Lecture 2:

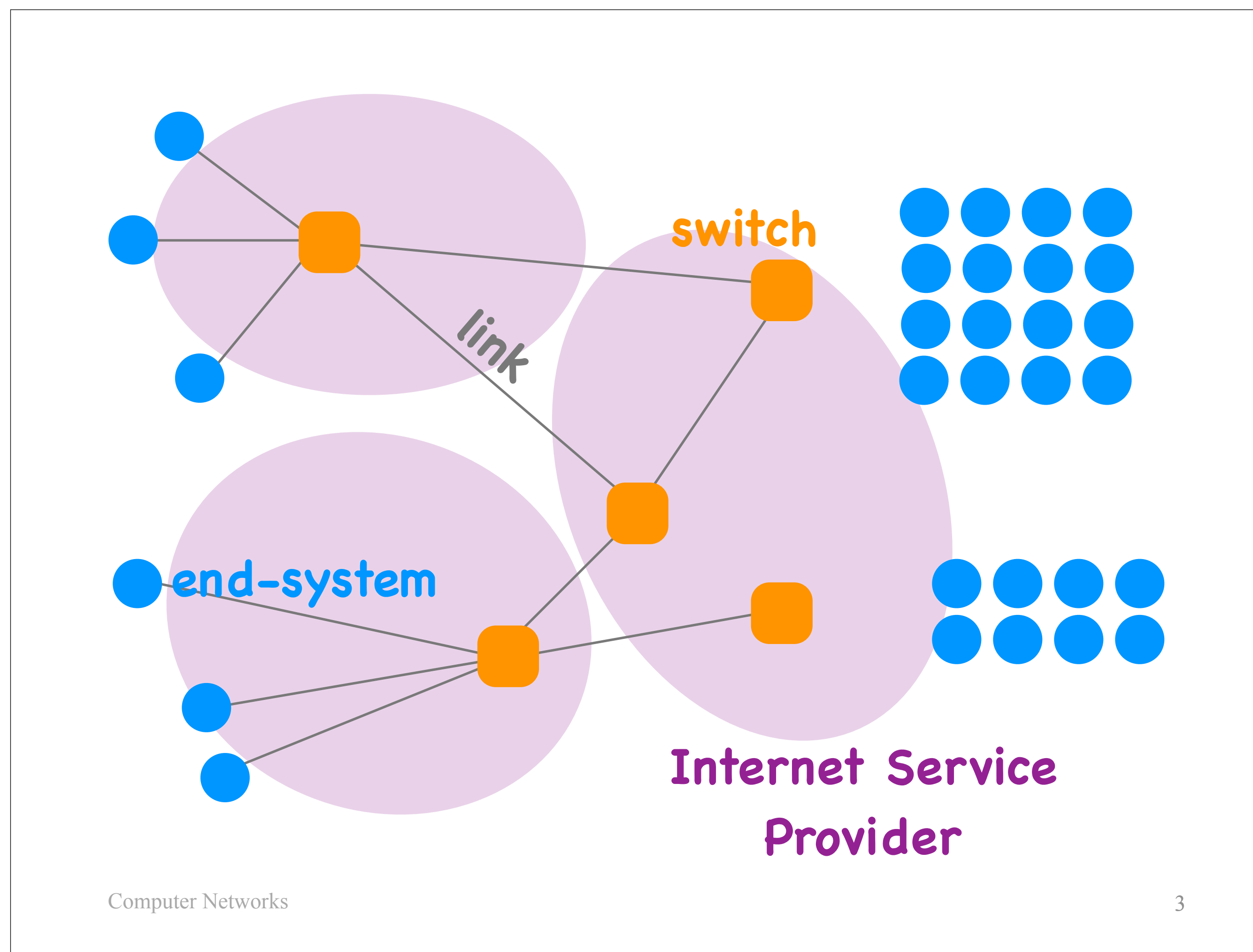
Introduction (part 2)

Katerina Argyraki, EPFL

Questions

- What's underneath?

Last week we started exploring basic questions about the Internet architecture.
We first asked what lies underneath? What infrastructure is used when end-systems communicate over the Internet? The answer is...

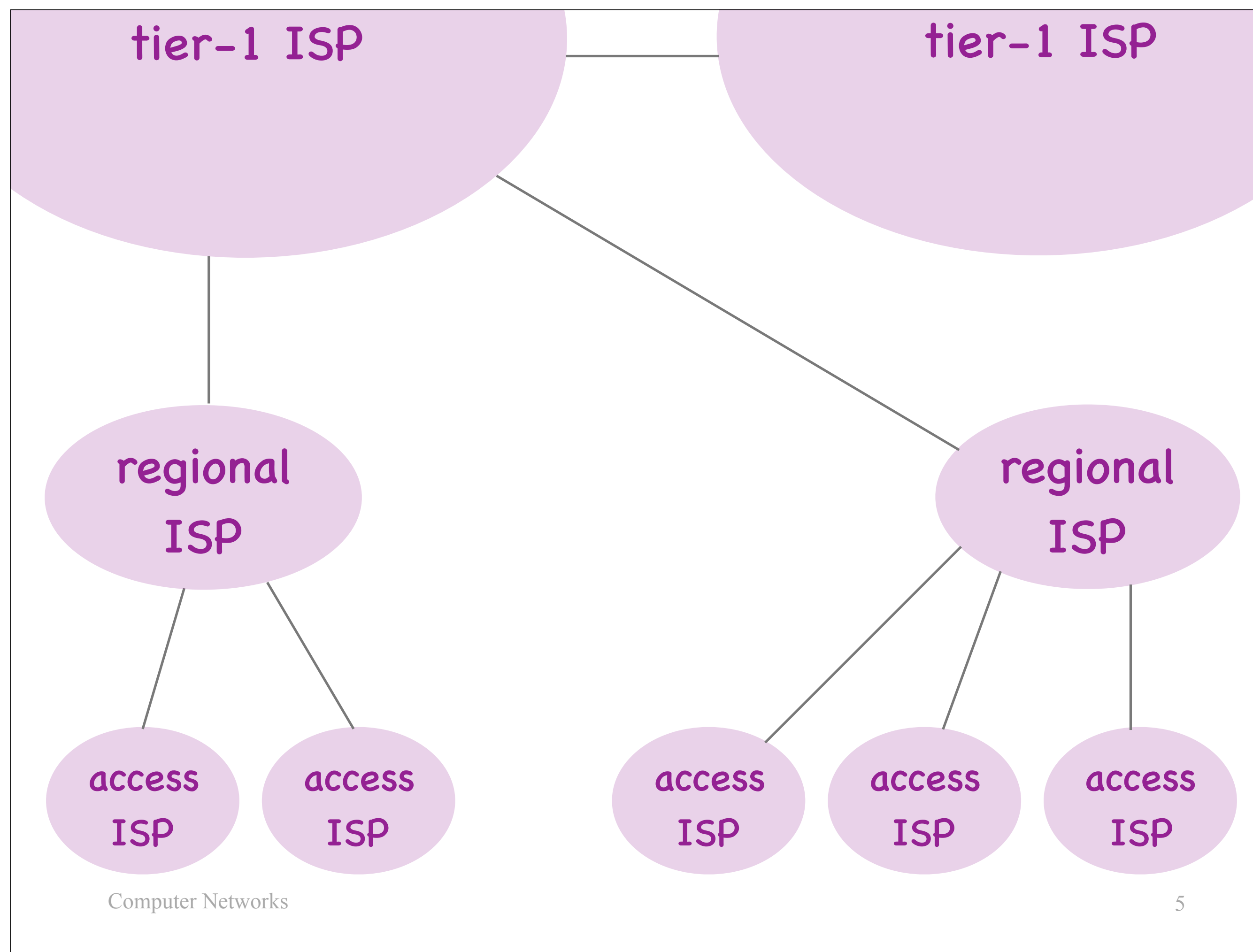


packet switches, which are the devices that interconnect end-systems;
network links, which connect packet switches to end-systems and between them;
and Internet Service Providers (ISPs), which own and manage most of these packet switches and network links.

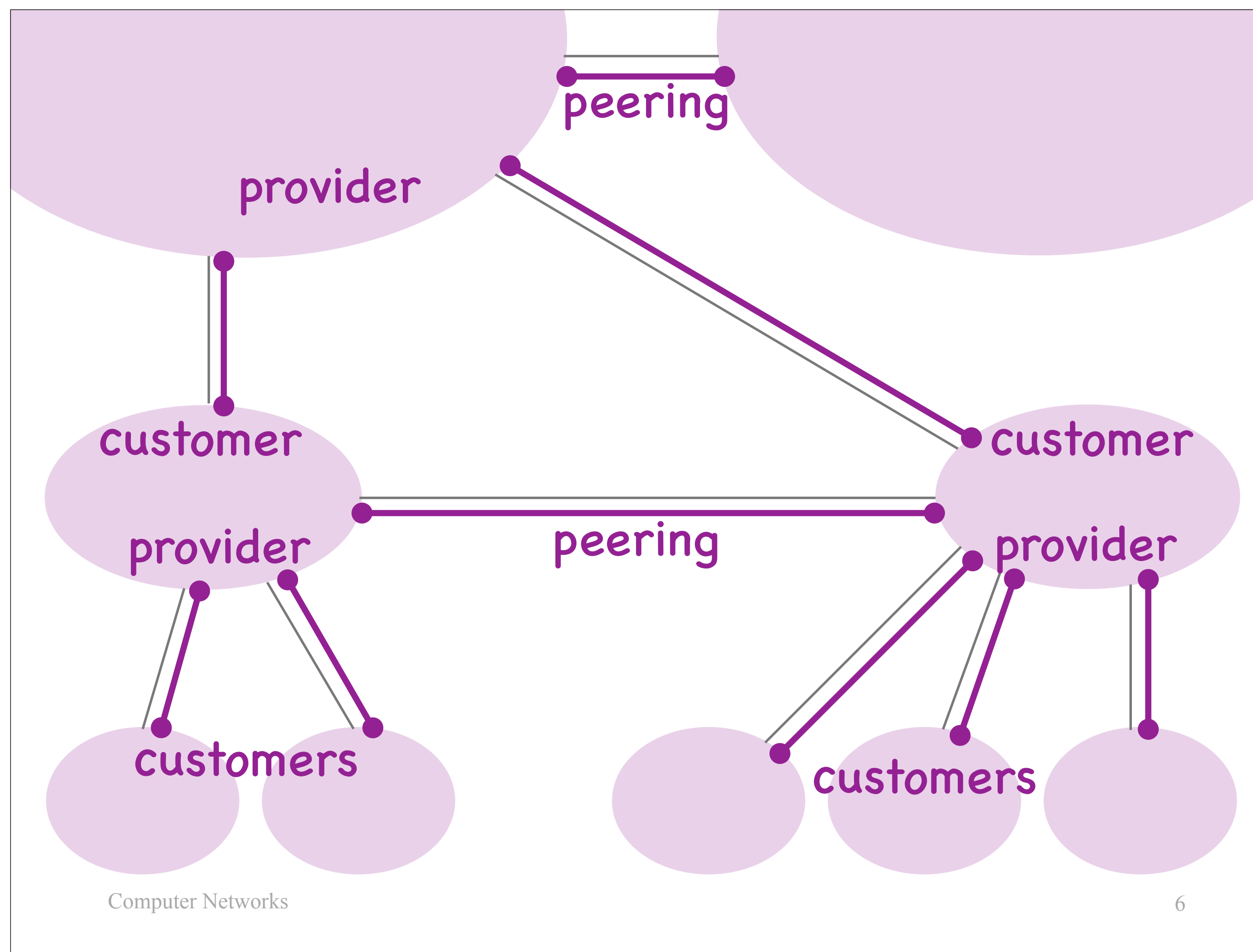
Questions

- What's underneath?
- Who owns what?

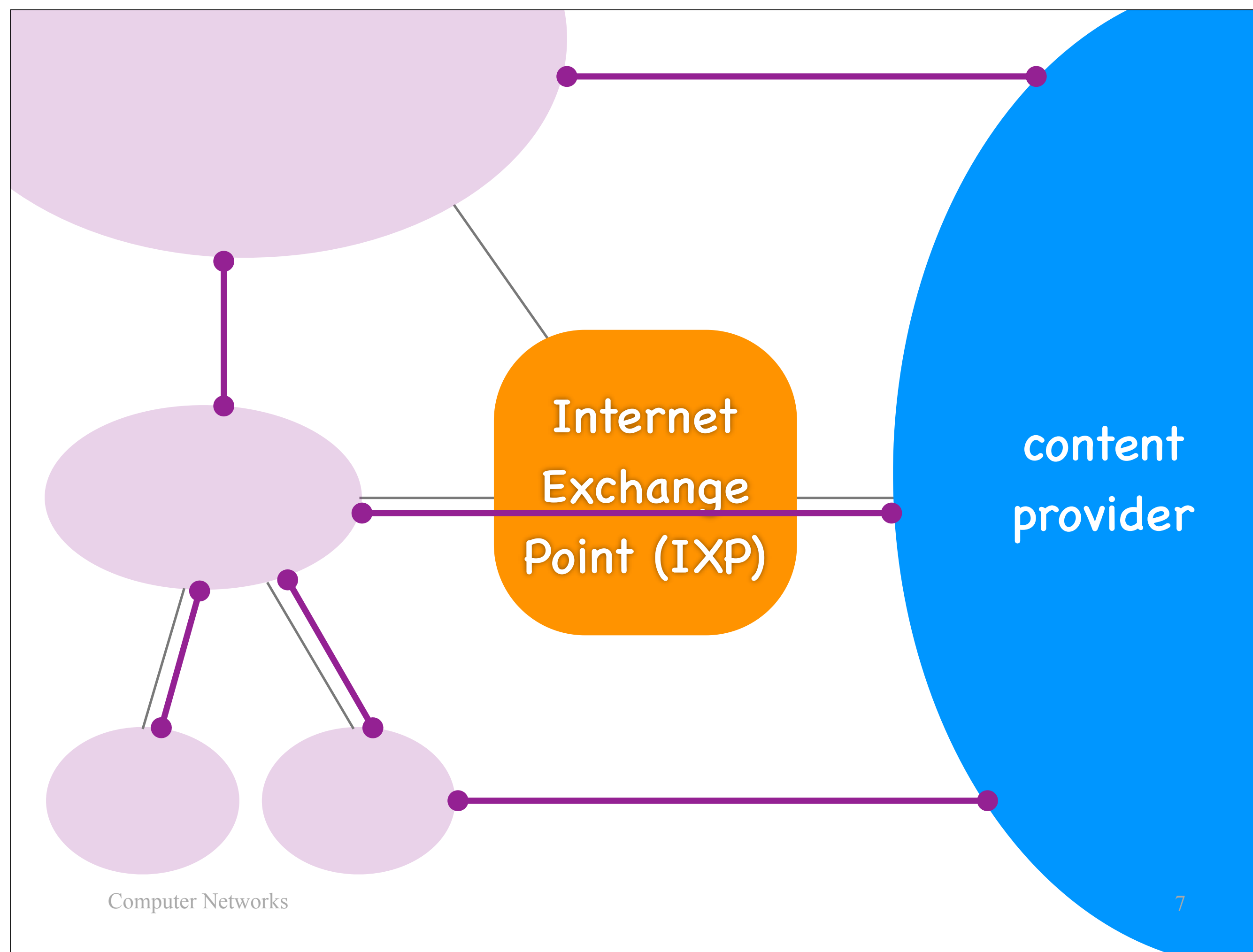
Then we spend some time discussing ISP relationships.



We said that ISPs were originally connected in a 3-level hierarchy, with many access ISPs at the bottom, fewer regional ISPs on top, and even fewer Tier-1 ISPs on top.



We also said that this hierarchy is not only physical, but also economic: Access ISPs are customers of regional ISPs, which are customers of Tier-1 ISPs, while Tier1 ISPs have peering relationships with each other. And sometimes, smaller ISPs may peer and exchange traffic directly, rather than pay their Tier-1 ISPs.



Then we complemented this picture with two elements:

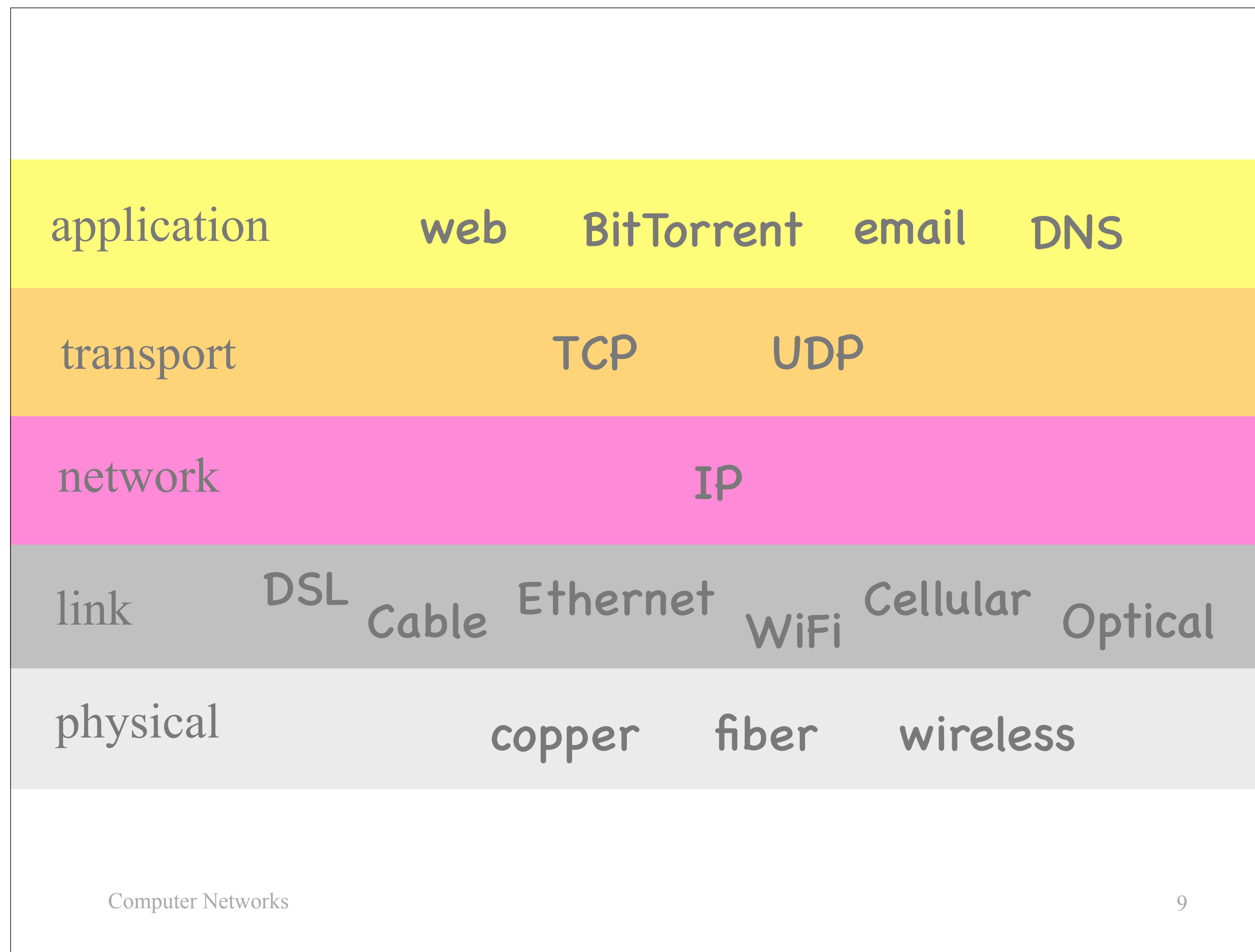
Internet eXchange Points (IXPs), which are essentially giant switches that provide physical connections between ISPs.

