Рождение, жизнь и смерть сокетов

(на примере python)

Нина Пакшина

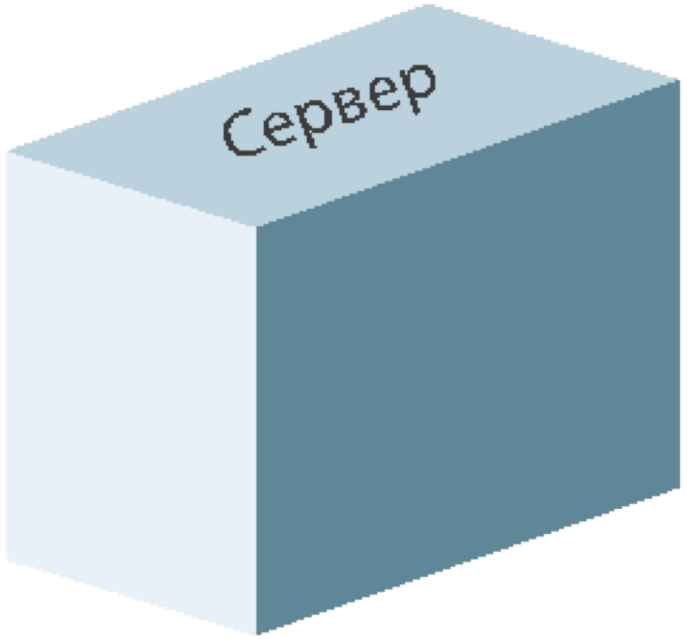
Глава 0

Начало

Сокеты Беркли

- 1983 (BSD Unix)
- 1988 стандарт POSIX.1

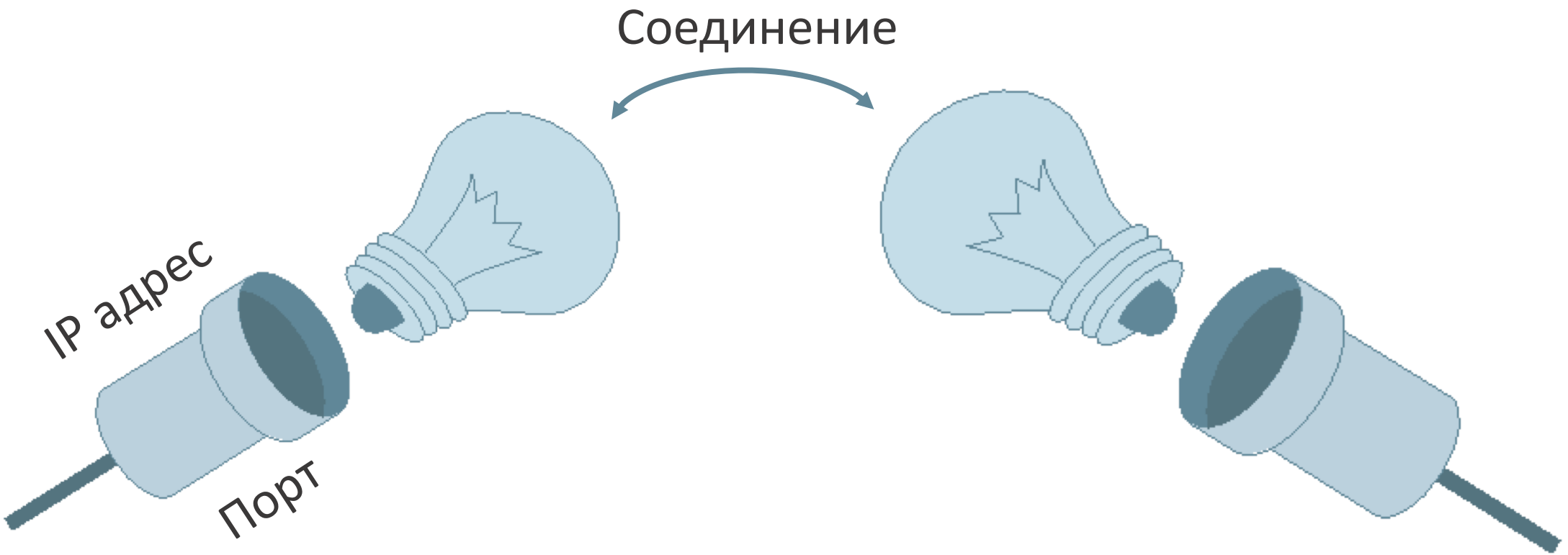
Серверные приложения



	Как связаться?
Почта	IP-адрес + 25 (SMTP)
FTP-сервер	21
Веб-сервер	80

IP-адрес + порт = TCP сокет

Что такое TCP сокет?



Что было до сокетов?

Сокеты в ARPANET

The elements of the name space are called sockets.

A **socket** forms one end of a connection and a connection is fully specified by a pair of sockets, one in each Host.

A socket is … **identified** by **a Host number** and a **16-bit socket number.**

RFC714

01.01.1983

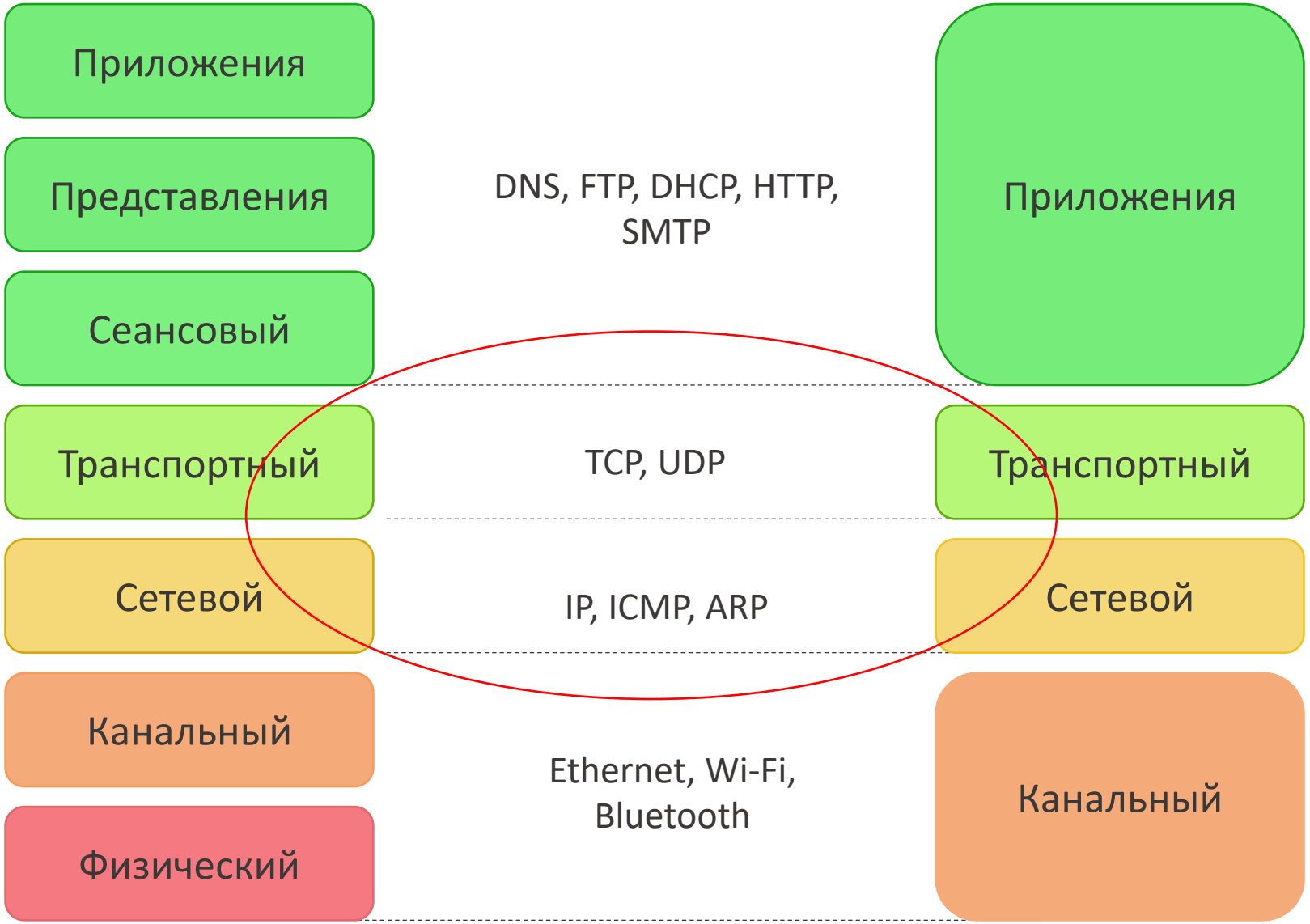
ARPANET отказался от протокола NCP в пользу TCP/IP

…we concatenate a NETWORK identifier, and a **TCP identifier** with a **port name** to create a **SOCKET** name which will be unique throughout all networks connected together.

