Networking fundamentals

car navigator

heart pacemaker

smartphone

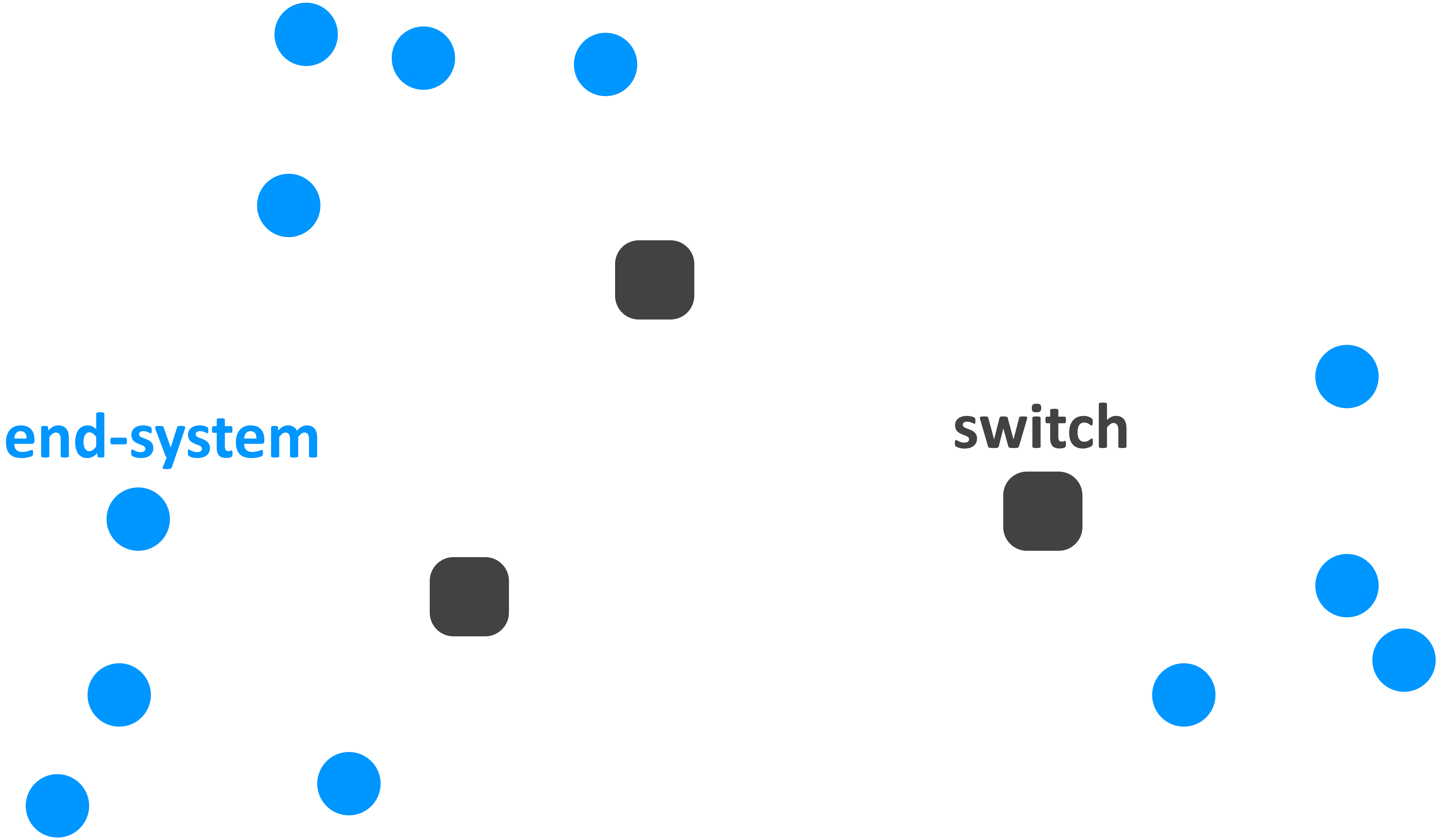
end-system

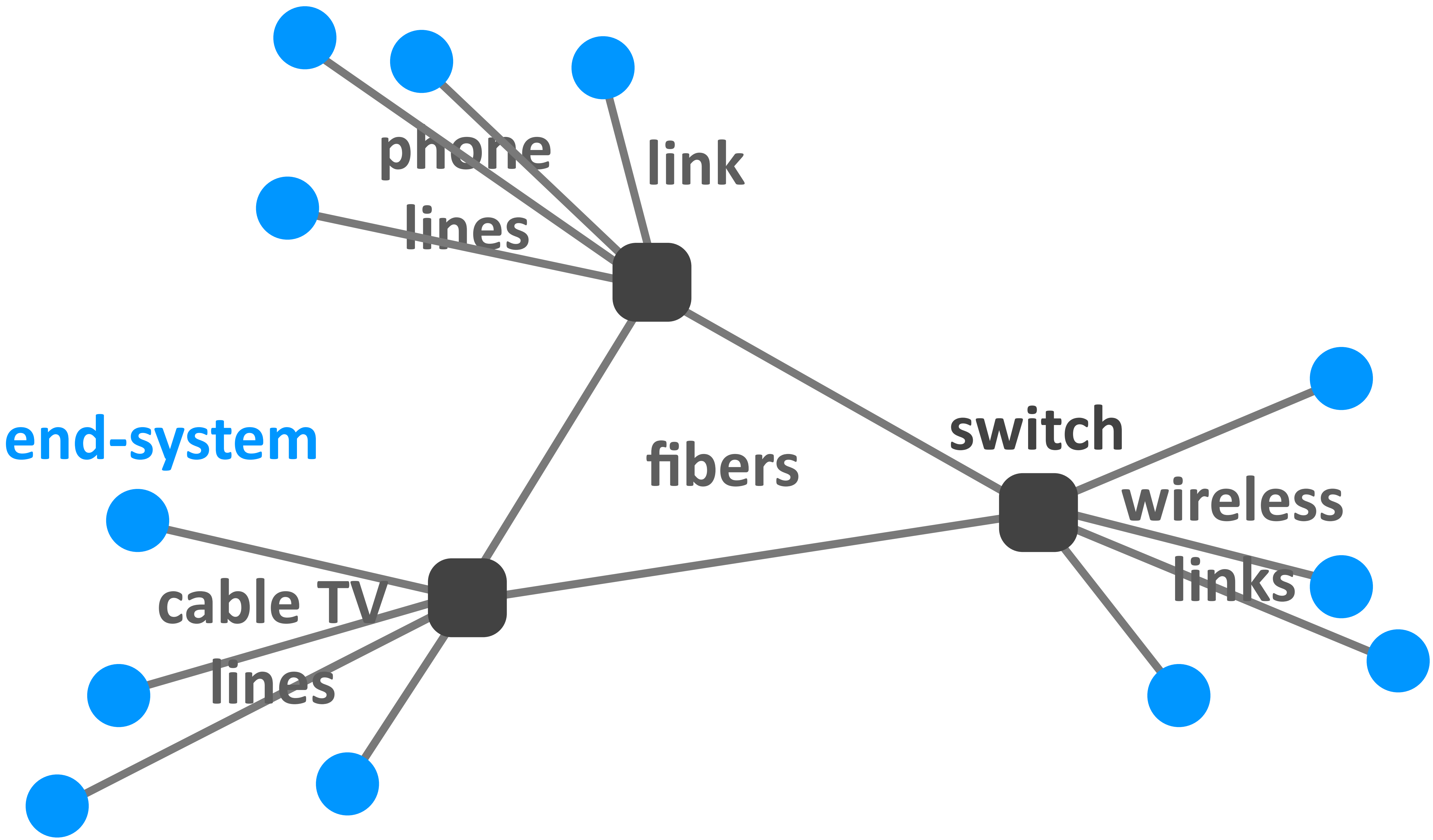
iPad

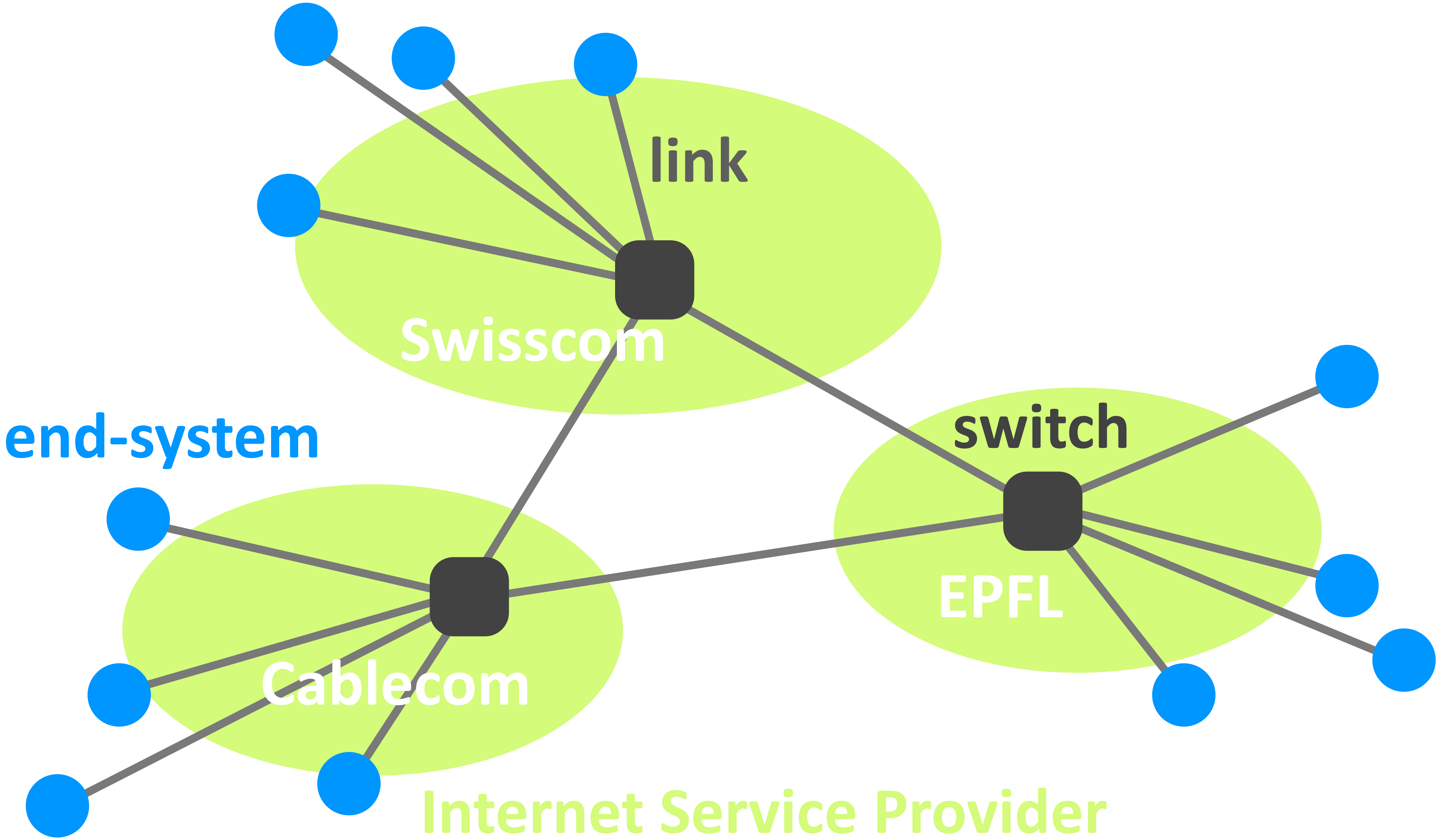
Linux server

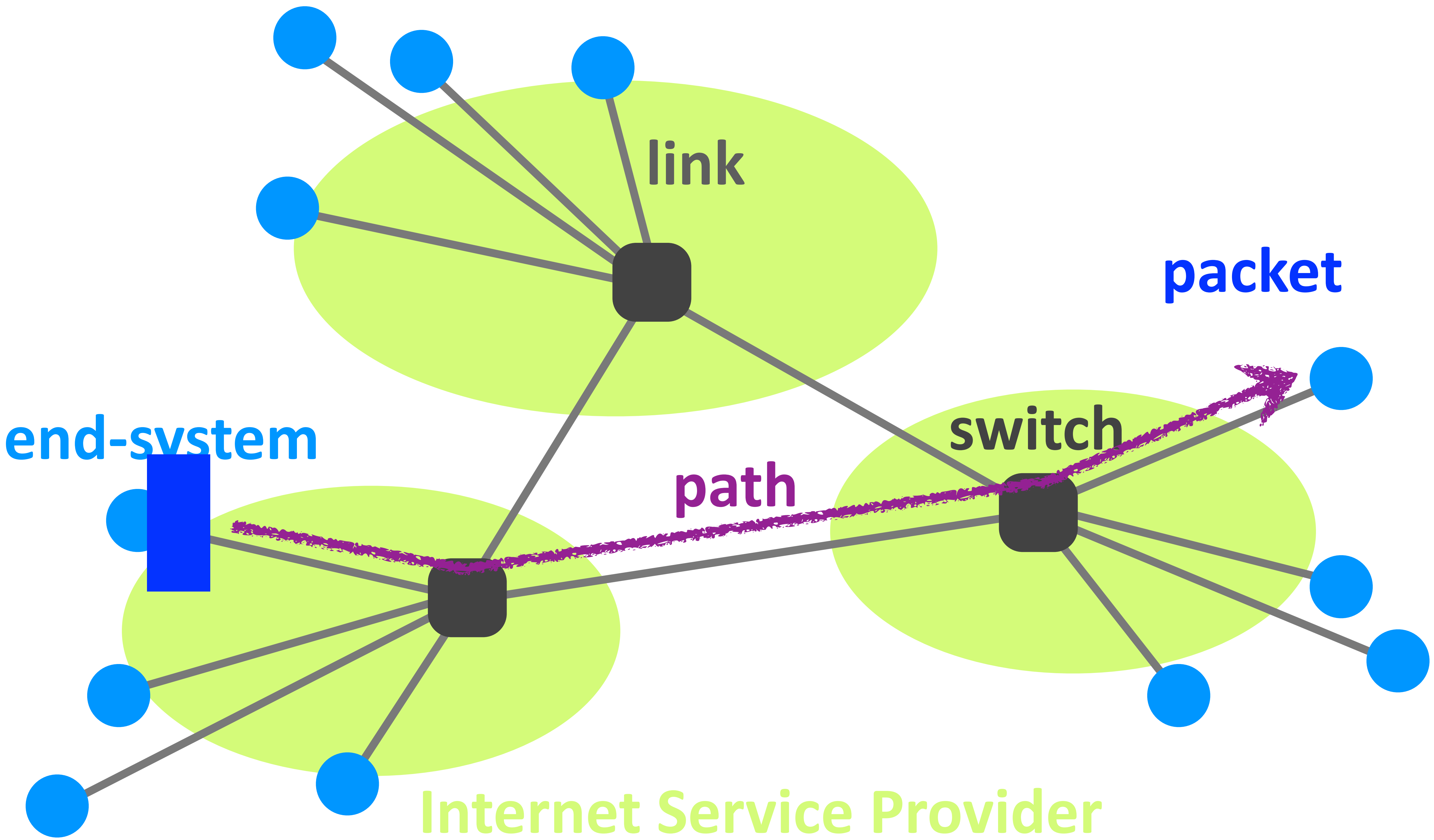
MAC laptop

Windows PC


end-system
switch


phone
lines
link
end-system
fibers
switch
wireless
links
cable TV
lines


link
Swisscom
end-system
switch
EPFL
Cablecom
Internet Service Provider

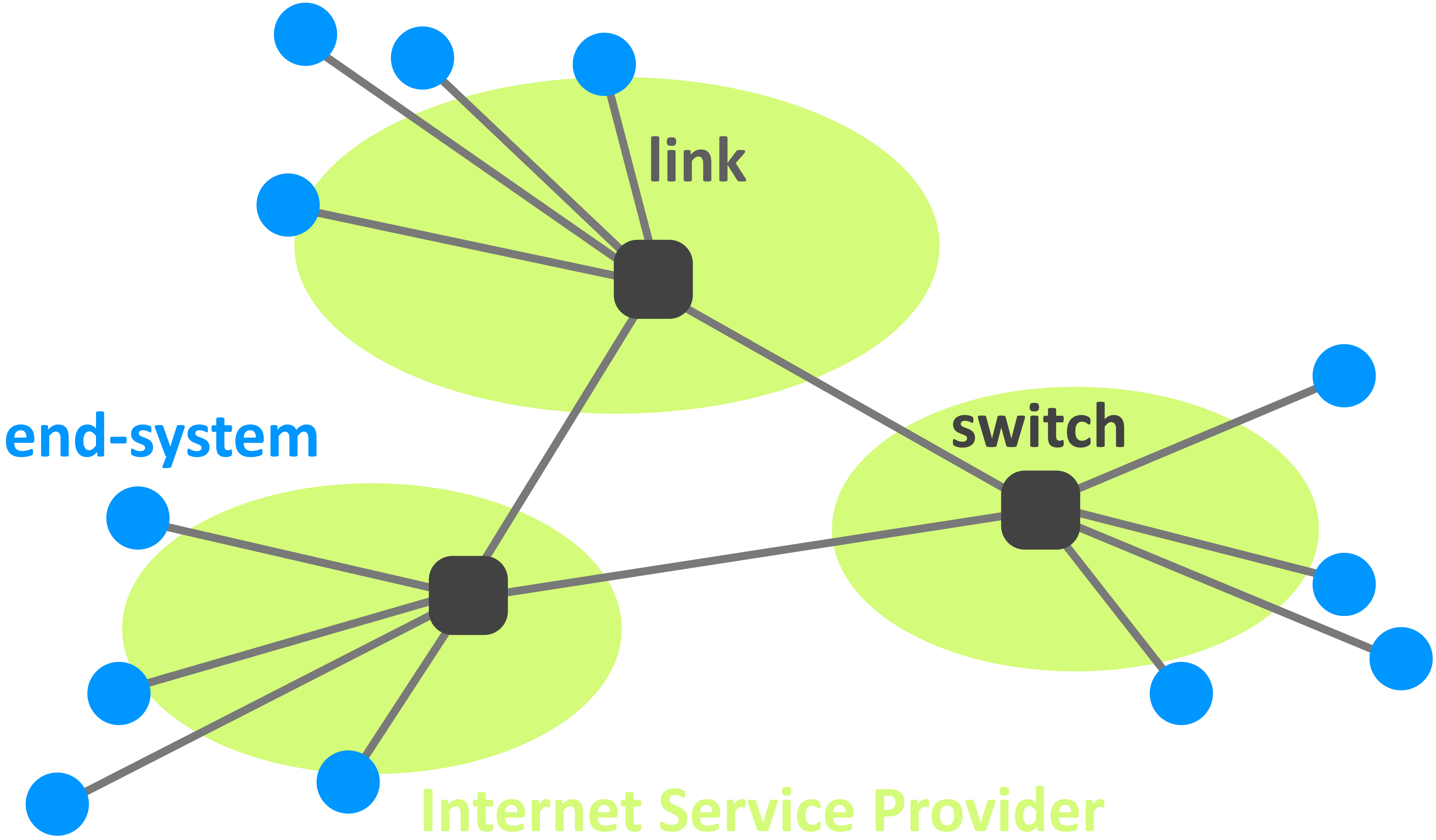

link
packet
end-system
switch
path
Internet Service Provider

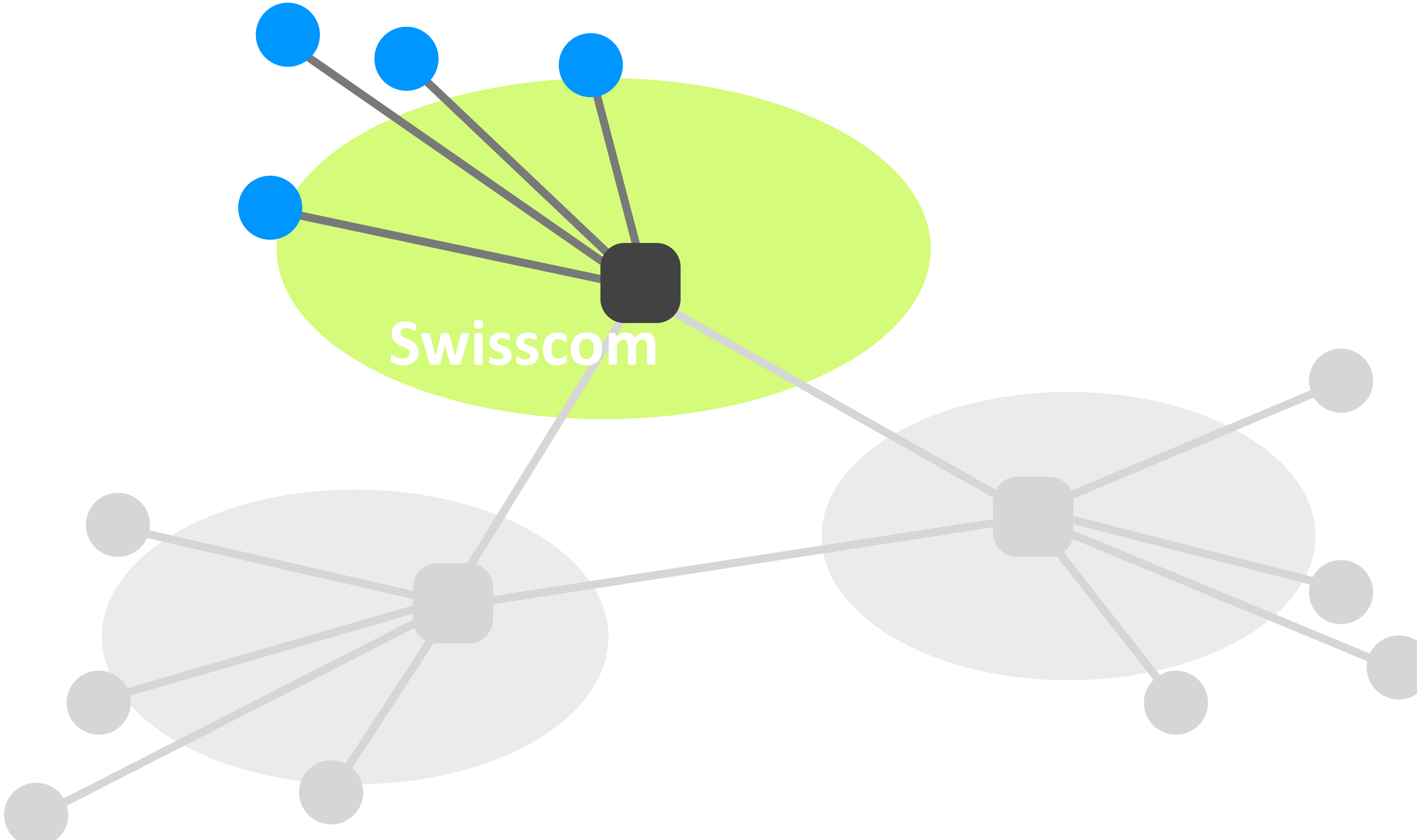
Outline

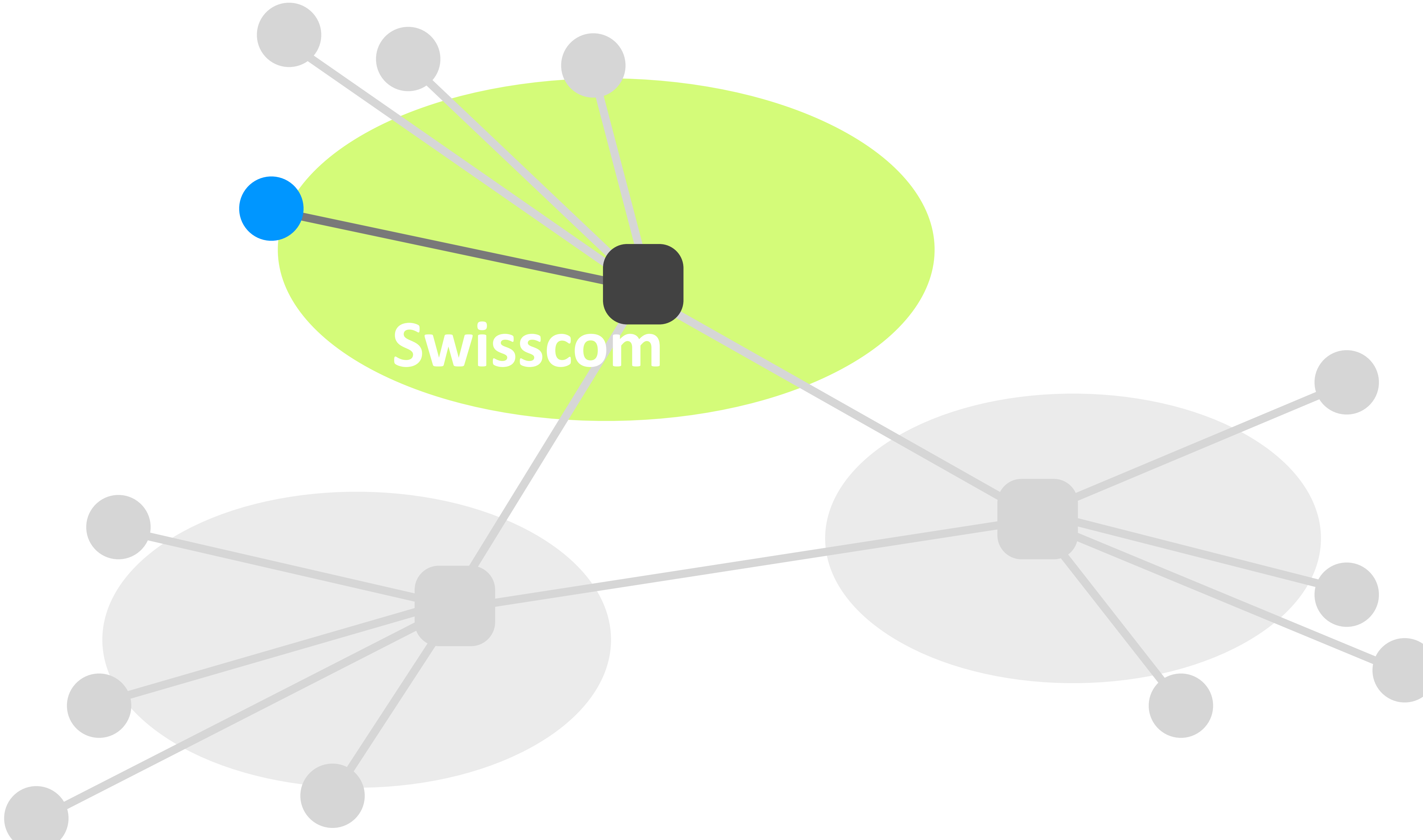
- Links & switches
- ISP relationships
- Performance metrics
- Layers

Outline

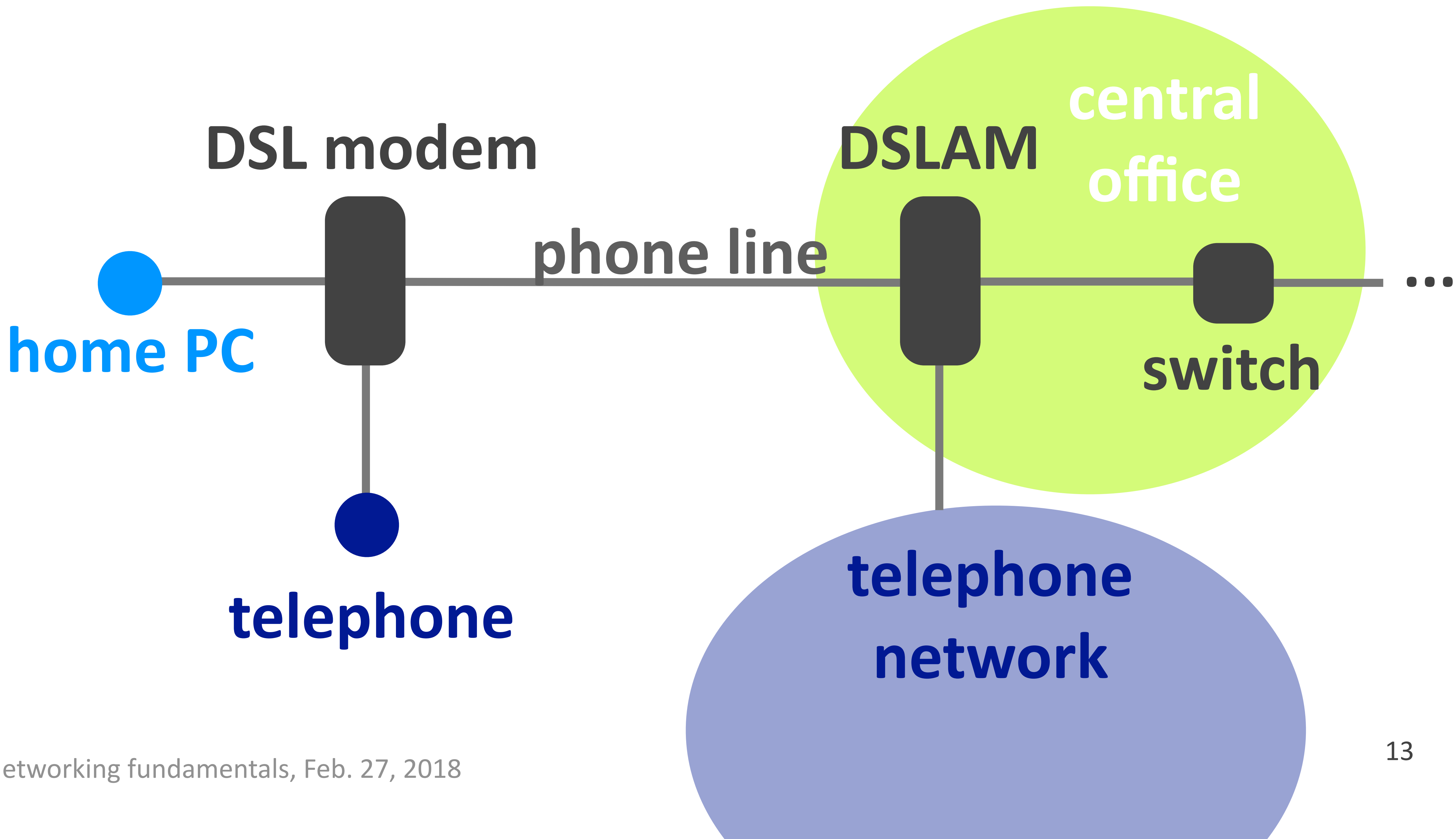
- Links & switches
- ISP relationships
- Performance metrics
- Layers


link
end-system
switch
Internet Service Provider

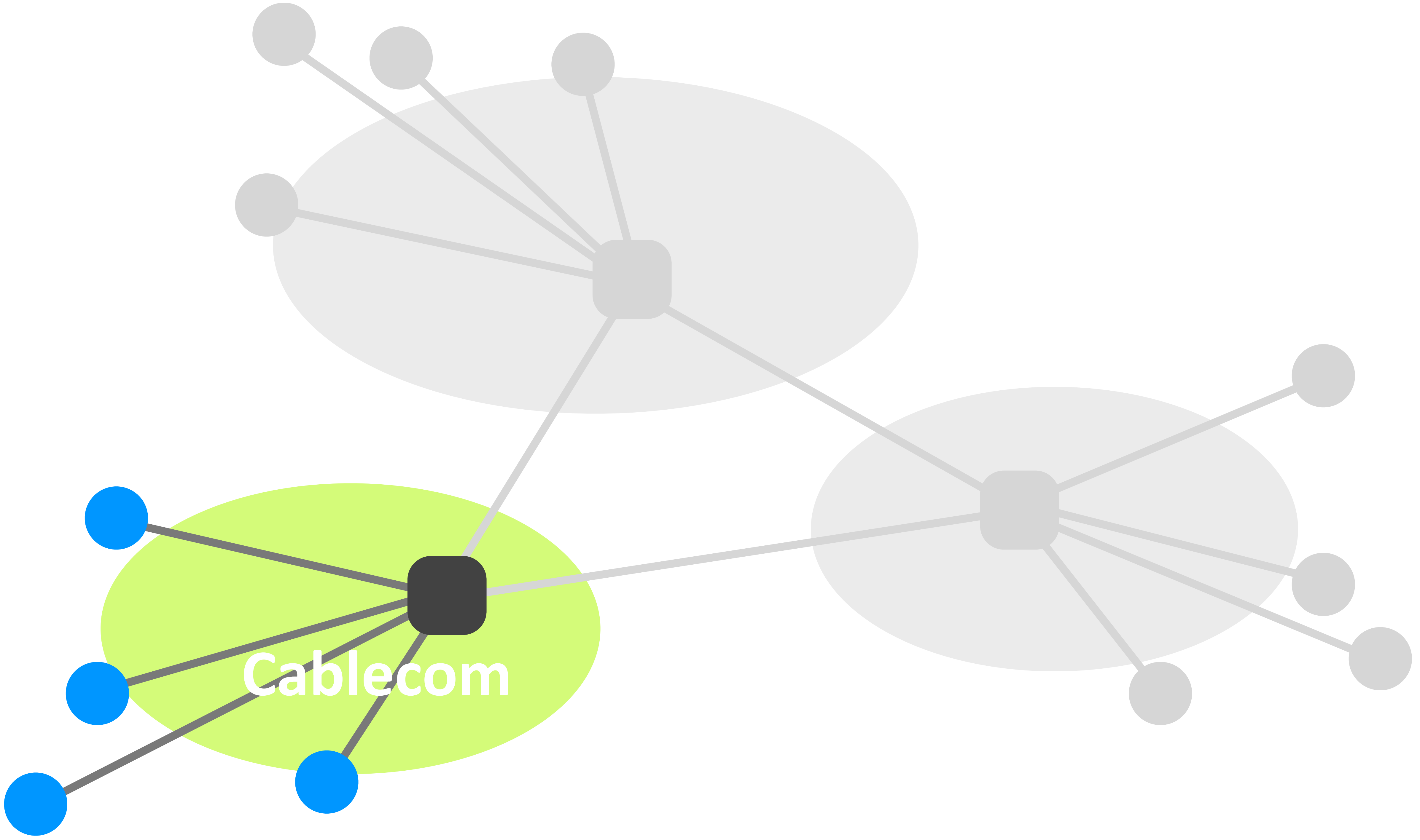

Swisscom

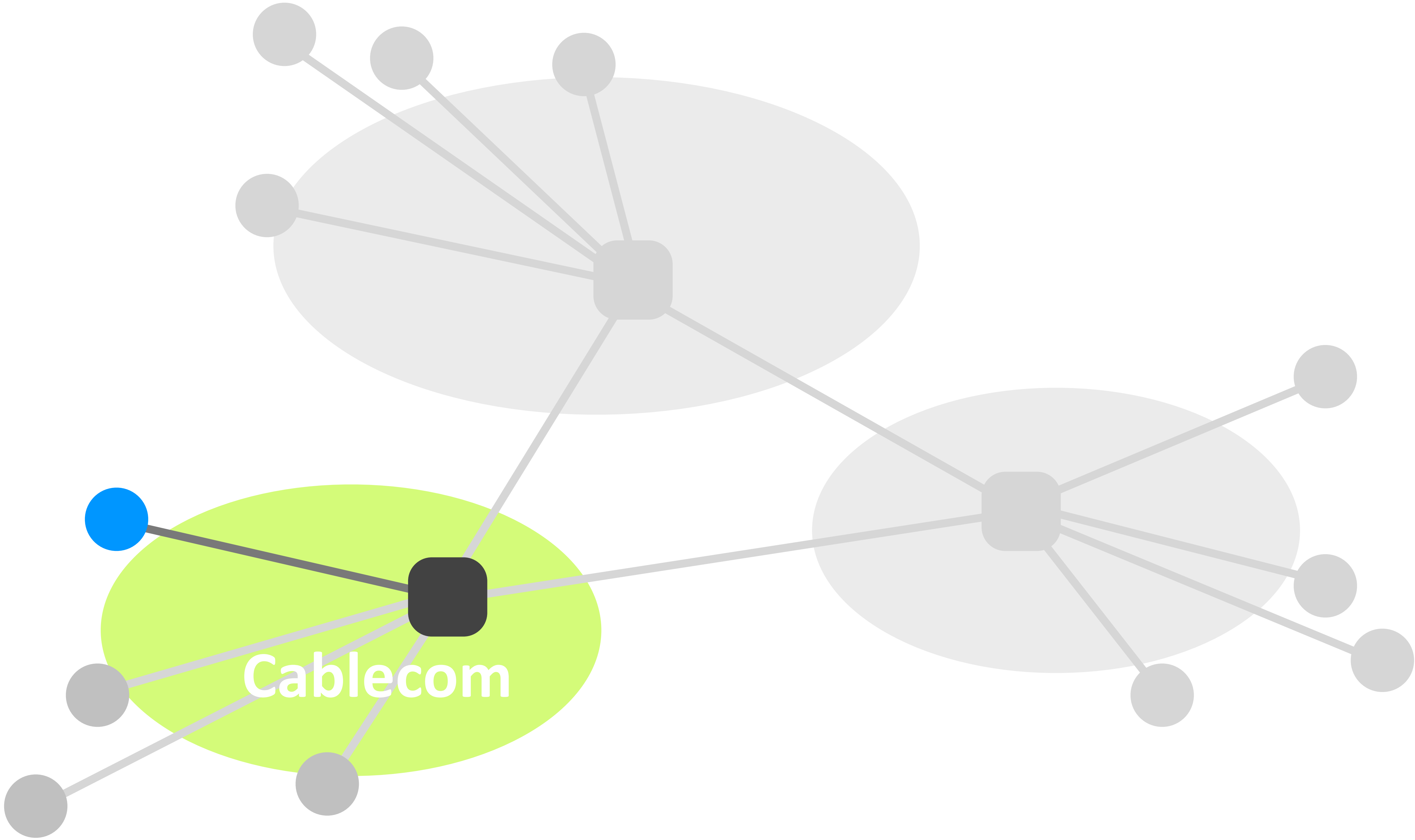

Swisscom


home PC
switch

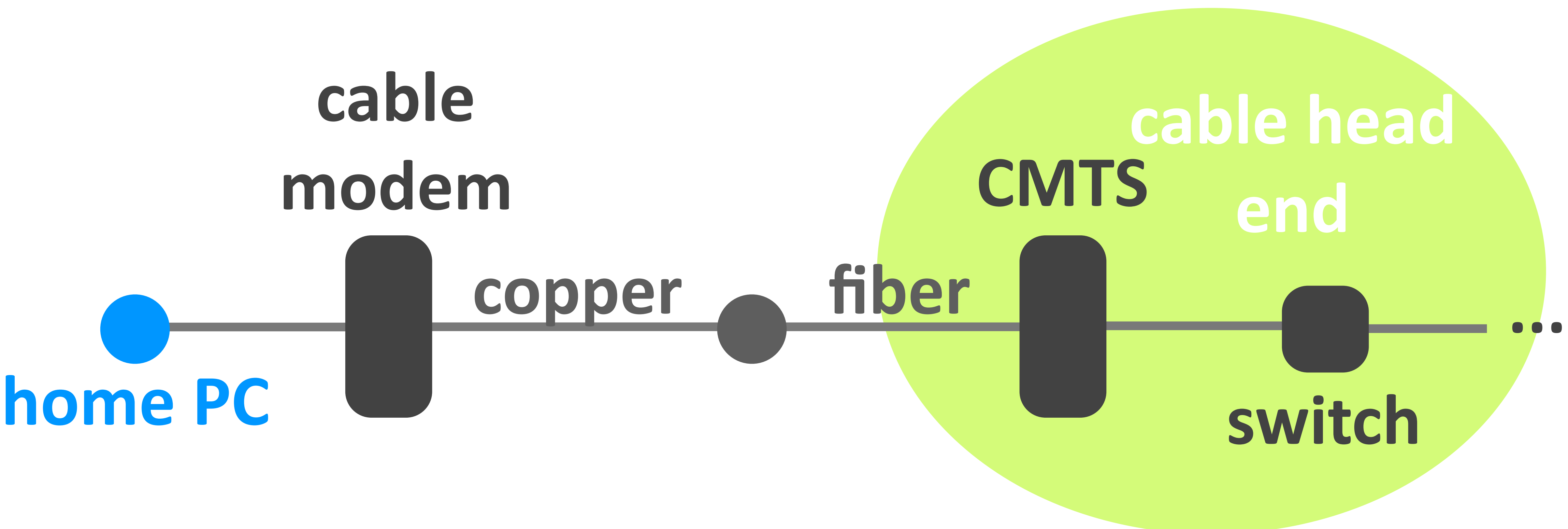

DSL modem
DSLAM
central office
phone line
home PC
switch
...
telephone
telephone network

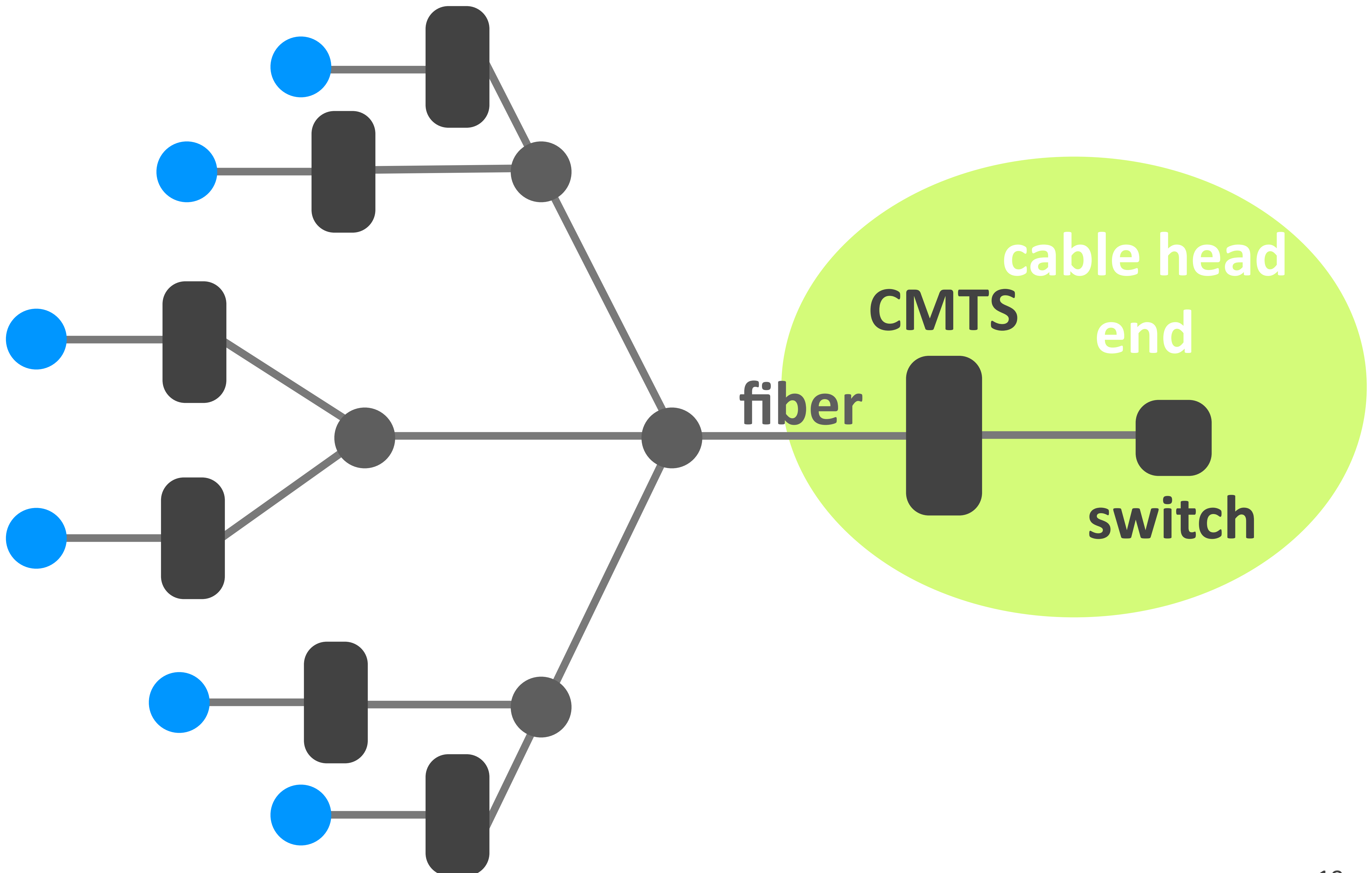
Why phone lines?

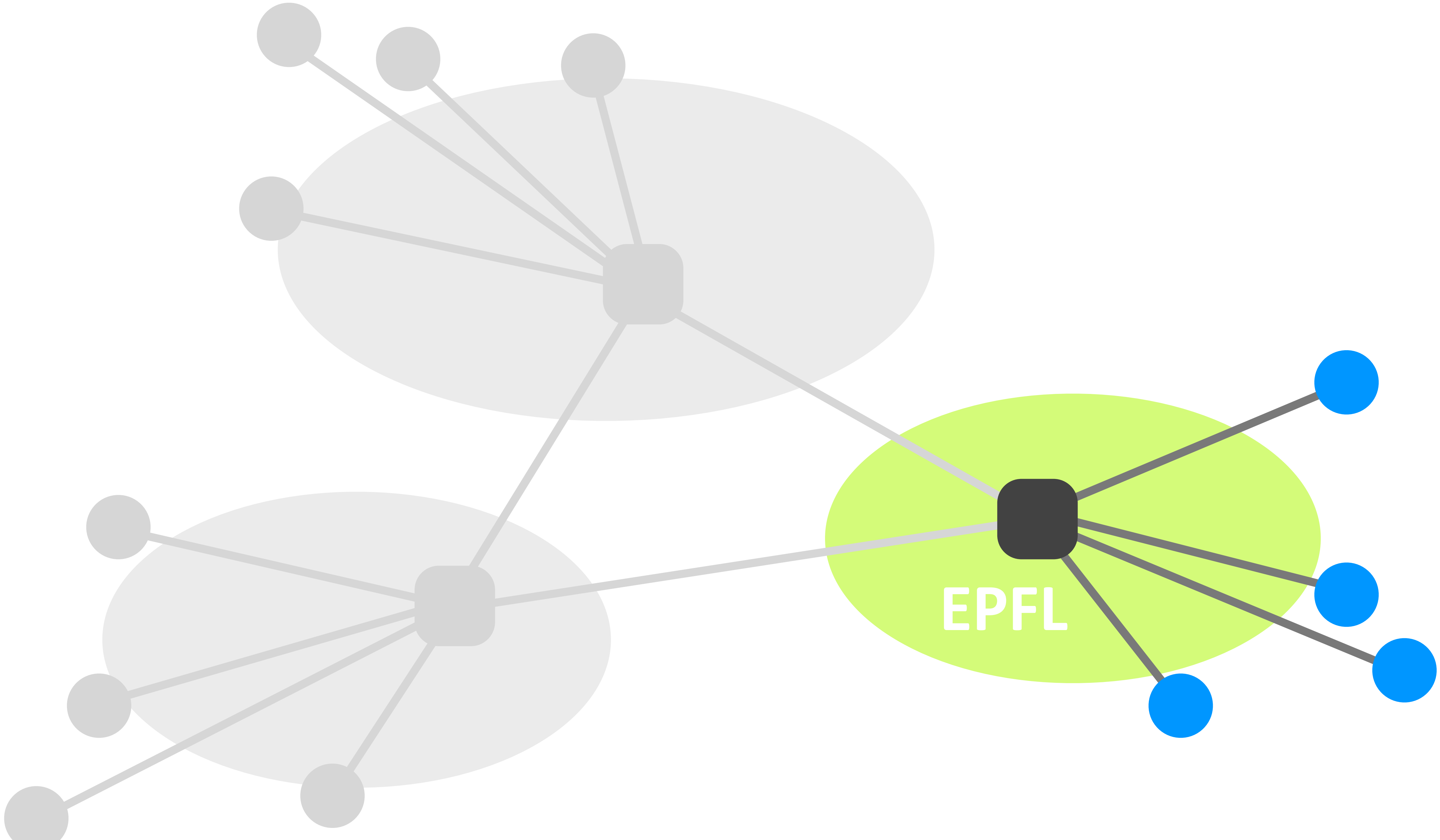

Cablecom

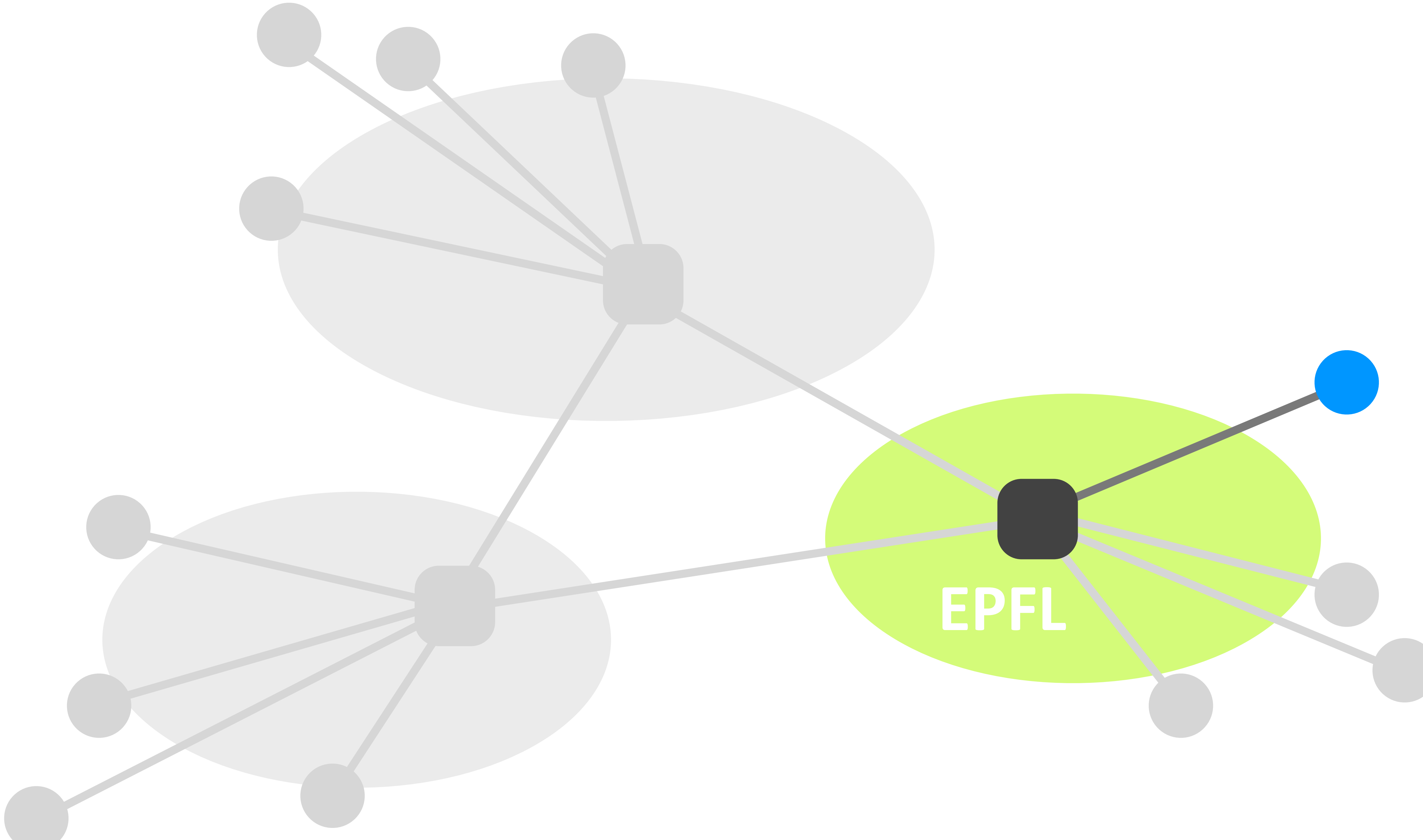

Cablecom

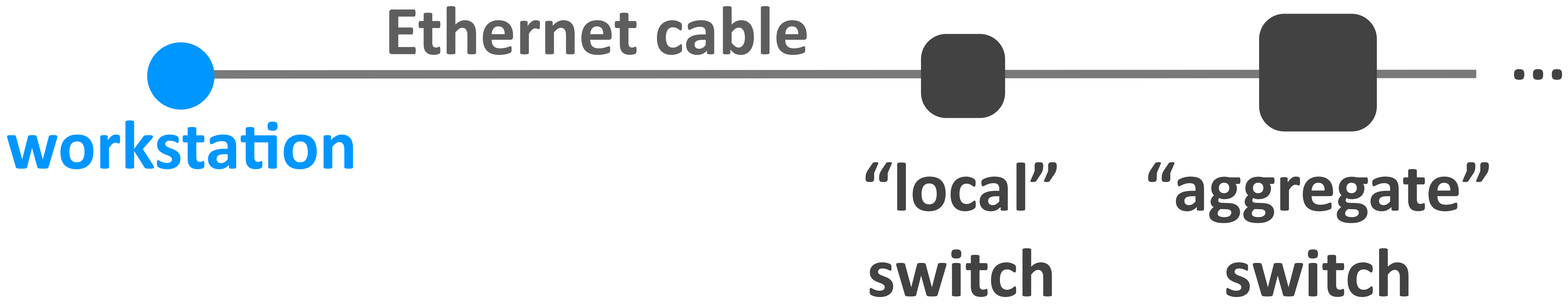

home PC
switch


cable
modem
cable head
end
CMTS
copper
fiber
home PC
switch
...


cable head end
CMTS
fiber
switch

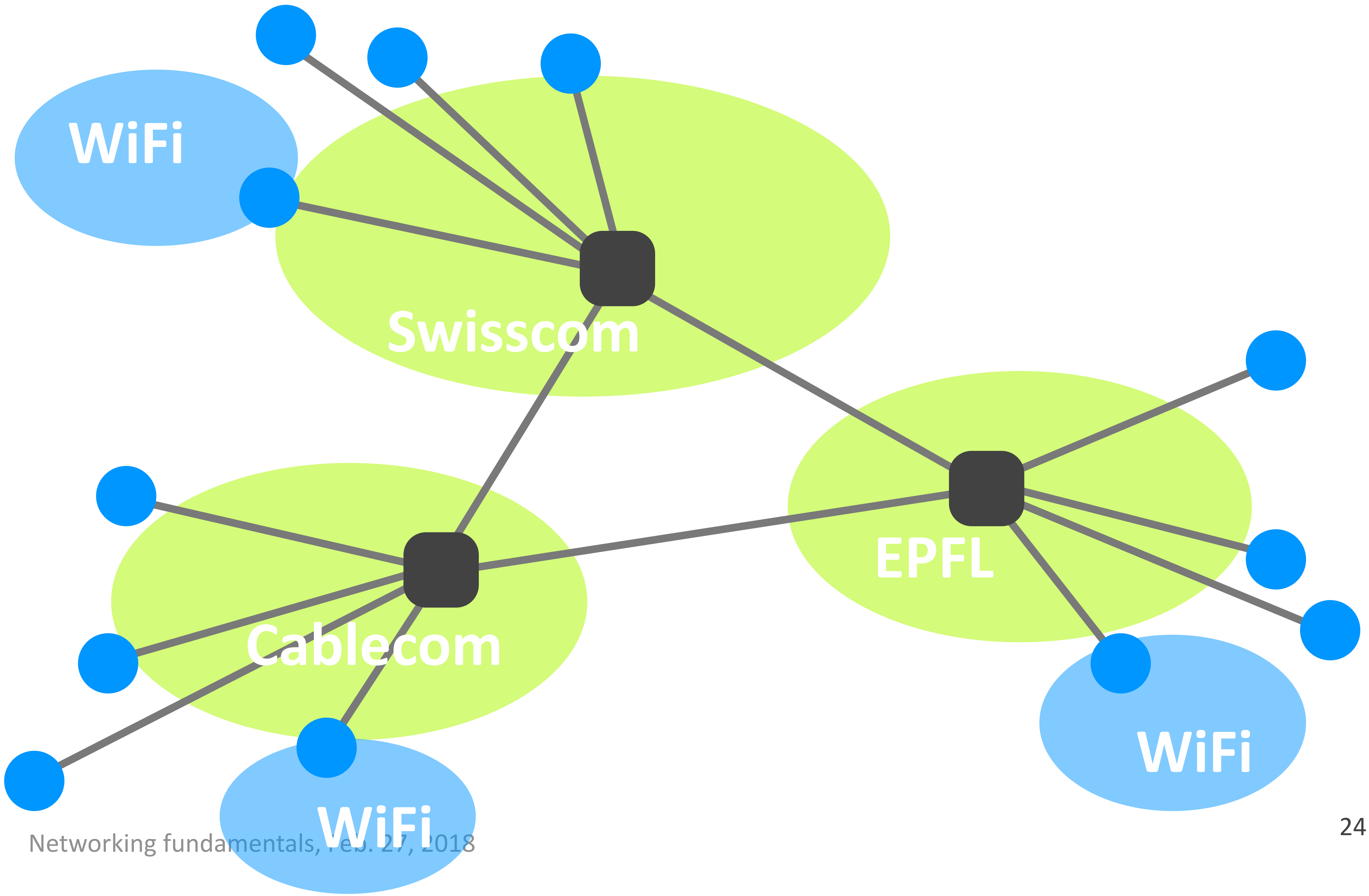

EPFL

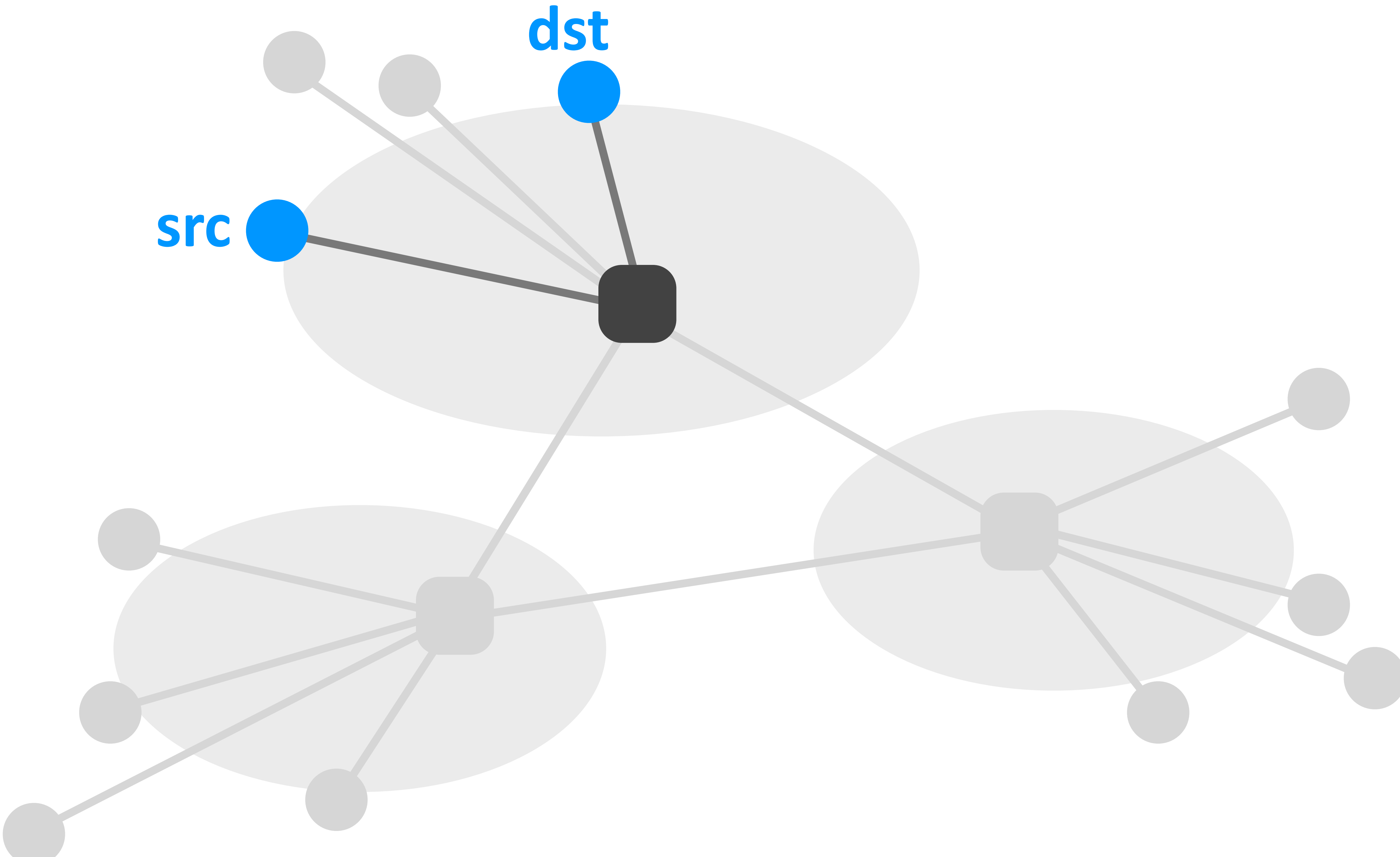

EPFL

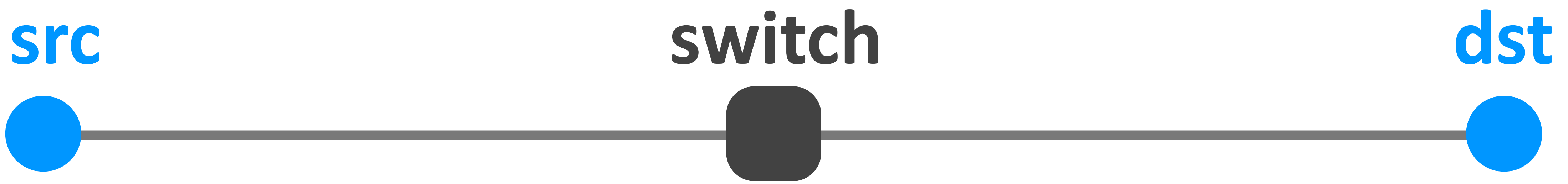

Ethernet cable
workstation
"local"
switch
"aggregate"
switch
...

