Writing TCP-based Servers

Chris Brown

In This Module …

Client-side and server-side operations

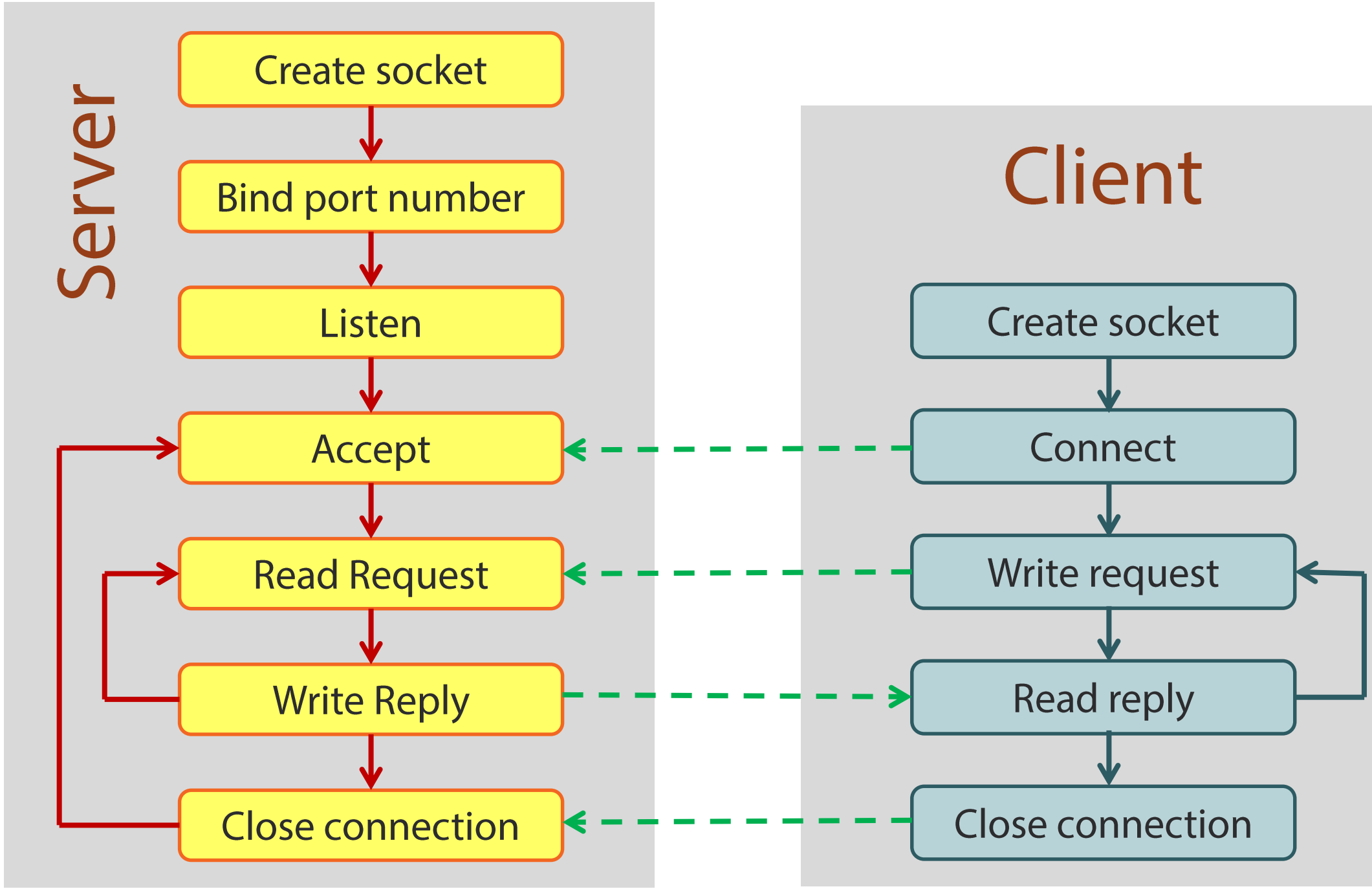
Key data structures

Key sockets system calls

Python as an alternative language

Demonstration:

