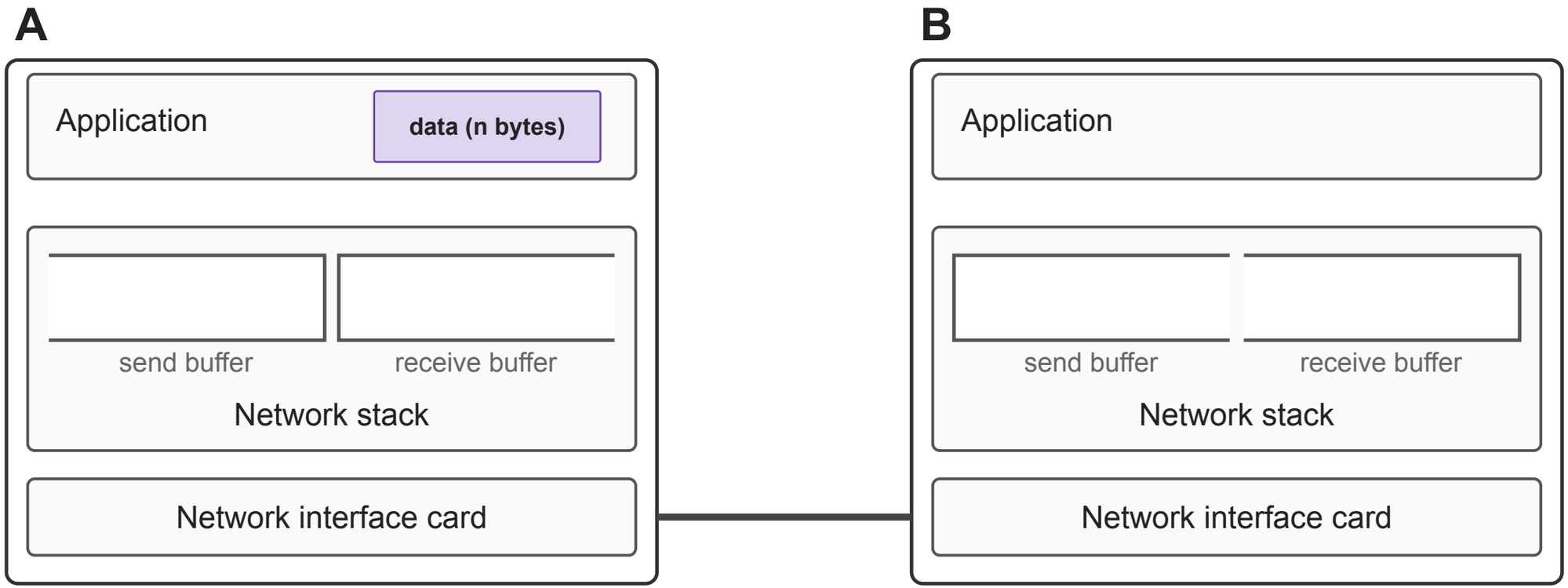
UNIVERSITY OF WATERLOO

The transport layer: Introduction

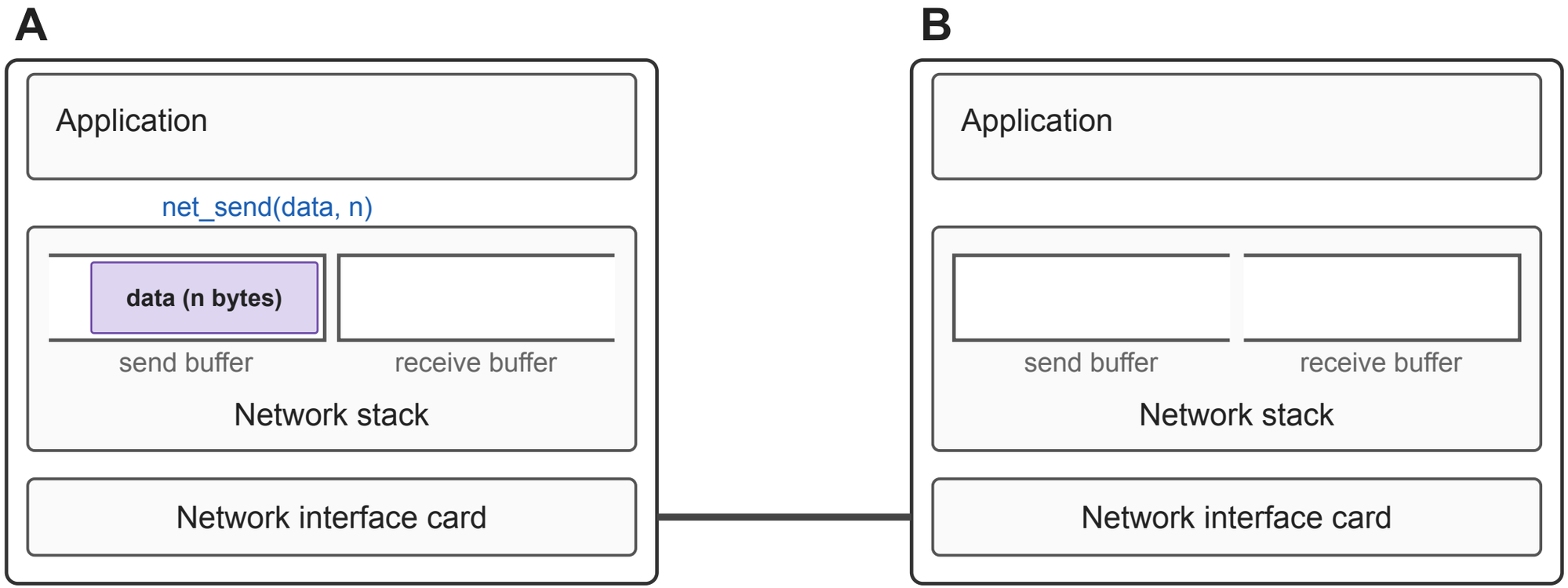
CS 456/656, Fall 2026

Mina Tahmasbi Arashloo

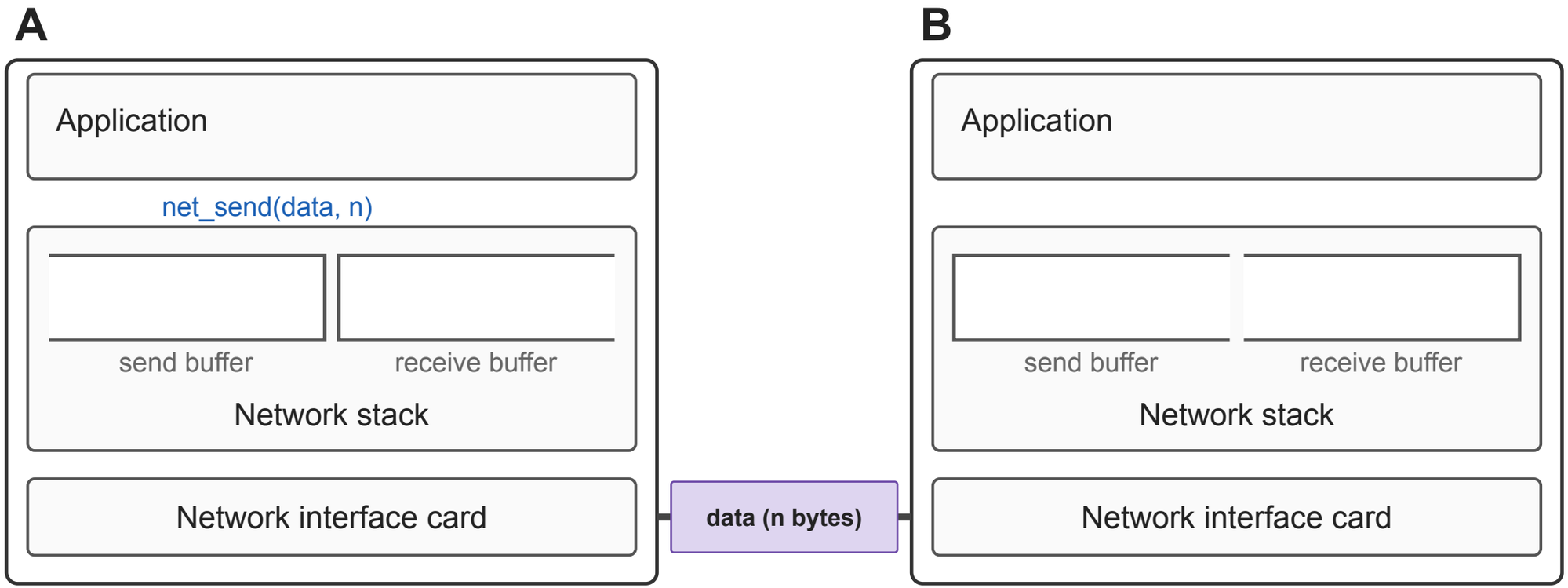
The picture we have been using



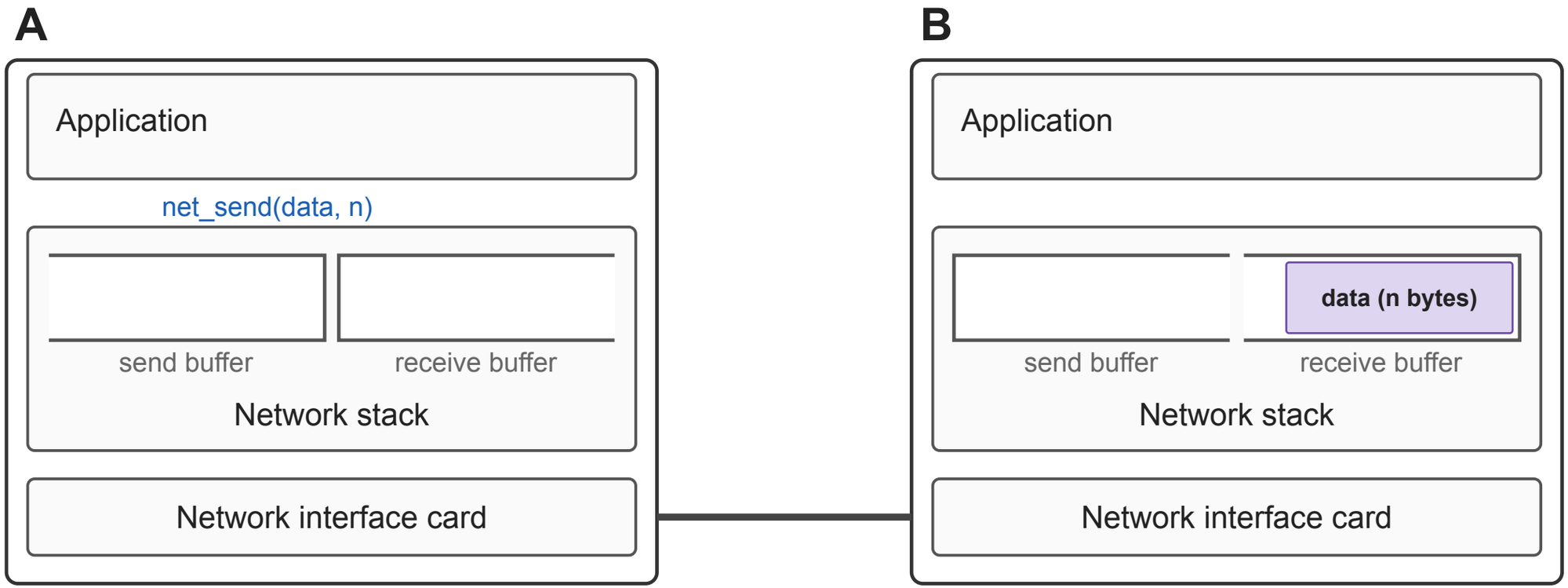
The picture we have been using



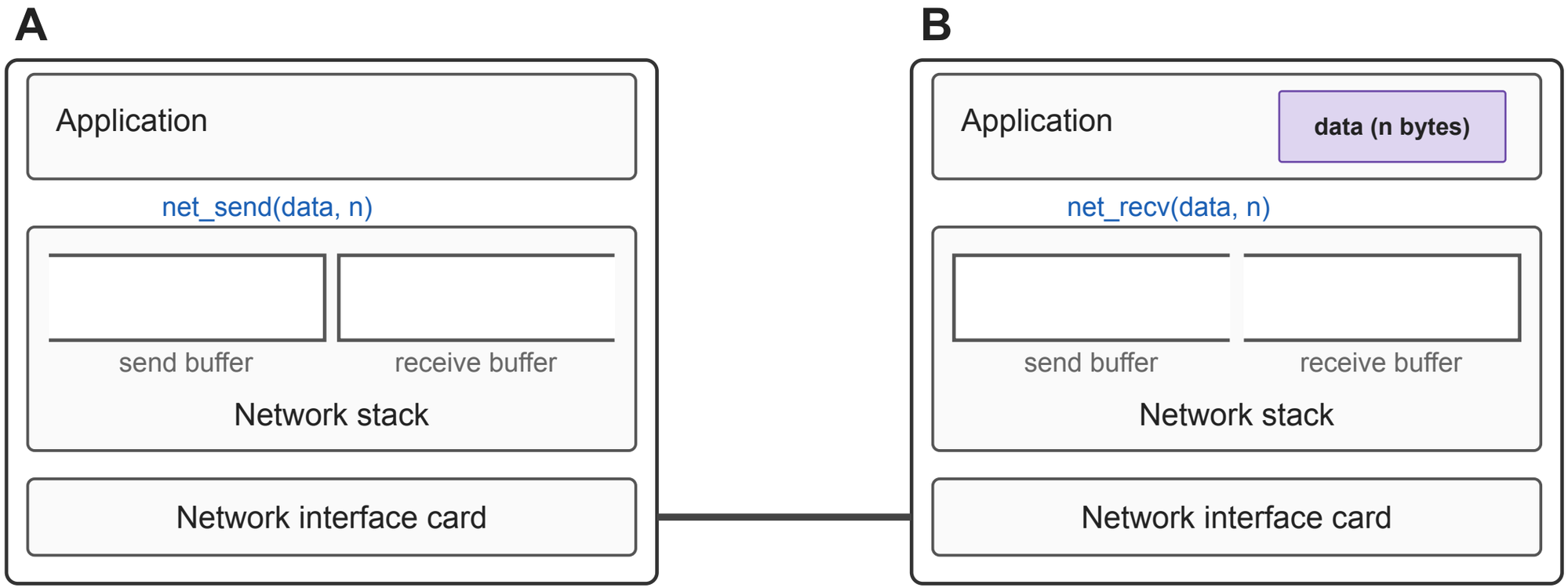
The picture we have been using



The picture we have been using

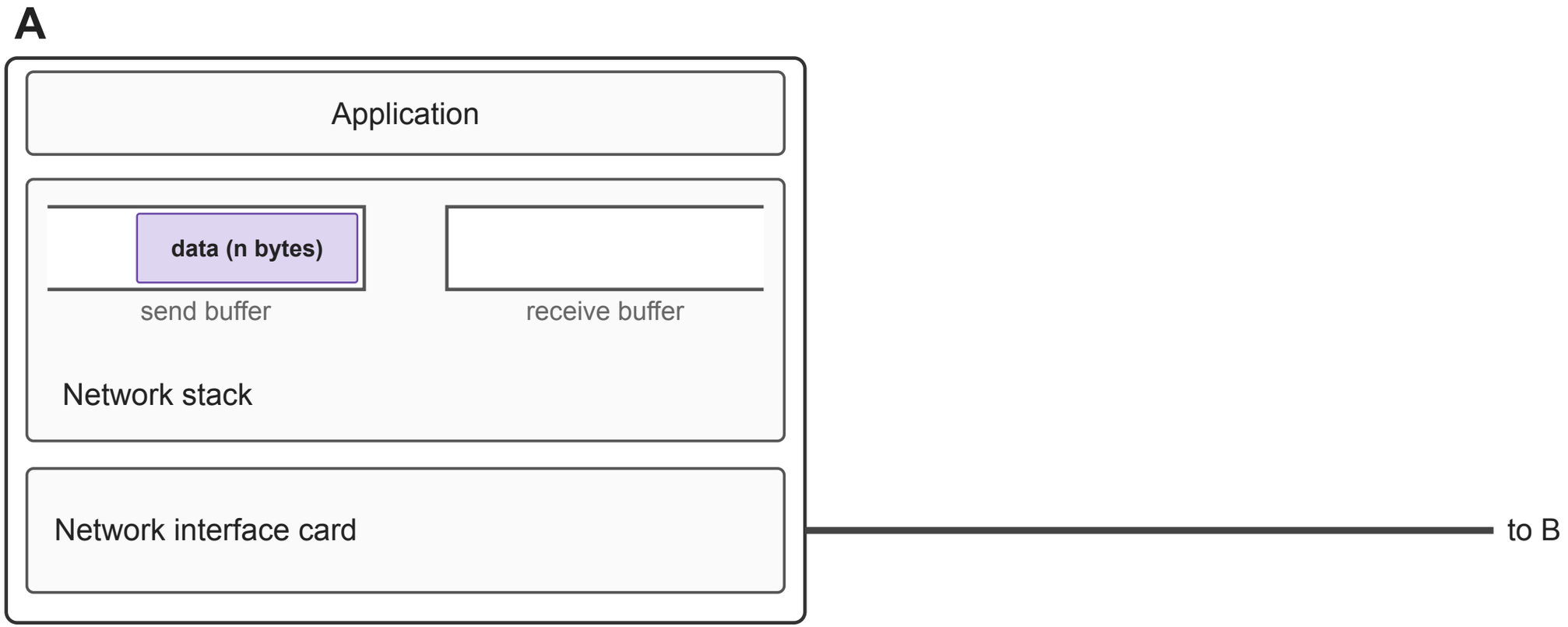


The picture we have been using

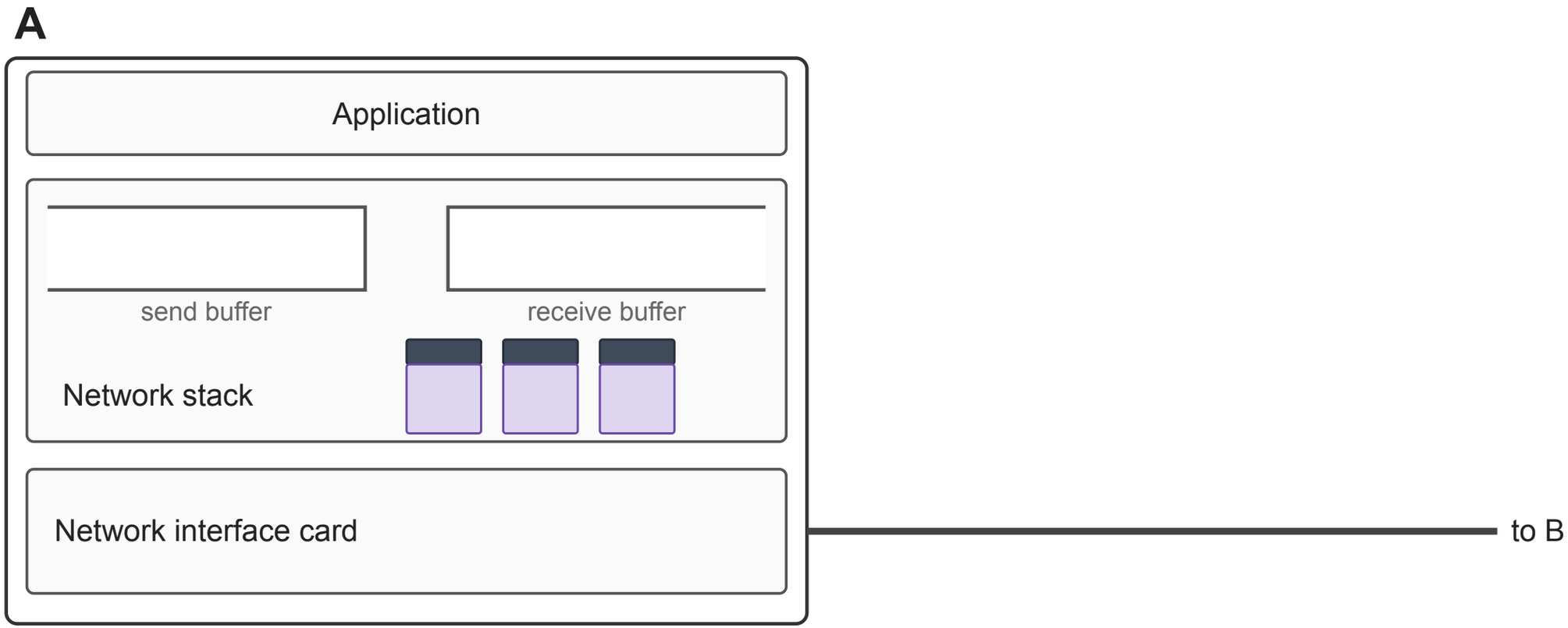


The network does not transfer data in units of arbitrary size

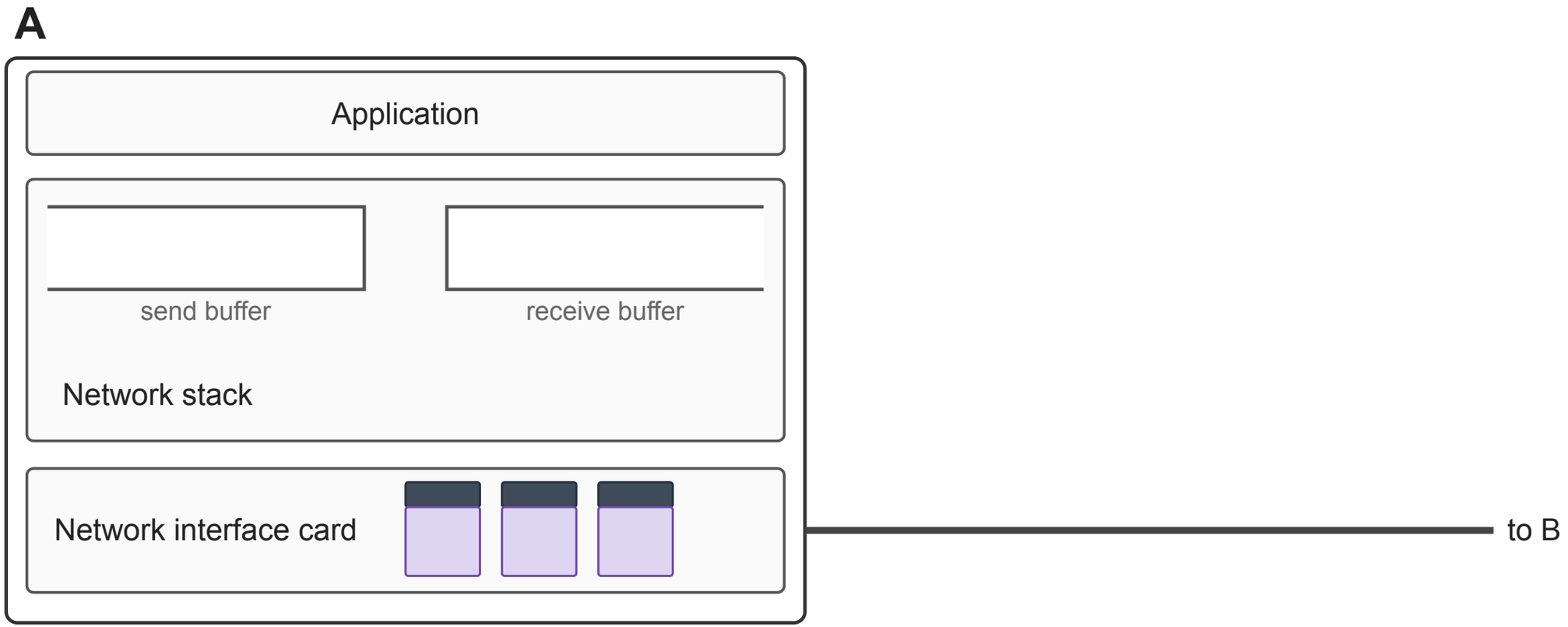
