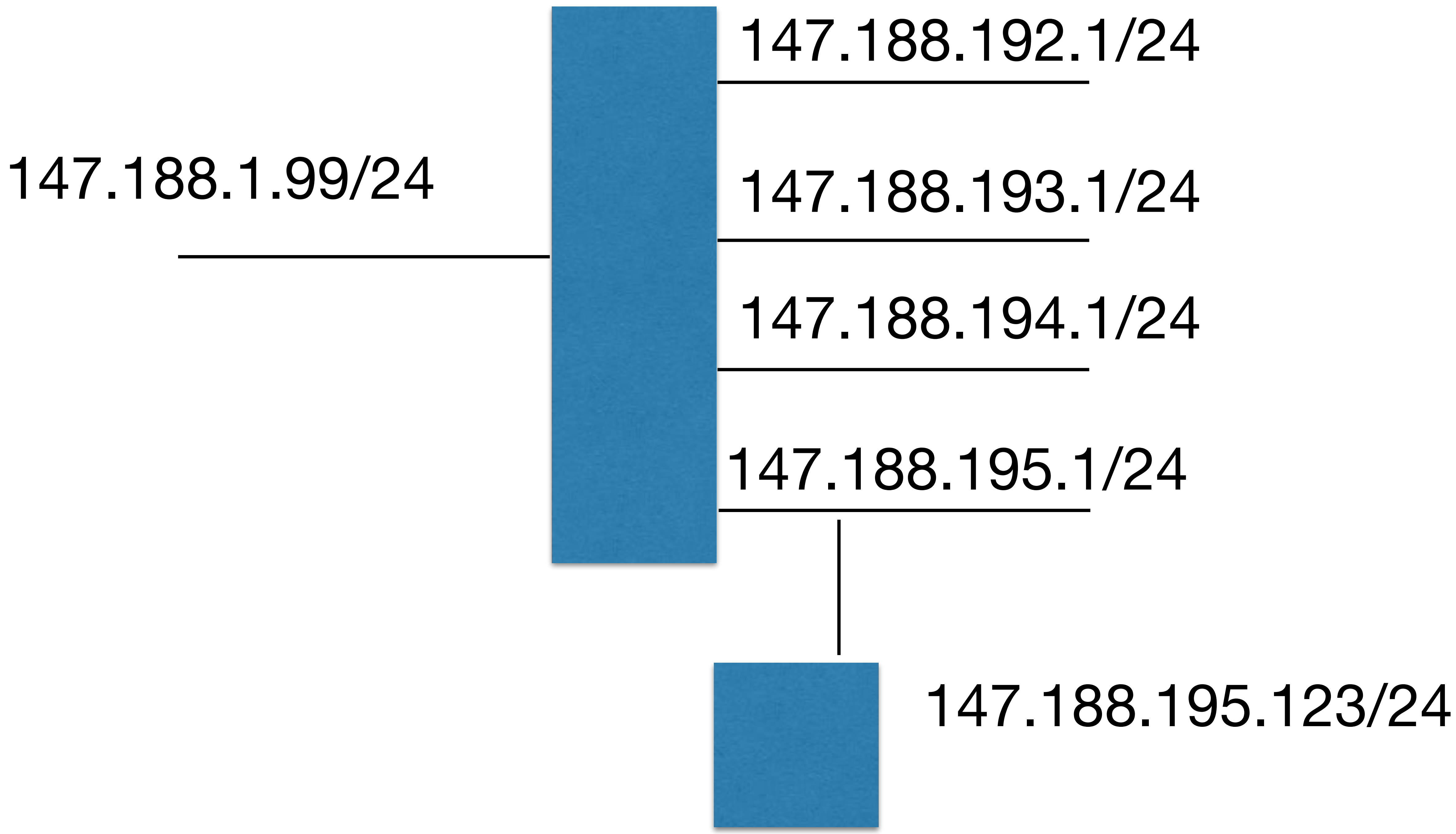
Networks 12: Interfaces, Plural

i.g.batten@bham.ac.uk

Multi-Home, Multi-Path, etc

- Typical desktop machine only has one interface (although virtual networks might be in use).
- Typical server has multiple interfaces
- What are they used for?
    - Lots of things!

Routing


147.188.192.1/24
147.188.1.99/24
147.188.193.1/24
147.188.194.1/24
147.188.195.1/24
147.188.195.123/24

Why do this?

- Historically, LAN routers were very expensive (two interface Cisco IGS/L or 2514 was price of decent Unix workstation) with difficult licensing terms.
- Firewalling and/or routing protocols made this worse (required extra-cost feature sets)
- Unix/Linux box, probably an old one, quite attractive proposition.

Why do this?

- Other reason performance.
- Early 1990s: big fileservers can saturate 10baseT, multiple times
- Late 1990s: big fileservers can saturate 100baseT or FDDI multiple times
- Sometimes need to segregate backup traffic from production traffic

Programming issues?

- Complicates writing servers, and clients need to be aware.
- Usual technique is to bind to INADDR_ANY, with specific port for server and an arbitrary port for a client.
  - You get this for free, as the local address is usually bzero'd and INADDR_ANY is all-zeros
  - Do you want to listen / send on / to all interfaces?
  - How do you ensure packets exit from correct interface?