A 'rot13' server


Server
Create socket
Bind port number
Listen
Accept
Read Request
Write Reply
Close connection
Client
Create socket
Connect
Write request
Read reply
Close connection

The sockaddr_in Structure

```
struct in_addr {
  inaddr_t  s_addr;
}
```

← 32-bit IP Address

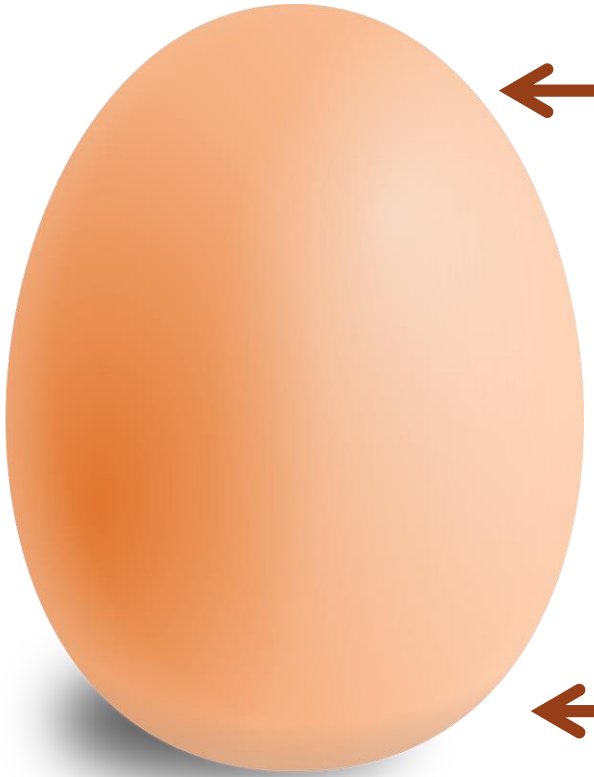
```
struct sockaddr_in {
  sa_family_t    sin_family;
  in_port_t      sin_port;
  struct in_addr sin_addr;
  unsigned char __pad[..];
}
```

← Address family (AF_INET)

