

The case for IPv6-only data centres

...and how to pull it off in today's IPv4-dominated world

Tore Anderson

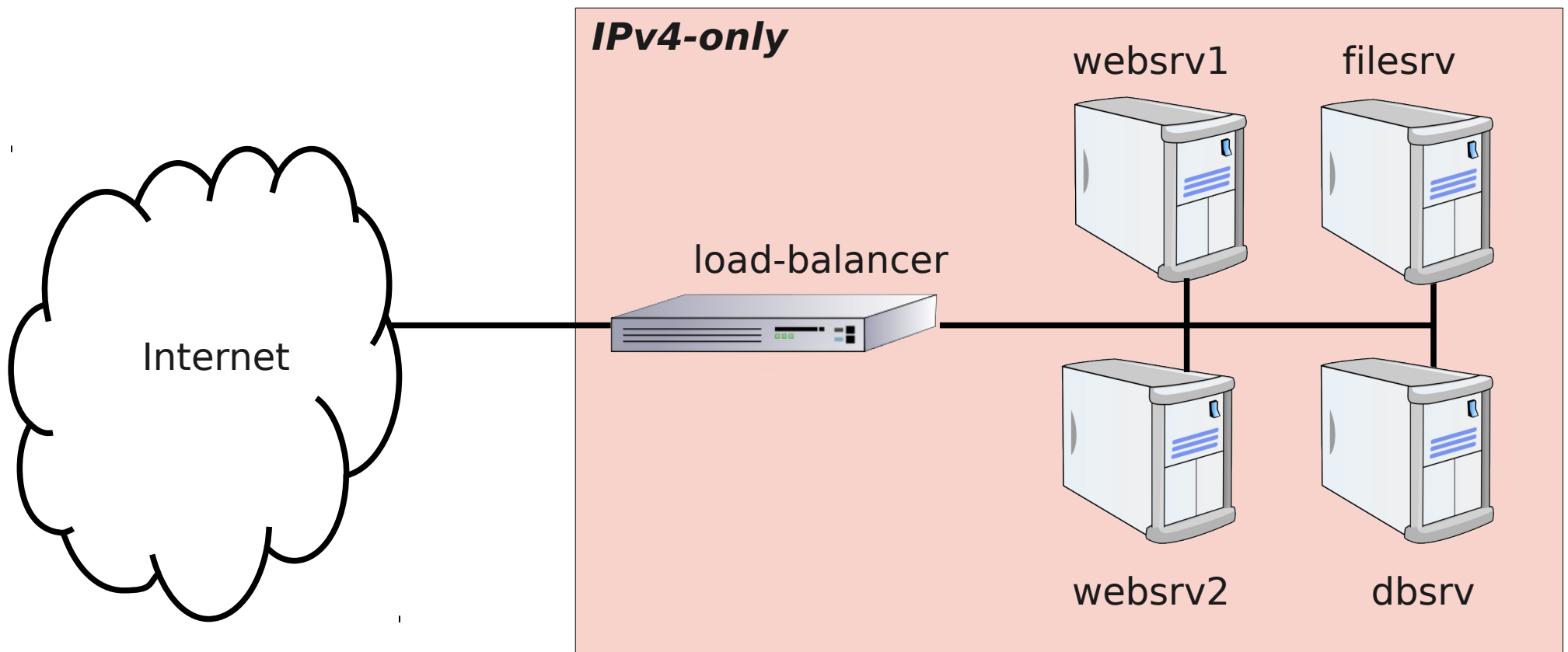
Redpill Linpro AS

V6 World Congress, Paris, February 2012



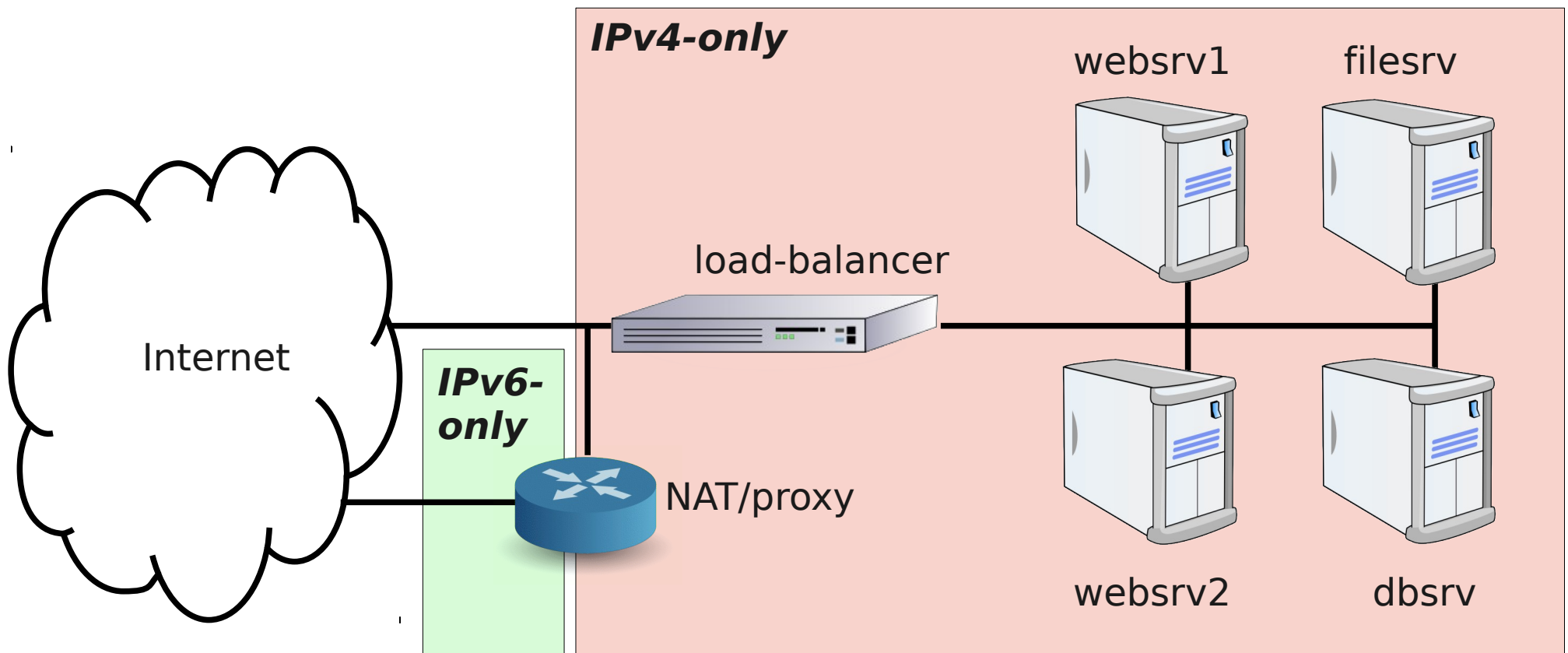
Incremental IPv6 deployment

0) Traditional IPv4-only DC environment



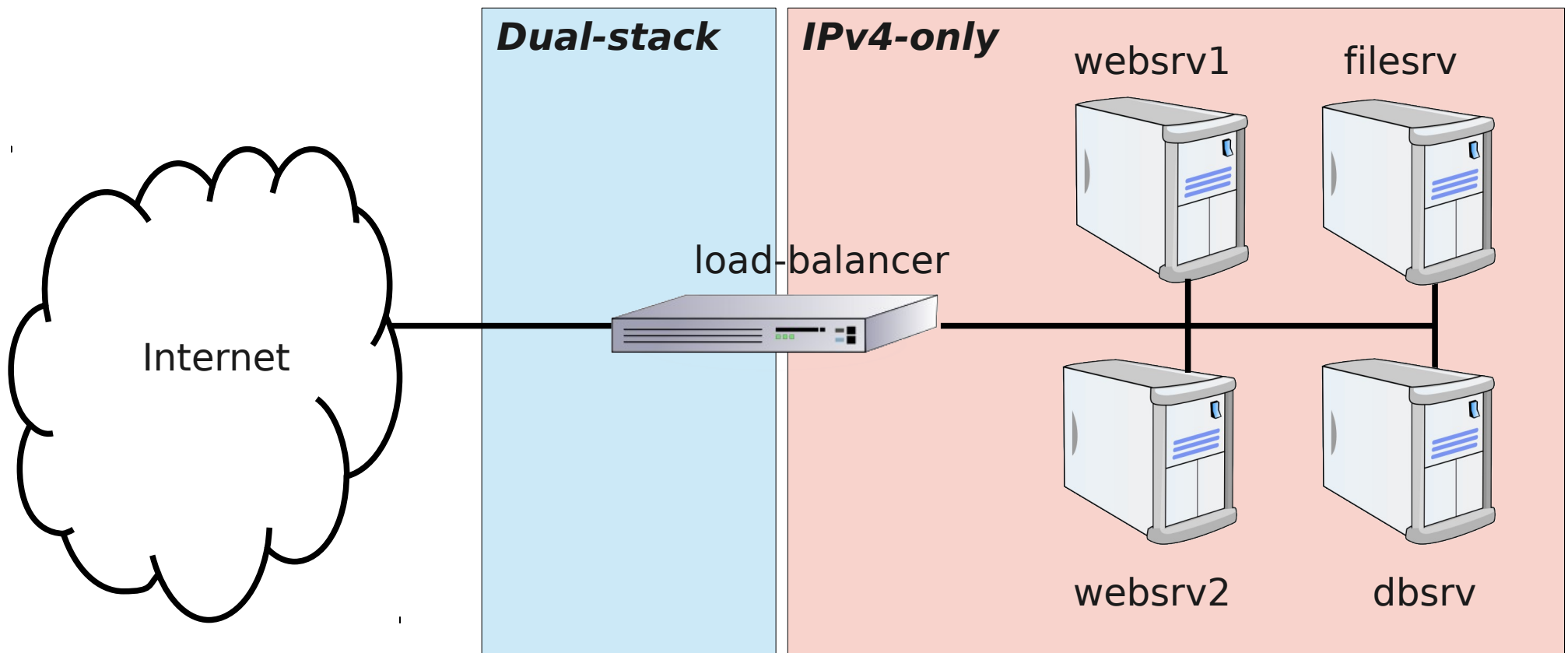
Incremental IPv6 deployment

1) IPv4-only + IPv6 via NAT or proxy



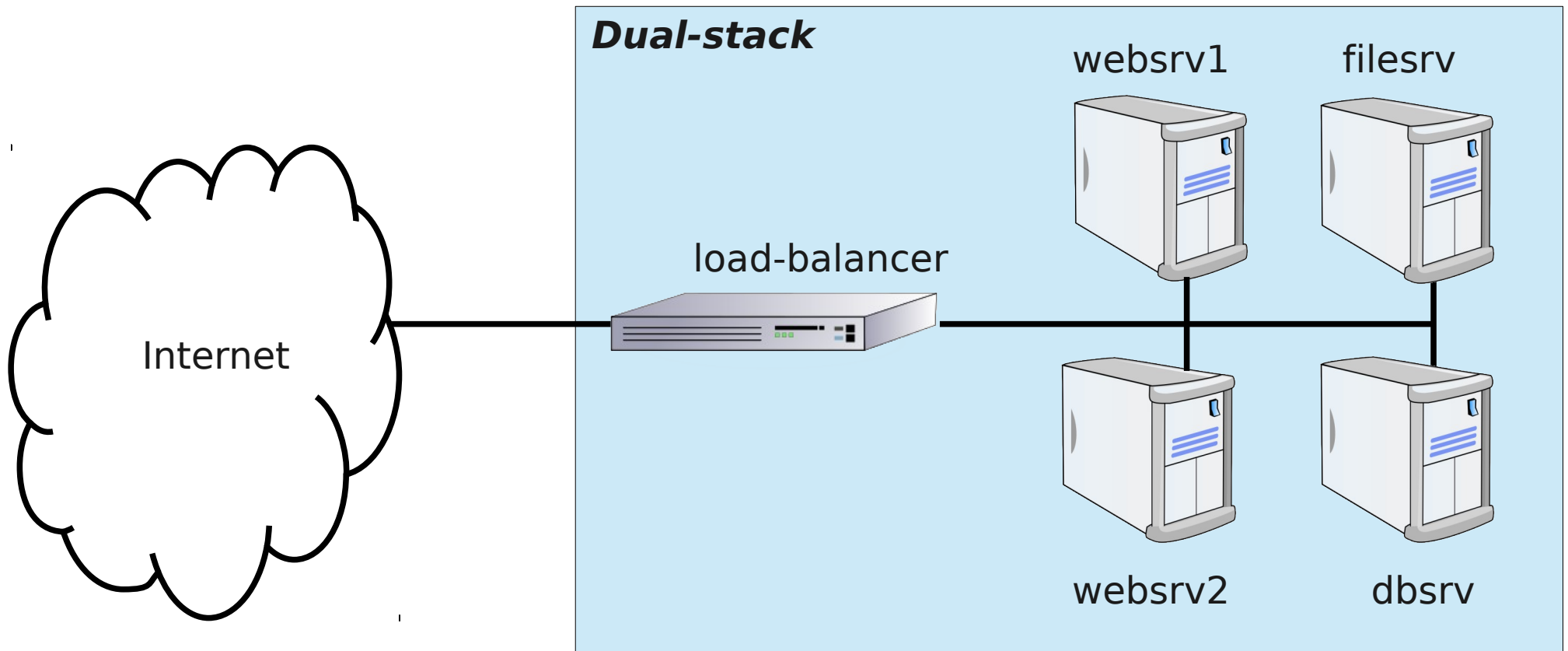
Incremental IPv6 deployment

2) Dual-stacked public front-end, IPv4 BE



