Writing UDP-based Servers and Clients

Chris Brown

In This Module ...

Client and server-side operations using UDP

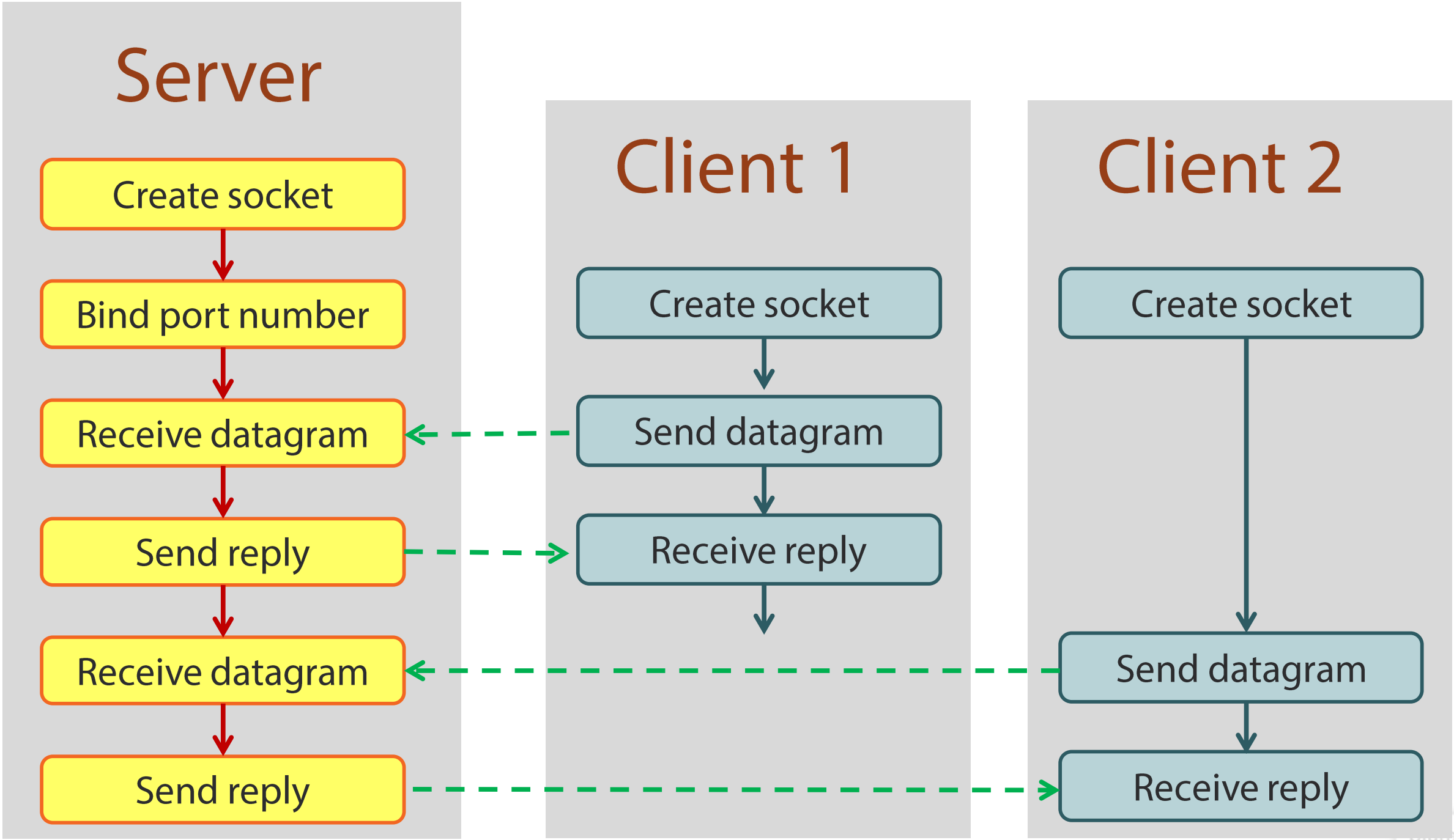
UDP sockets API

A binary application protocol (TFTP)

