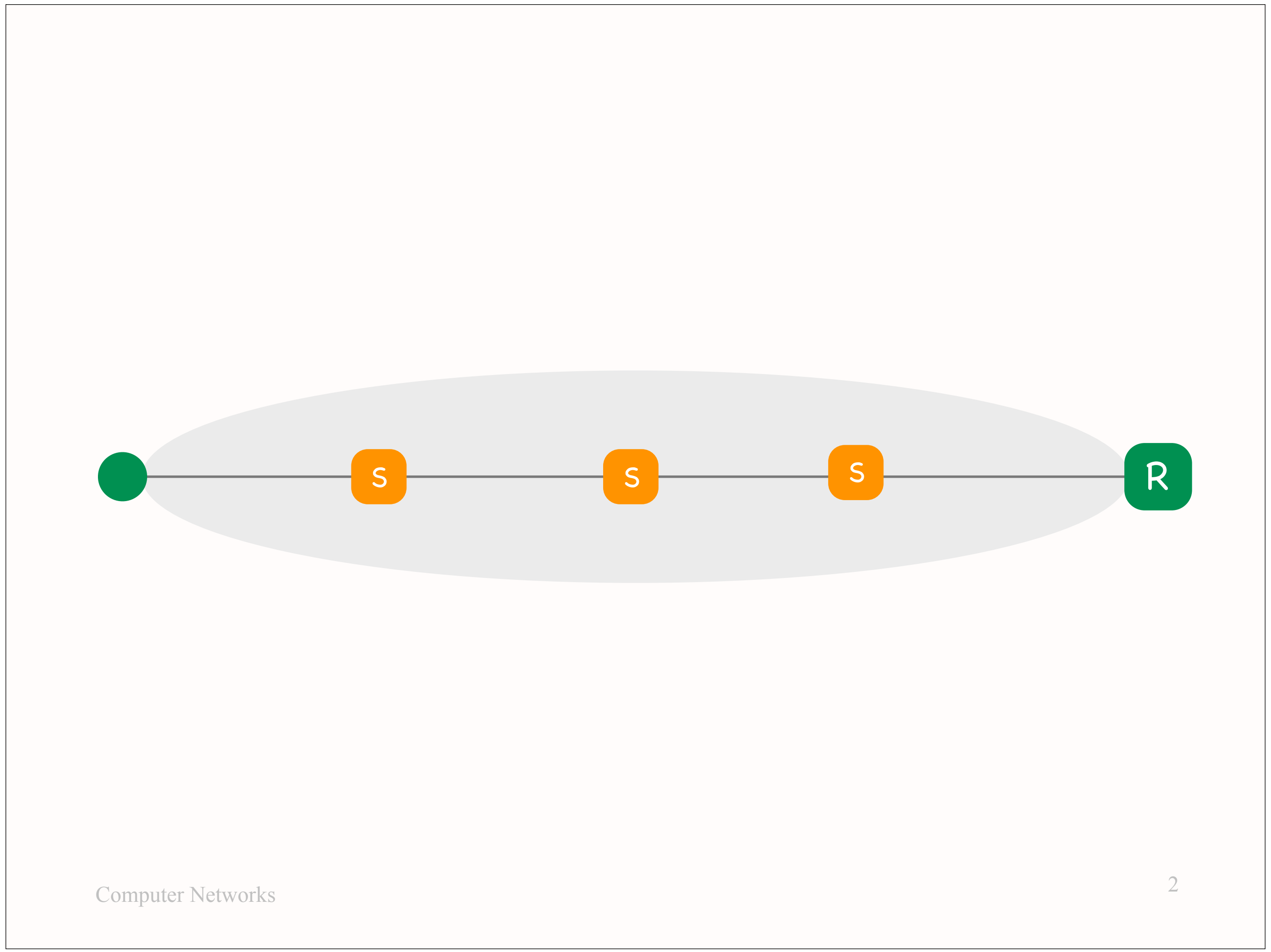
Three levels of hierarchy

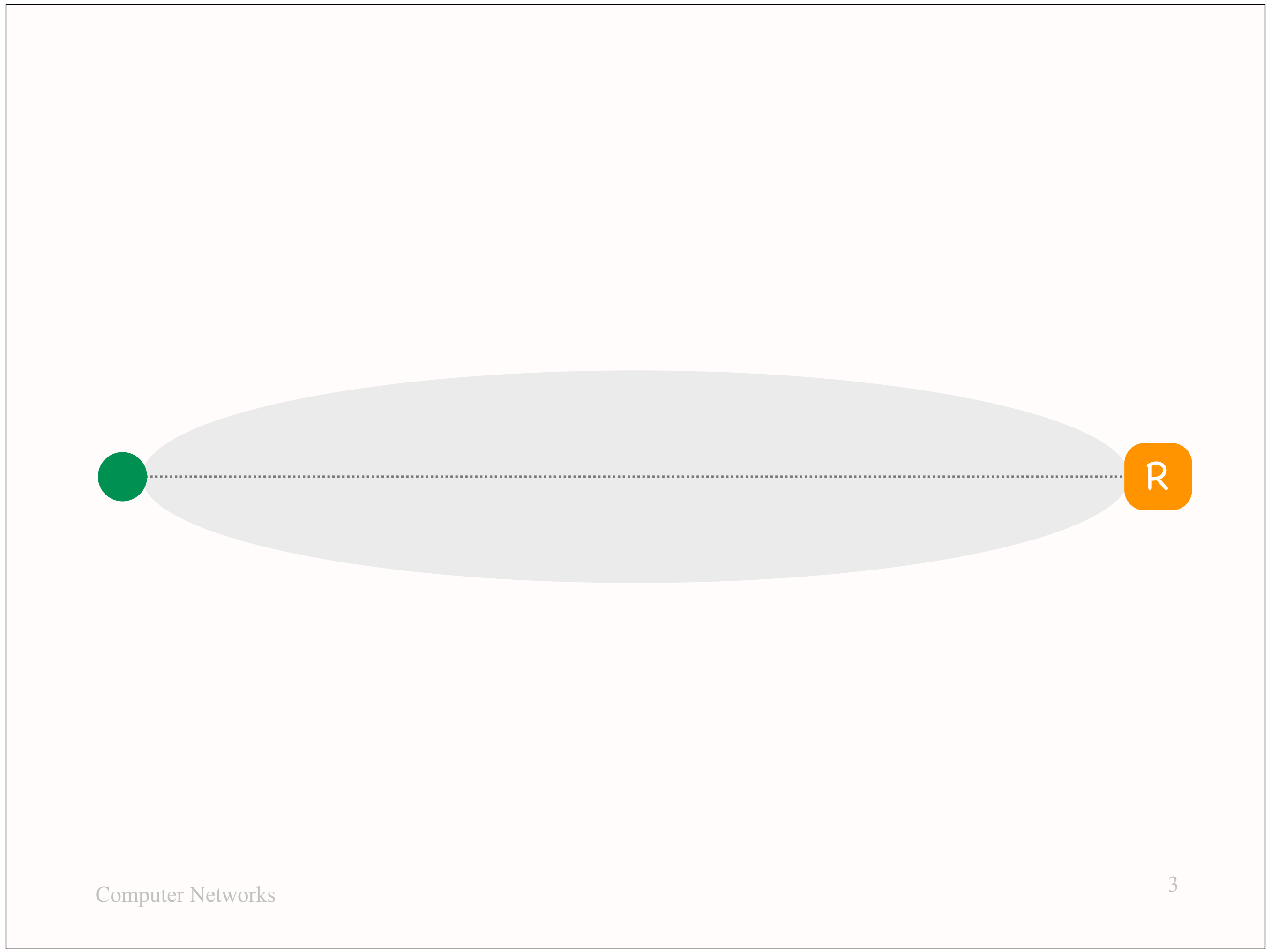
- IP subnet
  - L2 forwarding
  - L2 learning
- Autonomous System (AS)
  - IP (L3) forwarding
  - intra-domain routing
- Internet
  - IP (L3) forwarding
  - inter-domain routing (BGP)



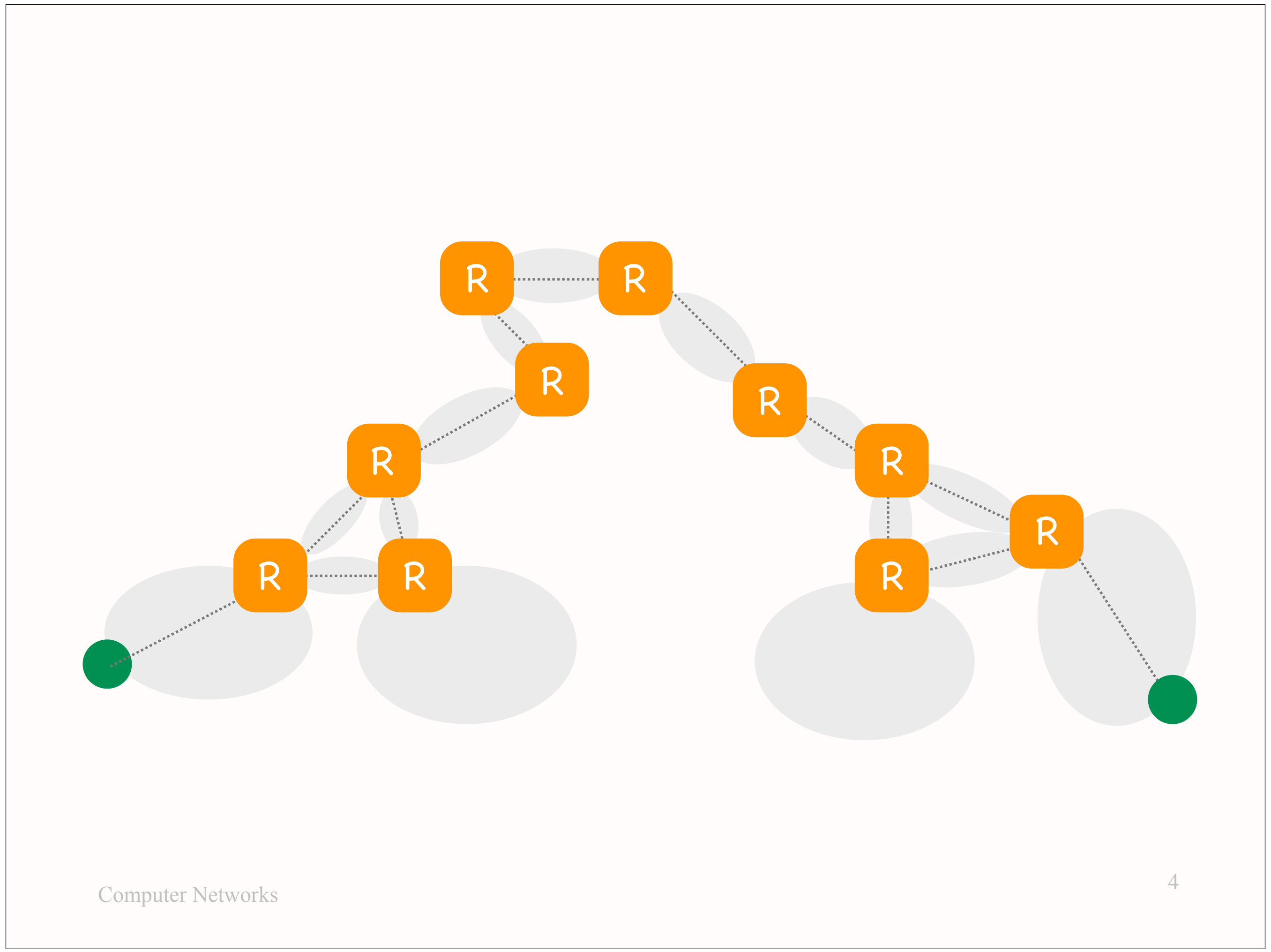
This is an IP subnet, which consists of:

- end-systems and routers (network-layer switches) at its edges;
- (link-layer) switches in the middle.

An IP subnet is a network, and it has its own network-layer protocol.
There exist different types of IP subnets, with different network-layer protocols.
The most popular one is Ethernet,
which we discussed in the last lecture.



From the point of view of the Internet, an IP subnet is a set of “links” from end-systems to one or more routers.



