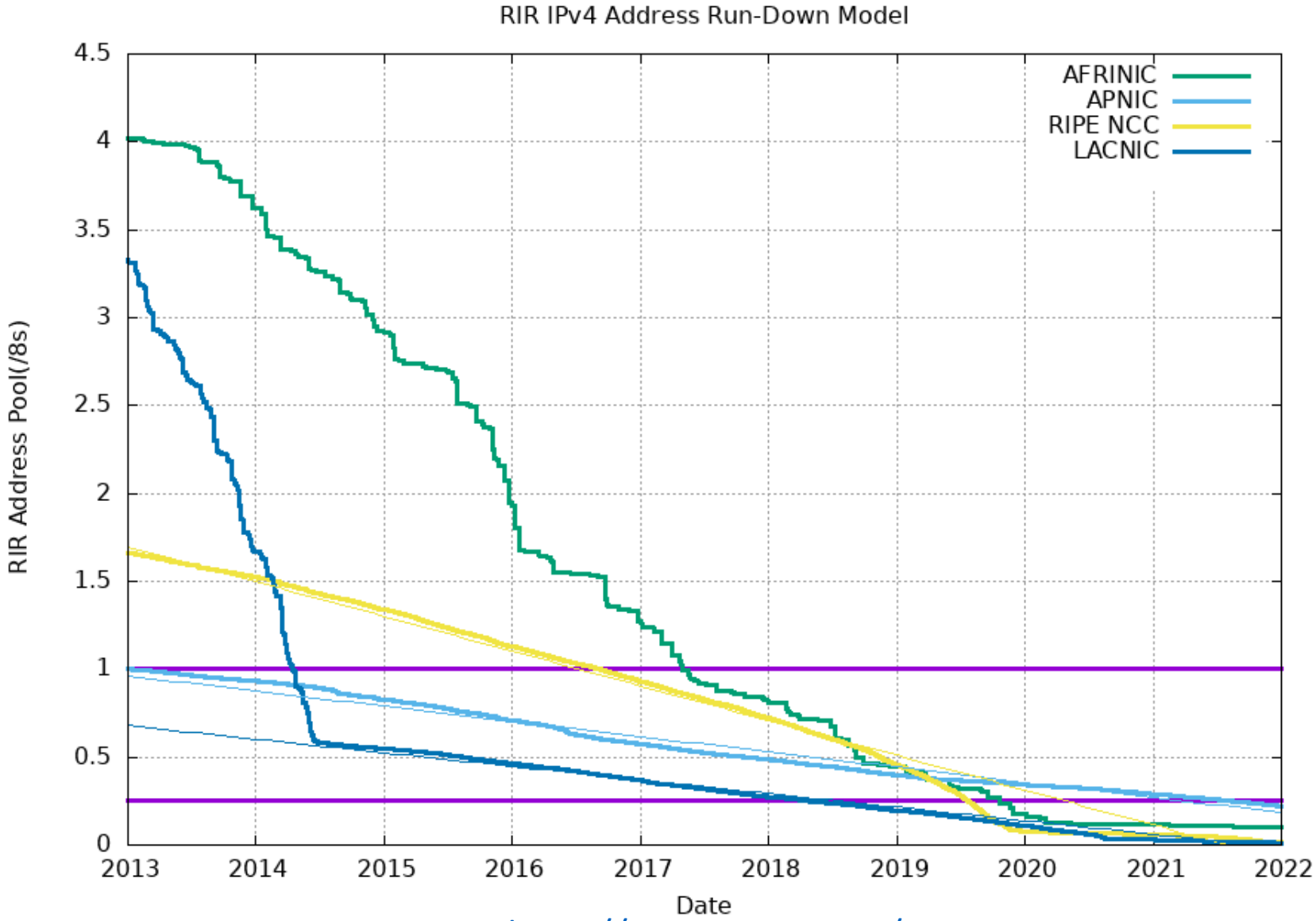

# Linux Administration (NSWI106)

# IPv6

Ing. Radek Zajíc, radek@zajic.v.pytli.cz • Jan 6 2025

# What’s IPv6 and why do we need it?

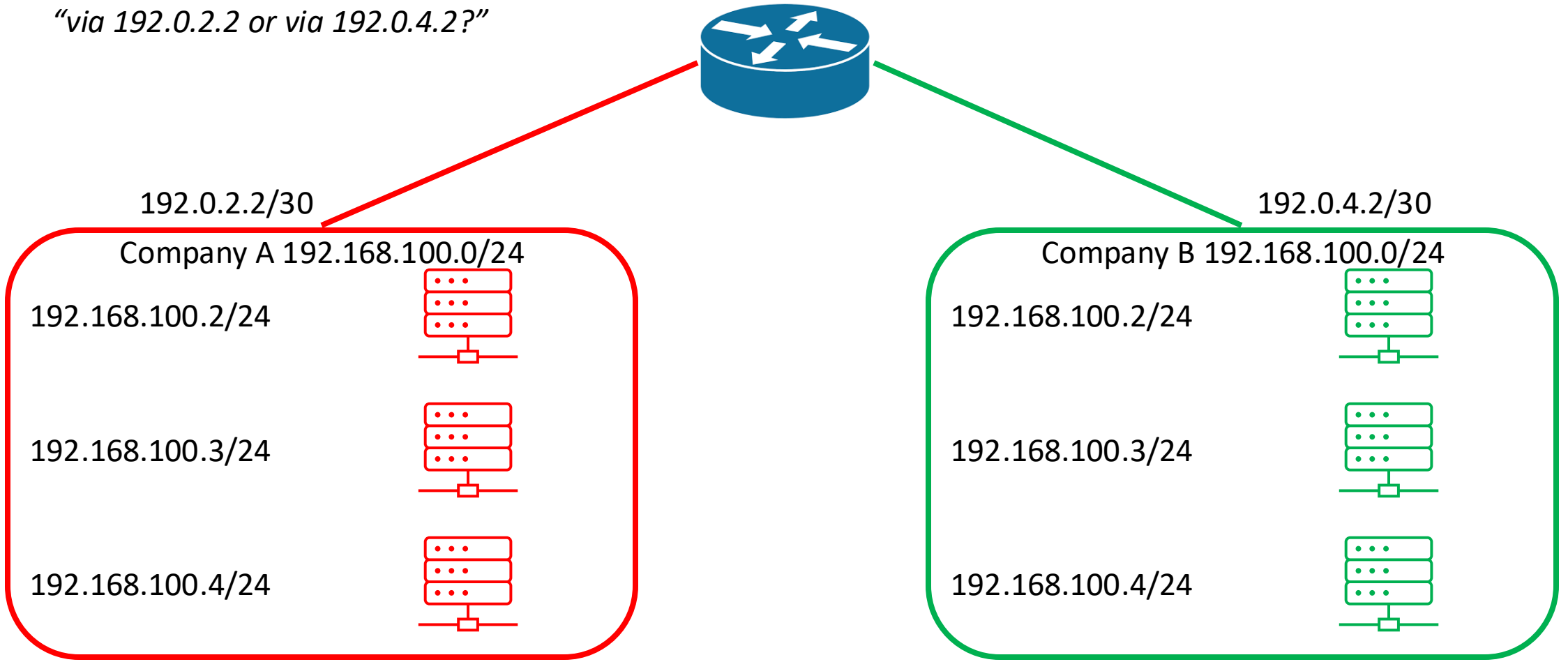
# Public IPv4 Exhaustion




https://ipv4.potaroo.net/

# Private IPv4 Exhaustion

*"Which route to pick to 192.168.100.0/24...?"*
*"via 192.0.2.2 or via 192.0.4.2?"*

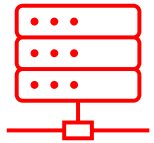


# Private IPv4 Exhaustion

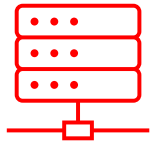
Which server will be available on port 80/tcp?
Which server will be available on port 443/tcp?
Which one will manage to obtain the certificate?
Why is this a problem for the Let‘s Encrypt certificate authority?

9.9.9.9

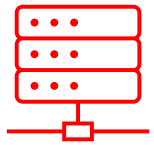
192.168.100.1/24

192.168.100.2/24 

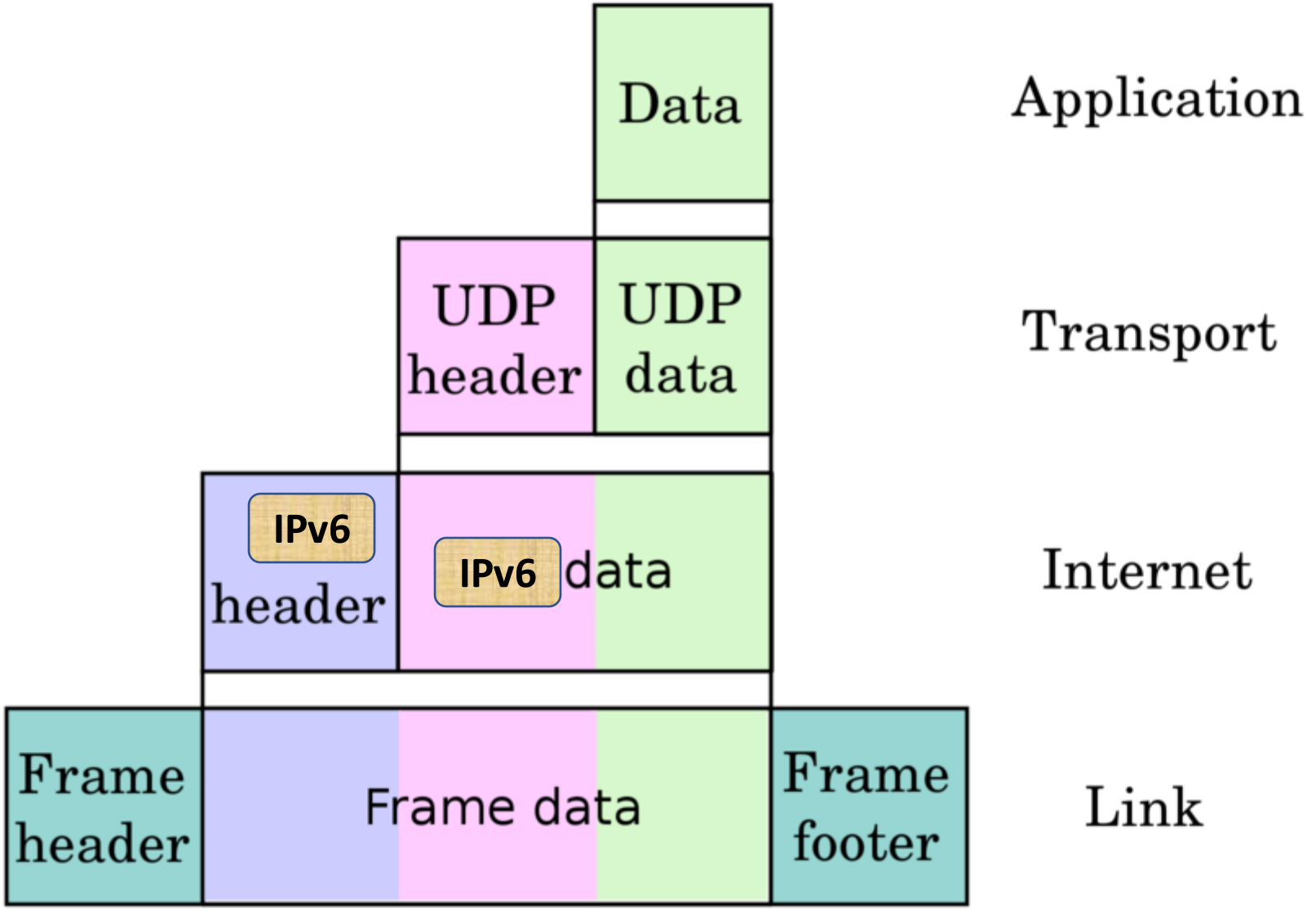
I want to be a webserver on 80/tcp and get a TLS certificate for bestdeals.com from Let‘s Encrypt!

192.168.100.3/24 

No, I want to be a webserver on 80/tcp and 443/tcp and get a TLS certificate for stealfromyou.com from Let‘s Encrypt!