The network transfers data in packets



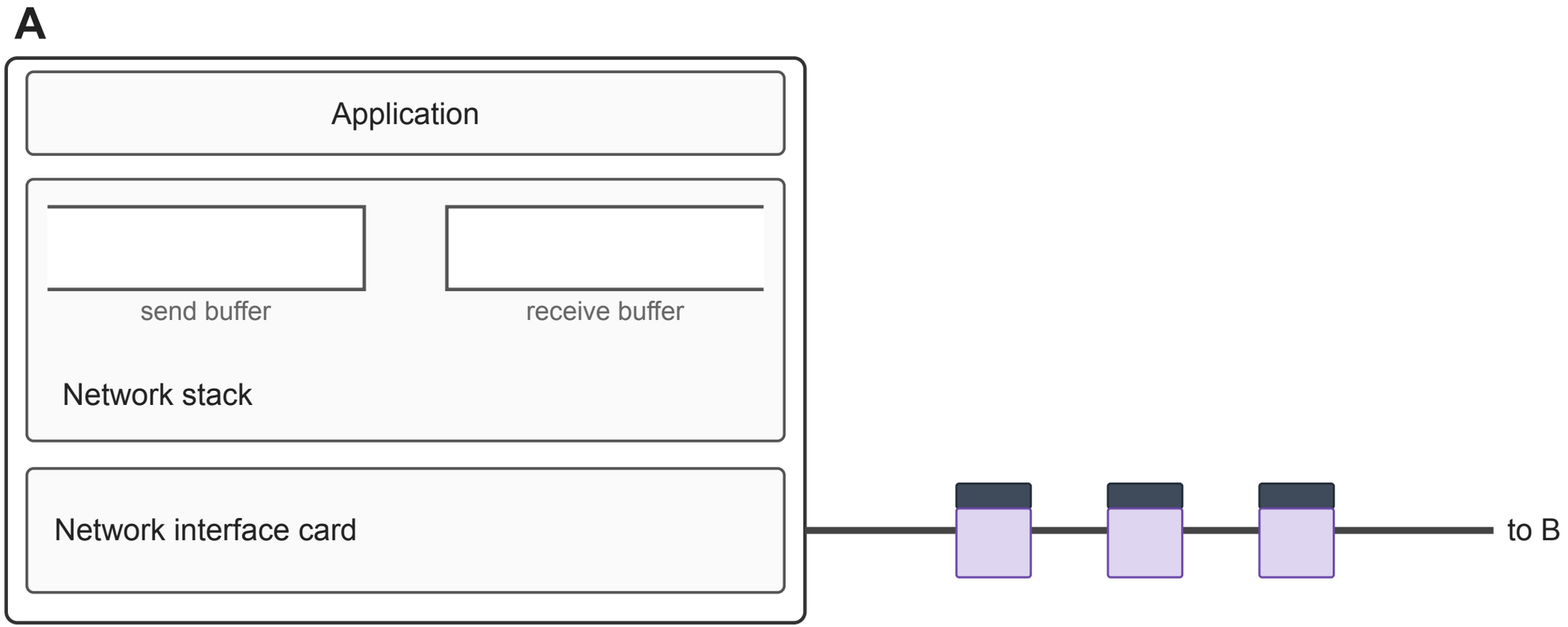
The network transfers data in packets



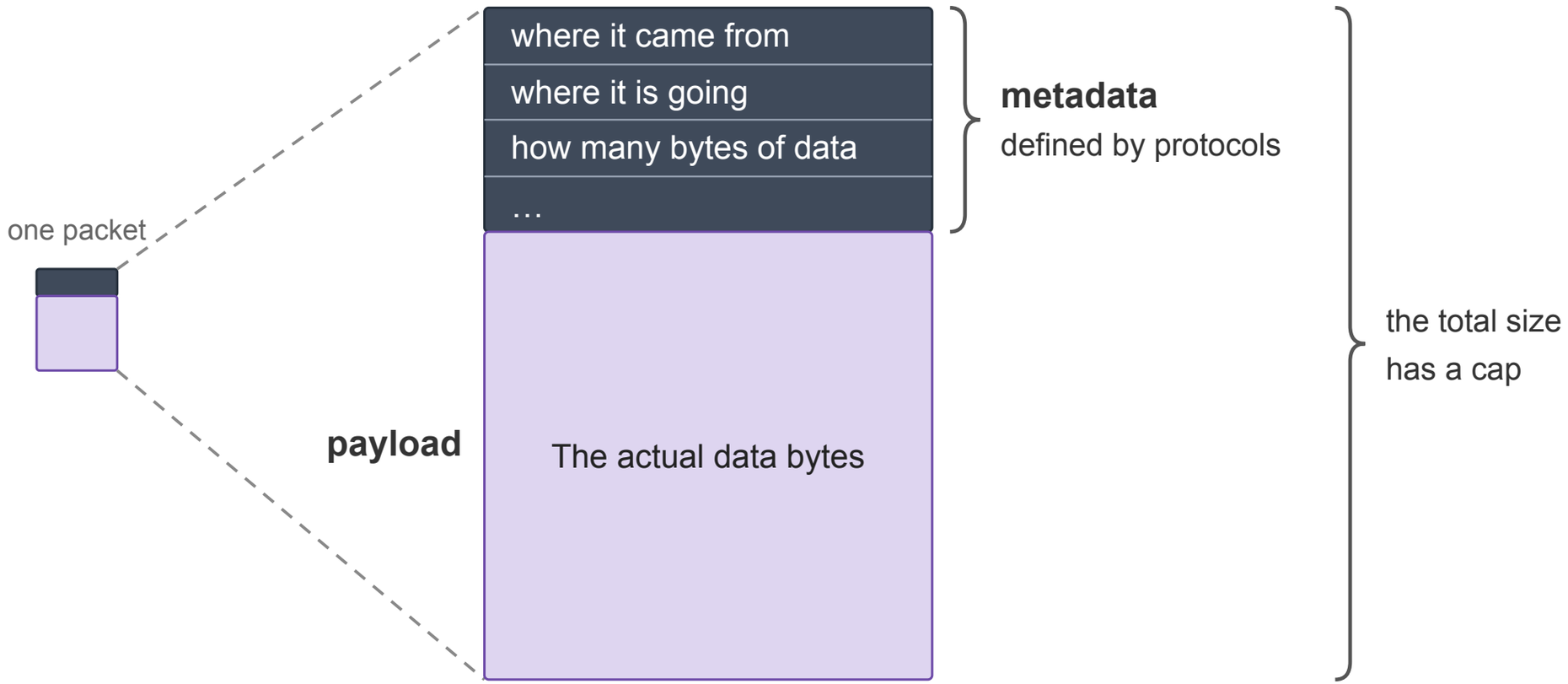
The network transfers data in packets



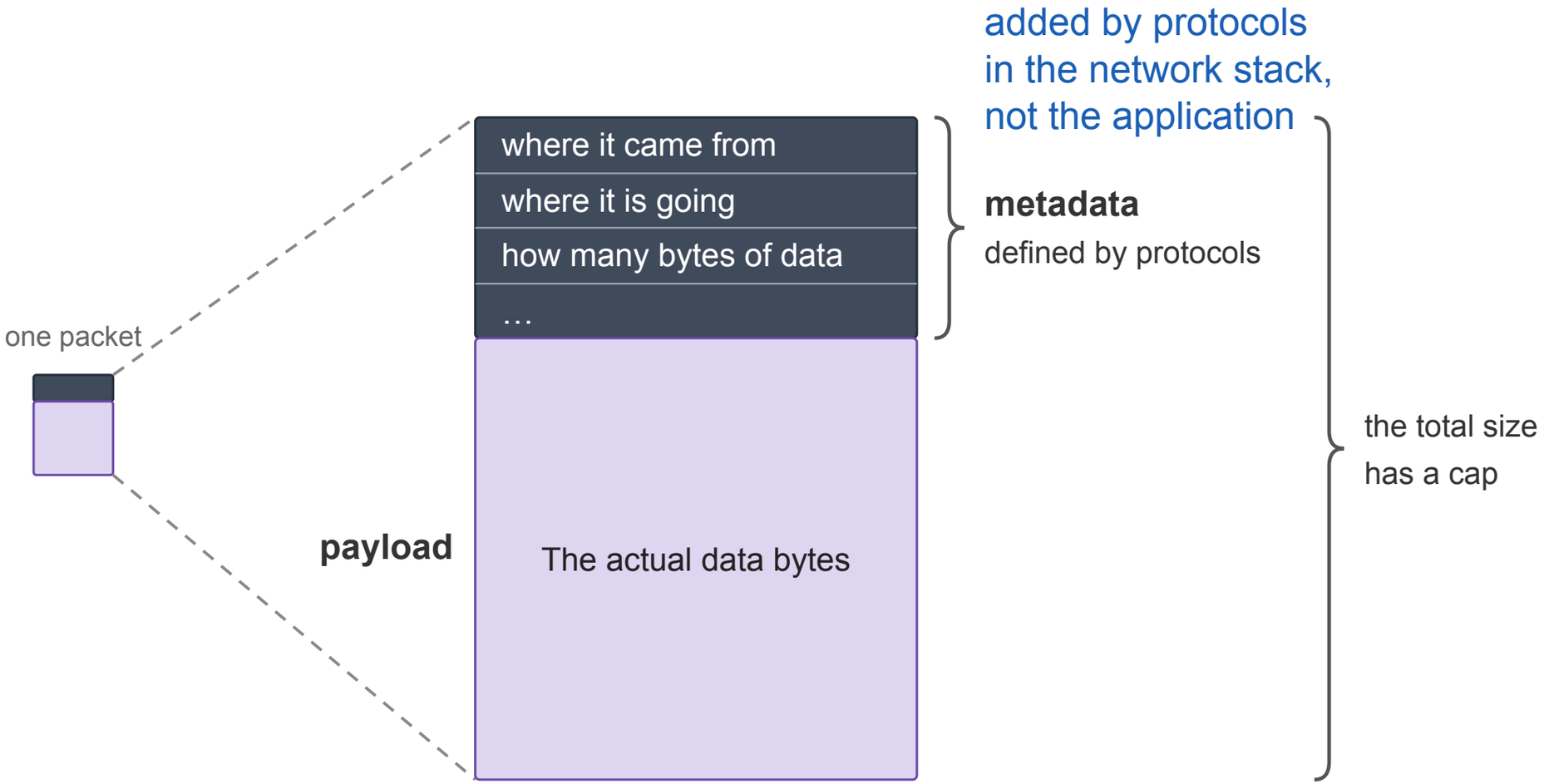
The network transfers data in packets



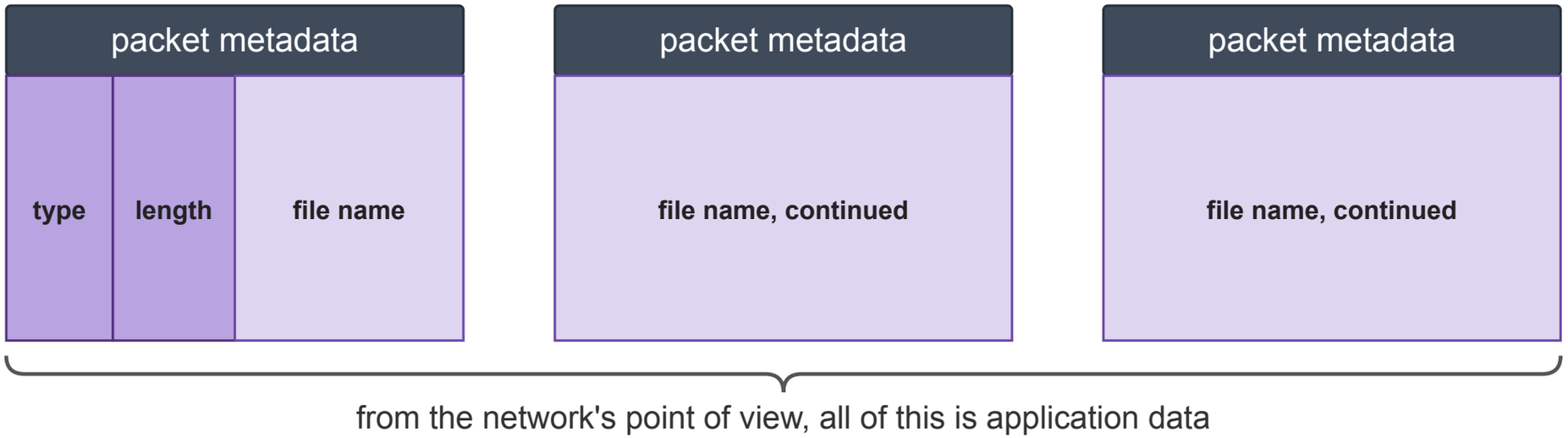
What is in a packet?



What is in a packet?



The application has its own metadata

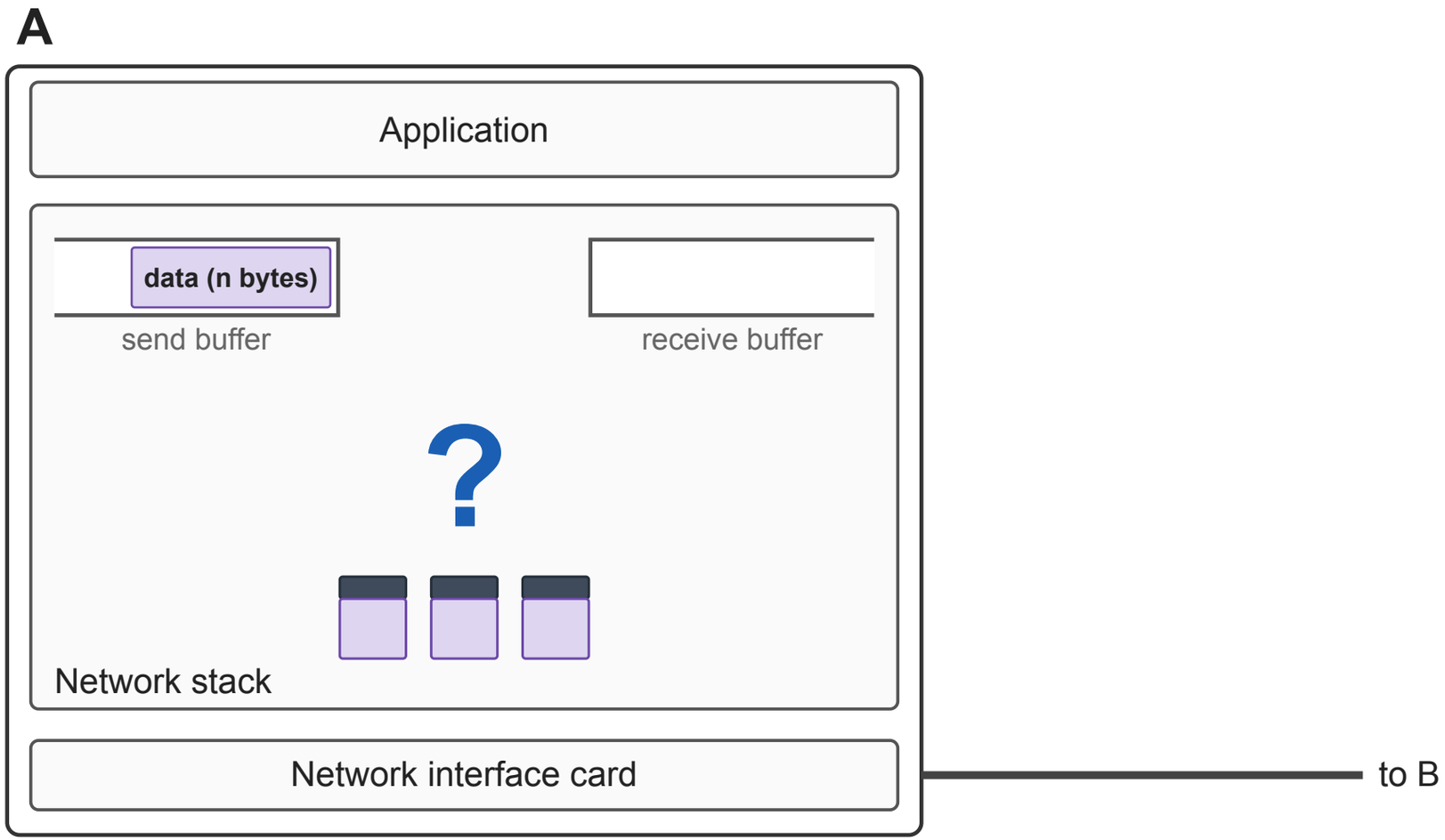


packet metadata, from the network stack

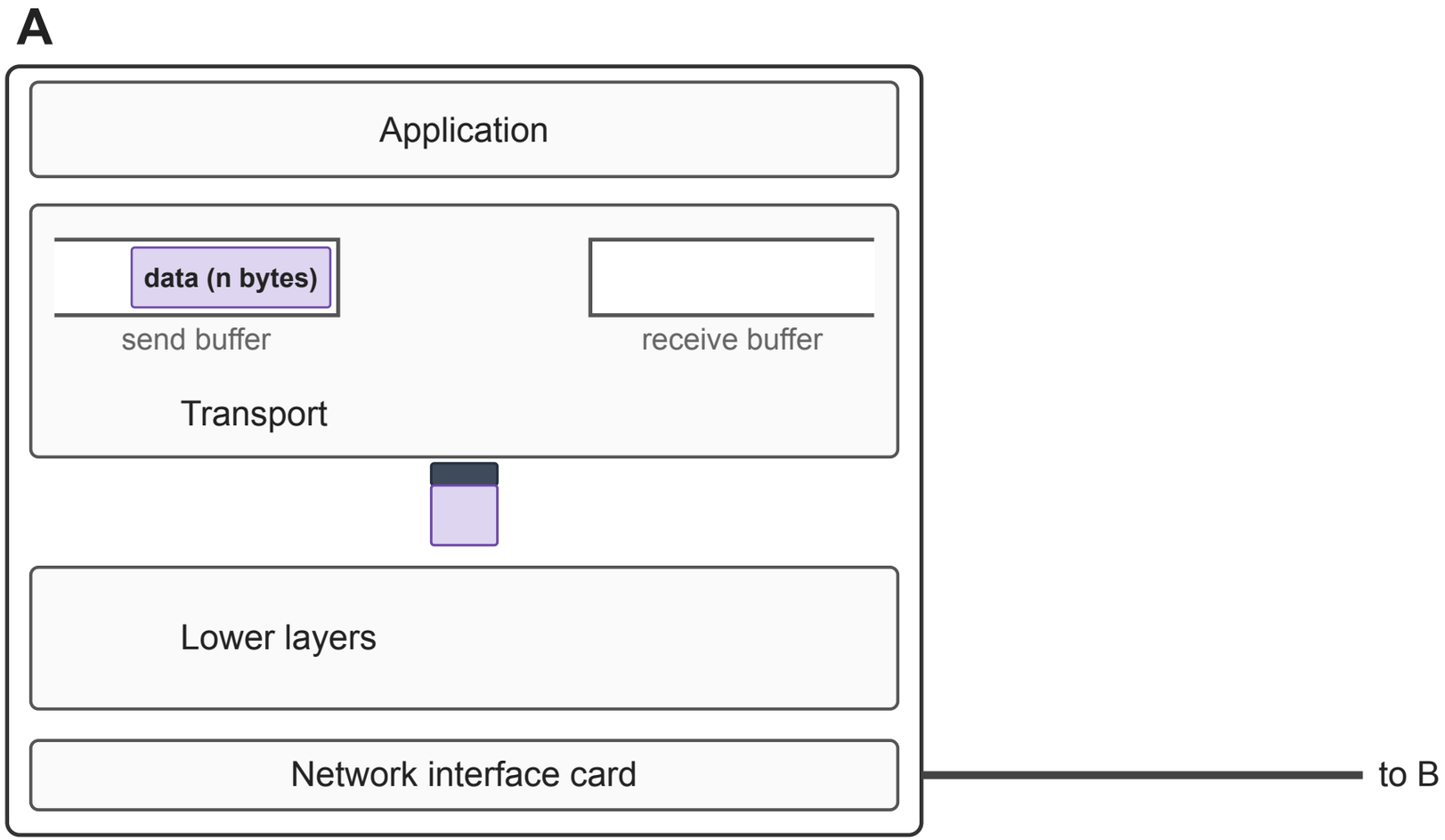
metadata from the application layer protocol, e.g. our web browser

application data

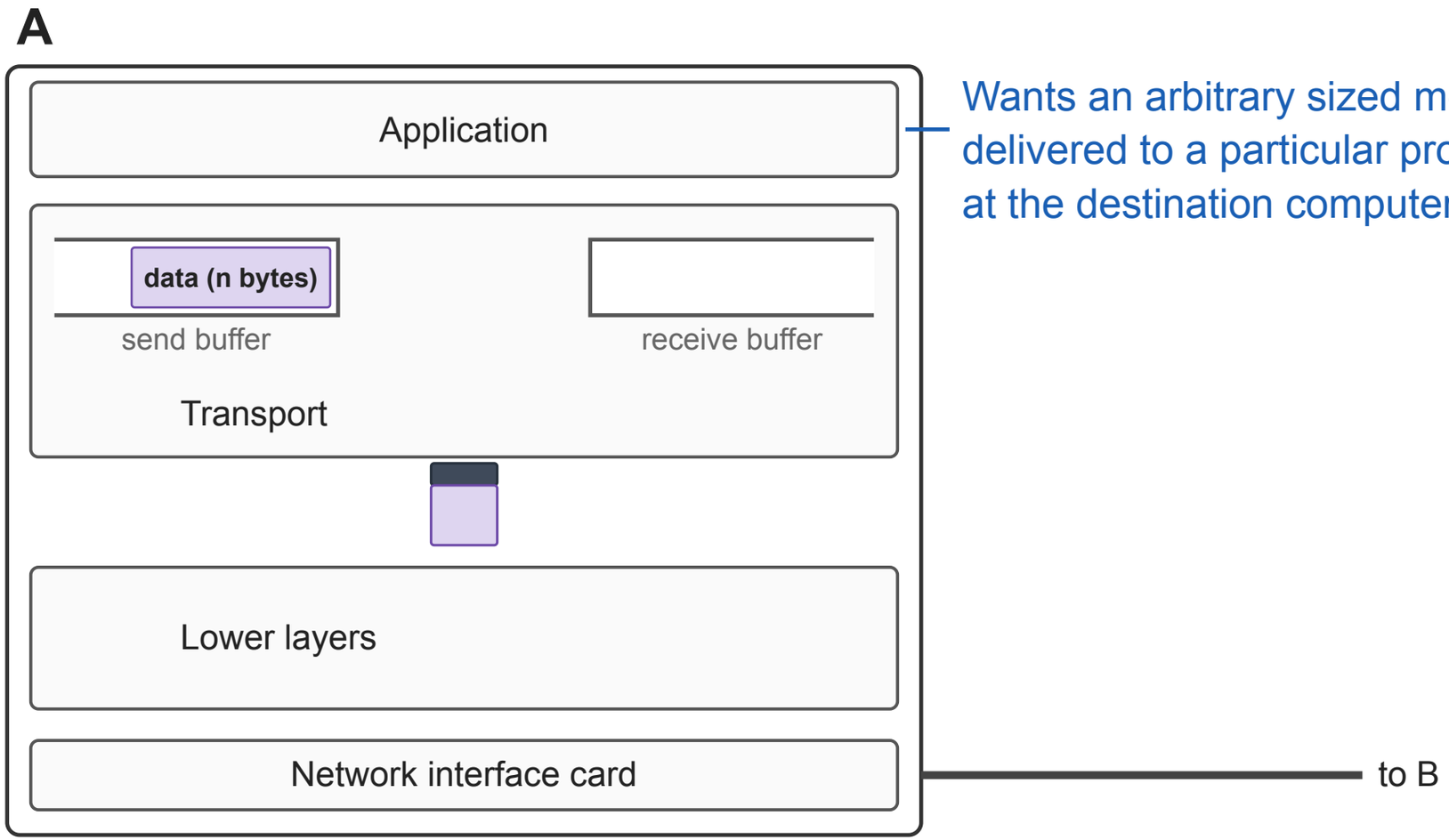
How do we bridge the gap?