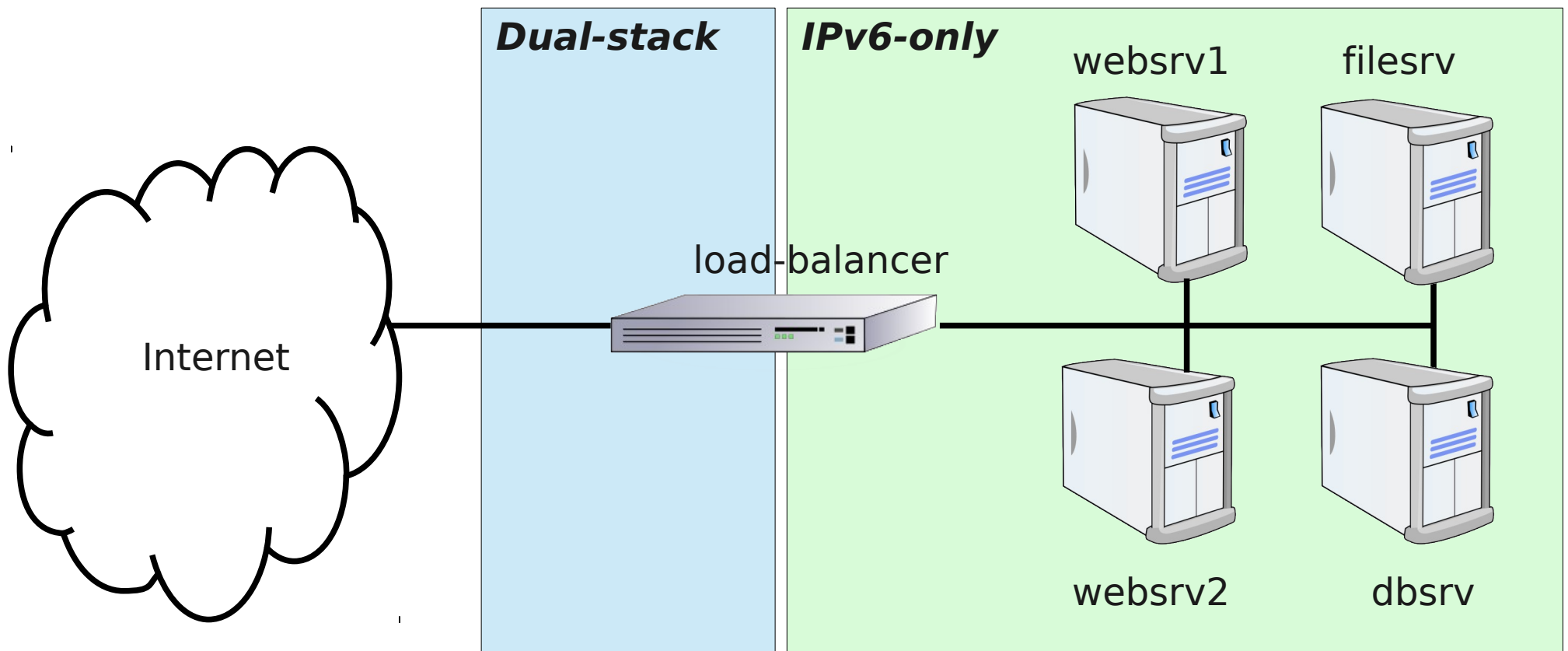
Incremental IPv6 deployment

3) Full dual-stack



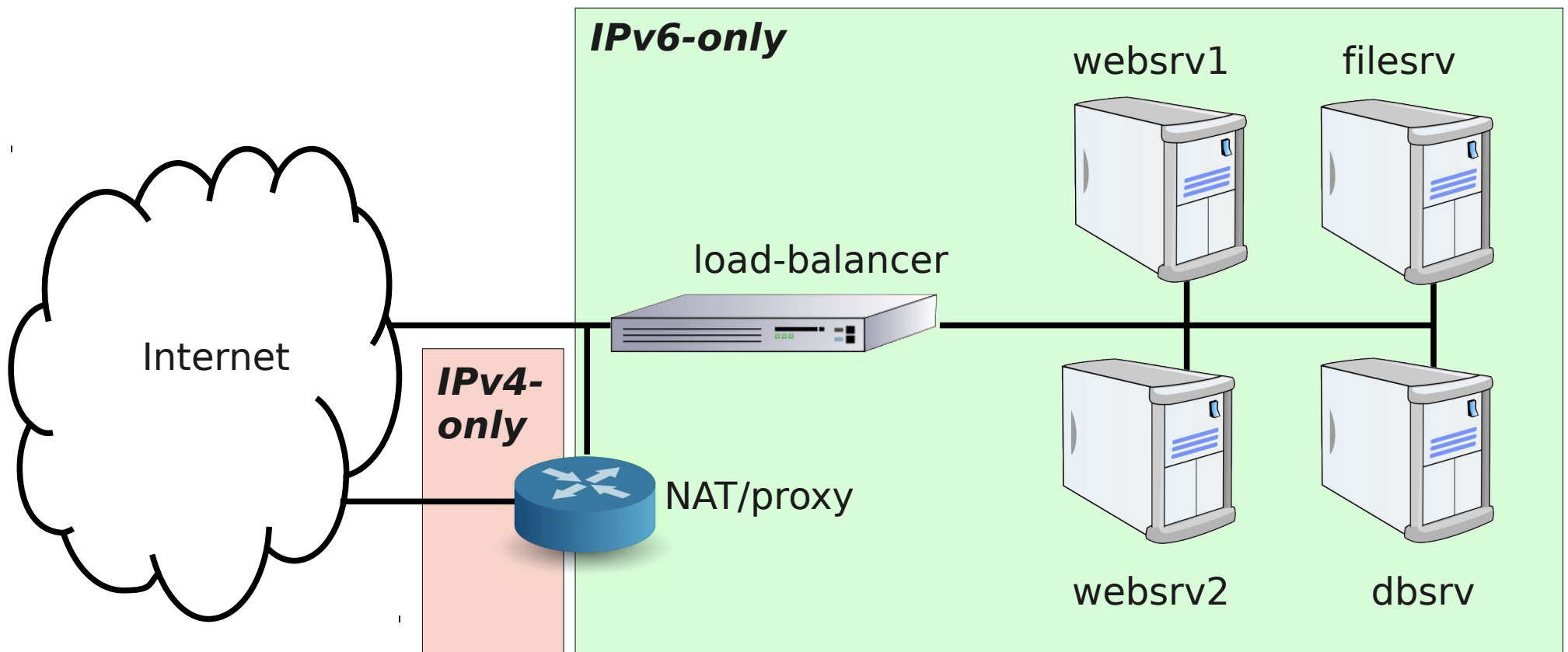
Incremental IPv6 deployment

4) Dual-stacked public front-end, IPv6 BE



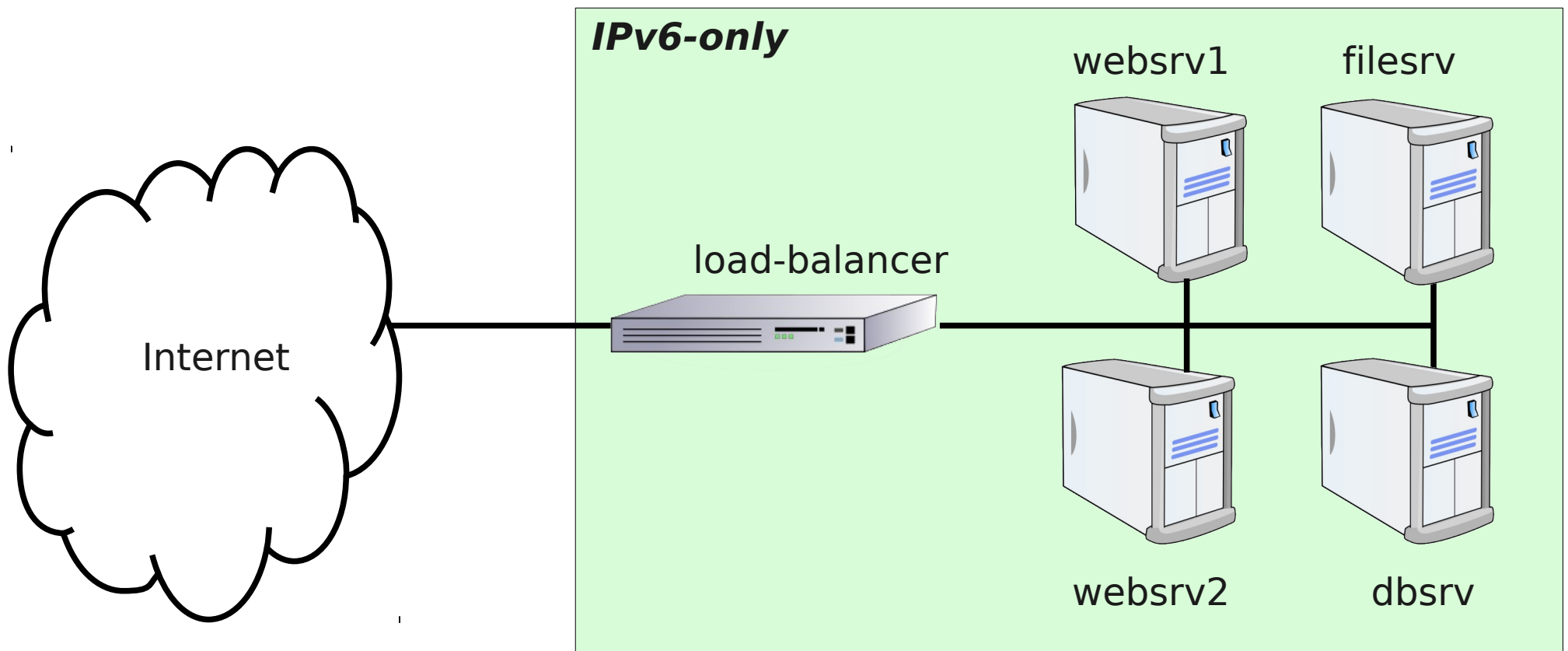
Incremental IPv6 deployment

5) IPv6-only + IPv4 via NAT or proxy



Incremental IPv6 deployment

6) IPv6-only, no IPv4 connectivity at all



Incremental IPv6 in summary

- IPv4-only

- IPv4-only + IPv6 via NAT/proxy

- Dual-stacked public frontend, IPv4 BE

- Full dual-stack