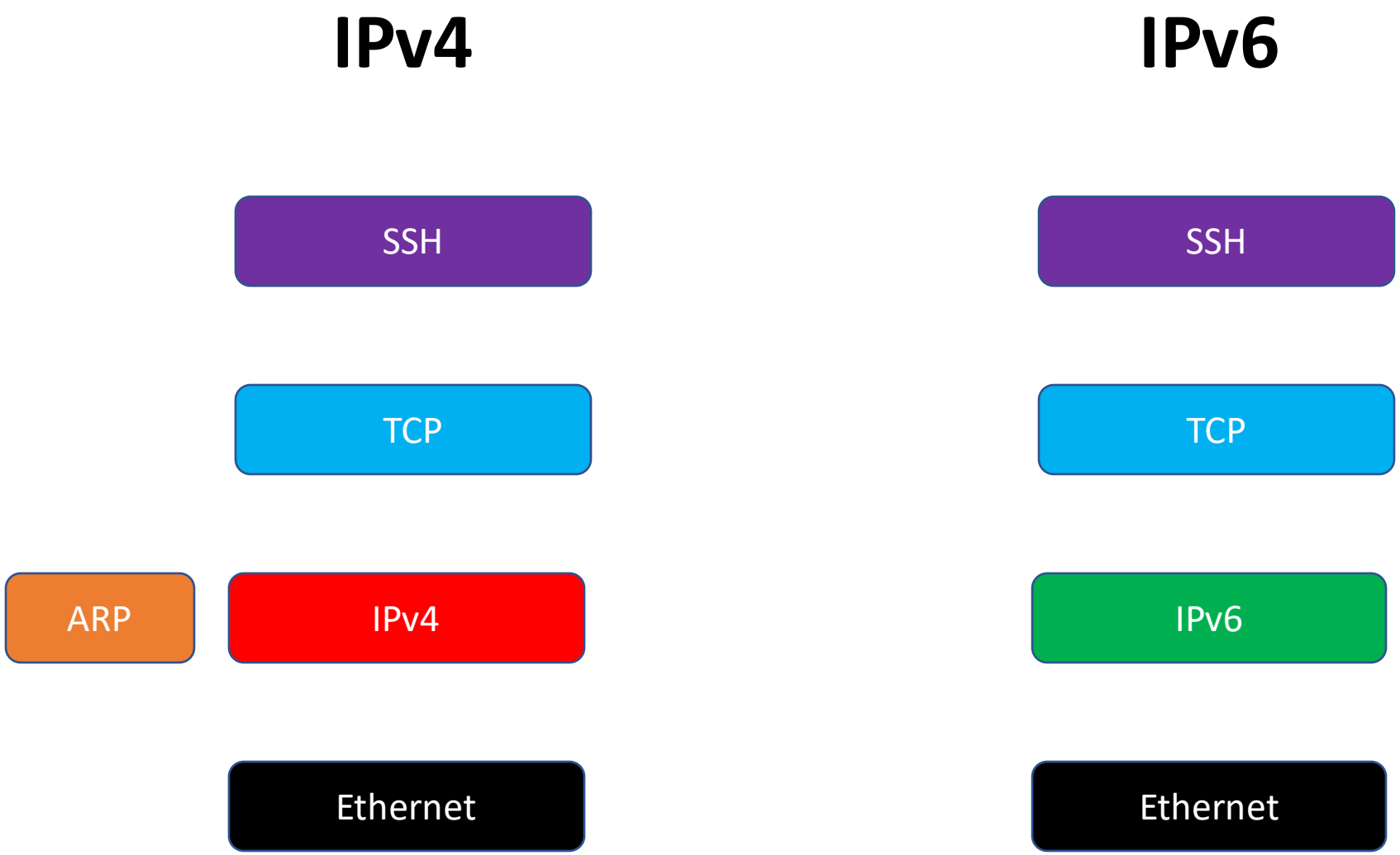
192.168.100.4/24 

Silence, you losers! I want all the ports on 9.9.9.9 just for myself, including 80/tcp and 443/tcp. I already have a certificate, don‘t care much about Let‘s Encrypt.




https://d3s.mff.cuni.cz/teaching/nswi106/lectures/04/

# Living next to each other



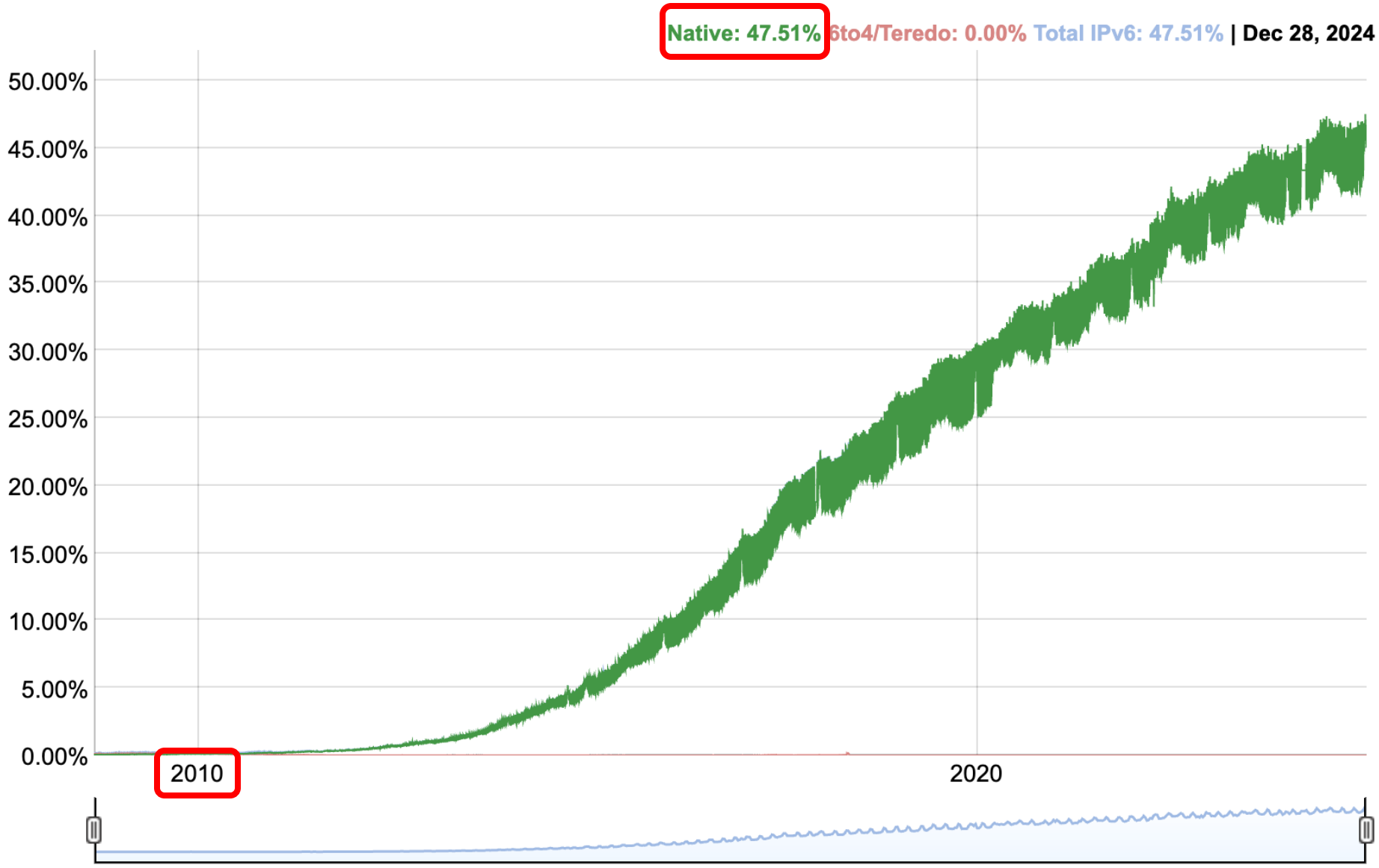
# IPv4 Exhaustion vs. IPv6

Addresses depleted,
market price **$50/address, $12,800 per /24**

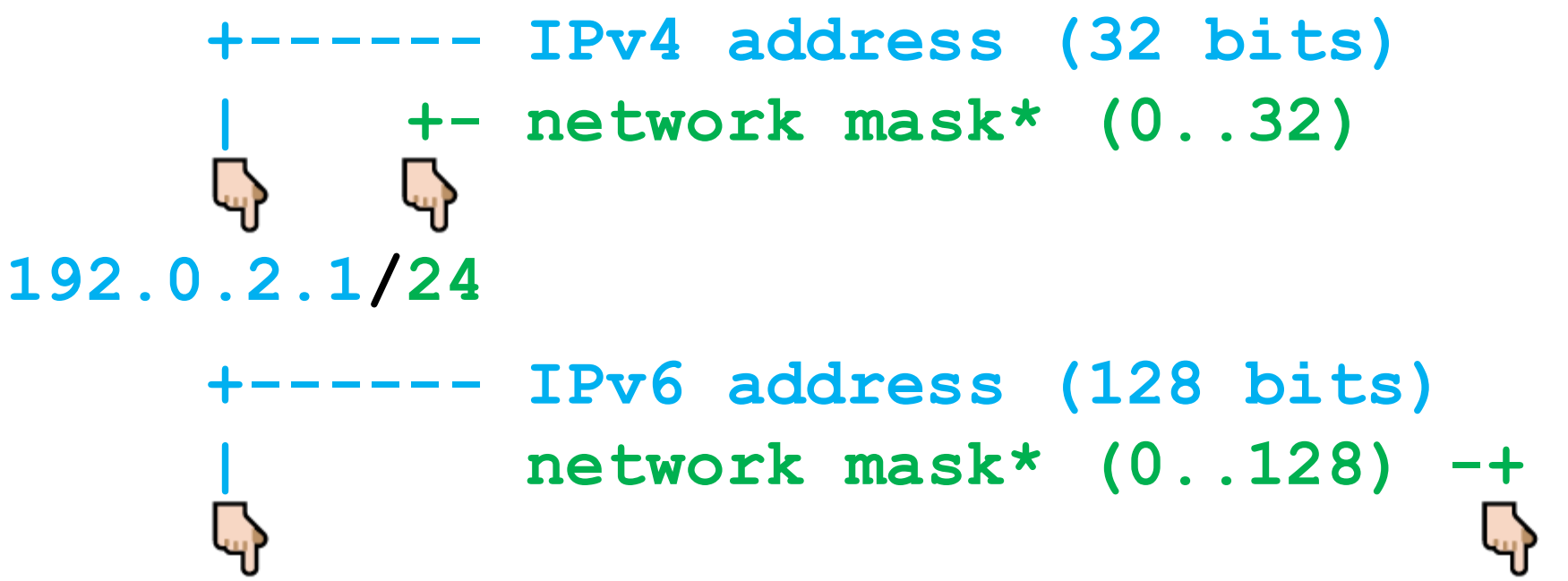
vs.

Almost **infinite** reserve of addresses,
$0/address

# State of IPv6 in 2025



# Address formats




*** network mask is always in decimal**

# Address compression and IPv6 canonical form

2001:0000:0000:0000:0caf:0000:0000:0001

2001:0:0:0:caf:0:0:1

2001::caf:0:0:1

2001:0:0:0:caf::1

~~2001::caf::1~~

0:0:0:0:0:0:0:0

::<-- "any", like 0.0.0.0

RFC5952, A Recommendation for IPv6 Address Text Representation

# Addresses of different uses

**Global Unicast Addresses (2000::/3)**

2001:db8:f001:a200:346b:decf:d923:4a1f/64

**Link-Scoped Unicast Addresses(fe80::/10)**

**(Link-Local Unicast, fe80::/64)**

fe80::78b4:8fff:fefd:8843/64

**Multicast Addresses (ff00::/8)**

ff02::2

**Unique Local Addresses (fc00::/7)**

fdf1:12d0:d3e6:0:72a7:41ff:feae:8b81/64

**RFC4291, IP Version 6 Addressing Architecture**

**RFC4193, Unique Local IPv6 Unicast Addresses**

# Unique Local Addresses

**Unique Local Addresses (fc00::/7)**

**fdf1:12d0:d3e6:0:72a7:41ff:feae:8b81/64**

**Like private IPv4 addresses...**

(e.g. 192.168.1.1/24, RFC1918)

