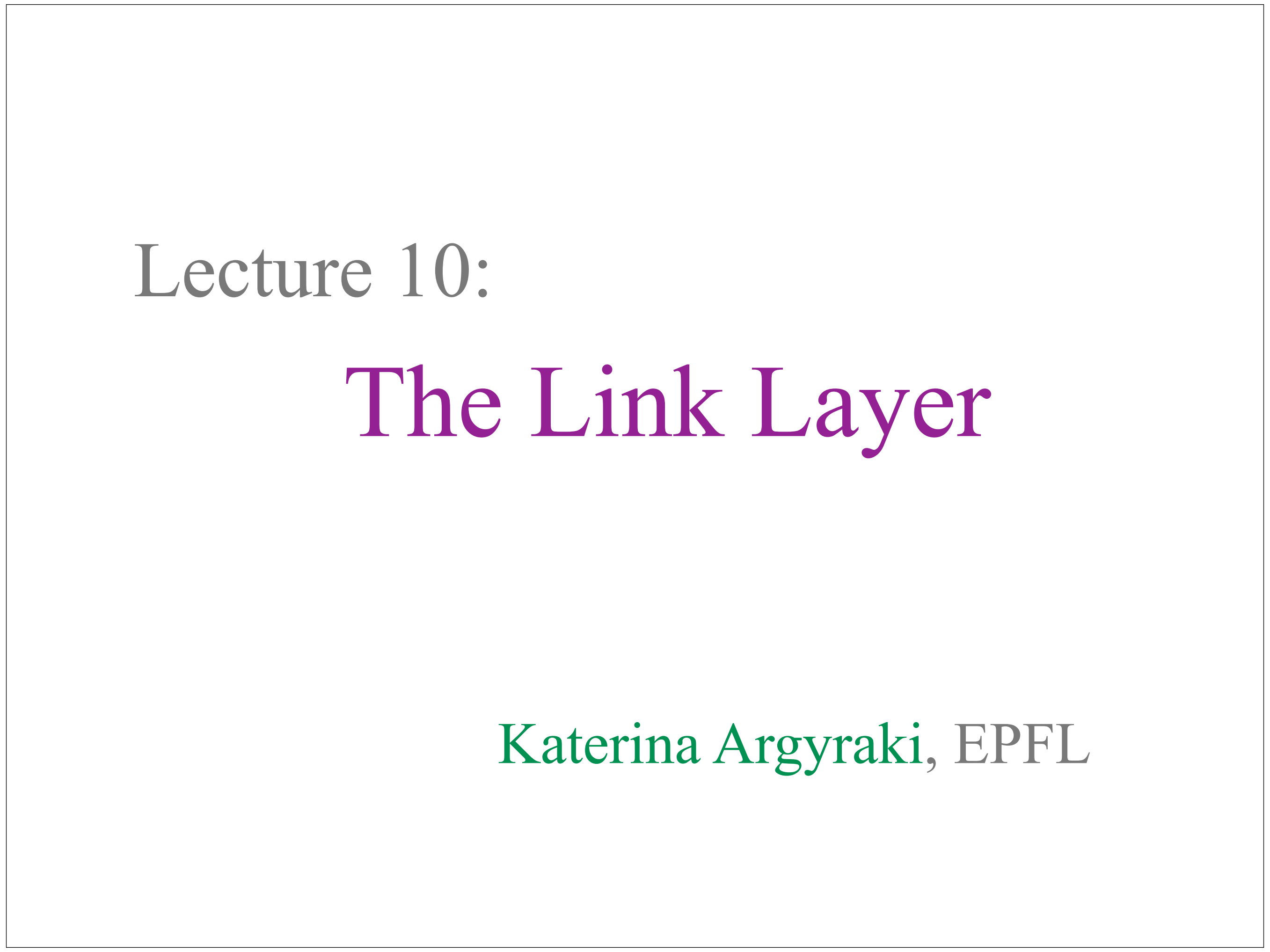
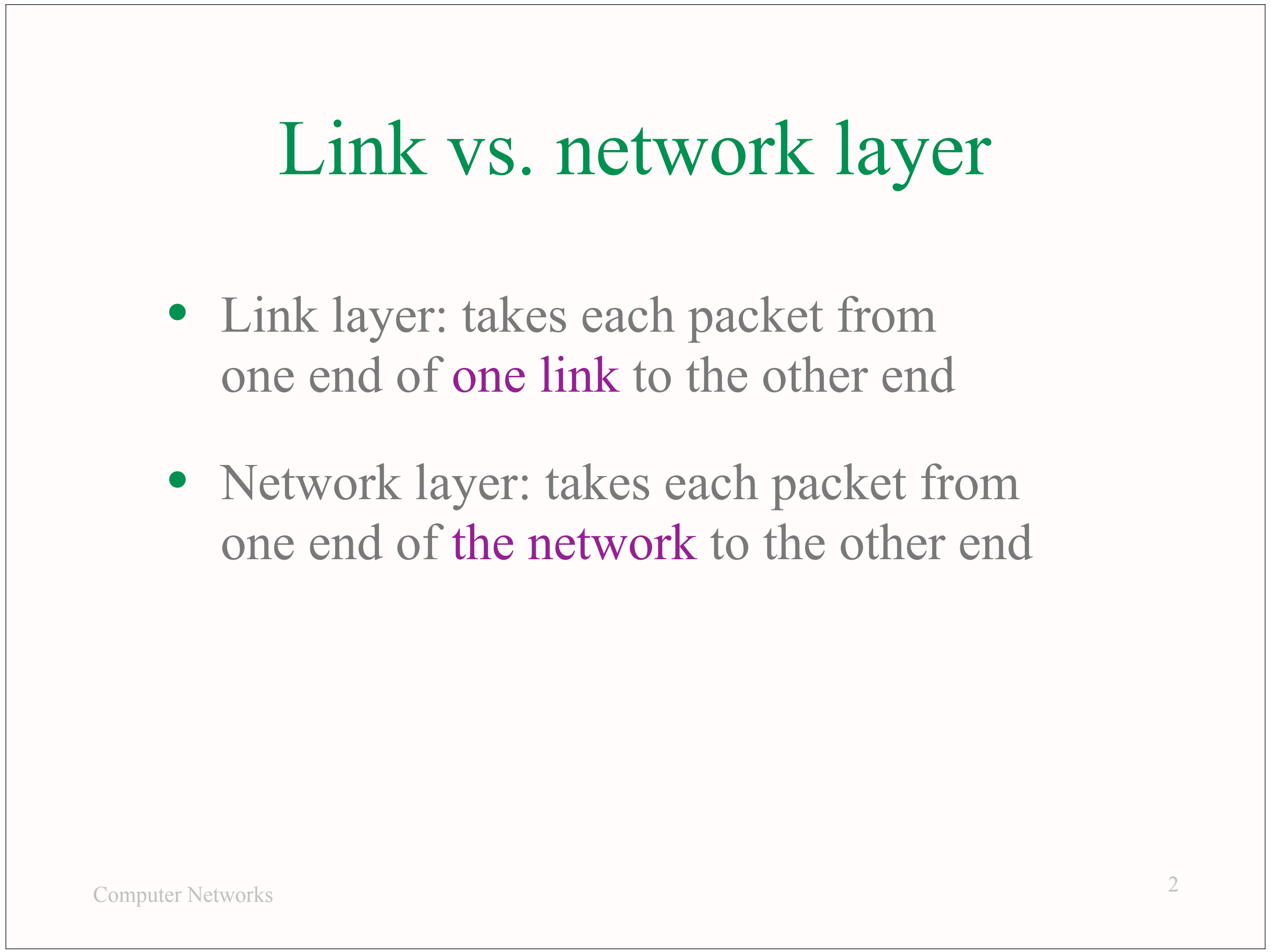


Lecture 10:

The Link Layer

Katerina Argyraki, EPFL

Welcome to the 10th lecture of Computer Networks, which is about the link layer.

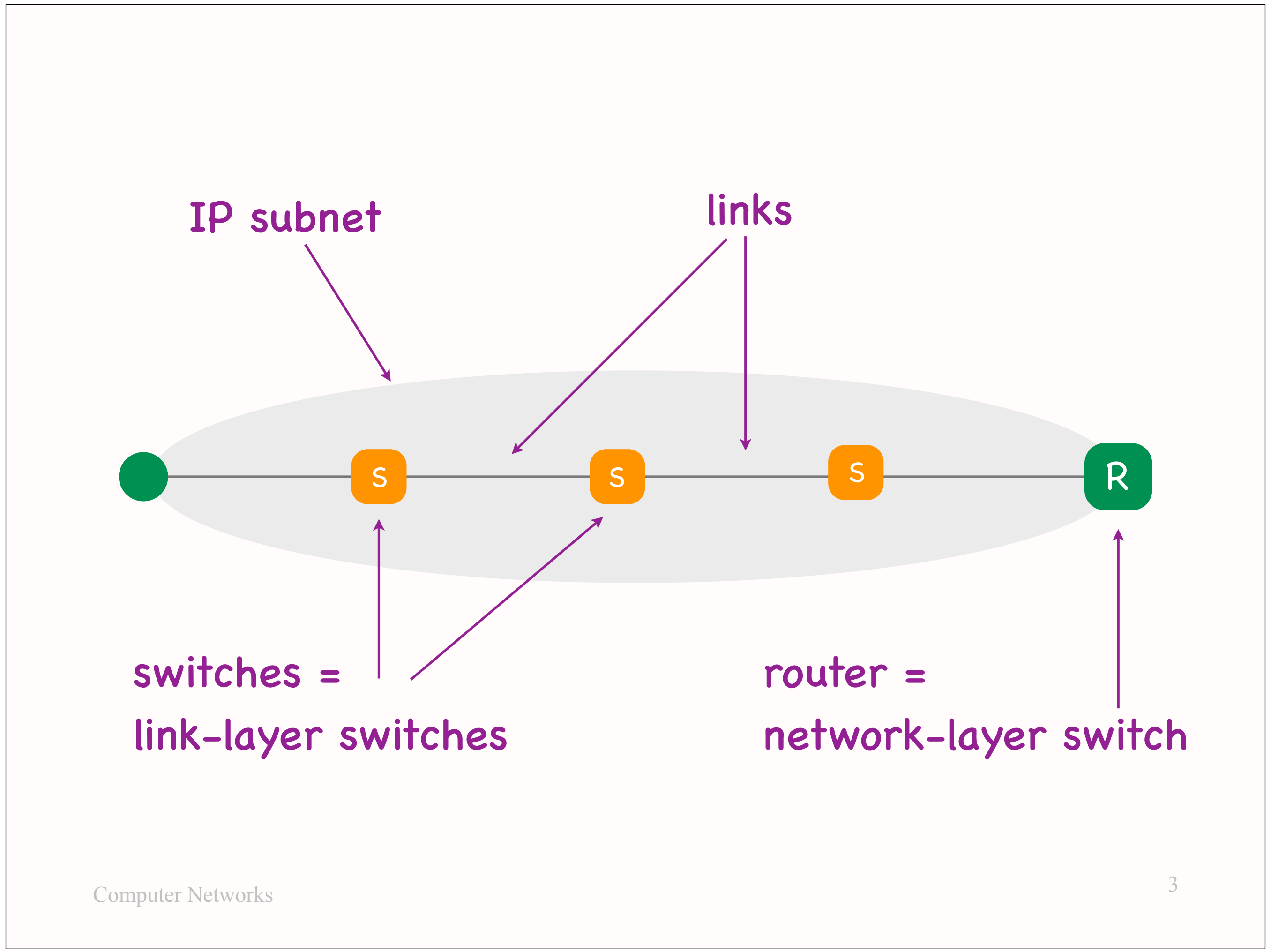


Link vs. network layer

- Link layer: takes each packet from one end of one link to the other end
- Network layer: takes each packet from one end of the network to the other end

We will start by comparing the link and network layers:
- The role of the link layer is to take one packet from one end of *one link* to the other end.
- The role of the network layer is to take a packet from one end of *the network* to the other end.

Let’s see what this means in more detail.



Consider a single IP subnet (the grey ellipsis).

An IP subnet is a network, which has end-systems and routers at its boundaries, which are interconnected by link-layer switches.

A note on terminology:

- When I say "switch" (without explaining what type of switch), I will mean link-layer switch. When I say "router," I will mean network-layer switch.

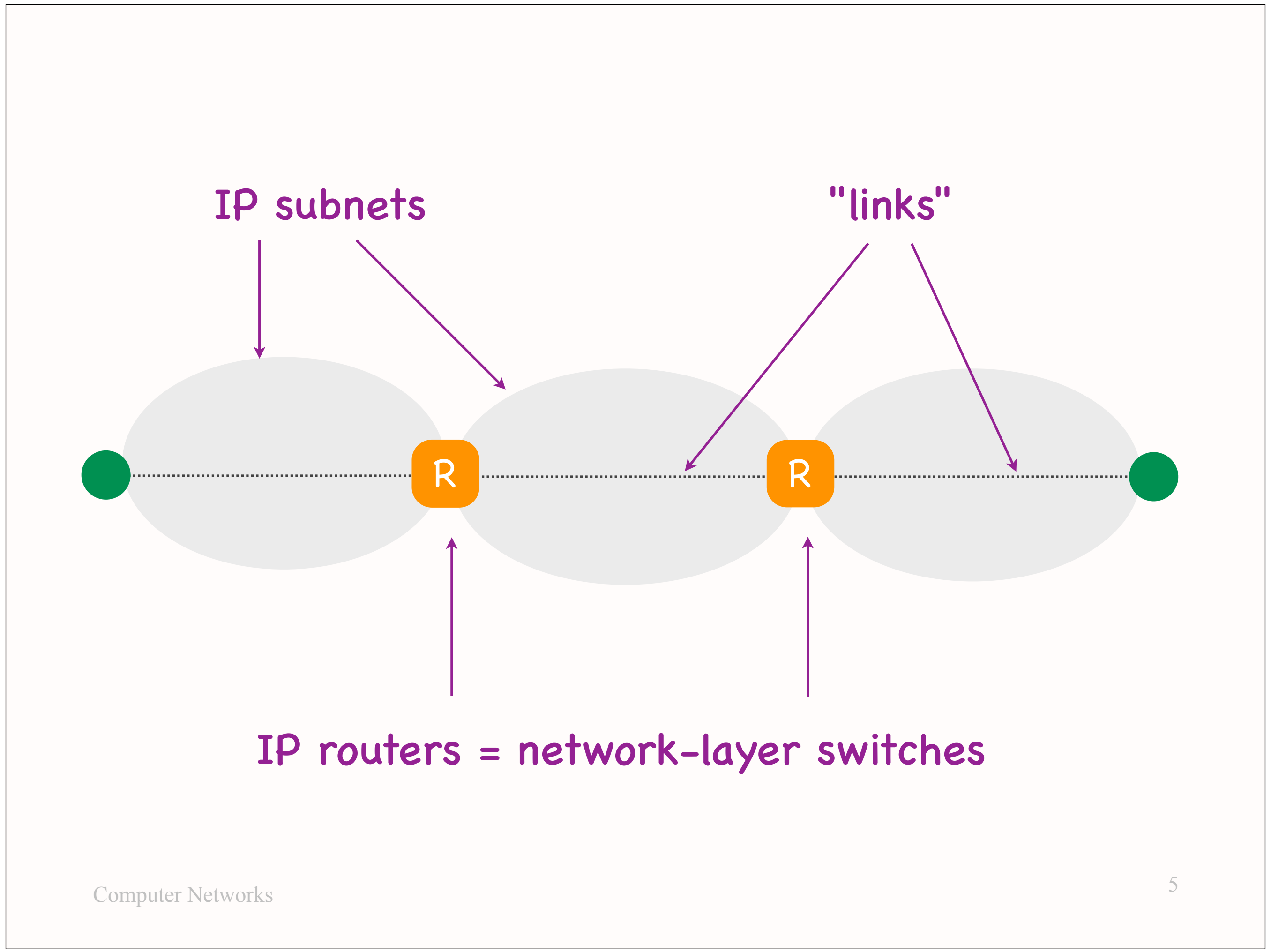
From the point of view of an IP subnet:

- The role of the link layer is to take a packet across a single physical link.
- The role of the network layer is to take a packet across the entire network, i.e., the entire IP subnet (e.g., from the green end-system on the left to router R).

IP subnet point of view

- Link layer: takes packet from
  one end of one physical link to the other end

- Network layer: takes packet from
  one end of the IP subnet to the other end



Consider the Internet.

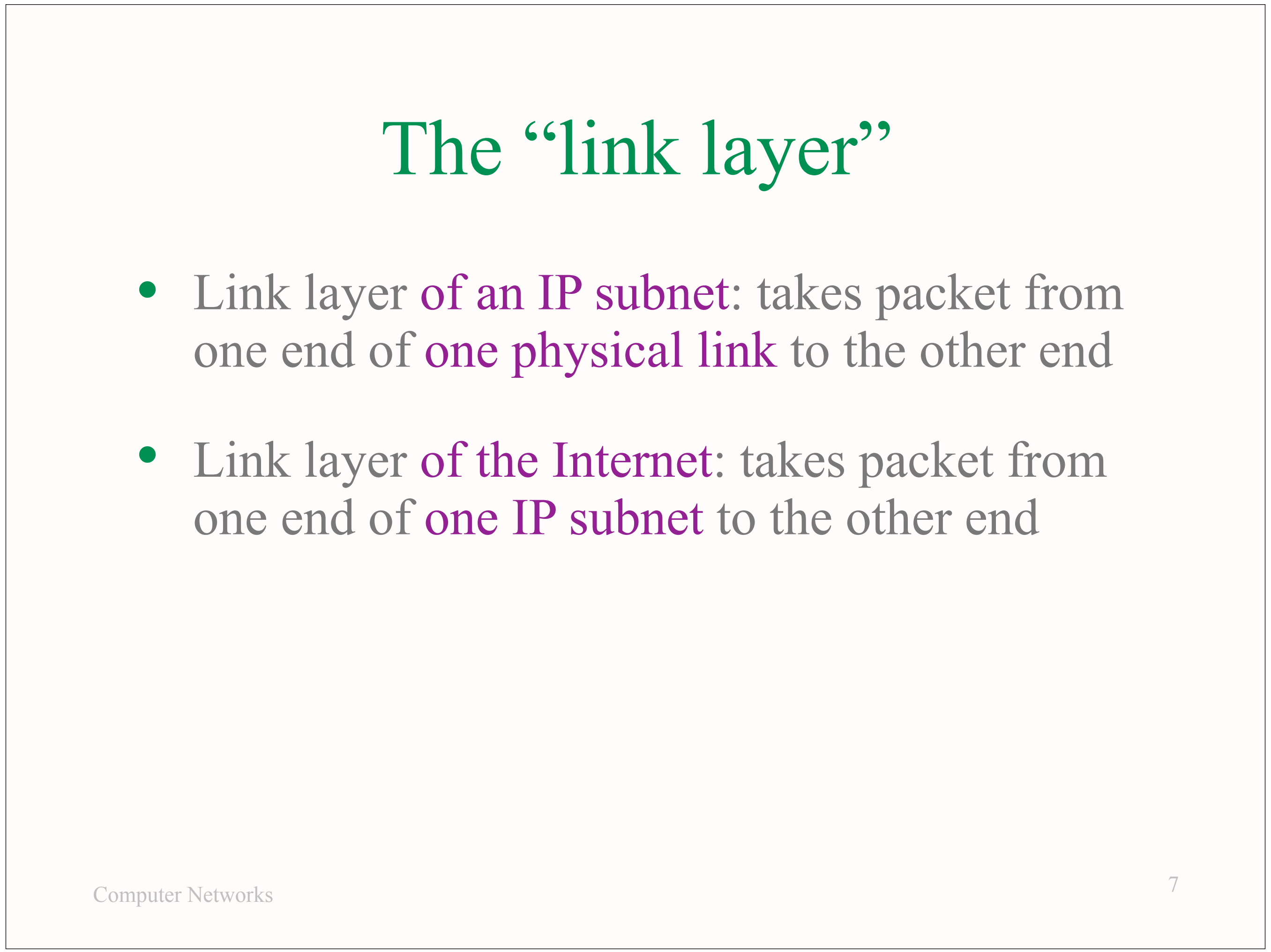
The Internet is a network of IP subnets, i.e., a network of networks (hence the name “Inter-network” or Internet).

From the point of view of the Internet, a “link” is a path segment across a single IP subnet.

So, from the point of view of the Internet:
- The role of the link layer is to take a packet across a single “link,” i.e., a single IP subnet.
- The role of the network layer is to take a packet across the entire network, i.e., the Internet.