The transport layer

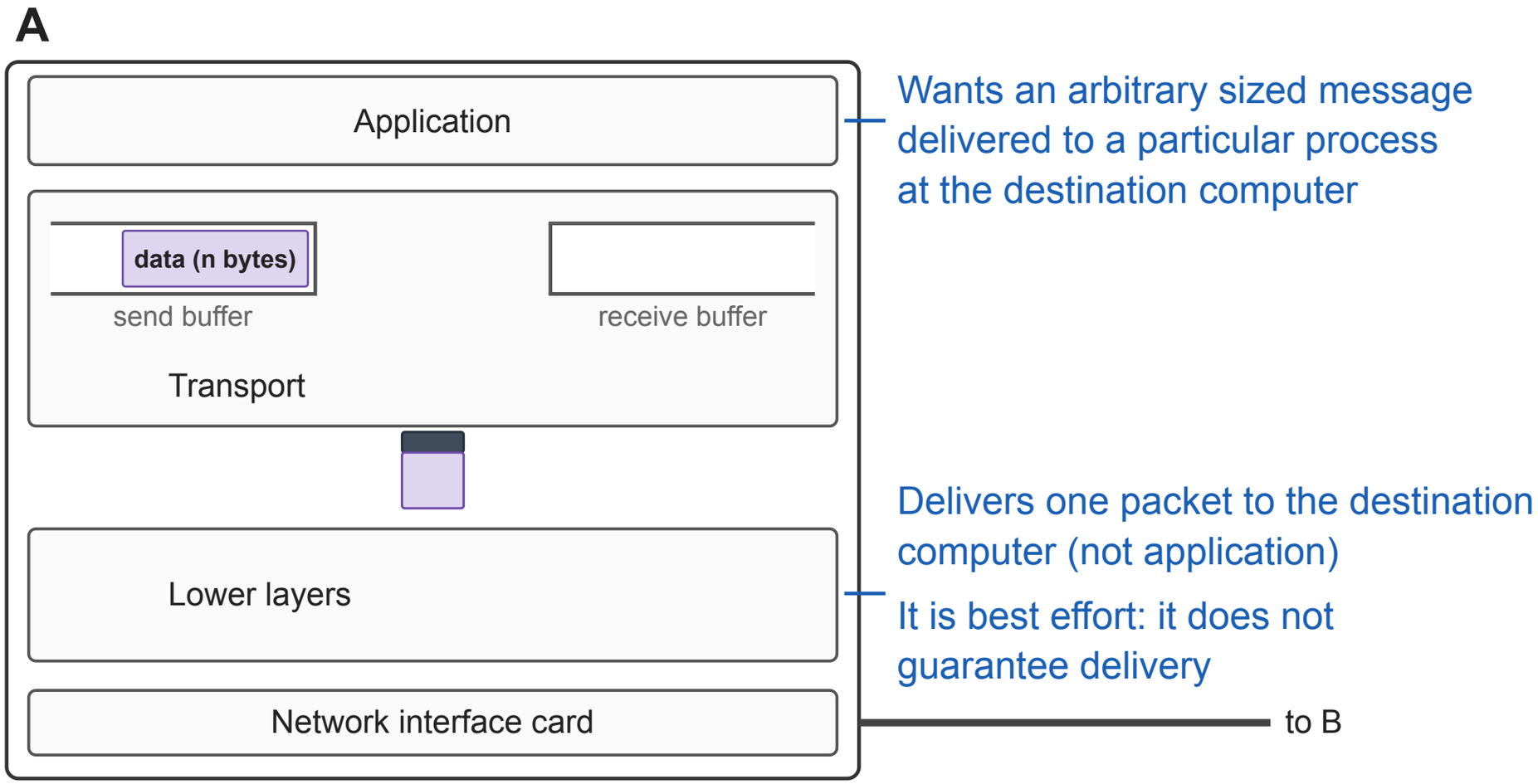


The transport layer



Wants an arbitrary sized message delivered to a particular process at the destination computer

The transport layer

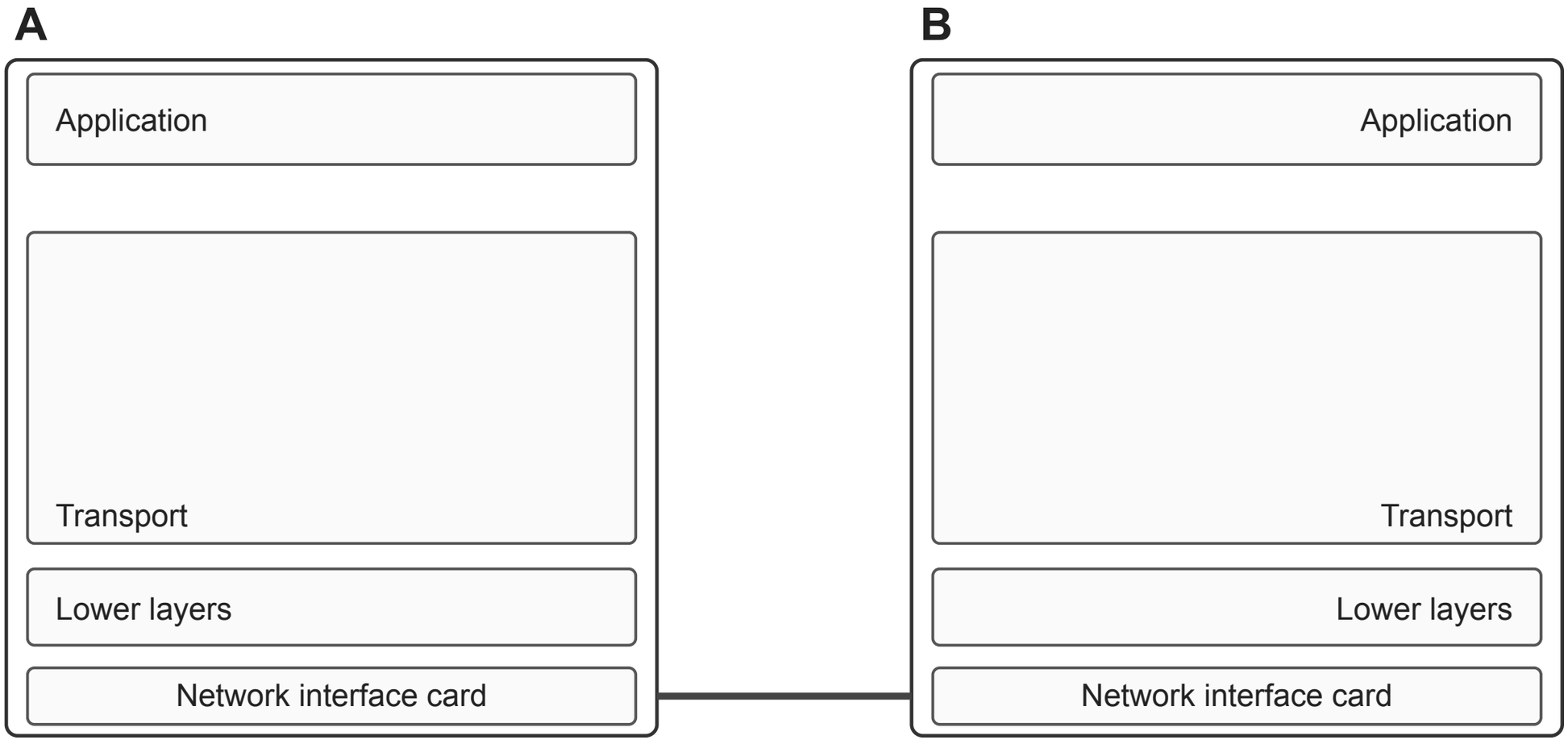


What does the transport layer have to take care of?

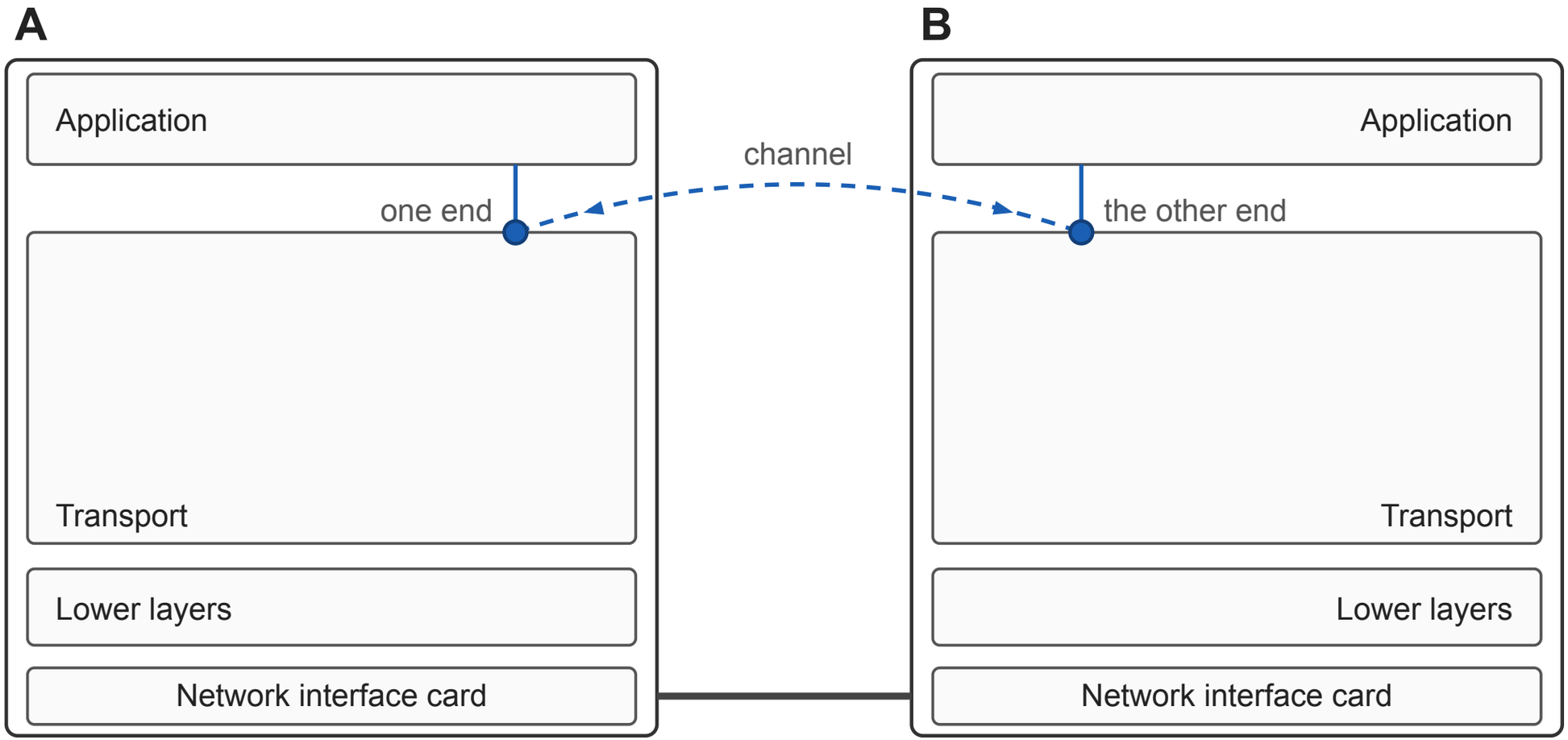
- Guarantee packets are delivered (ensure destination is reachable)
- Packetization of data (divide it up into appropriate sized packets) and add the appropriate metadata
- Delivering it to the right application (application should provide proper source and destination information)
- Extract data from packets and put it back in the right order (at the receiver)
- Security? data should be encrypted over the network
- Handle malformed packets

