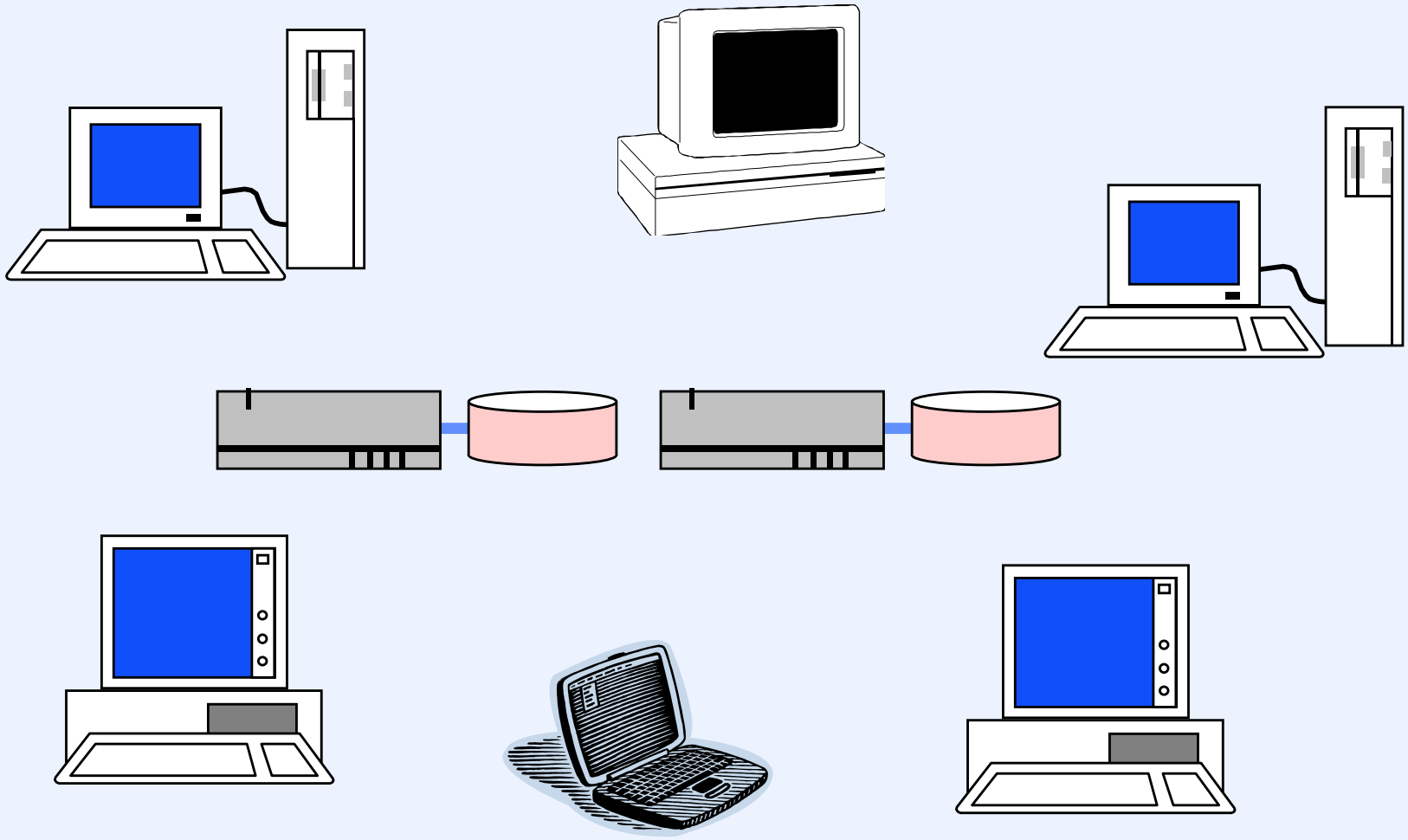
Introduction to Networking

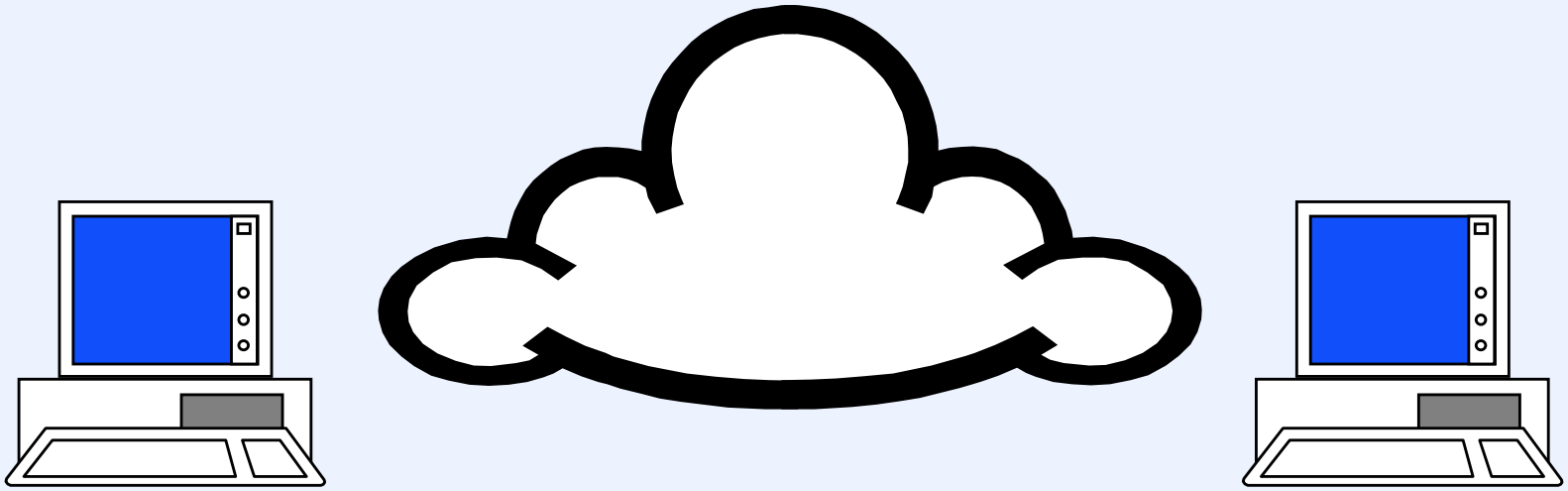
Distributed File Systems



What We Cover

- Communication protocols
- Remote procedure call protocols
- **Distributed file systems**

Communication



Some Issues

- Quantity: how many are communicating?
  - unicast
  - broadcast
  - multicast
- Quality: how good/reliable is the communication?
  - best effort
  - fully reliable
  - guaranteed bandwidth and delay

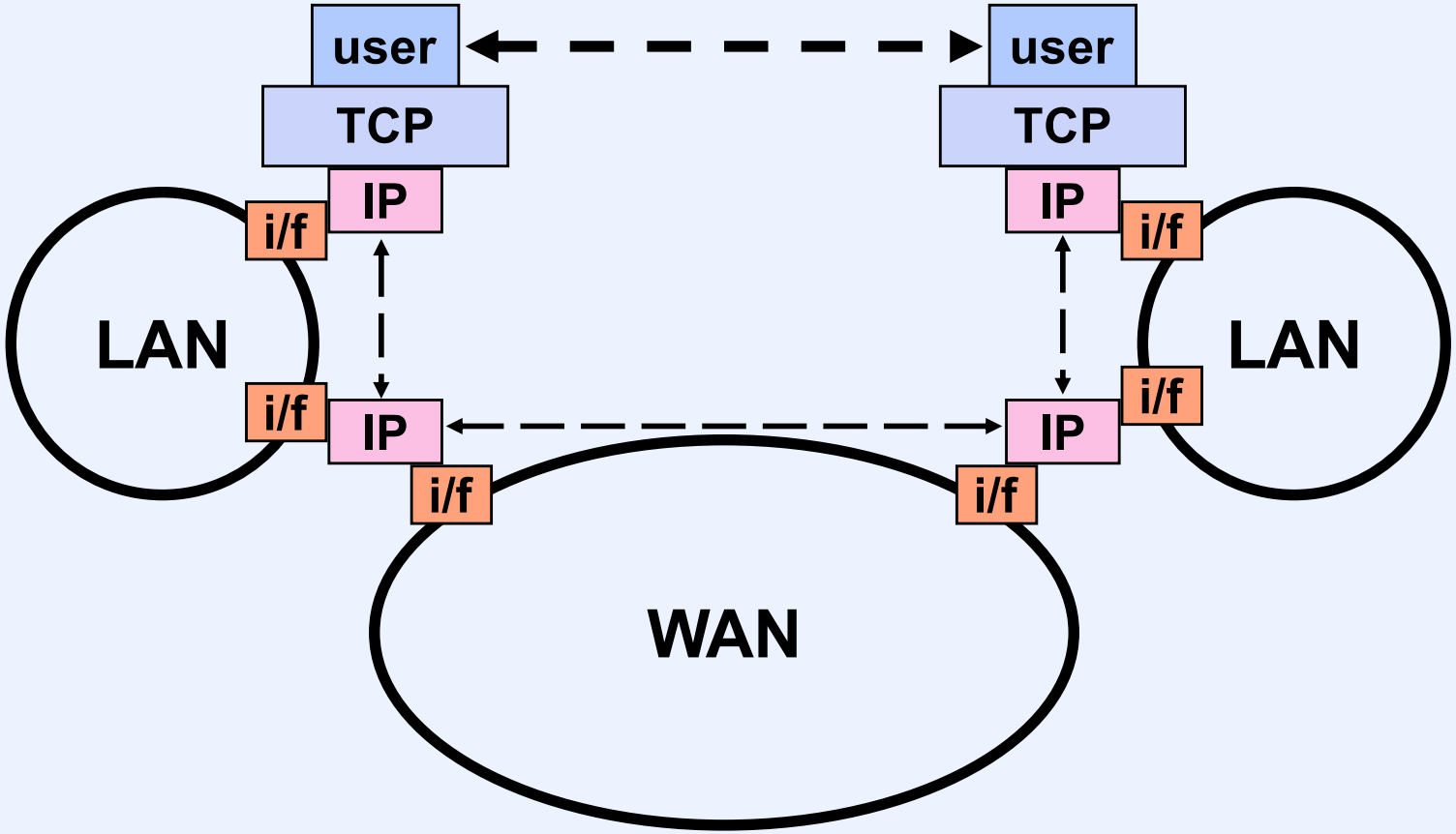
Some More Issues

- Naming and addressing
  - www.cs.brown.edu vs. 128.148.32.110
- Routing
- Congestion