**...but not really**

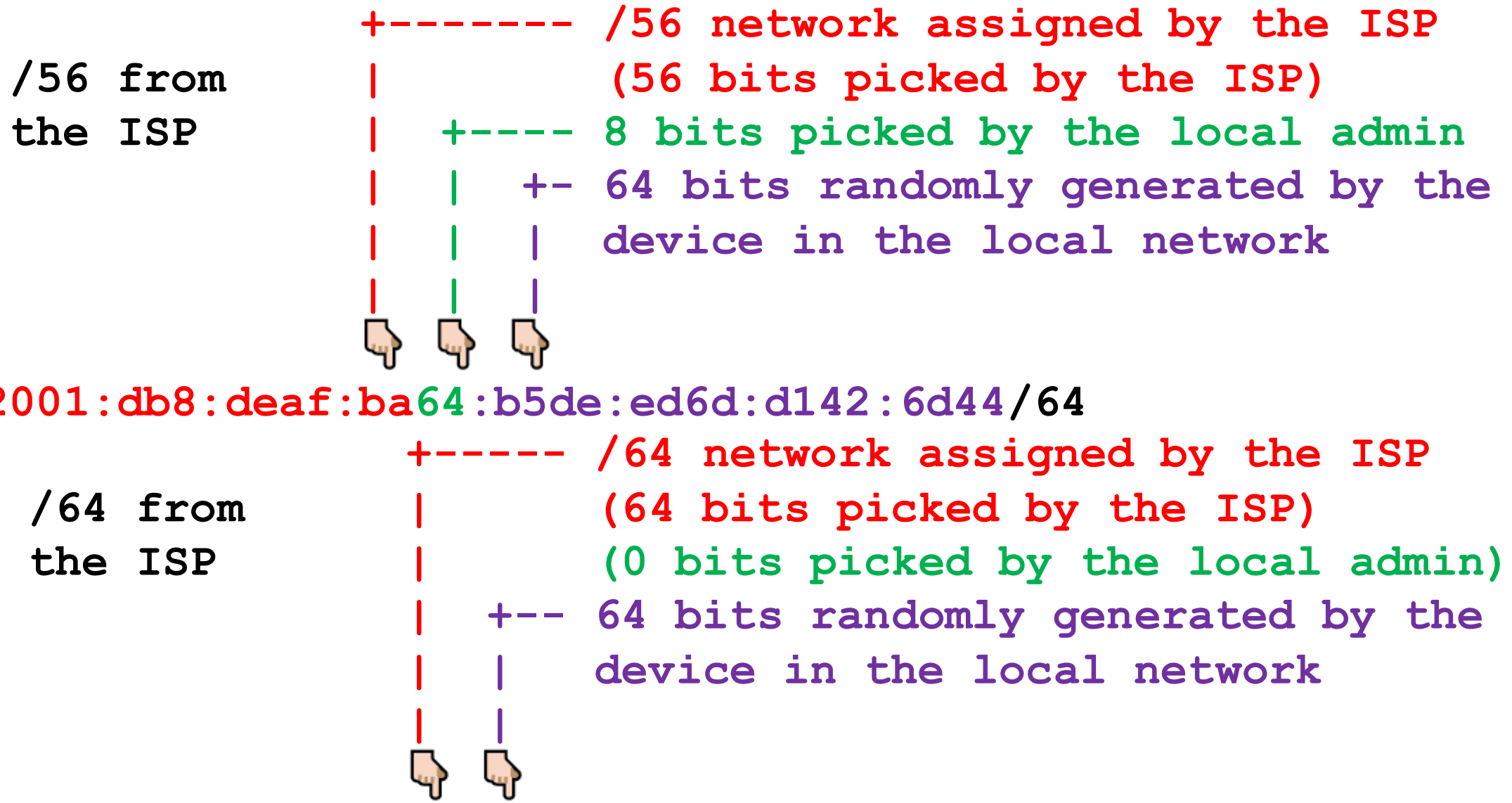
**Unintended Operational Issues With ULA**

# IPv6 subnetting

# Network prefix sizes and masks

```
/3   ... Globally routed IPv6 prefixes are allocated
         from 2000::/3 (up to 512 RIR assignments of /12)
/12  ... Block assigned from IANA to a RIR
         (2^17 = 131072 mid-sized ISP networks of /29)
/29  ... Block assigned by a RIR (RIPE NCC) to a mid-sized
         ISP (2^27 = 134217728 home user /56 networks)
/48  ... Block assigned by an ISP to a company
         (2^16 = 65536 SLAAC-enabled /64 networks)
/56  ... Block assigned by an ISP to a home user
         (2^8 = 256 SLAAC-enabled /64 networks)
/64  ... Stateless autoconfiguration (SLAAC) network size
         (up to 2^64 hosts in a network segment)
/80  ... Routed prefix to a single EC2 instance in AWS
         (2^16 /96 subnets, up to 2^32 containers/subnet)
```

# Address bits: ISP, local and autoconfigured



# Count prefixes, not addresses

**If you need many users' networks at home or in the office...**

**...count how many /64s do you have available**

**If you need multiple /96 networks for containers...**

**...count how many of those can you route to your servers**

**If you are building a datacenter...**

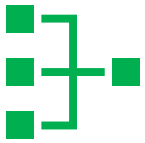
**...count how many prefixes of which sizes you need, then split the network into prefixes accordingly**

**If you are about to request IPv6 from an upstream ISP...**

**...ask for at least routed /56 for home and /48 for office**

# Where to get prefixes for your use case?

# Getting prefixes: DHCPv6 PD



(3) RA: IPv6 prefix on this network is 2001:db8:f001::/**64**, autonomous flag is on, generate addresses yourself.

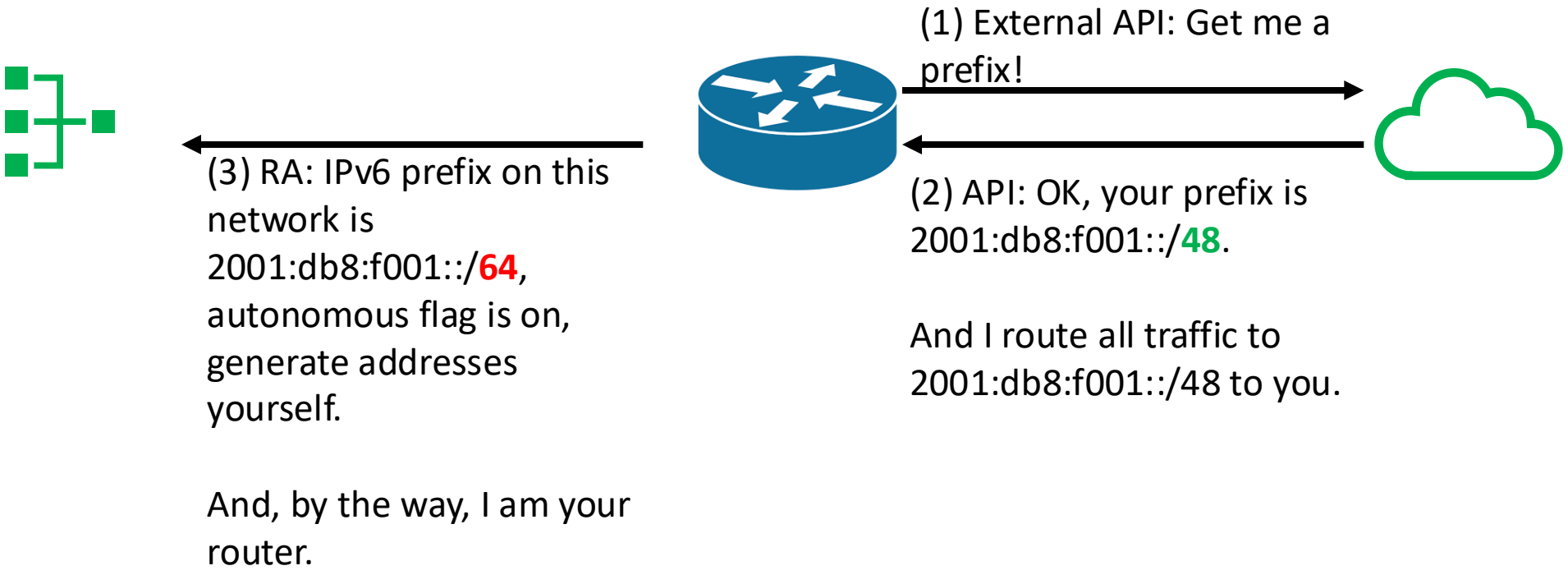
And, by the way, I am your router.

(1) DHCPv6: Get me a prefix!

(2) DHCPv6: OK, your prefix is 2001:db8:f001::/**48**.

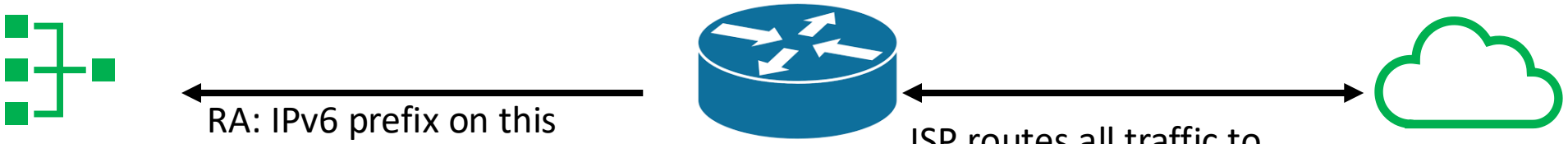
And, by the way, I route all traffic to 2001:db8:f001::/48 to you.

# Getting prefixes: out-of-band (OOB)



# Getting prefixes: static routing config

Customer’s IPv6 prefix is 2001:db8:f001::/48
Customer’s router’s WAN IPv6: 2001:db8:e:6::2/64
ISP router’s IPv6: 2001:db8:e:6::1/64



RA: IPv6 prefix on this network is 2001:db8:f001::/**64**, autonomous flag is on, generate addresses yourself.

And, by the way, I am your router.

ISP routes all traffic to 2001:db8:f001::/**48** via Client's IPv6 address, 2001:db8:e:6::2/64.

And by the way, ISP router's IPv6, 2001:db8:e:6::1, is your gateway to the Internet.

# IPv6 ICMPv6, RA, SLAAC, and DHCPv6

# IPv4 vs. IPv6: Discovery of nodes on a local network

**IPv4, ARP**

a2:ce:99:xx:xx:xx **>** **ff:ff:ff:ff:ff:ff**, **ARP: Request who-has 192.168.1.20 tell 192.168.1.1**

da:ae:67:yy:yy:yy **>** a2:ce:99:xx:xx:xx, **ARP: Reply 192.168.1.20 is-at da:ae:67:yy:yy:yy**

**IPv6, Neighbor solicitation, DAD**

**da:ae:67:yy:yy:yy > 33:33:ff:ad:ca:fe, IPv6: :: > ff02::1:ffad:cafe:**

ICMP6, neighbor solicitation, **who has 2001:db8:1337::bad:cafe**

**IPv6, Neighbor solicitation**

a2:ce:99:yy:yy:yy > **33:33:ff:ad:ca:fe, IPv6:** 2001:db8:1337::3 > **ff02::1:ffad:cafe:** ICMP6, neighbor solicitation, **who has 2001:db8:1337::bad:cafe**

**da:ae:67:xx:xx:xx** > a2:ce:99:yy:yy:yy, IPv6: **2001:db8:1337::bad:cafe** > 2001:db8:1337::3: ICMP6, neighbor advertisement, **tgt is 2001:db8:1337::bad:cafe**

# IPv6 Stateless Address Auto-Configuration (SLAAC)

```
IPv6, Router advertisements from a2:ce:99:yy:yy:yy (the router)
a2:ce:99:yy:yy:yy > 33:33:00:00:00:01, IPv6: fe80::a0ce:99ff:feyy:yyyy >
ff02::1: ICMP6, router advertisement

Stateful address conf.     :          No <-- do not use stateful DHCPv6
Stateful other conf.       :          No <-- do not use stateless DHCPv6
Prefix                     : 2001:db8:dead:beef::/64
  On-link                  :          Yes <-- talk to other nodes directly
  Autonomous address conf.:           Yes <-- perform SLAAC
Recursive DNS server       : 2001:4860:4860::8888 <-- use this DNS server
 Source link-layer address: A2:CE:99:YY:YY:YY
 from fe80::a0ce:99ff:feyy:yyyy <-- I'm your router
NETWORK_MASK=64 <-- always /64 with SLAAC
HOST_BITS=(random 1..2^64-1 or EUI-64)
HOST_BITS=::c87a:c4ff:fed9:79ec
IPV6_ADDRESS=<Prefix> + <HOST_BITS>/<NETWORK_MASK>

Address: 2001:db8:dead:beef:c87a:c4ff:fed9:79ec/64
Gateway: fe80::a0ce:99ff:feyy:yyyy
DNS:     2001:4860:4860::8888
```

# IPv6 Stateless Address Auto-Configuration (SLAAC)

```
root@randomvm:~# ip -6 a s dev enp1s0
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 state UNKNOWN qlen 1000
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    inet6 2001:db8:dead:babe:5054:ff:fe5f:7be0/64 scope global dynamic
mngtmpaddr noprefixroute
       valid_lft 290sec preferred_lft 110sec
    inet6 fe80::5054:ff:fe5f:7be0/64 scope link
       valid_lft forever preferred_lft forever

root@randomvm:~# ip -6 route
::1 dev lo proto kernel metric 256 pref medium
2001:db8:dead:babe::/64 dev enp1s0 proto ra metric 100 expires 281sec pref
medium
fe80::/64 dev enp1s0 proto kernel metric 256 pref medium
default via fe80::a0ce:99ff:feyy:yyyy dev enp1s0 proto ra metric 100 expires
1781sec mtu 1492 pref medium
```

