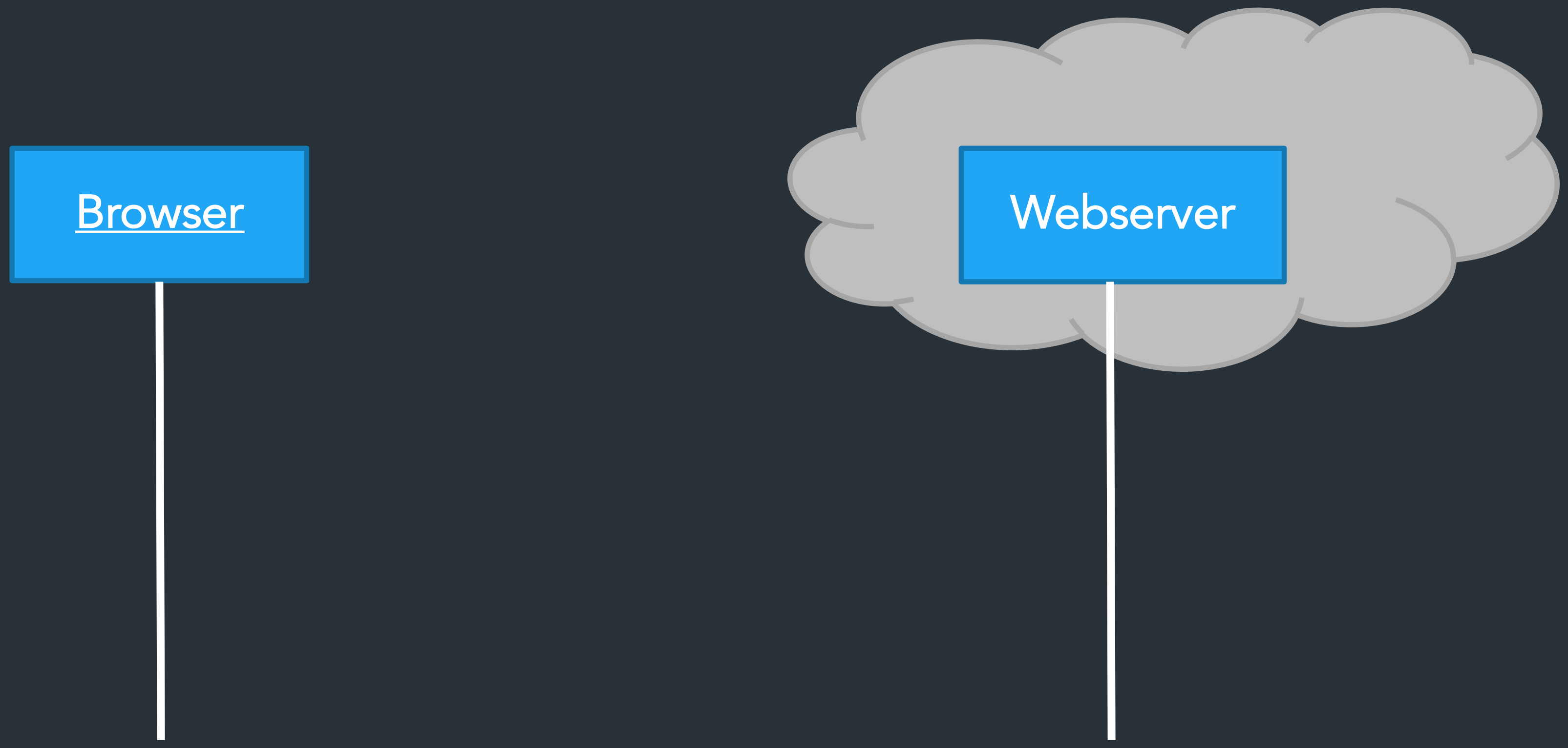
TCP is bidirectional, but the HTTP protocol is not.

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?

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)
}
```

Problems?

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)
}
```

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?