Reset

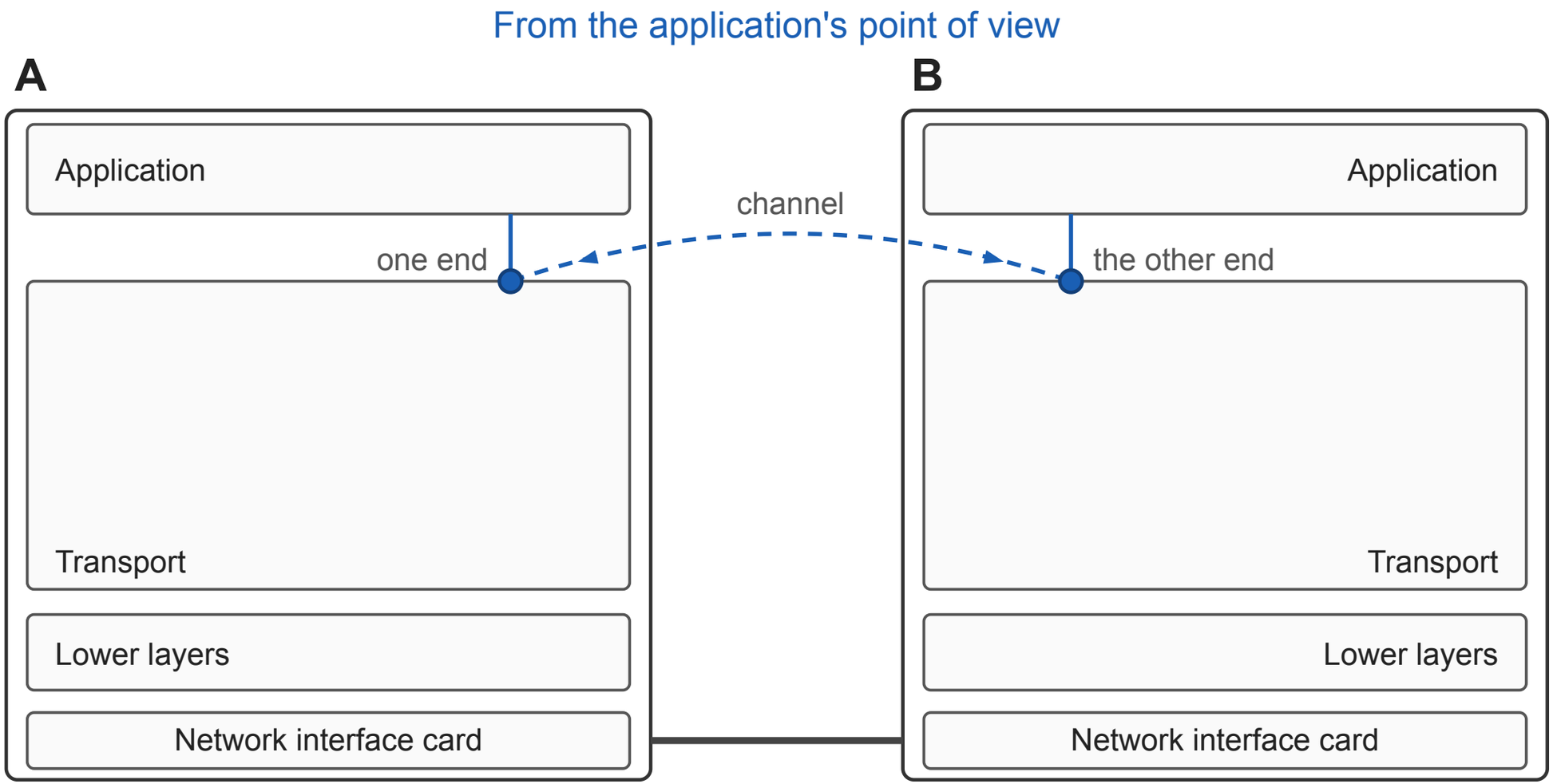
The application asks for a channel



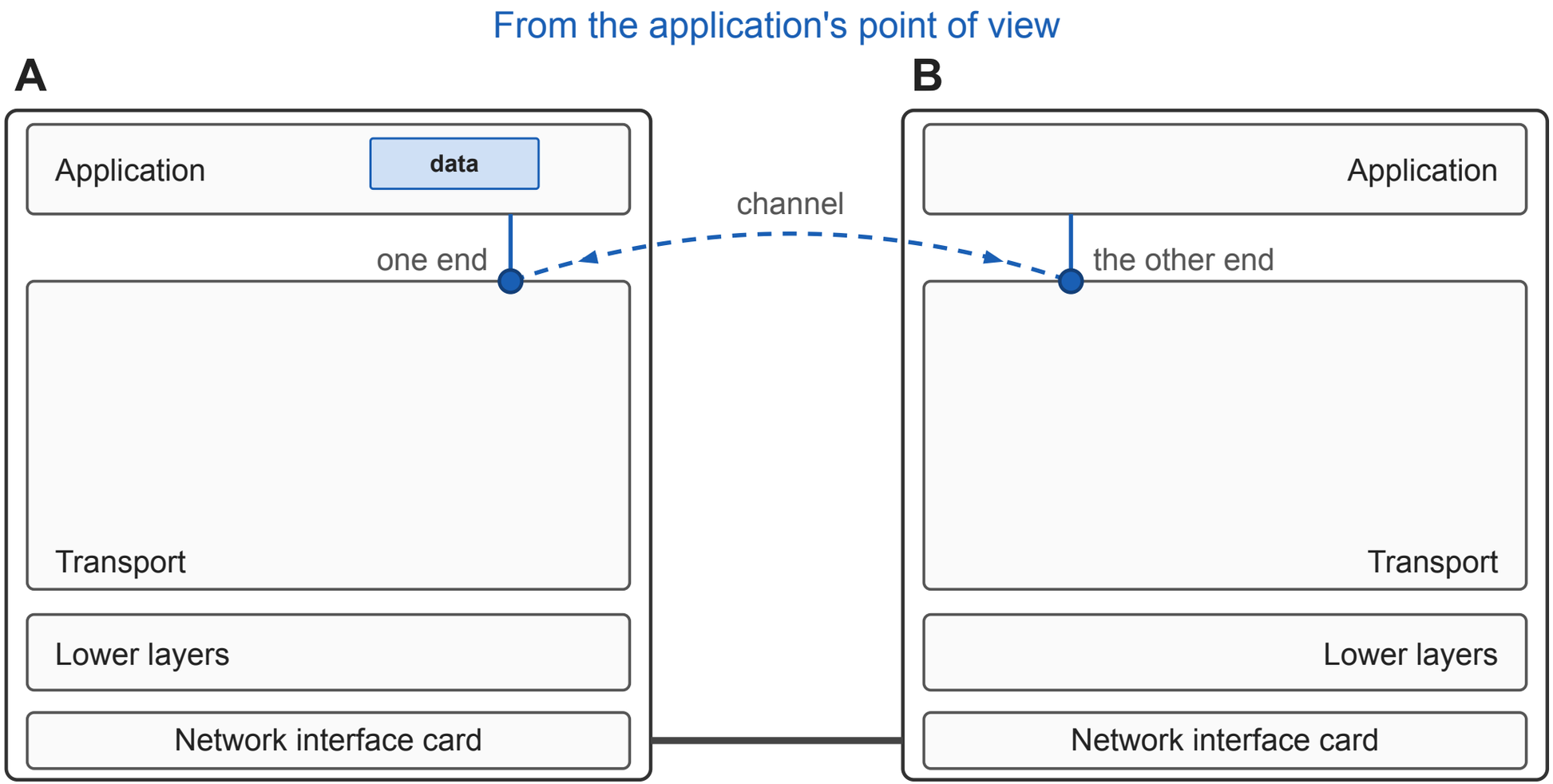
The application asks for a channel



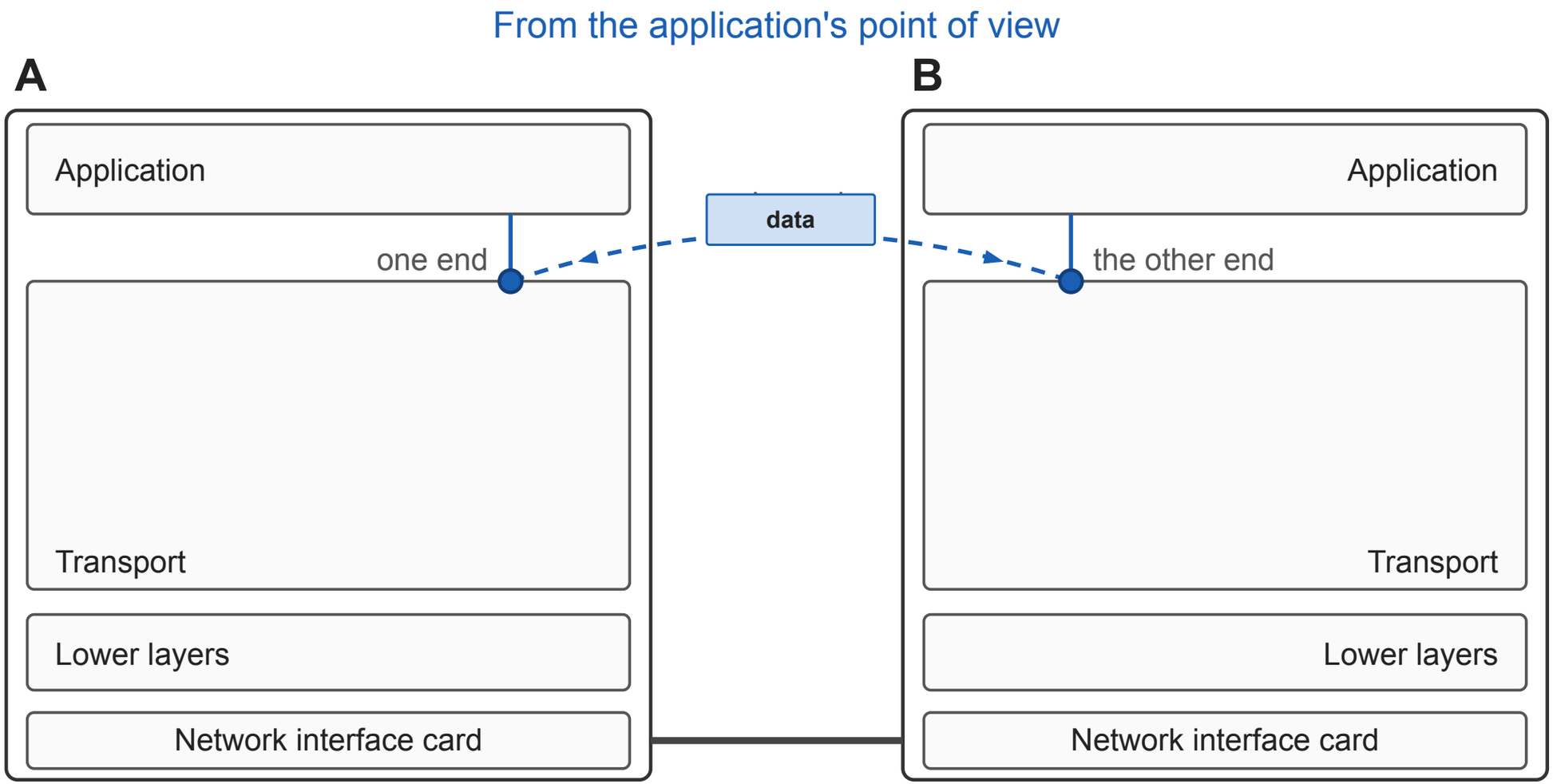
The application asks for a channel



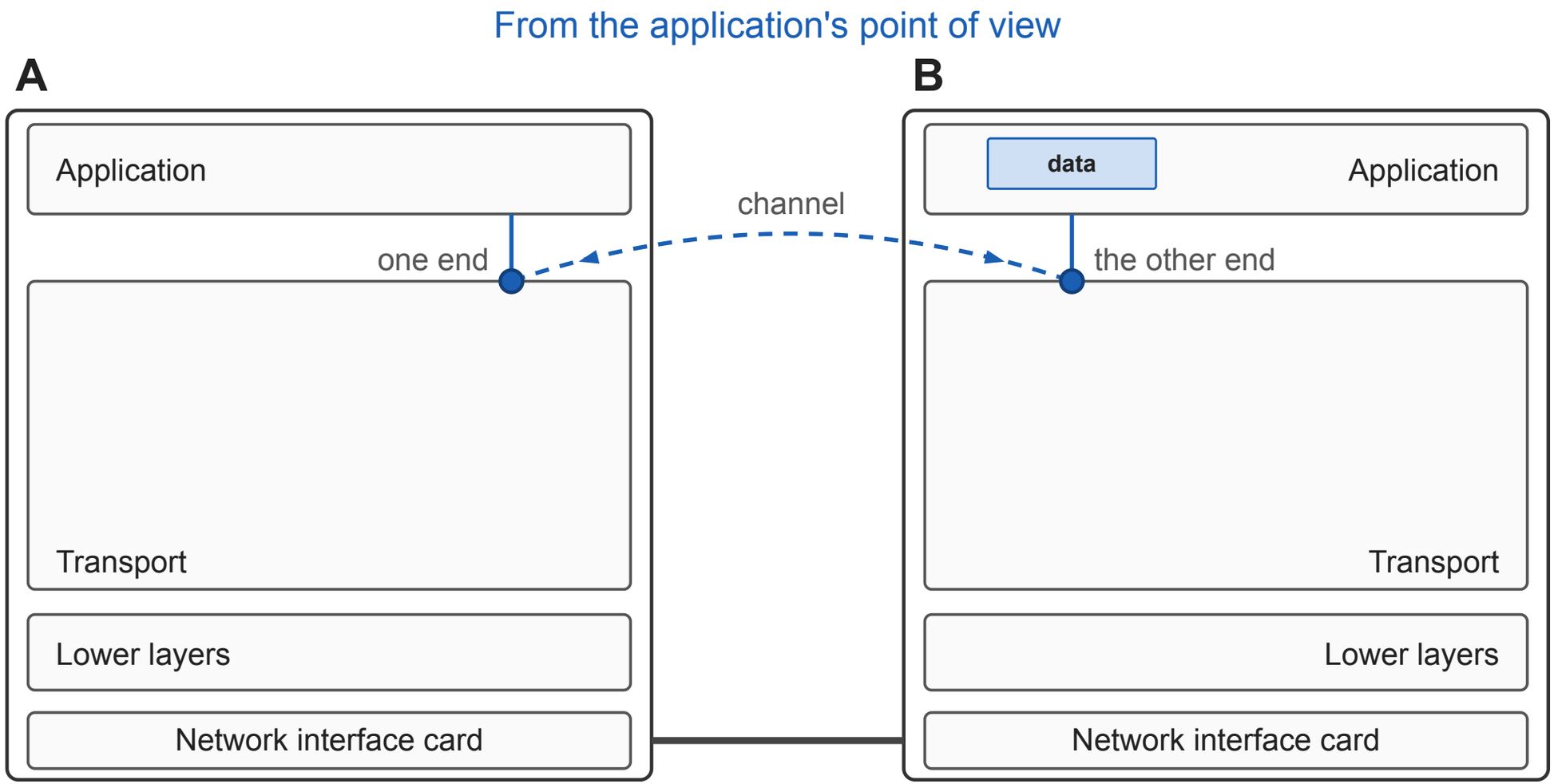
The application asks for a channel



The application asks for a channel

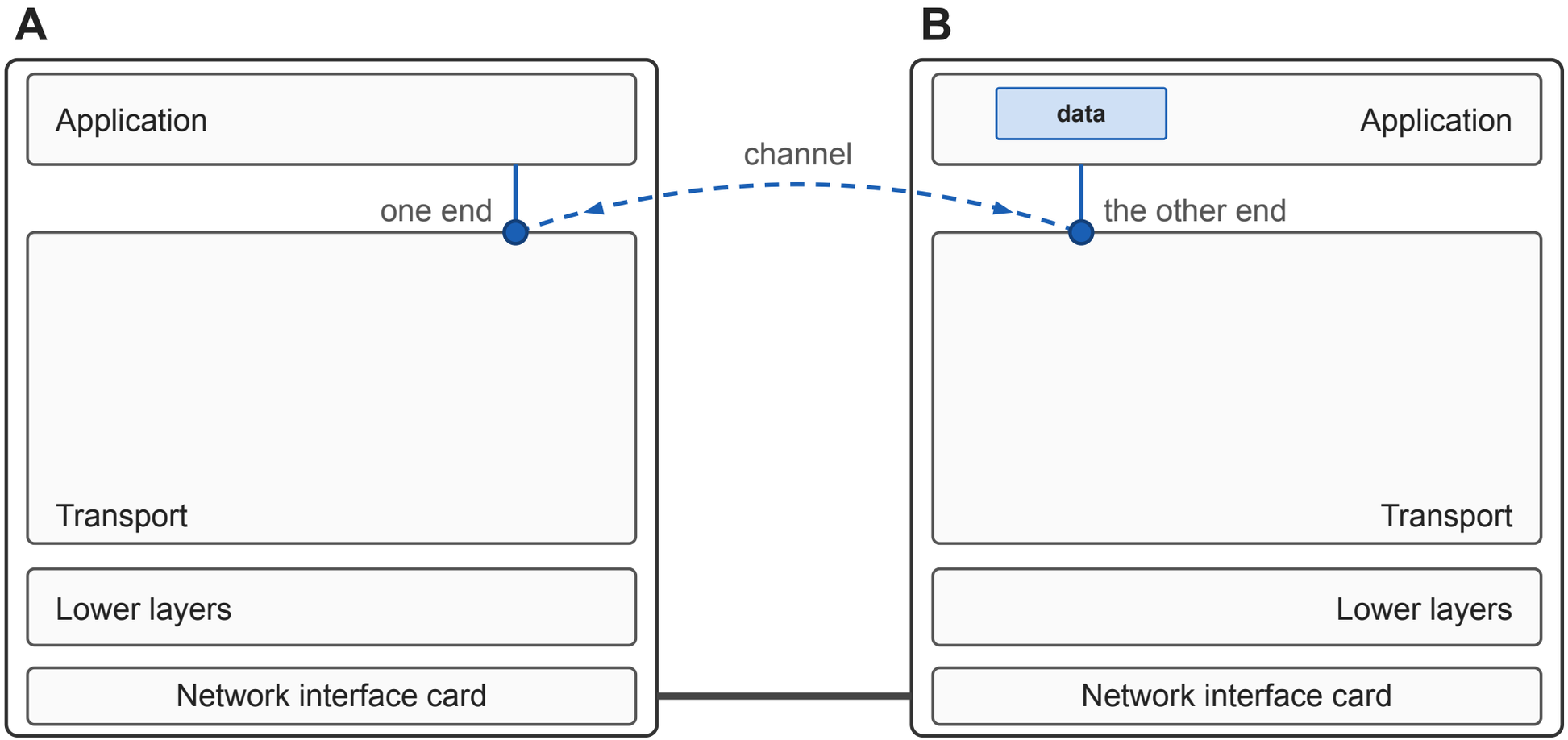


The application asks for a channel



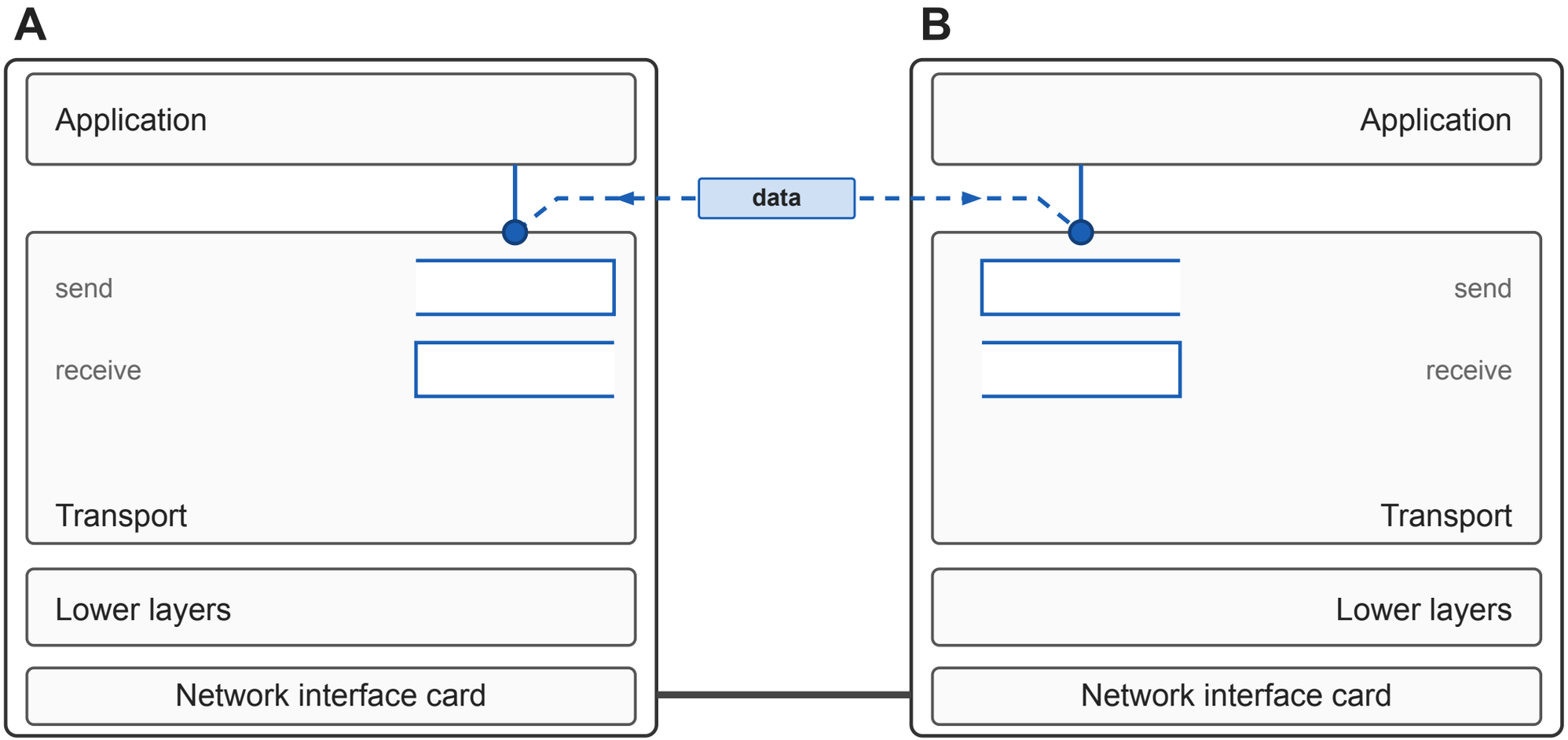
The application asks for a channel

From the application's point of view

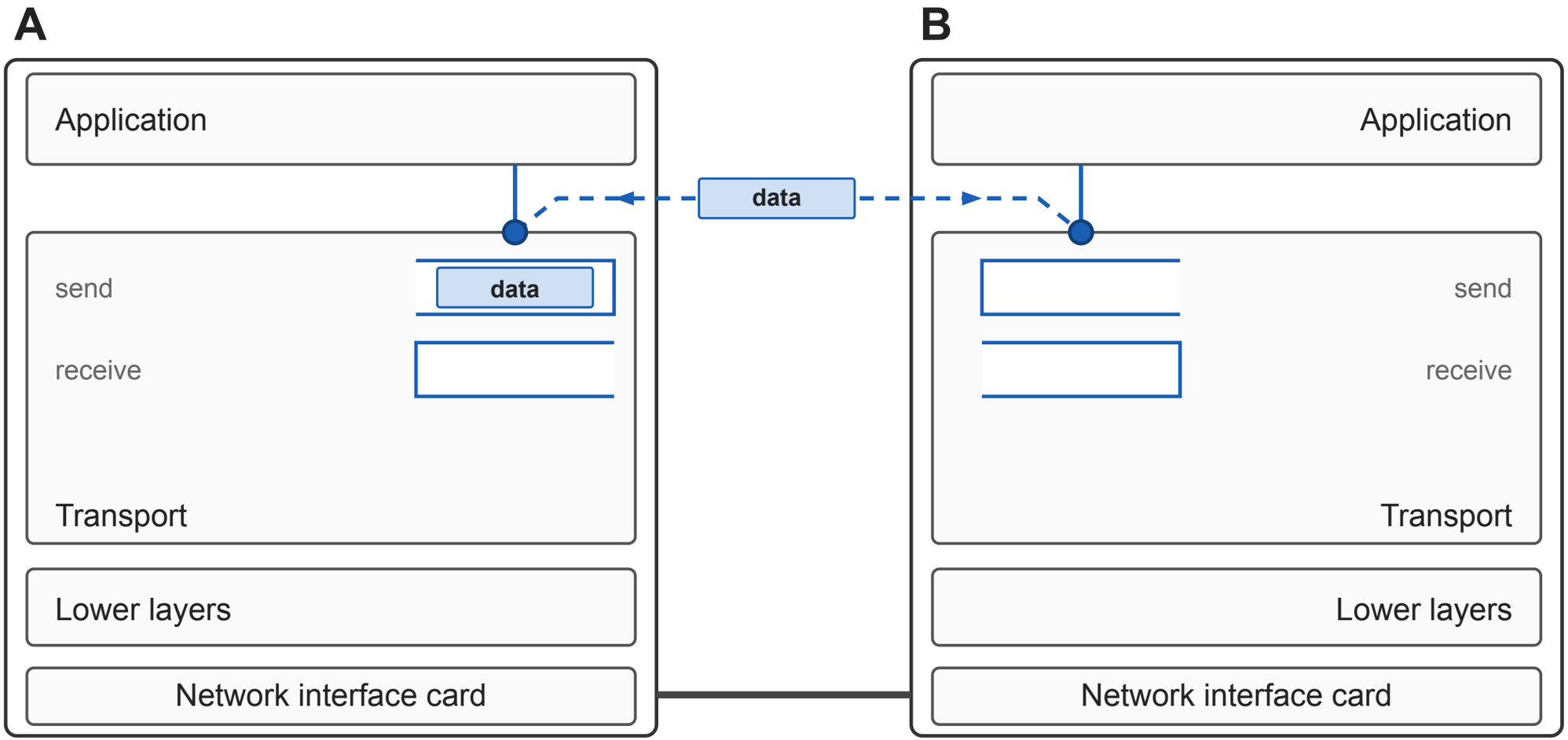