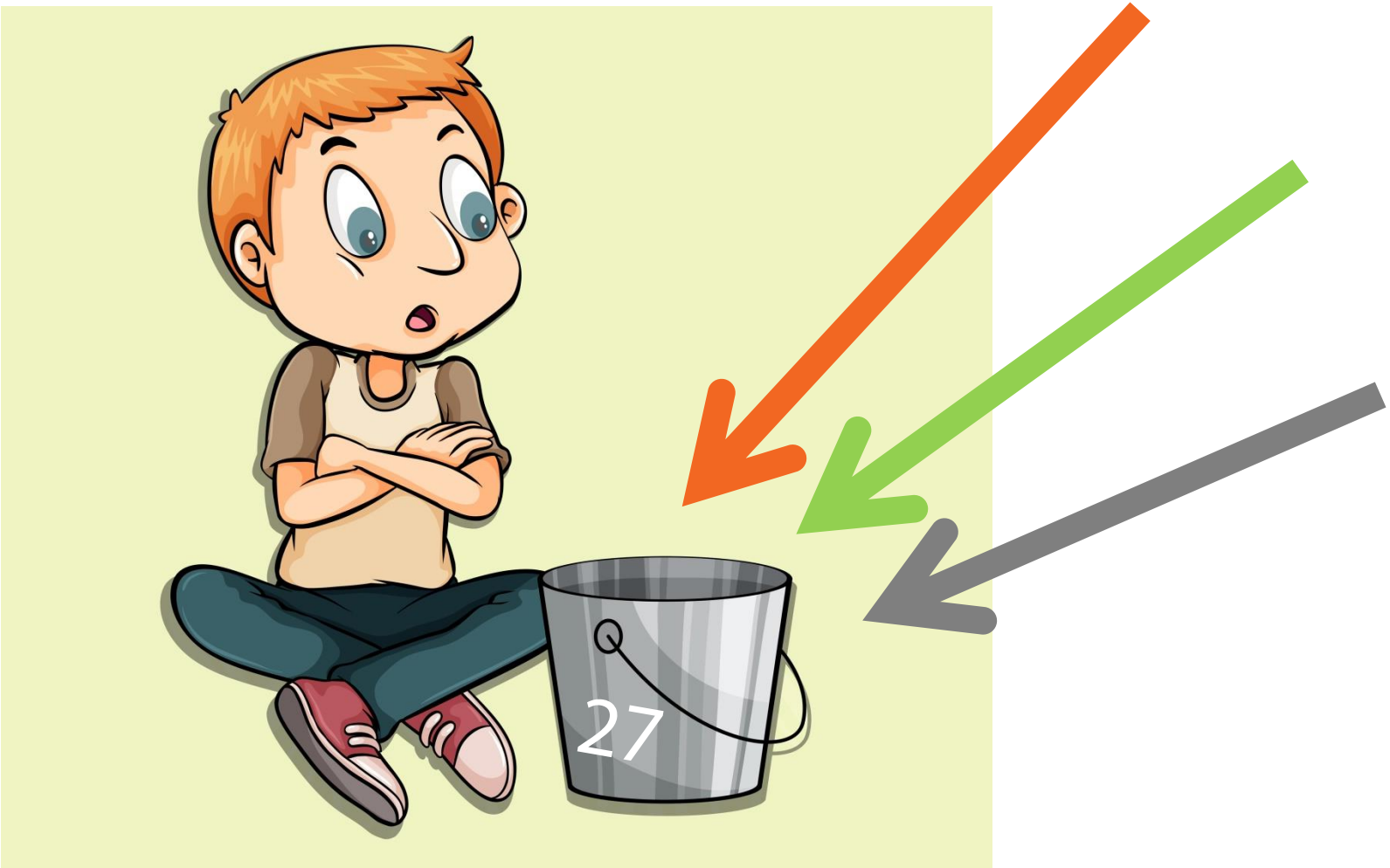
Demonstration: rcat – a TFTP client

Broadcasting with UDP

Demonstration: A UDP broadcast application

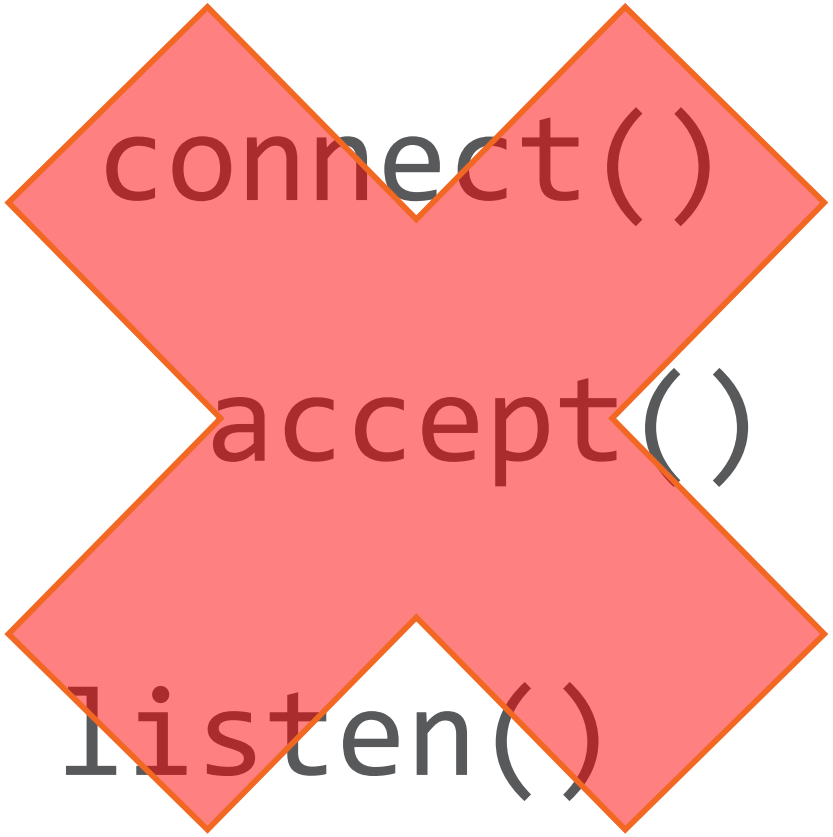

Server
Create socket
Bind port number
Receive datagram
Send reply
Receive datagram
Send reply
Client 1
Create socket
Send datagram
Receive reply
Client 2
Create socket
Send datagram
Receive reply

The Bucket Analogy



The Postbox Analogy

Each message is individually addressed



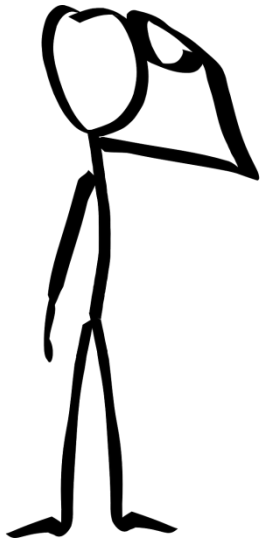
struct sockaddr_in{..}

Coping with Unreliability

UDP is "unreliable"
— packets may get dropped
— or may be mis-ordered

Is the underlying network "reliable enough"?

?



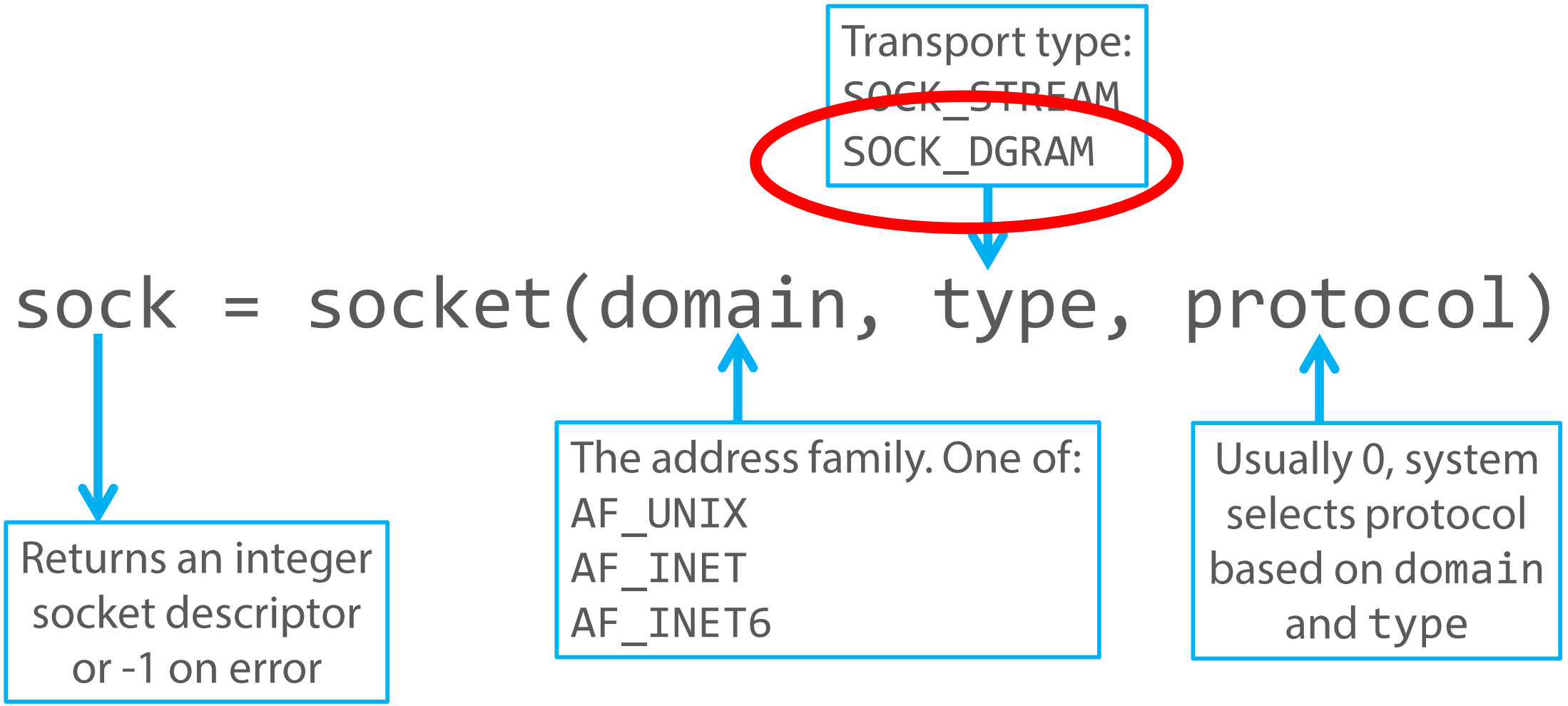
Can we tolerate occasional dropped packets?
— Catastrophic failure of application?
— Gradual degradation of performance?

?

