IN2140:
Introduction to Operating Systems and Data Communication

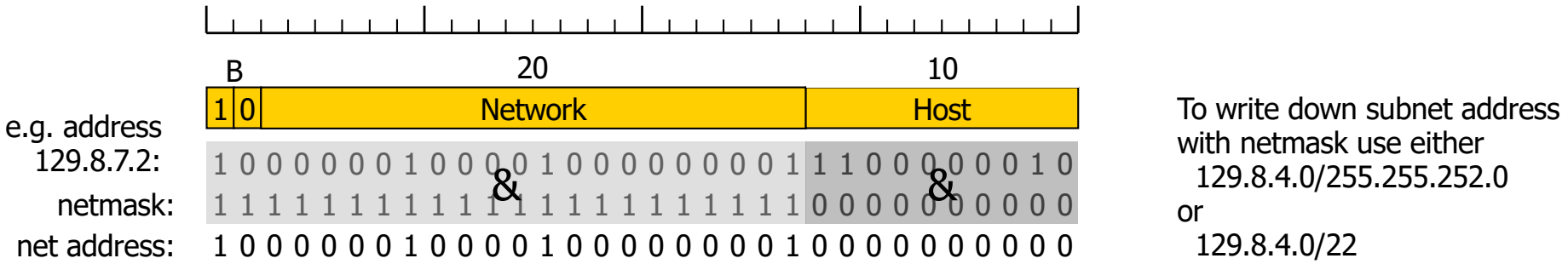


Addressing in the TCP/IP model

Layer 3 Addressing: IPv6 addresses

CIDR: Classless InterDomain Routing

- Idea
  - local decision for subdividing host share into network portion and end system portion



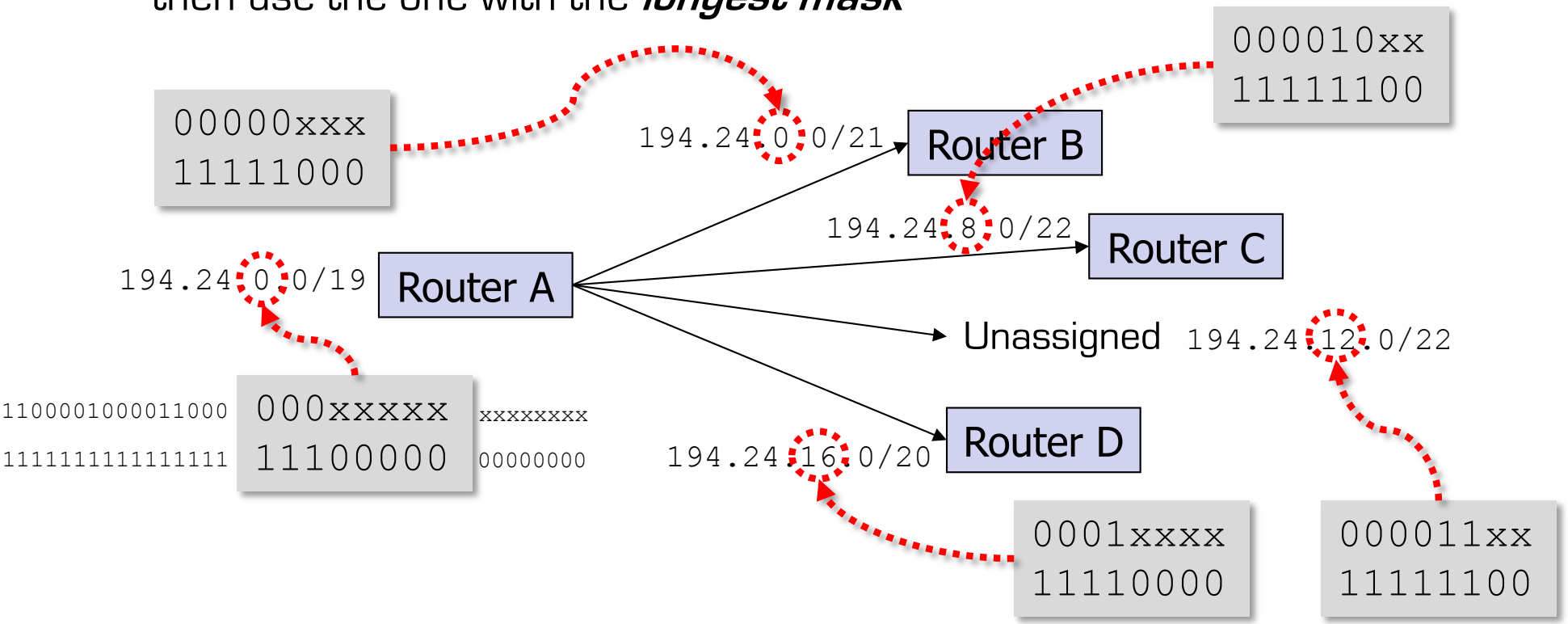
To write down subnet address with netmask use either
129.8.4.0/255.255.252.0
or
129.8.4.0/22

- Use "netmask" to distinguish network part from host part

- Routing with 3 levels of hierarchy
  - **end system**: compute "`dst addr & netmask == subnet addr`"
    - `TRUE`: packet to local end system (perhaps ARP, then deliver packet)
    - `FALSE`: packet to another network (send to this network's router)
  - **router**: compute "`dst addr & netmask == subnet addr`"
    - `TRUE`: packet to local end system (perhaps ARP, then deliver packet)
    - `FALSE`: packet to another network (look up in routing table, send to other router)

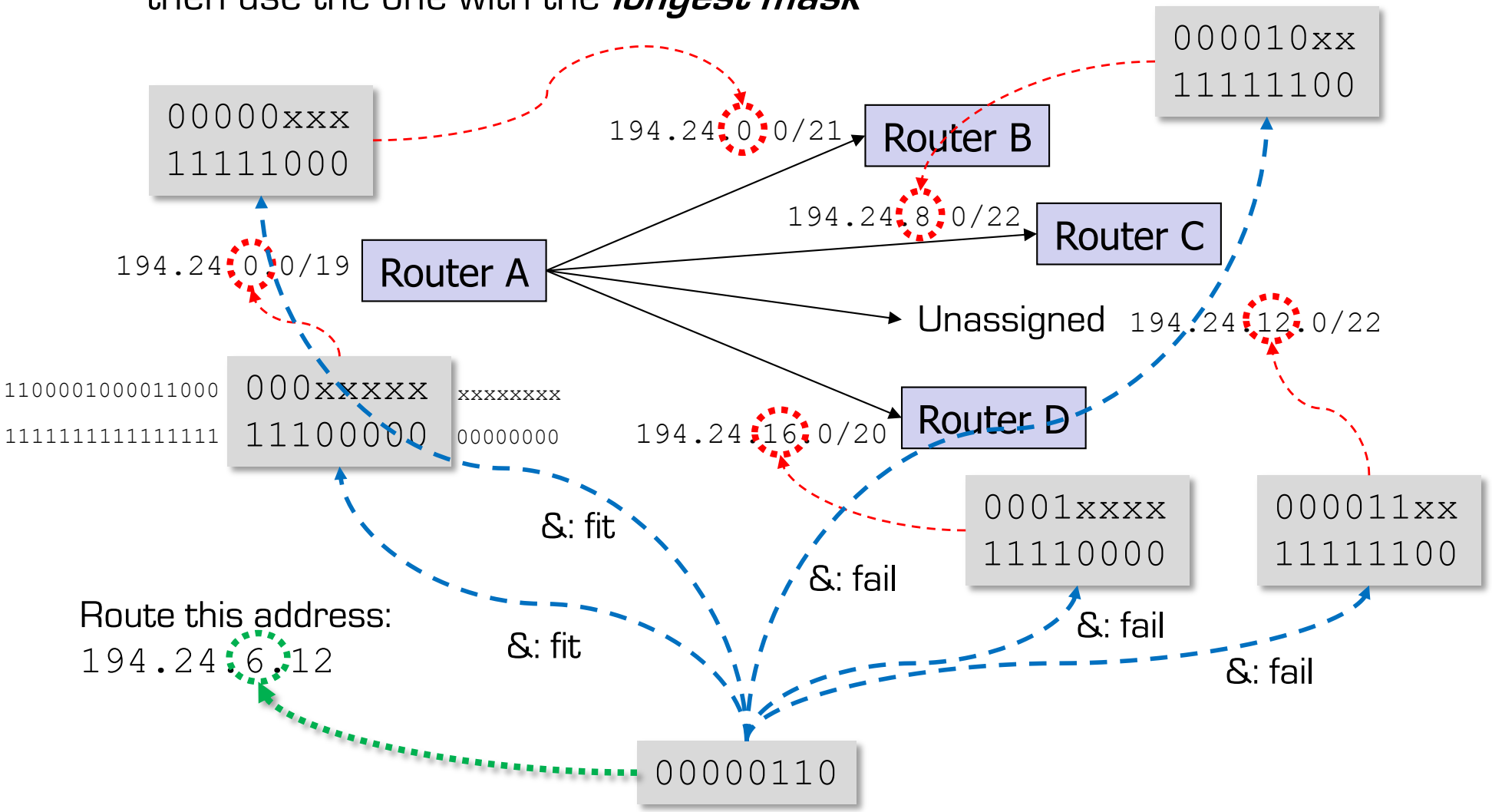
CIDR: Classless InterDomain Routing

- Rule of the longest match
  - if several entries with different subnet mask length may match then use the one with the ***longest mask***



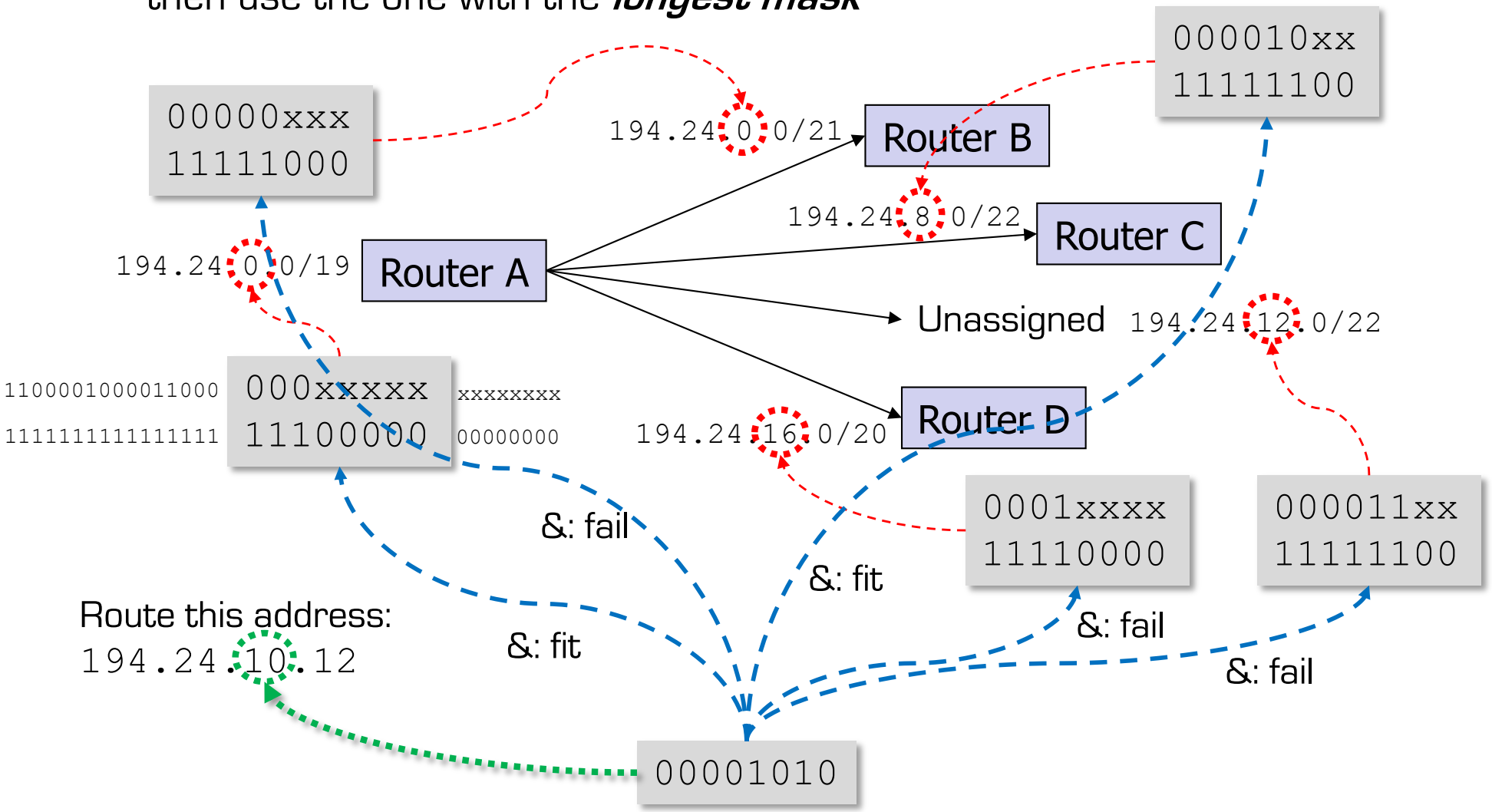
CIDR: Classless InterDomain Routing

- Rule of the longest match
  - if several entries with different subnet mask length may match then use the one with the ***longest mask***