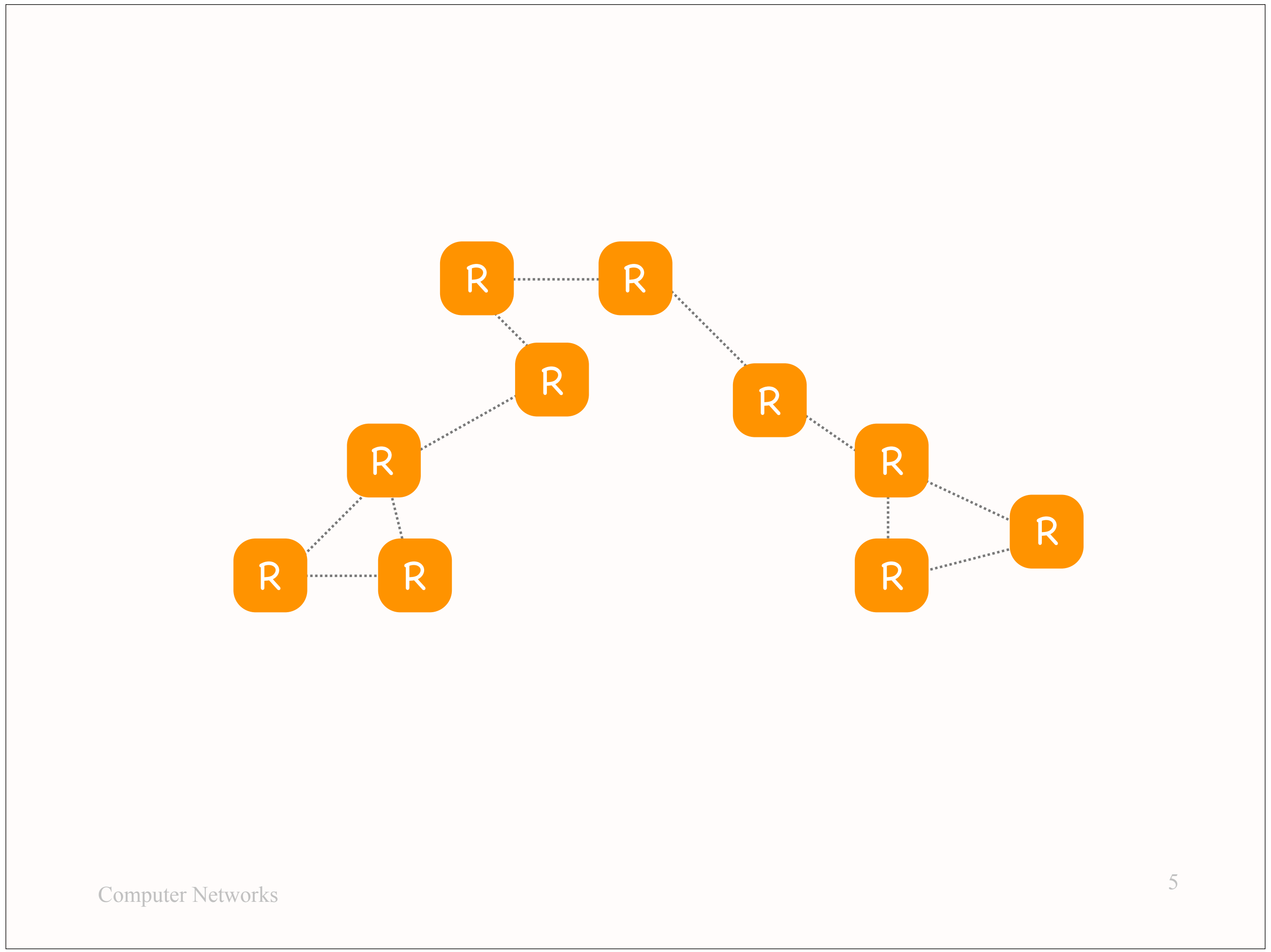
This is the Internet, which consists of:
- end-systems at its edges;
- routers in the middle.

The Internet is a network (of IP subnets),
and its has its own network-layer protocol, which is IP.

So, this, that I just described, is a two-level hierarchy:
- at the lower level we have IP subnets
- and at the higher level the Internet.

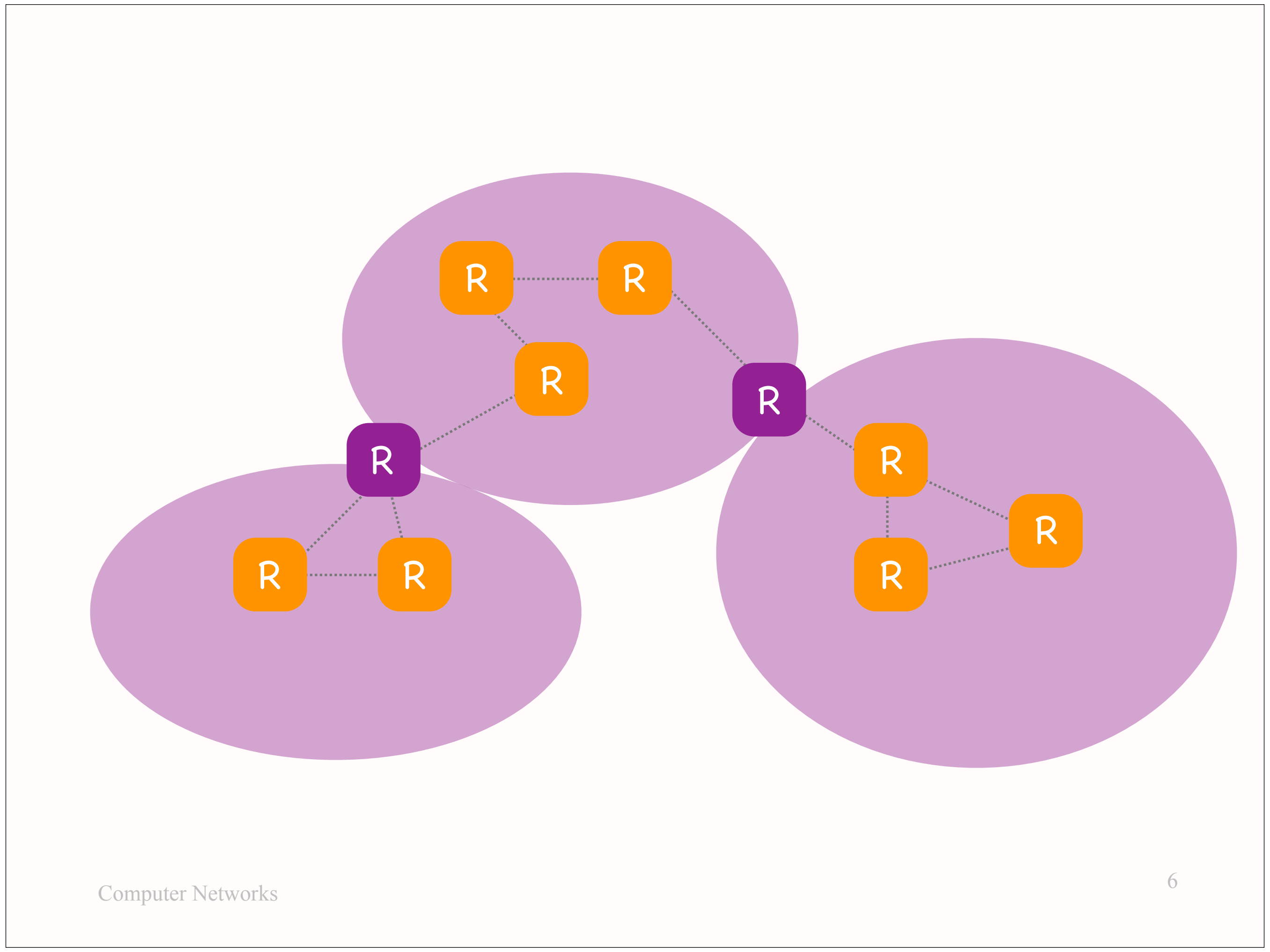
This hierarchy is reflected in packet headers: each Internet packet carries a link-layer header, and a network-layer (IP) header.

Moreover, this hierarchy is related to forwarding: (link-layer) switches forward differently than routers (network-layer switches).



And then there exists another hierarchy:

Forget about IP subnets for a moment, and consider only the routers.

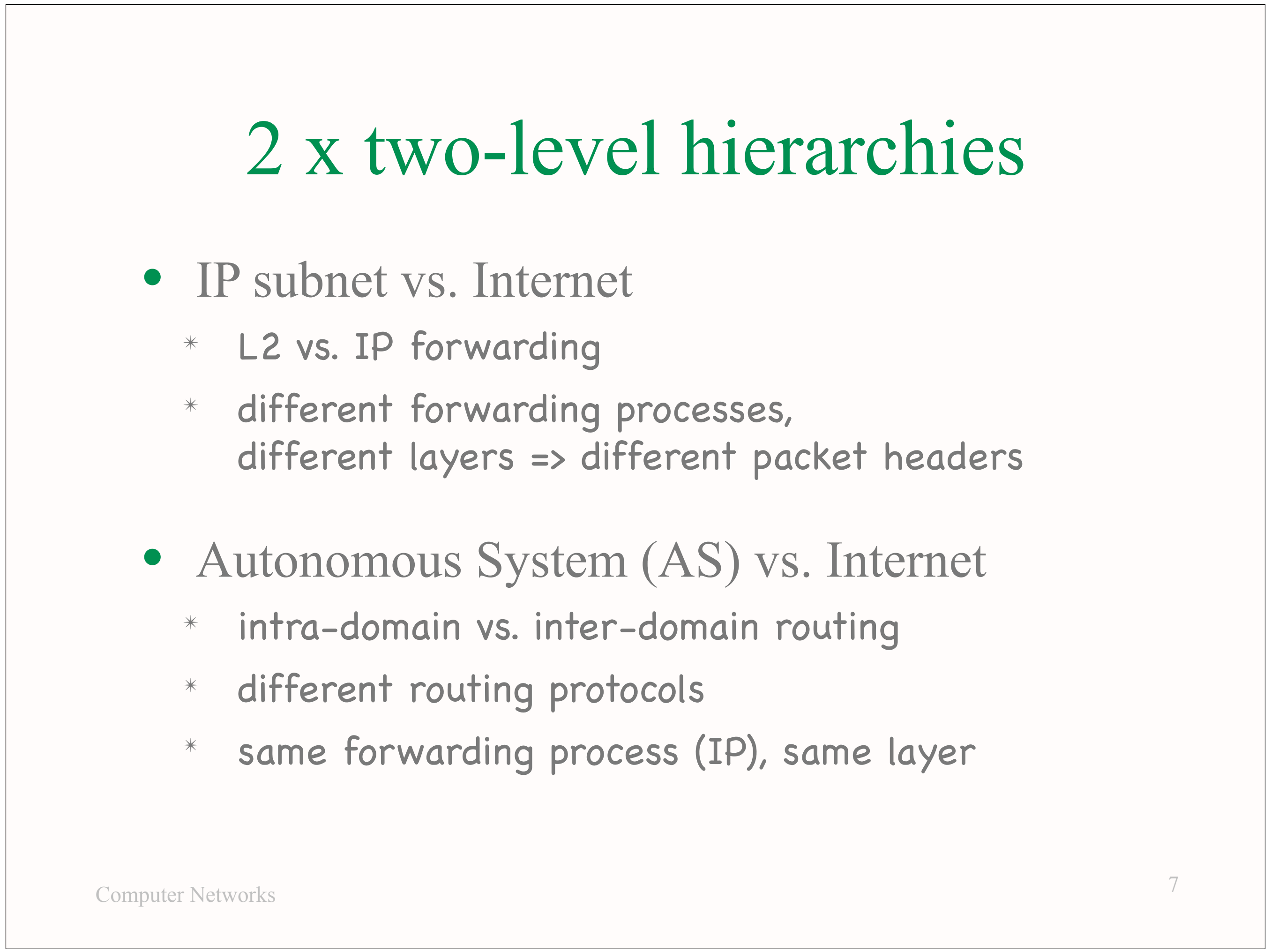


Not all routers are “the same”:
The Internet is organized into Autonomous Systems (ASes).
All the routers of each AS must run the same intra-AS routing protocol.
Moreover, the “border routers,” which are located at AS boundaries, must also run the inter-AS routing protocol of the Internet, which is the Border Gateway Protocol (BGP).

This is also a two-level hierarchy:
- at the lower level we have ASes;
- at the higher level we have the Internet.

This hierarchy, however, is not reflected in packet headers, and is unrelated to the forwarding process (a router forwards in the same way, whether it’s a border router or not).

This hierarchy is related to routing: routers within an AS advertize routes (paths) to the AS’s IP subnets, whereas border routers advertize routes (paths) to foreign ASes.



2 x two-level hierarchies

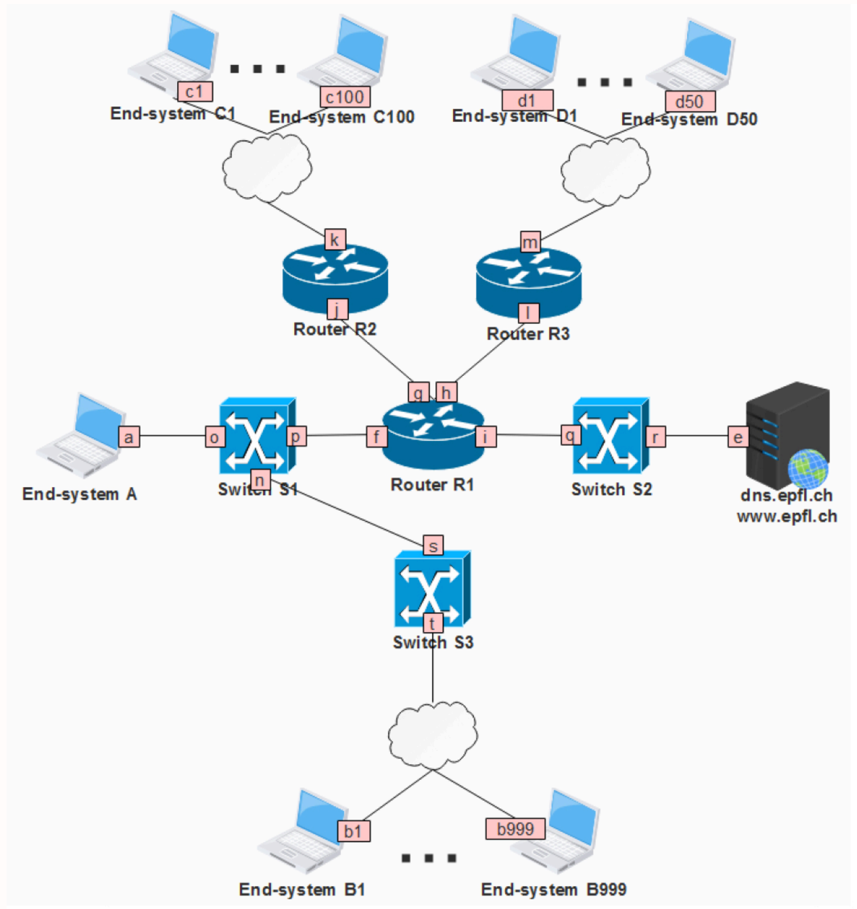
- IP subnet vs. Internet
  - L2 vs. IP forwarding
  - different forwarding processes, different layers => different packet headers
- Autonomous System (AS) vs. Internet
  - intra-domain vs. inter-domain routing
  - different routing protocols
  - same forwarding process (IP), same layer

So, it is more accurate to say that the Internet is organized as two different two-level hierarchies.

