IN2140:
Introduction to Operating Systems and Data Communication

Data Communication:

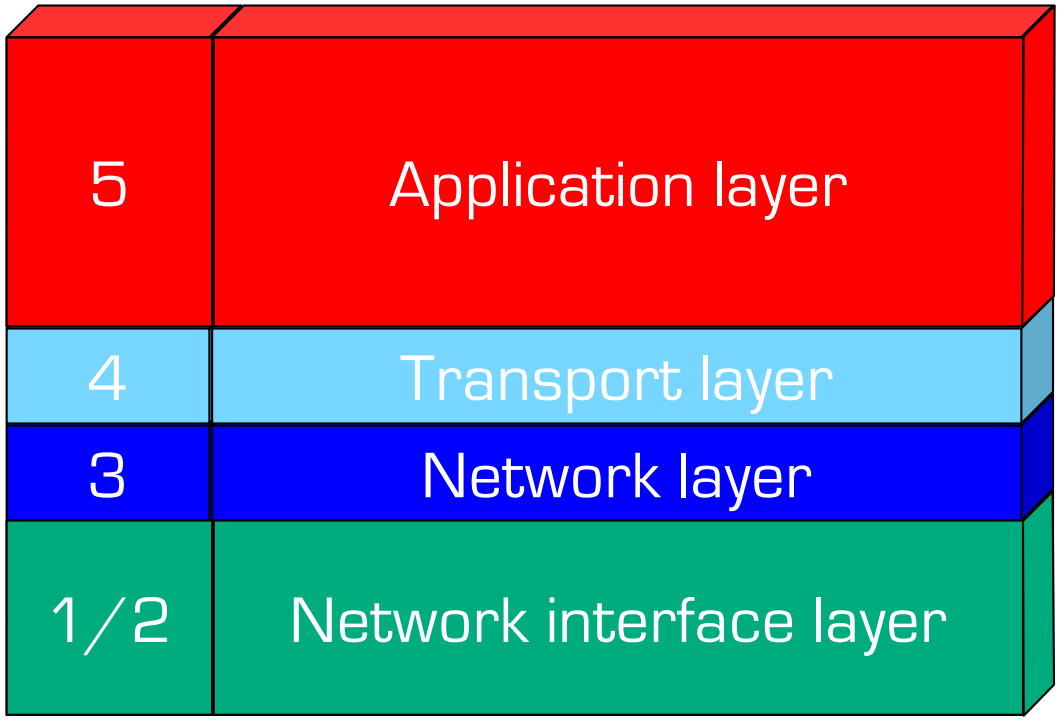
Naming

Layer encapsulation

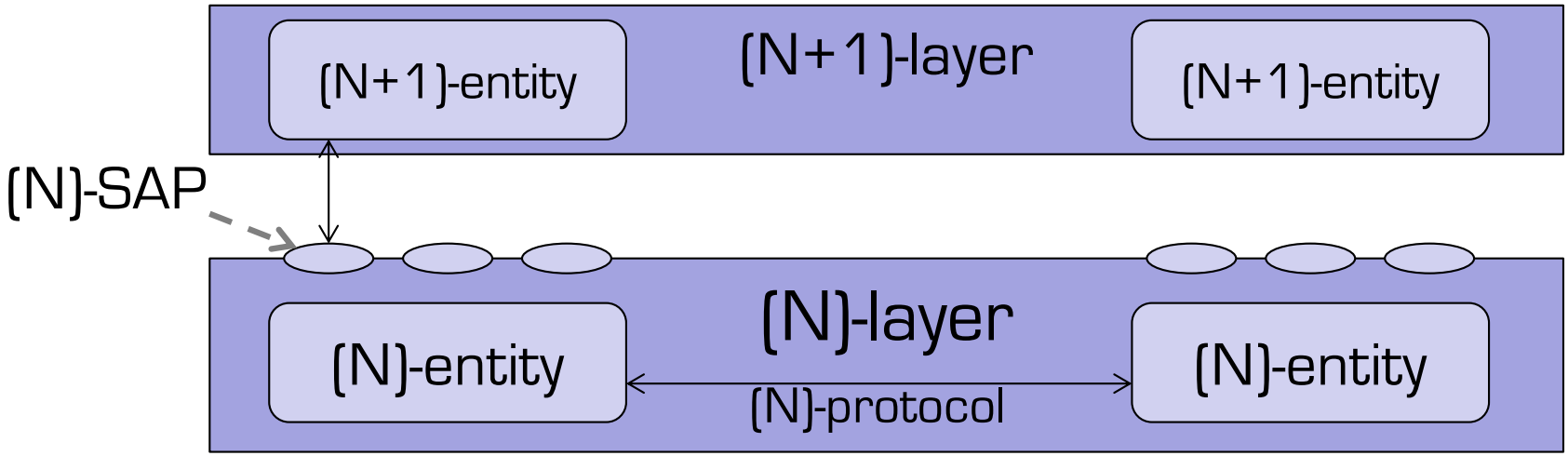
Five Layer Reference

TCP/IP Reference Model Internet Architecture

- ISO-OSI presentation, session and application layer merged
- ISO-OSI data link layer and physical layer merged to form Network Interface



Layers in General (OSI terminology)



(N)-Layer

- abstraction level with defined tasks

(N)-Entity

- active elements within a layer
- process or intelligent I/O module
- **peer entities**: corresponding entities on different systems

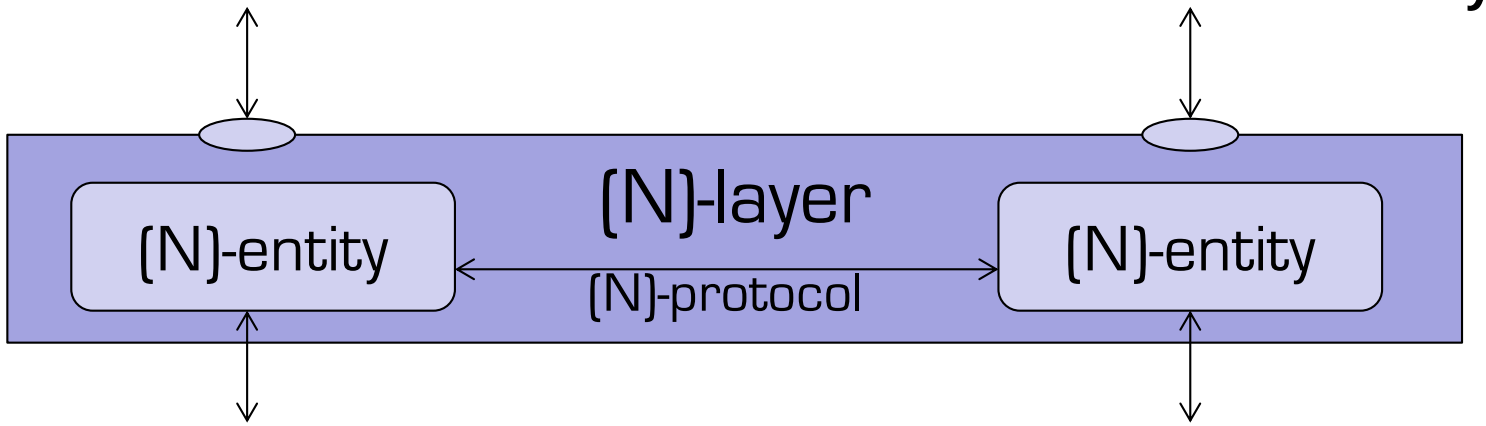
(N)-Service Access Point, (N)-SAP

- service identification
- describes how layer N provides a service for layer N+1
- an Entity can offer several services

(N)-Protocol

- a multitude of rules for transferring data between peers

Protocol: Communication between same Layers



Definition of protocol

- A protocol defines
  - the format
  - the order of messages
  - exchanged between two or more communicating entities
  - as well as the actions taken on transmission and/or reception of a message or other event
- It does not define
  - the services offered to layer N+1
  - the services used (N-1-SAP)

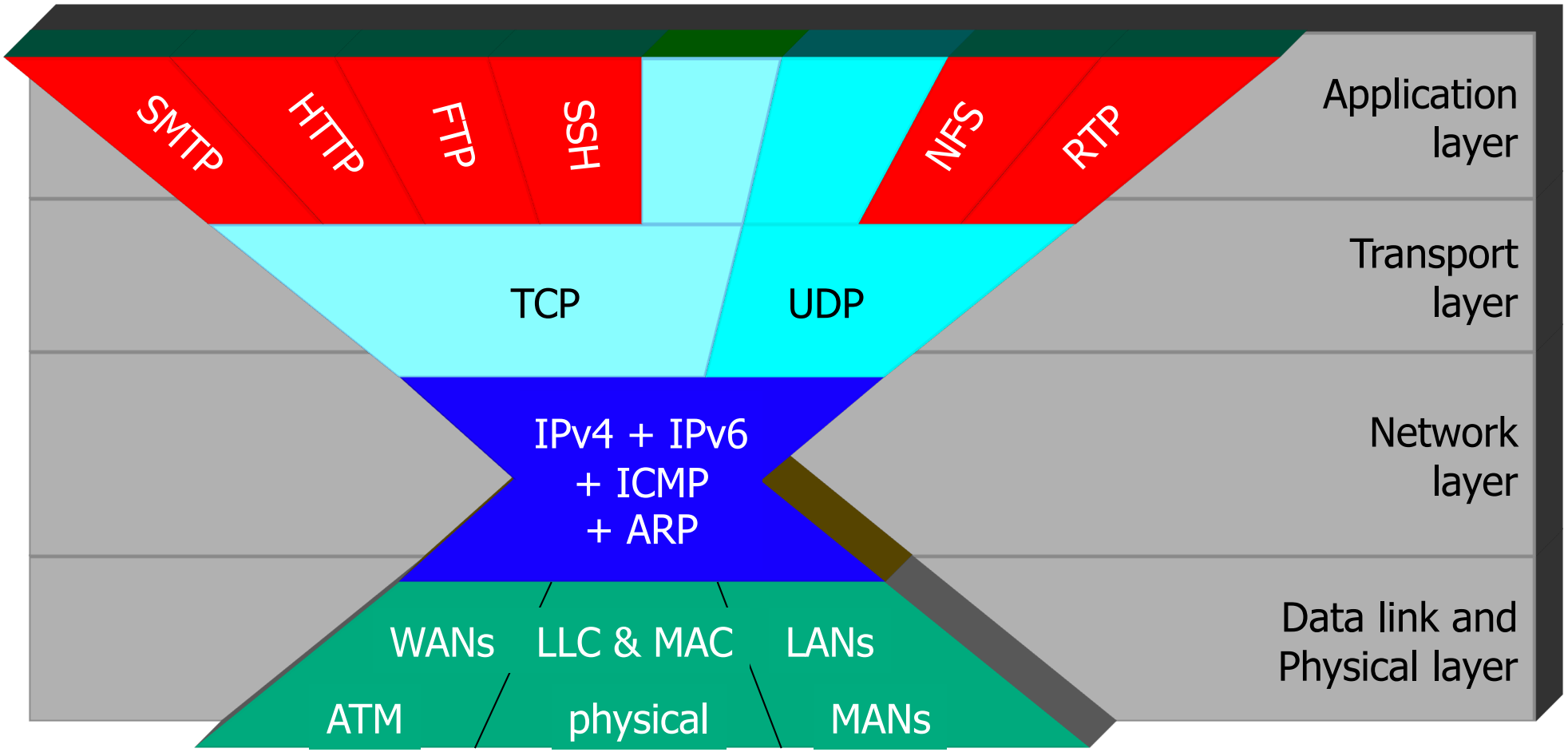
Protocol

- Protocol syntax: rules for formatting
- Protocol semantics: rules for actions in case of a message or event
- Note: semantics must be defined as behaviour of all communicating peers

Messages have lots of names

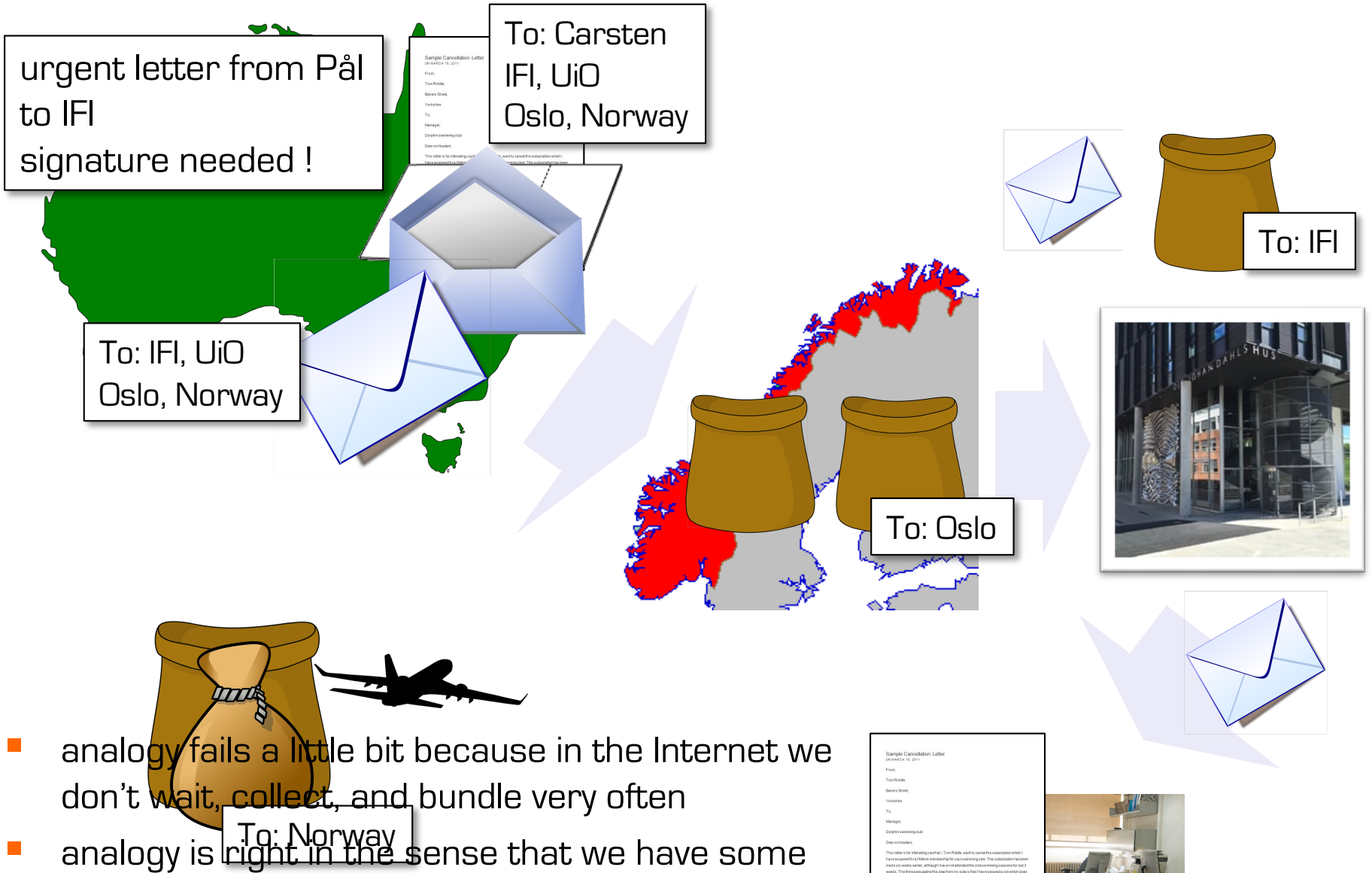
- protocol data unit (PDU)
- frame, packet, message, datagram
- symbol