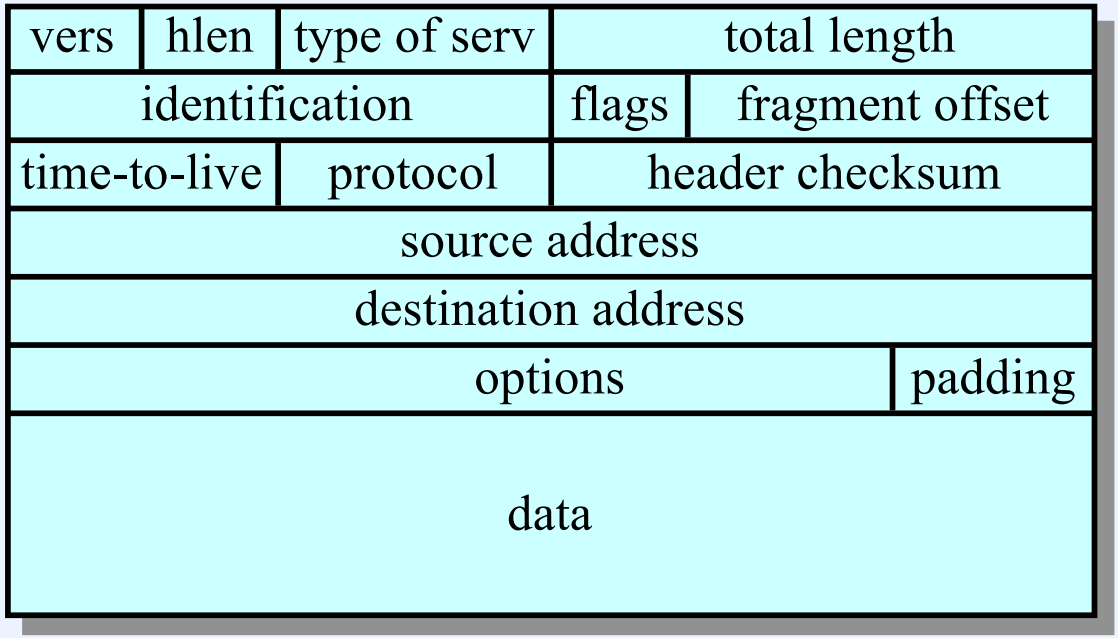
Circuits vs. Packets

- Circuit switching
  - reserve a circuit between communicating entities
- Packet switching
  - break data into packets
  - transmit each packet separately — each packet could travel a different route