Question: Allocate IP addresses

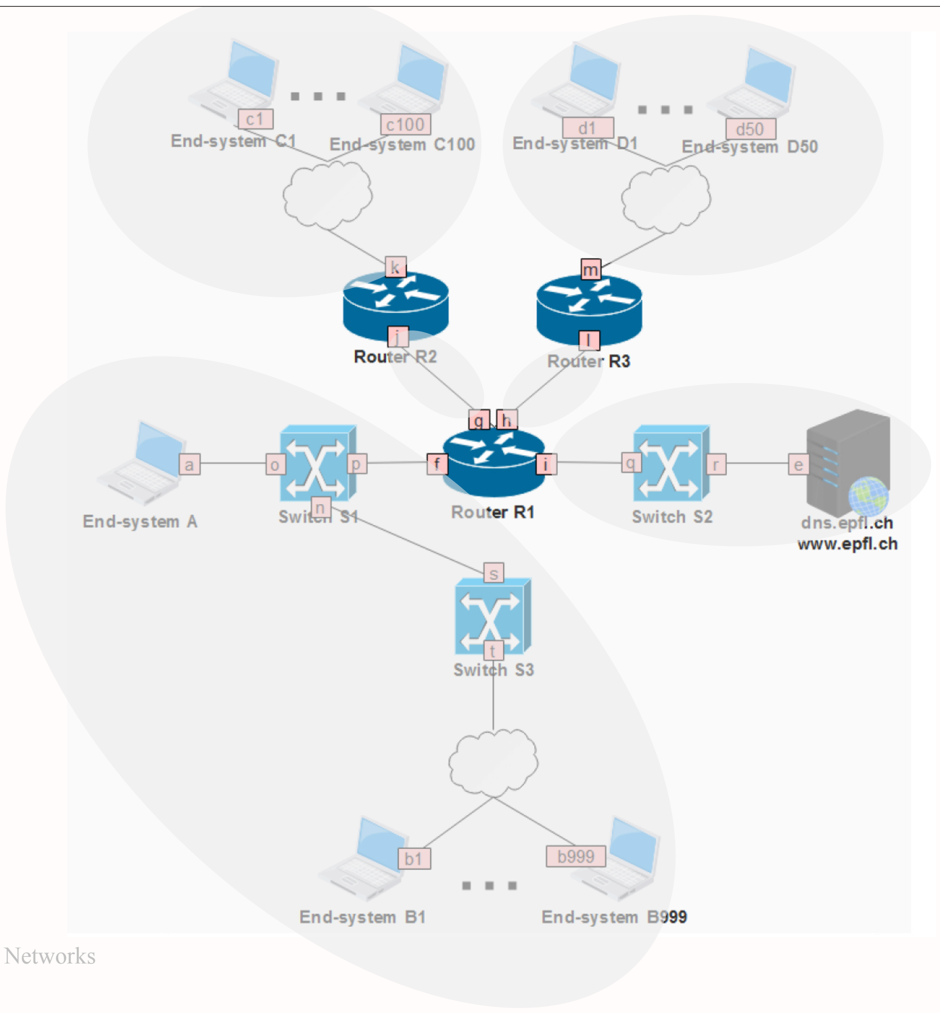
- Given network topology and IP prefix, allocate IP addresses using smallest possible range per IP subnet
- Final 2018, Problem 2, Question 1

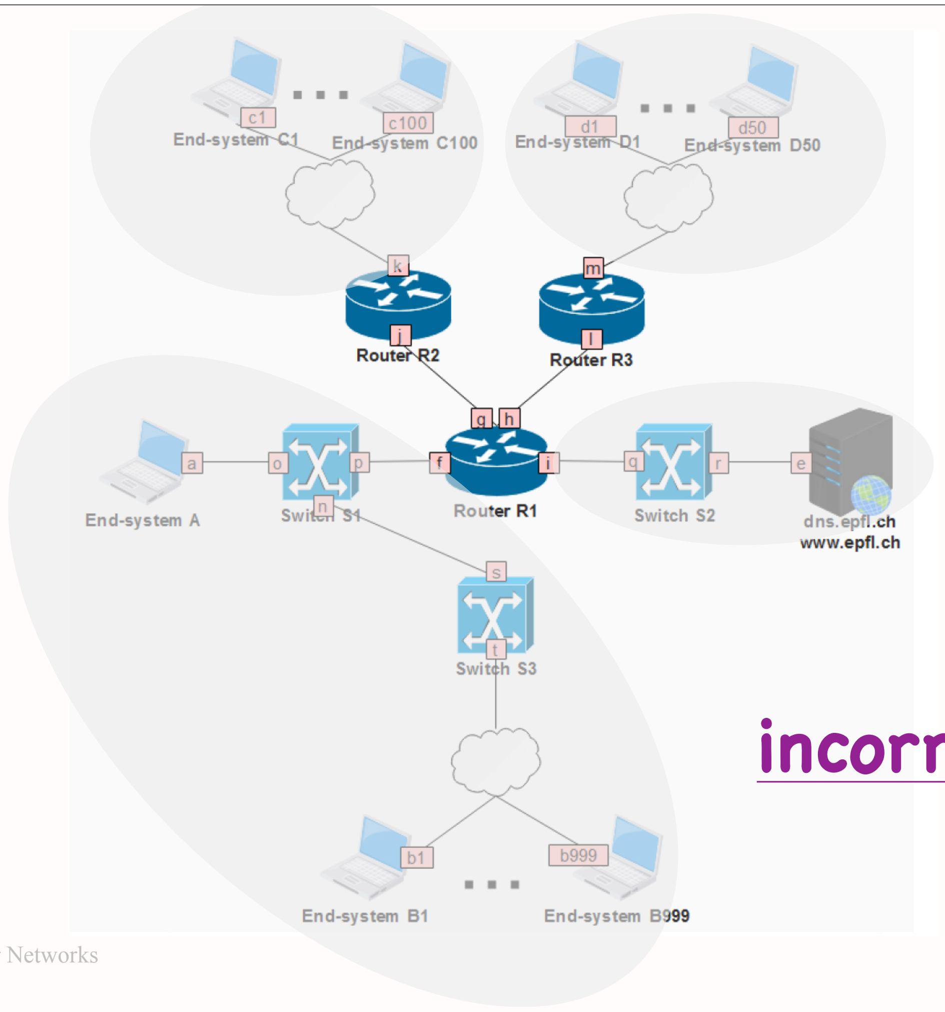
Allocate IP addresses from: 100.0.0.0/16



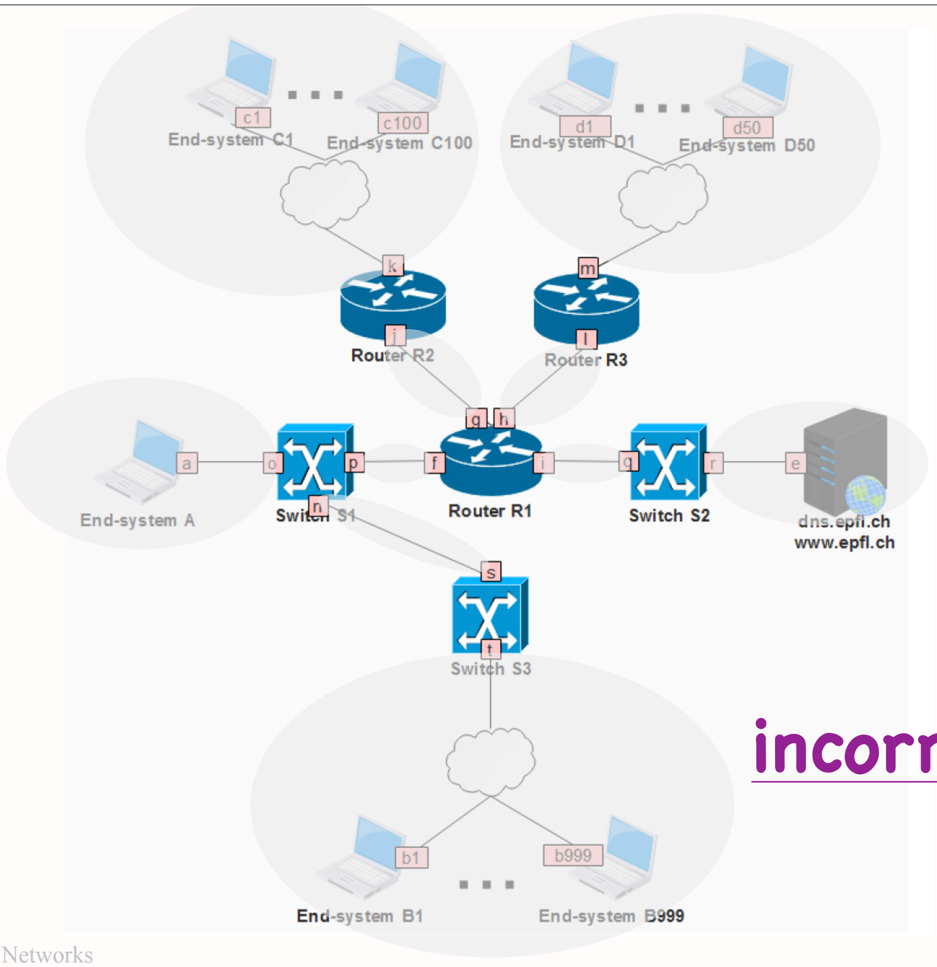
Step 1: Draw the IP subnets

- IP subnet = contiguous network area that has routers only at its boundaries
- Each interface of an IP router belongs to a different IP subnet

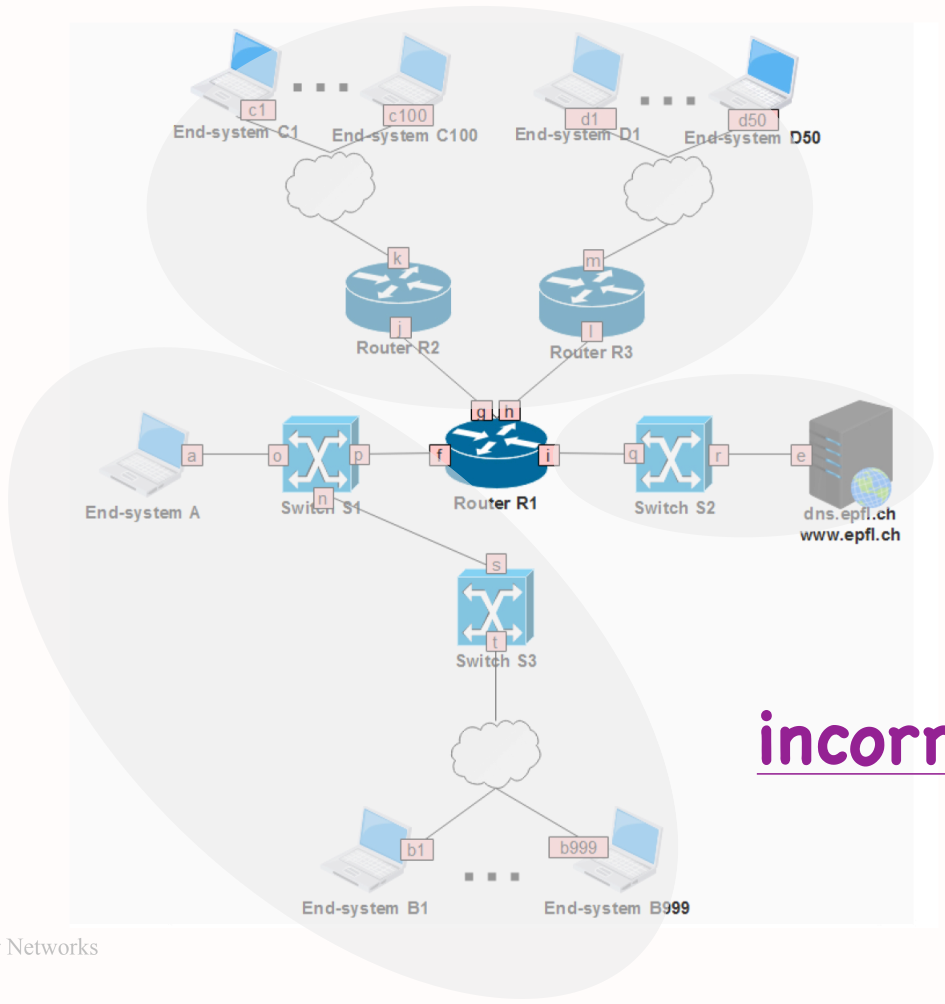

c1
End-system C1
c100
End-system C100
d1
End-system D1
d50
End-system D50
k
Router R2
j
m
Router R3
l
g
h
a
End-system A
o
Switch S1
p
n
f
Router R1
i
q
Switch S2
r
e
dns.epfl.ch
www.epfl.ch
s
Switch S3
t
b1
End-system B1
b999
End-system B999


c1
End-system C1
c100
End-system C100
d1
End-system D1
d50
End-system D50
k
m
Router R2
j
Router R3
l
g
h
a
o
p
f
i
q
r
e
End-system A
Switch S1
n
Router R1
Switch S2
dns.epfl.ch
www.epfl.ch
s
Switch S3
t
b1
b999
End-system B1
End-system B999