Internet Protocol Stack



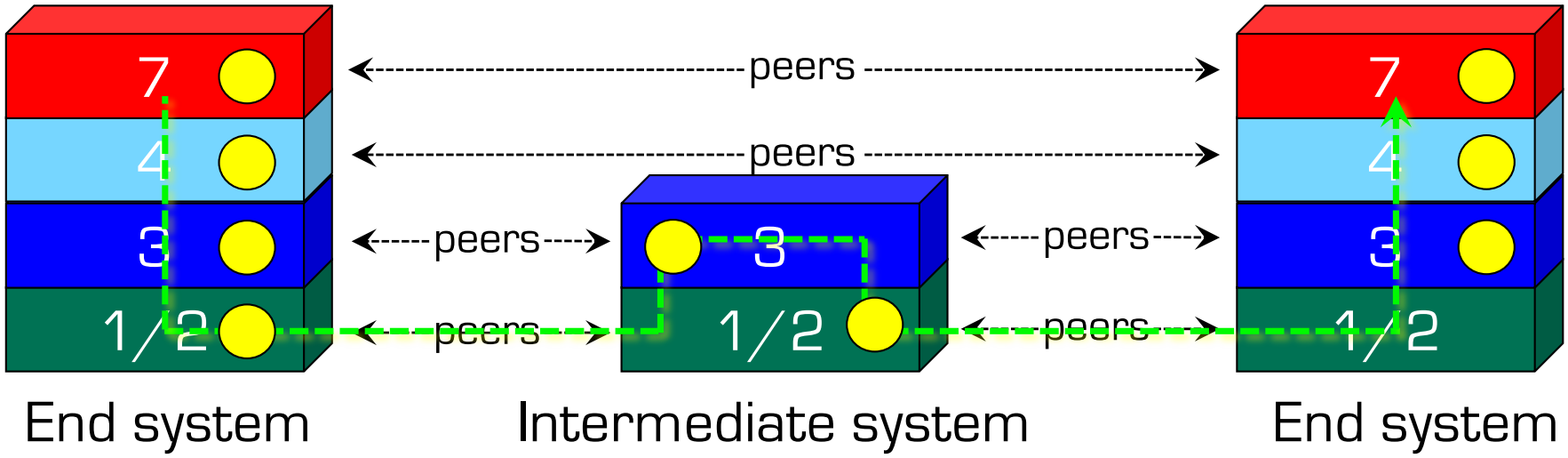
Nickname: “Hourglass Model”

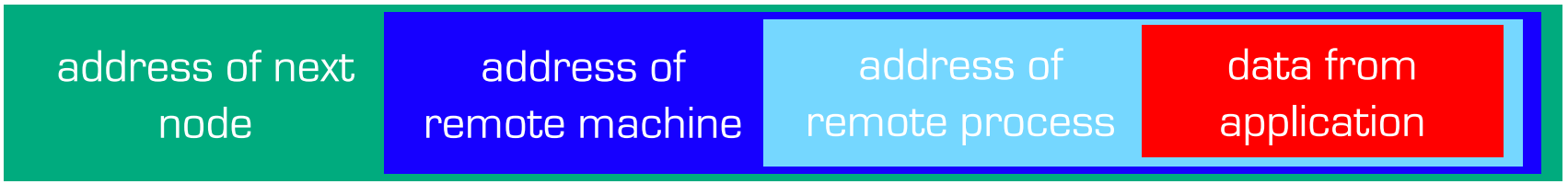
How to send packets



- analogy fails a little bit because in the Internet we don't wait, collect, and bundle very often
- analogy is right in the sense that we have some content to send, wrappers are put around it for sending, and repacking happens a lot

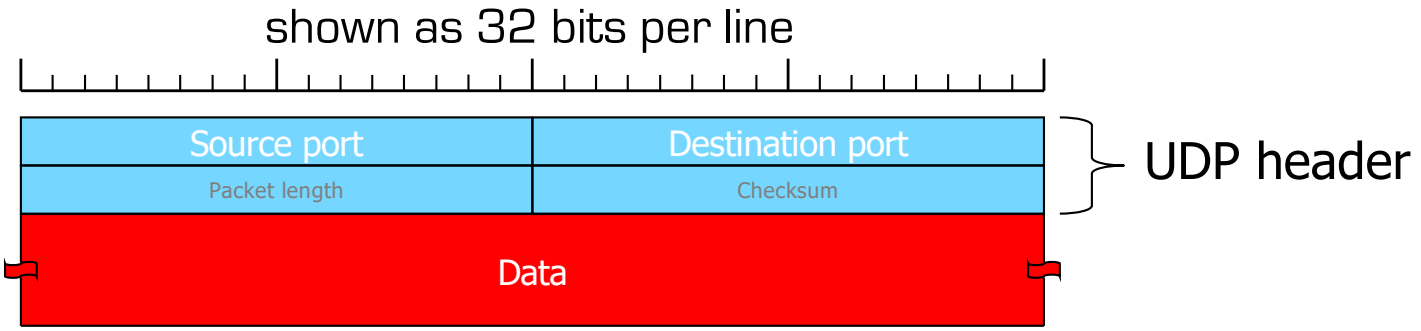
Data flow through the network





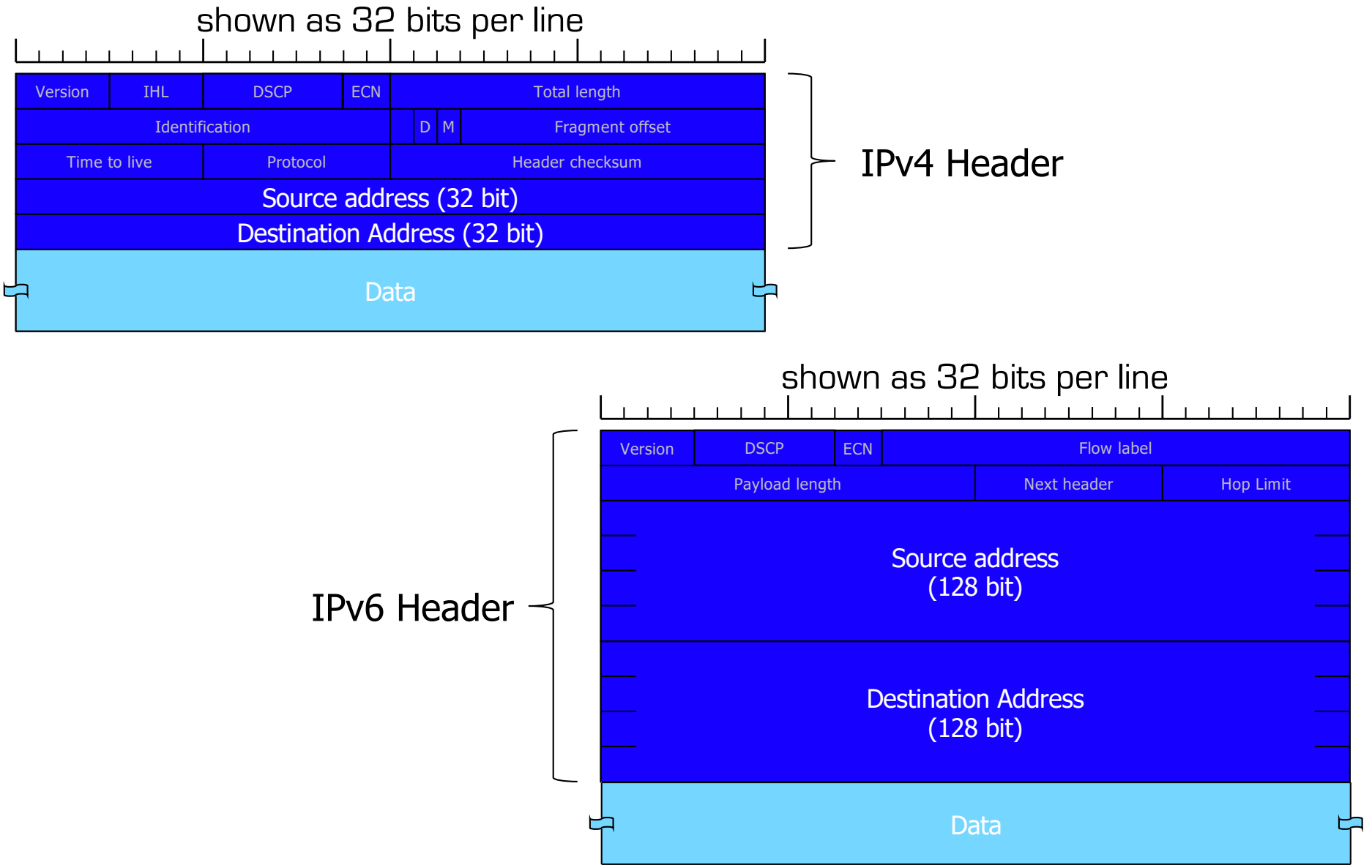
- Each sending N-entity at layer N adds N-protocol information
- ... which is important for its peer N-entity
- ... and the receiving N-entity removes it before passing the data to layer N+1

Transport layer header: UDP example

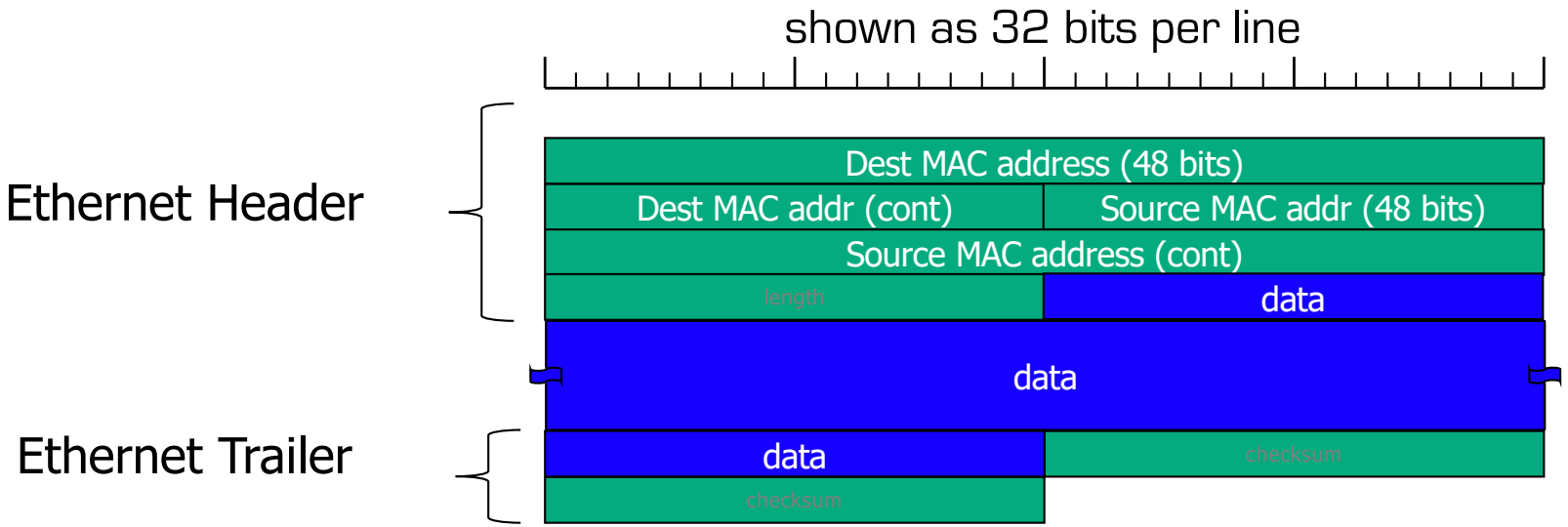


- **port**
- the term in Internet protocol for the address of a process on an end system
- the transport layer address
- Note: there are several transport layer protocols in the TCP/IP world, UDP is shown because it has the smallest header

Network layer headers: IPv4 and IPv6



Data link layer headers: Ethernet example



Network byte order

Why do we use
Big Endian
numbers in layers 2 – 4 ?

Big vs Little Endian

- Representing numbers
  - the decimal number 36
  - can be seen as 1*32 + 0*16 + 0*8 + 1*4 + 0*2 + 0*1
  - is identical to binary 100100
    - we prefer to think in whole bytes, and may write 00100100
  - is identical to hexadecimal `24 = 2*0x10 + 4*0x1`
    - for clarity we write 0x24
  - it is hard to transform directly from decimal to binary but easy to transform from hexadecimal to binary

sufficient to think about 4 bits at a time!

```
                    00100100
⇔                   0010 : 0100
⇔ 0*8 + 0*4 + 1*2 + 0*1 : 0*8 + 1*4 + 0*2 + 0*1
⇔                     2 : 4
⇔                    0x24
```

Big vs Little Endian

- when we want a bigger number than 255 = 0xff , we need more than 8 bits = 1 byte to store it
- 1 byte
  - 0 - 0xff
  - 0 - 255
- 2 bytes
  - 0 - 0xffff
  - 0 - 65.535
- 4 bytes
  - 0 - 0xffffffff
  - 0 - 4.294.967.295
- 8 bytes
  - 0 - 0xffffffffffffffff
  - 0 - 18.446.744.073.709.551.615

Big vs Little Endian

- It is very natural to write
  - "Hello"
  - and expect that it looks like this in code:

```
char buf[6];
buf[0]='H'; buf[1]='e'; buf[2]='l'; buf[3]='l'; buf[4]='o'; buf[5]='\0';
printf("%s ",buf); }
```

```
H e l l o
```

  - let's create a number from its byte-sized pieces!
  - when we use memory like this:

```
unsigned char byte[4];
byte[0]=0x0; byte[1]=0x0; byte[2]=0x2; byte[3]=0x14;
for( int i=0; i<4; i++ ) { printf("%x ",byte[i]); }
printf("\n");
int* ptr;
ptr = (int*)&byte[0];
printf("hex %x\n",*ptr);
```

```
00 00 02 14
```