CIDR: Classless InterDomain Routing

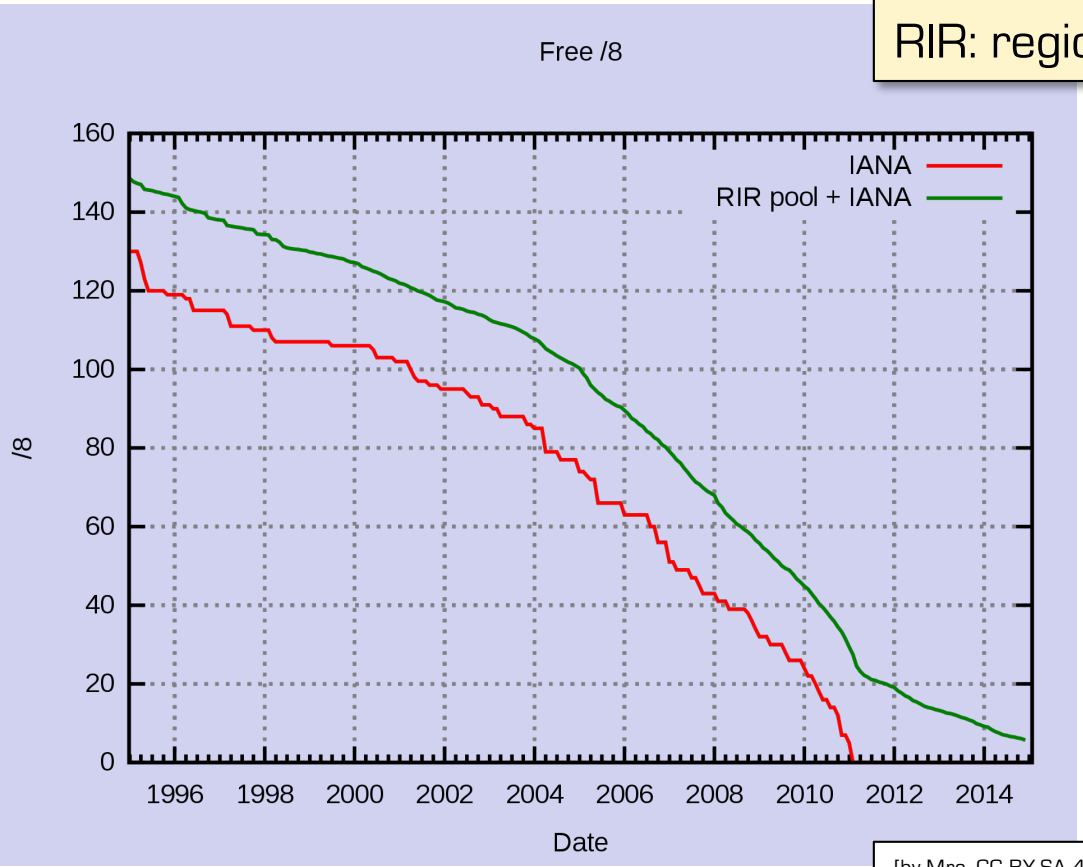
- Rule of the longest match
  - if several entries with different subnet mask length may match then use the one with the ***longest mask***



IP Version 6 (IPv6)

- Motivation for IPv6: problems with IPv4
  - Too few addresses
  - Bad support for QoS
  - Bad support for mobility
  - Many other shortcomings …

IANA: Internet assigned numbers authority
RIR: regional Internet registry



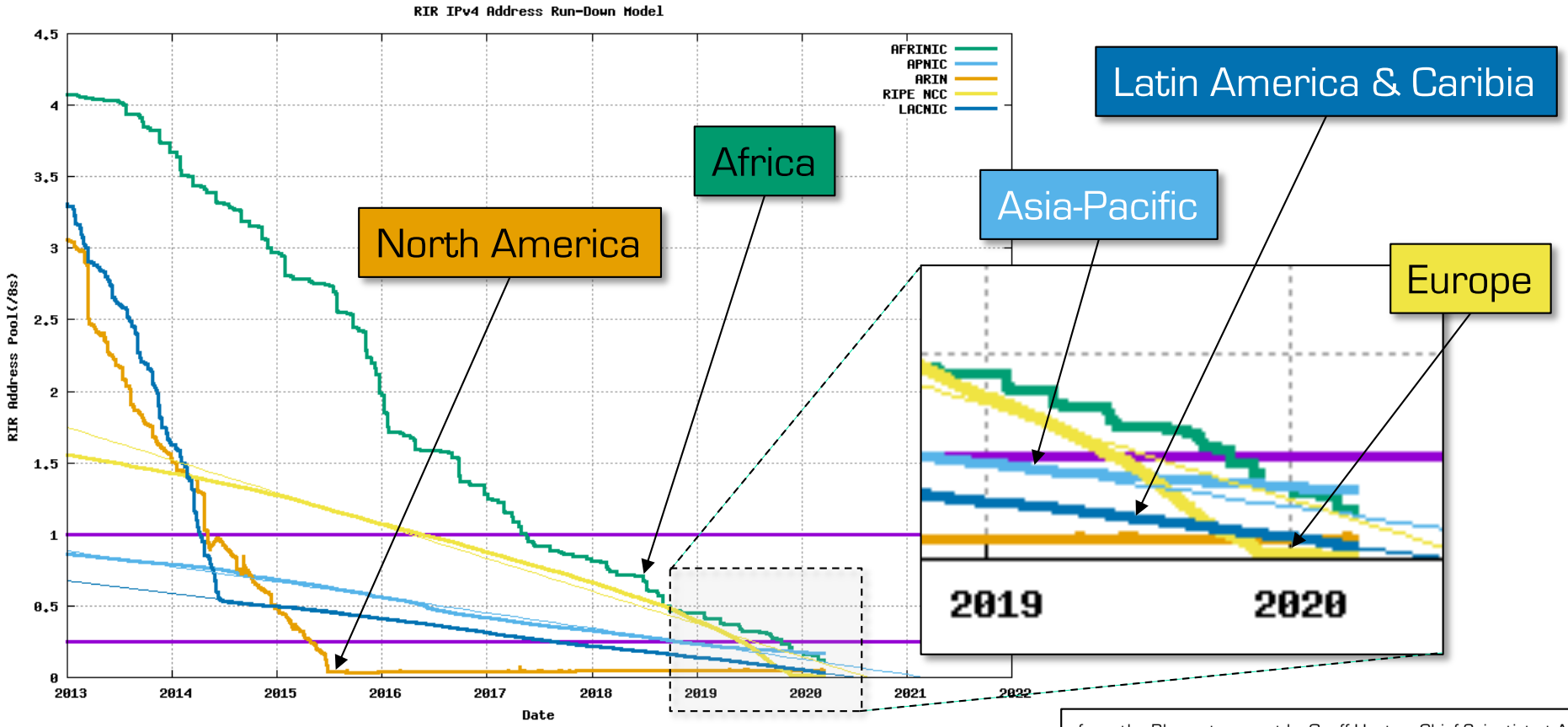
Example consequences:

- no IP addresses for individuals
- large-scale sharing of Internet addresses in local networks using NAT
- Microsoft using addresses from RIR LACNIC (Latin America & Caribbean NIC) for Cloud nodes in North America

[by Mro, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=10593349]

IP Version 6 (IPv6)

- Motivation for IPv6: problems with IPv4
  - Too few addresses
  - Bad support for QoS
  - Bad support for mobility
  - Many other shortcomings ...



from the Blog potaroo.net by Geoff Huston, Chief Scientist at APNIC

IPv6 Objectives

- To support billions of end systems
  - longer addresses
- To reduce routing tables
- To simplify protocol processing
  - simplified header

Scalability

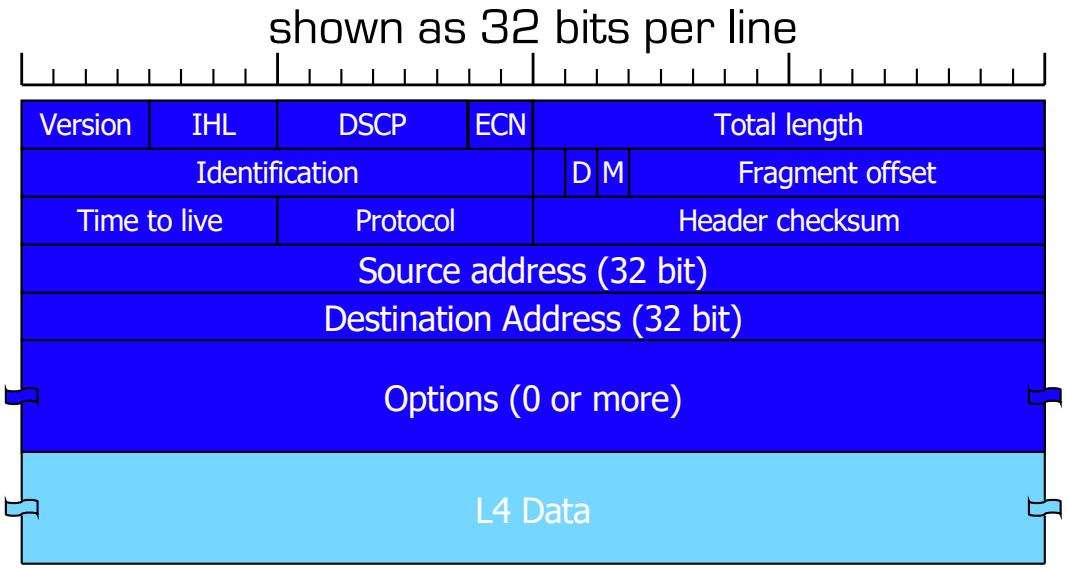
- To increase security
  - security integrated
- To support real-time data traffic
  - flow label, traffic class
- To provide multicasting
- To support mobility (roaming)

Addressing IPv4 limitations

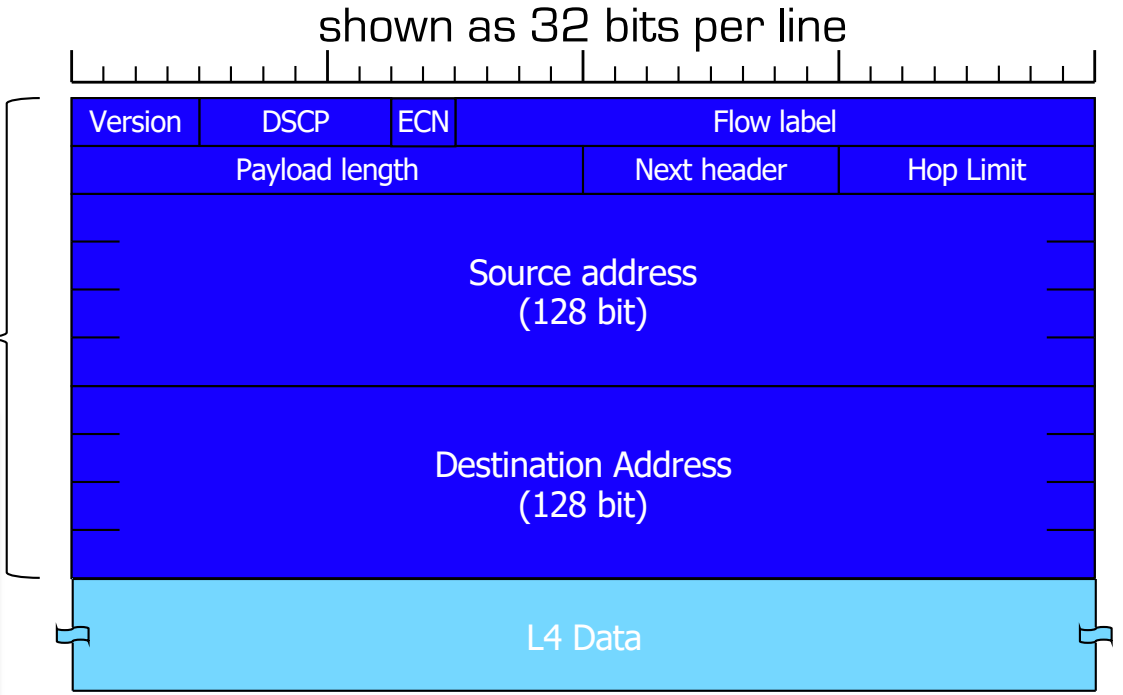
- To be open for change (future)
  - extension headers
- To coexist with existing protocols