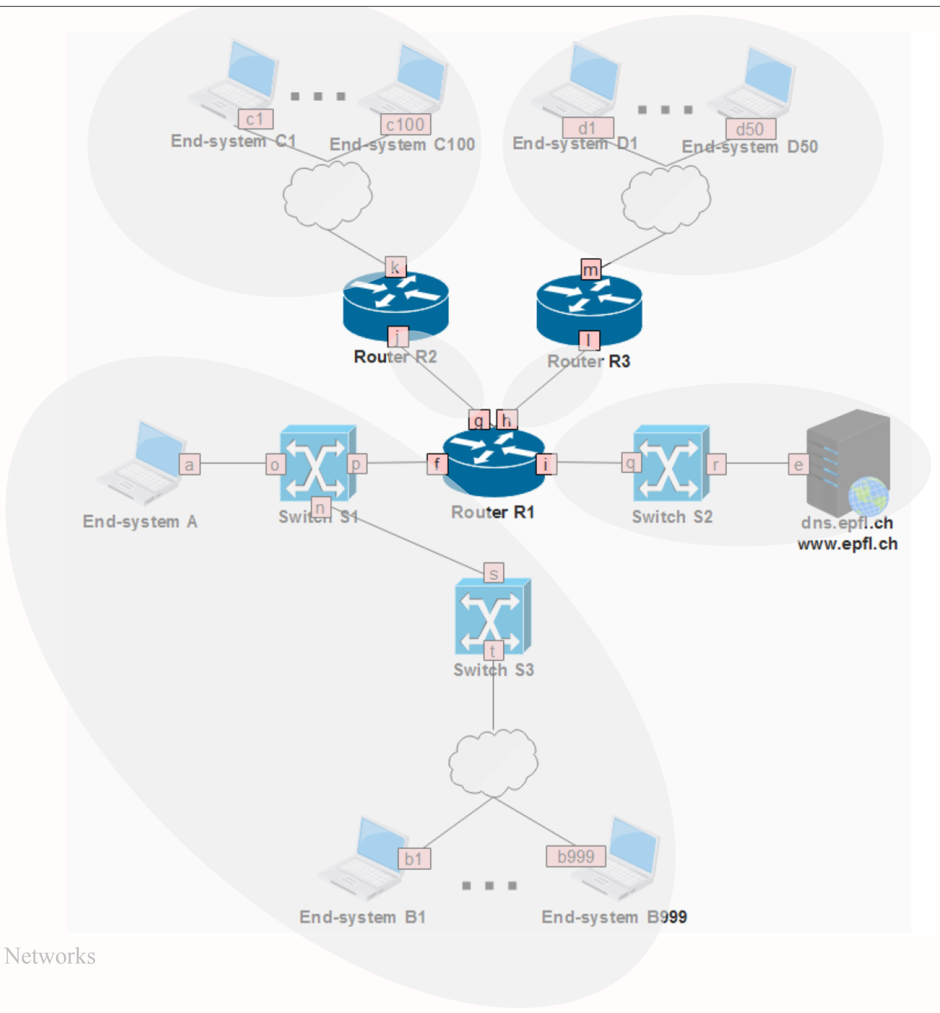
incorrect


c1
c100
End-system C1
End-system C100
d1
d50
End-system D1
End-system D50
k
m
Router R2
Router R3
j
l
g
h
a
o
p
f
i
q
r
e
End-system A
Switch S1
n
Router R1
Switch S2
dns.epfl.ch
www.epfl.ch
s
Switch S3
t
b1
b999
End-system B1
End-system B999

incorrect


c1
c100
d1
d50
End-system C1
End-system C100
End-system D1
End-system D50
k
m
Router R2
Router R3
j
l
g
h
a
o
p
f
i
q
r
e
End-system A
Switch S1
n
Router R1
Switch S2
dns.epfl.ch
www.epfl.ch
s
Switch S3
t
b1
b999
End-system B1
End-system B999

incorrect

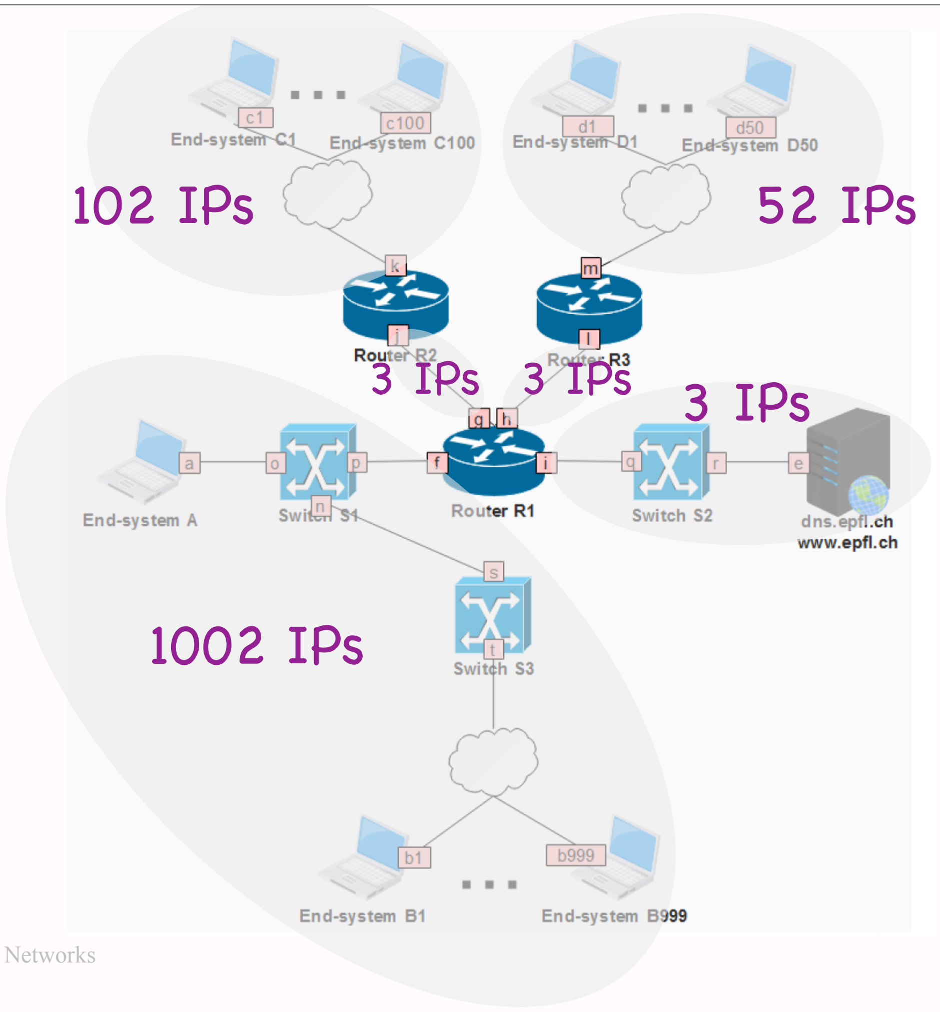

c1
c100
End-system C1
End-system C100
d1
d50
End-system D1
End-system D50
k
m
Router R2
j
l
Router R3
g
h
a
o
p
f
i
q
r
e
End-system A
Switch S1
n
Router R1
Switch S2
dns.epfl.ch
www.epfl.ch
s
Switch S3
t
b1
b999
End-system B1
End-system B999

Step 2: Count IPs per subnet

- One IP address per end-system interface
- One IP address per router interface
  - not needed for IP forwarding,
    but needed for other practical reasons
- No IP addresses for link-layer switches
  - in reality they have IP addresses,
    but ignore to simplify exam

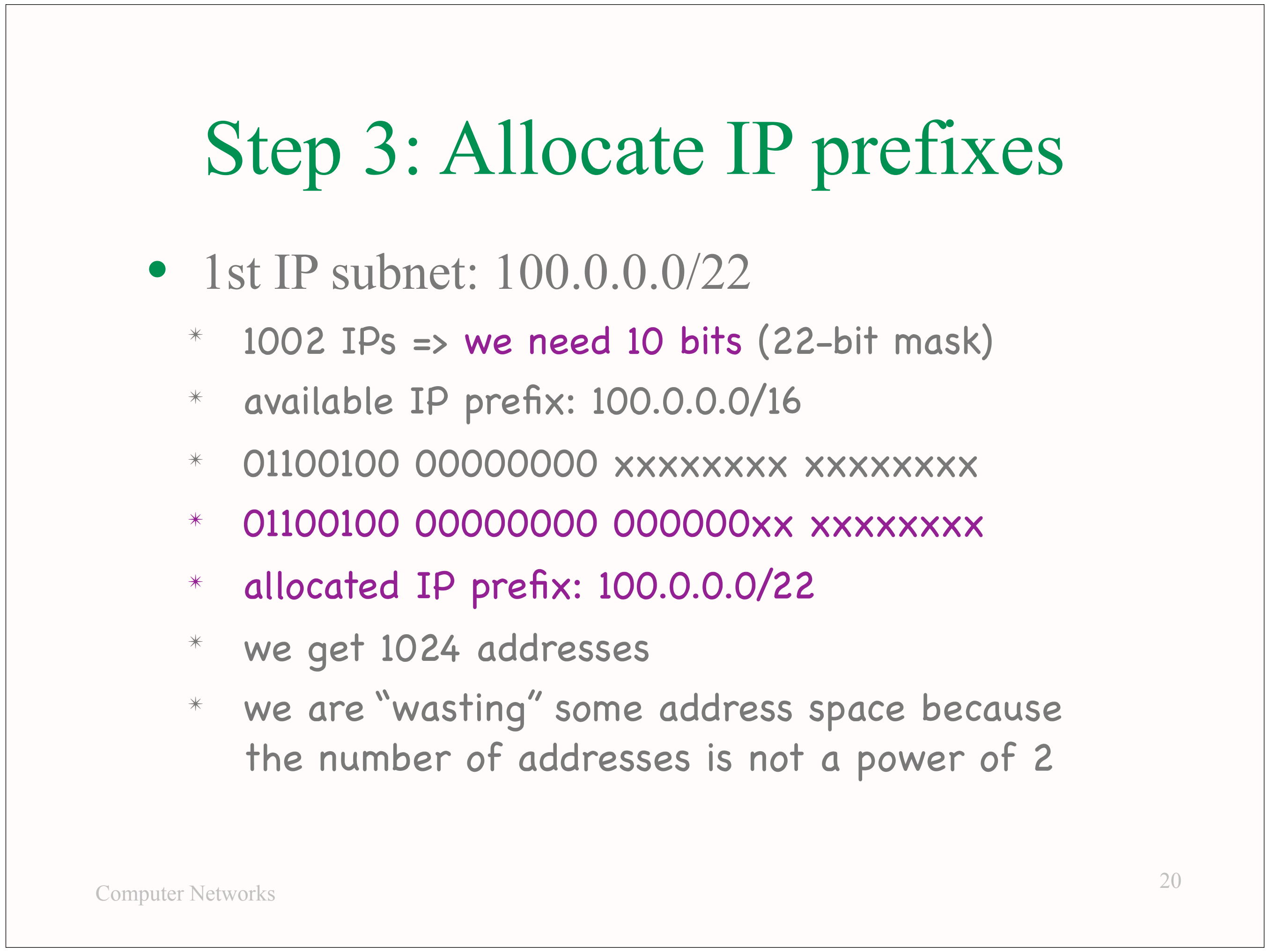
Step 2: Count IPs per subnet

- One broadcast IP address
  - the very last IP address covered by the IP prefix
  - addresses all entities with an IP address in the local subnet
- No network IP address
  - the very first IP address covered by the IP prefix
  - meant to have special meaning, but not typically used


102 IPs
52 IPs
3 IPs
3 IPs
3 IPs
1002 IPs
c1
c100
End-system C1
End-system C100
d1
d50
End-system D1
End-system D50
k
m
Router R2
Router R3
j
l
g
h
f
i
Router R1
a
o
p
n
End-system A
Switch S1
q
r
e
Switch S2
dns.epfl.ch
www.epfl.ch
s
t
Switch S3
b1
b999
End-system B1
End-system B999

Step 3: Allocate IP prefixes

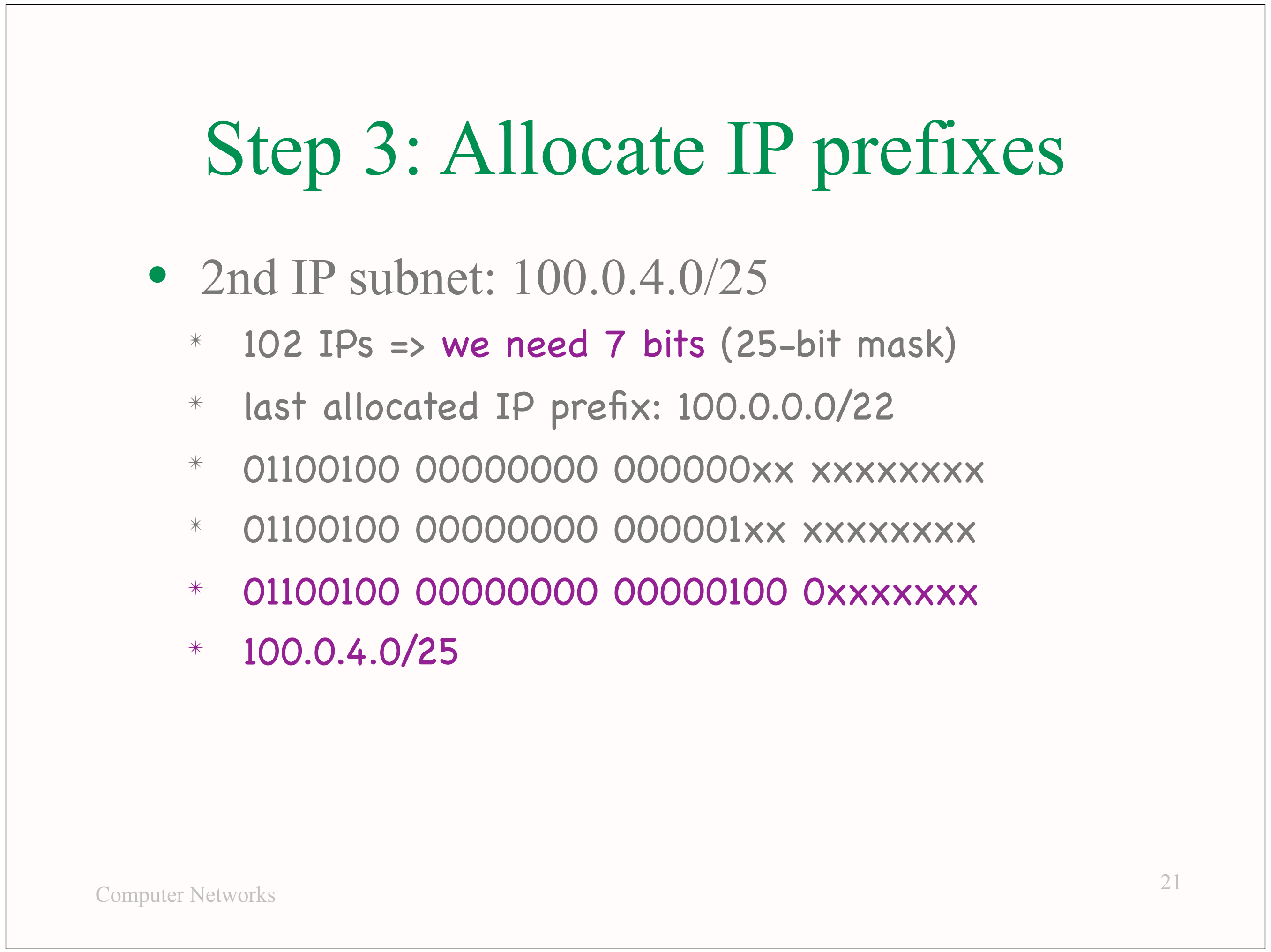
- One approach: start from the largest IP subnet, allocate consecutive prefixes
- Whatever approach you choose:
  IP prefixes allocated to different IP subnets must not overlap



