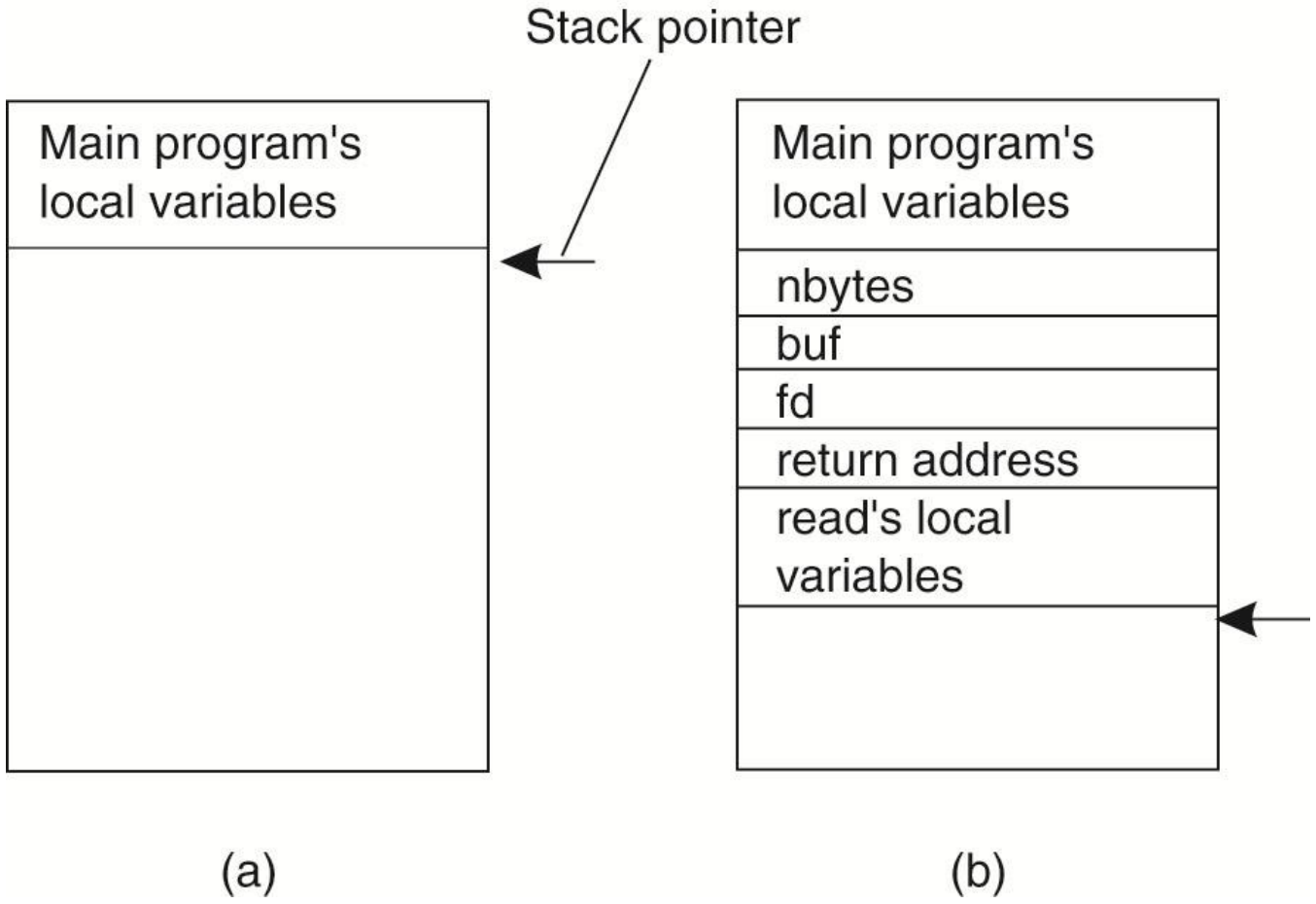
Communication protocols: Remote Procedure Call (RPC)

