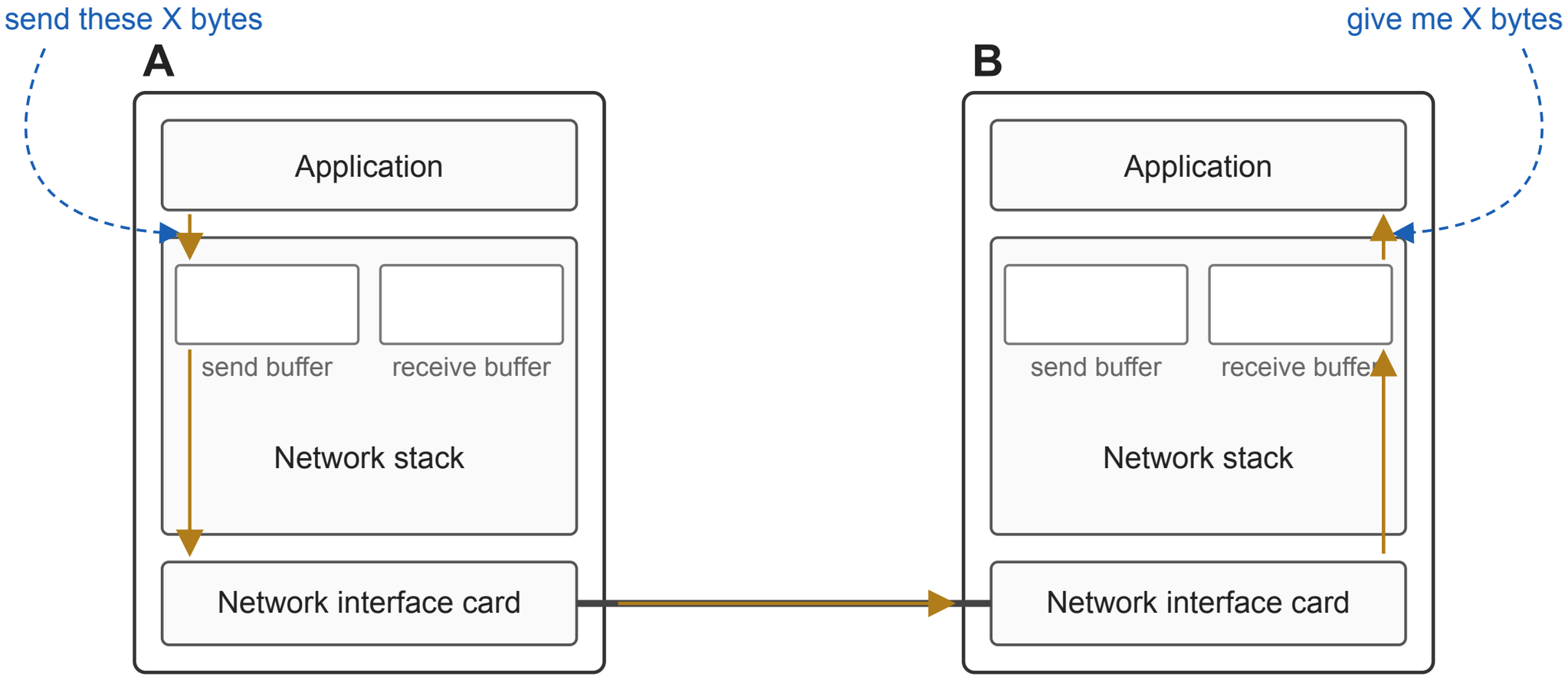
UNIVERSITY OF WATERLOO

Building networked applications

CS 456/656, Fall 2026

Mina Tahmasbi Arashloo

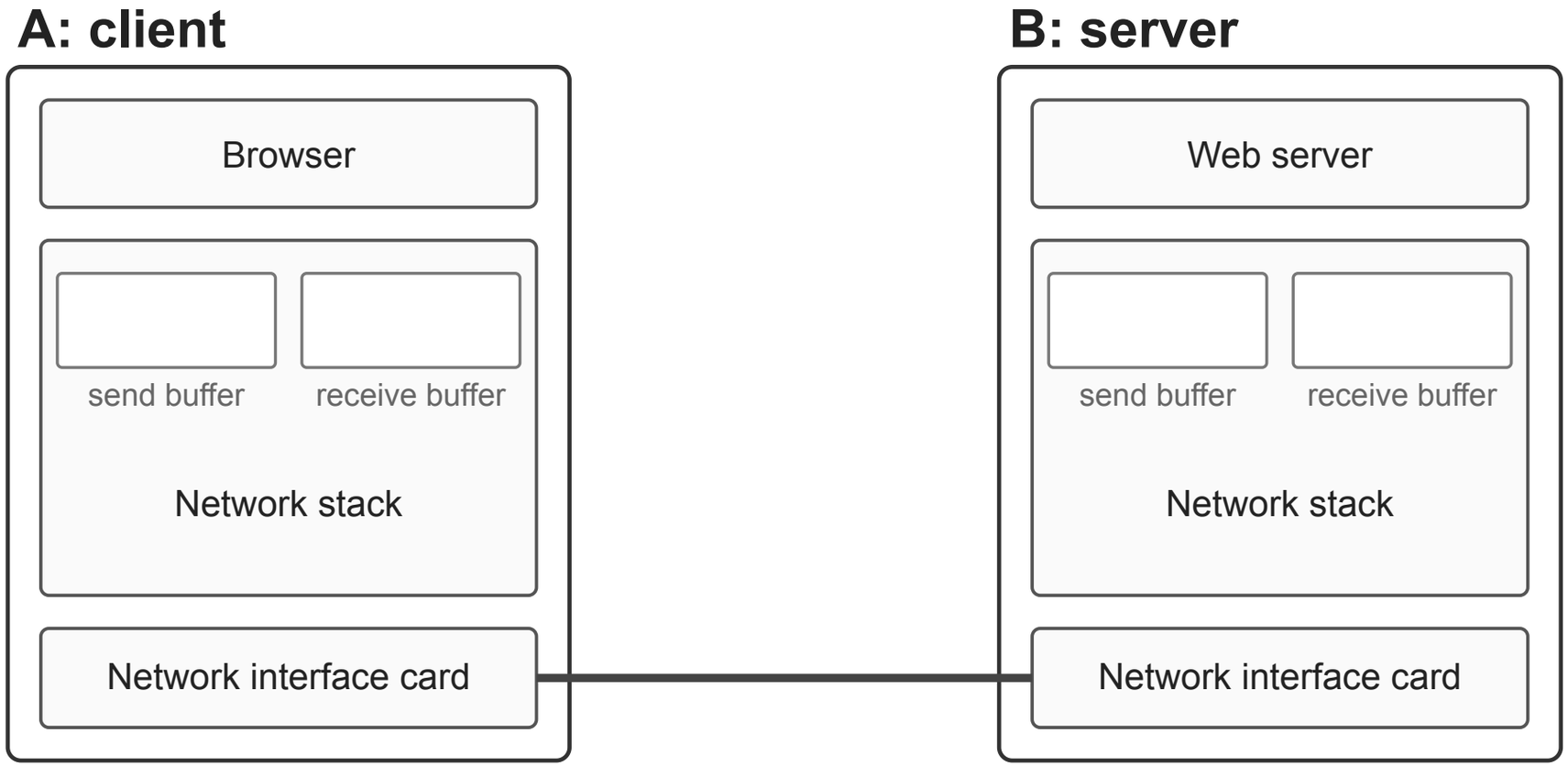
Last time: A wants to send data to B



Outline

- **Building a web browser**
- How long does it take to get a file?
- Pages with more than one file
- Takeaways

Let’s build a web browser



What you have

Browser (A)

```
void net_send(uint8 *bytes, int n);
void net_recv(uint8 *bytes, int n);

int main() {
    char *name = read_filename();



    display(page);
}
```

Server (B)

```
void net_send(uint8 *bytes, int n);
void net_recv(uint8 *bytes, int n);

int main() {

}
```

What we are trying to build

- The server has some `.html` files
- The client asks for a file by its name
- The server answers with the file

```
1 <html>
2   <body>
3     <h1>CS 456</h1>
4     <p>Welcome!</p>
5   </body>
6 </html>
```

An HTML file is a text file. The browser reads it and draws the page.

Fill it in

Browser (A)

```
void net_send(uint8 *bytes, int n);
void net_recv(uint8 *bytes, int n);

int main() {
    char *name = read_filename();

    net_send(len(name), 4);
    net_send(name, len(name));

    char* page_len_str;
    net_recv(page_len_str, 4);
    int page_len = int(page_len_str);
    char* page[page_len];
    rcvd = net_recv(page, page_len);

    display(page);
}
```

Server (B)

```
void net_send(uint8 *bytes, int n);
void net_recv(uint8 *bytes, int n);

int main() {
    char* fname_len_str;
    net_recv(fname_len, 4);
    int fname_len = int(fname_len_str);
    char* fname[frame_len];
    net_recv(fname, frame_len);

    char* page = get_file(fname)
    net_send(len(page), 4);
    net_send(page, page_len);
}
```

This is an example of a protocol

A protocol defines:

- the format and order of messages sent and received among network entities, and
- actions taken on message transmission and receipt

It has to be agreed upon **beforehand**

The protocol between A and B

- send the size (4 bytes) then the content
- If you are a server, the content you receive is a filename, look it up, send it back.

What else should they agree on?

Would the server always answer the way the client expects?

Browser (A)

```
void net_send(uint8 *bytes, int n);
void net_recv(uint8 *bytes, int n);

int main() {
    char *name = read_filename();

    net_send(len(name), 4);
    net_send(name, len(name));

    char* page_len_str;
    net_recv(page_len_str, 4);
    int page_len = int(page_len_str);
    if (page_len <= 0){
       display("not found");
    }
    else {
       char* page[page_len];
       rcvd = net_recv(page, page_len);

       display(page);
    }
}
```

Server (B)

```
void net_send(uint8 *bytes, int n);
void net_recv(uint8 *bytes, int n);

int main() {
    char* fname_len_str;
    net_recv(fname_len, 4);
    int fname_len = int(fname_len_str);
    char* fname[frame_len];
    net_recv(fname, frame_len);

    char* page = get_file(fname)
    if (!page){
        net_send(-1, 4);
    }
    else{
        net_send(len(page), 4);
        net_send(page, page_len);
    }
}
```

The protocol between A and B

- send the size (4 bytes) then the content
- If you are a server, the content you receive is a filename, look it up, send it back.
- if the file does not exist, just send one -1 in 4 bytes.

What else should they agree on?

What if A and B can act as both client and server?

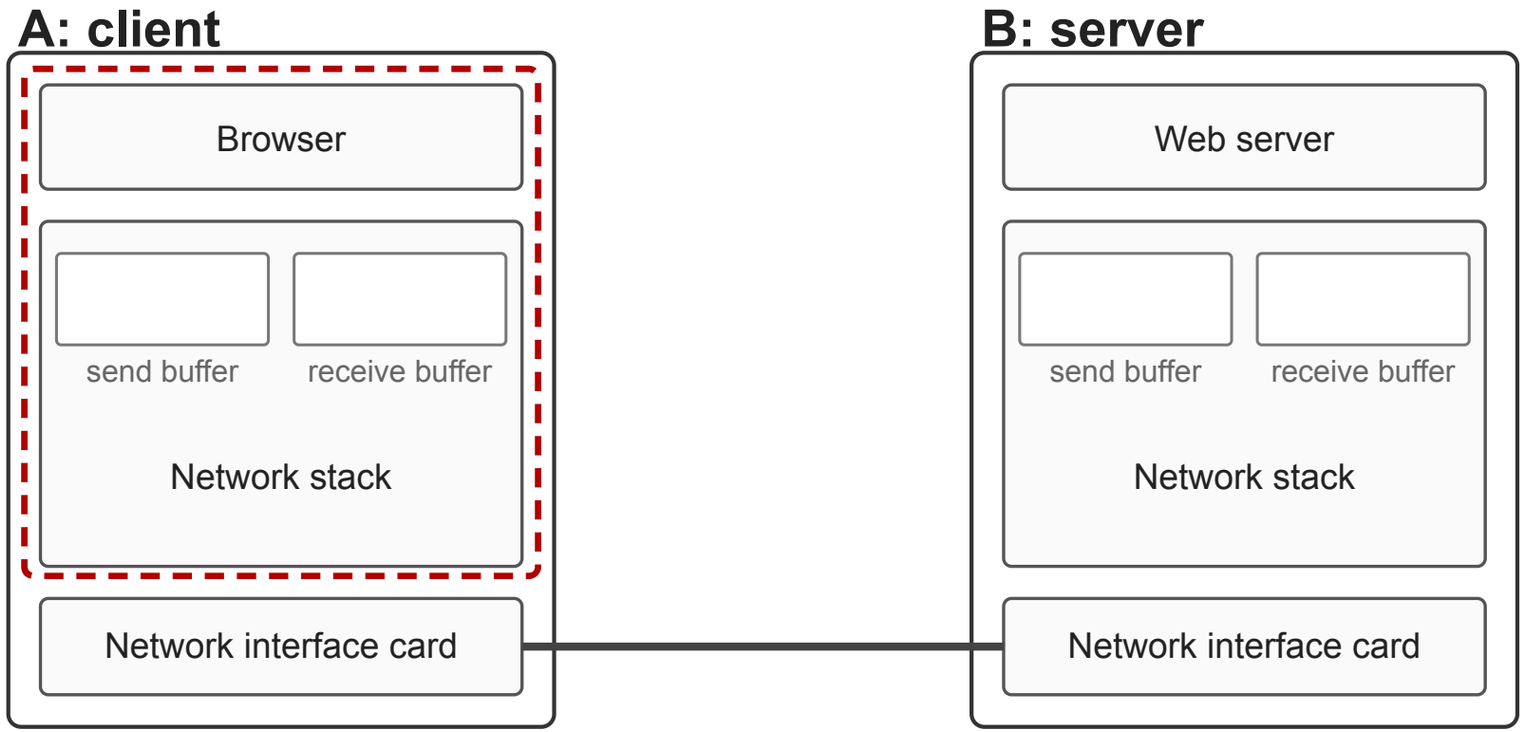
The protocol between A and B

- send message type (1 byte), then the size (4 bytes) then the content
- If you are a server, the content you receive is a filename, look it up, send it back.
- if the file does not exist, just send one -1 in 4 bytes.
- if what you are sending is a filename, use message type 1, if it is a file, use message type 0.

Outline

- Building a web browser
- **How long does it take to get a file?**
- Pages with more than one file
- Takeaways

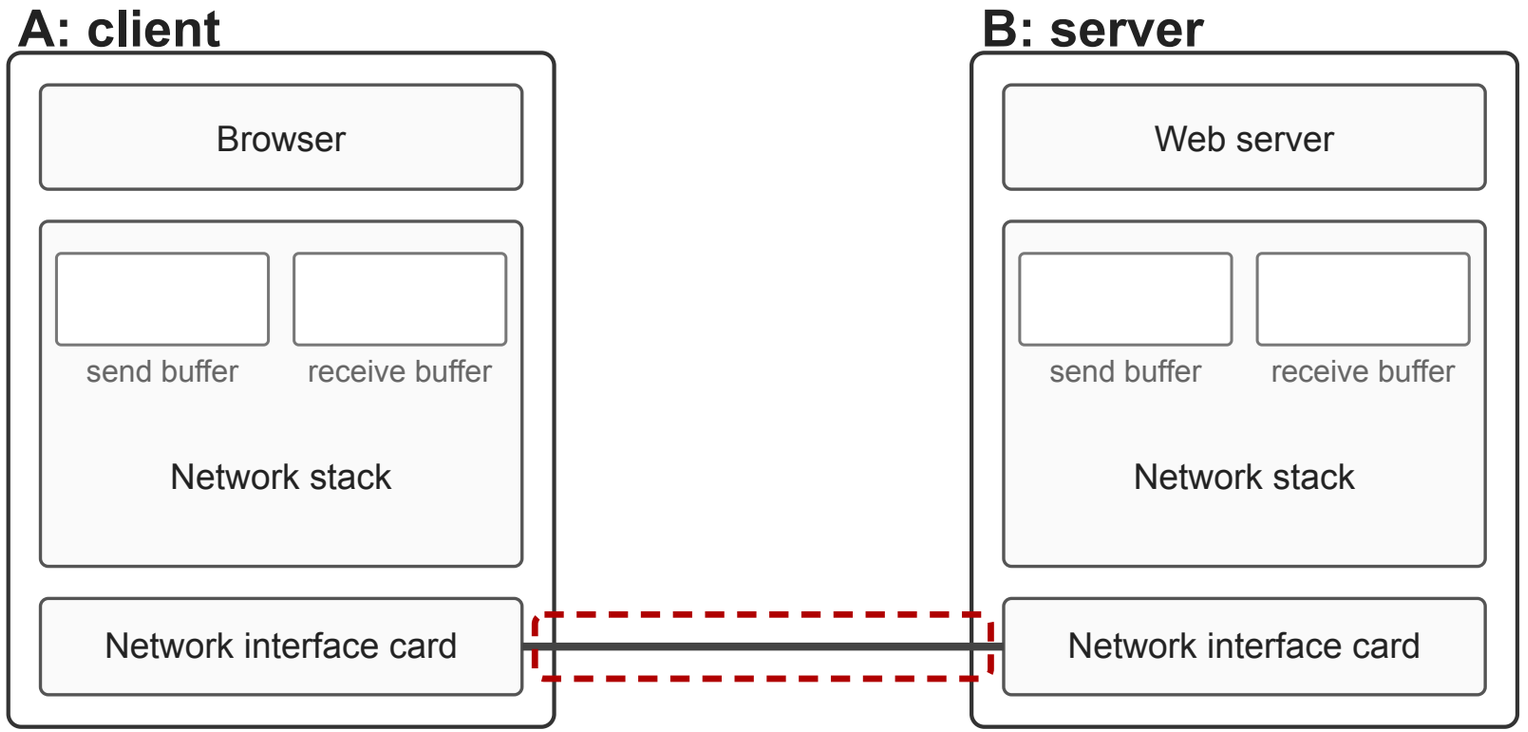
What impacts how fast A gets the file?



Client processing delay: the time A spends on the request before it can leave. The browser builds the request, and the network stack copies the bytes into its buffer and down to the card.

delay = $d_{proc,A}$

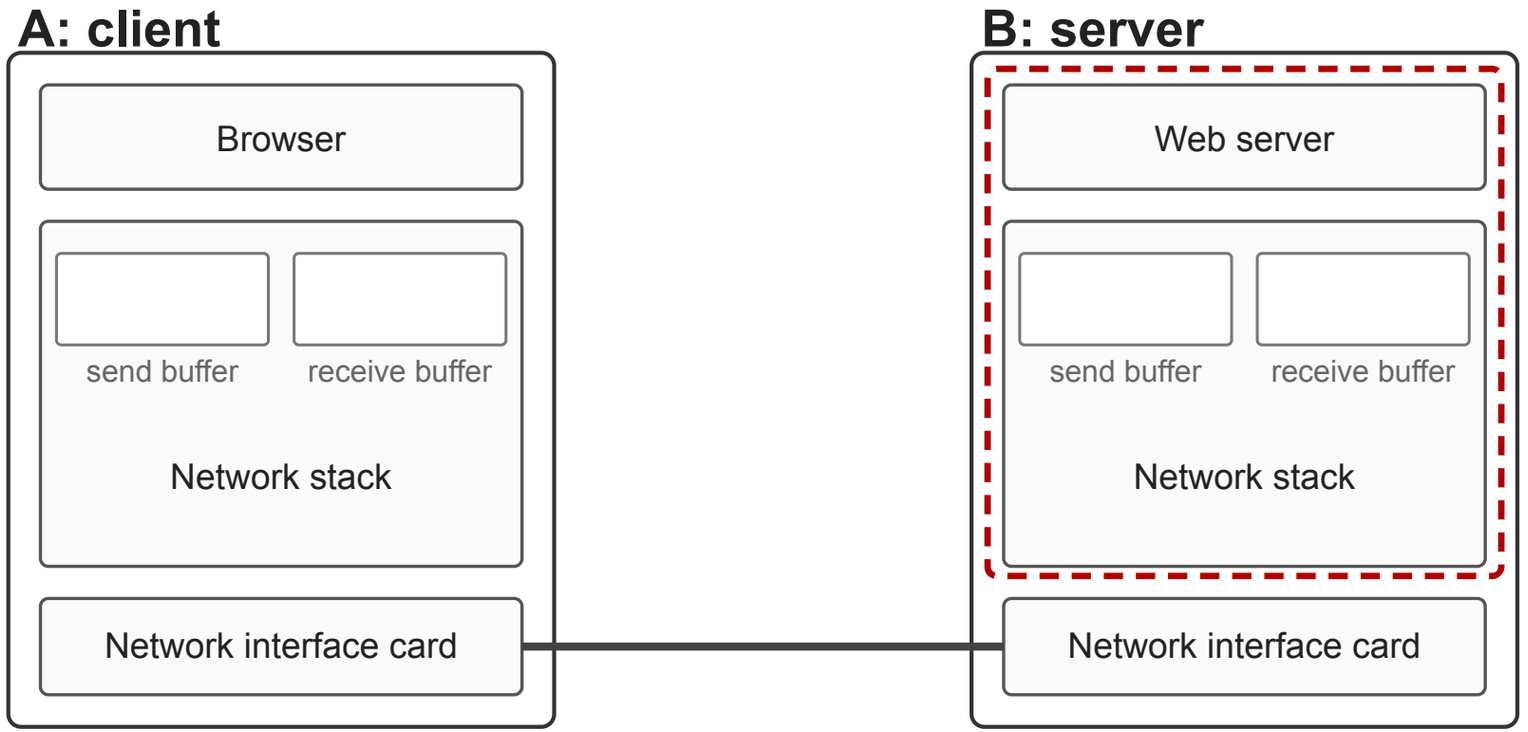
What impacts how fast A gets the file?