Coexistance

IPv4 and IPv6



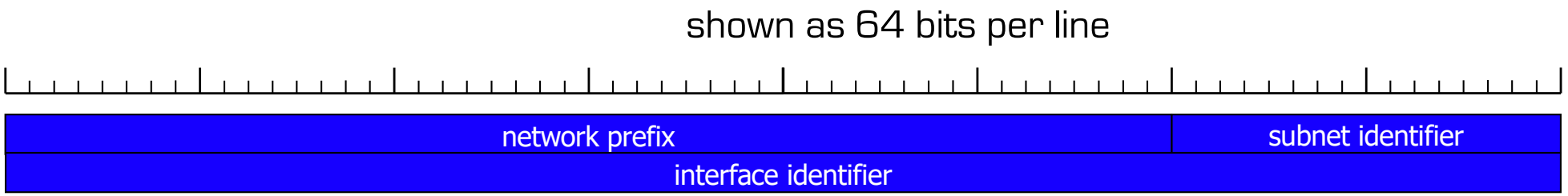
New IPv6 header
- is larger but simpler



- packet can never be fragmented, now an L4 task
- options are now payload (data)
- checksum is now an L2/L4 task

IPv6 addresses

- example of the IPv6 address spaces



a typical routed address

```
0010101000000000000101000101000001000000000011110000100000001010
0000000000000000000000000000000000000000000000000000000100000100
```

```
2a 00:14 50:40 0f:08 0a:00 00:00 00:00 00:20 04
```

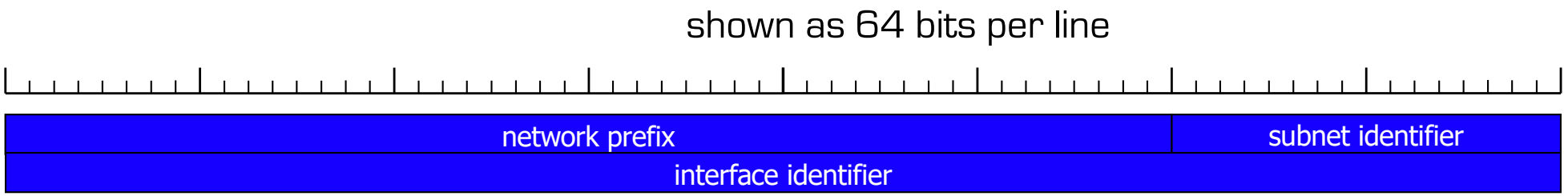
a bit more compact:

```
2a00:1450:400f:080a:0000:0000:0000:2004
```

```
This is an address for www.google.com
```

IPv6 addresses

- example of the IPv6 address spaces



a typical routed address

- IPv6 addresses are written in sets of 2 bytes in hexadecimal notation
- `2a00:1450:400f:080a:0000:0000:0000:2004`

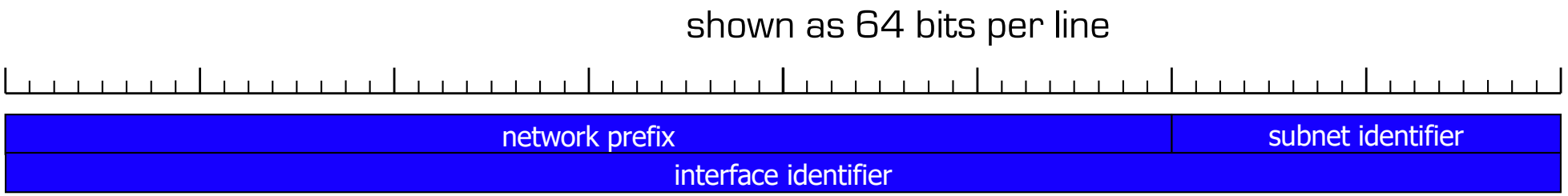
- sets of zero can be compressed:
- `2a00:1450:400f:80a::2004`

So, its netmask has 48 1-bits followed by 80 0-bits

- this address is part of the network
  `2a00:1450:400f::/48`
  which is known to be used by Google since 12/2018

IPv6 addresses

- example of the IPv6 address spaces



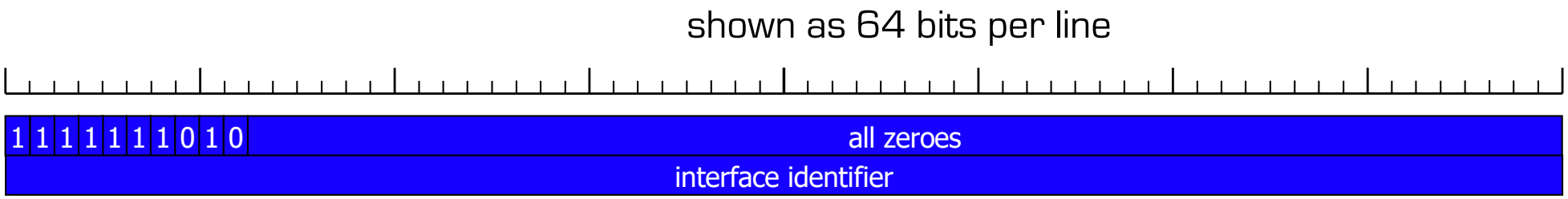
a typical routed address

Acquiring IPv6 addresses for mobile computers

1. getting a non-routable IPv6 address using auto-configuration
   - self-assigns an IPv6 address consisting of prefix FE80::0 followed by the interface identifier (RFC4291), which is created from the MAC address (RFC 8064)
   - before *using* the address, "Neighbour Solicitation" ICMP message must be sent to ensure the address is not in use yet (RFC4443) – variation of ARP Probe
2. DHCPv6 to ask for a routable address
   - requires auto-configured local address first
3. Mobile IPv6 to transfer routable home address to visited network
   - requires auto-configured local address first

IPv6 addresses

- example of the IPv6 address spaces



a link-local address

Acquiring IPv6 addresses for mobile computers

- example IPv6 address of jordin.ifi.uio.no:
  `fe80::250:56ff:fe80:3f82`
- which is an abbreviation for
  `fe80:0000:0000:0000:0250:56ff:fe80:3f82`

- for us, so far mostly worthless because UiO does not route IPv6 anyway

IN2140:
Introduction to Operating Systems and Data Communication

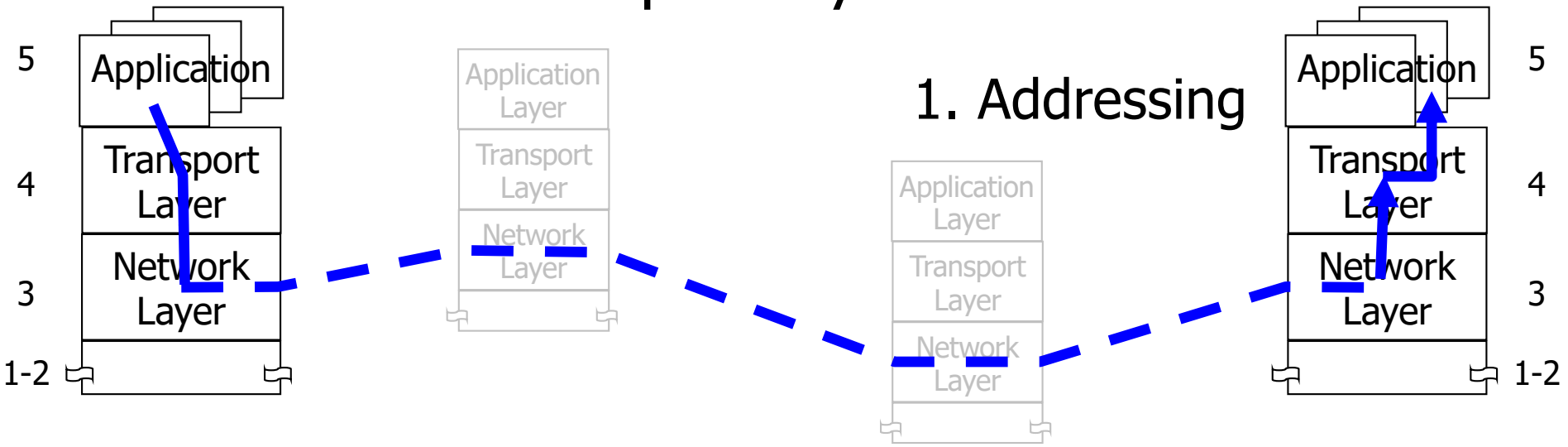


Addressing in the TCP/IP model

Layer 4 Address Resolution: Ports and more

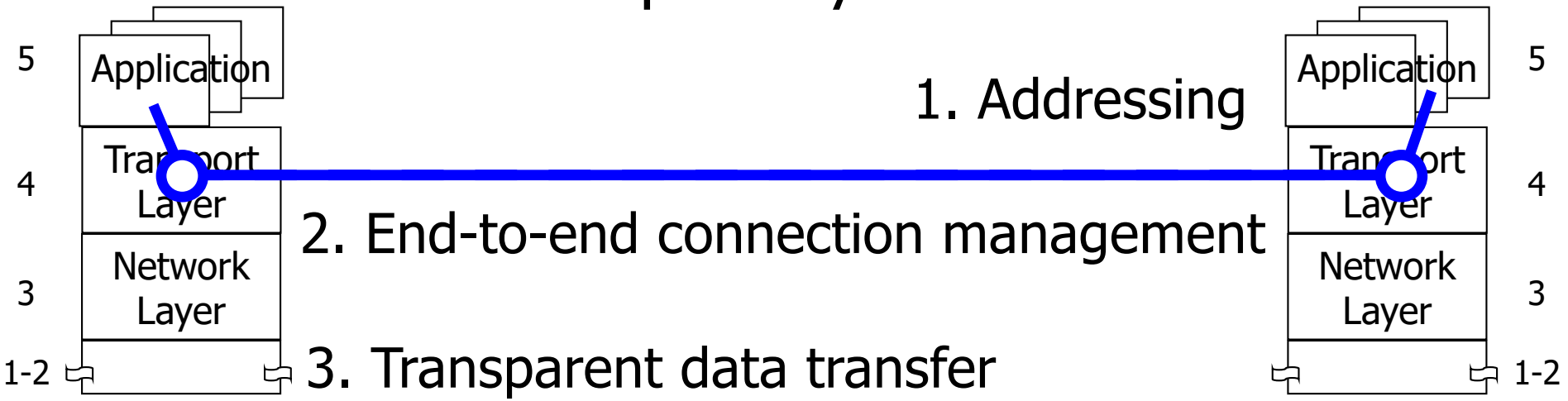
Transport Layer Function

Transport layer tasks



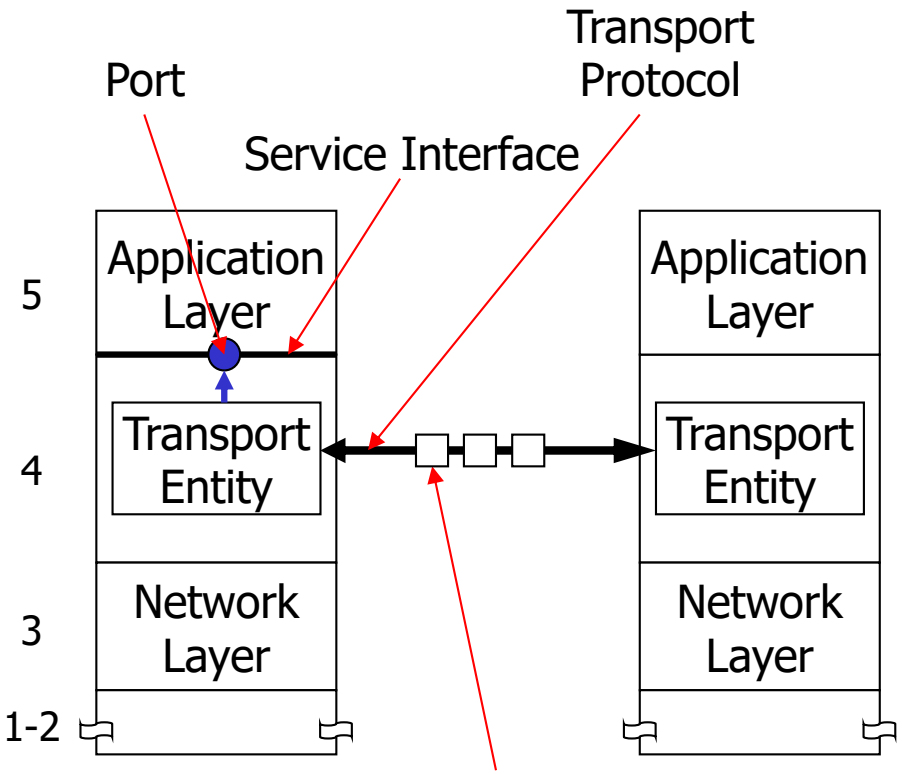
Transport Layer Function

Transport layer tasks



4. Quality of service options
   - Error recovery
   - Reliability
   - Flow control
   - Congestion control

Transport Layer Addressing



TCP: Message, UDP: Datagram, ISO: TPDU

Addresses

TCP/IP 1. Port
2. IP address !!