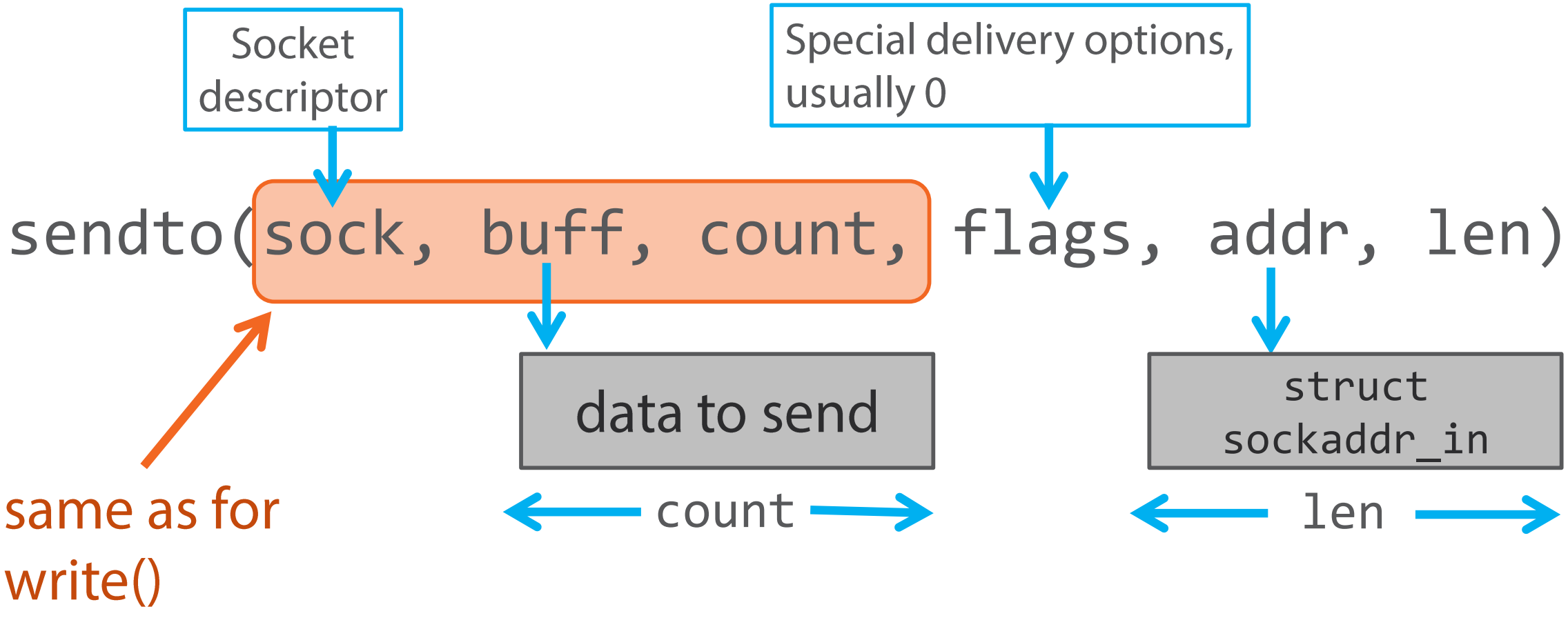
Handle acknowledgements, timeouts and re-transmissions within the application protocol

?