Internet point of view

- Link layer: takes packet from
  one end of one IP subnet to the other end
- Network layer: takes packet from
  one end of the Internet to the other end



The “link layer”

- Link layer of an IP subnet: takes packet from one end of one physical link to the other end
- Link layer of the Internet: takes packet from one end of one IP subnet to the other end

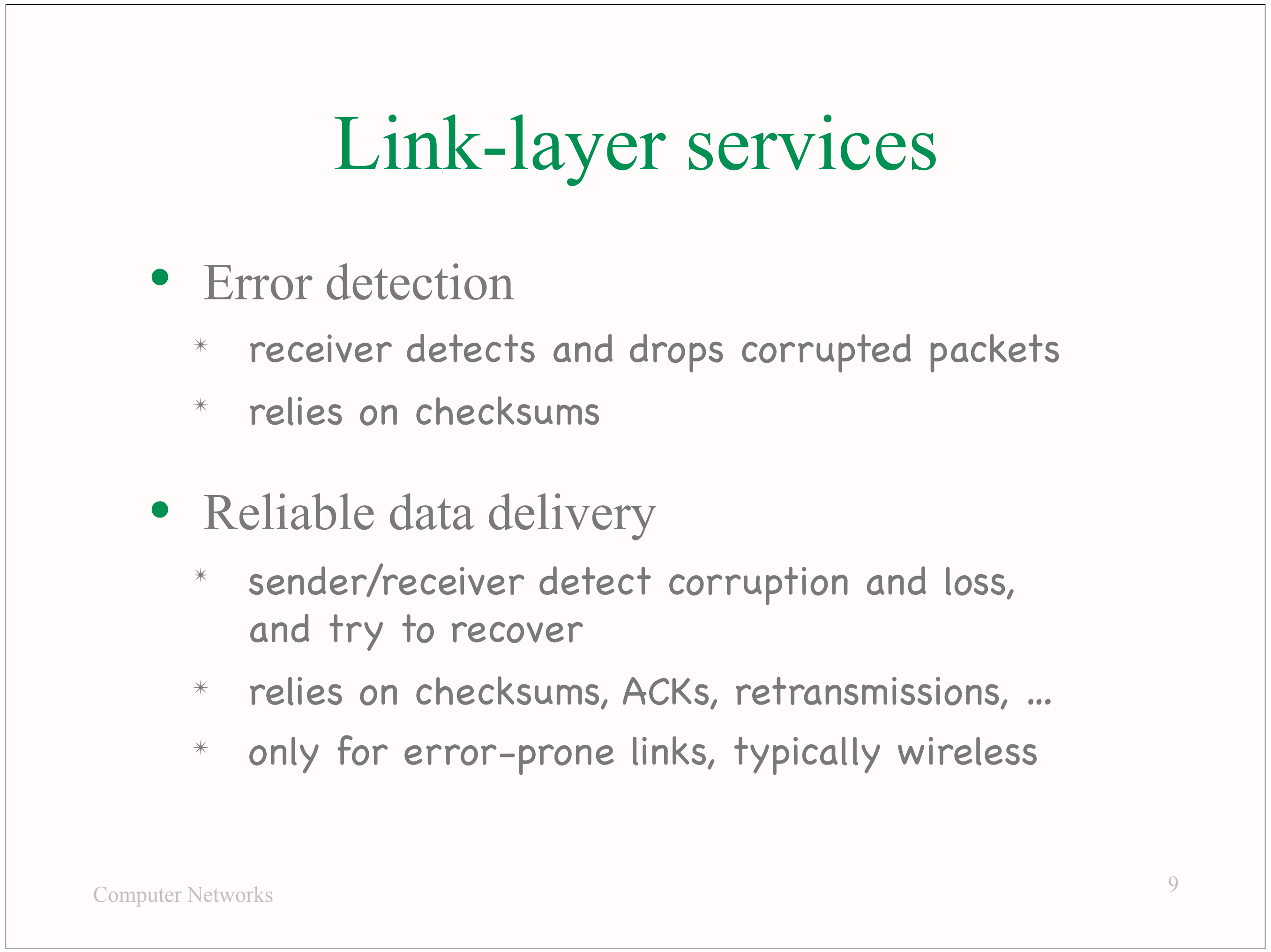
So, when we say “link layer,” we could mean one of two things: …

When people, in general, say “link layer,” they typically conflate the two, i.e., they refer to both of these things together.

The “link layer”

- Link layer of an IP subnet: takes packet from one end of one physical link to the other end
- Link layer of the Internet: takes packet from one end of one IP subnet to the other end

We will first discuss briefly the link layer of an IP subnet.


Link-layer services

- Error detection
  - receiver detects and drops corrupted packets
  - relies on checksums
- Reliable data delivery
  - sender/receiver detect corruption and loss, and try to recover
  - relies on checksums, ACKs, retransmissions, …
  - only for error-prone links, typically wireless

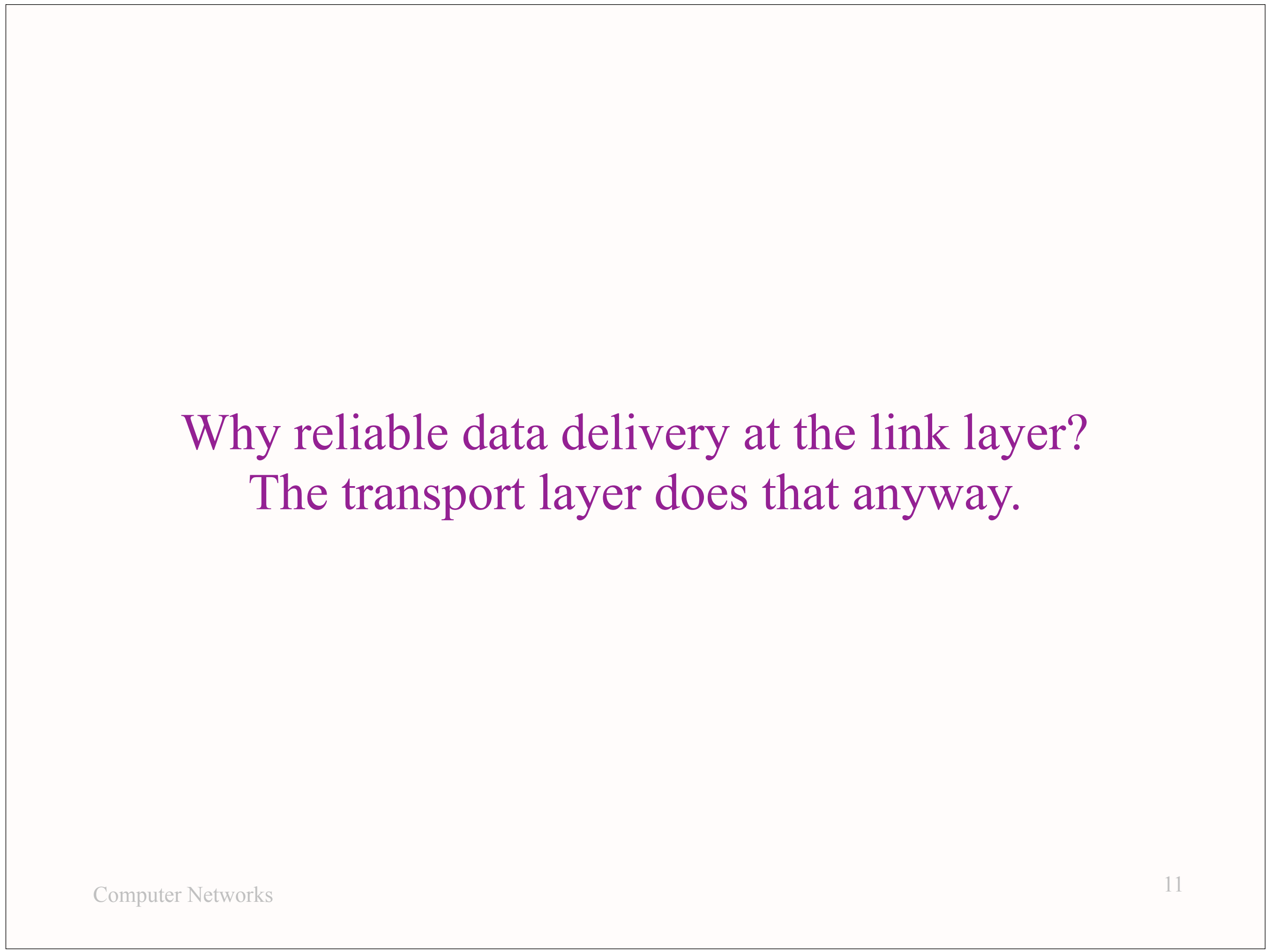
It offers three main services: …

Btw, when I say "receiver" on this slide, I mean a device that sits at the end of a physical link, which could be an end-system, a switch, or a router.

Similarly, when I say "sender," I mean a device that sits at the beginning of a physical link.

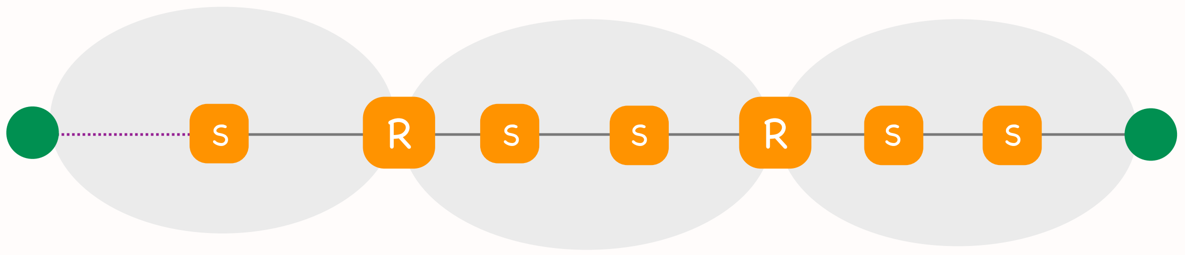
Link-layer services

- Medium access control (MAC)
  - sender manages access to shared medium (typically wireless link)
  - listens for ongoing transmissions or "collisions"
  - backs off and retries later


Why reliable data delivery at the link layer? The transport layer does that anyway.

Why provide reliable data delivery at the link layer when TCP can provide it at the transport layer?

Differently said: If the network between Alice and Bob loses or corrupts packets, Alice and Bob can typically recover using TCP. Why provide recovery from loss and corruption also at the link layer?



Because it is a useful performance optimization:

Consider the topology shown above, where two end-systems communicate over a long sequence of physical links.

Suppose the first link is a wireless link that suffers from occasional “glitches,” i.e., short time intervals of heavy packet loss.

Suppose the source end-system sends a sequence of segments to the destination end-system during such a glitch, and they all get lost on the wireless link.

Think about how TCP recovers from such loss:
- The source end-system times out waiting for ACKs.
- It resets its congestion window to 1 MSS and retransmits the oldest unacknowledged segment.
- When it receives an ACK for this segment, it increases the congestion window to 2 MSS’s, retransmits the next two segments, and so on.

Two important points:
- The source end-system must wait for a TCP timeout before acting.
- The throughput from the source to the destination end-system drops significantly.

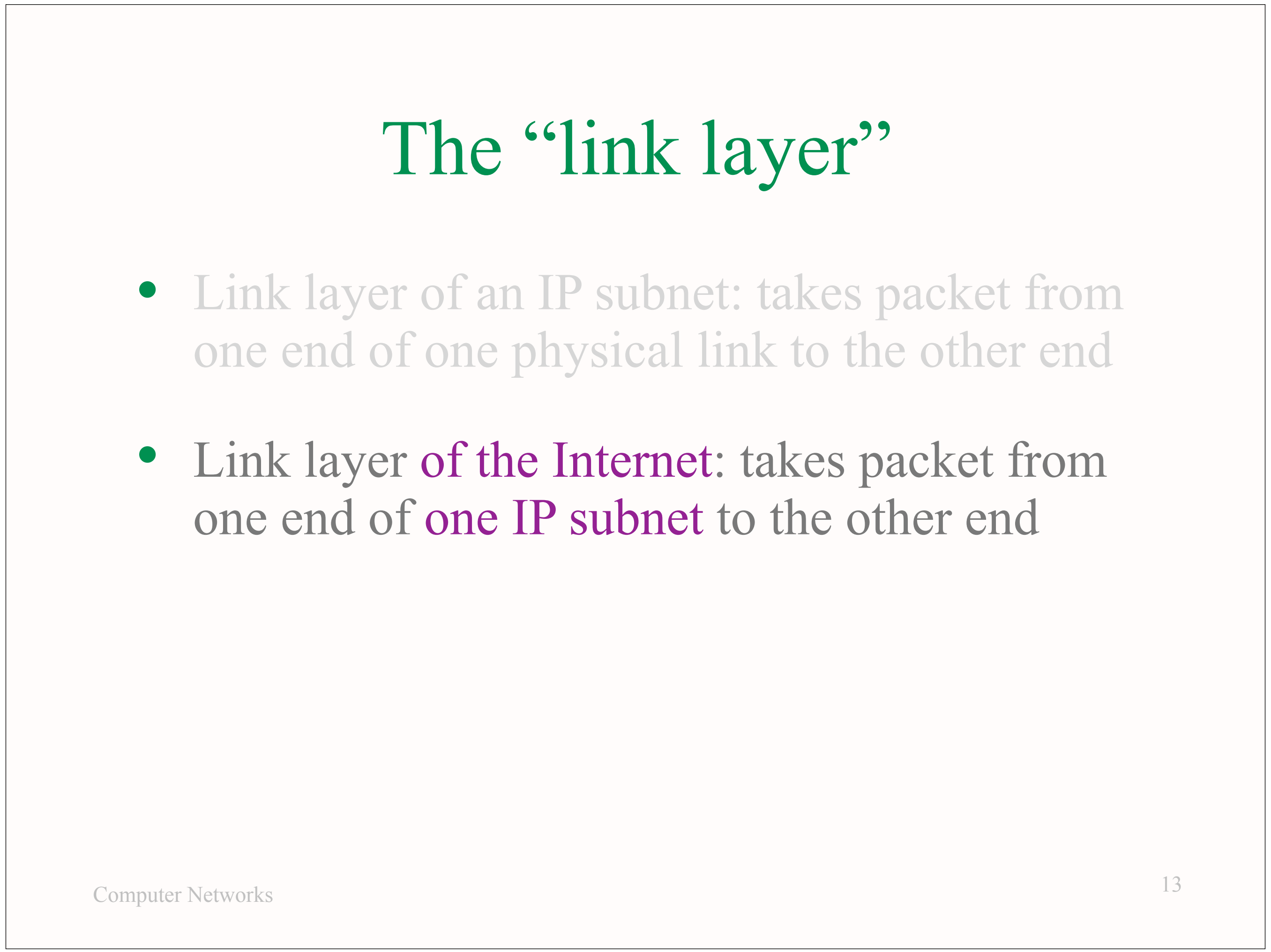
Now suppose the wireless link provides reliable data delivery (at the link layer). This means that the two devices at the ends of the wireless link (the source end-system and the first switch) try to **locally** recover from any packet loss that occurs on the wireless link.

Think about what happens:
- The link-layer part of the source end-system times out waiting for ACKs from the (link-layer part of the) first switch and retransmits the lost packets

(using some congestion-control algorithm that we have not discussed).

- The transport-layer part of the source end-system (the TCP code running at the source end-system) does not perceive any packet loss, so it does not timeout and does not reset its congestion window.

So: The link layer around the specific problematic link recovers quickly from the loss, hence TCP does not perceive the loss. This means no TCP timeout and no TCP congestion-window reset, hence higher throughput from the source to the destination end-system.



The “link layer”

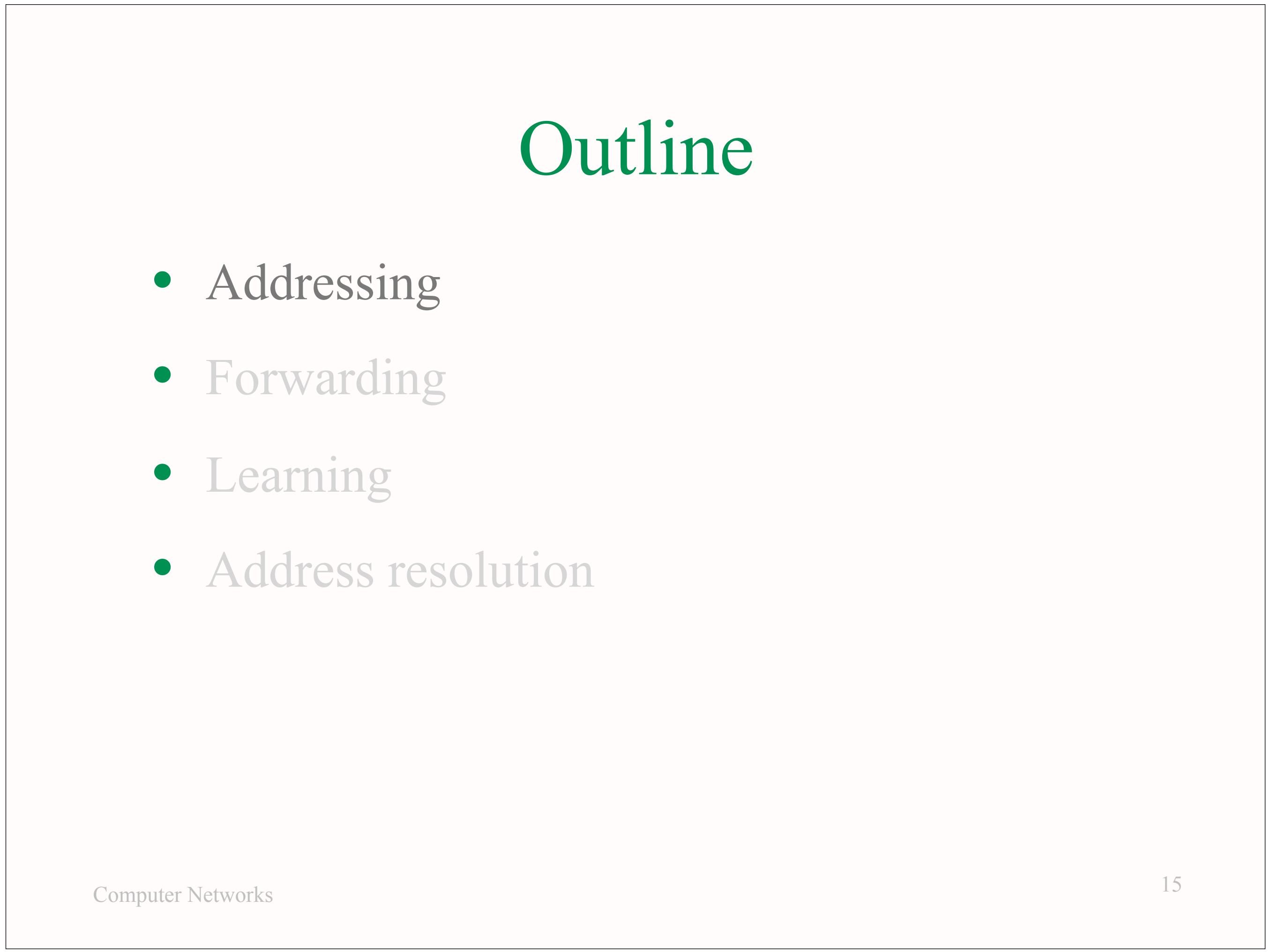
- Link layer of an IP subnet: takes packet from one end of one physical link to the other end
- Link layer of the Internet: takes packet from one end of one IP subnet to the other end

The rest of the lecture is dedicated to the link layer of the Internet,
i.e., how to get packets from one end of an IP subnet to the other.

There exist many different types of IP subnets, but we will discuss only the most popular one: Ethernet.