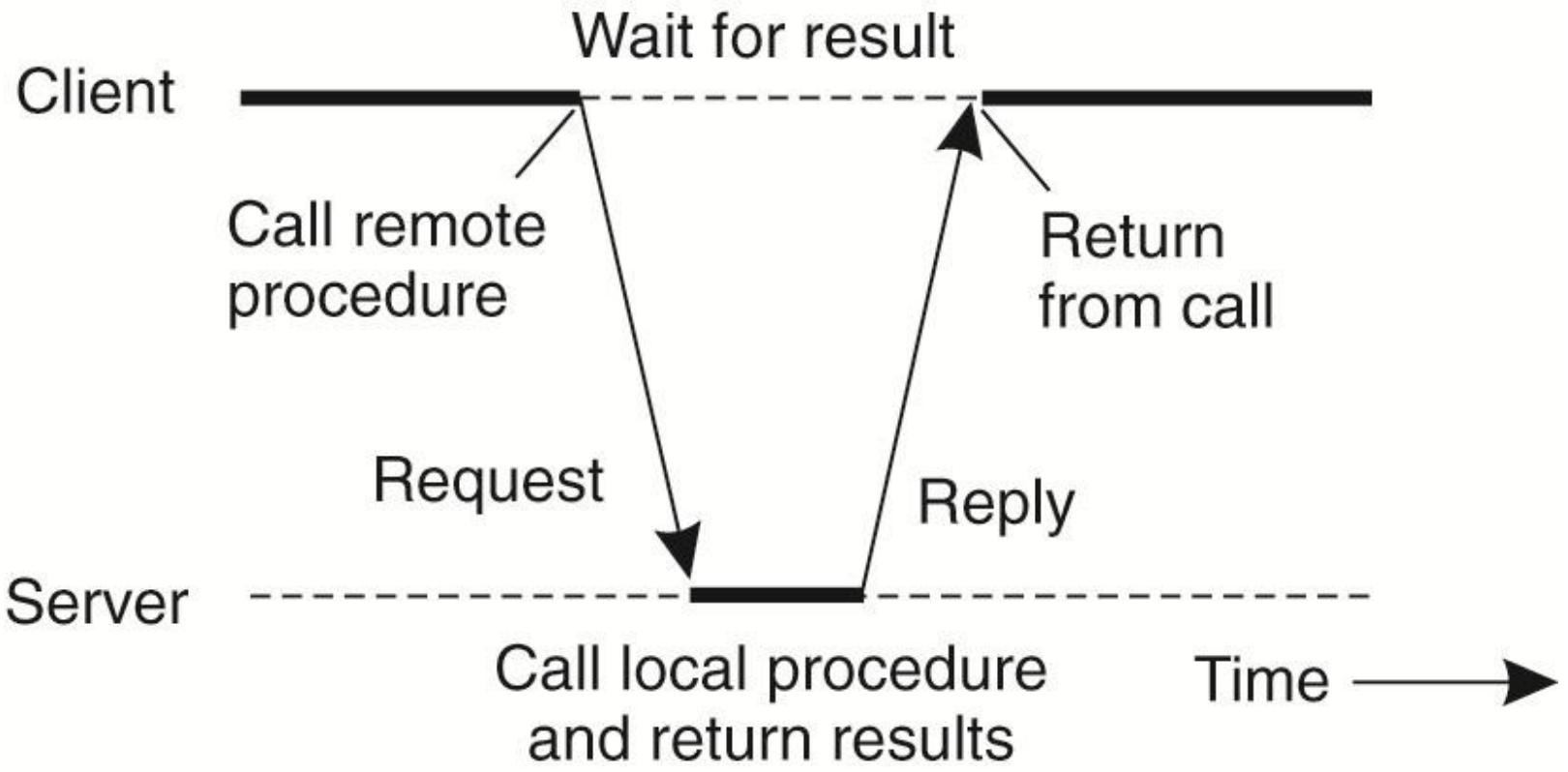
Conventional Procedure Call



(a) Parameter passing in a local procedure call: the stack before the call to read.

