Sockets

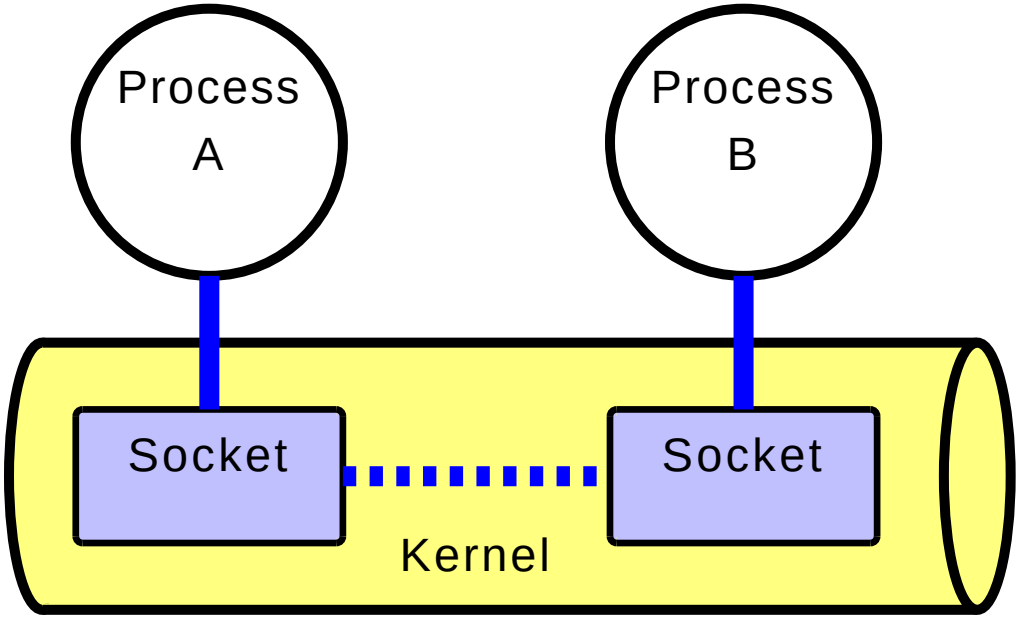
Distributed Systems

Philipp Kupferschmied

Universität Karlsruhe, System Architecture Group

May 6th, 2009

What are Sockets?



- Connection oriented IPC
- "Phone connection" between processes
- Protocol determines how the connection is used

Socket Types

- Socket
  - File descriptor (sock_fd)

Socket Types

- Socket
  - File descriptor (sock_fd)
- Protocol family
  - Unix-domain (AF_UNIX)
  - Internet-domain (AF_INET)

Socket Types

- Socket
  - File descriptor (sock_fd)
- Protocol family
  - Unix-domain (AF_UNIX)
  - Internet-domain (AF_INET)
- Socket type
  - Stream (SOCK_STREAM)
  - Datagram (SOCK_DGRAM)

Socket Types

- Socket
  - File descriptor (sock_fd)
- Protocol family
  - Unix-domain (AF_UNIX)
  - Internet-domain (AF_INET)
- Socket type
  - Stream (SOCK_STREAM)
  - Datagram (SOCK_DGRAM)
- Protocol
  - UNIX_STREAM, UNIX_DGRAM
  - TCP, UDP

Unix-Domain Sockets

- Implemented purely in memory
- Users can bind a socket to an "address"
- Socket becomes accessible at this "address"
- Unix-domain addresses are file names
  - /tmp/server

sockaddr_un

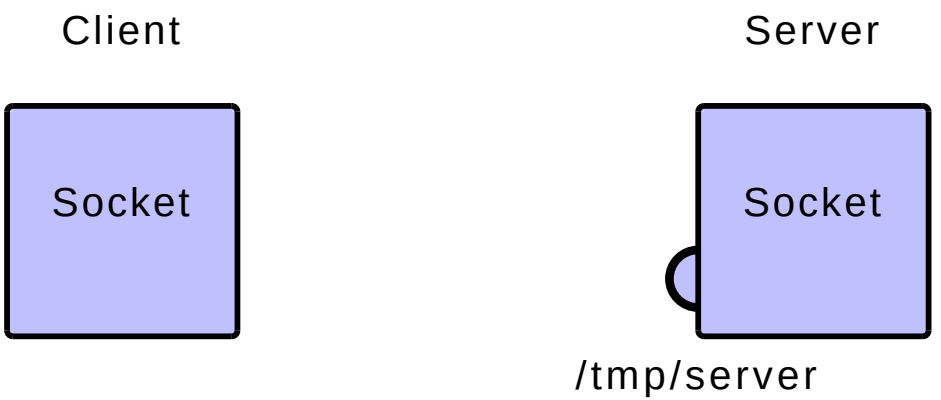
sun_family = AF_UNIX
sun_path = /tmp/server

Unix-Domain Sockets

- Implemented purely in memory
- Users can bind a socket to an "address"
- Socket becomes accessible at this "address"
- Unix-domain addresses are file names
  - /tmp/server

sockaddr_un

sun_family = AF_UNIX
sun_path = /tmp/server

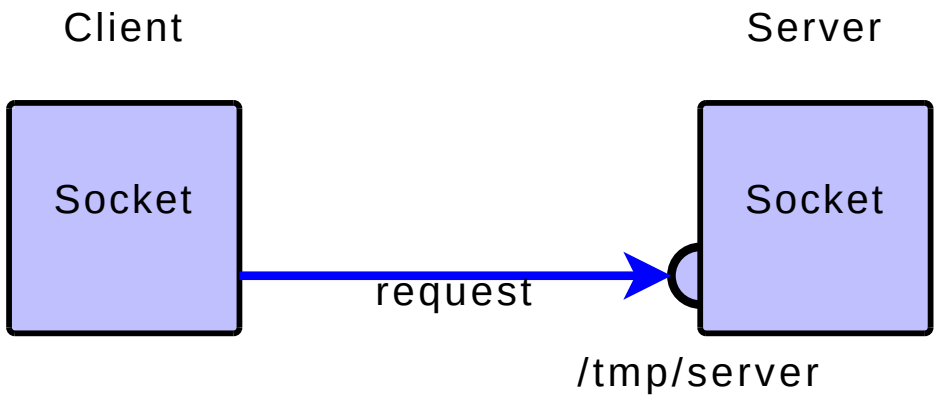


Unix-Domain Sockets

- Implemented purely in memory
- Users can bind a socket to an "address"
- Socket becomes accessible at this "address"
- Unix-domain addresses are file names
  - /tmp/server

sockaddr_un

sun_family = AF_UNIX
sun_path = /tmp/server

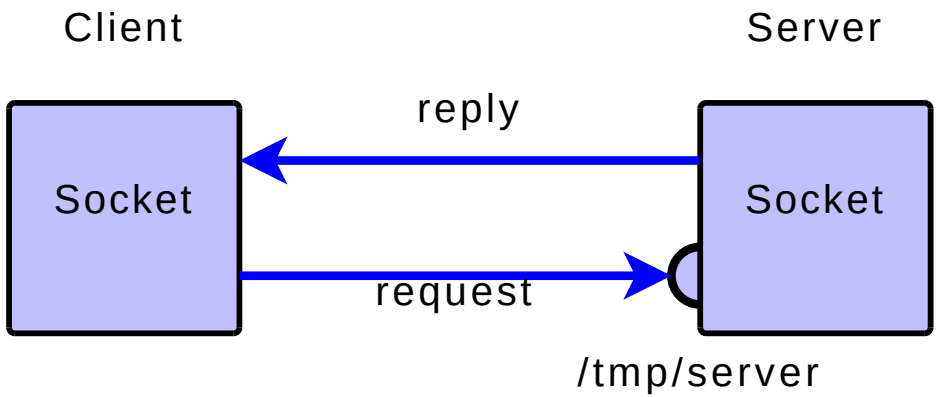


Unix-Domain Sockets

- Implemented purely in memory
- Users can bind a socket to an "address"
- Socket becomes accessible at this "address"
- Unix-domain addresses are file names
  - /tmp/server

sockaddr_un

sun_family = AF_UNIX
sun_path = /tmp/server



Internet-Domain Sockets

- Implemented via the network
- Users can bind a socket to an "address"
- Socket becomes accessible at this "address"
- Internet-domain addresses consist of IP and port number
  - 192.168.0.1:80

sockaddr_in

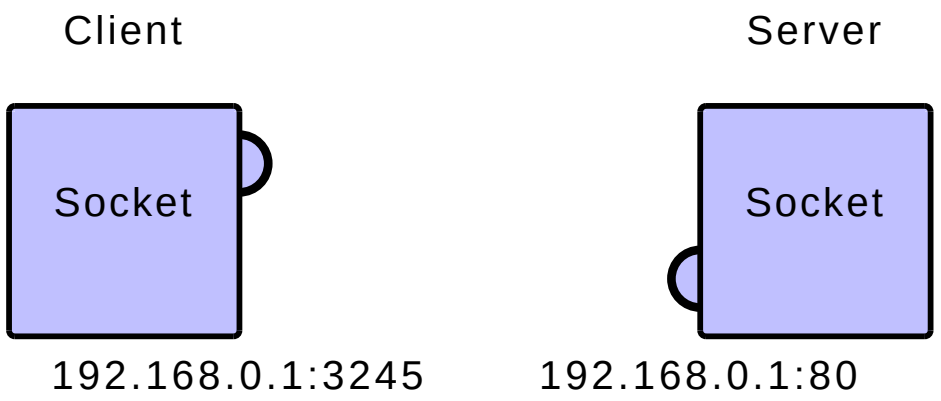
sin_family = AF_INET
sin_port = 80
sin_addr = 0x0100A8C0