And big content providers, who have built their own networks and peer with ISPs at all levels of the hierarchy.

Questions

- What's underneath?
- Who owns what?
- How does it work?

The last question we explored was: How does the Internet work? What happens under the covers when end-systems exchange messages? The answer is…

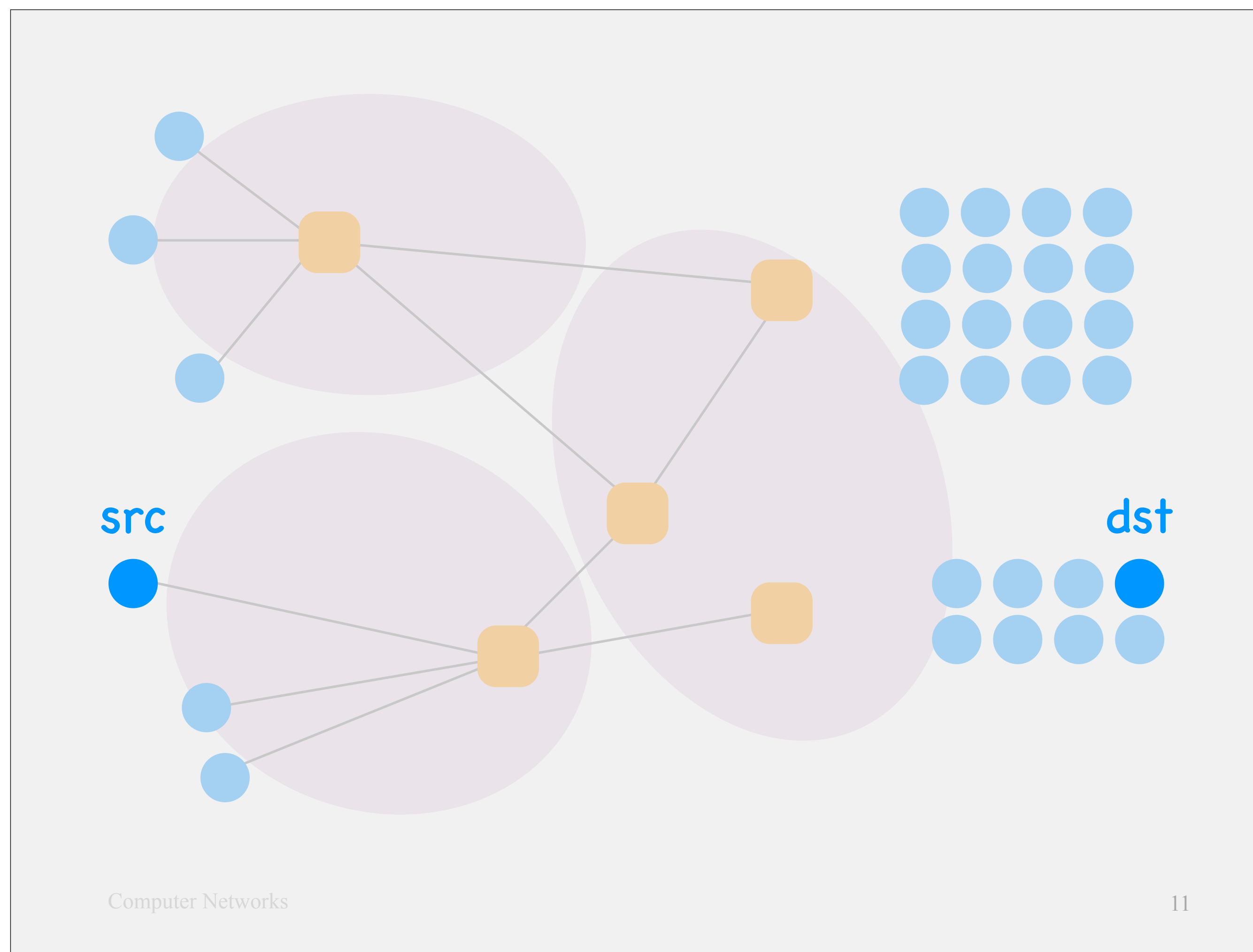


… layers. The Internet architecture is organised in 5 distinct layers that we will discover, one by one, over the semester.

Questions

- What's underneath?
- Who owns what?
- How does it work?
- How does one evaluate it?
- How do end-systems share it?

Now we will explore a new question: how does one evaluate the performance of Internet communication?



Consider a source end-system that is sending data to a destination end-system over the Internet.

Basic performance metrics

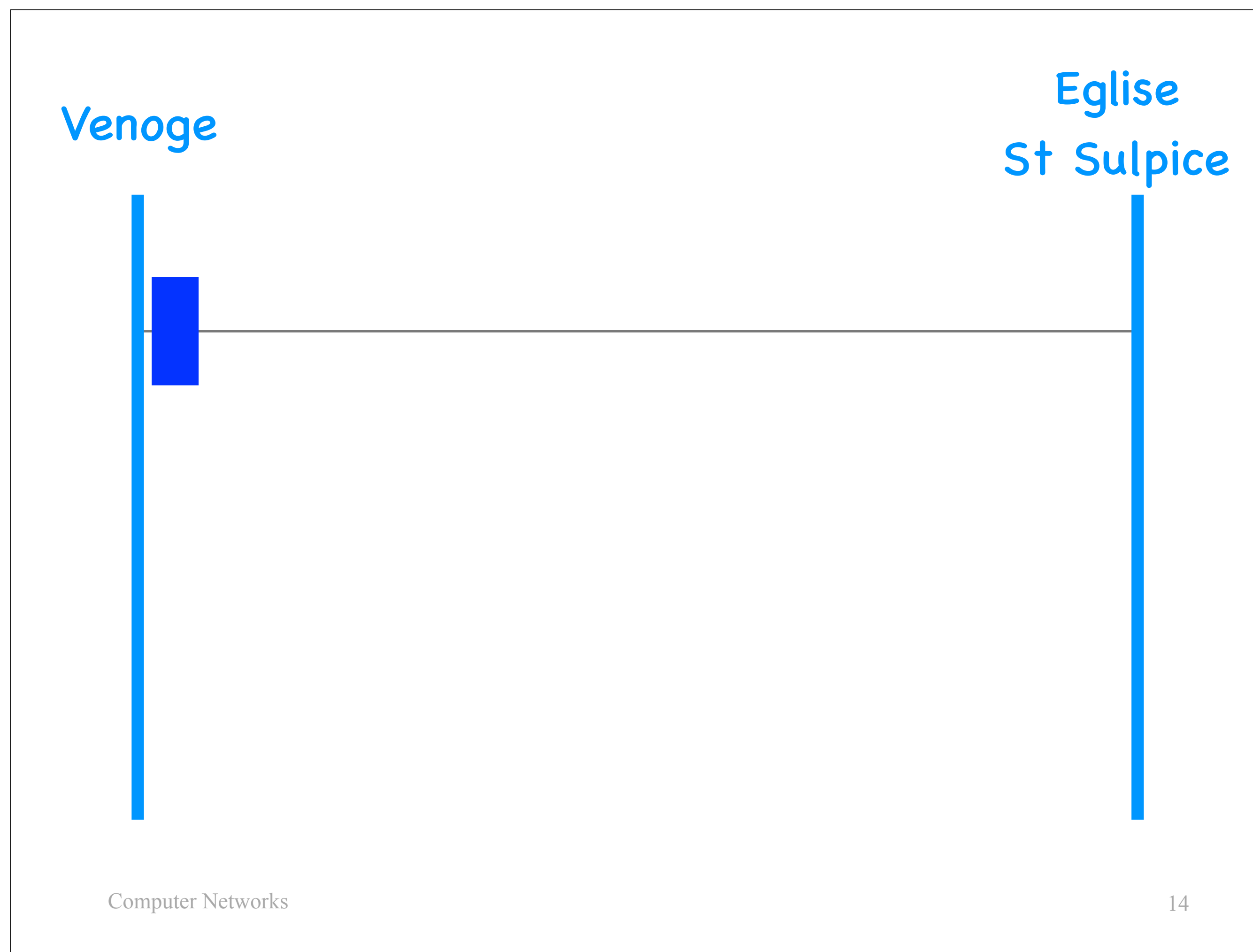
- Packet loss
  - the fraction of packets from src to dst that are lost on the way
  - in %, e.g., 1% packet loss
- Packet delay
  - the time it takes for a packet to get from src to dst
  - in time units, e.g., 10 msec

We use three simple performance metrics to quantify the quality of this communication: packet loss…, packet delay…,

Basic performance metrics

- Average throughput
  - the average rate at which dst receives data
  - in bits per second (bps)
  - e.g., dst receives 1 GB of data in 1 min; average throughput = 8 10^9 bits / 60 sec = 133.34 10^6 bps = 133.34 Mbps

… and average throughput.

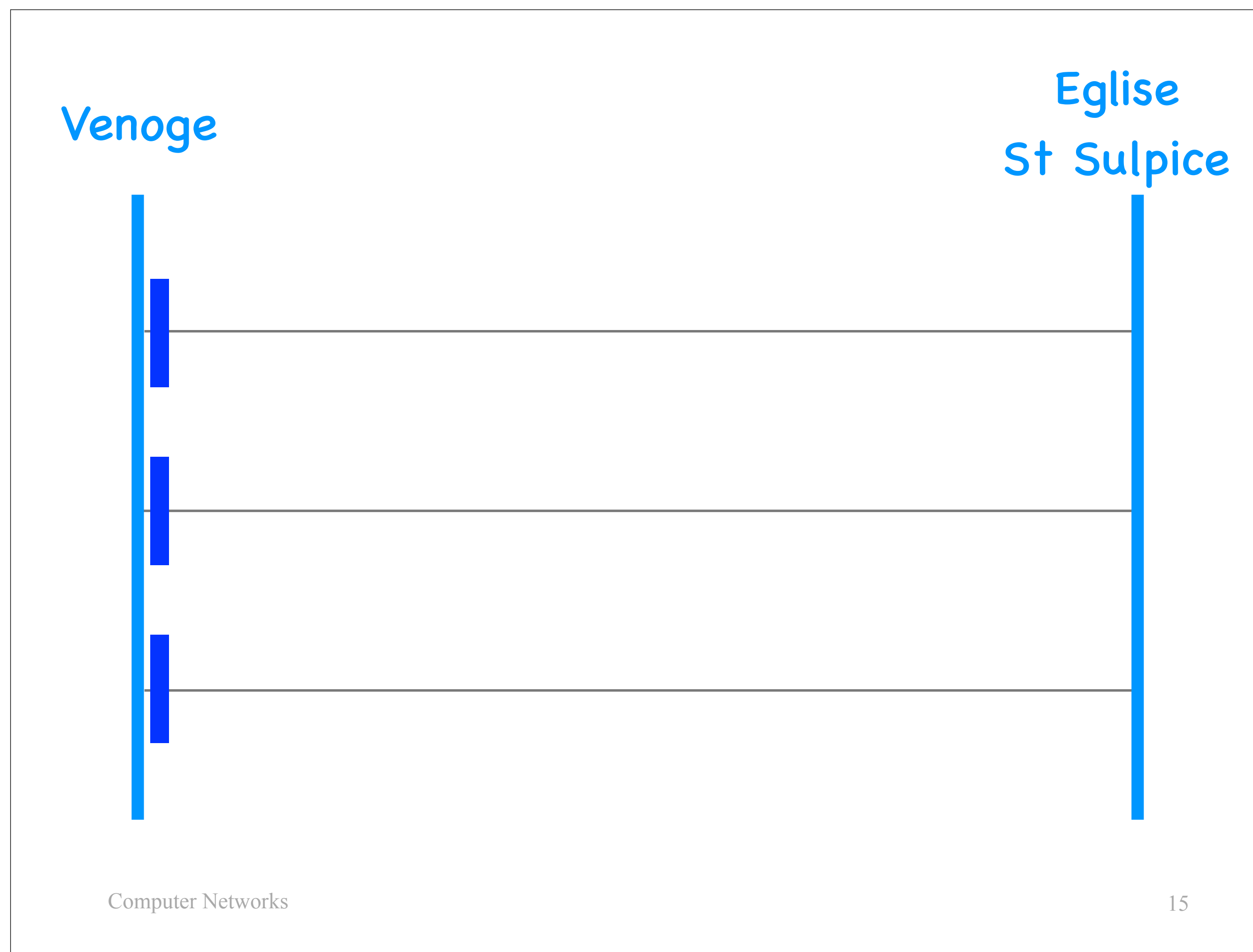


Let's discuss a little bit the difference between delay and throughput.

Consider the path by the lake that connects the Venoge river and the church in St Sulpice. In several places, this path becomes so narrow that it fits a single person.

Now consider a group of 3 friends and suppose it takes them 15 min to traverse the path, which seems to them too long.

So, one of them suggests to create…

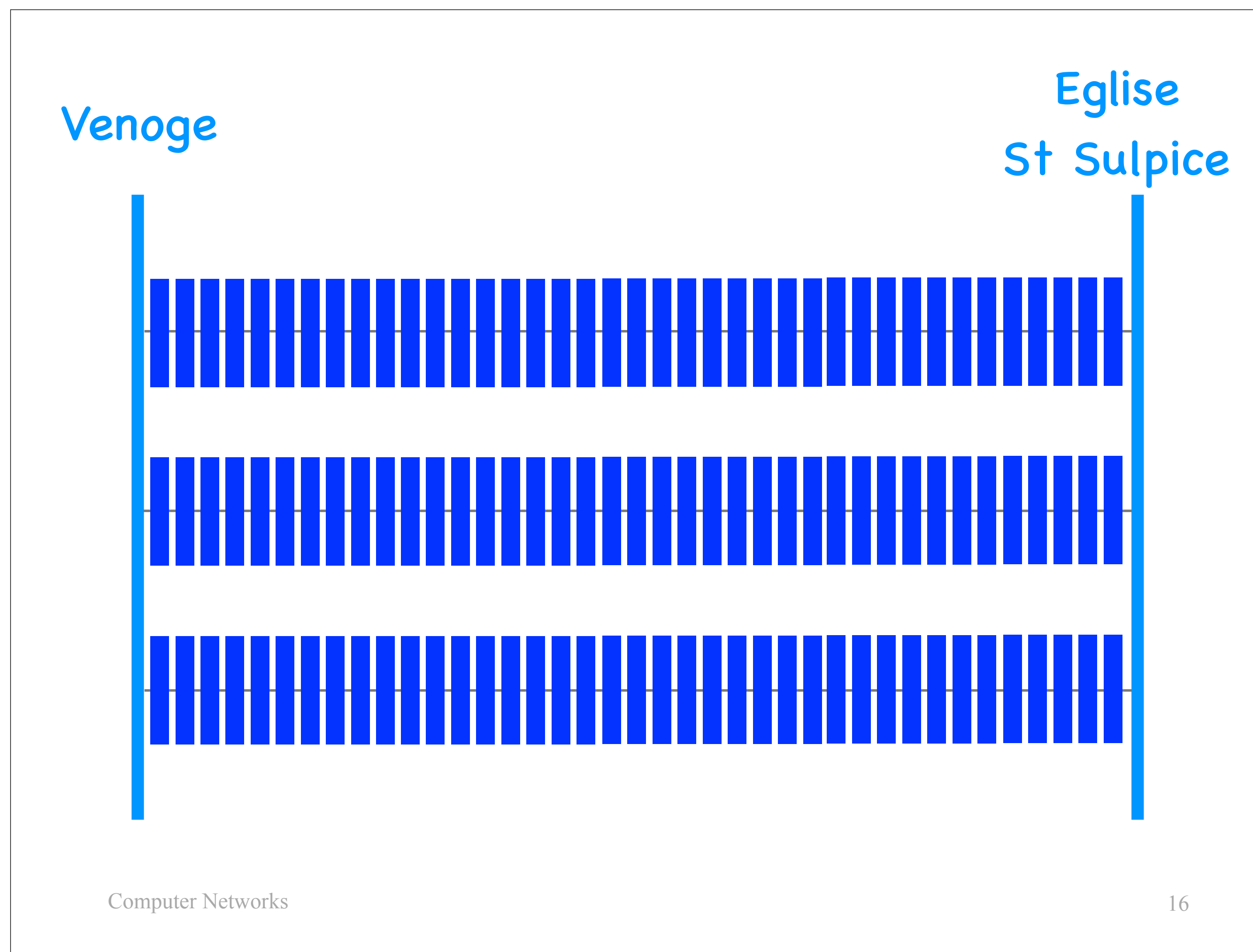


2 more paths of the same type between the two points.

—> Would this help the 3 friends get from one end to the other faster?
Only by an insignificant amount of time. Each friend could take a different path so they could proceed in parallel as opposed to one behind the other. However, the amount of time it would take for all 3 of them to reach the destination
would be almost the same as the amount of time it would take them if they were walking one behind the other.

Bottom line: By adding more paths of the same type, one can obviously not improve the delay for one person to get from one end to the other. And one cannot significantly improve the delay for a small group of people to get from one end to the other either.

The situation is different if we have a group of…



… 1000 people who want to get from one end to the other as fast as possible.
Then, of course, using multiple parallel paths would help significantly.

—> What metric do we improve by adding parallel paths between the two ends?
The throughput: the rate at which people get from one end to the other.

Delay vs. throughput

- Packet delay matters for small messages
- Average throughput matters for bulk transfers
- They are related to each other, but not in an obvious way

Back to computer networks:

Packet delay matters when end-systems need to exchange small messages fast,
e.g., in the context of an interactive application, like voice or gaming.

Average throughput matters when end-systems need to exchange a lot of data,
e.g., you are downloading the new Ubuntu distribution to your laptop.
In this case, what you care for is the amount of time it takes to download the whole distribution, not a few bits.

Estimating packet delay and average throughput between various end-systems under different scenarios is one of the main activities of network engineers.

src
dst
011

transmission delay

$$= \frac{\text{packet size}}{\text{link transmission rate}}$$

$$= \frac{\text{3 bits}}{\text{1 Gbps}} = \text{3 nsec}$$

Now we will compute packet delay in various scenarios.

In the first scenario, we have two end-systems, a source and a destination, are directly connected through a single link. The source wants to send a 3-bit packet to the destination. (In reality, there aren’t such small packets, but let’s pretend there are, to keep this simple.)

When the source transmits a packet over the link, it takes a certain amount of time to push all the bits of the packet onto the link. This amount of time is called the transmission delay, and it is equal to …

For example, if the link has transmission rate 1 Gbps, the transmission delay for 3 bits is 3 nsec.