Step 3: Allocate IP prefixes

- 1st IP subnet: 100.0.0.0/22
  - 1002 IPs => we need 10 bits (22-bit mask)
  - available IP prefix: 100.0.0.0/16
  - 01100100 00000000 xxxxxxxx xxxxxxxx
  - 01100100 00000000 000000xx xxxxxxxx
  - allocated IP prefix: 100.0.0.0/22
  - we get 1024 addresses
  - we are "wasting" some address space because the number of addresses is not a power of 2

In this approach, to allocate a prefix to the 1st IP subnet:
- You write down the available prefix: 100.0.0.0/16
- You convert it to binary format: 01100100 00000000 xxxxxxxx xxxxxxxx
- Since you need 10 unallocated bits (to allocate 1002 IPs), you keep the last 10 x's and convert the remaining x's to 0's: 01100100 00000000 000000xx xxxxxxxx
- You convert the result to the usual prefix notation: 100.0.0.0/22

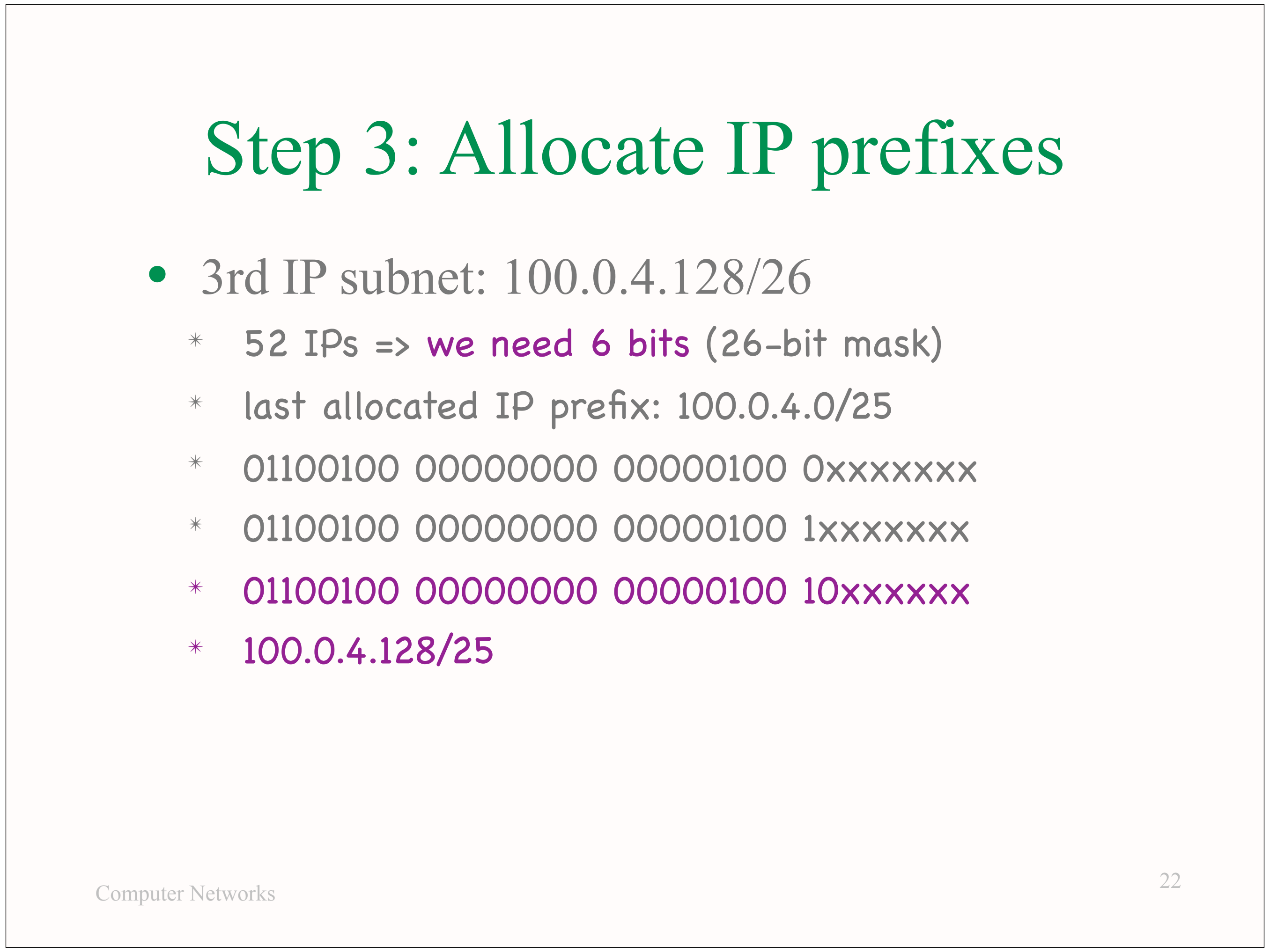


Step 3: Allocate IP prefixes

- 2nd IP subnet: 100.0.4.0/25
  - 102 IPs => we need 7 bits (25-bit mask)
  - last allocated IP prefix: 100.0.0.0/22
  - 01100100 00000000 000000xx xxxxxxxx
  - 01100100 00000000 000001xx xxxxxxxx
  - 01100100 00000000 00000100 0xxxxxxx
  - 100.0.4.0/25

To allocate a prefix to the 2nd IP subnet:

- You write down the last allocated prefix: 100.0.4.0/25
- You convert it to binary: 01100100 00000000 000000xx xxxxxxxx
- You need a prefix that does not overlap with this one. Hence, you need to change at least one of the digits of this prefix. The simplest approach is to change the least significant allocated digit: 01100100 00000000 000001xx xxxxxxxx
- This is indeed a prefix that does not overlap with the previous one, but contains more unallocated bits than needed. Since you need only 7 unallocated bits, you keep the last 7 x's and convert the remaining x's to 0's: 01100100 00000000 00000100 0xxxxxxx.



Step 3: Allocate IP prefixes

- 3rd IP subnet: 100.0.4.128/26
  - 52 IPs => we need 6 bits (26-bit mask)
  - last allocated IP prefix: 100.0.4.0/25
  - 01100100 00000000 00000100 0xxxxxxx
  - 01100100 00000000 00000100 1xxxxxxx
  - 01100100 00000000 00000100 10xxxxxx
  - 100.0.4.128/25

And so on…

Step 3: Allocate IP prefixes

- 4th IP subnet: 100.0.4.192/30
  - 3 IPs => we need 2 bits (30-bit mask)
  - last allocated IP prefix: 100.0.4.128/26
  - 01100100 00000000 00000100 10xxxxxx
  - 01100100 00000000 00000100 11xxxxxx
  - 01100100 00000000 00000100 110000xx
  - 100.0.4.192/30

Step 3: Allocate IP prefixes

- 5th IP subnet: 100.0.4.196/30
  - 3 IPs => we need 2 bits (30-bit mask)
  - last allocated IP prefix: 100.0.4.192/30
  - 01100100 00000000 00000100 110000xx
  - 01100100 00000000 00000100 110001xx
  - 100.0.4.196/30

Step 3: Allocate IP prefixes

- 6th IP subnet: 100.0.4.200/30
  - 3 IPs => we need 2 bits (30-bit mask)
  - last allocated IP prefix: 100.0.4.196/30
  - 01100100 00000000 00000100 110001xx
  - 01100100 00000000 00000100 110010xx
  - 100.0.4.200/30

Step 4: Allocate IP addresses

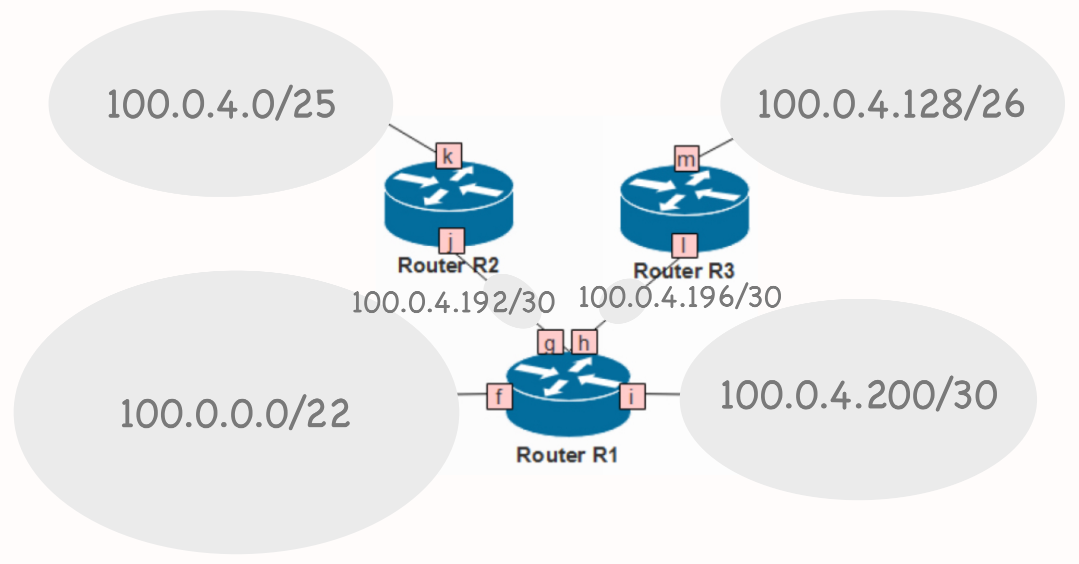
- 1st IP subnet: 1002 addresses from 100.0.0.0/22
  - 01100100 00000000 000000xx xxxxxxxx
  - 01100100 00000000 00000011 11111111
  - broadcast IP address: 100.0.3.255
  - 100.0.0.0 — 100.0.3.232

Step 4: Allocate IP addresses

- 2nd IP subnet: 102 addresses from 100.0.4.0/25
  - 01100100 00000000 00000100 0xxxxxxx
  - 01100100 00000000 00000100 01111111
  - broadcast IP address: 100.0.4.127
  - 100.0.4.0 — 100.0.4.100
- ...

Question: Show router tables

- Given network topology and allocated IPs, show router forwarding tables, assuming least-cost path routing protocol that has converged
- Final 2018, Problem 2, Question 2



100.0.0.0/22	f
100.0.4.0/25	g
100.0.4.128/26	h
100.0.4.192/30	g
100.0.4.196/30	h
100.0.4.200/30	i

Question: Show packets

- Given a communication scenario, show all the packets transmitted by end-systems and routers
- Final 2018, Problem 2, Question 3

DNS request from A to server

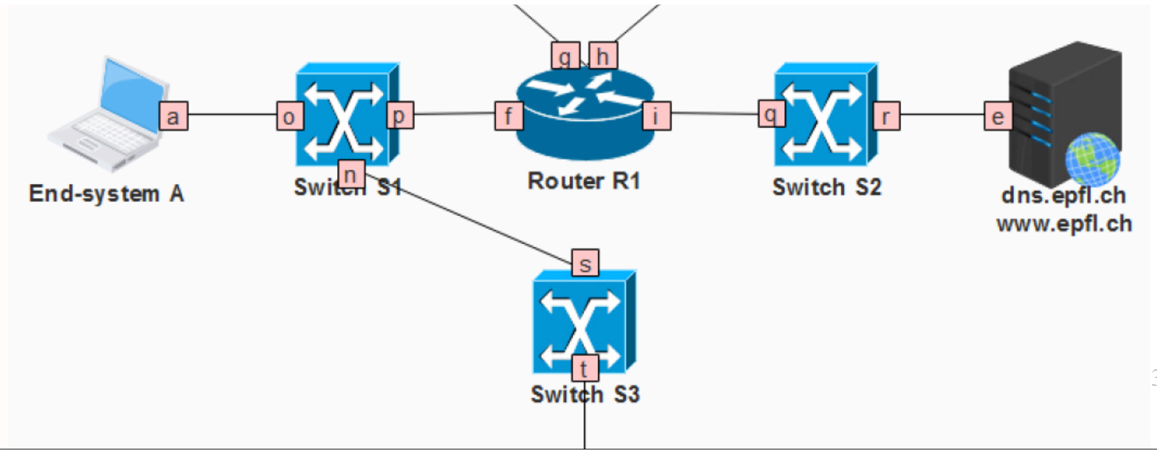
DNS response from server to A

HTTP GET request for base file from A to server

HTTP GET response from server to A

HTTP GET request for image file from A to server

HTTP GET response from server to A



1. Identify application-layer messages: always some combination of
   - DNS requests/responses and
   - HTTP GET requests/responses.