Outline

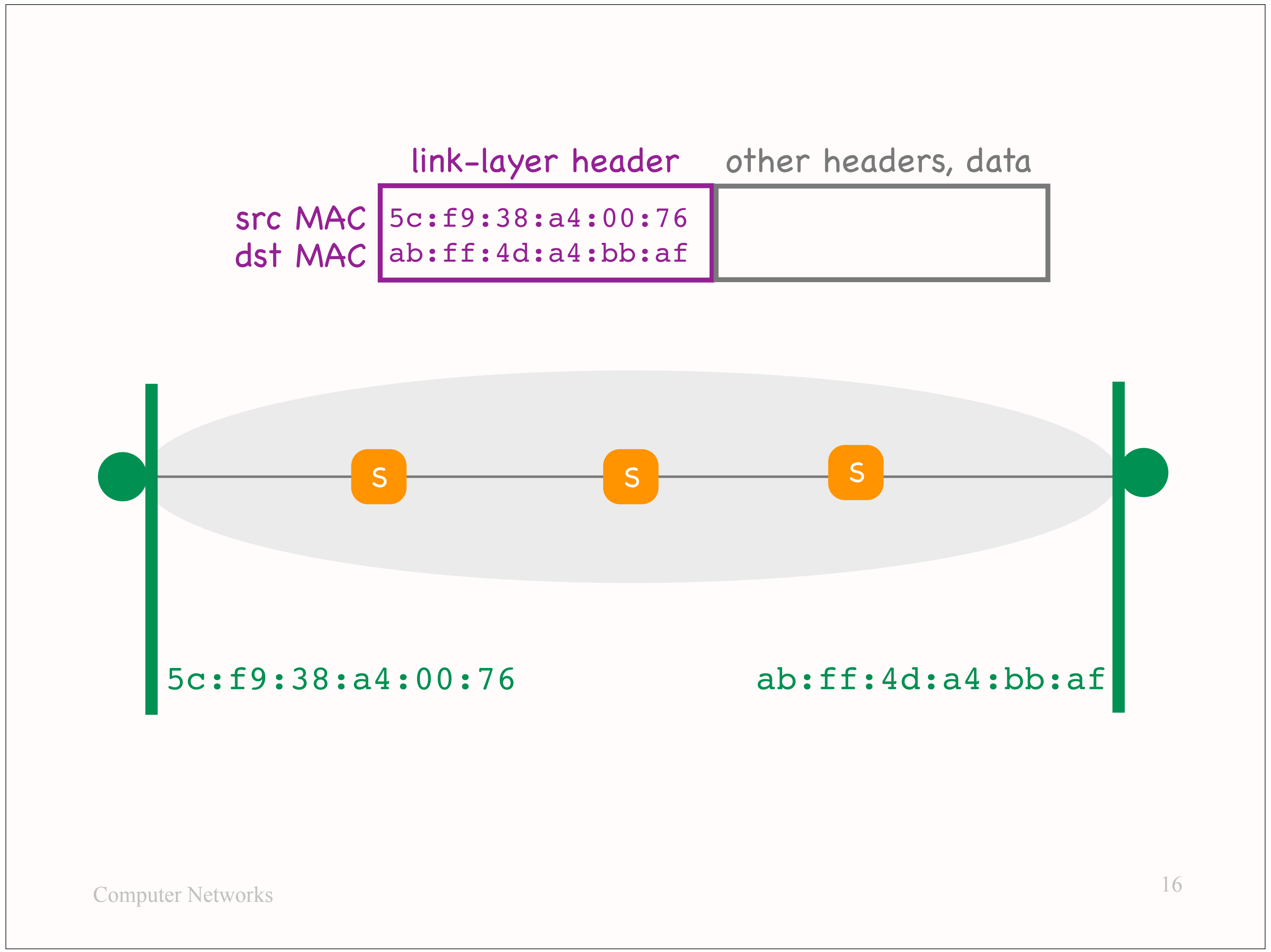
- Addressing
- Forwarding
- Learning
- Address resolution



Outline

- Addressing
- Forwarding
- Learning
- Address resolution

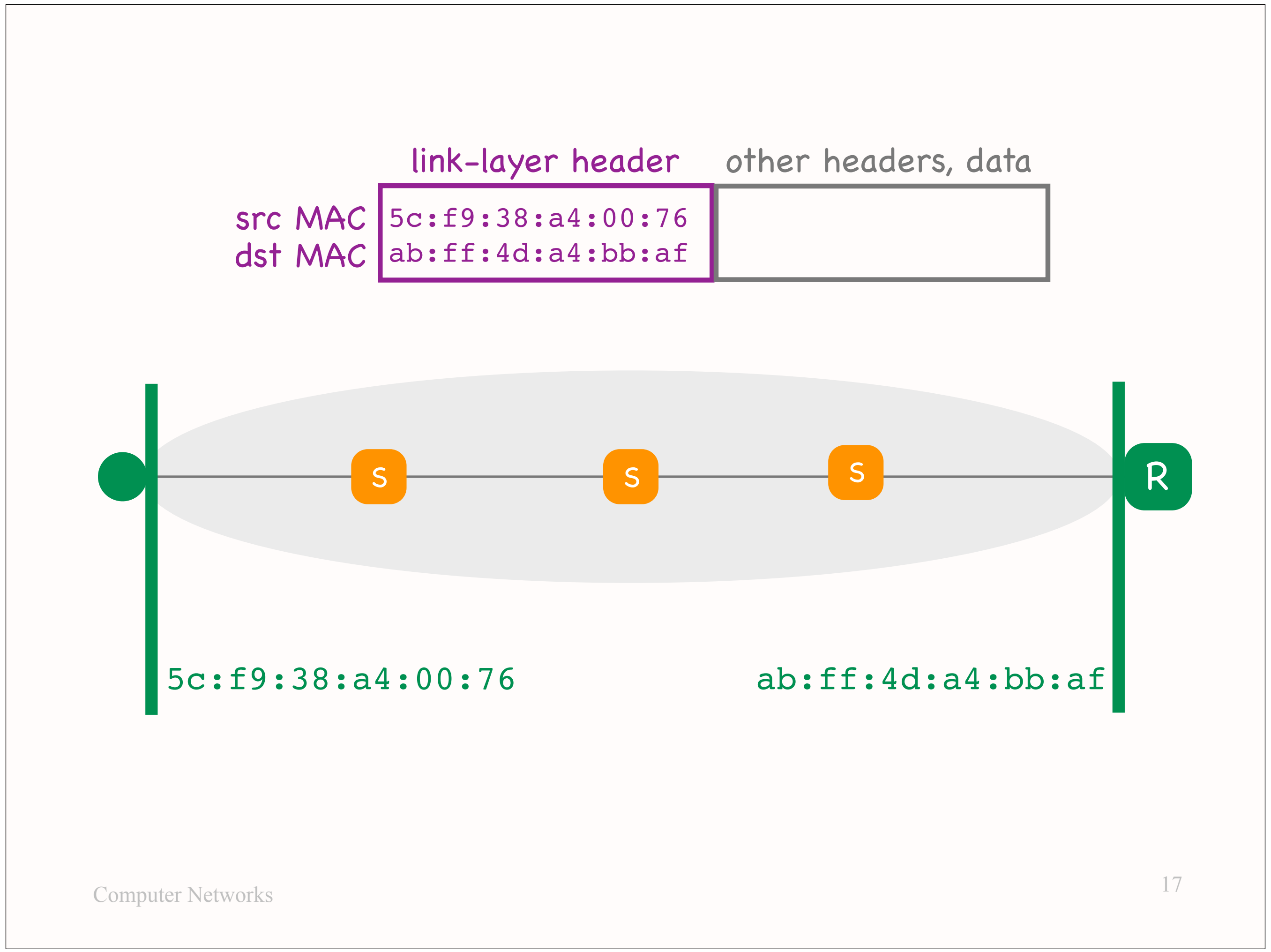
What type of addressing does Ethernet rely on?



Consider a single IP subnet of type Ethernet.

Every end-system in this IP subnet has at least one network interface,
and every network interface has a link-layer address (also called MAC address, or Ethernet address).

So, when one end-system sends a packet to another end-system, in the same IP subnet, the packet carries a link-layer header (also called MAC header, or Ethernet header), and inside this header there is a source MAC address and a destination MAC address (plus a few other fields).

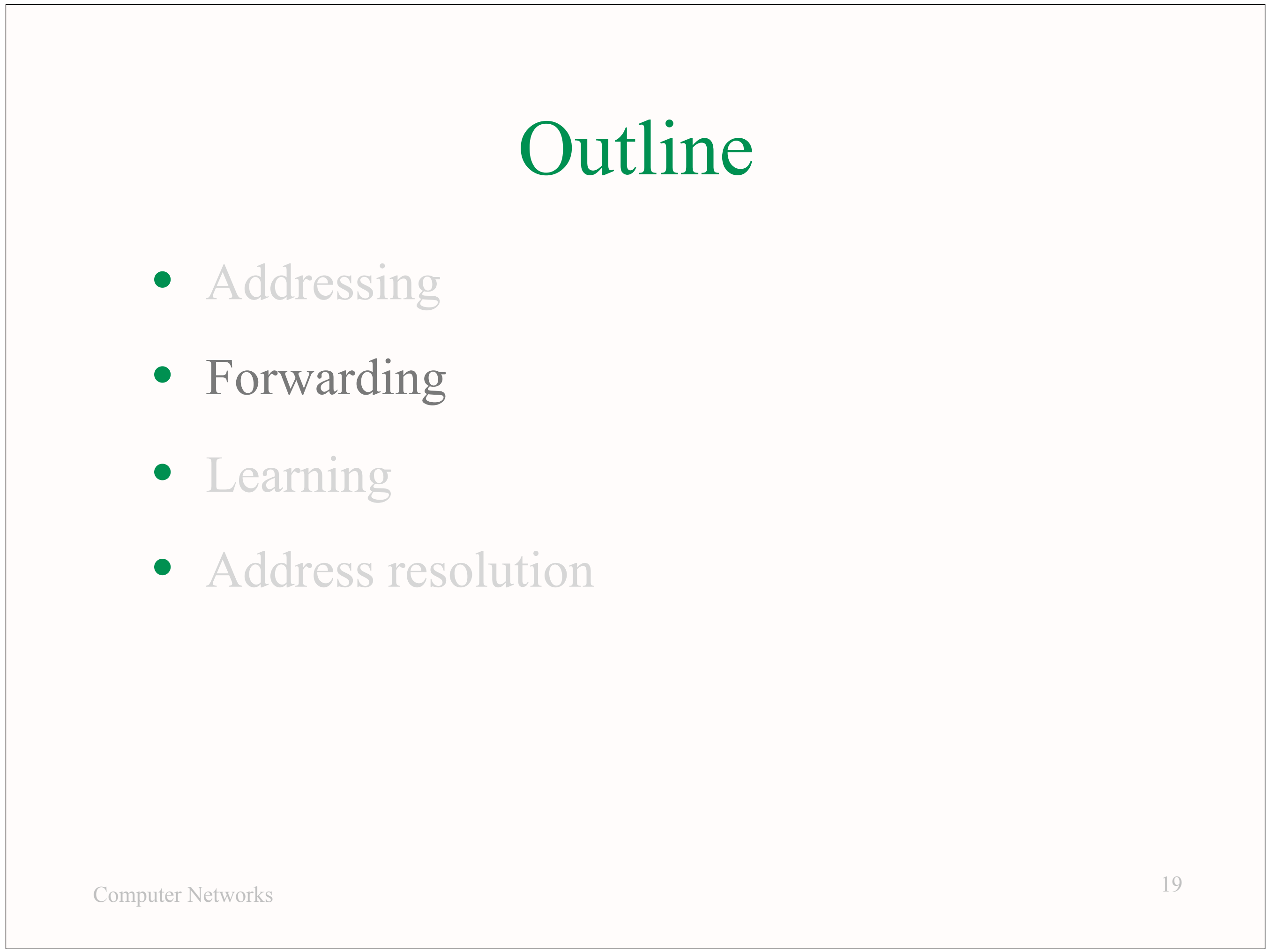


What if an end-system sends a packet to another end-system that is located in a different IP subnet?
In that case, the packet will necessarily cross a router located at the border of the source end-system's IP subnet (to exit the subnet).
While the packet is traveling inside the source end-system's IP subnet, its source MAC address is the MAC address of the source end-system, while the destination MAC address is the MAC address of this router.

In general, a packet traveling inside an IP subnet always carries source and destination MAC addresses from the current subnet. The source MAC address is the MAC address of the first device (end-system or router) to forward the packet in this subnet, while the destination MAC address is the last device to receive the packet in this subnet.

MAC address

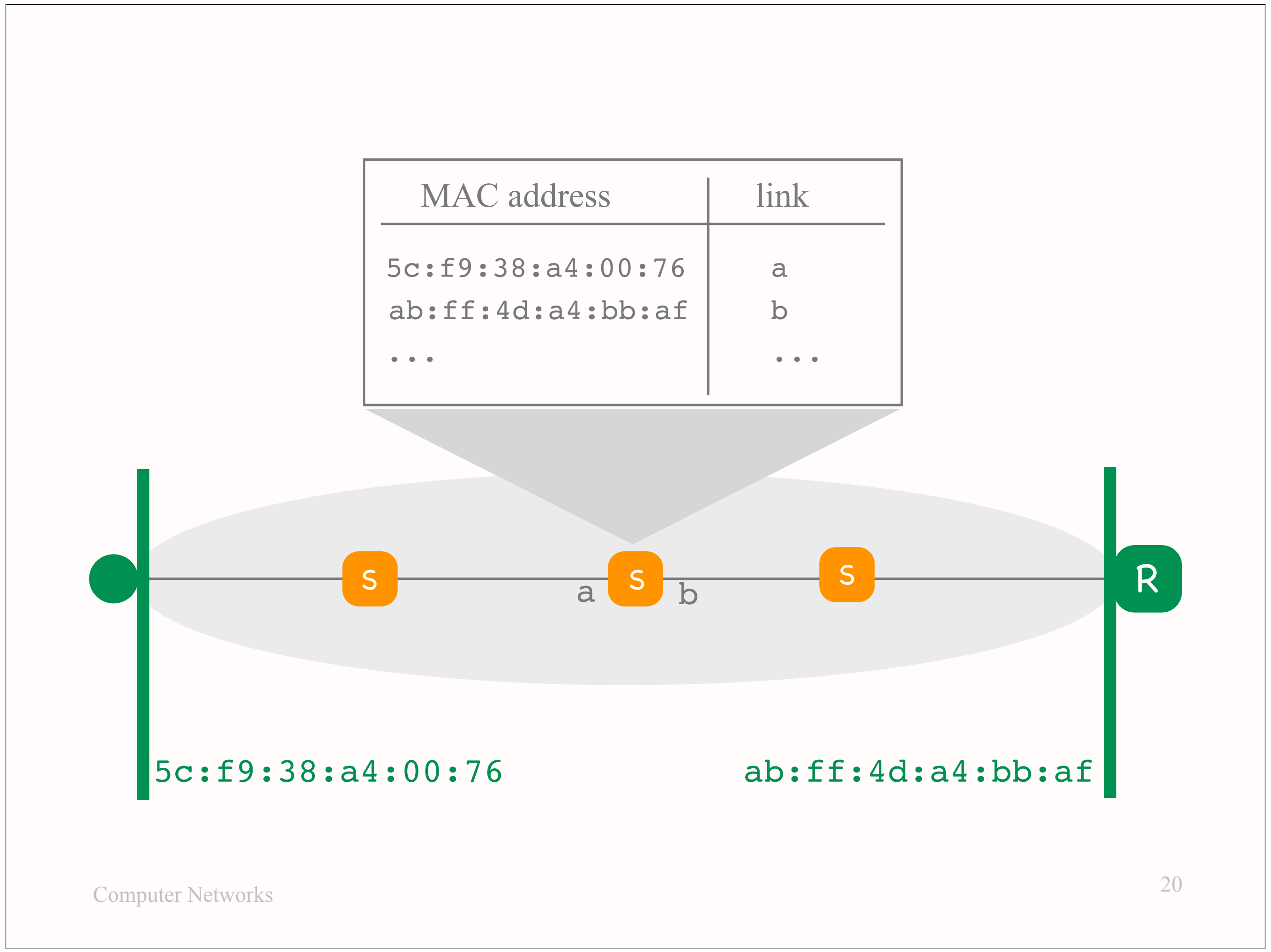
- 48-bit number
  - typical format: 1A-2B-DD-78-CF-CC
  - the value of each byte as hexadecimal
- Flat
  - not hierarchical like IP address
  - not location dependent



Outline

- Addressing
- Forwarding
- Learning
- Address resolution

How do Ethernet switches forward packets?



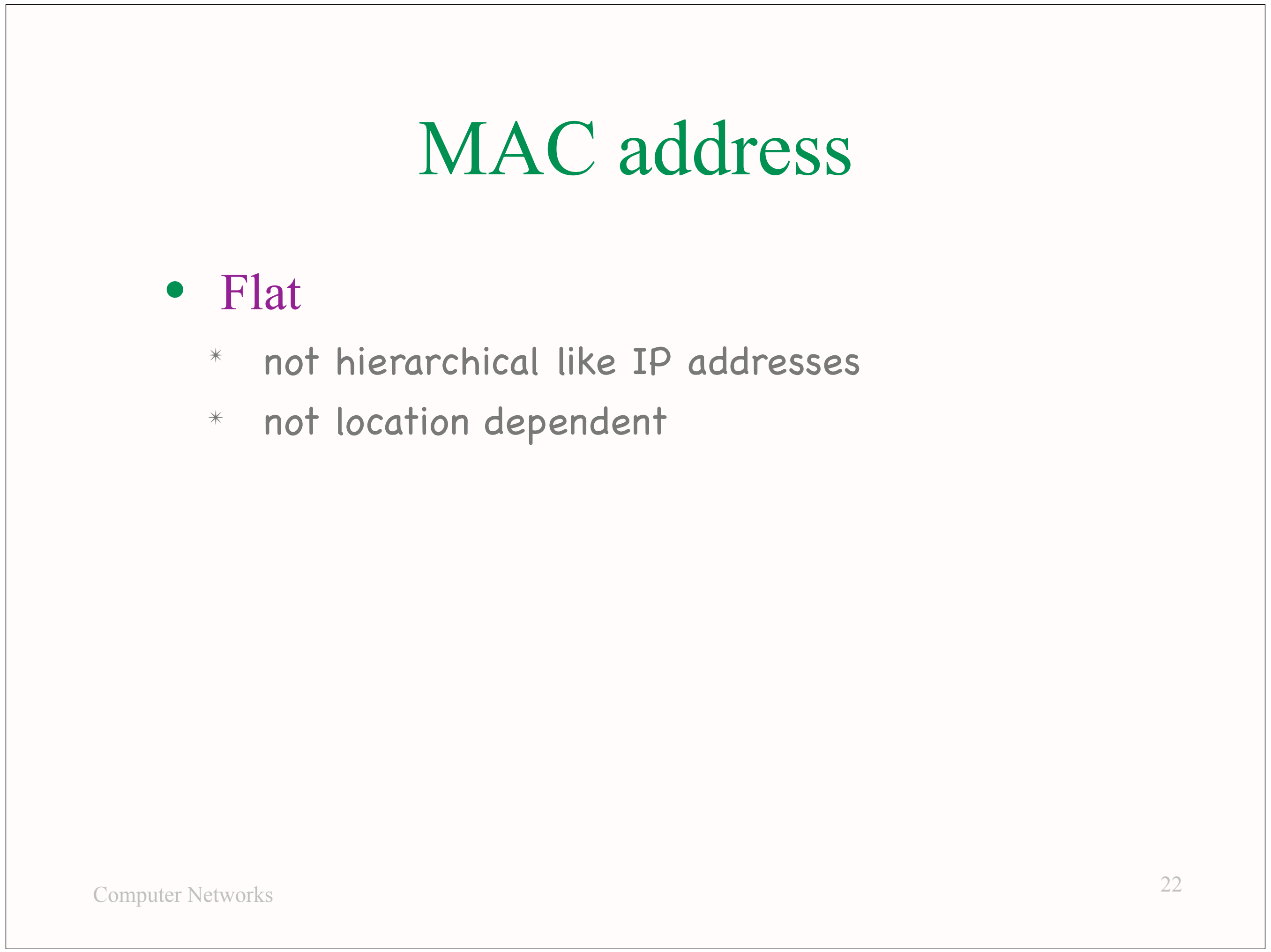
MAC address	link
5c:f9:38:a4:00:76	a
ab:ff:4d:a4:bb:af	b
...	...

Each switch names its network interfaces (also called links, or ports),
and keeps a forwarding table that maps MAC addresses to links.

When a packet arrives, the switch reads the destination MAC address from the MAC header, looks it up in the forwarding table,
and identifies the correct output link for this packet.

L2 forwarding

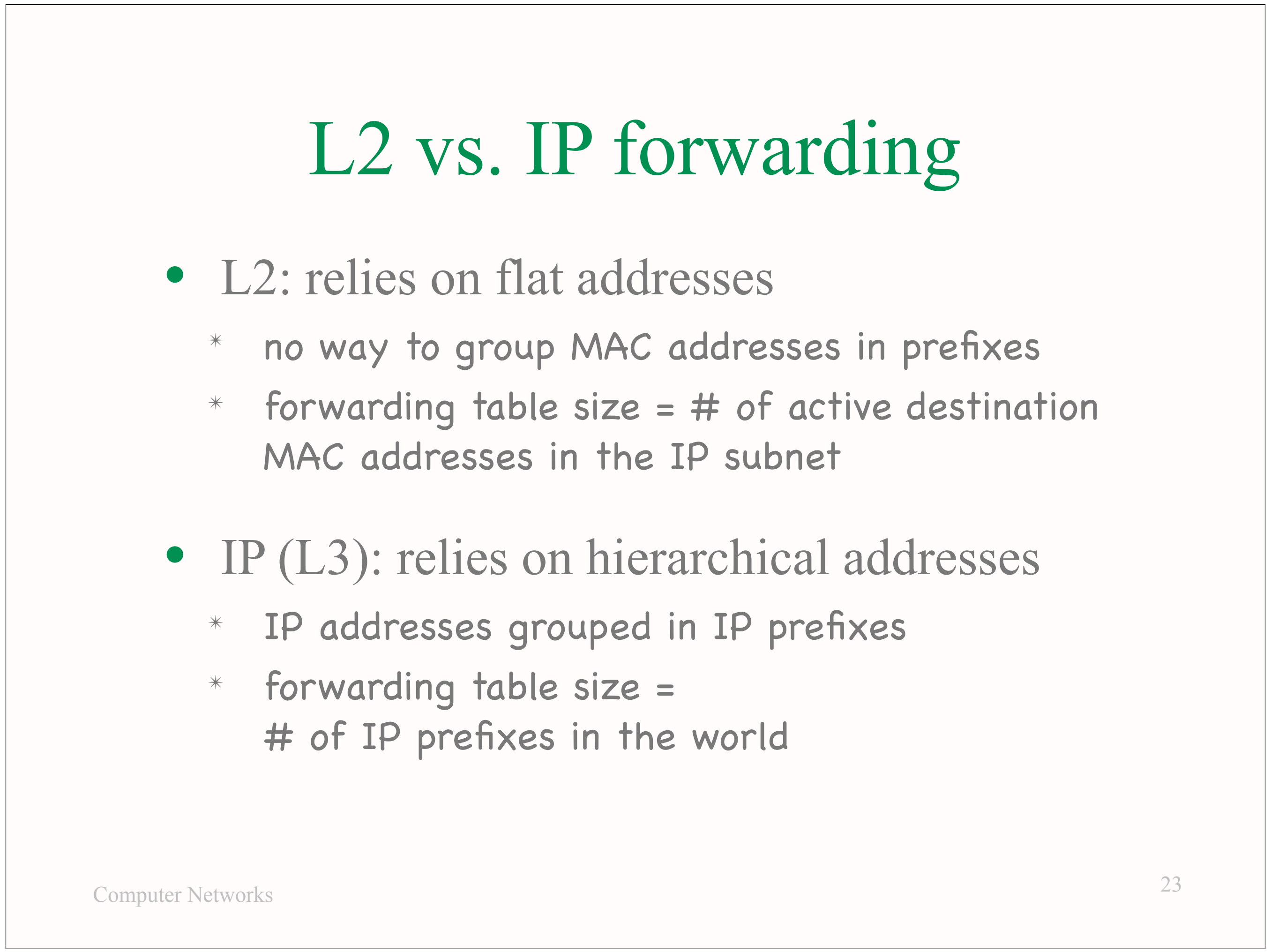
- Local switch process that determines output link for each packet
- Relies on forwarding table
  - maps destination MAC addresses to output links
- Similar to IP (L3) forwarding, except...