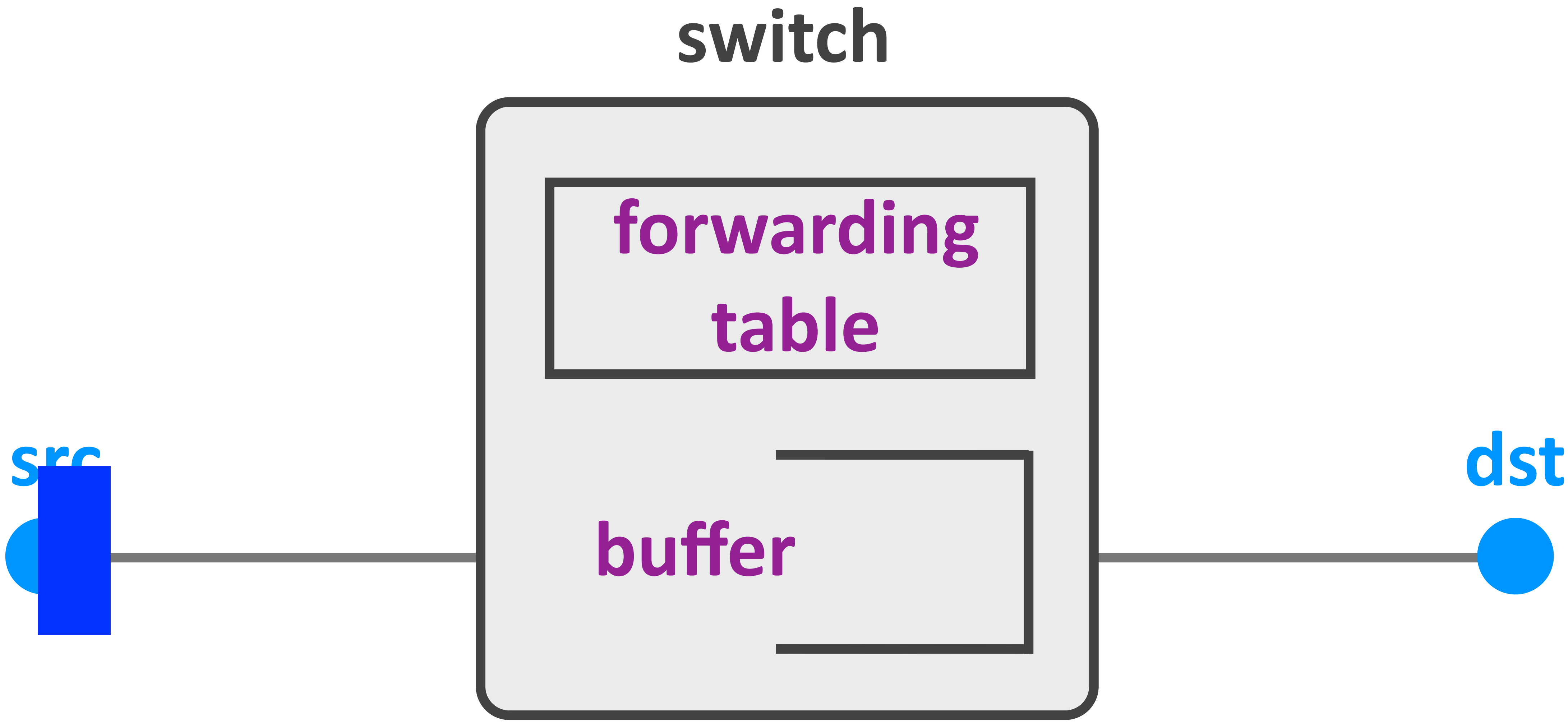
& more

- Cellular (smart phones)
- Satellite (remote areas)
- Fiber to the Home (home)
- Optical carrier (Internet backbone)


WiFi
Swisscom
EPFL
Cablecom
WiFi
WiFi

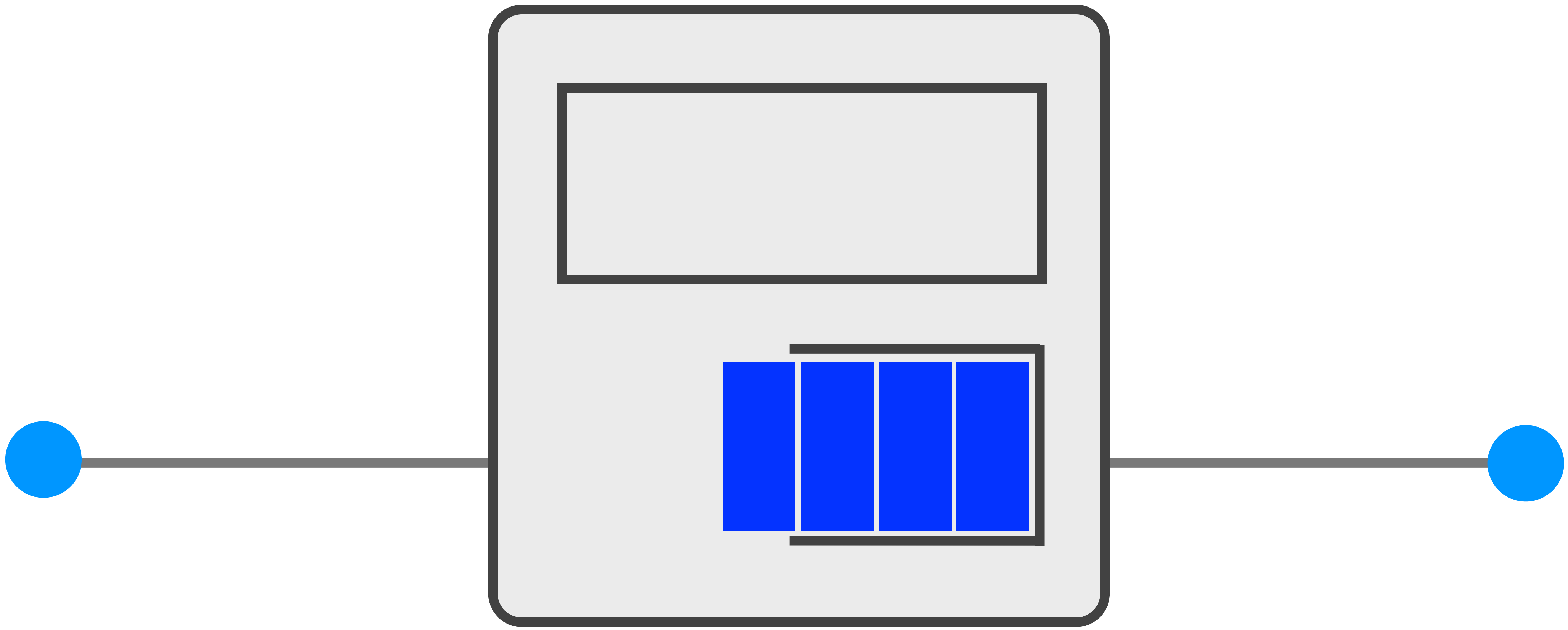

dst
src


src
switch
dst

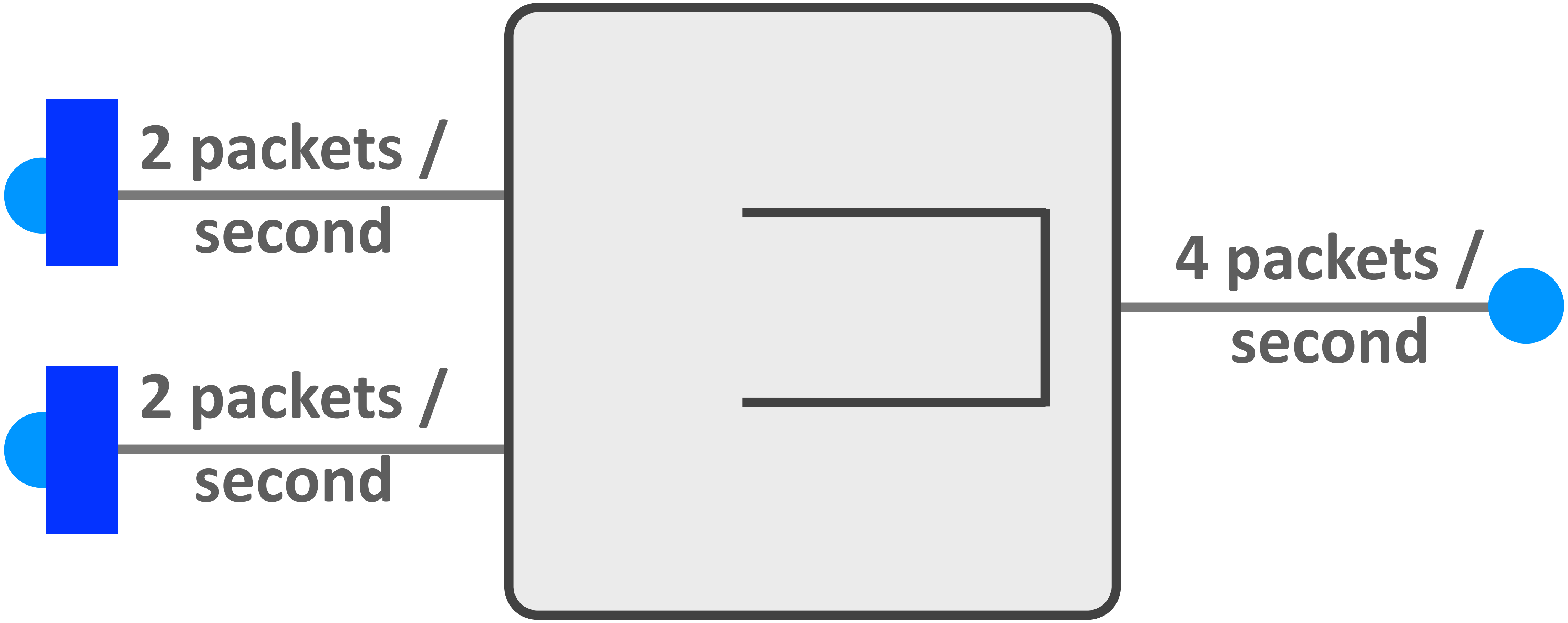

switch
forwarding table
buffer
src
dst

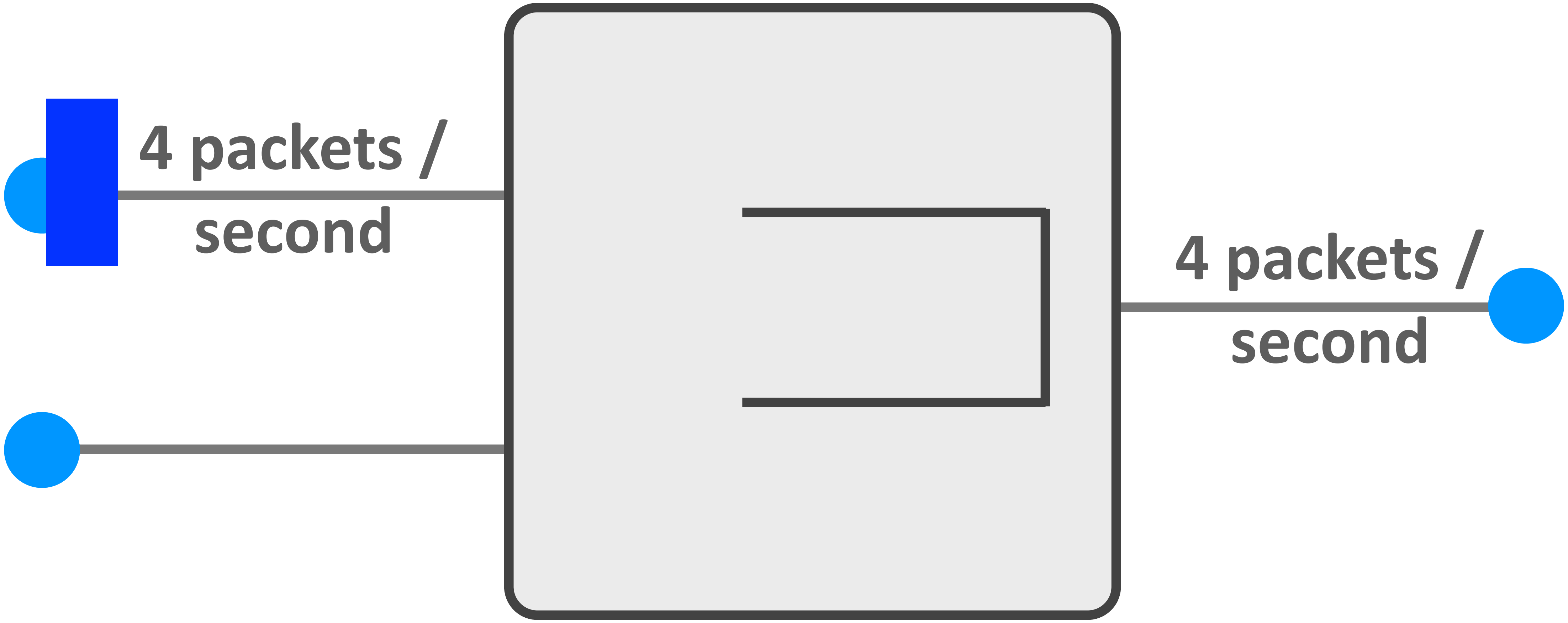
Switch contents

- Buffers
  - *store data*
- Forwarding table
  - *store meta-data*
  - *indicate where to send the data*

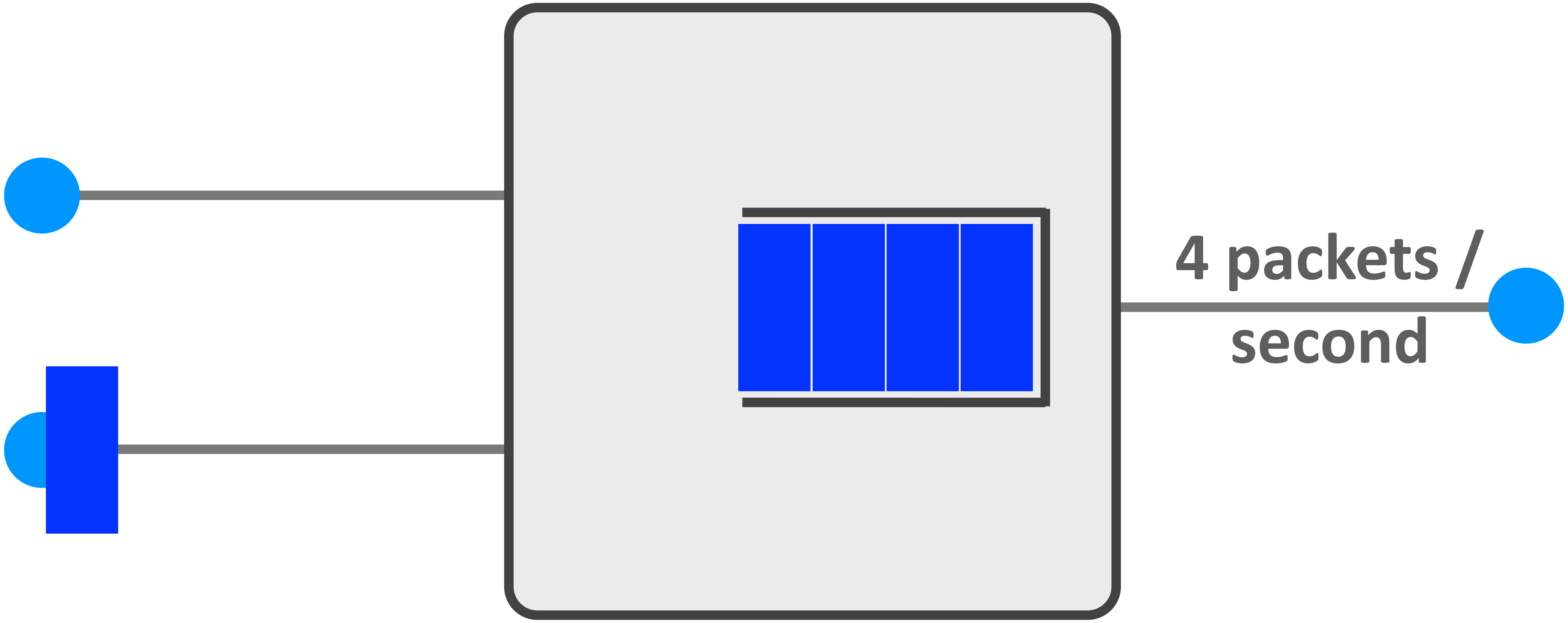


packet loss **queuing delay**


2 packets / second
2 packets / second
4 packets / second



Efficient use of resources



Unpredictable performance

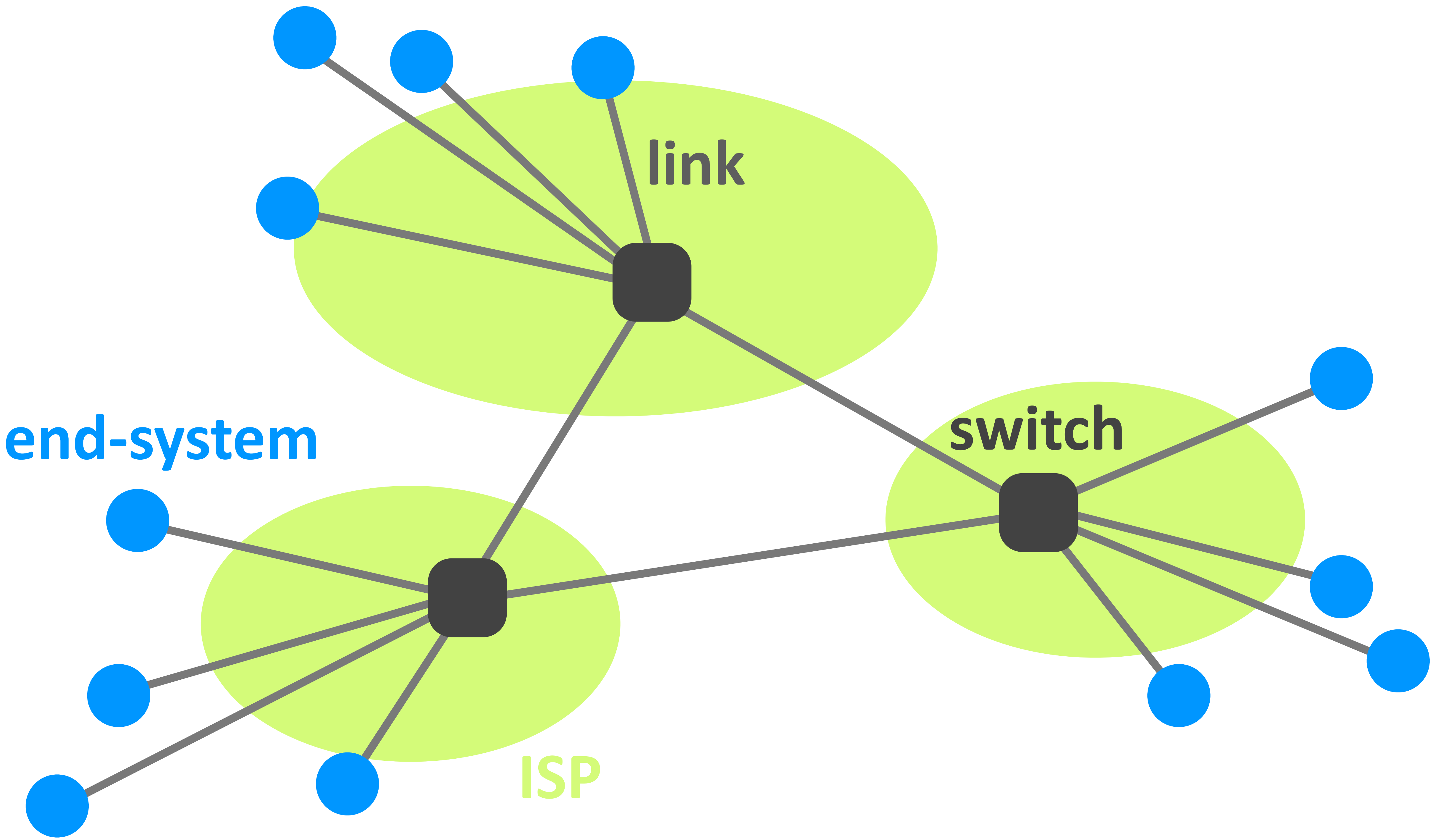
Best-effort service

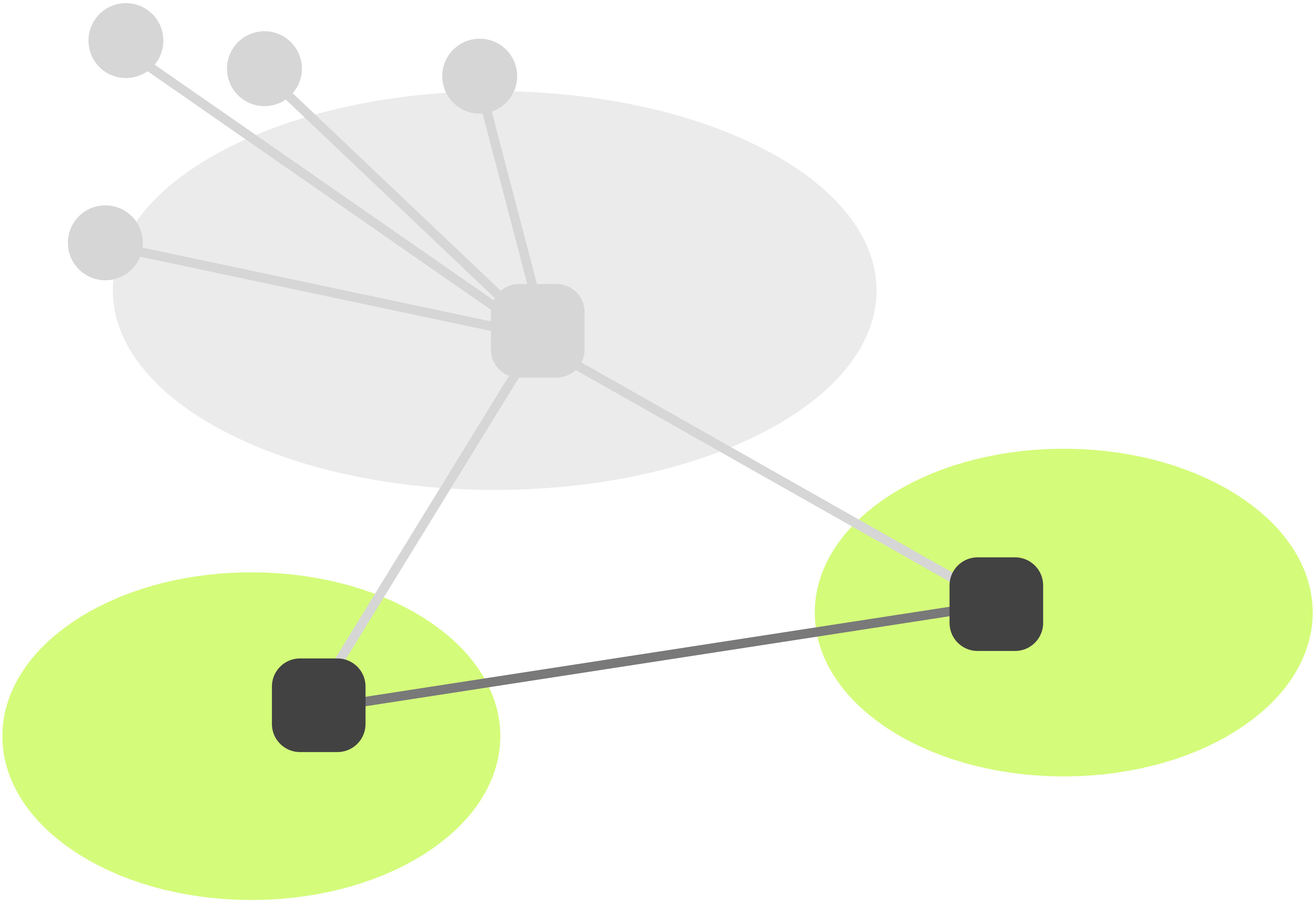
- Packets are treated on demand
  - *may be lost or experience queuing delay*
- Efficient use of resources
- But unpredictable performance

Why best-effort?

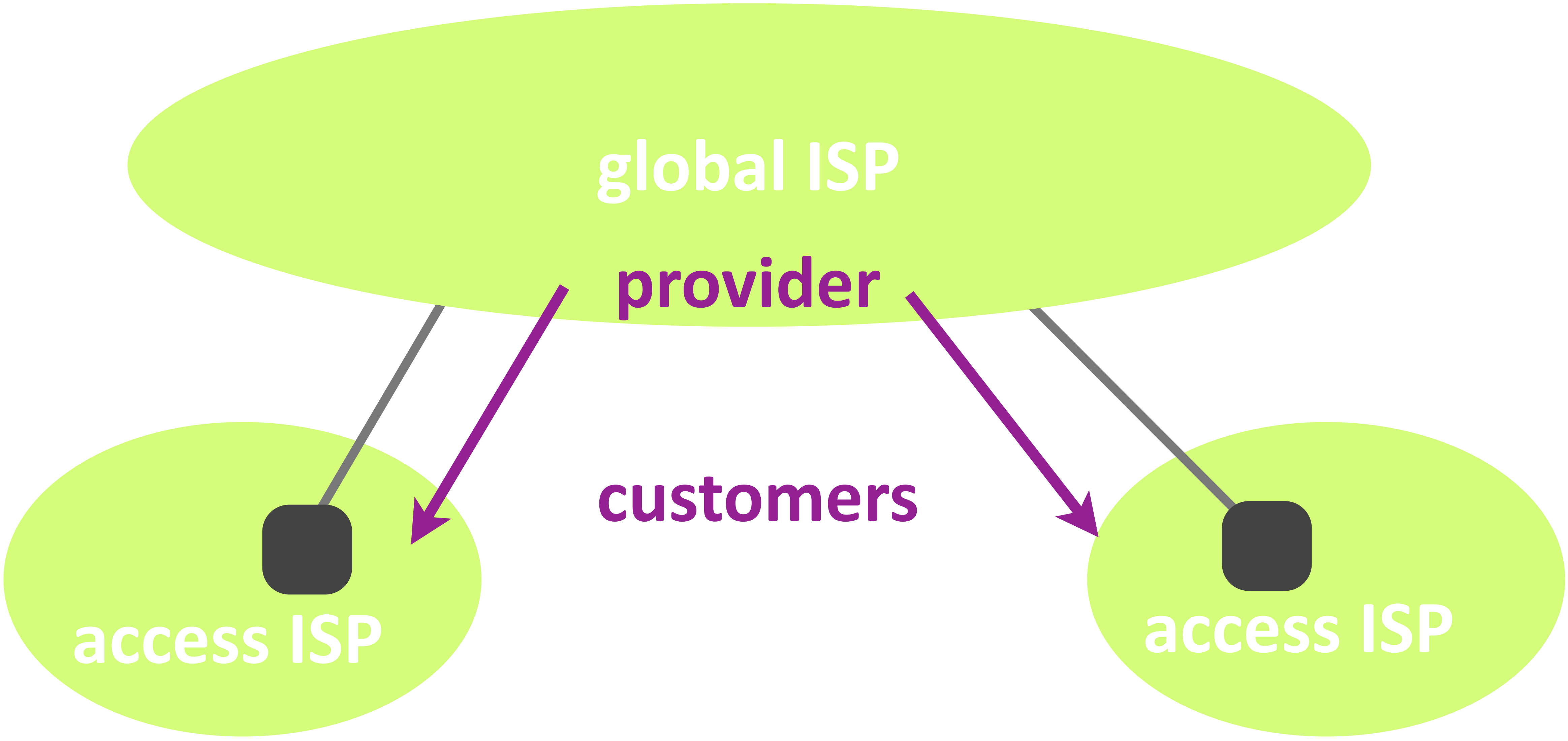
Outline

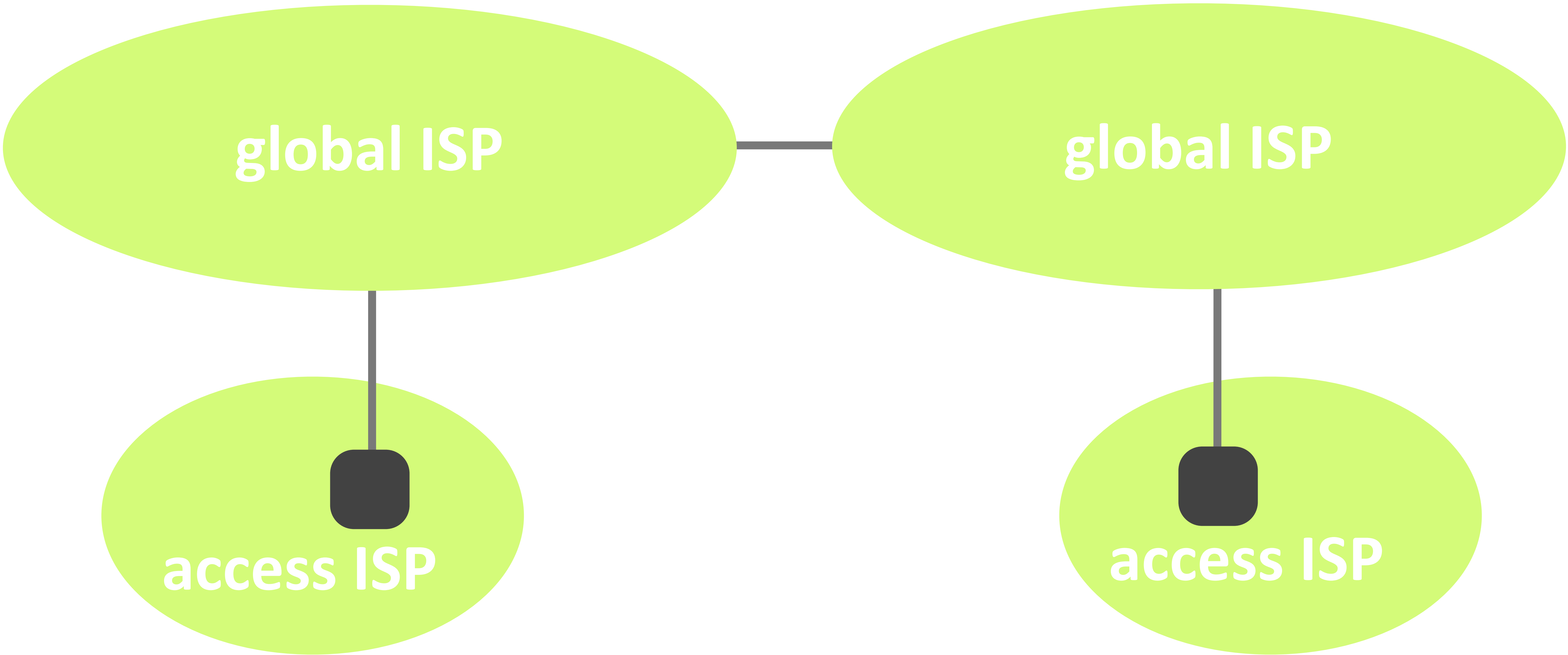
- Links & switches
- ISP relationships
- Performance metrics
- Layers

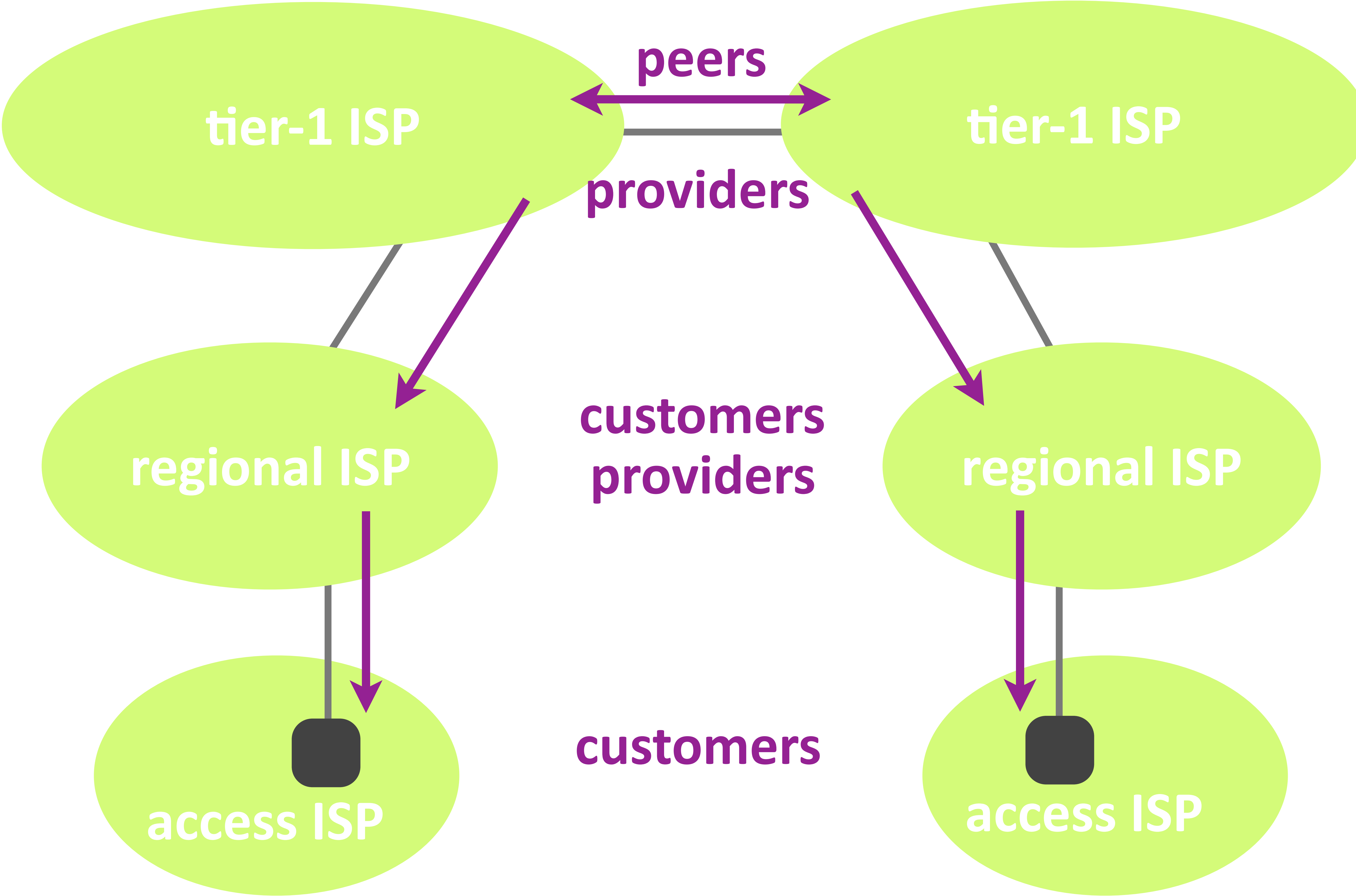

link
end-system
switch
ISP

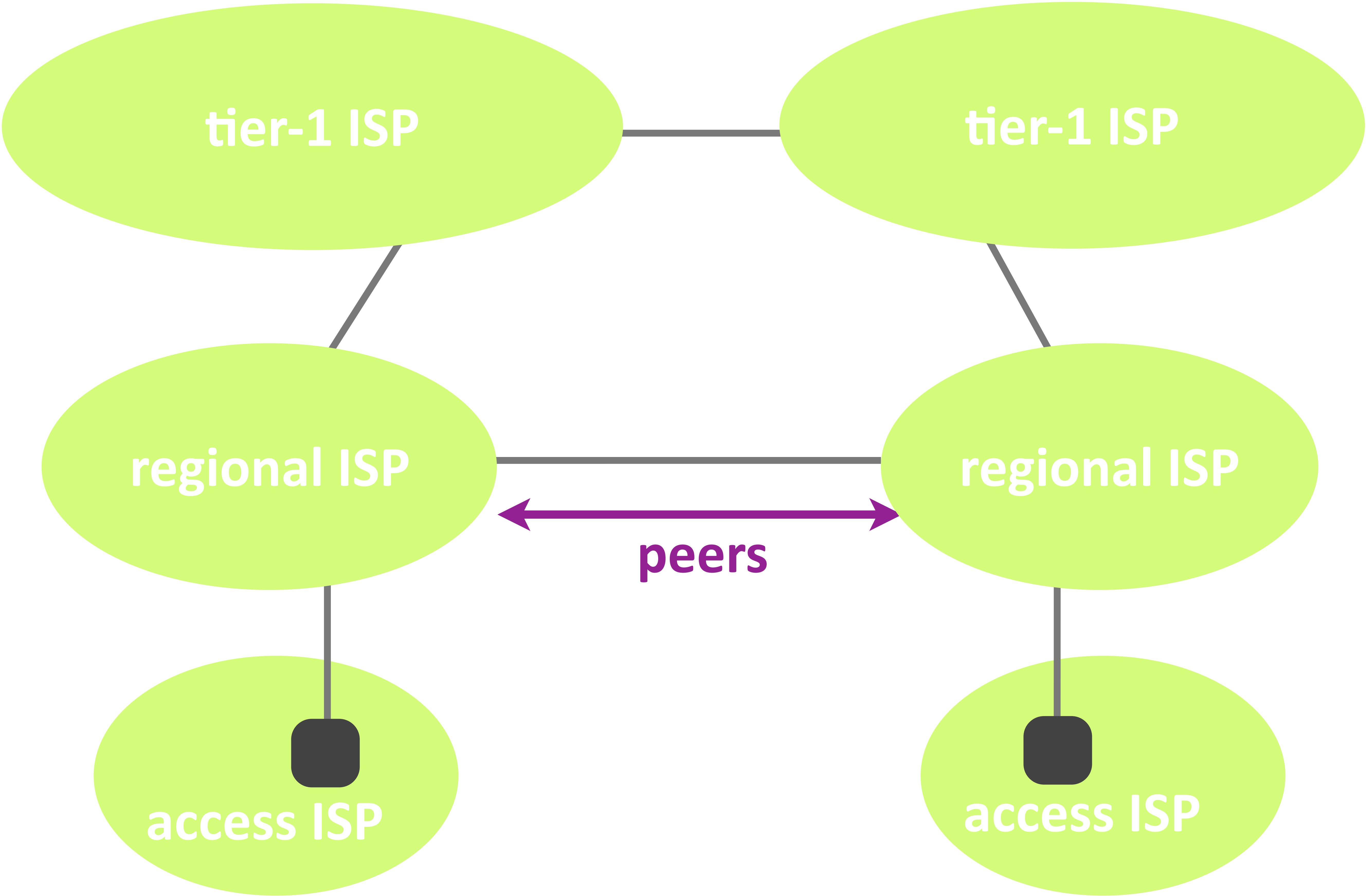


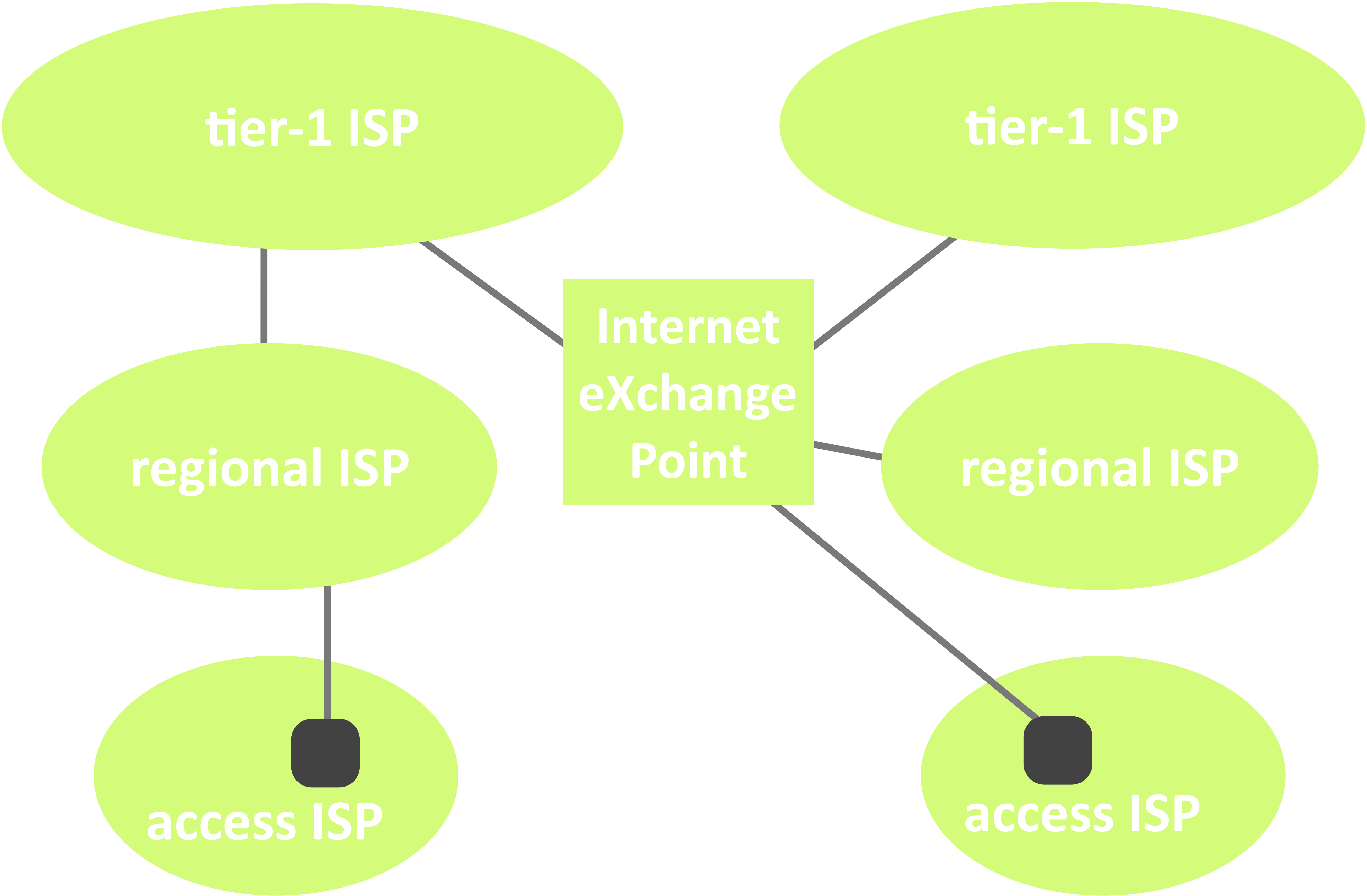

access ISP
access ISP

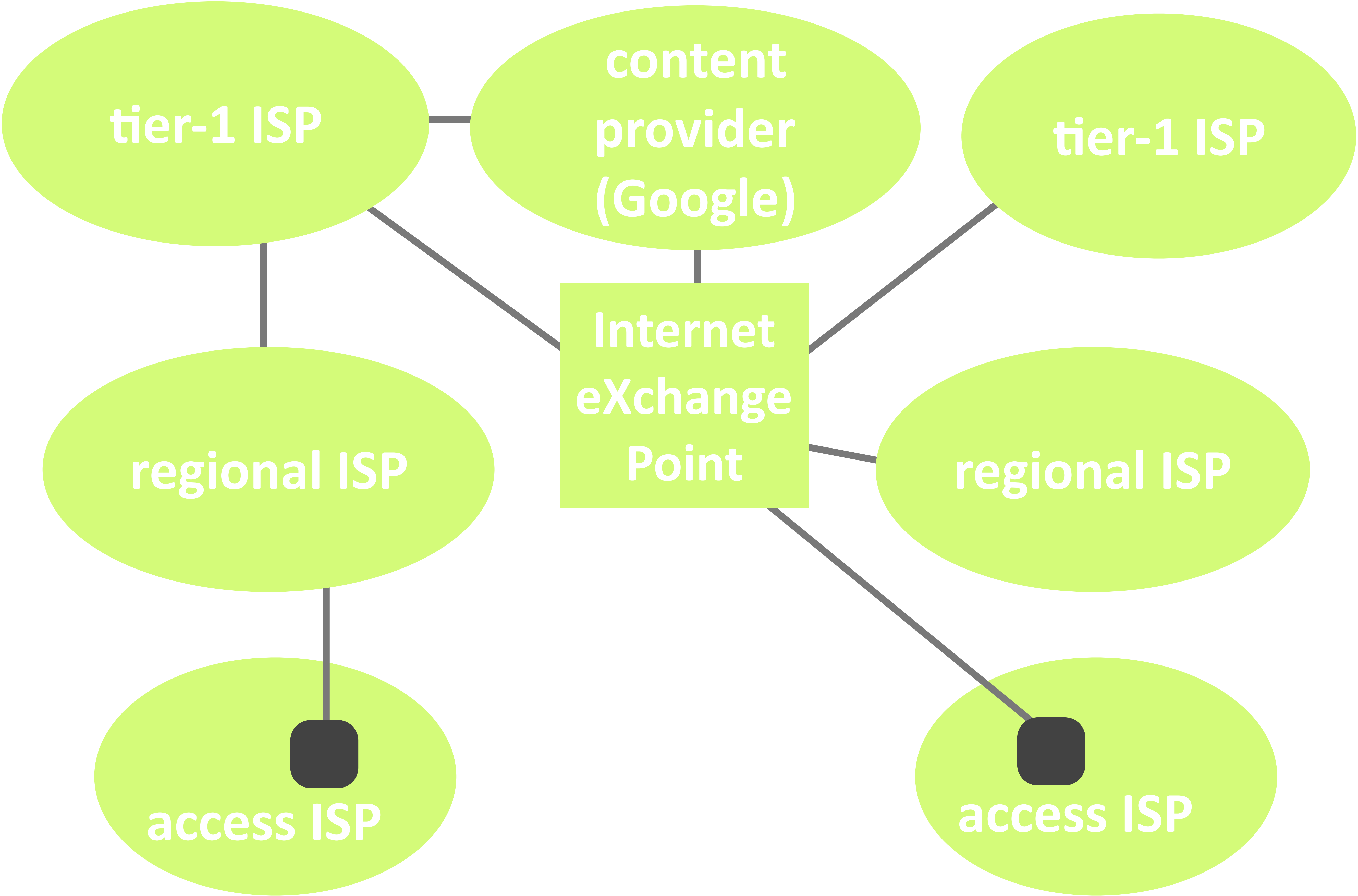

global ISP
provider
customers
access ISP
access ISP

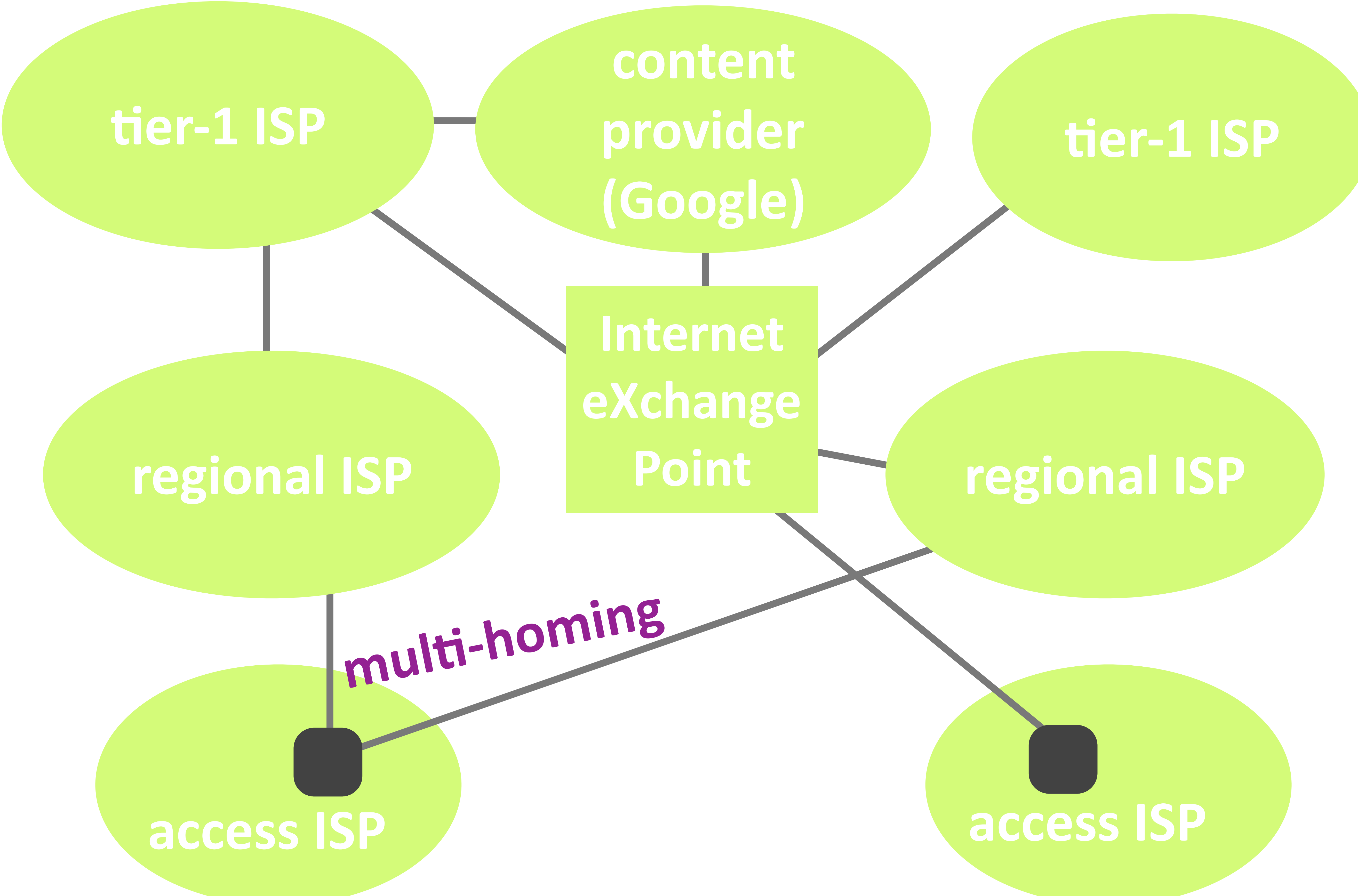

global ISP
global ISP
access ISP
access ISP

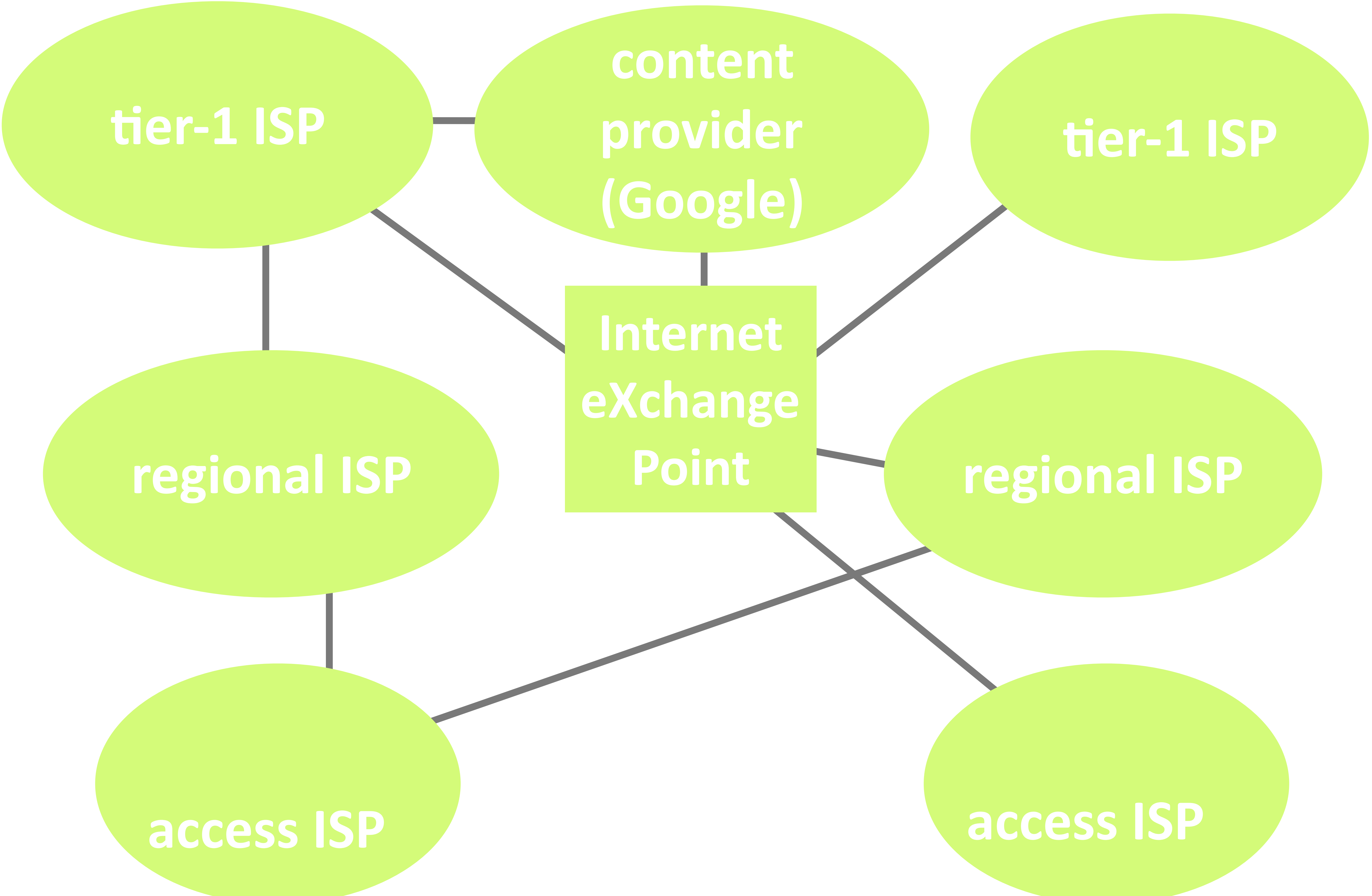

peers
tier-1 ISP
tier-1 ISP
providers
regional ISP
customers
providers
regional ISP
customers
access ISP
access ISP


tier-1 ISP
tier-1 ISP
regional ISP
regional ISP
peers
access ISP
access ISP


tier-1 ISP
tier-1 ISP
Internet eXchange Point
regional ISP
regional ISP
access ISP
access ISP

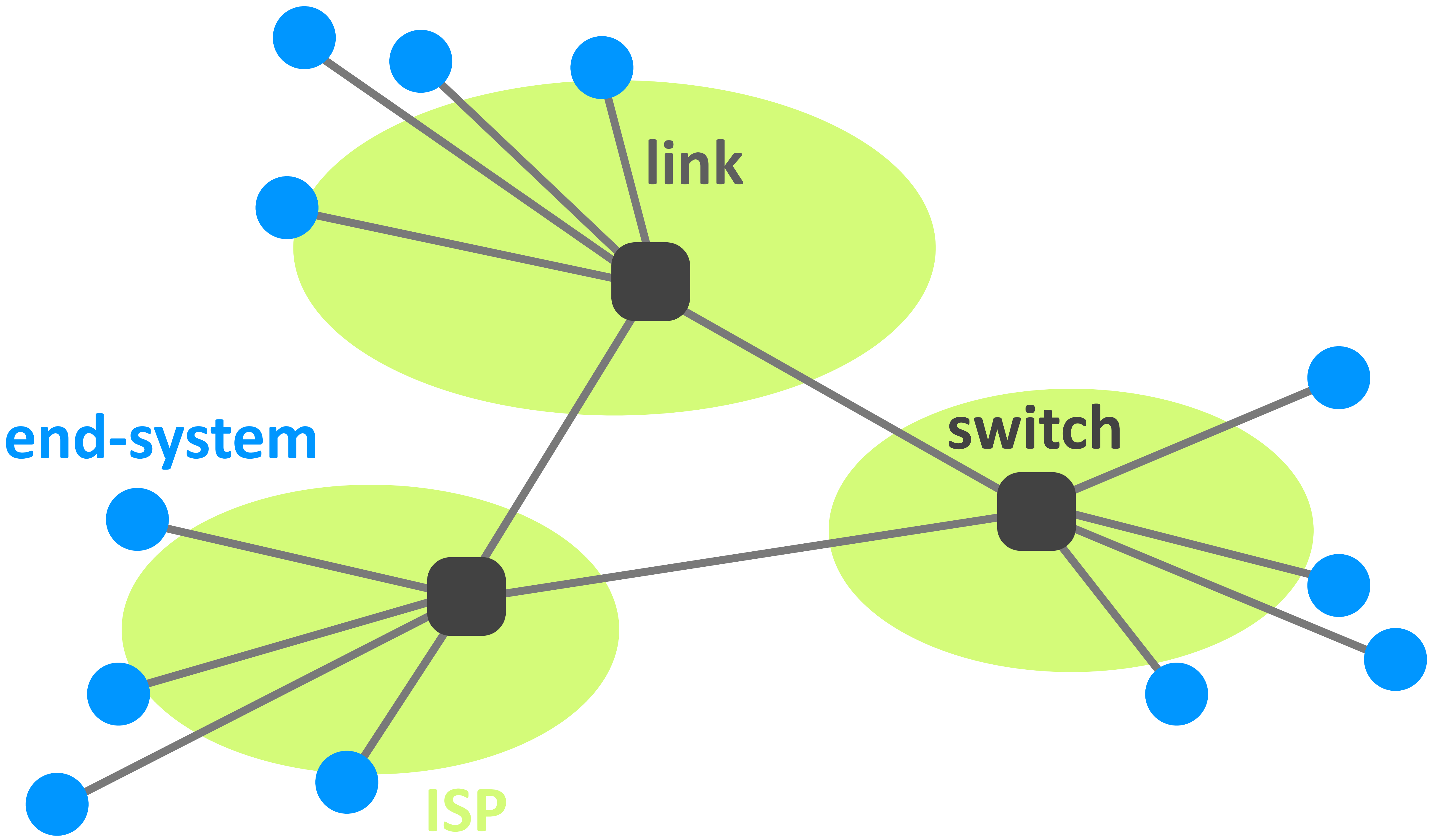

tier-1 ISP
content provider (Google)
tier-1 ISP
Internet eXchange Point
regional ISP
regional ISP
access ISP
access ISP

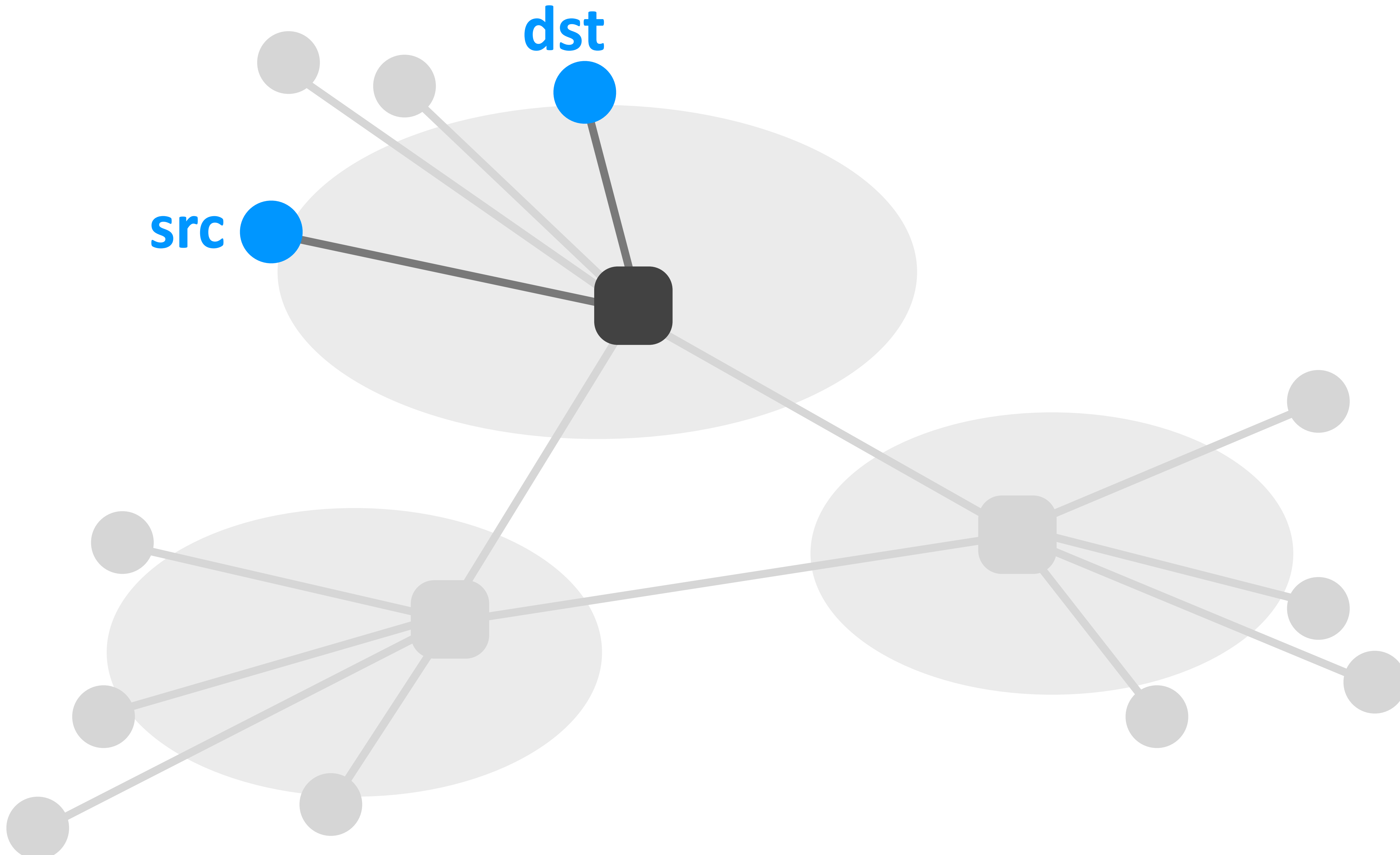

tier-1 ISP
content provider (Google)
tier-1 ISP
Internet eXchange Point
regional ISP
regional ISP
multi-homing
access ISP
access ISP


tier-1 ISP
content provider (Google)
tier-1 ISP
Internet eXchange Point
regional ISP
regional ISP
access ISP
access ISP

Outline

- Links & switches
- ISP relationships
- Performance metrics
- Layers


link
end-system
switch
ISP

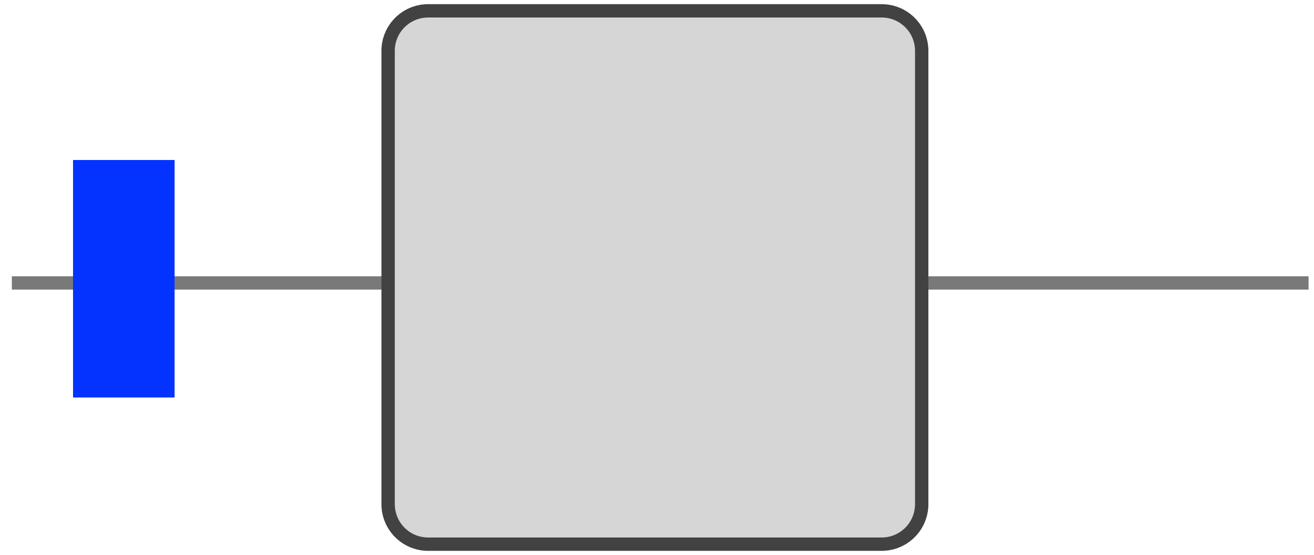

dst
src

Loss

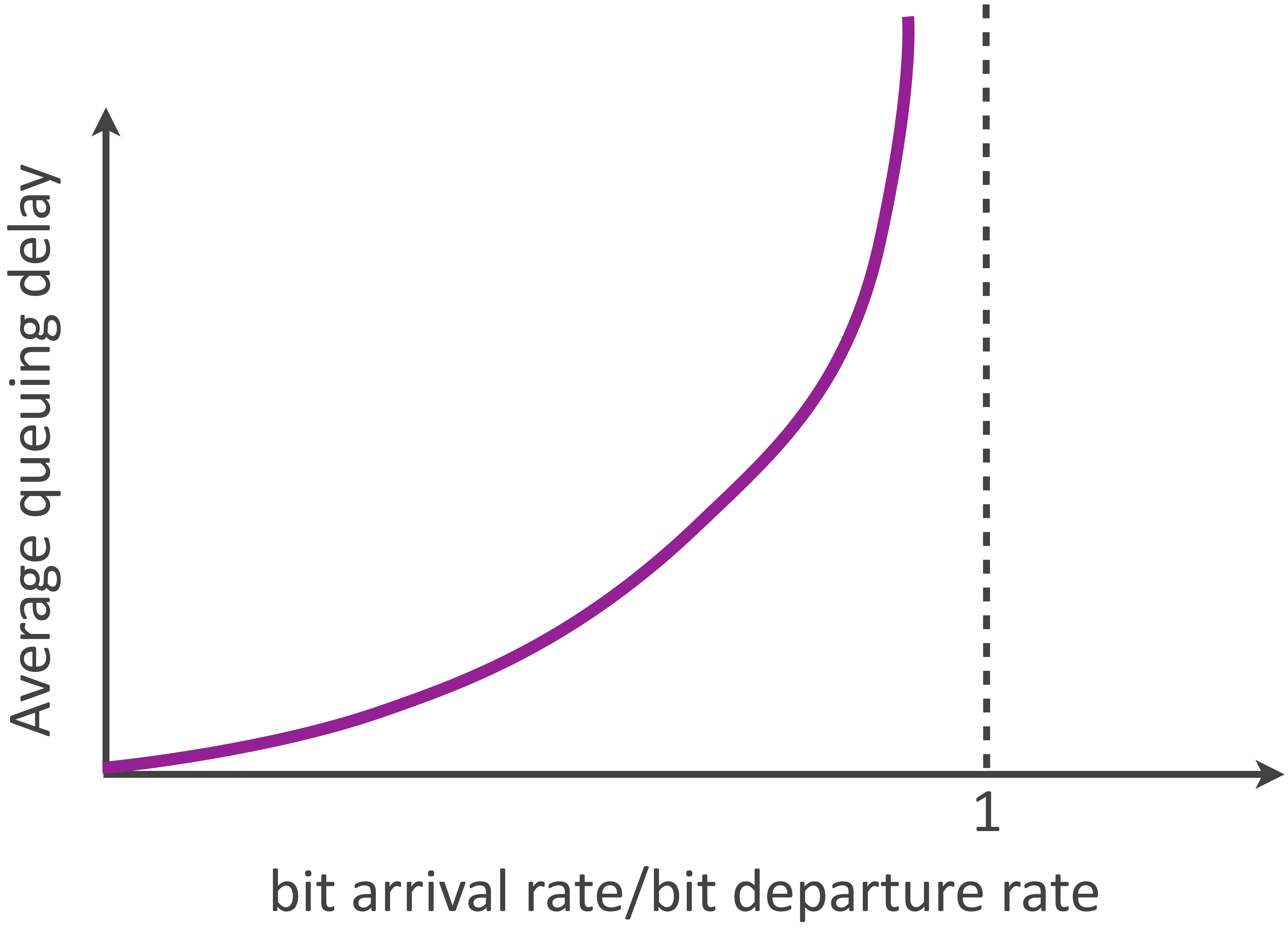
- What fraction of the packets sent from a source to a destination are dropped?

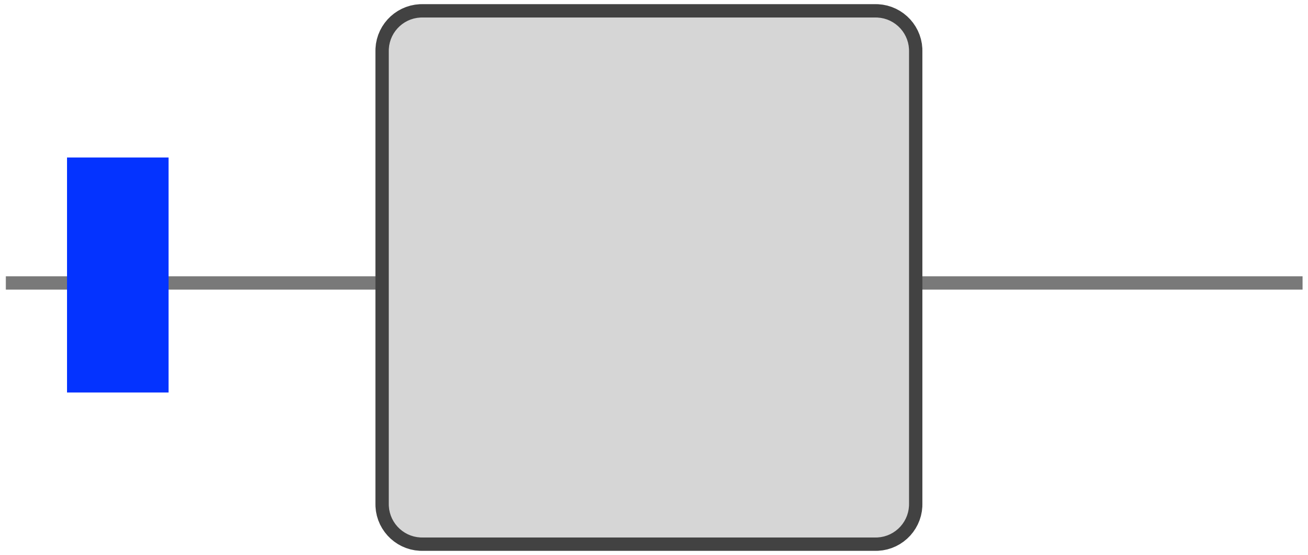
Delay

- How long does it take to send a packet from its source to its destination?

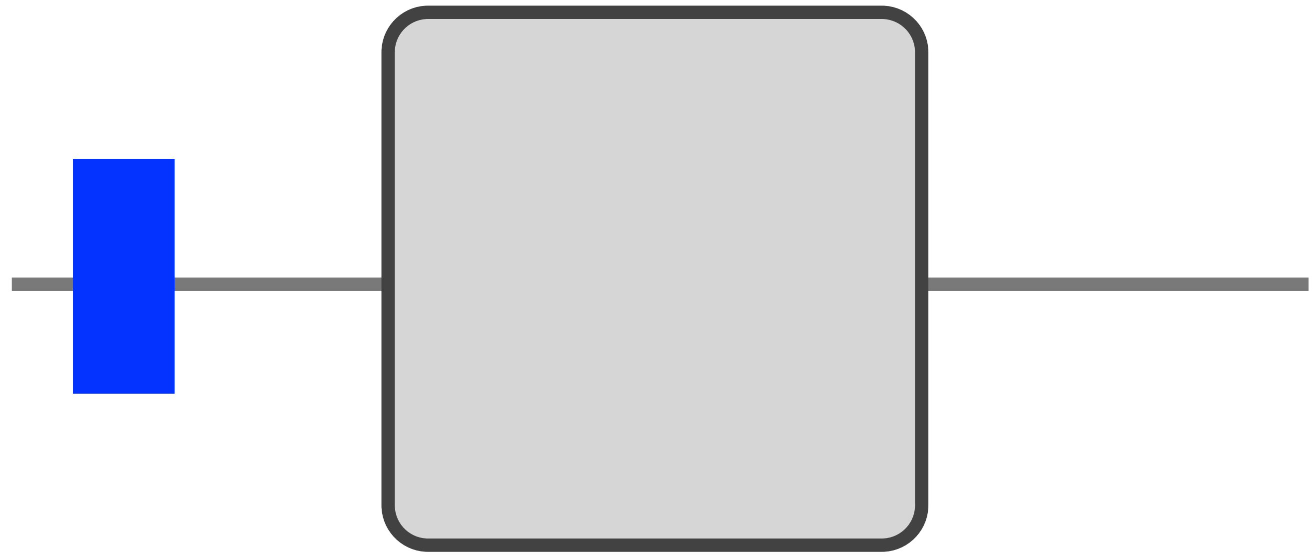


bit arrival rate > bit departure rate


Average queuing delay
1
bit arrival rate/bit departure rate



bit arrival rate $<=$ bit departure rate



0 msec

1 msec

2 msec

3 msec

bit arrival rate <= bit departure rate

Queuing delay

- Approaches infinity,
  if arrival rate > departure rate
  - *assuming an infinite buffer*
- Depends on burst size, otherwise

Throughput

- At what rate is the destination receiving data from the source?