(b) The stack while the called procedure is active.

Client and Server Stubs



Principle of RPC between a client and server program.

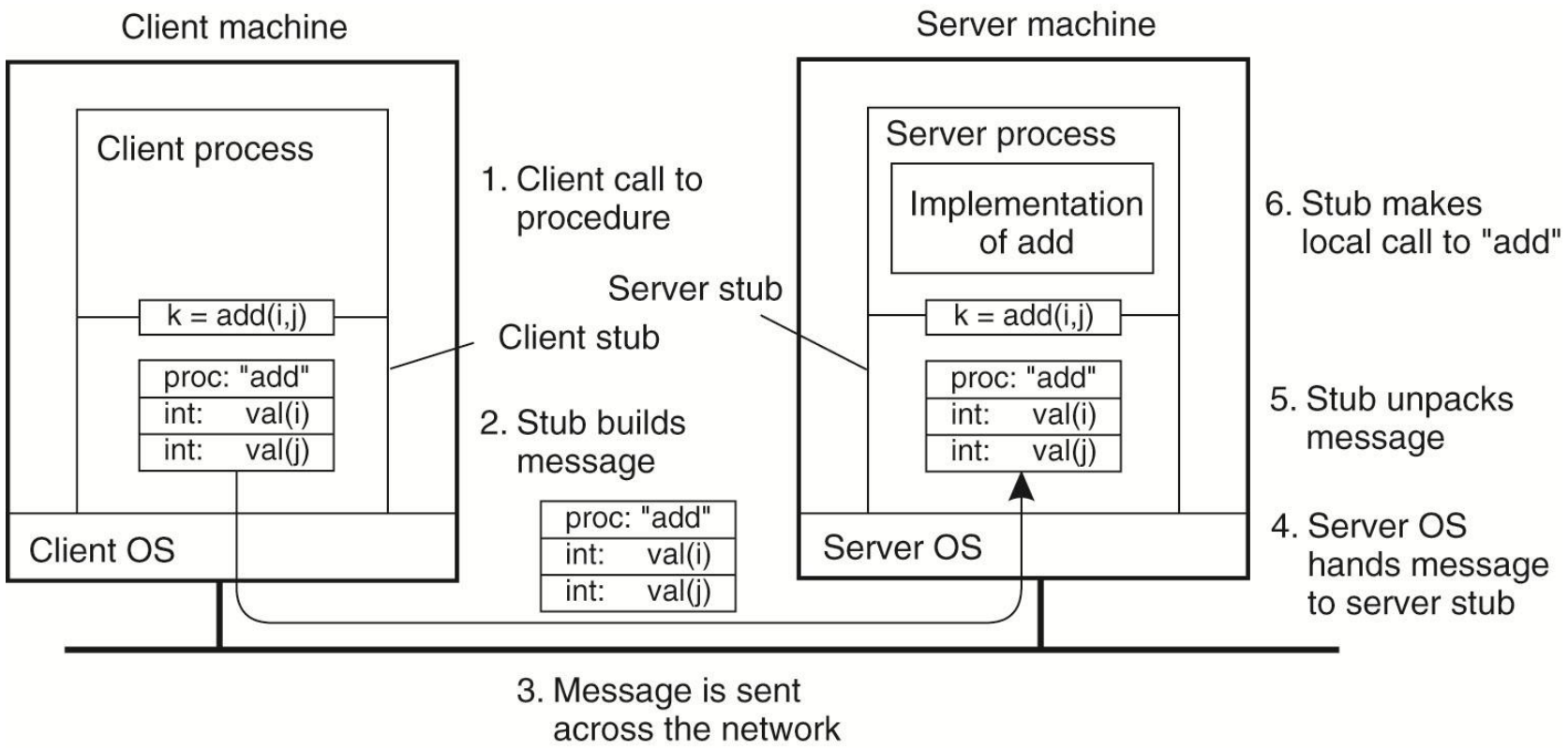
Remote Procedure Calls (1)

A remote procedure call occurs in the following steps:

1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message (marshalling the parameters) and calls the local operating system.
3. The client's OS sends the message across the network to the remote OS.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters
6. Calls the server implementing the function.