← IPV4 address

← Pad to size of generic sockaddr struct

Little-endian vs. Big-endian

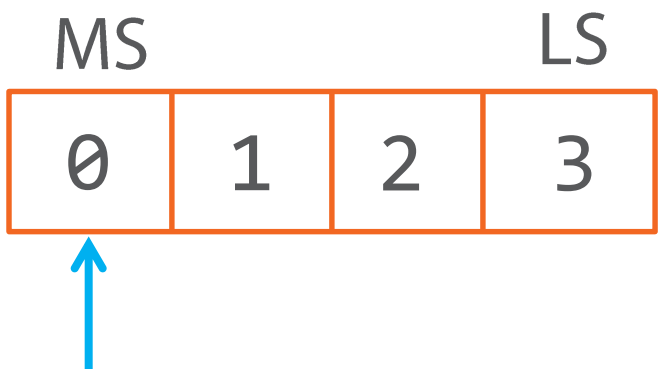


Little-Endian

Big-Endian

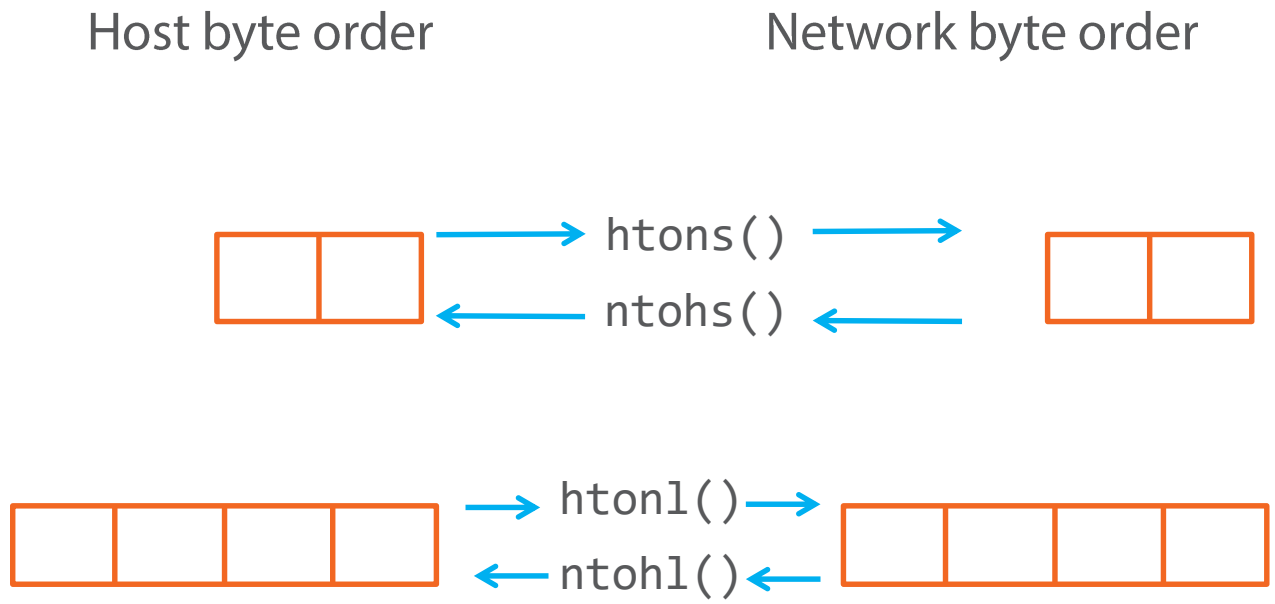
Network Byte Order

Socket addresses must be in network byte order (big-endian)