Propagation delay: the time the signal takes to travel along the link from A to B. It depends on how long the link is and how fast the signal moves in it, about 2×10^8 m/s in copper or fibre.

delay = $d_{proc,A} + d_{prop}$

What impacts how fast A gets the file?

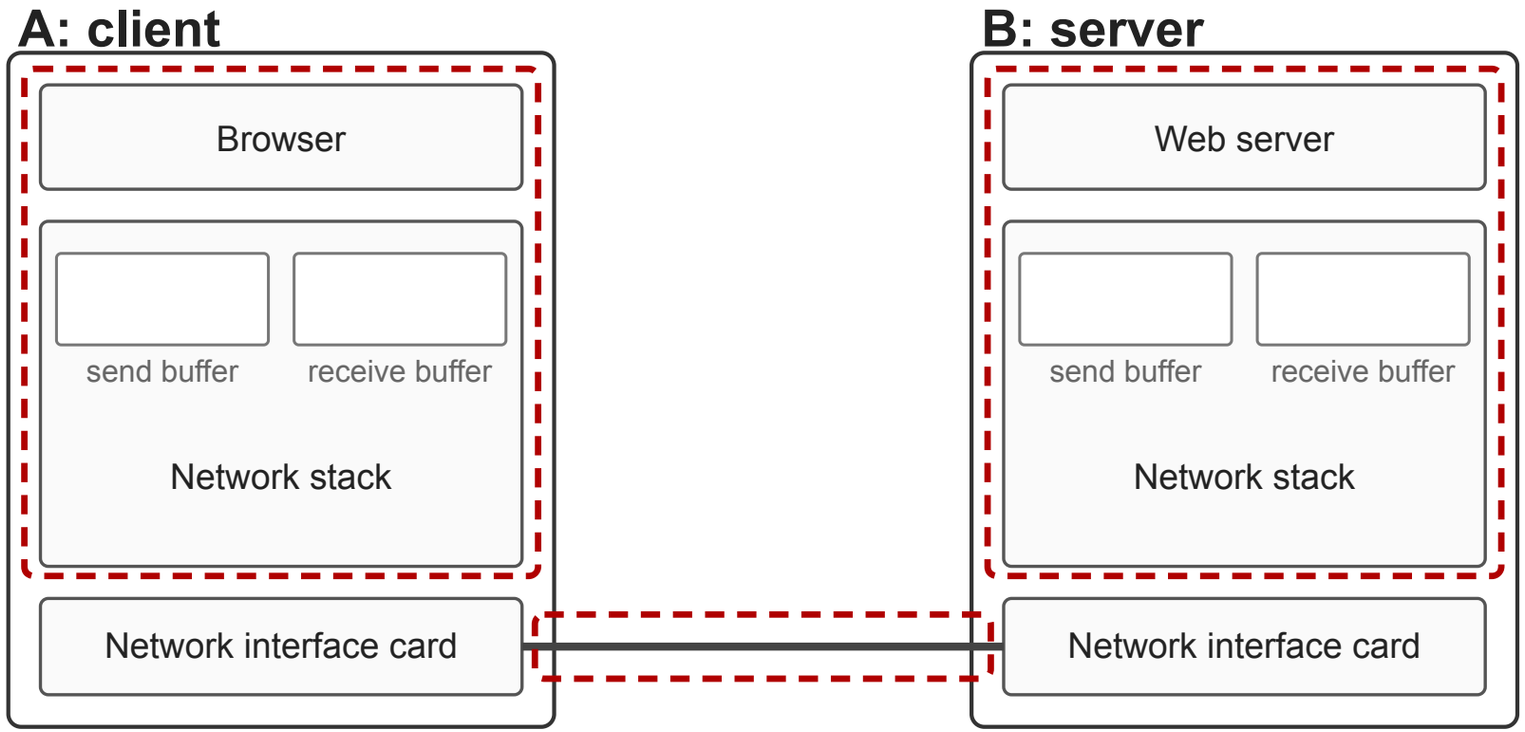


Server processing delay: the time B spends on the request once it arrives. The network stack copies the bytes up, and the web server reads the request and finds the file.

---

delay = $d_{proc,A} + d_{prop} + d_{proc,B}$

What impacts how fast A gets the file?

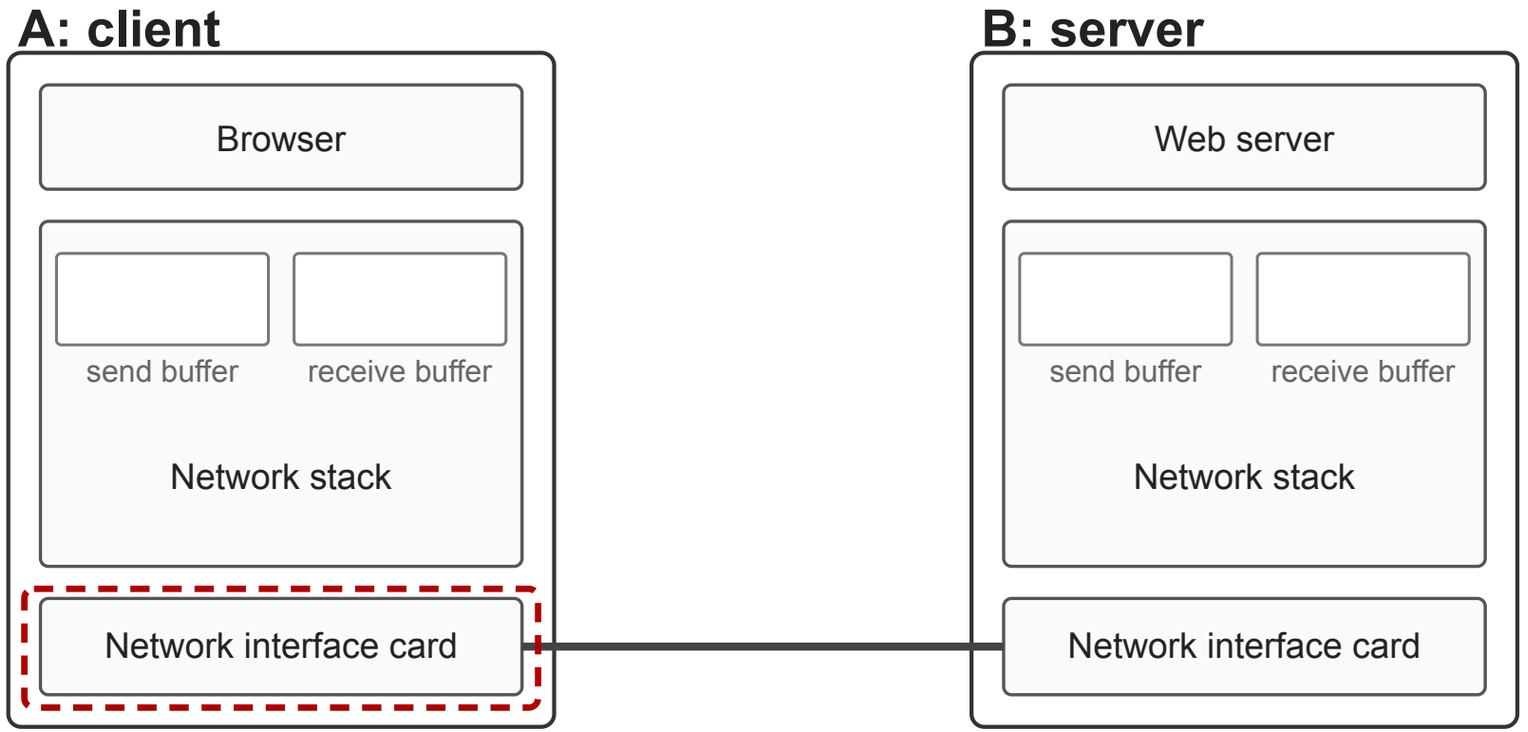


And back: the file makes the same trip the other way. B handles it on the way out, it crosses the link, and A handles it on the way in.

---

delay = 2 × ($d_{proc,A}$ + d_{prop} + $d_{proc,B}$)

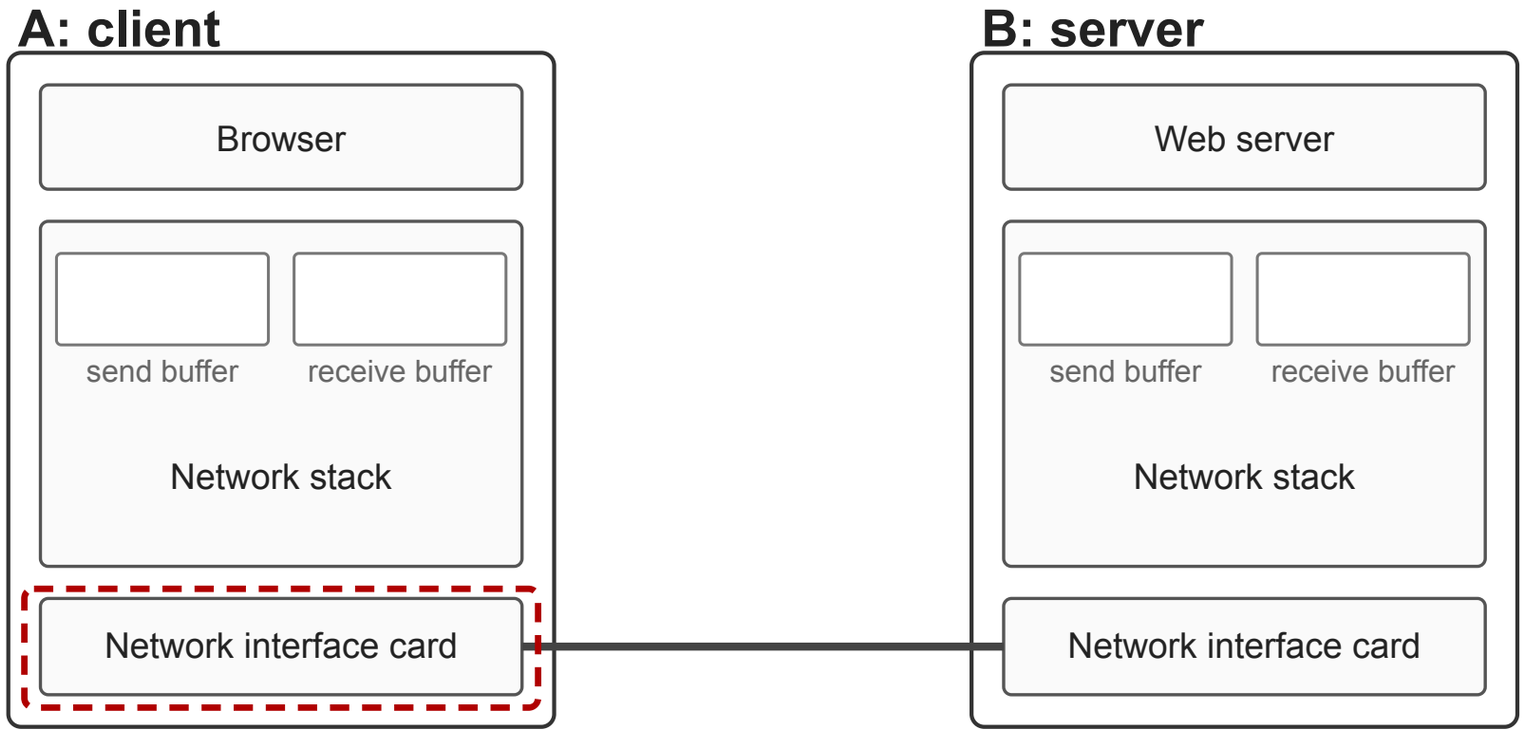
What impacts how fast A gets the file?



We missed something. The bytes do not jump from A's network stack straight onto the link. What happens in between?

delay = 2 × ($d_{proc,A}$ + d_{prop} + $d_{proc,B}$) + ?

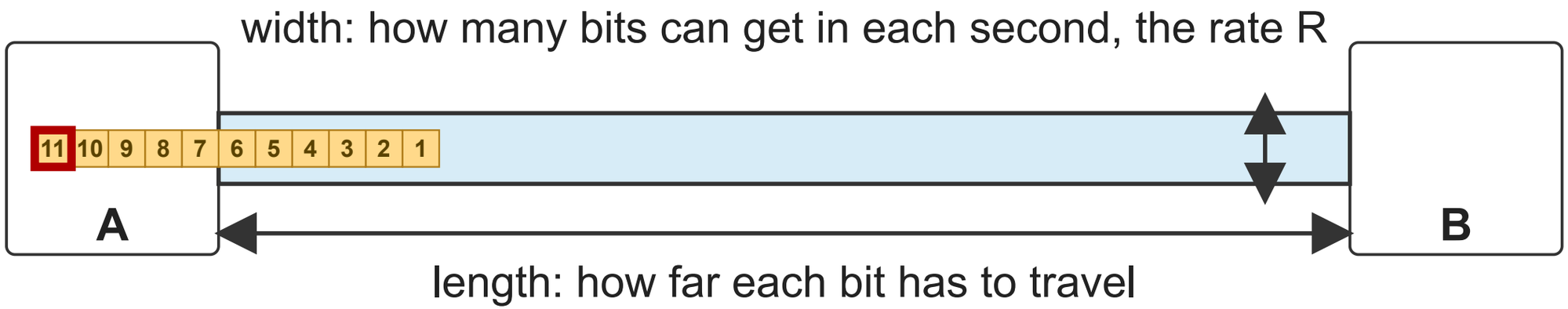
What impacts how fast A gets the file?



Transmission delay: the card puts the bits onto the link one after another, at the link's **rate**, R bits per second. Sending L bits takes L / R seconds. A 1 MB file is 8,000,000 bits, so at R = 100 Mbps it takes 0.08 s.

$$\textbf{delay} = 2 \times (d_{proc,A} + d_{prop} + d_{proc,B}) + L_{request} / R + L_{file} / R$$

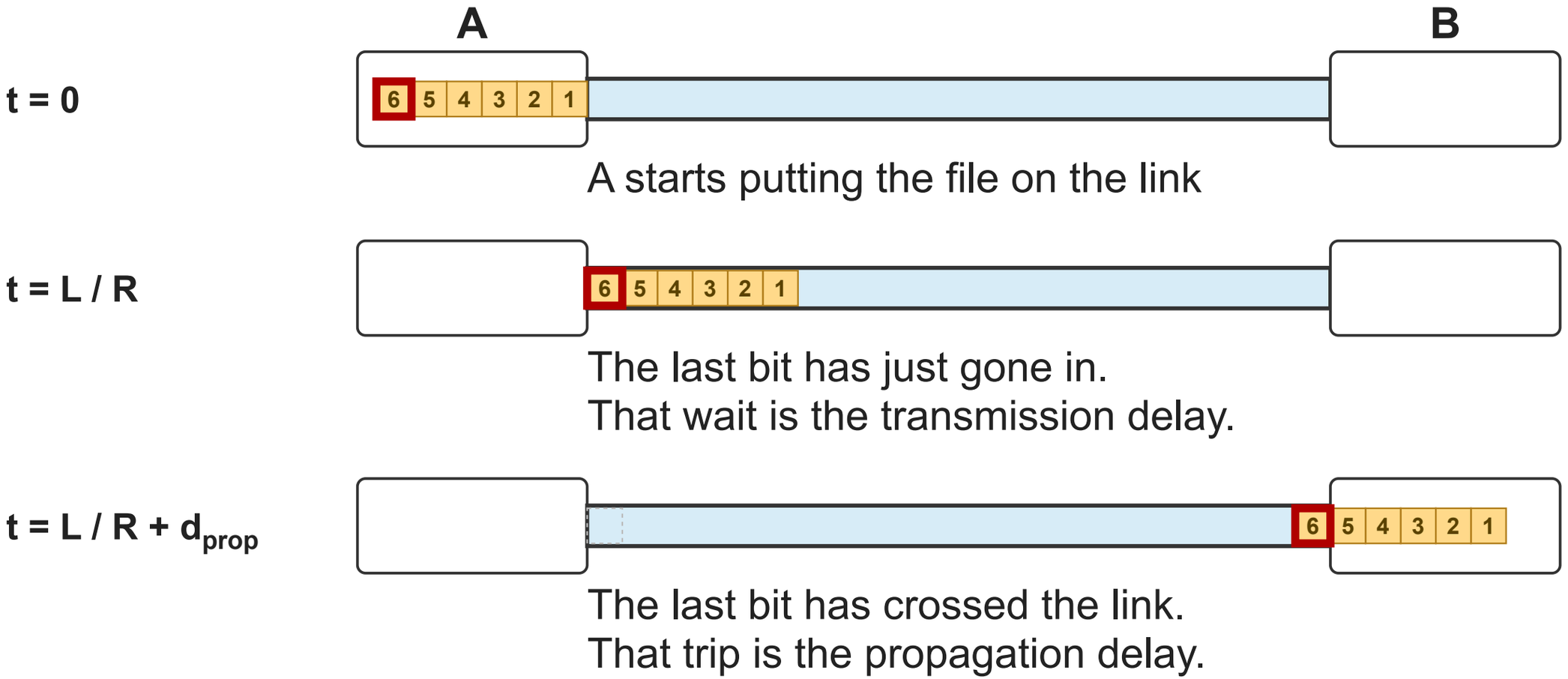
The link as a pipe



propagation delay: how long one bit takes to get from A to B

transmission delay: how long until the last bit, 11, is in the pipe

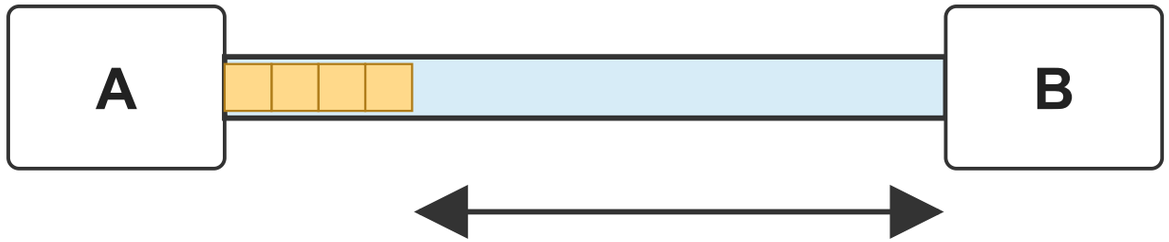
When does the last bit reach B?