Internet-Domain Sockets

- Implemented via the network
- Users can bind a socket to an "address"
- Socket becomes accessible at this "address"
- Internet-domain addresses consist of IP and port number
  - 192.168.0.1:80

sockaddr_in

sin_family = AF_INET
sin_port = 80
sin_addr = 0x0100A8C0

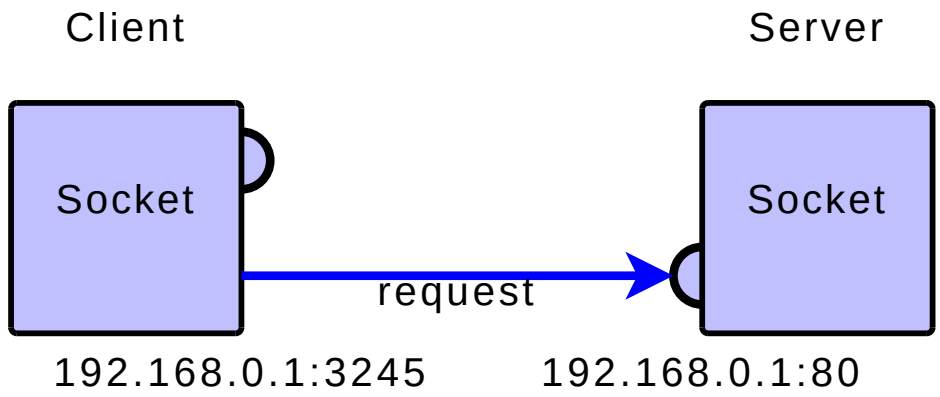


Internet-Domain Sockets

- Implemented via the network
- Users can bind a socket to an "address"
- Socket becomes accessible at this "address"
- Internet-domain addresses consist of IP and port number
  - 192.168.0.1:80

sockaddr_in

sin_family = AF_INET
sin_port = 80
sin_addr = 0x0100A8C0

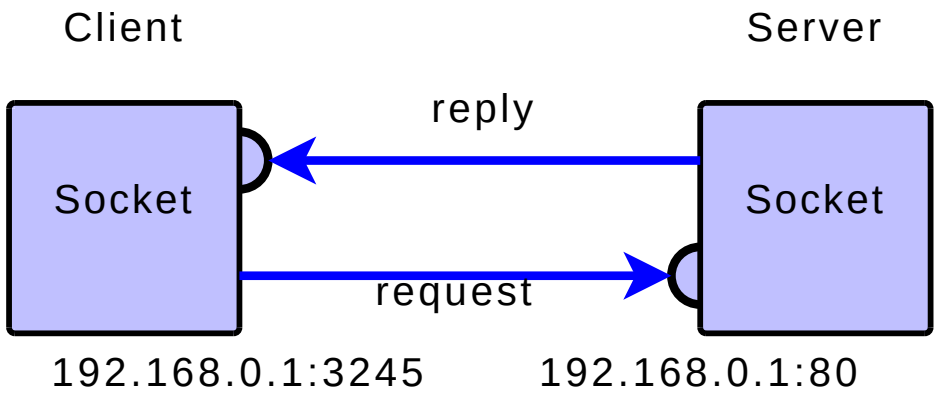


Internet-Domain Sockets

- Implemented via the network
- Users can bind a socket to an "address"
- Socket becomes accessible at this "address"
- Internet-domain addresses consist of IP and port number
  - 192.168.0.1:80

sockaddr_in

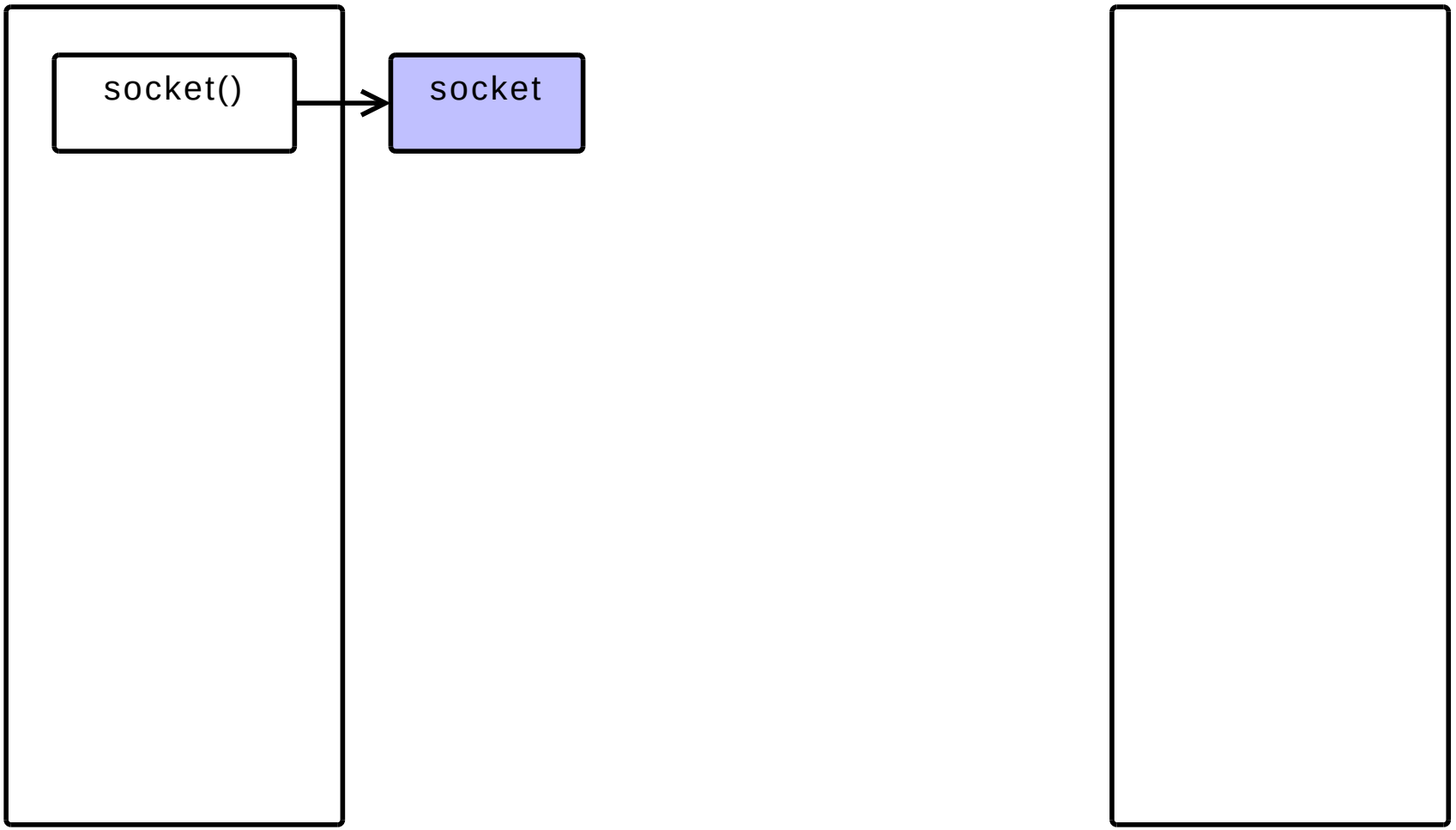
sin_family = AF_INET
sin_port = 80
sin_addr = 0x0100A8C0