MAC address

- Flat
  - not hierarchical like IP addresses
  - not location dependent

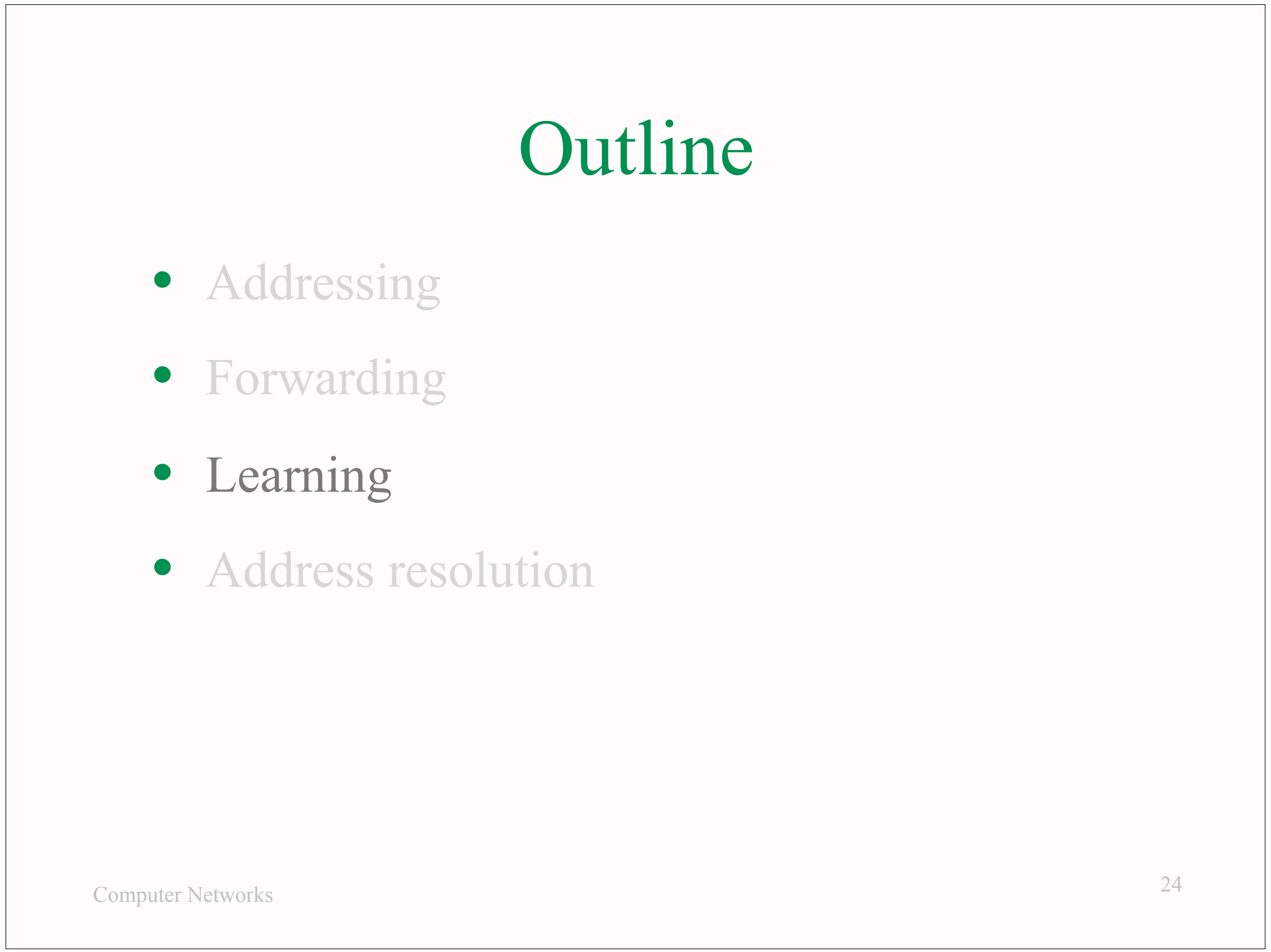
There is an important difference between IP addresses and MAC addresses:
MAC addresses are flat, not hierarchical.


L2 vs. IP forwarding

- L2: relies on flat addresses
  - no way to group MAC addresses in prefixes
  - forwarding table size = # of active destination MAC addresses in the IP subnet
- IP (L3): relies on hierarchical addresses
  - IP addresses grouped in IP prefixes
  - forwarding table size = # of IP prefixes in the world

So, if we compare L2 to IP (L3) forwarding, the former relies on flat addresses, whereas the latter relies on hierarchical addresses.

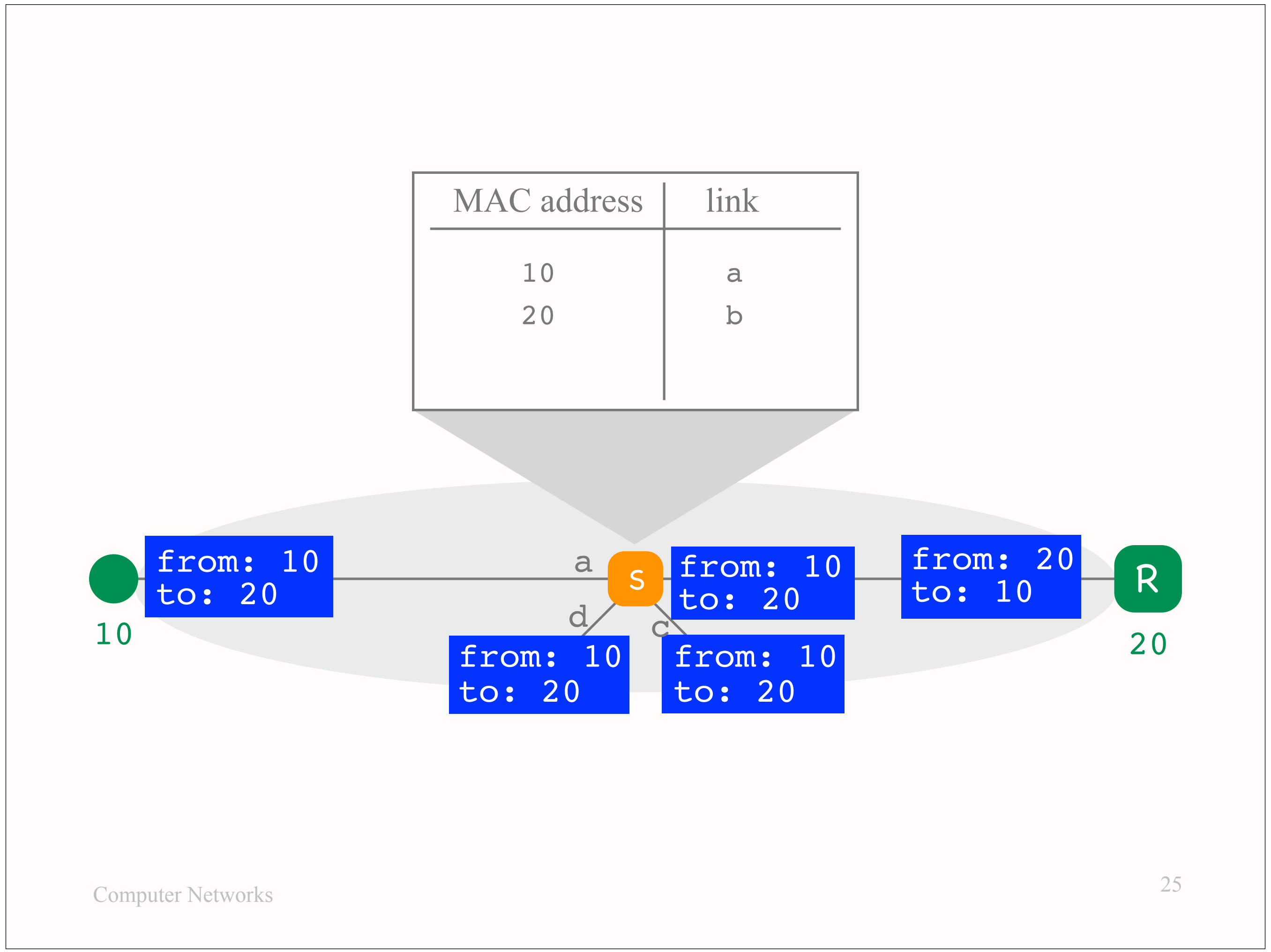
The fact that MAC addresses are flat means that we cannot group MAC addresses, e.g., by prefix, as we do with IP addresses.
As a result, the forwarding table of a switch keeps an entry for every individual MAC address that is currently active in the local IP subnet.



Outline

- Addressing
- Forwarding
- Learning
- Address resolution

Who populates the forwarding tables of Ethernet switches?



MAC address	link
10	a
20	b

The forwarding table of a link-layer switch is initially empty.
The switch fills the table based on the traffic it receives.

For example, if it receives a packet with source MAC address 10 at link (a),
it adds an entry to the forwarding table, indicating that:
if, in the future, it receives a packet with destination MAC address 10, it should forward that packet to link (a).

If the switch receives a packet with a destination MAC address for which no entry currently exists in its forwarding table,
it broadcasts that packet to all(*) links.

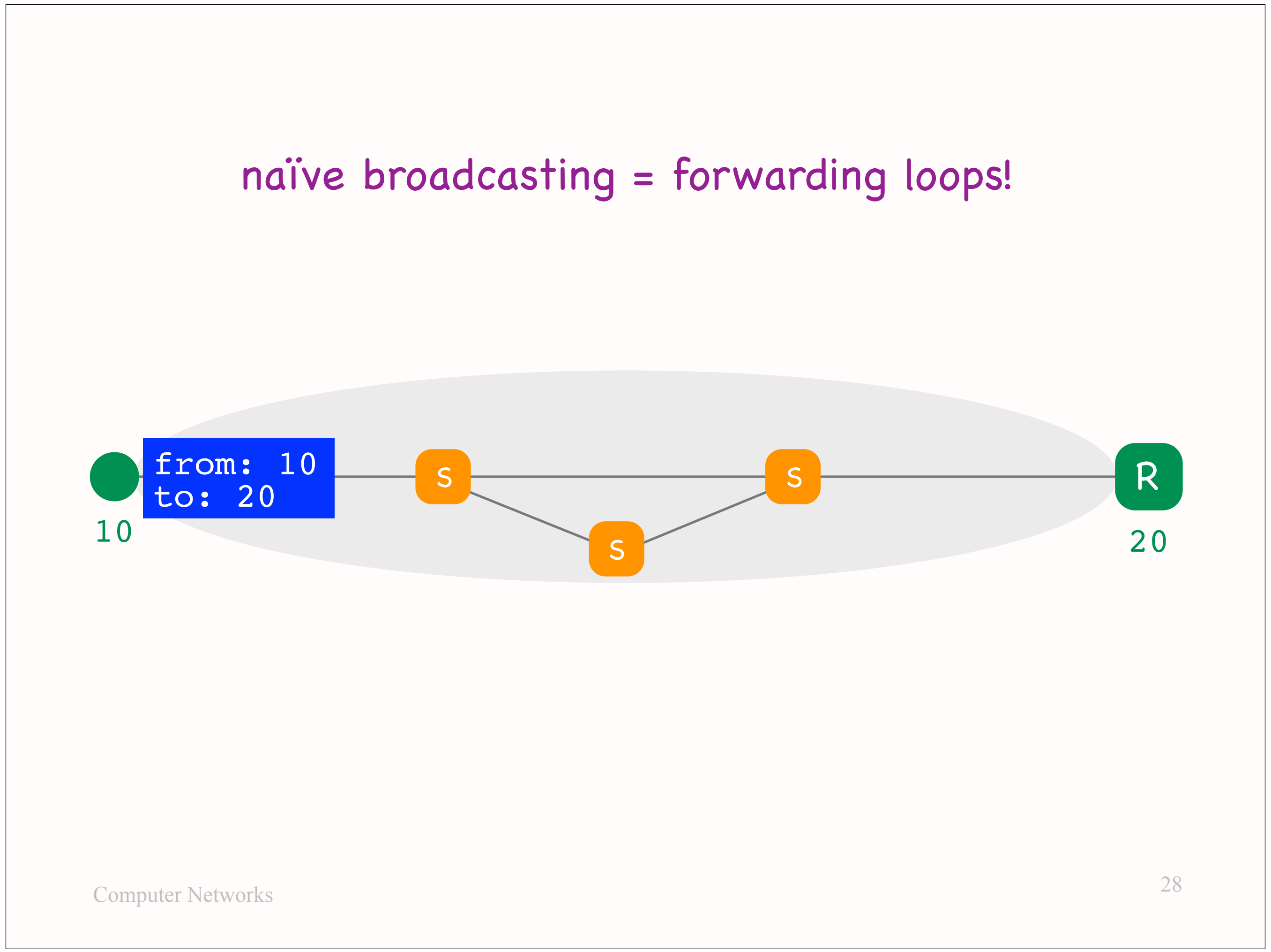
(*) It's not really all links, it's a subset of the links, in order to avoid loops.
Discussed in a couple of slides.

L2 learning

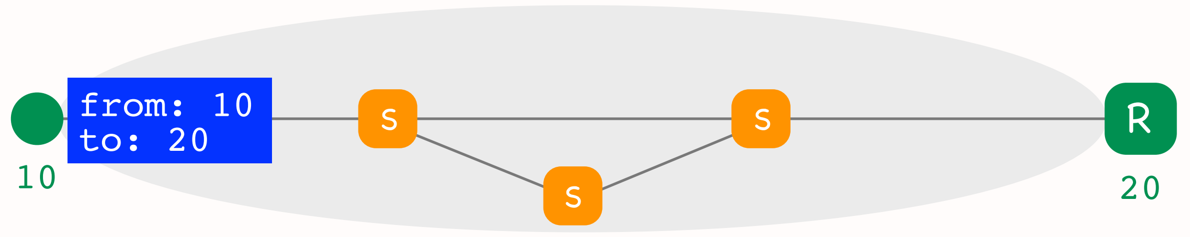
- Switch learns from traffic
  - when packet with src MAC x arrives at link y, switch adds MAC x --> link y mapping to forwarding table
- Broadcasts when it does not know
  - when packet with unknown dst MAC arrives, switch broadcasts the packet
- Serves similar role as IP routing, but…

L2 learning vs. IP routing

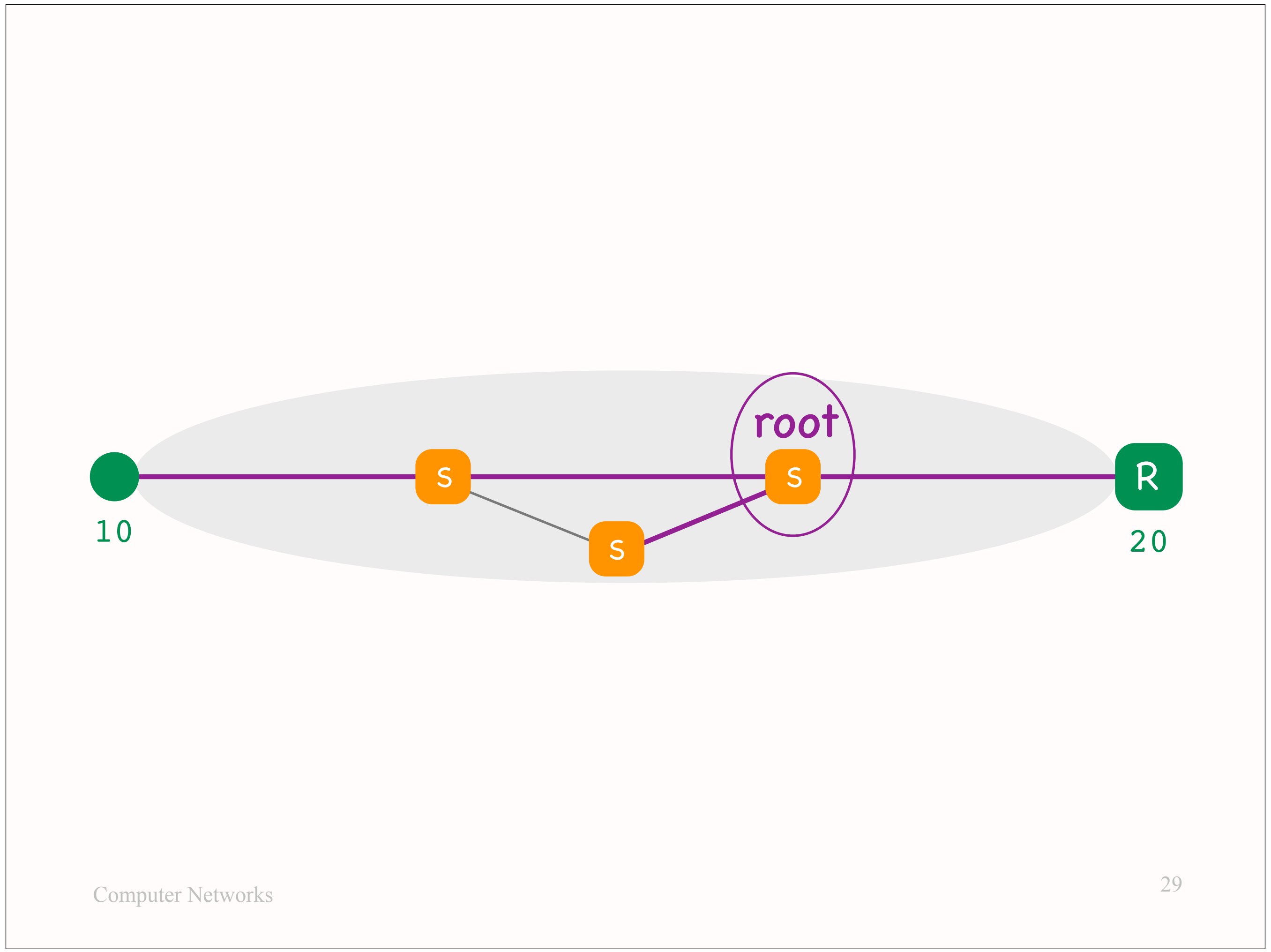
- L2 learning: relies on actual traffic
  - switches do not exchange explicit routing information
- IP routing: relies on routing protocol
  - routers exchange explicit routing messages



naïve broadcasting = forwarding loops!



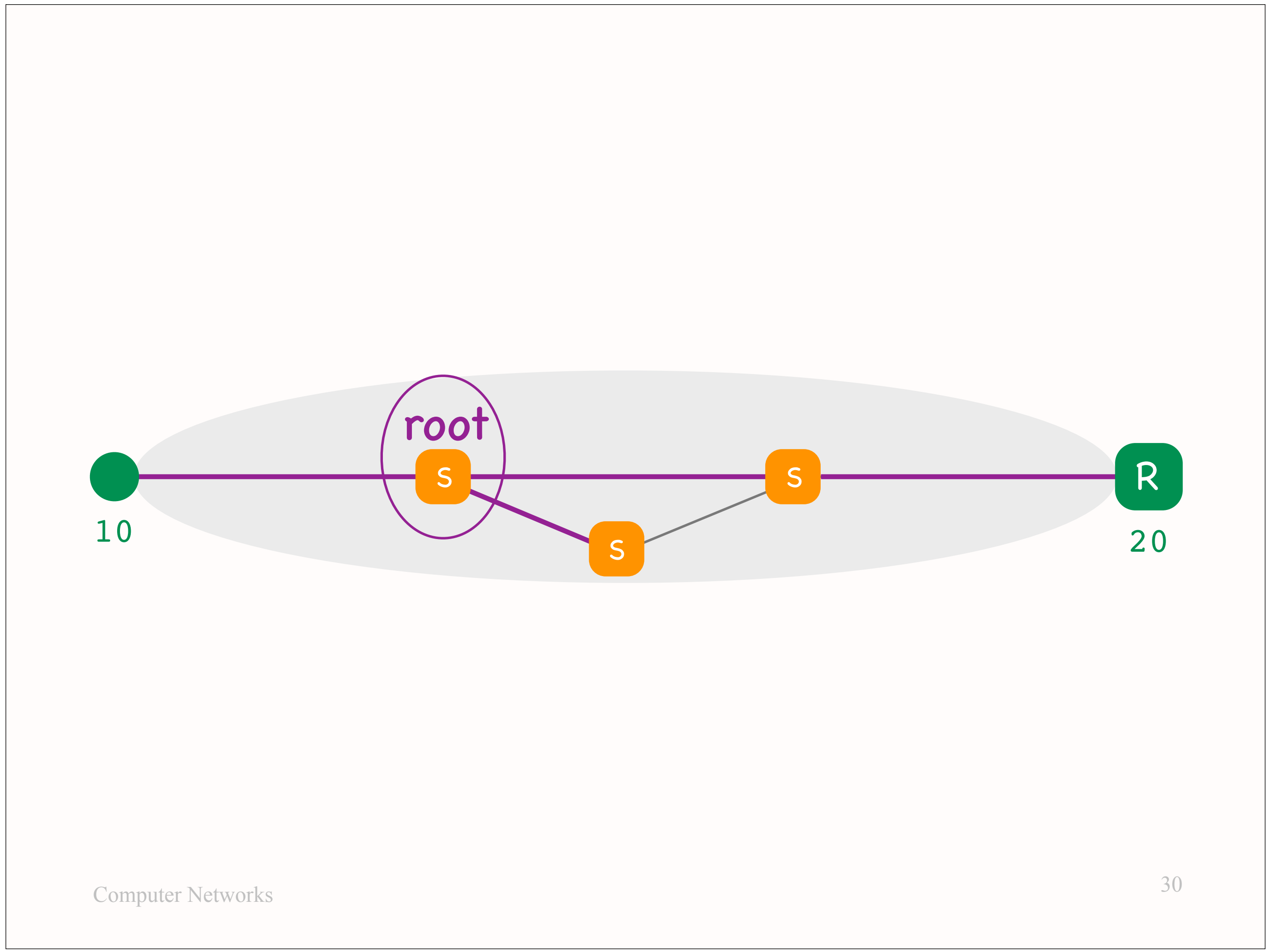
We said that, when a switch receives a packet with unknown destination MAC address, the switch broadcasts the packet.
If all switches broadcast packets with unknown destination MAC addresses to all links (except the one where the packet arrived), there may be forwarding loops.



