Web Security

Week 12 - DNS rebinding attacks

Old Dominion University
Department of Computer Science
CS 495/595 Spring 2022
Michael L. Nelson <mln@cs.odu.edu>

2022-04-11

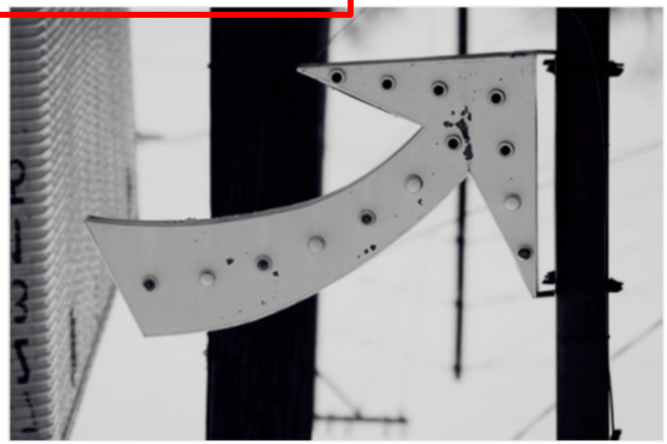
The Register®

Business ▸ The Channel

Demo shows how web attack threatens fabric of the universe

All hail the power of DNS rebinding

By Dan Goodin 9 Apr 2008 at 03:16 16 SHARE ▼



RSA Showing how the web's underpinnings can be abused to attack assets presumed to be secure, a researcher unveiled a website that can log into a home router and change key settings, such as administrator passwords and servers used to access trusted web destinations.

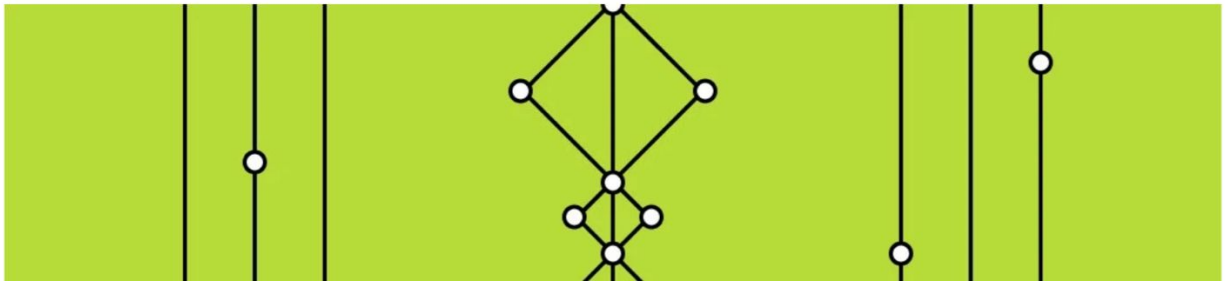
Rather than creating a trojan or other piece of specialized malware to access servers or other devices behind a firewall, researcher Dan Kaminsky, a director of penetration testing firm IOActive, showed how a

https://www.theregister.com/2008/04/09/dns_rebinding_attack/

LILY HAY NEWMAN SECURITY 06.19.2018 04:51 PM

Millions of Streaming Devices Are Vulnerable to a Retro Web Attack

Using a technique called DNS rebinding, one amateur hacker found vulnerabilities in devices from Google, Roku, Sonos, and more.



https://www.wired.com/story/chromecast-roku-sonos-dns-rebinding-vulnerability/

DNS rebinding attacks

- Pretty much every IoT device is/was vulnerable
- Pretty much every local server is/was vulnerable
- For a long time, folks did not take DNS rebinding attacks seriously
- Very effective attack, and yet very little awareness about it in industry

DNS rebinding attacks

- 165 million printers, or 66 percent, are vulnerable to DNS rebinding attacks. The company named Hewlett Packard, Epson, Konica, Lexmark, and Xerox as examples of representative manufacturers shipping vulnerable printers.
- 160 million, or 75 percent, of IP cameras by manufacturers such as Axis Communications, GoPro, Sony, and Vivotek are vulnerable.
- 124 million, or 77 percent, of IP phones are vulnerable; manufacturers include Avaya, Cisco, Dell, NEC, and Polycom.
- 28 million, or 57 percent, of smart TVs – Roku-integrated, Samsung, and Vizio – are vulnerable.
- 14 million, or 87 percent, of switches, routers and access points are vulnerable; manufacturers include Cisco, Netgear, Extreme, Aruba, and Avaya.
- 5.1 million, or 78 percent, of streaming media players and smart speakers by Apple, Google, Roku, and Sonos are vulnerable.

https://www.armis.com/resources/iot-security-blog/dns-rebinding-exposes-half-a-billion-iot-devices-in-the-enterprise/

Radio Thermostat CT50

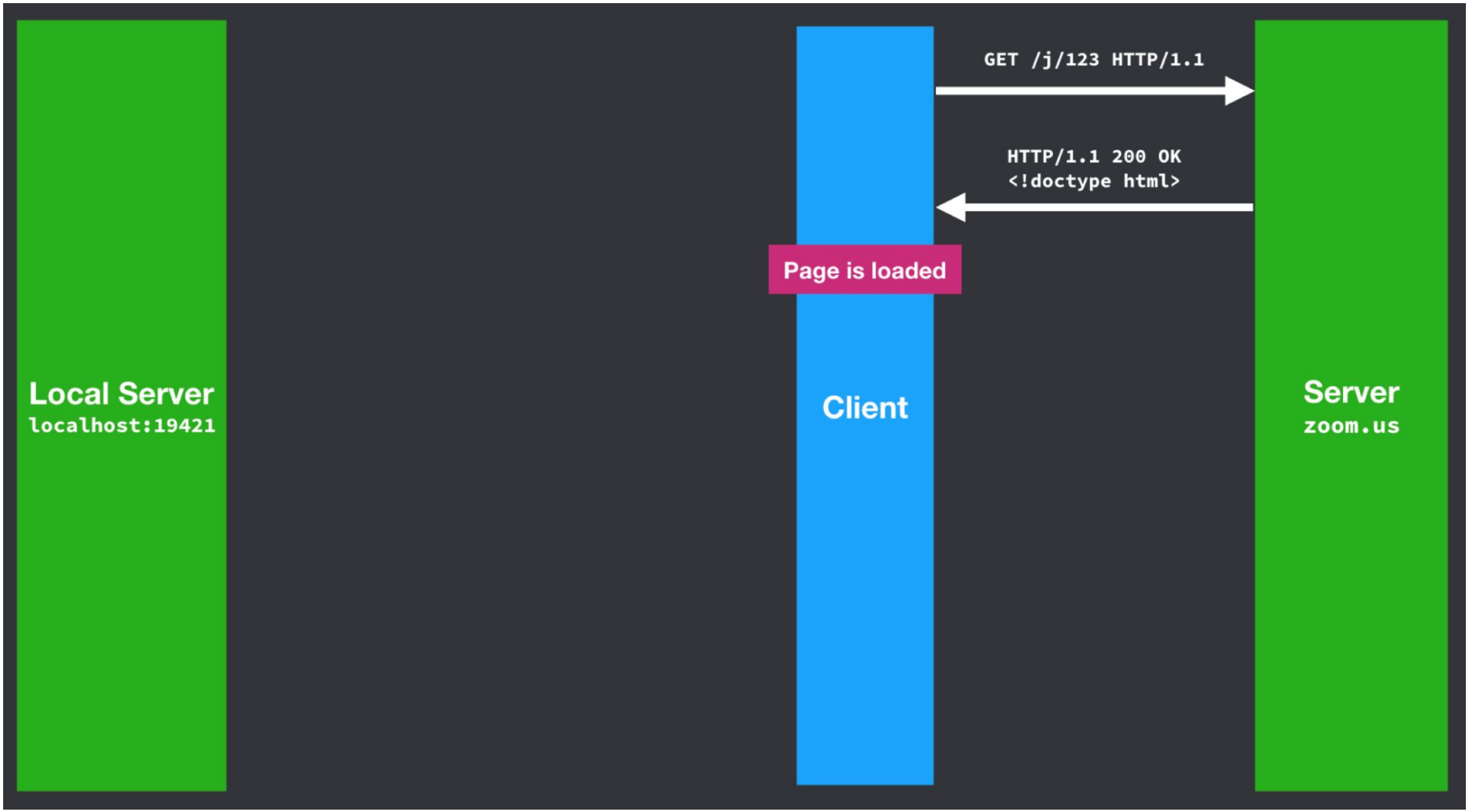
DNS rebinding attacks

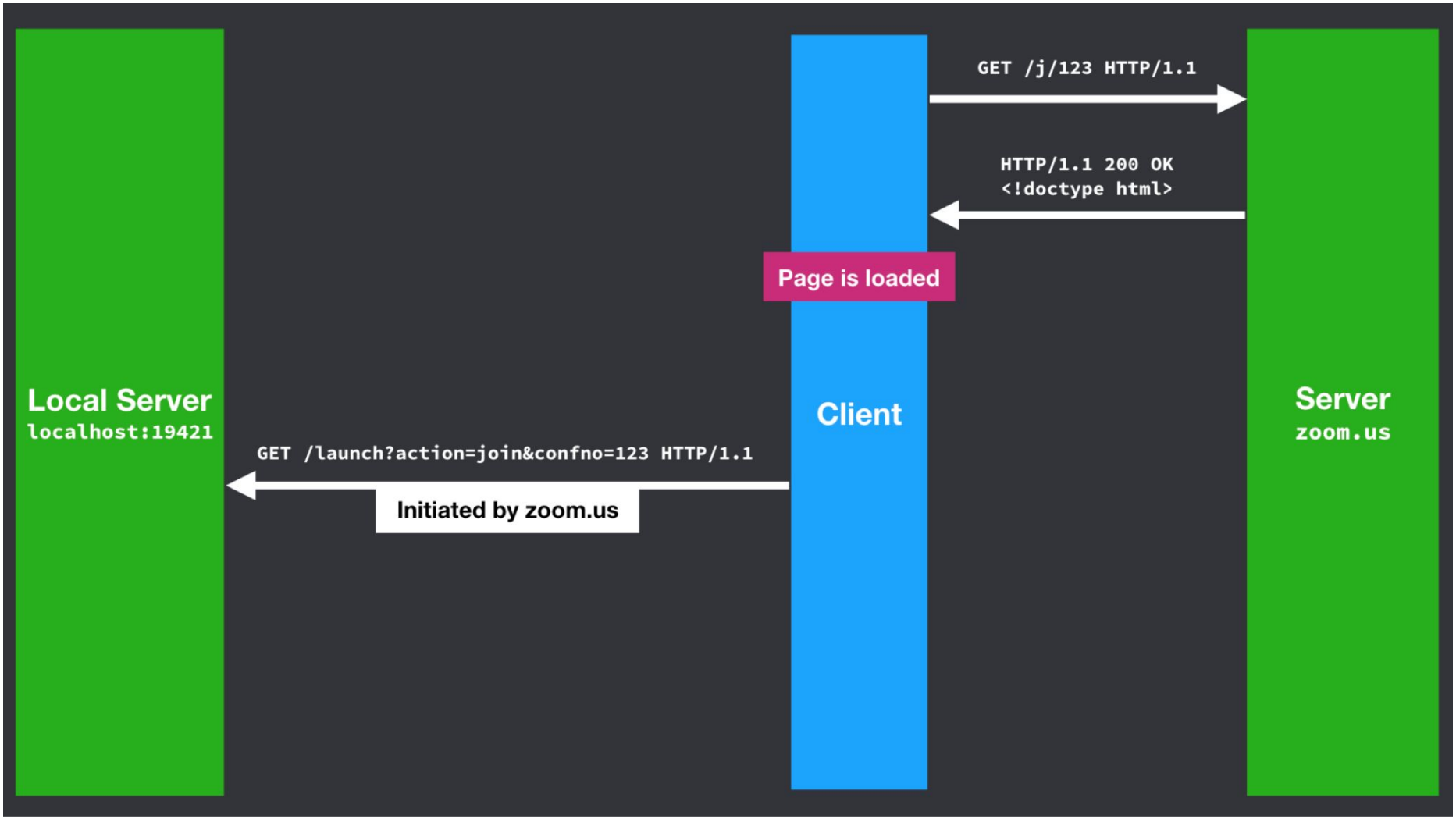
The Radio Thermostat CT50 & CT80 devices have by far the most consequential IoT device vulnerabilities I've found so far. These devices are some of the cheapest "smart" thermostats available on the market today. I purchased one to play with after being tipped off to their lack of security by CVE-2013–4860, which reported that the device had no form of authentication and could be controlled by anyone on the network. [...]

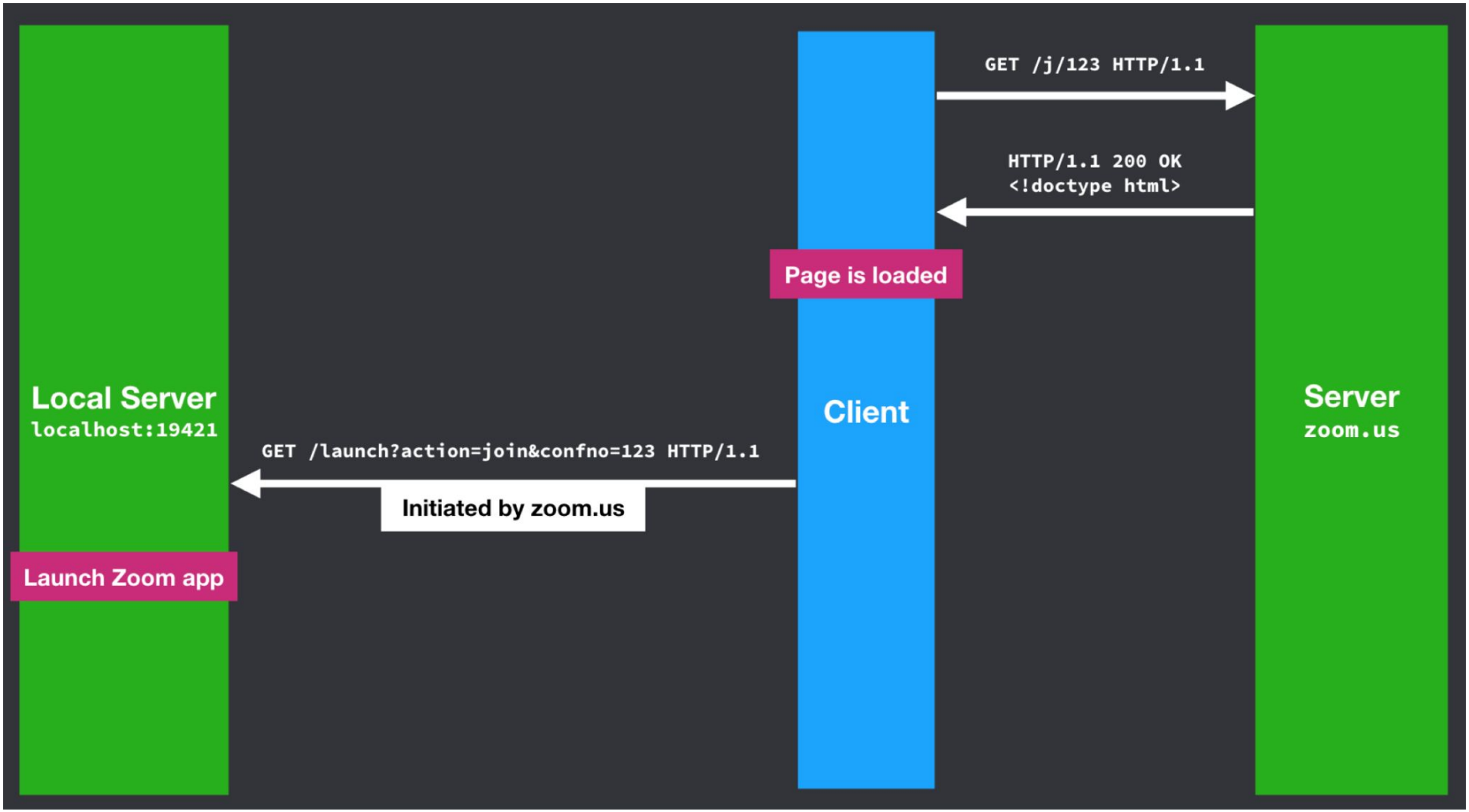
That assumption turned out to be correct and the thermostat's control API left the door wide open for DNS rebinding shenanigans. It's probably pretty obvious the kind of damage that can be done if your building's thermostat can be controlled by remote attackers. The PoC at http://rebind.network exfiltrates some basic information from the thermostat before setting the target temperature to 95° F. That temperature can be dangerous, or even deadly in the summer months to an elderly or disabled occupant. Not to mention that if your device is targeted while you're on vacation you could return home to a whopper of a utility bill.

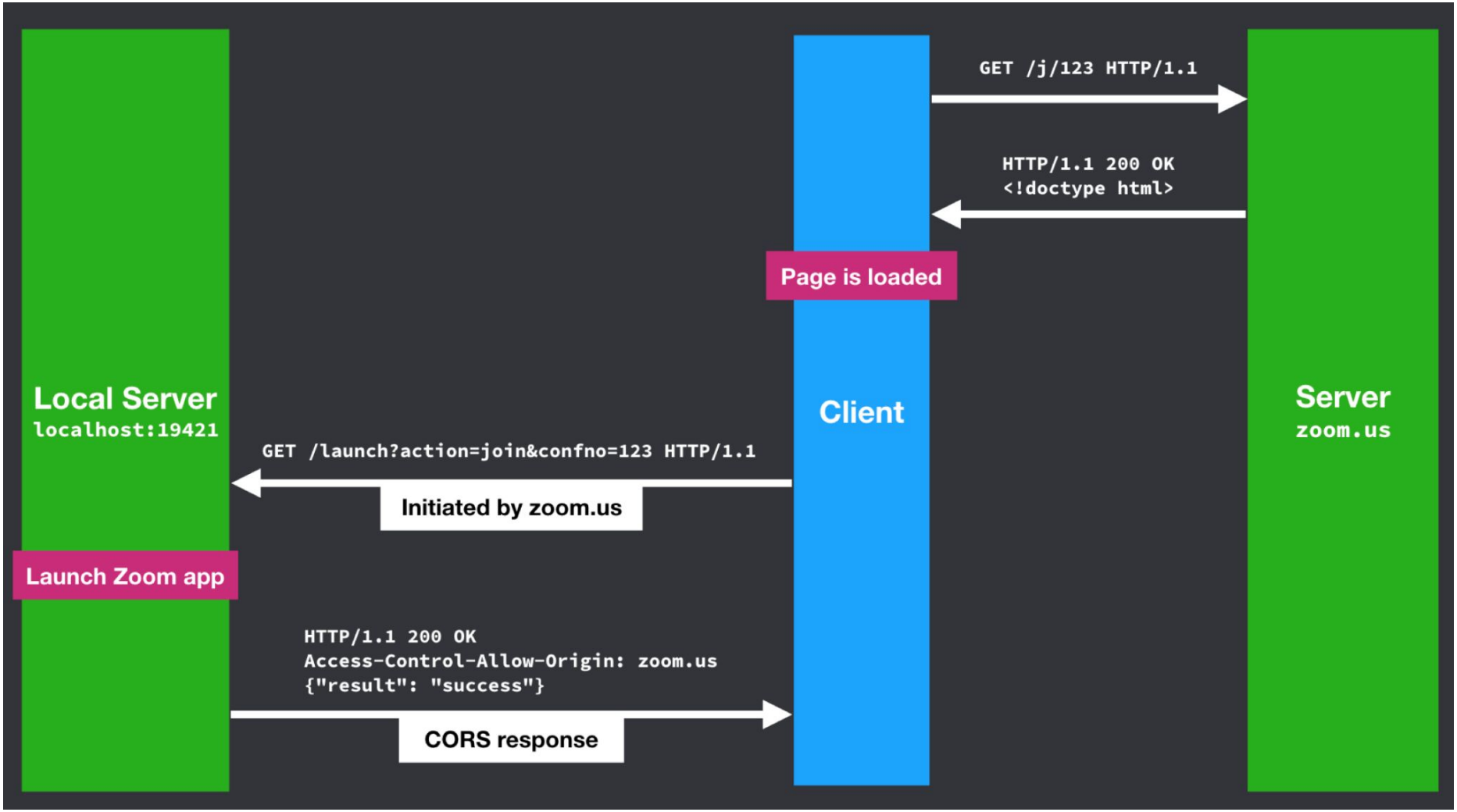
https://medium.com/@brannondorsey/attacking-private-networks-from-the-internet-with-dns-rebinding-ea7098a2d325

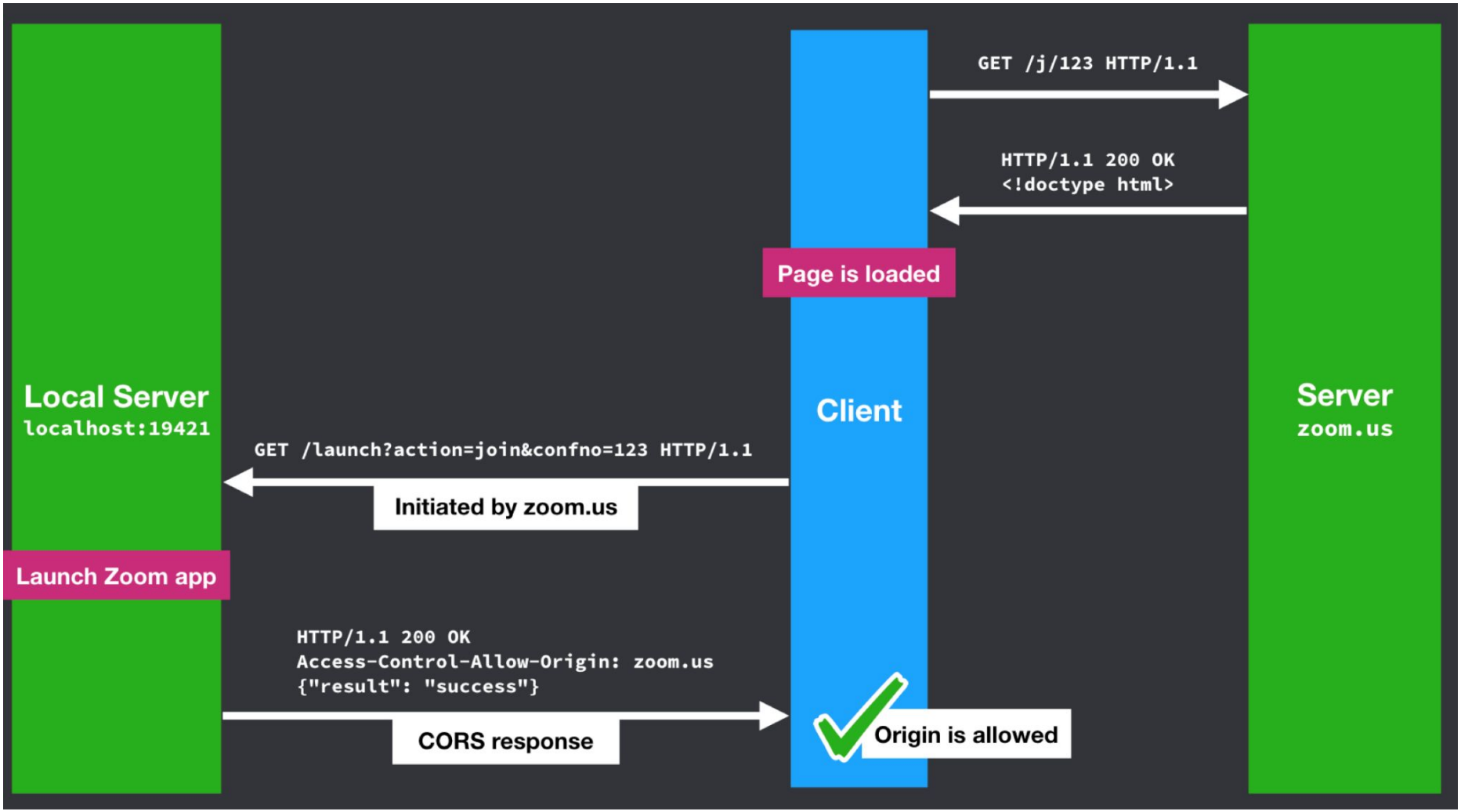
Recall from last week: User joins a zoom call (with CORS endpoint) (vulnerable)


Local Server
localhost:19421
Client
Server
zoom.us
GET /j/123 HTTP/1.1
HTTP/1.1 200 OK
<!doctype html>
Page is loaded

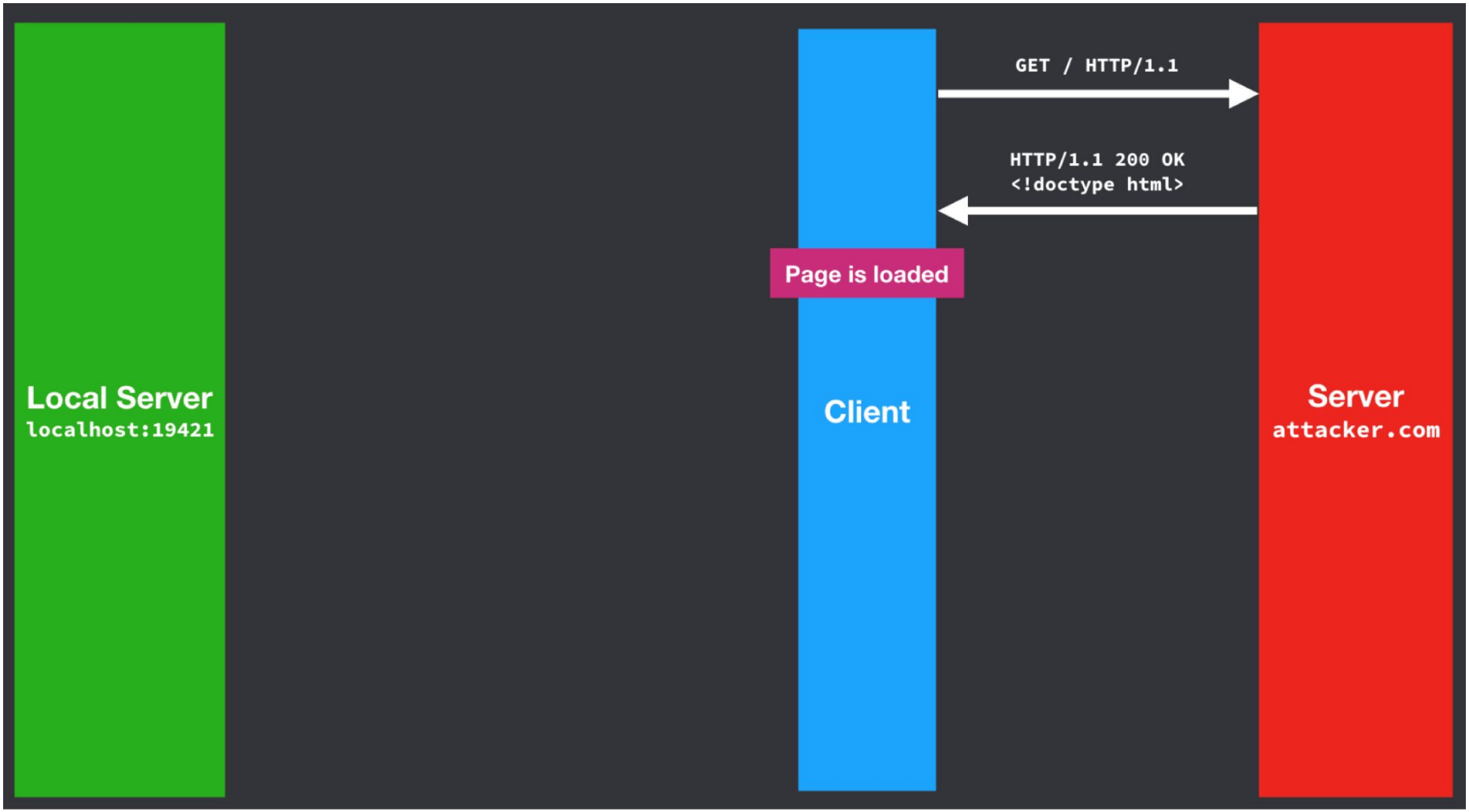

Local Server
localhost:19421
Client
Server
zoom.us
GET /j/123 HTTP/1.1
HTTP/1.1 200 OK
<!doctype html>
Page is loaded
GET /launch?action=join&confno=123 HTTP/1.1
Initiated by zoom.us

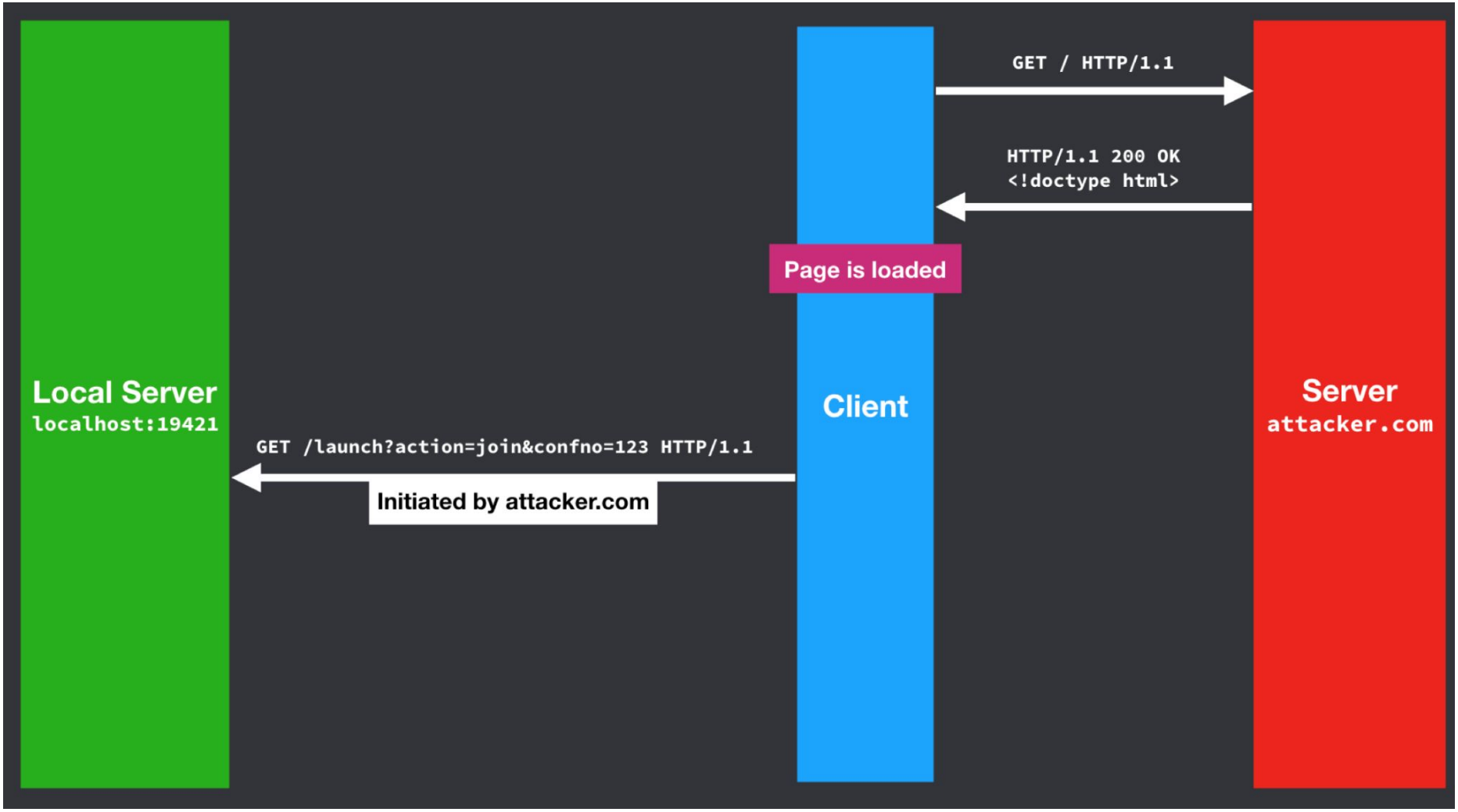

GET /j/123 HTTP/1.1
HTTP/1.1 200 OK
<!doctype html>
Page is loaded
Local Server
localhost:19421
Client
Server
zoom.us
GET /launch?action=join&confno=123 HTTP/1.1
Initiated by zoom.us
Launch Zoom app

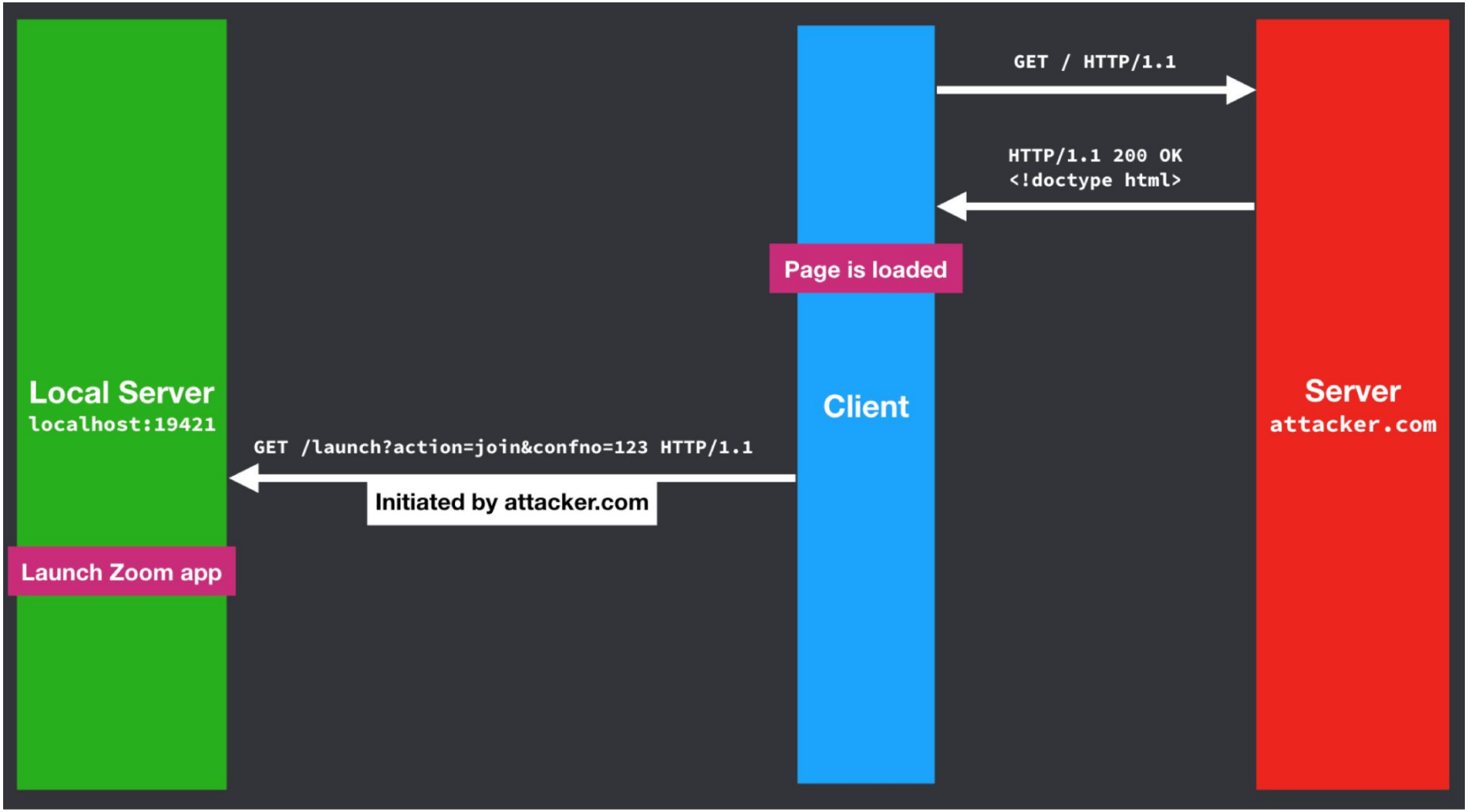

Local Server
localhost:19421
Client
Server
zoom.us
GET /j/123 HTTP/1.1
HTTP/1.1 200 OK
<!doctype html>
Page is loaded
GET /launch?action=join&confno=123 HTTP/1.1
Initiated by zoom.us
Launch Zoom app
HTTP/1.1 200 OK
Access-Control-Allow-Origin: zoom.us
{"result": "success"}
CORS response

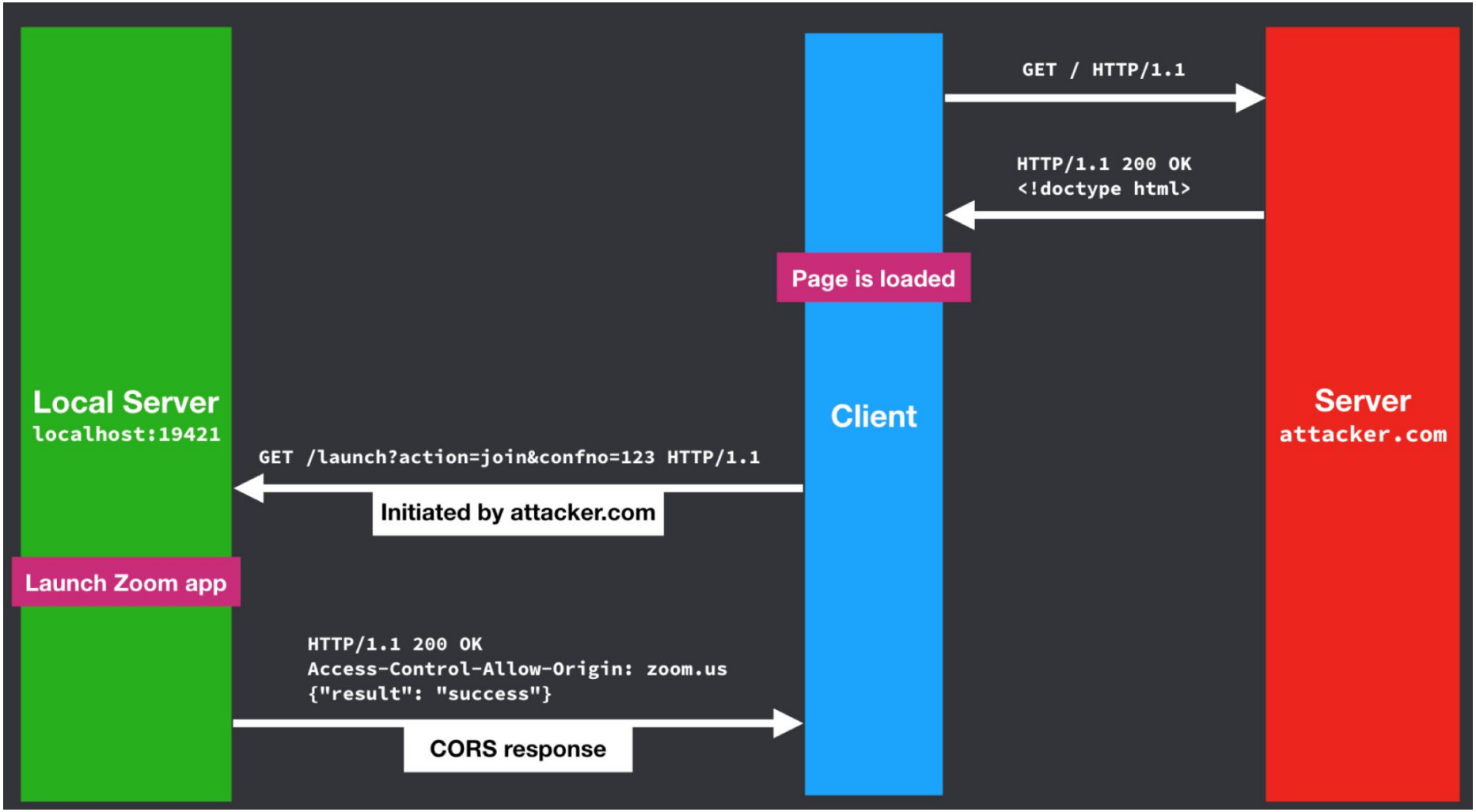

Local Server
localhost:19421
Client
Server
zoom.us
GET /j/123 HTTP/1.1
HTTP/1.1 200 OK
<!doctype html>
Page is loaded
GET /launch?action=join&confno=123 HTTP/1.1
Initiated by zoom.us
Launch Zoom app
HTTP/1.1 200 OK
Access-Control-Allow-Origin: zoom.us
{"result": "success"}
CORS response
Origin is allowed

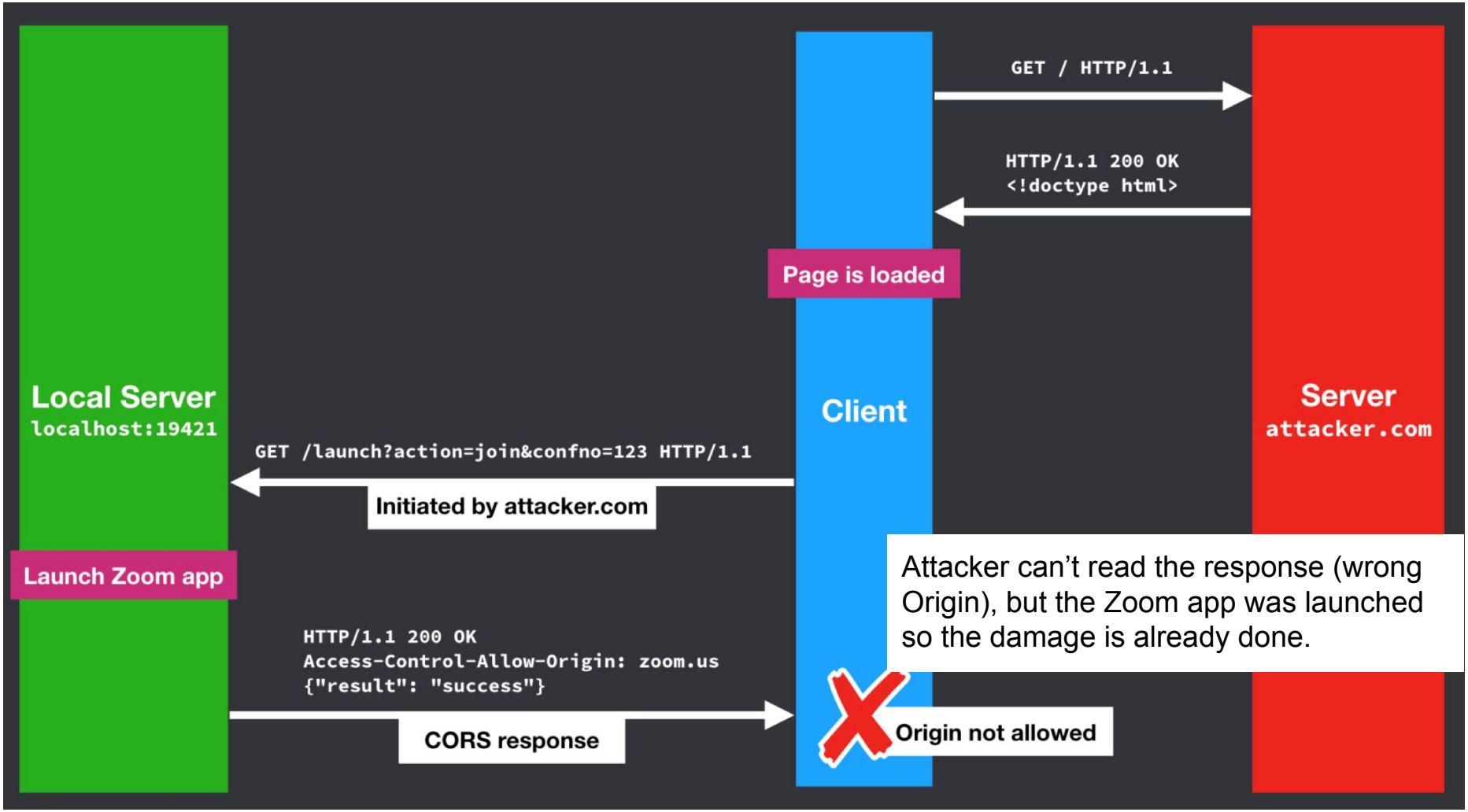
Attacker joins user into a zoom call (with CORS endpoint)


Local Server
localhost:19421
Client
Server
attacker.com
GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html>
Page is loaded


Local Server
localhost:19421
Client
Server
attacker.com
GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html>
Page is loaded
GET /launch?action=join&confno=123 HTTP/1.1
Initiated by attacker.com


GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html>
Page is loaded
Local Server
localhost:19421
Client
Server
attacker.com
GET /launch?action=join&confno=123 HTTP/1.1
Initiated by attacker.com
Launch Zoom app


GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html>
Page is loaded
Local Server
localhost:19421
Client
Server
attacker.com
GET /launch?action=join&confno=123 HTTP/1.1
Initiated by attacker.com
Launch Zoom app
HTTP/1.1 200 OK
Access-Control-Allow-Origin: zoom.us
{"result": "success"}
CORS response


Local Server
localhost:19421
Launch Zoom app
Client
Server
attacker.com
GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html>
Page is loaded
GET /launch?action=join&confno=123 HTTP/1.1
Initiated by attacker.com
HTTP/1.1 200 OK
Access-Control-Allow-Origin: zoom.us
{"result": "success"}
CORS response
Origin not allowed
Attacker can't read the response (wrong Origin), but the Zoom app was launched so the damage is already done.