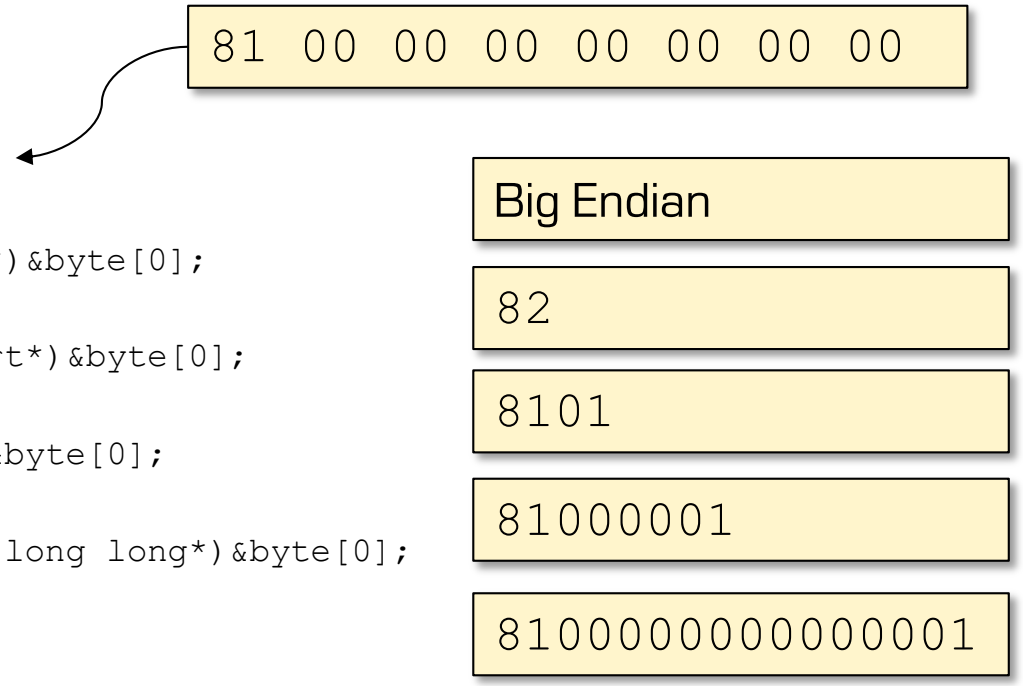
```
on Intel  → hex 14020000
on Sparc  → hex 214
```

Argument for Big Endian

- compatible with western-world writing direction
  - but when we use memory like this:

```
unsigned char byte[8];
byte[0] = 0x81;
for( int i=1; i<8; i++ ) byte[i] = 0;

unsigned char* ptr1 = (unsigned char*)&byte[0];
printf("%x\n", 1 + *ptr1);
unsigned short* ptr2 = (unsigned short*)&byte[0];
printf("%x\n", 1 + *ptr2);
unsigned int* ptr3 = (unsigned int*)&byte[0];
printf("%x\n", 1 + *ptr3);
unsigned long long* ptr4 = (unsigned long long*)&byte[0];
printf("%llx\n", 1 + *ptr4);
```



Argument for Little Endian

- easy to transform
  - when we use memory like this:

```
unsigned char byte[8];
byte[0] = 0x81;
for( int i=1; i<8; i++ ) byte[i] = 0;

unsigned char* ptr1 = (unsigned char*)&byte[0];
printf("%x\n", 1 + *ptr1);
unsigned short* ptr2 = (unsigned short*)&byte[0];
printf("%x\n", 1 + *ptr2);
unsigned int* ptr3 = (unsigned int*)&byte[0];
printf("%x\n", 1 + *ptr3);
unsigned long long* ptr4 = (unsigned long long*)&byte[0];
printf("%llx\n", 1 + *ptr4);
```

81 00 00 00 00 00 00 00

Little Endian

82

82

82

82

- cheap and easy to change the number of bytes used for an integer value

harder for the human mind
but faster to process



Bonus for Big Endian

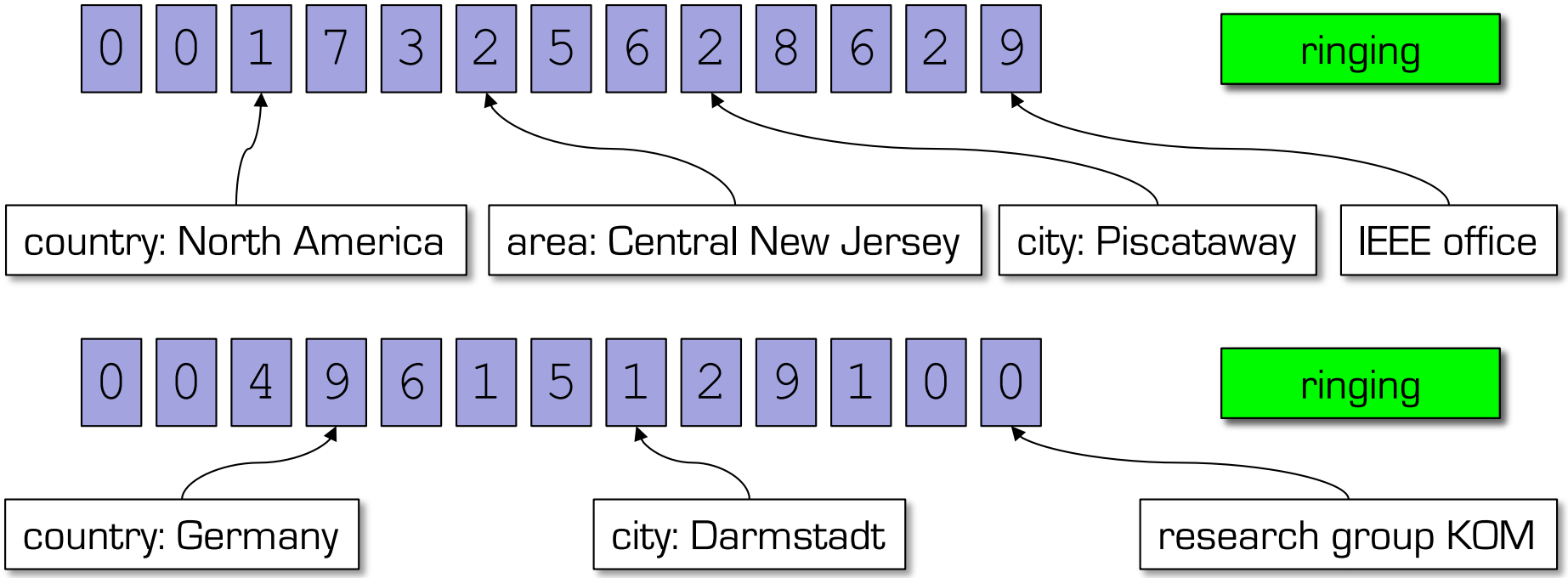
- L5 sends bytes to L4
- L4 passes packets to L3
- L3 adds a header for routing (and more)
- L3 passes frame content to L2
- L2 adds frame header for addressing (and more)
- L2 passes bits to L1
- L1 transfers bits

- L1 transfer starts at low memory addresses
  then continuing to high memory addresses

- speed matters
- headers are ***in front*** to process before all bits have arrived

Bonus for Big Endian

- analogue in telephone numbers



process first values that are sent first
because only last provider knows interpretation !

Bonus for Big Endian

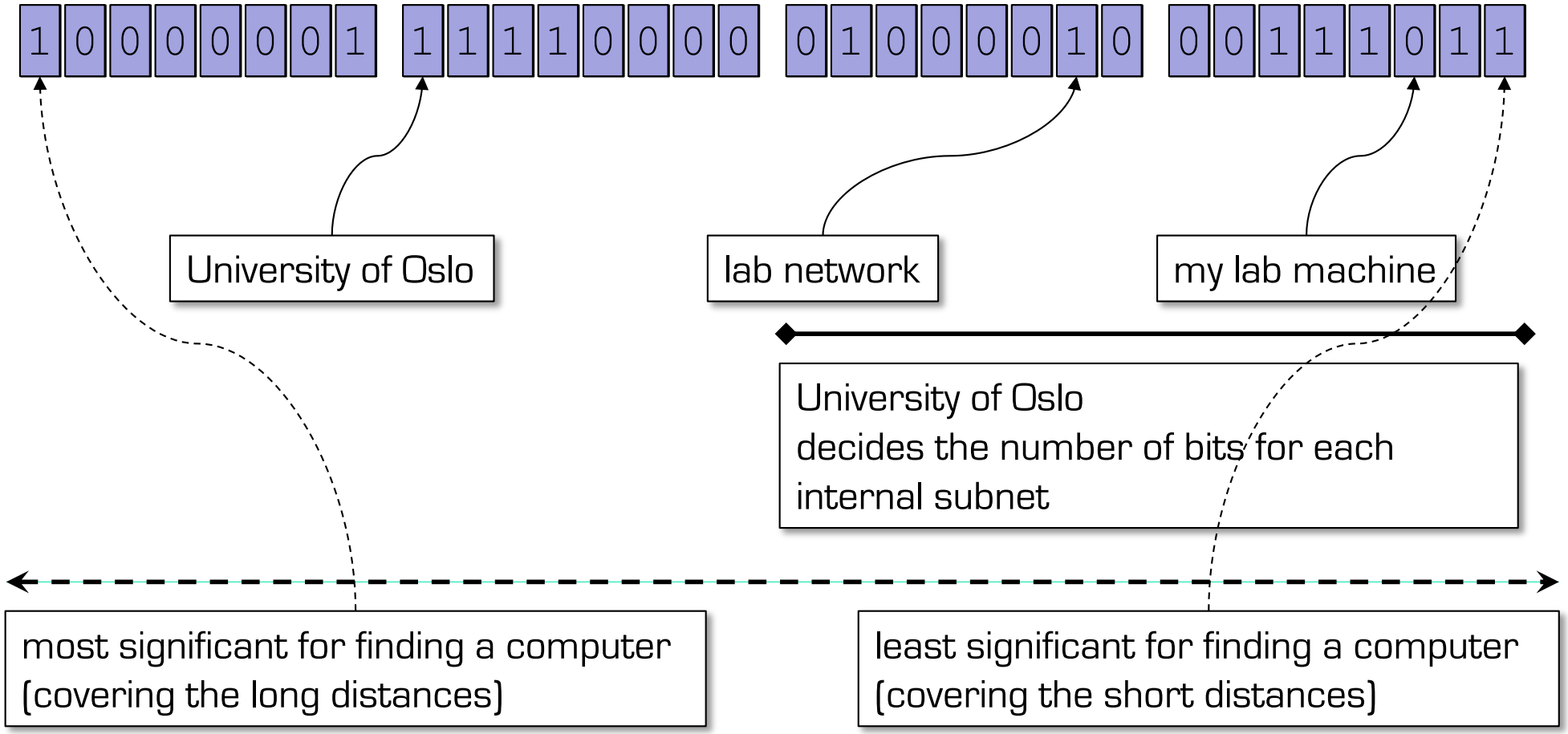
- my lab machine in our lab network
  129.240.66.59
  - this is called “dotted decimal notation”
  - this style is the usual way of writing the old IPv4 address

- 0x81 F0 42 3B
  - hexadecimal representation of the 4 bytes of the address

- `10000001 11110000 01000010 00111011`
  - binary representation of the 4 bytes

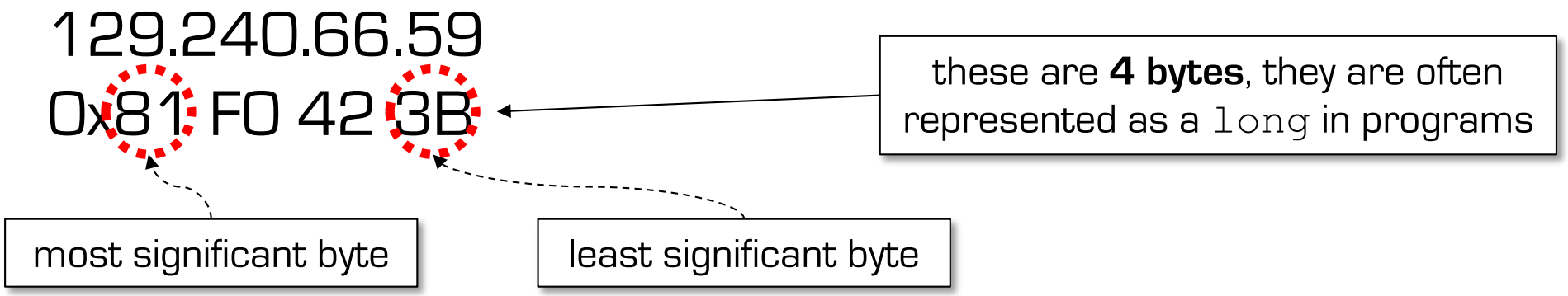
Bonus for Big Endian

- my lab machine in our lab network
  129.240.66.59
  0x81 F0 42 3B



Bonus for Big Endian

- my lab machine in our lab network



but building this address on a Little Endian machine is dangerous

Bonus for Big Endian

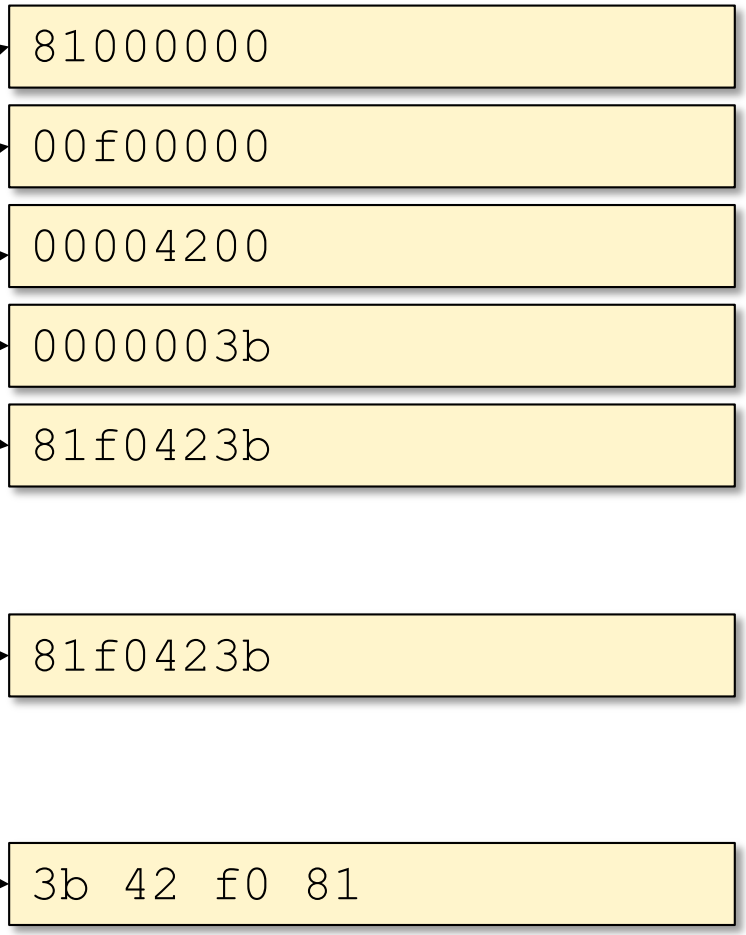
but building this address on a Little Endian machine is dangerous:

```
int main()
{
    int a = ( 0x81 << 24 );
    int b = ( 0xf0 << 16 );
    int c = ( 0x42 <<  8 );
    int d = 0x3b;
    int addr = a | b | c | d;

    unsigned char* ptr =
        (unsigned char*)&addr;

    printf("%x\n",addr);

    printf("%x ",ptr[0]);
    printf("%x ",ptr[1]);
    printf("%x ",ptr[2]);
    printf("%x\n",ptr[3]);
}
```