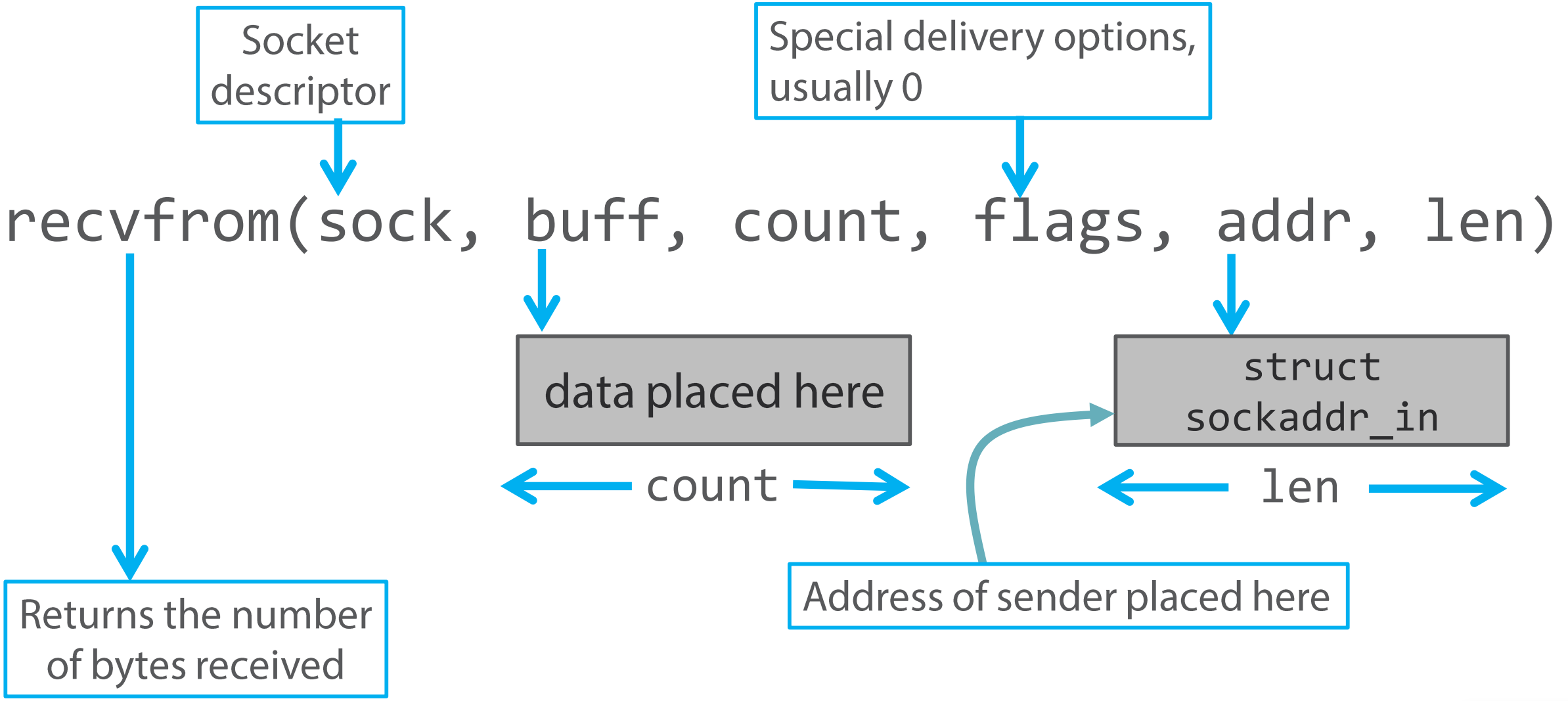
Creating a Socket



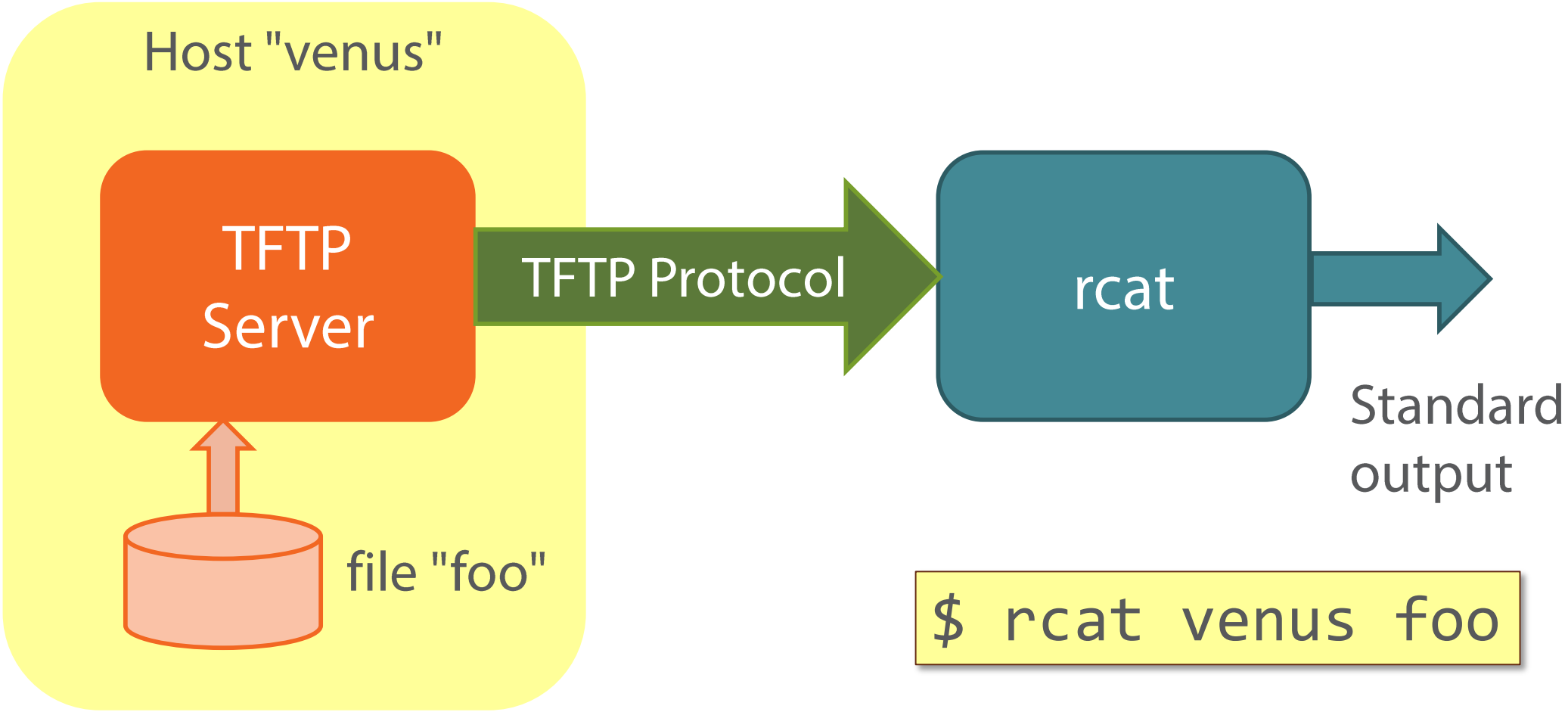
Sending a Datagram



Receiving a Datagram



Our UDP Demonstration Client



Protocols

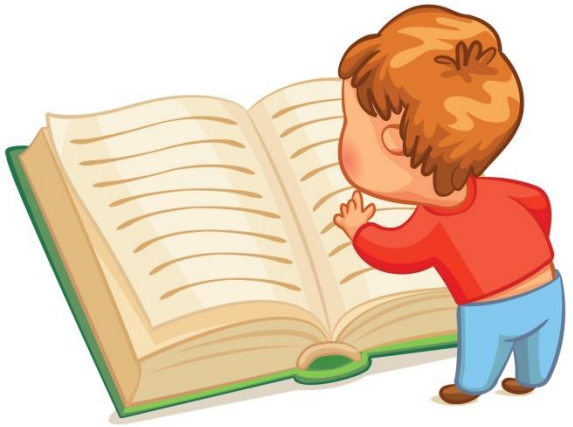
protocol – a set of rules governing the exchange or transmission of data between devices

Text-based:
- Based on text strings (human readable)
- HTTP, Telnet, SMTP, ...

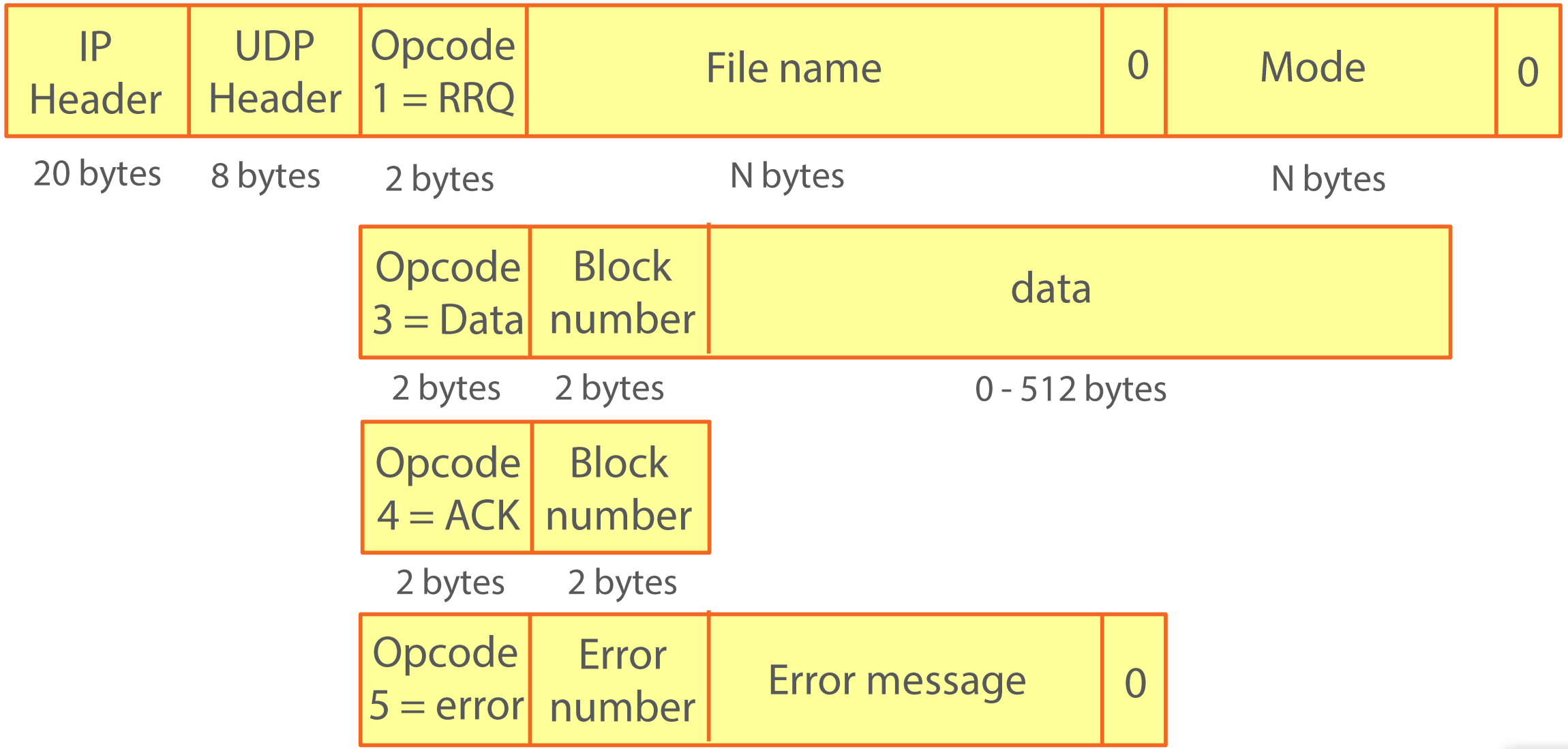
Binary:
- Based on data structures (machine-readable)
- IP, TCP, UDP, TFTP, ...