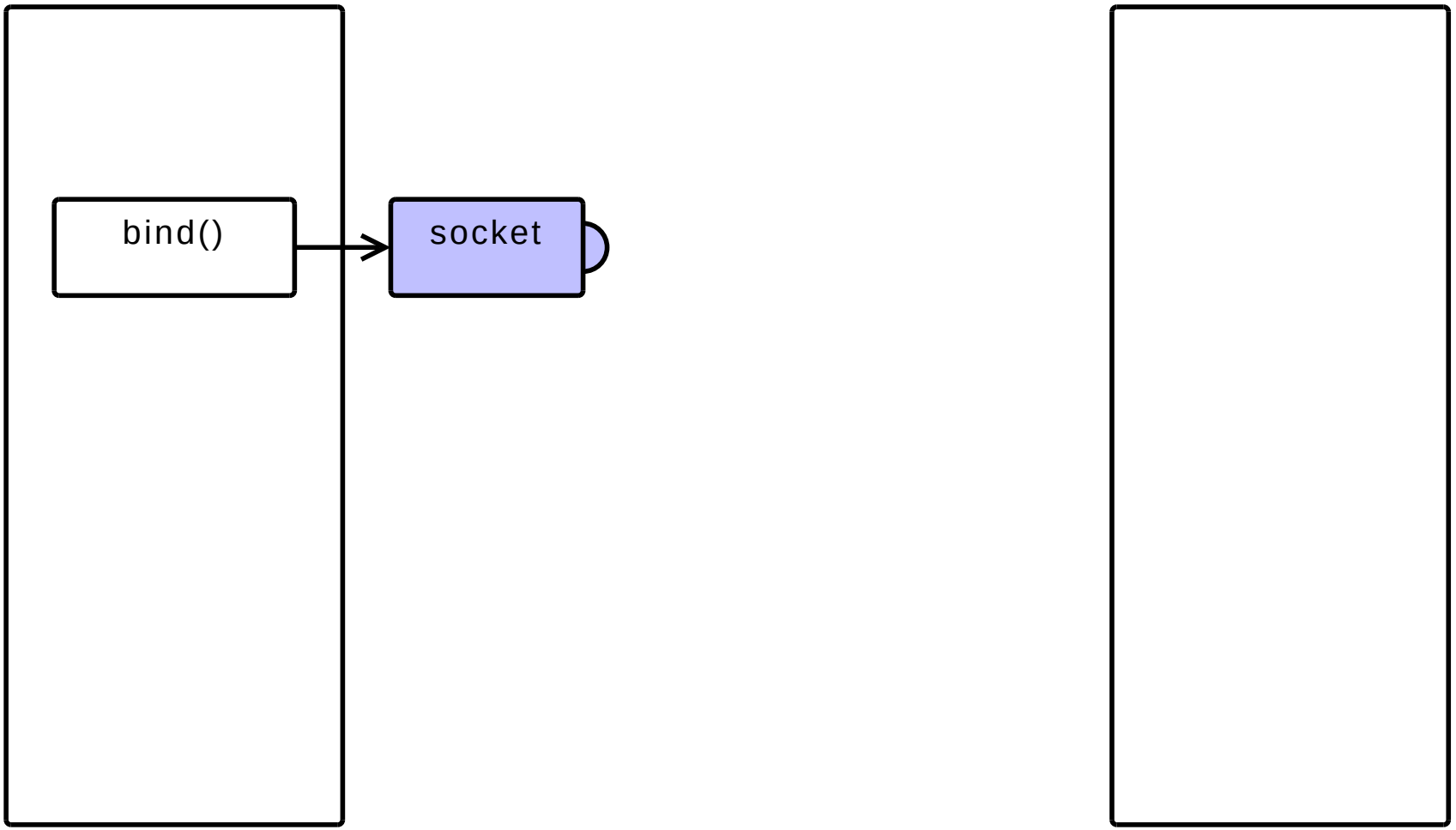
Characteristics

- Communication between two sockets
  - Connection-oriented or
  - Packet-based
- Allow for error control (TCP, UNIX_STREAM)
- Network-capable IPC
- Flexible
  - AX_25, APPLETALK, IPX, HTTP, RPC, ...