ISO TSAP
(transport service access point)

Transport Layer Function

UDP

```
int main()
{
    …
    if((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    { … }

    memset(&addr, 0, sizeof(addr));
    addr.sin_family      = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port        = htons(PORT);

    if(bind(sock,(struct sockaddr *)&addr,
            sizeof(addr))<0)
    { … }

    len = sizeof(struct sockaddr_in);
    n = recvfrom(sock,(char*)buffer,buflen,0,
                 (struct sockaddr*)&addr,&len);
    …
    sendto(sock,(char*)msg, strlen(msg),
           0,(struct sockaddr*)&addr,len);
    …
}
```

Connectionless

No error recovery
No reliability
No flow control
No congestion control

TCP

```
int main()
{
    …
    if((sock = socket(AF_INET, SOCK_STREAM, 0))<0)
    { … }
    …
    memset(&servaddr, sizeof(addr));
    addr.sin_family      = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port        = htons(PORT);
    …
    if((bind(sock,(struct sockaddr*)&addr,
             sizeof(addr)))<0)
    { … }
    if((listen(sock,10))<0)
    { … }

    len = sizeof(struct sockaddr_in);
    conn = accept(sock,(struct sockaddr*)&addr,&len);
    if(conn < 0) { … }

    len = sizeof(buf); i=0;
    while(len>0 && n=read(conn,&buf|i],len))
    { i+=n; len-=n; }
    if(n<=0) { … }
    …
    write(conn,buf,1);
    …
}
```

Connection-oriented

Error recovery
Reliability
Flow control
Congestion control

Transport Layer Function

UDP

```
int main()
{
    …
    if((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    { … }

    memset(&addr, 0, sizeof(addr));
    addr.sin_family      = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port        = htons(PORT);

    if(bind(sock,(struct sockaddr *)&addr,
            sizeof(addr))<0)
    { … }



    len = sizeof(struct sockaddr_in);
    n = recvfrom(sock,(char*)buffer,buflen,0,
                 (struct sockaddr*)&addr,&len);
    …
    sendto(sock,(char*)msg, strlen(msg),
           0,(struct sockaddr*)&addr,len);
    …
}
```

Bind the port PORT to the socket

TCP

```
int main()
{
    …
    if((sock = socket(AF_INET, SOCK_STREAM, 0))<0)
    {  … }
    …
    memset(&servaddr, sizeof(addr));
    addr.sin_family      = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port        = htons(PORT);
    …
    if((bind(sock,(struct sockaddr*)&addr,
             sizeof(addr)))<0)
    { … }
    if((listen(sock,10))<0)
    { … }

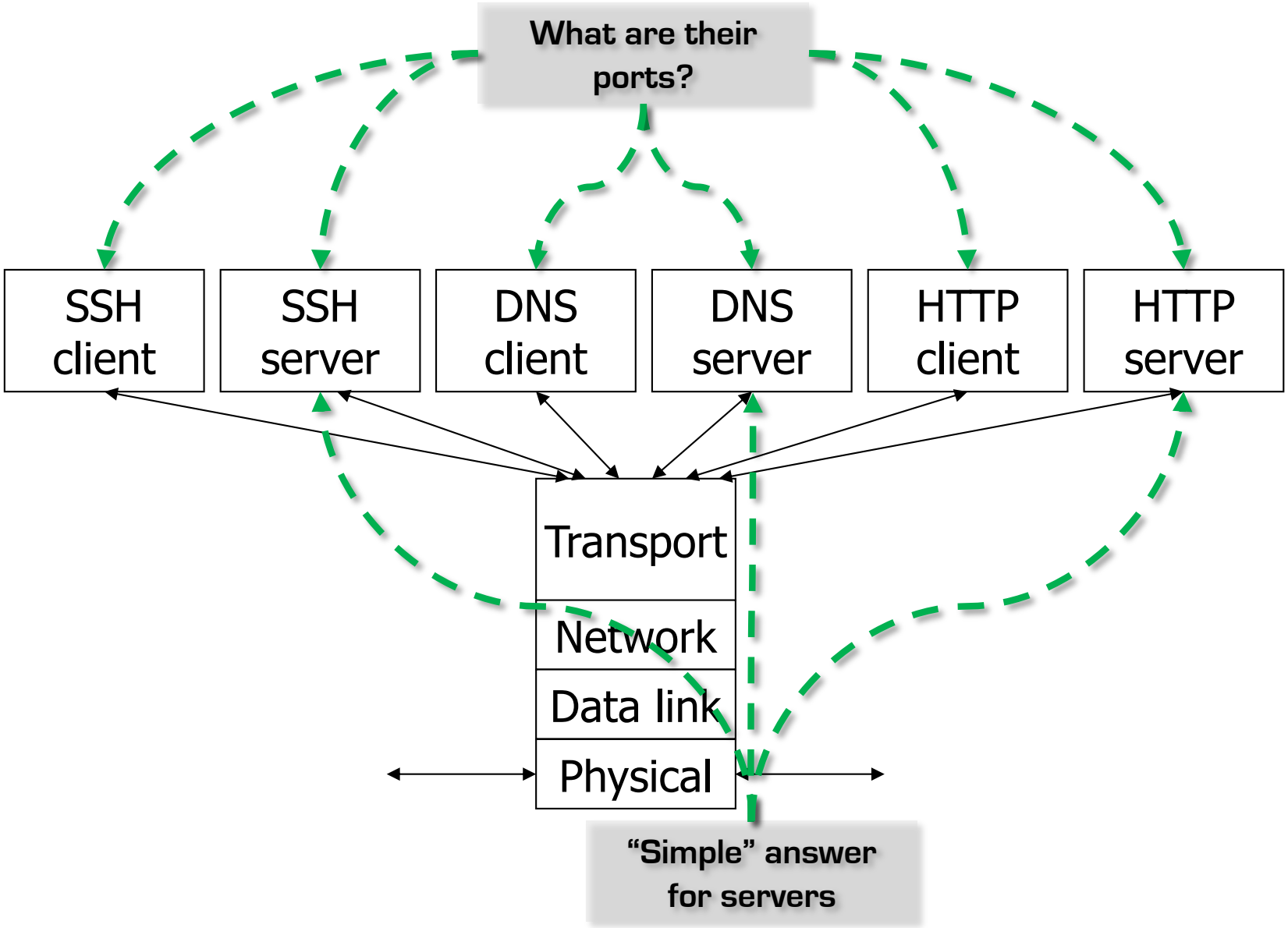
    len = sizeof(struct sockaddr_in);
    conn = accept(sock,(struct sockaddr*)&addr,&len);
    if(conn < 0) { … }

    len = sizeof(buf); i=0;
    while(len>0 && n=read(conn,&buf[i],len))
    { i+=n; len-=n; }
    if(n<=0) { … }
    …
```

Bind the port PORT to the socket

But how to inform the other side about this port number?

Addressing at the Transport Layer



Addressing at the Transport Layer

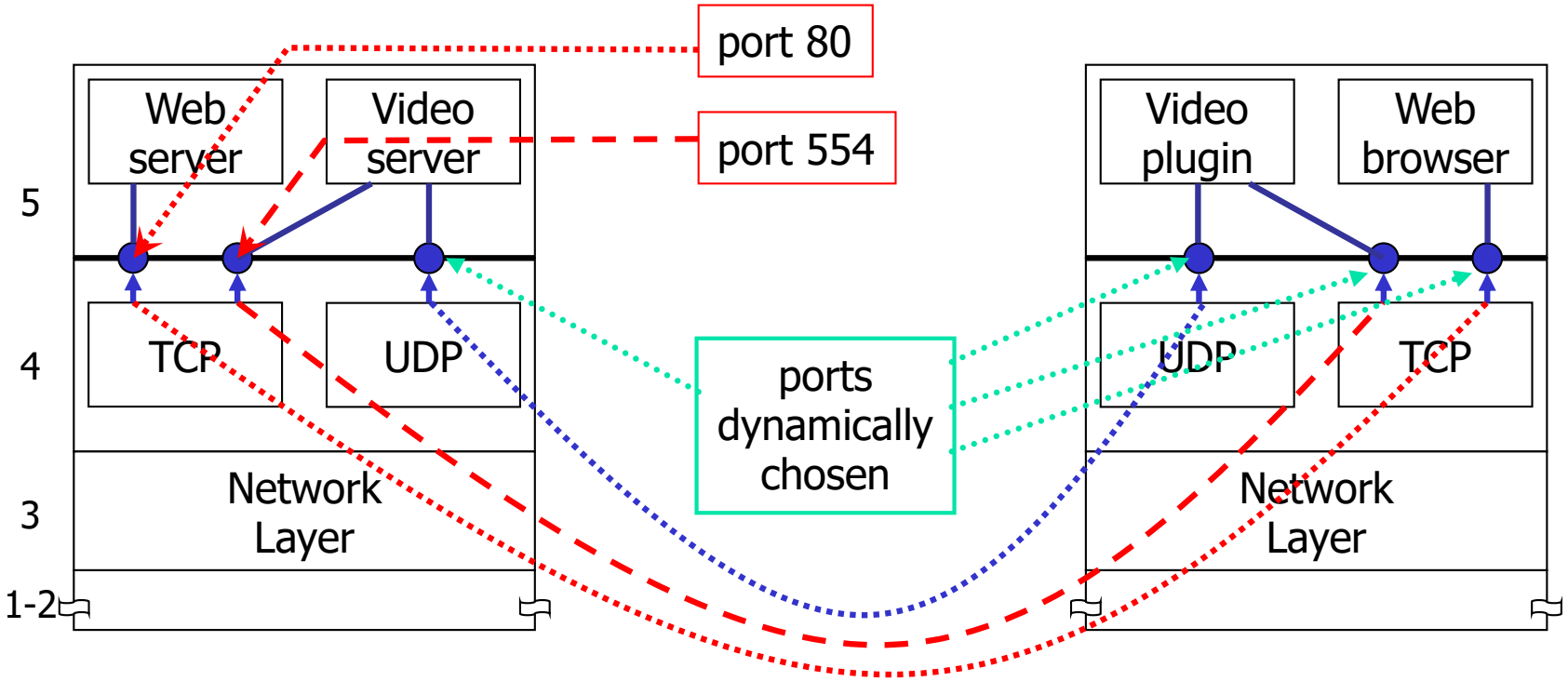
- There is a list maintained by IANA – Internet Assigned Numbers Authority

Port Number	Service	Service name
22	SSH	Secure Shell
25	SMTP	Simple Mail Transfer Protocol
53	DOMAIN	Domain Name Systems
80	HTTP	Hypertext Transport Protocol
110	POP3	Post Office Protocol, version 3
123	NTP	Network Time Protocol
143	IMAP	Internet Message Access Protocol

- TCP and UDP have separate assignments
  - same port number – different service
  - The table shows some examples for TCP (read /etc/services for more)

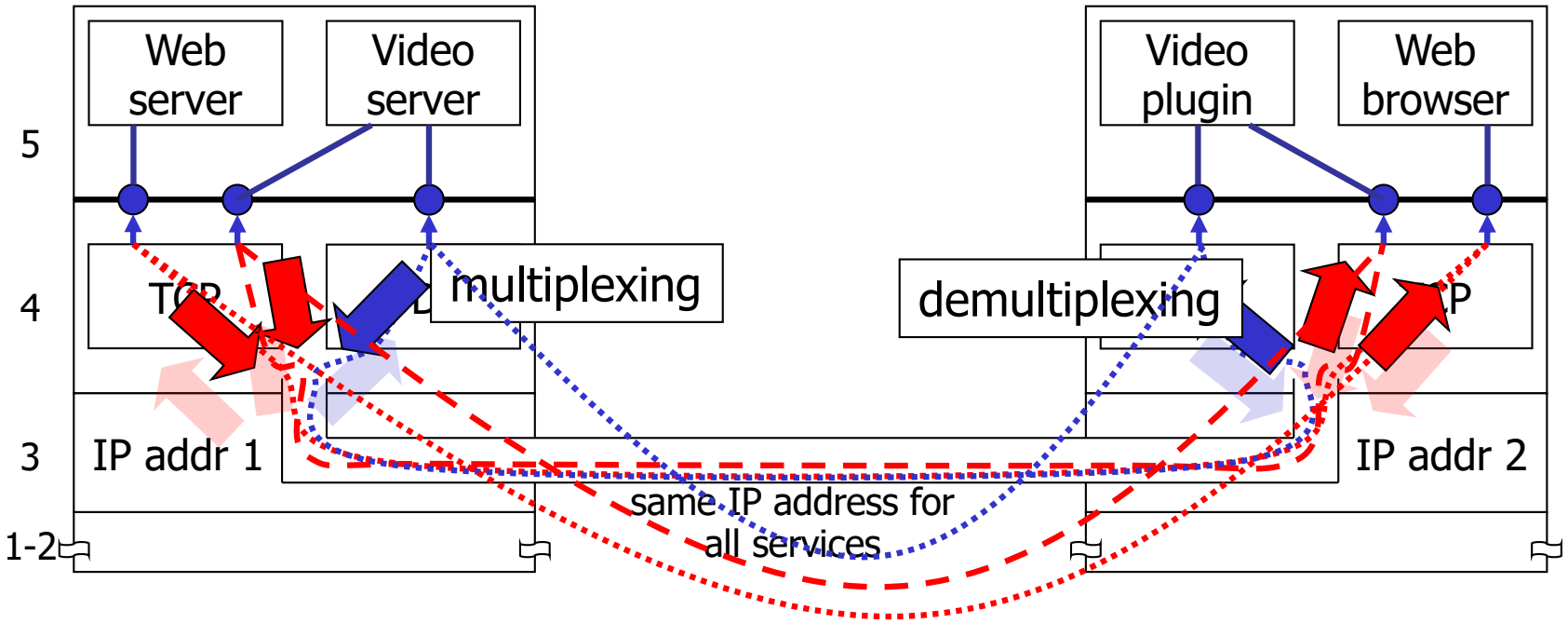
Multiplexing task of the Transport Layer