Introducing TFTP

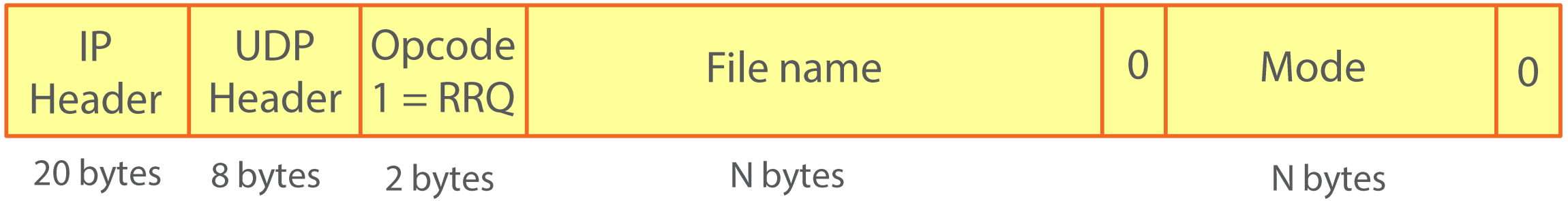
- Trivial File Transfer Protocol
- Simple design
  - Small code footprint
- Useful to download kernel image, etc. to small systems
- No authentication or access control
  - Server often confined to `/tftpboot` directory
- Consult RFC1350 for the full story