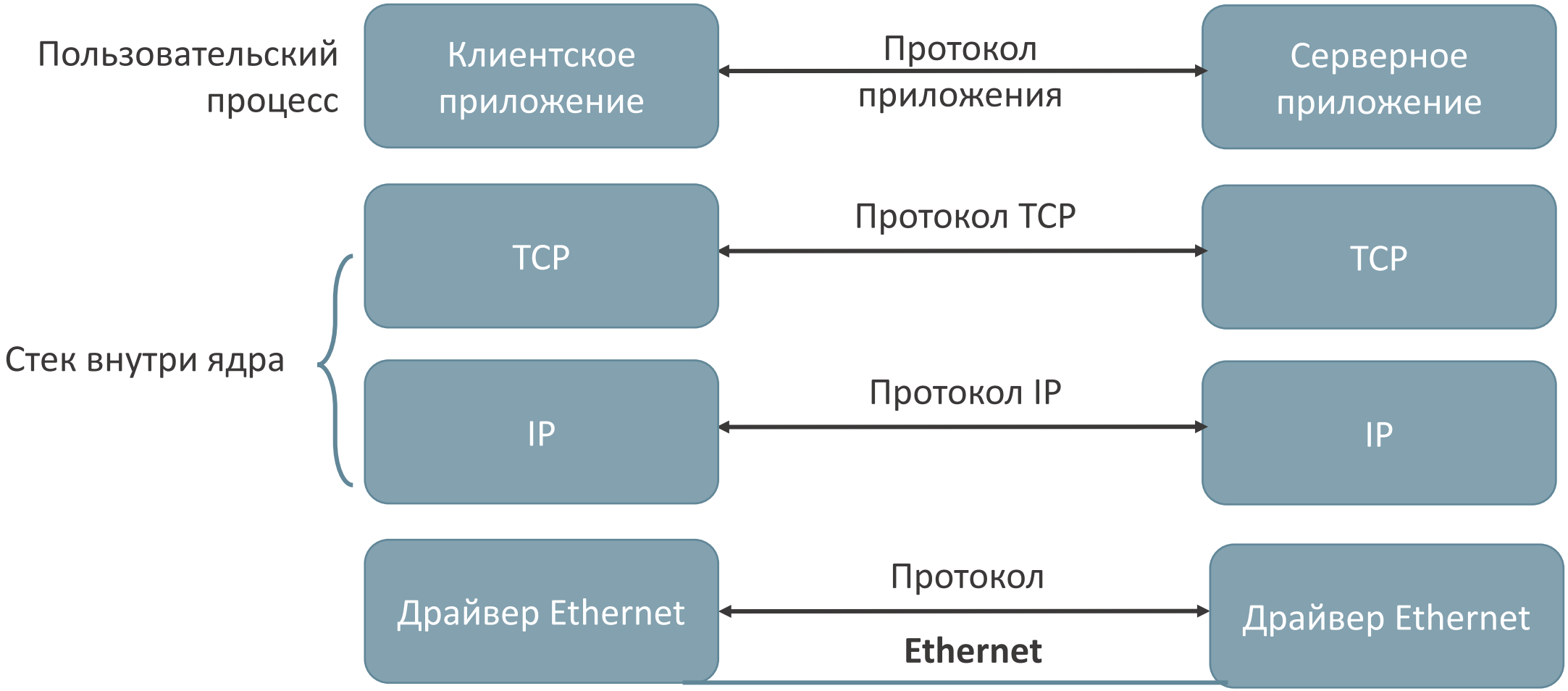
A **pair of sockets** form a **CONNECTION** which can be used to carry data in either direction [i.e. full duplex].

Зачем нужны сокеты?

Модель OSI и TCP/IP



Реальное взаимодействие



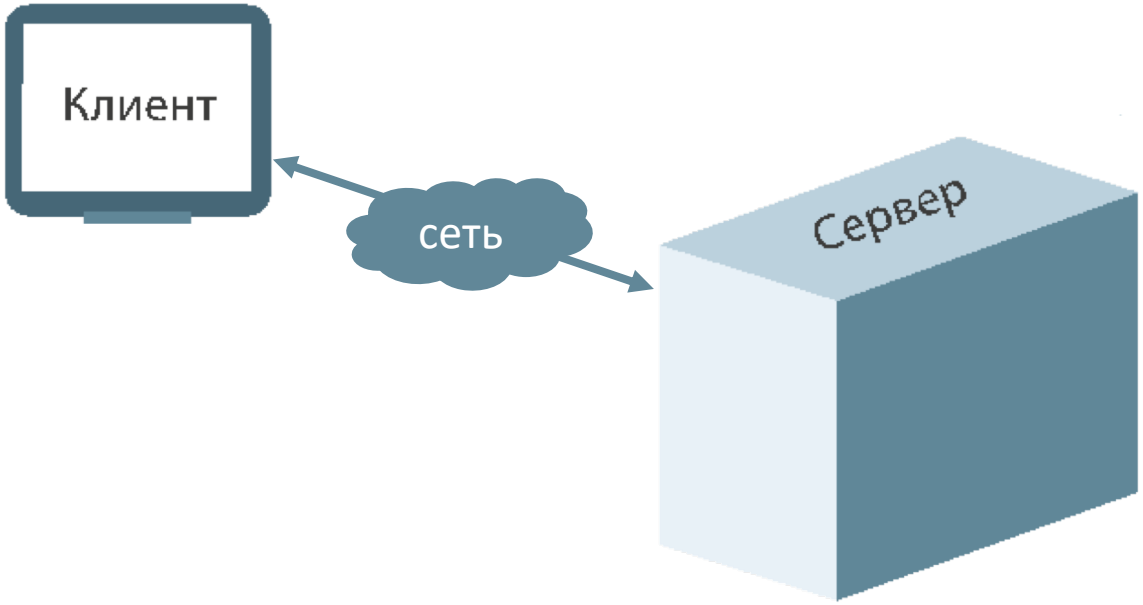
Фактический поток данных между клиентом и сервером