Let's fix the issue

- Best solution: remove the local HTTP server and just register a `zoom://` scheme (protocol handler)
  - https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml
- However, let's assume we need to keep the local HTTP server (probably a bad idea)
- Ideas for securing it:
  - Require user interaction before joining, don't allow host to automatically enable video
  - Only allow zoom.us to communicate with the local server

"Simple" HTTP requests

- An HTTP/1.1 GET, HEAD or a POST is the request method
- In the case of a POST, the Content-Type of the request body is one of `application/x-www-form-urlencoded`, `multipart/form-data`, or `text/plain`
- No custom HTTP headers are set (or, only CORS-safelisted headers are set)
- In other words, “simple” requests are requests that your browser can make *without* using Javascript

"Preflighted" HTTP requests

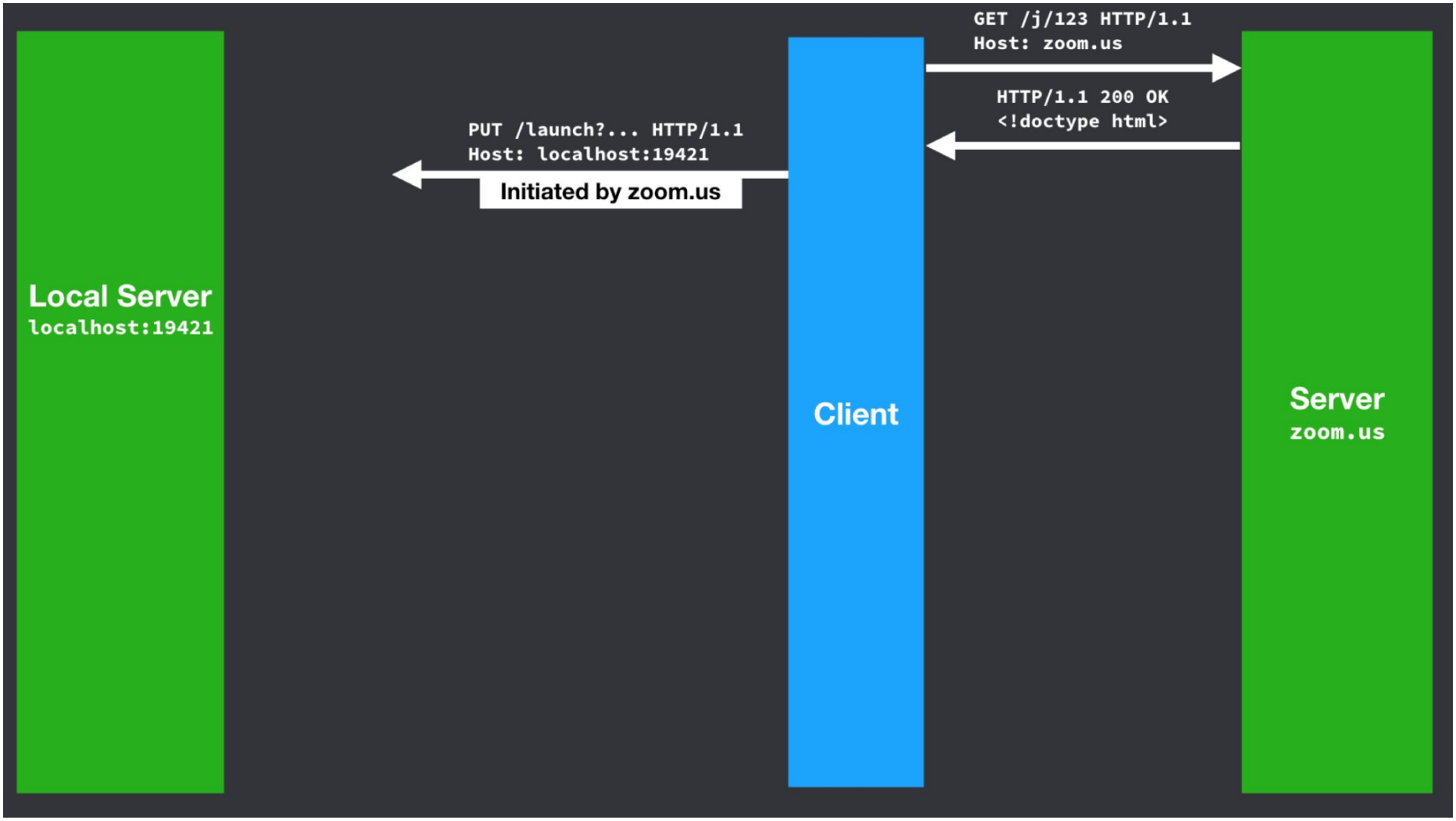
- Before a "preflighted" requests can be sent to the target server, the browser must check that it is safe to send
- So it first sends an HTTP request with the `OPTIONS` method to the same URL
- If server doesn't support `OPTIONS` (either because it is old or because it doesn't want to support preflighted requests) then, preflighted requests are denied

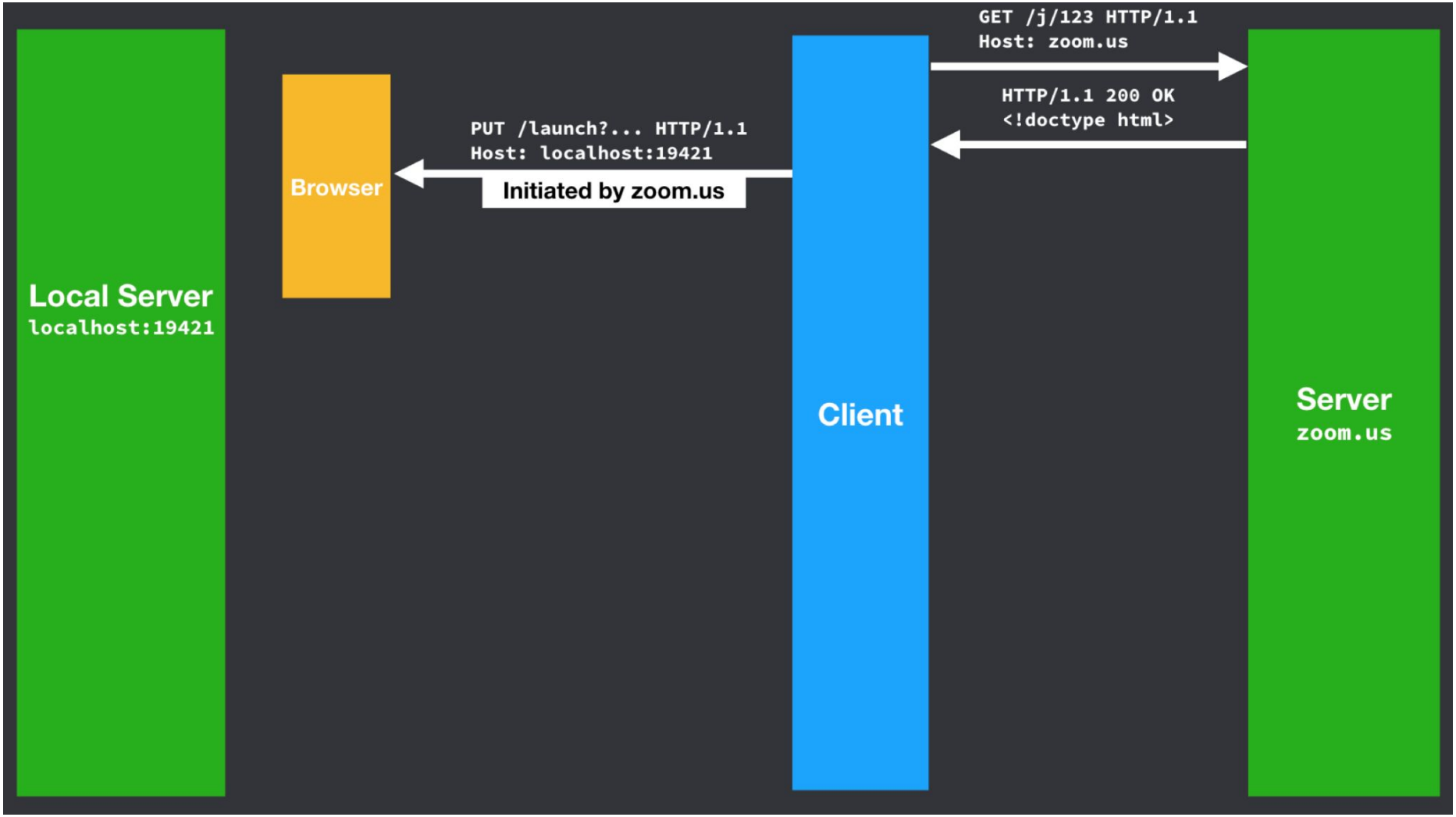
User joins a zoom call (local server requires “preflighted” request

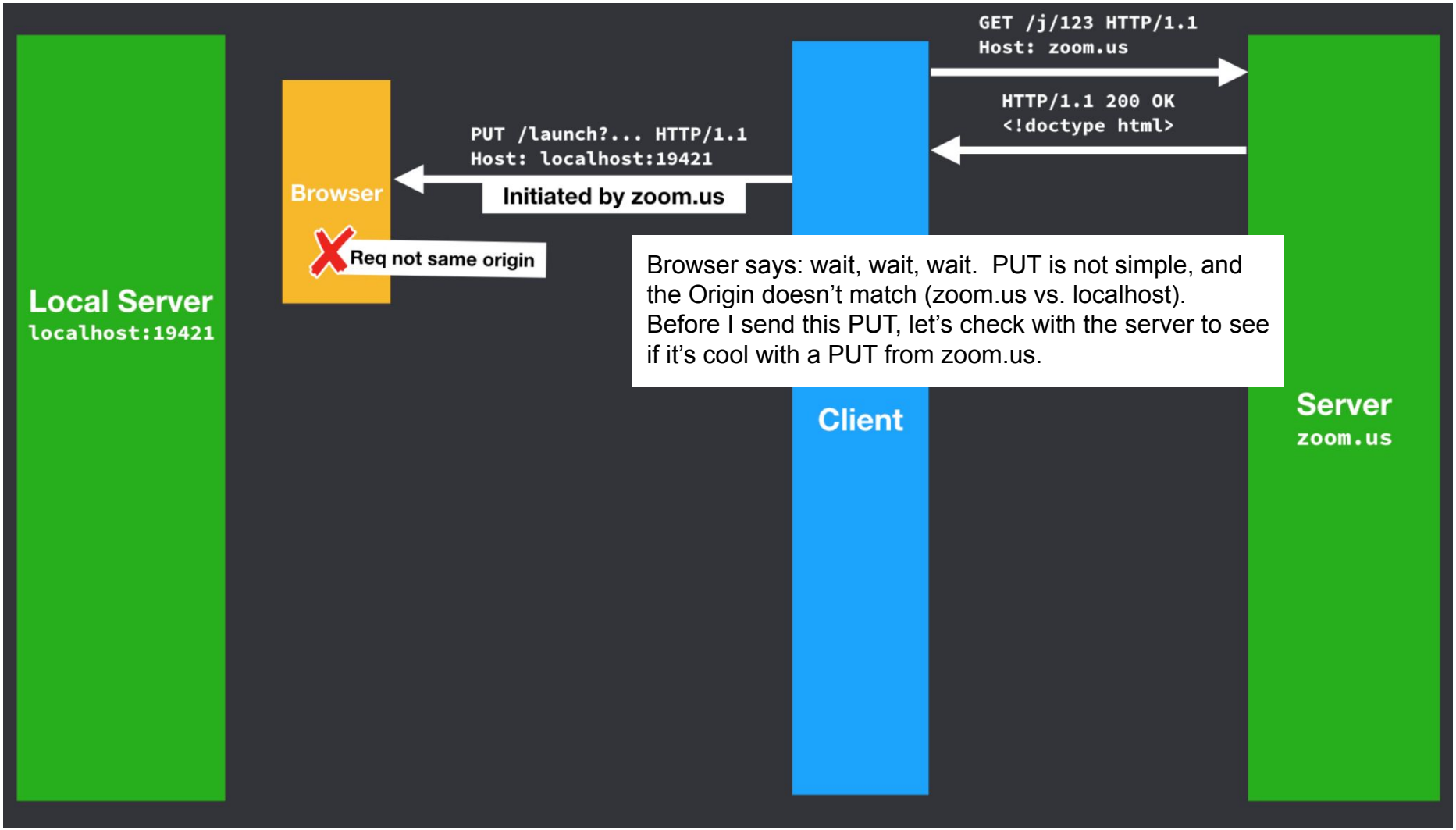

Local Server
localhost:19421
Client
Server
zoom.us

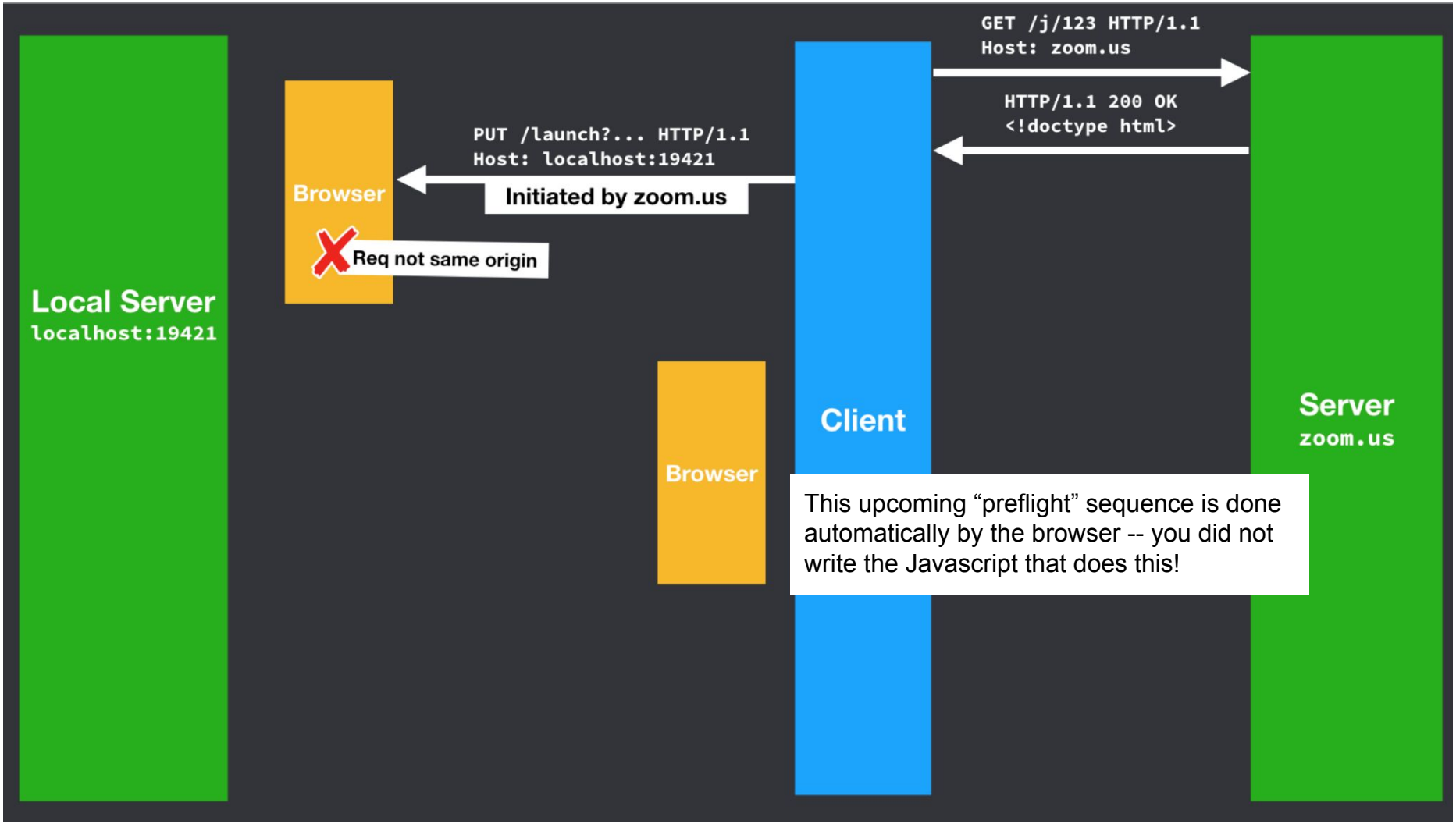

GET /j/123 HTTP/1.1
Host: zoom.us
Local Server
localhost:19421
Client
Server
zoom.us

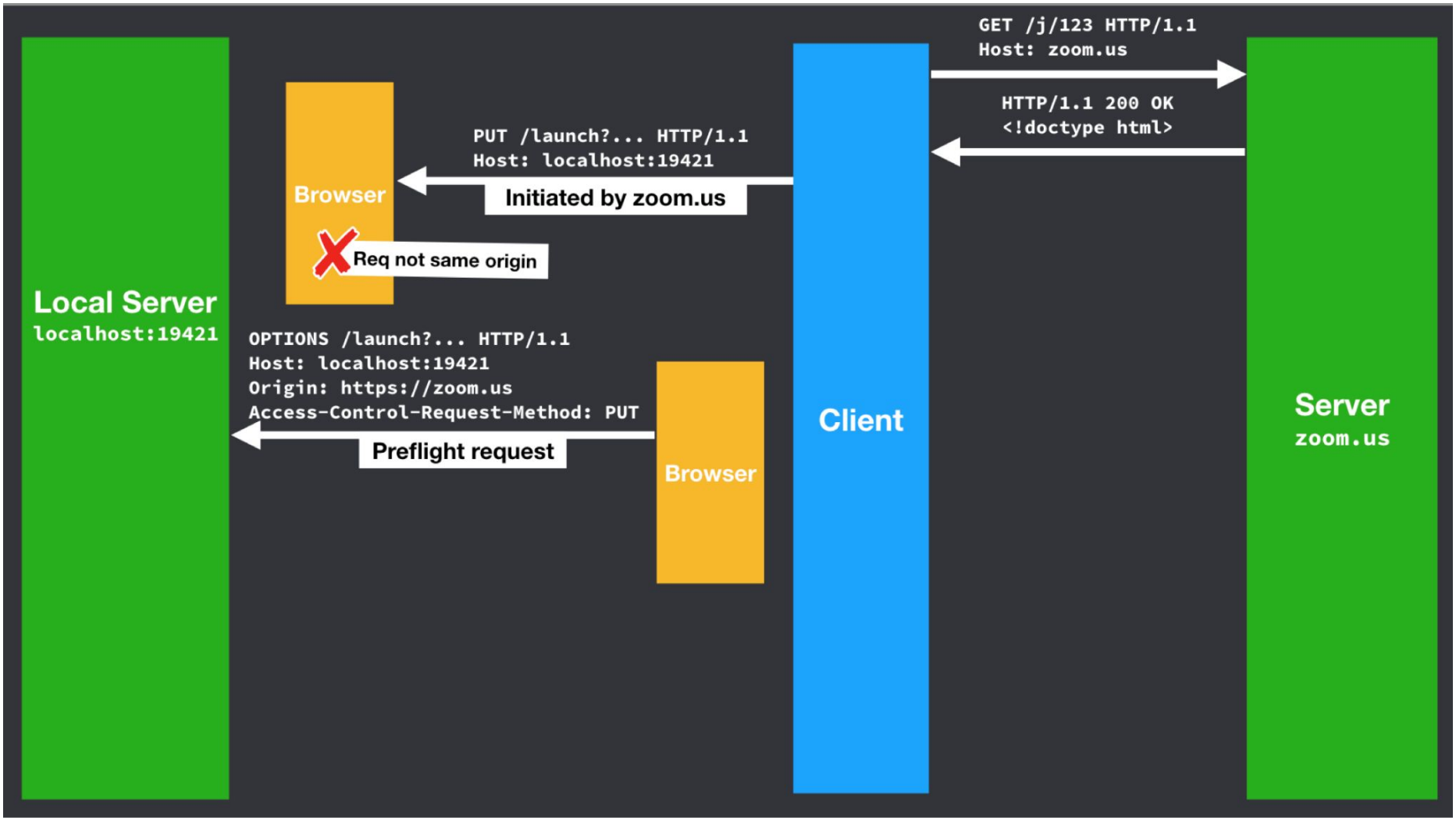

GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
Local Server
localhost:19421
Client
Server
zoom.us

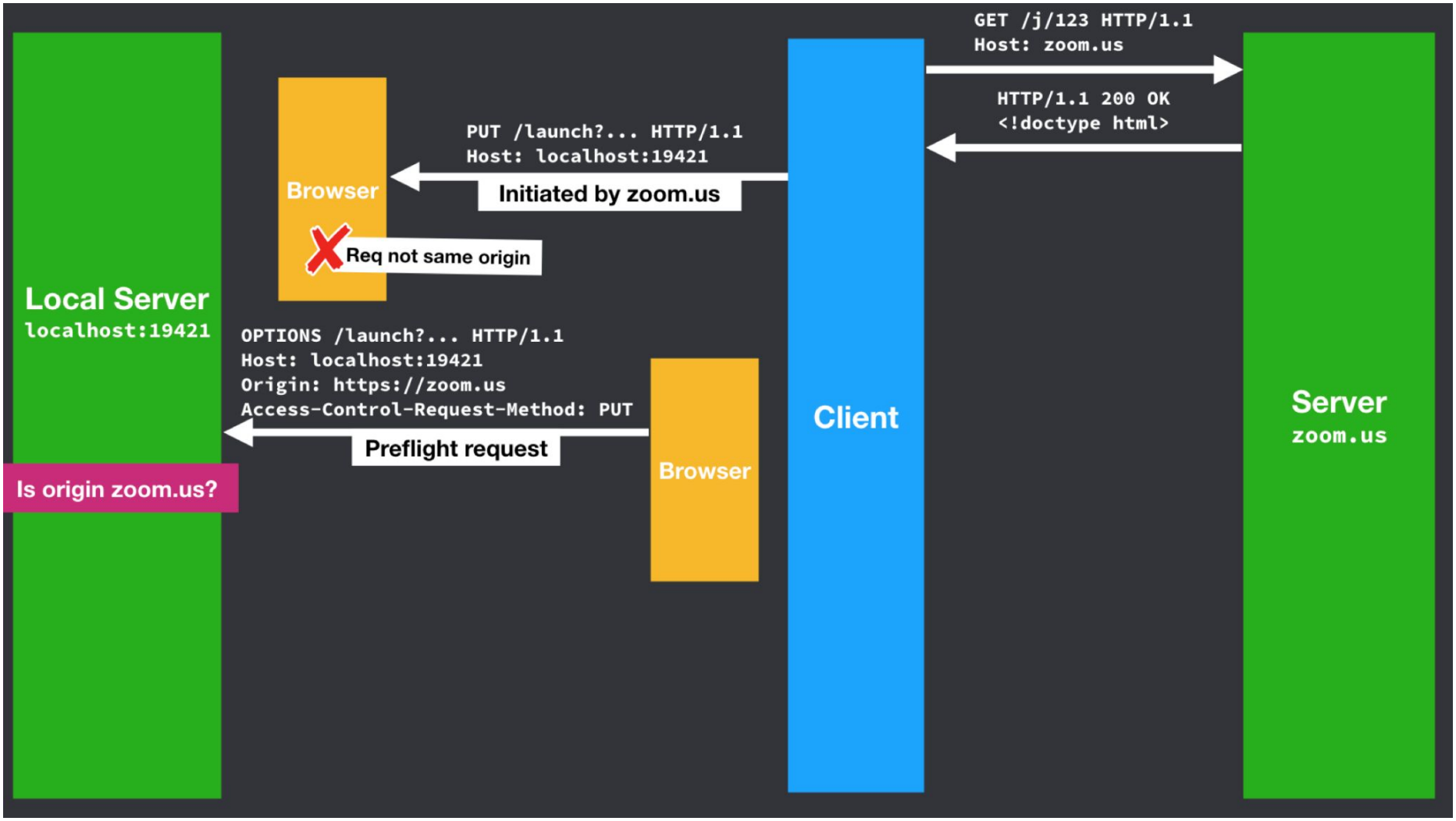

GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Local Server
localhost:19421
Client
Server
zoom.us

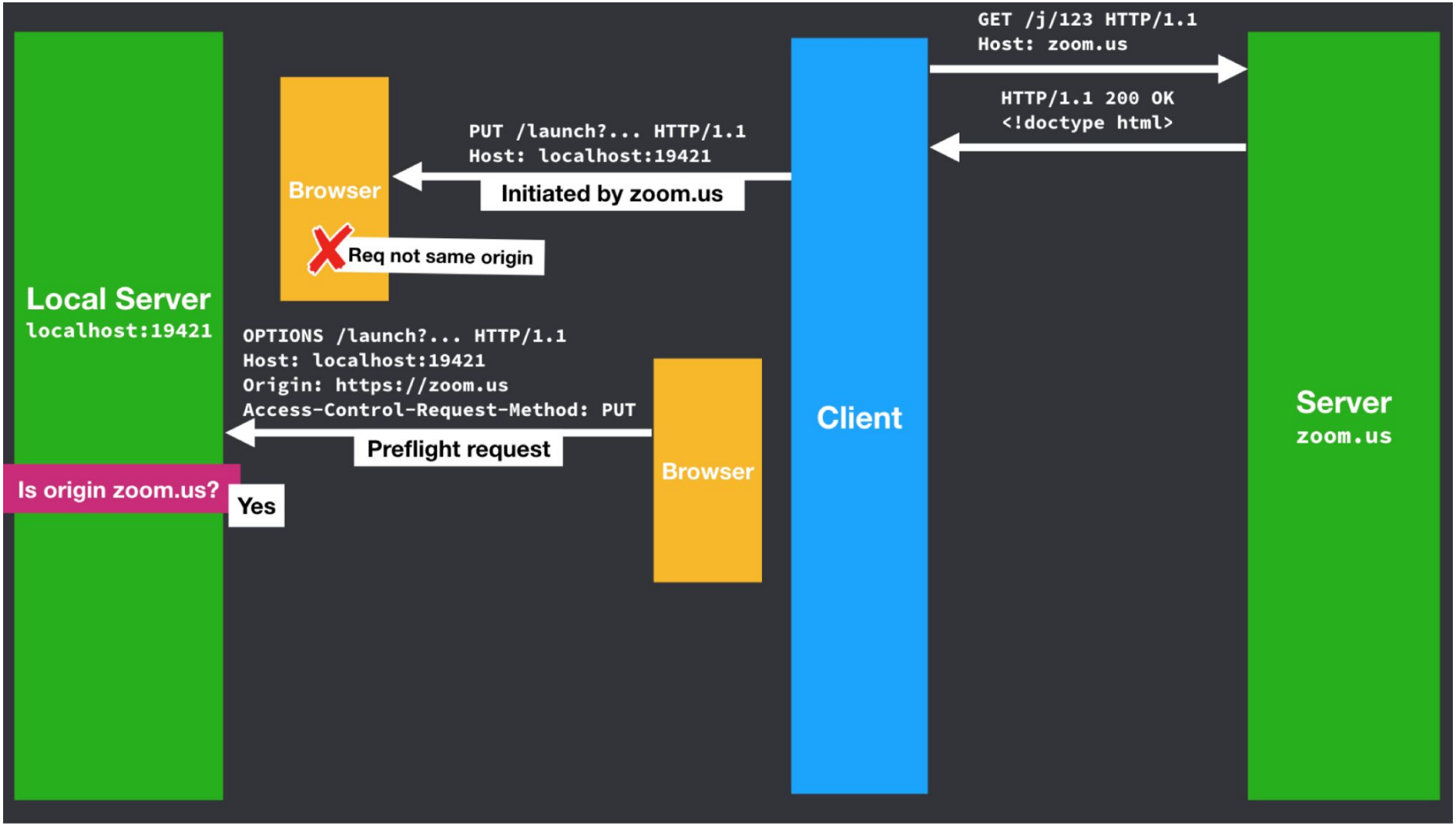

GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Browser
Local Server
localhost:19421
Client
Server
zoom.us

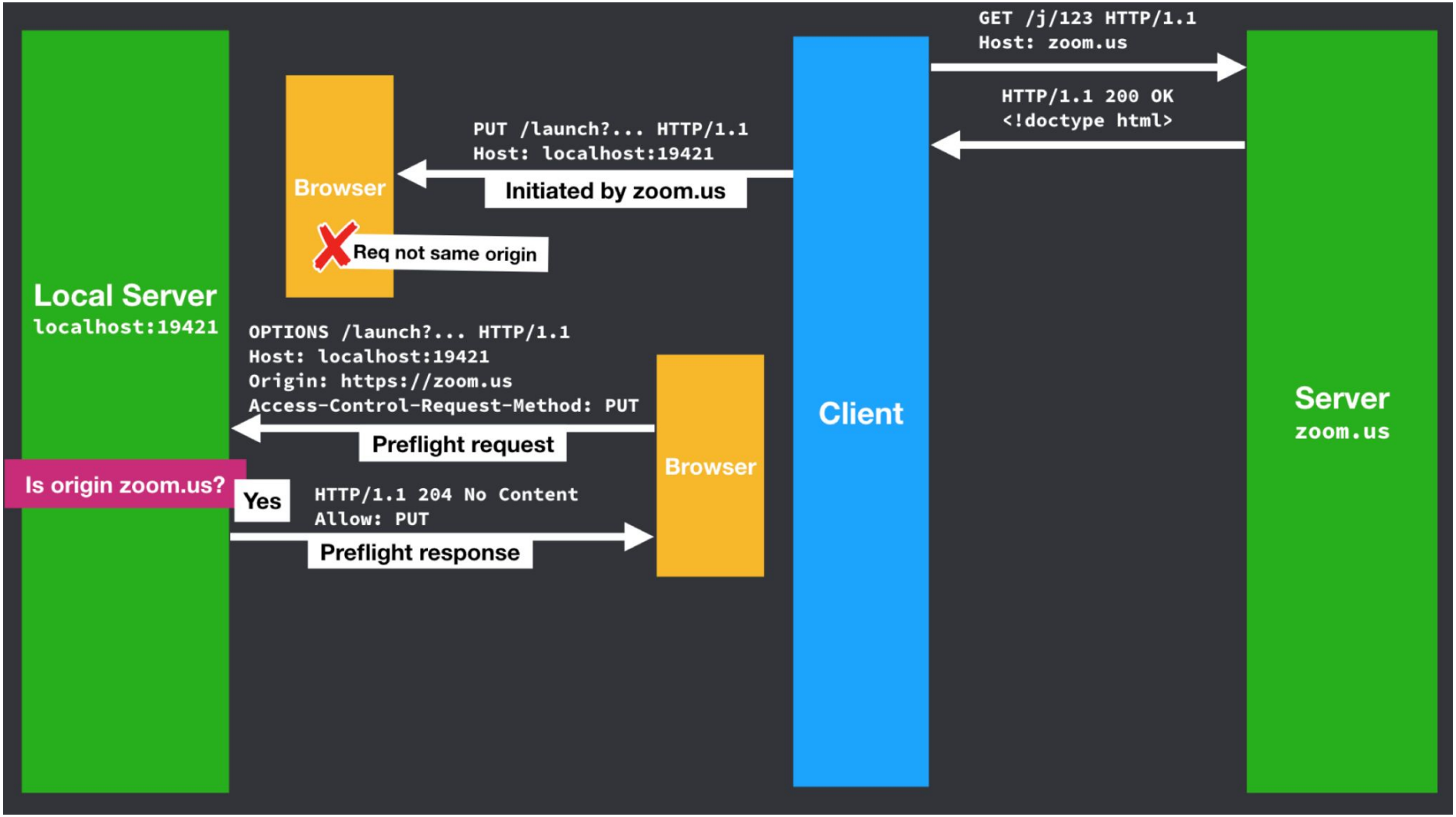

GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Browser
Req not same origin
Local Server
localhost:19421
Client
Server
zoom.us
Browser says: wait, wait, wait. PUT is not simple, and the Origin doesn't match (zoom.us vs. localhost). Before I send this PUT, let's check with the server to see if it's cool with a PUT from zoom.us.

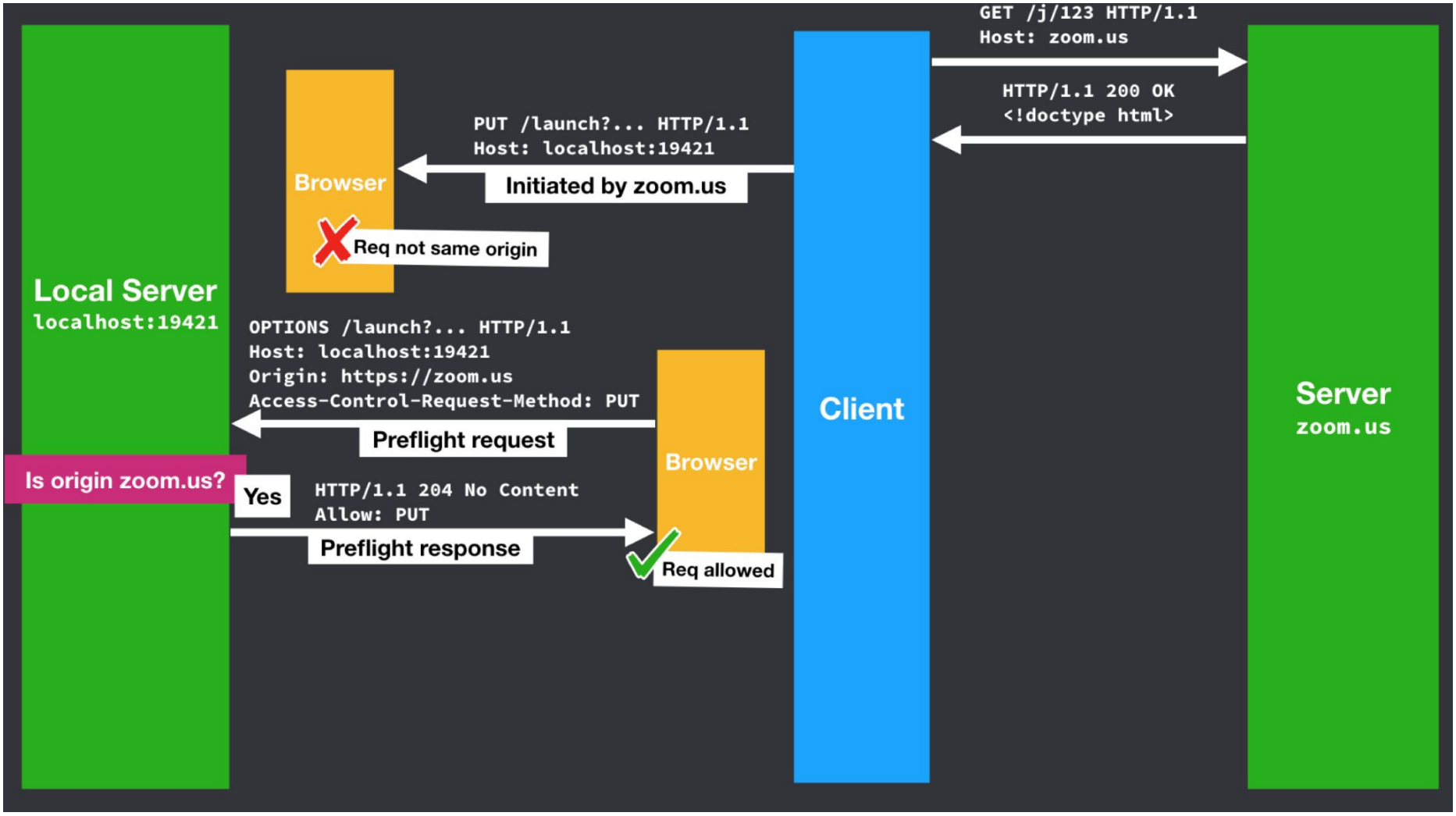

GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Browser
Req not same origin
Local Server
localhost:19421
Browser
Client
Server
zoom.us
This upcoming “preflight” sequence is done automatically by the browser -- you did not write the Javascript that does this!

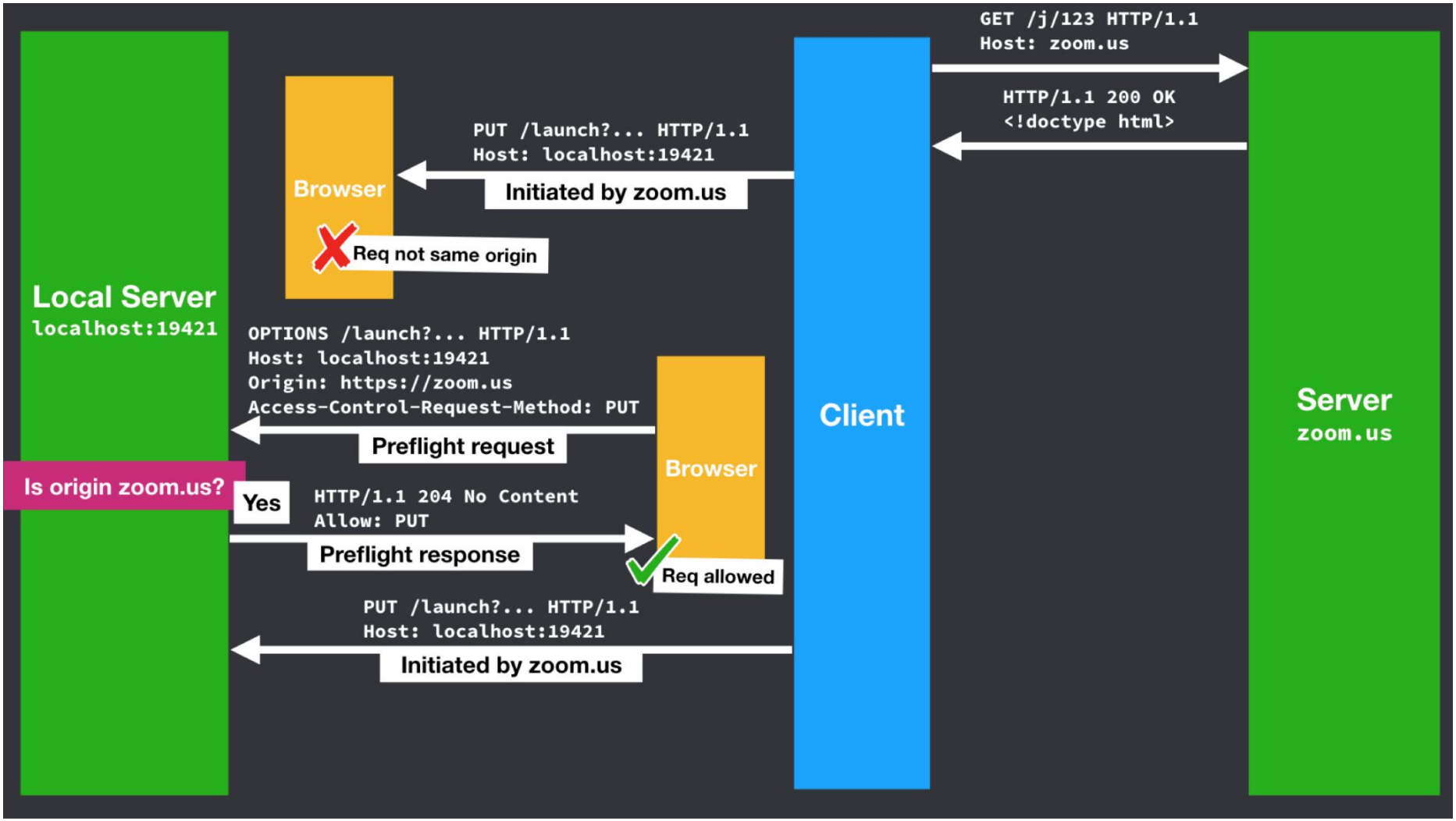

GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
Server
zoom.us
Client
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Browser
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us
Access-Control-Request-Method: PUT
Preflight request
Browser

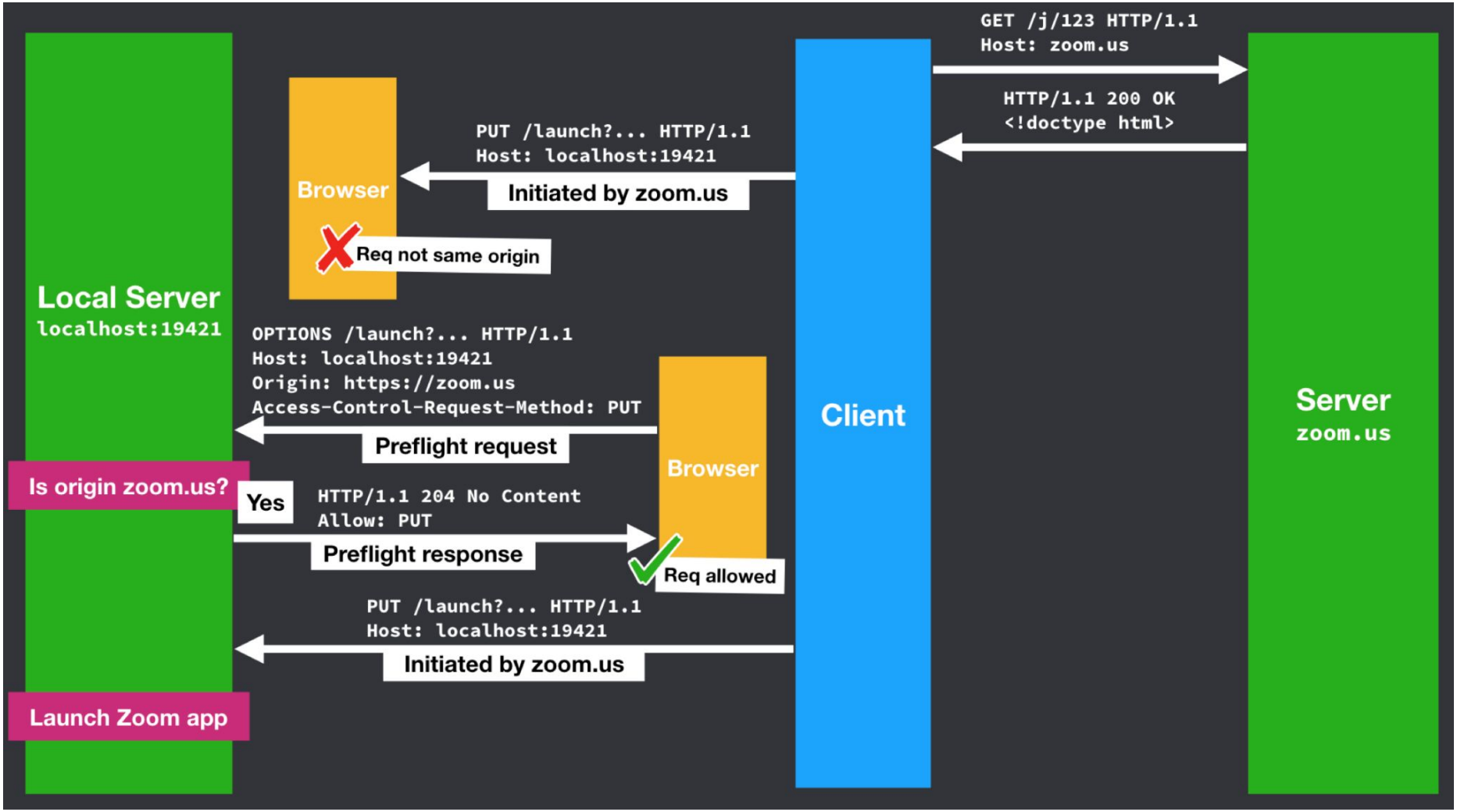

GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
Server
zoom.us
Client
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Browser
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us
Access-Control-Request-Method: PUT
Preflight request
Browser
Is origin zoom.us?