# IPv6 DHCPv6

```
Client (fe80::...), port 546/udp -> ff02::1:2, port 547/udp: DHCPv6 Solicit
Server (fe80::...), port 547/udp -> Client (fe80::...), port 546/udp: DHCPv6 Advertise
Client (fe80::...), port 546/udp -> ff02::1:2, port 547/udp: DHCPv6 Request
Server (fe80::...), port 547/udp -> Client (fe80::...), port 546/udp: DHCPv6 Reply
```

**DHCPv6 does not primarily use MAC address for client identification, instead it uses DUID (machine ID) and IAID (interface ID). Beware when cloning machines, always change the IDs!**

**DHCPv6 always configures IPv6 addresses with netmask /128**

**DHCPv6 does not configure routing (default gateway, on-link routes, additional routes) – only router advertisements can do this**

# IPv6 DHCPv6

```
root@ip-172-31-10-26:~# ip -6 a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 state UNKNOWN qlen 1000
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 state UP qlen 1000
    inet6 2a05:d014:879:35c0:af2:aece:403f:b748/128 scope global dynamic
noprefixroute
       valid_lft 383sec preferred_lft 73sec
    inet6 fe80::8e4:6fff:fec0:2e82/64 scope link
       valid_lft forever preferred_lft forever

root@ip-172-31-10-26:~# ip -6 route
::1 dev lo proto kernel metric 256 pref medium
2a05:d014:879:35c0::/64 dev ens5 proto ra metric 100 pref medium
fe80::/64 dev ens5 proto kernel metric 256 pref medium
default via fe80::881:60ff:fe0d:88c0 dev ens5 proto ra metric 100 expires
1799sec pref medium
```

# IPv6 DHCPv6 – related RA

```
root@ip-172-31-10-26:~# rdisc6 -1 ens5
Stateful address conf.     :          Yes <-- use DHCPv6
Stateful other conf.       :           No
Prefix                     : 2a05:d014:879:35c0::/64
  On-link                  :          Yes
  Autonomous address conf.:           No <-- don't use SLAAC to build own address
  Valid time               :     infinite (0xffffffff)
  Pref. time               :     infinite (0xffffffff)
 from fe80::881:60ff:fe0d:88c0

root@ip-172-31-10-26:~# ip -6 route
default via fe80::881:60ff:fe0d:88c0 dev ens5 proto ra metric 100 expires 1799sec
pref medium
```

# NAT and IPv6

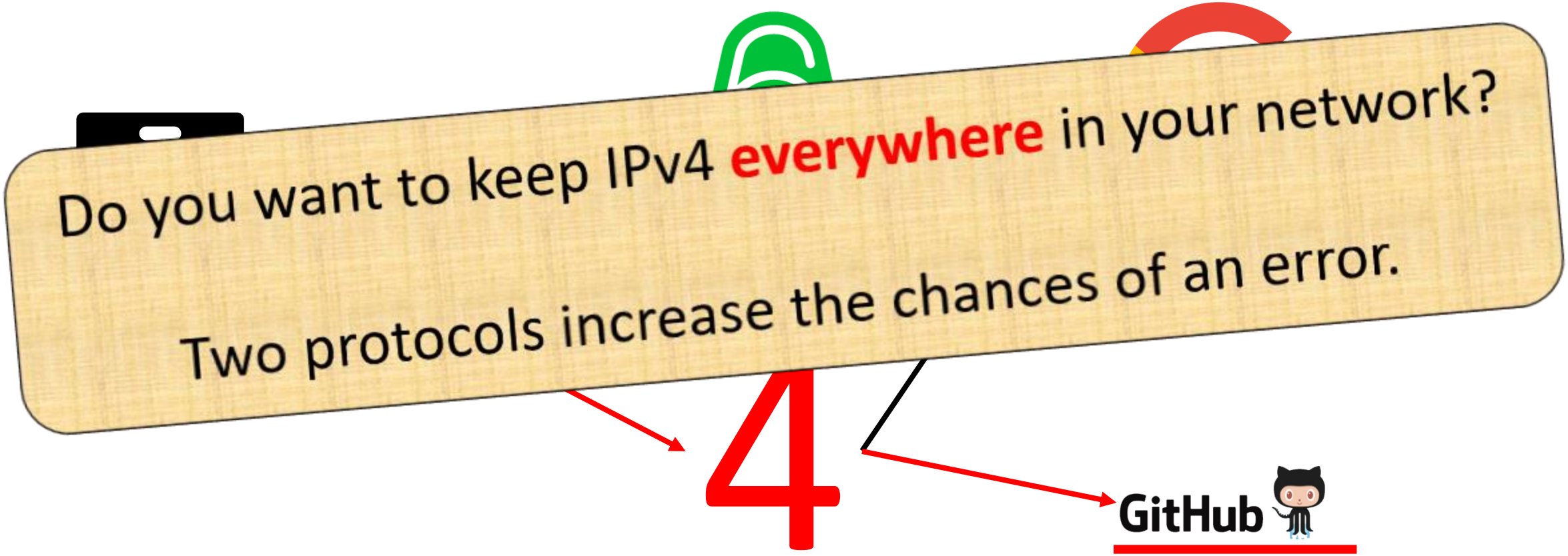
You usually don‘t need NAT from IPv6 to IPv6 (NAT66), because you have plenty of addresses and networks available.

(and, no, NAT is not a firewall!)

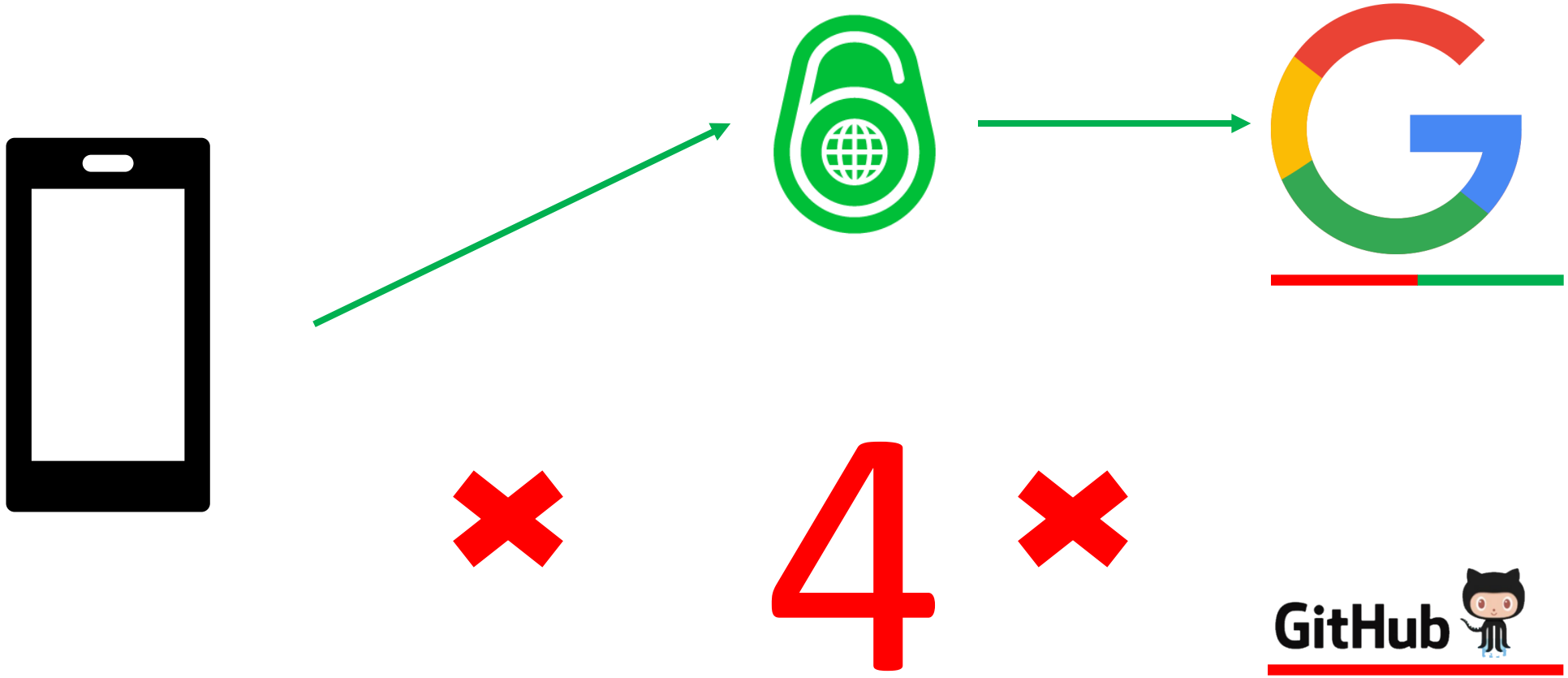
(there are niche use cases where you *may* need NAT66 and in that case, it‘s there in the latest Linux kernels, and works quite like the IPv4 to IPv4 NAT; in addition to that, there‘s also NPT66, which translates whole prefixes)