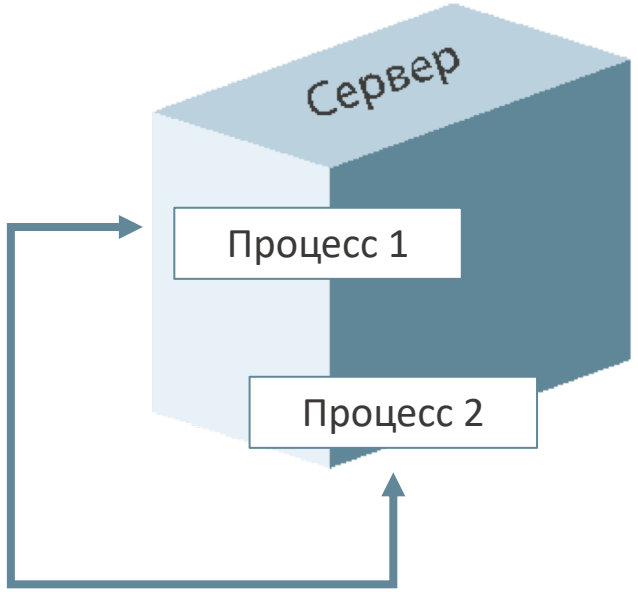
Глава 1

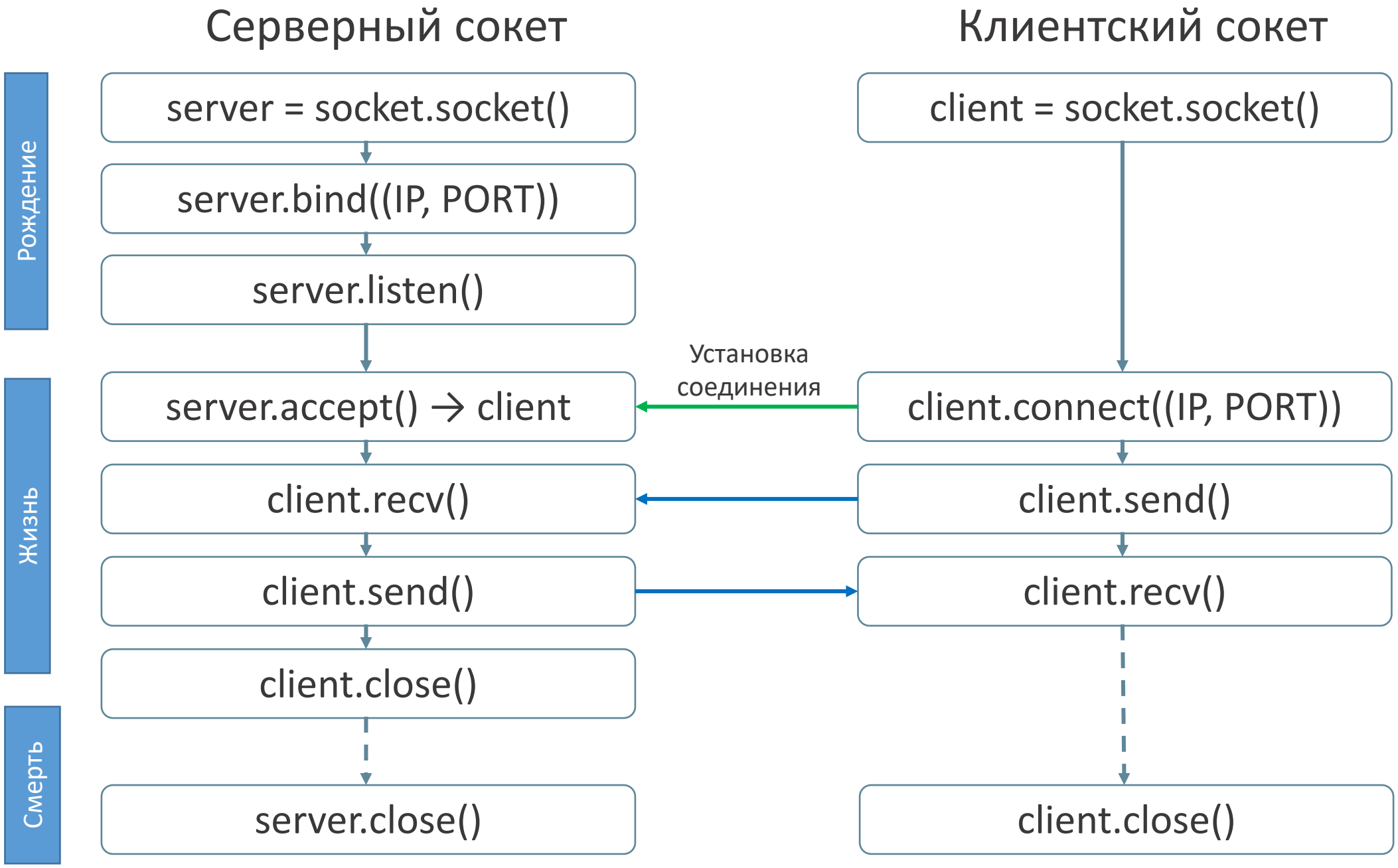
Как работают TCP сокеты

Для чего используются сокеты?

- Программный интерфейс для обеспечения обмена данными между процессами






Серверный сокет
Клиентский сокет
Рождение
Жизнь
Смерть
server = socket.socket()
server.bind((IP, PORT))
server.listen()
server.accept() → client
client.recv()
client.send()
client.close()
server.close()
Установка соединения
client = socket.socket()
client.connect((IP, PORT))
client.send()
client.recv()
client.close()

UNIX – все является файлом

Создание дескрипторов	Операции с дескрипторами
open()	read(), write()
creat()	recv(), send()
socket()	recvfrom(), sendto()
accept()	close()
pipe()	select()

Инициализация

server = socket.socket()

server.bind(address)

server.lister(MAX_CONNECTION)

```
server = socket.socket()
server.bind((‘192.168.220.5’, 8888))
server.listen(1)
```

Рабочий режим

server.accept() → client

client.recv(data_size)

client.send(data)

```
client, client_address = server.accept()
data_recv = client_socket.recv(4096)
client.send(b‘Hello from server’)
client.close()
```

Закрытие серверного сокета

server.close ()

```
server.close()
```

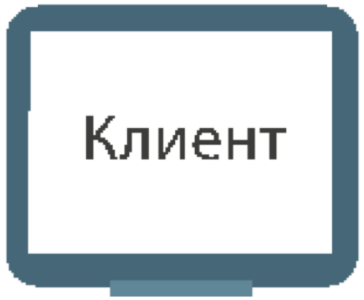
TCP сервер

```
ADDR = ('192.168.220.5‘, 8888)
MAX_CONN = 1

server = socket.socket()
server.bind(ADDR)
server.listen(MAX_CONN)
client, cl_addr = server.accept()
data_recv = client.recv(1024)
data_send = b‘Message from server’
client.send(data_send)

client.close()
server.close()
```

TCP клиент

```
ADDR = ('192.168.220.5‘, 8888)

client = socket.socket()
client.connect(ADDR)
data_send = b'Message from client'
client.send(data_send)
data_recv = client.recv(1024)
client.close()
```

Сеанс связи по протоколу TCP



Поболтаем?

Конечно, поболтаем

Отлично, подключились

Мне нужны твои данные

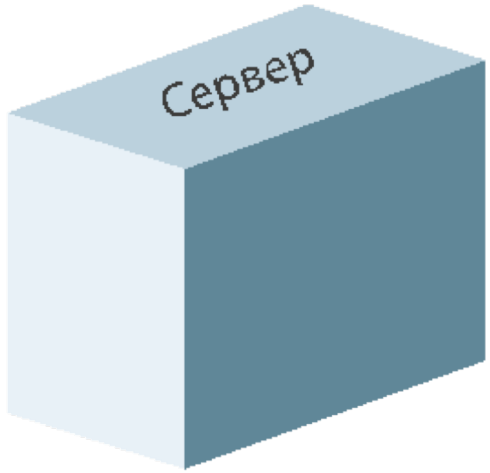
Спасибо запрос получен

Вот данные, которые ты просил

Спасибо, данные получил

Я закончил, больше слать нечего

Я тоже закончил, спасибо



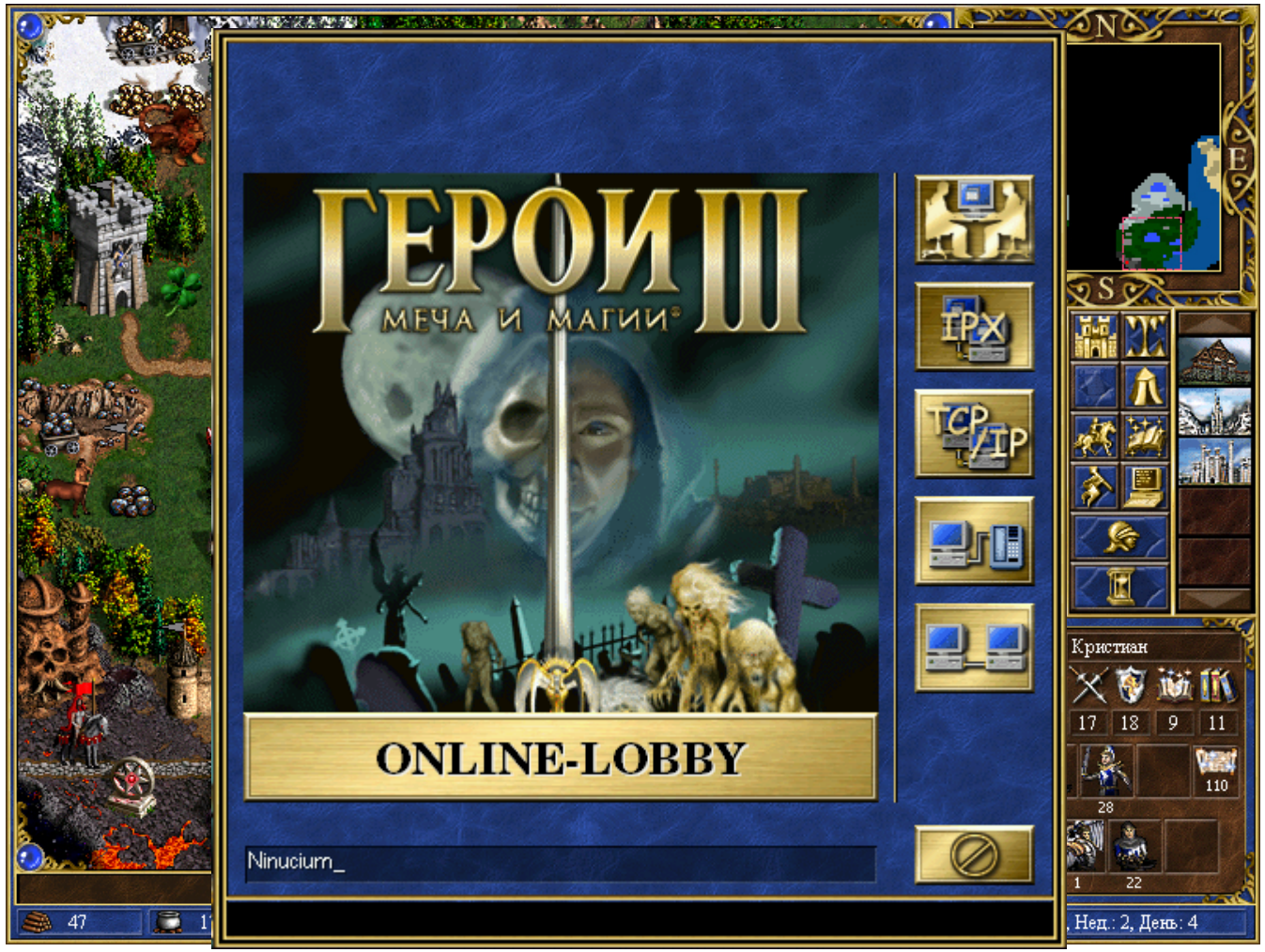
TCP сокеты

- Устанавливают соединение (тройное рукопожатие)
- Сегментация, контроль доставки и порядка сообщений
- Контроль ошибок и скорости передачи
- Избавление от дублированных сегментов