The application does not see packets

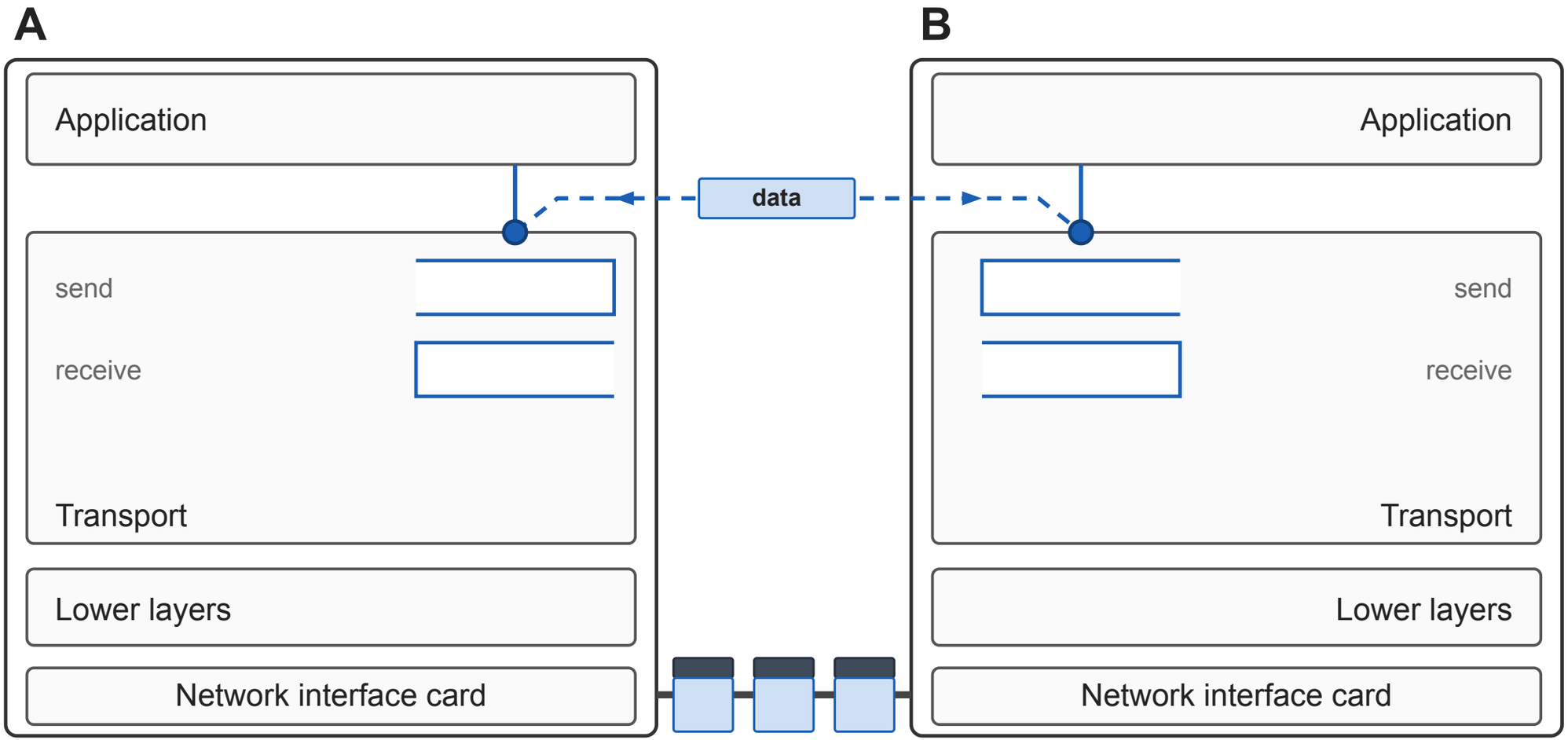
What happens underneath



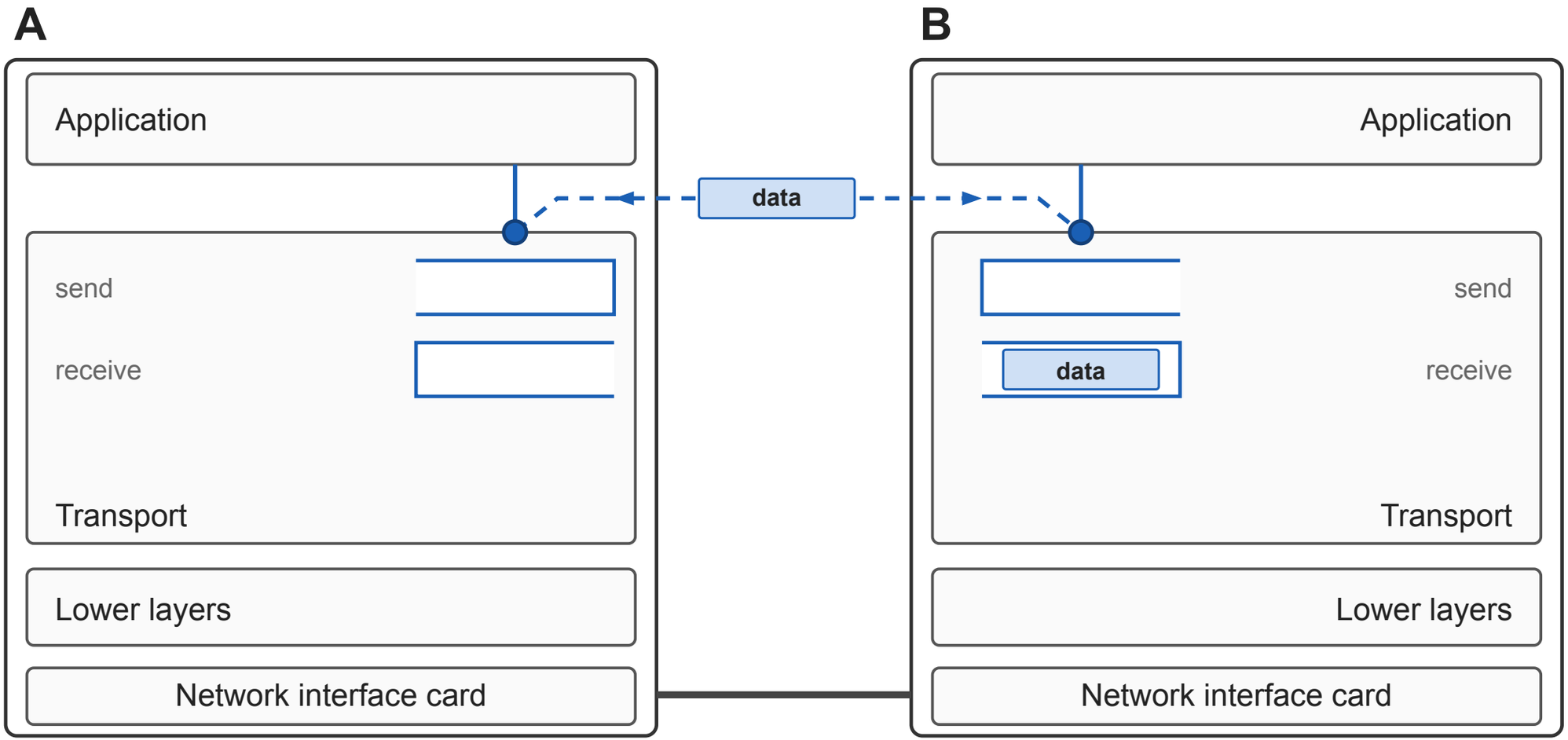
What happens underneath



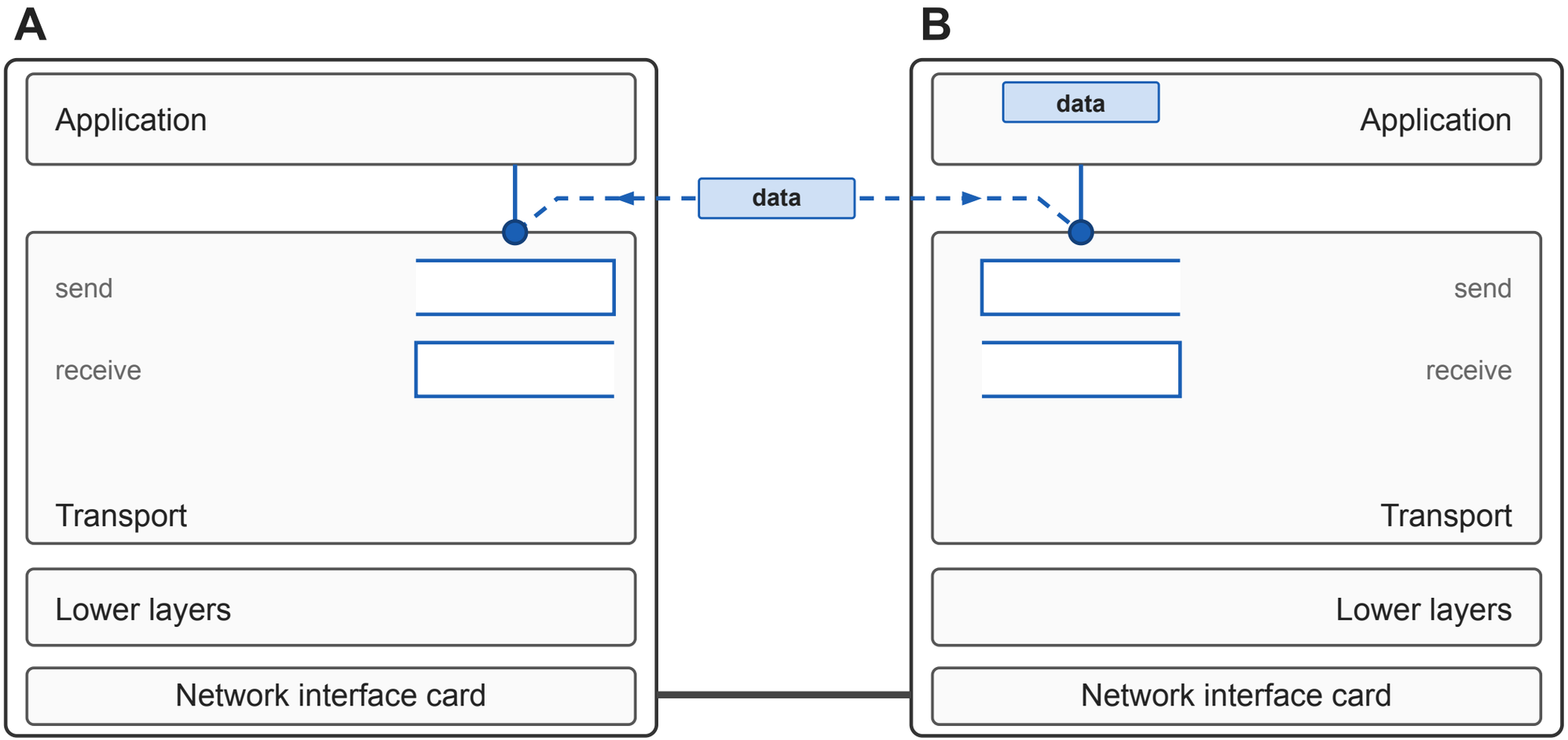
What happens underneath



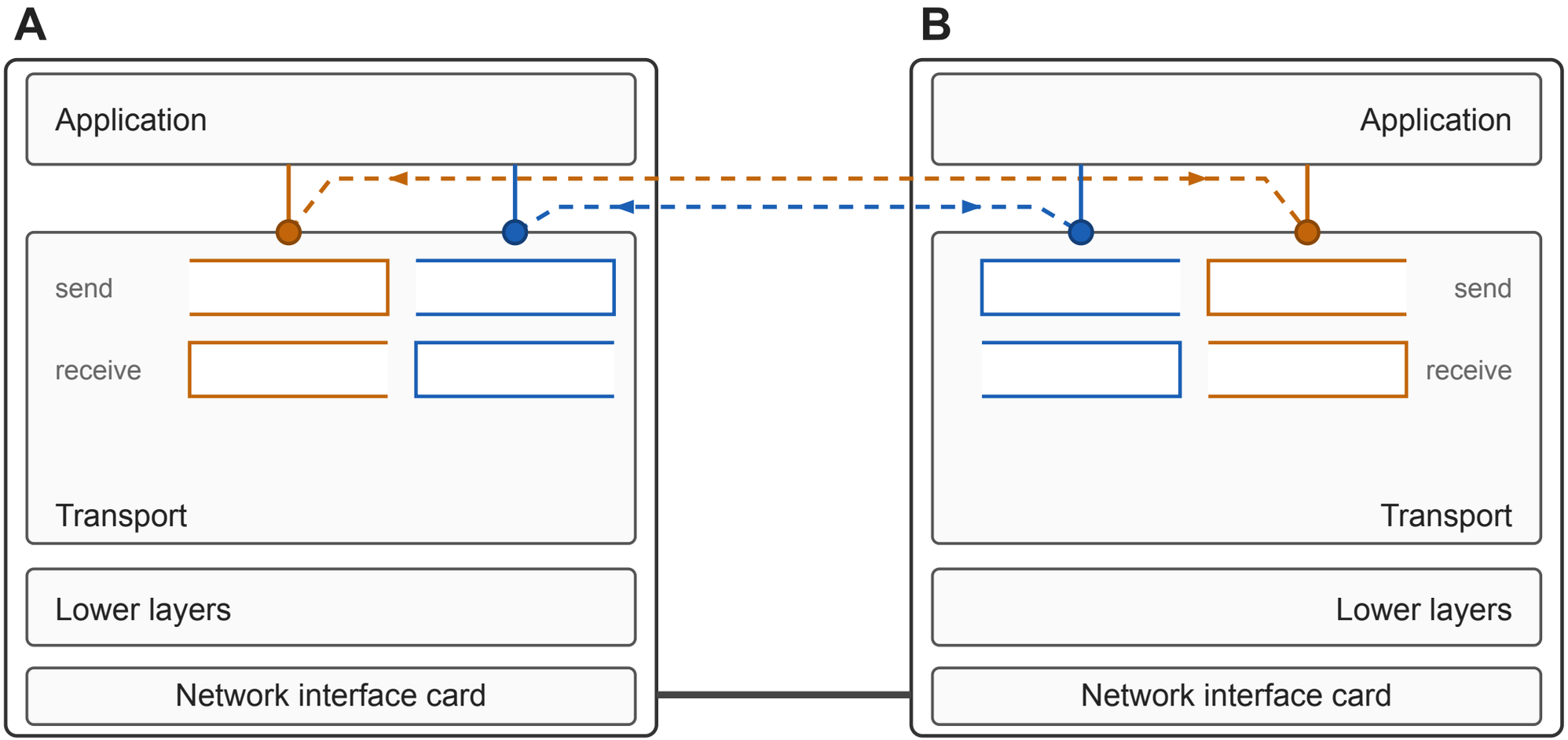
What happens underneath



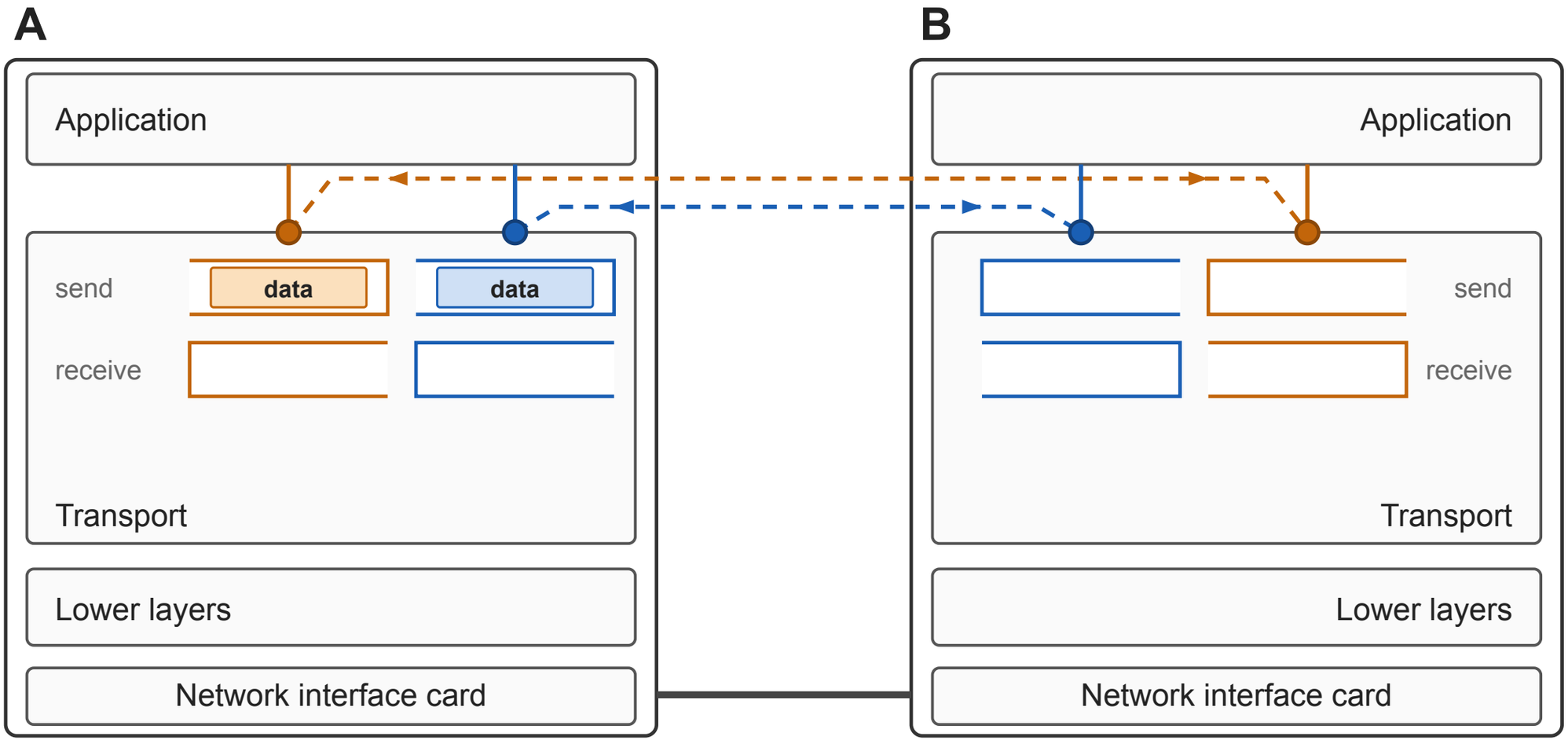
What happens underneath



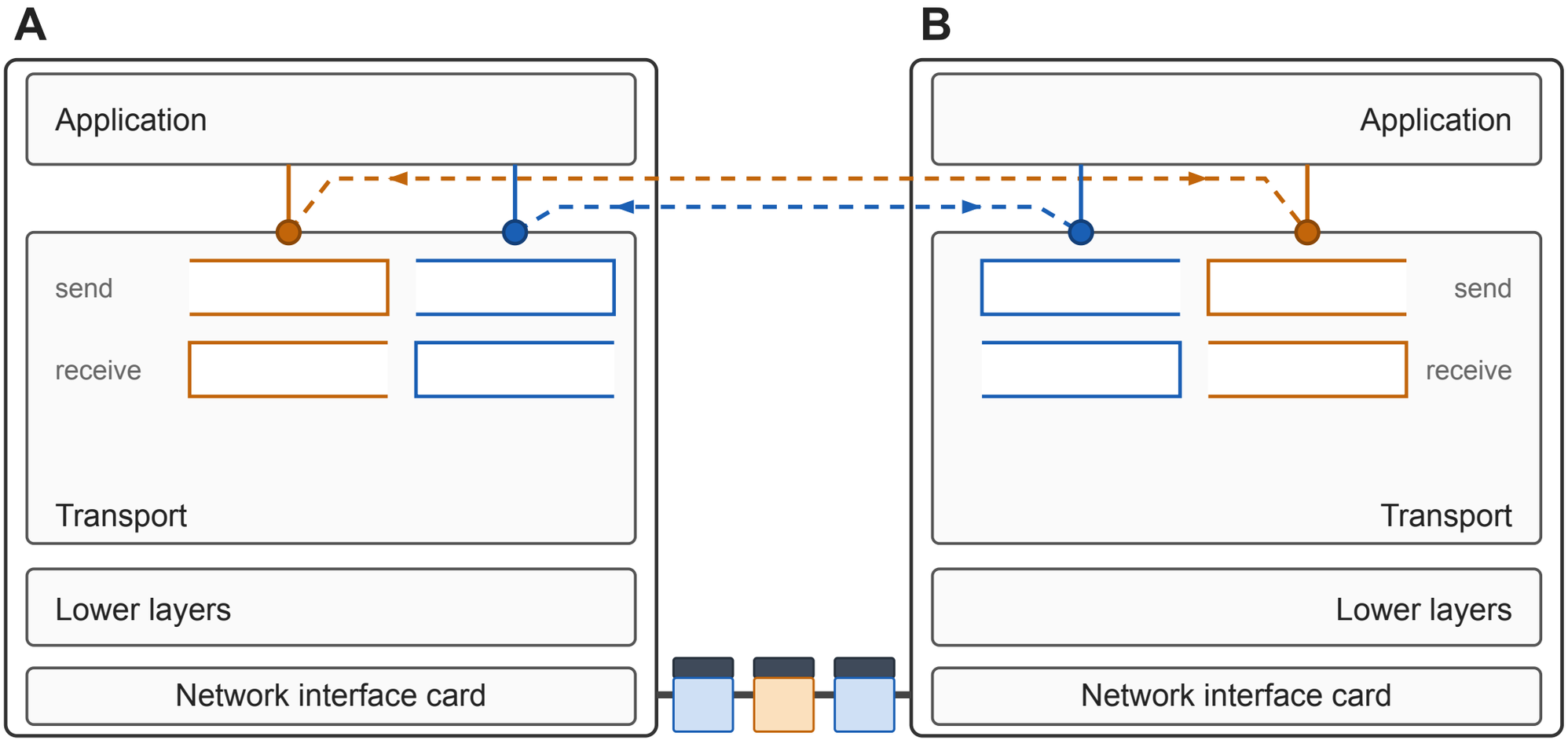
An application can open multiple channels



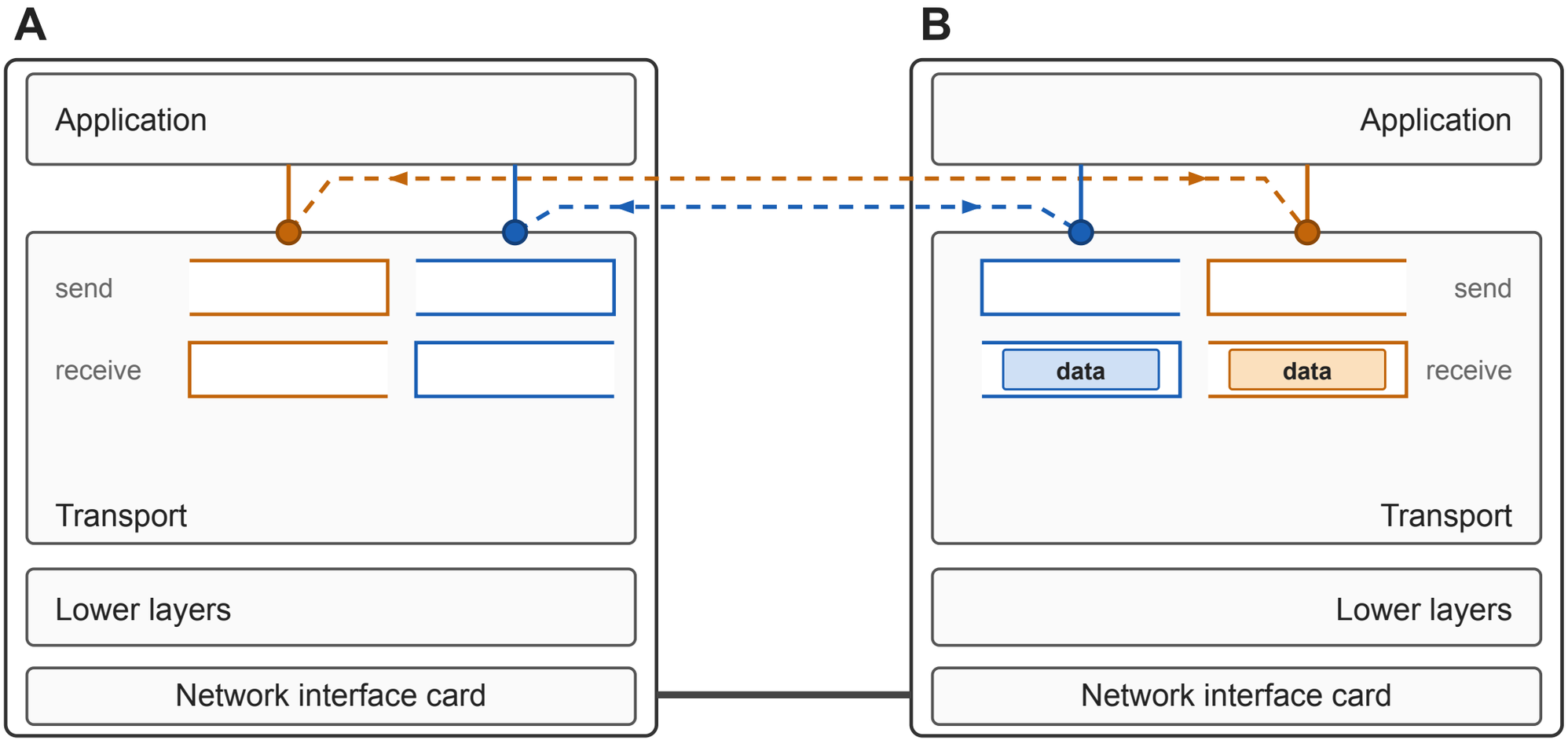
An application can open multiple channels



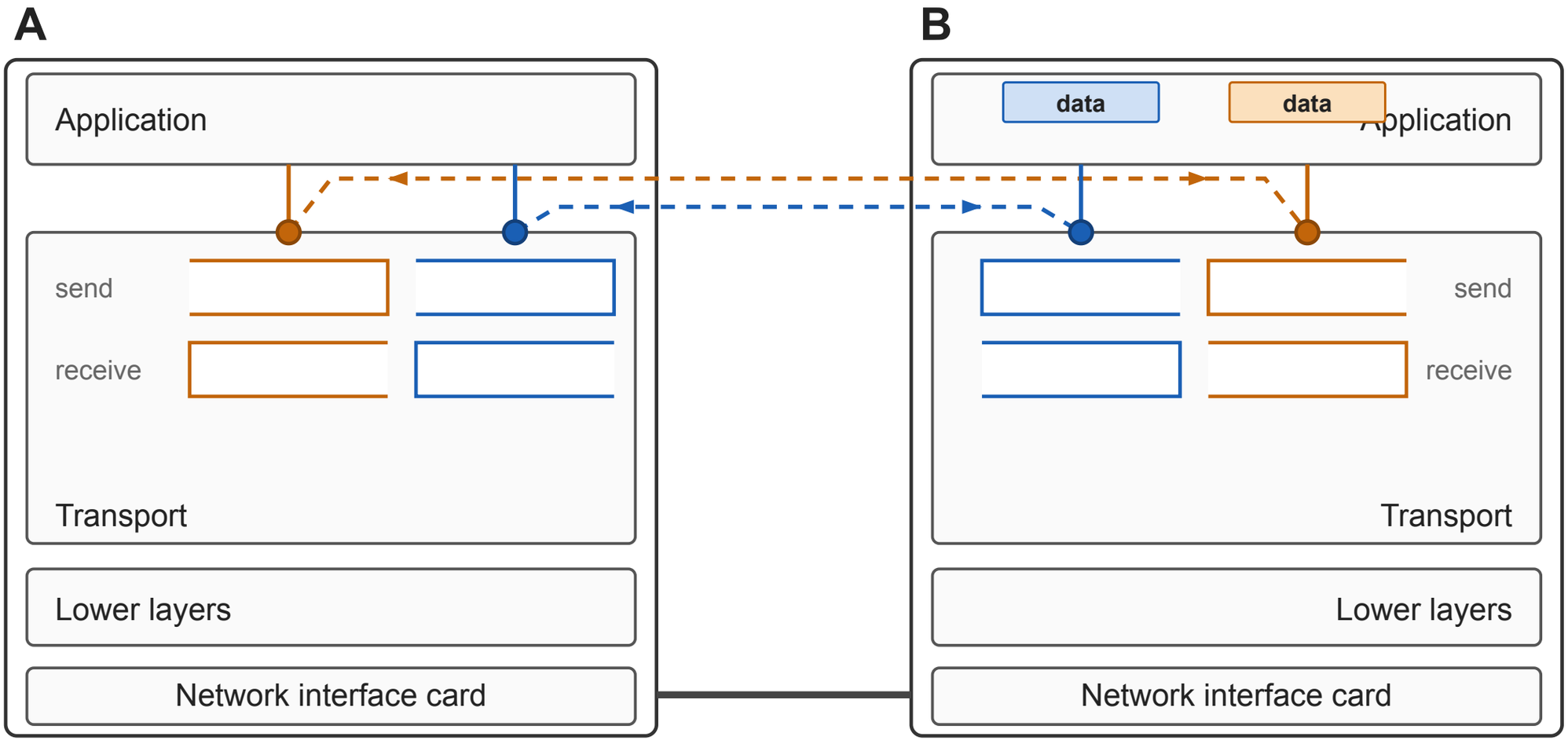
An application can open multiple channels