To avoid this, all the switches in an IP subnet participate in a protocol that creates a “spanning tree.”

This is a subgraph of the IP subnet, which includes all the end-systems, routers, and switches of the subnet, but only a subset of the links. It includes just enough links to reach all the end-systems and network devices.

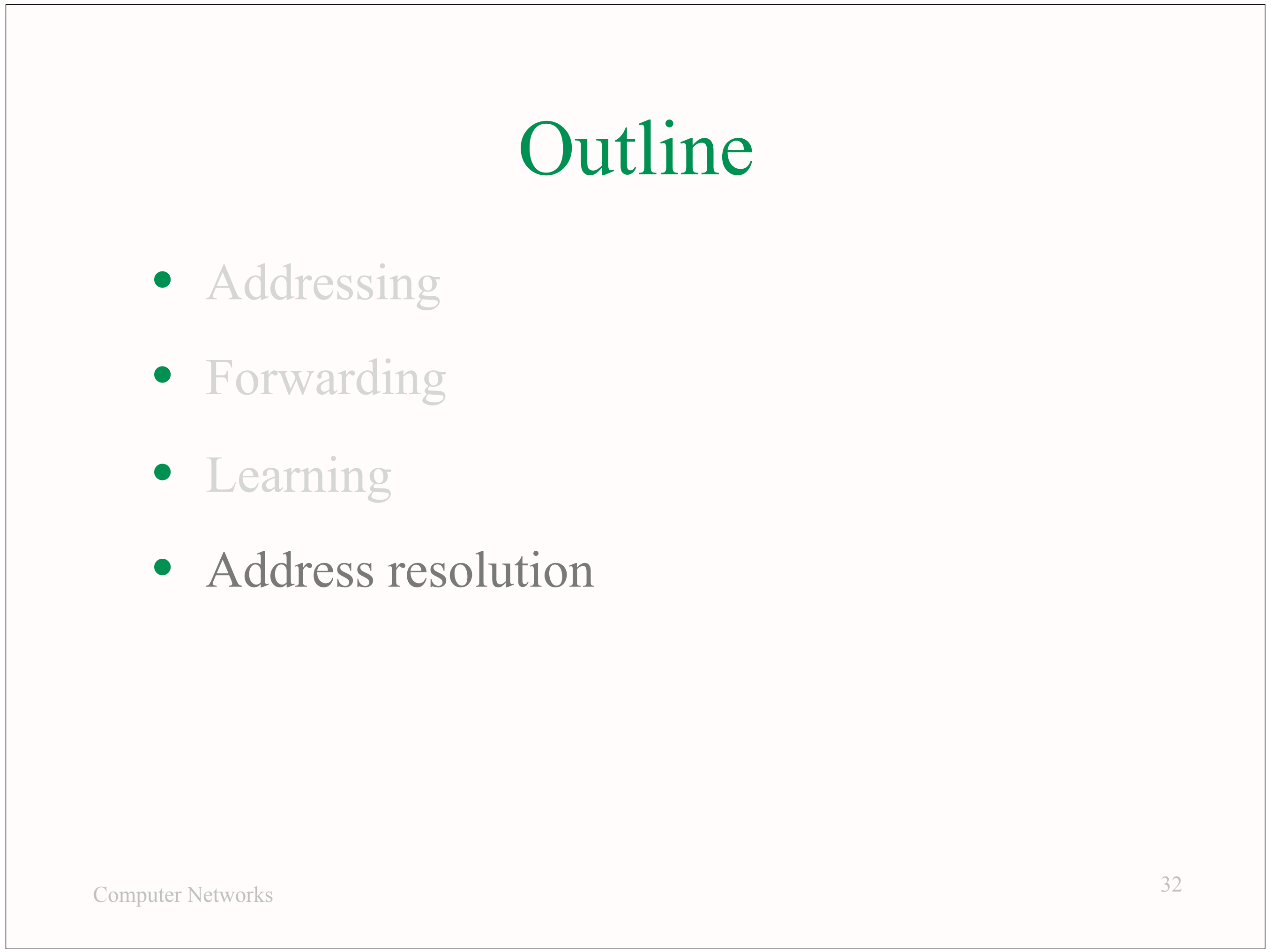
There may exist many spanning trees in an IP subnet.



Here is another example.

Spanning tree

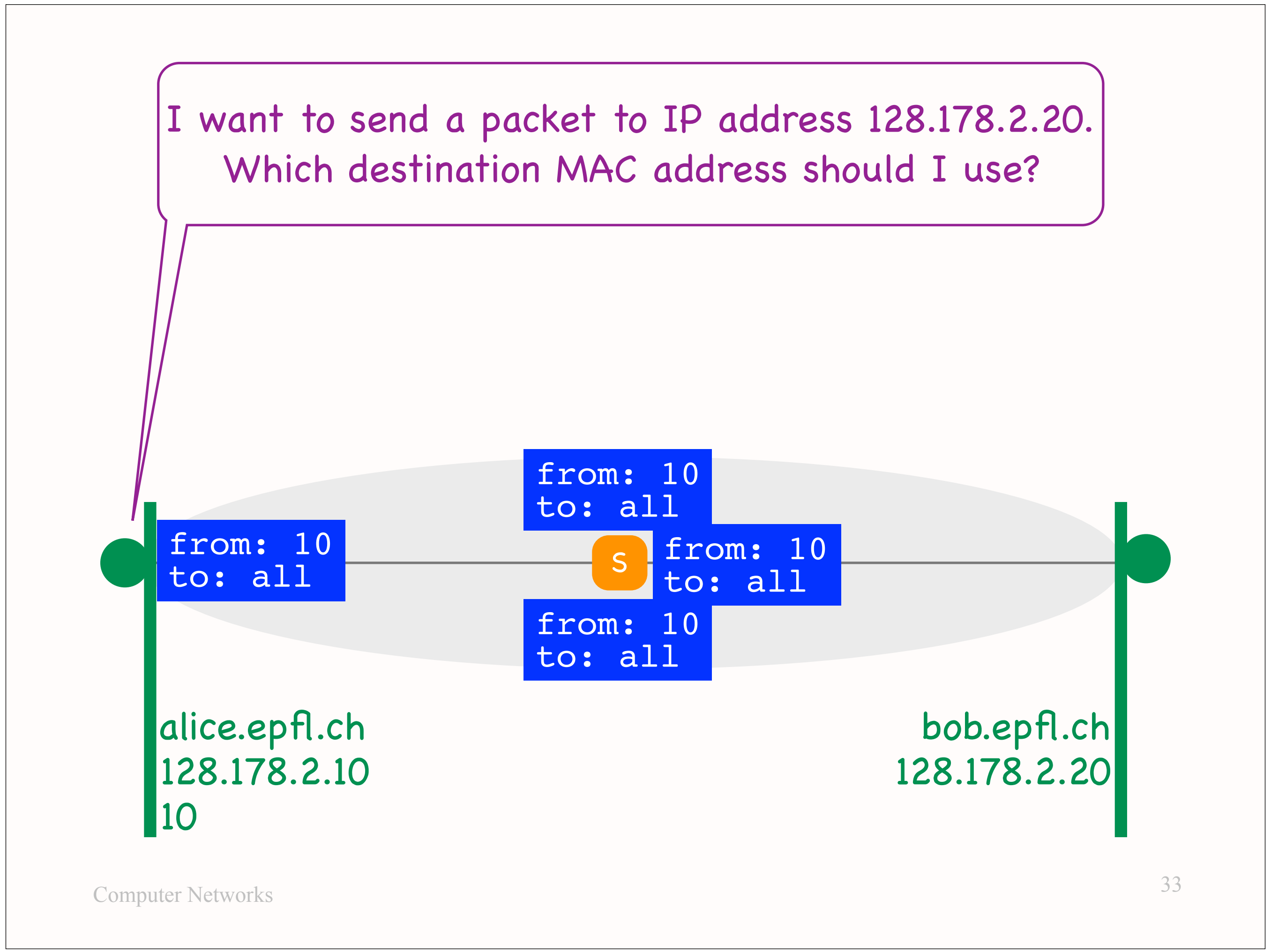
- A subgraph with special properties
  - includes all nodes + some edges
  - cannot remove an edge
    without disconnecting a node
- Useful for loop-free broadcasting
  - broadcast traffic propagated only along tree
  - prevents forwarding loops



Outline

- Addressing
- Forwarding
- Learning
- Address resolution

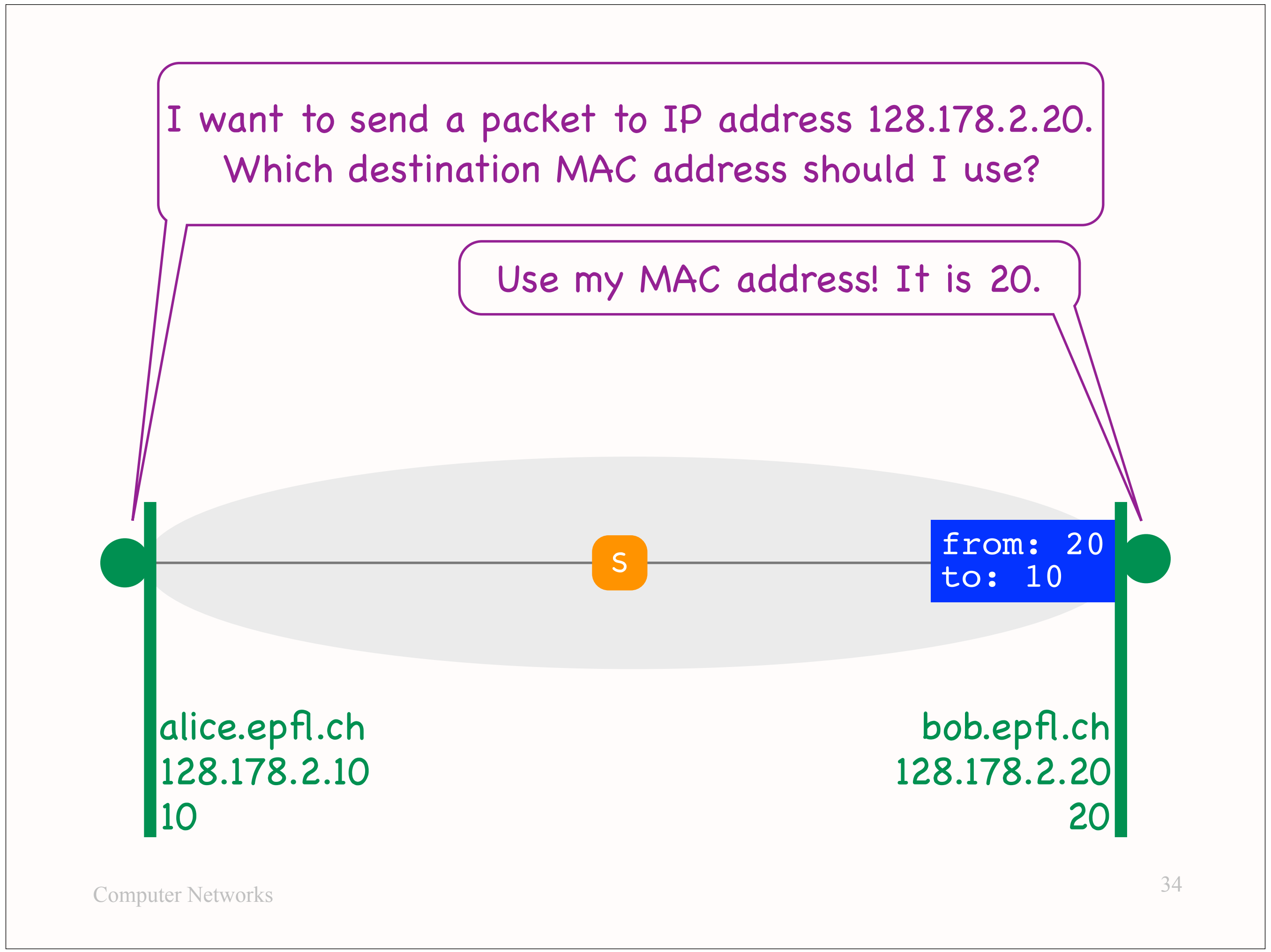
One last but crucial question remains: in an Ethernet, how does an end-system learn the destination MAC address that it should write on a packet that it wants to send?



Suppose Alice wants to send a packet to Bob, who happens to be in the same IP subnet as Alice.

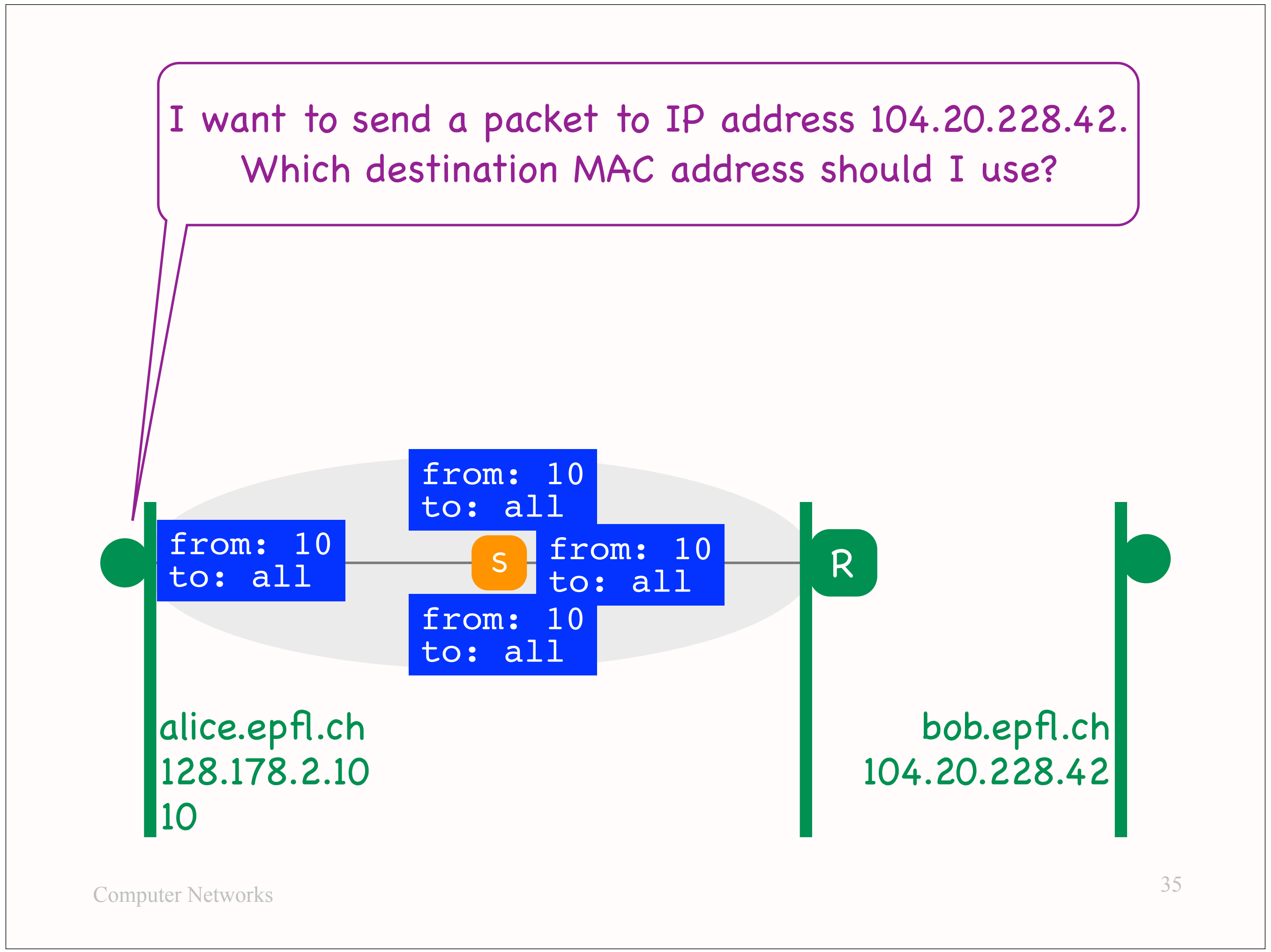
The first thing she needs is Bob's IP address, which will be the packet's destination IP address.
We know how she gets this: she asks a local DNS server.

The second thing she needs is Bob's MAC address, which will be the packet's destination MAC address.
To get this, she sends out a special request that states Bob's IP address.
This request (called an "ARP request," as we will see later) has a special destination MAC address, which is called a "broadcast address," meaning that the request is addressed to all the end-systems and routers in the local IP subnet.
So, every switch that receives this request broadcasts it.
Eventually, the request reaches Bob.


I want to send a packet to IP address 128.178.2.20.
Which destination MAC address should I use?
Use my MAC address! It is 20.
S
from: 20
to: 10
alice.epfl.ch
128.178.2.10
10
bob.epfl.ch
128.178.2.20
20
Computer Networks
34

When Bob receives the request, he recognises his IP address and sends a special response (called an “ARP response,” as we will see later), which states his MAC address.

When Alice receives Bob’s response, she learns his MAC address and is finally ready to send him a packet
(because she now knows both what destination IP address and what destination MAC address to write on the packet).



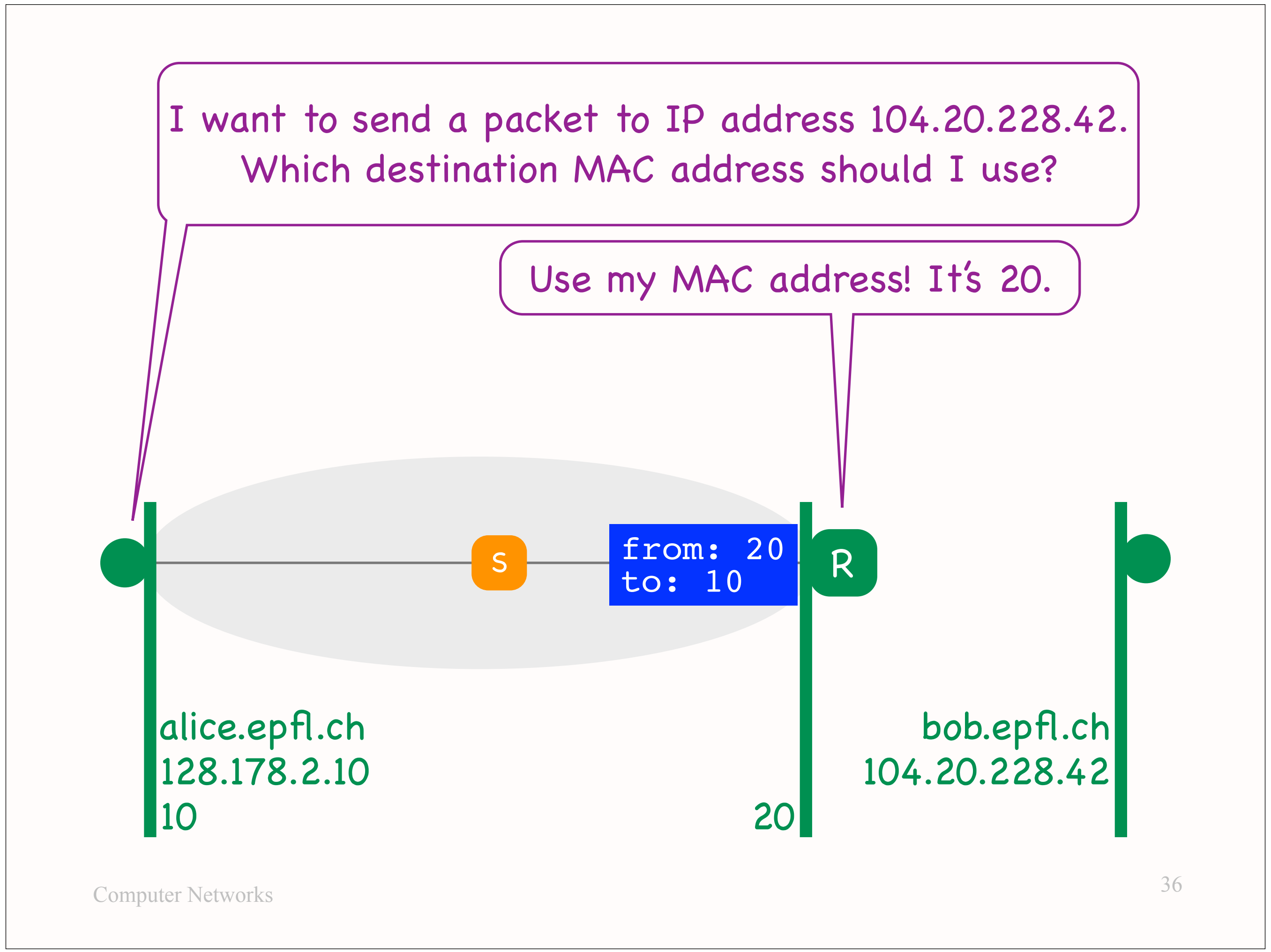
Now suppose Alice wants to send a packet to Bob, who happens to be in a different IP subnet from Alice.