Continued …

RPC WITH VALUE PARAMETERS



The steps involved in a doing a remote computation through RPC.

Remote Procedure Calls (2)

A remote procedure call occurs in the following steps (continued):

1. The server does the work and returns the result to the stub.
2. The server stub packs it in a message and calls its local OS.
3. The server's OS sends the message across the network to the client's OS.
4. The client's OS gives the message to the client stub.
5. The stub unpacks the result and returns to the client.

Passing Reference Parameters

- A pointer is only meaningful in the address space of the process where it is used.
- One solution is to forbid pointers and reference parameters in RPC
- More typically, ***marshalling*** involves changing the mechanism to copy/restore
  - Actual parameter is sent as a copy
  - If the formal parameter is changed, the changed version is sent back to use as a replacement for the actual
  - Optimizations include taking advantage of the direction of the changes (input only, output only)
  - Straightforward for some parameter types (arrays for example) but not in general.

Parameter Specification and Stub Generation

(a) A procedure.

(b) The corresponding message.

```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)

foobar's local variables	
	x
y	
5	
z[0]	
z[1]	
z[2]	
z[3]	
z[4]	

array represented as a pointer is copied in the invocation

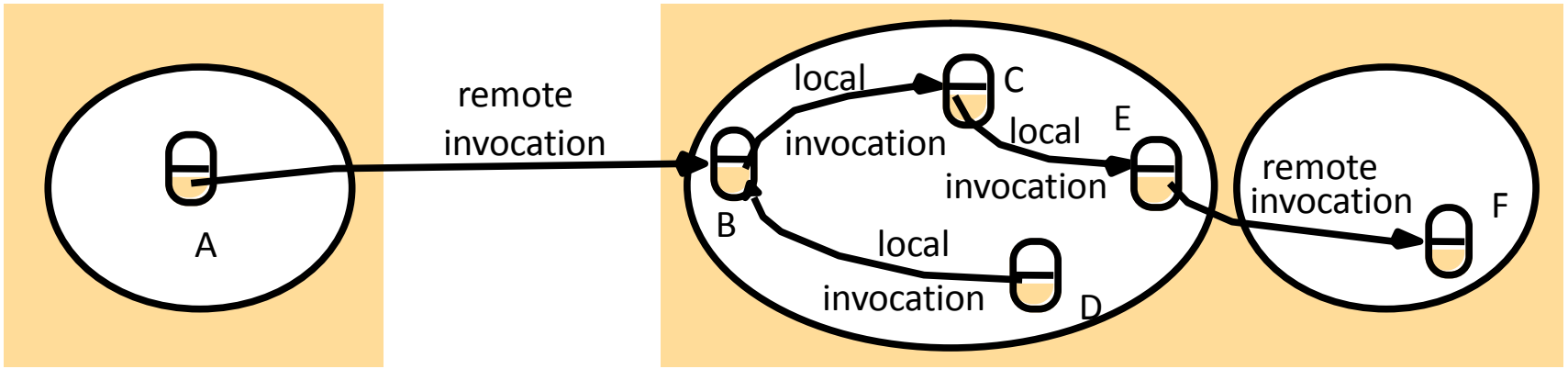
(b)

Communication protocols: Remote Method Invocation (RMI)

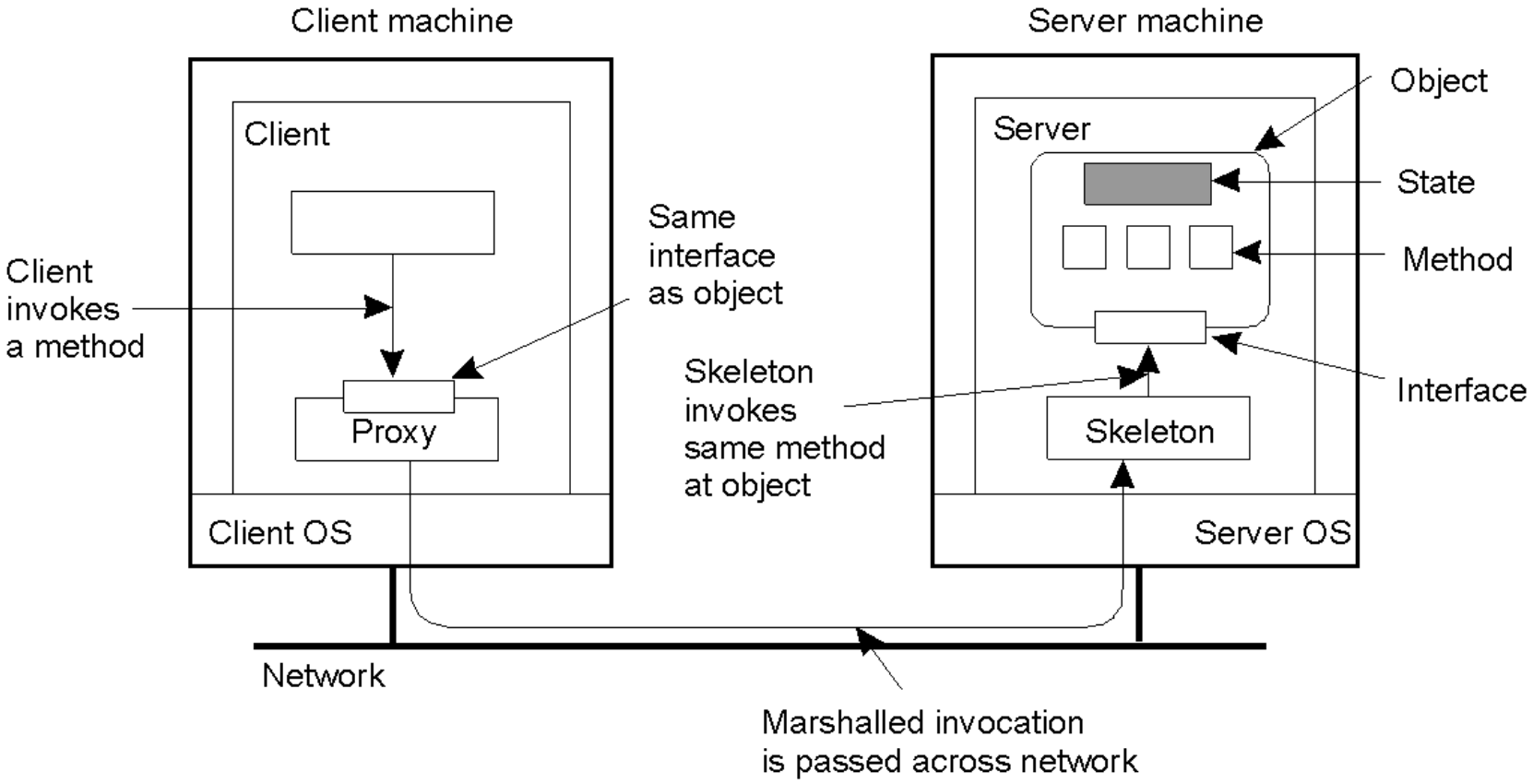
RMI

- RMI = RPC + Object-orientation
  - Java RMI
  - CORBA
    - Middleware that is language-independent
  - Microsoft DCOM/COM+
  - SOAP
    - RMI on top of HTTP

Remote and local method invocations



Distributed Objects



Common organization of a remote object with client-side proxy (loaded when the client binds to a remote object).

Distributed Objects

- Remote object references