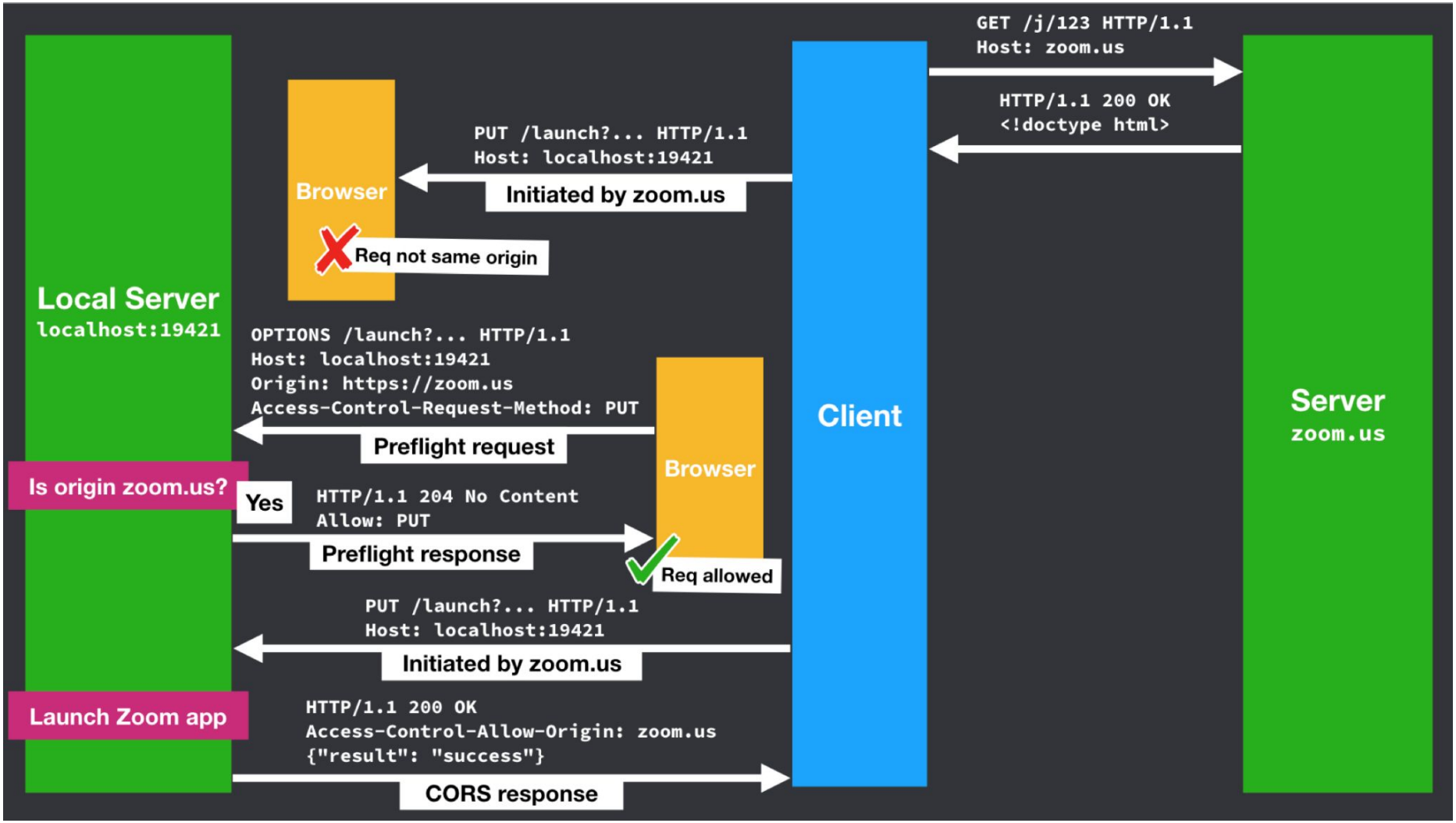

GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
Server
zoom.us
Client
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Browser
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us
Access-Control-Request-Method: PUT
Preflight request
Browser
Is origin zoom.us?
Yes

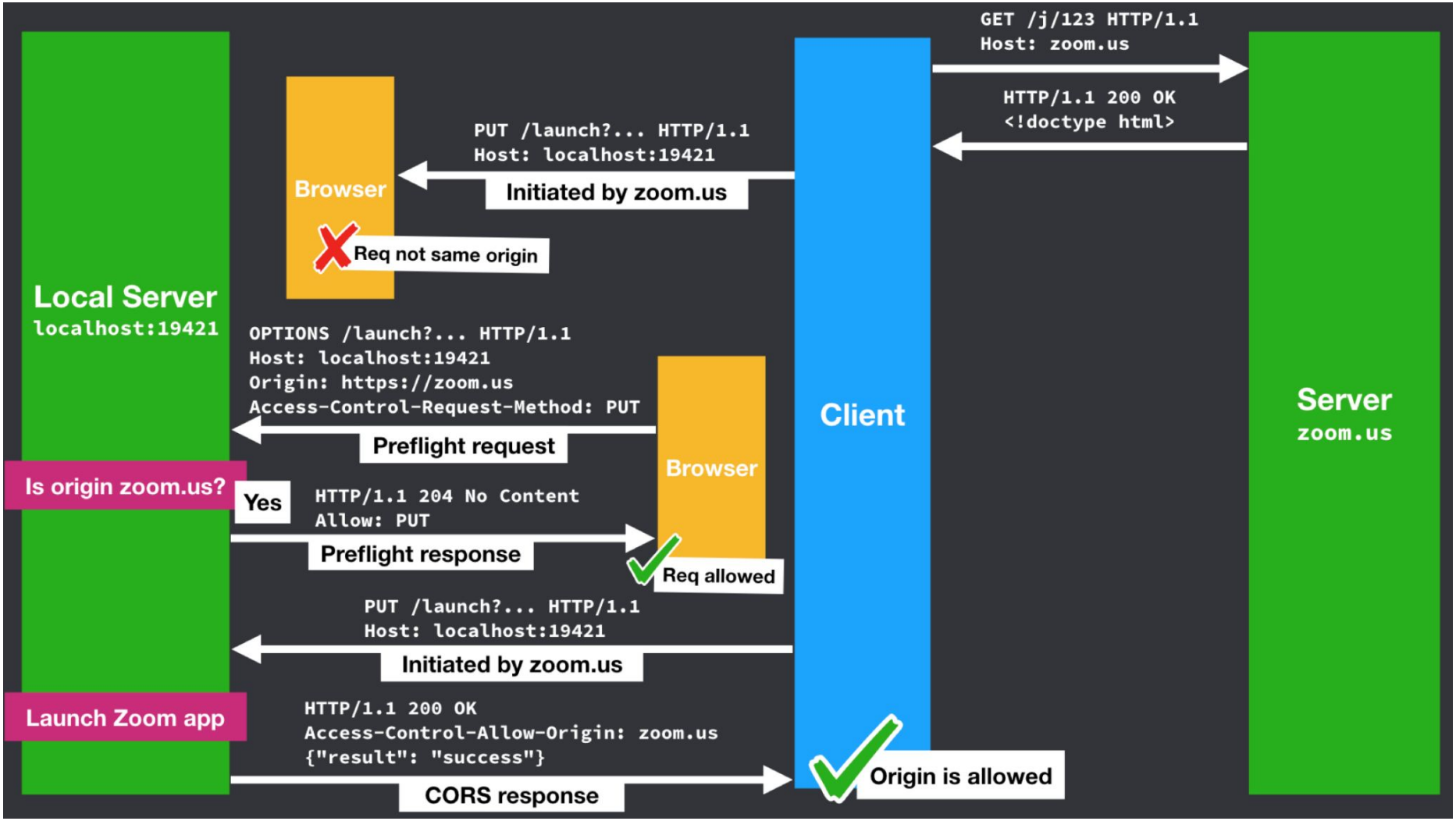

GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
Server
zoom.us
Client
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Browser
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us
Access-Control-Request-Method: PUT
Preflight request
Browser
Is origin zoom.us?
Yes
HTTP/1.1 204 No Content
Allow: PUT
Preflight response


GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
Server
zoom.us
Client
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Browser
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us
Access-Control-Request-Method: PUT
Preflight request
Browser
Is origin zoom.us?
Yes
HTTP/1.1 204 No Content
Allow: PUT
Preflight response
Req allowed

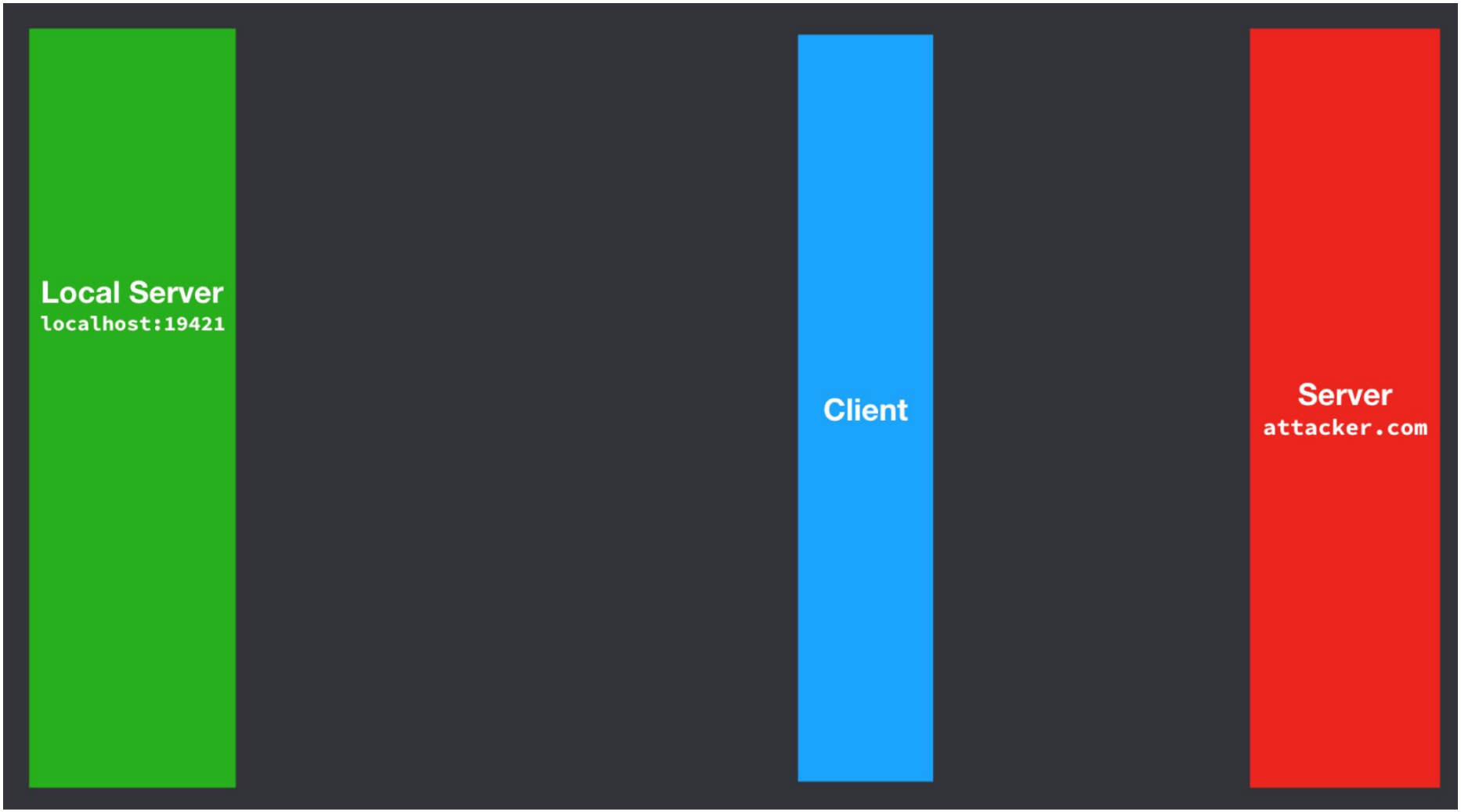

GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
Server
zoom.us
Client
Local Server
localhost:19421
Browser
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Req not same origin
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us
Access-Control-Request-Method: PUT
Preflight request
Browser
Is origin zoom.us?
Yes
HTTP/1.1 204 No Content
Allow: PUT
Preflight response
Req allowed
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us


GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
Server
zoom.us
Client
Browser
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us
Access-Control-Request-Method: PUT
Preflight request
Browser
Is origin zoom.us?
Yes
HTTP/1.1 204 No Content
Allow: PUT
Preflight response
Req allowed
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Launch Zoom app


GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
Server
zoom.us
Client
Local Server
localhost:19421
Browser
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Req not same origin
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us
Access-Control-Request-Method: PUT
Preflight request
Browser
Is origin zoom.us?
Yes
HTTP/1.1 204 No Content
Allow: PUT
Preflight response
Req allowed
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Launch Zoom app
HTTP/1.1 200 OK
Access-Control-Allow-Origin: zoom.us
{"result": "success"}
CORS response

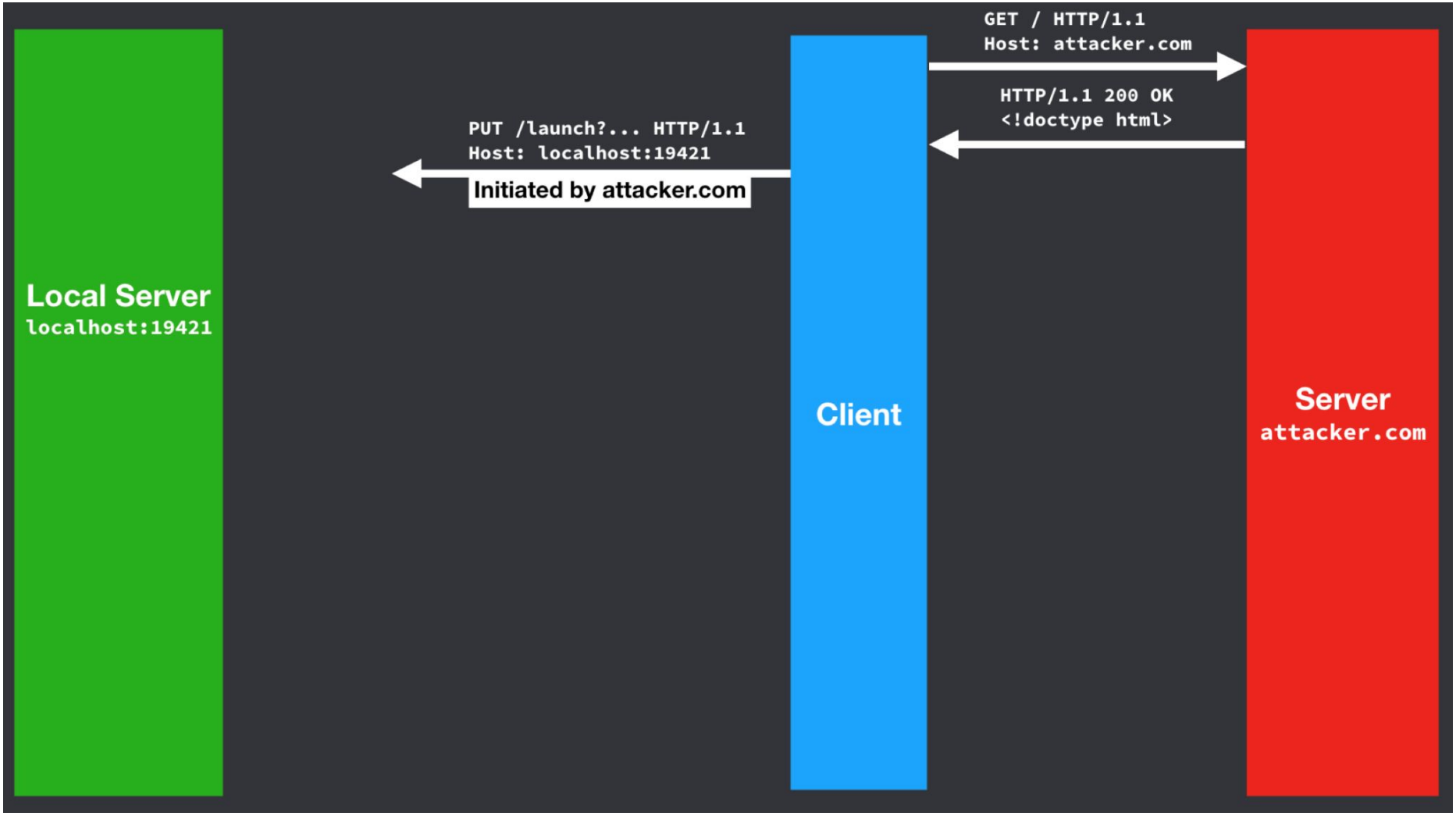

GET /j/123 HTTP/1.1
Host: zoom.us
HTTP/1.1 200 OK
<!doctype html>
Server
zoom.us
Client
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Browser
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us
Access-Control-Request-Method: PUT
Preflight request
Browser
Is origin zoom.us?
Yes
HTTP/1.1 204 No Content
Allow: PUT
Preflight response
Req allowed
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us
Launch Zoom app
HTTP/1.1 200 OK
Access-Control-Allow-Origin: zoom.us
{"result": "success"}
CORS response
Origin is allowed

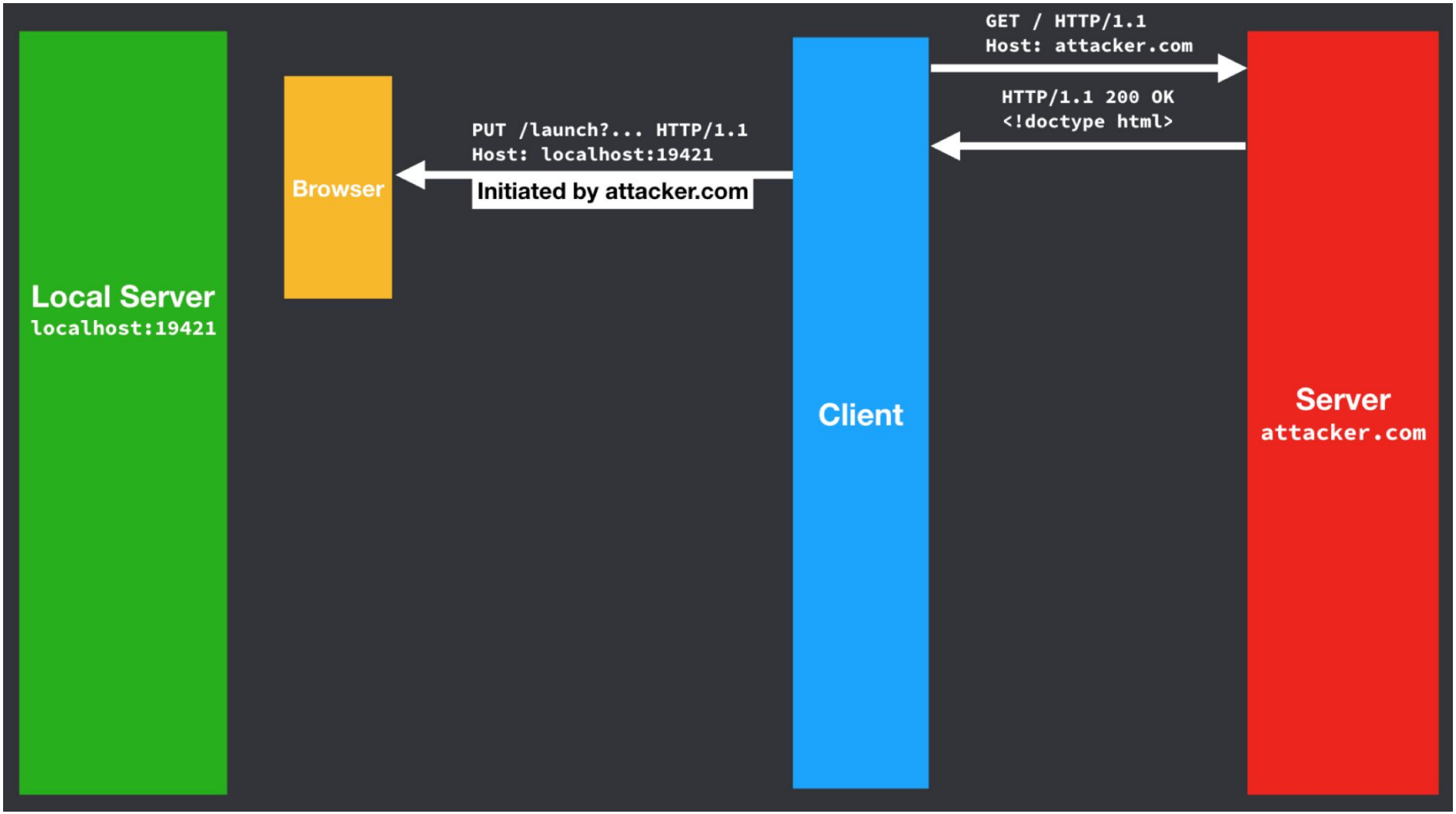
Attacker joins a zoom call (local server requires “preflighted” request

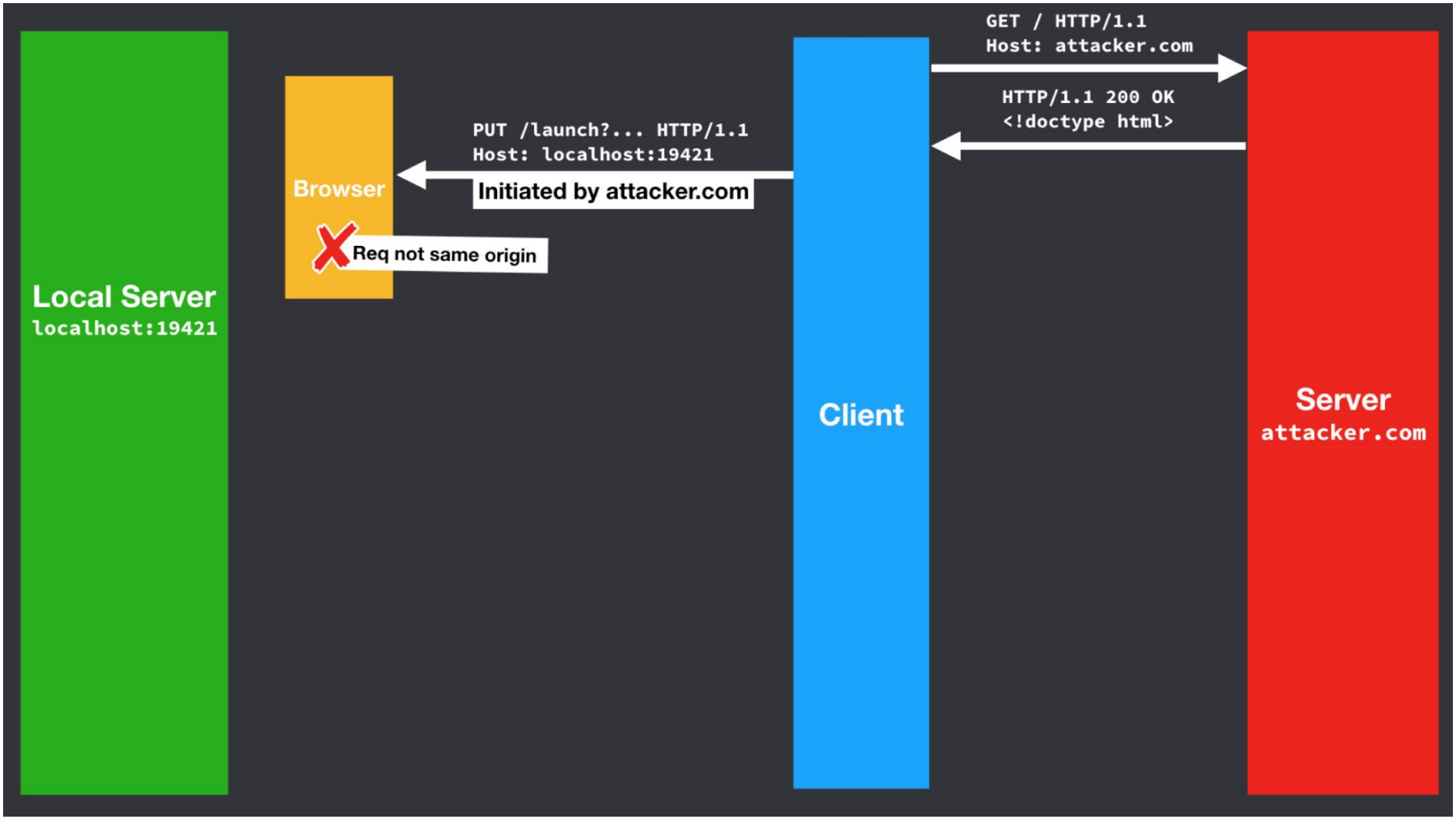

Local Server
localhost:19421
Client
Server
attacker.com

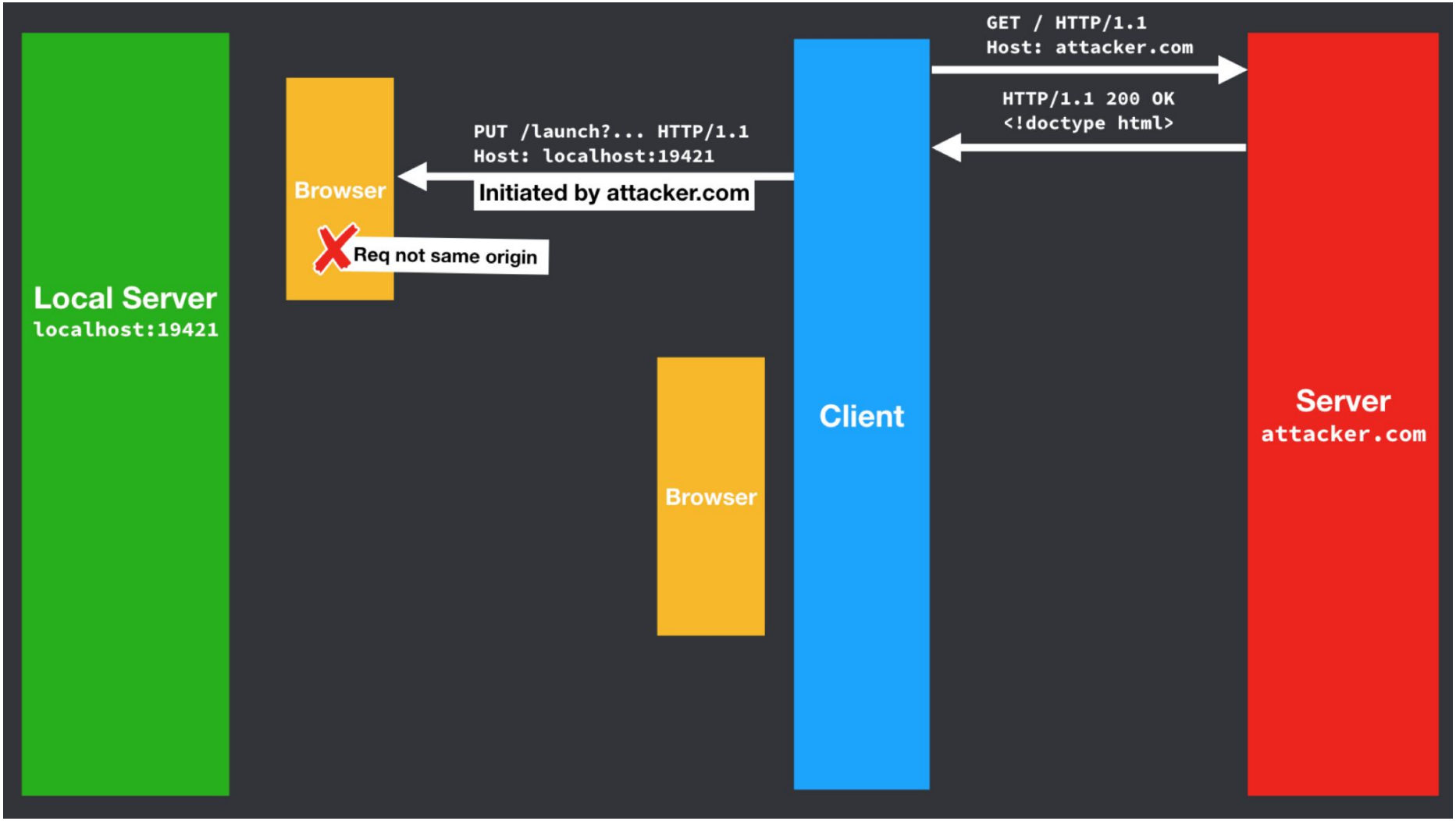

GET / HTTP/1.1
Host: attacker.com
Local Server
localhost:19421
Client
Server
attacker.com

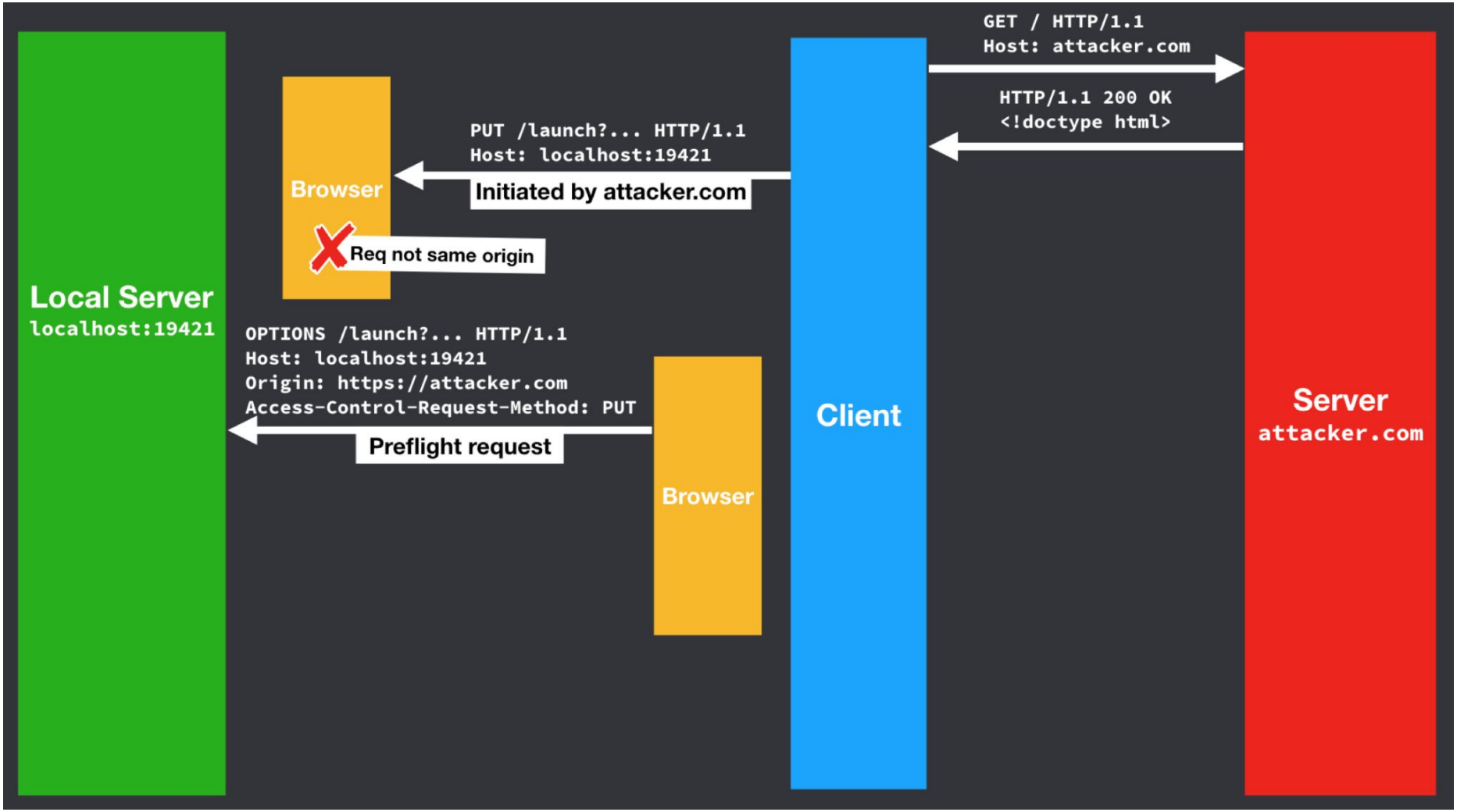

Local Server
localhost:19421
Client
GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
Server
attacker.com

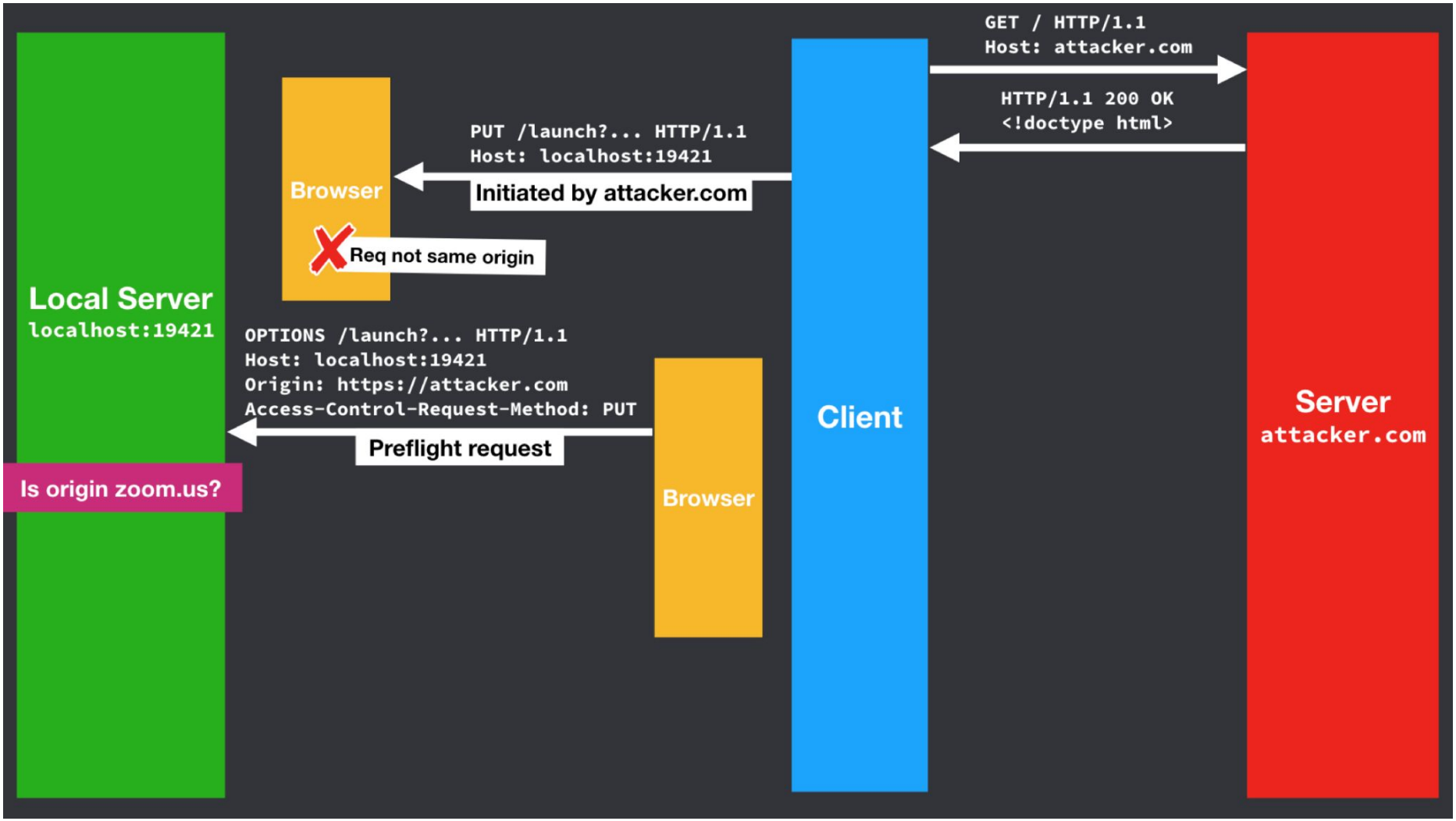

GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by attacker.com
Local Server
localhost:19421
Client
Server
attacker.com

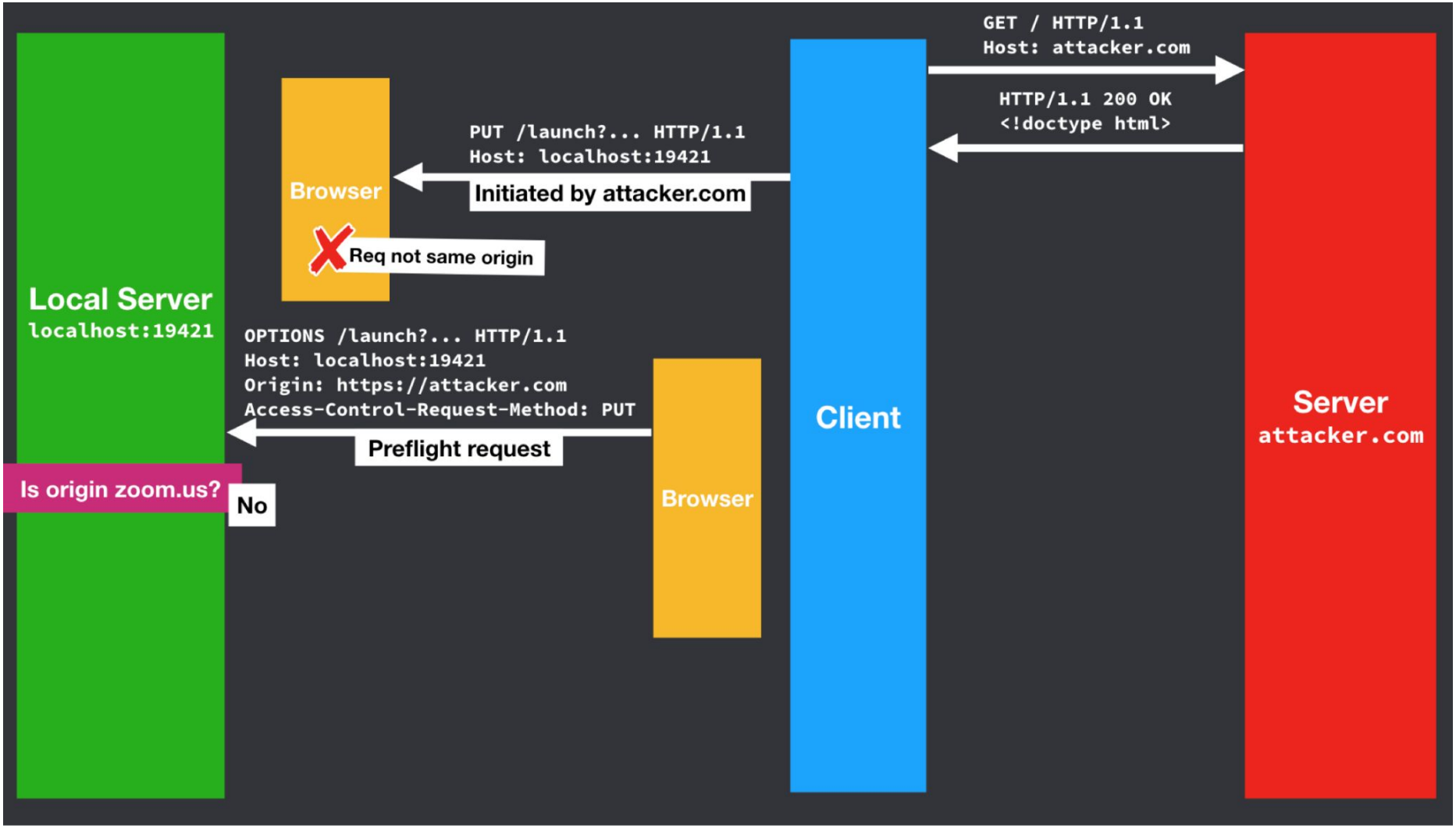

GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by attacker.com
Browser
Local Server
localhost:19421
Client
Server
attacker.com

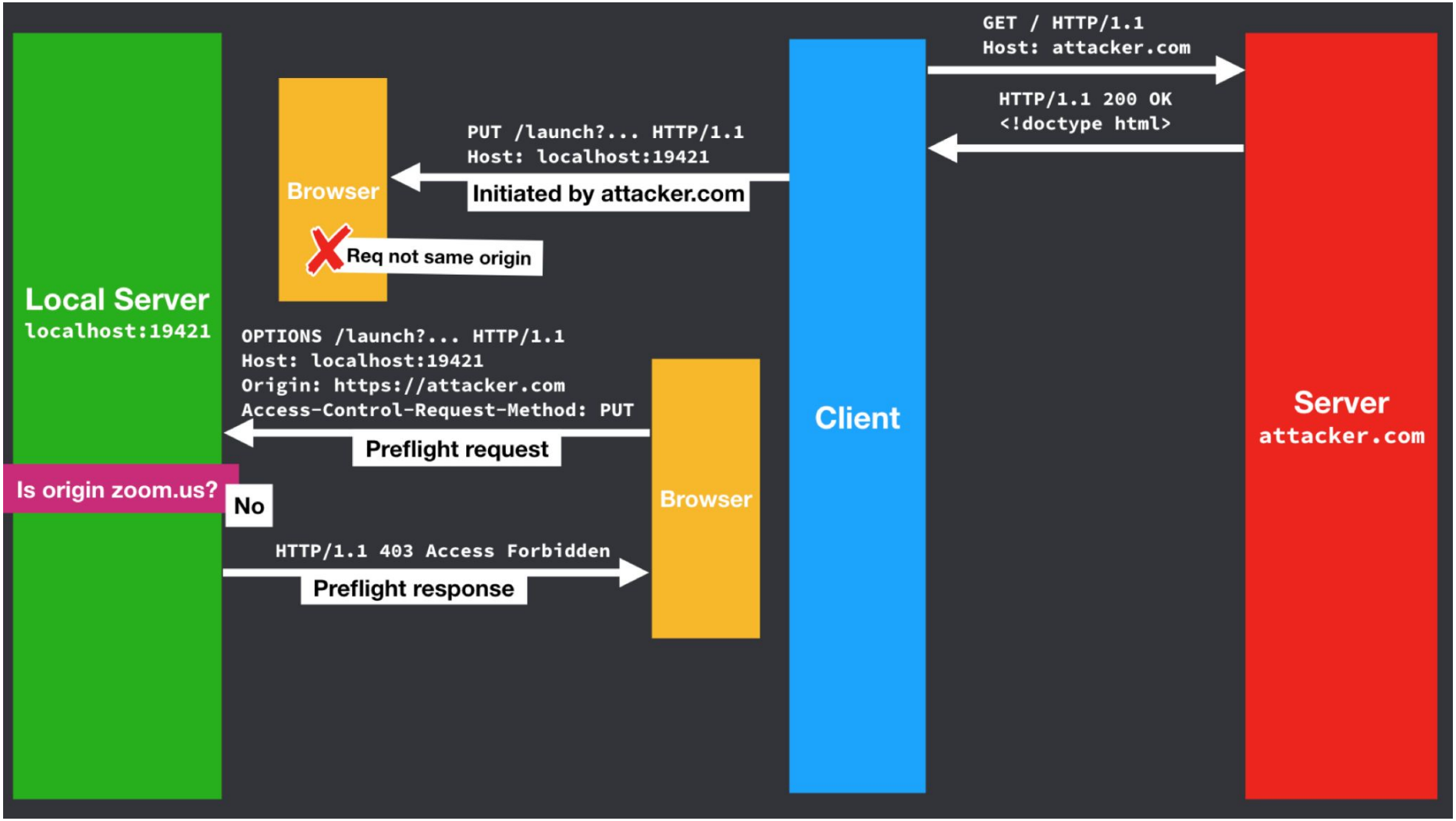

GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by attacker.com
Browser
Req not same origin
Local Server
localhost:19421
Client
Server
attacker.com

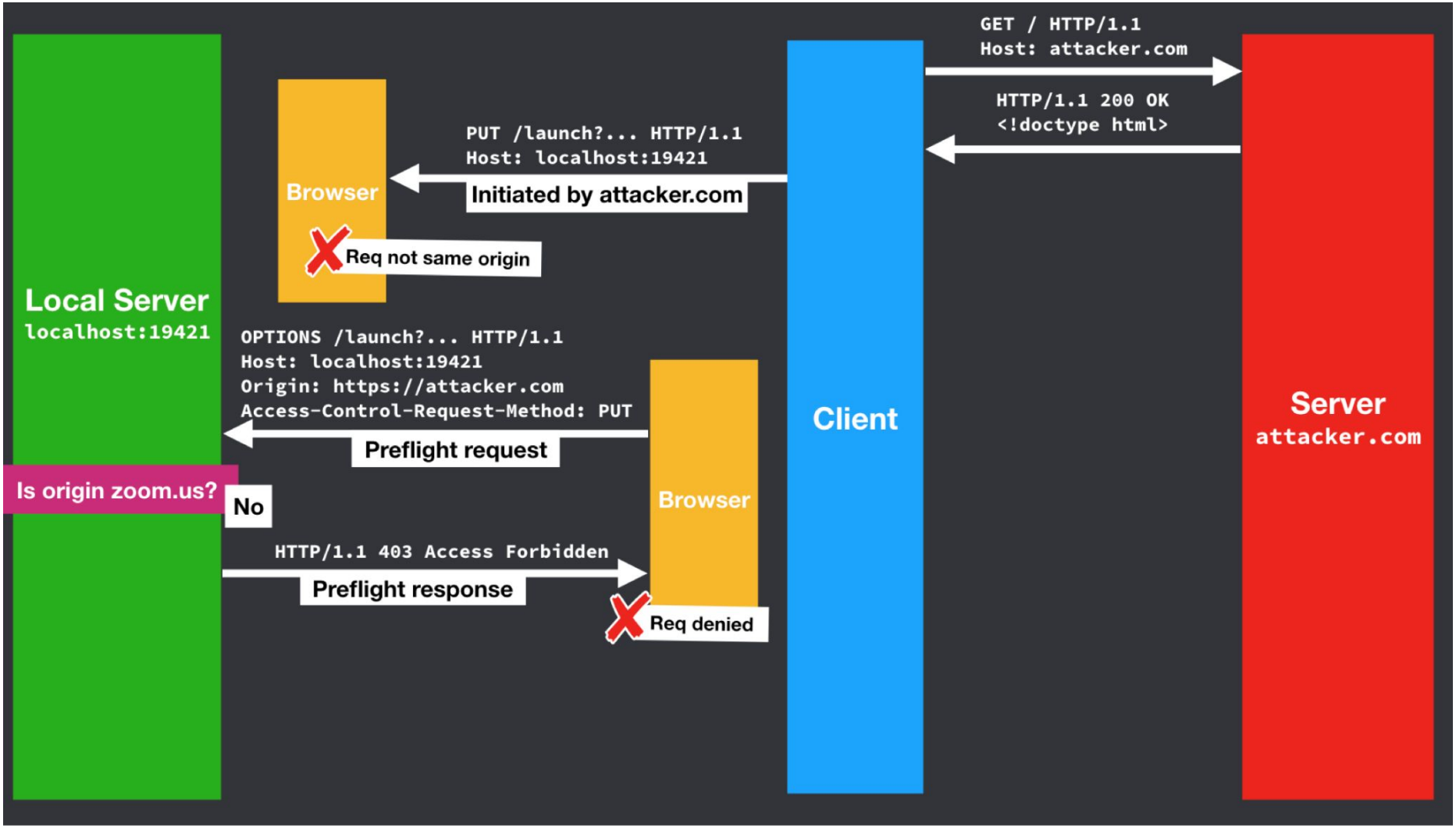

GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by attacker.com
Browser
Req not same origin
Local Server
localhost:19421
Browser
Client
Server
attacker.com


GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by attacker.com
Browser
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://attacker.com
Access-Control-Request-Method: PUT
Preflight request
Browser
Client
Server
attacker.com


GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by attacker.com
Browser
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://attacker.com
Access-Control-Request-Method: PUT
Preflight request
Is origin zoom.us?
Browser
Client
Server
attacker.com


GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
Server
attacker.com
Client
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by attacker.com
Browser
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://attacker.com
Access-Control-Request-Method: PUT
Preflight request
Browser
Is origin zoom.us?
No


GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
Server
attacker.com
Client
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by attacker.com
Browser
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://attacker.com
Access-Control-Request-Method: PUT
Preflight request
Is origin zoom.us?
No
Browser
HTTP/1.1 403 Access Forbidden
Preflight response


GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
Server
attacker.com
Client
PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by attacker.com
Browser
Req not same origin
Local Server
localhost:19421
OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://attacker.com
Access-Control-Request-Method: PUT
Preflight request
Is origin zoom.us?
No
Browser
HTTP/1.1 403 Access Forbidden
Preflight response
Req denied

Who can still launch the app from the local server?

- Preflight requests seems to allow the local server to distinguish requests from zoom.us and those from random sites
- However, other native apps running on the same device can still fool the local server
  - The browser enforces that sites can't tamper with the `Origin` header, but a native app (e.g., a Node.js or Python script) can make a request and set the `Origin` header to https://zoom.us
  - Note: if a script is already running on your local machine, you have a whole lot of other (potential) problems

One more thing...

- Every site on the web can send requests to our local HTTP server!
  - Works against the server that required "preflighted" requests as well as the server which just checked the `Origin` header
- “localhost” seems simple but is actually highly magical -- what if you could redefine what “localhost” meant?