# IPv6 deployment modes

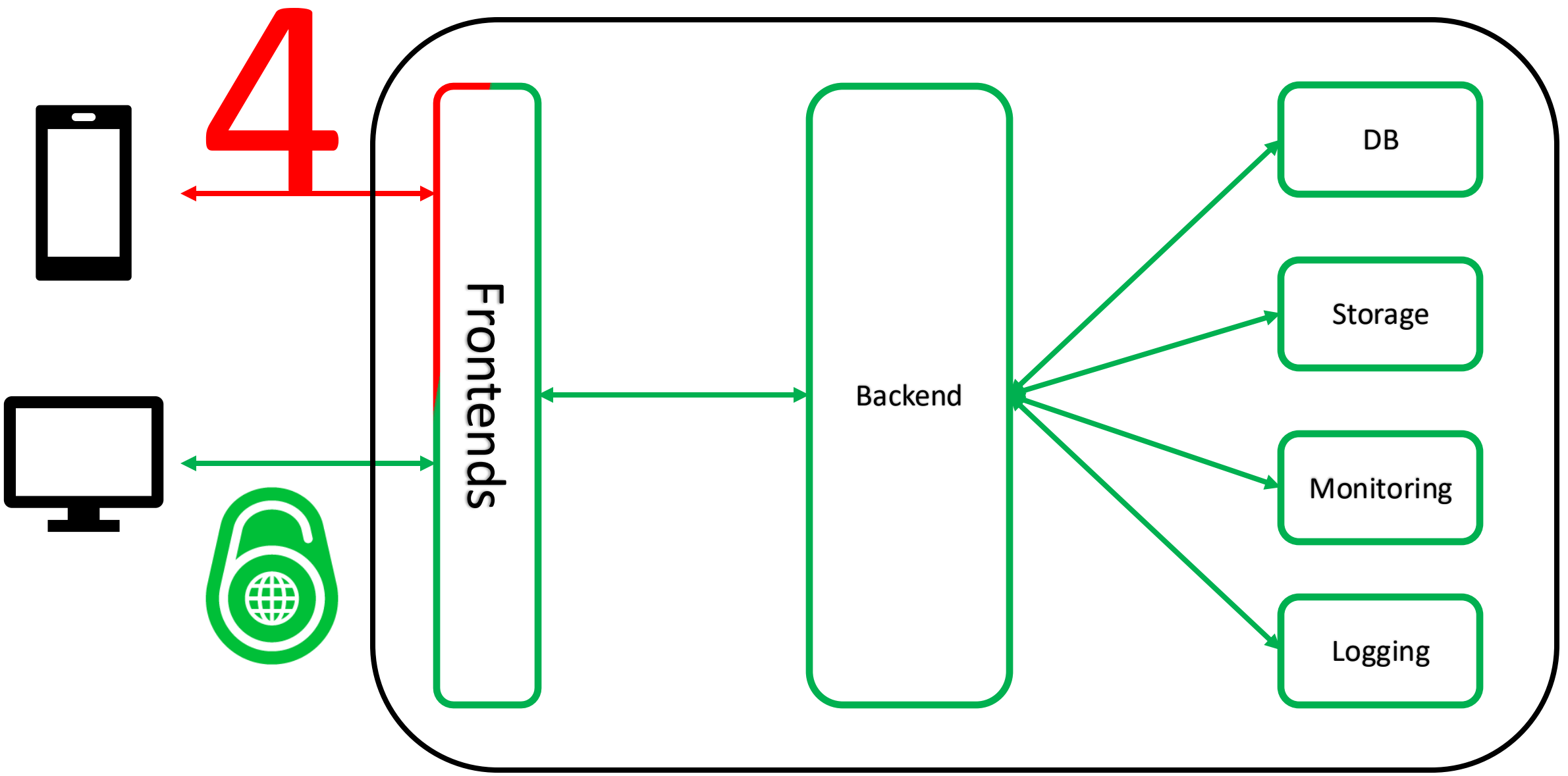
# IPv6 Deployment – Dual-stack



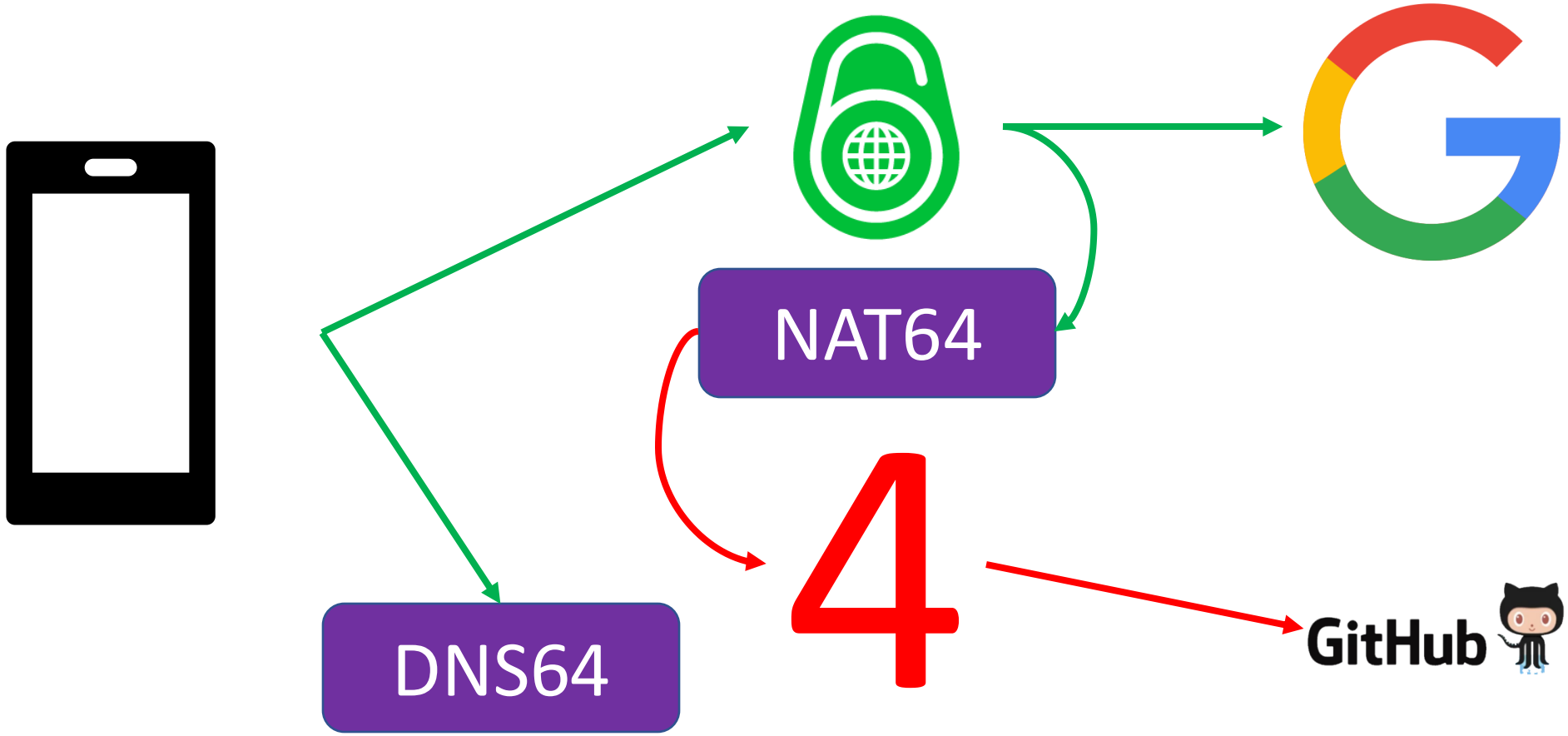
# IPv6 Deployment – IPv6-only



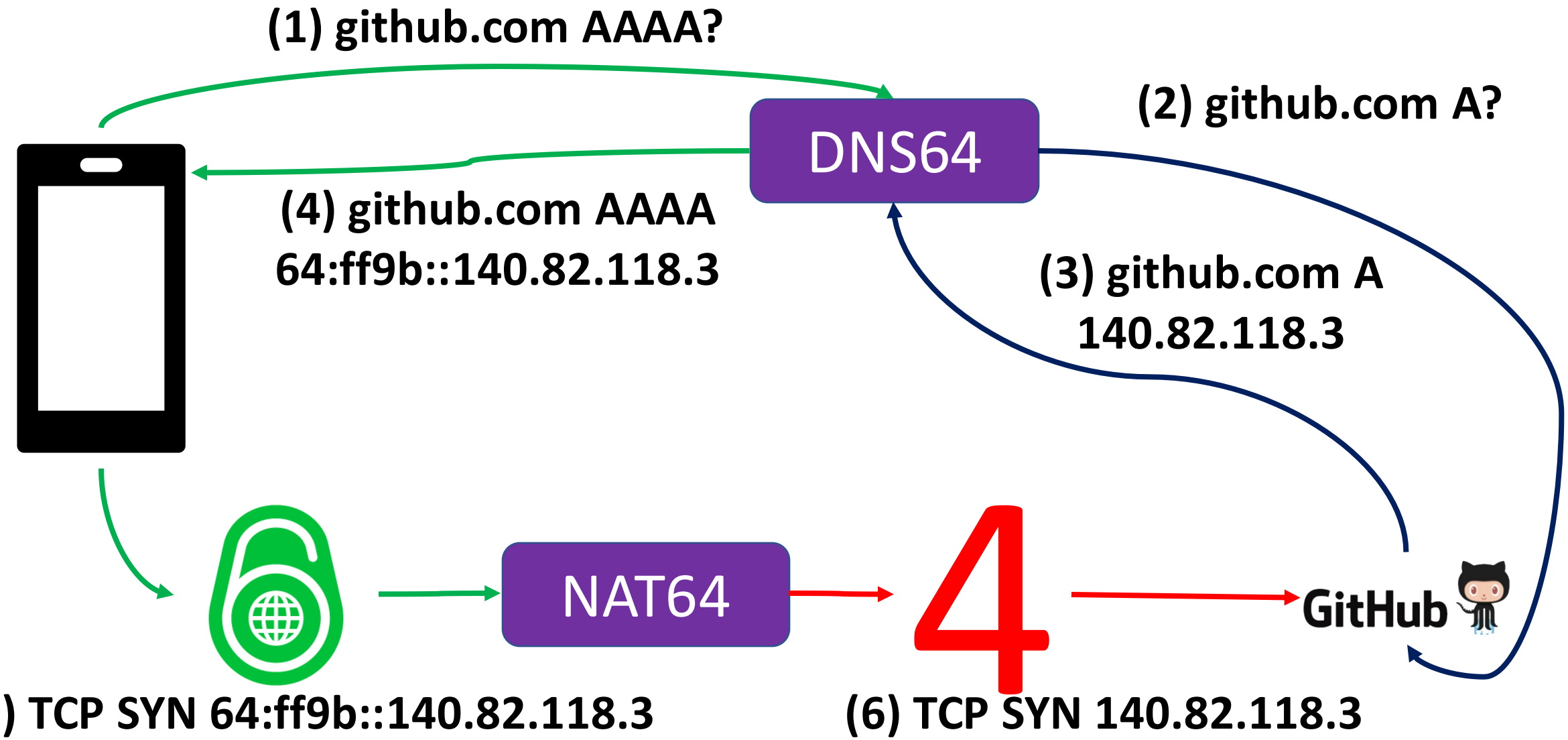
# IPv6 Deployment – IPv6-only internally, dual-stack at the edge



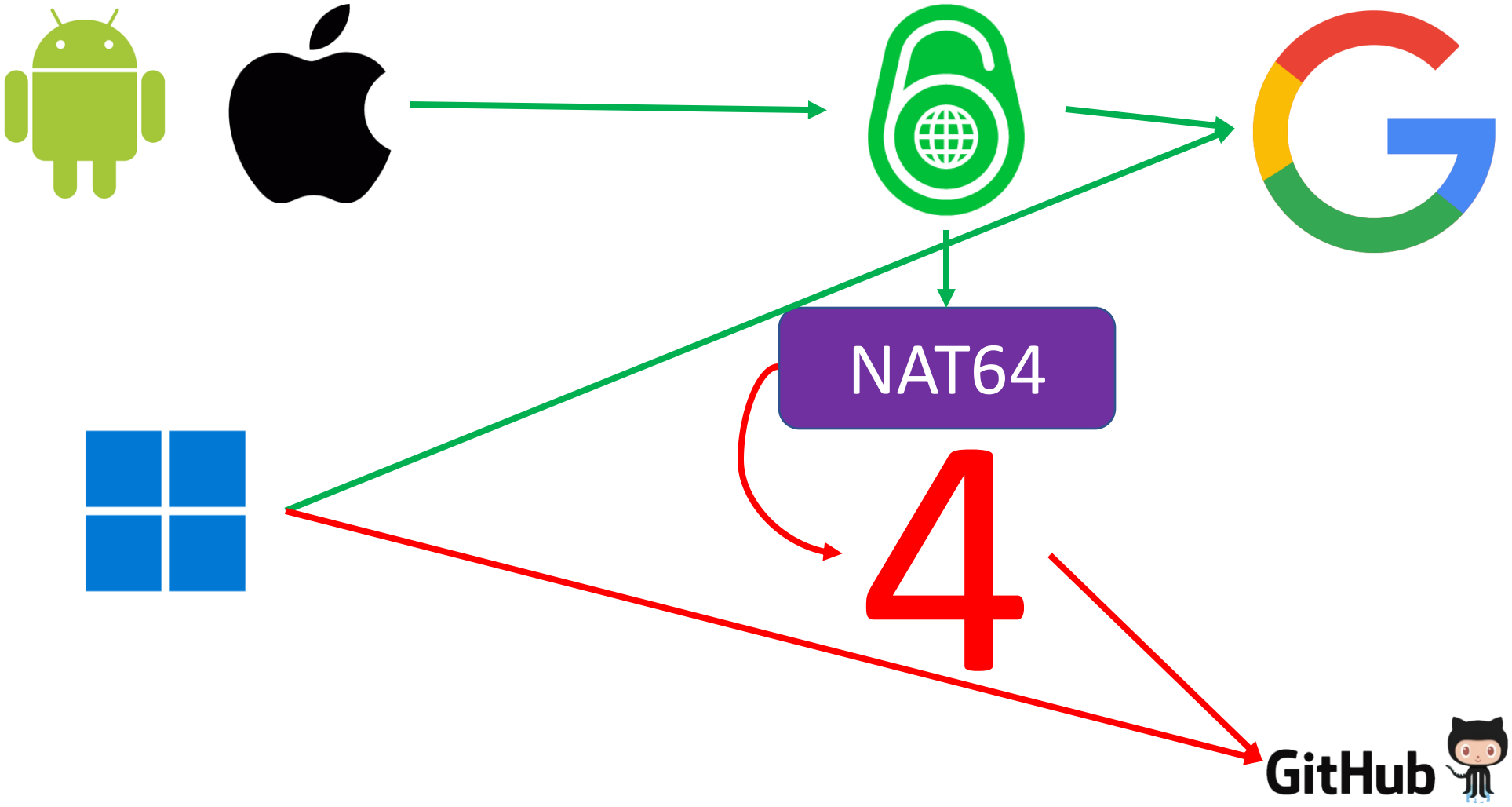
# IPv6 Deployment – IPv6-only internally, NAT64/DNS64 to IPv4 internet