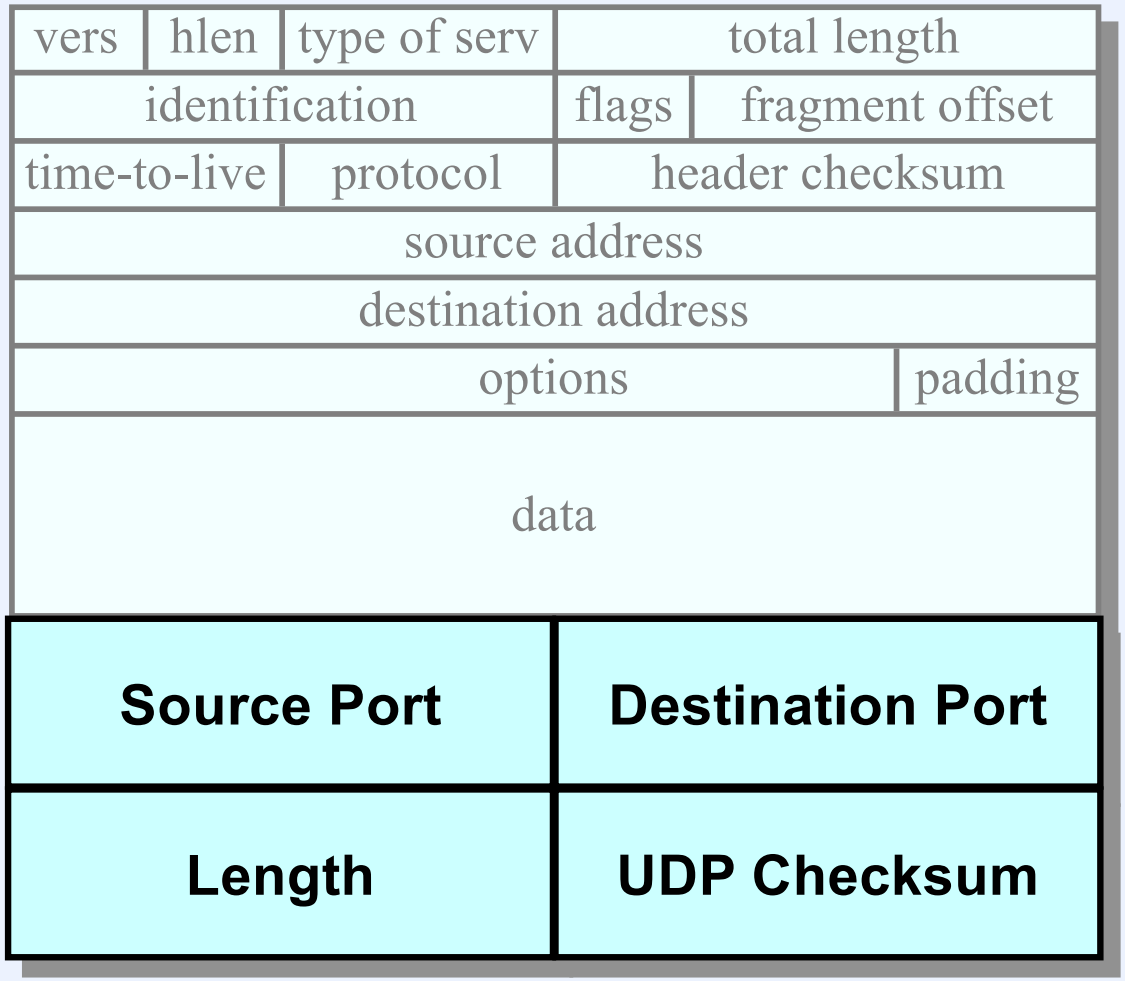
Internetworking with TCP/IP



IP Header



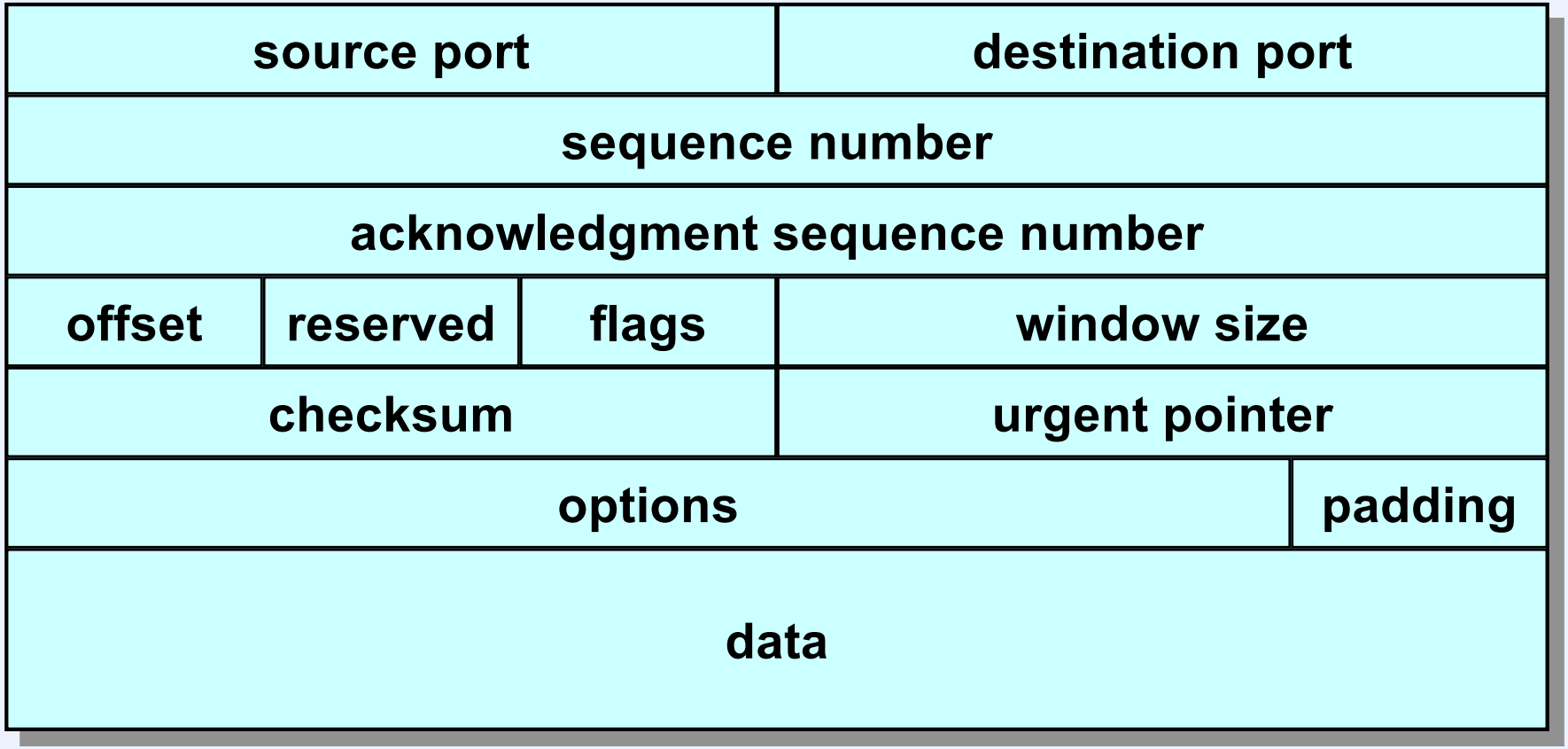
User Datagram Protocol



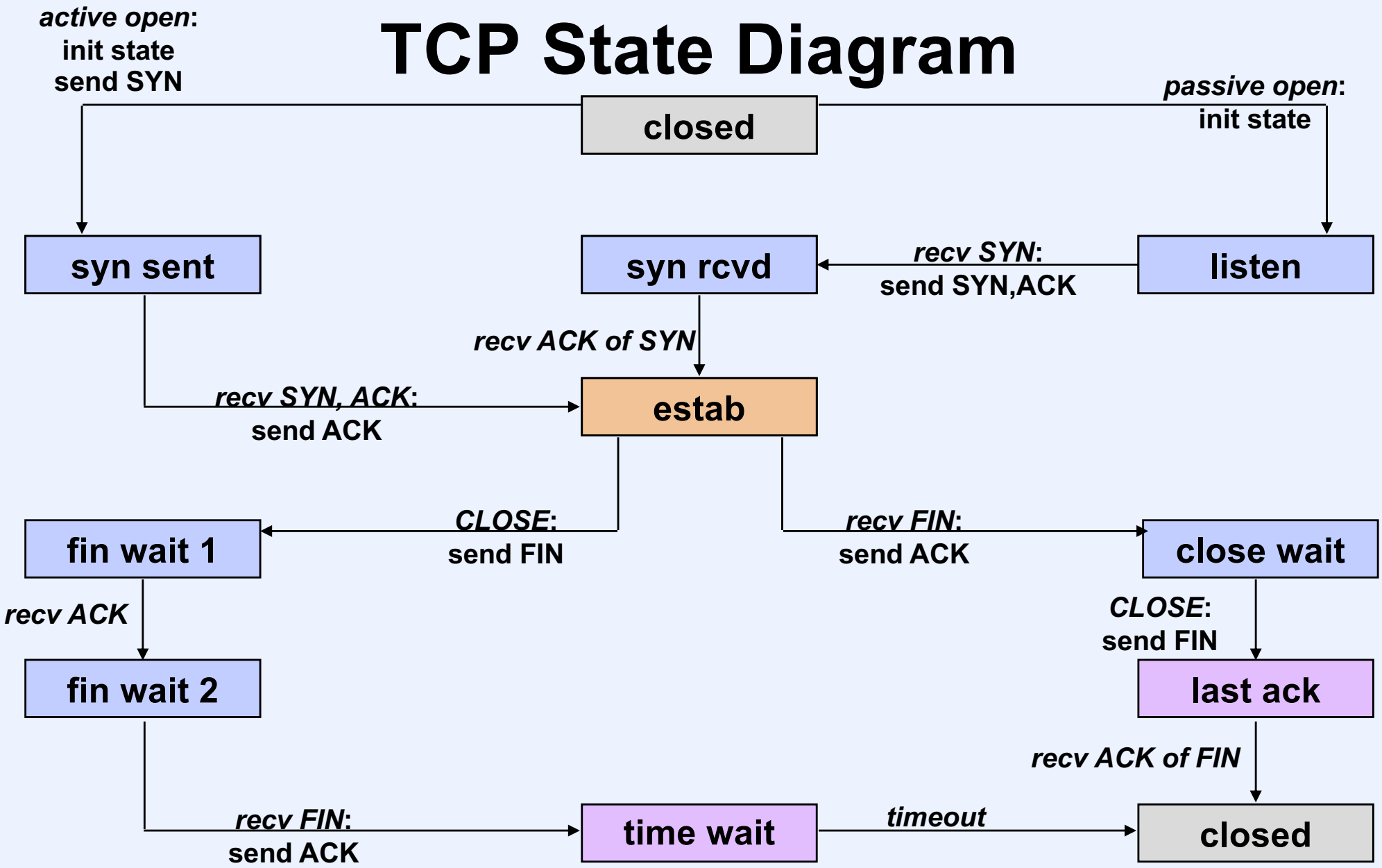
Transmission Control Protocol (TCP)

- A multiplexing service built on top of IP
- A full-duplex reliable stream protocol
- Provides reliable data transfer
- Packet boundaries are not seen by the application
  - it sees a sequence of bytes, not a sequence of packets
- A connection must be established for communication to take place
  - connections are flow-controlled
  - connections are congestion-controlled