How are source addresses set?

- Machine has multiple IP numbers, so the source address is chosen either by the programmer or the operating system

- Operating system will, with some variations, choose the IP number of the interface the packet is routed out over, either per-packet (UDP) or per-connection (set at SYN-time)

Clients to multihomed servers

- Clients should cope with servers with multiple interfaces, not all of which are up
- Client should try each interface in turn (doing it in parallel usually considered a bad idea)
- Caching “live” interface considered OK, choose a sensible time-out

Simple Clients

- Naive clients assume that the server they are connecting to only has one IP number, almost certainly IPv4 (gethostbyname is IPv4-only)
- Various DNS hacks mean that there is some “round robin” behaviour, and connections will be to some extent spread over multiple interfaces

```
ians-macbook-air:MSCJAVA14 igb$ dig +short rb2011-1.batten.eu.org
81.2.79.220
81.187.150.214
ians-macbook-air:MSCJAVA14 igb$ dig +short rb2011-1.batten.eu.org
81.187.150.214
81.2.79.220
ians-macbook-air:MSCJAVA14 igb$
```

Slightly Smarter Clients

- Will try all interfaces in order returned by gethostbyname
- Not in parallel: that will establish n connections and then drop n-1, which is abusive

Very Smart Clients

- Use getaddrinfo
  - Confusingly, getaddrinfo is names->addresses, like gethostbyname, getnameinfo is addresses->names, like gethostbyaddr.
- Returns IPv6 and IPv4 addresses
- Probably OK to parallelise IPv4 and IPv6, perhaps with one or other leading slightly, practice will emerge, policy best left to libraries.

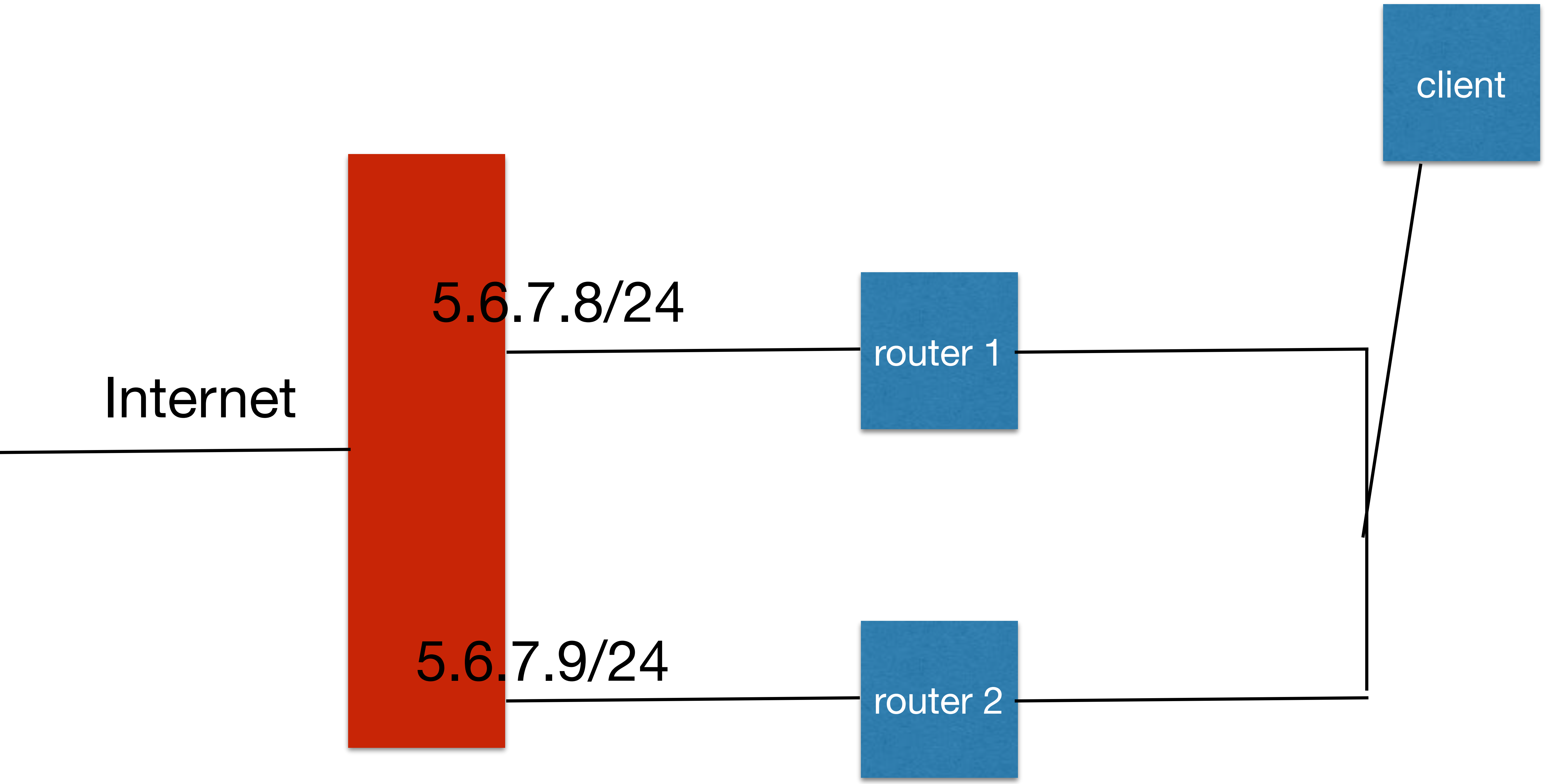
Use getaddrinfo anyway

- gethostbyname and gethostbyaddr are **vile**
- Worst problem is that they maintain static state, and therefore aren’t re-entrant (you can’t use them in threads)
- Re-entrant versions exist (gethostbyname_r, etc) but are messy to use
- getaddrinfo solves all this anyway
- Used correctly means code is IPv6-aware from the outset

Servers

- What about on the server side?