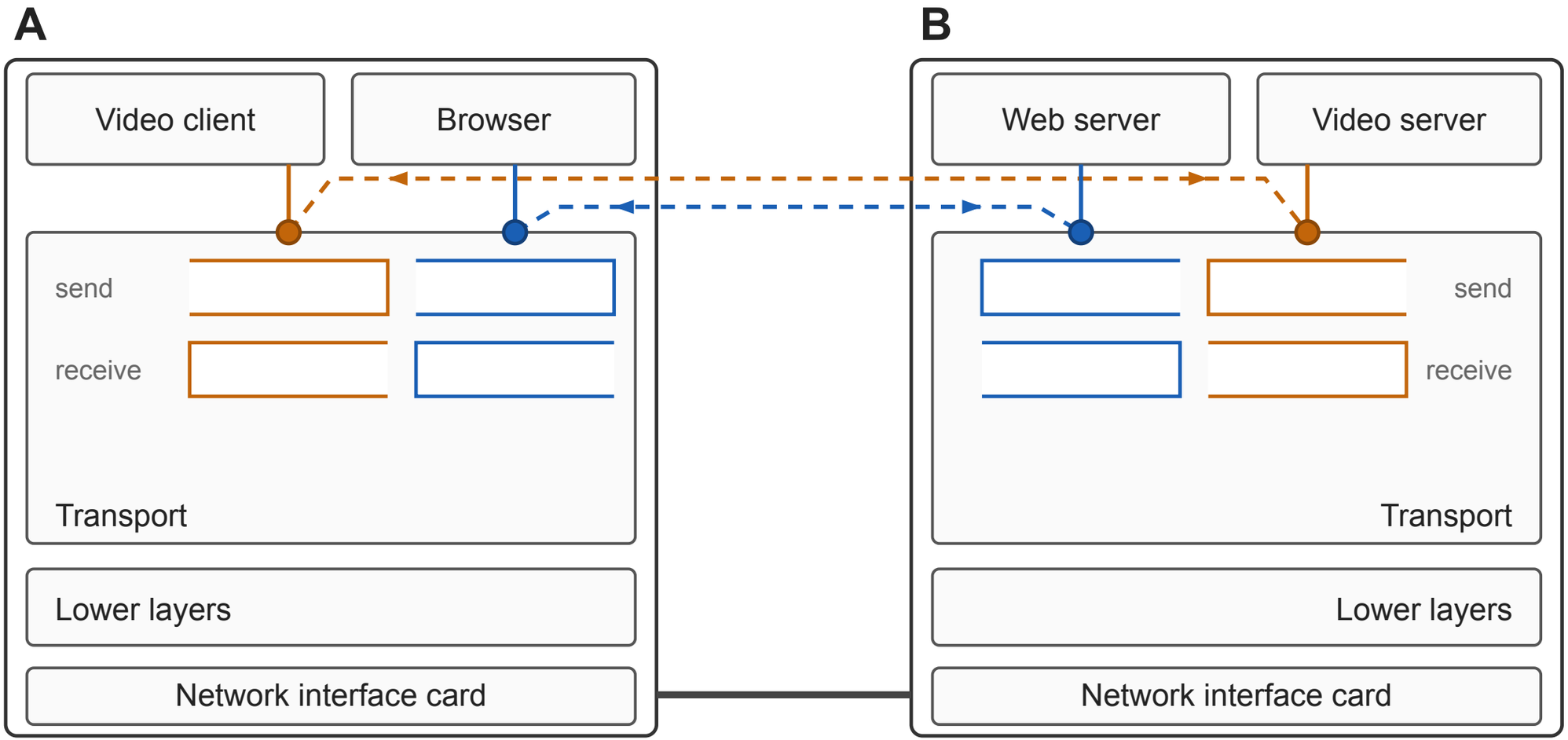
An application can open multiple channels



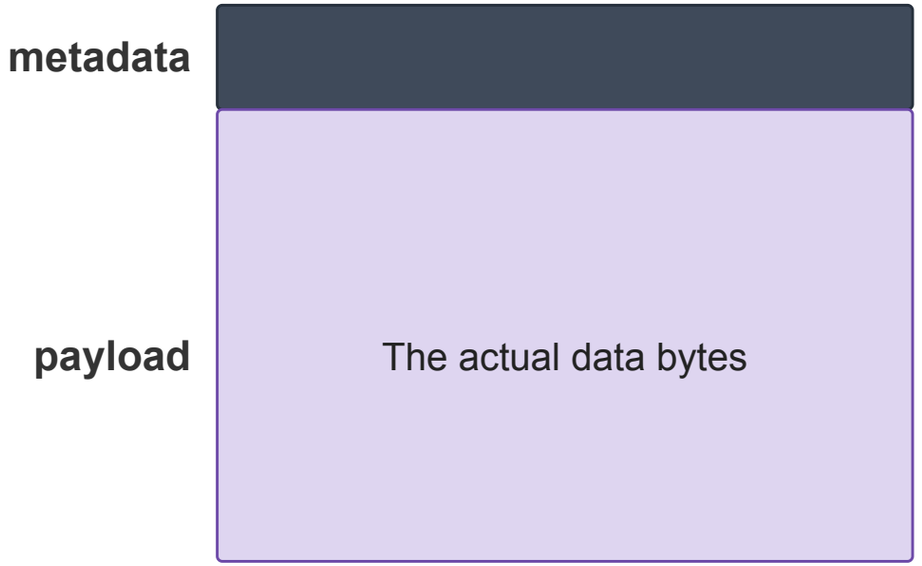
An application can open multiple channels



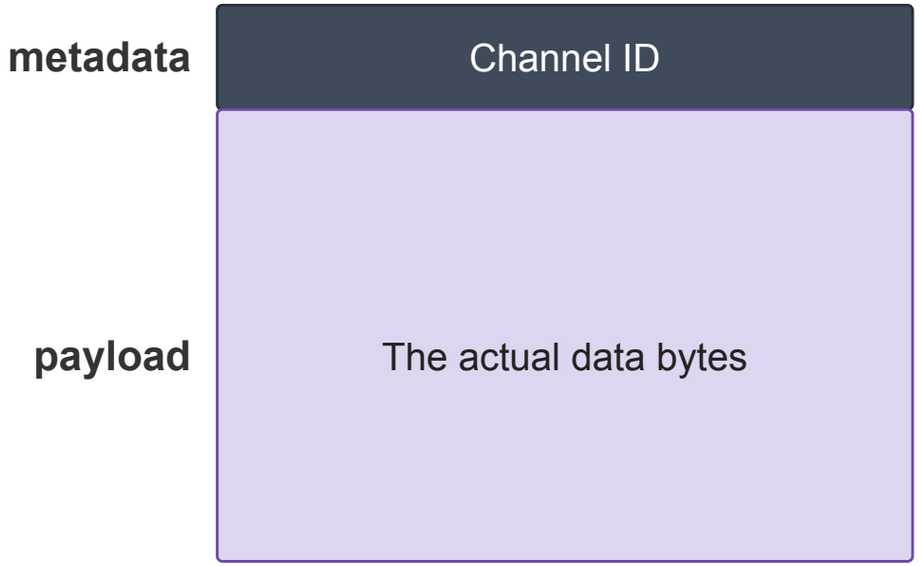
Channels can belong to different applications



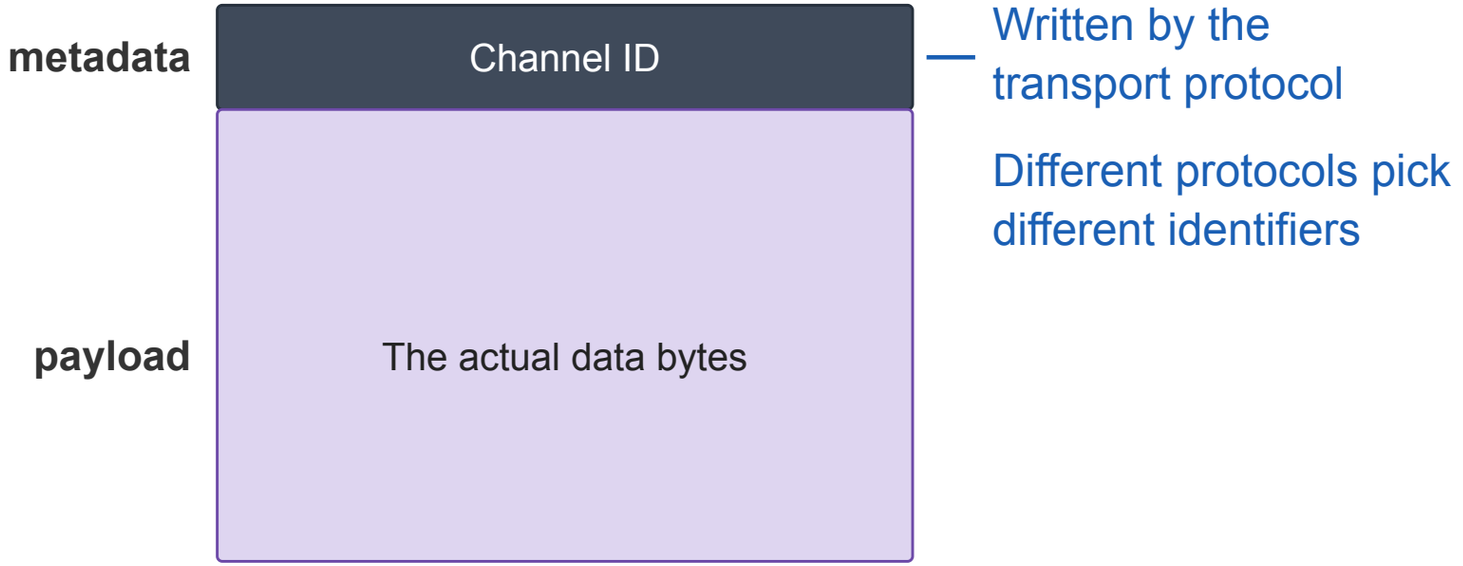
Transport metadata so far



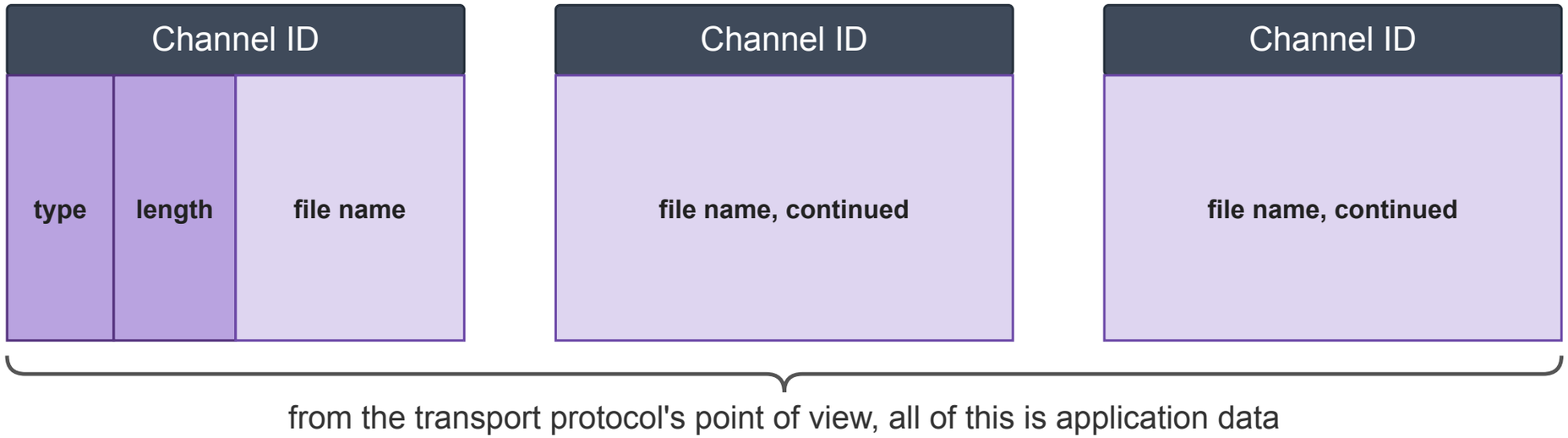
Transport metadata so far



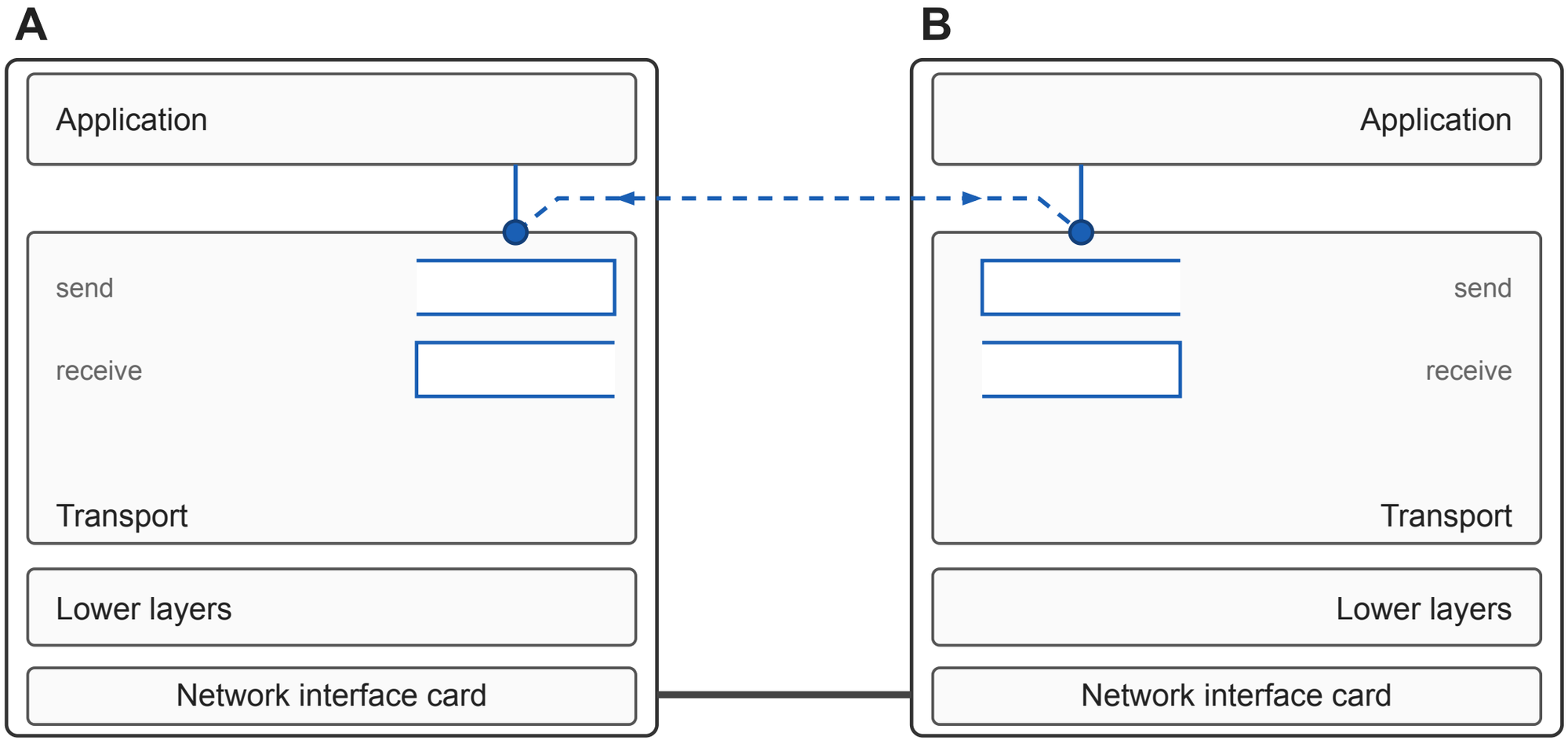
Transport metadata so far



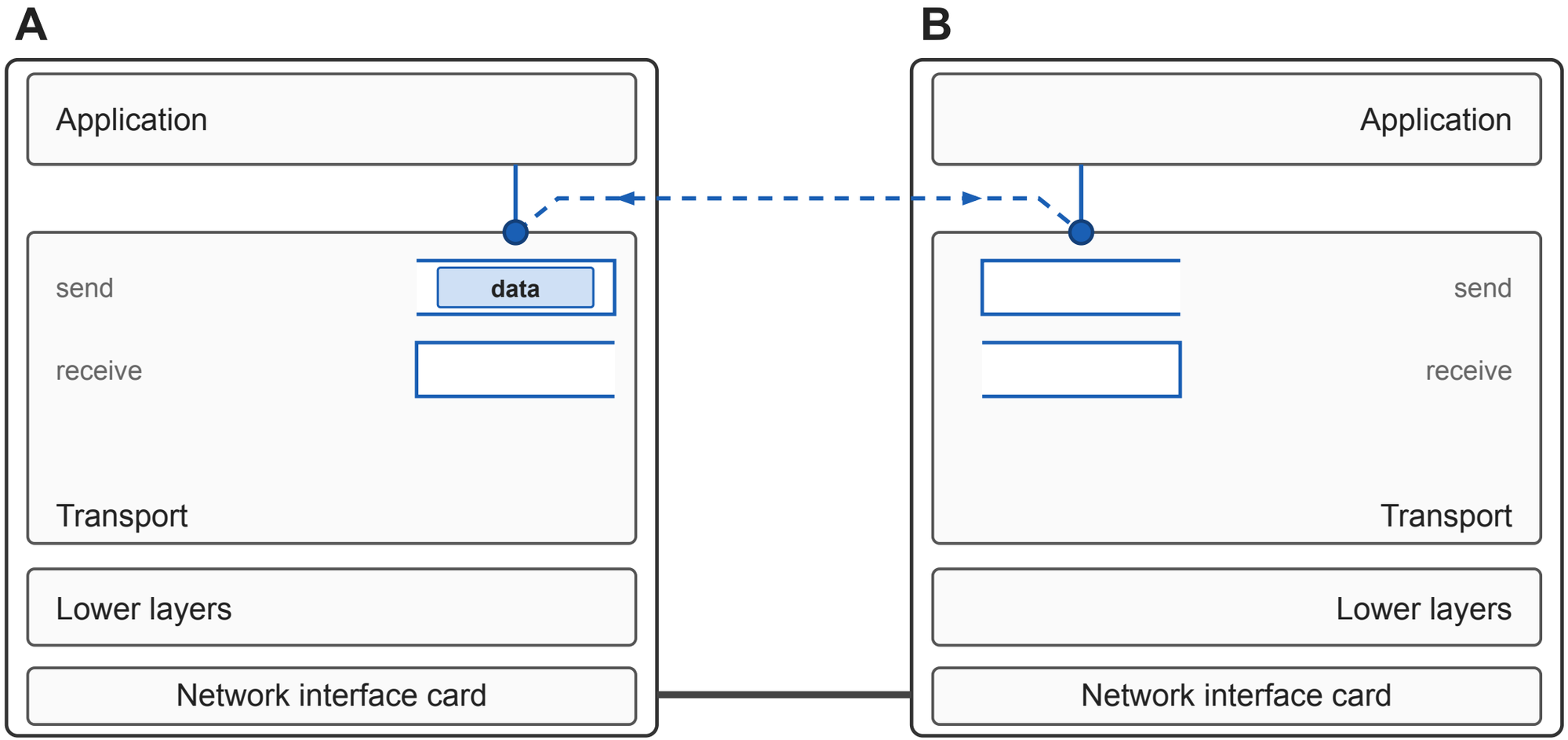
Transport metadata so far



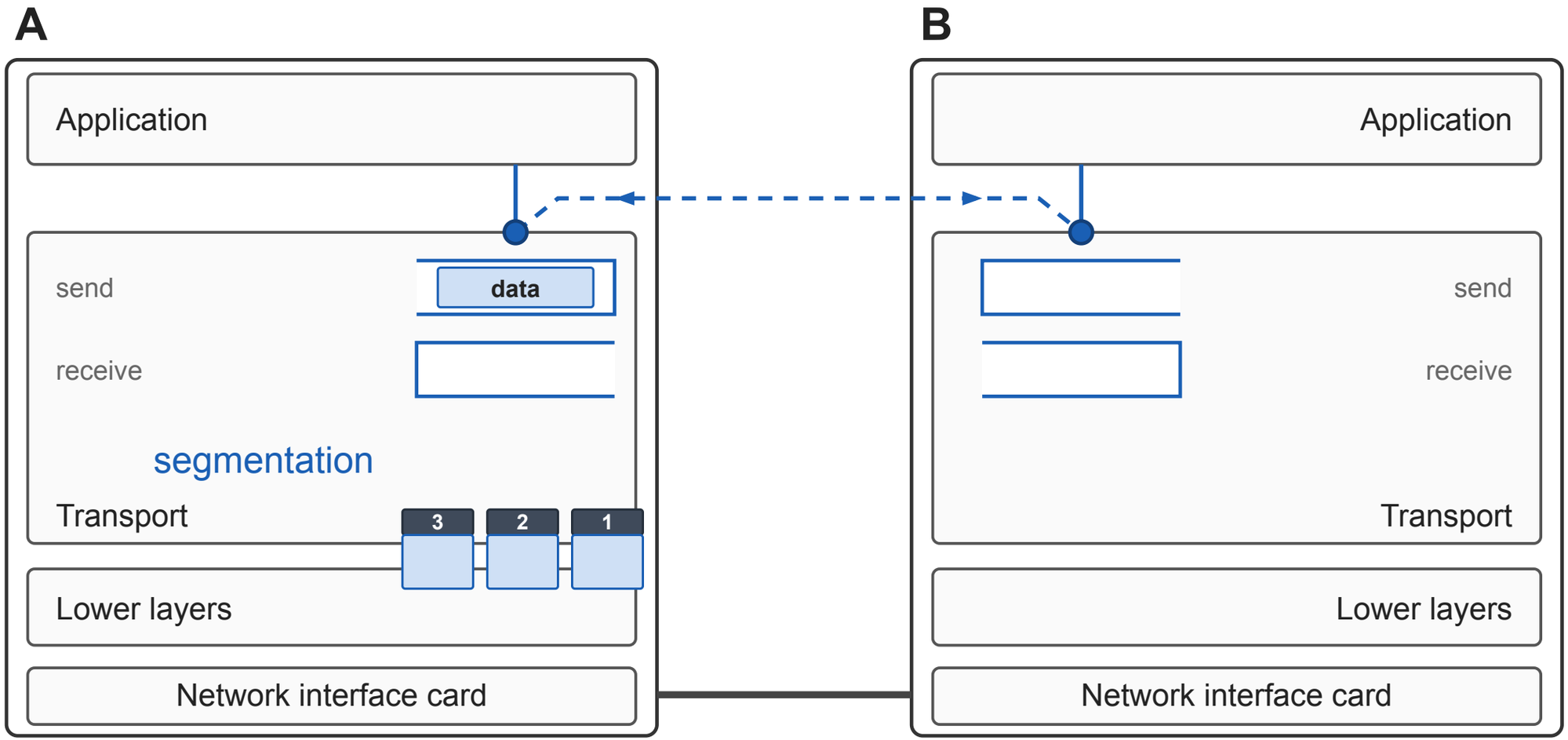
Cutting it up, and putting it back together