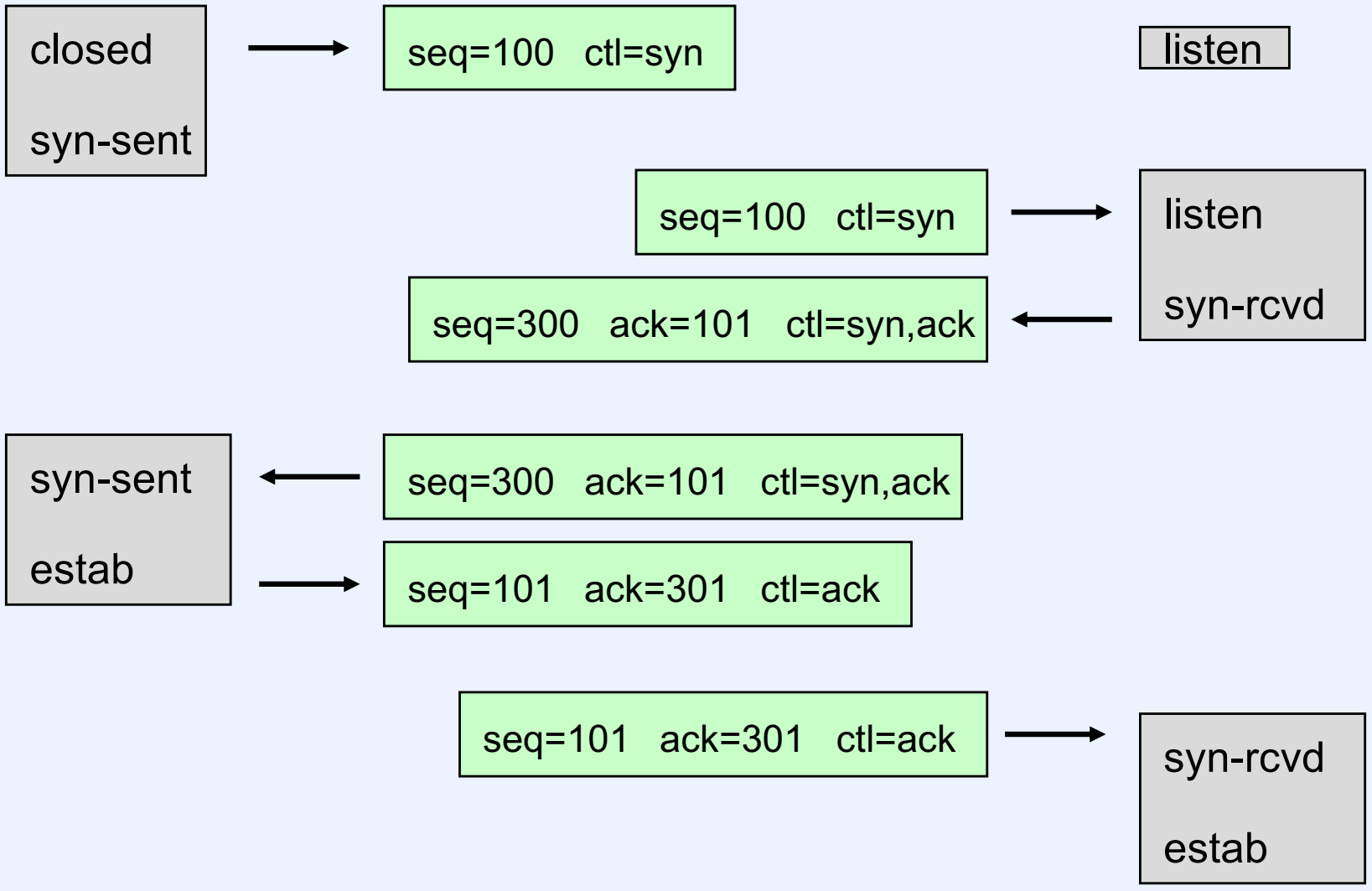
TCP Header Components



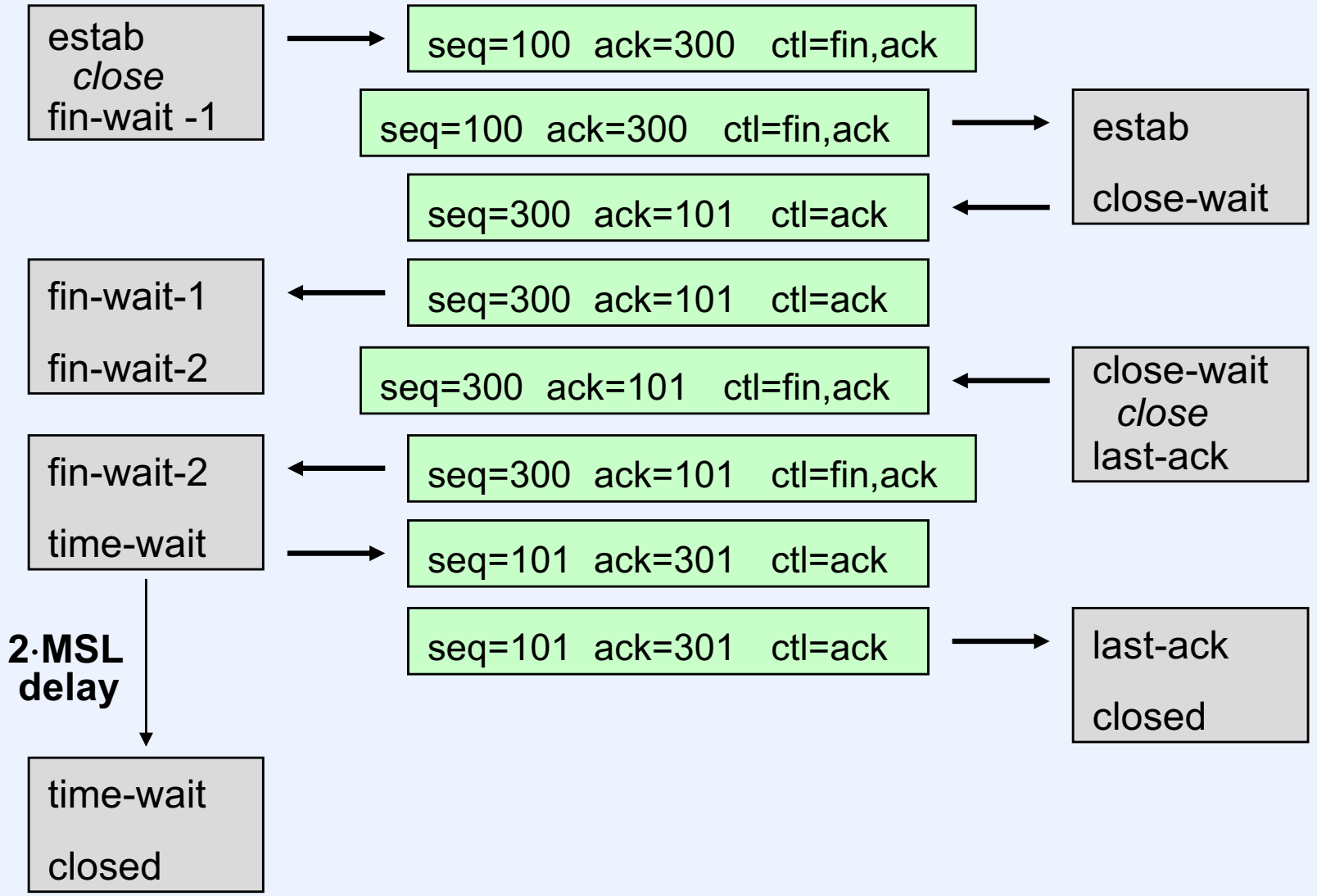
TCP State Diagram



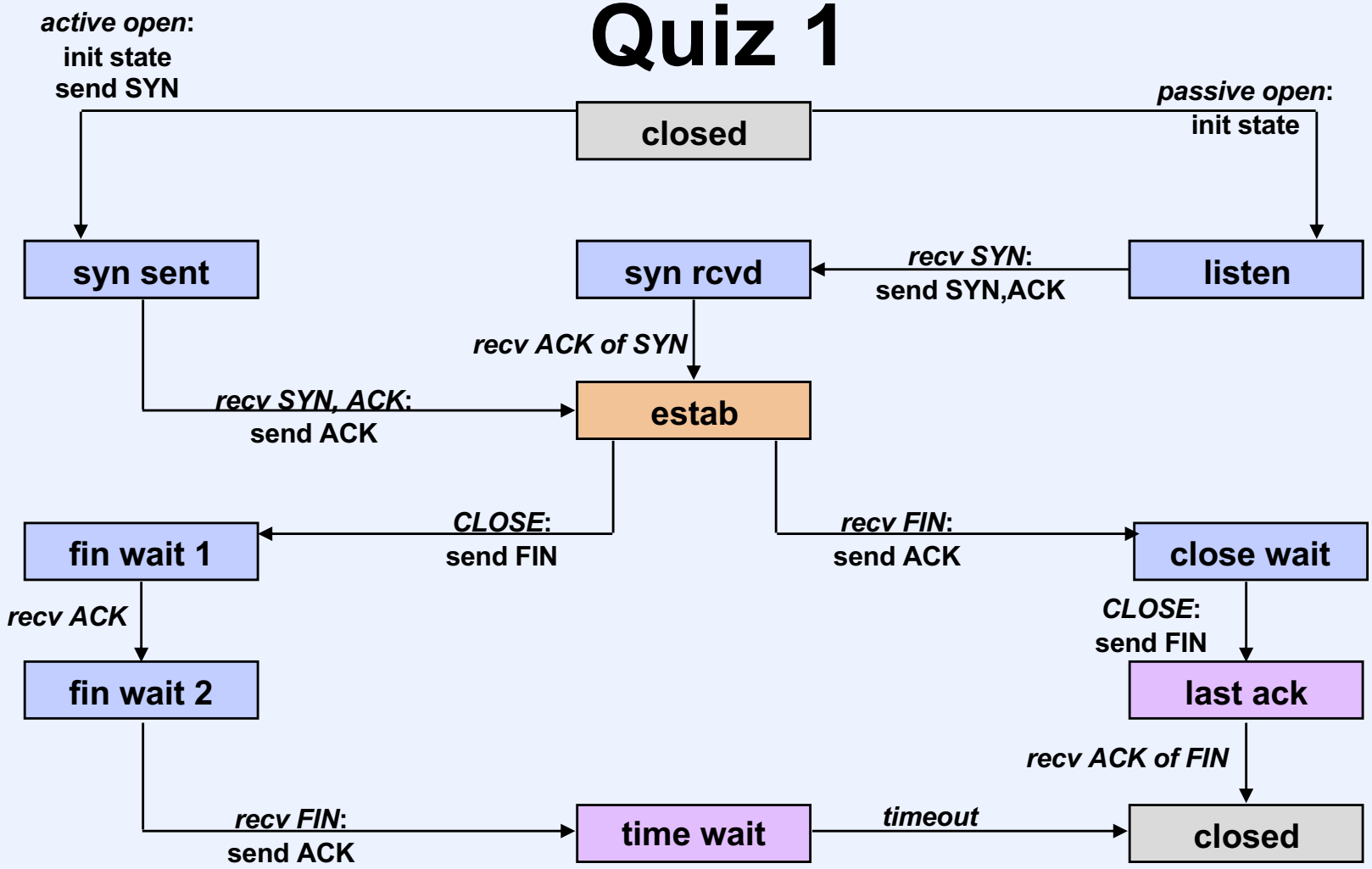
Connection Initiation



Shutdown



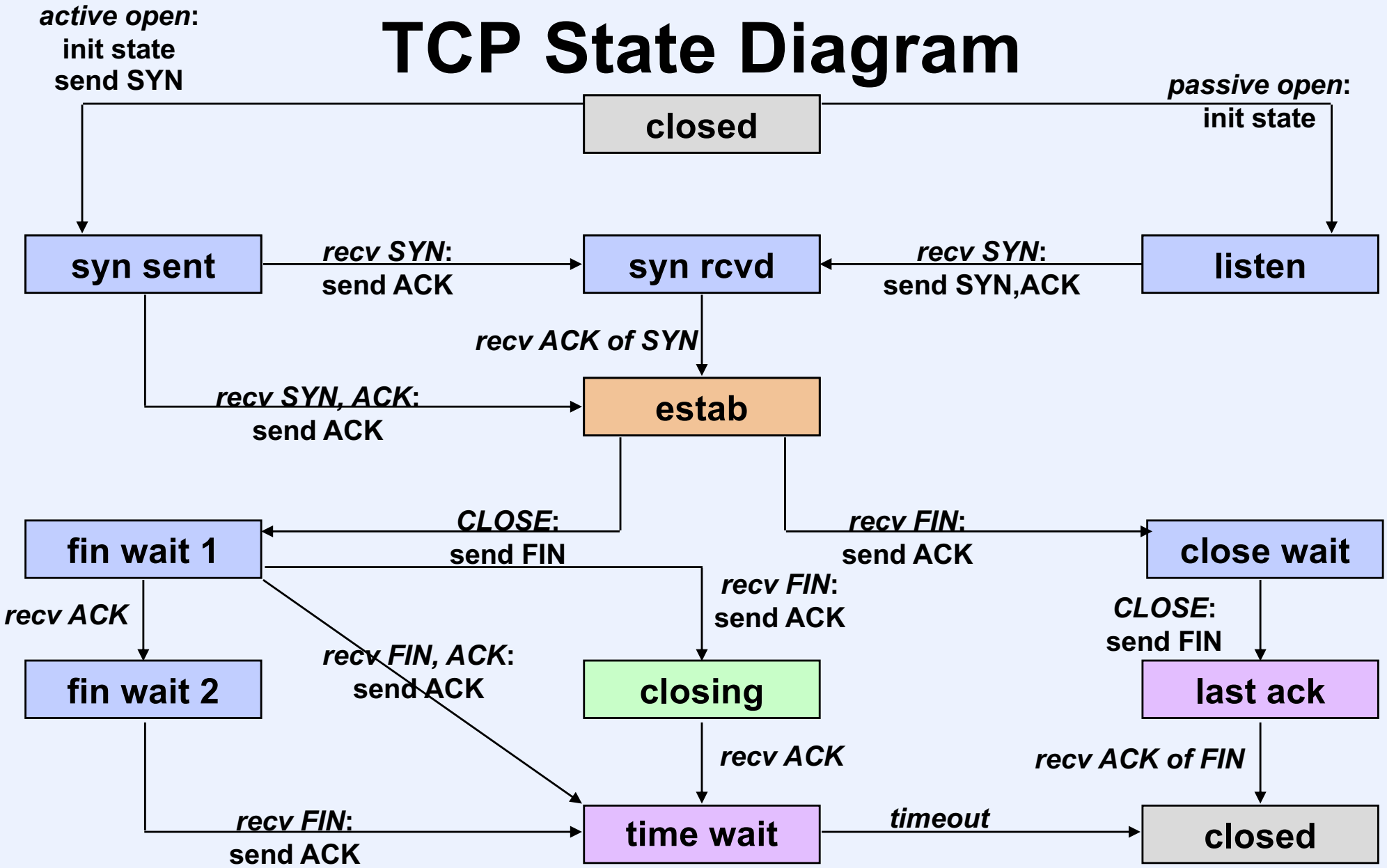
Quiz 1



Suppose both sides send the other a FIN while in the estab state; both then go to fin-wait-1. What happens next?

a) Both go to fin-wait-2, then time-wait
b) Both go to a new state that's not in the diagram
c) Such an event can't happen

