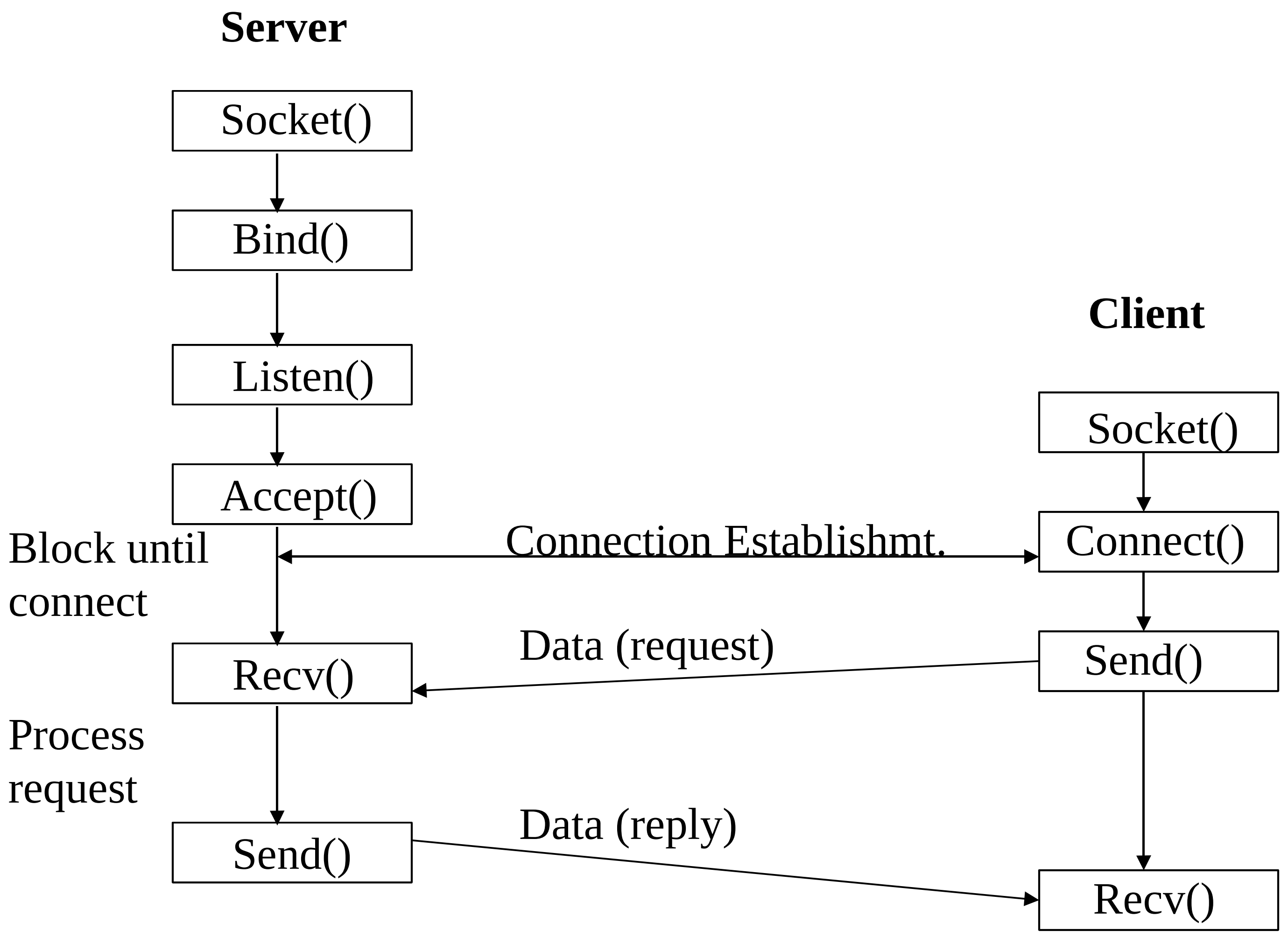
UNIX Sockets

Outline
- UNIX sockets

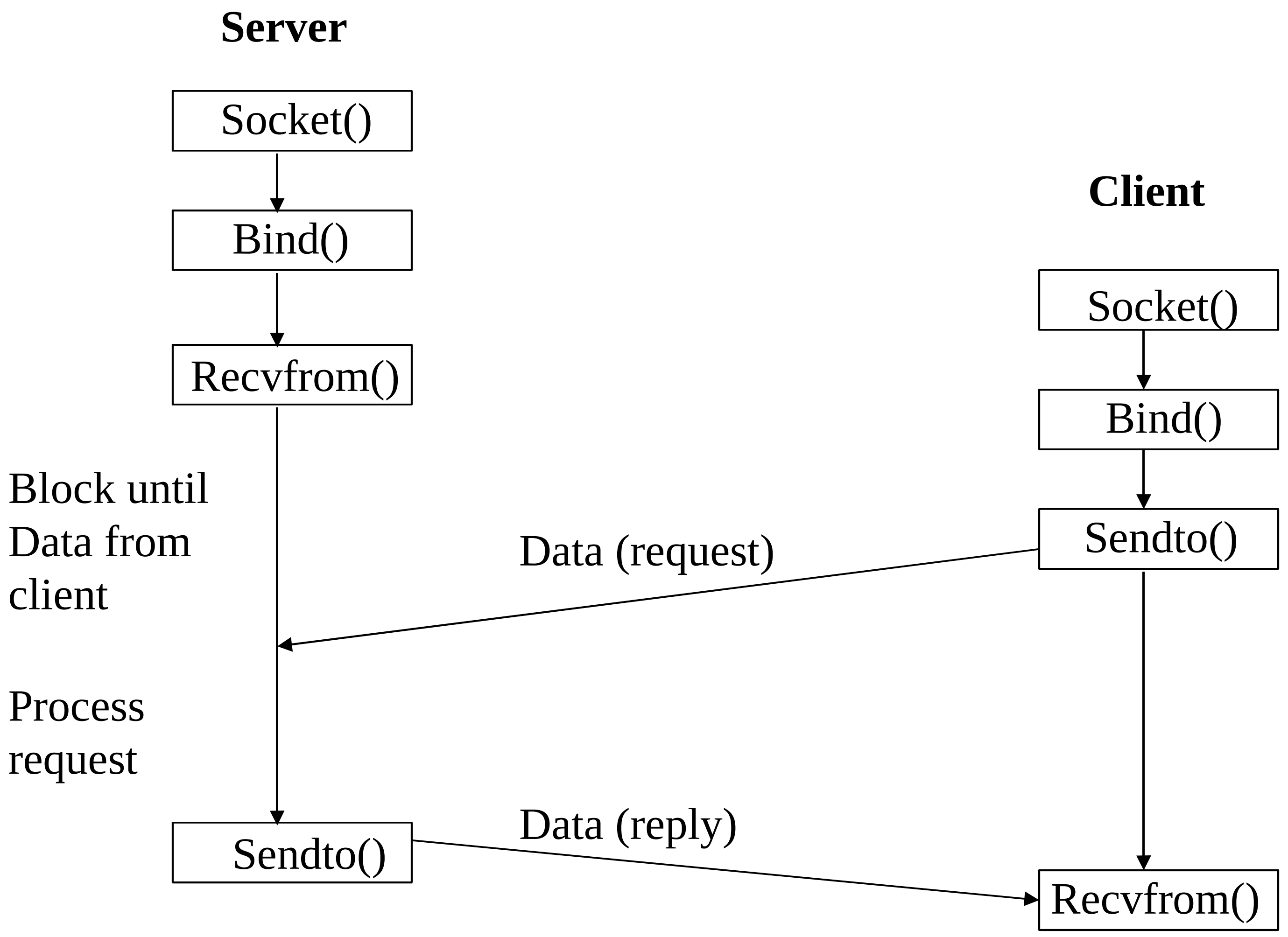
Berkeley Sockets

- Networking protocols are implemented as part of the OS
  - The networking API exported by most OS's is the *socket interface*
  - Originally provided by BSD 4.1c ~1982.
- The principal abstraction is a socket
  - Point at which an application attaches to the network
  - Defines operations for creating connections, attaching to network, sending/receiving data, closing.

Connection-oriented example (TCP)



Connectionless example (UDP)



Socket call

- Means by which an application attached to the network
- int socket(int family, int type, int protocol)
- *Family*: address family (protocol family)
  - AF_UNIX, AF_INET, AF_NS, AF_IMPLINK
- *Type*: semantics of communication
  - SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
  - Not all combinations of family and type are valid
- *Protocol*: Usually set to 0 but can be set to specific value.
  - Family and type usually imply the protocol
- Return value is a *handle* for new socket

Bind call

- Binds a newly created socket to the specified address
- Int bind(int socket, struct sockaddr *address, int addr_len)
- *Socket*: newly created socket handle
- *Address*: data structure of address of *local* system
  - IP address and port number (demux keys)
  - Same operation for both connection-oriented and connectionless servers
    - Can use well known port or unique port

Listen call

- Used by connection-oriented servers to indicate an application is willing to receive connections
- Int(int socket, int backlog)
- *Socket*: handle of newly creates socket
- *Backlog*: number of connection requests that can be queued by the system while waiting for server to execute accept call.

Accept call