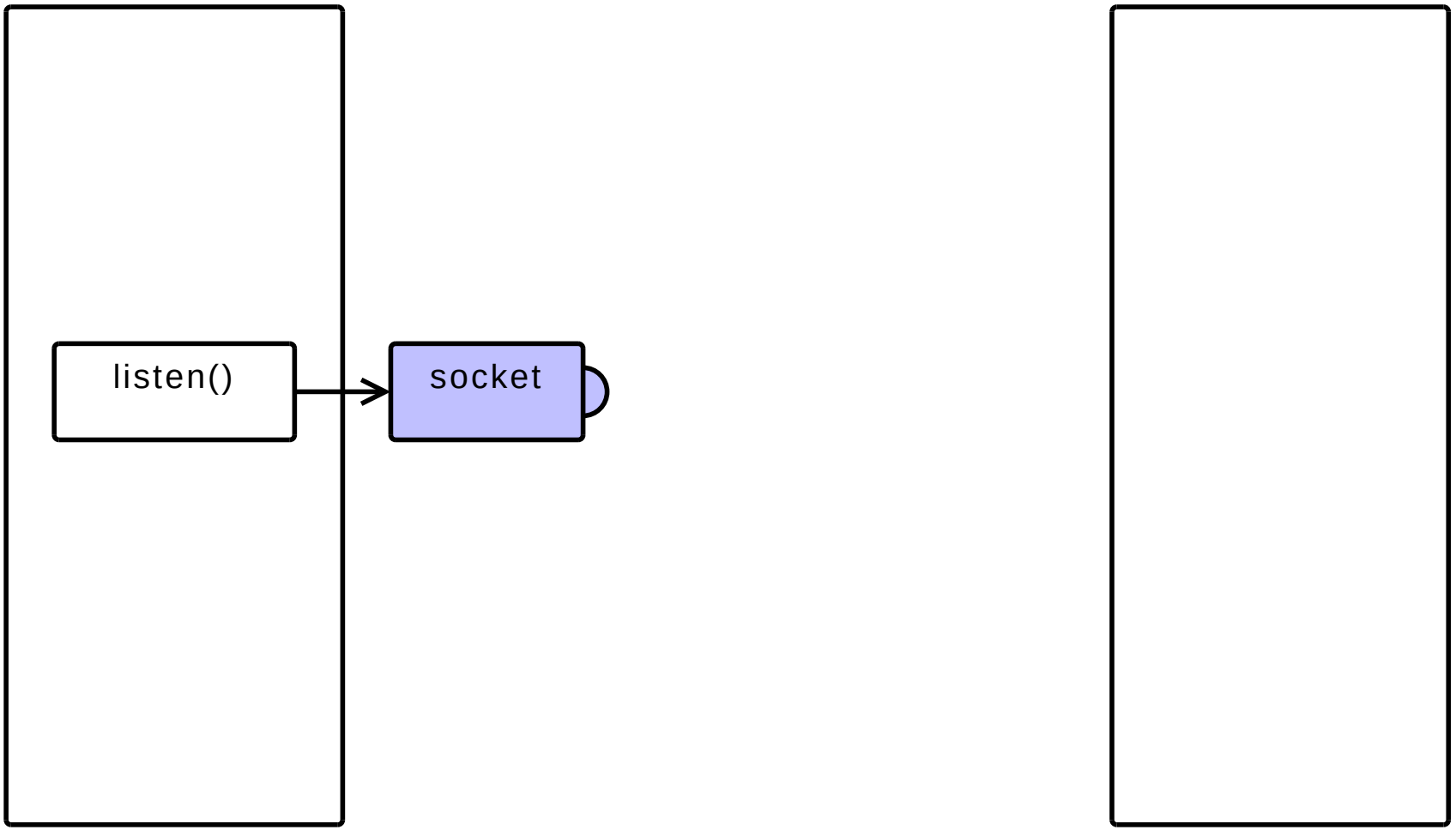
Establishing a Connection



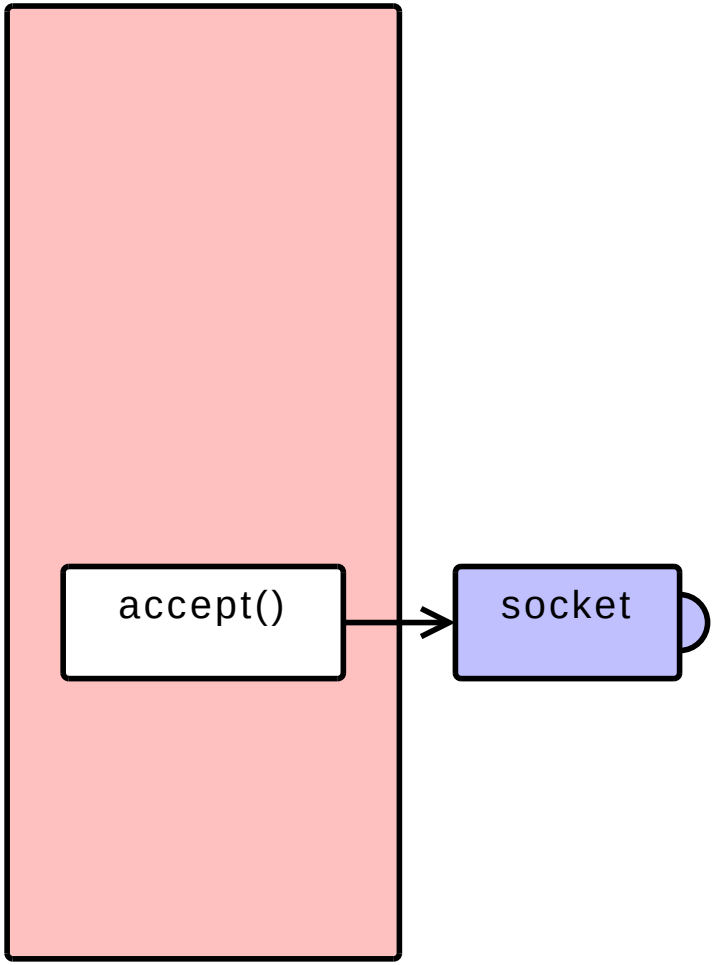
Establishing a Connection

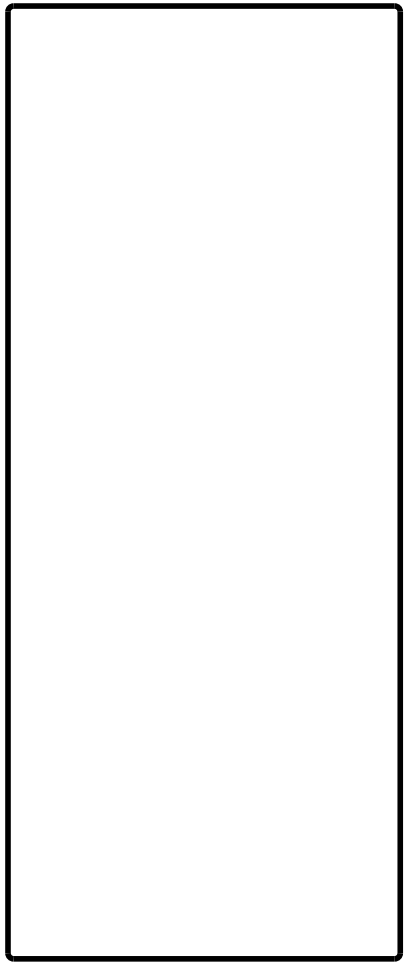


Establishing a Connection

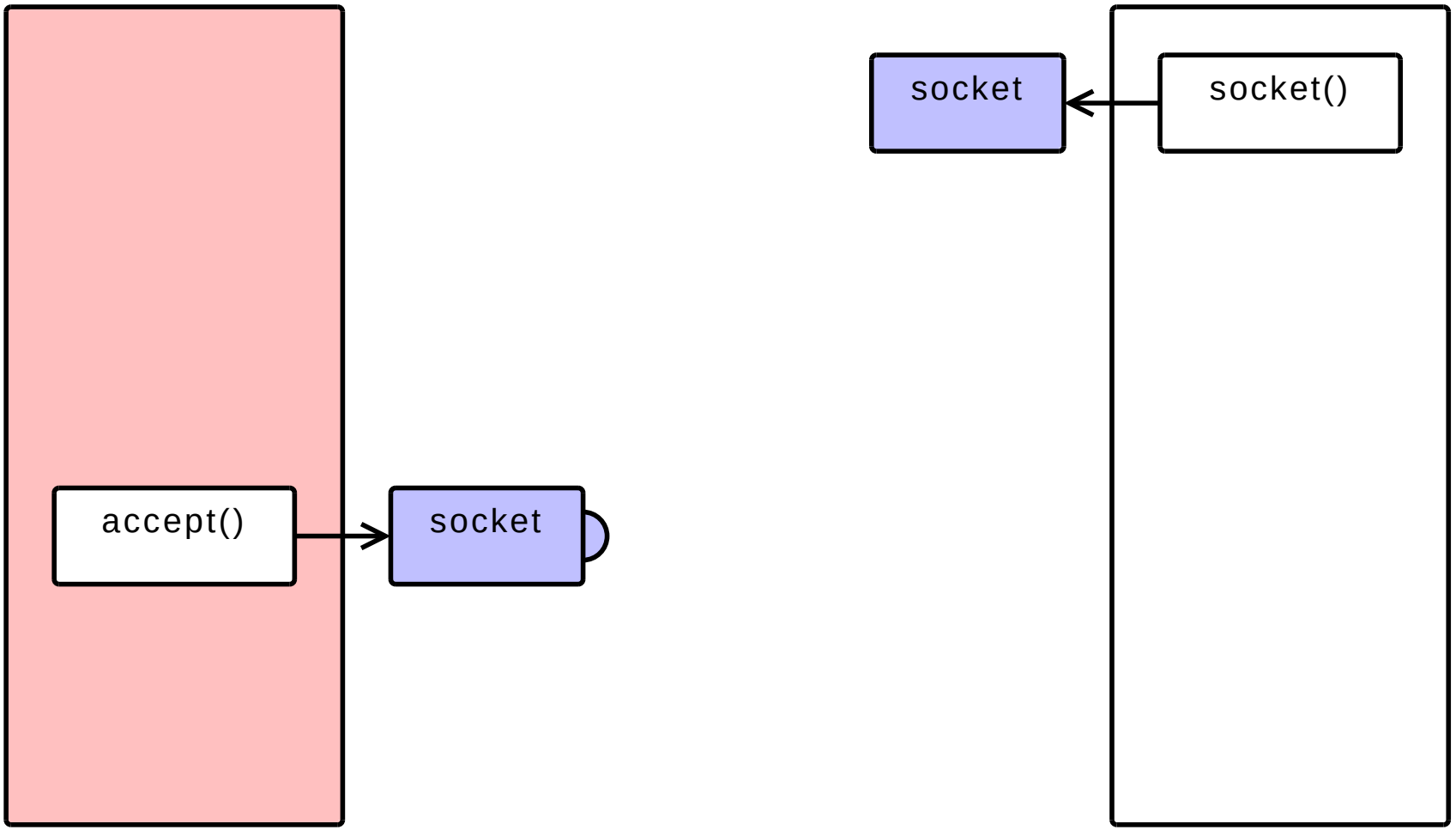


Establishing a Connection