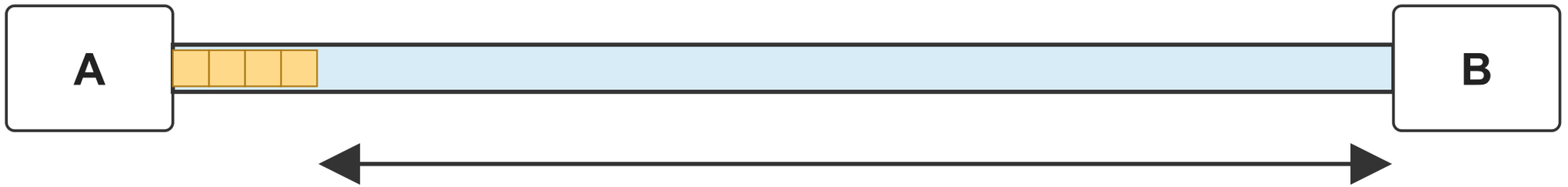
The last bit reaches B after transmission delay + propagation delay

A longer pipe

Short pipe



Long pipe



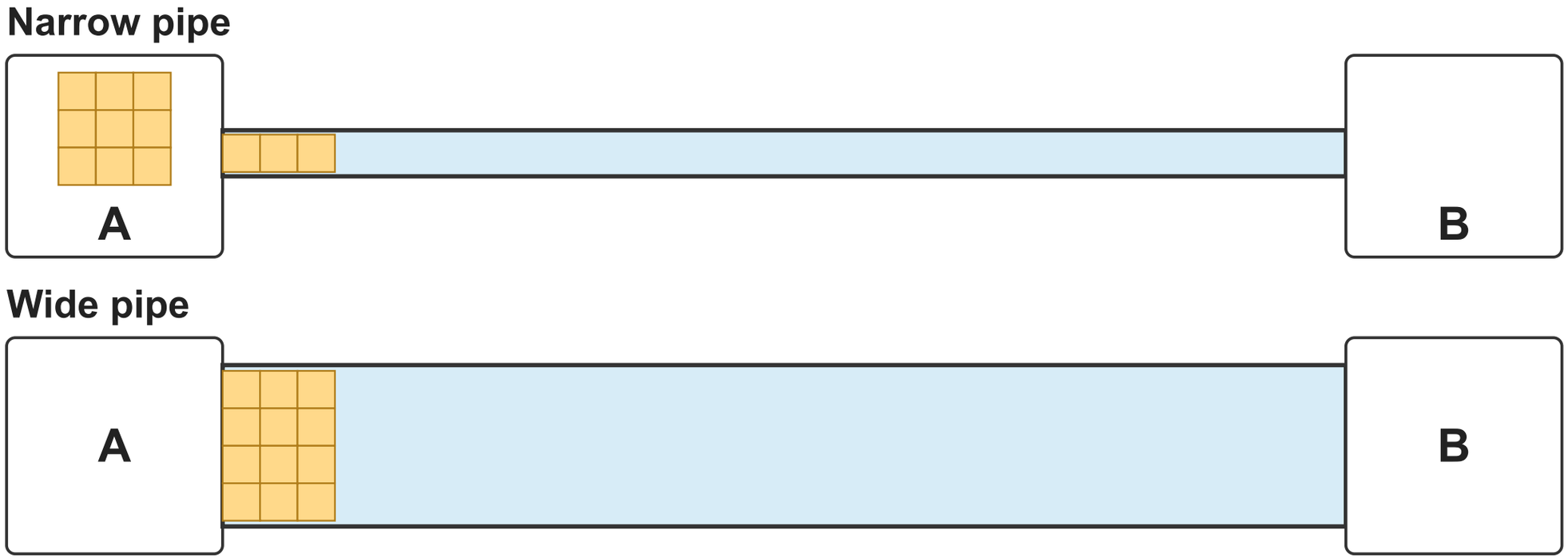
Each bit has further to go before it reaches B

→ more propagation delay

The same number of bits get in each second

→ same transmission delay

A wider pipe



More bits get in each second, so the file is on the link sooner

→ less transmission delay

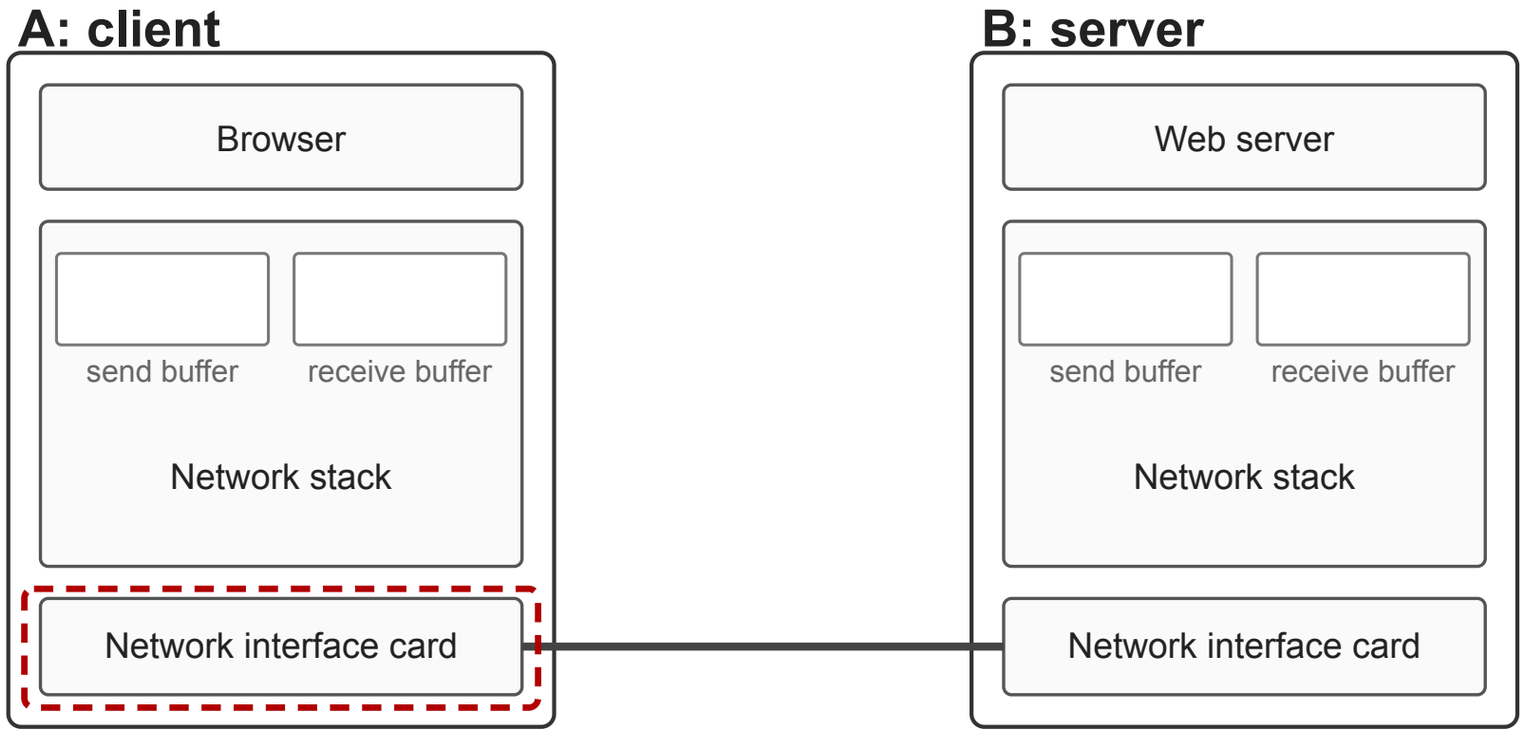
Each bit still travels the same distance, just as fast

→ same propagation delay

Two different delays

	Propagation delay	Transmission delay
Longer pipe	goes up	no change
Wider pipe (higher R)	no change	goes down
Bigger file	no change	goes up
Equals	**length / speed**	**L / R**

What impacts how fast A gets the file?



Transmission delay: the card puts the bits onto the link one after another, at the link's **rate**, R bits per second. Sending L bits takes L / R seconds. A 1 MB file is 8,000,000 bits, so at R = 100 Mbps it takes 0.08 s.

$$\textbf{delay} = 2 \times (d_{proc,A} + d_{prop} + d_{proc,B}) + L_{request} / R + L_{file} / R$$

When the file is large, what dominates the delay?

$$\text{delay} = 2 \times (d_{proc,A} + d_{prop} + d_{proc,B}) + L_{request} / R + L_{file} / R$$

Real numbers: a large file

A 1 GB video from Vancouver, about 4,000 km of fibre away, over a 100 Mbps home connection. The request is just the file name, about 20 bytes.

Part of the delay	Worked out	How long
Processing	2 × (1 ms + 1 ms)	4 ms
Propagation	2 × 4,000 km ÷ 200,000 km/s	40 ms
Transmission, request	160 bits ÷ 10^8 bits/s	0.0016 ms
Transmission, file	8 × 10^9 bits ÷ 10^8 bits/s	80 s

When the file is small, what dominates the delay?

$$\text{delay} = 2 \times (d_{proc,A} + d_{prop} + d_{proc,B}) + L_{request} / R + L_{file} / R$$

Real numbers: a small file

A 1 KB file from the same server, over the same connection. The request is the same file name, about 20 bytes.

Part of the delay	Worked out	How long
Processing	2 × (1 ms + 1 ms)	4 ms
Propagation	2 × 4,000 km ÷ 200,000 km/s	40 ms
Transmission, request	160 bits ÷ 10^8 bits/s	0.0016 ms
Transmission, file	8,000 bits ÷ 10^8 bits/s	0.08 ms

A rule of thumb

- **Large transfers** are limited by how many bits per second the network can carry: **throughput**
- **Small transfers** are limited by how long each bit takes to get there: **latency**

So far, we have abstracted the network as one link with a fixed length and rate. In a real network, the rate and the propagation delay are something network engineers can and do plan for. We come back to this after Part 1.

Outline

- Building a web browser
- How long does it take to get a file?
- **Pages with more than one file**
- Takeaways

Back to our browser

Browser (A)

```
void net_send(uint8 *bytes, int n);
void net_recv(uint8 *bytes, int n);

int main() {
    char *name = read_filename();



    display(page);
}
```

Server (B)

```
void net_send(uint8 *bytes, int n);
void net_recv(uint8 *bytes, int n);

int main() {

}
```

Reset

The first version of HTTP was this, with more features

A real HTTP request and response

Browser to server

```
GET /index.html HTTP/1.1
Host: cs.uwaterloo.ca
Accept: text/html
Connection: keep-alive
```

Server to browser

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 4096

<html>
  ...
```

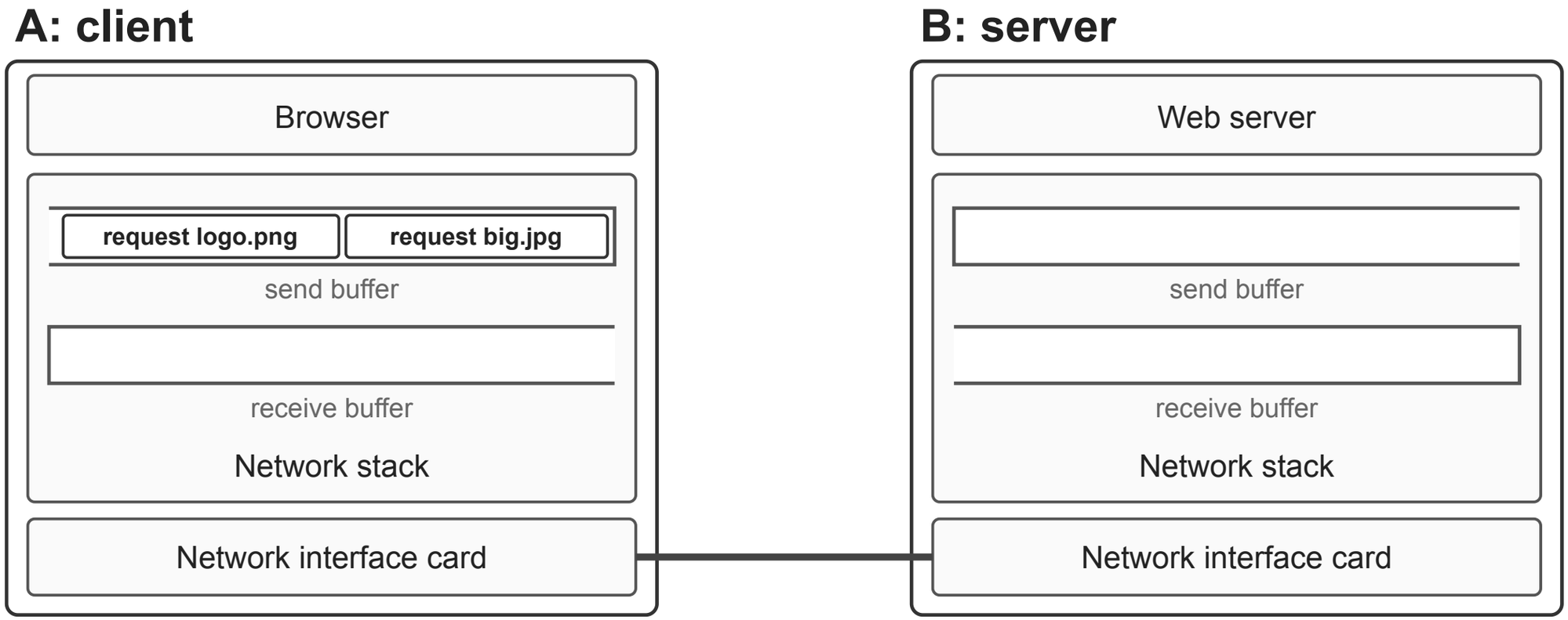
An HTML page is often more than one file