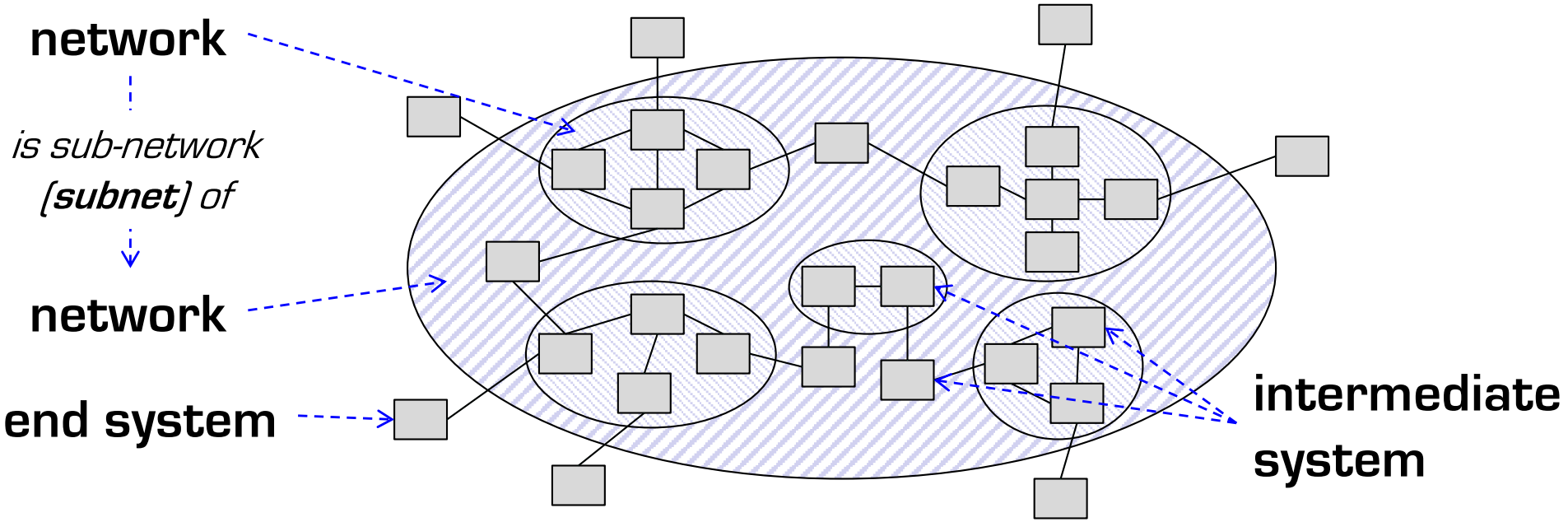


Addressing

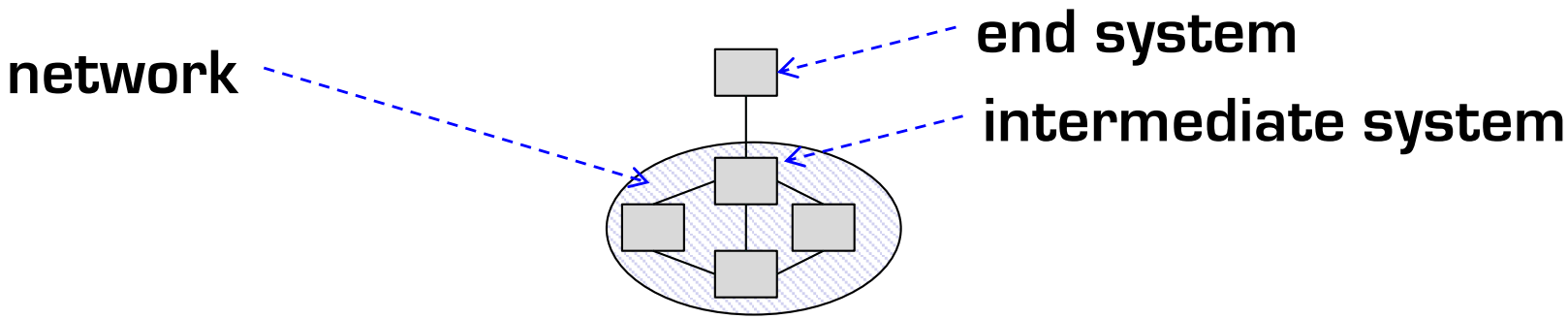
MAC addresses

in the TCP/IP model

Addressing

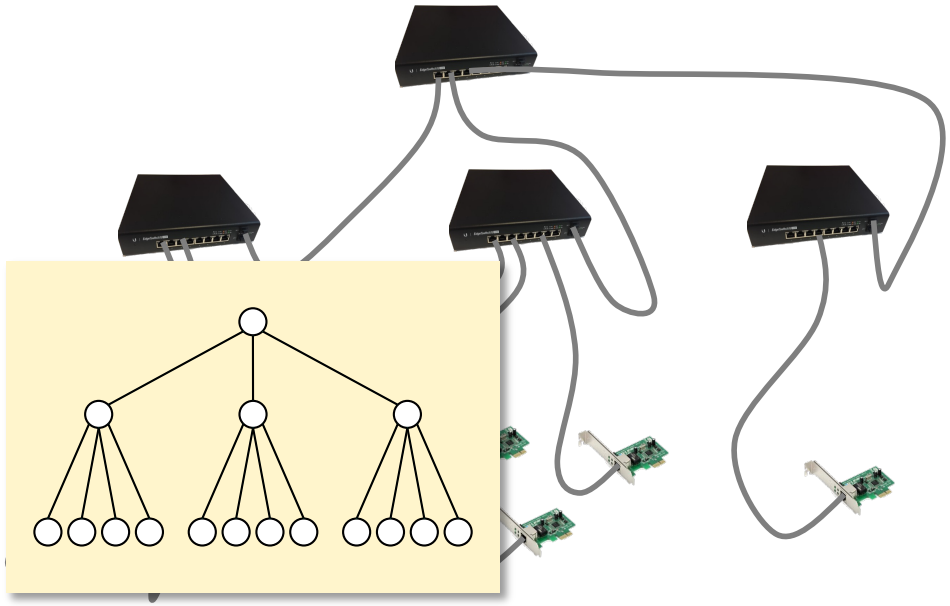

network
is sub-network
(subnet) of
network
end system
intermediate
system

Addressing

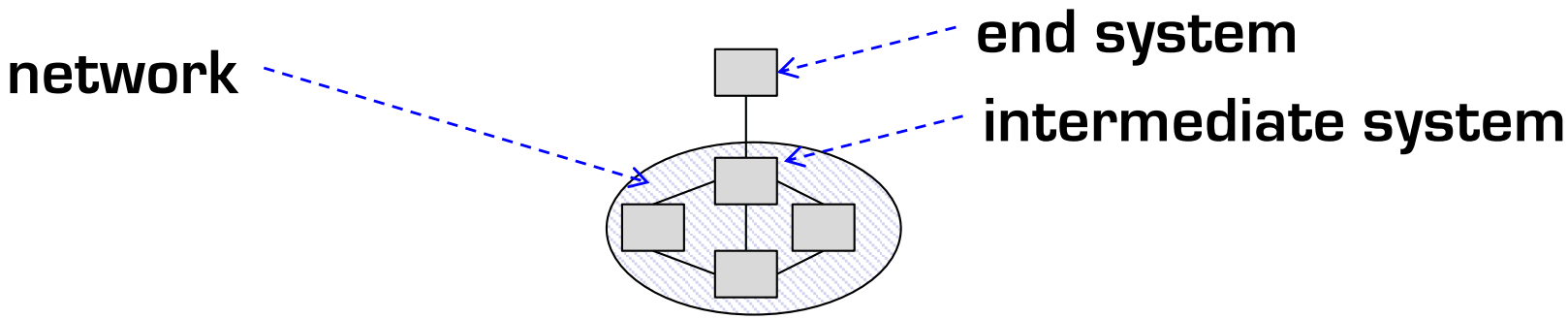


Point-to-point channels

- Gigabit Ethernet ("1GB Ethernet")

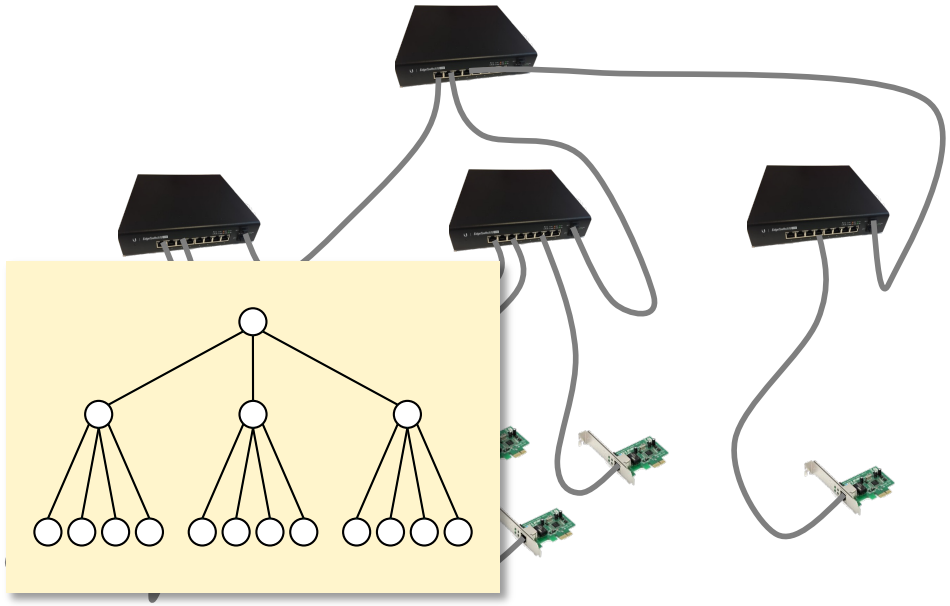


Addressing



Point-to-point channels

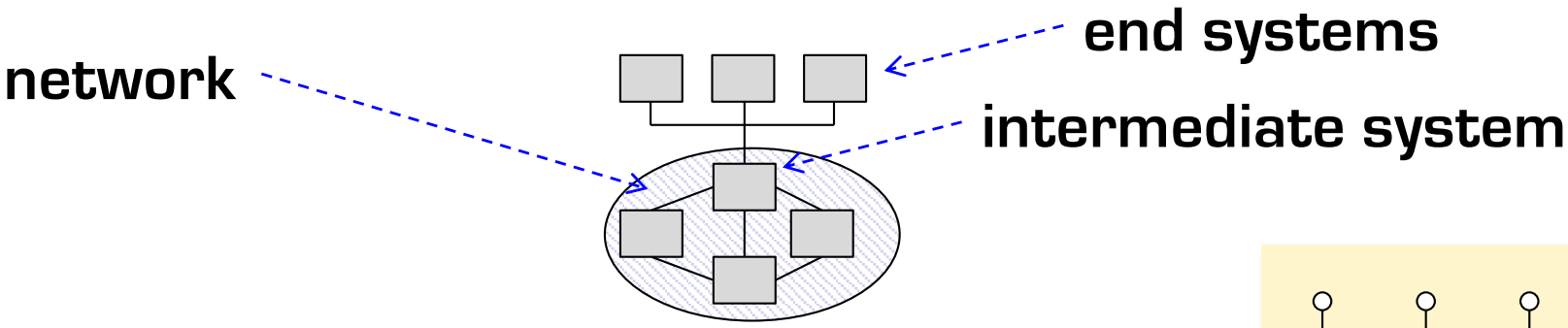
- Gigabit Ethernet ("1GB Ethernet")



MAC addresses

- are not required in a true point-to-point network
- when L2 passes frames to the correct L1 entity, the unique peer L1 entity will receive it

Addressing

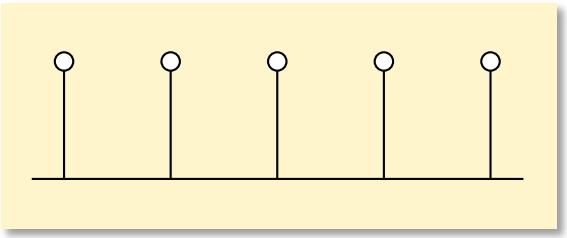


MAC addresses

- are important in a true broadcast network

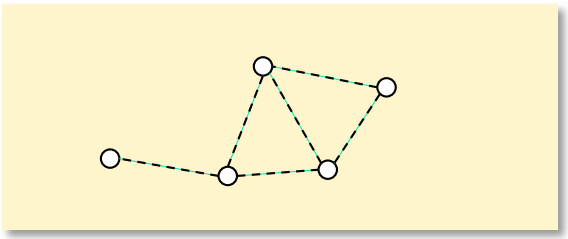
Challenge

- MAC addresses have only local meaning
- nodes on the “other side” of an IS may not know them

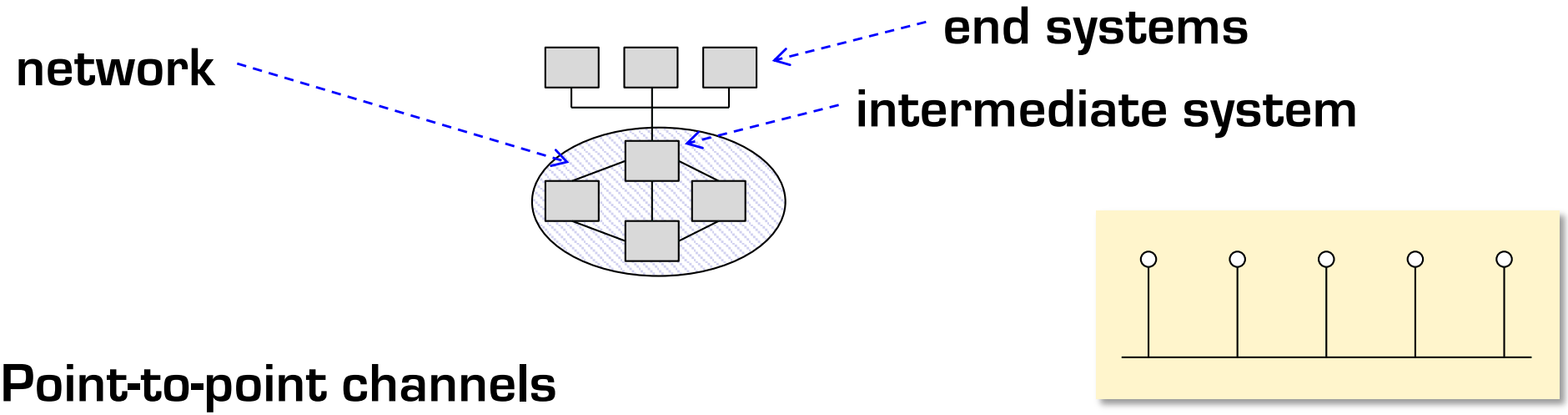


Broadcasting channels

- Cable
  - old-fashioned Ethernet
- Radio
  - WiFi (IEEE 802.11)