- Dual-stacked public frontend, IPv6 BE

- IPv6-only + IPv4 via NAT/proxy

- IPv6-only

What's possible today?

- IPv4-only

→ IPv4-only + IPv6 via NAT/proxy

→ Dual-stacked public frontend, IPv4 BE

→ Full dual-stack

→ Dual-stacked public frontend, IPv6 BE

→ IPv6-only + IPv4 via NAT/proxy

~~IPv6 only~~

< 1% of end-users world-wide have IPv6!

Let's take a shortcut...

- IPv4-only

~~IPv4-only + IPv6 via NAT/proxy~~

~~Dual-stacked public frontend, IPv4 BE~~

~~Full dual-stack~~

~~Dual-stacked public frontend, IPv6 BE~~

IPv6-only + IPv4 via NAT/proxy

~~IPv6 only~~

Why skip dual-stack?

- DS implies lots of unwanted complexity
 - More ACLs, monitoring, troubleshooting, possible failures, training, documentation, ...
- Saves lots of precious IPv4 addresses
 - 1 IPv4 per service, rather than 1+ per server
- Forces sysadmins to learn and use IPv6
 - They resist change and new technologies
 - Provision dual-stack and they'll use IPv4 only
- Perform a single IPv6 migration project

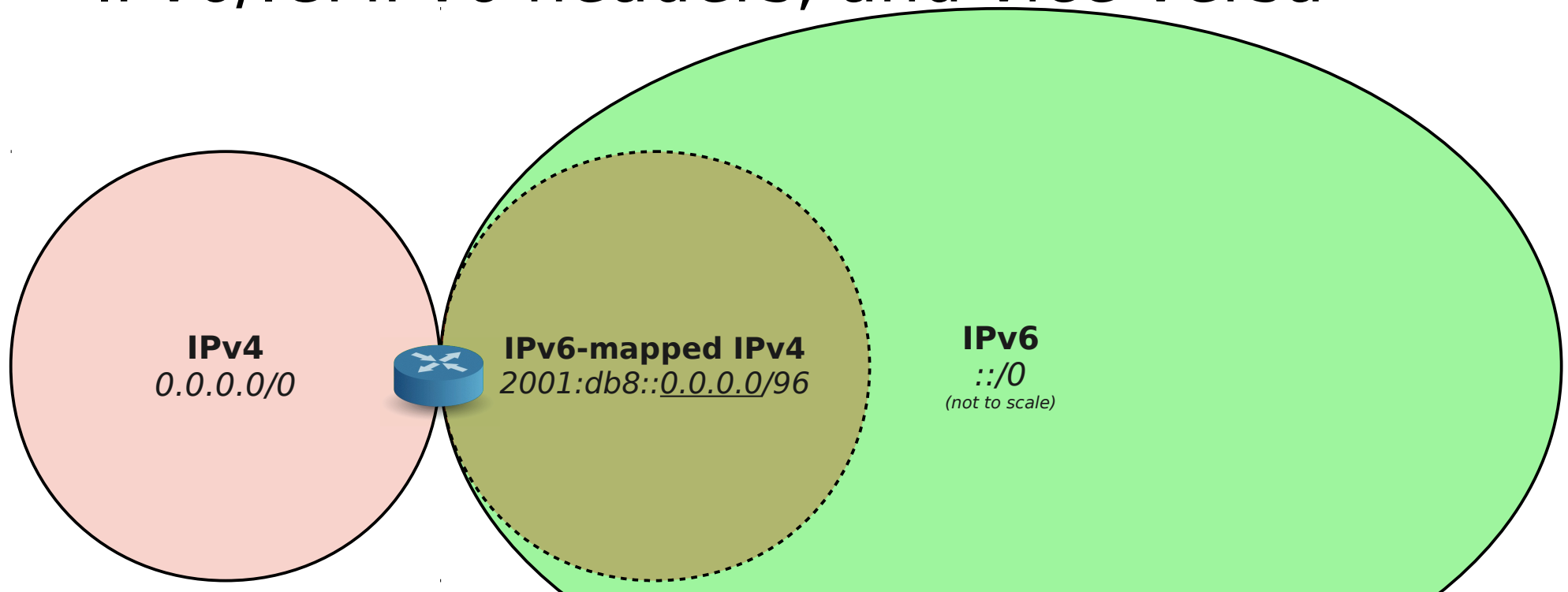
Stateless IP/ICMP Translation (SIIT)