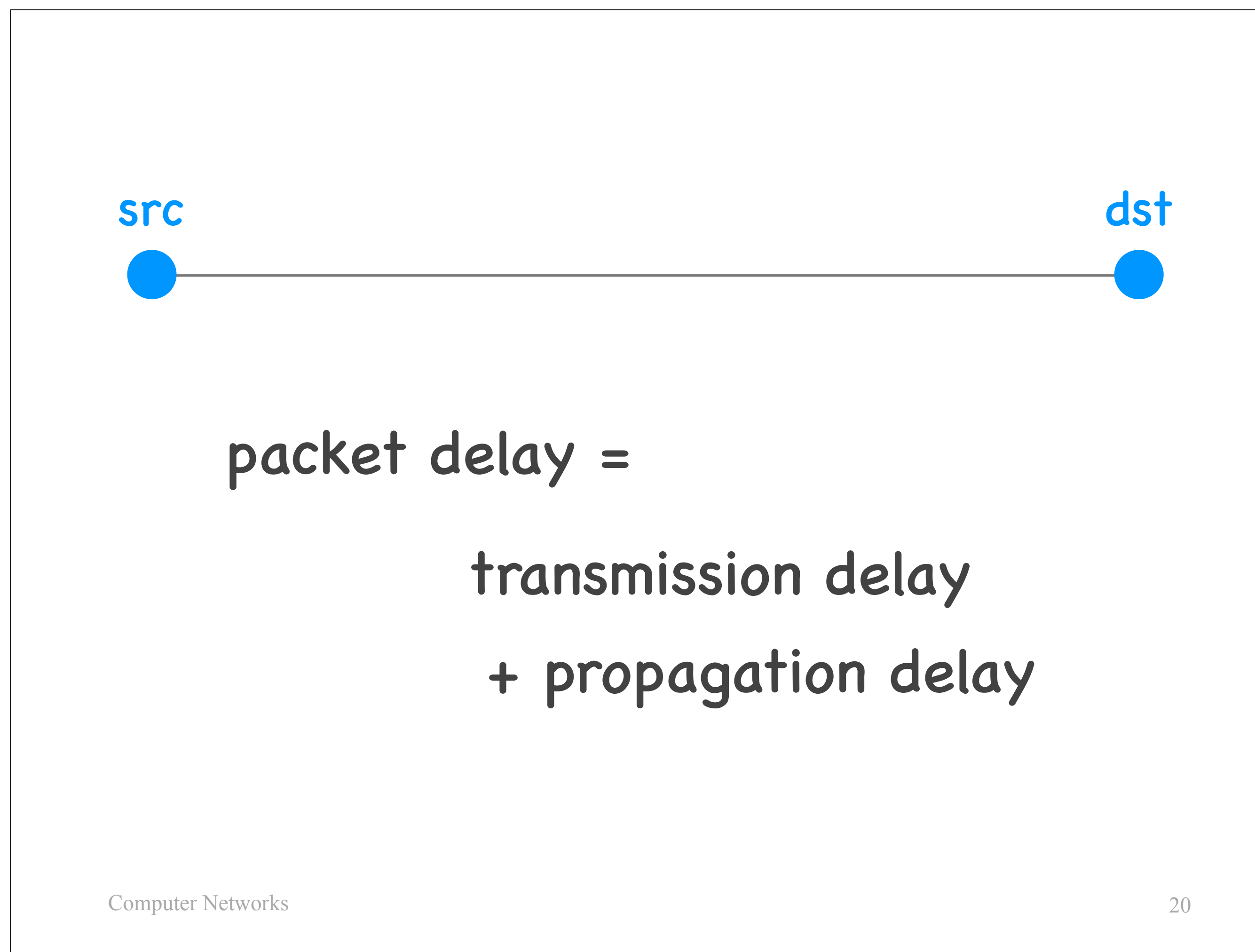
propagation delay

$$= \frac{\text{link length}}{\text{link propagation speed}}$$

$$= \frac{\text{1 meter}}{3\ 10^8 \text{ meters per sec}} = \text{3.34 nsec}$$

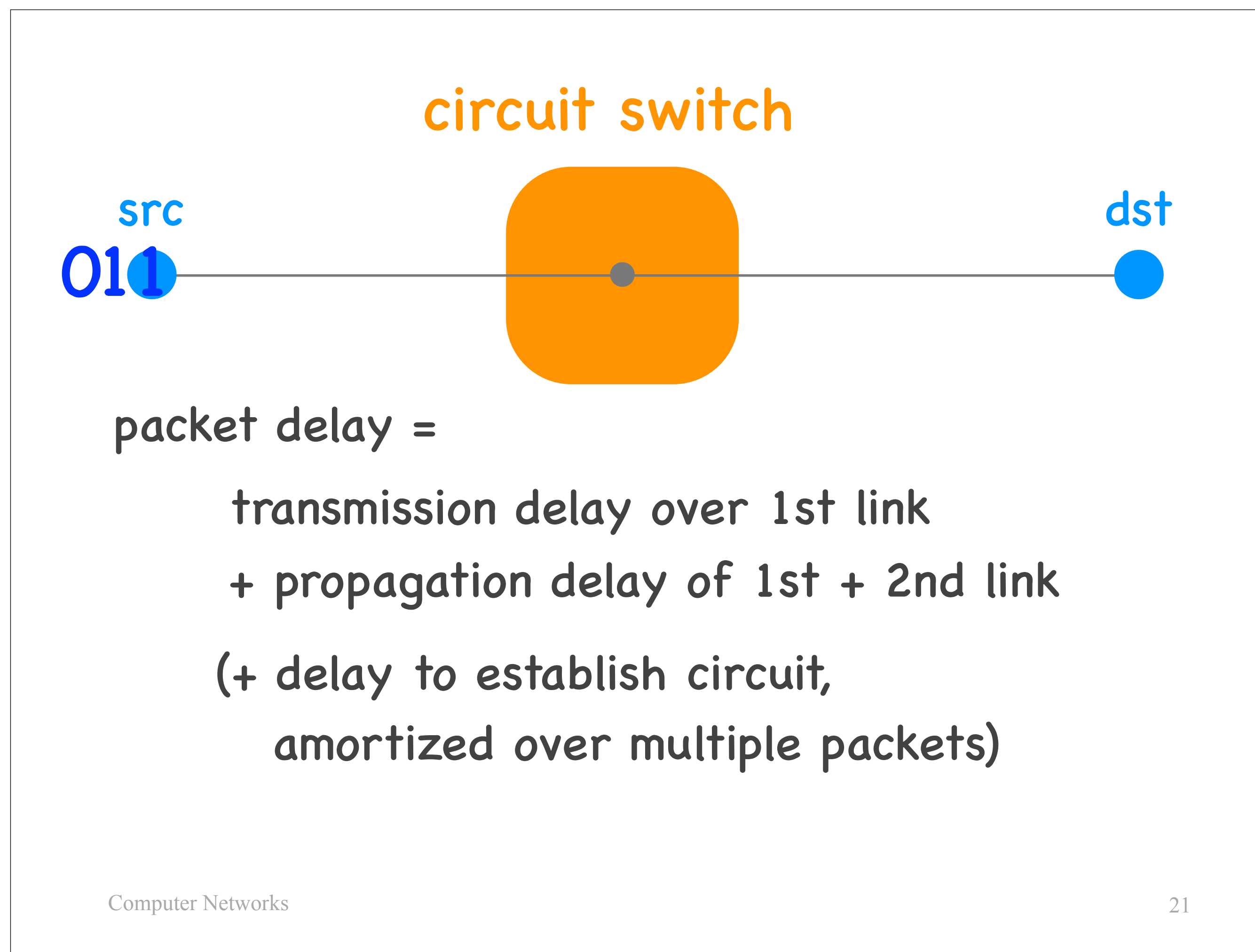
And then it takes a certain amount of time for the last bit of the packet to reach the destination. This is called propagation delay, and it is equal to …

For example, if the link is 1m long and data can travel in it with the speed of light,
then the propagation delay of the link is 3.34 nsec.



So: In the scenario where two end-systems are directly connected through a physical link, the packet delay is equal to the transmission delay of the packet over the link + the propagation delay of the link.

These two components — transmission and propagation delay — are very different from each other:
- Transmission delay depends on the transmission rate of the link and the size of the packet itself.
- Propagation delay depends on the propagation speed of the link and the length of the link, but not the packet.



In the next scenario, we have two end-systems connected through two network links, of the same properties, and one packet switch.

We will first assume that this packet switch is what we call a “circuit switch,” like the devices used in traditional telephone networks. In such networks, when two end-systems want to communicate, before they start exchanging data, a physical circuit is established between them. As a result, as soon as the switch receives a bit from the source, it can immediately transmit it to the destination.

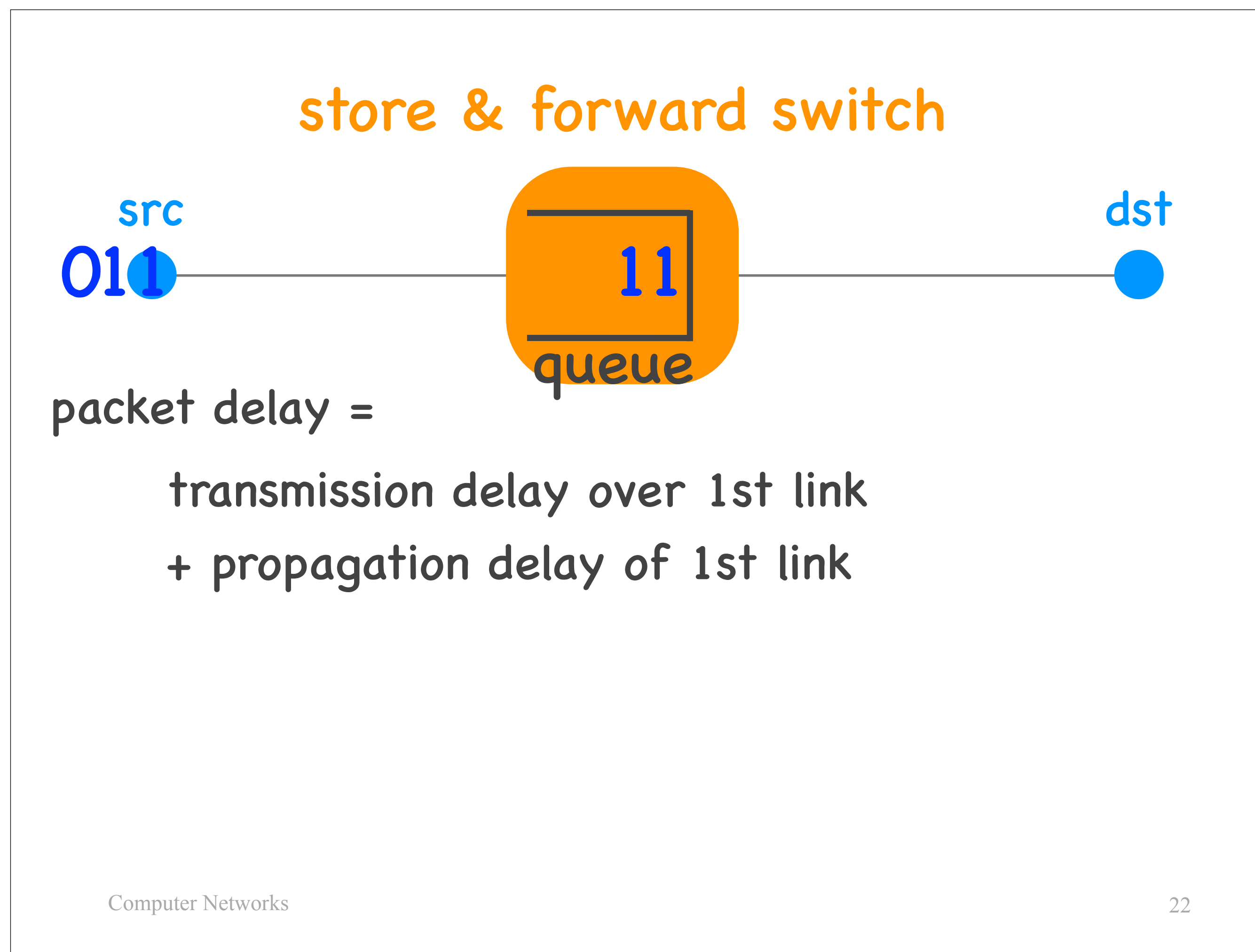
When the source transmits a packet:

- The source first pushes all the bits of the packet onto the first link.
- Then we have to wait for the last bit of the packet to get to the destination.

So: In this scenario, the packet delay is the sum of the transmission delay of the packet over the 1st link + the propagation delay of all the links.

In a way, it’s as if the packet switch is not there at all, it does not introduce any delay.

Except, there is of course the delay to establish the physical circuit before the communication starts. But we may not count that, if it gets amortised over many packets.



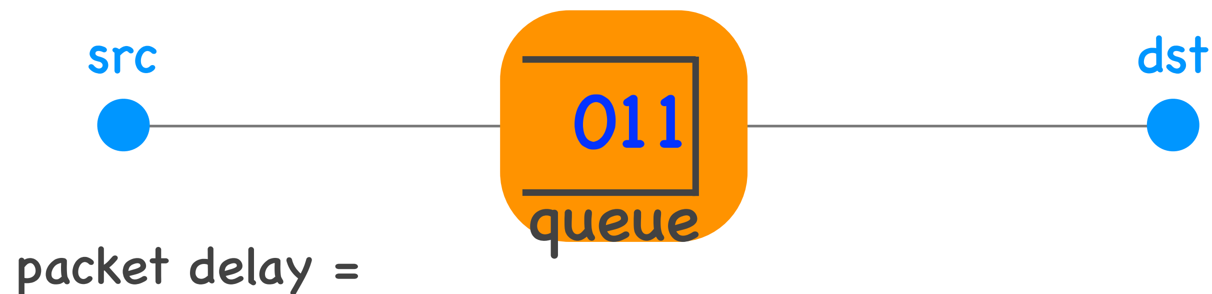
In the next scenario, we have again two end-systems, a source and a designation, connected through a packet switch.

This switch, however, is not a circuit switch, it is what we call a “store & forward” switch, which is much closer to the devices that are used on the Internet today. When the two end-systems want to communicate, there is no need to establish any physical circuit between them, the source starts sending packets to the destination. The switch has a queue, and when it starts receiving bits from a packet, it stores them in the queue until all the bits of the packet arrive. Only then can the switch transmit the packet onto the next link. This is why it is called a store-and-forward switch, because it stores the bits until the last bit of the packet has arrived.

When the source transmits a packet:

- The source first pushes all the bits of the packet onto the first link.
- Then we have to wait for the last bit of the packet to get to the switch.
  When the last bit arrives at the switch, the previous bits of the packet are waiting there.

store & forward switch



packet delay =

transmission delay over 1st link

+ propagation delay of 1st link

+ queuing delay

+ processing delay

+ transmission delay over 2nd link

+ propagation delay of 2nd link

- Then the switch processes the packet and
- pushes all the bits of the packet onto the second link.
- And then we have to wait for the last bit of the packet to get to the destination.

So: In this scenario, the packet delay is equal to …

Because this is a store & forward switch, it does not immediately transmit each bit as it receives it, it waits to receive all the bits of each packet, then processes the packet, and then retransmits all the bits of the packet onto the next link. And this introduces an extra transmission delay component.

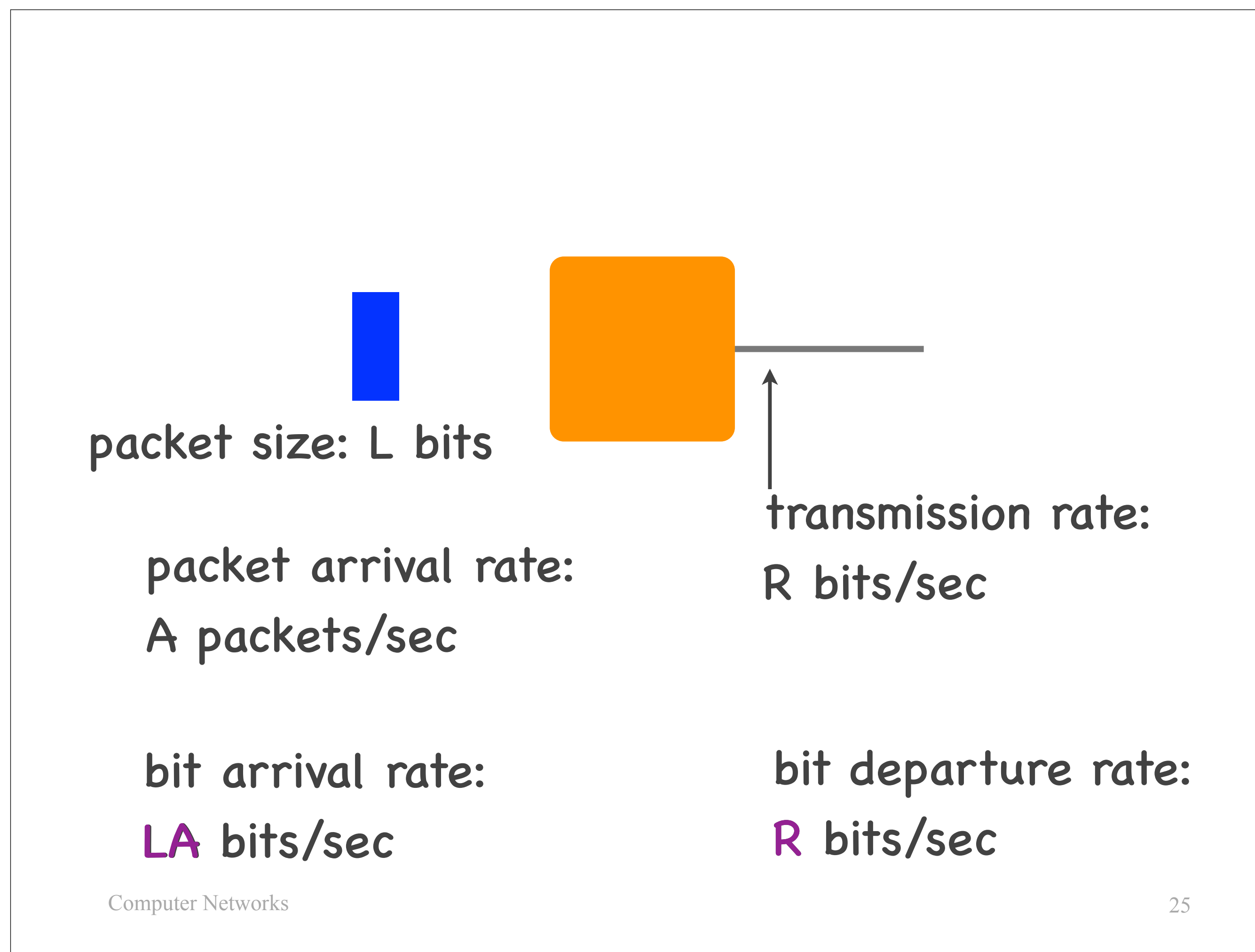
This may seem confusing: why did we not have this extra transmission delay component in the previous scenario? Because there we had a circuit switch: as soon as it receives a bit it transmits the bit, so the first bits of the packet do not need to sit at the switch waiting for the last bit of the packet to arrive.