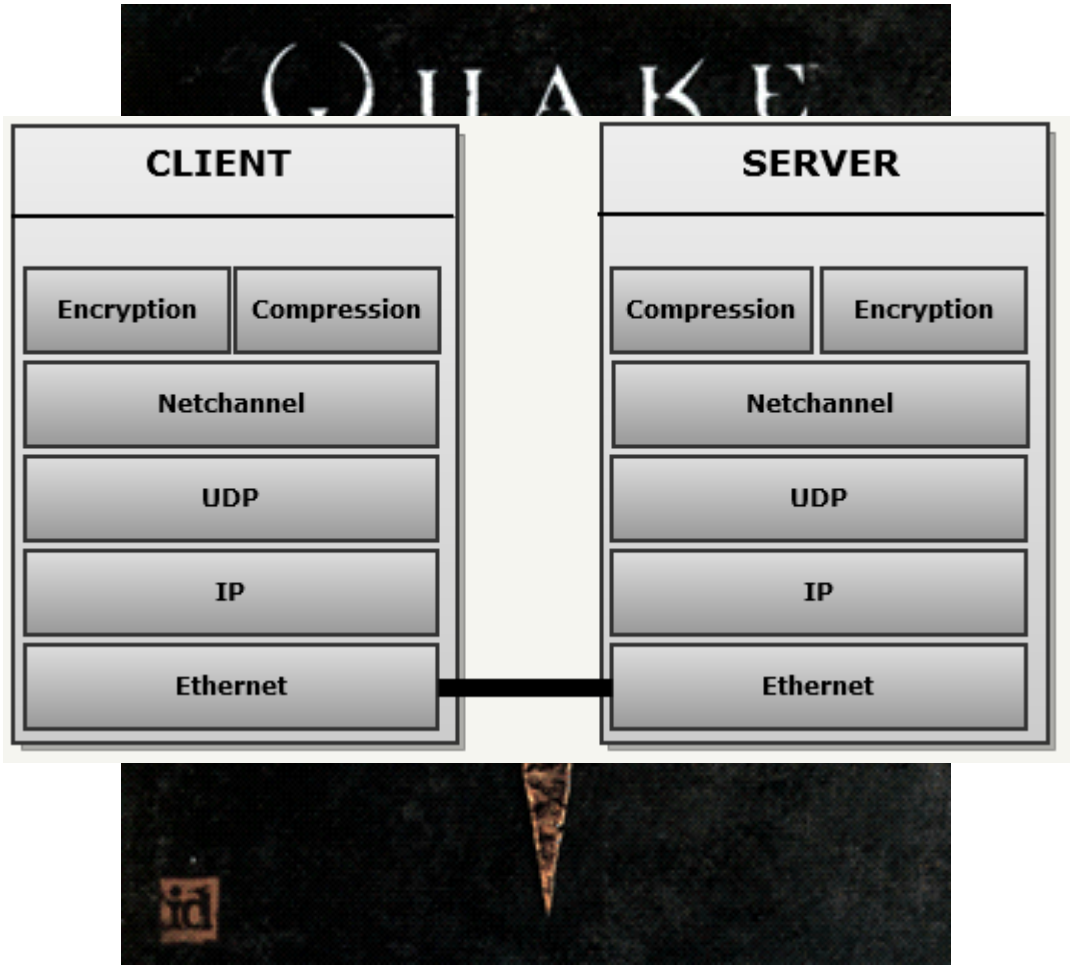
Идеальный протокол?

Что может пойти не так?

Какой протокол нужен для этой игры?



Какой протокол нужен для этой игры?



Глава 2

Не TCP единым: UDP сокет

Инициализация: создание

```
server = socket.socket()
```



```
server = socket.socket(
    family=socket.AF_INET,
    type=socket.SOCK_STREAM,
    proto=socket.IPPROTO_TCP
)
```

Инициализация: создание сокета

```
sock = socket.socket(family=..., type=..., proto=...)
```

Family (Домен) - семейство протоколов

AF_INET	IPv4, порт
AF_INET6	IPv6, порт, метка потока, мультикаст
AF_UNIX*	Путь к файлу ('/tmp/process.s')
AF_CAN*	Имя интерфейса ('can0')
AF_BLUETOOTH*	MAC адрес, порт

* Некоторые ОС могут не поддерживать

Type (Тип)

SOCK_STREAM	TCP
SOCK_DGRAM	UDP
SOCK_RAW	Сырой (нестандартные пакеты)
SOCK_SEQPACKET	Последовательные пакеты

Protocol - используемый транспортный протокол

IPPROTO_TCP	TCP
IPPROTO_UDP	UDP
IPPROTO_SCTP	SCTP

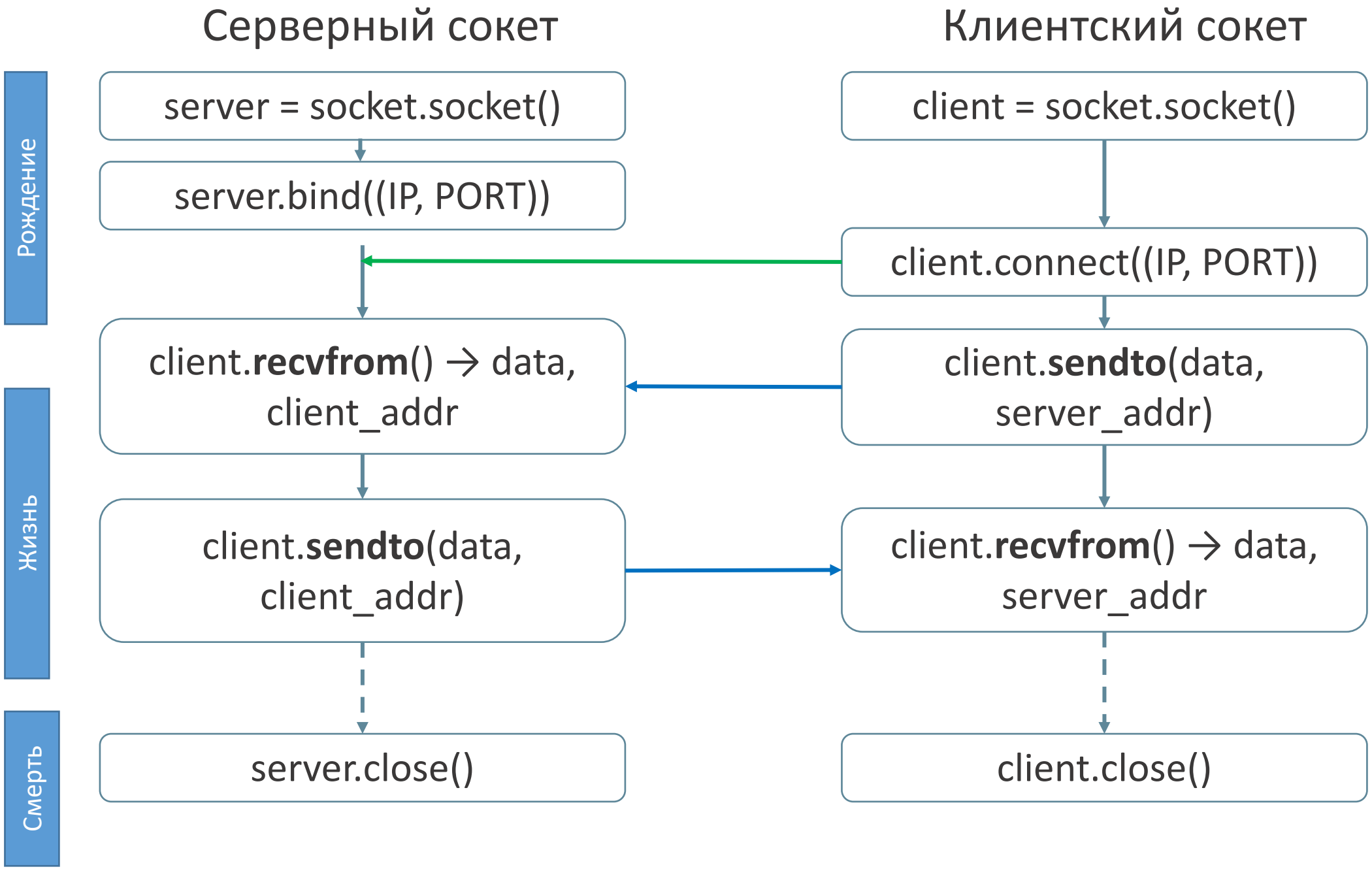
TCP и UDP сокеты

TCP

```
server = socket.socket(
   family=socket.AF_INET,
   type=socket.SOCK_STREAM,
   proto=socket.IPPROTO_TCP
)
```

UDP

```
server = socket.socket(
   family=socket.AF_INET,
   type=socket.SOCK_DGRAM,
   proto=socket.IPPROTO_UDP
)
```


Серверный сокет
Клиентский сокет
Рождение
Жизнь
Смерть
server = socket.socket()
server.bind((IP, PORT))
client = socket.socket()
client.connect((IP, PORT))
client.recvfrom() → data, client_addr
client.sendto(data, server_addr)
client.sendto(data, client_addr)
client.recvfrom() → data, server_addr
server.close()
client.close()

UDP сервер

```
ADDR = ('192.168.220.5‘, 8888)

server = socket.socket(
    family=socket.AF_INET,
    type=socket.SOCK_DGRAM,
    proto=socket.IPPROTO_UDP,
)
server.bind(ADDR)
data_recv, cl_addr = server.recvfrom(1024)
server.sendto(b‘Message from server’, cl_addr)
server.close()
```