RFCs 6052, 6145

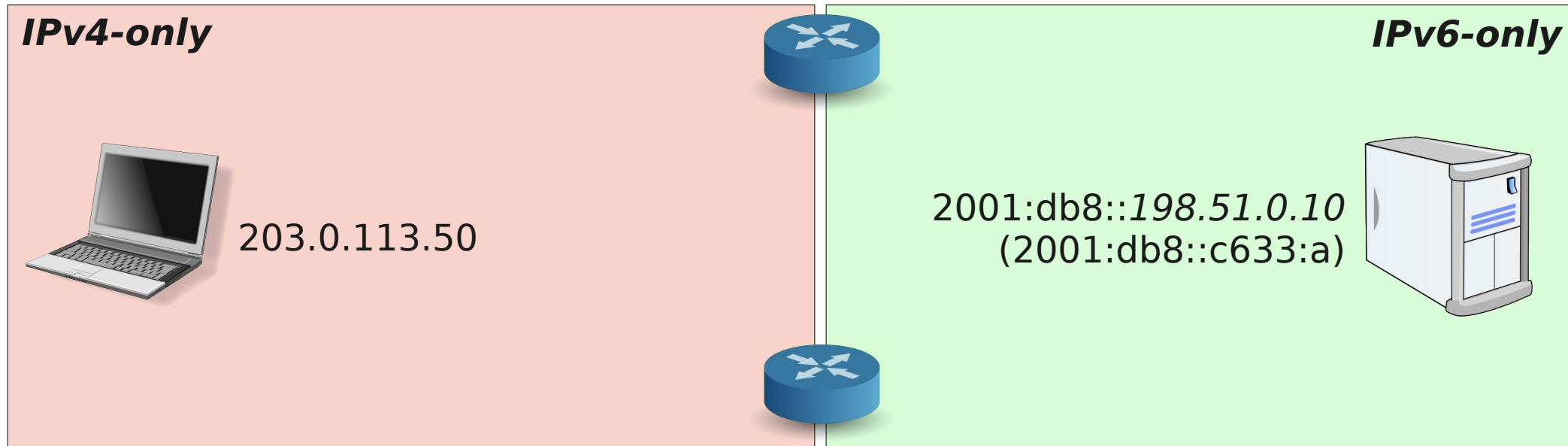
(Also known as Stateless
NAT64 andIVI)

SIIT in a nutshell

- Maps the entire IPv4 address space into an IPv6 prefix from the SP's address space
- Translates IPv4/ICMPv4 headers into IPv6/ICMPv6 headers, and vice versa

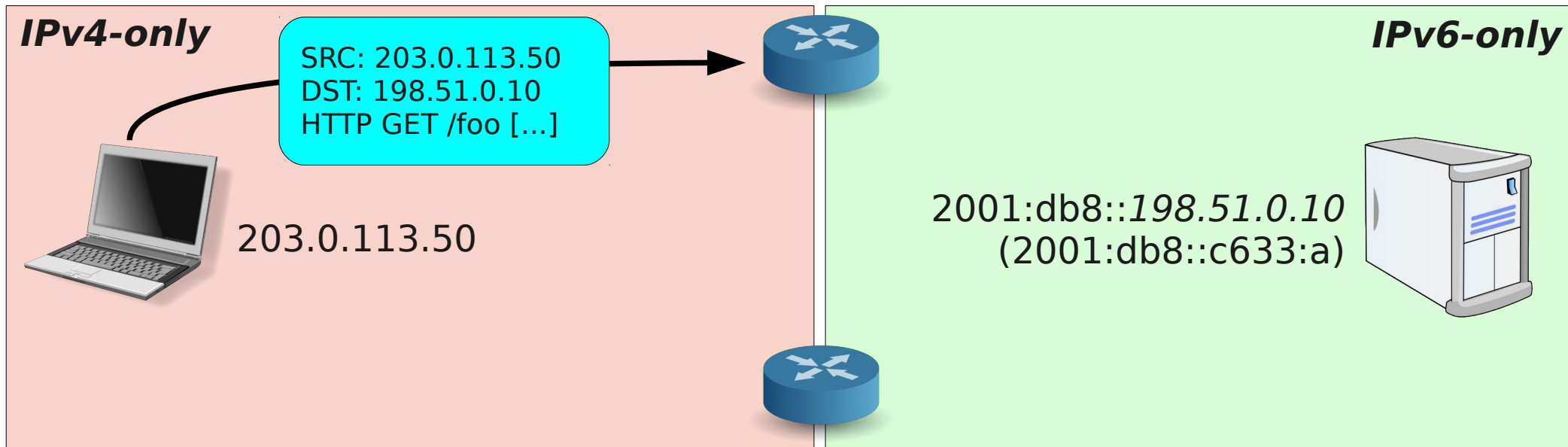


SIIT theory



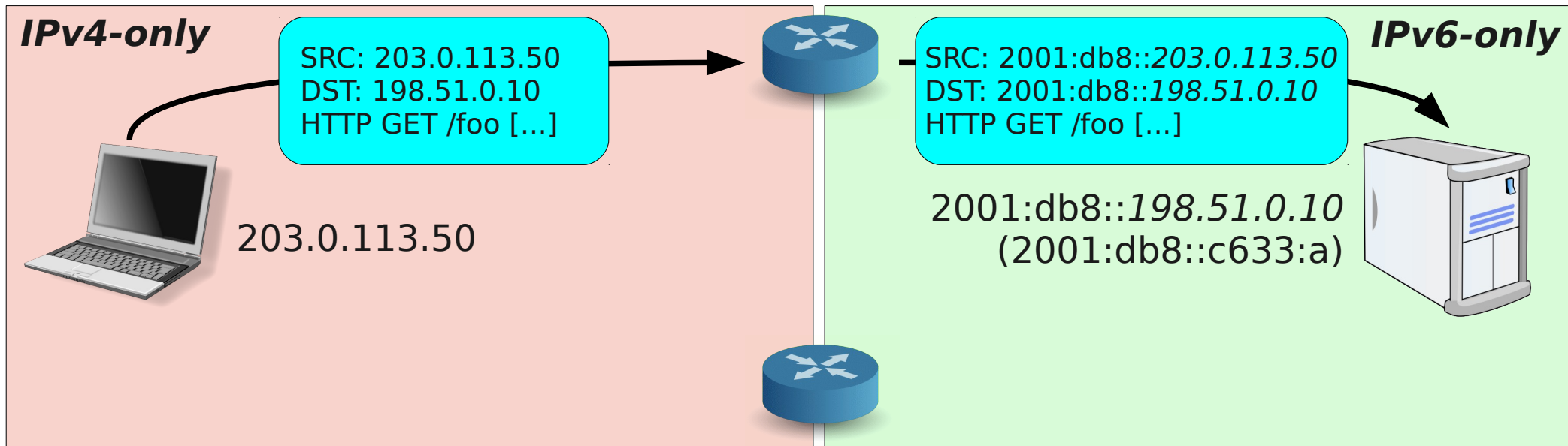
- An IPv4-mapped IPv6 address from a predefined /96 prefix (that represents the IPv4 internet) is configured on the server
- This address is routed to the server using regular IPv6 routing techniques