- After executing *listen*, the accept call carries out a *passive open* (server prepared to accept connects).
- Int accept(int socket, struct sockaddr *address, int addr_len)
- It blocks until a remote client carries out a connection request.
- When it does return, it returns with a *new* socket that corresponds with new connection and the address contains the clients address

Connect call

- Client executes an *active open* of a connection
- Int connect(int socket, struct sockaddr *address, int addr_len)
- Call does not return until the three-way handshake (TCP) is complete
- Address field contains remote system's address
- Client OS usually selects random, unused port

Send(to), Recv(from)

- After connection has been made, application uses send/recv to data
- Int send(int socket, char *message, int msg_len, int flags)
  - Send specified message using specified socket
- Int recv(int scoket, char *buffer, int buf_len, int flags)
  - Receive message from specified socket into specified buffer

Socket Implimentation

- Protocol implementation
  - Process per protocol
    - Use a separate process to implement each protocol
    - Messages are passes between processes
  - Process per message
    - Use one process to handle each message/communication
    - Generally more efficient
- Buffer use
  - Applications use buffers as do protocols
    - Copies are VERY expensive
    - Message abstraction enables pointers to be used and minimal copies

Practical issues – using sockets

- You have to be *very* careful when using these calls
  - Specific data structures and formats
  - Ports cannot be less than 1024
- You can use other tools to see if things are working
  - Tcpdump
  - /proc
  - netstat
- Client and server can be on same system
- Think about error handling methods
- Check reference text
- Baby steps!!