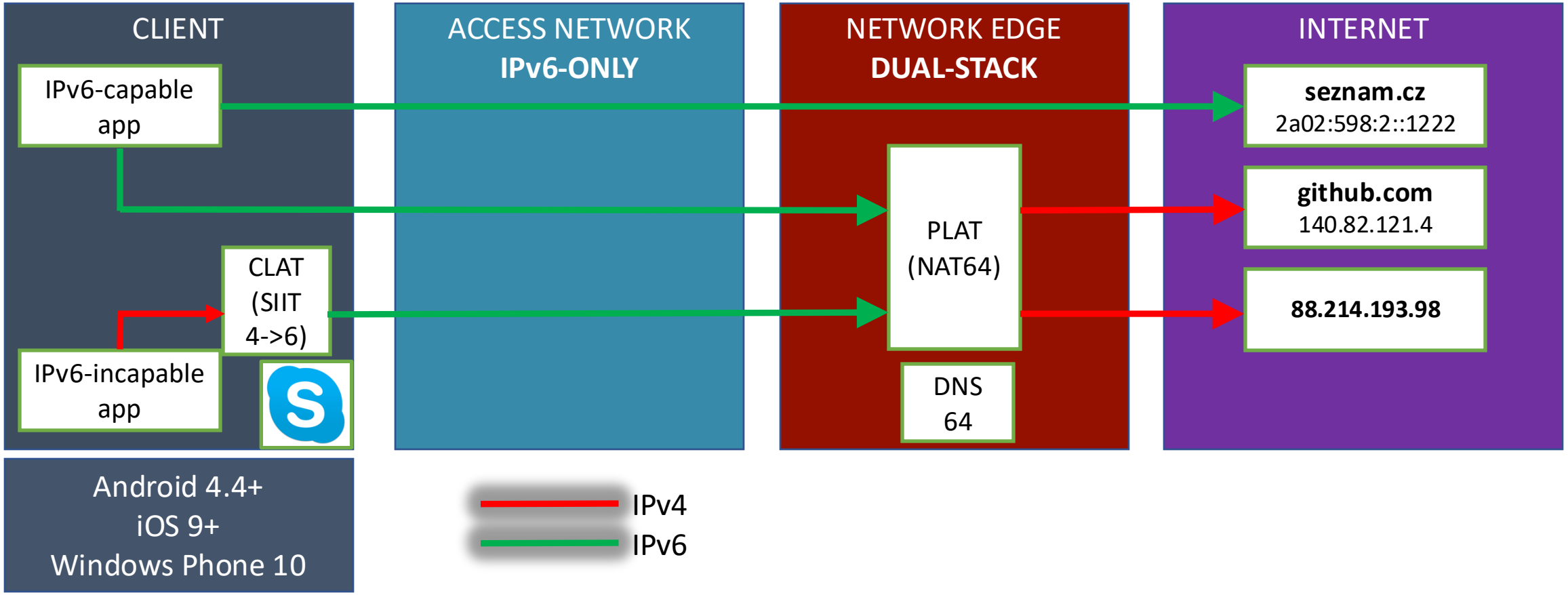
# IPv6 Deployment – IPv6-only internally, NAT64/DNS64 to IPv4 internet



# IPv6 Deployment – IPv6-mostly



# IPv6 Deployment – IPv6-mostly and the 464XLAT




**DNS64 is typically not used in IPv6-mostly networks so that dual-stacked clients think this is a „normal“ network**
**Instead, the NAT64 prefix is signalled via Router Advertisements**

# IPv6 Deployment – IPv6-mostly

**Client:** DHCPv4 Discover, Request: Option 108 (Request is for IP 0.0.0.0)

```
  Boot file name not given
  Magic cookie: DHCP
> Option: (53) DHCP Message Type (Discover)
v Option: (55) Parameter Request List
    Length: 9
    Parameter Request List Item: (1) Subnet Mask
    Parameter Request List Item: (121) Classless Static Route
    Parameter Request List Item: (3) Router
    Parameter Request List Item: (6) Domain Name Server
    Parameter Request List Item: (15) Domain Name
    Parameter Request List Item: (108) IPv6-Only Preferred
    Parameter Request List Item: (114) DHCP Captive-Portal
    Parameter Request List Item: (119) Domain Search
    Parameter Request List Item: (252) Private/Proxy autodiscove
> Option: (57) Maximum DHCP Message Size
> Option: (61) Client identifier
> Option: (51) IP Address Lease Time
> Option: (12) Host Name
> Option: (255) End
```

**Server:** DHCPv4 Offer, Ack: Option 108
Time: 0x708 = 1800 seconds

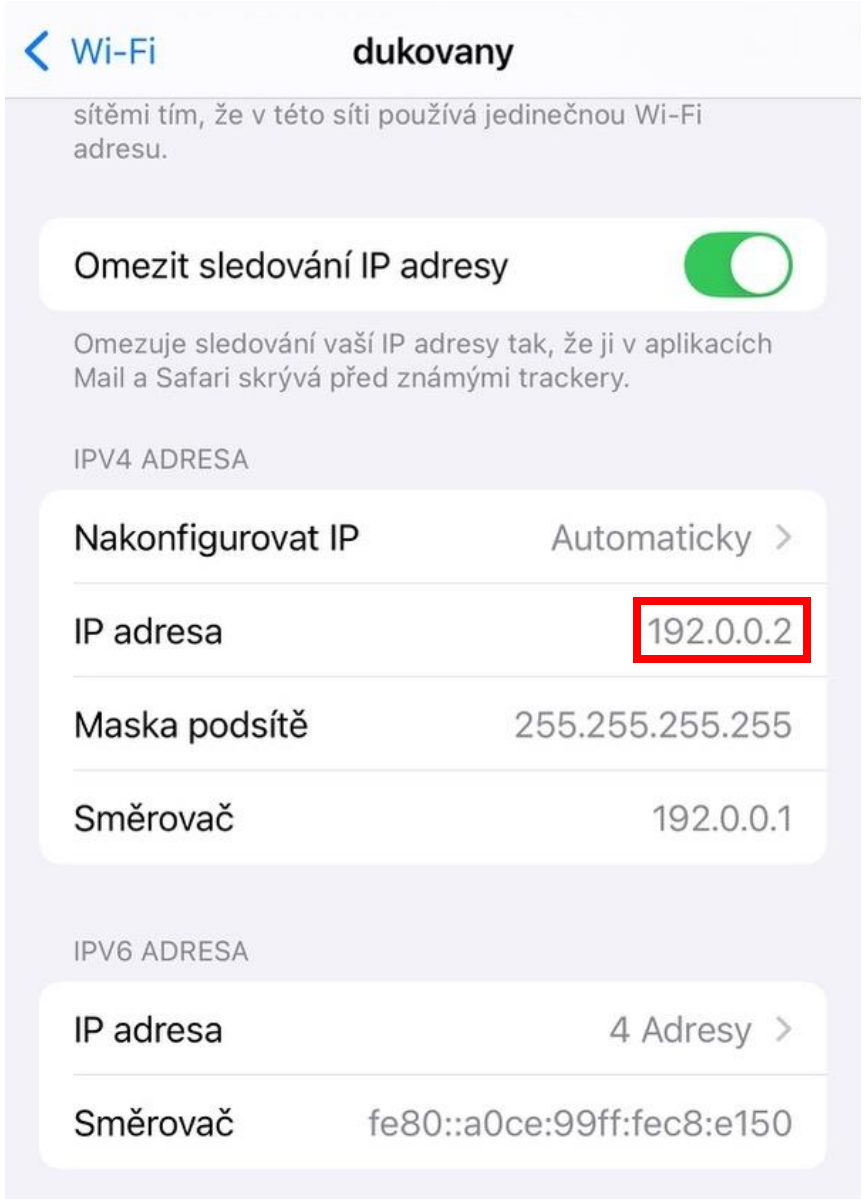
```
> Option: (53) DHCP Message Type (Offer)
> Option: (1) Subnet Mask (255.255.255.0)
> Option: (3) Router
> Option: (6) Domain Name Server
> Option: (12) Host Name
> Option: (15) Domain Name
> Option: (51) IP Address Lease Time
> Option: (54) DHCP Server Identifier
> Option: (61) Client identifier
v Option: (108) IPv6-Only Preferred
    Length: 4
    Value: 00000708
> Option: (255) End
```

# IPv6 Deployment – IPv6-mostly

**Router in the RA:** The NAT64 service is at **`fd00:64::/96`**.

```
> Flags: 0xc0, Managed address configuration, Other configuration, Prf (Default Router Preference): Medium
  Router lifetime (s): 1800
  Reachable time (ms): 300
  Retrans timer (ms): 15
> ICMPv6 Option (Prefix information : 2a03:c22:    ::/64)
> ICMPv6 Option (Prefix information : 2a0c:7040:         ::/64)
v ICMPv6 Option (PREF64 Option)
    Type: PREF64 Option (38)
    Length: 2 (16 bytes)
    0000 0111 0000 1... = Scaled Lifetime: 225
    .... .... .... .000 = PLC (Prefix Length Code): 96 bits prefix length (0x0)
    Prefix: fd00:64::
> ICMPv6 Option (Route Information : High fd00::/64)
> ICMPv6 Option (Route Information : High fd00:64::/64)
> ICMPv6 Option (Recursive DNS Server fd00::1)
> ICMPv6 Option (MTU : 1492)
> ICMPv6 Option (Source link-layer address :                 )
```

# IPv6 Deployment – IPv6-mostly



# Routing IPv4 via an IPv6 Next Hop

**BGP RFC 5549 (2009),** later replaced by **RFC 8950 (2020):**
**Advertising IPv4 Network Layer Reachability Information (NLRI) with an IPv6 Next Hop**

```
root@client:~# ip -4 route add 8.8.4.4 via inet6 fe80::a0de:adff:fe00:beef \
                dev lan src 192.168.0.2
root@client:~# ip -4 route get 8.8.4.4
8.8.4.4 via inet6 fe80::a0de:adff:fe00:beef dev br0.2 src 192.168.0.2

# mtr -w -c3 -b 8.8.4.4
Start: 2024-12-31T23:59:59+0100
HOST:                                     Loss%   Snt   Last   Avg  Best  Wrst StDev
  1.|-- router.local (192.168.0.1)        0.0%     3    0.1   0.1   0.1   0.2   0.0
    (...)
  8.|-- 216.239.56.239                    0.0%     3    5.8   5.7   5.6   5.8   0.1
  9.|-- dns.google (8.8.4.4)              0.0%     3    6.6   6.4   6.2   6.6   0.2
```

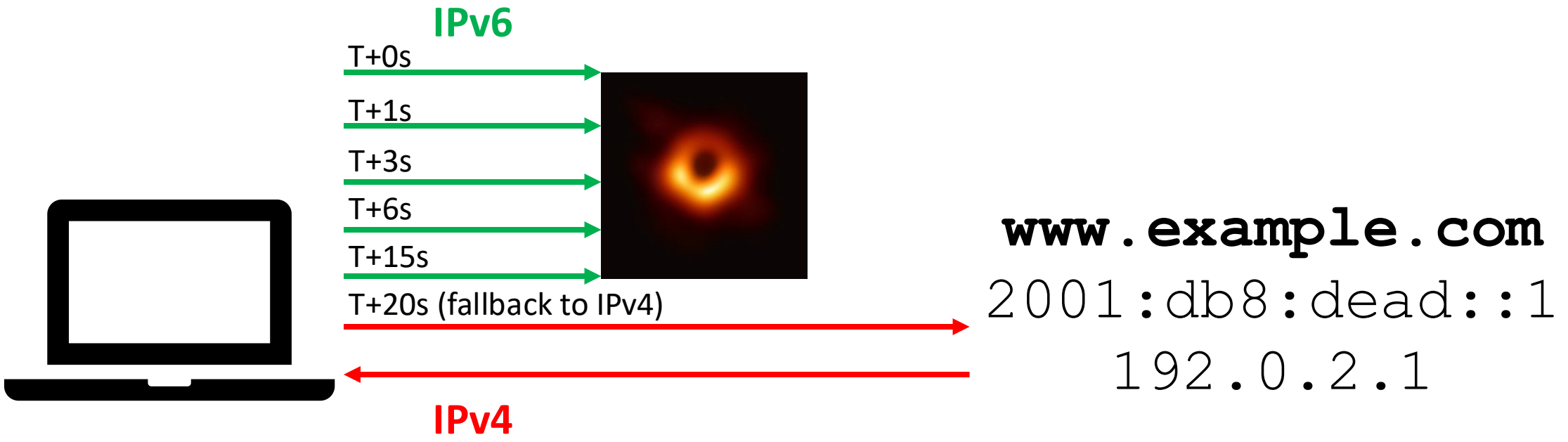
**192.168.0.2 – client's loopback address (no IPv4 on the `lan` network)**
**192.168.0.1 – router's loopback address (no IPv4 on the `lan` network)**
**fe80::a0de:adff:fe00:beef – router's link-local address on the `lan` network**

# Routing IPv4 via an IPv6 Next Hop

```
$ ip a s dev dummy0
4: dummy0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    inet 192.0.2.2/32 scope global dummy0

$ ip route
default proto bird src 192.0.2.2 metric 32
       nexthop via inet6 fe80::464c:deff:fead:beef dev uplink1 weight 1
       nexthop via inet6 fe80::c2d6:deff:fead:beef dev uplink2 weight 1

$ ip -4 neigh show
(no output - empty ARP table)

$ ip -6 neigh show
fe80::464c:deff:fead:beef dev uplink1 lladdr 44:4c:de:ad:be:ef router REACHABLE
fe80::c2d6:deff:fead:beef dev uplink2 lladdr c0:d6:de:ad:be:ef router REACHABLE

Router's loopback can be for example 192.0.2.254

Traditional environments: /31 per uplink, 1 dummy address = 5 IPv4 addresses
This model: 1 IPv4 address needed (for dummy0), 4 saved
```