There is a very important delay component that I skipped: It is possible that when the last bit of the packet arrives at the switch, the switch is still not ready to process the packet, because it is processing other packets that arrived earlier. As a result, the packet must wait in a queue even if all its bits have arrived. So, to our formula we must add the queuing delay that the packet experiences at the switch

Queuing delay

- Given info on traffic pattern
  - arrival rate at the queue
  - nature of arriving traffic (bursty or not?)
- Characterized with statistical measures
  - average queuing delay
  - variance of queuing delay
  - probability that it exceeds a certain value

I gave you precise formulas for the transmission delay of a packet over a link and for the propagation delay of a link.
—> Can I do the same for the queuing delay that a packet experiences at a switch?
No, because it depends on the other traffic arriving at the switch, and we typically don't know in advance what that will be. But, if we know something about the other traffic, e.g., its arrival rate at the queue and how bursty it is, then it is possible to compute statistical measures of the queuing delay, like …

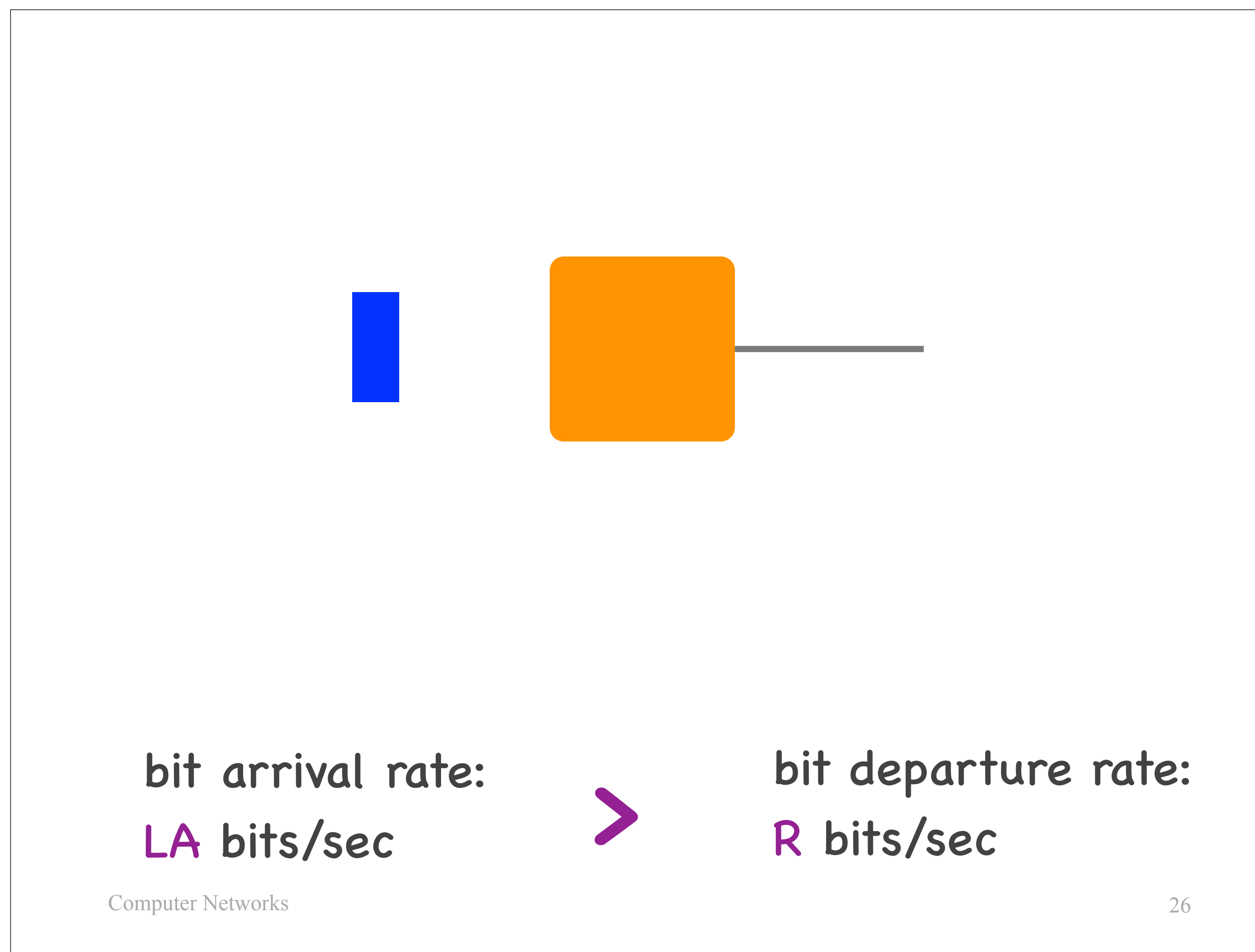


Consider a packet switch, with packets arriving and leaving (it does not matter from where).
Suppose all the packets are of size L bits.
The arrival rate is A packets/sec.
So, the rate at which bits arrive at the switch is La bits/sec.

On the other side, the transmission rate of the outgoing link is R bits/sec.
So, the rate at which bits depart from the switch is R bits/sec.

Let's assume (unrealistically, but for discussion's sake) that the switch has an infinite buffer. (In reality, we do not have infinite buffers, but we sometimes assume that we do to gain insight into the behavior of a system.)

To reason about the queuing delay experienced by packets arriving at the switch,
we must compare the bit arrival rate against the bit departure rate.

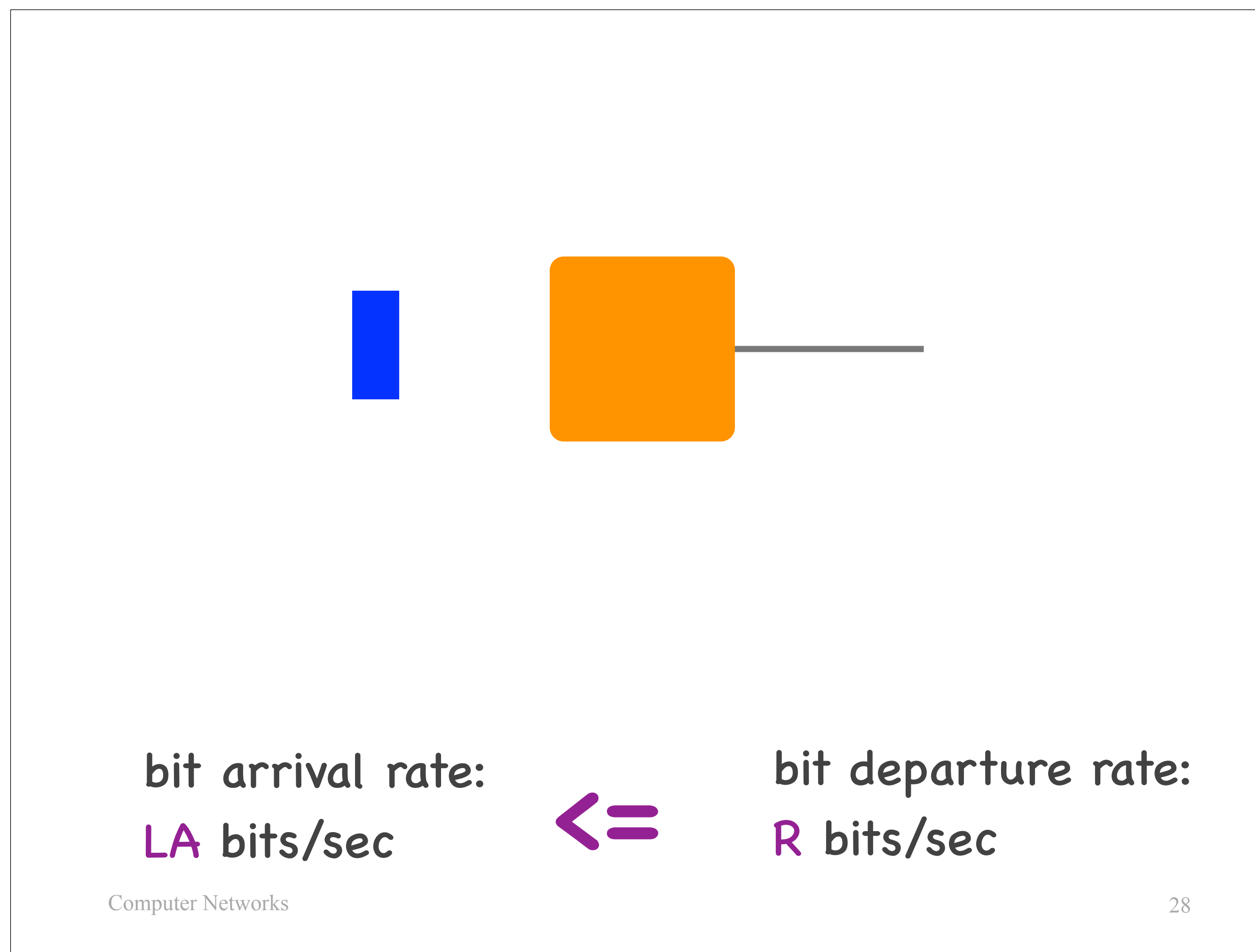


Scenario 1: the arrival rate is higher than the departure rate.

—> How much is the queuing delay?
It goes to infinity as more and more packets are queued up.

Queuing delay

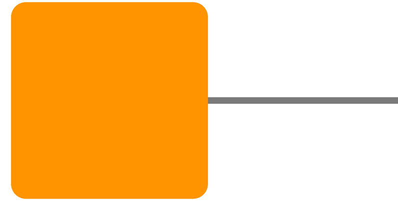
- (Assuming infinite queue)
- Approaches infinity,
  if arrival rate > departure rate



Scenario 2: The arrival rate is smaller than or equal to the departure rate.

—> Is the queuing delay always zero?
No, it depends on the burstiness of the arrivals.

For example, suppose that packets arrive in bursts of 4 at an instantaneous arrival rate that is higher than the departure rate.
After each burst, there are no more arrivals until the buffer has drained.
So, on average, the arrival rate is not higher than the departure rate.
But, because packets arrive instantaneously faster than they can depart, they do experience a certain amount of queuing delay.



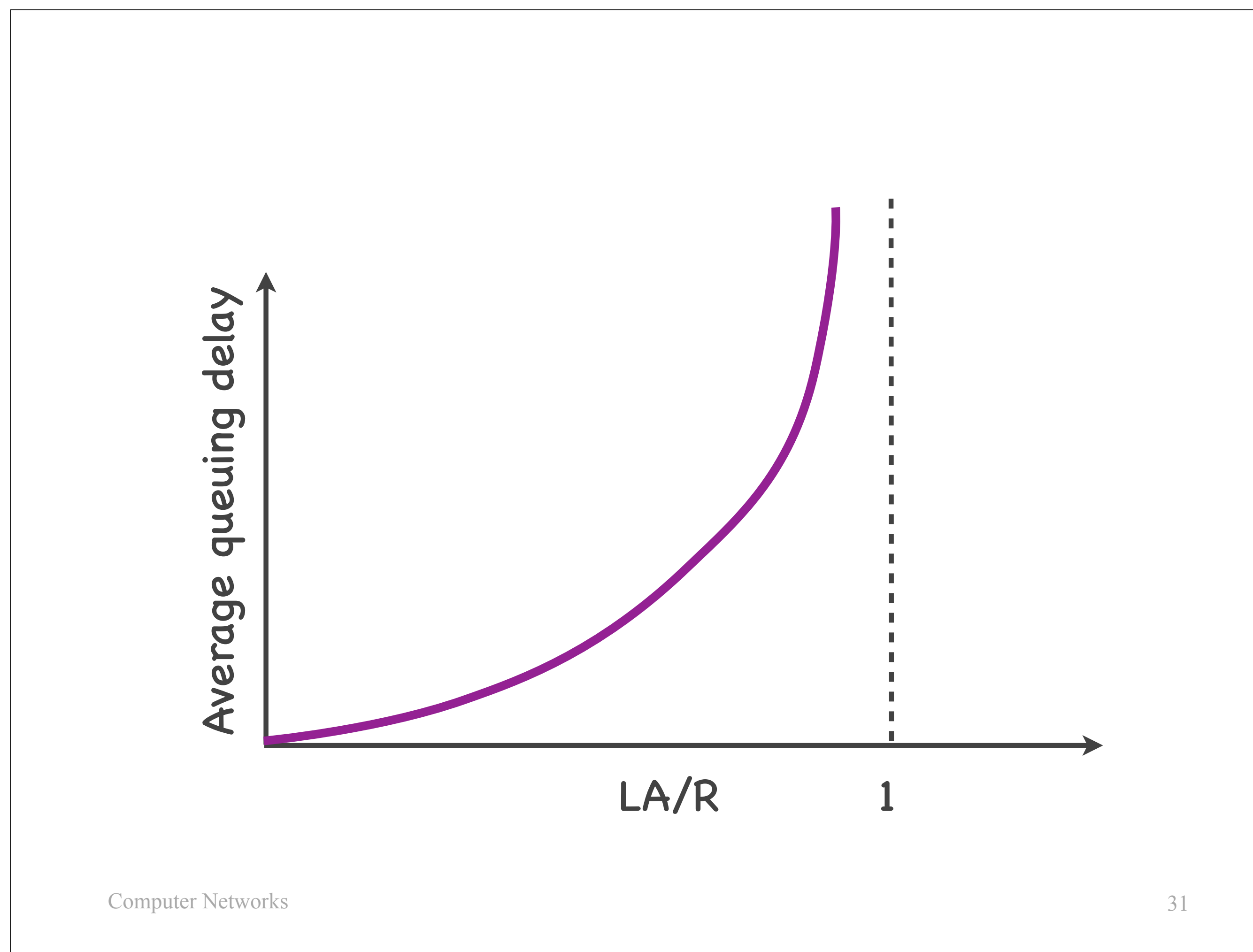
0 usec
1 usec
2 usec
3 usec

bit arrival rate:
LA bits/sec

bit departure rate:
R bits/sec

Queuing delay

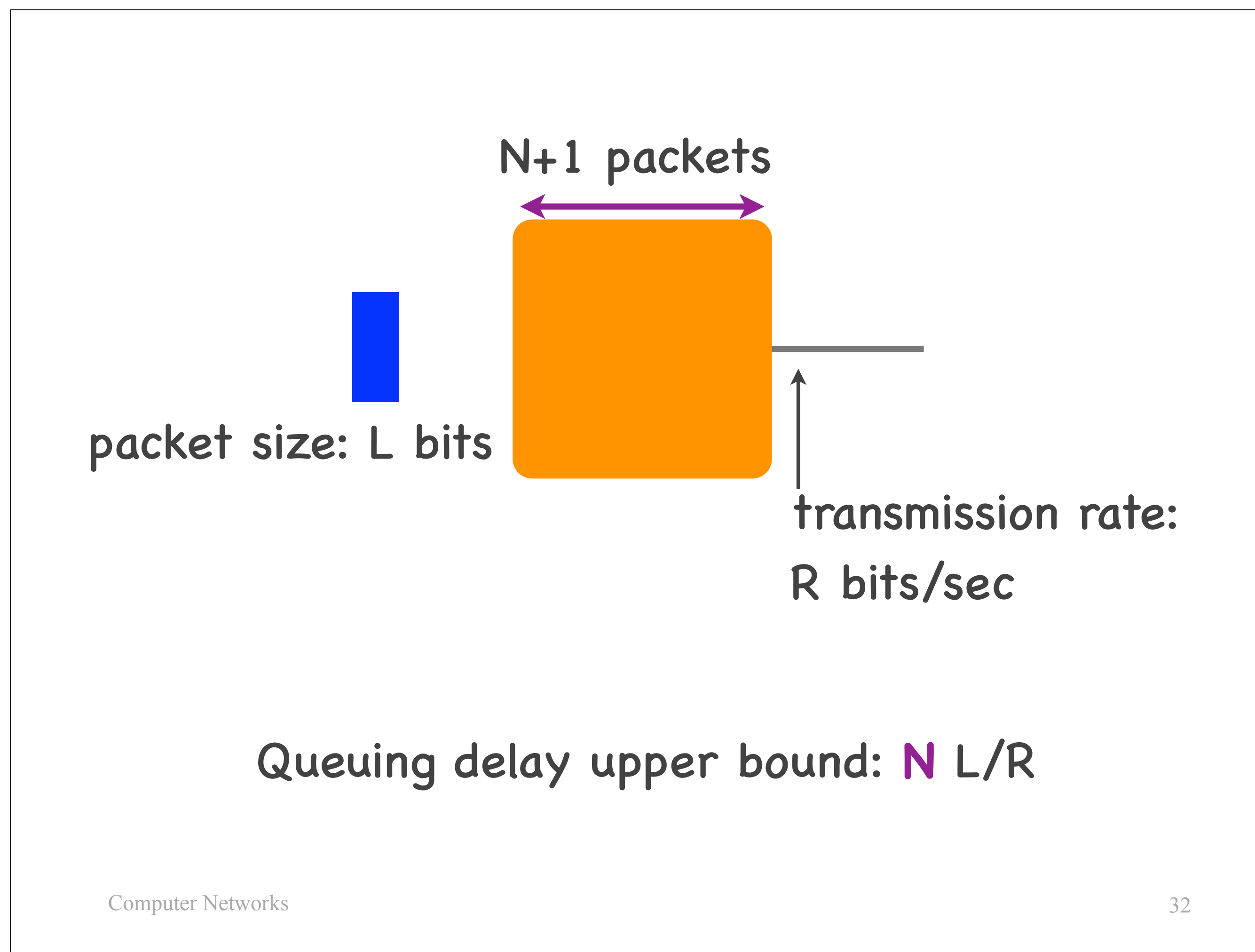
- (Assuming infinite queue)
- Approaches infinity,
  if arrival rate > departure rate
- Depends on burst size, otherwise



This plot shows the average queuing delay experienced by a packet arriving at a queue, as a function of the arrival rate divided by the departure rate.

As LA/R approaches 1 (the arrival rate approaches the departure rate), the average queuing delay goes to infinity.

The important thing about this curve is the shape: there exists a point on the X axis, beyond which the curve goes up very fast.
When a queue operates beyond this point, small changes in arrival rate can have a dramatic impact on queuing delay.
When you design a queuing system, you want it to operate far from this point.



In reality, switches don’t have infinite queues.
When a packet arrives and the queue is full, the switch drops the packet.