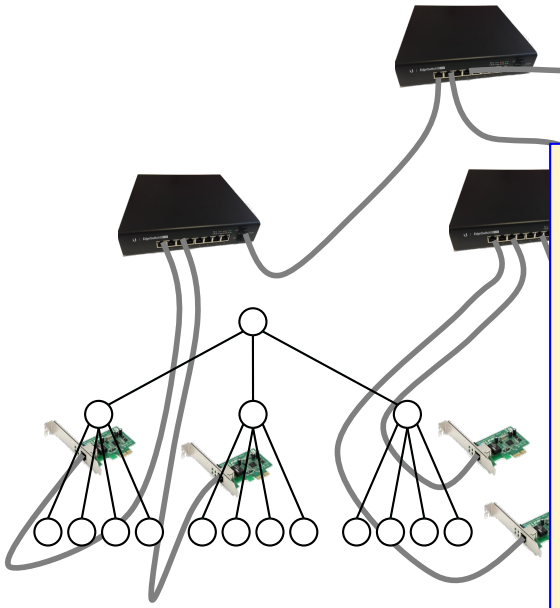


Addressing



Point-to-point channels

- Gigabit Ethernet ("1GB Ethernet")



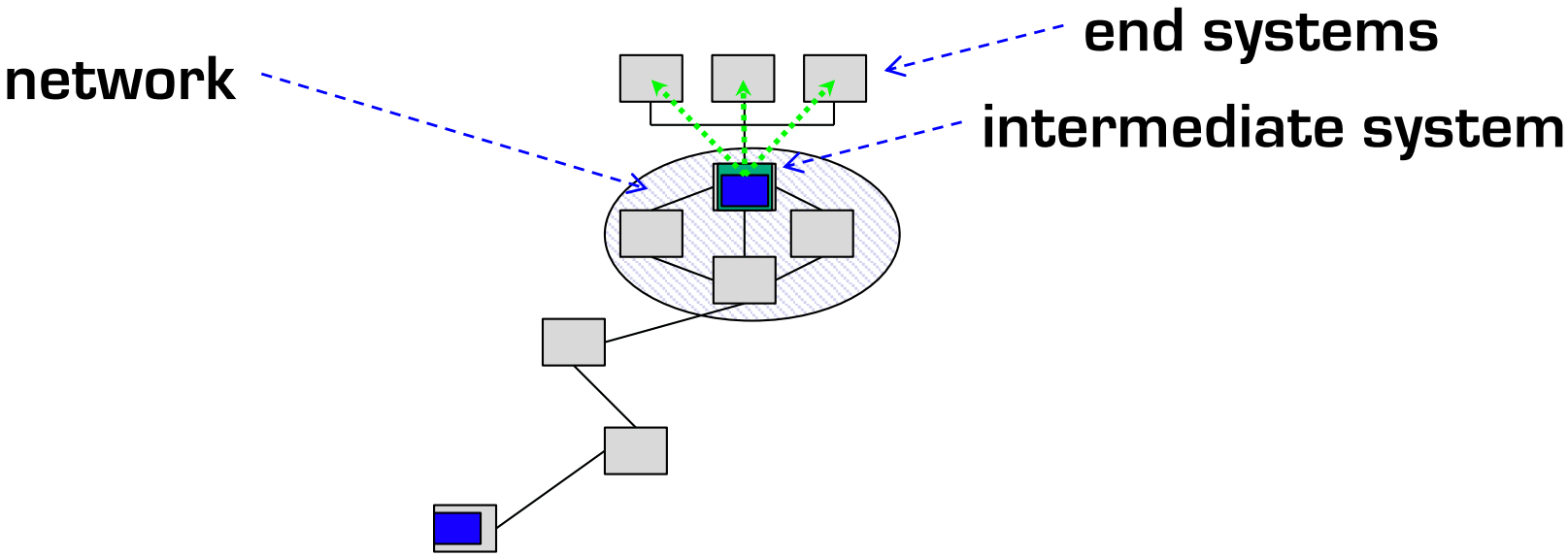
Broadcasting channels

- Cable
  - old-fashioned Ethernet

Actually, Gigabit Ethernet behaves like old-fashioned Ethernet. 2 good reasons:

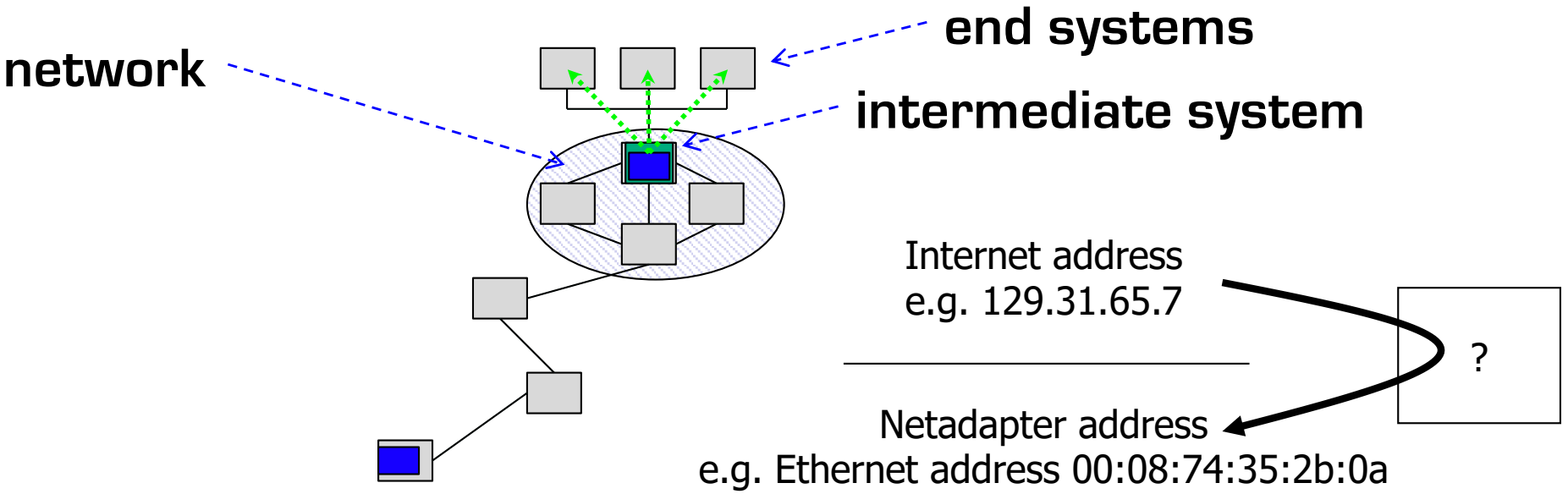
- backward compatibility
- no management needed when a PC is unplugged in one place and plugged back in elsewhere

Address resolution

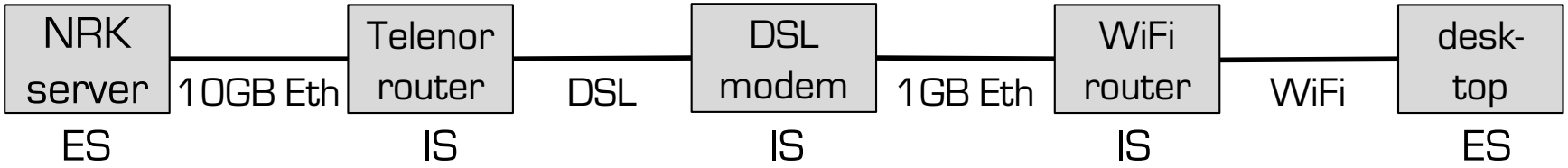


- given a packet with an L3 address,
  an IS must find the correct L2 address for this packet quickly
- what are the options?

Address resolution

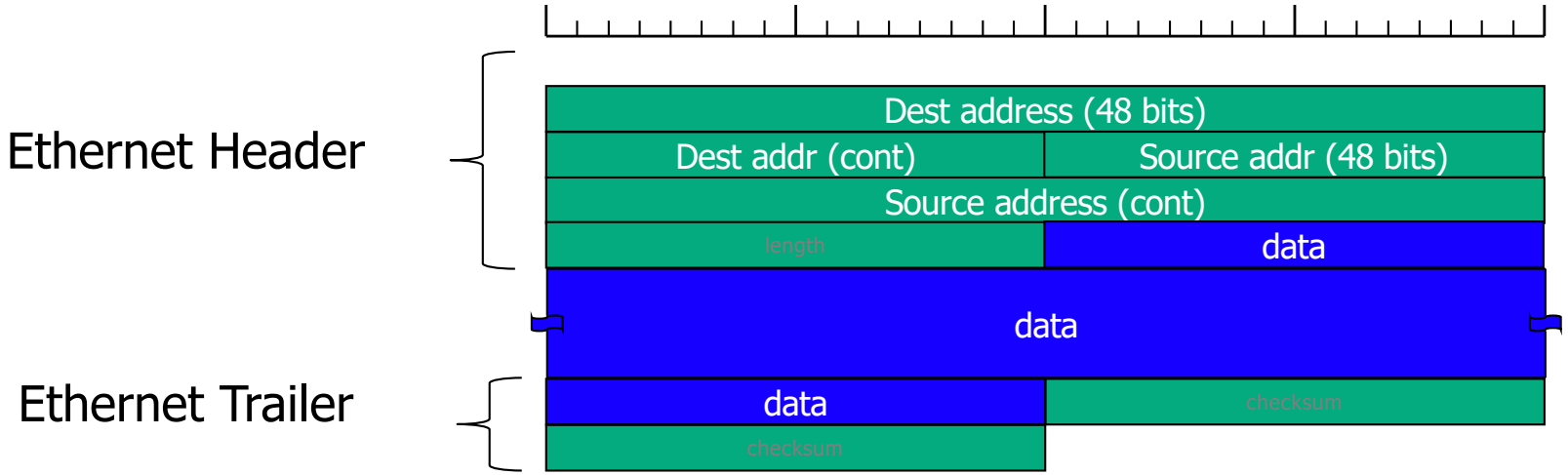


- Problem
  - Potentially every link can use a different L2 protocol



  - Different L2 protocols have different address styles
  - IP address must be mapped onto the MAC address
    48 bit for Ethernet and WiFi, DSL may use 20 or 48 bits

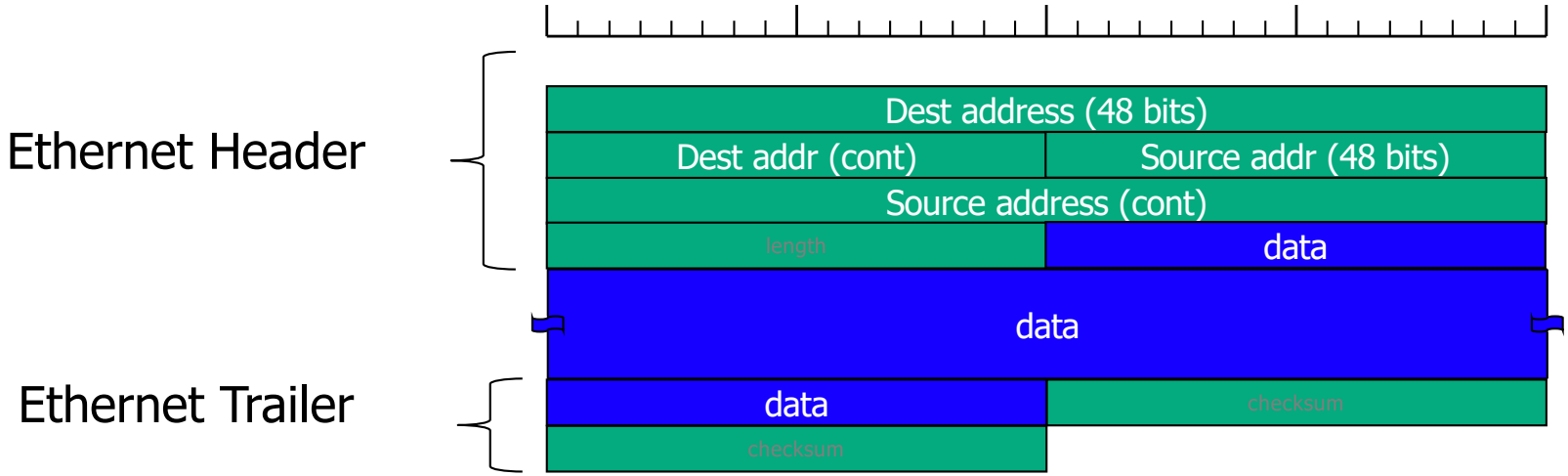
Address resolution: Ethernet example



- MAC address structure
  - Ethernet and WiFi are L2 layers using “EUI-48”: Extended Unique Identifier with 48 bits
  - 6 bytes, written like this: `f2:18:98:3a:b8:97`
  - to recognize easily that the text is supposed to mean a MAC address

- Ethernet MAC addresses should be globally unique

Address resolution: Ethernet example



- IANA and IEEE decide how to split the address space
  - first 3 bytes explain whether an address is special

  OR
  - first 3 bytes determine who owns the address range
  - e.g.:
    - F0:18-98 : Apple, Inc.
    - 78:45:C4: Dell Inc.
    - 00:50:56: VMWare, Inc.
    - B8:AC:6F: Dell Inc.