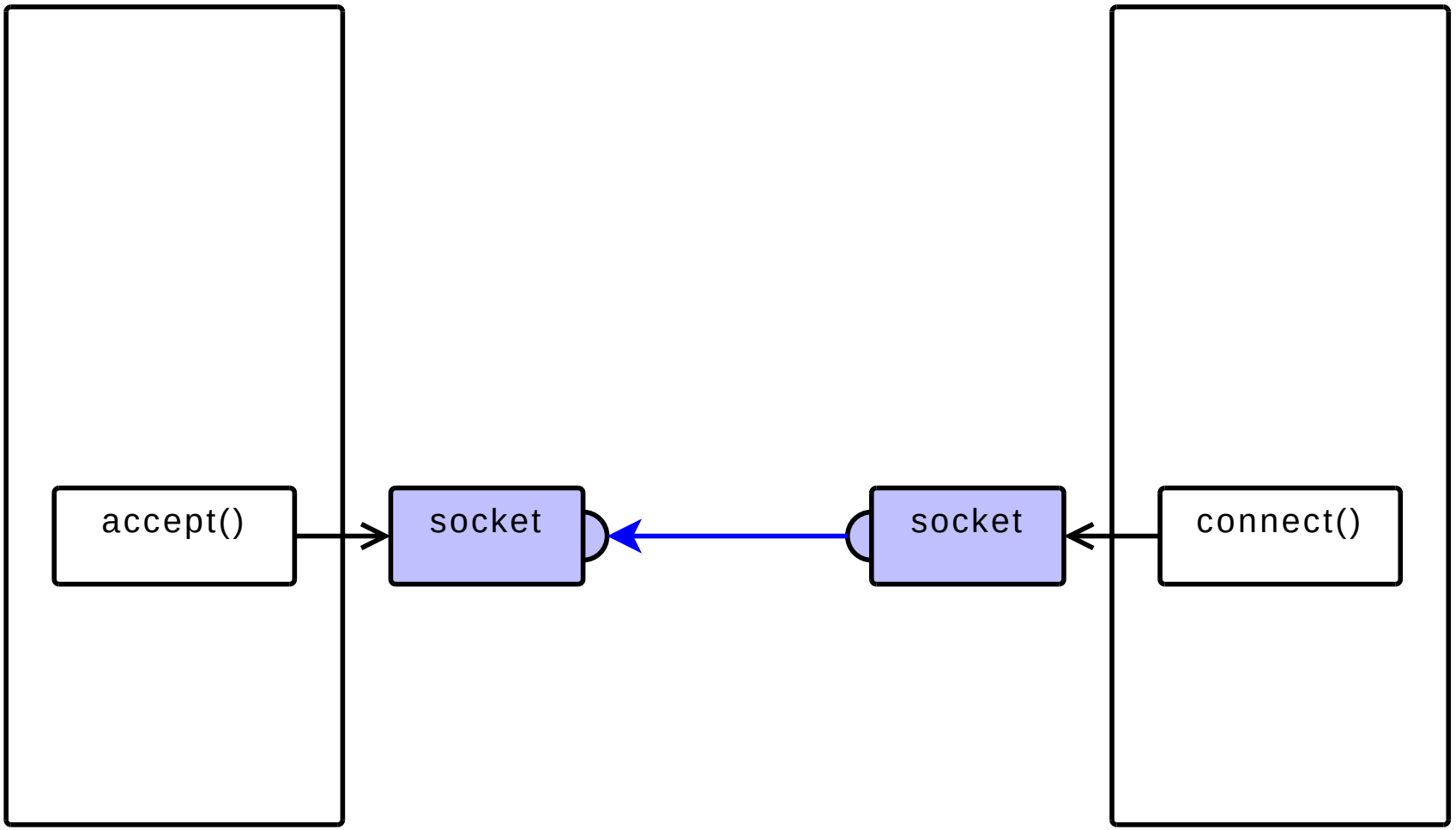




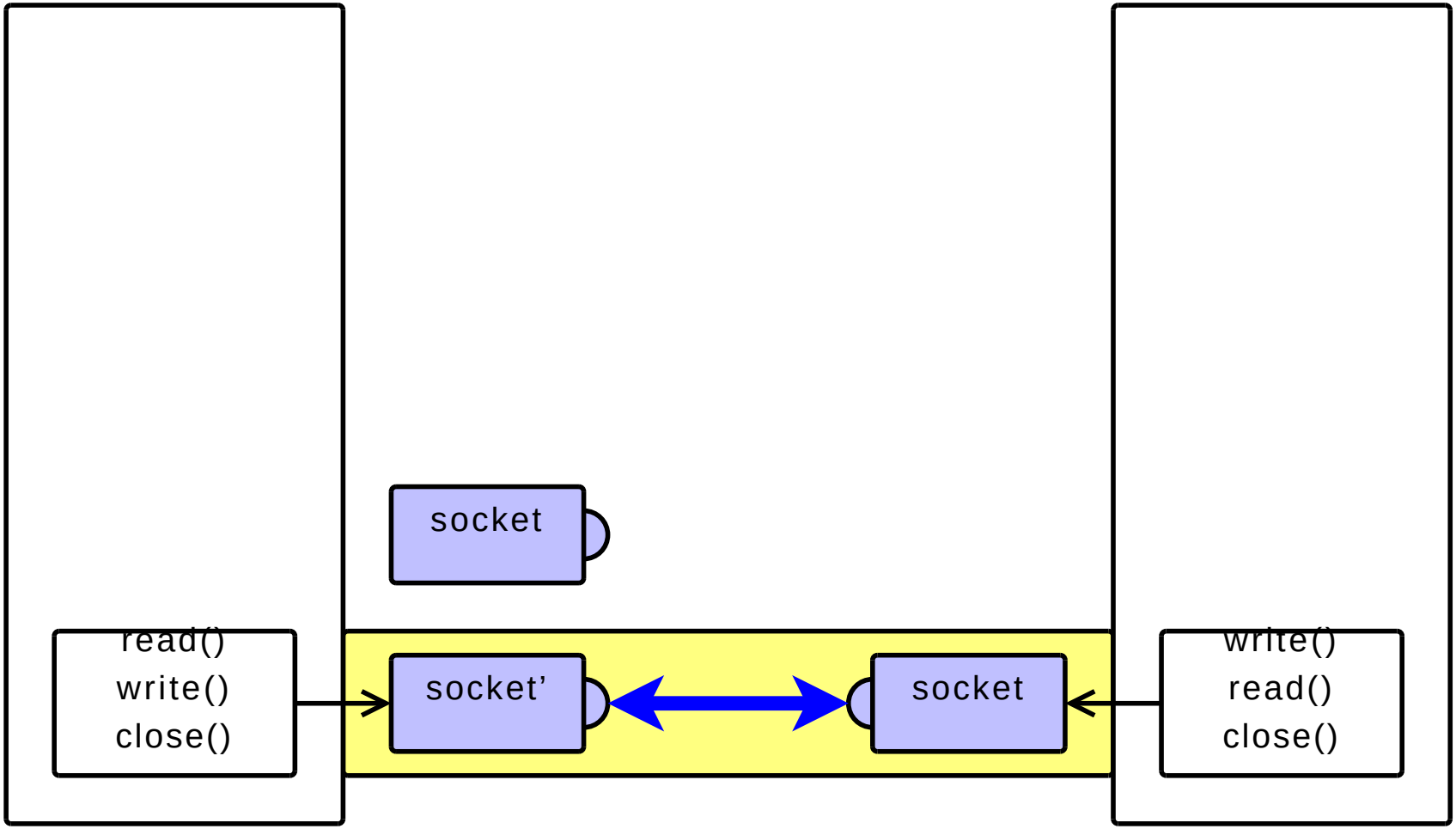
Establishing a Connection



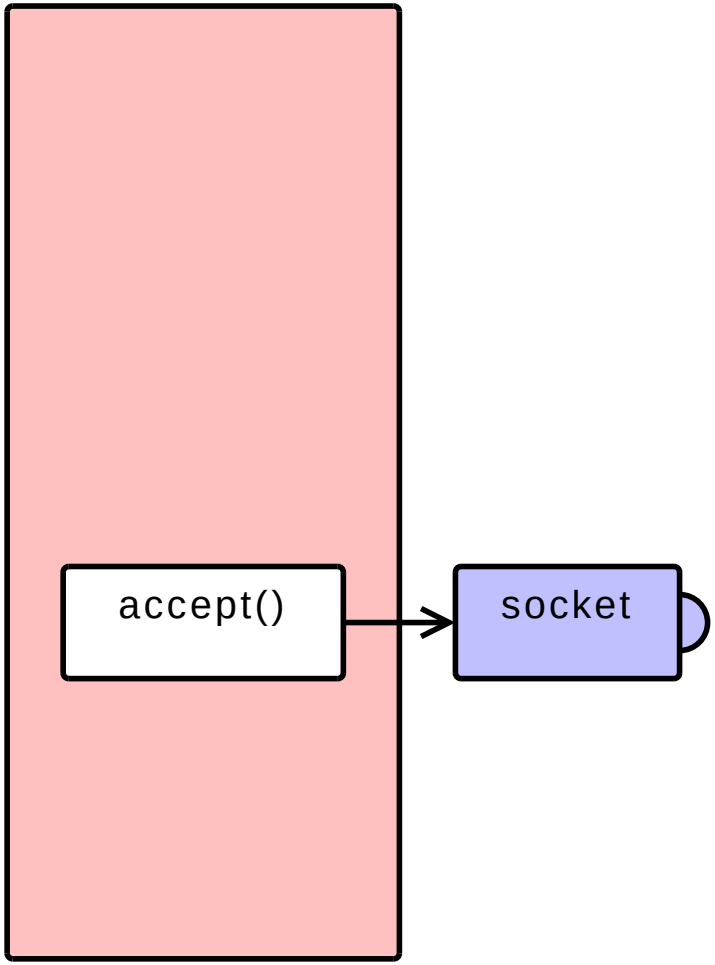
Establishing a Connection

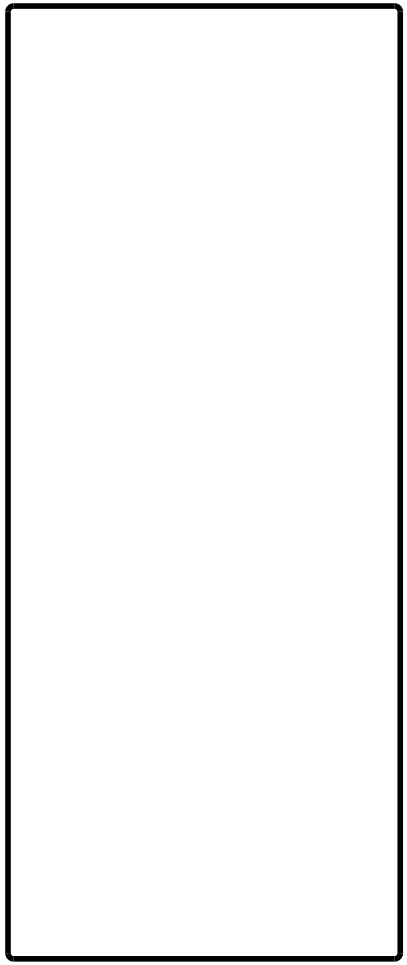


Establishing a Connection

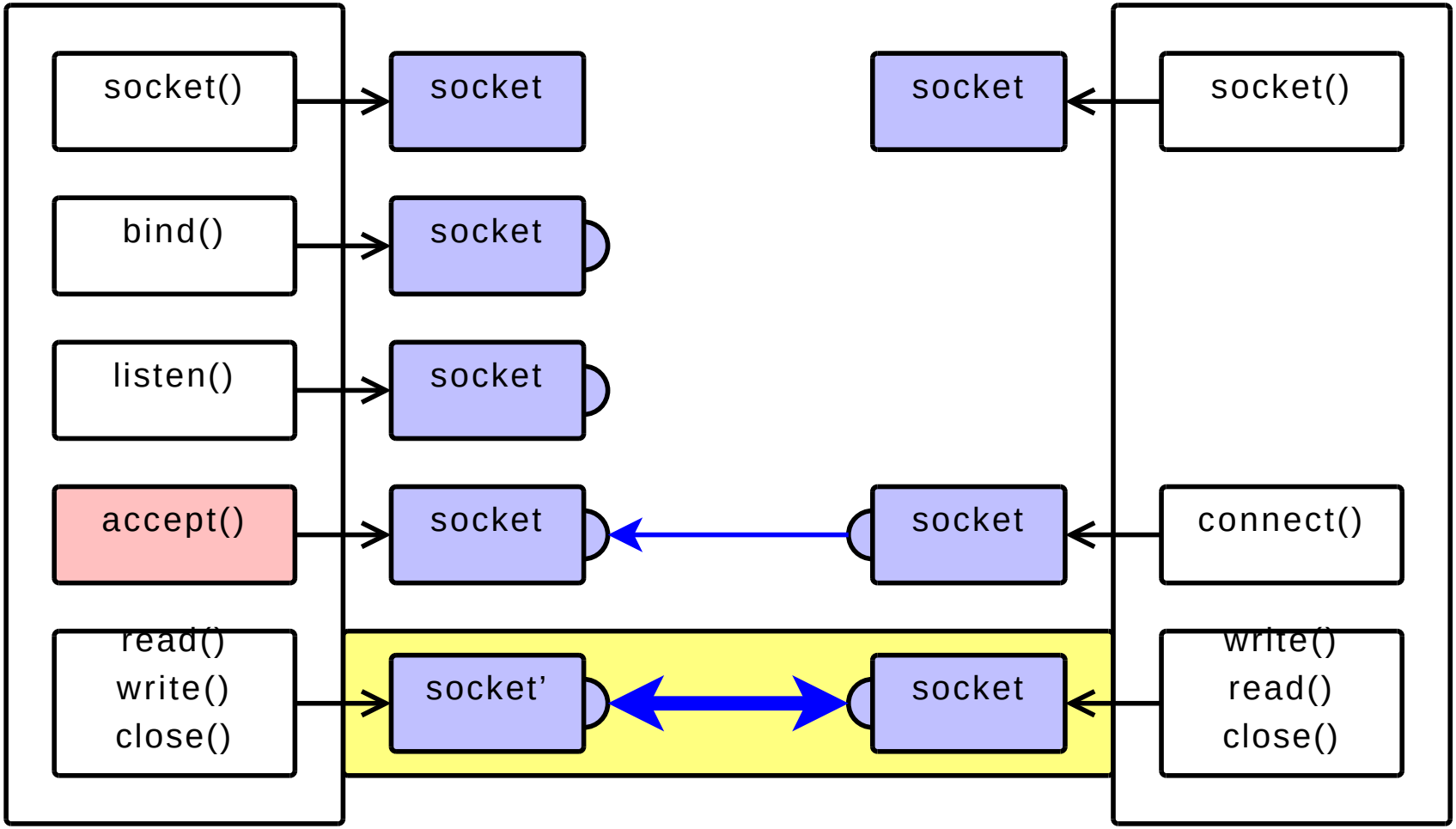