Average throughput

- Data size / Transfer time
  - *downloaded 100 Mbits in 1 second*
  - *average throughput = 100 Mbits/sec*

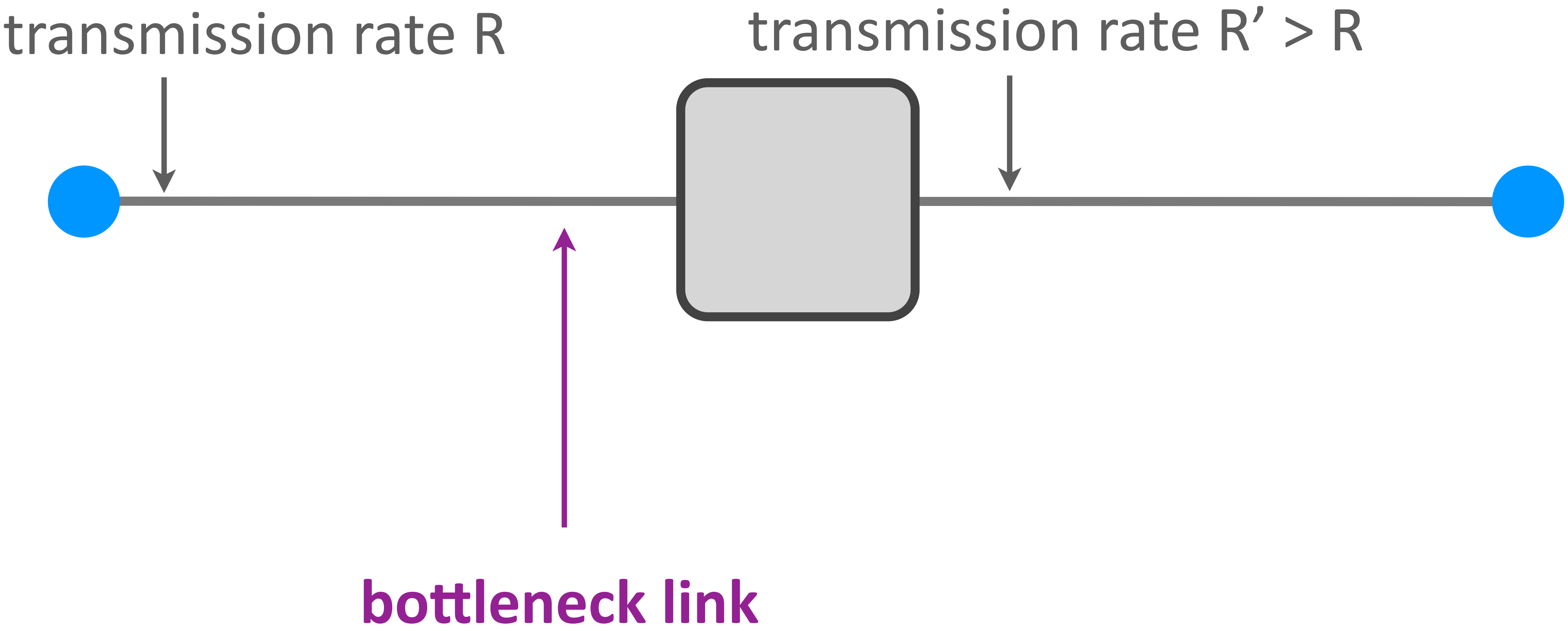
transmission rate R bits/sec

Source sends large file of size F bits to destination

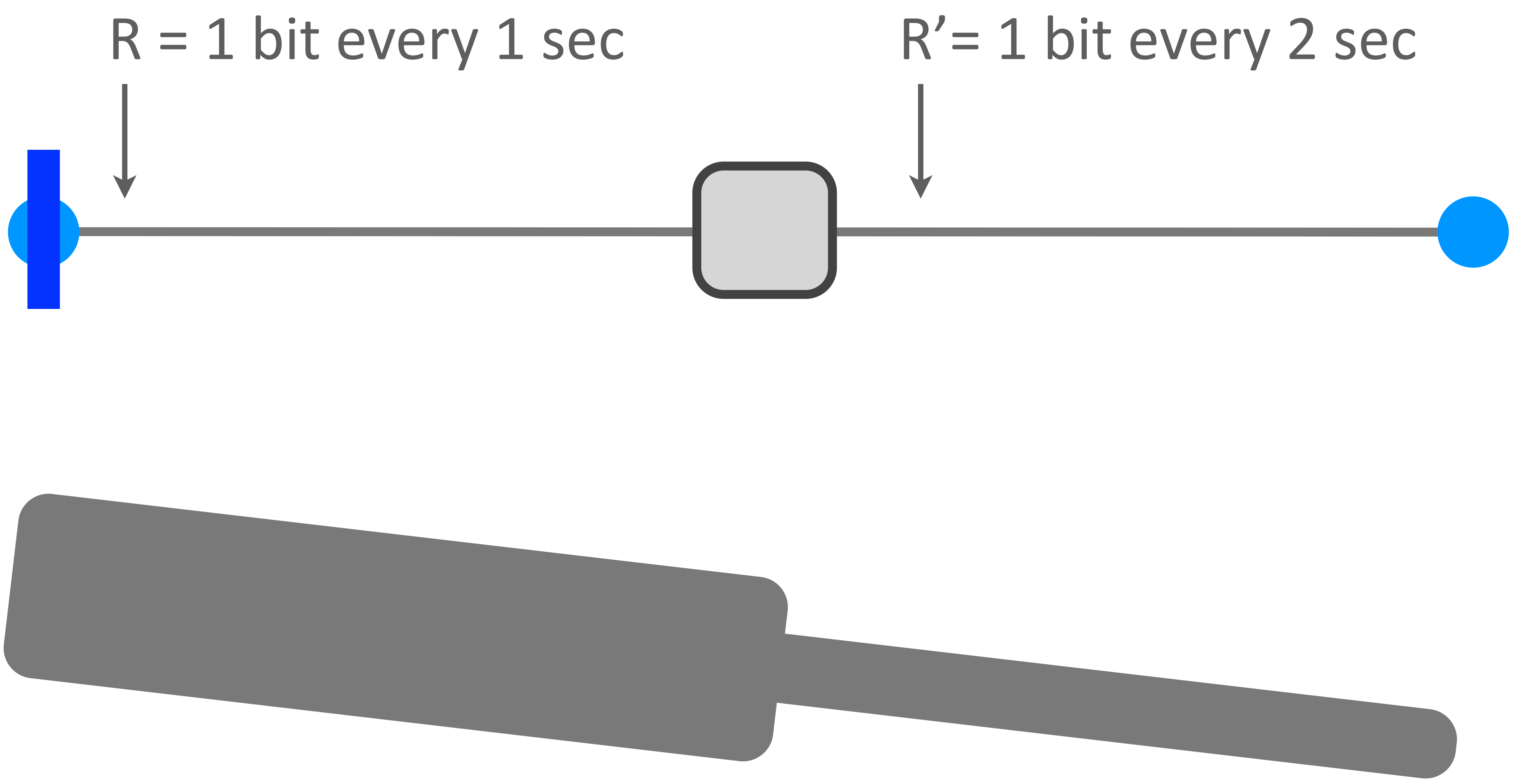
Amount of data = F bits

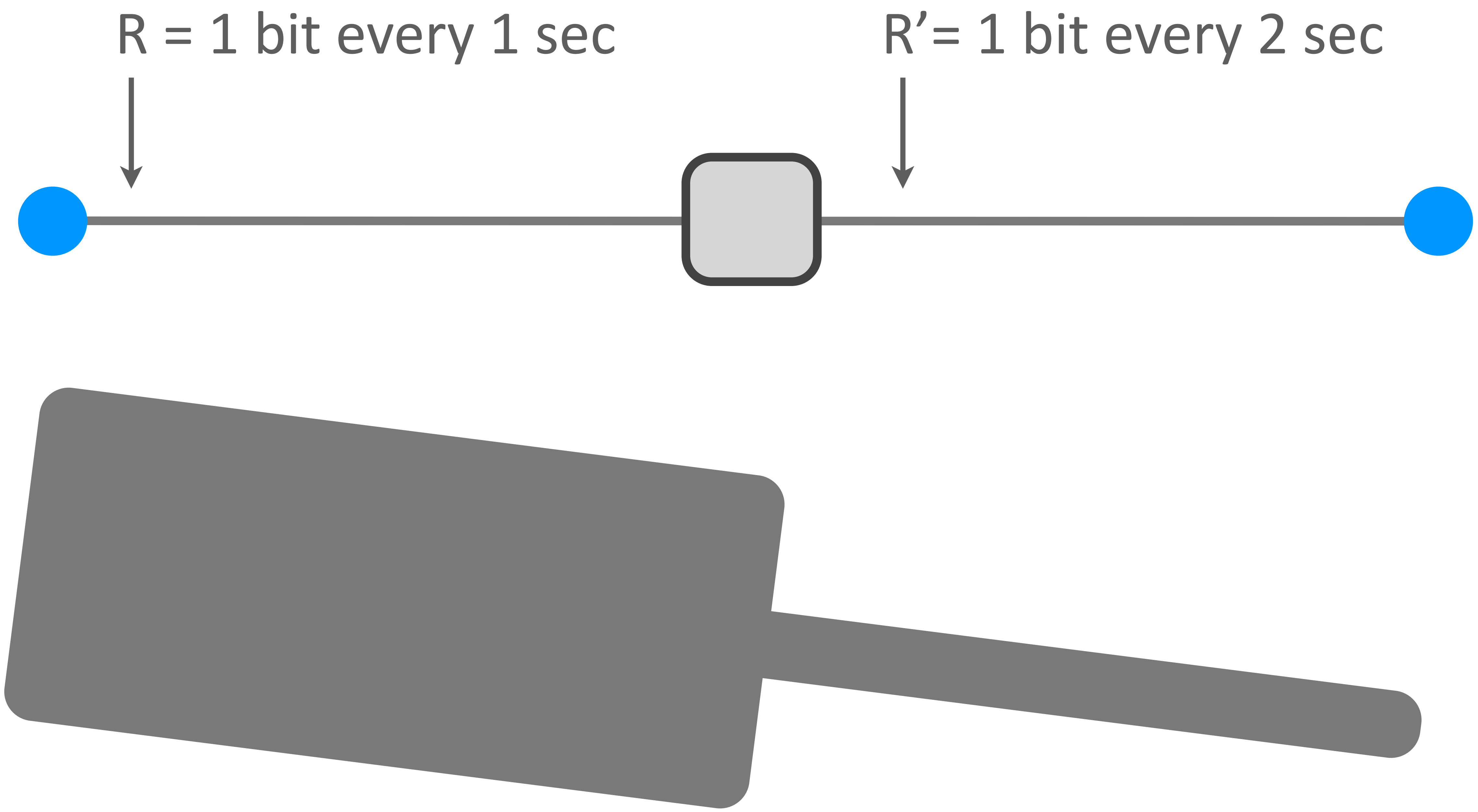
Transfer time = F/R sec

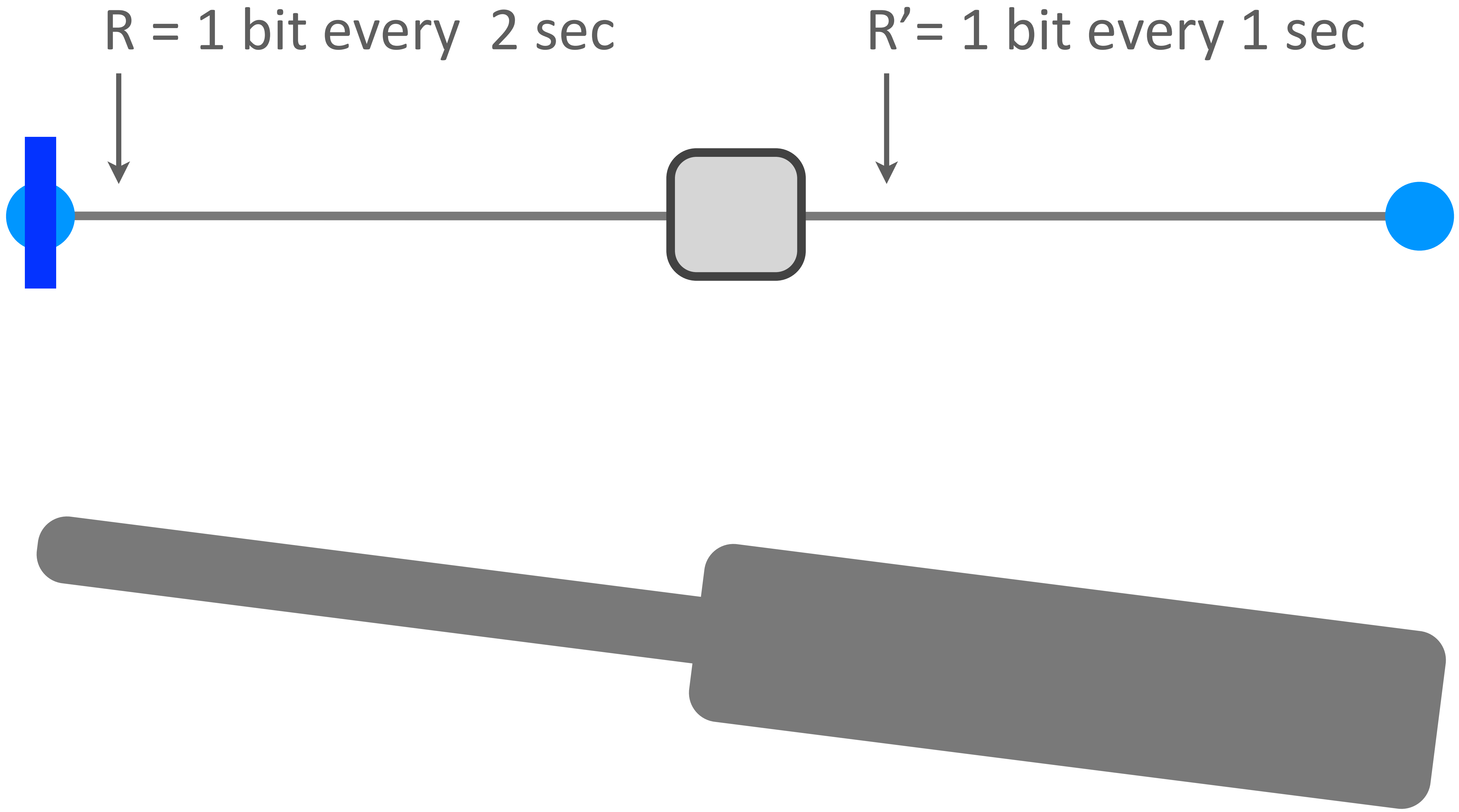
Average throughput = R bits/sec

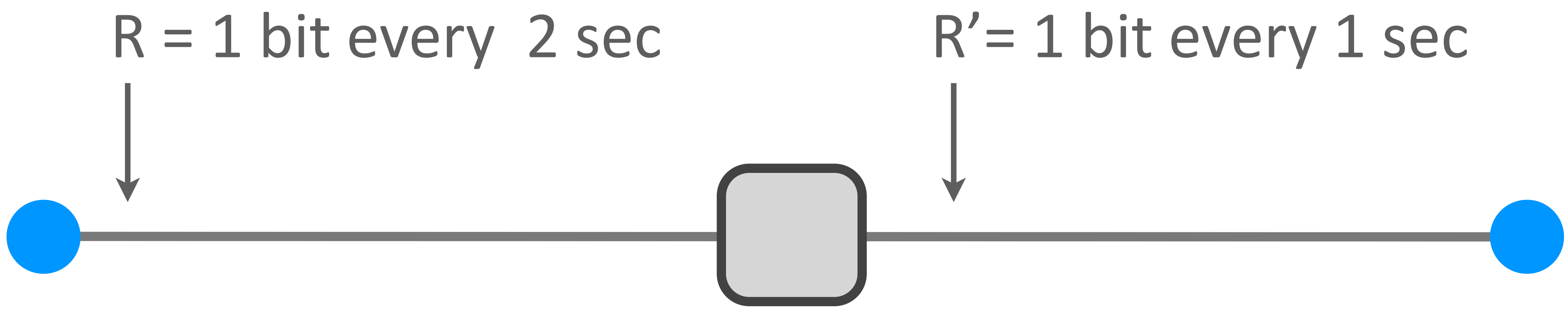


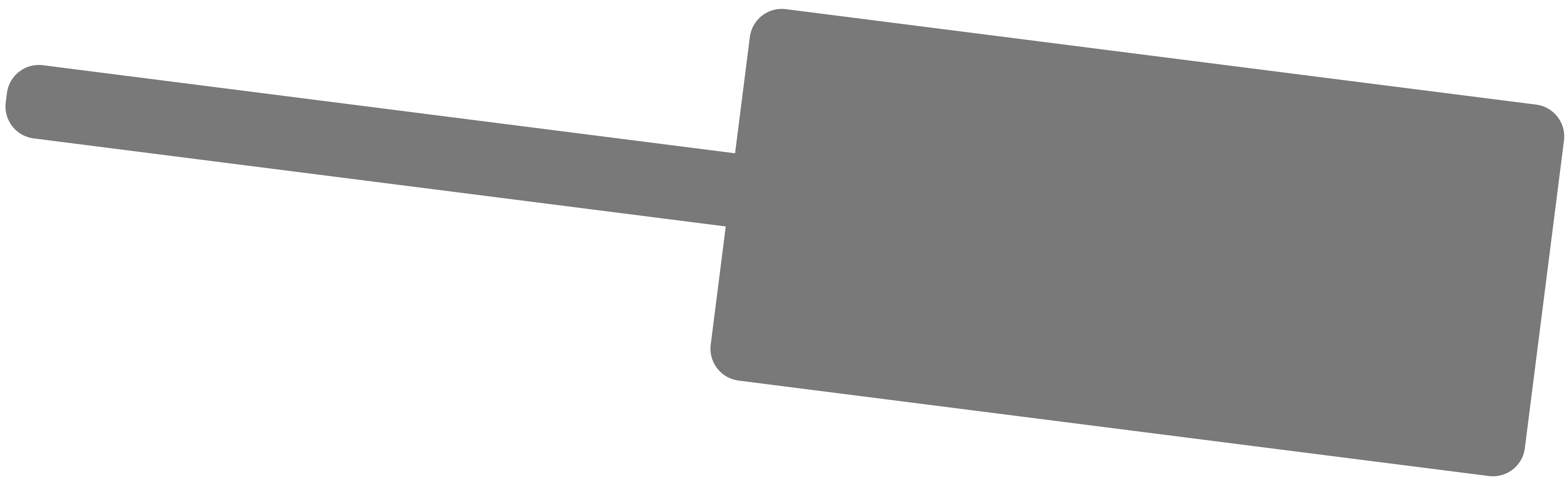
Average throughput = $\min\{R, R'\} = R$

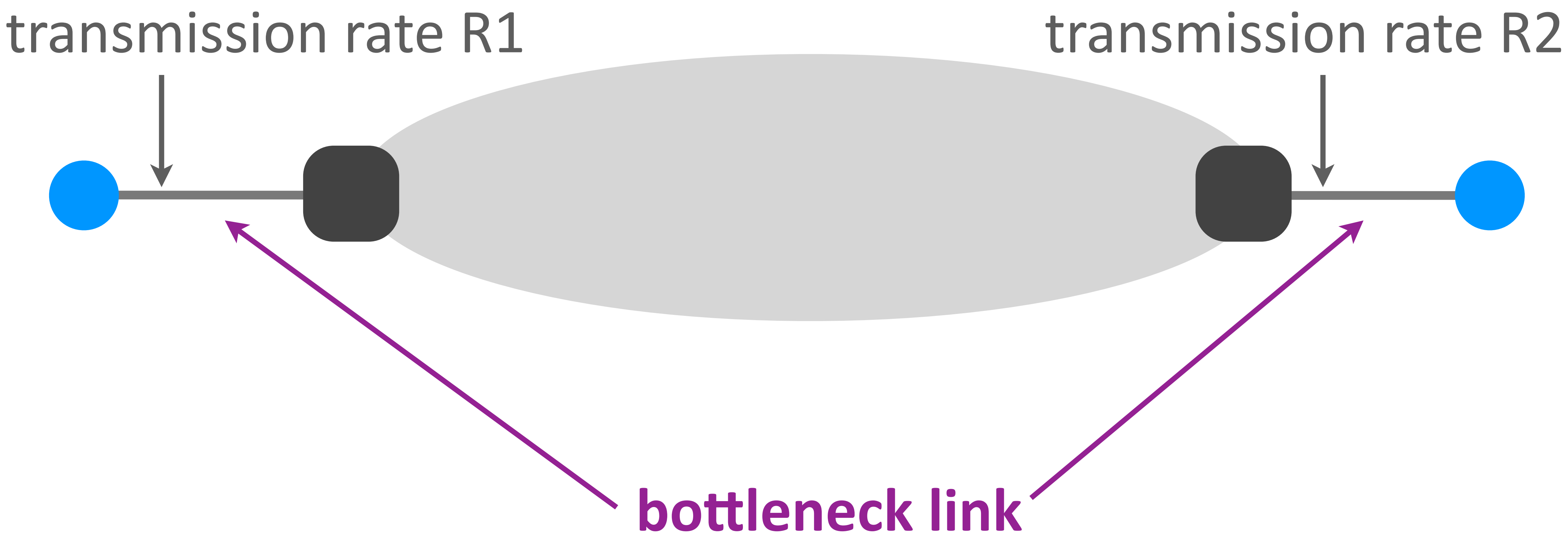

R = 1 bit every 1 sec
R’= 1 bit every 2 sec

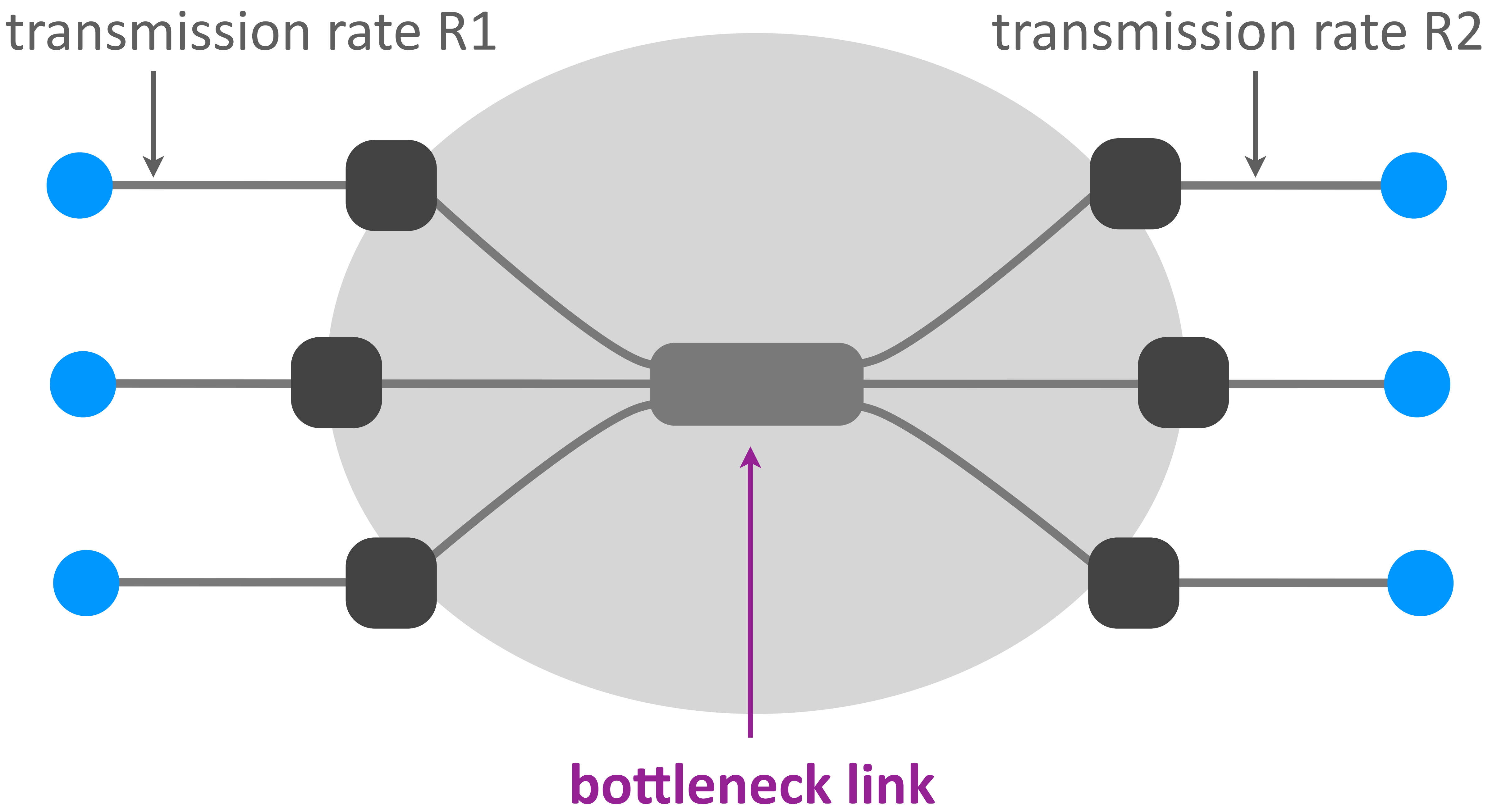

R = 1 bit every 1 sec
R’= 1 bit every 2 sec


R = 1 bit every 2 sec
R’= 1 bit every 1 sec


R = 1 bit every 2 sec
R’= 1 bit every 1 sec



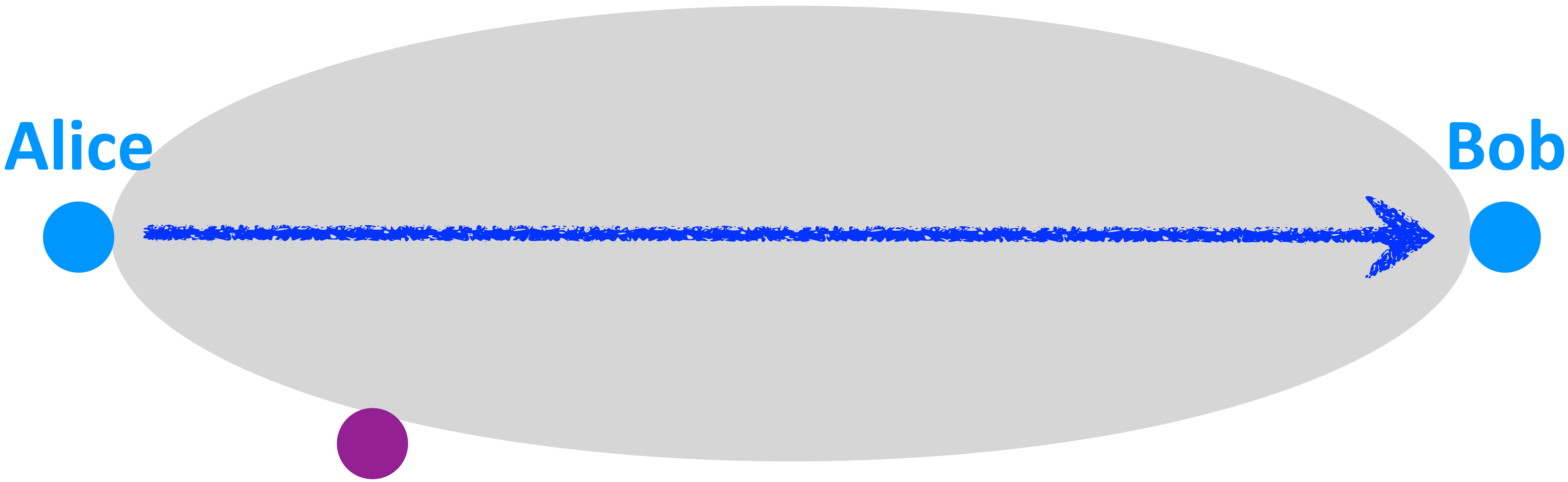

transmission rate R1
transmission rate R2
bottleneck link



transmission rate R1
transmission rate R2
bottleneck link

Security

- How does the network react to adversarial (= bad) behavior?
- What does the network assume about the behavior of end-systems and packet switches?

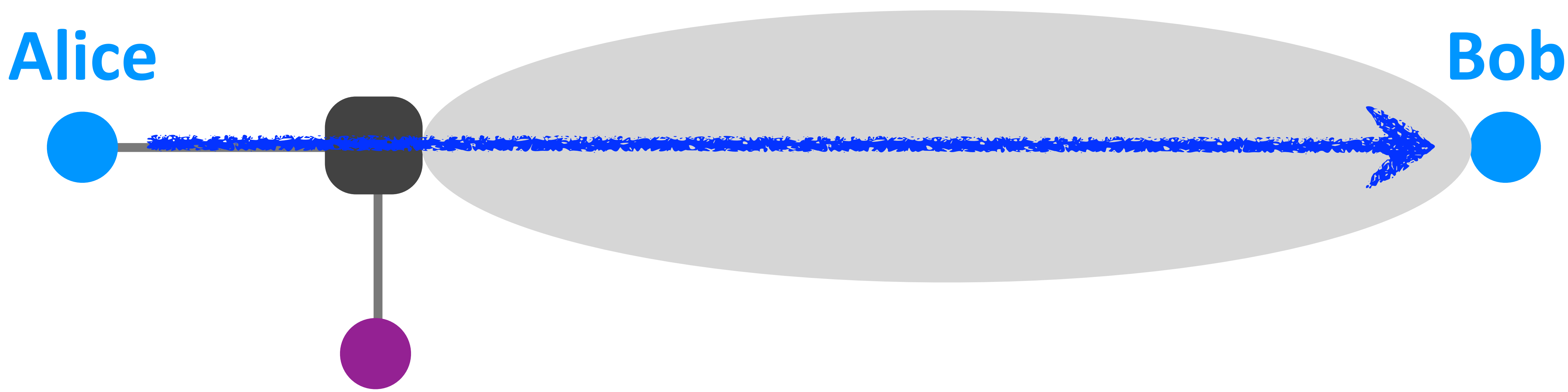


Eve (the eavesdropper)

tries to listen in on the communication
to obtain copies of the data

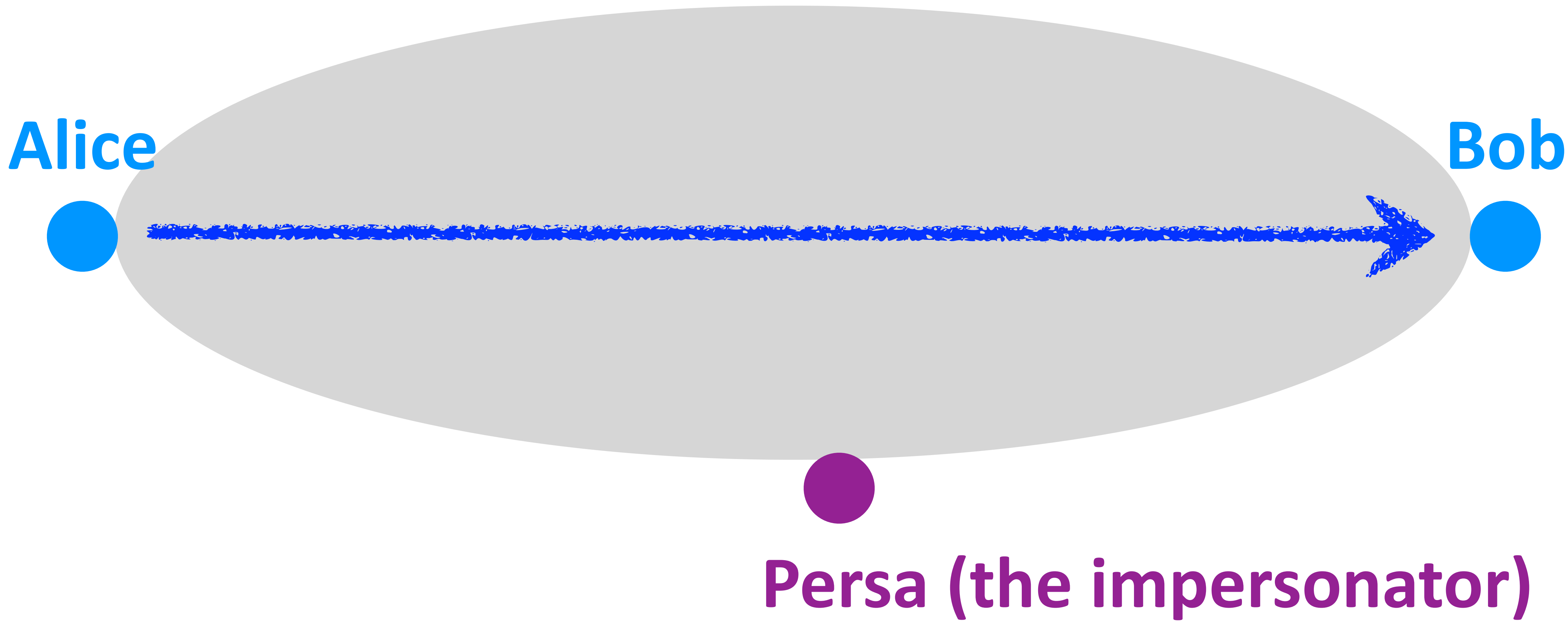
Eve (the eavesdropper)

tries to listen in on the communication
to obtain copies of the data

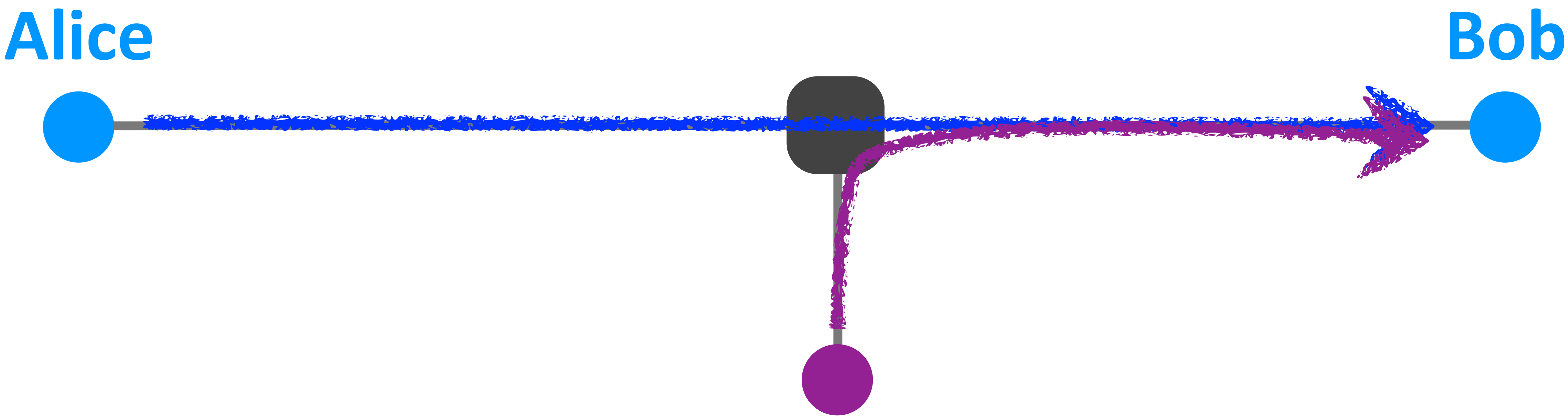


Eve (the eavesdropper)

tries to listen in on the communication
to obtain copies of the data

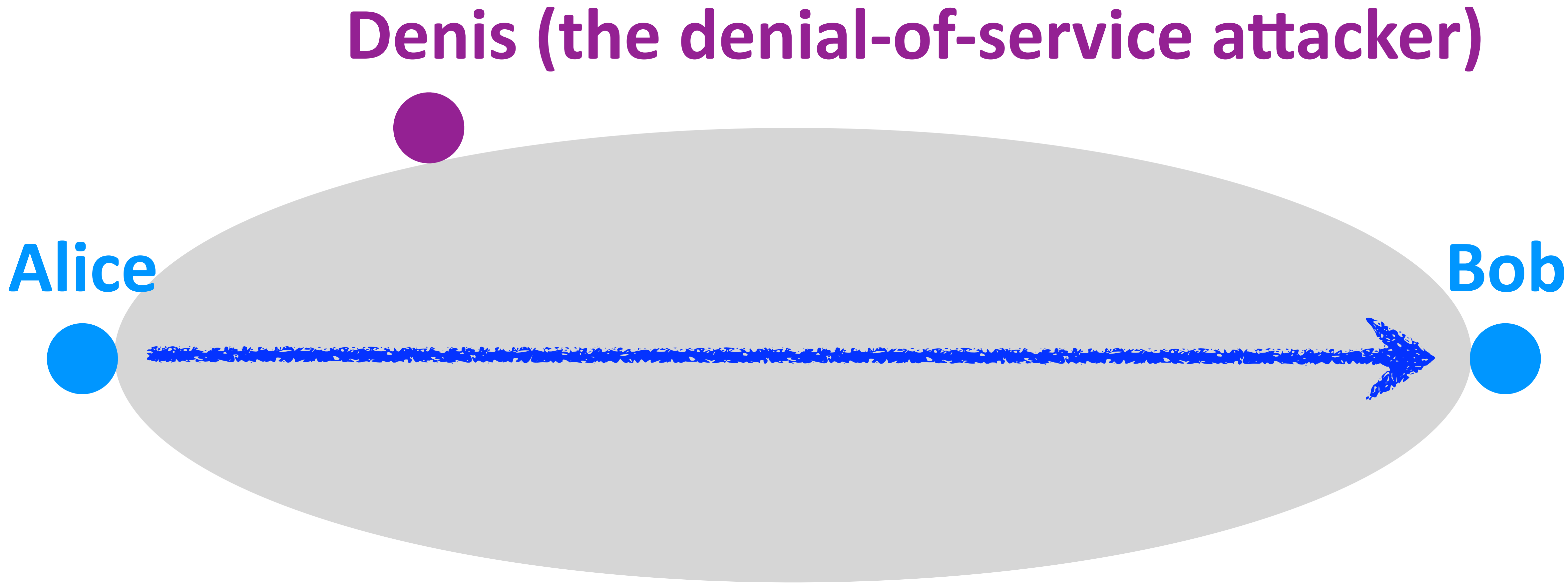


pretends she is Alice to extract information from Bob

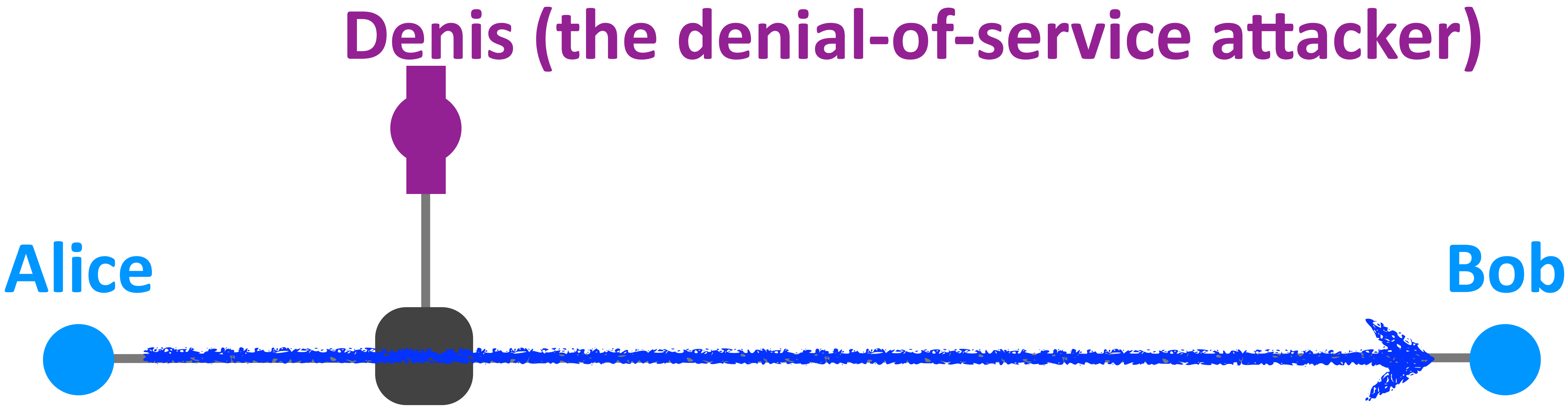


pretends she is Alice to extract information from Bob

makes Alice or Bob crash or disconnect
and disrupts the communication



makes Alice or Bob crash or disconnect
and disrupts the communication



vulnerability attack

makes Alice or Bob crash or disconnect
and disrupts the communication

vulnerability attack

bandwidth flooding

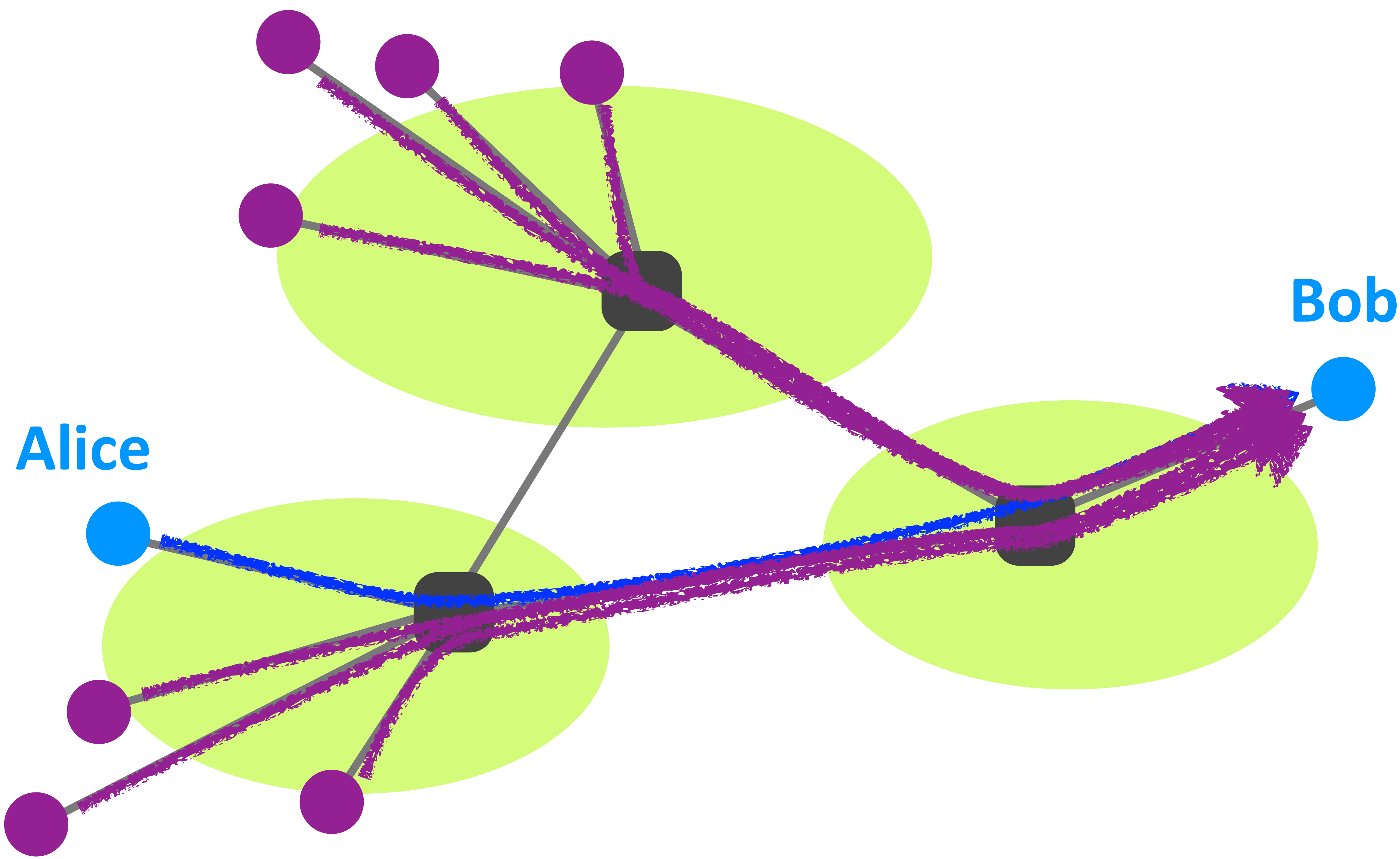
makes Alice or Bob crash or disconnect
and disrupts the communication

vulnerability attack

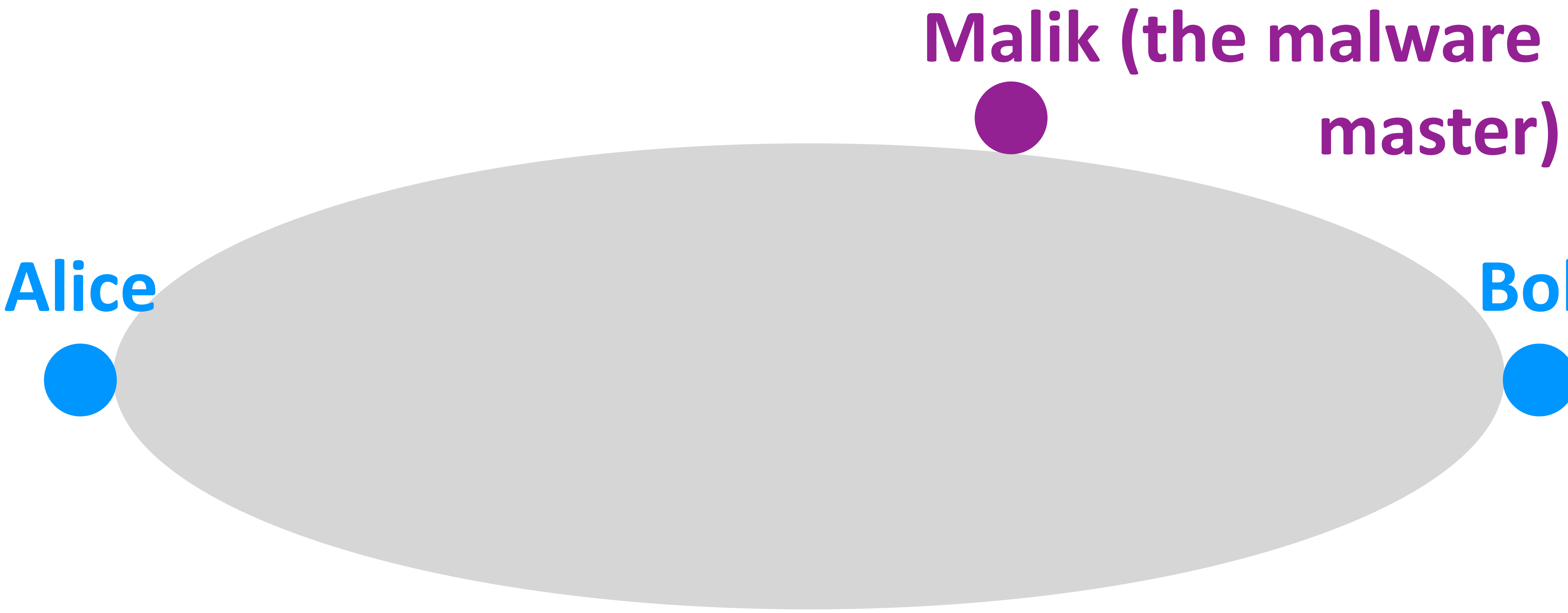
bandwidth flooding

connection flooding

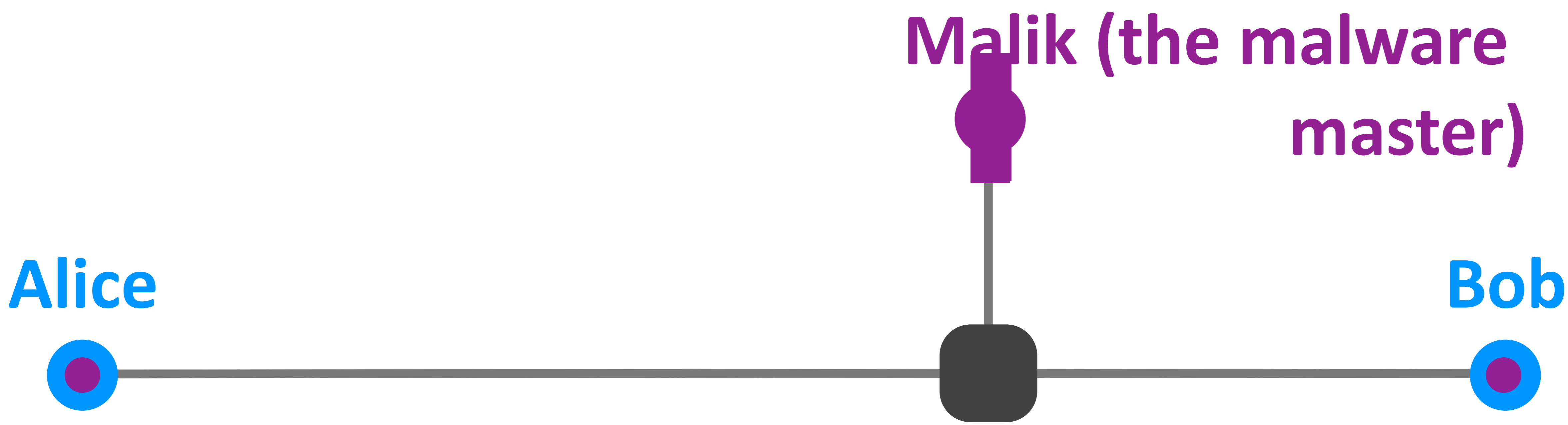
distributed denial-of-service attack



infects Alice or Bob with malware = bad software



infects Alice or Bob with
malware = bad software



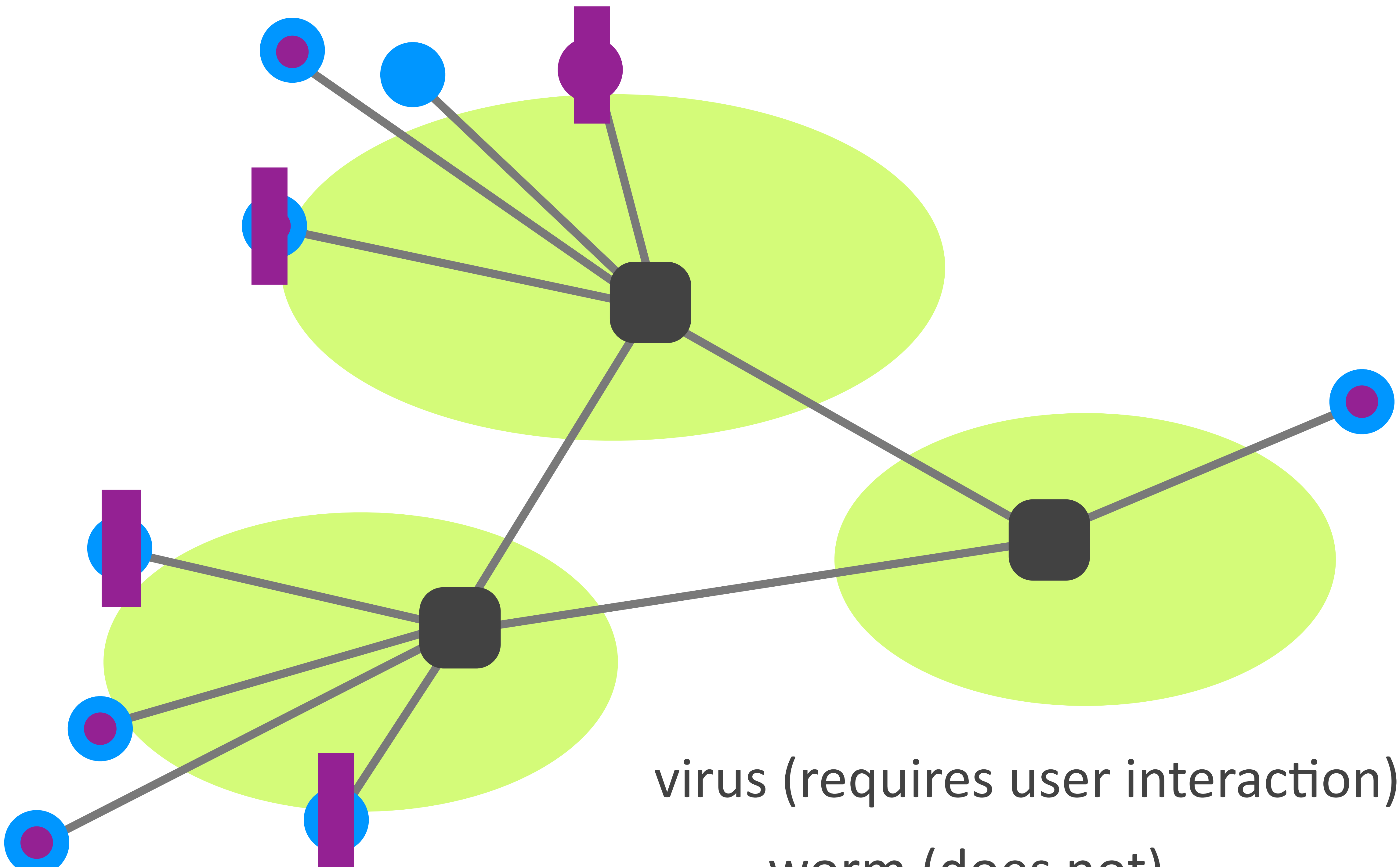
delete files

copy & export personal data

send spam email

launch denial of service

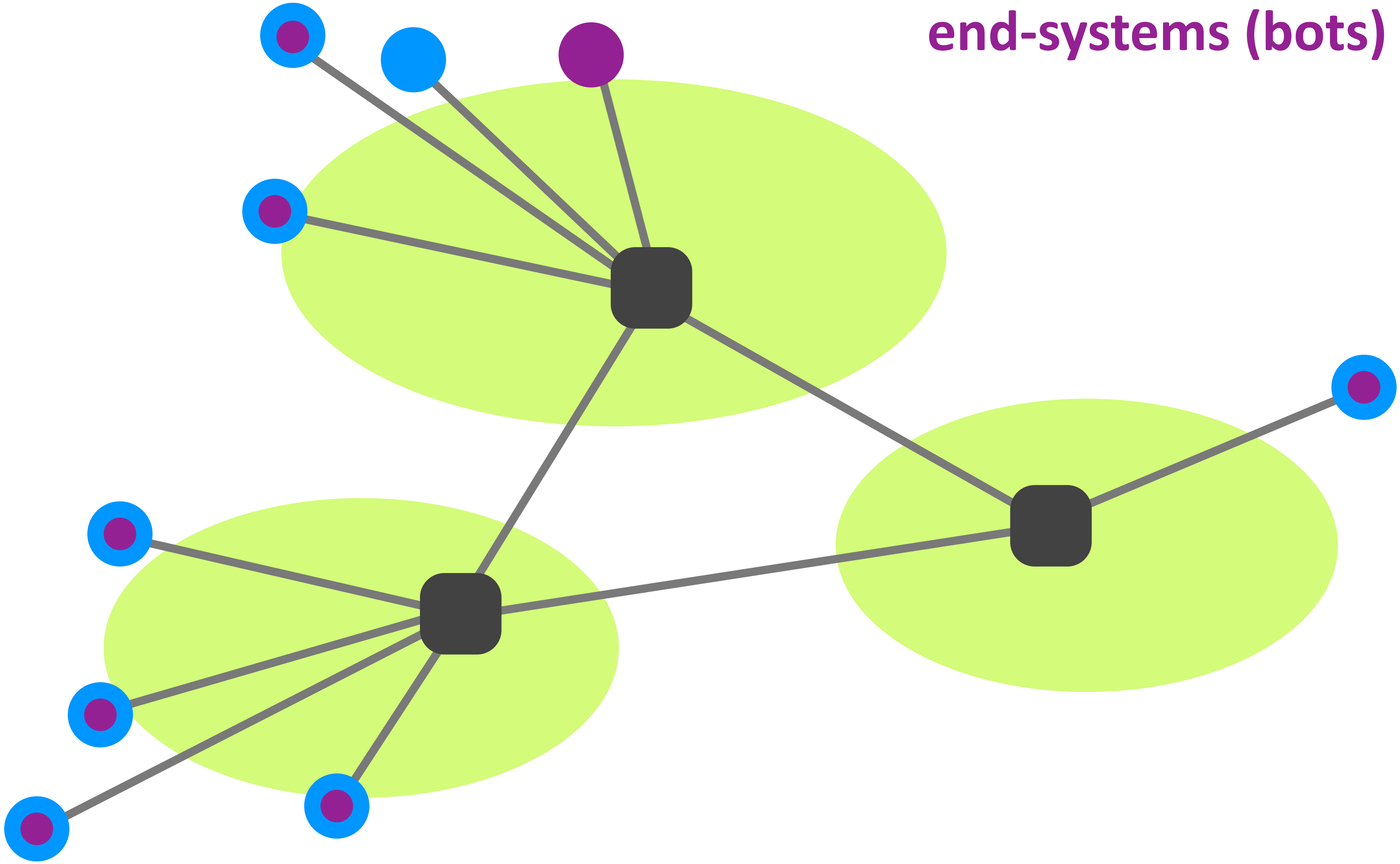
self-propagating malware



virus (requires user interaction)

worm (does not)

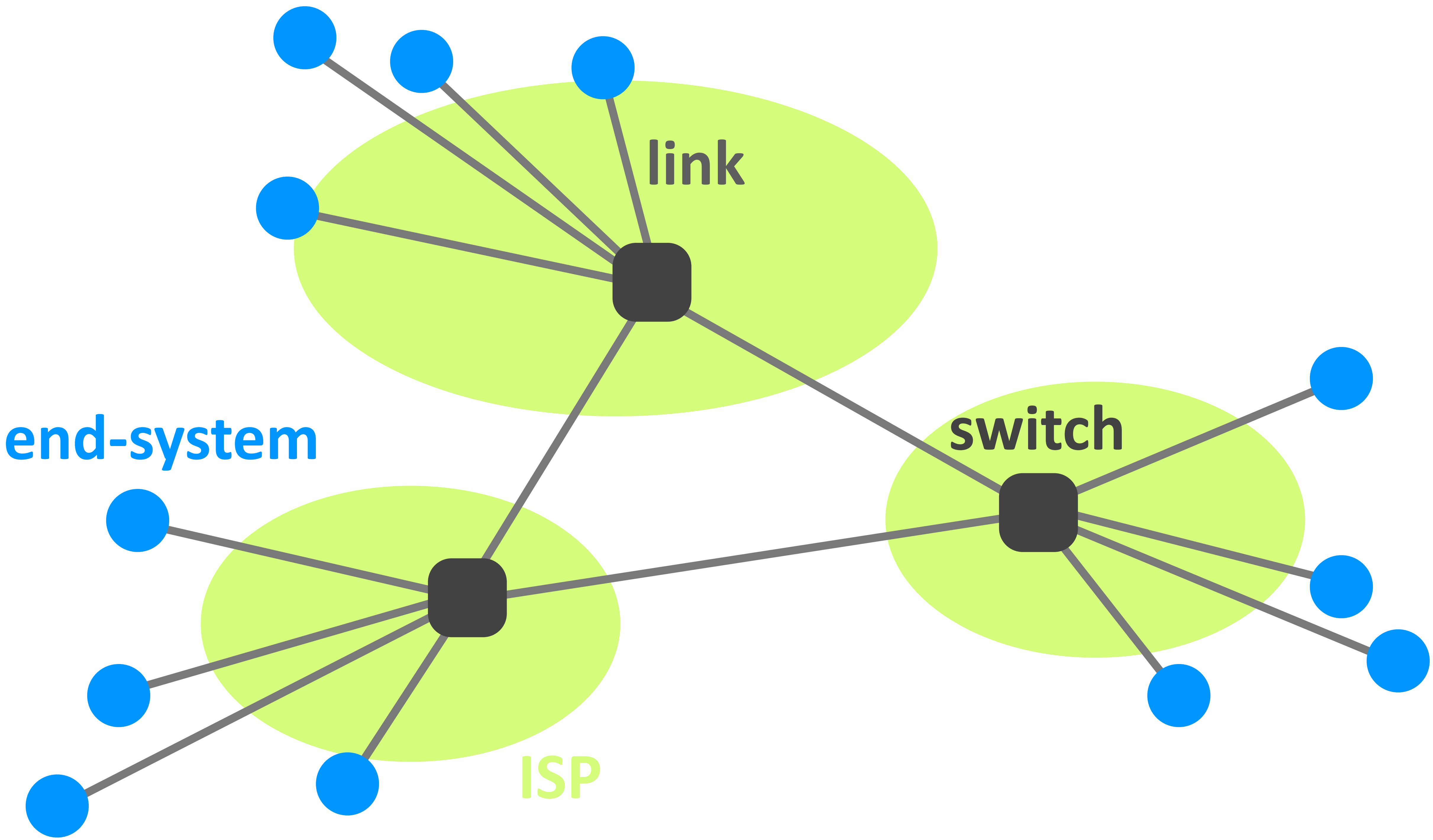
botnet =
army of compromised end-systems (bots)

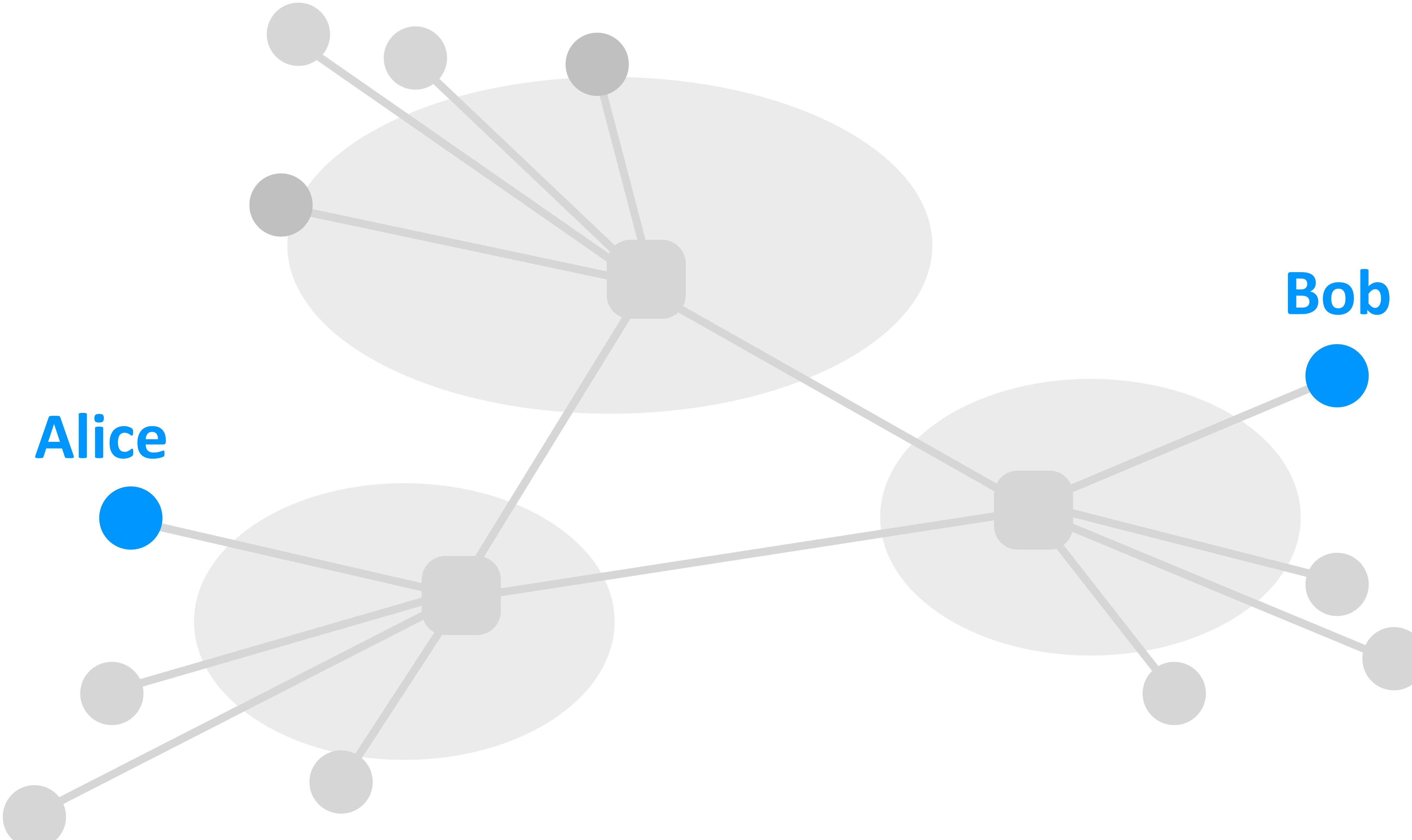



Denis
Malik
Alice
Bob
Eve
Persa

Outline

- Links & switches
- ISP relationships
- Performance metrics
- Layers


link
end-system
switch
ISP


Bob
Alice

Alice **Bob**

mail box *mail box*

local post office **local post office**

mail bag *mail bag*

central post office **central post office**

Layers

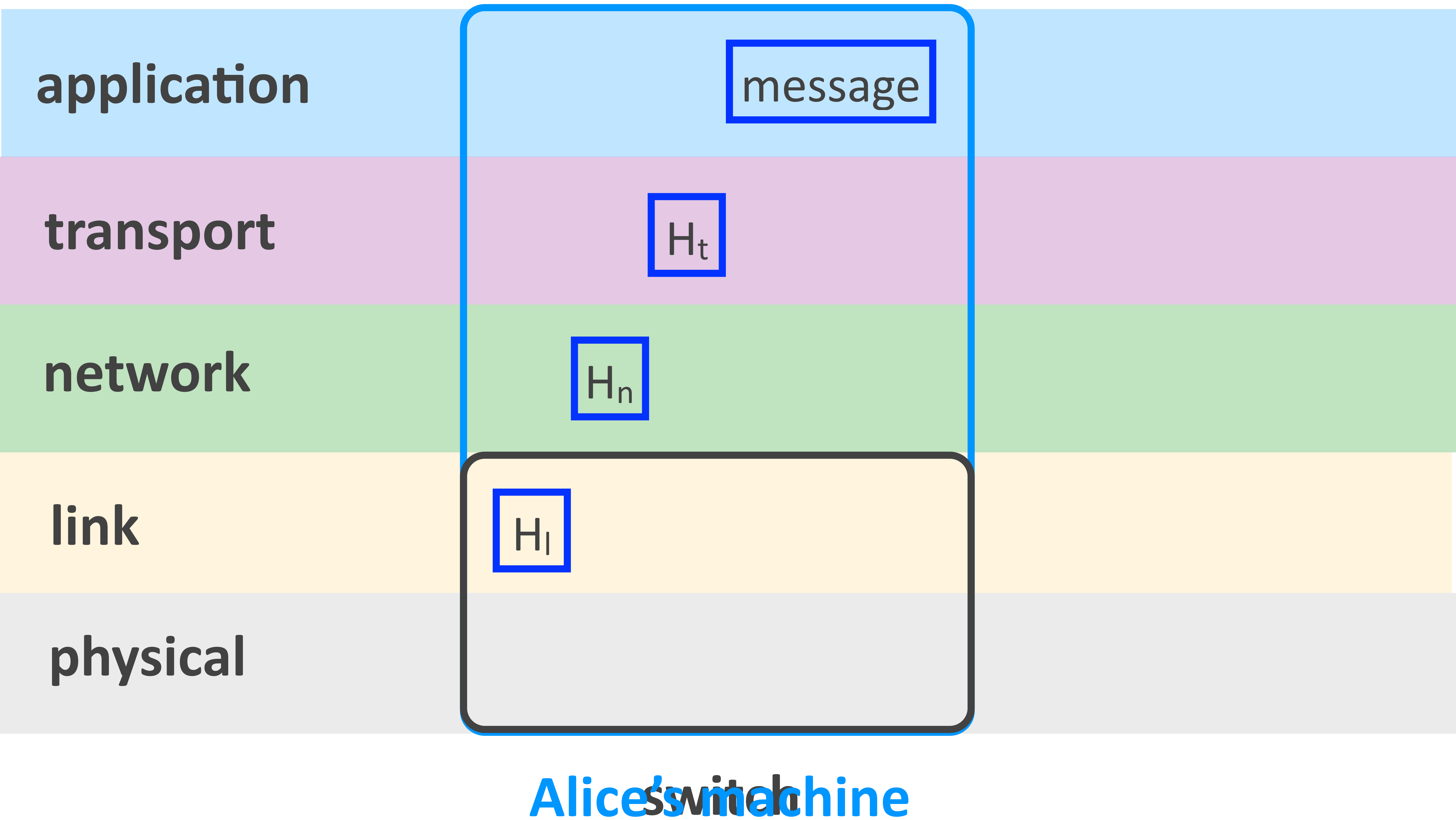
- Layer = a part of a system with
  well-defined interfaces to other parts

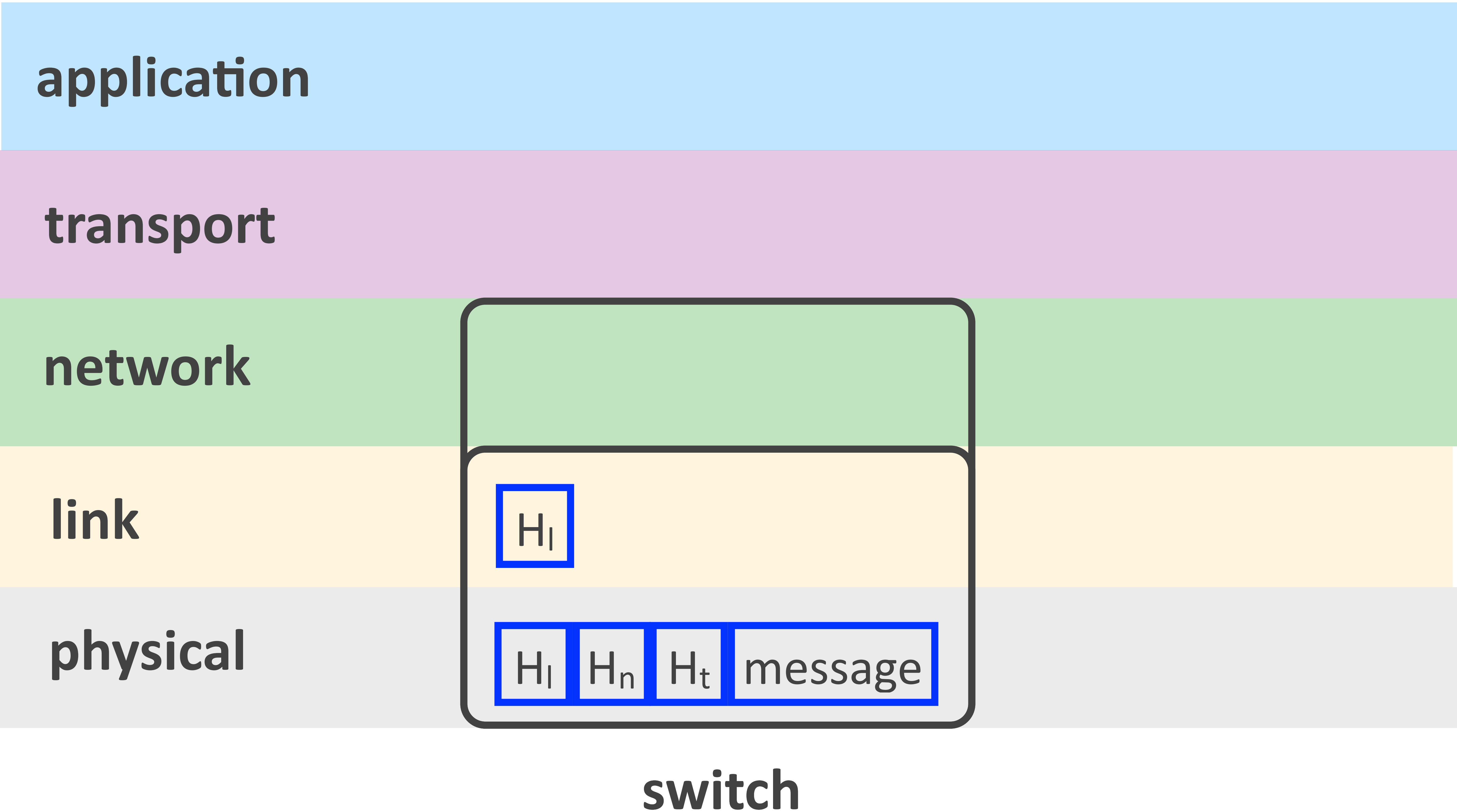
- Two layers interact only
  through the interface between them

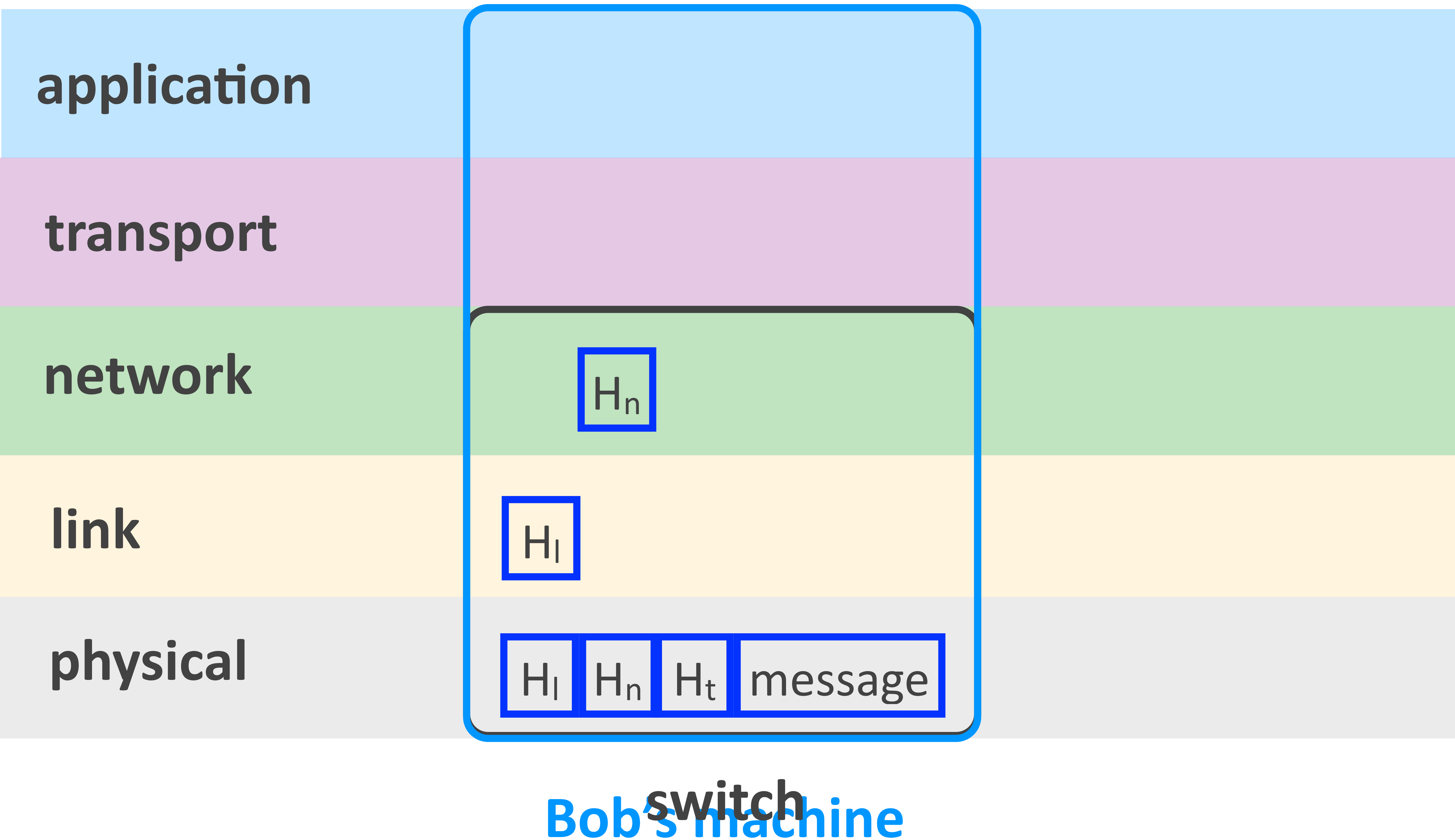
- One layer interacts only
  with layer above and layer below

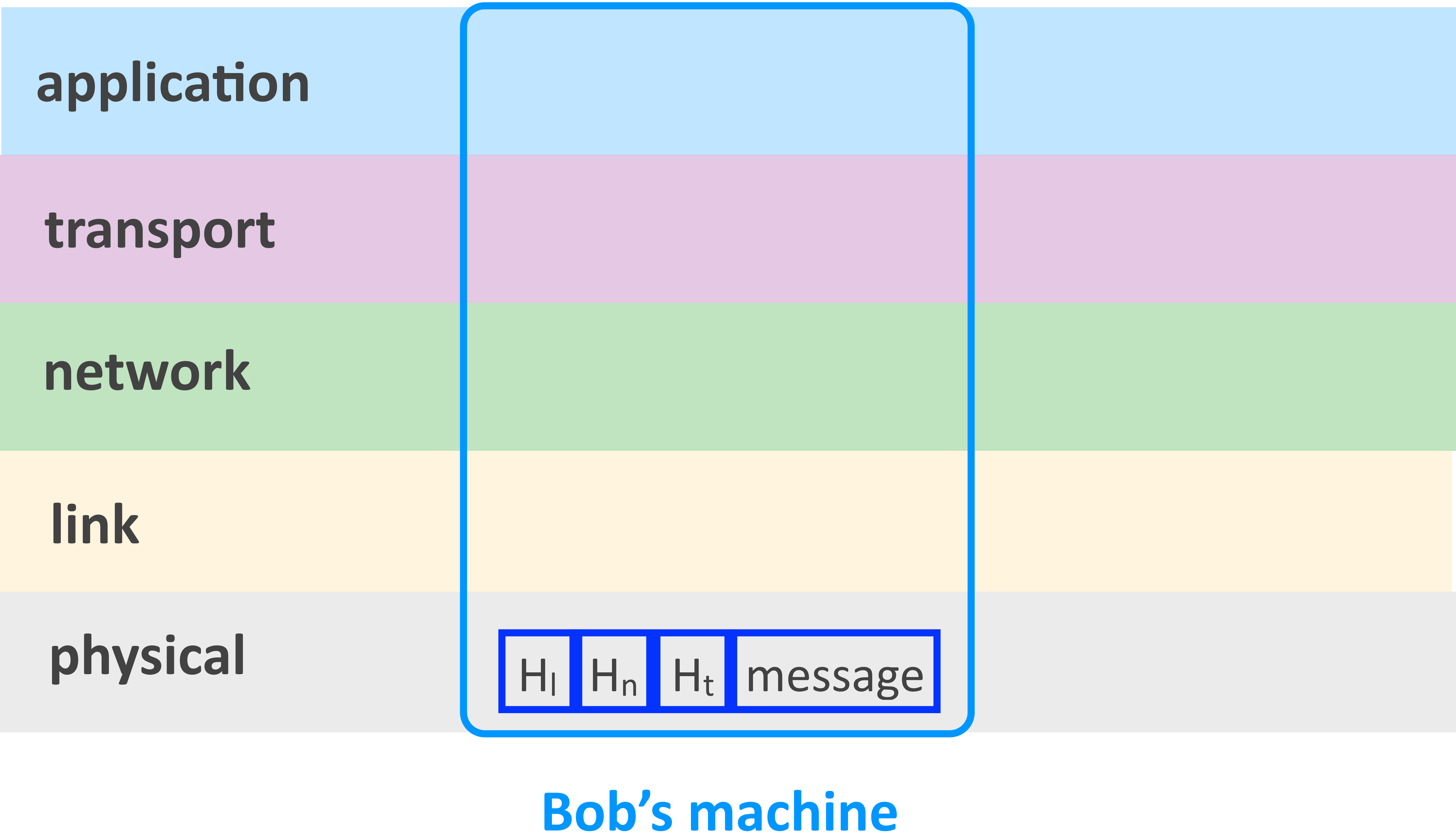
application	*applications that exchange data*
transport	*transports data between end-systems*
network	*moves data around the network*
link	*moves data across a link*
physical	*moves data across a physical medium*

application	*HTTP (web)* *SMTP (email)* *FTP (file transfer)*
transport	*TCP* *UDP*
network	*IP*
link	*Ethernet* *WiFi* *DSL* *Cable* *Cellular*
physical	*twisted pair* *fiber* *coaxial cable* *wireless*


application
transport
network
link
physical
message
H_t
H_n
H_l
Alice's machine
switch


application
transport
network
link
H_l
physical
H_l H_n H_t message
switch


application
transport
network
link
physical
H_n
H_l
H_l H_n H_t message
switch
Bob's machine


application
transport
network
link
physical
H_l
H_n
H_t
message
Bob's machine

Why layers?