IANA - Internet Assigned Number Authority
IEEE - Institute of Electrical and Electronics Engineers

Addresses

- my lab machine's MAC addresses:

```
griff@msi:~$ ifconfig -a
enp5s0    Link encap:Ethernet  HWaddr 4c:cc:6a:dc:22:8b
          inet addr:10.0.0.89  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::4ecc:6aff:fedc:228b/64 Scope:Link
          inet6 addr: fd00::4ecc:6aff:fedc:228b/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1129075 errors:0 dropped:0 overruns:0 frame:0
          TX packets:448210 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:678291511 (678.2 MB)  TX bytes:151105424 (151.1 MB)
          Interrupt:19

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:81839 errors:0 dropped:0 overruns:0 frame:0
          TX packets:81839 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:86621380 (86.6 MB)  TX bytes:86621380 (86.6 MB)

wlp3s0    Link encap:Ethernet  HWaddr 9c:b6:d0:19:1b:87
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:5727 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1268 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1571055 (1.5 MB)  TX bytes:192827 (192.8 KB)
```

Ethernet MAC
(Micro-Star, Taiwan)

WiFi MAC
(Rivet Networks, USA)

Addresses

- MAC addresses known to my lab machine

```
griff@msi:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
P-2812HNU.lan            ether   cc:5d:4e:07:d7:03   C                     enp5s0
P-2812HNU.lan            ether   cc:5d:4e:07:d7:03   C                     enp5s(
griff@msi:~$
```

DSL Modem's Ethernet MAC

- MAC addresses know to nordur, one of the login.ifi.uio.no machines, in 2024 (incomplete)

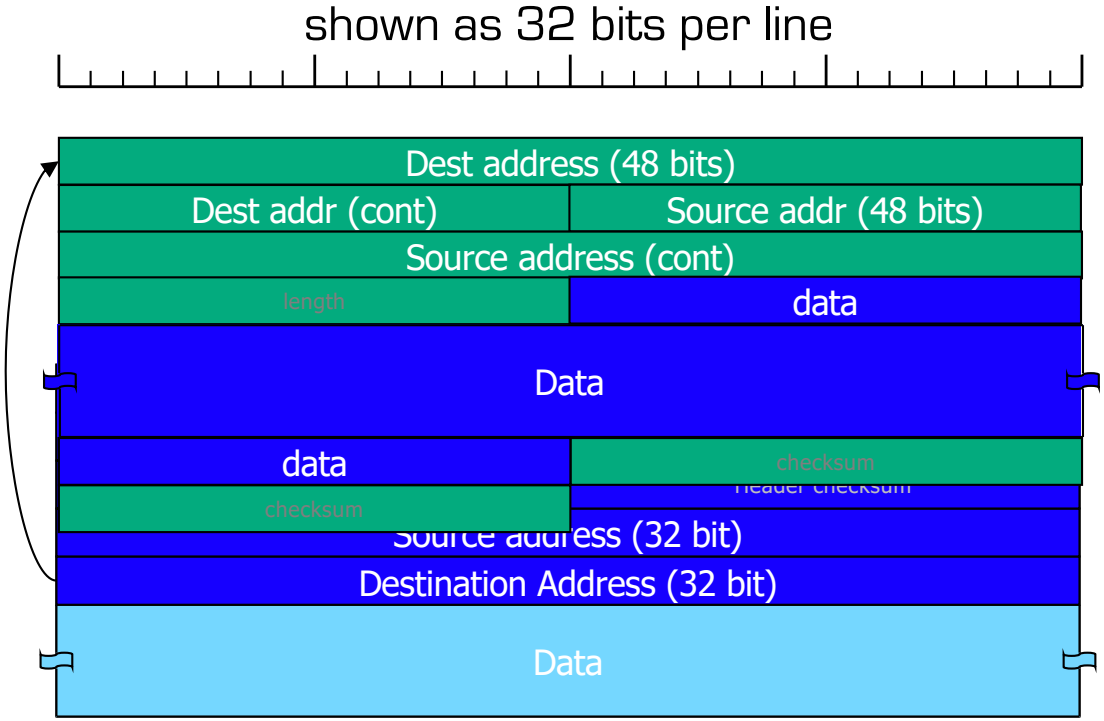
```
[griff@nordur ~]$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
angelica.ifi.uio.no      ether   5c:26:0a:fc:93:cc   C                     bond0
modir.ifi.uio.no         ether   b8:ac:6f:93:d0:7d   C                     bond0
skrifaskildabod.ifi.uio  ether   00:50:56:a9:19:f2   C                     bond0
bifrost.ifi.uio.no               (incomplete)                              bond0
spurningmin.ifi.uio.no   ether   00:50:56:a9:c6:51   C                     bond0
austur.ifi.uio.no        ether   b8:ac:6f:d2:6b:a0   C                     bond0
ljosmodirin.ifi.uio.no   ether   00:50:56:80:d4:6f   C                     bond0
synahverning.ifi.uio.no  ether   00:50:56:9f:29:bb   C                     bond0
breytavefur.ifi.uio.no   ether   00:50:56:a9:b8:c7   C                     bond0
heimdallr.ifi.uio.no     ether   00:50:56:80:71:96   C                     bond0
mikilvaegimest.ifi.uio.  ether   00:50:56:80:7d:29   C                     bond0
verkefni.ifi.uio.no      ether   00:50:56:a9:82:7f   C                     bond0
reglulega.ifi.uio.no     ether   00:50:56:9f:75:15   C                     bond0
athugidvefur.ifi.uio.no  ether   00:50:56:a9:a8:c0   C                     bond0
utgafamidlara.ifi.uio.n  ether   00:50:56:a9:13:cc   C                     bond0
asgard.ifi.uio.no                (incomplete)                              bond0
odyr.ifi.uio.no          ether   00:25:64:ff:06:a8   C                     bond0
bragdprof.ifi.uio.no     ether   00:50:56:a9:11:03   C                     bond0
prenta.ifi.uio.no        ether   00:50:56:a9:f1:90   C                     bond0
vestur.ifi.uio.no        ether   b8:ac:6f:d2:69:48   C                     bond0
virfir.ifi.uio.no        ether   b8:ac:6f:93:d1:7a   C                     bond0
```



Address Resolution

1st idea: direct mapping

the 32 bit destination IP address would fit into
the 48 bit destination MAC address



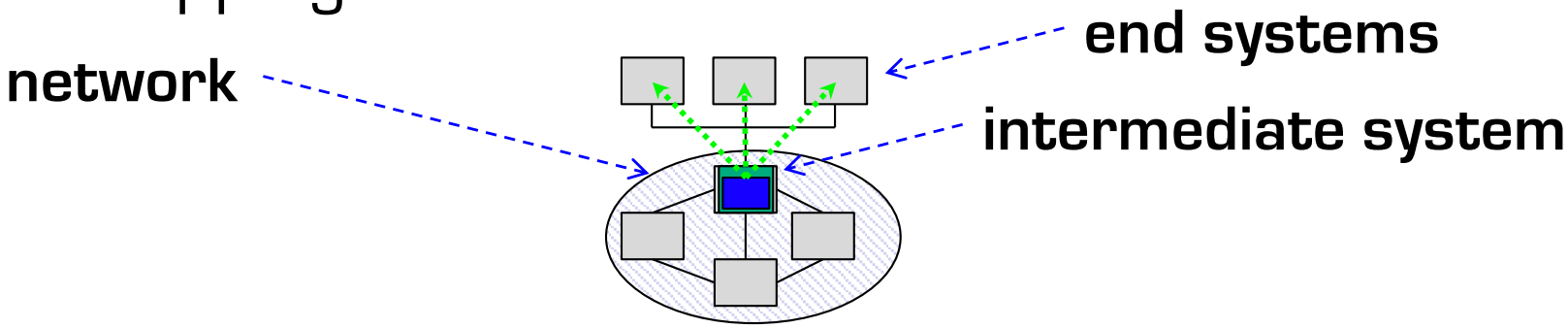
but:

- there is a new MAC address for every pair of direct neighbours
- need to re-write the destination IP address on every IS
- but IP addresses are globally unique

⇒ **does not work for the Internet**

Address Resolution

2nd idea: mapping table



every node maintains a table that maps

IP address ↔ MAC address

for every network interface and

for every directly reachable node (L2 neighbour)

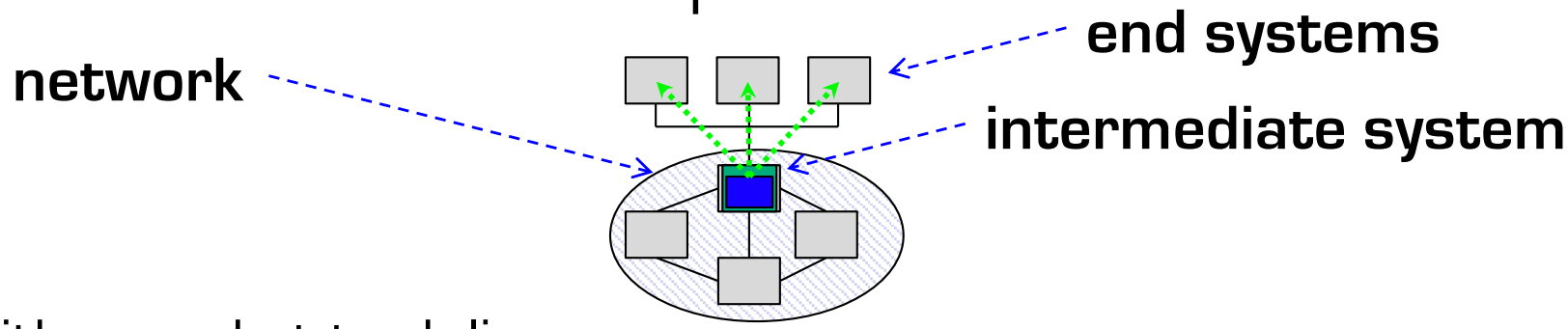
idea 2.1: manually maintained by people

- a lot of work, but not unrealistic – IFI allows only well-known MAC addresses in well-known network plugs – **could be used for this but is not**

idea 2.2: established by broadcasts from stations

Address Resolution

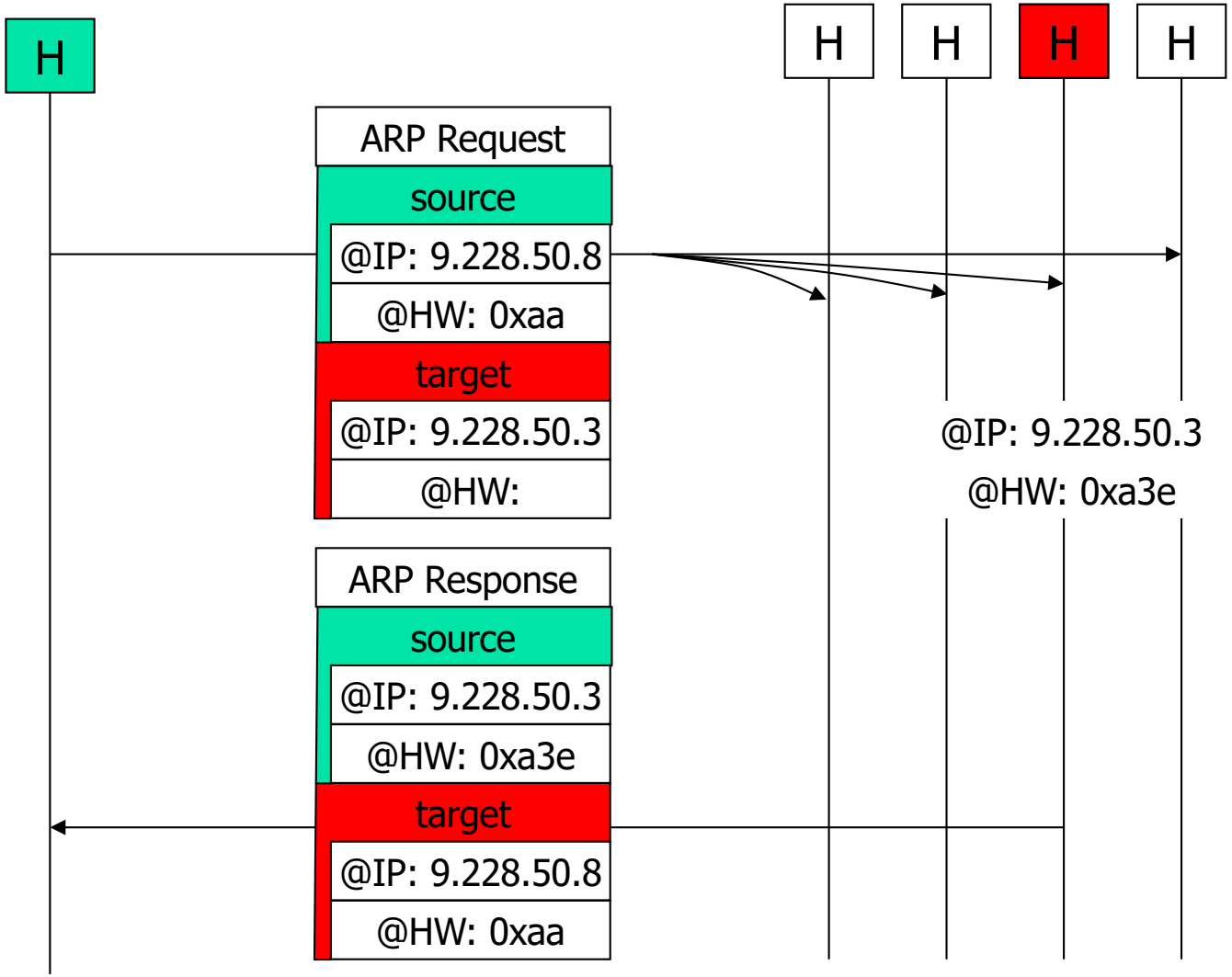
3nd idea: address resolution protocol



node with a packet to deliver:

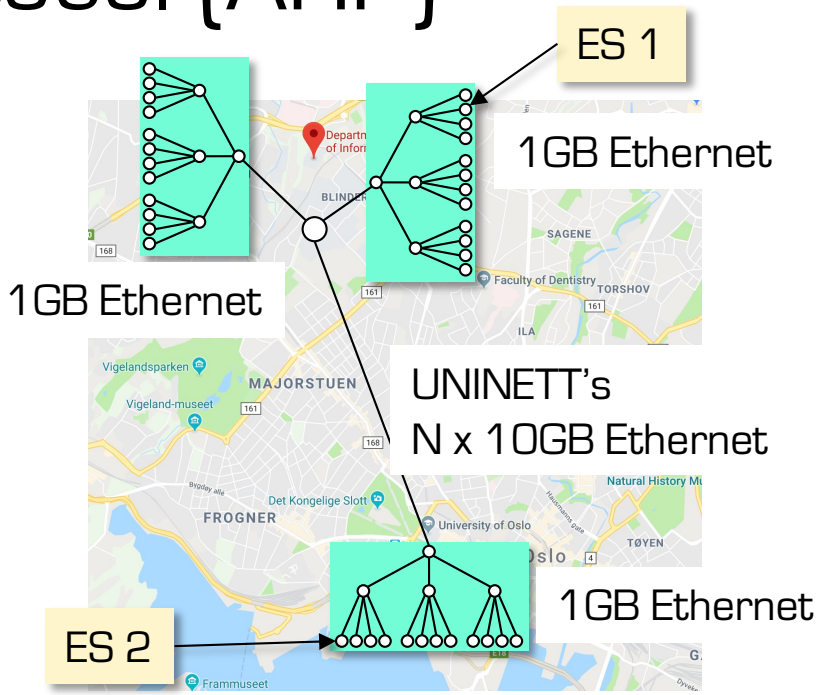
```
if a local cache contains IP address ↔ MAC address
    send packet & update cache removal timeout
else
    send broadcast to all stations
    "Who has IP address?"
    if one node responds
        add IP address ↔ MAC address mapping to cache
        set timeout for removal from cache to some minutes
        send packet
    else
        drop packet
```

Address Resolution Protocol (ARP)



Address Resolution Protocol (ARP)

- End system not directly reachable by broadcast
- Example: ES 1 to ES 2
  - ARP would not receive a response
    - Ethernet broadcast is not rerouted over a router



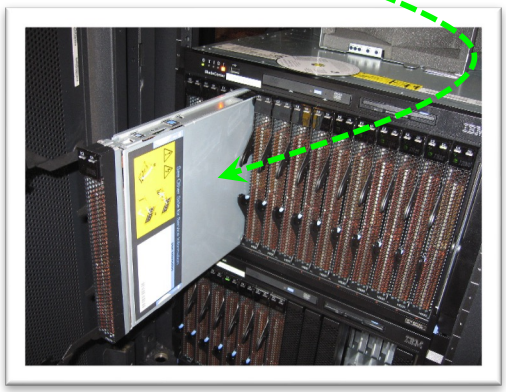
- L2 solution: proxy ARP
  - the local router knows all remote networks with their respective routers
    - responds to local ARP
  - local ES 1 sends data for ES 2 always to the local router, this router forwards the data (by interpreting the IP address contained in the data)