Establishing a Connection

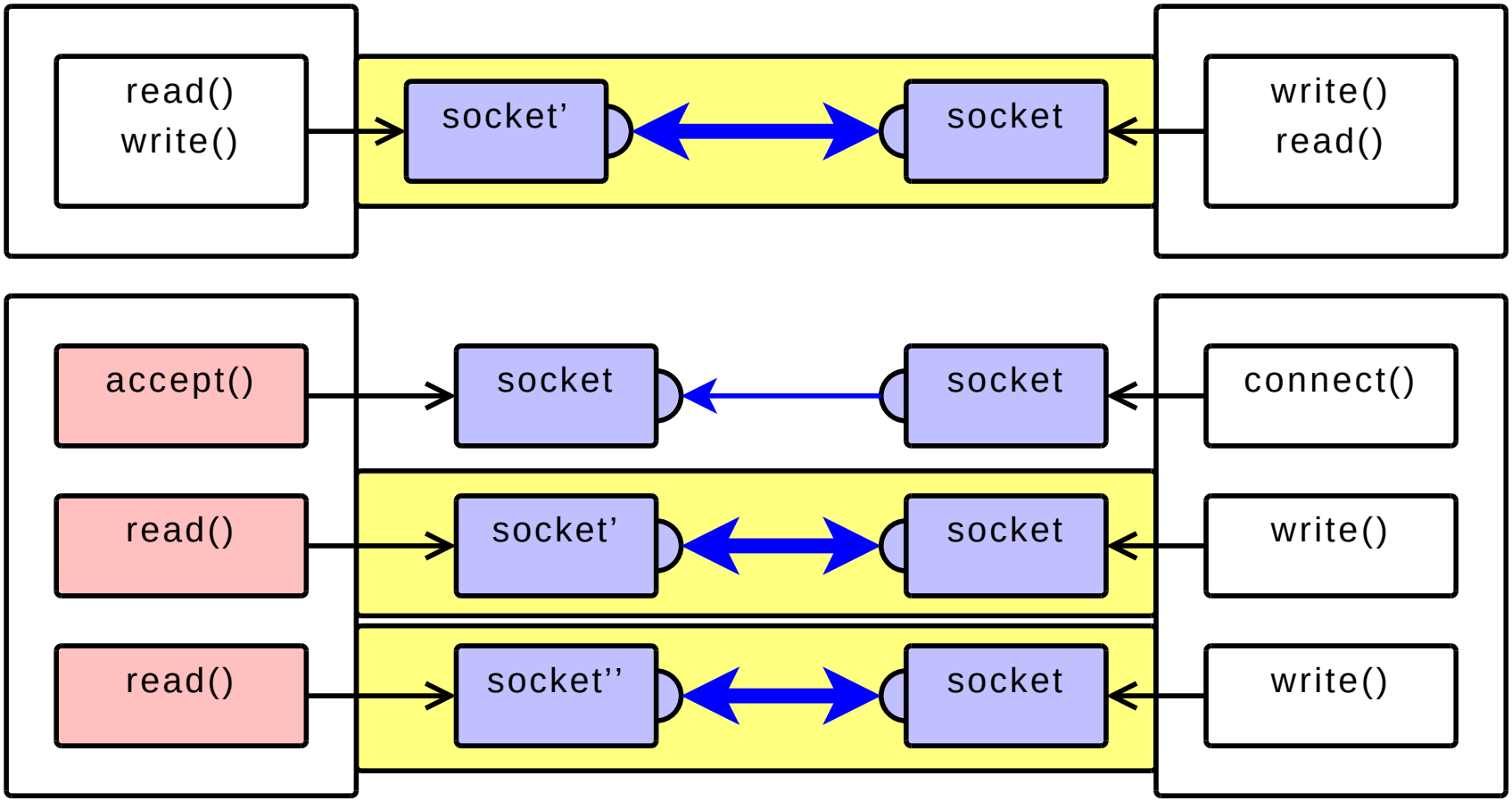




Establishing a Connection

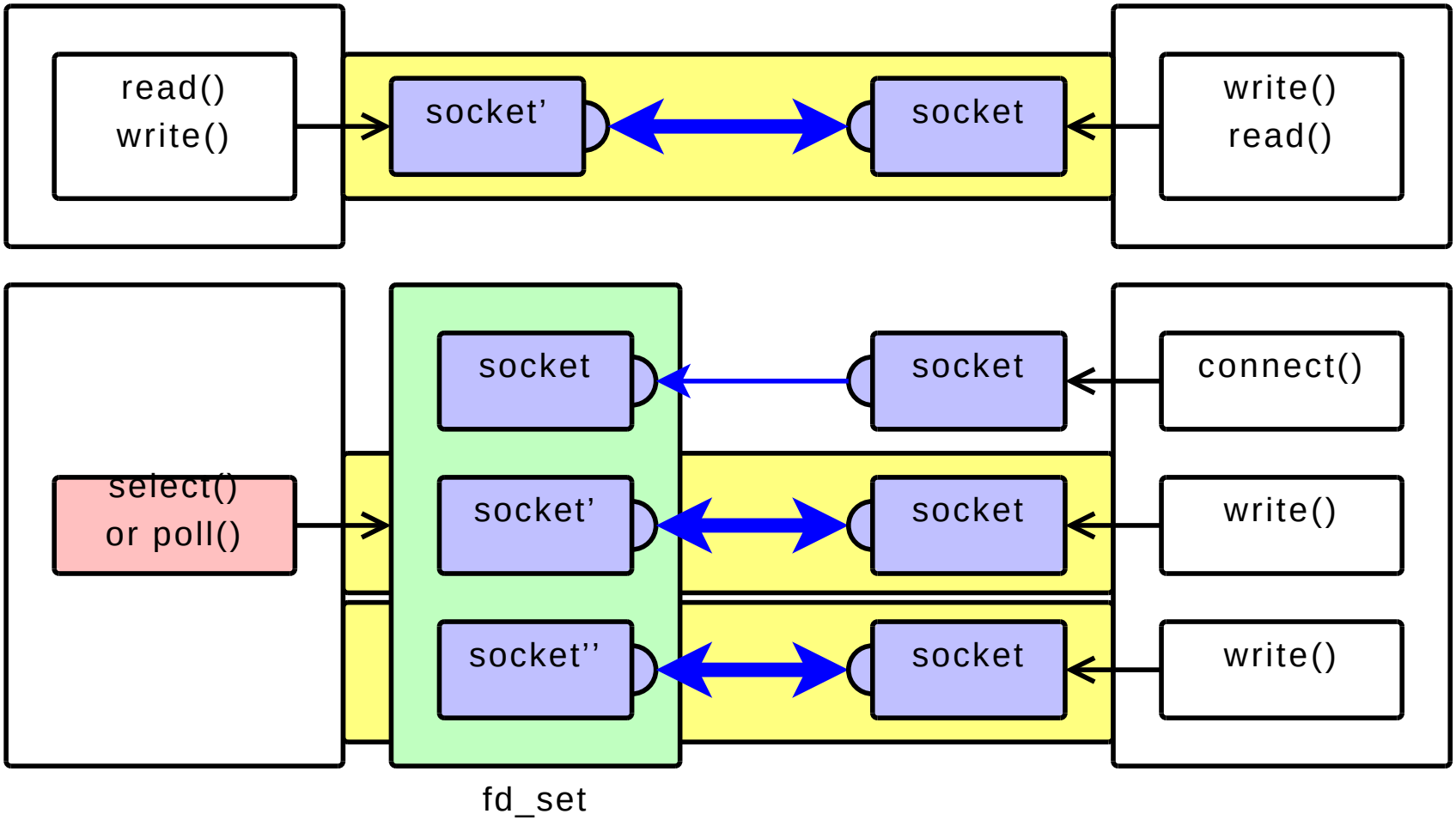


Data Transfer



Blocking syscalls 'accept' and 'read' require a thread per socket.

Data Transfer



Use 'select' or 'poll' to watch many sockets in one thread.