DNS Rebinding Attack

- DNS rebinding allows a remote attacker to:
  - Bypass cross-origin resource sharing (CORS) rules
  - Bypass a victim's network firewall and use their web browser as a proxy to communicate directly with vulnerable servers on the local network
- DNS rebinding exploits limitations in the same origin policy:
  - Origin is just defined as protocol + hostname + port
  - The actual IP address that the hostname resolves to is not included

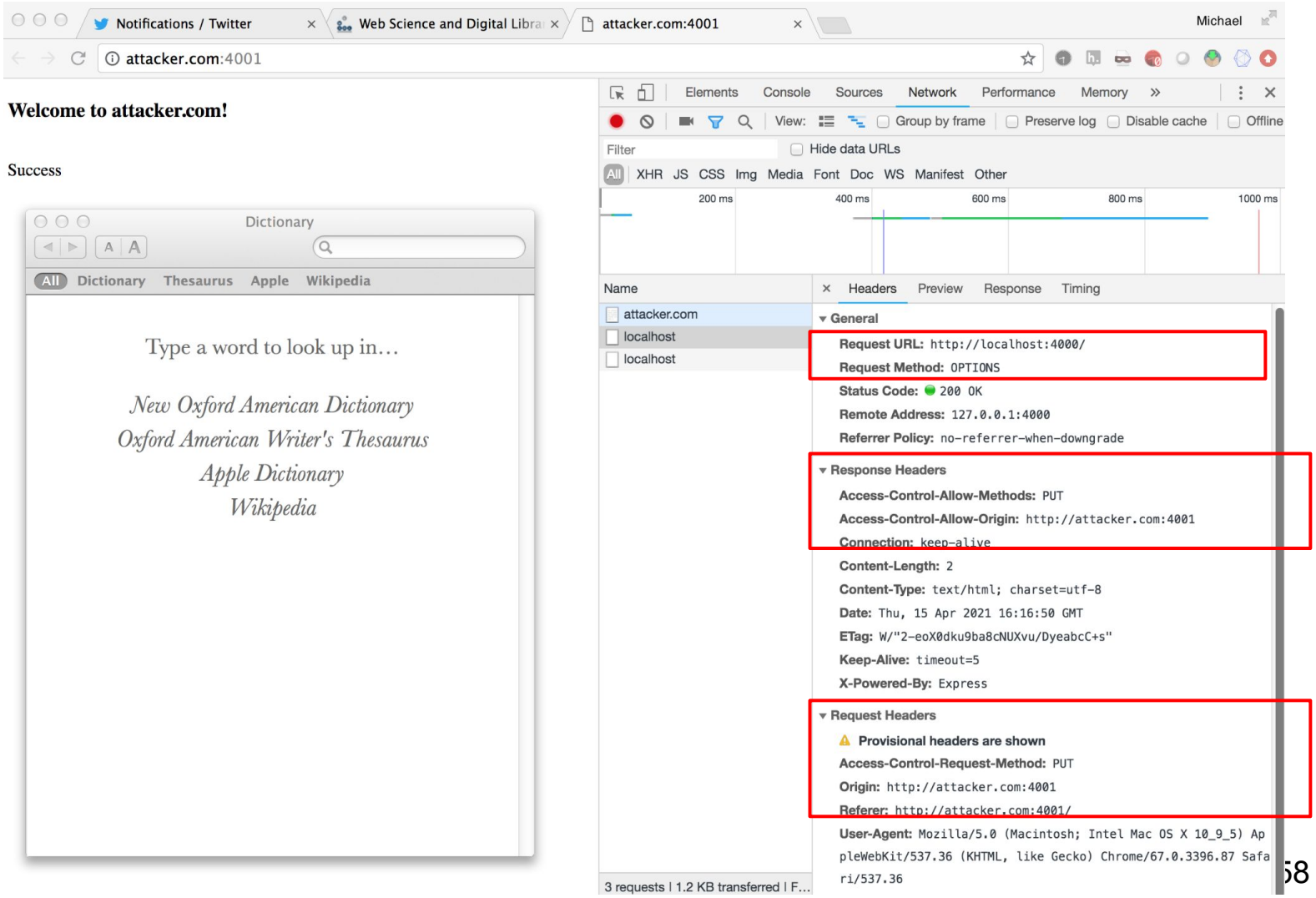
demo

- dictionary.js
  - dns-attack-1.js
  - dns-attack-2.js
- dictionary-2.js
  - dns-attack-2.js
- dictionary-3.js
  - dns-attack-3.js
- dictionary-4.js
  - dns-attack-3.js

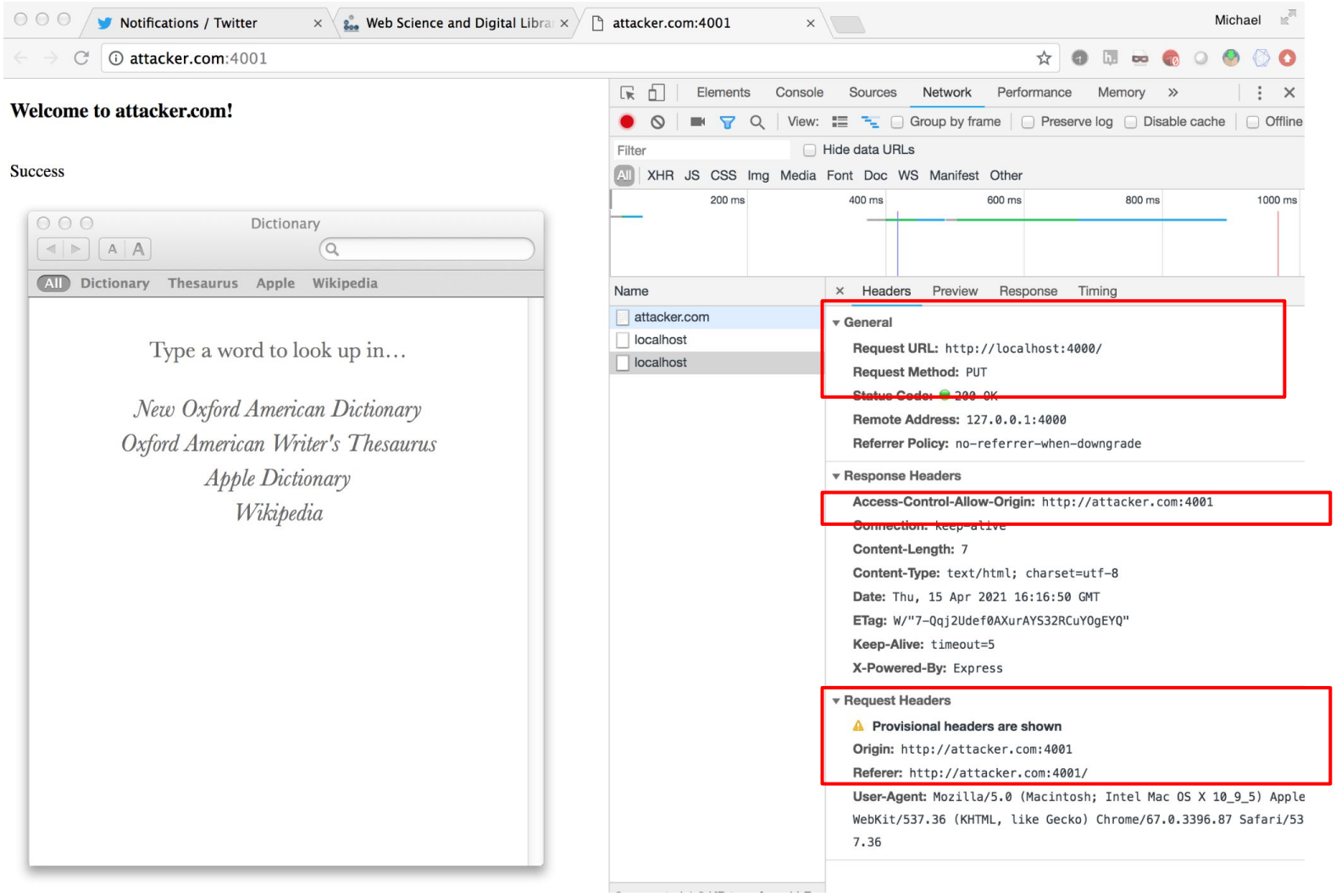
adding attacker.com to /etc/hosts

```
$ ifconfig en0
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
     ether b8:e8:56:40:6d:78
     inet6 fe80::bae8:56ff:fe40:6d78%en0 prefixlen 64 scopeid 0x4
     inet 192.168.0.211 netmask 0xffffff00 broadcast 192.168.0.255
     inet6 2600:8805:1510:a600:bae8:56ff:fe40:6d78 prefixlen 64 autoconf
     inet6 2600:8805:1510:a600:10bd:2de2:4e94:4ca2 prefixlen 64 deprecated
autoconf temporary
     inet6 2600:8805:1510:a600::eb52 prefixlen 64
     inet6 2600:8805:1510:a600:282c:c0a5:8025:17ac prefixlen 64 autoconf
temporary
     nd6 options=1<PERFORMNUD>
     media: autoselect
     status: active
$ sudo vi /etc/hosts
$ more /etc/hosts
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
fe80::1%lo0     localhost
192.168.0.211    attacker.com
192.168.0.211    mln-phonedude.com
```