# IPv6 brokennes and Happy eyeballs

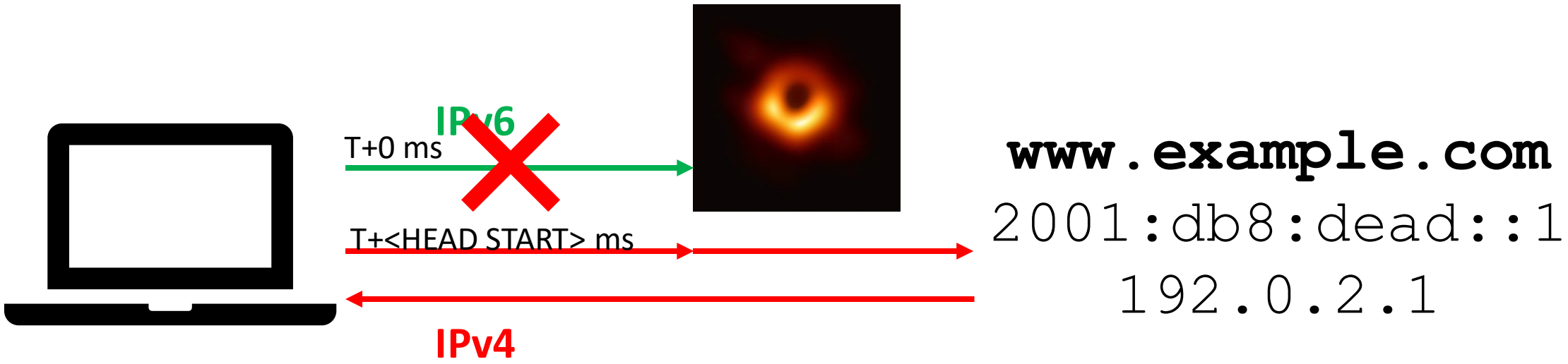
# Connection brokenness in dual-stack networks




Fallback can take from tens of seconds to tens of minutes, depending on a platform and number of AAAA records.

**User's view: the service is unreachable**

# Happy Eyeballs: Fixing brokenness on dual-stack networks




| Client | Head start |
|---|---|
| Python `rfc6555` | 0 ms |
| cURL | 200 ms |
| Chrome, Firefox | 300 ms |
| Safari (HE v2) | 25 ms |

# Linux: addressing, routing, firewalling, troubleshooting

# Linux: Configuring IPv6 addresses (ifupdown, Netplan)

```
network:
  ethernets:
    eth0:
      dhcp4: true
      dhcp6: true
      accept-ra: true
    eth1:
      dhcp4: false
      dhcp6: false
      accept-ra: false
      addresses:
      - 2001:db8:b0ba:deaf::9/64
      - 192.168.1.12/24
      routes:
      - to: default
        via: 192.168.1.1
      - to: default
        via: 2001:db8:b0ba:deaf::1
        on-link: true
  version: 2
```

```
auto eth0
iface eth0 inet dhcp
iface eth0 inet6 dhcp
    autoconf 1
    accept_ra 1

auto eth1
iface eth1 inet static
    address 192.168.1.12
    netmask 255.255.255.0
    gateway 192.168.1.1
iface eth1 inet6 static
    address 2001:db8:b0ba:deaf::9
    netmask 64
    gateway 2001:db8:b0ba:deaf::1
    autoconf 0
    accept_ra 0
```

Documentation: ifupdown (Debian), netplan

# Linux: Diagnostics – ping, mtr

```
root@server:~# ping6 -c3 nix.cz
PING nix.cz(2a02:38:2::171 (2a02:38:2::171)) 56 data bytes
64 bytes from 2a02:38:2::171 (2a02:38:2::171): icmp_seq=1 ttl=56 time=1.82 ms
64 bytes from 2a02:38:2::171 (2a02:38:2::171): icmp_seq=2 ttl=56 time=0.690 ms
64 bytes from 2a02:38:2::171 (2a02:38:2::171): icmp_seq=3 ttl=56 time=0.660 ms

--- nix.cz ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2015ms
rtt min/avg/max/mdev = 0.660/1.058/1.825/0.542 ms
```

```
root@server:~# mtr -6 -c3 -w -n nix.cz
Start: 2022-12-05T20:24:48+0000
HOST: server                      Loss%   Snt   Last   Avg  Best  Wrst StDev
  1.|-- 2a03:3b40:42:3:4::2        0.0%     3    2.4   2.3   2.2   2.4   0.1
  2.|-- 2a03:3b40:42:2:4::1        0.0%     3    3.2   1.4   0.5   3.2   1.5
  3.|-- 2a03:3b40:42:2:48::2       0.0%     3    0.4   0.4   0.3   0.4   0.1
  4.|-- 2a01:430:0:fe0b::1         0.0%     3    2.6   1.5   0.9   2.6   1.0
  5.|-- 2a01:430:200:d:1700:1:0:2  0.0%     3   41.9  14.4   0.7  41.9  23.8
  6.|-- 2a01:430:ff:1331:1::2      0.0%     3    6.9   2.6   0.4   6.9   3.7
  7.|-- 2001:7f8:14::4            33.3%     3    1.5   1.5   1.5   1.5   0.0
  8.|-- 2a02:38:f:f::4             0.0%     3    0.5   1.5   0.5   3.6   1.8
  9.|-- 2a02:38:2::171             0.0%     3    0.7   0.9   0.6   1.4   0.4
```

# Linux: Diagnostics – tcpdump

```
root@server:~# tcpdump -nn -i venet0 ip6 and tcp port not 22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on venet0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:26:48.777462 IP6 2a05:d018:474:1702:fae9:4328:1b9d:7f99 > 2a01:430:17:1::ffff:1111: ICMP6, echo request, seq 250, length 16
20:26:48.777550 IP6 2a01:430:17:1::ffff:1111 > 2a05:d018:474:1702:fae9:4328:1b9d:7f99: ICMP6, echo reply, seq 250, length 16
20:26:52.244703 IP6 2600:1f13:4bb:a02:4a3e:3ca1:bb69:d72c > 2a01:430:17:1::ffff:1111: ICMP6, echo request, seq 14862, length 16
20:26:52.244879 IP6 2a01:430:17:1::ffff:1111 > 2600:1f13:4bb:a02:4a3e:3ca1:bb69:d72c: ICMP6, echo reply, seq 14862, length 16
20:26:56.901196 IP6 2600:1f16:f13:8a01:34cd:1760:1117:8780 > 2a01:430:17:1::ffff:1111: ICMP6, echo request, seq 20472, length
16
20:26:56.901568 IP6 2a01:430:17:1::ffff:1111 > 2600:1f16:f13:8a01:34cd:1760:1117:8780: ICMP6, echo reply, seq 20472, length 16
20:26:57.287730 IP6 fe80::fc4f:6cff:fed7:c554 > 2a01:430:17:1::ffff:1111: ICMP6, neighbor solicitation, who has
2a01:430:17:1::ffff:1111, length 32
20:26:57.288705 IP6 2a01:430:17:1::ffff:1111 > fe80::fc4f:6cff:fed7:c554: ICMP6, neighbor advertisement, tgt is
2a01:430:17:1::ffff:1111, length 24

8 packets captured
930 packets received by filter
829 packets dropped by kernel
```

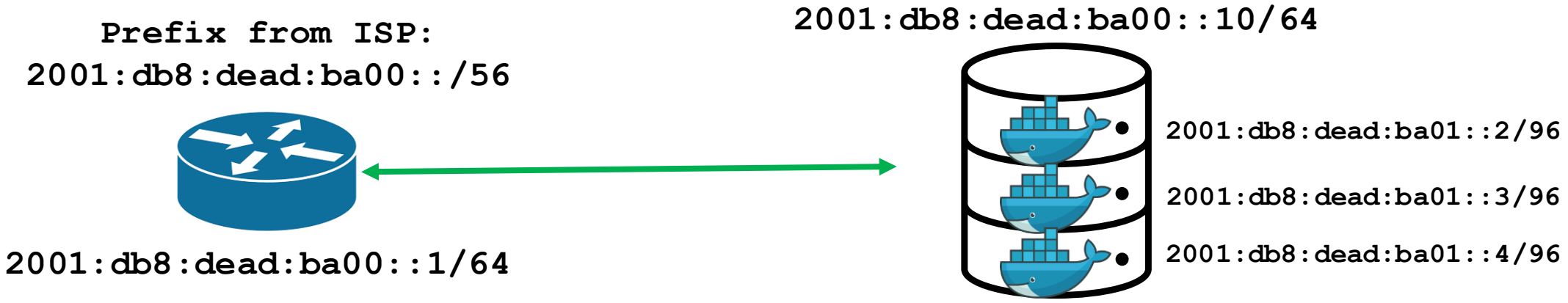
# Linux: Firewalling IPv6 using ip6tables (NAT is not a firewall!)

```
root@server:~# ip6tables-save
# Generated by ip6tables-save v1.6.1 on Mon Dec  5 20:29:24 2022
*filter
:INPUT ACCEPT [4952088:1556483044]
:FORWARD ACCEPT [109090:6997290]
:OUTPUT ACCEPT [3066241:816422658]
-A INPUT -p ipv6-icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
-A INPUT -i eth0 -j REJECT --reject-with icmp6-port-unreachable
-A FORWARD -p ipv6-icmp -j ACCEPT
-A FORWARD -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i eth0 -j REJECT --reject-with icmp6-port-unreachable
COMMIT
```

# Linux: Firewalling IPv6 using nftables (NAT is not a firewall!)

```
root@server:~# cat /etc/nftables.conf
#!/usr/sbin/nft -f

flush ruleset

table inet filter {
  chain input {
    type filter hook input priority 0;
    ct state invalid counter drop
    ct state {established, related} counter accept
    iifname "lo" accept
    iifname "lan" accept
    iif != lo ip6 daddr ::1/128 counter
    iif != lo ip daddr 127.0.0.1/8 counter
    tcp dport { ssh, 53, http, https } accept
    udp dport { 53, 5353 } accept
    ip protocol icmp counter accept
    ip6 nexthdr icmpv6 counter accept
    ip6 saddr fe80::/10 accept comment "accept link-local traffic"
    ip6 saddr ff00::/8 accept comment "accept multicast"
    drop
  }
}
```

# Linux: Routing /64 to a computer in LAN for Docker




```
root@router:~# ip -6 route add blackhole 2001:db8:dead:ba00::/56
root@router:~# ip -6 address add 2001:db8:dead:ba00::1/64 dev lan
root@router:~# ip -6 route add 2001:db8:dead:ba01::/96 via 2001:db8:dead:ba00::10
root@router:~# sysctl net/ipv6/conf/all/forwarding=1

root@containerhost:~# ip -6 address add 2001:db8:dead:ba00::10/64 dev lan
root@containerhost:~# ip -6 route add default via 2001:db8:dead:ba00::1 dev lan
root@containerhost:~# sysctl net/ipv6/conf/all/forwarding=1
root@containerhost:~# cat <<EOF > /etc/docker/daemon.json
{
    "fixed-cidr-v6": "2001:db8:dead:ba01::/96",
    "ipv6": true
}
EOF
root@containerhost:~# systemctl restart docker.service
```

# Applications: listening and connecting

# Listening on IPv6(-only)

NginX vs. Apache2

```
root@server1:~# grep listen /etc/nginx/sites-enabled/default
  listen 80 default_server;
  listen [::]:80 default_server;
  listen 443 ssl default_server;
  listen [::]:443 ssl default_server;
```

```
root@erver:~# ss -lnt | egrep ':(80|443)'
LISTEN  0.0.0.0:443  0.0.0.0:*
LISTEN  0.0.0.0:80   0.0.0.0:*
LISTEN  [::]:443     [::]:*
LISTEN  [::]:80      [::]:*
```

```
root@server1:~# netstat -lntp | egrep ':(80|443)'
tcp     0.0.0.0:443             0.0.0.0:*
tcp     0.0.0.0:80              0.0.0.0:*
tcp6    :::443                  :::*
tcp6    :::80                   :::*
```

```
root@server2:~# grep  -ri listen /etc/apache2/ports.conf
  Listen 80
  Listen 443
```

```
root@server2:~# ss -lnt | egrep ':(80|443)'
LISTEN  *:443         *:*
LISTEN  *:80          *:*
```

```
root@server2:~# netstat -lntp | egrep ':(80|443)'
tcp6    :::80                   :::*
tcp6    :::443                  :::*
```