Problem



Asymmetric Routes

- Packets may travel via different routes out and back
- But must cross same NAT point or same stateful firewall with same source address: almost impossible to deal with connections where NAT/FW only sees half of traffic or it is mis-labelled
  - Interesting research topic, incredibly hard to implement with modern speeds and feeds

UDP

- It’s important that if you receive a query sent to a particular IP number, the response goes out from the same IP number
- So query sent to 5.6.7.8:53 must have a reply coming from 5.6.7.8:53, as otherwise sender will discard it (even if a firewall will pass it, which it usually won’t)

UDP

- If routing table changes, then source address of packets to same destination may change

- And you cannot guarantee packets arrive over the interface that a reply will leave over (asymmetric routing)

UDP

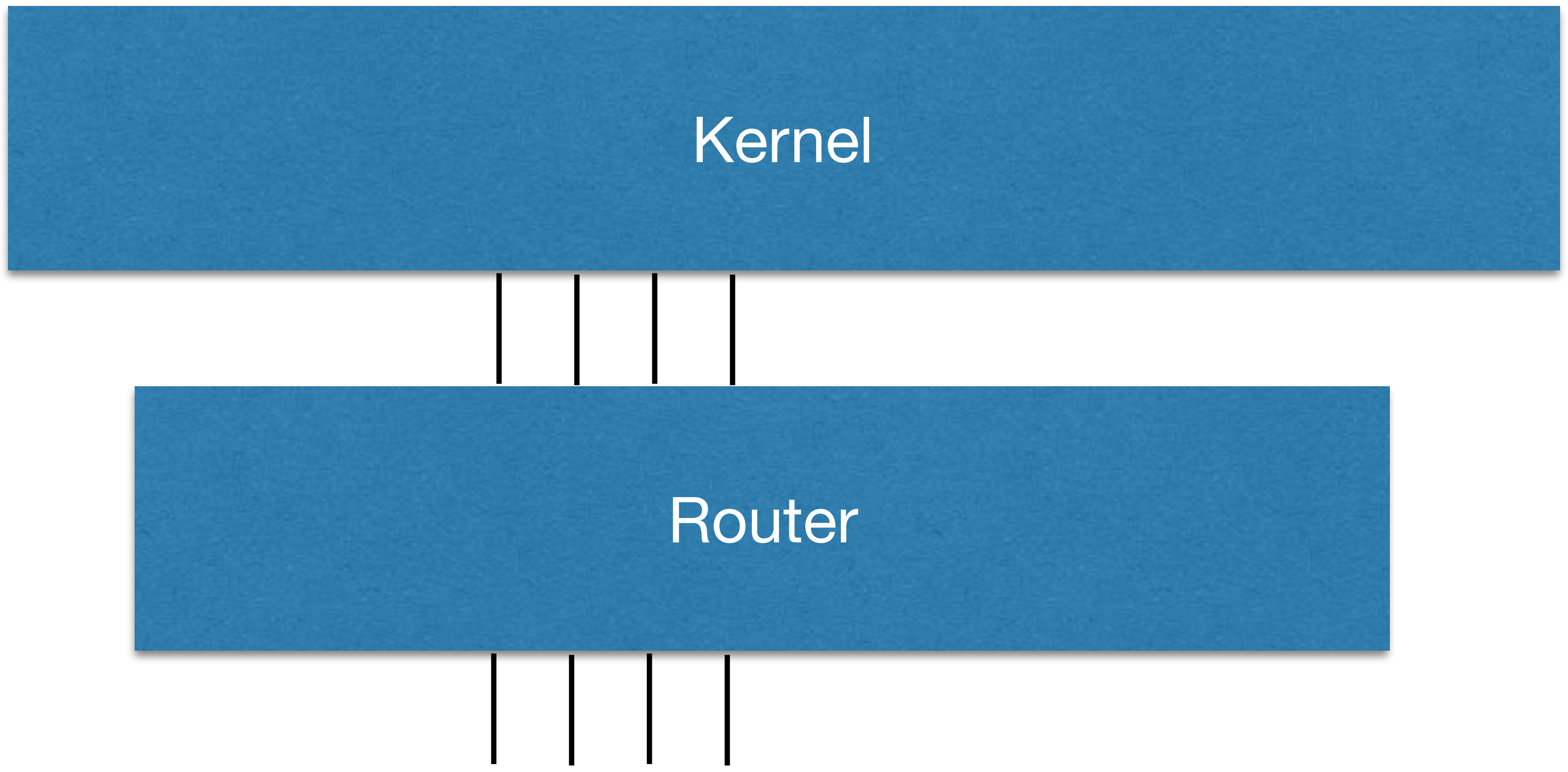
- Two techniques:
  - listen on 0.0.0.0:53, look at destination address on each packet delivered to application, copy destination address into source address of response
  - listen on each interface individually, so you have (effectively) multiple copies of the application, each only knowing about one interface
  - second technique more flexible, but means application needs to iterate over installed interfaces (SIOCGIFCONF is notoriously painful to use) and will need to respond to changes to interface config.