- Reduce complexity
- Improve flexibility

Restaurant layers

customer tables

waiting service

cooking

Fast-food layers

customer queue

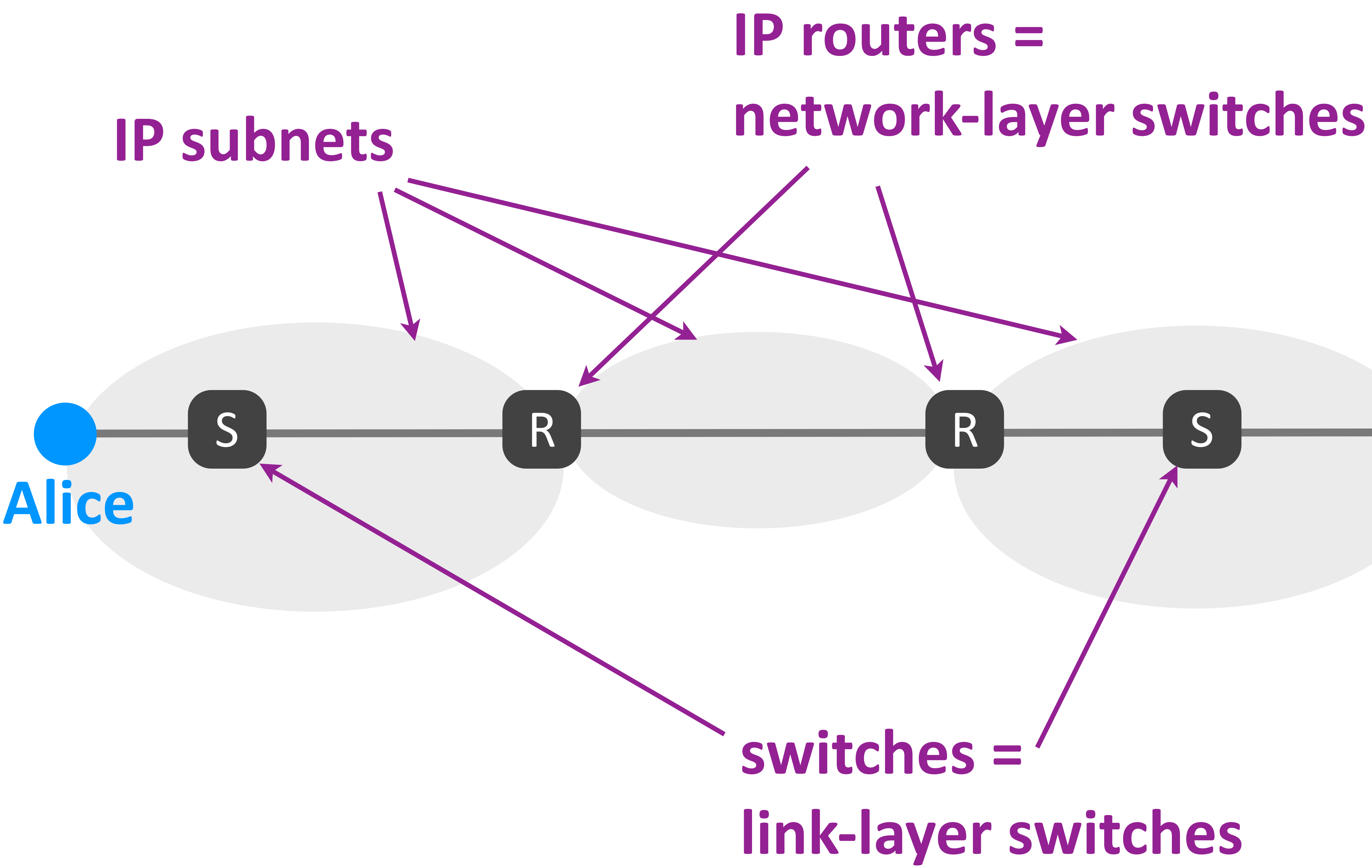
customer service

food packaging

food unfreezing & cooking

food preparation & freezing

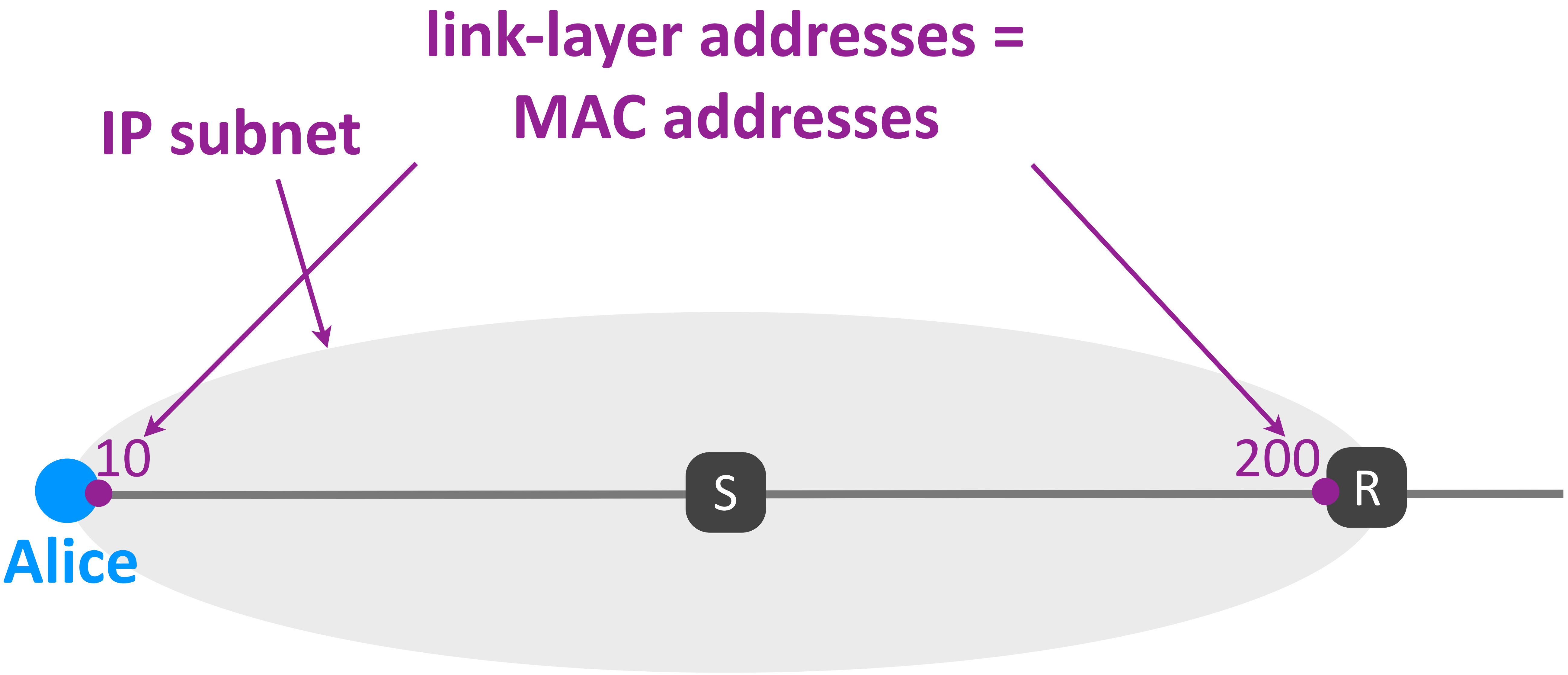
Link layer


IP routers =
network-layer switches
IP subnets
S
R
R
S
Alice
Bob
switches =
link-layer switches

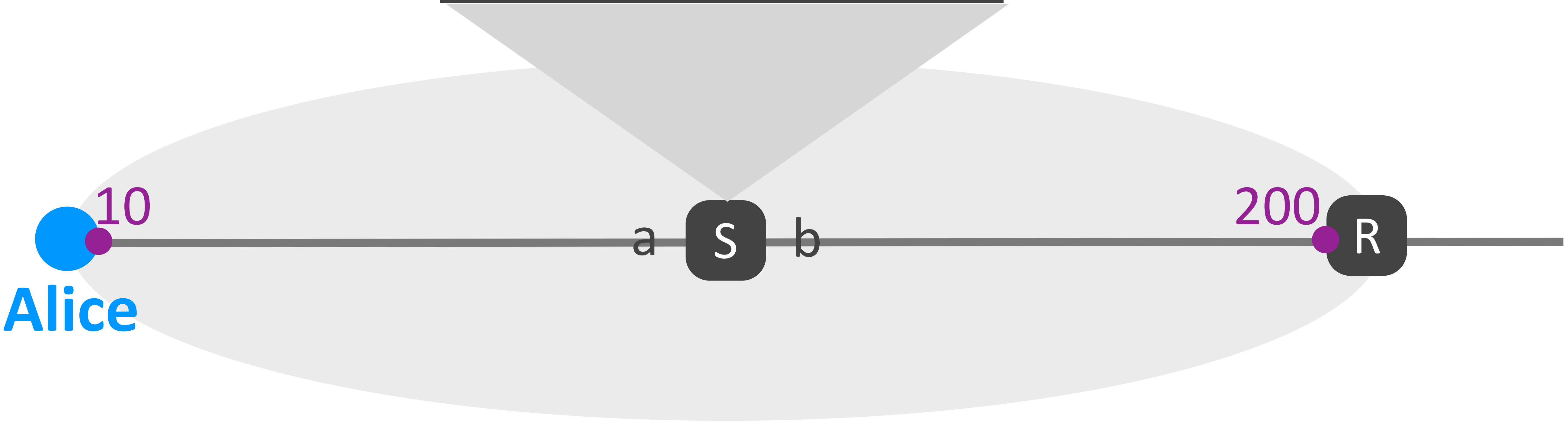
Link vs. network layer

- Link layer: takes packet from one device to the next
  - *across one IP subnet*

- Network layer: takes packet from source end-system to destination end-system
  - *across the entire network*
  - *across a sequence of IP subnets*


link-layer addresses =
MAC addresses
IP subnet
10
200
S
R
Alice

MAC address	output link
10	a
200	b
...	...



Link-layer (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...

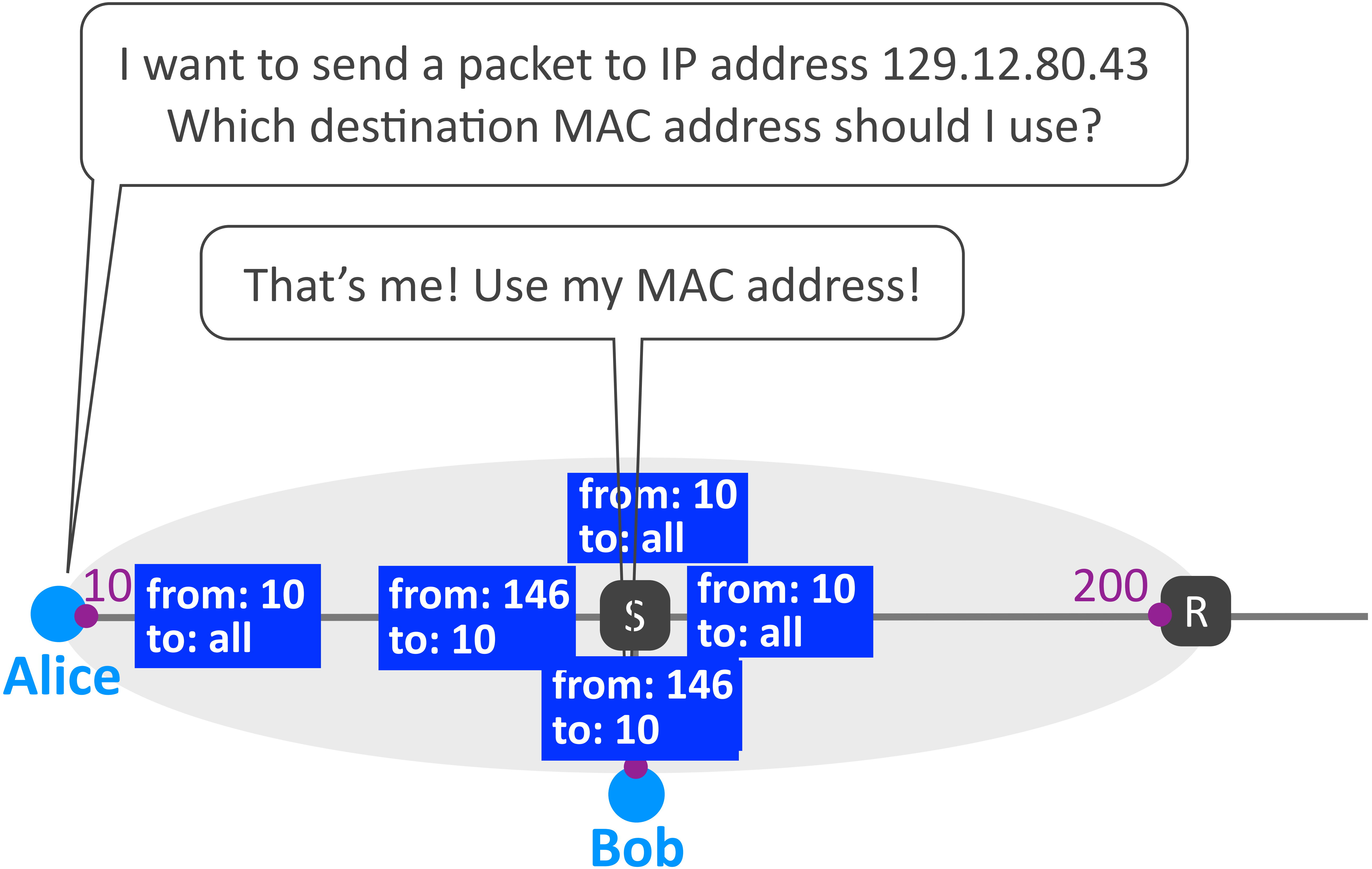
MAC addresses

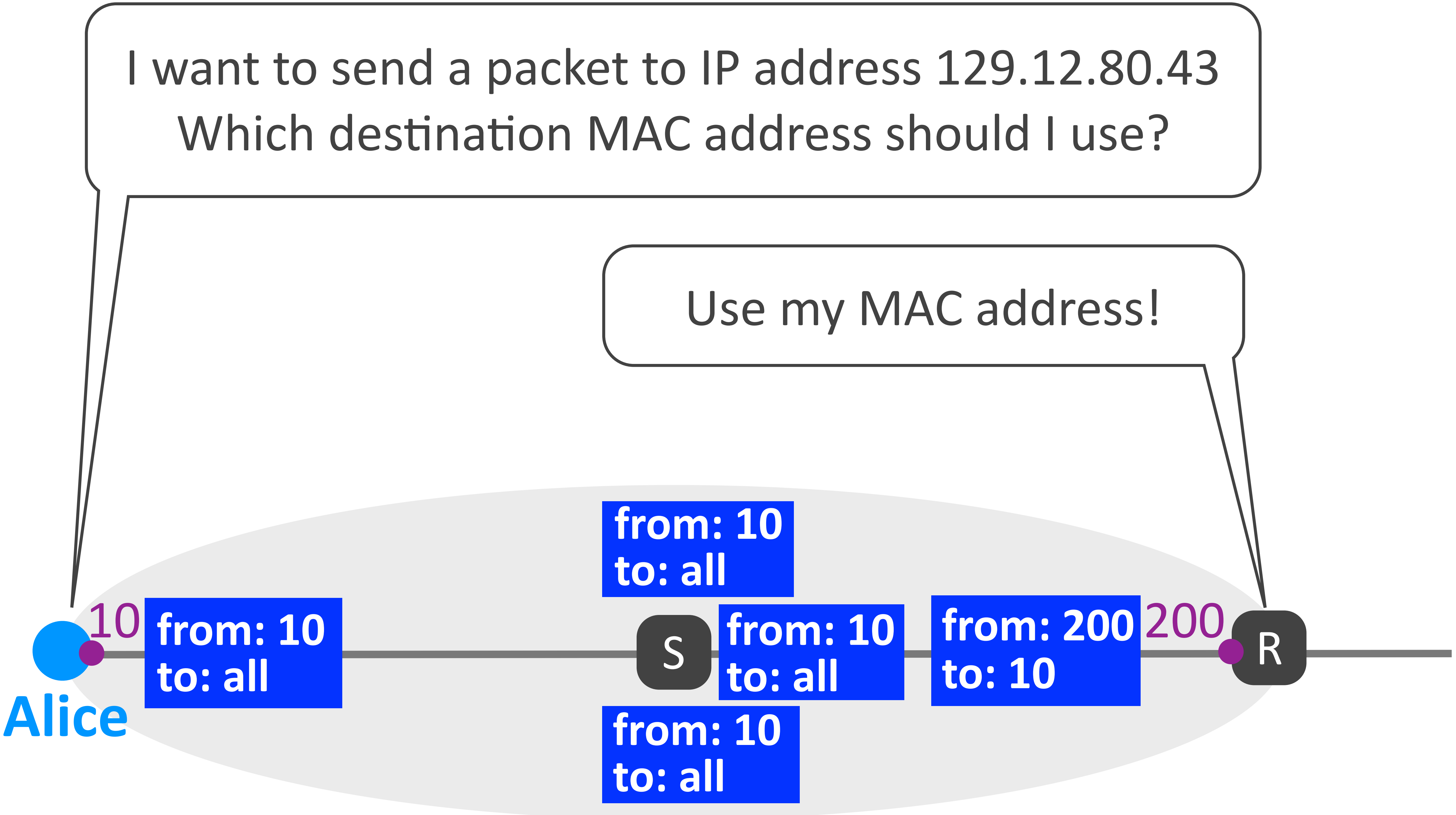
- Flat
  - *not hierarchical like IP addresses*
  - *not location dependent*
- 48-bit = 6-byte long
  - *typical format: 1A-2B-DD-78-CF-CC*
  - *the value of each byte as hexadecimal*

Link-layer vs. IP forwarding

- Link-layer (L2): based 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): based on hierarchical addresses
  - *IP addresses grouped in IP prefixes*
  - *forwarding table size = # of IP prefixes in the world*


I want to send a packet to IP address 129.12.80.43
Which destination MAC address should I use?
That's me! Use my MAC address!
from: 10
to: all
10
from: 10
to: all
from: 146
to: 10
S
from: 10
to: all
200
R
from: 146
to: 10
Alice
Bob


I want to send a packet to IP address 129.12.80.43
Which destination MAC address should I use?
Use my MAC address!
from: 10
to: all
10
from: 10
to: all
S
from: 10
to: all
from: 200
to: 10
200
R
Alice
from: 10
to: all

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, except...

Broadcasting

- Alice sends request to special, broadcast destination MAC address
  - *FF-FF-FF-FF-FF-FF*

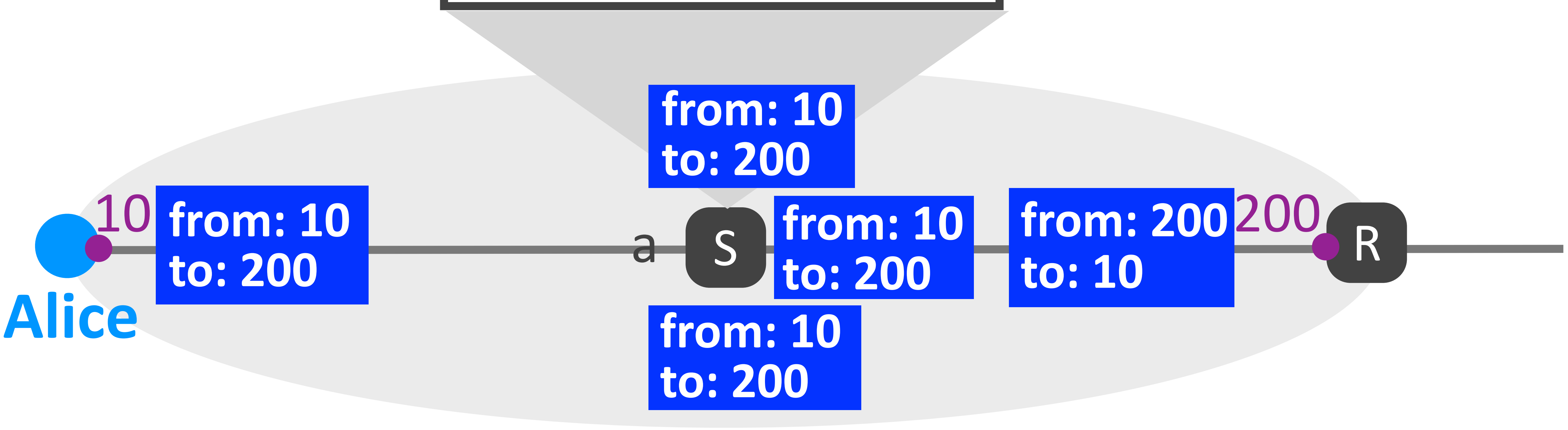
- Reaches every entity in this IP subnet that has a MAC address

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*

MAC address	output link
10	a
200	b



Self-learning

- Switch learns from traffic
  - *when packet with src MAC x arrives at link y,*
  - *switch adds MAC x --> link y mapping to fwding table*

- ... and broadcasts when it does not know
  - *when packet with unknown dst MAC arrives,*
  - *switch broadcasts the packet*

- Serves similar role as routing, except...

Self-learning vs. routing

- Self-learning: relies on actual traffic
  - *switches do not exchange explicit routing information*
- Routing: relies on routing protocol
  - *routers exchange explicit routing messages*

Link-layer elements

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

Link-layer vs. IP forwarding (revisited)

- Link-layer: flat addresses + self-learning/broadcasting
  - *designed for flexibility*

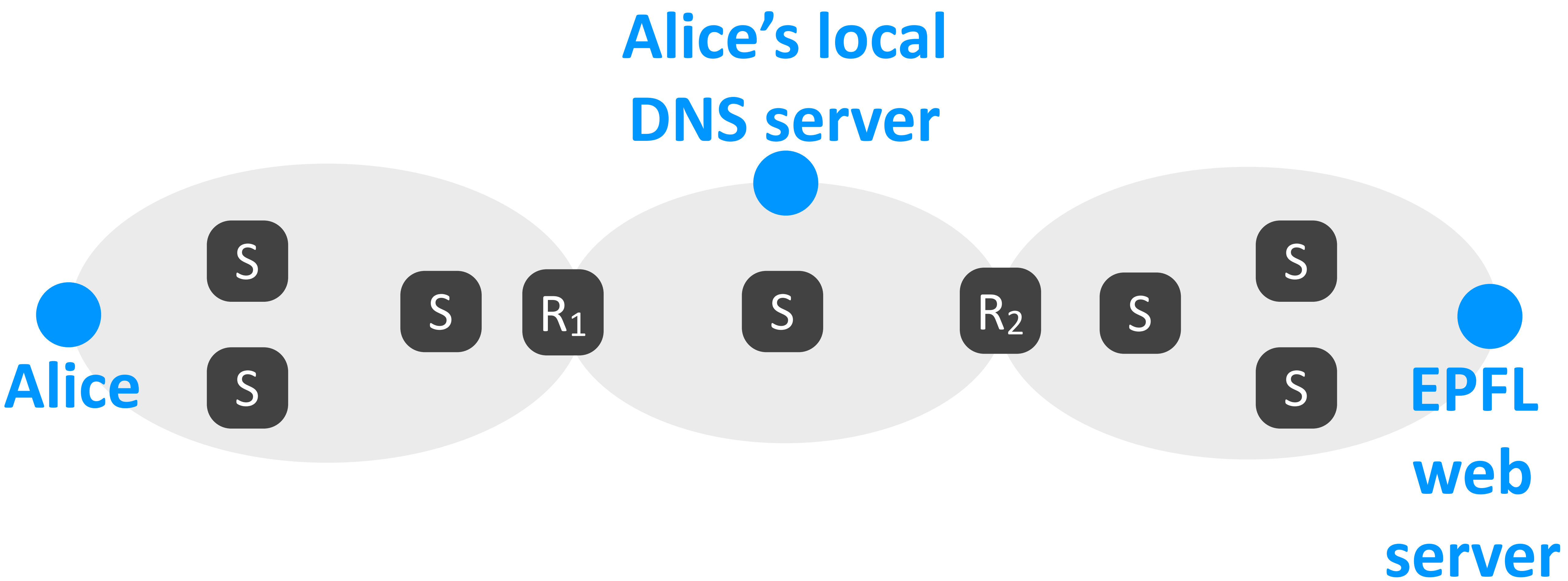
- IP: hierarchical addresses + routing
  - *designed for scalability*

Get rid of IP forwarding?

Get rid of link-layer forwarding?

3 levels of hierarchy

- IP subnet
  - *link-layer (L2) forwarding*
  - *self-learning/broadcasting*
- Autonomous System (AS)
  - *IP (L3) forwarding*
  - *intra-domain routing (usually link-state)*
- Internet
  - *IP (L3) forwarding*
  - *inter-domain routing (distance-vector, BGP)*



R: routers
S: switches
gray circles: IP subnets

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

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

1. A’s DNS client process creates DNS request

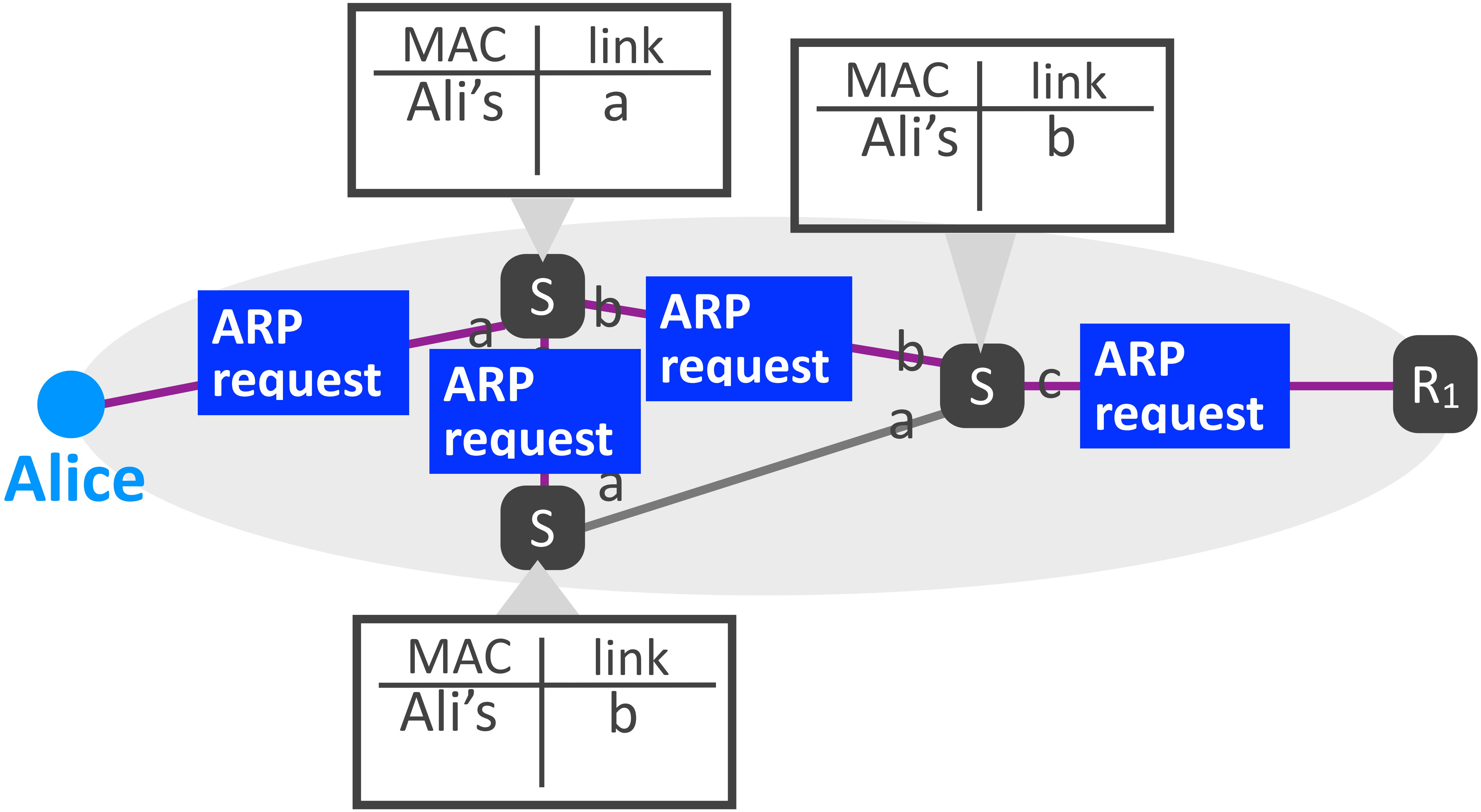
2. Passed down to transport, network layer

- *IP src: A’s IP address*
- *IP dst: local DNS server’s IP address*

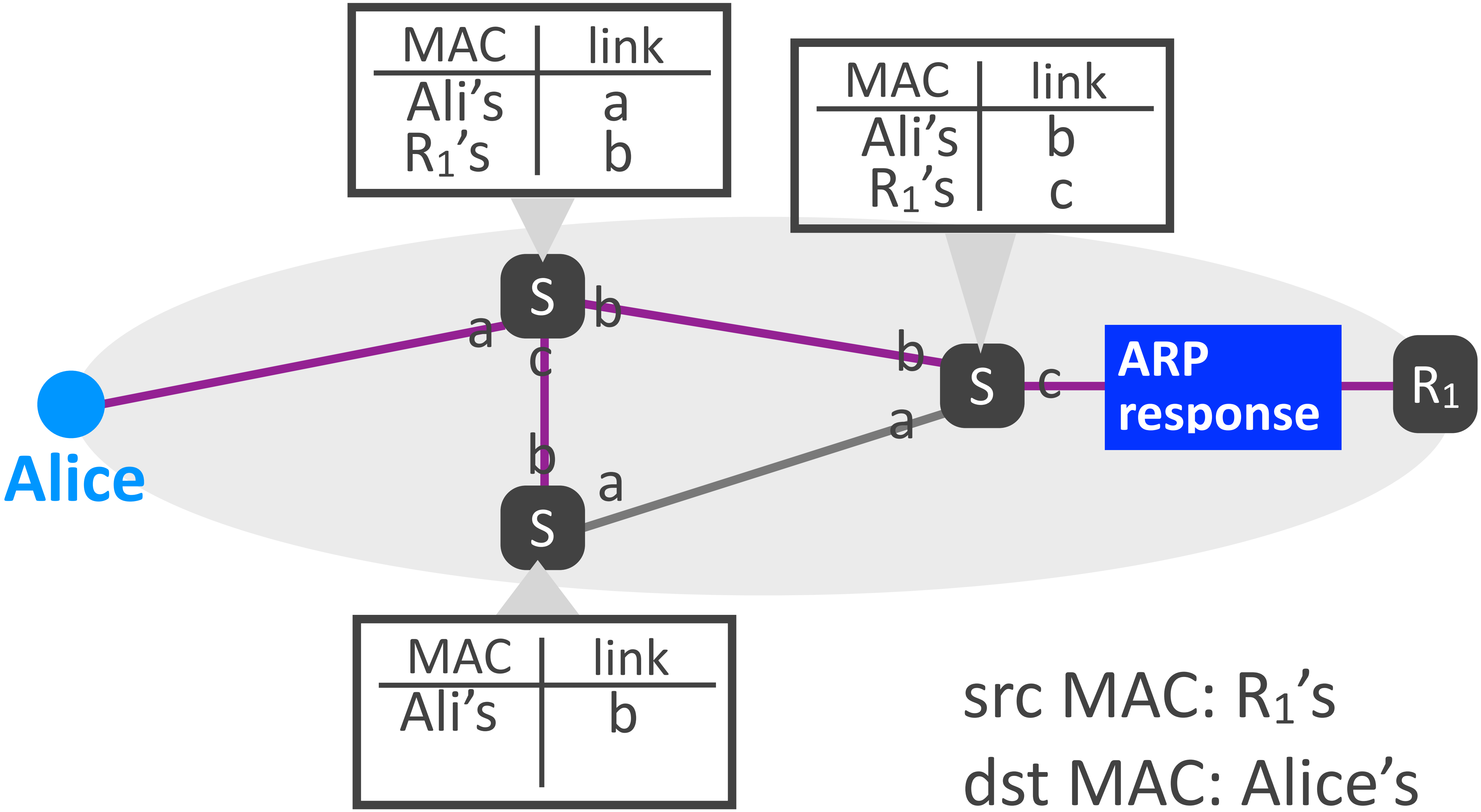
3. A’s network layer sends ARP request

- *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

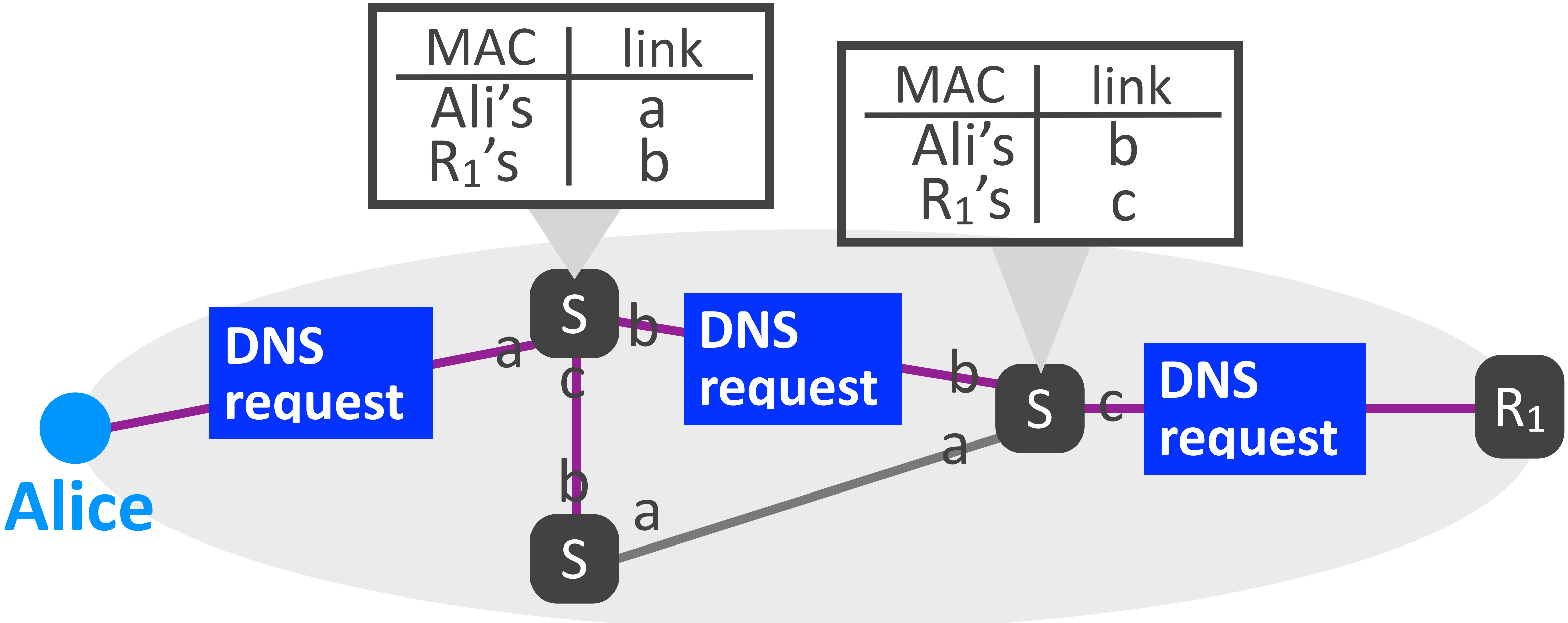
dst MAC: Alice's

5. A’s network layer sends DNS request

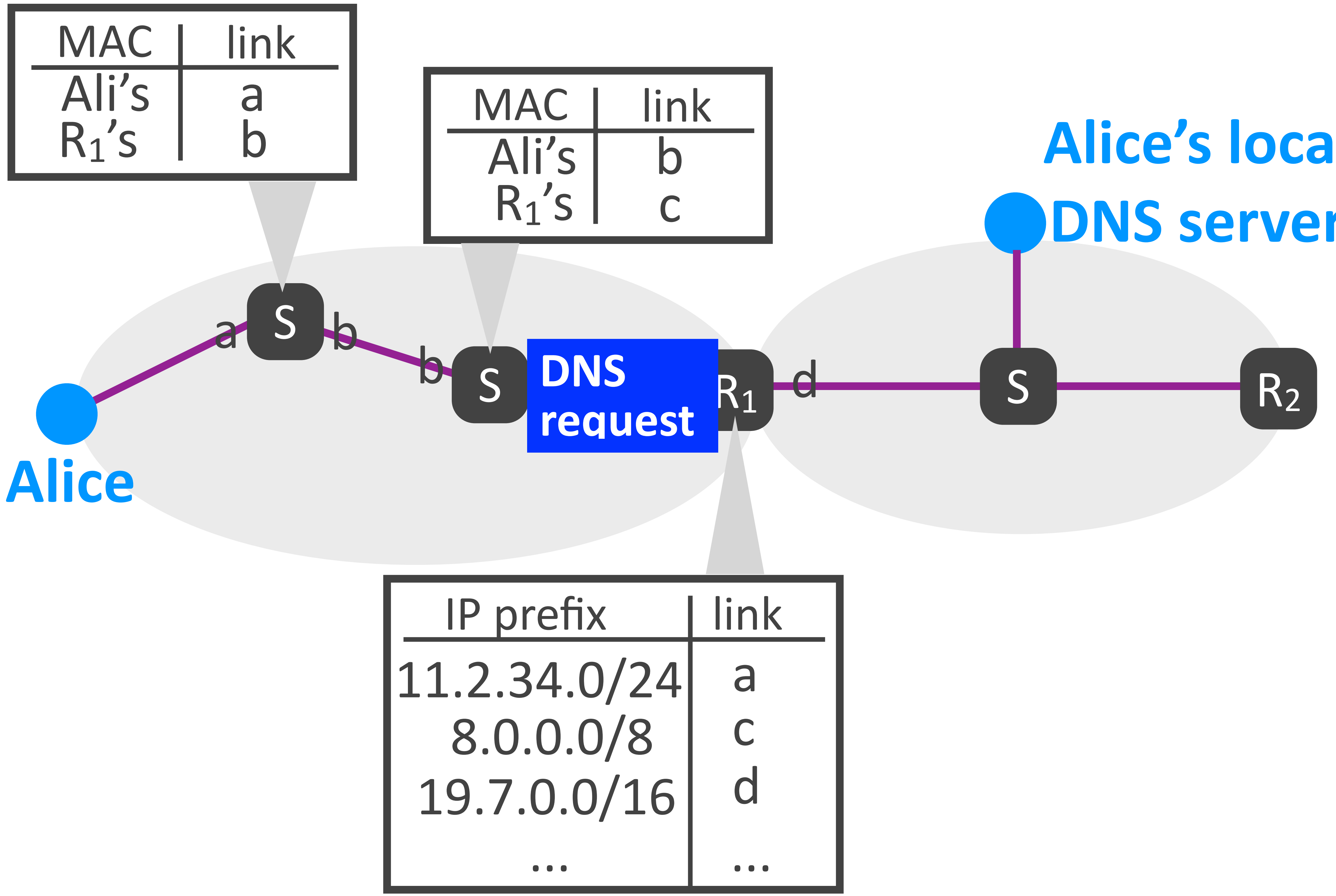
- *it now knows the right MAC address to use*

src MAC: Alice's
dst MAC: R_1's

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



6. R_1's network layer performs L3 forwarding

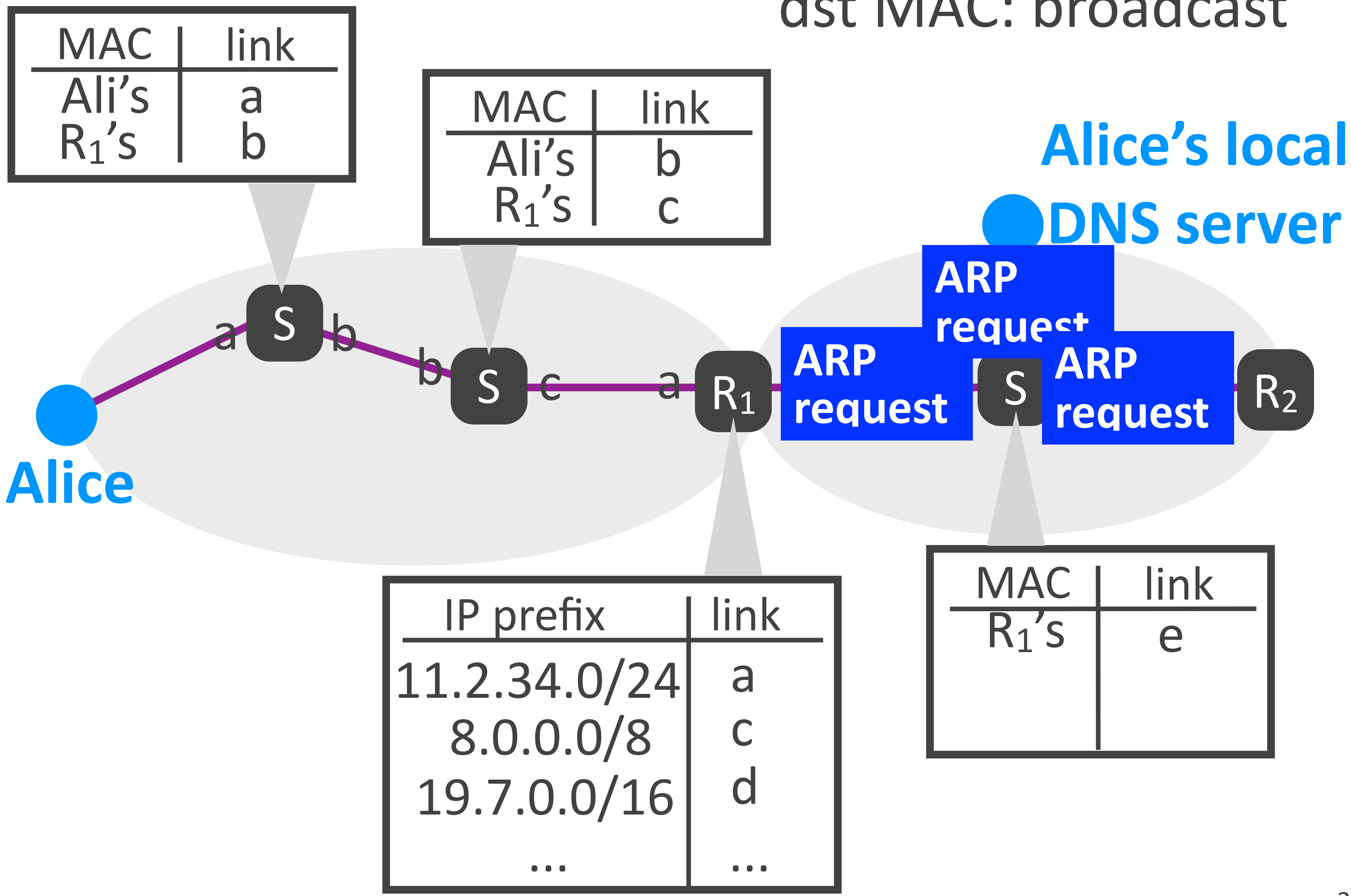

MAC | link
Ali's | a
R1's | b
MAC | link
Ali's | b
R1's | c
Alice's local DNS server
a
S
b
b
S
DNS request
R1
d
S
R2
Alice
IP prefix | link
11.2.34.0/24 | a
8.0.0.0/8 | c
19.7.0.0/16 | d
... | ...

7. R_1's network layer sends ARP request

- *to resolve DNS server's IP address*

src MAC: R_1's

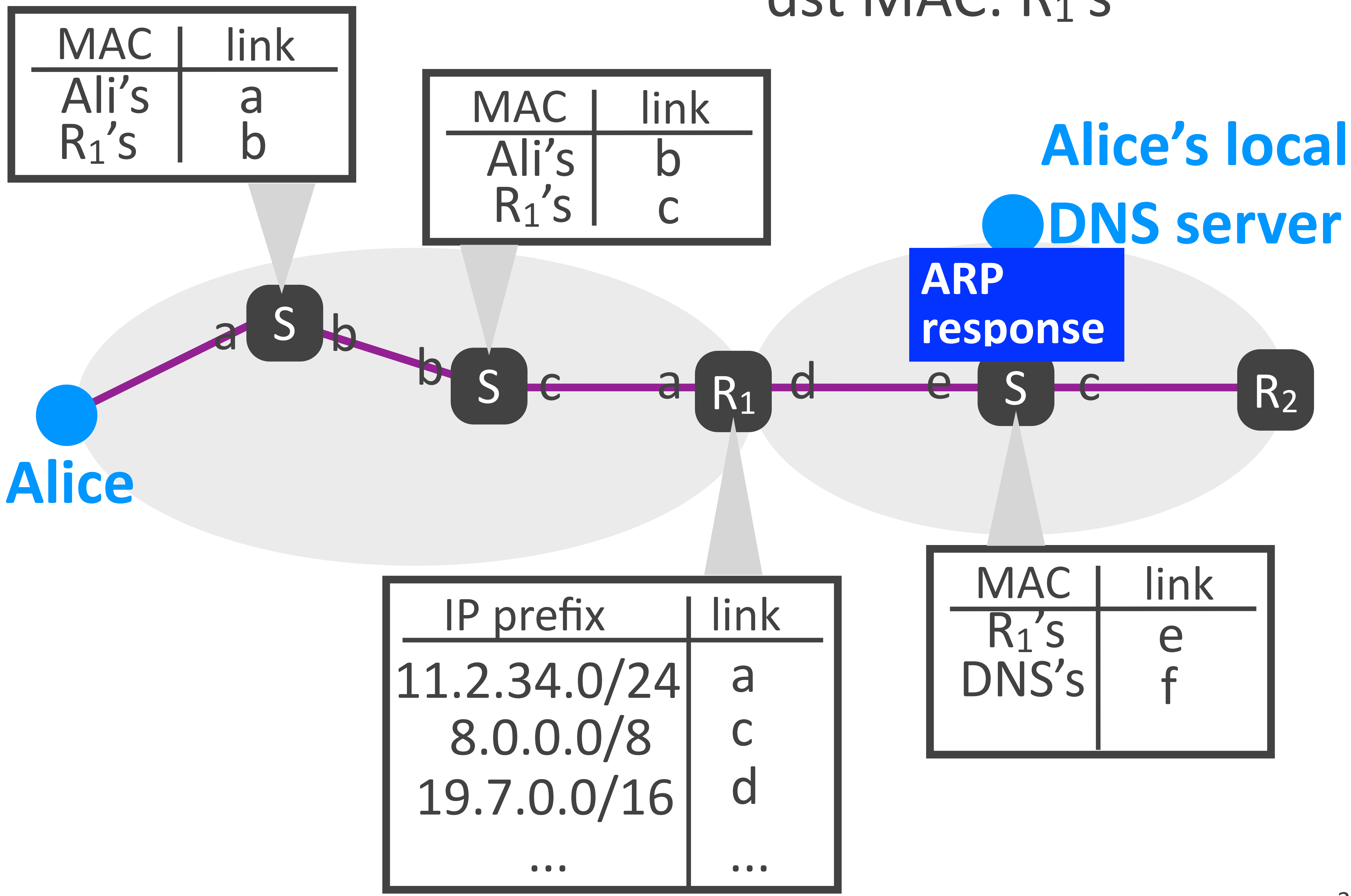
dst MAC: broadcast



8. DNS server's network layer sends ARP response

src MAC: DNS servers's

dst MAC: R_1's

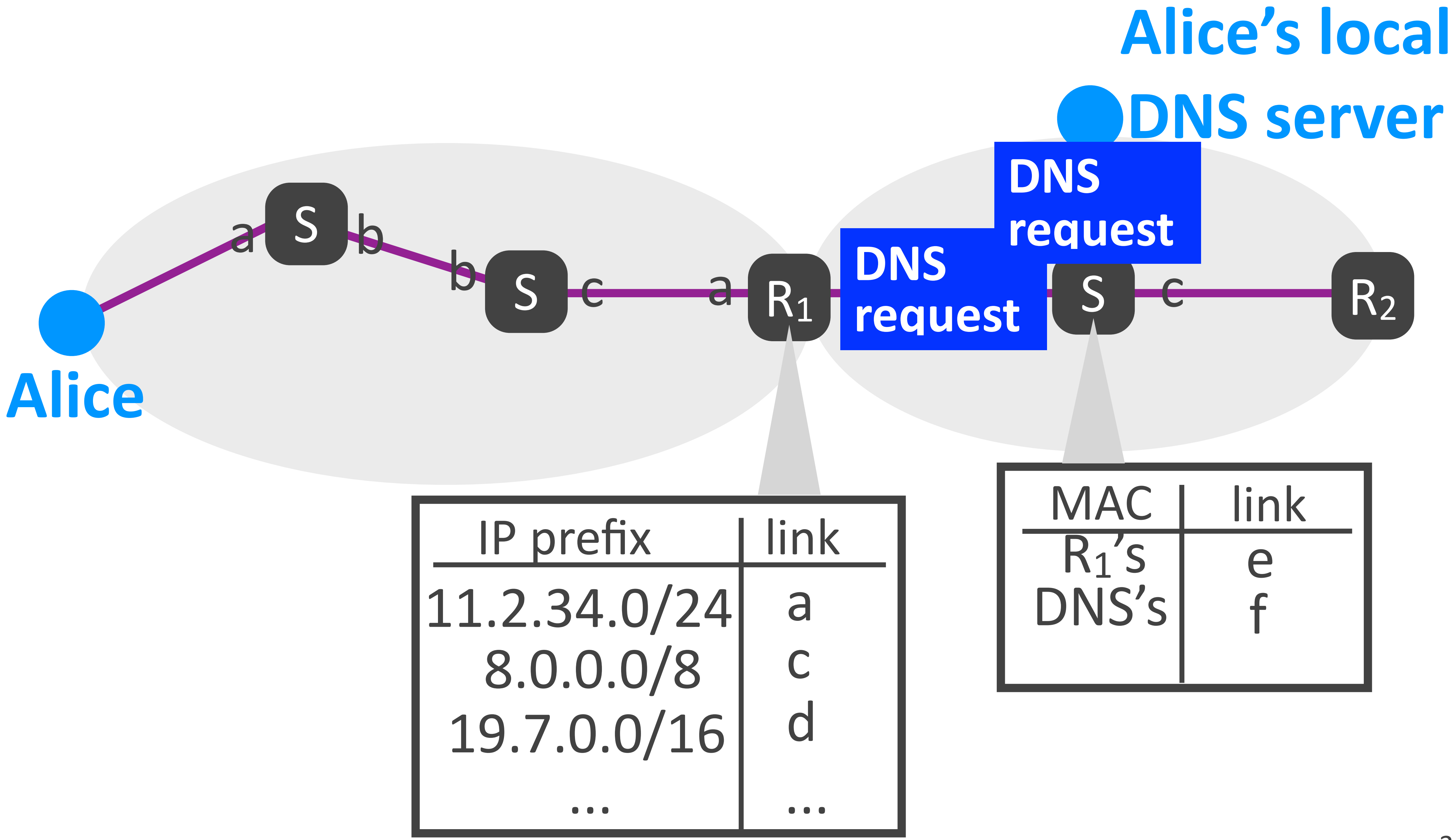


9. R_1's network layer forwards DNS request

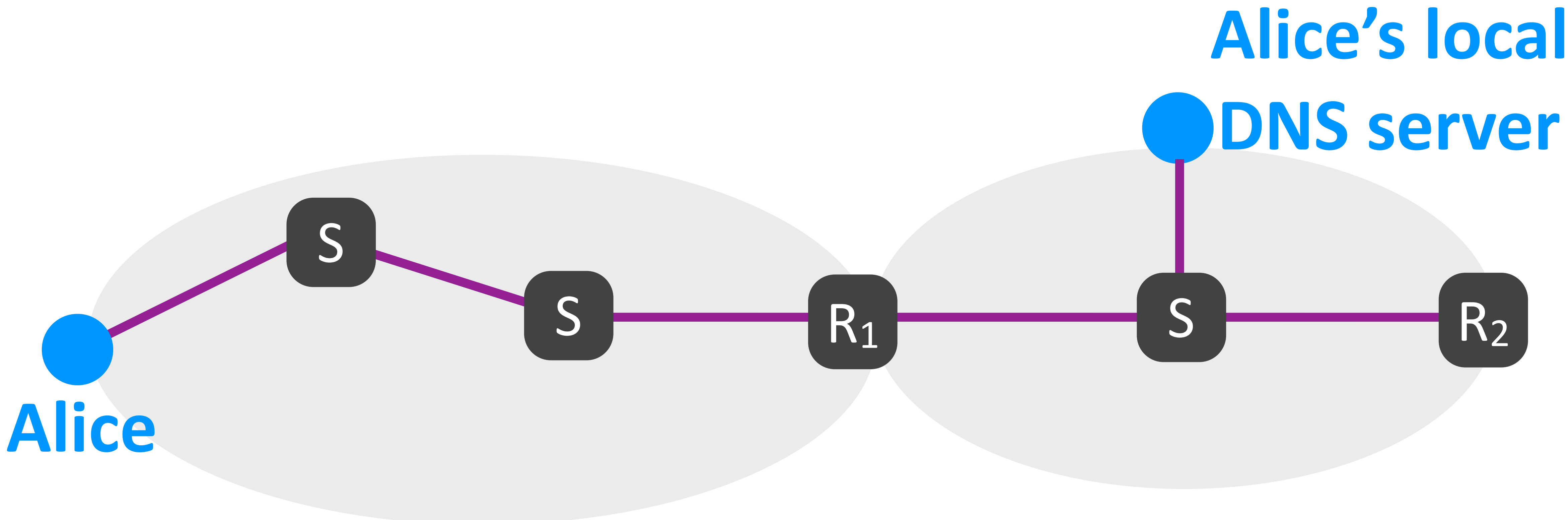
- *it now knows the right MAC address to use*

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

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

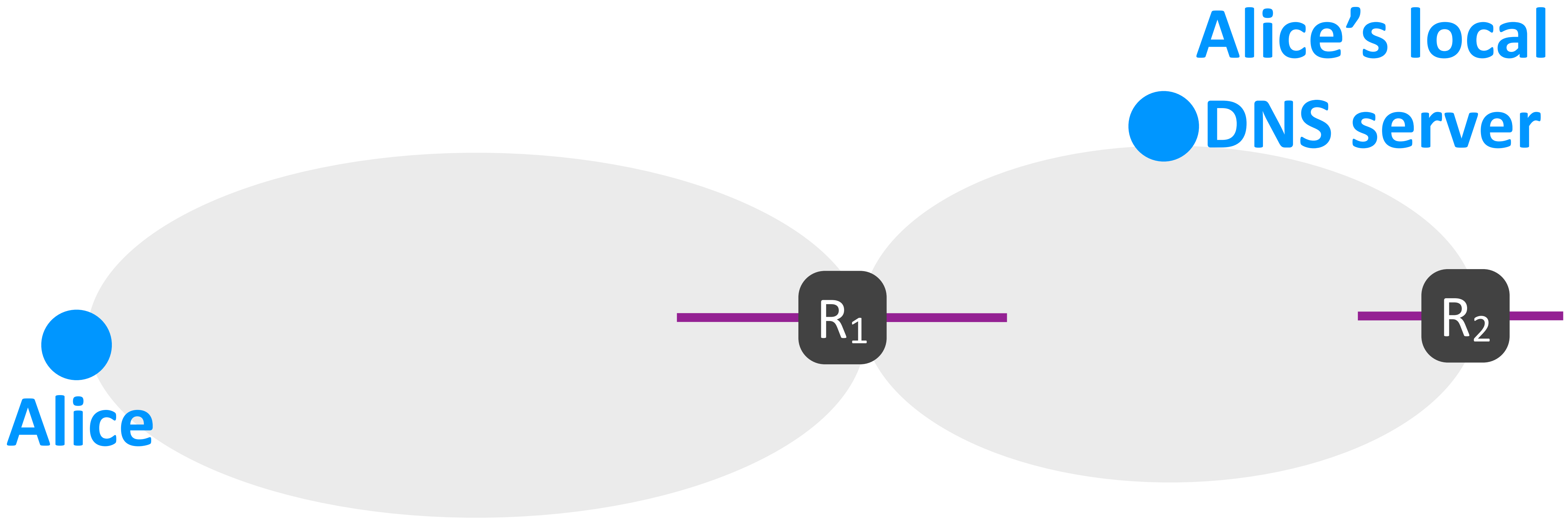


The switches forward traffic within local IP subnet between end-systems and routers



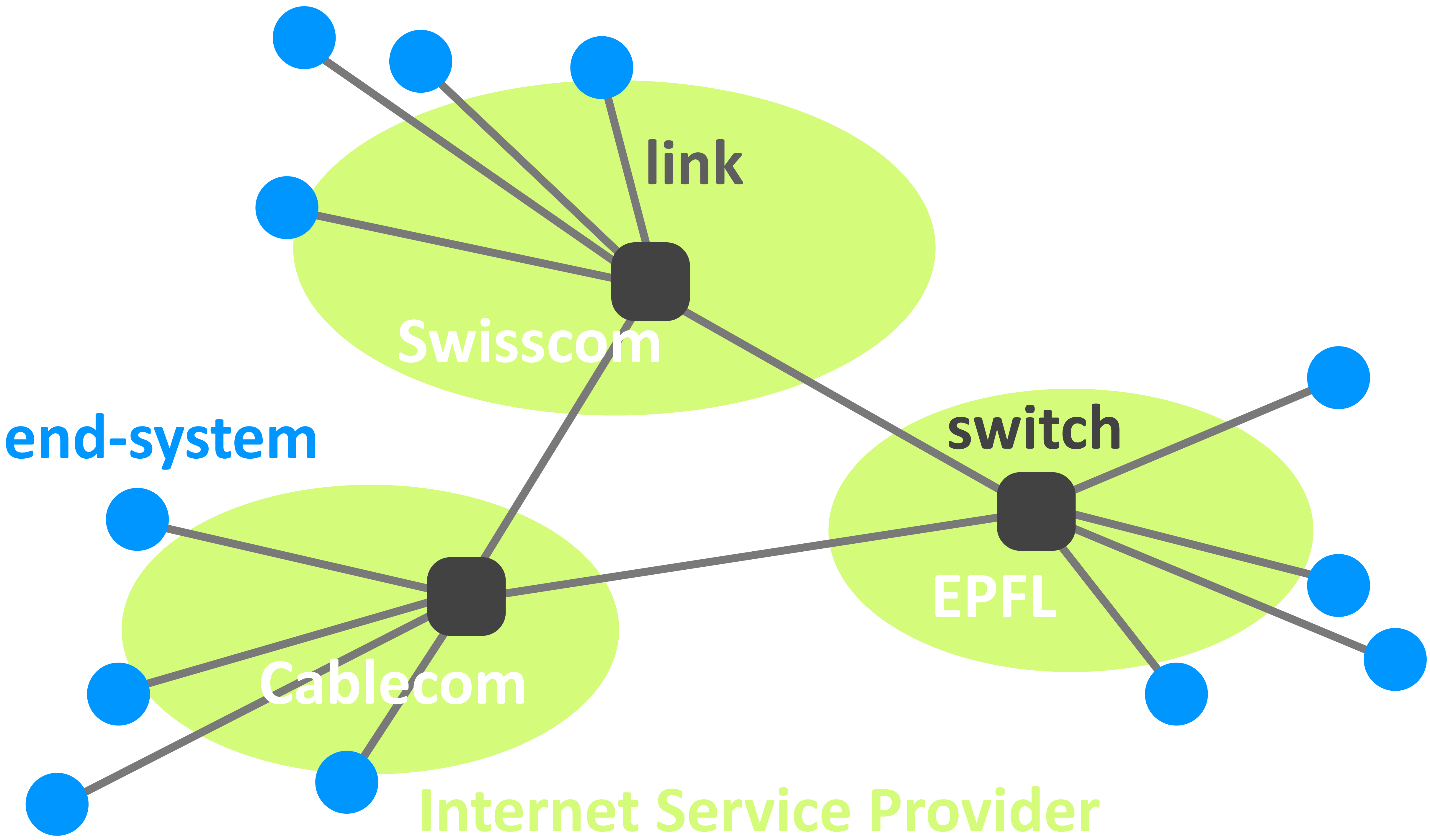
End-systems and routers need MAC addresses

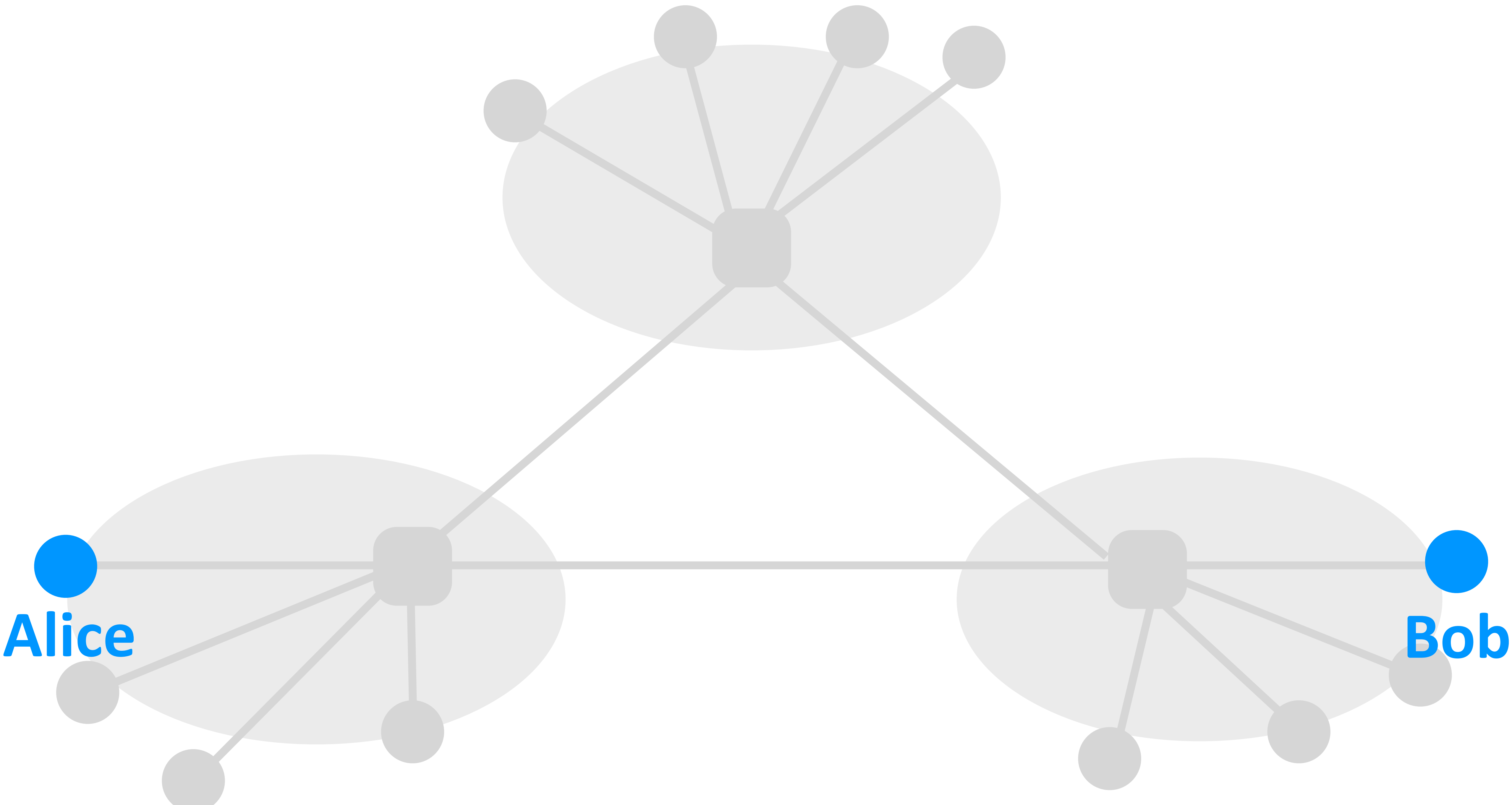
The routers forward traffic end-to-end between source and destination end-systems