UDP клиент

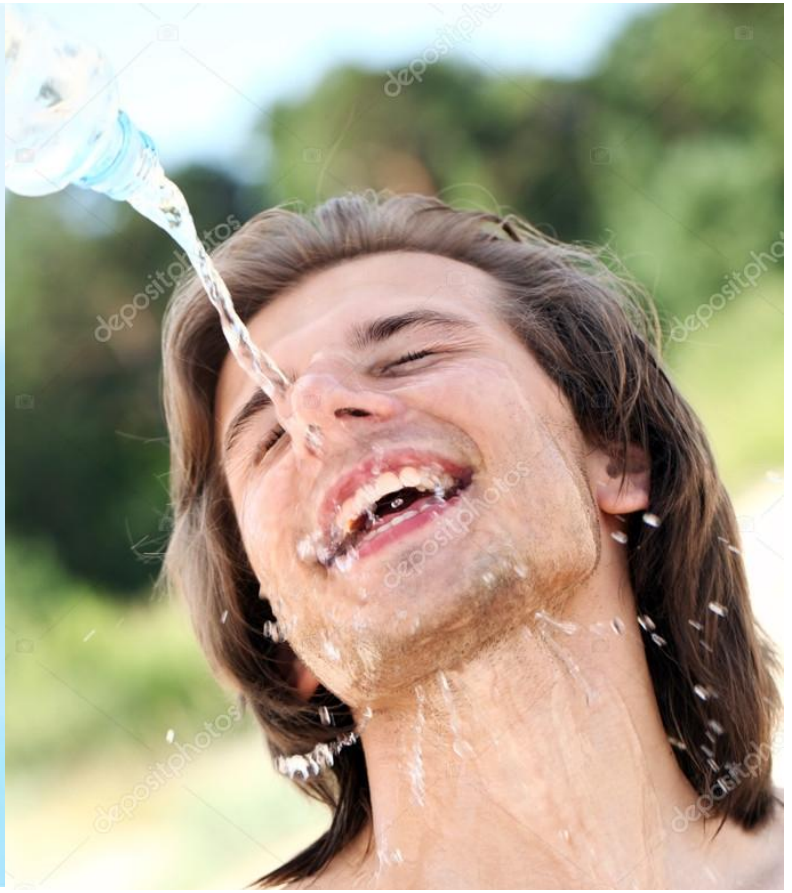
```
ADDR = (‘192.168.220.5’, 8888)

client = socket.socket(
    family=socket.AF_INET,
    type=socket.SOCK_DGRAM,
    proto=socket.IPPROTO_UDP,
)
client.connect(server_addr)
client.sendto(b'Message from client', server_addr)
data_recv, server_addr = client.recvfrom(1024)
client.close()
```

Какой сокет лучше?

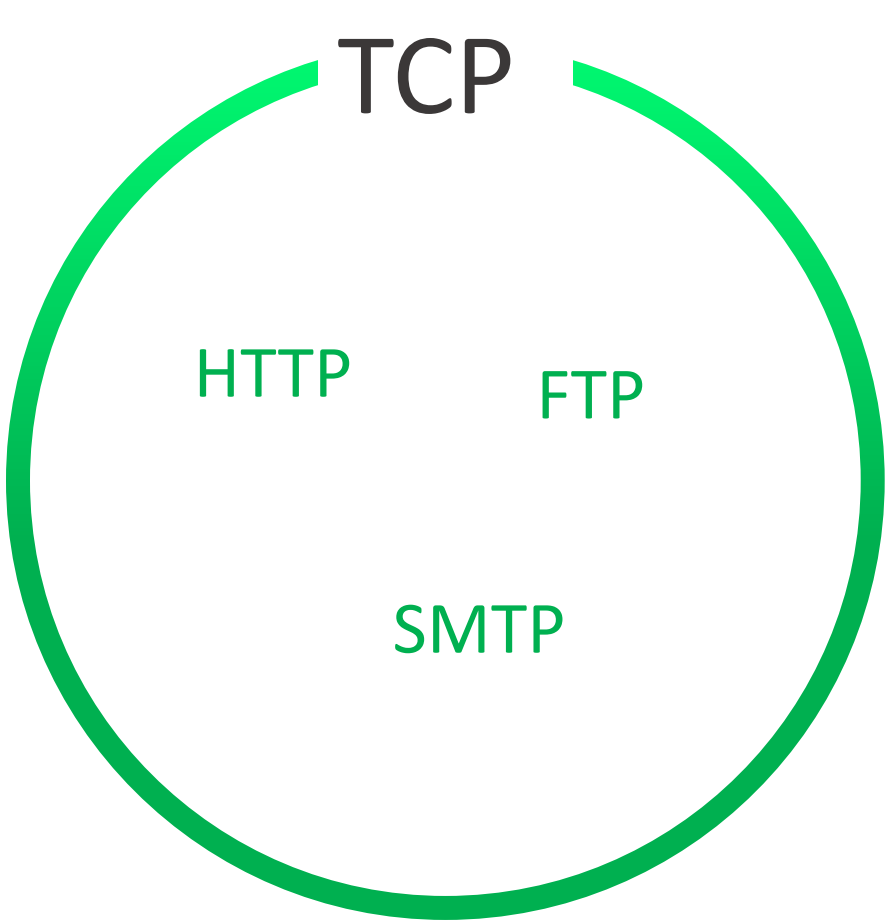
	UDP	TCP
Установка соединения	нет	да
Гарантия доставки сообщений	нет	да
Широковещательная рассылка	да	нет
Доставка сообщений в правильном порядке	нет	да