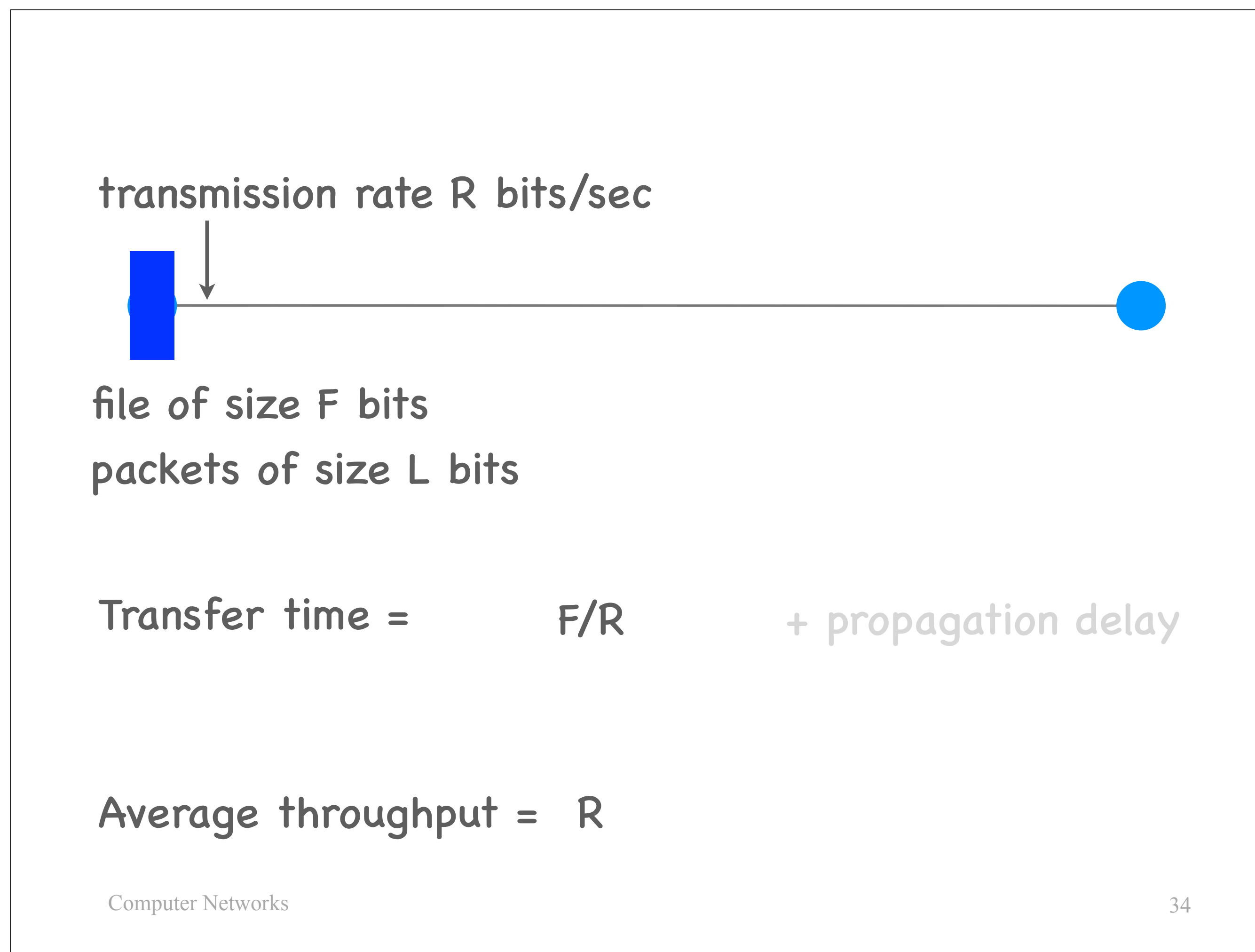
The capacity of the queue imposes an upper bound on the queuing delay that a packet can suffer.
—> If the queue can fit N+1 packets, what is the maximum queuing delay that a packet may experience in this queue?
N times the transmission delay (L/R).
That’s because the worst that can happen to a packets is that it has to wait for N other packets to be transmitted over the outgoing link.

Packet delay

- Many components: transmission, propagation, queuing, processing
- Depends on network topology, link properties, switch operation, queue capacity, other traffic

To summarize:

The packet delay between two end-systems has many components: ...
How many components exactly and how important each of them is depends on the network topology, link properties (transmission rates and propagation delays), the operation of the switches, the capacity of the queues in the switches, and the traffic itself.



Consider again scenario 1 that we saw earlier, where we have two end-systems directly connected through a physical link and suppose this link has rate R bps.

Previously, we computed the delay for one packet of size L bits to get from the source to the destination.
Now we will compute the delay for a file of size F bits to get from the source to the destination.
We will assume that the source cuts the file into multiple packets each of size L bits and sends them back to back to the destination.

Let's compute the transfer time:

- First, the source pushes all the bits of the 1st packet onto the link = transmission delay of the 1st packet.
- Second, the source pushes all the bits of the 2nd packet onto the link = transmission delay of the 2nd packet.
- And so on for all the packets until all the F bits of the file are pushed onto the link.
- Once all the packets are on the link, we have to wait for the last bit of the last packet to get to the destination = propagation delay of the link.

This is very similar to the delay of getting one packet from the source to the destination, just that instead of L bits (the size of one packet), we have F bits (the size of the entire file) in the formula.

In practice, F/R is typically large enough that the propagation delay of the link becomes insignificant and we can ignore it.

Now let's compute the average throughput achieved by this communication: The source sent F bits to the destination and it took approx. F/R time units. Hence, the average throughput is approx. F/(F/R) = R.

transmission rate R

transmission rate $R' > R$

file of size F bits

packets of size L bits

bottleneck link

Transfer time = F/R + propagation delay 1st link + L/R' + propagation delay 2nd link

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

Now consider the same scenario, but let's add a store & forward packet switch in the middle.
Assume that the switch introduces insignificant processing delay and receives no other traffic (except the traffic between the two end-systems).
Assume the second link has transmission rate R' > R.
—> Will the average throughput achieved be higher or lower (or the same)?

Let's compute the transfer time:

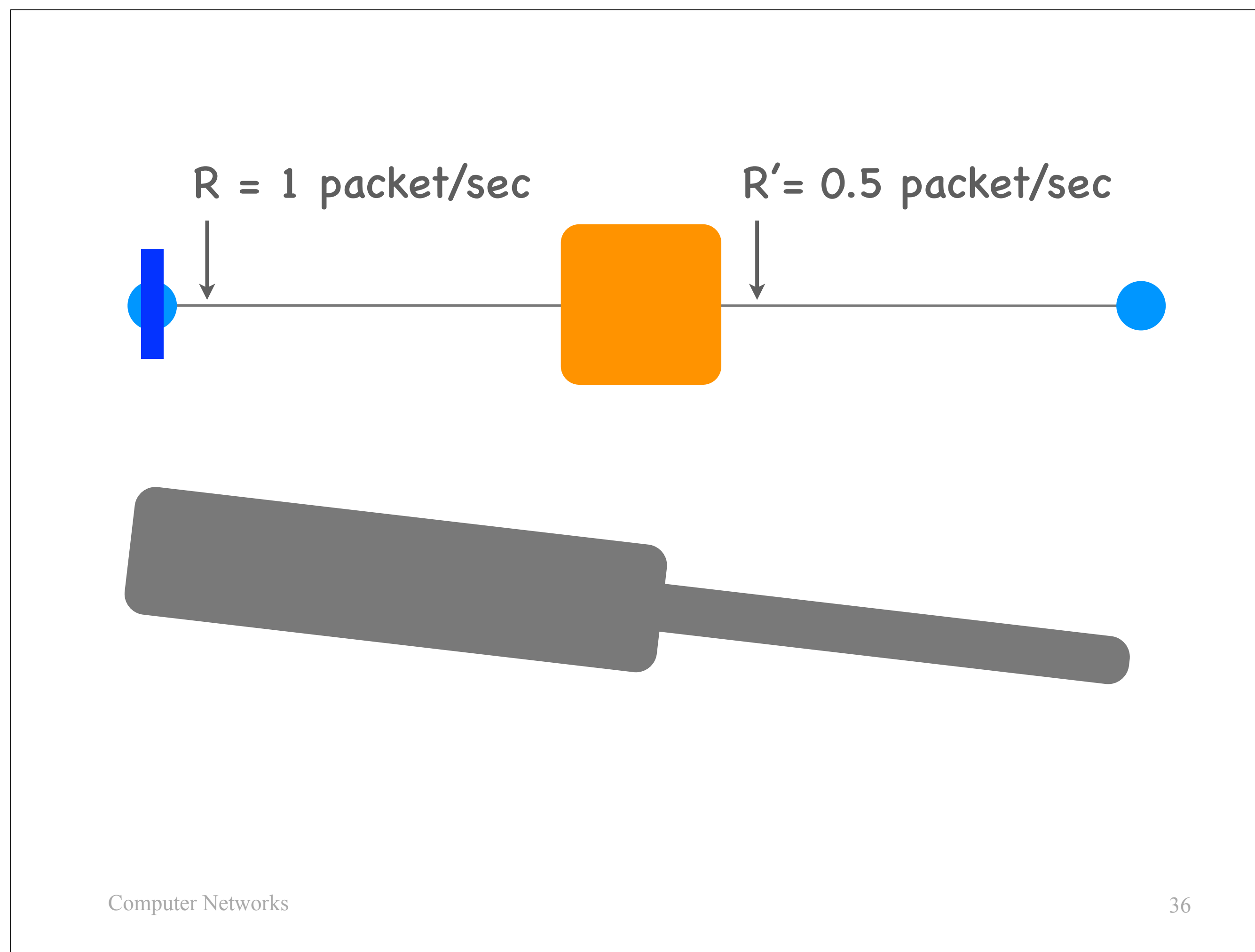
- First, the source pushes all the F bits of all the packets onto the 1st link.
- Second, we have to wait for the last bit of the last packet to get to the switch = propagation delay of 1st link.
- By the time the last bit of the last packet gets to the switch, all the other packets have already left the switch, because the second link has a higher transmission rate than the first link. Hence, the next delay component is the time to push all the L bits of the last packet onto the 2nd link = transmission delay of the last packet onto the 2nd link.
- Finally, we have to wait for the last bit of the last packet to get to the destination = propagation delay of 2nd link.

Again, in practice, F/R is large enough that the remaining components become insignificant and we can ignore them.

Now let's compute the average throughput: The source sent F bits to the destination and it took approx. F/R time units. Hence, the average throughput is approx. F/(F/R) = R.

—> What if we add 100 switches between the source and the destination?
The average throughput will be approx. equal to the transmission rate of the slowest link.

In this scenario, we say that this link is the "bottleneck" link between the two end-systems, because this is where the traffic between the two end-systems flows at the slowest rate.

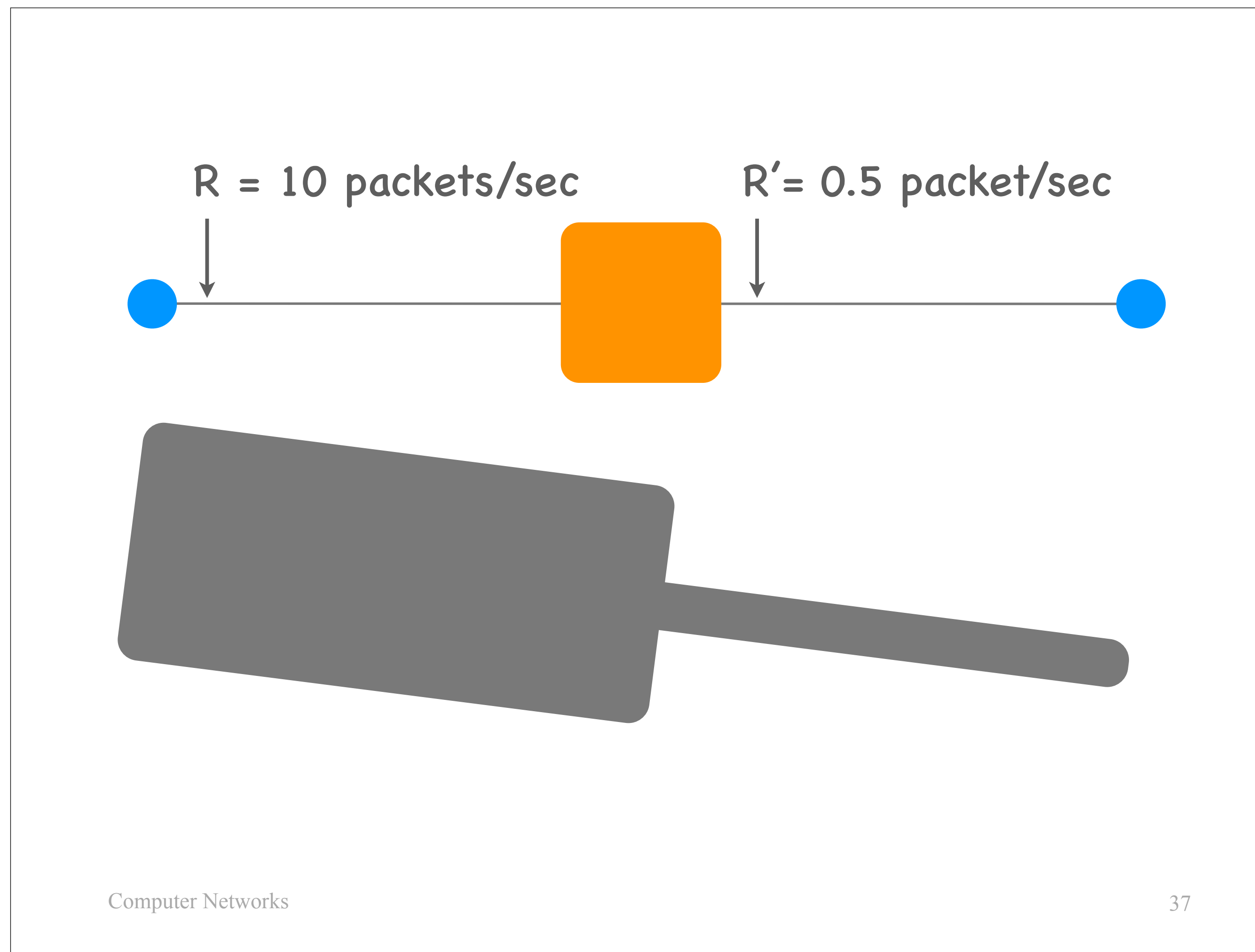


Let’s make the concept of the bottleneck more concrete:

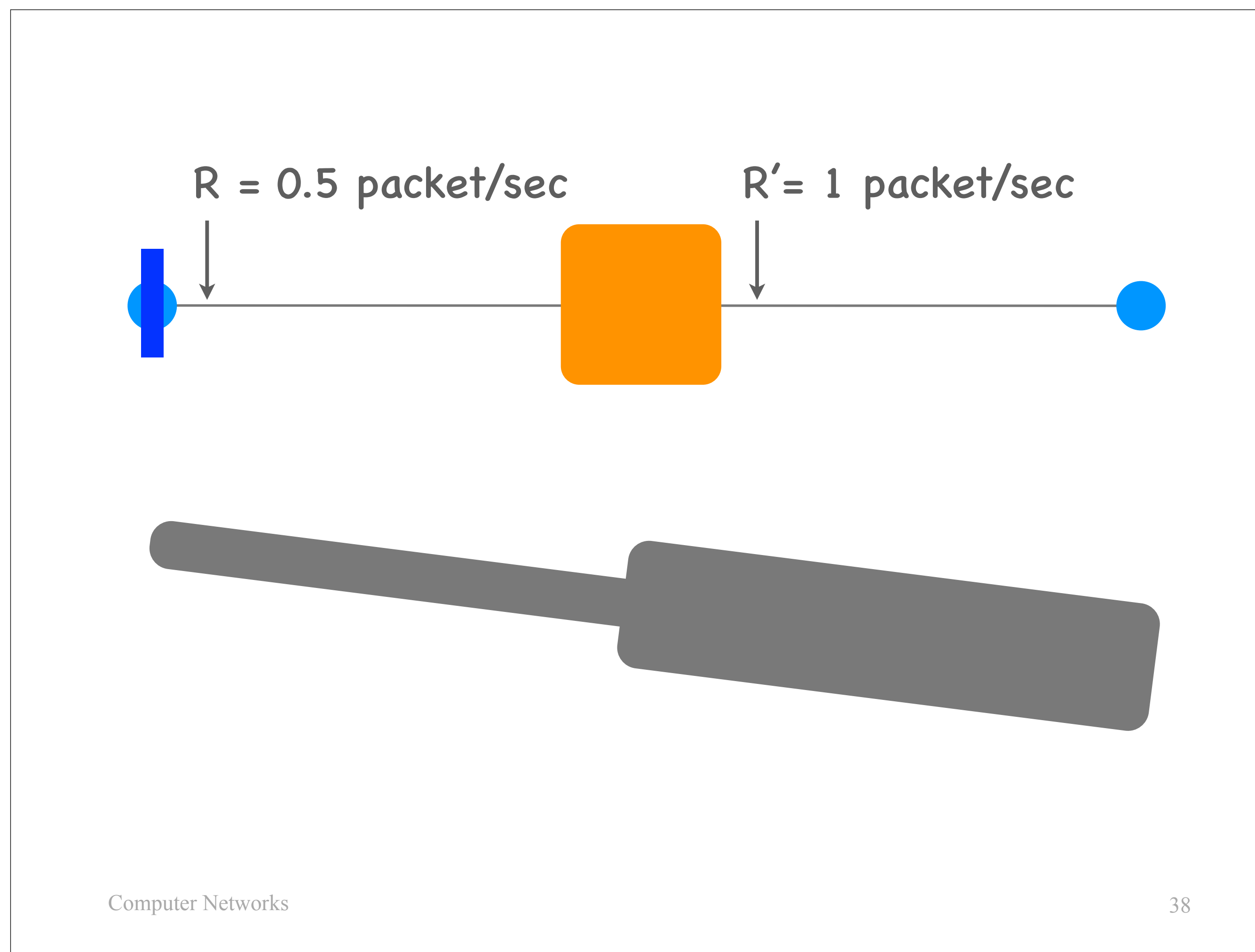
Suppose the first link has twice the transmission rate of the second one. So, if the first one supports 1 packet/sec for a given packet size, the second one supports 0.5 packet/sec.

Suppose the source starts transmitting 1 packet/sec. As the packets arrive at the switch, the switch must space them out, because it can only transmit 0.5 packet/sec on the second link.

It’s as if we have a bottle, and the second link is the bottleneck.
It is the bottleneck that determines how fast liquid flows out of the bottle.



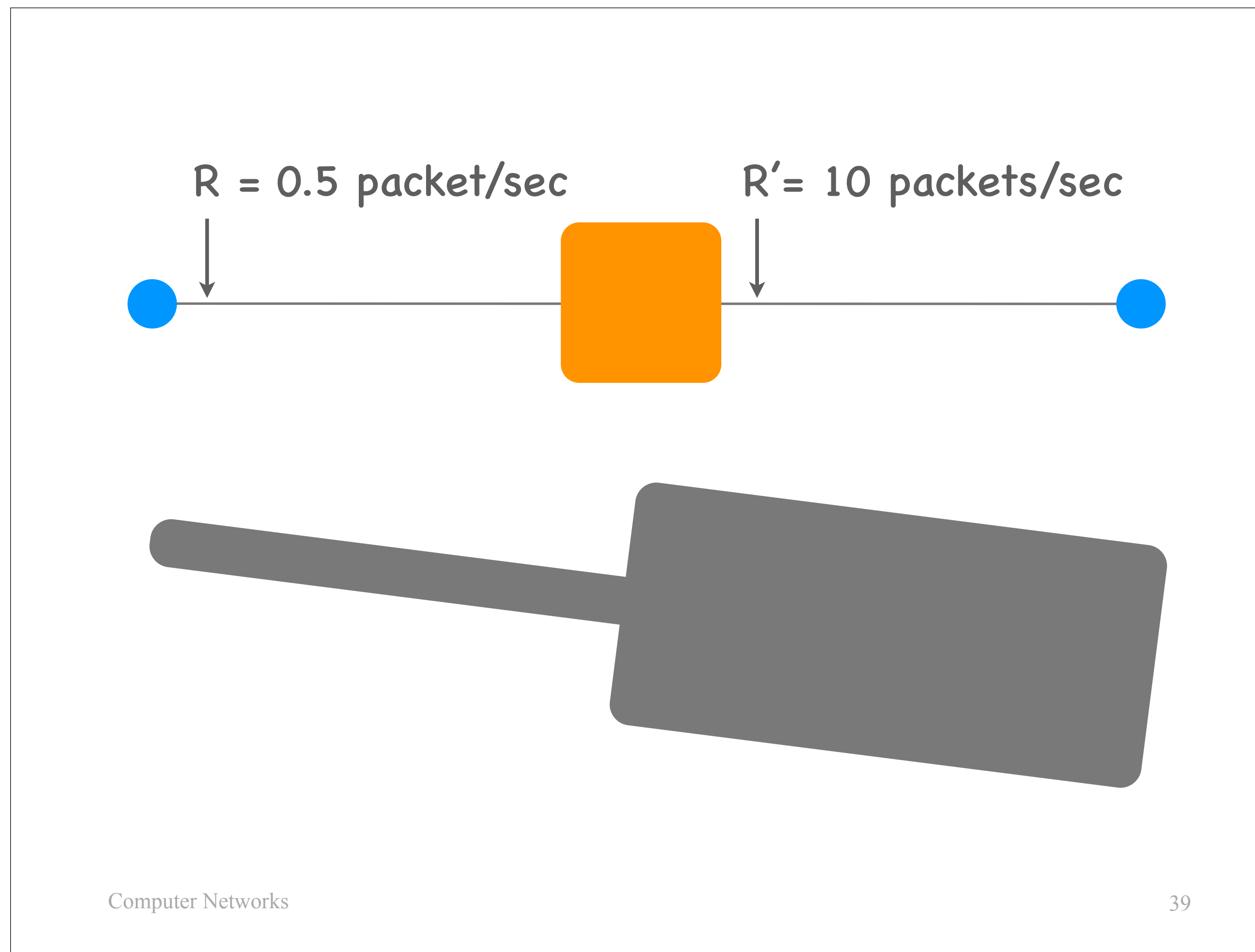
This doesn't change if we double the size of the bottle but leave the bottleneck the same.