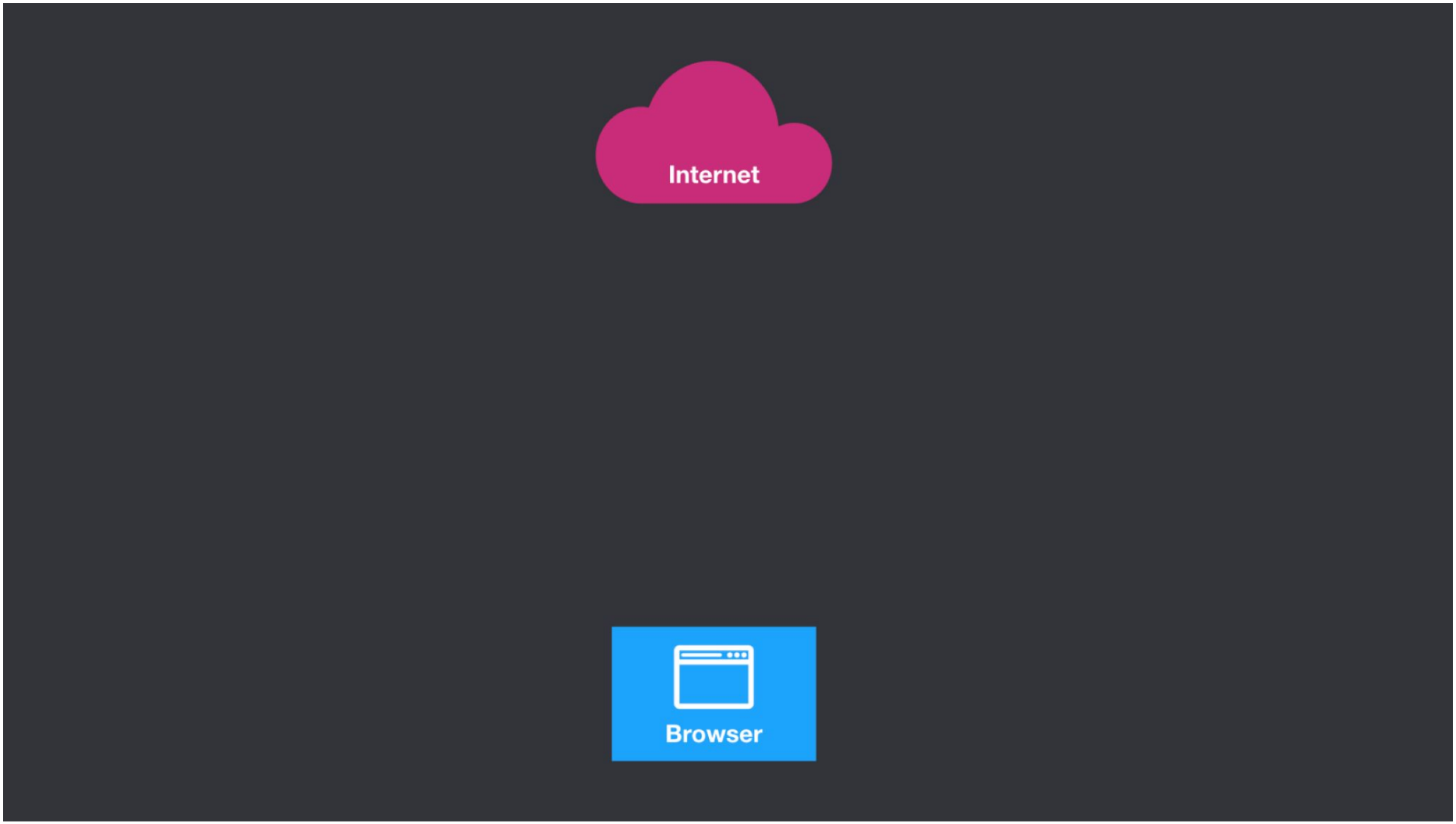
OPTIONS preflight

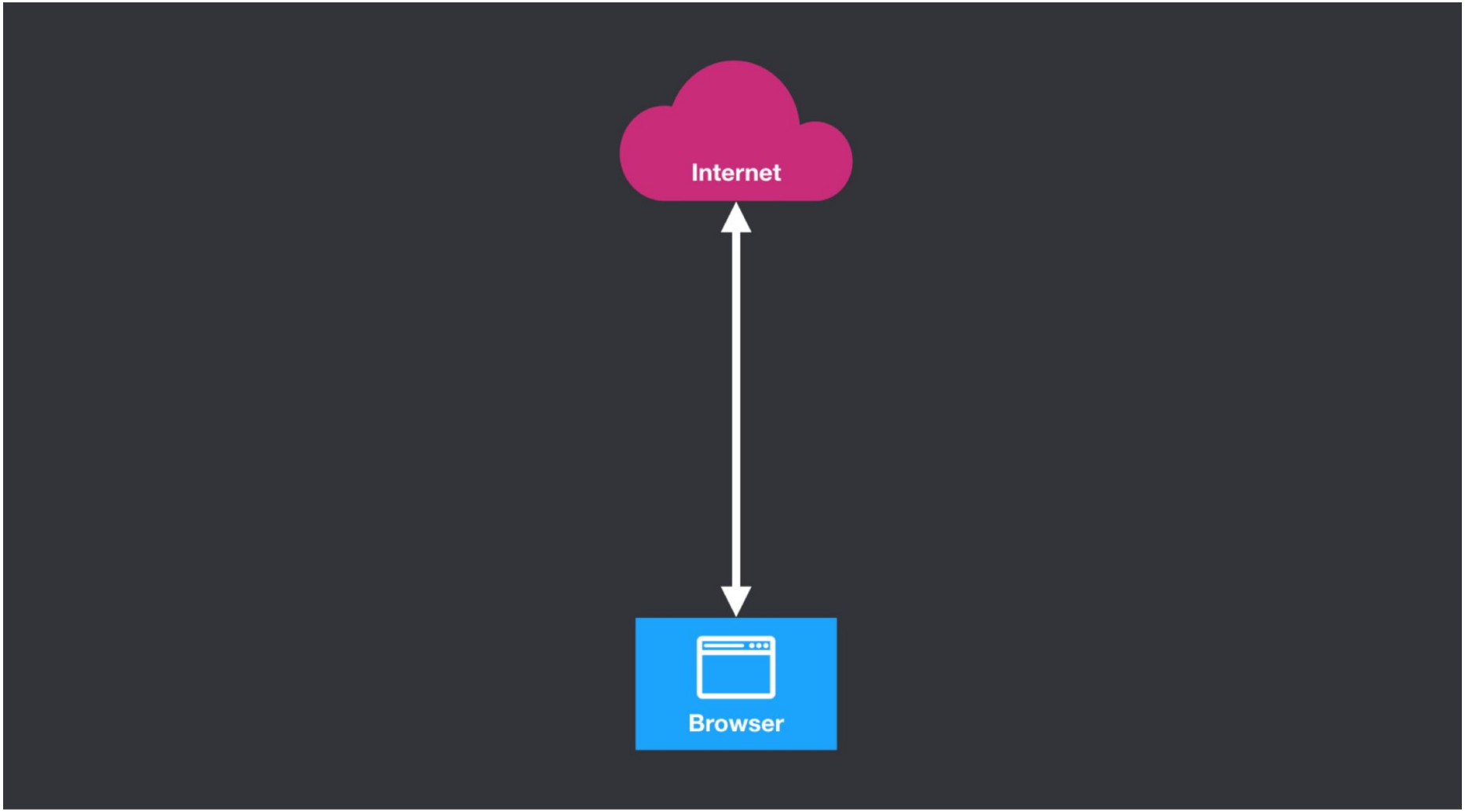


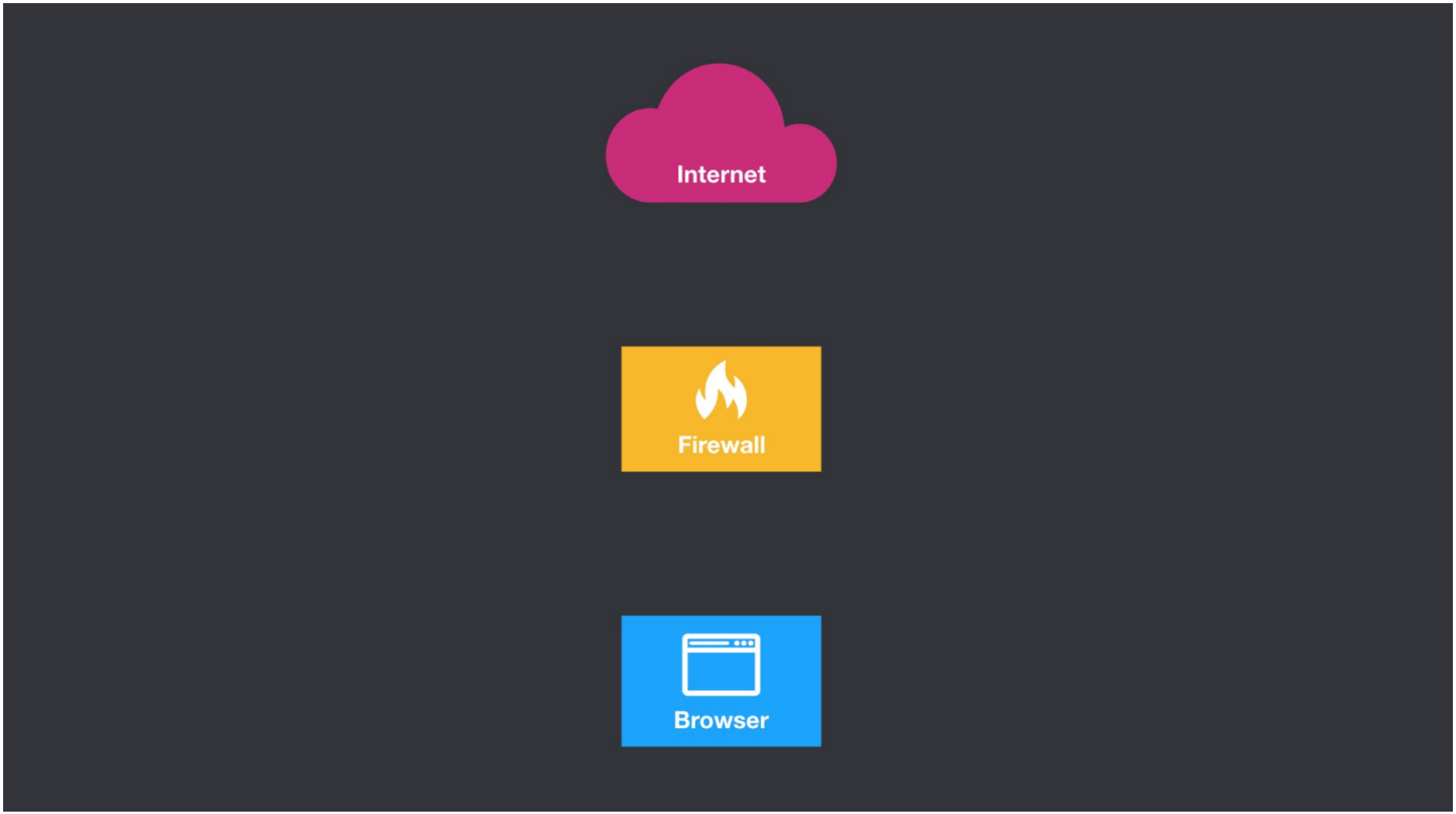
PUT after preflight

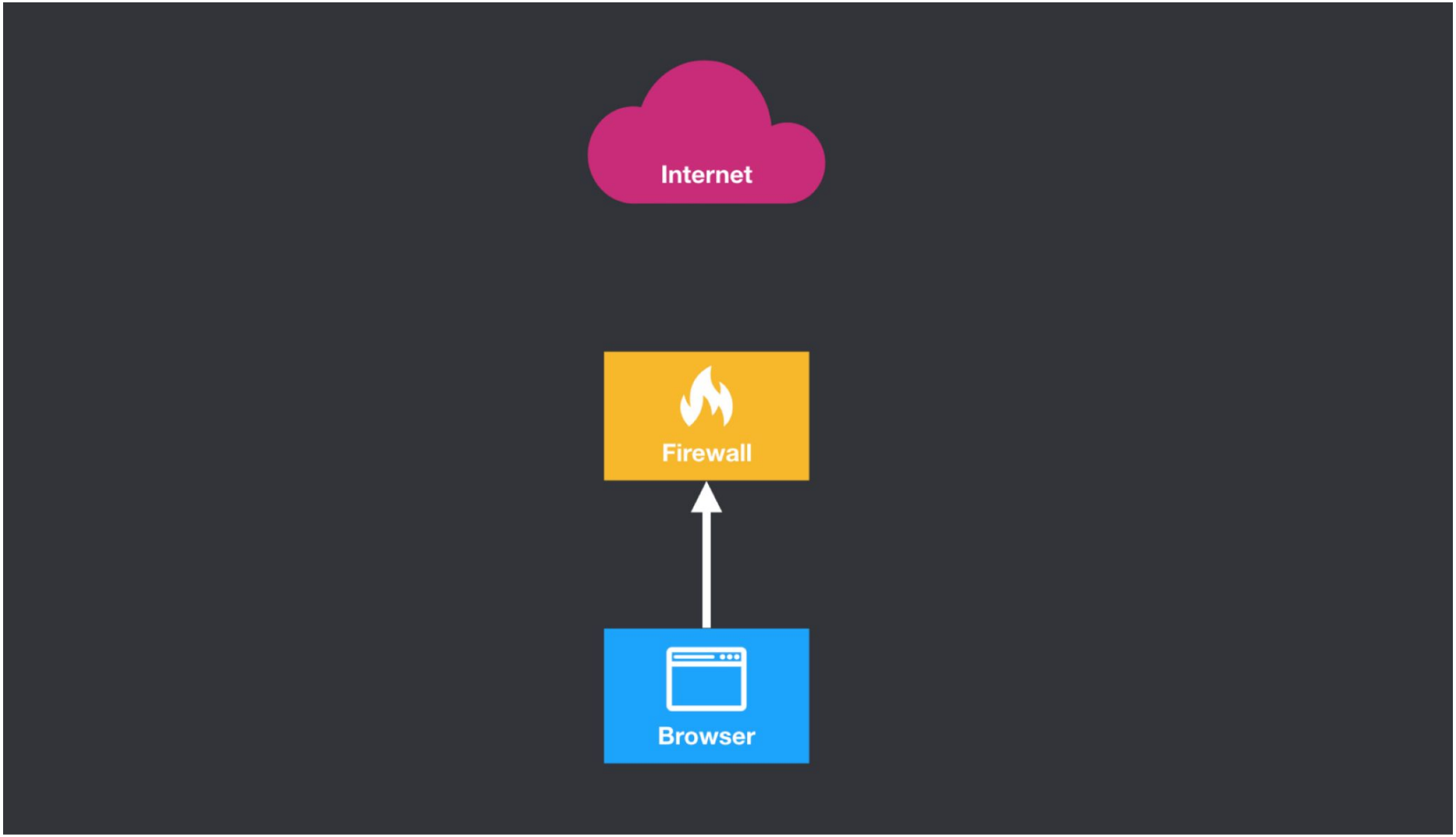


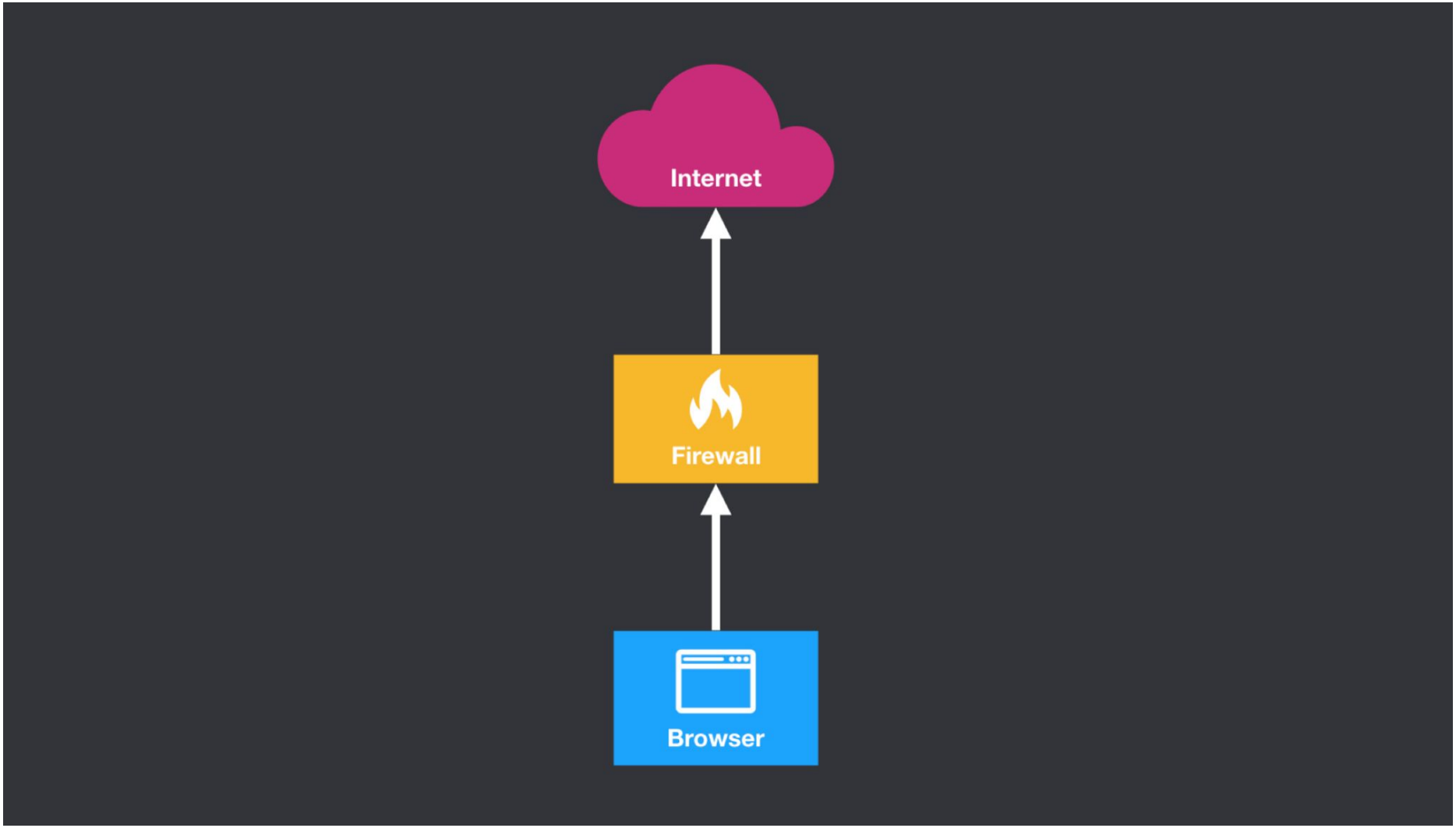
Enterprise Firewall

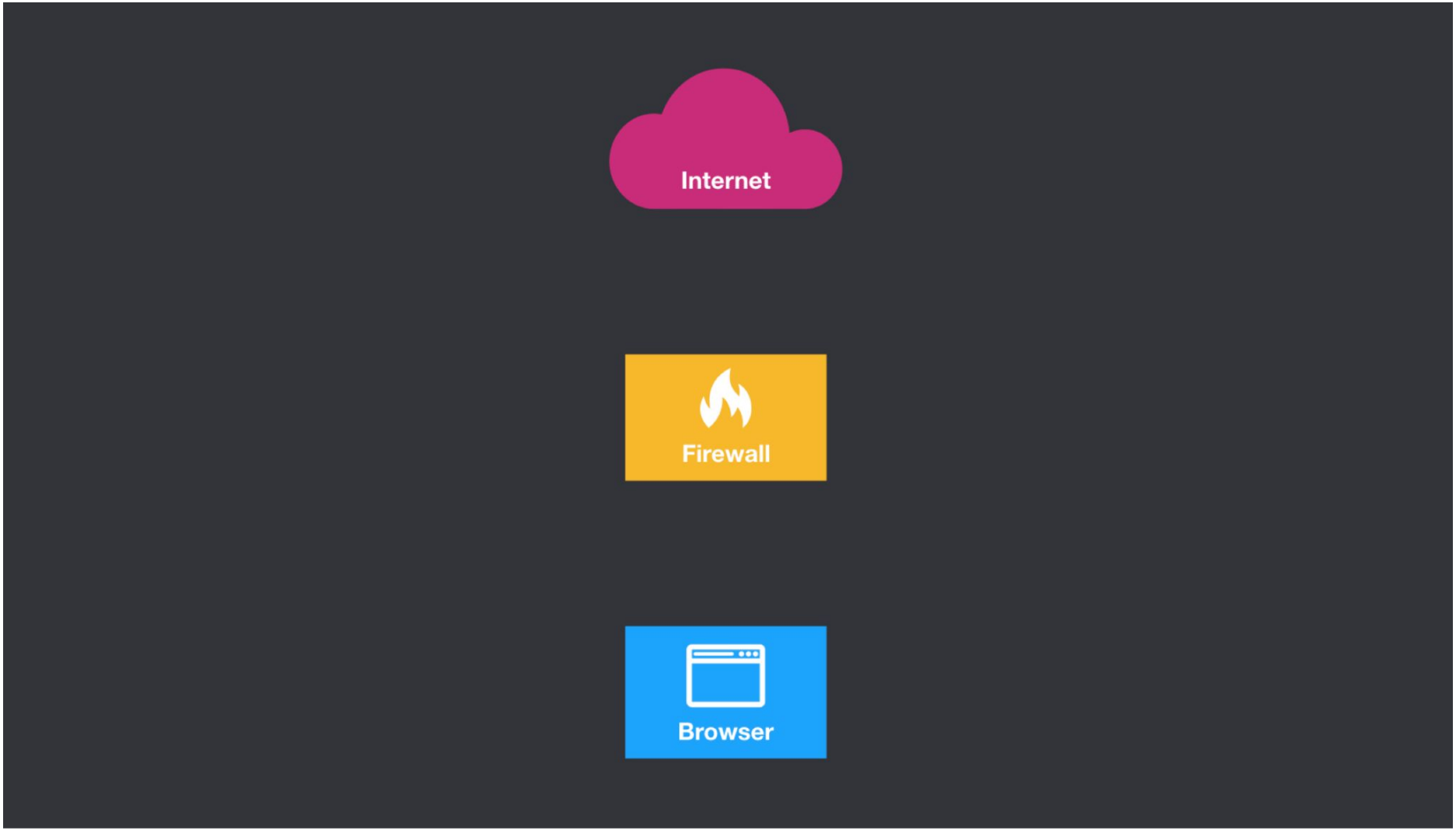

Internet
Browser

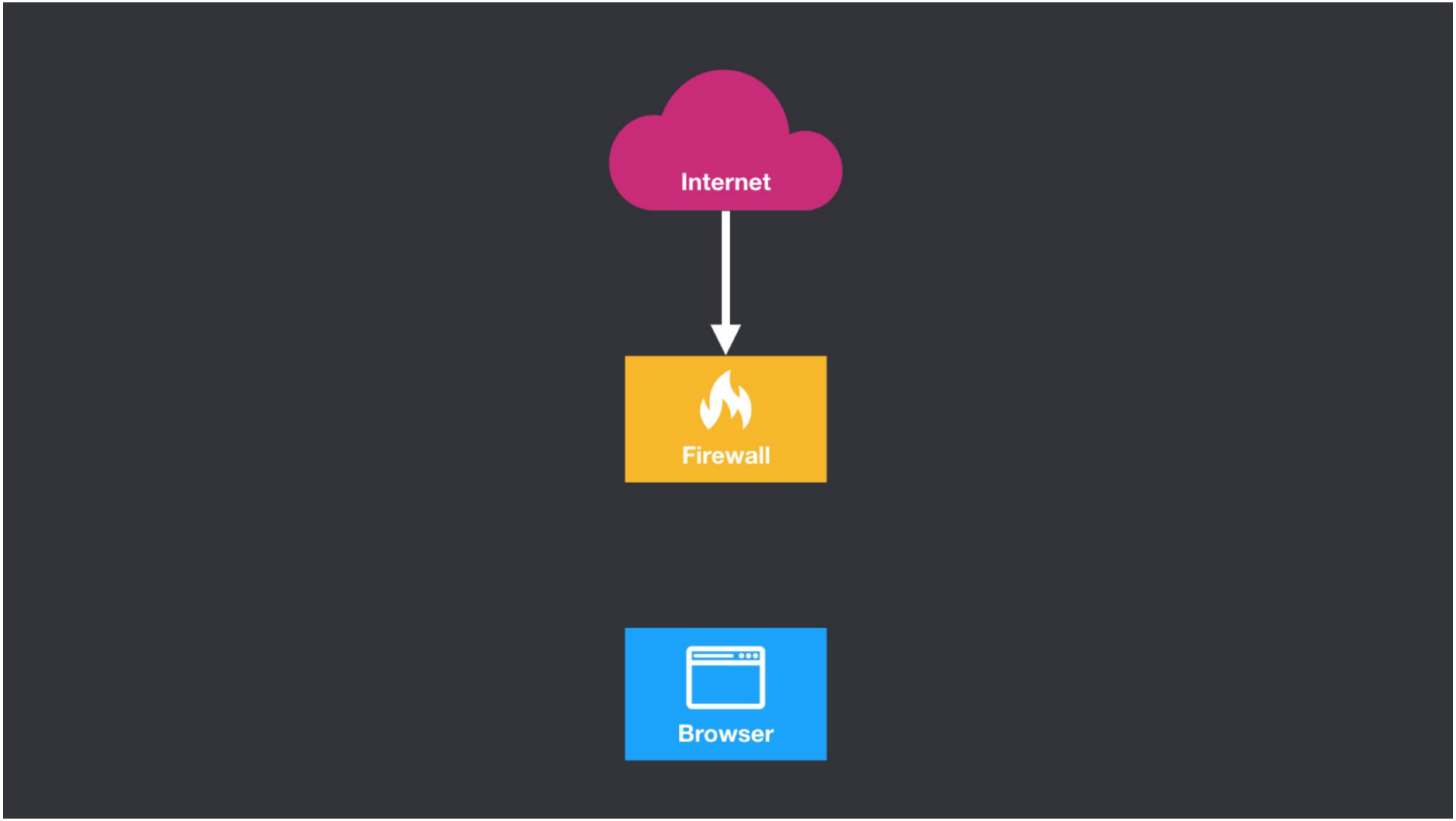

Internet
Browser

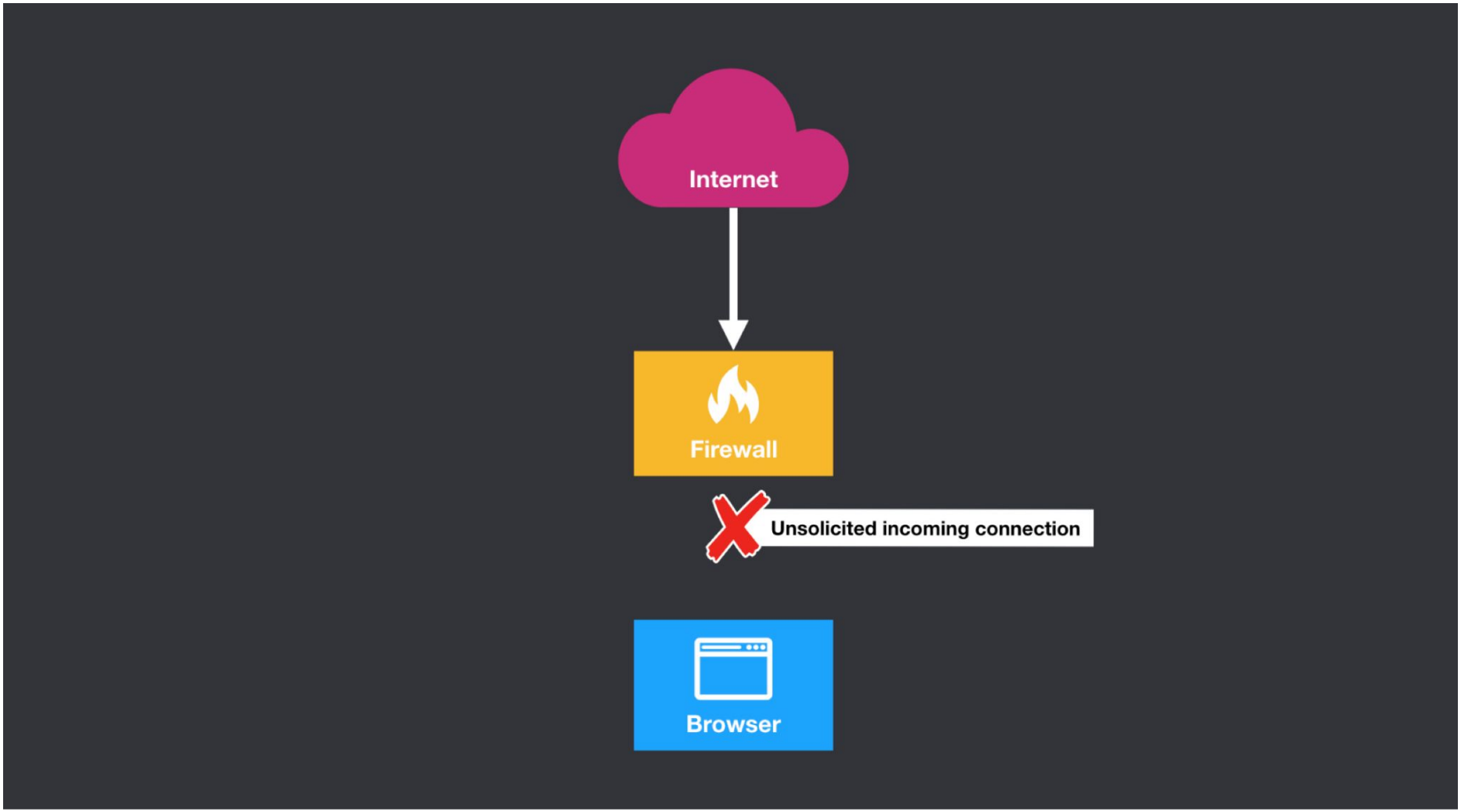

Internet
Firewall
Browser

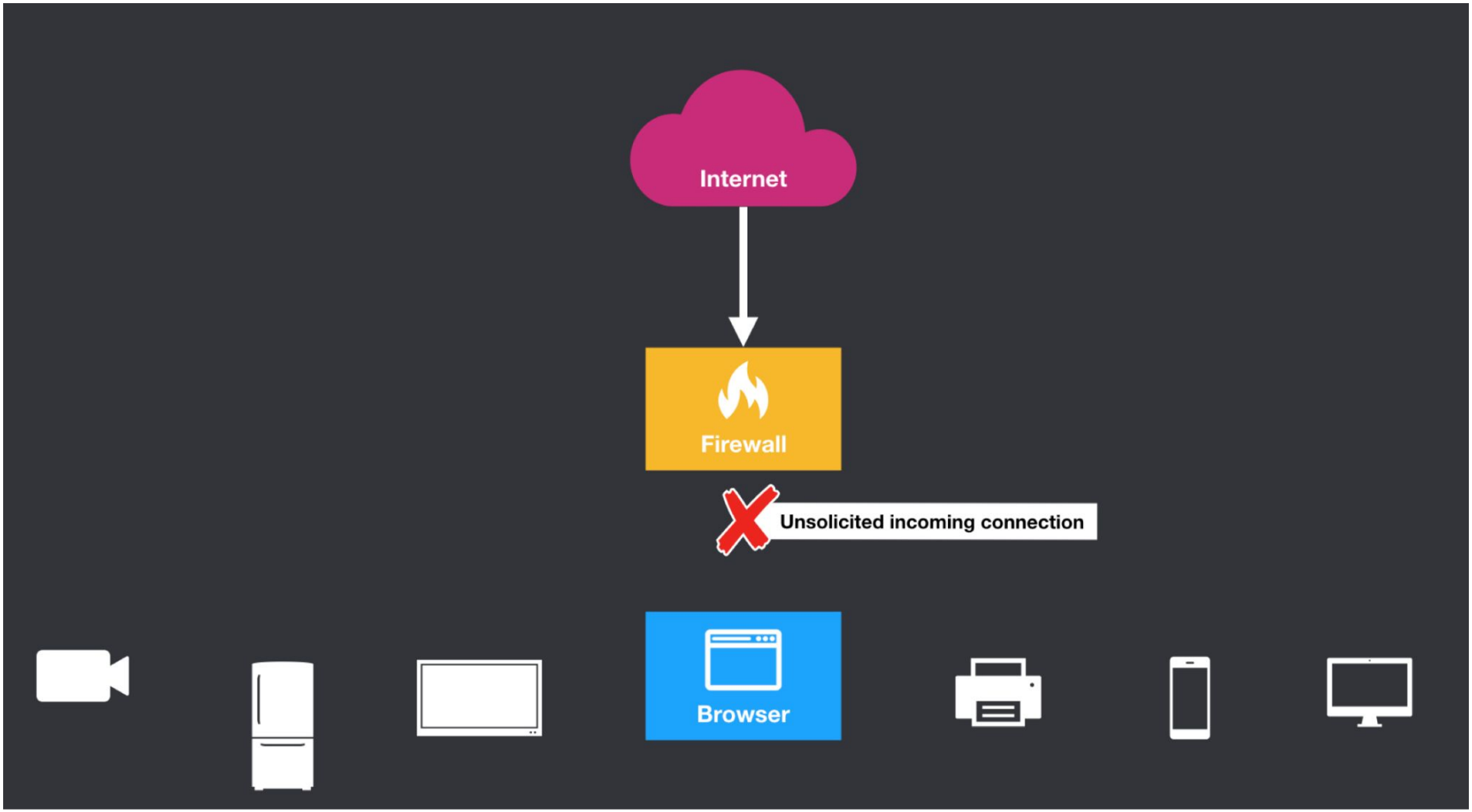

Internet
Firewall
Browser

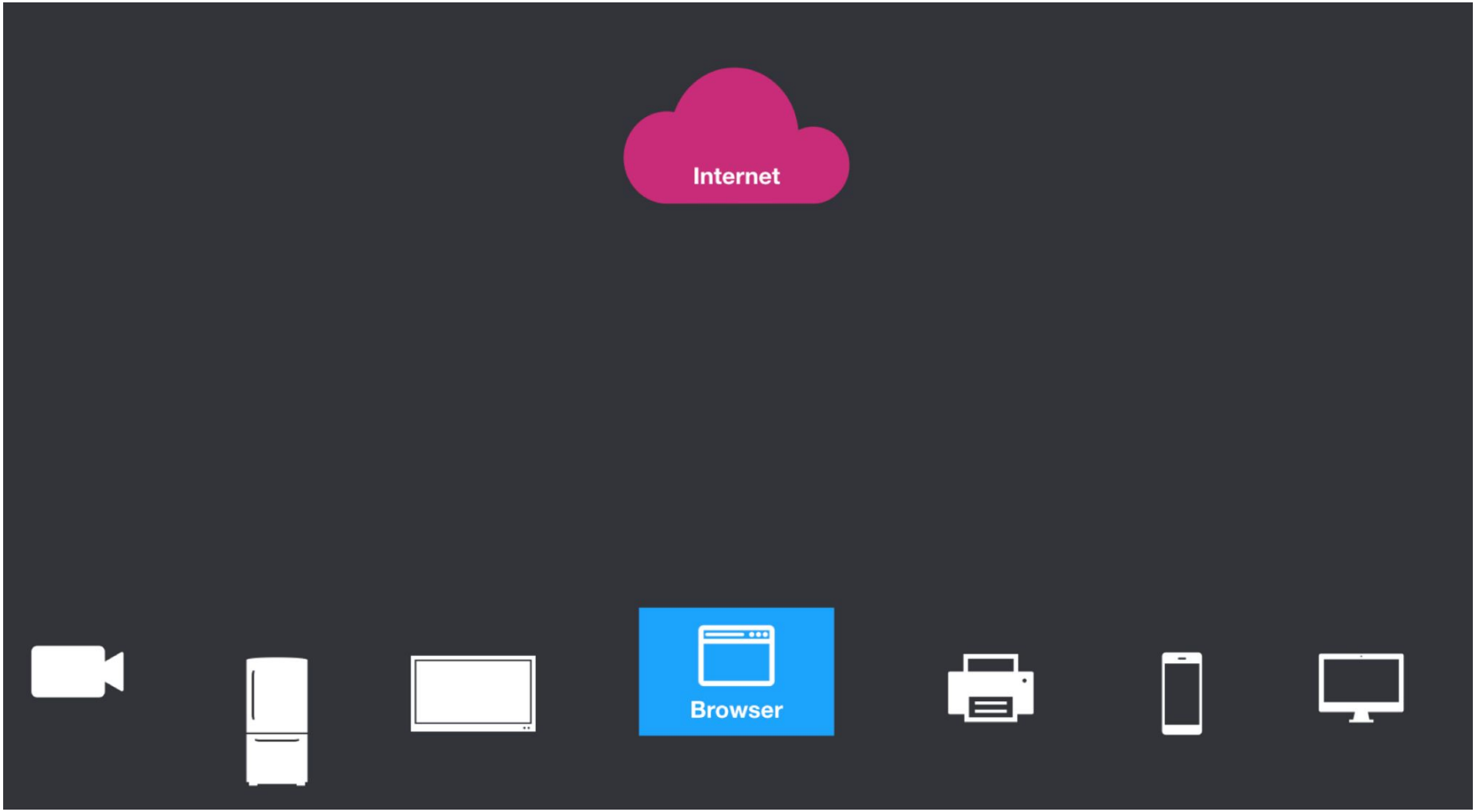

Internet
Firewall
Browser

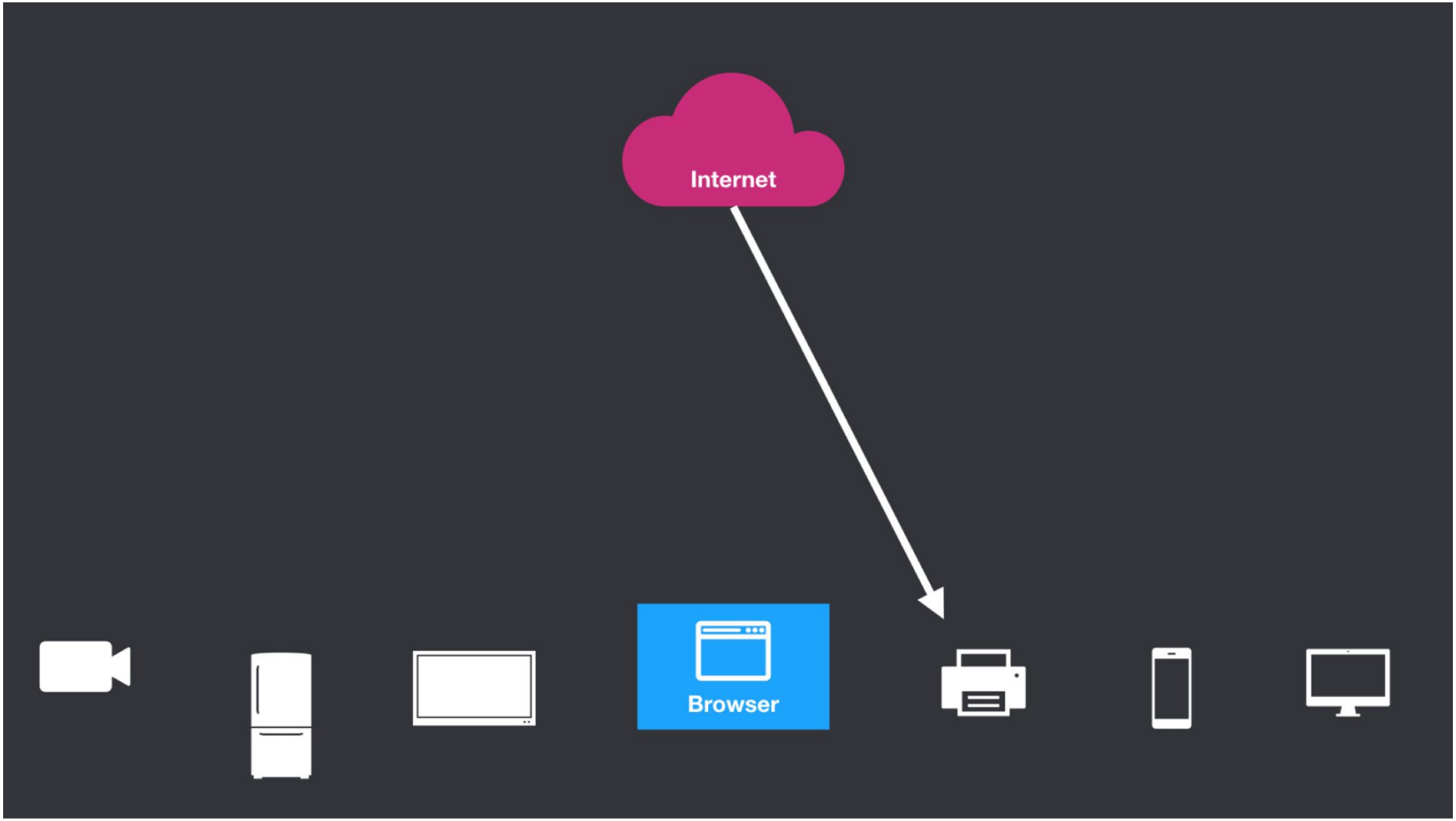

Internet
Firewall
Browser

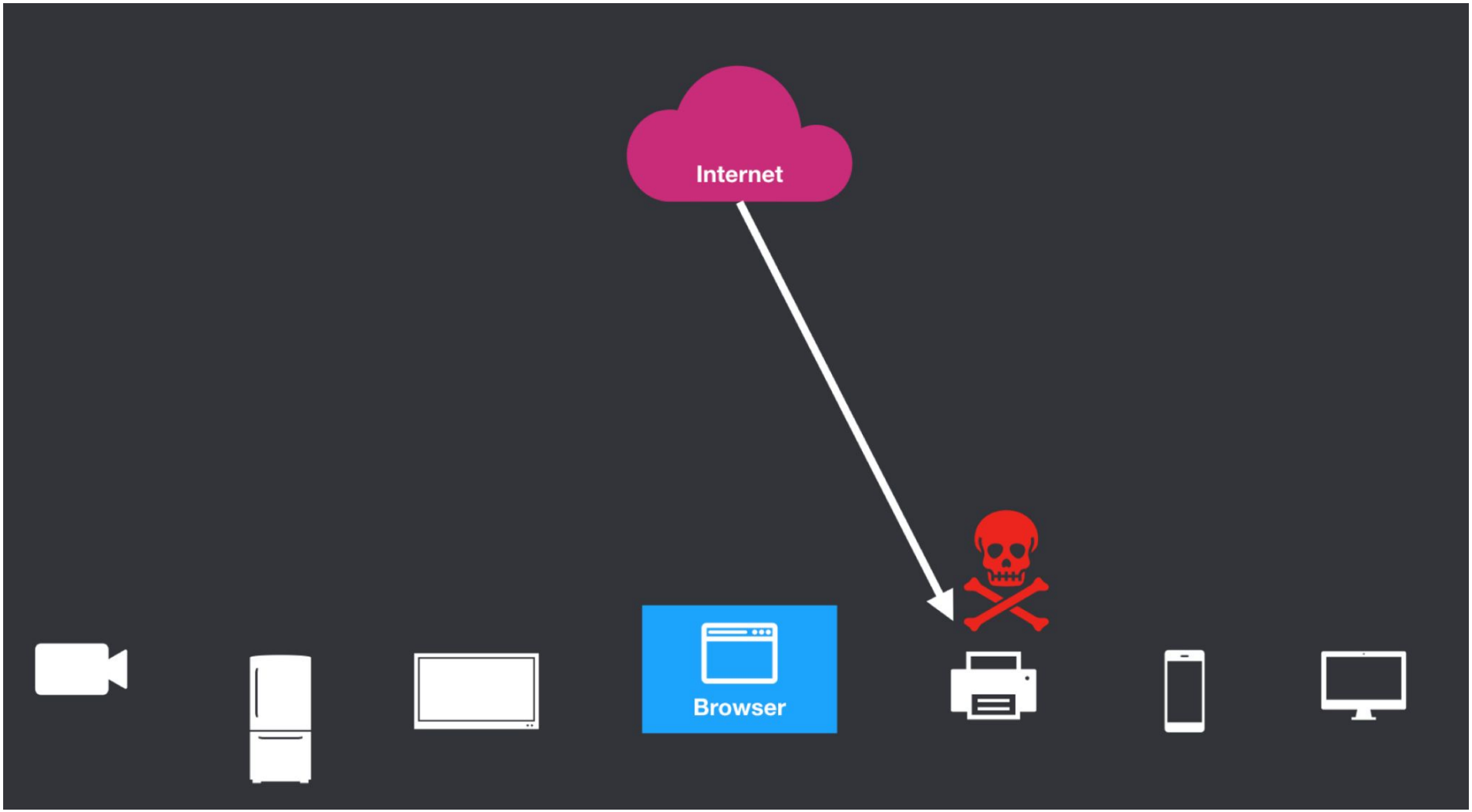

Internet
Firewall
Browser

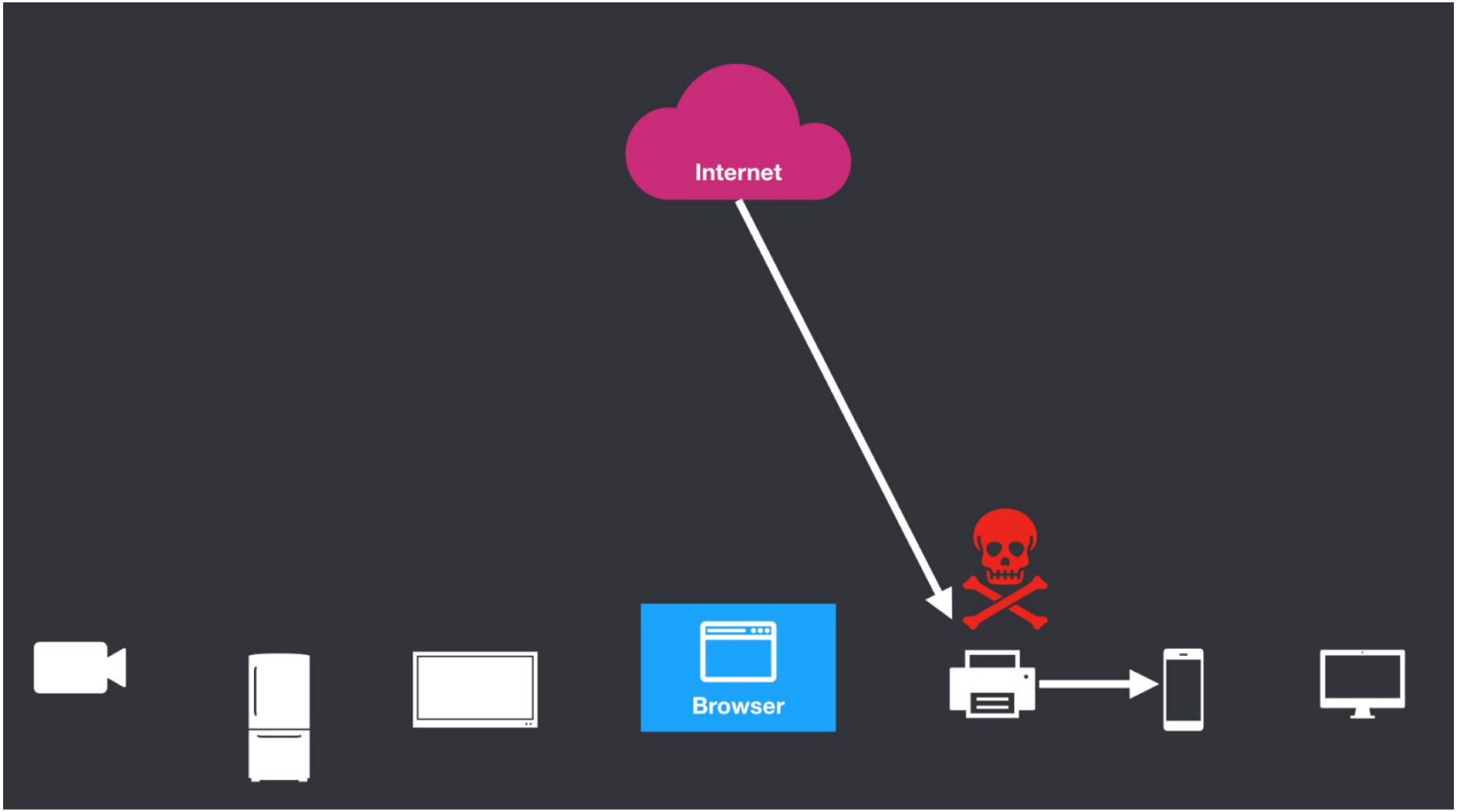

Internet
Firewall
Unsolicited incoming connection
Browser

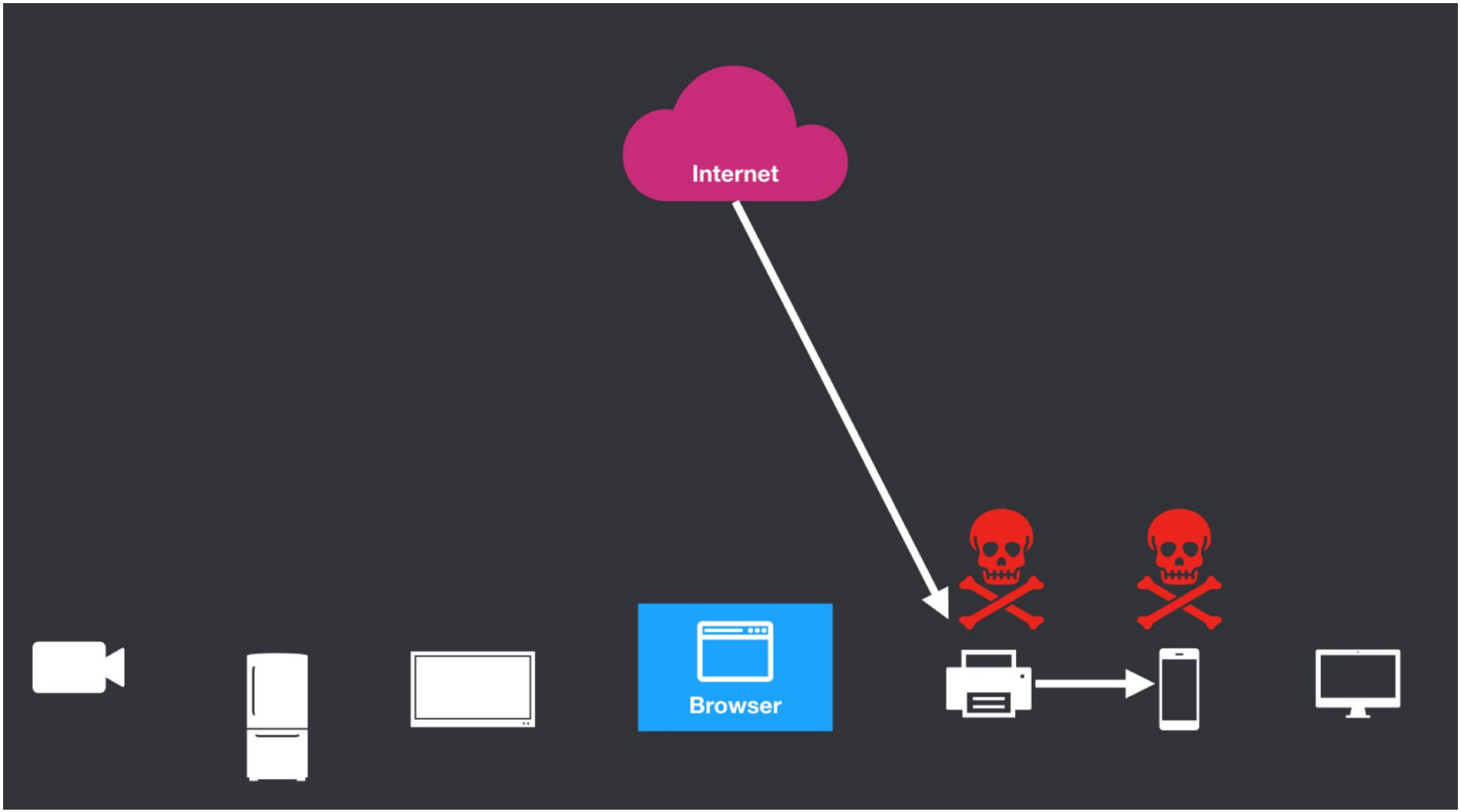

Internet
Firewall
Unsolicited incoming connection
Browser

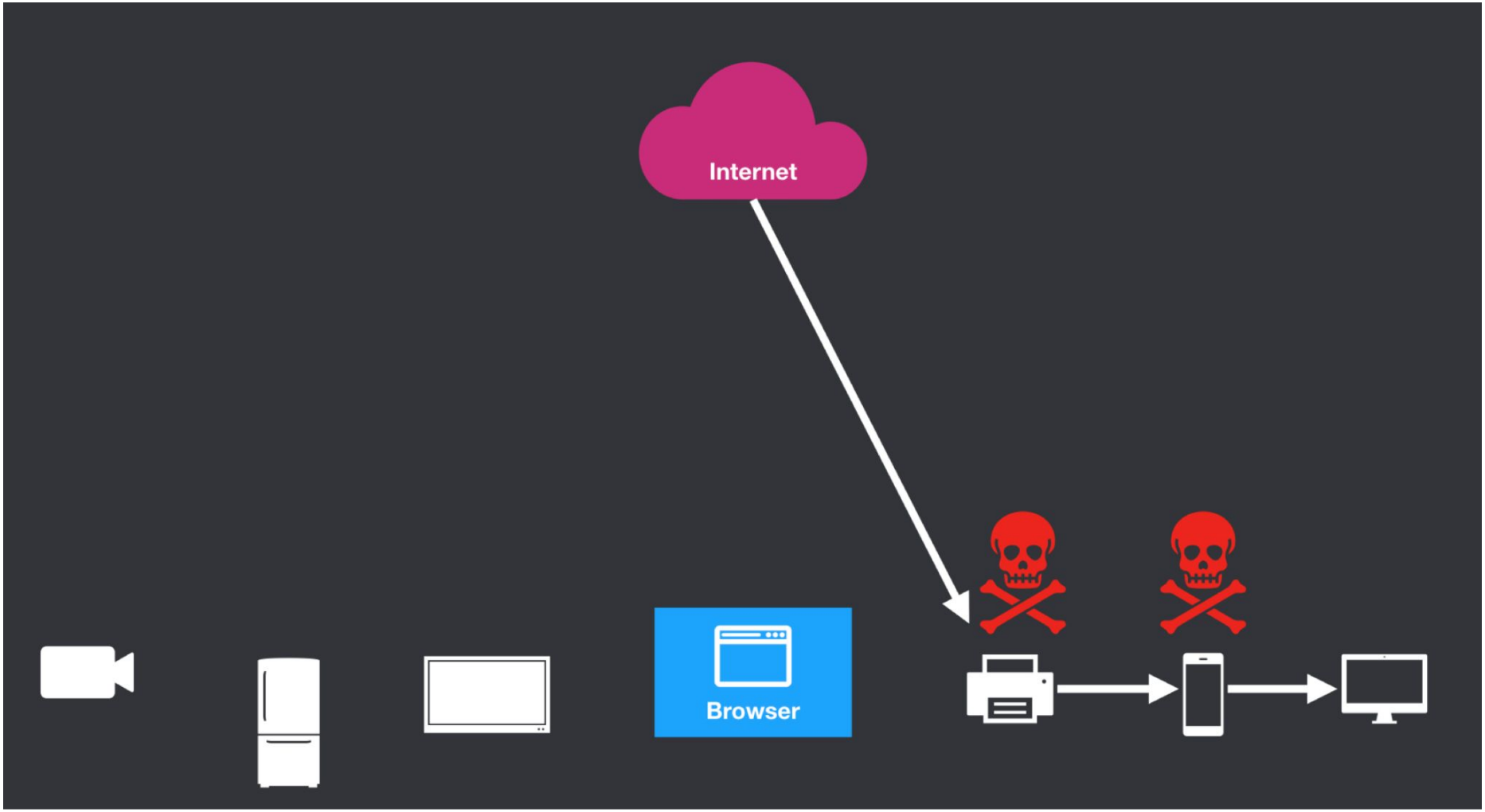

Internet
Browser

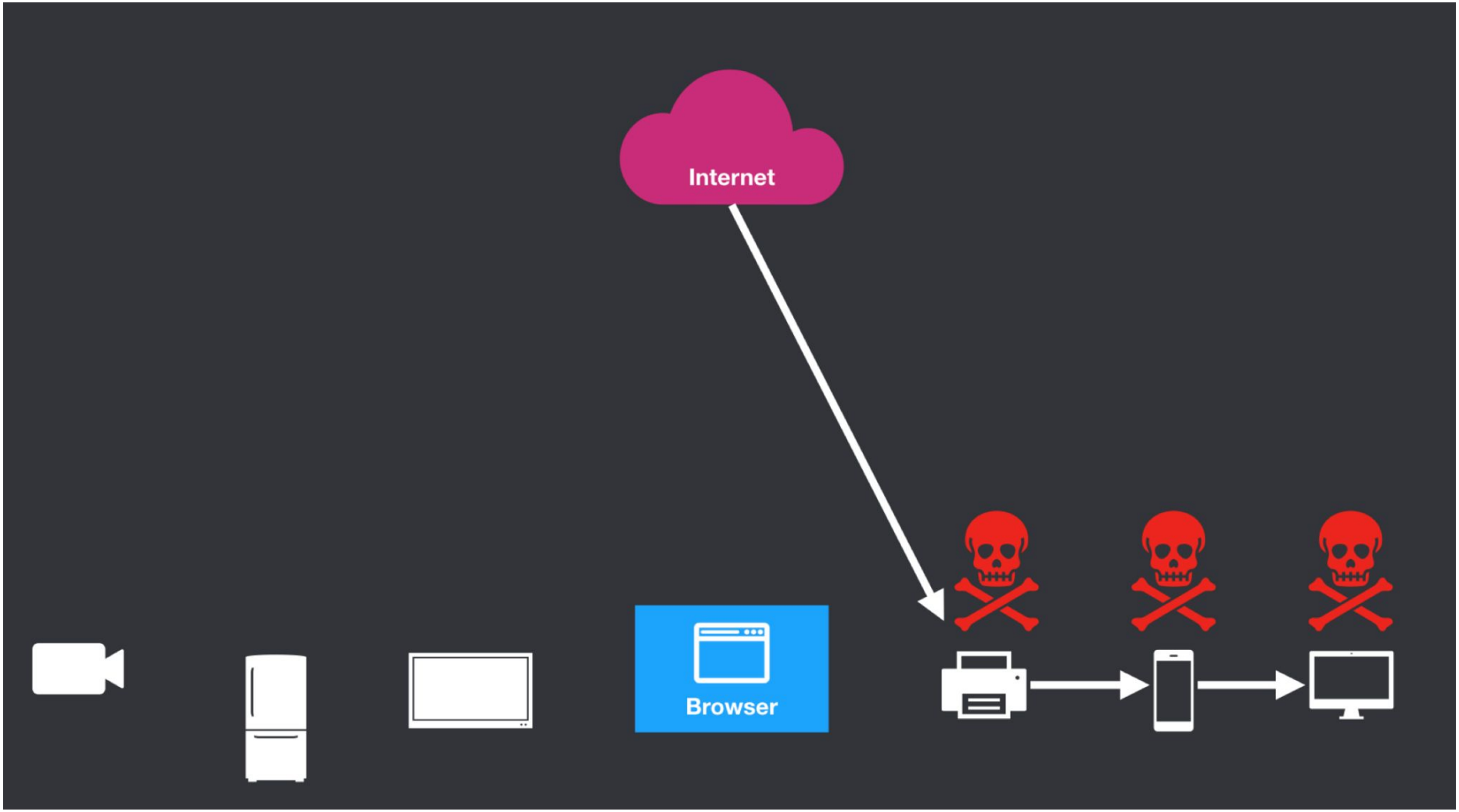

Internet
Browser

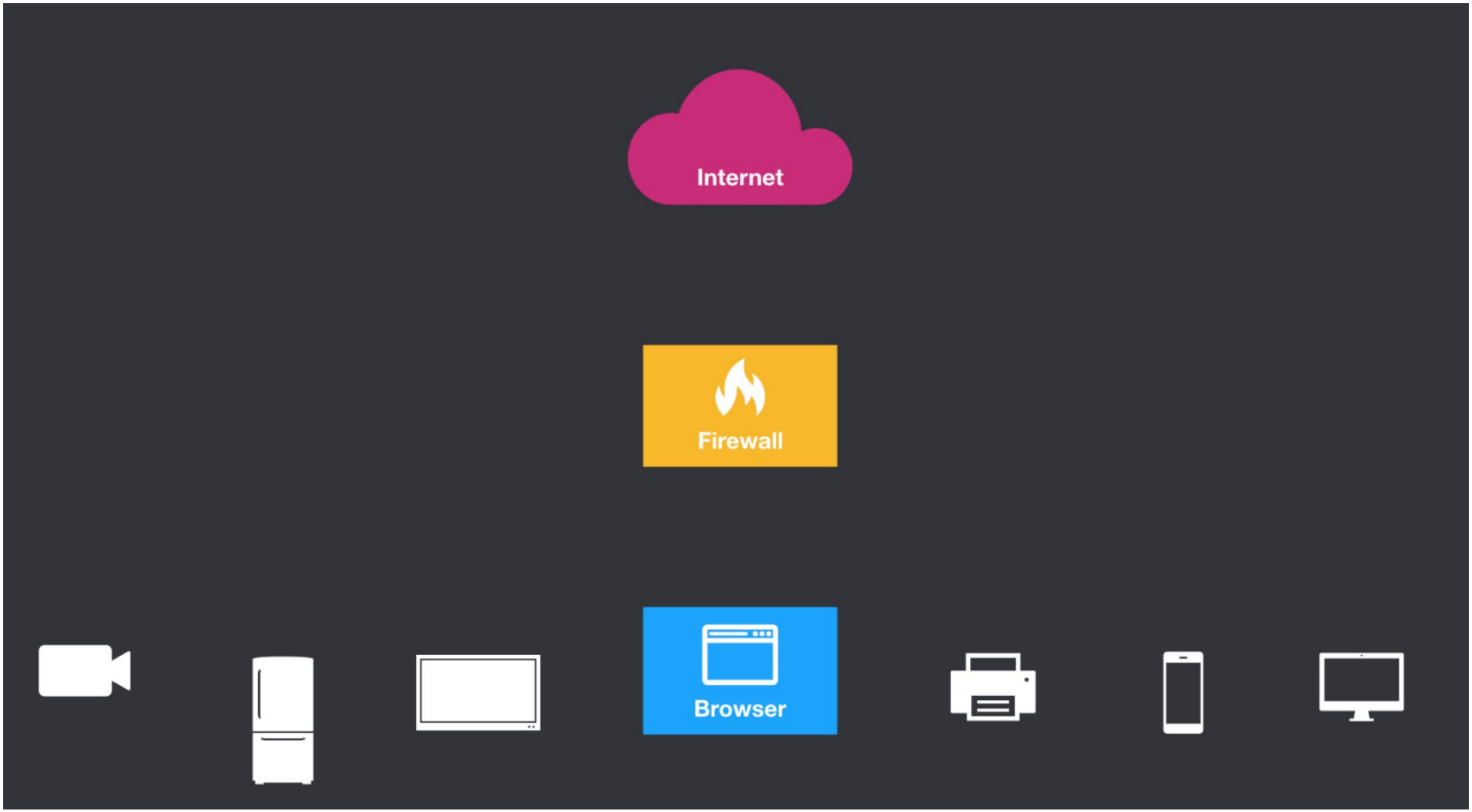

Internet
Browser

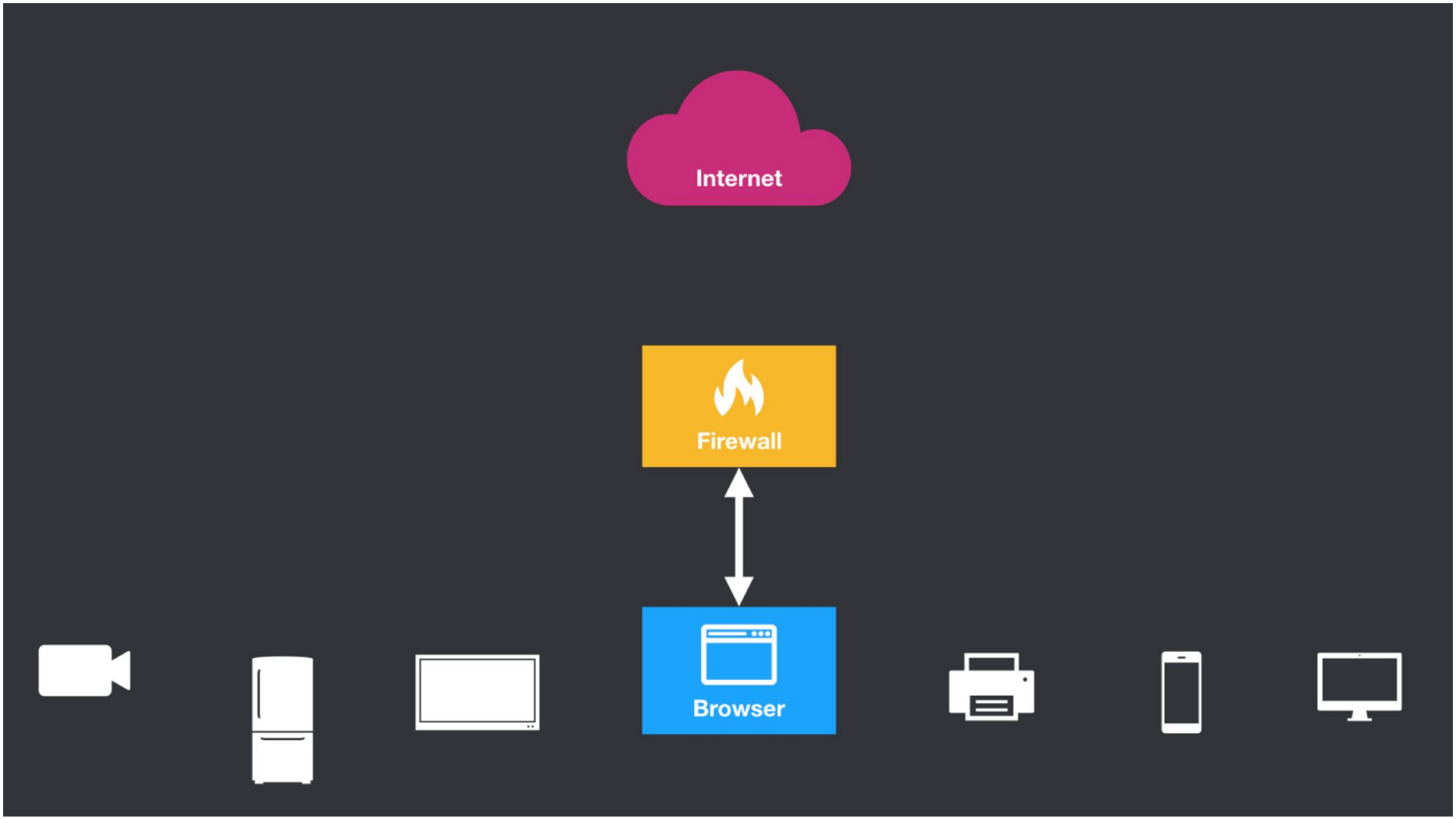

Internet
Browser

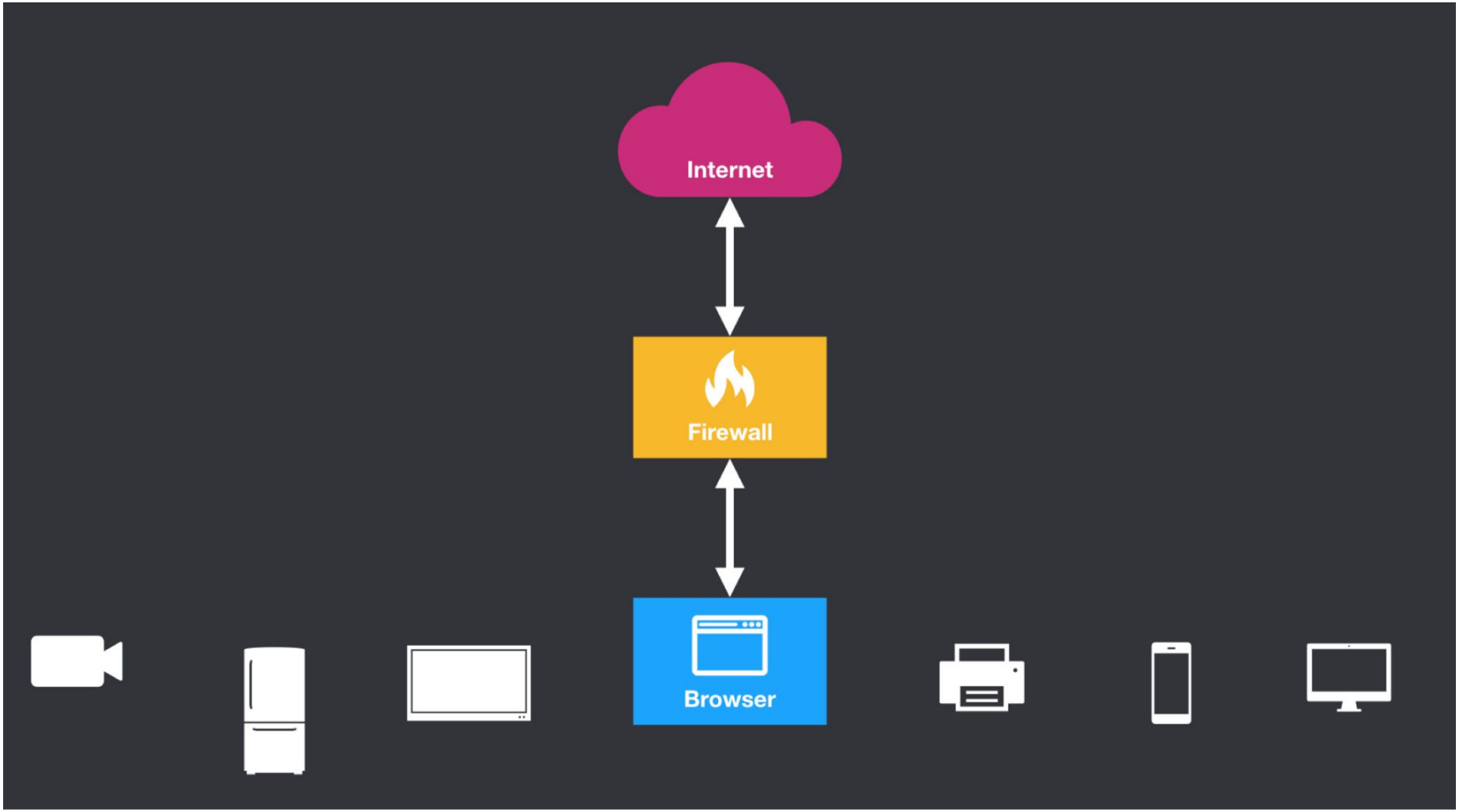

Internet
Browser