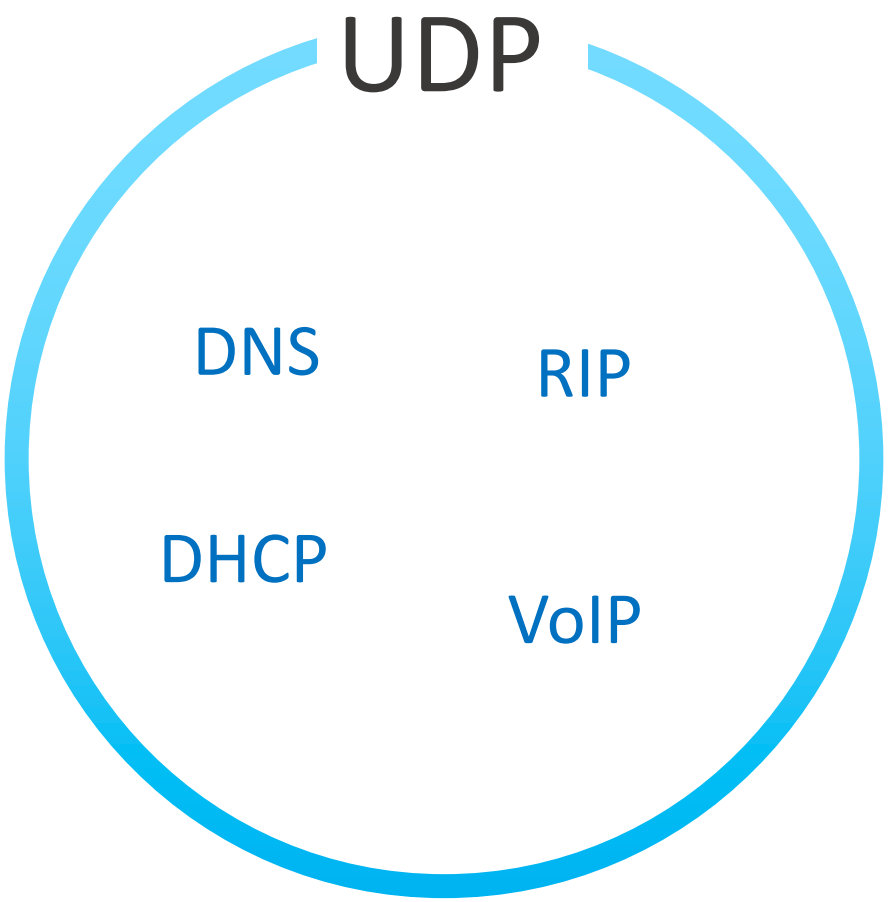
TCP

UDP



Протоколы

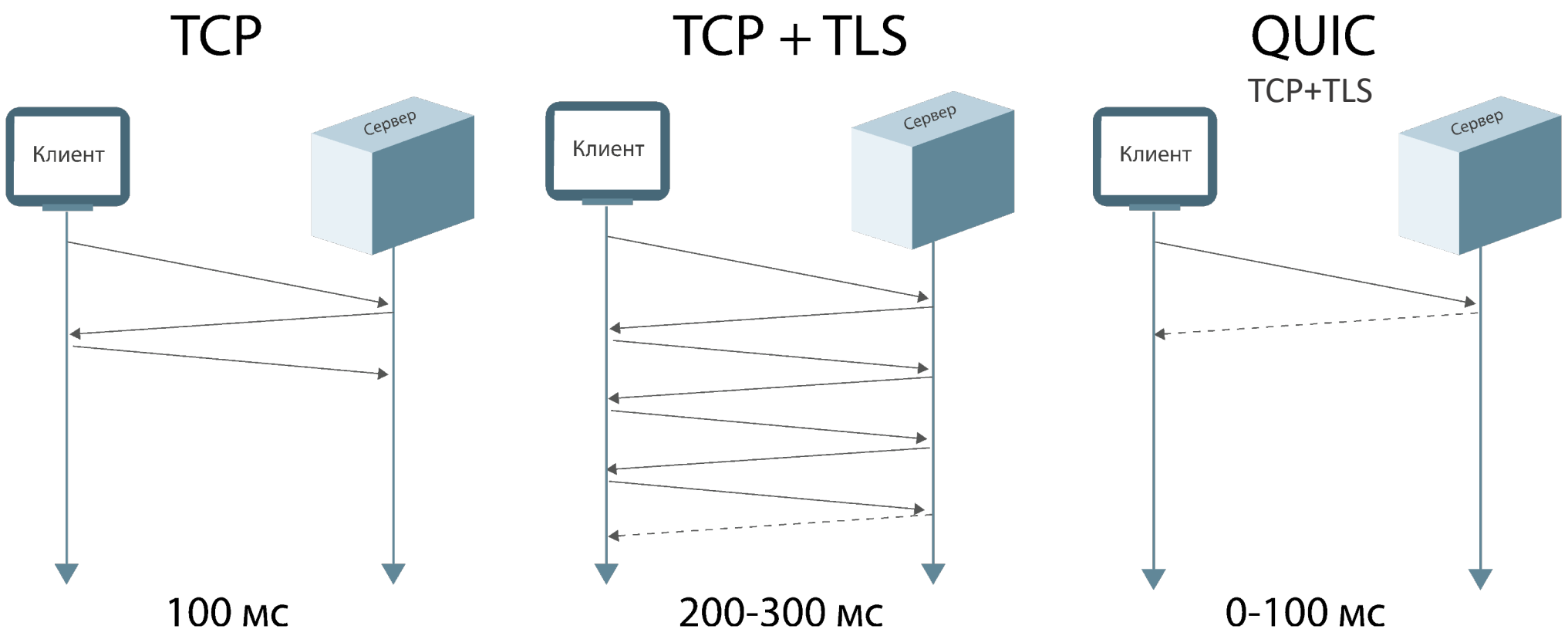




QUIC

Quick UDP Internet Connections

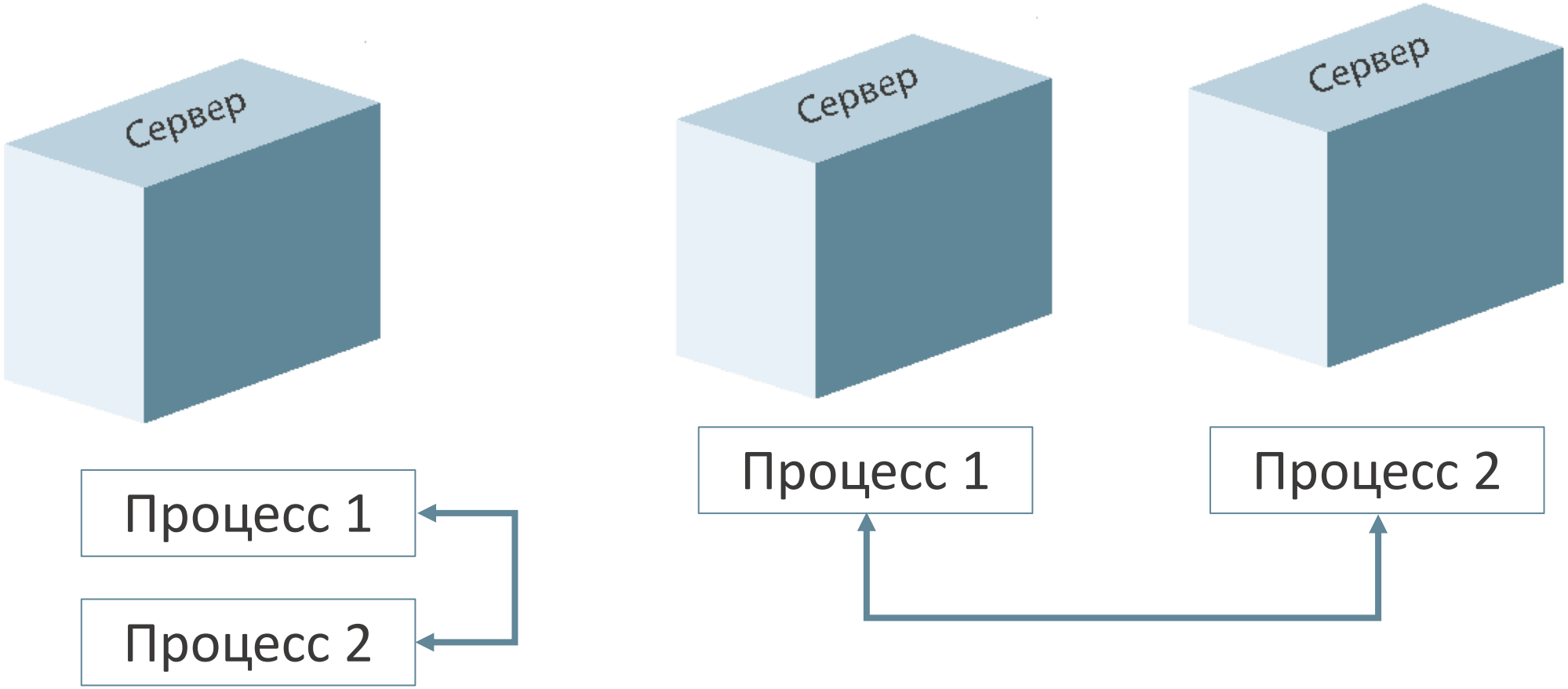
Безопасность на уровне TLS
Контроль целостности потока
Очень быстрое подключение



Глава 2

UNIX сокеты

Взаимодействие между сокетами



UNIX сервер

```
server = socket.socket(socket.AF_UNIX,
socket.SOCK_DGRAM)
server.bind("/tmp/python_unix_sockets_example")

while True:
    datagram = server.recv(1024)
    if not datagram:
        break
    else:
       if "DONE" == datagram.decode('utf-8'):
           break
server.close()
```

UNIX на WINDOWS

AF_UNIX comes to Windows

Sarah

December 19th, 2017

Глава 3

Усложняем ~~жизнь~~ код

Переиспользование адреса

server = socket.socket()

server.bind((HOST, PORT))

server.close()

Почему мы ждем?

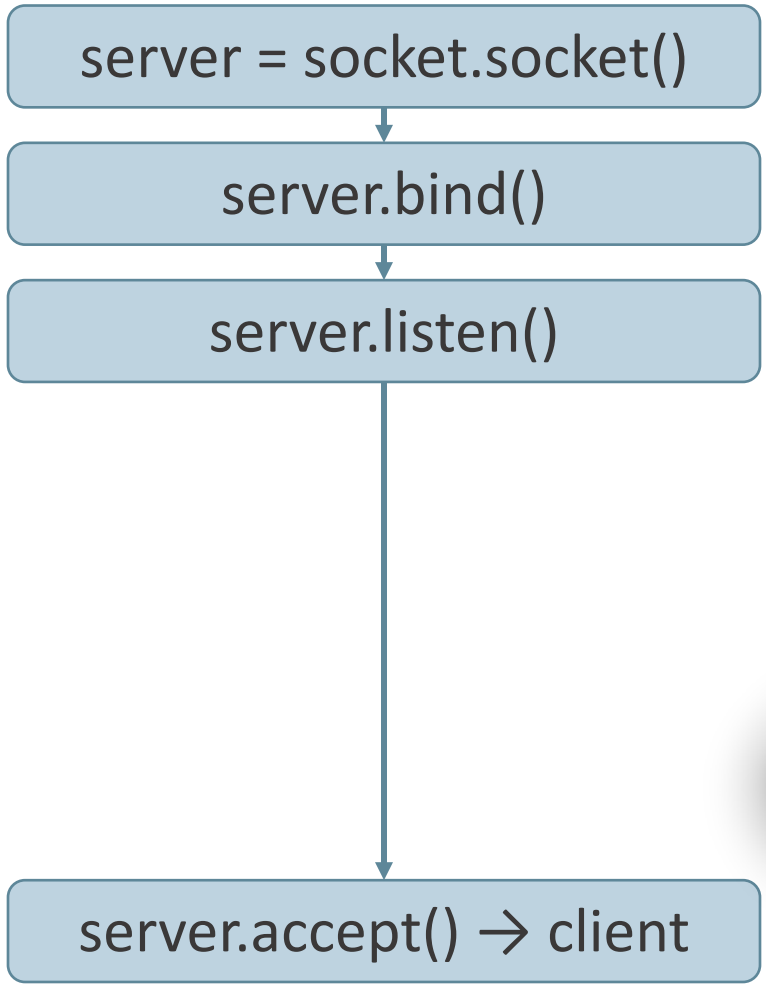
2 x MSL!

server = socket.socket()