2. Consider each message and identify how many lower-layer packets this message will lead to.

DNS request from A to server

ARP request, MAC: A—broadcast

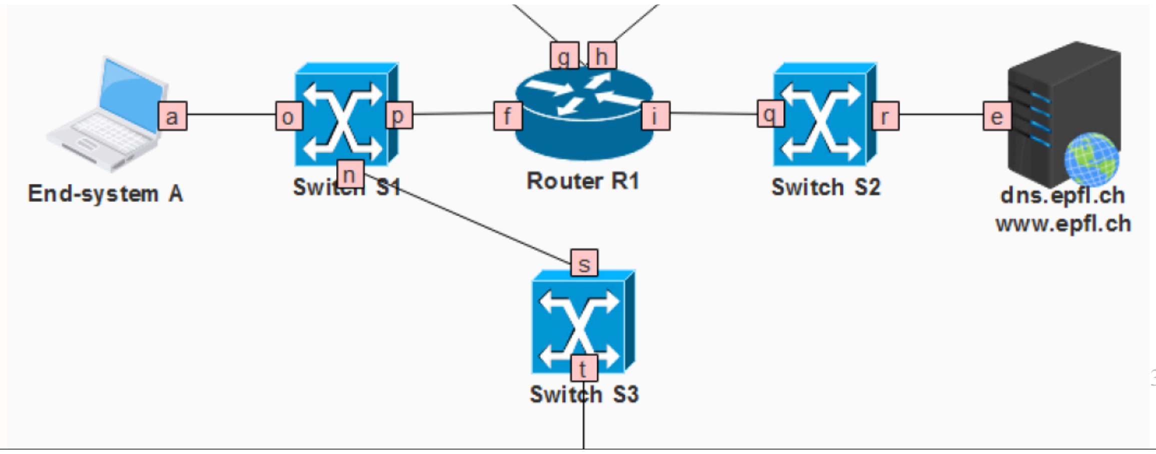
ARP response, MAC: R1—A

DNS request, MAC: A—R1, IP: A—DNS

ARP request, MAC: R1—broadcast

ARP response, MAC: DNS—R1

DNS request, MAC: R1—DNS, IP: A—DNS

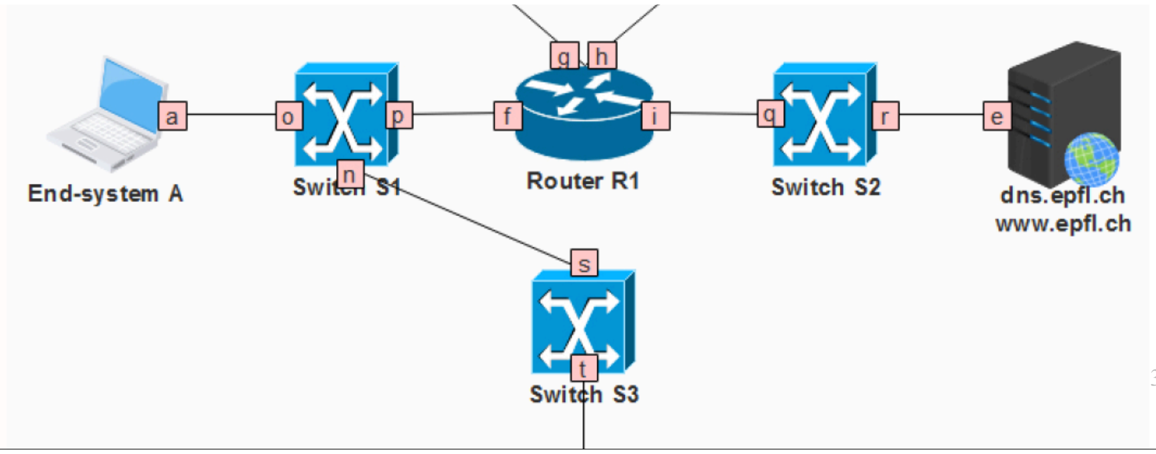


For each message ask:
- Does the sender know the destination MAC address?
  - If not, the sender broadcasts an ARP request.

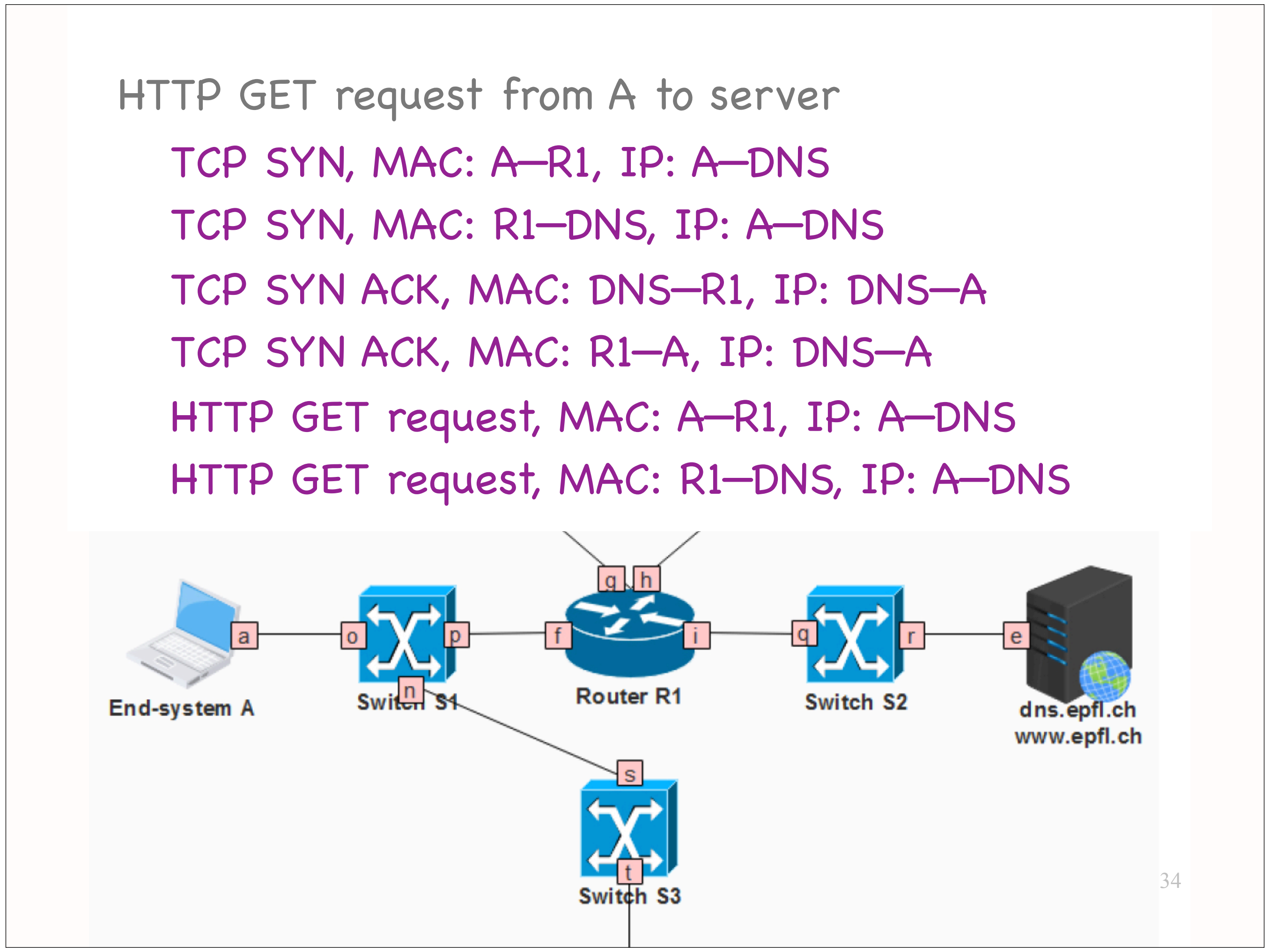
DNS response from server to A

DNS response, MAC: DNS–R1, IP: DNS–A

DNS response, MAC: R1–A, IP: DNS–A



For each message ask:
- Does the sender know the destination MAC address?
  - If not, the sender broadcasts an ARP request.
  - Could the sender have learned the destination MAC address from a previous ARP request that it received?
    - If yes, the sender does not broadcast an ARP request.



HTTP GET request from A to server

TCP SYN, MAC: A—R1, IP: A—DNS

TCP SYN, MAC: R1—DNS, IP: A—DNS

TCP SYN ACK, MAC: DNS—R1, IP: DNS—A

TCP SYN ACK, MAC: R1—A, IP: DNS—A

HTTP GET request, MAC: A—R1, IP: A—DNS

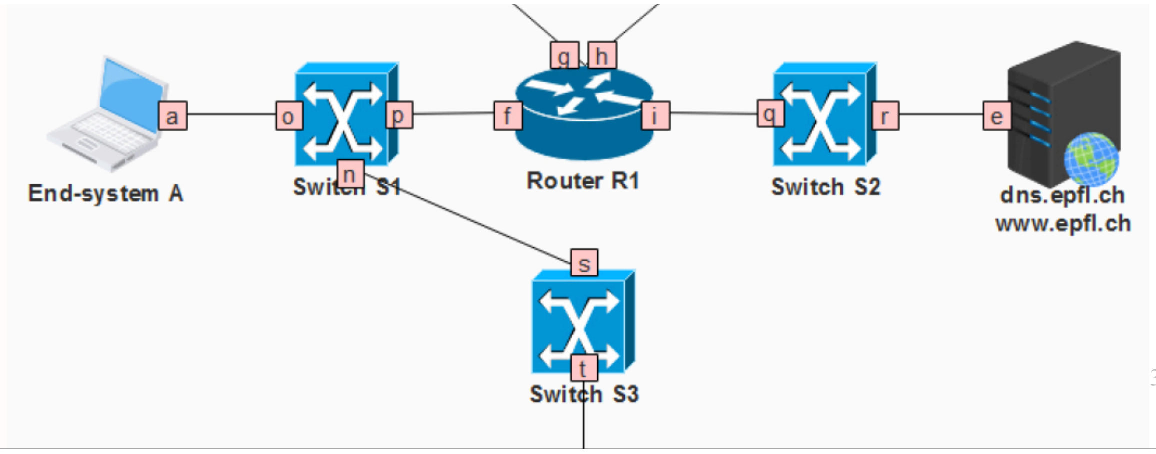
HTTP GET request, MAC: R1—DNS, IP: A—DNS

For each message ask:
- Does the sender know the destination MAC address?
  - If not, the sender broadcasts an ARP request.
  - Could the sender have learned the destination MAC address from a previous ARP request that it received?
    - If yes, the sender does not broadcast an ARP request.
  - Could the sender have learned the destination MAC address from a previous ARP response that it received?
    - If yes, the sender does not broadcast an ARP request.
- Does this message belong to a protocol that uses TCP or UDP?
  - If TCP, the sender must establish a TCP connection.

HTTP GET response from server to A

HTTP GET response, MAC: DNS—R1, IP: DNS—A

HTTP GET response, MAC: R1—A, IP: DNS—A

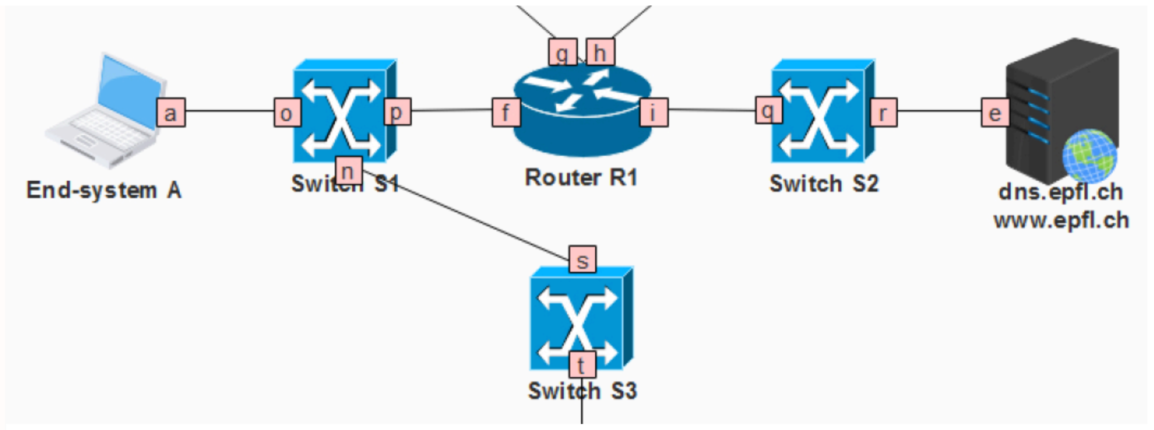


Question: Show switch tables

- Given a communication scenario, show switch forwarding tables
- Final 2018, Problem 2, Question 4