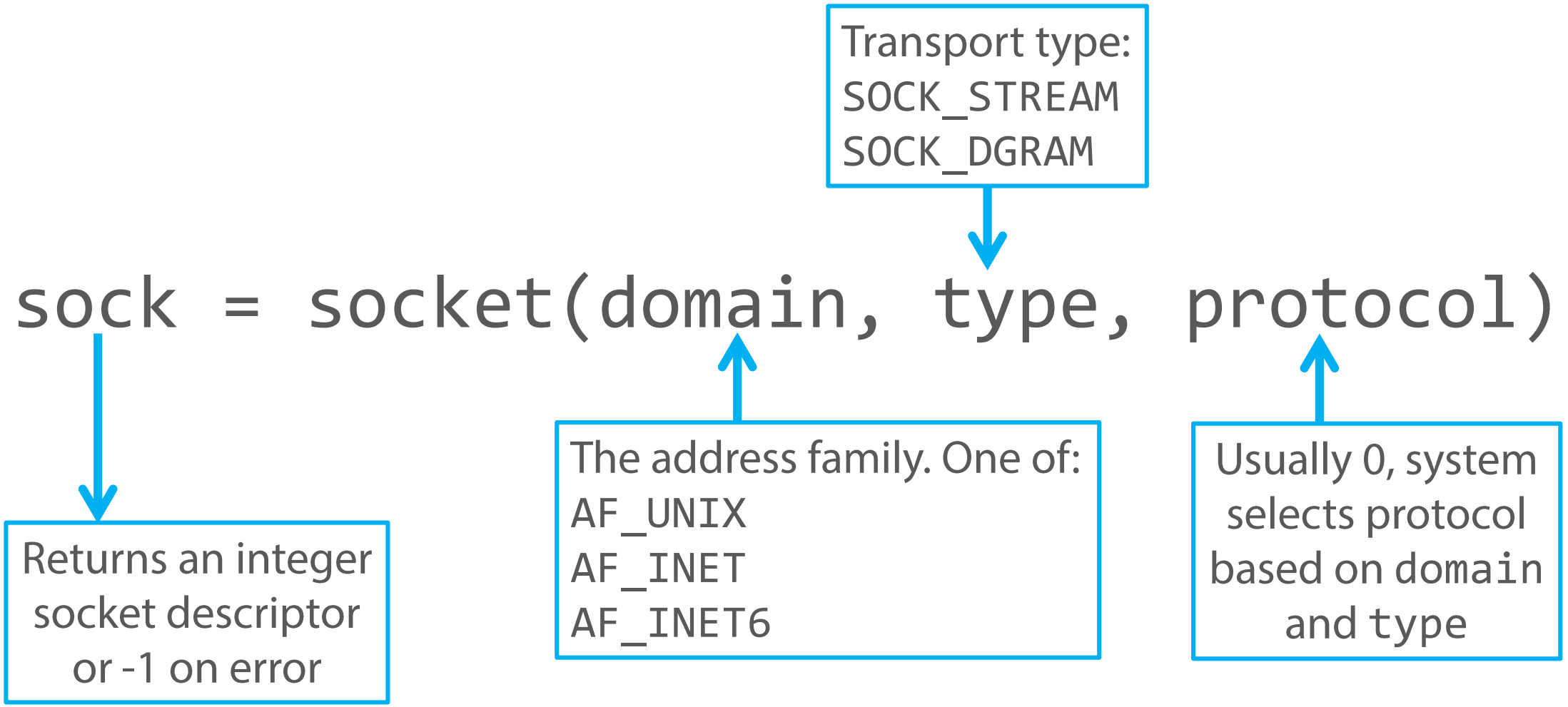


Most significant byte stored at lowest memory address, sent across network first

Macros convert to/from the machine's internal byte order.



Creating a Socket



Setting the Local Address

```
#define SERVER_PORT     1067
struct sockaddr_in      server;

server.sin_family      = AF_INET;
server.sin_addr.saddr = htonl(INADDR_ANY)
server.sin_port        = htons(SERVER_PORT);

bind(sock, (struct sockaddr *)&server, sizeof server);
```

Waiting for Business

```
struct sockaddr_in client;
int fd, client_len;

listen(sock, 5);

client_len = sizeof(client);
fd = accept(sock, (struct sockaddr *)&client, &client_len);
```

Set up a queue for pending connections

Connection descriptor

Rendezvous descriptor

Client's endpoint address returned here.

Talking to the Client

- The connection descriptor returned from `accept()` looks like an open file
  - Supports `read()` and `write()` system calls

```
unsigned char buf[1024];
int count;
while ((count = read(fd, buf, 1024)) > 0)
{
  rot13(buf, count);
  write(fd, buf, count);
}
```

Demonstration: The rot13 Server

Doing It in Python

- Python's `sockets` module exposes the traditional sockets API
- Python language features hide some of the messier stuff
  - Automatic buffer allocation on receiving
  - Implicit buffer length management on sending
  - Optional arguments to methods
  - Passing and returning tuples
  - Exception handling replaces error checking
- A socket is a *class*
  - Methods `bind(), listen(), accept() ...` expose the API

Doing It in Python


```
s = socket(AF_INET, SOCK_STREAM)
s.bind(('', 1068))
s.listen(5)

while True:
    client,addr = s.accept()
    rot13_service(client)
    client.close()
```

Empty address string means `INADDR_ANY`

Tuple assignment:
(connection descriptor, client endpoint address)

Demonstration: Python Server

Moving Forward …

In this module:

- Sequence of operations
- Key data structures
- Key system calls
- The rot13 server
- Doing it in Python

Coming up in the next module:

- The client side of TCP/IP