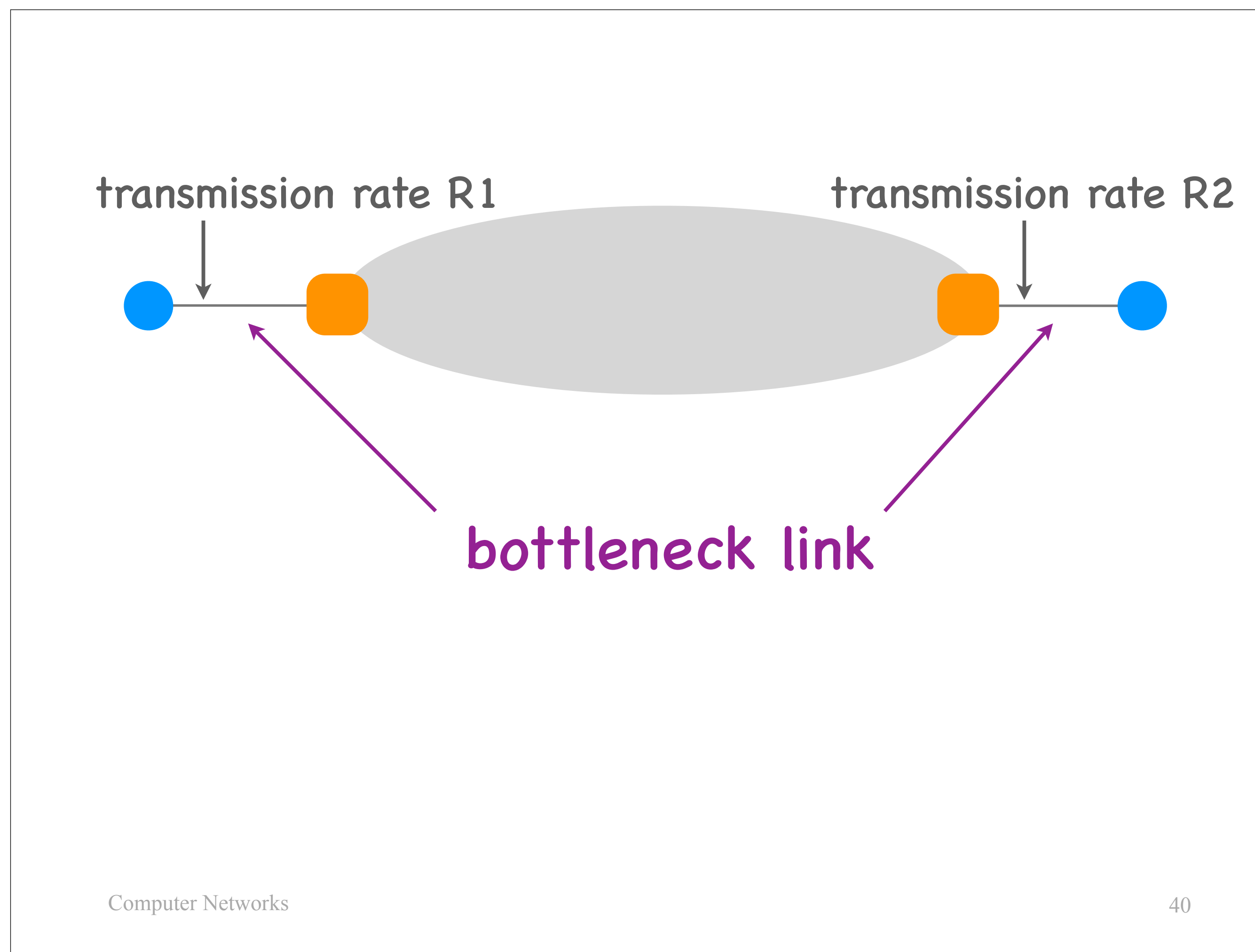
Now suppose the second link has twice the transmission rate of the first link.

Suppose again that the source starts transmitting 0.5 packet/sec. Now as the packets arrive at the switch, the switch does not need to space them out, because it can transmit them immediately on the second link. However, it cannot bring them closer either, the best it can do is maintain the existing spacing.

It's as if we have an upside down bottle, and water is coming in from the bottleneck and coming out from the fat part of the bottle. Still, it is the bottleneck that determines how fast water flows out of the bottle.

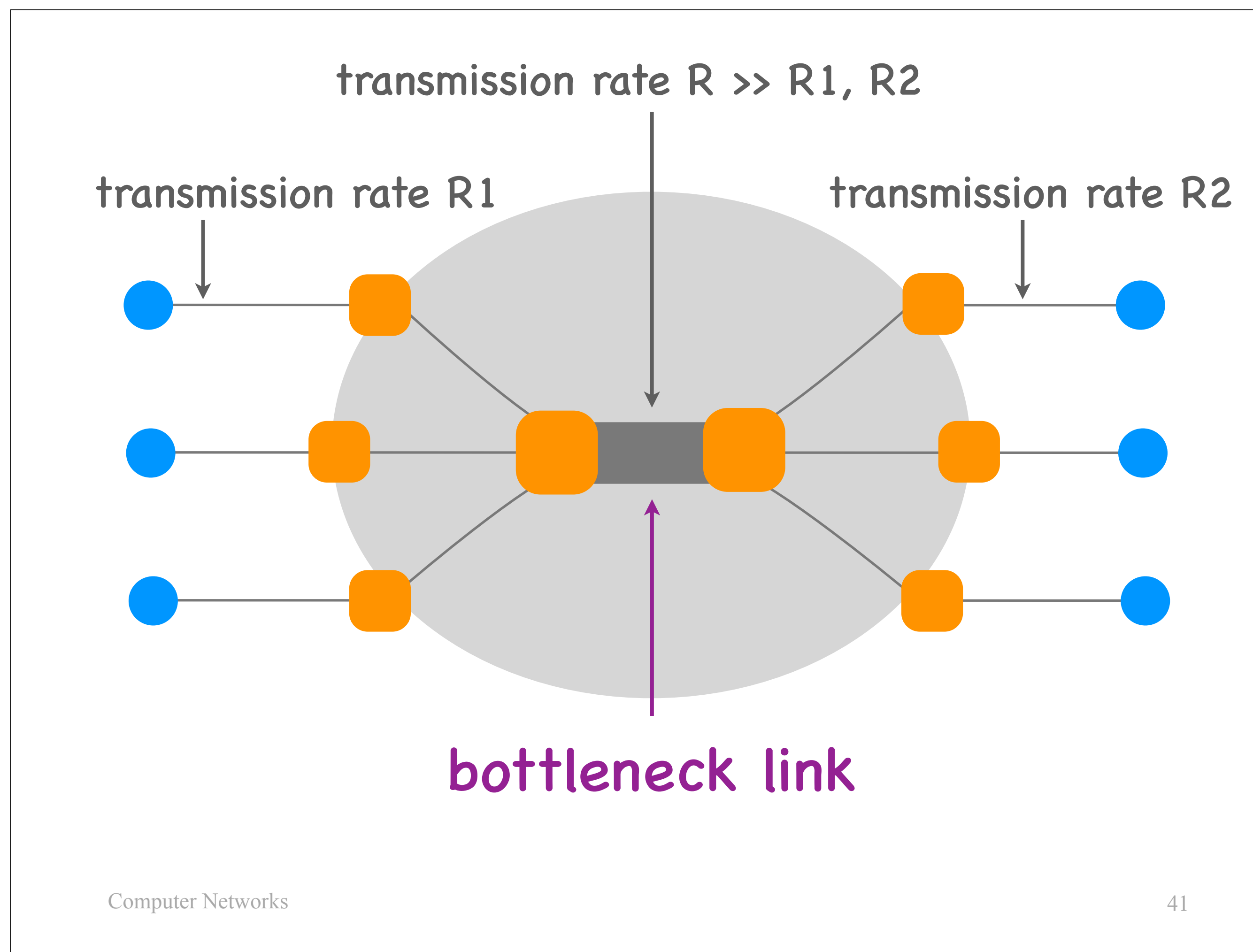


And it doesn’t change if we double the size of the bottle but leave the bottleneck the same.



When two end-systems communicate over the Internet, the bottleneck link is often at the edge of their communication path, i.e., near one of the two end-systems.

However, the bottleneck link is not always the one with the smallest transmission rate.



For example, consider this scenario, where many pairs of communicating end-systems use the same link in the middle of the Internet.
This link has a much bigger transmission rate than any of the links at the edges, however, it receives so much traffic (from all the communicating pairs) that it experiences significant queuing delay.
As a result, it becomes the bottleneck link.

Bottleneck link

- The link where traffic flows at the slowest rate
- Could be because of the link's transmission rate or because of queuing delay

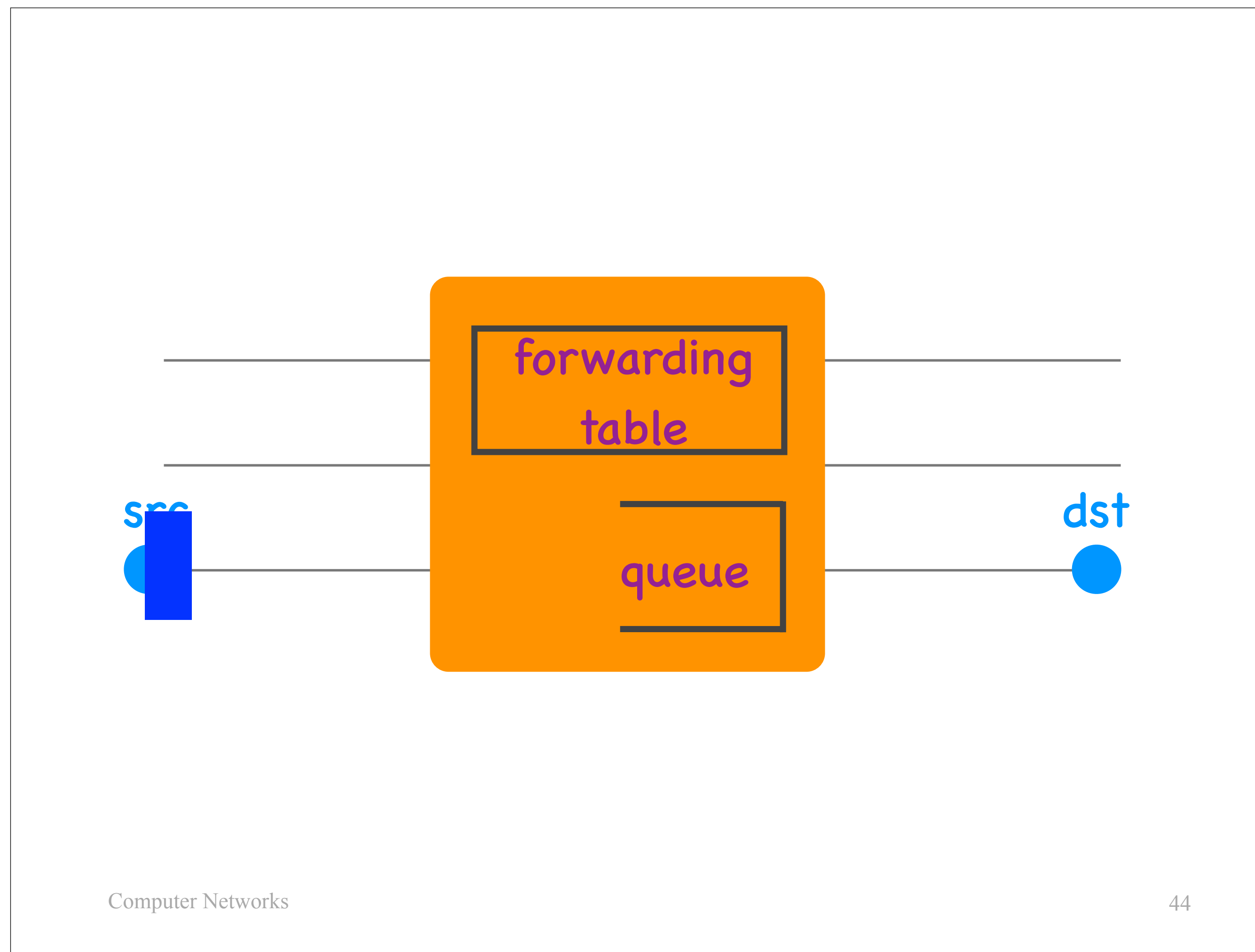
So, the bottleneck link between a pair of communicating end-systems is the link where traffic between the two end-systems flows at the slowest rate.

It could be …

Questions

- What's underneath?
- Who owns what?
- How does it work?
- How does one evaluate it?
- How do end-systems share it?

We are ready to explore the final question of the introduction:
how do billions of end-systems share the Internet infrastructure?

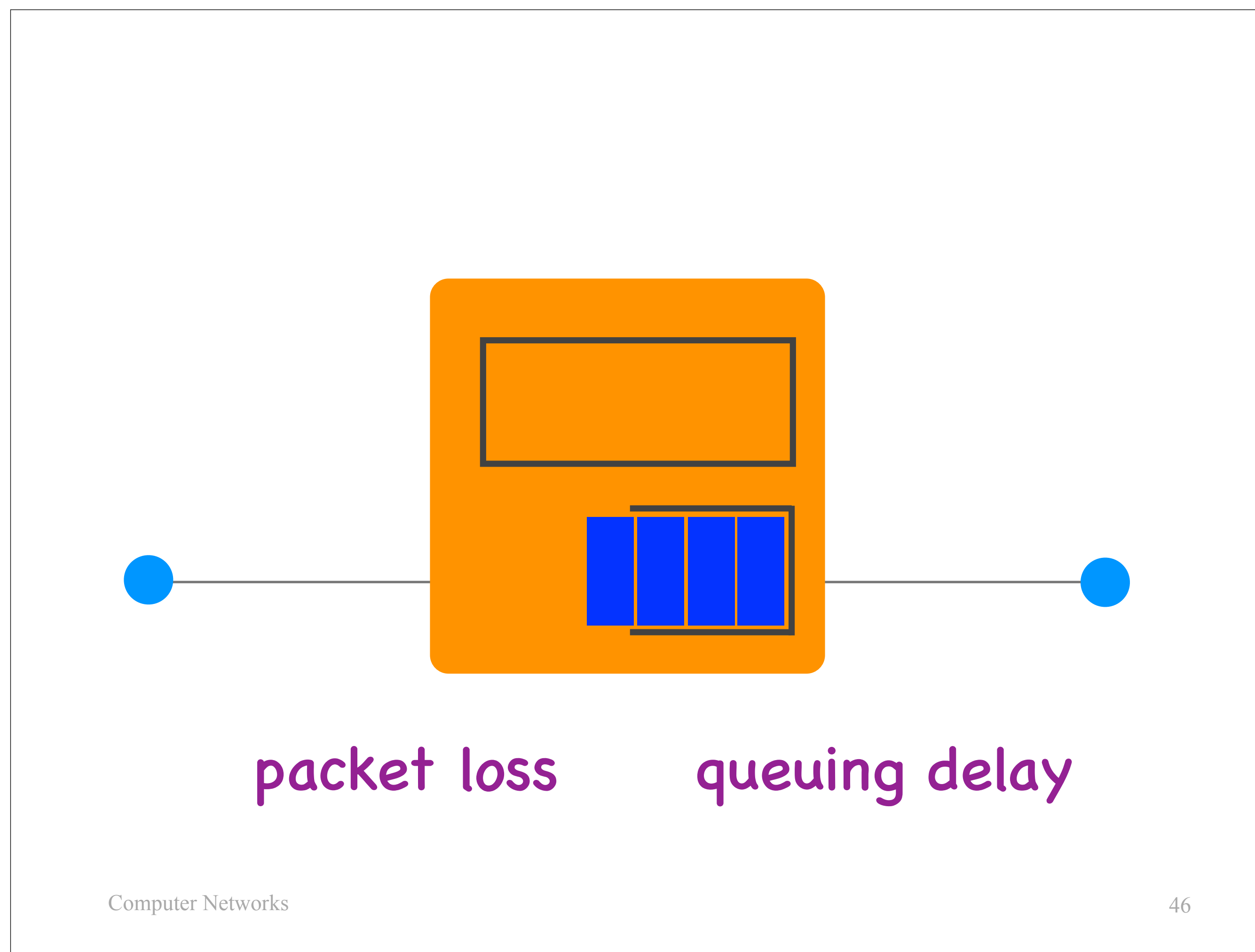


Consider again two end-systems connected through a store-and-forward packet switch.
Suppose that the source sends a packet to the destination.
When the packet arrives at the switch, the switch stores it in a queue (also called a buffer).
Moreover, the switch consults something that is called the “forwarding table” to decide what to do with the packet, where to send it next.

Switch contents

- Queue
  - stores packets
- Forwarding table
  - store meta-data
  - indicate where to send each packet

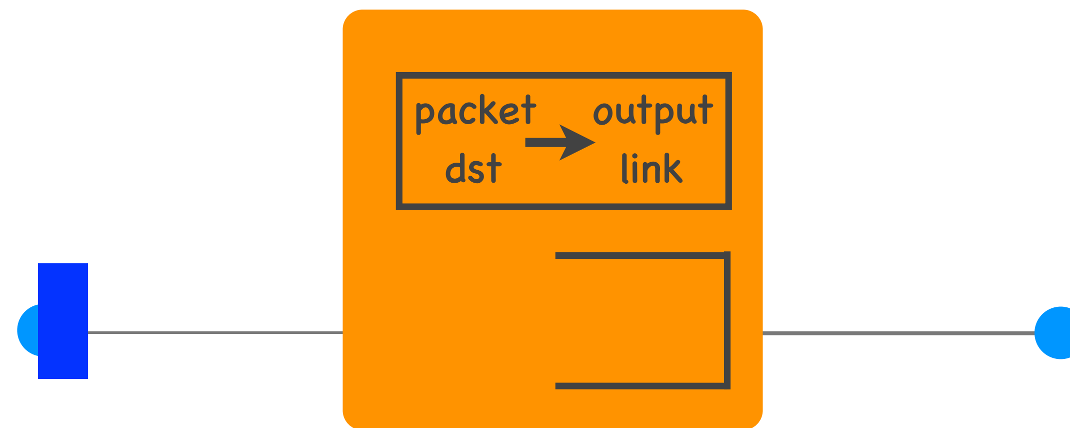
So, inside a switch, we typically have at least a queue and a forwarding table.