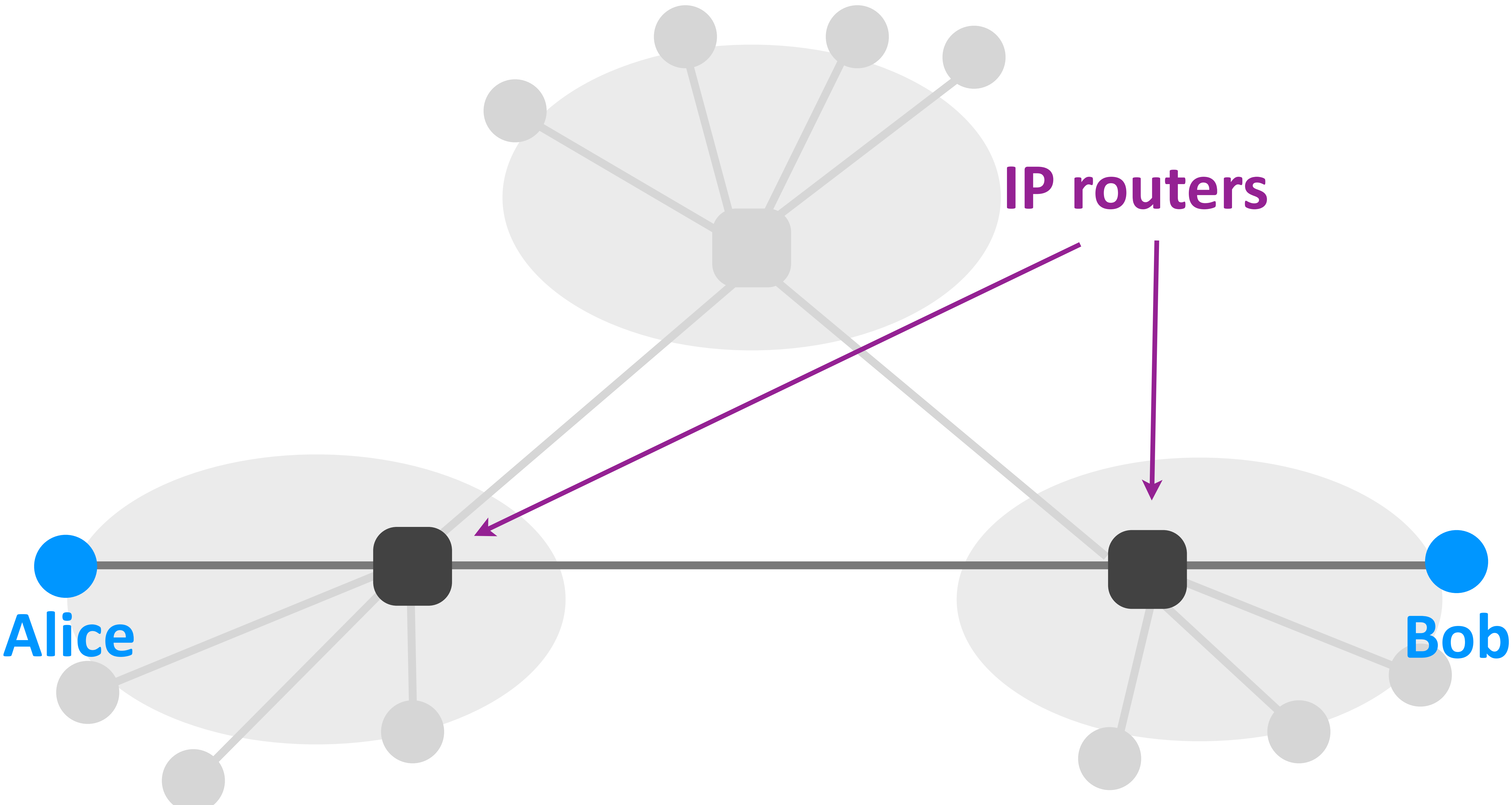


End-systems need IP addresses

Network layer


link
Swisscom
switch
end-system
EPFL
Cablecom
Internet Service Provider


Alice
Bob


IP routers
Alice
Bob

dest. address	output link
0	1
1	2
2	2
3	3

dest. address	output link
0	3
1	1
2	2
3	4

dest. address	output link
0 - 1000	2
1001 - 1500	1
1501 - 1502	3
otherwise	1

dest. address	output link
0 - 255	1
256 - 46780	2
46781 - ...	3
otherwise	4

dest. address range			output link
0 - 3	0000 - 0011	00**	1
4 - 7	0100 - 0111	01**	2
8 - 11	1000 - 1011	10**	3
12 - 15	1100 - 1111	11**	4

0100
1110

dest. address range			output link
0 - 2	0000 - 0010	00**	1
3	0011	0011	2
4, 6, 7	0100, 0110, 0111	01**	3
5	0101	0101	2
8 - 15	1000 - 1111	1***	4

0111

dest. address range			output link
0 - 1	0000 - 0001	000*	1
2 - 3	0010 - 0011	001*	2
4 - 7	0100 - 0111	01**	3
8	1000	1000	2
9 - 15	1001 - 1111	1***	4

prefixes

1000

longest prefix matching

dest. address range			output link
0 - 3	0000 - 0011	00**	1
4 - 7	0100 - 0111	01**	2
8 - 11	1000 - 1011	10**	3
12 - 15	1100 - 1111	11**	4

dest. address range			output link
0 - 2	0000 - 0010	00**	1
3	0011	0011	2
4, 6, 7	0100, 0110, 0111	01**	3
5	0101	0101	2
8 - 15	1000 - 1111	1***	4

dest. address range			output link
0 - 1	0000 - 0001	000*	1
2	0010	0010	2
3	0011	0011	3
4, 6, 7	0100, 0110, 0111	01**	2
5	0101	0101	4
8 - 15	1000 - 1111	1***	1
10	1010	1010	3

Location-dependent addresses

- Address embeds location information
  - *address proximity implies location proximity*
- Significantly reduces forwarding state
  - *per destination prefix*
  - *(otherwise, it would be per destination)*

IP address format

IP address = number from 0 to 2^{32}-1

11011111 00000001 00000001 00000001

223 . 1 . 1 . 1

IP prefix format

IP prefix = range of IP addresses

223.1.1.0 / 24 ← mask

11011111 00000001 00000001 00000000

11011111 00000001 00000001 ********

223.1.1.*

IP prefix format

IP prefix = range of IP addresses

223.1.1.74 / 24 ← mask

11011111 00000001 00000001 01001001

11011111 00000001 00000001 ********

223.1.1.*

IP prefix format

IP prefix = range of IP addresses

223.1.1.74 / 24 ⟵ mask

223.1.1.0 / 24

223.1.1.113 / 24

223.1.1.*

IP prefix format

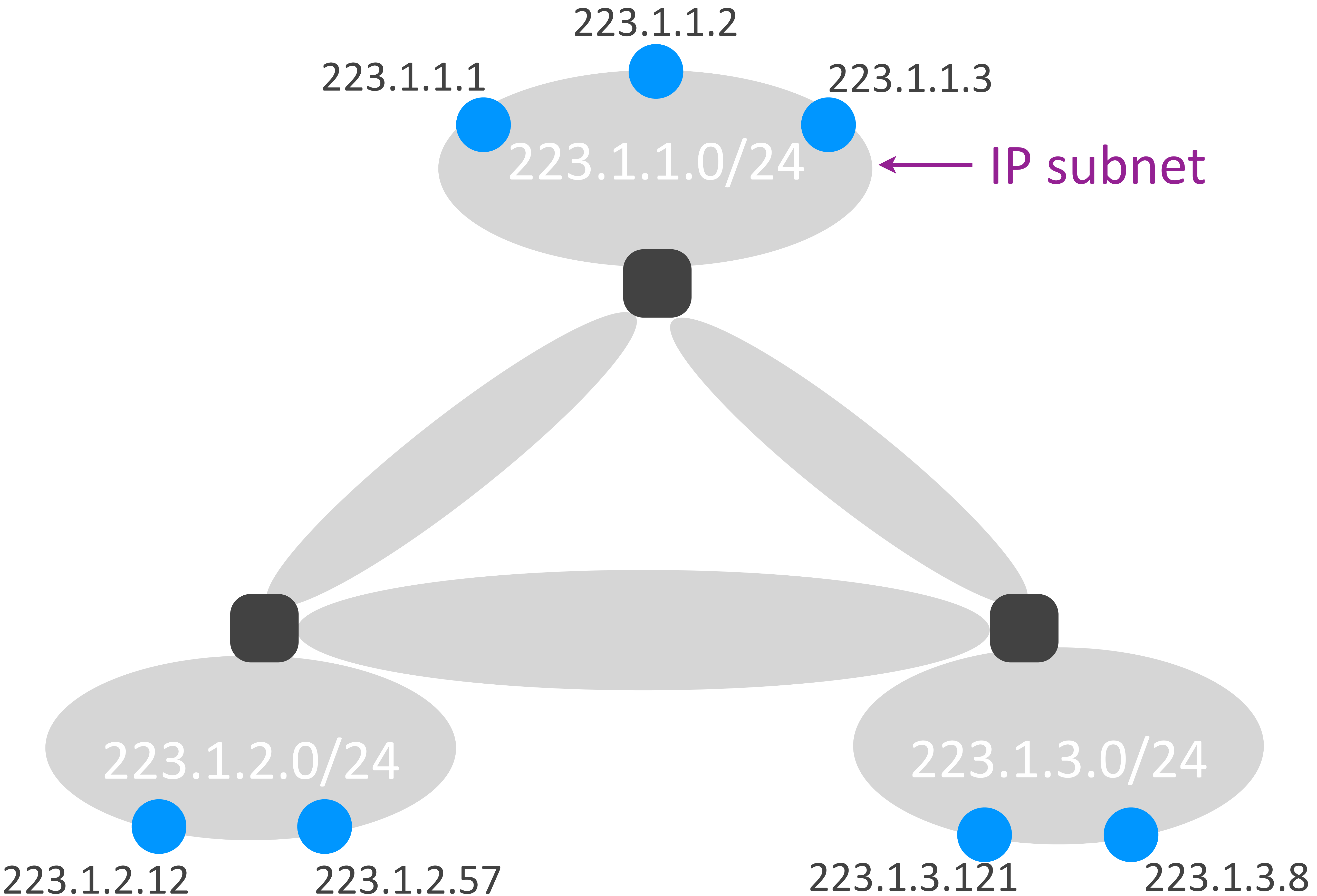
IP prefix = range of IP addresses

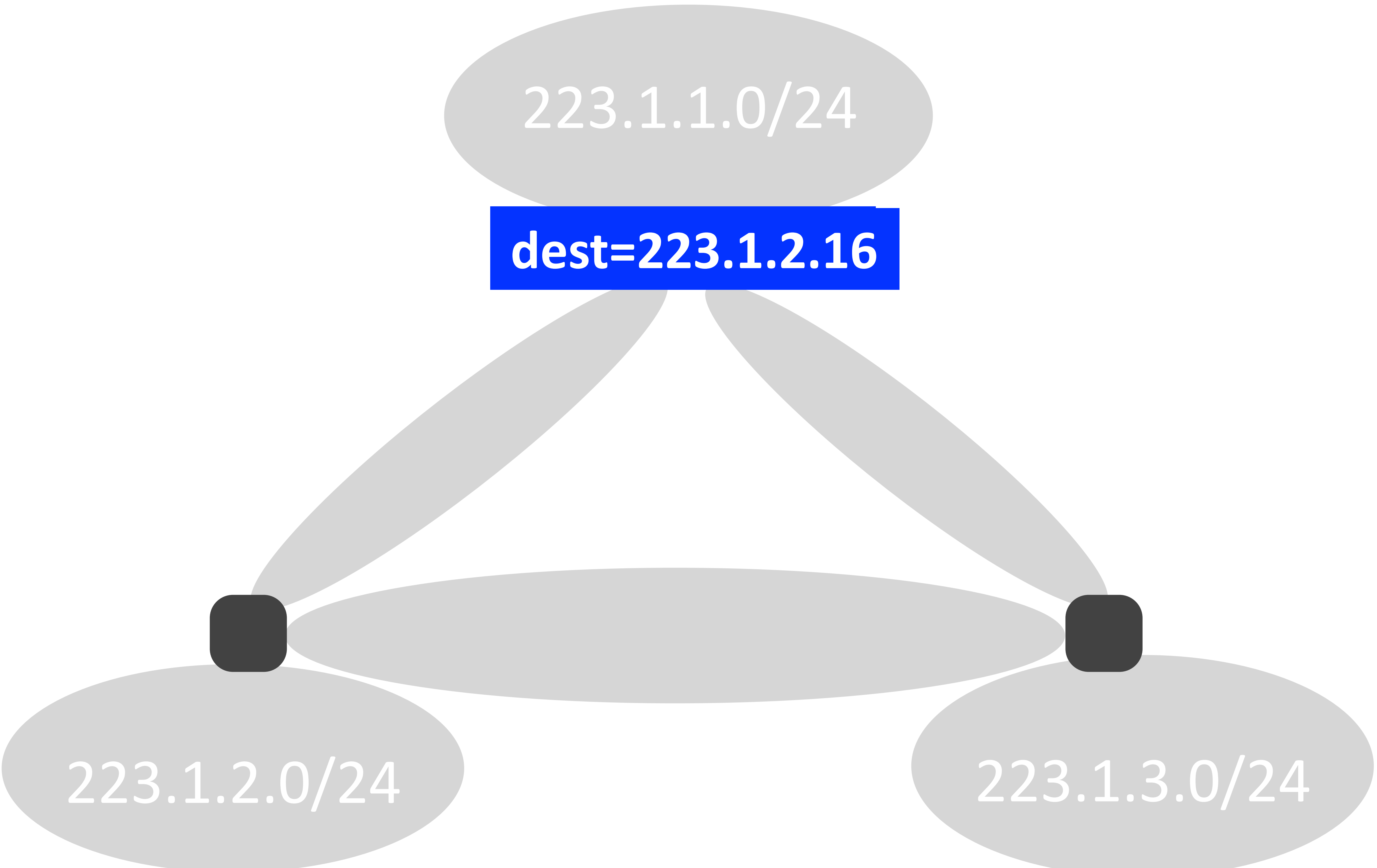
223.1.1.0 / 8 ⟵ mask

11011111 00000001 00000001 00000000

11011111 ******** ******** ********

223.*.*.*


223.1.1.2
223.1.1.1
223.1.1.3
223.1.1.0/24
IP subnet
223.1.2.0/24
223.1.3.0/24
223.1.2.12
223.1.2.57
223.1.3.121
223.1.3.8

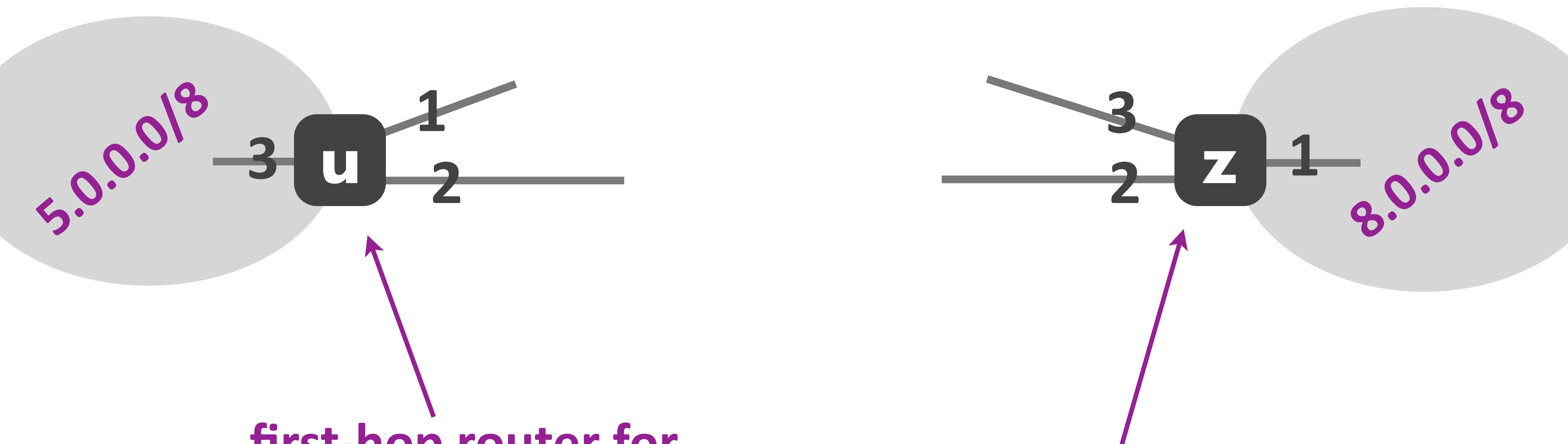

223.1.1.0/24
dest=223.1.2.16
223.1.2.0/24
223.1.3.0/24

IP subnet

- (Informal) Contiguous network area that does not “include” any routers
- All its end-systems and incident routers have IP addresses from the same IP prefix

dest.	out. link
5.0.0.0/8	3
8.0.0.0/8	2

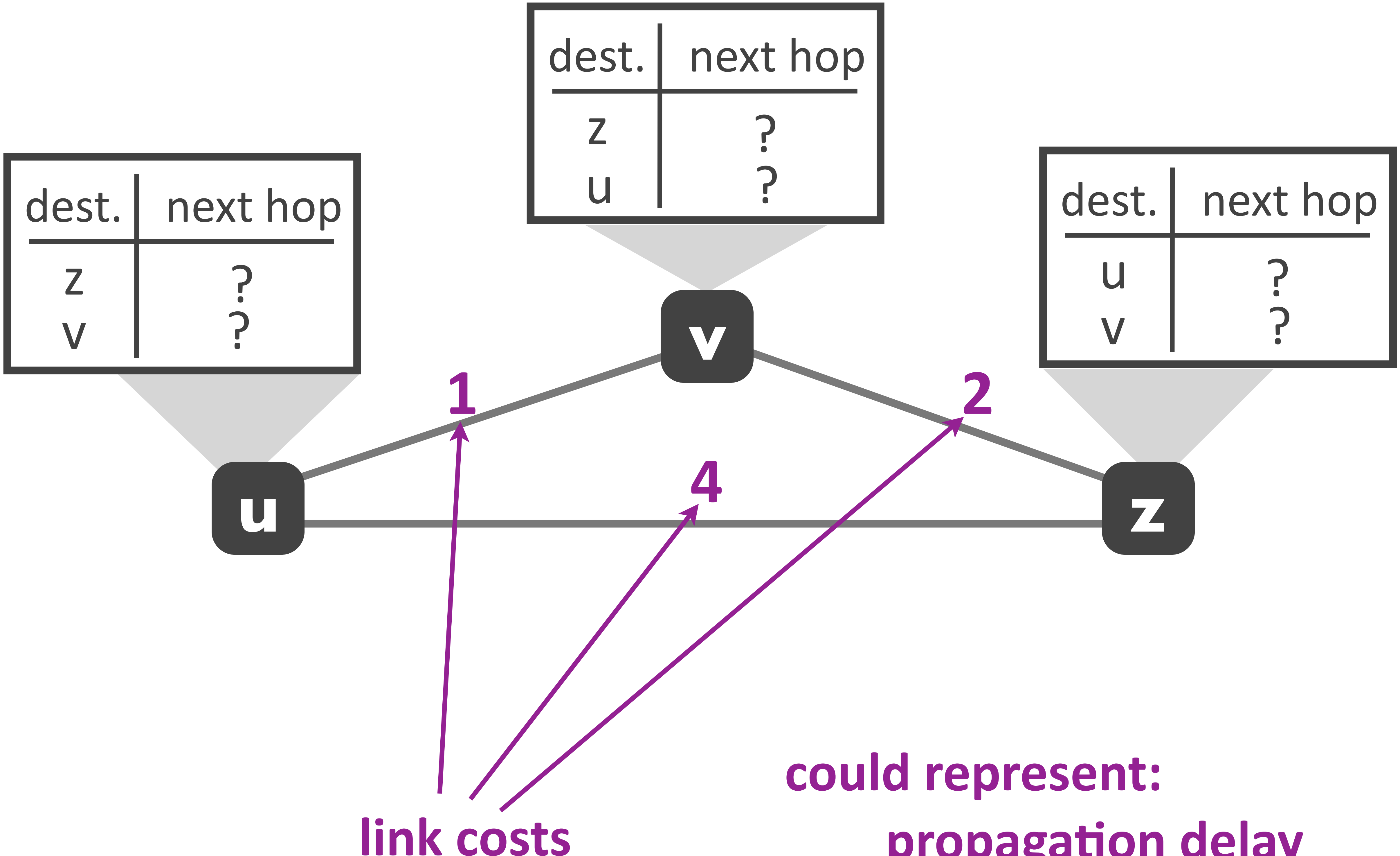
dest.	out. link
8.0.0.0/8	1
5.0.0.0/8	2



first-hop router for IP subnet 5.0.0.0/8

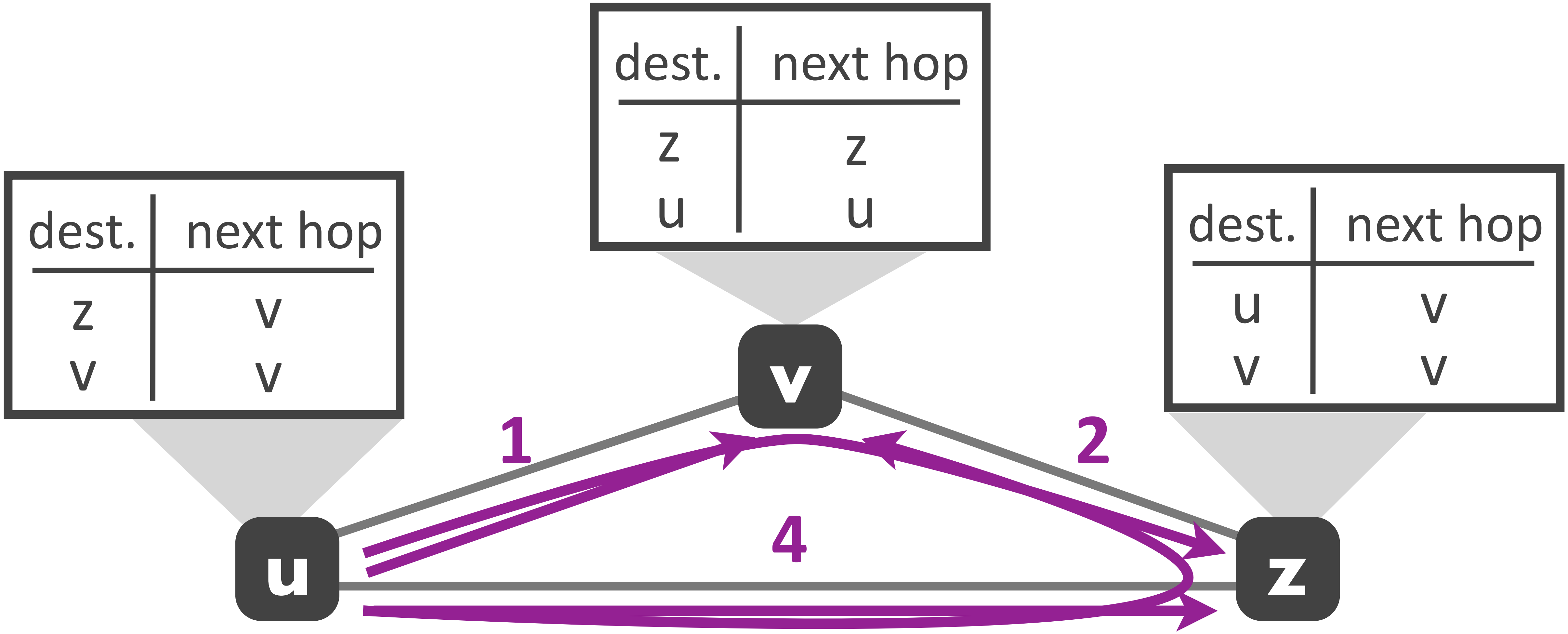
first-hop router for IP subnet 8.0.0.0/8


u
z



could represent:

- propagation delay
- load



least-cost path from u to z: u v z

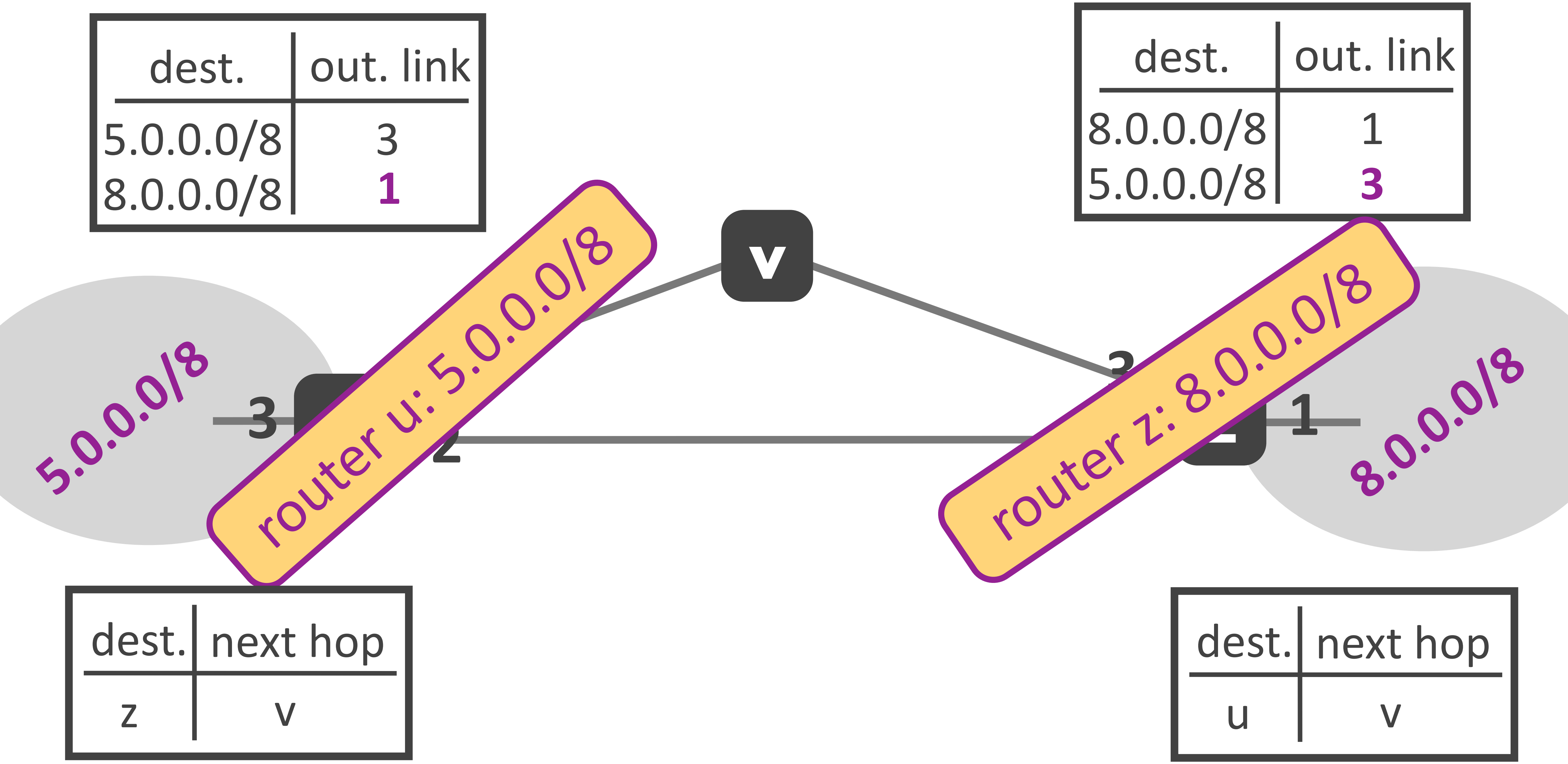
least-cost path from u to v: u v

Least-cost path routing

- Given: router graph & link costs
- Goal: find least-cost path
  from each source router
  to each destination router

dest.	out. link
5.0.0.0/8	3
8.0.0.0/8	?

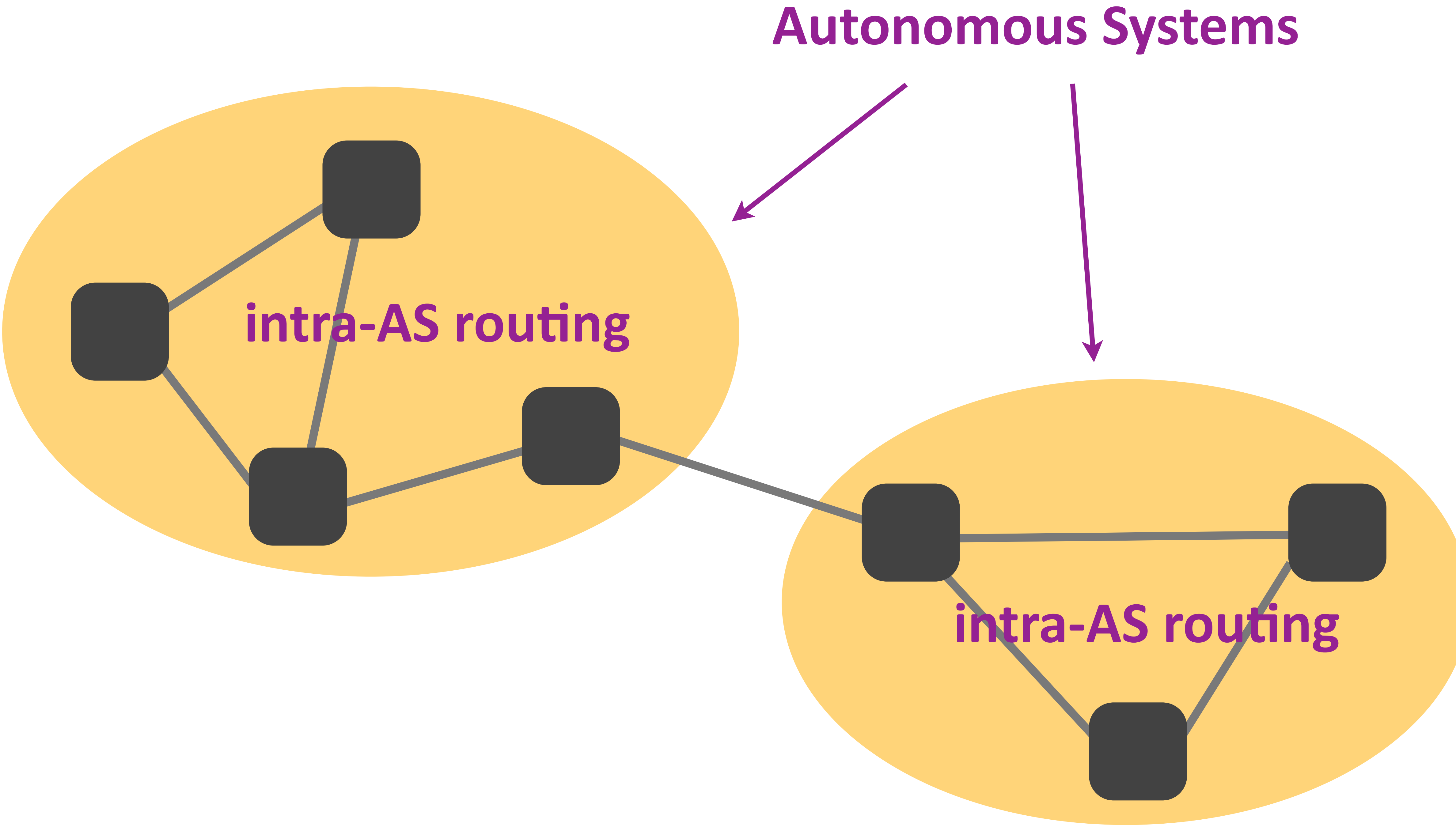
dest.	out. link
8.0.0.0/8	1
5.0.0.0/8	?


dest. | out. link
5.0.0.0/8 | 3
8.0.0.0/8 | 1
dest. | out. link
8.0.0.0/8 | 1
5.0.0.0/8 | 3
v
3
2
1
3
5.0.0.0/8
8.0.0.0/8
router u: 5.0.0.0/8
router z: 8.0.0.0/8
dest. | next hop
z | v
dest. | next hop
u | v

Internet routing challenges

- Scale
  - *link-state would cause flooding*
  - *distance-vector would not converge*

- Administrative autonomy
  - *an ISP may not want to do least-cost routing*
  - *may want to hide its link costs from the world*


Autonomous Systems
intra-AS routing
intra-AS routing

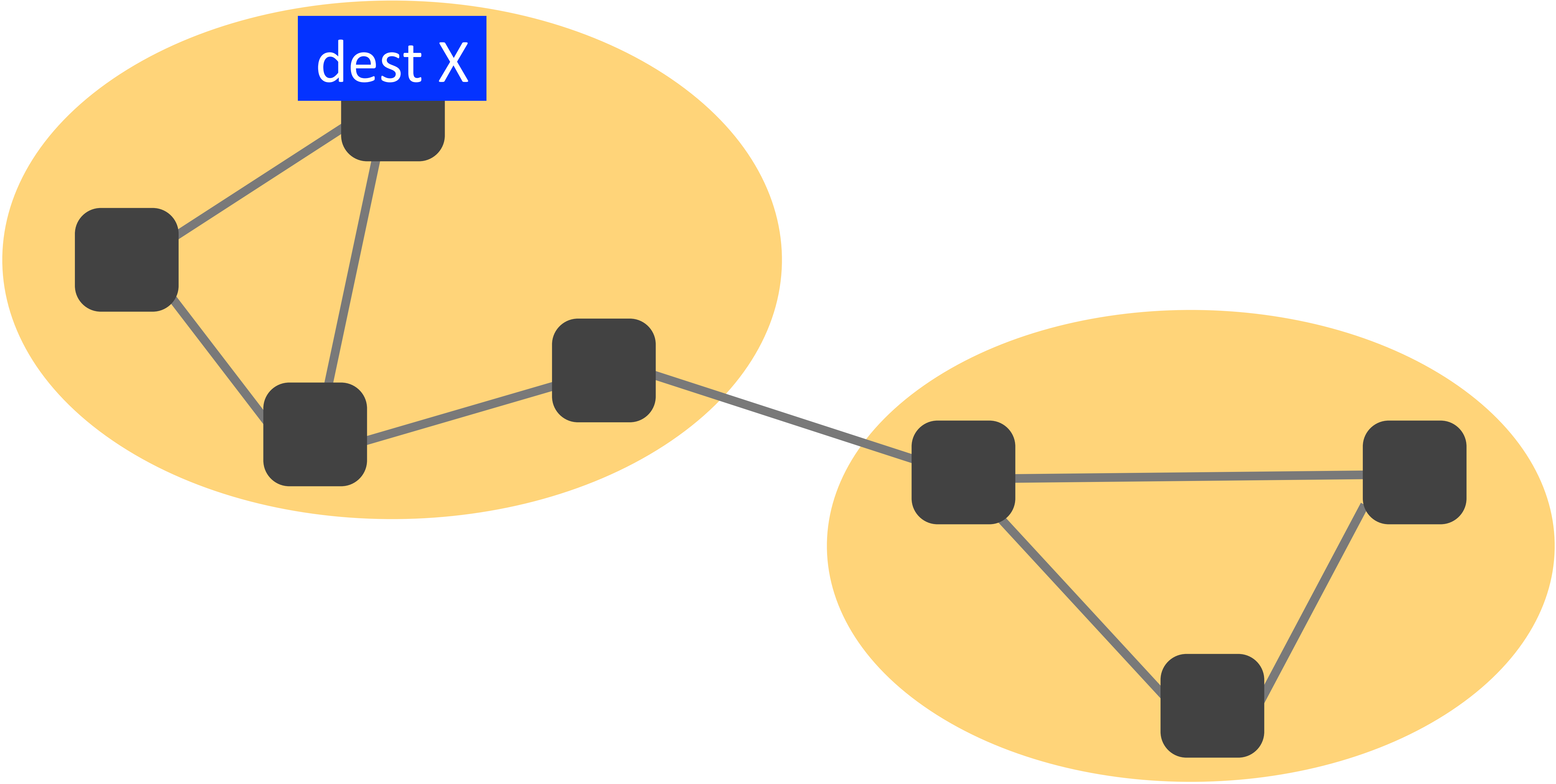
Intra-AS routing

- Run by all routers in the same AS
- Every router learns how to reach every local router and every local IP prefix

Is destination X in the local AS?

yes: route as indicated by intra-AS routing

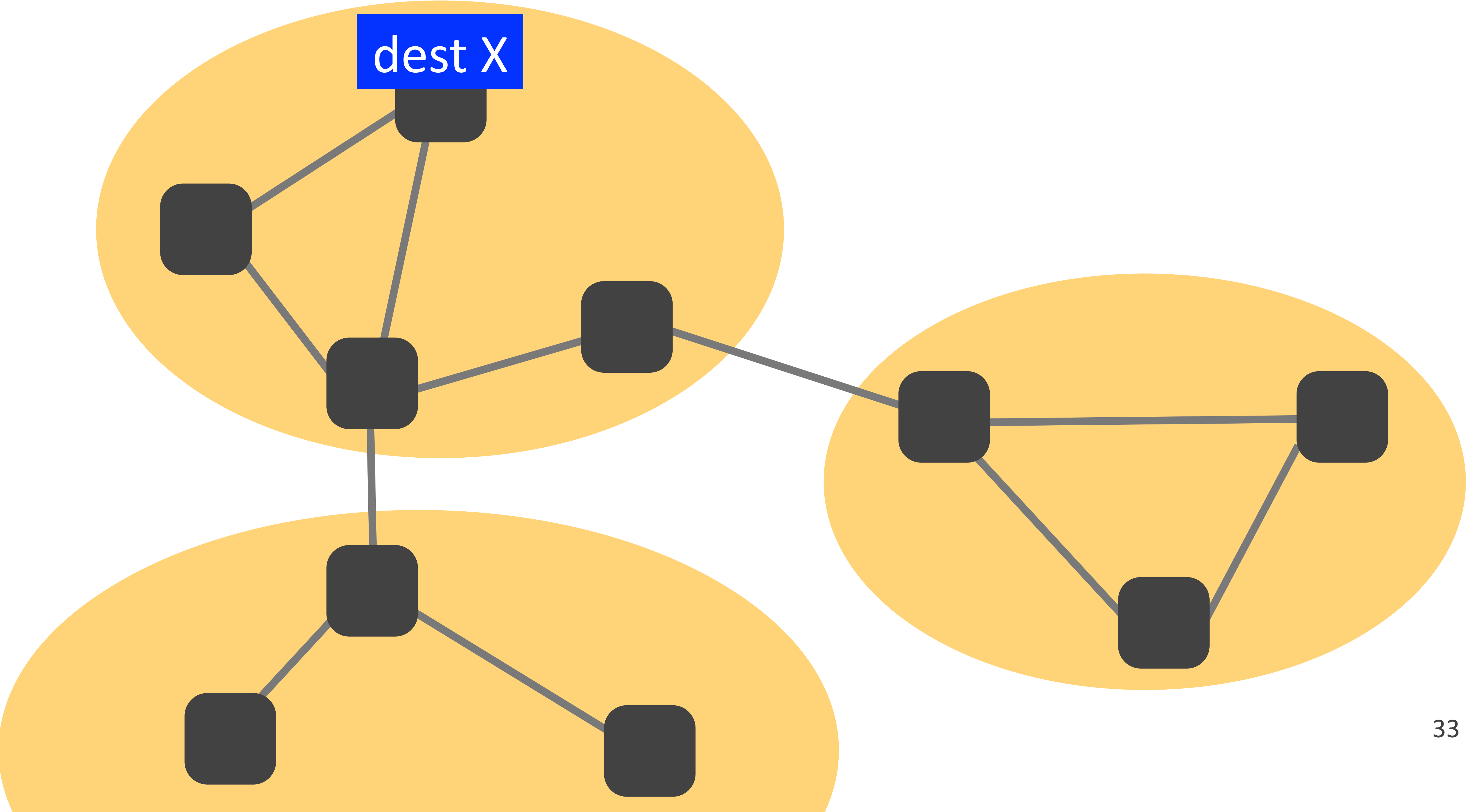
no: send out of the local AS



Is destination X in the local AS?

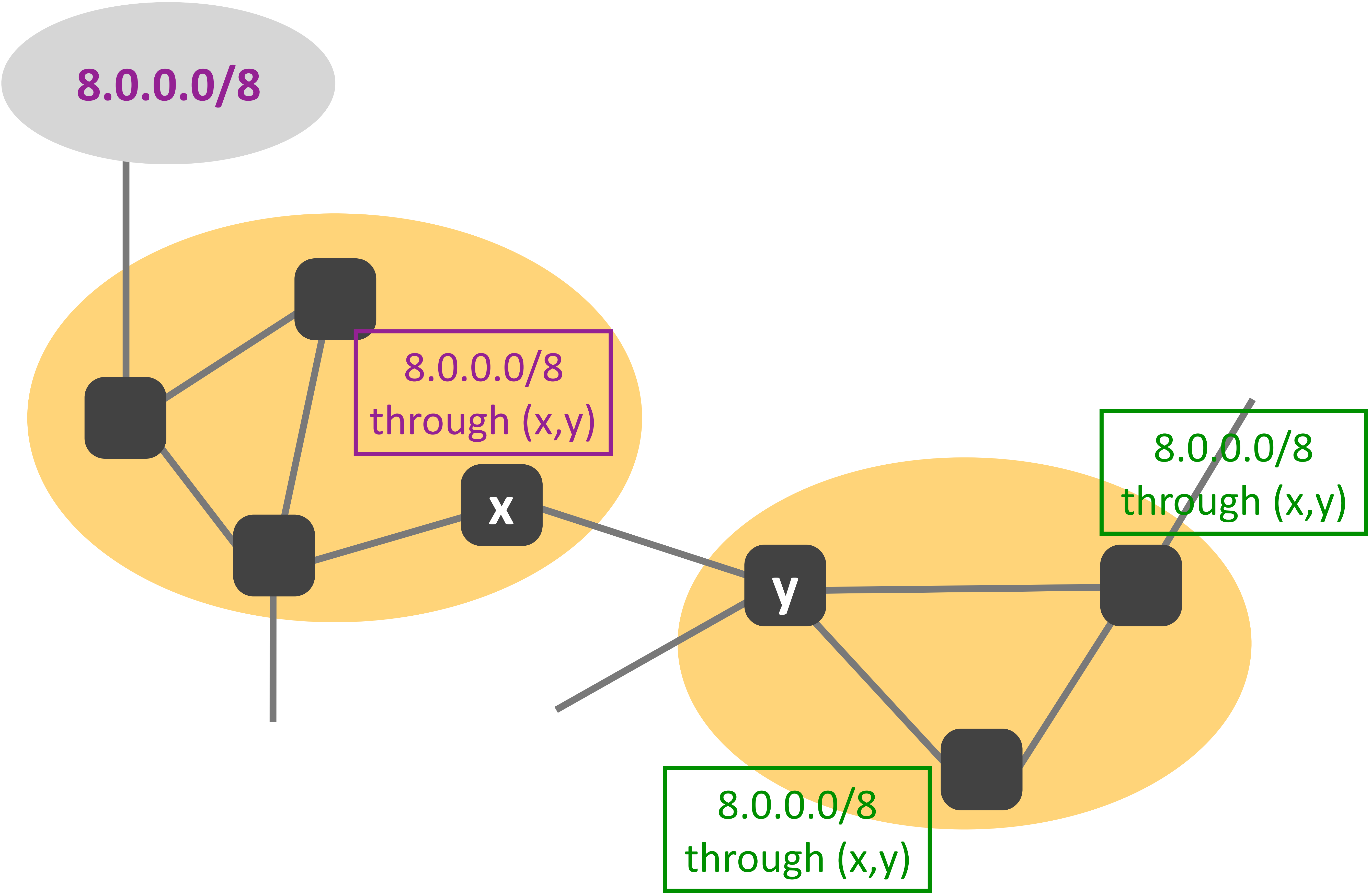
yes: route as indicated by intra-AS routing

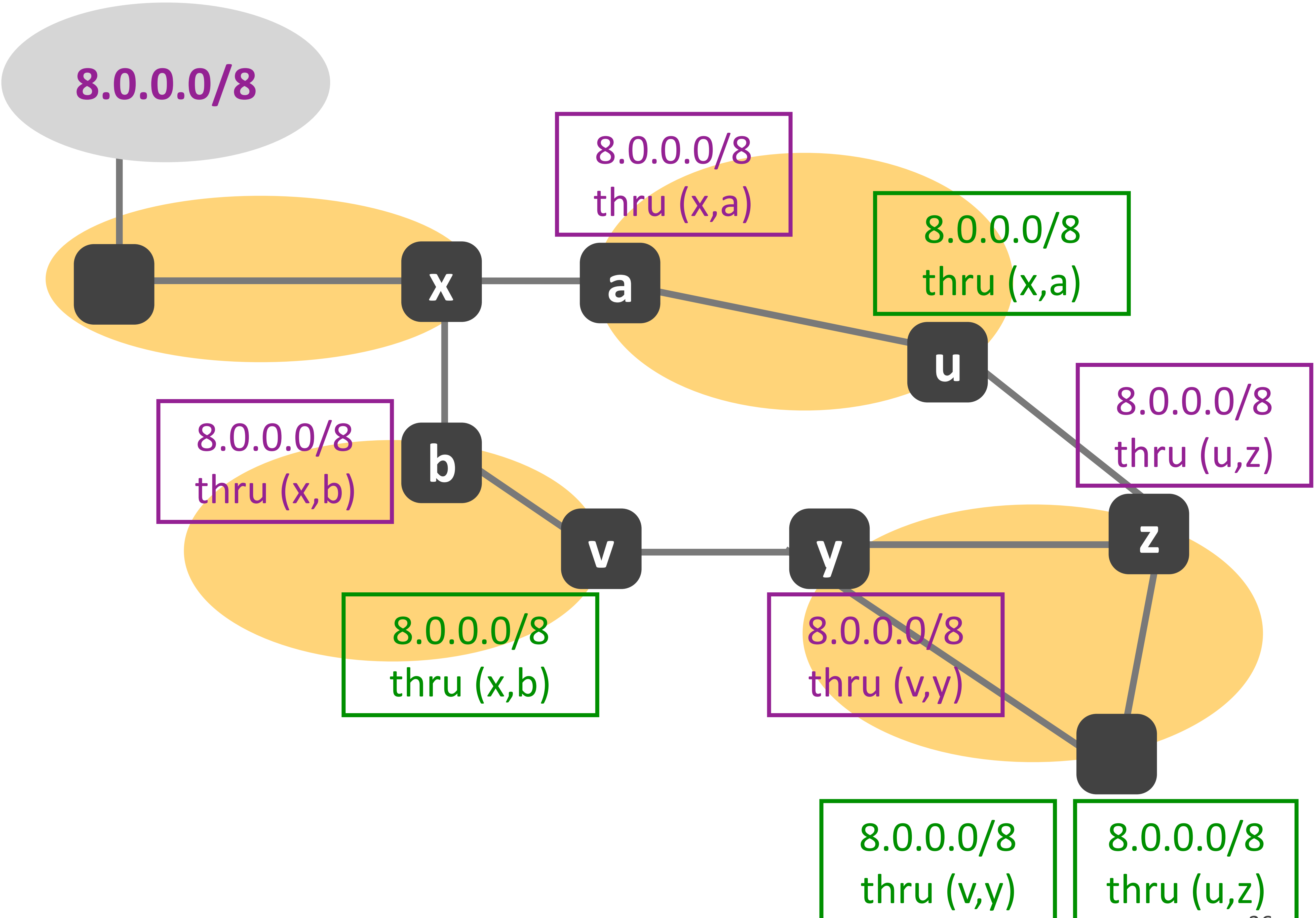
no: send to **the right** AS



Inter-AS routing

- Run by all Internet routers
- Every router learns how to reach every foreign IP prefix


8.0.0.0/8
8.0.0.0/8
through (x,y)
x
y
8.0.0.0/8
through (x,y)
8.0.0.0/8
through (x,y)


8.0.0.0/8
8.0.0.0/8 thru (x,a)
x
a
8.0.0.0/8 thru (x,a)
u
8.0.0.0/8 thru (u,z)
8.0.0.0/8 thru (x,b)
b
v
y
z
8.0.0.0/8 thru (x,b)
8.0.0.0/8 thru (v,y)
8.0.0.0/8 thru (v,y)
8.0.0.0/8 thru (u,z)

Internet routing

- Internet organized in Autonomous Systems (ASes)
- Within each AS: intra-AS routing
  - *every router learns how to reach every local router and every local IP prefix*
- Across ASes: inter-AS routing
  - *for every foreign IP prefix,*
  - *every router identifies a local router or a directly connected foreign router*
  - *which knows how to reach that foreign IP prefix*

Internet routing protocols

- Intra-AS: RIP, OSPF
- Inter-AS: Border Gateway Protocol (BGP)

Internet routing challenges

- Scale
  - *link-state would cause flooding*
  - *distance-vector would not converge*

- Administrative autonomy
  - *an ISP may not want to do least-cost routing*
  - *may want to hide its link costs from the world*

Solution: hierarchy

- Scale: an Internet router does not need to learn how to reach every other Internet router
  - *every router in local AS*
  - *one router (local or directly connected foreign) per foreign IP prefix*

- Administrative autonomy:
  an AS chooses its own intra-AS routing

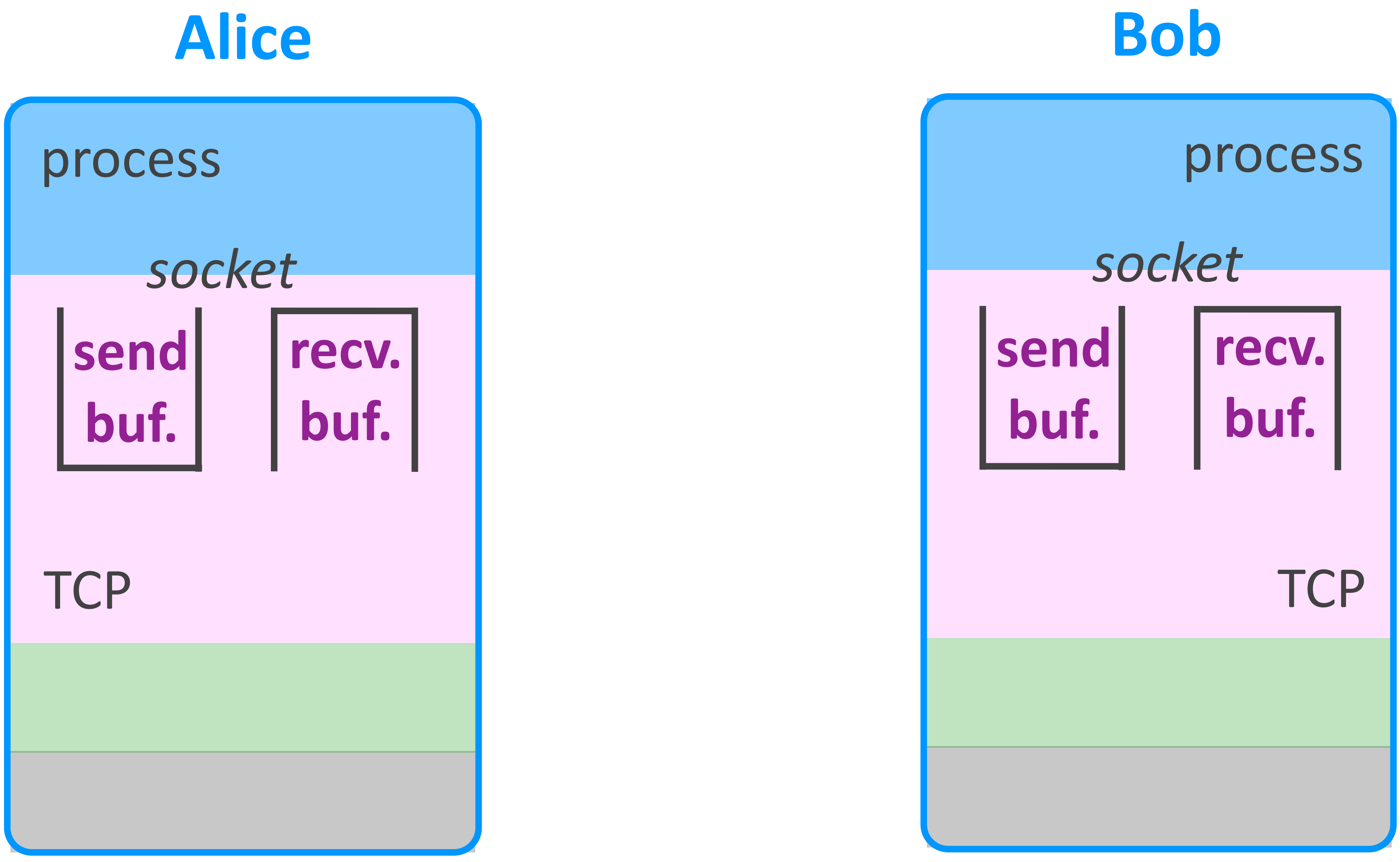
Transport layer

Outline

- TCP connection
- Reliability
- Flow control
- Security
- Congestion control

Outline

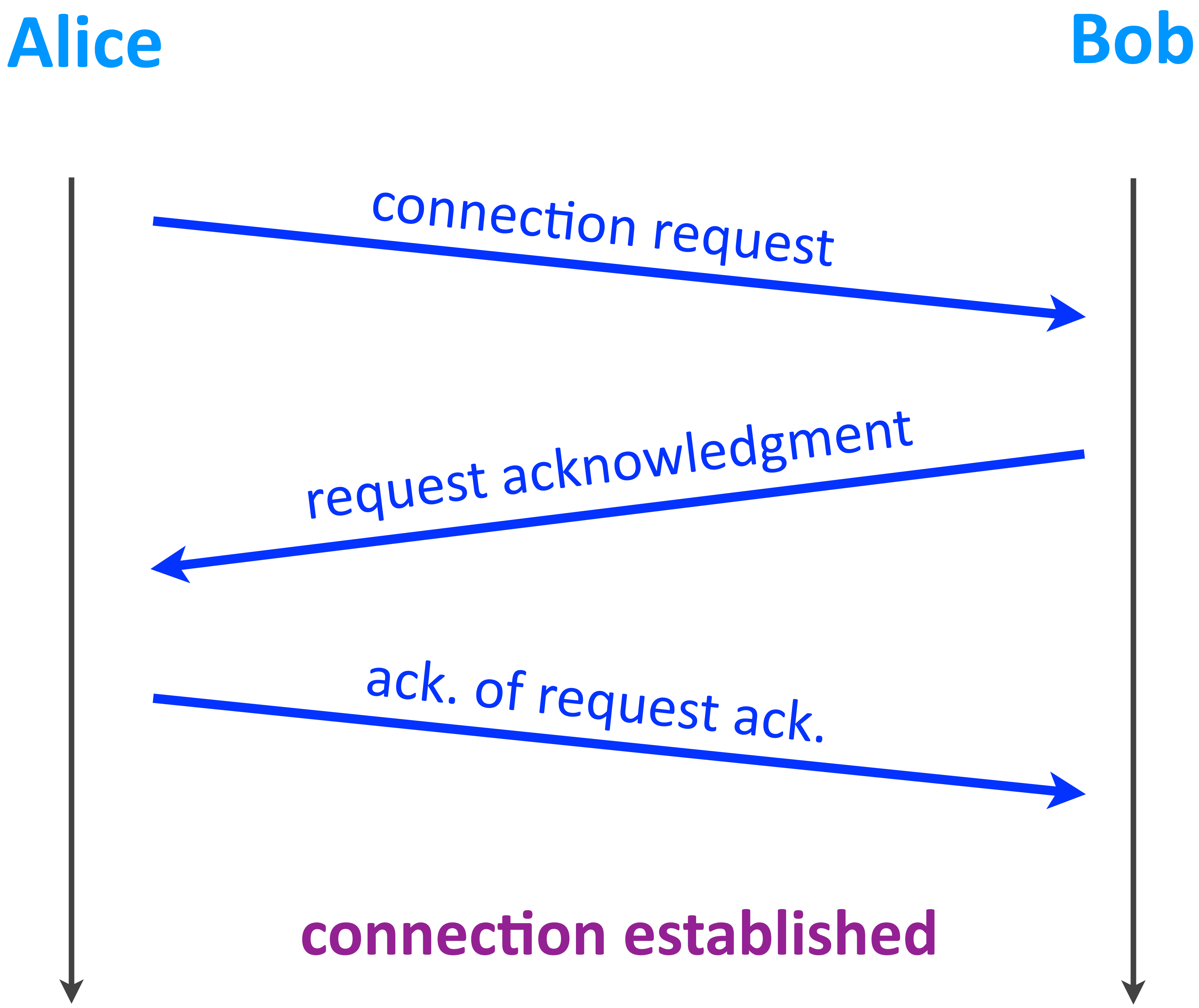
- TCP connection
- Reliability
- Flow control
- (Loose ends)
- Security
- Congestion control


Alice
process
socket
send buf.
recv. buf.
TCP
Bob
process
socket
send buf.
recv. buf.
TCP

What is a TCP connection?

- Sockets
  - *pass data between app-layer process & TCP*
- Buffers
  - *store sent/received data*
  - *(bidirectional or "full-duplex" communication)*
- Variables
  - *will discuss in a moment*

A set of resources allocated at the end-systems



Alice
Bob
connection request
request acknowledgment
ack. of request ack.
connection established

How is it established?

- 3-way handshake between end-systems
  - *“client” = the initiating process*
  - *“server” = the other process*
  - *(but data may flow both directions)*

Outline

- TCP connection
- Reliability
- Flow control
- Security
- Congestion control

SEQ & ACK numbers

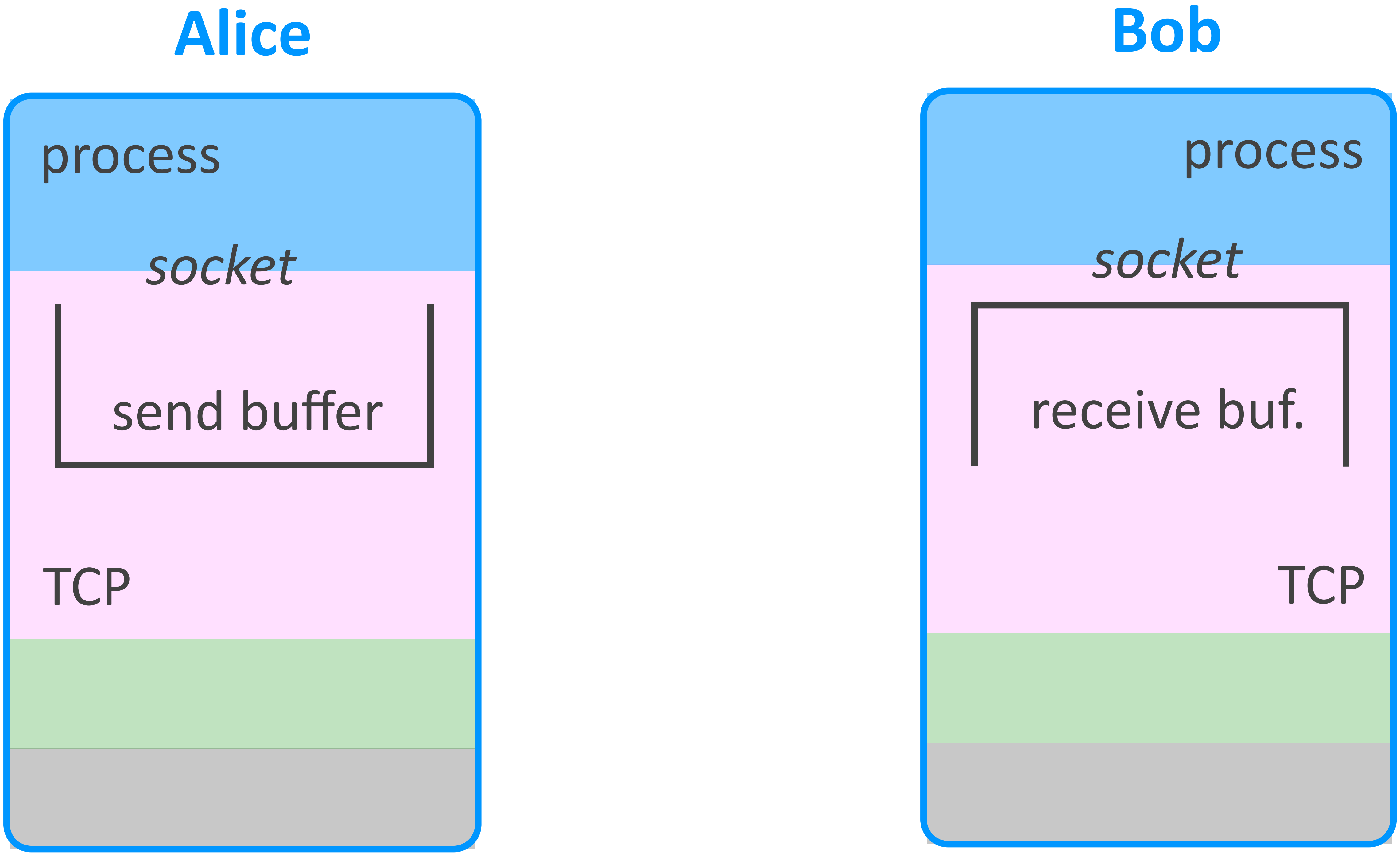
- TCP data bytes are implicitly numbered
- Sequence number (TCP header field)
  - *# of first byte of data*
- ACK number (TCP header field)
  - *# of oldest byte missing*
  - *cumulative*

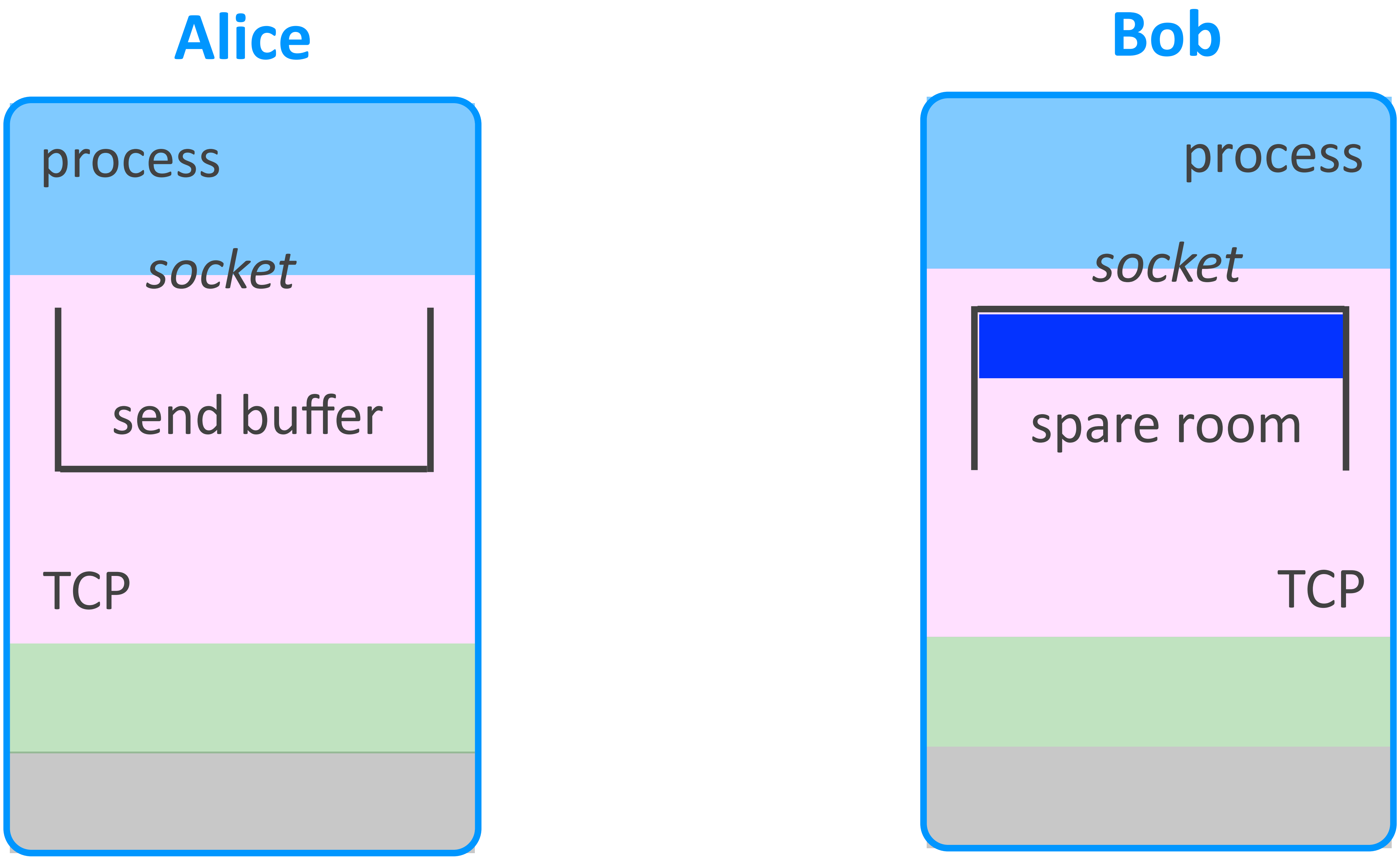
Timeout & retransmit

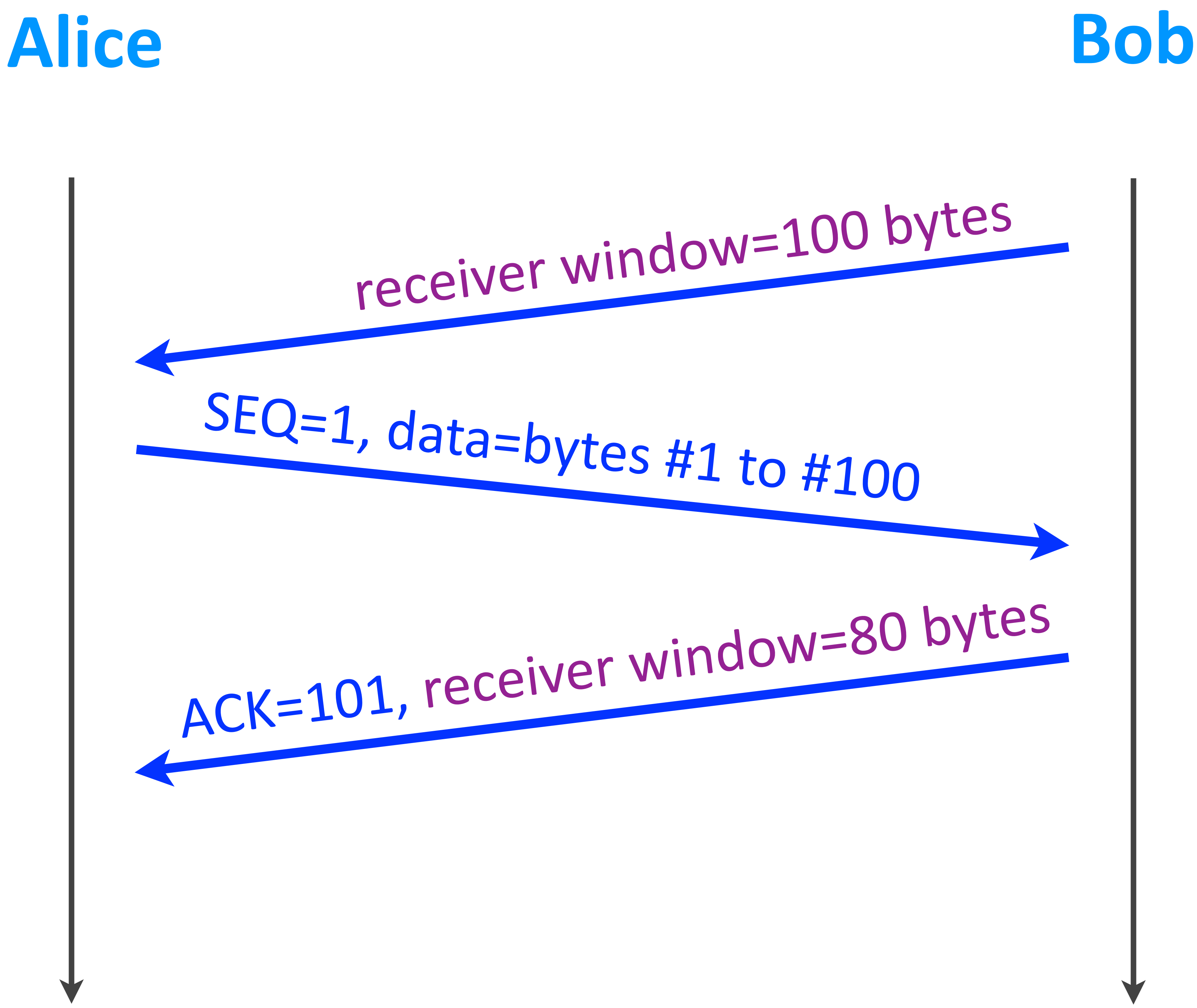
- Sender times out
  - *segment not ACK-ed within timeout*
- Sender retransmits the segment
  with oldest un-ACKed sequence number