- L3 solution: remote network address is known
  - local ES 1 sends data to the appropriate remote router
  - local router forwards packets

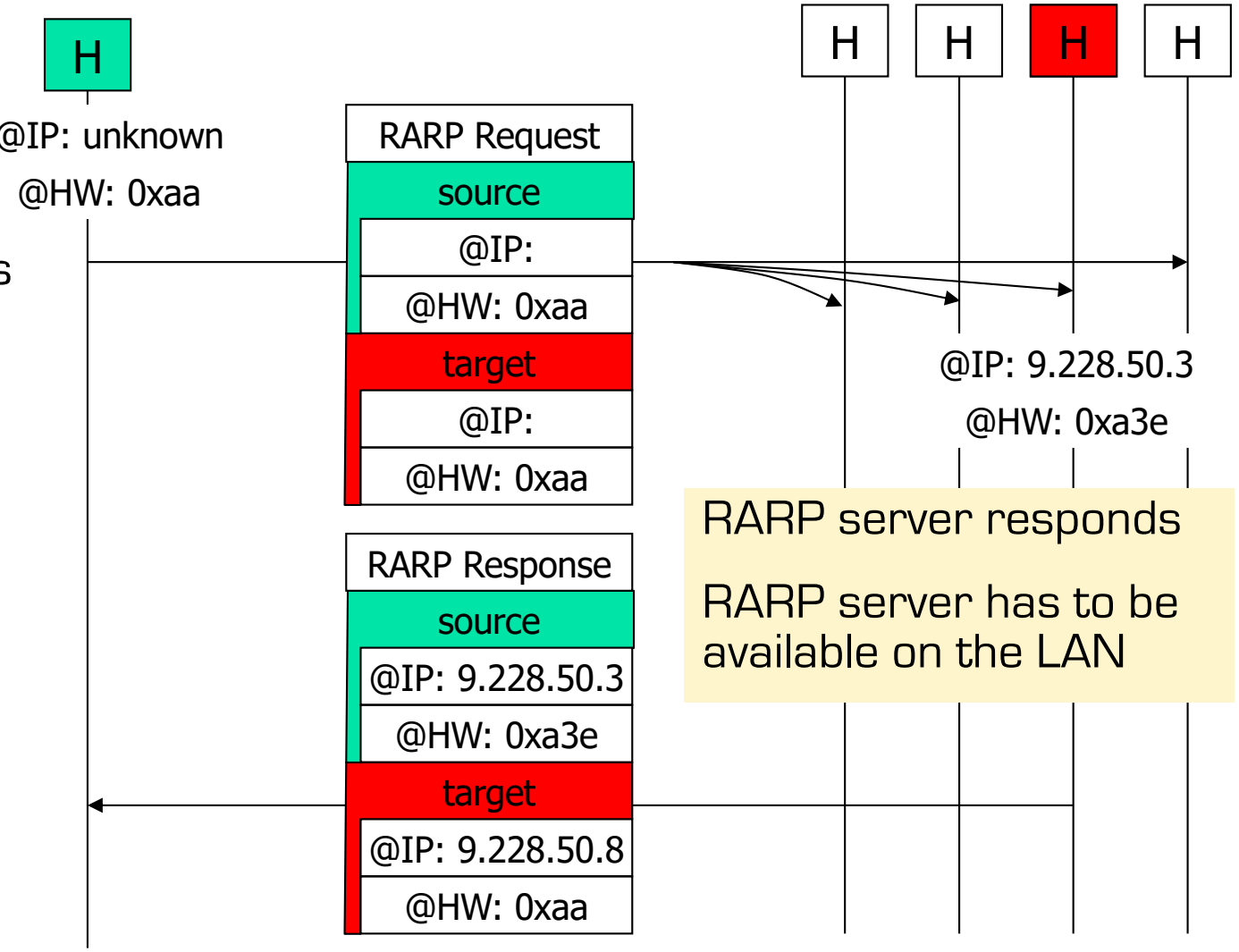
Reverse Address Resolution Protocol (RARP)

Retrieve Internet address from knowledge of hardware address

Application today:
blades in large clusters are physically moved



for other uses mostly replaced by newer protocols BOOTP and DHCP



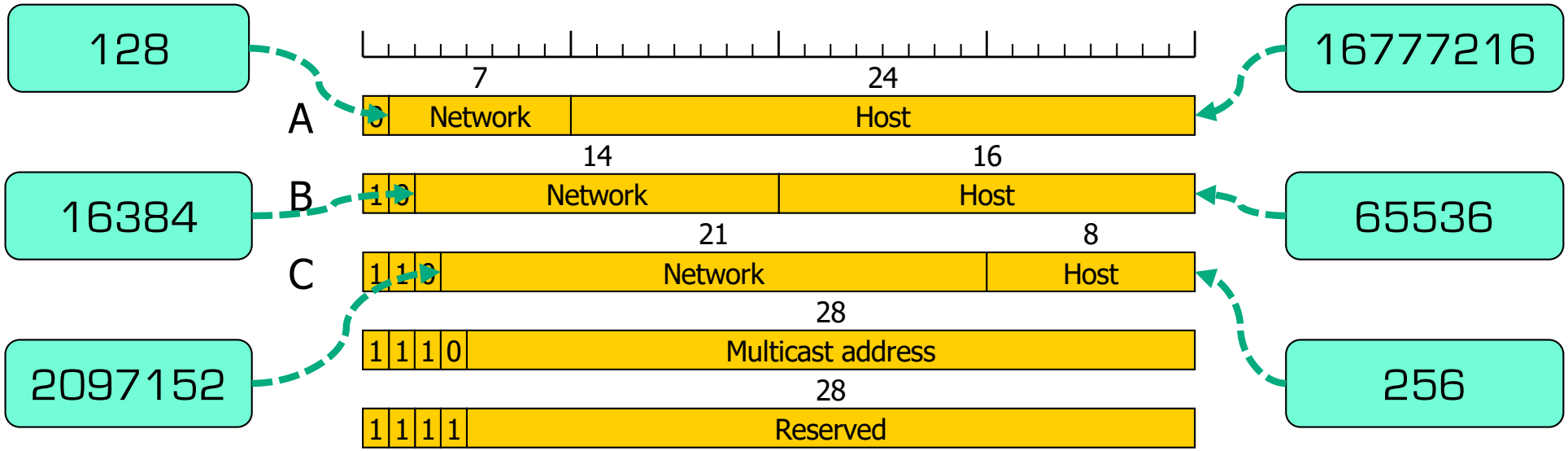
Addressing

IP addresses

in the TCP/IP model

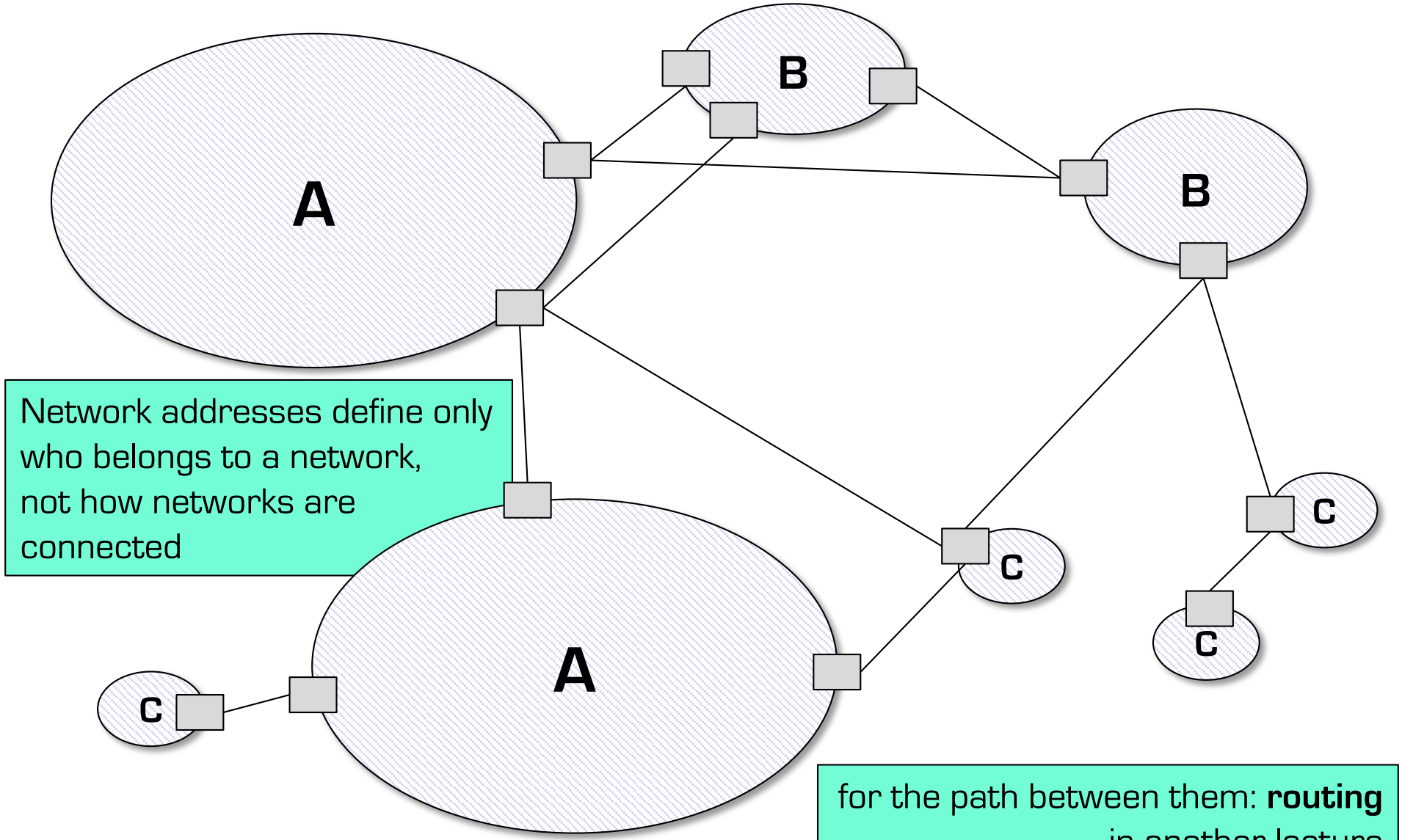
Internet Addresses and Internet Subnetworks

- Original global addressing concept for the Internet
  - For addressing end systems and intermediate systems
  - each network interface (not ES) has its own unique address
  - 5 classes



- ICANN (Internet Corporation for Assigned Numbers and Names)
  - manages network numbers
  - delegates parts of the address space to regional authorities

Internet Addresses and Internet Subnetworks



Network addresses define only who belongs to a network, not how networks are connected

for the path between them: **routing** in another lecture

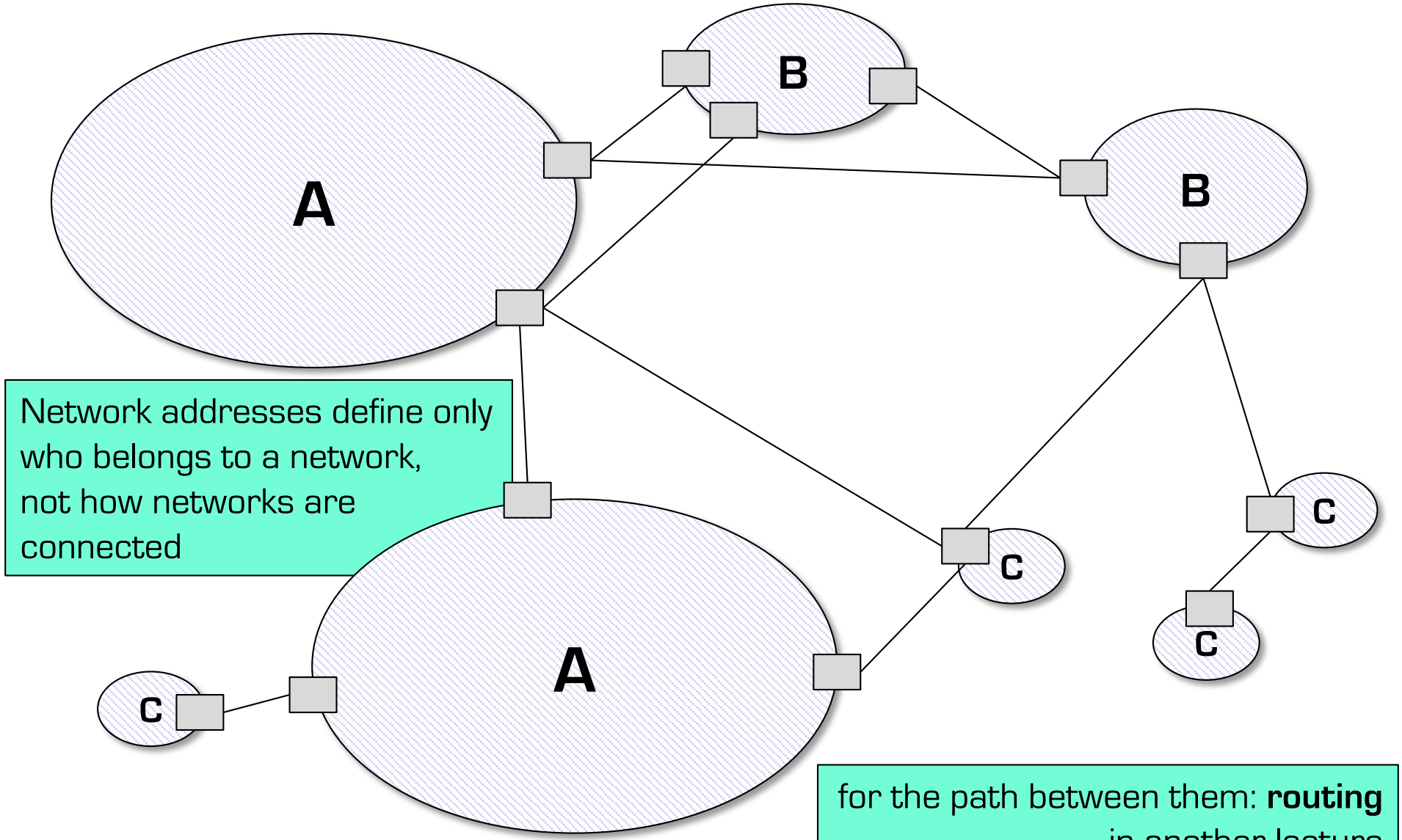
Internet Address and Internet Subnetworks

- Networks grow and should be somehow structured
  - several networks instead of one preferable
  - but getting several address areas is hard
    - since address space is limited
    - e.g., university may have started with class B address, doesn't get second one

- Problem
  - class A, B, C refer to one network, not collection of LANs

⇒ Allow a network to be split into several parts
  - for internal use
  - still looks like single network to outside world