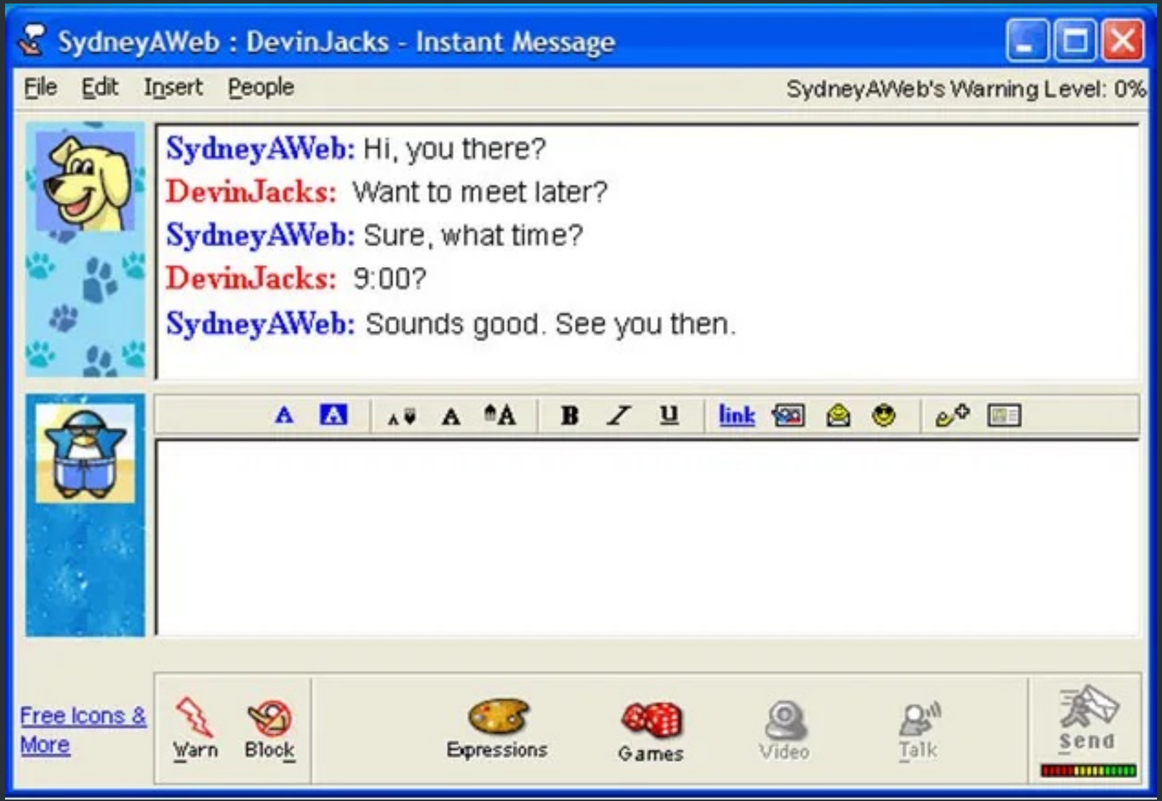
Another way: websockets (RFC6455, 2011)

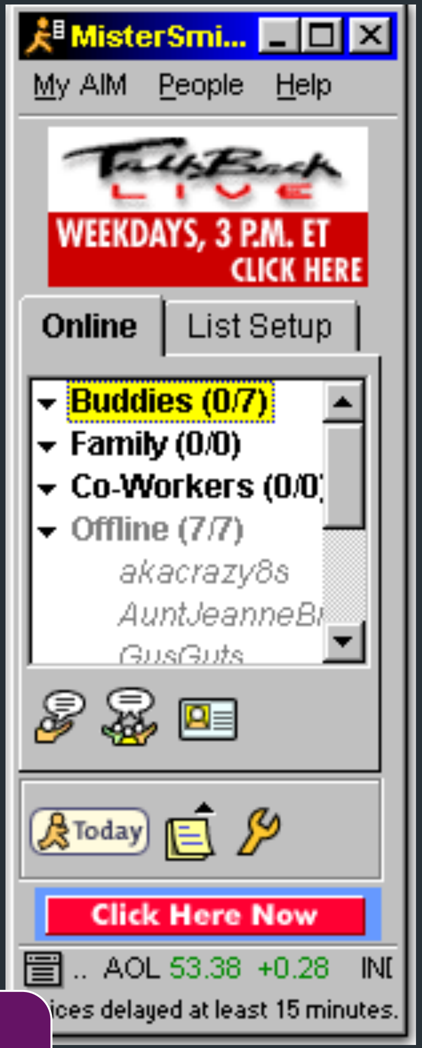
Persistent, bidirectional transport layer between browser and server