```
1 <html>
2   <body>
3     <img src="big.jpg">
4     <img src="logo.png">
5   </body>
6 </html>
```

- The browser only learns what else to ask for once it has read the page

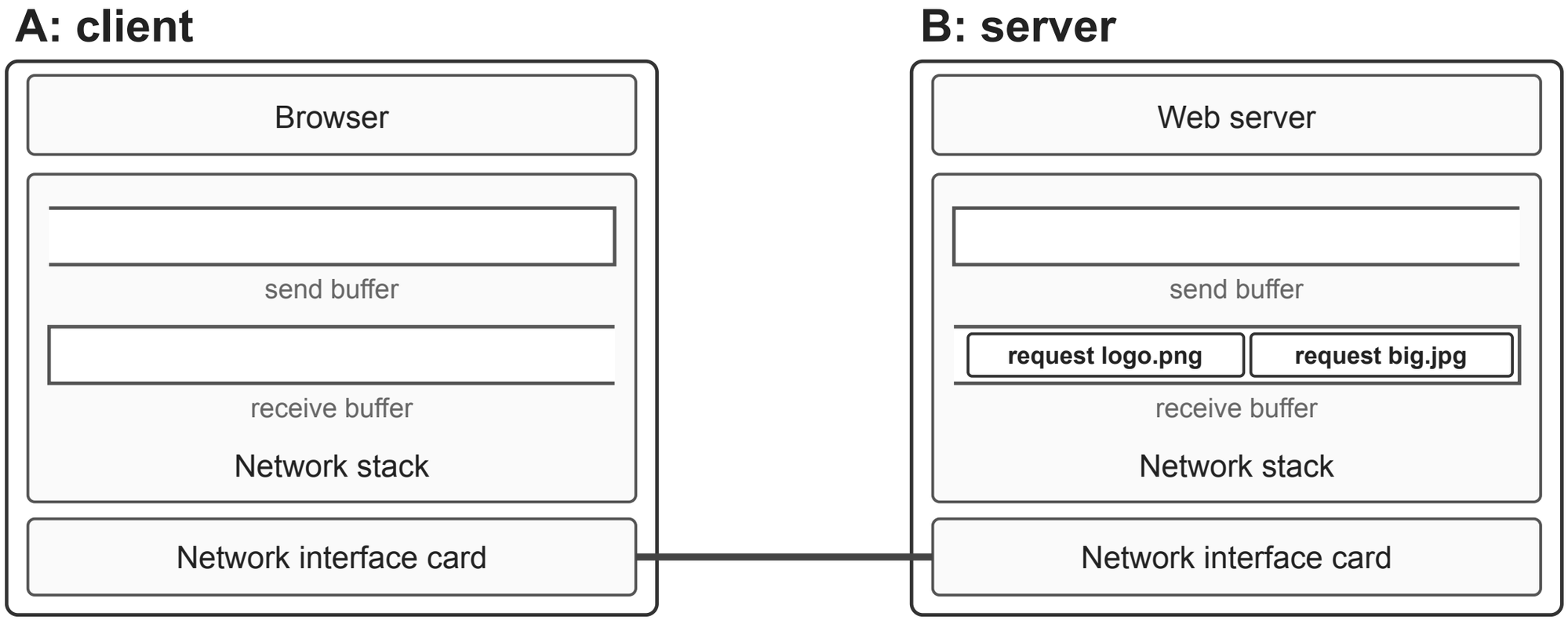
> How does it get the rest?

The browser asks for both files

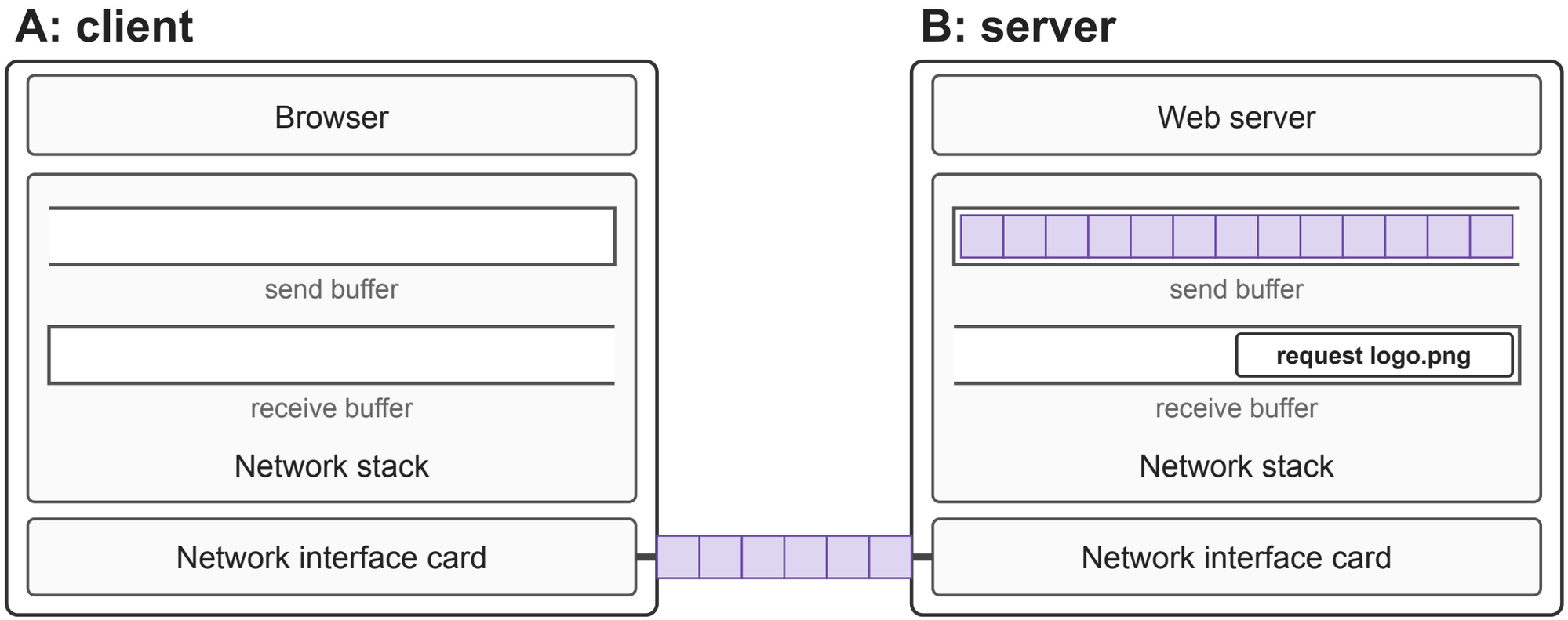


Two requests, one after the other, in the same stream of bytes

What does the server do?

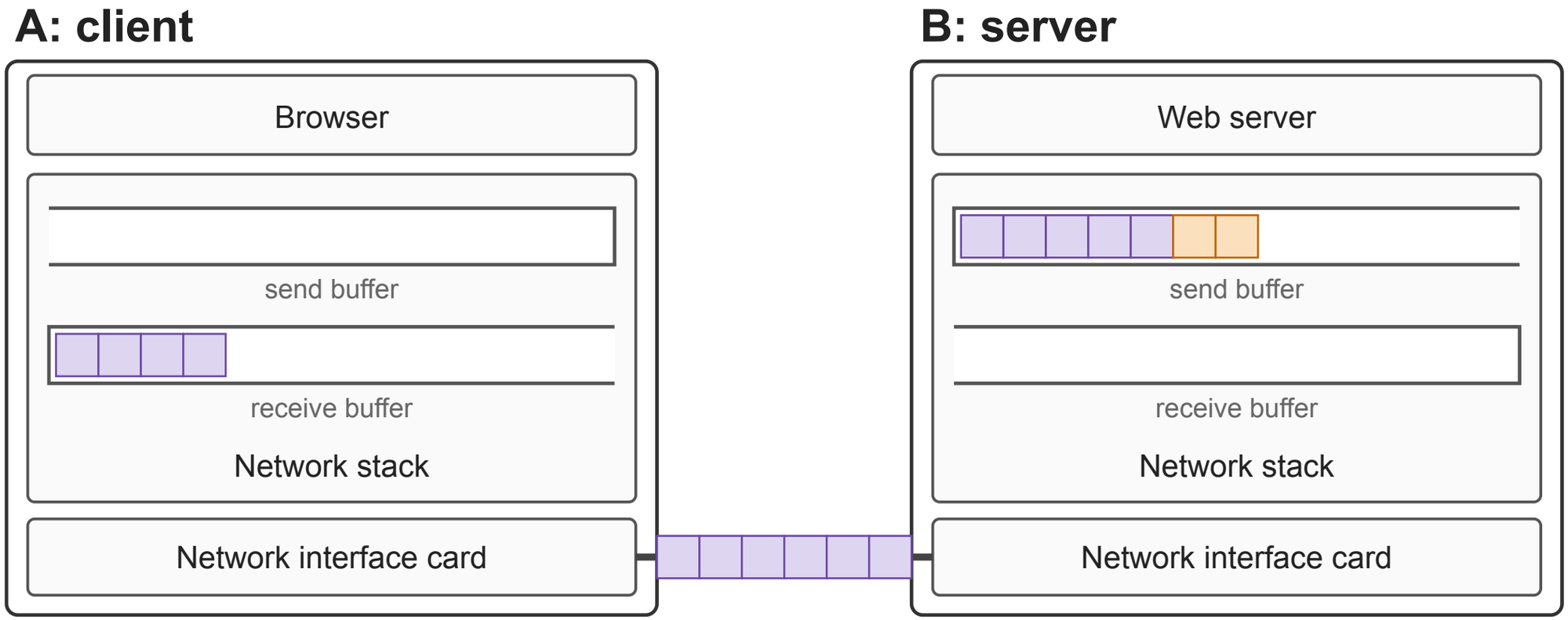


It reads the first request, and writes big.jpg



big.jpg does not fit in the send buffer, so writing it takes a while

Then it reads the second, and writes logo.png

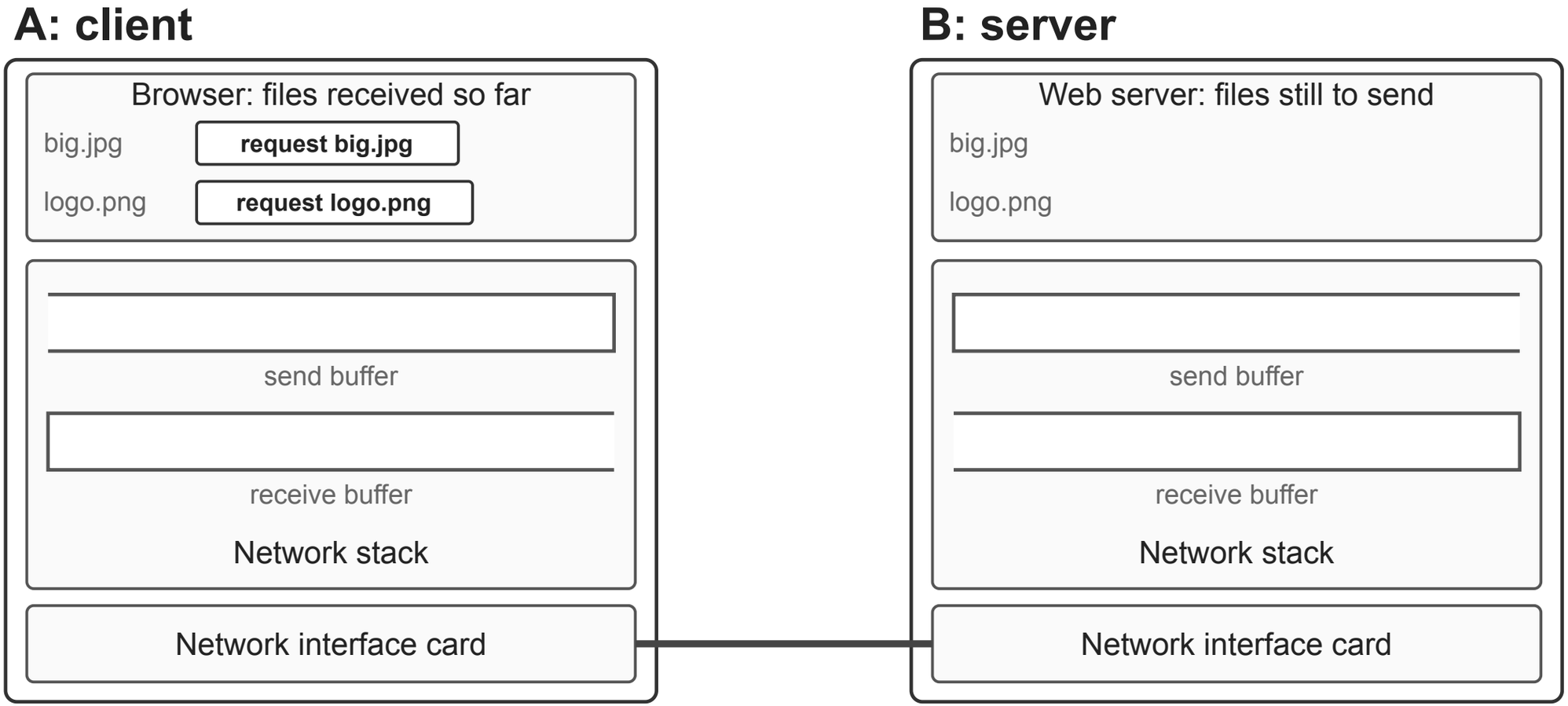


logo.png is small, and it is ready

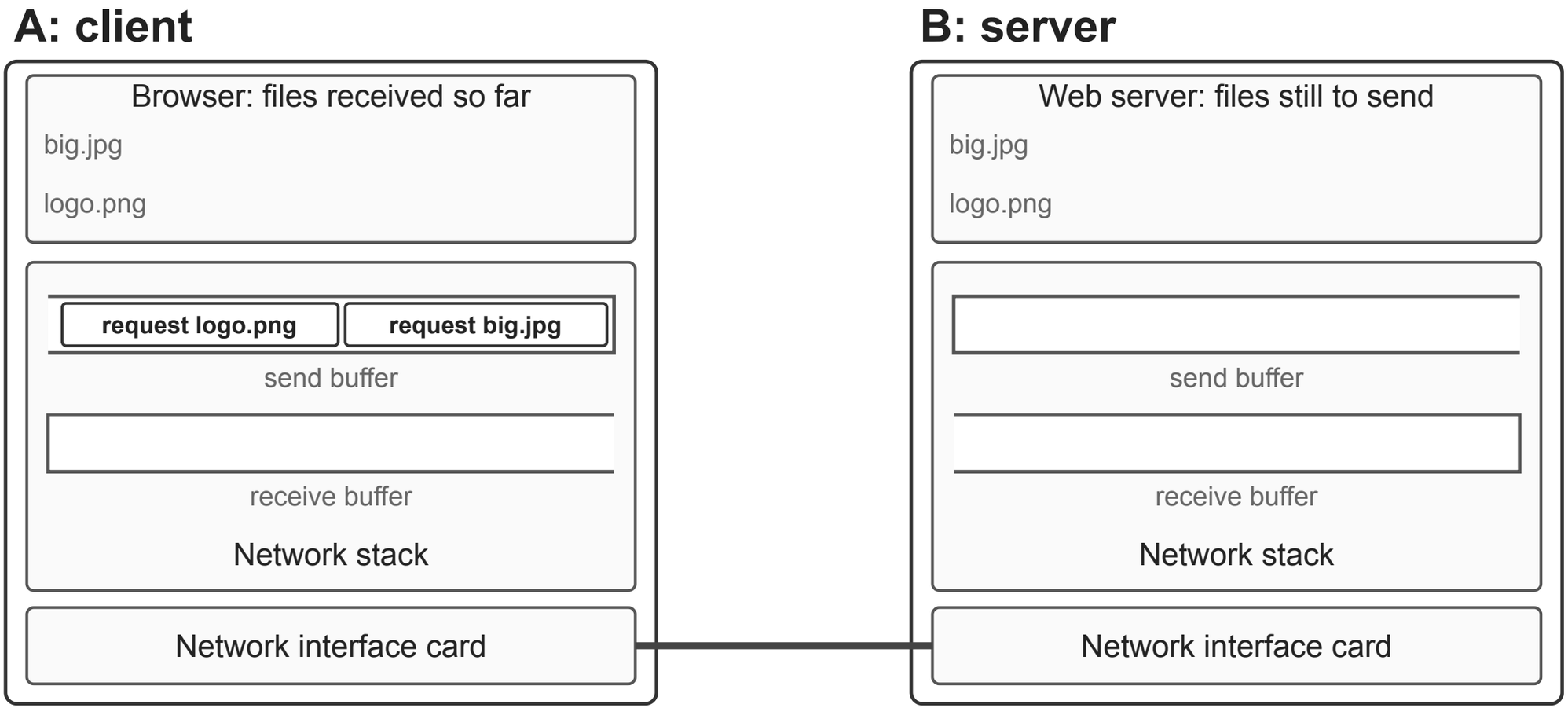
Why is this bad?

How would you fix it?

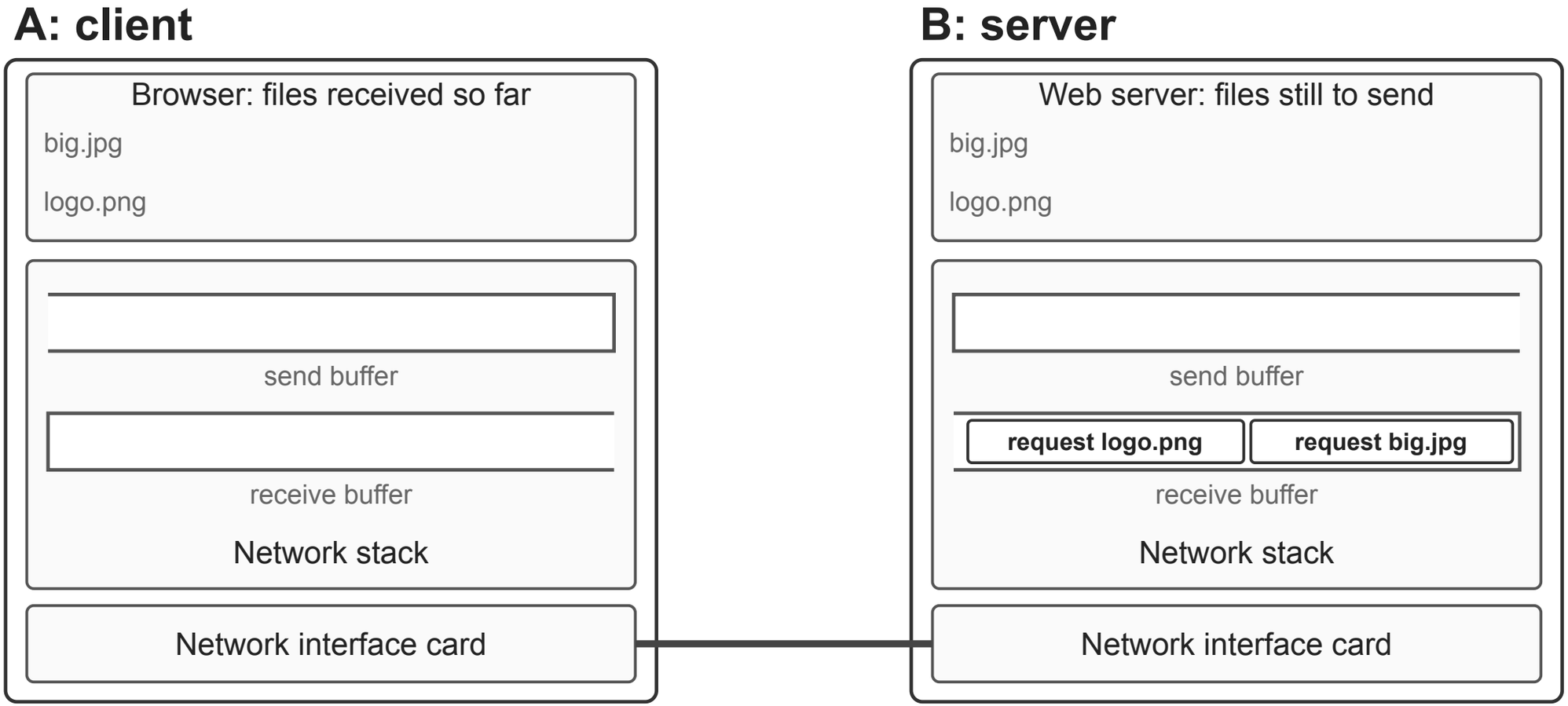
Fix 1: the server interleaves file chunks



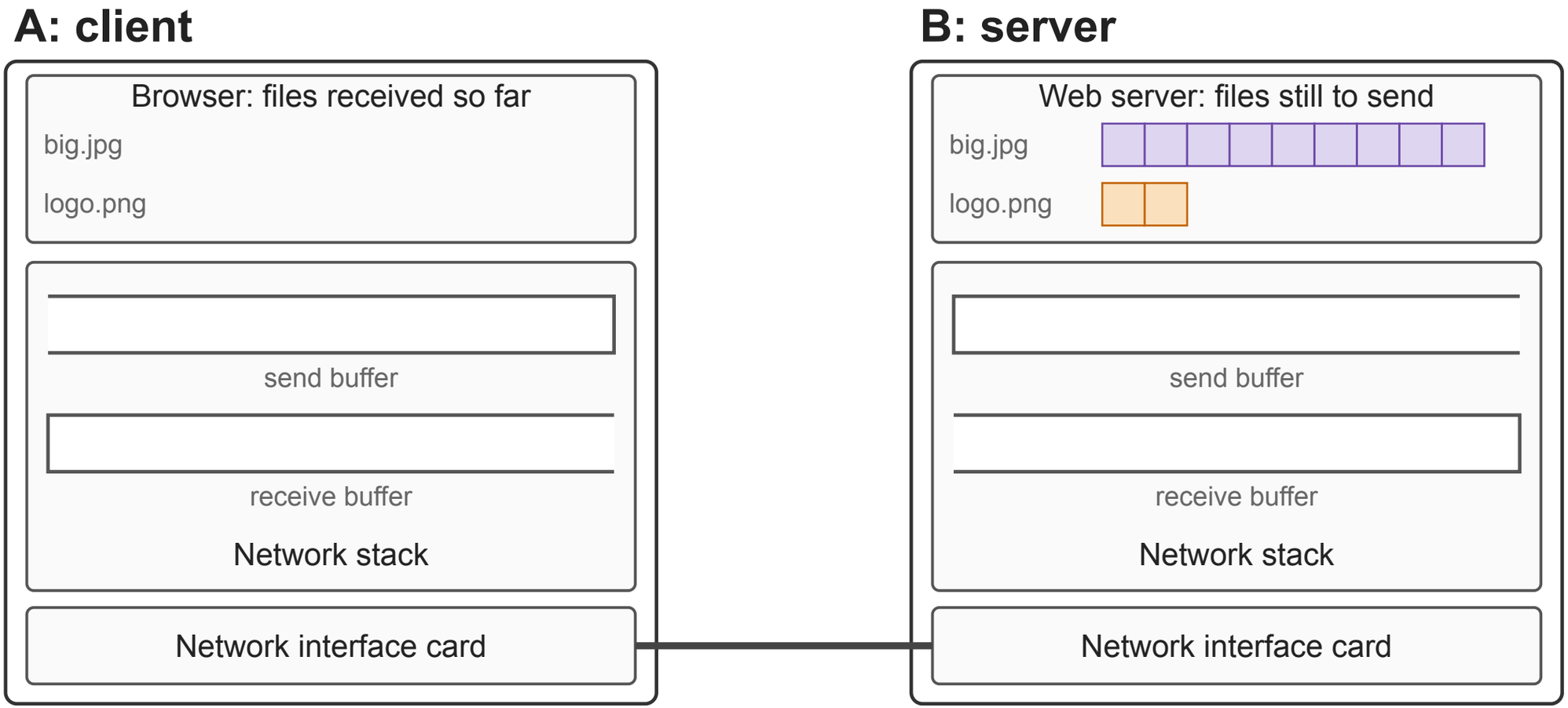
Fix 1: the server interleaves file chunks



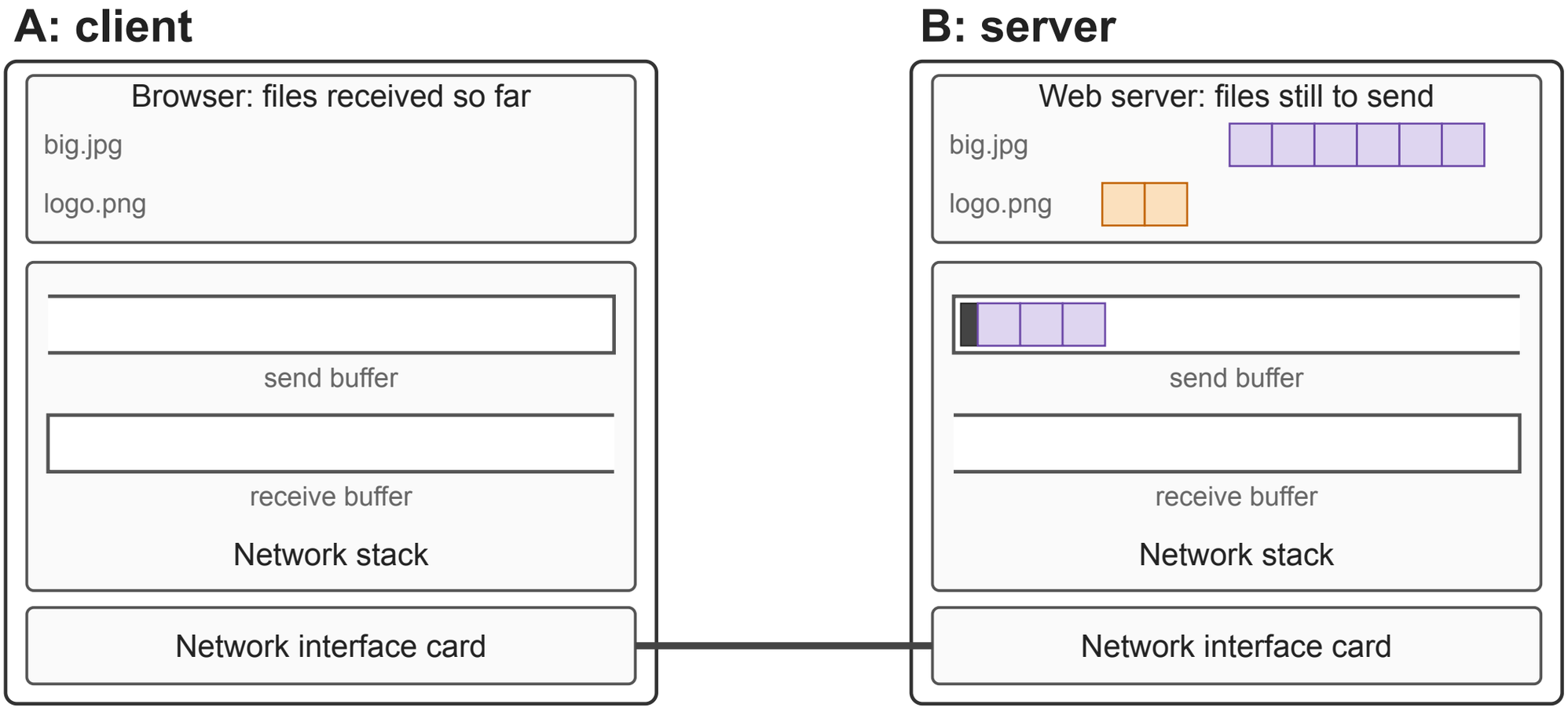
Fix 1: the server interleaves file chunks



Fix 1: the server interleaves file chunks

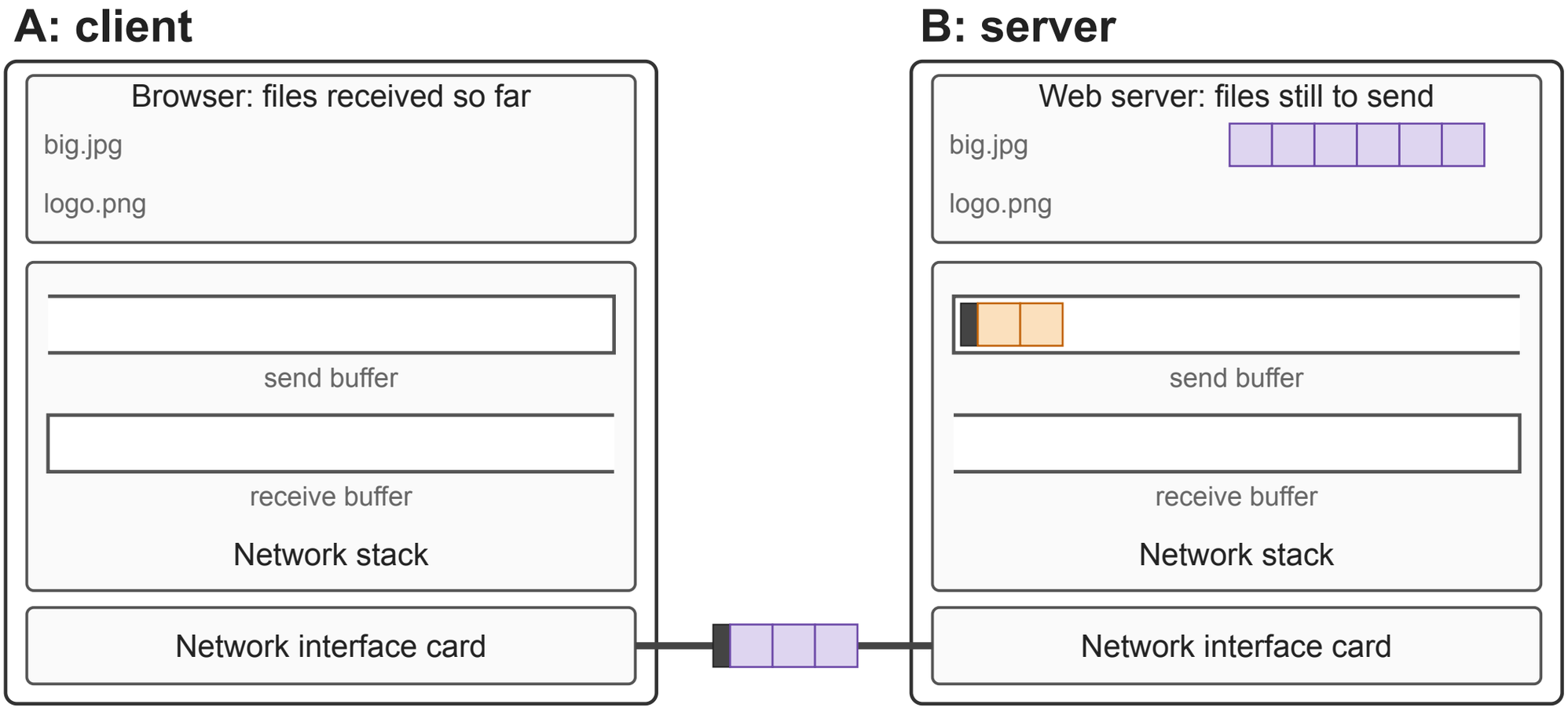


Fix 1: the server interleaves file chunks