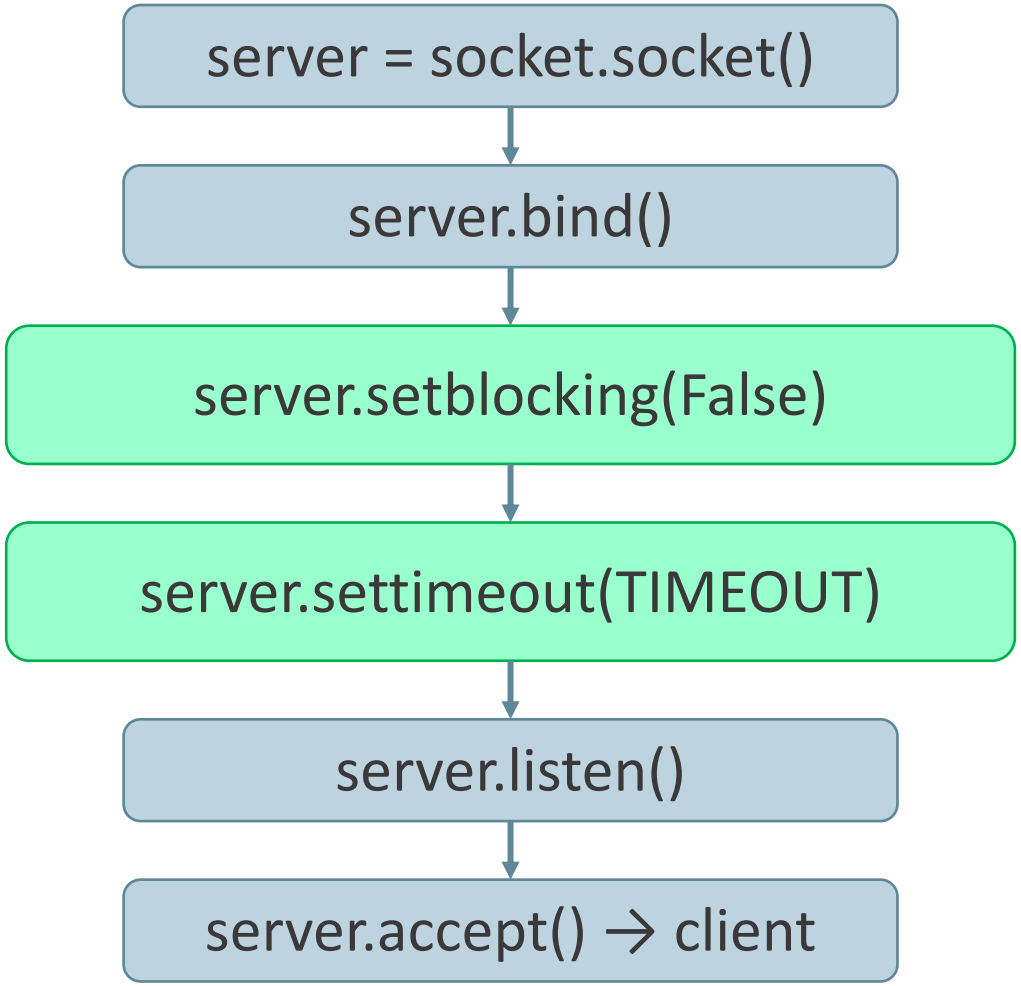
server.**setsockopt**(
socket.SOL_SOCKET,
socket.SO_REUSEADDR,
1
)

server.bind((HOST, PORT))

Блокирующая функция



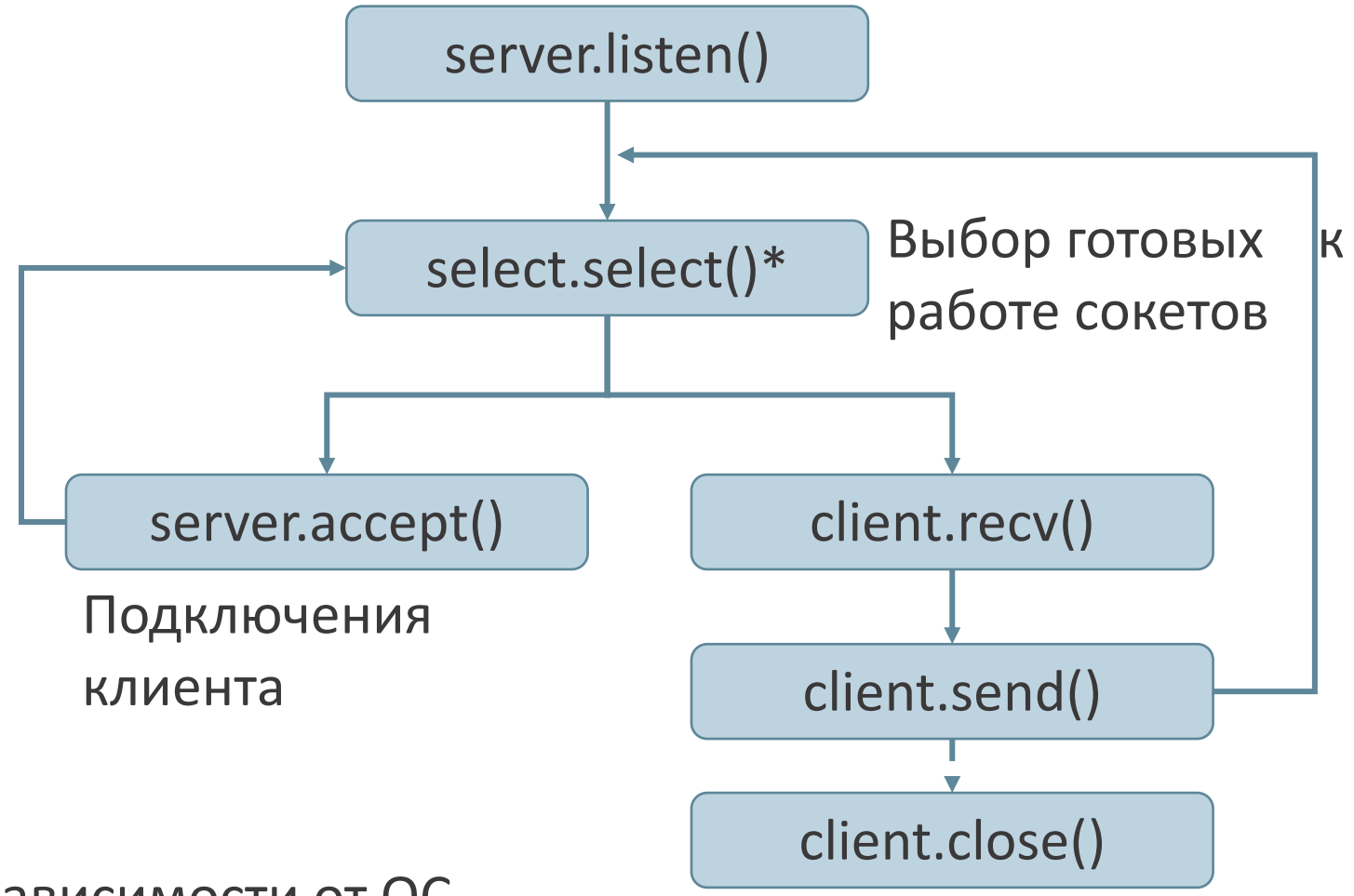
Неблокирующая с таймеров



Потоки asyncio

```
import asyncio
async def tcp_echo_client(message):
    reader, writer = await asyncio.open_connection(
        '127.0.0.1', 8888)
    writer.write(message.encode())
    data = await reader.read(100)
    writer.close()
    await writer.wait_closed()

asyncio.run(tcp_echo_client('Hello World!'))
```