Internal Routing

- You can set the source address to **any** local addresses, and the kernel will still route the packet out over the interface it believes to be closest to the destination

  - In fact, most operating systems will let you set UDP source addresses to any address, without checking it is local. You might need to be root, but you had to be root to bind to port<1024 anyway.

Conceptually…



Forwarding Switch

- All operating systems now have a switch to determine if packets are forward between interfaces
    - Used to be default on for anything with more than one interface, now always default off
- Behaviour if you try to send packet with source address X to a destination routed via interface Y with forwarding turned off slightly unclear
    - but in practice, switch only applies to non-local packets arriving over interfaces

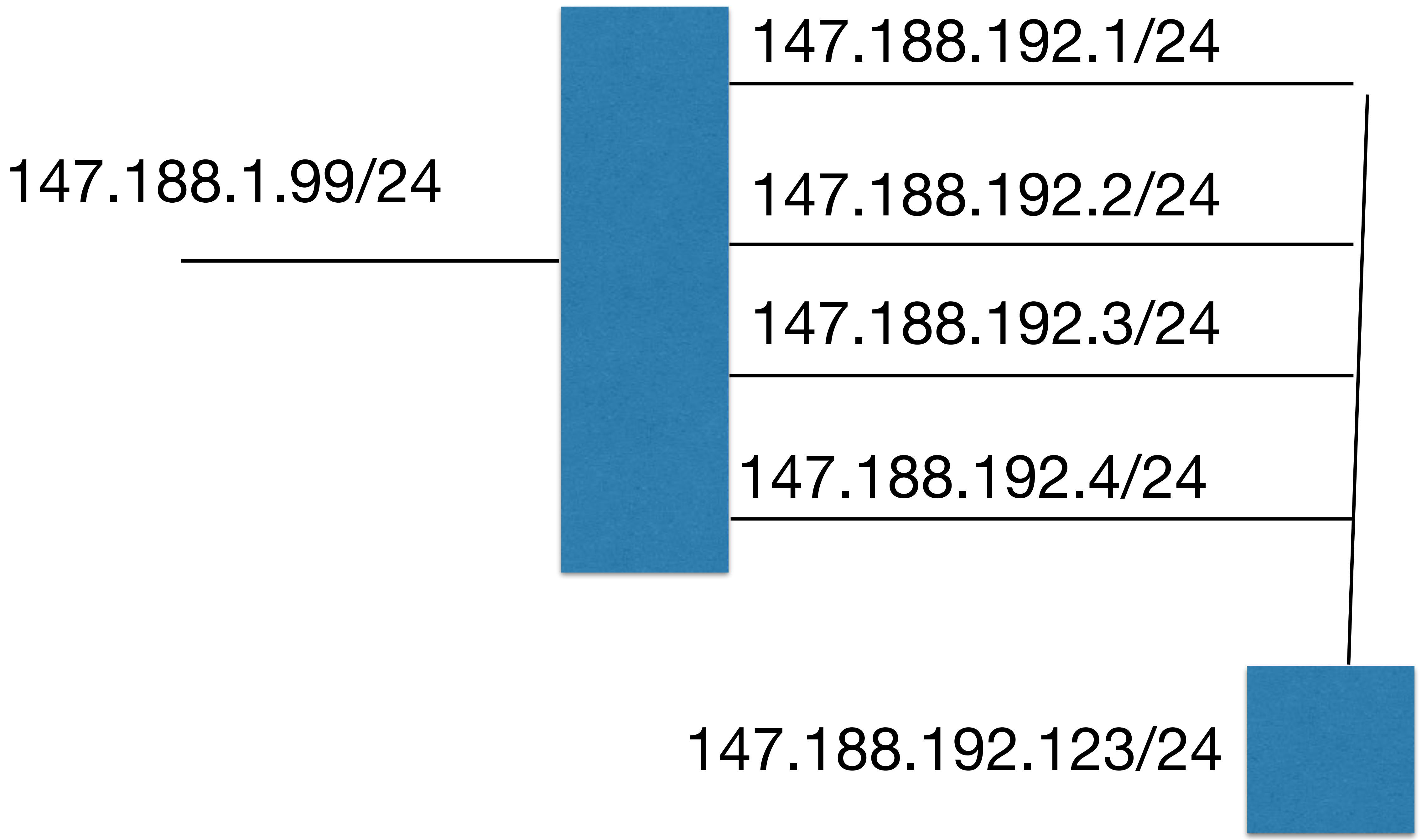
TCP

- Kernel ensures that replies have same source address as original destination address (definition of TCP)
- So much easier to just listen on 0.0.0.0: reasons to listen on specific addresses more about security than connectivity
- No control over which interface packets leave over