As in the previous scenario, the first thing she needs is Bob’s IP address, which will be the packet’s destination IP address.
We know how she gets this: she asks a local DNS server.

Again, as in the previous scenario, the second thing Alice needs is the correct destination MAC address to write on her packet.
That’s the MAC address of router R’s network interface that belongs to the local IP subnet.

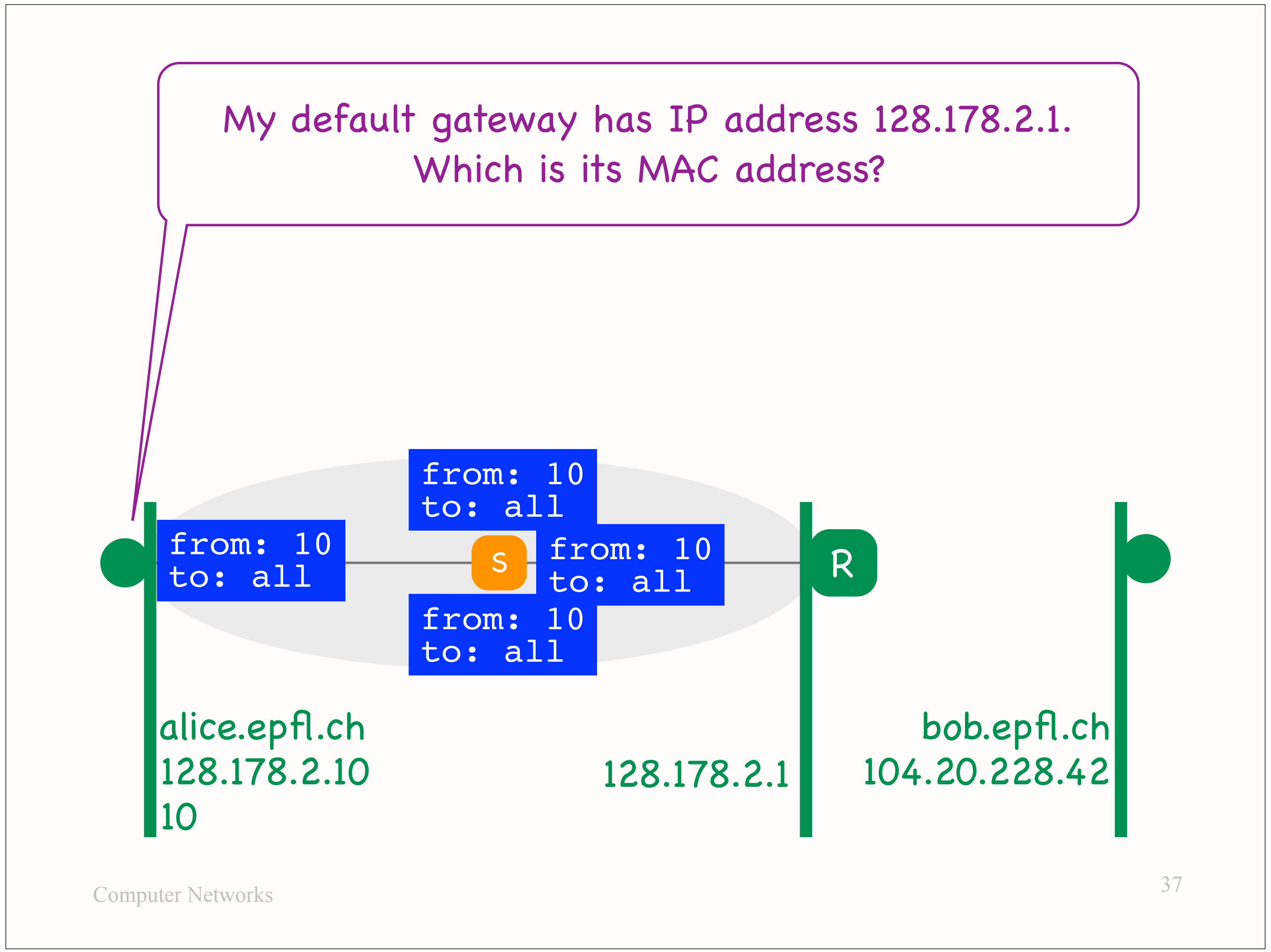
There are two ways to get this MAC address.

(1) Proxy ARP:
As in the previous scenario, Alice broadcasts an ARP request that states Bob’s IP address.
Eventually, the ARP request reaches R.


I want to send a packet to IP address 104.20.228.42. Which destination MAC address should I use?
Use my MAC address! It's 20.
S
from: 20
to: 10
R
alice.epfl.ch
128.178.2.10
10
20
bob.epfl.ch
104.20.228.42
Computer Networks
36

When R receives the ARP request, it recognizes that the target IP address belongs to a foreign IP subnet, meaning that any packet addressed to that IP address will have to exit the local IP subnet through R. Hence, R sends an ARP response that states its own MAC address.

When Alice receives R's ARP response, she learns the correct MAC address to put in her packet, and she is finally ready to send her packet to Bob.

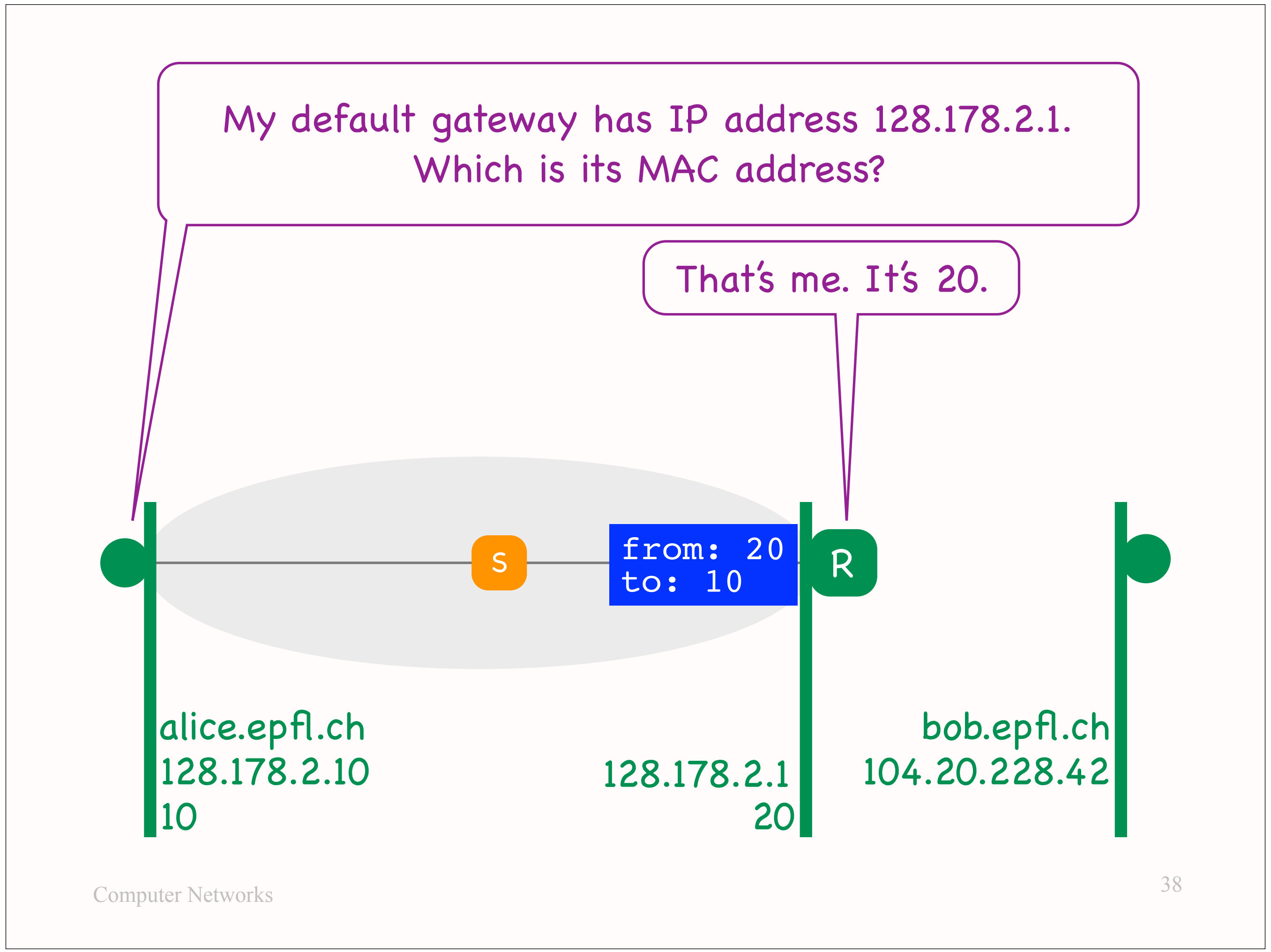


The other way for Alice to learn R's MAC address is:

(2) Default gateway:

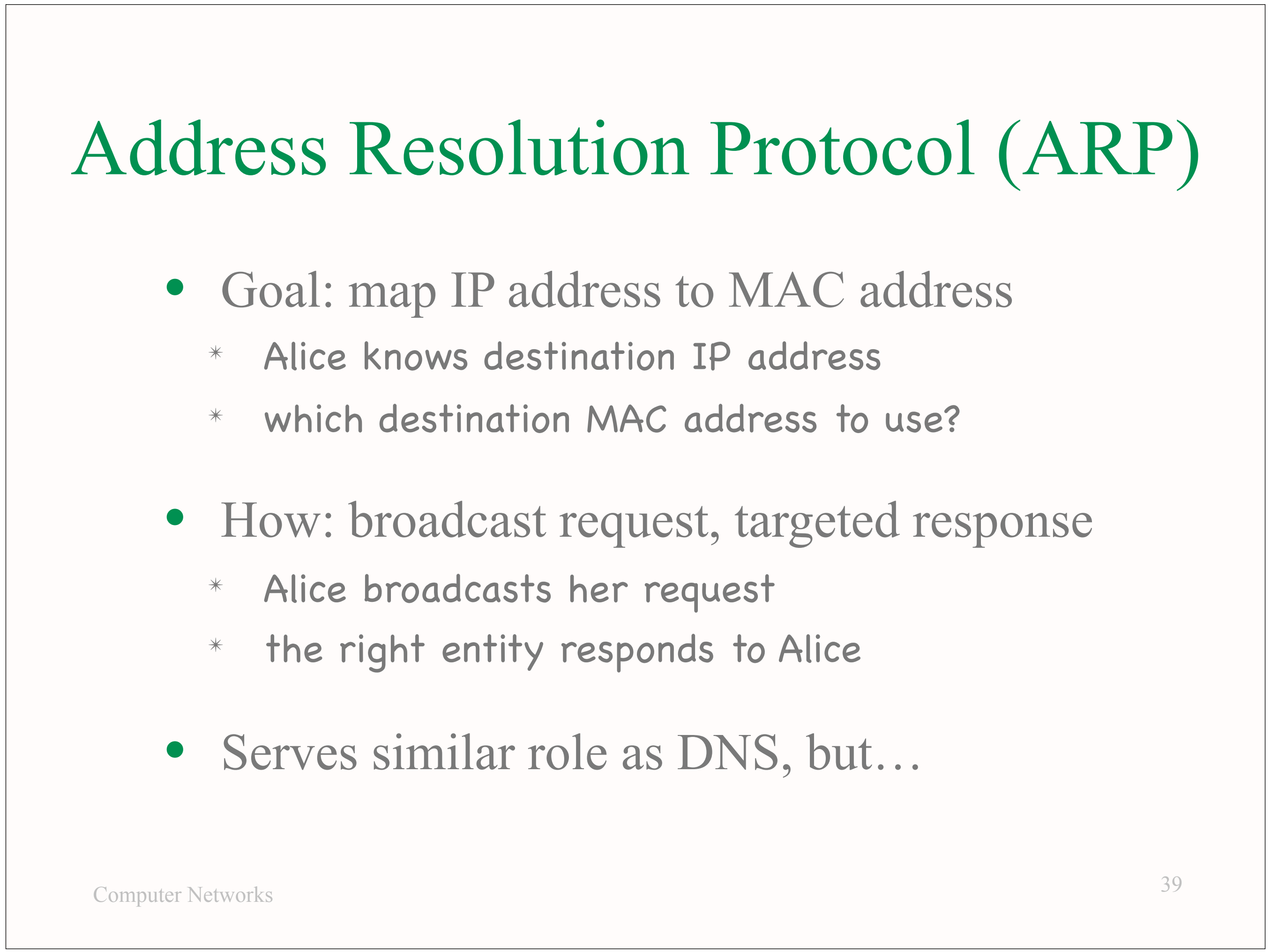
By configuration, Alice knows that R is her "default gateway," meaning that any traffic that Alice sends outside the local IP subnet will exit the subnet through R. More precisely, Alice knows R's IP address.

Hence, Alice broadcasts an ARP request that states R's IP address.
Eventually, the ARP request reaches R.



When R receives the ARP request, it recognizes its IP address and sends an ARP response that states its MAC address.

When Alice receives R's ARP response, she learns R's MAC address,
and is finally ready to send her packet to Bob.



Address Resolution Protocol (ARP)

- Goal: map IP address to MAC address
  - Alice knows destination IP address
  - which destination MAC address to use?
- How: broadcast request, targeted response
  - Alice broadcasts her request
  - the right entity responds to Alice
- Serves similar role as DNS, but…

ARP requests and responses are part of the Address Resolution Protocol (ARP)…

Broadcasting

- Alice sends request to special, broadcast destination MAC address
  - FF-FF-FF-FF-FF-FF
- Reaches every end-system and router in the local IP subnet

ARP vs. DNS

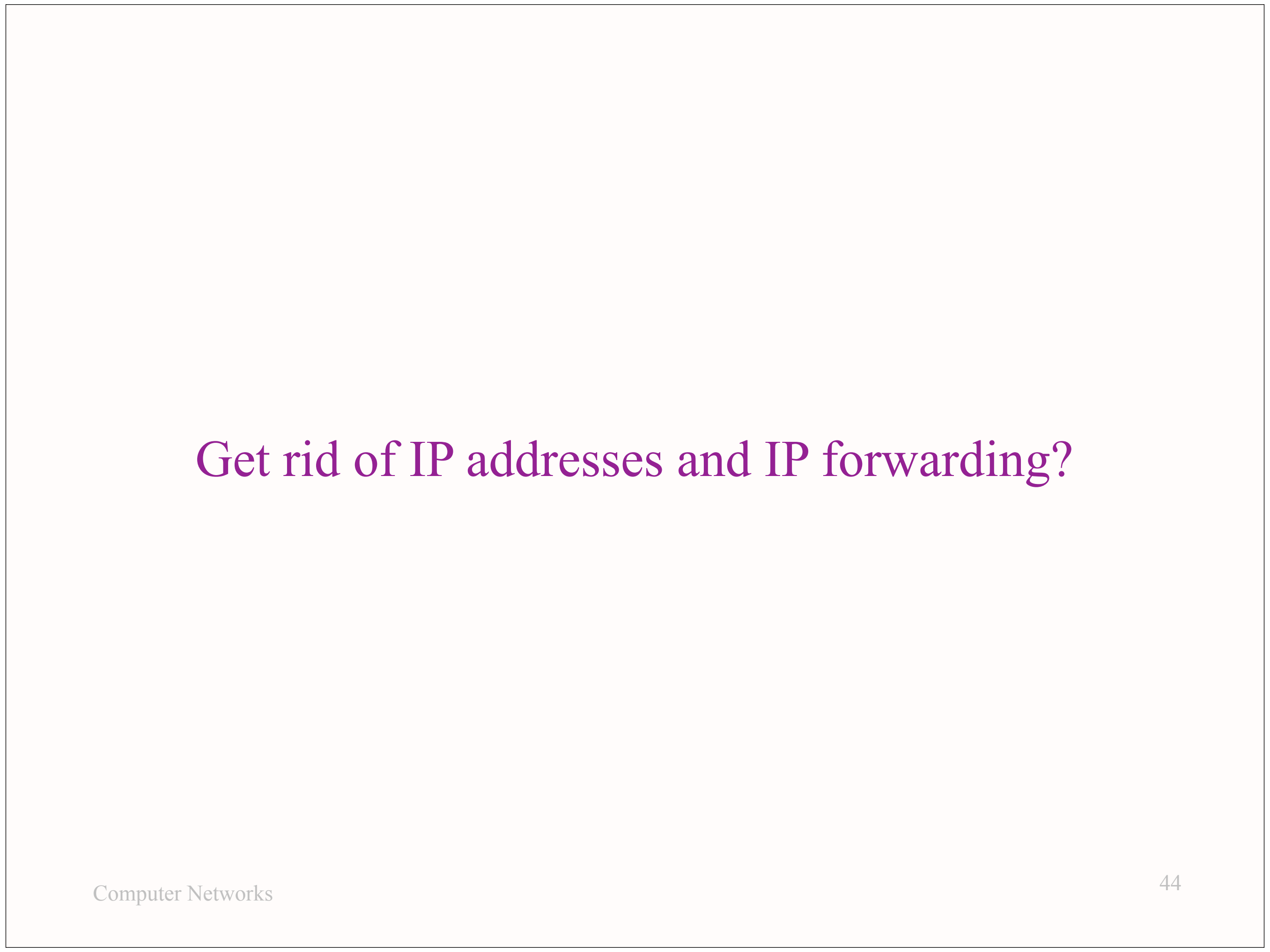
- ARP: relies on broadcasting
  - no logically centralized map
  - each entity knows its own MAC address and knows which requests to respond to
- DNS: relies on DNS infrastructure
  - logically centralized map
  - stored in DNS servers

Basic Ethernet elements

- Address Resolution Protocol
  - resolves IP address to MAC address
- L2 forwarding
  - based on MAC addresses (which are flat)
- L2 learning
  - populates switch forwarding table

Basic Ethernet elements

- Address Resolution Protocol [rel. to DNS]
  - resolves IP address to MAC address
- L2 forwarding [rel. to IP forwarding]
  - based on MAC addresses (which are flat)
- L2 learning [rel. to IP routing]
  - populates switch forwarding table

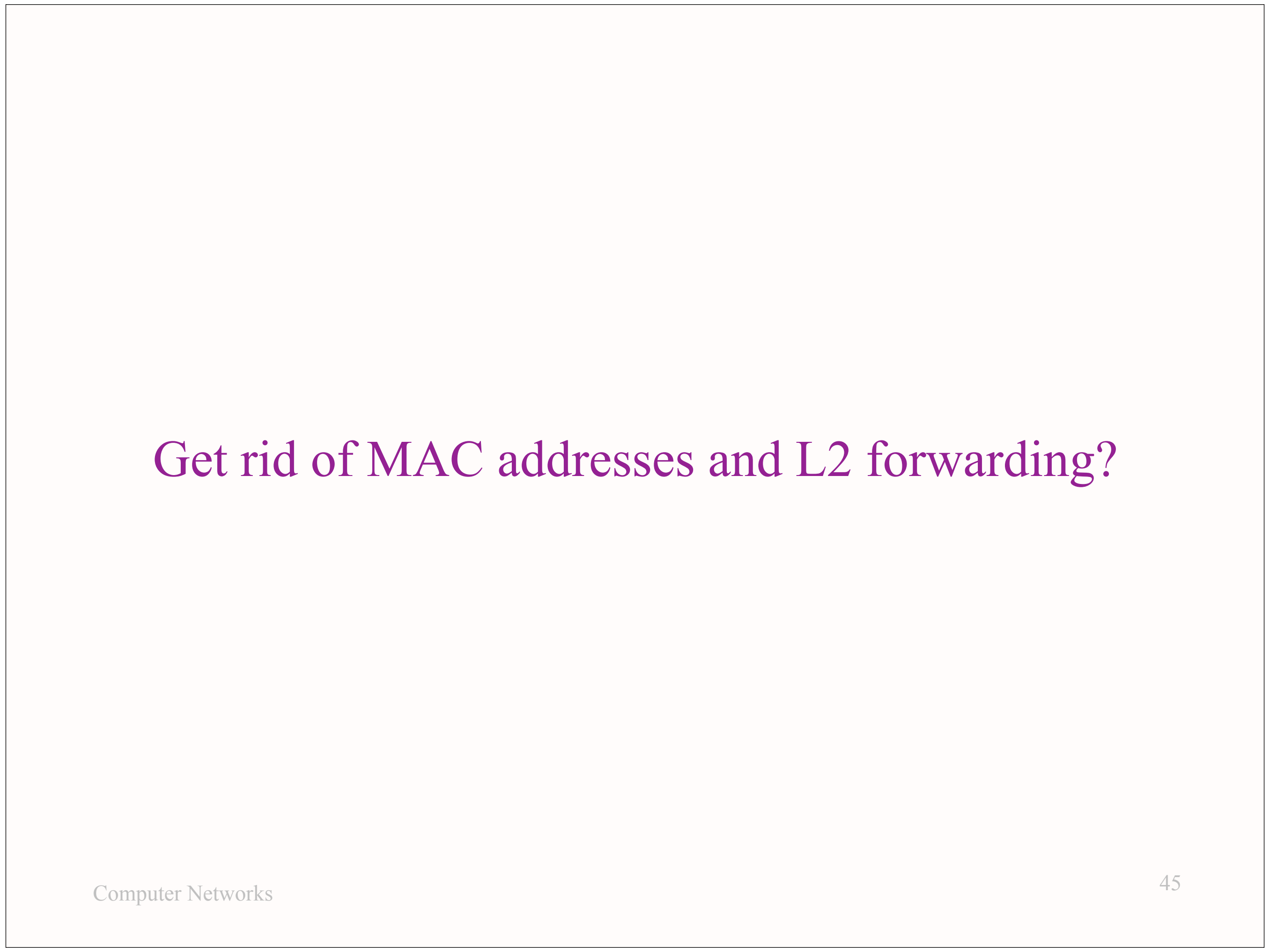


Get rid of IP addresses and IP forwarding?