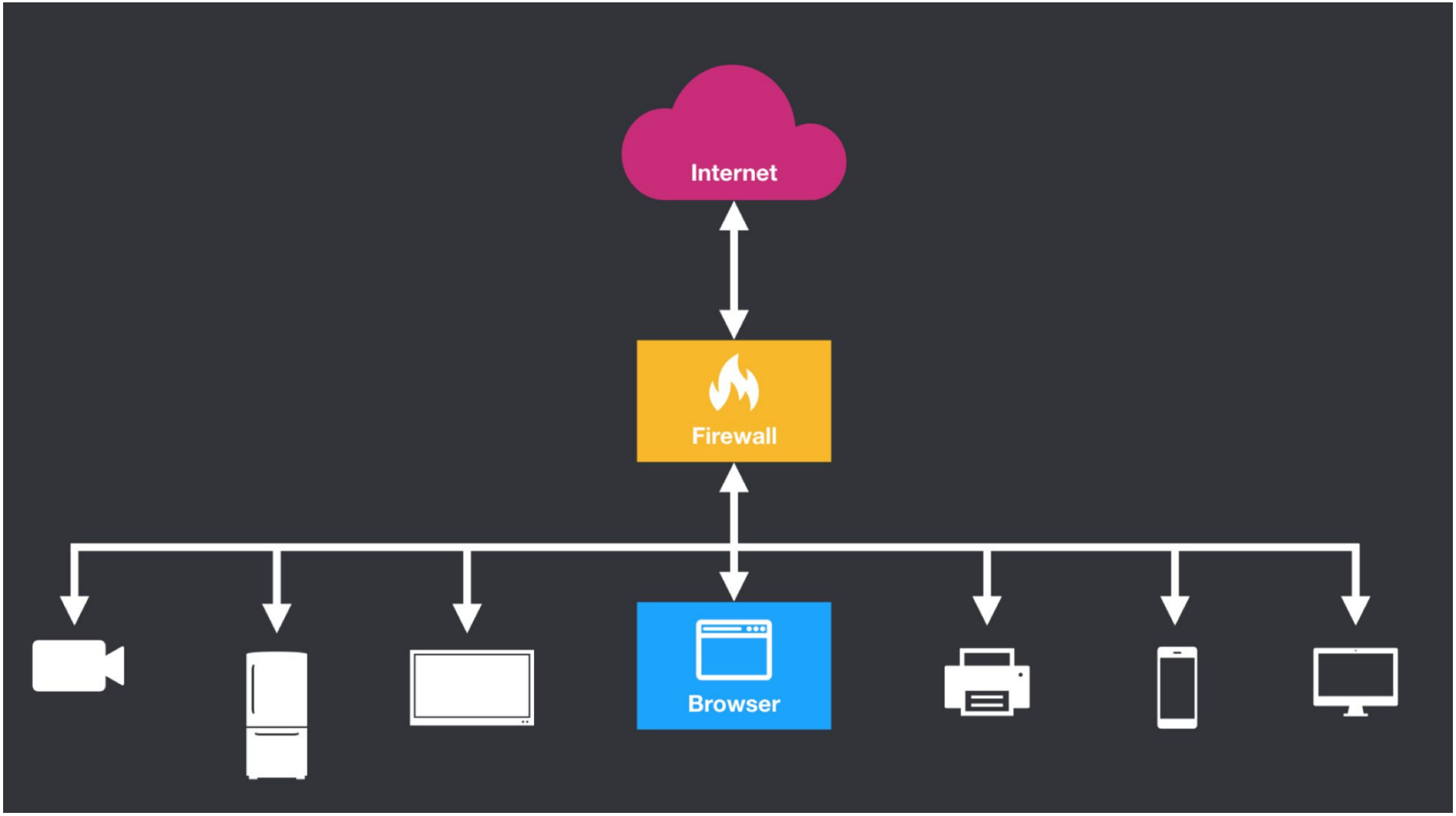

Internet
Browser


Internet
Browser

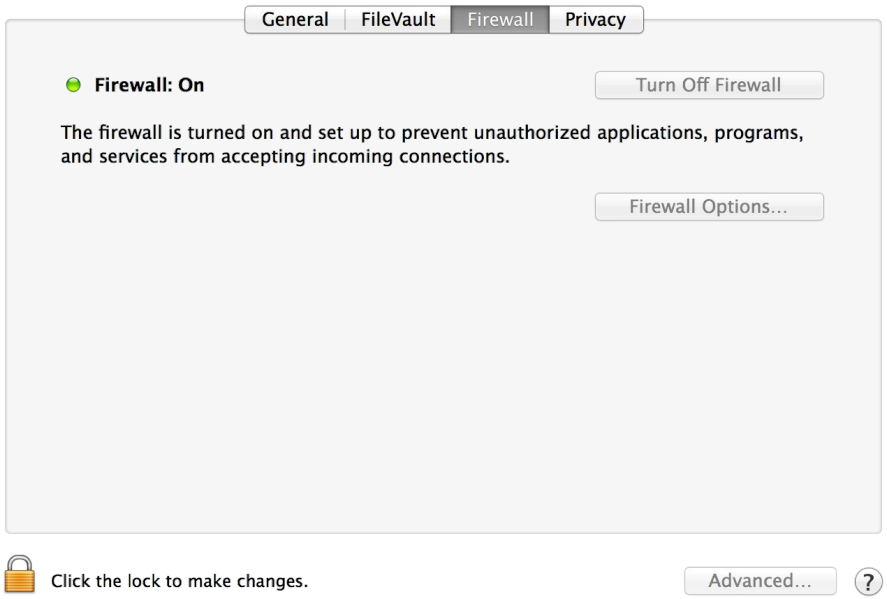

Internet
Firewall
Browser

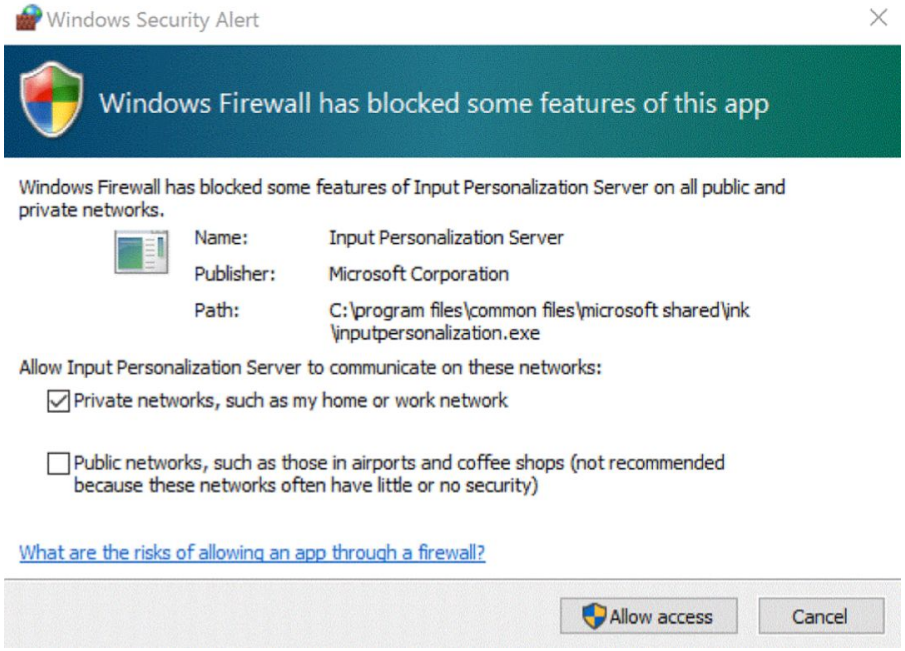

Internet
Firewall
Browser


Internet
Firewall
Browser

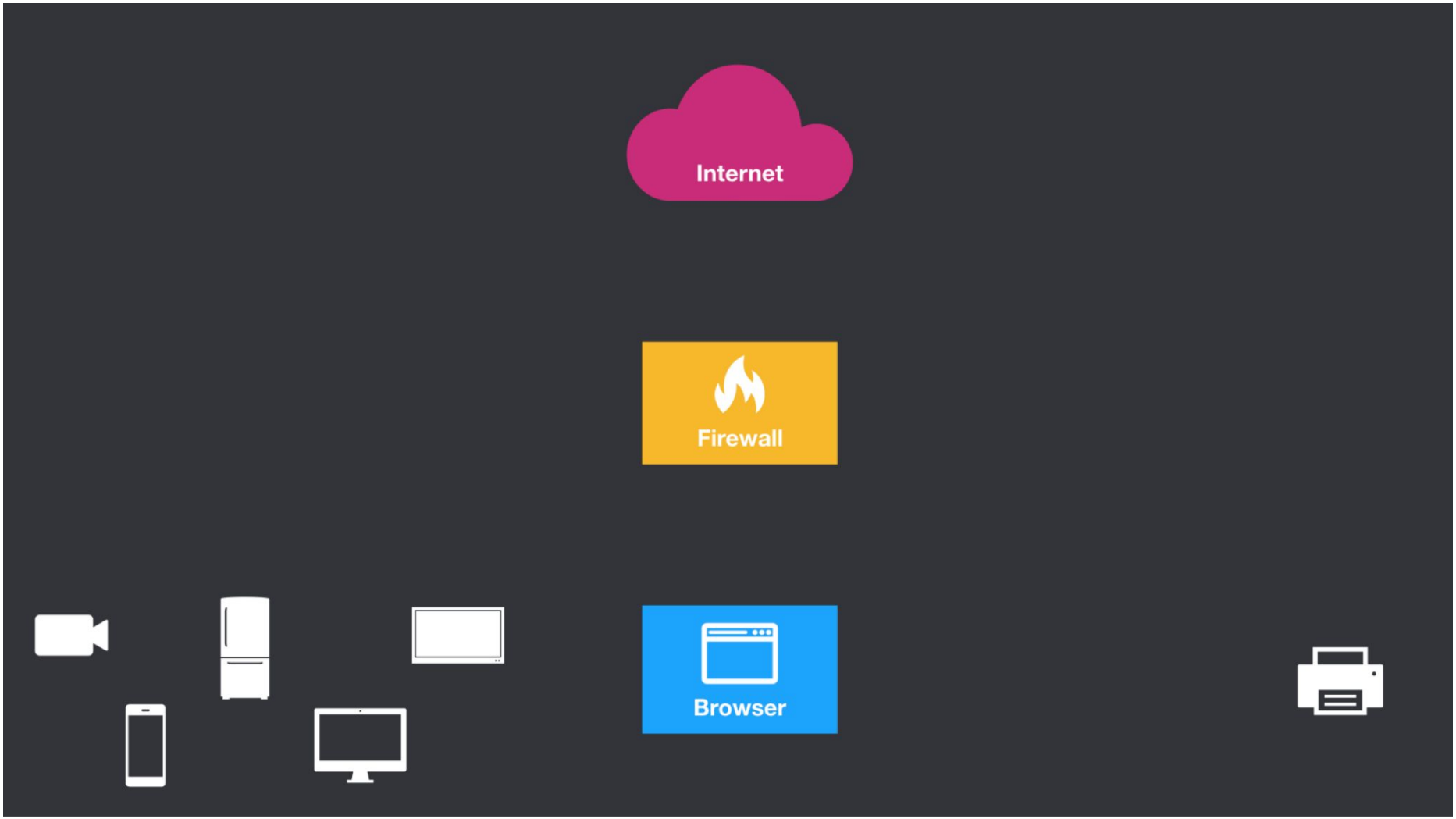

Internet
Firewall
Browser

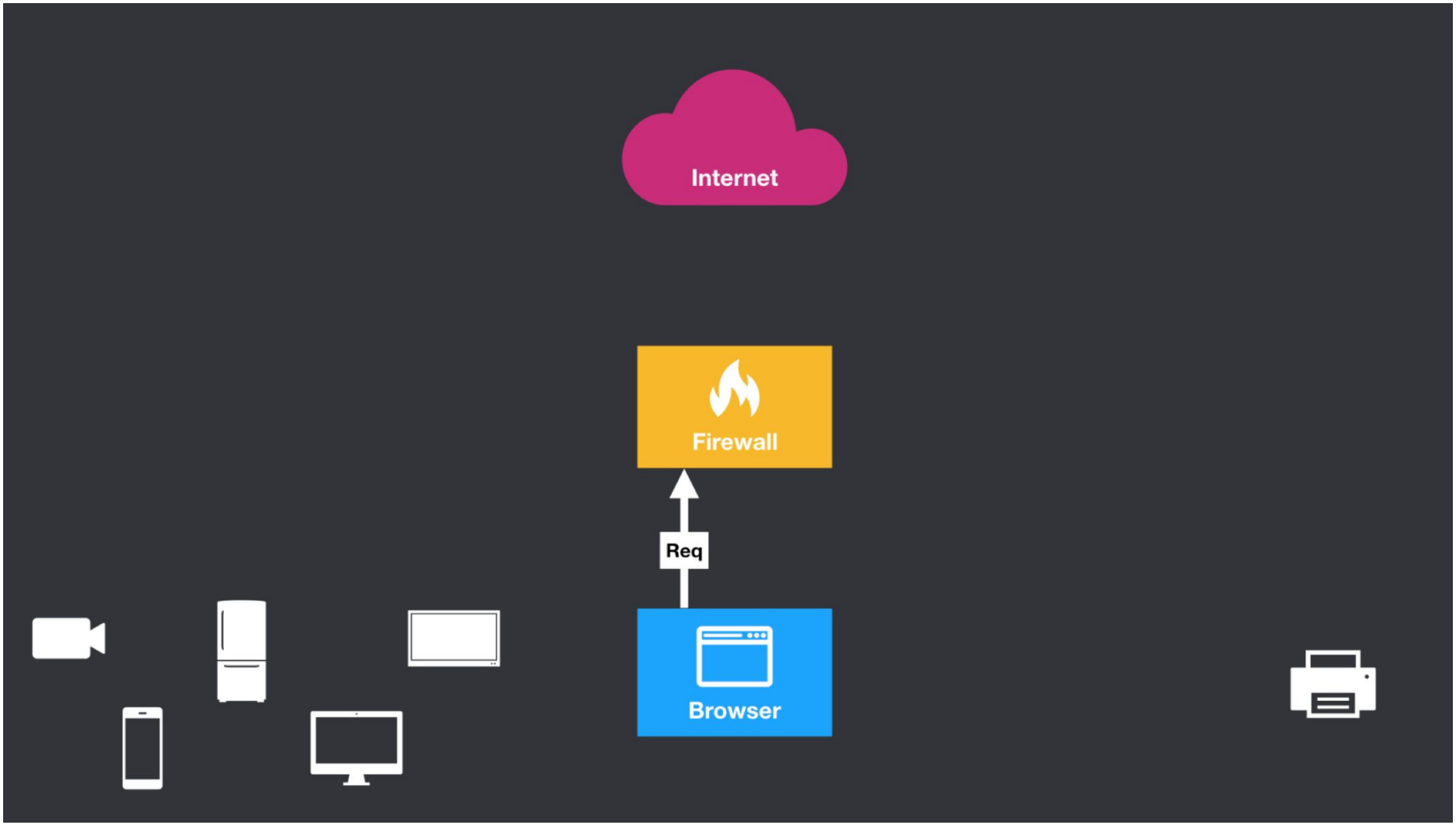
Your OS has a firewall too

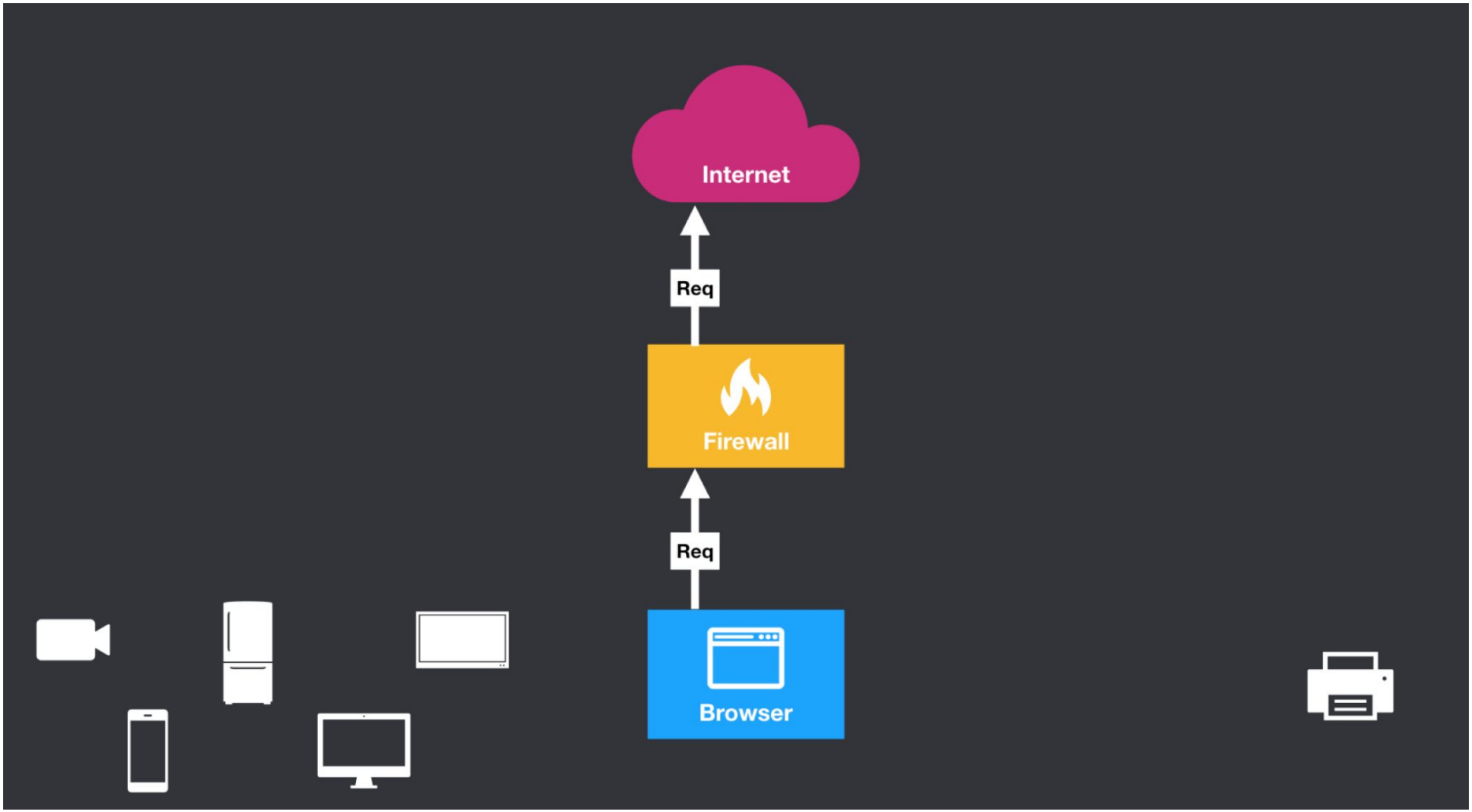


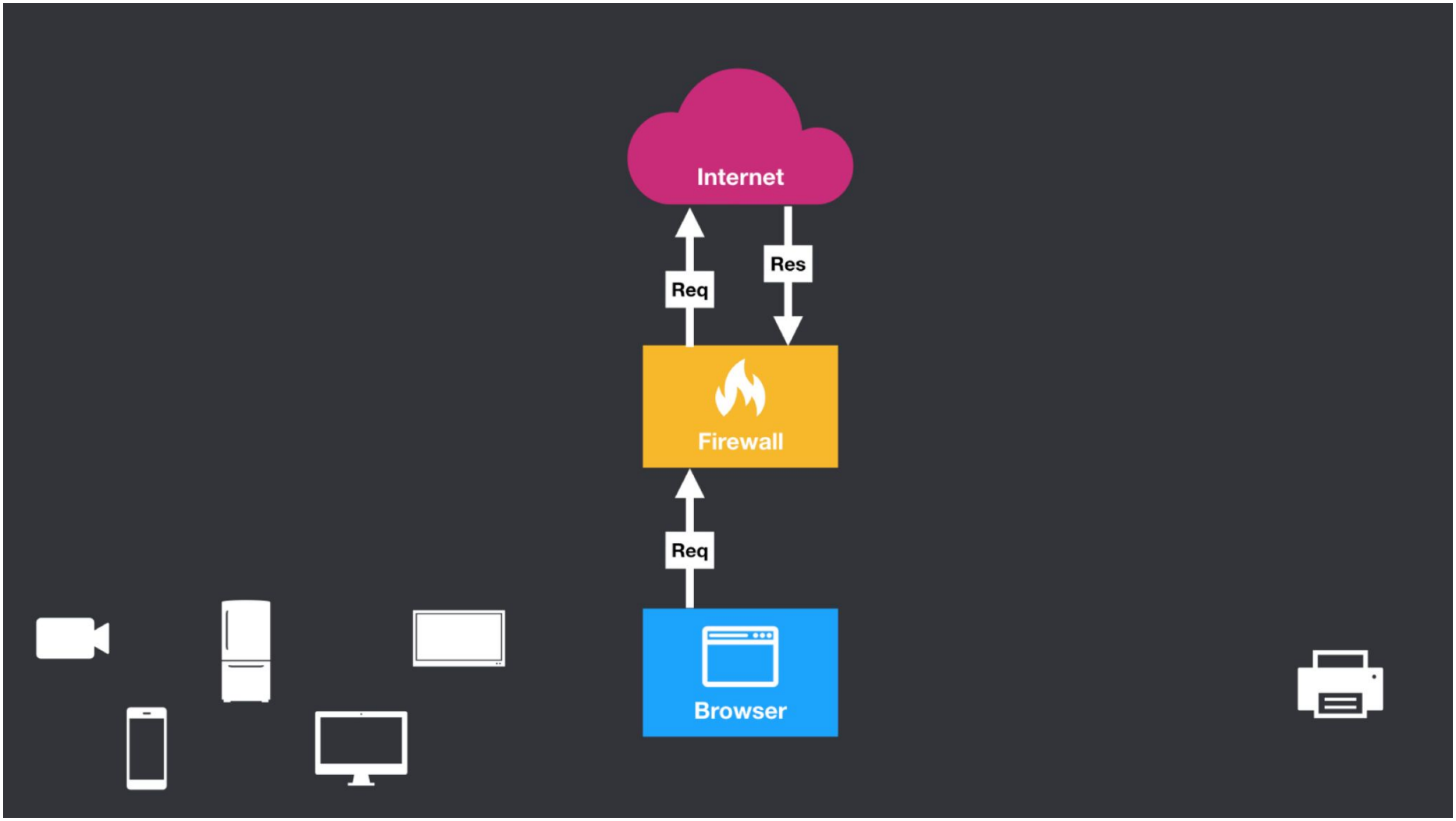


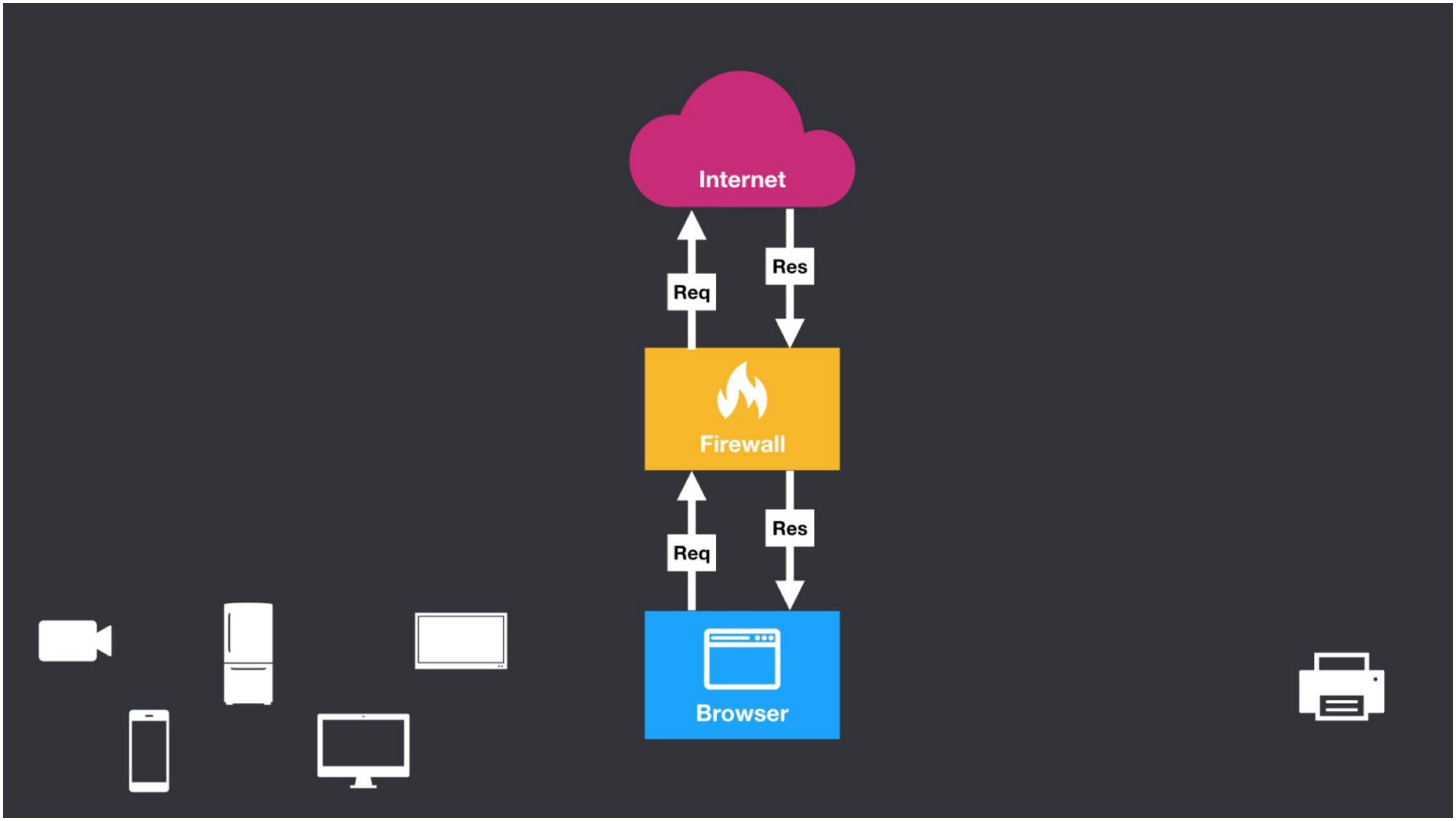
Using the browser as a proxy to make requests to the local network

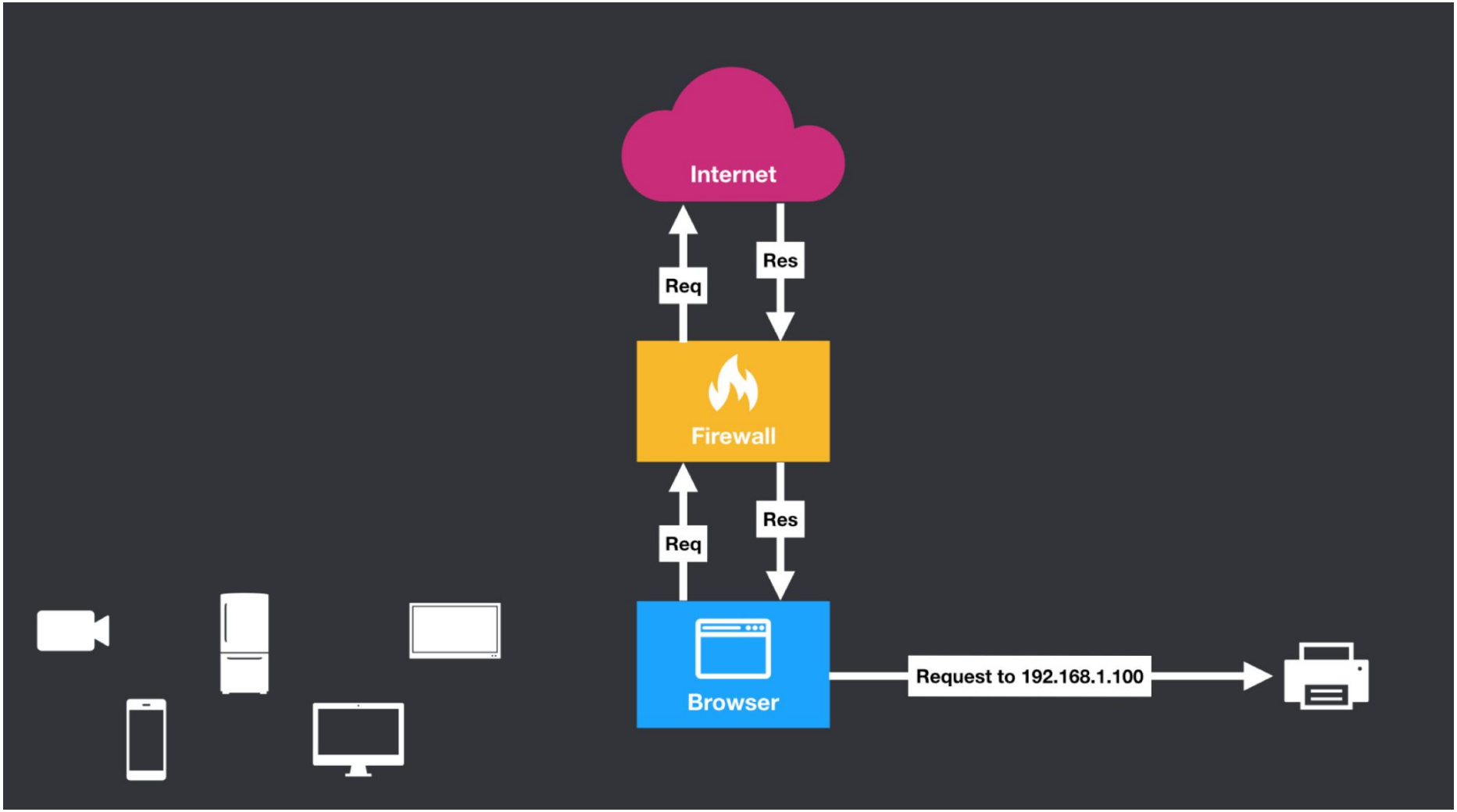

Internet
Firewall
Browser

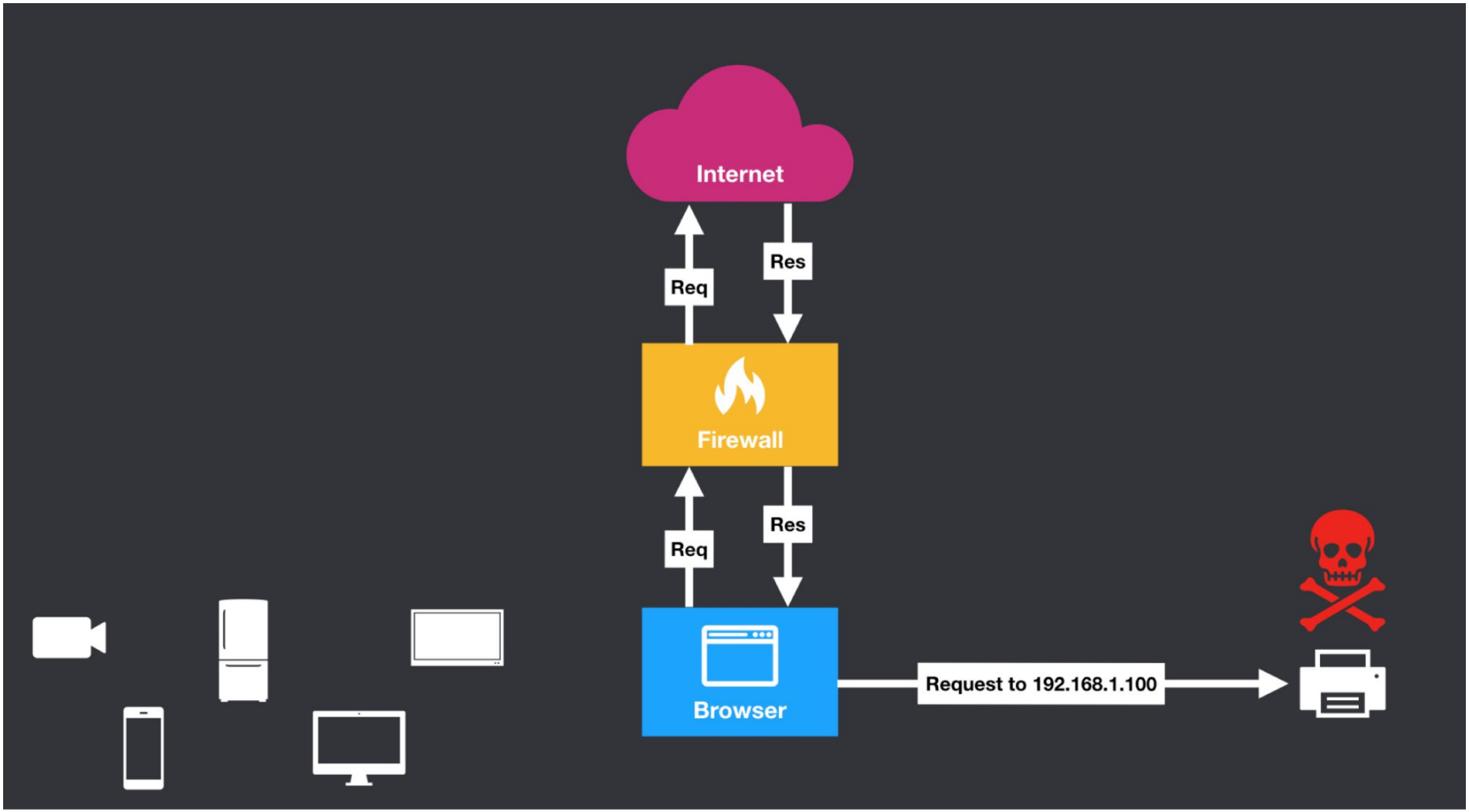

Internet
Firewall
Req
Browser


Internet
Req
Firewall
Req
Browser

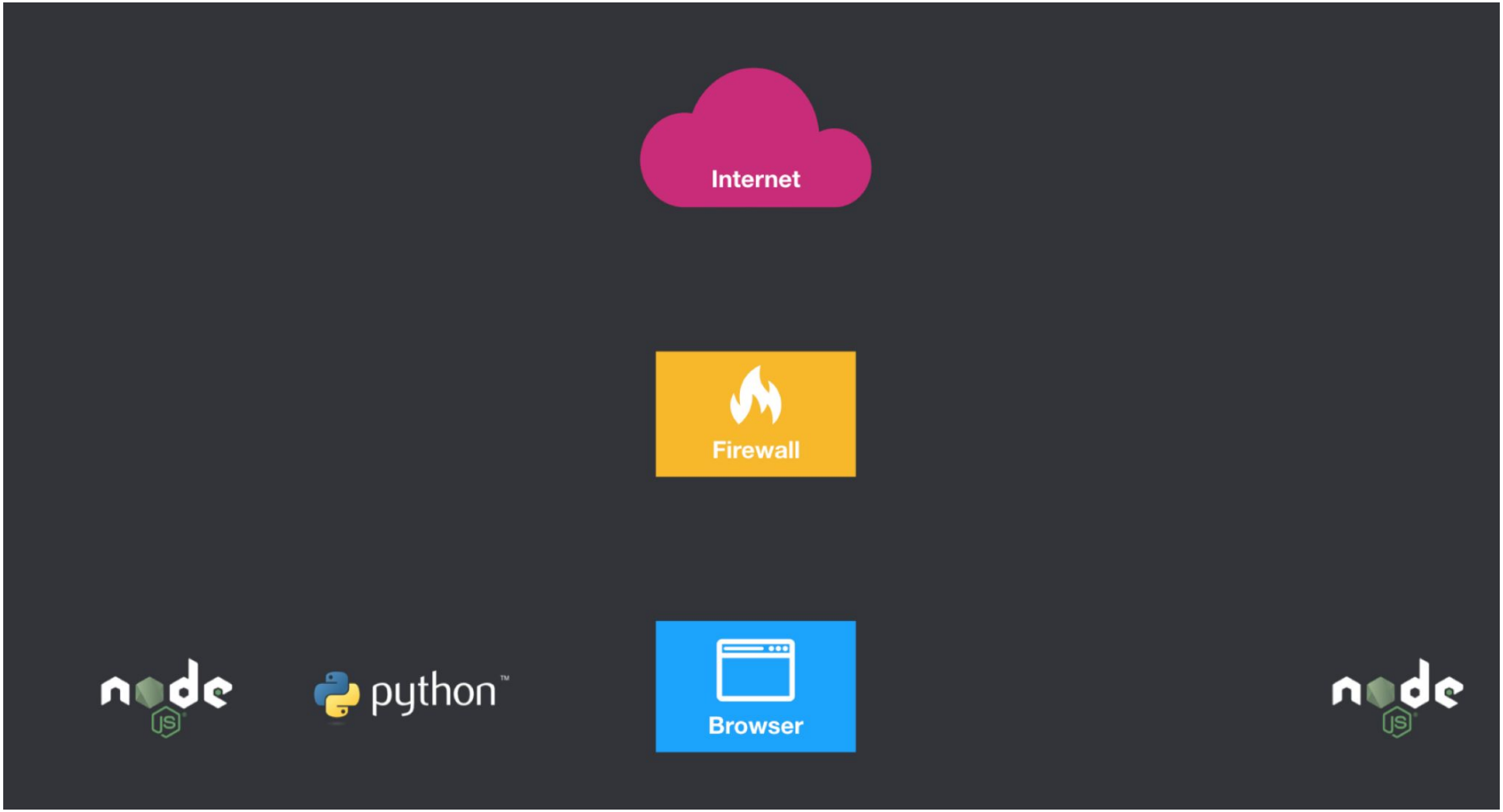

Internet
Req
Res
Firewall
Req
Browser

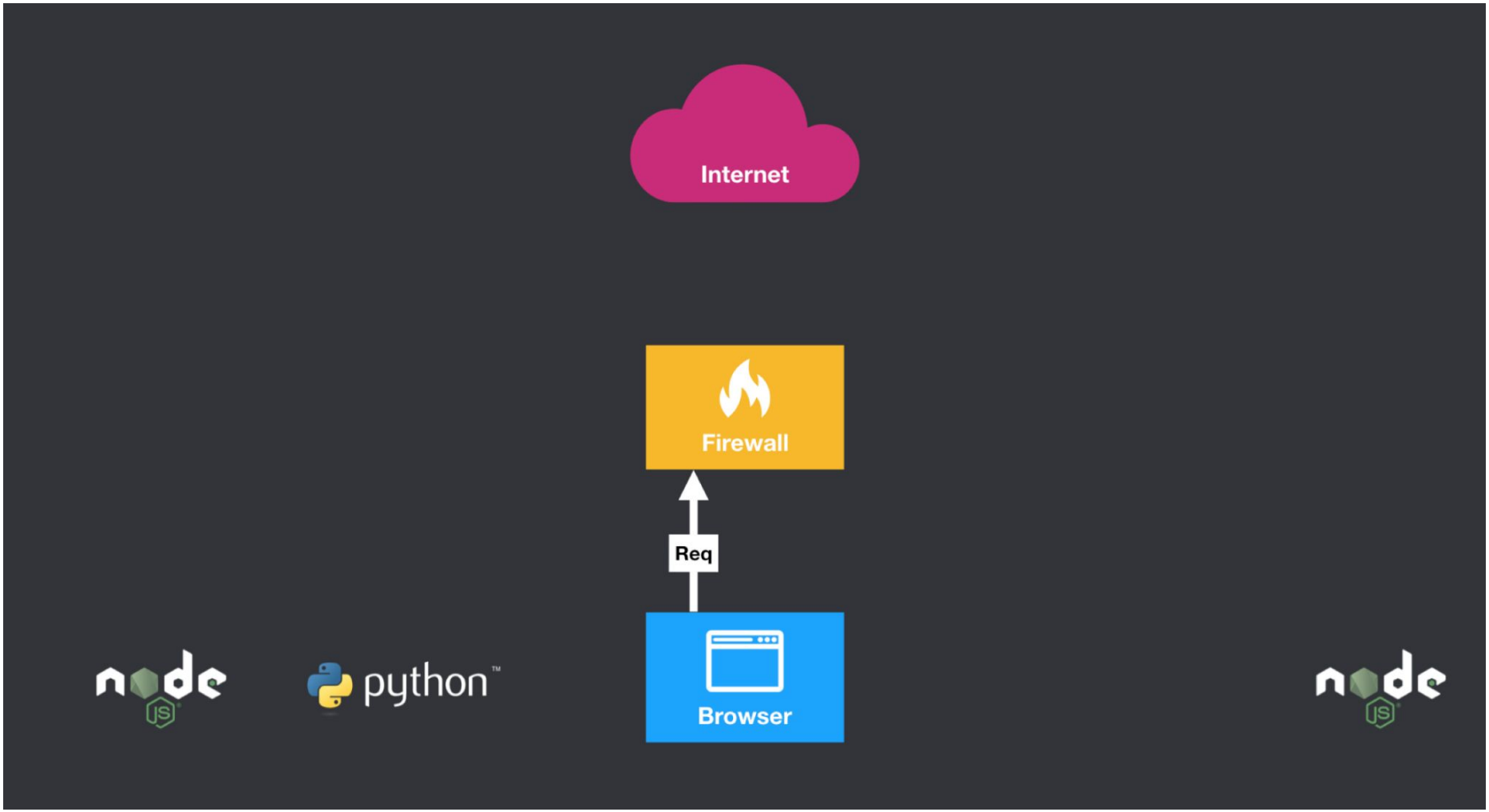

Internet
Req
Res
Firewall
Req
Res
Browser

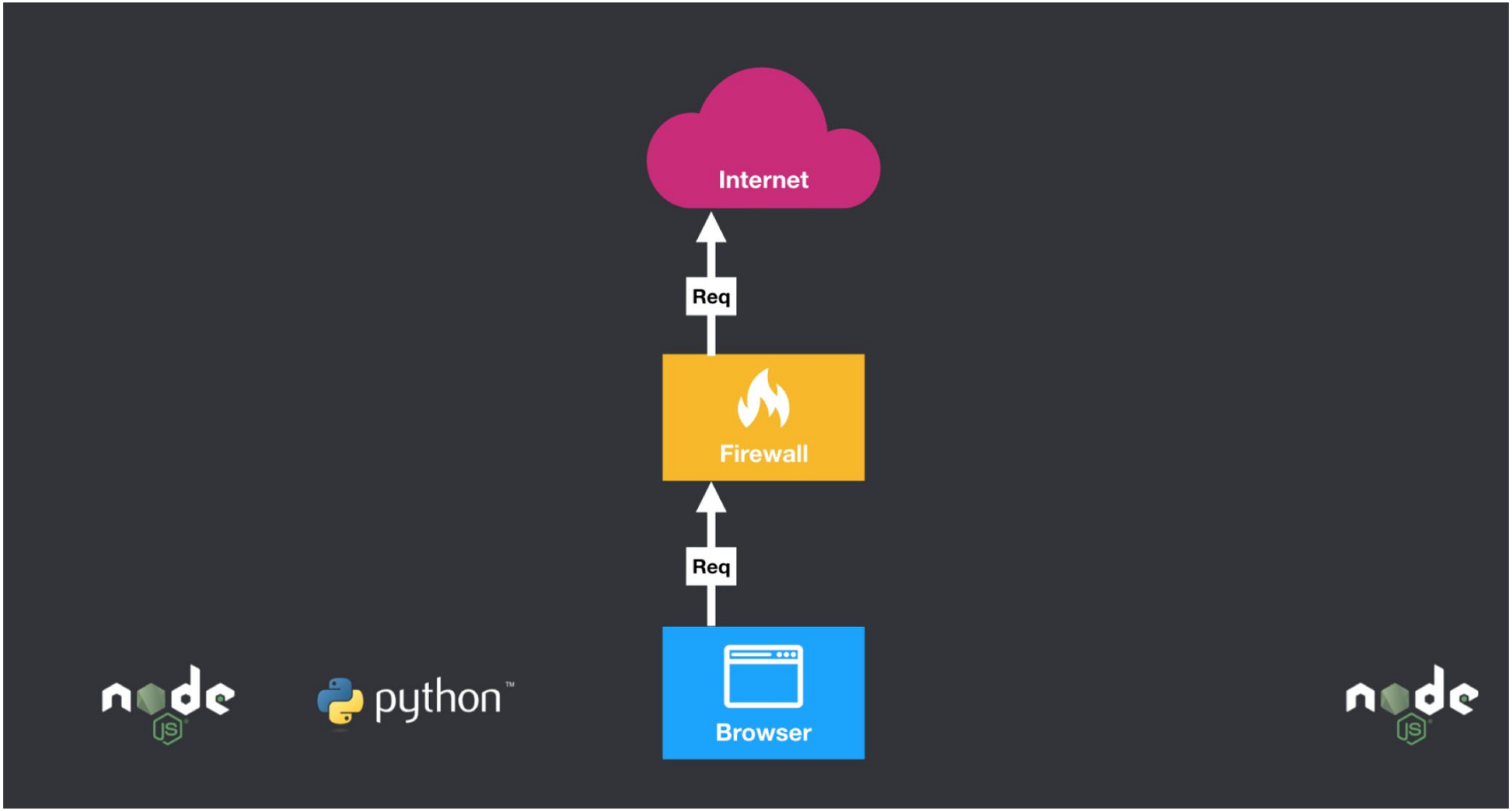

Internet
Req
Res
Firewall
Req
Res
Browser
Request to 192.168.1.100

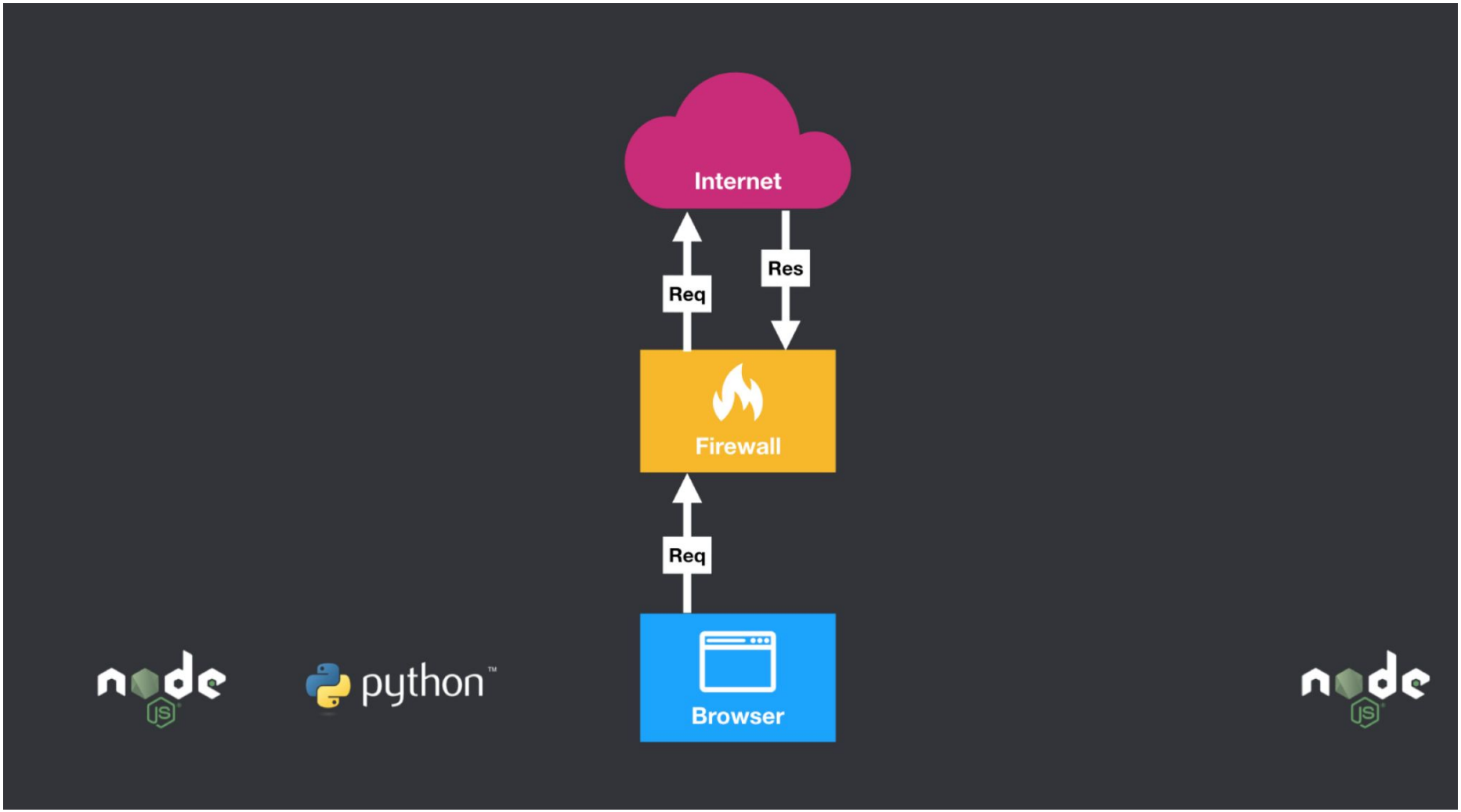

Internet
Req
Res
Firewall
Req
Res
Browser
Request to 192.168.1.100

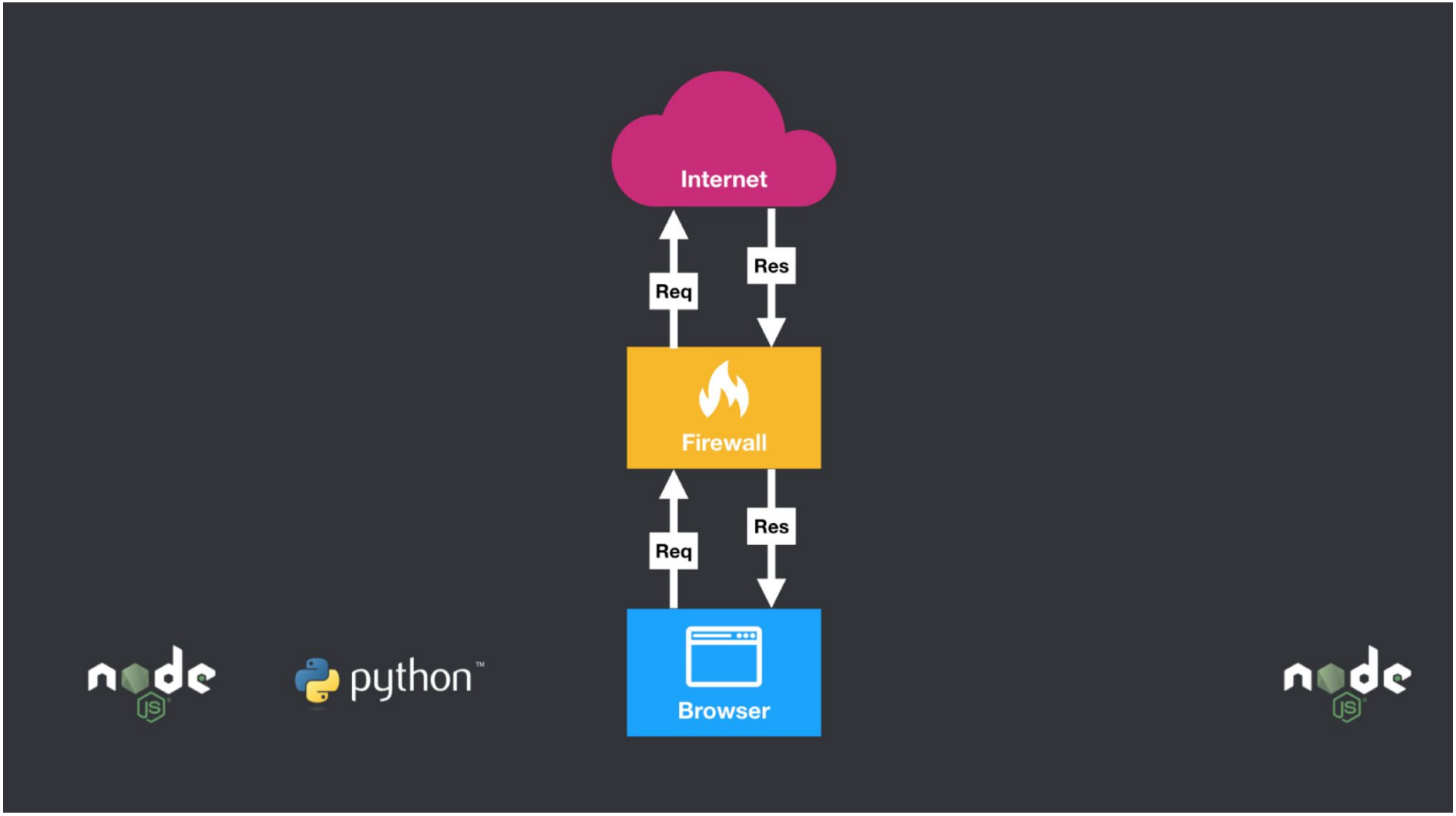
Using the browser as a proxy to make requests to local HTTP servers