  - An identifier that can be used throughout a distributed system to refer to a particular remote object
- Remote interfaces
  - JAVA RMI: Java interface that extends *Remote* interface
- Actions: remote invocations
- Remote Exceptions may arise for reasons such as partial failure or message loss
- Distributed Garbage Collection: cooperation between local garbage collectors needed

RMI Programming

- RMI software
  - Generated by IDL compiler
  - Proxy (client side)
    - Behaves like remote object to clients (invoker)
    - Marshals arguments, forwards message to remote object, unmarshals results, returns results to client
  - Skeleton (server side)
    - Server side stub;
    - Unmarshals arguments, invokes method, marshals results and sends to sending proxy's method
  - Dispatcher (server side)
    - Receives the request message from communication module, passes on the message to the appropriate method in the skeleton

RMI Programming

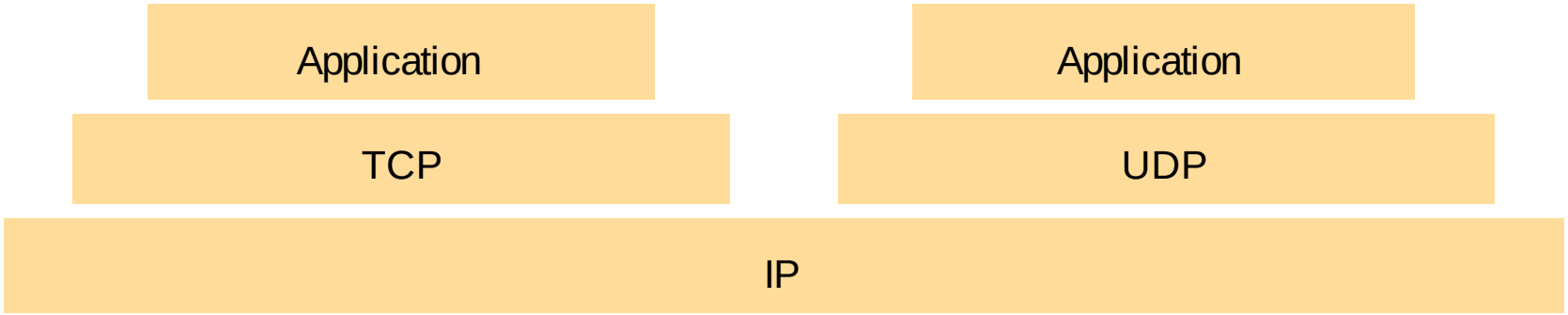
- Binder
  - Client programs need a means of obtaining a remote object reference