Could we get rid of IP addresses and IP routers/forwarding? Could all Internet end-systems have only MAC addresses and be interconnected through link-layer switches? In other words, could the Internet be one big IP subnet?

No, this would not scale:

- A network device located in the middle of the Internet may see traffic from millions of end-systems. The forwarding table would need to be large enough to fit individual entries for each active end-system.
- Imagine what would happen if every DNS request was broadcast to everyone on the Internet in the hope that the right entity will eventually answer.



Get rid of MAC addresses and L2 forwarding?

Could all Internet end-systems have only IP addresses and be interconnected through IP routers?

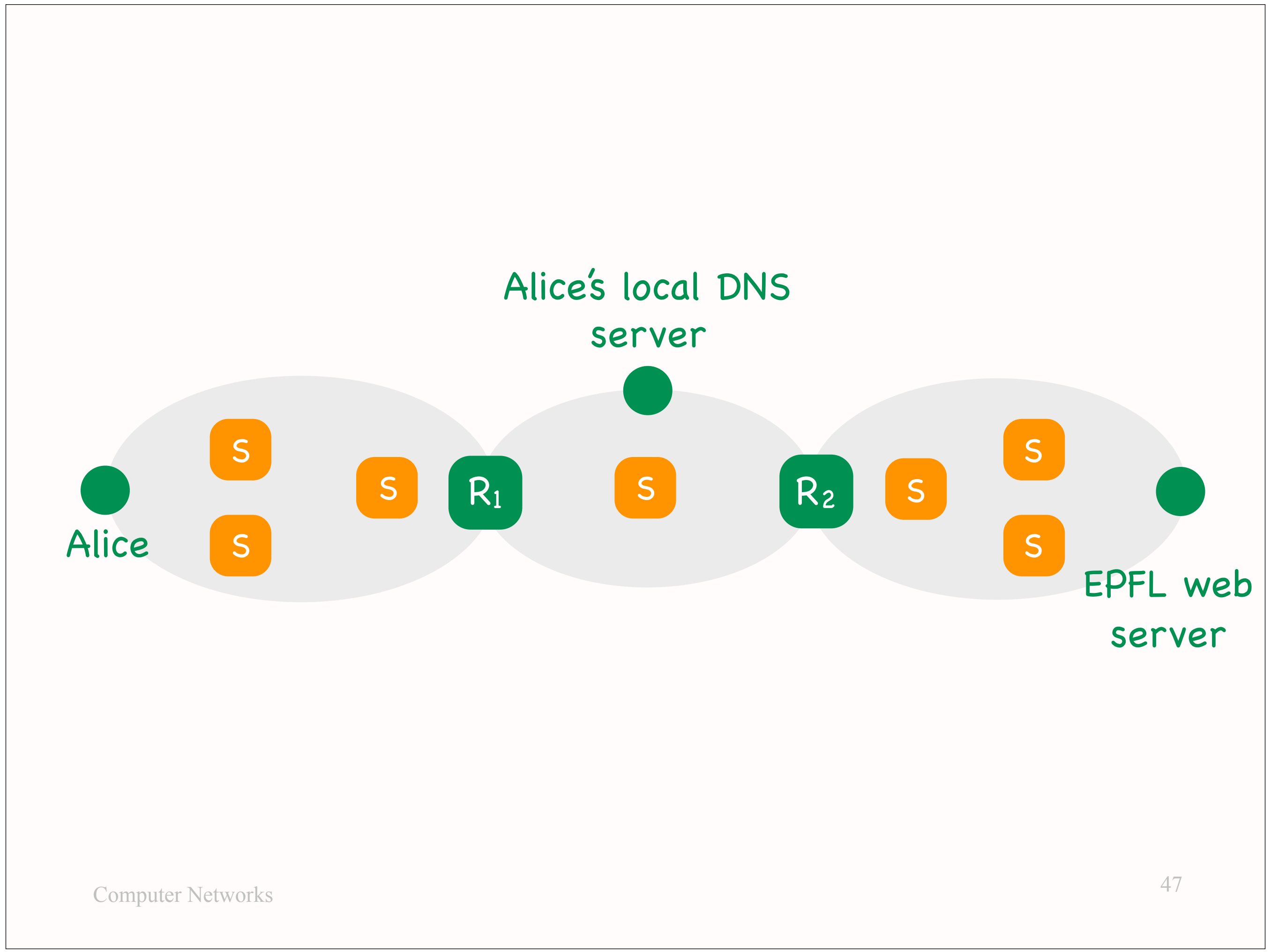
Yes, that would scale.

But, it would remove flexibility.

The Internet was fundamentally designed as a network of networks (IP subnets), not a single network. In this lecture, we saw only one type of IP subnet (Ethernet), but there are many others, and we want to be able to interconnect all of them. If we only used IP addresses and IP routers, then we would need to replace all the network devices in all these different types of IP subnets with IP routers.

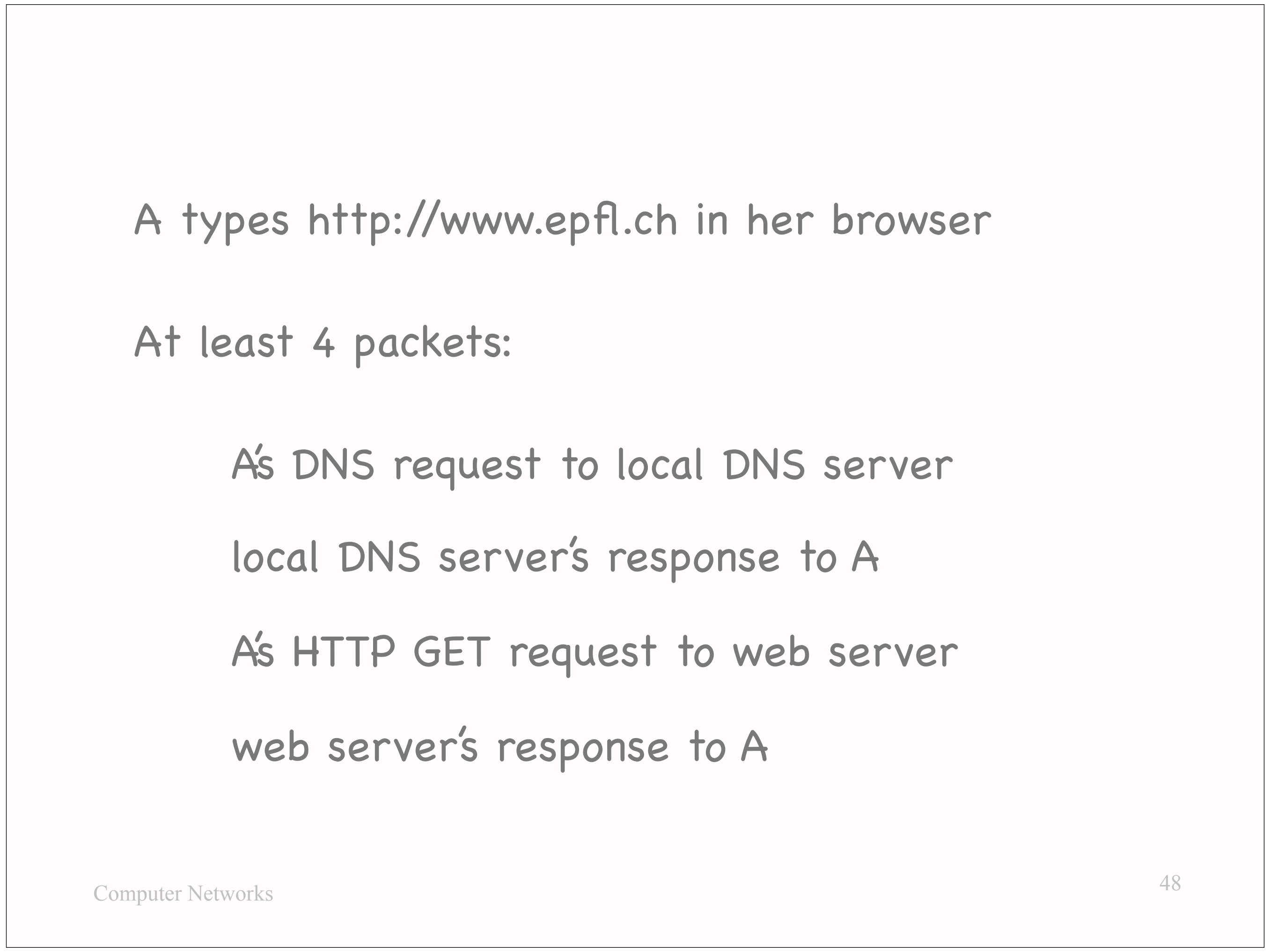
Three levels of hierarchy

- IP subnet
  - L2 forwarding
  - L2 learning
- Autonomous System (AS)
  - IP (L3) forwarding
  - intra-domain routing
- Internet
  - IP (L3) forwarding
  - inter-domain routing (BGP)



Now we will try to “put it all together,” i.e., discuss what happens from beginning to end, when an Internet end-system sends a packet to another Internet end-system.

Consider the topology shown above.

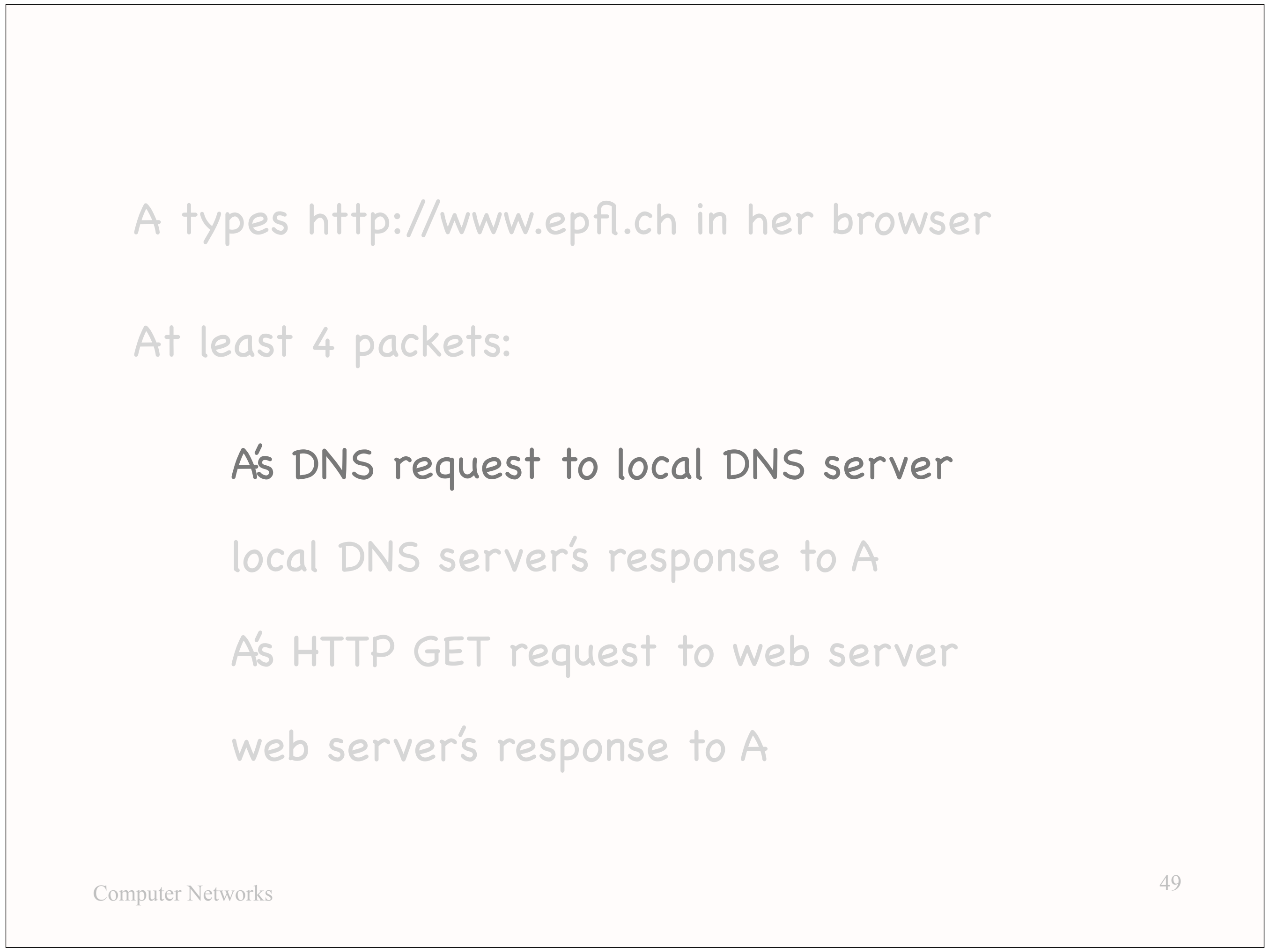


A types http://www.epfl.ch in her browser

At least 4 packets:

- A's DNS request to local DNS server
- local DNS server's response to A
- A's HTTP GET request to web server
- web server's response to A

Suppose Alice (A) types a URL in her web browser.
This results in at least 4 packets…



A types http://www.epfl.ch in her browser

At least 4 packets:

- A's DNS request to local DNS server
- local DNS server's response to A
- A's HTTP GET request to web server
- web server's response to A

We will focus on one single packet: The DNS request that Alice sends to her local DNS server to obtain the IP address of the EPFL web server.

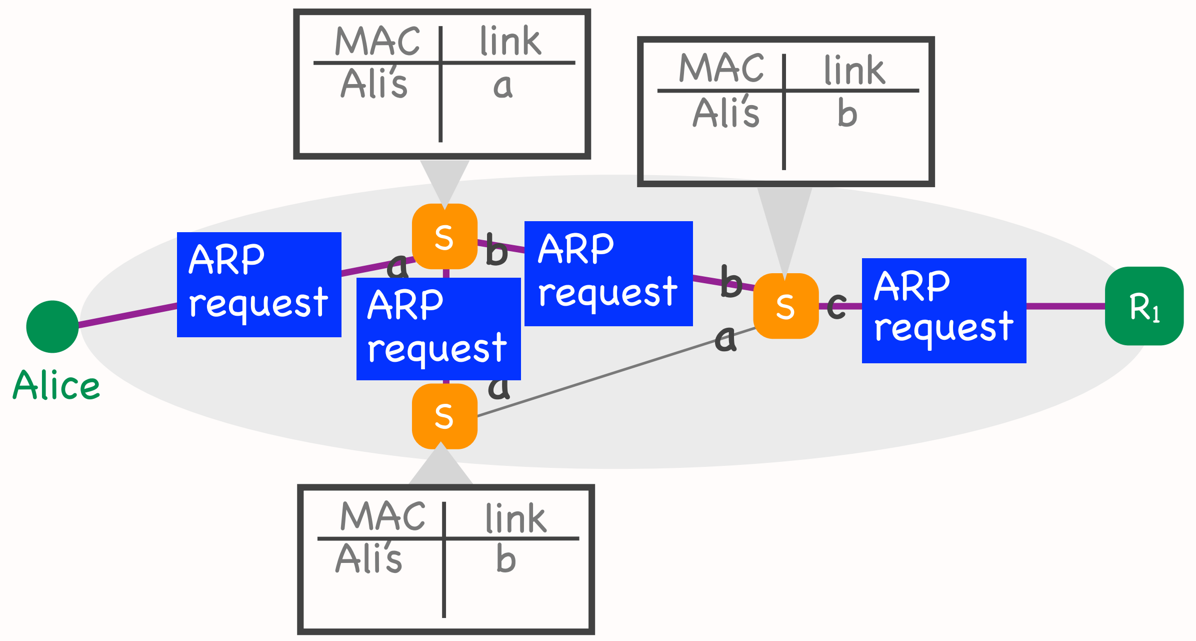
1. A's DNS client process creates DNS request

2. Passed down to transport, network layer
   - IP src: A's IP address
   - IP dst: local DNS server's IP address

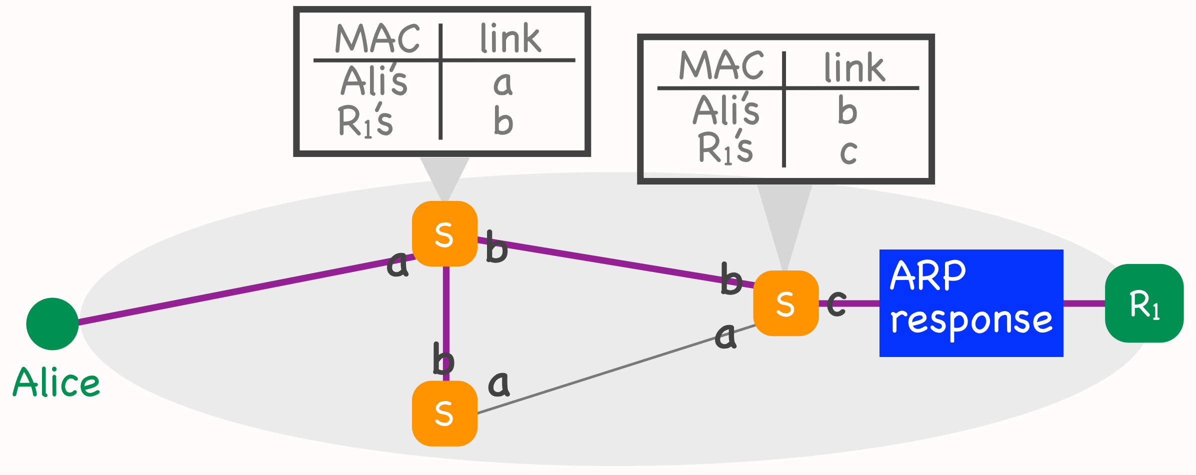
3. A's network layer sends ARP request
   - to resolve DNS server's IP address