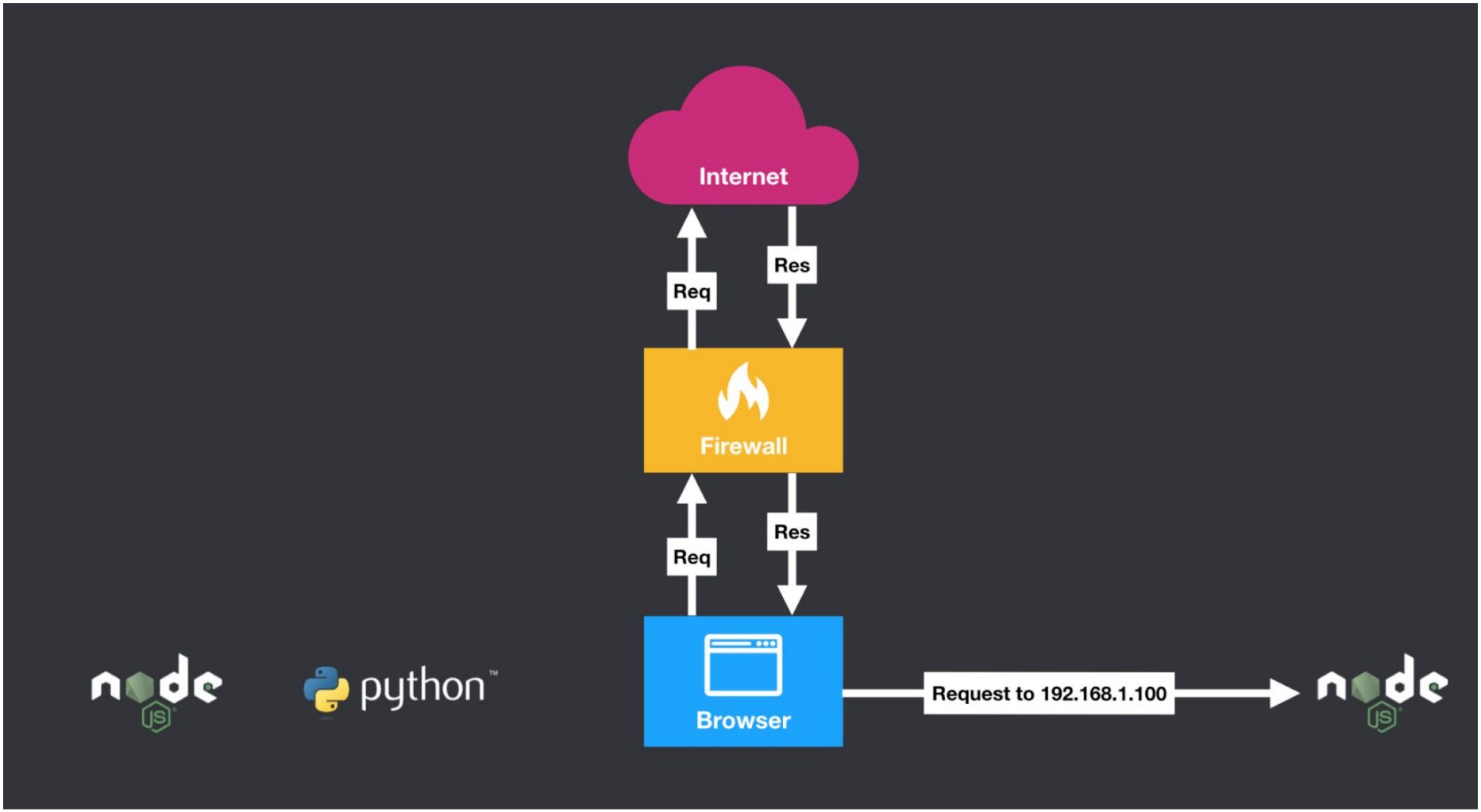

Internet
Firewall
Browser
node
python
node

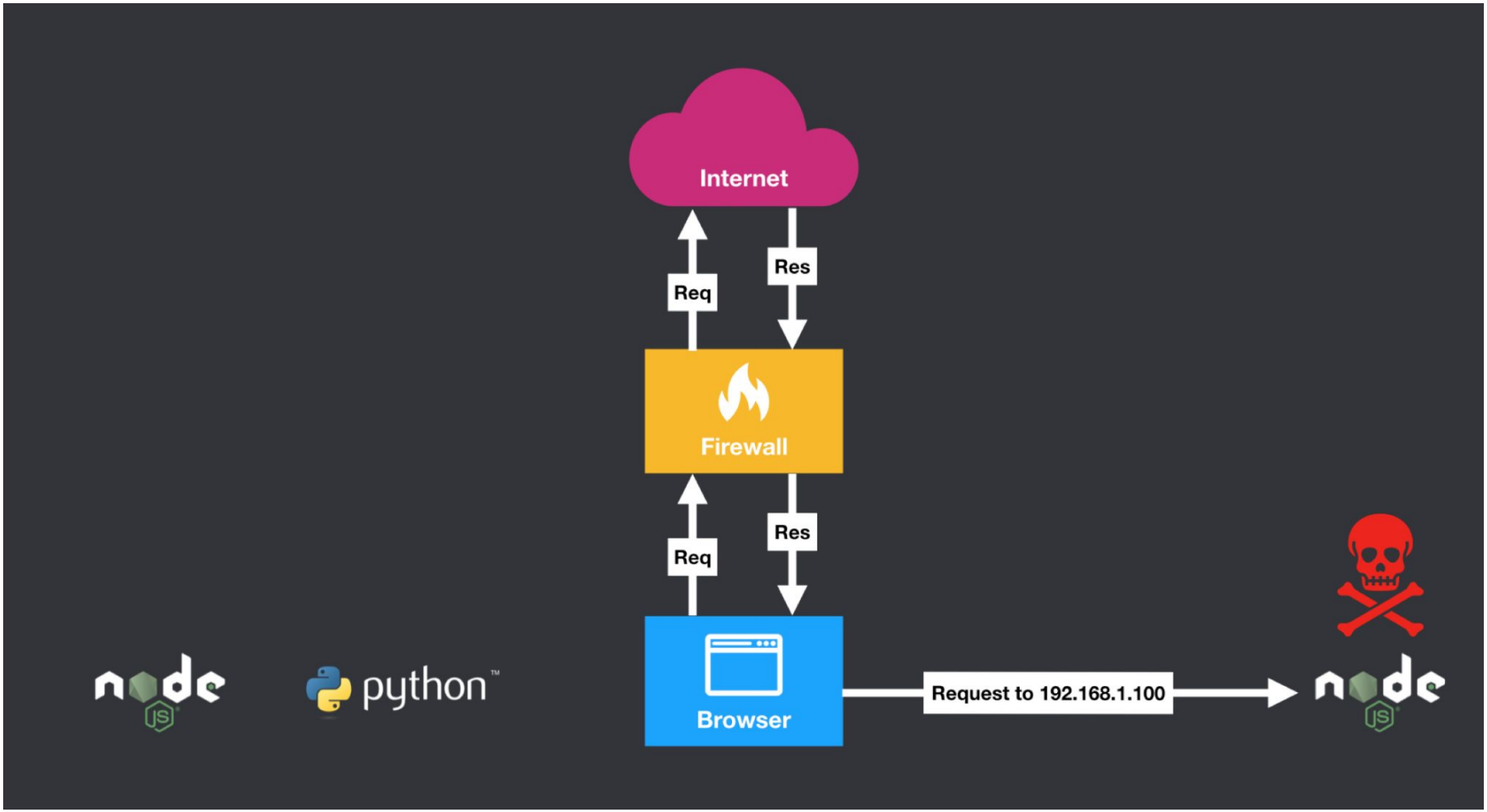

Internet
Firewall
Req
Browser
node
python
node


Internet
Req
Firewall
Req
Browser
node
python
node


Internet
Res
Req
Firewall
Req
Browser
node
python
node

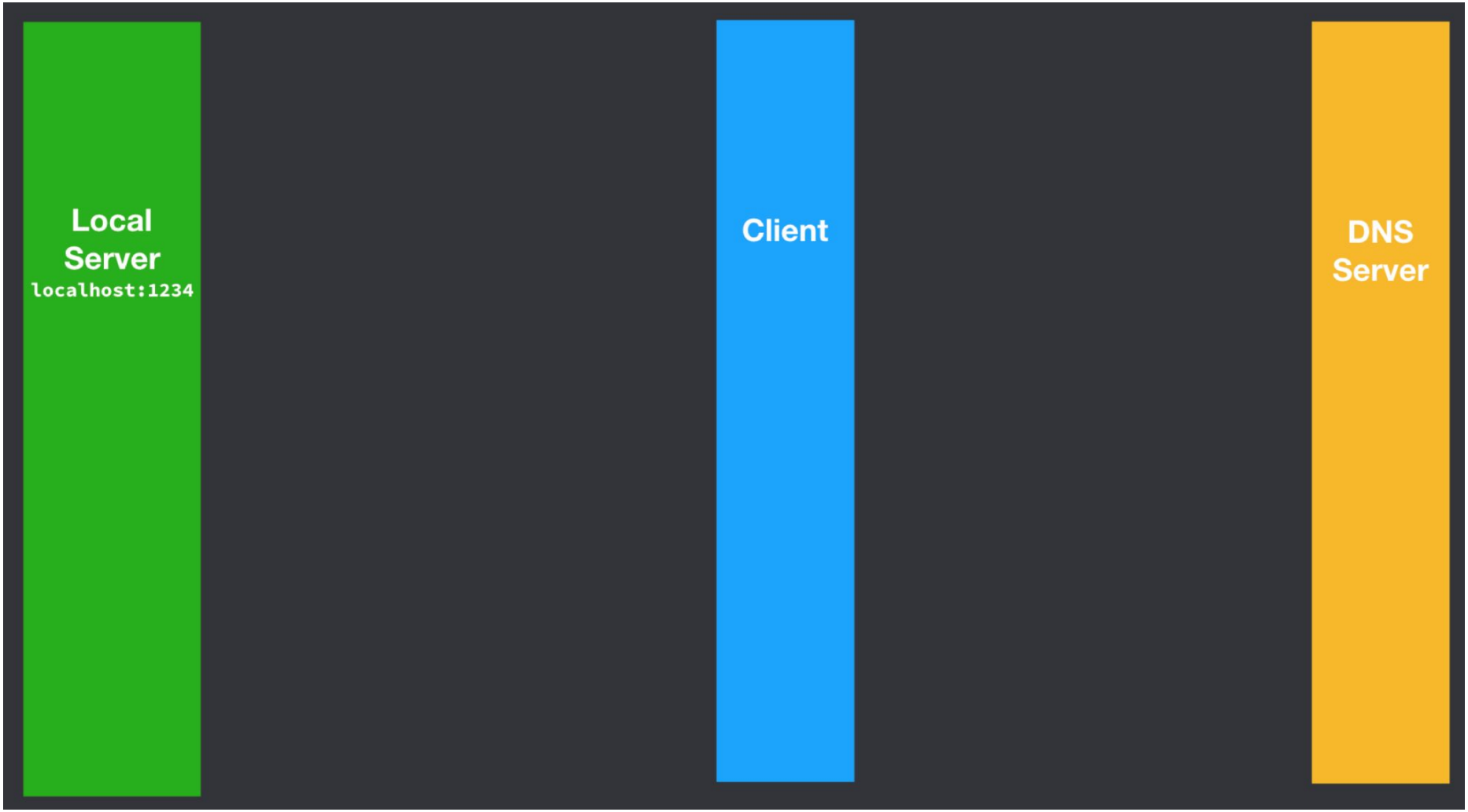

Internet
Req
Res
Firewall
Req
Res
Browser
node
python
node

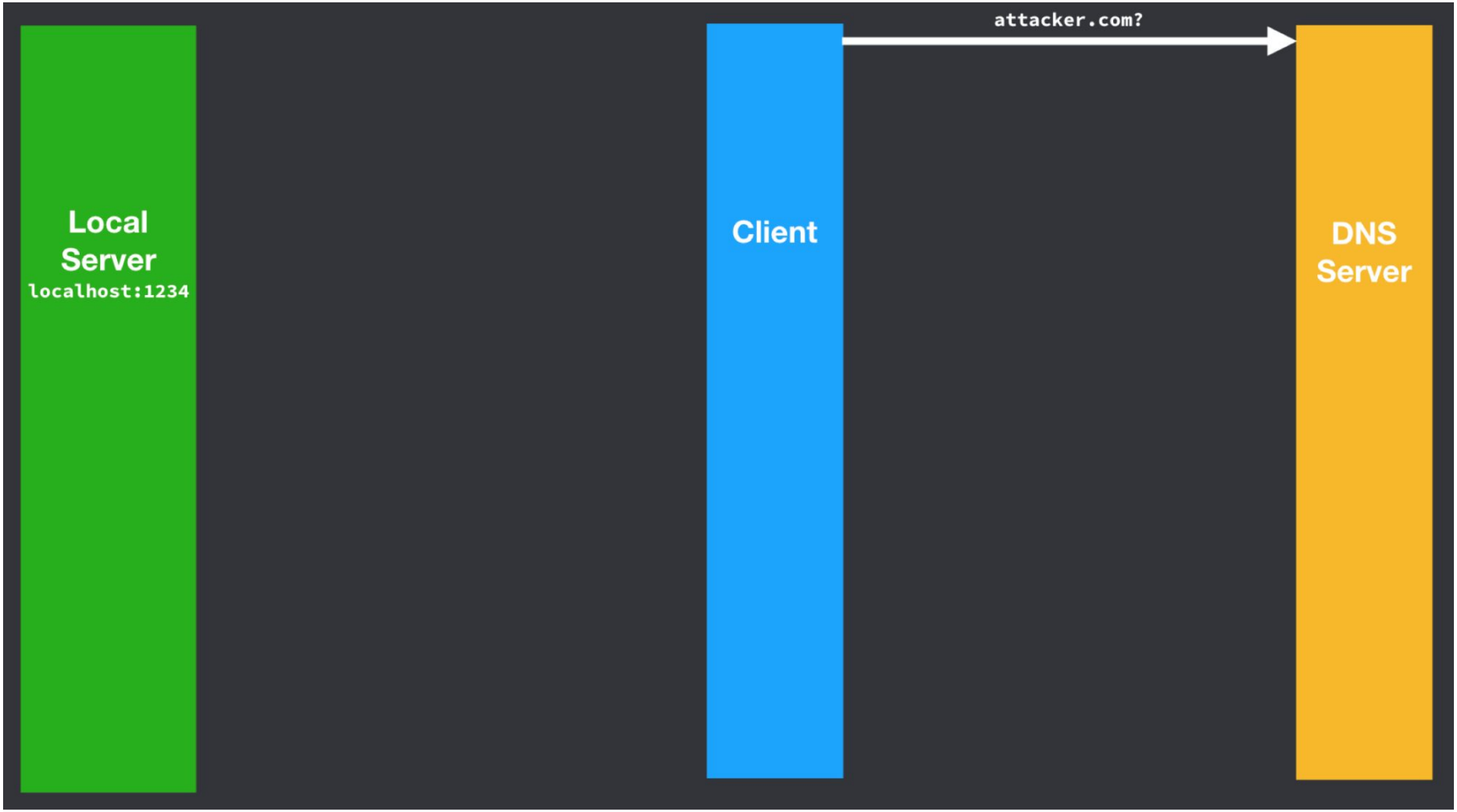

Internet
Req
Res
Firewall
Req
Res
Browser
Request to 192.168.1.100

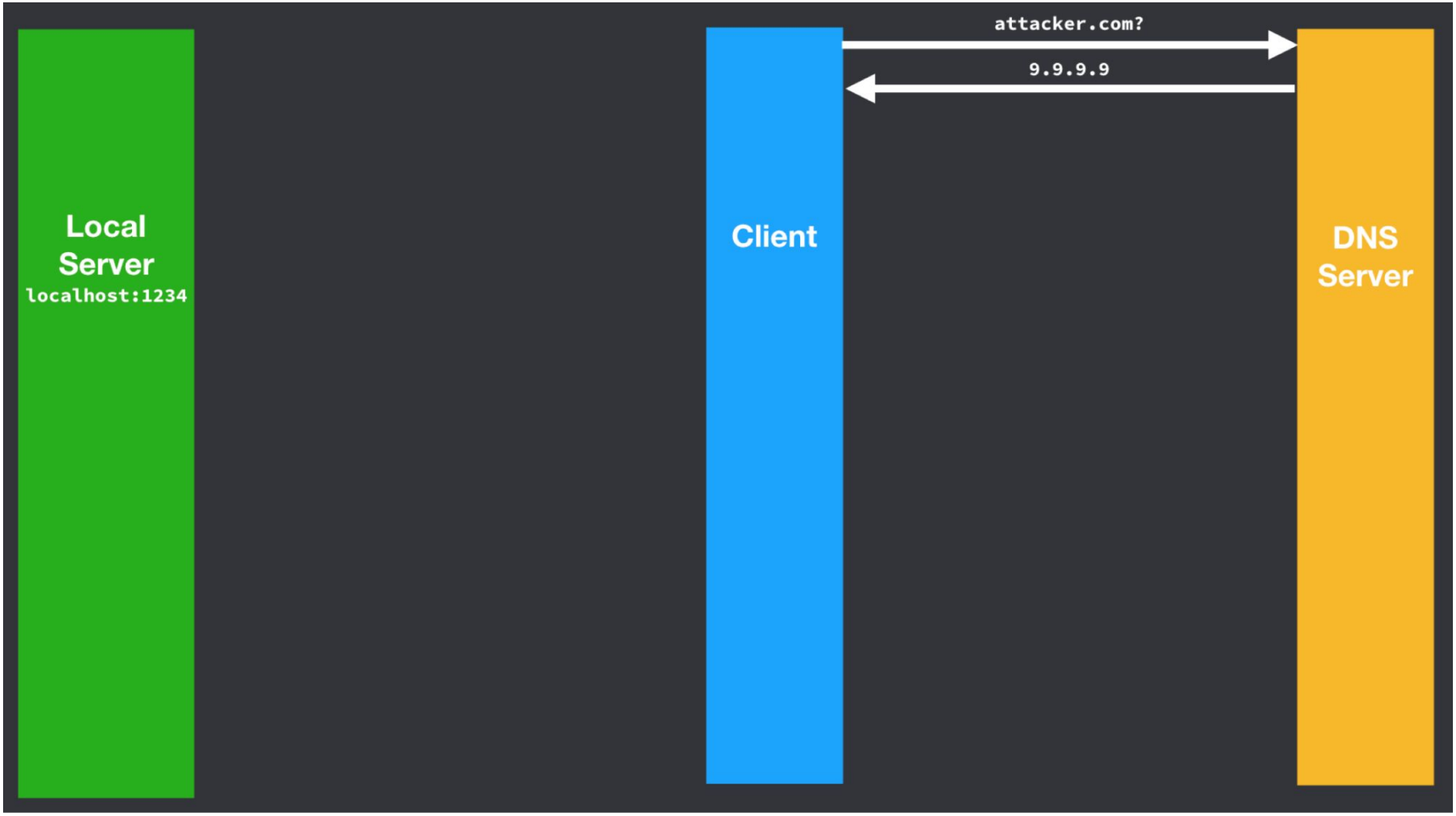

Internet
Req
Res
Firewall
Req
Res
Browser
Request to 192.168.1.100
node
python
node

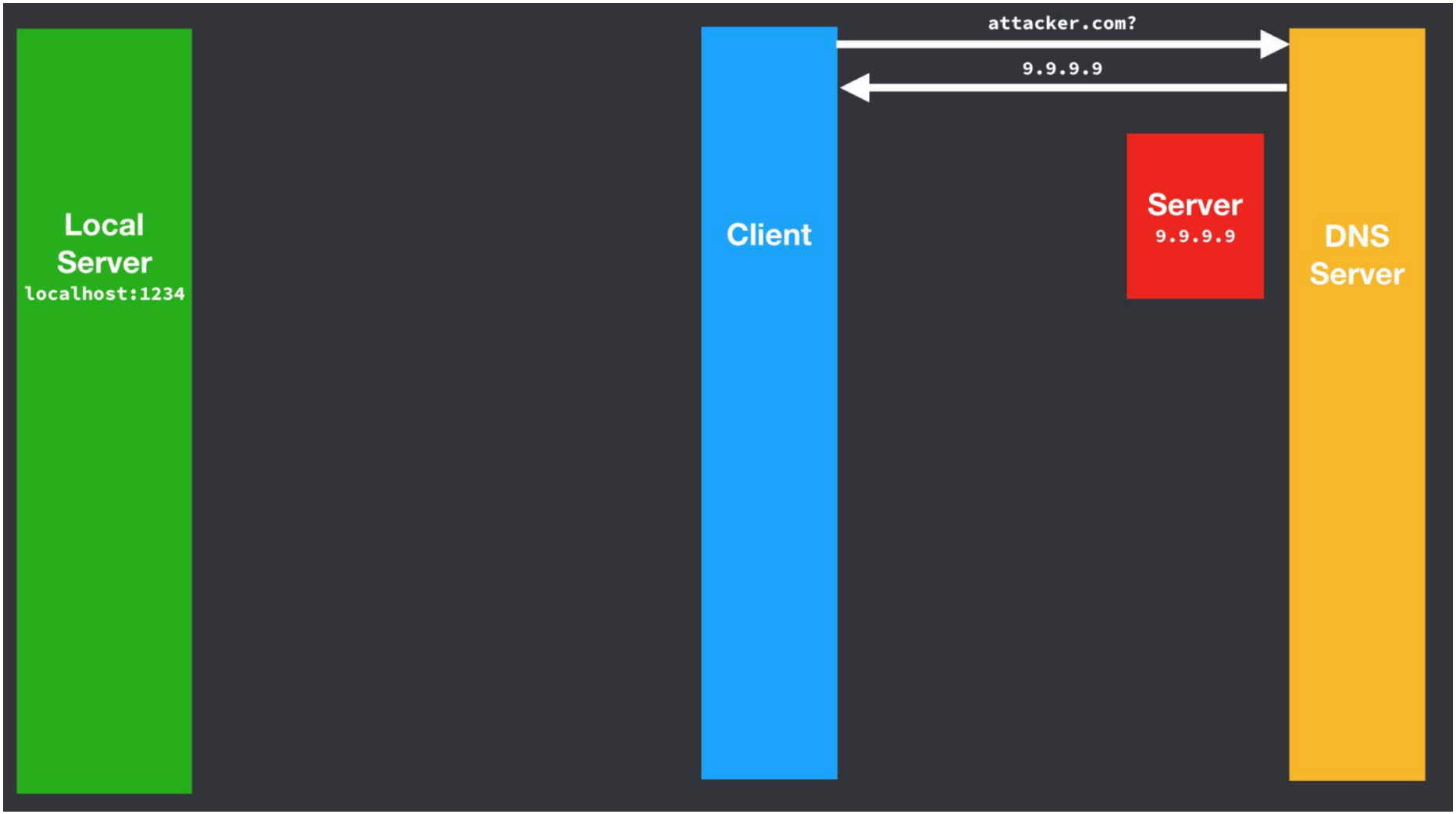
Demo DNS

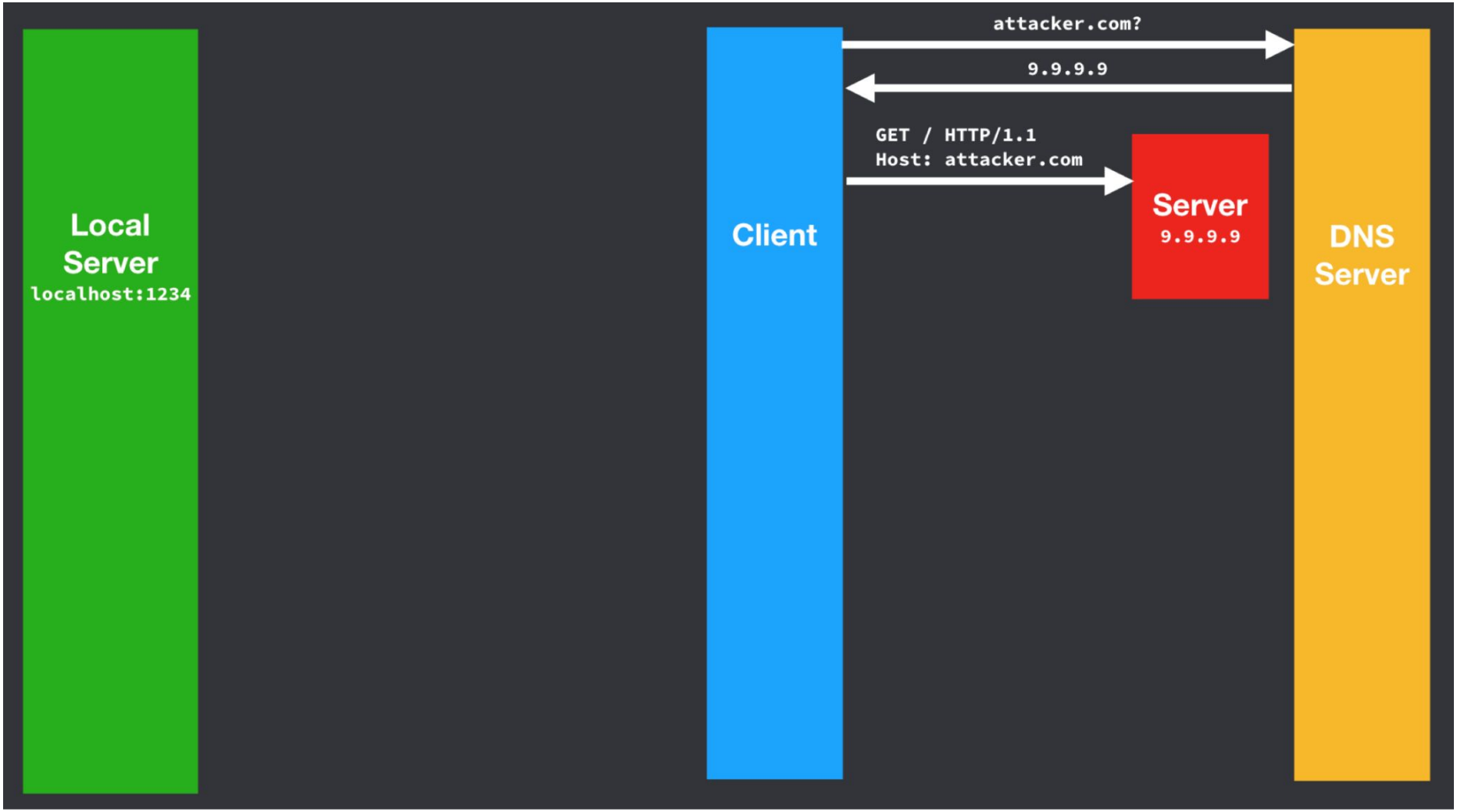
DNS rebinding attack

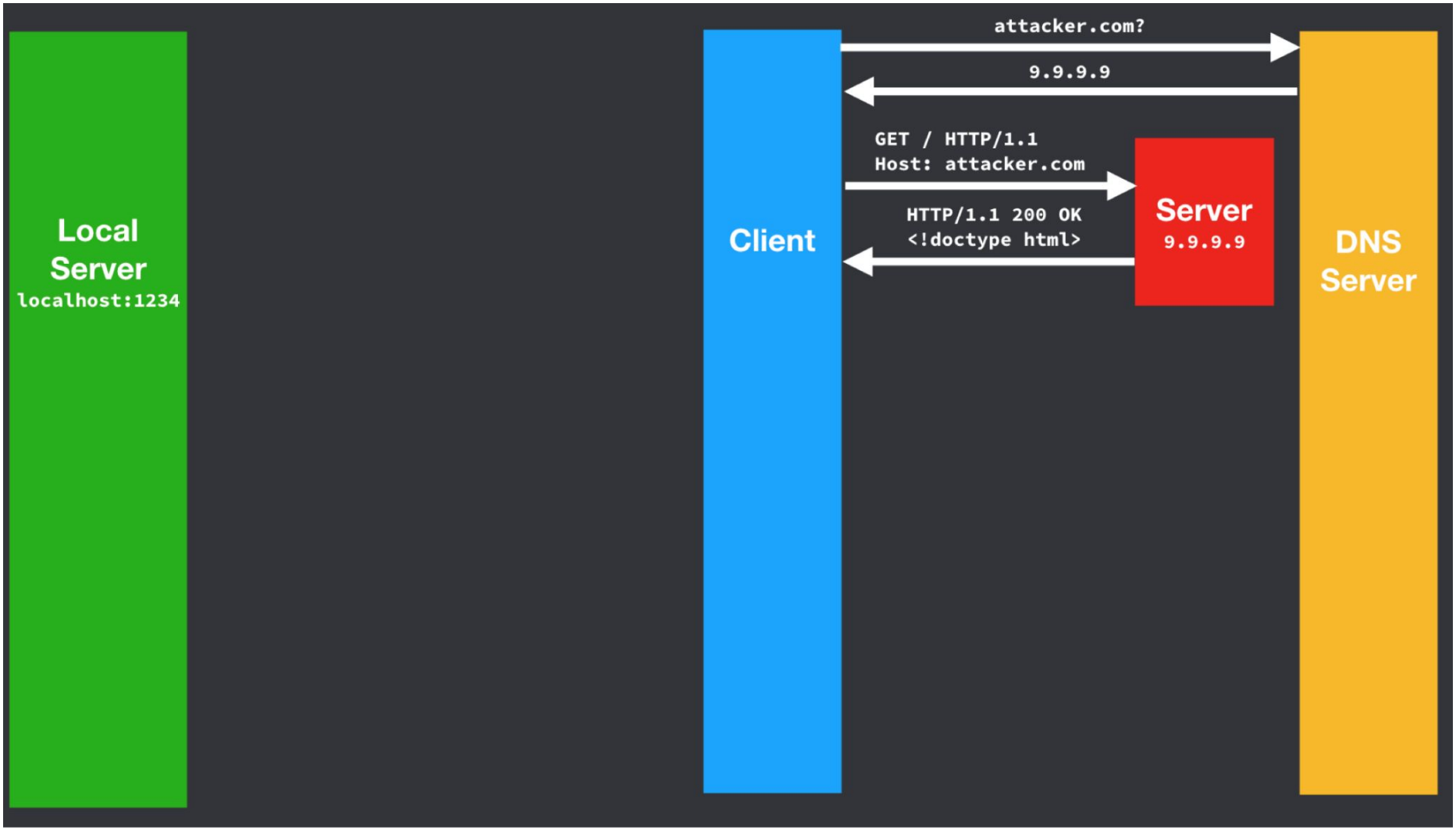

Local
Server
localhost:1234
Client
DNS
Server

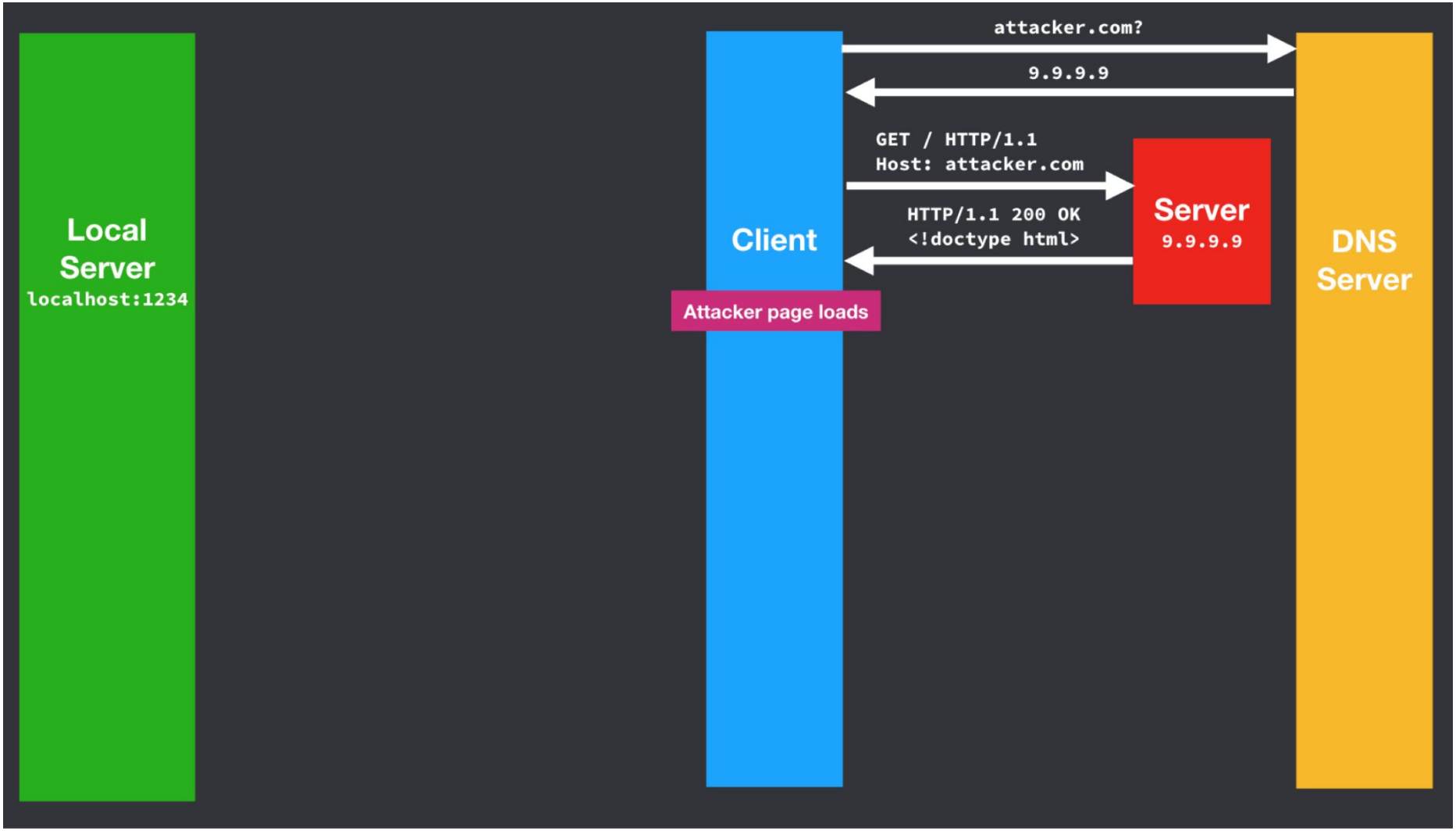

attacker.com?
Local Server
localhost:1234
Client
DNS Server

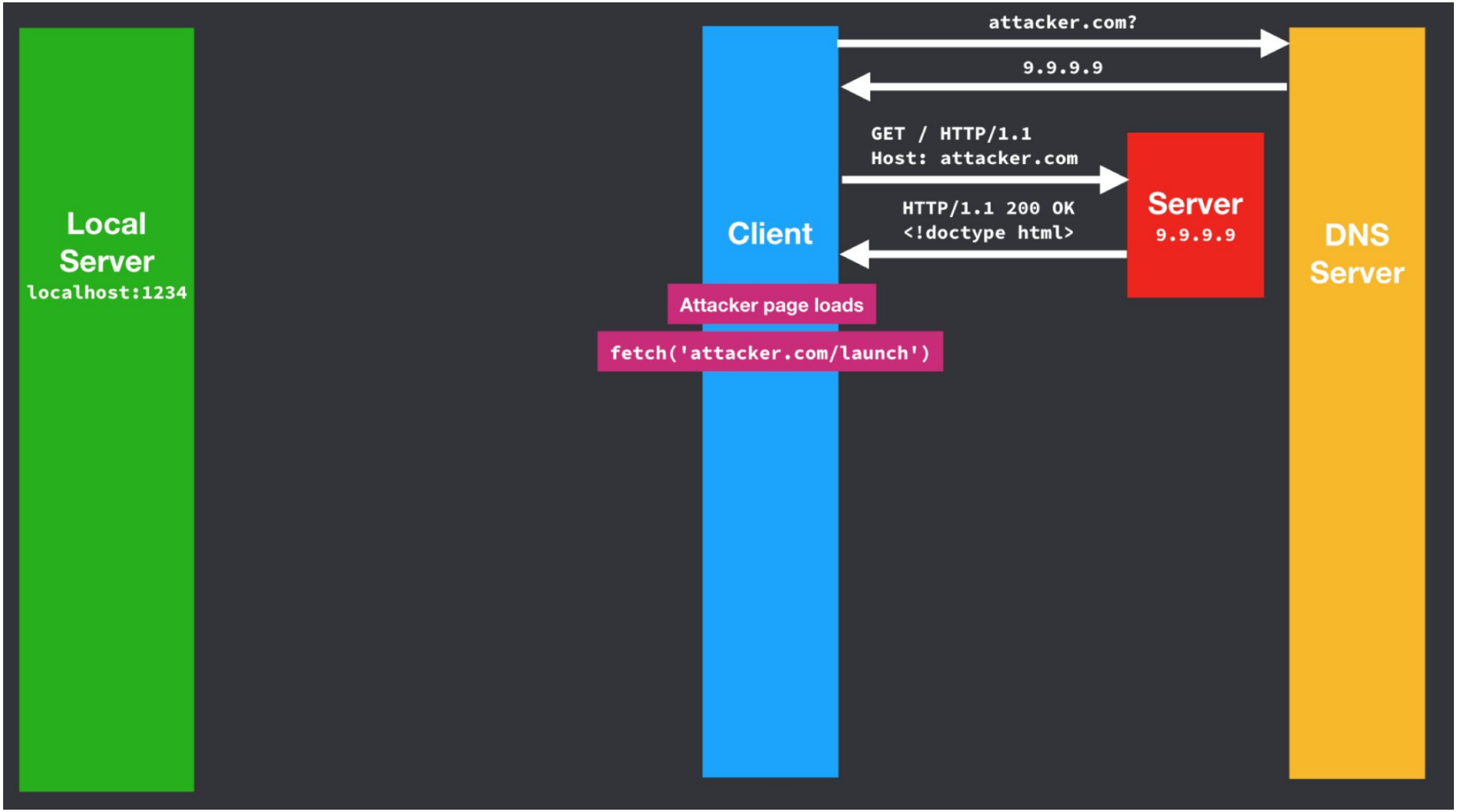

Local Server
localhost:1234
Client
attacker.com?
9.9.9.9
DNS Server

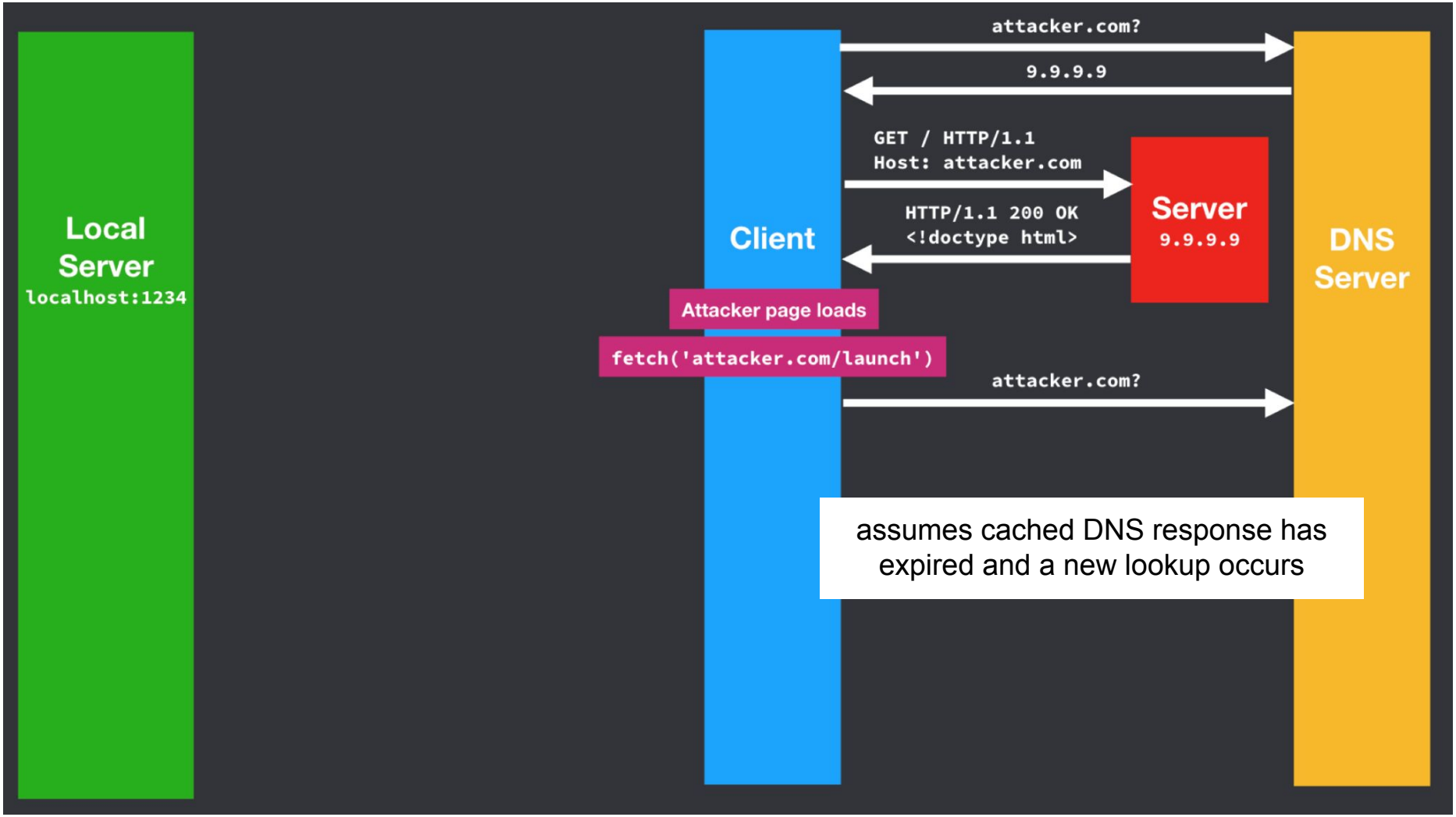

Local Server
localhost:1234
Client
attacker.com?
9.9.9.9
Server
9.9.9.9
DNS Server

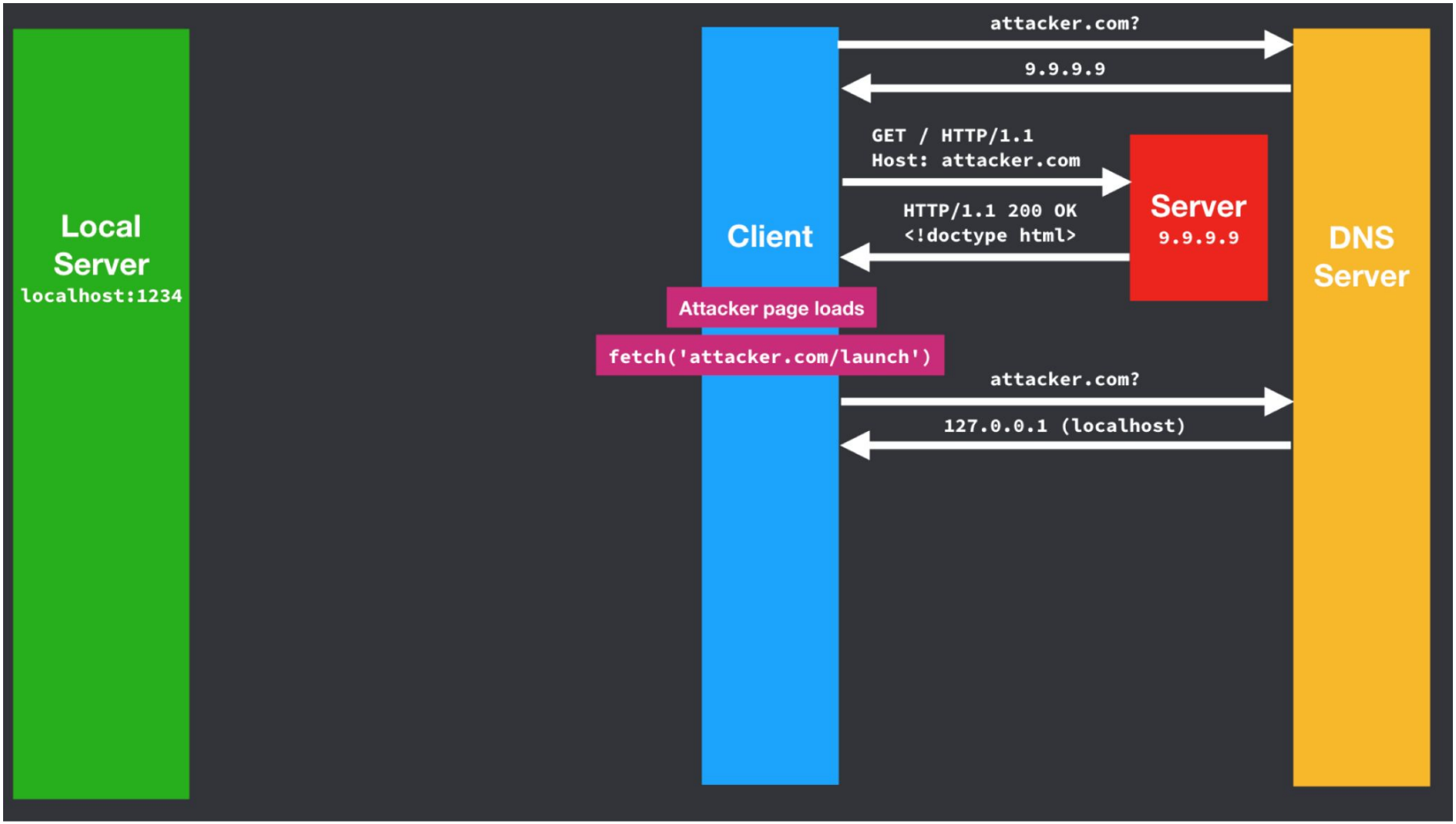

Local Server
localhost:1234
Client
attacker.com?
9.9.9.9
GET / HTTP/1.1
Host: attacker.com
Server
9.9.9.9
DNS Server

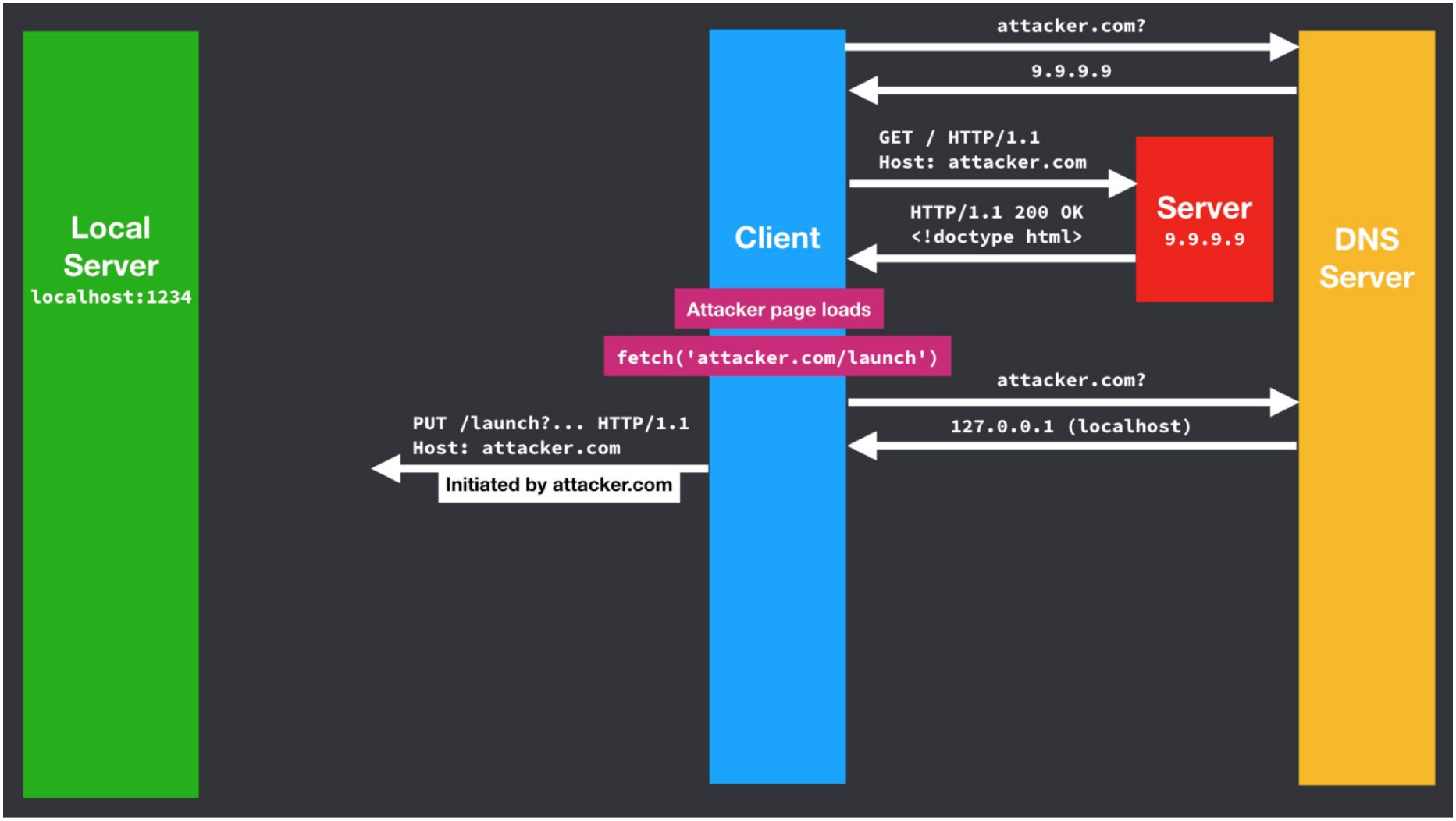

Local Server
localhost:1234
Client
attacker.com?
9.9.9.9
GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
Server
9.9.9.9
DNS Server

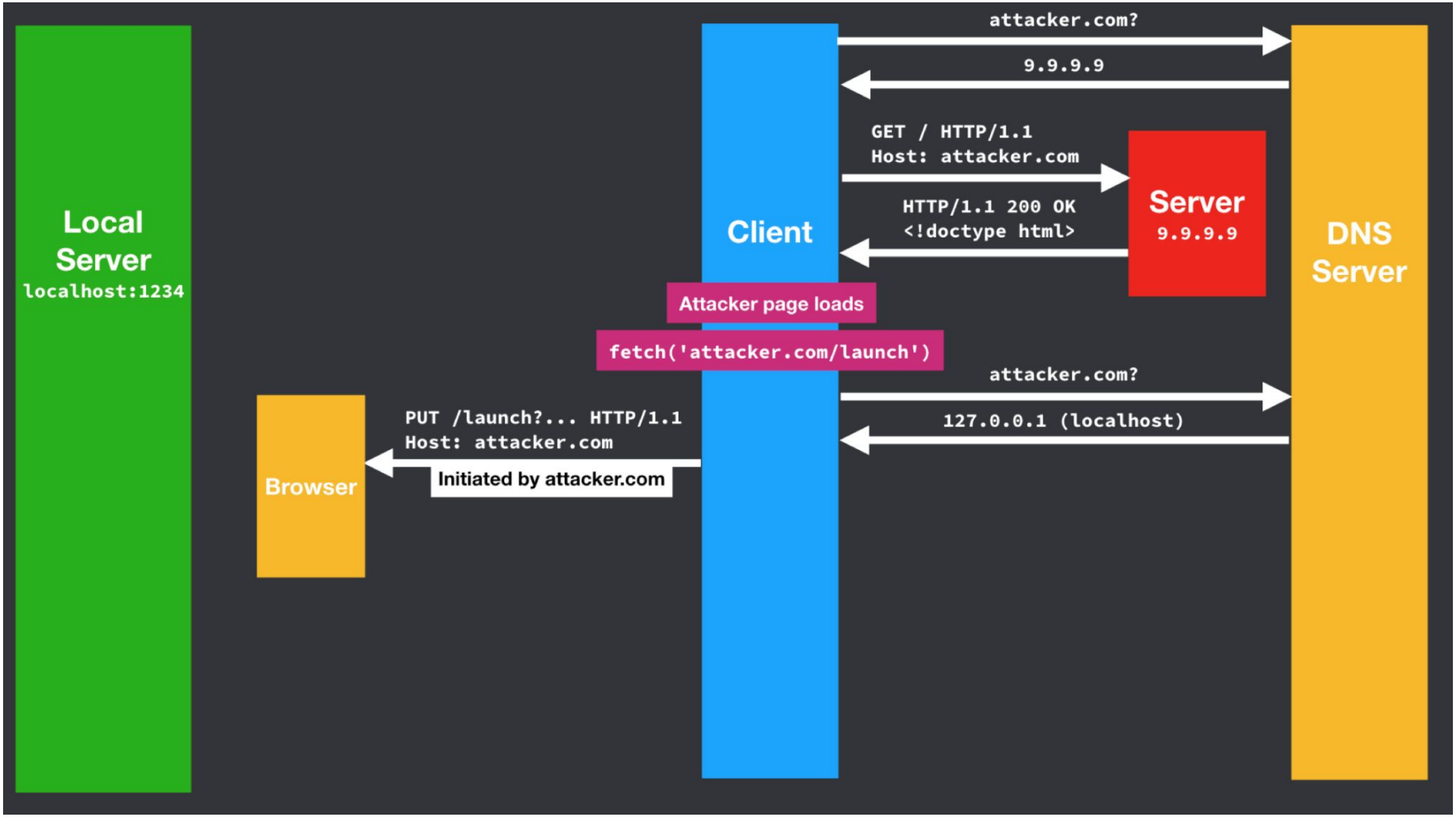

Local Server
localhost:1234
Client
Attacker page loads
attacker.com?
9.9.9.9
GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
Server
9.9.9.9
DNS Server