An IPv4 client connecting



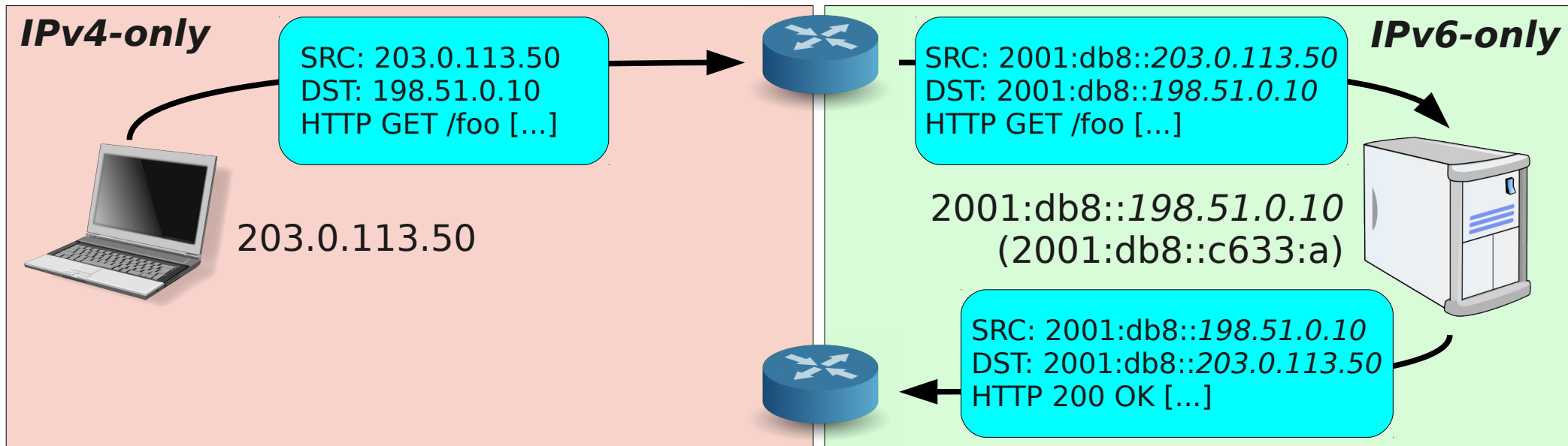
- The IPv4 service address is published as a regular A record for the service in DNS
- It's routed to the provider's SIIT gateways using standard IPv4 routing techniques
- IPv4 clients connect to it in a normal way

IPv4->IPv6 translation



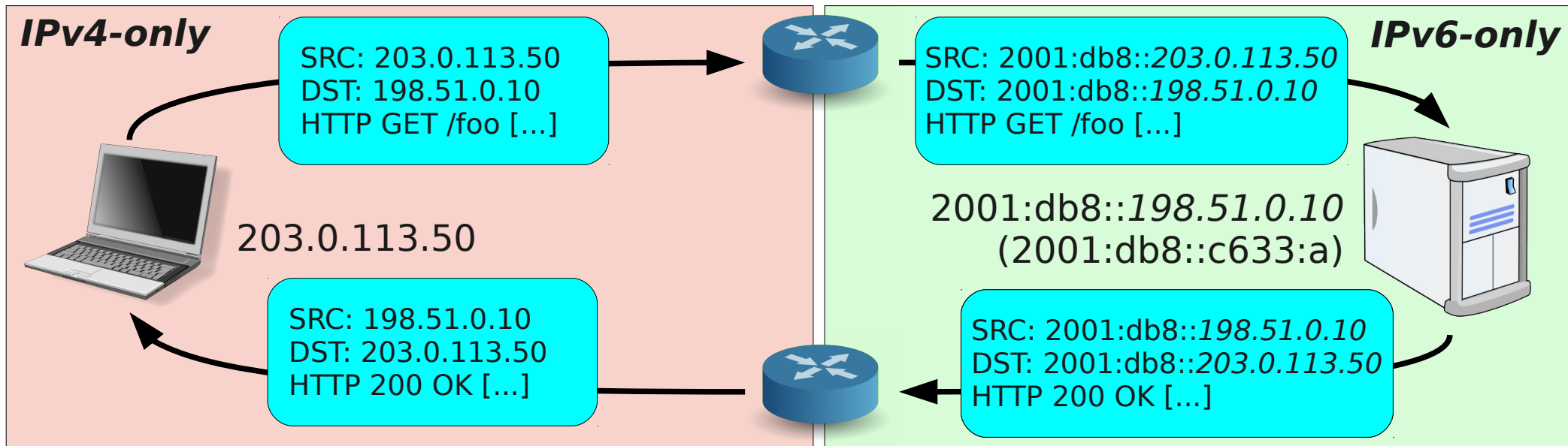
- The pre-defined /96 prefix is prepended to the IPv4 packet's SRC and DST fields
- Layer 4 payload is copied verbatim
- The packet is then routed to the server as a completely ordinary IPv6 packet

IPv6 server processing



- The server responds to the packet just as it would with any other IPv6 packet
- The original IPv4 source address isn't lost
- The /96 prefix (equivalent to the IPv4 default route) is routed to a SIIT gateway

IPv6->IPv4 translation



- The /96 prefix is stripped from the IPv6 packet's SRC and DST fields
- Layer 4 payload is untouched
- The resulting IPv4 packet is returned to the client which processes it normally

SIIT highlights

- Stateless per-packet operation
 - You can use anycast, ECMP load balancing, ...
 - Does not require flows to flow bidirectionally across a single translator
 - Concurrent flow count and fps are irrelevant for performance (unlike NAT44 and proxies)
- The original IPv4 address remains known
 - Applications may geolocate IPv4 users
- Very simple to understand and implement

Applicability

- If the application doesn't work through NAT44, it will likely not work with SIIT
 - e.g., FTP (uses IP literals in Layer 7 payload)
 - implementations might provide ALGs though
- If the application does work with NAT44, it will likely work with SIIT as well
 - e.g., HTTP and HTTPS
- The server software must support IPv6