=> 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
```

Speaking of chat…





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

Push notifications

HTTP

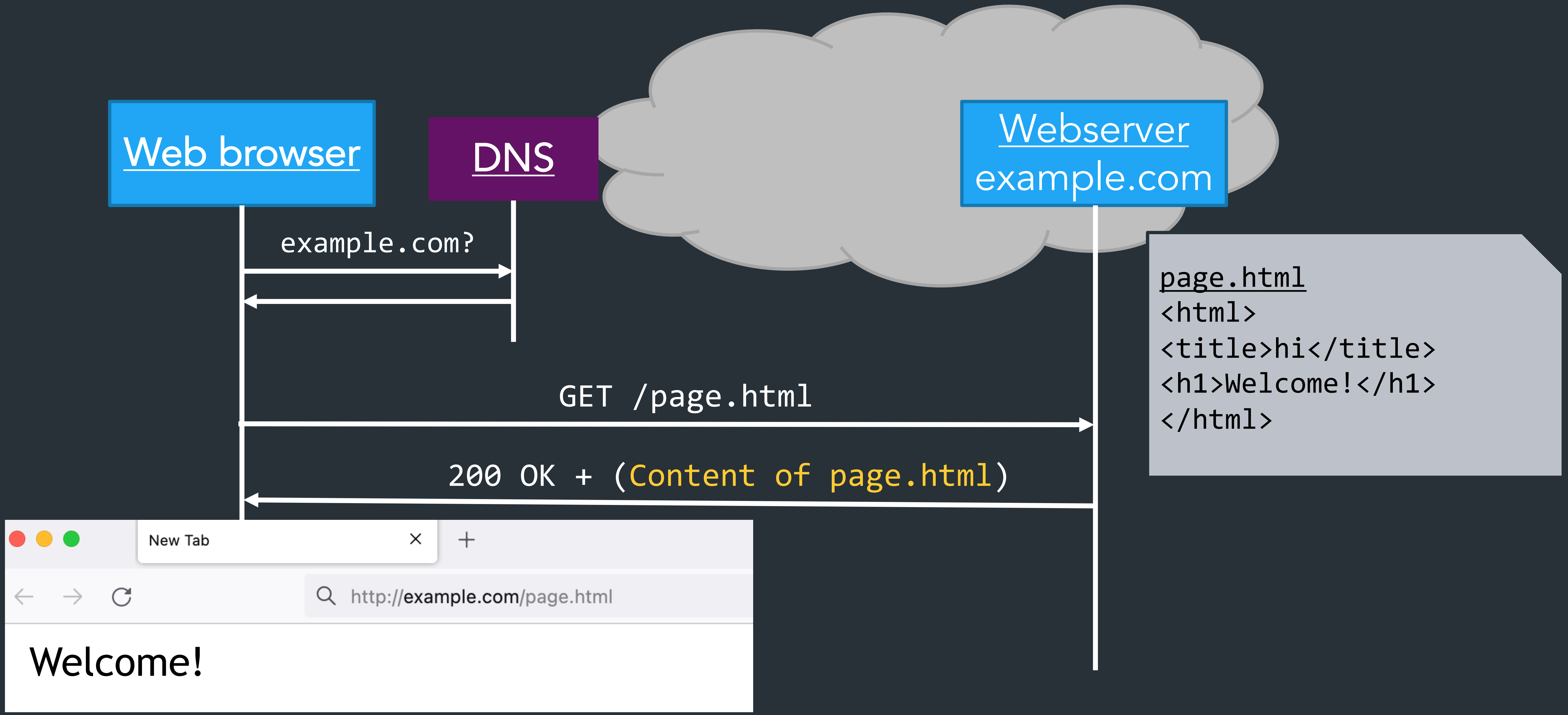
```
> telnet www.cs.brown.edu 80
Trying 128.148.32.110...
Connected to www.cs.brown.edu.
Escape character is '^]'.
GET / HTTP/1.0

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,
 . . .
}
```



Server returns response (in this case, with HTML)