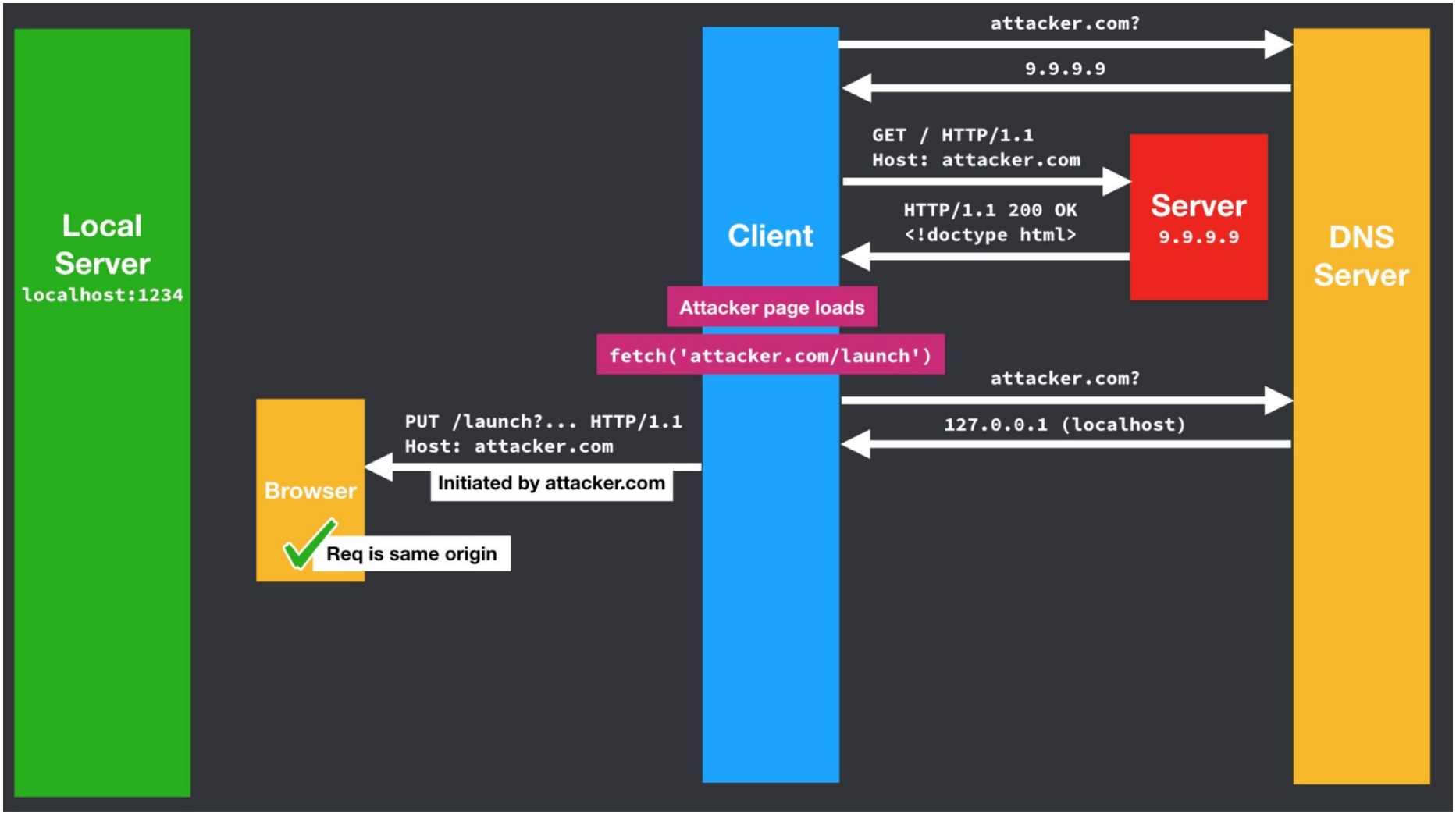

Local Server
localhost:1234
Client
attacker.com?
9.9.9.9
GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
Server
9.9.9.9
DNS Server
Attacker page loads
fetch('attacker.com/launch')

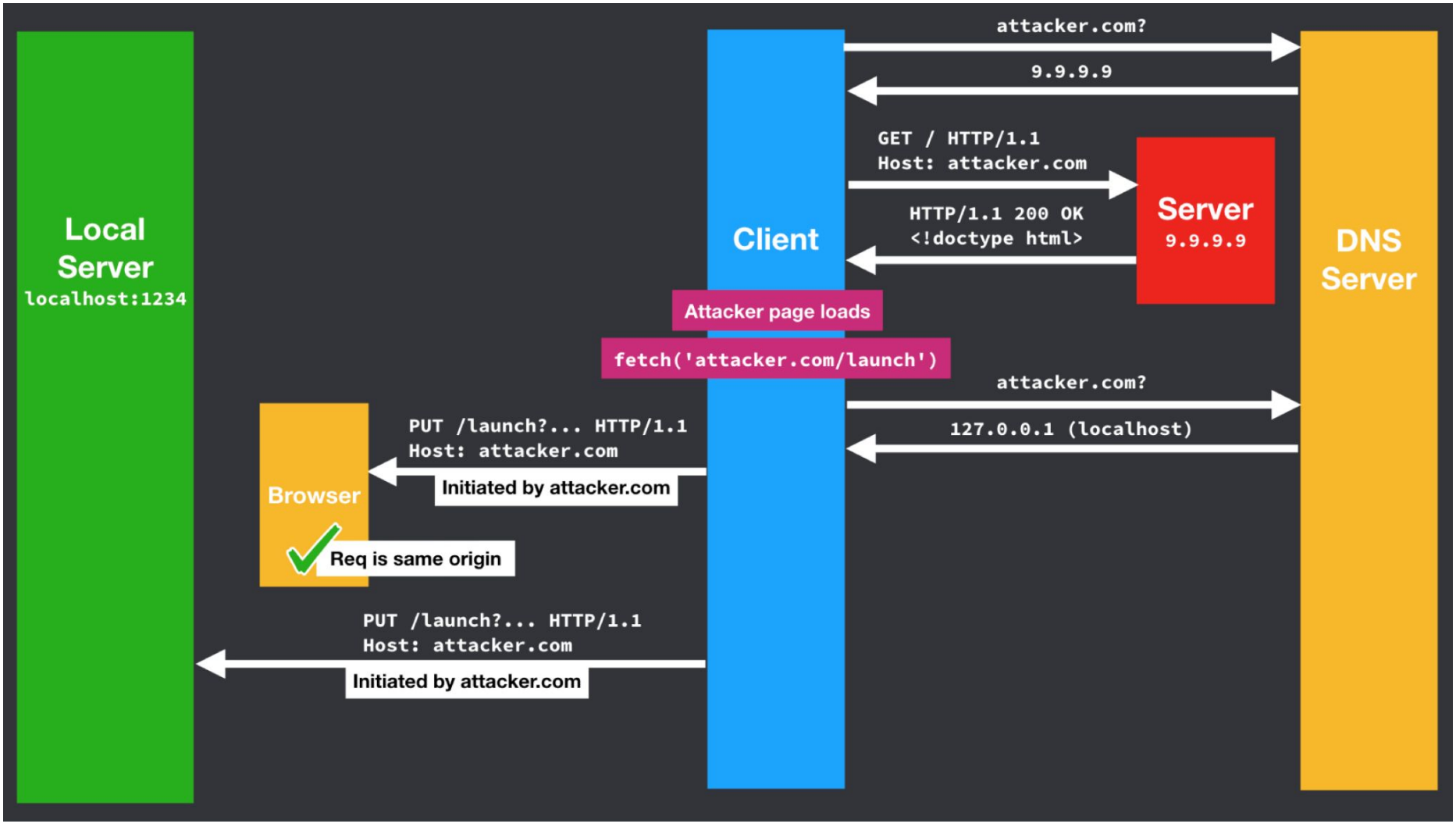

Local Server
localhost:1234
Client
attacker.com?
9.9.9.9
GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
Server
9.9.9.9
DNS Server
Attacker page loads
fetch('attacker.com/launch')
attacker.com?
assumes cached DNS response has expired and a new lookup occurs

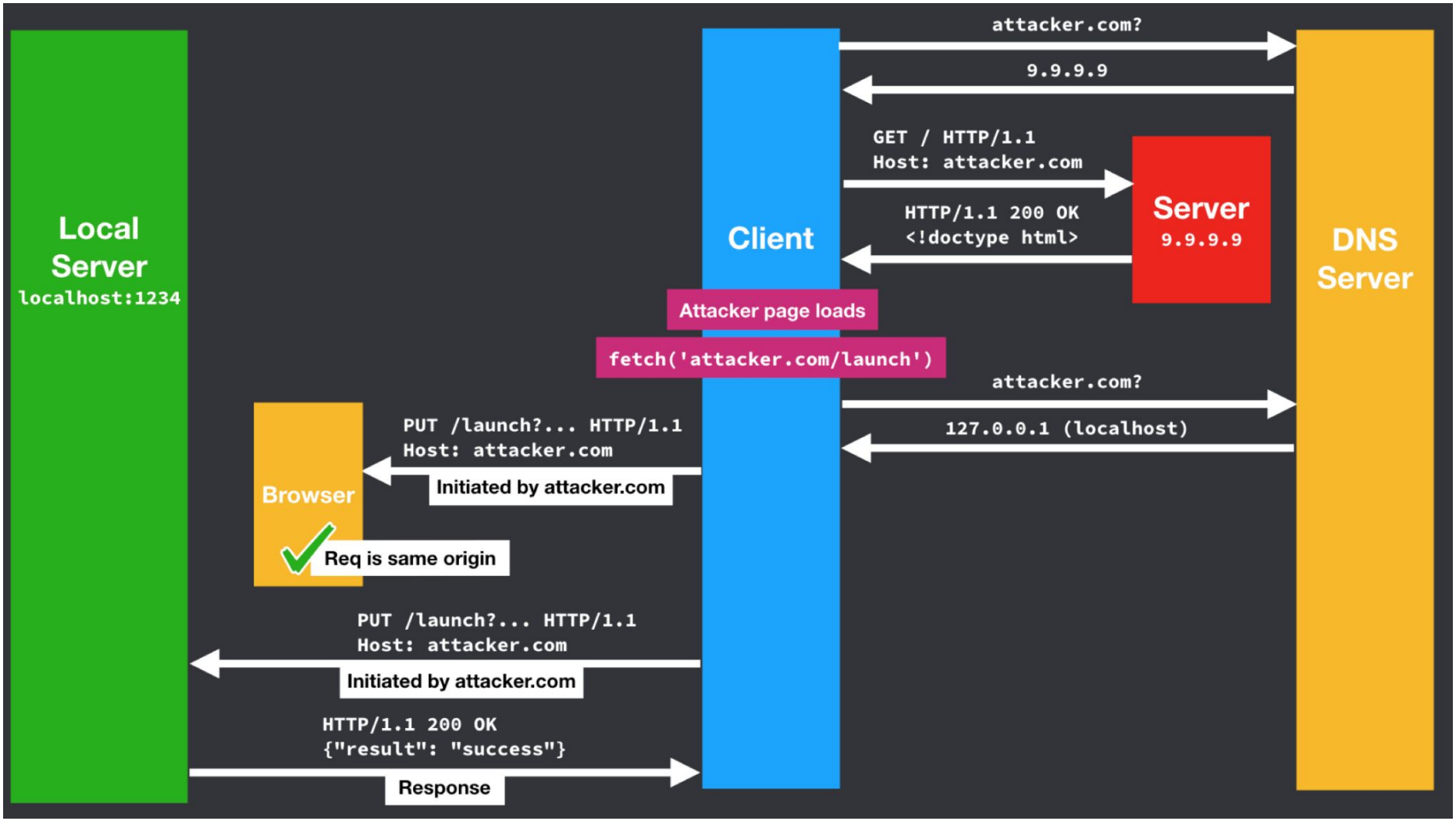

Local Server
localhost:1234
Client
DNS Server
attacker.com?
9.9.9.9
GET / HTTP/1.1
Host: attacker.com
Server
9.9.9.9
HTTP/1.1 200 OK
<!doctype html>
Attacker page loads
fetch('attacker.com/launch')
attacker.com?
127.0.0.1 (localhost)


Local Server
localhost:1234
Client
DNS Server
attacker.com?
9.9.9.9
GET / HTTP/1.1
Host: attacker.com
Server
9.9.9.9
HTTP/1.1 200 OK
<!doctype html>
Attacker page loads
fetch('attacker.com/launch')
attacker.com?
127.0.0.1 (localhost)
PUT /launch?... HTTP/1.1
Host: attacker.com
Initiated by attacker.com


Local Server
localhost:1234
Client
attacker.com?
9.9.9.9
GET / HTTP/1.1
Host: attacker.com
Server
9.9.9.9
HTTP/1.1 200 OK
<!doctype html>
DNS Server
Attacker page loads
fetch('attacker.com/launch')
attacker.com?
127.0.0.1 (localhost)
Browser
PUT /launch?... HTTP/1.1
Host: attacker.com
Initiated by attacker.com


Local Server
localhost:1234
Client
attacker.com?
9.9.9.9
GET / HTTP/1.1
Host: attacker.com
HTTP/1.1 200 OK
<!doctype html>
Server
9.9.9.9
DNS Server
Attacker page loads
fetch('attacker.com/launch')
attacker.com?
127.0.0.1 (localhost)
PUT /launch?... HTTP/1.1
Host: attacker.com
Initiated by attacker.com
Browser
Req is same origin


Local Server
localhost:1234
Client
DNS Server
attacker.com?
9.9.9.9
GET / HTTP/1.1
Host: attacker.com
Server
9.9.9.9
HTTP/1.1 200 OK
<!doctype html>
Attacker page loads
fetch('attacker.com/launch')
attacker.com?
127.0.0.1 (localhost)
Browser
PUT /launch?... HTTP/1.1
Host: attacker.com
Initiated by attacker.com
Req is same origin
PUT /launch?... HTTP/1.1
Host: attacker.com
Initiated by attacker.com

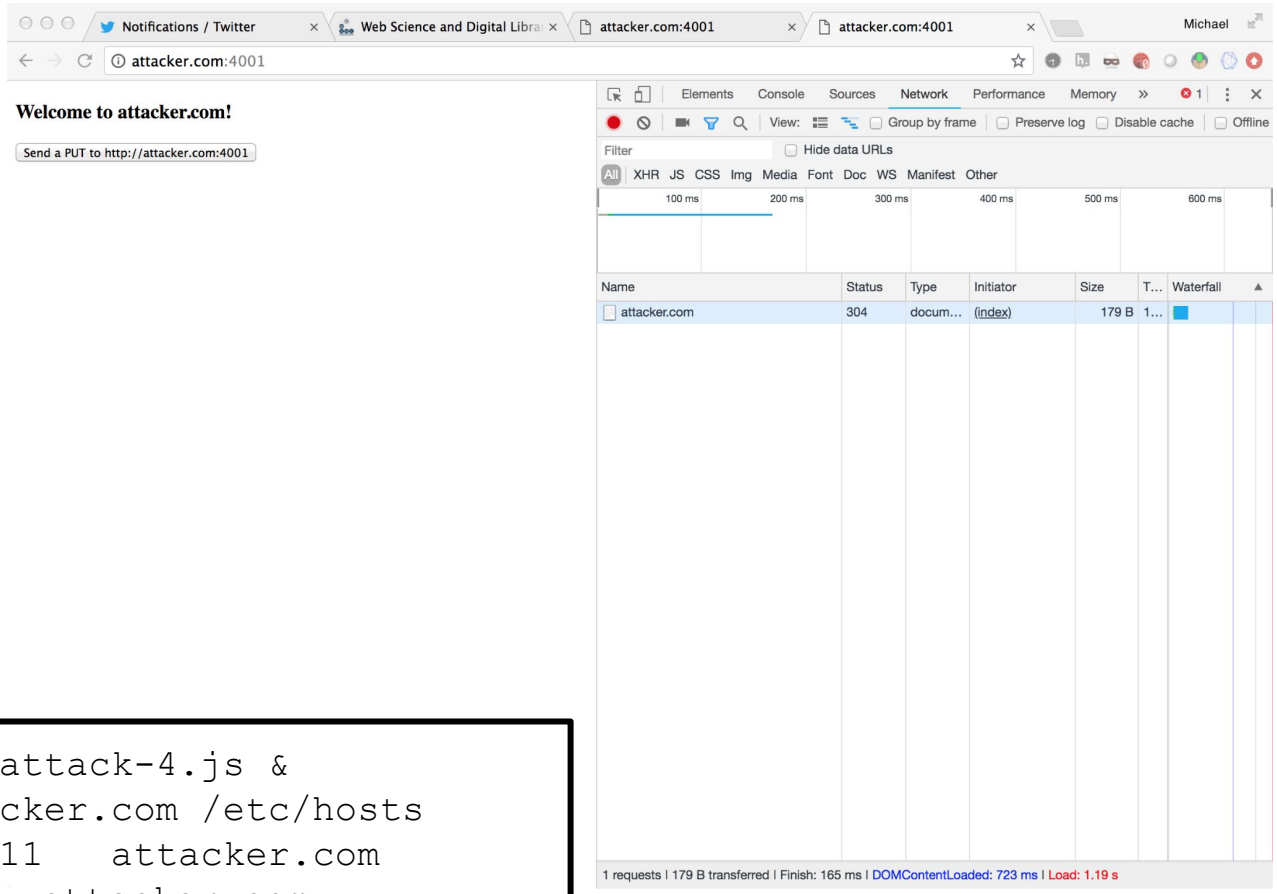

Local Server
localhost:1234
Client
DNS Server
attacker.com?
9.9.9.9
GET / HTTP/1.1
Host: attacker.com
Server
9.9.9.9
HTTP/1.1 200 OK
<!doctype html>
Attacker page loads
fetch('attacker.com/launch')
attacker.com?
127.0.0.1 (localhost)
Browser
PUT /launch?... HTTP/1.1
Host: attacker.com
Initiated by attacker.com
Req is same origin
PUT /launch?... HTTP/1.1
Host: attacker.com
Initiated by attacker.com
HTTP/1.1 200 OK
{"result": "success"}
Response

Preventing DNS rebinding attacks

demo

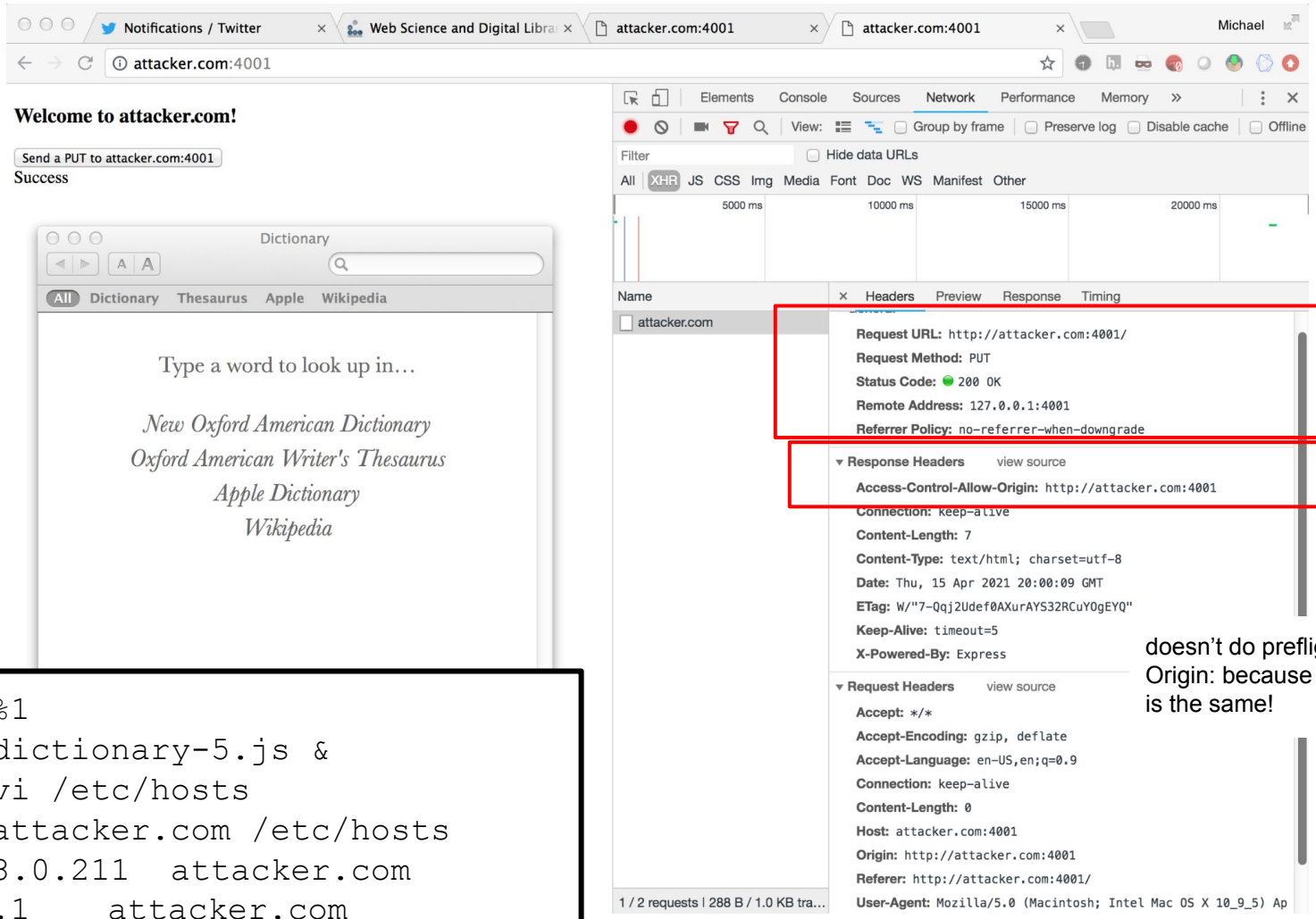
Works better with two different machines running servers on the same port. We're going to fake it by starting "node dns-attack-4.js", then killing it. Then we'll swap the DNS entries and start "node dictionary-5.js".

attacker.com is 192.168.0.211




```
$ node dns-attack-4.js &
$ grep attacker.com /etc/hosts
192.168.0.211   attacker.com
#127.0.0.1   attacker.com
```

attacker.com is 127.0.0.1




```
$ kill %1
$ node dictionary-5.js &
$ sudo vi /etc/hosts
$ grep attacker.com /etc/hosts
#192.168.0.211  attacker.com
127.0.0.1    attacker.com
```

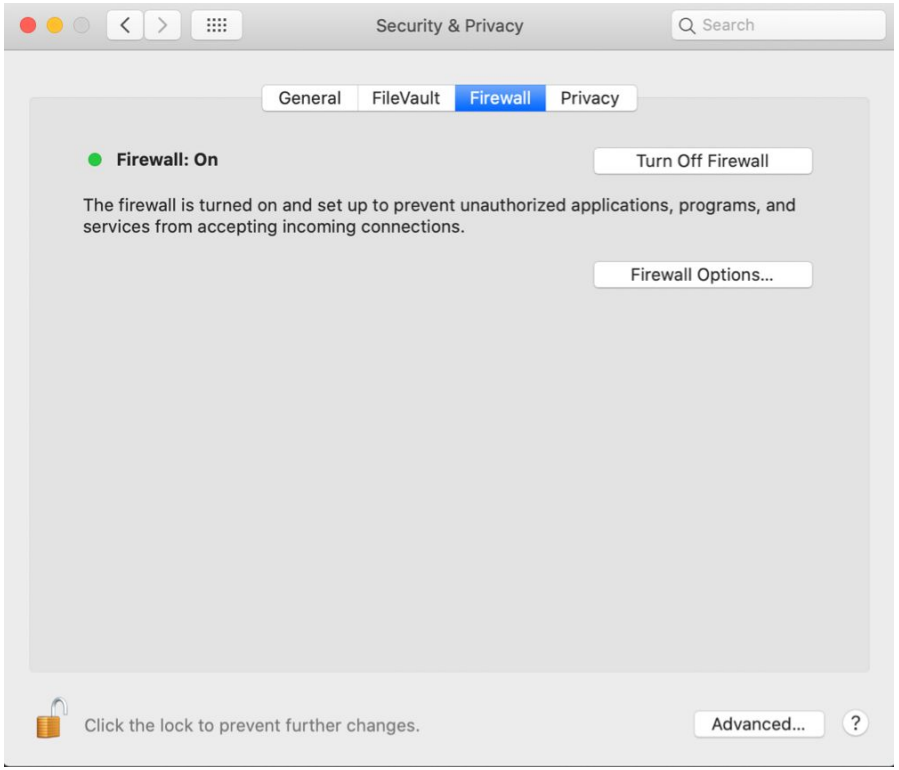
doesn't do preflight or send Origin: because the Origin is the same!

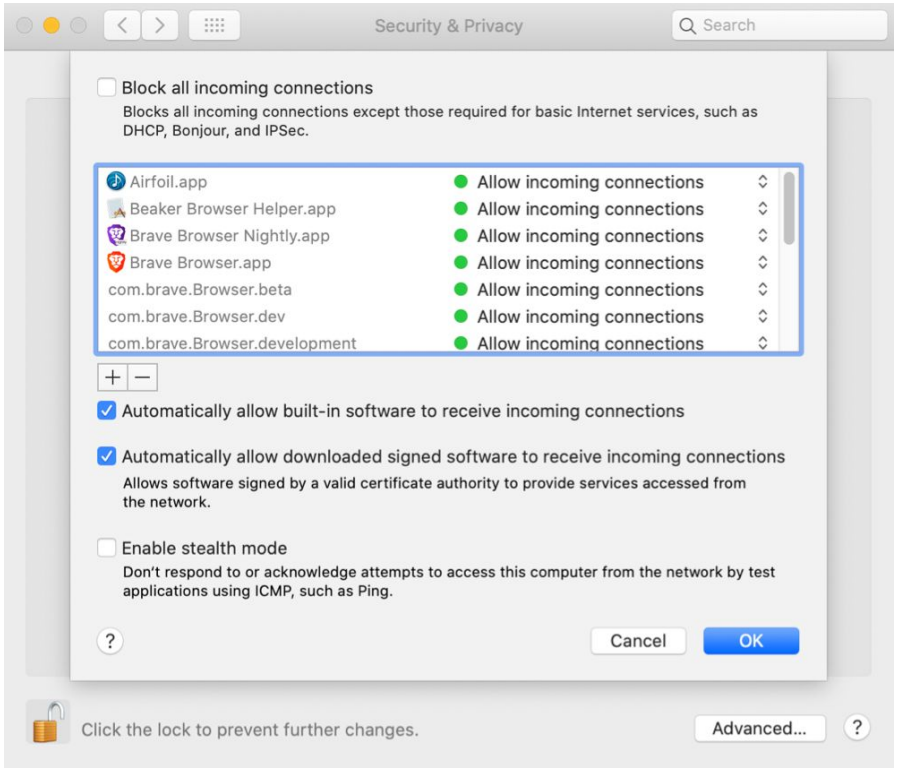
Prevent DNS rebinding

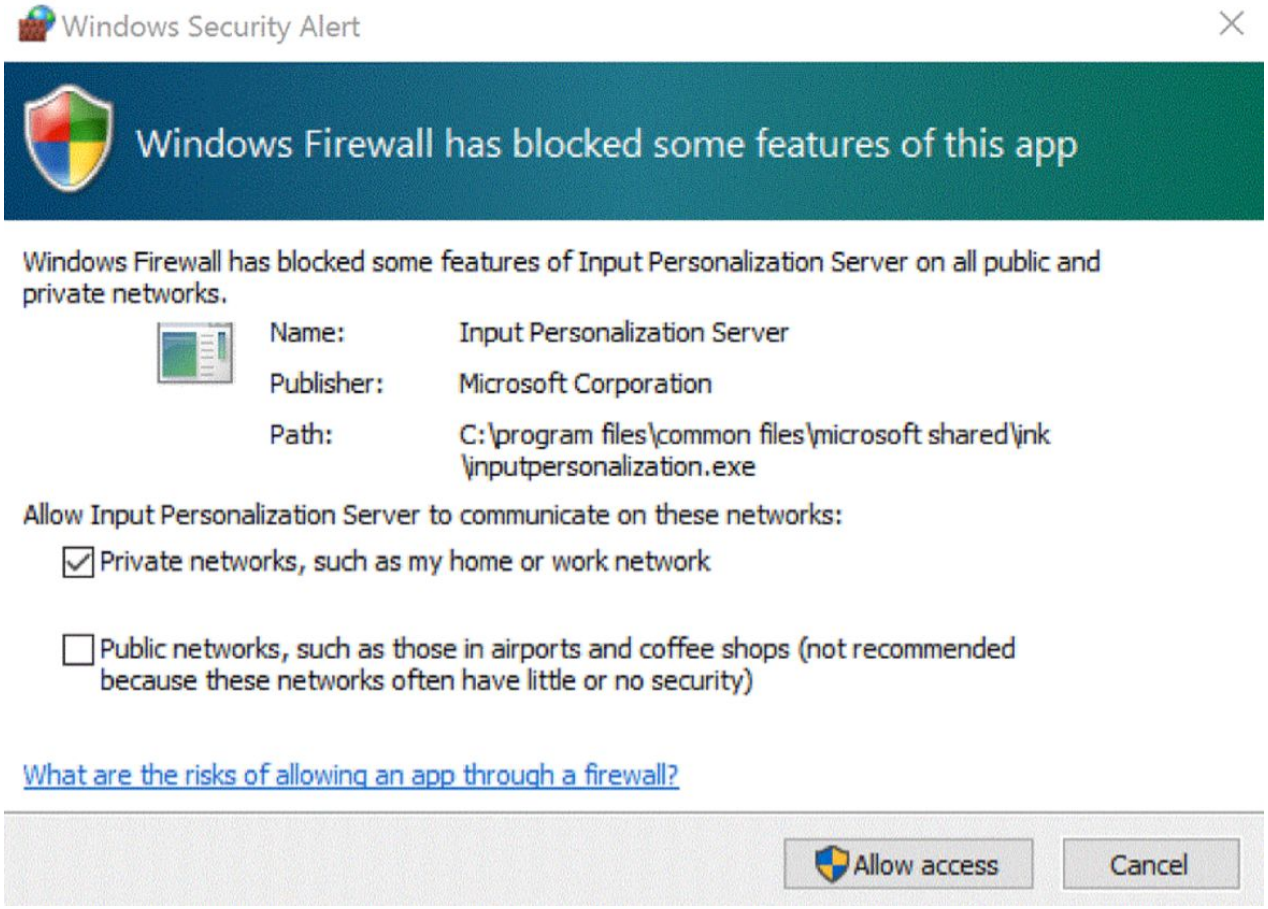
- **Idea**: DNS rebinding allows the attacker to trick the browser into thinking the local victim server is same origin to `attacker.com`
- **Solution**: Ensure that the `Host` header is not for a random origin, but instead for `localhost` or equivalent
- Remember: Cannot rely on `Origin` header
  - `Origin` header is not sent for same origin requests

Attack surface of local servers

- Usually bound to localhost with `server.listen(4000, '127.0.0.1')` so other devices on the network cannot connect
- When bound to all interfaces, incoming connections are usually still blocked by the operating system firewall
- Vulnerable to DNS rebinding unless proactive measures are taken (check the `Host` header!)


Security & Privacy
Search
General
FileVault
Firewall
Privacy
Firewall: On
Turn Off Firewall
The firewall is turned on and set up to prevent unauthorized applications, programs, and services from accepting incoming connections.
Firewall Options...
Click the lock to prevent further changes.
Advanced...
?


Security & Privacy
Search
Block all incoming connections
Blocks all incoming connections except those required for basic Internet services, such as DHCP, Bonjour, and IPSec.
Airfoil.app
Allow incoming connections
Beaker Browser Helper.app
Allow incoming connections
Brave Browser Nightly.app
Allow incoming connections
Brave Browser.app
Allow incoming connections
com.brave.Browser.beta
Allow incoming connections
com.brave.Browser.dev
Allow incoming connections
com.brave.Browser.development
Allow incoming connections
Automatically allow built-in software to receive incoming connections
Automatically allow downloaded signed software to receive incoming connections
Allows software signed by a valid certificate authority to provide services accessed from the network.
Enable stealth mode
Don't respond to or acknowledge attempts to access this computer from the network by test applications using ICMP, such as Ping.
?
Cancel
OK
Click the lock to prevent further changes.
Advanced...
?


Windows Security Alert
Windows Firewall has blocked some features of this app
Windows Firewall has blocked some features of Input Personalization Server on all public and private networks.
Name: Input Personalization Server
Publisher: Microsoft Corporation
Path: C:\program files\common files\microsoft shared\ink \inputpersonalization.exe
Allow Input Personalization Server to communicate on these networks:
Private networks, such as my home or work network
Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)
What are the risks of allowing an app through a firewall?
Allow access
Cancel

Attack surface of IoT devices

- Usually bound to all interfaces since they expect connections from other devices on the network
- Usually no operating system firewall or configured to allow incoming connections
- Vulnerable to DNS rebinding unless proactive measures are taken (check the `Host` header!)
- Hard or impossible to update after shipped to the customer, cheaply produced, price pressure, consumers don't care about IoT security

Home > Security

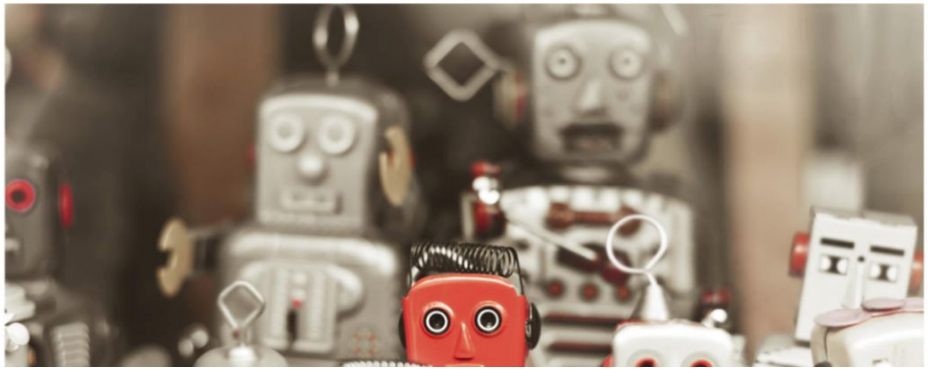
FEATURE

The Mirai botnet explained: How teen scammers and CCTV cameras almost brought down the internet

Mirai took advantage of insecure IoT devices in a simple but clever way. It scanned big blocks of the internet for open Telnet ports, then attempted to log in default passwords. In this way, it was able to amass a botnet army.

By Josh Fruhlinger

CSO | MAR 9, 2018 3:00 AM PST



https://www.csoonline.com/article/3258748/the-mirai-botnet-explained-how-teen-scammers-and-cctv-cameras-almost-brought- down-the-internet.html

IoT devices botnet (Mirai)

- Minecraft game server hosts launch DDoS attacks against their rivals, hoping to knock their servers offline and attract their business
- Many IoT devices use weak default passwords
- Mirai can launch both HTTP flood and network-level attacks
- Mirai is hard-wired to avoid IPs owned by GE, Hewlett-Packard, and the U.S. DoD
- Contains Russian-language strings which were a red herring

DNS rebinding attack: Blizzard games

- All Blizzard games install a tool called "Blizzard Update Agent"
- The tool starts an HTTP server which accepts commands to install, uninstall, change settings, update and other maintenance related options
- Used a scheme where the HTTP request sender has to prove they can read response data (which is not allowed cross-origin)
- Server was vulnerable to DNS rebinding allowing remote code execution (RCE) from JavaScript on any origin
- See: https://bugs.chromium.org/p/project-zero/issues/detail?id=1471

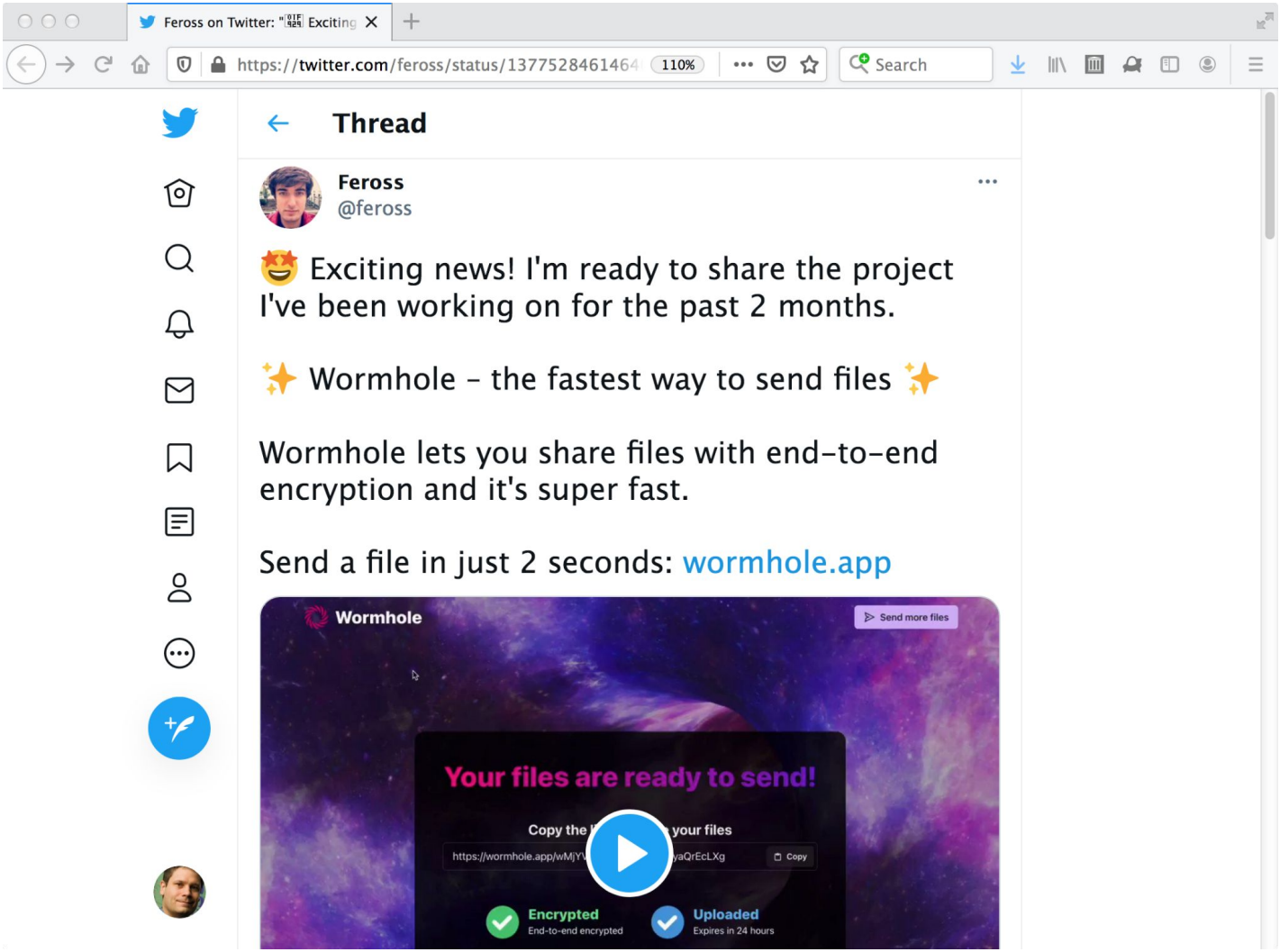
DNS rebinding attack: Transmission

- Transmission is a BitTorrent app, installed locally on the user's device
- Uses a client/server architecture where the UI is a client to the app logic which resides in the server
- Used a scheme where the HTTP request sender has to prove they can read response data (which is not allowed cross-origin)
- Server was vulnerable to DNS rebinding allowing remote code execution (RCE) from JavaScript on any origin
- See: https://bugs.chromium.org/p/project-zero/issues/detail?id=1447

DNS rebinding attack: WebTorrent

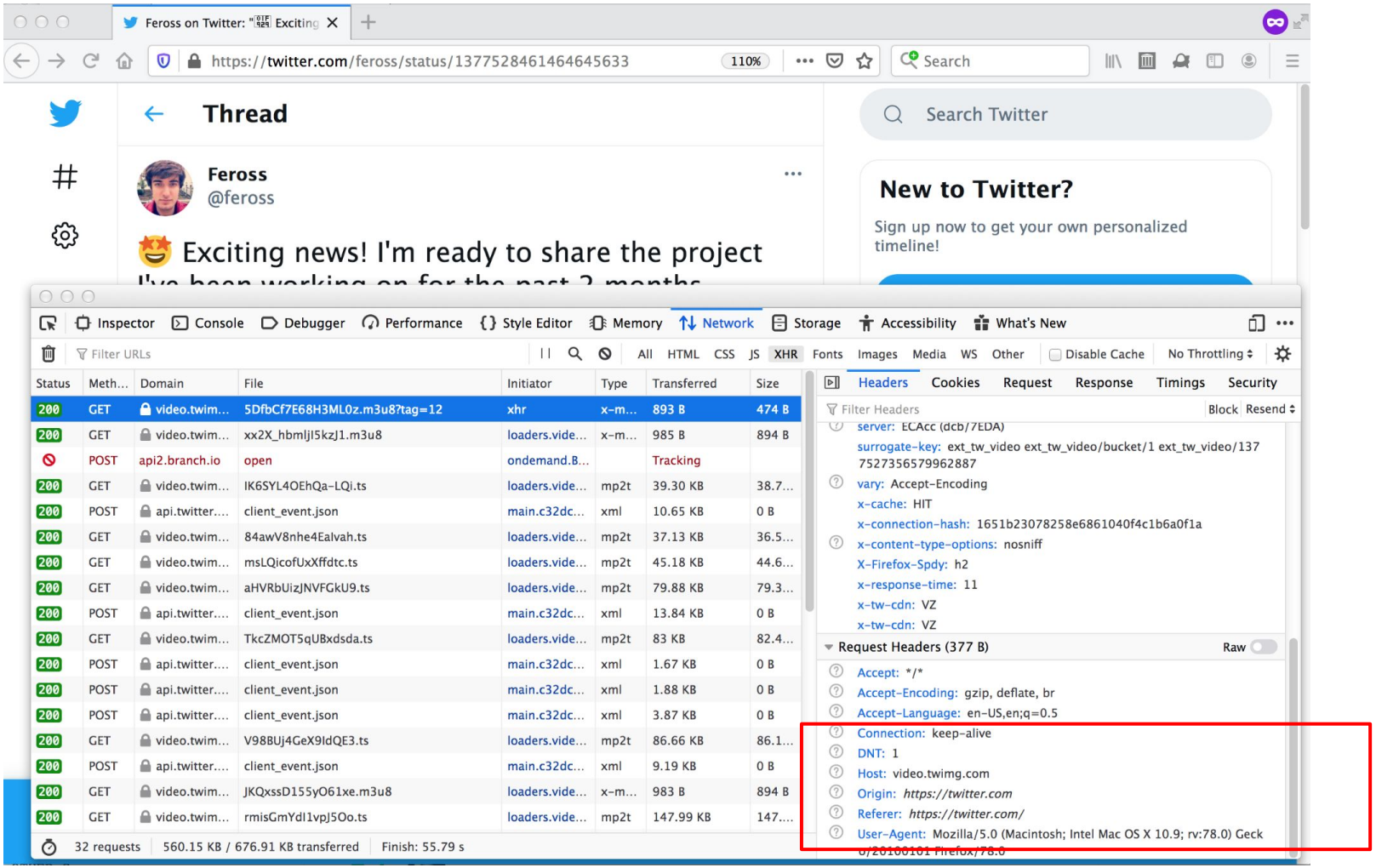
- WebTorrent has a feature that allows users to serve torrent contents via a local HTTP server
- Useful for streaming video to apps like VLC, allows dynamic piece prioritization
- Server was vulnerable to DNS rebinding allowing attacker to read torrent content (i.e. see the what torrent the user is viewing)
- See: https://github.com/webtorrent/webtorrent/commit/eea73a38ed8552c6a99cdd0dea5c9619dc955a21#diff-c945a46d13b34fcaff544d966cffcaba
- See: https://github.com/webtorrent/webtorrent/commit/30adf6a19b50b6e013c8ad9532c7e59d349df461#diff-c945a46d13b34fcaff544d966cffcaba

See also:



https://twitter.com/feross/status/1377528461464645633

Origin vs. Host



Why isn't DNS rebinding an effective attack against remote servers?

- attacker.com doesn't have the authority to change amazon.com (104.72.7.31; see below) to point to its site
- attacker.com could change its DNS so attacker.com points to amazon.com, but browser:
  - won’t send amazon.com cookies to attacker.com
  - thinks it’s talking to attacker.com but the certs will be for amazon.com, so TLS handshake will fail
- DNS rebinding works against localhost because most local HTTP servers (unwisely) trust anyone who can send them a connection

```
$ nslookup www.amazon.com
Server:      68.105.28.11
Address:     68.105.28.11#53

Non-authoritative answer:
www.amazon.com  canonical name = tp.47cf2c8c9-frontier.amazon.com.
tp.47cf2c8c9-frontier.amazon.com        canonical name = www.amazon.com.edgekey.net.
www.amazon.com.edgekey.net       canonical name = e15316.e22.akamaiedge.net.
Name:   e15316.e22.akamaiedge.net
Address: 104.72.7.31
```

Final thoughts

- Don't ship a local HTTP server with your software
- If you do, it better be for a really good reason
- If you do, you better understand DNS rebinding attacks
- Check the value of the `Host` header!