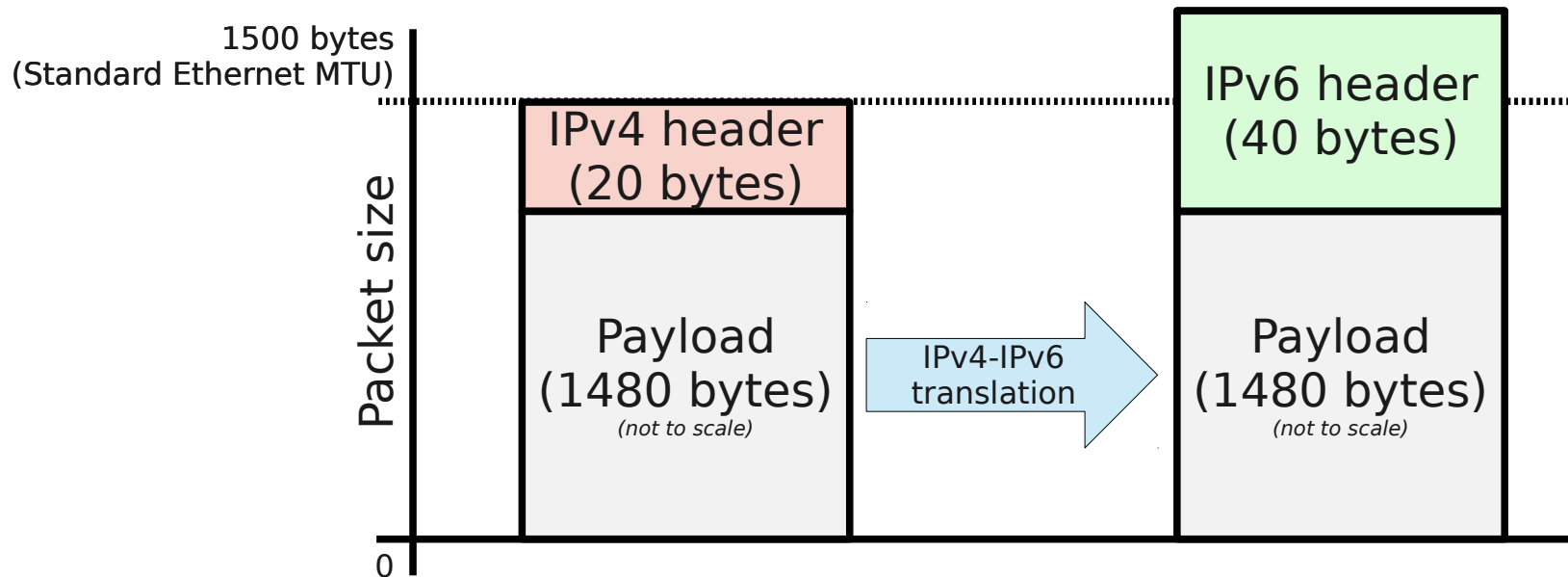
Overriding the address mapping

- With plain SIIT, servers must have the IPv4-embedded IPv6 address assigned
 - Means /128s must be carried in the IPv6 IGP
 - Extra work for the systems administrators
- Better: Statically map IPv4 addresses to arbitrary IPv6 addresses (and vice versa)
 - No extra IPv6 routes or addresses required
 - The sysadmin must only request a public IPv4 frontend for the server's primary IPv6 address
 - Vendor extension (on the Cisco ASR roadmap)

Appendix A

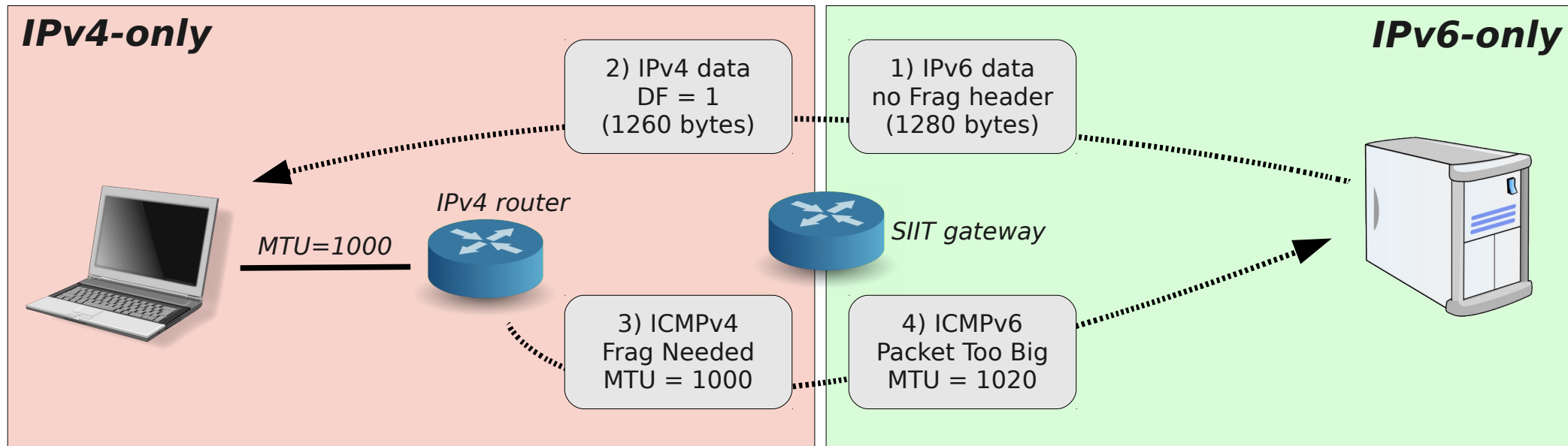
MTU considerations

IP header size difference



- The IPv6 header is (usually) 20 bytes larger than the IPv4 header
- Assuming identical MTUs, full-sized IPv4 packets cannot fit in a single IPv6 packet
- This is generally not a problem for TCP applications, because the IPv6 server will advertise a Maximum Segment Size of 1440 bytes, which in turn limits the TCP Protocol Data Unit to 1460 bytes (and thus the IPv4 packet size to 1480 bytes)
- IPv4 packets larger than 1480 bytes with the Don't Fragment flag set will cause the translator to return an ICMPv4 Need To Fragment advertising a Path MTU of 1480
 - At the time of writing, this is buggy in the Cisco ASR implementation (CSCtw77053)
- Non-DF IPv4 packets >1480 bytes must be split into two IPv6 fragments by the translator
- Fragments can be avoided completely by running the data centre with MTU >= 1520

IPv6 PMTUD with PMTU < 1280



- A link in the IPv4 path towards the client may have an IPv4 MTU smaller than 1260 bytes
 - This translates into an IPv6 Path MTU smaller than 1280 bytes
- According to RFC 2460, an IPv6 node receiving an ICMPv6 Packet Too Big indicating a Path MTU smaller than 1280 bytes has two choices on how to deal with it:
 - a) limit the size of subsequent packets to the indicated Path MTU, or
 - b) limit the size of subsequent packets to exactly 1280 bytes, and also include a Fragmentation header. The Fragmentation header instructs the translator to clear the DF flag in the translated IPv4 packet, and provides the Identification value. This allows the IPv4 router to fragment the packet as it is being forwarded onto the link with the small MTU
- The Linux kernel takes the second approach, but it is at the time of writing buggy:
https://bugzilla.kernel.org/show_bug.cgi?id=42595 & https://bugzilla.kernel.org/show_bug.cgi?id=42572

Appendix B

Cisco ASR1K configuration example

Basic IPv4/IPv6 connectivity

```
!  
interface GigabitEthernet0/0/0  
description IPv4 uplink interface  
ip address 87.238.62.27 255.255.255.254  
nat64 enable  
!  
interface GigabitEthernet0/0/1  
description IPv6 uplink interface  
ipv6 address 2A02:C0:1:102::2/64  
nat64 enable  
nat64 settings mtu minimum 1500  
!  
ip route 0.0.0.0 0.0.0.0 87.238.62.26  
!  
ipv6 route 2A02:C0::46:0:57EE:3D80/121 2A02:C0:1:102::1  
!  
nat64 prefix stateless 2A02:C0:0:0:46::/96  
nat64 settings fragmentation header disable  
nat64 route 87.238.61.128/25 GigabitEthernet0/0/1  
!
```