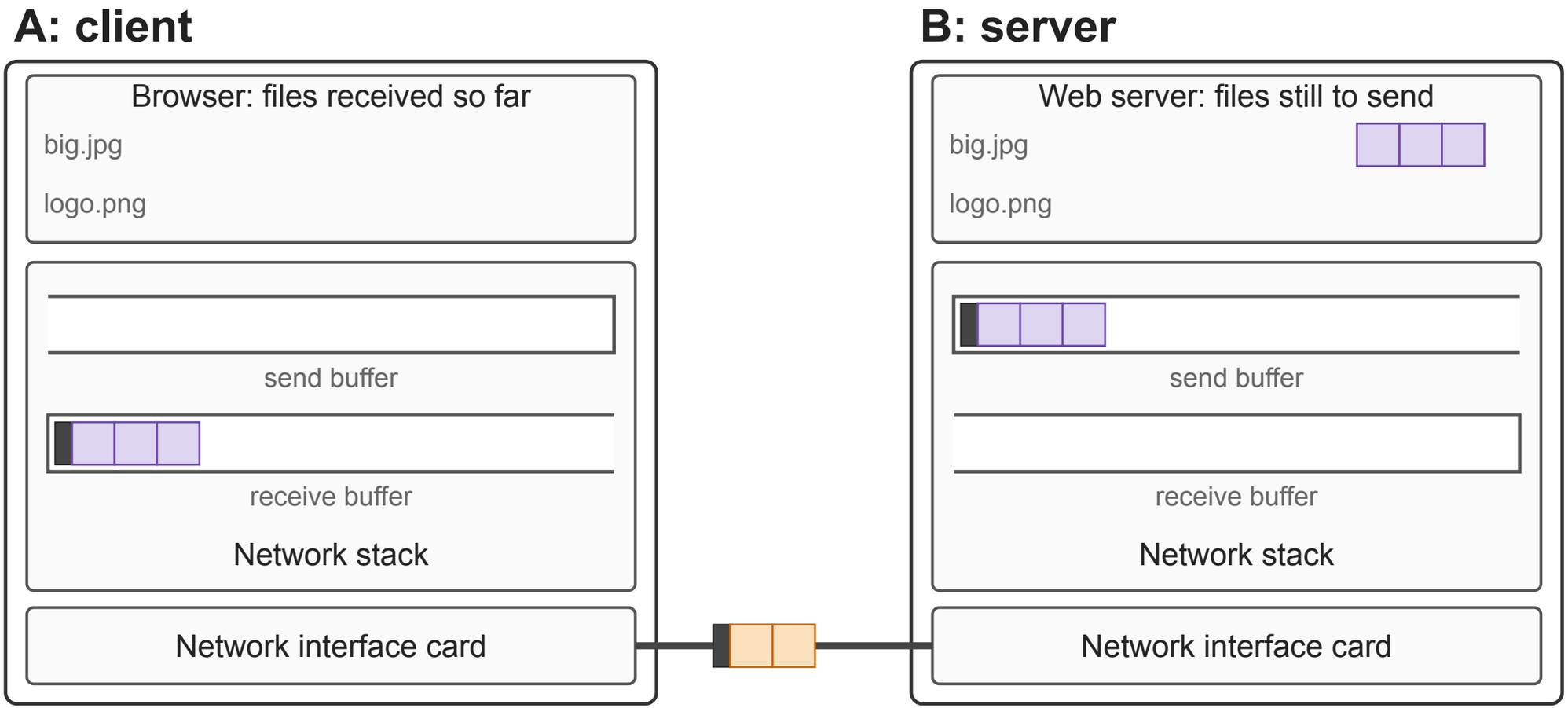
The server sends one piece at a time, each with a header

Fix 1: the server interleaves file chunks



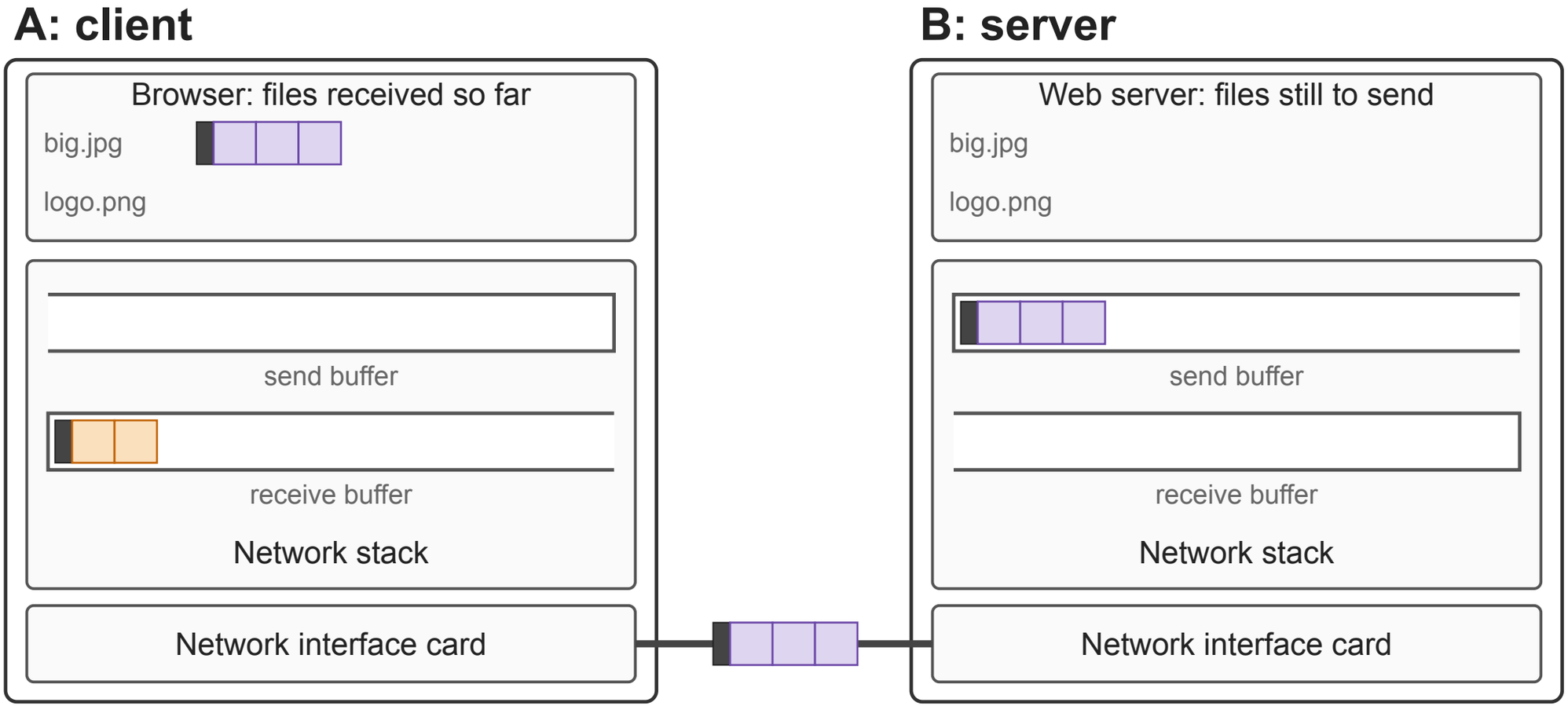
The server sends one piece at a time, each with a header

Fix 1: the server interleaves file chunks



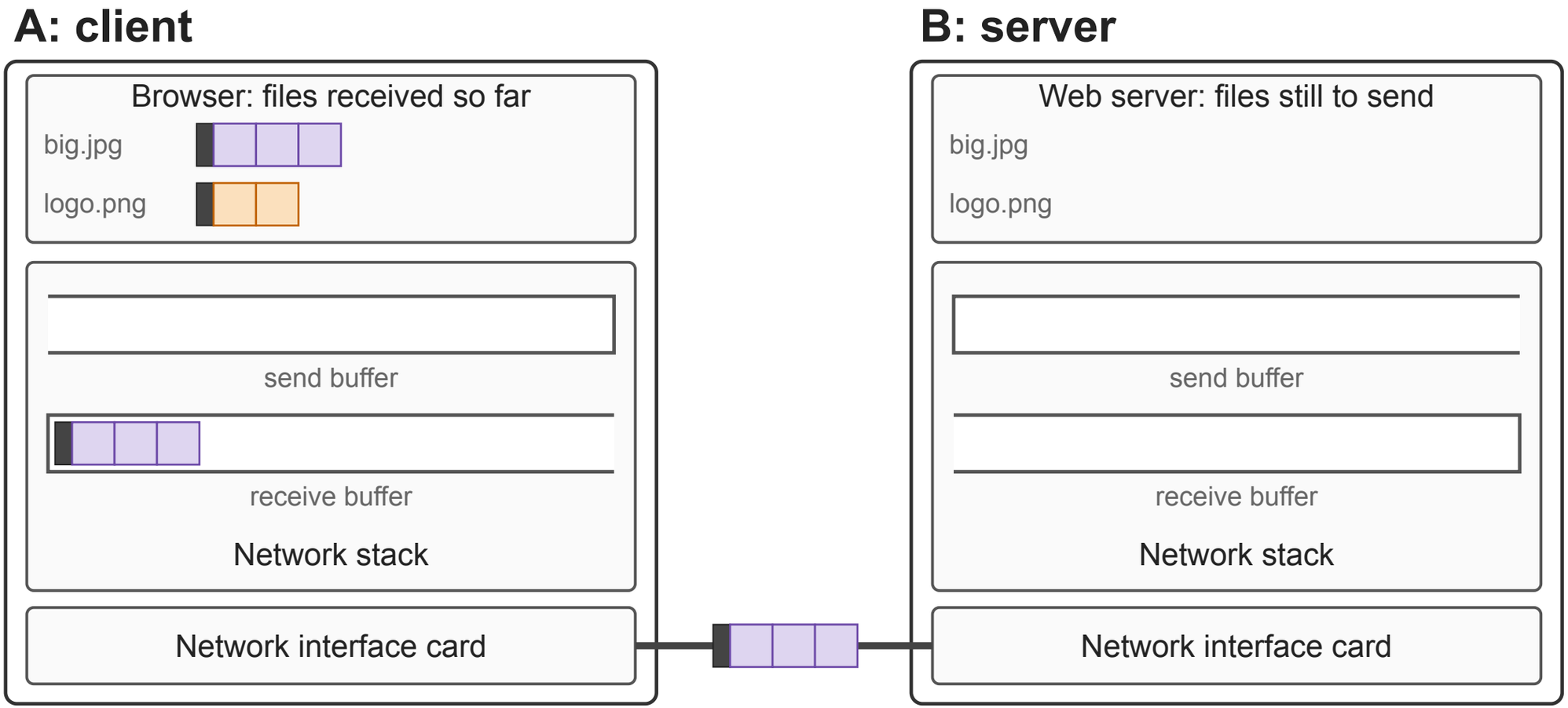
The server sends one piece at a time, each with a header

Fix 1: the server interleaves file chunks



The server sends one piece at a time, each with a header

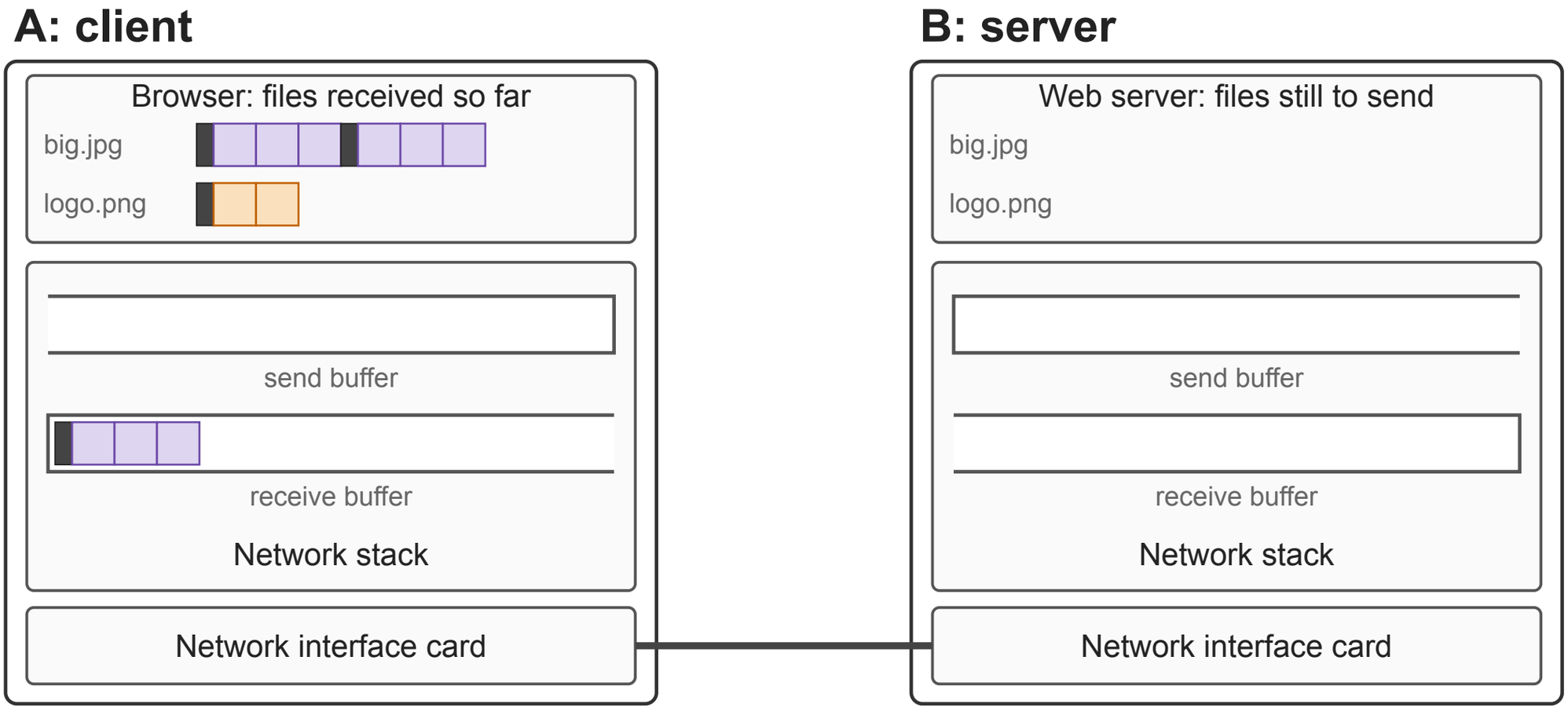
Fix 1: the server interleaves file chunks



The server sends one piece at a time, each with a header

logo.png is complete, while big.jpg is still arriving

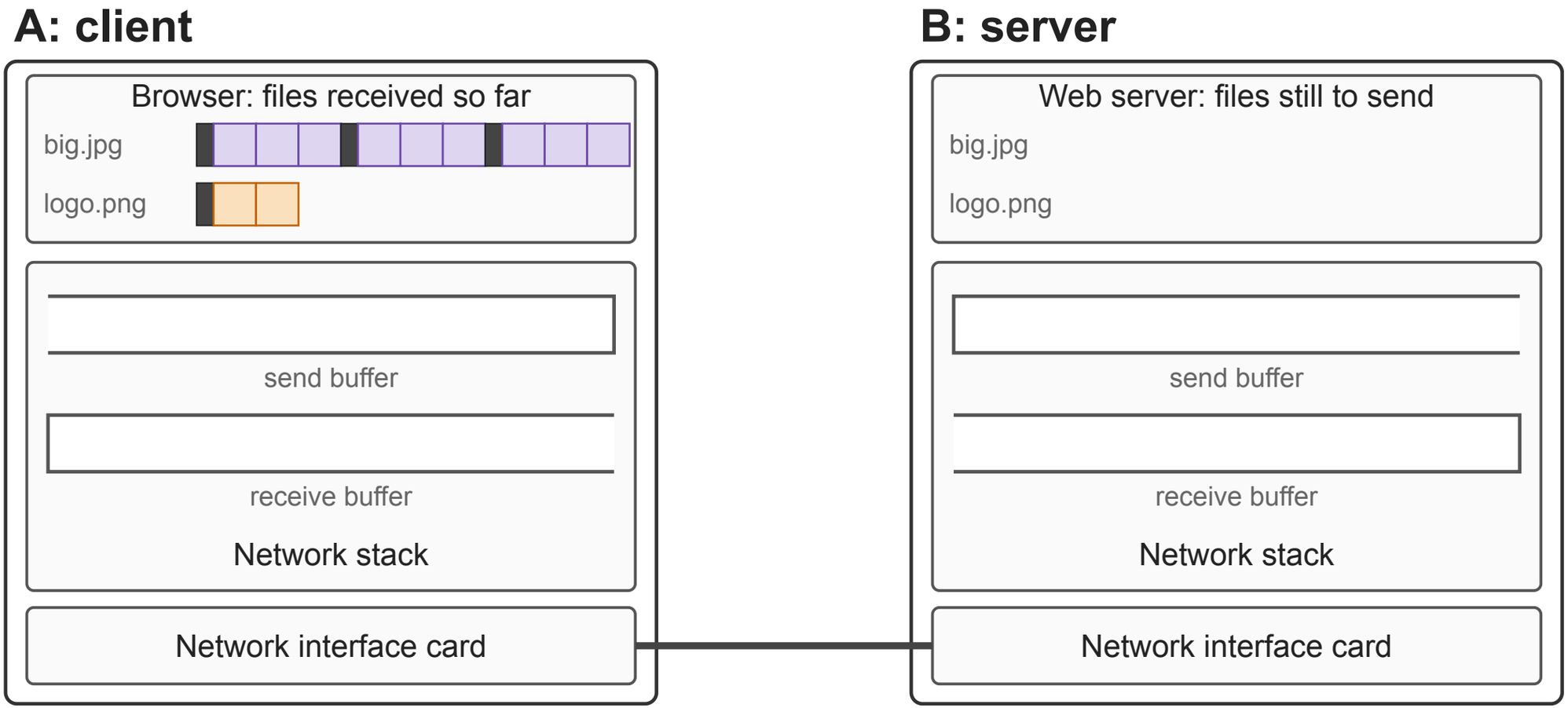
Fix 1: the server interleaves file chunks



The server sends one piece at a time, each with a header

logo.png is complete, while big.jpg is still arriving

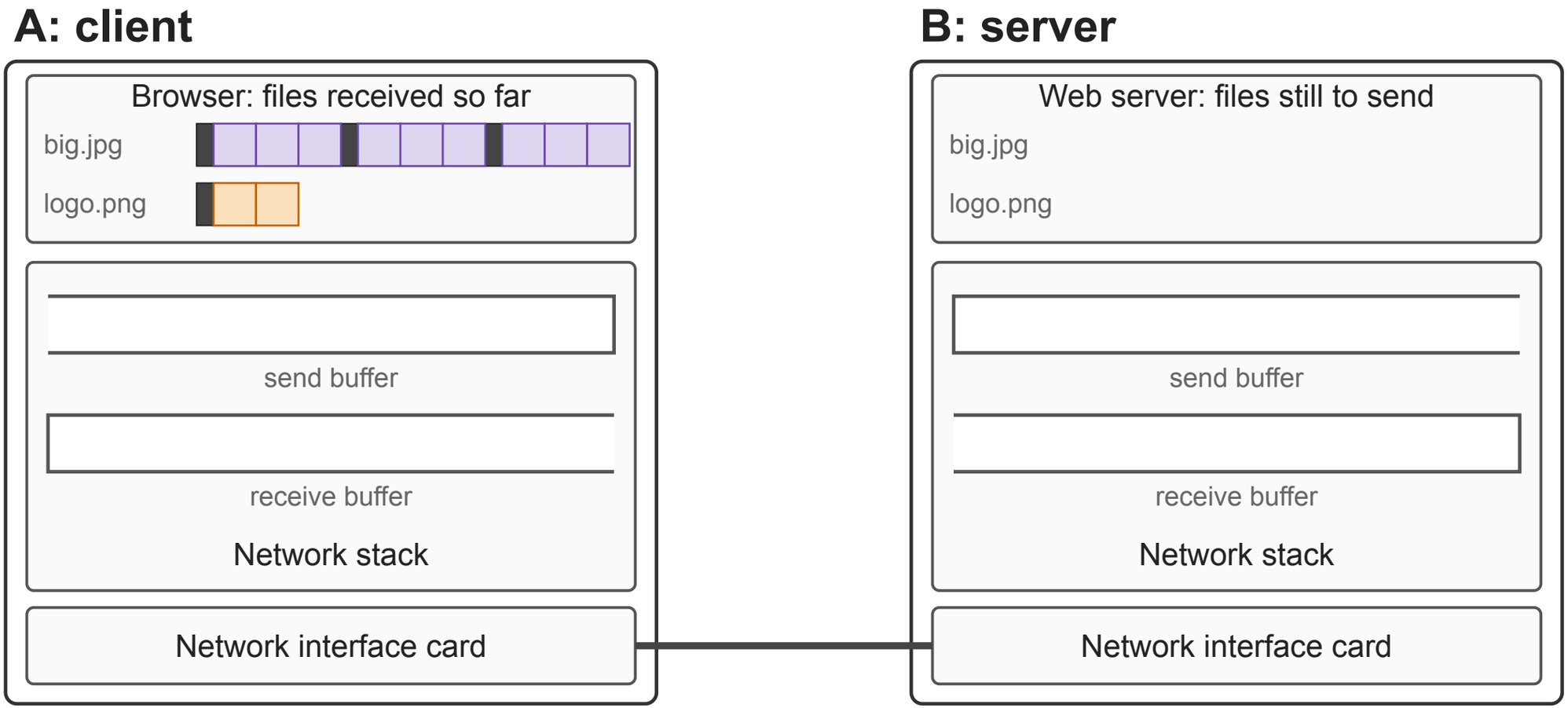
Fix 1: the server interleaves file chunks



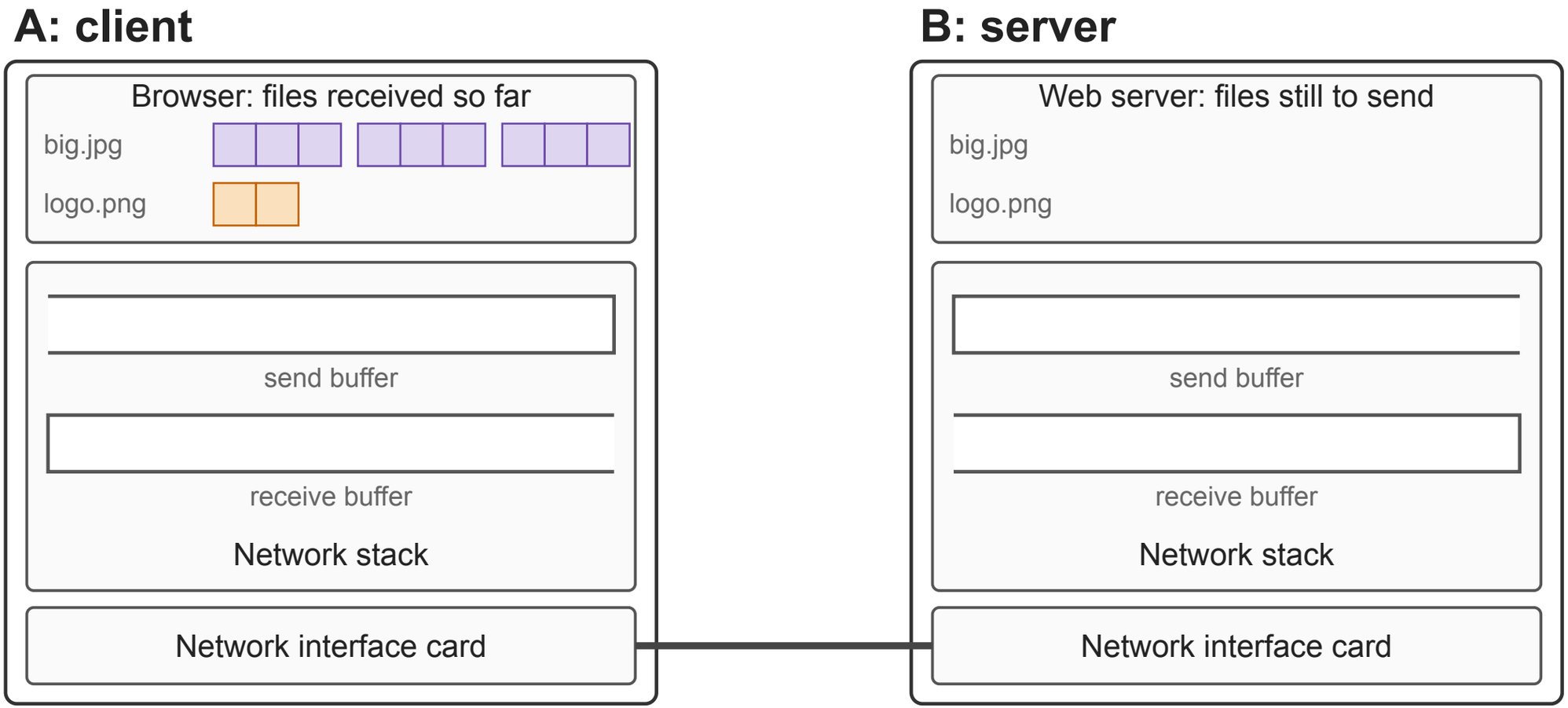
The server sends one piece at a time, each with a header

logo.png is complete, while big.jpg is still arriving

The browser puts the files back together