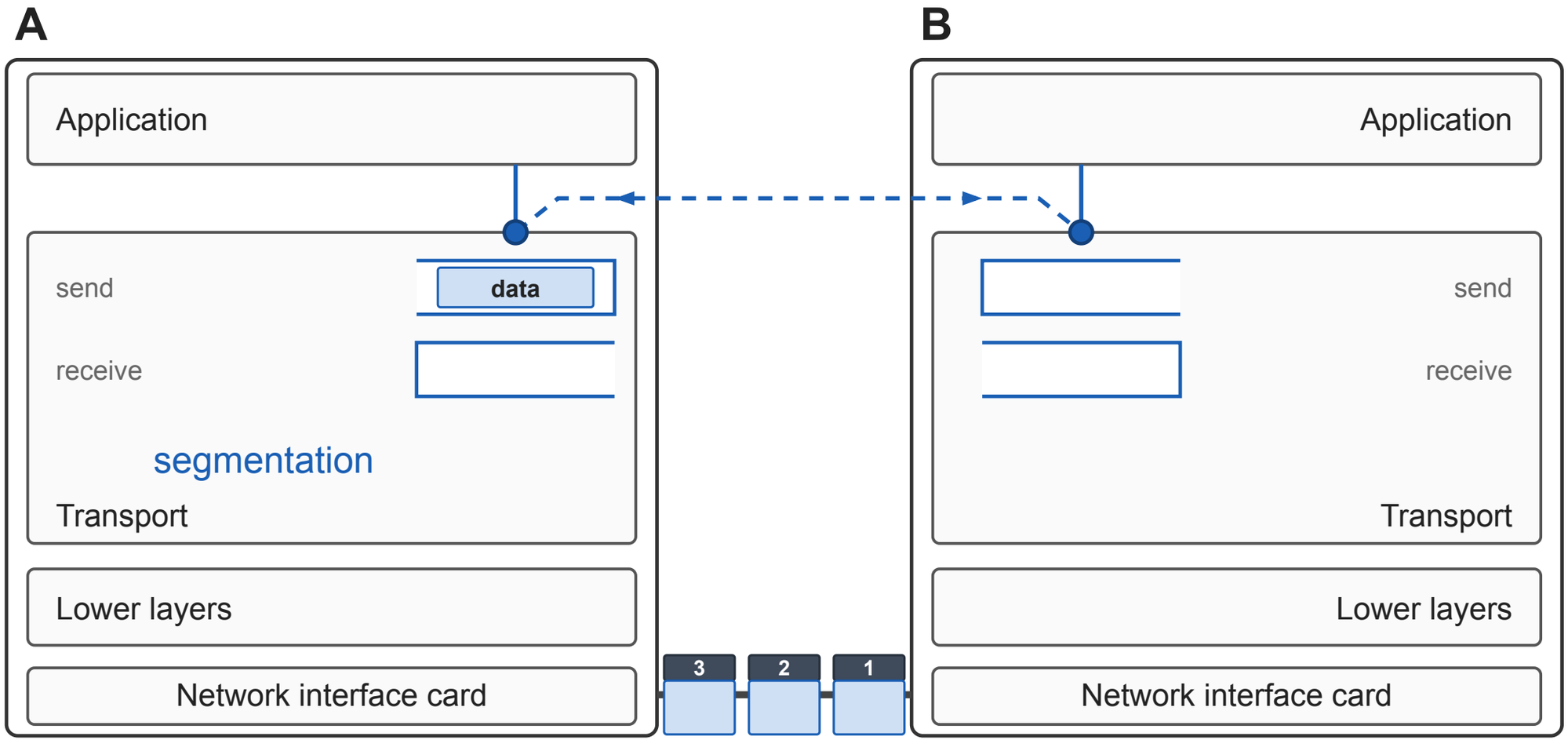
Cutting it up, and putting it back together



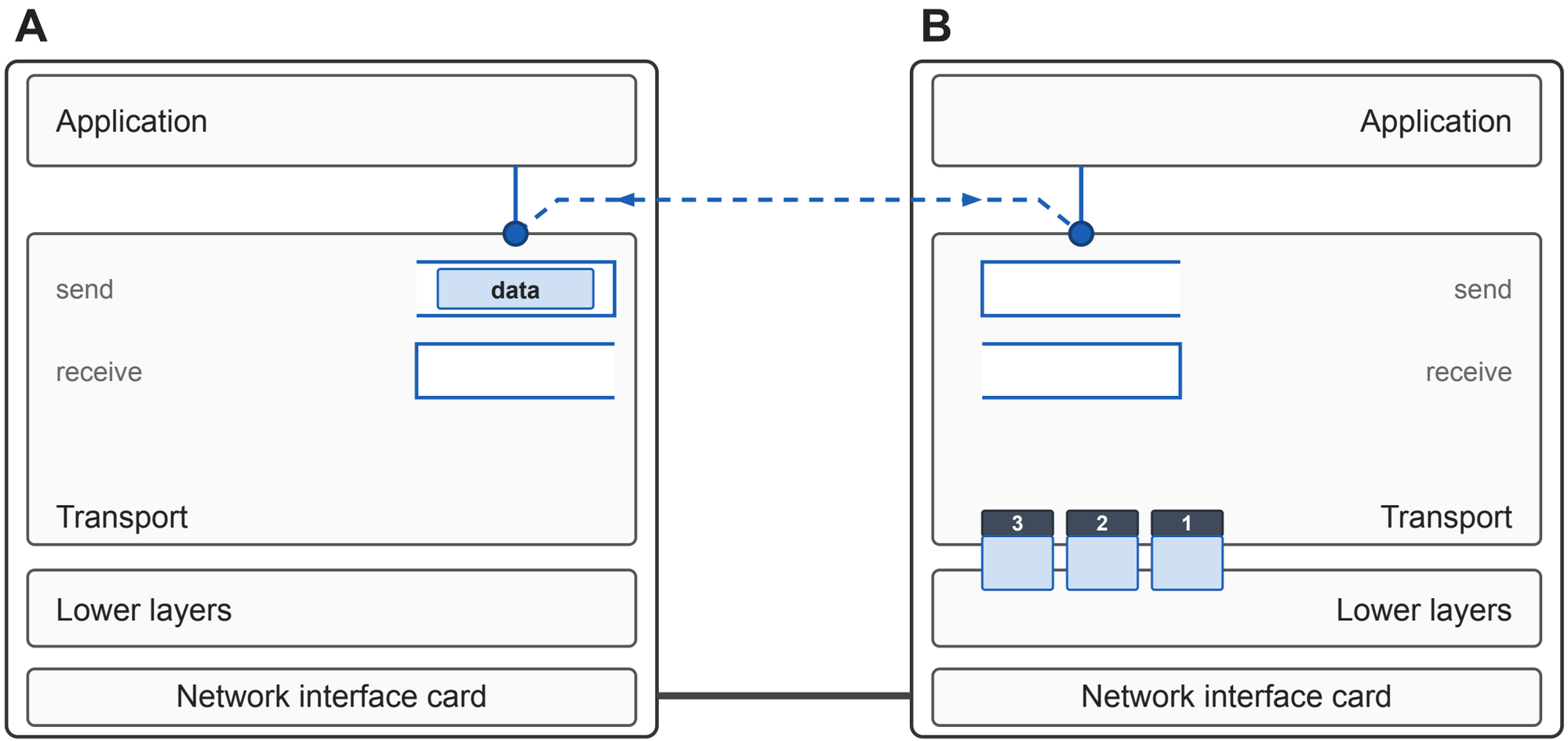
Cutting it up, and putting it back together



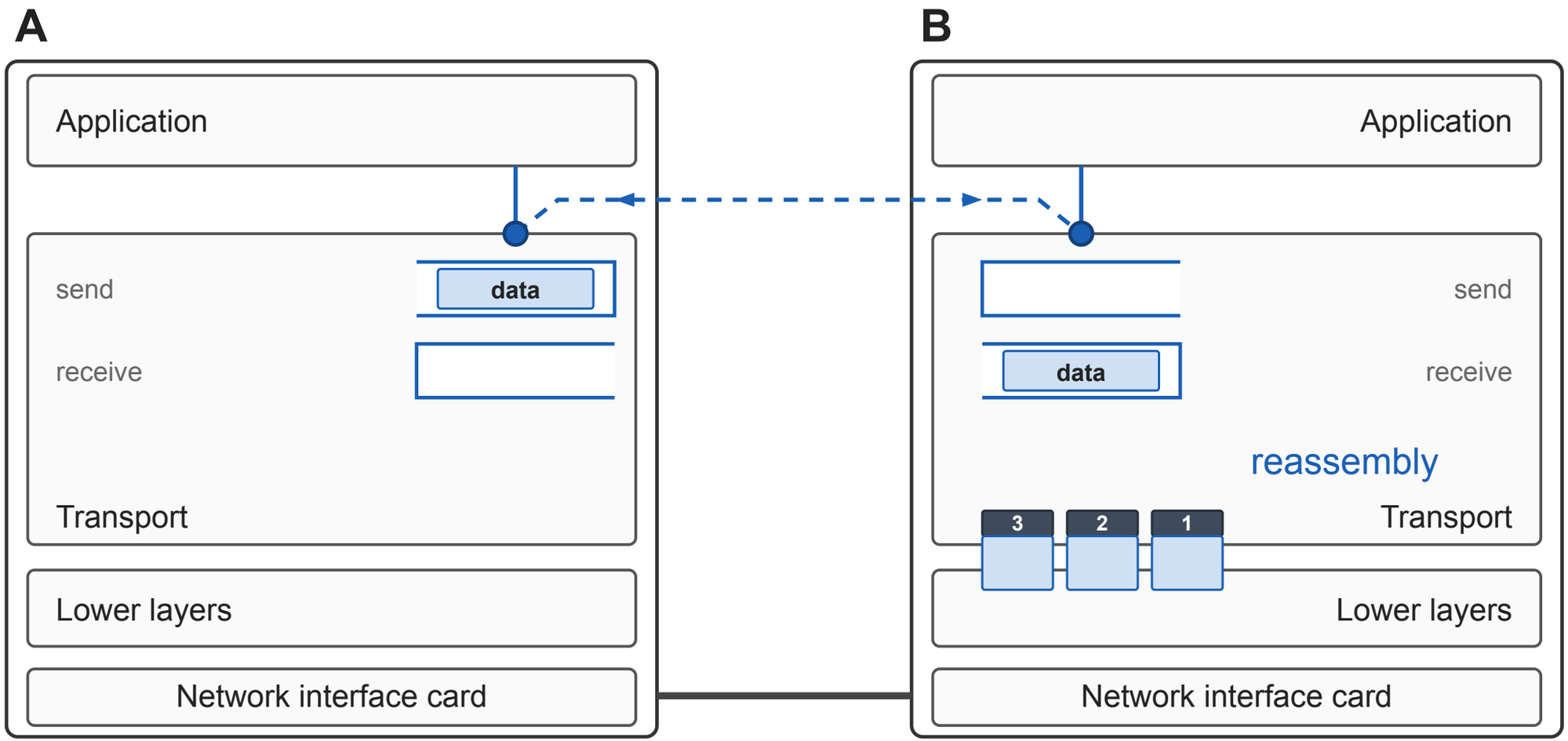
Cutting it up, and putting it back together



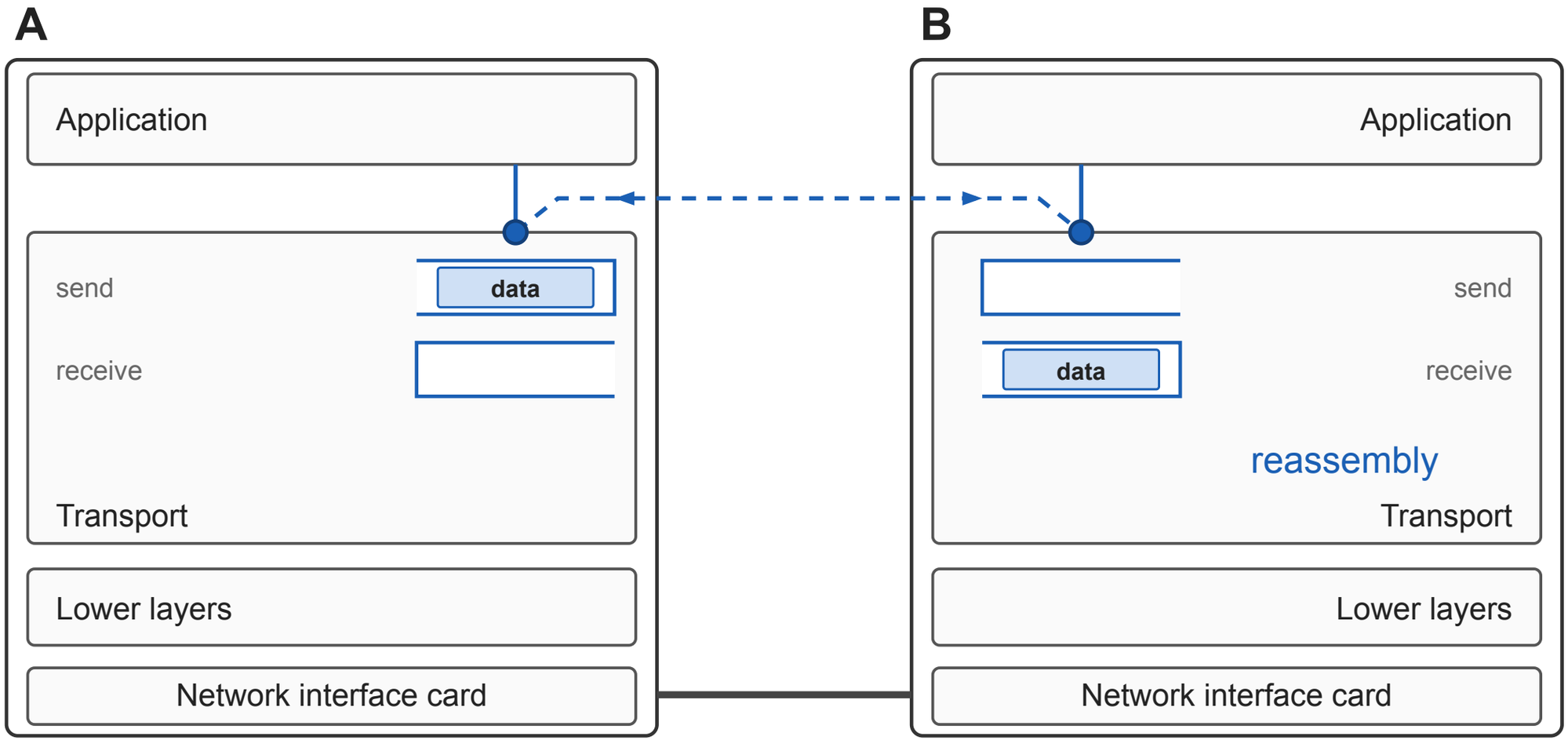
Cutting it up, and putting it back together



Cutting it up, and putting it back together

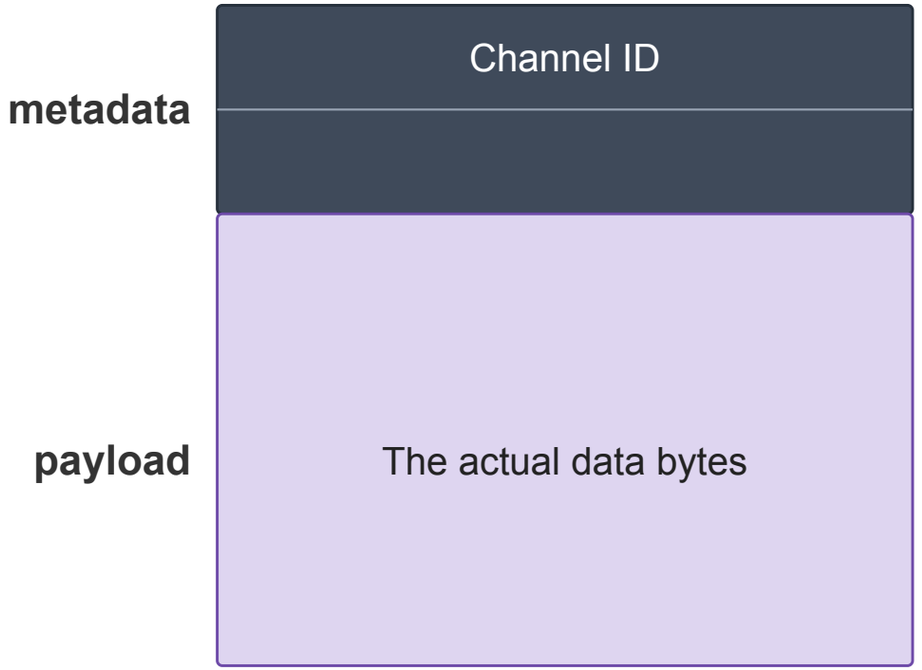


Cutting it up, and putting it back together

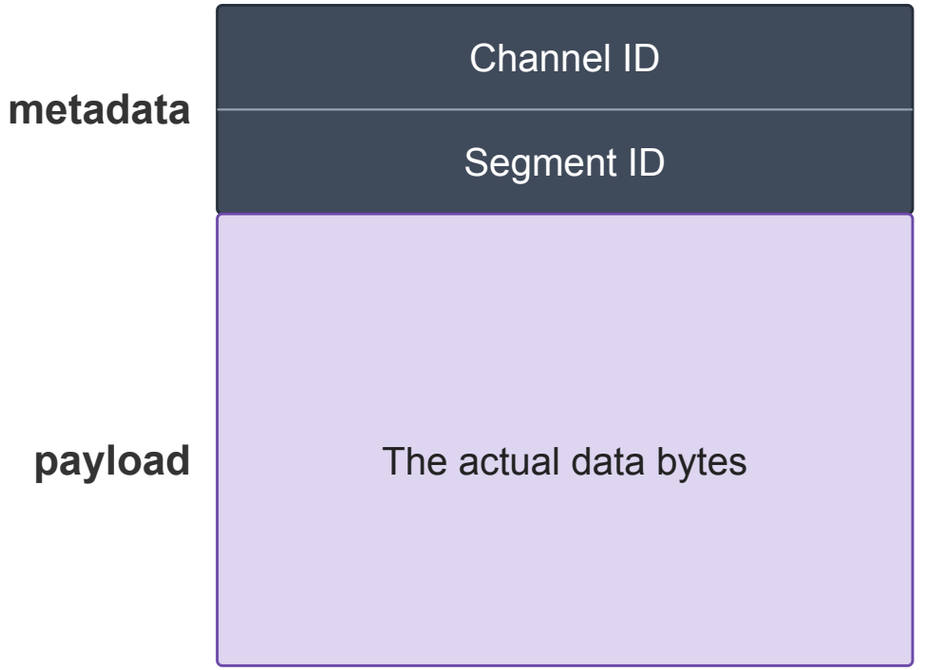


Does that change the metadata?