  - Binder is a service that maintains a mapping from textual names to remote object references
  - Servers need to register the services they are exporting with the binder
  - Java RMIregistry
- Server threads
  - Several choices: thread per object, thread per invocation
  - Remote method invocations must allow for concurrent execution

Java RMI

- Features
  - Integrated with Java language + libraries
    - Security, write once run anywhere, multithreaded
    - Object orientation
  - Can pass “behavior”
    - Mobile code
    - Not possible in CORBA, traditional RPC systems
  - Distributed Garbage Collection
  - *Remoteness of objects **intentionally** not transparent*

Communication protocols: Socket API

The programmer's conceptual view of a TCP/IP Internet



Socket programming

Goal: learn how to build client/server application that communicate using sockets

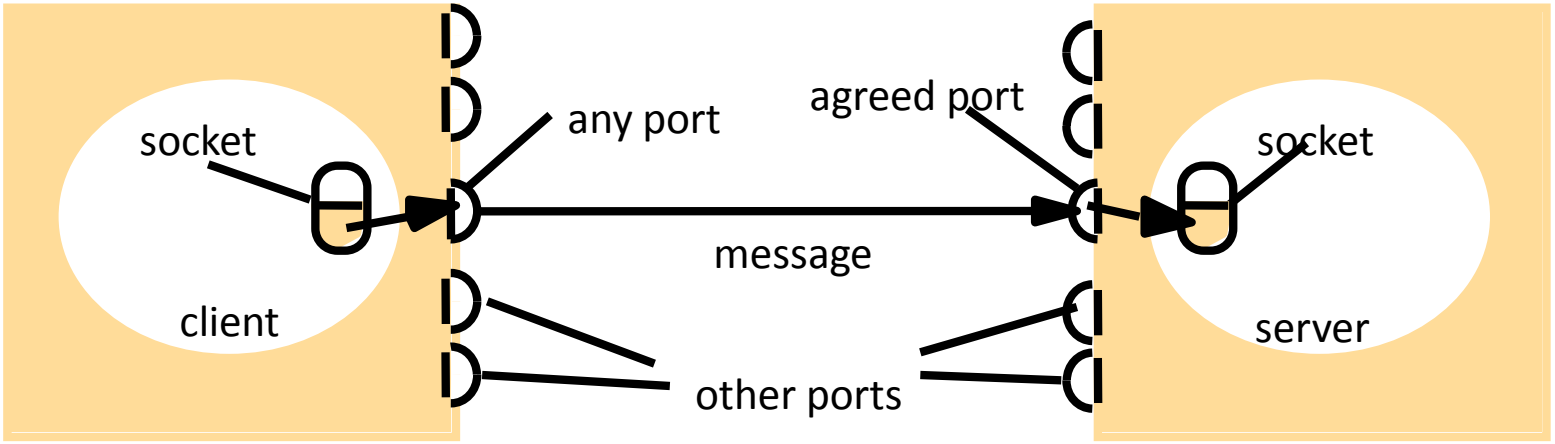
Socket API

- introduced in BSD4.1 UNIX