A's MAC	o
R1's MAC	p

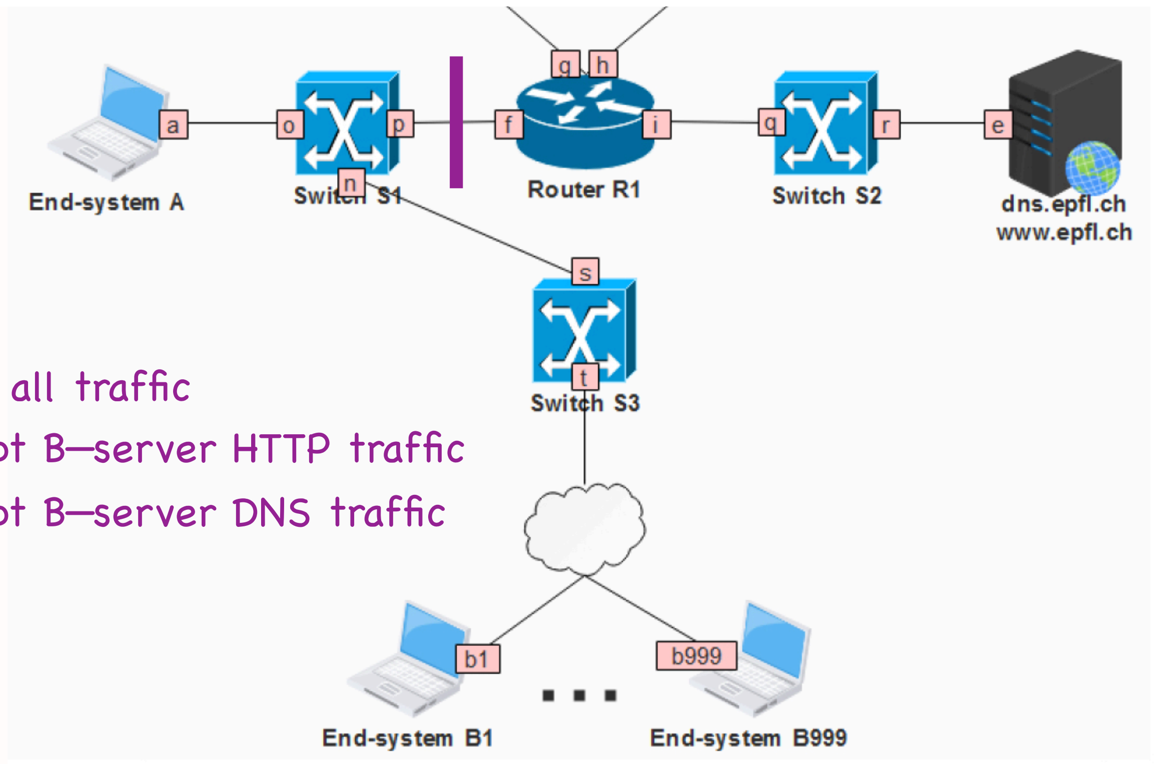
R1's MAC	q
DNS's MAC	r



A's MAC	s

Question: Show filtering table

- Show filtering table that allows a given communication pattern
- Final 2018, Problem 2, Question 5



Deny all traffic

Except B–server HTTP traffic

Except B–server DNS traffic

Question: Show filtering table

- List filtering table fields
  - action, TCP/UDP, src IP, dst IP, src port, dst port
- List entries that achieve given pattern
  - allow TCP B-prefix server-IP any 80
  - allow TCP server-IP B-prefix 80 any
  - allow UDP B-prefix server-IP any 53
  - allow UDP server-IP B-prefix 53 any
  - deny any any any any any any

TCP elements

- Connection setup and teardown
- Connection hijacking
- Connection setup (SYN) flooding
- Flow control
- Congestion control

TCP elements

- Connection setup and teardown
- Connection hijacking
- Connection setup (SYN) flooding
- Flow control
- Congestion control

Flow control

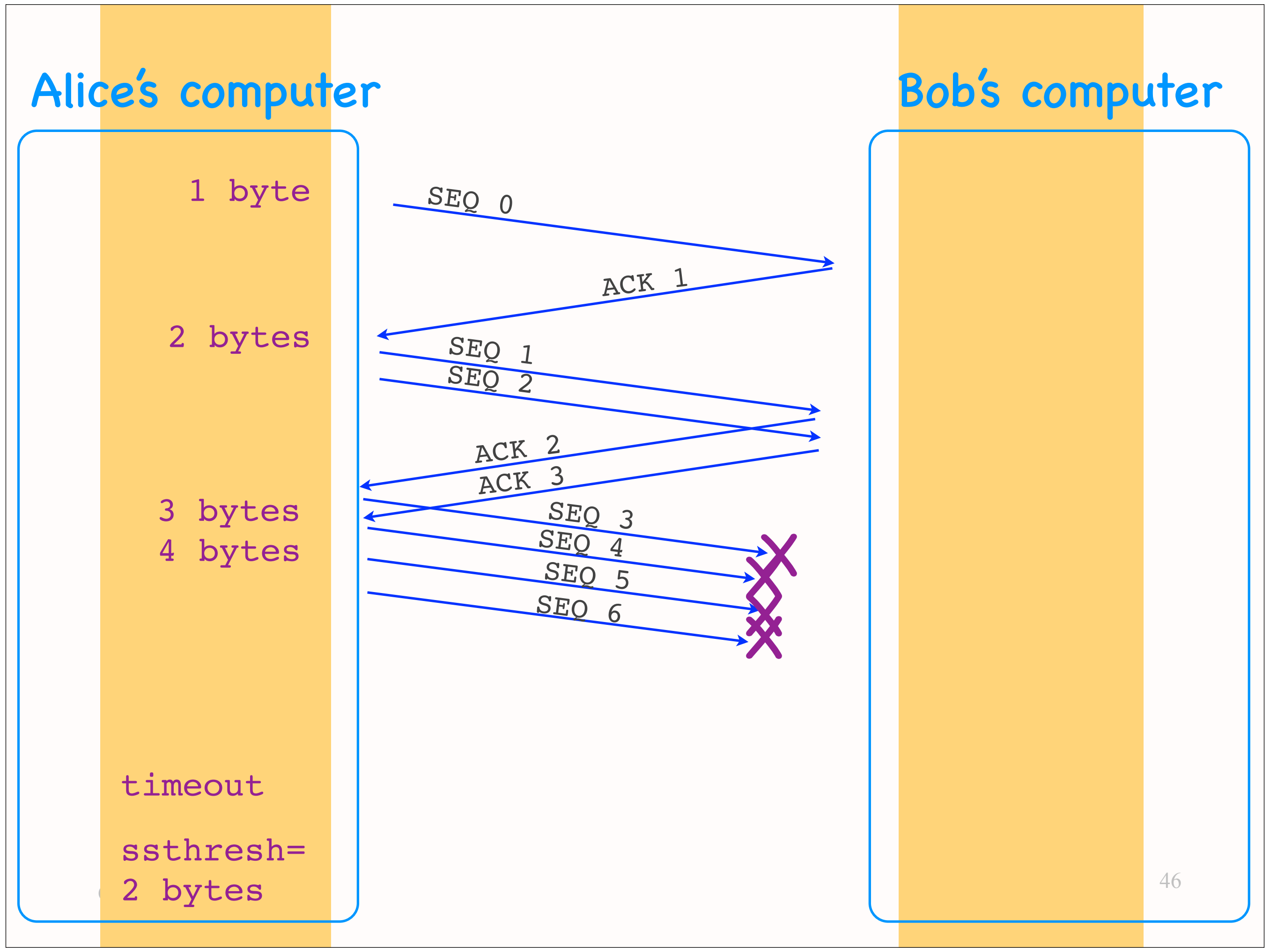
- Goal: not overwhelm the receiver
  - not send at a rate that the receiver cannot handle
- How: "receiver window"
  - spare room in receiver's rx buffer
  - receiver communicates it to sender as TCP header field

Congestion control

- Goal: not overwhelm the network
  - not send at a rate that the would create network congestion
- How: “congestion window”
  - number of unacknowledged bytes that the sender can transmit without creating congestion
  - sender estimates it on its own

Self-clocking

- Sender guesses the “right” congestion window based on the ACKs
- ACK = no congestion, increase window
- No ACK = congestion, decrease window



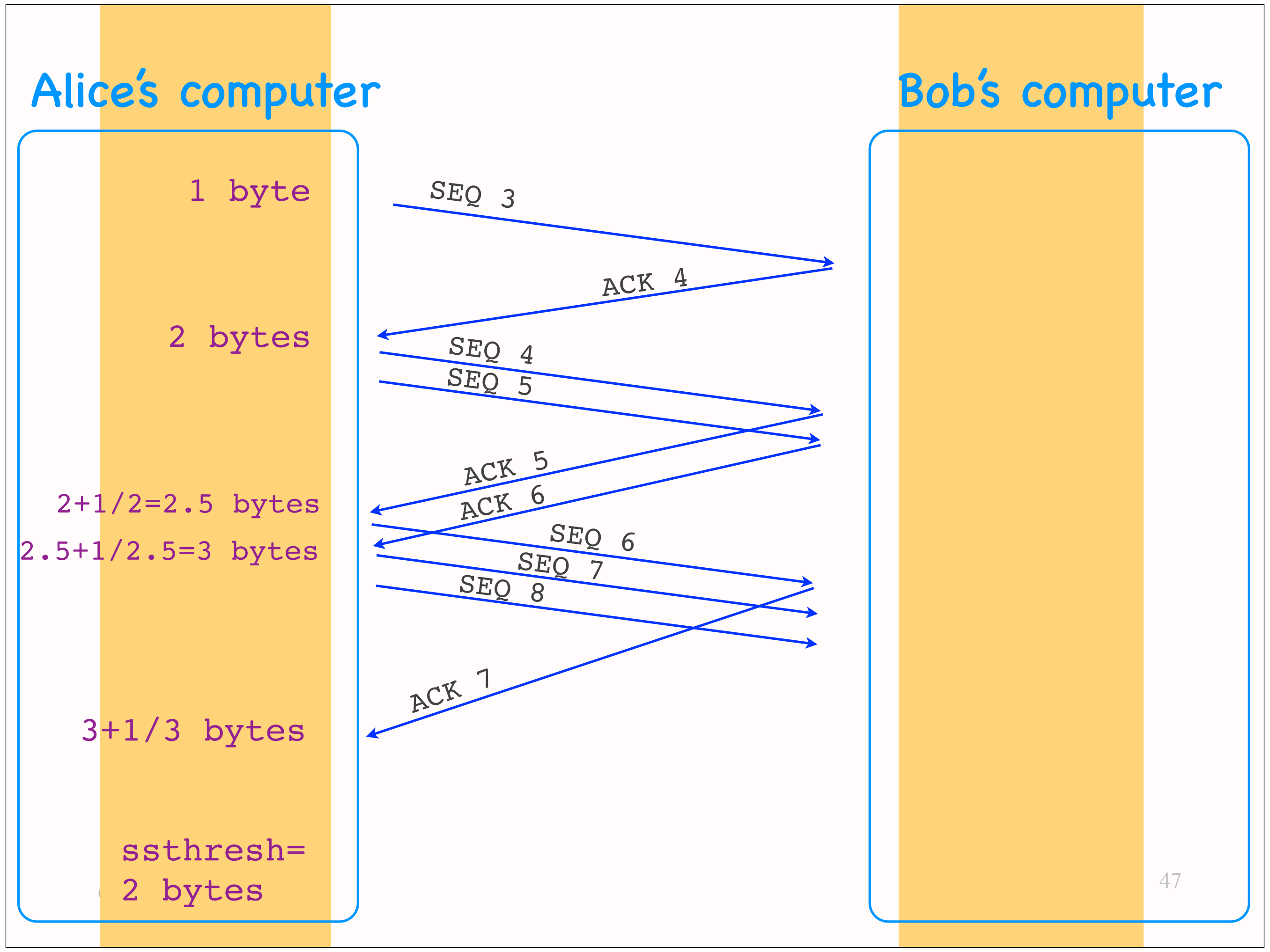
TCP congestion control example.
MSS=1 byte.

The TCP congestion control algorithm starts at the exponential increase state:

- Alice sets her congestion window size to 1 MSS (1 byte, in our example).
- Every time she gets an ACK, she increases her congestion-window size by 1 MSS.

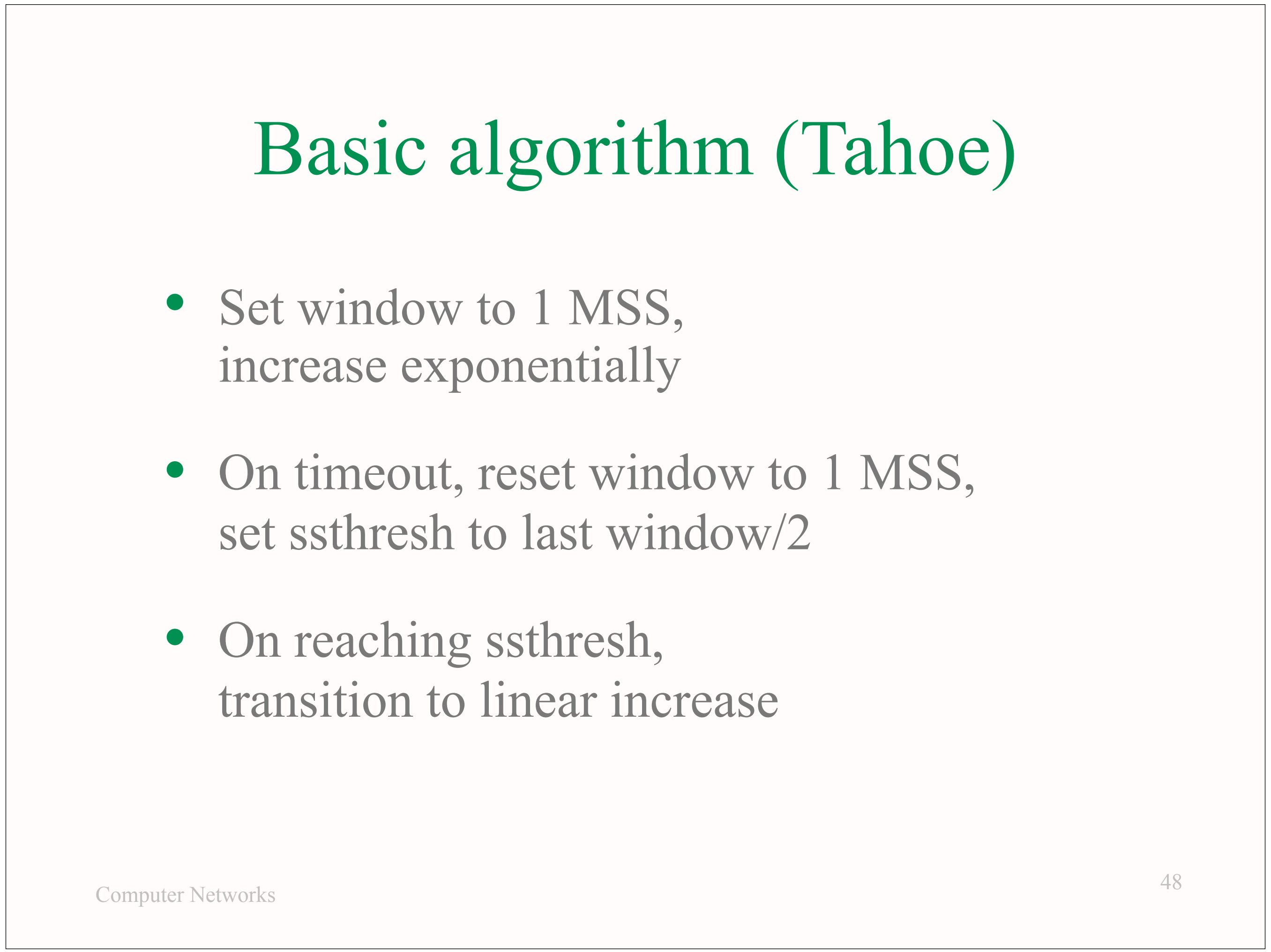
Suppose that, when the congestion-window size is 4, multiple packets are lost,
and Alice times out. At that point:

- She resets her congestion-window size to 1 MSS (1 byte).
- She also sets a threshold at half the value of what the congestion window size was when the timeout occurred (2 bytes).