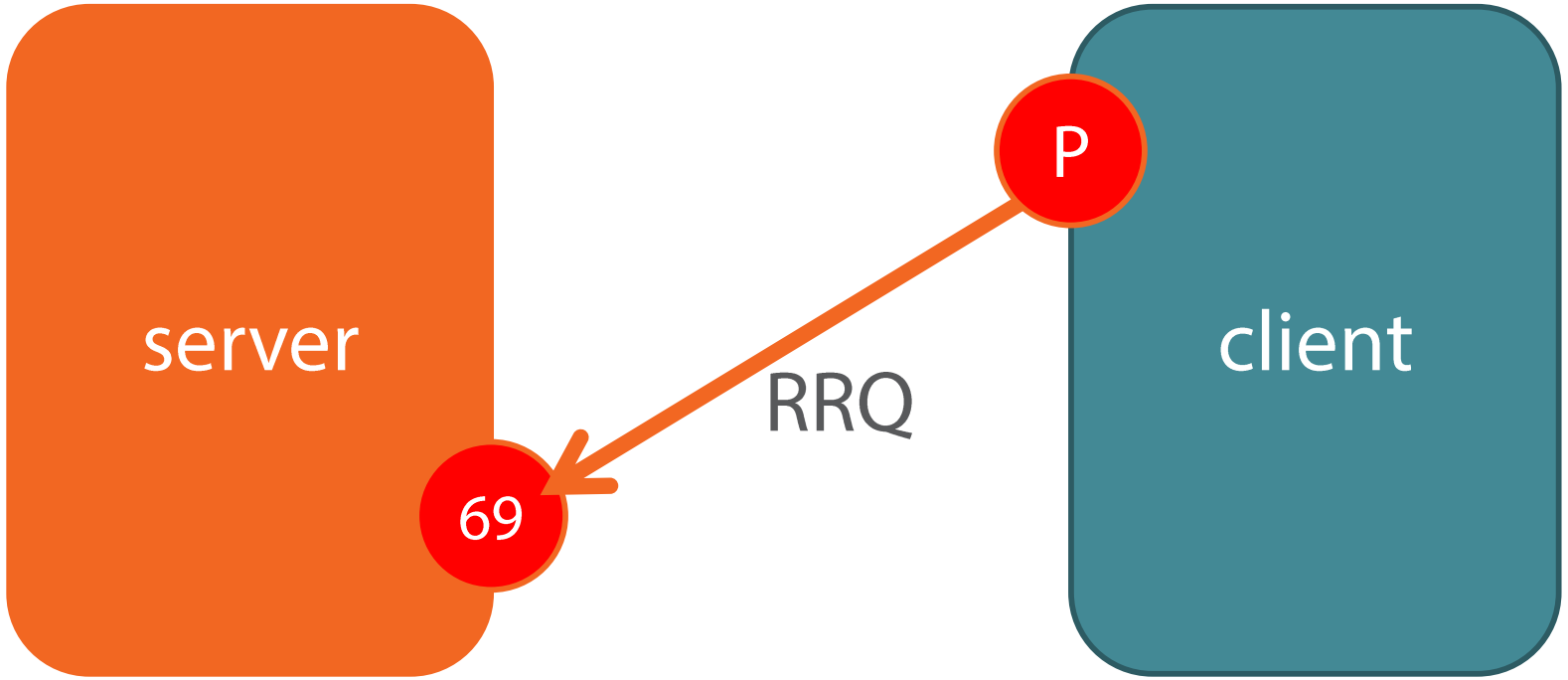
TFTP Packet Formats



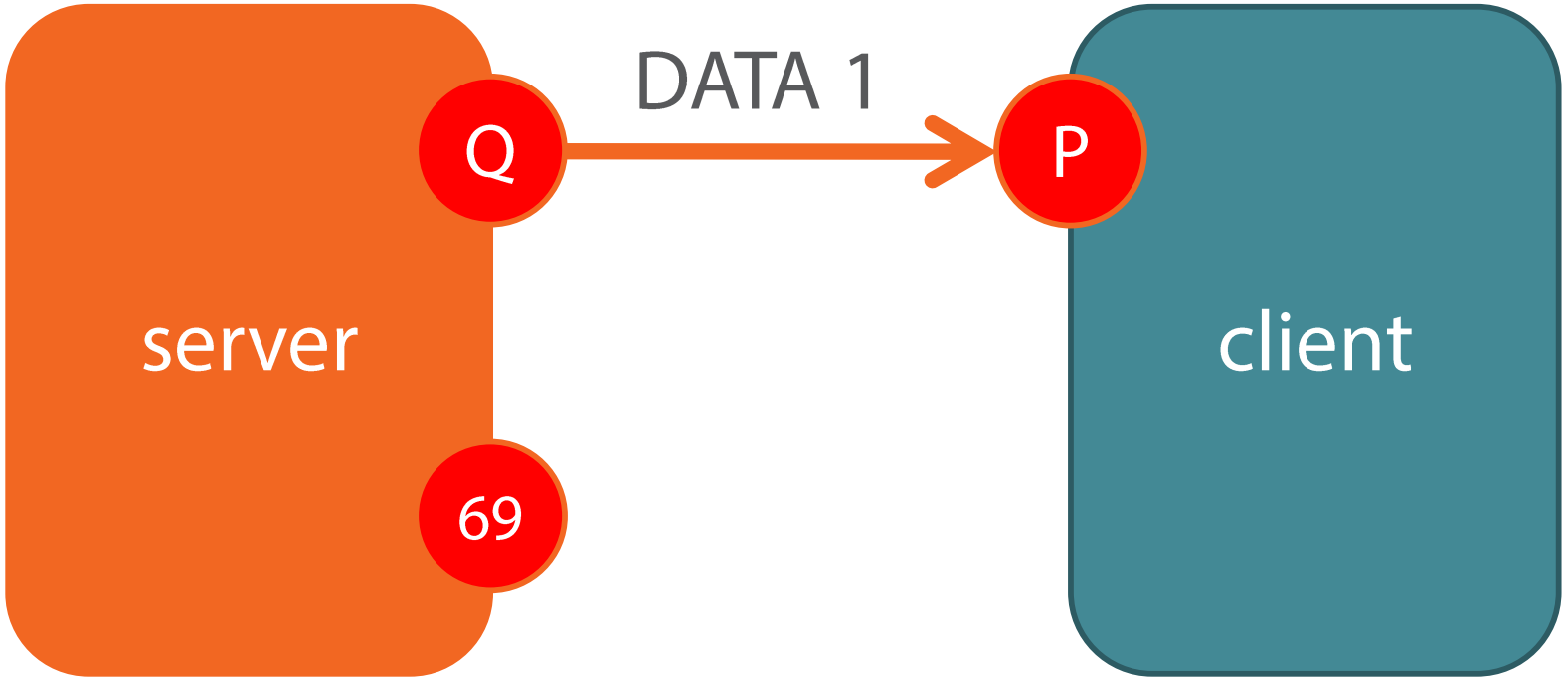
TFTP Packet Formats



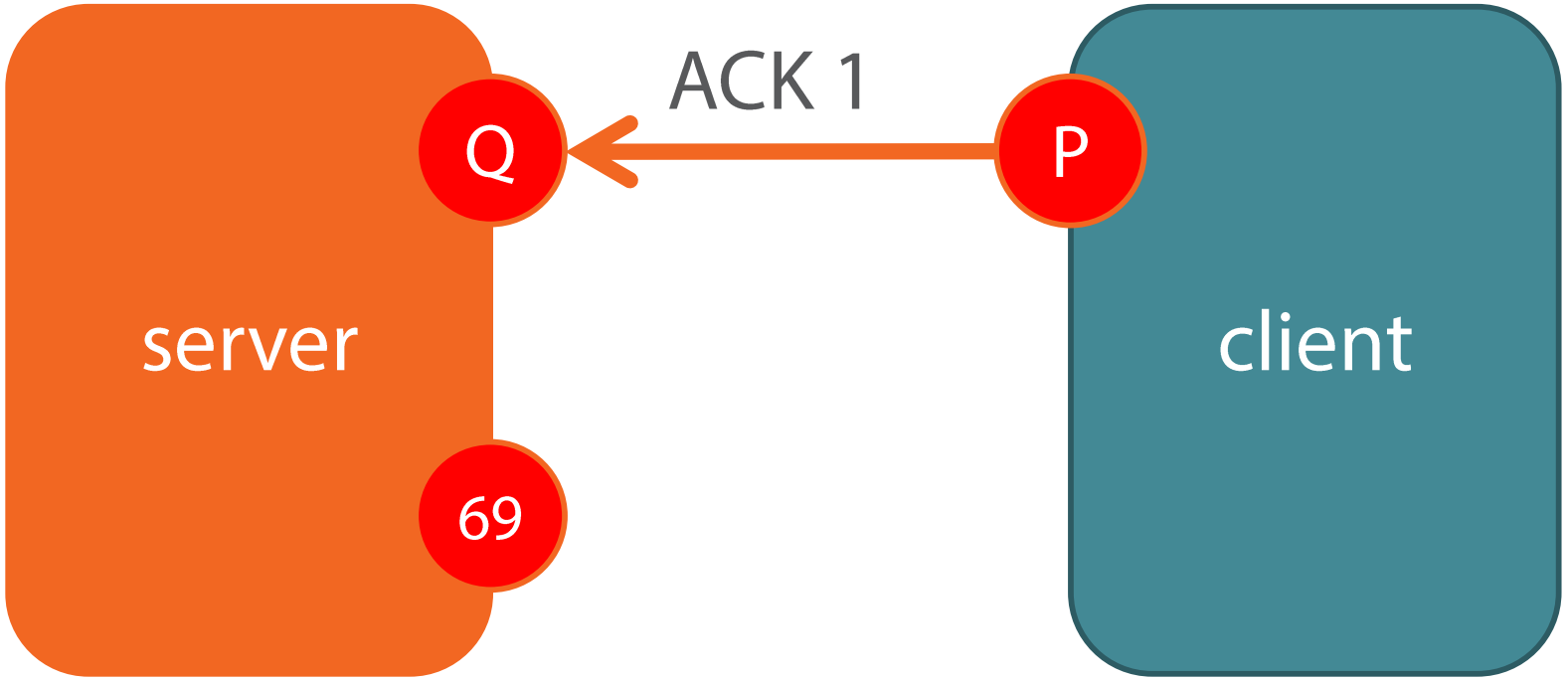
TFTP Operation



TFTP Operation



TFTP Operation



Demonstration

`rcat` – a TFTP client

UDP Packet Trace

UDP Broadcasting

UDP supports broadcasting

Datagram sent using IP host ID of "all ones"

All recipients must listen on same port

Packet size limited by MTU of underlying network

Does not work through routers

Broadcasting – Core Code

```
int sock, yes = TRUE;
struct sockaddr_in bcast;
char data[100];

sock = socket(AF_INET, SOCK_DGRAM, 0);
setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &yes, sizeof yes);

bcast.sin_family = AF_INET;
bcast.sin_addr.s_addr = 0xffffffff;
bcast.sin_port = htons(MY_PORT);

sendto(sock, &data, sizeof data, 0, &bcast, sizeof bcast);
```