src MAC: Alice's

dst MAC: broadcast



4. R_1's network layer sends ARP response

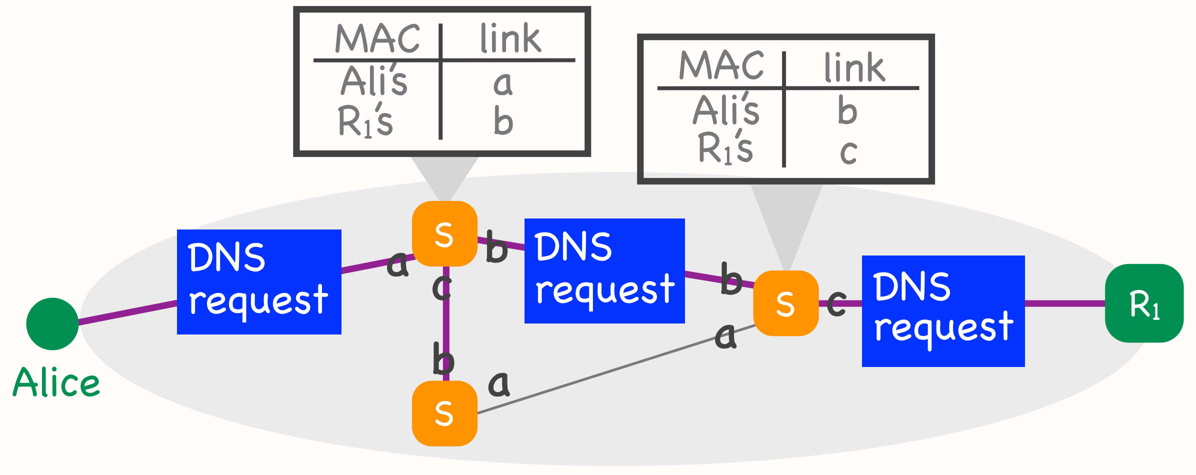


src MAC: R_1's

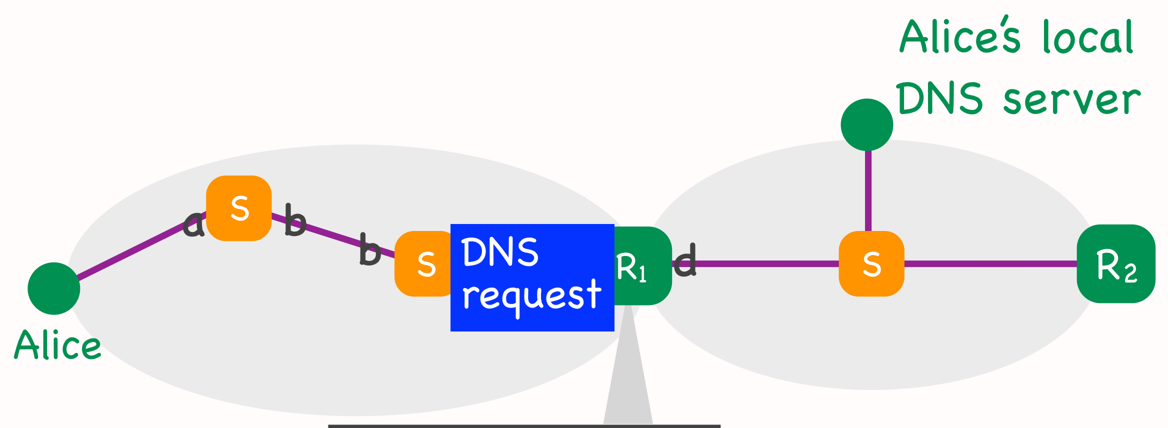
dst MAC: Alice's

5. A's network layer sends DNS request
    - it now knows what dst MAC address to use

src MAC: Alice's	src IP: Alice's
dst MAC: R_1's	dst IP: DNS server's



6. R_1's network layer performs IP forwarding



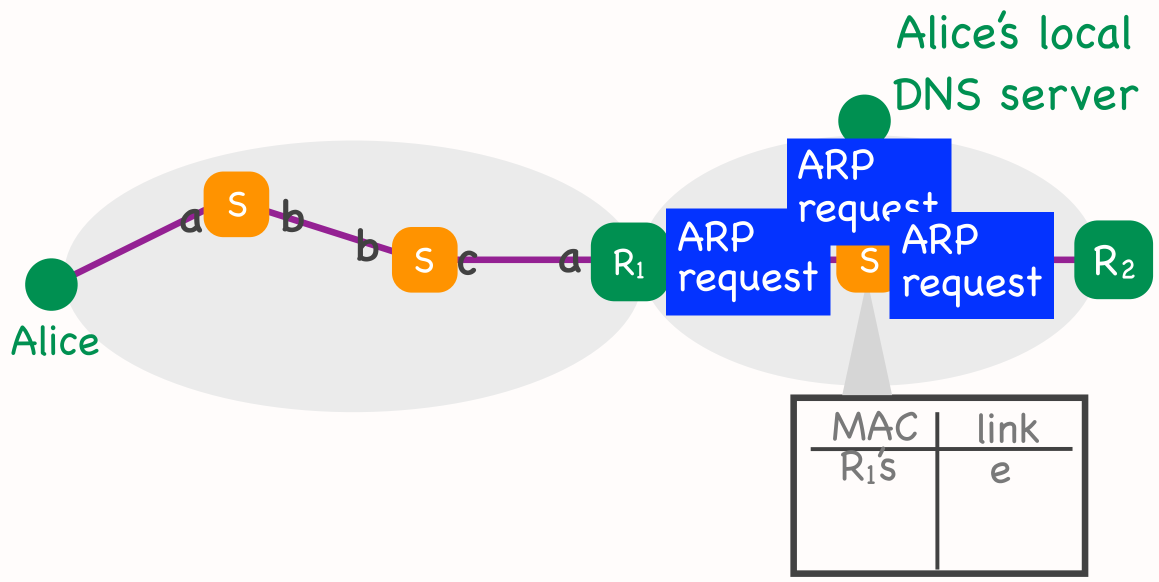
IP prefix	link
11.2.34.0/24	a
8.0.0.0/8	c
19.7.0.0/16	d
...	...

7. R_1's network layer sends ARP request

- to resolve DNS server's IP address

src MAC: R_1's

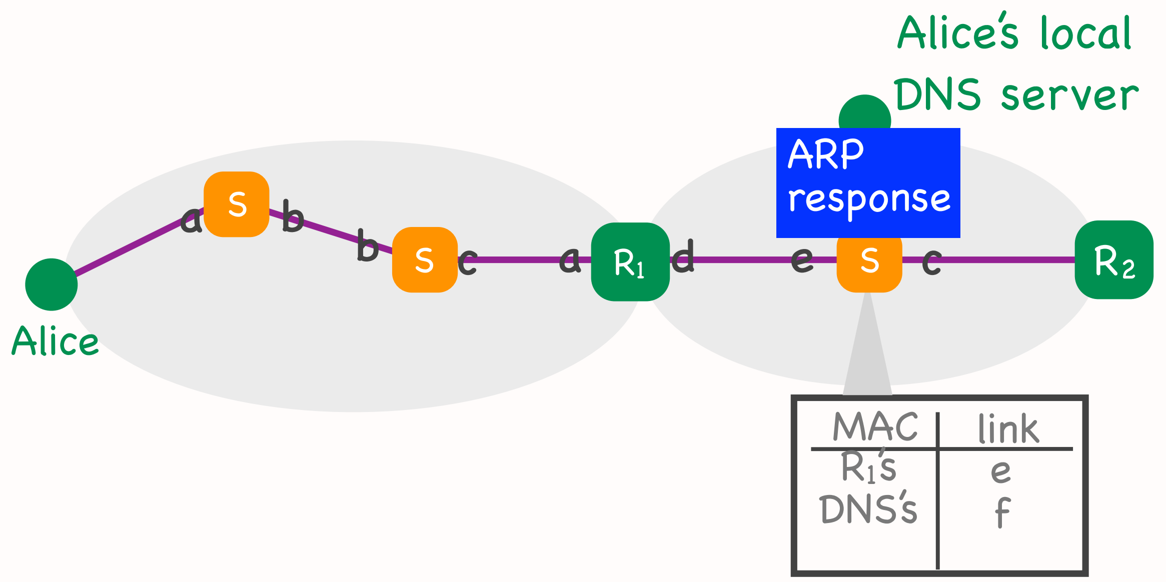
dst MAC: broadcast



8. DNS server's network layer sends ARP response

src MAC: DNS server's

dst MAC: R_1's



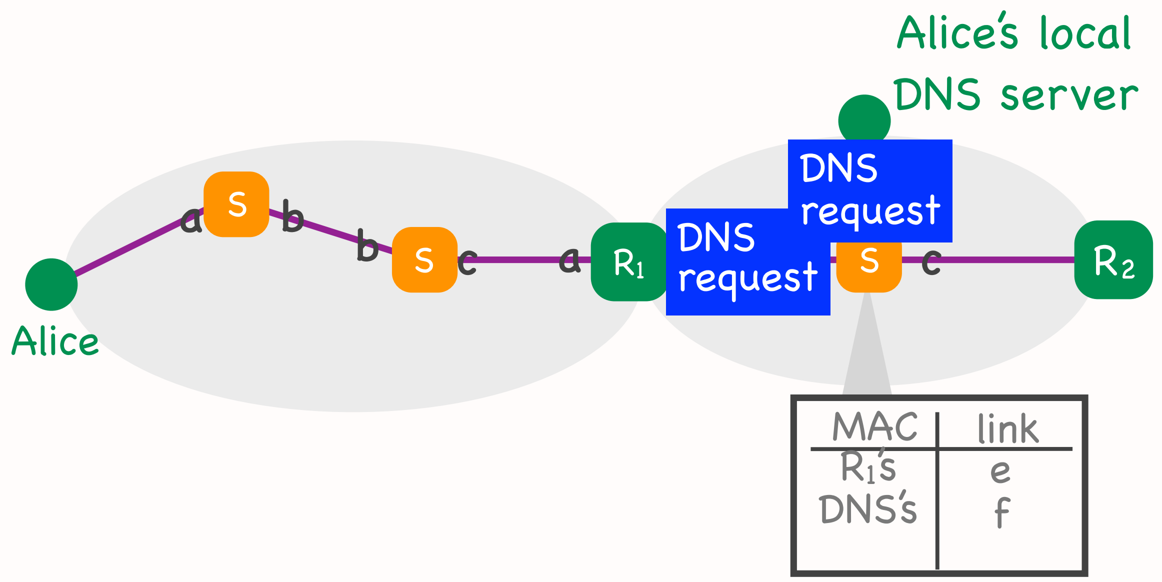
MAC	link
R_1's	e
DNS's	f

9. R_1's network layer forwards DNS request

- it now knows what dst MAC address to use

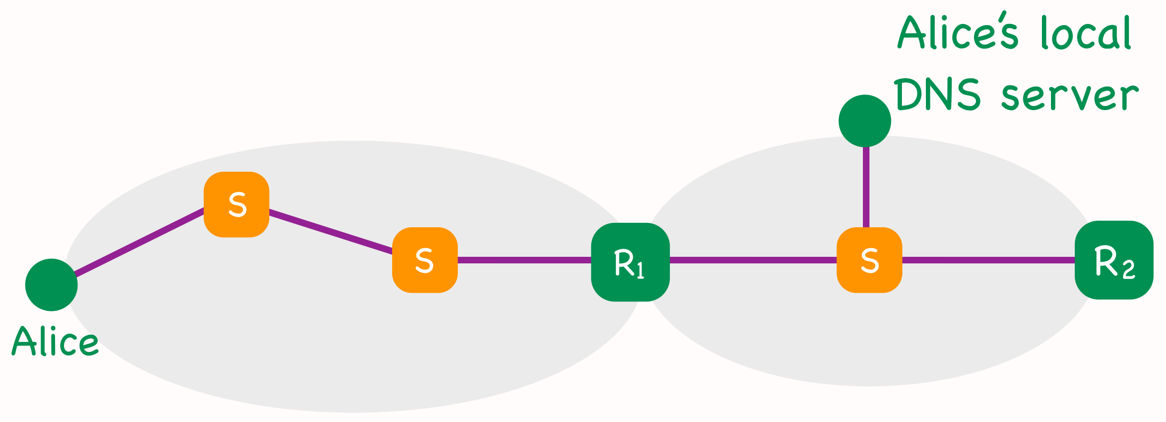
src MAC: R_1's
dst MAC: DNS server's

src IP: Alice's
dst IP: DNS server's



MAC	link
R_1's	e
DNS's	f

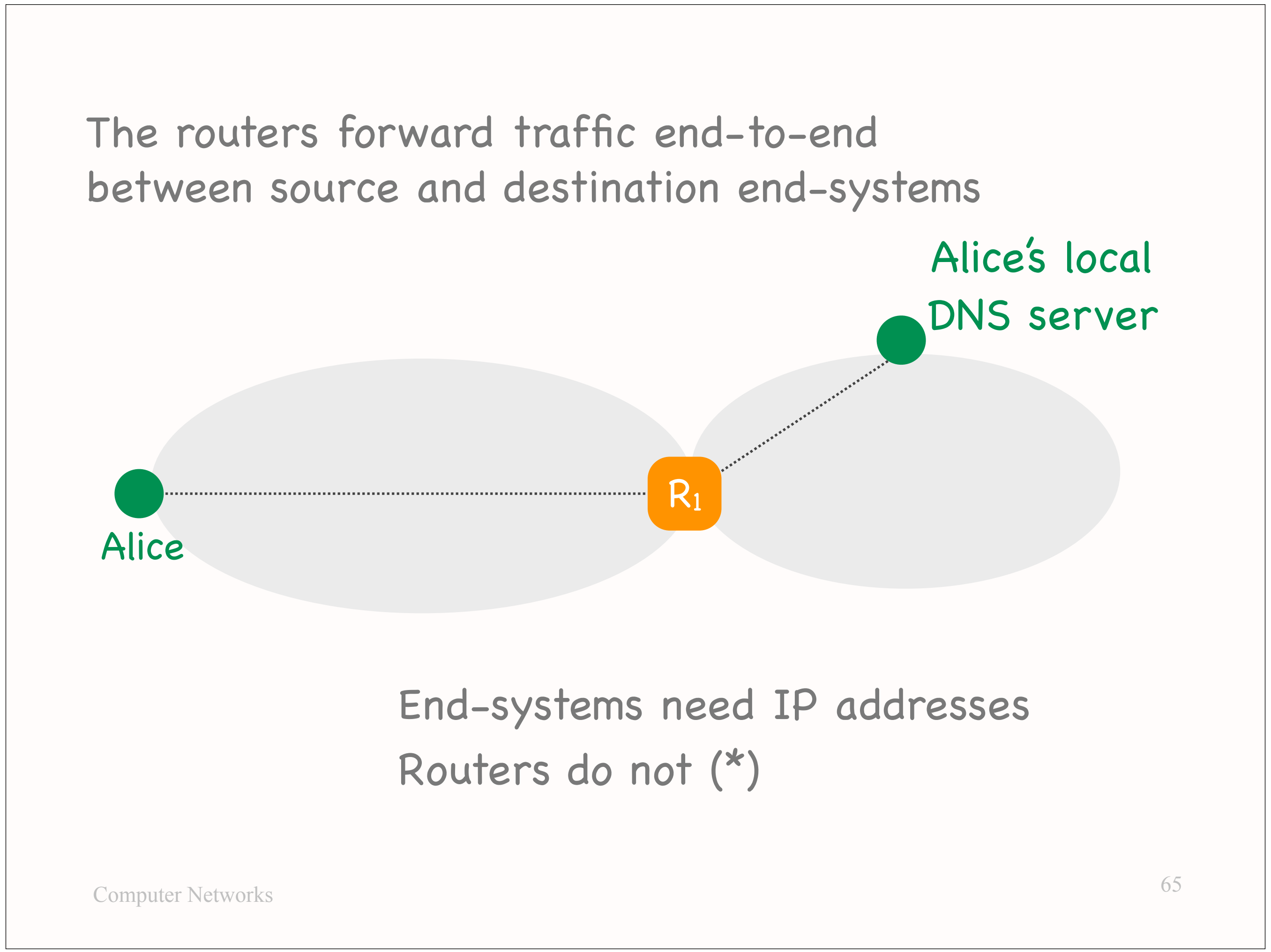
The switches forward traffic within local IP subnet between end-systems and routers



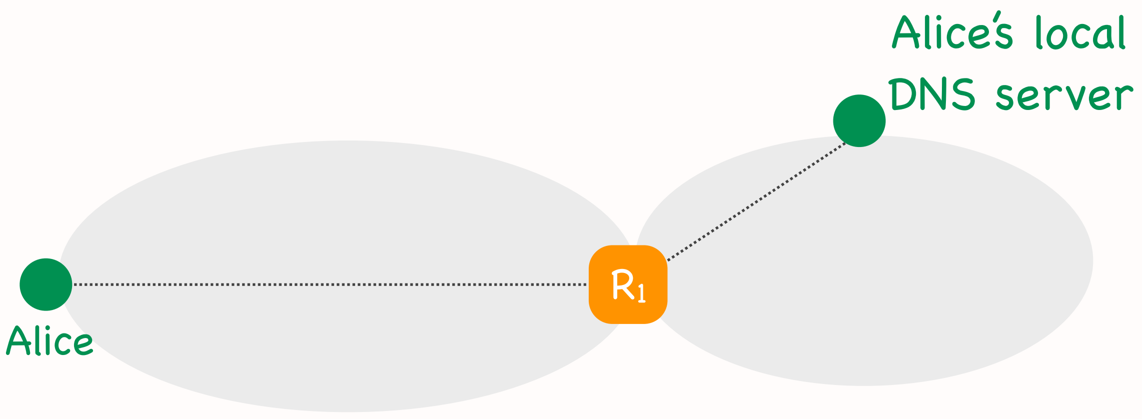
End-systems and routers need MAC addresses

Switches do not (*)

(*) But, as we will see, switches do have MAC addresses.

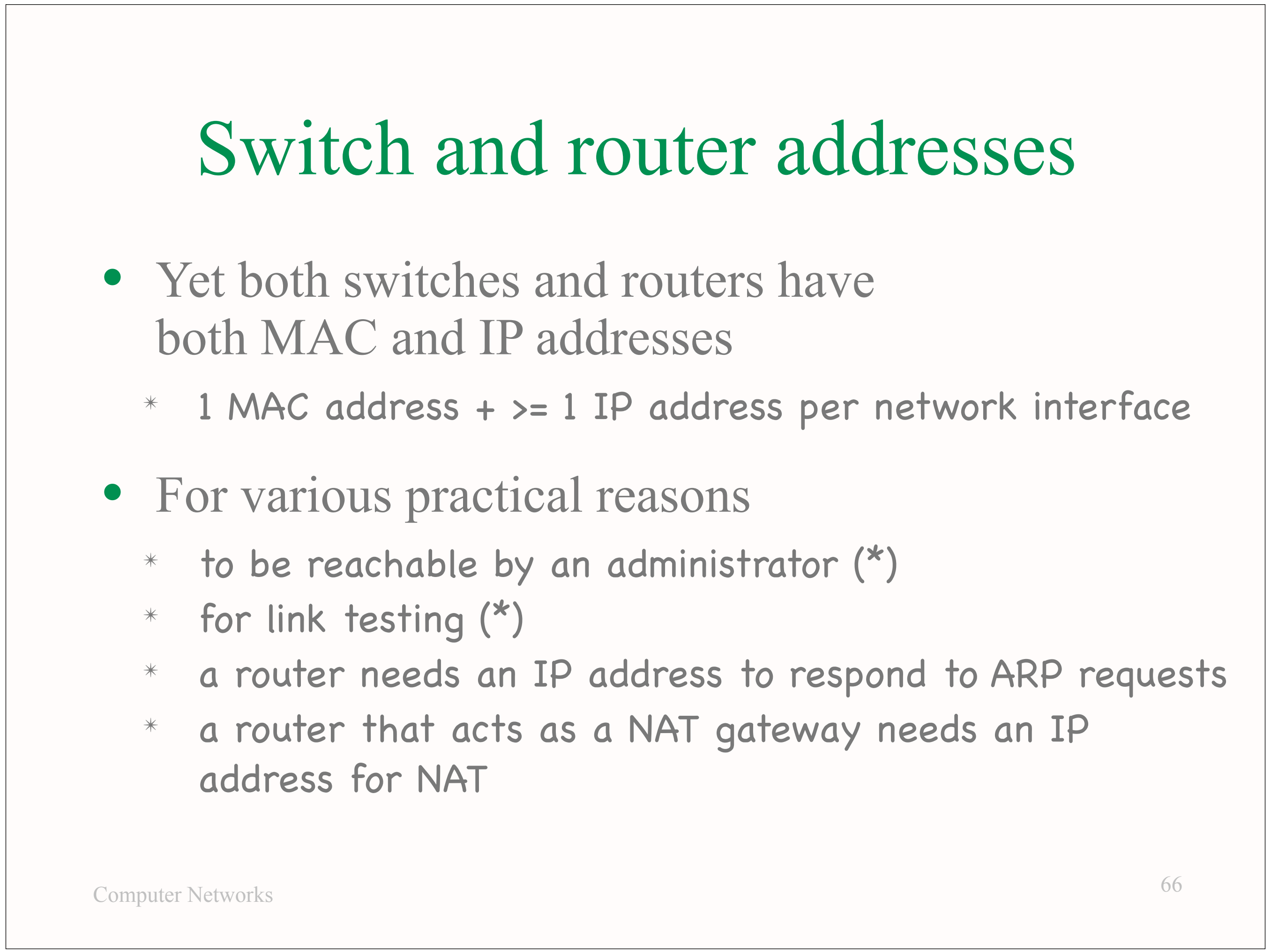


The routers forward traffic end-to-end
between source and destination end-systems



End-systems need IP addresses
Routers do not (*)

(*) But, as we will see, routers do have IP addresses.



Switch and router addresses

- Yet both switches and routers have both MAC and IP addresses
  - 1 MAC address + >= 1 IP address per network interface
- For various practical reasons
  - to be reachable by an administrator (*)
  - for link testing (*)
  - a router needs an IP address to respond to ARP requests
  - a router that acts as a NAT gateway needs an IP address for NAT

(*) A network administrator often needs to log into a network device and inspect or configure it. For this reason, network devices also act as standard computers, i.e., one can login to them and run processes. To be reachable as a standard computer, a network device must have at least one MAC address and one IP address (though not necessarily 1 MAC address and 1 IP address **per network interface**).

(**) About link testing: Imagine that an administrator wants to test a specific physical link (does it lose or corrupt packets? what is its latency?) One way to do this is to put packets on one end of the link and count/inspect the packets that arrive at the other end. One way to implement such a simple test, is to have sender and receiver processes (e.g., a simple UDP sender and receiver) running on the devices that are at the two ends of the link. In order for these processes to exchange packets over the specific link, each of the two network interfaces at the two ends of the link must have 1 MAC and at least 1 IP address.