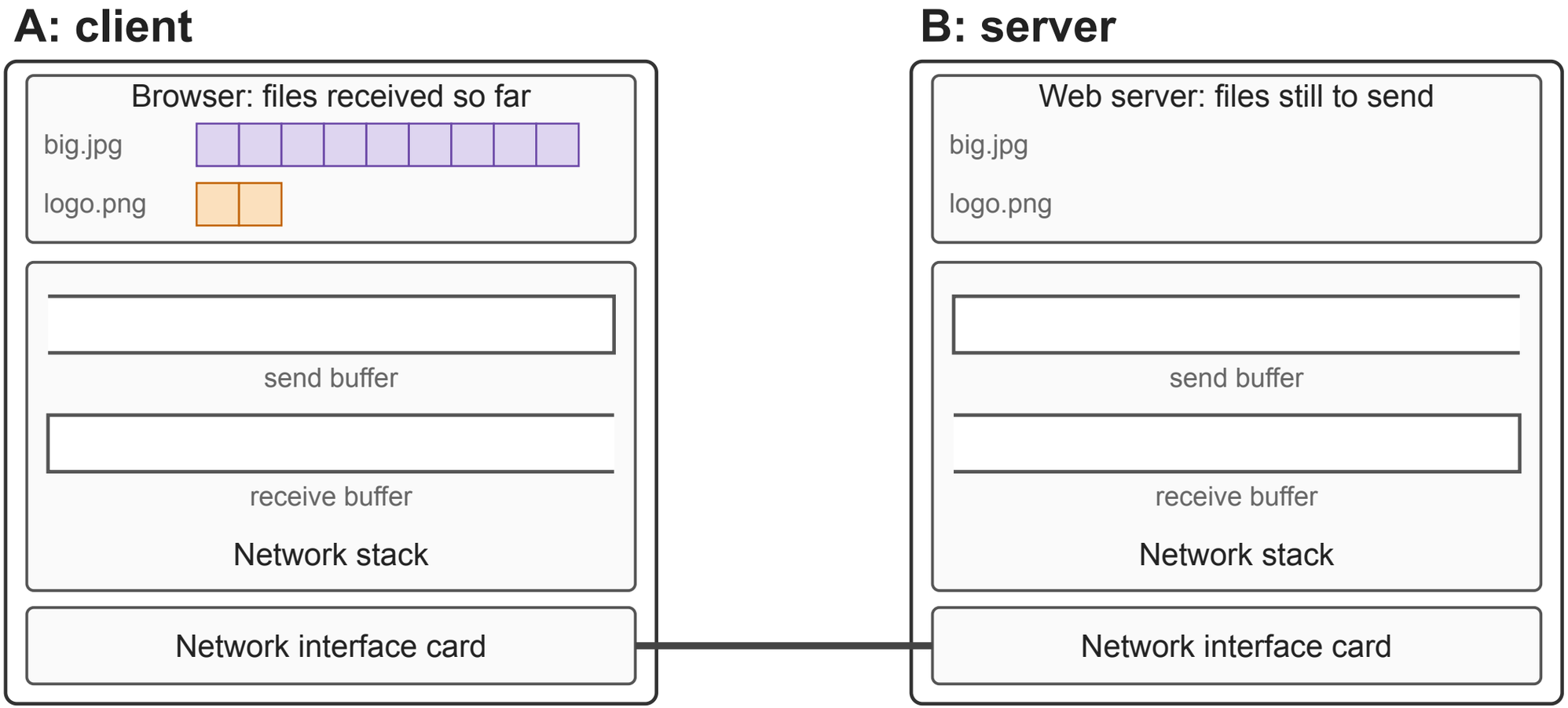


The browser puts the files back together



The browser has read the headers, so it can drop them

The browser puts the files back together



The browser has read the headers, so it can drop them

What is left of each file is its bytes, back together in order

The protocol between A and B

Have to attach metadata to each data chunk specifying while file, how far in the file, start and end.

Now the server decides what goes next

- The smallest files first? Whatever is on the screen first? Whatever the browser says it needs most?
- HTTP lets the browser say how much each file matters, and servers schedule their responses around it.

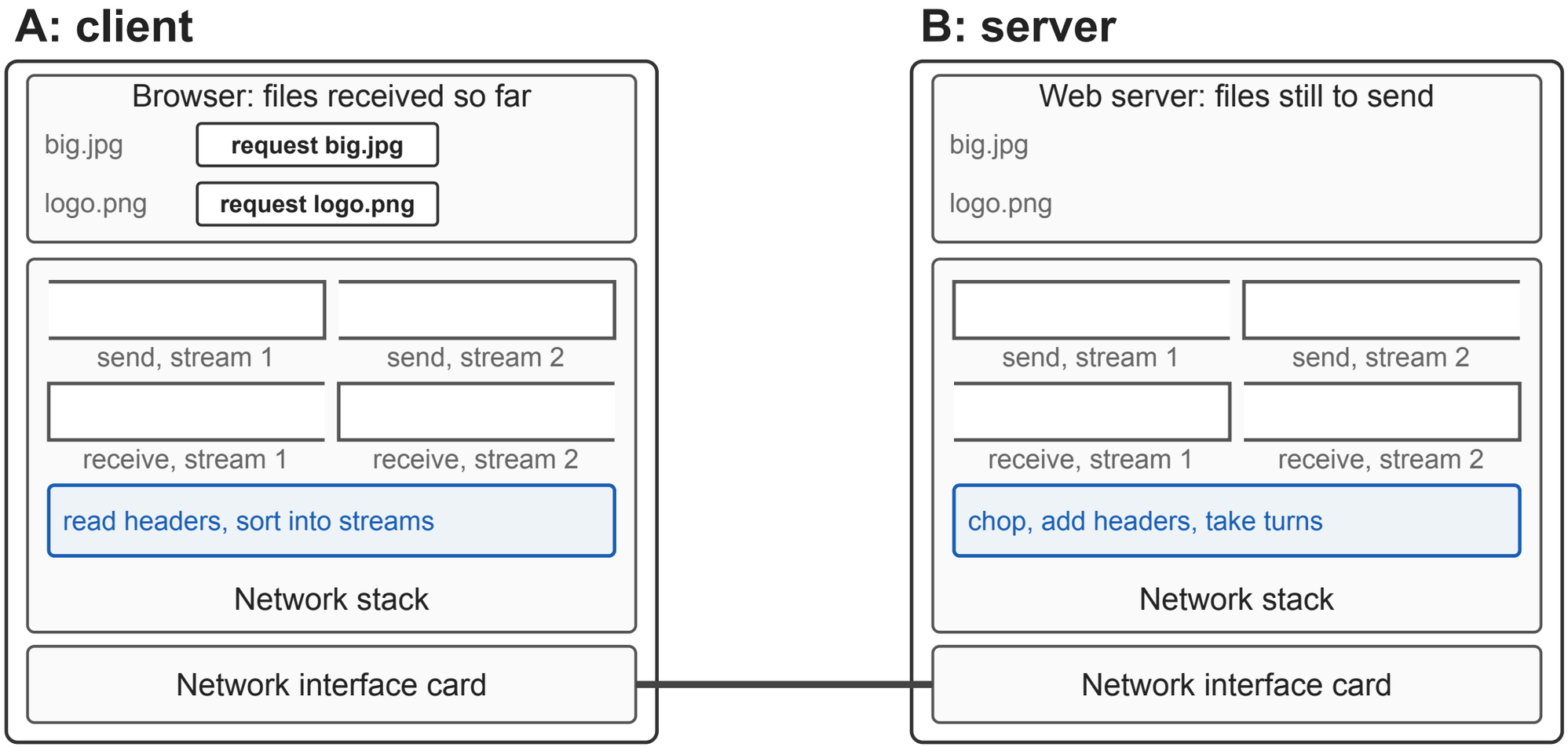
Fix 2: change what the layer below gives you

Instead of one stream of bytes, ask for as many as you like:

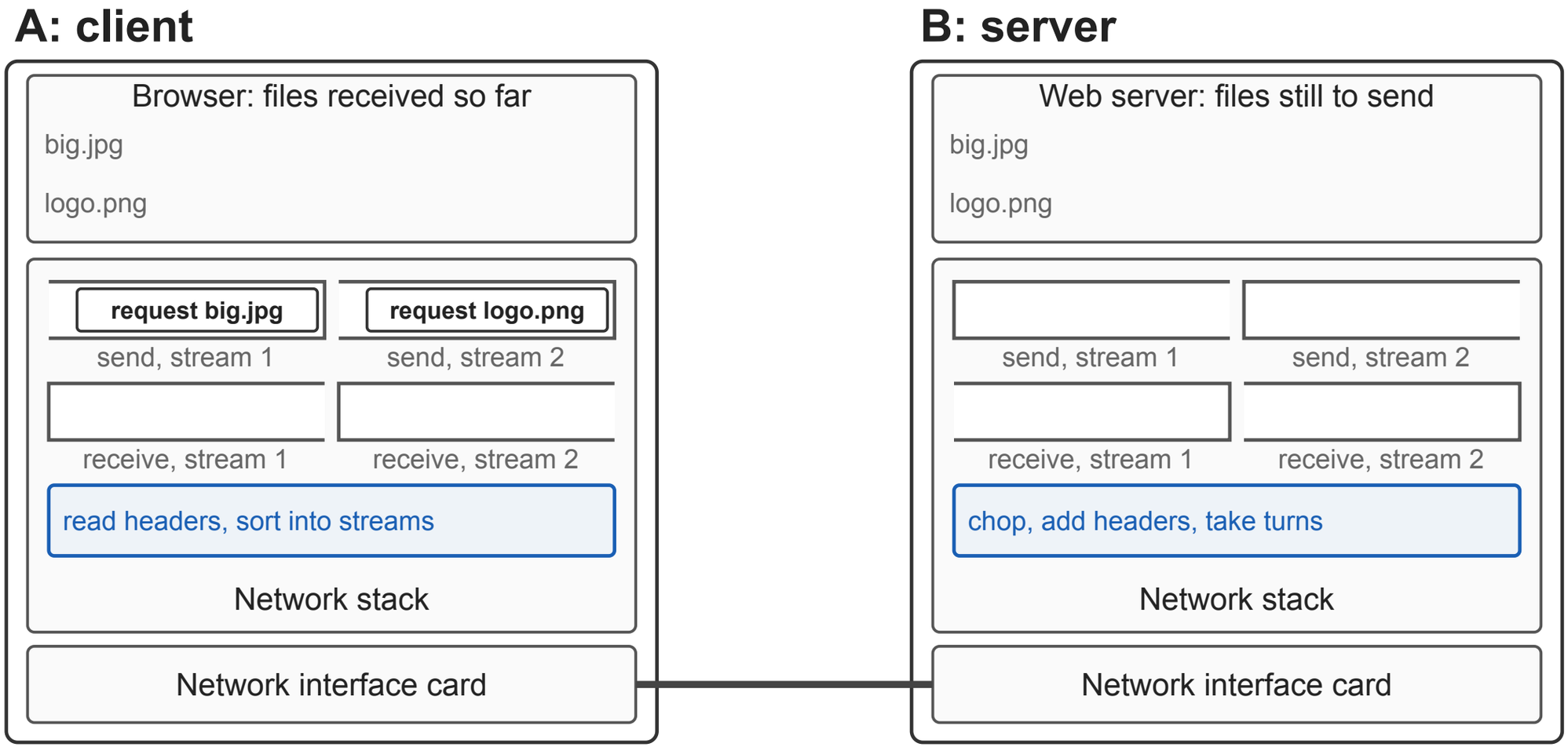
```
1 int  net_open_stream();
2 void net_send(int stream, uint8 *bytes, int n);
3 void net_recv(int stream, uint8 *bytes, int n);
```

- One stream per file
- Each file arrives on its own, without waiting for the others

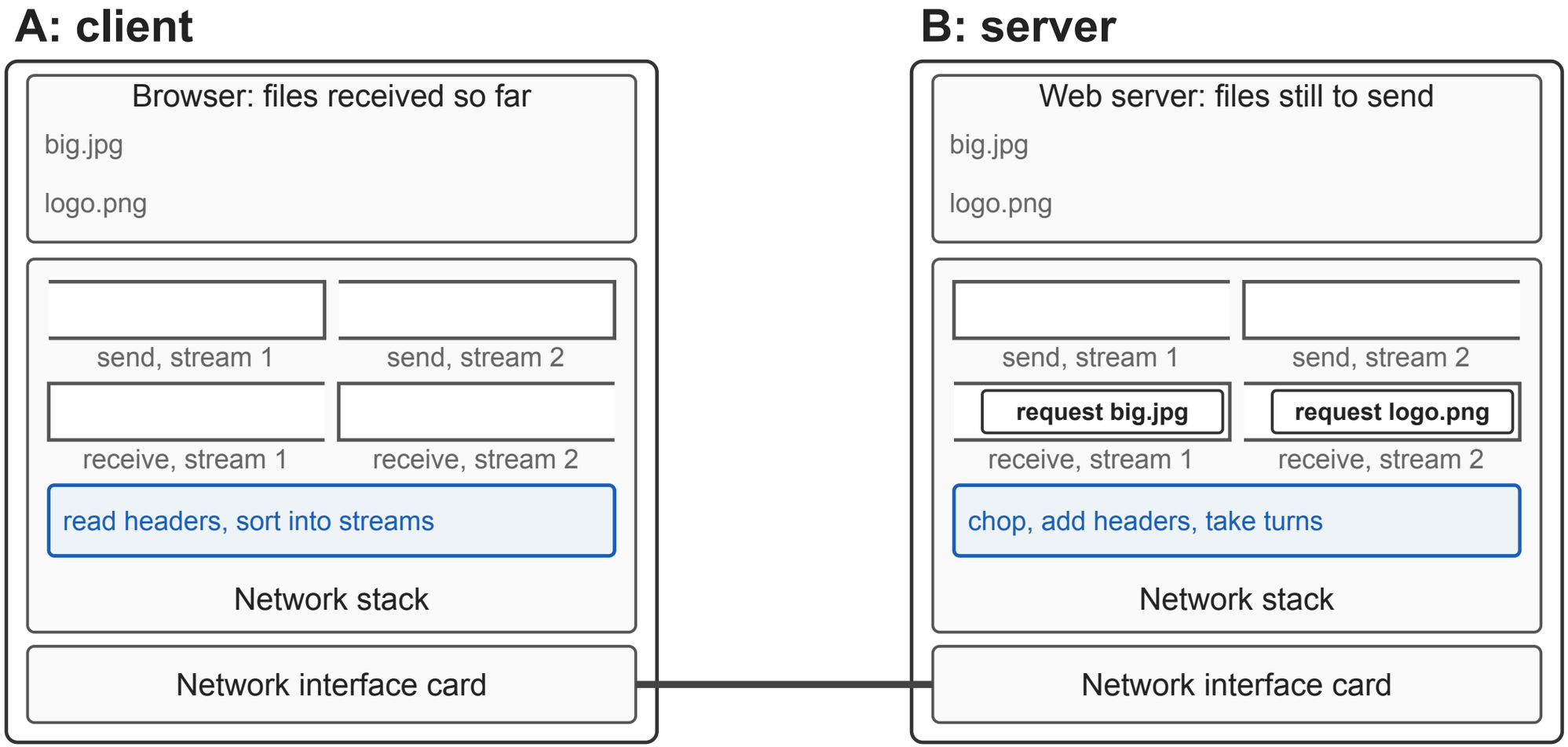
Fix 2: the network stack does the chopping