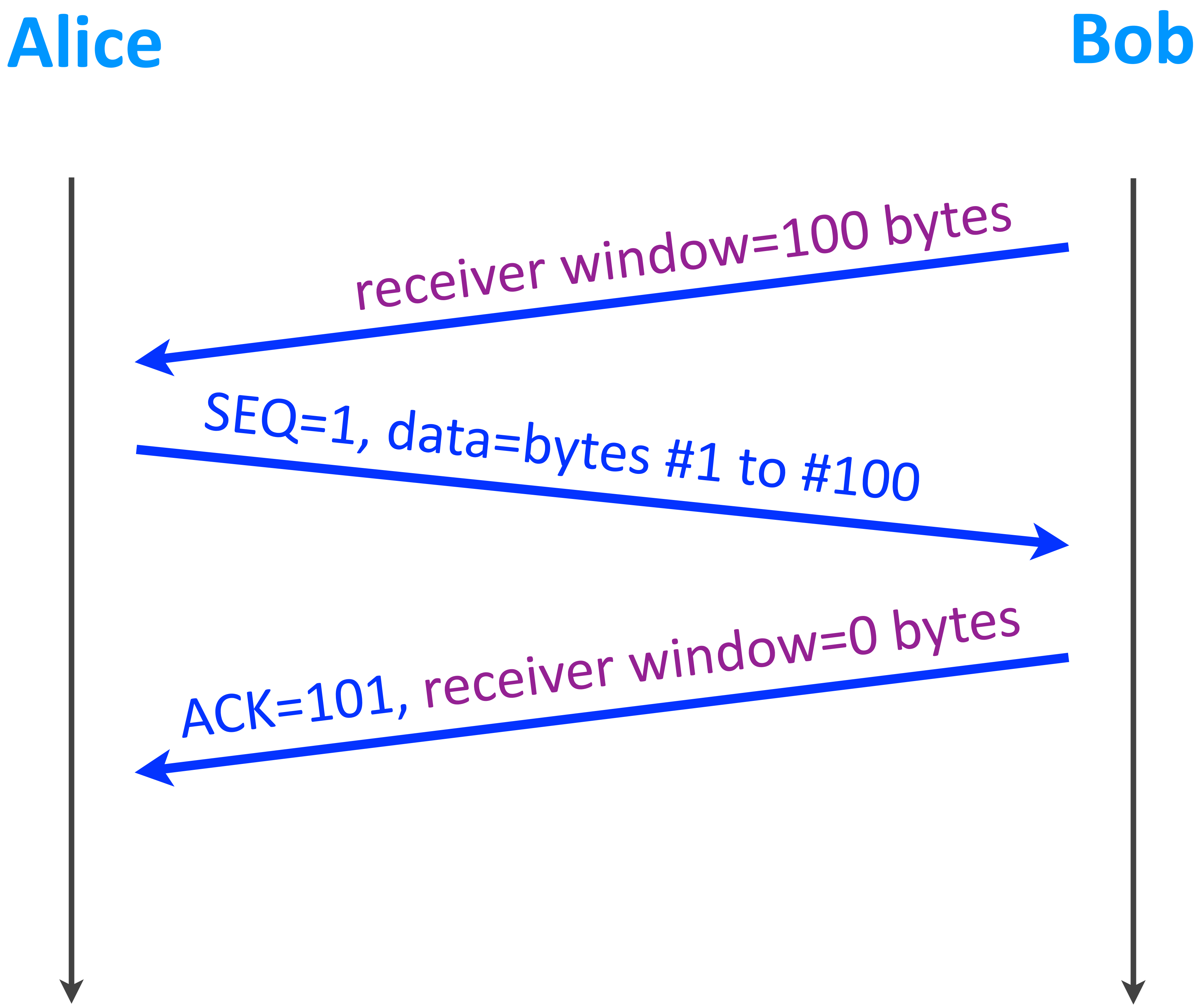
Outline

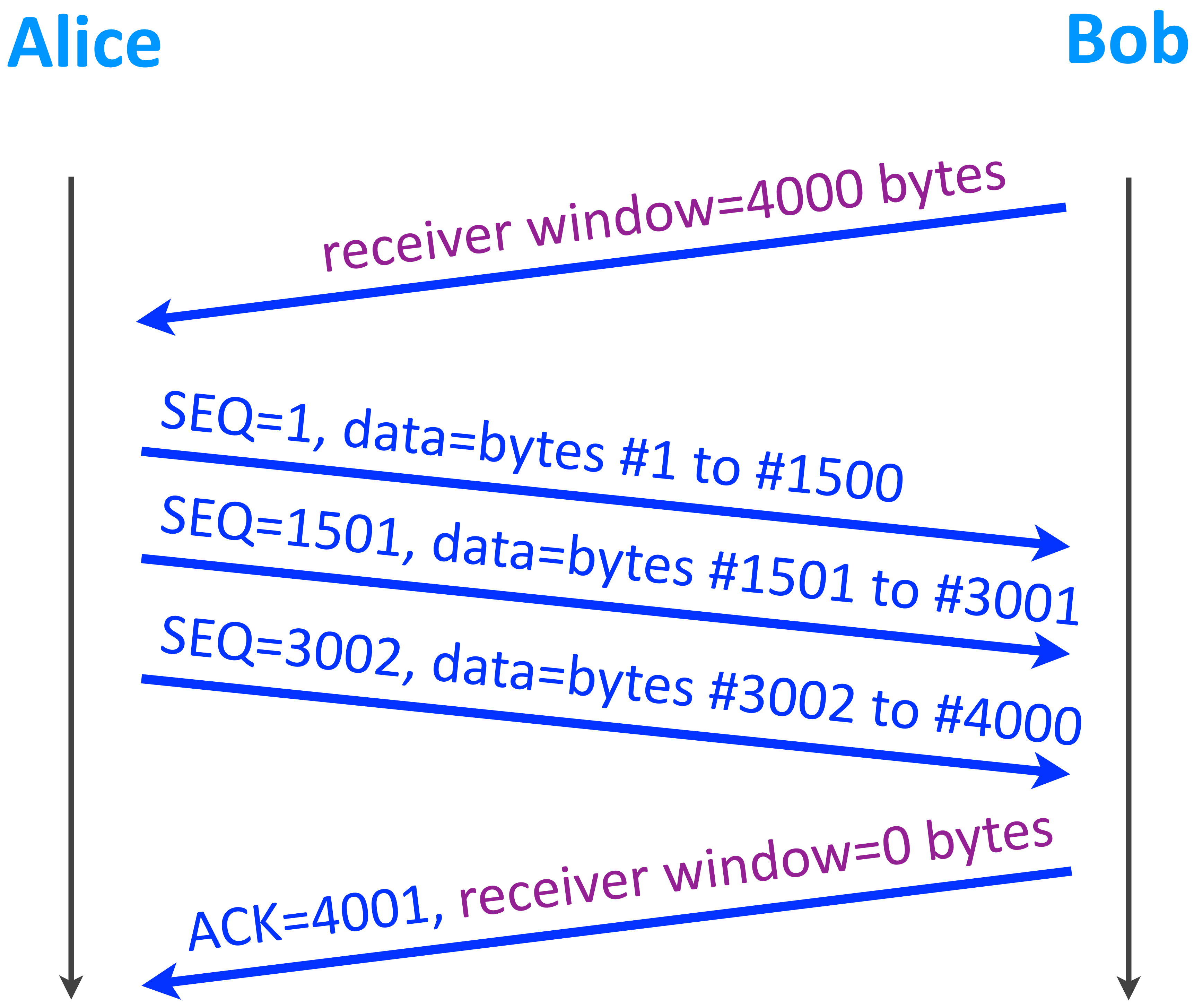
- TCP connection
- Reliability
- Flow control
- Security
- Congestion control


Alice
process
socket
send buffer
TCP
Bob
process
socket
receive buf.
TCP

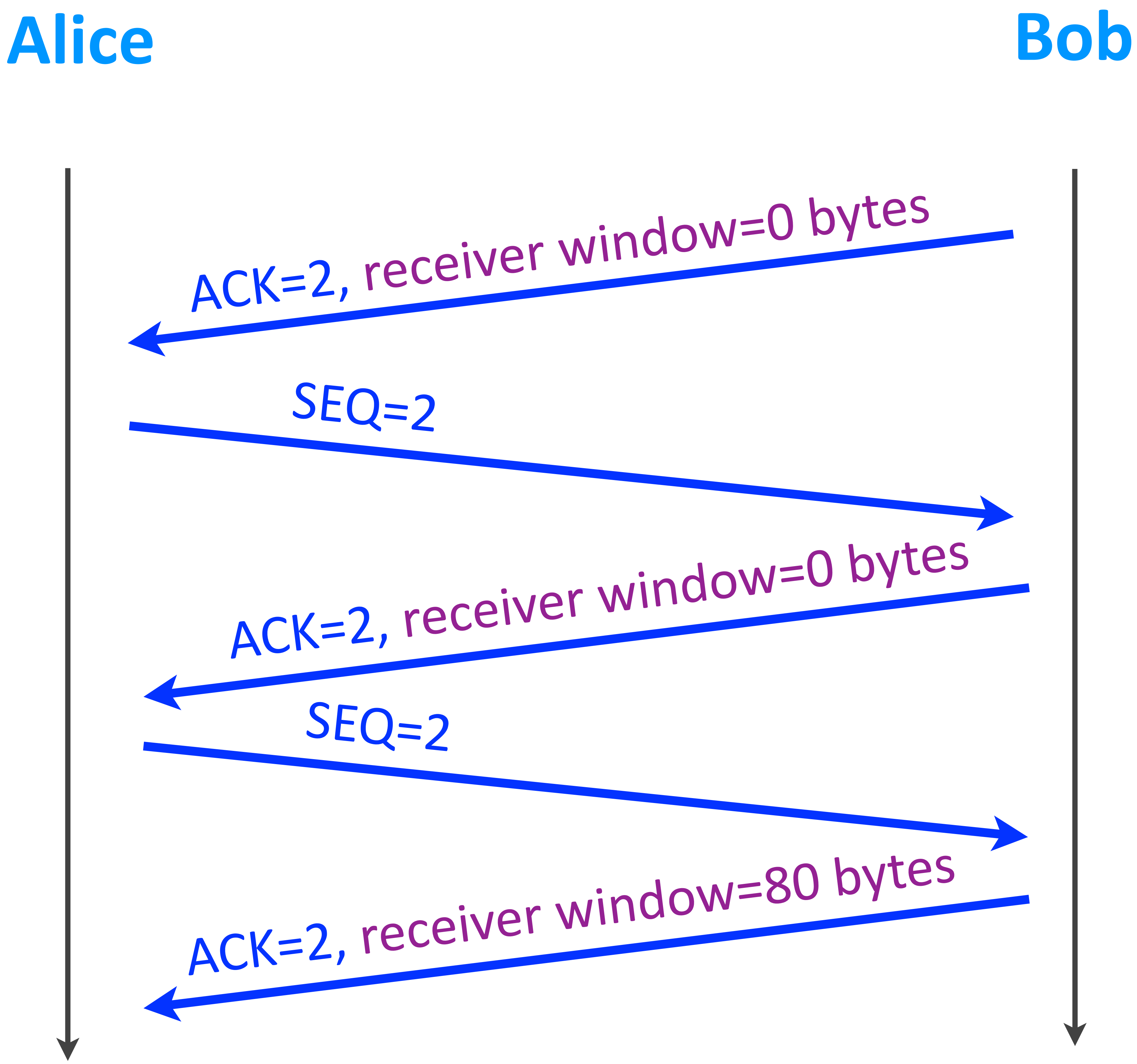

Alice
process
socket
send buffer
TCP
Bob
process
socket
spare room
TCP


Alice
Bob
receiver window=100 bytes
SEQ=1, data=bytes #1 to #100
ACK=101, receiver window=80 bytes


Alice
Bob
receiver window=100 bytes
SEQ=1, data=bytes #1 to #100
ACK=101, receiver window=0 bytes



Alice
Bob
receiver window=4000 bytes
SEQ=1, data=bytes #1 to #1500
SEQ=1501, data=bytes #1501 to #3001
SEQ=3002, data=bytes #3002 to #4000
ACK=4001, receiver window=0 bytes


Alice
Bob
ACK=2, receiver window=0 bytes
SEQ=2
ACK=2, receiver window=0 bytes
SEQ=2
ACK=2, receiver window=80 bytes

Flow control

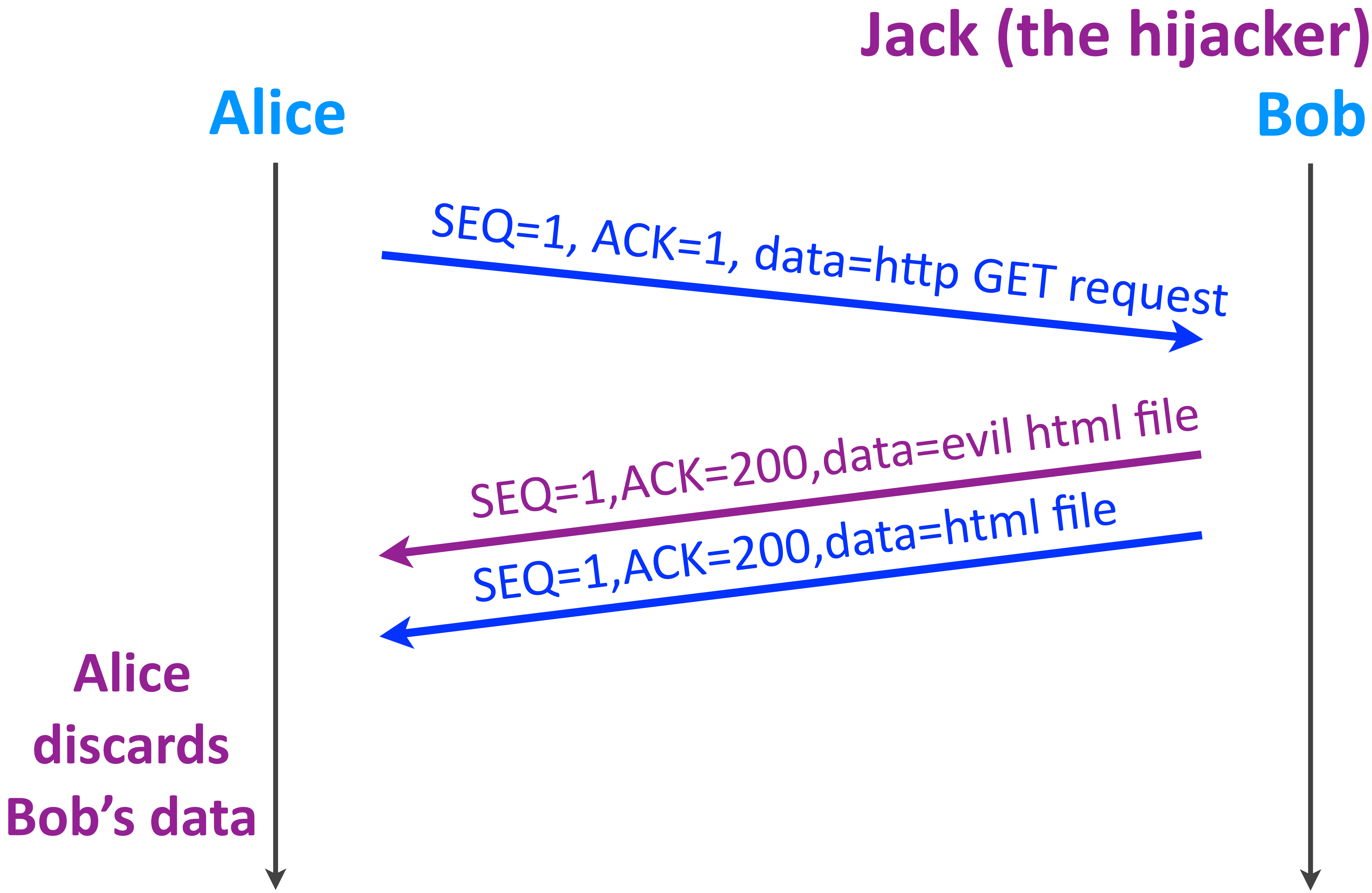
- Receiver provides receiver window
  - *equal to free space in TCP receive buffer*
  - *specifies how many bytes it can receive*

- Sender sends up to this # of bytes
  - *must wait for receiver window to "open"*

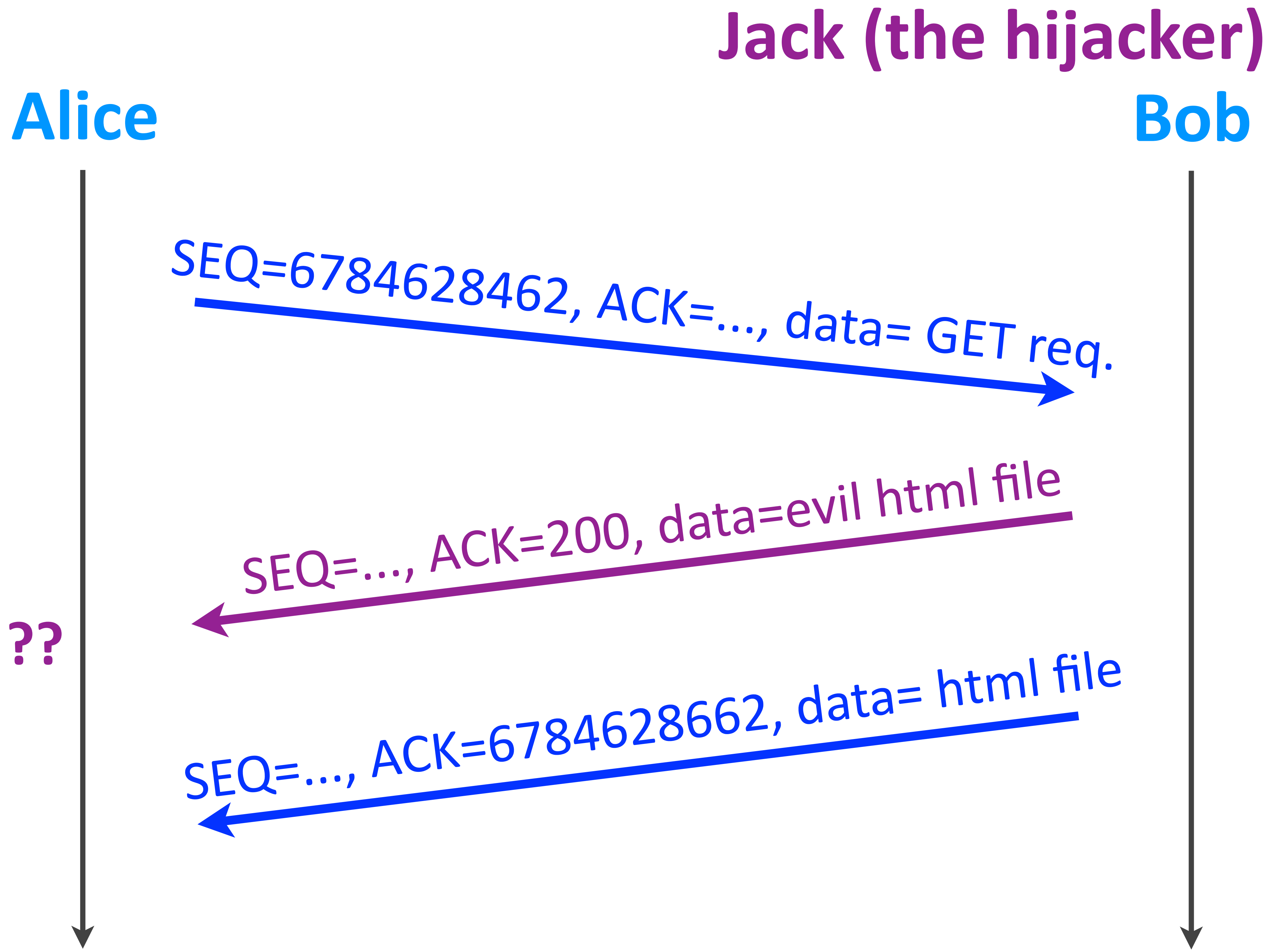
Slows down sender based on receiver status

Outline

- TCP connection
- Reliability
- Flow control
- Security
- Congestion control



Jack (the hijacker)
Alice
Bob
SEQ=1, ACK=1, data=http GET request
SEQ=1,ACK=200,data=evil html file
SEQ=1,ACK=200,data=html file
Alice discards Bob's data

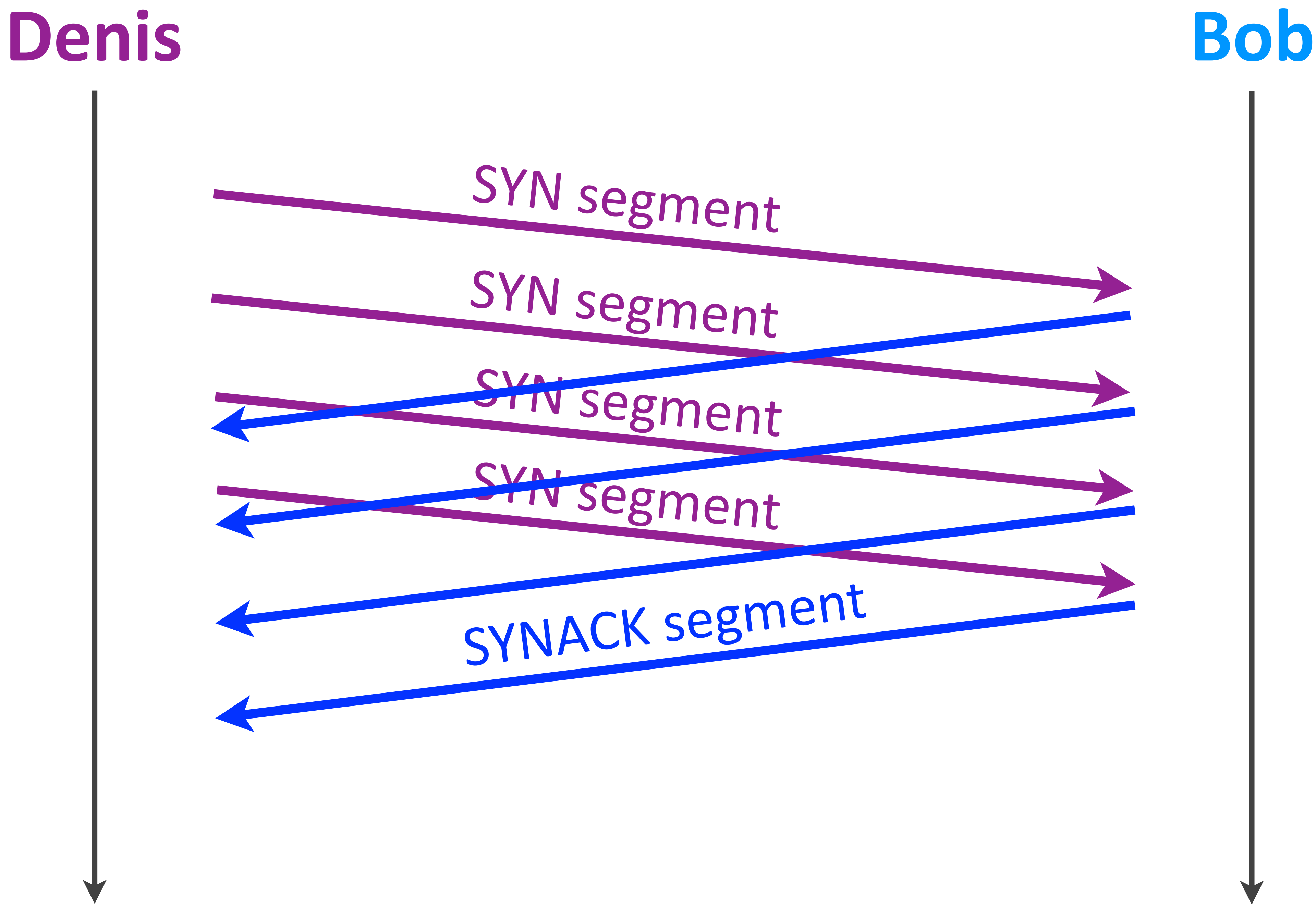

Jack (the hijacker)
Alice
Bob
SEQ=6784628462, ACK=..., data= GET req.
SEQ=..., ACK=200, data=evil html file
??
SEQ=..., ACK=6784628662, data= html file

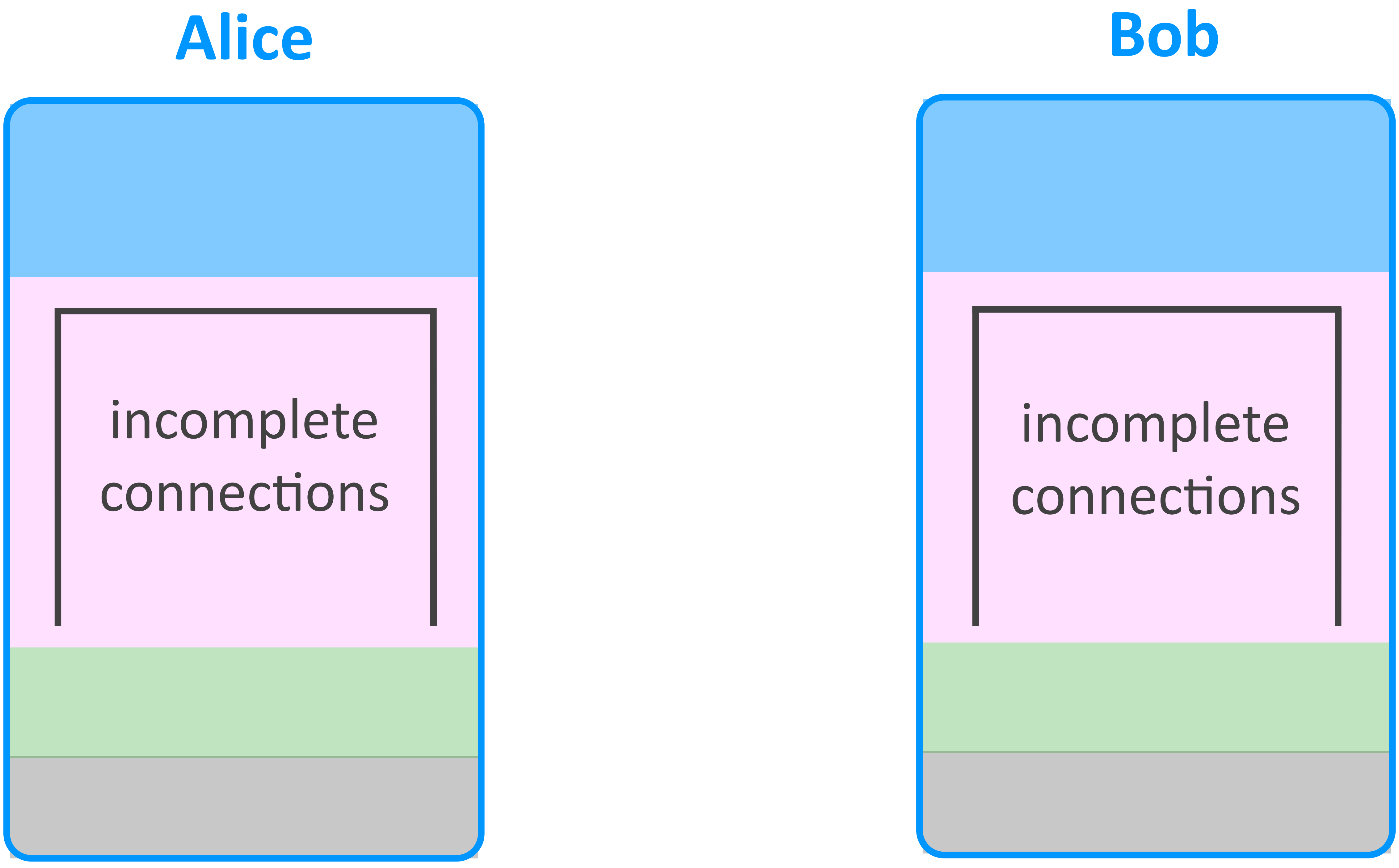
TCP hijacking

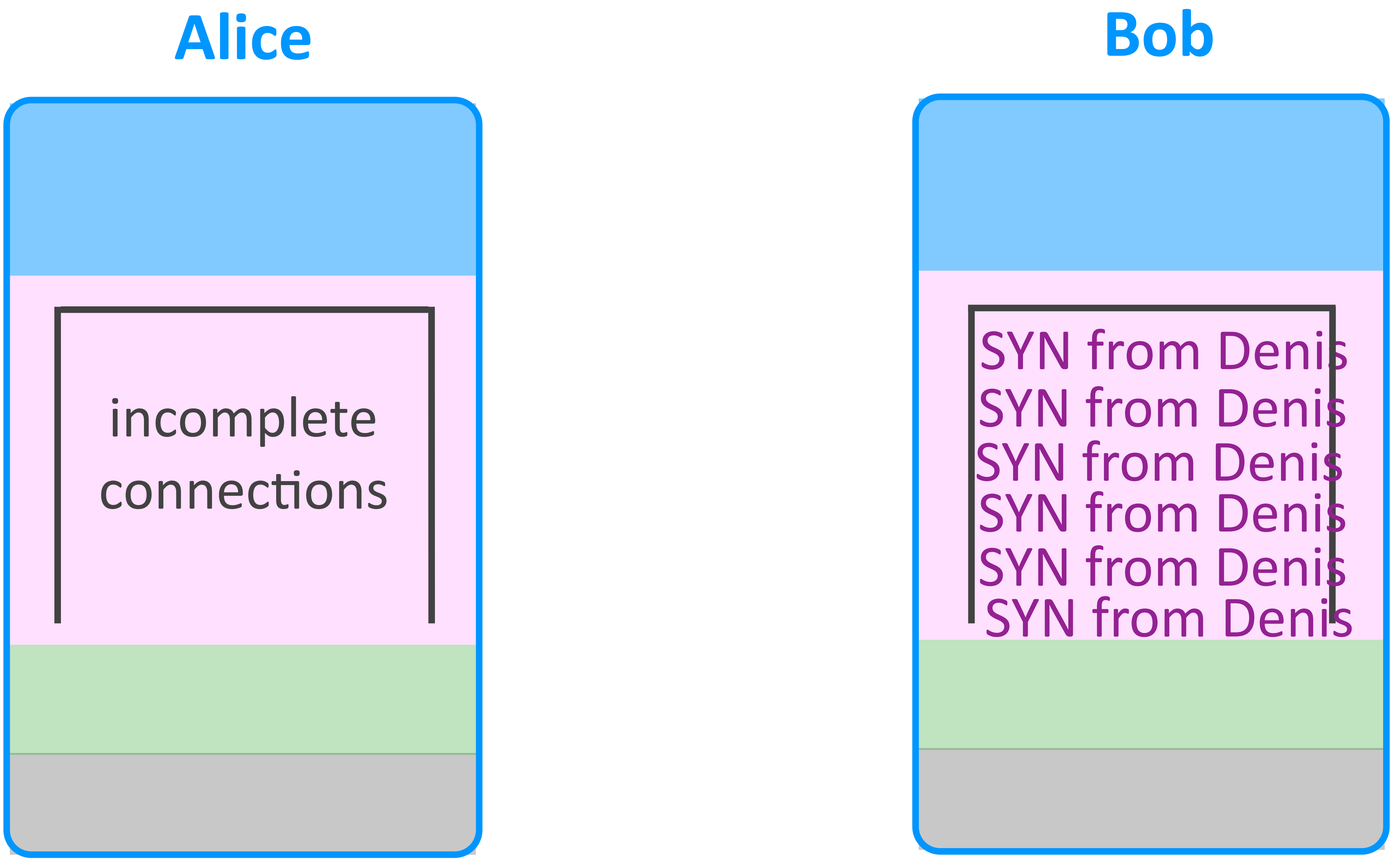
- Attack: impersonate one of the parties
  & provide fake content

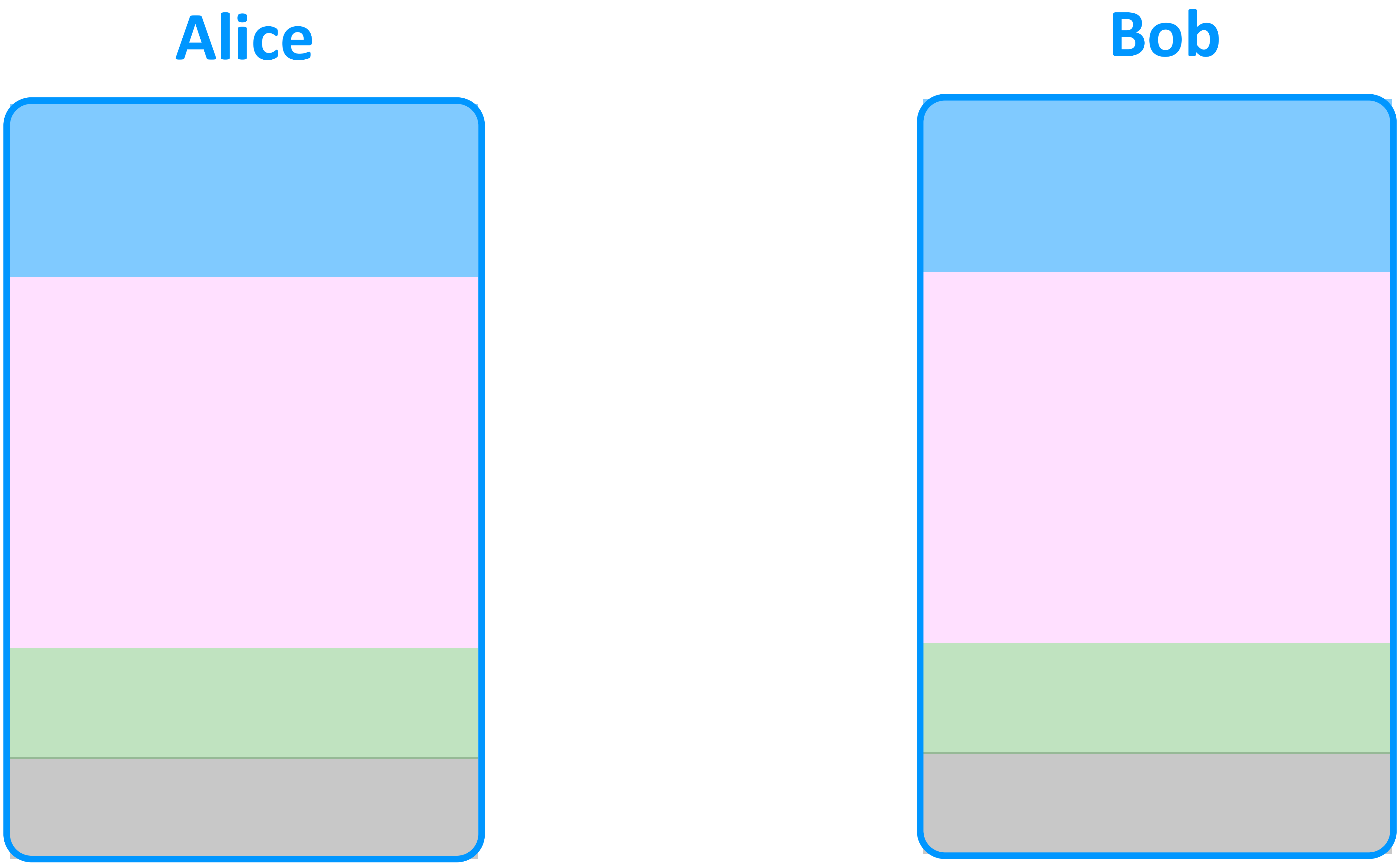
- Defense: randomize sequence numbers

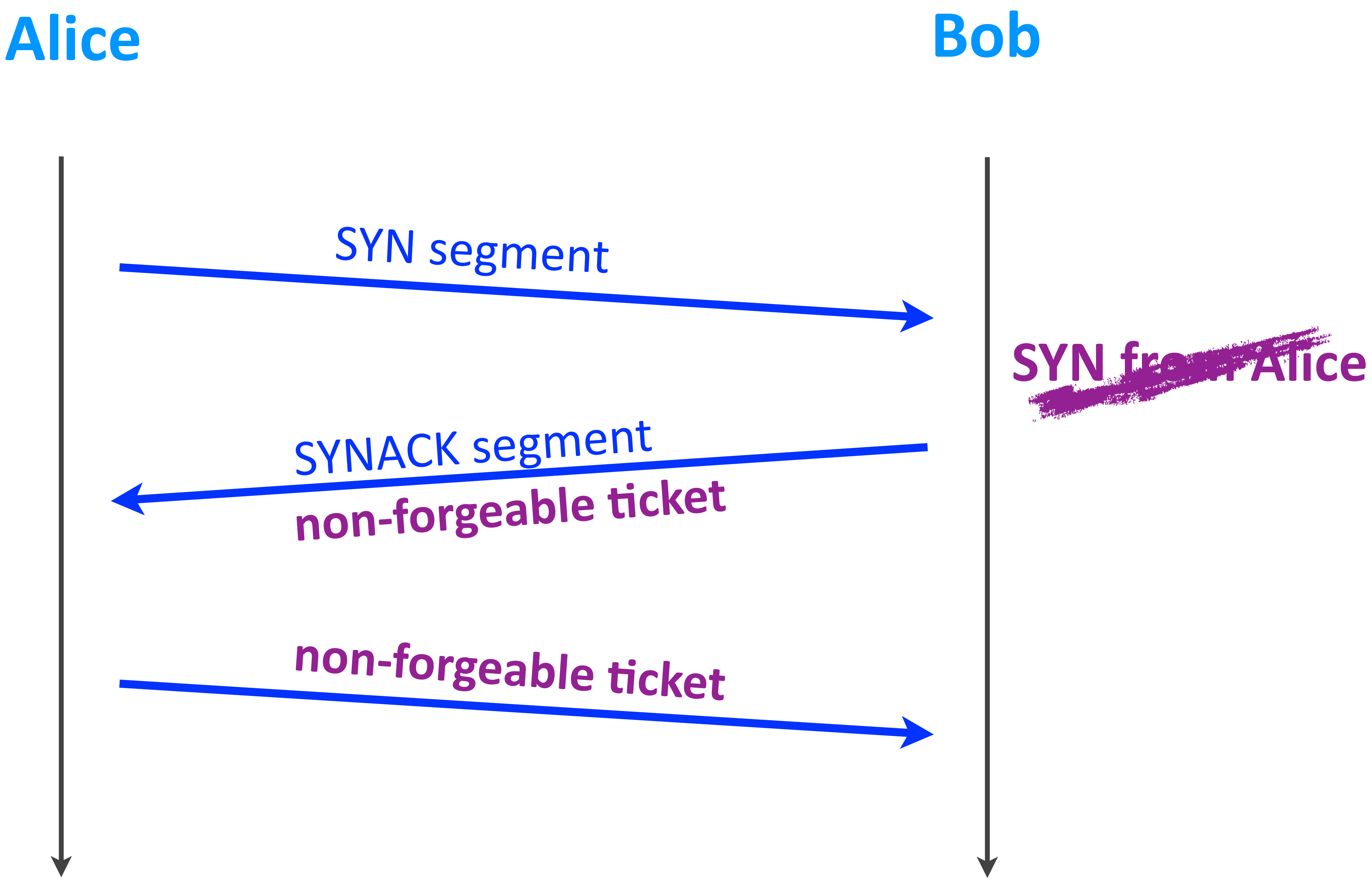
Make segment content unpredictable


Denis
Bob
SYN segment
SYN segment
SYN segment
SYN segment
SYNACK segment


Alice
incomplete connections
Bob
incomplete connections


Alice
incomplete
connections
Bob
SYN from Denis
SYN from Denis
SYN from Denis
SYN from Denis
SYN from Denis
SYN from Denis


Alice
Bob



Alice
Bob
SYN segment
SYN from Alice
SYNACK segment
non-forgeable ticket
non-forgeable ticket

SYN flooding

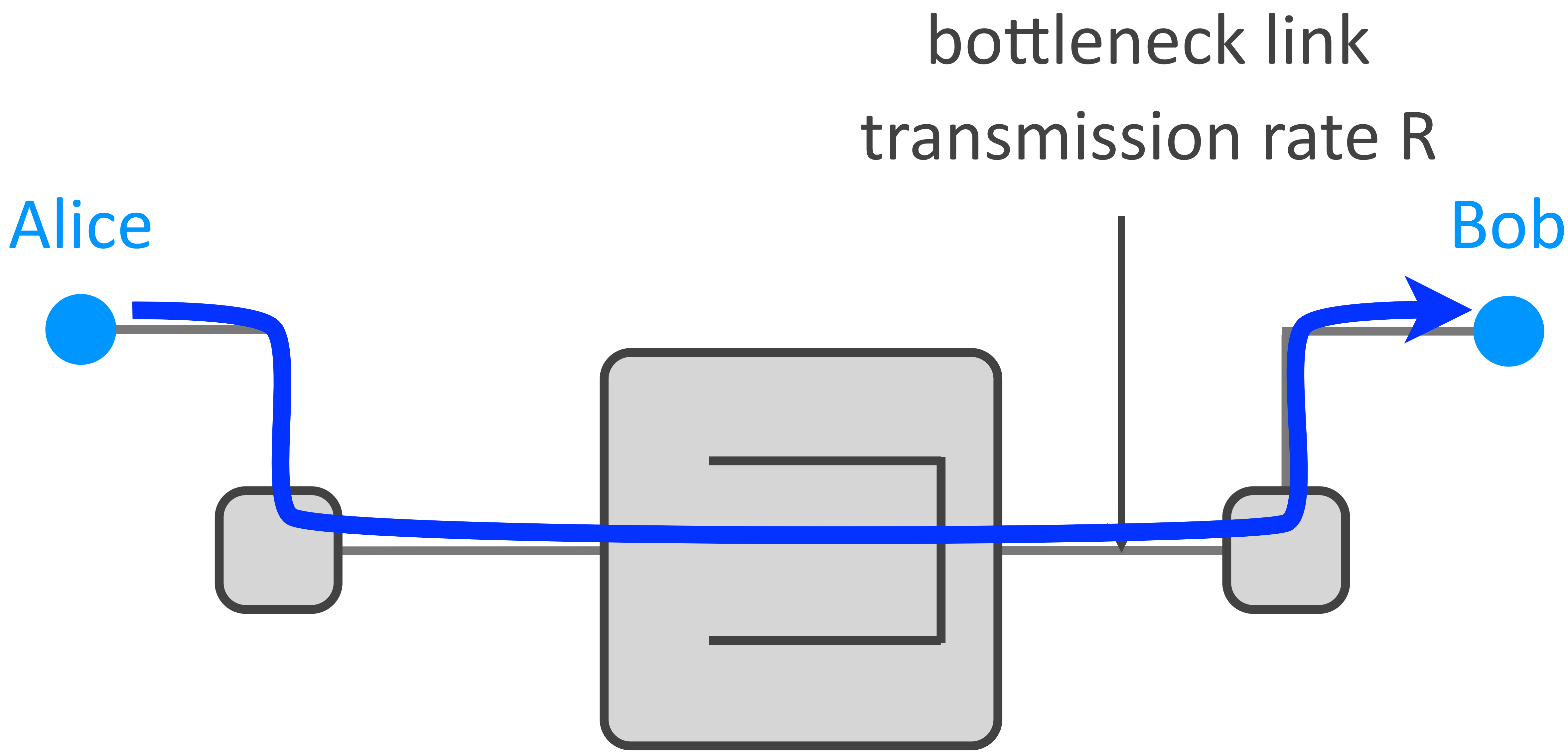
- Attack: exhaust the SYN buffer

- Defense: get rid of the SYN buffer
  instead use non-forgeable ticket

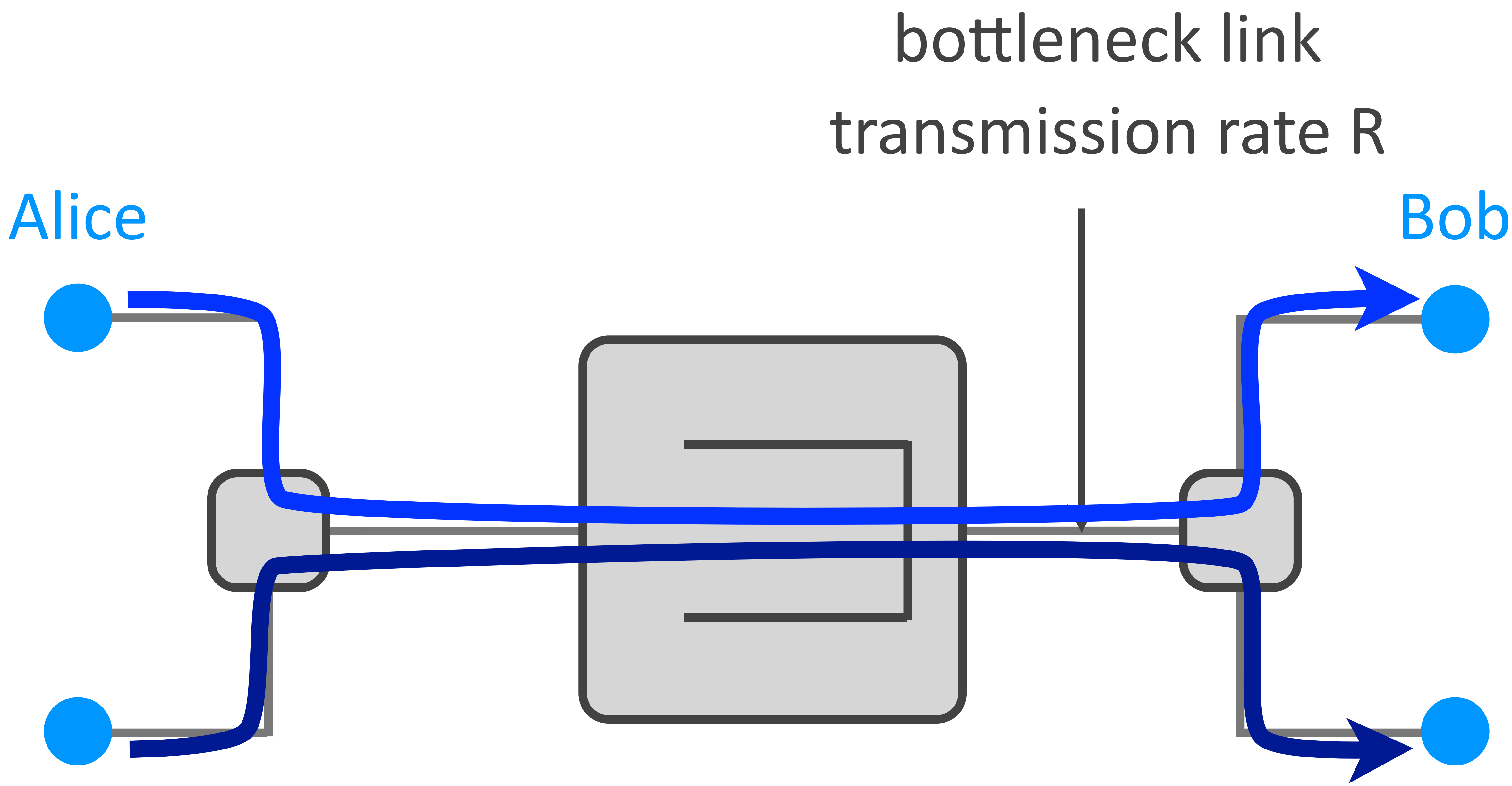
Pass the state to the TCP client

Outline

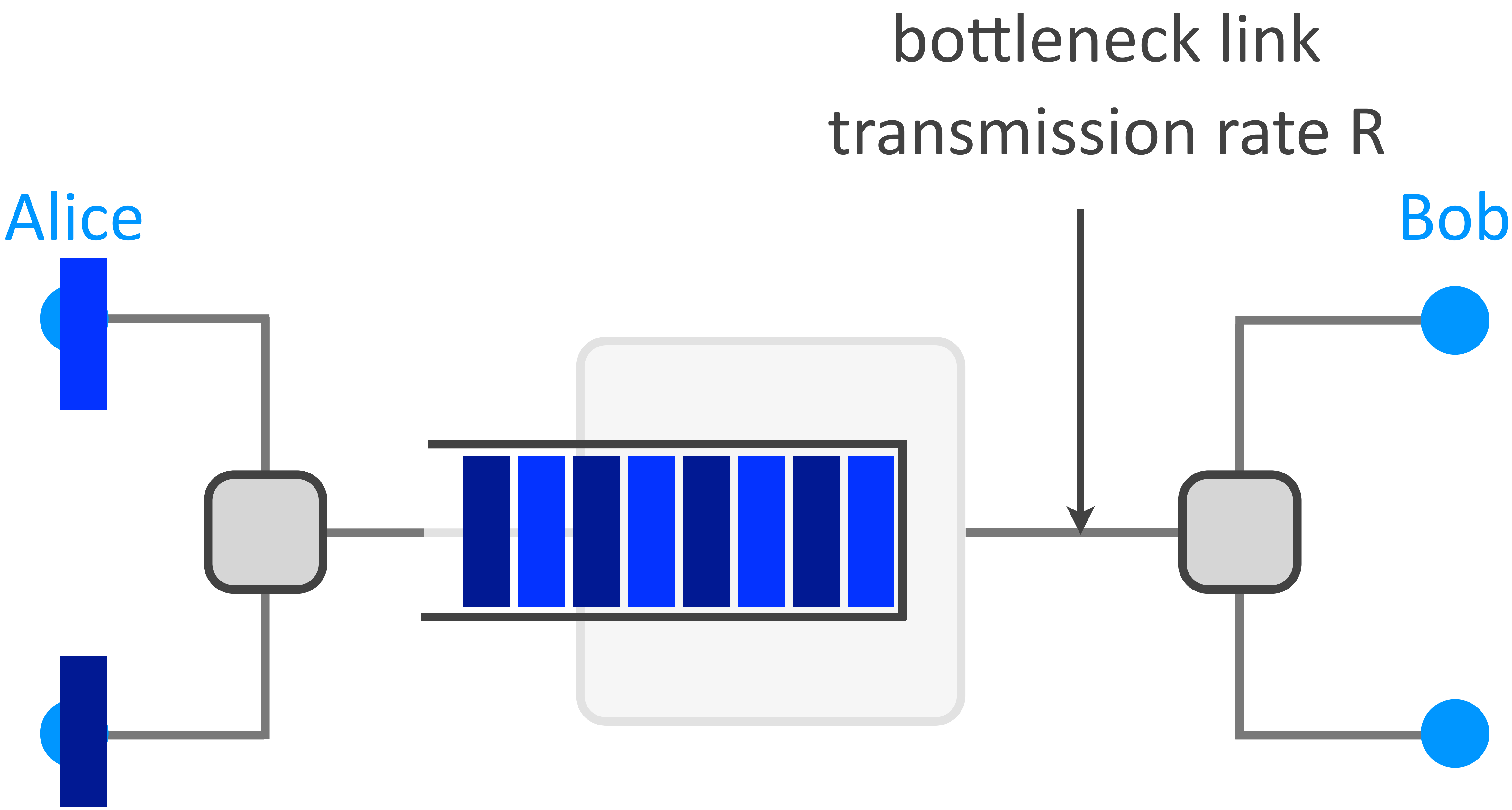
- TCP connection
- Reliability
- Flow control
- Security
- Congestion control



Alice’s max throughput is R



Alice’s max throughput is R/2

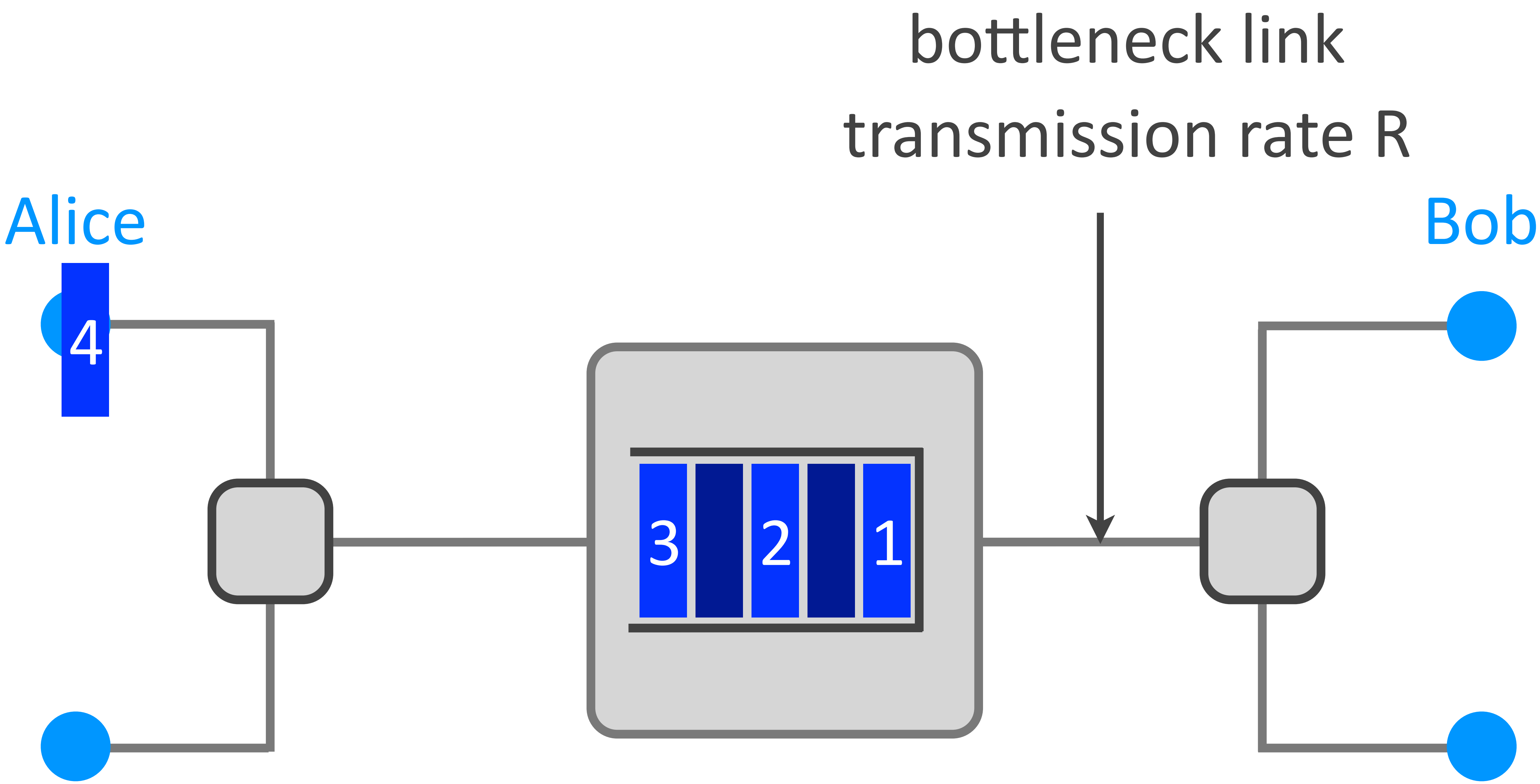


If Alice's transport transmits at rate R/2,

she experiences high queuing delay

Bad congestion effects

- Long queuing delays



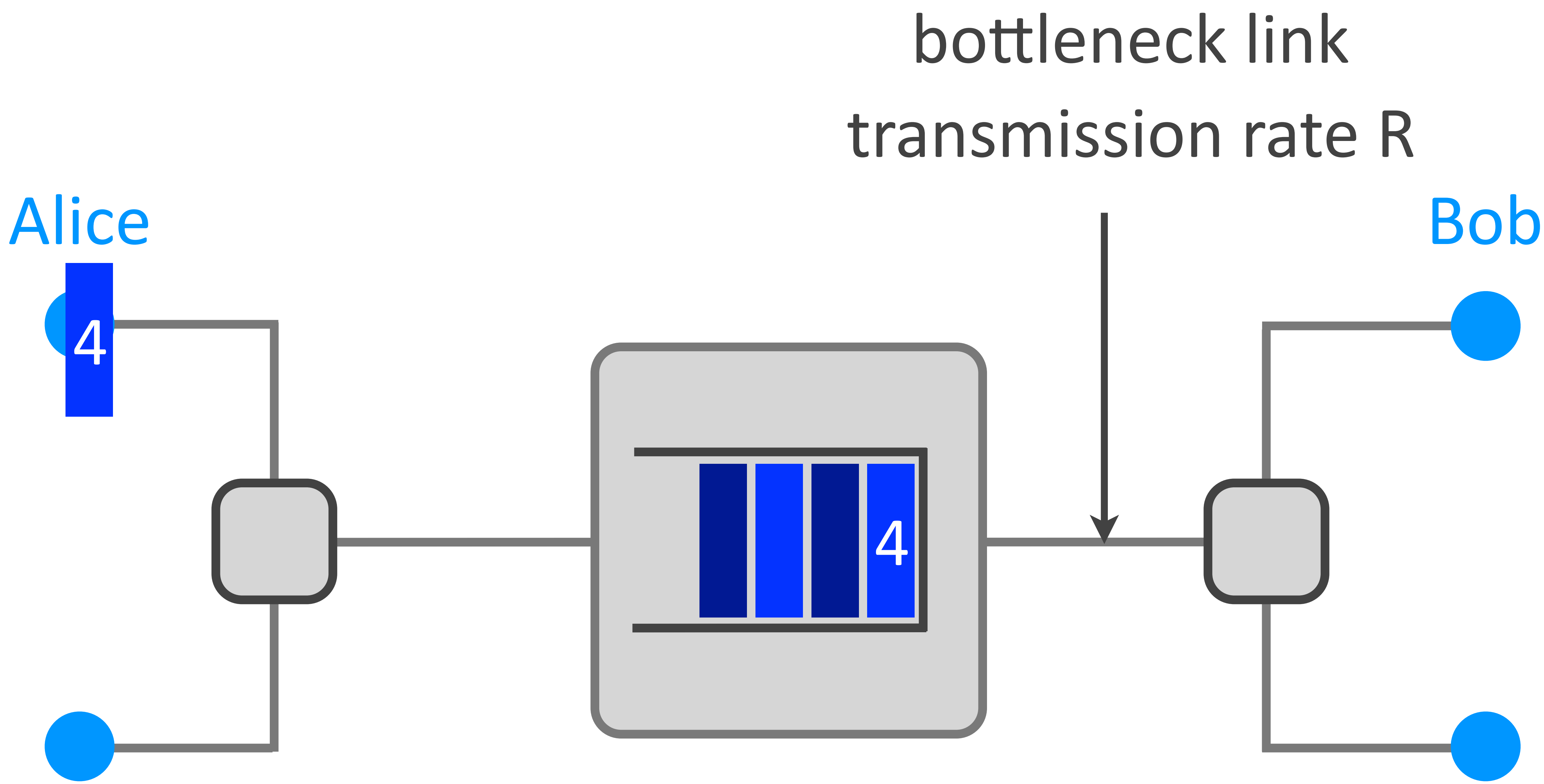
If Alice's transport transmits at rate R/2,

part of that rate is spent on retransmissions,

so, her effective throughput is < R/2

Bad congestion effects

- Long queuing delays
- Resource waste
  - *sender has to retransmit*



If Alice times out prematurely,

and needlessly (re)transmits packets,

the switch performs useless transmissions

Bad congestion effects

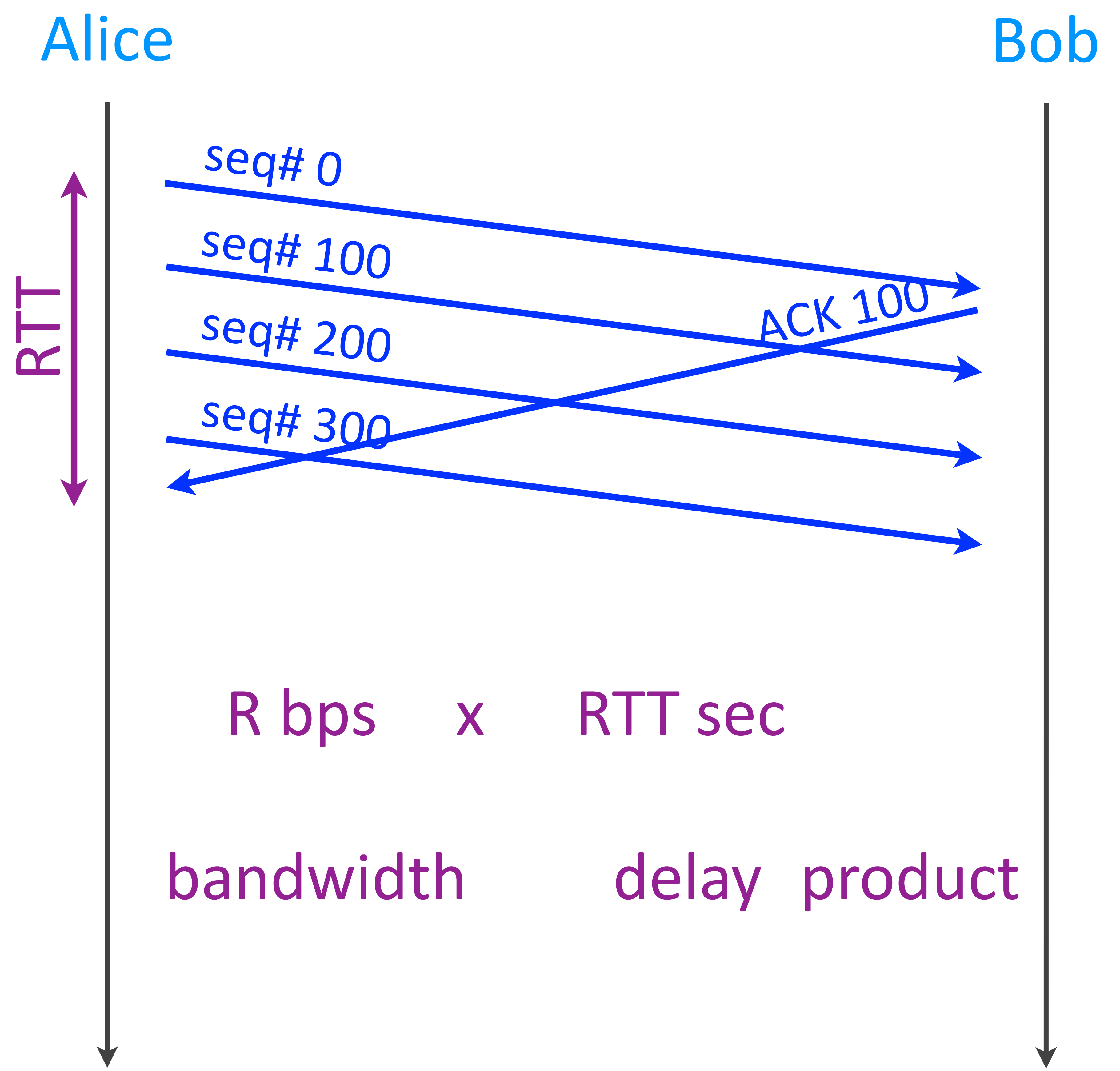
- Long queuing delays
- Resource waste
  - *sender has to retransmit*
  - *switches transmit duplicate packets*
  - *switches transmit packets that will be dropped*

Congestion-control approaches

- At the network layer
  - *packet switches signal congestion to end-hosts*
- At the transport layer
  - *end-hosts signal congestion to each other*

Congestion window

- The number of unacknowledged bytes that the sender may transmit...
- ... so as to avoid "creating congestion"

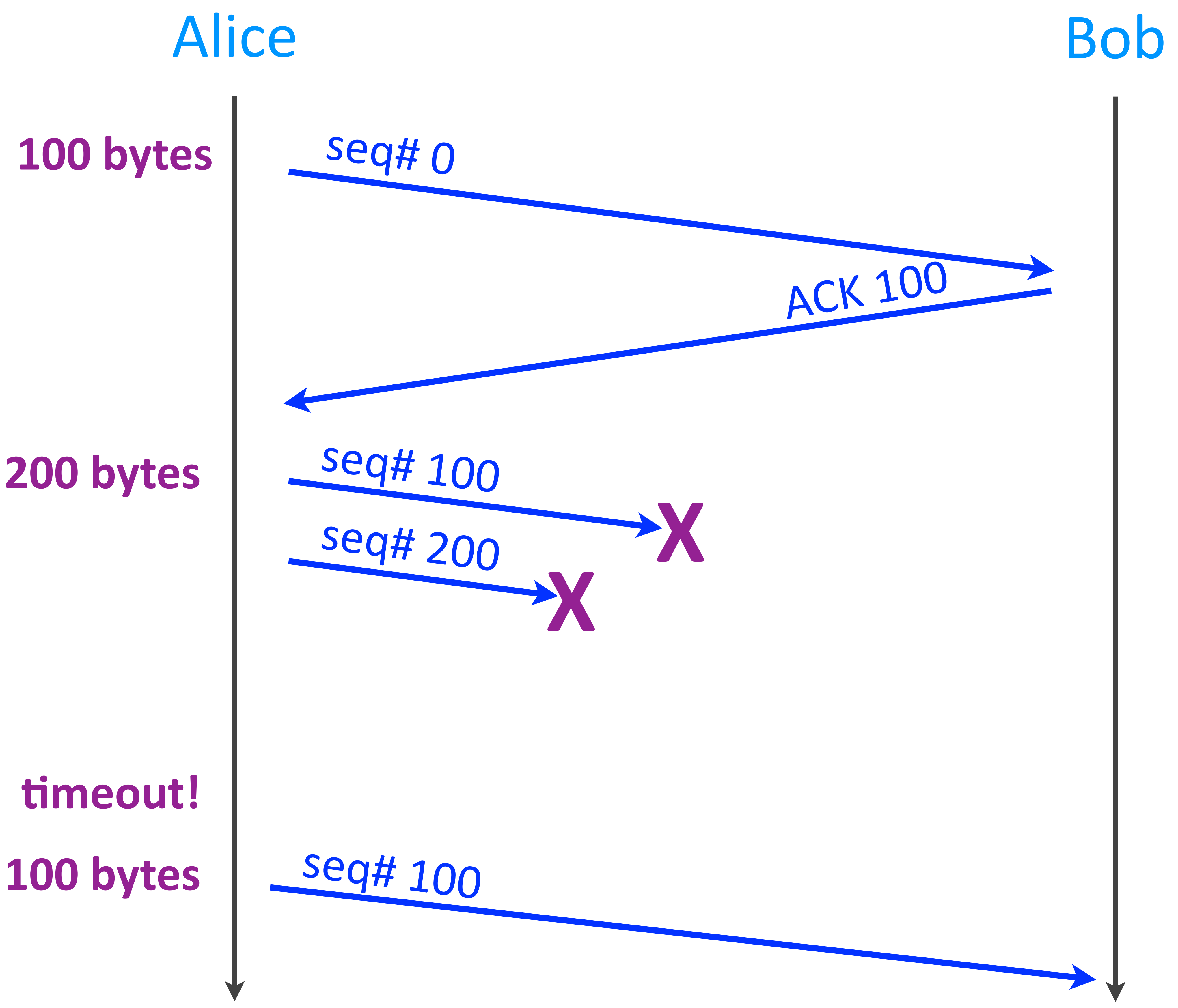


R bps x RTT sec

bandwidth delay product

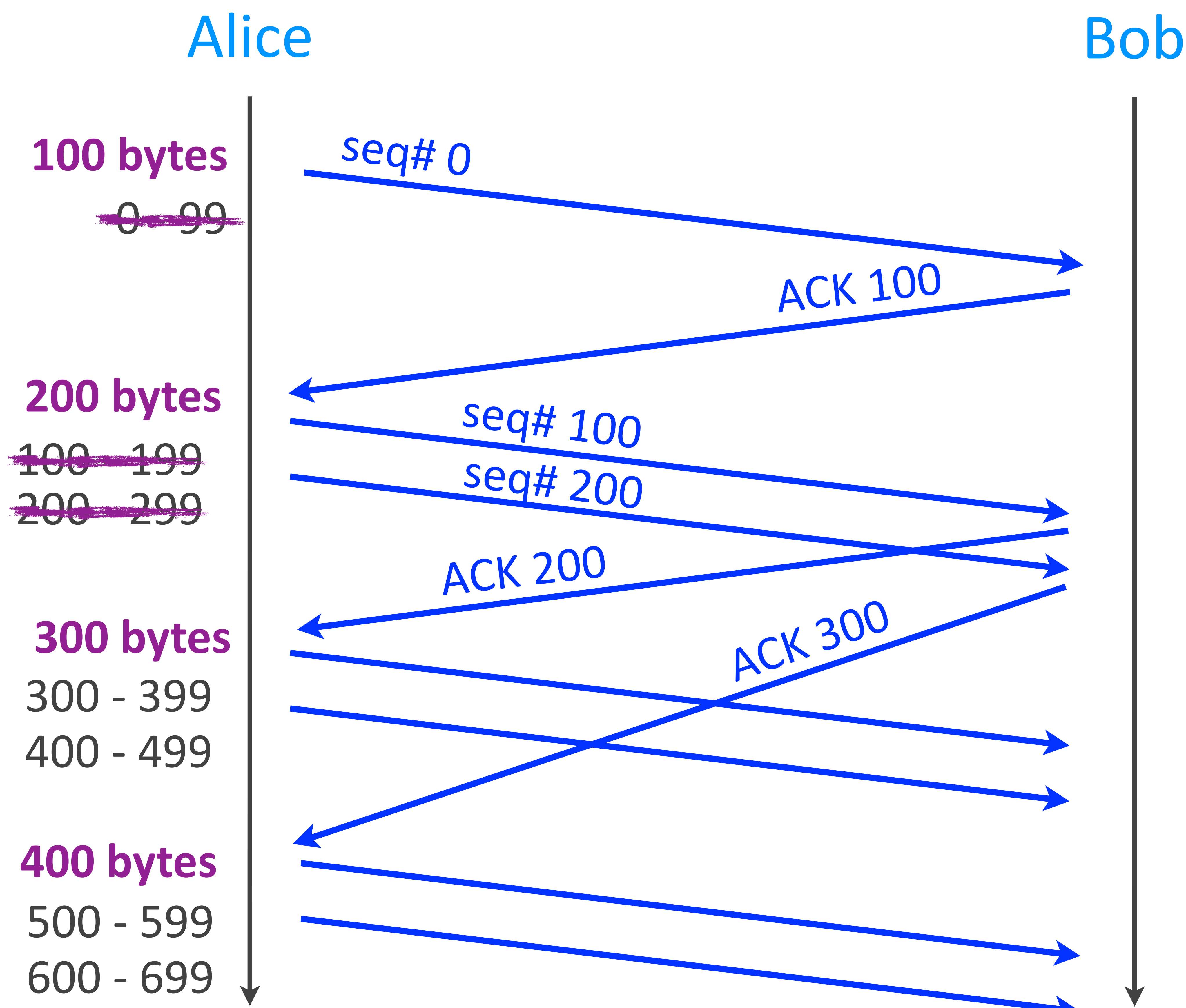
Bandwidth-delay product

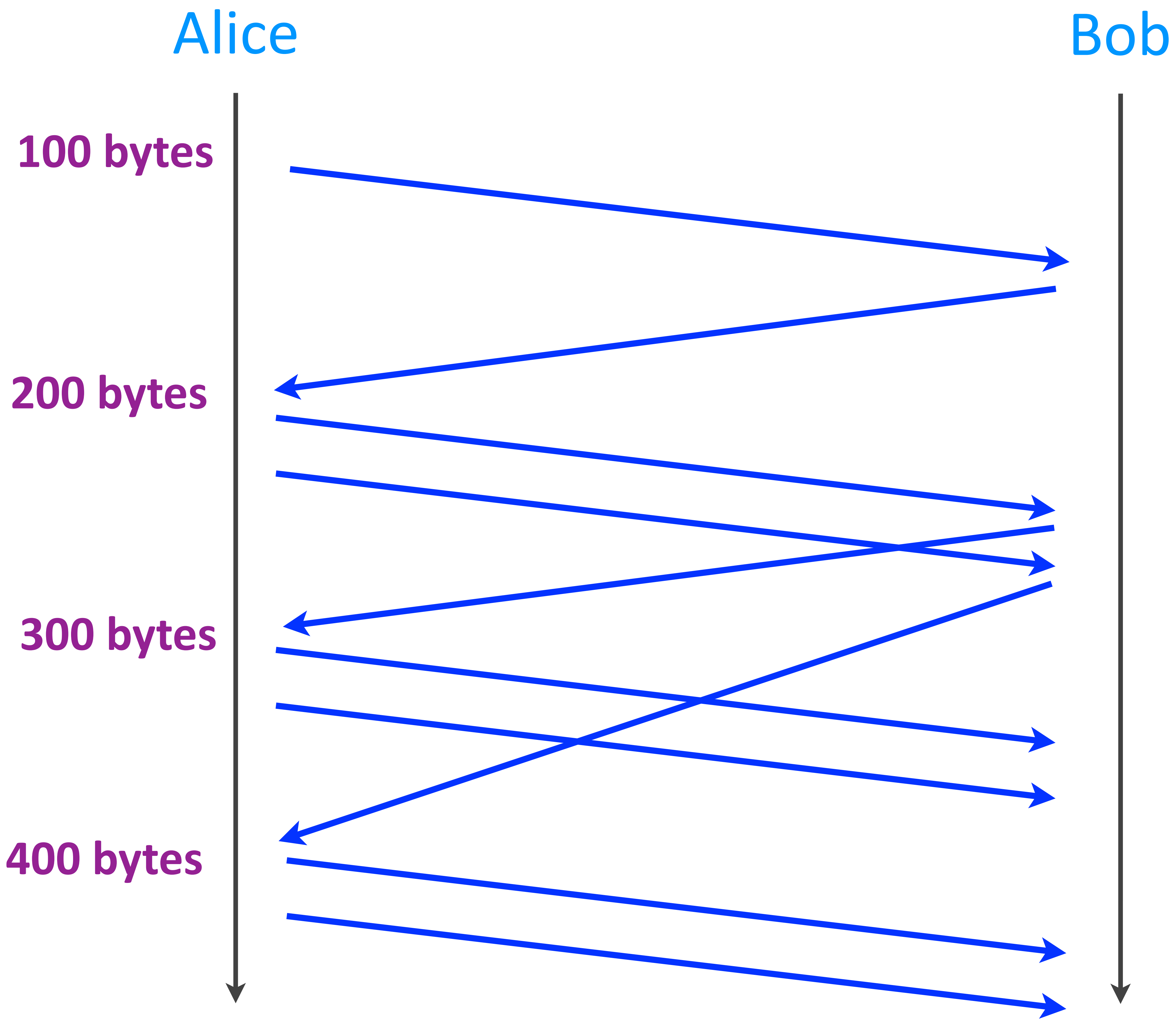
- The max amount of traffic that the sender can transmit until he gets the first ACK
- = the maximum sender window size