A Distributed Update Service

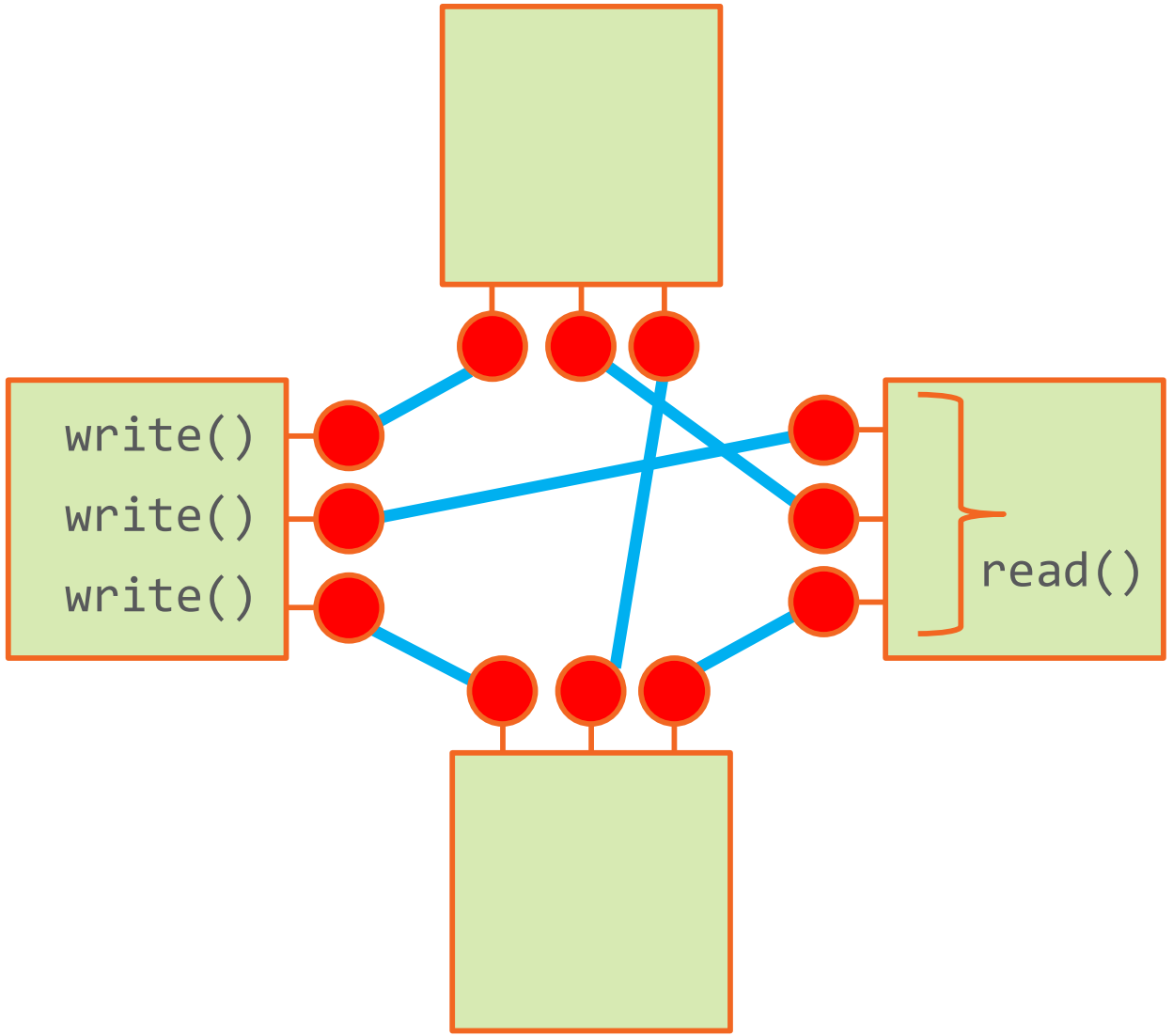
Multiple Participants

Each generates status information periodically

Each needs to receive status updates from all the others

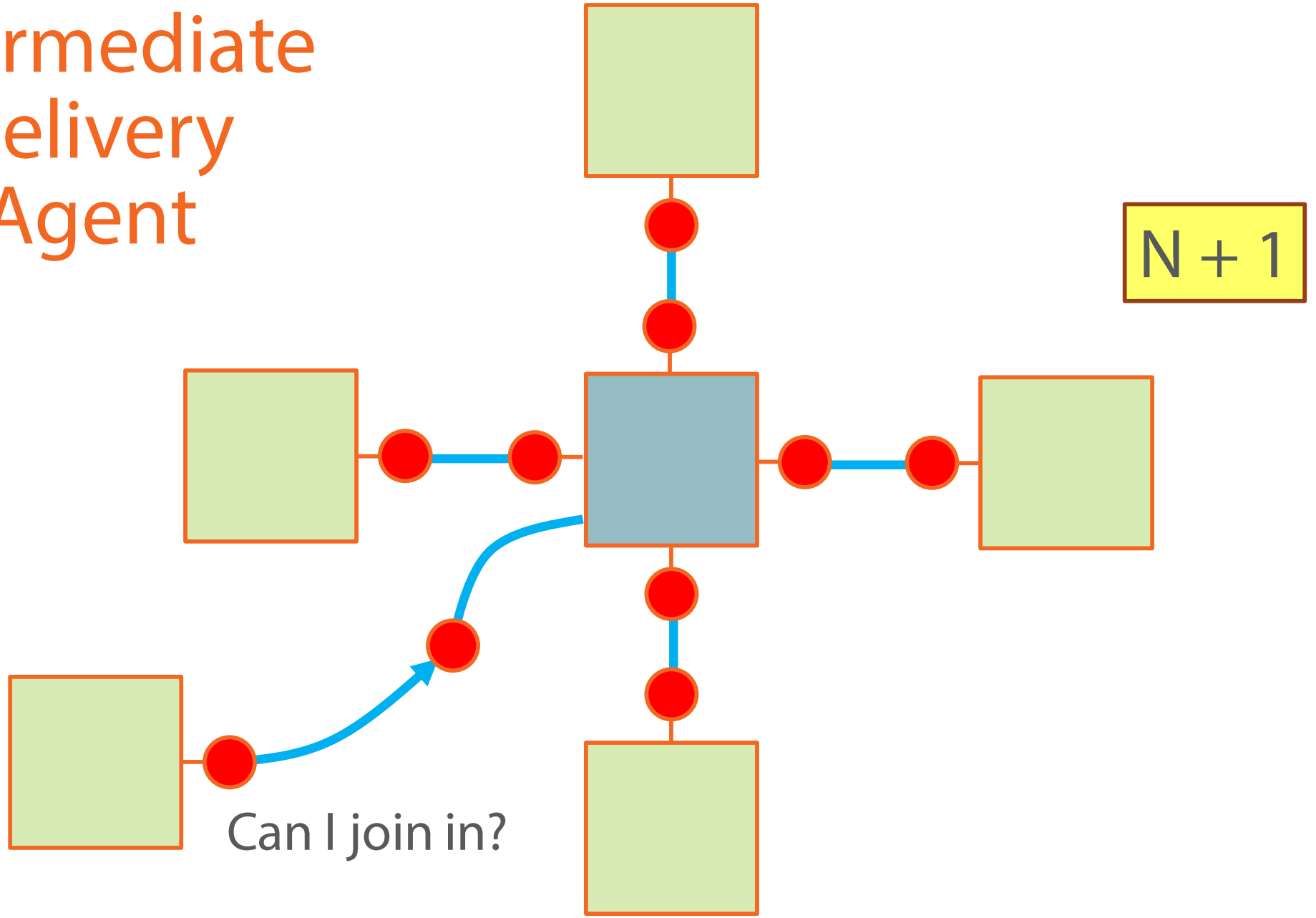
Each displays the overall status

Fully Connected Model

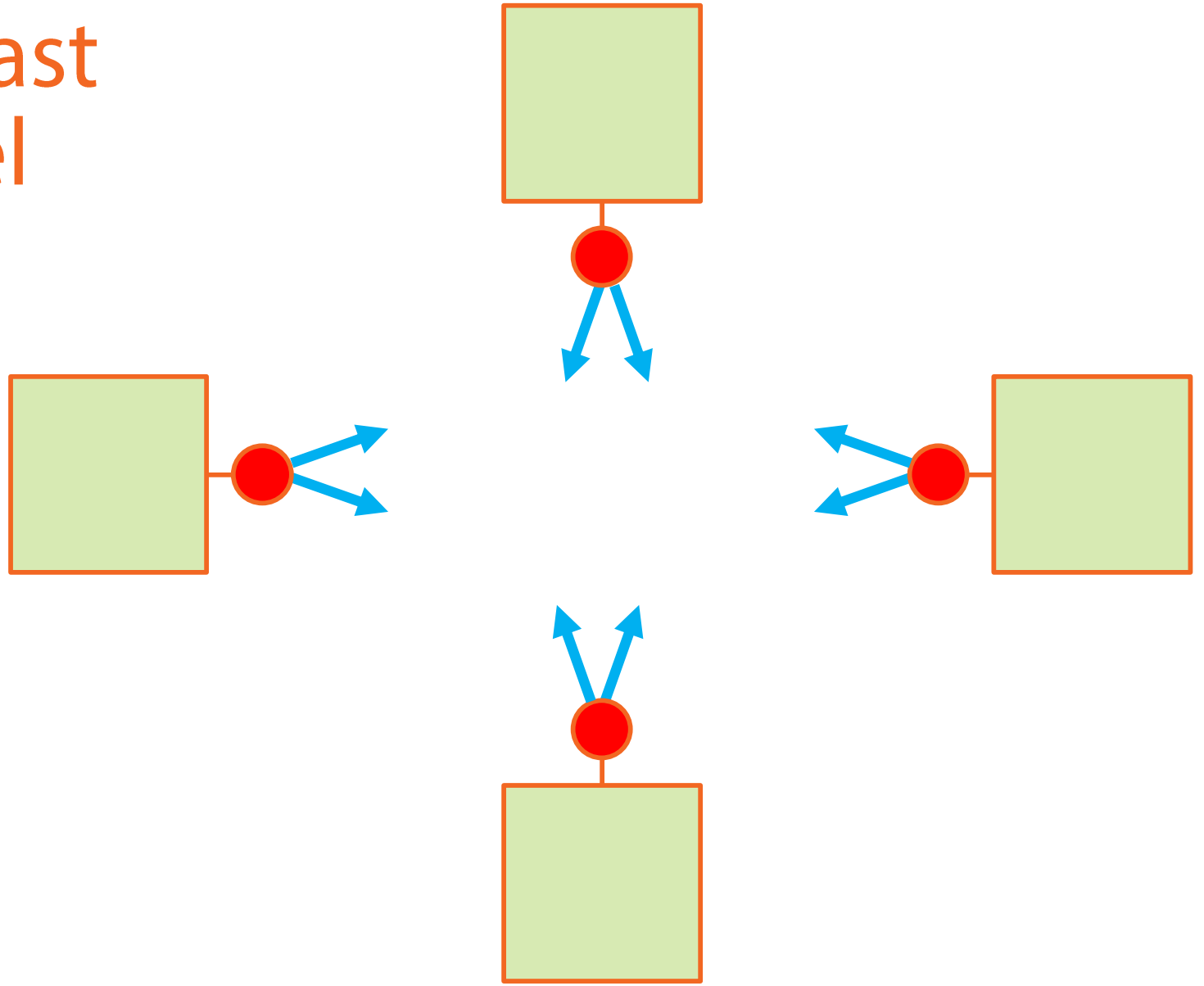


N * (N – 1)