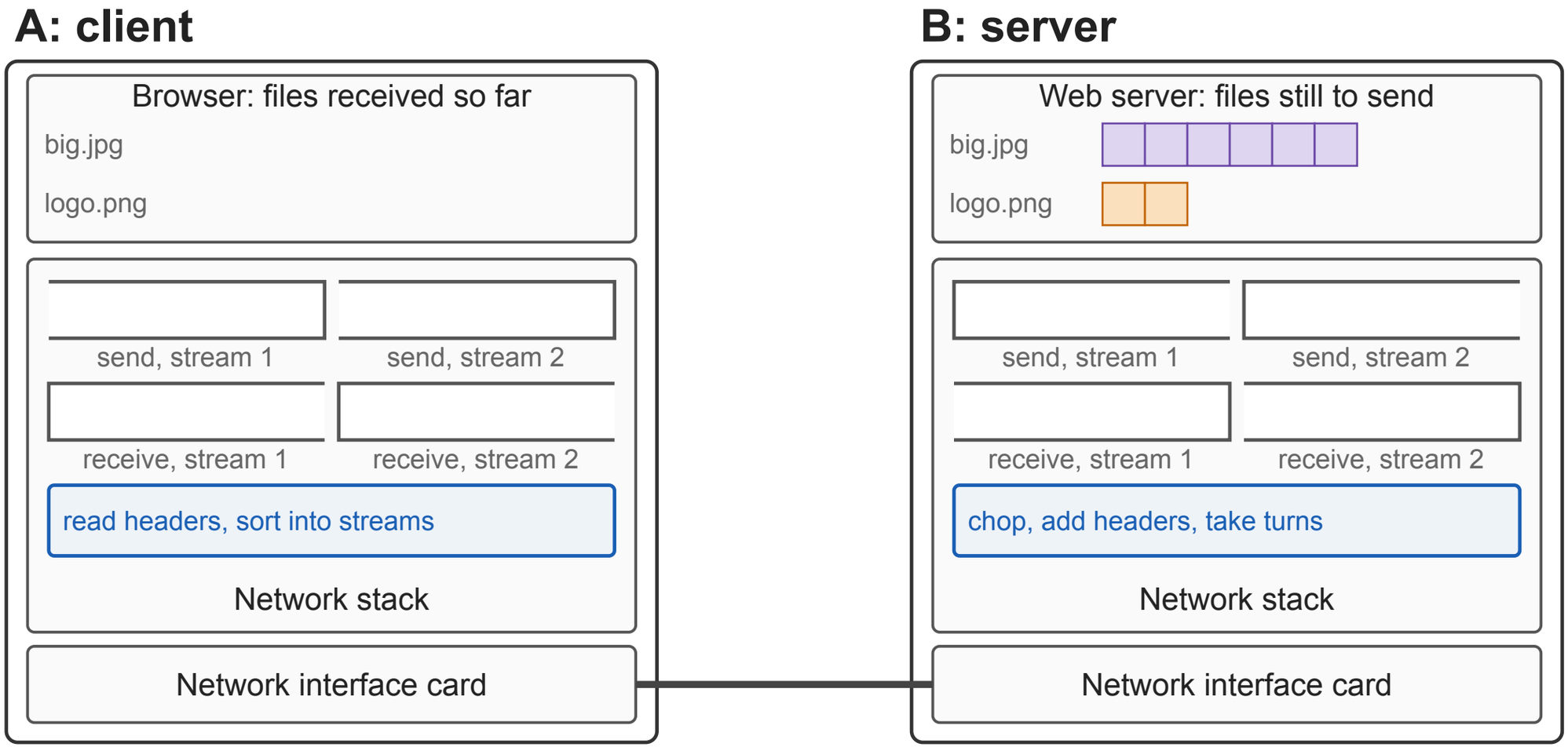
Fix 2: the network stack does the chopping



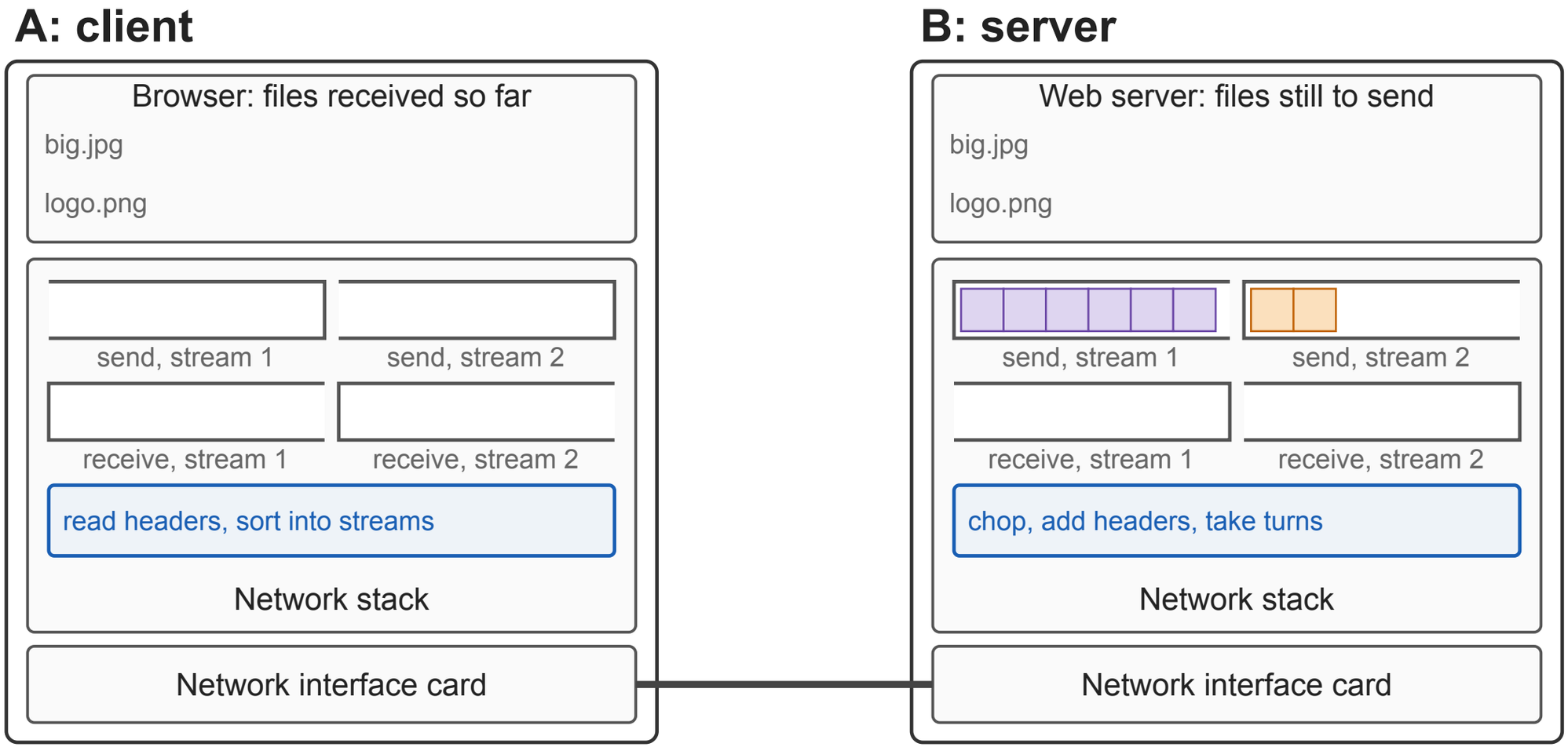
Fix 2: the network stack does the chopping



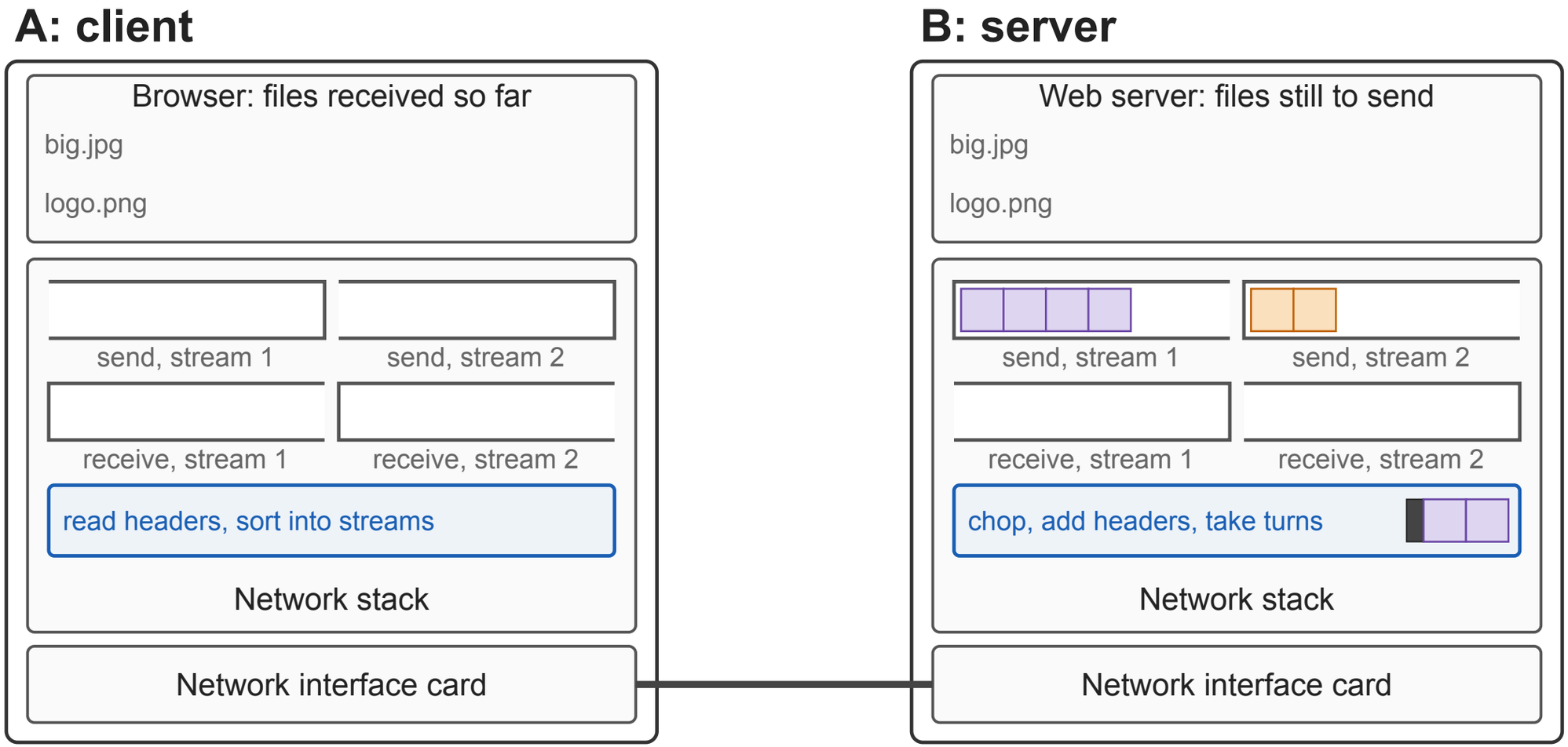
Fix 2: the network stack does the chopping



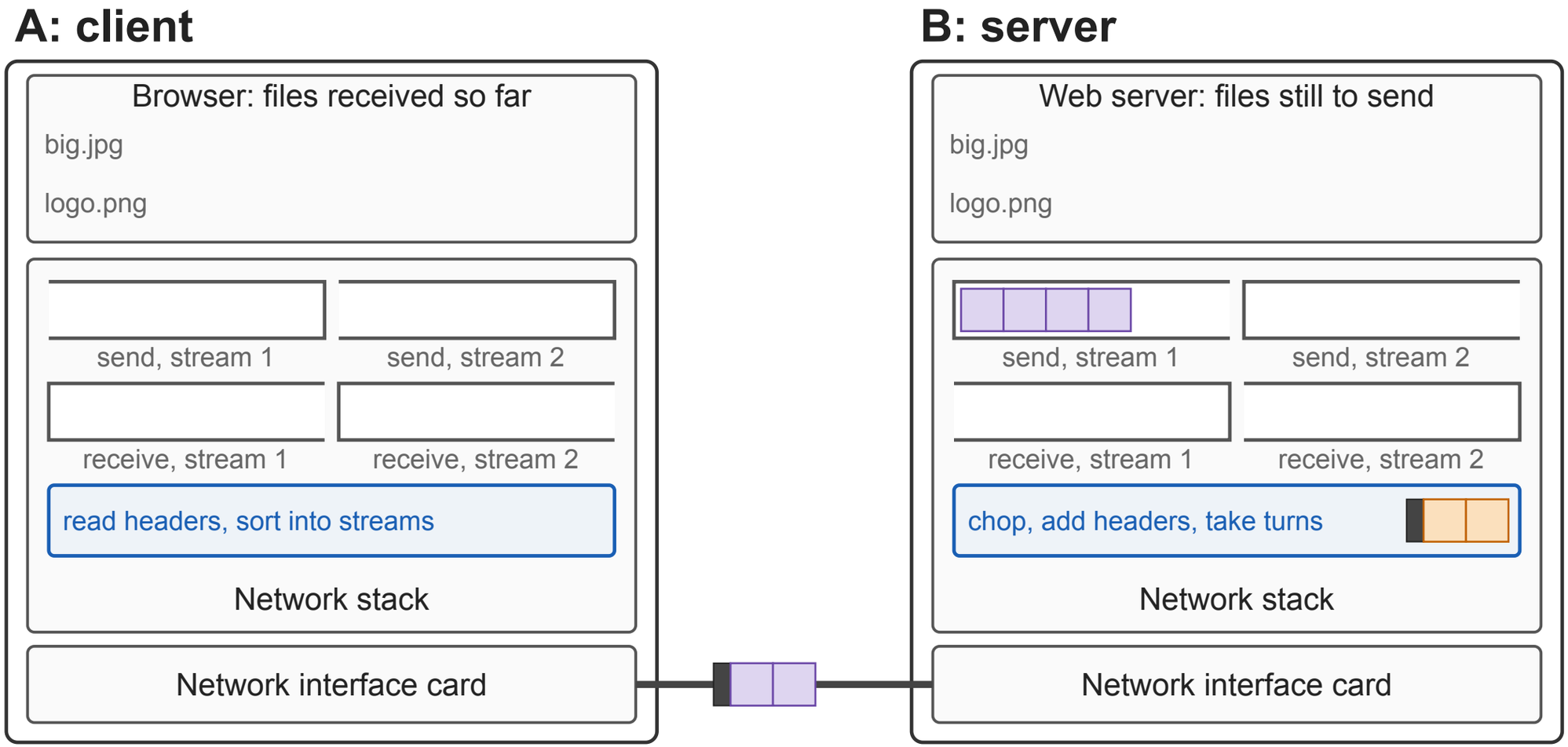
Fix 2: the network stack does the chopping



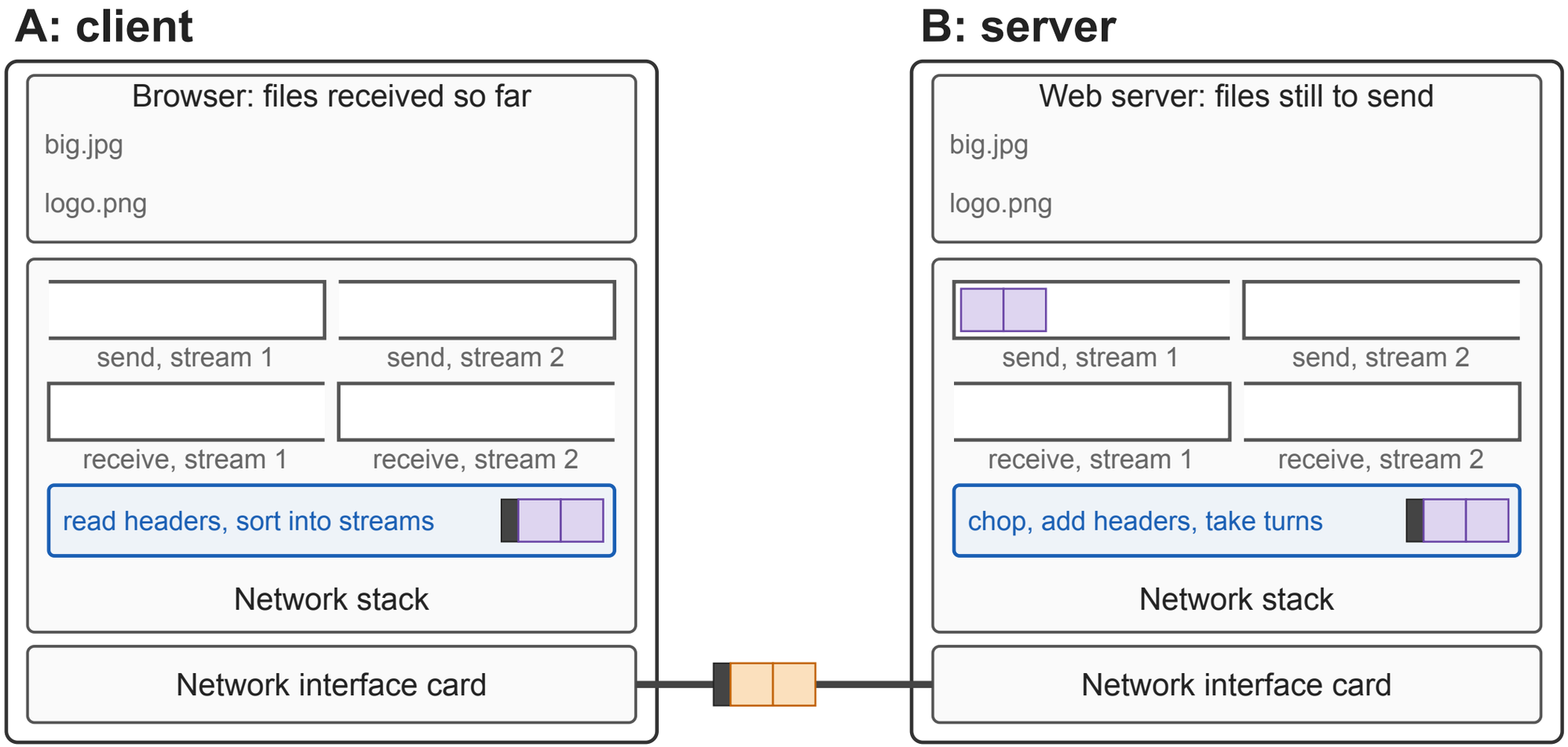
Fix 2: the network stack does the chopping



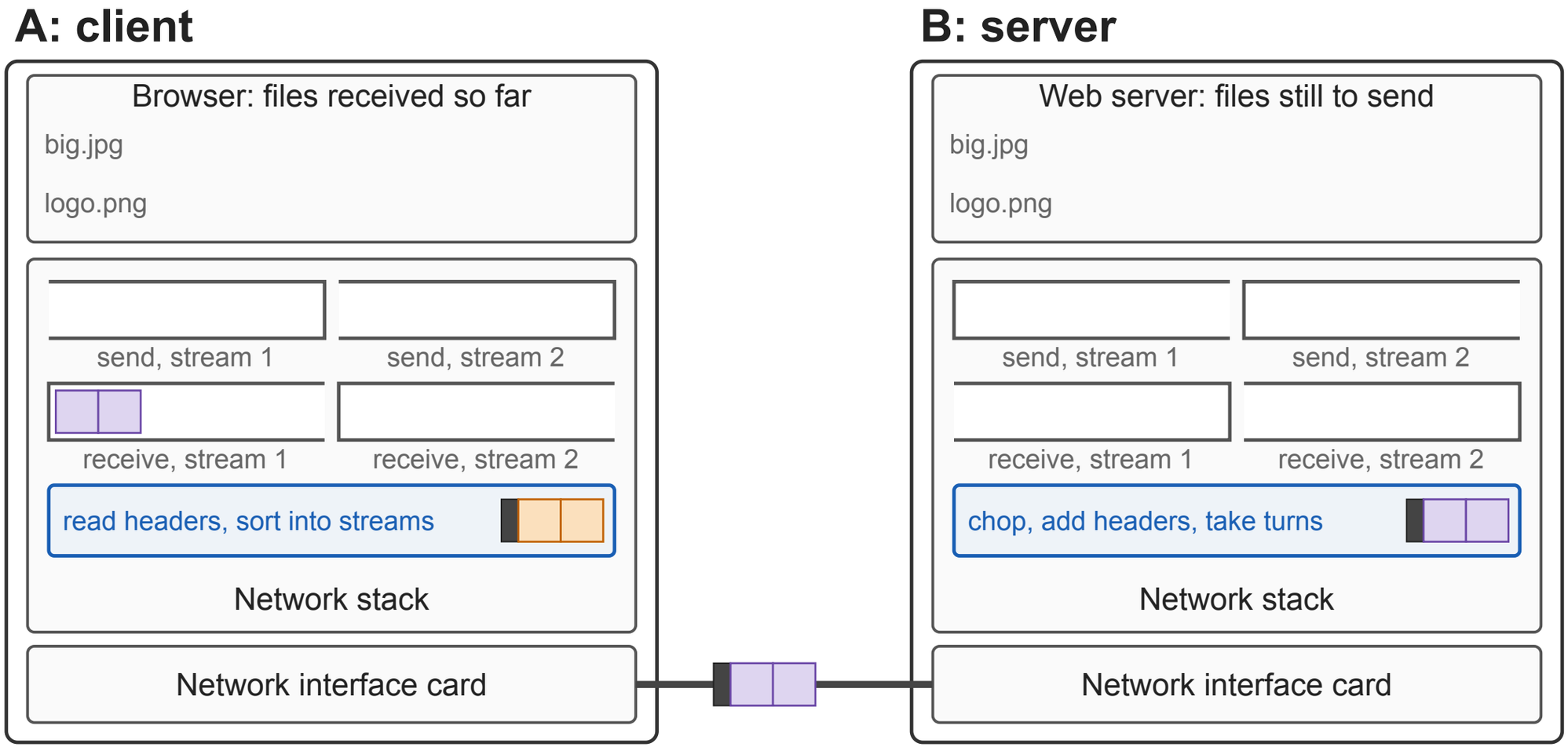
Fix 2: the network stack does the chopping



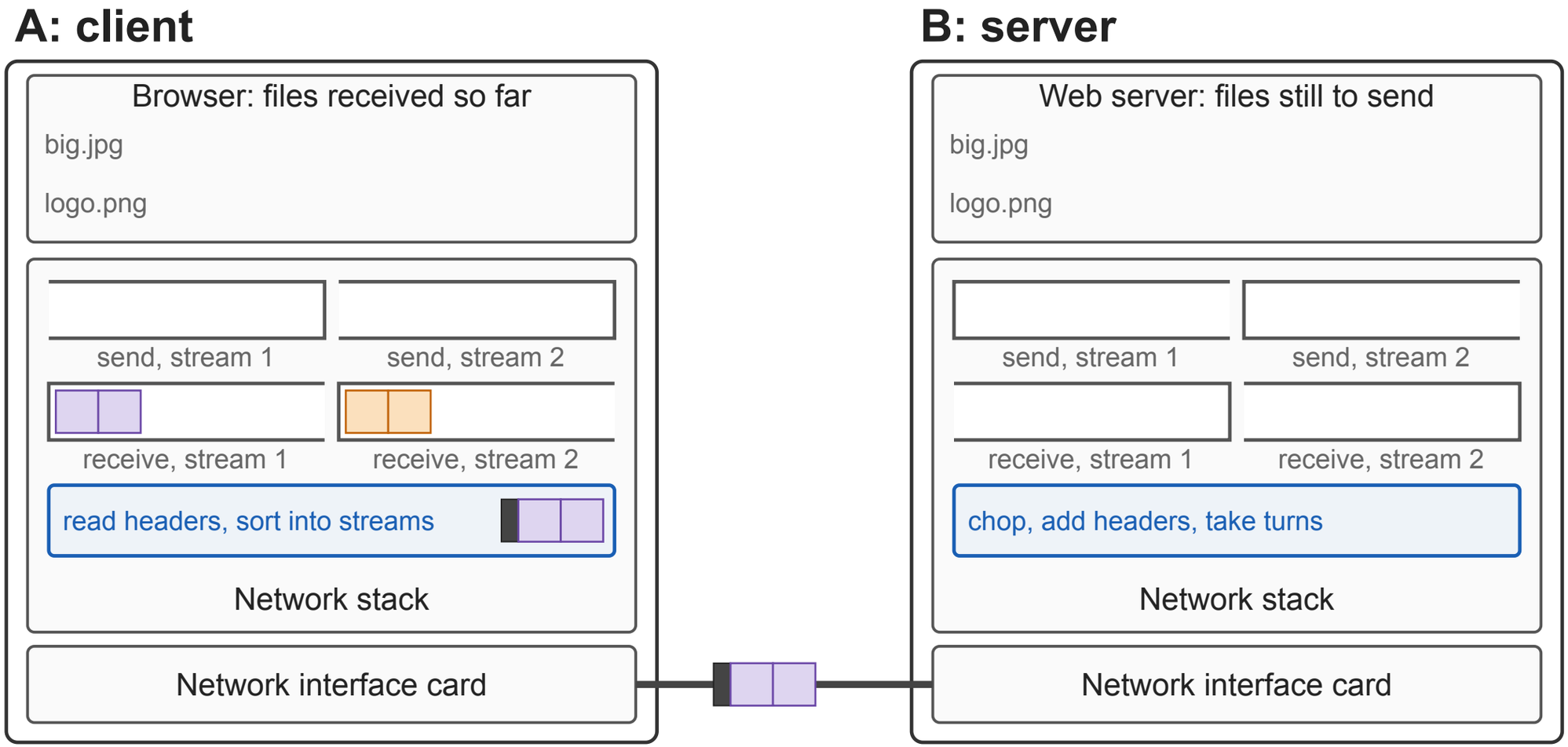
Fix 2: the network stack does the chopping



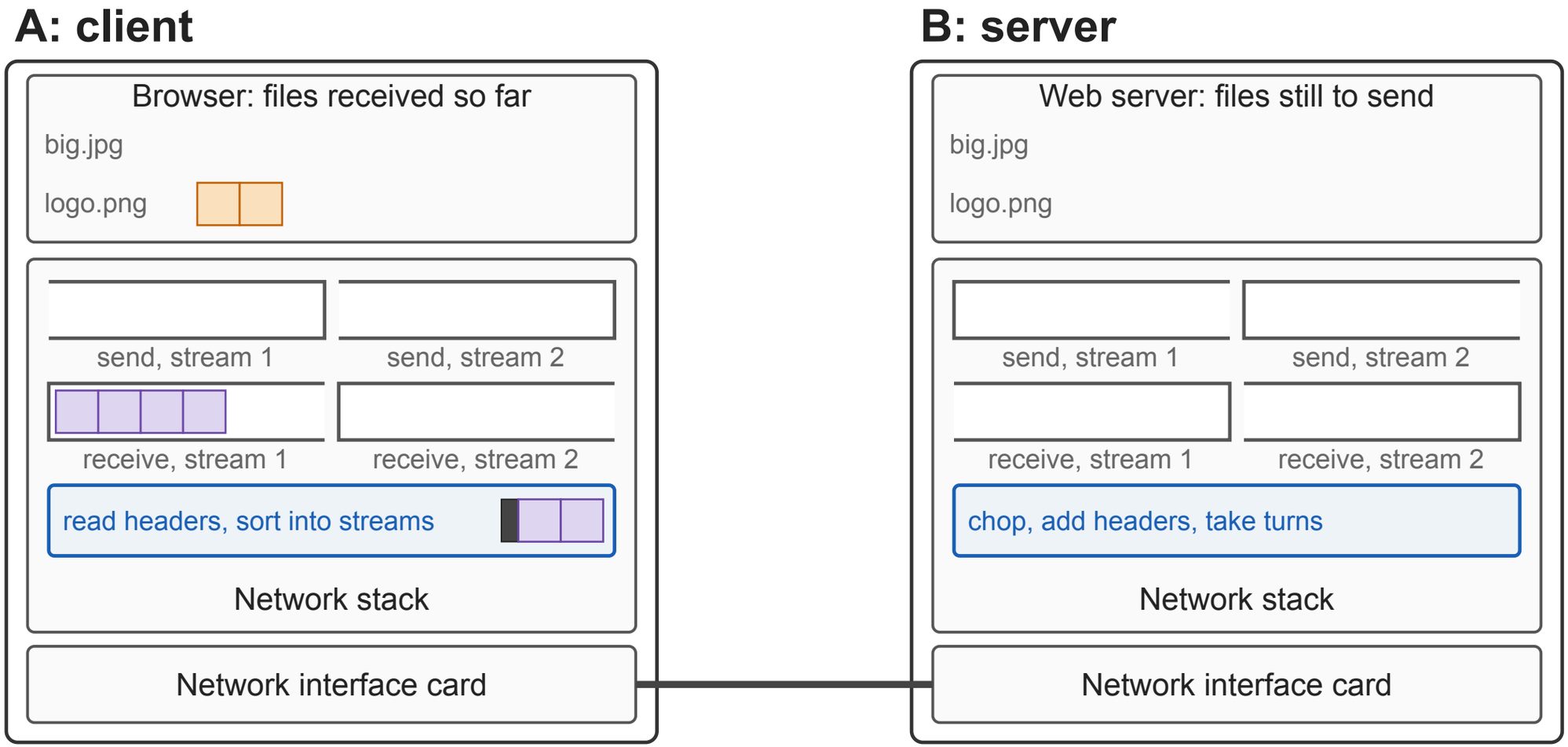
Fix 2: the network stack does the chopping



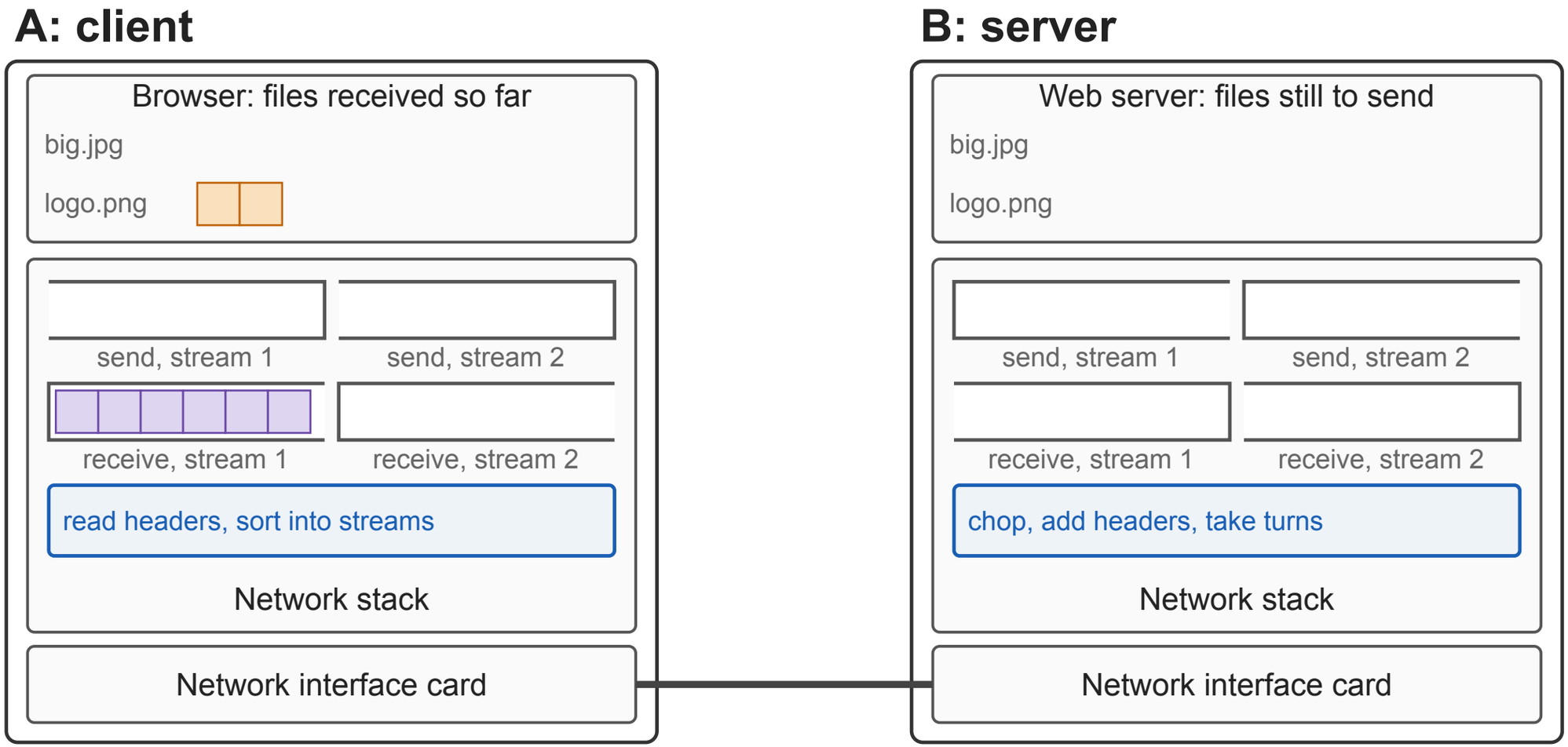
Fix 2: the network stack does the chopping



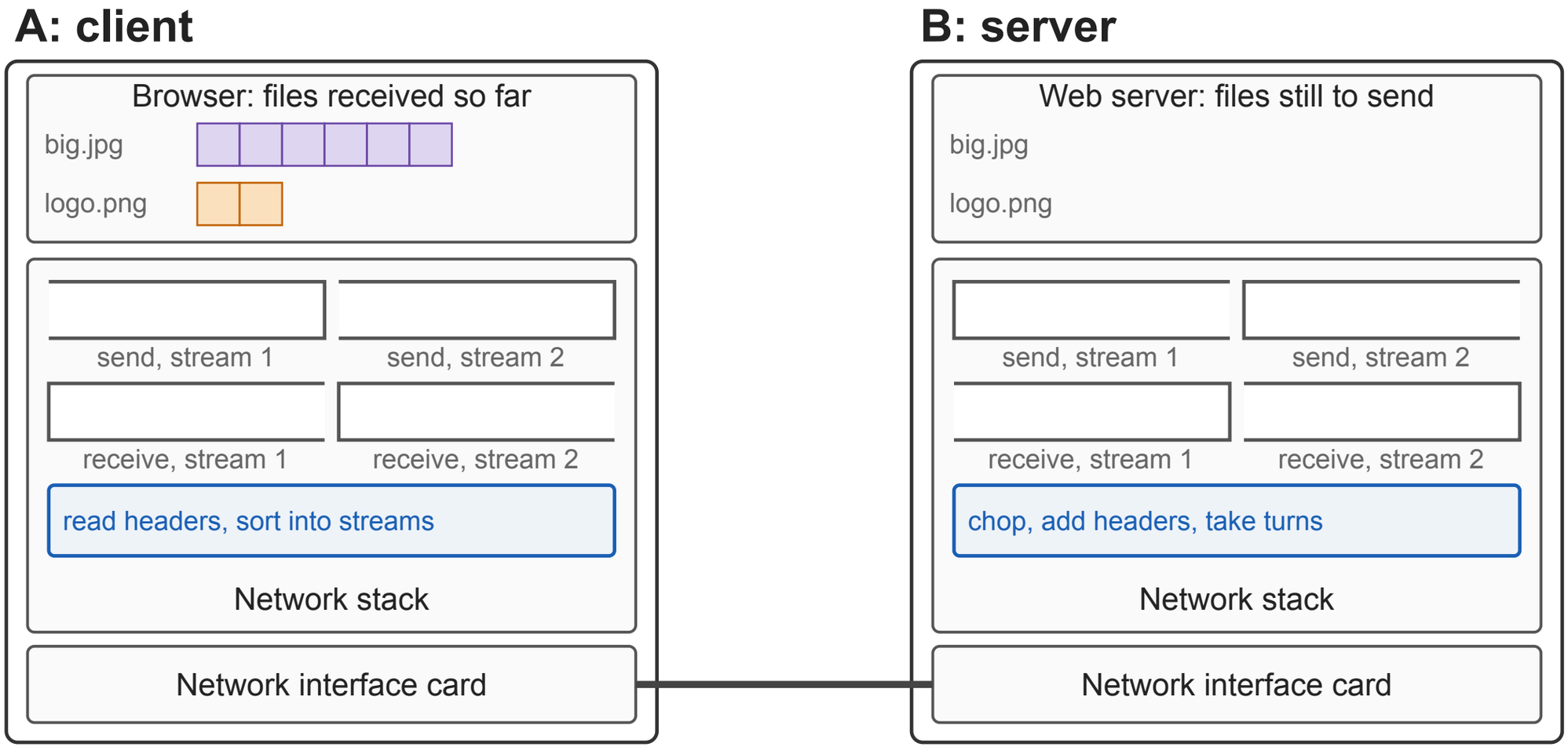
Fix 2: the network stack does the chopping



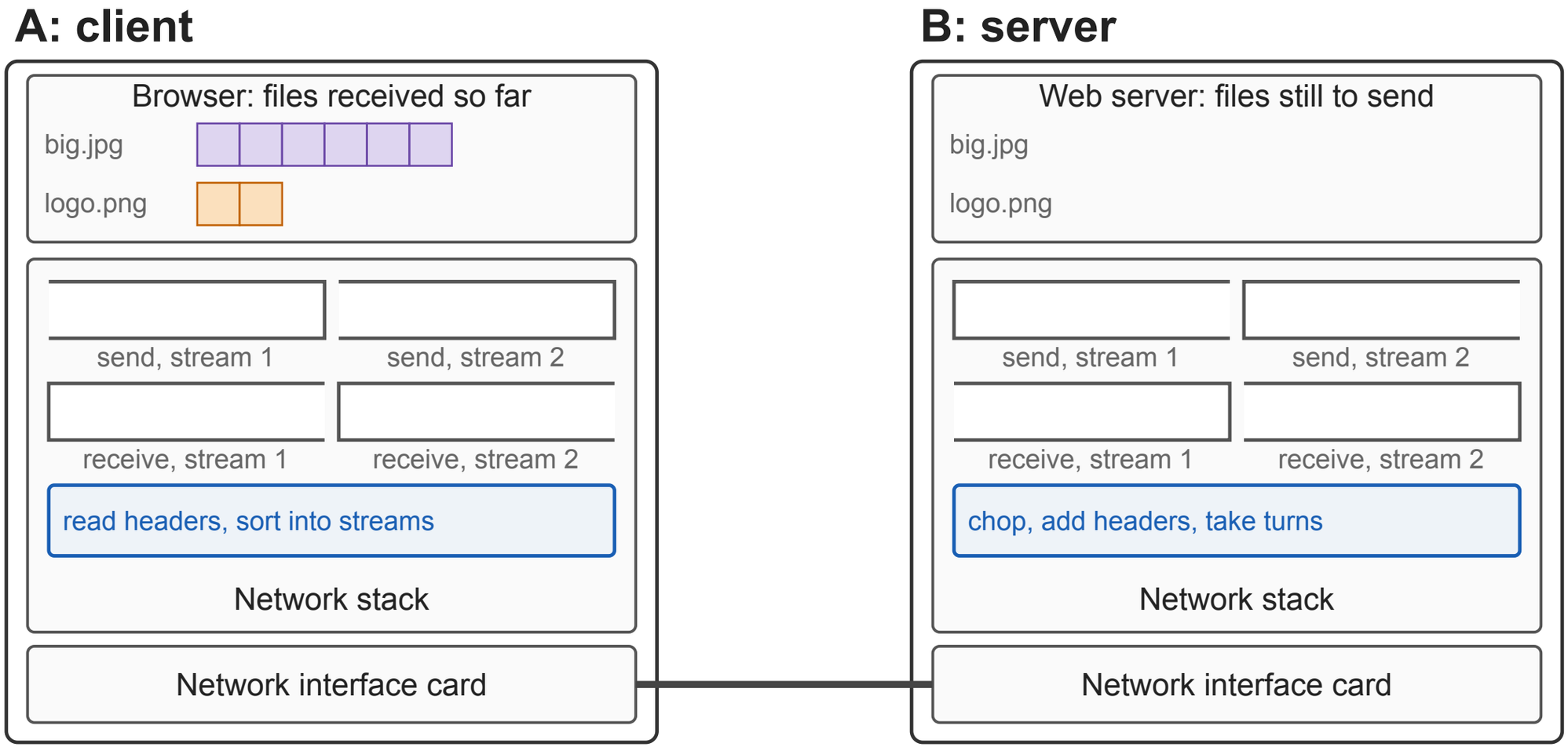
Fix 2: the network stack does the chopping



Fix 2: the network stack does the chopping



Fix 2: the network stack does the chopping



QUIC + HTTP/3, we'll come back to this at the end of Part 1

The two fixes use a conceptually similar idea

- Both fixes interleave the bytes of different messages
- Fix 1 is done entirely in the application. The network stack does not change.
- Fix 2 is done mostly in the network stack. The application only says which message goes on which stream.
- **Where** to put a function is a design decision that keeps coming back: in the application, in the network stack, or in the network devices in between?

Outline

- Building a web browser
- How long does it take to get a file?
- Pages with more than one file
- **Takeaways**

Takeaways

- Protocols let independent devices communicate. A protocol defines:
  - the format and order of the messages they exchange
  - what each side does when it sends or receives one
- End-to-end delay (direct link):
  - processing + transmission + propagation
- A rule of thumb:
  - large transfers are limited by throughput
  - small transfers are limited by latency

Takeaways (cont.)

- Head-of-line blocking:
  - when messages share one stream, a big one at the front holds up the small ones behind it
- The same function can be implemented in different places along the end-to-end path:
  - interleaving messages in the application (Fix 1) or in the network stack (Fix 2)