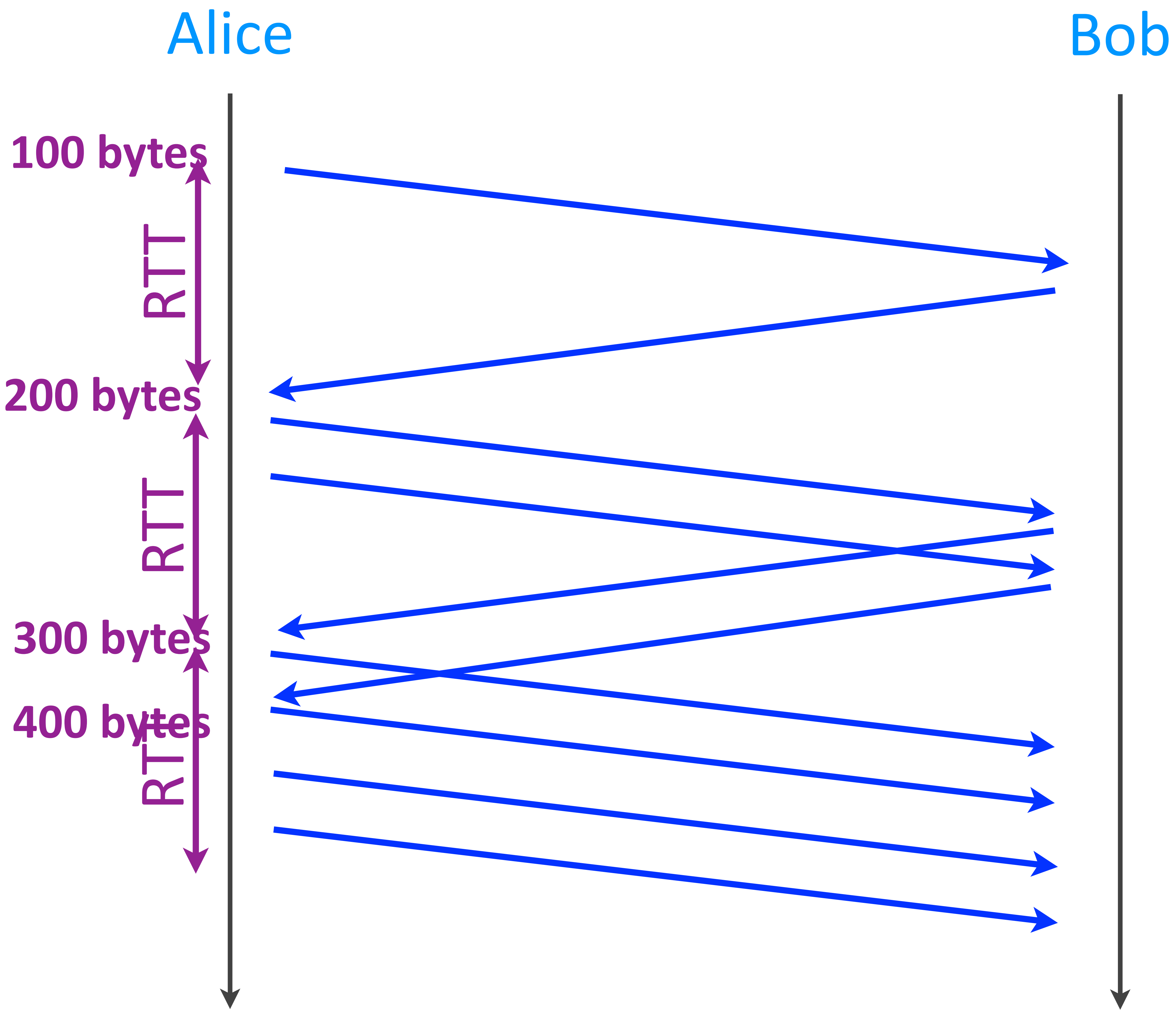

Alice
Bob
100 bytes
seq# 0
ACK 100
200 bytes
seq# 100
X
seq# 200
X
timeout!
100 bytes
seq# 100

Self-clocking

- Inferring the "right" congestion window based on the ACKs
- ACK = no congestion, increase window
- No ACK = congestion, decrease window

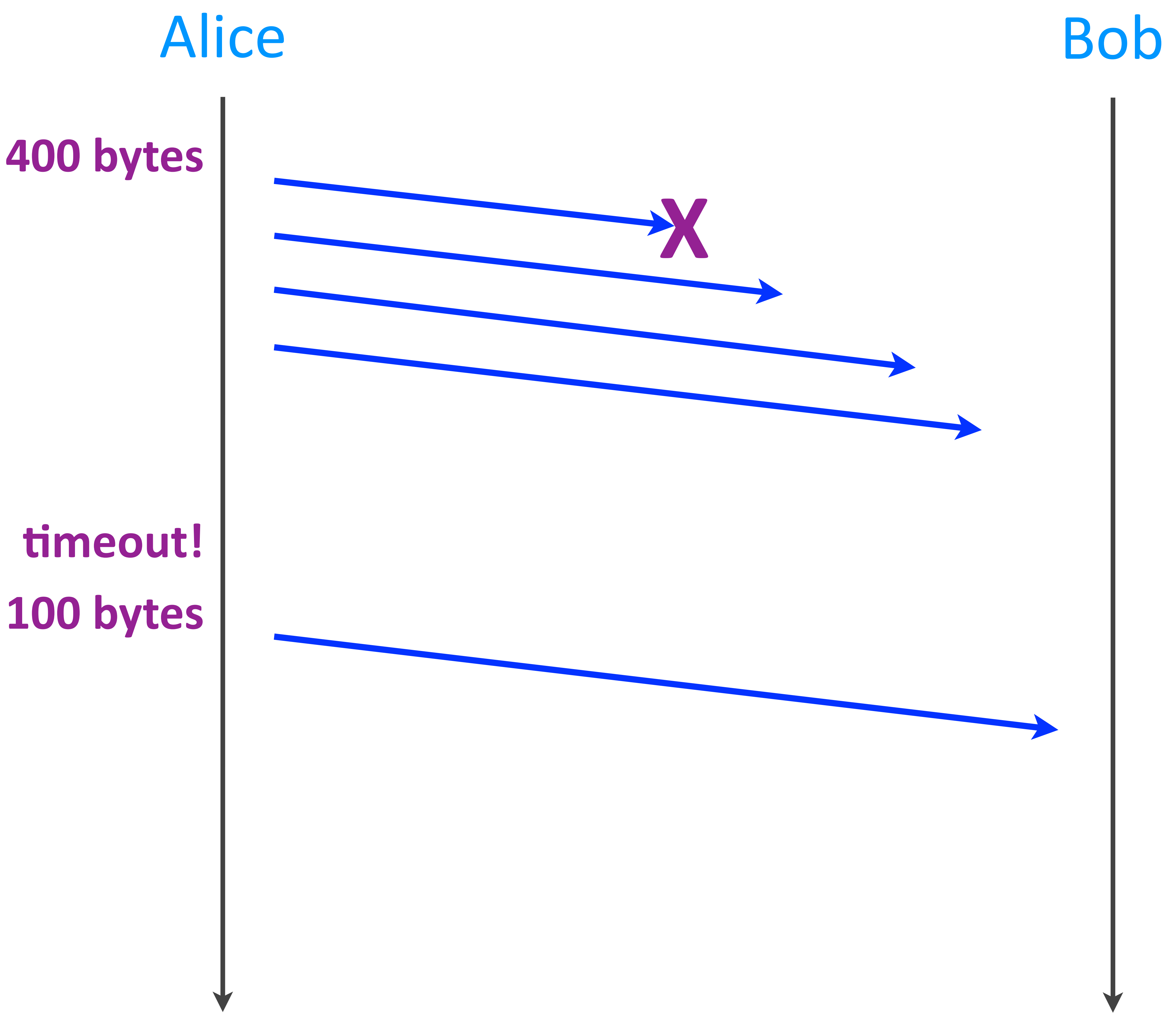

Alice
Bob
100 bytes
0 - 99
seq# 0
ACK 100
200 bytes
100 - 199
200 - 299
seq# 100
seq# 200
ACK 200
300 bytes
300 - 399
400 - 499
ACK 300
400 bytes
500 - 599
600 - 699

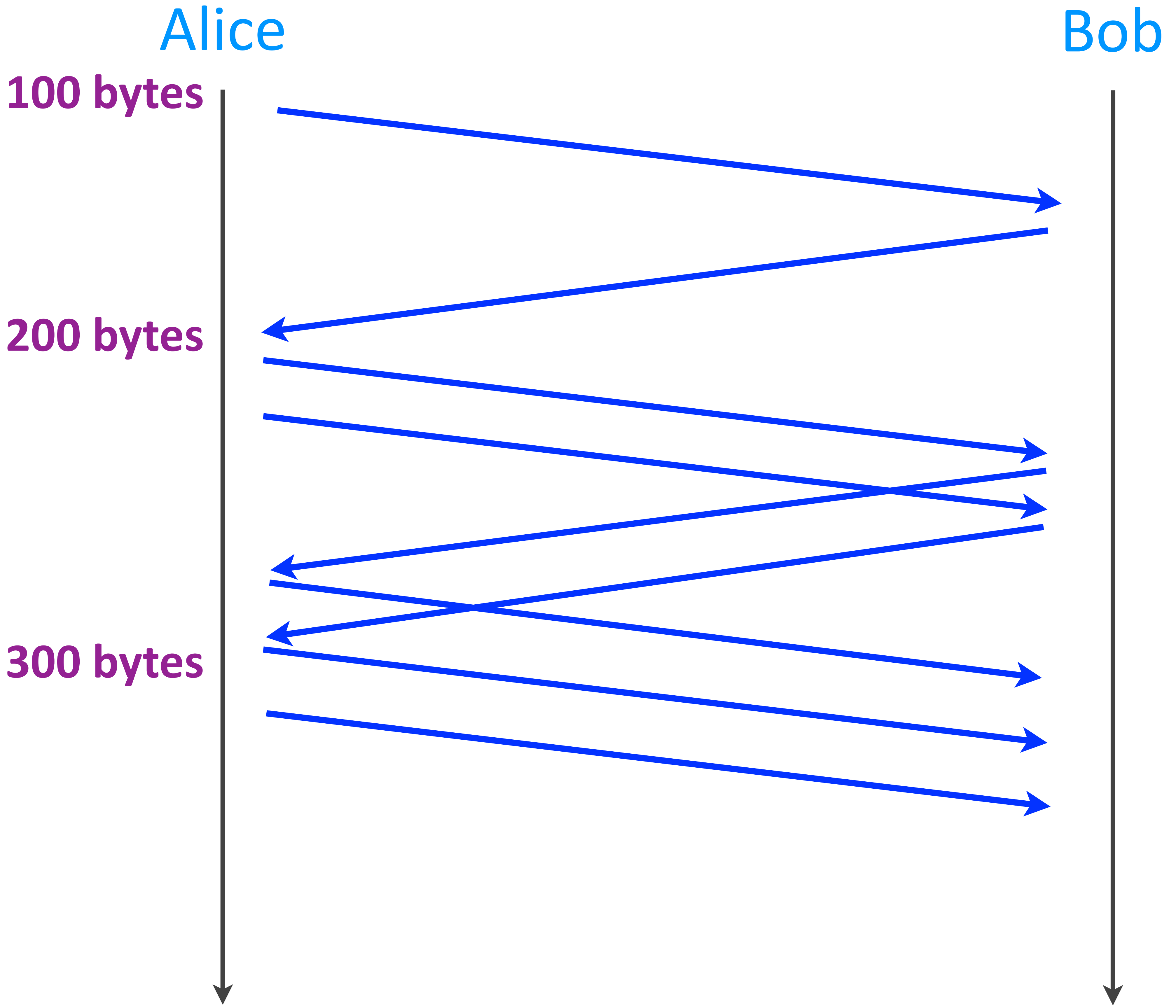

Alice
Bob
100 bytes
200 bytes
300 bytes
400 bytes


Alice
Bob
100 bytes
RTT
200 bytes
RTT
300 bytes
400 bytes
RTT

Increase window size

- Exponentially
  - *by 1 MSS for every ACKed segment*
  - *= window doubles every RTT*
  - *when we do not expect congestion*

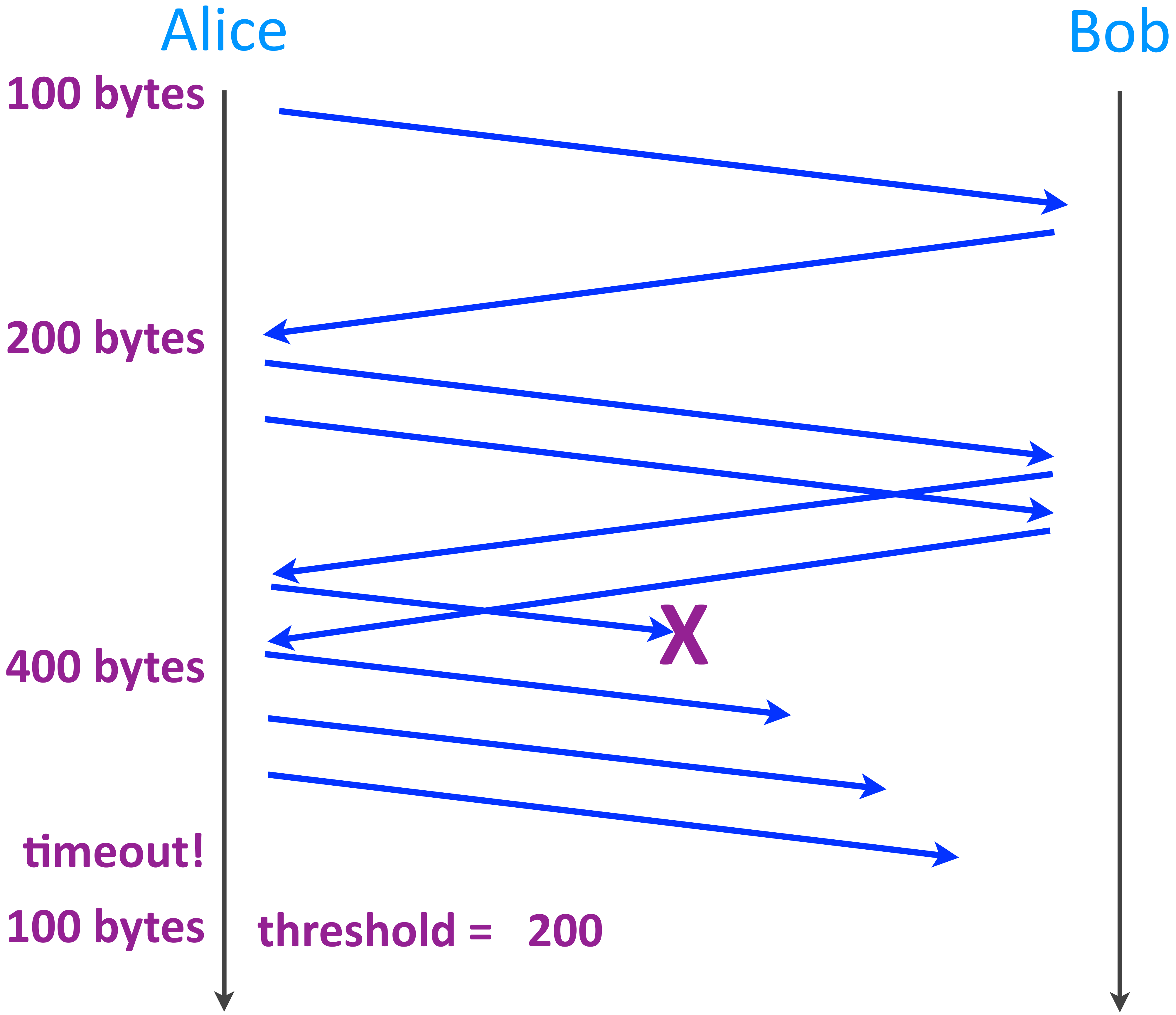

Alice
Bob
400 bytes
X
timeout!
100 bytes

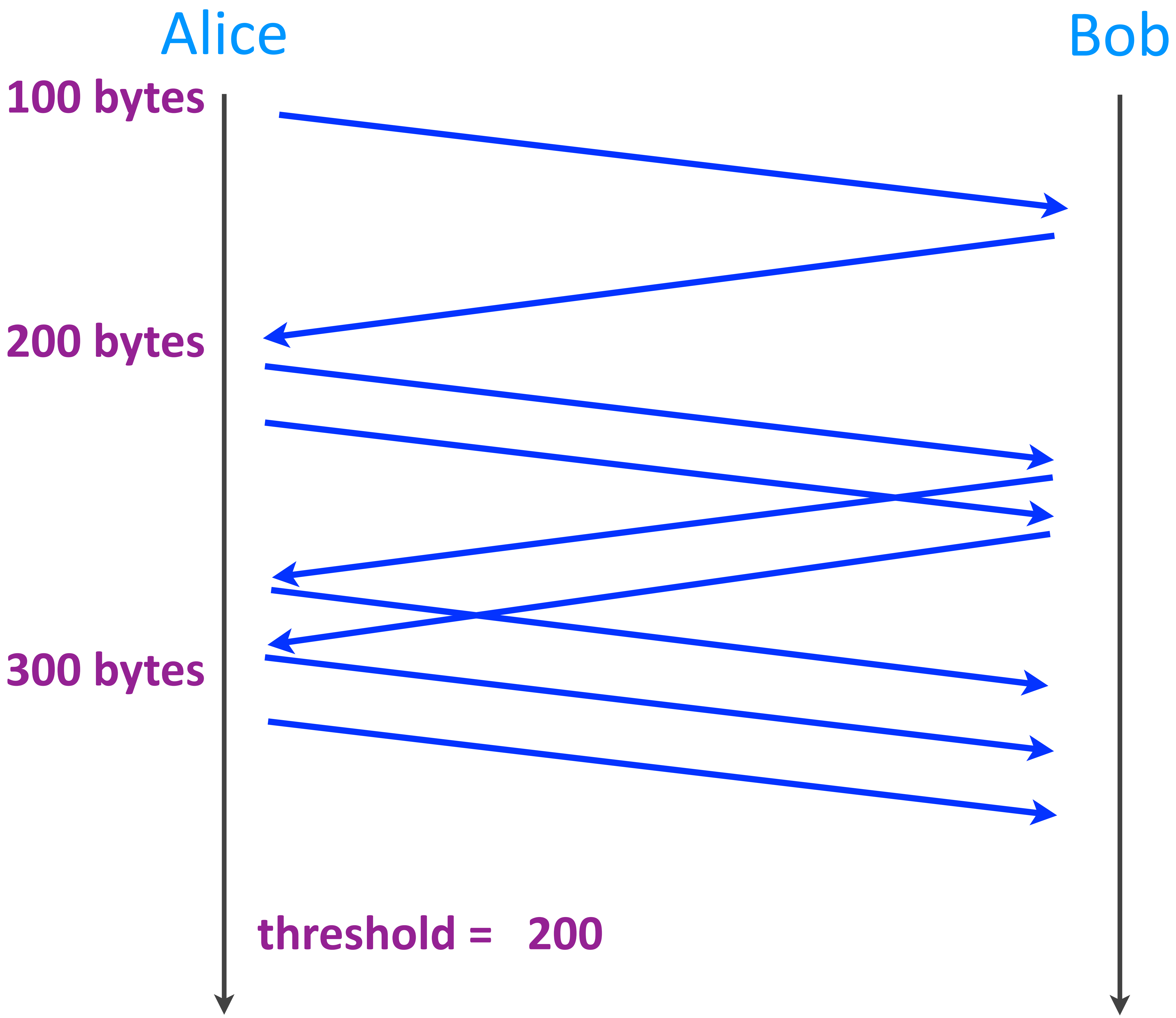

Alice
Bob
100 bytes
200 bytes
300 bytes

Increase window size

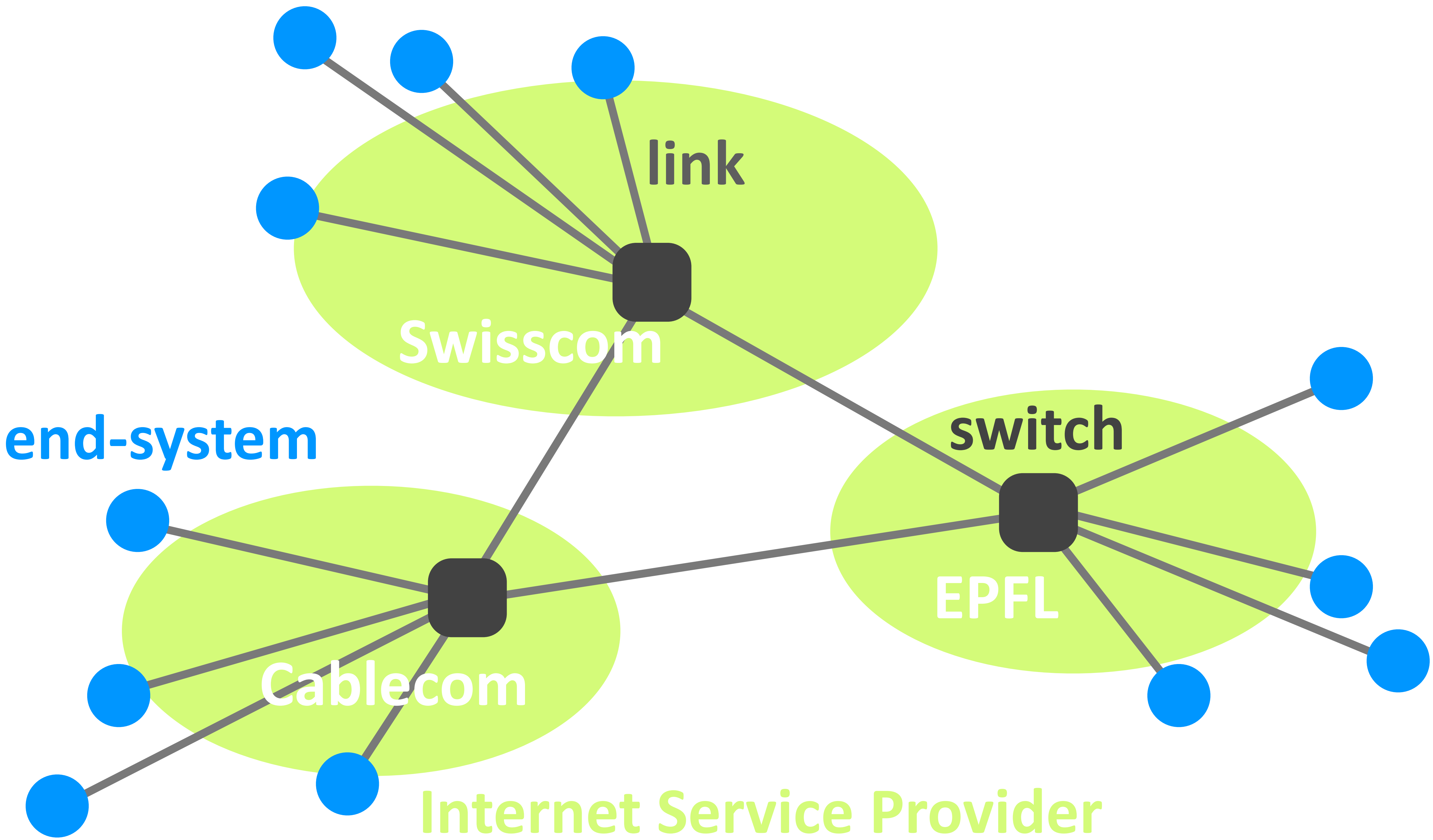
- Exponentially
  - *by 1 MSS for every ACKed segment*
  - *= window doubles every RTT*
  - *when we do not expect congestion*

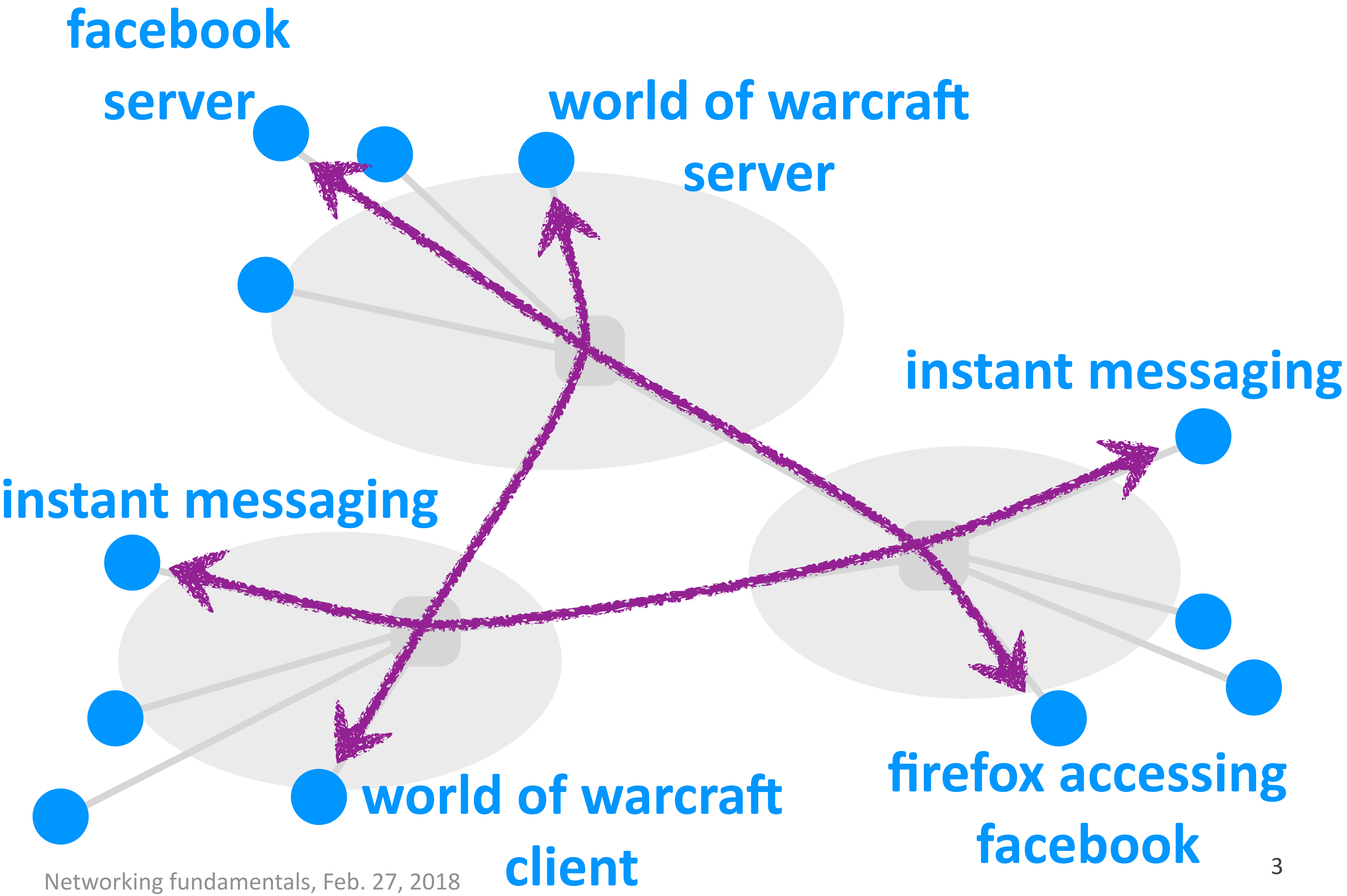
- Linearly
  - *by 1 MSS every RTT*
  - *when we expect congestion*

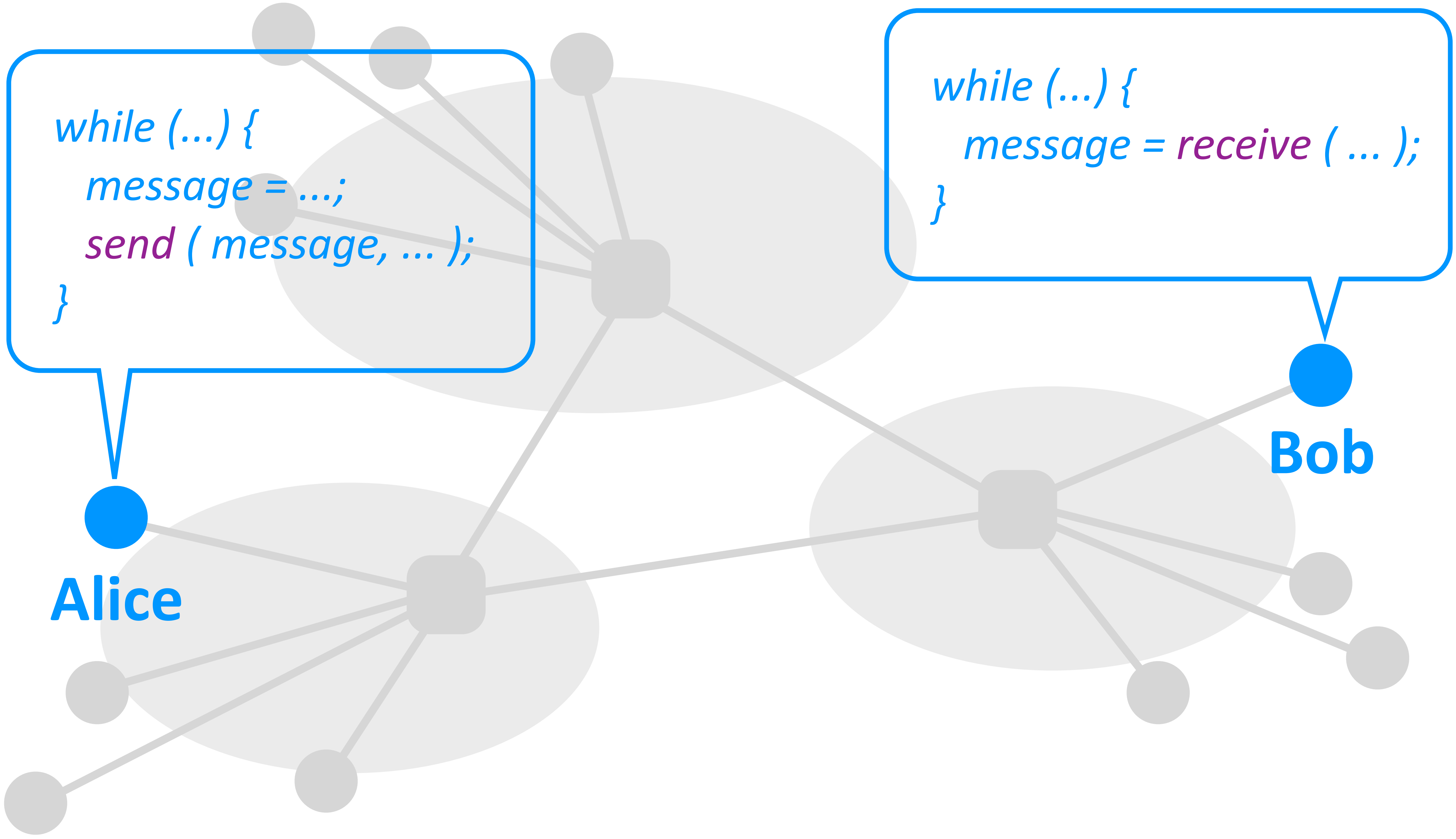

Alice
Bob
100 bytes
200 bytes
X
400 bytes
timeout!
100 bytes
threshold = 200

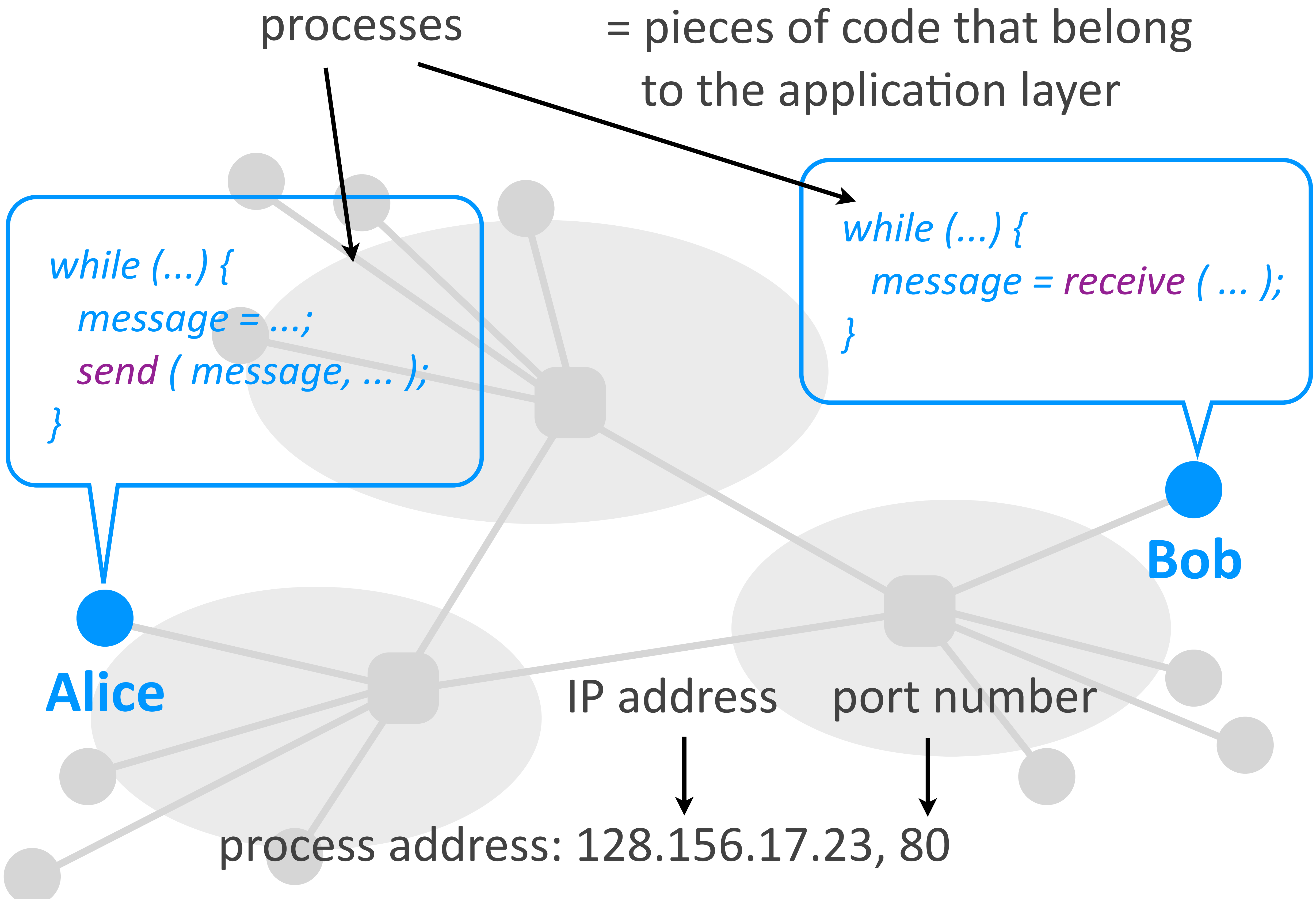

Alice
Bob
100 bytes
200 bytes
300 bytes
threshold = 200

Application layer


link
Swisscom
switch
end-system
EPFL
Cablecom
Internet Service Provider


facebook
server
world of warcraft
server
instant messaging
instant messaging
world of warcraft
client
firefox accessing
facebook


while (...) {
message = ...;
send (message, ...);
}
while (...) {
message = receive (...);
}
Alice
Bob


processes
= pieces of code that belong
to the application layer
while (...) {
message = ...;
send (message, ...);
}
while (...) {
message = receive (...);
}
Alice
Bob
IP address
port number
process address: 128.156.17.23, 80

Designing distributed apps

- How is the functionality of the application distributed over the processes?

Outline

- Client-server vs. peer-to-peer
- Example 1: web
- Examlpe 2: DNS
- Example 3: P2P file sharing

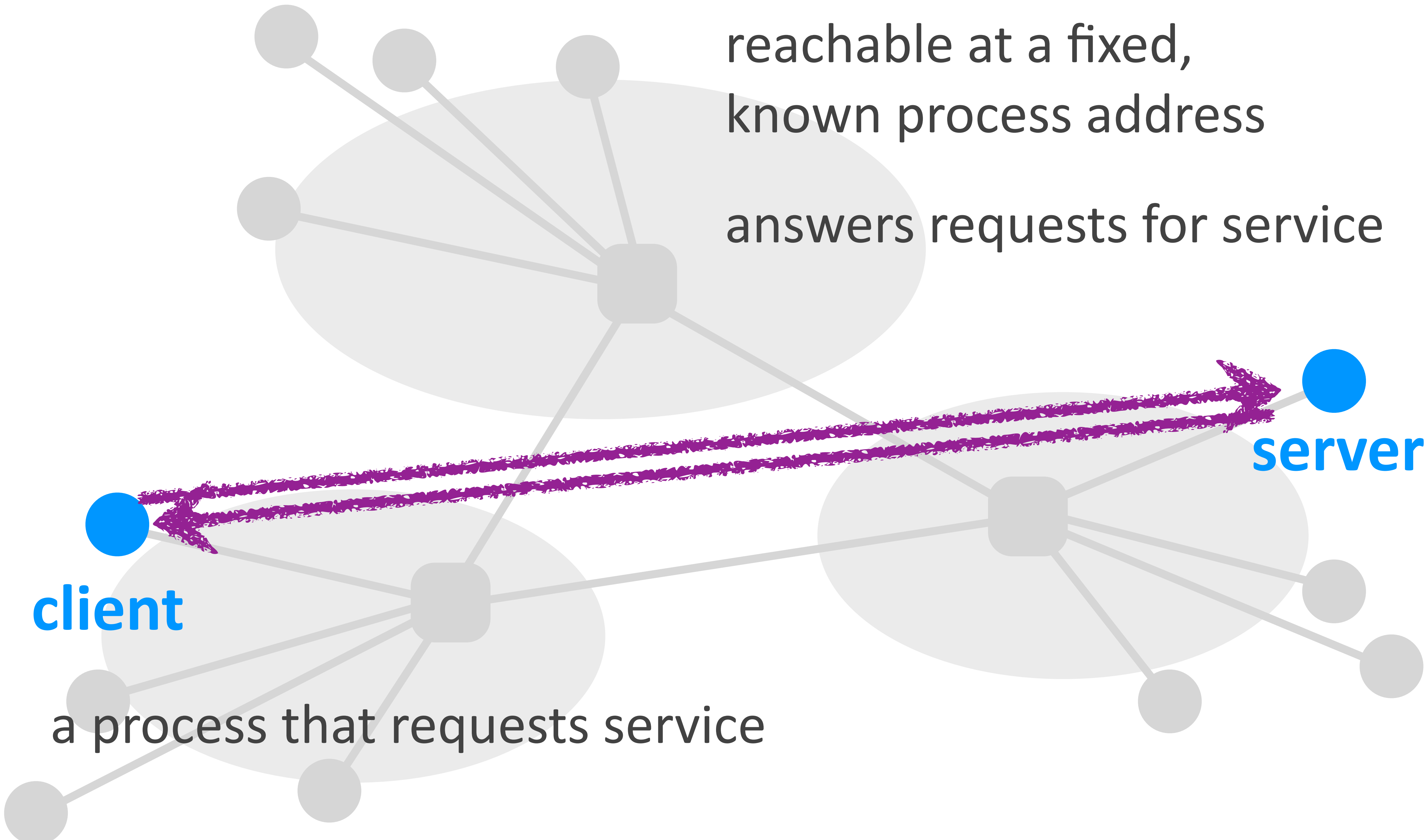
Outline

- Client-server vs. peer-to-peer
- Example 1: web
- Examlpe 2: DNS
- Example 3: P2P file sharing

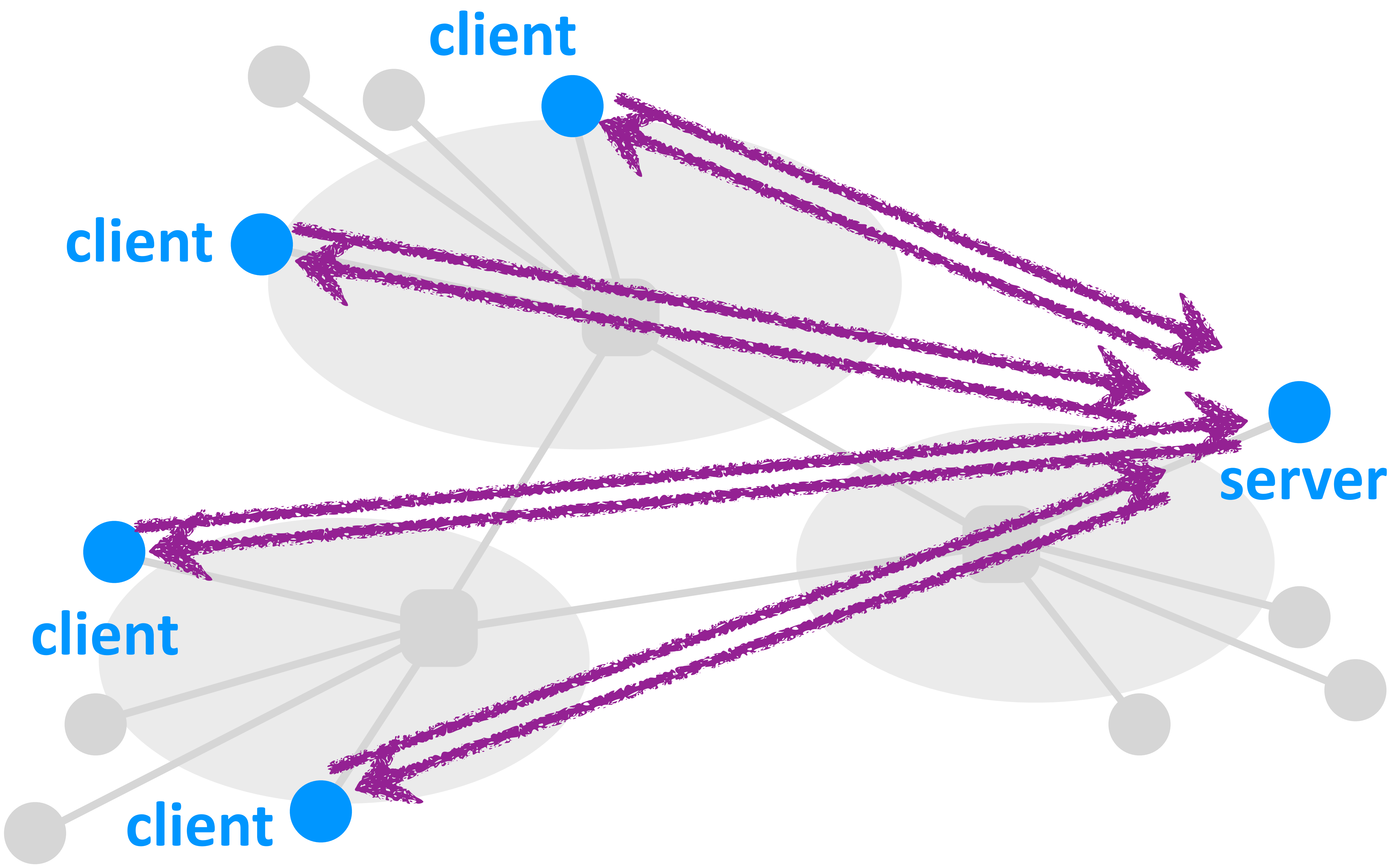
a process that is always running

reachable at a fixed,
known process address

answers requests for service

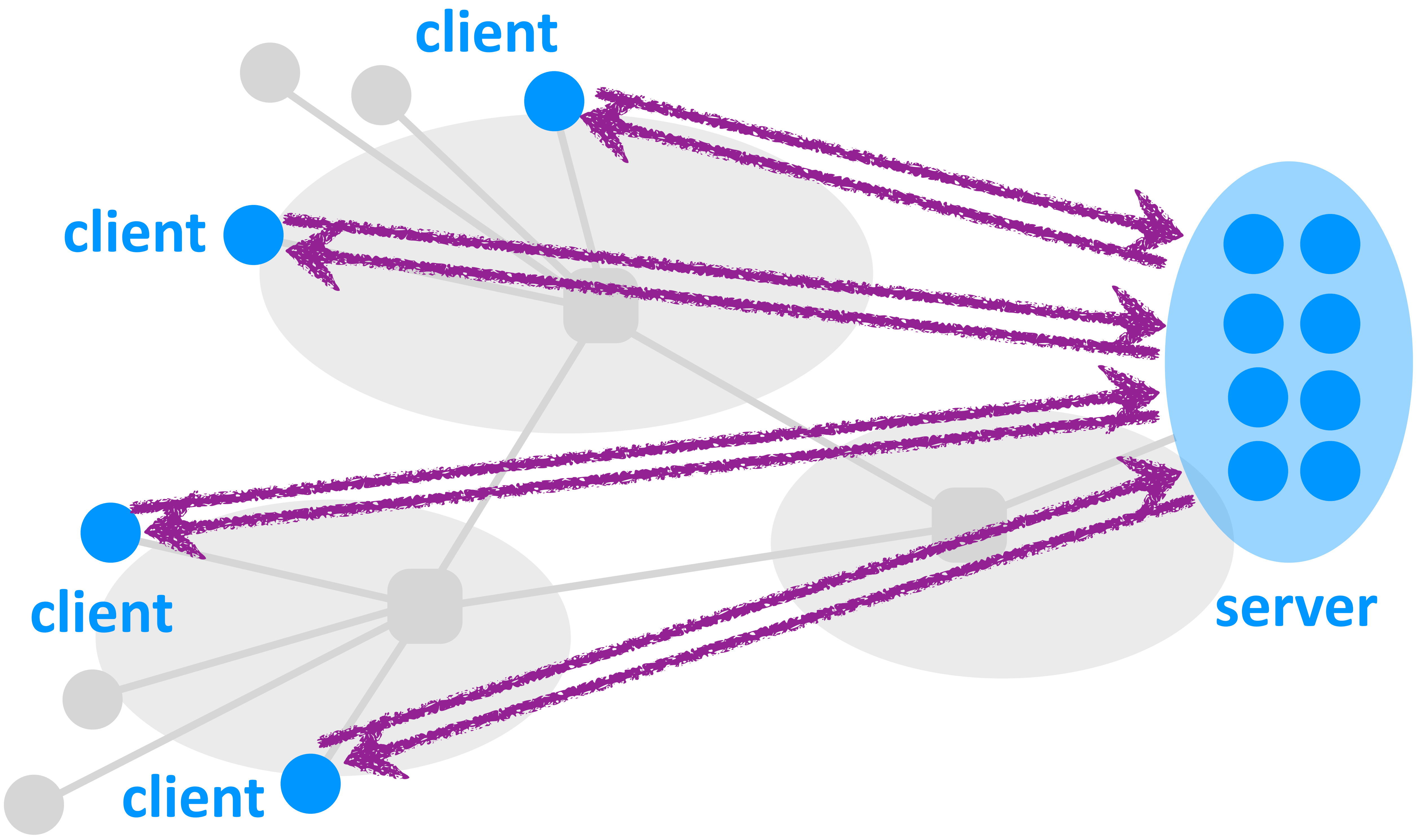


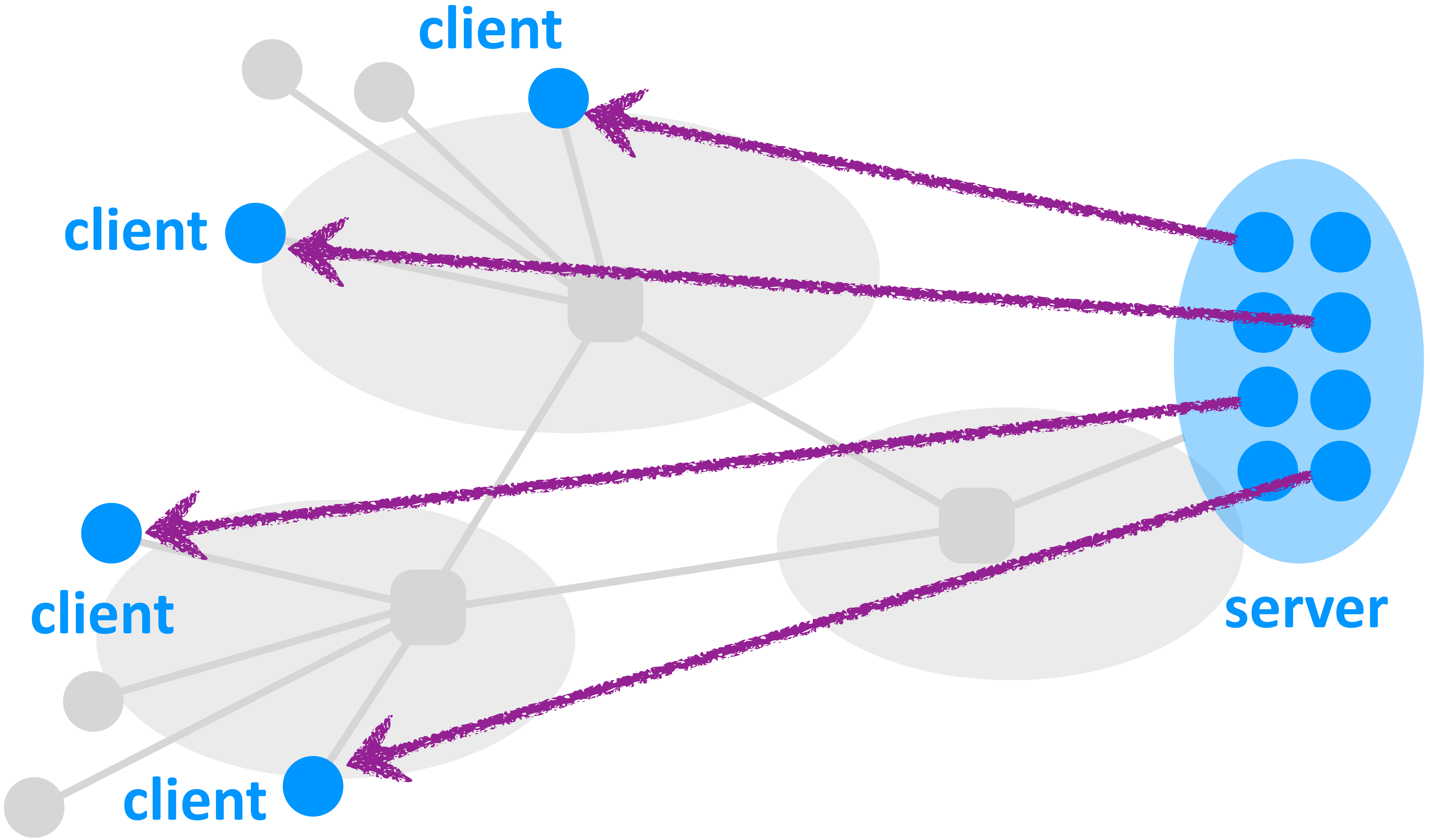
a process that requests service


client
client
server
client
client

Client-server architecture

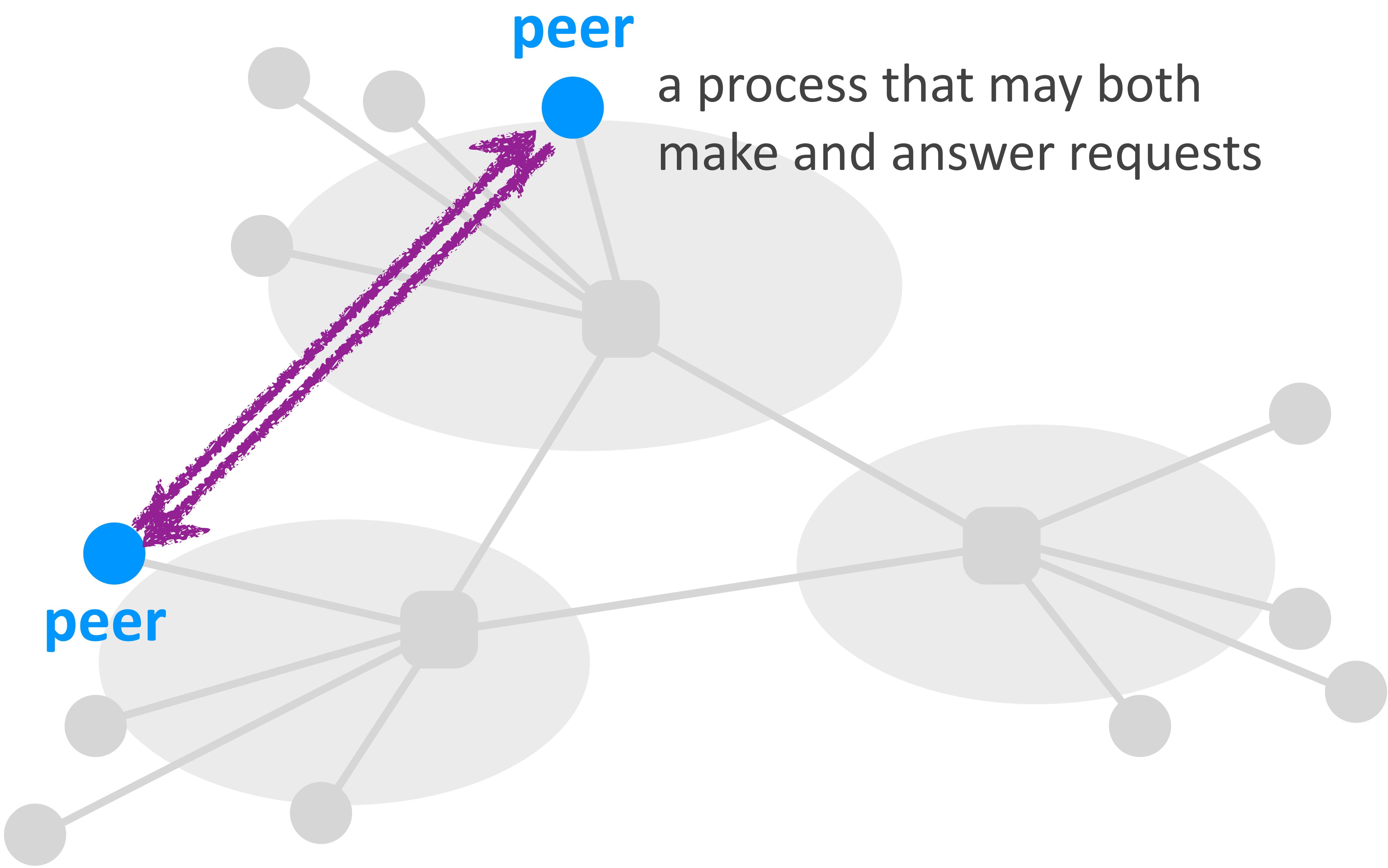
- Clear separation of roles
  - *a client process makes requests for service*
  - *a server process answers (or denies) the requests*


client
client
client
client
server

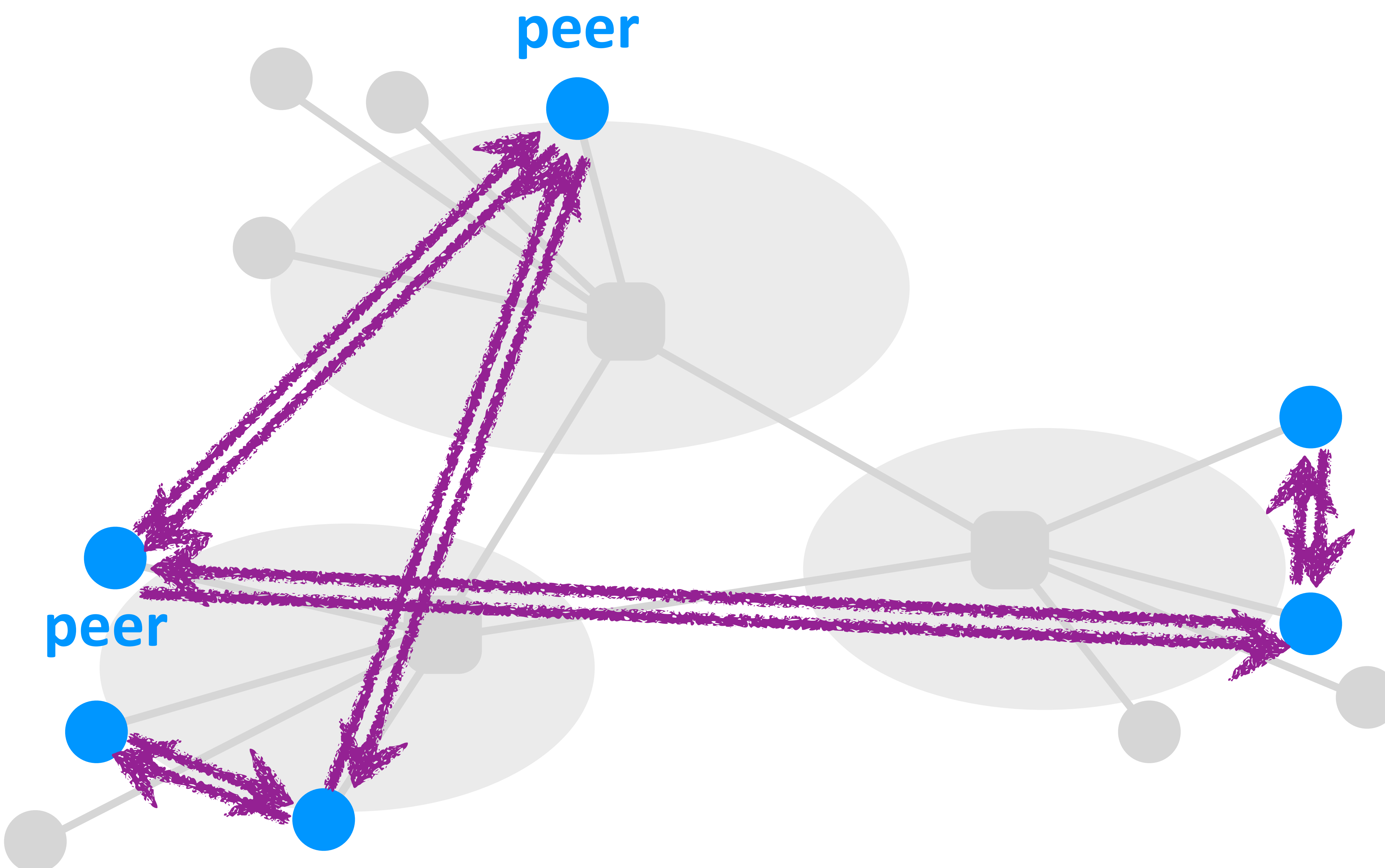

client
client
client
client
server

Client-server architecture

- Clear separation of roles
  - *a client makes requests for service*
  - *a server answers (or denies) the requests*
- Server runs on dedicated infrastructure
  - *could be one machine*
  - *more likely a data-center*



a process that may both
make and answer requests


peer
peer

Peer-to-peer architecture

- A peer may act both as client and server
  - *a peer may request service from another peer*
  - *or provide service to another peer*

- Peer runs on personally owned end-system
  - *PC, laptop, smartphone*
  - *no dedicated infrastructure*

Two architecture choices

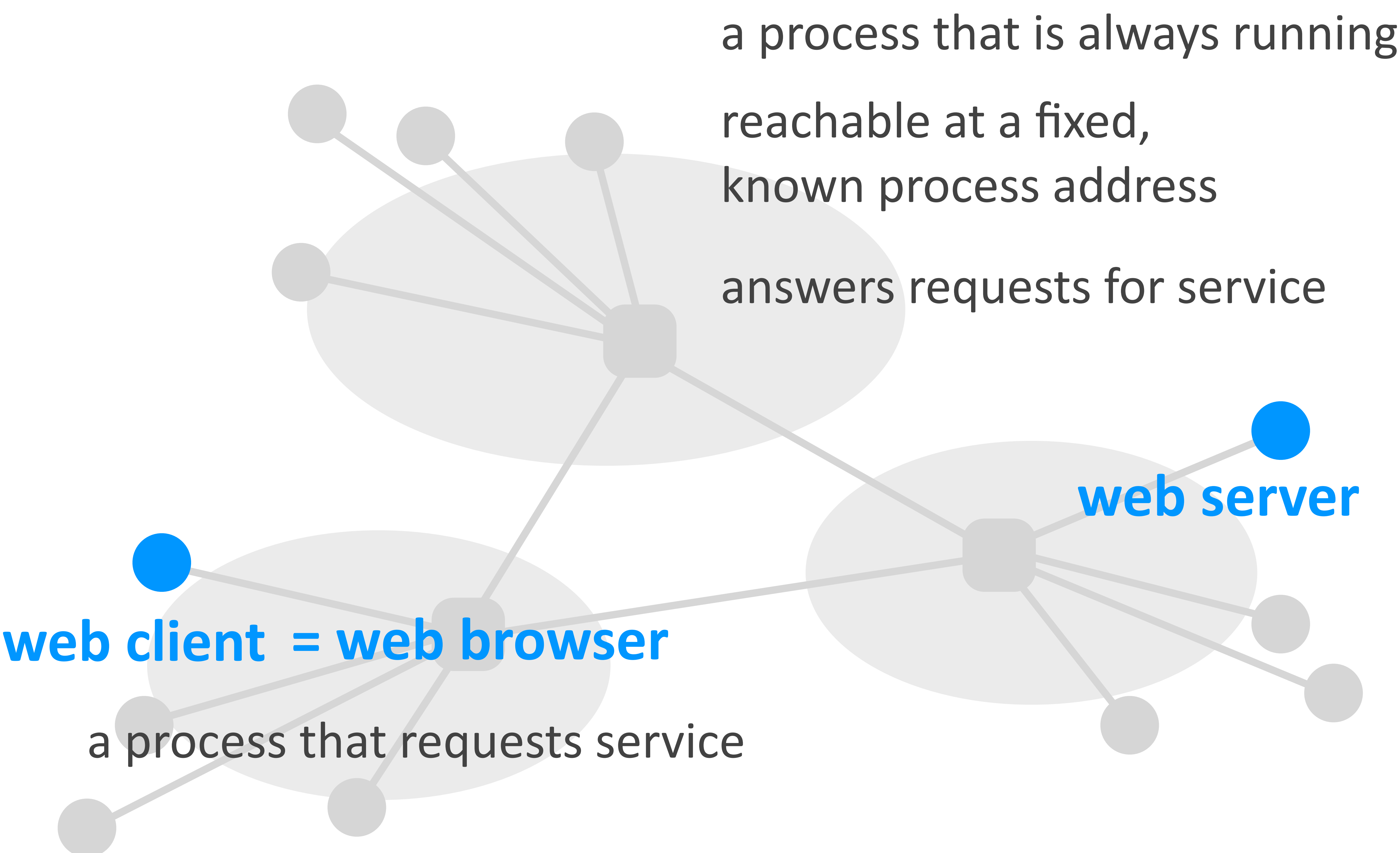
- Client-server architecture
  - *clear separation of roles*
  - *server runs on dedicated infrastructure*

- Peer-to-peer architecture
  - *peers act both as servers and clients*
  - *peer runs on personally owned end-system*

Which one to choose?