- Multiplexing and demultiplexing task of the transport layer
- Example: accessing a web page with video element
  - Three protocols used (minor simplification)
    - HTTP for web page
    - RTSP for video control
    - RTP for video data



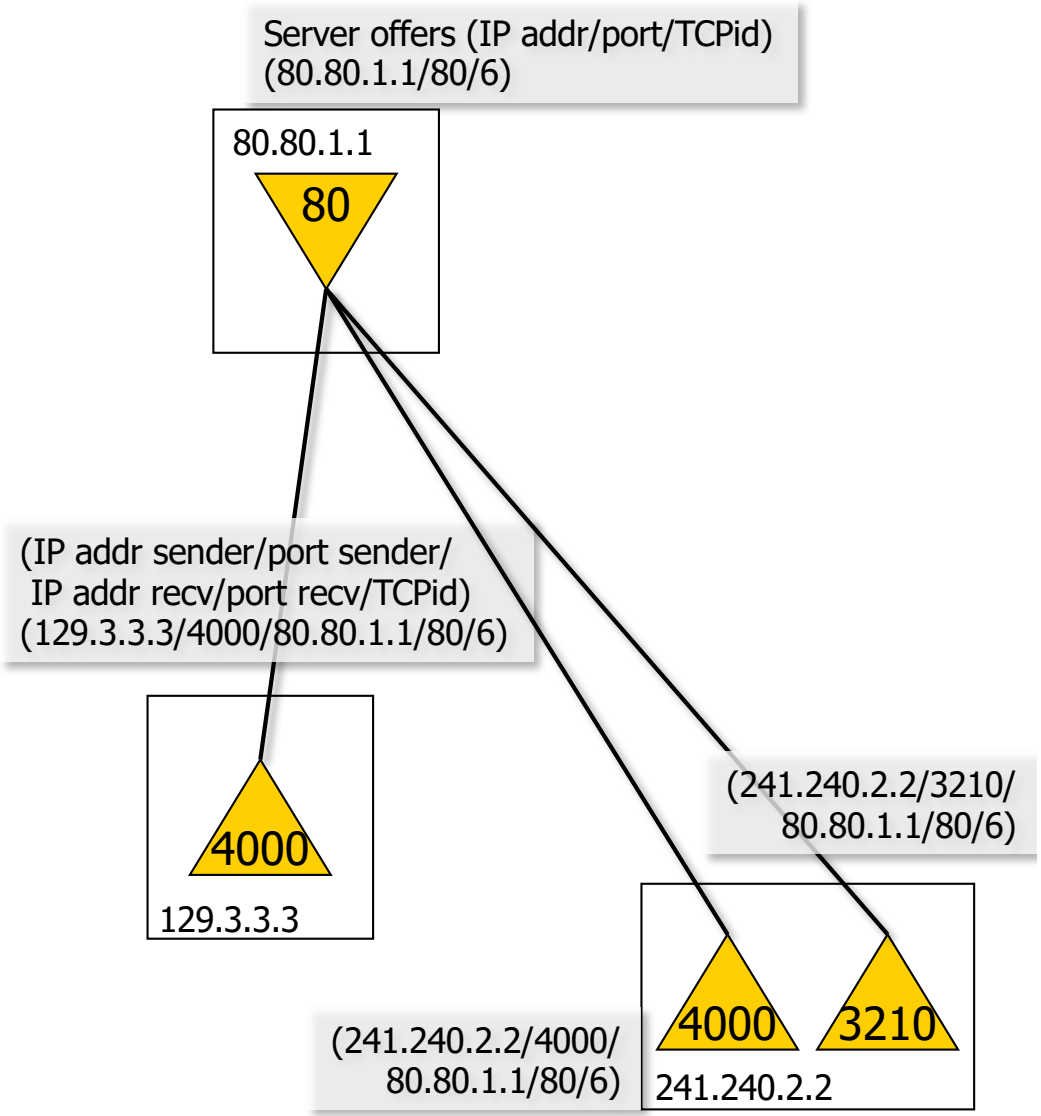
Multiplexing task of the Transport Layer

- Multiplexing and demultiplexing task of the transport layer
- Example: accessing a web page with video element
  - Three protocols used (minor simplification)
    - HTTP for web page
    - RTSP for video control
    - RTP for video data



Connection – Addressing

- **TCP *service*** obtained via service endpoints on sender and receiver
  - Typically socket
  - Socket number consists of a ***3-tuple***
    - IP address of host and
    - 16-bit local number (port)
    - TCP protocol identifier
- Transport Service Access Point
  - Port
- **TCP *connection*** is clearly defined by a ***5-tuple*** consisting of
  - IP address of sender and receiver
  - Port address of sender and receiver
  - TCP protocol identifier
- Applications can use the same local ports for several TCP connections
  - if the remote side is different



IN2140:
Introduction to Operating Systems and Data Communication



Addressing in the TCP/IP model

Layer 5 Address Resolution: DNS – Domain Name System

How to connect to a remote computer?

Connect to <hostname,port>

- e.g. `telnet 127.0.0.1 23` `or telnet ::1 23`
  talking to my own machine
  special addresses

- or `wget http://173.194.39.31:80/`
  talking to one of Google's machines
  possible to remember

- or `ssh 9.228.93.3`
  trying to talk to my desktop that had this address in 1995
  impossible to remember unless you've typed it 100 times a day

- If you want short names, write them into /etc/hosts
- originally globally maintained by SRI, changes re-distributed by email and ftp (no more, ancient history)

How to connect to a remote computer?

Use “reasonable” names

- e.g.

```
ssh login.ifi.uio.no
wget www.google.com
```

- not only easier to remember
- reflects also organisation structures

- although the hierarchical structure may not fulfill all purposes
- somewhat related to physical network structure, at least locally

Domain Name System (DNS)

DNS at a High-Level

Domain Name System

Hierarchical namespace

As opposed to original, flat namespace

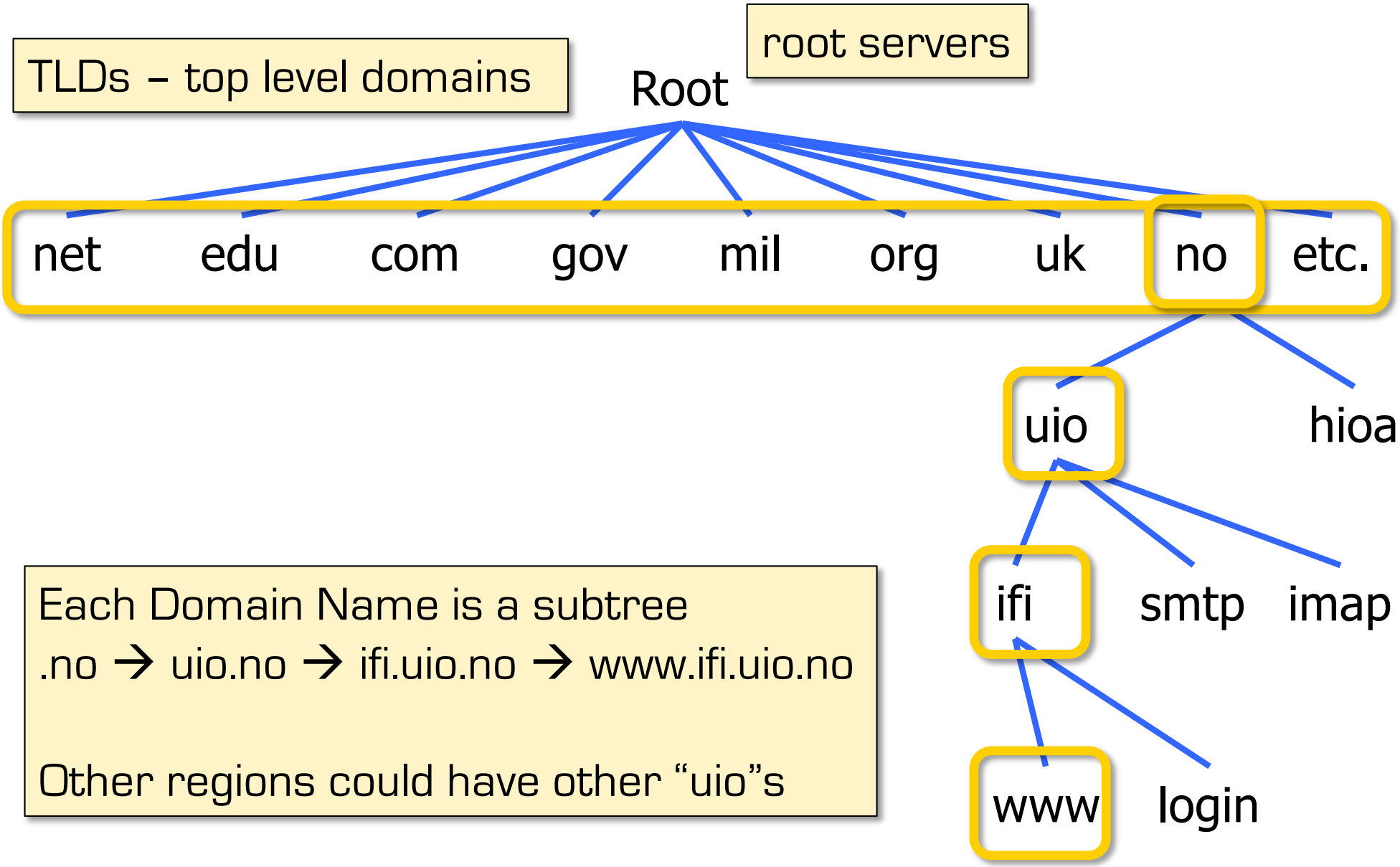
e.g. .com → google.com → mail.google.com

Distributed database

Simple client/server architecture

- UDP or TCP port 53
- servers must use TCP nowadays
- clients using TCP are mostly rejected
  - reduces server load
  - is a security problem