- The SIIT gateway needs full connectivity to the IPv4 internet
- In this case I use a static default route to the upstream IPv4 router 87.238.62.26 (connected to Gi0/0/0)
 - However, the necessary IPv4 connectivity could just as well have been provided by a default route learned from an interior gateway protocol, or from a full BGP feed
- It also needs to be connected to the provider's IPv6 network
 - A default IPv6 route is not necessary (but doesn't cause any problems either)

Basic SIIT configuration

```
!  
interface GigabitEthernet0/0/0  
  description IPv4 uplink interface  
  ip address 87.238.62.27 255.255.255.254  
nat64 enable  
!  
interface GigabitEthernet0/0/1  
  description IPv6 uplink interface  
  ipv6 address 2A02:C0:1:102::2/64  
nat64 enable  
  nat64 settings mtu minimum 1500  
!  
ip route 0.0.0.0 0.0.0.0 87.238.62.26  
!  
ipv6 route 2A02:C0::46:0:57EE:3D80/121 2A02:C0:1:102::1  
!  
nat64 prefix stateless 2A02:C0:0:0:46::/96  
nat64 settings fragmentation header disable  
nat64 route 87.238.61.128/25 GigabitEthernet0/0/1  
!
```

- NAT64 functionality (which includes stateless translation) is enabled on both the IPv4 and the IPv6 uplink interface
- An IPv6 prefix for stateless translation is chosen and configured on the translator
 - This prefix can have any of the following prefix lengths: /32, /40, /48, /56, /64, or /96 (cf. RFC 6052 section 2.2)
 - When using a /96, the embedded IPv4 addresses will occupy the last 32 bits of the resulting IPv6 address
 - The prefix should be taken from the operator's public IPv6 pool and be globally reachable
 - The reason for this is that the servers will be configured with addresses from within the translation prefix, and could potentially attempt to use them as source addresses when establishing connections to native IPv6 destinations off-net

The IPv4 service address prefix

```
!  
interface GigabitEthernet0/0/0  
  description IPv4 uplink interface  
  ip address 87.238.62.27 255.255.255.254  
  nat64 enable  
!  
interface GigabitEthernet0/0/1  
  description IPv6 uplink interface  
  ipv6 address 2A02:C0:1:102::2/64  
  nat64 enable  
  nat64 settings mtu minimum 1500  
!  
ip route 0.0.0.0 0.0.0.0 87.238.62.26  
!  
ipv6 route 2A02:C0::46:0:57EE:3D80/121 2A02:C0:1:102::1  
!  
nat64 prefix stateless 2A02:C0:0:0:46::/96  
nat64 settings fragmentation header disable  
nat64 route 87.238.61.128/25 GigabitEthernet0/0/1  
!
```

- An IPv4 prefix is allocated from the operator's public address pool (in this example 87.238.61.128/25) and configured as a *nat64 route*
 - The mandatory interface parameter is meaningless unless you're configuring *nat64 prefix* at the interface level, in which case the parameter determines which *nat64 prefix* that gets used
- The prefix contains the IPv4 service addresses IPv4 clients will make connections to (that will in turn be translated to IPv6)
- The gateway must also have a specific IPv6 route to the translated destination prefixes
 - Without it, the translated IPv6 packets would have followed the /96 route and loop back into the NAT64 function
 - In this example I use a static route to 2a02:c0::46:0:57ee:3d80/121 (the translated equivalent of 87.238.61.128/25) to the uplink router on Gi0/0/1, but /128s learned from the IGP would also work

Preventing unnecessary frags 1

```
!  
interface GigabitEthernet0/0/0  
  description IPv4 uplink interface  
  ip address 87.238.62.27 255.255.255.254  
  nat64 enable  
!  
interface GigabitEthernet0/0/1  
  description IPv6 uplink interface  
  ipv6 address 2A02:C0:1:102::2/64  
  nat64 enable  
nat64 settings mtu minimum 1500  
!  
ip route 0.0.0.0 0.0.0.0 87.238.62.26  
!  
ipv6 route 2A02:C0::46:0:57EE:3D80/121 2A02:C0:1:102::1  
!  
nat64 prefix stateless 2A02:C0:0:0:46::/96  
nat64 settings fragmentation header disable  
nat64 route 87.238.61.128/25 GigabitEthernet0/0/1  
!
```

- The RFC takes a conservative approach by default, and assumes that the IPv6 Path MTU is 1280 bytes
 - Inbound IPv4 w/DF=0 packets that would result in IPv6 packets exceeding 1280 bytes in size will instead be split in two fragments (both smaller than 1280)
- In a data centre environment, however, we usually know that IPv6 Path MTU between the translator and the servers is (at least) 1500 bytes, so this behaviour is not necessary
- *nat64 settings mtu minimum 1500* tells the translator that the Path MTU between itself and the IPv6 servers (behind Gi0/0/1) is at least 1500 bytes, this avoiding fragmentation of DF=0 IPv4 packets exceeding 1260 bytes
 - However, IPv4 packets exceeding 1480 bytes will still be fragmented (as they translate into IPv6 packets exceeding 1500 bytes)
- You can avoid all IPv6 fragmentation by increasing both the IPv6 Interface and Path MTU to at least 1520 bytes

Preventing unnecessary frags 2

```
!  
interface GigabitEthernet0/0/0  
  description IPv4 uplink interface  
  ip address 87.238.62.27 255.255.255.254  
  nat64 enable  
!  
interface GigabitEthernet0/0/1  
  description IPv6 uplink interface  
  ipv6 address 2A02:C0:1:102::2/64  
  nat64 enable  
  nat64 settings mtu minimum 1500  
!  
ip route 0.0.0.0 0.0.0.0 87.238.62.26  
!  
ipv6 route 2A02:C0::46:0:57EE:3D80/121 2A02:C0:1:102::1  
!  
nat64 prefix stateless 2A02:C0:0:0:46::/96  
nat64 settings fragmentation header disable  
nat64 route 87.238.61.128/25 GigabitEthernet0/0/1  
!
```

- A translator will by default add an IPv6 Fragmentation header when translating any IPv4 packet with the Don't Fragment flag unset
 - This happens even though the resulting IPv6 packet does not exceed the IPv6 Path MTU, and therefore aren't actually fragmented
 - According to RFC 6145, it is done in order «to indicate that the sender allows fragmentation»
- This behaviour can be avoided by configuring *nat64 settings fragmentation header disable*
 - Note that the translator will still insert Fragmentation headers when it is actually fragmenting, i.e., when it is translating an IPv4 packet which would exceed the configured IPv6 Path MTU

Questions?

Thank you for listening!

Further reading:

RFC 6052 - IPv6 Addressing of IPv4/IPv6 Translators

RFC 6145 - IP/ICMP Translation Algorithm

RFC 6219 - The China Education and Research Network (CERNET) IVI

<http://toreanderson.no> - My personal home page (contact info, social media links, slides from this and earlier talks)

<http://redpill-linpro.com> - My employer and sponsor of this project

Note: IPv4 traffic to both of the above URLs is routed through a SIIT gateway (eating my own dog food)