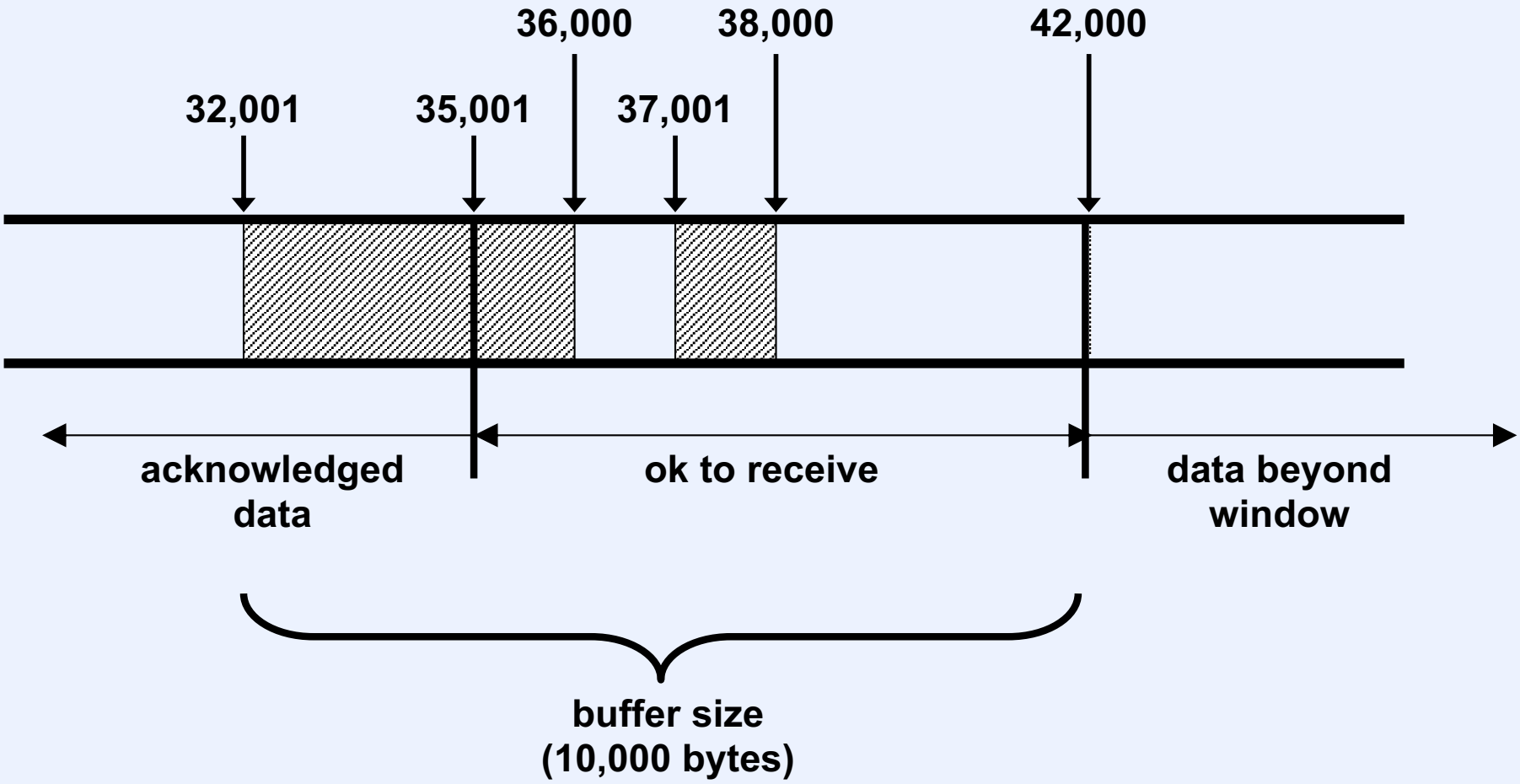
TCP State Diagram



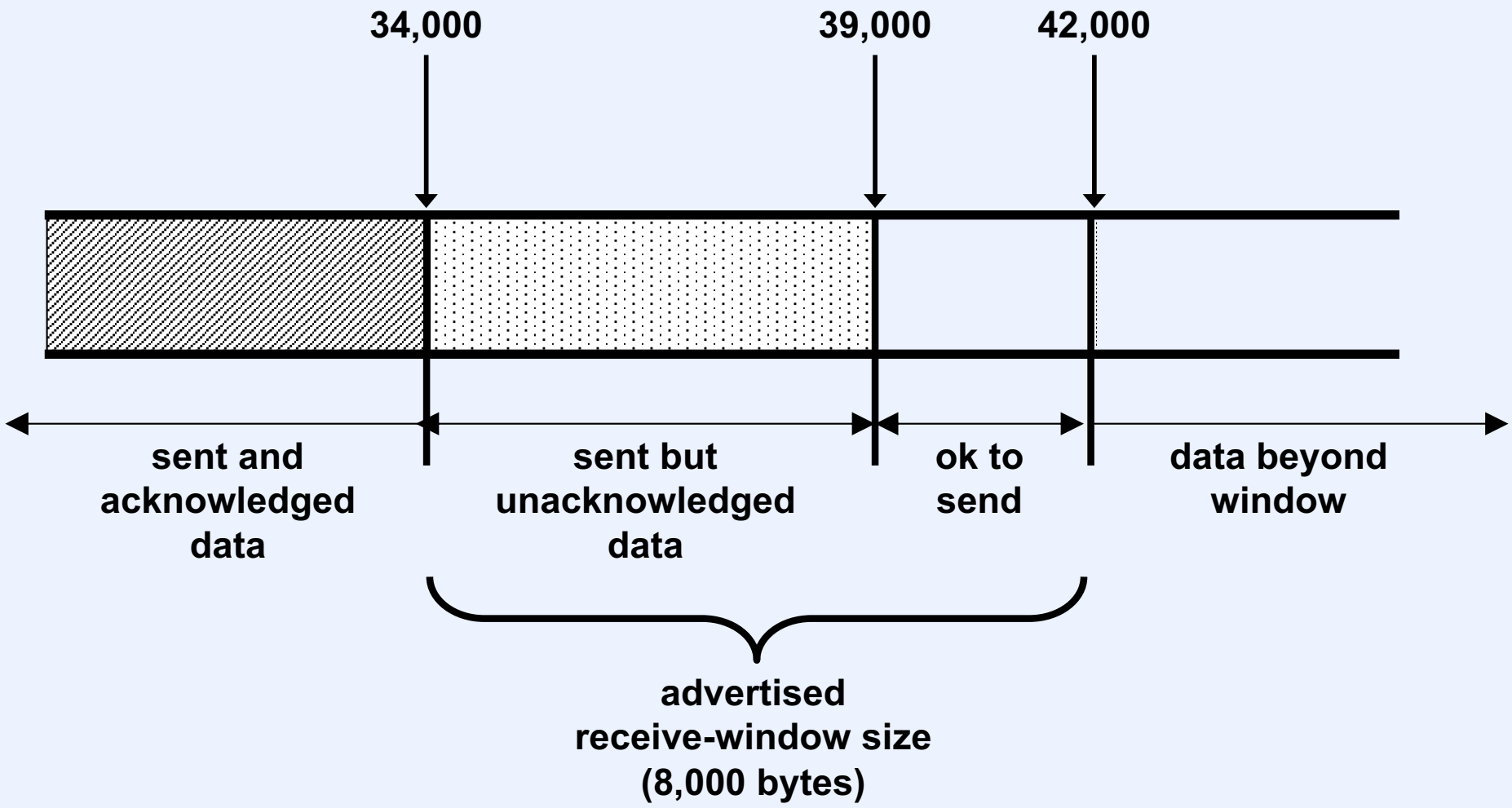
Sliding-Window Protocol

- Used for:
  - reliable delivery
  - flow control
- Windows
  - send
  - receive

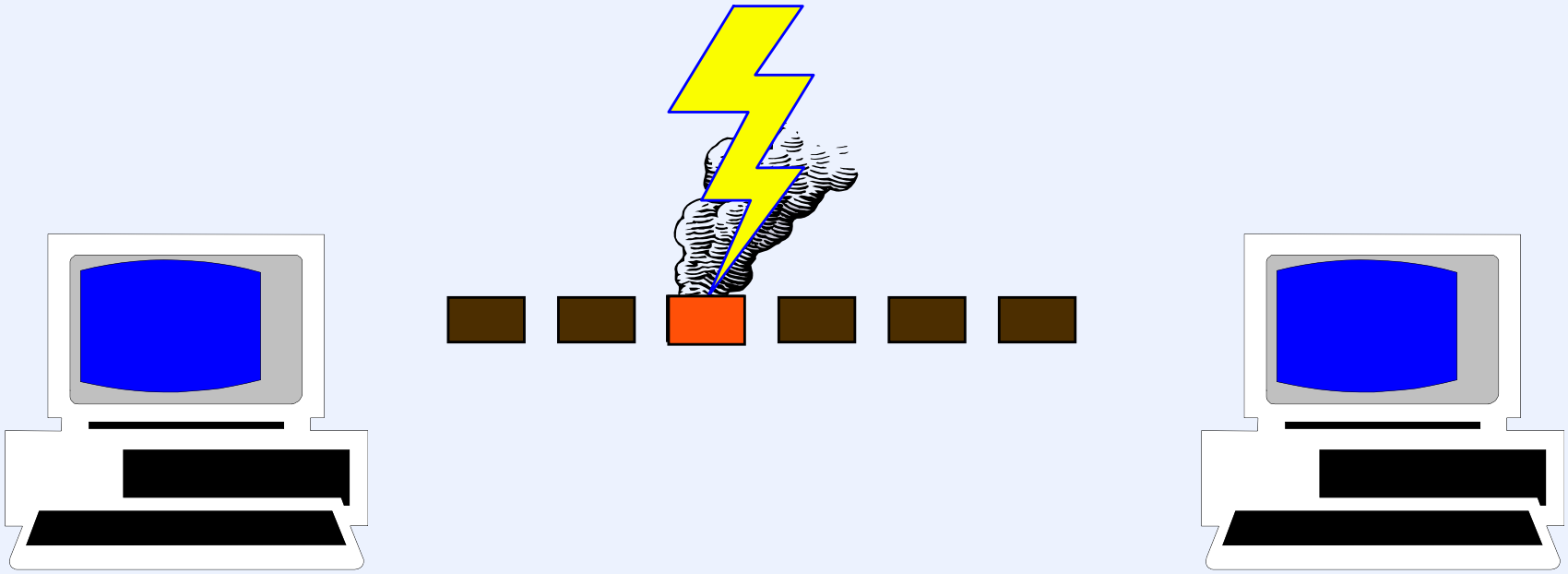
TCP Receive Window



TCP Send Window



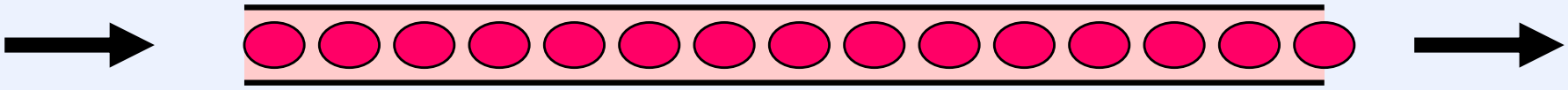
Coping With Lost Data



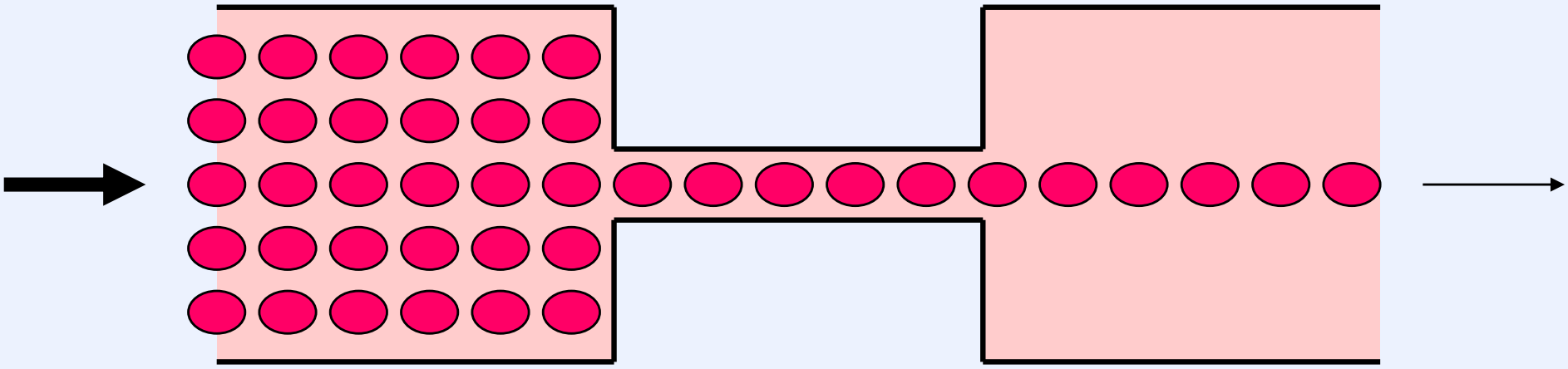
When to Retransmit?