Naming Hierarchy

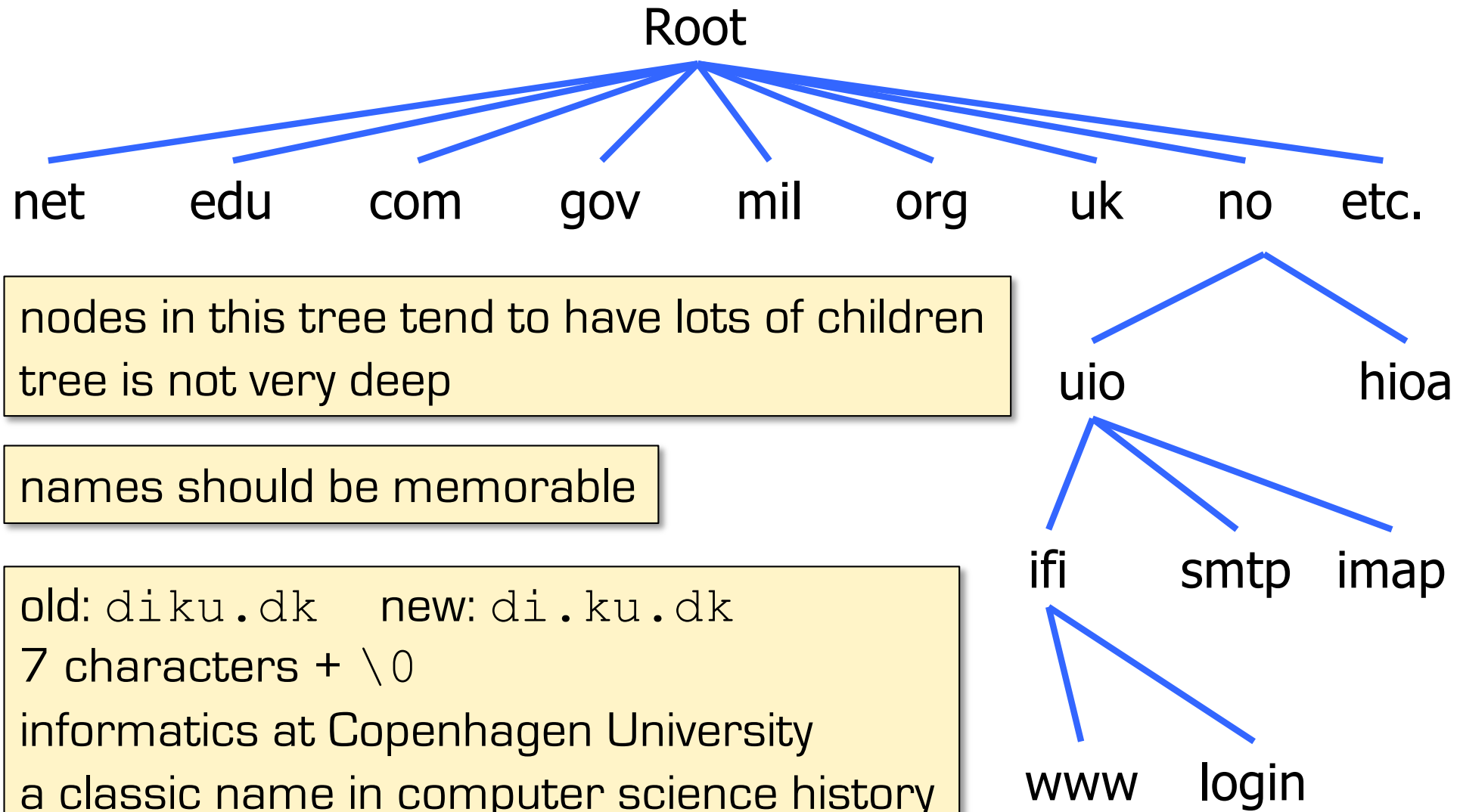


Each Domain Name is a subtree

.no → uio.no → ifi.uio.no → www.ifi.uio.no

Other regions could have other “uio”s

Naming Hierarchy

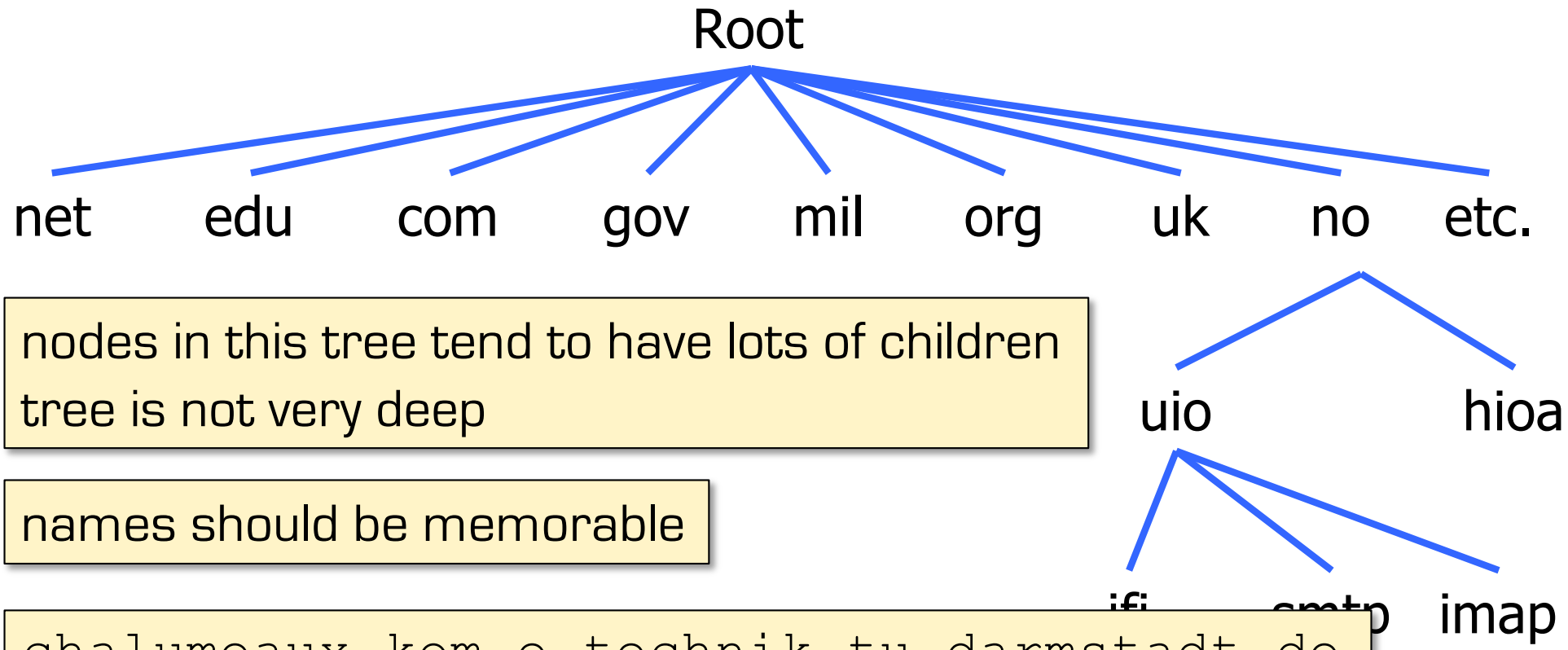


nodes in this tree tend to have lots of children
tree is not very deep

names should be memorable

old: `diku.dk` new: `di.ku.dk`
7 characters + `\0`
informatics at Copenhagen University
a classic name in computer science history
not obvious but memorable

Naming Hierarchy

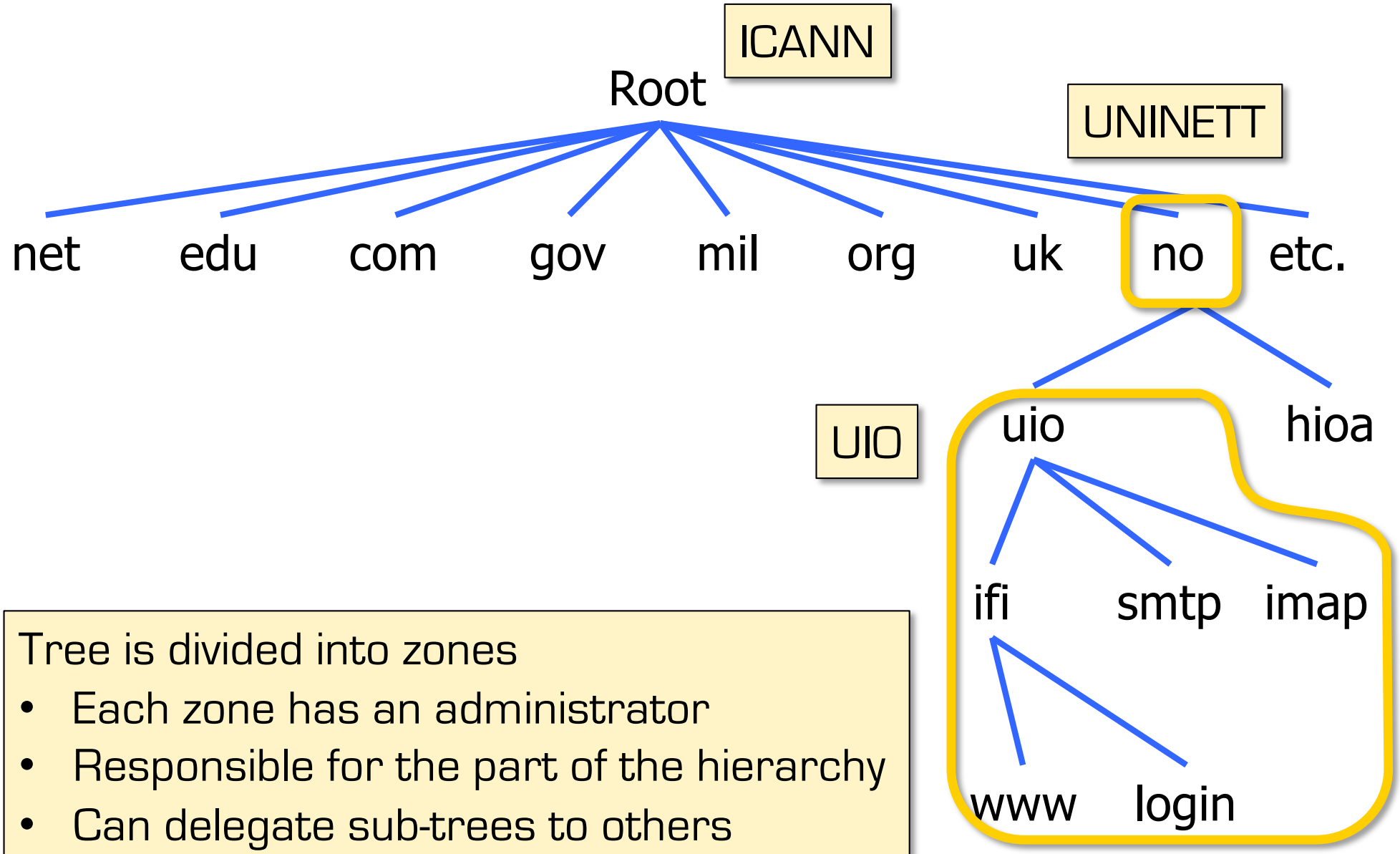


nodes in this tree tend to have lots of children
tree is not very deep

names should be memorable

`chalumeaux.kom.e-technik.tu-darmstadt.de`
40 characters + `\0`
login from Mac & BSD still failed in the 2000s: name was cut after 32 characters

Hierarchical Administration



Tree is divided into zones

- Each zone has an administrator
- Responsible for the part of the hierarchy
- Can delegate sub-trees to others

Server Hierarchy

Functions of each DNS server

- Authority over a portion of the hierarchy
  - No need to store all DNS names
- Store all the records for hosts/domains in its zone
  - **Must** be replicated for robustness (at least 2 servers)
- Know the addresses of the root servers
  - Resolve queries for unknown names

Root servers know about all TLDs

Root Name Servers

Responsible for the Root Zone File

- Lists the TLDs and who controls them

```
com.            172800 IN     NS     a.gtld-servers.net.
com.            172800 IN     NS     b.gtld-servers.net.
com.            172800 IN     NS     c.gtld-servers.net.
```

Administered by ICANN

- 13 root servers, labeled A→M
- 6 are anycasted, i.e. they are globally replicated

Contacted when names cannot be resolved

- In practice, most systems cache this information
- DDoS attacks designed to reach root (30.11 & 1.12. 2015: 5 mio queries per seccond)
- infrastructure bugs (e.g. old Telenor modems converted IPv6 lookup into broken IPv4 lookup)

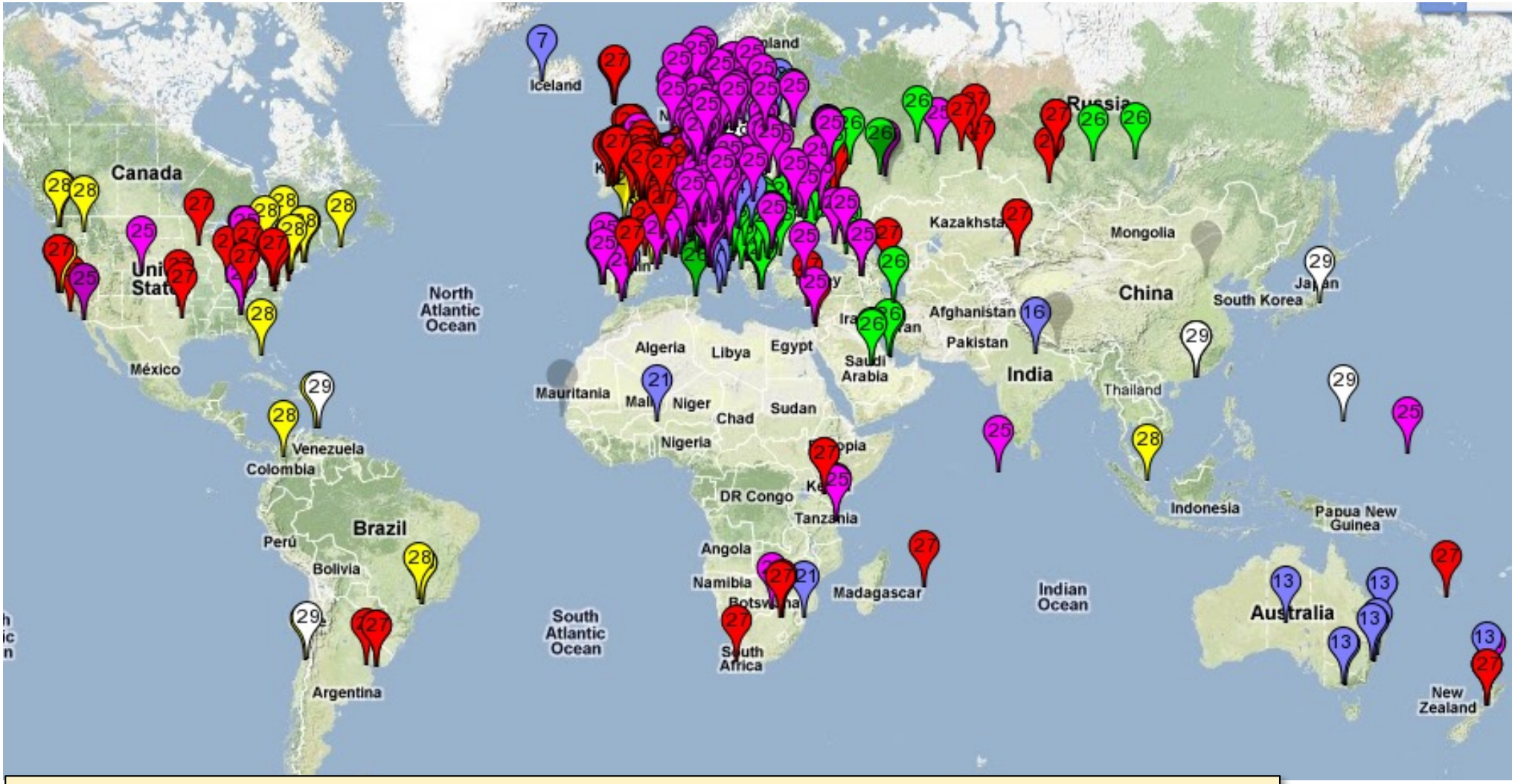
ICANN

Map of the Root Servers



from: http://www.icann.org/en/news/correspondence/roberts-testimony-14feb01-en.htm