Strengths and Weaknesses

+ Encapsulation of data transfer
+ Extensibility with new protocols
+ Flexibility (e.g., X server)
  - Unix-domain sockets for local clients
  - Internet-domain sockets for remote clients
  - Same code, except connect()
– Inconvenient programming interface
– Each process has to close its local socket

Example (Unix-Domain)

Client:

```
int s = socket ( AF_UNIX, SOCK_STREAM, 0 );

struct sockaddr_un server_addr;
server_addr.sun_family = AF_UNIX;
strcpy( server_addr.sun_path, "/tmp/serversock" );

connect( s, (struct sockaddr *) &server_addr, sizeof(server_addr) );
```

Server:

```
s = socket( AF_UNIX, SOCK_STREAM, 0 );

struct sockaddr_un server_addr;
server_addr.sun_family = AF_UNIX;
strcpy( server_addr.sun_path, "/tmp/serversock" );

bind( s, (struct sockaddr *)&server_addr, sizeof(server_addr) );
listen( s, 5 );
```

Example (Internet-Domain)

Client:

```
int s = socket( AF_INET, SOCK_STREAM, 0 );

struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(15001);
inet_aton( "127.0.0.1", &server_addr.sin_addr );

connect( s, (struct sockaddr *)&server_addr, sizeof(server_addr) );
```

Server:

```
int s = socket( AF_INET, SOCK_STREAM, 0 );

struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(15001);
inet_aton( "127.0.0.1", &server_addr.sin_addr );

bind( s, (struct sockaddr *)&server_addr, sizeof(server_addr) );
listen( s, 5 );
```

struct socket

struct socket

- state
- flags
- proto_ops
- fasync_list
- file
- sk
- wait
- type

struct socket

struct socket

state
flags
proto_ops
fasync_list
file
sk
wait
type

struct proto_ops

inet_write
inet_read
...

struct socket

struct socket

- state
- flags
- proto_ops
- fasync_list
- file
- sk
- wait
- type

struct proto_ops

- inet_write
- inet_read
- ...

struct sock

- __sock_common
- sk_protocol
- sk_type
- ...
- head
- tail

struct socket

struct socket

- state
- flags
- proto_ops
- fasync_list
- file
- sk
- wait
- type

struct proto_ops

- inet_write
- inet_read
- ...

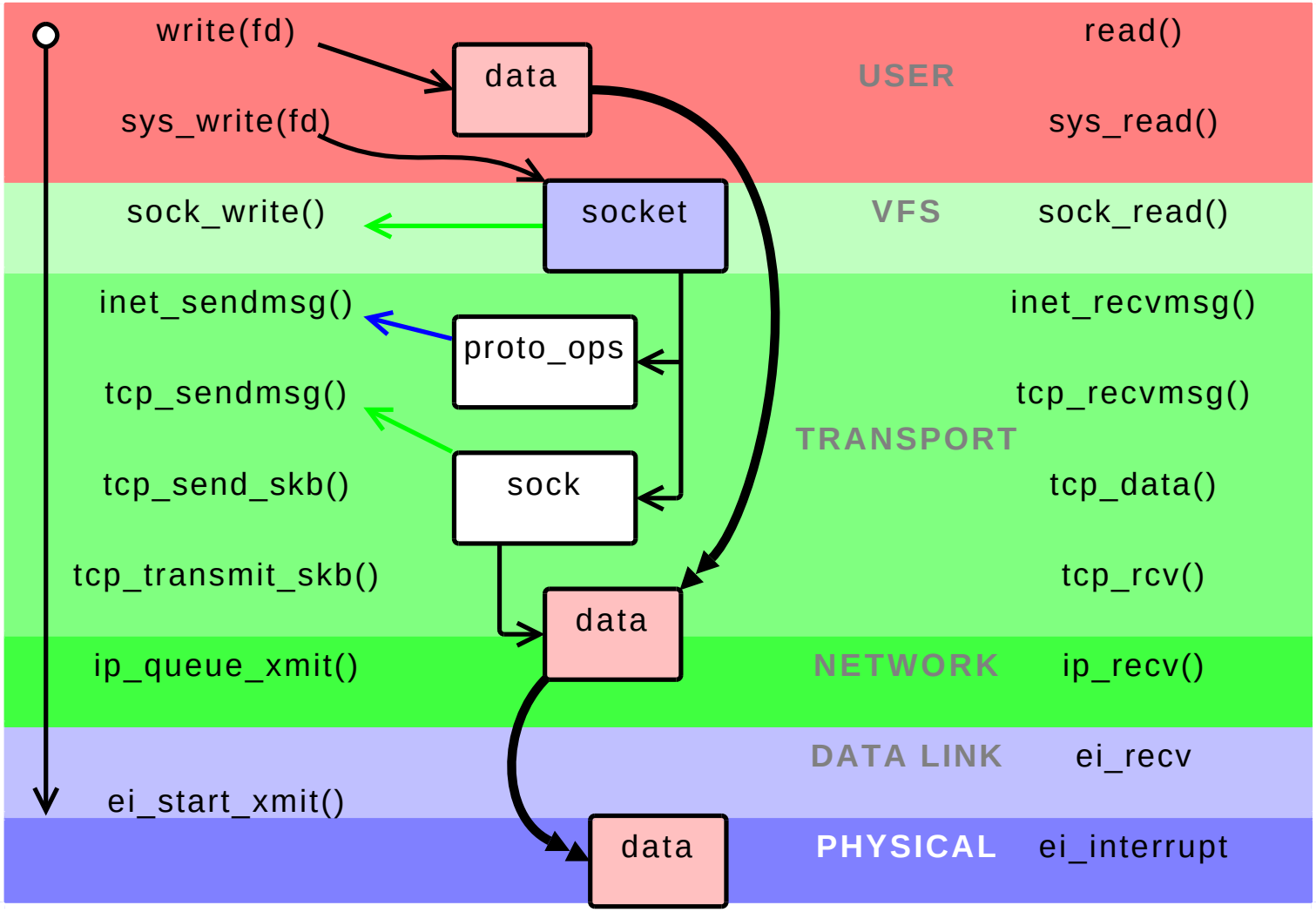
struct sock

- __sock_common
- sk_protocol
- sk_type
- ...
- head
- tail

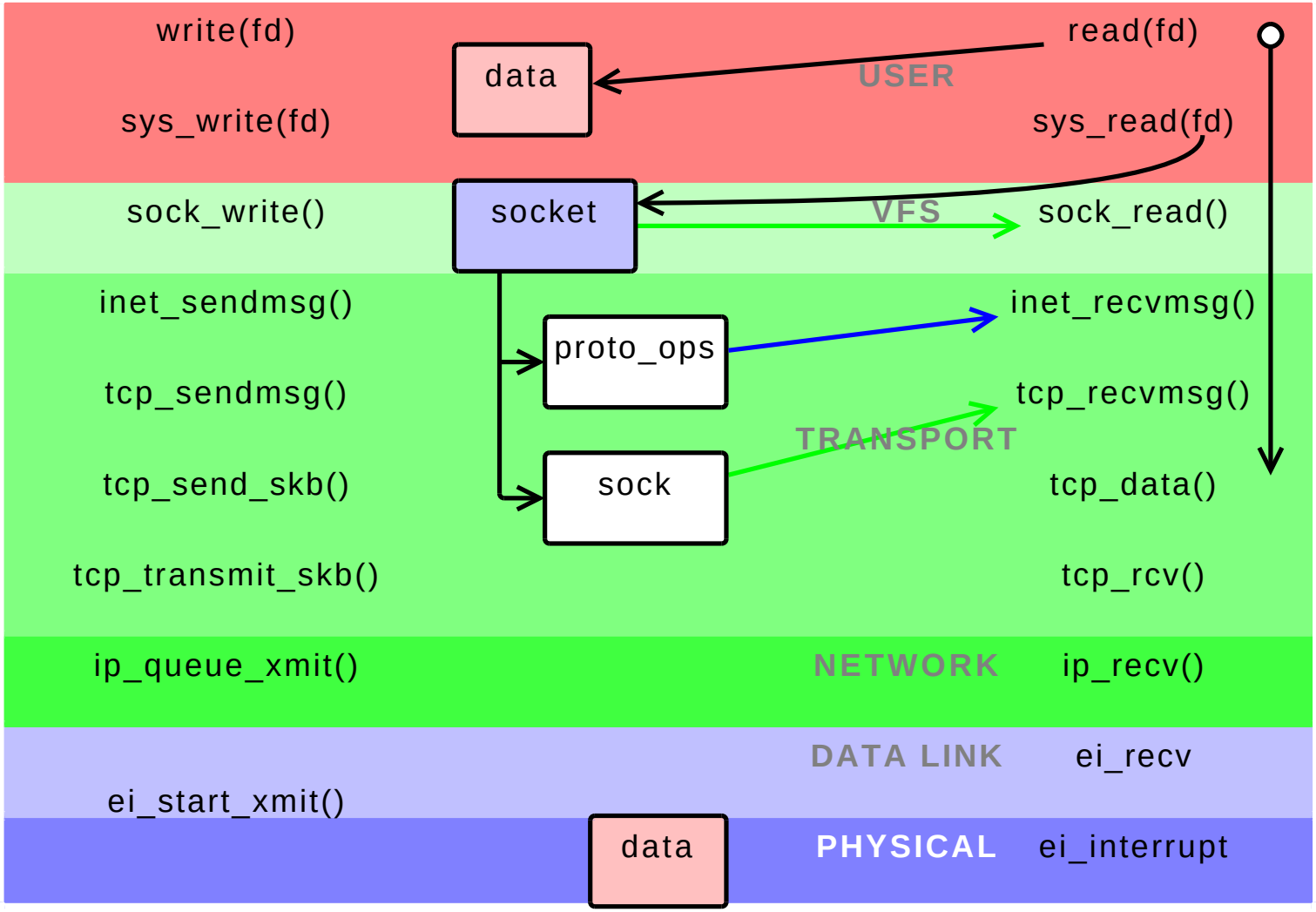
struct sk_buff

- next
- prev
- sk
- dev
- data
- ...