- After waiting significantly longer than the average ("smoothed") roundtrip time
- How much longer?
  - significantly longer than the average deviation
- What if one retransmission doesn't do it?
  - transmit again …
- When?
  - use exponential backoff
- When does one give up?
  - eventually …

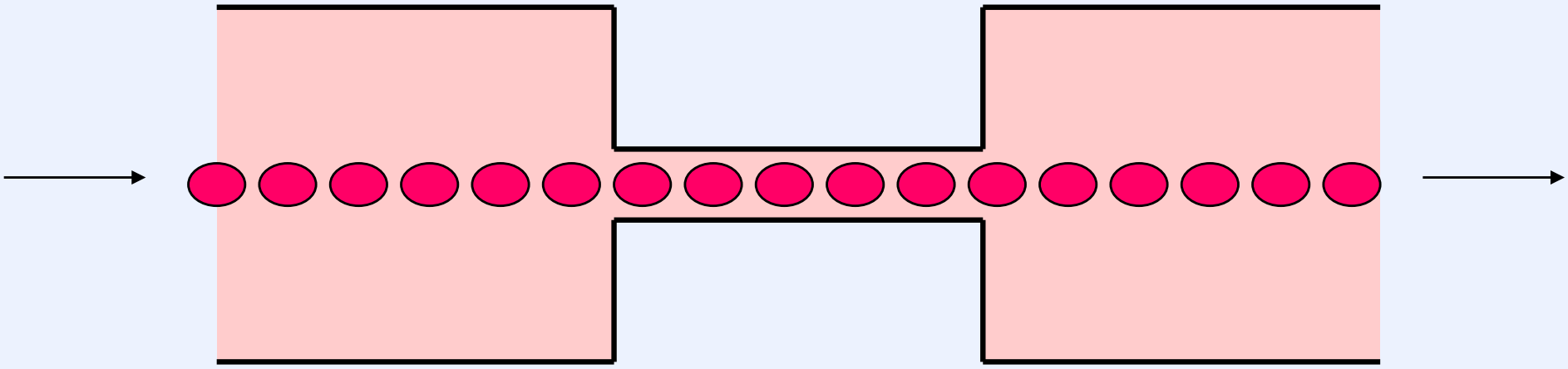
Ack Clocking



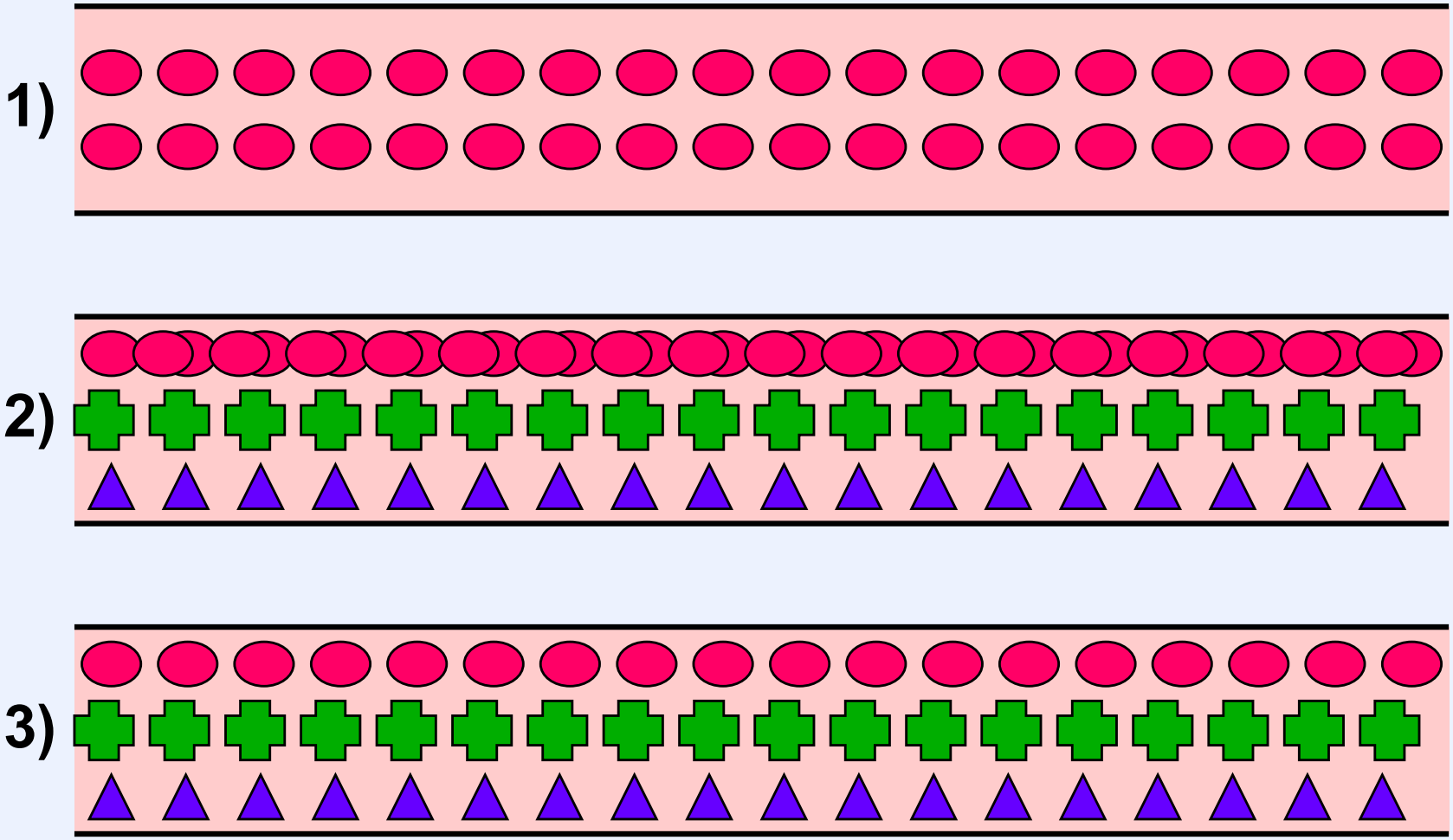
Fast Start



Slow Start

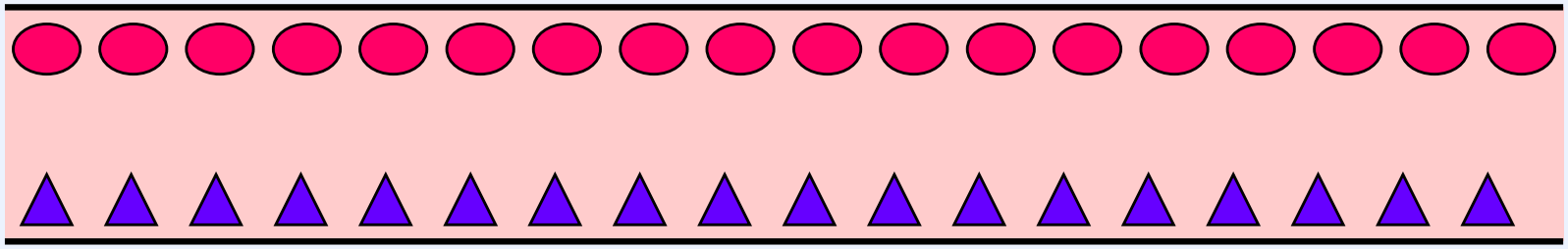


Congestion Control (1)

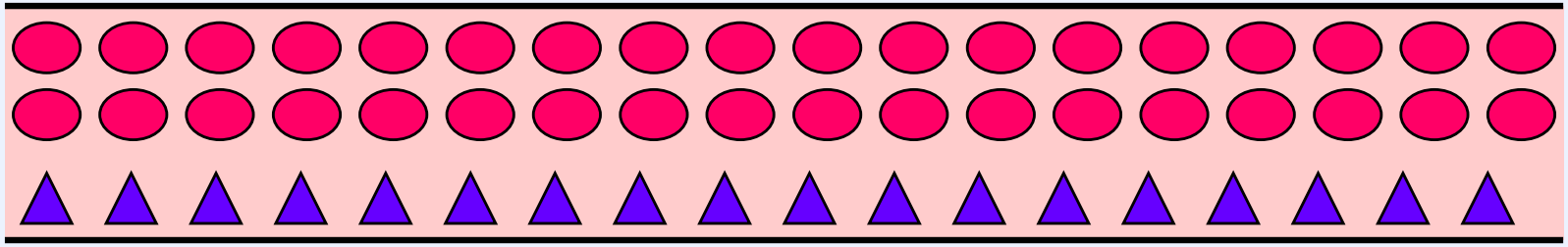


Congestion Control (2)

4)



5)



Acks

- When a segment is received, the highest permissible Ack is sent back
  - if data up through i has been received, the ack sequence number is i+1
  - if data up through i has been received, as well as i+100 through i+200, the ack sequence number is i+1
    - a higher value would imply that data in the range [i+1, i+99] has been received
- Every segment sent contains the most up-to-date Ack