Map of the Roots



k-root (Europe) is an anycast root node

This is RIPE's map of probing which of the 6 k-root copies get accessed

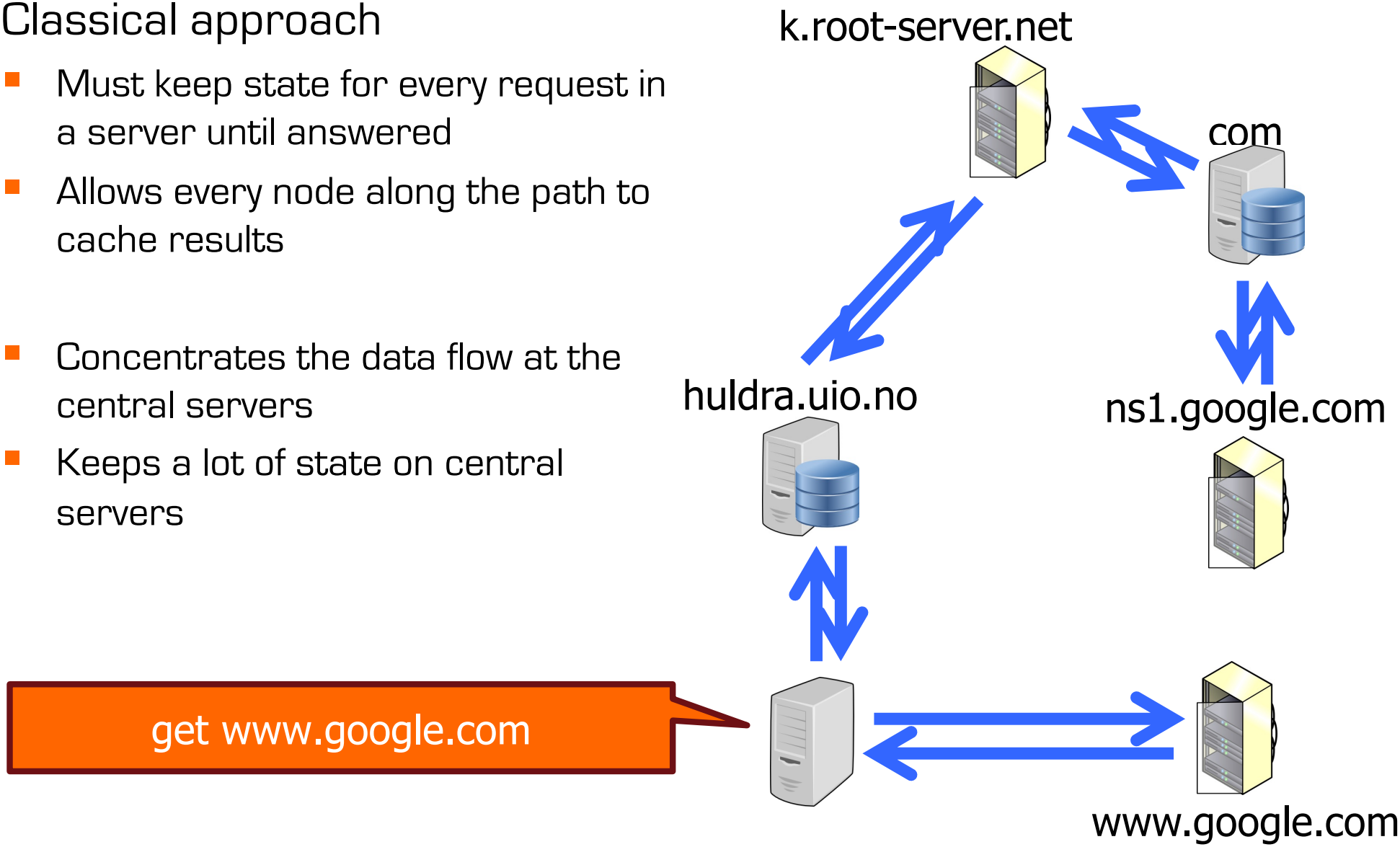
from https://labs.ripe.net/Members/kistel/dns-measurements-with-ripe-atlas-data

Recursive DNS Query

Classical approach

- Must keep state for every request in a server until answered
- Allows every node along the path to cache results

- Concentrates the data flow at the central servers
- Keeps a lot of state on central servers

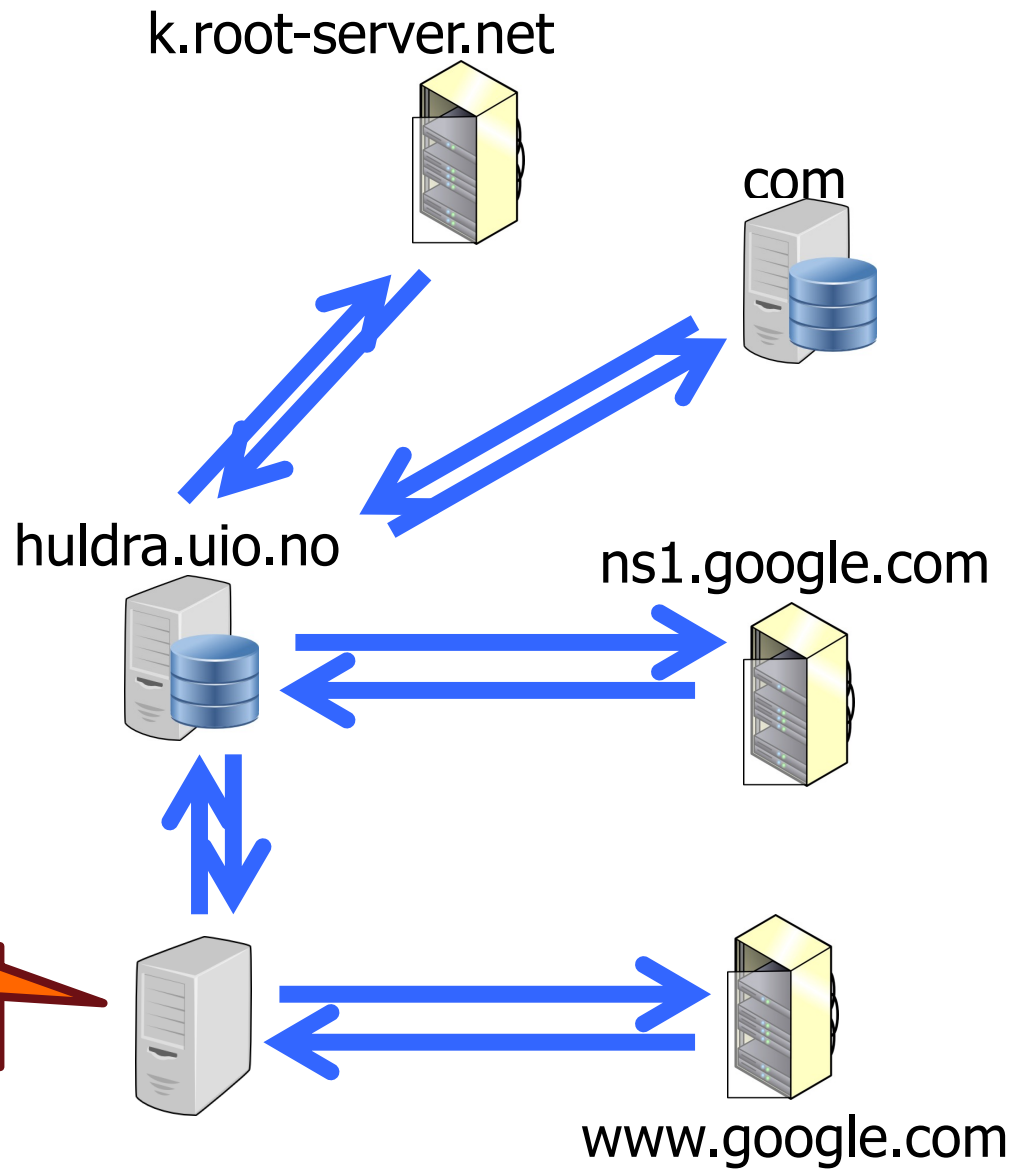


Iterated DNS Query

Newer approach

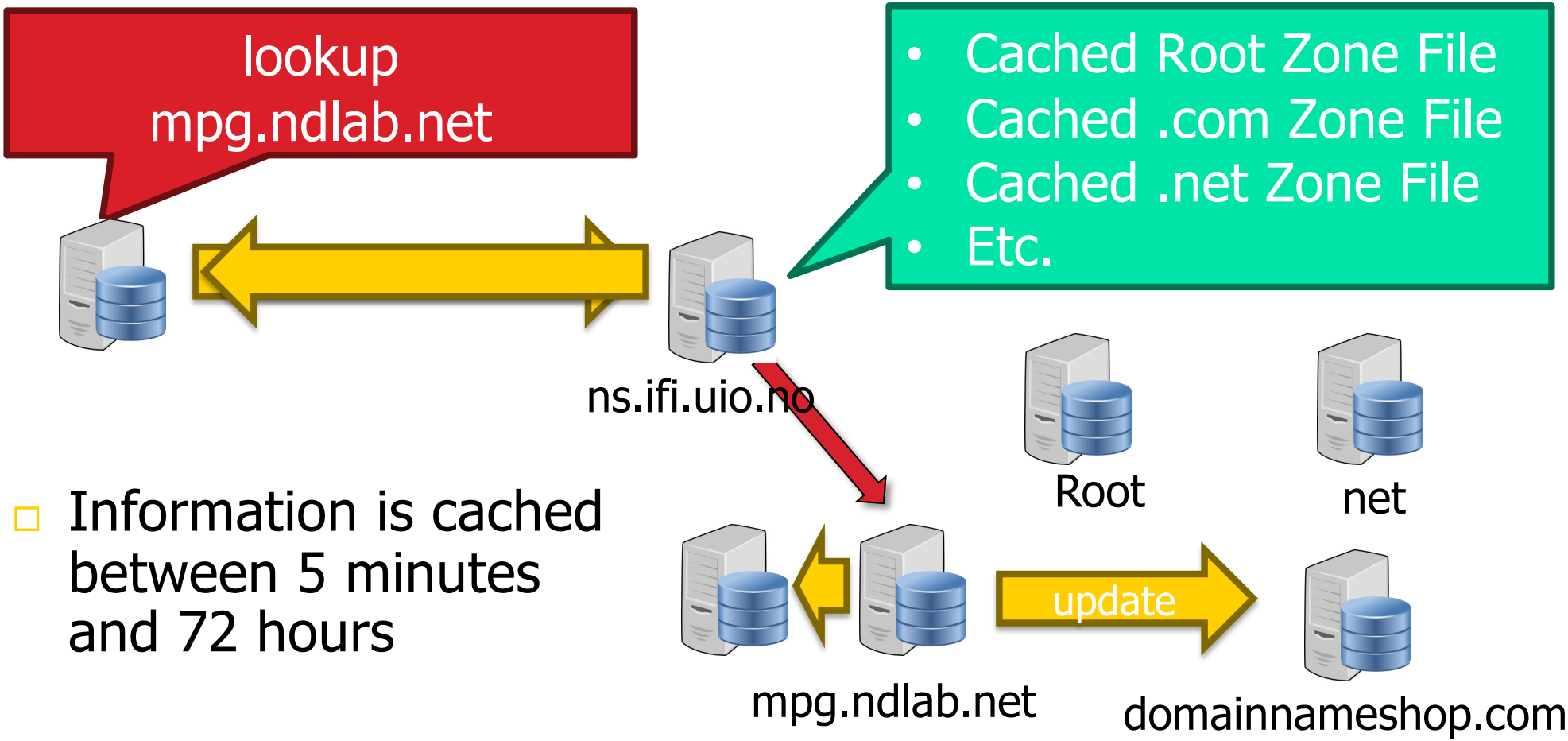
- Redirects request
- Keep state only at local server (or some servers) until answered

- Allows few nodes to cache results
- Halves number of requests at central servers
- Avoids state on central servers entirely



Caching vs. Freshness

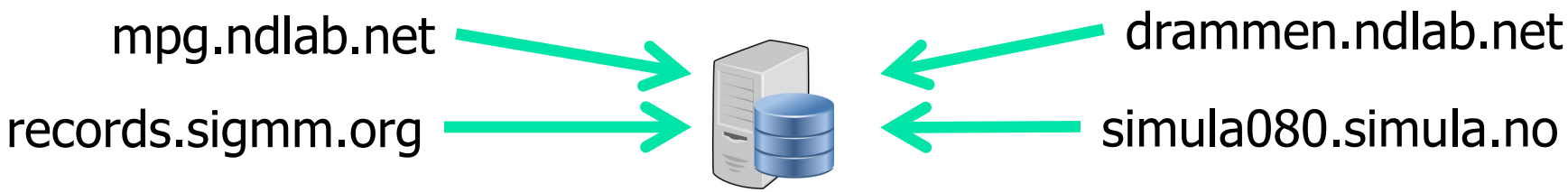
- Caching reduces DNS resolution latency
- Caching reduces server load
- Caching delays updates