Outline

- Client-server vs. peer-to-peer
- Example 1: web
- Examlpe 2: DNS
- Example 3: P2P file sharing


a process that is always running
reachable at a fixed,
known process address
answers requests for service
web server
web client = web browser
a process that requests service

URLs

- URL = address for web objects
  - *example: www.epfl.ch/index.fr.html*

- URL format: hostname + file name
  - *www.epfl.ch is an end-system (a host)*
  - *index.fr.html is a file*

Processes

- Process = app-layer piece of code
  - *example of process address: 128.178.50.12, 80*

- Address format: IP address + port number
  - *128.178.50.12 is an end-system (a host)*
  - *80 is the port number for web server processes*

Web request

- You enter a URL into your web client
  - *www.epfl.ch/index.fr.html*
- Web client extracts hostname
  - *www.epfl.ch*
- Translates hostname to IP address
  - *128.178.50.12*
- Forms web-server process address
  - *128.178.50.12, 80*

Web request

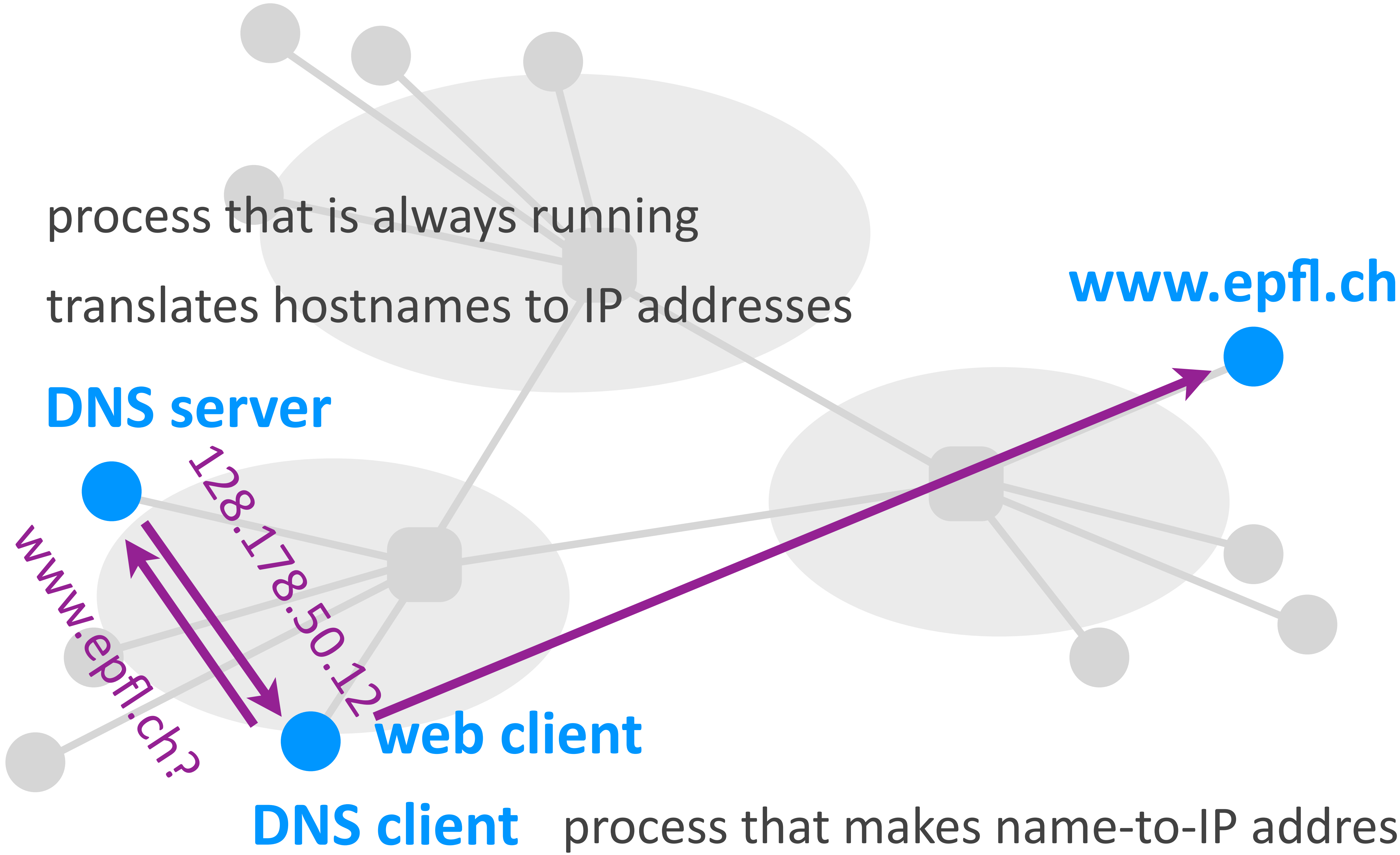
- You enter a URL into your web client
  - *www.epfl.ch/index.fr.html*
- Web client extracts hostname
  - *www.epfl.ch*
- **Translates hostname to IP address**
  - *128.178.50.12*
- Forms web-server process address
  - *128.178.50.12, 80*

Outline

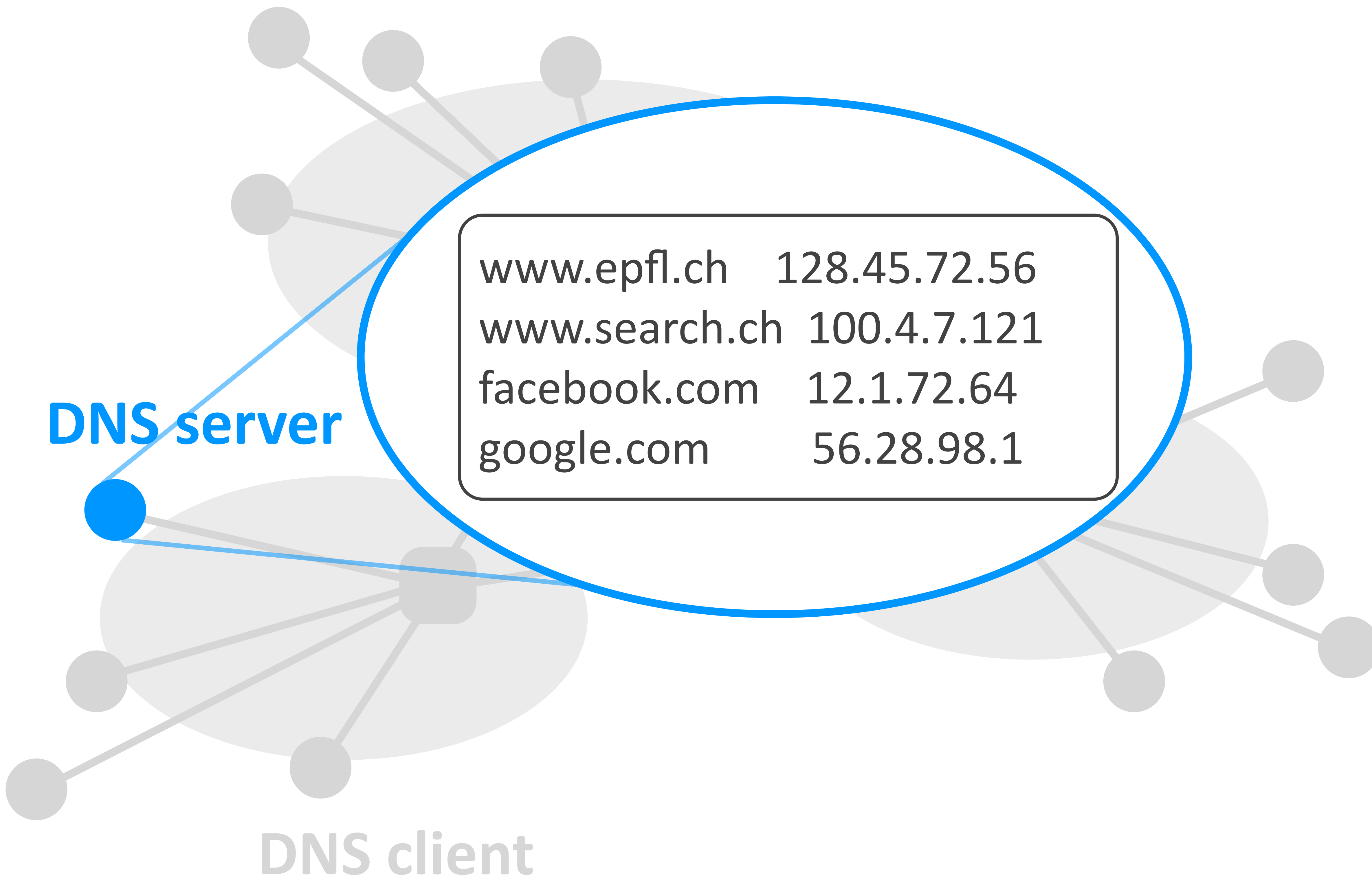
- Client-server vs. peer-to-peer
- Example 1: web
- Examlpe 2: DNS
- Example 3: P2P file sharing

process that is always running

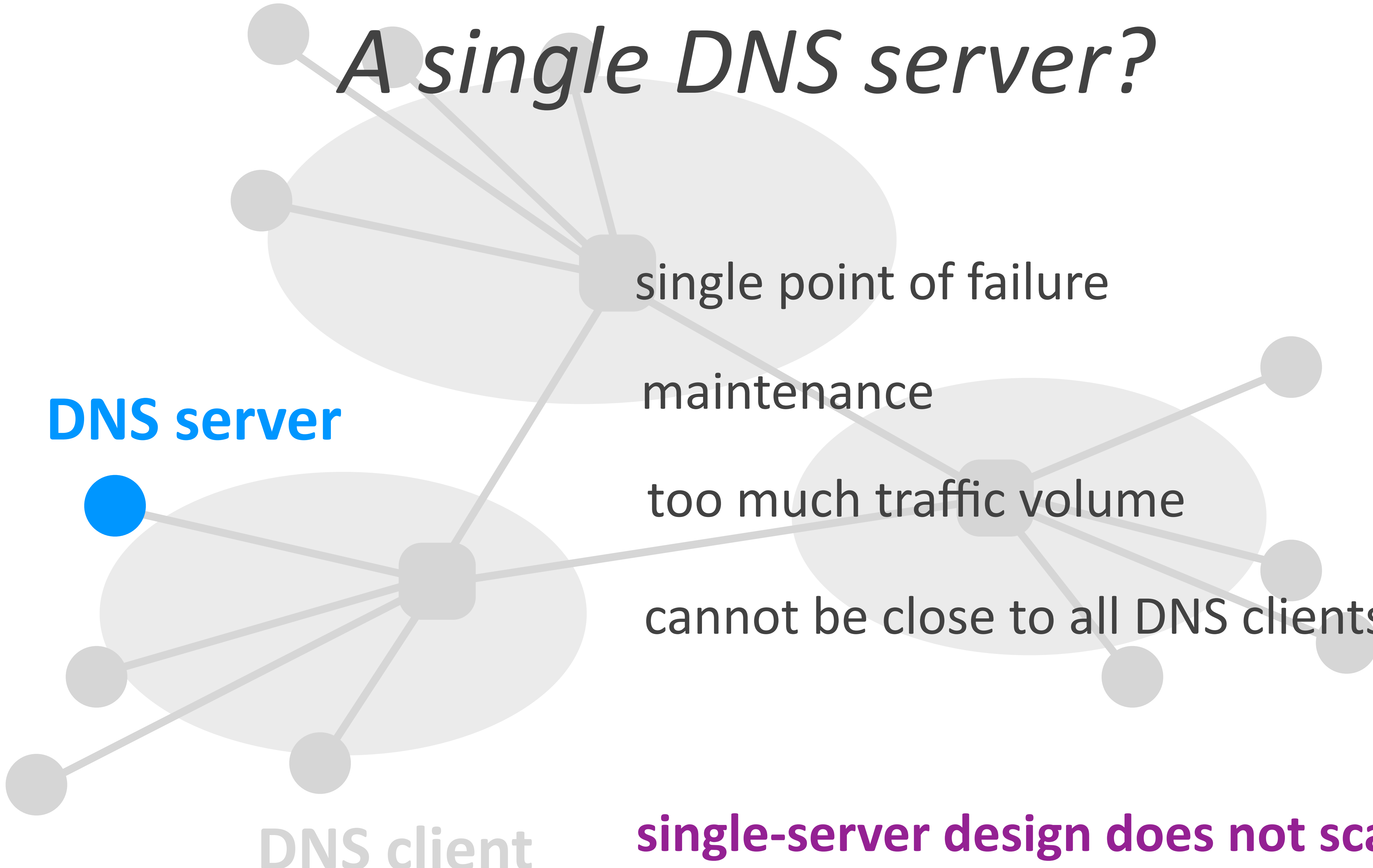
translates hostnames to IP addresses



process that makes name-to-IP address translation requests


www.epfl.ch 128.45.72.56
www.search.ch 100.4.7.121
facebook.com 12.1.72.64
google.com 56.28.98.1
DNS server
DNS client

A single DNS server?



single point of failure

maintenance

too much traffic volume

cannot be close to all DNS clients

single-server design does not scale

Informally:

System does not scale =

does not work well with many users

you cannot simply add resources to fix it

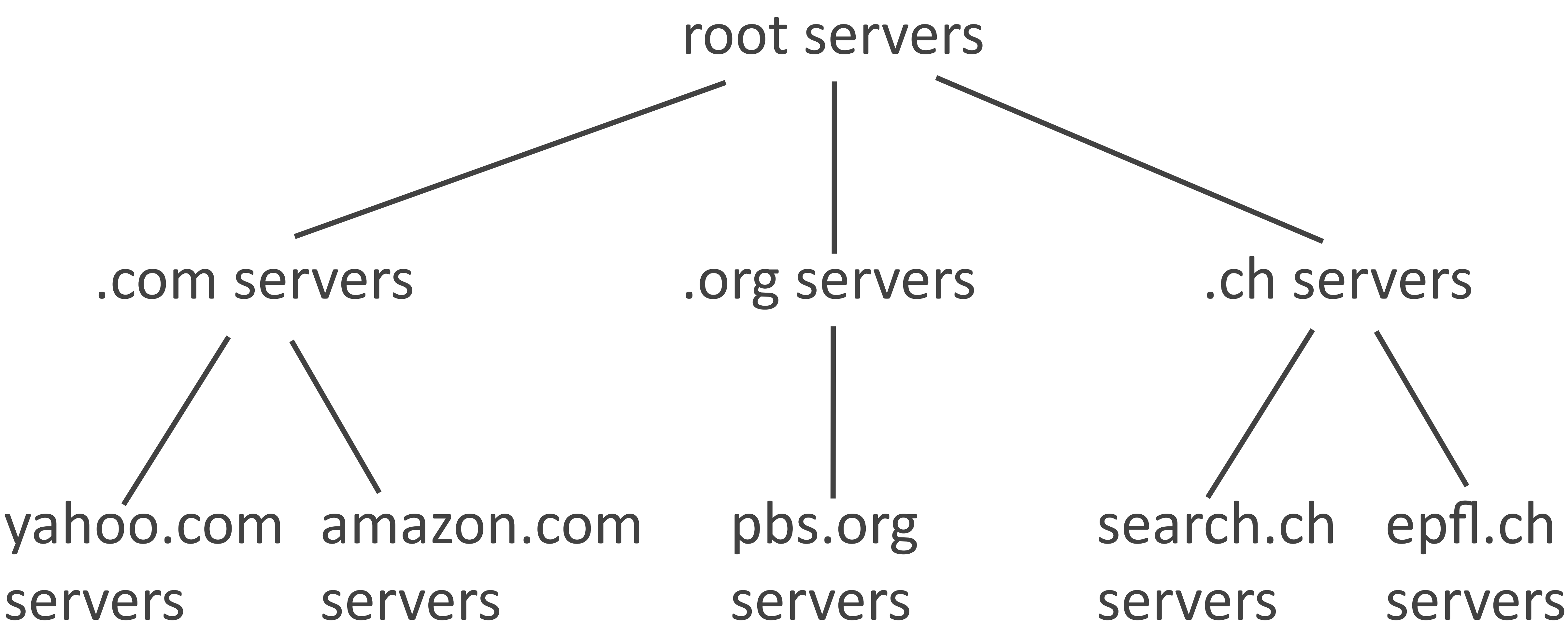
Hierarchy of DNS servers

root servers

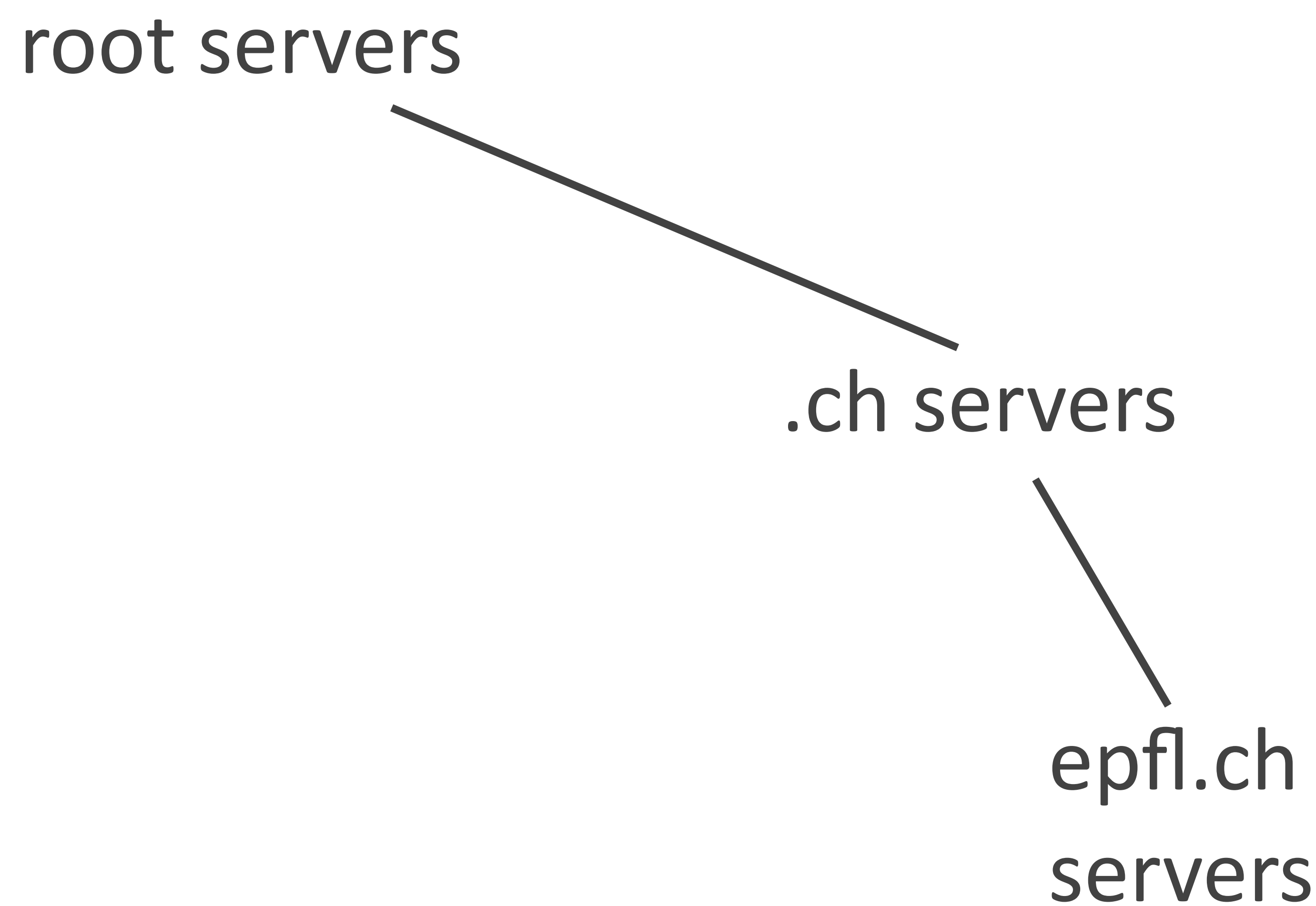
TLD (top-level domain) servers

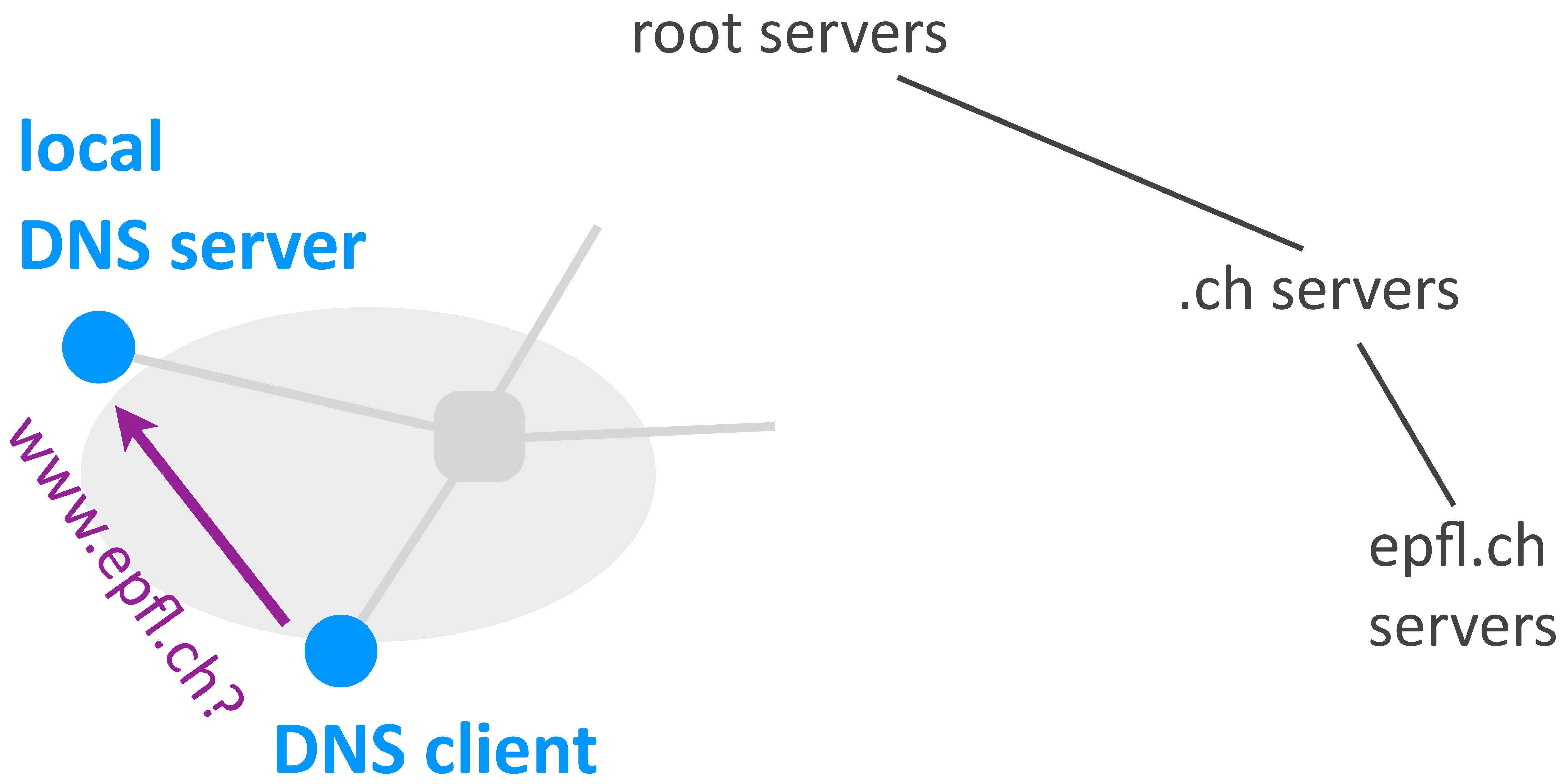
authoritative servers

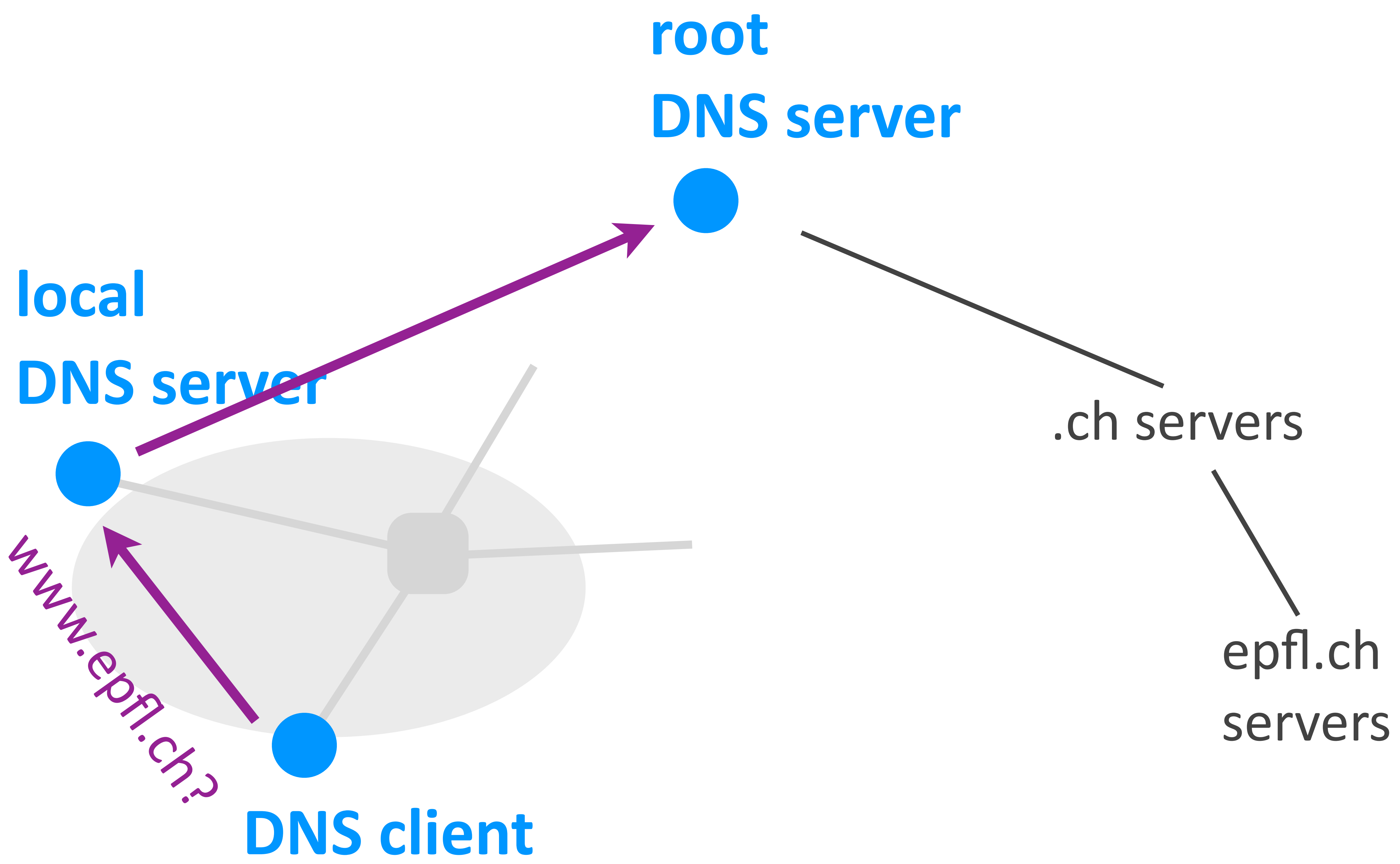
Hierarchy of DNS servers

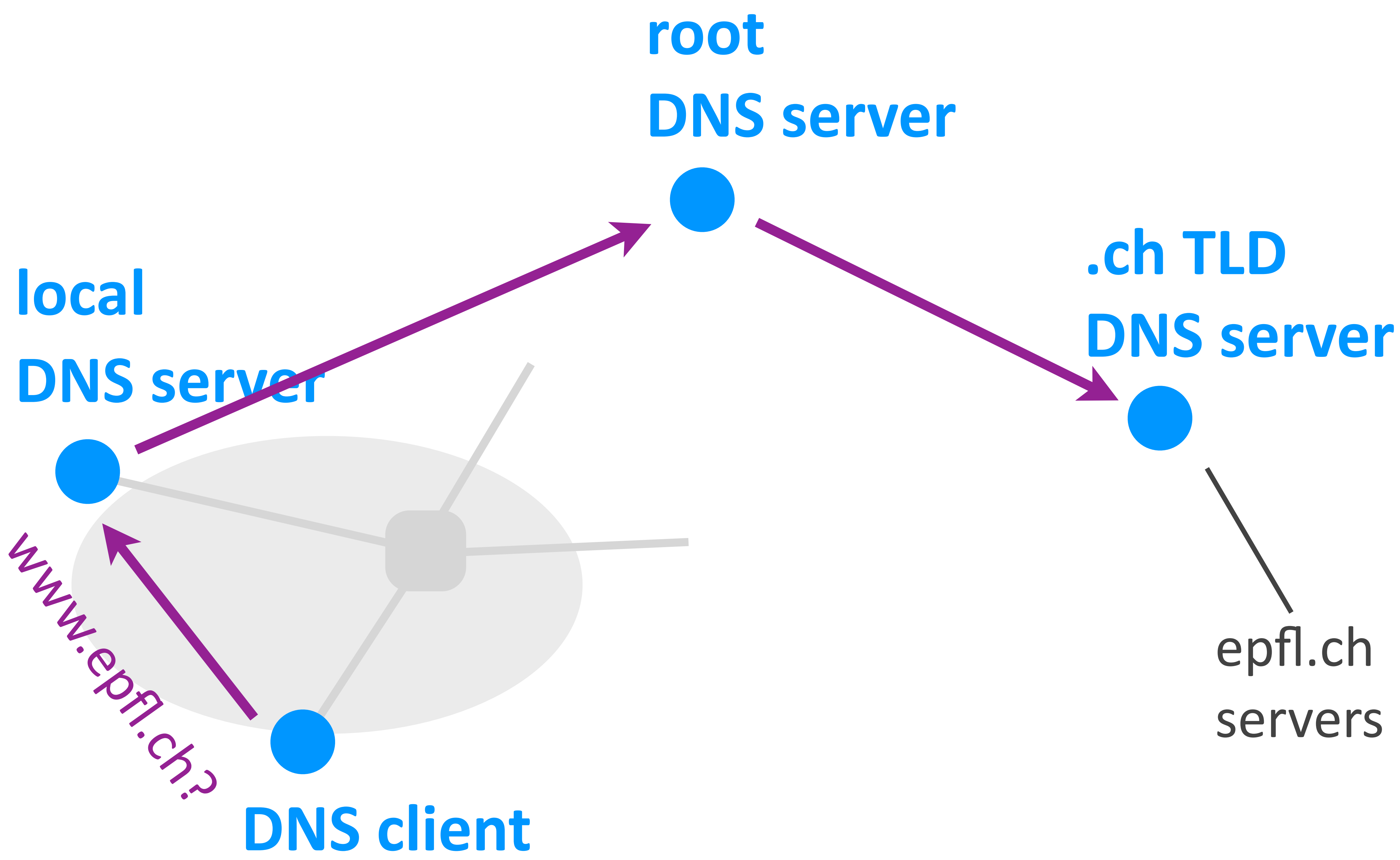


Hierarchy of DNS servers

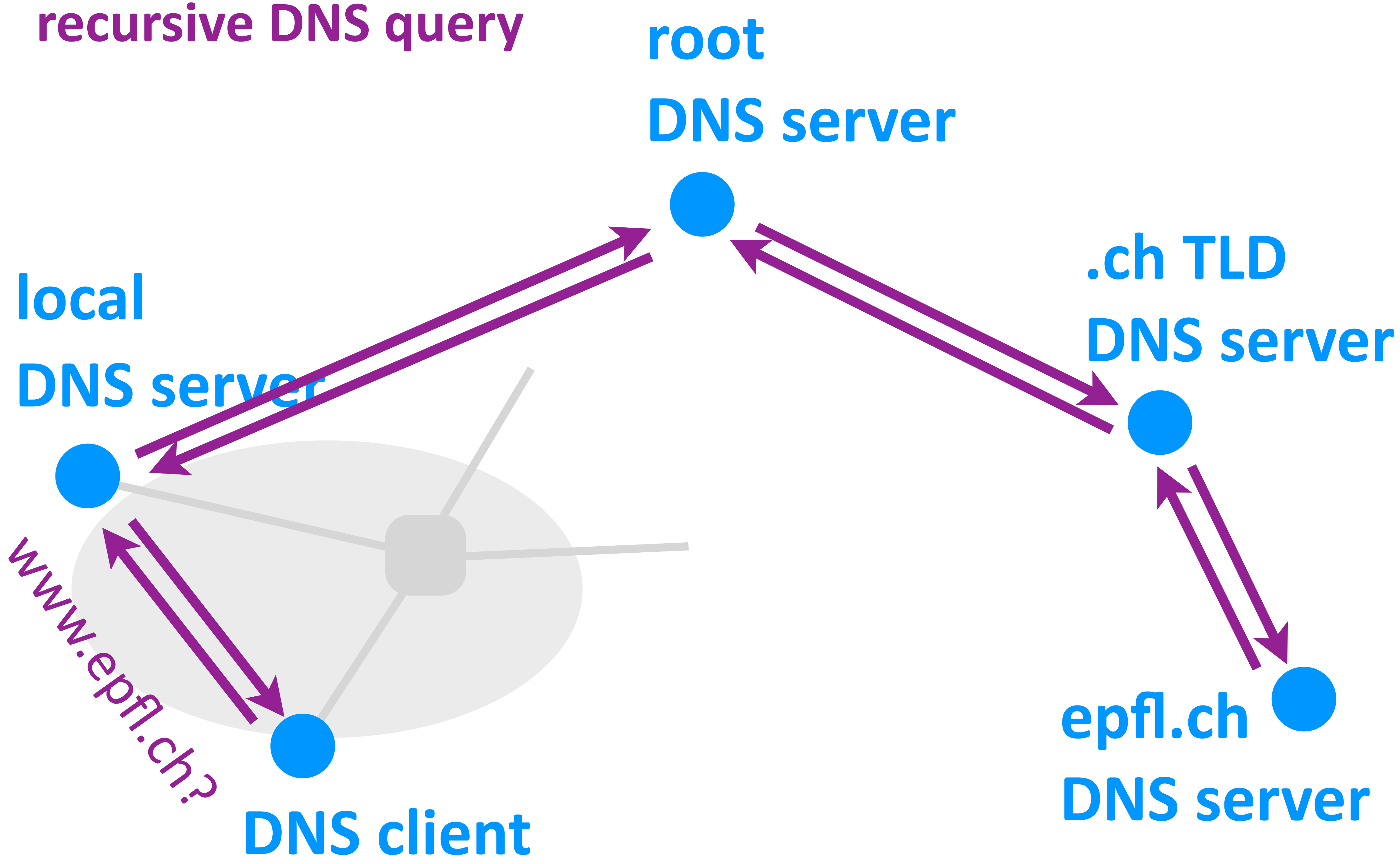


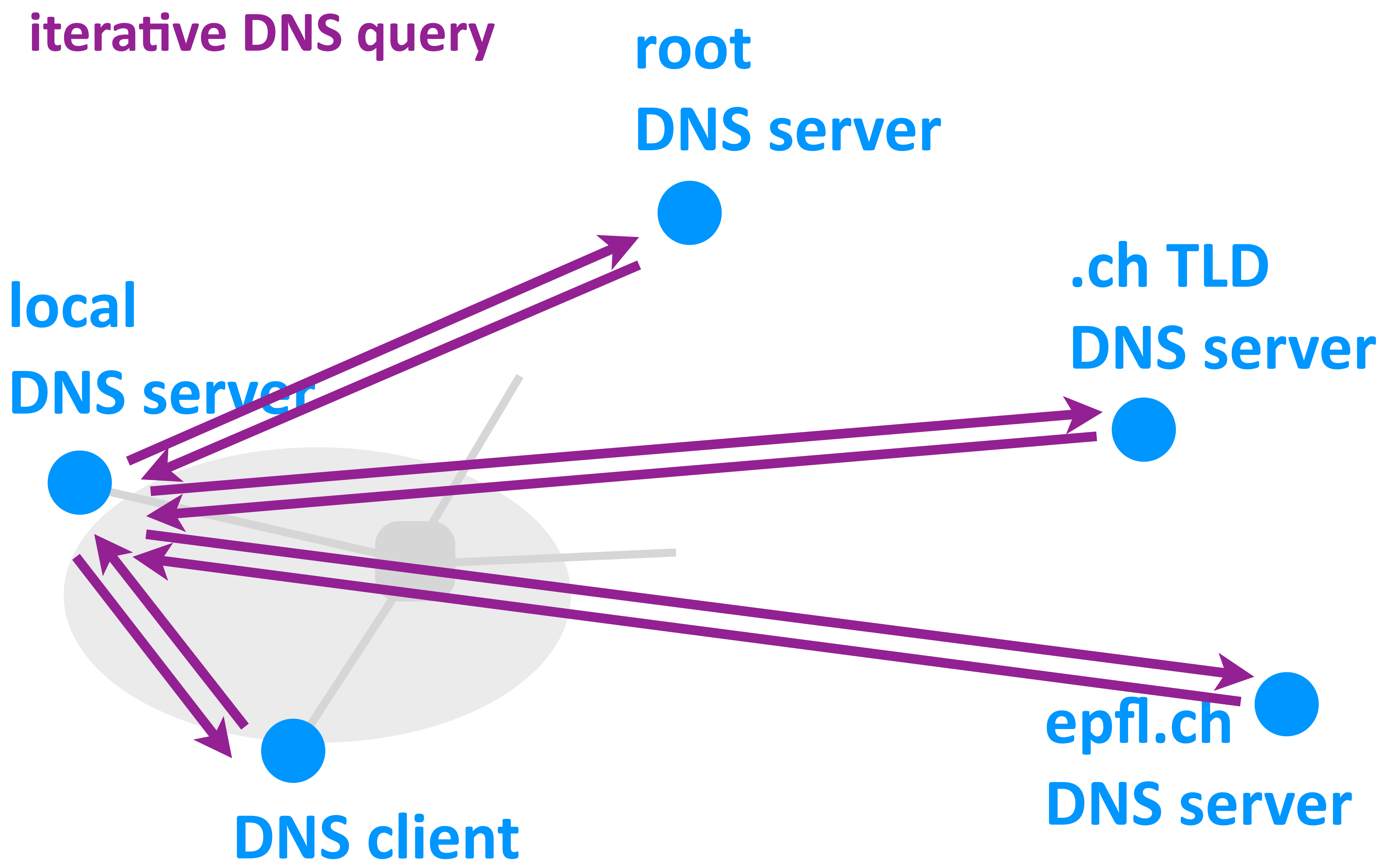

root servers
.ch servers
epfl.ch servers
local DNS server
www.epfl.ch?
DNS client


root
DNS server
local
DNS server
.ch servers
epfl.ch
servers
www.epfl.ch?
DNS client


root
DNS server
.ch TLD
DNS server
local
DNS server
www.epfl.ch?
DNS client
epfl.ch
servers

recursive DNS query



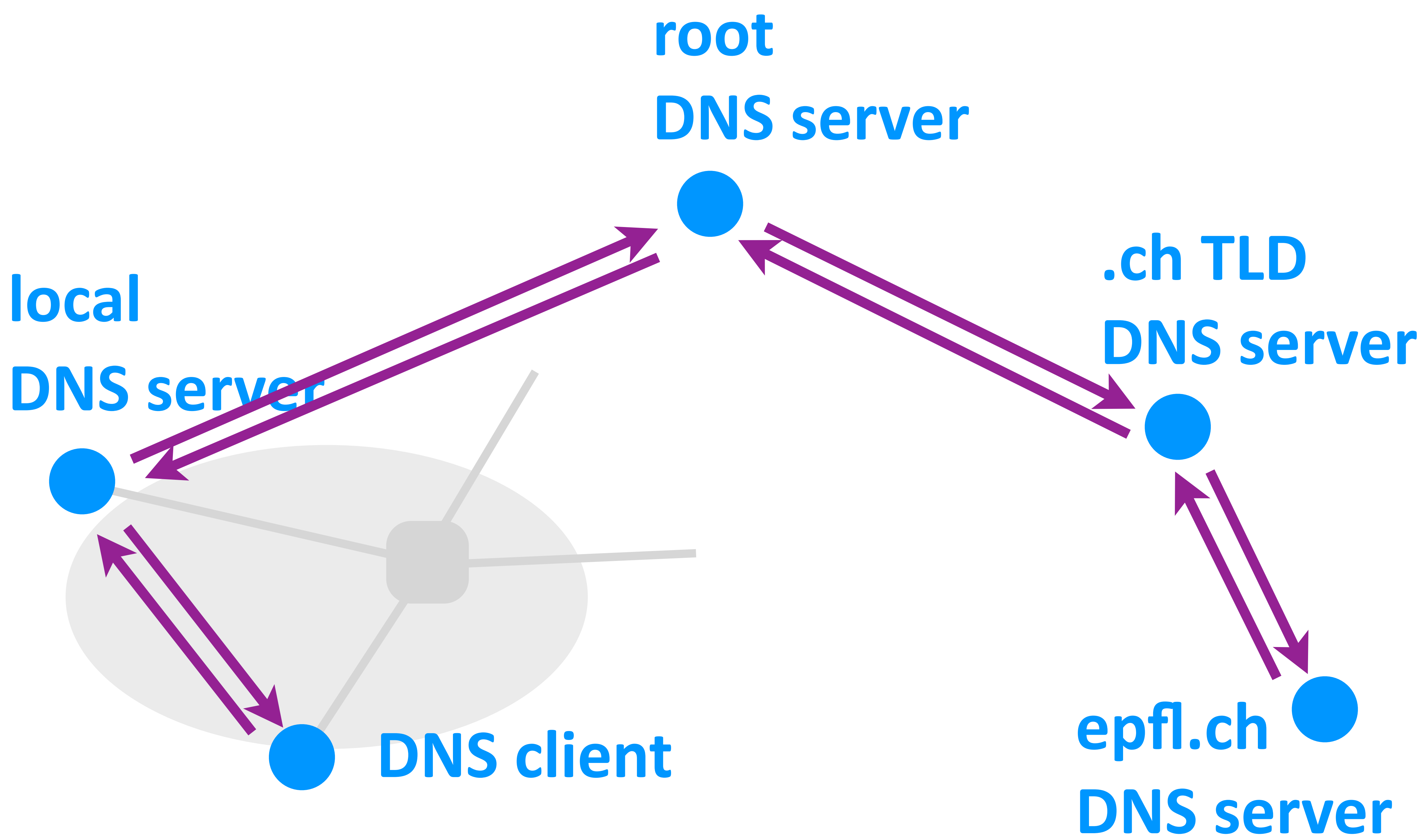

iterative DNS query
root
DNS server
.ch TLD
DNS server
local
DNS server
epfl.ch
DNS server
DNS client

DNS processes

- DNS client
  - *helps apps map hostnames to IP addresses*
- Local DNS server
  - *answers queries from nearby DNS clients*
- Hierarchy of DNS servers
  - *answers queries form local DNS servers*

Hierarchy of DNS servers

- Three levels
  - *root, TLD, authoritative DNS servers*
- Each level talks only to one level down
  - *root server knows which TLD server to query*
  - *TLD server knows which authoritative server*


root
DNS server
.ch TLD
DNS server
local
DNS server
DNS client
epfl.ch
DNS server

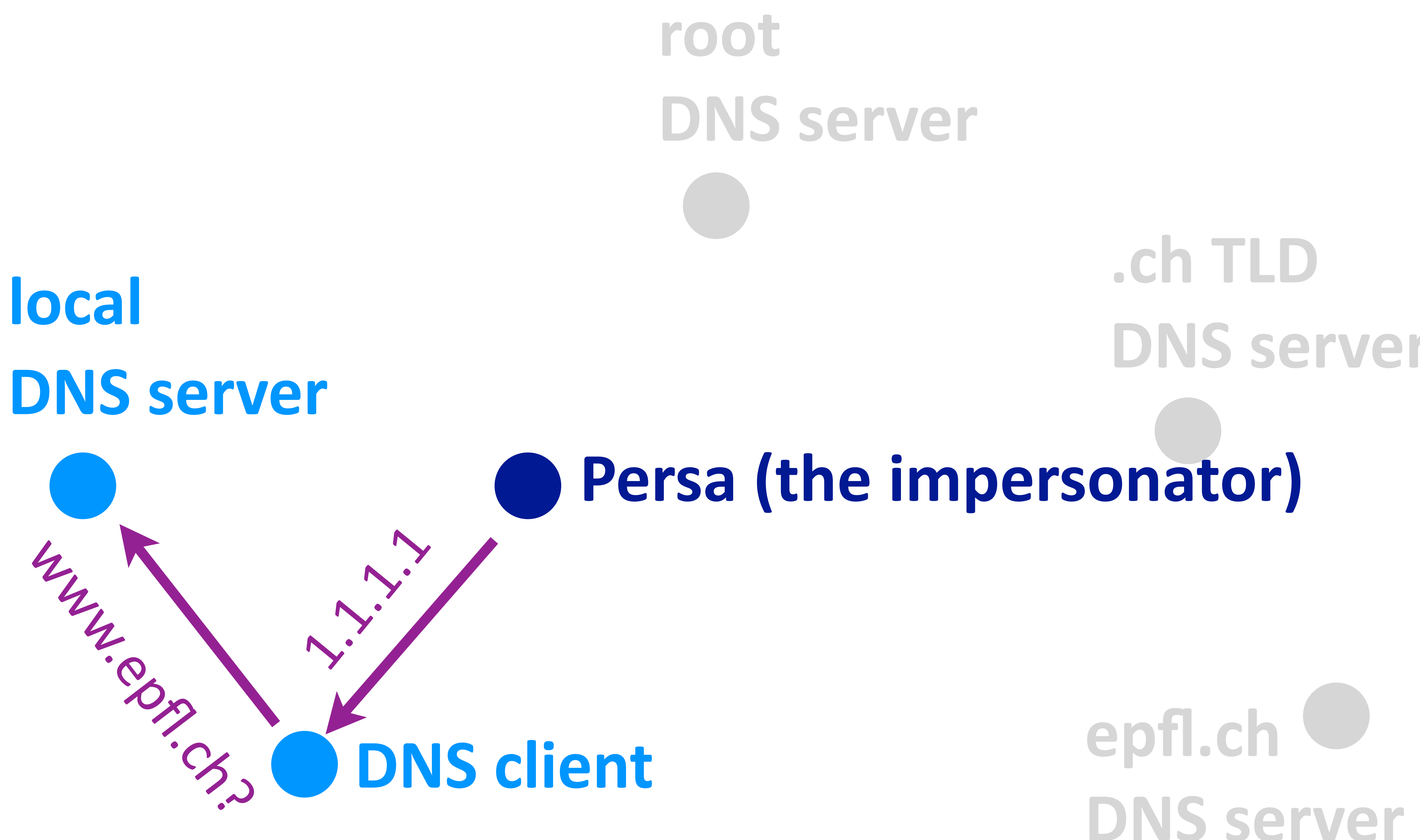
Caching

- Caching of DNS responses
  at all DNS servers + clients

- Reduces load at all levels

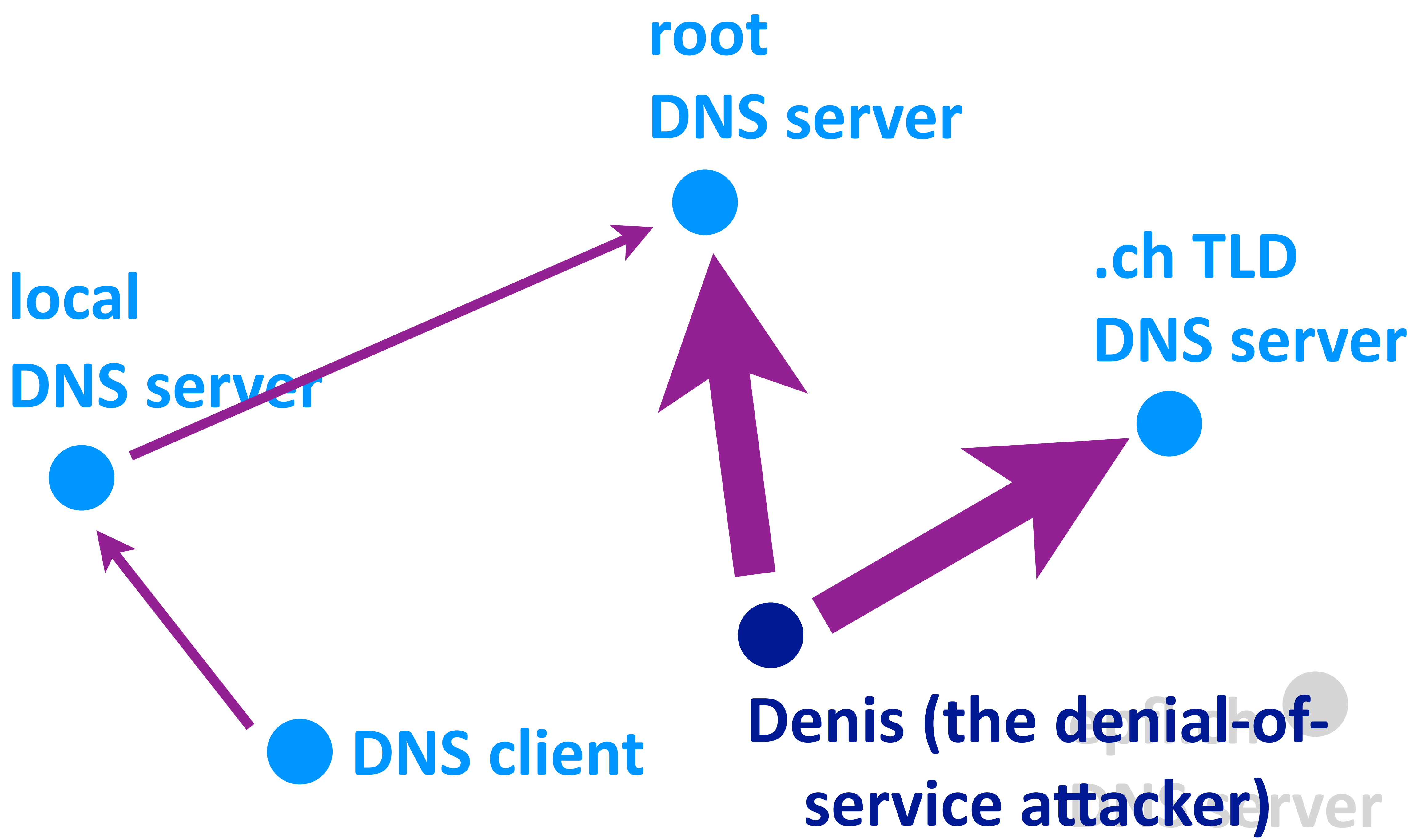
- Reduces delay experienced by DNS clients

How can one attack DNS?


root
DNS server
.ch TLD
DNS server
local
DNS server
Persa (the impersonator)
www.epfl.ch?
1.1.1.1
DNS client
epfl.ch
DNS server

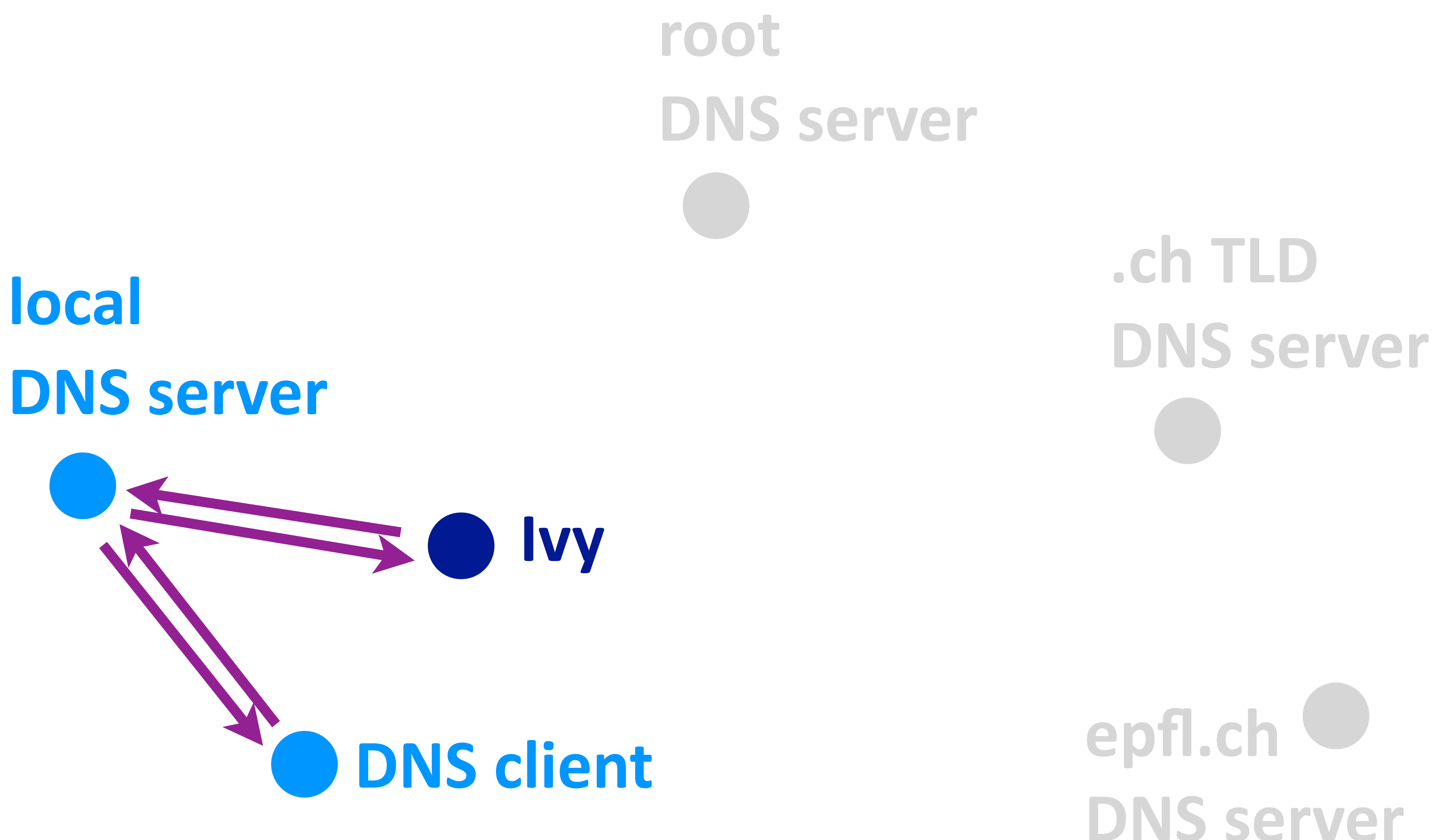
How can one attack DNS?

- Impersonate the local DNS server
  - *give the wrong IP address to the DNS client*


root
DNS server
.ch TLD
DNS server
local
DNS server
DNS client
Denis (the denial-of-
service attacker)

How can one attack DNS?

- Impersonate the local DNS server
  - *give the wrong IP address to the DNS client*
- Denial-of-service the root or TLD servers
  - *make them unavailable to the rest of the world*


root
DNS server
.ch TLD
DNS server
local
DNS server
Ivy
DNS client
epfl.ch
DNS server

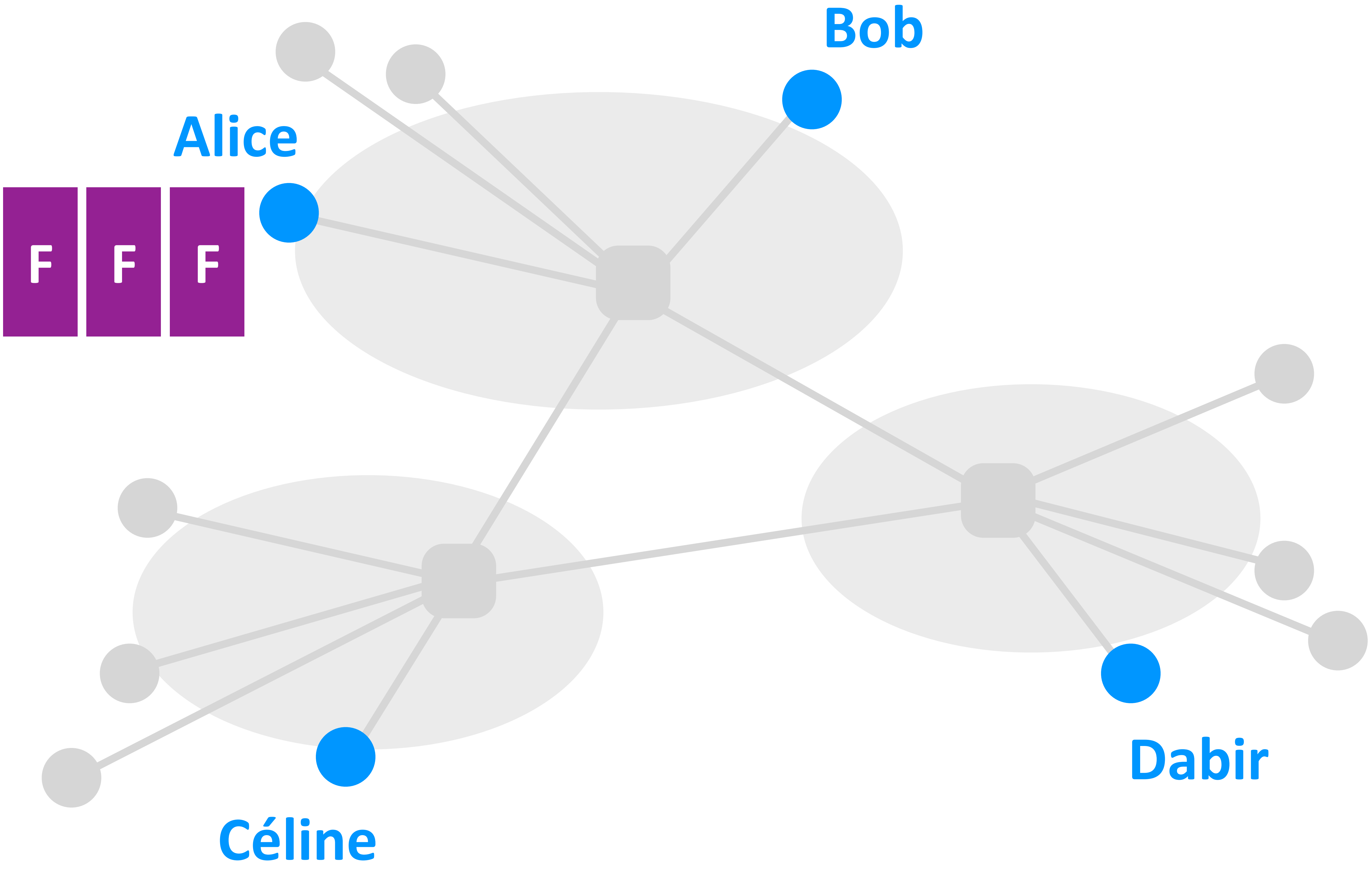
How can one attack DNS?

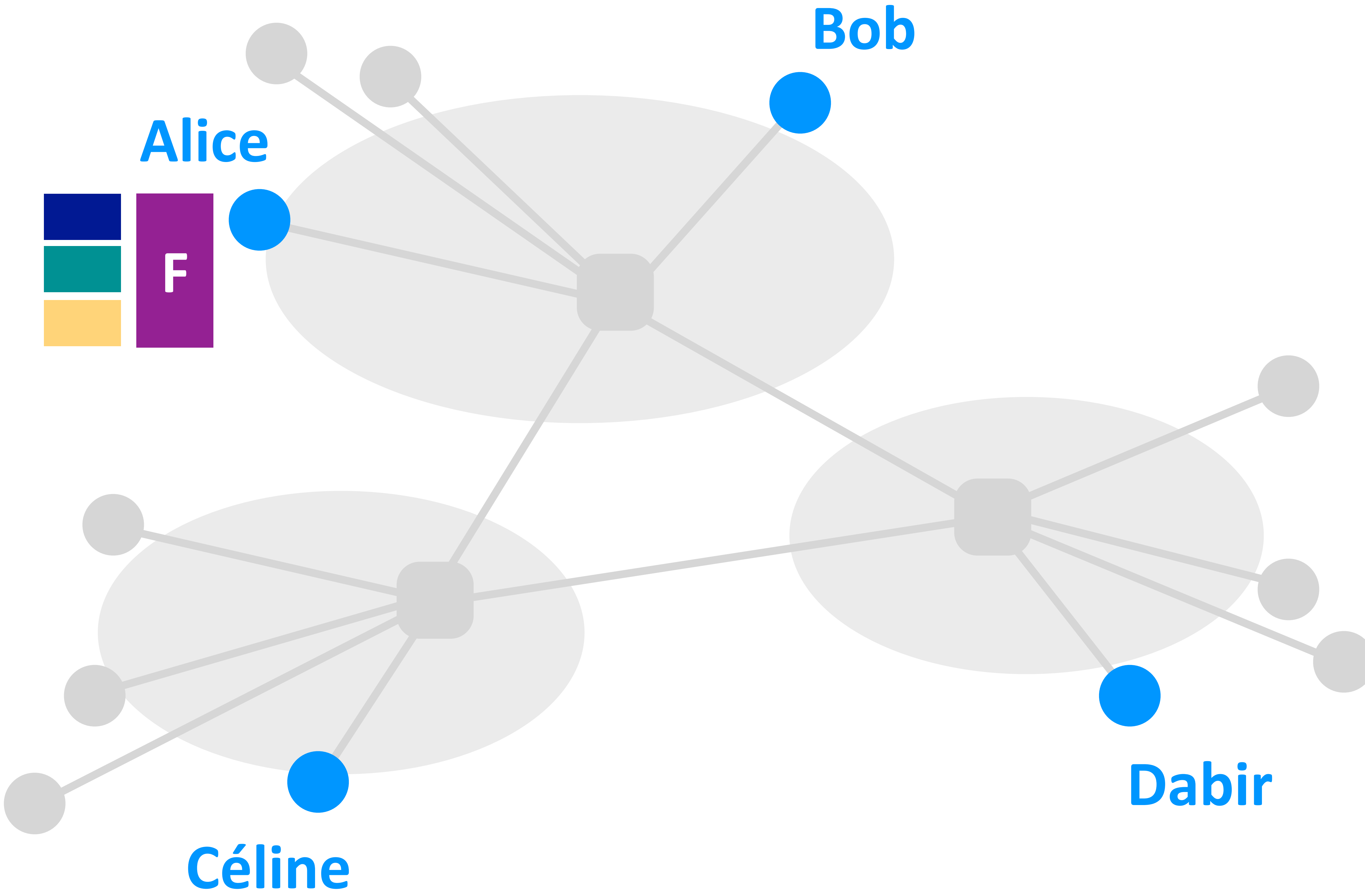
- Impersonate the local DNS server
  - *give the wrong IP address to the DNS client*
- Denial-of-service the root or TLD servers
  - *make them unavailable to the rest of the world*
- Poison the cache of a DNS server
  - *increase the delay experienced by DNS clients*

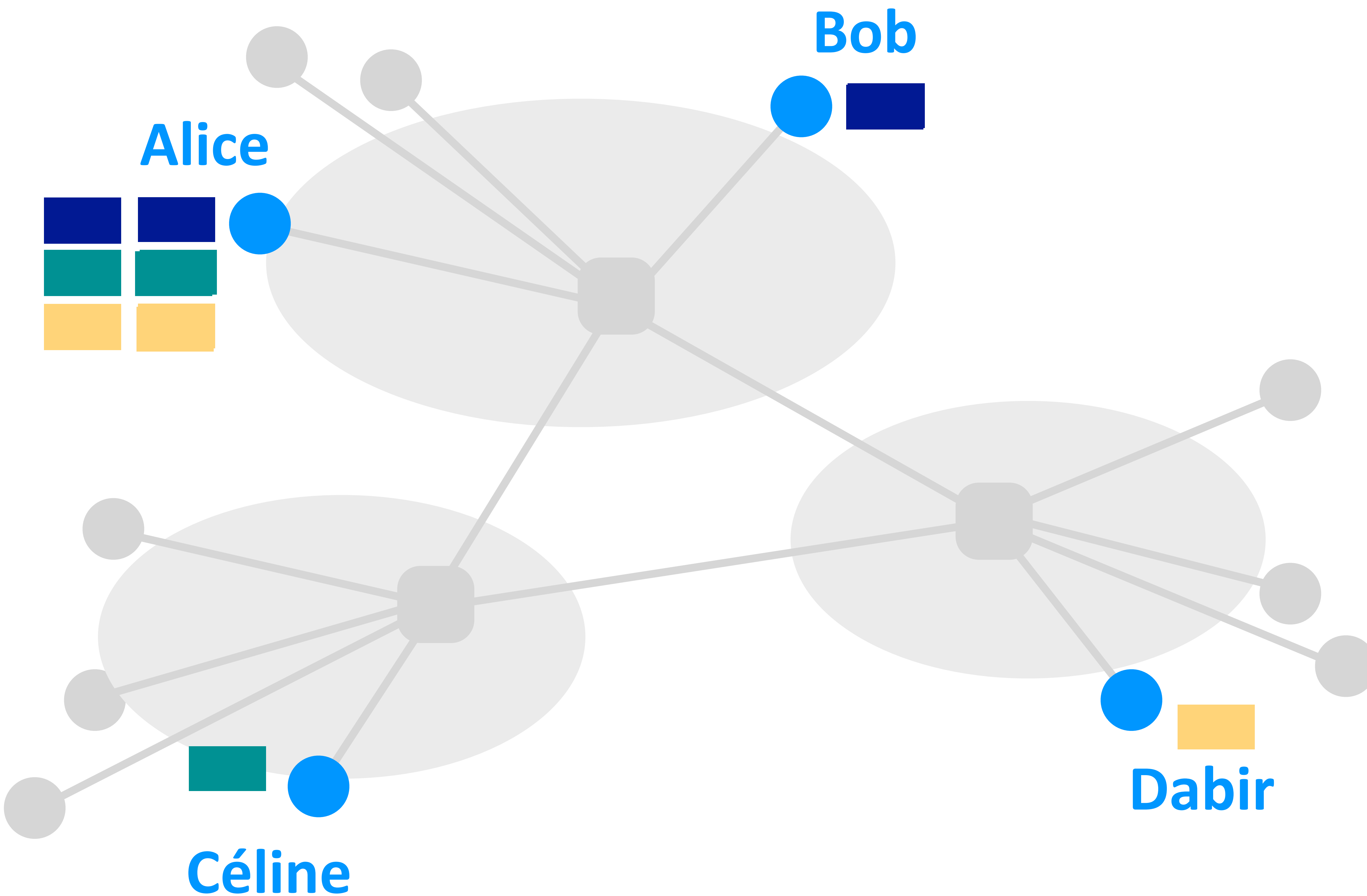
Outline

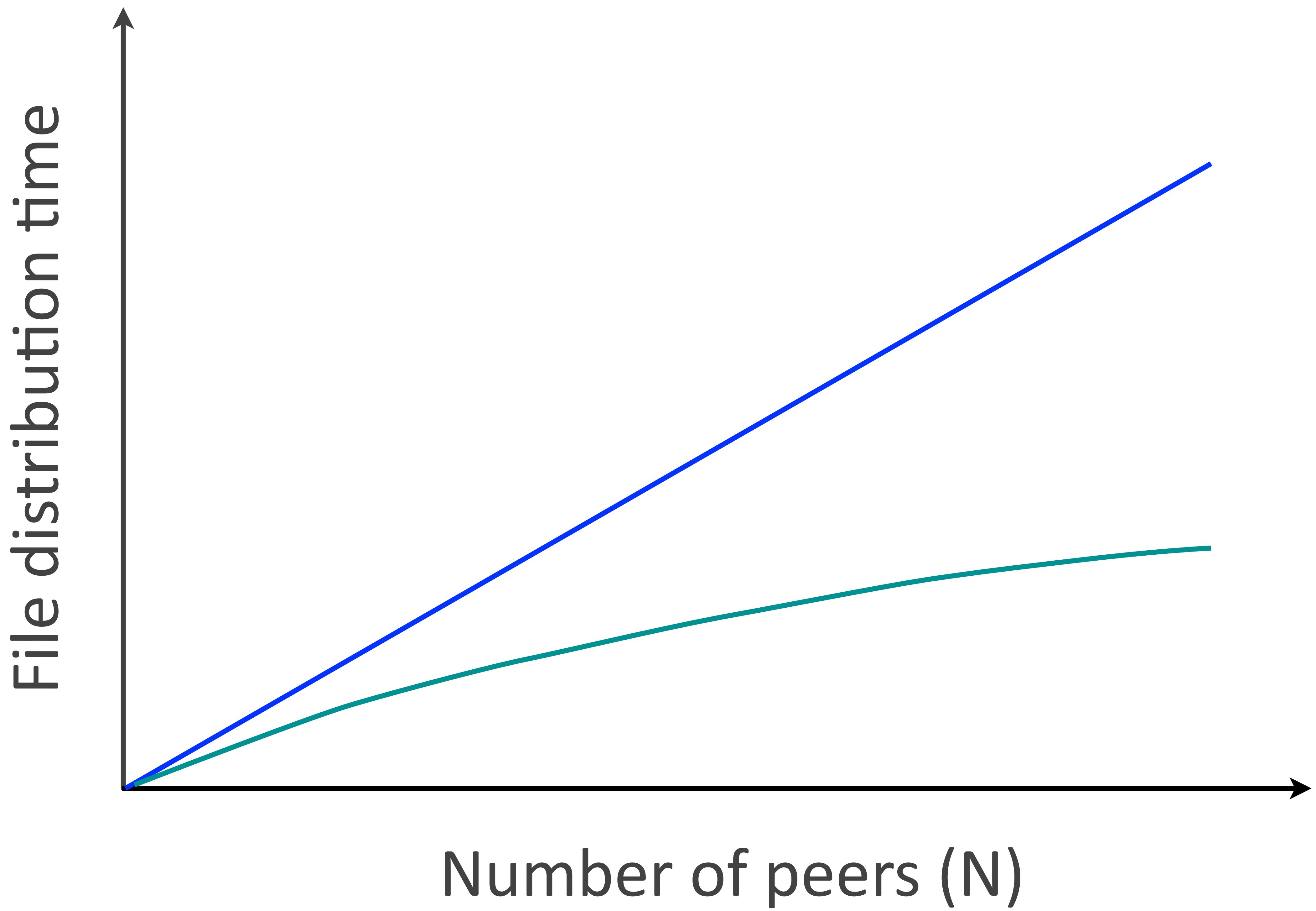
- Client-server vs. peer-to-peer
- Example 1: web
- Examlpe 2: DNS
- Example 3: P2P file sharing

Example 3: P2P file sharing


Bob
Alice
F
F
F
Céline
Dabir


Bob
Alice
F
Dabir
Céline


Bob
Alice
Céline
Dabir


File distribution time
Number of peers (N)

File distribution

- Client-server: time increases linearly with the number of clients
- Peer-to-peer: time increases sub-linearly with the number of peers

Informally:

System does not scale =

does not work well with many users

you cannot simply add resources to fix it