As we saw earlier, it is possible that packets arrive at the switch in such a manner that some of them experience queuing delay or they are even dropped.

In other words, the switch does not have enough resources to serve all the packets perfectly, which means that it must do resource management.

Packet switching



Packets treated on demand

One approach to resource management is “packet switching”.

When the source has a packet to send to the destination, it writes the destination on the packet and sends the packet to the switch.

Once the packet arrives at the switch, the switch decides whether it has enough resources (e.g., enough queue space) to store and process this packet.

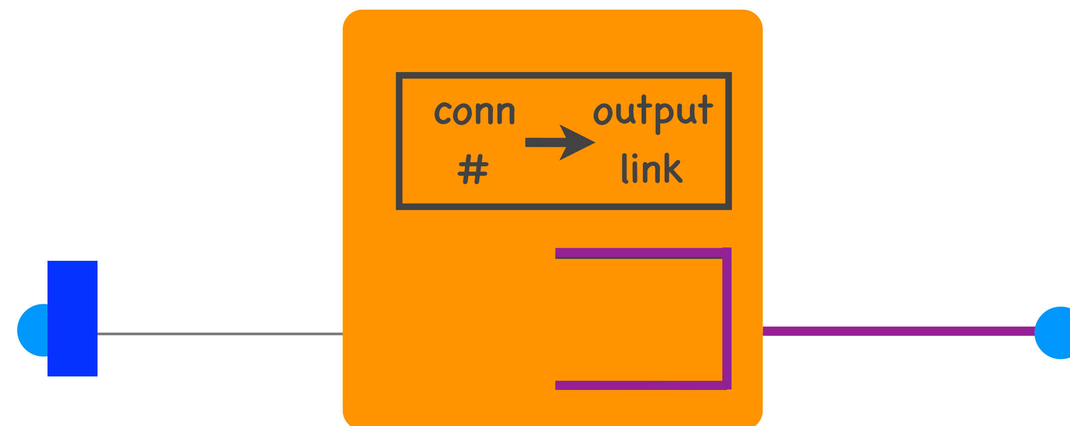
Moreover, the switch maintains, in its forwarding table, a data structure that says where to send each packet based on its destination.

If the switch decides that it does have the resources to forward this packet, it looks up in this data structure where to send it next.

So, with this approach, the switch treats each packet as an entity that is separate from the rest of the traffic.

So, packets are treated on demand.

“Connection switching”



Resources reserved in advance

A different approach to resource management is what I call “connection switching”. I put it in quotes because the term is mine, but I think it captures better than the typically used term what this approach is all about.

When the source has data to send to the destination, it does not just send packets to the switch.
The source first talks to the switch and says “I am establishing a *connection* to that destination, and I will be sending that many bits and packets per second, so, please reserve enough resources to forward all my packets without queuing delays and without packet loss”.

At that point, the switch decides whether it has the necessary resources to store and process all the packets that will belong this connection.
If it does, it *reserves* resources in *advance* for this connection: it determines that some fraction of its queue, some fraction of the transmission rate of the outgoing link will be dedicated to this connection alone.
Moreover, the switch updates its forwarding table and adds information that says what it should do with the packets that belong to this connection (where it should send them).

When all this setup has been done, the source starts sending packets to the switch, after writing on each packet the connection that the packet belongs to.

If everything goes according to plan (there is no technical failure), these packets do not experience any unexpected queuing delay or any packet loss.

This is because the switch has reserved queue space for them, so there are no other packets queued up in there to cause unpredictable delay and loss.

So, with this approach, the switch treats the entire connection as one entity.
Packets are *not* treated on demand, resources are reserved for an entire connection in advance.

Resource management

- Packet switching
  - packets treated on demand
  - admission control & forwarding decision: per packet
- "Connection switching"
  - resources reserved per active connection
  - admission control & forwarding decision: per connection

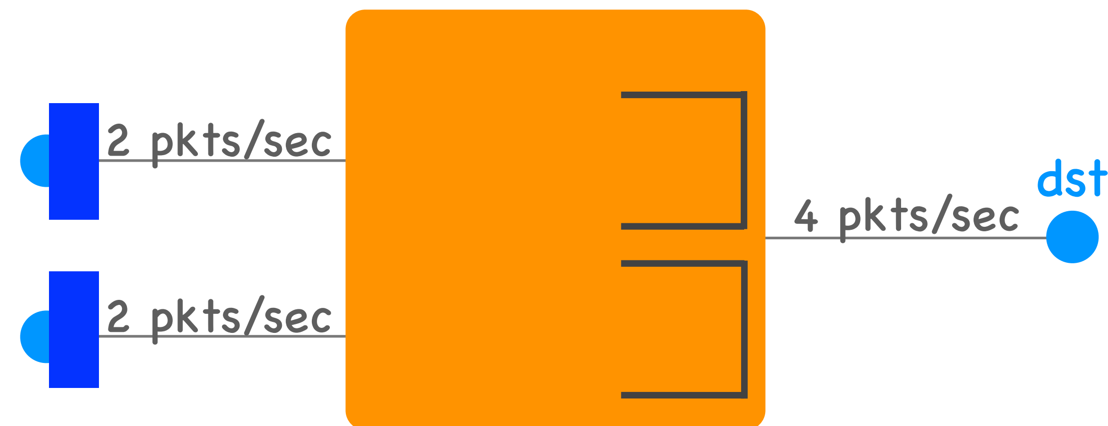
Treat on demand or reserve?

So, two ways for a packet switch to do resource management:

- With packet switching, packets are treated on demand.
  This means that admission control (should I take this packet or not) and forwarding decisions are done per packet.
- With connection switching, resources are reserved in advance for an entire connection.
  This means that admission control and forwarding decisions are done per connection.

The question is which approach is better, packet switching or connection switching?

“Connection switching”



Predictable performance

Let’s look at connection switching first, which is the more sophisticated of the two.

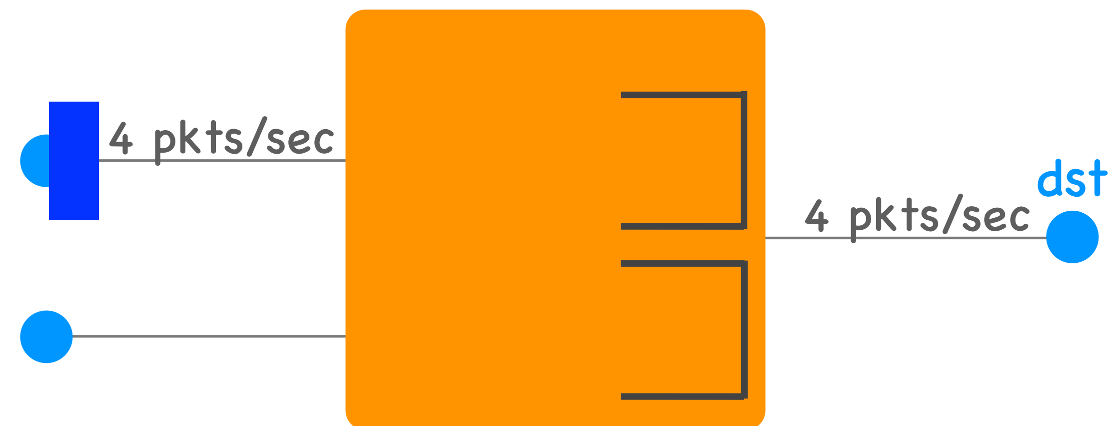
Let’s see what is good about connection switching.

Suppose we have two source end-systems and one destination connected to one packet switch in the middle.
The switch can send at most 4 packets/second to this link.

Now suppose the first source talks to the switch and says “I want to establish a connection to the destination, and I will be sending 2 packets/second”.
And the second source asks the same thing.
The switch says “OK, I can handle 2 + 2 = 4 packets/second. I will accept both requests. I will reserve enough resources to handle 2 packets /second for each connection.”
Indeed, as long as each source sends no more than 2 packets/second (as agreed), these packets experience no unexpected queuing delay and no packet loss.

So, “connection switching” offers predictable performance.
In this particular example, it guarantees to each source that it can send up to 2 packets/second to the destination.

“Connection switching”



Inefficient use of resources

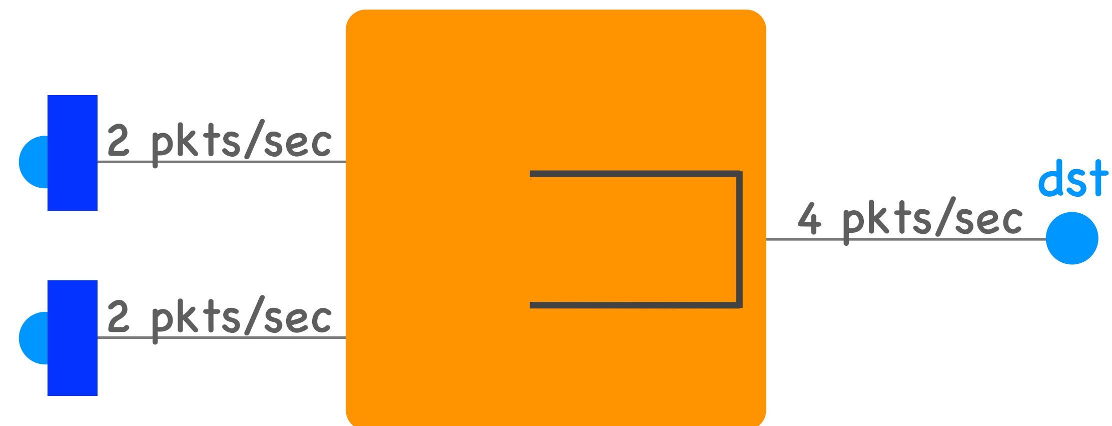
Now let’s see what is bad about connection switching.

We have the same setup as in the previous slide.
However, the second source happens to be silent for some period of time.
Incidentally, during that same period of time, the first source happens to have more than 2 packets/second to send, it sends 4 packets/second.
The switch would normally have the resources to store and process 4 packets/second.
However, because of the way connection switching works (at least the way I have described it so far), it cannot do that.
It has reserved some resources for the second source and it must keep them available in case the second source decides to send.
So, if the first source sends extra packets, they get dropped.

This is similar to what happens when somebody makes a reservation in a restaurant but does not show up.
The restaurant has to turn down customers, even though a table is available.

This is called inefficiency.
When a system has to turn down requests for service, while it has idle (unused) resources, we call that system inefficient.
So, the bad thing about connection switching is that it can be inefficient.

Packet switching

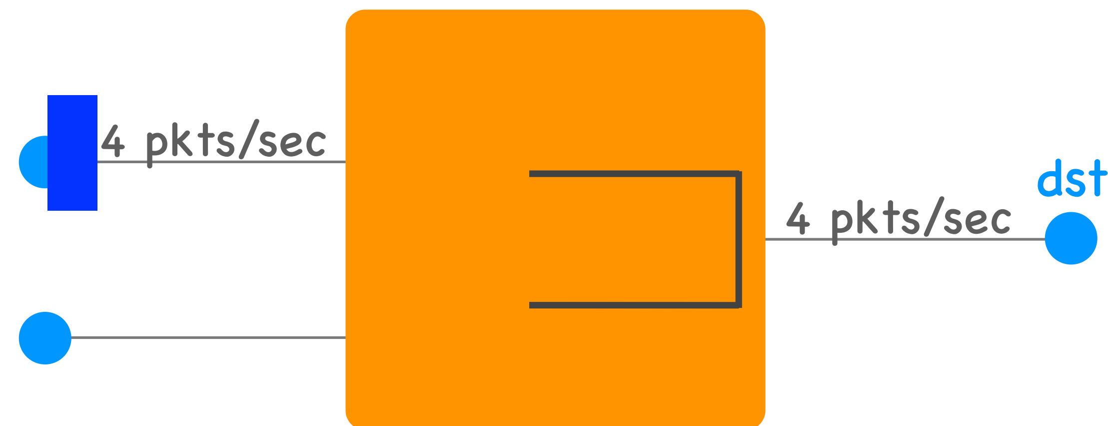


Now let’s turn to packet switching.

There are no reservations, each source just sends whatever packets it has to send, and the switch puts them in a common queue.

First scenario: Each source sends 2 packets/second.
That works fine, since the switch can store and process up to 4 packets/second.

Packet switching



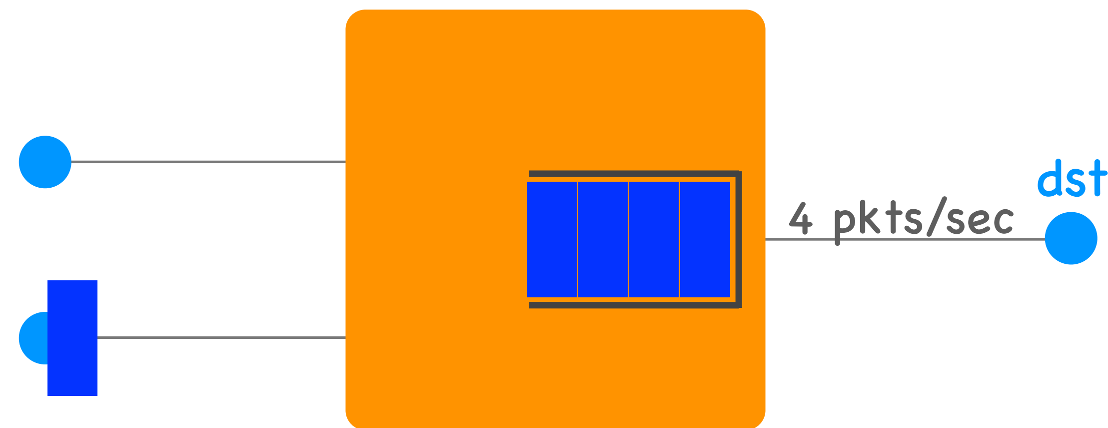
Efficient use of resources

Second scenario: One of the two sources sends 4 packets/second.
That also works fine, because the switch has not pre-allocated a fixed amount of resources for each source.

So, with packet switching, as long as the total amount of traffic going to the destination is up to 4 packets/second, there is no unexpected queuing delay and no packet loss, no matter who generates that traffic.
A packet never gets dropped if the switch has the resources to store and process it.

This is called efficiency.
When a system never turns down requests for service unless all its resources are taken, we call that system efficient.
So, unlike connection switching, packet switching is an efficient approach to resource management.

Packet switching



Unpredictable performance

And what is bad about packet switching?

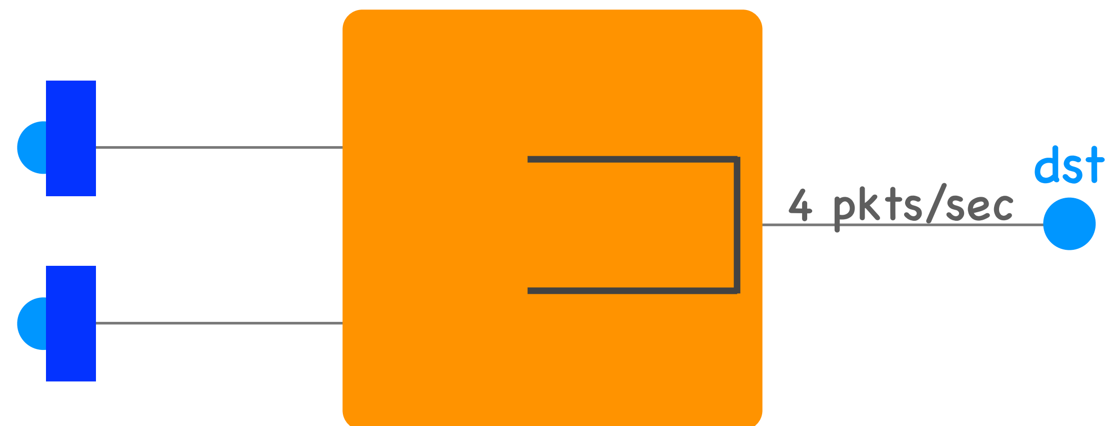
The last scenario we examined is that the second source is silent for a while, and the first source sends 4 packets/second.
Suppose the second source wakes up and also wants to send a packet.
What will happen to that packet?
It will get dropped, because the first source has consumed all of the available resources (all of the available buffer space).

So, what is bad about packet switching is that, unlike connection switching, performance is unpredictable.

—> Which one is easier to implement?
Packet switching, because you don’t need to do anything special at the switches.
To implement connection switching, switches need to be smarter:
they need to be able to decide which connections to accept, how many resources to reserve per connection, and do the actual resource reservation.

However, packet switching does have its complexities as well:

Packet switching



We need congestion control

Suppose that the first source here keeps sending packets at a very high rate and keeps filling the switch's buffer.
As a result, the second source gets to send very few packets, because most of the switch's resources are already taken by the first source.

So: When we use packet switching, we need what is called "congestion control",
which is a mechanism that prevents misbehaving sources from taking up all the resources.

Resource management

- Packet switching
  - efficient resource use
  - no performance guarantees
  - simpler to implement,
    but requires congestion control
- "Connection switching"
  - performance guarantees
  - inefficient resource use

So:

Packet switching enables efficient use of resources,
but leads to unpredictable performance.

Connection switching, on the other hand, can use resources inefficiently,
but offers predictable performance.

Of the two, packet switching is simpler to implement,
but requires an additional mechanism for congestion control.

—> Which approach does the Internet use?
The Internet uses packet switching.
This is why we say that the Internet offers a "best effort" service.
There is no guarantee that your packets will go through.

Normally, when you send traffic, it goes through multiple packet switches, not just one, as in our example.
Each of the switches that receives your packets does the best it can to store and process them, but has not typically reserved any resources for them.

Each user is active w.p. 10%

With 35 users, 10 or fewer users are active w.p. 99.96%

Connection switching: 10 users
Packet switching: about 35 users

We will look at two examples that are also in your book,
which illustrate the benefits of packet switching.

We have a video server connected to a switch with a 10 Gbps link.

On the other side of the switch, we have users who are connected to the switch
and are downloading videos from the server.

The minimum rate at which a user must download in order to experience reasonable performance is 1Gbps.

How many users can this video server serve at the same time?
10 users.

But now I tell you that each user is active only 10% of the time:
10% of the time she is downloading videos and 90% of the time she is watching them.

—> If the switch uses connection switching, how many users can it serve?
With connection switching, when the switch accepts a new client,
it has to reserve resources to handle 1 Gbps.
The link to the video server supports up to 10 Gbps, so how many clients can the switch accept? 10.

—> If the switch uses packet switching, how many users can it serve?
We need to do a bit of math on the side.

If we have 35 users, the probability that 10 or fewer users are active is 99.96%.
So, if the switch accepts 35 users, it will serve them well for 99.96% of the time.