Multihomed clients

- Might need to explicitly set a source address: actually trickier than server for TCP
- Usually config option: by default bind outgoing requests to INADDR_ANY and let operating system decide, but give administrator option on a per-application basis

Interfaces into **same** network



Similar problems…

- Usually done for performance or reliability reasons
- Better techniques available if switches support it (see later)
- Usually one IP number is treated as primary, and used for all source addresses, with packets spread out over interfaces.
- Other interfaces might listen, but are deprecated (some operating systems have DEPRECATED state, which means address is never used as source)

Useful for transition

```
mailprod0: flags=100001000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,PHYSRUNNING> mtu 1500 index 2
        inet 147.188.192.250 netmask ffffff00 broadcast 147.188.192.255
mailprod0:1: flags=100001000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,PHYSRUNNING> mtu 1500 index 2
        inet 147.188.192.251 netmask ffffff00 broadcast 147.188.192.255
```

147.188.192.251 address of older system, now decommissioned: doesn’t need physical interface

Link-Agg

- If switch supports it, you can put interfaces into a Link Aggregation group, controlled with LACP
- Appears to operating system as one “**logical**” interface, one IP number, one entry in routing table, etc.
- Modern solution for performance and availability
  - Failure detection much faster and simpler
  - Fancy equipment lets you link-agg to multiple switches, or at least multiple independent cards in same switch

Benefit of LinkAgg

- Only one IP number
- Fast failover
- Efficient multiplexing and load spreading (header hashes, sometimes done by hardware)

Problem of LinkAgg

- Only goes as far as your nearest switch
- Doesn’t provide for (say) spreading over multiple internet connections

VLANs

- Way to get multiple networks delivered over single cable
- Ethernet frames have optional extra “tag” (0–1023 minimum, 0–4095 more common) identifying which network it is
- Untagged frames are in “default” network
- OS sees them as two separate networks

VLAN host-side

Untagged

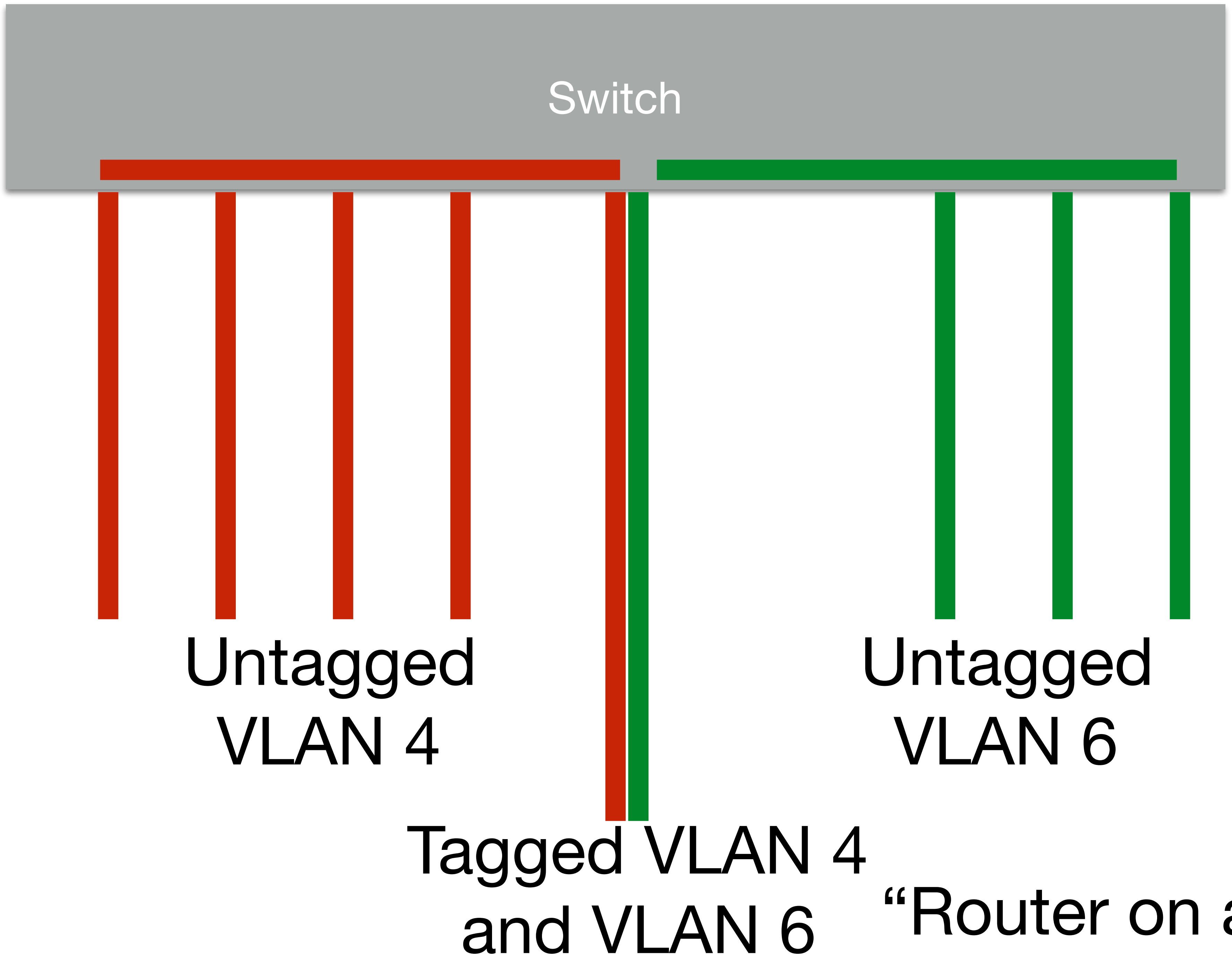
```
igb@pi-one:~$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr b8:27:eb:8c:0c:34
          inet addr:10.92.213.231  Bcast:10.92.213.255  Mask:255.255.255.0
          inet6 addr: 2001:8b0:129f:a90f:3141:5926:5359:1/64 Scope:Global
          inet6 addr: fe80::ba27:ebff:fe8c:c34/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:44938697 errors:0 dropped:0 overruns:0 frame:0
          TX packets:33296383 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:49825123 (47.5 MiB)  TX bytes:441607425 (421.1 MiB)

eth0.5    Link encap:Ethernet  HWaddr b8:27:eb:8c:0c:34
          inet addr:81.187.150.211  Bcast:81.187.150.223  Mask:255.255.255.240
          inet6 addr: 2001:8b0:129f:a90e:3141:5926:5359:1/64 Scope:Global
          inet6 addr: fe80::ba27:ebff:fe8c:c34/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:25759488 errors:0 dropped:0 overruns:0 frame:0
          TX packets:21680755 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2927238990 (2.7 GiB)  TX bytes:2669983251 (2.4 GiB)
```