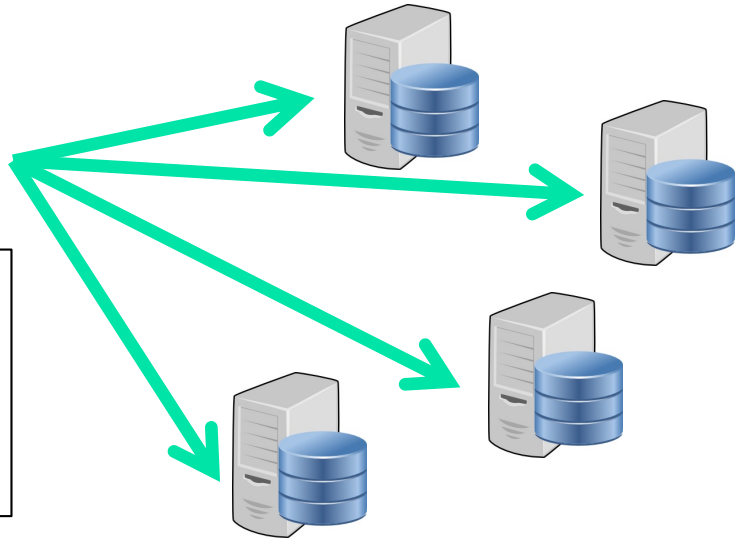
- Information is cached between 5 minutes and 72 hours

Aliasing and Load Balancing

One machine can have many aliases



One name can map to multiple machines

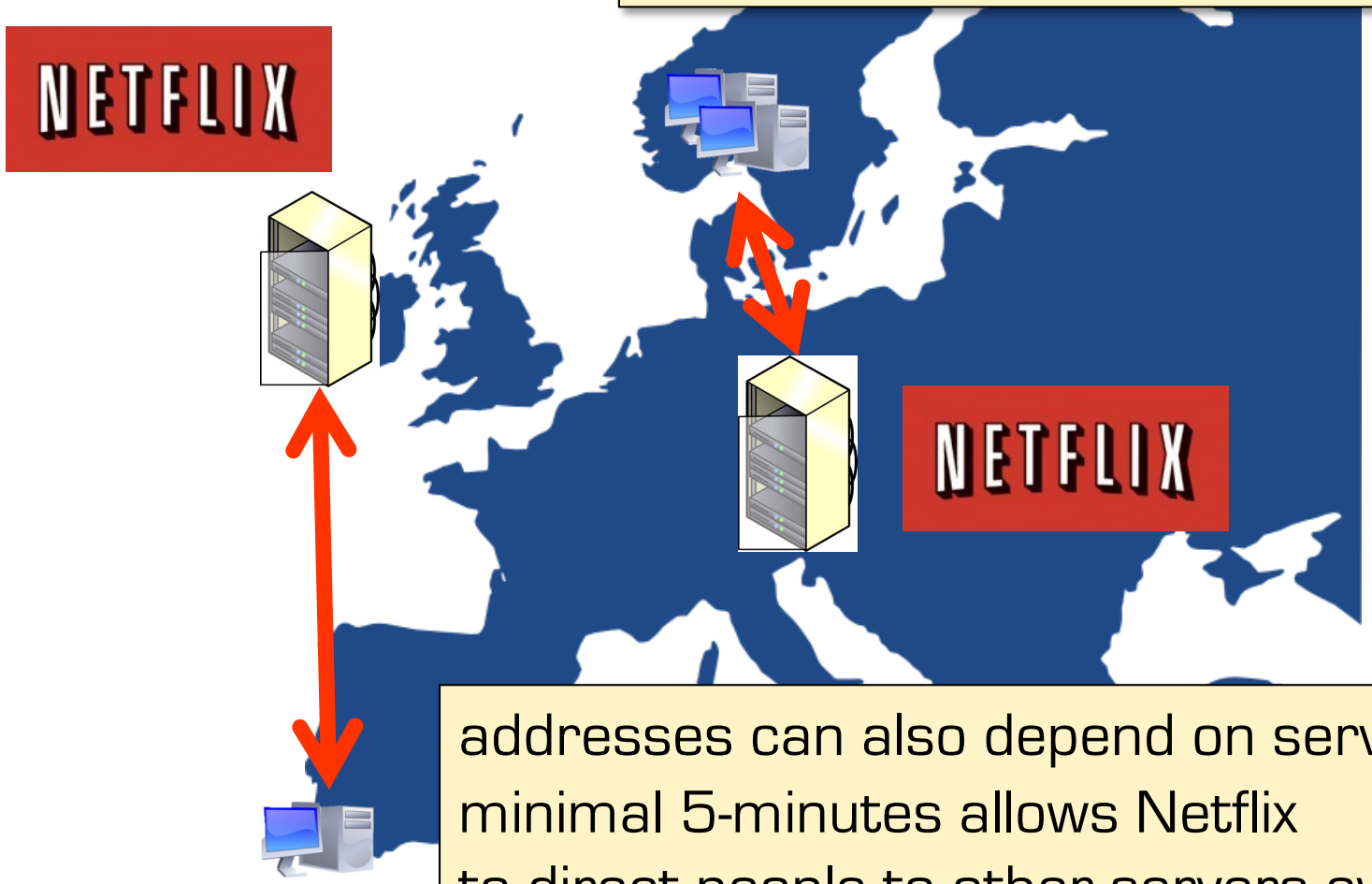


That includes k.root-server.net
and
login.ifi.uio.no

Content Delivery Networks

DNS allows zoning

e.g. Netflix (and Google) addresses depend on the origin of your connection
geography, ISP, ...



addresses can also depend on server load
minimal 5-minutes allows Netflix
to direct people to other servers every 5 minutes

Content Delivery Networks

DNS allows zoning

e.g. Netflix (and Google) addresses depend on the origin of your connection
geography, ISP, ...

"Small problem" with this technique

- modern to use external ***resolvers***
- e.g. ***ALL*** Chrome DNS lookups seem to originate from 8.8.8.8 (an address owned by Google)

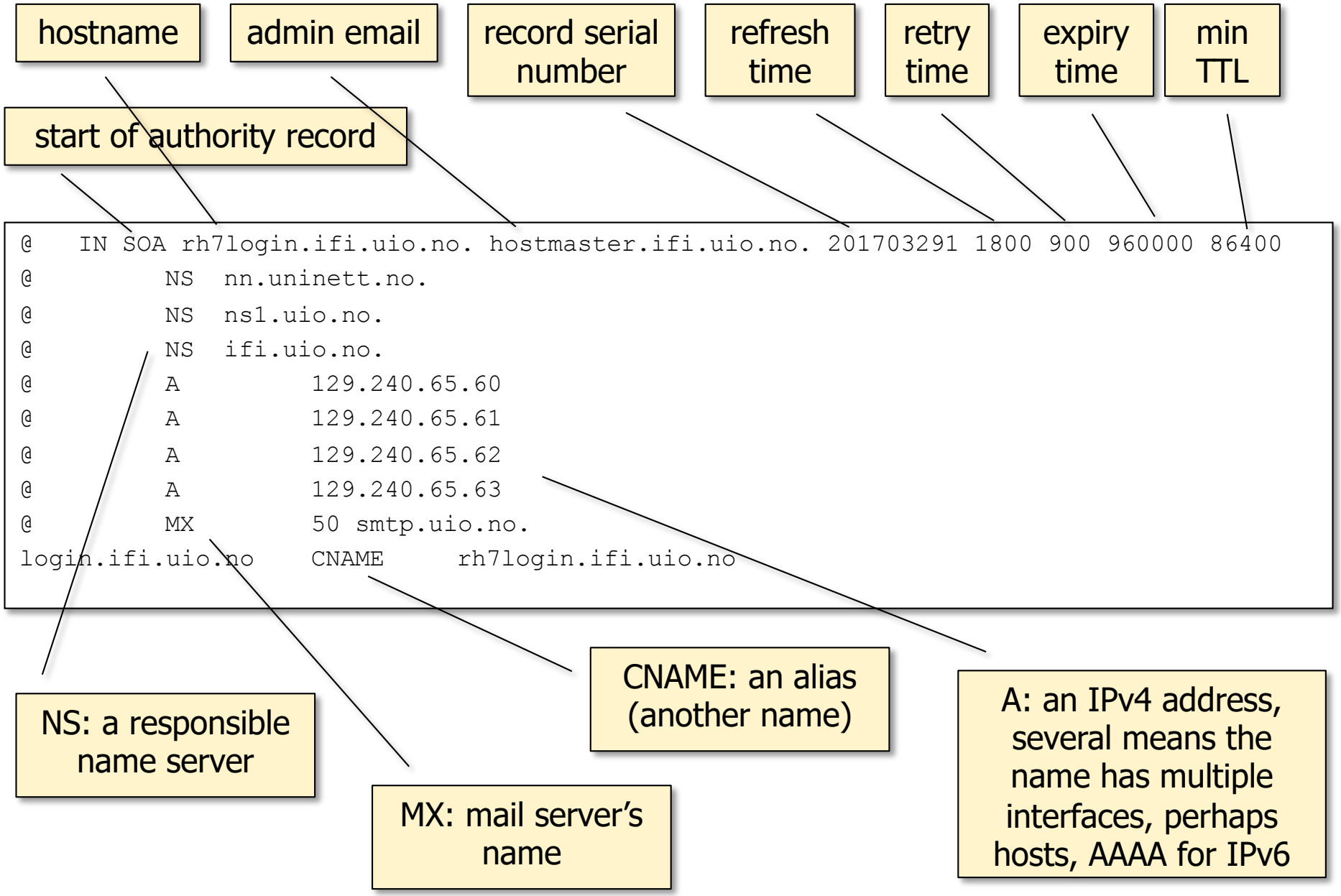
Consequences

- user stays more anonymous
- Netflix and others make wrong decisions



addresses can also depend on server load
minimal 5-minutes allows Netflix
to direct people to other servers every 5 minutes

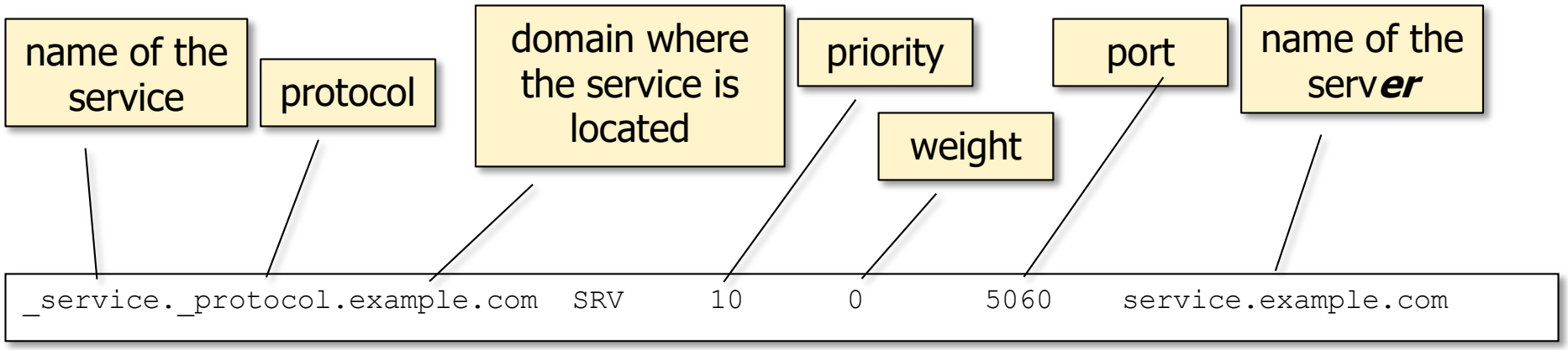
DNS Record


hostname
admin email
record serial number
refresh time
retry time
expiry time
min TTL
start of authority record
@ IN SOA rh7login.ifi.uio.no. hostmaster.ifi.uio.no. 201703291 1800 900 960000 86400
@ NS nn.uninett.no.
@ NS ns1.uio.no.
@ NS ifi.uio.no.
@ A 129.240.65.60
@ A 129.240.65.61
@ A 129.240.65.62
@ A 129.240.65.63
@ MX 50 smtp.uio.no.
login.ifi.uio.no CNAME rh7login.ifi.uio.no
NS: a responsible name server
CNAME: an alias (another name)
A: an IPv4 address, several means the name has multiple interfaces, perhaps hosts, AAAA for IPv6
MX: mail server's name

mDNS

A way of discovering services by announcing them with IP multicast

- RFC 6762 (2013): multicast DNS
- records announce services (as well as link-local hostnames that are invisible outside the current multicast domain, like mymac.local)
- records are never authoritative and mDNS can never redirect or recurse



Example from my machine:

```
_ssh._tcp.example.com  SRV     10      0       22 1x-193-157-212-9.uio.no
```