The point is:
With packet switching, the switch can serve more than three times the number of clients than with connection switching, while offering *almost* the same performance, because it is more efficient.

This is what we call…

Statistical multiplexing

- Many users share the same resource
- Not all of them can share it at the same time…
- but we do not expect them to be all active at the same time

Statistical multiplexing…

Only 1 user active
Downloading a 10 Gbit video file

Connection switching: 10 seconds
Packet switching: 1 second

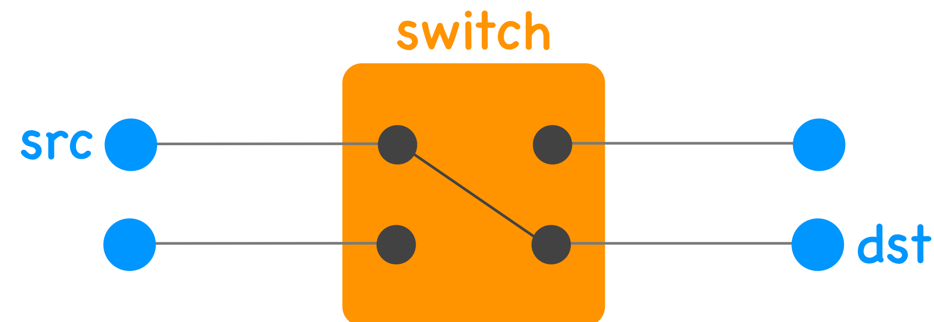
Let’s look at another example.

We have a similar setup as before, with the following difference:
There are exactly 10 users that could be using the system, and only one of them happens to be active at the moment.
She is downloading a video file of 10 Gb.

If the switch uses connection switching and has dedicated 1 Gbps to each client, it will take the client 10 seconds to download the file.
If the switch uses packet switching, since all the other clients are inactive, the client will be able to use all the available resources, and it will take her only 1 second to download the same file.

This is a perfect example to illustrate what efficient use of resources means:
If there’s one active user, she can take up all the available resources to improve the performance she experiences.

Circuit switching



Connection switching
through physical circuits

A term that you will hear often in the networking and communications world is “circuit switching”. What is that?

Not all switches are designed the way I showed you before (with a buffer and a forwarding table).

Some switches (usually of older technology) are designed like this:

When two end-systems that are connected to the switch want to talk to each other, the switch creates an actual physical connection (it draws a wire) between the two end-systems.

You can view this as just one way of implementing what I called “connection switching”.

What did we say about connection switching?

That the switch performs admission control and makes forwarding decisions and reserves resources for an entire connection, not for individual packets.

This is exactly what is happening here.

When the switch accepts a connection from the source to the destination, it draws that wire between them, which automatically reserves a fixed amount of resources to the connection and determines where *all* the data that belongs to this connection will be forwarded.

The term “circuit switching” is often used to refer to “connection switching” in general, even if there are no actual physical circuits involved.

Many kinds of “circuits”

- Physical circuits
  - separate sequence of physical links per connection
- Virtual circuits
  - manage resources as if there was a separate sequence of physical links per connection

There are many different kinds of “circuits”.

There are physical circuits, where, for each connection, we reserve a separate sequence of physical links.

There are virtual circuits, where, for each connection, we reserve resources (e.g., queue space), and we make it appear to the users as if there was a separate sequence of physical links.

Many kinds of "circuits"

- Time division multiplexing
  - divide time in time slots
  - separate time slot per connection
- Frequency division multiplexing
  - divide frequency spectrum in frequency bands
  - separate frequency band per connection

There is "time division multiplexing", where we use a single physical circuit, but divide time in slots.
Each connection is assigned a separate time slot.
For example, a classroom is used for many courses, but each course uses it during a different time slot.
Each class has the illusion that they have their own personal classroom, but they don't.

There is "frequency division multiplexing", where again we use a single physical circuit, but divide its bandwidth in frequency bands.
Each connection is assigned a separate frequency band.
For example, the air is used by many radio stations, but each station uses a different frequency band.

Many kinds of “circuits”

- Different ways to implement “connection switching”
- Create the illusion of a separate physical circuit per connection

These are all different ways to implement the idea of connection switching,
i.e., to create the illusion of a separate physical circuit per connection.

I am not saying that they are all the same.
Each one faces its own technical challenges and, as engineers, you may have to learn some of these details depending on your specialization.
But for now, at a high level, you can think of them as different ways to make admission control and forwarding decisions on a per-connection basis.

Treat on demand or take reservations?

Another key question to ask when designing a network is whether to treat users and their traffic on demand or take reservations. How can one make such a decision?

Let's forget about computer networks for a moment.
Let's say you open a restaurant, and you need to decide whether to accept reservations or not.
If you accept reservations, whenever a customer reserves, you have to hold the table until the customer shows up.
If you don't accept reservations, when there are more customers than available tables, some have to wait, and some go away.

Assuming customers keep coming, which approach will keep your restaurant full?
Not accepting reservations.

Which approach will make you more money?
It depends.
As long as customers keep coming, if you don't accept reservations, your restaurant is always full.
But then, customers have to wait or to go away, which makes them unhappy, so they may not come back.

So, there is a trade-off between: not accepting reservations and keeping your restaurant full, versus accepting reservations and keeping your customers happy.

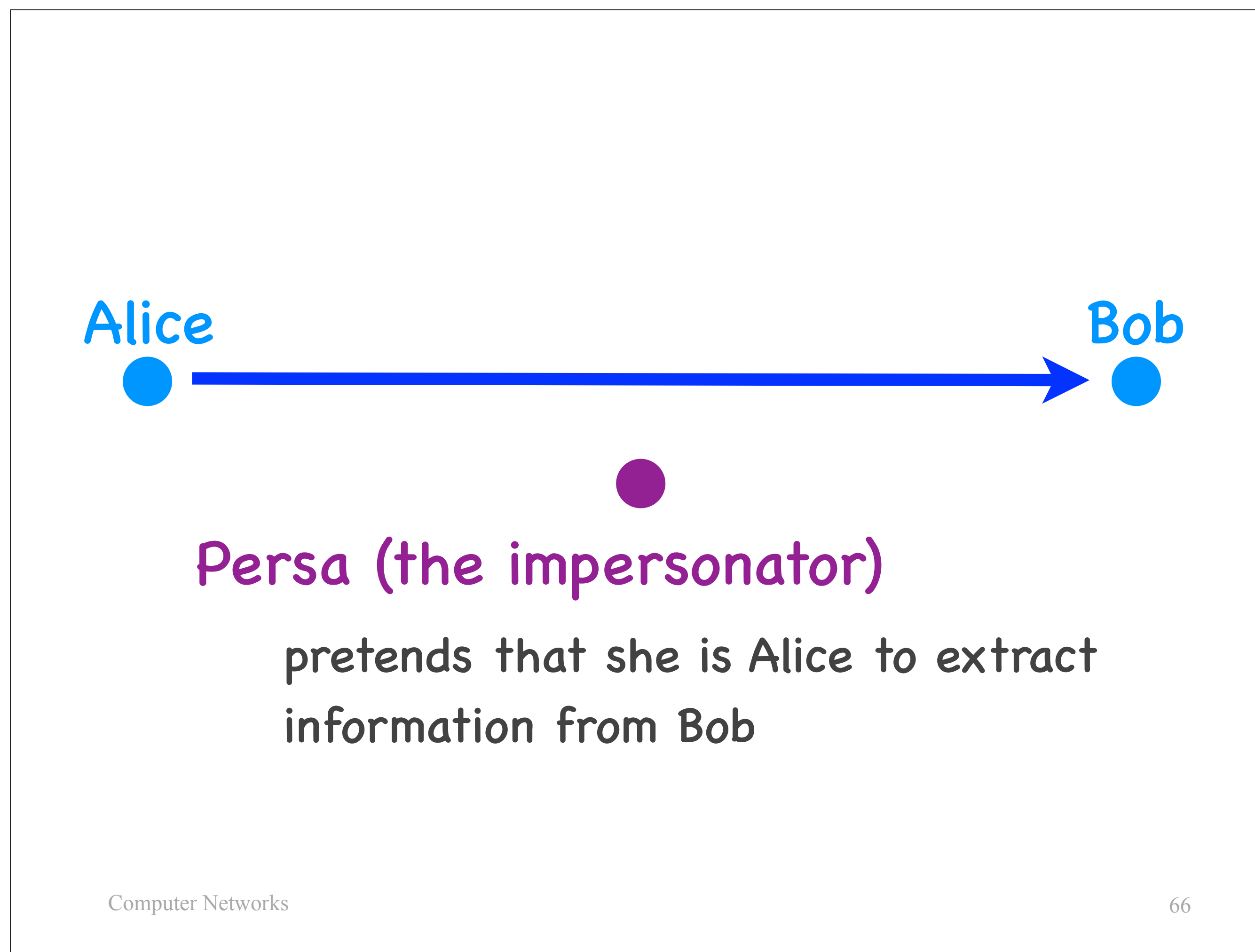
Back to computer networks:
To decide whether to use packet or connection switching, you need to balance the cost and complexity of the infrastructure you are willing to tolerate versus the quality of service that you want to offer to the users.

Eve (the eavesdropper)

tries to listen in on the communication,
i.e., obtain copies of the data

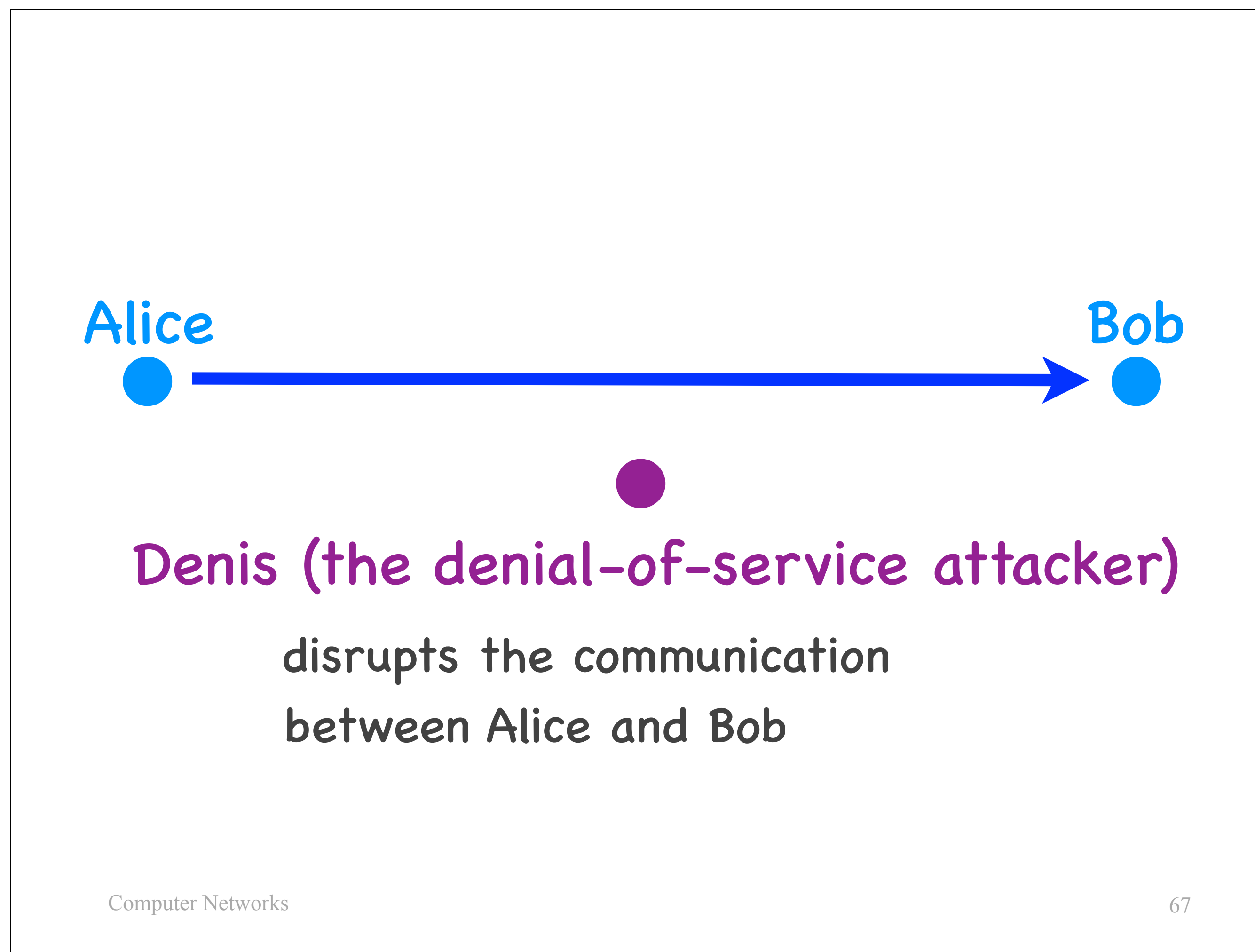
When multiple users share a network, there is typically opportunity for misbehavior.
So, we will close this lecture (and our introduction) with a brief discussion on Internet misbehavior.

One type of misbehaving network user is Eve, the eavesdropper: she tries to listen in on the communication between Alice and Bob and copy the data they exchange.



Another type is Persa, thee impersonator: she pretends that she is Alice to extract information from Bob.

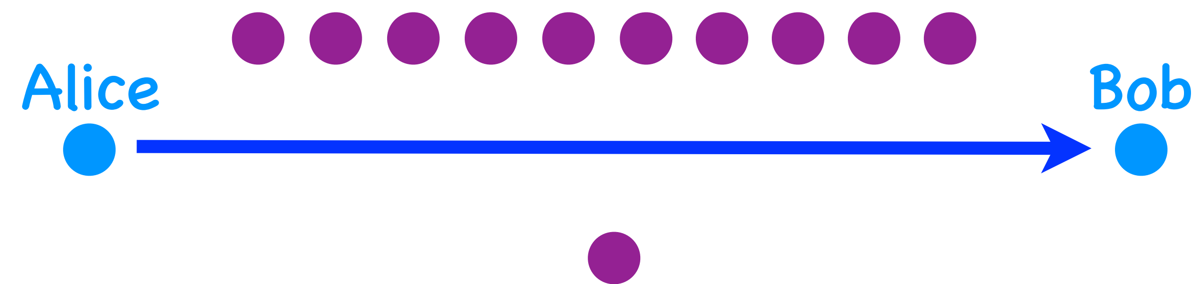
Unlike Eve, Persa is an active misbehaving user, she does not only listen in, she sends traffic.



A third type is Denis, the denial-of-service attacker, who disrupts the communication between Alice and Bob.

One way to do this is to send a lot of junk traffic to Bob and consume all of Bob's resources, so that there are no resources left to receive and process Alice's traffic.

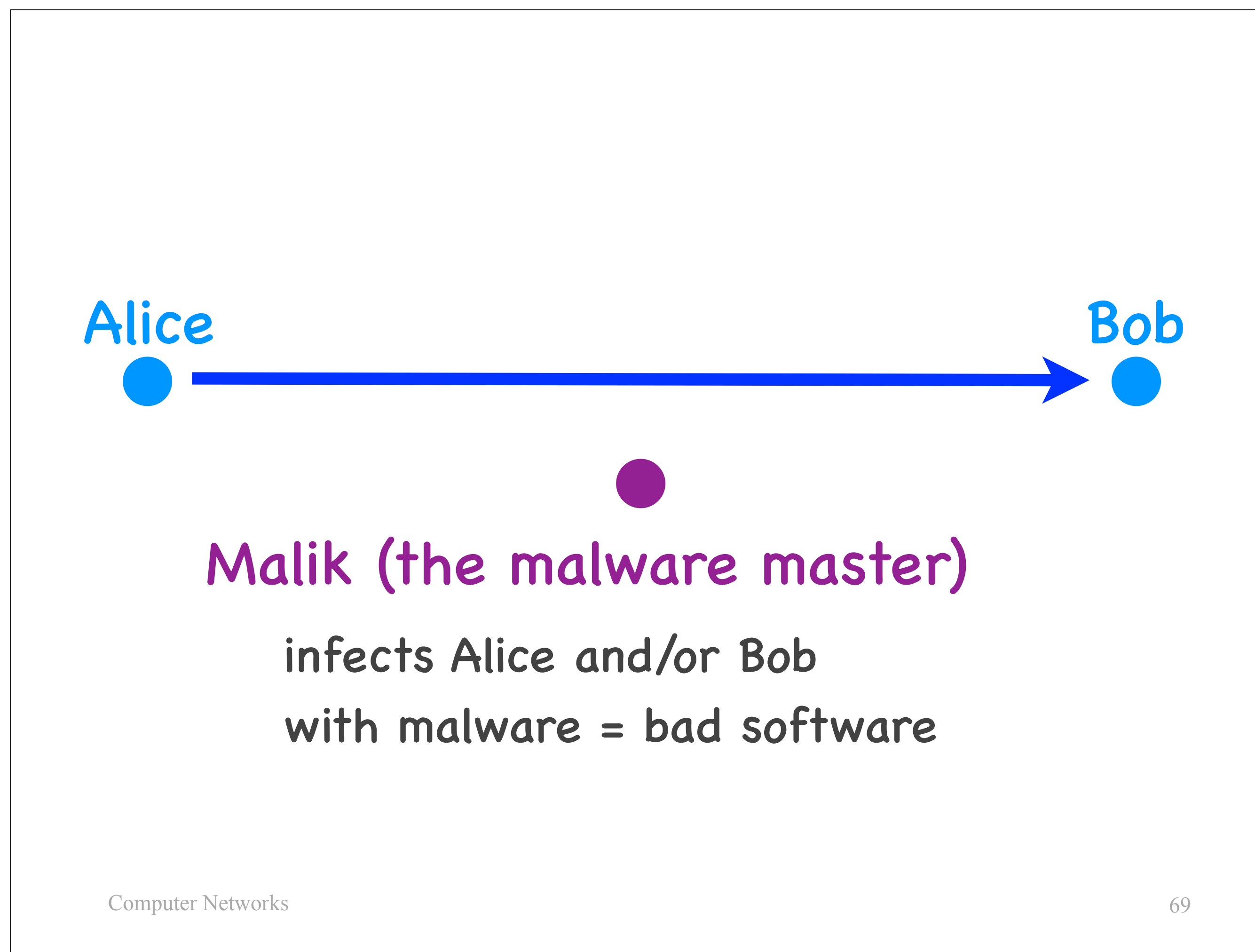
He can do that alone or…



distributed denial-of-service attack

disrupts the communication
between Alice and Bob

… even better, enlist the help of many compromised end-systems, also known as bots.



Finally, another type of misbehaving user is Malik, the malware master, who infects Alice’s and/or Bob’s computer with malware, meaning bad software that does bad things their computers, like delete their data, or steal their data, or makes their computers send spam or participate in denial-of-service attacks.

Internet vulnerabilities

- Eavesdropping (sniffing)
- Impersonation (spoofing)
- Denial of service (dos-ing)
- Malware

So, there are at least 4 kinds of vulnerabilities in a network, in general, and on the Internet, in particular: …

What trust model to design for?

So, one more key question to ask when designing a network is: what to assume about the behavior of its users?

What physical infrastructure is already available?

What modularity & hierarchy?

What layers to define?

Treat on demand or take reservations?

What trust model to design for?

I will close the introduction by restating 5 key questions to ask when designing a network: …