`/proc/sys/net/ipv6/bindv6only`

(global default, set to "`0`" in modern distros)

**`IPV6_V6ONLY` IPv6 socket option**

**(Apache = not set by default, NginX = set by default)**

# Listening on `localhost`

```
root@server1:~# ss -lnt | grep 8953 # (unbound control interface)
LISTEN 127.0.0.1:8953       0.0.0.0:*
LISTEN [::1]:8953           [::]:*

/opt/nginx_exporter/bin/nginx_exporter -telemetry.address 127.0.0.1:9113 (...)

root@server2:~# ss -lnt | grep 9113 # (nginx_exporter)
LISTEN 127.0.0.1:9113       0.0.0.0:*

root@server2:~# wget -O /dev/null http://localhost:9113/metrics
--2022-12-05 21:44:00--  http://localhost:9113/metrics
Connecting to localhost (localhost)|::1|:9113... failed: Connection refused.
Connecting to localhost (localhost)|127.0.0.1|:9113... connected.
HTTP request sent, awaiting response... 200 OK

/opt/nginx_exporter/bin/nginx_exporter -telemetry.address [::]:9113 (...)

root@server2:~# ss -lnt | grep 9113
LISTEN    *:9113              *:*

root@server2:~# wget -O /dev/null http://localhost:9113/metrics
--2022-12-05 21:47:02--  http://localhost:9113/metrics
Connecting to localhost (localhost)|::1|:9113... connected.
HTTP request sent, awaiting response... 200 OK
```

# Listening recommendations

On dual-stacked hosts, **all** published services **shall** listen on both IPv4 and IPv6
(either option is fine: using dual-protocol socket or using one socket per protocol)

On IPv6-only hosts, **all** published services **must** listen on IPv6
(local communication using IPv4 localhost **may** be accepted)

**Default** service configuration **will often not meet your needs**; always verify its settings

If dual-stacking, **always** monitor your services via **both** protocols

# Connecting from your app to an IPv6-enabled host

```
$ python3 # on node with IPv6 enabled and working
>>> import socket
>>> for struc in socket.getaddrinfo("www.google.com", "443", 0, socket.SOCK_STREAM):
...     print(struc)
...
(<AF_INET6: 30>, <SOCK_STREAM: 1>, 6, '', ('2a00:1450:4014:80e::2004', 443, 0, 0))
(<AF_INET: 2>, <SOCK_STREAM: 1>, 6, '', ('142.251.36.132', 443))

$ python3 # on node without IPv6
>>> import socket
>>> for struc in socket.getaddrinfo("www.google.com", "443", 0, socket.SOCK_STREAM):
...     print(struc)
...
(<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('142.251.36.132', 443))
(<AddressFamily.AF_INET6: 10>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('2a00:1450:4014:80b::2004', 443,
0, 0))
```

**Use getaddrinfo**, follow the order it gives you (better: use Happy Eyeballs-enabled libraries, if available)
node.js forced IPv4 (which broke IPv6-only deployments) until v17

**Listen on both IPv4 and IPv6 sockets** on the machine, even if only listening on `localhost`
many have failed to follow this recommendation and now want node.js to revert to the broken behavior

# Application examples

# IPv6 and DNS: IPv6-enabled hostname

```
server.homelab.cz. IN A    192.0.2.1
server.homelab.cz. IN AAAA 2001:db8:deaf:beef::c001:cafe
test.homelab.cz.   IN AAAA 2001:db8:deaf:beef::1337:b00c
```

```
2a06:4880:3000::f - - [02/Dec/2022:17:09:36 +0100] "GET /
HTTP/1.1" 302 188 "-" "Mozilla/5.0 (compatible;
InternetMeasurement/1.0; +https://internet-
measurement.com/)"
```

# IPv6 and DNS: IPv6-enabled transport

```
$ dig -4 aaaa nix.cz @dns.google

;; ANSWER SECTION:
nix.cz.                 19395   IN      AAAA    2a02:38:2::171

;; Query time: 14 msec
;; SERVER: 8.8.4.4#53(8.8.4.4)
;; WHEN: Mon Dec 05 22:18:19 CET 2022
;; MSG SIZE  rcvd: 63


$ dig -6 a nix.cz @dns.google

;; ANSWER SECTION:
nix.cz.                 21600   IN      A       93.190.134.171

;; Query time: 31 msec
;; SERVER: 2001:4860:4860::8844#53(2001:4860:4860::8844)
;; WHEN: Mon Dec 05 22:18:15 CET 2022
;; MSG SIZE  rcvd: 51
```

# IPv6 and VPN: IPv6-enabled VPN server

```
$ host vpn.somecompany.com
vpn.somecompany.com has address 192.0.2.66
vpn.somecompany.com has IPv6 address 2001:db8:1000::66
```

# IPv6 and VPN: IPv6-enabled OpenVPN tunnel

```
root@myverysecurecomputer:~# ip address show dev tuncompany
271: tuncompany: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc
fq_codel state UNKNOWN group default qlen 100
    link/none
    inet 172.31.66.14/24 brd 172.32.66.255 scope global tuncompany
       valid_lft forever preferred_lft forever
    inet6 2001:db8:f00d:beef:ce5e:2:0:100c/112 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::ee1a:4701:230e:16bd/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
```

**Sample openvpn server extra config lines for IPv6 (/etc/openvpn/server.conf):**

```
proto udp6
server-ipv6 2001:db8:f00d:beef:ce5e:2::/112
```

(plus proper routing of that /112 prefix)

# IPv6 and VPN: IPv6-enabled Wireguard tunnel

**Sample wireguard server config for IPv6 (/etc/wireguard/wg0.conf;** plus proper routing of that /64 prefix**):**

```
[Interface]
Address = 192.168.102.1/24, 2001:db8:7:6543::1/64
ListenPort = 12345
PrivateKey = (...)

[Peer]
AllowedIPs = 192.168.102.2/32, 2001:db8:7:6543::1001/128
PublicKey = (...)
```

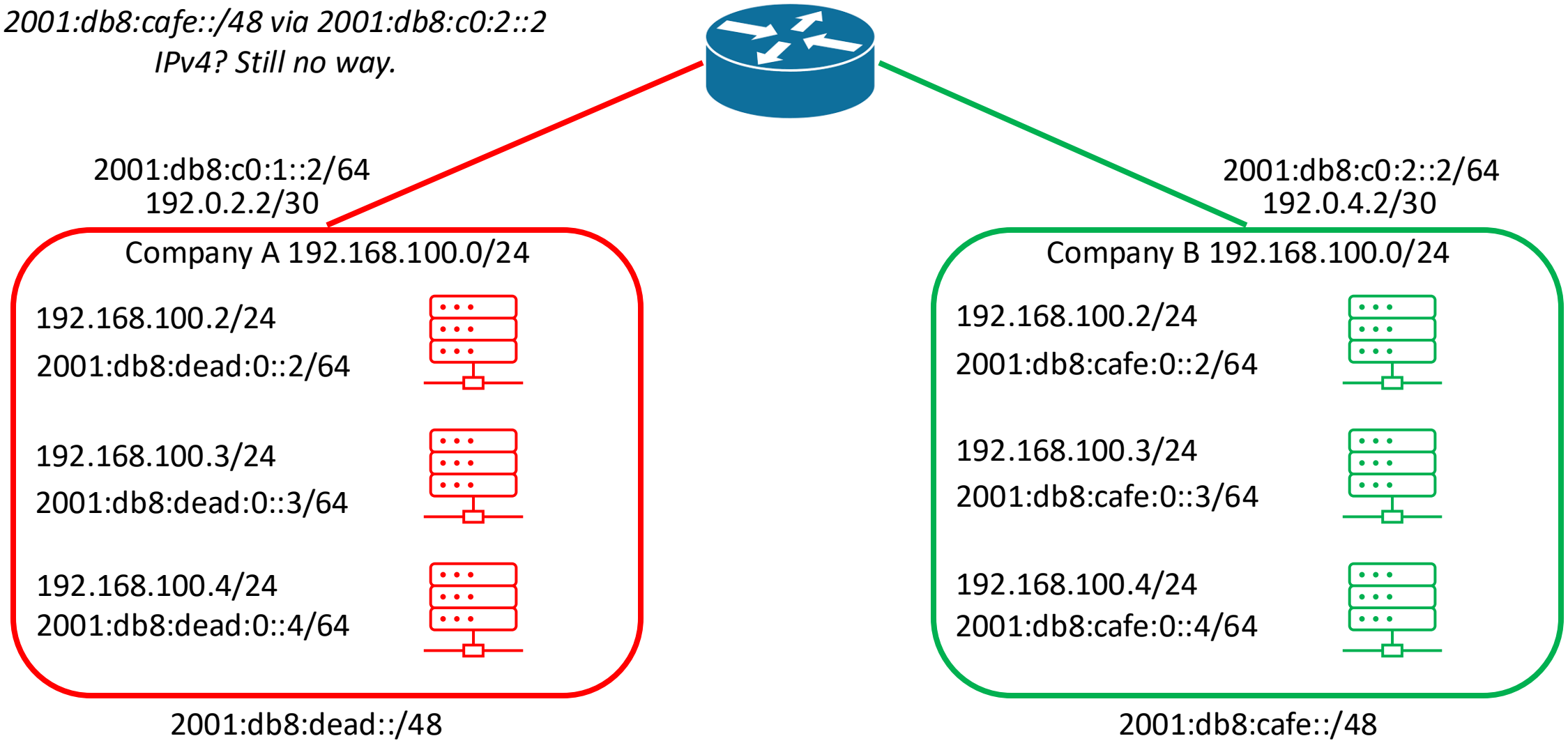
**Sample wireguard client config for IPv6:**

```
[Interface]
PrivateKey = (...)
Address = 192.168.102.2/32, 2001:db8:7:6543::1001/128
DNS = 8.8.8.8, 2001:4860:4860::8888

[Peer]
PublicKey = (...)
AllowedIPs = ::/0, 0.0.0.0/0
Endpoint = vpn.somecompany.com:12345
```

# Examples revisited with IPv6

# IPv6 connecting multiple networks

*2001:db8:dead::/48 via 2001:db8:c0:1::2*
*2001:db8:cafe::/48 via 2001:db8:c0:2::2*
*IPv4? Still no way.*



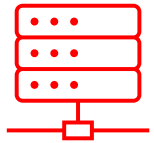
# IPv6 to all the things!

Which server will be available on port 80/tcp on IPv4? And on IPv6?
Which server will be available on port 443/tcp on IPv4? And on IPv6?
Which one will manage to obtain the certificate from Let‘s Encrypt?

**9.9.9.9, 2001:db8:dead::/48**

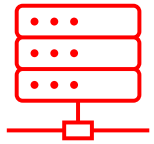
192.168.100.1/24
2001:db8:dead::1/64

192.168.100.2/24
2001:db8:dead::2/48



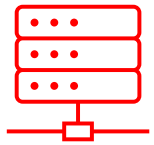
I want to be a webserver on 80/tcp and get a TLS certificate for **`burninghearts.com`** from Let‘s Encrypt!

~~192.168.100.3/24~~
2001:db8:dead::3/48



No, I want to be a webserver on 80/tcp and 443/tcp and get a TLS certificate for **`restinpeace.com`** from Let‘s Encrypt!

192.168.100.4/24
2001:db8:dead::4/48



Silence, you losers! I want all the ports on 9.9.9.9 just for myself, including 80/tcp and 443/tcp. I already have a certificate, don‘t care much about Let‘s Encrypt.

# Applications: (un)expected surprises

# Applications: unexpected surprises

Even if your firewall, routing and addressing is fine and the application listens on IPv6 sockets, it might still be broken in ways you don't anticipate. For example:

- It may store IP addresses in databases as text, in colum size too small for IPv6
- It may be comparing IP addresses as text (not all v6 may be canonicalized)
- It may be doing weird things when connecting to the world (remember node.js?)
- It may not be capable of properly configuring IPv6 ACLs
- It may not present IPv6 addresses properly (e.g., in audit logs)
- It may be using IP-based sessions (in general, doesn't work well with Happy Eyeballs and/or with v6, where host bits in your IP may change at any time)
- It may be connecting to IP addresses instead of hostnames
- It may not work in IPv6-only environment at all due to bad coding standards

Follow the Cisco guide, ARIN guide, Andrew Yourtchenko presentation or the Apple guide. There's also a great guide by ARCEP (French telco regulator) on deploying IPv6 in the Enterprise environment.

**TRUST YOUR VENDORS, BUT ALWAYS VERIFY THE REALITY.**

# Thank you

# &

# Q & A

linkedin.com/in/radek-zajic/

github.com/zajdee

 @zajDee