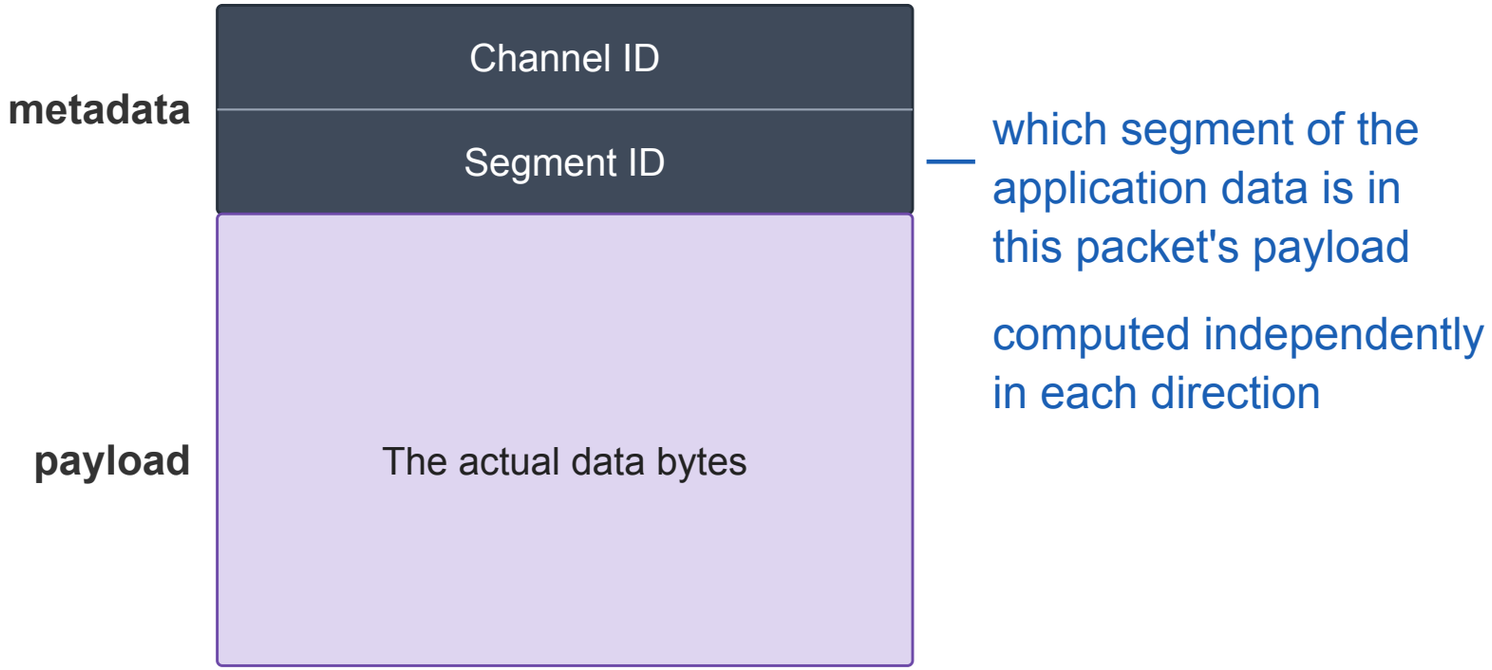
The metadata so far



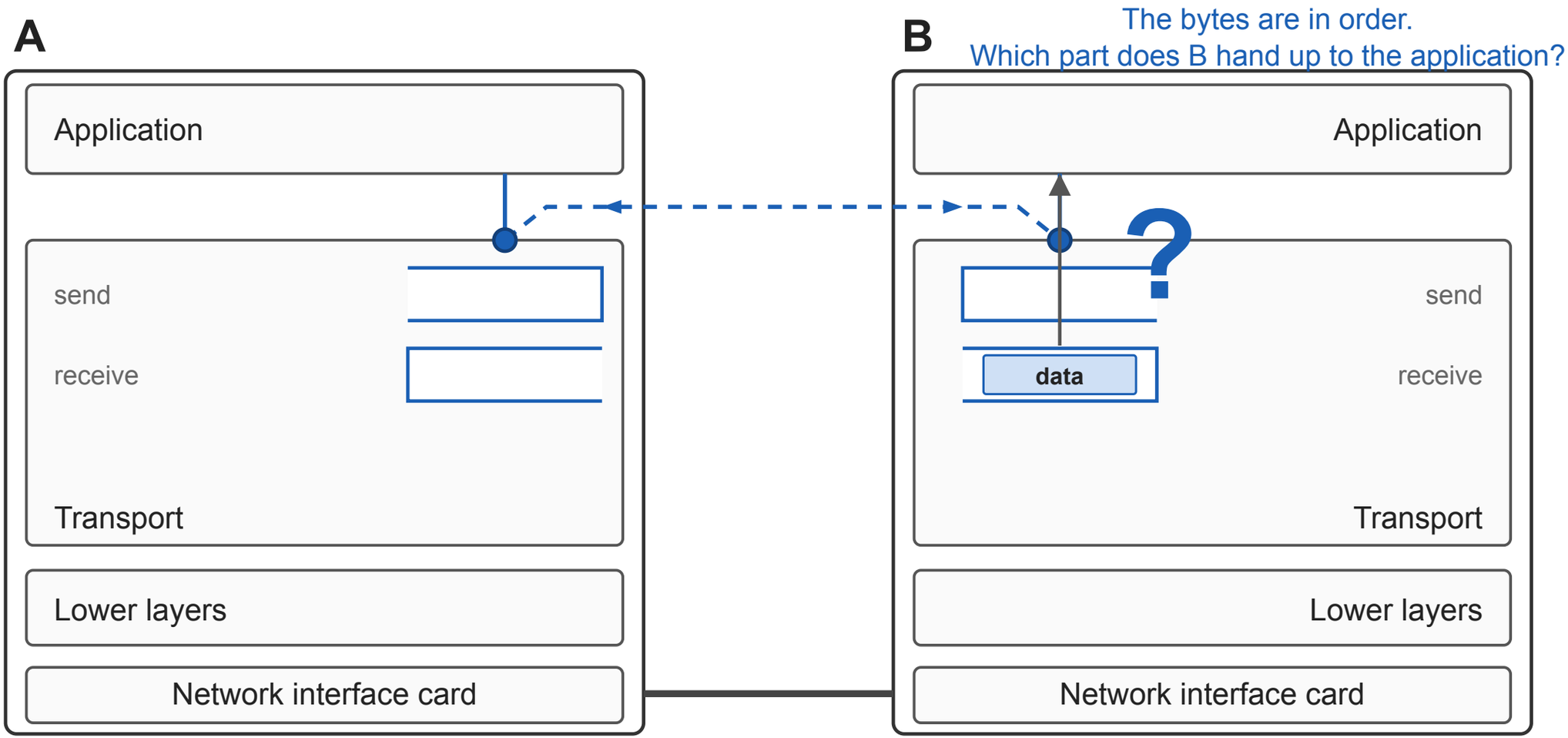
The metadata so far



The metadata so far



What comes out at B?



Which part does B hand up to the application?

The bytes are in order. Which part of them does B hand up to the application, and when?

Which part does B hand up to the application?

The bytes are in order. Which part of them does B hand up to the application, and when?

It depends on the interface that the transport channel offers the application.

Option 1: one stream of bytes (in each direction)

```
ch = open_channel(B)
net_send(ch, bytes, n)
net_recv(ch, bytes, n)
```

- The same two calls as in lecture 1. We just added the channel ID.
- The application can count on:
  - the bytes come out on the other end in the order they were sent
- Which part does B hand up to the application?
  - the bytes it has received, independent of message boundaries

Option 1: one stream of bytes (in each direction)

```
ch = open_channel(B)
net_send(ch, bytes, n)
net_recv(ch, bytes, n)
```

- The same two calls as in lecture 1. We just added the channel ID.
- The application can count on:
  - the bytes come out on the other end in the order they were sent
- Which part does B hand up to the application?
  - the bytes it has received, independent of message boundaries

- The application can not count on the message boundaries being preserved

A stream-based interface does not preserve message boundaries

A's application calls

```
net_send(ch, "URL-REQ 6 a.html", 16)
net_send(ch, "URL-REQ 6 b.html", 16)
```

B's application calls

```
net_recv(ch, bytes, 20)   ->   bytes: "URL-REQ 6 a.htmlURL-"
net_recv(ch, bytes, 20)   ->   bytes: "REQ 6 b.html"
```

A stream-based interface does not preserve message boundaries

A's application calls

```
net_send(ch, "URL-REQ 6 a.html", 16)
net_send(ch, "URL-REQ 6 b.html", 16)
```

B's application calls

```
net_recv(ch, bytes, 20)   ->   bytes: "URL-REQ 6 a.htmlURL-"
net_recv(ch, bytes, 20)   ->   bytes: "REQ 6 b.html"
```

The transport layer does not know about or preserve message boundaries

Option 2: several streams in one channel (in each direction)

```
ch = open_channel(B)
s  = open_stream(ch)
net_send(ch, s, bytes, n)
net_recv(ch, s, bytes, n)
```

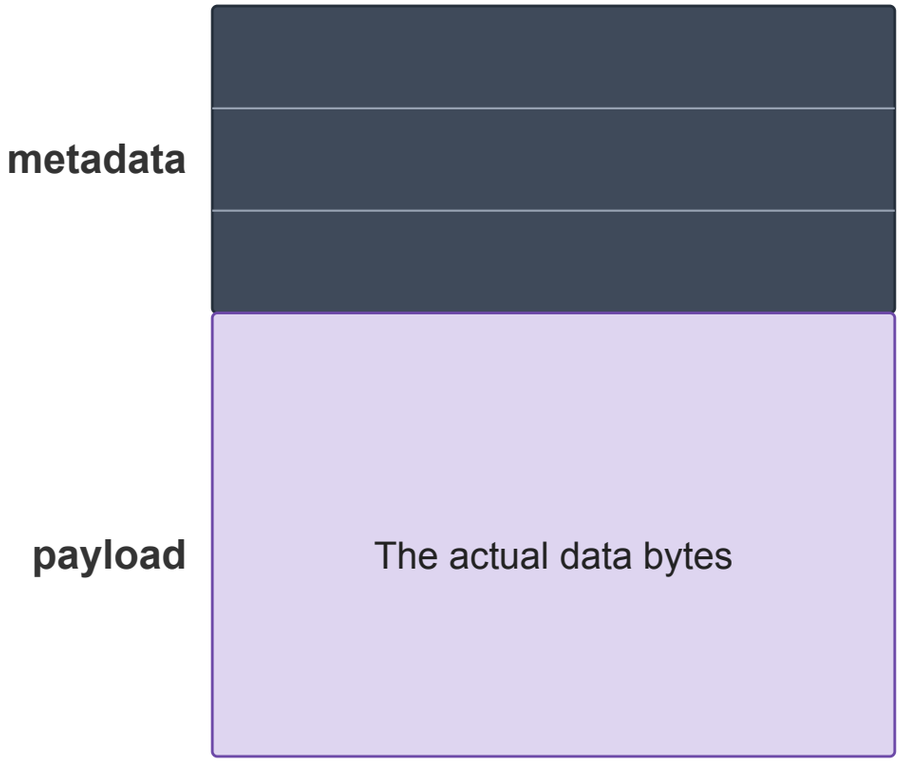
- This is one of the fixes we discussed for head-of-line blocking (lecture 1)
- The application can expect:
  - within one stream, bytes come out on the other end in the order they were sent
  - bytes can be delivered out of order across streams

Option 2: several streams in one channel (in each direction)

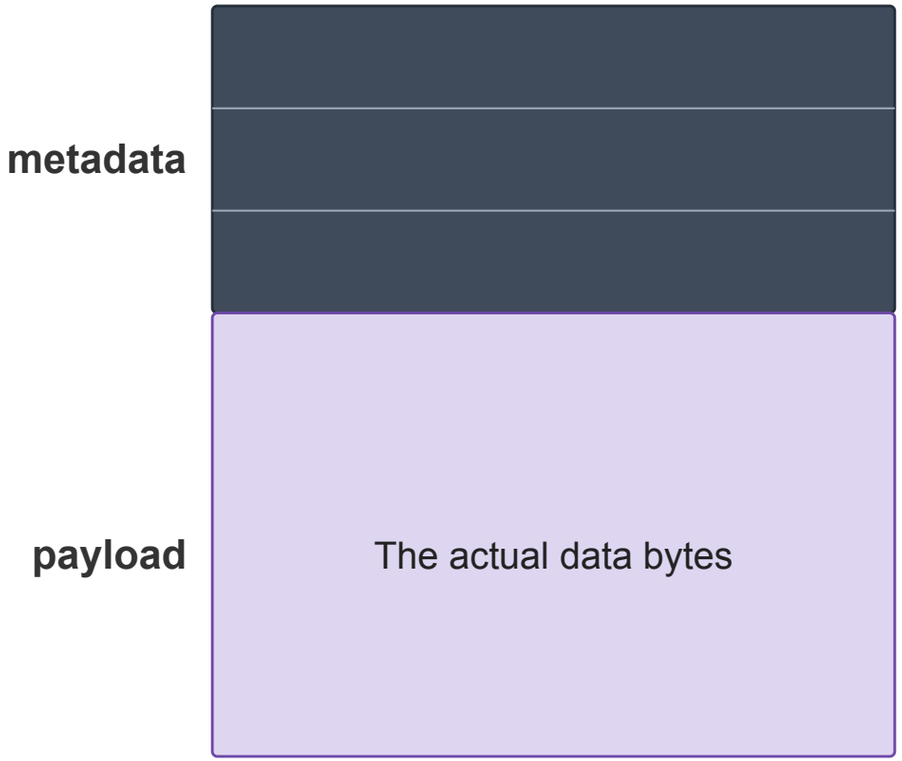
```
ch = open_channel(B)
s  = open_stream(ch)
net_send(ch, s, bytes, n)
net_recv(ch, s, bytes, n)
```

- This is one of the fixes we discussed for head-of-line blocking (lecture 1)
- The application can expect:
  - within one stream, bytes come out on the other end in the order they were sent
  - bytes can be delivered out of order across streams
- The application can not count on the message boundaries being preserved

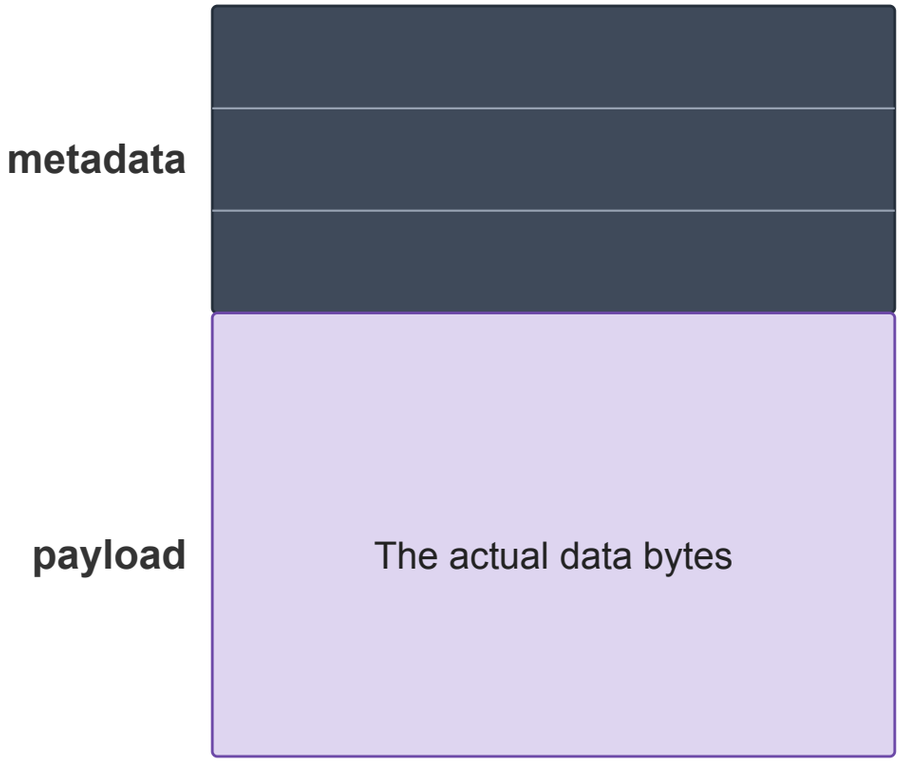
The metadata with several streams



The metadata with several streams



The metadata with several streams



Option 3: independent messages

```
ch = open_channel(B)
net_send(ch, message)
net_recv(ch, message)
```

- Each send is one message, and each receive returns one whole message
- The application can count on:
  - receiving each message as a whole, with all its bytes in order
- The application can not count on:
  - ordering being preserved between messages

A message-based interface keeps the boundaries

A’s application calls

```
net_send(ch, "URL-REQ 6 a.html")
net_send(ch, "URL-REQ 6 b.html")
```

B’s application calls

```
net_recv(ch, bytes)   ->   bytes: "URL-REQ 6 b.html"
net_recv(ch, bytes)   ->   bytes: "URL-REQ 6 a.html"
```

A message-based interface keeps the boundaries

A's application calls

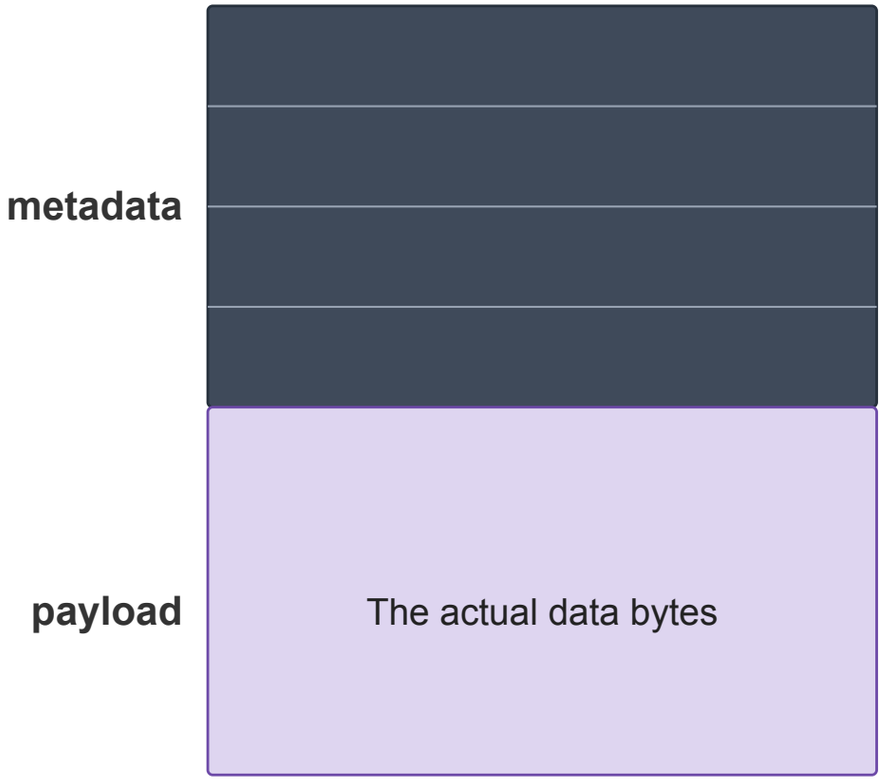
```
net_send(ch, "URL-REQ 6 a.html")
net_send(ch, "URL-REQ 6 b.html")
```

B's application calls

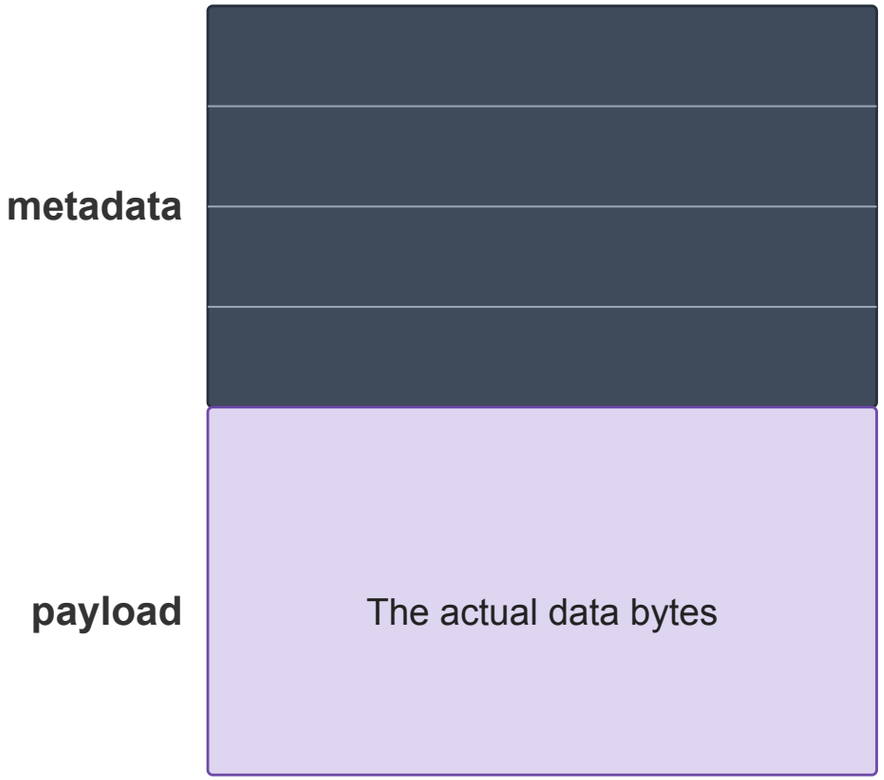
```
net_recv(ch, bytes)   ->   bytes: "URL-REQ 6 b.html"
net_recv(ch, bytes)   ->   bytes: "URL-REQ 6 a.html"
```

Each receive returns exactly one whole message, but not always in the order they were sent

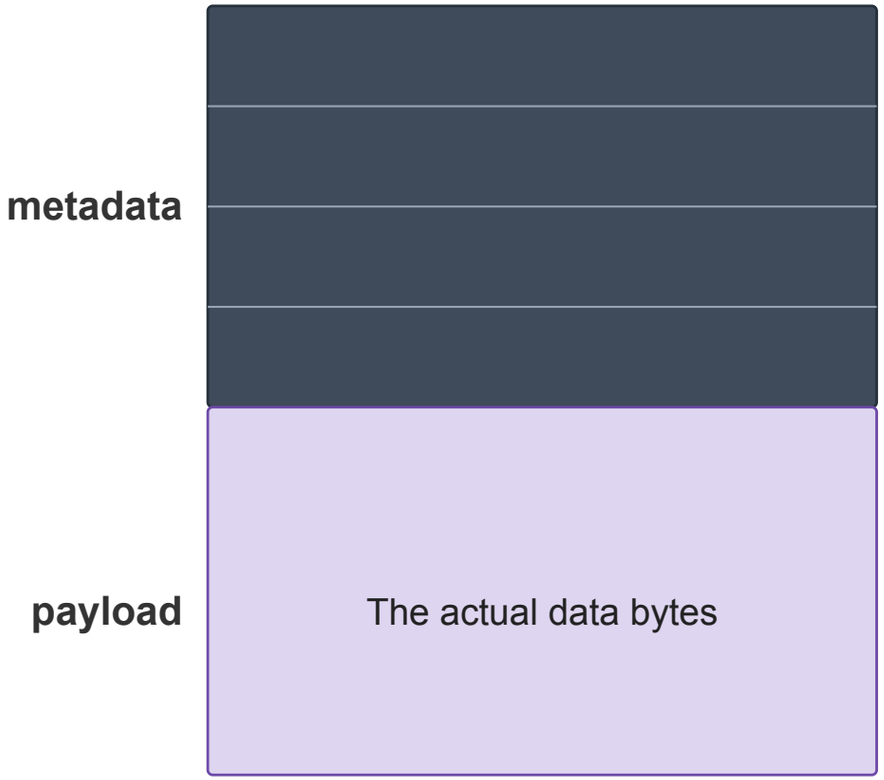
The metadata with messages



The metadata with messages



The metadata with messages



Interface options we have discussed so far

1. One stream of bytes, in each direction
2. Several streams in one channel, in each direction
3. Independent messages
4. ?

Which one would you want, and for what?
Any other options?

Reset

What we assume from here on

For the rest of the course, we mostly assume the channel exposes one stream to the application in each direction (option 1).

What does the transport layer have to take care of?

1. Get the data to the right application on the destination computer ✓
2. Segmentation and reassembly ✓
3. **next lecture** → What to do when the lower layers lose, corrupt, or slow down packets