Here's what happens next:

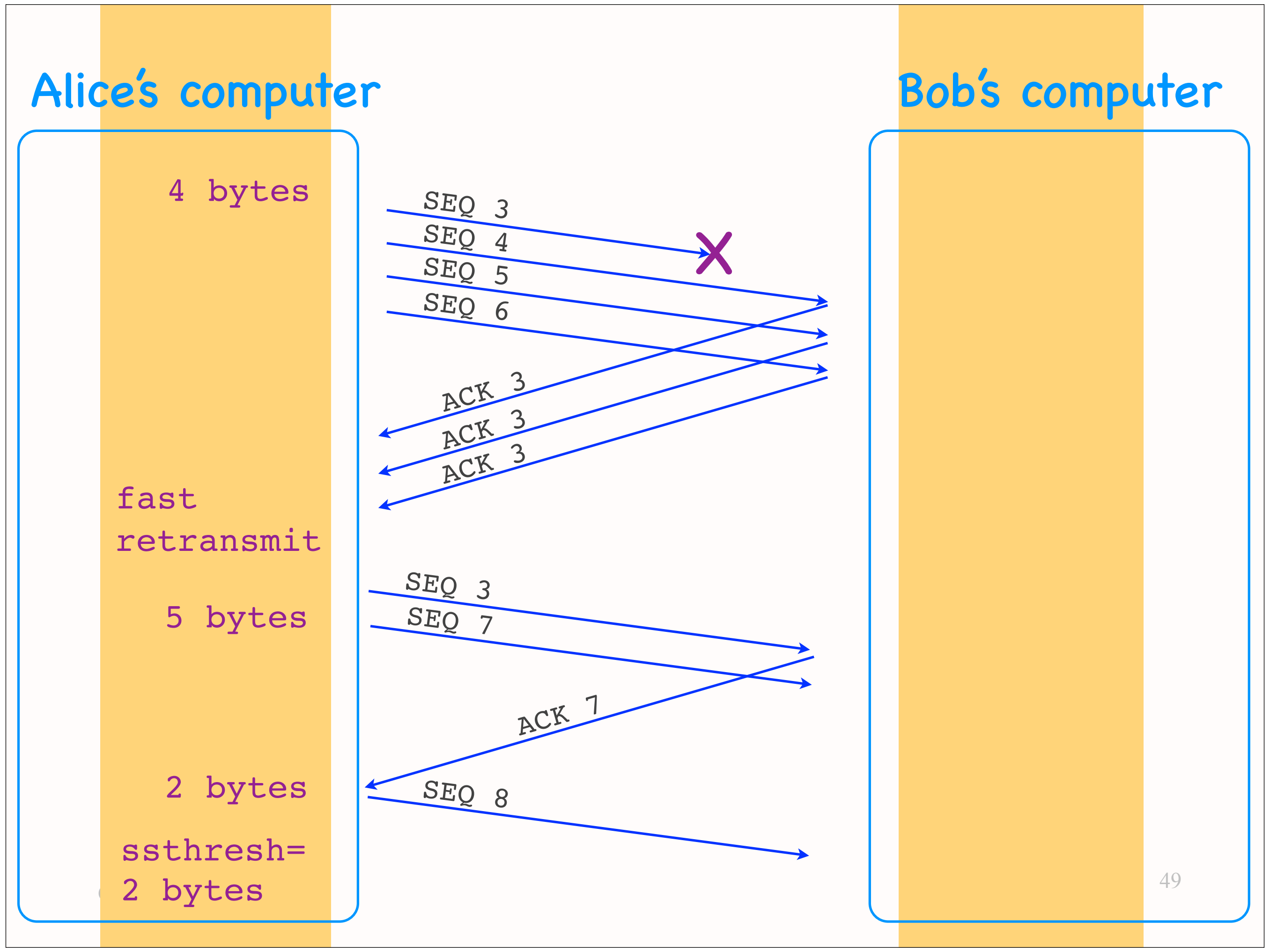
- Alice sends 1 MSS.
- When she gets an ACK, she increases her congestion-window size by 1 MSS (to 2 bytes), and she sends 2 more segments.
- At this point, the congestion-window size has reached the value of the threshold, and TCP transitions to the linear-increase state. Which means that Alice will now increase her congestion-window size by 1 MSS *every RTT* (not with every ACK she receives).
- So, when Alice gets another ACK, she increases her congestion-window size to N + MSS^2/N = 2+1/2 byte.
- When she gets another ACK, she again increases her congestion-window size to N+ MSS^2/N = 2.5+1/2.5 byte = approx. 3 bytes.
- And so on.



Basic algorithm (Tahoe)

- Set window to 1 MSS,
  increase exponentially
- On timeout, reset window to 1 MSS,
  set ssthresh to last window/2
- On reaching ssthresh,
  transition to linear increase

This is the basic TCP congestion control algorithm:
- Start with exponential increase,
- then transition to linear increase when congestion is expected.



Suppose we are at a point where Alice has set her congestion-window size to 4 MSSs (4 bytes), has sent 4 un-ACK-ed segments, and the first one has been lost.

Here is what happens next:

- Alice gets 3 duplicate ACKs, so she decides to do a fast retransmit.
- She wants to send as many bytes as indicated by her congestion threshold, i.e., 2 bytes.
- To do that, she inflates her congestion-window size by 3 bytes (as many as the bytes that are implicitly acknowledged through the 3 duplicate ACKs), i.e., to 5 bytes. This allows her to send 1 more byte (byte No 7) after the retransmitted byte (byte. No 3). Then, TCP transitions to the Fast Recovery state.
- Once Alice receives a new ACK (ACK 7), which means that Bob received the retransmitted byte, Alice deflates her window back to 2 bytes, which allows her to send one more unacknowledged byte (byte No 8). Then, TCP transitions to the Linear Increase (Congestion Avoidance) state.

Basic algorithm (Reno)

- Set window to 1 MSS,
  increase exponentially
- On timeout, reset window to 1 MSS,
  set ssthresh to last window/2
- On reaching ssthresh or 3 duplicate ACKs,
  transition to linear increase

Two retransmission triggers

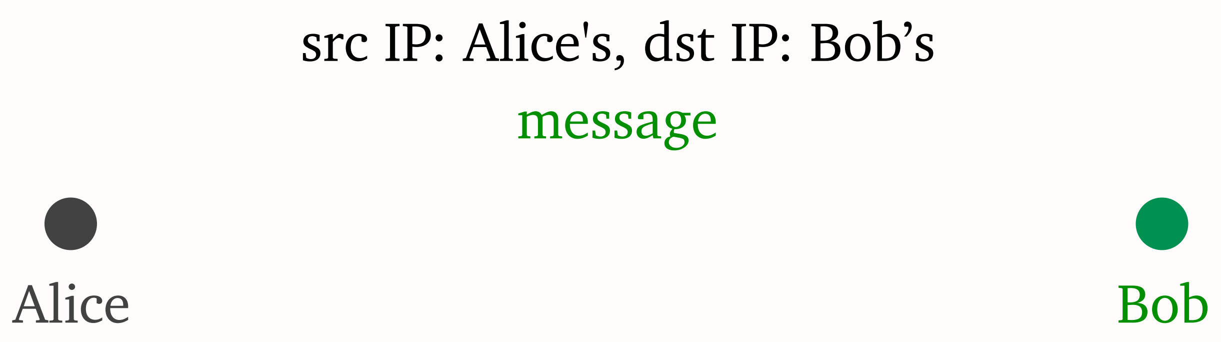
- Timeout => retransmission of oldest unacknowledged segment
- 3 duplicate ACKs => fast retransmit of oldest unacknowledged segment
  - avoid unnecessary wait for timeout
  - 1 duplicate ACK not enough <= network may have reordered a data segment or duplicated an ACK

TCP terminology

- Exponential increase = slow start
  - on timeout, reset window to 1 MSS
  - set ssthresh to last window/2
- Linear increase = congestion avoidance
  - on window reaching ssthresh
  - on receiving 3 duplicate ACKs

Exam material

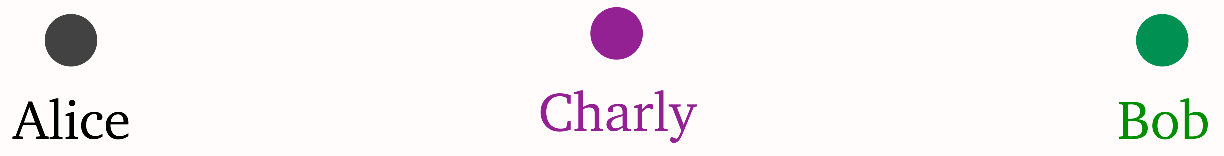
- All lectures, homework, labs
  from semester start
- Emphasis on material after midterm + TCP
- Lab-related questions: <=20% of the points


src IP: Alice's, dst IP: Bob’s
message
Alice
Bob

src IP: Alice’s, dst IP: Charly’s

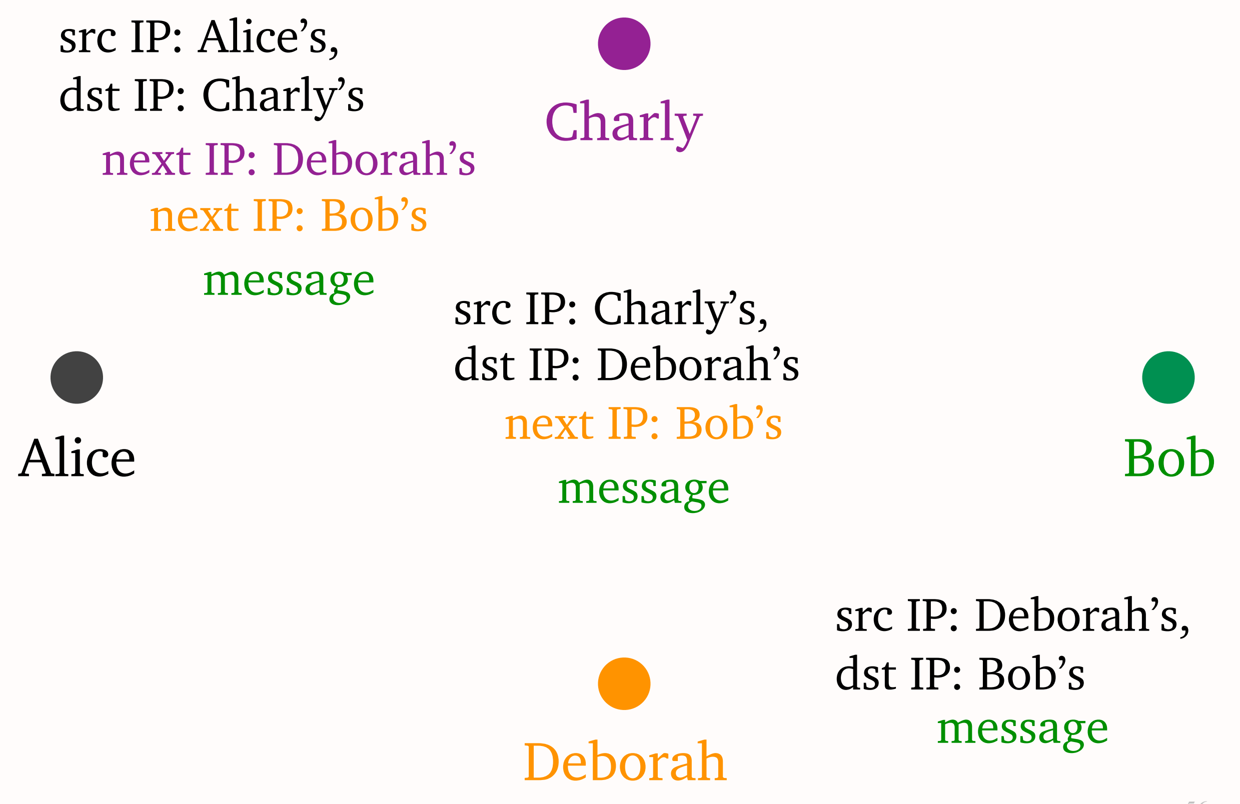
next IP: Bob’s

message



src IP: Charly’s, dst IP: Bob’s

message


src IP: Alice’s,
dst IP: Charly’s
next IP: Deborah’s
next IP: Bob’s
message
Charly
Alice
src IP: Charly’s,
dst IP: Deborah’s
next IP: Bob’s
message
Bob
src IP: Deborah’s,
dst IP: Bob’s
message
Deborah