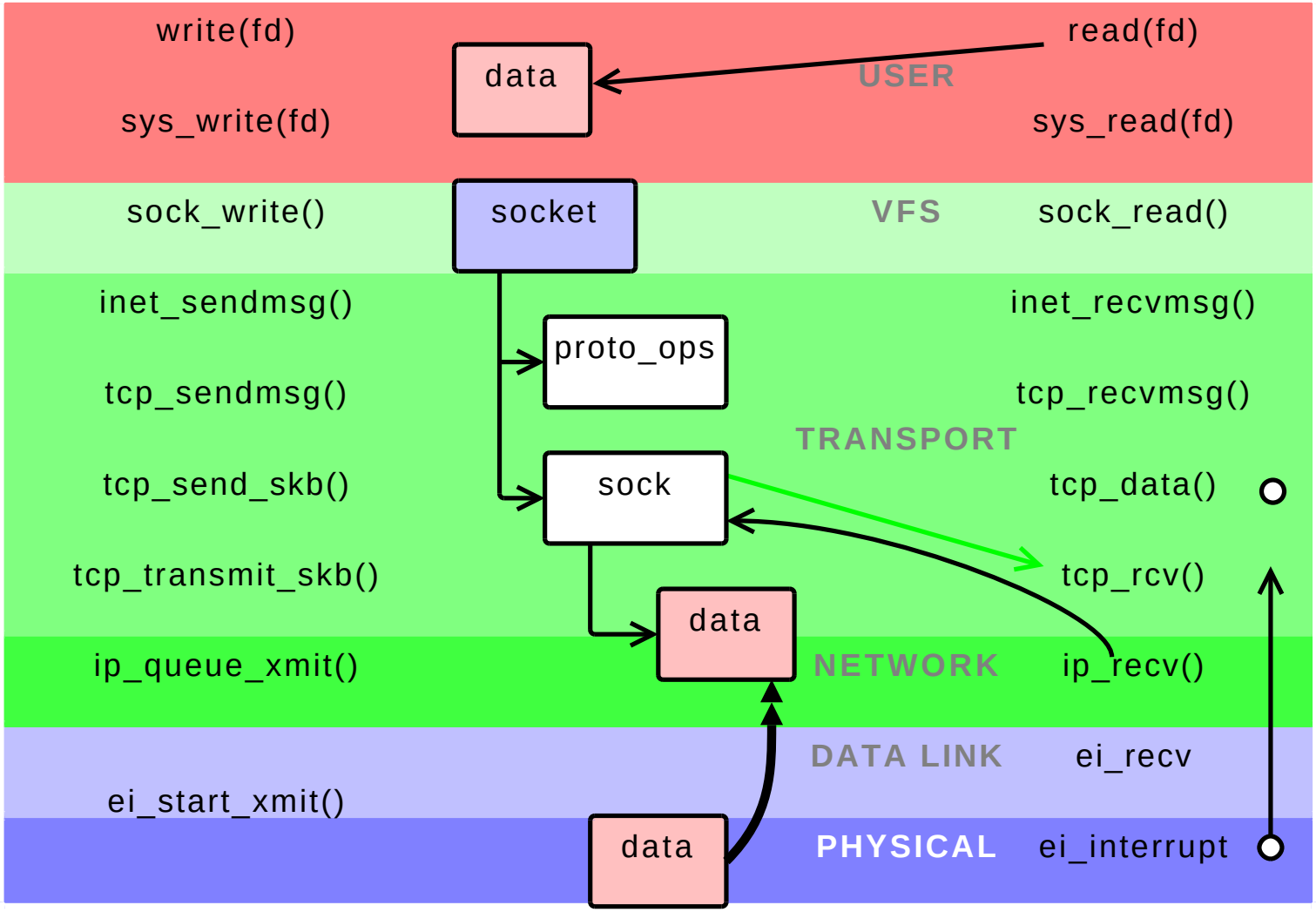
Linux Sockets in Action



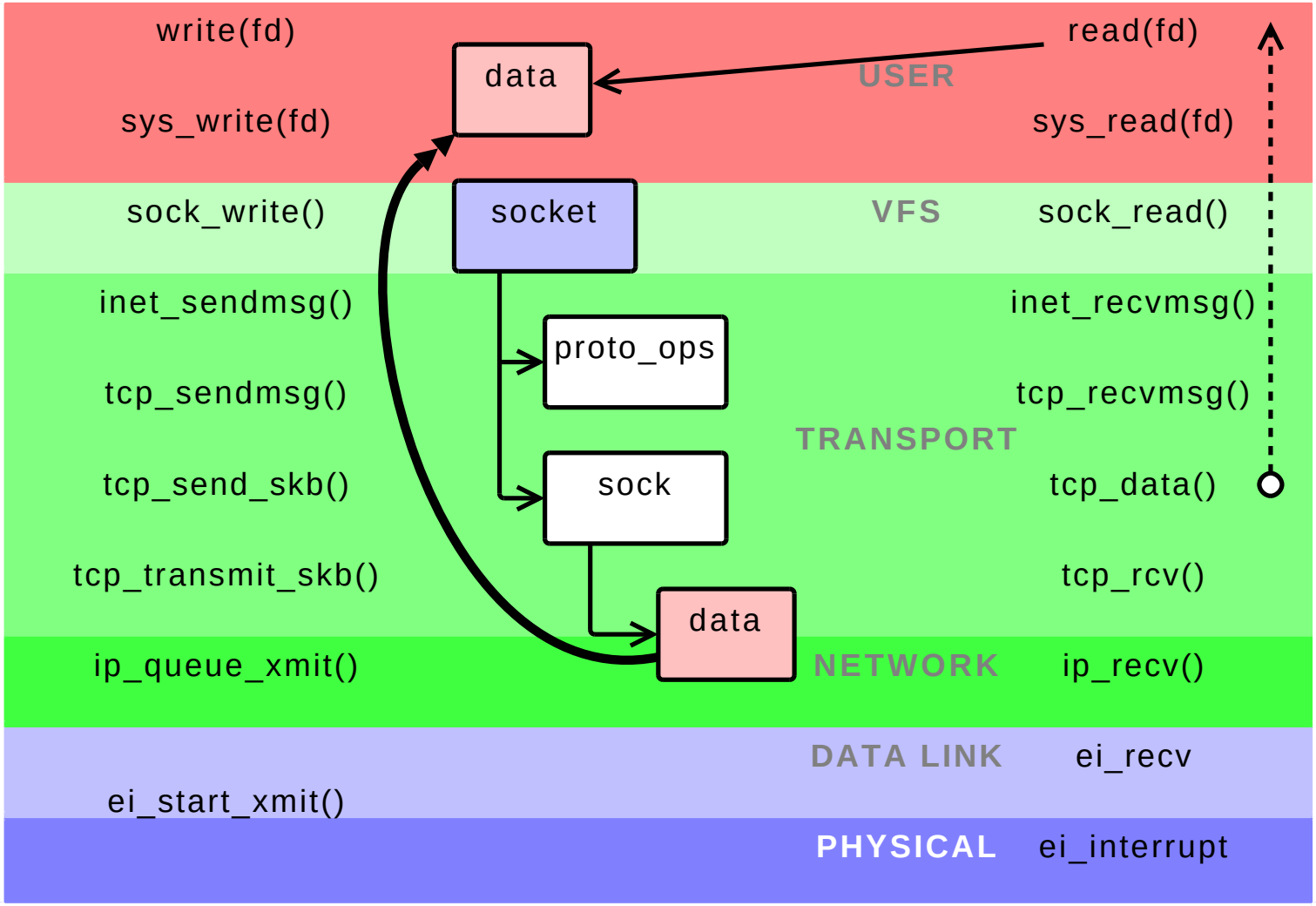
Linux Sockets in Action



Linux Sockets in Action



Linux Sockets in Action



Conclusions

- Sockets are an abstract communication interface
  - Hide different protocols (Unix-domain vs. Internet-domain)
  - Protocol dictates communication model
    - Transient vs. persistent
- Unix'ish "everything is a file" paradigm
- Linux implementations are similar for different protocols
  - Realized via function pointers and matching data structures
    - Homebrew inheritance, "object-oriented C"
  - (Questionable) solution to both performance and structured software

Literature

- Sources of the Linux kernel 2.4/2.6
- Helmut Herold: Linux/Unix Systemprogrammierung
- Jürgen Wolf: Linux-Unix-Programmierung
  http://www.pronix.de/pronix-6.html
- Beck: Linux Kernelprogrammierung
- Online Tutorial:
  http://www.cs.rpi.edu/academics/courses/netprog/

Programming Assignment

You can gain experience in socket programming by implementing RPC via sockets in the programming assignment!