Quiz 2

A TCP sender has sent four hundred-byte segments starting with sequence numbers 1000, 1100, 1200, and 1300, respectively. It receives from the other side three consecutive ACKs, all mentioning sequence number 1100. It may conclude that

a) The first segment was received, but nothing more

b) The first, third, and fourth segments were received, but not the second

c) The first segment was received, but not the second; nothing is known about the others

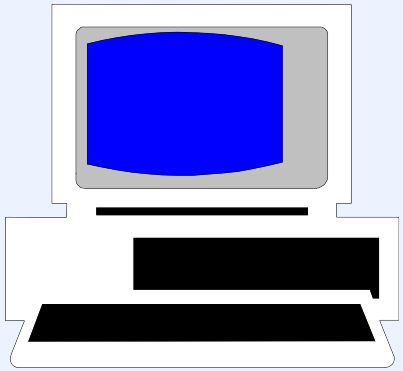
d) There's a bug in the receiver

Fast Retransmit and Recovery

- Waiting an entire RTO before retransmitting causes the "pipeline" to become empty
  - must slow-start to get going again
- If one receives three acks that all repeat the same sequence number:
  - some data is getting through
  - one segment is lost
  - immediately retransmit the lost segment
  - halve the congestion window (i.e., perform congestion control)
  - don't slow-start (there is still data in the pipeline)

Remote Procedure Call Protocols

Local Procedure Calls

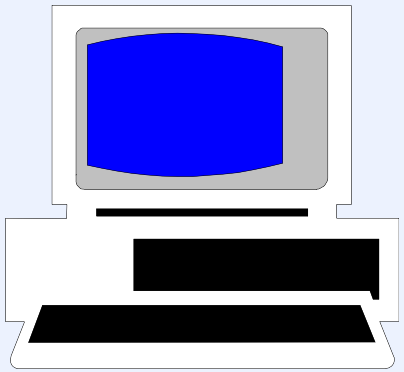


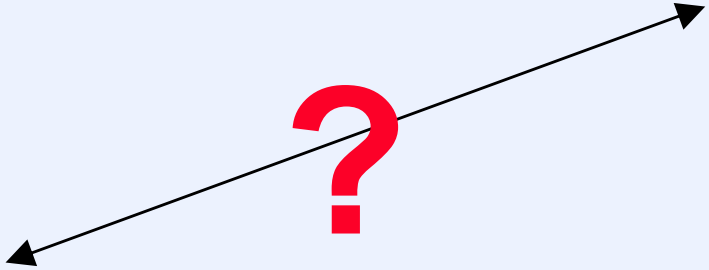
```
// Client code
      . . .
result = procedure(arg1, arg2);
      . . .
```

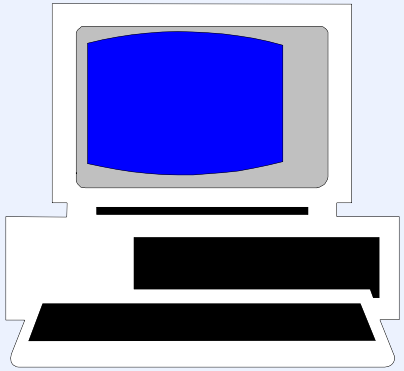
```
// Server code
result_t procedure(a1_t arg1, a2_t arg2) {
      . . .
      return(result);
}
```

Remote Procedure Calls (1)

```
// Client code
     . . .
result = procedure(arg1, arg2);
     . . .
```





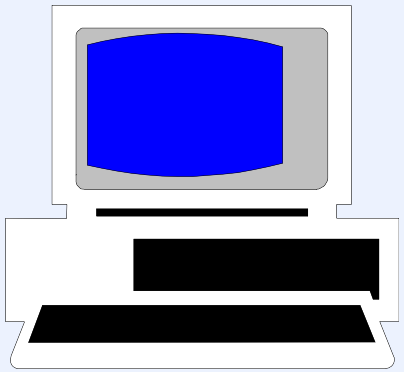


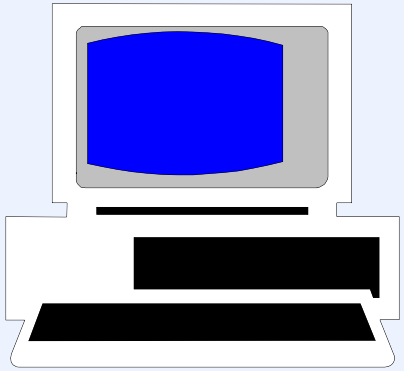
```
// Server code
result_t procedure(a1_t arg1, a2_t arg2) {
     . . .
     return(result);
}
```

Remote Procedure Calls (2)

```
// Client code
    . . .
result = procedure(arg1, arg2);
    . . .
```

Client-Side Stub

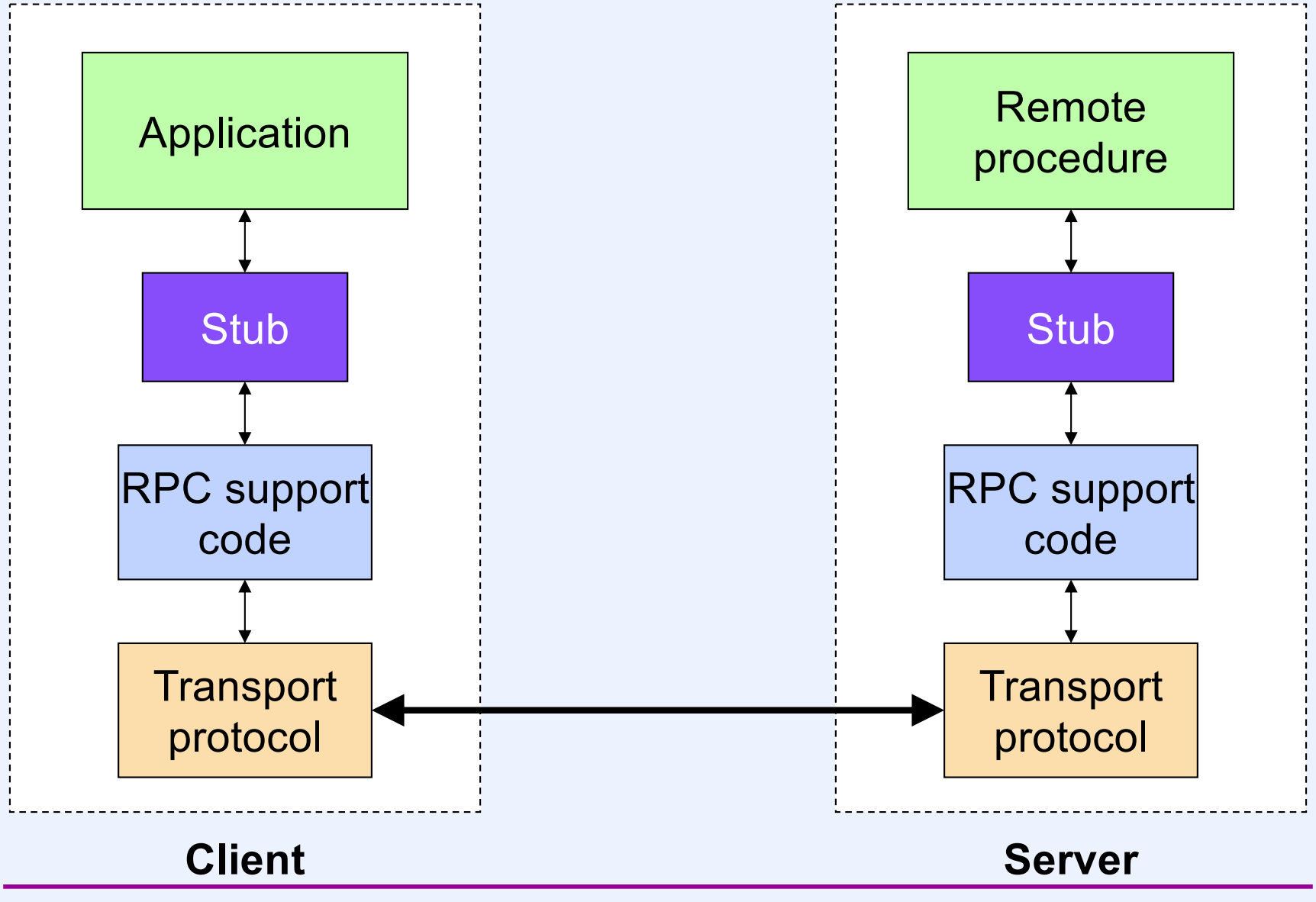




Server-Side Stub

```
// Server code
result_t procedure(a1_t arg1, a2_t arg2) {
    . . .
    return(result);
}
```