VLAN tag = 5

VLAN on switches

- Each VLAN has a tag
- Each port of a switch can output one or more VLANs, one of them optionally untagged
- Flexible, and a good way to confuse yourself
- Ports outputting untagged frames sometimes called “access”, tagged frames “trunk”
- Can mix tagged and untagged on one cable, untagged packets assumed to be VLAN 0

Logically divide switches



“Router on a stick”

Untagged ports

- AKA “access ports”
- Devices attached to them don’t have to understand VLANs
- Packets from one VLAN on the switch are transmitted with the tag stripped
- Packets arriving without a tag have a pre-defined tag added
- Can also have tagged packets mixed in, to/from VLAN-aware devices
  - Freshly installing a machine to use a tagged interface is sometimes tricky, so attractive to use untagged for “primary” interface while still having access to other VLANs, cf. my home example

Tagged Ports

- All packets are transmitted with a tag, and only tagged packets are expected to be received
- Common for inter-switch links
- Tags can be filtered for security reasons (not very good security)
- Tags can also be re-written, although that way madness lies

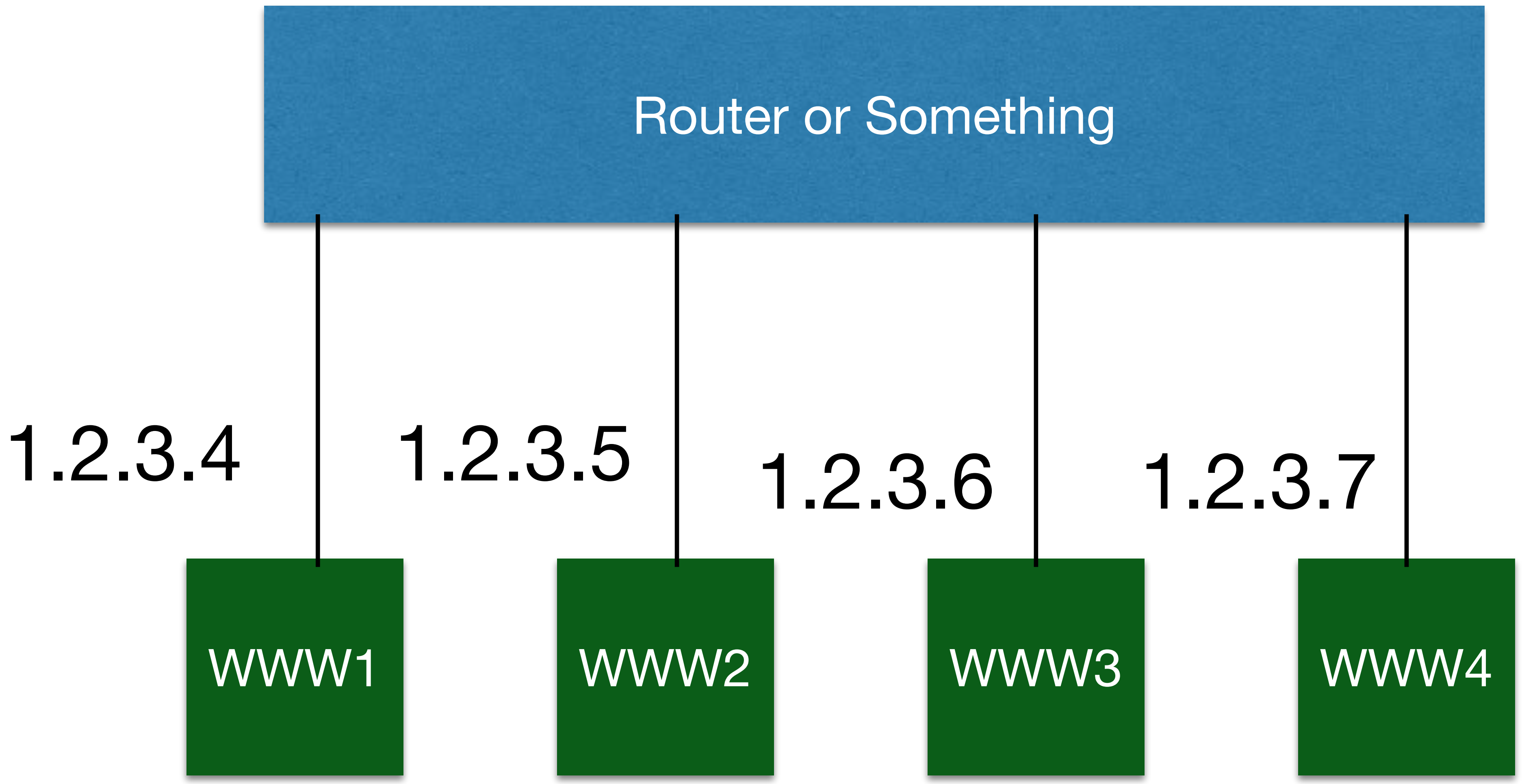
VLAN security

- Quite weak
- Akin to writing “private” on an envelope but not sticking the flap down
- Splitting a switch with untagged ports might be OK, as might inter-switch links, but in general using VLANs for anything involving hard separation will fail assurance
    - Anyone clever enough to use VLANs for this is clever enough to not use VLANs for this
    - All security fails in face of attacker who can reconfigure switch, and most switches aren’t very secure

Link-Agg + VLANs

- You can deliver two or more networks over two or more cables, with full load balancing and failover
- Aggregate using Link Aggregation
- Then apply VLAN tagging for the separate networks
- Not enough people do this: it is very, very effective