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram (UDP)
  - reliable, byte stream-oriented (TCP)

socket

a *host-local*, *application-created/owned*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another (remote or local) application process

Sockets and ports



Internet address = 138.37.94.248

Internet address = 138.37.88.249

Socket programming with TCP

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

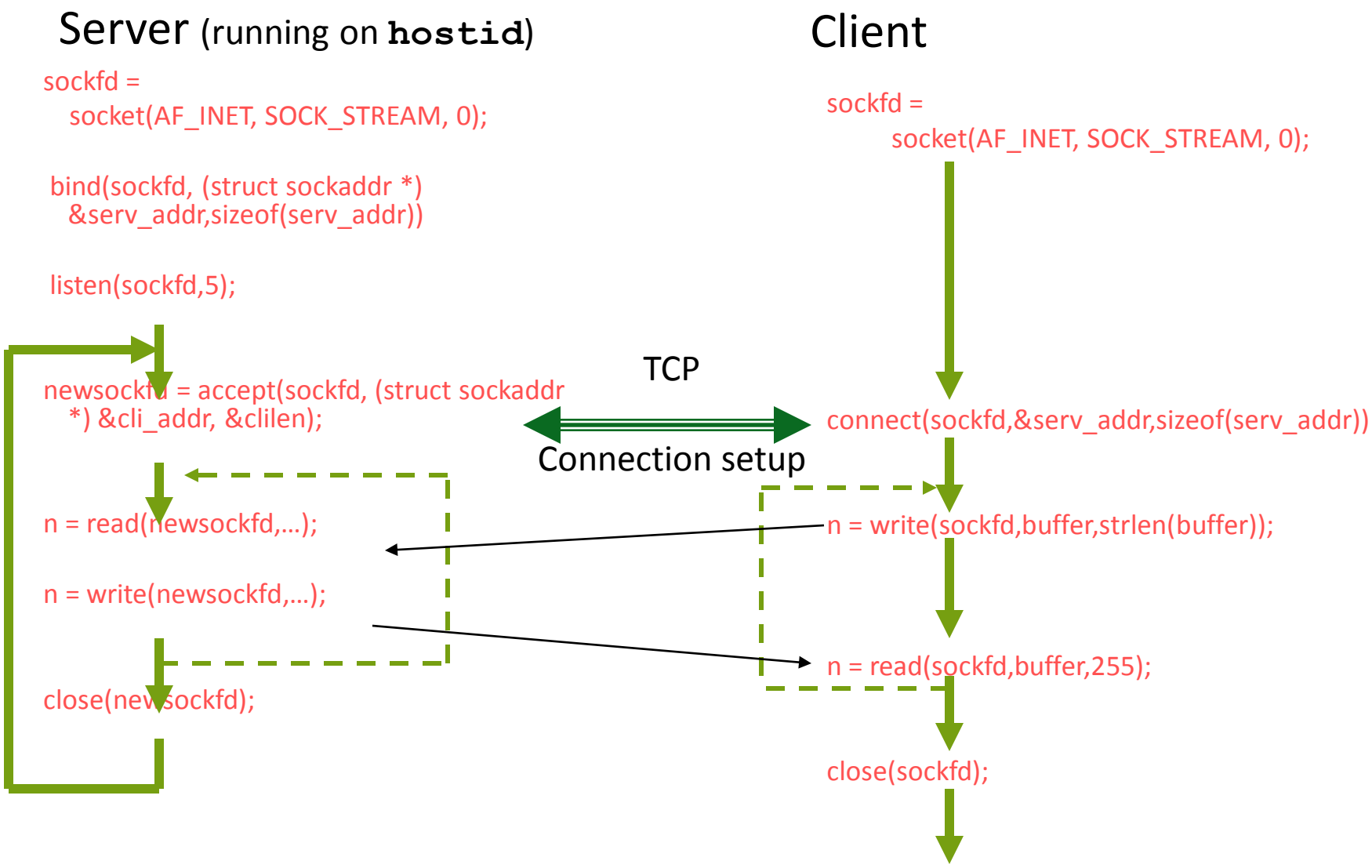
- creating client-local TCP socket
- specifying IP address, port number of server process

- When client creates socket: client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients

application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Client/server socket interaction: TCP

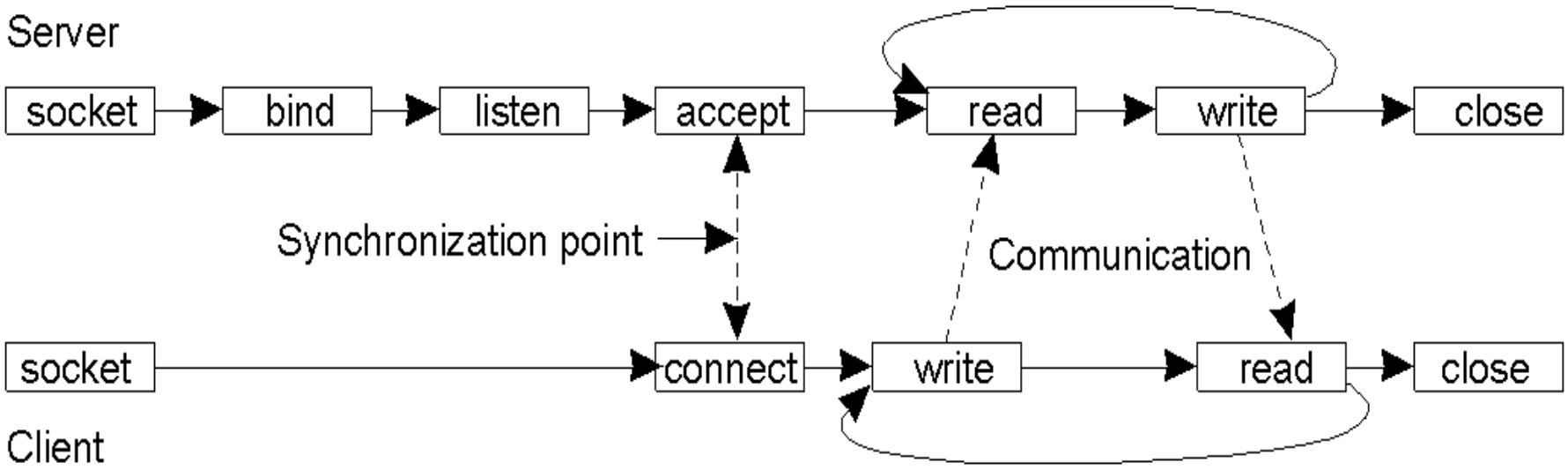


Berkeley Sockets API (1)

Socket primitives for TCP/IP.

Primitive	**Meaning**
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Berkeley Sockets (2)



Connection-oriented communication pattern using sockets.