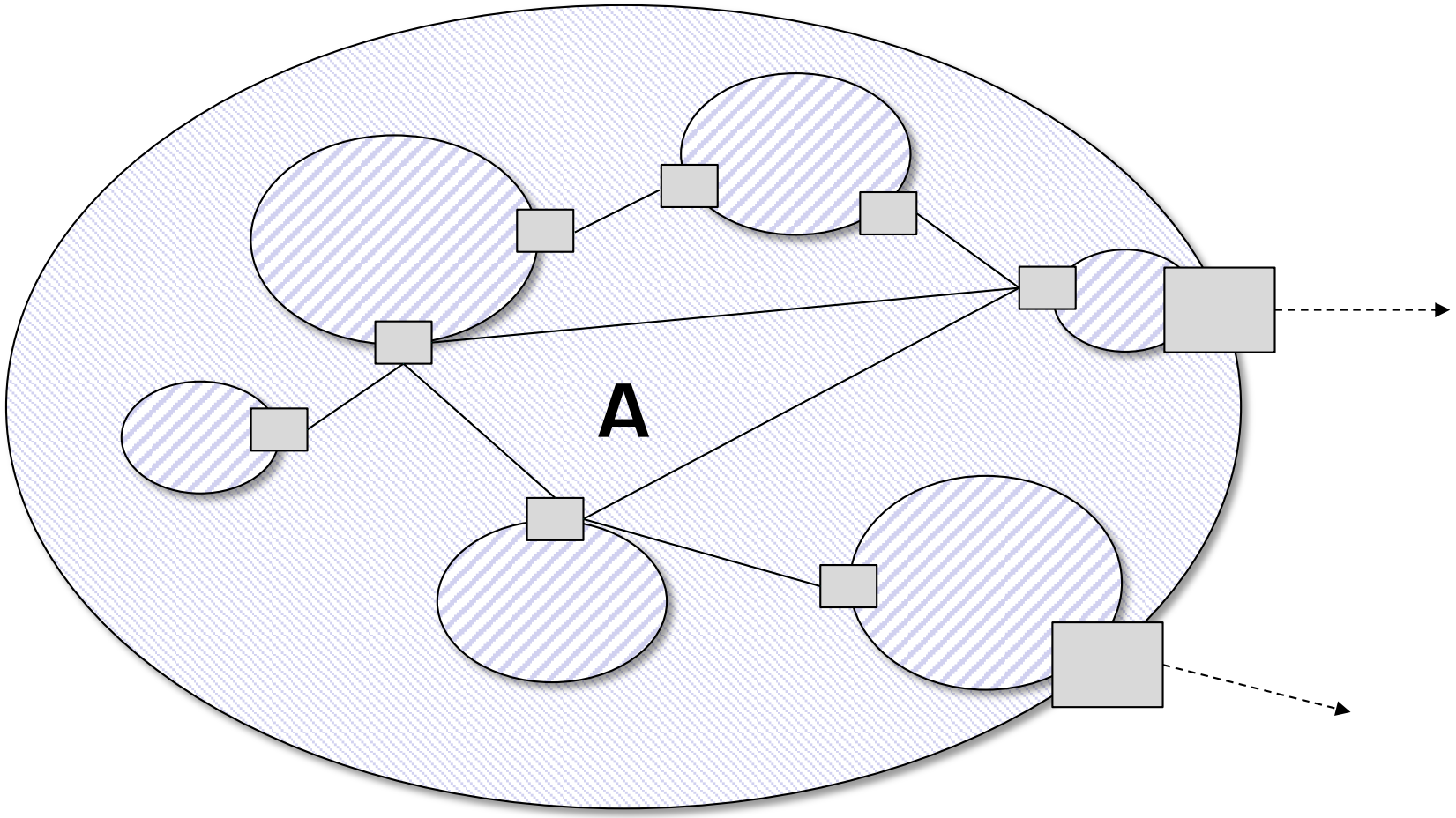
Internet Addresses and Internet Subnetworks



Network addresses define only who belongs to a network, not how networks are connected

for the path between them: **routing** in another lecture

Internet Addresses and Internet Subnetworks

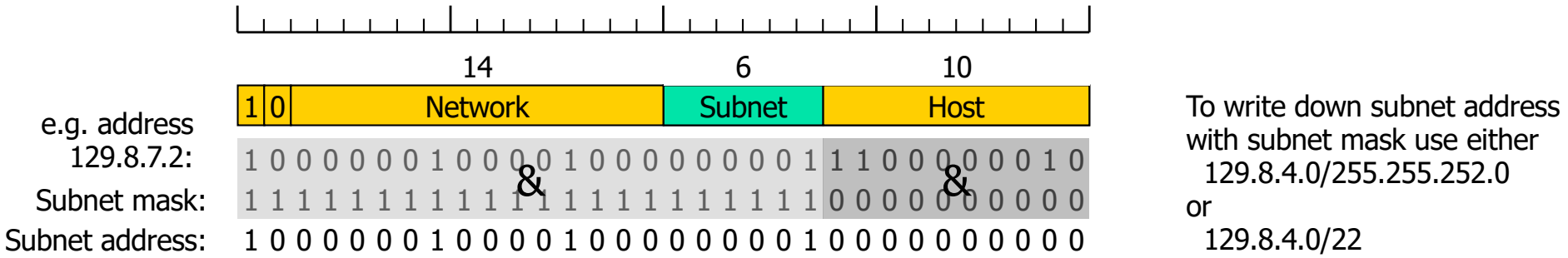


for the path inside one of them: also **routing**

but how can we carve subnets out of the class A, B or C network?

Internet Address and Internet Subnetworks

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



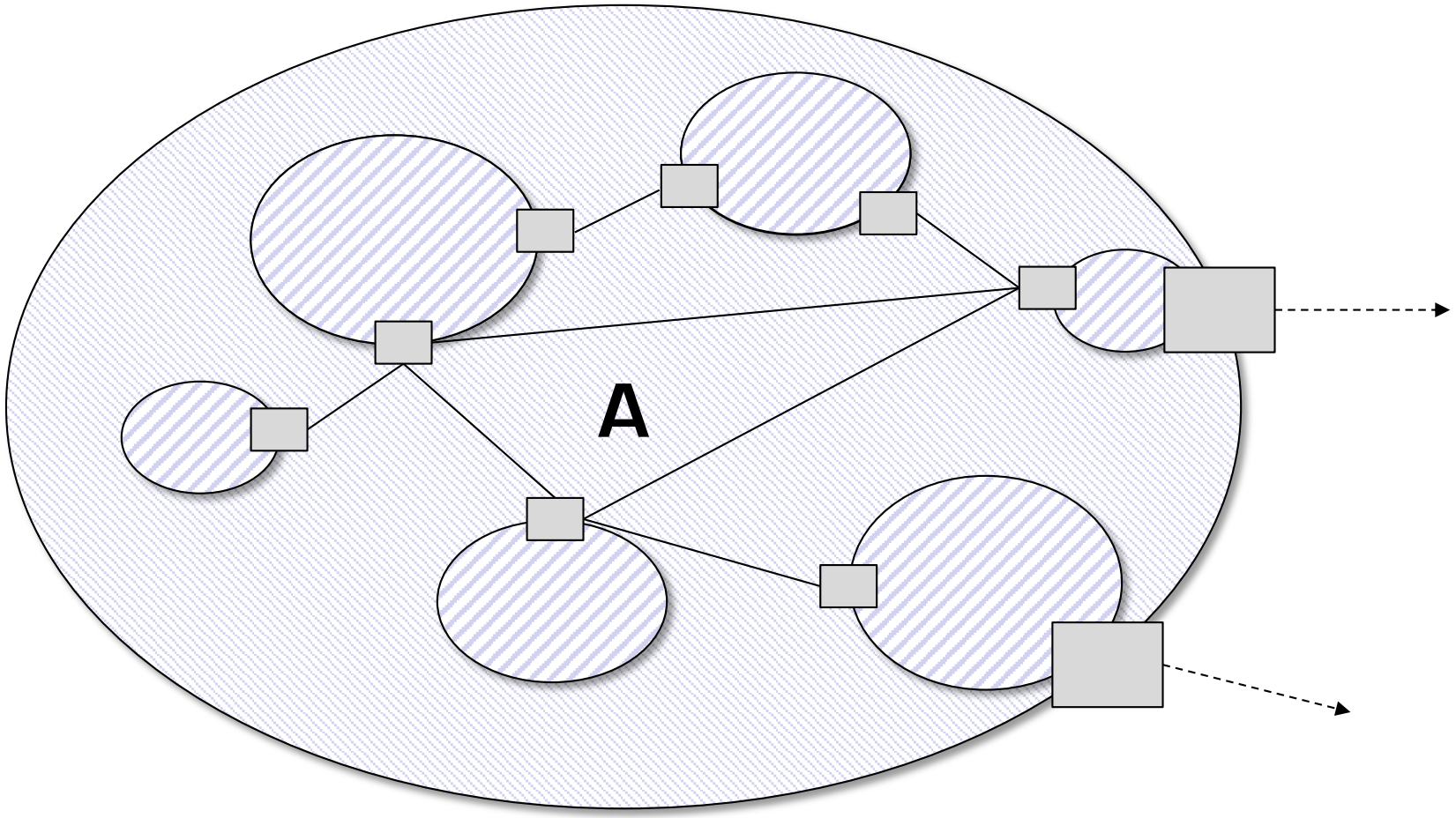
- Use “subnet mask” to distinguish network and subnet part from host part

- Routing with 3 levels of hierarchy
  - Algorithm in router (by masking bits: AND between address and subnet mask):
    - packet to another network (AND network != 0: then to main router)
    - packet to local end system (AND network and subnet == 0: yes, then deliver packet)
    - packet to other subnetwork (AND network == 0, subnet != 0: then reroute to appropriate router)

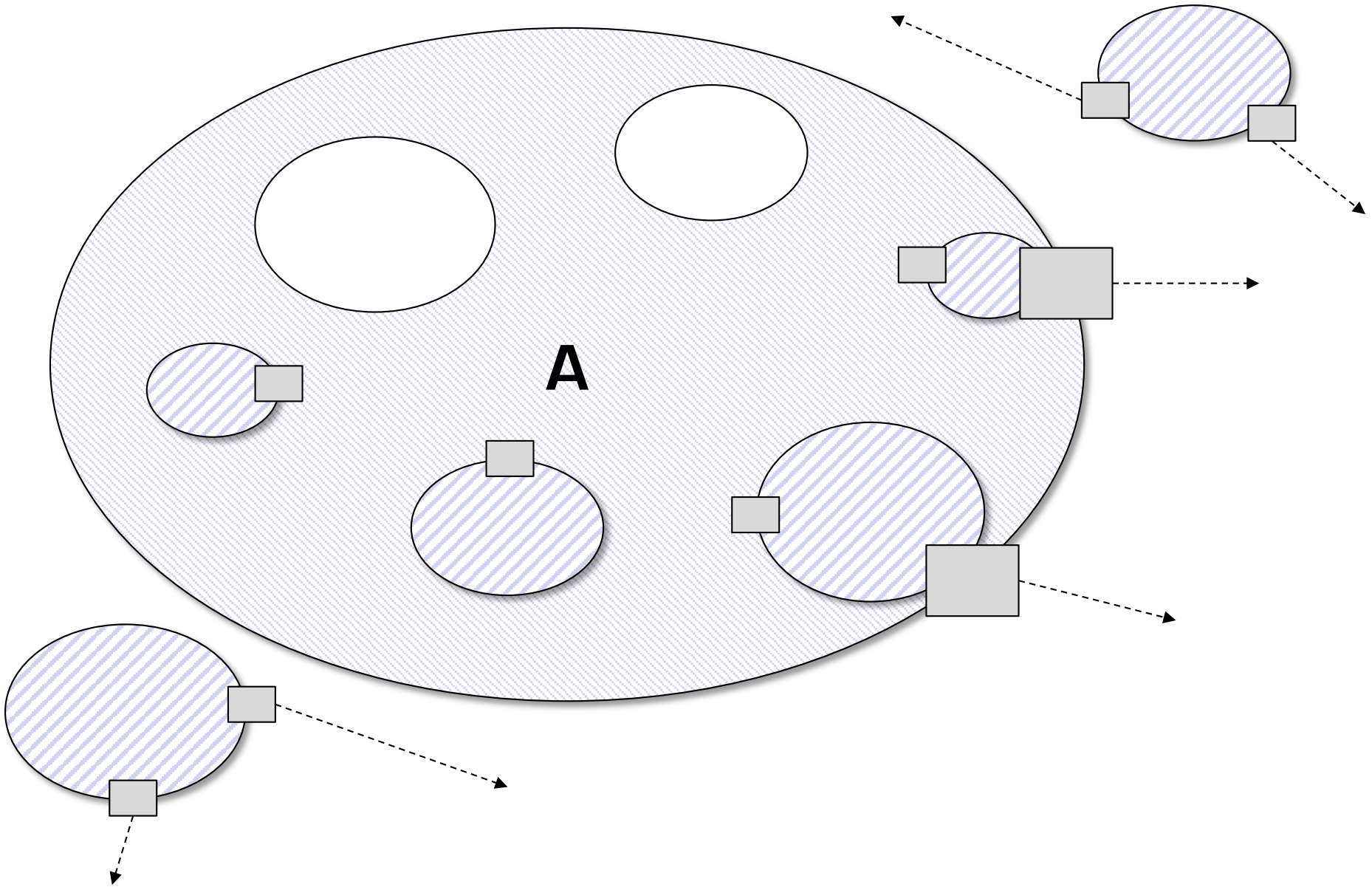
CIDR: Classless InterDomain Routing

- Subnetting not good enough
  - Too many organizations require addresses
    - in principle many addresses due to 32-bit address space
    - but inefficient allocation due to class-based organization
      - class A network with 16 million addresses too big for most cases
      - class C network with 256 addresses is too small
      - most organizations are interested in class B network, but there are only 16384 (in reality, class B too large for many organizations)

Internet Addresses and Internet Subnetworks



Internet Addresses and Internet Subnetworks



CIDR: Classless InterDomain Routing

- Subnetting not good enough
  - Too many organizations require addresses
    - in principle many addresses due to 32-bit address space
    - but inefficient allocation due to class-based organization
      - class A network with 16 million addresses too big for most cases
      - class C network with 256 addresses is too small
      - most organizations are interested in class B network, but there are only 16384 (in reality, class B too large for many organizations)

⇒ Introduction of CIDR (Classless InterDomain Routing) (RFC1519)

- CIDR Principle
  - to allocate IP addresses in variable-sized blocks
    - (without regard to classes)
  - e.g., a request for 2000 addresses would lead to
    - assignment of 2048 address block starting on 2048 byte boundary

- but, dropping classes makes forwarding more complicated
- Large number of networks leads to large routing tables

CIDR: Classless InterDomain Routing

- Search for longest matching prefix
  - if several entries with different subnet mask length may match
    - then use the one with the ***longest mask***
  - i.e., AND operation for address & mask must be done for each table entry

- Entries may be aggregated to reduce routing tables