Load Balancing



www.dom.ain -> four addresses

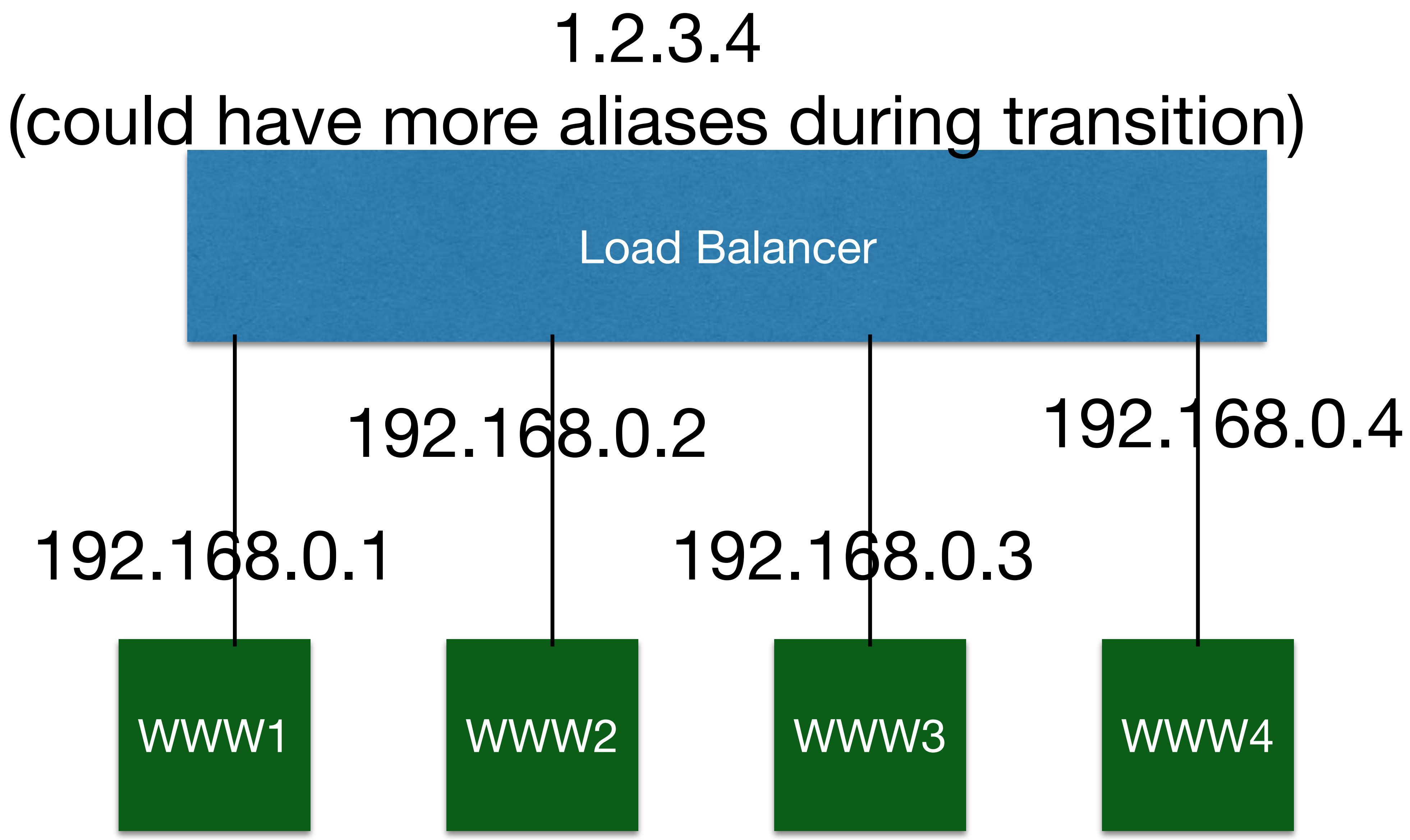
Naive Approach

- Treat as one machine with four addresses
- Rely on DNS round-robins to deal with balancing
    - Very rough balance
    - Poor tolerance of failure if clients don’t try other addresses, slow tolerance of failure anyway

Load Balancers

- Inbound NAT
- Constantly ping devices and/or count live connections
- Sends incoming connection to least loaded, active machine
  - Can also use weighted balances, magic protocols to report load, measures of bandwidth, etc.
- Service lives on **one** IP number, balanced over multiple servers

Load Balance



<u>www.dom.ain</u> -> one address

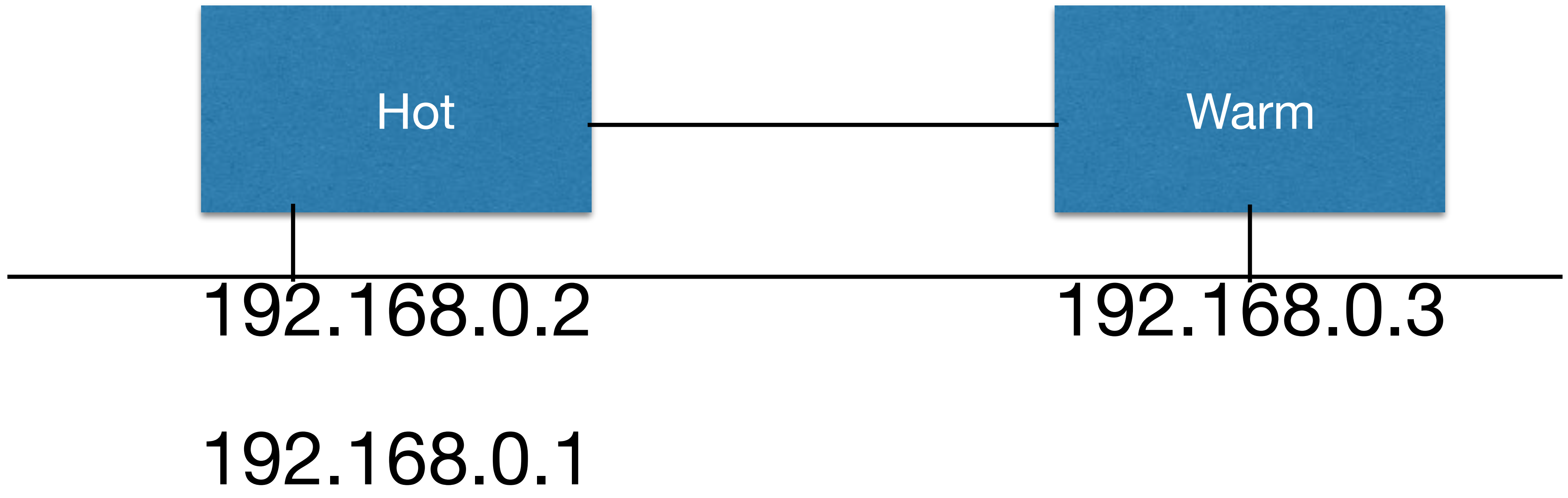
Load Balancing HTTP and HTTPS

- Sometimes needs slightly more than Inbound NAT, as connections to servers need to be slightly “sticky” — successive TCP connections from same source need to go to same destination because of session cookies and so on.
- Load balancer stores source of old sessions, uses same server for subsequent requests from same source.
    - Also a problem for load-spread web caches, but those are much less common in 2017

VRRP

- What about routers, firewalls, etc which need to see all traffic?
- Can’t easily load balance, but can do failover

VRRP



VRRP

- Advertise shared IP number as real service address
    - Router for DHCP, IMAP service for IMAP, etc
- Mutual monitoring and takeover of shared address
- Requires multiple redundant networks between peers, as “split brain” situation potentially quite nasty

So why Multipath TCP?

- Solves some of this
  - In 2017, presumption that new protocols will be TCP-based
  - Many traditional UDP protocols have TCP analogues (DNS, NFS, sadly not NTP) which work better anyway

Multipath TCP

- Establish link from any interface to any other interface
- Hosts exchange information about other interfaces they have, which might be in the DNS, or might be a little more private
- Hosts can then attempt to make more connections
- Kernel marshals them as required
- Applications only see the original connection
- Will handle IPv4 and IPv6 on single connection
    - Still needs application to try both initially to deal with single-stacked case

Summary

- Multi homing is complex
- Applications should deal with multiple interfaces, but often don’t and “deal with” isn’t a simple definition
- Load balancing and failover is complex but beneficial
- Multipath TCP solves some of the issues
- Use getaddrinfo () / getnameinfo () in all new code