Intermediate Delivery Agent

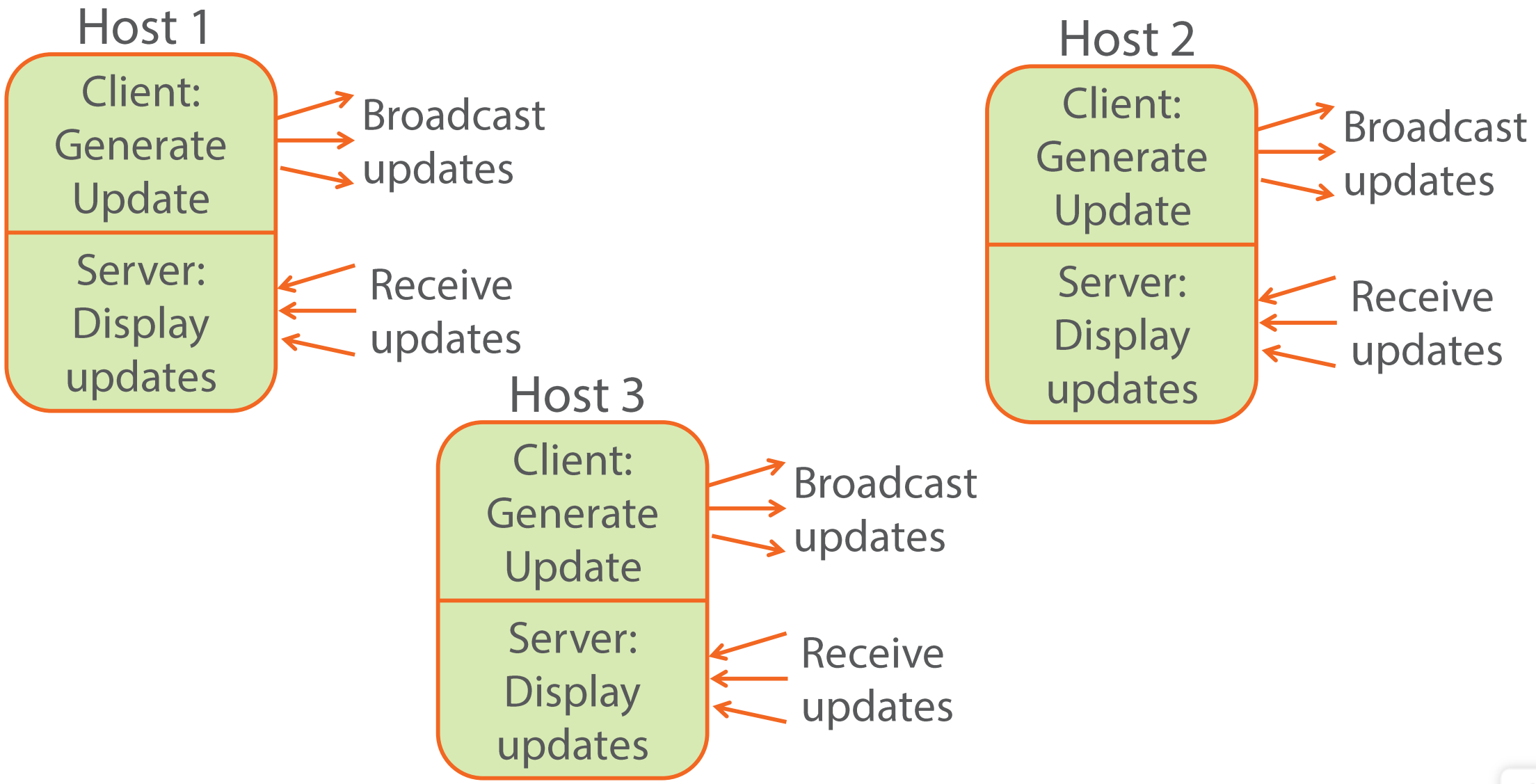


Broadcast Model

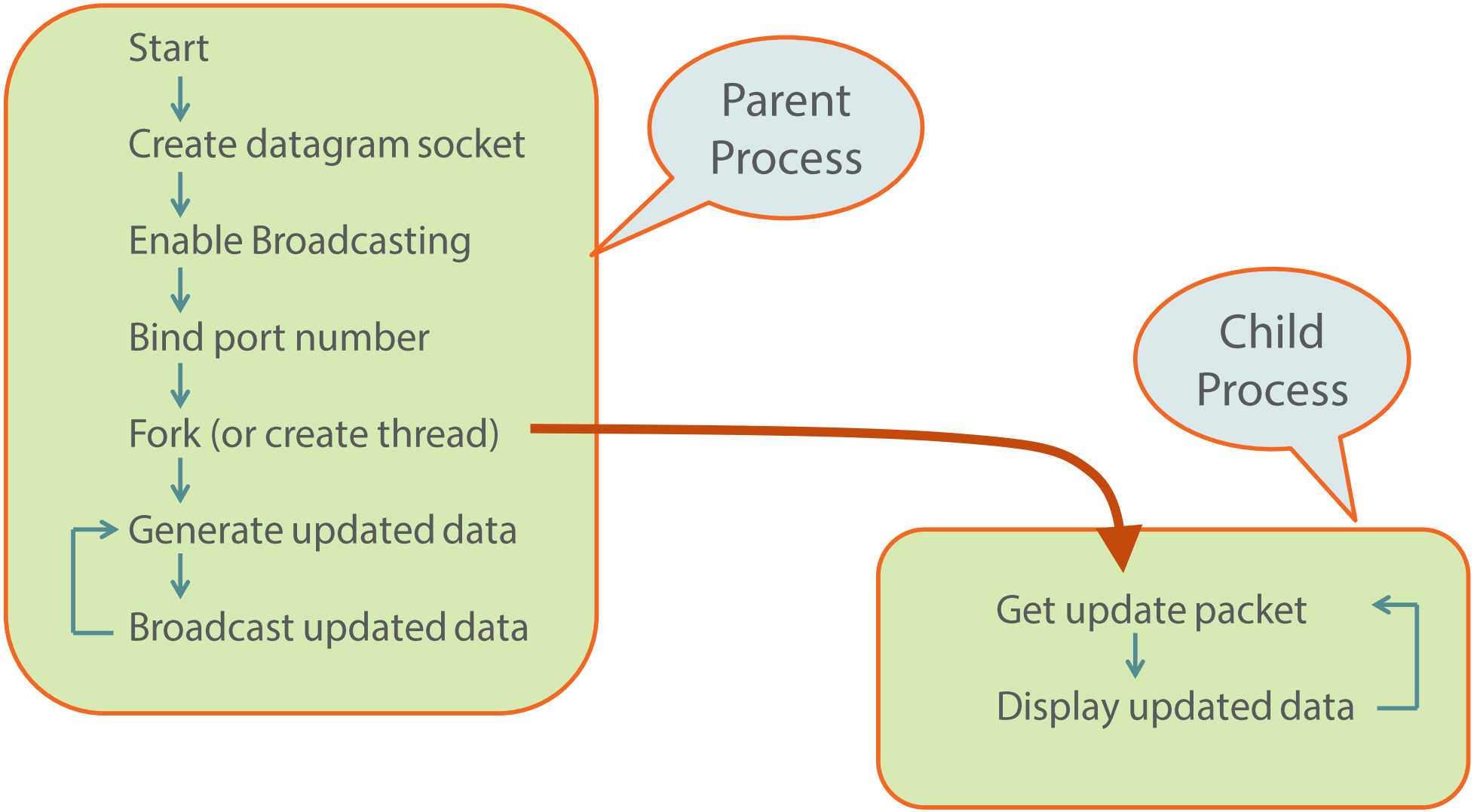


N

Peer-to-Peer Operation



Program Structure



Demonstration:

A Distributed Update Service

Moving Forward …

In this module:

- UDP sequence of operations
- UDP sockets API
- Text and binary protocols
- The rcat client
- UDP broadcasting
- Distributed update service

Coming up in the next module:

- Concurrent servers and clients