Block Diagram



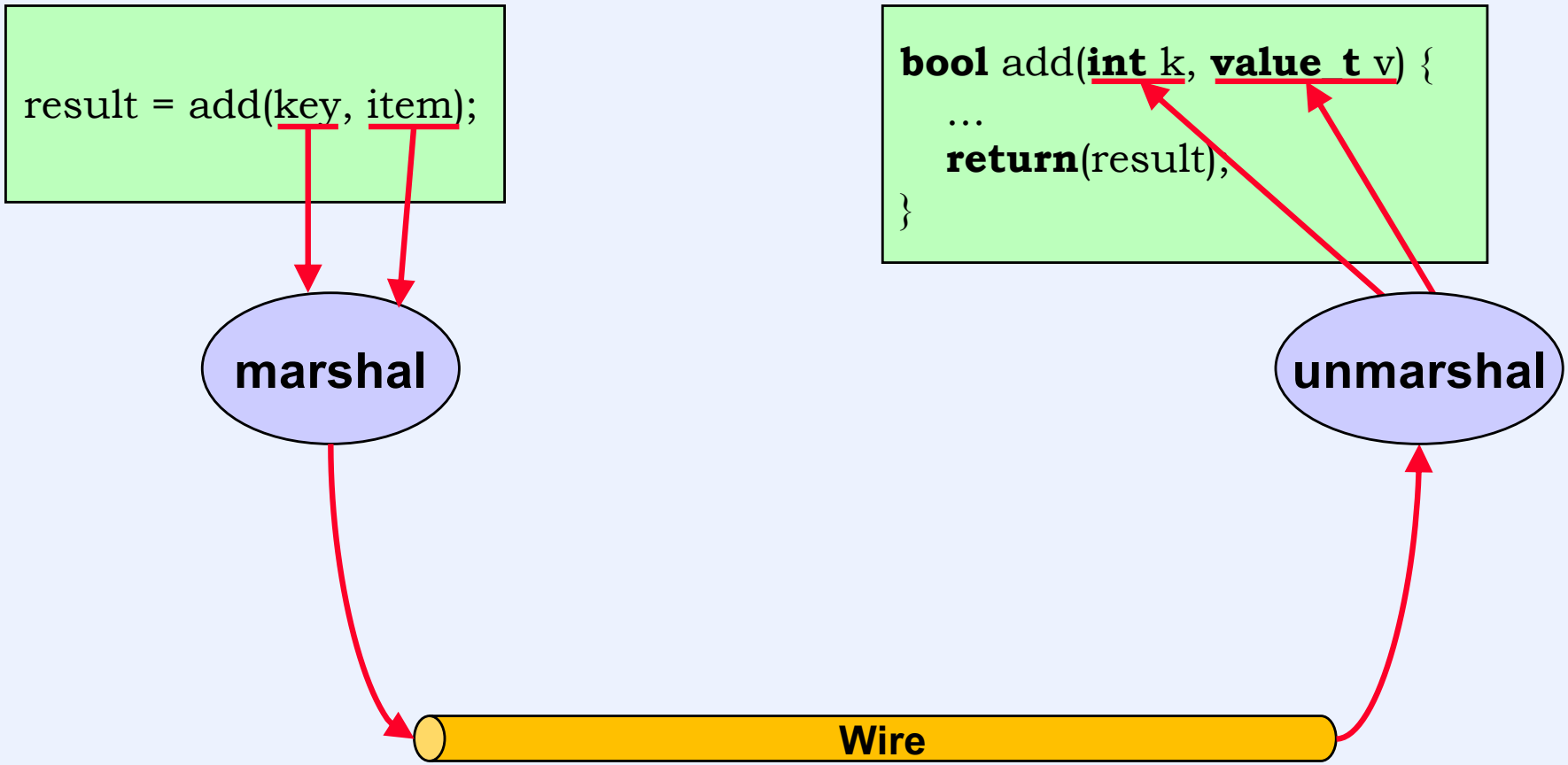
ONC RPC

- Used with NFS
- eXternal Data Representation (XDR)
  - specification for how data is transmitted
  - language for specifying interfaces

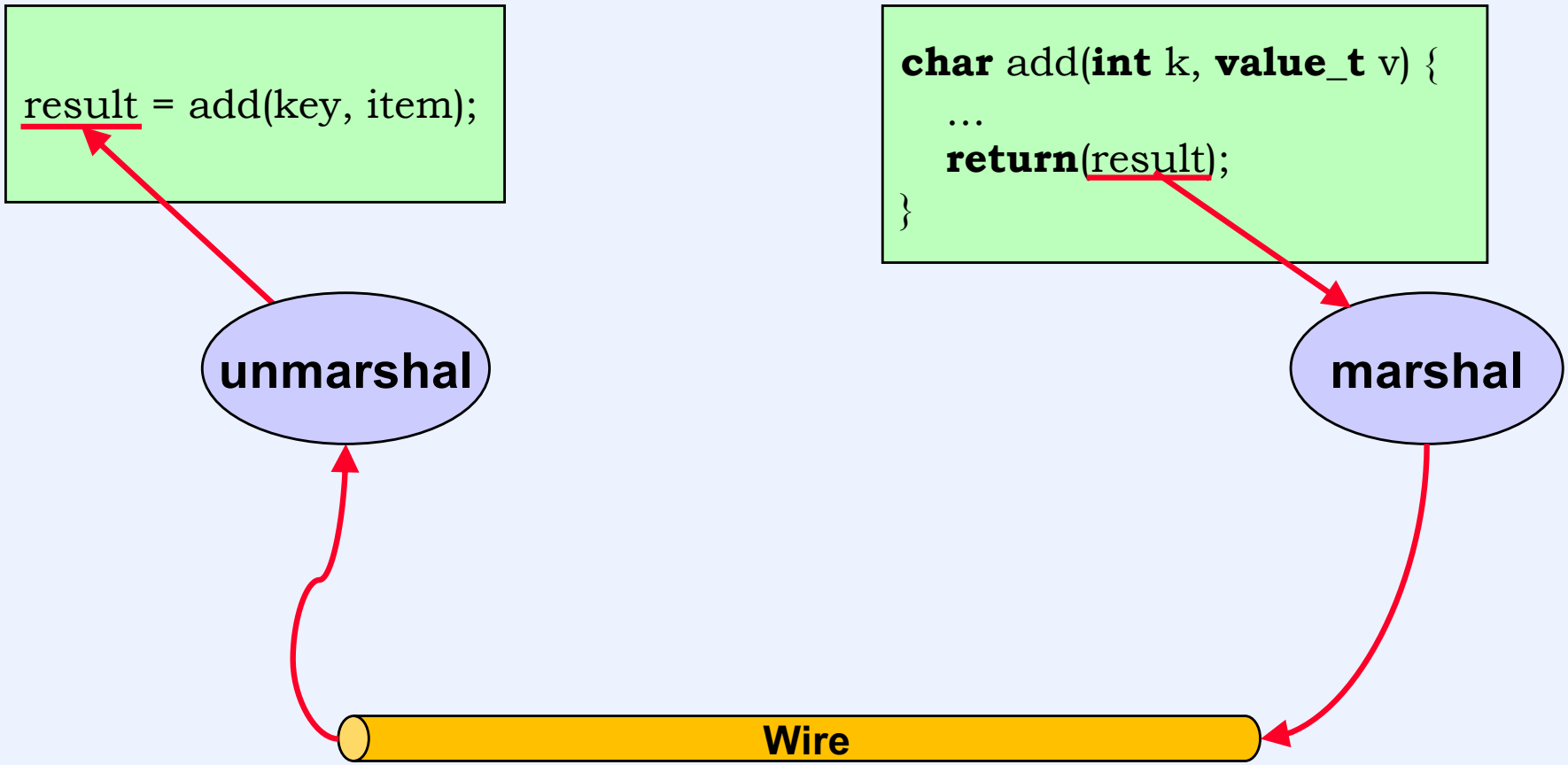
Example

```
typedef struct {
   int   comp1;
   float comp2[6];
   char  *annotation;
} value_t;

typedef struct {
   value_t  item;
   list_t   *next;
} list_t;

bool add(int key, value_t item);
bool remove(int key, value_t item);
list_t query(int key);
```

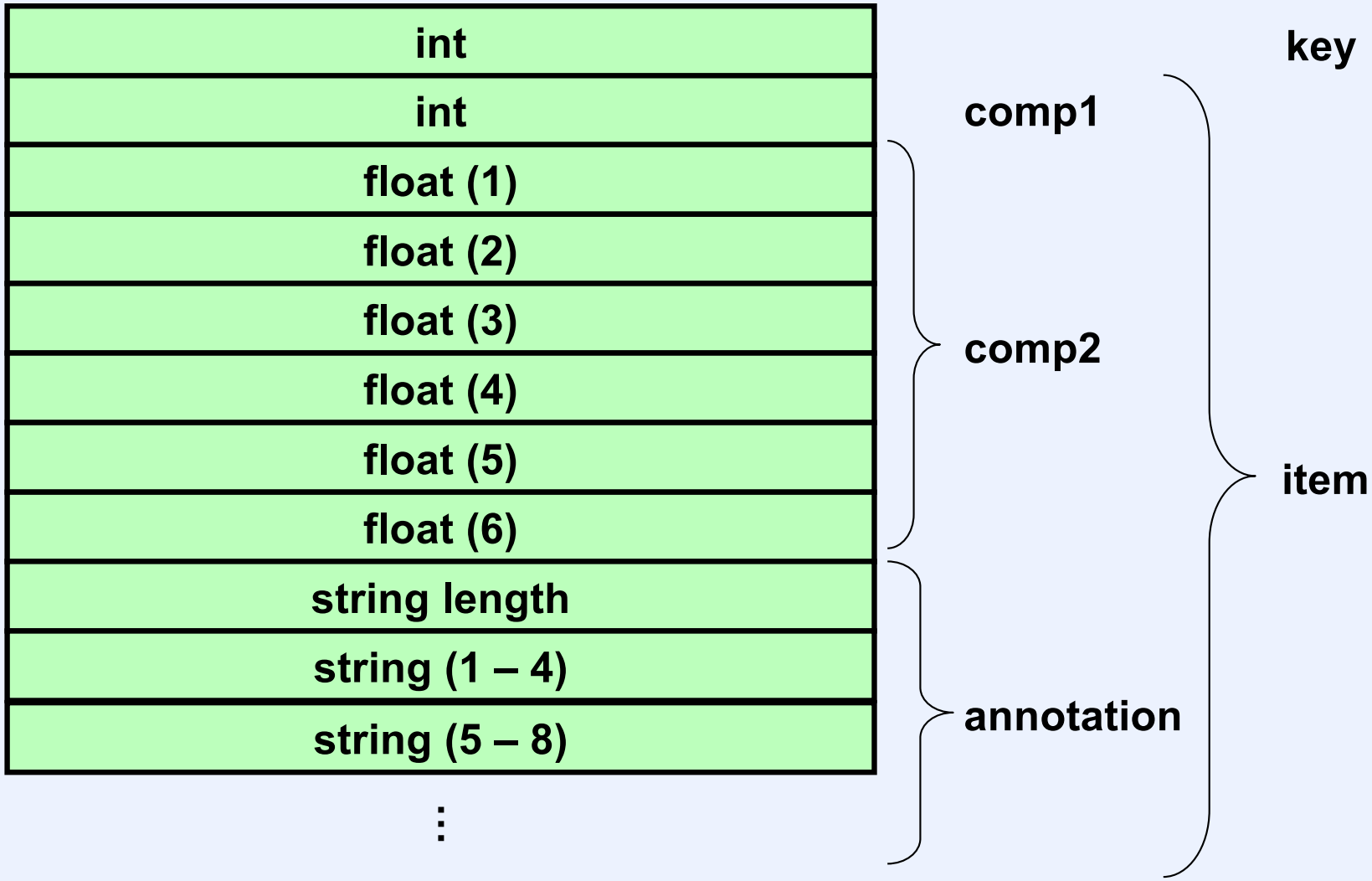
Placing a Call



Returning From the Call



Marshalled Arguments



Marshalled Linked List