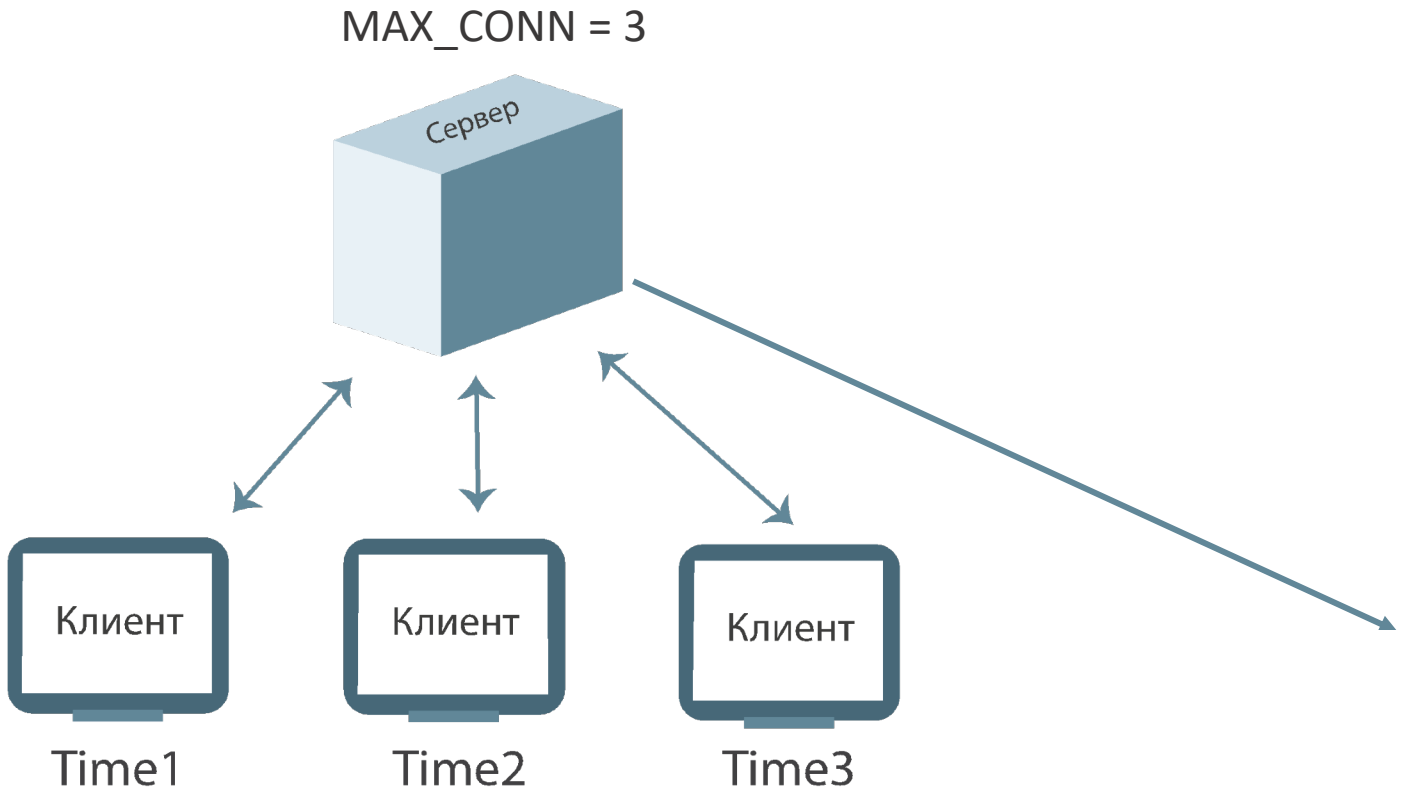
Обработка нескольких клиентов



* Выбирается в зависимости от ОС

Обработка новых подключений

Очередь на подключение



Глава 4

Безопасность

Сокеты – как открытая книга

- Передача публичных данных
- Защищенный контур

SSL

```
import socket
import ssl
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain('/path/to/certchain.pem', '/path/to/private.key')

with socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0) as sock:
    sock.bind(('127.0.0.1', 8443))
    sock.listen(5)
    with context.wrap_socket(sock, server_side=True) as ssock:
        conn, addr = ssock.accept()
        ...
```

Глава 4

Сокеты на других языках

Server C/C++

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8080
int main(int argc, char const *argv[])
{
	int server_fd, new_socket, valread;
	struct sockaddr_in address;
	int opt = 1;
	int addrlen = sizeof(address);
	char buffer[1024] = {0};
	char *hello = "Hello from server";

	// Creating socket file descriptor
	if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
	{
		perror("socket failed");
		exit(EXIT_FAILURE);
	}
```

```
// Forcefully attaching socket to the port 8080
        if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                                                &opt, sizeof(opt)))
        {
                perror("setsockopt");
                exit(EXIT_FAILURE);
        }
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = INADDR_ANY;
        address.sin_port = htons( PORT );

        // Forcefully attaching socket to the port 8080
        if (bind(server_fd, (struct sockaddr *)&address,
                                                sizeof(address))<0)
        {
                perror("bind failed");
                exit(EXIT_FAILURE);
        }
```

```
if (listen(server_fd, 3) < 0)
        {
                perror("listen");
                exit(EXIT_FAILURE);
        }
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                                (socklen_t*)&addrlen))<0)
        {
                perror("accept");
                exit(EXIT_FAILURE);
        }
        valread = read( new_socket , buffer, 1024);
        printf("%s\n",buffer );
        send(new_socket , hello , strlen(hello) , 0 );
        printf("Hello message sent\n");
        return 0;
}
```

Java серверный сокет

```
import java.net.*;
import java.io.*;
public class Server {
   public static void main(String[] ar)    {
     int port = 8888;
       try {
         ServerSocket ss = new ServerSocket(port);
         System.out.println("Waiting for a client...");

         Socket socket = ss.accept();
```

Java серверный сокет

```
InputStream sin = socket.getInputStream();
OutputStream sout = socket.getOutputStream();

DataInputStream in = new DataInputStream(sin);
DataOutputStream out = new DataOutputStream(sout);
```

Java серверный сокет

```
            String line = null;
            while(true) {
              line = in.readUTF();
              out.writeUTF(line);
              out.flush();
            }
        } catch(Exception x) { x.printStackTrace(); }
    }
}
```

socket() でソケットをオ プンするときには，ネットワ ク のプロトコルとトランスポ ト のプロトコルを する．ソケットをクロ ズするには UNIX では close()，Windows では closesocket() を う．TCP ではコネクションの など， を させるために shutdown() が われることもある．

bind() で で する IP アドレス，ポ ト を し，connect() で が する IP アドレス，ポ ト を する．connect() に， のホストで する IP アドレス，ポ ト が まっていない には，OS が に する．

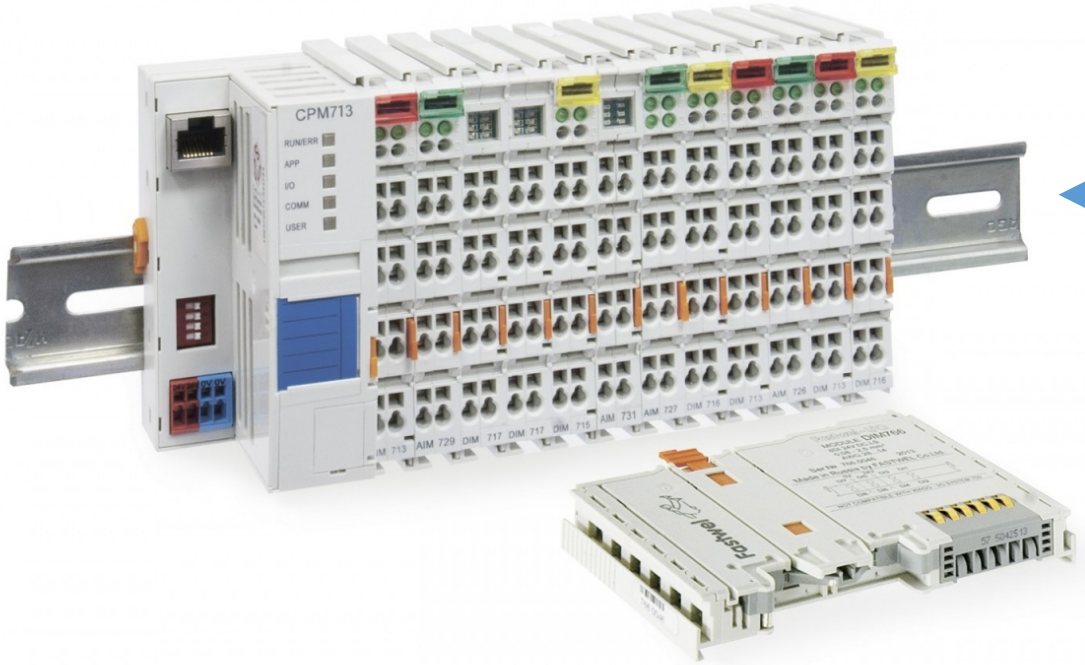
TCP の ，コネクション け け のソケットと，メッセ ジ のソケットは される．socket() でオ プンしたソケットは，コネクション け け のソケットであり，アプリケ ションメッセ ジの には されない．socket() でオ プンしたソケットに して listen() するとコネクションの が される．TCP コネクションが されると しいソケットが られ，accept() で られたソケットのディスクリプタをることができる．このディスクリプタを ってアプリケ ションメッセ ジを することになる．

TCP ではコネクション accept() しなければサ バ ではメッセ ジを できない．accept() しなくても できる TCP コネクション の を listen() で できる．

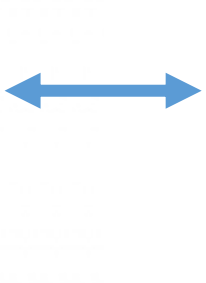
メッセ ジの は send()，sendto()，sendmsg()， は recv()，recvfrom()，recvmsg() を する．

send()，recv() は，TCP のとき，または，UDP で connect() したときに する．つまり，コネクション である IP， ポ ト， IP， ポ トが されているときに する．sendto()，recvfrom() は，UDP で IP， ポ トが されていないときに する．

Подключение к ПЛК



TCP сервер



TCP клиент

Глава 5

Итоги

И это только опции сокетов

SO_REUSEADDR
SO_KEEPALIVE
SO_DONTROUTE
SO_SNDBUF
SO_RCVBUF
SO_LINGER
SO_USELOOPBACK
SO_OOBINLINE

SO_SNDLOWAT
SO_RCVLOWAT
SO_SNDTIMEO
SO_RCVTIMEO
SO_TYPE
SO_ERROR
SO_BROADCAST
SO_REUSEPORT

Подводим итоги сокетов

- Позволяет быстро реализовать обмен между удаленными машинами или процессами;
- Сокеты универсальны и гибки в применении;
- Тип сокета зависит от области применения;
- Для безопасности необходимо дополнительные действия.

Спасибо за внимание!